ML/AI for Memory System Design & Memory System Design for AI/ML

Onur Mutlu
omutlu@gmail.com
https://people.inf.ethz.ch/omutlu
23 August 2024
Intel ArchFest

SAFARI

ETH zürich

The Problem

Computing

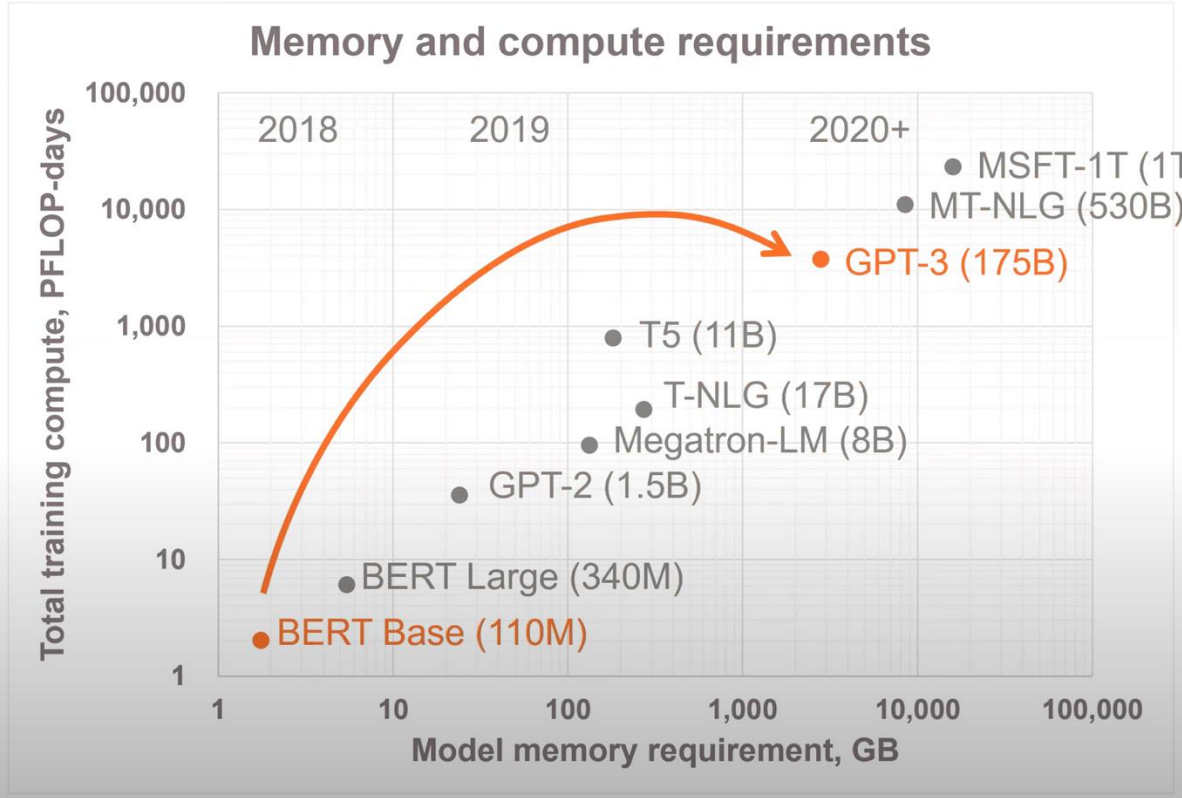
is Bottlenecked by Data

Data is Key for AI, ML, Genomics, …

- Important workloads are all data intensive
- They require rapid and efficient processing of large amounts of data
- Data is increasing
  - We can generate more than we can process
  - We need to perform more sophisticated analyses on more data

Huge Demand for Performance & Efficiency

Exponential Growth of Neural Networks



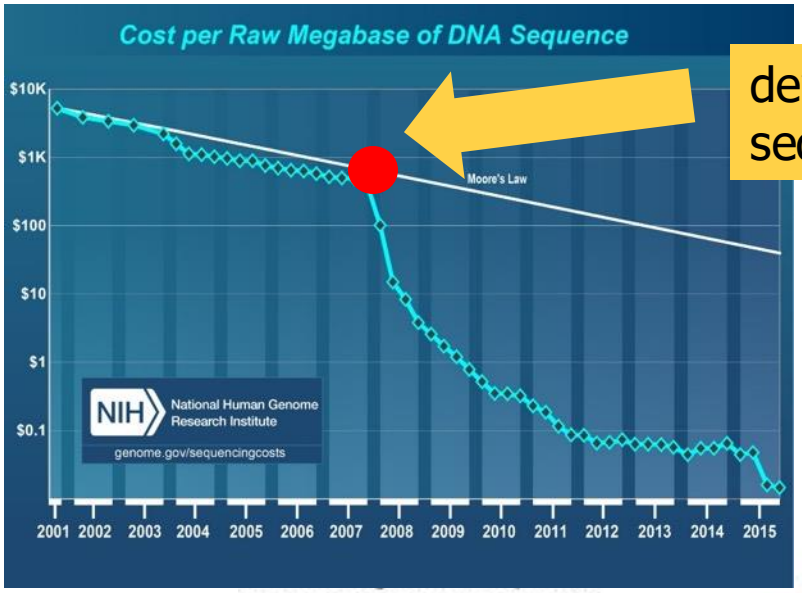
1800x more compute

In just **2 years**

Tomorrow, **multi-trillion** parameter models

Source: https://youtu.be/Bh13Idwcb0Q?t=283

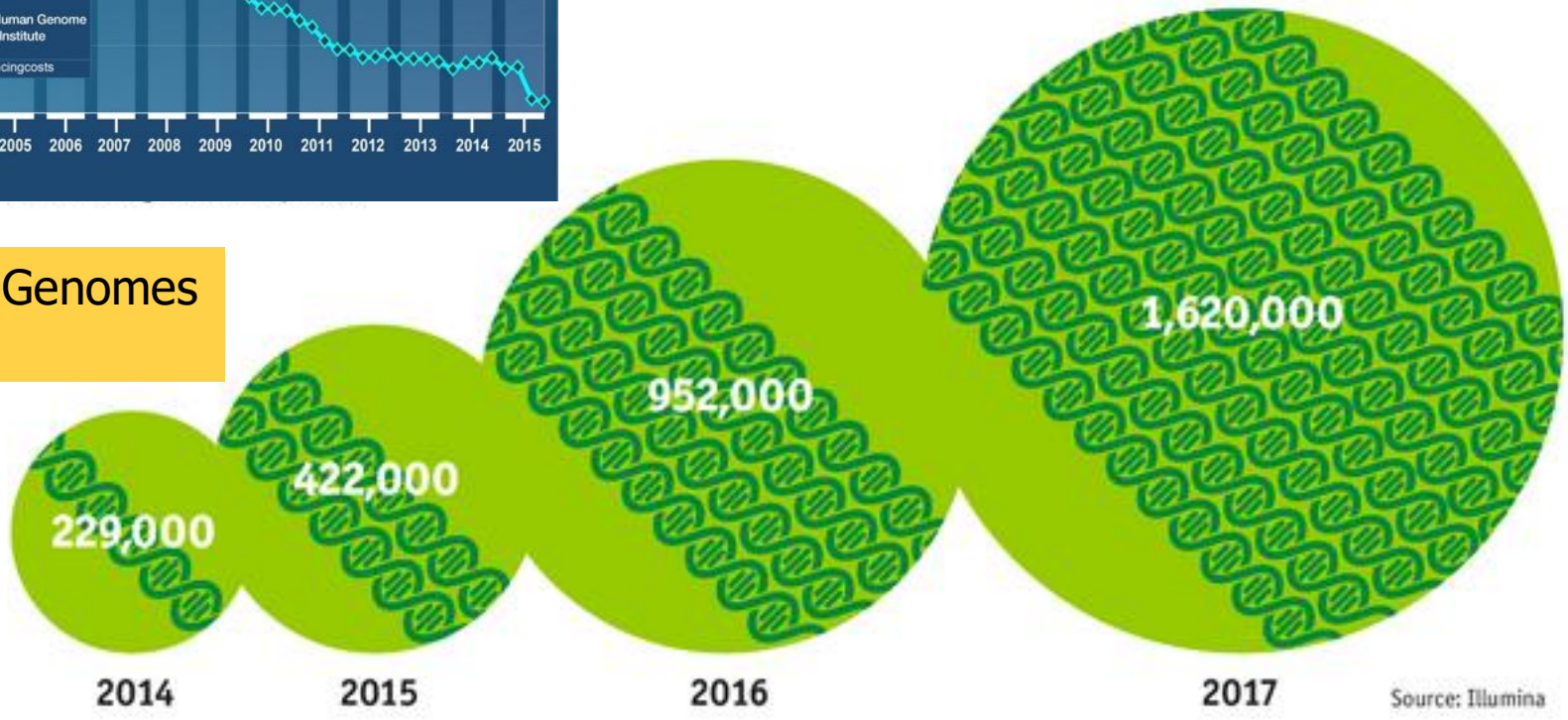
Huge Demand for Performance & Efficiency



development of new sequencing technologies

Oxford Nanopore MinION

Number of Genomes Sequenced



The Economist

http://www.economist.com/news/21631808-so-much-genetic-data-so-many-uses-genes-unzipped

Data Overwhelms Modern Machines …

- Storage/memory capability
- Communication capability
- Computation capability
- Greatly impacts robustness, energy, performance, cost

Data Movement Overwhelms Modern Machines

- Amirali Boroumand, Saugata Ghose, Youngsok Kim, Rachata Ausavarungnirun, Eric Shiu, Rahul Thakur, Daehyun Kim, Aki Kuusela, Allan Knies, Parthasarathy Ranganathan, and Onur Mutlu, **"Google Workloads for Consumer Devices: Mitigating Data Movement Bottlenecks"** *Proceedings of the 23rd International Conference on Architectural Support for Programming Languages and Operating Systems (**ASPLOS**)*, Williamsburg, VA, USA, March 2018.

62.7% of the total system energy is spent on data movement

Google Workloads for Consumer Devices: Mitigating Data Movement Bottlenecks

Amirali Boroumand[1] Saugata Ghose[1] Youngsok Kim[2]
Rachata Ausavarungnirun[1] Eric Shiu[3] Rahul Thakur[3] Daehyun Kim[4,3]
Aki Kuusela[3] Allan Knies[3] Parthasarathy Ranganathan[3] Onur Mutlu[5,1]

Data Movement Overwhelms Accelerators

- Amirali Boroumand, Saugata Ghose, Berkin Akin, Ravi Narayanaswami, Geraldo F. Oliveira, Xiaoyu Ma, Eric Shiu, and Onur Mutlu,
  "Google Neural Network Models for Edge Devices: Analyzing and Mitigating Machine Learning Inference Bottlenecks"
  *Proceedings of the 30th International Conference on Parallel Architectures and Compilation Techniques (**PACT**)*, Virtual, September 2021.
  [Slides (pptx) (pdf)]
  [Talk Video (14 minutes)]

> 90% of the total system energy is spent on memory in large ML models

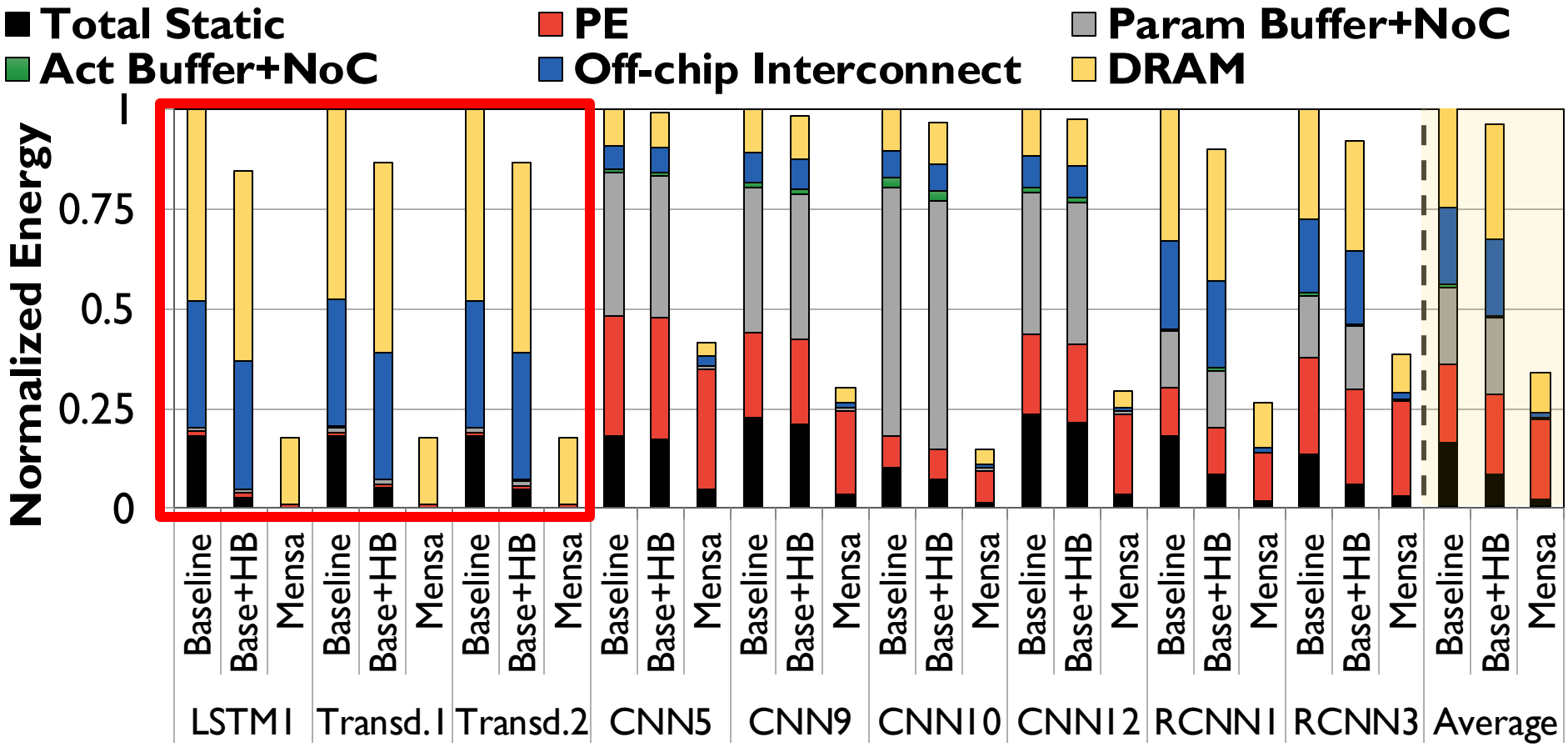
Google Neural Network Models for Edge Devices: Analyzing and Mitigating Machine Learning Inference Bottlenecks

Amirali Boroumand[†◇] Saugata Ghose[‡] Berkin Akin[§] Ravi Narayanaswami[§]
Geraldo F. Oliveira[⋆] Xiaoyu Ma[§] Eric Shiu[§] Onur Mutlu[⋆†]

[†]*Carnegie Mellon Univ.* [◇]*Stanford Univ.* [‡]*Univ. of Illinois Urbana-Champaign* [§]*Google* [⋆]*ETH Zürich*

https://arxiv.org/pdf/2109.14320

Example Energy Breakdowns



In LSTMs and Transducers used by Google,
>90% energy spent on off-chip interconnect and DRAM

https://arxiv.org/pdf/2109.14320

An Intelligent Architecture Handles Data Well

Corollaries: Computing Systems Today …

- Are processor-centric vs. **data-centric**
- Make designer-dictated decisions vs. **data-driven**
- Make component-based myopic decisions vs. **data-aware**

Architectures for Intelligent Machines

Data-centric

Data-driven

Data-aware

A Blueprint for Fundamentally Better Architectures

- Onur Mutlu,
  "Intelligent Architectures for Intelligent Computing Systems"
  *Invited Paper in Proceedings of the Design, Automation, and Test in Europe Conference (**DATE**)*, Virtual, February 2021.
  [Slides (pptx) (pdf)]
  [IEDM Tutorial Slides (pptx) (pdf)]
  [Short DATE Talk Video (11 minutes)]
  [Longer IEDM Tutorial Video (1 hr 51 minutes)]

Intelligent Architectures for Intelligent Computing Systems

Onur Mutlu
ETH Zurich
omutlu@gmail.com

Our Goals: ML-Memory System Duo

1. Memory system design for AI/ML workloads/accelerators

→ in-depth exploration of memory system designs for cutting-edge and emerging machine learning accelerators

→ more efficient on-chip and off-chip memory systems

2. AI/ML techniques for improving memory system designs

→ comprehensive look at memory system design to make it data driven, i.e., based on machine learning

→ more effective cache/memory/prefetch/thread controllers and data/resource management/mapping/scheduling policies

Two Major Directions

1. Memory system design for AI/ML workloads/accelerators

2. AI/ML techniques for improving memory system designs

Data-Driven (Self-Optimizing) Architectures

System Architecture Design Today

- Human-driven
  - Humans design the policies (how to do things)
- Many (too) simple, short-sighted policies all over the system
- No automatic data-driven policy learning
- (Almost) no learning: cannot take lessons from past actions

Can we design fundamentally intelligent architectures?

An Intelligent Architecture

- Data-driven
  - Machine learns the "best" policies (how to do things)
- Sophisticated, workload-driven, changing, far-sighted policies
- Automatic data-driven policy learning
- All controllers are intelligent data-driven agents

We need to rethink design (of all controllers)

Self-Optimizing Memory Controllers

- Engin Ipek, Onur Mutlu, José F. Martínez, and Rich Caruana,
  "Self Optimizing Memory Controllers: A Reinforcement Learning Approach"
  *Proceedings of the 35th International Symposium on Computer Architecture (**ISCA**)*, pages 39-50, Beijing, China, June 2008.
  Selected to the ISCA-50 25-Year Retrospective Issue covering 1996-2020 in 2023 (Retrospective (pdf) Full Issue).


Self-Optimizing Memory Controllers: A Reinforcement Learning Approach

Engin İpek[1,2] Onur Mutlu[2] José F. Martínez[1] Rich Caruana[1]

[1]Cornell University, Ithaca, NY 14850 USA
[2] Microsoft Research, Redmond, WA 98052 USA

Self-Optimizing Memory Prefetchers

Rahul Bera, Konstantinos Kanellopoulos, Anant Nori, Taha Shahroodi, Sreenivas Subramoney, and Onur Mutlu,
"Pythia: A Customizable Hardware Prefetching Framework Using Online Reinforcement Learning"
Proceedings of the 54th International Symposium on Microarchitecture (***MICRO***), Virtual, October 2021.
[Slides (pptx) (pdf)]
[Short Talk Slides (pptx) (pdf)]
[Lightning Talk Slides (pptx) (pdf)]
[Talk Video (20 minutes)]
[Lightning Talk Video (1.5 minutes)]
[Pythia Source Code (Officially Artifact Evaluated with All Badges)]
[arXiv version]
Officially artifact evaluated as available, reusable and reproducible.

Pythia: A Customizable Hardware Prefetching Framework Using Online Reinforcement Learning

Rahul Bera[1] Konstantinos Kanellopoulos[1] Anant V. Nori[2] Taha Shahroodi[3,1]
Sreenivas Subramoney[2] Onur Mutlu[1]
[1]ETH Zürich [2]Processor Architecture Research Labs, Intel Labs [3]TU Delft

https://arxiv.org/pdf/2109.12021.pdf

Learning-Based Off-Chip Load Predictors

- Rahul Bera, Konstantinos Kanellopoulos, Shankar Balachandran, David Novo, Ataberk Olgun, Mohammad Sadrosadati, and Onur Mutlu,
  "Hermes: Accelerating Long-Latency Load Requests via Perceptron-Based Off-Chip Load Prediction"
  Proceedings of the 55th International Symposium on Microarchitecture (***MICRO***), Chicago, IL, USA, October 2022.
  [Slides (pptx) (pdf)]
  [Longer Lecture Slides (pptx) (pdf)]
  [Talk Video (12 minutes)]
  [Lecture Video (25 minutes)]
  [arXiv version]
  [Source Code (Officially Artifact Evaluated with All Badges)]
  Officially artifact evaluated as available, reusable and reproducible.
  Best paper award at MICRO 2022.

Hermes: Accelerating Long-Latency Load Requests via Perceptron-Based Off-Chip Load Prediction

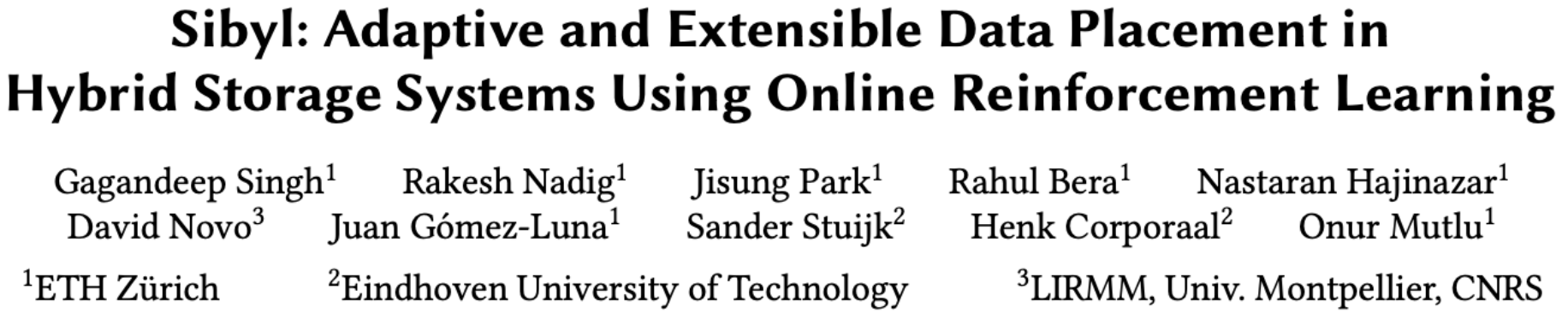
Rahul Bera[1] Konstantinos Kanellopoulos[1] Shankar Balachandran[2] David Novo[3]
Ataberk Olgun[1] Mohammad Sadrosadati[1] Onur Mutlu[1]

[1]ETH Zürich [2]Intel Processor Architecture Research Lab [3]LIRMM, Univ. Montpellier, CNRS

https://arxiv.org/pdf/2209.00188.pdf

Self-Optimizing Hybrid SSD Controllers

Gagandeep Singh, Rakesh Nadig, Jisung Park, Rahul Bera, Nastaran Hajinazar, David Novo, Juan Gomez-Luna, Sander Stuijk, Henk Corporaal, and Onur Mutlu,
"Sibyl: Adaptive and Extensible Data Placement in Hybrid Storage Systems Using Online Reinforcement Learning"
*Proceedings of the 49th International Symposium on Computer Architecture (**ISCA**)*, New York, June 2022.
[Slides (pptx) (pdf)]
[arXiv version]
[Sibyl Source Code]
[Talk Video (16 minutes)]



Sibyl: Adaptive and Extensible Data Placement in Hybrid Storage Systems Using Online Reinforcement Learning

Gagandeep Singh[1] Rakesh Nadig[1] Jisung Park[1] Rahul Bera[1] Nastaran Hajinazar[1]
David Novo[3] Juan Gómez-Luna[1] Sander Stuijk[2] Henk Corporaal[2] Onur Mutlu[1]

[1]ETH Zürich [2]Eindhoven University of Technology [3]LIRMM, Univ. Montpellier, CNRS

https://arxiv.org/pdf/2205.07394.pdf

A Blueprint for Fundamentally Better Architectures

- Onur Mutlu,
  "Intelligent Architectures for Intelligent Computing Systems"
  *Invited Paper in Proceedings of the Design, Automation, and Test in Europe Conference (**DATE**)*, Virtual, February 2021.
  [Slides (pptx) (pdf)]
  [IEDM Tutorial Slides (pptx) (pdf)]
  [Short DATE Talk Video (11 minutes)]
  [Longer IEDM Tutorial Video (1 hr 51 minutes)]

Fundamentally Better Architectures

Data-centric

Data-driven

Data-aware

Pythia: Prefetching using Reinforcement Learning

Self-Optimizing Memory Prefetchers

Rahul Bera, Konstantinos Kanellopoulos, Anant Nori, Taha Shahroodi, Sreenivas Subramoney, and Onur Mutlu,
"Pythia: A Customizable Hardware Prefetching Framework Using Online Reinforcement Learning"
Proceedings of the 54th International Symposium on Microarchitecture (***MICRO***), Virtual, October 2021.
[Slides (pptx) (pdf)]
[Short Talk Slides (pptx) (pdf)]
[Lightning Talk Slides (pptx) (pdf)]
[Talk Video (20 minutes)]
[Lightning Talk Video (1.5 minutes)]
[Pythia Source Code (Officially Artifact Evaluated with All Badges)]
[arXiv version]
Officially artifact evaluated as available, reusable and reproducible.

Pythia: A Customizable Hardware Prefetching Framework Using Online Reinforcement Learning

Rahul Bera[1] Konstantinos Kanellopoulos[1] Anant V. Nori[2] Taha Shahroodi[3,1]
Sreenivas Subramoney[2] Onur Mutlu[1]
[1]ETH Zürich [2]Processor Architecture Research Labs, Intel Labs [3]TU Delft

https://arxiv.org/pdf/2109.12021.pdf

Pythia

A Customizable Hardware Prefetching Framework Using Online Reinforcement Learning

Rahul Bera, Konstantinos Kanellopoulos, Anant V. Nori,
Taha Shahroodi, Sreenivas Subramoney, Onur Mutlu

https://github.com/CMU-SAFARI/Pythia

SAFARI
SAFARI Research Group
safari.ethz.ch

ETH zürich

intel

TU Delft

1 Mainly use one program context info. for prediction

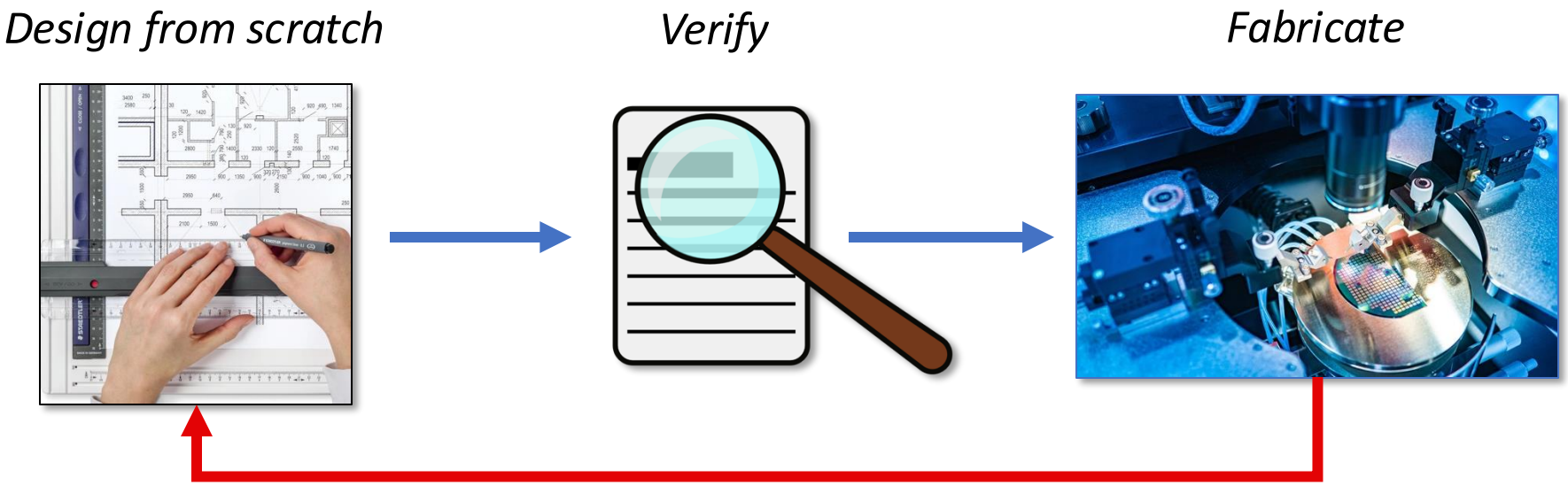
2 Lack inherent system awareness

3 Lack in-silicon customizability

Why do prefetchers not perform well?

Lack of In-silicon Customizability

- Feature **statically** selected at design time
  - **Rigid hardware** designed specifically to exploit that feature
- **No way to change** program feature and/or change prefetcher's objective **in silicon**
  - **Cannot adapt** to a wide range of workload demands



Our Goal

A **prefetching framework** that can:

1. Learn to prefetch using **multiple features** and **inherent system-level feedback** information

2. Be **easily customized in silicon** to use different features and/or change prefetcher's objectives

Our Proposal

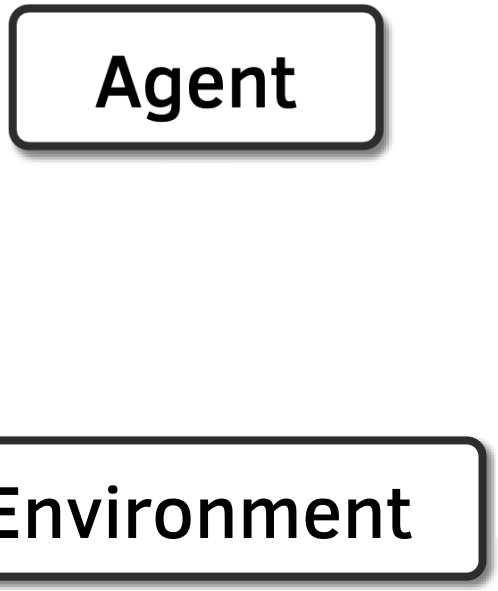
Pythia

Formulates prefetching as a
reinforcement learning problem

Pythia is named after the oracle of Delphi, who is known for her accurate prophecies
https://en.wikipedia.org/wiki/Pythia

Basics of Reinforcement Learning (RL)

- Algorithmic approach to learn to take an **action** in a given **situation** to maximize a numerical **reward**



- Agent stores **Q-values** for *every* state-action pair
  - **Expected return** for taking an action in a state
  - Given a state, selects action that provides **highest** Q-value

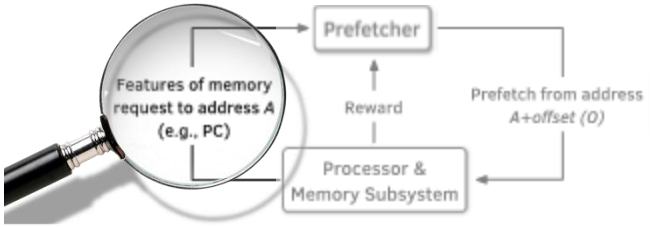
Formulating Prefetching as RL

What is State?

- ***k*-dimensional** vector of features

$$S \equiv \{\phi_S^1, \phi_S^2, \ldots, \phi_S^k\}$$

- Feature = control-flow + data-flow



- **Control-flow examples**
  - PC
  - Branch PC
  - Last-3 PCs, …
- **Data-flow examples**
  - Cacheline address
  - Physical page number
  - Delta between two cacheline addresses
  - Last 4 deltas, …

What is Action?

Given a demand access to address A
the action is to **select prefetch offset "O"**

- **Action-space**: 127 actions in the range [-63, +63]
  - For a machine with 4KB page and 64B cacheline
- Upper and lower limits ensure prefetches do not cross **physical page boundary**
- A **zero offset** means **no prefetch** is generated
- We further **prune** action-space by design-space exploration

What is Reward?

- Defines the **objective** of Pythia

- Encapsulates two metrics:
  - **Prefetch usefulness** (e.g., accurate, late, out-of-page, …)
  - **System-level feedback** (e.g., mem. b/w usage, cache pollution, energy, …)

- We demonstrate Pythia with **memory bandwidth usage** as the system-level feedback in the paper

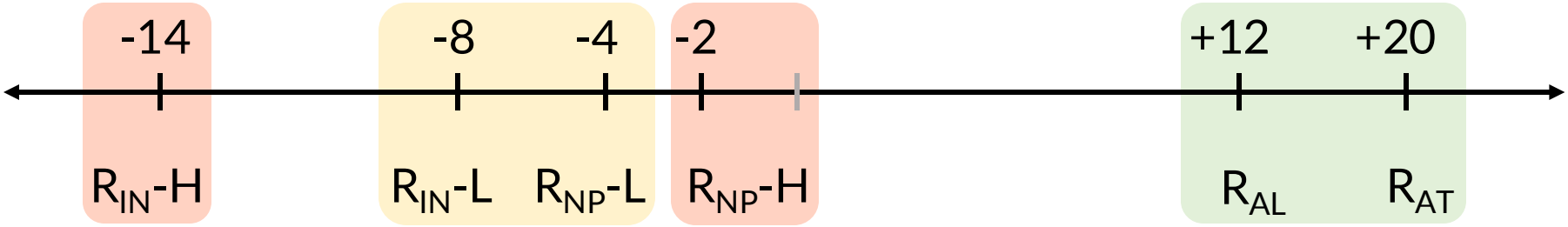
What is Reward?

- **Seven** distinct reward levels
  - *Accurate and timely* (R_{AT})
  - *Accurate but late* (R_{AL})
  - *Loss of coverage* (R_{CL})
  - *Inaccurate*
    - With low memory b/w usage (R_{IN}-L)
    - With high memory b/w usage (R_{IN}-H)
  - *No-prefetch*
    - With low memory b/w usage (R_{NP}-L)
    - With high memory b/w usage(R_{NP}-H)

- Values are set at design time via **automatic design-space exploration**
  - Can be **customized** further in silicon for higher performance

Steering Pythia's Objective via Reward Values

- Example reward configuration for
  - Generating **accurate prefetches**
  - Making **bandwidth-aware** prefetch decisions



AT = *Accurate & timely*; **AL** = *Accurate & late*; **NP** = *No-prefetching*; **IN** = *Inaccurate*;
H = *High mem. b/w*; **L** = *Low mem. b/w*

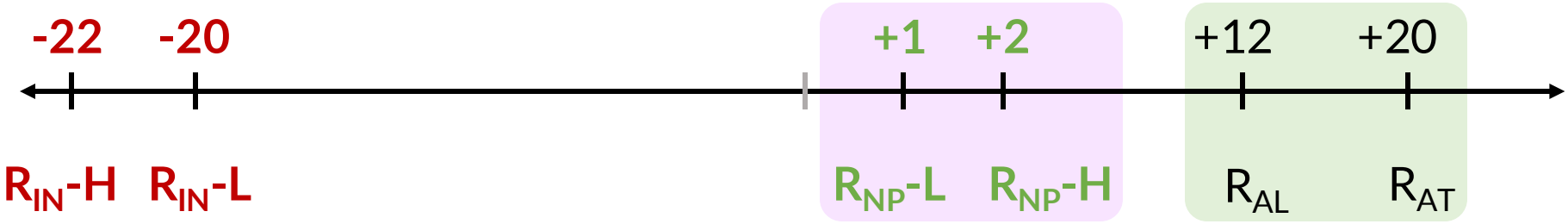
1 **Highly prefers to generate accurate prefetches**

2 **Prefers not to prefetch if memory bandwidth usage is low**

3 **Strongly prefers not to prefetch if memory bandwidth usage is high**

Steering Pythia's Objective via Reward Values

- Customizing reward values to make Pythia **conservative** towards prefetching



AT = *Accurate & timely*; **AL** = *Accurate & late*; **NP** = *No-prefetching*; **IN** = *Inaccurate*;
H = *High mem. b/w*; **L** = *Low mem. b/w*

1 **Highly prefers to generate accurate prefetches**

2 **Otherwise prefers not to prefetch**

Basic Pythia Configuration

- Derived from **automatic design-space exploration**

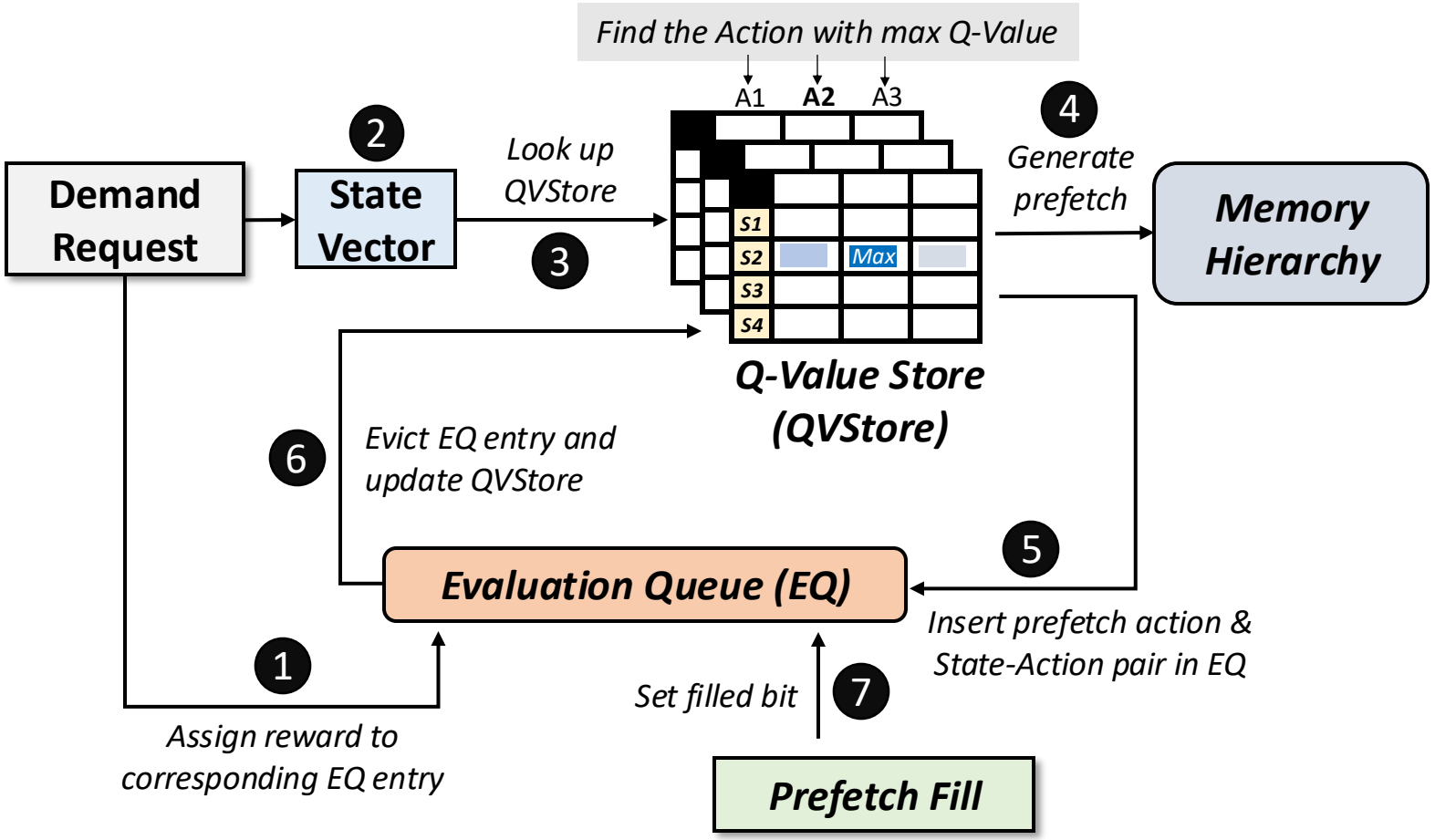
- **State:** 2 features
  - PC+Delta
  - Sequence of last-4 deltas

- **Actions:** 16 prefetch offsets
  - Ranging between -6 to +32. Including 0.

- **Rewards:**
  - R_{AT} = +20; R_{AL} = +12; R_{NP}-H=-2; R_{NP}-L=-4;
  - R_{IN}-H=-14; R_{IN}-L=-8; R_{CL}=-12

More Detailed Pythia Overview

- **Q-Value Store**: Records Q-values for *all* state-action pairs
- **Evaluation Queue**: A FIFO queue of recently-taken actions

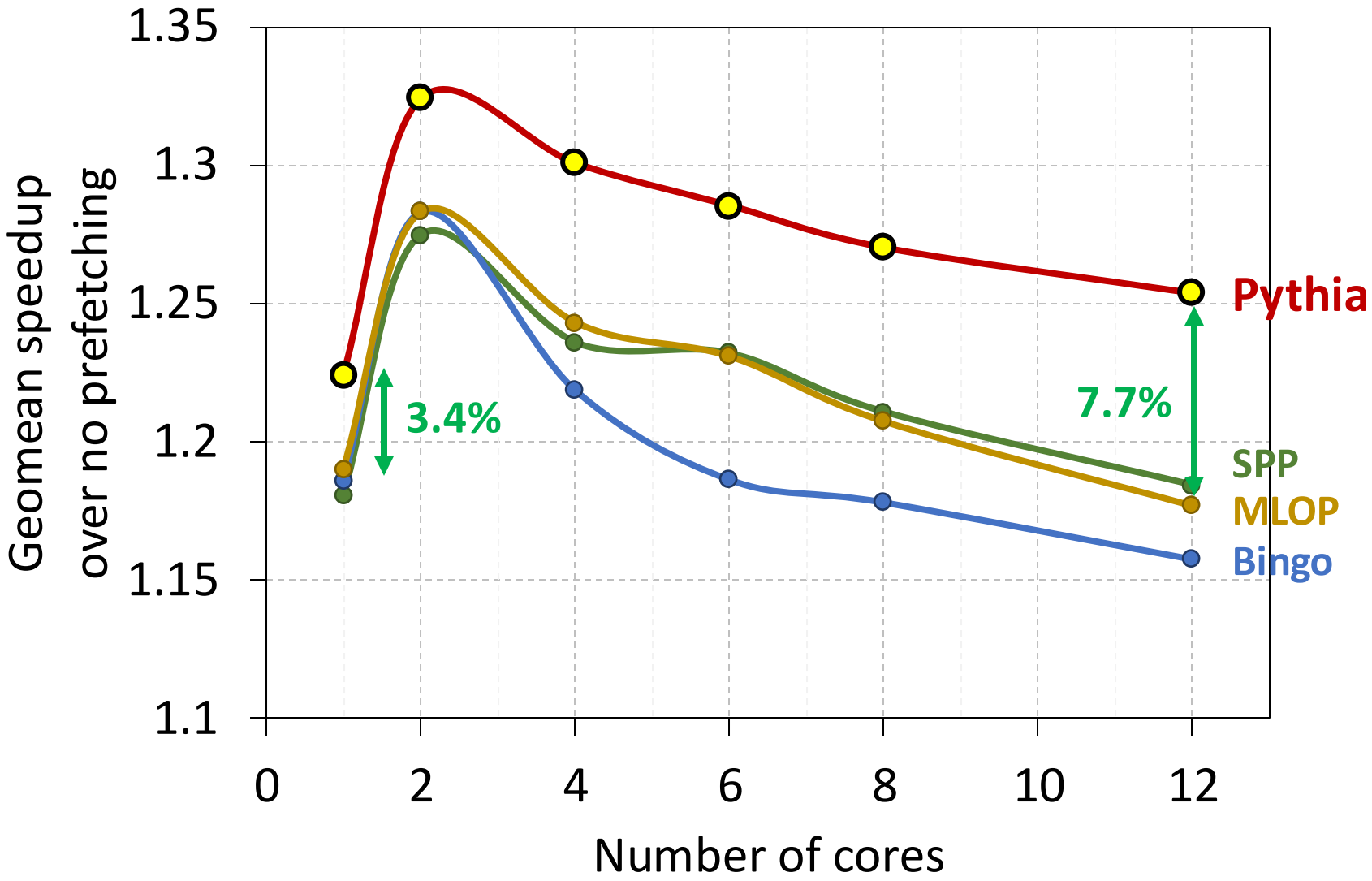


Simulation Methodology

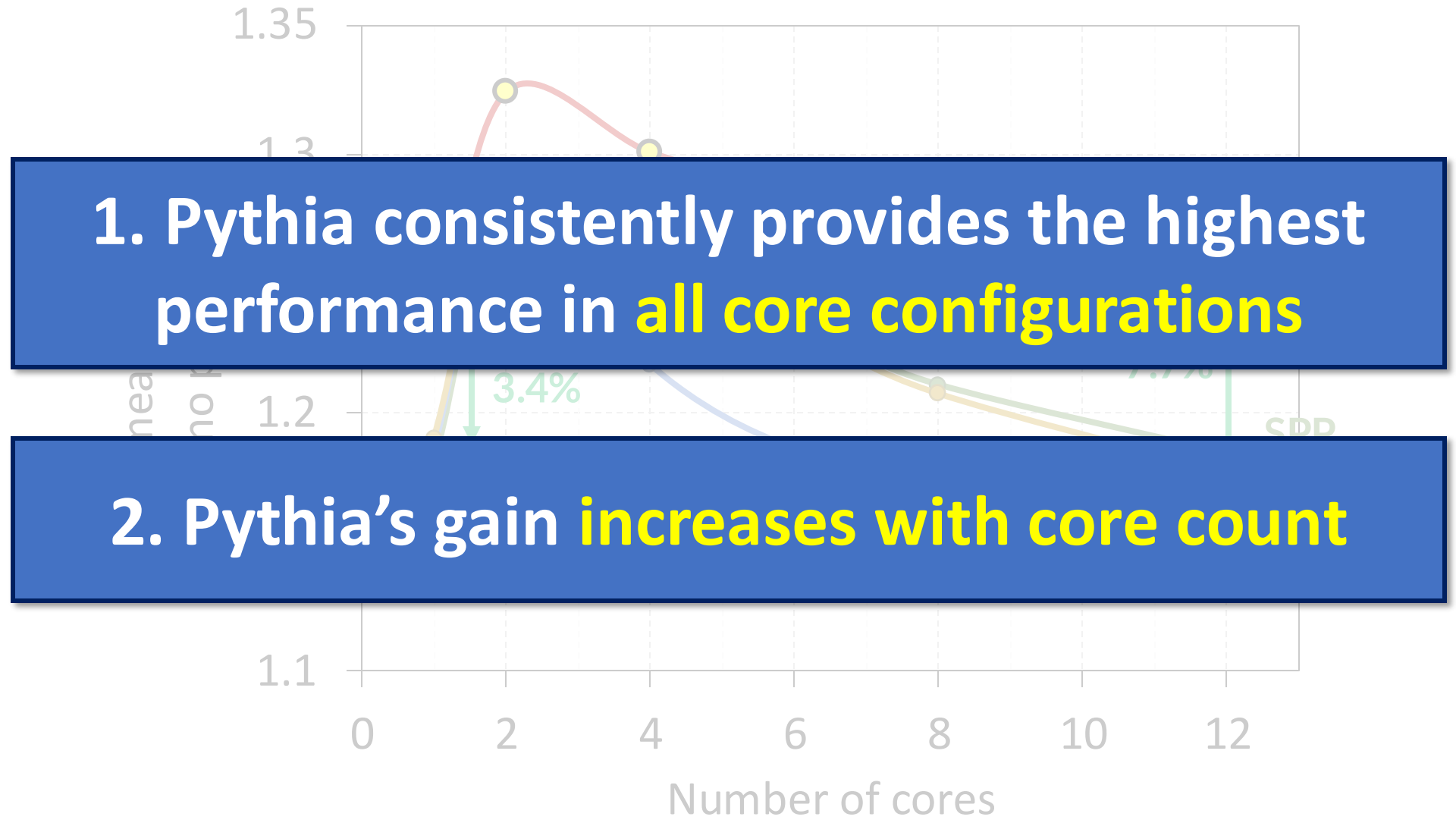
- **Champsim** [3] trace-driven simulator
- **150** single-core memory-intensive workload traces
  - SPEC CPU2006 and CPU2017
  - PARSEC 2.1
  - Ligra
  - Cloudsuite
- Homogeneous and heterogeneous multi-core mixes
- **Five** state-of-the-art prefetchers
  - SPP [Kim+, MICRO'16]
  - Bingo [Bakhshalipour+, HPCA'19]
  - MLOP [Shakerinava+, 3rd Prefetching Championship, 2019]
  - SPP+DSPatch [Bera+, MICRO'19]
  - SPP+PPF [Bhatia+, ISCA'20]

[3] https://github.com/ChampSim/ChampSim

Performance with Varying Core Count



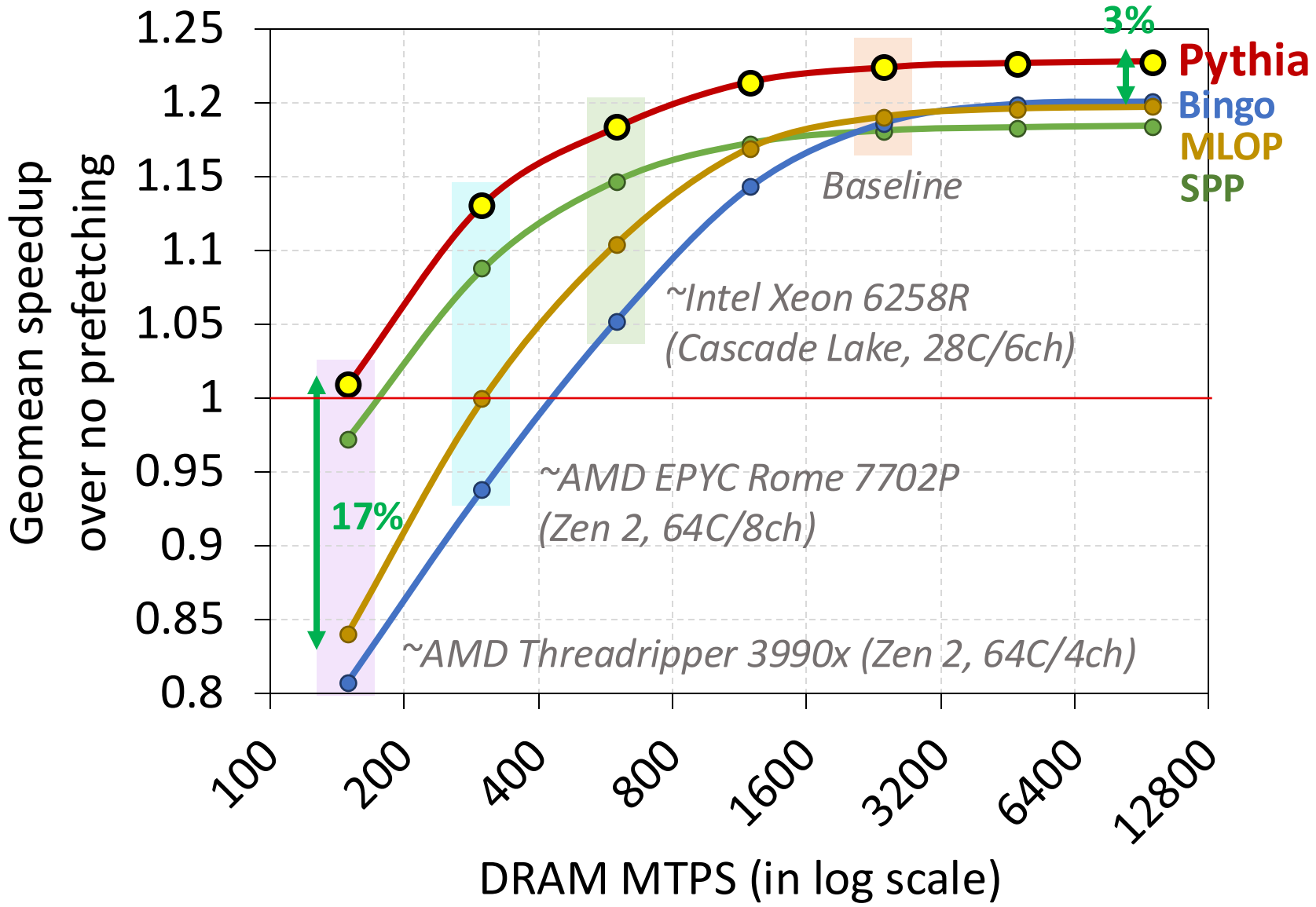
Performance with Varying Core Count



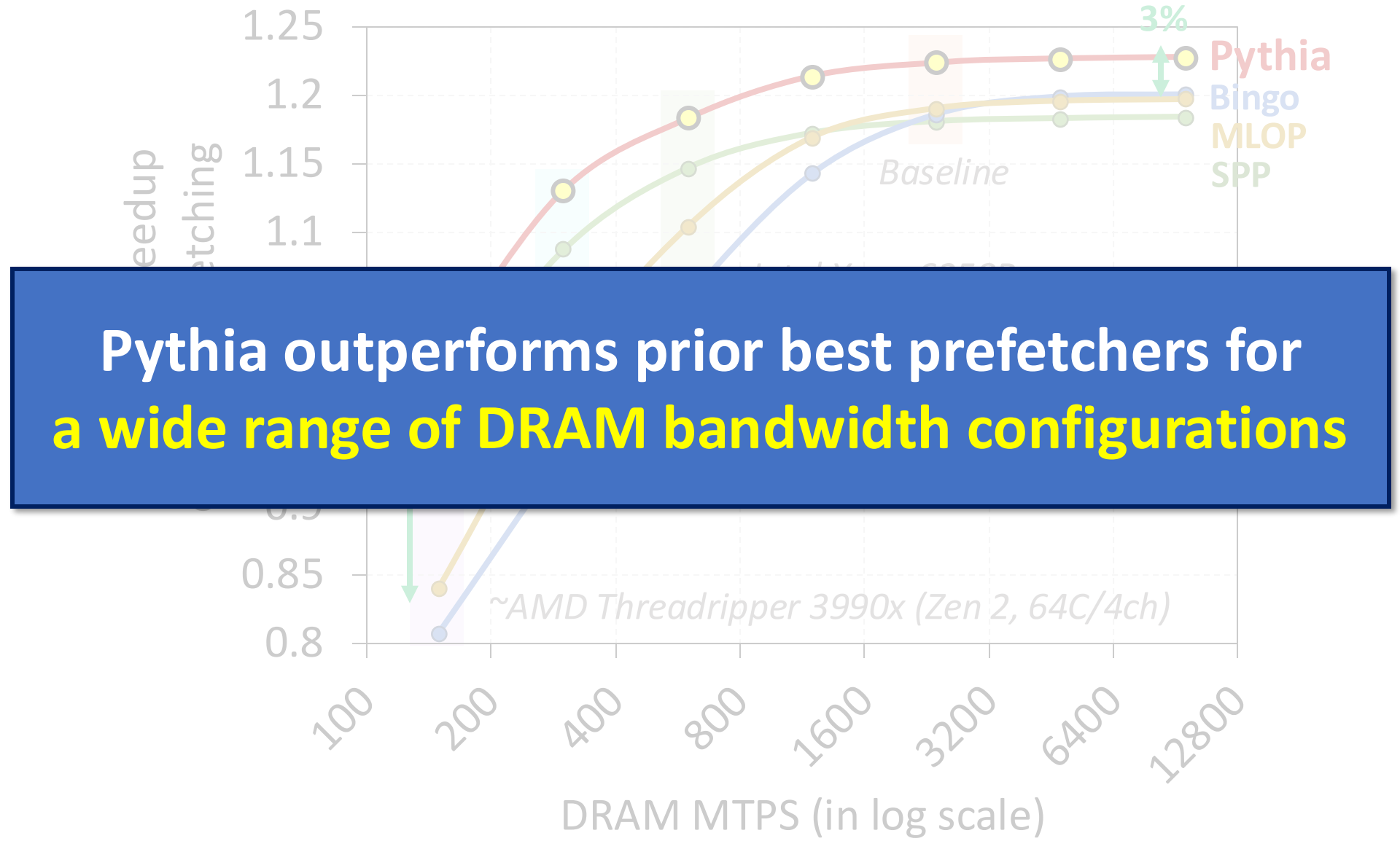
1. Pythia consistently provides the highest performance in all core configurations

2. Pythia's gain increases with core count

Performance with Varying DRAM Bandwidth

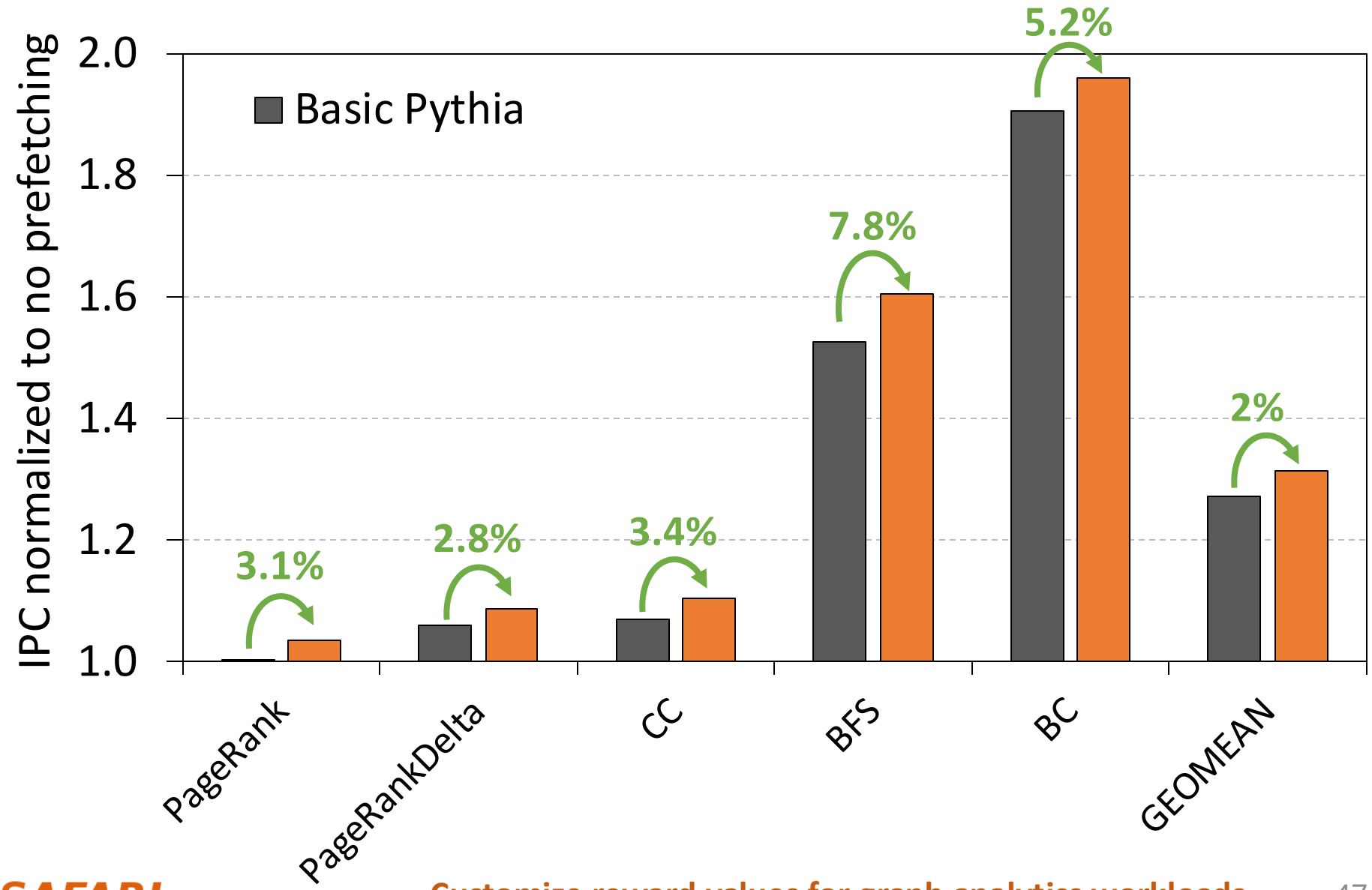


Performance with Varying DRAM Bandwidth



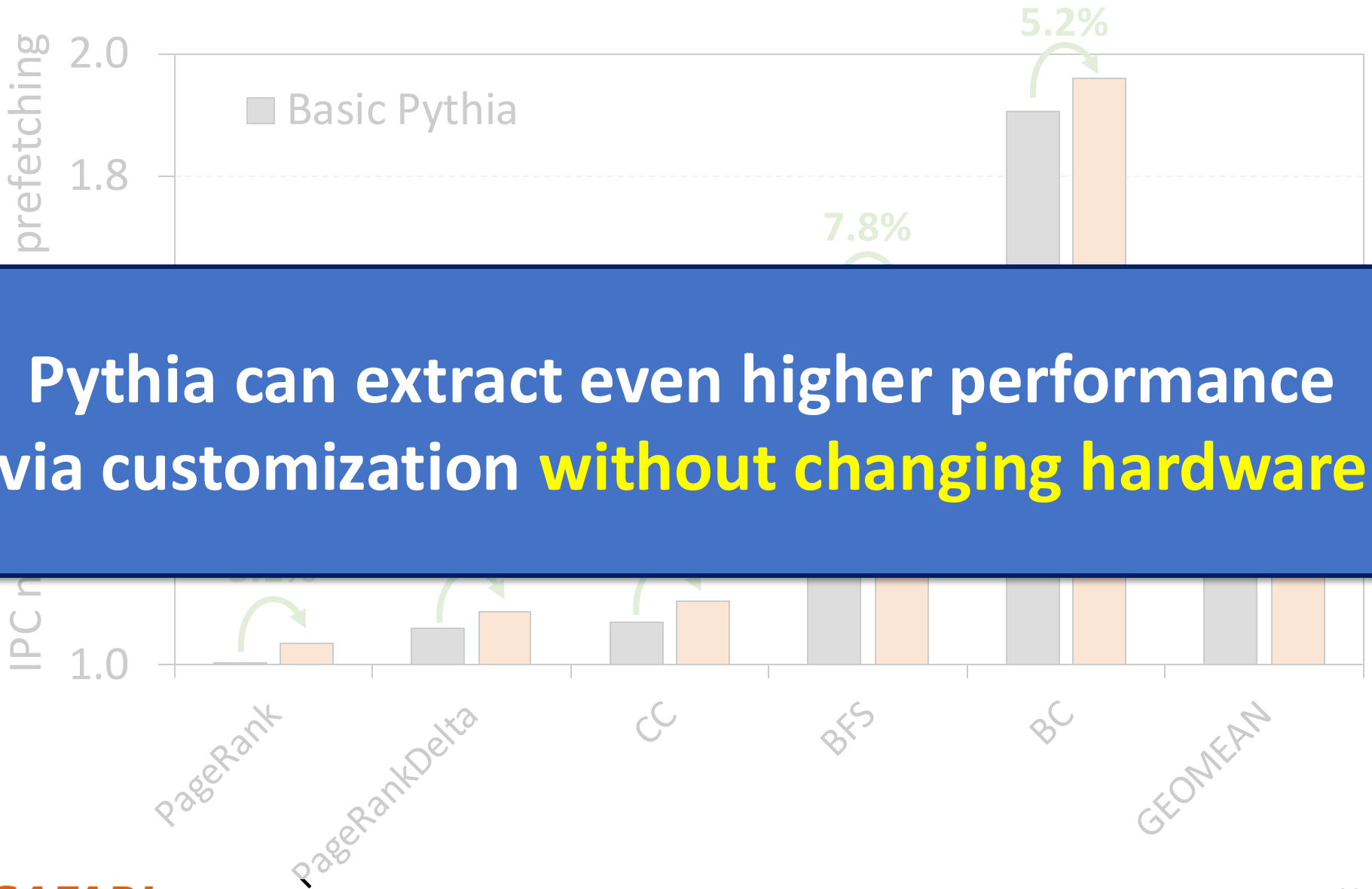
Pythia outperforms prior best prefetchers for a wide range of DRAM bandwidth configurations

Performance Improvement via Customization



Customize reward values for graph analytics workloads

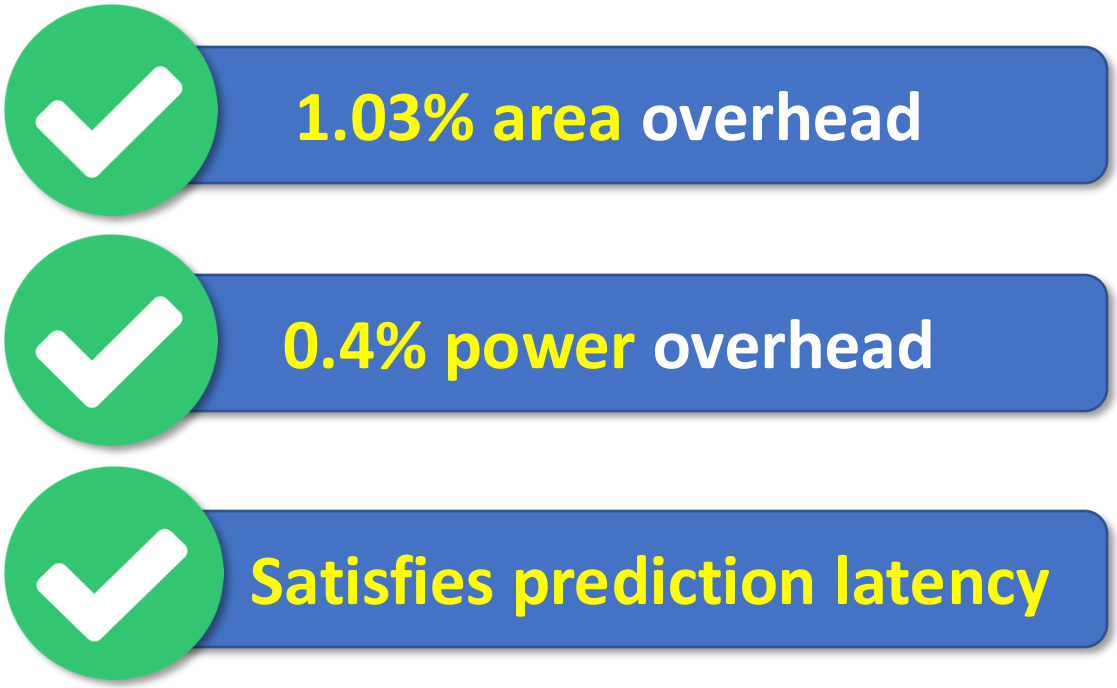
Performance Improvement via Customization



Pythia can extract even higher performance via customization without changing hardware

Pythia's Overhead

- **25.5 KB** of total metadata storage **per core**
  - Only simple tables
- We also model functionally-accurate Pythia with full complexity in **Chisel** [4] HDL



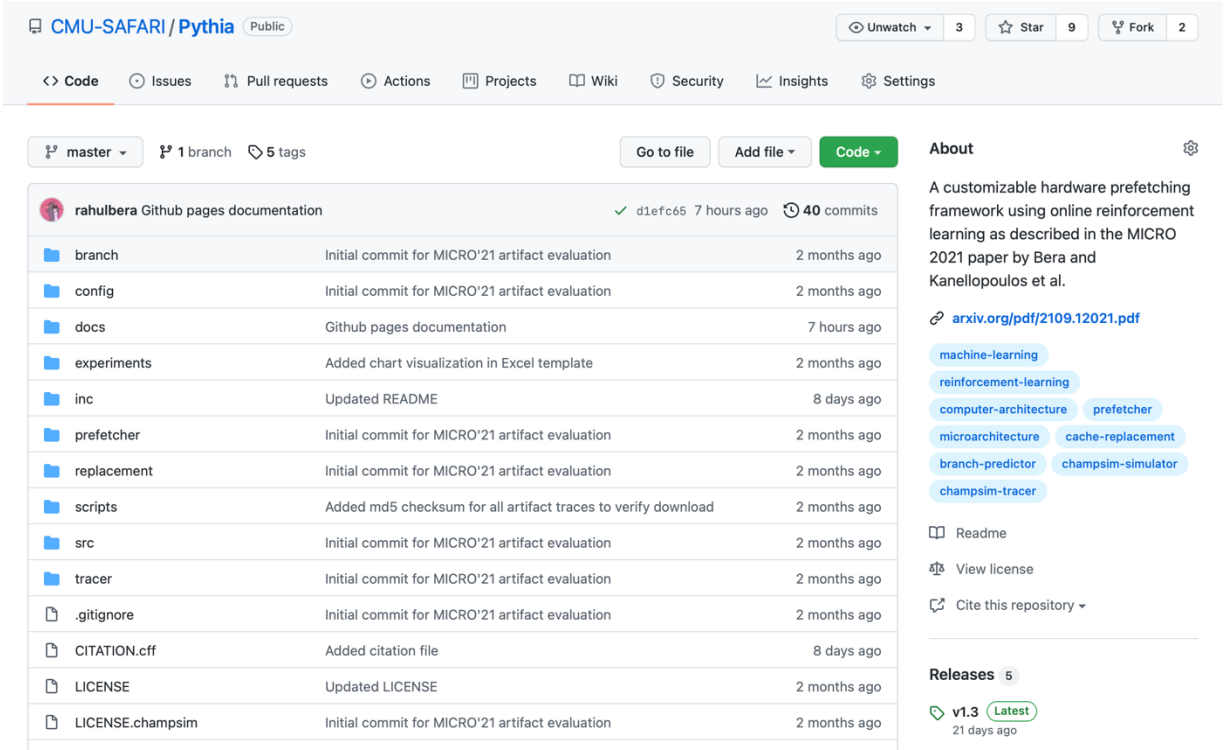
of a desktop-class 4-core Skylake processor (Xeon D2132IT, 60W)

[4] https://www.chisel-lang.org

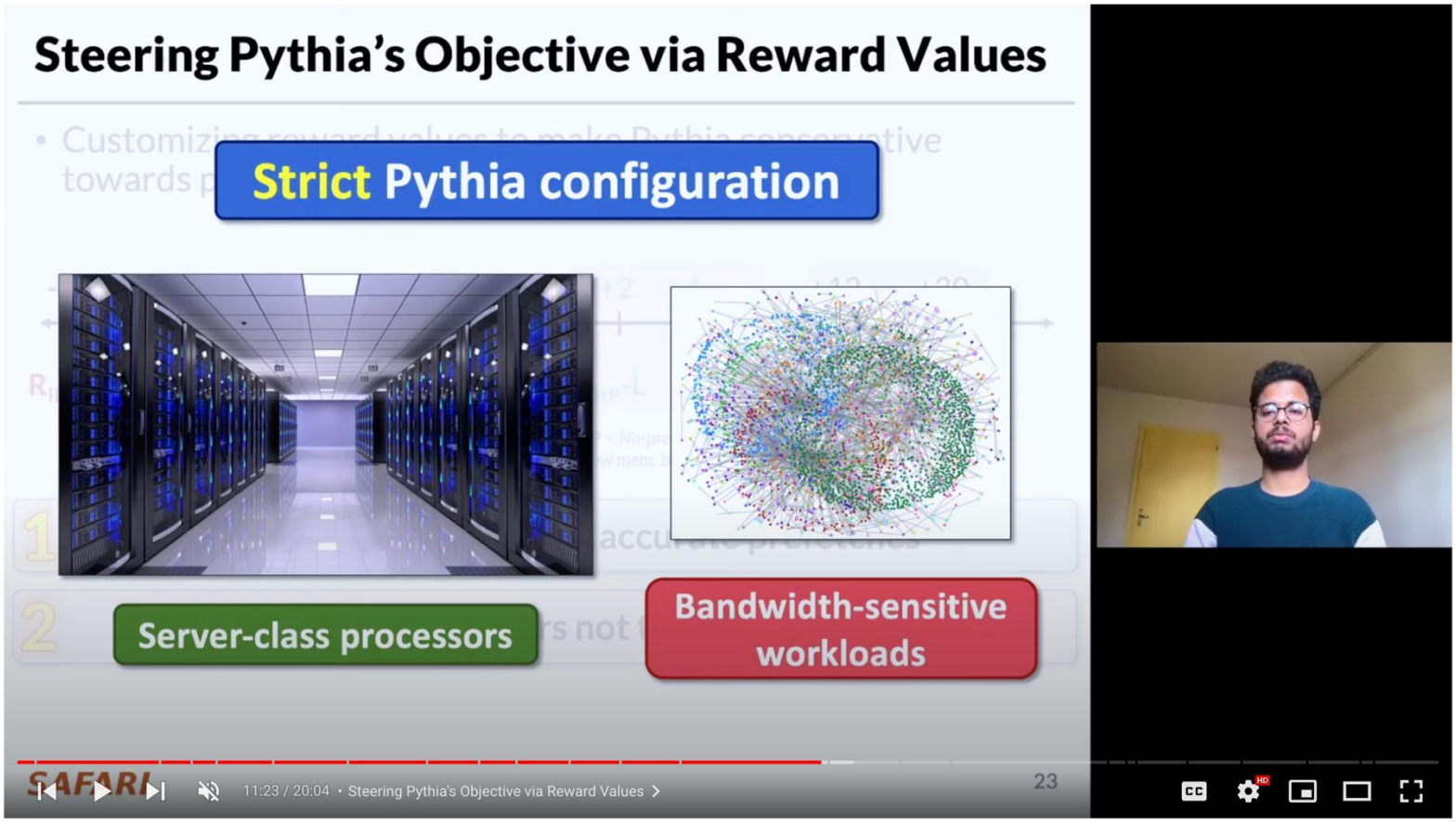
Pythia is Open Source

https://github.com/CMU-SAFARI/Pythia

- MICRO'21 **artifact evaluated**
- **Champsim source** code + **Chisel** modeling code
- **All traces** used for evaluation



Pythia Talk Video



MICRO 2021 Conference Presentations

Pythia: A Customizable Prefetching Framework Using Reinforcement Learning - MICRO'21 Long Talk

Onur Mutlu Lectures
28.9K subscribers

Analytics | Edit video | 22 | Share | Download | Clip | Save

661 views 11 months ago
Talk: "Pythia: A Customizable Hardware Prefetching Framework Using Online Reinforcement Learning"
Full Conference Talk at MICRO 2021 by Rahul Bera

https://www.youtube.com/watch?v=6UMFRW3VFPo&list=PL5Q2soXY2Zi--0LrXSQ9sST3N0k0bXp51&index=8

A Lot More in the Pythia Paper

Rahul Bera, Konstantinos Kanellopoulos, Anant Nori, Taha Shahroodi, Sreenivas Subramoney, and Onur Mutlu,
"Pythia: A Customizable Hardware Prefetching Framework Using Online Reinforcement Learning"
Proceedings of the 54th International Symposium on Microarchitecture (***MICRO***), Virtual, October 2021.
[Slides (pptx) (pdf)]
[Short Talk Slides (pptx) (pdf)]
[Lightning Talk Slides (pptx) (pdf)]
[Talk Video (20 minutes)]
[Lightning Talk Video (1.5 minutes)]
[Pythia Source Code (Officially Artifact Evaluated with All Badges)]
[arXiv version]
Officially artifact evaluated as available, reusable and reproducible.

Pythia: A Customizable Hardware Prefetching Framework Using Online Reinforcement Learning

Rahul Bera[1] Konstantinos Kanellopoulos[1] Anant V. Nori[2] Taha Shahroodi[3,1]
Sreenivas Subramoney[2] Onur Mutlu[1]
[1]ETH Zürich [2]Processor Architecture Research Labs, Intel Labs [3]TU Delft

https://arxiv.org/pdf/2109.12021.pdf

Pythia

A Customizable Hardware Prefetching Framework Using Online Reinforcement Learning

Rahul Bera, Konstantinos Kanellopoulos, Anant V. Nori,
Taha Shahroodi, Sreenivas Subramoney, Onur Mutlu

https://github.com/CMU-SAFARI/Pythia

SAFARI
SAFARI Research Group
safari.ethz.ch

ETH zürich

intel

TU Delft

Hermes: Perceptron-Based Off-Chip Load Prediction

Learning-Based Off-Chip Load Predictors

- Rahul Bera, Konstantinos Kanellopoulos, Shankar Balachandran, David Novo, Ataberk Olgun, Mohammad Sadrosadati, and Onur Mutlu,
  "Hermes: Accelerating Long-Latency Load Requests via Perceptron-Based Off-Chip Load Prediction"
  Proceedings of the 55th International Symposium on Microarchitecture (***MICRO***), Chicago, IL, USA, October 2022.
  [Slides (pptx) (pdf)]
  [Longer Lecture Slides (pptx) (pdf)]
  [Talk Video (12 minutes)]
  [Lecture Video (25 minutes)]
  [arXiv version]
  [Source Code (Officially Artifact Evaluated with All Badges)]
  Officially artifact evaluated as available, reusable and reproducible.
  Best paper award at MICRO 2022.

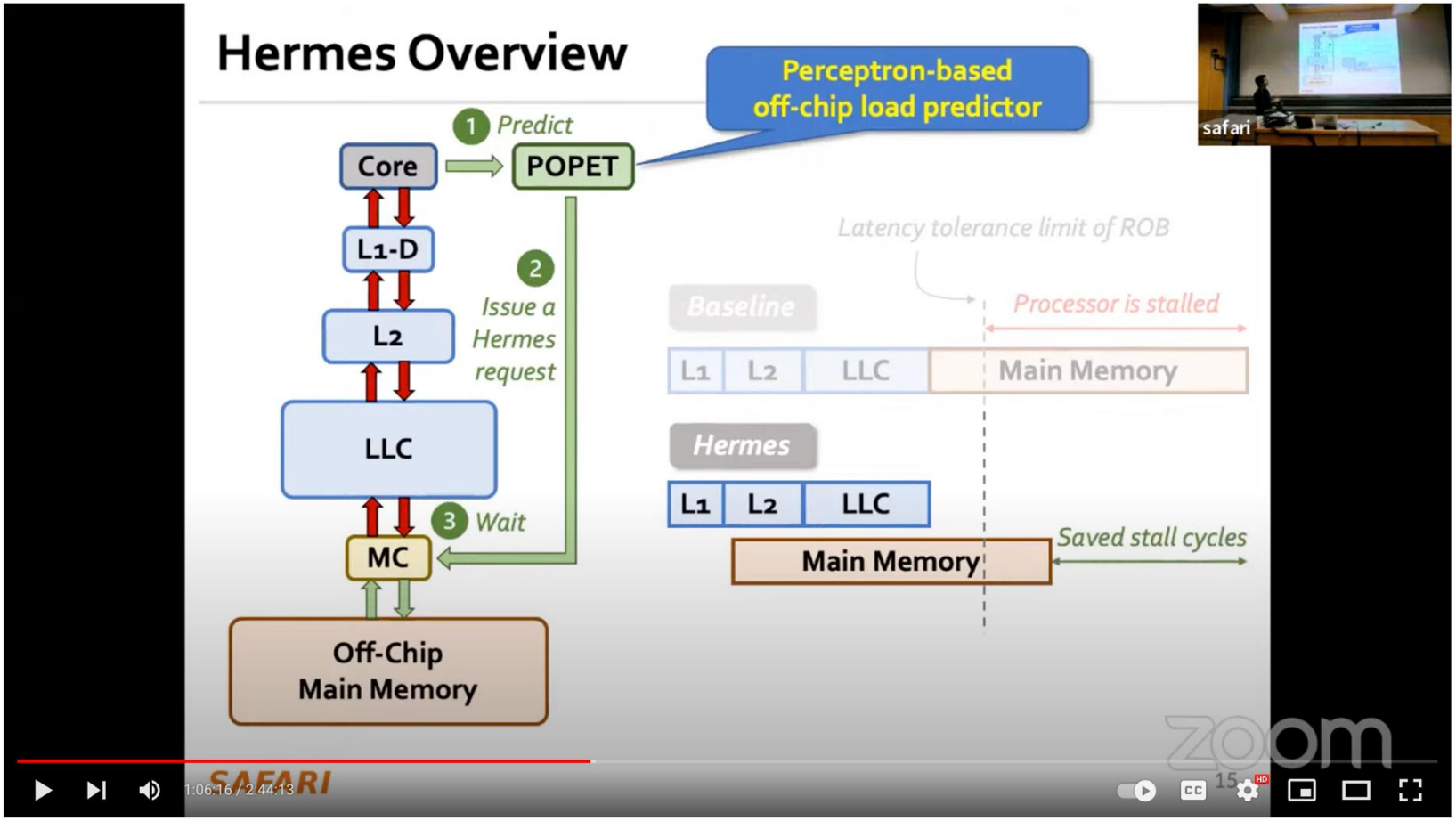
Hermes: Accelerating Long-Latency Load Requests via Perceptron-Based Off-Chip Load Prediction

Rahul Bera[1] Konstantinos Kanellopoulos[1] Shankar Balachandran[2] David Novo[3]
Ataberk Olgun[1] Mohammad Sadrosadati[1] Onur Mutlu[1]

[1]ETH Zürich [2]Intel Processor Architecture Research Lab [3]LIRMM, Univ. Montpellier, CNRS

https://arxiv.org/pdf/2209.00188.pdf

Hermes Talk Video



Computer Architecture - Lecture 18: Cutting-Edge Research in Computer Architecture (Fall 2022)

Onur Mutlu Lectures
32.9K subscribers
Analytics | Edit video
23 | Share | Download | Clip | Save

2.4K views Streamed 5 months ago Livestream - Computer Architecture - ETH Zürich (Fall 2022)
Computer Architecture, ETH Zürich, Fall 2022 (https://safari.ethz.ch/architecture/f...)

https://www.youtube.com/watch?v=PWWBtrL60dQ&t=3609s

HERMES

Accelerating Long-Latency Load Requests via Perceptron-Based Off-Chip Load Prediction

Rahul Bera, Konstantinos Kanellopoulos, Shankar Balachandran, David Novo, Ataberk Olgun, Mohammad Sadrosadati, Onur Mutlu

https://github.com/CMU-SAFARI/Hermes

SAFARI Research Group
safari.ethz.ch

ETH zürich

intel

LIRMM

Problem

Long-latency off-chip load requests

Often **stall** processor by
blocking instruction retirement from
Reorder Buffer (ROB)

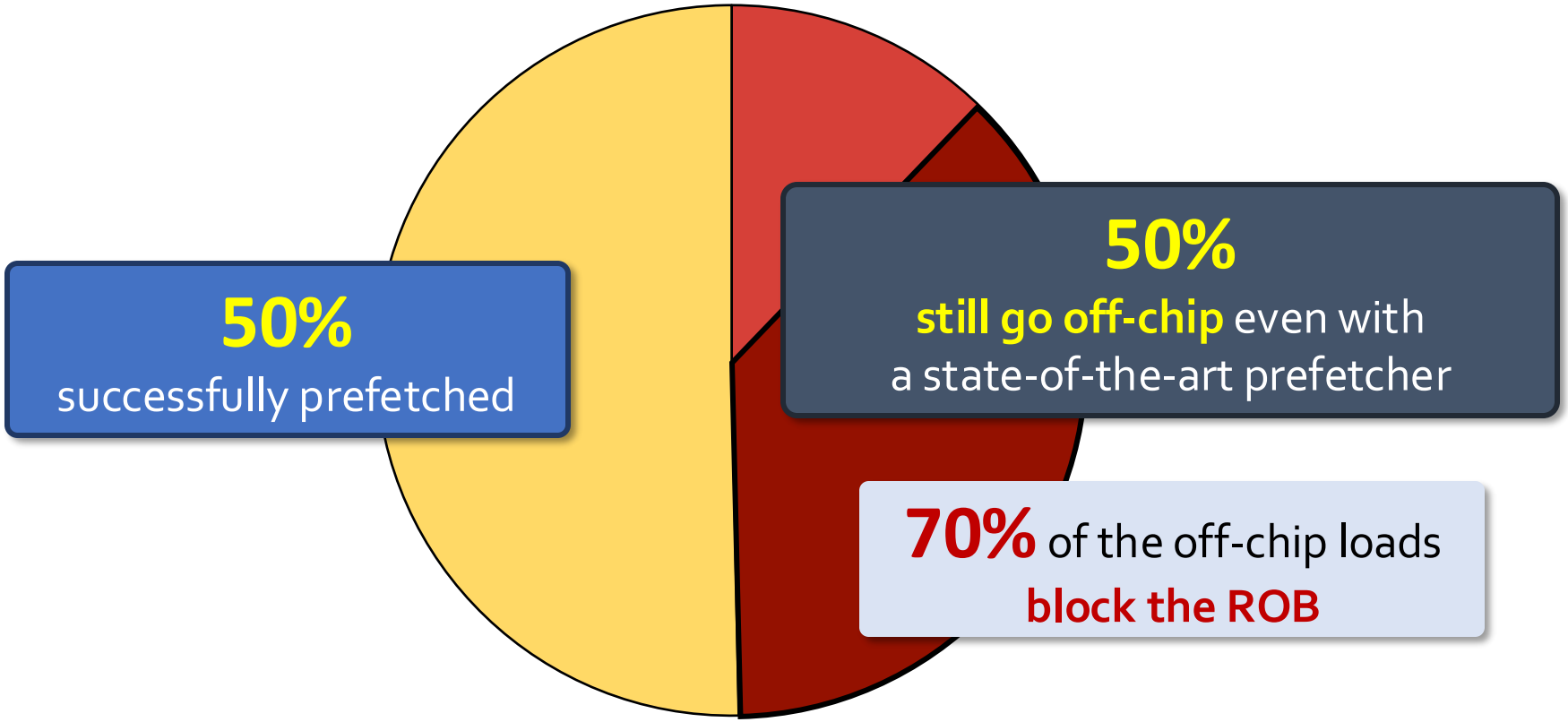
Limit performance

Traditional Solutions

1. Employ sophisticated **prefetchers**
2. **Increase size** of on-chip caches

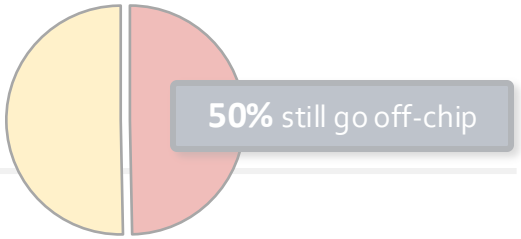
Key Observation 1

Many loads still go off-chip

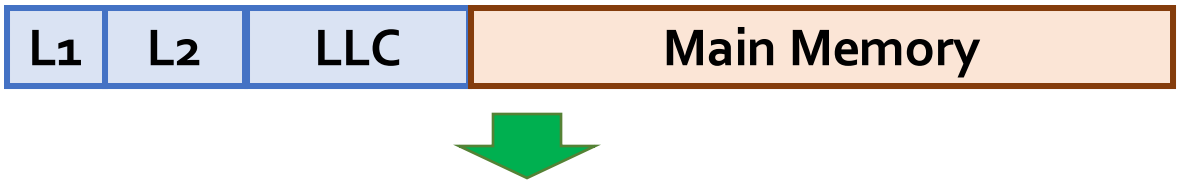


 # off-chip loads without any prefetcher

Key Observation 2

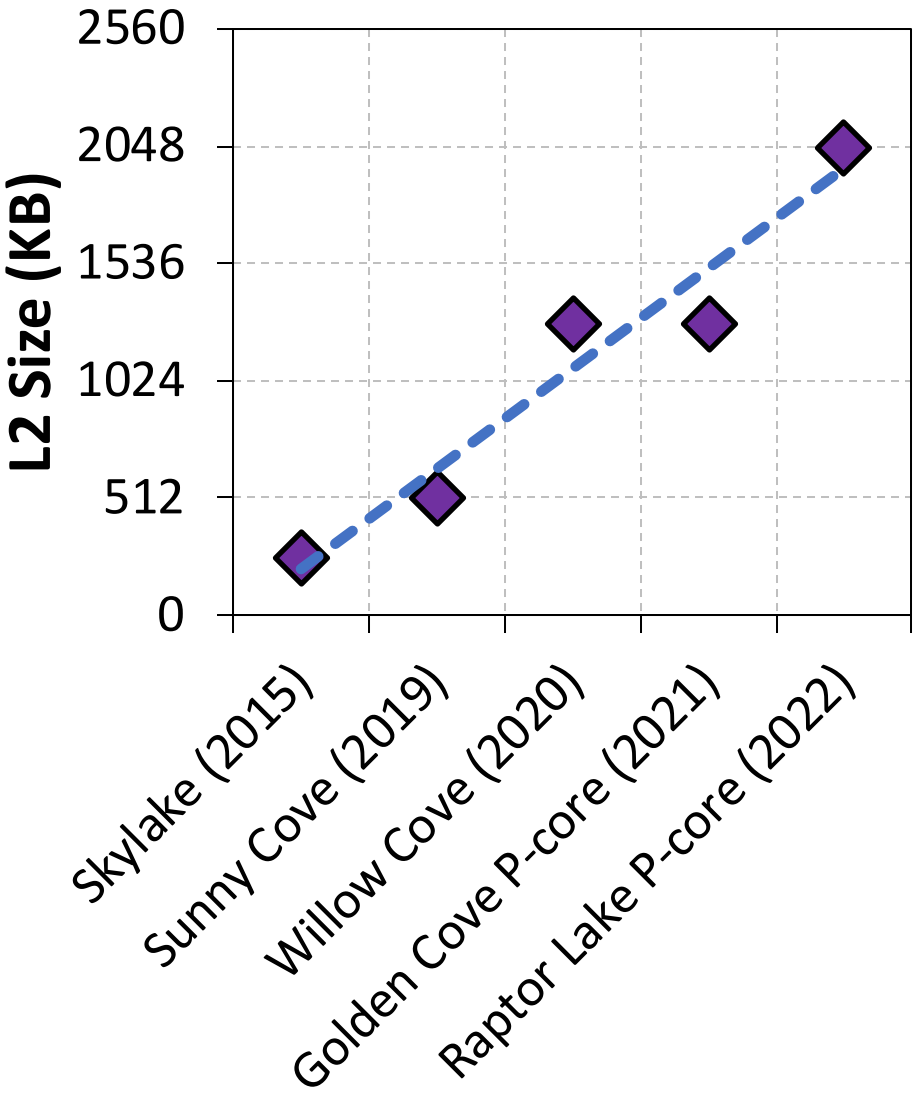


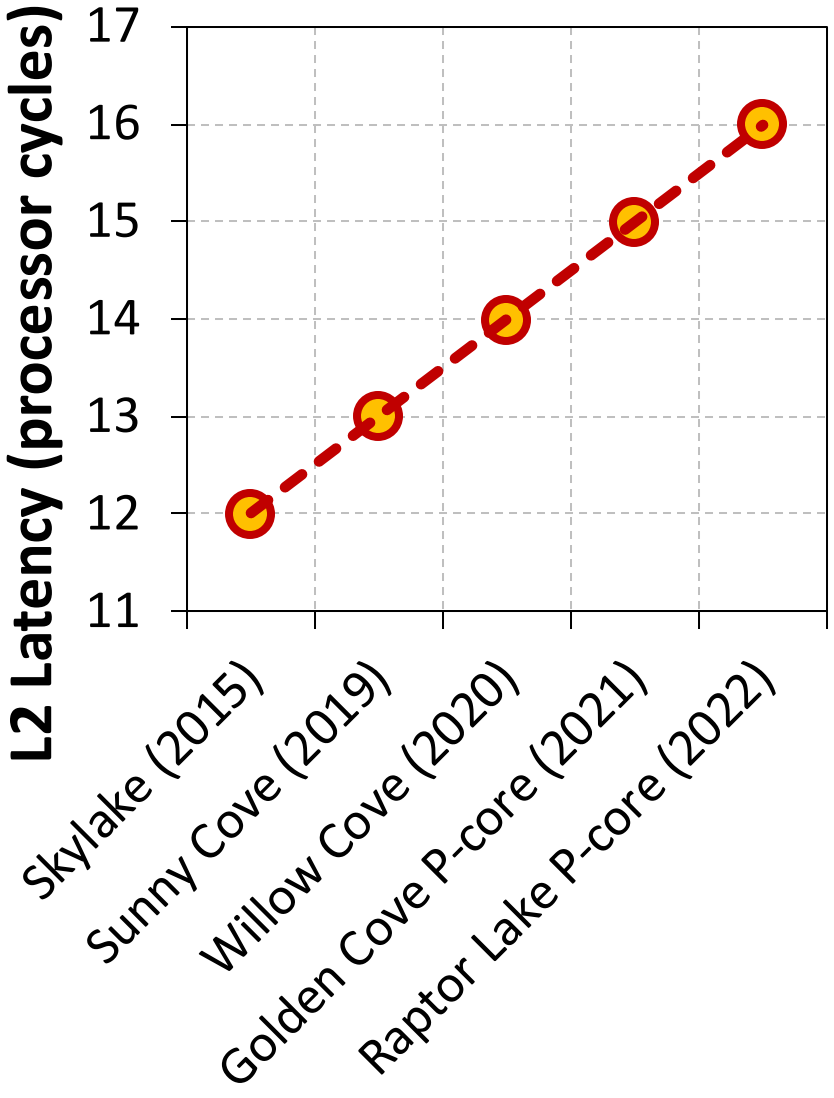
On-chip cache access latency
significantly contributes to off-chip load latency



40% of the **stalls** can be eliminated by **removing on-chip cache access latency from critical path**

Caches are Getting Bigger and Slower…





Our Goal

Improve processor performance
by **removing on-chip cache access latency**
from the **critical path of off-chip loads**

HERMES

Predicts which load requests
are likely to **go off-chip**

Starts **fetching** data **directly** from **main memory**
while concurrently accessing the cache hierarchy

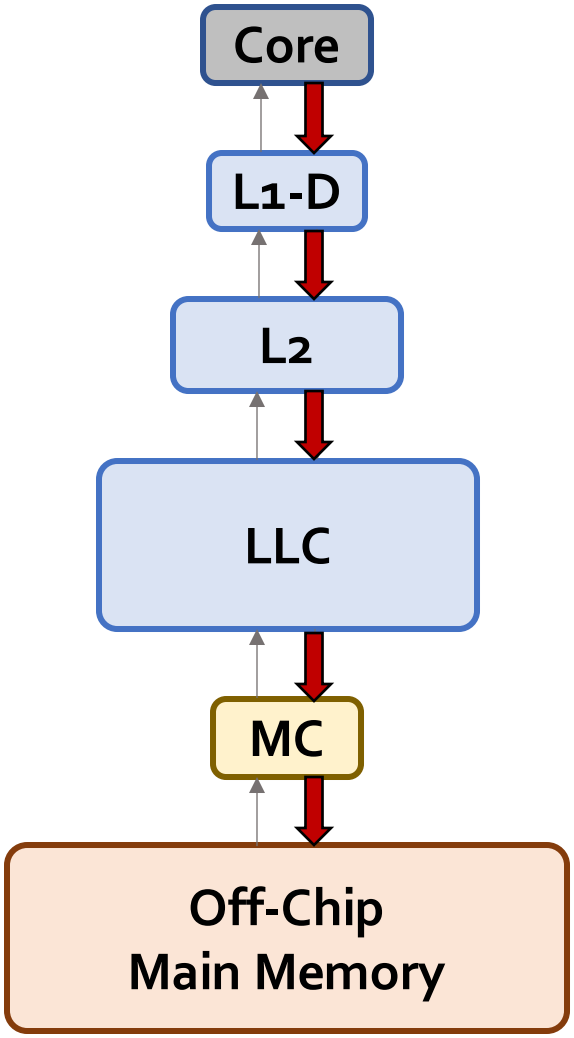
Hermes: Key Contribution

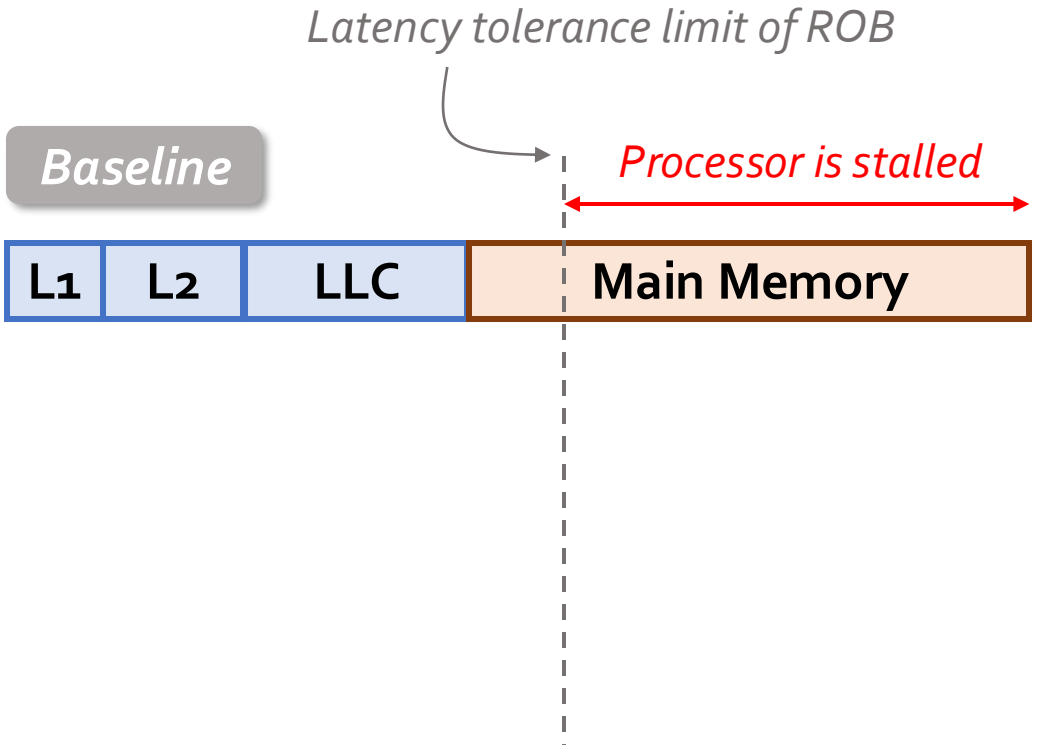
Hermes employs **the first perceptron-based** off-chip load predictor

That predicts which loads are likely to **go off-chip**

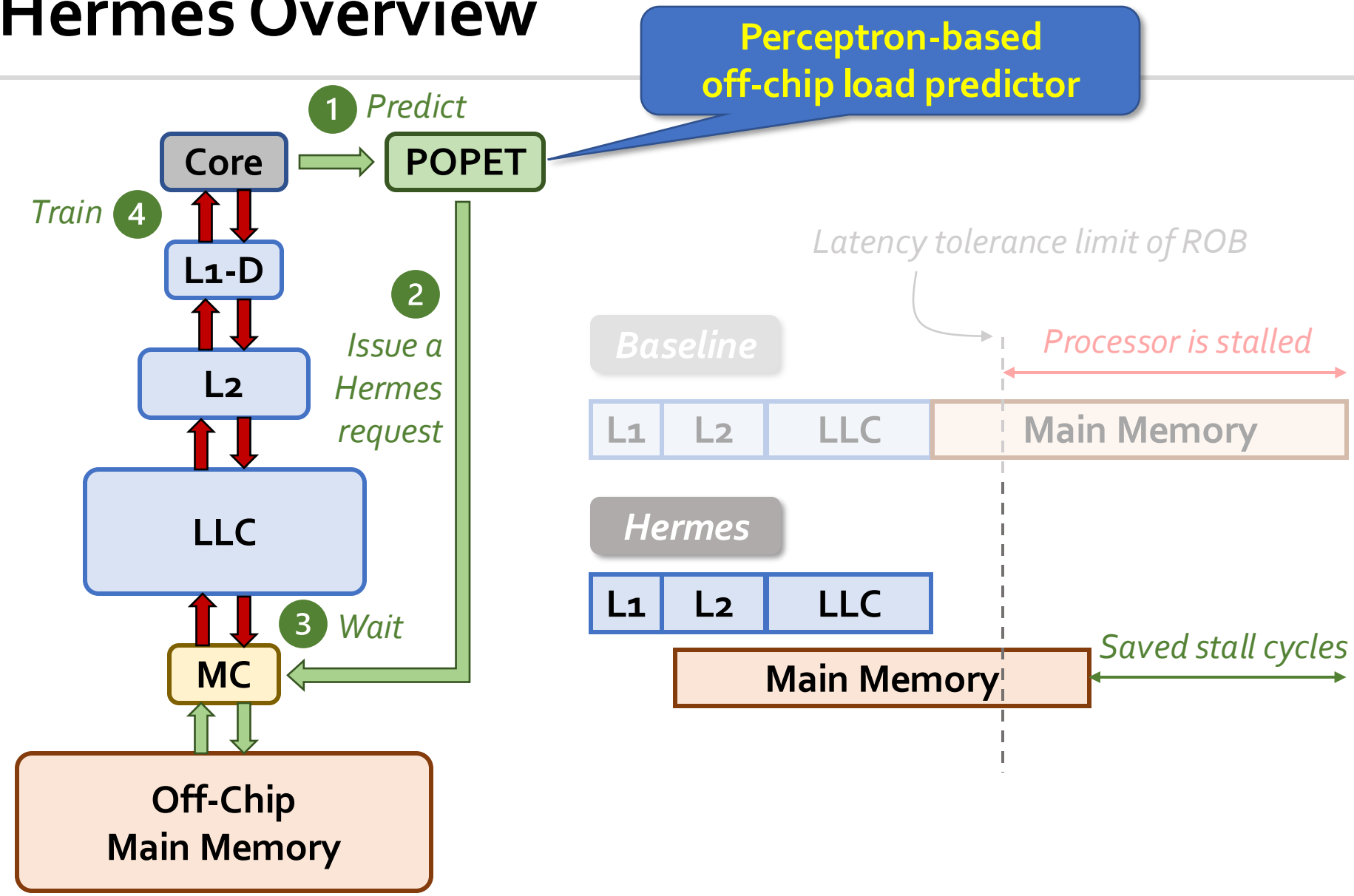
By **learning** from **multiple program context information**

Hermes Overview





Hermes Overview



Designing the Off-Chip Load Predictor

History-based prediction

HMP [Yoaz+, ISCA'99] for the L1-D cache

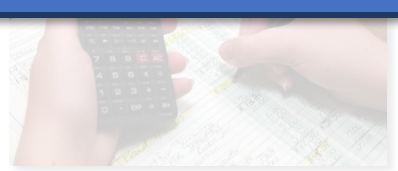
Using branch-predictor-like hybrid predictor:
Global, Gshare, and GSkew

POPET provides
both **higher accuracy** and **higher performance**
than predictors inspired from these previous works

- Metadata size increases with cache hierarchy size

May need to track **all** cache operations

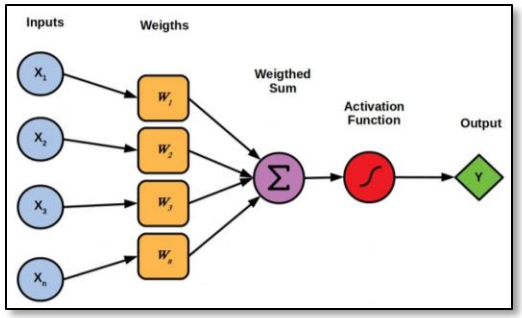
- Gets complex depending on the cache hierarchy configuration (e.g., inclusivity, bypassing,...)



Learning from program behavior

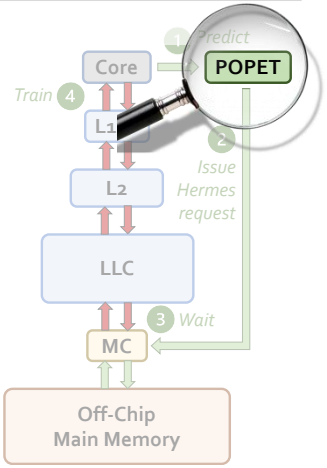
Correlate different program features with off-chip loads

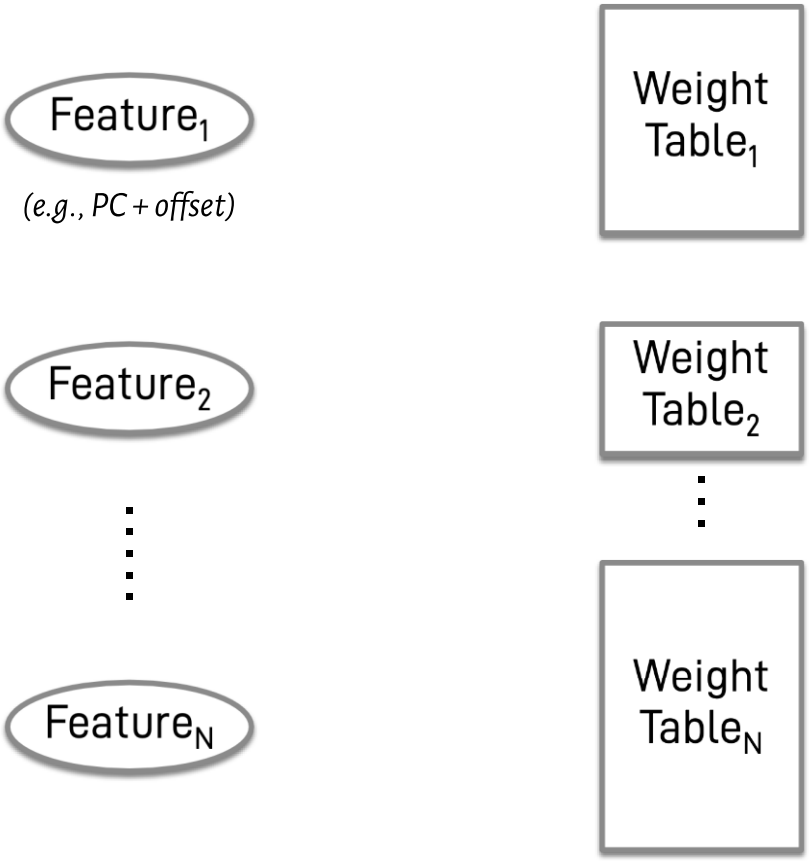
Low storage overhead Low design complexity



POPET: Perceptron-Based Off-Chip Predictor

- Multi-feature hashed perceptron model[1]
  - Each feature has its own *weight table*
    - Stores correlation between feature value and off-chip prediction

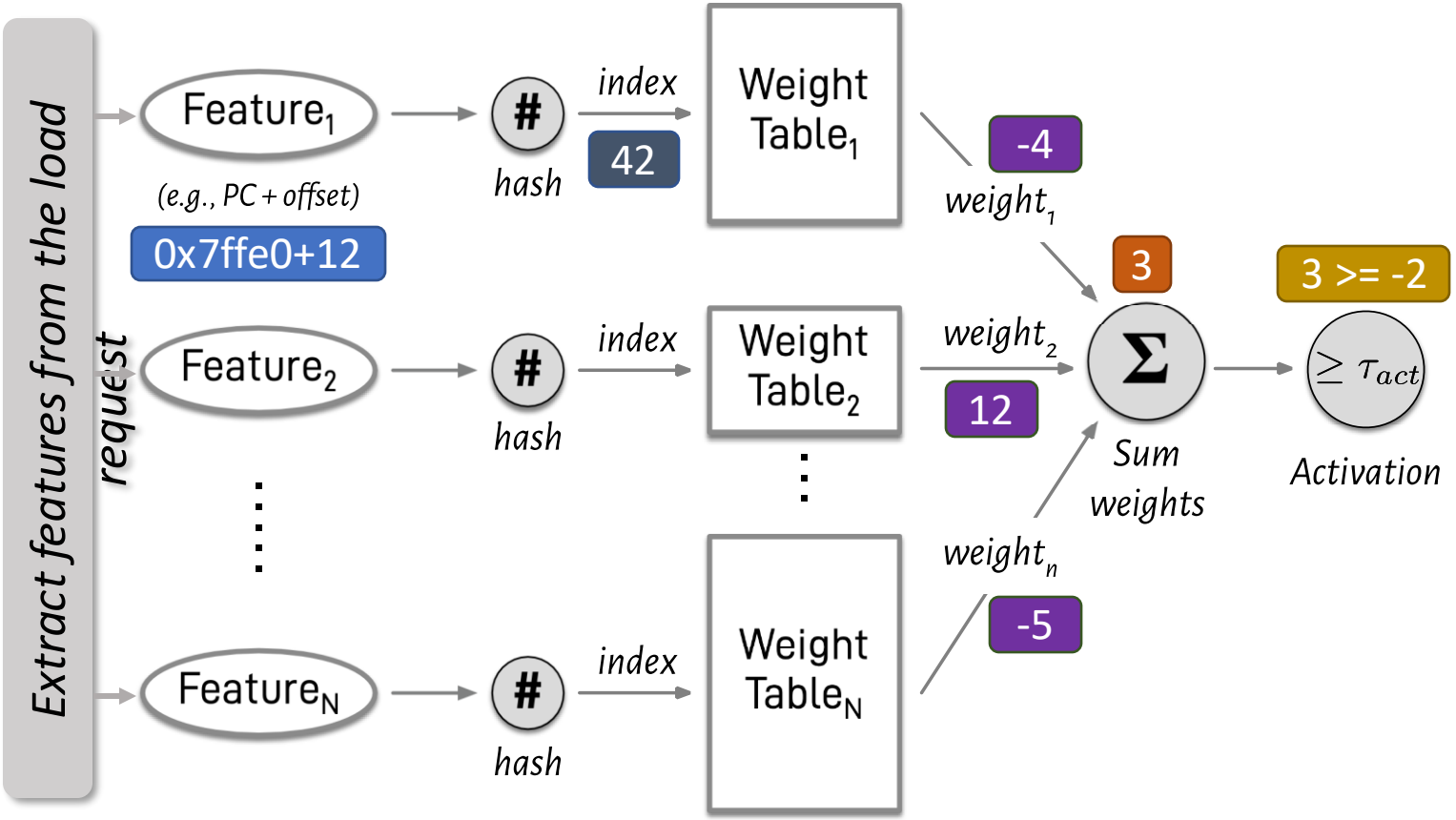


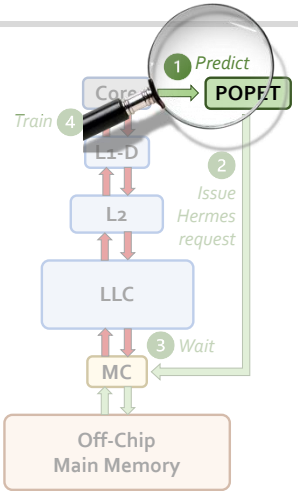


[1] D. Tarjan and K. Skadron, "Merging Path and Gshare Indexing in Perceptron Branch Prediction," TACO, 2005

Predicting using POPET

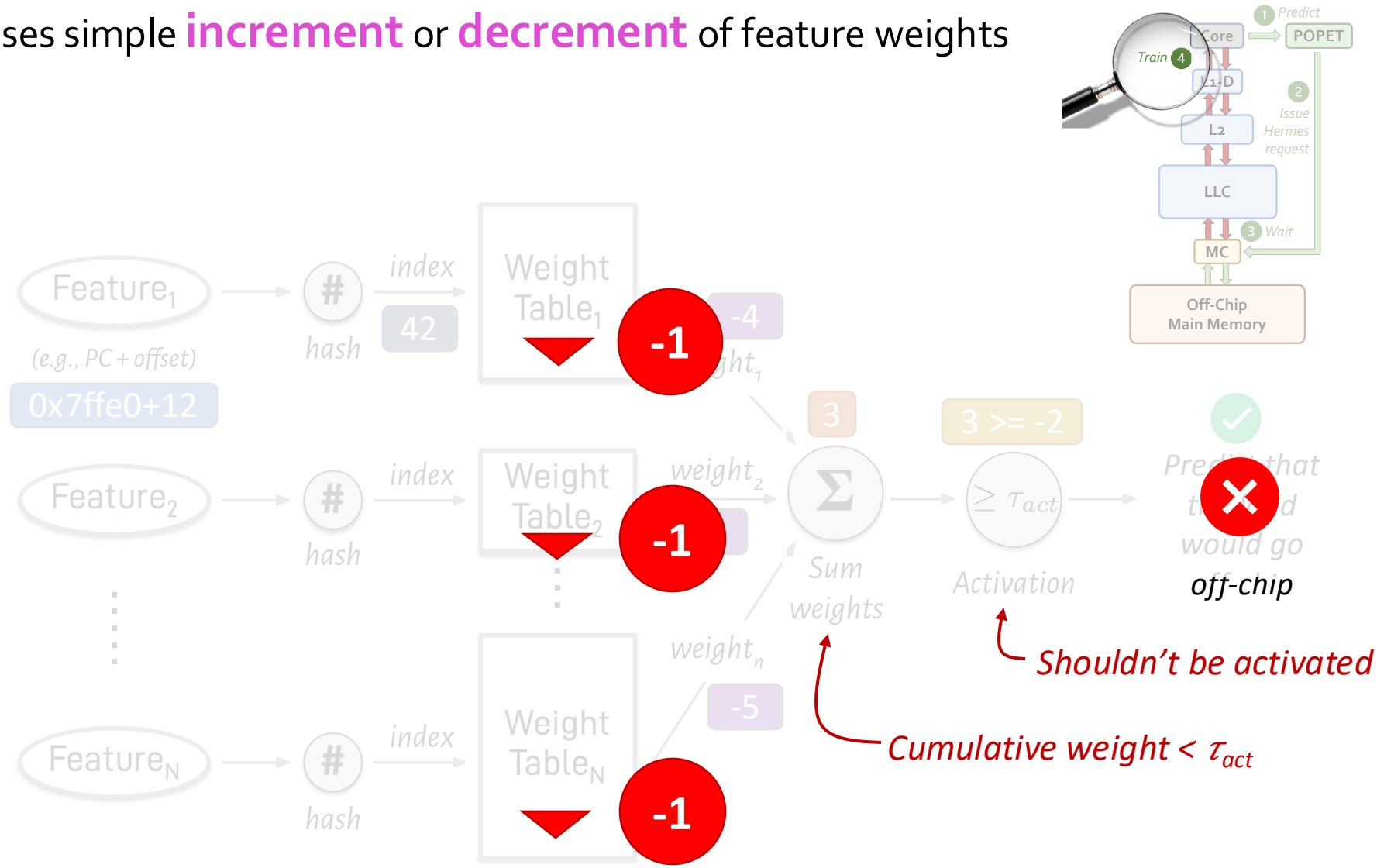
- Uses simple **table lookups**, **addition**, and **comparison**





Training POPET

- Uses simple **increment** or **decrement** of feature weights



Evaluation

Simulation Methodology

- **ChampSim** trace driven simulator
- **110 single-core** memory-intensive traces
  - SPEC CPU 2006 and 2017
  - PARSEC 2.1
  - Ligra
  - Real-world applications
- **220 eight-core** memory-intensive trace mixes

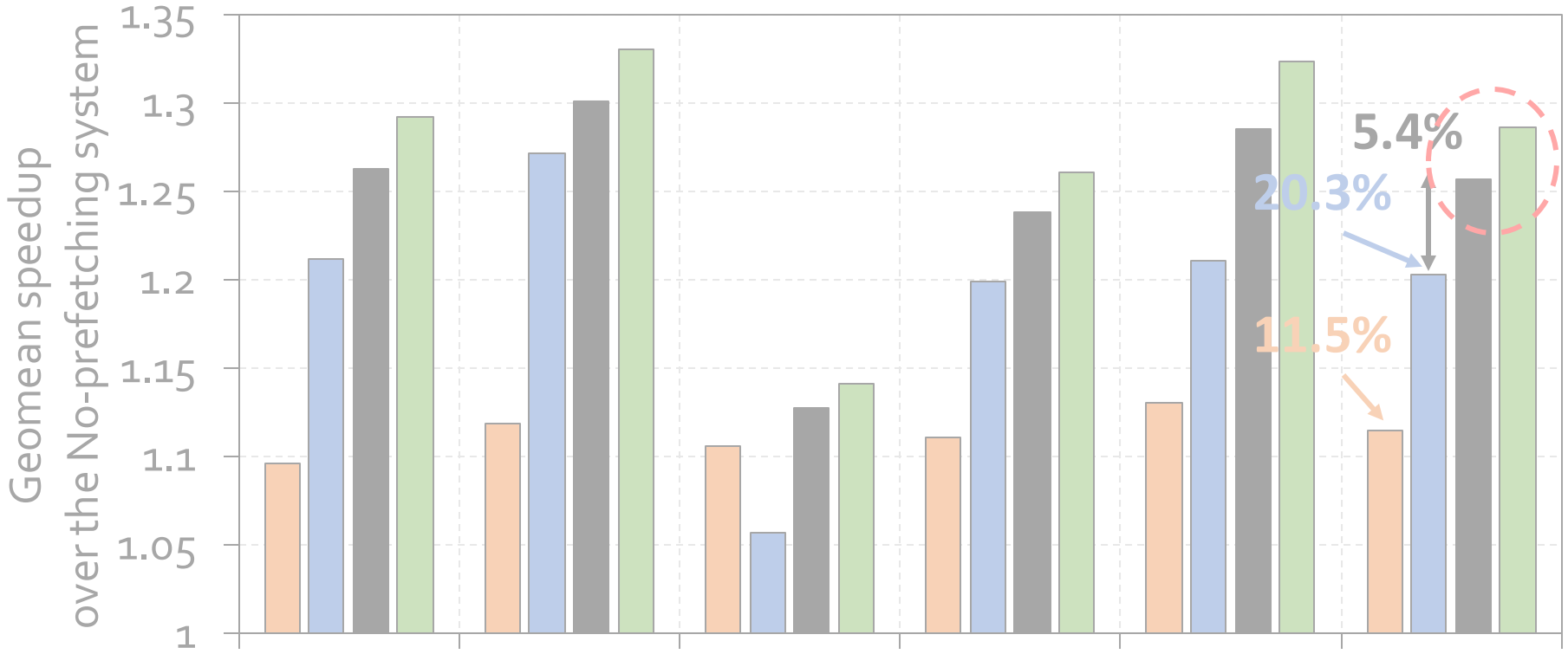
LLC Prefetchers

- Pythia [Bera+, MICRO'21]
- Bingo [Bakshalipour+, HPCA'19]
- MLOP [Shakerinava+, 3rd Prefetching Championship'19]
- SPP + Perceptron filter [Bhatia+, ISCA'20]
- SMS [Somogyi+, ISCA'06]

Off-Chip Predictors

- **History-based**: **HMP** [Yoaz+, ISCA'99]
- **Tracking-based**: Address Tag-Tracking based Predictor (**TTP**)
- **Ideal Off-chip Predictor**

Single-Core Performance Improvement

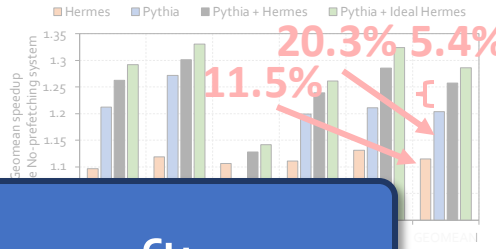


Hermes alone provides nearly

Hermes provides nearly **90%** of performance benefit of **Ideal Hermes** that has an **ideal off-chip load predictor**

with only **1/5th** storage overhead

Increase in Main Memory Requests

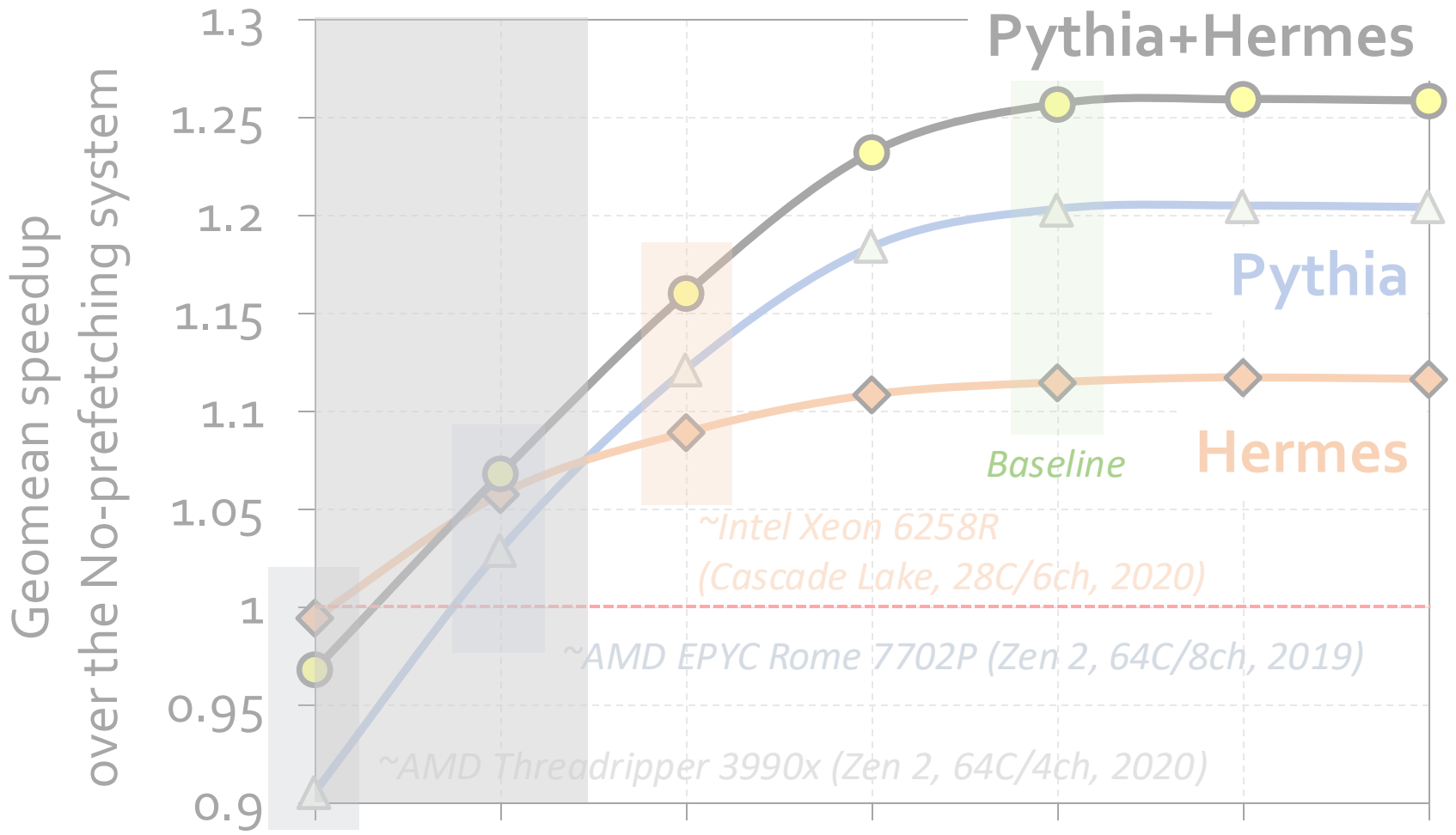


For **every 1% performance** benefit,
increase in main memory requests

Pythia	**2%**
Hermes on top of Pythia	**1%**
Hermes alone	**0.5%**

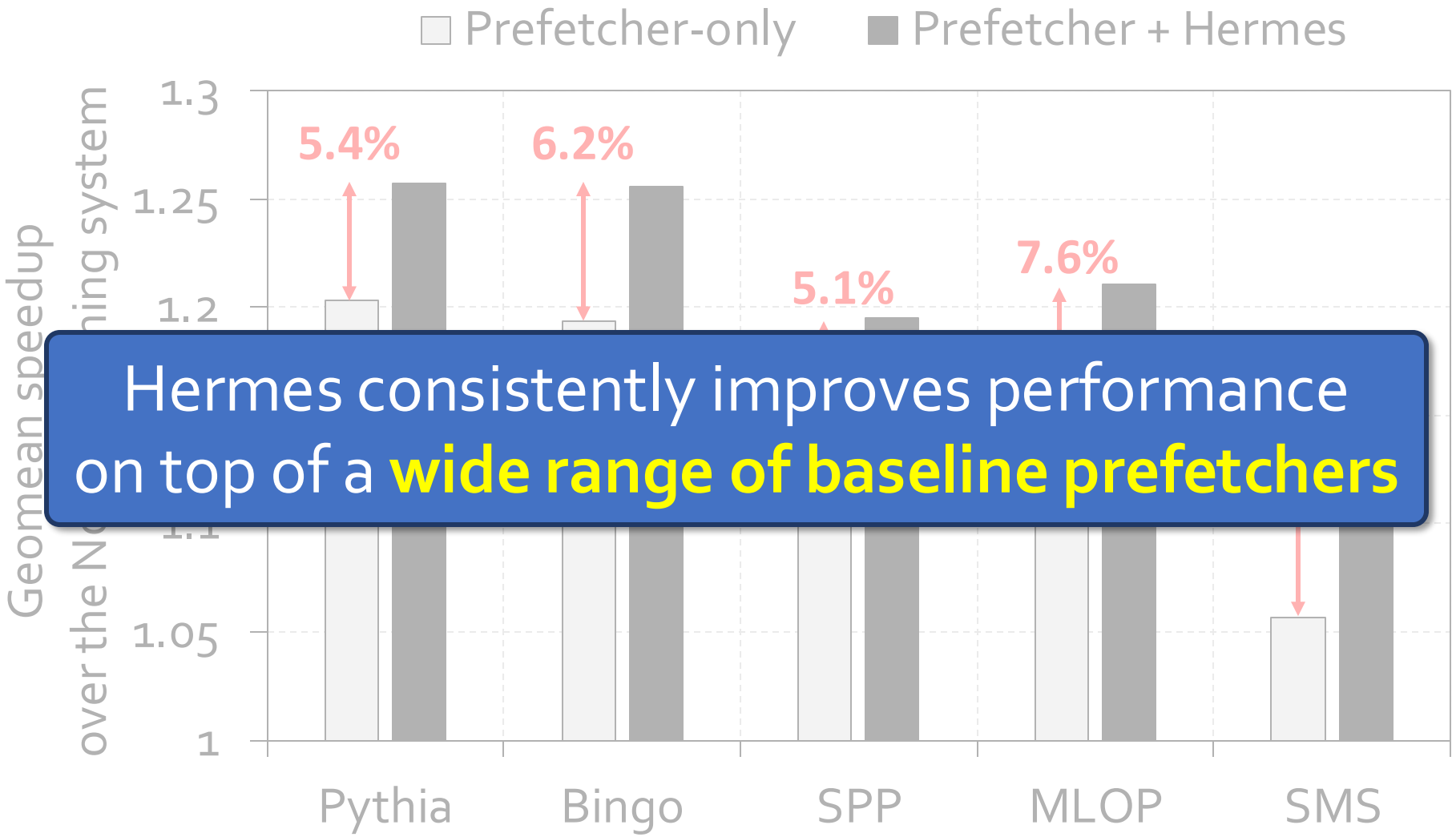
Hermes is more **bandwidth-efficient**
than even an efficient prefetcher like Pythia

Performance with Varying Memory Bandwidth



Hermes+Pythia outperforms Pythia
across all bandwidth configurations

Performance with Varying Baseline Prefetcher



Hermes consistently improves performance on top of a **wide range of baseline prefetchers**

Overhead of Hermes

4 KB storage overhead

1.5% power overhead*

**On top of an Intel Alder Lake-like performance-core [2] configuration*

[2] https://www.anandtech.com/show/16881/a-deep-dive-into-intels-alder-lake-microarchitectures/3

A Lot More in the Hermes Paper

Hermes: Accelerating Long-Latency Load Requests via Perceptron-Based Off-Chip Load Prediction

Rahul Bera[1] Konstantinos Kanellopoulos[1] Shankar Balachandran[2] David Novo[3]
Ataberk Olgun[1] Mohammad Sadrosadati[1] Onur Mutlu[1]

[1]ETH Zürich [2]Intel Processor Architecture Research Lab [3]LIRMM, Univ. Montpellier, CNRS

Long-latency load requests continue to limit the performance of modern high-performance processors. To increase the latency tolerance of a processor, architects have primarily relied on two key techniques: sophisticated data prefetchers and large on-chip caches. In this work, we show that: (1) even a sophisticated state-of-the-art prefetcher can only predict half of the off-chip load requests on average across a wide range of workloads, and (2) due to the increasing size and complexity of on-chip caches, a large fraction of the latency of an off-chip load request is spent accessing the on-chip cache hierarchy to solely determine that it needs to go off-chip.

The goal of this work is to accelerate off-chip load requests by removing the on-chip cache access latency from their critical path. To this end, we propose a new technique called Hermes, whose key idea is to: (1) accurately predict which load requests

off-chip main memory (i.e., an *off-chip load*) often stalls the processor core by blocking the instruction retirement from the reorder buffer (ROB), thus limiting the core's performance [88, 91, 92]. To increase the latency tolerance of a core, computer architects primarily rely on two key techniques. First, they employ increasingly sophisticated hardware prefetchers that can learn complex memory address patterns and fetch data required by future load requests before the core demands them [28, 32, 33, 35, 75]. Second, they significantly scale up the size of the on-chip cache hierarchy with each new generation of processors [10, 11, 16].

Key problem. Despite recent advances in processor core design, we observe two key trends in new processor designs that leave a significant opportunity for performance improvement on the table. First, even a sophisticated state-of-the-art

https://arxiv.org/pdf/2209.00188.pdf

A New Approach to Latency Reduction

Hermes advocates for **off-chip load prediction**, a **different** form of speculation than **load address prediction** employed by prefetchers

Off-chip load prediction can be applied **by itself** or **combined with load address prediction** to provide performance improvement

Hermes: Summary

Hermes employs **the first perceptron-based** off-chip load predictor

High accuracy
(77%)

High coverage
(74%)

Low storage overhead
(4KB/core)

High performance improvement
over best prior baseline
(5.4%)

High performance per bandwidth

Hermes is Open Source

All workload traces

13 prefetchers

- Stride [Fu+, MICRO'92]
- Streamer [Chen and Baer, IEEE TC'95]
- SMS [Somogyi+, ISCA'06]
- AMPM [Ishii+, ICS'09]
- Sandbox [Pugsley+, HPCA'14]
- BOP [Michaud, HPCA'16]
- SPP [Kim+, MICRO'16]
- Bingo [Bakshalipour+, HPCA'19]
- SPP+PPF [Bhatia+, ISCA'19]
- DSPatch [Bera+, MICRO'19]
- MLOP [Shakerinava+, DPC-3'19]
- IPCP [Pakalapati+, ISCA'20]
- Pythia [Bera+, MICRO'21]

9 off-chip predictors

Predictor type	Description
Base	Always NO
Basic	Simple confidence counter-based threshold
Random	Random Hit-miss predictor with a given positive probability
HMP-Local	Hit-miss predictor [Yoaz+, ISCA'99] with local prediction
HMP-GShare	Hit-miss predictor with GShare prediction
HMP-GSkew	Hit-miss predictor with GSkew prediction
HMP-Ensemble	Hit-miss predictor with all three types combined
TTP	Tag-tracking based predictor
Perc	Perceptron-based OCP used in this paper

https://github.com/CMU-SAFARI/Hermes

Easy To Define Your Own Off-Chip Predictor

- Just extend the OffchipPredBase class

```
8   class OffchipPredBase
9   {
10  public:
11      uint32_t cpu;
12      string type;
13      uint64_t seed;
14      uint8_t dram_bw; // current DRAM bandwidth bucket
15
16      OffchipPredBase(uint32_t _cpu, string _type, uint64_t _seed) : cpu(_cpu), type(_type), seed(_seed)
17      {
18          srand(seed);
19          dram_bw = 0;
20      }
21      ~OffchipPredBase() {}
22      void update_dram_bw(uint8_t _dram_bw) { dram_bw = _dram_bw; }
23
24      virtual void print_config();
25      virtual void dump_stats();
26      virtual void reset_stats();
27      virtual void train(ooo_model_instr *arch_instr, uint32_t data_index, LSQ_ENTRY *lq_entry);
28      virtual bool predict(ooo_model_instr *arch_instr, uint32_t data_index, LSQ_ENTRY *lq_entry);
29  };
30
31  #endif /* OFFCHIP_PRED_BASE_H */
32
```

Easy To Define Your Own Off-Chip Predictor

- Define your own `train()` and `predict()` functions

```
19  void OffchipPredBase::train(ooo_model_instr *arch_instr, uint32_t data_index, LSQ_ENTRY *lq_entry)
20  {
21      // nothing to train
22  }
23
24  bool OffchipPredBase::predict(ooo_model_instr *arch_instr, uint32_t data_index, LSQ_ENTRY *lq_entry)
25  {
26      // predict randomly
27      // return (rand() % 2) ? true : false;
28      return false;
29  }
```

- Get statistics like accuracy (stat name *precision*) and coverage (stat name *recall*) out of the box

```
Core_0_offchip_pred_true_pos 2358716
Core_0_offchip_pred_false_pos 276883
Core_0_offchip_pred_false_neg 132145
Core_0_offchip_pred_precision 89.49
Core_0_offchip_pred_recall 94.69
```

Off-Chip Prediction Can Further Enable…

Prioritizing loads that are likely go off-chip in **cache queues** and **on-chip network routing**

Better instruction scheduling of data-dependent instructions

Other ideas to improve **performance** and **fairness** in multi-core system design…

Learning-Based Off-Chip Load Predictors

- Rahul Bera, Konstantinos Kanellopoulos, Shankar Balachandran, David Novo, Ataberk Olgun, Mohammad Sadrosadati, and Onur Mutlu,
  "Hermes: Accelerating Long-Latency Load Requests via Perceptron-Based Off-Chip Load Prediction"
  Proceedings of the 55th International Symposium on Microarchitecture (***MICRO***), Chicago, IL, USA, October 2022.
  [Slides (pptx) (pdf)]
  [Longer Lecture Slides (pptx) (pdf)]
  [Talk Video (12 minutes)]
  [Lecture Video (25 minutes)]
  [arXiv version]
  [Source Code (Officially Artifact Evaluated with All Badges)]
  Officially artifact evaluated as available, reusable and reproducible.
  Best paper award at MICRO 2022.

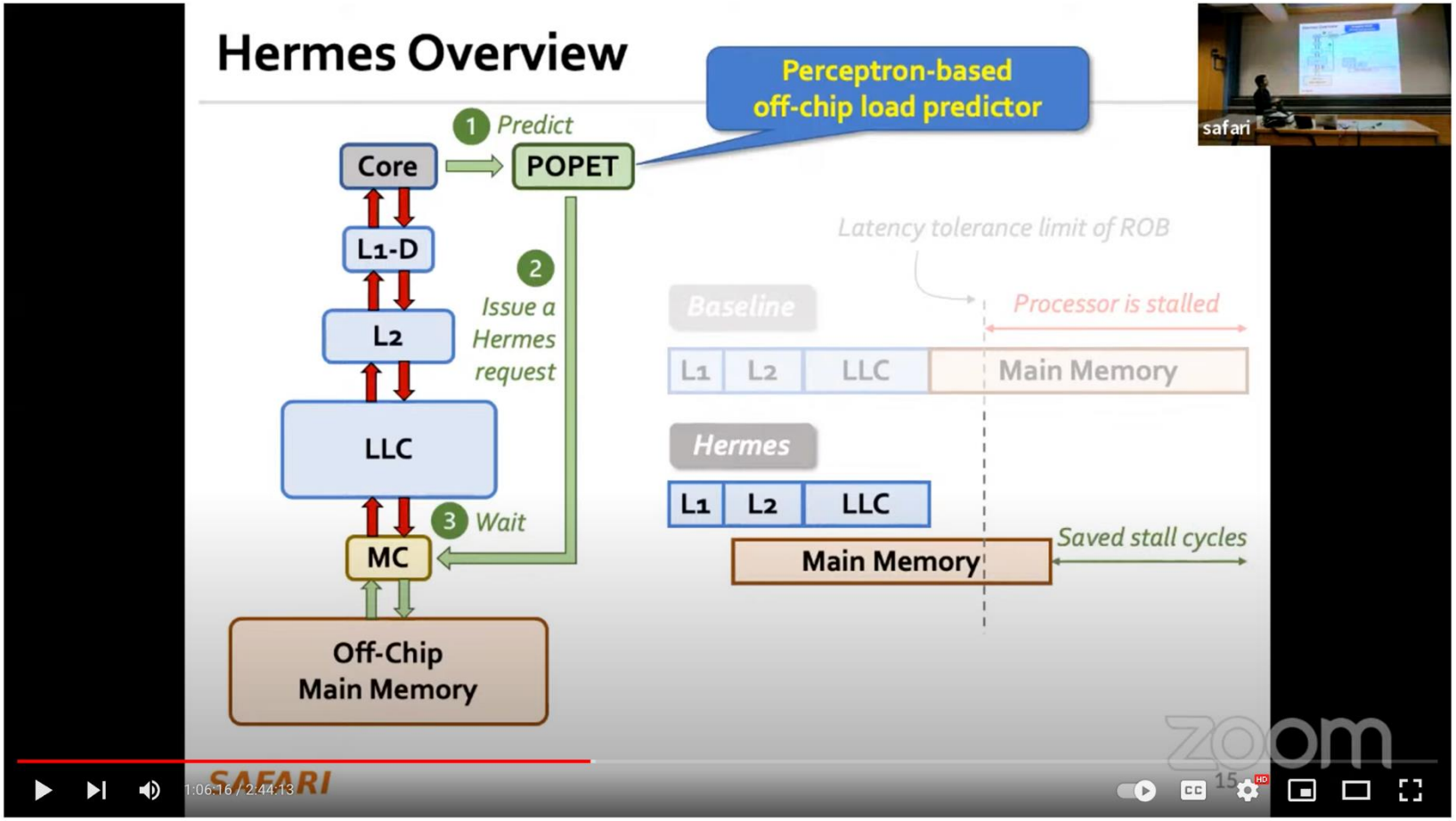
Hermes: Accelerating Long-Latency Load Requests via Perceptron-Based Off-Chip Load Prediction

Rahul Bera[1] Konstantinos Kanellopoulos[1] Shankar Balachandran[2] David Novo[3]
Ataberk Olgun[1] Mohammad Sadrosadati[1] Onur Mutlu[1]

[1]ETH Zürich [2]Intel Processor Architecture Research Lab [3]LIRMM, Univ. Montpellier, CNRS

https://arxiv.org/pdf/2209.00188.pdf

Hermes Talk Video



Computer Architecture - Lecture 18: Cutting-Edge Research in Computer Architecture (Fall 2022)

Onur Mutlu Lectures
32.9K subscribers
Analytics | Edit video
23 | Share | Download | Clip | Save

2.4K views Streamed 5 months ago Livestream - Computer Architecture - ETH Zürich (Fall 2022)
Computer Architecture, ETH Zürich, Fall 2022 (https://safari.ethz.ch/architecture/f...)

https://www.youtube.com/watch?v=PWWBtrL60dQ&t=3609s

HERMES

Accelerating Long-Latency Load Requests via Perceptron-Based Off-Chip Load Prediction

Rahul Bera, Konstantinos Kanellopoulos, Shankar Balachandran, David Novo, Ataberk Olgun, Mohammad Sadrosadati, Onur Mutlu

https://github.com/CMU-SAFARI/Hermes

SAFARI Research Group
safari.ethz.ch

ETH zürich

intel

LIRMM

Reinforcement Learning Based **DRAM Controllers**

DRAM Controller: Functions

- Ensure correct operation of DRAM (refresh and timing)
- Service DRAM requests while obeying timing constraints of DRAM chips
  - Constraints: resource conflicts (bank, bus, channel), minimum write-to-read delays
  - Translate requests to DRAM command sequences
- Buffer and schedule requests for high performance + QoS
  - Reordering, row-buffer, bank, rank, bus management
- Manage power consumption and thermals in DRAM
  - Turn on/off DRAM chips, manage power modes

Why Are DRAM Controllers Difficult to Design?

- Need to obey DRAM timing constraints for correctness
  - There are many (50+) timing constraints in DRAM
  - tWTR: Minimum number of cycles to wait before issuing a read command after a write command is issued
  - tRC: Minimum number of cycles between the issuing of two consecutive activate commands to the same bank
  - …
- Need to keep track of many resources to prevent conflicts
  - Channels, banks, ranks, data bus, address bus, row buffers
- Need to handle DRAM refresh
- Need to manage power consumption
- Need to optimize performance & QoS (in the presence of constraints)
  - Reordering is not simple
  - Fairness and QoS needs complicates the scheduling problem

Many DRAM Timing Constraints

Latency	Symbol	DRAM cycles	Latency	Symbol	DRAM cycles
Precharge	^tRP	11	Activate to read/write	^tRCD	11
Read column address strobe	CL	11	Write column address strobe	CWL	8
Additive	AL	0	Activate to activate	^tRC	39
Activate to precharge	^tRAS	28	Read to precharge	^tRTP	6
Burst length	^tBL	4	Column address strobe to column address strobe	^tCCD	4
Activate to activate (different bank)	^tRRD	6	Four activate windows	^tFAW	24
Write to read	^tWTR	6	Write recovery	^tWR	12

Table 4. DDR3 1600 DRAM timing specifications

- From Lee et al., "DRAM-Aware Last-Level Cache Writeback: Reducing Write-Caused Interference in Memory Systems," HPS Technical Report, April 2010.

More on DRAM Operation

- Kim et al., "A Case for Exploiting Subarray-Level Parallelism (SALP) in DRAM," ISCA 2012.
- Lee et al., "Tiered-Latency DRAM: A Low Latency and Low Cost DRAM Architecture," HPCA 2013.

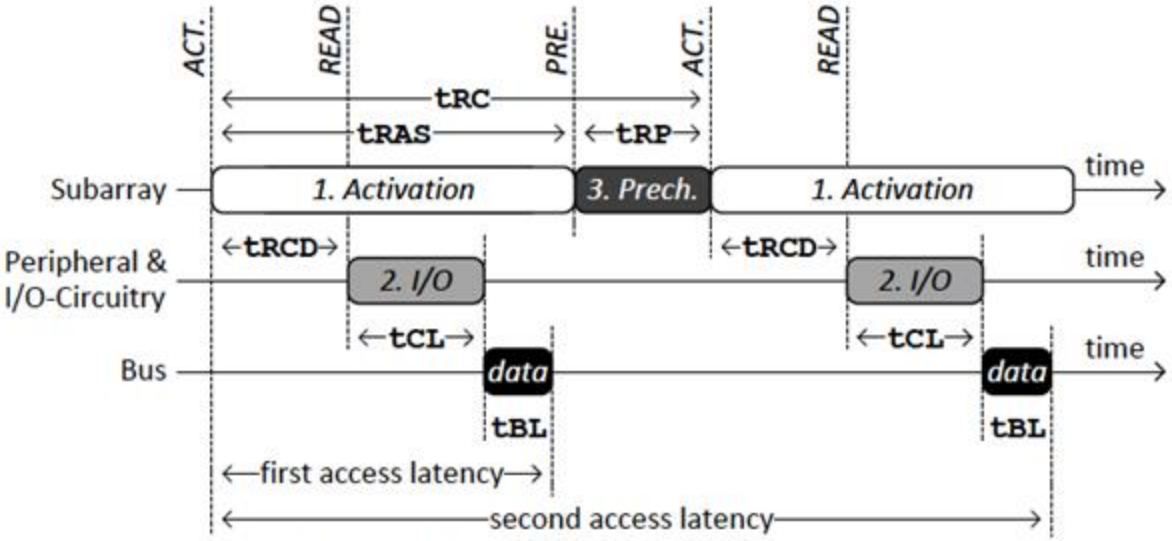


Figure 5. Three Phases of DRAM Access

Table 2. Timing Constraints (DDR3-1066) [43]

Phase	Commands	Name	Value
1	ACT → READ ACT → WRITE	tRCD	15ns
	ACT → PRE	tRAS	37.5ns
2	READ → *data* WRITE → *data*	tCL tCWL	15ns 11.25ns
	data burst	tBL	7.5ns
3	PRE → ACT	tRP	15ns
1 & 3	ACT → ACT	tRC (tRAS+tRP)	52.5ns

DRAM Scheduling Policies (I)

- FCFS (first come first served)
  - Oldest request first

- FR-FCFS (first ready, first come first served)
  1. Row-hit first
  2. Oldest first

  Goal: Maximize row buffer hit rate → maximize DRAM throughput

DRAM Scheduling Policies (II)

- A scheduling policy is a request prioritization order

- Prioritization can be based on
  - Request age
  - Row buffer hit/miss status
  - Request type (prefetch, read, write)
  - Requestor type (load miss or store miss)
  - Request criticality
    - Oldest miss in the core?
    - How many instructions in core are dependent on it?
    - Will it stall the processor?
  - Interference caused to other cores
  - ...

Memory Performance Attacks [USENIX SEC'07]

- Thomas Moscibroda and Onur Mutlu,
  "Memory Performance Attacks: Denial of Memory Service in Multi-Core Systems"
  *Proceedings of the 16th USENIX Security Symposium (**USENIX SECURITY**)*, pages 257-274, Boston, MA, August 2007. Slides (ppt)

STFM [MICRO'07]

- Onur Mutlu and Thomas Moscibroda,
  "Stall-Time Fair Memory Access Scheduling for Chip Multiprocessors"
  *Proceedings of the 40th International Symposium on Microarchitecture (**MICRO**)*, pages 146-158, Chicago, IL, December 2007. [Summary] [Slides (ppt)]

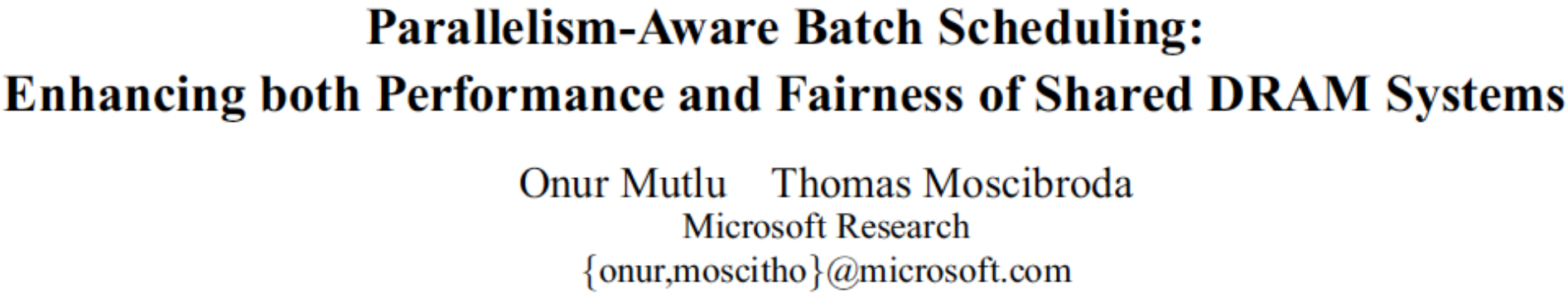
Stall-Time Fair Memory Access Scheduling for Chip Multiprocessors

Onur Mutlu Thomas Moscibroda

Microsoft Research
{onur,moscitho}@microsoft.com

PAR-BS [ISCA'08]

- Onur Mutlu and Thomas Moscibroda,
  "Parallelism-Aware Batch Scheduling: Enhancing both Performance and Fairness of Shared DRAM Systems"
  *Proceedings of the 35th International Symposium on Computer Architecture (**ISCA**)*, pages 63-74, Beijing, China, June 2008.
  [Summary] [Slides (ppt)]



On PAR-BS

- Variants implemented in Samsung SoC memory controllers

Effective platform level approach and DRAM accesses are crucial to system performance. This paper touches this topics and suggest a superior approach to current known techniques.

Review from ISCA 2008

ATLAS Memory Scheduler [HPCA'10]

- Yoongu Kim, Dongsu Han, Onur Mutlu, and Mor Harchol-Balter, **"ATLAS: A Scalable and High-Performance Scheduling Algorithm for Multiple Memory Controllers"** *Proceedings of the 16th International Symposium on High-Performance Computer Architecture (**HPCA**)*, Bangalore, India, January 2010. Slides (pptx)

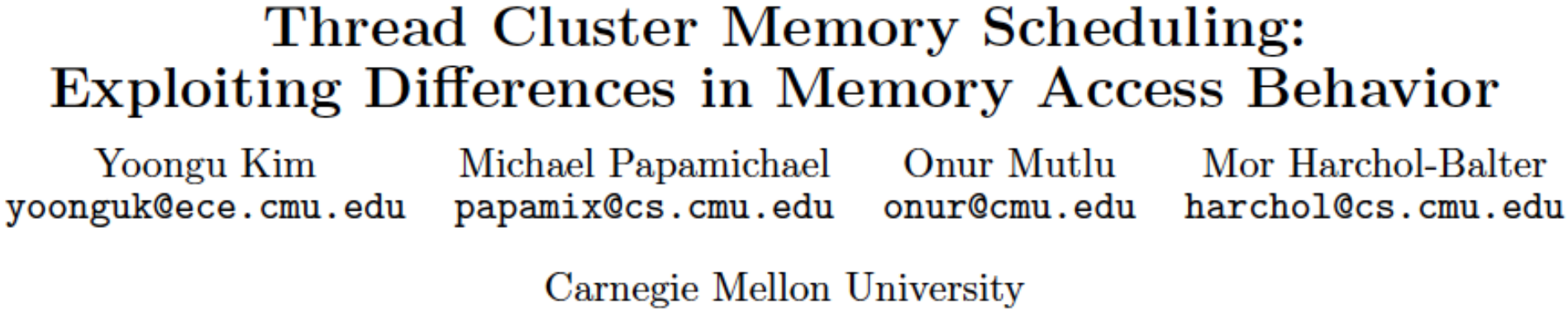
ATLAS: A Scalable and High-Performance Scheduling Algorithm for Multiple Memory Controllers

Yoongu Kim Dongsu Han Onur Mutlu Mor Harchol-Balter

Carnegie Mellon University

Thread Cluster Memory Scheduling [MICRO'10]

- Yoongu Kim, Michael Papamichael, Onur Mutlu, and Mor Harchol-Balter,
  "Thread Cluster Memory Scheduling: Exploiting Differences in Memory Access Behavior"
  *Proceedings of the 43rd International Symposium on Microarchitecture (**MICRO**)*, pages 65-76, Atlanta, GA, December 2010. Slides (pptx) (pdf)



Thread Cluster Memory Scheduling:
Exploiting Differences in Memory Access Behavior

Yoongu Kim yoonguk@ece.cmu.edu
Michael Papamichael papamix@cs.cmu.edu
Onur Mutlu onur@cmu.edu
Mor Harchol-Balter harchol@cs.cmu.edu

Carnegie Mellon University

BLISS [ICCD'14, TPDS'16]

- Lavanya Subramanian, Donghyuk Lee, Vivek Seshadri, Harsha Rastogi, and Onur Mutlu,
  "The Blacklisting Memory Scheduler: Achieving High Performance and Fairness at Low Cost"
  *Proceedings of the 32nd IEEE International Conference on Computer Design (**ICCD**)*, Seoul, South Korea, October 2014.
  [Slides (pptx) (pdf)]

The Blacklisting Memory Scheduler:
Achieving High Performance and Fairness at Low Cost

Lavanya Subramanian, Donghyuk Lee, Vivek Seshadri, Harsha Rastogi, Onur Mutlu
Carnegie Mellon University
{lsubrama,donghyu1,visesh,harshar,onur}@cmu.edu

Staged Memory Scheduling: CPU-GPU [ISCA'12]

- Rachata Ausavarungnirun, Kevin Chang, Lavanya Subramanian, Gabriel Loh, and Onur Mutlu,
  "Staged Memory Scheduling: Achieving High Performance and Scalability in Heterogeneous Systems"
  *Proceedings of the 39th International Symposium on Computer Architecture (**ISCA**)*, Portland, OR, June 2012. Slides (pptx)

Staged Memory Scheduling: Achieving High Performance and Scalability in Heterogeneous Systems

Rachata Ausavarungnirun[†] Kevin Kai-Wei Chang[†] Lavanya Subramanian[†] Gabriel H. Loh[‡] Onur Mutlu[†]

[†]Carnegie Mellon University
{rachata,kevincha,lsubrama,onur}@cmu.edu

[‡]Advanced Micro Devices, Inc.
gabe.loh@amd.com

DASH: Heterogeneous Systems [TACO'16]

- Hiroyuki Usui, Lavanya Subramanian, Kevin Kai-Wei Chang, and Onur Mutlu,
  "DASH: Deadline-Aware High-Performance Memory Scheduler for Heterogeneous Systems with Hardware Accelerators"
  ACM Transactions on Architecture and Code Optimization (***TACO***),
  Vol. 12, January 2016.
  Presented at the 11th HiPEAC Conference, Prague, Czech Republic, January 2016.
  [Slides (pptx) (pdf)]
  [Source Code]

DASH: Deadline-Aware High-Performance Memory Scheduler for Heterogeneous Systems with Hardware Accelerators

HIROYUKI USUI, LAVANYA SUBRAMANIAN, KEVIN KAI-WEI CHANG, and ONUR MUTLU, Carnegie Mellon University

MISE: Predictable Performance [HPCA'13]

- Lavanya Subramanian, Vivek Seshadri, Yoongu Kim, Ben Jaiyen, and Onur Mutlu,
  "MISE: Providing Performance Predictability and Improving Fairness in Shared Main Memory Systems"
  *Proceedings of the 19th International Symposium on High-Performance Computer Architecture (**HPCA**)*, Shenzhen, China, February 2013. Slides (pptx)

MISE: Providing Performance Predictability and Improving Fairness in Shared Main Memory Systems

Lavanya Subramanian Vivek Seshadri Yoongu Kim Ben Jaiyen Onur Mutlu
Carnegie Mellon University

ASM: Predictable Performance [MICRO'15]

- Lavanya Subramanian, Vivek Seshadri, Arnab Ghosh, Samira Khan, and Onur Mutlu,
  "The Application Slowdown Model: Quantifying and Controlling the Impact of Inter-Application Interference at Shared Caches and Main Memory"
  *Proceedings of the 48th International Symposium on Microarchitecture (**MICRO**)*, Waikiki, Hawaii, USA, December 2015.
  [Slides (pptx) (pdf)] [Lightning Session Slides (pptx) (pdf)] [Poster (pptx) (pdf)]
  [Source Code]

The Application Slowdown Model: Quantifying and Controlling the Impact of Inter-Application Interference at Shared Caches and Main Memory

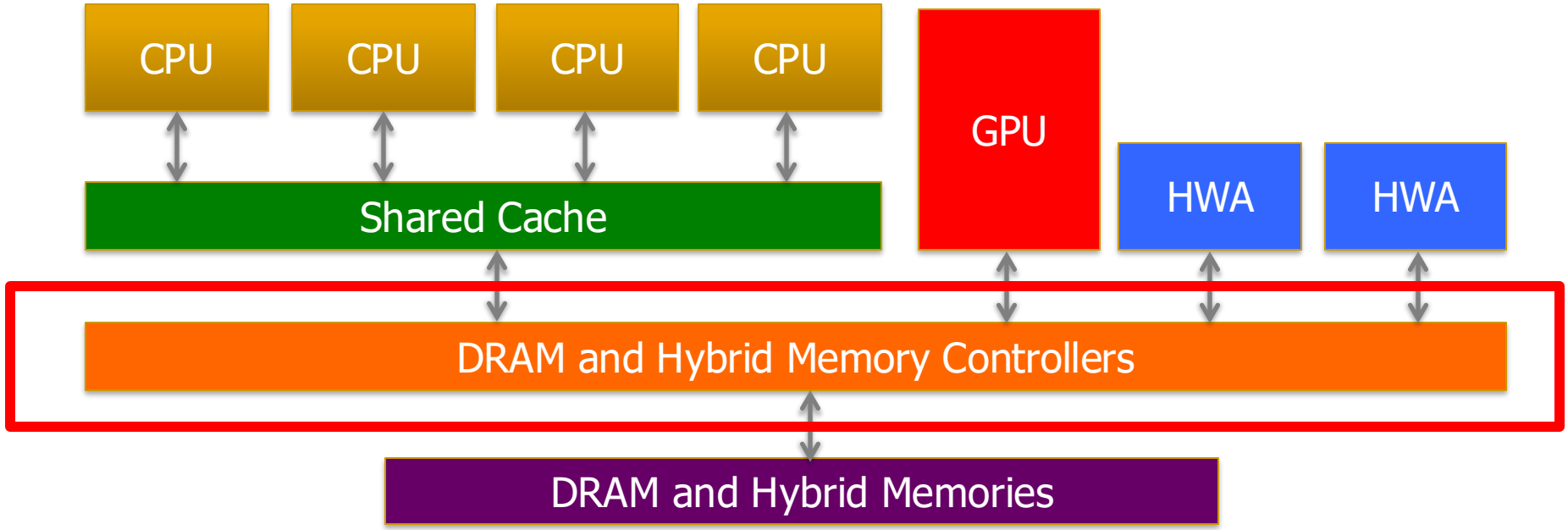
Lavanya Subramanian*§ Vivek Seshadri* Arnab Ghosh*†
Samira Khan*‡ Onur Mutlu*

*Carnegie Mellon University §Intel Labs †IIT Kanpur ‡University of Virginia

Memory Controllers are critical to research

They will become even more important

Memory Control is Getting More Complex



- Heterogeneous agents: CPUs, GPUs, and HWAs
- Main memory interference between CPUs, GPUs, HWAs

Many goals, many constraints, many metrics …

Reality and Dream

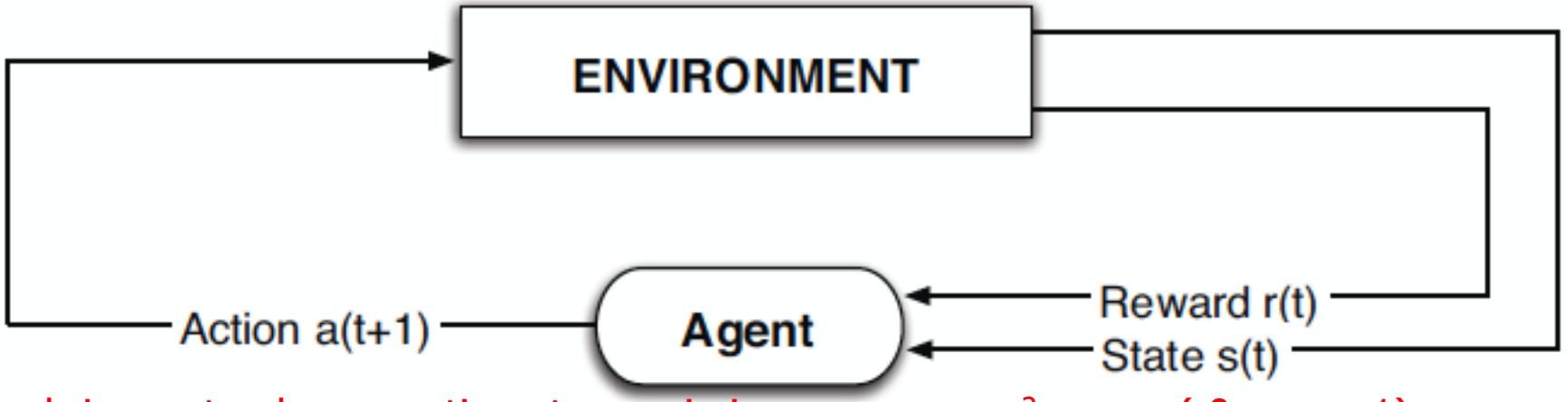
- Reality: It is difficult to design a policy that maximizes performance, QoS, energy-efficiency, …
  - Too many things to think about
  - Continuously changing workload and system behavior

- Dream: Wouldn't it be nice if the DRAM controller automatically found a good scheduling policy on its own?

Self-Optimizing DRAM Controllers

- Problem: DRAM controllers are difficult to design
  - It is difficult for human designers to design a policy that can adapt itself very well to different workloads and different system conditions
- Idea: A memory controller that adapts its scheduling policy to workload behavior and system conditions using machine learning.
- Observation: Reinforcement learning maps nicely to memory control.
- Design: Memory controller is a reinforcement learning agent
  - It dynamically and continuously learns and employs the best scheduling policy to maximize long-term performance.

Self-Optimizing DRAM Controllers

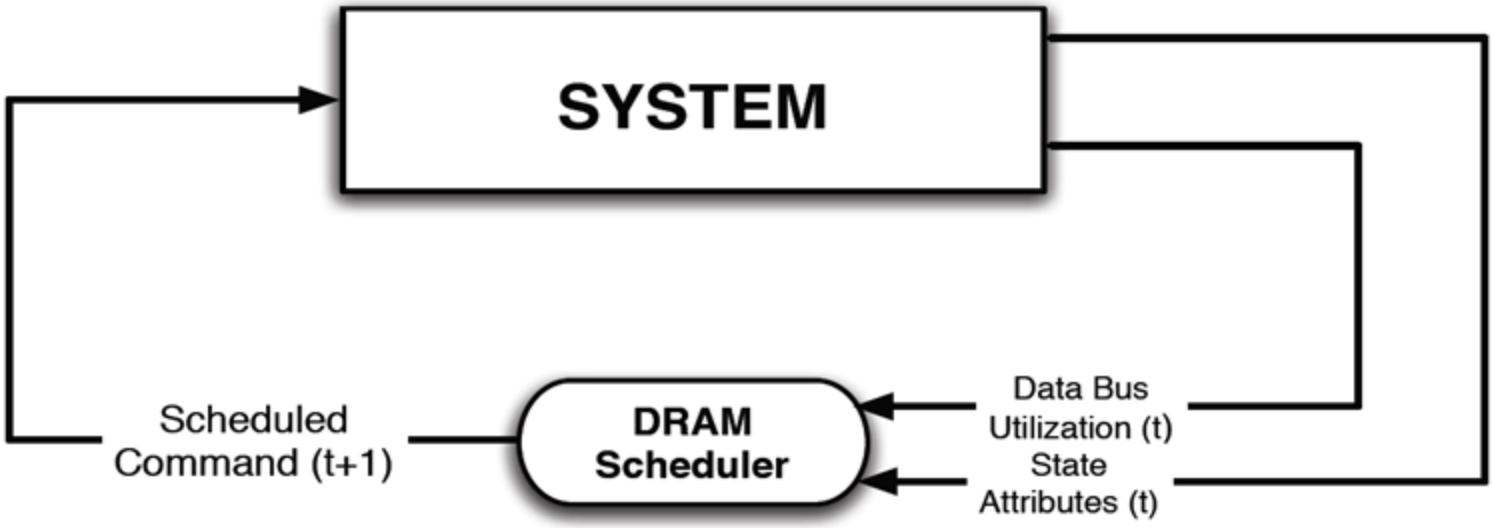


Goal: Learn to choose actions to maximize $r_0 + \gamma r_1 + \gamma^2 r_2 + \ldots$ ($0 \leq \gamma < 1$)

Figure 2: (a) Intelligent agent based on reinforcement learning principles;

Self-Optimizing DRAM Controllers

- Dynamically adapt the memory scheduling policy via interaction with the system at runtime
  - Associate system states and actions (commands) with long term reward values: each action at a given state leads to a learned reward
  - Schedule command with highest estimated long-term reward value in each state
  - Continuously update reward values for <state, action> pairs based on feedback from system



Self-Optimizing DRAM Controllers

- Engin Ipek, Onur Mutlu, José F. Martínez, and Rich Caruana, **"Self Optimizing Memory Controllers: A Reinforcement Learning Approach"**
  *Proceedings of the 35th International Symposium on Computer Architecture (**ISCA**)*, pages 39-50, Beijing, China, June 2008.

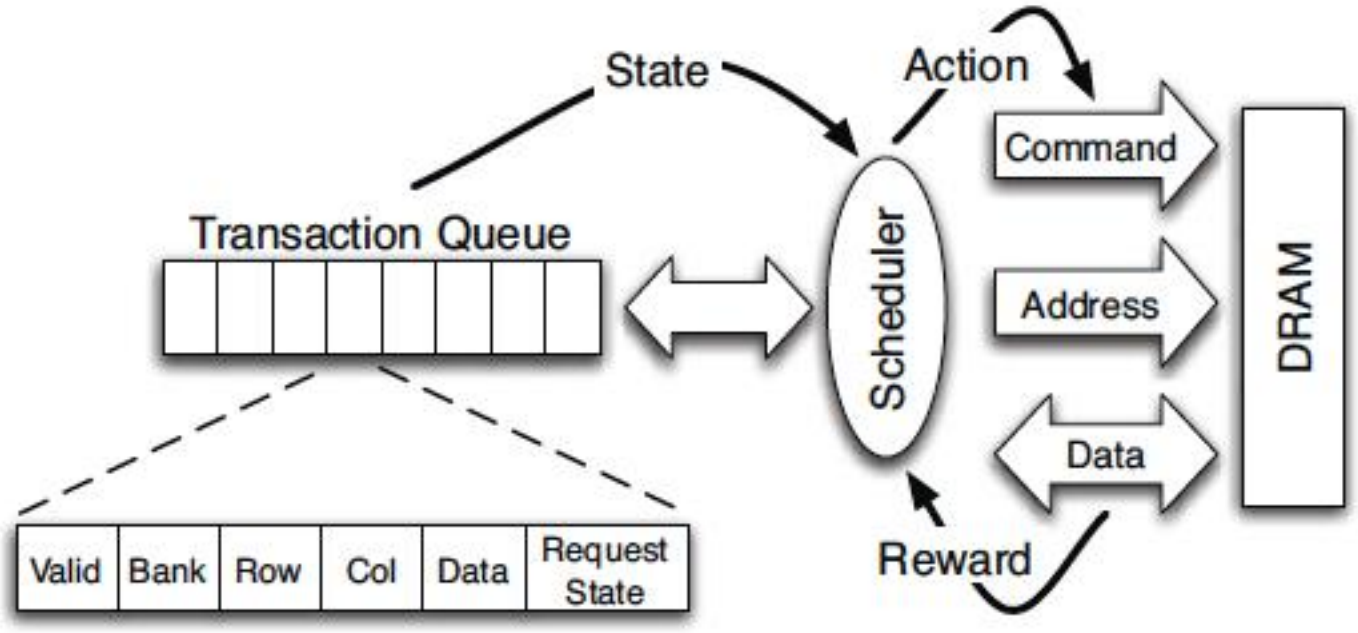


Figure 4: High-level overview of an RL-based scheduler.

States, Actions, Rewards

- Reward function
  - +1 for scheduling Read and Write commands
  - 0 at all other times

  Goal is to maximize long-term data bus utilization
- State attributes
  - Number of reads, writes, and load misses in transaction queue
  - Number of pending writes and ROB heads waiting for referenced row
  - Request's relative ROB order
- Actions
  - Activate
  - Write
  - Read - load miss
  - Read - store miss
  - Precharge - pending
  - Precharge - preemptive
  - NOP

Performance Results

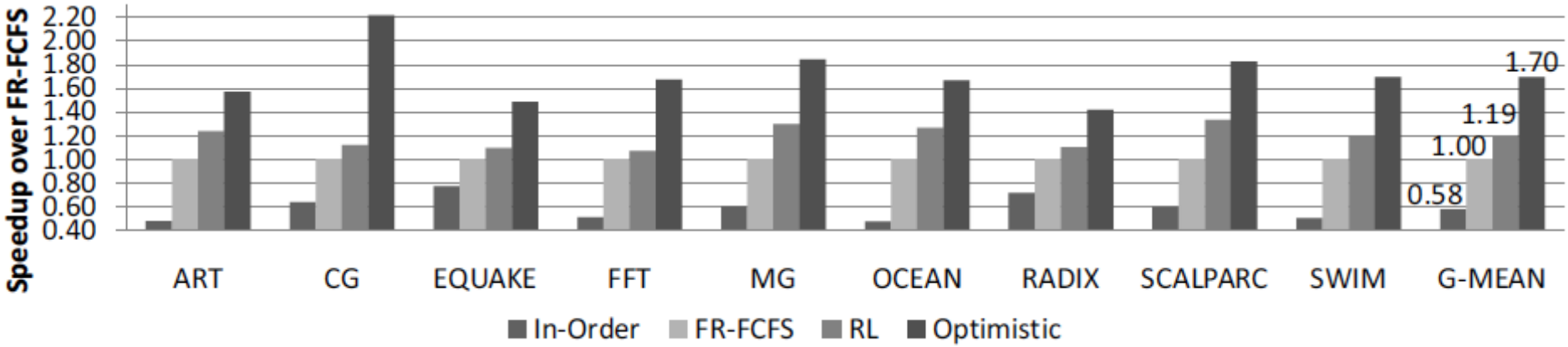


Figure 7: Performance comparison of in-order, FR-FCFS, RL-based, and optimistic memory controllers

Large, robust performance improvements over many human-designed policies

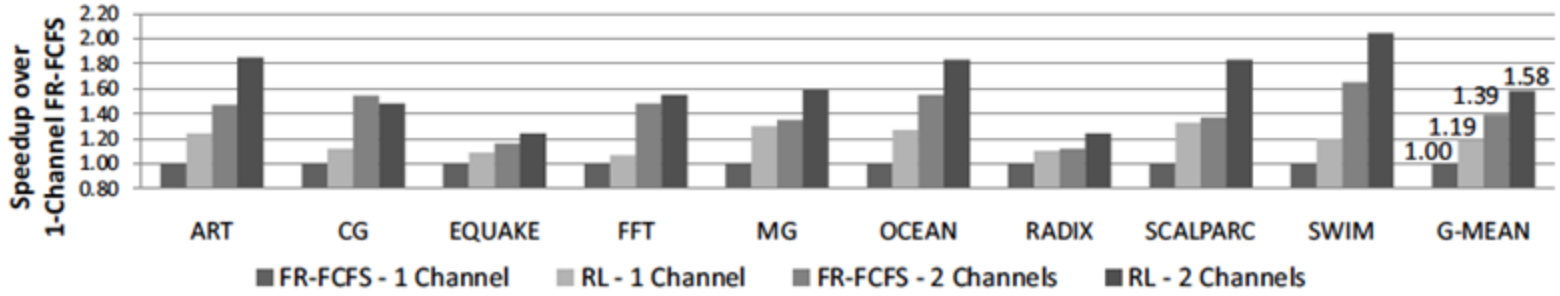


Figure 15: Performance comparison of FR-FCFS and RL-based memory controllers on systems with 6.4GB/s and 12.8GB/s peak DRAM bandwidth

Self Optimizing DRAM Controllers

+ Continuous learning in the presence of changing environment

+ Reduced designer burden in finding a good scheduling policy.
Designer specifies:

1) What system variables might be useful
2) What target to optimize, but not how to optimize it

-- How to specify different objectives? (e.g., fairness, QoS, …)

-- Hardware complexity?

-- Design **mindset** and flow

More on Self-Optimizing DRAM Controllers (I)

- Engin Ipek, Onur Mutlu, José F. Martínez, and Rich Caruana, **"Self Optimizing Memory Controllers: A Reinforcement Learning Approach"**
  *Proceedings of the 35th International Symposium on Computer Architecture (**ISCA**)*, pages 39-50, Beijing, China, June 2008.

Self-Optimizing Memory Controllers: A Reinforcement Learning Approach

Engin İpek[1,2] **Onur Mutlu**[2] **José F. Martínez**[1] **Rich Caruana**[1]

[1]Cornell University, Ithaca, NY 14850 USA
[2] Microsoft Research, Redmond, WA 98052 USA

More on Self-Optimizing DRAM Controllers (II)

- Janani Mukundan and José F. Martinez
  "MORSE: Multi-Objective Reconfigurable Self-Optimizing Memory Scheduler"
  *Proceedings of the 18th International Symposium on High Performance Computer Architecture (**HPCA**)*, New Orleans, Louisiana, February 2012.

MORSE: Multi-objective Reconfigurable Self-optimizing Memory Scheduler

Janani Mukundan **José F. Martínez**

Computer Systems Laboratory
Cornell University
Ithaca, NY, 14850 USA

http://m3.csl.cornell.edu/

Memory Controllers are critical to research

They will become even more important

Sibyl: Reinforcement Learning based Data Placement in Hybrid SSDs

Self-Optimizing Hybrid SSD Controllers

Gagandeep Singh, Rakesh Nadig, Jisung Park, Rahul Bera, Nastaran Hajinazar, David Novo, Juan Gomez-Luna, Sander Stuijk, Henk Corporaal, and Onur Mutlu,
"Sibyl: Adaptive and Extensible Data Placement in Hybrid Storage Systems Using Online Reinforcement Learning"
*Proceedings of the 49th International Symposium on Computer Architecture (**ISCA**)*, New York, June 2022.
[Slides (pptx) (pdf)]
[arXiv version]
[Sibyl Source Code]
[Talk Video (16 minutes)]

Sibyl: Adaptive and Extensible Data Placement in Hybrid Storage Systems Using Online Reinforcement Learning

Gagandeep Singh[1] Rakesh Nadig[1] Jisung Park[1] Rahul Bera[1] Nastaran Hajinazar[1]
David Novo[3] Juan Gómez-Luna[1] Sander Stuijk[2] Henk Corporaal[2] Onur Mutlu[1]
[1]ETH Zürich [2]Eindhoven University of Technology [3]LIRMM, Univ. Montpellier, CNRS

https://arxiv.org/pdf/2205.07394.pdf

Sibyl

Adaptive and Extensible Data Placement in Hybrid Storage Systems Using Online Reinforcement Learning

Gagandeep Singh, Rakesh Nadig, Jisung Park, Rahul Bera, Nastaran Hajinazar, David Novo, Juan Gómez Luna, Sander Stuijk, Henk Corporaal, Onur Mutlu

SAFARI Research Group
safari.ethz.ch

ETH zürich

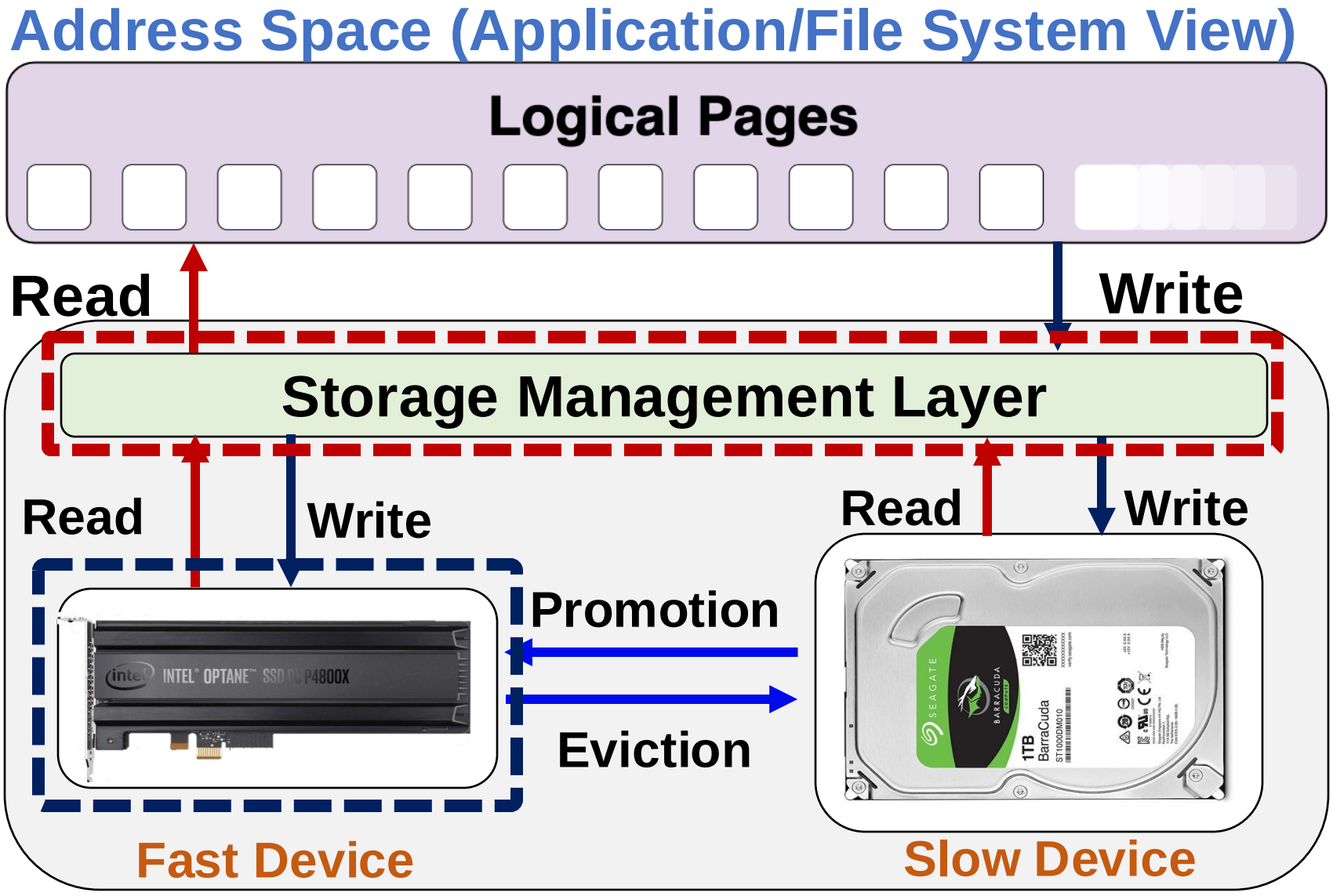
TU/e EINDHOVEN UNIVERSITY OF TECHNOLOGY

Executive Summary

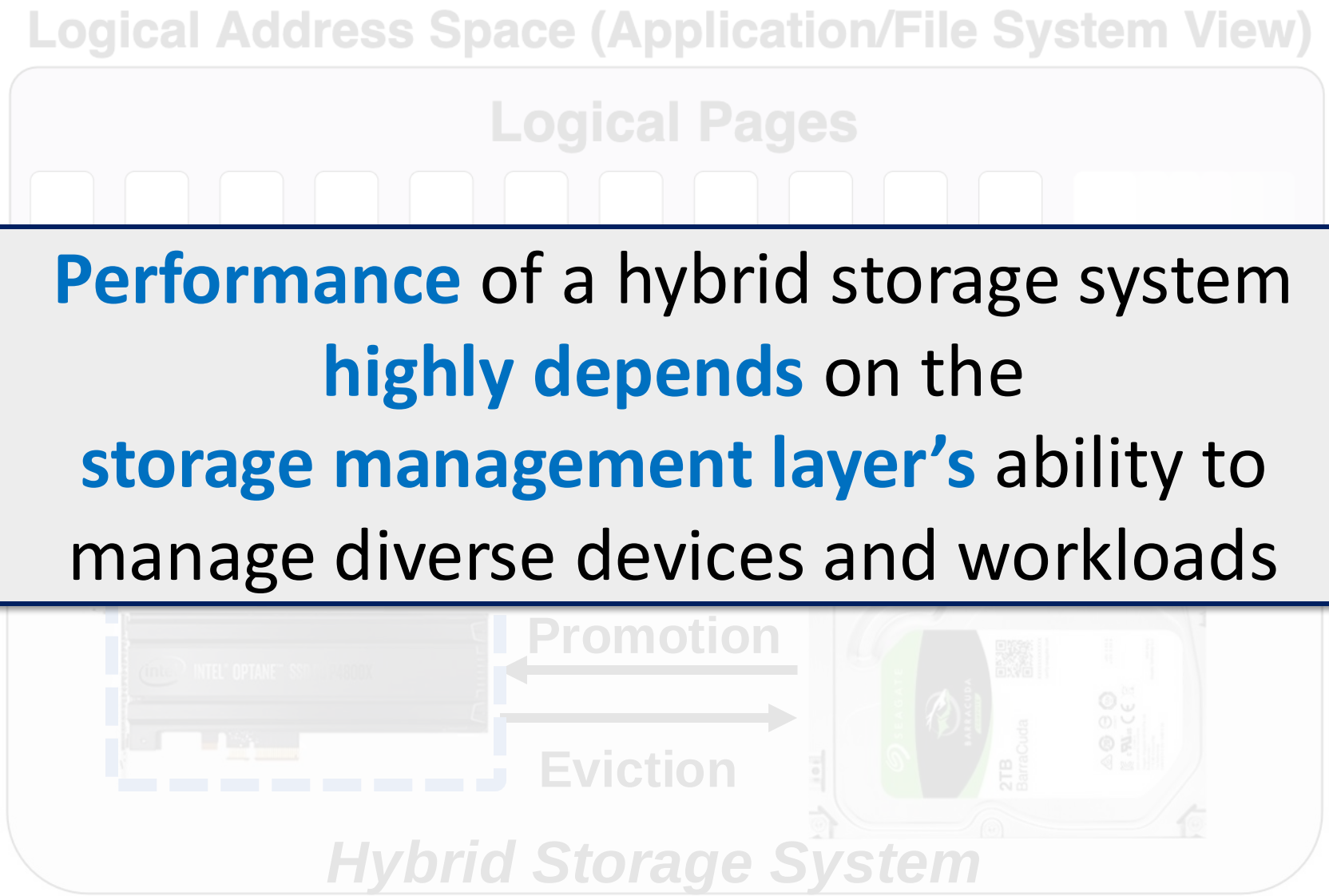
- **Background**: A hybrid storage system (HSS) uses multiple different storage devices to provide high and scalable storage capacity at high performance
- **Problem**: Two key shortcomings of prior data placement policies:
  - Lack of **adaptivity to:**
    - **Workload changes**
    - **Changes in device types and configurations**
  - Lack of **extensibility** to more devices
- **Goal**: Design a data placement technique that provides:
  - ***Adaptivity***, by **continuously learning and adapting** to the **application and underlying device characteristics**
  - ***Easy extensibility*** to incorporate a wide range of hybrid storage configurations
- **Contribution**: Sibyl, the first reinforcement learning-based data placement technique in hybrid storage systems that:
  - Provides **adaptivity** to changing workload demands and underlying device characteristics
  - Can **easily extend** to any number of storage devices
  - Provides **ease of design and implementation** that requires only a small computation overhead
- **Key Results**: Evaluate on **real systems** using a wide range of workloads
  - Sibyl **improves performance by 21.6%** compared to the best previous data placement technique in dual-HSS configuration
  - In a tri-HSS configuration, Sibyl outperforms the state-of-the-art-policy policy by **48.2%**
  - Sibyl achieves **80% of the performance** of an oracle policy with storage overhead of only **124.4 KiB**

https://github.com/CMU-SAFARI/Sibyl

Hybrid Storage System Basics



Hybrid Storage System Basics



Performance of a hybrid storage system **highly depends** on the **storage management layer's** ability to manage diverse devices and workloads

Key Shortcomings in Prior Techniques

We observe **two key shortcomings** that significantly limit the performance benefits of prior techniques

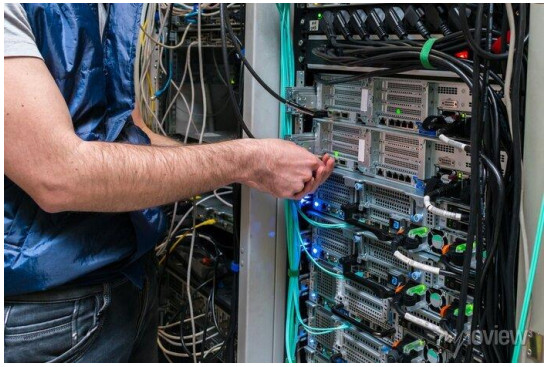
1. Lack of **adaptivity to**:
   a) **Workload changes**
   b) **Changes in device types and configuration**

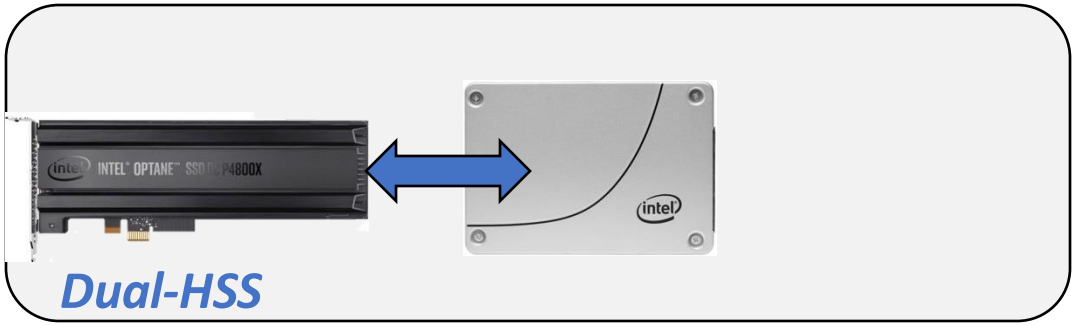
2. Lack of **extensibility** to more devices

Lack of Extensibility (1/2)

Rigid techniques that require significant effort to accommodate more than two devices

Change in storage configuration

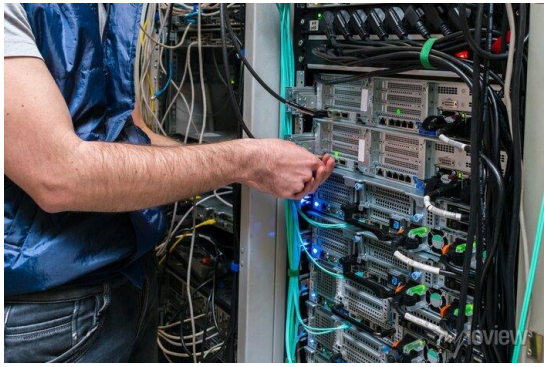




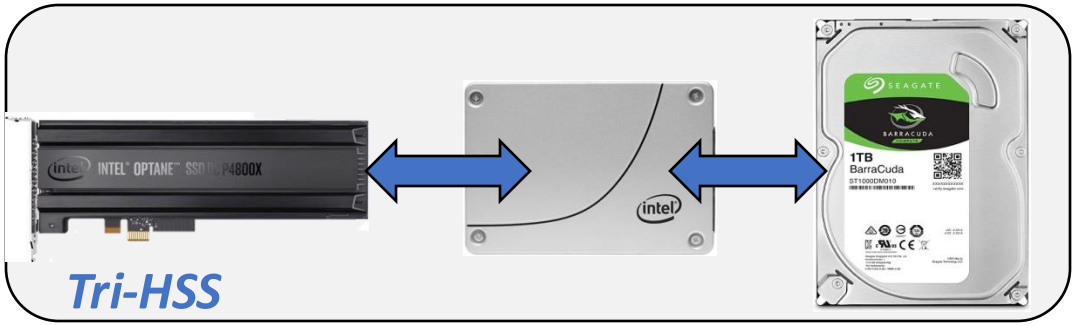
Lack of Extensibility (2/2)

Rigid techniques that require significant effort to accommodate more than two devices

Change in storage configuration



Design a new policy



Our Goal

A **data-placement mechanism** that can provide:

1. **Adaptivity,** by **continuously learning** and **adapting** to the application and underlying device characteristics
2. **Easy extensibility** to incorporate a wide range of hybrid storage configurations

Our Proposal

Sibyl

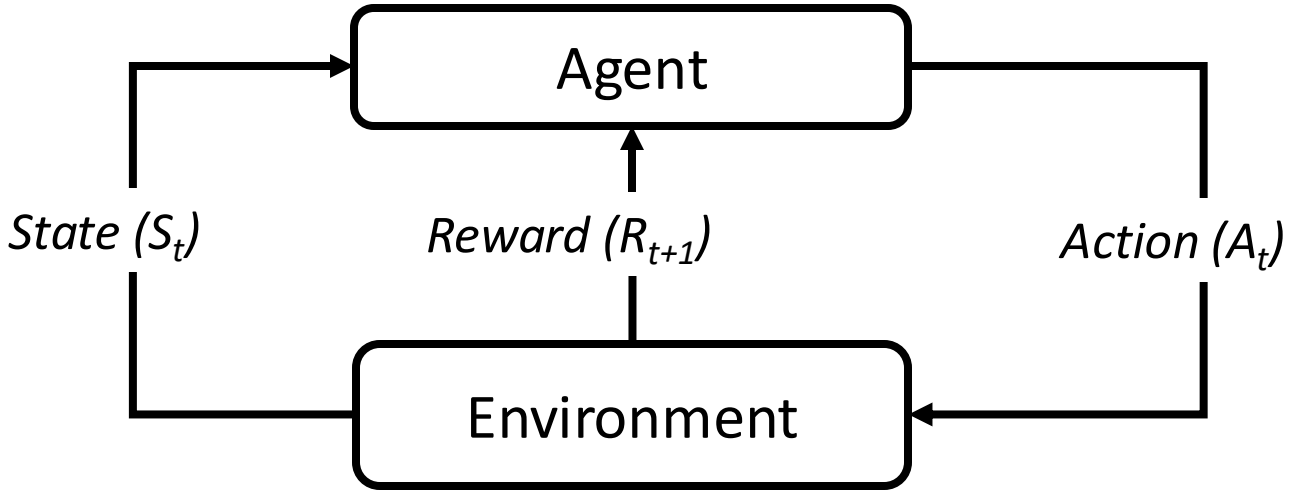
Formulates data placement in hybrid storage systems as a **reinforcement learning problem**

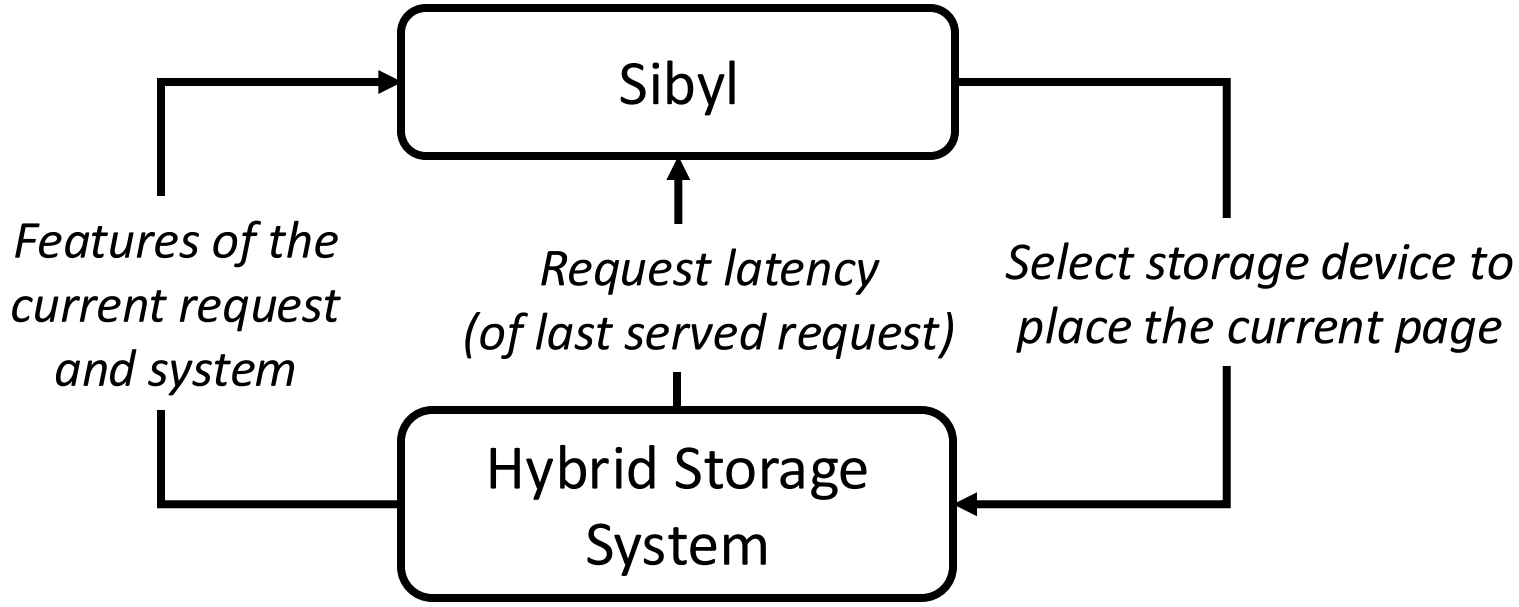
Sibyl is an oracle that makes accurate prophecies
https://en.wikipedia.org/wiki/Sibyl

Basics of Reinforcement Learning (RL)

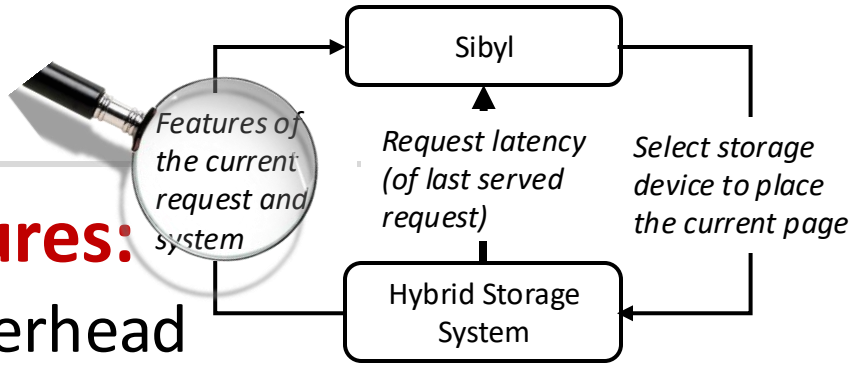
Agent learns to take an **action** in a given **state**
to maximize a numerical **reward**

Formulating Data Placement as RL





What is State?



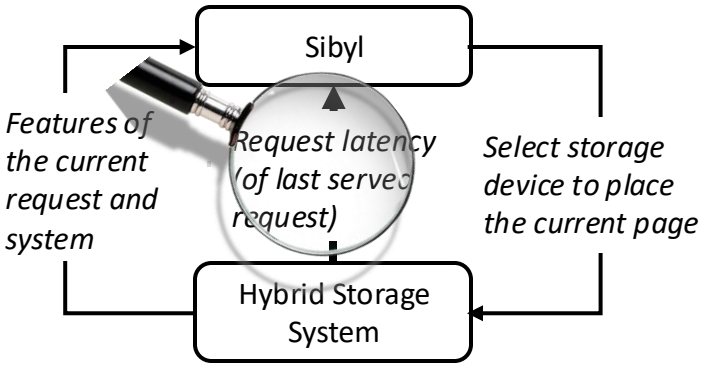
- **Limited number of state features:**
  - Reduce the implementation overhead
  - RL agent is more sensitive to reward

- ***6*-dimensional** vector of state features

$$O_t = (size_t, type_t, intr_t, cnt_t, cap_t, curr_t)$$

- We **quantize the state representation** into bins to reduce storage overhead

What is Reward?



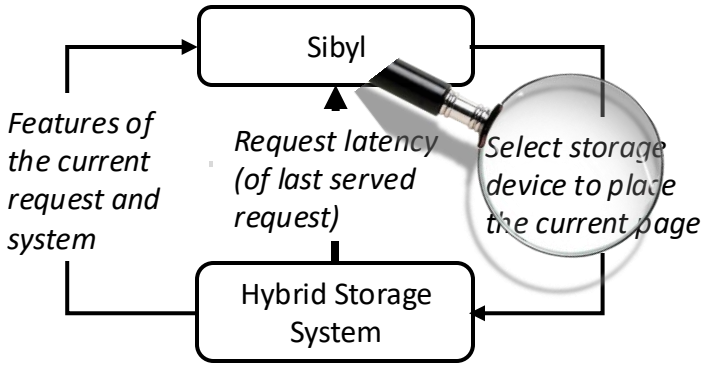
- Defines the **objective** of Sibyl

- We formulate the reward as a function of the **request latency**

- Encapsulates three key aspects:
  - **Internal state of the device** (e.g., read/write latencies, the latency of garbage collection, queuing delays, …)
  - **Throughput**
  - **Evictions**

- **More details in the paper**

What is Action?



- At every new page request, the action is to **select a storage device**
- Action can be **easily extended** to any number of storage devices
- Sibyl evicts a page when the fast device utilization is 100%
- Sibyl promotes a page when there is an update from the application

Talk Outline

Key Shortcomings of Prior Data Placement Techniques

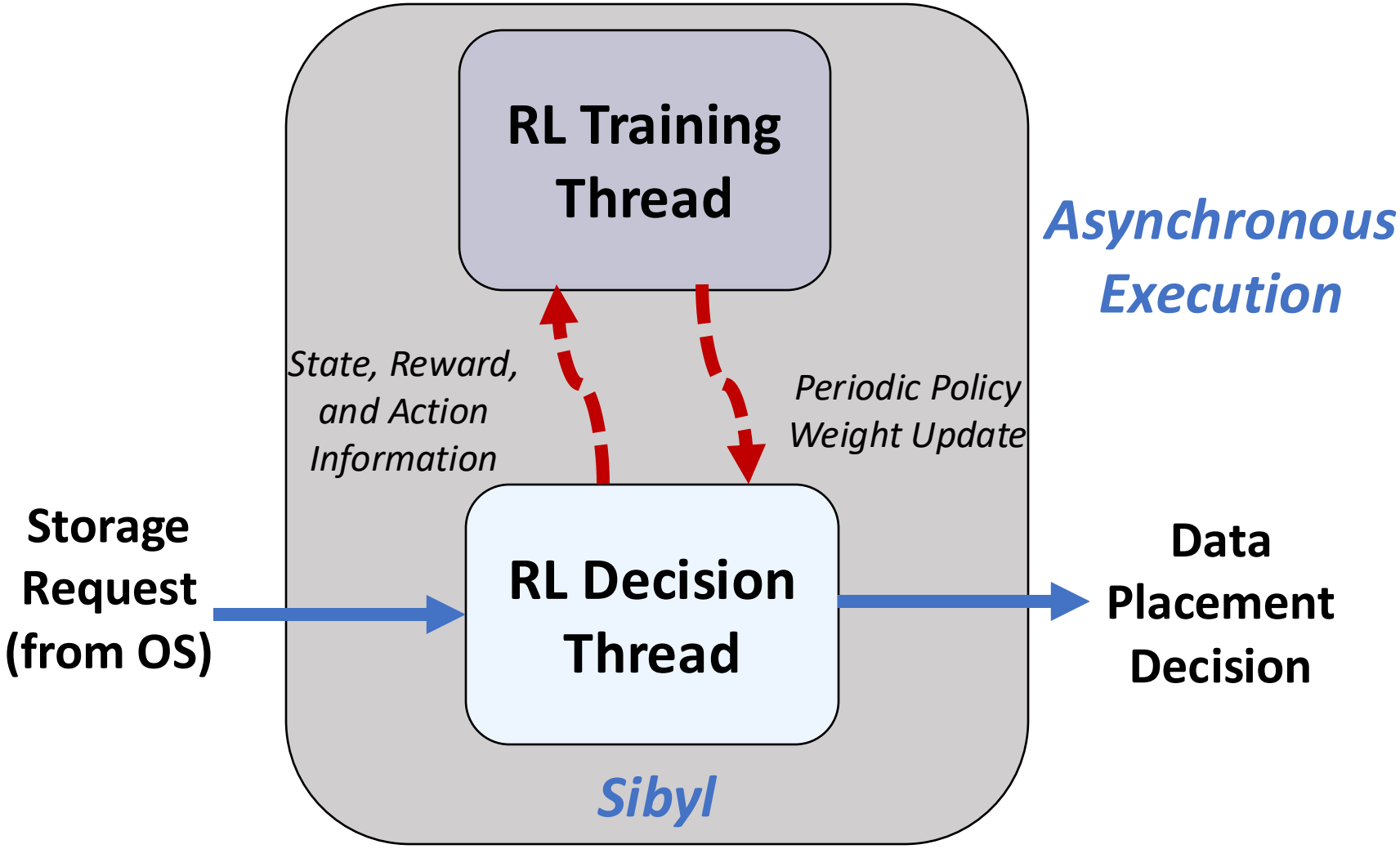
Formulating Data Placement as Reinforcement Learning

Sibyl: Overview

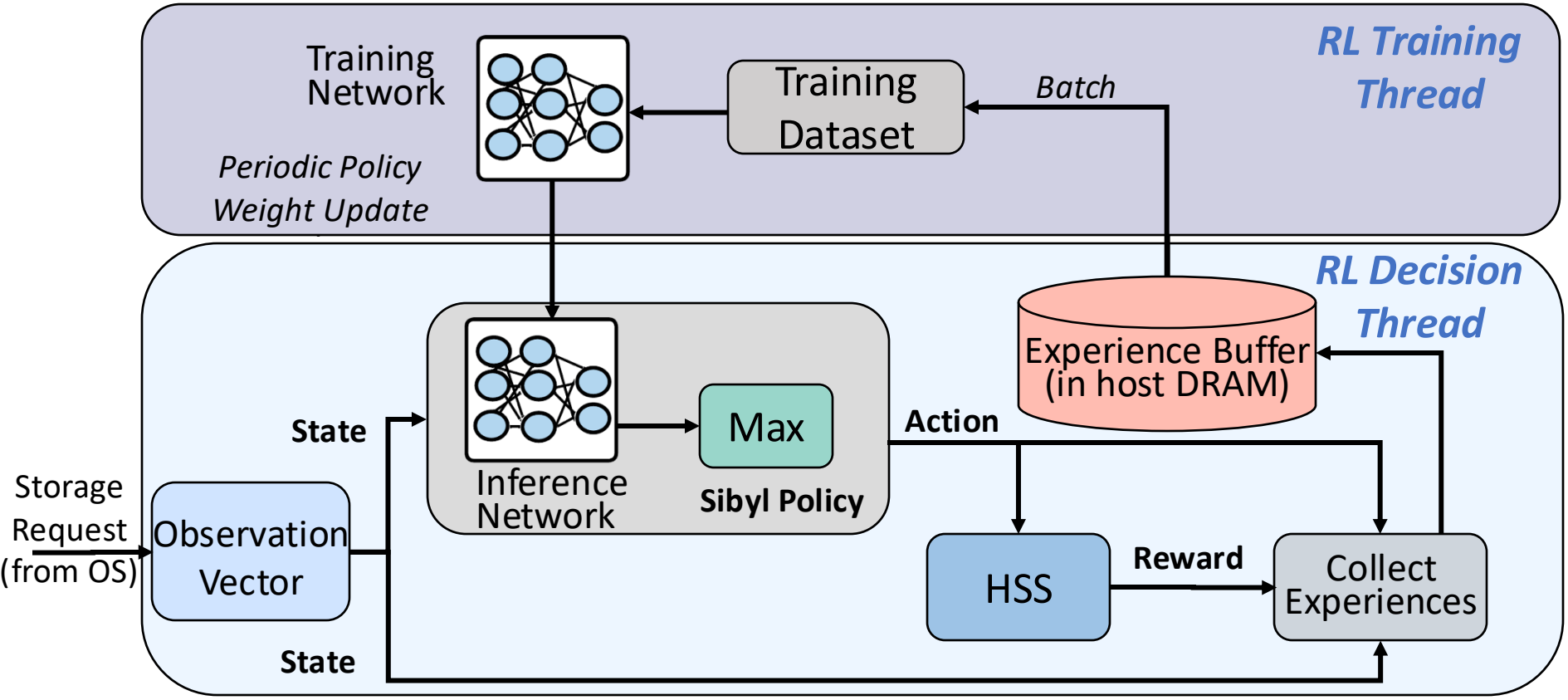
Evaluation of Sibyl and Key Results

Conclusion

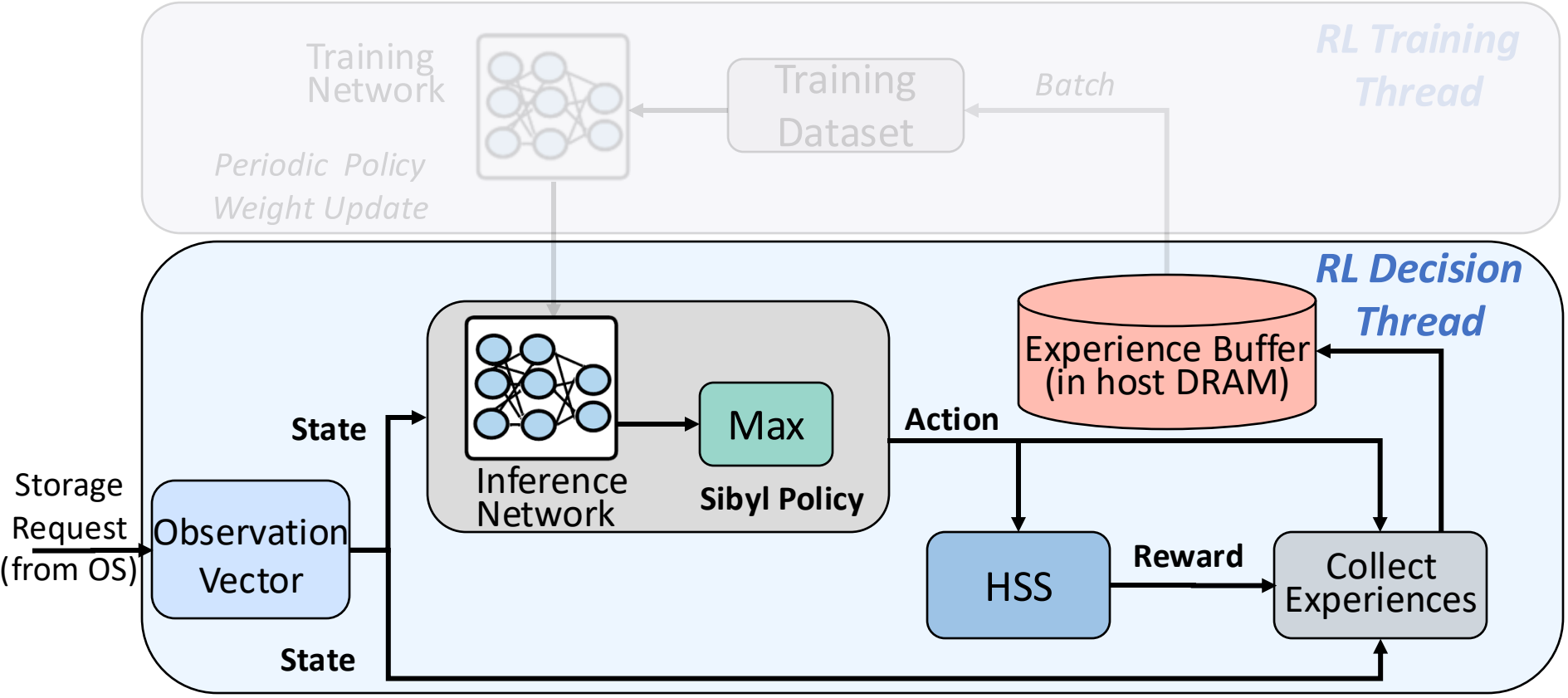
Sibyl Execution



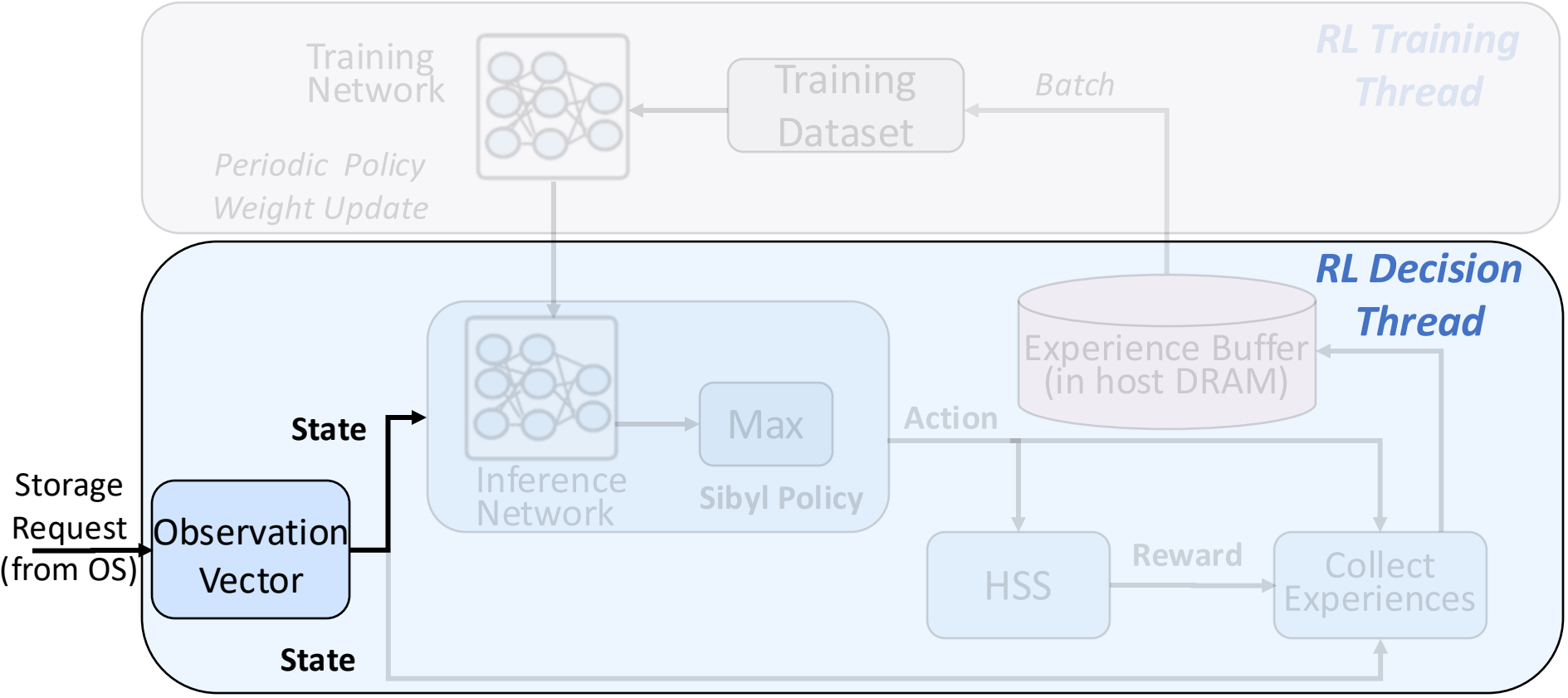
Sibyl Design: Overview



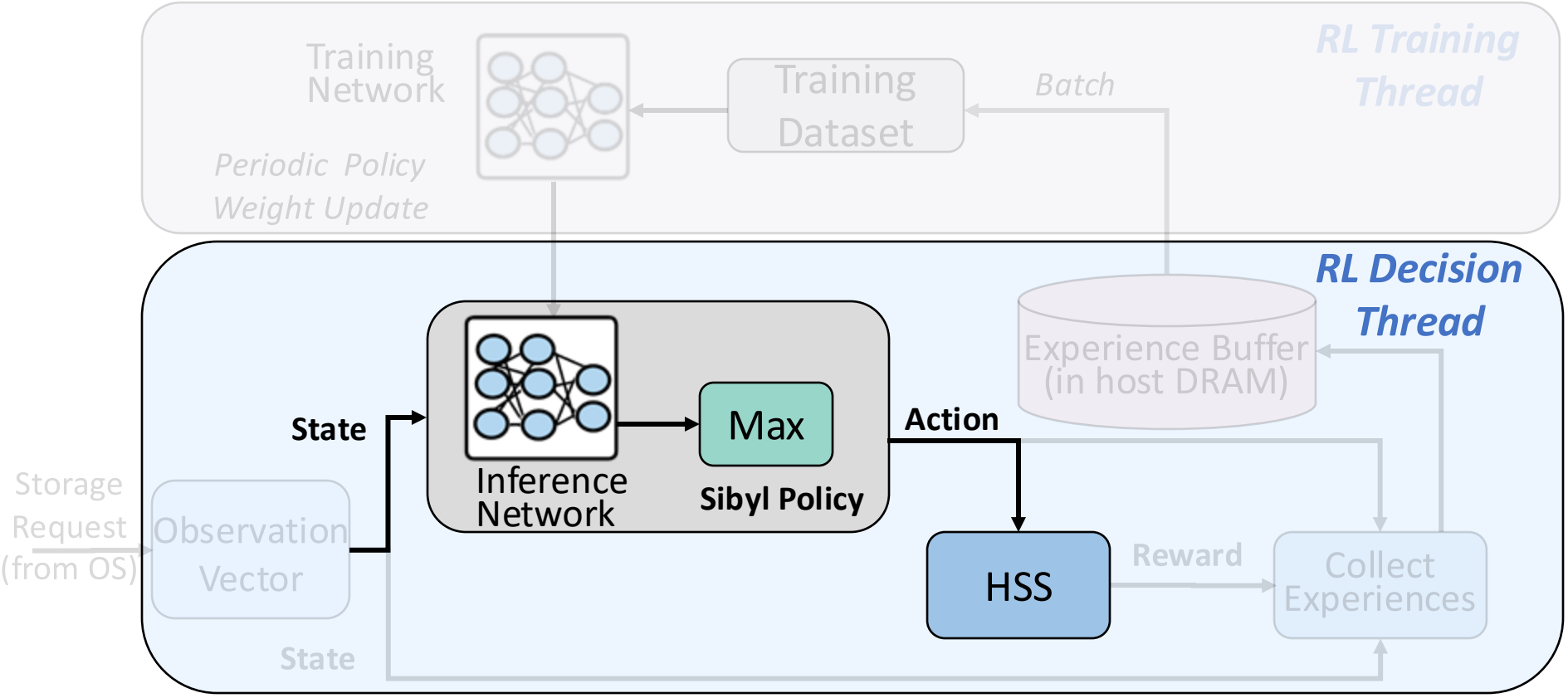
RL Decision Thread



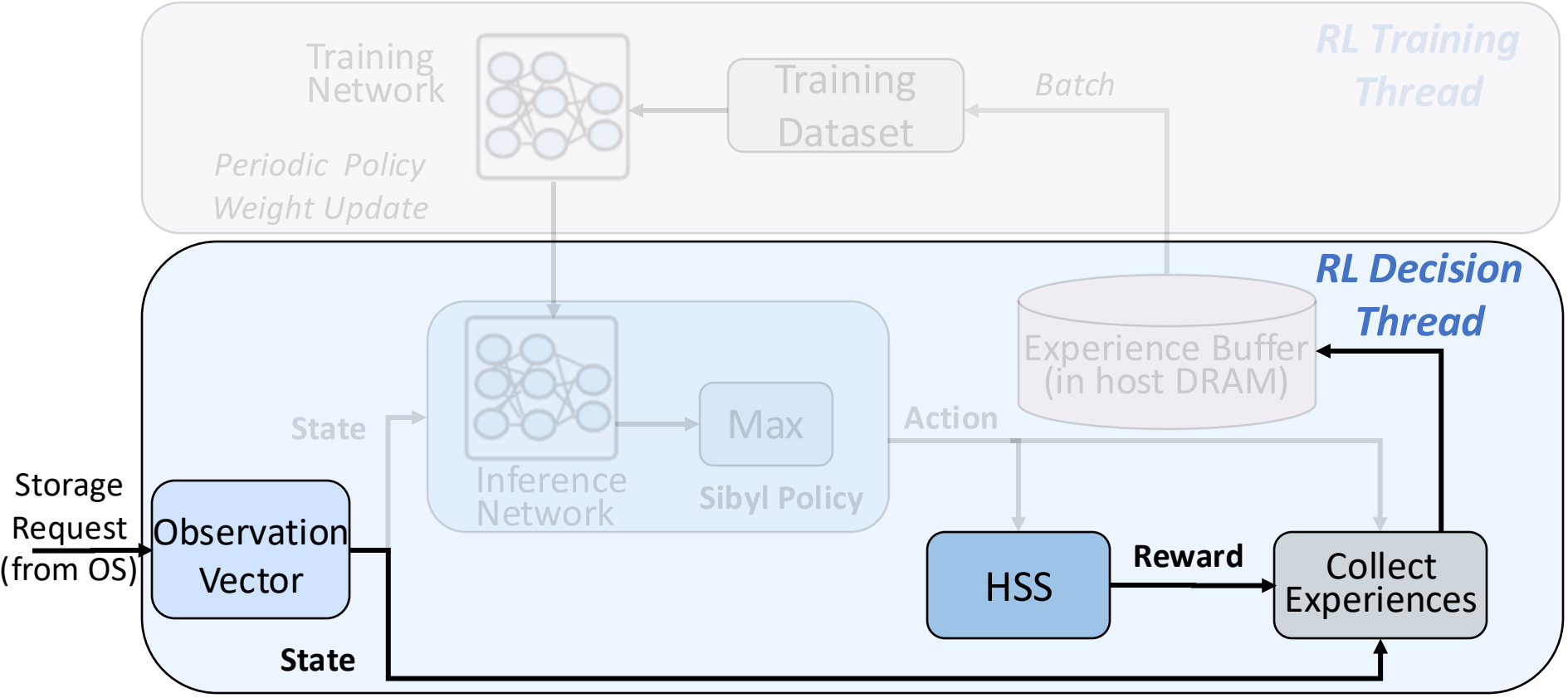
RL Decision Thread



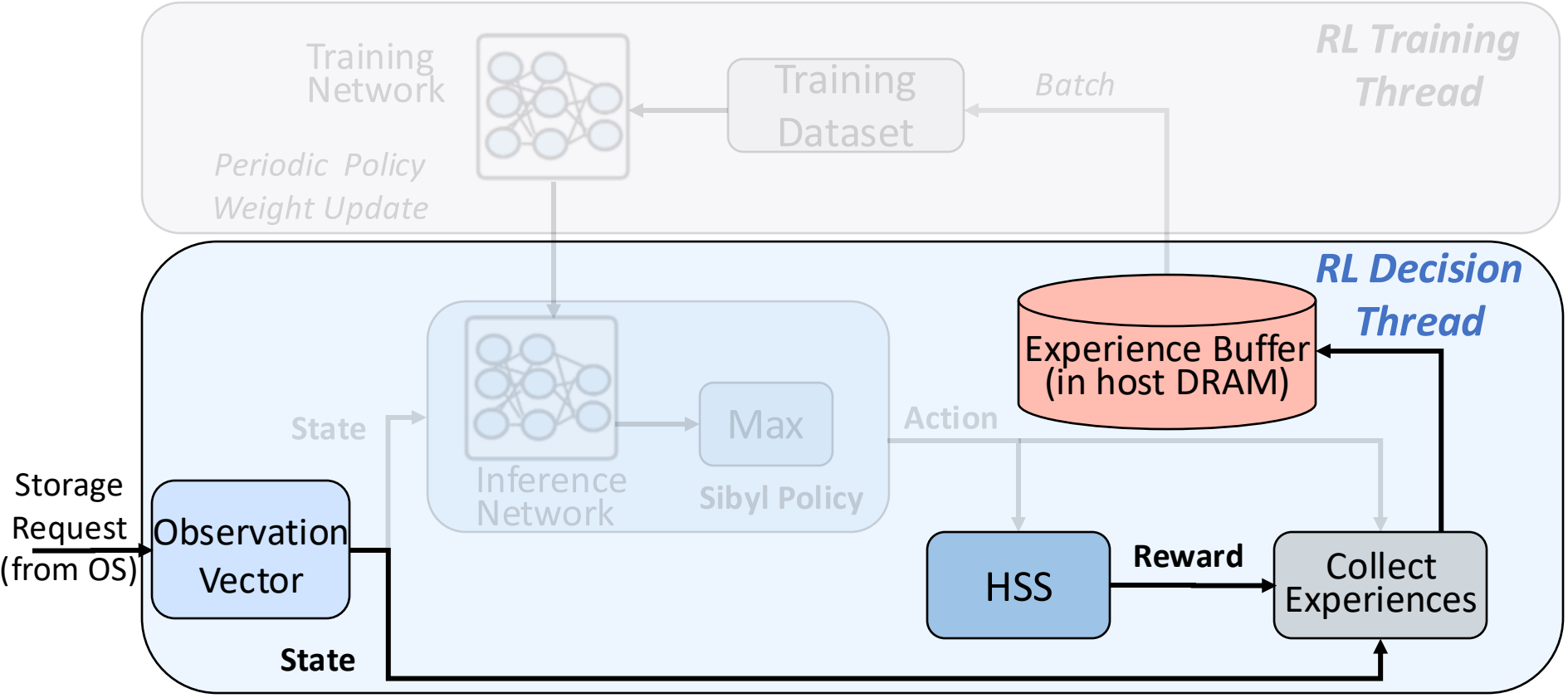
RL Decision Thread



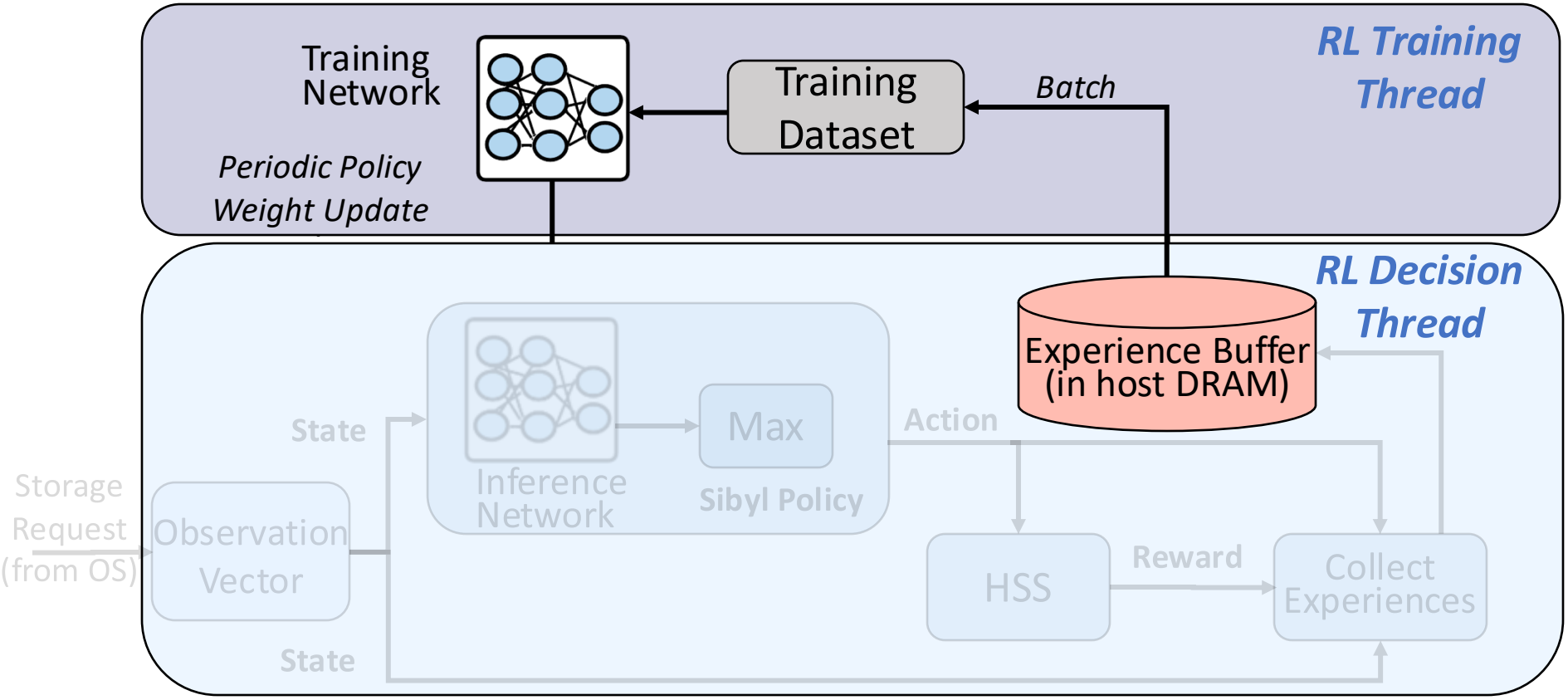
RL Decision Thread



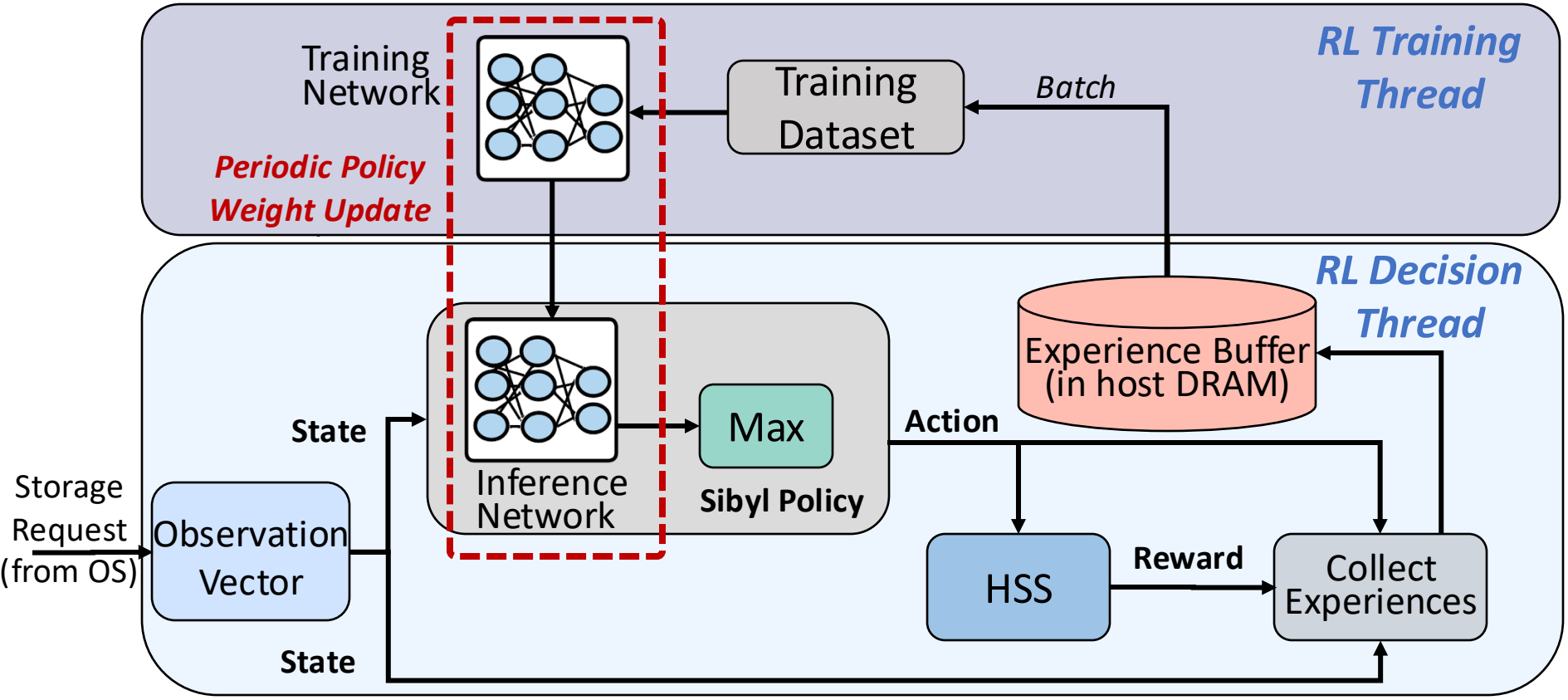
RL Decision Thread



RL Training Thread



Periodic Weight Transfer

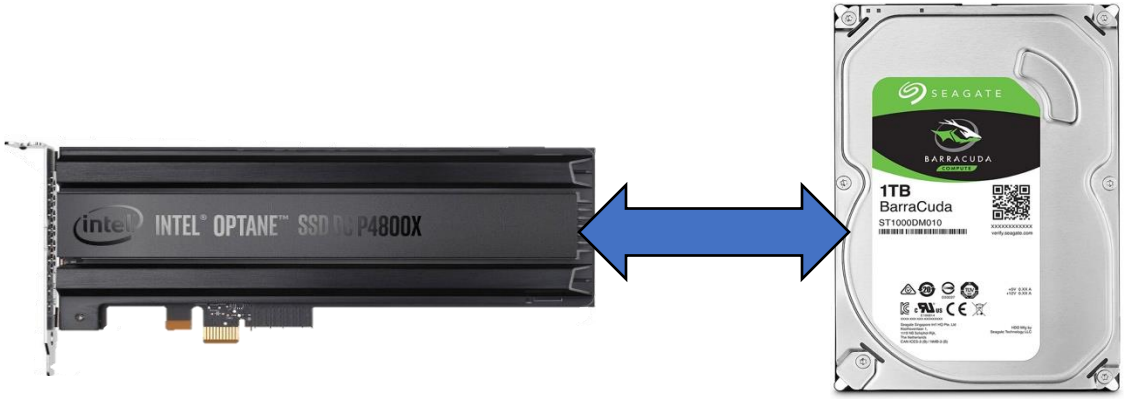


Evaluation Methodology (1/3)

- **Real system with various HSS configurations**
  - Dual-hybrid and tri-hybrid systems

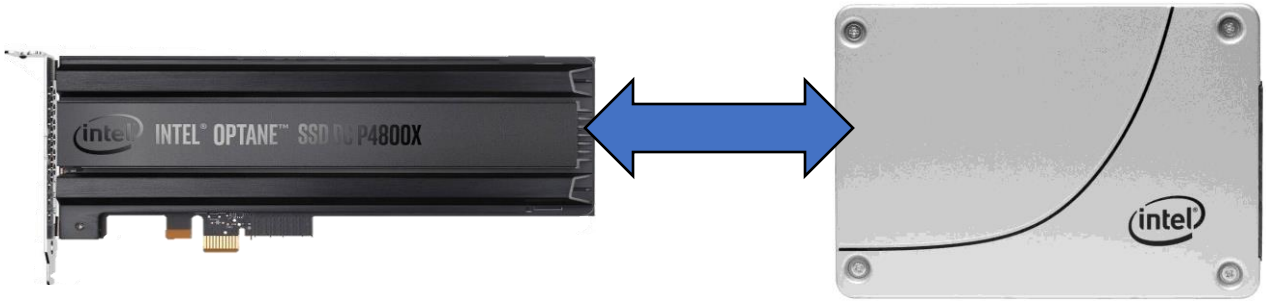
Evaluation Methodology (2/3)

Cost-Oriented HSS Configuration



High-end SSD *Low-end HDD*

Performance-Oriented HSS Configuration

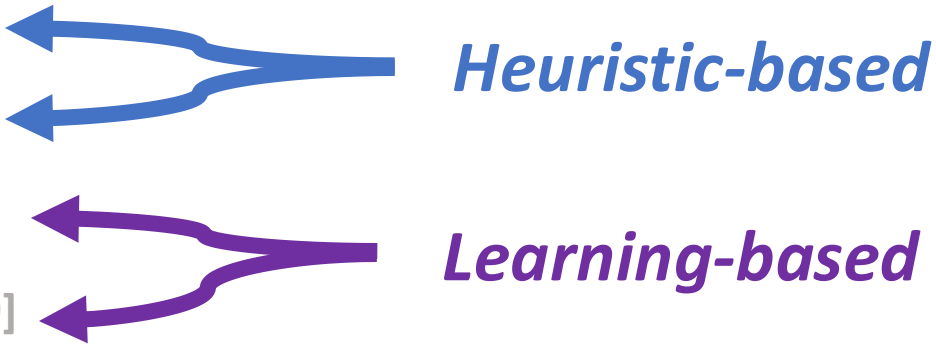


High-end SSD *Middle-end SSD*

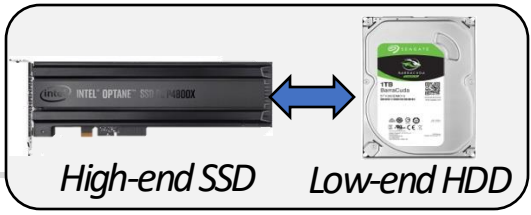
Evaluation Methodology (3/3)

- **18 different workloads** from:
  - MSR Cambridge and Filebench Suites

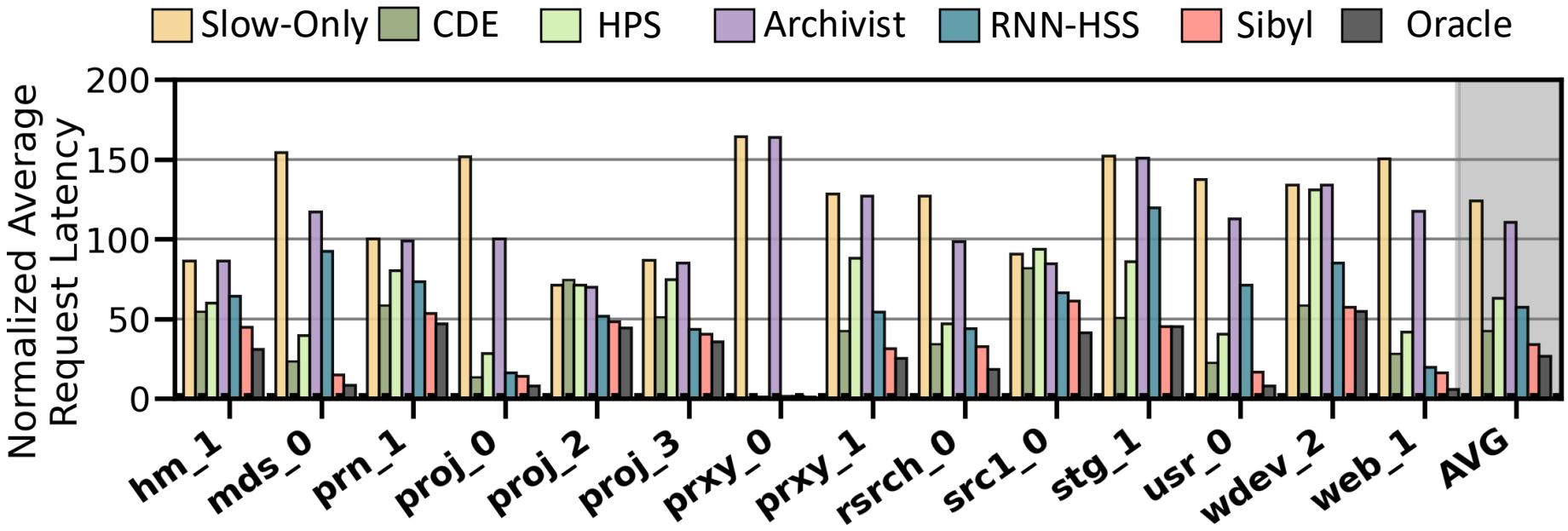
- **Four** state-of-the-art data placement baselines:
  - CDE [Matsui+, Proc. IEEE'17]
  - HPS [Meswani+, HPCA'15]
  - Archivist [Ren+, ICCD'19]
  - RNN-HSS [Doudali+, HPDC'19]



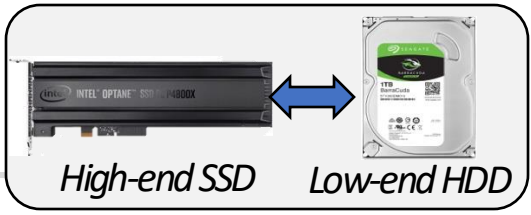
Performance Analysis



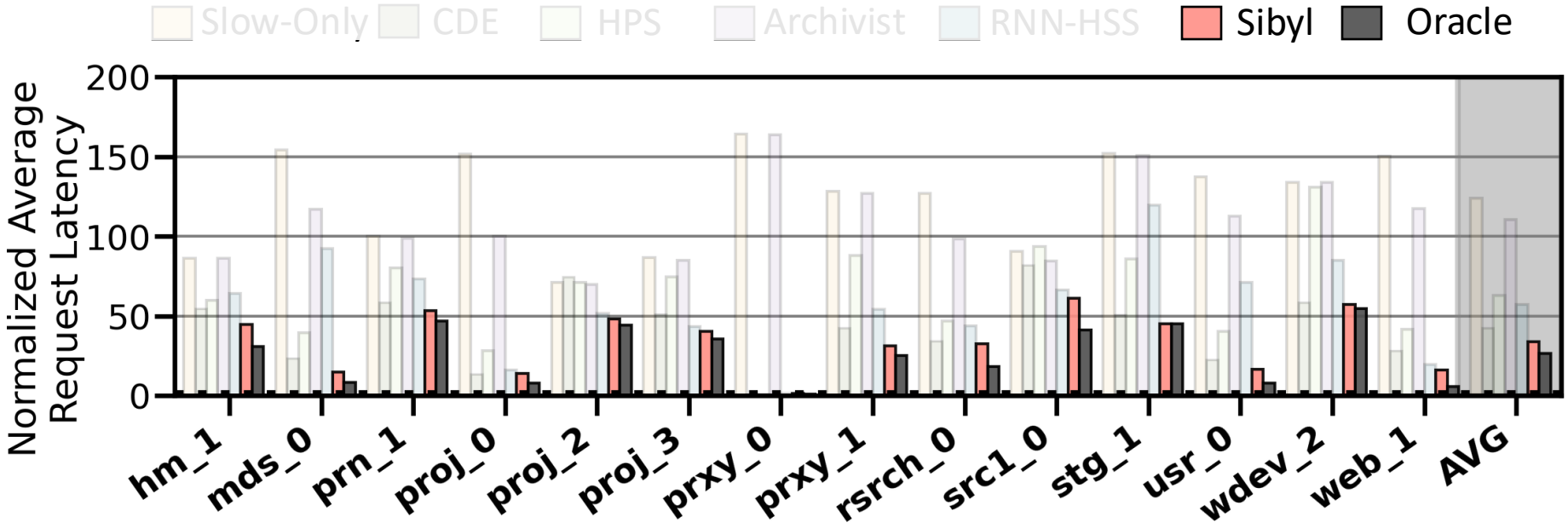
Cost-Oriented HSS Configuration



Performance Analysis

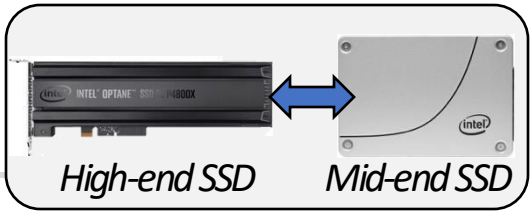


Cost-Oriented HSS Configuration

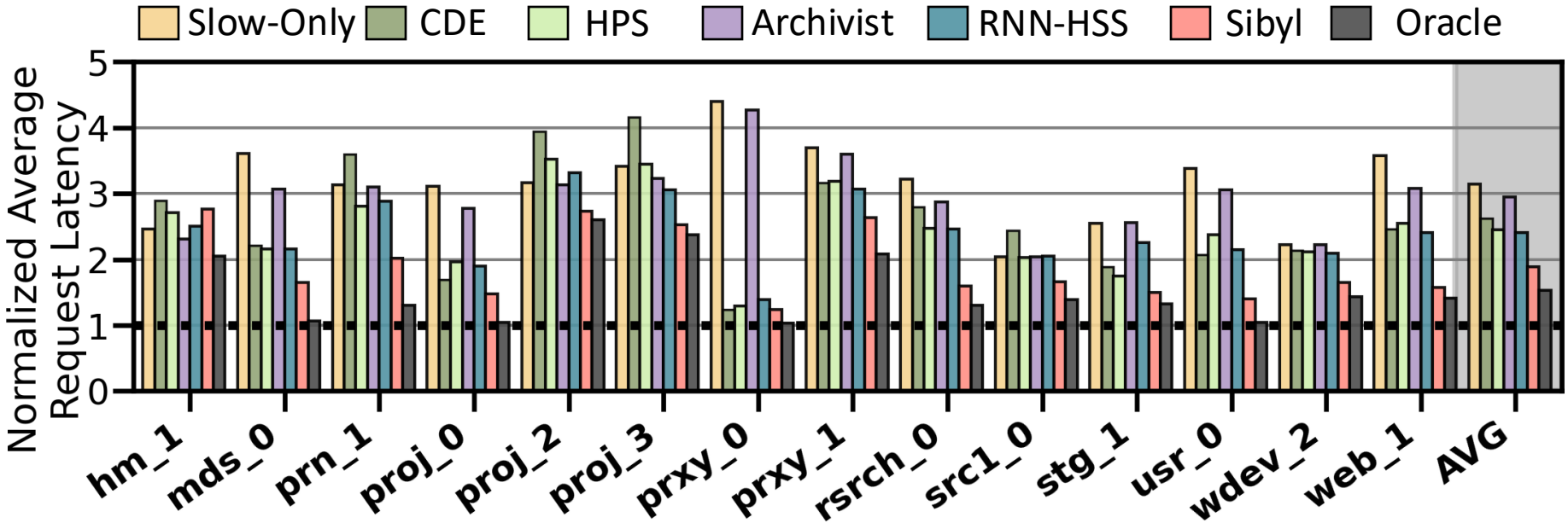


Sibyl consistently **outperforms all the baselines for all the workloads**

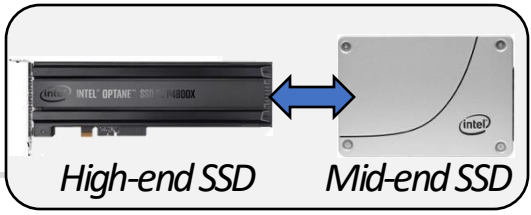
Performance Analysis



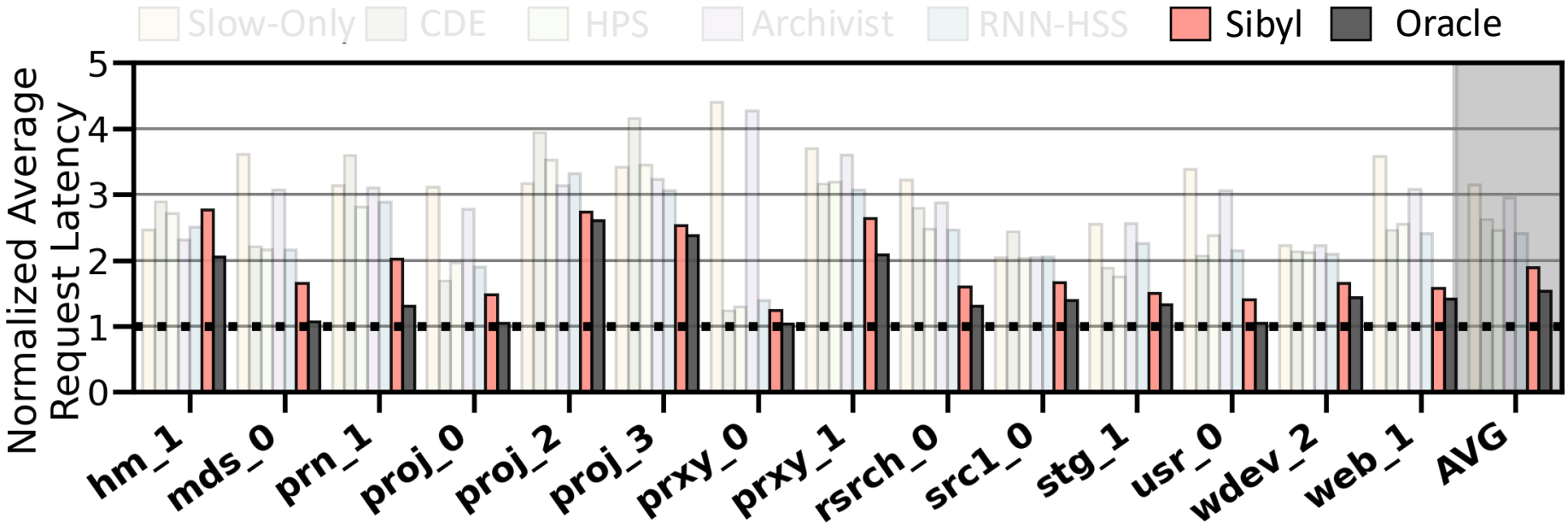
Performance-Oriented HSS Configuration



Performance Analysis

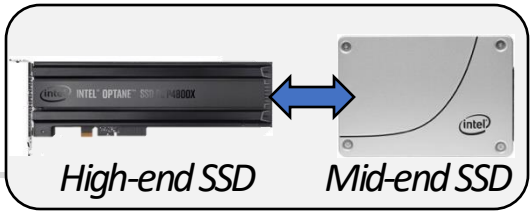


Performance-Oriented HSS Configuration

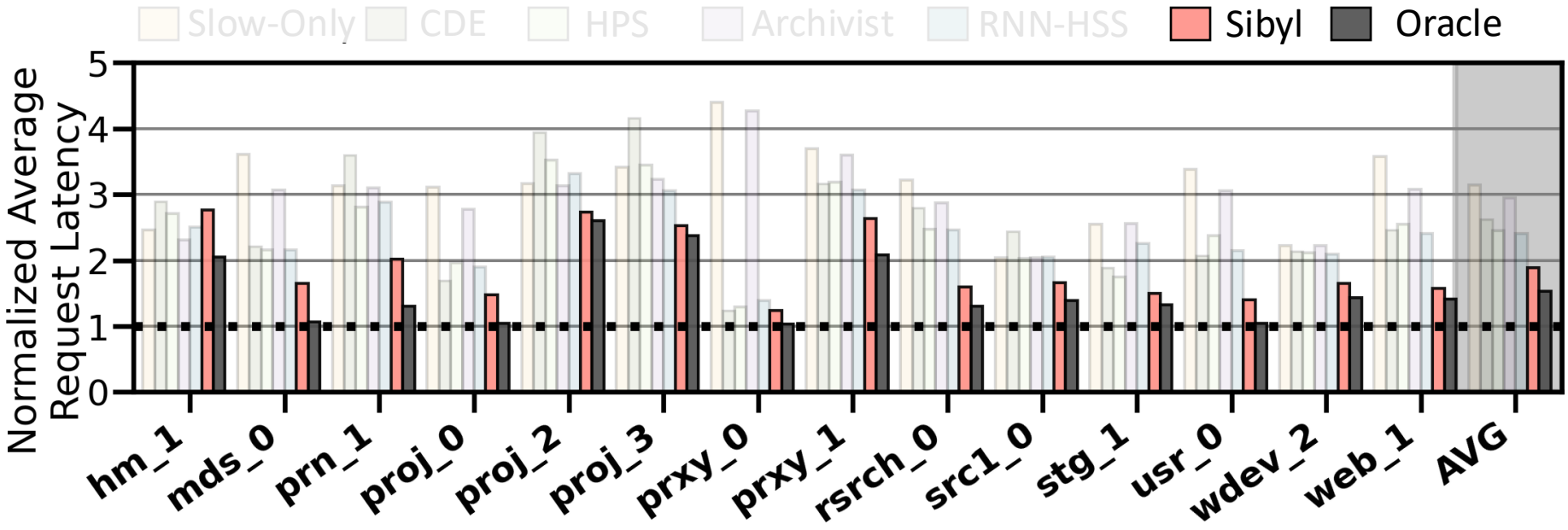


Sibyl provides **21.6% performance improvement** by **dynamically adapting its data placement policy**

Performance Analysis

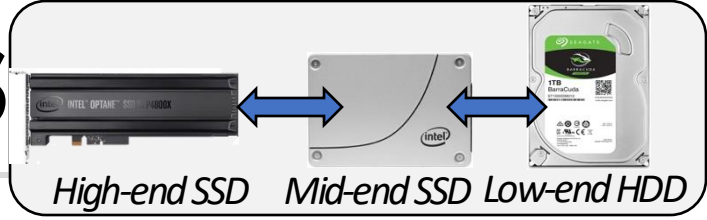


Performance-Oriented HSS Configuration



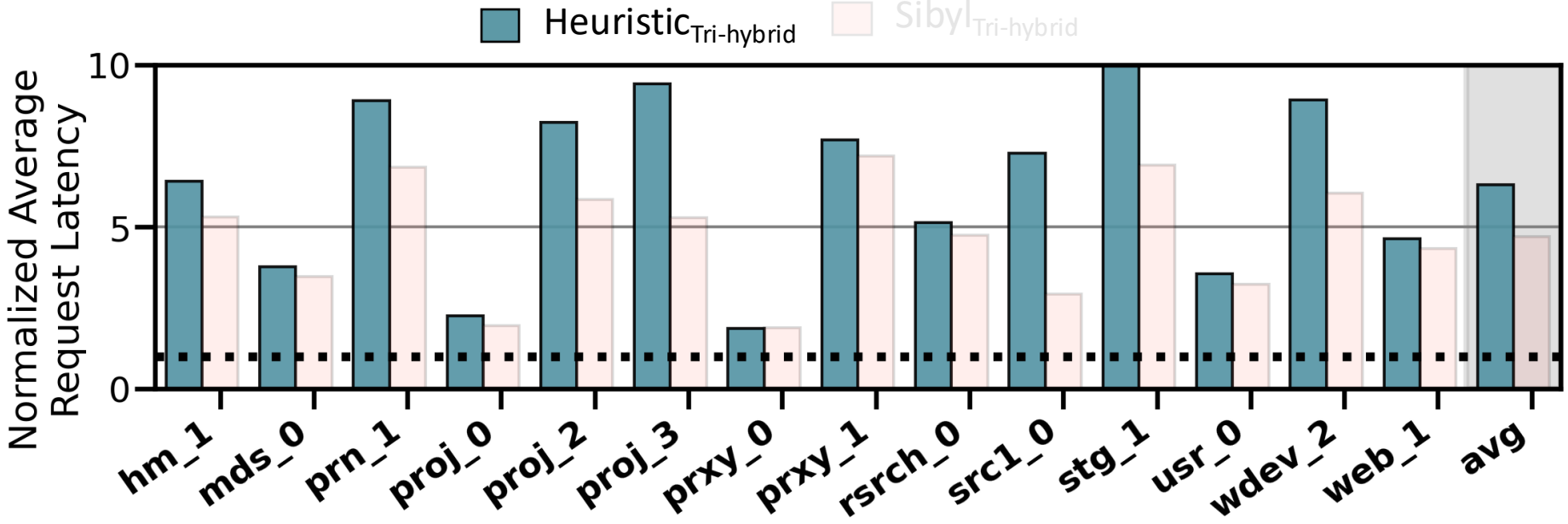
Sibyl achieves **80% of the performance of an oracle policy** that has complete knowledge of future access patterns

Performance on Tri-HSS

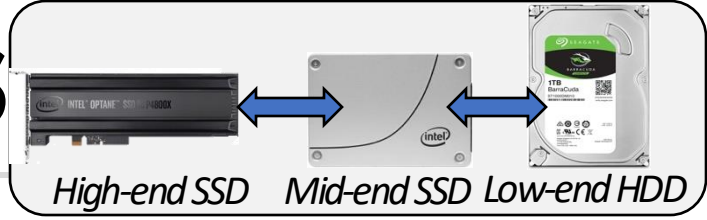


Extending Sibyl for **more devices**:

1. **Add a new action**
2. **Add the remaining capacity** of the new device as a state feature

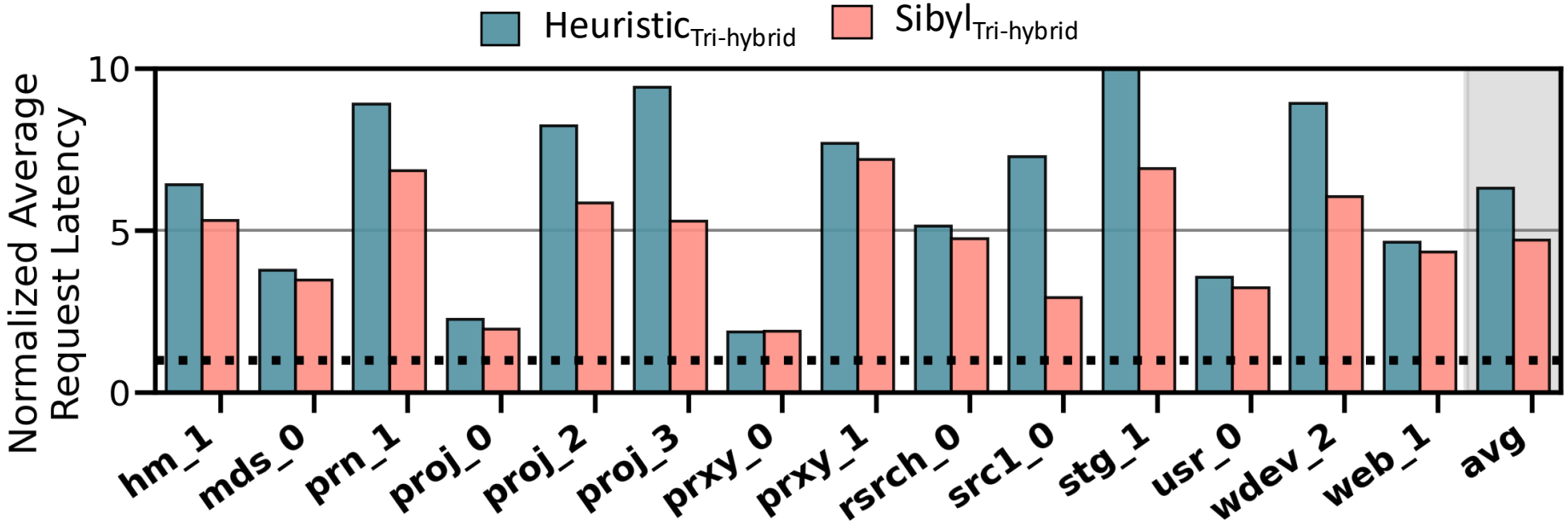


Performance on Tri-HSS

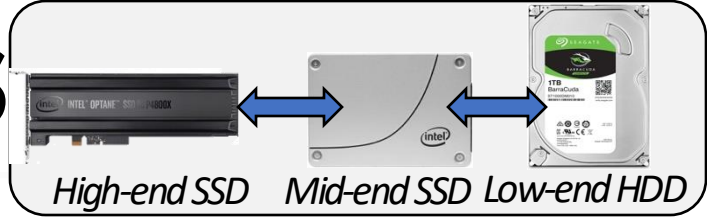


Extending Sibyl for **more devices**:

1. **Add a new action**
2. **Add the remaining capacity** of the new device as a state feature



Performance on Tri-HSS



Extending Sibyl for more devices:

1. Add a new action

Sibyl **outperforms** the state-of-the-art data placement policy by **48.2% in a real tri-hybrid system**

Sibyl reduces the system architect's burden by providing **ease of extensibility**

Sibyl's Overhead

- **124.4 KiB** of total storage cost
  - Experience buffer, inference and training network
- **40-bit** metadata overhead per page for state features
- Inference latency of **~10ns**
- Training latency of **~2us**

More in the Paper (1/3)

- **Throughput (IOPS) evaluation**
  - Sibyl provides high IOPS compared to baseline policies because it indirectly captures throughput (size/latency)

- Evaluation on **unseen workloads**
  - Sibyl can effectively adapt its policy to highly dynamic workloads

- Evaluation on **mixed workloads**
  - Sibyl provides equally-high performance benefits as in single workloads

More in the Paper (2/3)

- Evaluation on **different features**
  - Sibyl autonomously decides which features are important to maximize the performance
- Evaluation with **different hyperparameter values**
- Sensitivity to **fast storage capacity**
  - Sibyl provides scalability by dynamically adapting its policy to available storage size
- **Explainability analysis** of Sibyl's decision making
  - Explain Sibyl's actions for different workload characteristics and device configurations

More in the Paper (3/3)

Sibyl: Adaptive and Extensible Data Placement in Hybrid Storage Systems Using Online Reinforcement Learning

Gagandeep Singh[1] Rakesh Nadig[1] Jisung Park[1] Rahul Bera[1] Nastaran Hajinazar[1]
David Novo[3] Juan Gómez-Luna[1] Sander Stuijk[2] Henk Corporaal[2] Onur Mutlu[1]
[1]ETH Zürich [2]Eindhoven University of Technology [3]LIRMM, Univ. Montpellier, CNRS

https://arxiv.org/pdf/2205.07394.pdf

https://github.com/CMU-SAFARI/Sibyl

Conclusion

- **We introduced Sibyl,** the first reinforcement learning-based data placement technique in hybrid storage systems that provides
  - **Adaptivity**
  - **Easily extensibility**
  - **Ease of design and implementation**
- **We evaluated Sibyl** on **real systems** using many different workloads
  - Sibyl **improves performance by 21.6%** compared to the best prior data placement policy in a dual-HSS configuration
  - In a tri-HSS configuration, Sibyl **outperforms** the state-of-the-art-data placement policy by **48.2%**
  - Sibyl achieves **80% of the performance** of an oracle policy with a storage overhead of only **124.4 KiB**

https://github.com/CMU-SAFARI/Sibyl

Major Directions

- Consider other optimization objectives
  - Energy consumption, endurance of storage devices…..
  - Design better reward structures

- Optimize data migration in hybrid storage systems
  - Explore machine learning (ML) techniques to make data migration adaptive and extensible
  - How do we coordinate multiple ML techniques?

- How do we improve these policies in other heterogeneous memory systems?
  - DRAM + NVM, CPU Caches + DRAM
  - Design RL models keeping latency constraints in mind

ISCA 2022 Paper, Slides, Videos

- Gagandeep Singh, Rakesh Nadig, Jisung Park, Rahul Bera, Nastaran Hajinazar, David Novo, Juan Gomez-Luna, Sander Stuijk, Henk Corporaal, and Onur Mutlu, **"Sibyl: Adaptive and Extensible Data Placement in Hybrid Storage Systems Using Online Reinforcement Learning"**
  *Proceedings of the 49th International Symposium on Computer Architecture (**ISCA**)*, New York, June 2022.
  [Slides (pptx) (pdf)]
  [arXiv version]
  [Sibyl Source Code]
  [Talk Video (16 minutes)]

Sibyl: Adaptive and Extensible Data Placement in Hybrid Storage Systems Using Online Reinforcement Learning

Gagandeep Singh[1] Rakesh Nadig[1] Jisung Park[1] Rahul Bera[1] Nastaran Hajinazar[1]
David Novo[3] Juan Gómez-Luna[1] Sander Stuijk[2] Henk Corporaal[2] Onur Mutlu[1]
[1]ETH Zürich [2]Eindhoven University of Technology [3]LIRMM, Univ. Montpellier, CNRS

https://arxiv.org/pdf/2205.07394.pdf

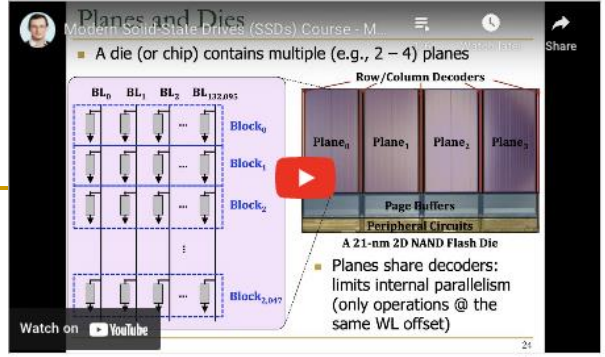
SSD Course (Spring 2023)

- **Spring 2023 Edition:**
  - https://safari.ethz.ch/projects_and_seminars/spring2023/doku.php?id=modern_ssds
- **Fall 2022 Edition:**
  - https://safari.ethz.ch/projects_and_seminars/fall2022/doku.php?id=modern_ssds

- **Youtube Livestream (Spring 2023):**
  - https://www.youtube.com/watch?v=4VTwOMmsnJY&list=PL5Q2soXY2Zi_8qOM5Icpp8hB2SHtm4z57&pp=iAQB
- **Youtube Livestream (Fall 2022):**
  - https://www.youtube.com/watch?v=hqLrd-Uj0aU&list=PL5Q2soXY2Zi9BJhenUq4JI5bwhAMpAp13&pp=iAQB

- Project course
  - Taken by Bachelor's/Master's students
  - SSD Basics and Advanced Topics
  - Hands-on research exploration
  - Many research readings

https://www.youtube.com/onurmutlulectures



Fall 2022 Meetings/Schedule

Week	Date	Livestream	Meeting	Learning Materials	Assignments
W1	06.10		M1: P&S Course Presentation PDF PPT	Required Recommended	
W2	12.10	YouTube Live	M2: Basics of NAND Flash-Based SSDs PDF PPT	Required Recommended	
W3	19.10	YouTube Live	M3: NAND Flash Read/Write Operations PDF PPT	Required Recommended	
W4	26.10	YouTube Live	M4: Processing inside NAND Flash PDF PPT	Required Recommended	
W5	02.11	YouTube Live	M5: Advanced NAND Flash Commands & Mapping PDF PPT	Required Recommended	
W6	09.11	YouTube Live	M6: Processing inside Storage PDF PPT	Required Recommended	
W7	23.11	YouTube Live	M7: Address Mapping & Garbage Collection PDF PPT	Required Recommended	
W8	30.11	YouTube Live	M8: Introduction to MQSim PDF PPT	Required Recommended	
W9	14.12	YouTube Live	M9: Fine-Grained Mapping and Multi-Plane Operation-Aware Block Management PDF PPT	Required Recommended	
W10	04.01.2023	YouTube Premiere	M10a: NAND Flash Basics PDF PPT	Required Recommended	
			M10b: Reducing Solid-State Drive Read Latency by Optimizing Read-Retry PDF PPT Paper	Required Recommended	
			M10c: Evanesco: Architectural Support for Efficient Data Sanitization in Modern Flash-Based Storage Systems PDF PPT Paper	Required Recommended	
			M10d: DeepSketch: A New Machine Learning-Based Reference Search Technique for Post-Deduplication Delta Compression PDF PPT Paper	Required Recommended	
W11	11.01	YouTube Live	M11: FLIN: Enabling Fairness and Enhancing Performance in Modern NVMe Solid State Drives PDF PPT	Required	
W12	25.01	YouTube Premiere	M12: Flash Memory and Solid-State Drives PDF PPT	Recommended	

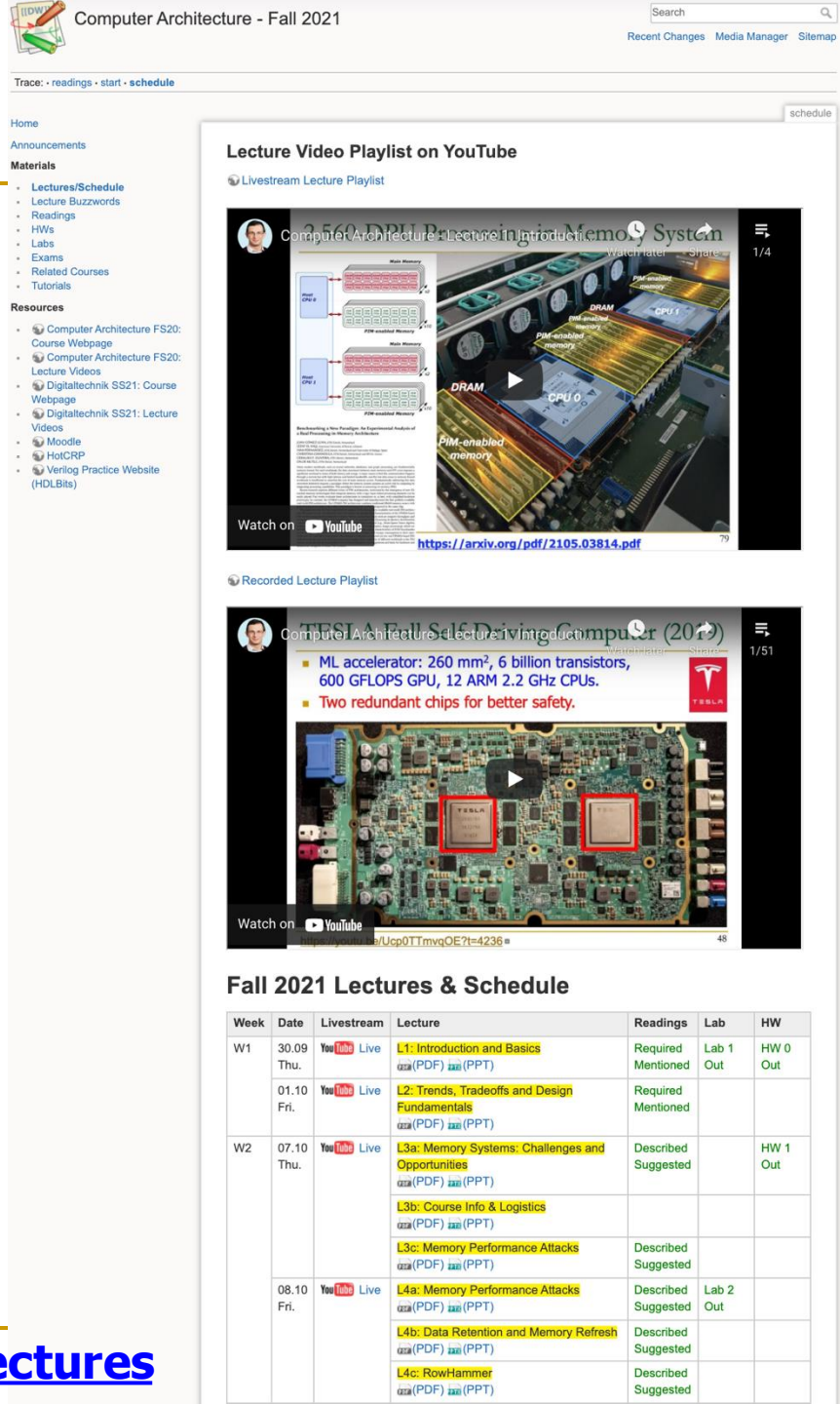
Comp Arch (Fall 2021)

- **Fall 2021 Edition:**
  - https://safari.ethz.ch/architecture/fall2021/doku.php?id=schedule
- **Fall 2020 Edition:**
  - https://safari.ethz.ch/architecture/fall2020/doku.php?id=schedule

- **Youtube Livestream (2021):**
  - https://www.youtube.com/watch?v=4yfkM_5EFgo&list=PL5Q2soXY2Zi-Mnk1PxjEIG32HAGILkTOF
- **Youtube Livestream (2020):**
  - https://www.youtube.com/watch?v=c3mPdZA-Fmc&list=PL5Q2soXY2Zi9xidyIgBxUz7xRPS-wisBN

- Master's level course
  - Taken by Bachelor's/Masters/PhD students
  - Cutting-edge research topics + fundamentals in Computer Architecture
  - 5 Simulator-based Lab Assignments
  - Potential research exploration
  - Many research readings

https://www.youtube.com/onurmutlulectures



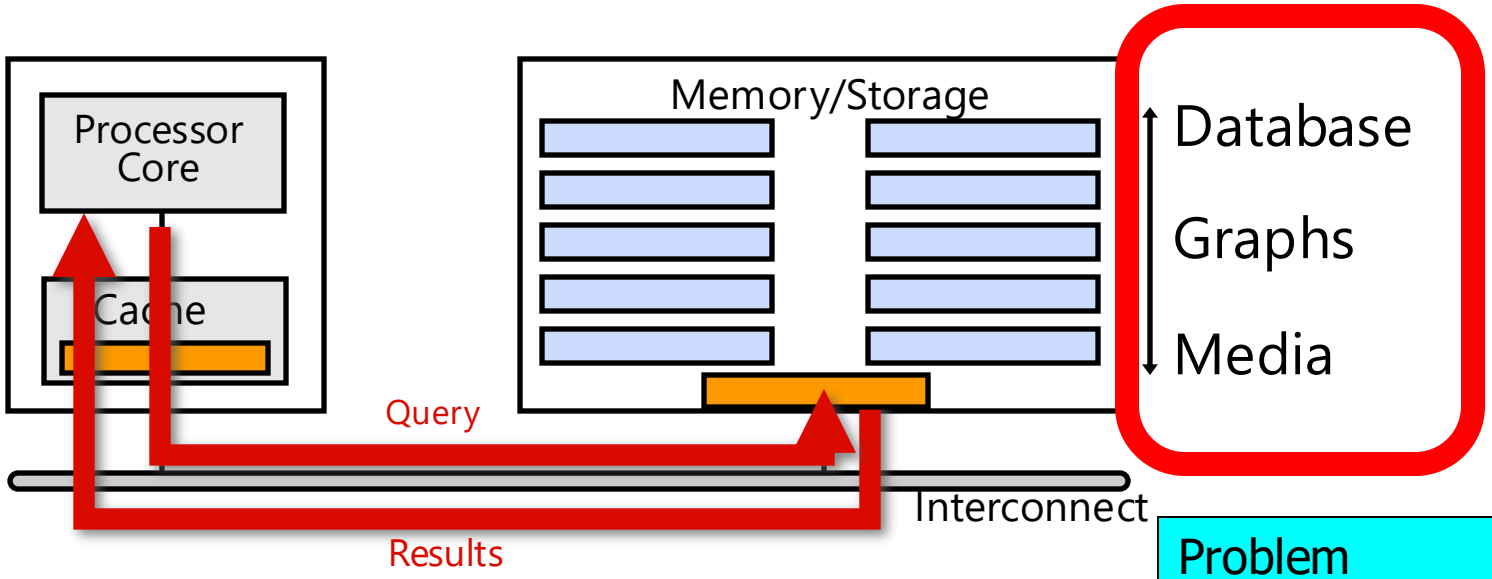
Week	Date	Livestream	Lecture	Readings	Lab	HW
W1	30.09 Thu.	Live	L1: Introduction and Basics (PDF) (PPT)	Required Mentioned	Lab 1 Out	HW 0 Out
	01.10 Fri.	Live	L2: Trends, Tradeoffs and Design Fundamentals (PDF) (PPT)	Required Mentioned		
W2	07.10 Thu.	Live	L3a: Memory Systems: Challenges and Opportunities (PDF) (PPT)	Described Suggested		HW 1 Out
			L3b: Course Info & Logistics (PDF) (PPT)			
			L3c: Memory Performance Attacks (PDF) (PPT)	Described Suggested		
	08.10 Fri.	Live	L4a: Memory Performance Attacks (PDF) (PPT)	Described Suggested	Lab 2 Out	
			L4b: Data Retention and Memory Refresh (PDF) (PPT)	Described Suggested		
			L4c: RowHammer (PDF) (PPT)	Described Suggested		

Two Major Directions

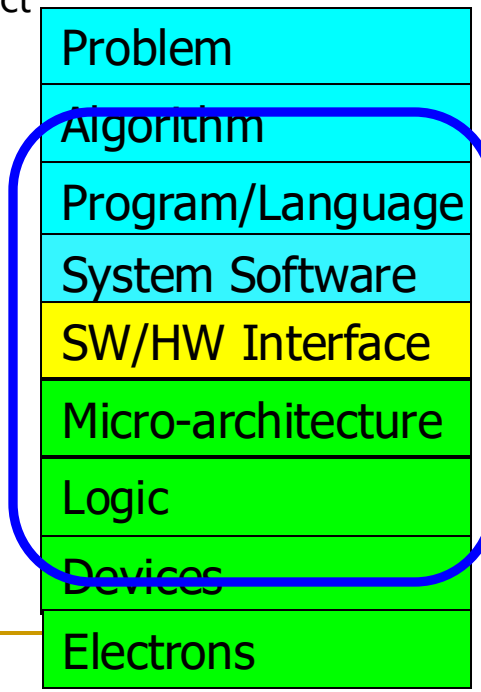
1. Memory system design for AI/ML workloads/accelerators

2. AI/ML techniques for improving memory system designs

Goal: Processing Inside Memory/Storage



- Many questions … How do we design the:
  - compute-capable memory & controllers?
  - processors & communication units?
  - software & hardware interfaces?
  - system software, compilers, languages?
  - algorithms & theoretical foundations?



Why In-Memory Computation Today?

- **Huge demand from Applications & Systems**
  - Data access bottleneck
  - Energy & power bottlenecks
  - Data movement energy dominates computation energy
  - Need all at the same time: performance, energy, sustainability
  - We can improve all metrics by minimizing data movement

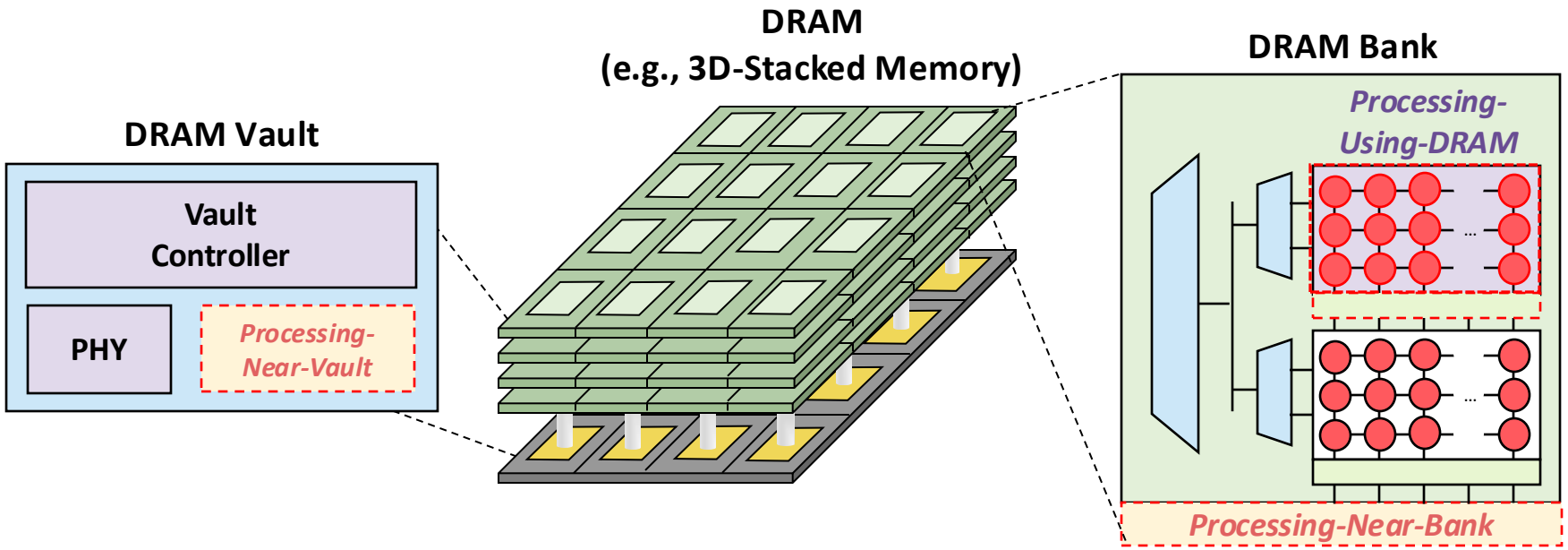
- **Huge problems with Memory Technology**
  - Memory technology scaling is not going well (e.g., RowHammer)
  - Scaling issues demand intelligence in memory + new technology

- **Designs are squeezed in the middle**

Processing-in-Memory:
Nature of Computation

Two main approaches for Processing-in-Memory:

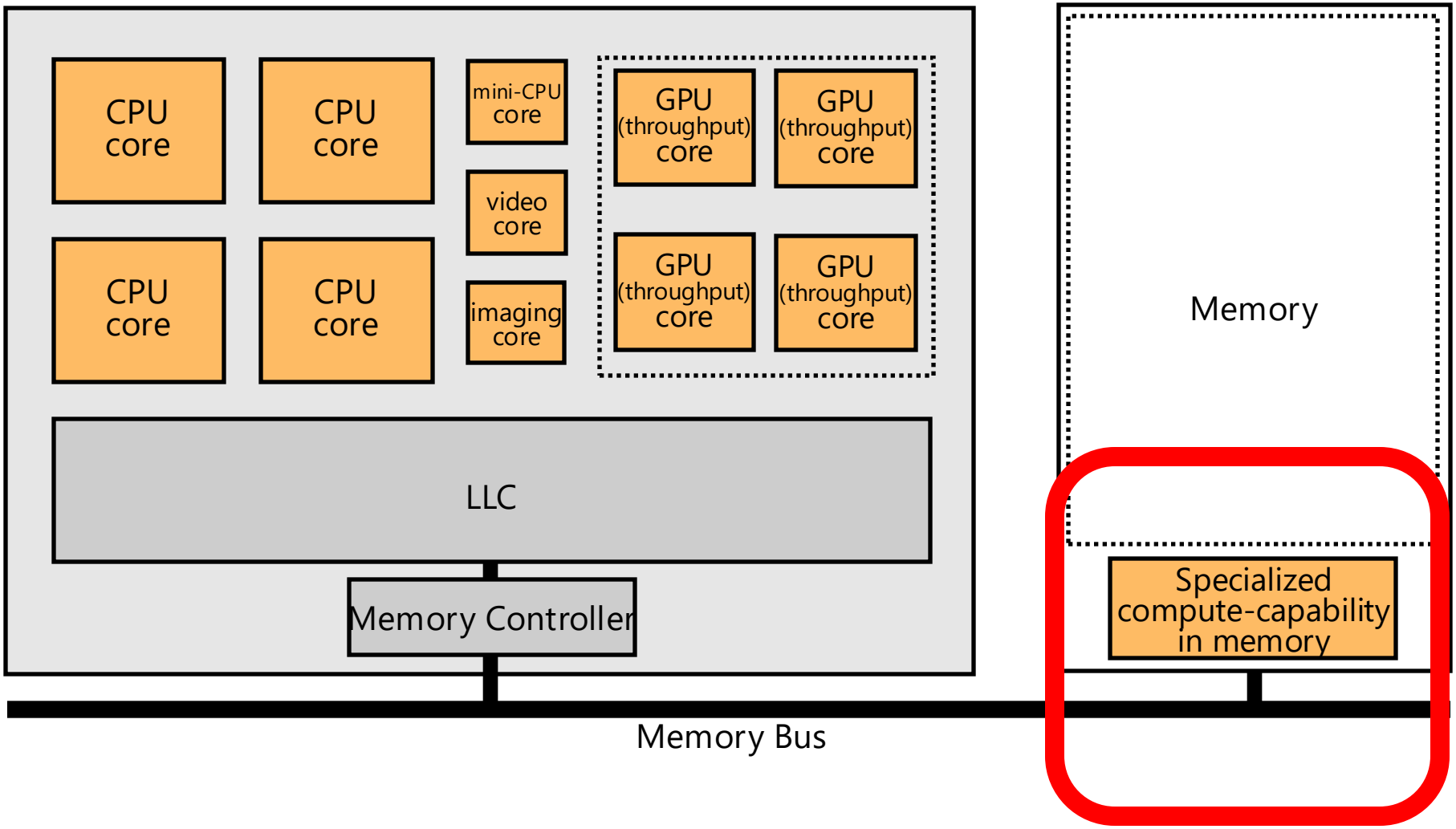
1. **Processing-Near-Memory**: Design compute logic and memory separately (as today) and integrate logic closer to memory
2. **Processing-Using-Memory**: Use analog operational principles of memory circuitry to perform computation (no compute logic)



A PIM Taxonomy

- **Nature** (of computation)
  - Using: Use operational properties of memory structures
  - Near: Add logic close to memory structures
- **Technology**
  - Flash, DRAM, SRAM, RRAM, MRAM, FeRAM, PCM, 3D, …
- **Location**
  - Sensor, Cold Storage, Hard Disk, SSD, Main Memory, Cache, Register File, Memory Controller, Interconnect, …
- A tuple of the three determines "PIM type"
- One can combine multiple "PIM types" in a system

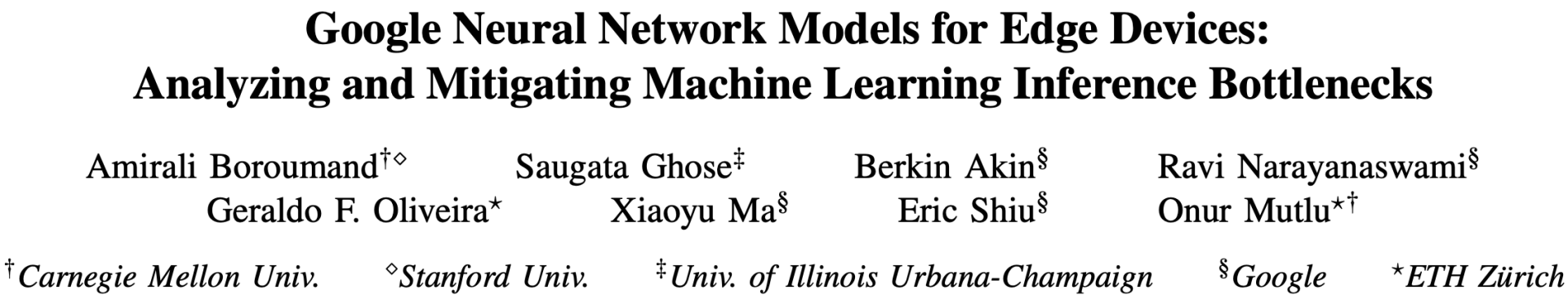
Mindset: Memory as an Accelerator



Memory similar to a "conventional" accelerator

Accelerating Neural Network Inference

- Amirali Boroumand, Saugata Ghose, Berkin Akin, Ravi Narayanaswami, Geraldo F. Oliveira, Xiaoyu Ma, Eric Shiu, and Onur Mutlu,
  "Google Neural Network Models for Edge Devices: Analyzing and Mitigating Machine Learning Inference Bottlenecks"
  *Proceedings of the 30th International Conference on Parallel Architectures and Compilation Techniques (**PACT**)*, Virtual, September 2021.
  [Slides (pptx) (pdf)]
  [Talk Video (14 minutes)]



Google Neural Network Models for Edge Devices: Analyzing and Mitigating Machine Learning Inference Bottlenecks

Amirali Boroumand[†◇] Saugata Ghose[‡] Berkin Akin[§] Ravi Narayanaswami[§]
Geraldo F. Oliveira[⋆] Xiaoyu Ma[§] Eric Shiu[§] Onur Mutlu[⋆†]

[†]*Carnegie Mellon Univ.* [◇]*Stanford Univ.* [‡]*Univ. of Illinois Urbana-Champaign* [§]*Google* [⋆]*ETH Zürich*

Google Neural Network Models for Edge Devices: Analyzing and Mitigating Machine Learning Inference Bottlenecks

Amirali Boroumand Saugata Ghose Berkin Akin

Ravi Narayanaswami Geraldo F. Oliveira Xiaoyu Ma

Eric Shiu Onur Mutlu

PACT 2021

SAFARI

Carnegie Mellon UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN Google ETH zürich

Executive Summary

Context: We extensively analyze a state-of-the-art edge ML accelerator (Google Edge TPU) using 24 Google edge models

- Wide range of models (CNNs, LSTMs, Transducers, RCNNs)

Problem: The Edge TPU accelerator suffers from three challenges:

- It operates **significantly below** its peak throughput
- It operates **significantly below** its theoretical energy efficiency
- It **inefficiently** handles memory accesses

Key Insight: These shortcomings arise from the monolithic design of the Edge TPU accelerator

- The Edge TPU accelerator design does not account for **layer heterogeneity**

Key Mechanism: A new framework called Mensa

- Mensa consists of heterogeneous accelerators whose dataflow and hardware are specialized for specific families of layers

Key Results: We design a version of Mensa for Google edge ML models

- Mensa improves performance and energy by **3.0X** and **3.1X**
- Mensa reduces cost and improves area efficiency

Google Edge Neural Network Models

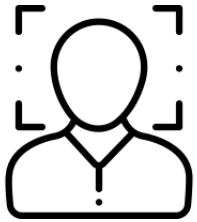
We analyze inference execution using 24 edge NN models

Speech Recognition

6 RNN Transducers

2 LSTMs

Language Translation



Face Detection

13 CNN

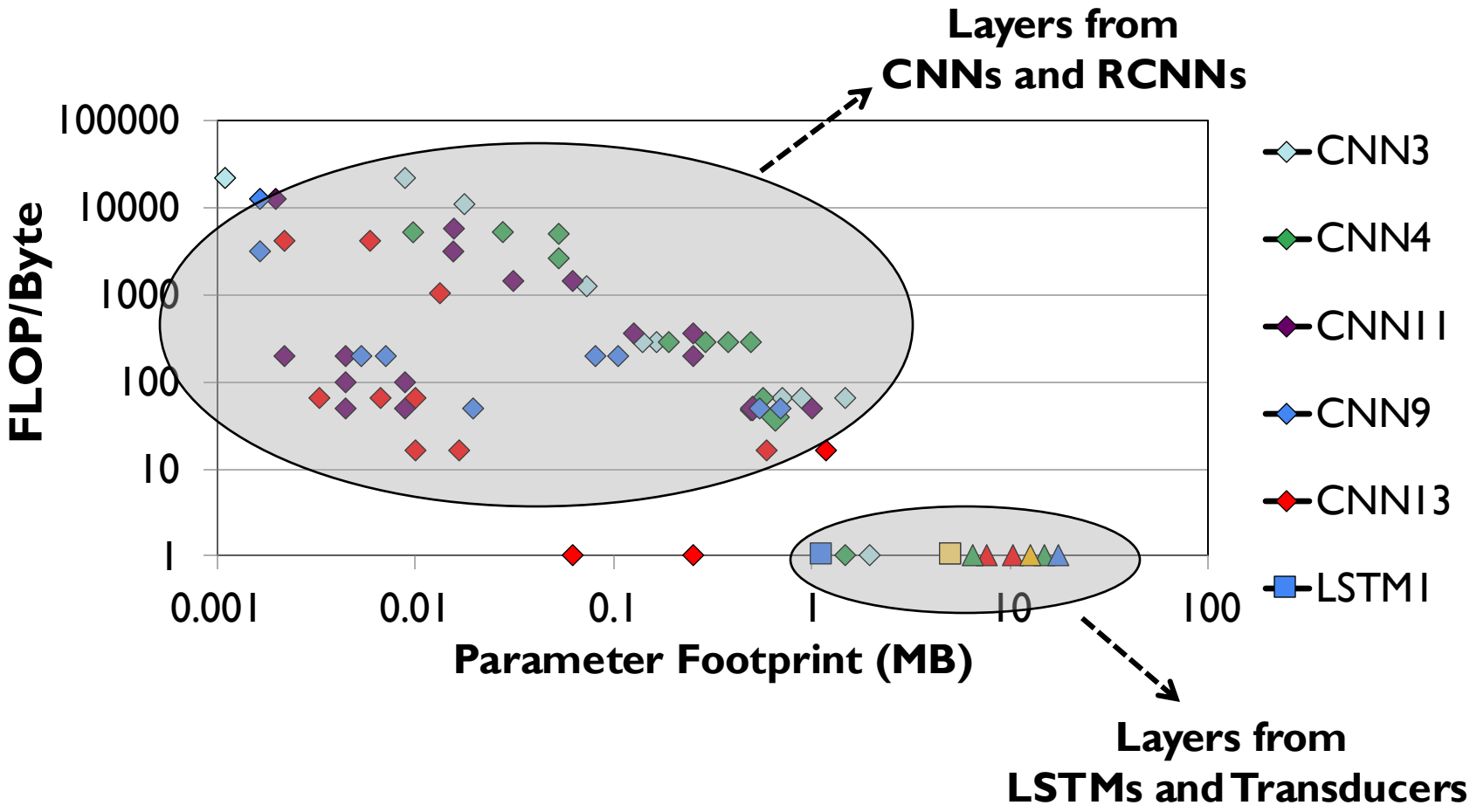
Google Edge TPU

3 RCNN

Image Captioning

Diversity Across the Models

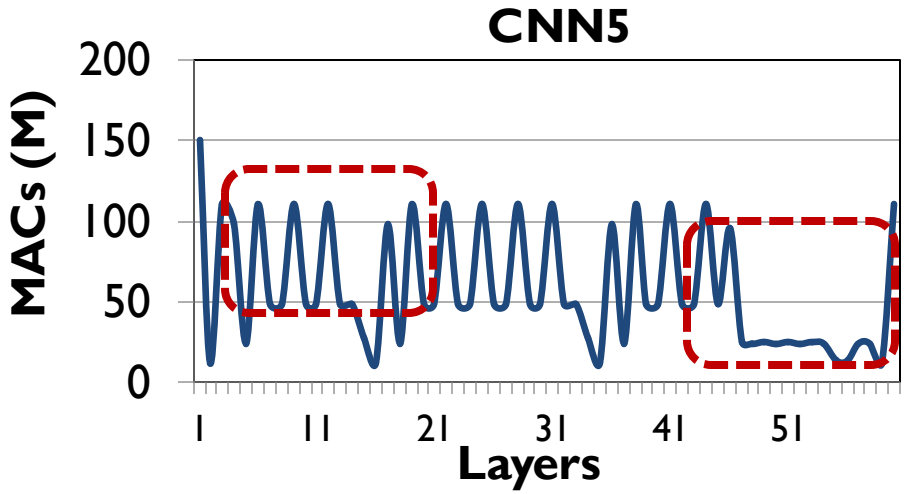
Insight 1: there is significant variation in terms of layer characteristics across the models

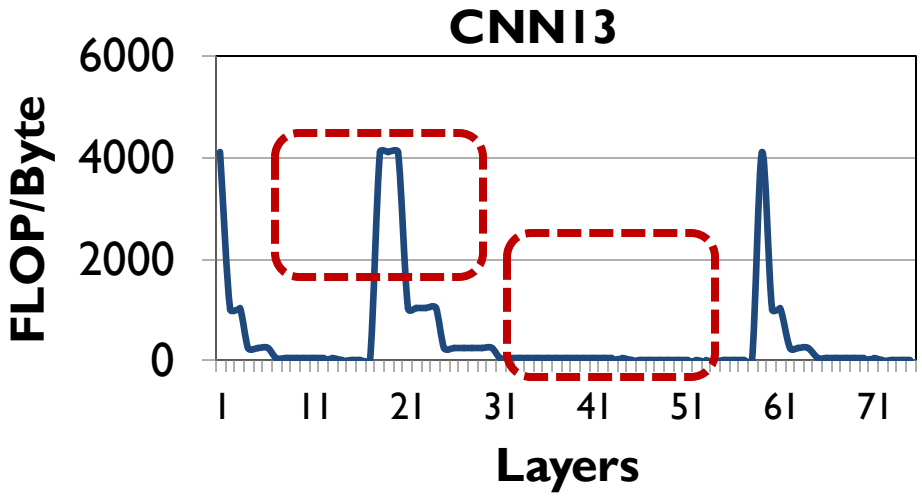


Diversity Within the Models

Insight 2: even within each model, layers exhibit significant variation in terms of layer characteristics

For example, our analysis of edge CNN models shows:



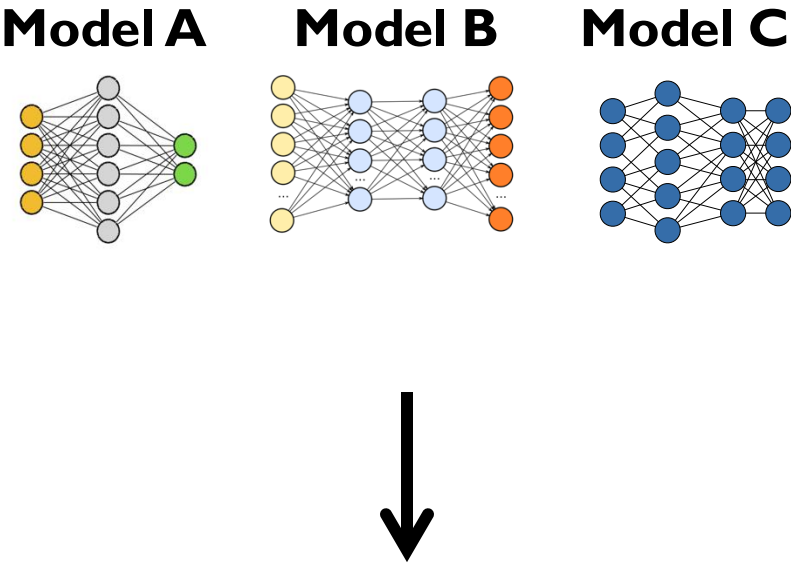


Variation in MAC intensity: up to 200x across layers

Variation in FLOP/Byte: up to 244x across layers

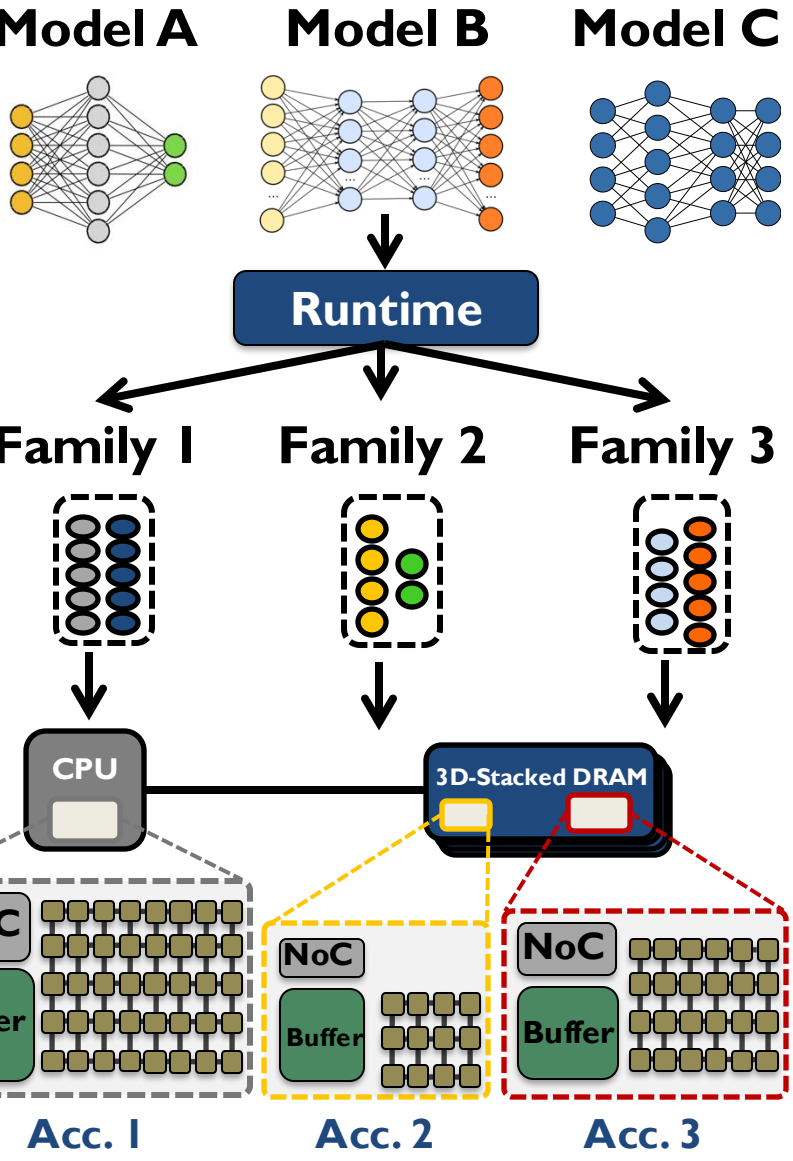
Mensa High-Level Overview

Edge TPU Accelerator



Monolithic Accelerator

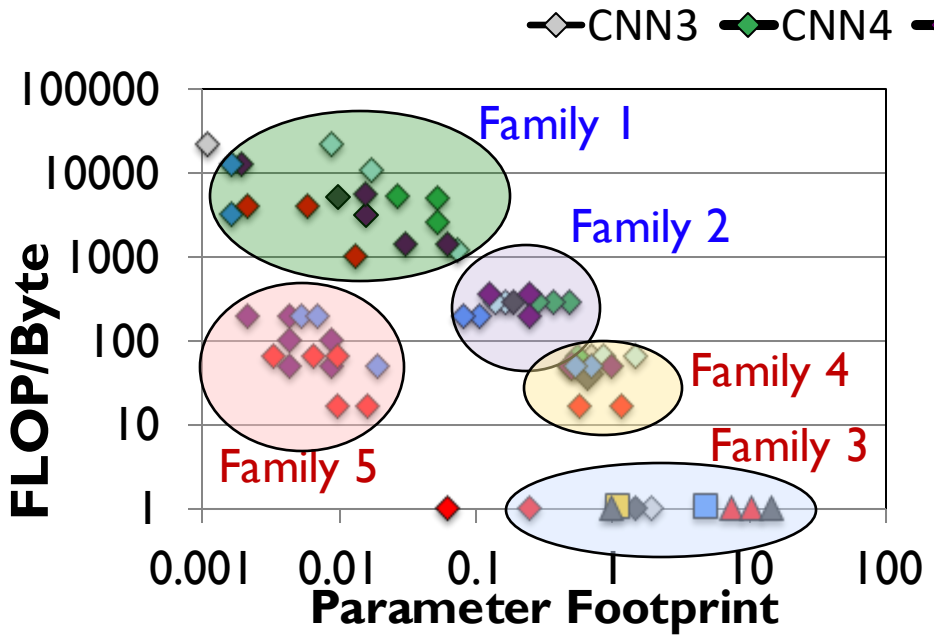
Mensa

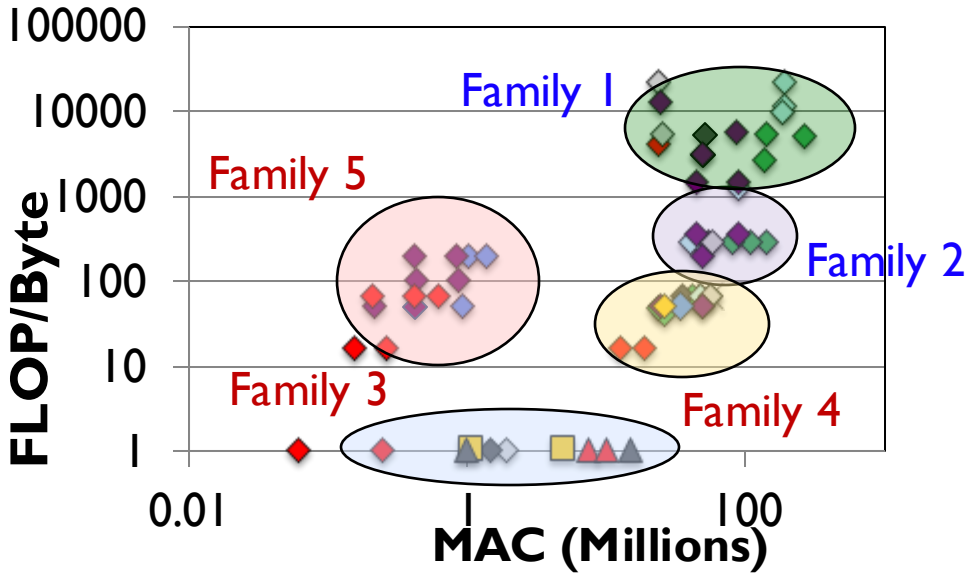


Heterogeneous Accelerators

Identifying Layer Families

Key observation: the majority of layers group into a small number of layer families

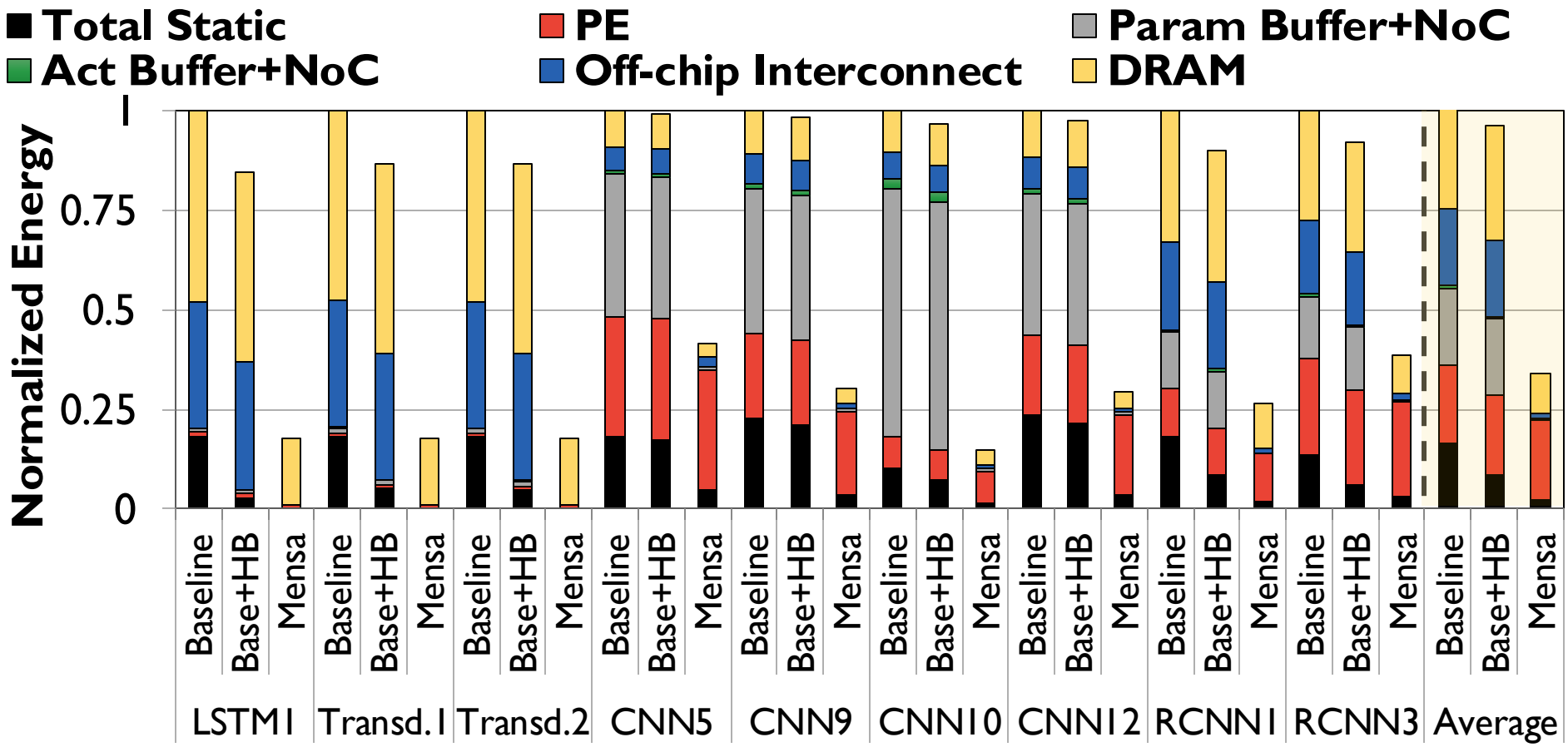




Families 1 & 2: low parameter footprint, high data reuse and MAC intensity
→ compute-centric layers

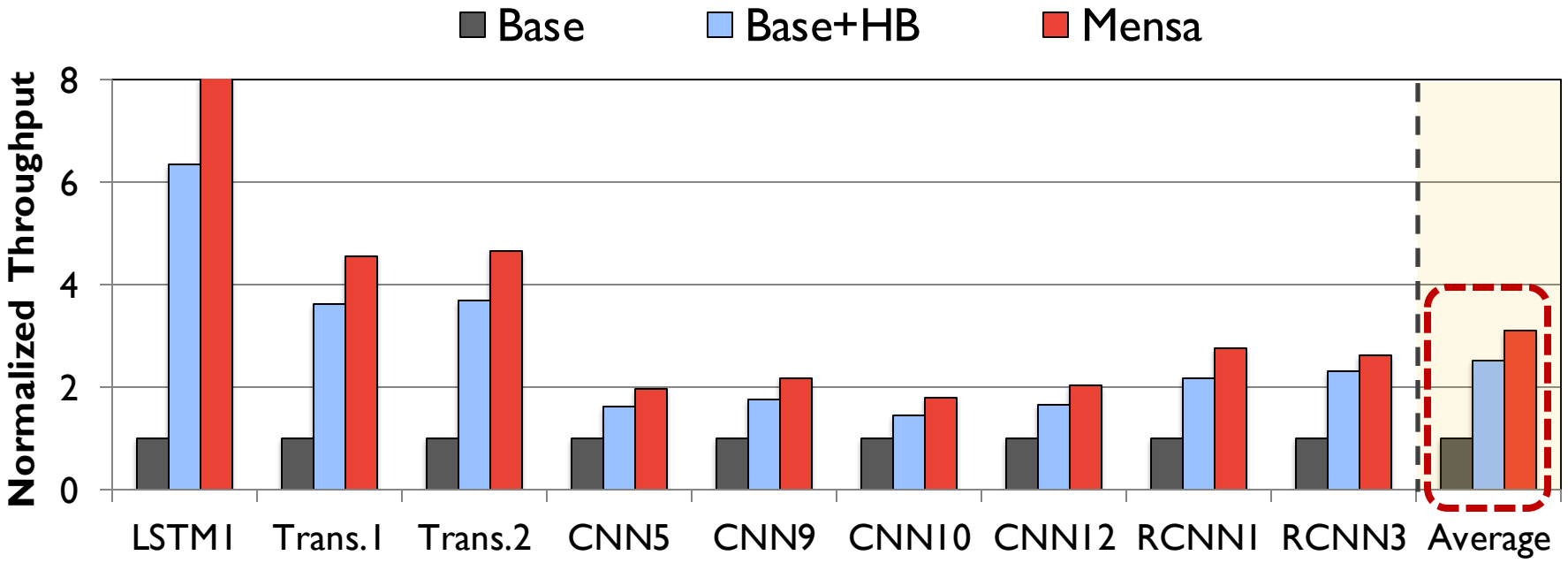
Families 3, 4 & 5: high parameter footprint, low data reuse and MAC intensity
→ data-centric layers

Mensa: Energy Reduction



Mensa-G reduces energy consumption by 3.0X compared to the baseline Edge TPU

Mensa: Throughput Improvement



Mensa-G improves inference throughput by 3.1X compared to the baseline Edge TPU

Mensa: Highly-Efficient ML Inference

- Amirali Boroumand, Saugata Ghose, Berkin Akin, Ravi Narayanaswami, Geraldo F. Oliveira, Xiaoyu Ma, Eric Shiu, and Onur Mutlu,
  "Google Neural Network Models for Edge Devices: Analyzing and Mitigating Machine Learning Inference Bottlenecks"
  *Proceedings of the 30th International Conference on Parallel Architectures and Compilation Techniques (**PACT**)*, Virtual, September 2021.
  [Slides (pptx) (pdf)]
  [Talk Video (14 minutes)]

Google Neural Network Models for Edge Devices: Analyzing and Mitigating Machine Learning Inference Bottlenecks

Amirali Boroumand[†◇] Saugata Ghose[‡] Berkin Akin[§] Ravi Narayanaswami[§]
Geraldo F. Oliveira[⋆] Xiaoyu Ma[§] Eric Shiu[§] Onur Mutlu[⋆†]

[†]*Carnegie Mellon Univ.* [◇]*Stanford Univ.* [‡]*Univ. of Illinois Urbana-Champaign* [§]*Google* [⋆]*ETH Zürich*

Accelerating In-Memory Graph Analytics

- Large graphs are everywhere (circa 2015)

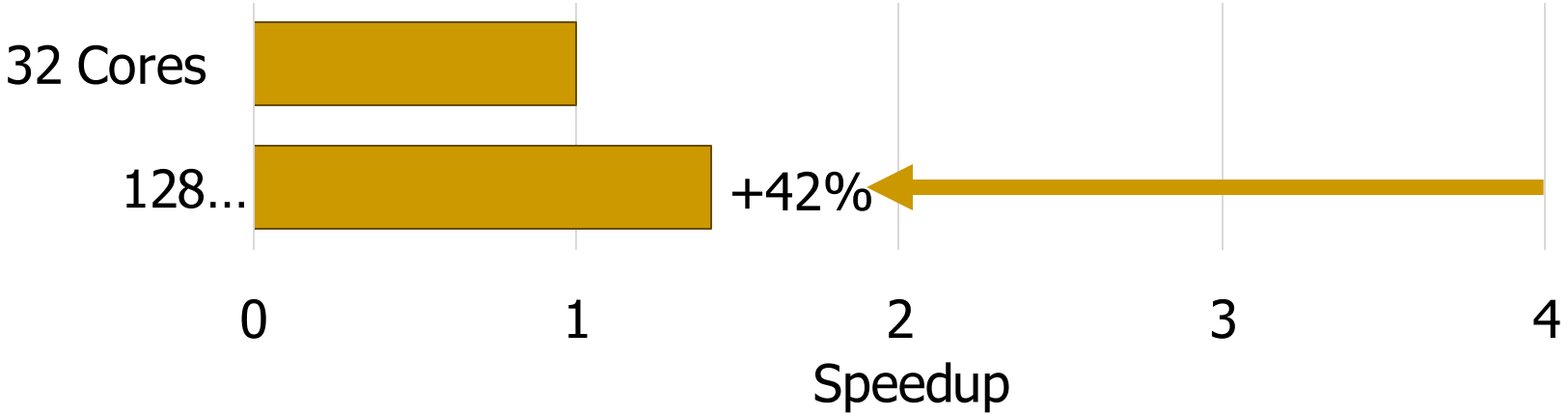
36 Million
Wikipedia Pages

1.4 Billion
Facebook Users

300 Million
Twitter Users

30 Billion
Instagram Photos

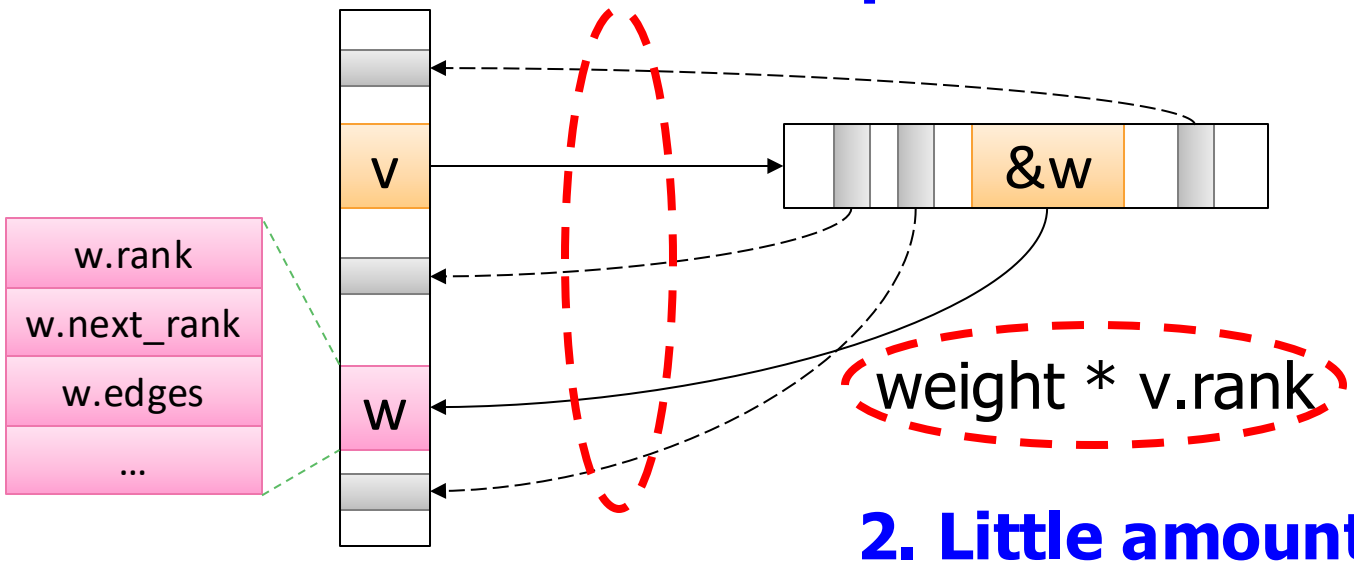
- Scalable large-scale graph processing is challenging



Key Bottlenecks in Graph Processing

```
for (v: graph.vertices) {
    for (w: v.successors) {
        w.next_rank += weight * v.rank;
    }
}
```

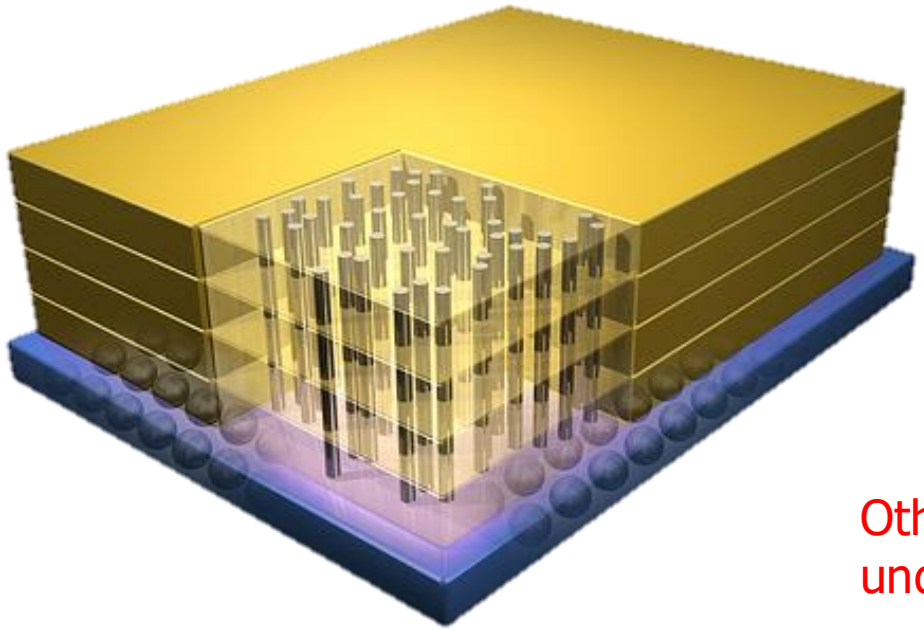
1. Frequent random memory accesses



2. Little amount of computation

Opportunity: 3D-Stacked Logic+Memory

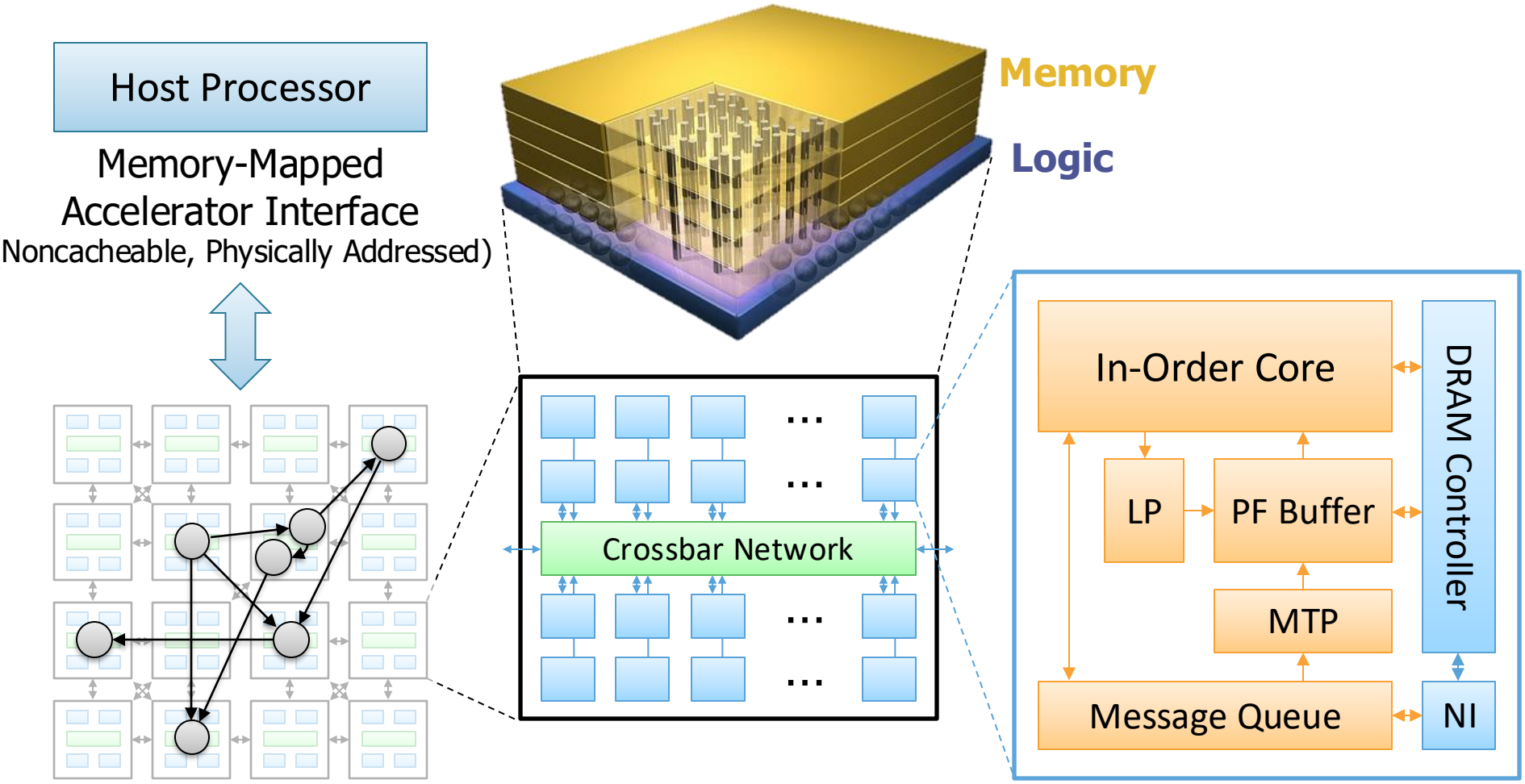
Hybrid Memory Cube
CONSORTIUM



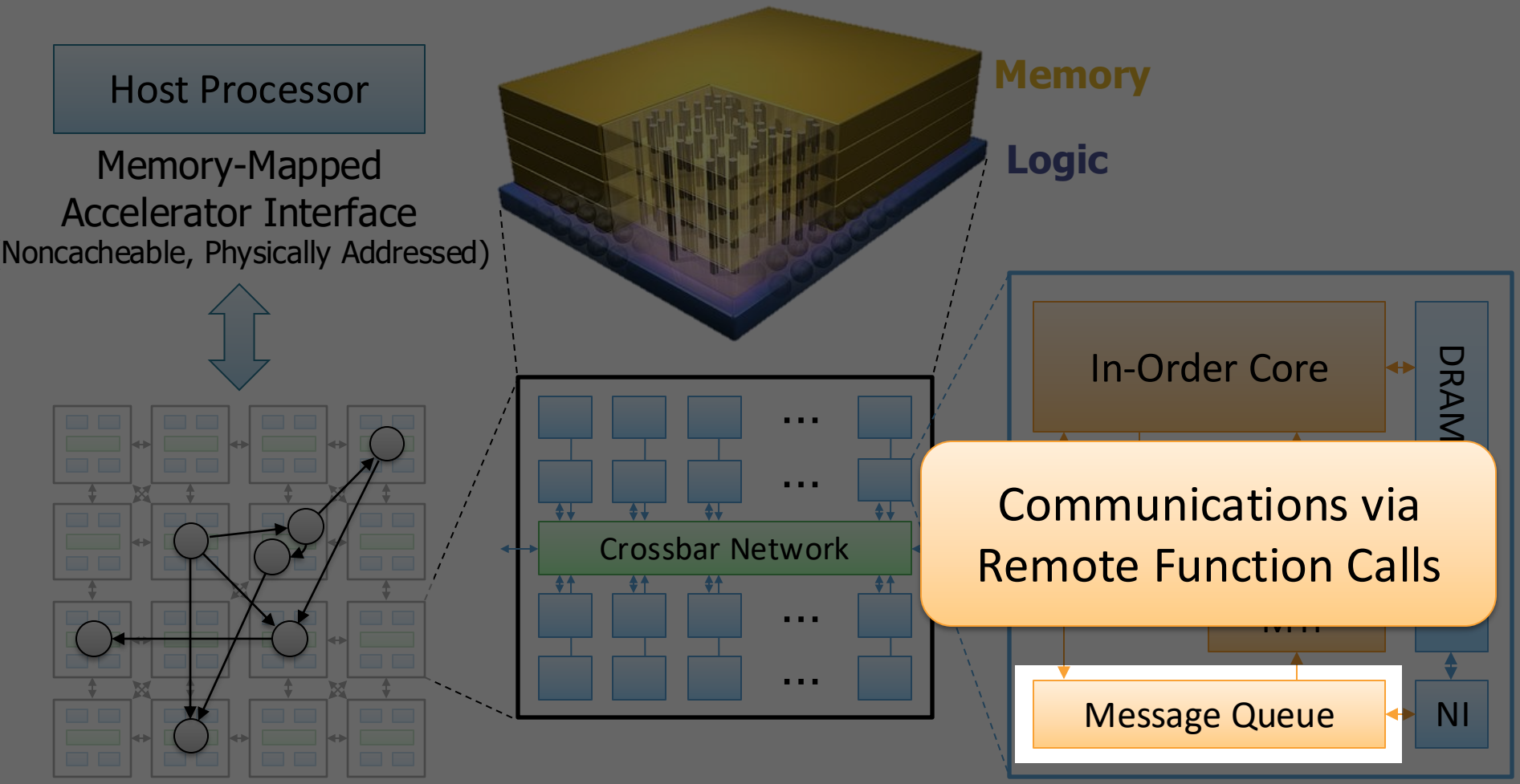
Other "True 3D" technologies under development

Tesseract System for Graph Processing

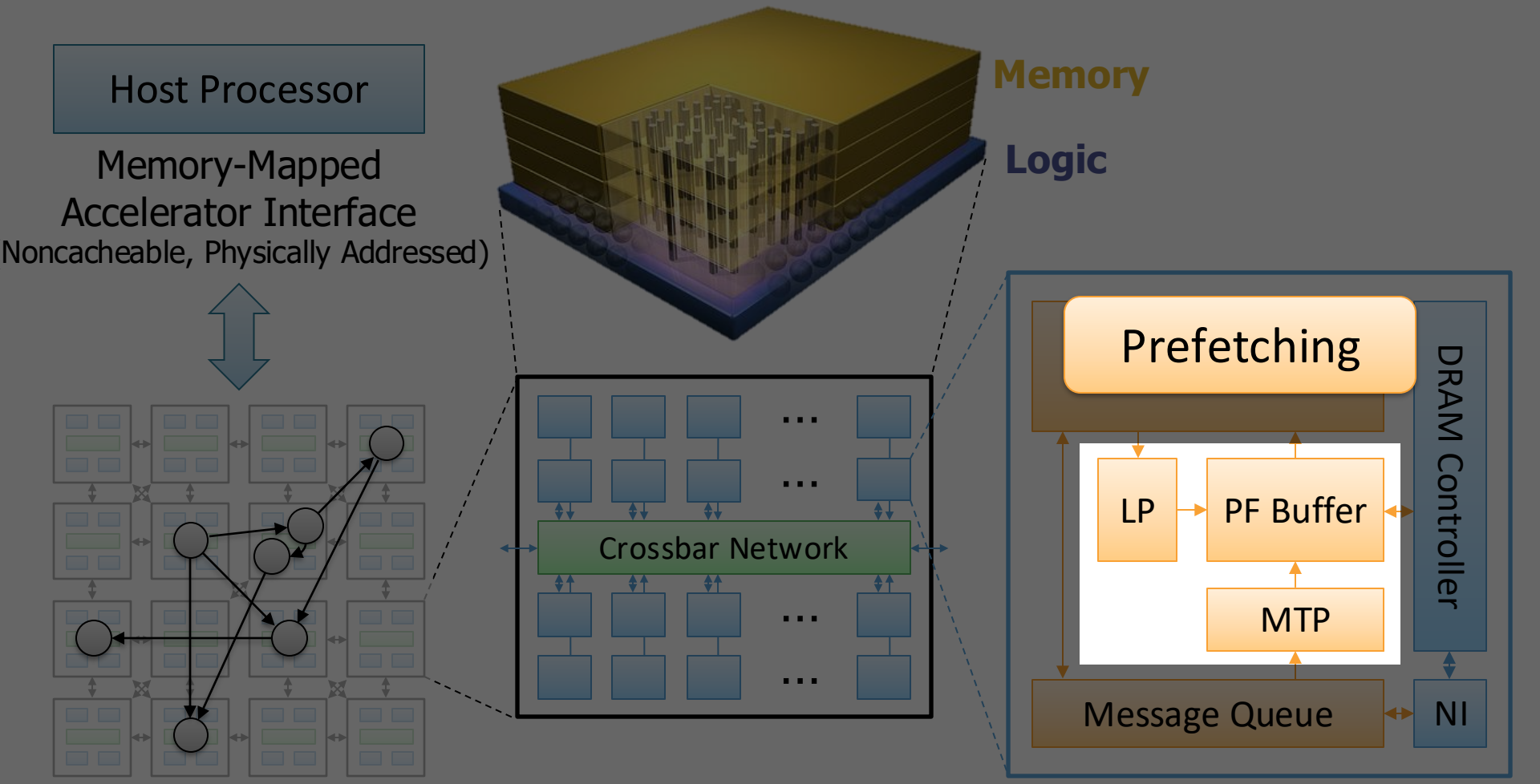
Interconnected set of 3D-stacked memory+logic chips with simple cores



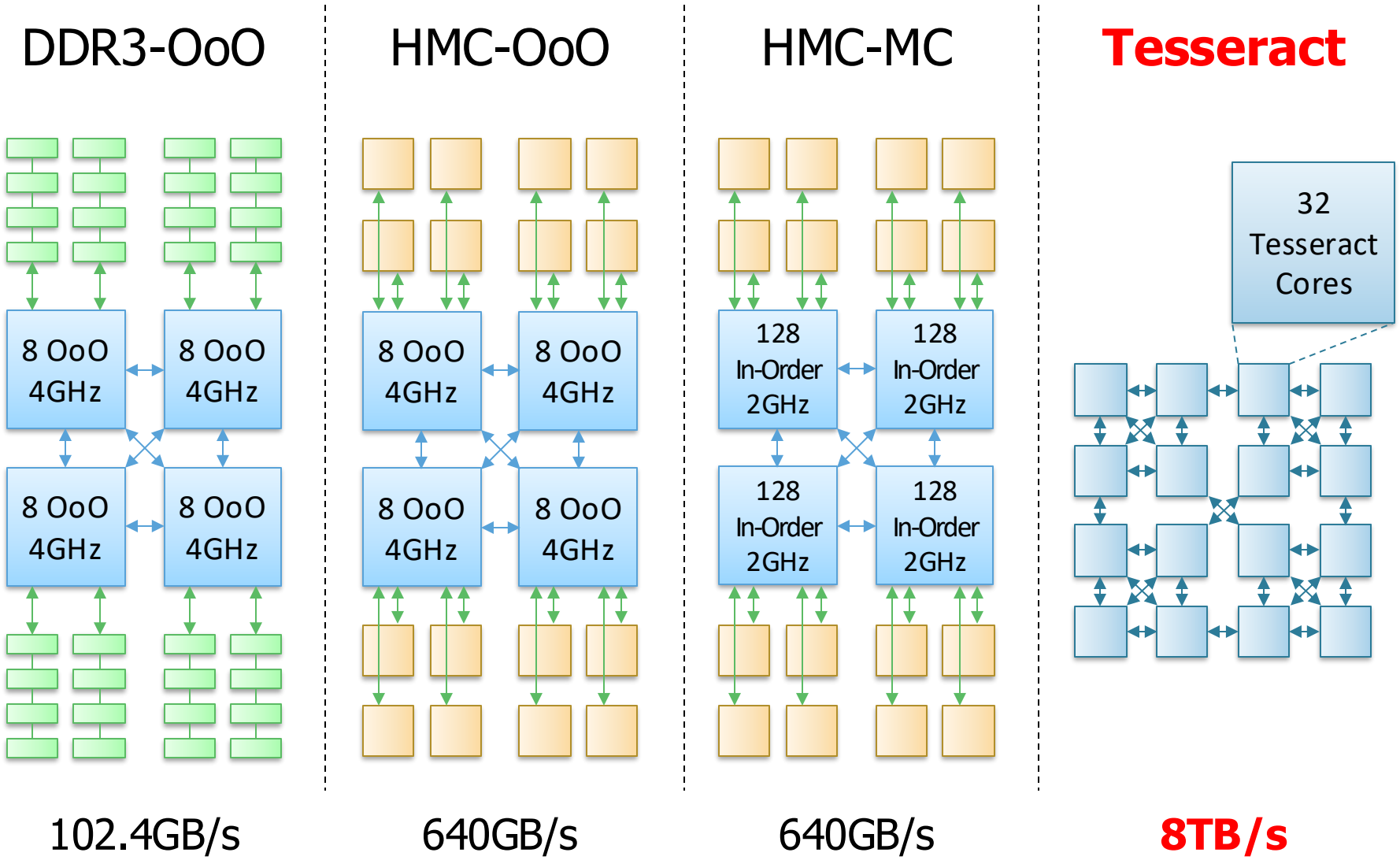
Tesseract System for Graph Processing



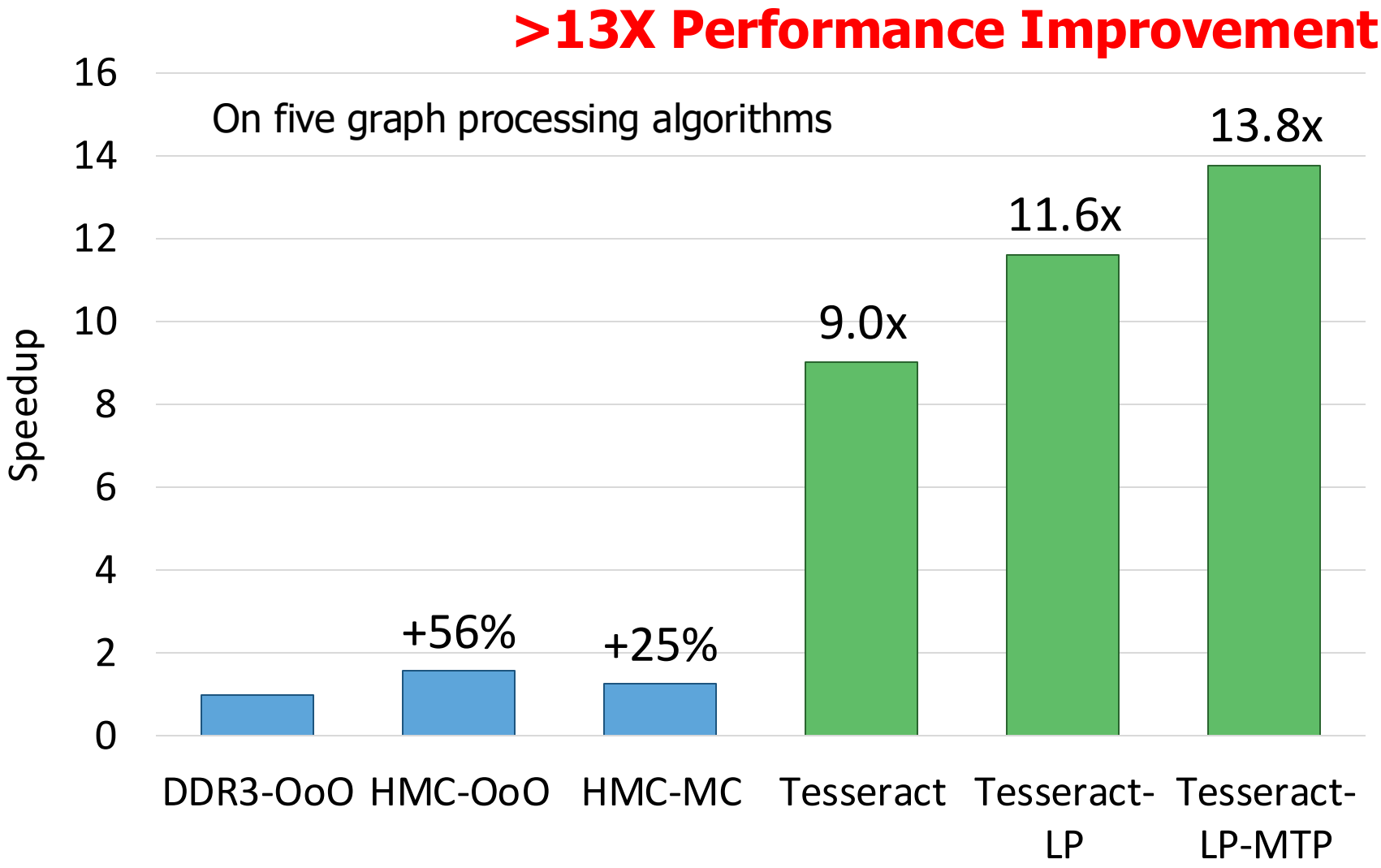
Tesseract System for Graph Processing



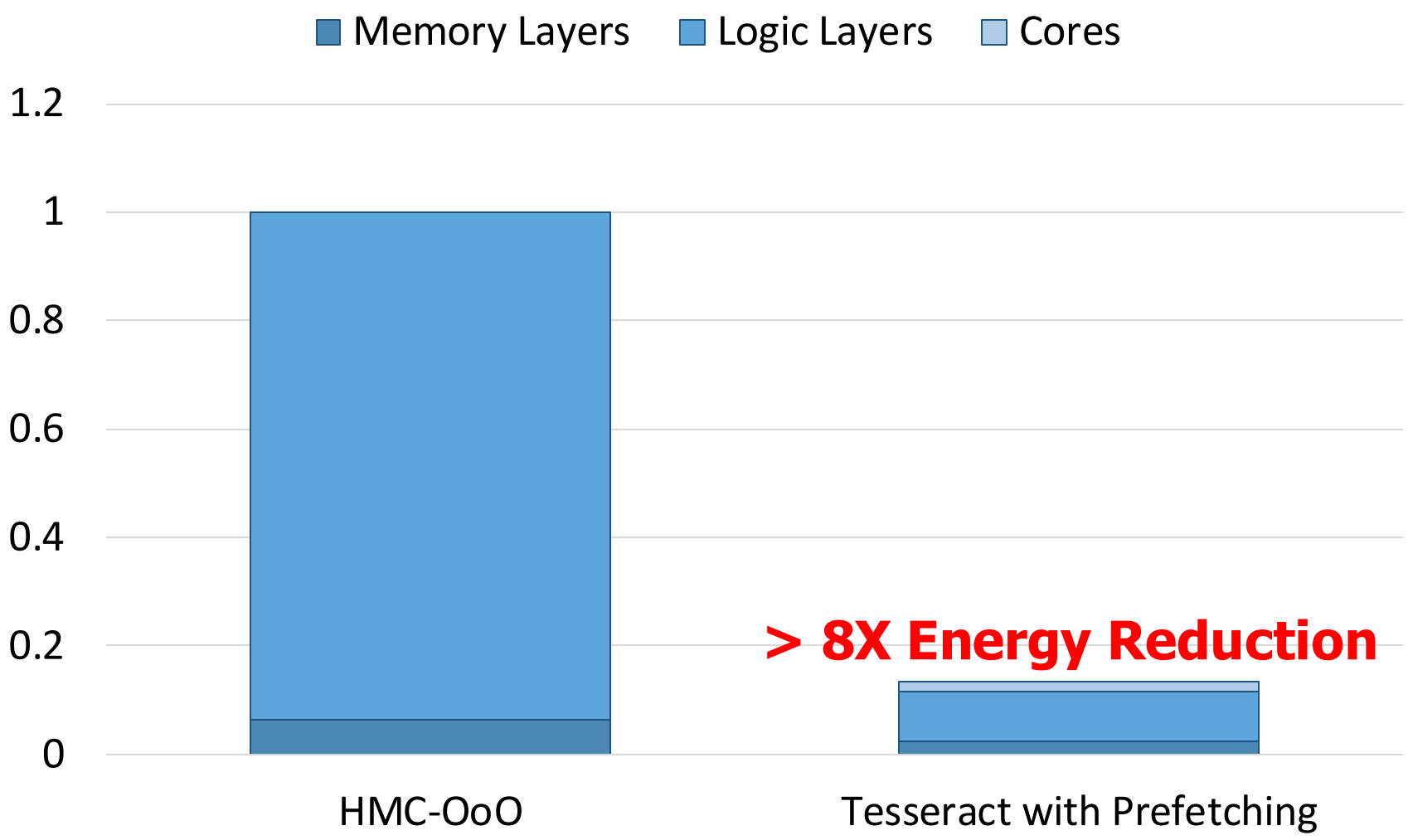
Evaluated Systems



Tesseract Graph Processing Performance



Tesseract Graph Processing System Energy



More on Tesseract

- Junwhan Ahn, Sungpack Hong, Sungjoo Yoo, Onur Mutlu, and Kiyoung Choi,
  "A Scalable Processing-in-Memory Accelerator for Parallel Graph Processing"
  *Proceedings of the 42nd International Symposium on Computer Architecture (**ISCA**)*, Portland, OR, June 2015.
  [Slides (pptx) (pdf)] [Lightning Session Slides (pptx) (pdf)]
  Top Picks Honorable Mention by IEEE Micro.
  Selected to the ISCA-50 25-Year Retrospective Issue covering 1996-2020 in 2023 (Retrospective (pdf) Full Issue).

A Scalable Processing-in-Memory Accelerator for Parallel Graph Processing

Junwhan Ahn Sungpack Hong[§] Sungjoo Yoo Onur Mutlu[†] Kiyoung Choi
junwhan@snu.ac.kr, sungpack.hong@oracle.com, sungjoo.yoo@gmail.com, onur@cmu.edu, kchoi@snu.ac.kr
Seoul National University [§]Oracle Labs [†]Carnegie Mellon University

A Short Retrospective @ 50 Years of ISCA

Retrospective: A Scalable Processing-in-Memory Accelerator for Parallel Graph Processing

Junwhan Ahn[†] Sungpack Hong[‡] Sungjoo Yoo[▽] Onur Mutlu[§] Kiyoung Choi[▽]
[†]*Google DeepMind* [‡]*Oracle Labs* [§]*ETH Zürich* [▽]*Seoul National University*

***Abstract*—Our ISCA 2015 paper [1] provides a new programmable processing-in-memory (PIM) architecture and system design that can accelerate key data-intensive applications, with a focus on graph processing workloads. Our major idea was to completely rethink the system, including the programming model, data partitioning mechanisms, system support, instruction set architecture, along with near-memory execution units and their communication architecture, such that an important workload can be accelerated at a maximum level using a distributed system of well-connected near-memory accelerators. We built our accelerator system, Tesseract, using 3D-stacked memories with logic layers, where each logic layer contains general-purpose processing cores and cores communicate with each other using a message-passing programming model. Cores could be specialized for graph processing (or any other application to be accelerated).**

To our knowledge, our paper was the first to completely design a near-memory accelerator system from scratch such that it is both generally programmable and specifically customizable to accelerate important applications, with a case study on major graph processing workloads. Ensuing work in academia and industry showed that similar approaches to system design can greatly benefit both graph processing workloads and other applications, such as machine learning, for which ideas from Tesseract seem to have been influential.

This short retrospective provides a brief analysis of our ISCA 2015 paper and its impact. We briefly describe the major ideas and contributions of the work, discuss later works that built on it or were influenced by it, and make some educated guesses on what the future may bring on PIM and accelerator systems.

I. Background, Approach & Mindset

We started our research when 3D-stacked memories (e.g., [2–4]) were viable and seemed to have promise for building effective and practical processing-near-memory systems. Such near-memory systems could lead to improvements, but there was little to no research that examined how an accelerator could be completely (re-)designed using such near-memory technology, from its hardware architecture to its programming model and software system, and what the performance and energy benefits could be of such a re-design. We set out to answer these questions in our ISCA 2015 paper [1].

We followed several major principles to design our accelerator from the ground up. We believe these principles are still important: a major contribution and influence of our work was in putting all of these together in a cohesive full-system design and demonstrating the large performance and energy benefits that can be obtained from such a design. We see a similar approach in many modern large-scale accelerator systems in machine learning today (e.g., [5–9]). Our principles are:

1. *Near-memory execution* to enable/exploit the high data access bandwidth modern workloads (e.g., graph processing) need and to reduce data movement and access latency.
2. *General programmability* so that the system can be easily adopted, extended, and customized for many workloads.
3. *Maximal acceleration capability* to maximize the performance and energy benefits. We set ourselves free from backward compatibility and cost constraints. We aimed to completely re-design the system stack. Our goal was to explore the maximal performance and energy efficiency benefits we can gain from a near-memory accelerator if we had complete freedom to change things as much as we needed. We contrast this approach to the *minimal intrusion* approach we also explored in a separate ISCA 2015 paper [10].
4. *Customizable to specific workloads*, such that we can maximize acceleration benefits. Our focus workload was graph analytics/processing, a key workload at the time and today. However, our design principles are not limited to graph processing and the system we built is customizable to other workloads as well, e.g., machine learning, genome analysis.
5. *Memory-capacity-proportional performance*, i.e., processing capability should proportionally grow (i.e., scale) as memory capacity increases and vice versa. This enables scaling of data-intensive workloads that need both memory and compute.
6. *Exploit new technology (3D stacking)* that enables tight integration of memory and logic and helps multiple above principles (e.g., enables customizable near-memory acceleration capability in the logic layer of a 3D-stacked memory chip).
7. *Good communication and scaling capability* to support scalability to large dataset sizes and to enable memory-capacity-proportional performance. To this end, we provided scalable communication mechanisms between execution cores and carefully interconnected small accelerator chips to form a large distributed system of accelerator chips.
8. *Maximal and efficient use of memory bandwidth* to supply the high-bandwidth data access that modern workloads need. To this end, we introduced new, specialized mechanisms for prefetching and a programming model that helps leverage application semantics for hardware optimization.

II. Contributions and Influence

We believe the major contributions of our work were 1) complete rethinking of how an accelerator system should be designed to enable maximal acceleration capability, and 2) the design and analysis of such an accelerator with this mindset and using the aforementioned principles to demonstrate its effectiveness in an important class of workloads.

One can find examples of our approach in modern large-scale machine learning (ML) accelerators, which are perhaps the most successful incarnation of scalable near-memory execution architectures. ML infrastructure today (e.g., [5–9]) consists of accelerator chips, each containing compute units and high-bandwidth memory tightly packaged together, and features scale-up capability enabled by connecting thousands of such chips with high-bandwidth interconnection links. The system-wide rethinking that was done to enable such accelerators and many of the principles used in such accelerators resemble our ISCA 2015 paper's approach.

The "memory-capacity-proportional performance" principle we explored in the paper shares similarities with how ML workloads are scaled up today. Similar to how we carefully sharded graphs across our accelerator chips to greatly improve effective memory bandwidth in our paper, today's ML workloads are sharded across a large number of accelerators by leveraging data/model parallelism and optimizing the placement to balance communication overheads and compute scalability [11, 12]. With the advent of large generative models requiring high memory bandwidth for fast training and inference, the scaling behavior where capacity and bandwidth are scaled together has become an essential architectural property to support modern data-intensive workloads.

The "maximal acceleration capability" principle we used in Tesseract provides much larger performance and energy improvements and better customization than the "minimalist" approach that our other ISCA 2015 paper on *PIM-Enabled Instructions* [10] explored: "minimally change" an existing system to incorporate (near-memory) acceleration capability to ease programming and keep costs low. So far, the industry has more widely adopted the maximal approach to overcome the pressing scaling bottlenecks of major workloads. The key enabler that bridges the programmability gap between the maximal approach favoring large performance & energy benefits and the minimal approach favoring ease of programming is compilation techniques. These techniques lower well-defined high-level constructs into lower-level primitives [12, 13]; our ISCA 2015 papers [1, 10] and a follow-up work [14] explore them lightly. We believe that a good programming model that enables large benefits coupled with support for it across the entire system stack (including compilers & hardware) will continue to be important for effective near-memory system and accelerator designs [14]. We also believe that the maximal versus minimal approaches that are initially explored in our two ISCA 2015 papers is a useful way of exploring emerging technologies (e.g., near-memory accelerators) to better understand the tradeoffs of system designs that exploit such technologies.

III. Influence on Later Works

Our paper was at the beginning of a proliferation of scalable near-memory processing systems designed to accelerate key applications (see [15] for many works on the topic). Tesseract has inspired many near-memory system ideas (e.g., [16–28]) and served as the de facto comparison point for such systems, including near-memory graph processing accelerators that built on Tesseract and improved various aspects of Tesseract. Since machine learning accelerators that use high-bandwidth memory (e.g., [5, 29]) and industrial PIM prototypes (e.g., [30–41]) are now in the market, near-memory processing is no longer an "eccentric" architecture it used to be when Tesseract was originally published.

Graph processing & analytics workloads remain as an important and growing class of applications in various forms, ranging from large-scale industrial graph analysis engines (e.g., [42]) to graph neural networks [43]. Our focus on large-scale graph processing in our ISCA 2015 paper increased attention to this domain in the computer architecture community, resulting in subsequent research on efficient hardware architectures for graph processing (e.g., [44–46]).

IV. Summary and Future Outlook

We believe that our ISCA 2015 paper's principled rethinking of system design to accelerate an important class of data-intensive workloads provided significant value and enabled/influenced a large body of follow-on works and ideas. We expect that such rethinking of system design for key workloads, especially with a focus on "maximal acceleration capability," will continue to be critical as pressing technology and application scaling challenges increasingly require us to think differently to substantially improve performance and energy (as well as other metrics). We believe the principles exploited in Tesseract are fundamental and they will remain useful and likely become even more important as systems become more constrained due to the continuously-increasing memory access and computation demands of future workloads. We also project that as hardware substrates for near-memory acceleration (e.g., 3D stacking, in-DRAM computation, NVM-based PIM, processing using memory [15]) evolve and mature, systems will take advantage of them even more, likely using principles similar to those used in the design of Tesseract.

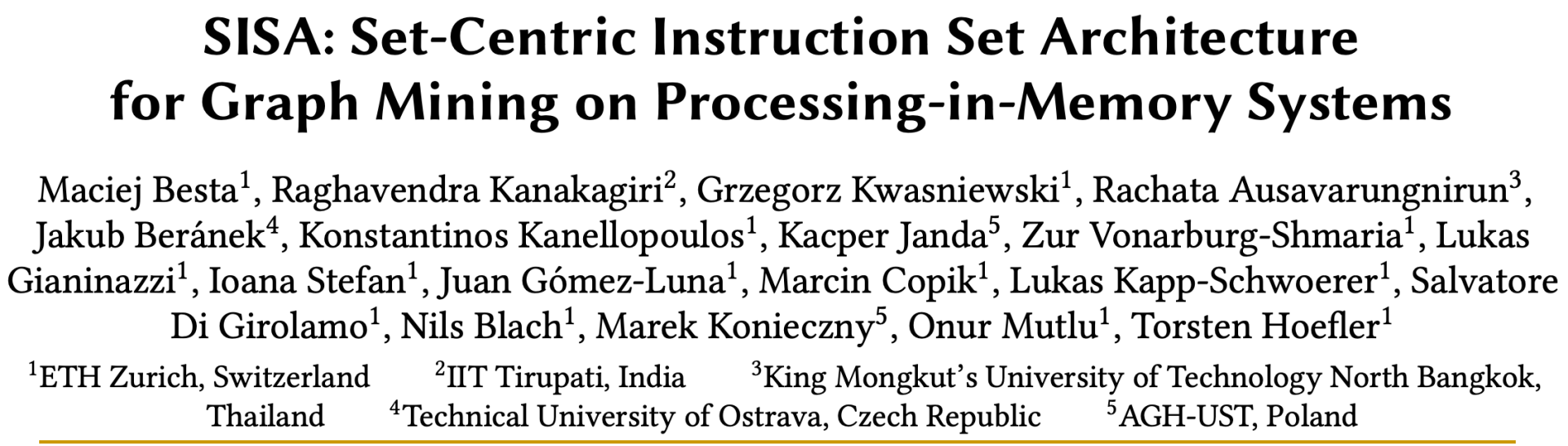
References

[1] J. Ahn *et al.*, "A Scalable Processing-in-Memory Accelerator for Parallel Graph Processing," in *ISCA*, 2015.
[2] Hybrid Memory Cube Consortium, "HMC Specification 1.1," 2013.
[3] J. Jeddeloh and B. Keeth, "Hybrid Memory Cube: New DRAM Architecture Increases Density and Performance," in *VLSIT*, 2012.
[4] JEDEC, "High Bandwidth Memory (HBM) DRAM," Standard No. JESD235, 2013.
[5] N. Jouppi *et al.*, "TPU v4: An Optically Reconfigurable Supercomputer for Machine Learning with Hardware Support for Embedding," in *ISCA*, 2023.
[6] J. Fowers *et al.*, "A Configurable Cloud-Scale DNN Processor for Real-Time AI," in *ISCA*, 2018.
[7] S. Lie, "Cerebras Architecture Deep Dive: First Look Inside the Hardware/Software Co-Design for Deep Learning," in *IEEE Micro*, 2023.
[8] E. Talpes *et al.*, "The Microarchitecture of DOJO, Tesla's Exa-Scale Computer," in *IEEE Micro*, 2023.
[9] A. Ishii and R. Wells, "NVLink-Network Switch - NVIDIA's Switch Chip for High Communication-Bandwidth SuperPODs," in *Hot Chips*, 2022.
[10] J. Ahn *et al.*, "PIM-Enabled Instructions: A Low-Overhead, Locality-Aware Processing-in-Memory Architecture," in *ISCA*, 2015.
[11] R. Pope *et al.*, "Efficiently Scaling Transformer Inference," in *MLSys*, 2023.
[12] D. Lepikhin *et al.*, "GShard: Scaling Giant Models with Conditional Computation and Automatic Sharding," in *ICLR*, 2021.
[13] S. Wang *et al.*, "Overlap Communication with Dependent Computation via Decomposition in Large Deep Learning Models," in *ASPLOS*, 2023.
[14] J. Ahn *et al.*, "AIM: Energy-Efficient Aggregation Inside the Memory Hierarchy," *ACM TACO*, vol. 13, no. 4, 2016.
[15] O. Mutlu *et al.*, "A Modern Primer on Processing in Memory," *Emerging Computing: From Devices to Systems*, 2021, https://arxiv.org/abs/2012.03112.
[16] M. Zhang *et al.*, "GraphP: Reducing Communication for PIM-Based Graph Processing with Efficient Data Partition," in *HPCA*, 2018.
[17] L. Song, "GraphR: Accelerating Graph Processing Using ReRAM," in *HPCA*, 2018.
[18] Y. Zhuo *et al.*, "GraphQ: Scalable PIM-Based Graph Processing," in *MICRO*, 2019.
[19] G. Dai *et al.*, "GraphH: A Processing-in-Memory Architecture for Large-Scale Graph Processing," *IEEE TCAD*, 2018.
[20] G. Li *et al.*, "GraphIA: An In-situ Accelerator for Large-scale Graph Processing ," in *MEMSYS*, 2018.
[21] S. Rheindt *et al.*, "NEMESYS: Near-Memory Graph Copy Enhanced System-Software," in *MEMSYS*, 2019.
[22] L. Belayneh and V. Bertacco, "GraphVine: Exploiting Multicast for Scalable Graph Analytics," in *DATE*, 2020.
[23] N. Challapalle *et al.*, "GaaS-X: Graph Analytics Accelerator Supporting Sparse Data Representation using Crossbar Architectures," in *ISCA*, 2020.
[24] M. Zhou *et al.*, "Ultra Efficient Acceleration for De Novo Genome Assembly via Near-Memory Computing," in *PACT*, 2021.
[25] X. Xie *et al.*, "SpaceA: Sparse Matrix Vector Multiplication on Processing-in-Memory Accelerator," in *HPCA*, 2021.
[26] M. Zhou *et al.*, "HyGraph: Accelerating Graph Processing with Hybrid Memory-Centric Computing," in *DATE*, 2021.
[27] M. Lenjani *et al.*, "Gearbox: A Case for Supporting Accumulation Dispatching and Hybrid Partitioning in PIM-based Accelerators," in *ISCA*, 2022.
[28] M. Orenes-Vera *et al.*, "Dalorex: A Data-Local Program Execution and Architecture for Memory-Bound Applications," in *HPCA*, 2023.
[29] J. Choquette, "Nvidia Hopper GPU: Scaling Performance," in *Hot Chips*, 2022.
[30] F. Devaux, "The True Processing In Memory Accelerator," in *Hot Chips*, 2019.
[31] J. Gómez-Luna *et al.*, "Benchmarking a New Paradigm: Experimental Analysis and Characterization of a Real Processing-in-Memory System," *IEEE Access*, 2022.
[32] J. Gomez-Luna *et al.*, "Evaluating Machine Learning Workloads on Memory-Centric Computing Systems," in *ISPASS*, 2023.
[33] S. Lee *et al.*, "Hardware Architecture and Software Stack for PIM Based on Commercial DRAM Technology: Industrial Product," in *ISCA*, 2021.
[34] Y.-C. Kwon *et al.*, "25.4 A 20nm 6GB Function-In-Memory DRAM, Based on HBM2 with a 1.2 TFLOPS Programmable Computing Unit Using Bank-Level Parallelism, for Machine Learning Applications," in *ISSCC*, 2021.
[35] L. Ke *et al.*, "Near-Memory Processing in Action: Accelerating Personalized Recommendation with AxDIMM," *IEEE Micro*, 2021.
[36] D. Lee *et al.*, "Improving In-Memory Database Operations with Acceleration DIMM (AXDIMM)," in *DaMoN*, 2022.
[37] S. Lee *et al.*, "A 1ynm 1.25V 8Gb, 16Gb/s/pin GDDR6-based Accelerator-in-Memory supporting 1TFLOPS MAC Operation and Various Activation Functions for Deep-Learning Applications," in *ISSCC*, 2022.
[38] D. Niu *et al.*, "184QPS/W 64Mb/mm^2 3D Logic-to-DRAM Hybrid Bonding with Process-Near-Memory Engine for Recommendation System," in *ISSCC*, 2022.
[39] Y. Kwon, "System Architecture and Software Stack for GDDR6-AiM," in *HCS*, 2022.
[40] G. Singh *et al.*, "FPGA-based Near-Memory Acceleration of Modern Data-Intensive Applications," *IEEE Micro*, 2021.
[41] G. Singh *et al.*, "Accelerating Weather Prediction using Near-Memory Reconfigurable Fabric," *ACM TRETS*, 2021.
[42] S. Hong *et al.*, "PGX.D: A Fast Distributed Graph Processing Engine," in *SC*, 2015.
[43] T. N. Kipf and M. Welling, "Semi-Supervised Classification with Graph Convolutional Networks," in *ICLR*, 2017.
[44] L. Nai *et al.*, "GraphPIM: Enabling Instruction-Level PIM Offloading in Graph Computing Frameworks," in *HPCA*, 2017.
[45] M. Besta *et al.*, "SISA: Set-Centric Instruction Set Architecture for Graph Mining on Processing-in-Memory Systems," in *MICRO*, 2021.
[46] T. J. Ham *et al.*, "Graphicionado: A High-Performance and Energy-Efficient Accelerator for Graph Analytics," in *MICRO*, 2016.

https://arxiv.org/pdf/2306.16093

Accelerating Graph Pattern Mining

- Maciej Besta, Raghavendra Kanakagiri, Grzegorz Kwasniewski, Rachata Ausavarungnirun, Jakub Beránek, Konstantinos Kanellopoulos, Kacper Janda, Zur Vonarburg-Shmaria, Lukas Gianinazzi, Ioana Stefan, Juan Gómez-Luna, Marcin Copik, Lukas Kapp-Schwoerer, Salvatore Di Girolamo, Nils Blach, Marek Konieczny, Onur Mutlu, and Torsten Hoefler,
  "SISA: Set-Centric Instruction Set Architecture for Graph Mining on Processing-in-Memory Systems"
  *Proceedings of the 54th International Symposium on Microarchitecture (**MICRO**)*, Virtual, October 2021.
  [Slides (pdf)]
  [Talk Video (22 minutes)]
  [Lightning Talk Video (1.5 minutes)]
  [Full arXiv version]



SISA: Set-Centric Instruction Set Architecture for Graph Mining on Processing-in-Memory Systems

Maciej Besta[1], Raghavendra Kanakagiri[2], Grzegorz Kwasniewski[1], Rachata Ausavarungnirun[3], Jakub Beránek[4], Konstantinos Kanellopoulos[1], Kacper Janda[5], Zur Vonarburg-Shmaria[1], Lukas Gianinazzi[1], Ioana Stefan[1], Juan Gómez-Luna[1], Marcin Copik[1], Lukas Kapp-Schwoerer[1], Salvatore Di Girolamo[1], Nils Blach[1], Marek Konieczny[5], Onur Mutlu[1], Torsten Hoefler[1]

[1]ETH Zurich, Switzerland [2]IIT Tirupati, India [3]King Mongkut's University of Technology North Bangkok, Thailand [4]Technical University of Ostrava, Czech Republic [5]AGH-UST, Poland

Processing using DRAM

Background Work: RowClone

- Vivek Seshadri, Yoongu Kim, Chris Fallin, Donghyuk Lee, Rachata Ausavarungnirun, Gennady Pekhimenko, Yixin Luo, Onur Mutlu, Michael A. Kozuch, Phillip B. Gibbons, and Todd C. Mowry,
  "RowClone: Fast and Energy-Efficient In-DRAM Bulk Data Copy and Initialization"
  *Proceedings of the 46th International Symposium on Microarchitecture (**MICRO**)*, Davis, CA, December 2013. [Slides (pptx) (pdf)] [Lightning Session Slides (pptx) (pdf)] [Poster (pptx) (pdf)]

RowClone: Fast and Energy-Efficient In-DRAM Bulk Data Copy and Initialization

Vivek Seshadri vseshadr@cs.cmu.edu
Yoongu Kim yoongukim@cmu.edu
Chris Fallin* cfallin@c1f.net
Donghyuk Lee donghyuk1@cmu.edu
Rachata Ausavarungnirun rachata@cmu.edu
Gennady Pekhimenko gpekhime@cs.cmu.edu
Yixin Luo yixinluo@andrew.cmu.edu
Onur Mutlu onur@cmu.edu
Phillip B. Gibbons† phillip.b.gibbons@intel.com
Michael A. Kozuch† michael.a.kozuch@intel.com
Todd C. Mowry tcm@cs.cmu.edu
Carnegie Mellon University †Intel Pittsburgh

Background Work: PiDRAM

- Ataberk Olgun, Juan Gomez Luna, Konstantinos Kanellopoulos, Behzad Salami, Hasan Hassan, Oguz Ergin, and Onur Mutlu,
  "PiDRAM: A Holistic End-to-end FPGA-based Framework for Processing-in-DRAM"
  ACM Transactions on Architecture and Code Optimization (***TACO***), March 2023.
  [arXiv version]
  Presented at the 18th HiPEAC Conference, Toulouse, France, January 2023.
  [Slides (pptx) (pdf)]
  [Longer Lecture Slides (pptx) (pdf)]
  [Lecture Video (40 minutes)]
  [PiDRAM Source Code]

PiDRAM: A Holistic End-to-end FPGA-based Framework for Processing-in-DRAM

Ataberk Olgun[§] Juan Gómez Luna[§] Konstantinos Kanellopoulos[§] Behzad Salami[§]
Hasan Hassan[§] Oğuz Ergin[†] Onur Mutlu[§]

[§]*ETH Zürich* [†]*TOBB University of Economics and Technology*

Background Work: In-DRAM Bulk AND/OR

- Vivek Seshadri, Kevin Hsieh, Amirali Boroumand, Donghyuk Lee, Michael A. Kozuch, Onur Mutlu, Phillip B. Gibbons, and Todd C. Mowry,
  "Fast Bulk Bitwise AND and OR in DRAM"
  IEEE Computer Architecture Letters (***CAL***), April 2015.

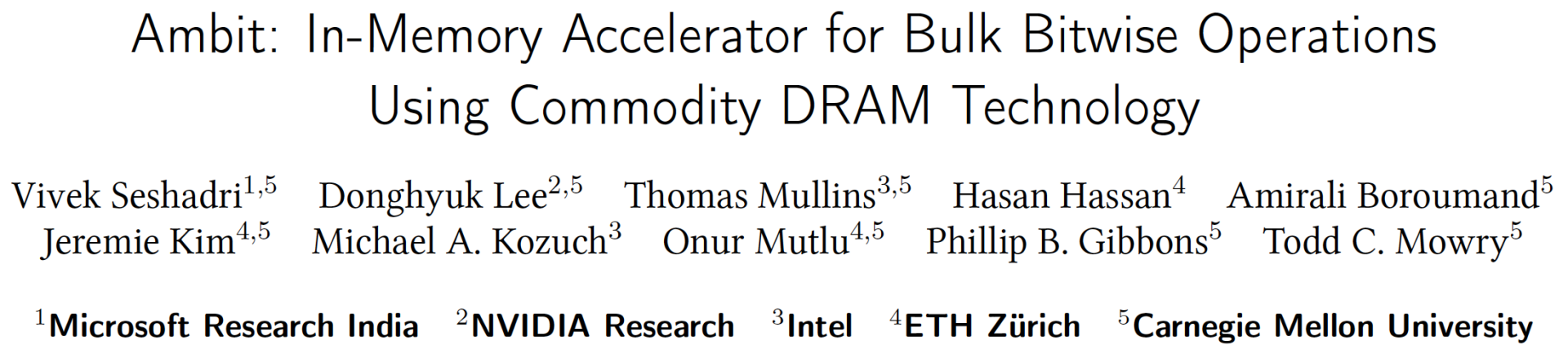
Fast Bulk Bitwise AND and OR in DRAM

Vivek Seshadri*, Kevin Hsieh*, Amirali Boroumand*, Donghyuk Lee*,
Michael A. Kozuch†, Onur Mutlu*, Phillip B. Gibbons†, Todd C. Mowry*

*Carnegie Mellon University †Intel Pittsburgh

Background Work: Ambit

- Vivek Seshadri, Donghyuk Lee, Thomas Mullins, Hasan Hassan, Amirali Boroumand, Jeremie Kim, Michael A. Kozuch, Onur Mutlu, Phillip B. Gibbons, and Todd C. Mowry,
  "Ambit: In-Memory Accelerator for Bulk Bitwise Operations Using Commodity DRAM Technology"
  *Proceedings of the 50th International Symposium on Microarchitecture (**MICRO**)*, Boston, MA, USA, October 2017.
  [Slides (pptx) (pdf)] [Lightning Session Slides (pptx) (pdf)] [Poster (pptx) (pdf)]


Ambit: In-Memory Accelerator for Bulk Bitwise Operations Using Commodity DRAM Technology

Vivek Seshadri[1,5] Donghyuk Lee[2,5] Thomas Mullins[3,5] Hasan Hassan[4] Amirali Boroumand[5]
Jeremie Kim[4,5] Michael A. Kozuch[3] Onur Mutlu[4,5] Phillip B. Gibbons[5] Todd C. Mowry[5]

[1]Microsoft Research India [2]NVIDIA Research [3]Intel [4]ETH Zürich [5]Carnegie Mellon University

Background: In-DRAM Bulk Bitwise Execution

- Vivek Seshadri and Onur Mutlu,
  "In-DRAM Bulk Bitwise Execution Engine"
  Invited Book Chapter in Advances in Computers, to appear in 2020.
  [Preliminary arXiv version]

In-DRAM Bulk Bitwise Execution Engine

Vivek Seshadri
Microsoft Research India
visesha@microsoft.com

Onur Mutlu
ETH Zürich
onur.mutlu@inf.ethz.ch

Background: SIMDRAM Framework

- Nastaran Hajinazar, Geraldo F. Oliveira, Sven Gregorio, Joao Dinis Ferreira, Nika Mansouri Ghiasi, Minesh Patel, Mohammed Alser, Saugata Ghose, Juan Gomez-Luna, and Onur Mutlu, **"SIMDRAM: An End-to-End Framework for Bit-Serial SIMD Computing in DRAM"** *Proceedings of the 26th International Conference on Architectural Support for Programming Languages and Operating Systems* (***ASPLOS***), Virtual, March-April 2021.
  [2-page Extended Abstract]
  [Short Talk Slides (pptx) (pdf)]
  [Talk Slides (pptx) (pdf)]
  [Short Talk Video (5 mins)]
  [Full Talk Video (27 mins)]

SIMDRAM: A Framework for Bit-Serial SIMD Processing using DRAM

*Nastaran Hajinazar[1,2] *Geraldo F. Oliveira[1] Sven Gregorio[1] João Dinis Ferreira[1]
Nika Mansouri Ghiasi[1] Minesh Patel[1] Mohammed Alser[1] Saugata Ghose[3]
Juan Gómez-Luna[1] Onur Mutlu[1]

[1]ETH Zürich [2]Simon Fraser University [3]University of Illinois at Urbana–Champaign

In-DRAM Lookup-Table Based Execution

João Dinis Ferreira, Gabriel Falcao, Juan Gómez-Luna, Mohammed Alser, Lois Orosa, Mohammad Sadrosadati, Jeremie S. Kim, Geraldo F. Oliveira, Taha Shahroodi, Anant Nori, and Onur Mutlu,
"pLUTo: Enabling Massively Parallel Computation in DRAM via Lookup Tables"
*Proceedings of the 55th International Symposium on Microarchitecture (**MICRO**)*, Chicago, IL, USA, October 2022.
[Slides (pptx) (pdf)]
[Longer Lecture Slides (pptx) (pdf)]
[Lecture Video (26 minutes)]
[arXiv version]
[Source Code (Officially Artifact Evaluated with All Badges)]
Officially artifact evaluated as available, reusable and reproducible.

pLUTo: Enabling Massively Parallel Computation in DRAM via Lookup Tables

João Dinis Ferreira[§] Gabriel Falcao[†] Juan Gómez-Luna[§] Mohammed Alser[§]
Lois Orosa[§▽] Mohammad Sadrosadati[§] Jeremie S. Kim[§] Geraldo F. Oliveira[§]
Taha Shahroodi[‡] Anant Nori[⋆] Onur Mutlu[§]
[§]*ETH Zürich* [†]*IT, University of Coimbra* [▽]*Galicia Supercomputing Center* [‡]*TU Delft* [⋆]*Intel*

https://arxiv.org/pdf/2104.07699.pdf

MIMDRAM: More Flexible Processing using DRAM

- **Appears at HPCA 2024** **https://arxiv.org/pdf/2402.19080.pdf**

MIMDRAM: An End-to-End Processing-Using-DRAM System for High-Throughput, Energy-Efficient and Programmer-Transparent Multiple-Instruction Multiple-Data Computing

Geraldo F. Oliveira[†] Ataberk Olgun[†] Abdullah Giray Yağlıkçı[†] F. Nisa Bostancı[†]
Juan Gómez-Luna[†] Saugata Ghose[‡] Onur Mutlu[†]

[†] *ETH Züirich* [‡] *Univ. of Illinois Urbana-Champaign*

*Our **goal** is to design a flexible PUD system that overcomes the limitations caused by the large and rigid granularity of PUD. To this end, we propose MIMDRAM, a hardware/software co-designed PUD system that introduces new mechanisms to allocate and control only the necessary resources for a given PUD operation. The key idea of MIMDRAM is to leverage fine-grained DRAM (i.e., the ability to independently access smaller segments of a large DRAM row) for PUD computation. MIMDRAM exploits this key idea to enable a multiple-instruction multiple-data (MIMD) execution model in each DRAM subarray (and SIMD execution within each DRAM row segment).*

MIMDRAM: Executive Summary

Problem: Processing-Using-DRAM (PUD) suffers from three issues caused by DRAM's large and rigid access granularity

- Underutilization due to data parallelism variation in (and across) applications
- Limited computation support due to a lack of interconnects
- Challenging programming model due to a lack of compilers

Goal: Design a flexible PUD system that overcomes the three limitations caused by DRAM's large and rigid access granularity

Key Mechanism: MIMDRAM, a hardware/software co-design PUD system

- **Key idea**: leverage fine-grained DRAM for PUD operation
- **HW**: - simple changes to the DRAM array, enabling concurrent PUD operations
  - low-cost interconnects at the DRAM peripherals for data reduction
- **SW**: - compiler and OS support to generate and map PUD instructions

Key Results: MIMDRAM achieves

- **14.3x**, **30.6x**, and **6.8x** the energy efficiency of state-of-the-art PUD systems, a high-end CPU and GPU, respectively
- Small area cost to a DRAM chip (**1.11%**) and CPU die (**0.6%**)

Real DRAM Chips Are Already Quite Capable: FC-DRAM & SiMRA

DRAM Chips Are Already (Quite) Capable!

- **Appears at HPCA 2024** https://arxiv.org/pdf/2402.18736.pdf

Functionally-Complete Boolean Logic in Real DRAM Chips: Experimental Characterization and Analysis

İsmail Emir Yüksel Yahya Can Tuğrul Ataberk Olgun F. Nisa Bostancı A. Giray Yağlıkçı
Geraldo F. Oliveira Haocong Luo Juan Gómez-Luna Mohammad Sadrosadati Onur Mutlu

ETH Zürich

We experimentally demonstrate that COTS DRAM chips are capable of performing 1) functionally-complete *Boolean operations: NOT, NAND, and NOR and 2) many-input (i.e., more than two-input) AND and OR operations. We present an extensive characterization of* new *bulk bitwise operations in 256 off-the-shelf modern DDR4 DRAM chips. We evaluate the reliability of these operations using a metric called success rate: the fraction of correctly performed bitwise operations. Among our 19 new observations, we highlight four major results. First, we can perform the NOT operation on COTS DRAM chips with 98.37% success rate on average. Second, we can perform up to 16-input NAND, NOR, AND, and OR operations on COTS DRAM chips with high reliability (e.g., 16-input NAND, NOR, AND, and OR with average success rate of 94.94%, 95.87%, 94.94%, and 95.85%, respectively). Third, data pattern only slightly*

DRAM Chips Are Already (Quite) Capable!

- **https://arxiv.org/pdf/2312.02880.pdf**

PULSAR: Simultaneous Many-Row Activation for Reliable and High-Performance Computing in Off-the-Shelf DRAM Chips

Ismail Emir Yuksel Yahya Can Tugrul F. Nisa Bostanci Abdullah Giray Yaglikci Ataberk Olgun
Geraldo F. Oliveira Melina Soysal Haocong Luo Juan Gomez Luna Mohammad Sadrosadati
Onur Mutlu

ETH Zurich

We propose PULSAR, a new technique to enable high-success-rate and high-performance PuM operations in off-the-shelf DRAM chips. PULSAR leverages our new observation that a carefully-crafted sequence of DRAM commands simultaneously activates up to 32 DRAM rows. PULSAR overcomes the limitations of existing techniques by 1) replicating the input data to improve the success rate and 2) enabling new bulk bitwise operations (e.g., many-input majority, *Multi-RowInit*, and *Bulk-Write*) to improve the performance.

DRAM Chips Are Already (Quite) Capable!

- **Appears at DSN 2024**

Simultaneous Many-Row Activation in Off-the-Shelf DRAM Chips: Experimental Characterization and Analysis

İsmail Emir Yüksel[1] Yahya Can Tuğrul[1,2] F. Nisa Bostancı[1] Geraldo F. Oliveira[1]
A. Giray Yağlıkçı[1] Ataberk Olgun[1] Melina Soysal[1] Haocong Luo[1]
Juan Gómez-Luna[1] Mohammad Sadrosadati[1] Onur Mutlu[1]
[1]*ETH Zürich* [2]*TOBB University of Economics and Technology*

https://arxiv.org/pdf/2405.06081

The Capability of COTS DRAM Chips

We **demonstrate** that **COTS DRAM chips:**

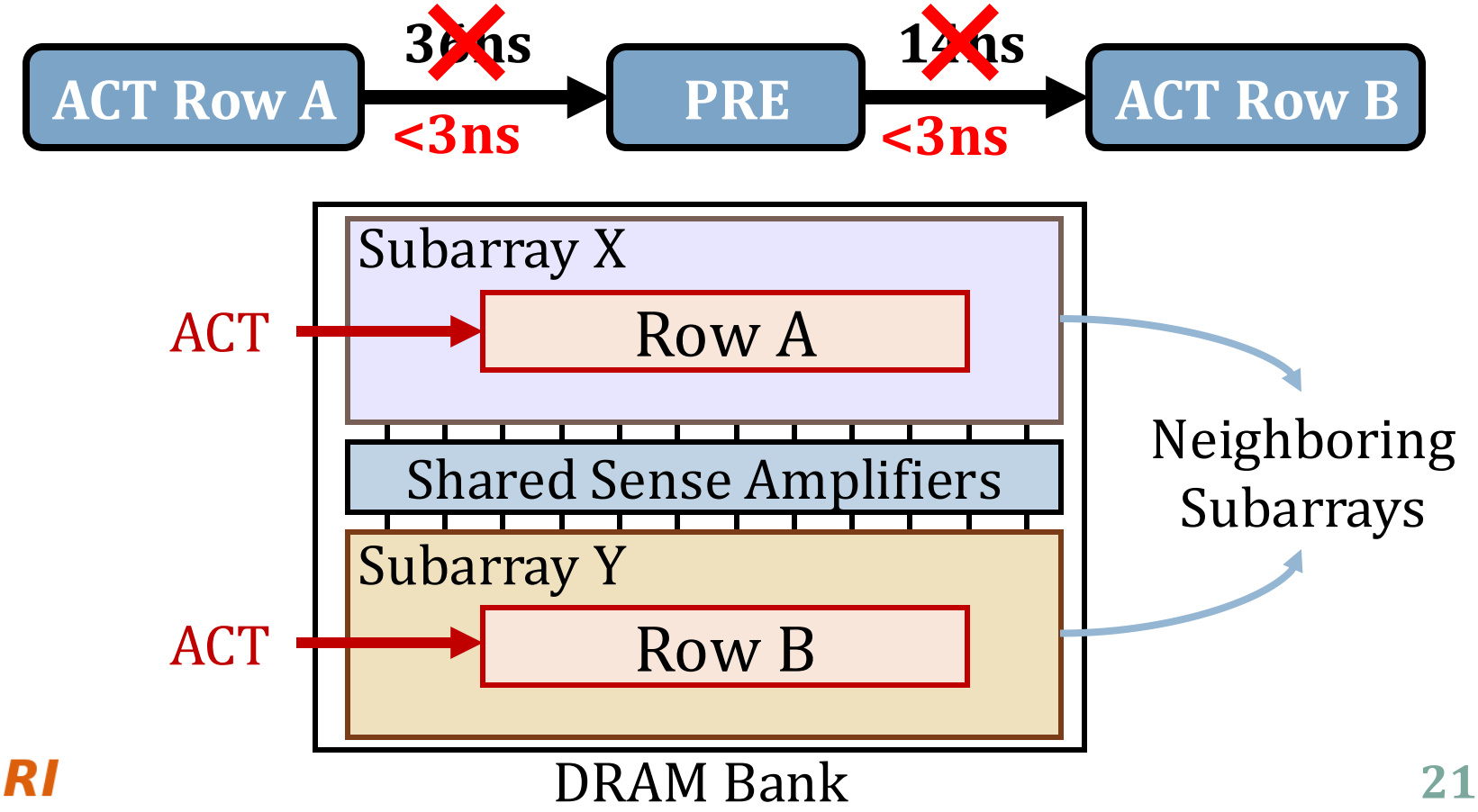
1 Can **simultaneously activate** up to **48 rows** in **two neighboring subarrays**

2 Can perform **NOT operation** with up to **32 output operands**

3 Can perform up to **16-input AND, NAND, OR, and NOR** operations

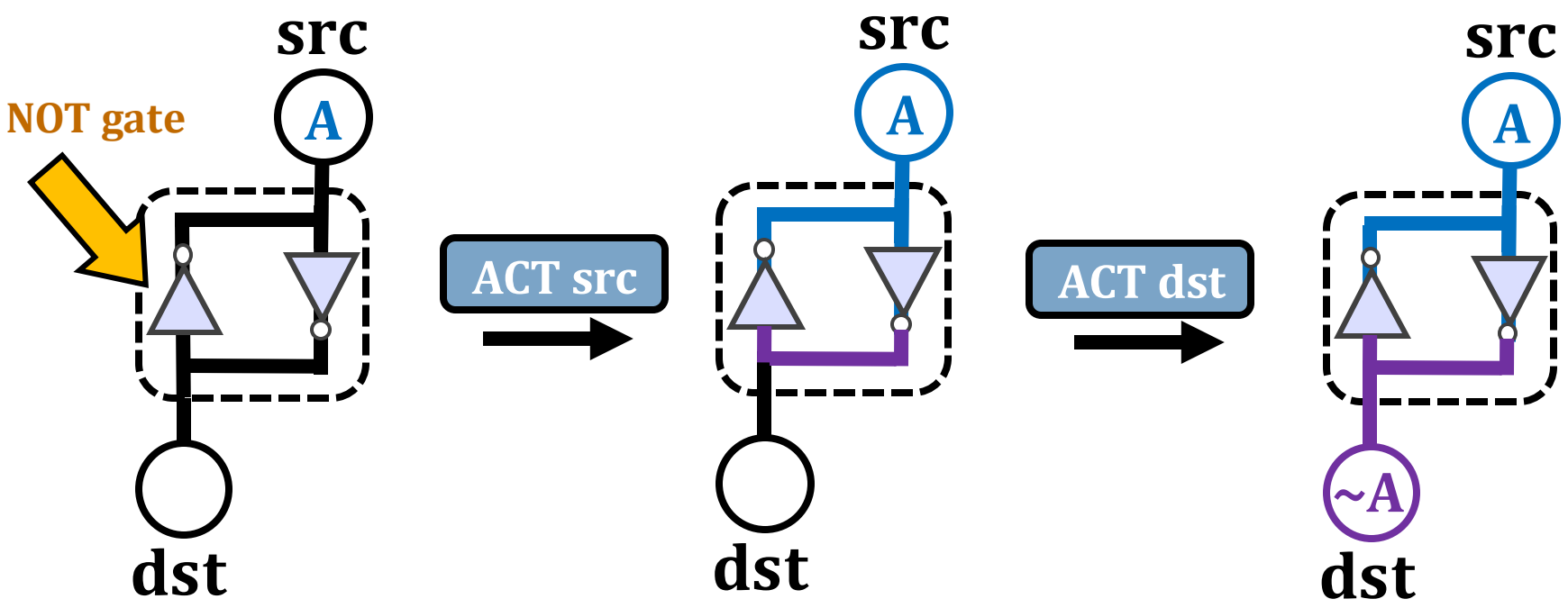
Finding: SiMRA Across Subarrays

Activating two rows in **quick succession**
can **simultaneously** activate
multiple rows in neighboring subarrays



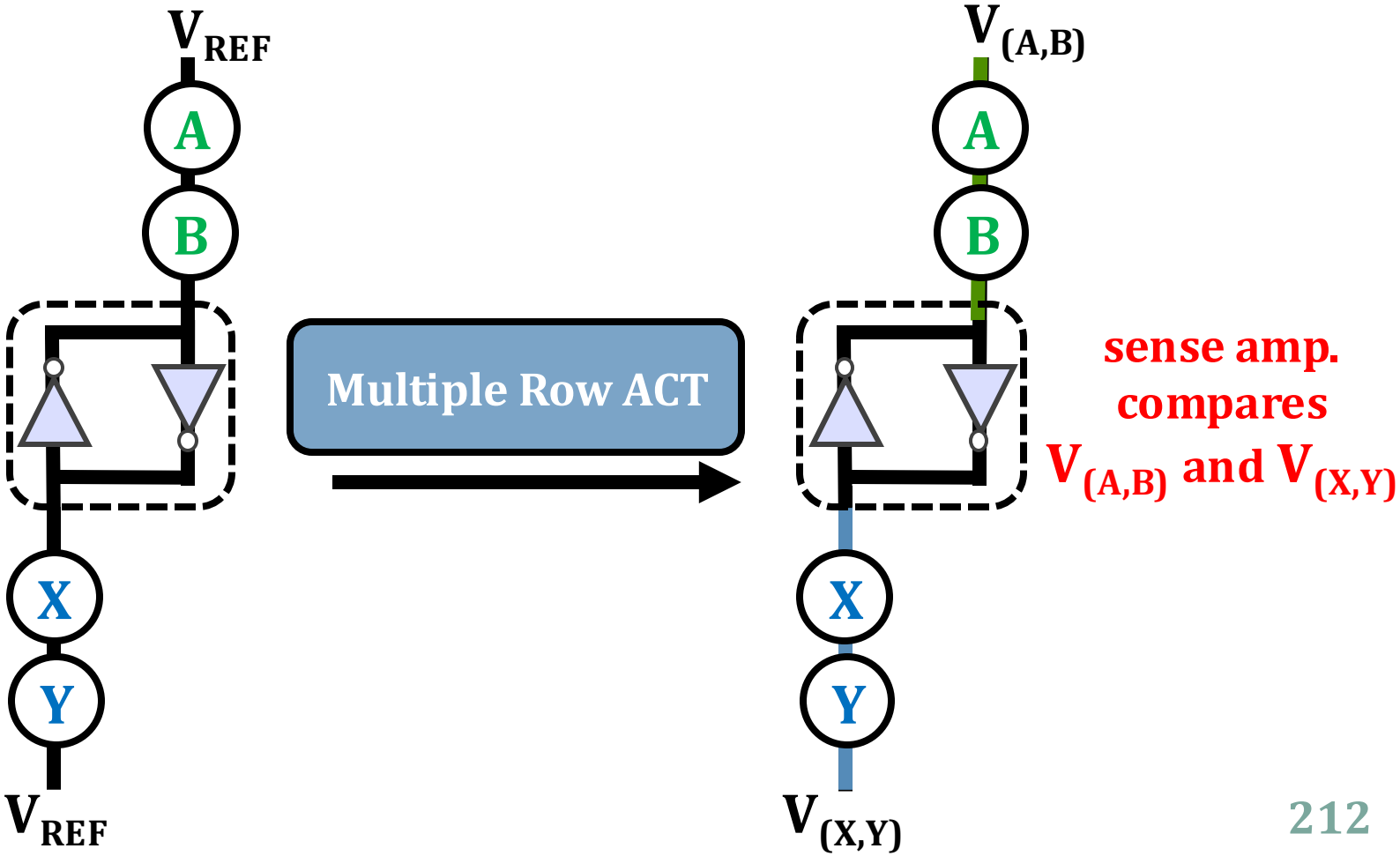
Key Idea: NOT Operation

Connect rows in neighboring subarrays
through **a NOT gate** by simultaneously activating rows

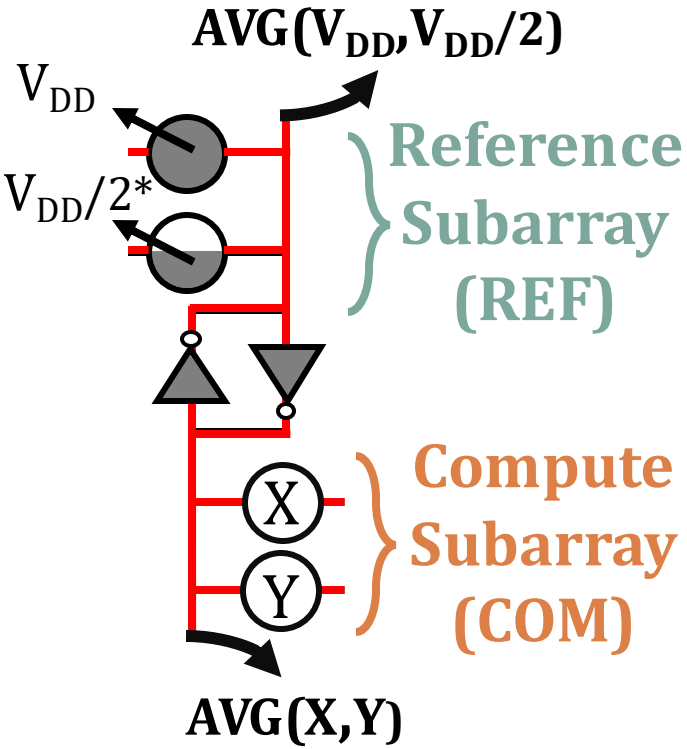


Key Idea: NAND, NOR, AND, OR

Manipulate the bitline voltage to express
a wide variety of functions using
multiple-row activation in neighboring subarrays

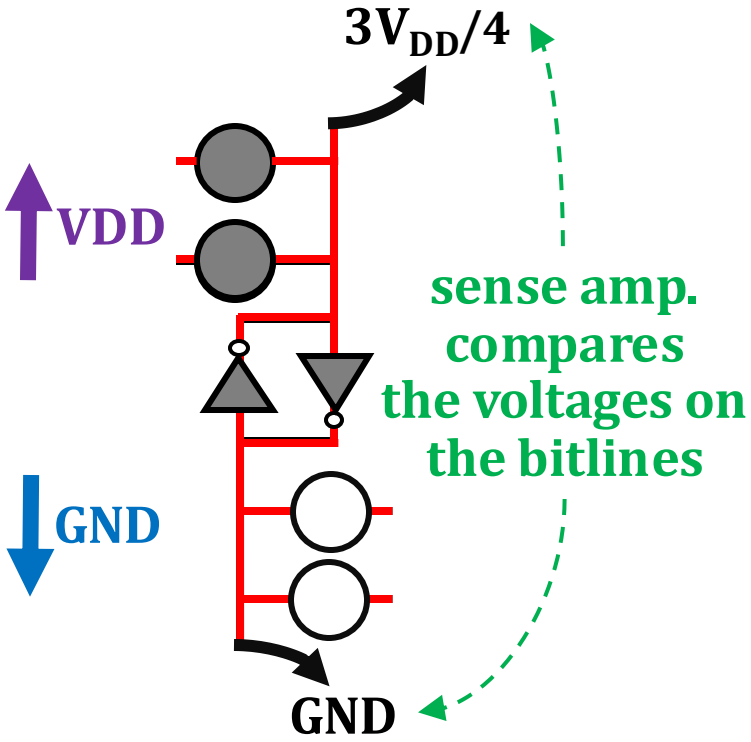


Two-Input AND and NAND Operations



*Gao et al., "FracDRAM: Fractional Values in Off-the-Shelf DRAM," in MICRO, 2022.

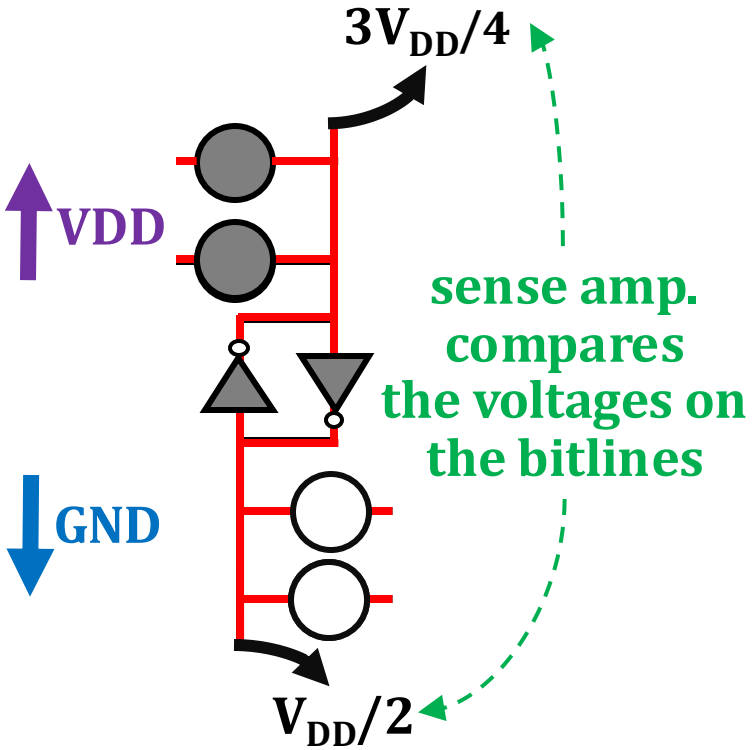
Two-Input AND and NAND Operations



V_{DD}=1 & GND = 0

X	Y	COM	REF
0	0	0	1

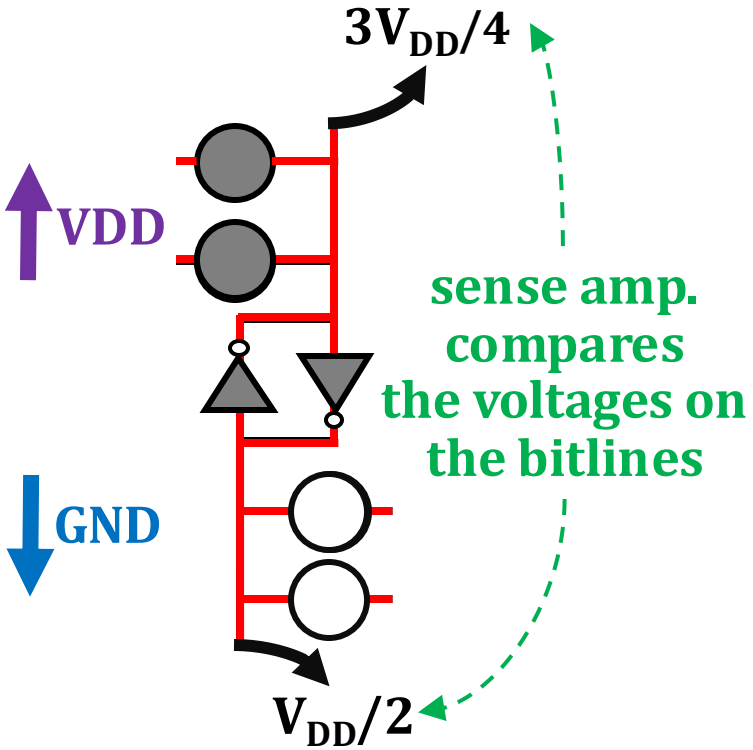
Two-Input AND and NAND Operations



V_{DD}=1 & GND = 0

X	Y	COM	REF
0	0	0	1
0	1	0	1

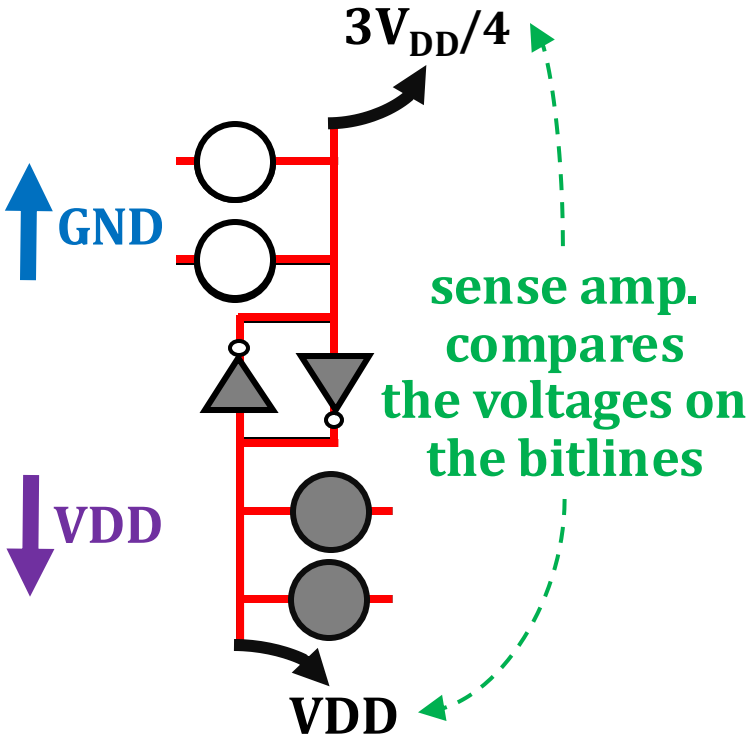
Two-Input AND and NAND Operations



V_{DD}=1 & GND = 0

X	Y	COM	REF
0	0	0	1
0	1	0	1
1	0	0	1

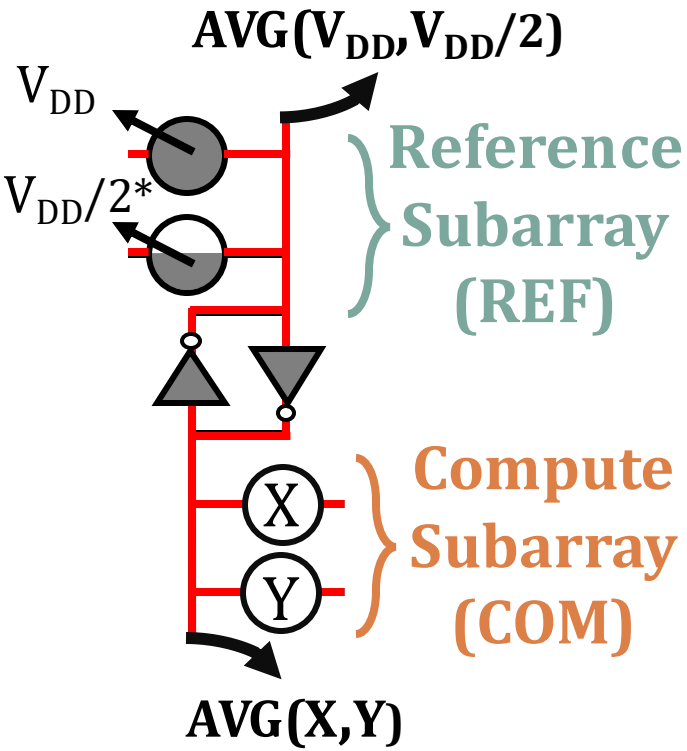
Two-Input AND and NAND Operations



V_{DD}=1 & GND = 0

X	Y	COM	REF
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

Two-Input AND and NAND Operations



V_{DD}=1 & GND = 0

X	Y	COM	REF
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0
		AND	NAND

*Gao et al., "FracDRAM: Fractional Values in Off-the-Shelf DRAM," in MICRO, 2022.

Many-Input AND, NAND, OR, and NOR Operations

We can express AND, NAND, OR, and NOR operations by carefully manipulating the **reference voltage**

Functionally-Complete Boolean Logic in Real DRAM Chips: Experimental Characterization and Analysis

İsmail Emir Yüksel Yahya Can Tuğrul Ataberk Olgun F. Nisa Bostancı A. Giray Yağlıkçı
Geraldo F. Oliveira Haocong Luo Juan Gómez-Luna Mohammad Sadrosadati Onur Mutlu

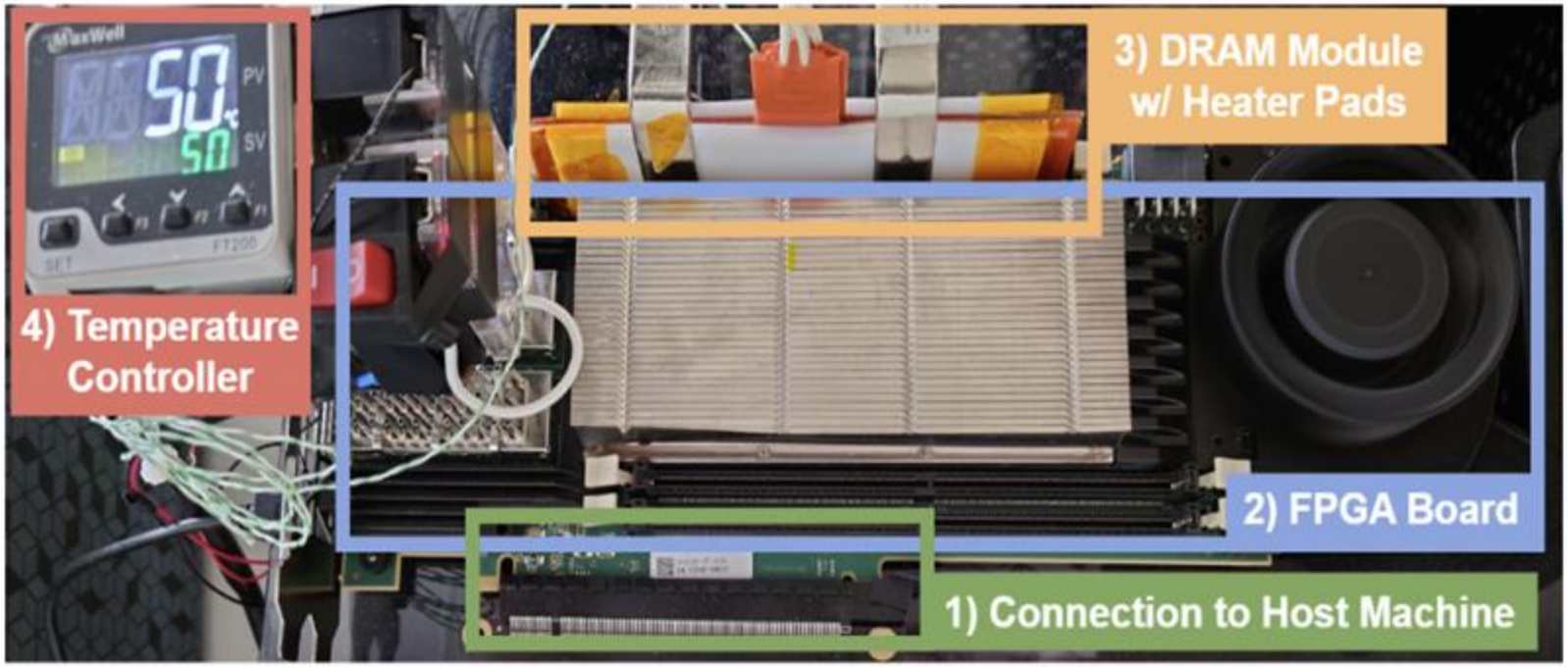
ETH Zürich

(More details in the paper)

https://arxiv.org/pdf/2402.18736.pdf

DRAM Testing Infrastructure

- Developed from DRAM Bender [Olgun+, TCAD'23]*
- Fine-grained control over DRAM commands, timings, and temperature



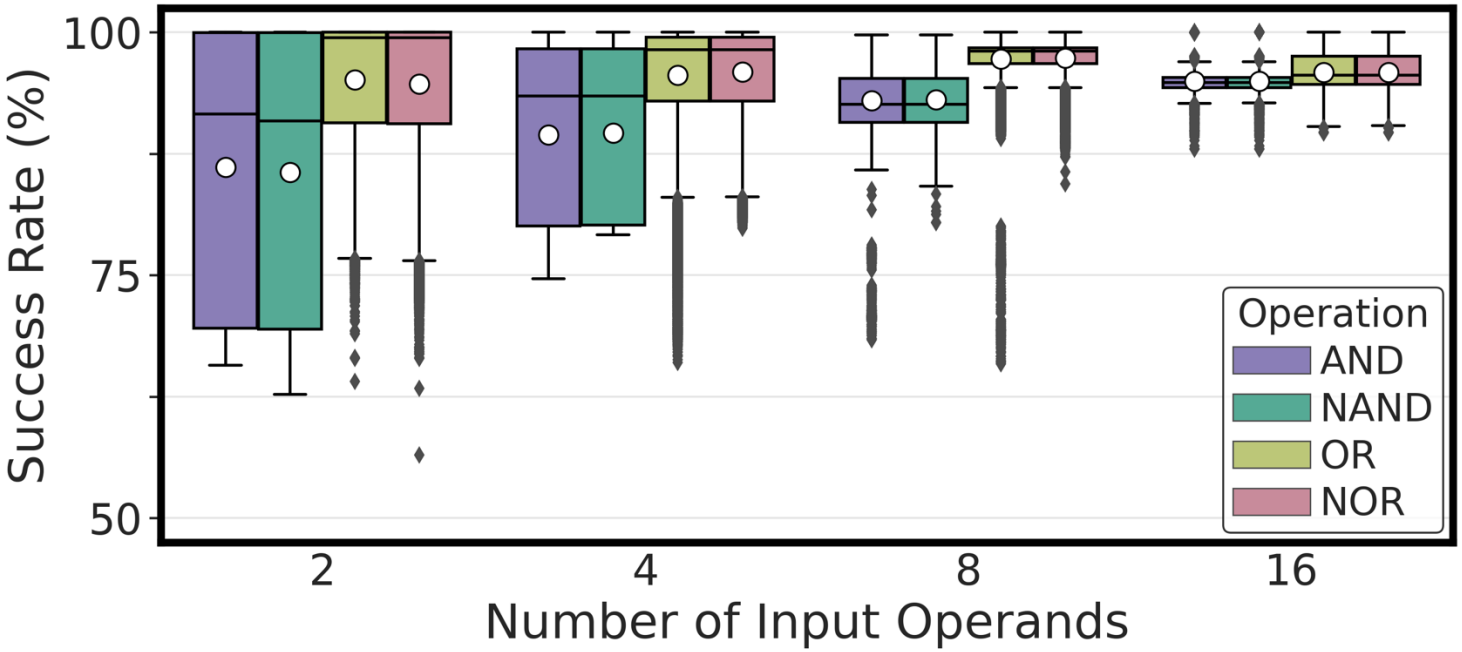
*Olgun et al., "DRAM Bender: An Extensible and Versatile FPGA-based Infrastructure to Easily Test State-of-the-art DRAM Chips," TCAD, 2023.

DRAM Chips Tested

- 256 DDR4 chips from two major DRAM manufacturers
- Covers different die revisions and chip densities

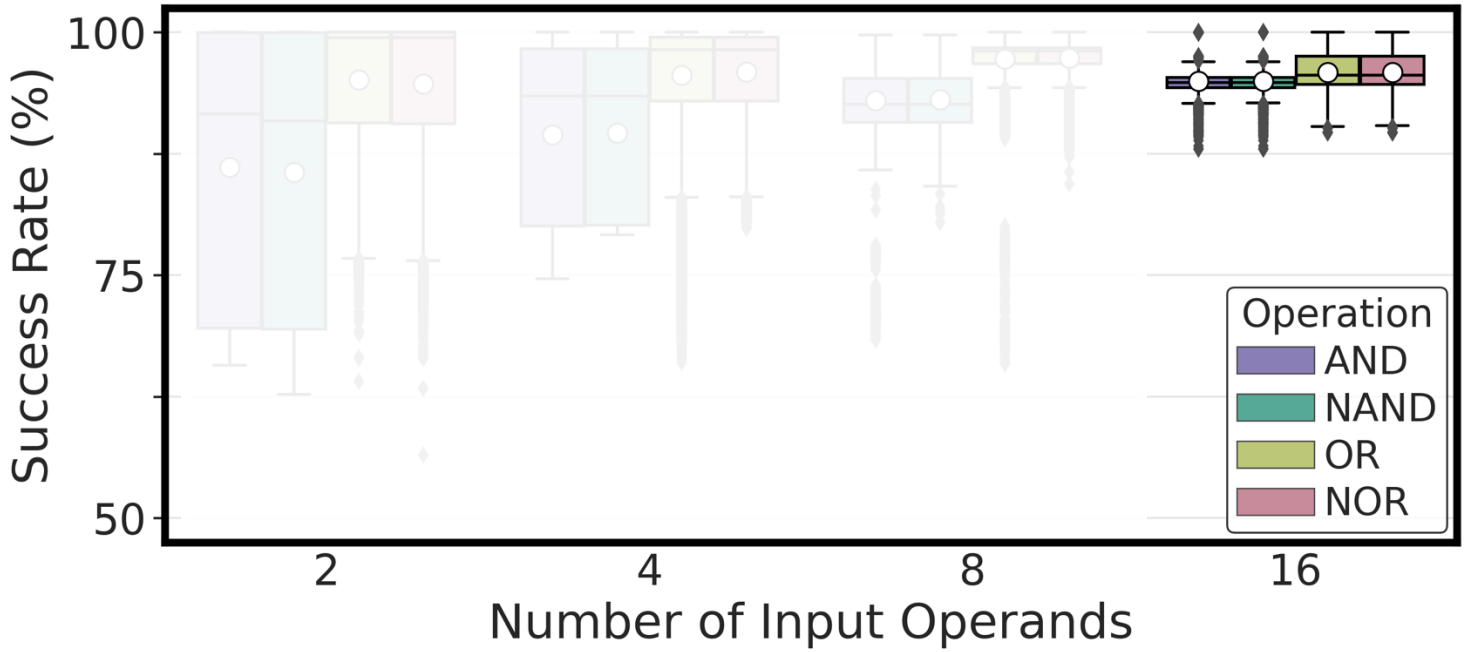
Chip Mfr.	#Modules (#Chips)	Die Rev.	Mfr. Date[a]	Chip Density	Chip Org.	Speed Rate
SK Hynix	9 (72)	M	N/A	4Gb	x8	2666MT/s
	5 (40)	A	N/A	4Gb	x8	2133MT/s
	1 (16)	A	N/A	8Gb	x8	2666MT/s
	1 (32)	A	18-14	4Gb	x4	2400MT/s
	1 (32)	A	16-49	8Gb	x4	2400MT/s
	1 (32)	M	16-22	8Gb	x4	2666MT/s
Samsung	1 (8)	F	21-02	4Gb	x8	2666MT/s
	2 (16)	D	21-10	8Gb	x8	2133MT/s
	1 (8)	A	22-12	8Gb	x8	3200MT/s

Performing AND, NAND, OR, and NOR



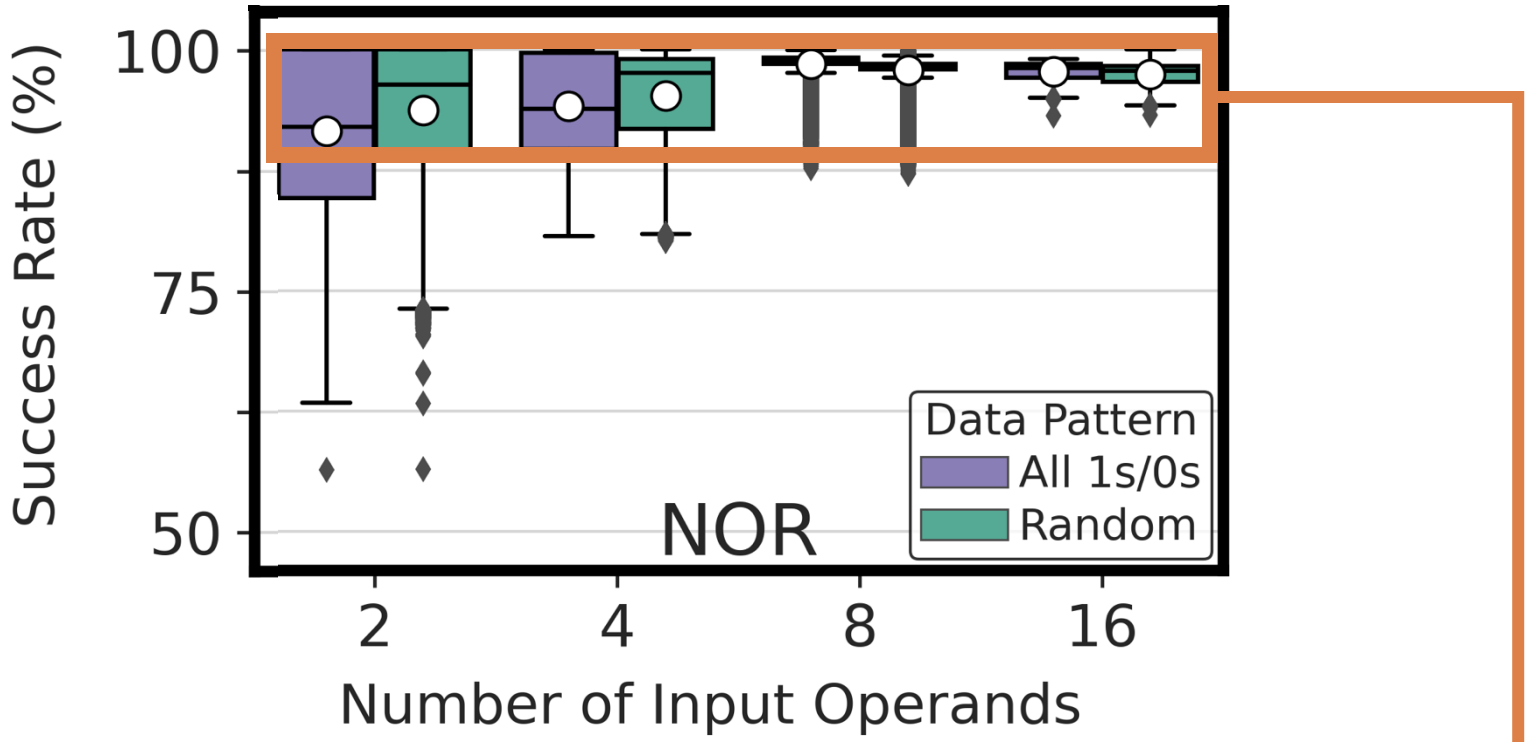
COTS DRAM chips can perform {2, 4, 8, 16}-input AND, NAND, OR, and NOR operations

Performing AND, NAND, OR, and NOR



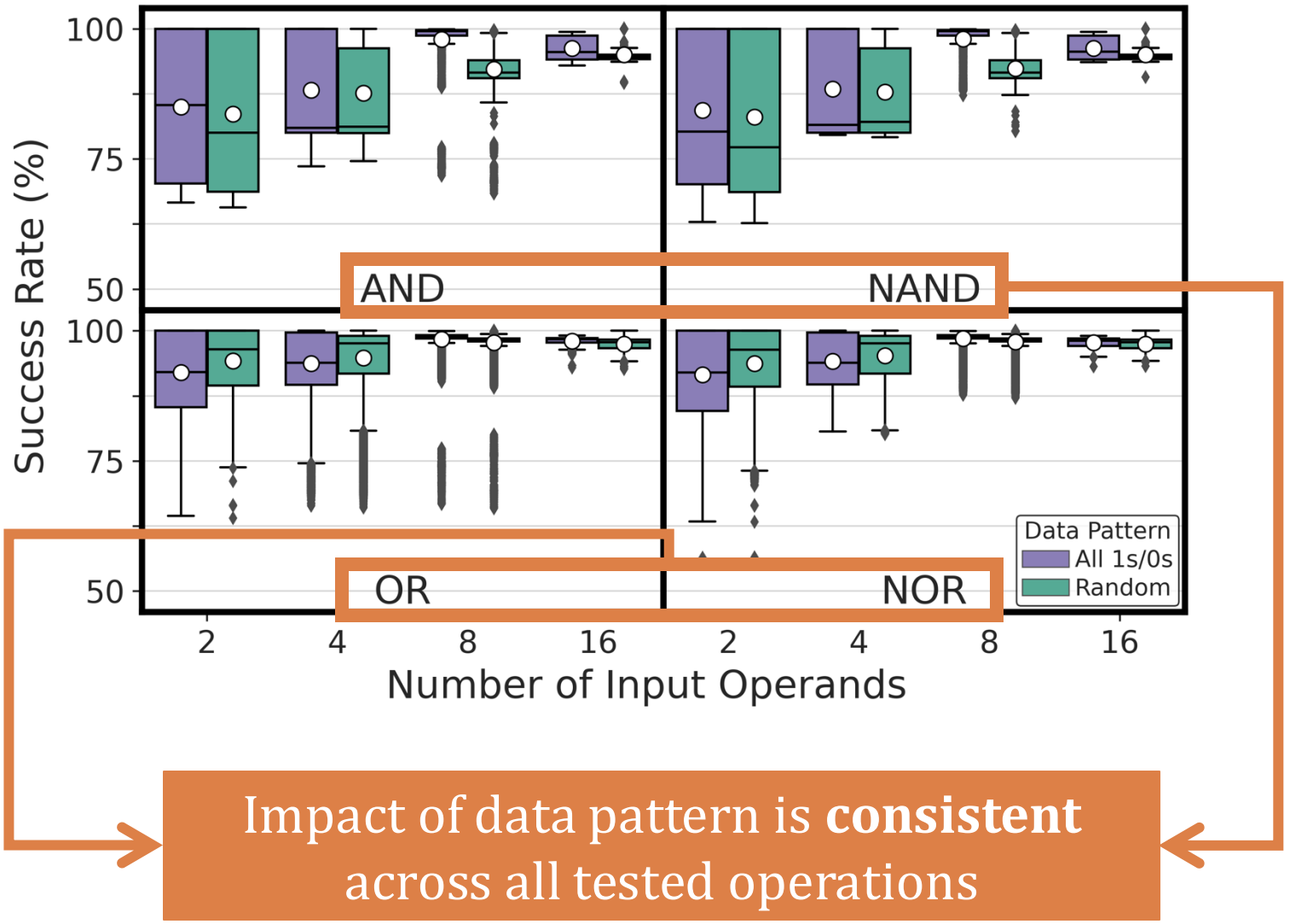
COTS DRAM chips can perform 16-input AND, NAND, OR, and NOR operations with very high success rate (>94%)

Impact of Data Pattern



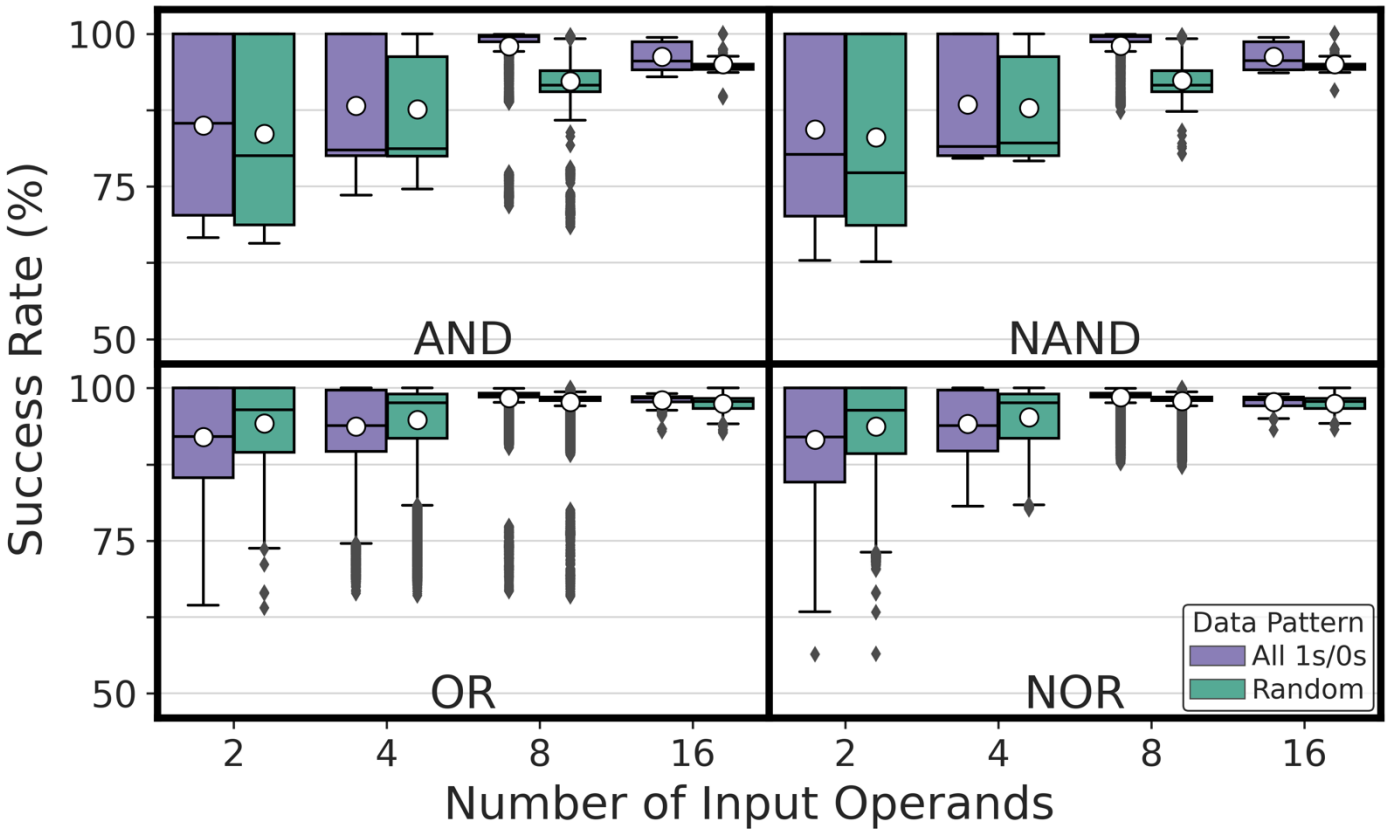
1.98% variation in **average success rate** across **all number of input operands**

Impact of Data Pattern



Impact of data pattern is **consistent** across all tested operations

Impact of Data Pattern



Data pattern slightly affects the reliability of AND, NAND, OR, and NOR operations

Available on arXiv

Functionally-Complete Boolean Logic in Real DRAM Chips: Experimental Characterization and Analysis

İsmail Emir Yüksel Yahya Can Tuğrul Ataberk Olgun F. Nisa Bostancı A. Giray Yağlıkçı
Geraldo F. Oliveira Haocong Luo Juan Gómez-Luna Mohammad Sadrosadati Onur Mutlu

ETH Zürich

Processing-using-DRAM (PuD) is an emerging paradigm that leverages the analog operational properties of DRAM circuitry to enable massively parallel in-DRAM computation. PuD has the potential to significantly reduce or eliminate costly data movement between processing elements and main memory. A common approach for PuD architectures is to make use of bulk bitwise computation (e.g., AND, OR, NOT). Prior works experimentally demonstrate three-input MAJ (i.e., MAJ3) and two-input AND and OR operations in commercial off-the-shelf (COTS) DRAM chips. Yet, demonstrations on COTS DRAM chips do not *provide a functionally complete set of operations (e.g., NAND or AND and NOT).*

We experimentally demonstrate that COTS DRAM chips are capable of performing 1) functionally-complete *Boolean operations: NOT, NAND, and NOR and 2) many-input (i.e., more than two-input) AND and OR operations. We present an extensive*

systems and applications [12, 13]. Processing-using-DRAM (PuD) [29–32] is a promising paradigm that can alleviate the data movement bottleneck. PuD uses the analog operational properties of the DRAM circuitry to enable massively parallel in-DRAM computation. Many prior works [29–53] demonstrate that PuD can greatly reduce or eliminate data movement.

A widely used approach for PuD is to perform bulk bitwise operations, i.e., bitwise operations on large bit vectors. To perform bulk bitwise operations using DRAM, prior works propose modifications to the DRAM circuitry [29–31, 33, 35, 36, 43, 44, 46, 48–58]. Recent works [38, 41, 42, 45] experimentally demonstrate the feasibility of executing data copy & initialization [42, 45], i.e., the RowClone operation [49], and a subset of bitwise operations, i.e., three-input bitwise majority (MAJ3) and two-input AND and OR operations in unmodified commercial off-the-shelf (COTS) DRAM chips by operating beyond

https://arxiv.org/pdf/2402.18736.pdf

Summary

- We experimentally demonstrate that **commercial off-the-shelf (COTS)** DRAM chips can perform:
  - **Functionally-complete** Boolean operations: NOT, NAND, and NOR
  - **Up to 16-input** AND, NAND, OR, and NOR operations
- We characterize **the success rate** of these operations on **256 COTS DDR4 chips** from **two major manufacturers**
- We highlight **two key results:**
  - We can perform **NOT** and **{2, 4, 8, 16}-input AND, NAND, OR, and NOR** operations on COTS DRAM chips with **very high success rates (>94%)**
  - **Data pattern** and **temperature** only slightly affect the reliability of these operations

We believe these empirical results demonstrate the promising potential of using DRAM as a computation substrate

Simultaneous Many-Row Activation in Off-the-Shelf DRAM Chips

Experimental Characterization and Analysis

İsmail Emir Yüksel

Yahya C. Tuğrul F. Nisa Bostancı Geraldo F. Oliveira

A. Giray Yağlıkçı Ataberk Olgun Melina Soysal Haocong Luo

Juan Gómez–Luna Mohammad Sadr Onur Mutlu

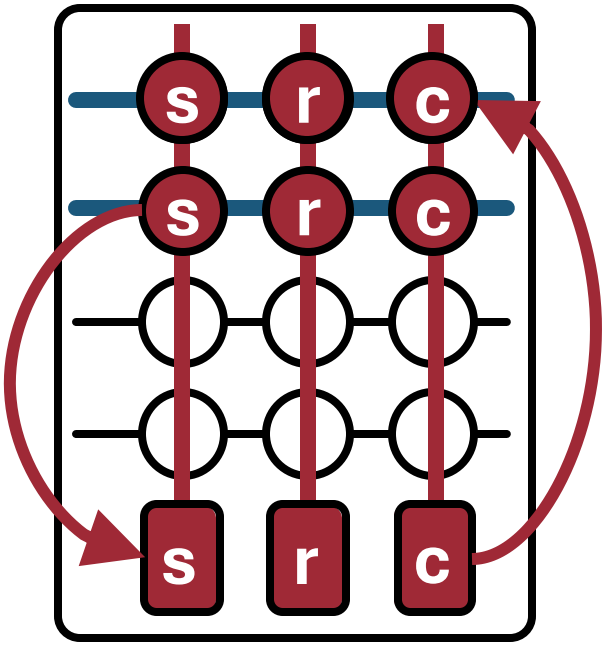
SAFARI

ETH zürich

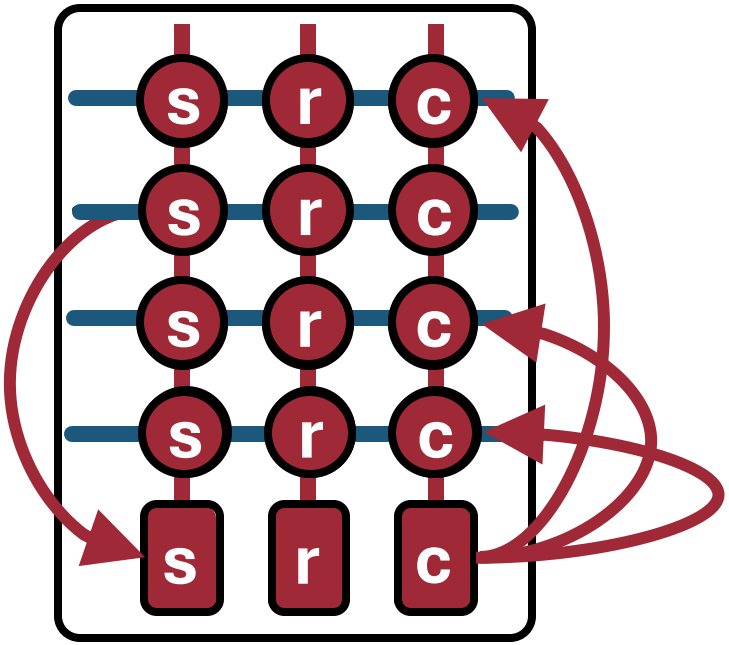
In-DRAM Multiple Row Copy (Multi-RowCopy)

Simultaneously activate many rows to
copy **one row's content** to **multiple destination rows**

RowClone



Multi-RowCopy



[Seshadri+ MICRO'13]

Key Takeaways from Multi-RowCopy

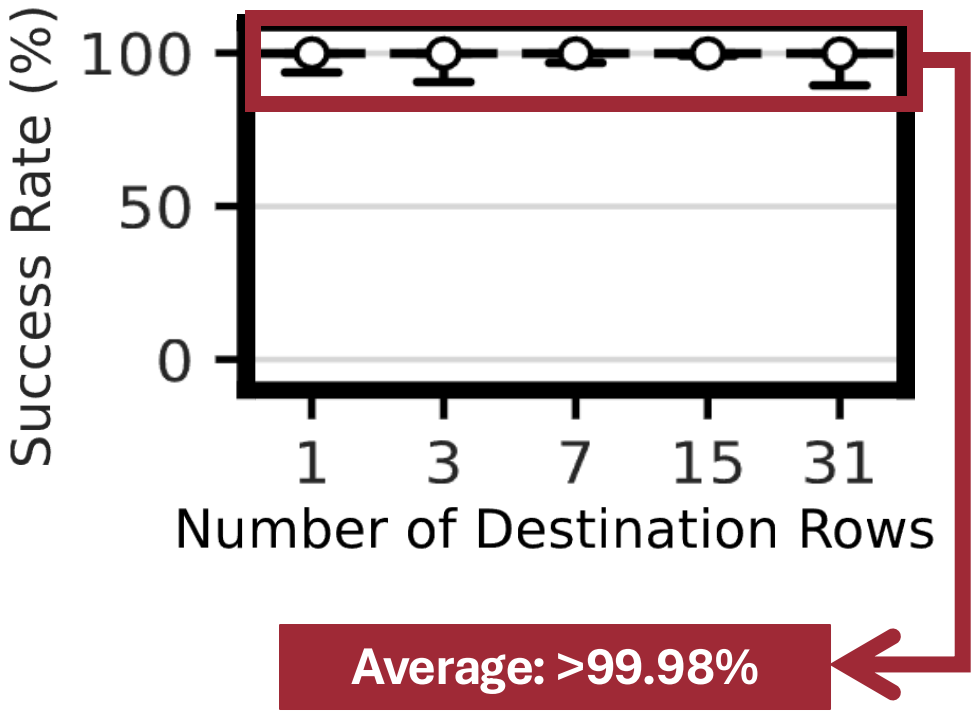
Key Takeaway 1

COTS DRAM chips are capable of copying one row's data to 1, 3, 7, 15, and 31 other rows at very high success rates

Key Takeaway 2

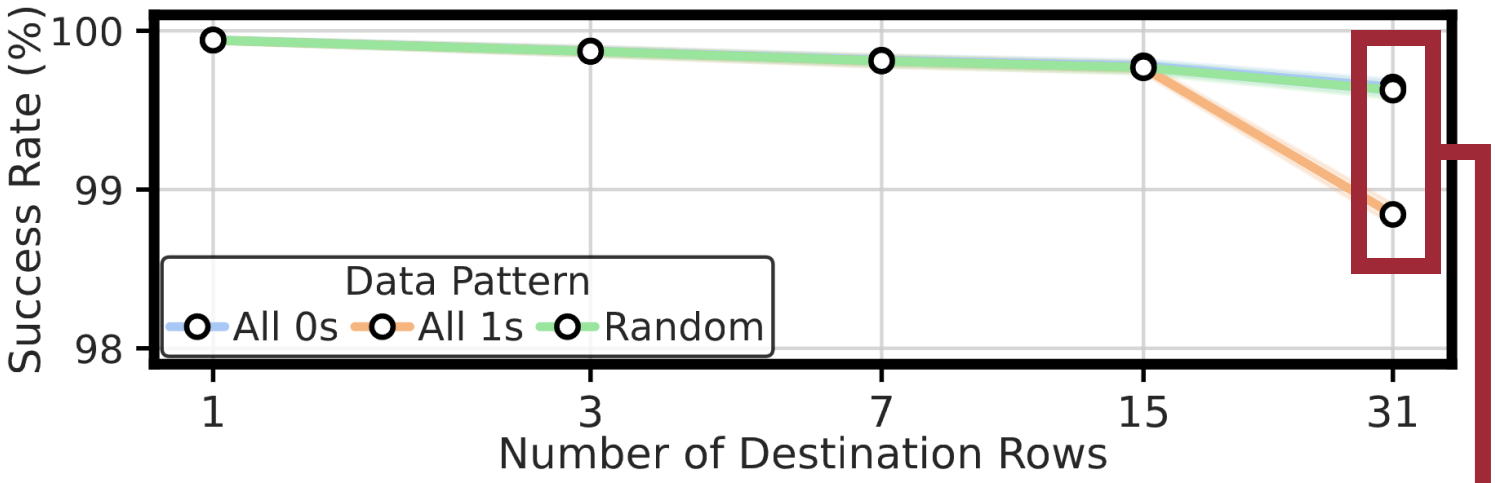
Multi-RowCopy in COTS DRAM chips is highly resilient to changes in data pattern, temperature, and wordline voltage

Robustness of Multi-RowCopy



COTS DRAM chips can copy one row's content to up to 31 rows with a very high success rate

Impact of Data Pattern



At most 0.79% decrease in average success rate

Data pattern has a small effect on the success rate of the Multi-RowCopy operation

Also in the Paper: Impact of Temperature & Voltage

Temperature

50 °C → 90 °C

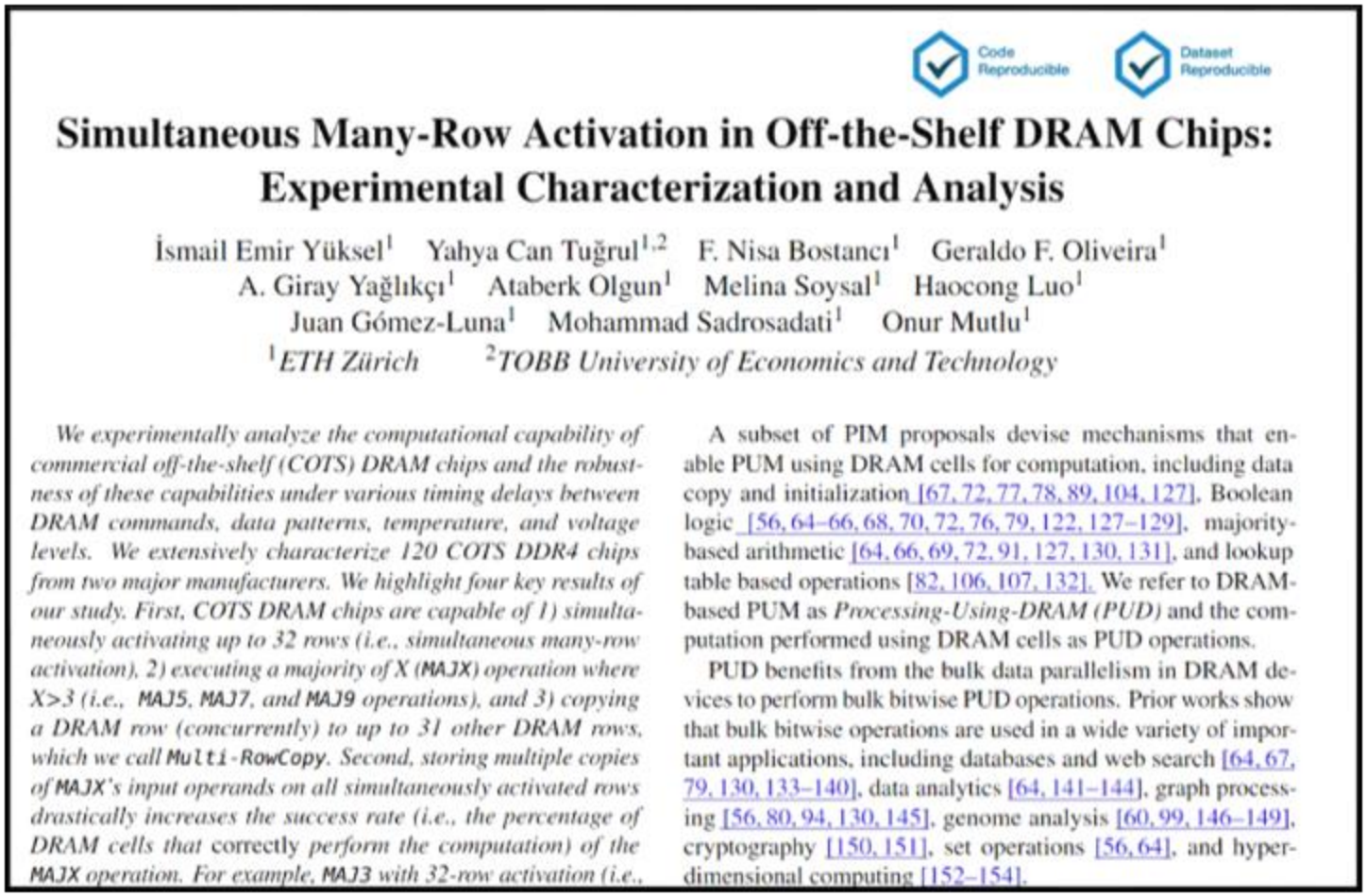
Increasing temperature up to 90°C has a very small effect on the success rate of the Multi-RowCopy operation

Wordline Voltage

2.5V → 2.1V

Reducing the wordline voltage only slightly affects the success rate of the Multi-RowCopy operation

Available on arXiv



Code Reproducible

Dataset Reproducible

Simultaneous Many-Row Activation in Off-the-Shelf DRAM Chips: Experimental Characterization and Analysis

İsmail Emir Yüksel[1] Yahya Can Tuğrul[1,2] F. Nisa Bostancı[1] Geraldo F. Oliveira[1]
A. Giray Yağlıkçı[1] Ataberk Olgun[1] Melina Soysal[1] Haocong Luo[1]
Juan Gómez-Luna[1] Mohammad Sadrosadati[1] Onur Mutlu[1]
[1]*ETH Zürich* [2]*TOBB University of Economics and Technology*

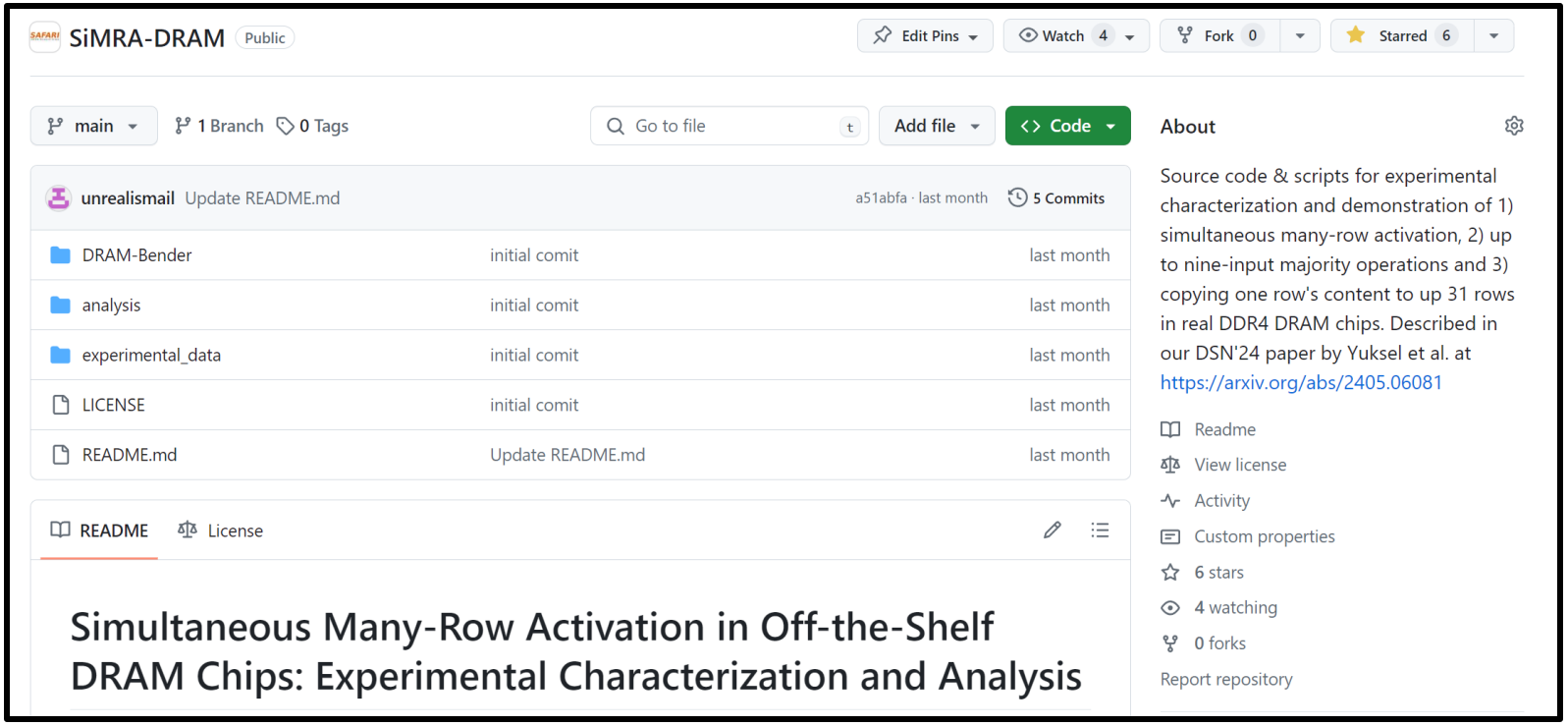
We experimentally analyze the computational capability of commercial off-the-shelf (COTS) DRAM chips and the robustness of these capabilities under various timing delays between DRAM commands, data patterns, temperature, and voltage levels. We extensively characterize 120 COTS DDR4 chips from two major manufacturers. We highlight four key results of our study. First, COTS DRAM chips are capable of 1) simultaneously activating up to 32 rows (i.e., simultaneous many-row activation), 2) executing a majority of X (`MAJX`) operation where X>3 (i.e., `MAJ5`, `MAJ7`, and `MAJ9` operations), and 3) copying a DRAM row (concurrently) to up to 31 other DRAM rows, which we call `Multi-RowCopy`. Second, storing multiple copies of `MAJX`'s input operands on all simultaneously activated rows drastically increases the success rate (i.e., the percentage of DRAM cells that correctly *perform the computation) of the `MAJX` operation. For example, `MAJ3` with 32-row activation (i.e.,*

A subset of PIM proposals devise mechanisms that enable PUM using DRAM cells for computation, including data copy and initialization [67, 72, 77, 78, 89, 104, 127], Boolean logic [56, 64–66, 68, 70, 72, 76, 79, 122, 127–129], majority-based arithmetic [64, 66, 69, 72, 91, 127, 130, 131], and lookup table based operations [82, 106, 107, 132]. We refer to DRAM-based PUM as *Processing-Using-DRAM (PUD)* and the computation performed using DRAM cells as PUD operations.

PUD benefits from the bulk data parallelism in DRAM devices to perform bulk bitwise PUD operations. Prior works show that bulk bitwise operations are used in a wide variety of important applications, including databases and web search [64, 67, 79, 130, 133–140], data analytics [64, 141–144], graph processing [56, 80, 94, 130, 145], genome analysis [60, 99, 146–149], cryptography [150, 151], set operations [56, 64], and hyperdimensional computing [152–154].

https://arxiv.org/pdf/2405.06081

Our Work is Open Source and Artifact Evaluated



https://github.com/CMU-SAFARI/SiMRA-DRAM

PIM Review and Open Problems

A Modern Primer on Processing in Memory

Onur Mutlu[a,b], Saugata Ghose[b,c], Juan Gómez-Luna[a], Rachata Ausavarungnirun[d]

SAFARI Research Group

[a] *ETH Zürich*
[b] *Carnegie Mellon University*
[c] *University of Illinois at Urbana-Champaign*
[d] *King Mongkut's University of Technology North Bangkok*

Onur Mutlu, Saugata Ghose, Juan Gomez-Luna, and Rachata Ausavarungnirun,
"A Modern Primer on Processing in Memory"
Invited Book Chapter in ***Emerging Computing: From Devices to Systems - Looking Beyond Moore and Von Neumann***, Springer, to be published in 2021.

https://arxiv.org/pdf/2012.03112.pdf

PIM Review and Open Problems (II)

A Workload and Programming Ease Driven Perspective of Processing-in-Memory

Saugata Ghose[†] Amirali Boroumand[†] Jeremie S. Kim[†§] Juan Gómez-Luna[§] Onur Mutlu[§†]

[†]*Carnegie Mellon University* [§]*ETH Zürich*

Saugata Ghose, Amirali Boroumand, Jeremie S. Kim, Juan Gomez-Luna, and Onur Mutlu,
"Processing-in-Memory: A Workload-Driven Perspective"
Invited Article in IBM Journal of Research & Development, Special Issue on Hardware for Artificial Intelligence, to appear in November 2019.
[Preliminary arXiv version]

https://arxiv.org/pdf/1907.12947.pdf

Processing in Memory: Adoption Challenges

1. Processing **using** Memory
2. Processing **near** Memory

Eliminating the Adoption Barriers

How to Enable Adoption of Processing in Memory

Potential Barriers to Adoption of PIM

1. **Applications** & **software** for PIM

2. Ease of **programming** (interfaces and compiler/HW support)

3. **System** and **security** support: coherence, synchronization, virtual memory, isolation, communication interfaces, …

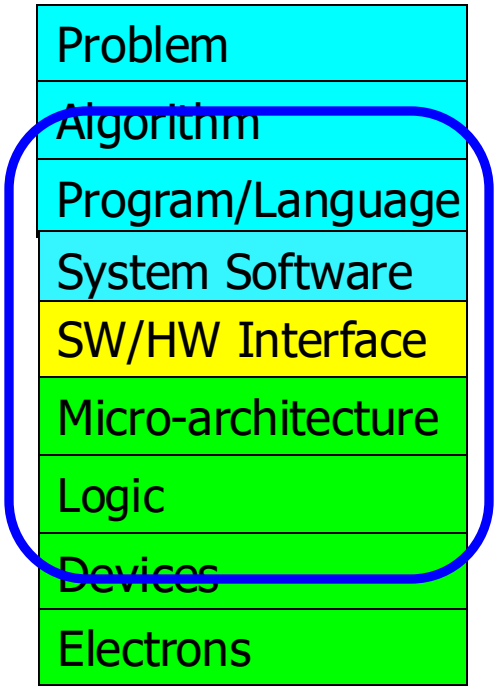
4. **Runtime** and **compilation** systems for adaptive scheduling, data mapping, access/sharing control, …

5. **Infrastructures** to assess benefits and feasibility

All can be solved with change of mindset

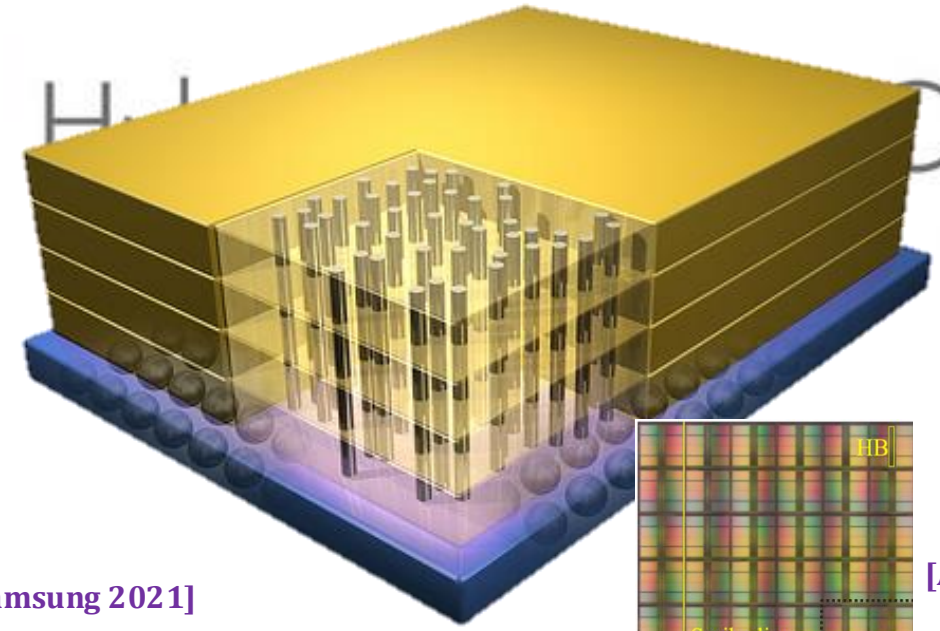
We Need to Revisit the Entire Stack

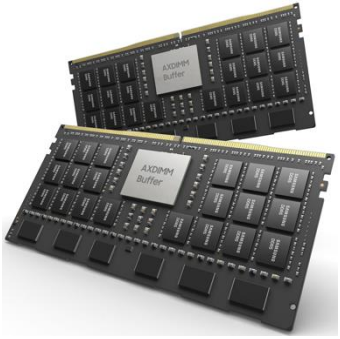
- With a **memory-centric mindset**



We can get there step by step

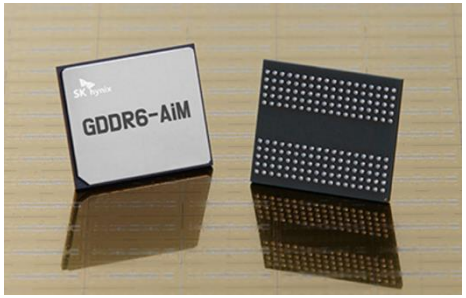
Processing-in-Memory Landscape Today



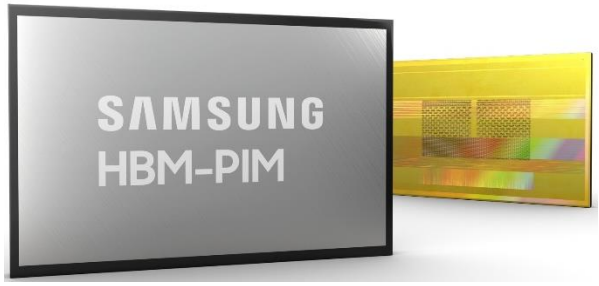


[Samsung 2021]

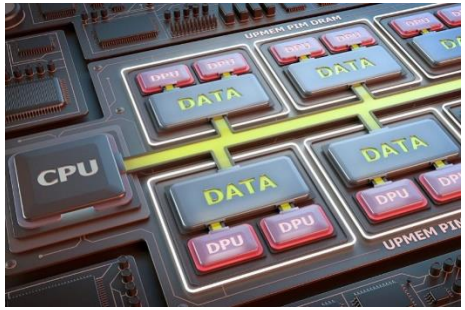
[Alibaba 2022]



[SK Hynix 2022]



[Samsung 2021]



[UPMEM 2019]

And, many other experimental chips and startups

Adoption: How to Keep It **Simple**?

- Junwhan Ahn, Sungjoo Yoo, Onur Mutlu, and Kiyoung Choi, **"PIM-Enabled Instructions: A Low-Overhead, Locality-Aware Processing-in-Memory Architecture"** *Proceedings of the 42nd International Symposium on Computer Architecture (**ISCA**)*, Portland, OR, June 2015. [Slides (pdf)] [Lightning Session Slides (pdf)]

PIM-Enabled Instructions: A Low-Overhead, Locality-Aware Processing-in-Memory Architecture

Junwhan Ahn Sungjoo Yoo Onur Mutlu† Kiyoung Choi
junwhan@snu.ac.kr, sungjoo.yoo@gmail.com, onur@cmu.edu, kchoi@snu.ac.kr
Seoul National University †Carnegie Mellon University

Adoption: How to Keep It **Simple**?

- Junwhan Ahn, Sungjoo Yoo, Onur Mutlu, and Kiyoung Choi, **"PIM-Enabled Instructions: A Low-Overhead, Locality-Aware Processing-in-Memory Architecture"** *Proceedings of the 42nd International Symposium on Computer Architecture (**ISCA**)*, Portland, OR, June 2015. [Slides (pdf)] [Lightning Session Slides (pdf)]

PIM-Enabled Instructions: A Low-Overhead, Locality-Aware Processing-in-Memory Architecture

Junwhan Ahn Sungjoo Yoo Onur Mutlu† Kiyoung Choi
junwhan@snu.ac.kr, sungjoo.yoo@gmail.com, onur@cmu.edu, kchoi@snu.ac.kr
Seoul National University †Carnegie Mellon University

Adoption: How to Ease **Programmability**? (I)

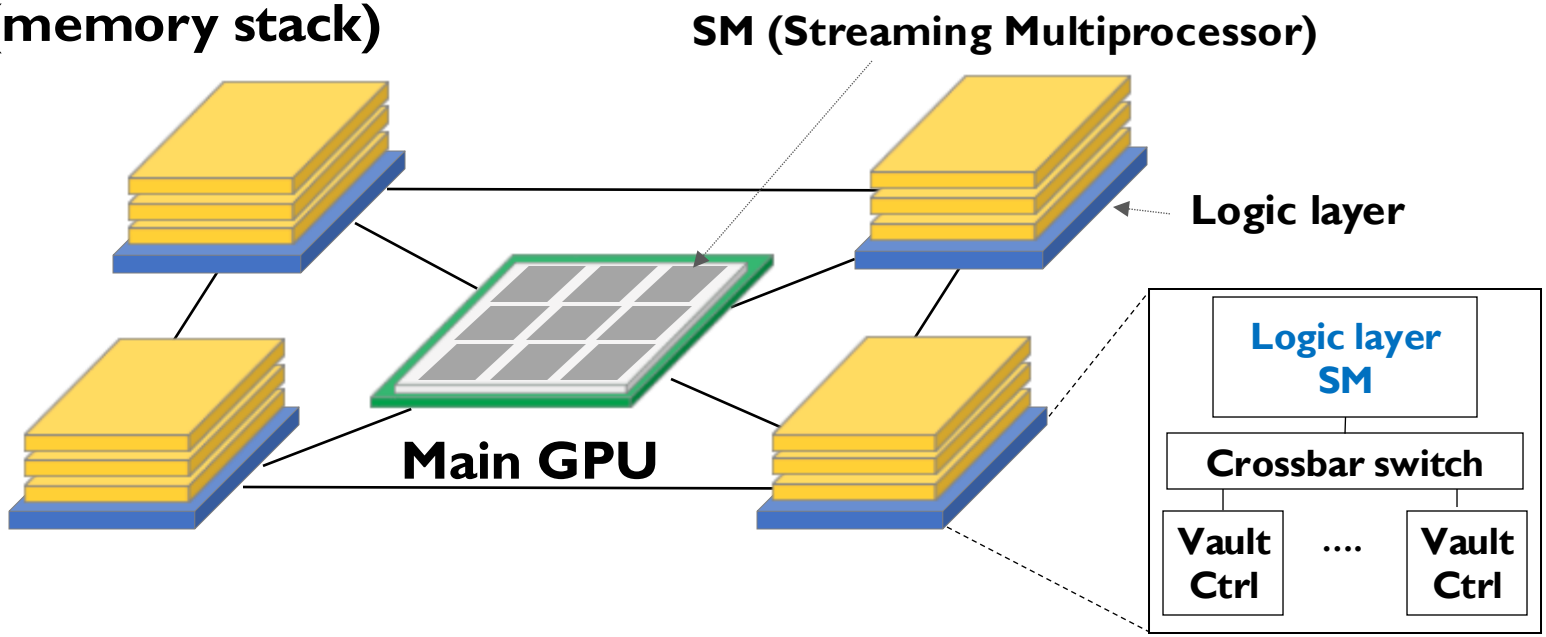
- Kevin Hsieh, Eiman Ebrahimi, Gwangsun Kim, Niladrish Chatterjee, Mike O'Connor, Nandita Vijaykumar, Onur Mutlu, and Stephen W. Keckler, **"Transparent Offloading and Mapping (TOM): Enabling Programmer-Transparent Near-Data Processing in GPU Systems"**
  *Proceedings of the 43rd International Symposium on Computer Architecture (**ISCA**)*, Seoul, South Korea, June 2016.
  [Slides (pptx) (pdf)]
  [Lightning Session Slides (pptx) (pdf)]

Transparent Offloading and Mapping (TOM):
Enabling Programmer-Transparent Near-Data Processing in GPU Systems

Kevin Hsieh[‡] Eiman Ebrahimi[†] Gwangsun Kim[*] Niladrish Chatterjee[†] Mike O'Connor[†]
Nandita Vijaykumar[‡] Onur Mutlu[§‡] Stephen W. Keckler[†]
[‡]Carnegie Mellon University [†]NVIDIA [*]KAIST [§]ETH Zürich

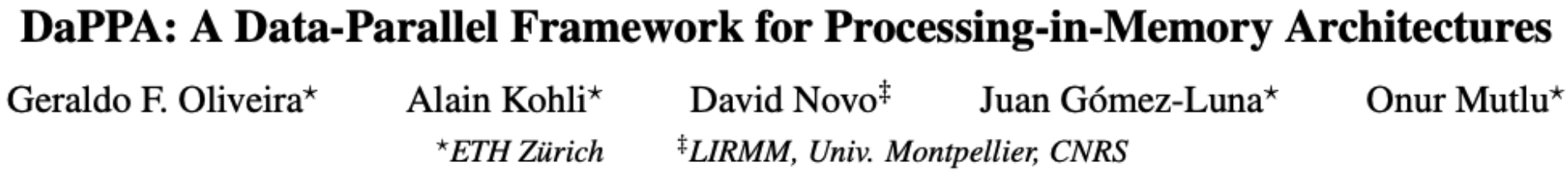
Truly Distributed GPU Processing with PIM

```
__global__
void applyScaleFactorsKernel( uint8_T * const out,
    uint8_T const * const in, const double *factor,
    size_t const numRows, size_t const numCols )
{
    // Work out which pixel we are working on.
    const int rowIdx = blockIdx.x * blockDim.x + threadIdx.x;
    const int colIdx = blockIdx.y;
    const int sliceIdx = threadIdx.z;

    // Check this thread isn't off the image
    if( rowIdx >= numRows ) return;

    // Compute the index of my element
    size_t linearIdx = rowIdx + colIdx*numRows +
        sliceIdx*numRows*numCols;
```



Adoption: How to Ease **Programmability**? (II)

- Geraldo F. Oliveira, Alain Kohli, David Novo, Juan Gómez-Luna, Onur Mutlu, "**DaPPA: A Data-Parallel Framework for Processing-in-Memory Architectures,**" in *PACT SRC Student Competition,* Vienna, Austria, October 2023.



DaPPA: A Data-Parallel Framework for Processing-in-Memory Architectures

Geraldo F. Oliveira⋆ Alain Kohli⋆ David Novo‡ Juan Gómez-Luna⋆ Onur Mutlu⋆

⋆*ETH Zürich* ‡*LIRMM, Univ. Montpellier, CNRS*

Adoption: How to Ease **Programmability**? (III)

- Jinfan Chen, Juan Gómez-Luna, Izzat El Hajj, YuXin Guo, and Onur Mutlu,
  "SimplePIM: A Software Framework for Productive and Efficient Processing in Memory"
  *Proceedings of the 32nd International Conference on Parallel Architectures and Compilation Techniques (**PACT**)*, Vienna, Austria, October 2023.

SimplePIM: A Software Framework for Productive and Efficient Processing-in-Memory

Jinfan Chen[1] Juan Gómez-Luna[1] Izzat El Hajj[2] Yuxin Guo[1] Onur Mutlu[1]
[1]ETH Zürich [2]American University of Beirut

Adoption: How to Ease **Programmability**? (IV)

- Geraldo F. Oliveira, Juan Gomez-Luna, Lois Orosa, Saugata Ghose, Nandita Vijaykumar, Ivan fernandez, Mohammad Sadrosadati, and Onur Mutlu,
  "DAMOV: A New Methodology and Benchmark Suite for Evaluating Data Movement Bottlenecks"
  IEEE Access, 8 September 2021.
  Preprint in ***arXiv***, 8 May 2021.
  [arXiv preprint]
  [IEEE Access version]
  [DAMOV Suite and Simulator Source Code]
  [SAFARI Live Seminar Video (2 hrs 40 mins)]
  [Short Talk Video (21 minutes)]

DAMOV: A New Methodology and Benchmark Suite for Evaluating Data Movement Bottlenecks

GERALDO F. OLIVEIRA, ETH Zürich, Switzerland
JUAN GÓMEZ-LUNA, ETH Zürich, Switzerland
LOIS OROSA, ETH Zürich, Switzerland
SAUGATA GHOSE, University of Illinois at Urbana–Champaign, USA
NANDITA VIJAYKUMAR, University of Toronto, Canada
IVAN FERNANDEZ, University of Malaga, Spain & ETH Zürich, Switzerland
MOHAMMAD SADROSADATI, ETH Zürich, Switzerland
ONUR MUTLU, ETH Zürich, Switzerland

Adoption: How to Ease **Programmability**? (V)

- **Appears in IEEE TETC 2023**

ALP: Alleviating CPU-Memory Data Movement Overheads in Memory-Centric Systems

Nika Mansouri Ghiasi, Nandita Vijaykumar, Geraldo F. Oliveira, Lois Orosa, Ivan Fernandez, Mohammad Sadrosadati, Konstantinos Kanellopoulos, Nastaran Hajinazar, Juan Gómez Luna, Onur Mutlu

Abstract—Recent advances in memory technology have enabled near-data processing (NDP) to tackle main memory bottlenecks in modern systems. Prior works partition applications into segments (e.g., instructions, loops, functions) and execute memory-bound segments of the applications on NDP computation units, while mapping the cache-friendly application segments to host CPU cores that access a deeper cache hierarchy. Partitioning applications between NDP and host cores causes inter-segment data movement overhead, which is the overhead from moving data generated from one segment and used in the consecutive segments. This overhead can be large if the segments map to cores in different parts of the system (i.e., host and NDP). Prior works take two approaches to the inter-segment data movement overhead when partitioning applications between NDP and host cores. The first class of works maps segments to NDP or host cores based on the properties of each segment, neglecting the performance impact of the inter-segment data movement. Such partitioning techniques suffer from inter-segment data movement overhead. The second class of works maps segments to host or NDP cores based on the overall memory bandwidth savings of each segment (which depends on the memory bandwidth savings within each segment and the inter-segment data movement overhead between other segments). These works do not offload each segment to the best-fitting core if they incur high inter-segment data movement overhead. Therefore these works miss some of the potential NDP performance benefits. We show that mapping each segment (here basic block) to its best-fitting core based on the properties of each segment, assuming no inter-segment data movement, can provide substantial performance benefits. However, we show that the inter-segment data movement reduces this benefit significantly.

To this end, we introduce ALP, a new programmer-transparent technique to leverage the performance benefits of NDP by *alleviating* the performance impact of inter-segment data movement between host and memory and enabling efficient partitioning of applications between host and NDP cores. ALP alleviates the inter-segment data movement overhead by *proactively and accurately* transferring the required data between the segments mapped on host and NDP cores. This is based on the key observation that the instructions that generate the inter-segment data stay the same across different executions of a program on different input sets. ALP uses a compiler pass to identify these instructions and uses specialized hardware support to transfer data between the host and NDP cores at runtime. Using both the compiler and runtime information, ALP efficiently maps application segments to either host or NDP cores considering 1) the properties of each segment, 2) the inter-segment data movement overhead between different segments, and 3) whether this inter-segment data movement overhead can be alleviated proactively and in a timely manner. We evaluate ALP across a wide range of workloads and show on average 54.3% and 45.4% speedup compared to executing the application only on the host CPU or only the NDP cores, respectively.

https://arxiv.org/pdf/2212.06292

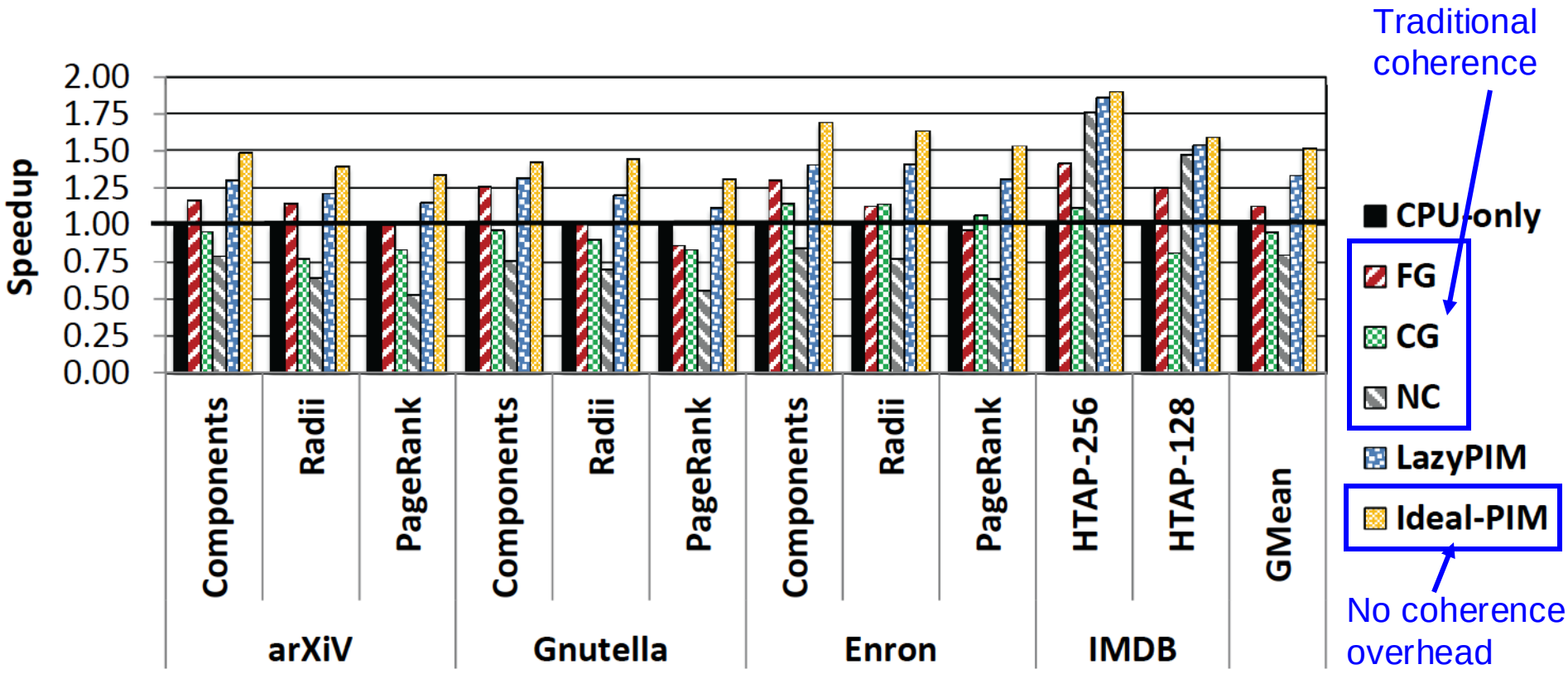
Adoption: How to Maintain **Coherence**? (I)

- Amirali Boroumand, Saugata Ghose, Minesh Patel, Hasan Hassan, Brandon Lucia, Kevin Hsieh, Krishna T. Malladi, Hongzhong Zheng, and Onur Mutlu,
  "LazyPIM: An Efficient Cache Coherence Mechanism for Processing-in-Memory"
  IEEE Computer Architecture Letters (***CAL***), June 2016.

LazyPIM: An Efficient Cache Coherence Mechanism for Processing-in-Memory

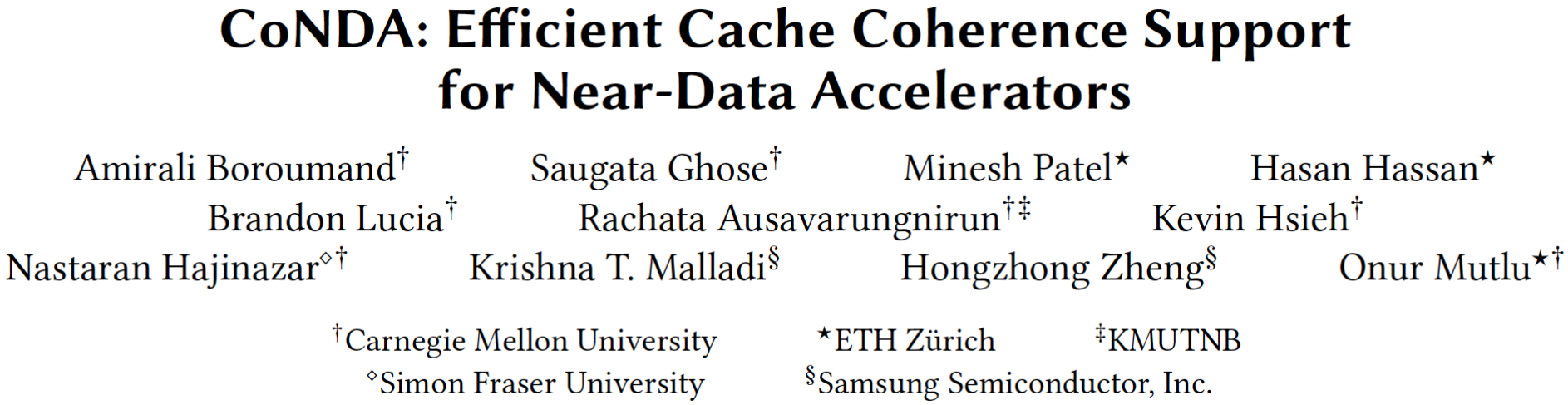
Amirali Boroumand[†], Saugata Ghose[†], Minesh Patel[†], Hasan Hassan[†§], Brandon Lucia[†],
Kevin Hsieh[†], Krishna T. Malladi[⋆], Hongzhong Zheng[⋆], and Onur Mutlu[‡†]
[†] *Carnegie Mellon University* [⋆] *Samsung Semiconductor, Inc.* [§] *TOBB ETÜ* [‡] *ETH Zürich*

Challenge: Coherence for Hybrid CPU-PIM Apps



Adoption: How to Maintain **Coherence**? (II)

- Amirali Boroumand, Saugata Ghose, Minesh Patel, Hasan Hassan, Brandon Lucia, Kevin Hsieh, Krishna T. Malladi, Hongzhong Zheng, and Onur Mutlu,
  "CoNDA: Efficient Cache Coherence Support for Near-Data Accelerators"
  *Proceedings of the 46th International Symposium on Computer Architecture (**ISCA**)*, Phoenix, AZ, USA, June 2019.



CoNDA: Efficient Cache Coherence Support for Near-Data Accelerators

Amirali Boroumand[†] Saugata Ghose[†] Minesh Patel[⋆] Hasan Hassan[⋆]
Brandon Lucia[†] Rachata Ausavarungnirun[†‡] Kevin Hsieh[†]
Nastaran Hajinazar[◇†] Krishna T. Malladi[§] Hongzhong Zheng[§] Onur Mutlu[⋆†]

[†]Carnegie Mellon University [⋆]ETH Zürich [‡]KMUTNB
[◇]Simon Fraser University [§]Samsung Semiconductor, Inc.

Adoption: How to Support **Synchronization**?

- Christina Giannoula, Nandita Vijaykumar, Nikela Papadopoulou, Vasileios Karakostas, Ivan Fernandez, Juan Gómez-Luna, Lois Orosa, Nectarios Koziris, Georgios Goumas, Onur Mutlu, **"SynCron: Efficient Synchronization Support for Near-Data-Processing Architectures"**
  *Proceedings of the 27th International Symposium on High-Performance Computer Architecture (**HPCA**)*, Virtual, February-March 2021.
  [Slides (pptx) (pdf)]
  [Short Talk Slides (pptx) (pdf)]
  [Talk Video (21 minutes)]
  [Short Talk Video (7 minutes)]

SynCron: Efficient Synchronization Support for Near-Data-Processing Architectures

Christina Giannoula[†‡] Nandita Vijaykumar[*‡] Nikela Papadopoulou[†] Vasileios Karakostas[†] Ivan Fernandez[§‡]
Juan Gómez-Luna[‡] Lois Orosa[‡] Nectarios Koziris[†] Georgios Goumas[†] Onur Mutlu[‡]
[†]*National Technical University of Athens* [‡]*ETH Zürich* [*]*University of Toronto* [§]*University of Malaga*

Adoption: How to Support **Virtual Memory**?

- Kevin Hsieh, Samira Khan, Nandita Vijaykumar, Kevin K. Chang, Amirali Boroumand, Saugata Ghose, and Onur Mutlu,
  "Accelerating Pointer Chasing in 3D-Stacked Memory: Challenges, Mechanisms, Evaluation"
  *Proceedings of the 34th IEEE International Conference on Computer Design (**ICCD**)*, Phoenix, AZ, USA, October 2016.

Accelerating Pointer Chasing in 3D-Stacked Memory: Challenges, Mechanisms, Evaluation

Kevin Hsieh[†] Samira Khan[‡] Nandita Vijaykumar[†]
Kevin K. Chang[†] Amirali Boroumand[†] Saugata Ghose[†] Onur Mutlu[§†]
[†]*Carnegie Mellon University* [‡]*University of Virginia* [§]*ETH Zürich*

Adoption: Evaluation Infrastructures

- Haocong Luo, Yahya Can Tugrul, F. Nisa Bostanci, Ataberk Olgun, A. Giray Yaglikci, and Onur Mutlu,
  "Ramulator 2.0: A Modern, Modular, and Extensible DRAM Simulator"
  Preprint on ***arxiv***, August 2023.
  [arXiv version]
  [Ramulator 2.0 Source Code]

https://arxiv.org/pdf/2308.11030.pdf

https://github.com/CMU-SAFARI/ramulator2

Processing-in-Memory: Challenges

To fully support PIM systems, we need to develop:

1. **Workload characterization methodologies and benchmark suites targeting PIM architectures**
2. **Frameworks that can facilitate the implementation of complex operations and algorithms using PIM primitives**
3. **Compiler support and compiler optimizations targeting PIM architectures**
4. **Operating system support for PIM-aware virtual memory, memory management, data allocation and mapping**
5. **End-to-End System-on-Chip Design Beyond DRAM**

The lack of tools and system support for PIM architectures limit the adoption of PIM systems

An Example: SimplePIM Framework

- Jinfan Chen, Juan Gómez-Luna, Izzat El Hajj, YuXin Guo, and Onur Mutlu,
  "SimplePIM: A Software Framework for Productive and Efficient Processing in Memory"
  *Proceedings of the 32nd International Conference on Parallel Architectures and Compilation Techniques (**PACT**)*, Vienna, Austria, October 2023.

SimplePIM: A Software Framework for Productive and Efficient Processing-in-Memory

Jinfan Chen[1] Juan Gómez-Luna[1] Izzat El Hajj[2] Yuxin Guo[1] Onur Mutlu[1]
[1]ETH Zürich [2]American University of Beirut

Executive Summary

- Real PIM hardware is now available, e.g., UPMEM PIM
- However, programming real PIM hardware is challenging, e.g., need to:
  - Distribute data across PIM memory banks,
  - Manage data transfers between host cores and PIM cores, between PIM cores, and between DRAM bank and PIM scratchpad
  - Launch PIM kernels on the PIM cores, etc.
  - Synchronize properly between threads
- SimplePIM is a high-level programming framework for real PIM hardware
  - Iterators such as `map`, `reduce`, and `zip`
  - Collective communication with `broadcast`, `scatter`, and `gather`
- Implementation on UPMEM and evaluation with six different workloads
  - Reduction, vector add, histogram, linear/logistic regression, K-means
  - **4.4x fewer lines of code compared to hand-optimized code**
  - Between **15% and 43% faster than hand-optimized code** for three workloads
- Source code: https://github.com/CMU-SAFARI/SimplePIM

Concluding Remarks

Challenge and Opportunity for Future

Fundamentally
Energy-Efficient
(Data-Centric)
Computing Architectures

Challenge and Opportunity for Future

Fundamentally
High-Performance
(Data-Centric)
Computing Architectures

Computing Architectures with Minimal Data Movement

Concluding Remarks

- It is time to design principled system architectures to solve the memory problem
- We must design systems to be **balanced, high-performance, and energy-efficient** → **memory-centric**
  - Enable computation capabilities in memory
- This can
  - Lead to **orders-of-magnitude** improvements
  - **Enable new applications & computing platforms**
  - **Enable better understanding of nature**
  - **…**
- Future of **truly memory-centric computing** is bright
  - We need to do research & design across the computing stack

Fundamentally Better Architectures

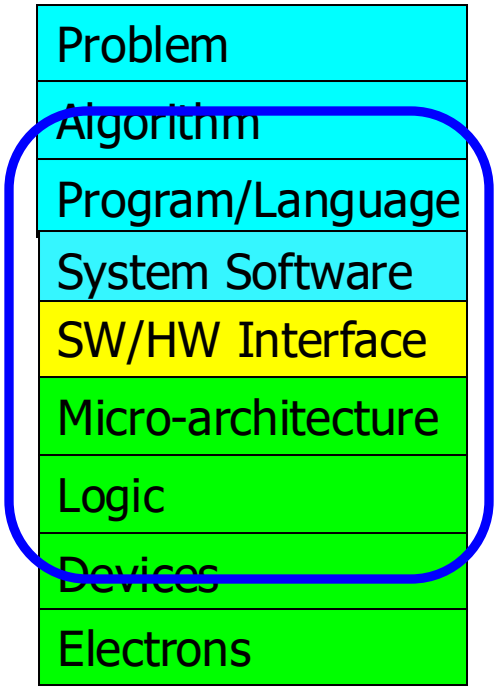
Data-centric

Data-driven

Data-aware

We Need to Revisit the Entire Stack

- With a **memory-centric mindset**



We can get there step by step

PIM Review and Open Problems

A Modern Primer on Processing in Memory

Onur Mutlu[a,b], Saugata Ghose[b,c], Juan Gómez-Luna[a], Rachata Ausavarungnirun[d]

SAFARI Research Group

[a] *ETH Zürich*
[b] *Carnegie Mellon University*
[c] *University of Illinois at Urbana-Champaign*
[d] *King Mongkut's University of Technology North Bangkok*

Onur Mutlu, Saugata Ghose, Juan Gomez-Luna, and Rachata Ausavarungnirun,
"A Modern Primer on Processing in Memory"
Invited Book Chapter in ***Emerging Computing: From Devices to Systems - Looking Beyond Moore and Von Neumann***, Springer, to be published in 2021.

https://arxiv.org/pdf/2012.03112.pdf

Referenced Papers, Talks, Artifacts

- All are available at

  https://people.inf.ethz.ch/omutlu/projects.htm

  https://www.youtube.com/onurmutlulectures

  https://github.com/CMU-SAFARI/

Acknowledgments

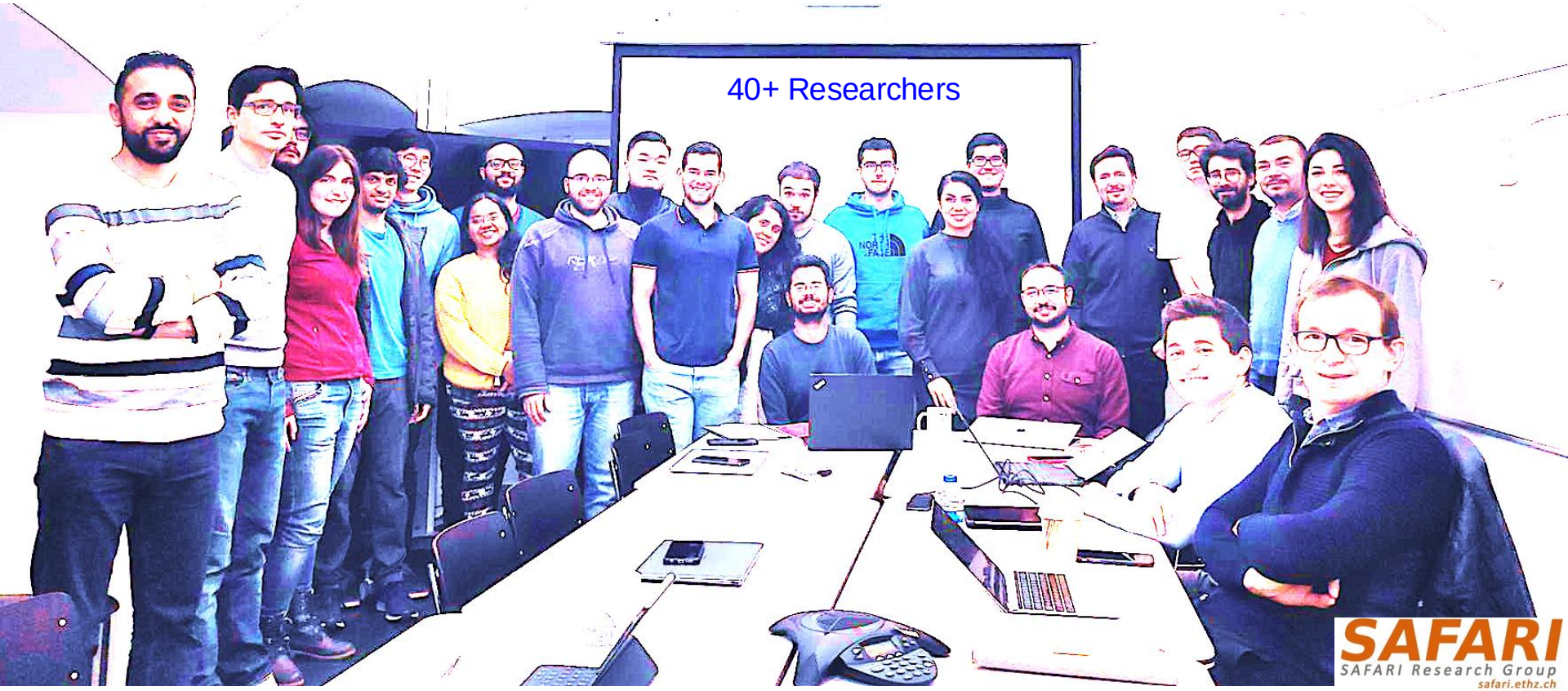
Think BIG, Aim HIGH!

https://safari.ethz.ch

SAFARI Research Group

Computer architecture, HW/SW, systems, bioinformatics, security, memory

https://safari.ethz.ch/safari-newsletter-january-2021/



Think BIG, Aim HIGH!

https://safari.ethz.ch

SAFARI Research Group: December 2021

- https://safari.ethz.ch/safari-newsletter-december-2021/

SAFARI
SAFARI Research Group

Think Big, Aim High

ETH zürich

View in your browser
December 2021

SAFARI Newsletter June 2023 Edition

- https://safari.ethz.ch/safari-newsletter-june-2023/

SAFARI
SAFARI Research Group

Think Big, Aim High

ETH zürich

View in your browser
June 2023

SAFARI Newsletter July 2024 Edition

- **https://safari.ethz.ch/safari-newsletter-july-2024/**

SAFARI Introduction & Research

Computer architecture, HW/SW, systems, bioinformatics, security, memory

SAFARI
SAFARI Research Group
safari.ethz.ch

Think BIG, Aim HIGH!

https://www.youtube.com/watch?v=mV2OuB2djEs

Open Source Tools: SAFARI GitHub

SAFARI Research Group at ETH Zurich and Carnegie Mellon University

Site for source code and tools distribution from SAFARI Research Group at ETH Zurich and Carnegie Mellon University.

440 followers | ETH Zurich and Carnegie Mellon U... | https://safari.ethz.ch/ | omutlu@gmail.com

Overview | Repositories 98 | Projects | Packages | People 13

ramulator Public

A Fast and Extensible DRAM Simulator, with built-in support for modeling many different DRAM technologies including DDRx, LPDDRx, GDDRx, WIOx, HBMx, and various academic proposals. Described in the...

C++ ☆ 519 206

prim-benchmarks Public

PrIM (Processing-In-Memory benchmarks) is the first benchmark suite for a real-world processing-in-memory (PIM) architecture. PrIM is developed to evaluate, analyze, and characterize the first publ...

C ☆ 125 45

MQSim Public

MQSim is a fast and accurate simulator modeling the performance of modern multi-queue (MQ) SSDs as well as traditional SATA based SSDs. MQSim faithfully models new high-bandwidth protocol implement...

C++ ☆ 265 144

rowhammer Public

Source code for testing the Row Hammer error mechanism in DRAM devices. Described in the ISCA 2014 paper by Kim et al. at http://users.ece.cmu.edu/~omutlu/pub/dram-row-hammer_isca14.pdf.

C ☆ 210 43

SoftMC Public

SoftMC is an experimental FPGA-based memory controller design that can be used to develop tests for DDR3 SODIMMs using a C++ based API. The design, the interface, and its capabilities and limitatio...

Verilog ☆ 120 27

Pythia Public

A customizable hardware prefetching framework using online reinforcement learning as described in the MICRO 2021 paper by Bera et al. (https://arxiv.org/pdf/2109.12021.pdf).

C++ ☆ 107 34

https://github.com/CMU-SAFARI/

ML/AI for Memory System Design & Memory System Design for AI/ML

Onur Mutlu
omutlu@gmail.com
https://people.inf.ethz.ch/omutlu
23 August 2024
Intel ArchFest

SAFARI

ETH zürich

Backup Slides

Processing-in-Memory: Challenges

To fully support PIM systems, we need to develop:

1. **Workload characterization methodologies and benchmark suites targeting PIM architectures**
2. **Frameworks that can facilitate the implementation of complex operations and algorithms using PIM primitives**
3. **Compiler support and compiler optimizations targeting PIM architectures**
4. **Operating system support for PIM-aware virtual memory, memory management, data allocation and mapping**
5. **End-to-End System-on-Chip Design Beyond DRAM**

The lack of tools and system support for PIM architectures limit the adoption of PIM systems

Security Issues in Processing in Memory

- Does PIM make security better or easier?
- Does PIM make security worse?
- Many interesting questions here
- Topic of a separate talk, but we highlight some papers
  - Evaluating Homomorphic Operations on a Real-World Processing-In-Memory System **[IISWC 2023]**
  - Amplifying Main Memory-Based Timing Covert and Side Channels using Processing-in-Memory Operations **[arxiv 2024]**

MIMDRAM

MIMDRAM: More Flexible Processing using DRAM

- **Appears at HPCA 2024** https://arxiv.org/pdf/2402.19080.pdf

MIMDRAM: An End-to-End Processing-Using-DRAM System for High-Throughput, Energy-Efficient and Programmer-Transparent Multiple-Instruction Multiple-Data Computing

Geraldo F. Oliveira[†] Ataberk Olgun[†] Abdullah Giray Yağlıkçı[†] F. Nisa Bostancı[†]
Juan Gómez-Luna[†] Saugata Ghose[‡] Onur Mutlu[†]

[†] *ETH Zürich* [‡] *Univ. of Illinois Urbana-Champaign*

*Our **goal** is to design a flexible PUD system that overcomes the limitations caused by the large and rigid granularity of PUD. To this end, we propose MIMDRAM, a hardware/software co-designed PUD system that introduces new mechanisms to allocate and control only the necessary resources for a given PUD operation. The key idea of MIMDRAM is to leverage fine-grained DRAM (i.e., the ability to independently access smaller segments of a large DRAM row) for PUD computation. MIMDRAM exploits this key idea to enable a multiple-instruction multiple-data (MIMD) execution model in each DRAM subarray (and SIMD execution within each DRAM row segment).*

Processing-in-Memory: Challenges

To fully support PIM systems, we need to develop:

1. Workload characterization methodologies and benchmark suites targeting PIM architectures
2. Frameworks that can facilitate the implementation of complex operations and algorithms using PIM primitives
3. **Compiler support and compiler optimizations targeting PIM architectures**
4. Operating system support for PIM-aware virtual memory, memory management, data allocation and mapping
5. End-to-End System-on-Chip Design Beyond DRAM

The lack of tools and system support for PIM architectures limit the adoption of PIM systems

MIMDRAM

An End-to-End Processing-Using-DRAM System for High-Throughput, Energy-Efficient and Programmer-Transparent Multiple-Instruction Multiple-Data Computing

Geraldo F. Oliveira

Ataberk Olgun | A. Giray Yağlıkçı | F. Nisa Bostancı

Saugata Ghose | Juan Gómez-Luna | Onur Mutlu

ETH zürich

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

SAFARI

Executive Summary

Problem: Processing-Using-DRAM (PUD) suffers from three issues caused by DRAM's large and rigid access granularity

- Underutilization due to data parallelism variation in (and across) applications
- Limited computation support due to a lack of interconnects
- Challenging programming model due to a lack of compilers

Goal: Design a flexible PUD system that overcomes the three limitations caused by DRAM's large and rigid access granularity

Key Mechanism: MIMDRAM, a hardware/software co-design PUD system

- **Key idea**: leverage fine-grained DRAM for PUD operation
- **HW**: - simple changes to the DRAM array, enabling concurrent PUD operations
  - low-cost interconnects at the DRAM peripherals for data reduction
- **SW**: - compiler and OS support to generate and map PUD instructions

Key Results: MIMDRAM achieves

- **14.3x**, **30.6x**, and **6.8x** the energy efficiency of state-of-the-art PUD systems, a high-end CPU and GPU, respectively
- Small area cost to a DRAM chip (**1.11%**) and CPU die (**0.6%**)

Recall: Processing using DRAM

- We can support
  - Bulk bitwise AND, OR, NOT, MAJ
  - Bulk bitwise COPY and INIT/ZERO
  - True Random Number Generation; Physical Unclonable Functions
  - Lookup Table based more complex computation
- At low cost
- Using analog computation capability of DRAM
  - Idea: activating (multiple) rows performs computation
    - Even in commodity off-the-shelf DRAM chips!

- 30-77X performance and energy improvement
  - Seshadri+, "Ambit: In-Memory Accelerator for Bulk Bitwise Operations Using Commodity DRAM Technology," MICRO 2017.
  - Seshadri+"RowClone: Fast and Efficient In-DRAM Copy and Initialization of Bulk Data," MICRO 2013.

Background Work: RowClone

- Vivek Seshadri, Yoongu Kim, Chris Fallin, Donghyuk Lee, Rachata Ausavarungnirun, Gennady Pekhimenko, Yixin Luo, Onur Mutlu, Michael A. Kozuch, Phillip B. Gibbons, and Todd C. Mowry,
  "RowClone: Fast and Energy-Efficient In-DRAM Bulk Data Copy and Initialization"
  *Proceedings of the 46th International Symposium on Microarchitecture (**MICRO**)*, Davis, CA, December 2013. [Slides (pptx) (pdf)] [Lightning Session Slides (pptx) (pdf)] [Poster (pptx) (pdf)]

RowClone: Fast and Energy-Efficient In-DRAM Bulk Data Copy and Initialization

Vivek Seshadri vseshadr@cs.cmu.edu
Yoongu Kim yoongukim@cmu.edu
Chris Fallin* cfallin@c1f.net
Donghyuk Lee donghyuk1@cmu.edu
Rachata Ausavarungnirun rachata@cmu.edu
Gennady Pekhimenko gpekhime@cs.cmu.edu
Yixin Luo yixinluo@andrew.cmu.edu
Onur Mutlu onur@cmu.edu
Phillip B. Gibbons† phillip.b.gibbons@intel.com
Michael A. Kozuch† michael.a.kozuch@intel.com
Todd C. Mowry tcm@cs.cmu.edu
Carnegie Mellon University †Intel Pittsburgh

Background Work: PiDRAM

- Ataberk Olgun, Juan Gomez Luna, Konstantinos Kanellopoulos, Behzad Salami, Hasan Hassan, Oguz Ergin, and Onur Mutlu,
  "PiDRAM: A Holistic End-to-end FPGA-based Framework for Processing-in-DRAM"
  ACM Transactions on Architecture and Code Optimization (***TACO***), March 2023.
  [arXiv version]
  Presented at the 18th HiPEAC Conference, Toulouse, France, January 2023.
  [Slides (pptx) (pdf)]
  [Longer Lecture Slides (pptx) (pdf)]
  [Lecture Video (40 minutes)]
  [PiDRAM Source Code]

PiDRAM: A Holistic End-to-end FPGA-based Framework for Processing-in-DRAM

Ataberk Olgun[§] Juan Gómez Luna[§] Konstantinos Kanellopoulos[§] Behzad Salami[§]
Hasan Hassan[§] Oğuz Ergin[†] Onur Mutlu[§]

[§]*ETH Zürich* [†]*TOBB University of Economics and Technology*

Background Work: In-DRAM Bulk AND/OR

- Vivek Seshadri, Kevin Hsieh, Amirali Boroumand, Donghyuk Lee, Michael A. Kozuch, Onur Mutlu, Phillip B. Gibbons, and Todd C. Mowry,
  "Fast Bulk Bitwise AND and OR in DRAM"
  IEEE Computer Architecture Letters (***CAL***), April 2015.

Fast Bulk Bitwise AND and OR in DRAM

Vivek Seshadri*, Kevin Hsieh*, Amirali Boroumand*, Donghyuk Lee*,
Michael A. Kozuch†, Onur Mutlu*, Phillip B. Gibbons†, Todd C. Mowry*

*Carnegie Mellon University †Intel Pittsburgh

Background Work: Ambit

- Vivek Seshadri, Donghyuk Lee, Thomas Mullins, Hasan Hassan, Amirali Boroumand, Jeremie Kim, Michael A. Kozuch, Onur Mutlu, Phillip B. Gibbons, and Todd C. Mowry,
  "Ambit: In-Memory Accelerator for Bulk Bitwise Operations Using Commodity DRAM Technology"
  *Proceedings of the 50th International Symposium on Microarchitecture (**MICRO**)*, Boston, MA, USA, October 2017.
  [Slides (pptx) (pdf)] [Lightning Session Slides (pptx) (pdf)] [Poster (pptx) (pdf)]

Ambit: In-Memory Accelerator for Bulk Bitwise Operations Using Commodity DRAM Technology

Vivek Seshadri[1,5] Donghyuk Lee[2,5] Thomas Mullins[3,5] Hasan Hassan[4] Amirali Boroumand[5]
Jeremie Kim[4,5] Michael A. Kozuch[3] Onur Mutlu[4,5] Phillip B. Gibbons[5] Todd C. Mowry[5]

[1]Microsoft Research India [2]NVIDIA Research [3]Intel [4]ETH Zürich [5]Carnegie Mellon University

Background: In-DRAM Bulk Bitwise Execution

- Vivek Seshadri and Onur Mutlu,
  "In-DRAM Bulk Bitwise Execution Engine"
  Invited Book Chapter in Advances in Computers, to appear in 2020.
  [Preliminary arXiv version]

In-DRAM Bulk Bitwise Execution Engine

Vivek Seshadri
Microsoft Research India
visesha@microsoft.com

Onur Mutlu
ETH Zürich
onur.mutlu@inf.ethz.ch

Recall: SIMDRAM Framework

- Nastaran Hajinazar, Geraldo F. Oliveira, Sven Gregorio, Joao Dinis Ferreira, Nika Mansouri Ghiasi, Minesh Patel, Mohammed Alser, Saugata Ghose, Juan Gomez-Luna, and Onur Mutlu, **"SIMDRAM: An End-to-End Framework for Bit-Serial SIMD Computing in DRAM"** *Proceedings of the 26th International Conference on Architectural Support for Programming Languages and Operating Systems* (***ASPLOS***), Virtual, March-April 2021.
  [2-page Extended Abstract]
  [Short Talk Slides (pptx) (pdf)]
  [Talk Slides (pptx) (pdf)]
  [Short Talk Video (5 mins)]
  [Full Talk Video (27 mins)]

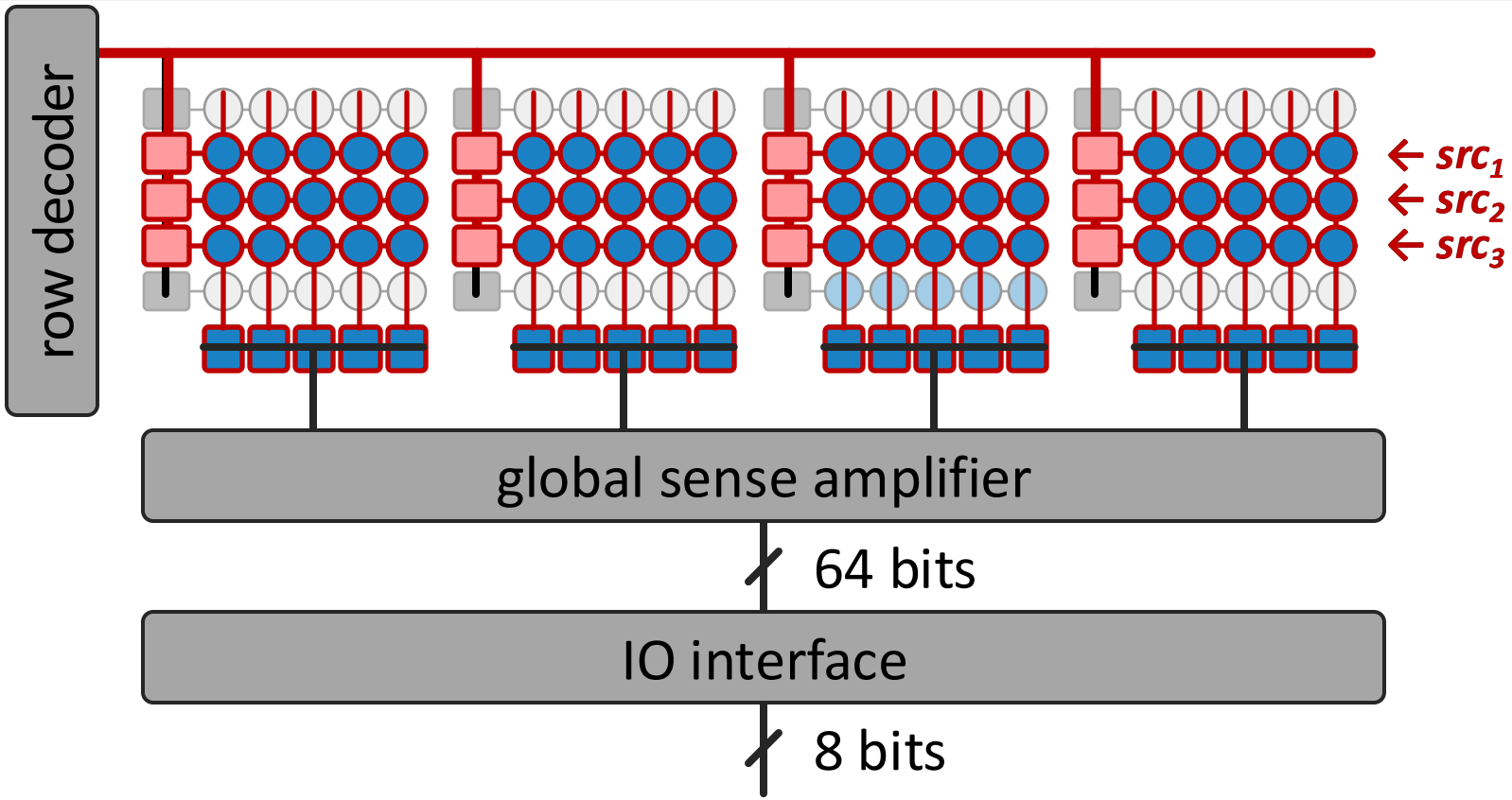
SIMDRAM: A Framework for Bit-Serial SIMD Processing using DRAM

*Nastaran Hajinazar[1,2] *Geraldo F. Oliveira[1] Sven Gregorio[1] João Dinis Ferreira[1]
Nika Mansouri Ghiasi[1] Minesh Patel[1] Mohammed Alser[1] Saugata Ghose[3]
Juan Gómez-Luna[1] Onur Mutlu[1]

[1]ETH Zürich [2]Simon Fraser University [3]University of Illinois at Urbana–Champaign

Background:
In-DRAM Copy/Init, Majority & NOT Operations

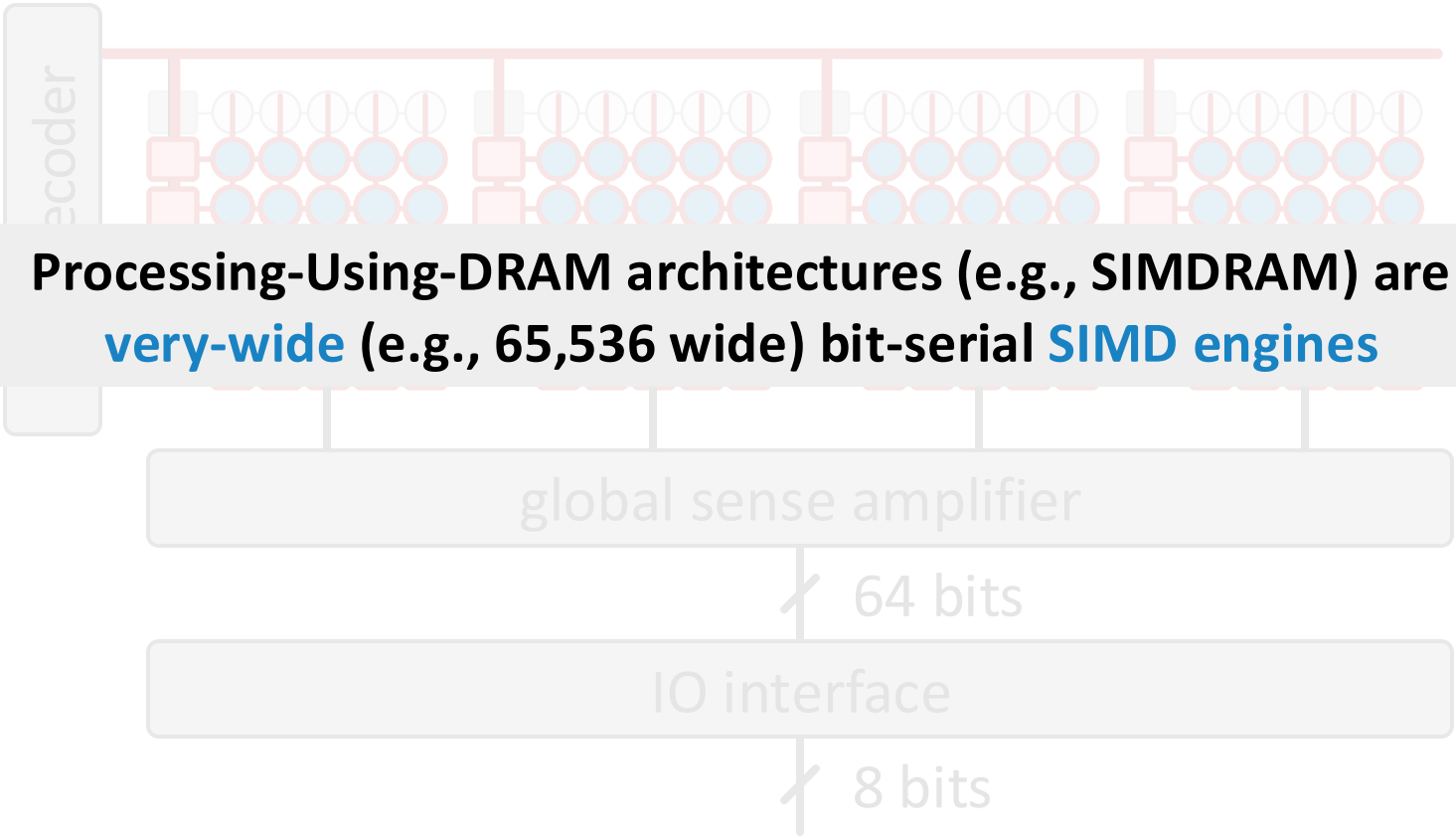
In-DRAM majority is performed by
simultaneously activating three DRAM rows



Seshadri, Vivek, et al. " **Ambit: In-Memory Accelerator for Bulk Bitwise Operations Using Commodity DRAM Technology**," in MICRO, 2017

Background:
In-DRAM Majority Operations

Seshadri, Vivek, et al. "**Ambit: In-Memory Accelerator for Bulk Bitwise Operations Using Commodity DRAM Technology**," in MICRO, 2017



Processing-Using-DRAM architectures (e.g., SIMDRAM) are very-wide (e.g., 65,536 wide) bit-serial SIMD engines

Oliveira, Geraldo F., et al. " **SIMDRAM: An End-to-End Framework for Bit-Serial SIMD Computing in DRAM**," in ASPLOS, 2021

Limitations of PUD Systems: Overview

PUD systems suffer from three sources of inefficiency due to the large and rigid DRAM access granularity

1 SIMD Underutilization

- due to data parallelism variation within and across applications
- leads to throughput and energy waste

2 Limited Computation Support

- due to a lack of low-cost interconnects across columns
- limits PUD operations to only parallel map constructs

3 Challenging Programming Model

- due to a lack of compiler support for PUD systems
- creates a burden on programmers, limiting PUD adoption

Limitations of PUD Systems:
Challenging Programming Model

Programmer's Tasks:

Map & align data structures

Goal:

Just write my kernel

High-level code for

C[i] = (A[i] > pred[i])? A[i] + B[i] : A[i] – B[i]

```
for (int i = 0; i < size ; ++ i){
   bool cond = A[i] > pred[i];
   if (cond) C[i] = A[i] + B[i];
   else  C[i] = A[i] - B[i];
}
```

Limitations of PUD Systems:
Challenging Programming Model

Programmer's Tasks:

- Map & align data structures
- Identify array boundaries

Goal:

- Just write my kernel

High-level code for

C[i] = (A[i] > pred[i])? A[i] + B[i] : A[i] – B[i]

```
for (int i = 0; i < size ; ++ i){
   bool cond = A[i] > pred[i];
   if (cond) C[i] = A[i] + B[i];
   else  C[i] = A[i] - B[i];
}
```

Limitations of PUD Systems:
Challenging Programming Model

Programmer's Tasks:

Map & align data structures

Identify array boundaries

Manually unroll loop

Map C to PUD instructions

Goal:

Just write my kernel

High-level code for

C[i] = (A[i] > pred[i])? A[i] + B[i] : A[i] – B[i]

```
for (int i = 0; i < size ; ++ i){
   bool cond = A[i] > pred[i];
   if (cond) C[i] = A[i] + B[i];
   else  C[i] = A[i] - B[i];
}
```

Limitations of PUD Systems:
Challenging Programming Model

Programmer's Tasks:

Map & align data structures	Identify array boundaries	Manually unroll loop	Map C to PUD instructions	Orchestrate data movement

Goal: Just write my kernel

High-level code for

C[i] = (A[i] > pred[i])? A[i] + B[i] : A[i] – B[i]

```
for (int i = 0; i < size ; ++ i){
   bool cond = A[i] > pred[i];
   if (cond) C[i] = A[i] + B[i];
   else  C[i] = A[i] - B[i];
}
```

Limitations of PUD Systems:
Challenging Programming Model

Programmer's Tasks:

Map & align data structures	Identify array boundaries	Manually unroll loop	Map C to PUD instructions	Orchestrate data movement	**Goal:** Just write my kernel

PUD's assembly-like code for

C[i] = (A[i] > pred[i])? A[i] + B[i] : A[i] – B[i]

```
bbop_trsp_init(A , size , elm_size);
bbop_trsp_init(B , size , elm_size);
bbop_trsp_init(C , size , elm_size);

bbop_add(D , A , B , size , elm_size);
bbop_sub(E , A , B , size , elm_size);
bbop_greater(F , A , pred , size , elm_size);
bbop_if_else(C , D , E , F , size , elm_size);
```

Problem & Goal

Problem

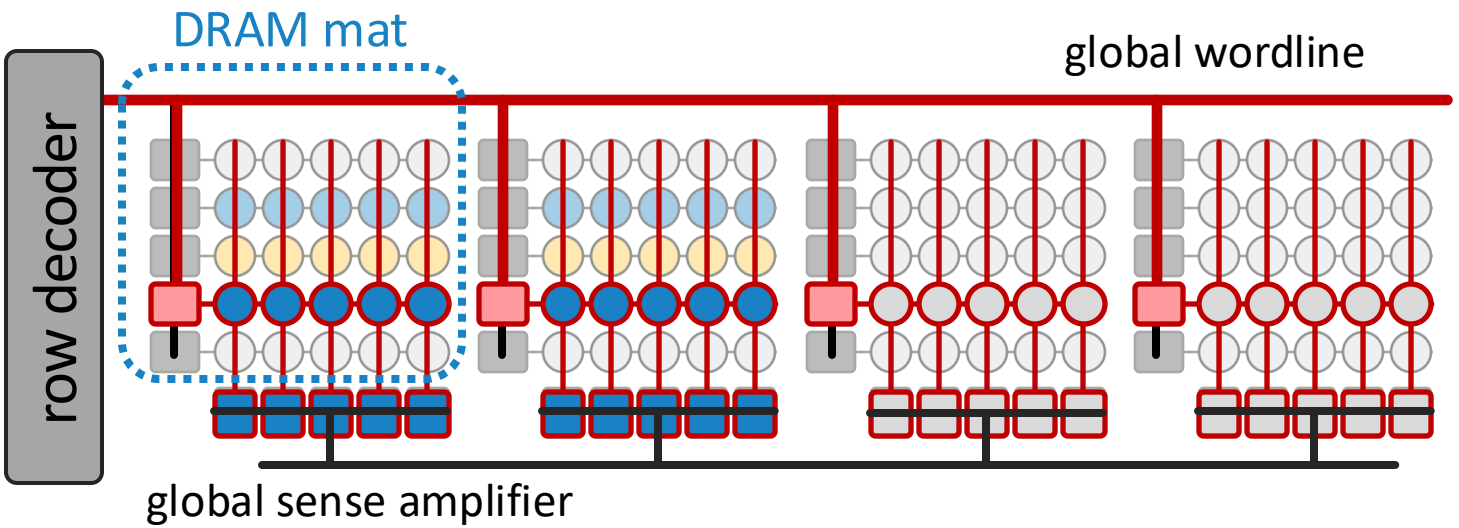
Processing-Using-DRAM's large and rigid granularity limits its applicability and efficiency for different applications

Goal

Design a flexible PUD system that overcomes the three limitations caused by large and rigid DRAM access granularity

MIMDRAM:
Key Idea (I)

DRAM's hierarchical organization can enable <u>fine-grained access</u>



Key Issue:

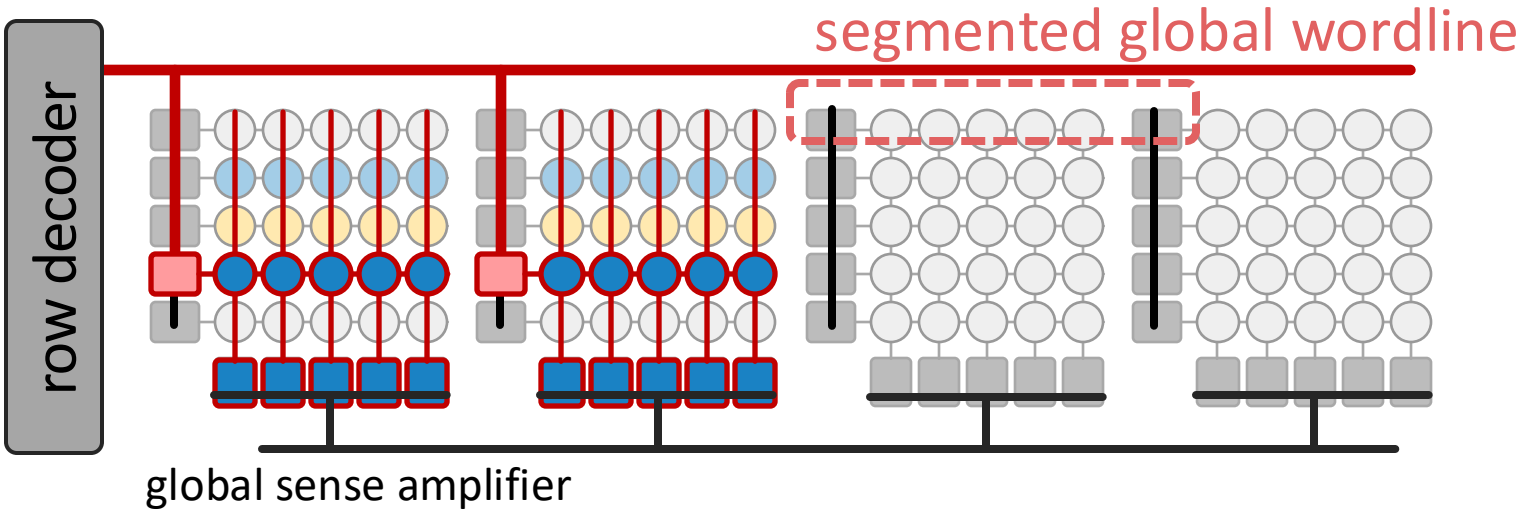
on a DRAM access, the global wordline propagates across all DRAM mats

Fine-Grained DRAM:

segments the global wordline to access individual DRAM mats

MIMDRAM:
Key Idea (II)

Fine-Grained DRAM:
segments the global wordline to access individual DRAM mats



Fine-grained DRAM for energy-efficient DRAM access:

[Cooper-Balis+, 2010]: Fine-Grained Activation for Power Reduction in DRAM
[Udipi+, 2010]: Rethinking DRAM Design and Organization for Energy-Constrained Multi-Cores
[Zhang+, 2014]: Half-DRAM
[Ha+, 2016]: Improving Energy Efficiency of DRAM by Exploiting Half Page Row Access
[O'Connor+, 2017]: Fine-Grained DRAM
[Olgun+, 2024]: Sectored DRAM

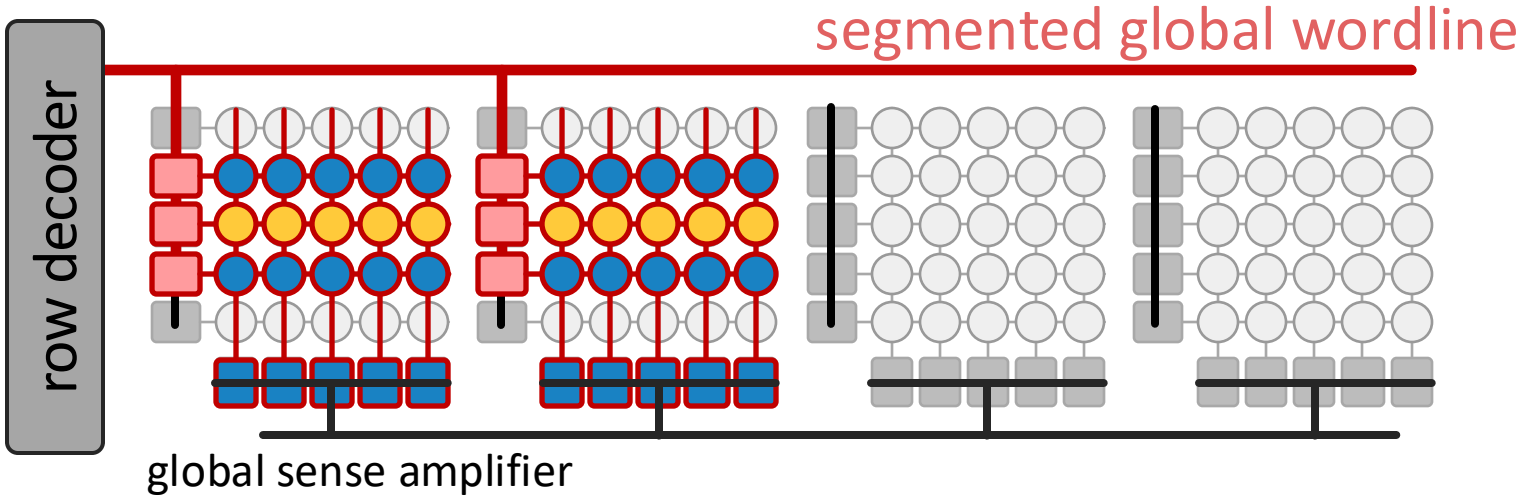
Sectored DRAM

- Ataberk Olgun, F. Nisa Bostanci, Geraldo F. Oliveira, Yahya Can Tugrul, Rahul Bera, A. Giray Yaglikci, Hasan Hassan, Oguz Ergin, and Onur Mutlu, **"Sectored DRAM: A Practical Energy-Efficient and High-Performance Fine-Grained DRAM Architecture"**
  ACM Transactions on Architecture and Code Optimization (***TACO***), [online] June 2024.
  [arXiv version]
  [ACM Digital Library version]

Sectored DRAM: A Practical Energy-Efficient and High-Performance Fine-Grained DRAM Architecture

Ataberk Olgun[§] F. Nisa Bostancı[§†] Geraldo F. Oliveira[§] Yahya Can Tuğrul[§†] Rahul Bera[§]
A. Giray Yağlıkcı[§] Hasan Hassan[§] Oğuz Ergin[†] Onur Mutlu[§]

MIMDRAM:
Key Idea (III)

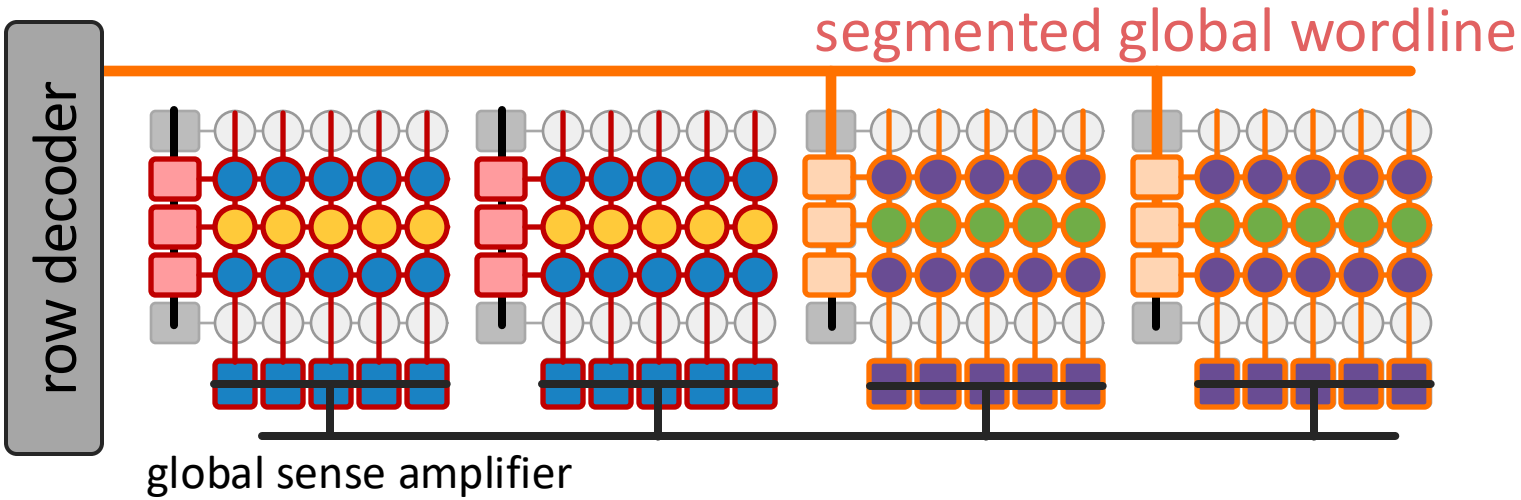


Fine-grained DRAM for processing-using-DRAM:

1 Improves SIMD utilization

- for a single PUD operation, only access the DRAM mats with target data

MIMDRAM:
Key Idea (III)

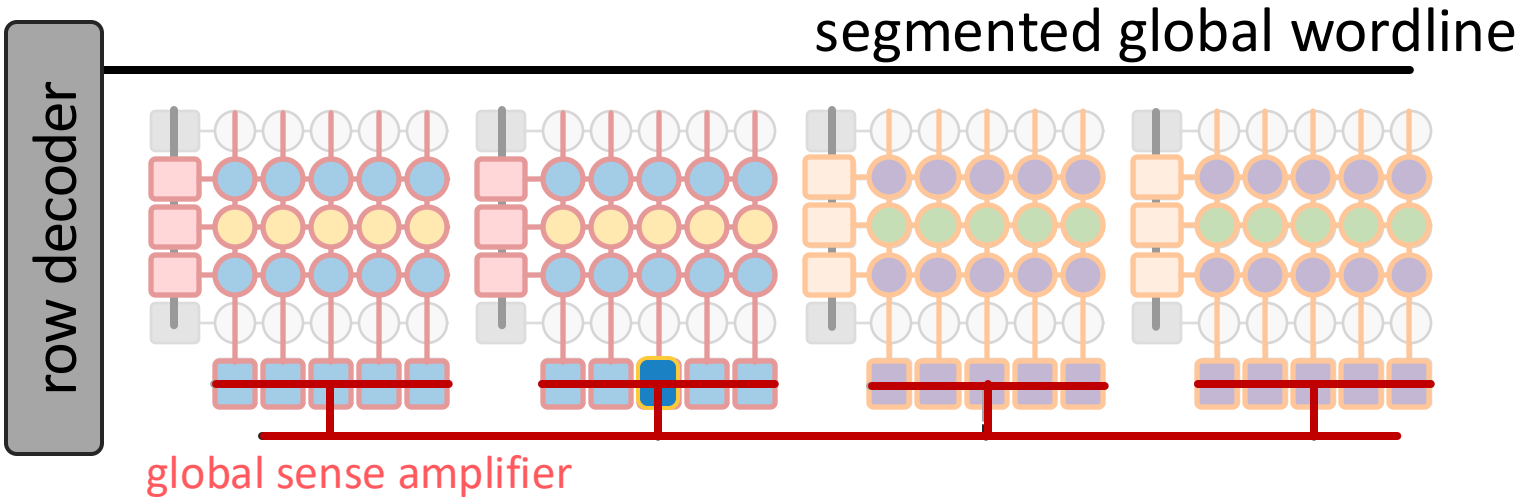


Fine-grained DRAM for processing-using-DRAM:

1 Improves SIMD utilization

- for a single PUD operation, only access the DRAM mats with target data
- for multiple PUD operations, execute independent operations concurrently
  → **multiple instruction, multiple data (MIMD) execution model**

MIMDRAM:
Key Idea (III)



Fine-grained DRAM for processing-using-DRAM:

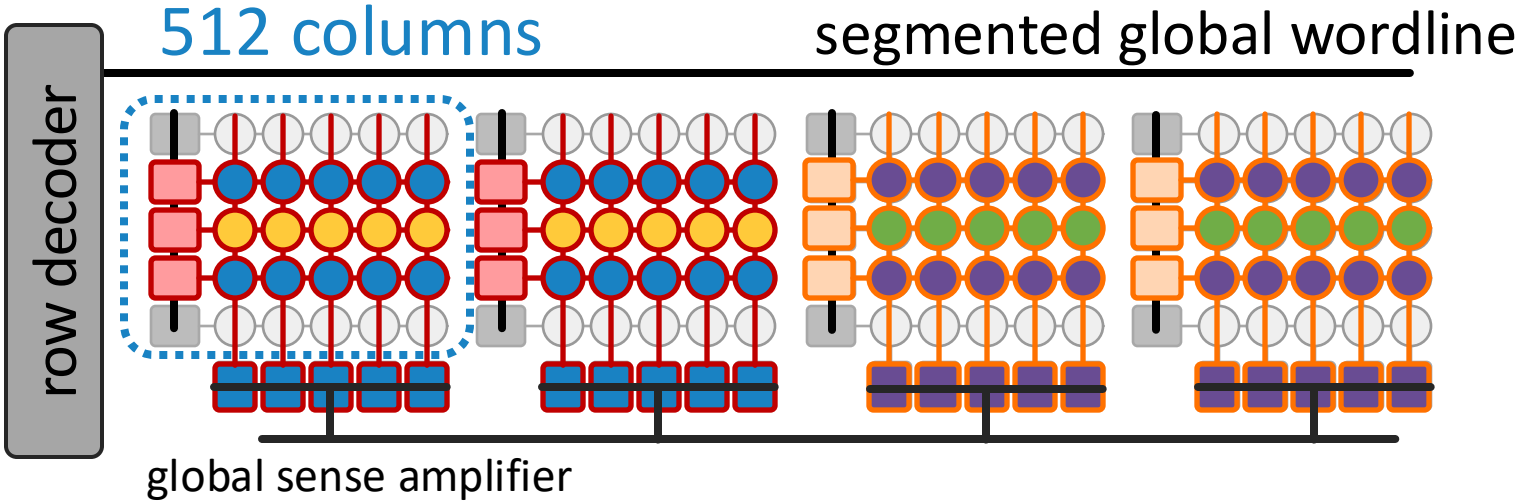
1 Improves SIMD utilization

- for a single PUD operation, only access the DRAM mats with target data
- for multiple PUD operations, execute independent operations concurrently
  → multiple instruction, multiple data (MIMD) execution model

2 Enables low-cost interconnects for vector reduction

- global and local data buses can be used for inter-/intra-mat communication

MIMDRAM:
Key Idea (III)



Fine-grained DRAM for processing-using-DRAM:

1 Improves SIMD utilization
- for a single PUD operation, only access the DRAM mats with target data
- for multiple PUD operations, execute independent operations concurrently
  → **multiple instruction, multiple data (MIMD) execution model**

2 Enables low-cost interconnects for vector reduction
- global and local data buses can be used for inter-/intra-mat communication

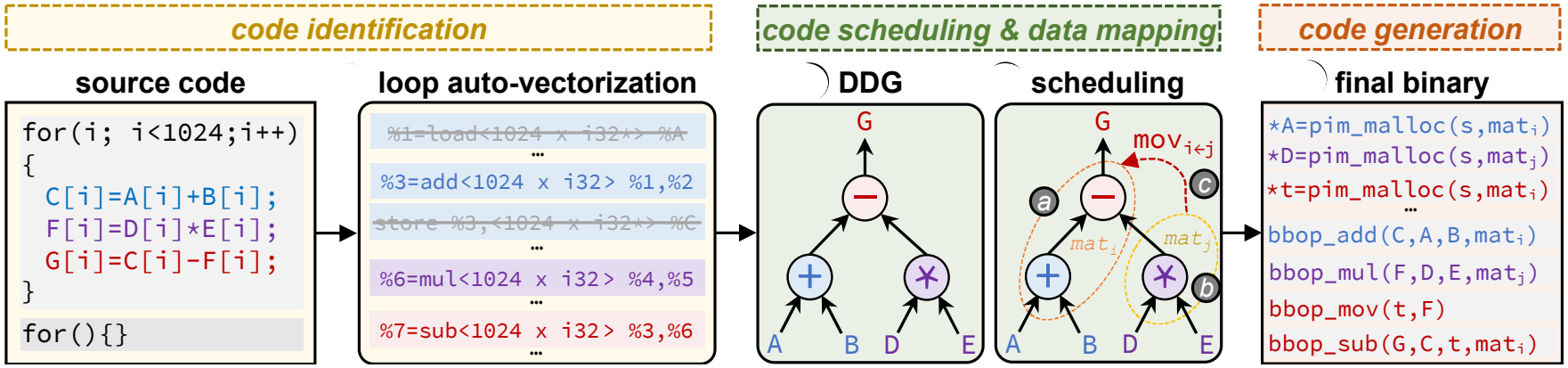
3 Eases programmability
- SIMD parallelism in a DRAM mat is on par with vector ISAs' SIMD width

MIMDRAM:
Compiler Support (I)

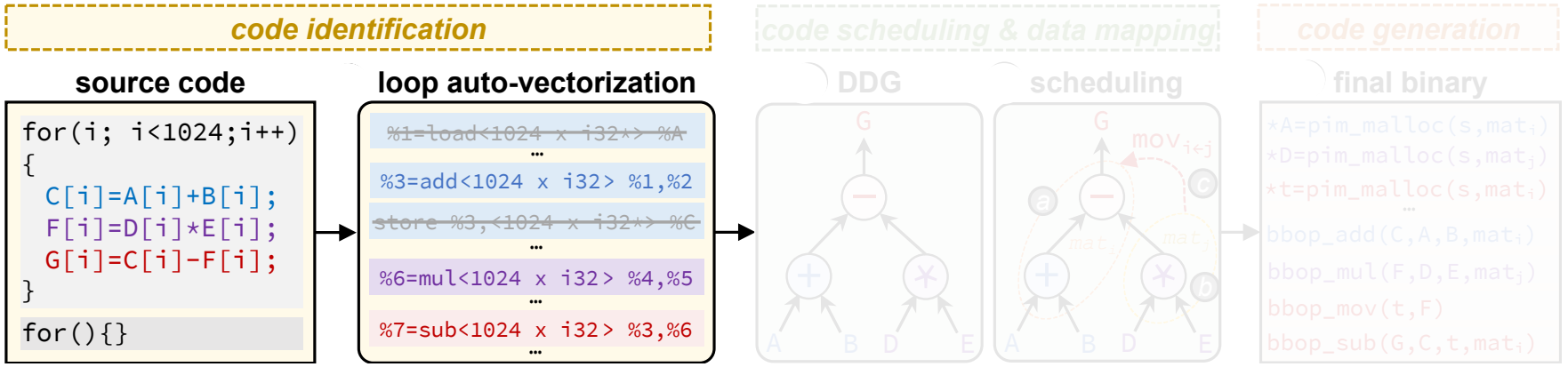
Goal

Transparently:
extract SIMD parallelism from an application, and
schedule PUD instructions while maximizing utilization

Three new LLVM-based passes targeting PUD execution



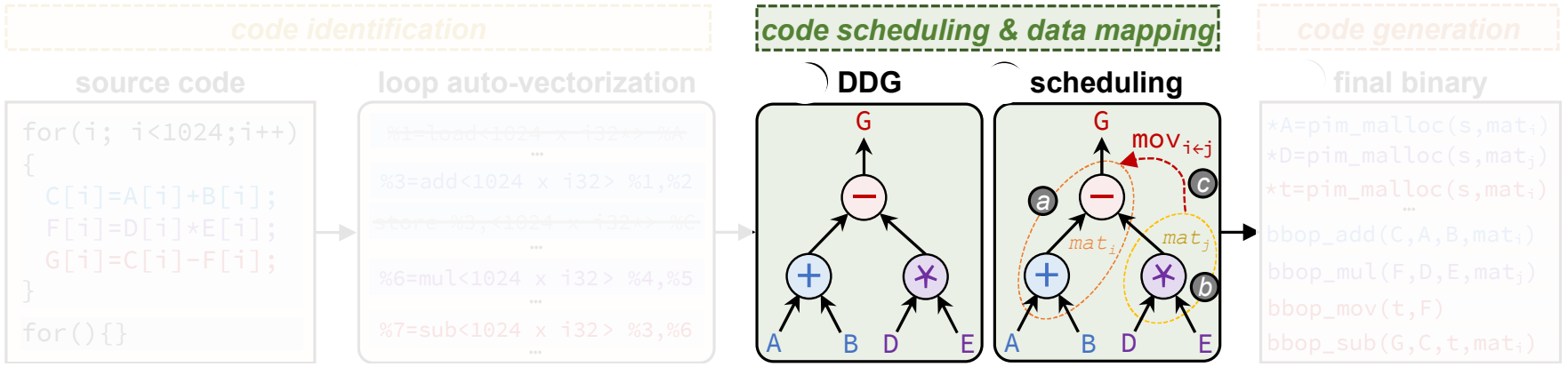
MIMDRAM:
Compiler Support (II)



Goal

Identify SIMD parallelism, generate PUD instructions, and set the appropriate vectorization factor

MIMDRAM:
Compiler Support (II)



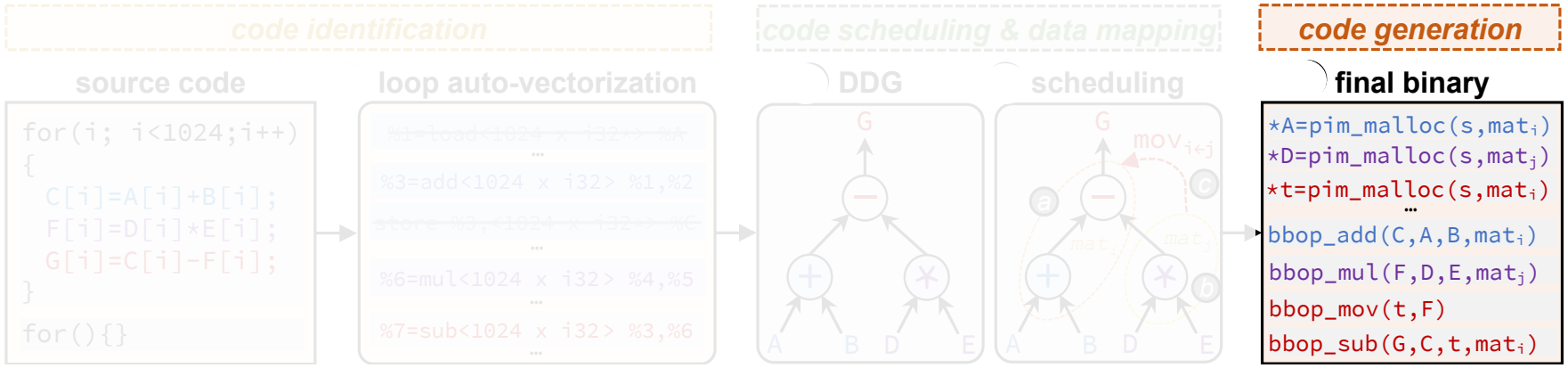
Goal

Identify SIMD parallelism, generate PUD instructions, and set the appropriate vectorization factor

Goal

Improve SIMD utilization by allowing the distribution of independent PUD instructions across DRAM mats

MIMDRAM:
Compiler Support (III)



Goal: Identify SIMD parallelism, generate PUD instructions, and set the appropriate vectorization factor

Goal: Improve SIMD utilization by allowing the distribution of independent PUD instructions across DRAM mats

Goal: **Generate the appropriate binary for data allocation and PUD instructions**

MIMDRAM:
System Support

- Instruction set architecture
- Execution & data transposition
- Data coherence
- Address translation
- Data allocation & alignment
- Mat label translation

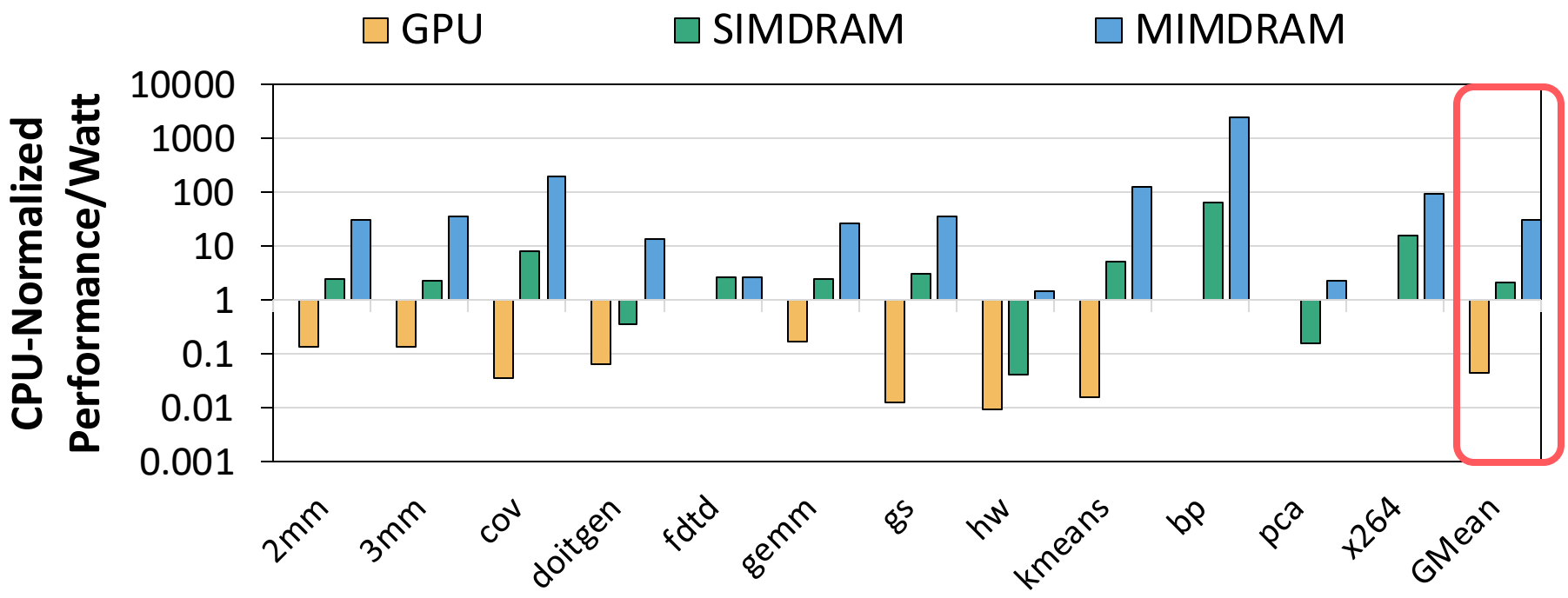
Evaluation:
Methodology Overview

- **Evaluation Setup**
  - **CPU:** Intel Skylake CPU
  - **GPU:** NVIDIA A100 GPU
  - **PUD:** SIMDRAM [Oliveira+, 2021] and DRISA [Li+, 2017]
  - **PND:** Fulcrum [Lenjani+, 2020]
  - **https://github.com/CMU-SAFARI/MIMDRAM**

- **Workloads:**
  - 12 workloads from Polybench, Rodinia, Phoenix, and SPEC2017
  - 495 multi-programmed application mixes

- **Two-Level Analysis**
  - **Single application** → leverages intra-application data parallelism
  - **Multi-programmed workload** → leverages inter-application data parallelism

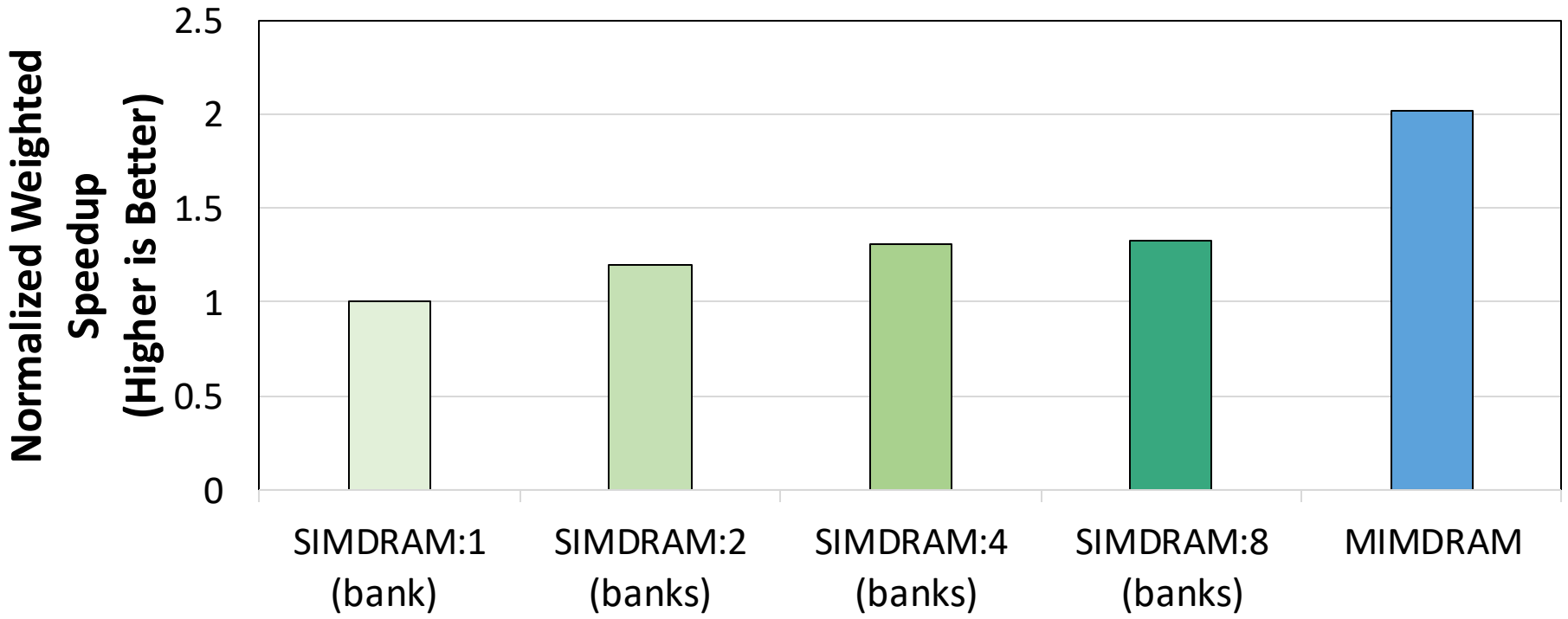
Evaluation:
Single Application Analysis – Energy Efficiency



Takeaway

MIMDRAM significantly improves energy efficiency compared to CPU (30.6x), GPU (6.8x), and SIMDRAM (14.3x)

Evaluation:
Multi-Programmed Workload Analysis



Takeaway

MIMDRAM significantly improves system throughput (1.68x) compared to SIMDRAM

Evaluation:
More in the Paper

- **MIMDRAM with subarray and bank-level parallelism**
  - MIMDRAM provides significant performance gains compared to the baseline CPU (**13.2x**) and GPU (**2x**)

- **Comparison to DRISA and Fulcrum for multi-programmed workloads**
  - MIMDRAM achieves system throughput on par with DRISA and Fulcrum

- **MIMDRAM's SIMD utilization versus SIMDRAM**
  - MIMDRAM provides **15.6x** the utilization of SIMDRAM

- **Area analysis**
  - MIMDRAM adds small area cost to a DRAM chip (**1.11%**) and CPU die (**0.6%**)

MIMDRAM: Summary

We introduced MIMDRAM,
a hardware/software co-designed processing-using-DRAM system

- **Key idea**: leverage fine-grained DRAM for processing-using-DRAM operation
- **HW**: - simple changes to DRAM, enabling concurrent instruction execution
  - low-cost interconnects at the DRAM peripherals for data reduction
- **SW**: - compiler and OS support to generate and map instructions

Our evaluation demonstrates that MIMDRAM

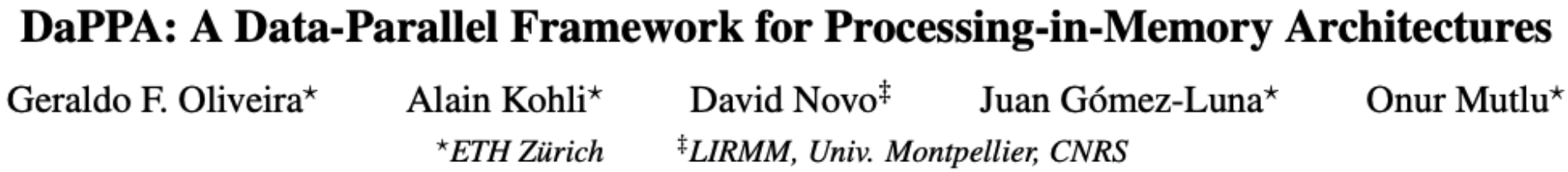
- **significantly** improves **performance**, **energy efficiency**, and **throughput** compared to processor-centric (CPU and GPU) and memory-centric (SIMDRAM, DRISA, and Fulcrum) architectures
- incurs **small area cost** to a DRAM chip and CPU die

https://github.com/CMU-SAFARI/MIMDRAM

Two Other Works on PIM Programmability

Adoption: How to Ease **Programmability**? (I)

- Geraldo F. Oliveira, Alain Kohli, David Novo, Juan Gómez-Luna, Onur Mutlu, "**DaPPA: A Data-Parallel Framework for Processing-in-Memory Architectures,**" in *PACT SRC Student Competition,* Vienna, Austria, October 2023.



DaPPA: A Data-Parallel Framework for Processing-in-Memory Architectures

Geraldo F. Oliveira⋆ Alain Kohli⋆ David Novo‡ Juan Gómez-Luna⋆ Onur Mutlu⋆

⋆*ETH Zürich* ‡*LIRMM, Univ. Montpellier, CNRS*

Adoption: How to Ease **Programmability**? (II)

- Jinfan Chen, Juan Gómez-Luna, Izzat El Hajj, YuXin Guo, and Onur Mutlu,
  "SimplePIM: A Software Framework for Productive and Efficient Processing in Memory"
  *Proceedings of the 32nd International Conference on Parallel Architectures and Compilation Techniques (**PACT**)*, Vienna, Austria, October 2023.

SimplePIM: A Software Framework for Productive and Efficient Processing-in-Memory

Jinfan Chen[1] Juan Gómez-Luna[1] Izzat El Hajj[2] Yuxin Guo[1] Onur Mutlu[1]
[1]ETH Zürich [2]American University of Beirut

SimplePIM

Adoption: How to Ease **Programmability**? (II)

- Jinfan Chen, Juan Gómez-Luna, Izzat El Hajj, YuXin Guo, and Onur Mutlu,
  "SimplePIM: A Software Framework for Productive and Efficient Processing in Memory"
  *Proceedings of the 32nd International Conference on Parallel Architectures and Compilation Techniques (**PACT**)*, Vienna, Austria, October 2023.

SimplePIM: A Software Framework for Productive and Efficient Processing-in-Memory

Jinfan Chen[1] Juan Gómez-Luna[1] Izzat El Hajj[2] Yuxin Guo[1] Onur Mutlu[1]
[1]ETH Zürich [2]American University of Beirut

SimplePIM:

A Software Framework for Productive and Efficient Processing-in-Memory

Jinfan Chen, Juan Gómez Luna, Izzat El Hajj, Yuxin Guo, Onur Mutlu

https://arxiv.org/pdf/2310.01893.pdf

https://github.com/CMU-SAFARI/SimplePIM

ETH Zürich

SAFARI

Executive Summary

- Processing-in-Memory (PIM) promises to alleviate the *data movement bottleneck*
- Real PIM hardware is now available, e.g., UPMEM PIM
- However, programming real PIM hardware is challenging, e.g.:
  - Distribute data across PIM memory banks,
  - Manage data transfers between host cores and PIM cores, and between PIM cores,
  - Launch PIM kernels on the PIM cores, etc.
- SimplePIM is a high-level programming framework for real PIM hardware
  - Iterators such as `map`, `reduce`, and `zip`
  - Collective communication with `broadcast`, `scatter`, and `gather`
- Implementation on UPMEM and evaluation with six different workloads
  - Reduction, vector add, histogram, linear/logistic regression, K-means
  - **4.4x fewer lines of code compared to hand-optimized code**
  - Between **15% and 43% faster than hand-optimized code** for three workloads
- Source code: https://github.com/CMU-SAFARI/SimplePIM

Real DRAM Chips Are Already Quite Capable: FC-DRAM & SiMRA

DRAM Chips Are Already (Quite) Capable!

- **Appears at HPCA 2024** https://arxiv.org/pdf/2402.18736.pdf

Functionally-Complete Boolean Logic in Real DRAM Chips: Experimental Characterization and Analysis

İsmail Emir Yüksel Yahya Can Tuğrul Ataberk Olgun F. Nisa Bostancı A. Giray Yağlıkçı
Geraldo F. Oliveira Haocong Luo Juan Gómez-Luna Mohammad Sadrosadati Onur Mutlu

ETH Zürich

We experimentally demonstrate that COTS DRAM chips are capable of performing 1) functionally-complete *Boolean operations: NOT, NAND, and NOR and 2) many-input (i.e., more than two-input) AND and OR operations. We present an extensive characterization of* new *bulk bitwise operations in 256 off-the-shelf modern DDR4 DRAM chips. We evaluate the reliability of these operations using a metric called success rate: the fraction of correctly performed bitwise operations. Among our 19 new observations, we highlight four major results. First, we can perform the NOT operation on COTS DRAM chips with 98.37% success rate on average. Second, we can perform up to 16-input NAND, NOR, AND, and OR operations on COTS DRAM chips with high reliability (e.g., 16-input NAND, NOR, AND, and OR with average success rate of 94.94%, 95.87%, 94.94%, and 95.85%, respectively). Third, data pattern only slightly*

DRAM Chips Are Already (Quite) Capable!

- **https://arxiv.org/pdf/2312.02880.pdf**

PULSAR: Simultaneous Many-Row Activation for Reliable and High-Performance Computing in Off-the-Shelf DRAM Chips

Ismail Emir Yuksel Yahya Can Tugrul F. Nisa Bostanci Abdullah Giray Yaglikci Ataberk Olgun
Geraldo F. Oliveira Melina Soysal Haocong Luo Juan Gomez Luna Mohammad Sadrosadati
Onur Mutlu

ETH Zurich

We propose PULSAR, a new technique to enable high-success-rate and high-performance PuM operations in off-the-shelf DRAM chips. PULSAR leverages our new observation that a carefully-crafted sequence of DRAM commands simultaneously activates up to 32 DRAM rows. PULSAR overcomes the limitations of existing techniques by 1) replicating the input data to improve the success rate and 2) enabling new bulk bitwise operations (e.g., many-input majority, *Multi-RowInit*, and *Bulk-Write*) to improve the performance.

DRAM Chips Are Already (Quite) Capable!

- **Appears at DSN 2024**

Simultaneous Many-Row Activation in Off-the-Shelf DRAM Chips: Experimental Characterization and Analysis

İsmail Emir Yüksel[1] Yahya Can Tuğrul[1,2] F. Nisa Bostancı[1] Geraldo F. Oliveira[1]
A. Giray Yağlıkçı[1] Ataberk Olgun[1] Melina Soysal[1] Haocong Luo[1]
Juan Gómez-Luna[1] Mohammad Sadrosadati[1] Onur Mutlu[1]
[1]*ETH Zürich* [2]*TOBB University of Economics and Technology*

https://arxiv.org/pdf/2405.06081

The Capability of COTS DRAM Chips

We **demonstrate** that **COTS DRAM chips:**

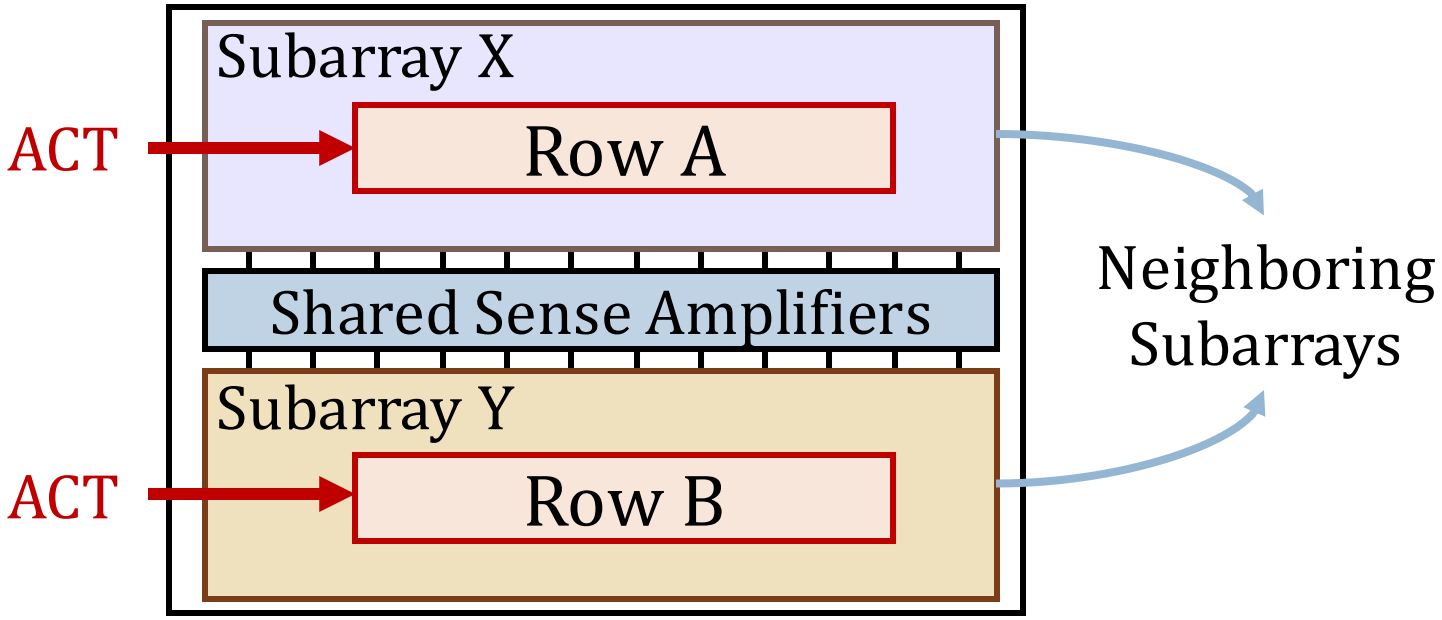
1 Can **simultaneously activate** up to **48 rows** in **two neighboring subarrays**

2 Can perform **NOT operation** with up to **32 output operands**

3 Can perform up to **16-input AND, NAND, OR, and NOR** operations

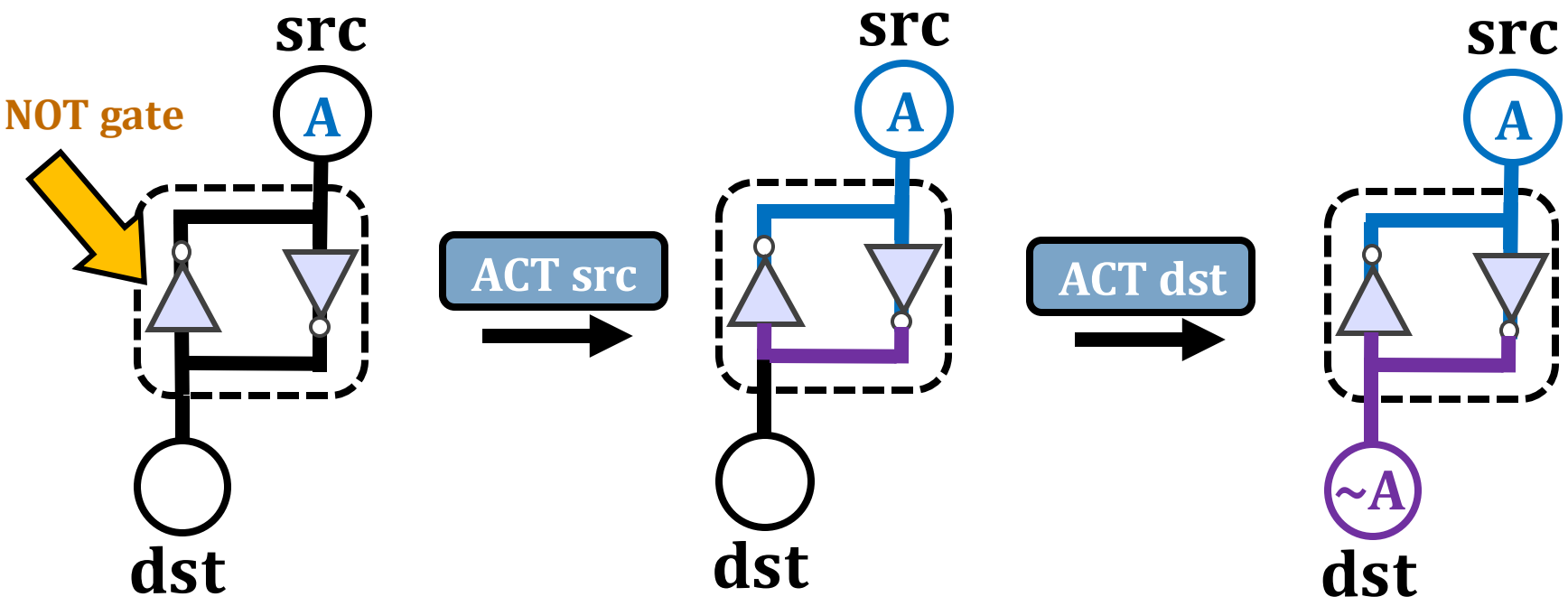
Finding: SiMRA Across Subarrays

Activating two rows in **quick succession**
can **simultaneously** activate
multiple rows in neighboring subarrays



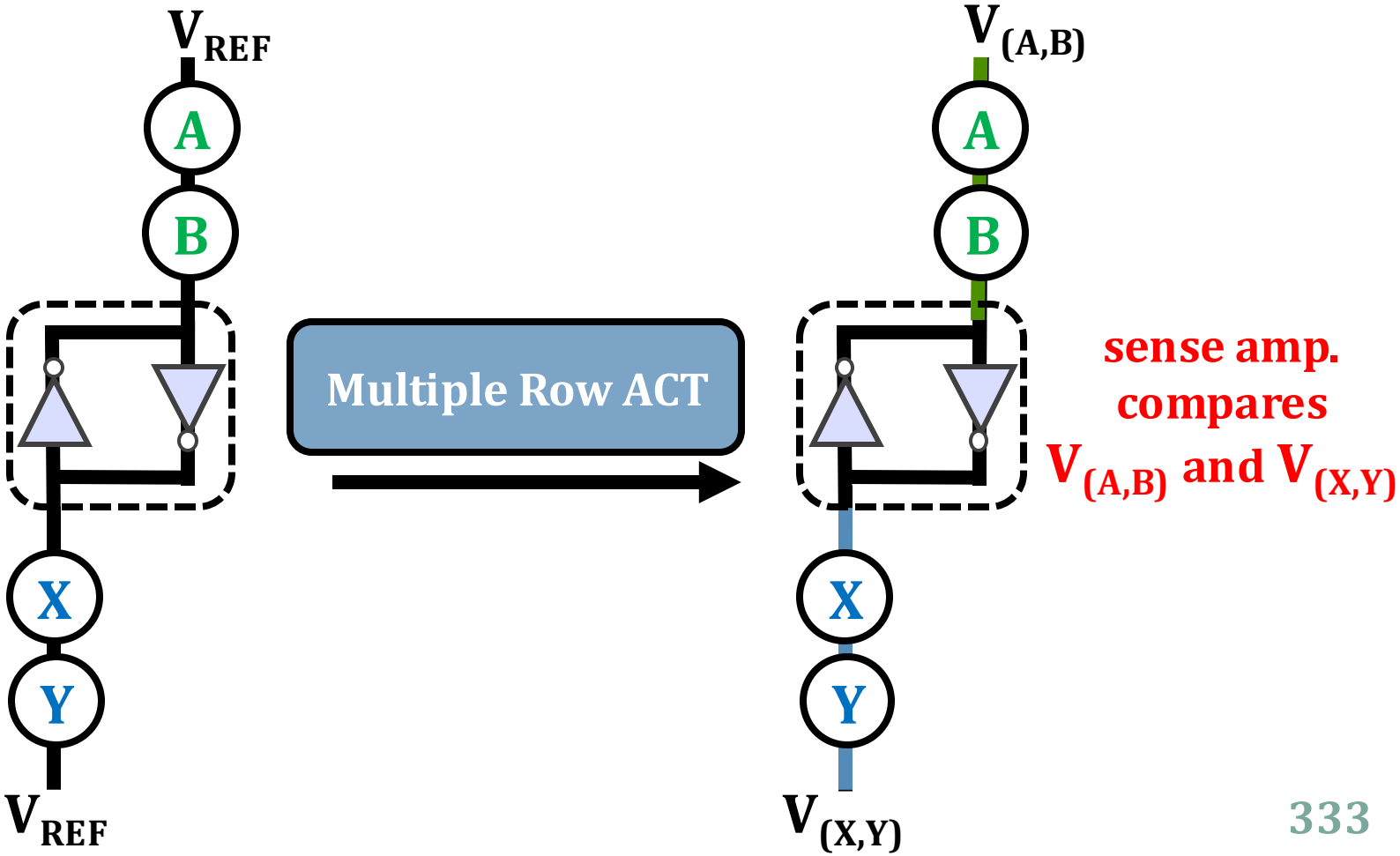
Key Idea: NOT Operation

Connect rows in neighboring subarrays
through **a NOT gate** by simultaneously activating rows

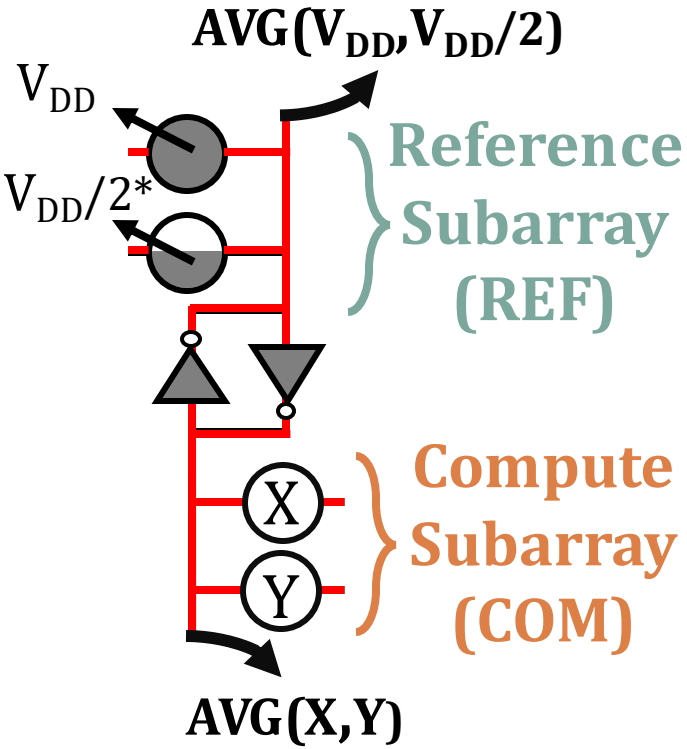


Key Idea: NAND, NOR, AND, OR

Manipulate the bitline voltage to express
a wide variety of functions using
multiple-row activation in neighboring subarrays

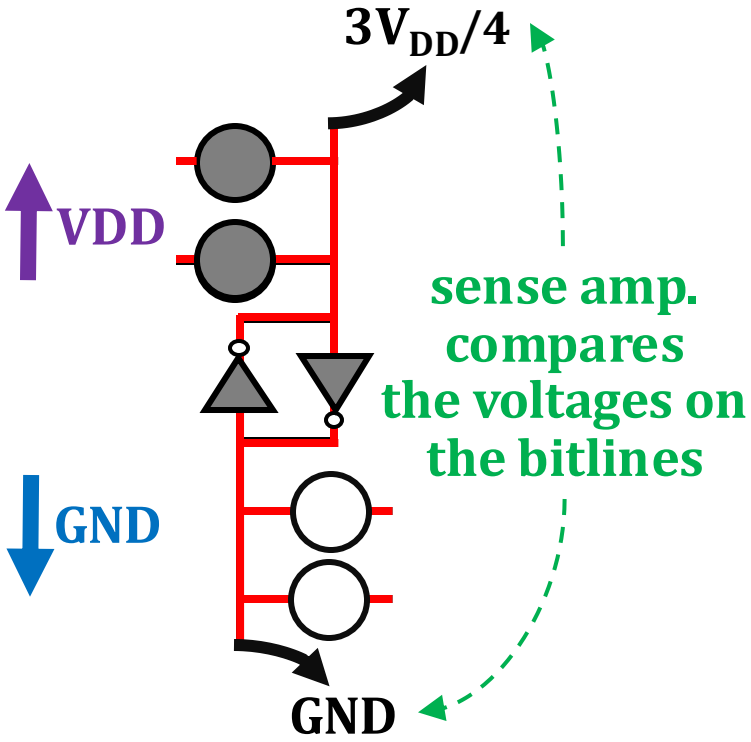


Two-Input AND and NAND Operations



*Gao et al., "FracDRAM: Fractional Values in Off-the-Shelf DRAM," in MICRO, 2022.

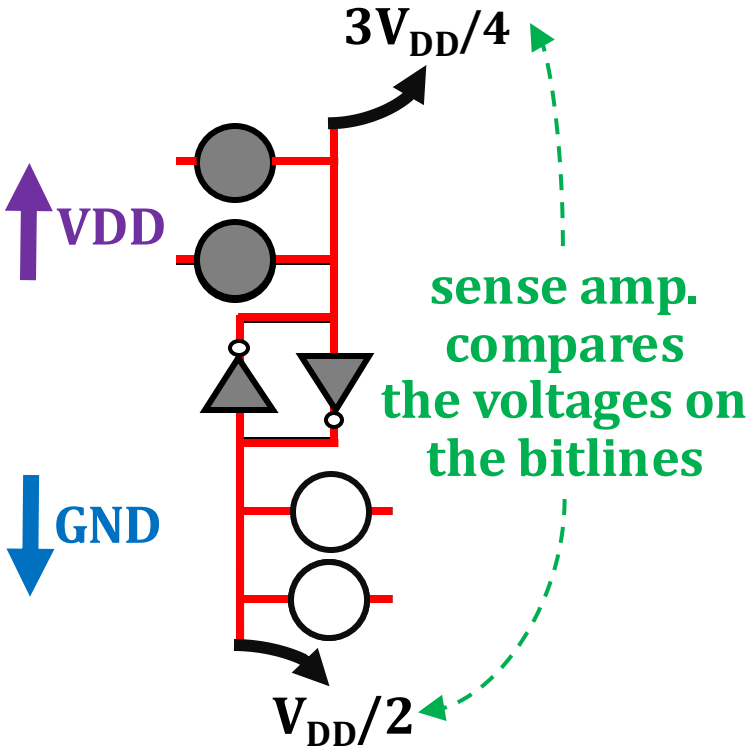
Two-Input AND and NAND Operations



V_{DD}=1 & GND = 0

X	Y	COM	REF
0	0	0	1

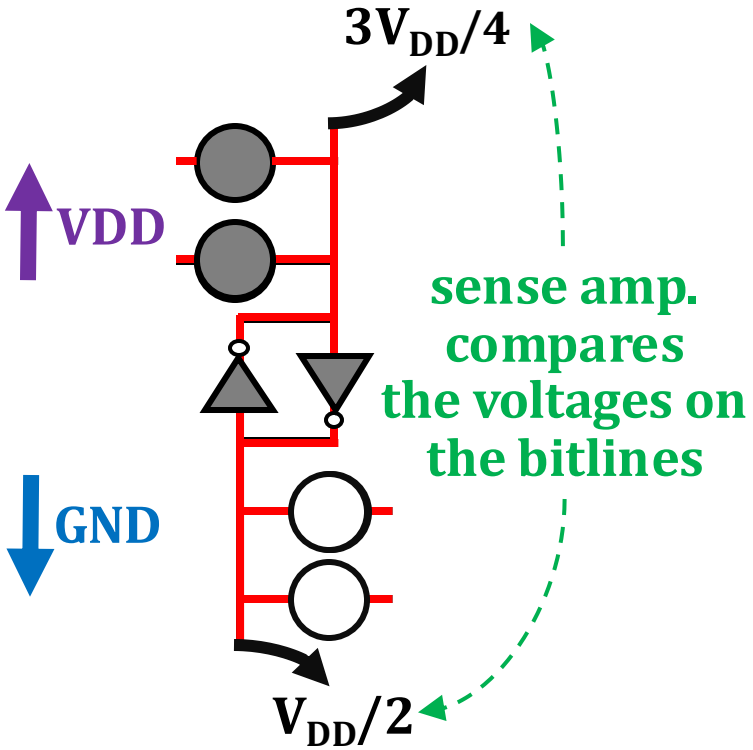
Two-Input AND and NAND Operations



V_{DD}=1 & GND = 0

X	Y	COM	REF
0	0	0	1
0	1	0	1

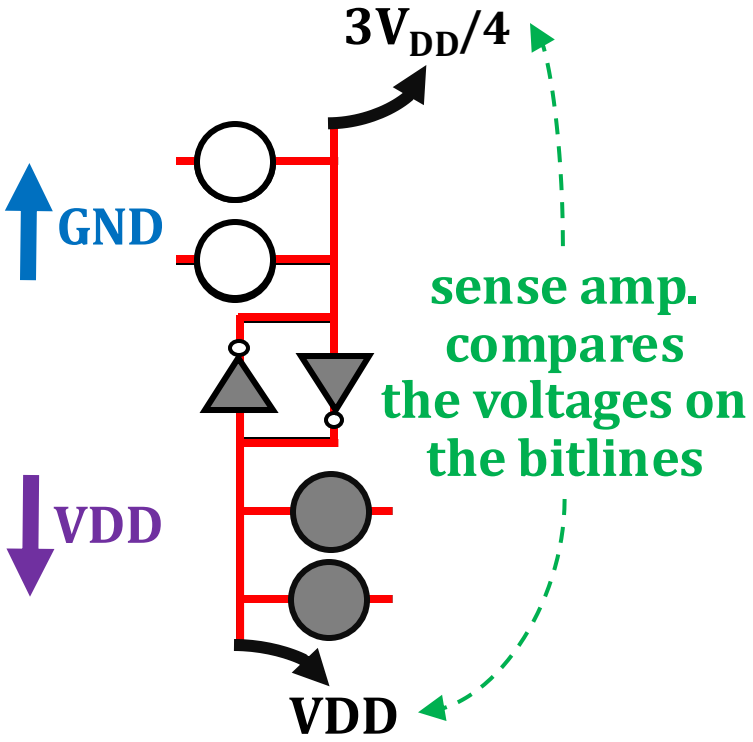
Two-Input AND and NAND Operations



V_{DD}=1 & GND = 0

X	Y	COM	REF
0	0	0	1
0	1	0	1
1	0	0	1

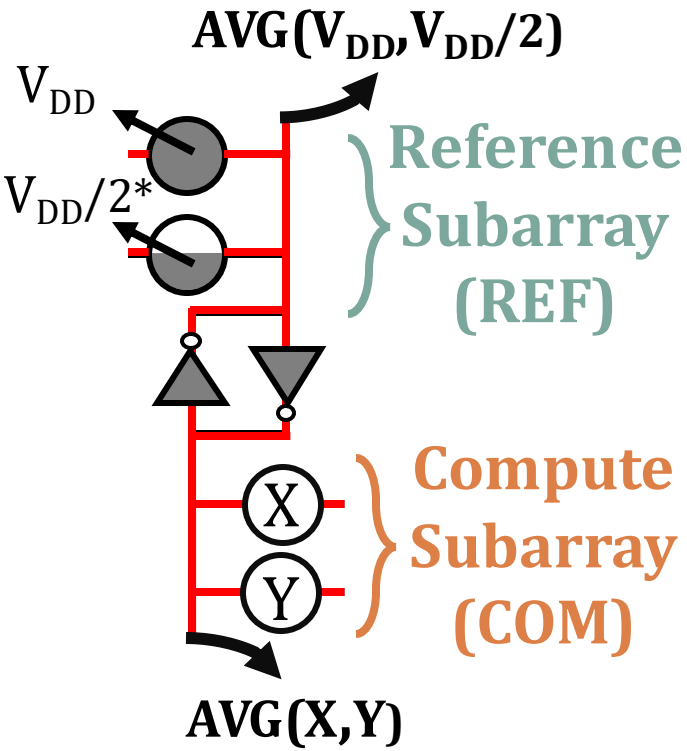
Two-Input AND and NAND Operations



V_{DD}=1 & GND = 0

X	Y	COM	REF
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

Two-Input AND and NAND Operations



V_{DD}=1 & GND = 0

X	Y	COM	REF
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0
		AND	NAND

*Gao et al., "FracDRAM: Fractional Values in Off-the-Shelf DRAM," in MICRO, 2022.

Many-Input AND, NAND, OR, and NOR Operations

We can express AND, NAND, OR, and NOR operations by carefully manipulating the **reference voltage**

Functionally-Complete Boolean Logic in Real DRAM Chips: Experimental Characterization and Analysis

İsmail Emir Yüksel Yahya Can Tuğrul Ataberk Olgun F. Nisa Bostancı A. Giray Yağlıkçı
Geraldo F. Oliveira Haocong Luo Juan Gómez-Luna Mohammad Sadrosadati Onur Mutlu

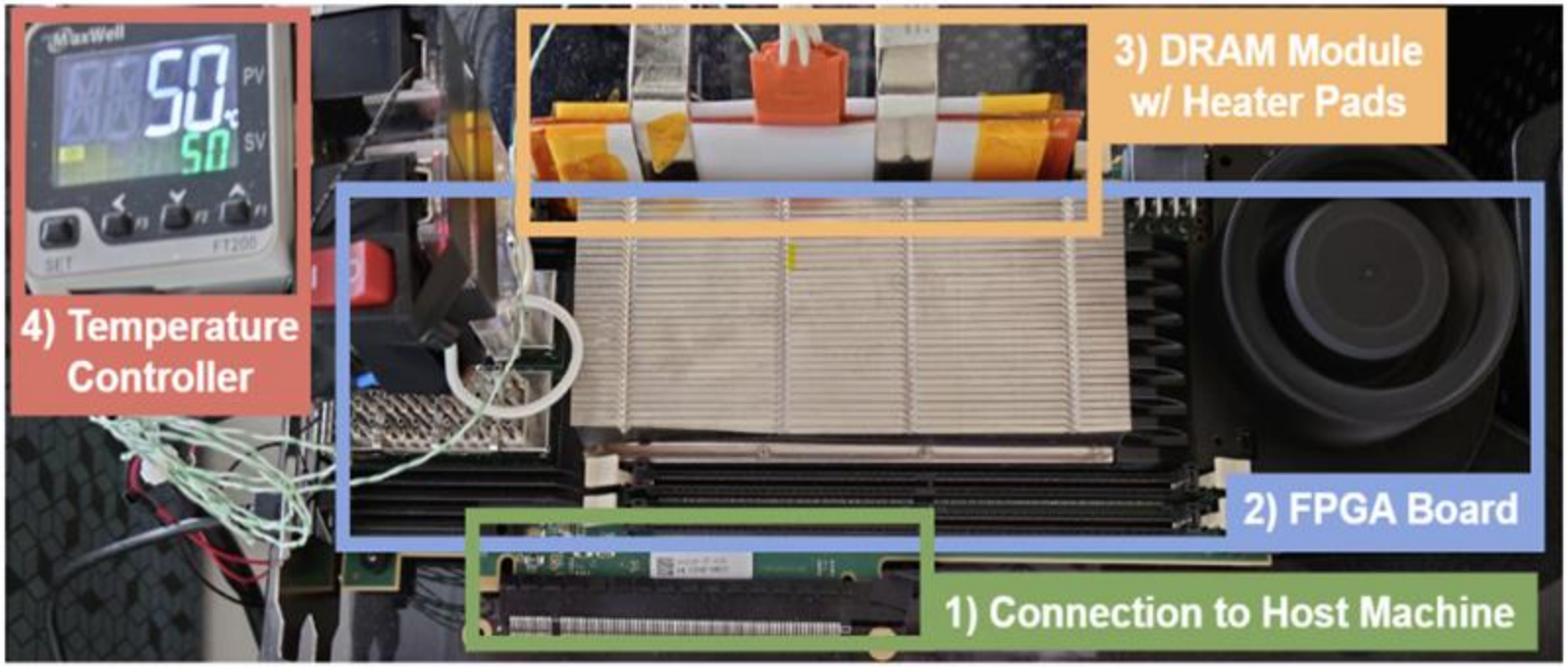
ETH Zürich

(More details in the paper)

https://arxiv.org/pdf/2402.18736.pdf

DRAM Testing Infrastructure

- Developed from DRAM Bender [Olgun+, TCAD'23]*
- Fine-grained control over DRAM commands, timings, and temperature



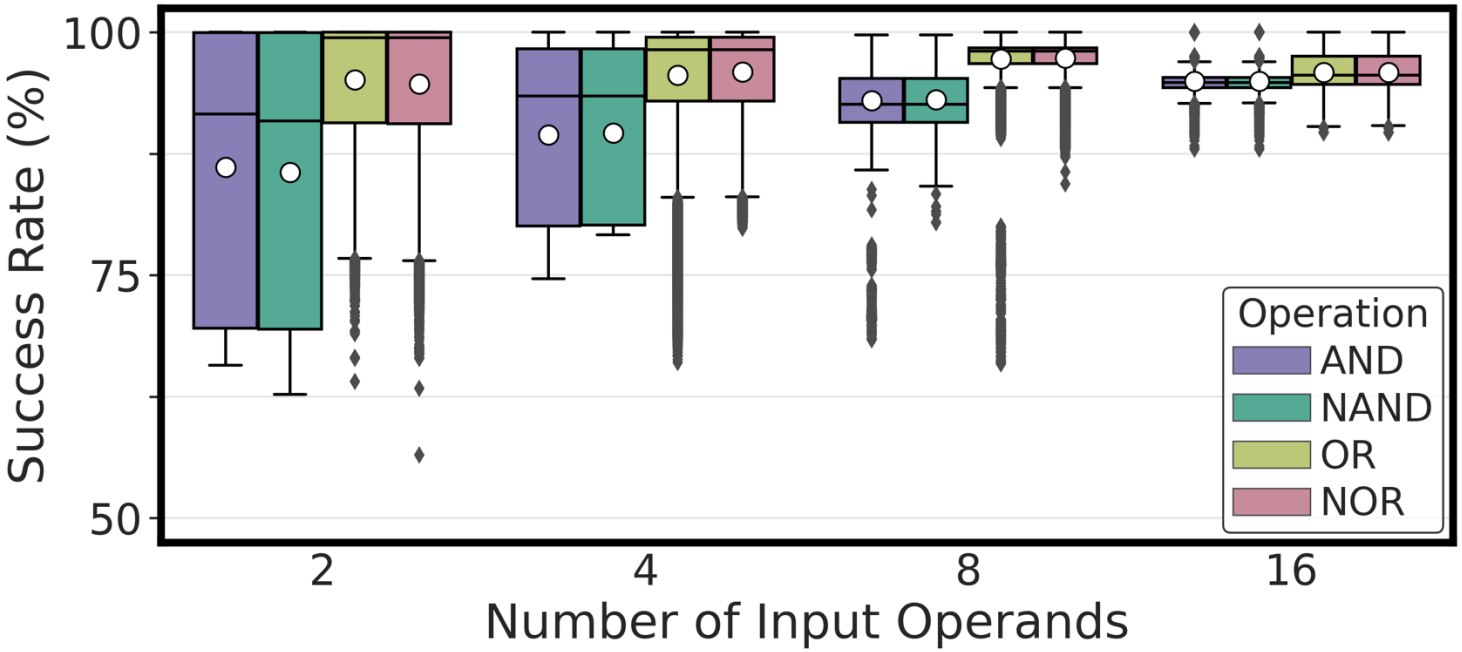
*Olgun et al., "DRAM Bender: An Extensible and Versatile FPGA-based Infrastructure to Easily Test State-of-the-art DRAM Chips," TCAD, 2023.

DRAM Chips Tested

- 256 DDR4 chips from two major DRAM manufacturers
- Covers different die revisions and chip densities

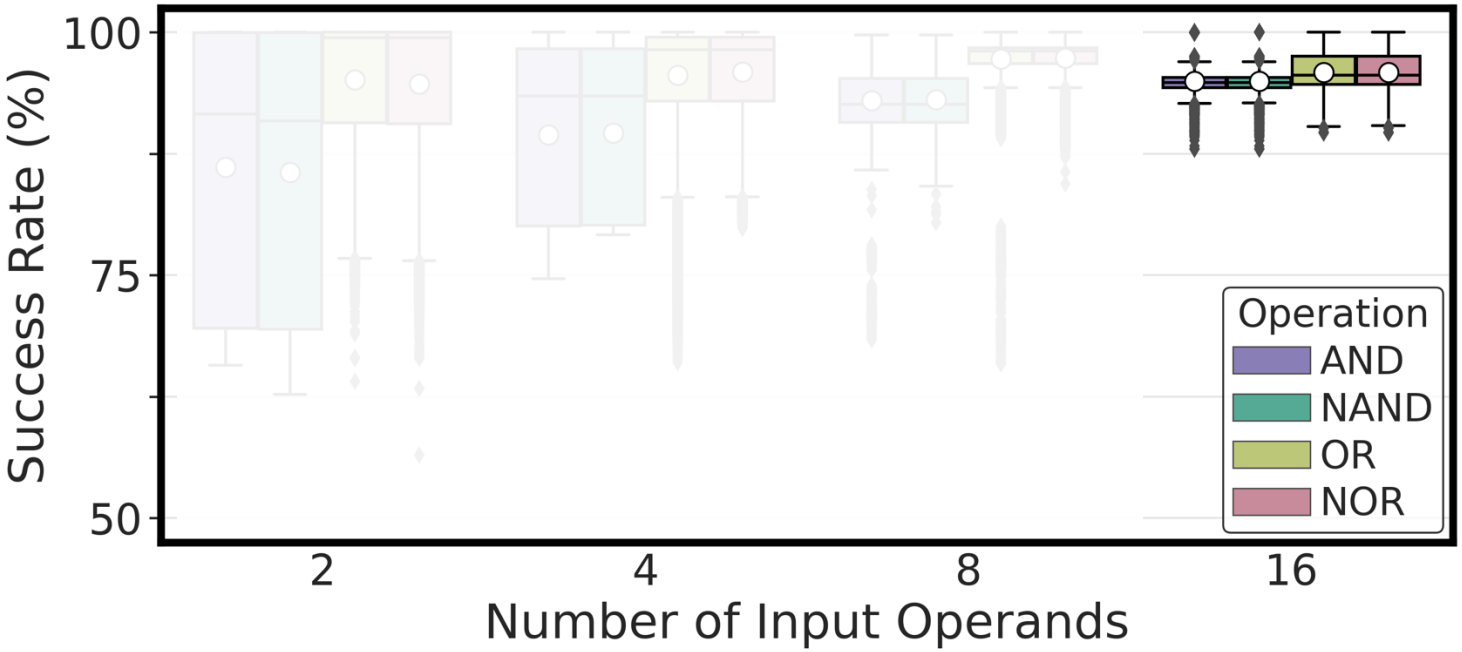
Chip Mfr.	#Modules (#Chips)	Die Rev.	Mfr. Date[a]	Chip Density	Chip Org.	Speed Rate
SK Hynix	9 (72)	M	N/A	4Gb	x8	2666MT/s
	5 (40)	A	N/A	4Gb	x8	2133MT/s
	1 (16)	A	N/A	8Gb	x8	2666MT/s
	1 (32)	A	18-14	4Gb	x4	2400MT/s
	1 (32)	A	16-49	8Gb	x4	2400MT/s
	1 (32)	M	16-22	8Gb	x4	2666MT/s
Samsung	1 (8)	F	21-02	4Gb	x8	2666MT/s
	2 (16)	D	21-10	8Gb	x8	2133MT/s
	1 (8)	A	22-12	8Gb	x8	3200MT/s

Performing AND, NAND, OR, and NOR



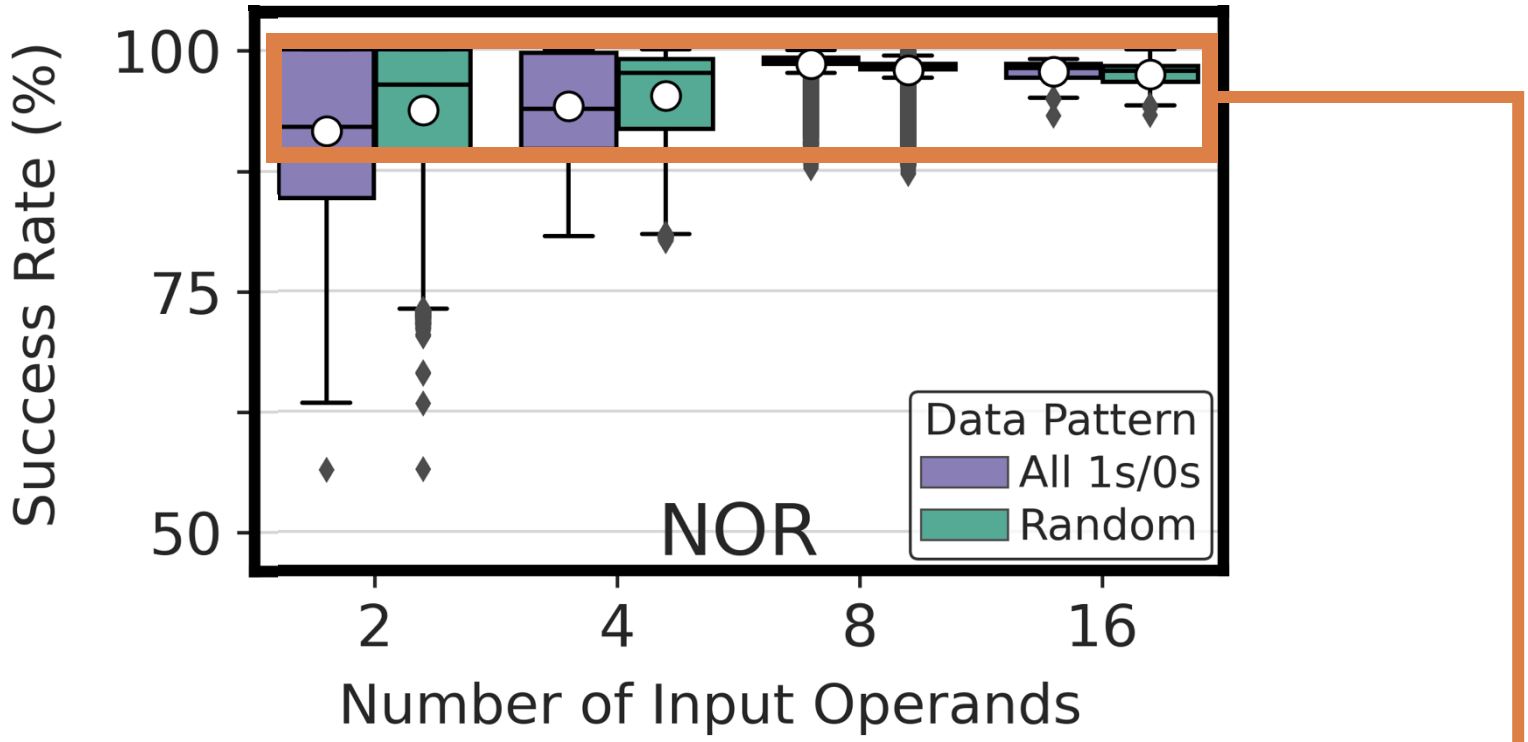
COTS DRAM chips can perform {2, 4, 8, 16}-input AND, NAND, OR, and NOR operations

Performing AND, NAND, OR, and NOR



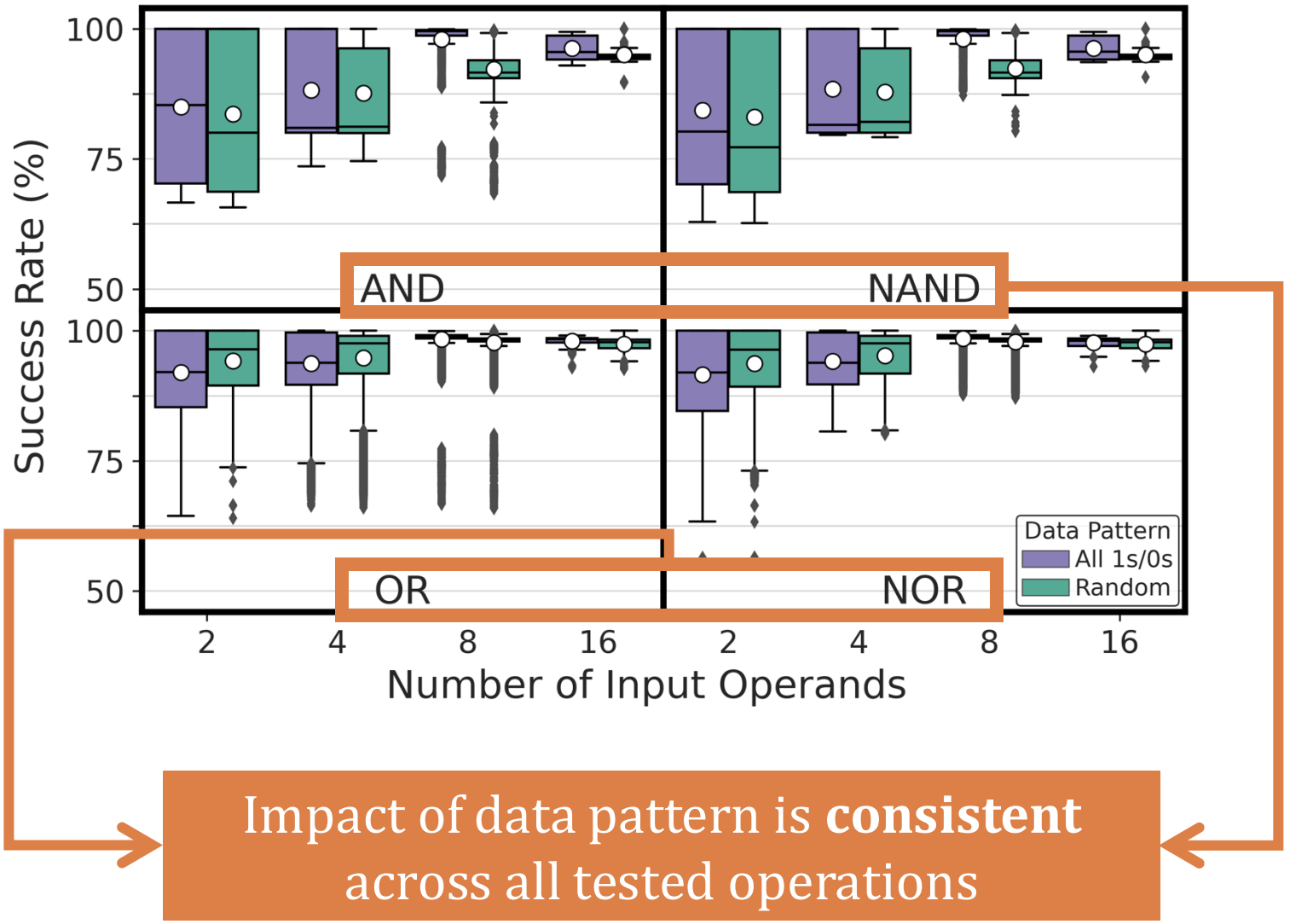
COTS DRAM chips can perform 16-input AND, NAND, OR, and NOR operations with very high success rate (>94%)

Impact of Data Pattern



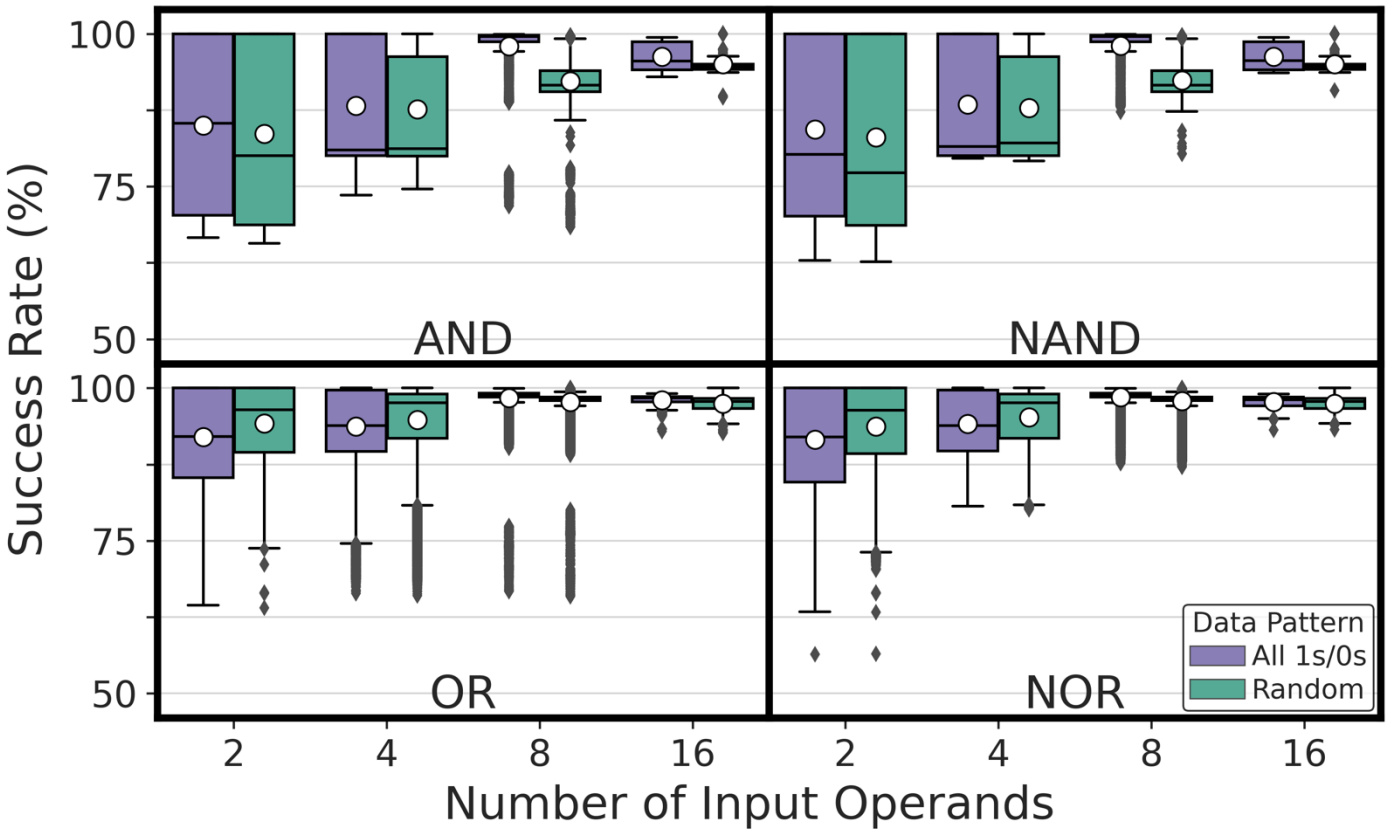
1.98% variation in **average success rate** across **all number of input operands**

Impact of Data Pattern



Impact of data pattern is **consistent** across all tested operations

Impact of Data Pattern



Data pattern slightly affects the reliability of AND, NAND, OR, and NOR operations

More in the Paper

- Detailed hypotheses & key ideas to perform
  - NOT operation
  - Many-input AND, NAND, OR, and NOR operations

- How the reliability of bitwise operations are affected by
  - The location of activated rows
  - Temperature (for AND, NAND, OR, and NOR)
  - DRAM speed rate
  - Chip density and die revision

- Discussion on the limitations of COTS DRAM chips

Available on arXiv

Functionally-Complete Boolean Logic in Real DRAM Chips: Experimental Characterization and Analysis

İsmail Emir Yüksel Yahya Can Tuğrul Ataberk Olgun F. Nisa Bostancı A. Giray Yağlıkçı
Geraldo F. Oliveira Haocong Luo Juan Gómez-Luna Mohammad Sadrosadati Onur Mutlu

ETH Zürich

Processing-using-DRAM (PuD) is an emerging paradigm that leverages the analog operational properties of DRAM circuitry to enable massively parallel in-DRAM computation. PuD has the potential to significantly reduce or eliminate costly data movement between processing elements and main memory. A common approach for PuD architectures is to make use of bulk bitwise computation (e.g., AND, OR, NOT). Prior works experimentally demonstrate three-input MAJ (i.e., MAJ3) and two-input AND and OR operations in commercial off-the-shelf (COTS) DRAM chips. Yet, demonstrations on COTS DRAM chips do not *provide a functionally complete set of operations (e.g., NAND or AND and NOT).*

We experimentally demonstrate that COTS DRAM chips are capable of performing 1) functionally-complete *Boolean operations: NOT, NAND, and NOR and 2) many-input (i.e., more than two-input) AND and OR operations. We present an extensive*

systems and applications [12, 13]. Processing-using-DRAM (PuD) [29–32] is a promising paradigm that can alleviate the data movement bottleneck. PuD uses the analog operational properties of the DRAM circuitry to enable massively parallel in-DRAM computation. Many prior works [29–53] demonstrate that PuD can greatly reduce or eliminate data movement.

A widely used approach for PuD is to perform bulk bitwise operations, i.e., bitwise operations on large bit vectors. To perform bulk bitwise operations using DRAM, prior works propose modifications to the DRAM circuitry [29–31, 33, 35, 36, 43, 44, 46, 48–58]. Recent works [38, 41, 42, 45] experimentally demonstrate the feasibility of executing data copy & initialization [42, 45], i.e., the RowClone operation [49], and a subset of bitwise operations, i.e., three-input bitwise majority (MAJ3) and two-input AND and OR operations in unmodified commercial off-the-shelf (COTS) DRAM chips by operating beyond

https://arxiv.org/pdf/2402.18736.pdf

Summary

- We experimentally demonstrate that **commercial off-the-shelf (COTS)** DRAM chips can perform:
  - **Functionally-complete** Boolean operations: NOT, NAND, and NOR
  - **Up to 16-input** AND, NAND, OR, and NOR operations
- We characterize **the success rate** of these operations on **256 COTS DDR4 chips** from **two major manufacturers**
- We highlight **two key results:**
  - We can perform **NOT** and **{2, 4, 8, 16}-input AND, NAND, OR, and NOR** operations on COTS DRAM chips with **very high success rates (>94%)**
  - **Data pattern** and **temperature** only slightly affect the reliability of these operations

We believe these empirical results demonstrate the promising potential of using DRAM as a computation substrate

Simultaneous Many-Row Activation in Off-the-Shelf DRAM Chips

Experimental Characterization and Analysis

İsmail Emir Yüksel

Yahya C. Tuğrul F. Nisa Bostancı Geraldo F. Oliveira

A. Giray Yağlıkçı Ataberk Olgun Melina Soysal Haocong Luo

Juan Gómez–Luna Mohammad Sadr Onur Mutlu

SAFARI ETH zürich

Executive Summary

Motivation:

- **Processing-Using-DRAM (PUD)** alleviates **data movement bottlenecks**
- Commercial off-the-shelf (COTS) DRAM chips can perform **three-input majority (MAJ3)** and **in-DRAM copy** operations

Goal: To experimentally analyze and understand

- The **computational capability** of COTS DRAM chips beyond that of prior works
- The **robustness** of such capability under various **operating conditions**

Experimental Study: 120 DDR4 chips from **two major manufacturers**

- COTS DRAM chips can perform **MAJ5, MAJ7, and MAJ9** operations and **copy** one DRAM row to **up to 31 different rows** at once
- Storing **multiple redundant copies** of MAJ's input operands (i.e., input replication) drastically increases **robustness** (>30% higher success rate)
- **Operating conditions** (temperature, voltage, and data pattern) **affect** the robustness of in-DRAM operations (by up to 11.52% success rate)

Leveraging Simultaneous Many-Row Activation

1 Perform **MAJX (where X>3)** operations

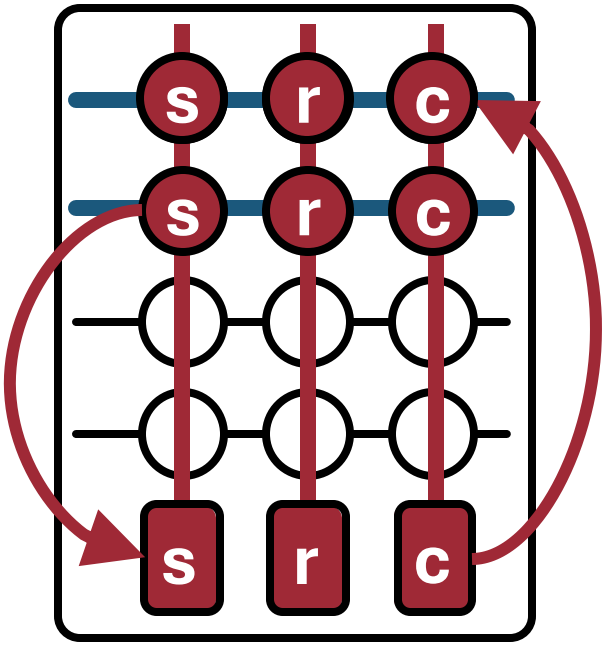
2 **Increase the robustness** of MAJX operations

3 Copy **one row's content** to **multiple rows**

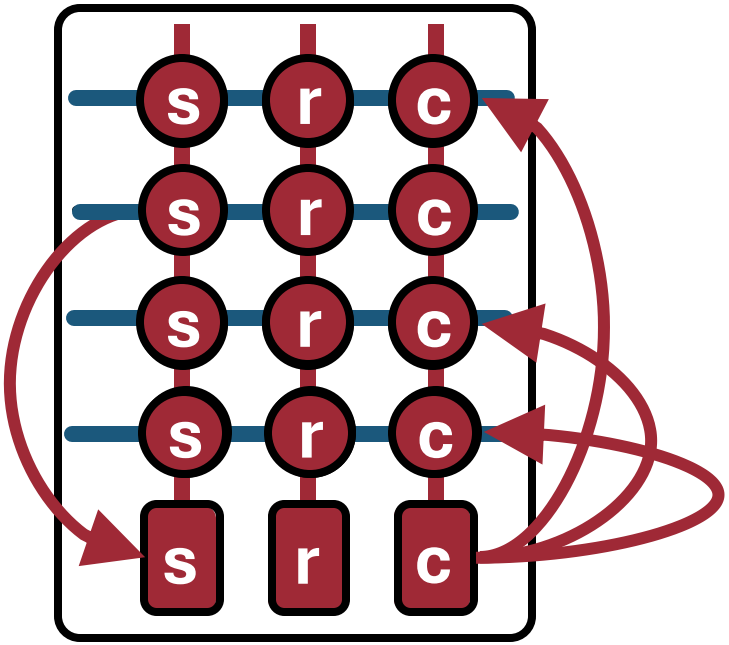
In-DRAM Multiple Row Copy (Multi-RowCopy)

Simultaneously activate many rows to
copy **one row's content** to **multiple destination rows**

RowClone



Multi-RowCopy



[Seshadri+ MICRO'13]

Key Takeaways from Multi-RowCopy

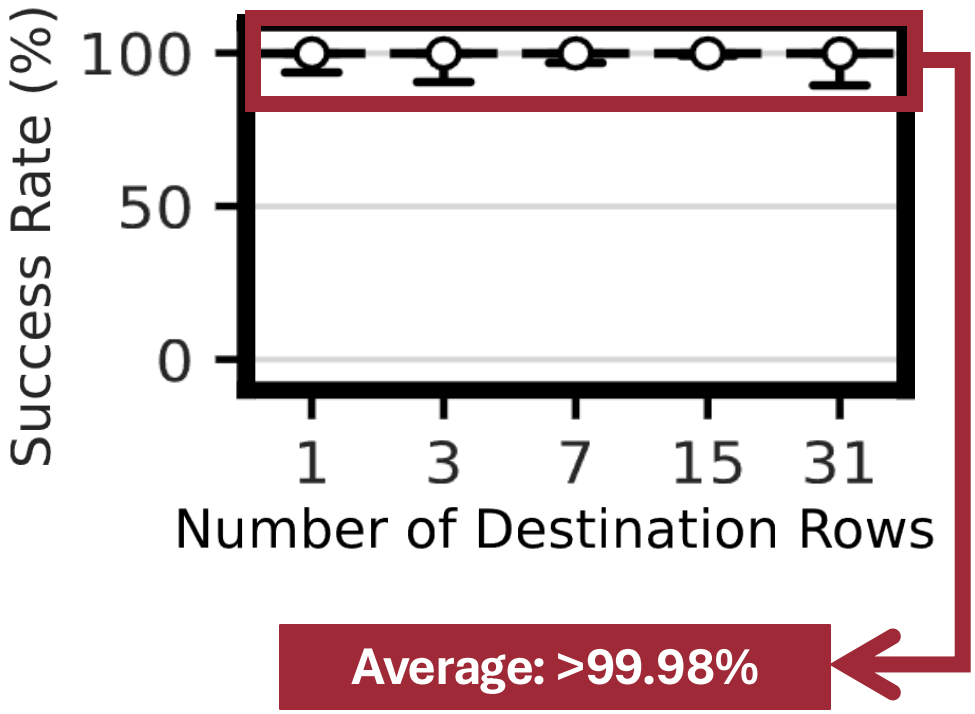
Key Takeaway 1

COTS DRAM chips are capable of copying one row's data to 1, 3, 7, 15, and 31 other rows at very high success rates

Key Takeaway 2

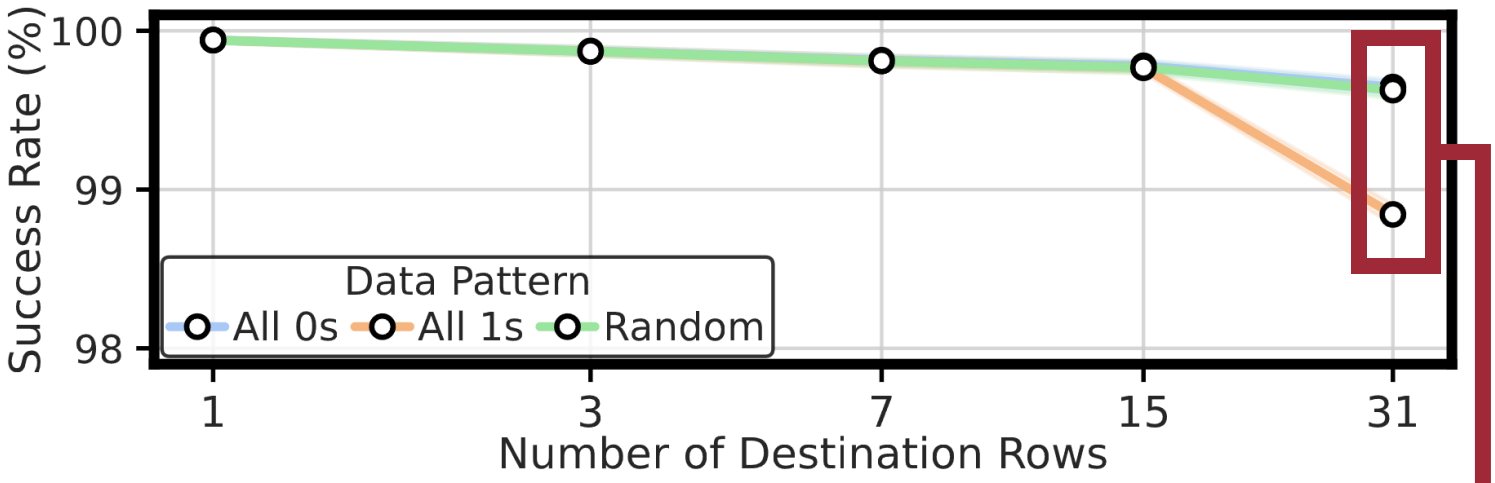
Multi-RowCopy in COTS DRAM chips is highly resilient to changes in data pattern, temperature, and wordline voltage

Robustness of Multi-RowCopy



COTS DRAM chips can copy one row's content to up to 31 rows with a very high success rate

Impact of Data Pattern



At most 0.79% decrease in average success rate

Data pattern has a small effect on the success rate of the Multi-RowCopy operation

Also in the Paper: Impact of Temperature & Voltage

Temperature

50 °C → 90 °C

Increasing temperature up to 90°C
has a very small effect on
the success rate of the Multi-RowCopy operation

Wordline Voltage

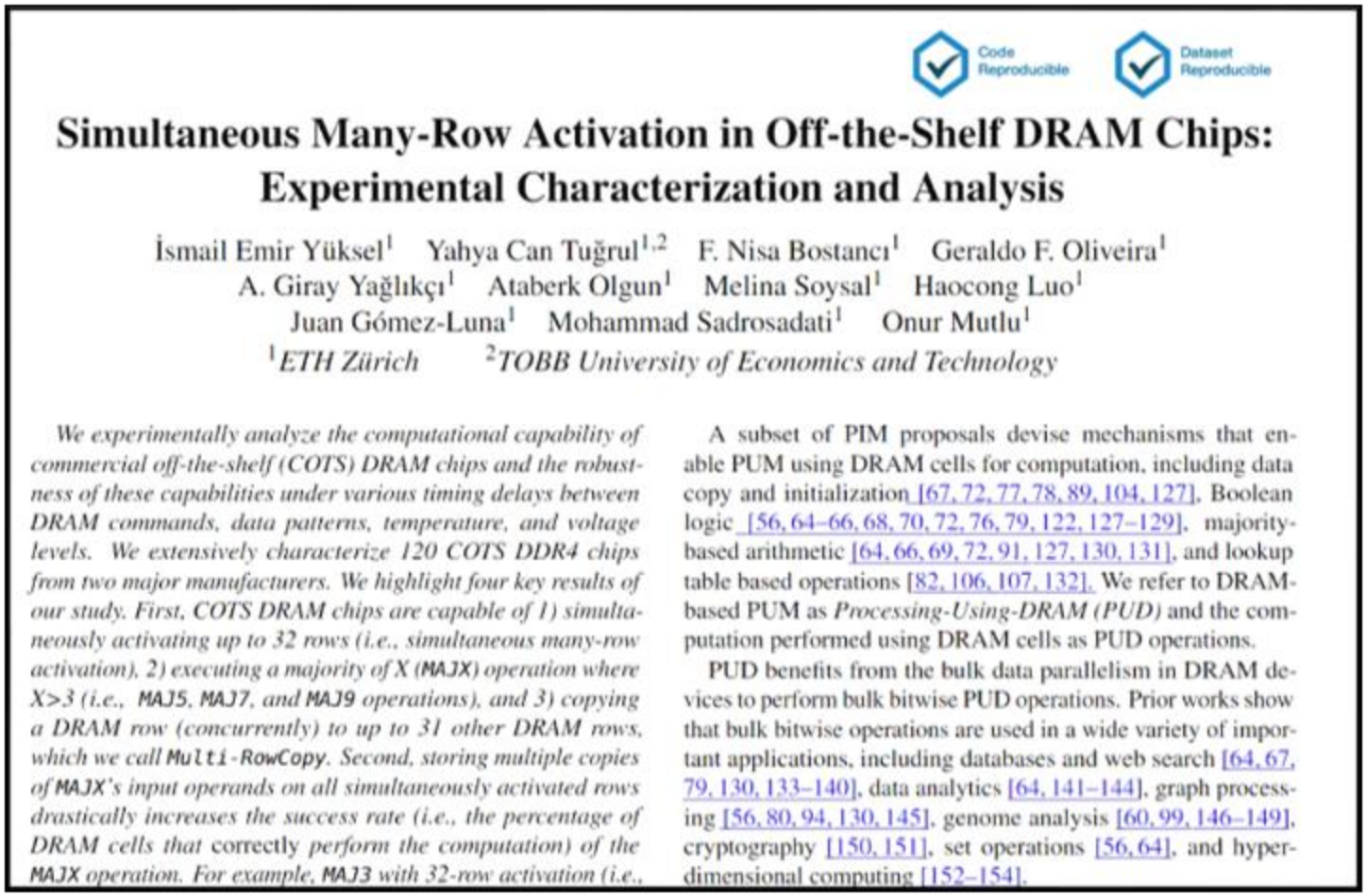
2.5V → 2.1V

Reducing the wordline voltage
only slightly affects
the success rate of the Multi-RowCopy operation

More in the Paper

- Detailed hypotheses and key ideas on
  - Hypothetical row decoder circuitry
  - Input Replication
- More characterization results
  - Power consumption of simultaneous many-row activation
  - Effect of timing delays between ACT-PRE and PRE-ACT commands
  - Effect of temperature and wordline voltage
- Circuit-level (SPICE) experiments for input replication
- Potential performance benefits of enabling new in-DRAM operations
  - Majority-based computation
  - Content destruction-based cold-boot attack prevention
- Discussions on the limitations of tested COTS DRAM chips

Available on arXiv



Code Reproducible

Dataset Reproducible

Simultaneous Many-Row Activation in Off-the-Shelf DRAM Chips: Experimental Characterization and Analysis

İsmail Emir Yüksel[1] Yahya Can Tuğrul[1,2] F. Nisa Bostancı[1] Geraldo F. Oliveira[1]
A. Giray Yağlıkçı[1] Ataberk Olgun[1] Melina Soysal[1] Haocong Luo[1]
Juan Gómez-Luna[1] Mohammad Sadrosadati[1] Onur Mutlu[1]
[1]*ETH Zürich* [2]*TOBB University of Economics and Technology*

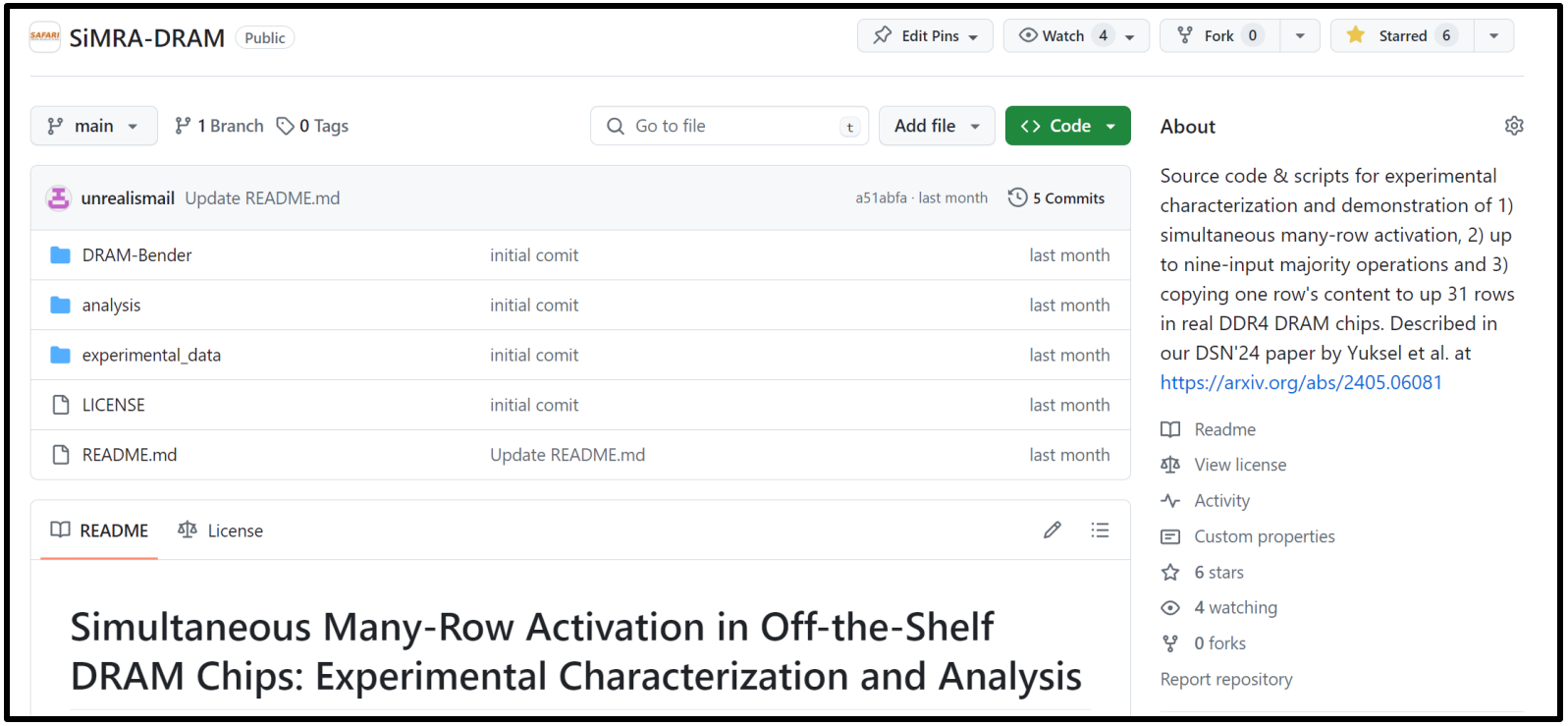
We experimentally analyze the computational capability of commercial off-the-shelf (COTS) DRAM chips and the robustness of these capabilities under various timing delays between DRAM commands, data patterns, temperature, and voltage levels. We extensively characterize 120 COTS DDR4 chips from two major manufacturers. We highlight four key results of our study. First, COTS DRAM chips are capable of 1) simultaneously activating up to 32 rows (i.e., simultaneous many-row activation), 2) executing a majority of X (`MAJX`) operation where X>3 (i.e., `MAJ5`, `MAJ7`, and `MAJ9` operations), and 3) copying a DRAM row (concurrently) to up to 31 other DRAM rows, which we call `Multi-RowCopy`. Second, storing multiple copies of `MAJX`'s input operands on all simultaneously activated rows drastically increases the success rate (i.e., the percentage of DRAM cells that correctly *perform the computation) of the `MAJX` operation. For example, `MAJ3` with 32-row activation (i.e.,*

A subset of PIM proposals devise mechanisms that enable PUM using DRAM cells for computation, including data copy and initialization [67, 72, 77, 78, 89, 104, 127], Boolean logic [56, 64–66, 68, 70, 72, 76, 79, 122, 127–129], majority-based arithmetic [64, 66, 69, 72, 91, 127, 130, 131], and lookup table based operations [82, 106, 107, 132]. We refer to DRAM-based PUM as *Processing-Using-DRAM (PUD)* and the computation performed using DRAM cells as PUD operations.

PUD benefits from the bulk data parallelism in DRAM devices to perform bulk bitwise PUD operations. Prior works show that bulk bitwise operations are used in a wide variety of important applications, including databases and web search [64, 67, 79, 130, 133–140], data analytics [64, 141–144], graph processing [56, 80, 94, 130, 145], genome analysis [60, 99, 146–149], cryptography [150, 151], set operations [56, 64], and hyperdimensional computing [152–154].

https://arxiv.org/pdf/2405.06081

Our Work is Open Source and Artifact Evaluated



https://github.com/CMU-SAFARI/SiMRA-DRAM

MegIS: Metagenomics In Storage

Background: GenStore

- Nika Mansouri Ghiasi, Jisung Park, Harun Mustafa, Jeremie Kim, Ataberk Olgun, Arvid Gollwitzer, Damla Senol Cali, Can Firtina, Haiyu Mao, Nour Almadhoun Alserr, Rachata Ausavarungnirun, Nandita Vijaykumar, Mohammed Alser, and Onur Mutlu,
  "GenStore: A High-Performance and Energy-Efficient In-Storage Computing System for Genome Sequence Analysis"
  *Proceedings of the 27th International Conference on Architectural Support for Programming Languages and Operating Systems (**ASPLOS**)*, Virtual, February-March 2022.
  [Lightning Talk Slides (pptx) (pdf)]
  [Lightning Talk Video (90 seconds)]

GenStore: A High-Performance In-Storage Processing System for Genome Sequence Analysis

Nika Mansouri Ghiasi[1] Jisung Park[1] Harun Mustafa[1] Jeremie Kim[1] Ataberk Olgun[1]
Arvid Gollwitzer[1] Damla Senol Cali[2] Can Firtina[1] Haiyu Mao[1] Nour Almadhoun Alserr[1]
Rachata Ausavarungnirun[3] Nandita Vijaykumar[4] Mohammed Alser[1] Onur Mutlu[1]

[1]ETH Zürich [2]Bionano Genomics [3]KMUTNB [4]University of Toronto

MegIS

High-Performance, Energy-Efficient, and Low-Cost Metagenomic Analysis with In-Storage Processing

Nika Mansouri Ghiasi

Mohammad Sadrosadati Harun Mustafa Arvid Gollwitzer Can Firtina

Julien Eudine Haiyu Mao Joël Lindegger Meryem Banu Cavlak

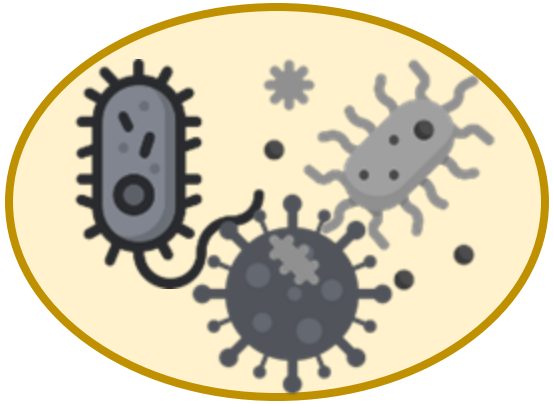
Mohammed Alser Jisung Park Onur Mutlu

SAFARI

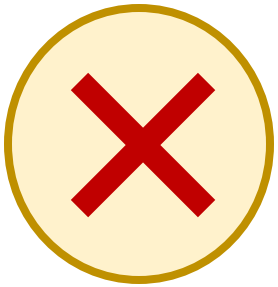
ETH zürich POSTECH

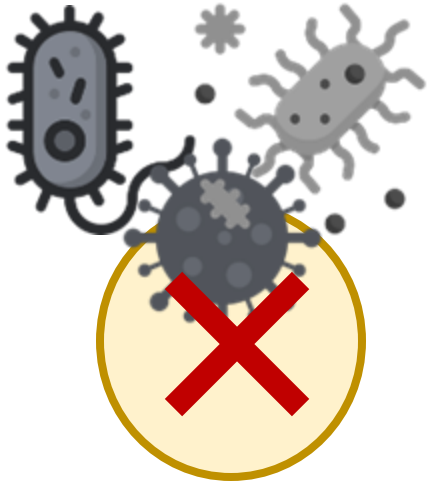
What is Metagenomics?

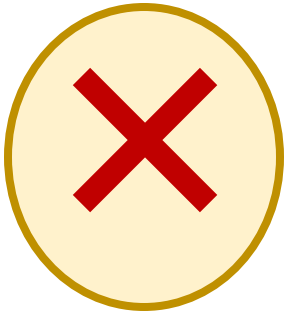
- ***Metagenomics:*** Study of genome sequences of **diverse organisms** within a **shared environment** (e.g., blood, ocean, soil)



- **Overcomes the limitations of traditional genomics**
  - Bypasses the need for culturing individual species in isolation

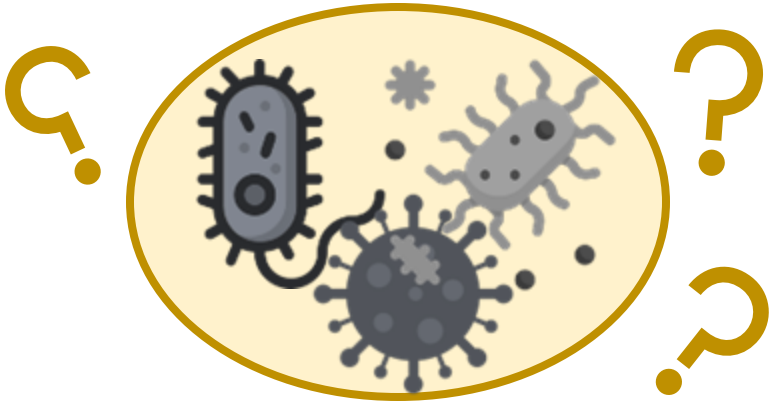






What is Metagenomics?

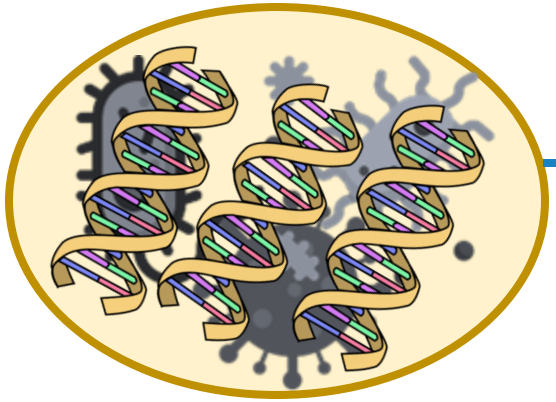
- ***Metagenomics:*** Study of genome sequences of **diverse organisms** within a **shared environment** (e.g., blood, ocean, soil)



Has led to groundbreaking advances

- Precision medicine
- Understanding microbial diversity of an environment
- Discovering early warnings of communicable diseases

Metagenomic Analysis



Metagenomic sample with species that are **not known** in advance

A large database containing information on **many species**

Preparation of Input Queries

Query K-mers: GCTCA, CTCAT, TCATG, ...

Presence/Absence Identification

V. cholerae

SARS-CoV-2

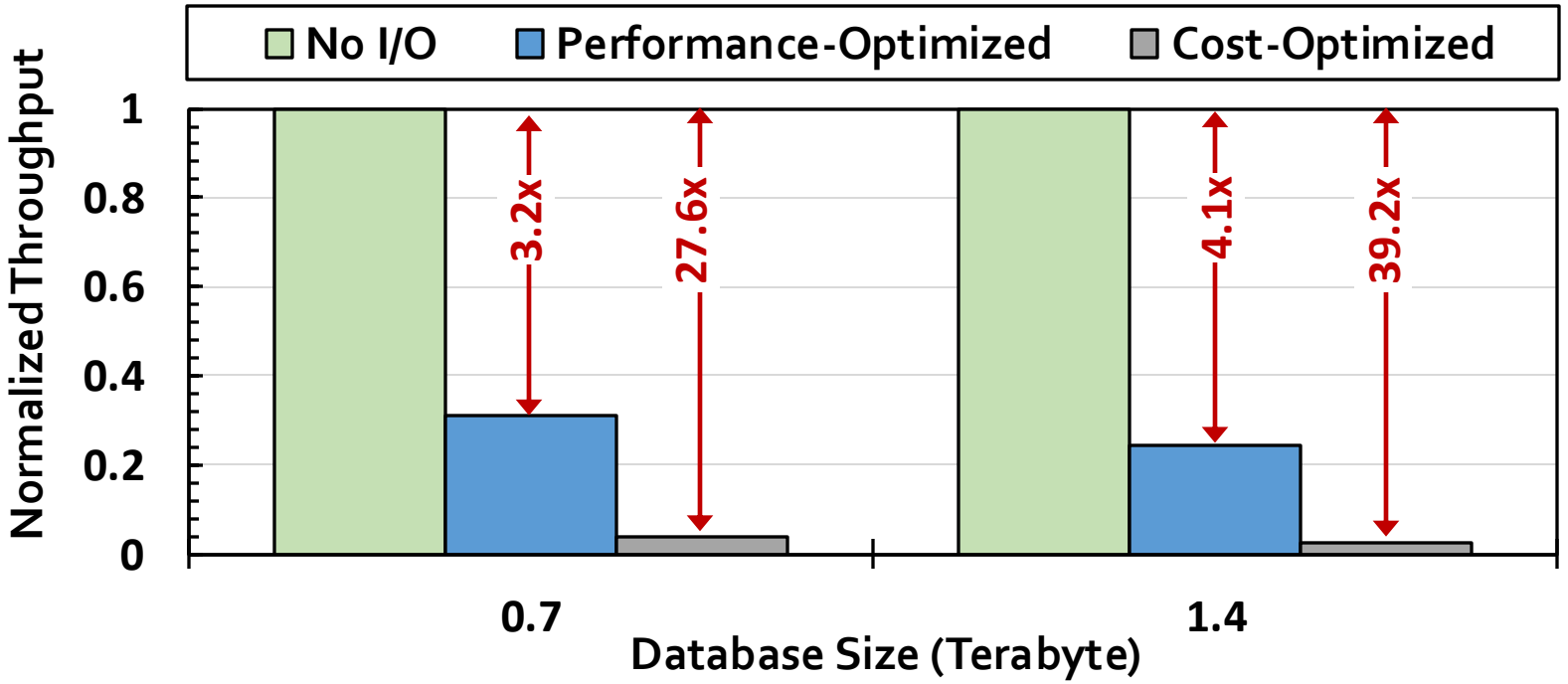
E. coli

Abundance Estimation

(e.g., > 100 TBs in emerging databases)

Motivation

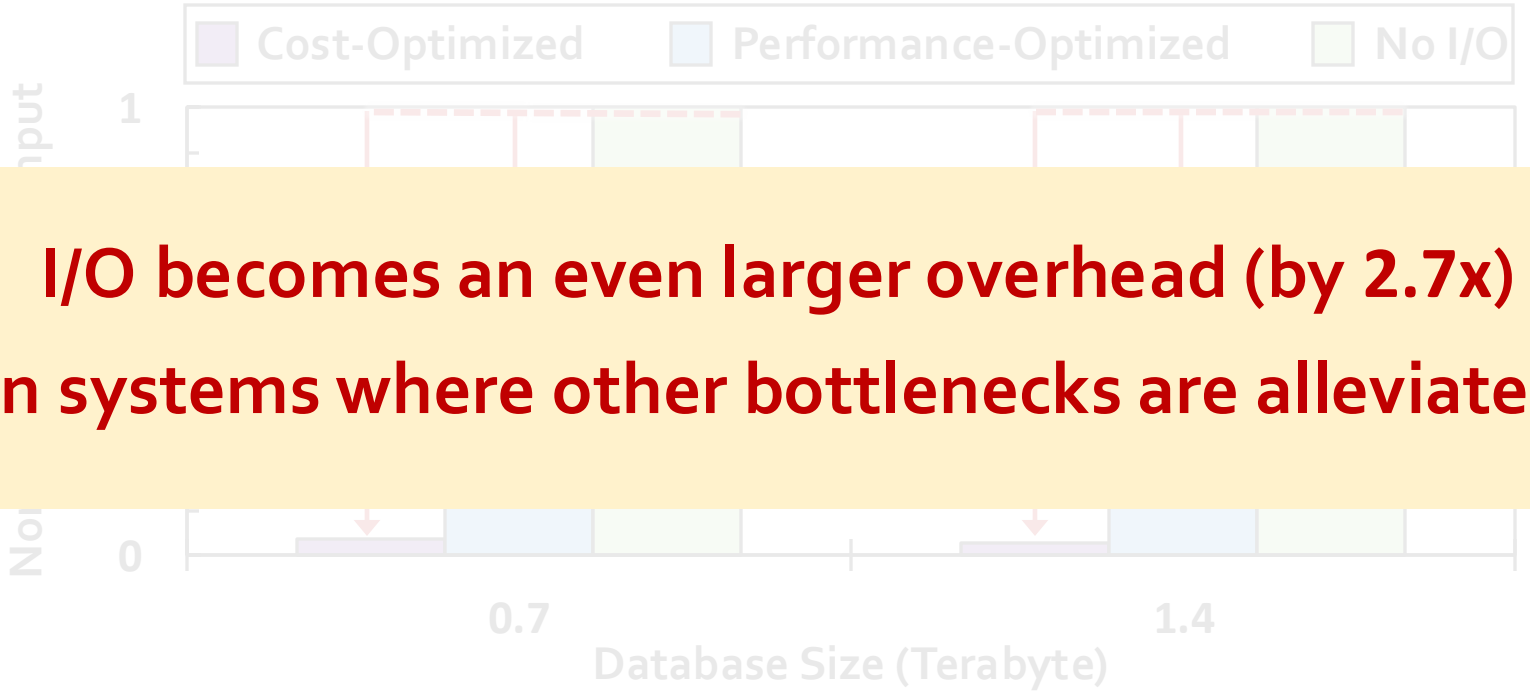
- Case study of the performance of metagenomic analysis tools
- With various state-of-the-art SSD configurations



I/O data movement causes significant performance overhead

Motivation

- Case study on the throughput of metagenomic analysis tools
- With Various state-of-the-art SSD configurations



I/O becomes an even larger overhead (by 2.7x) in systems where other bottlenecks are alleviated

I/O data movement causes significant performance overhead

I/O Overhead is Hard to Avoid

I/O overhead due to accessing **large**, **low-reuse** data is hard to avoid

Sampling techniques to shrink database sizes

✗ *Reduce accuracy to levels unacceptable for many use cases*

Keeping all data required by metagenomic analysis completely and always resident in main memory

✗ *Energy inefficient, costly, unscalable, and unsustainable*

- Database sizes **increase rapidly** (doubling every few months)
- Different analyses need **different databases**

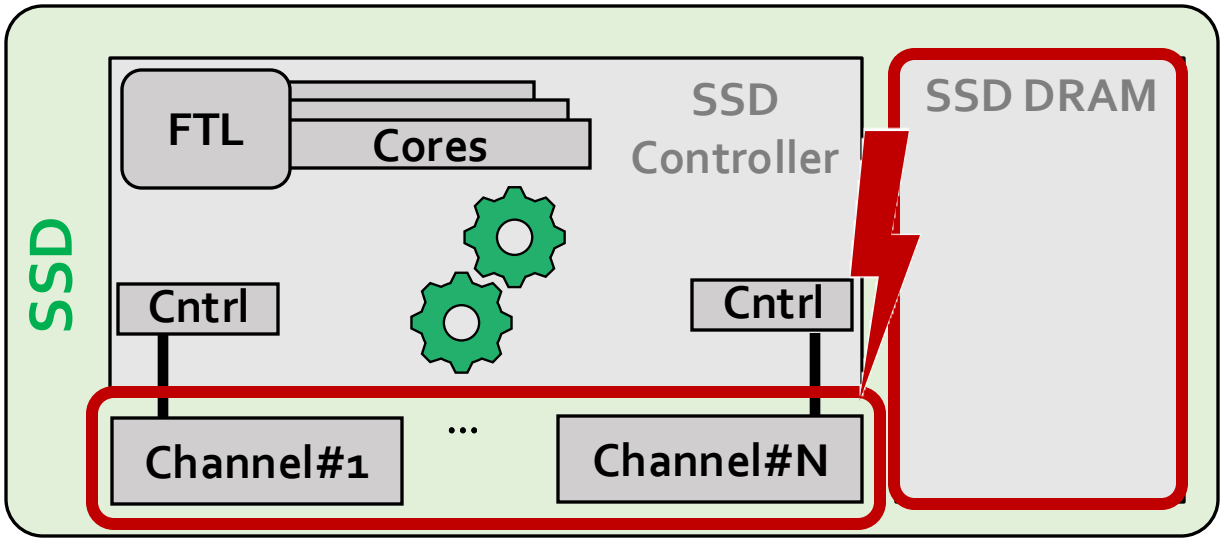
Our Goal

Improve metagenomic analysis ***performance***
by reducing large ***data movement overhead***
from the storage system
in a ***cost-effective*** *manner*

Challenges of In-Storage Processing

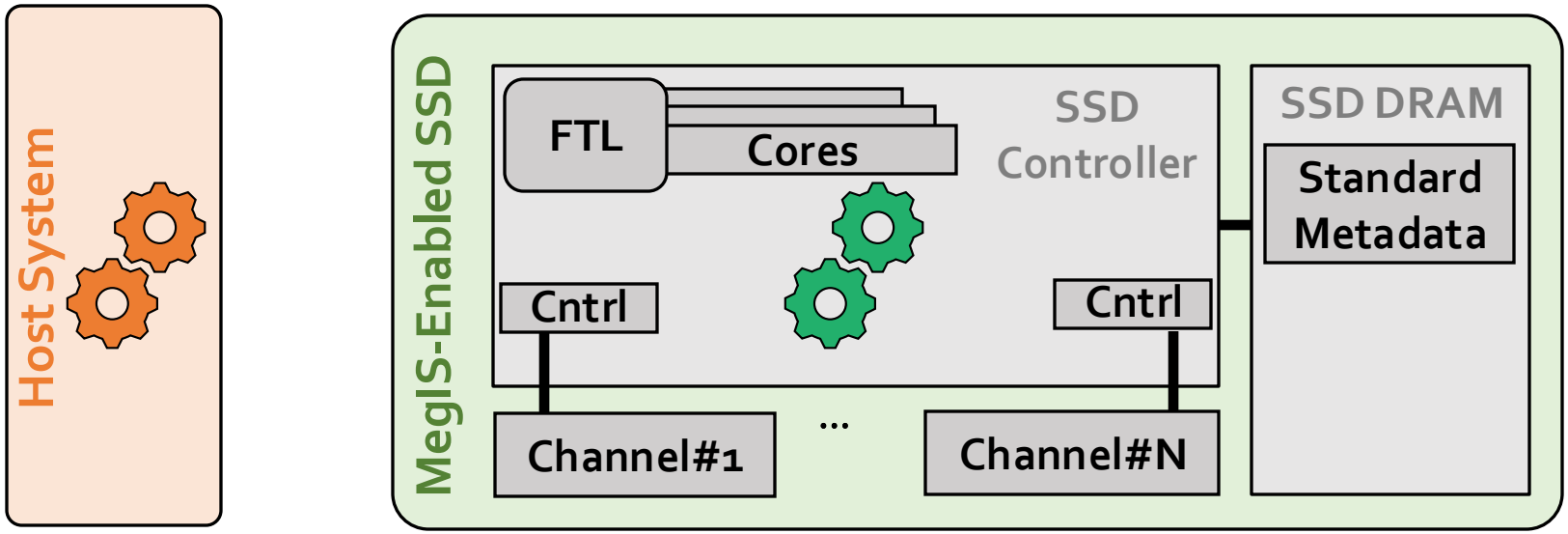
Existing metagenomic analysis approaches cannot be implemented as an in-storage processing system due to **SSD hardware limitations**

- Long **latency of NAND flash** chips
- Limited **DRAM capacity** inside the SSD
- Limited **DRAM bandwidth** inside the SSD

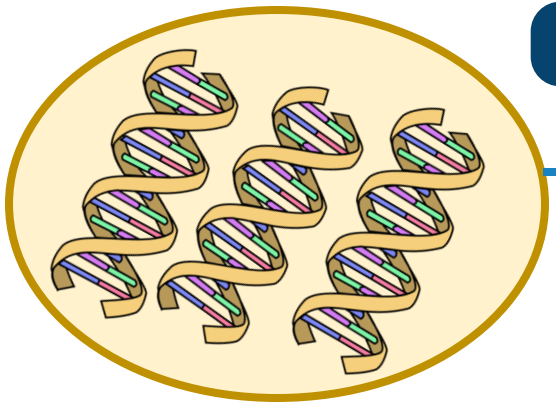


MegIS: Metagenomics In-Storage

- First in-storage system for *end-to-end* metagenomic analysis
- **Idea:** Cooperative in-storage processing for metagenomic analysis
  - Hardware/software co-design between



MegIS's Steps



Metagenomic sample with species that are **not known** in advance

A large database containing information on **many species**

Step 1

Preparation of Input Queries

Query K-mers: GCTCA, CTCAT, TCATG, ...

Step 2

Presence/Absence Identification

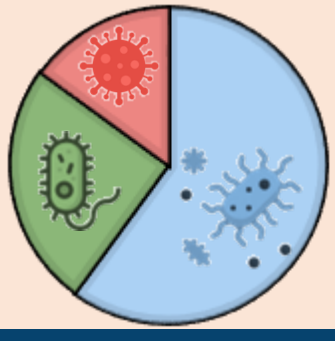
V. cholerae

SARS-CoV-2

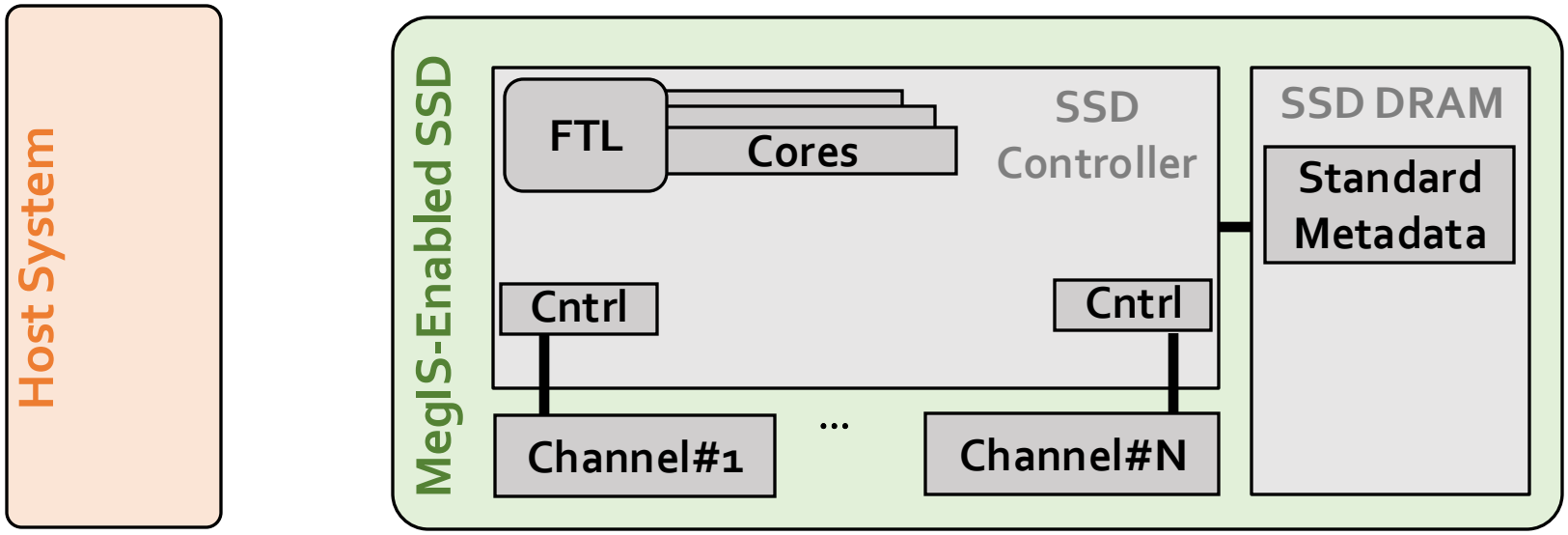
E. coli

Step 3

Abundance Estimation



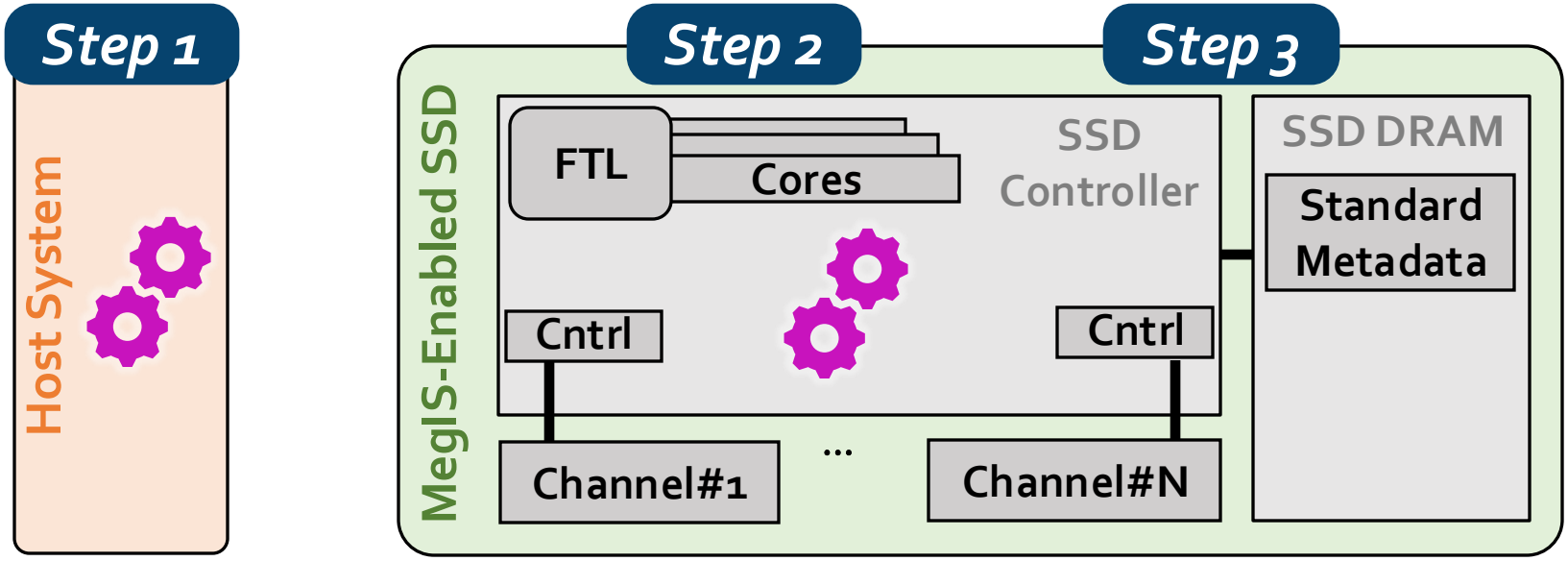
MegIS Hardware-Software Co-Design



MegIS Hardware-Software Co-Design

Task partitioning and mapping

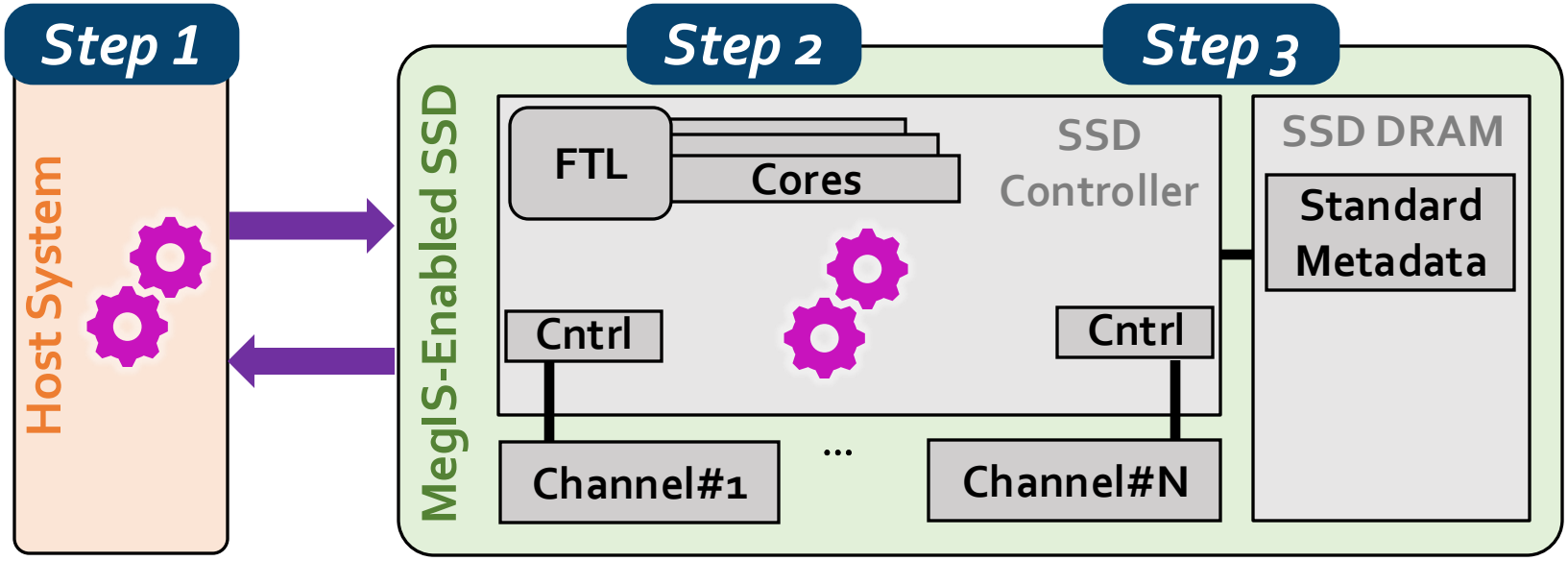
- *Each step executes in its most suitable system*



MegIS Hardware-Software Co-Design

Task partitioning and mapping
- *Each step executes in its most suitable system*

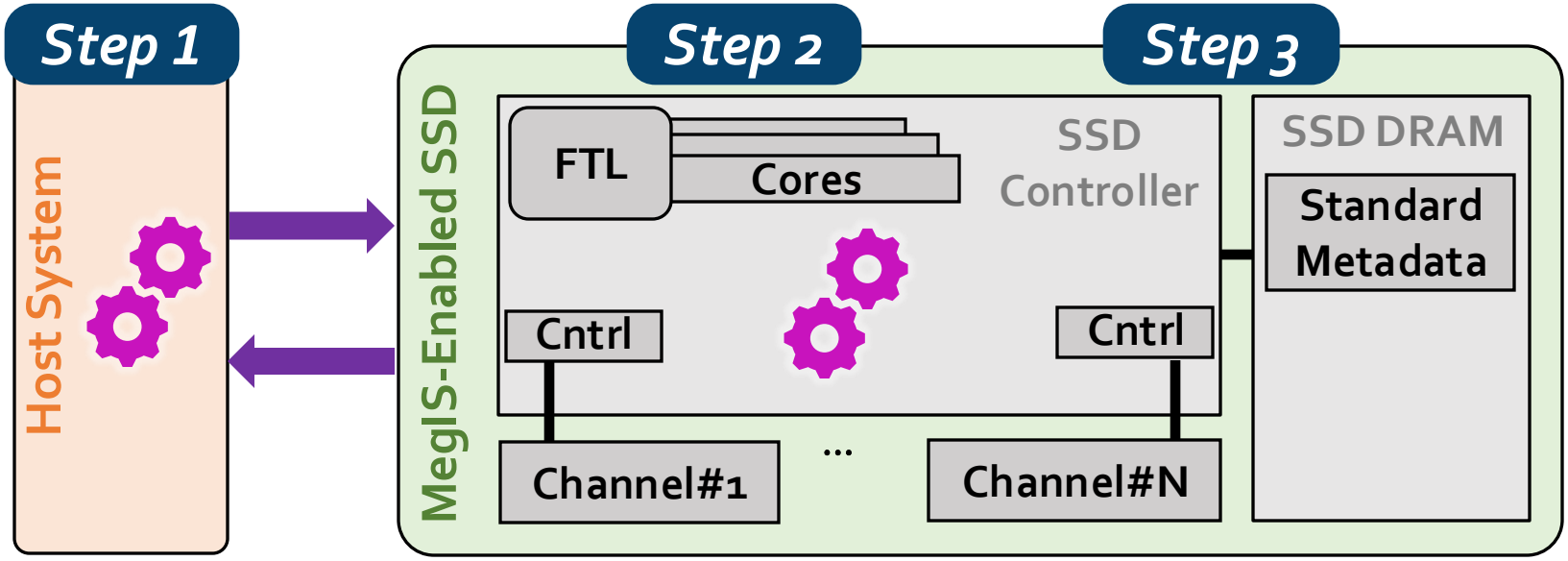
Data/computation flow coordination
- *Reduce communication overhead*
- *Reduce #writes to flash chips*



MegIS Hardware-Software Co-Design

Task partitioning and mapping
- *Each step executes in its most suitable system*

Data/computation flow coordination
- *Reduce communication overhead*
- *Reduce #writes to flash chips*

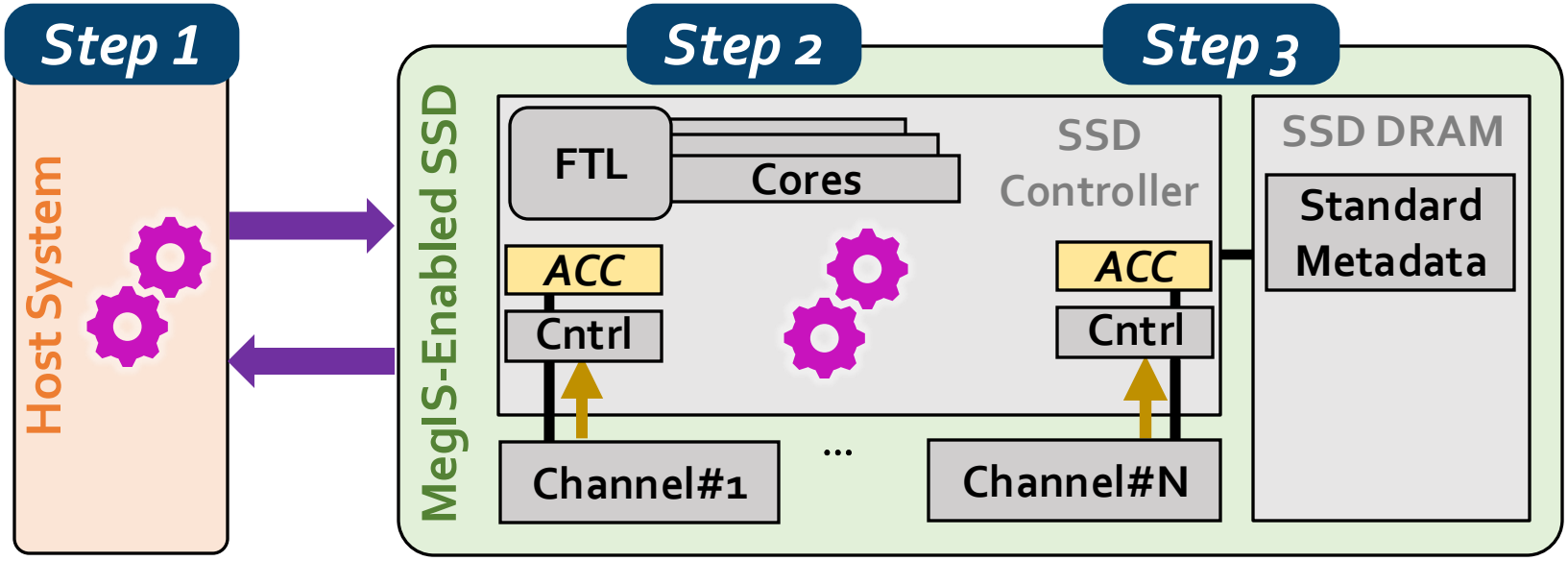


Storage-aware algorithms
- *Enable efficient access patterns to the SSD*

MegIS Hardware-Software Co-Design

Task partitioning and mapping
- *Each step executes in its most suitable system*

Data/computation flow coordination
- *Reduce communication overhead*
- *Reduce #writes to flash chips*



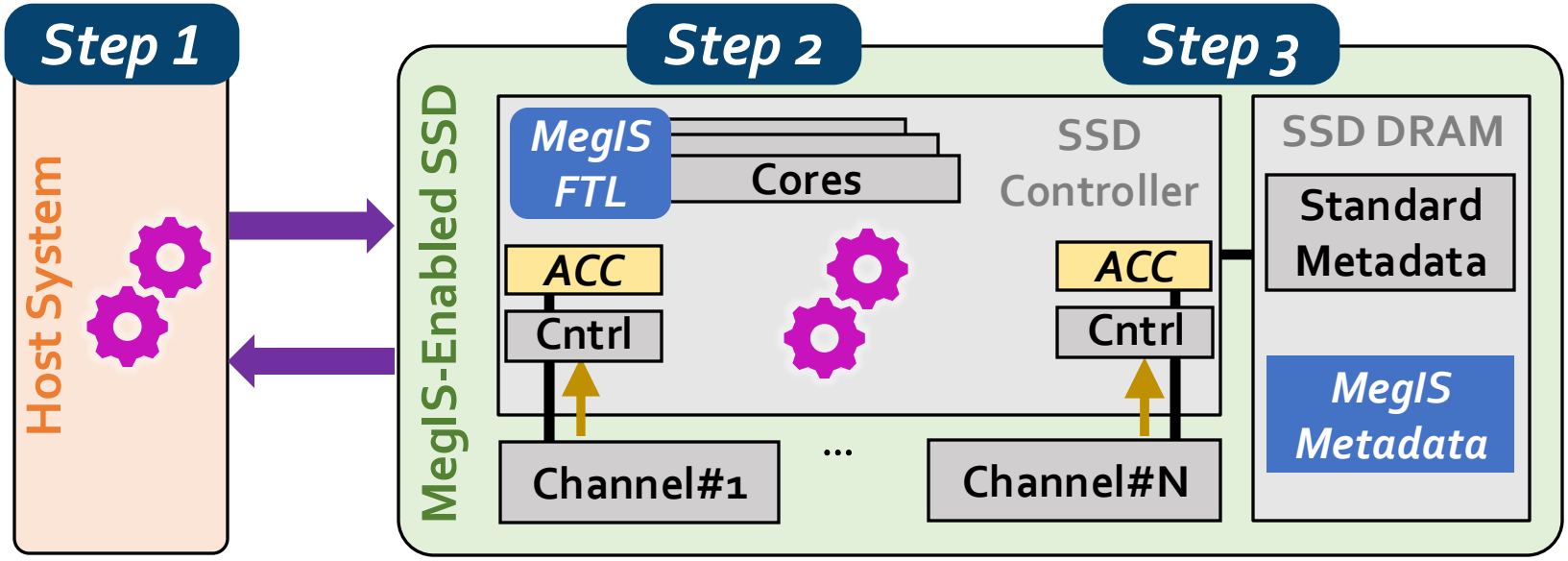
Storage-aware algorithms
- *Enable efficient access patterns to the SSD*

Lightweight in-storage accelerators
- *Minimize SRAM/DRAM buffer spaces needed inside the SSD*

MegIS Hardware-Software Co-Design

Task partitioning and mapping
- *Each step executes in its most suitable system*

Data/computation flow coordination
- *Reduce communication overhead*
- *Reduce #writes to flash chips*



Storage-aware algorithms
- *Enable efficient access patterns to the SSD*

Lightweight in-storage accelerators
- *Minimize SRAM/DRAM buffer spaces needed inside the SSD*

Data mapping scheme and Flash Translation Layer (FTL)
- *Specialize to the characteristics of metagenomic analysis*
- *Leverage the SSD's full internal bandwidth*

Evaluation: Methodology Overview

Performance, Energy, and Power Analysis

Hardware Components

- Synthesized Verilog model for the in-storage accelerators
- MQSim *[Tavakkol+, FAST'18]* for SSD's internal operations
- Ramulator *[Kim+, CAL'15]* for SSD's internal DRAM

Software Components

Measure on a real system:

- AMD® EPYC® CPU with 128 physical cores
- 1-TB DRAM

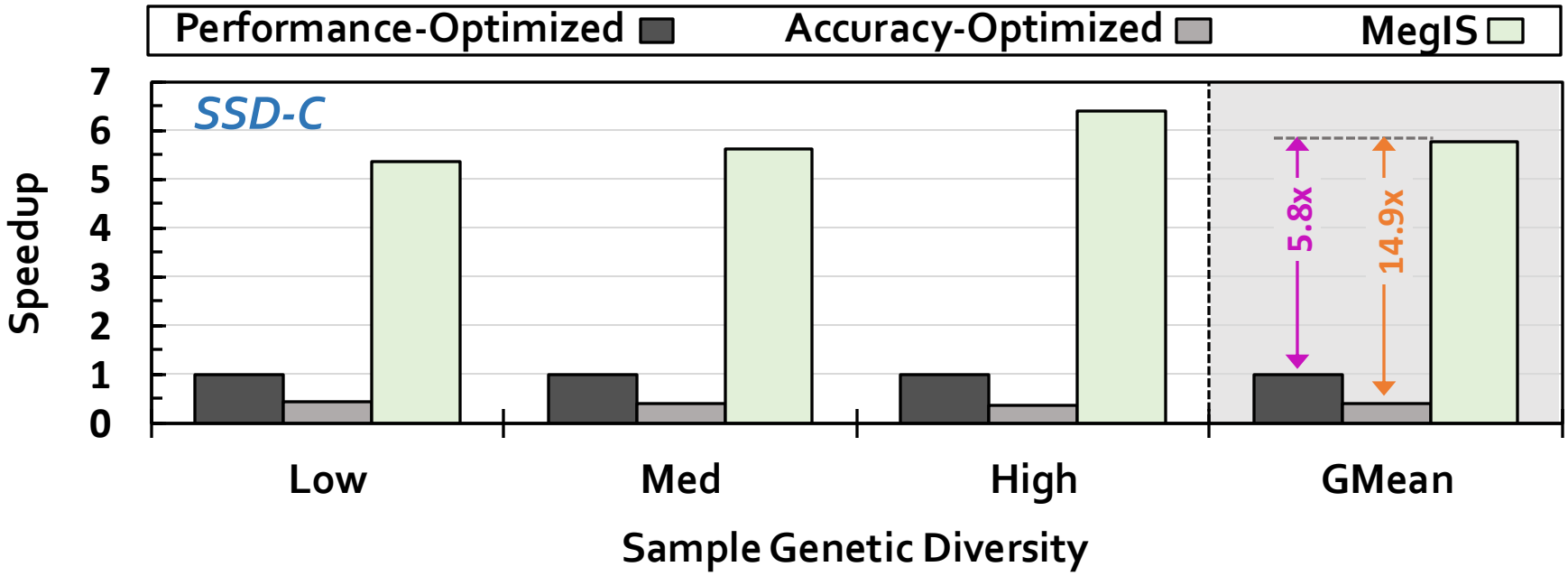
Baseline Comparison Points

- **Performance-optimized software**, Kraken2 *[Genome Biology'19]*
- **Accuracy-optimized software**, Metalign *[Genome Biology'20]*
- **PIM hardware-accelerated tool** (using processing-in-memory), Sieve *[ISCA'21]*

SSD Configurations

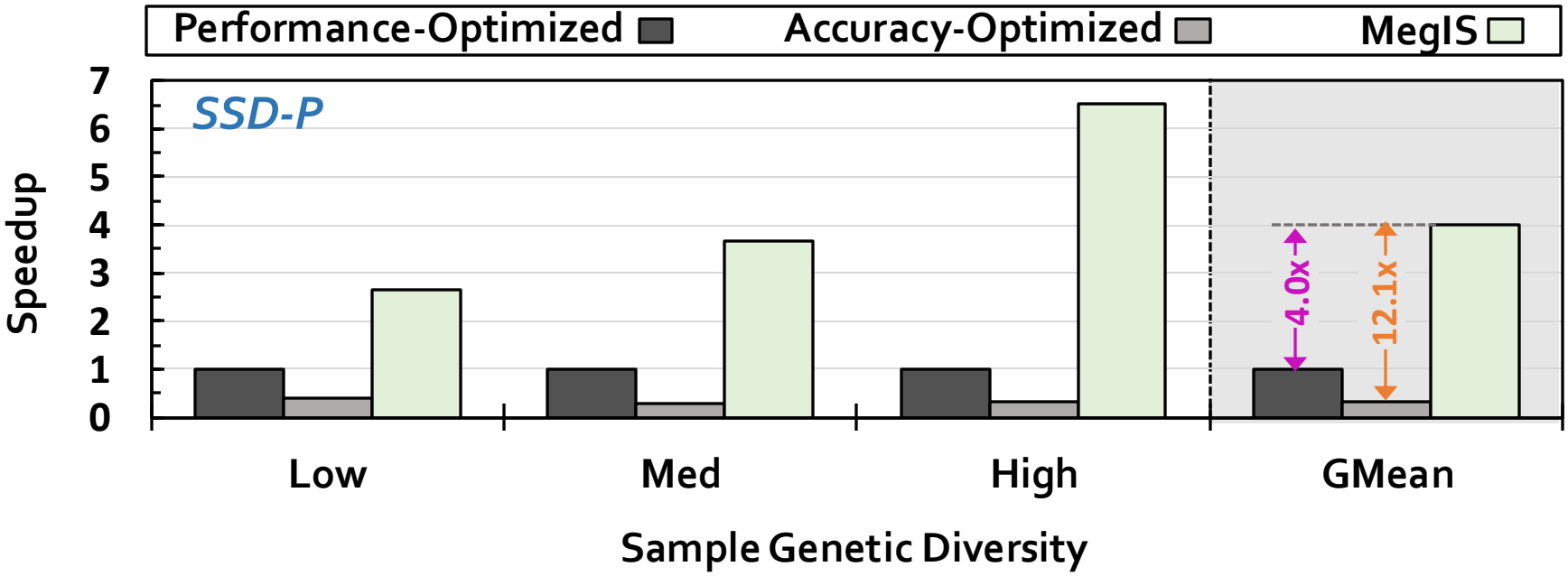
- **SSD-C:** with SATA3 interface (0.5 GB/s sequential read bandwidth)
- **SSD-P:** with PCIe Gen4 interface (7 GB/s sequential read bandwidth)

Evaluation: Speedup over the Software Baselines



MegIS provides significant speedup over both Performance-Optimized and Accuracy-Optimized baselines

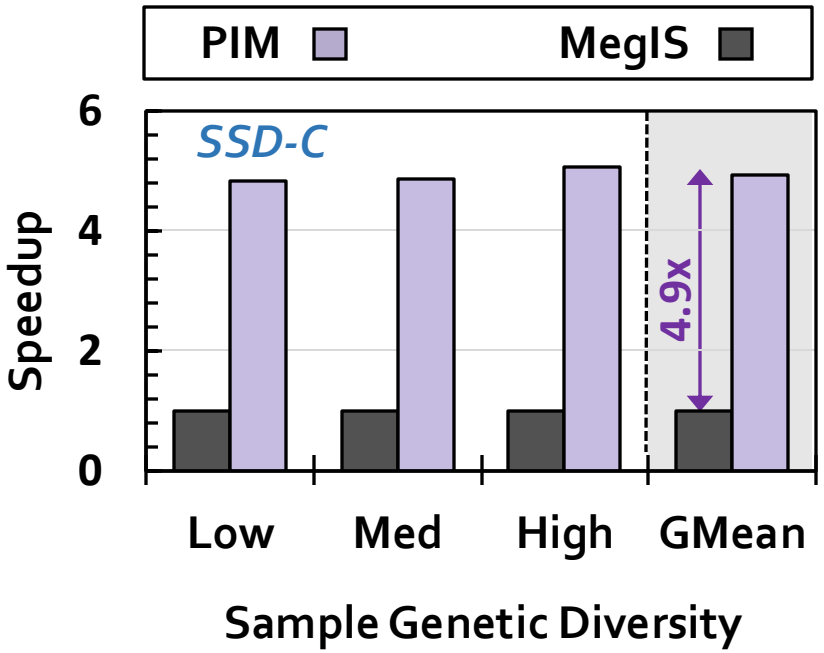
Evaluation: Speedup over the Software Baselines

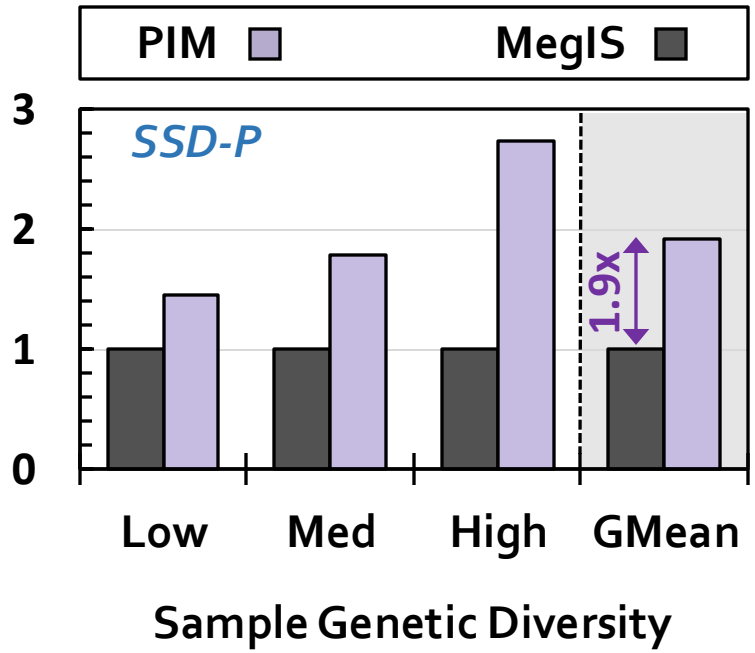


MegIS provides significant speedup over both Performance-Optimized and Accuracy-Optimized baselines

MegIS improves performance on both cost-optimized and performance-optimized SSDs

Evaluation: Speedup over the PIM Baseline

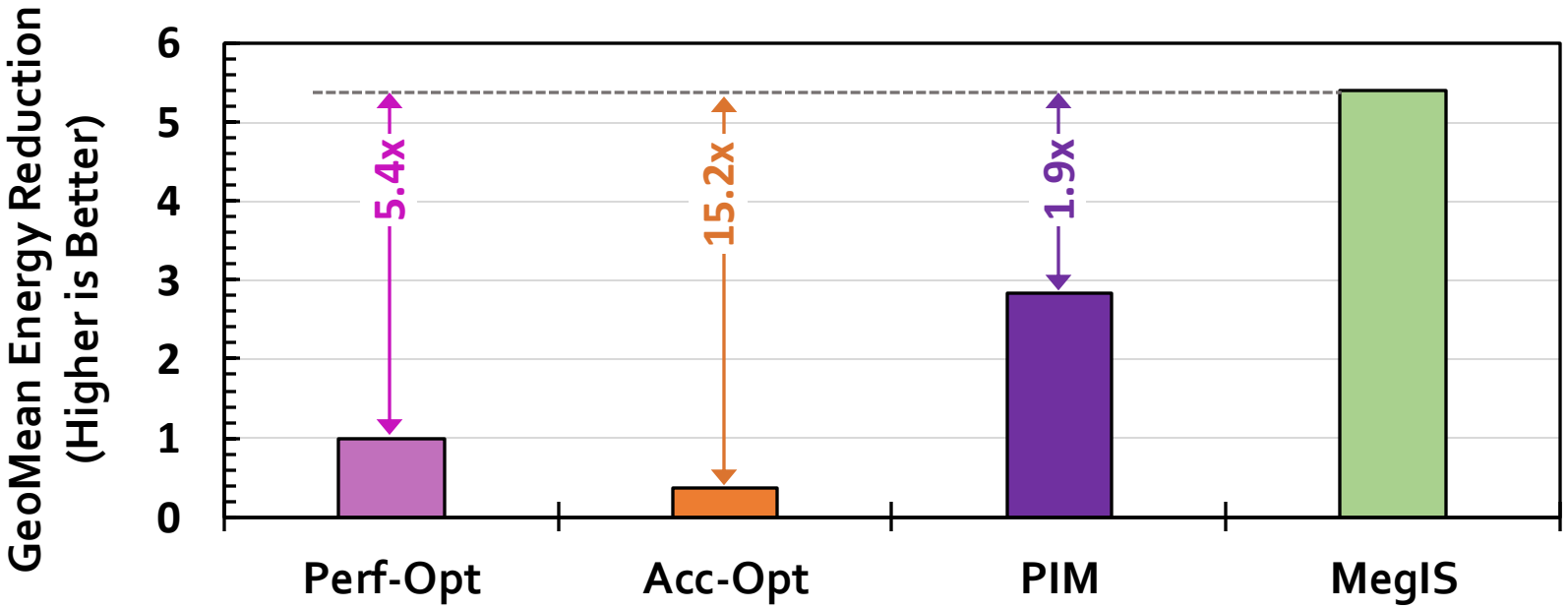




MegIS provides significant speedup over the PIM baseline

Evaluation: Reduction in Energy Consumption

- On average across different input sets and SSDs



MegIS provides significant energy reduction over the Performance-Optimized, Accuracy-Optimized, and PIM baselines

Evaluation: Accuracy, Area, and Power

Accuracy

- **Same accuracy** as the **accuracy-optimized** baseline
- **Significantly higher accuracy** than the **performance-optimized** and **PIM** baselines
  - 4.6 – 5.2× higher F1 scores
  - 3 – 24% lower L1 norm error

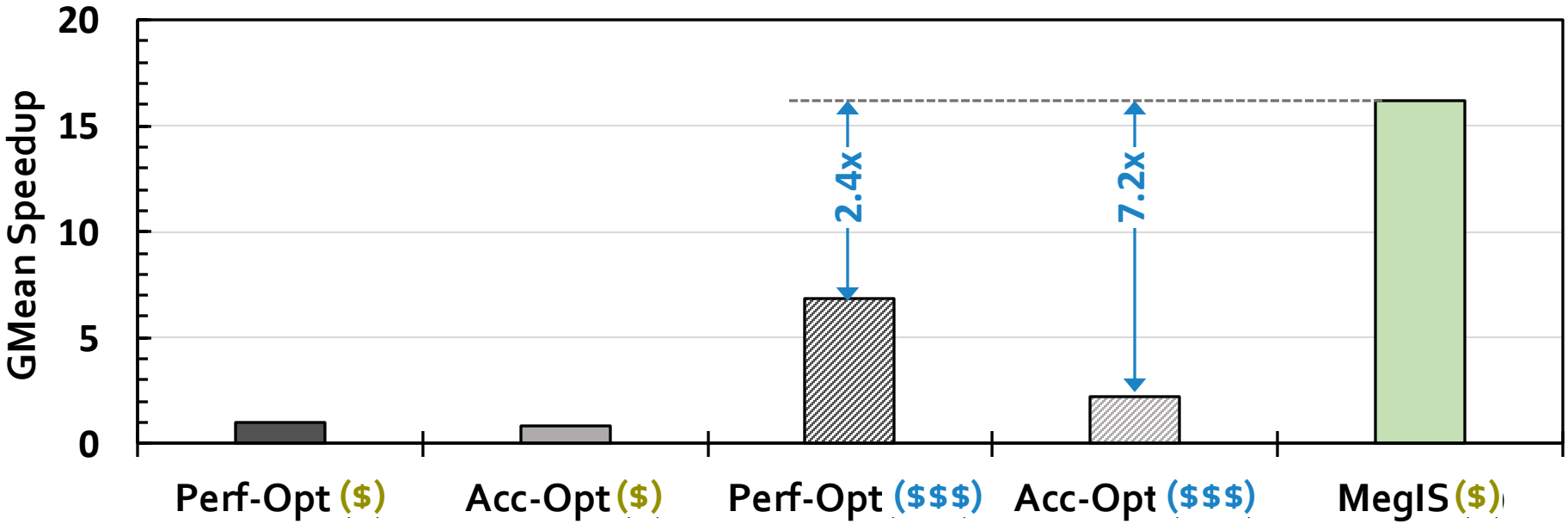
Area and Power

Total for an 8-channel SSD:

- **Area:** 0.04 mm^2 *(Only **1.7%** of the area of three ARM Cortex R4 cores in an SSD controller)*
- **Power:** 7.658 mW

Evaluation: System Cost-Efficiency

- **Cost-optimized system ($):** With SSD-C and 64-GB DRAM
- **Performance-optimized system ($$$):** With SSD-P and 1-TB DRAM

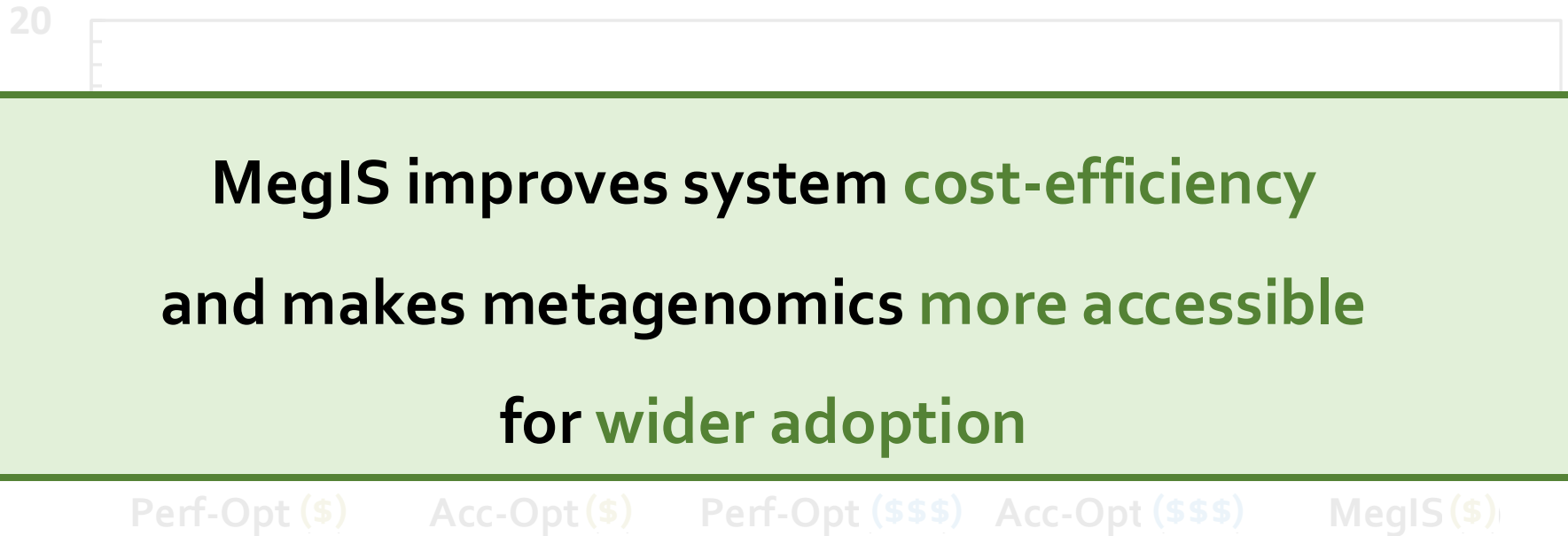


MegIS outperforms the baselines

***even* when running on a much less costly system**

Evaluation: System Cost-Efficiency

- **Cost-optimized system ($):** With SSD-C and 64-GB DRAM
- **Performance-optimized system ($$$):** With SSD-P and 1-TB DRAM



MegIS improves system cost-efficiency

and makes metagenomics more accessible

for wider adoption

MegIS outperforms the baselines

even **when running on a much less costly system**

More in the Paper

- MegIS's performance when running in-storage processing operations on the **cores existing in the SSD controller**
- MegIS's performance when using the same accelerators **outside SSD**
- **Sensitivity analysis with varying**
  - Database sizes
  - Memory capacities
  - #SSDs
  - #Channels
  - #Samples
- MegIS's performance for **abundance estimation**

More in the Paper

MegIS: High-Performance, Energy-Efficient, and Low-Cost Metagenomic Analysis with In-Storage Processing

Nika Mansouri Ghiasi[1] Mohammad Sadrosadati[1] Harun Mustafa[1] Arvid Gollwitzer[1]
Can Firtina[1] Julien Eudine[1] Haiyu Mao[1] Joël Lindegger[1] Meryem Banu Cavlak[1]
Mohammed Alser[1] Jisung Park[2] Onur Mutlu[1]
[1]ETH Zürich [2]POSTECH

- Database sizes
- Memory capacities
- #SSDs
- #Channels
- #Samples

• MegIS's performance for abundance estimation

https://arxiv.org/abs/2406.19113

Conclusion

Metagenomic analysis suffers from
significant storage I/O data movement overhead

MegIS

The *first* **in-storage processing** system for *end-to-end* metagenomic analysis
Leverages and orchestrates **processing inside** and **outside** the storage system

Improves performance

2.7×–37.2× over **performance-optimized** software
6.9×–100.2× over **accuracy-optimized** software
1.5×–5.1× over **hardware-accelerated PIM** baseline

High accuracy

Same as accuracy-optimized
4.8× higher F1 scores
over **performance-optimized/PIM**

Reduces energy consumption

5.4× over **performance-optimized** software
15.2× over **accuracy-optimized** software
1.9× over **hardware-accelerated PIM** baseline

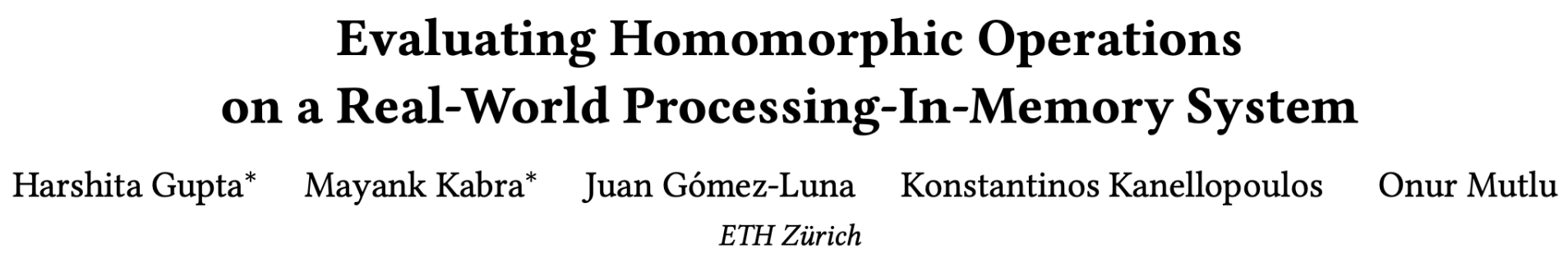
Low area overhead

1.7% of the three cores
in an SSD controller

Homomorphic Operations on Real PIM Systems

Homomorphic Operations on Real PIM Systems

- Harshita Gupta, Mayank Kabra, Juan Gómez-Luna, Konstantinos Kanellopoulos, and Onur Mutlu,
  "Evaluating Homomorphic Operations on a Real-World Processing-In-Memory System"
  *Proceedings of the 2023 IEEE International Symposium on Workload Characterization Poster Session (**IISWC**)*, Ghent, Belgium, October 2023.
  [arXiv version]
  [Lightning Talk Slides (pptx) (pdf)]
  [Poster (pptx) (pdf)]



Evaluating Homomorphic Operations on a Real-World Processing-In-Memory System

Harshita Gupta* Mayank Kabra* Juan Gómez-Luna Konstantinos Kanellopoulos Onur Mutlu

ETH Zürich

https://arxiv.org/pdf/2309.06545.pdf

Side Channels on PIM Systems

PIM Amplifies Covert & Side Channels

Amplifying Main Memory-Based Timing Covert and Side Channels using Processing-in-Memory Operations

Konstantinos Kanellopoulos[†*] F. Nisa Bostancı[†*] Ataberk Olgun[†]
A. Giray Yağlıkçı[†] İsmail Emir Yüksel[†] Nika Mansouri Ghiasi[†]
Zülal Bingöl[†‡] Mohammad Sadrosadati[†] Onur Mutlu[†]

[†]ETH Zürich [‡]Bilkent University

https://arxiv.org/pdf/2404.11284

Distributed ML Training on Real PIM Systems

Accelerating ML Training on Real PIM Systems

- **To appear at PACT 2024**

Analysis of Distributed Optimization Algorithms on a Real Processing-In-Memory System

Steve Rhyner[1] Haocong Luo[1] Juan Gómez-Luna[2] Mohammad Sadrosadati[1]
Jiawei Jiang[3] Ataberk Olgun[1] Harshita Gupta[1] Ce Zhang[4] Onur Mutlu[1]

[1]ETH Zurich [2]NVIDIA [3]Wuhan University [4]University of Chicago

https://arxiv.org/pdf/2404.07164

Reinforcement Learning on Real PIM Systems

SwiftRL

- Kailash Gogineni, Sai Santosh Dayapule, Juan Gomez-Luna, Karthikeya Gogineni, Peng Wei, Tian Lan, Mohammad Sadrosadati, Onur Mutlu, Guru Venkataramani, **"SwiftRL: Towards Efficient Reinforcement Learning on Real Processing-In-Memory Systems"**
  *Proceedings of the 2024 IEEE International Symposium on Performance Analysis of Systems and Software (**ISPASS**)*, Indianapolis, Indiana, May 2024.
  [Slides (pptx) (pdf)]
  [arXiv version]

SwiftRL: Towards Efficient Reinforcement Learning on Real Processing-In-Memory Systems

Kailash Gogineni[1] Sai Santosh Dayapule[1] Juan Gómez-Luna[2] Karthikeya Gogineni[3]
Peng Wei[1] Tian Lan[1] Mohammad Sadrosadati[2] Onur Mutlu[2] Guru Venkataramani[1]
[1]George Washington University, USA [2]ETH Zürich, Switzerland [3]Independent

https://arxiv.org/pdf/2405.03967

SwiftRL: Summary

- **Adapted and implemented** RL algorithms on a PIM architecture for exploring memory-centric systems in RL training
- Explored **optimization strategies** for enhancing RL workload performance across
  - **various data types,**
  - **sampling strategies (SEQ, RAN, STR)**
- Compared **PIM-based** Q-learning & SARSA on UPMEM PIM (2000 cores) to **CPU & GPU**
- Achieved near-linear **scaling of 15x** in performance with a **16x increase in PIM cores (125 to 2000)**

MATSA

MATSA

- Ivan Fernandez, Christina Giannoula, Aditya Manglik, Ricardo Quislant, Nika Mansouri Ghiasi, Juan Gomez Luna, Eladio Gutierrez, Oscar Plata and Onur Mutlu,
  "MATSA: An MRAM-Based Energy-Efficient Accelerator for Time Series Analysis"
  IEEE Access, March 2024.
  [arXiv version]
  [IEEE Access version]

Accelerating Time Series Analysis via Processing using Non-Volatile Memories

Ivan Fernandez[§†¶] *Christina Giannoula[†‡] *Aditya Manglik[†] Ricardo Quislant[§] Nika Mansouri Ghiasi[†]
Juan Gómez-Luna[†] Eladio Gutierrez[§] Oscar Plata[§] Onur Mutlu[†]
[§]*University of Malaga* [†]*ETH Zürich* [¶]*Barcelona Supercomputing Center* [‡]*National Technical University of Athens*

https://arxiv.org/pdf/2211.04369

ApHMM

ApHMM

- Can Firtina, Kamlesh Pillai, Gurpreet S. Kalsi, Bharathwaj Suresh, Damla Senol Cali, Jeremie S. Kim, Taha Shahroodi, Meryem Banu Cavlak, Joel Lindegger, Mohammed Alser, Juan Gomez Luna, Sreenivas Subramoney and Onur Mutlu,
"ApHMM: Accelerating Profile Hidden Markov Models for Fast and Energy-efficient Genome Analysis"
ACM Transactions on Architecture and Code Optimization (***TACO***), February 2024.
[arXiv version]
[ApHMM Source Code]
[ACM Digital Library version]
[Talk Video HiPEAC]

ApHMM: Accelerating Profile Hidden Markov Models for Fast and Energy-Efficient Genome Analysis

Can Firtina[1] Kamlesh Pillai[2] Gurpreet S. Kalsi[2] Bharathwaj Suresh[2] Damla Senol Cali[3]
Jeremie S. Kim[1] Taha Shahroodi[4] Meryem Banu Cavlak[1] Joel Lindegger[1] Mohammed Alser[1]
Juan Gómez Luna[1] Sreenivas Subramoney[2] Onur Mutlu[1]

https://arxiv.org/pdf/2207.09765

Executive Summary

Motivation: Graph structures such as **profile Hidden Markov Models (pHMMs)** are commonly used to accurately analyze biological sequences

Problem: The parameters used in pHMMs are mainly trained and used with a **computationally intensive Baum-Welch algorithm**, causing major performance and energy overhead for many genomics workloads

Goal: Enable rapid, power-efficient, and flexible use of pHMMs for genomics workloads

ApHMM: the first flexible and hardware-software accelerator for pHMMs that can

1) Substantially reduce unnecessary data storage, data movement, and computations by effectively co-designing hardware and software together
2) Provide a flexible design to support several genomics workloads that use pHMMs

Key Results: Our ASIC implementation compared to CPU, GPU, and FPGA baselines across 3 workloads

- **15.55×–260.03×, 1.83×–5.34×, and 27.97× better performance**
- **Up to 2622.94×** reduction in **energy consumption**

https://github.com/CMU-SAFARI/ApHMM-GPU

RUBICON

RUBICON

- Gagandeep Singh, Mohammed Alser, Kristof Denolf, Can Firtina, Alireza Khodamoradi, Meryem Banu Cavlak, Henk Corporaal and Onur Mutlu, **"RUBICON: A Framework for Designing Efficient Deep Learning-Based Genomic Basecallers"**
  Genome Biology, February 2024.
  [arXiv version]
  [Journal Article]
  [RUBICON Source Code]

RUBICON: A Framework for Designing Efficient Deep Learning-Based Genomic Basecallers

Gagandeep Singh[a,c] Mohammed Alser[a] Kristof Denolf[c]
Can Firtina[a,*] Alireza Khodamoradi[c] Meryem Banu Cavlak[a]
Henk Corporaal[b] Onur Mutlu[a,*]

[a]Department of Information Technology and Electrical Engineering, ETH Zürich, Switzerland
[b]Department of Electrical Engineering, Eindhoven University of Technology, The Netherlands
[c]Research and Advanced Development, AMD, USA

https://arxiv.org/pdf/2211.03079

Utopia

Better Virtual Memory: Utopia

Konstantinos Kanellopoulos, Rahul Bera, Kosta Stojiljkovic, Nisa Bostanci, Can Firtina, Rachata Ausavarungnirun, Rakesh Kumar, Nastaran Hajinazar, Mohammad Sadrosadati, Nandita Vijaykumar, and Onur Mutlu,
"Utopia: Fast and Efficient Address Translation via Hybrid Restrictive & Flexible Virtual-to-Physical Address Mappings"
*Proceedings of the 56th International Symposium on Microarchitecture (**MICRO**)*, Toronto, ON, Canada, November 2023.
[Slides (pptx) (pdf)]
[arXiv version]
[Utopia Source Code]

Utopia: Fast and Efficient Address Translation via Hybrid Restrictive & Flexible Virtual-to-Physical Address Mappings

Konstantinos Kanellopoulos[1] Rahul Bera[1] Kosta Stojiljkovic[1] Nisa Bostanci[1] Can Firtina[1]
Rachata Ausavarungnirun[2] Rakesh Kumar[3] Nastaran Hajinazar[4] Mohammad Sadrosadati[1]
Nandita Vijaykumar[5] Onur Mutlu[1]

[1]ETH Zürich [2]King Mongkut's University of Technology North Bangkok
[3]Norwegian University of Science and Technology [4]Intel Labs [5]University of Toronto

https://arxiv.org/abs/2211.12205

Utopia: Executive Summary

Problem: Conventional virtual memory (VM) frameworks enable a virtual address to flexibly map to any physical address. This flexibility necessitates large translation structures leading to:

(1) high translation **latency** and (2) large translation-induced **interference** in the memory hierarchy

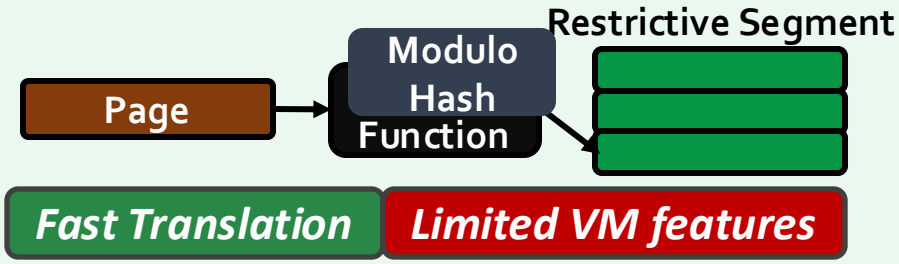
Motivation: Restricting the address mapping leads to compact translation structures and reduces the overheads of address translation. Doing so across the **entire memory** has two major drawbacks:

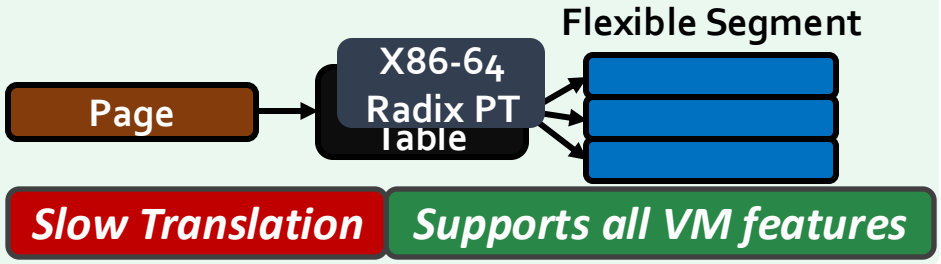
(1) Limits core VM functionalities (e.g., data sharing)

(2) Increases swapping activity in the presence of free physical memory

Key Idea: Utopia is a new hybrid virtual-to-physical address mapping scheme that allows both **flexible** and **restrictive** hash-based address mappings to **harmoniously co-exist** in the system

Utopia manages physical memory using two types of physical memory segments:





Key Results: Outperforms (i) the state-of-the-art **contiguity-aware translation scheme by 13%,** and (ii) achieves **95% of the performance of an ideal perfect-TLB**

https://github.com/CMU-SAFARI/Utopia

Victima

Better Virtual Memory: Victima

Konstantinos Kanellopoulos, Hong Chul Nam, F. Nisa Bostanci, Rahul Bera, Mohammad Sadrosadati, Rakesh Kumar, Davide Basilio Bartolini, and Onur Mutlu,
"Victima: Drastically Increasing Address Translation Reach by Leveraging Underutilized Cache Resources"
*Proceedings of the 56th International Symposium on Microarchitecture (**MICRO**)*, Toronto, ON, Canada, November 2023.
[Slides (pptx) (pdf)]
[arXiv version]
[Victima Source Code (Officially Artifact Evaluated with All Badges)]
Officially artifact evaluated as available, functional, reusable and reproducible.
Distinguished artifact award at MICRO 2023.

Victima: Drastically Increasing Address Translation Reach by Leveraging Underutilized Cache Resources

Konstantinos Kanellopoulos[1] Hong Chul Nam[1] F. Nisa Bostanci[1] Rahul Bera[1]
Mohammad Sadrosadati[1] Rakesh Kumar[2] Davide Basilio Bartolini[3] Onur Mutlu[1]

[1]ETH Zürich [2]Norwegian University of Science and Technology [3]Huawei Zurich Research Center

https://arxiv.org/pdf/2310.04158

Executive Summary

Problem: Address translation is a major **performance bottleneck** in data-intensive workloads

Large datasets and irregular memory access patterns lead to **frequent L2 TLB misses** (e.g., 20-50 MPKI) and **frequent high-latency** (e.g., 100-150 cycles) page table walks (PTW)

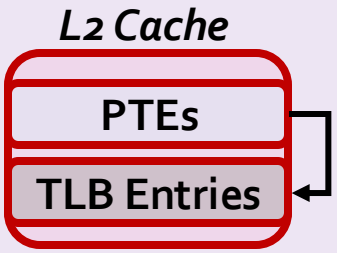
Motivation: Increasing the translation reach (i.e., memory covered by the TLBs) reduces PTWs. However, employing large TLBs leads to increased area, power and latency overheads.

Opportunity: Increase the translation reach of the TLB hierarchy by storing the existing TLB entries within the ***existing cache hierarchy***

Victima: New software-transparent scheme that drastically increases the address translation reach of the processor's TLB hierarchy by leveraging the underutilized cache resources

Key Idea:

Transform L2 cache blocks that store PTEs into blocks that store TLB entries



Key Benefits:

+ **Efficient** in native/virtualized environments
+ **Fully transparent** to application/OS software
+ **Compatible** with huge page schemes

Key Results: Victima (i) **outperforms by 5.1%** a state-of-the-art large TLB design and (ii) achieves **similar performance** to an optimistically fast 128K-entry L2 TLB

https://github.com/CMU-SAFARI/Victima

Sectored DRAM

Sectored DRAM

- Ataberk Olgun, F. Nisa Bostanci, Geraldo F. Oliveira, Yahya Can Tugrul, Rahul Bera, A. Giray Yaglikci, Hasan Hassan, Oguz Ergin, and Onur Mutlu, **"Sectored DRAM: A Practical Energy-Efficient and High-Performance Fine-Grained DRAM Architecture"**
  ACM Transactions on Architecture and Code Optimization (***TACO***), [online] June 2024.
  [arXiv version]
  [ACM Digital Library version]

Sectored DRAM: A Practical Energy-Efficient and High-Performance Fine-Grained DRAM Architecture

Ataberk Olgun[§] F. Nisa Bostancı[§†] Geraldo F. Oliveira[§] Yahya Can Tuğrul[§†] Rahul Bera[§]
A. Giray Yağlıkcı[§] Hasan Hassan[§] Oğuz Ergin[†] Onur Mutlu[§]

https://arxiv.org/pdf/2207.13795

SimplePIM

Adoption: How to Ease **Programmability**? (II)

- Jinfan Chen, Juan Gómez-Luna, Izzat El Hajj, YuXin Guo, and Onur Mutlu,
  "SimplePIM: A Software Framework for Productive and Efficient Processing in Memory"
  *Proceedings of the 32nd International Conference on Parallel Architectures and Compilation Techniques (**PACT**)*, Vienna, Austria, October 2023.

SimplePIM: A Software Framework for Productive and Efficient Processing-in-Memory

Jinfan Chen[1] Juan Gómez-Luna[1] Izzat El Hajj[2] Yuxin Guo[1] Onur Mutlu[1]
[1]ETH Zürich [2]American University of Beirut

SimplePIM:

A Software Framework for Productive and Efficient Processing-in-Memory

Jinfan Chen, Juan Gómez Luna, Izzat El Hajj, Yuxin Guo, Onur Mutlu

https://arxiv.org/pdf/2310.01893.pdf

https://github.com/CMU-SAFARI/SimplePIM

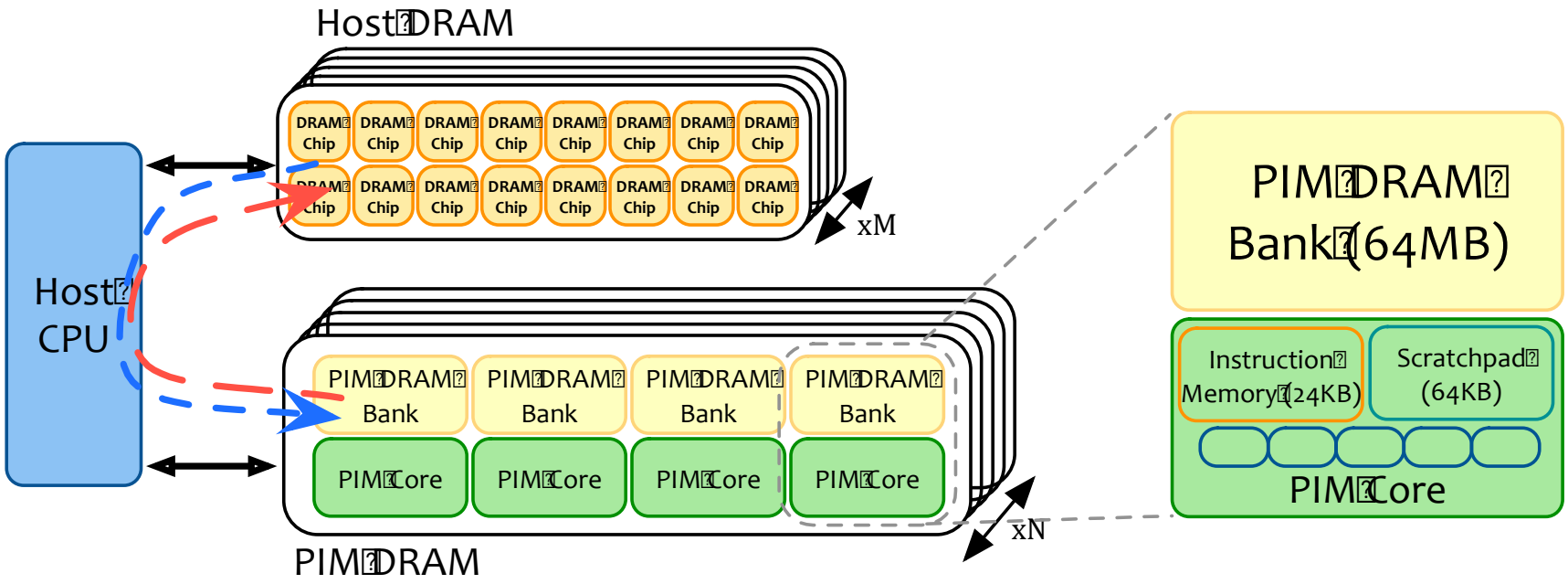
ETH Zürich

SAFARI

Executive Summary

- Processing-in-Memory (PIM) promises to alleviate the *data movement bottleneck*
- Real PIM hardware is now available, e.g., UPMEM PIM
- However, programming real PIM hardware is challenging, e.g.:
  - Distribute data across PIM memory banks,
  - Manage data transfers between host cores and PIM cores, and between PIM cores,
  - Launch PIM kernels on the PIM cores, etc.
- SimplePIM is a high-level programming framework for real PIM hardware
  - Iterators such as `map`, `reduce`, and `zip`
  - Collective communication with `broadcast`, `scatter`, and `gather`
- Implementation on UPMEM and evaluation with six different workloads
  - Reduction, vector add, histogram, linear/logistic regression, K-means
  - 4.4x fewer lines of code compared to hand-optimized code
  - Between 15% and 43% faster than hand-optimized code for three workloads
- Source code: https://github.com/CMU-SAFARI/SimplePIM

A State-of-the-Art PIM System



- In our work, we use the UPMEM PIM architecture
  - General-purpose processing cores called *DRAM Processing Units* (*DPUs*)
    - Up to 24 PIM threads, called *tasklets*
    - 32-bit integer arithmetic, but multiplication/division are emulated*, as well as floating-point operations
  - 64-MB DRAM bank (*MRAM*), 64-KB scratchpad (*WRAM*)

* 8-bit integer multiplication is natively supported

Programming a PIM System (I)

- Example: Hand-optimized histogram with UPMEM SDK

```
... // Initialize global variables and functions for histogram
int main_kernel() {
  if (tasklet_id == 0)
    mem_reset(); // Reset the heap
  ... // Initialize variables and the histogram
  T *input_buff_A = (T*)mem_alloc(2048); // Allocate buffer in scratchpad memory

  for (unsigned int byte_index = base_tasklet; byte_index < input_size; byte_index += stride) {
    // Boundary checking
    uint32_t l_size_bytes = (byte_index + 2048 >= input_size) ? (input_size - byte_index) : 2048;
    // Load scratchpad with a DRAM block
    mram_read((const __mram_ptr void*)(mram_base_addr_A + byte_index), input_buff_A, l_size_bytes);
    // Histogram calculation
    histogram(hist, bins, input_buff_A, l_size_bytes/sizeof(uint32_t));
  }
  ...
  barrier_wait(&my_barrier); // Barrier to synchronize PIM threads
  ... // Merging histograms from different tasklets into one histo_dpu
  // Write result from scratchpad to DRAM
  if (tasklet_id == 0)
    if (bins * sizeof(uint32_t) <= 2048)
      mram_write(histo_dpu, (__mram_ptr void*)mram_base_addr_histo, bins * sizeof(uint32_t));
    else
      for (unsigned int offset = 0; offset < ((bins * sizeof(uint32_t)) >> 11); offset++) {
        mram_write(histo_dpu + (offset << 9), (__mram_ptr void*)(mram_base_addr_histo +
                   (offset << 11)), 2048);
      }
  return 0;
}
```

Programming a PIM System (II)

- PIM programming is challenging
  - Manage data movement between host DRAM and PIM DRAM
    - Parallel, serial, broadcast, and gather/scatter transfers
  - Manage data movement between PIM DRAM bank and scratchpad
    - 8-byte aligned and maximum of 2,048 bytes
  - Multithreaded programming model
  - Inter-thread synchronization
    - Barriers, handshakes, mutexes, and semaphores

Our Goal

Design a high-level programming framework that abstracts these hardware-specific complexities and provides a clean yet powerful interface for ease of use and high program performance

The SimplePIM Programming Framework

- SimplePIM provides standard abstractions to build and deploy applications on PIM systems
  - Management interface
    - Metadata for PIM-resident arrays
  - Communication interface
    - Abstractions for host-PIM and PIM-PIM communication
  - Processing interface
    - Iterators (`map`, `reduce`, `zip`) to implement workloads

Management Interface

- Metadata for PIM-resident arrays
  - `array_meta_data_t` describes a PIM-resident array
  - `simple_pim_management_t` for managing PIM-resident arrays

- `lookup`: Retrieves all relevant information of an array

```
array_meta_data_t* simple_pim_array_lookup(const char* id,
simple_pim_management_t* management);
```

- `register`: Registers the metadata of an array

```
void simple_pim_array_register(array_meta_data_t* meta_data,
simple_pim_management_t* management);
```

- `free`: Removes the metadata of an array

```
void simple_pim_array_free(const char* id, simple_pim_management_t* management);
```

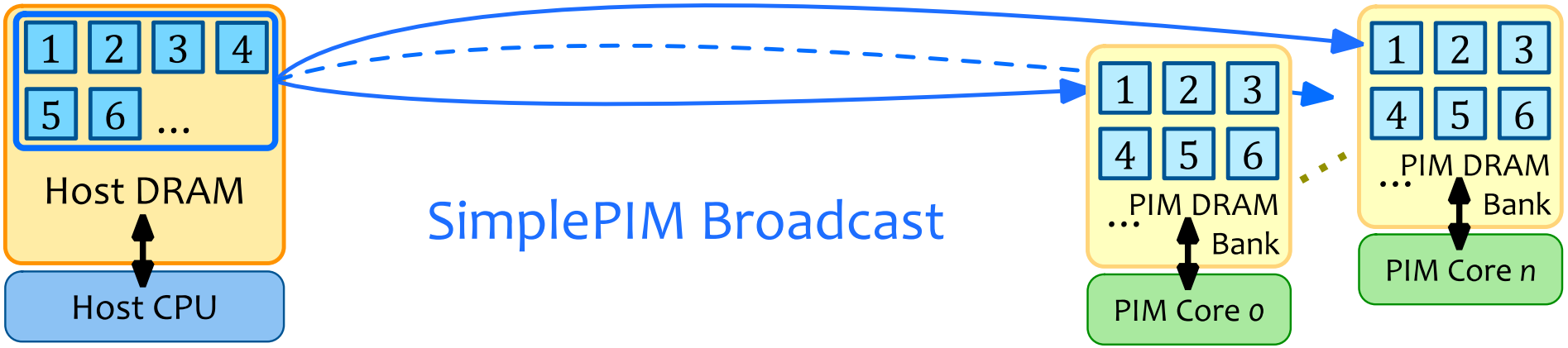
The SimplePIM Programming Framework

- SimplePIM provides standard abstractions to build and deploy applications on PIM systems
  - Management interface
    - Metadata for PIM-resident arrays
  - Communication interface
    - Abstractions for host-PIM and PIM-PIM communication
  - Processing interface
    - Iterators (`map`, `reduce`, `zip`) to implement workloads

Host-to-PIM Communication: Broadcast

- SimplePIM Broadcast
  - Transfers a host array to all PIM cores in the system

```
void simple_pim_array_broadcast(char* const id, void* arr, uint64_t len,
uint32_t type_size, simple_pim_management_t* management);
```



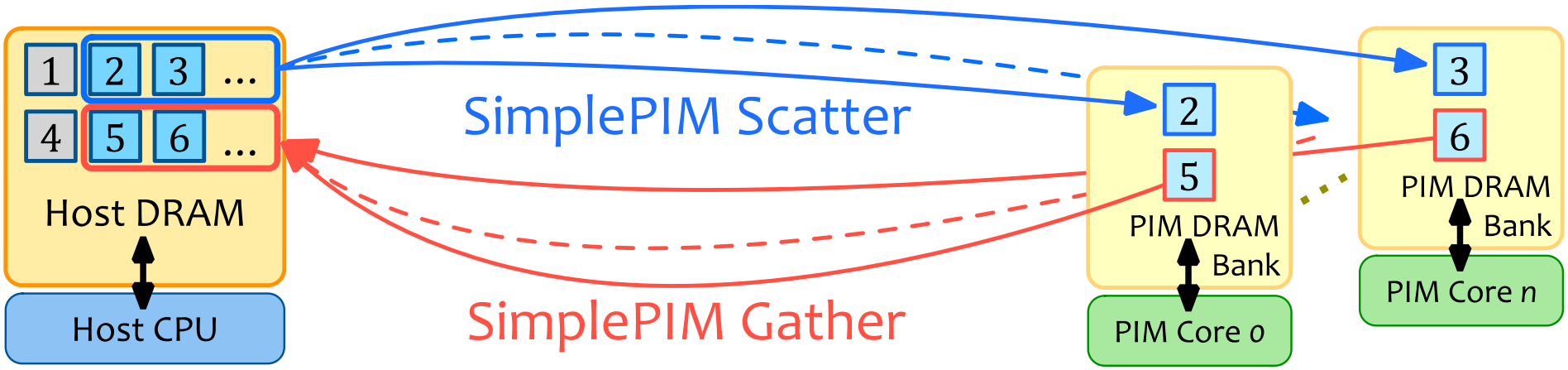
Host-to-PIM Communication: Scatter/Gather

- SimplePIM Scatter
  - Distributes an array to PIM DRAM banks

```
void simple_pim_array_scatter(char* const id, void* arr, uint64_t len,
uint32_t type_size, simple_pim_management_t* management);
```

- SimplePIM Gather
  - Collects portions of an array from PIM DRAM banks

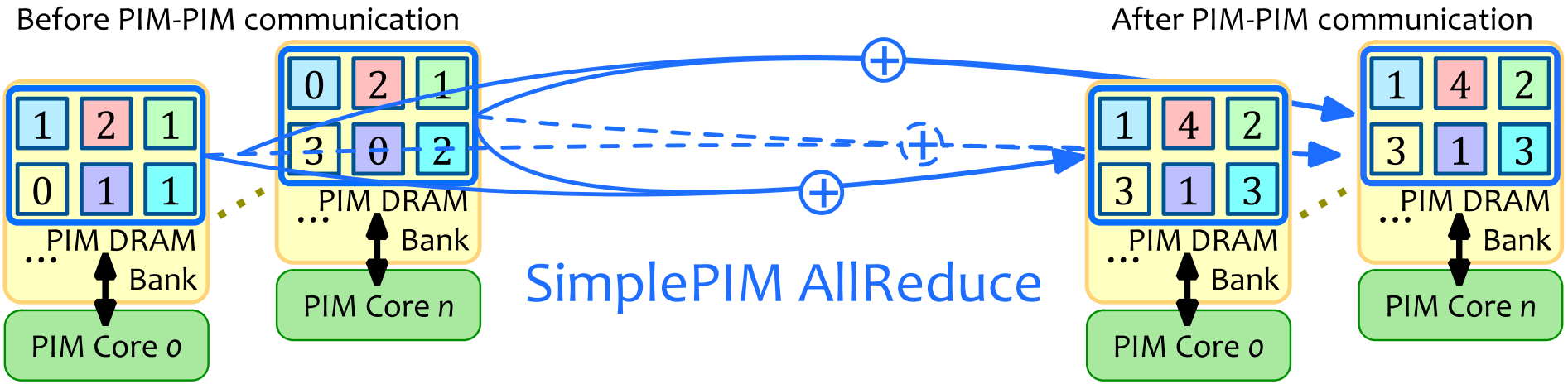
```
void* simple_pim_array_gather(char* const id, simple_pim_management_t*
management);
```



PIM-PIM Communication: AllReduce

- SimplePIM AllReduce
  - Used for algorithm synchronization
  - The programmer specifies an accumulative function

```
void simple_pim_array_allreduce(char* const id, handle_t* handle,
simple_pim_management_t* management);
```



PIM-PIM Communication: AllGather

- SimplePIM AllGather
  - Combines array pieces and distributes the complete array to all PIM cores

```
void simple_pim_array_allgather(char* const id, char* new_id,
simple_pim_management_t* management);
```

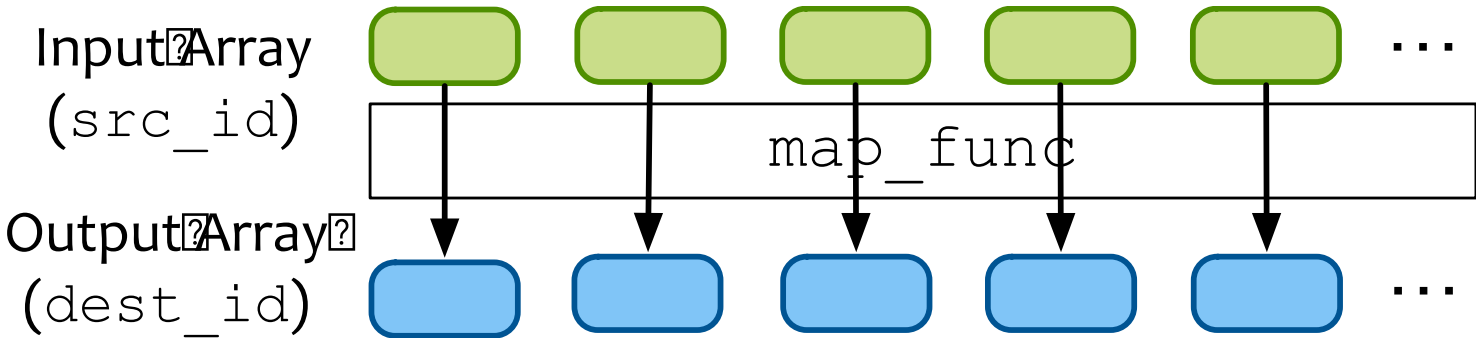
The SimplePIM Programming Framework

- SimplePIM provides standard abstractions to build and deploy applications on PIM systems
  - Management interface
    - Metadata for PIM-resident arrays
  - Communication interface
    - Abstractions for host-PIM and PIM-PIM communication
  - Processing interface
    - Iterators (`map`, `reduce`, `zip`) to implement workloads

Processing Interface: Map

- Array Map
  - Applies `map_func` to every element of the data array

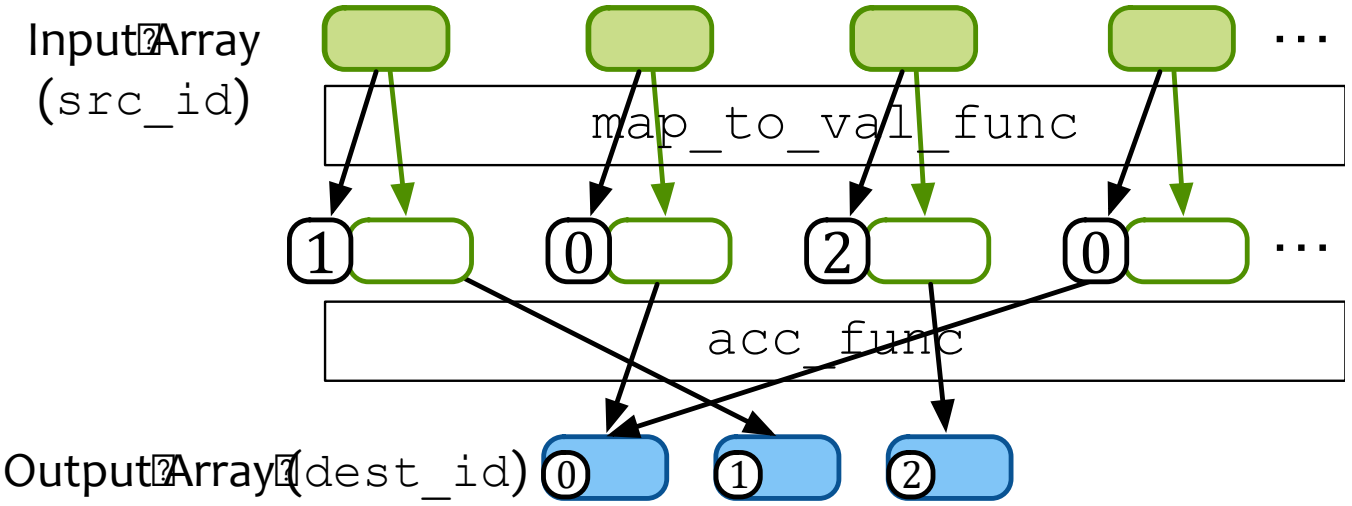
```
void simple_pim_array_map(const char* src_id, const char* dest_id,
uint32_t output_type, handle_t* handle, simple_pim_management_t* management);
```



Processing Interface: Reduction

- Array Reduction
  - The `map_to_val_func` function transforms an input element to an output value and an output index
  - The `acc_func` function accumulates the output values onto the output array

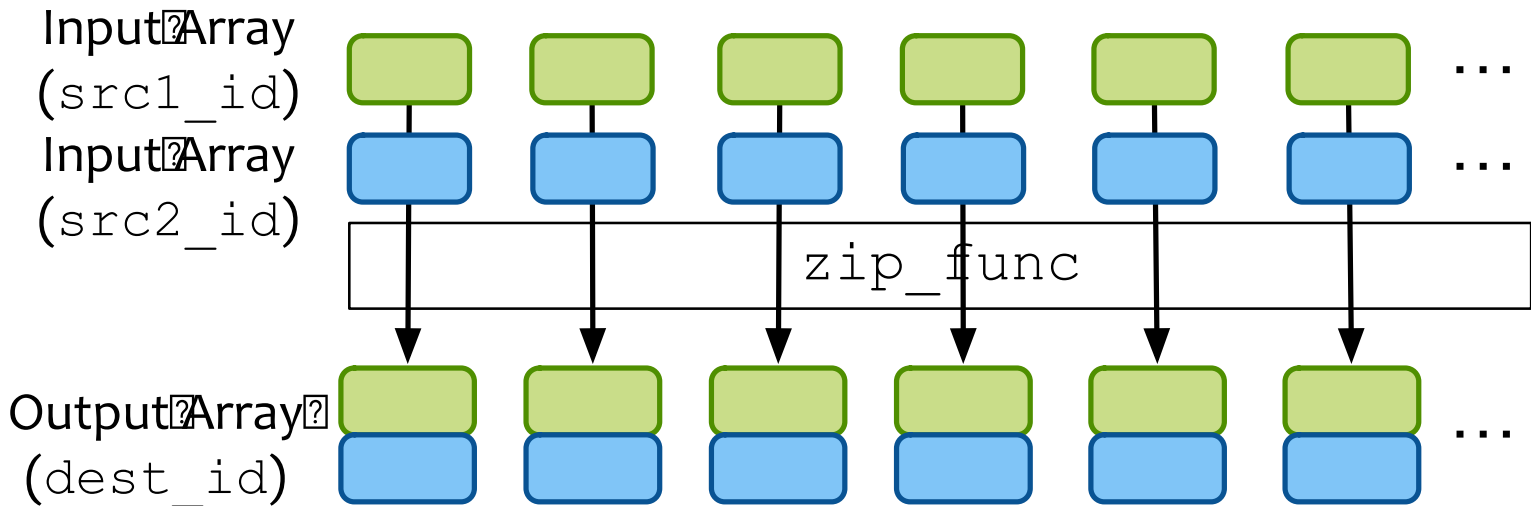
```
void simple_pim_array_red(const char* src_id, const char* dest_id,
uint32_t output_type, uint32_t output_len, handle_t* handle,
simple_pim_management_t* management);
```



Processing Interface: Zip

- Array Zip
  - Takes two input arrays and combines their elements into an output array

```
void simple_pim_array_zip(const char* src1_id, const char* src2_id,
const char* dest_id, simple_pim_management_t* management);
```



General Code Optimizations

- Strength reduction
- Loop unrolling
- Avoiding boundary checks
- Function inlining
- Adjustment of data transfer sizes

More in the Paper

- Strength reduction
- Loop unrolling

SimplePIM: A Software Framework for Productive and Efficient Processing-in-Memory

Jinfan Chen[1] Juan Gómez-Luna[1] Izzat El Hajj[2] Yuxin Guo[1] Onur Mutlu[1]
[1]ETH Zürich [2]American University of Beirut

- Function inlining
- Adju

https://arxiv.org/pdf/2310.01893.pdf

Evaluation Methodology

- Evaluated system
  - UPMEM PIM system with 2,432 PIM cores with 159 GB of PIM DRAM
- Real-world Benchmarks
  - Vector addition
  - Reduction
  - Histogram
  - K-Means
  - Linear regression
  - Logistic regression
- Comparison to hand-optimized codes in terms of programming productivity and performance

Productivity Improvement (I)

- Example: Hand-optimized histogram with UPMEM SDK

```
... // Initialize global variables and functions for histogram
int main_kernel() {
  if (tasklet_id == 0)
    mem_reset(); // Reset the heap
  ... // Initialize variables and the histogram
  T *input_buff_A = (T*)mem_alloc(2048); // Allocate buffer in scratchpad memory

  for (unsigned int byte_index = base_tasklet; byte_index < input_size; byte_index += stride) {
    // Boundary checking
    uint32_t l_size_bytes = (byte_index + 2048 >= input_size) ? (input_size - byte_index) : 2048;
    // Load scratchpad with a DRAM block
    mram_read((const __mram_ptr void*)(mram_base_addr_A + byte_index), input_buff_A, l_size_bytes);
    // Histogram calculation
    histogram(hist, bins, input_buff_A, l_size_bytes/sizeof(uint32_t));
  }
  ...
  barrier_wait(&my_barrier); // Barrier to synchronize PIM threads
  ... // Merging histograms from different tasklets into one histo_dpu
  // Write result from scratchpad to DRAM
  if (tasklet_id == 0)
    if (bins * sizeof(uint32_t) <= 2048)
      mram_write(histo_dpu, (__mram_ptr void*)mram_base_addr_histo, bins * sizeof(uint32_t));
    else
      for (unsigned int offset = 0; offset < ((bins * sizeof(uint32_t)) >> 11); offset++) {
        mram_write(histo_dpu + (offset << 9), (__mram_ptr void*)(mram_base_addr_histo +
                   (offset << 11)), 2048);
      }
  return 0;
}
```

Productivity Improvement (II)

- Example: SimplePIM histogram

```
// Programmer-defined functions in the file "histo_filepath"
void init_func (uint32_t size, void* ptr) {
  char* casted_value_ptr = (char*) ptr;
  for (int i = 0; i < size; i++)
    casted_value_ptr[i] = 0;
}

void acc_func (void* dest, void* src) {
  *(uint32_t*)dest += *(uint32_t*)src;
}

void map_to_val_func (void* input, void* output, uint32_t* key) {
  uint32_t d = *((uint32_t*)input);
  *(uint32_t*)output = 1;
  *key = d * bins >> 12;
}

// Host side handle creation and iterator call
handle_t* handle = simple_pim_create_handle("histo_filepath", REDUCE, NULL, 0);

// Transfer (scatter) data to PIM, register as "t1"
simple_pim_array_scatter("t1", src, bins, sizeof(T), management);

// Run histogram on "t1" and produce "t2"
simple_pim_array_red("t1", "t2", sizeof(T), bins, handle, management);
```

Productivity Improvement (III)

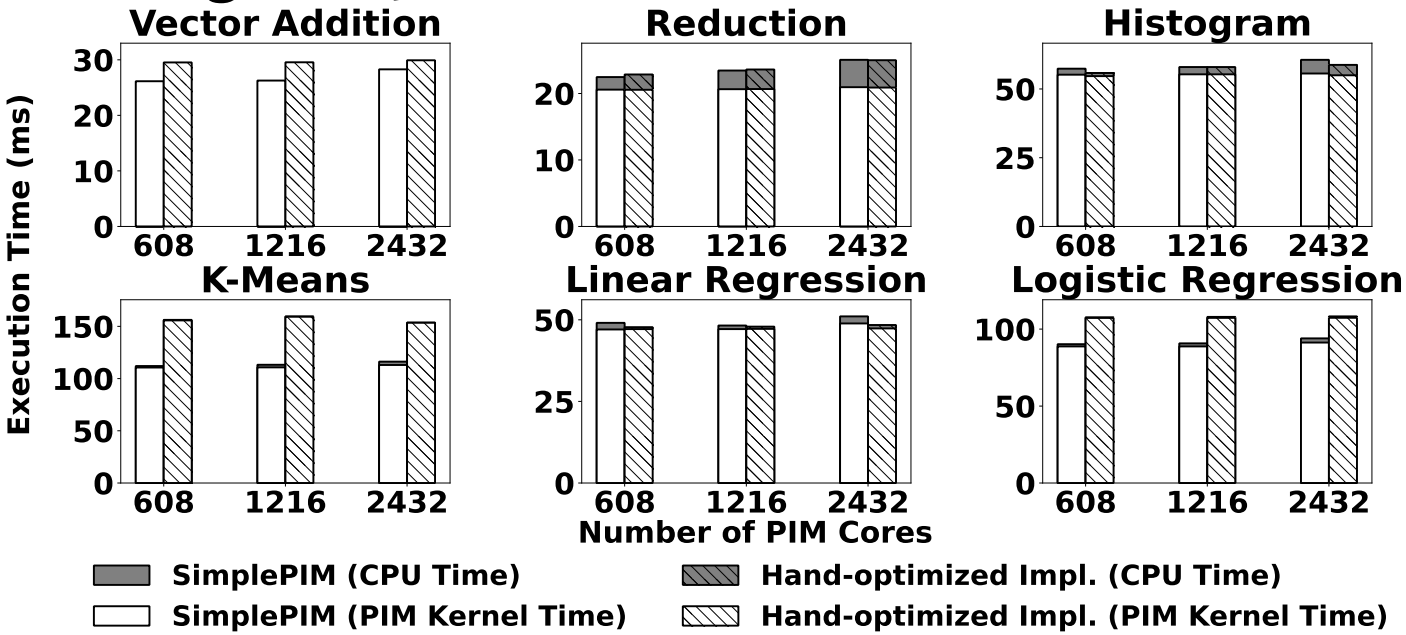
- Lines of code (LoC) reduction

	SimplePIM	Hand-optimized	LoC Reduction
Reduction	14	83	5.93×
Vector Addition	14	82	5.86×
Histogram	21	114	5.43×
Linear Regression	48	157	3.27×
Logistic Regression	59	176	2.98×
K-Means	68	206	3.03×

SimplePIM reduces the number of lines of effective code by a factor of 2.98× to 5.93×

Performance Evaluation (I)

- Weak scaling analysis

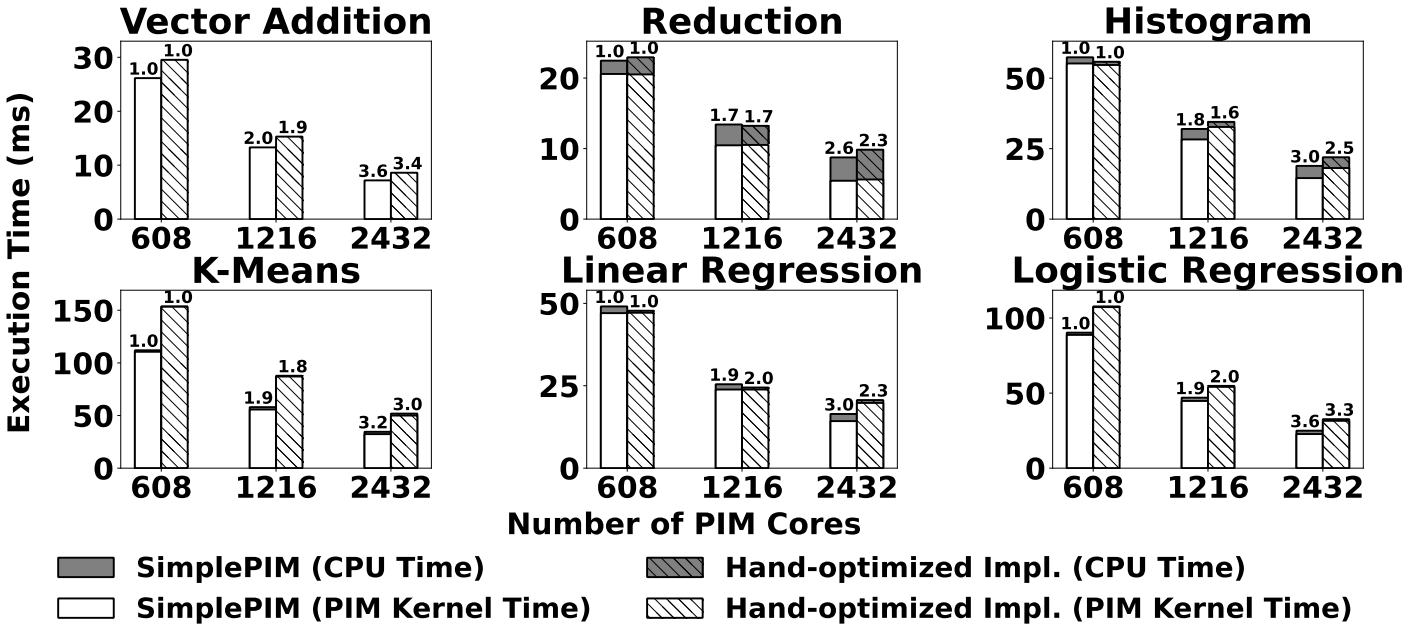


SimplePIM achieves comparable performance for `reduction`, `histogram`, and `linear regression`

SimplePIM outperforms hand-optimized implementations for `vector addition`, `logistic regression`, and `k-means` by 10%-37%

Performance Evaluation (II)

- Strong scaling analysis



SimplePIM scales better than hand-optimized implementations for `reduction`, `histogram`, and `linear regression`

SimplePIM outperforms hand-optimized implementations for `vector addition`, `logistic regression`, and `k-means` by 15%-43%

Discussion

- SimplePIM is devised for PIM architectures with
  - A host processor with access to standard main memory and PIM-enabled memory
  - PIM processing elements (PEs) that communicate via the host processor
  - The number of PIM PEs scales with memory capacity
- SimplePIM emulates the communication between PIM cores via the host processor
- Other parallel patterns can be incorporated in future work
  - Prefix sum and filter can be easily added
  - Stencil and convolution would require fine-grained scatter-gather for halo cells
  - Random access patterns would be hard to support

SimplePIM: arXiv Version

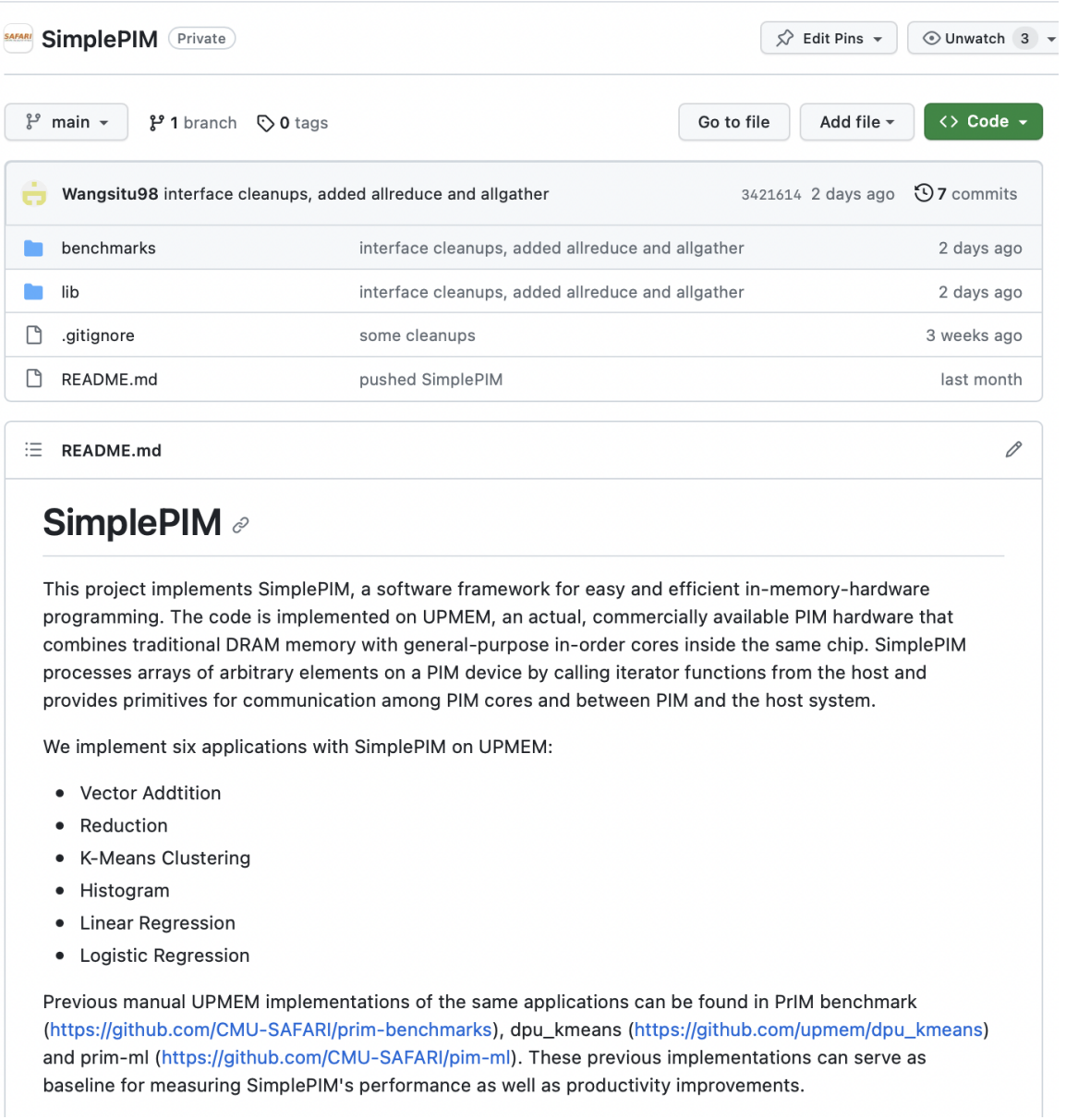
SimplePIM: A Software Framework for Productive and Efficient Processing-in-Memory

Jinfan Chen[1] Juan Gómez-Luna[1] Izzat El Hajj[2] Yuxin Guo[1] Onur Mutlu[1]
[1]ETH Zürich [2]American University of Beirut

https://arxiv.org/pdf/2310.01893.pdf

Source Code

https://github.com/CMU-SAFARI/SimplePIM



SimplePIM Private

Edit Pins | Unwatch 3

main | 1 branch | 0 tags | Go to file | Add file | Code

Wangsitu98 interface cleanups, added allreduce and allgather 3421614 2 days ago 7 commits

benchmarks	interface cleanups, added allreduce and allgather	2 days ago
lib	interface cleanups, added allreduce and allgather	2 days ago
.gitignore	some cleanups	3 weeks ago
README.md	pushed SimplePIM	last month

README.md

SimplePIM

This project implements SimplePIM, a software framework for easy and efficient in-memory-hardware programming. The code is implemented on UPMEM, an actual, commercially available PIM hardware that combines traditional DRAM memory with general-purpose in-order cores inside the same chip. SimplePIM processes arrays of arbitrary elements on a PIM device by calling iterator functions from the host and provides primitives for communication among PIM cores and between PIM and the host system.

We implement six applications with SimplePIM on UPMEM:

- Vector Addtition
- Reduction
- K-Means Clustering
- Histogram
- Linear Regression
- Logistic Regression

Previous manual UPMEM implementations of the same applications can be found in PrIM benchmark (https://github.com/CMU-SAFARI/prim-benchmarks), dpu_kmeans (https://github.com/upmem/dpu_kmeans) and prim-ml (https://github.com/CMU-SAFARI/pim-ml). These previous implementations can serve as baseline for measuring SimplePIM's performance as well as productivity improvements.

SimplePIM: Summary

- Processing-in-Memory (PIM) promises to alleviate the *data movement bottleneck*
- Real PIM hardware is now available, e.g., UPMEM PIM
- However, programming real PIM hardware is challenging, e.g.:
  - Distribute data across PIM memory banks,
  - Manage data transfers between host cores and PIM cores, and between PIM cores,
  - Launch PIM kernels on the PIM cores, etc.
- SimplePIM is a high-level programming framework for real PIM hardware
  - Iterators such as `map`, `reduce`, and `zip`
  - Collective communication with `broadcast`, `scatter`, and `gather`
- Implementation on UPMEM and evaluation with six different workloads
  - Reduction, vector add, histogram, linear/logistic regression, K-means
  - 4.4x fewer lines of code compared to hand-optimized code
  - Between 15% and 43% faster than hand-optimized code for three workloads
- Source code: https://github.com/CMU-SAFARI/SimplePIM

Constable

CONSTABLE

Improving Performance and Power Efficiency by Safely Eliminating Load Instruction Execution

Rahul Bera* Adithya Ranganathan* Joydeep Rakshit Sujit Mahto

Anant V. Nori Jayesh Gaur Ataberk Olgun Konstantinos Kanellopoulos

Mohammad Sadrosadati Sreenivas Subramoney Onur Mutlu

SAFARI Research Group
safari.ethz.ch

ETH zürich

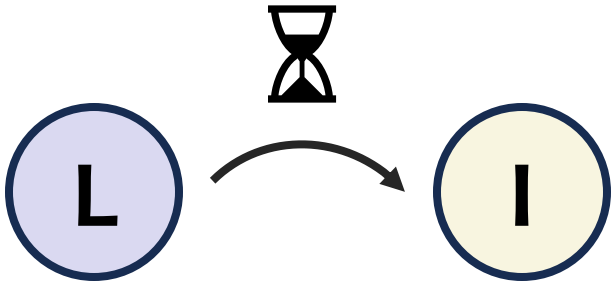
intel

Key Problem

Load instructions are a key limiter of instruction-level parallelism (ILP)

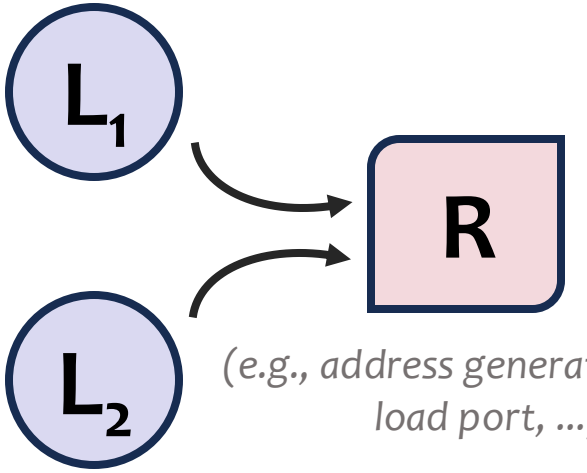
Data Dependence

Stall load-dependent instructions due to **long load execution latency**



Resource Dependence

Stall other loads due to contention **in load execution resources**



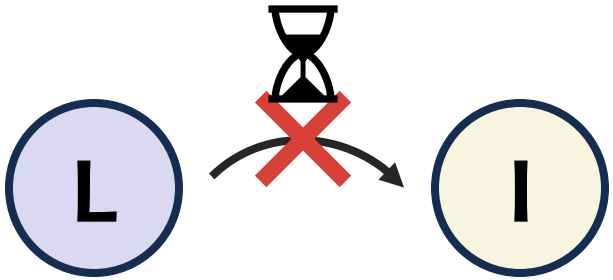
(e.g., address generation unit, load port, ...)

Prior Works on Tolerating Load Latency

- Load value prediction (LVP) [Lipasti+, ASPLOS'96; Sazeides+, MICRO'96; ...]
- Memory renaming (MRN) [Moshovos+, ISCA'97; Tyson+, MICRO'97; ...]

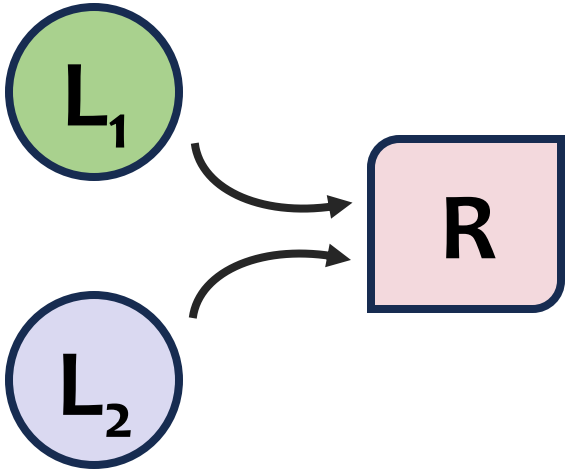
Mitigate Data Dependence

By **speculatively executing** load-dependent instructions using a predicted load value



Do Not Mitigate Resource Dependence

Predicted load still gets executed to verify speculation, consuming execution resources



Motivation

Safely breaking load data dependency
without executing a load instruction
may provide additional performance benefits

How do we start?

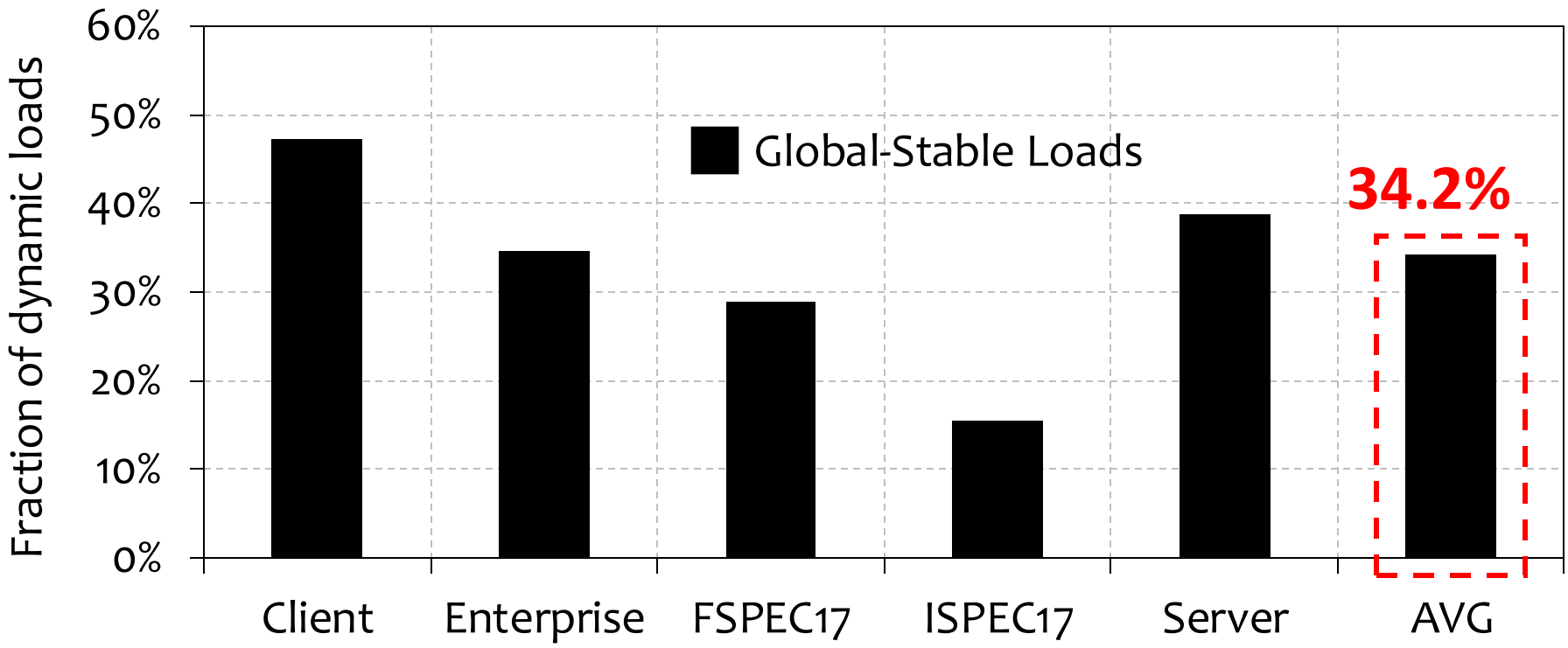
By finding load instructions that repeatedly produce identical results across dynamic instances

Key Finding I: Global-Stable Loads

- Some loads repeatedly fetch **the same data value** from **same load address** across **entire workload**
  - Both operations, **address generation** & **data fetch**, produce identical results across all dynamic instances
  - Prime targets for **breaking data dependency without execution**

Global-Stable Load

Key Finding I: Global-Stable Loads



Nearly **1 in every 3** dynamic loads is a **global-stable load**

Across a wide range of 90 workloads

In the Paper: Analysis of Global-Stable Loads

- **Why do these loads even exist** in well-optimized real-world workloads?
  - Accessing **global-scope variables**
  - Accessing **local variables** of **inline functions**
  - **Limited** set of architectural **registers**

- **Can increasing architectural registers help**?
  - **Very small change** even after doubling x64 registers

- **Deeper characterization** of global-stable loads
  - Which **addressing mode** do they use?
  - How **far away do they appear** in a workload?

In the Paper: Analysis of Global-Stable Loads

- Why do these loads even exist in well optimized
-
-
- How

Constable: Improving Performance and Power Efficiency by Safely Eliminating Load Instruction Execution

*Rahul Bera[1] *Adithya Ranganathan[2] Joydeep Rakshit[2] Sujit Mahto[2]
Anant V. Nori[2] Jayesh Gaur[2] Ataberk Olgun[1] Konstantinos Kanellopoulos[1]
Mohammad Sadrosadati[1] Sreenivas Subramoney[2] Onur Mutlu[1]

[1]ETH Zürich [2]Intel Processor Architecture Research Lab

Load instructions often limit instruction-level parallelism (ILP) in modern processors due to data and resource dependences they cause. Prior techniques like Load Value Prediction (LVP) and Memory Renaming (MRN) mitigate load data dependence by predicting the data value of a load instruction. However, they fail to mitigate load resource dependence as the predicted load instruction gets executed nonetheless (even on a correct prediction), which consumes hard-to-scale pipeline resources that otherwise could have been used to execute other load instructions.

Our goal in this work is to improve ILP by mitigating both *load data dependence and resource dependence. To this end, we propose a purely-microarchitectural technique called* Constable*, that safely eliminates the execution of load instructions. Constable dynamically identifies load instructions that have re-*

stall for multiple cycles, which can limit ILP. On the other hand, a load instruction consumes multiple hard-to-scale hardware resources (e.g., reservation station entry, port to address generation unit, L1-data cache read port) which often causes resource dependence in the pipeline, also limiting ILP.

Prior works propose many latency tolerance techniques to improve ILP by mitigating load data dependence. Load Value Prediction (LVP) [32,42,43,71,98,107,114,139–143,151,153–155, 159,160] and Memory Renaming (MRN) [120,121,147,177,178] are two such widely-studied techniques that mitigate load data dependence via data value speculation. LVP and MRN speculatively execute load-data-dependent instructions using the predicted value of the load instruction, thus improving ILP.

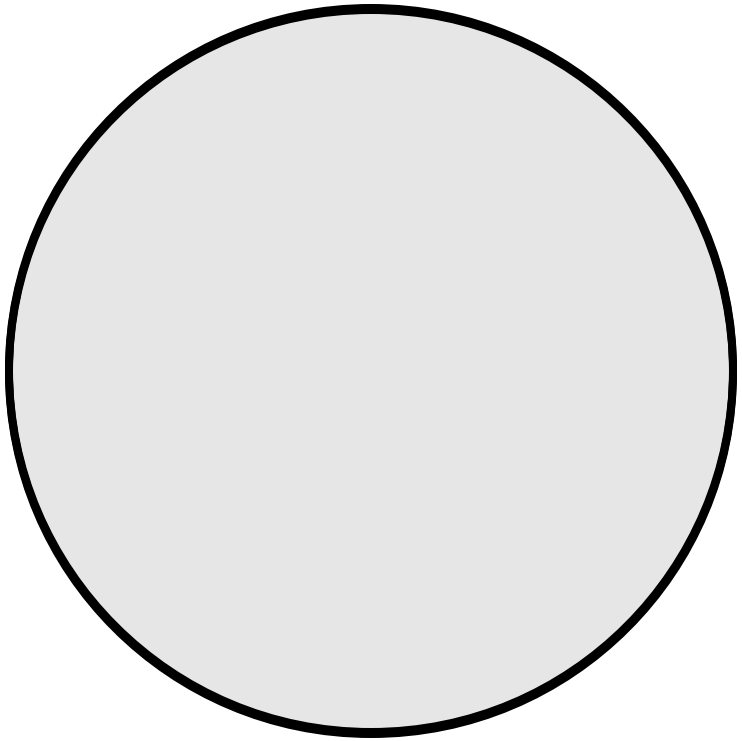
https://arxiv.org/pdf/2406.18786

A significant fraction of loads are **global-stable**

But do they limit ILP even when using load value prediction and memory renaming?

Key Finding II: Global-Stable Loads Cause Resource Dependence

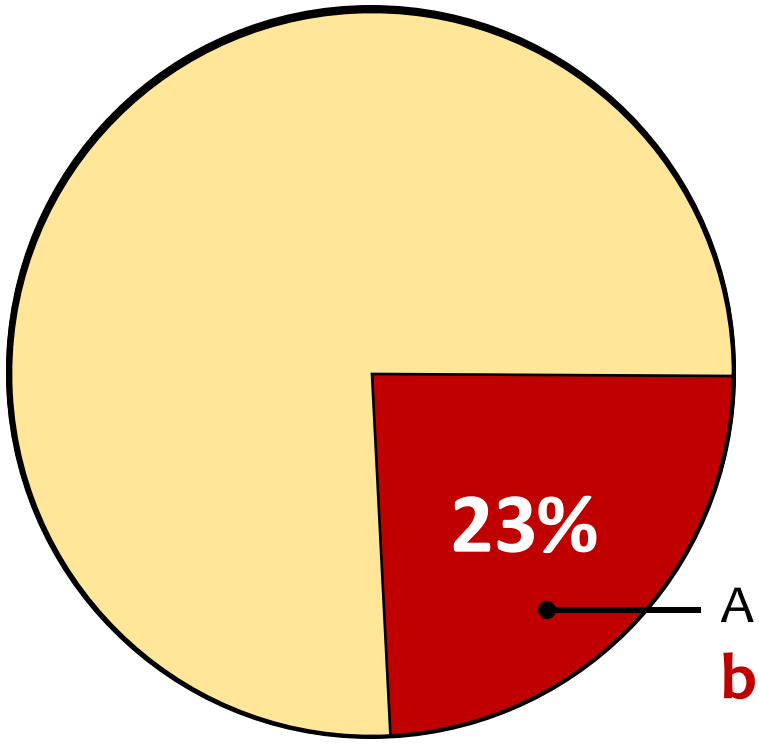
All execution cycles where
at least one load port is utilized



*In an aggressive OoO processor with 6-wide issue, 3 load ports, a **load value predictor** (EVES [Seznec, CVP'18]), and **memory renaming enabled***

Key Finding II: Global-Stable Loads Cause Resource Dependence

All execution cycles where
at least one load port is utilized



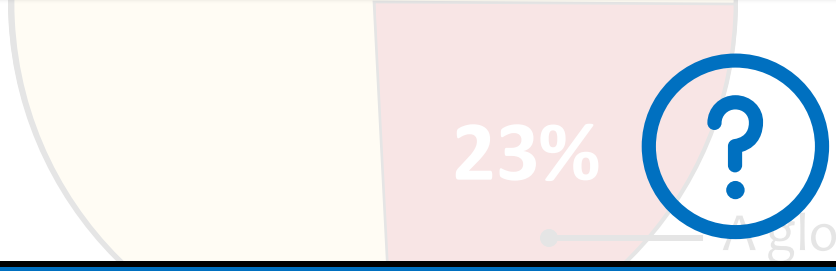
A global-stable load utilizes a load port
blocking a non-global-stable load

*In an aggressive OoO processor with 6-wide issue, 3 load ports, a **load value predictor** (EVES [Seznec, CVP'18]), and **memory renaming enabled***

Key Finding II: Global-Stable Loads Cause Resource Dependence

All execution cycles where
at least one load port is utilized

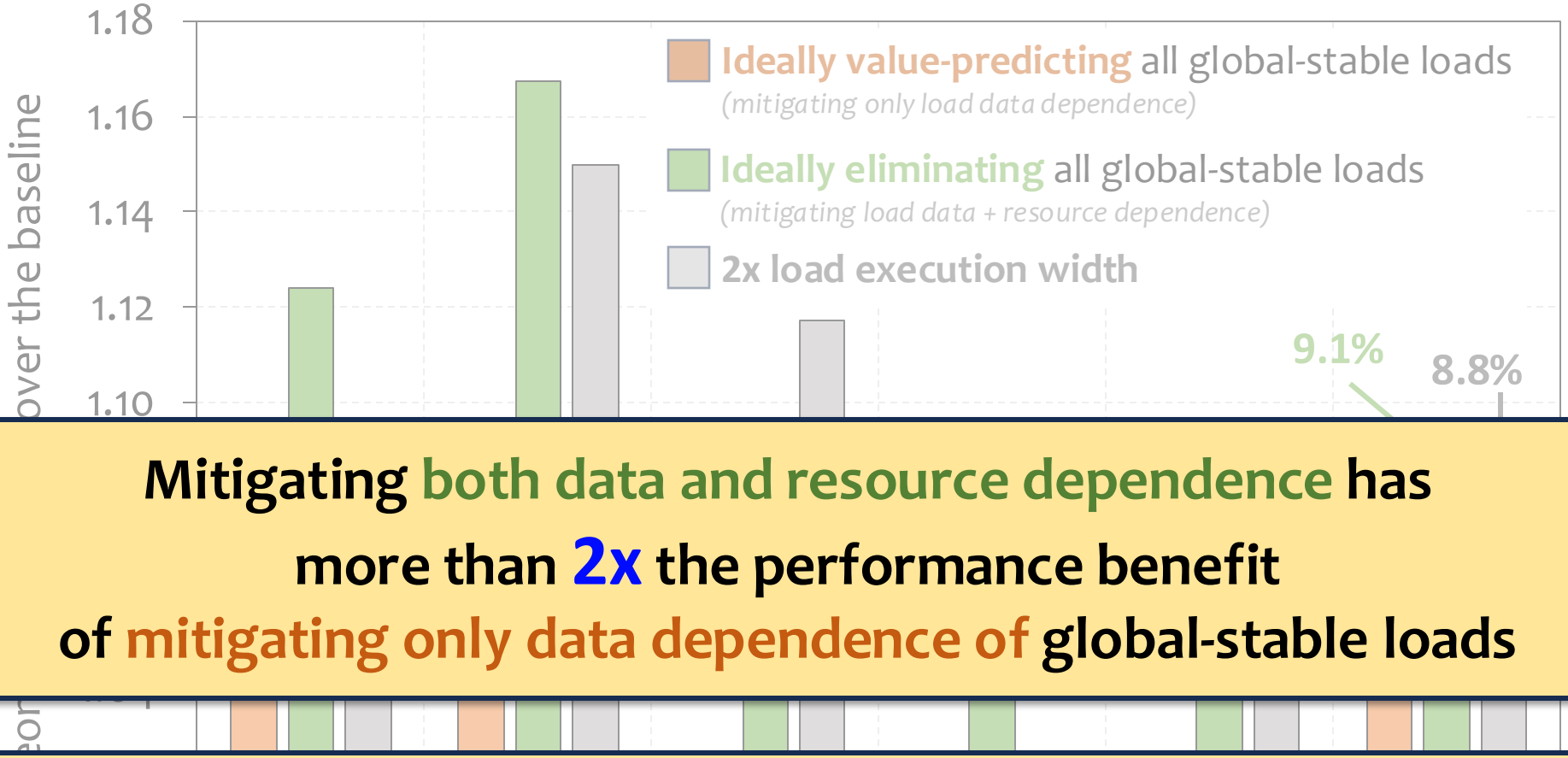
Even when using load value prediction and memory renaming, global-stable loads limit ILP due to resource dependence



What's the performance headroom of mitigating the resource dependence?

a load value predictor (EVES [Seznec, CVP'18]), and memory renaming enabled

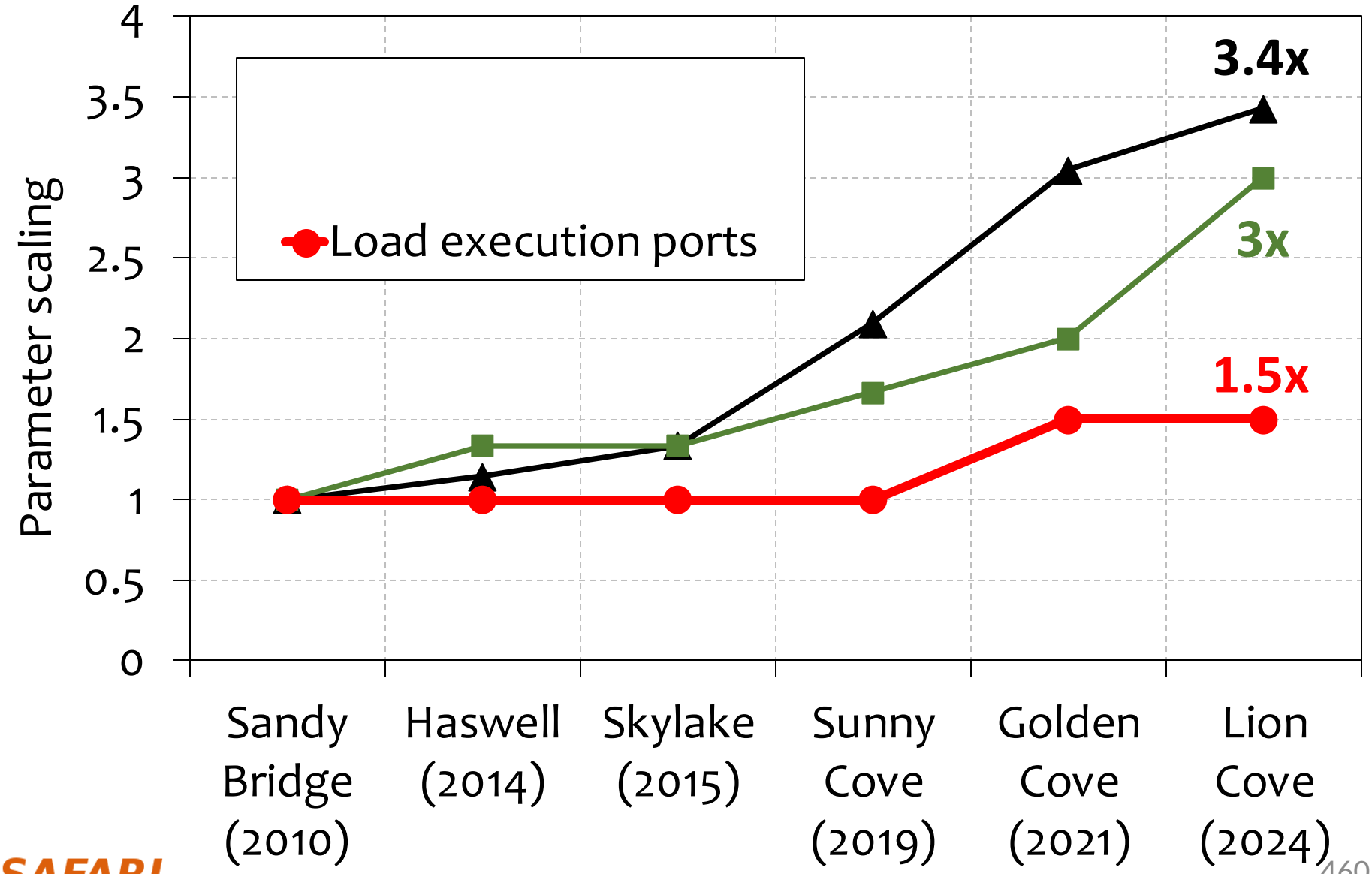
Key Finding III: High Performance Headroom



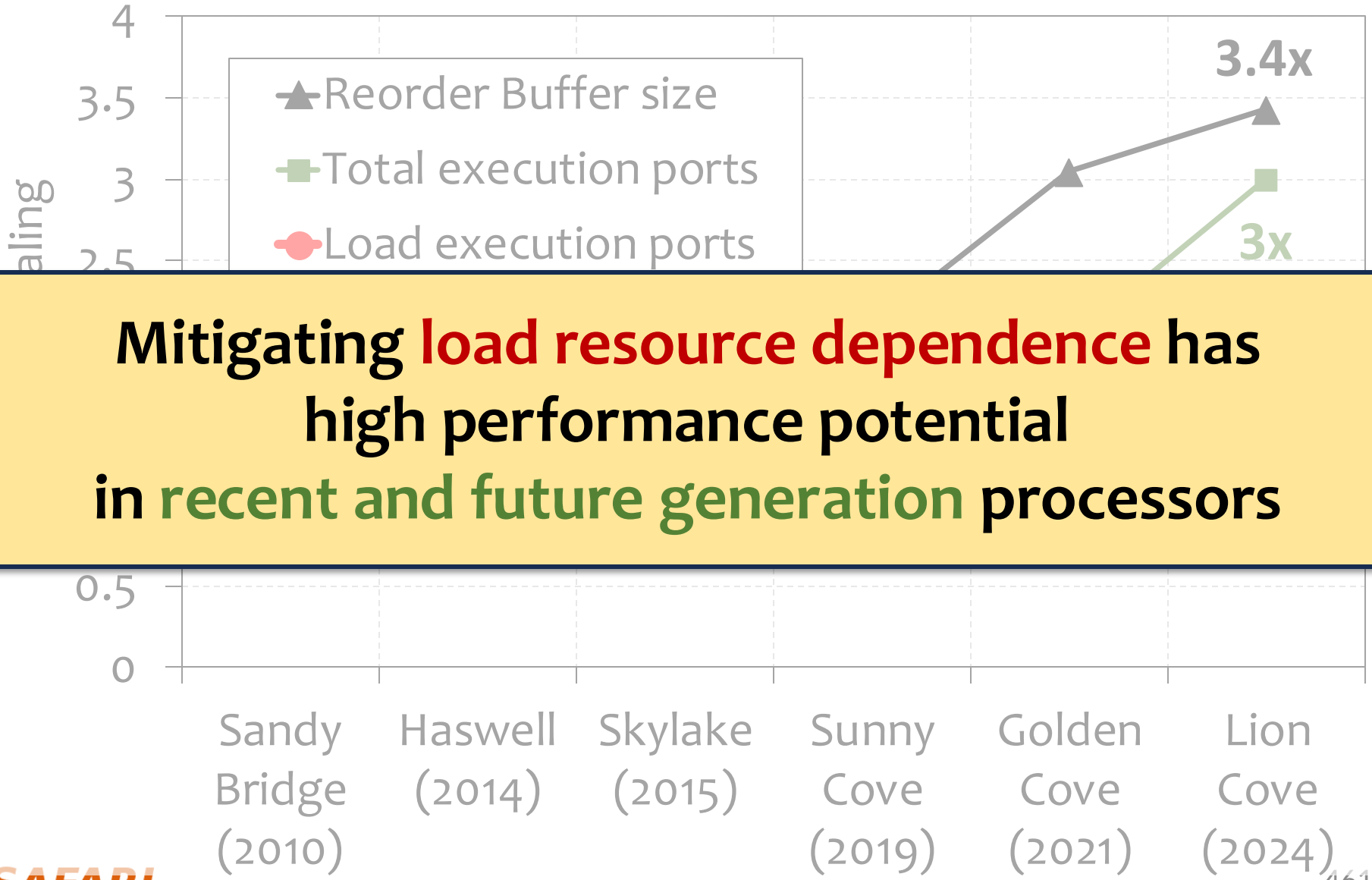
Mitigating both data and resource dependence has more than 2x the performance benefit of mitigating only data dependence of global-stable loads

Ideal elimination of global-stable loads exceeds performance of a processor with 2x wider load execution

Load Execution Resources Lag Behind



Load Execution Resources Lag Behind



Mitigating load resource dependence has high performance potential in recent and future generation processors

Our Goal

To **improve instruction-level parallelism** by mitigating ***both*** **load data dependence** and **resource dependence**

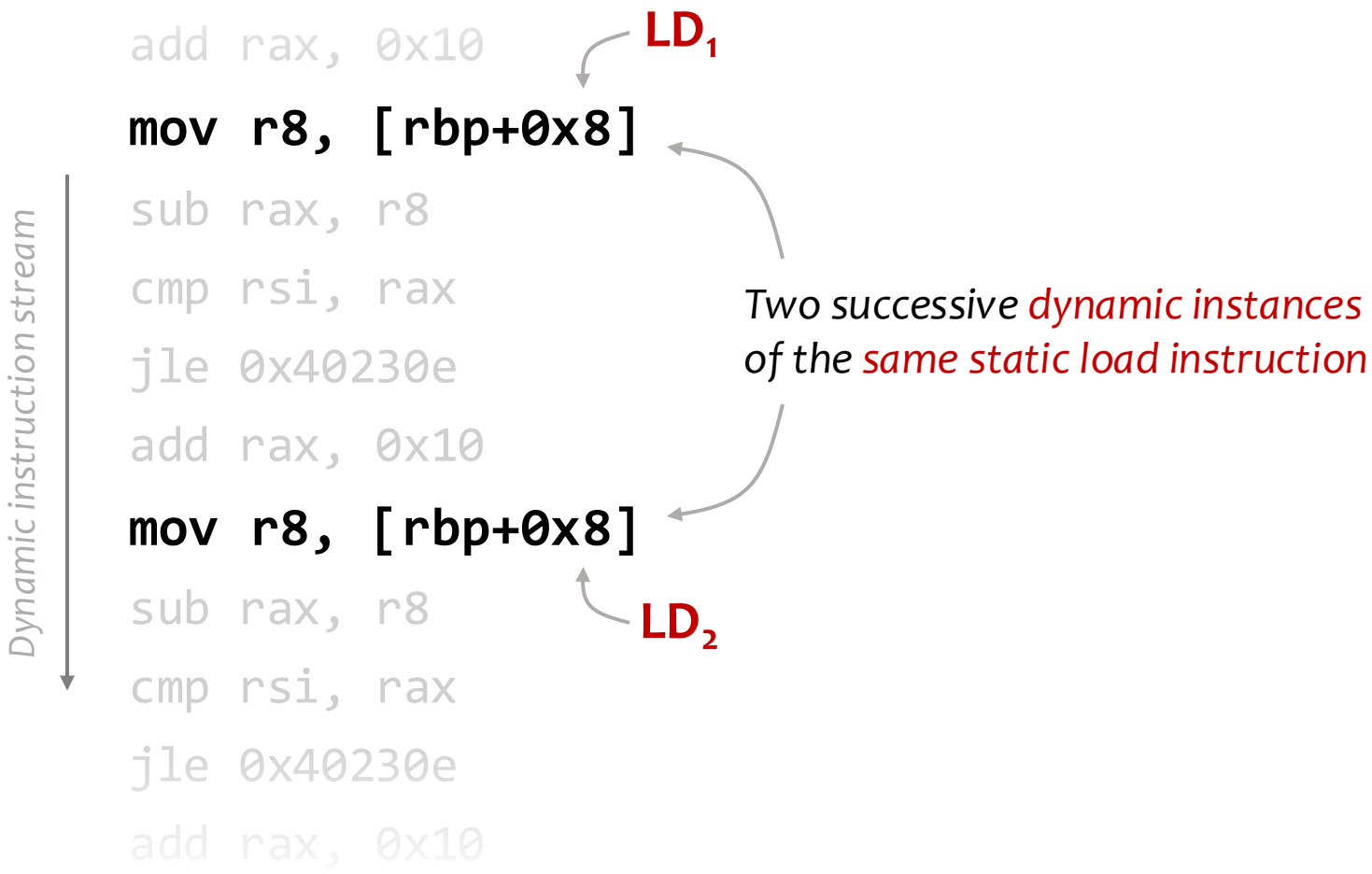
CONSTABLE

A purely-microarchitectural technique

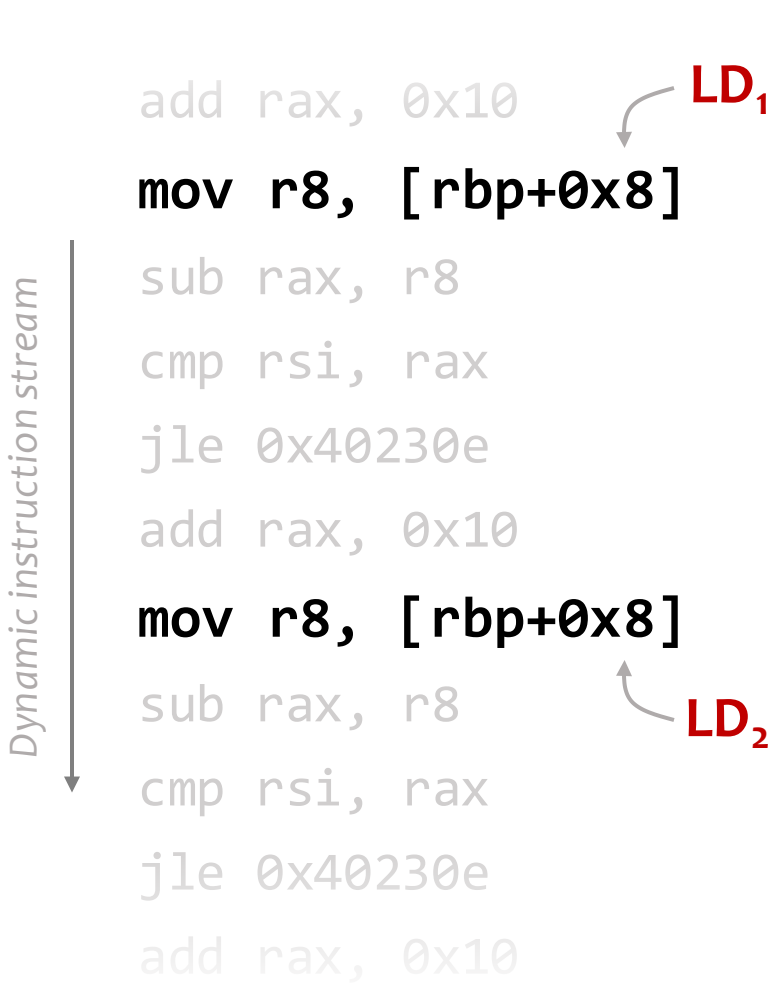
Mitigates both **load data dependence** and **load resource dependence**

By **safely** eliminating the **entire execution** of a load instruction

Constable: Key Insight



Constable: Key Insight



If the source register **rbp** has not been modified

LD_2 would have the same address as LD_1

☑ **Address generation** of LD_2 can be eliminated

If no store or snoop request to address **[rbp+0x8]**

LD_2 would fetch the same data as LD_1

☑ **Data fetching** of LD_2 can be eliminated

Constable: Key Steps

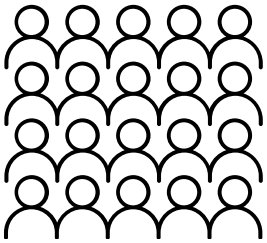
Dynamically identify load instructions
that have historically fetched
the same data from the same load address
(i.e., *likely-stable*)

Eliminate execution of likely-stable loads
by tracking modifications to
their source registers and their load addresses

Prior Related Literature

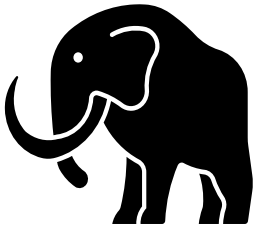
Rich literature on skipping redundant computations
by **memoizing previously-computed results**

[Michie, Nature'68; Harbison+, ASPLOS'82; Richardson, SCA'93; Sodani+, ISCA'97; González+, ICPP'99; ...]



Aim to memoize every instruction

including multiple dynamic instances of each instruction



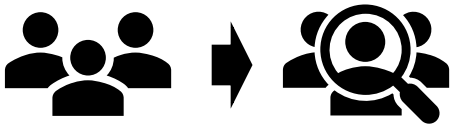
Require large memoization buffer

Often bigger than the size of L1 data cache

Key Improvements over Literature

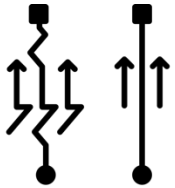
Rich literature on skipping redundant computations
by **memoizing previously-computed results**

[Michie, Nature'68; Harbison+, ASPLOS'82; Richardson, SCA'93; Sodani+, ISCA'97; González+, ICPP'99; ...]



1 **Focus** only on loads that are likely stable

Lower storage overhead
with high load elimination coverage



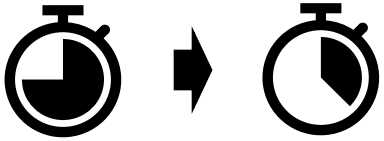
Lower design complexity
Fewer port requirements, lower power

Key Improvements over Literature

Rich literature on skipping redundant computations
by **memoizing previously-computed results**

[Michie, Nature'68; Harbison+, ASPLOS'82; Richardson, SCA'93; Sodani+, ISCA'97; González+, ICPP'99; ...]

1. **Focus** only on loads that are likely stable



2. Eliminate loads **early** in the pipeline

Elimination at rename stage
by explicitly monitoring changes to the source registers
and load address of a likely-stable load

Key Improvements over Literature

Rich literature on skipping redundant computations
by **memoizing previously-computed results**

[Michie, Nature'68; Harbison+, ASPLOS'82; Richardson, SCA'93; Sodani+, ISCA'97; González+, ICPP'99; ...]

1. **Focus** only on loads that are likely stable
2. Eliminate loads **early** in the pipeline
3. Ensure **correctness** in today's processors

- Maintain correctness in presence of **out-of-order load issue**
- Maintain coherence in **multi-threaded & multi-core execution**

Design Overview

Constable: Key Steps

Identify

likely-stable loads

Eliminate

by tracking modifications

Identify a Likely-Stable Load

```
add rax, 0x10
mov r8, [rbp+0x8]      5
sub rax, r8
cmp rsi, rax
add rax, 0x10
mov r8, [rbp+0x8]      6
sub rax, r8
cmp rsi, rax
ret
mov r8, [rbp+0x8]      3
sub rax, r8
cmp rsi, rax
jle 0x40230e
```

- Using a **stability confidence** counter per load instruction

Same data & address as last dynamic instance

+1

Stability Confidence

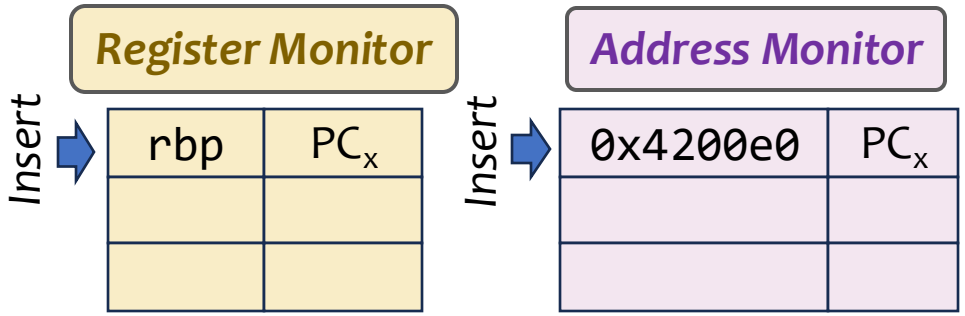
/2

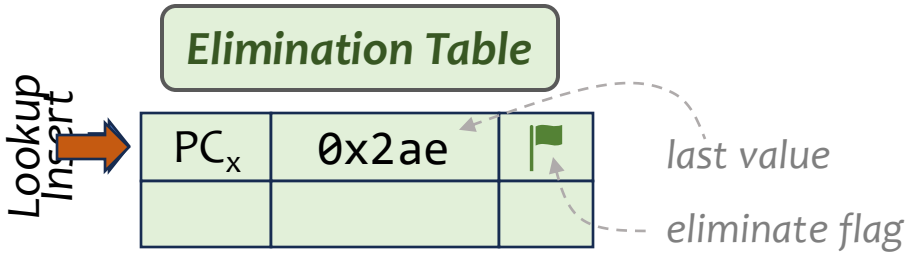
Different data or different address

Eliminate a Likely-Stable Load


```
jle 0x40230e
add rax, 0x10
mov r8, [rbp+0x8]      30
sub rax, r8
cmp rsi, rax
add rax, 0x10
mov r8, [rbp+0x8]      30
sub rax, r8
cmp rsi, rax
add rax, 0x10
mov r8, [rbp+0x8]      30
sub rax, r8
cmp rsi, rax
```

Stability confidence crosses threshold

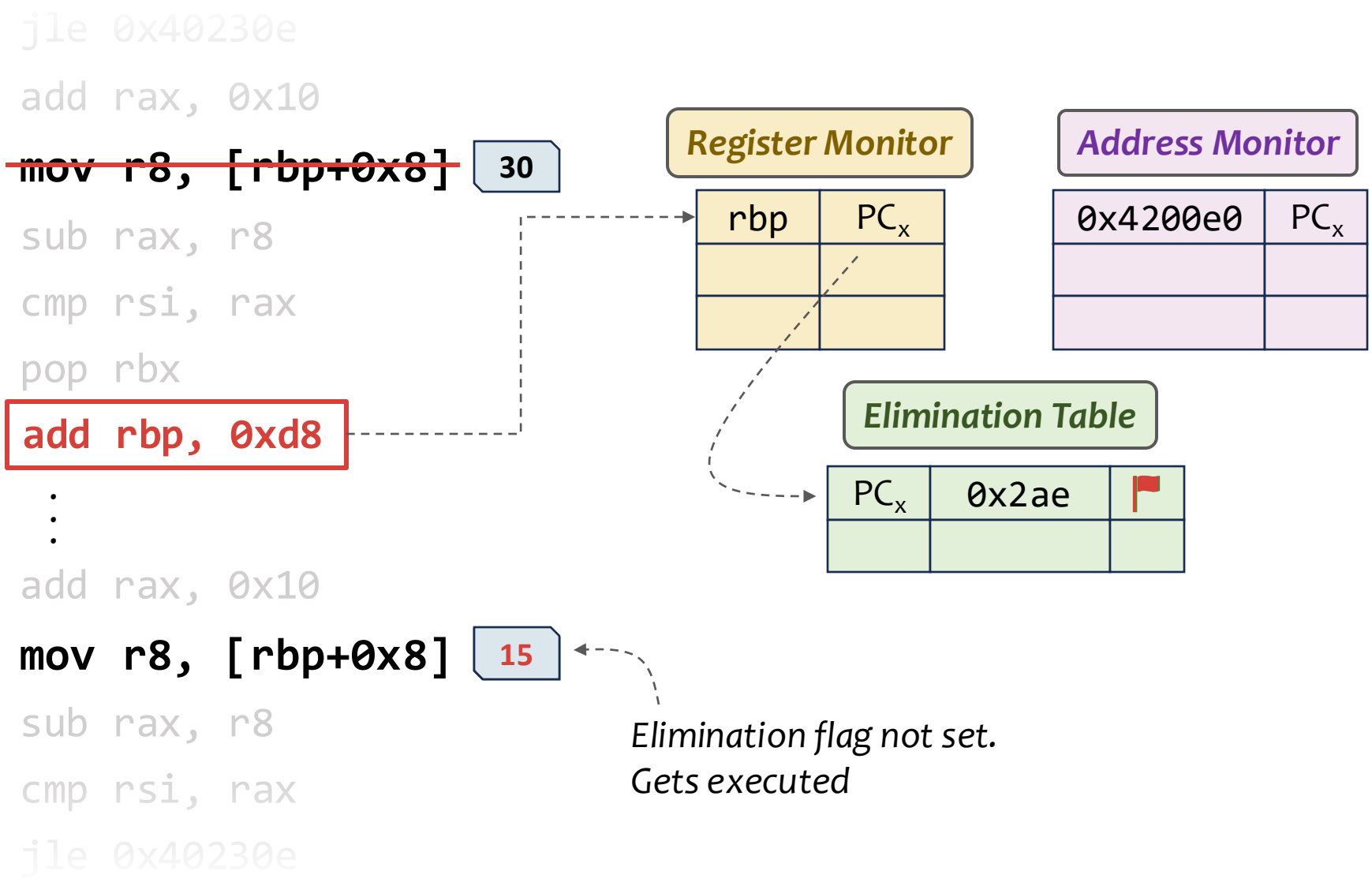




- No reservation station
- No address generation unit
- No load port
- Still takes ROB and load buffer

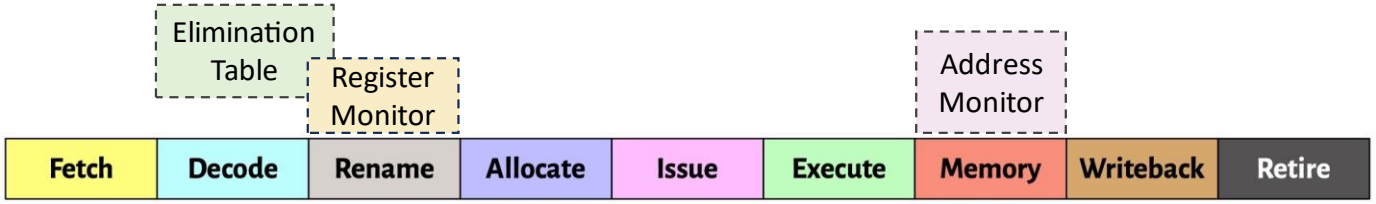
to handle correct elimination of in-flight loads

Stop Elimination of a Likely-Stable Load



More in the Paper

- **Ensuring safe and correct elimination** in presence of
  - Out-of-order load issue
  - Multi-threaded & multi-core execution
  - Wrong-path execution
- **Integration of Constable** into the processor pipeline



- **Microarchitecture for breaking data** dependence on the eliminated loads
- **Microarchitecture of Constable's own structures**
  - Read and write port requirements

More in the Paper

- Ensuring safe and correct elimination in presence of
  - Out-of-order load issue
- In
- M
- Microarchitecture of Constable's own structures
  - Rea

Constable: Improving Performance and Power Efficiency by Safely Eliminating Load Instruction Execution

*Rahul Bera[1] *Adithya Ranganathan[2] Joydeep Rakshit[2] Sujit Mahto[2]
Anant V. Nori[2] Jayesh Gaur[2] Ataberk Olgun[1] Konstantinos Kanellopoulos[1]
Mohammad Sadrosadati[1] Sreenivas Subramoney[2] Onur Mutlu[1]

[1]ETH Zürich [2]Intel Processor Architecture Research Lab

Load instructions often limit instruction-level parallelism (ILP) in modern processors due to data and resource dependences they cause. Prior techniques like Load Value Prediction (LVP) and Memory Renaming (MRN) mitigate load data dependence by predicting the data value of a load instruction. However, they fail to mitigate load resource dependence as the predicted load instruction gets executed nonetheless (even on a correct prediction), which consumes hard-to-scale pipeline resources that otherwise could have been used to execute other load instructions.

Our goal in this work is to improve ILP by mitigating both *load data dependence and resource dependence. To this end, we propose a purely-microarchitectural technique called* Constable, *that safely eliminates the execution of load instructions. Constable dynamically identifies load instructions that have re-*

stall for multiple cycles, which can limit ILP. On the other hand, a load instruction consumes multiple hard-to-scale hardware resources (e.g., reservation station entry, port to address generation unit, L1-data cache read port) which often causes resource dependence in the pipeline, also limiting ILP.

Prior works propose many latency tolerance techniques to improve ILP by mitigating load data dependence. Load Value Prediction (LVP) [32,42,43,71,98,107,114,139–143,151,153–155, 159,160] and Memory Renaming (MRN) [120,121,147,177,178] are two such widely-studied techniques that mitigate load data dependence via data value speculation. LVP and MRN speculatively execute load-data-dependent instructions using the predicted value of the load instruction, thus improving ILP.

https://arxiv.org/pdf/2406.18786

Evaluation

Methodology

- Industry-grade x86-64 simulator modeling aggressive OoO processor
  - 8-wide fetch, 6-wide issue to 3 load ports, 512-entry ROB
  - With memory renaming, zero/constant/move elimination, branch folding
  - Five prefetchers throughout cache hierarchy

- **90 workloads** of wide variety
  - All from **SPEC CPU** 2017
  - **Client** (SYSMark, DaCapo, ...)
  - **Enterprise** (SPECjbb, SPECjEnterprise, ...)
  - **Server** (BigBench, Hadoop, ...)

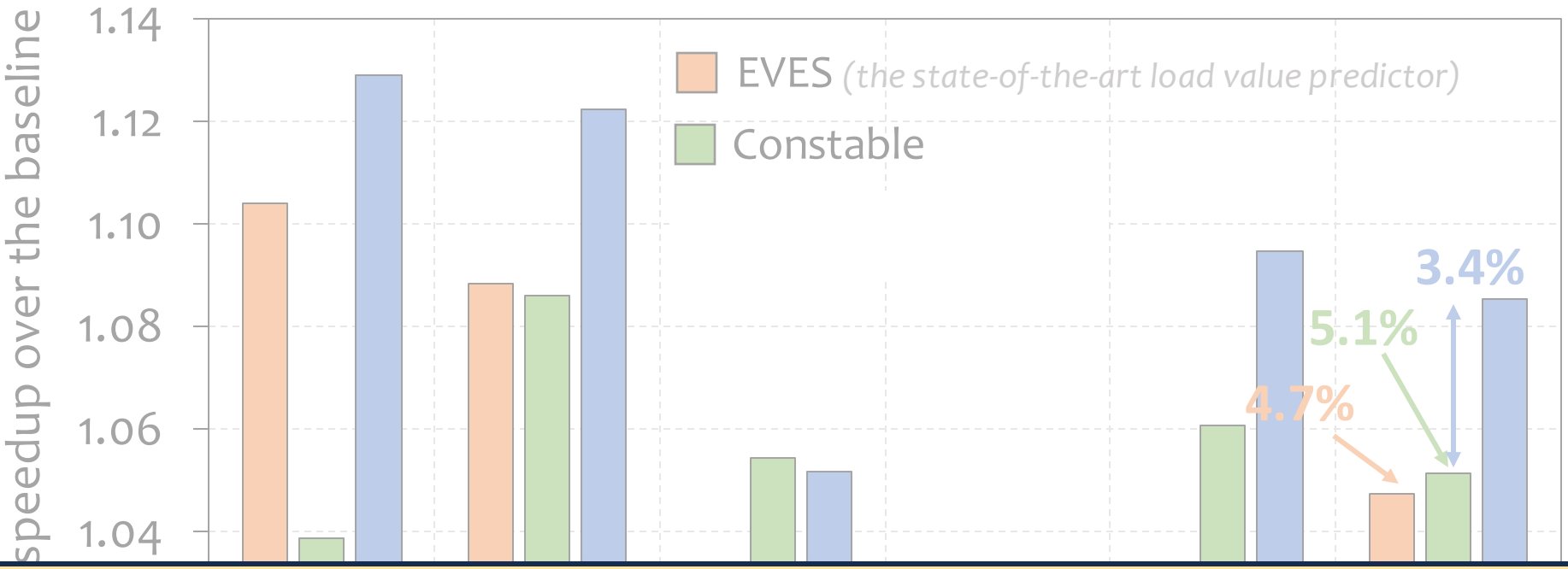
Mechanisms compared against

- EVES, the state-of-the-art load value predictor [Seznec, CVP'18]
- Early Load Address Resolution [Bekerman+, ISCA'00]
- Register File Prefetching [Shukla+, ISCA'22]

Configurations

- No simultaneous multi-threading (SMT)
- 2-way SMT

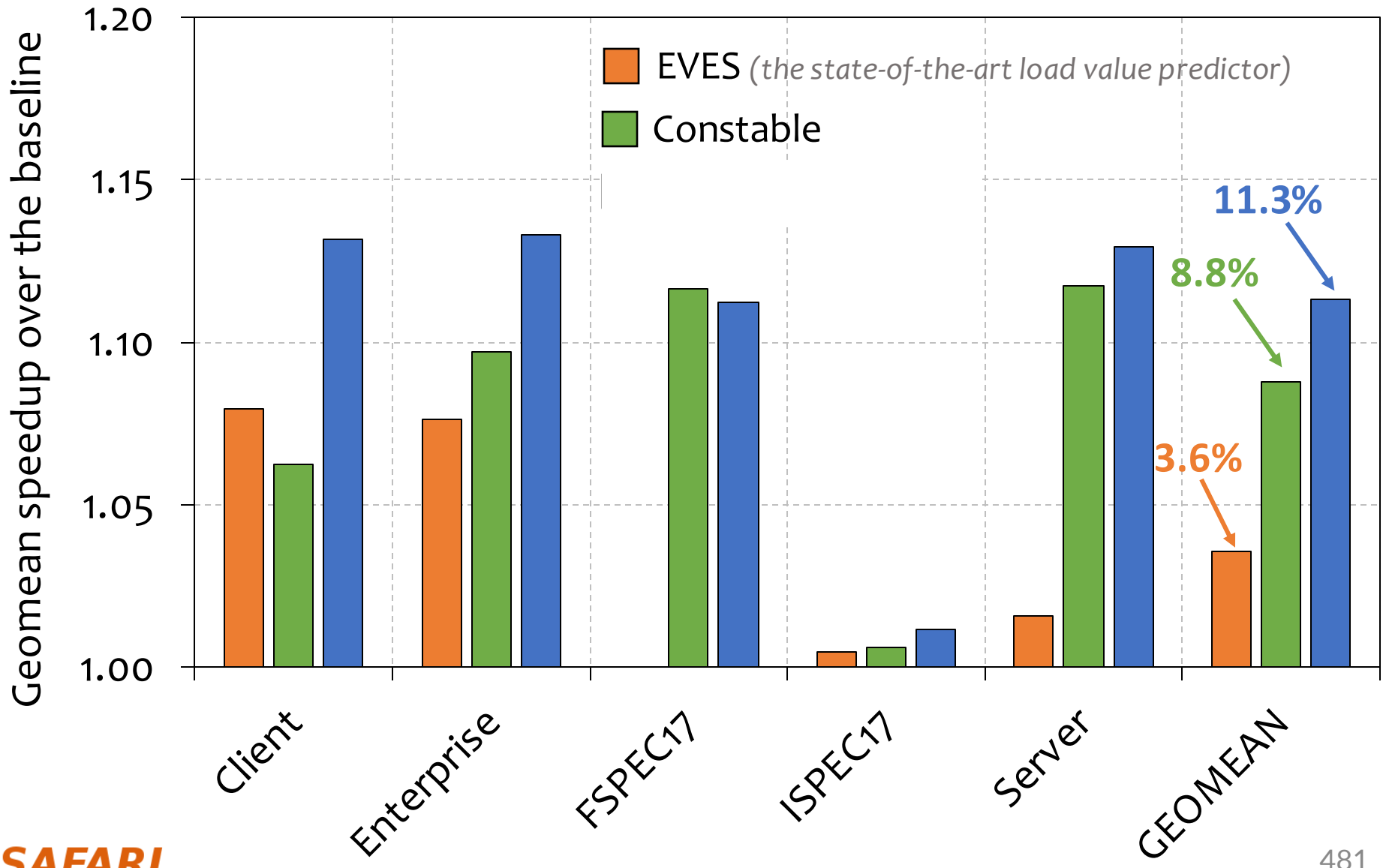
Performance Improvement in noSMT



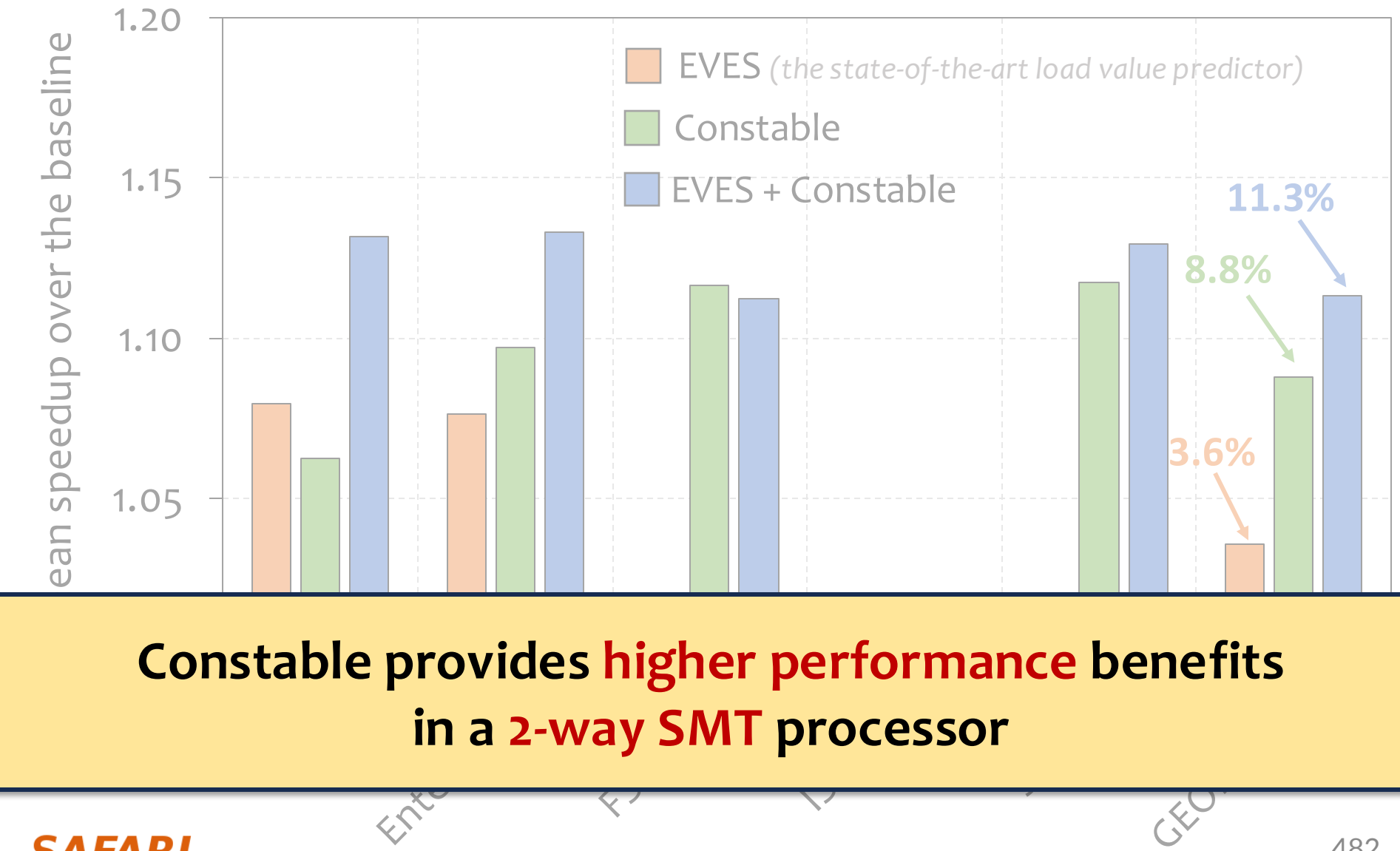
Constable alone provides similar performance as EVES with only ½ of EVES' storage overhead

Constable on top of EVES outperforms EVES alone

Performance Improvement in 2-way SMT

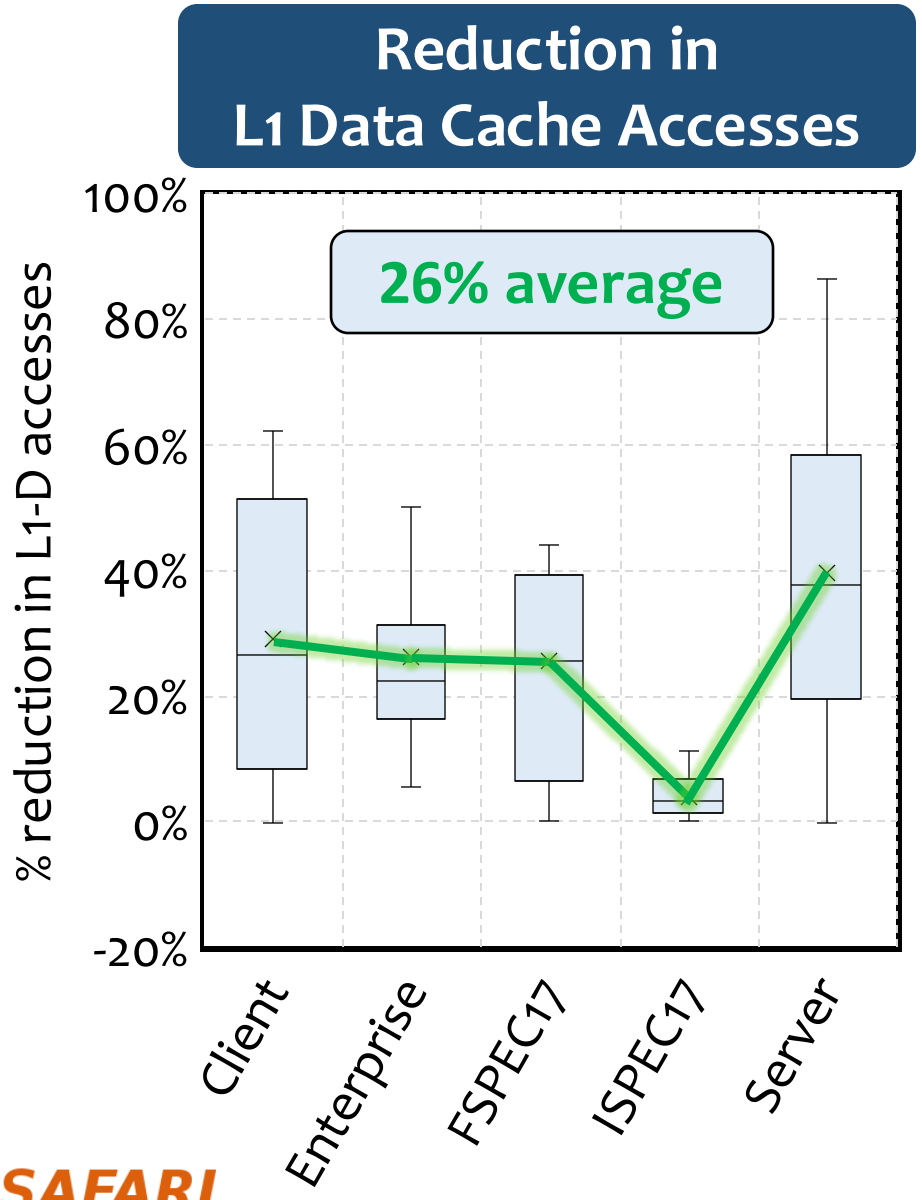


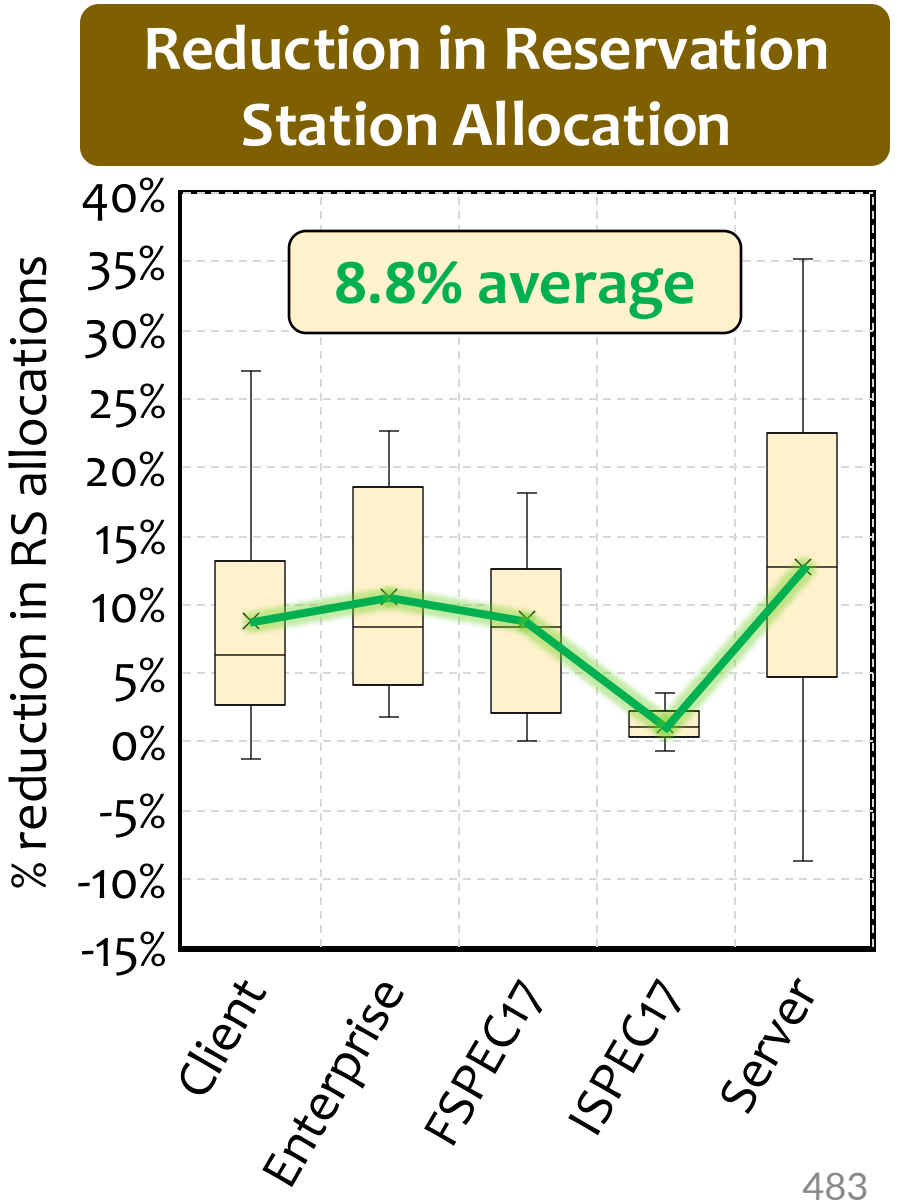
Performance Improvement in 2-way SMT



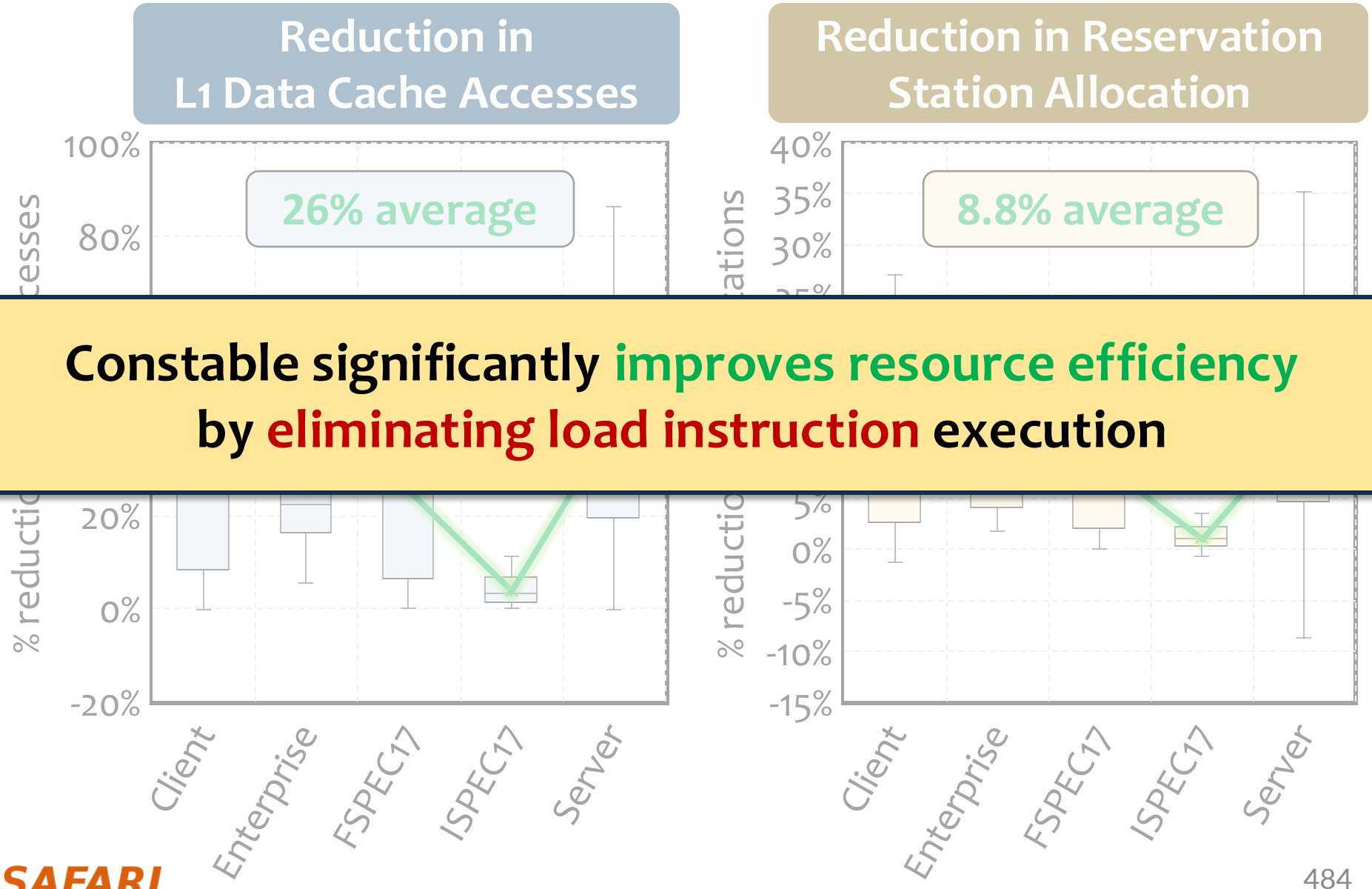
Constable provides higher performance benefits in a 2-way SMT processor

Improvement in Resource Efficiency





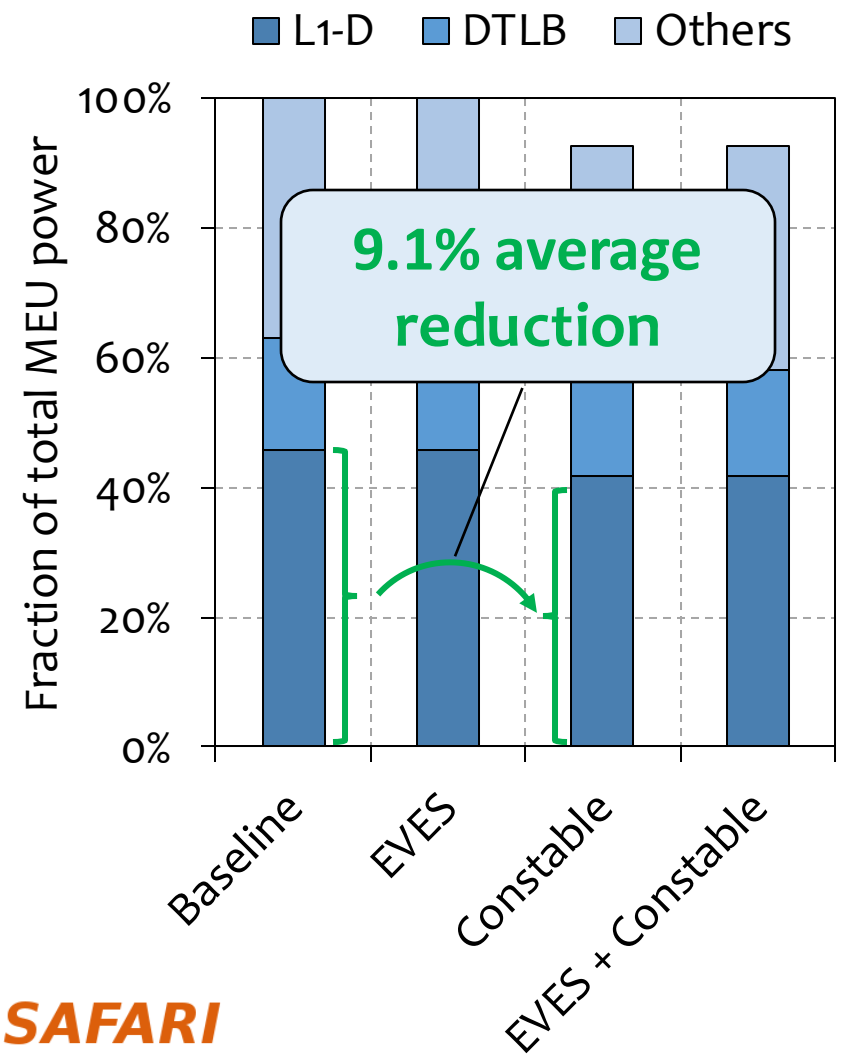
Improvement in Resource Efficiency



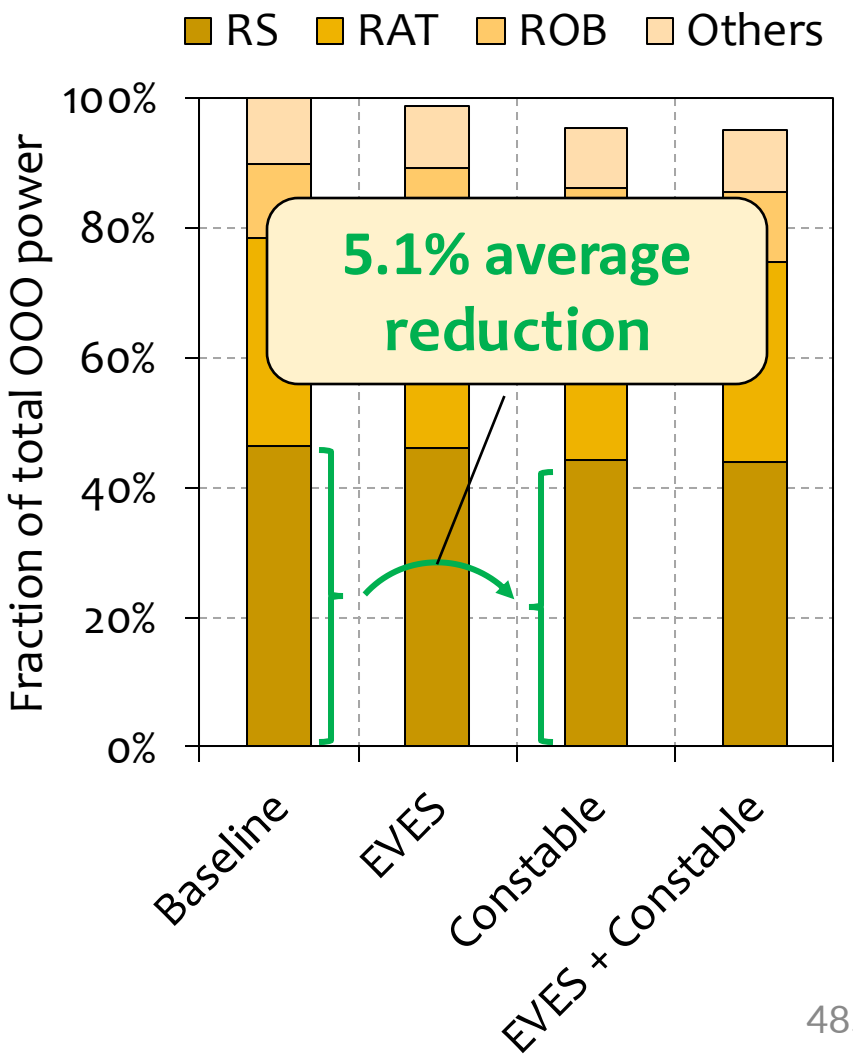
Constable significantly improves resource efficiency by eliminating load instruction execution

Reduction in Dynamic Power

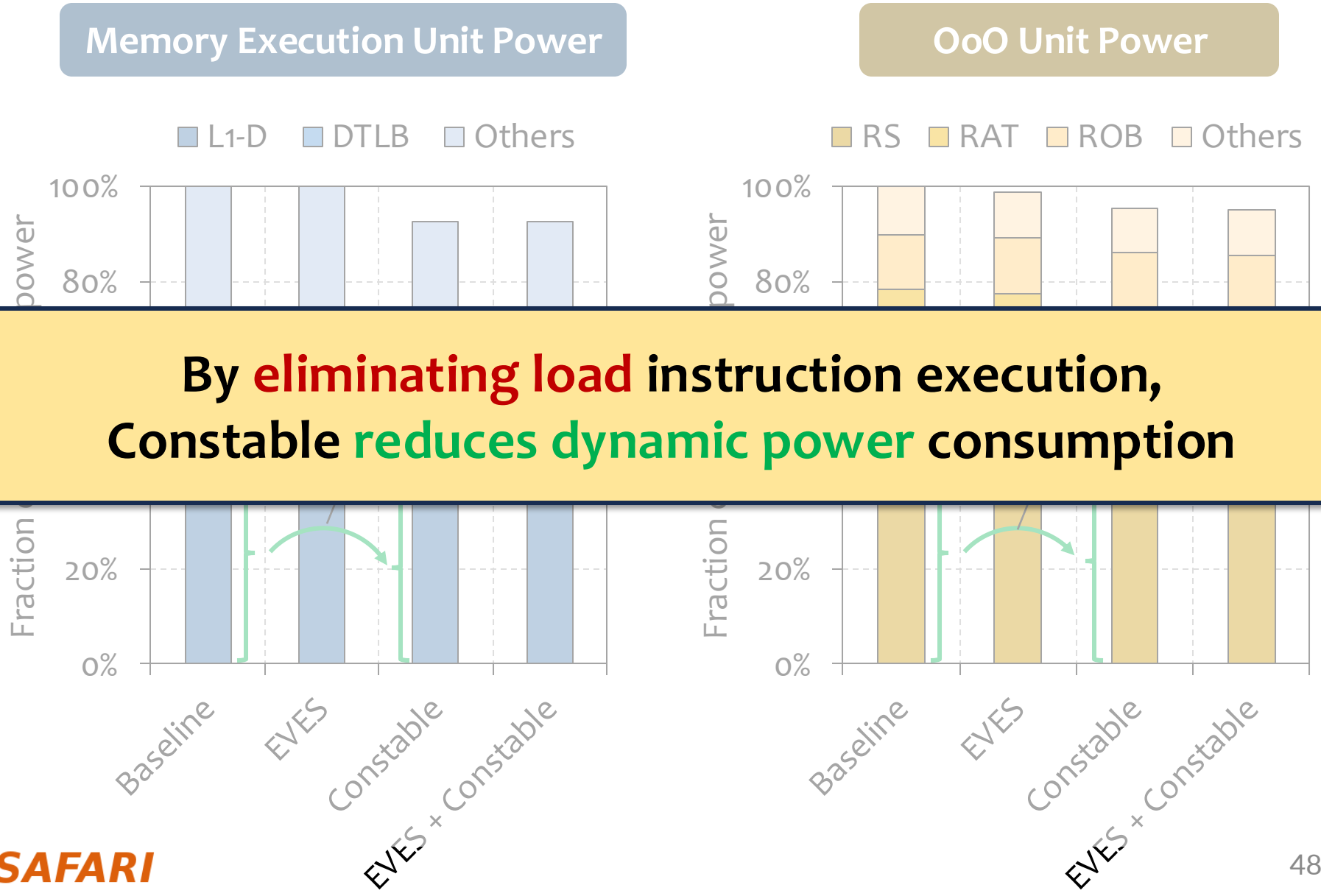
Memory Execution Unit Power



OoO Unit Power

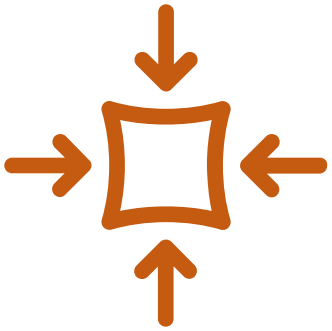


Reduction in Dynamic Power



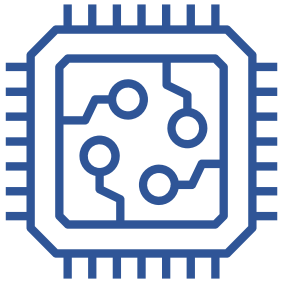
By eliminating load instruction execution, Constable reduces dynamic power consumption

Area and Power Overhead of Constable's Own Structures



12.4 KB

Storage overhead per core



0.232 mm^2

0.0061% area of Intel Alderlake-S processor

Low Energy

Up to 10.8 pJ/read and 16.7 pJ/write

More in the Paper

- **Load elimination coverage** of Constable
  - **23.5%** of all dynamic loads are eliminated

- **Per-workload performance** analysis
  - Up to **31.2%** over baseline
  - 60/90 workloads outperforms EVES by more than 5%

- **Performance contribution per load category**
  - Stack loads contribute the highest

- Performance improvement over **prior works**
  - **4.7%** over **Early load address resolution**
  - **3.6%** over **Register file prefetching**

- Performance **sensitivity**:
  - Higher performance in every configuration up to **2X load execution width**
  - Higher performance in every configuration up to **2X pipeline depth**

More in the Paper

- Load elimination coverage of Constable
  - 23.5% of all dynamic loads are eliminated
- Pe
- Pe
- Pe
- Pe
  - Higher performance in every configuration up to 2X load execution width
  - High

Constable: Improving Performance and Power Efficiency by Safely Eliminating Load Instruction Execution

*Rahul Bera[1] *Adithya Ranganathan[2] Joydeep Rakshit[2] Sujit Mahto[2]
Anant V. Nori[2] Jayesh Gaur[2] Ataberk Olgun[1] Konstantinos Kanellopoulos[1]
Mohammad Sadrosadati[1] Sreenivas Subramoney[2] Onur Mutlu[1]

[1]ETH Zürich [2]Intel Processor Architecture Research Lab

Load instructions often limit instruction-level parallelism (ILP) in modern processors due to data and resource dependences they cause. Prior techniques like Load Value Prediction (LVP) and Memory Renaming (MRN) mitigate load data dependence by predicting the data value of a load instruction. However, they fail to mitigate load resource dependence as the predicted load instruction gets executed nonetheless (even on a correct prediction), which consumes hard-to-scale pipeline resources that otherwise could have been used to execute other load instructions.

Our goal in this work is to improve ILP by mitigating both *load data dependence and resource dependence. To this end, we propose a purely-microarchitectural technique called* Constable, *that safely eliminates the execution of load instructions. Constable dynamically identifies load instructions that have re-*

stall for multiple cycles, which can limit ILP. On the other hand, a load instruction consumes multiple hard-to-scale hardware resources (e.g., reservation station entry, port to address generation unit, L1-data cache read port) which often causes resource dependence in the pipeline, also limiting ILP.

Prior works propose many latency tolerance techniques to improve ILP by mitigating load data dependence. Load Value Prediction (LVP) [32,42,43,71,98,107,114,139–143,151,153–155, 159,160] and Memory Renaming (MRN) [120,121,147,177,178] are two such widely-studied techniques that mitigate load data dependence via data value speculation. LVP and MRN speculatively execute load-data-dependent instructions using the predicted value of the load instruction, thus improving ILP.

https://arxiv.org/pdf/2406.18786

To Summarize...

Our Key Findings

1. A large fraction (**34%**) of dynamic loads fetch the same data from the same address throughout the entire workload

2. These global-stable loads cause significant ILP loss due to **resource dependence**

3. **Eliminating** global-stable load execution provides more than **2x the performance** benefit of just **breaking** their **load data dependency**

Our Proposal

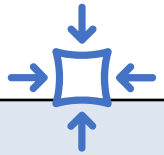
Constable

Identifies and eliminates loads
that repeatedly fetch same data from same address

High performance benefit
over a strong baseline system
without (**5.1%**) and with SMT (**8.8%**)

Improves resource efficiency
L1-D access reduction by **26%**
RS allocation reduction by **8.8%**

Reduces dynamic power
L1-D power by **9.1%**
RS power by **5.1%**



Low storage overhead
Only **12.4 KB**/core,
0.232 mm^2 in 14-nm technology

There's Still Headroom...

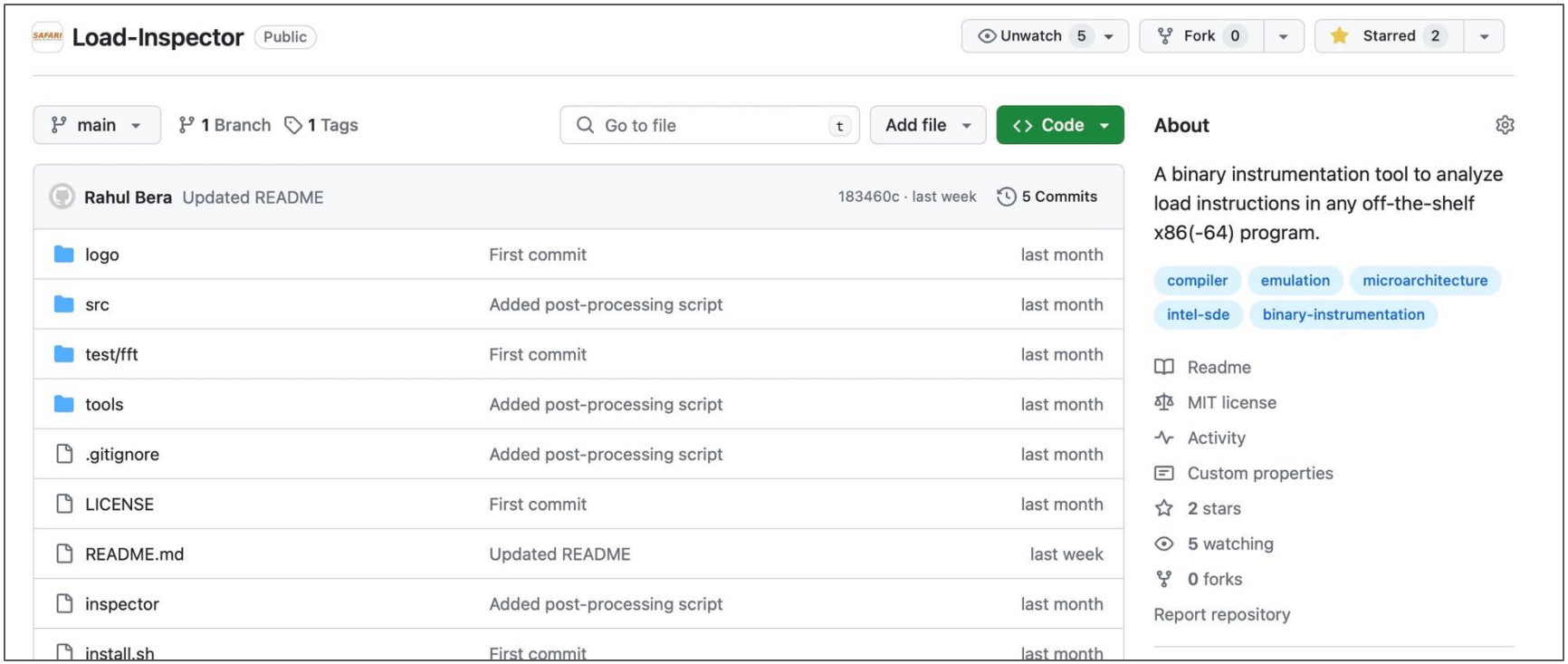
Constable successfully eliminates
57% of all global-stable loads at runtime

43% of global-stable loads
do not get eliminated

We need to understand more
software primitives that generate global-stable loads

Open-Source Tool

A tool to analyze load instructions in any off-the-shelf x86(-64) program



https://github.com/CMU-SAFARI/Load-Inspector

Open-Source Tool

A tool to analyze load instructions in any off-the-shelf x86(-64) program

Study global-stable loads

Study the effects of increasing architectural registers using APX extension to x64 ISA

https://github.com/CMU-SAFARI/Load-Inspector

CONSTABLE

Improving Performance and Power Efficiency by Safely Eliminating Load Instruction Execution

arXiv

Load Inspector

SAFARI
SAFARI Research Group
safari.ethz.ch

ETH zürich

intel