

(Intelligent) Architectures for Intelligent Machines

Onur Mutlu

omutlu@gmail.com

<https://people.inf.ethz.ch/omutlu>

15 April 2019

Mubadala-SRC AI Hardware Systems Forum Keynote Talk

SAFARI

ETH zürich

Carnegie Mellon

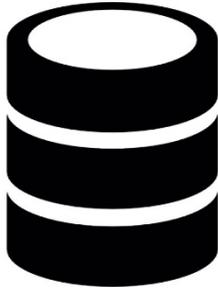
Computing

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

Data is Key for Future Workloads



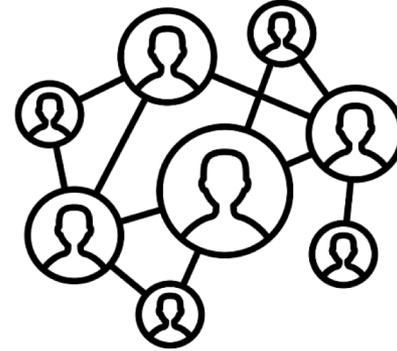
In-memory Databases

[Mao+, EuroSys'12;
Clapp+ (Intel), IISWC'15]



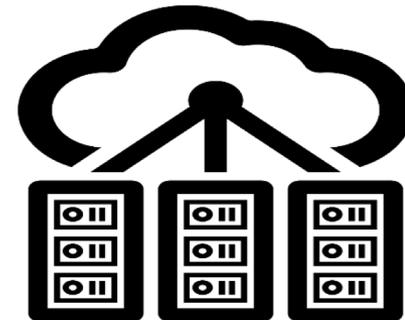
In-Memory Data Analytics

[Clapp+ (Intel), IISWC'15;
Awan+, BDCloud'15]



Graph/Tree Processing

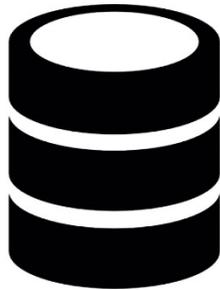
[Xu+, IISWC'12; Umuroglu+, FPL'15]



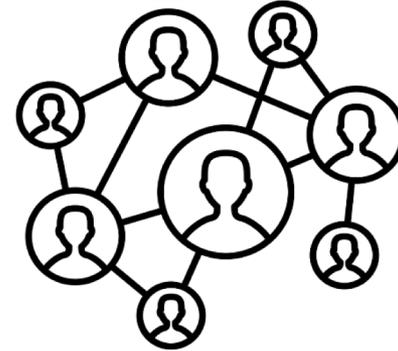
Datacenter Workloads

[Kanev+ (Google), ISCA'15]

Data Overwhelms Modern Machines



In-memory Databases



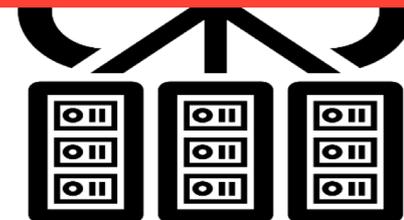
Graph/Tree Processing

Data → performance & energy bottleneck



In-Memory Data Analytics

[Clapp+ (Intel), IISWC'15;
Awan+, BDCloud'15]



Datacenter Workloads

[Kanev+ (Google), ISCA'15]

Data is Key for Future Workloads



Chrome

Google's web browser



TensorFlow Mobile

Google's machine learning framework

VP9



Video Playback

Google's **video codec**

VP9



Video Capture

Google's **video codec**

Data Overwhelms Modern Machines



Chrome



TensorFlow Mobile

Data → performance & energy bottleneck

VP9



Video Playback

Google's **video codec**

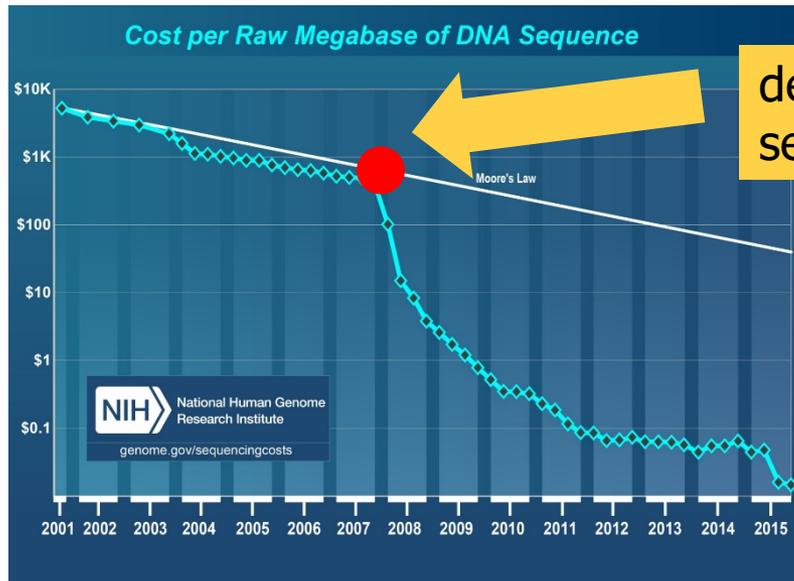
VP9



Video Capture

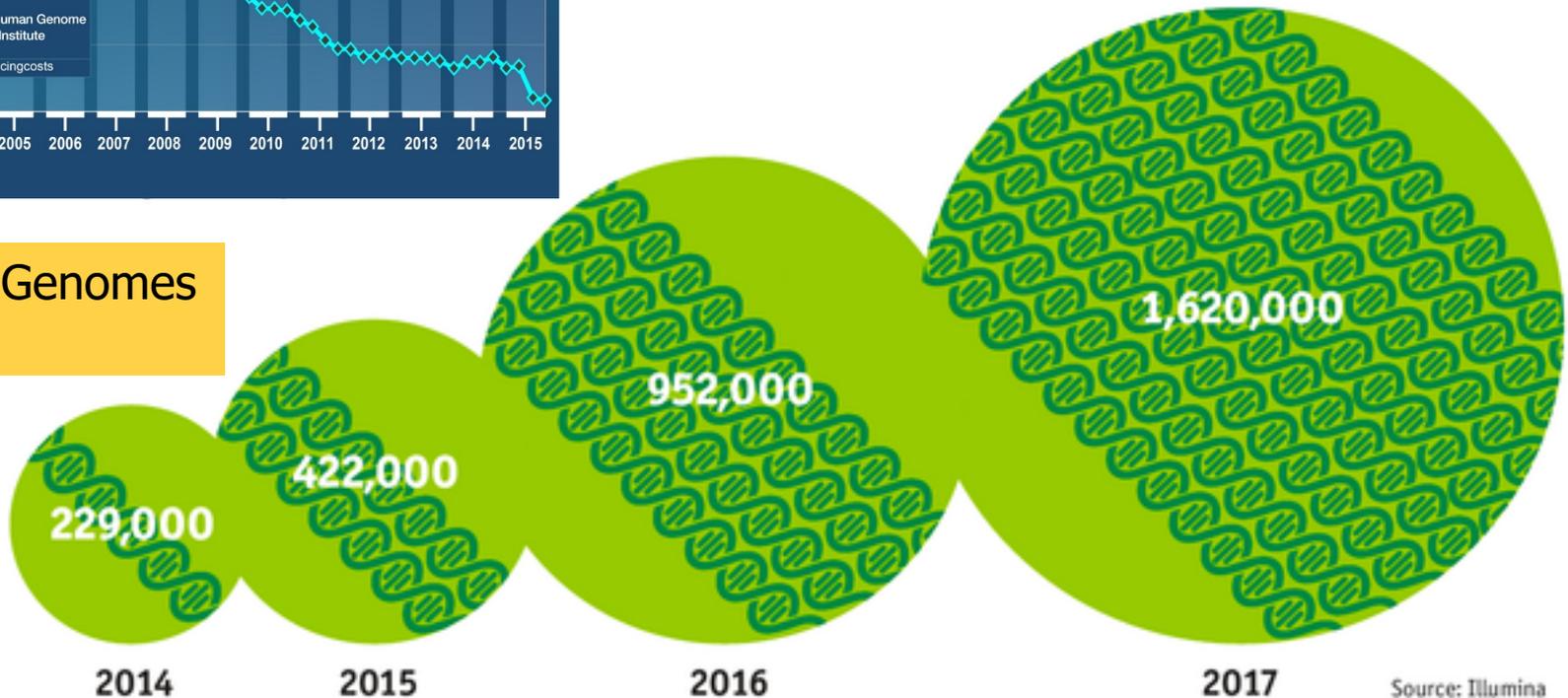
Google's **video codec**

Data is Key for Future Workloads

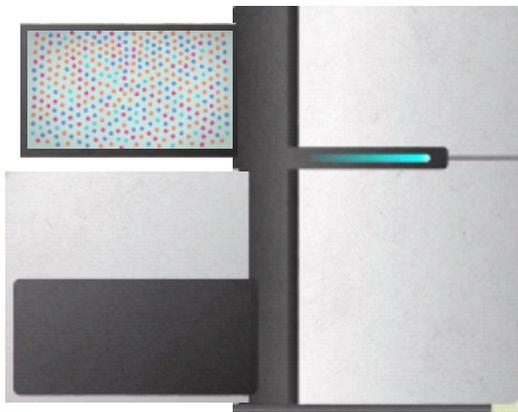


development of high-throughput sequencing (HTS) technologies

Number of Genomes Sequenced

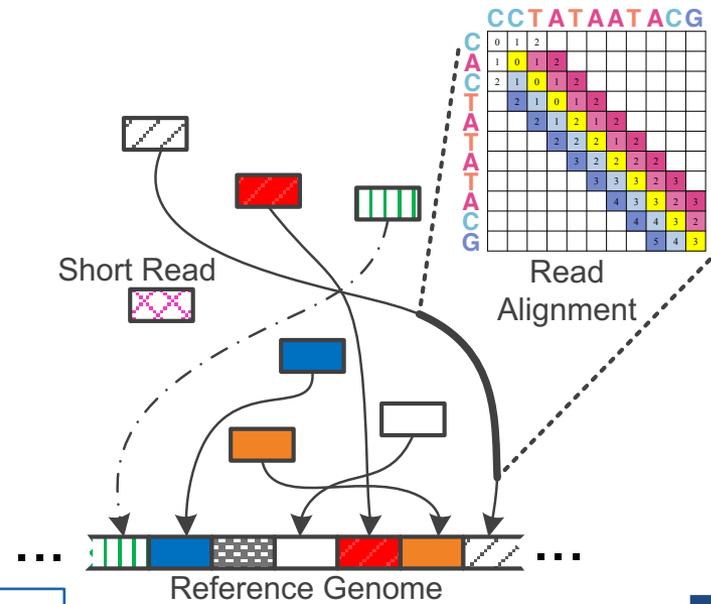


The Economist



Billions of Short Reads

ATATATACGTA
 TTAGTACGTACGT
 ATACGTA
 CG CCCCTACGTA
 CGTACTAGTACGT
 TTAGTACGTACGT
 TACGTA
 TACGTA
 TTTAAACGTA
 CGTACTAGTACGT
 GGGAGTACGTACGT



1 Sequencing

Genome Analysis

Read Mapping 2

Data → performance & energy bottleneck

read4: CGCTTCCAT
 read5: CCATGACGC
 read6: TTCCATGAC



3 Variant Calling

Scientific Discovery 4

New Genome Sequencing Technologies

Nanopore sequencing technology and tools for genome assembly: computational analysis of the current state, bottlenecks and future directions

Damla Senol Cali ✉, Jeremie S Kim, Saugata Ghose, Can Alkan, Onur Mutlu

Briefings in Bioinformatics, bby017, <https://doi.org/10.1093/bib/bby017>

Published: 02 April 2018 **Article history** ▼



Oxford Nanopore MinION

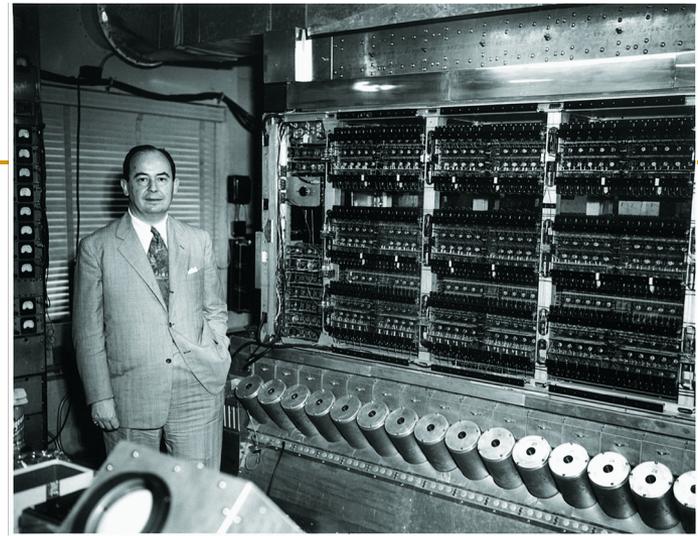
Data → performance & energy bottleneck

Data Overwhelms Modern Machines ...

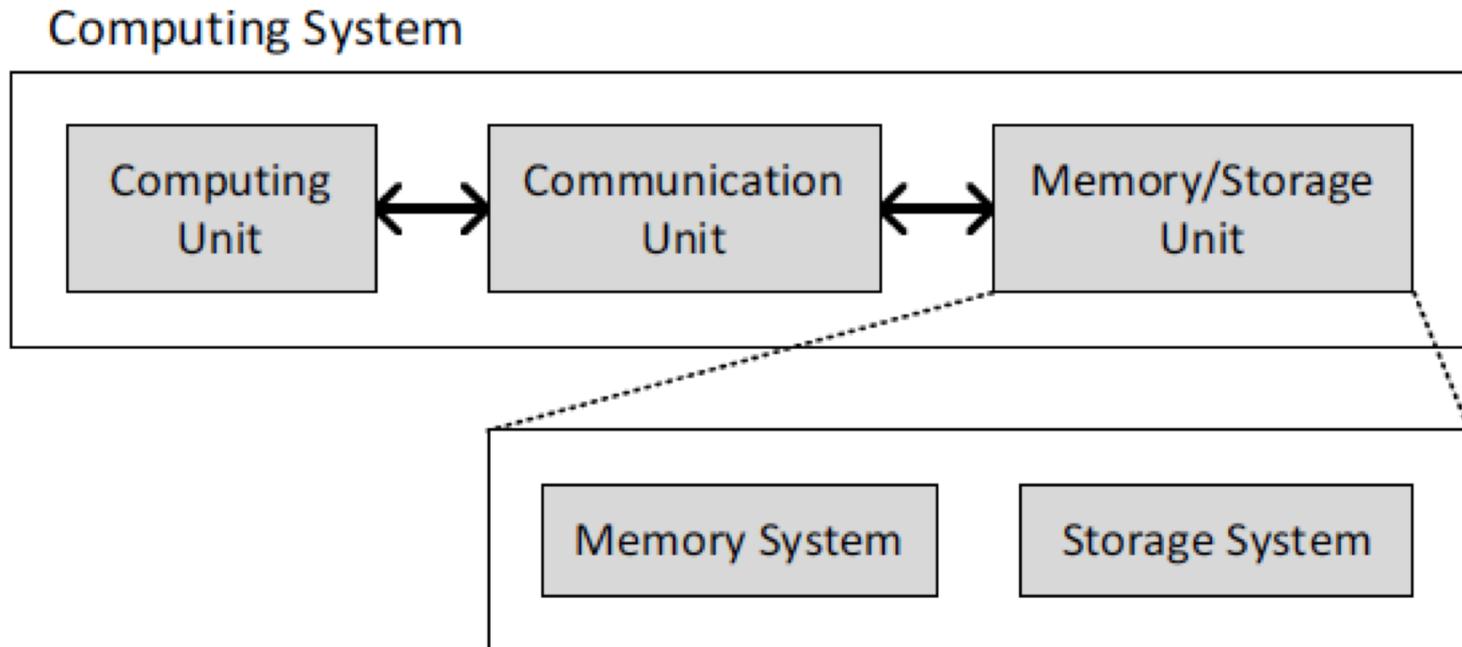
- Storage/memory capability
- Communication capability
- Computation capability

A Computing System

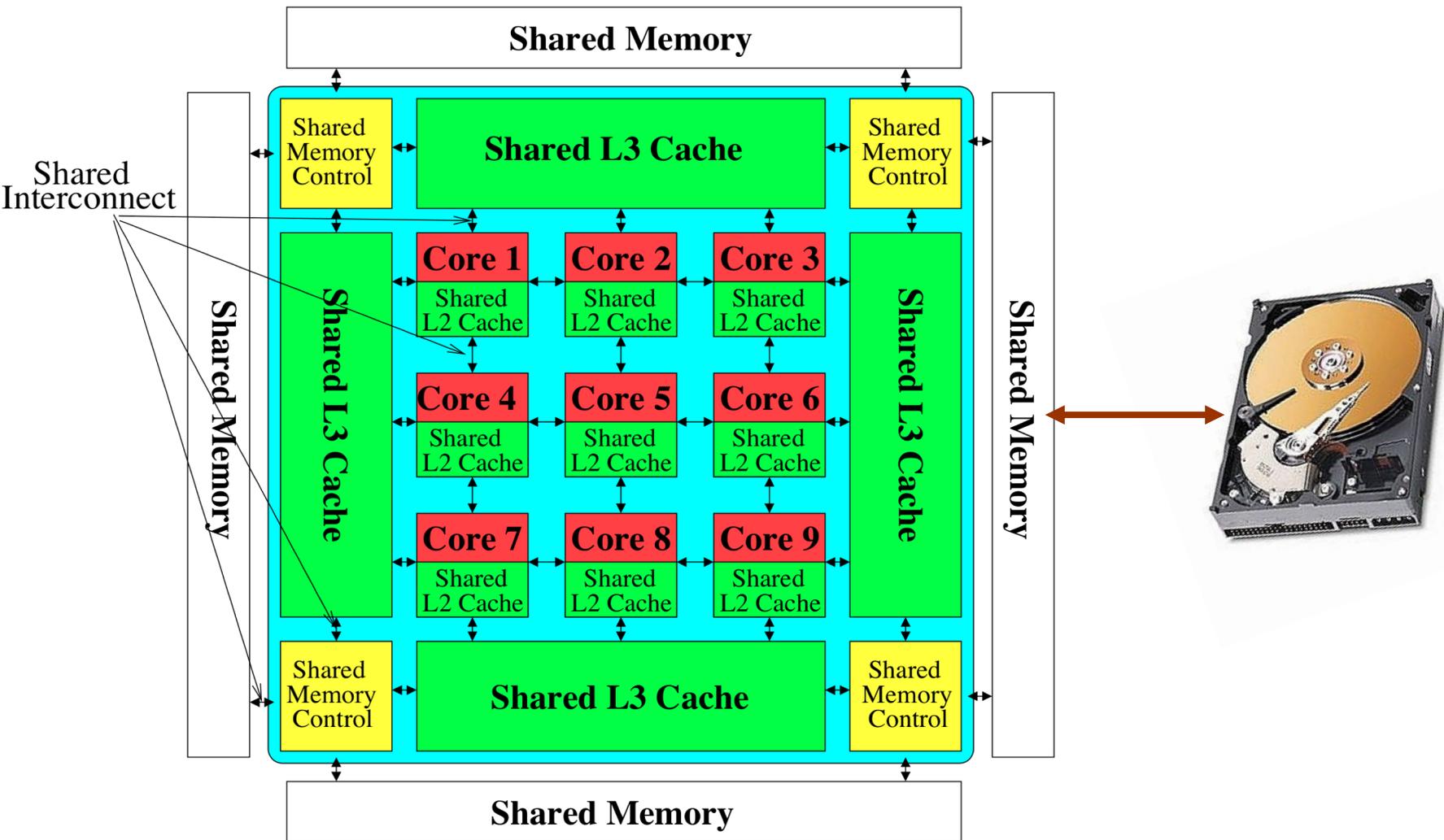
- Three key components
- Computation
- Communication
- Storage/memory



Burks, Goldstein, von Neumann, "Preliminary discussion of the logical design of an electronic computing instrument," 1946.



Perils of Processor-Centric Design



Most of the system is dedicated to storing and moving data

Data Overwhelms Modern Machines



Chrome



TensorFlow Mobile

Data → performance & energy bottleneck

VP9



Video Playback

Google's **video codec**

VP9



Video Capture

Google's **video codec**

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 Boroumand¹

Saugata Ghose¹

Youngsok Kim²

Rachata Ausavarungnirun¹

Eric Shiu³

Rahul Thakur³

Daehyun Kim^{4,3}

Aki Kuusela³

Allan Knies³

Parthasarathy Ranganathan³

Onur Mutlu^{5,1}

An Intelligent Architecture Handles Data Well

How to Handle Data Well

- Ensure data does not overwhelm the components
 - via intelligent algorithms
 - via intelligent architectures
 - via whole system designs: algorithm-architecture-devices

- Take advantage of vast amounts of data and metadata
 - to improve architectural & system-level decisions

- Understand and exploit properties of (different) data
 - to improve algorithms & architectures in various metrics

Corollaries: Architectures Today ...

- Architectures are **terrible at dealing with data**
 - Designed to mainly store and move data vs. to compute
 - They are **processor-centric** as opposed to **data-centric**
- Architectures are **terrible at taking advantage of vast amounts of data** (and metadata) available to them
 - Designed to make simple decisions, ignoring lots of data
 - They make **human-driven decisions** vs. **data-driven** decisions
- Architectures are **terrible at knowing and exploiting different properties of application data**
 - Designed to treat all data as the same
 - They make **component-aware decisions** vs. **data-aware**

Processing Data

Where It Makes Sense

Three Key Systems Trends

1. Data access is a major bottleneck

- Applications are increasingly data hungry

2. Energy consumption is a key limiter

3. Data movement energy dominates compute

- Especially true for off-chip to on-chip movement

Do We Want This?



Or This?



Challenge and Opportunity for Future

High Performance,
Energy Efficient,
Sustainable

The Problem

Data access is the major performance and energy bottleneck

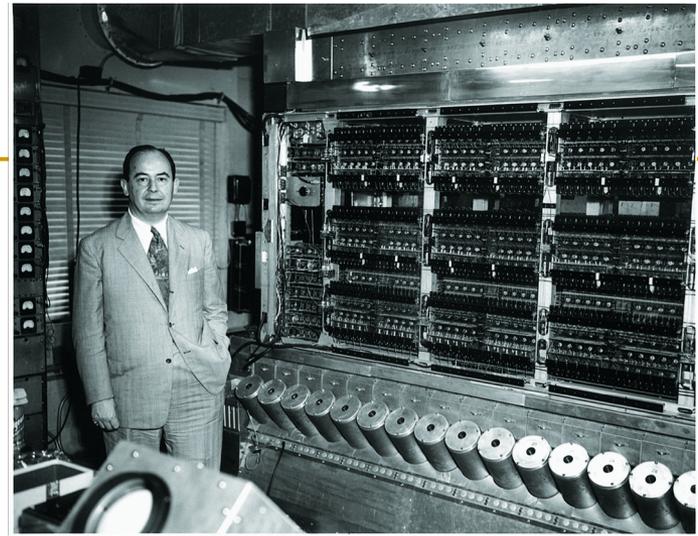
Our current
design principles
cause great energy waste
(and great performance loss)

The Problem

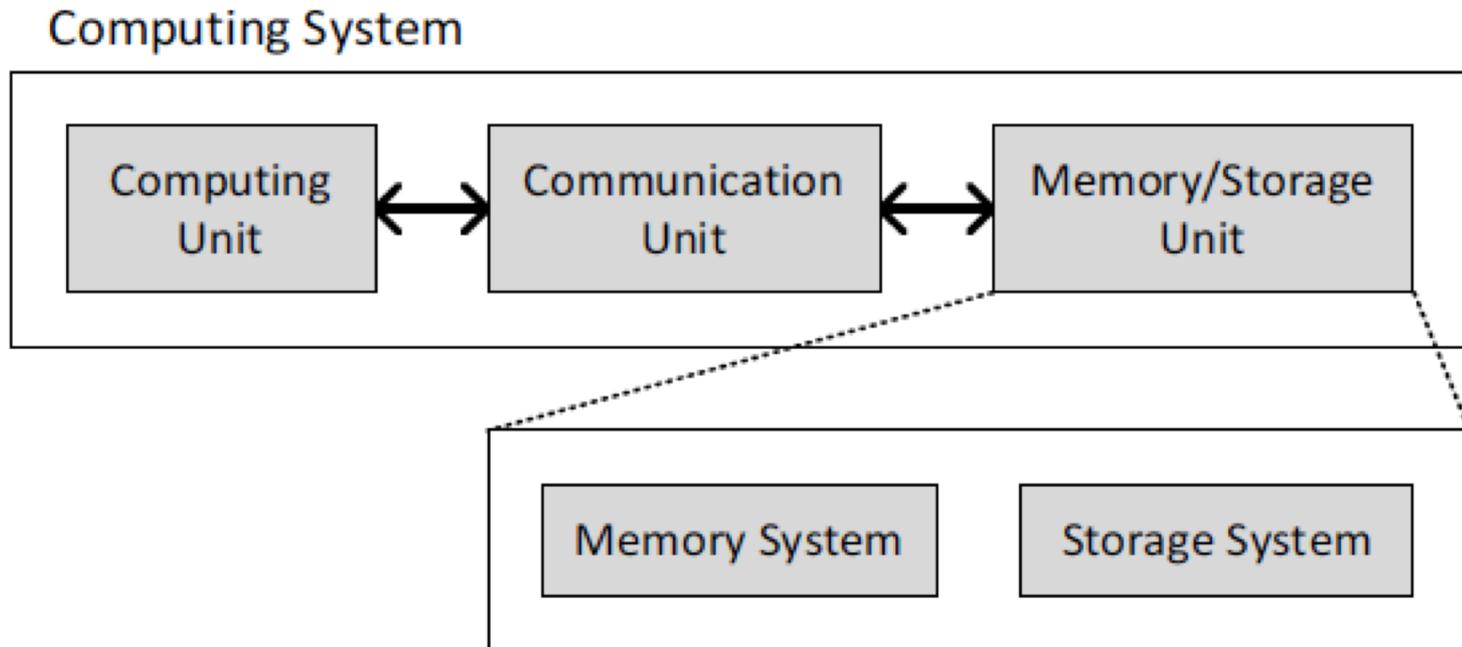
Processing of data
is performed
far away from the data

A Computing System

- Three key components
- Computation
- Communication
- Storage/memory

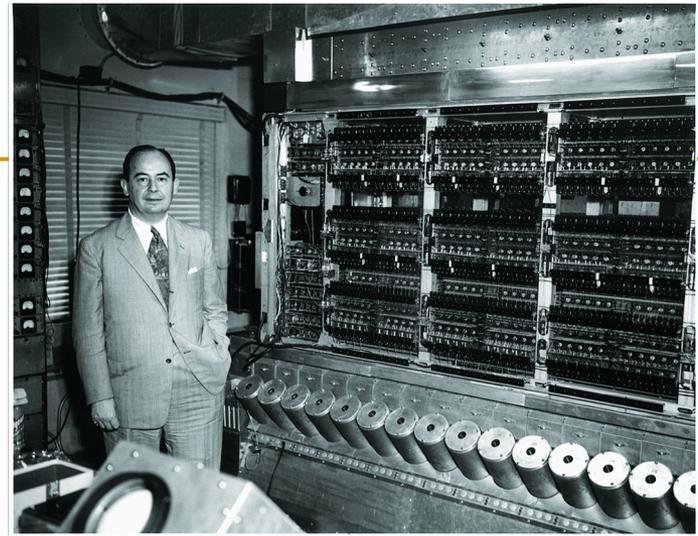


Burks, Goldstein, von Neumann, "Preliminary discussion of the logical design of an electronic computing instrument," 1946.



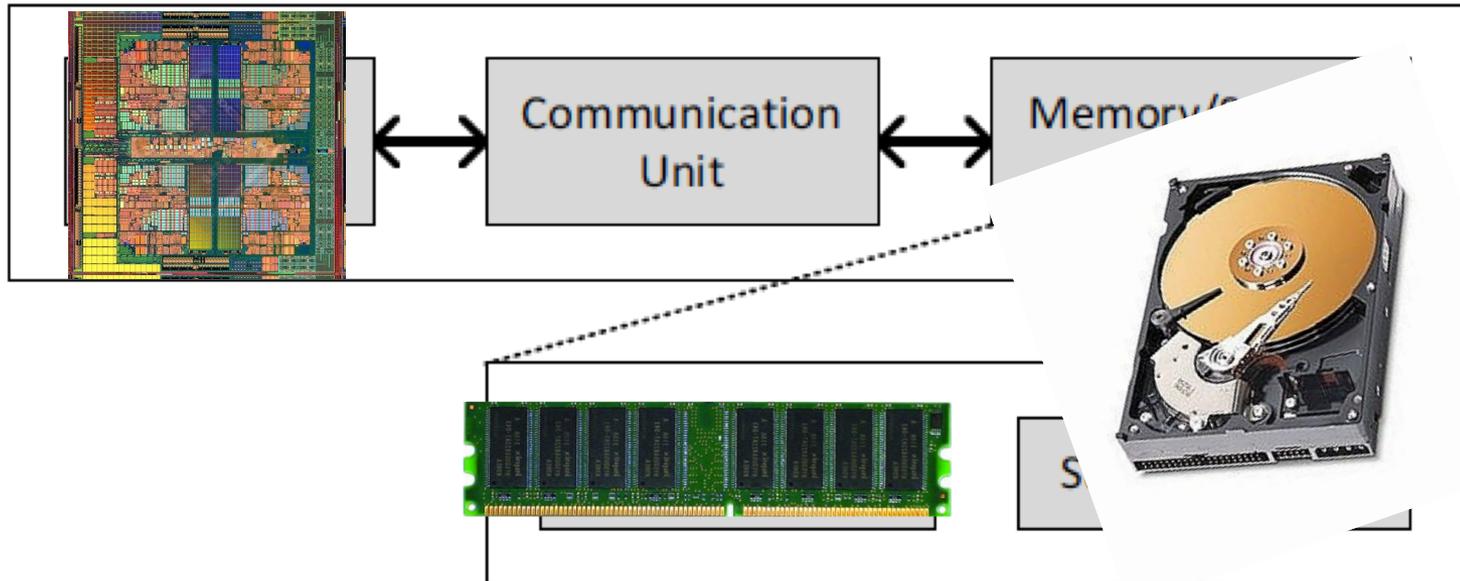
A Computing System

- Three key components
- Computation
- Communication
- Storage/memory



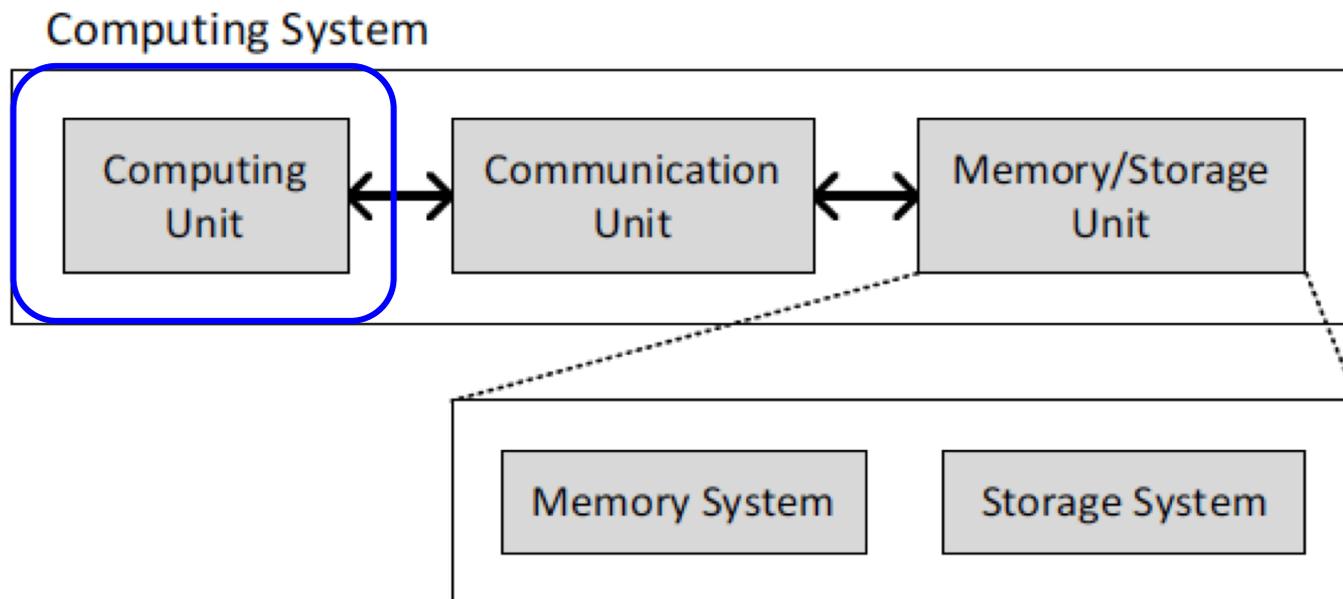
Burks, Goldstein, von Neumann, "Preliminary discussion of the logical design of an electronic computing instrument," 1946.

Computing System



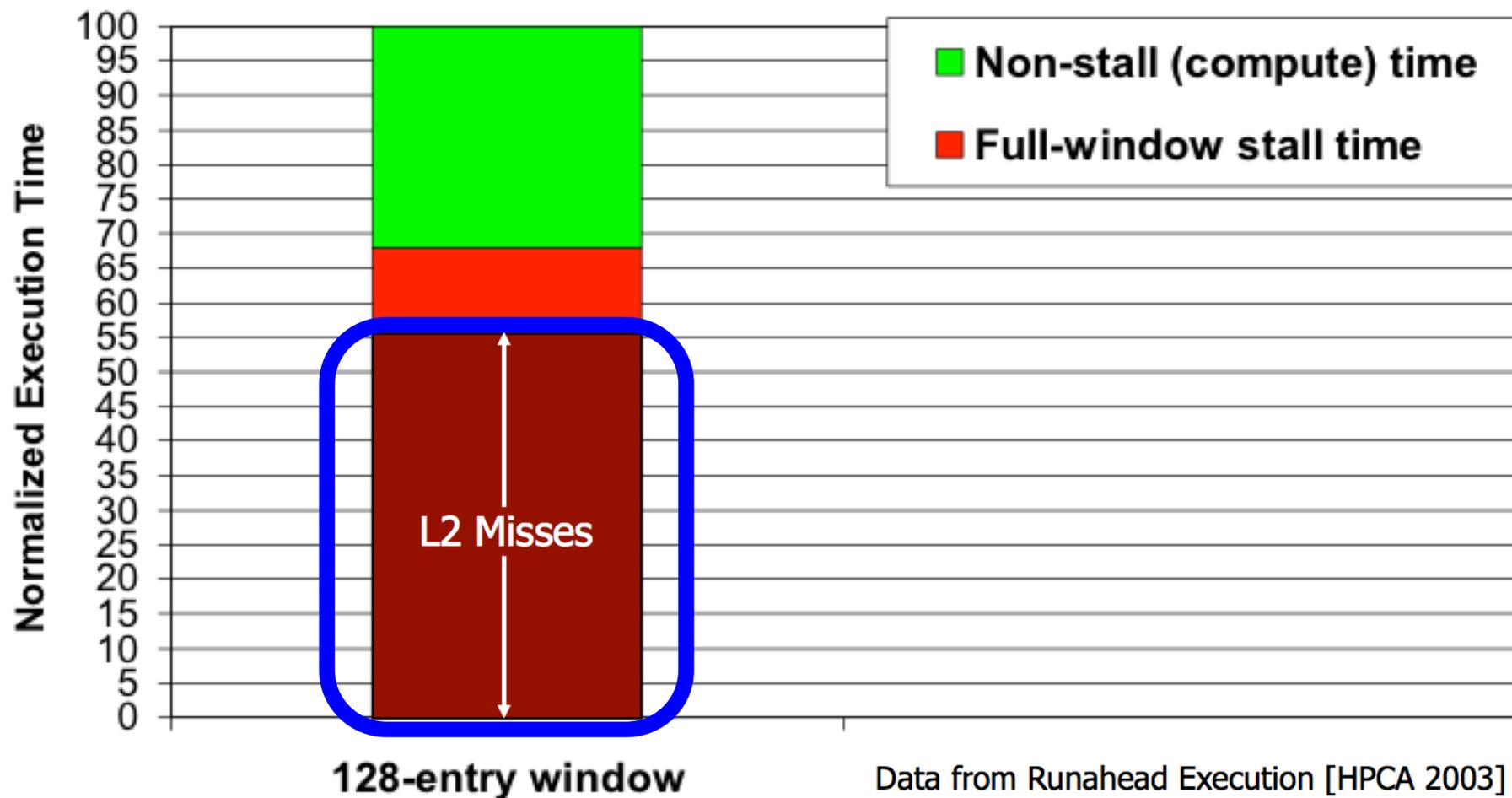
Today's Computing Systems

- Are overwhelmingly processor centric
- **All data processed in the processor** → at great system cost
- Processor is heavily optimized and is considered the master
- **Data storage units are dumb** and are largely unoptimized (except for some that are on the processor die)



Yet ...

- **“It’s the Memory, Stupid!”** (Richard Sites, MPR, 1996)



The Performance Perspective

- Onur Mutlu, Jared Stark, Chris Wilkerson, and Yale N. Patt, **"Runahead Execution: An Alternative to Very Large Instruction Windows for Out-of-order Processors"**
Proceedings of the 9th International Symposium on High-Performance Computer Architecture (HPCA), pages 129-140, Anaheim, CA, February 2003. [Slides \(pdf\)](#)

Runahead Execution: An Alternative to Very Large Instruction Windows for Out-of-order Processors

Onur Mutlu § Jared Stark † Chris Wilkerson ‡ Yale N. Patt §

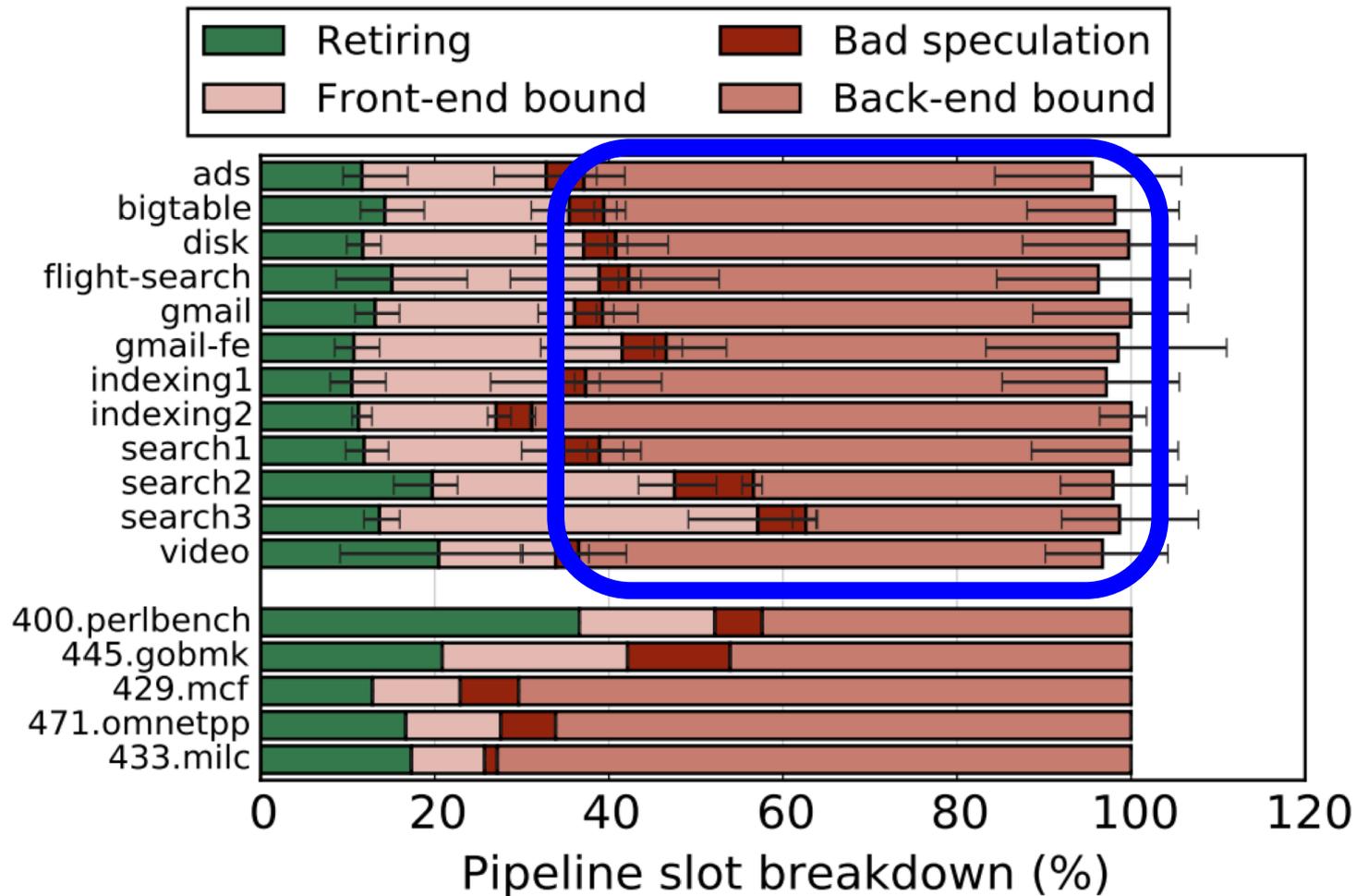
§ECE Department
The University of Texas at Austin
{onur,patt}@ece.utexas.edu

†Microprocessor Research
Intel Labs
jared.w.stark@intel.com

‡Desktop Platforms Group
Intel Corporation
chris.wilkerson@intel.com

The Performance Perspective (Today)

- All of Google's Data Center Workloads (2015):



The Performance Perspective (Today)

- All of Google's Data Center Workloads (2015):

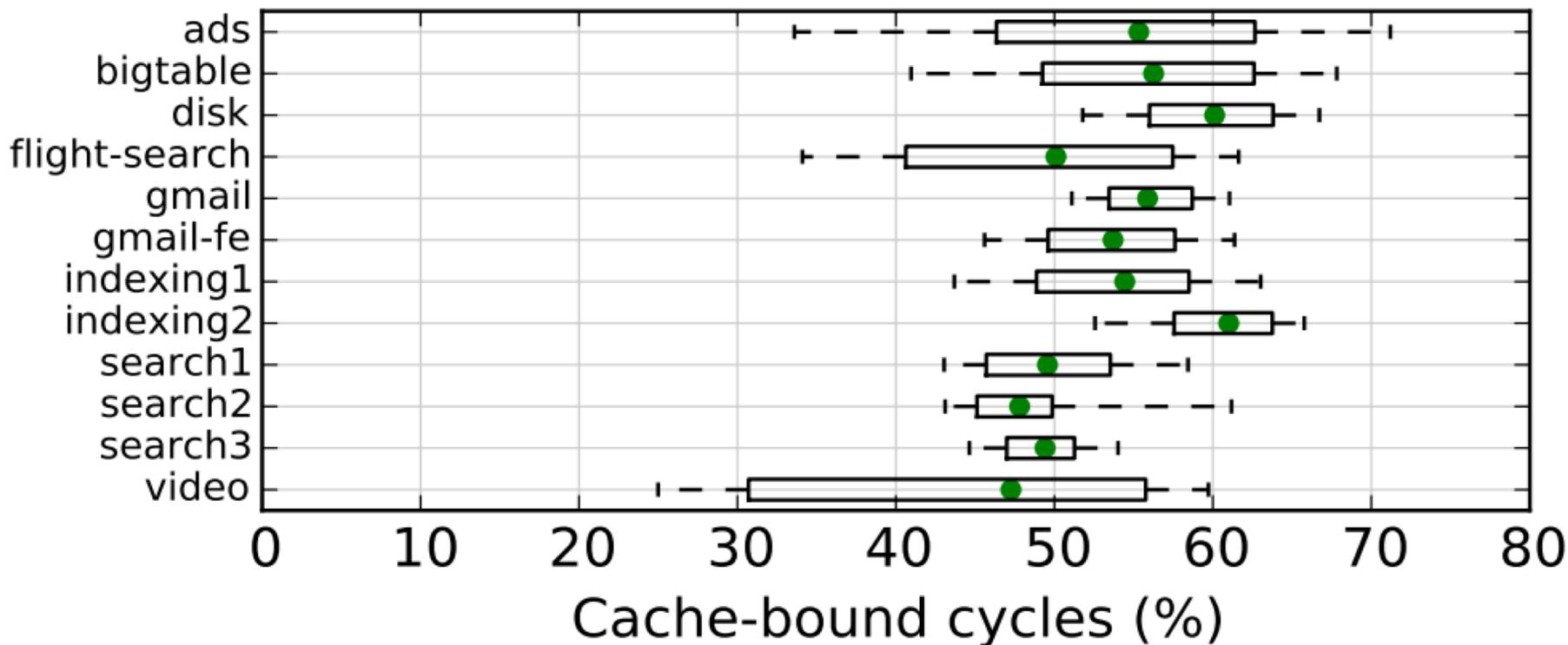


Figure 11: Half of cycles are spent stalled on caches.

Perils of Processor-Centric Design

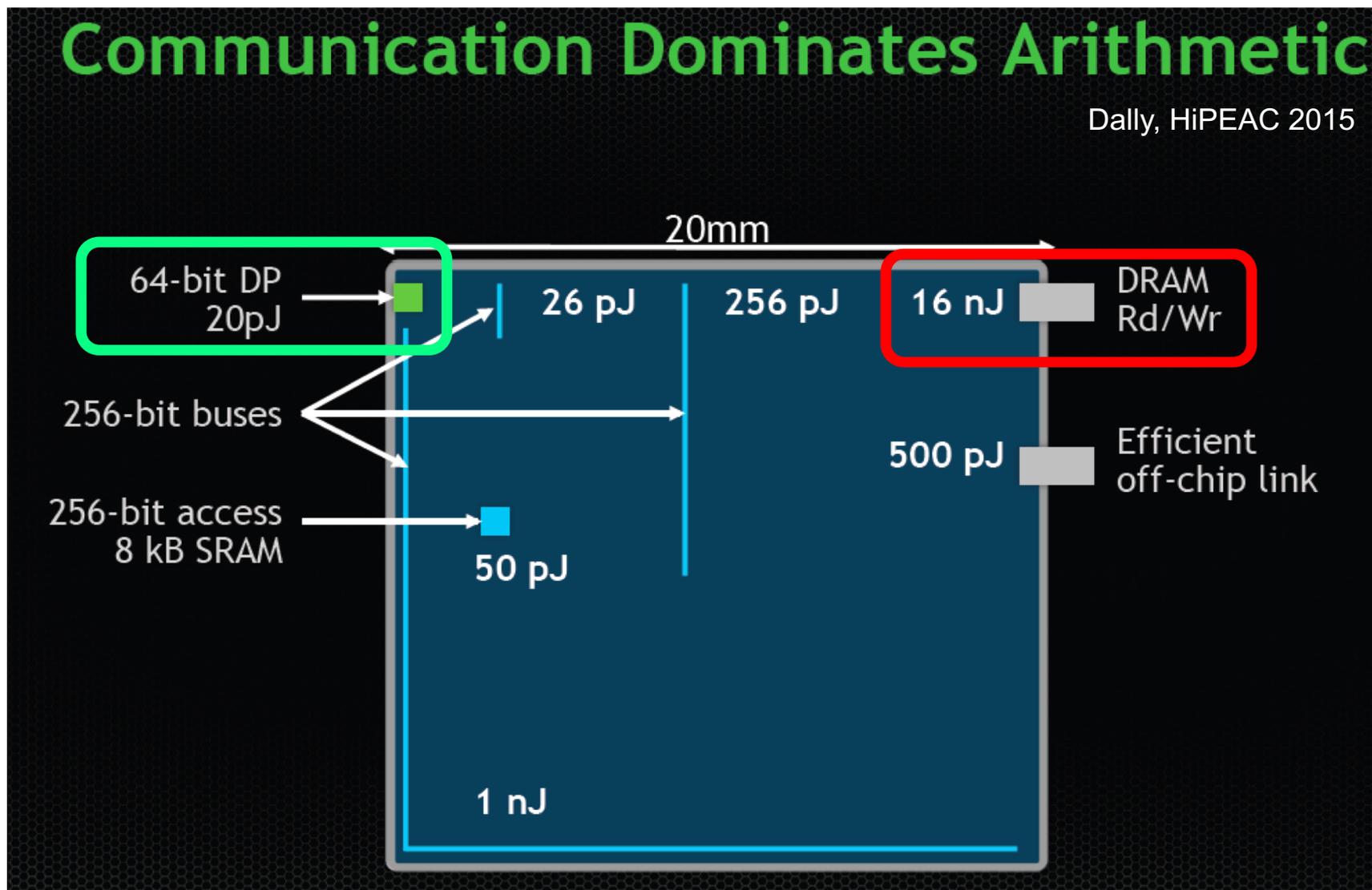
- **Grossly-imbalanced systems**
 - ❑ Processing done only in **one place**
 - ❑ Everything else just stores and moves data: **data moves a lot**
 - Energy inefficient
 - Low performance
 - Complex

- **Overly complex and bloated processor (and accelerators)**
 - ❑ To tolerate data access from memory
 - ❑ Complex hierarchies and mechanisms
 - Energy inefficient
 - Low performance
 - Complex

The Energy Perspective

Communication Dominates Arithmetic

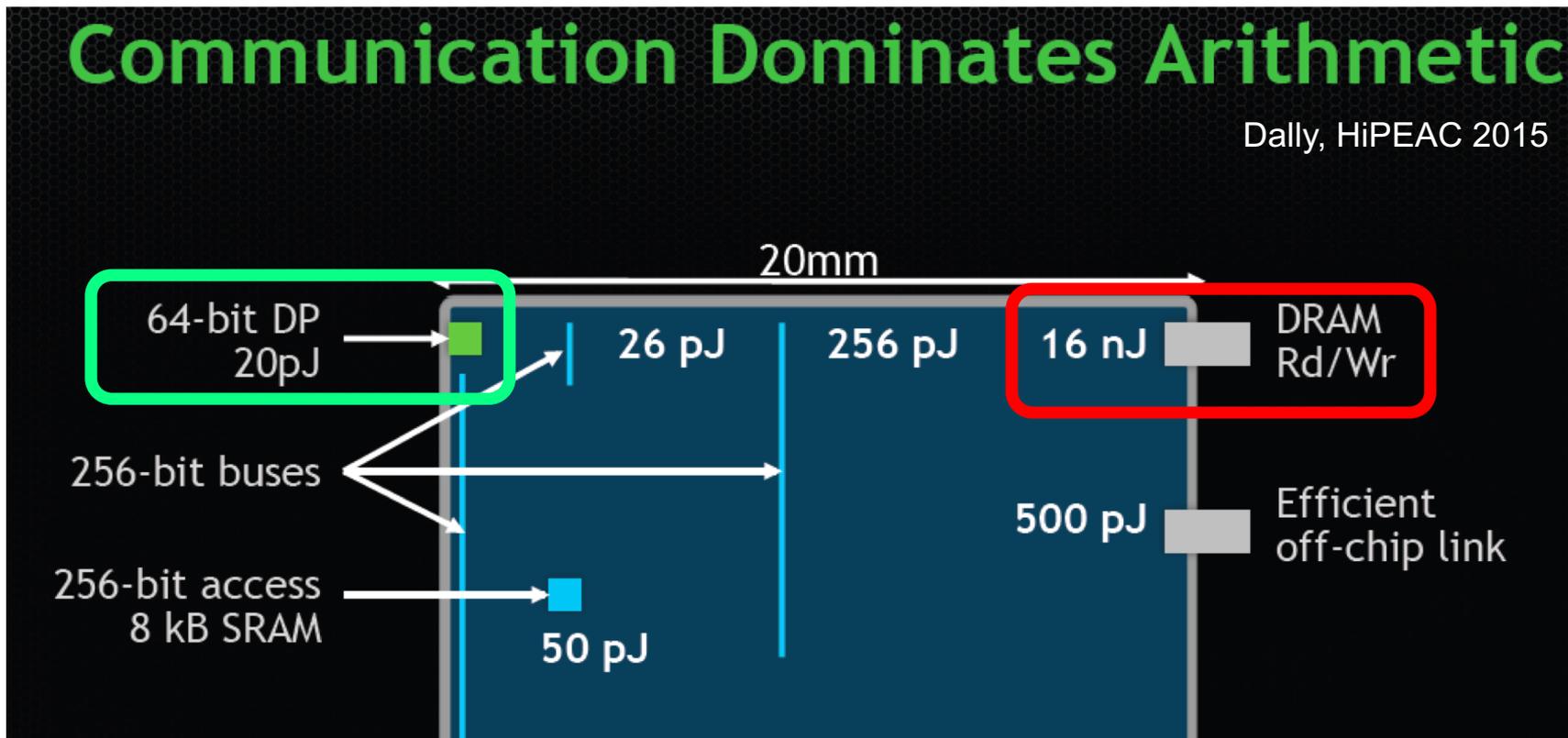
Dally, HiPEAC 2015



Data Movement vs. Computation Energy

Communication Dominates Arithmetic

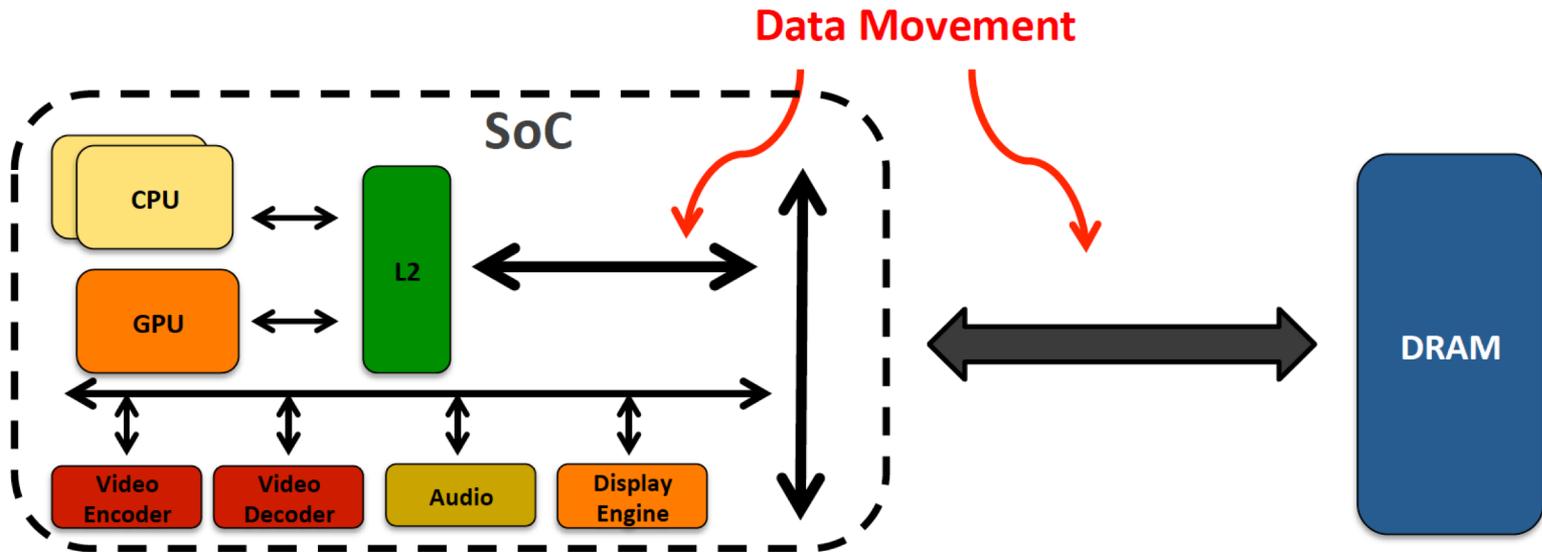
Dally, HiPEAC 2015



A memory access consumes $\sim 1000X$ the energy of a complex addition

Data Movement vs. Computation Energy

- **Data movement** is a major system energy bottleneck
 - Comprises 41% of mobile system energy during web browsing [2]
 - Costs ~ 115 times as much energy as an ADD operation [1, 2]



[1]: Reducing data Movement Energy via Online Data Clustering and Encoding (MICRO'16)

[2]: Quantifying the energy cost of data movement for emerging smart phone workloads on mobile platforms (IISWC'14)

Energy Waste in Mobile Devices

- 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 Boroumand¹

Saugata Ghose¹

Youngsok Kim²

Rachata Ausavarungnirun¹

Eric Shiu³

Rahul Thakur³

Daehyun Kim^{4,3}

Aki Kuusela³

Allan Knies³

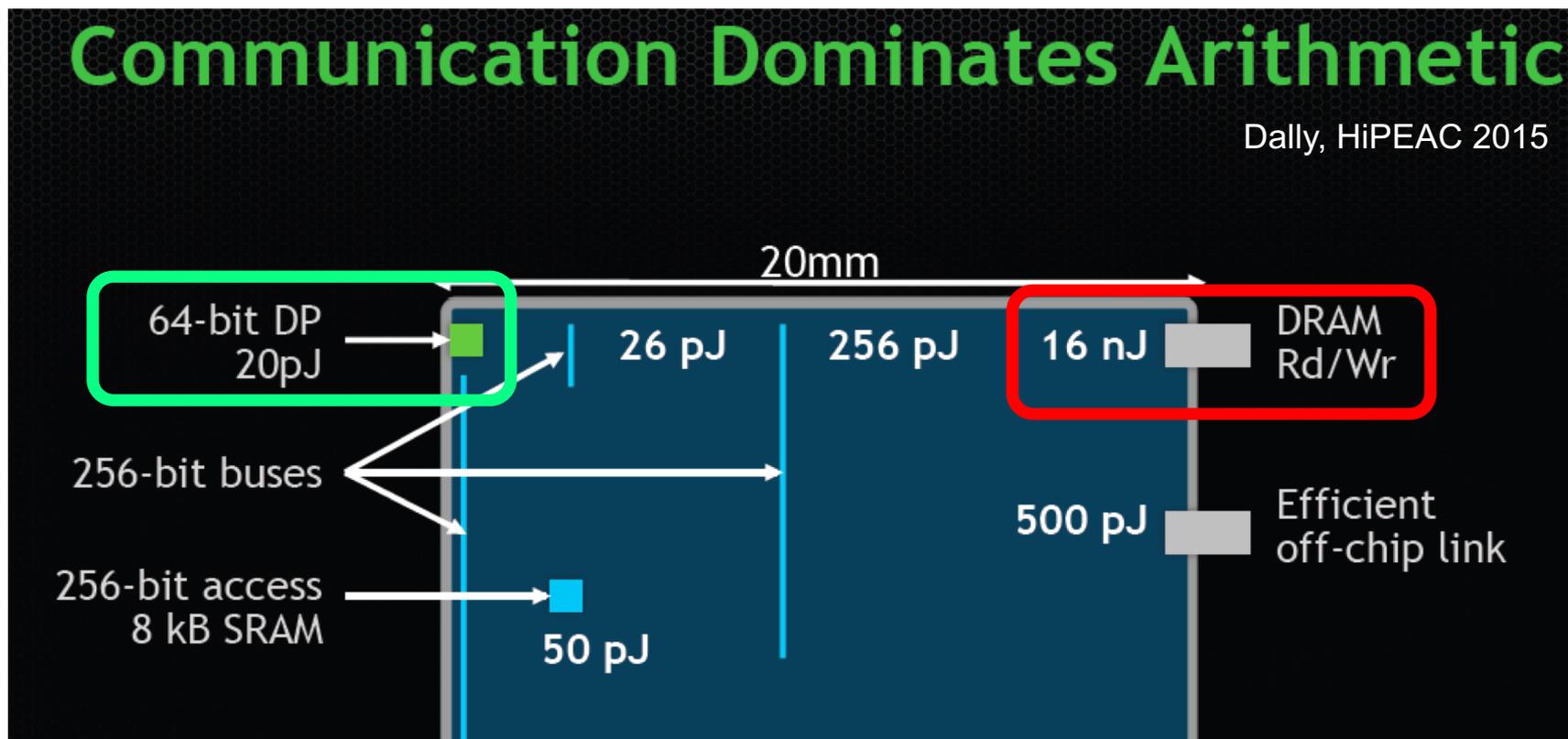
Parthasarathy Ranganathan³

Onur Mutlu^{5,1}

We Do Not Want to Move Data!

Communication Dominates Arithmetic

Dally, HiPEAC 2015

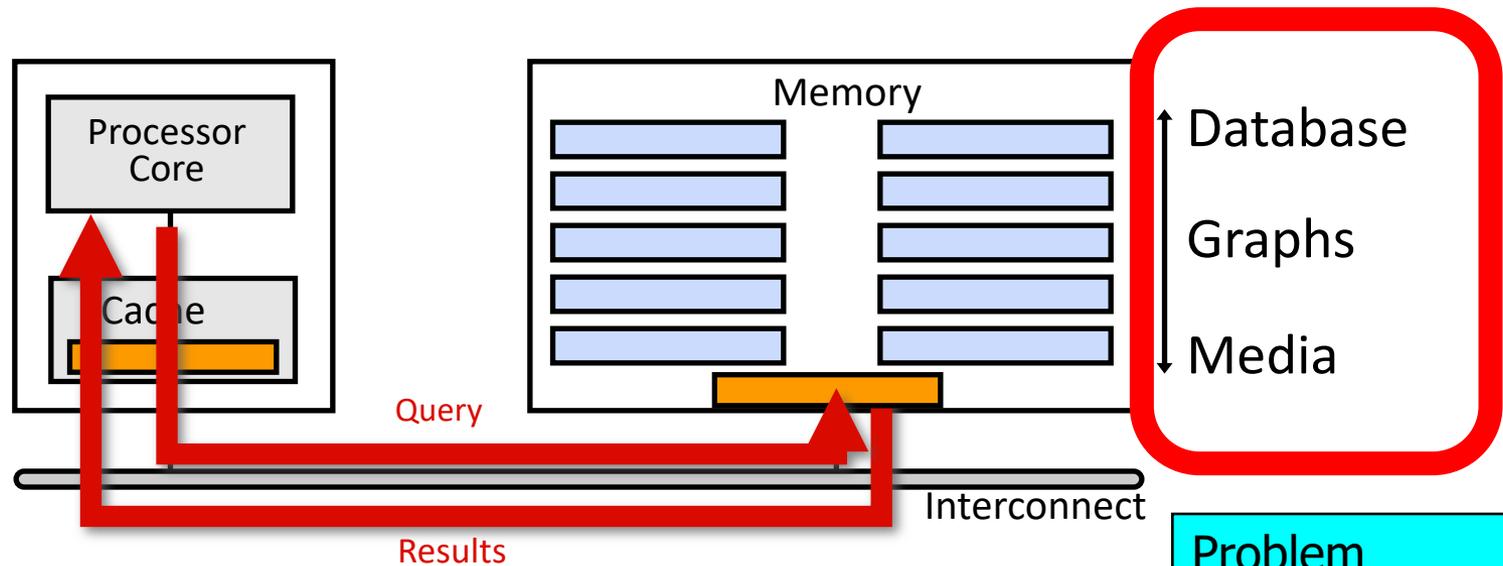


A memory access consumes $\sim 1000X$
the energy of a complex addition

We Need A Paradigm Shift To ...

- Enable computation with minimal data movement
- Compute where it makes sense (where data resides)
- Make computing architectures more data-centric

Goal: Processing Inside Memory



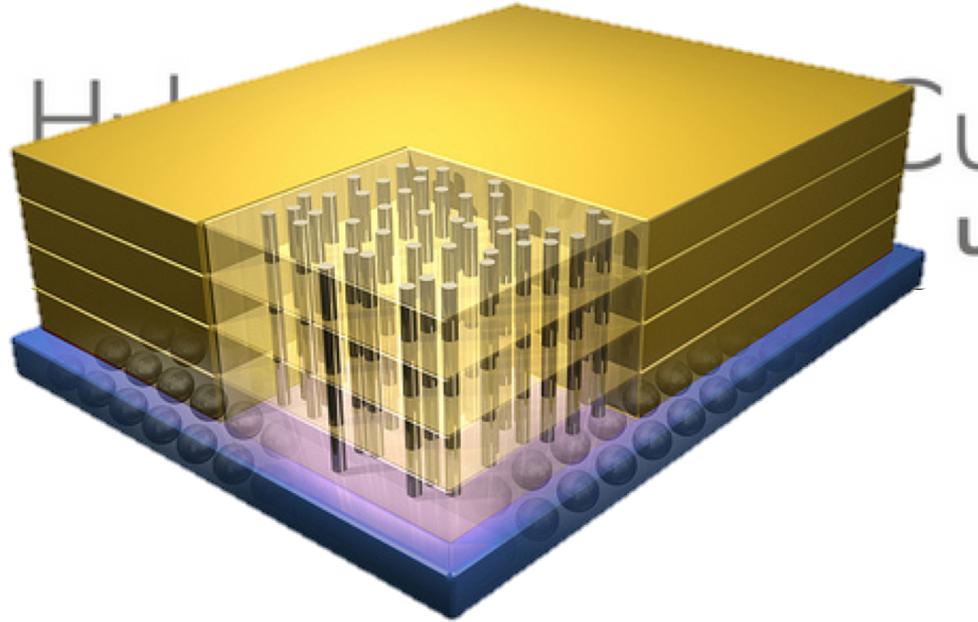
- Many questions ... How do we design the:
 - compute-capable memory & controllers?
 - processor chip and in-memory units?
 - software and hardware interfaces?
 - system software and languages?
 - algorithms?

Problem
Algorithm
Program/Language
System Software
SW/HW Interface
Micro-architecture
Logic
Devices
Electrons

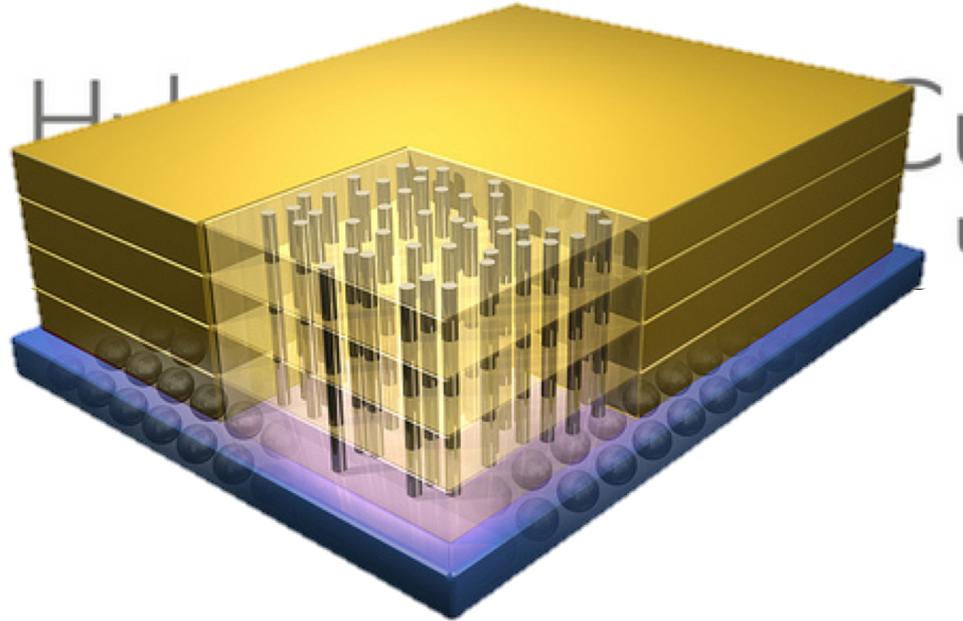
Why In-Memory Computation Today?

- **Push from Technology**
 - **DRAM Scaling at jeopardy**
 - Controllers close to DRAM
 - Industry open to new memory architectures

Why In-Memory Computation Today?



Why In-Memory Computation Today?



- Pull from Systems and Applications
 - Data access is a major system and application bottleneck
 - Systems are energy limited
 - Data movement much more energy-hungry than computation

Processing in Memory: Two Approaches

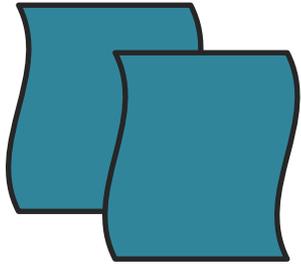
1. Minimally changing memory chips
2. Exploiting 3D-stacked memory

Approach 1: Minimally Changing DRAM

- DRAM has great capability to perform **bulk data movement and computation** internally with small changes
 - Can **exploit internal connectivity** to move data
 - Can **exploit analog computation capability**
 - ...
- Examples: RowClone, In-DRAM AND/OR, Gather/Scatter DRAM
 - RowClone: Fast and Efficient In-DRAM Copy and Initialization of Bulk Data (Seshadri et al., MICRO 2013)
 - Fast Bulk Bitwise AND and OR in DRAM (Seshadri et al., IEEE CAL 2015)
 - Gather-Scatter DRAM: In-DRAM Address Translation to Improve the Spatial Locality of Non-unit Strided Accesses (Seshadri et al., MICRO 2015)
 - "Ambit: In-Memory Accelerator for Bulk Bitwise Operations Using Commodity DRAM Technology" (Seshadri et al., MICRO 2017)

Starting Simple: Data Copy and Initialization

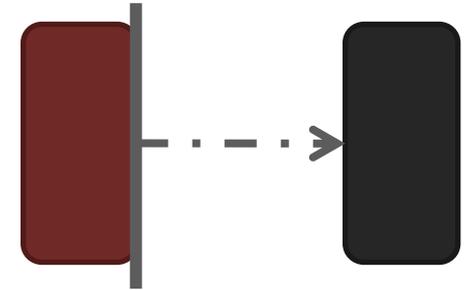
memmove & memcpy: 5% cycles in Google's datacenter [Kanev+ ISCA'15]



Forking



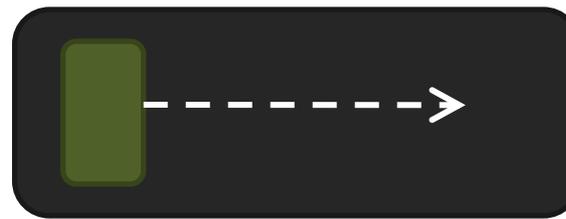
**Zero initialization
(e.g., security)**



Checkpointing



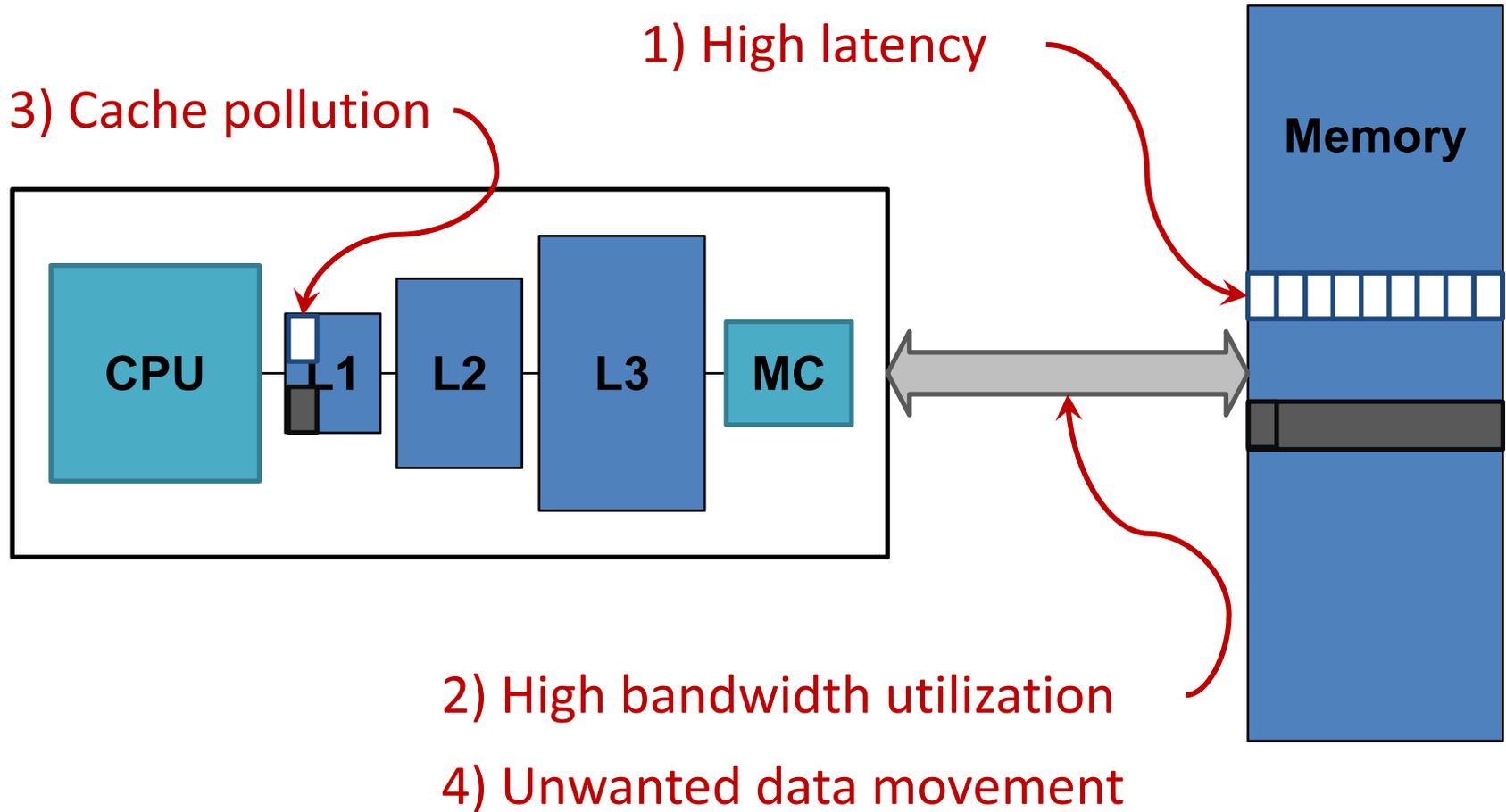
**VM Cloning
Deduplication**



Page Migration

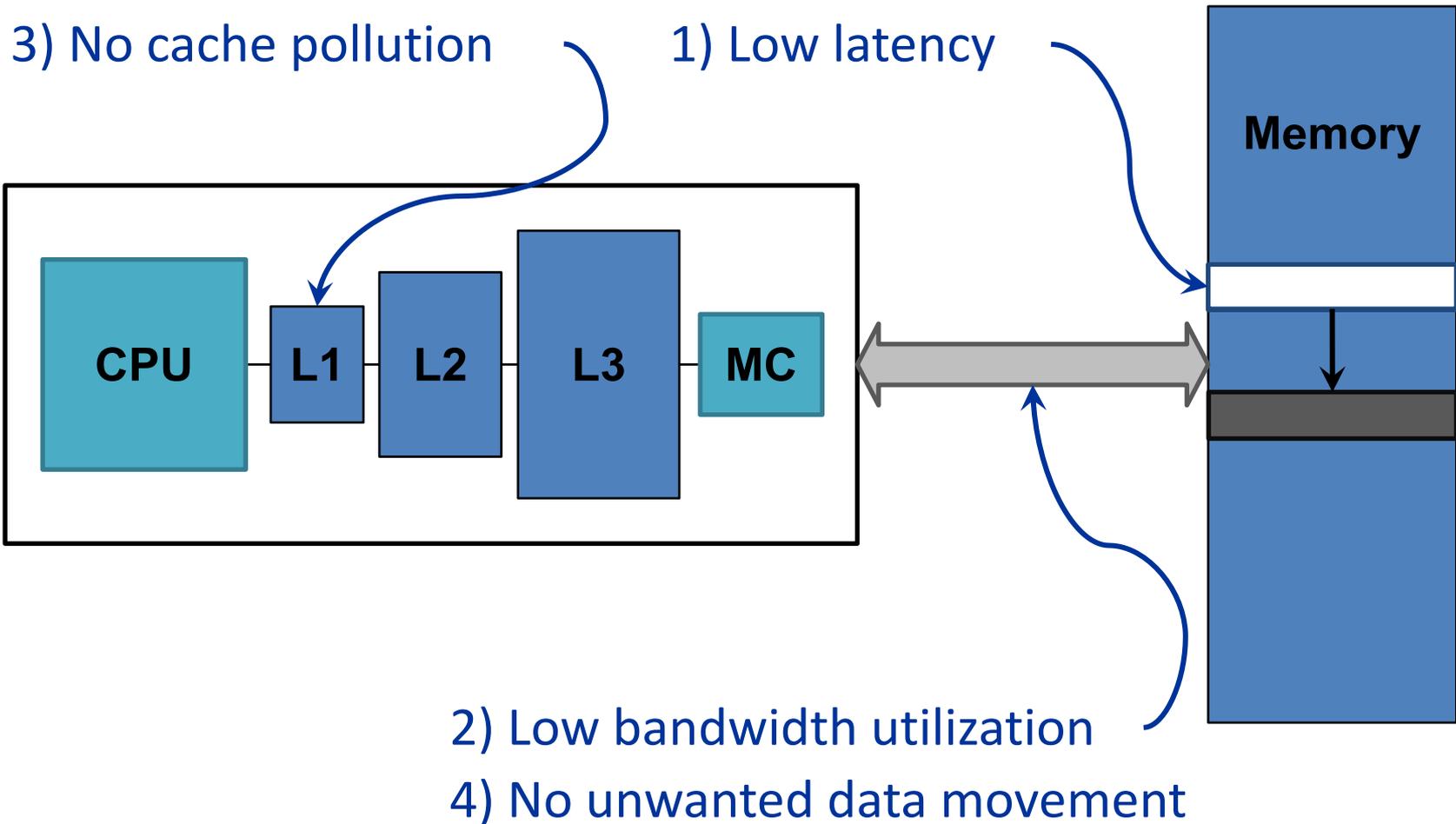
•••
Many more

Today's Systems: Bulk Data Copy



1046ns, 3.6uJ (for 4KB page copy via DMA)

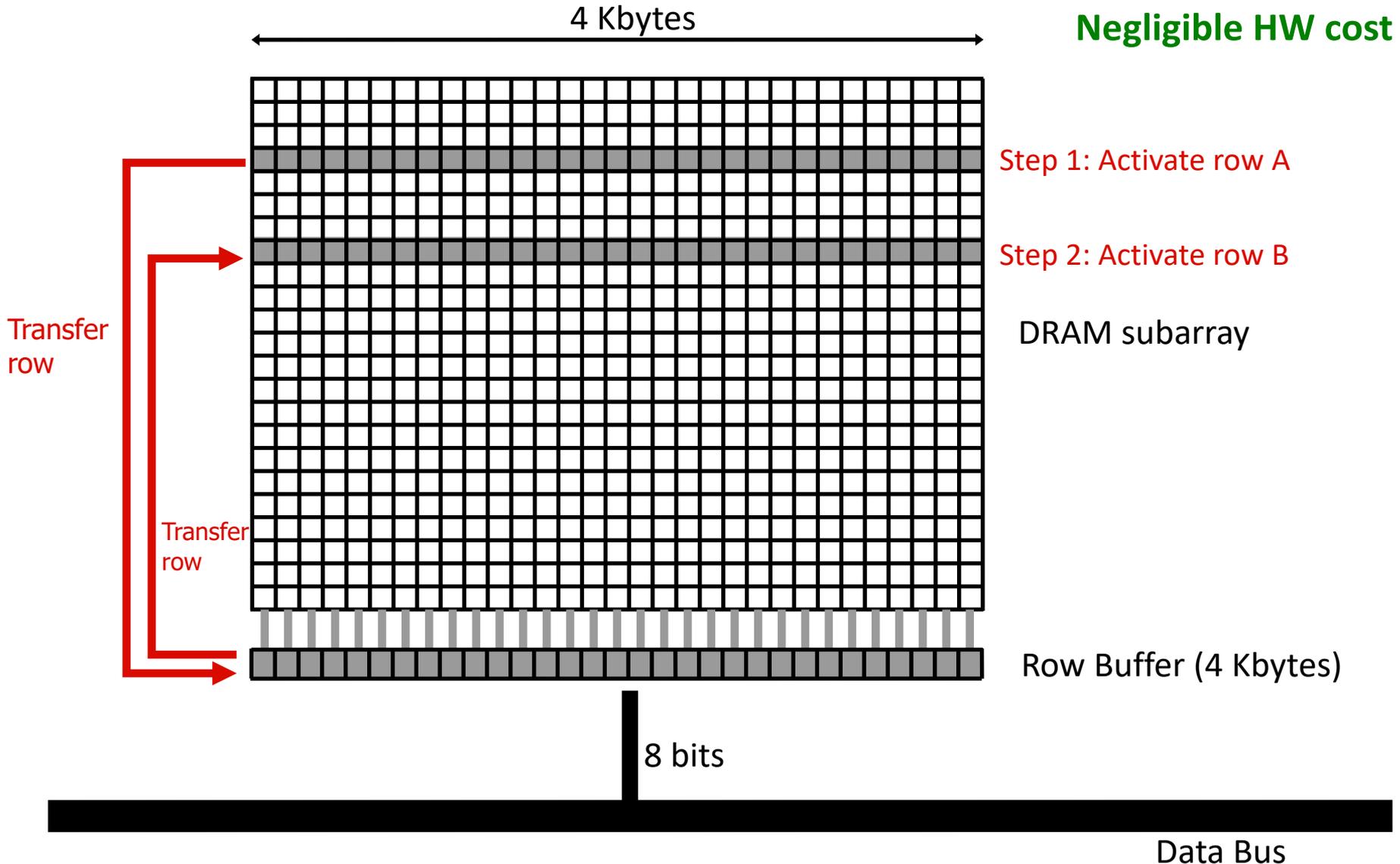
Future Systems: In-Memory Copy



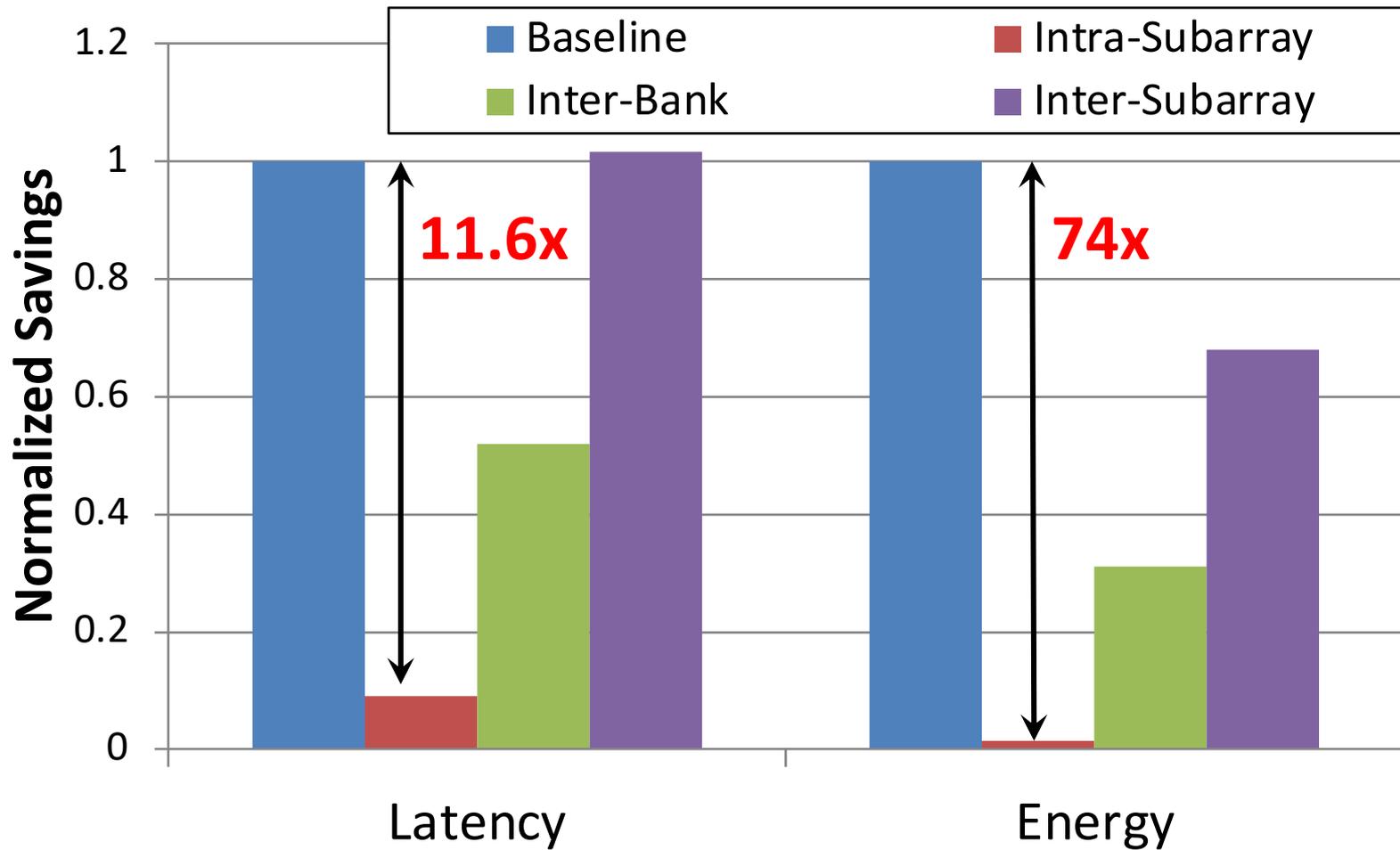
1046ns, 3.6uJ → 90ns, 0.04uJ

RowClone: In-DRAM Row Copy

**Idea: Two consecutive ACTivates
Negligible HW cost**



RowClone: Latency and Energy Savings



Seshadri et al., "RowClone: Fast and Efficient In-DRAM Copy and Initialization of Bulk Data," MICRO 2013.

More on 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

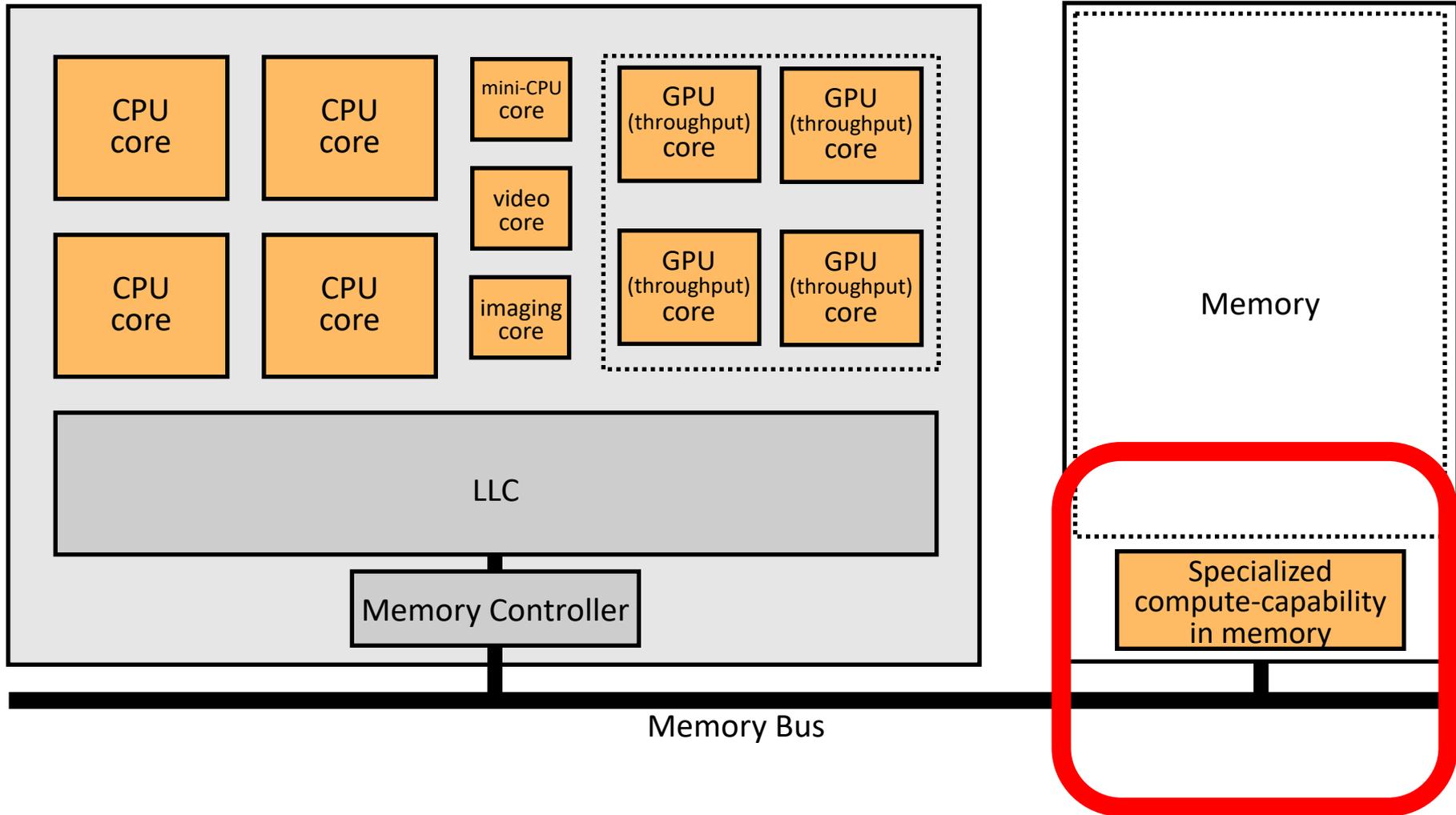
Vivek Seshadri Yoongu Kim Chris Fallin* Donghyuk Lee
vseshadr@cs.cmu.edu yoongukim@cmu.edu cfallin@c1f.net donghyuk1@cmu.edu

Rachata Ausavarungnirun Gennady Pekhimenko Yixin Luo
rachata@cmu.edu gpekhime@cs.cmu.edu yixinluo@andrew.cmu.edu

Onur Mutlu Phillip B. Gibbons† Michael A. Kozuch† Todd C. Mowry
onur@cmu.edu phillip.b.gibbons@intel.com michael.a.kozuch@intel.com tcm@cs.cmu.edu

Carnegie Mellon University †Intel Pittsburgh

Memory as an Accelerator



Memory similar to a "conventional" accelerator

In-Memory Bulk Bitwise Operations

- We can support **in-DRAM COPY, ZERO, AND, OR, NOT, MAJ**
- At low cost
- Using analog computation capability of DRAM
 - Idea: activating multiple rows performs computation
- **30-60X performance and energy improvement**
 - Seshadri+, "Ambit: In-Memory Accelerator for Bulk Bitwise Operations Using Commodity DRAM Technology," MICRO 2017.

- **New memory technologies** enable even more opportunities
 - Memristors, resistive RAM, phase change mem, STT-MRAM, ...
 - Can operate on data **with minimal movement**

In-DRAM NOT: Dual Contact Cell

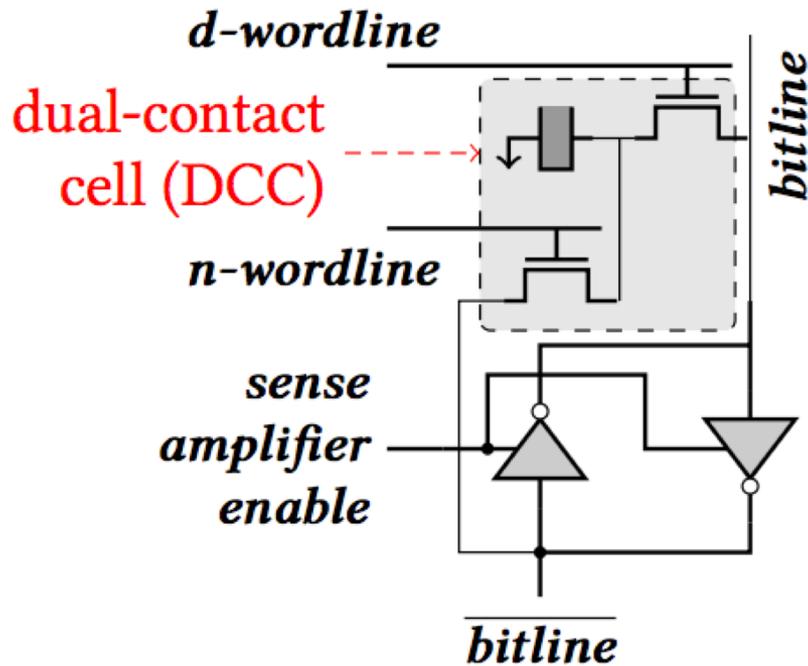
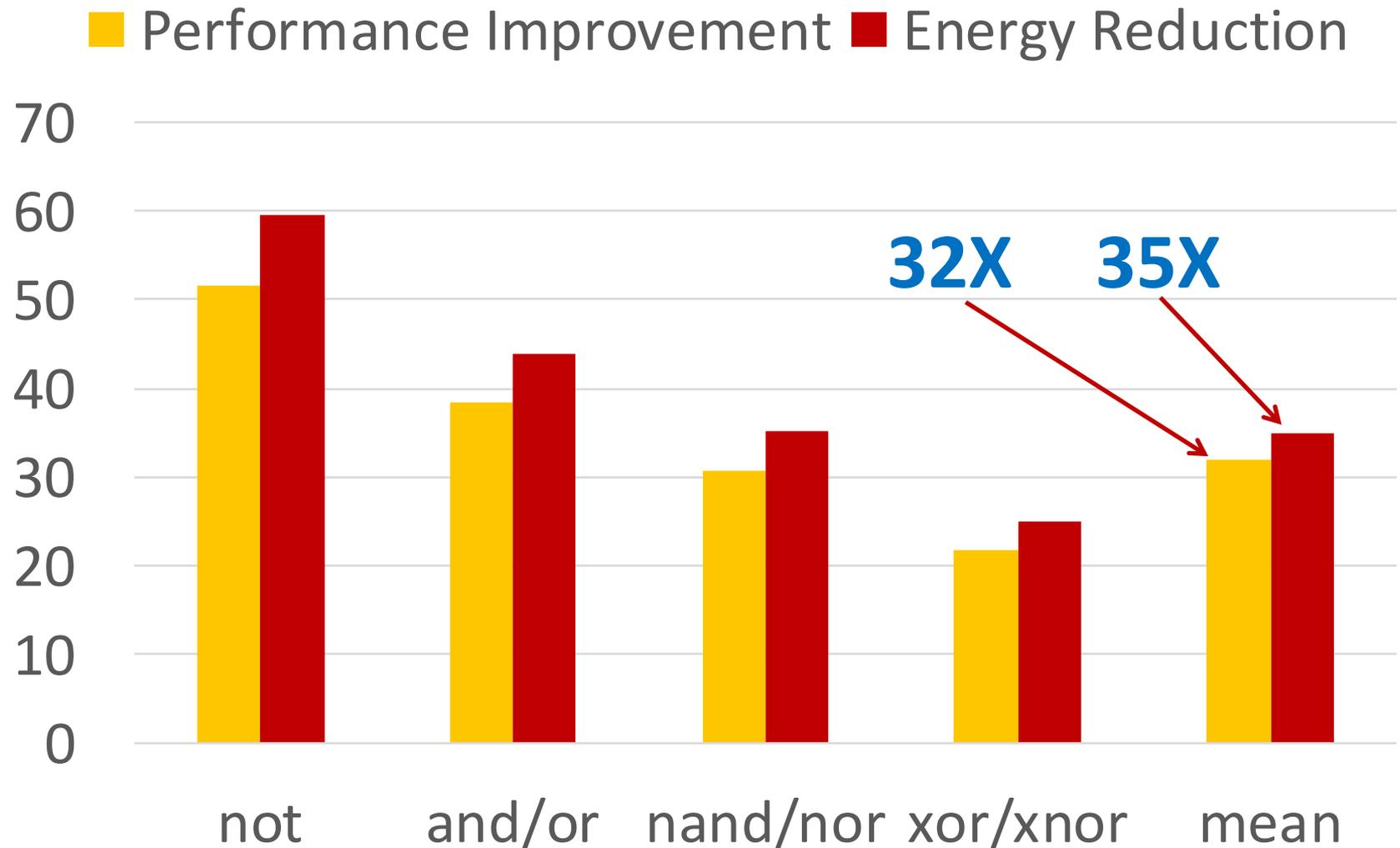


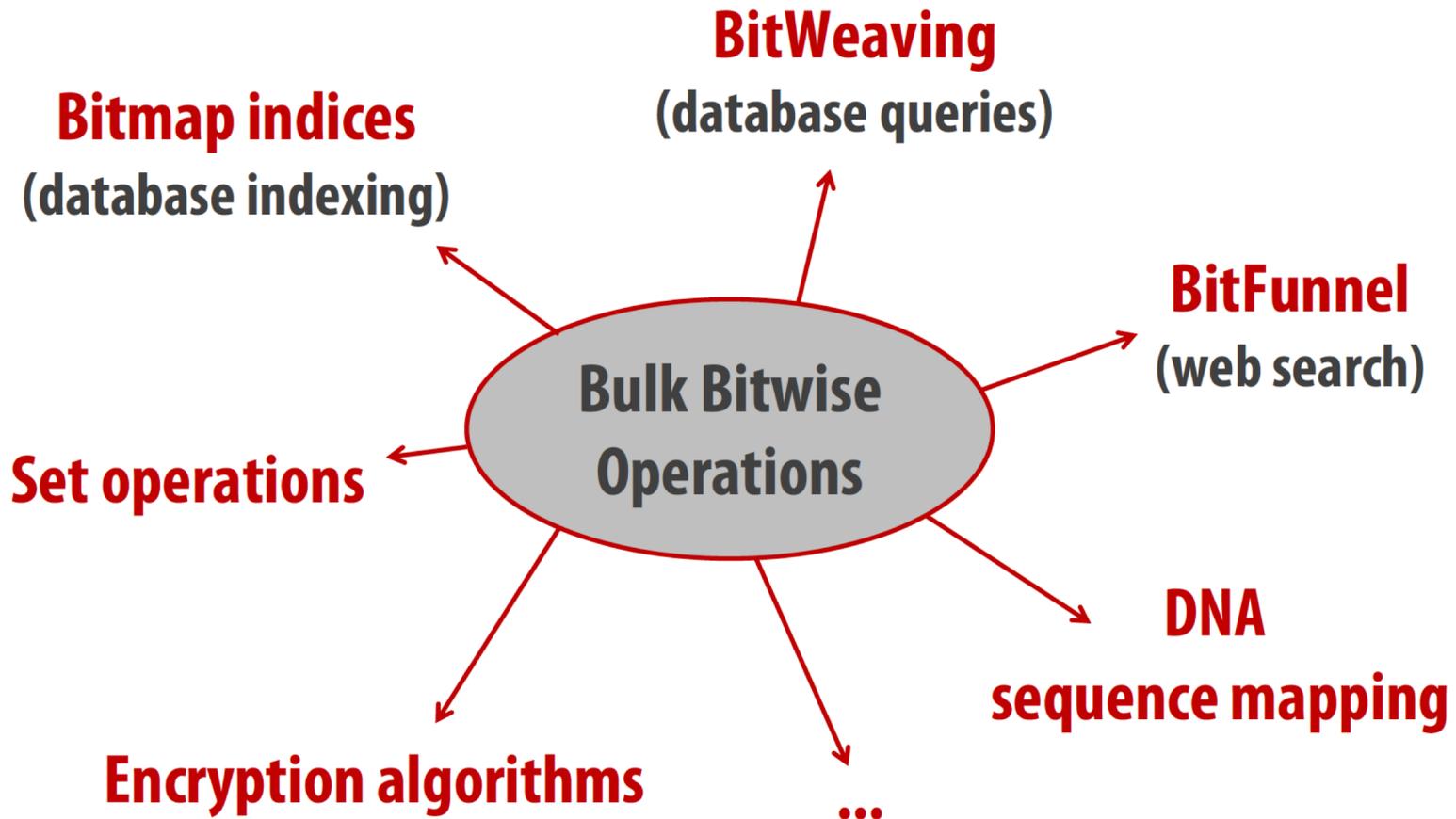
Figure 5: A dual-contact cell connected to both ends of a sense amplifier

Idea:
Feed the
negated value
in the sense amplifier
into a special row

Ambit vs. DDR3: Performance and Energy



Bulk Bitwise Operations in Workloads



Performance: Bitmap Index on Ambit

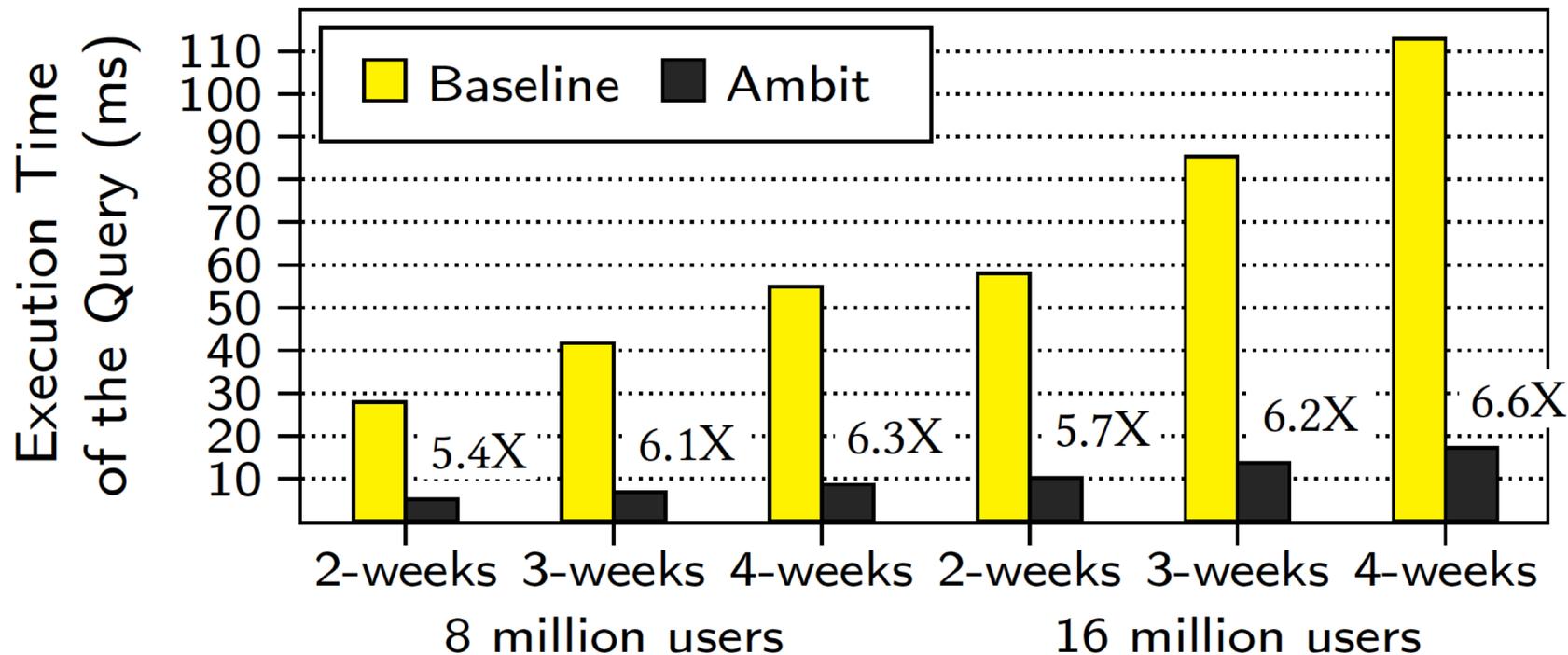


Figure 10: Bitmap index performance. The value above each bar indicates the reduction in execution time due to Ambit.

Seshadri+, "Ambit: In-Memory Accelerator for Bulk Bitwise Operations using Commodity DRAM Technology," MICRO 2017.

Performance: BitWeaving on Ambit

```
'select count(*) from T where c1 <= val <= c2'
```

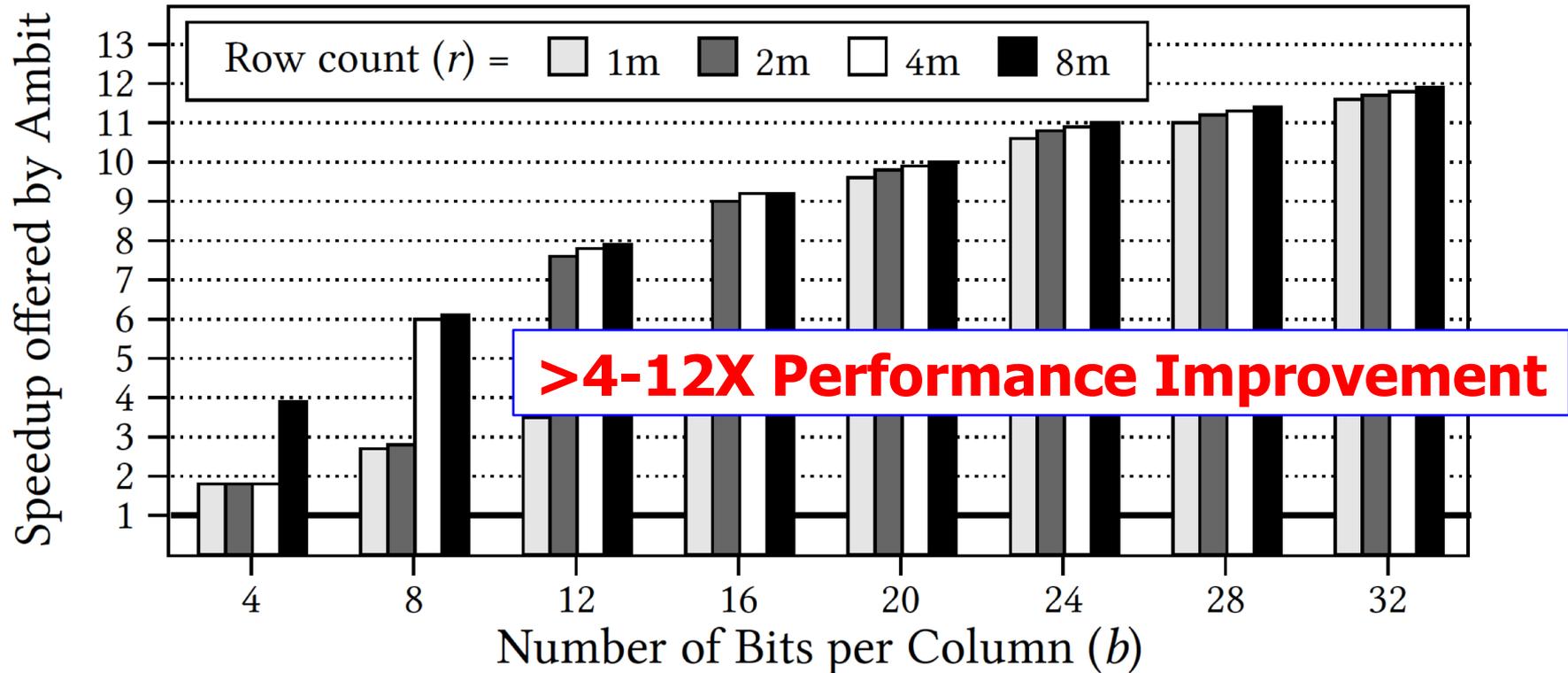


Figure 11: Speedup offered by Ambit over baseline CPU with SIMD for BitWeaving

Seshadri+, "Ambit: In-Memory Accelerator for Bulk Bitwise Operations using Commodity DRAM Technology," MICRO 2017.

More on Ambit

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

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 Hassan⁴ Amirali Boroumand⁵
Jeremie Kim^{4,5} Michael A. Kozuch³ Onur Mutlu^{4,5} Phillip B. Gibbons⁵ Todd C. Mowry⁵

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

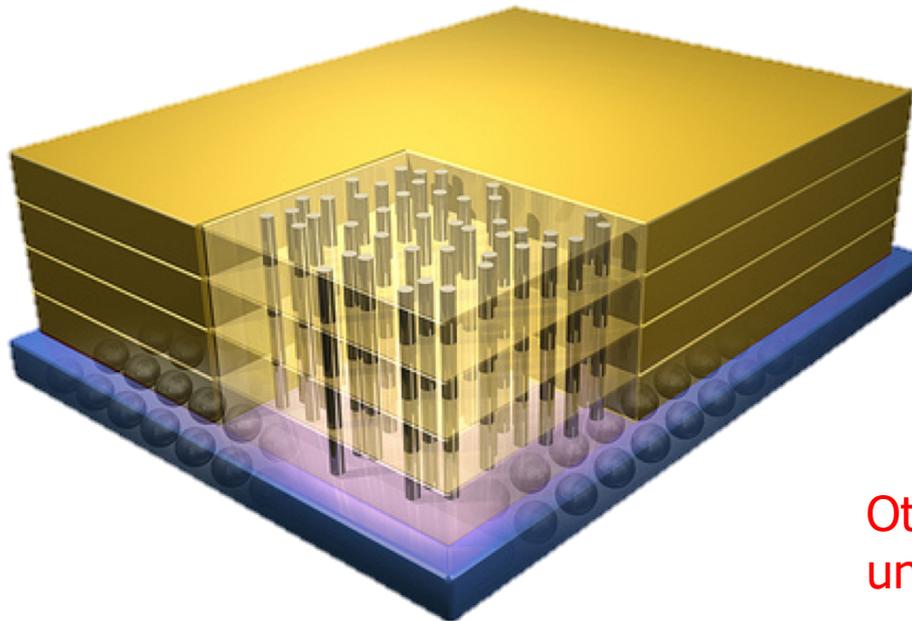
Processing in Memory: Two Approaches

1. Minimally changing memory chips
2. Exploiting 3D-stacked memory

Opportunity: 3D-Stacked Logic+Memory



Hybrid Memory Cube
C O N S O R T I U M



Memory

Logic

Other "True 3D" technologies
under development

DRAM Landscape (circa 2015)

<i>Segment</i>	<i>DRAM Standards & Architectures</i>
Commodity	DDR3 (2007) [14]; DDR4 (2012) [18]
Low-Power	LPDDR3 (2012) [17]; LPDDR4 (2014) [20]
Graphics	GDDR5 (2009) [15]
Performance	eDRAM [28], [32]; RLD RAM3 (2011) [29]
3D-Stacked	WIO (2011) [16]; WIO2 (2014) [21]; MCDRAM (2015) [13]; HBM (2013) [19]; HMC1.0 (2013) [10]; HMC1.1 (2014) [11]
Academic	SBA/SSA (2010) [38]; Staged Reads (2012) [8]; RAIDR (2012) [27]; SALP (2012) [24]; TL-DRAM (2013) [26]; RowClone (2013) [37]; Half-DRAM (2014) [39]; Row-Buffer Decoupling (2014) [33]; SARP (2014) [6]; AL-DRAM (2015) [25]

Table 1. Landscape of DRAM-based memory

Kim+, "Ramulator: A Flexible and Extensible DRAM Simulator", IEEE CAL 2015.

Two Key Questions in 3D-Stacked PIM

- What are the performance and energy benefits of using 3D-stacked memory as a coarse-grained accelerator?
 - By changing the entire system
 - By performing simple function offloading

- What is the minimal processing-in-memory support we can provide?
 - With minimal changes to system and programming

Graph Processing

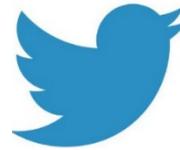
- Large graphs are everywhere (circa 2015)



36 Million
Wikipedia Pages



1.4 Billion
Facebook Users

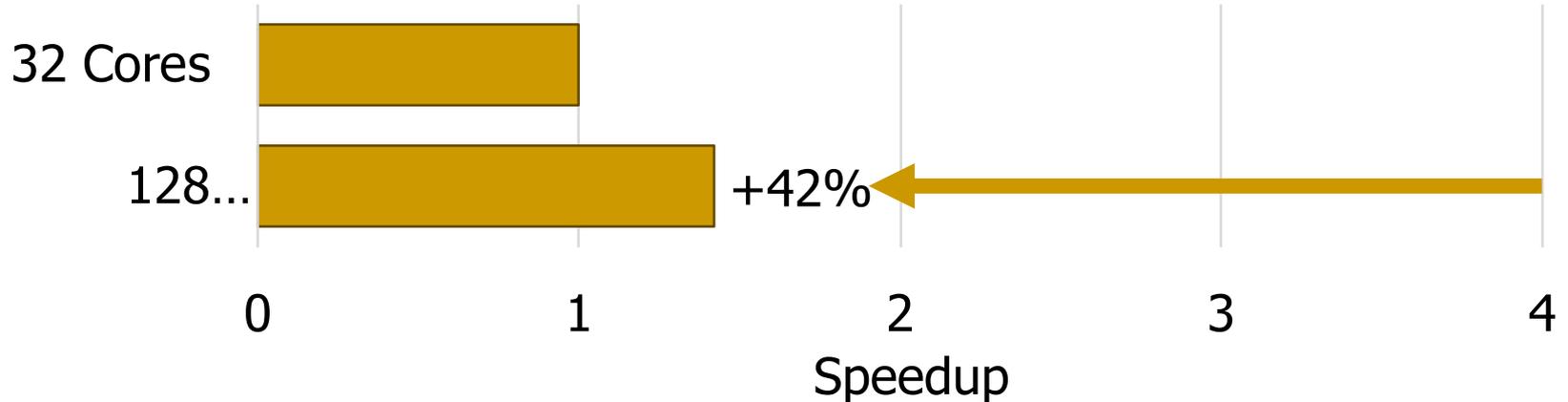


300 Million
Twitter Users



30 Billion
Instagram Photos

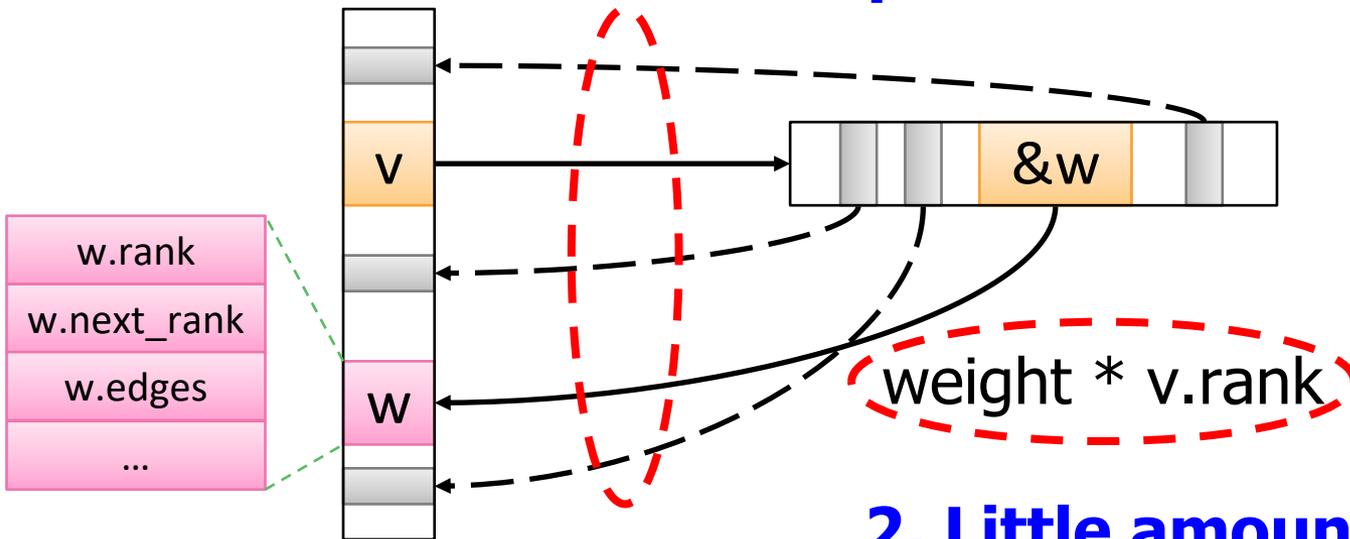
- Scalable large-scale graph processing is challenging



Key Bottlenecks in Graph Processing

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

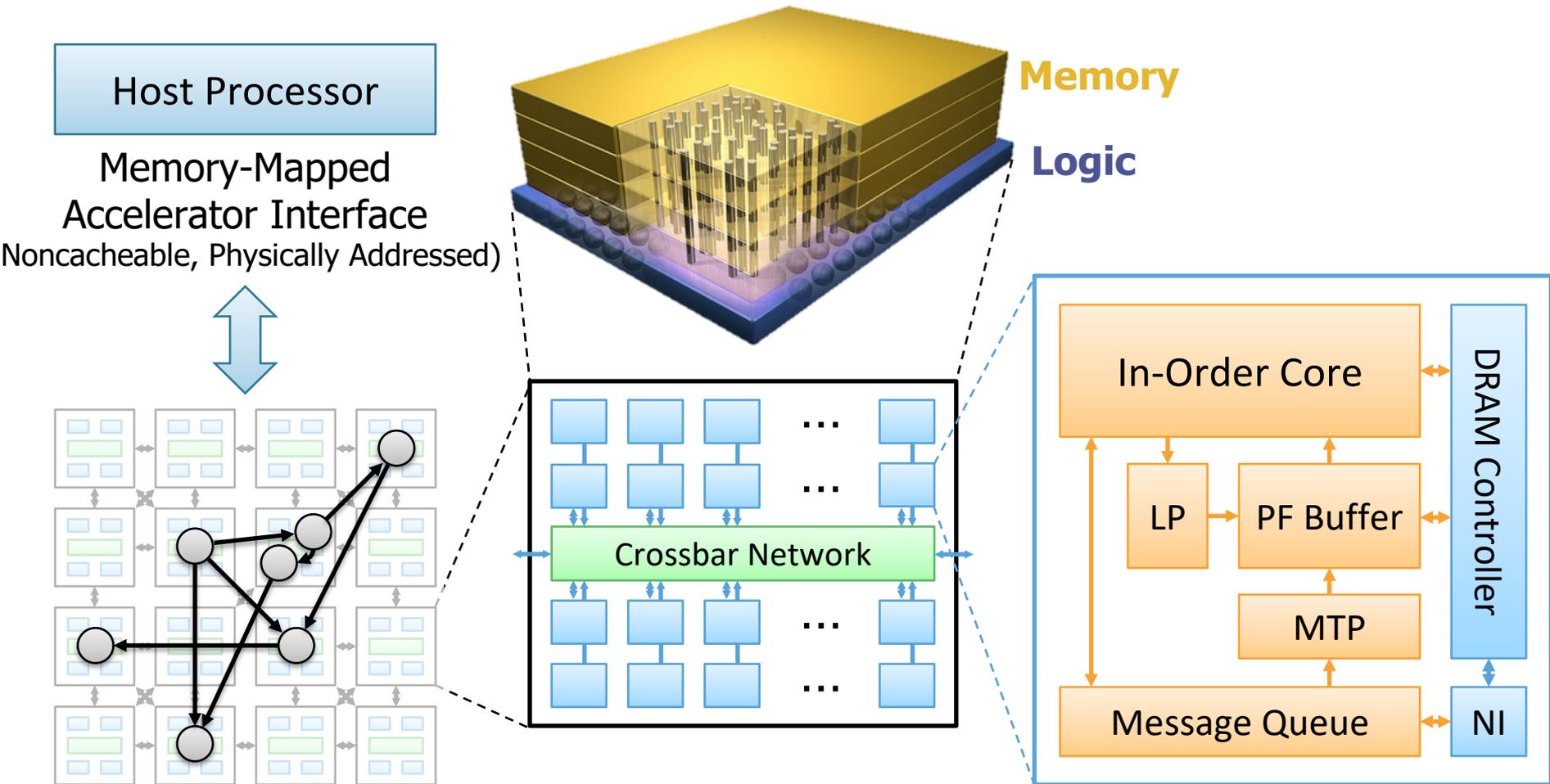
1. Frequent random memory accesses



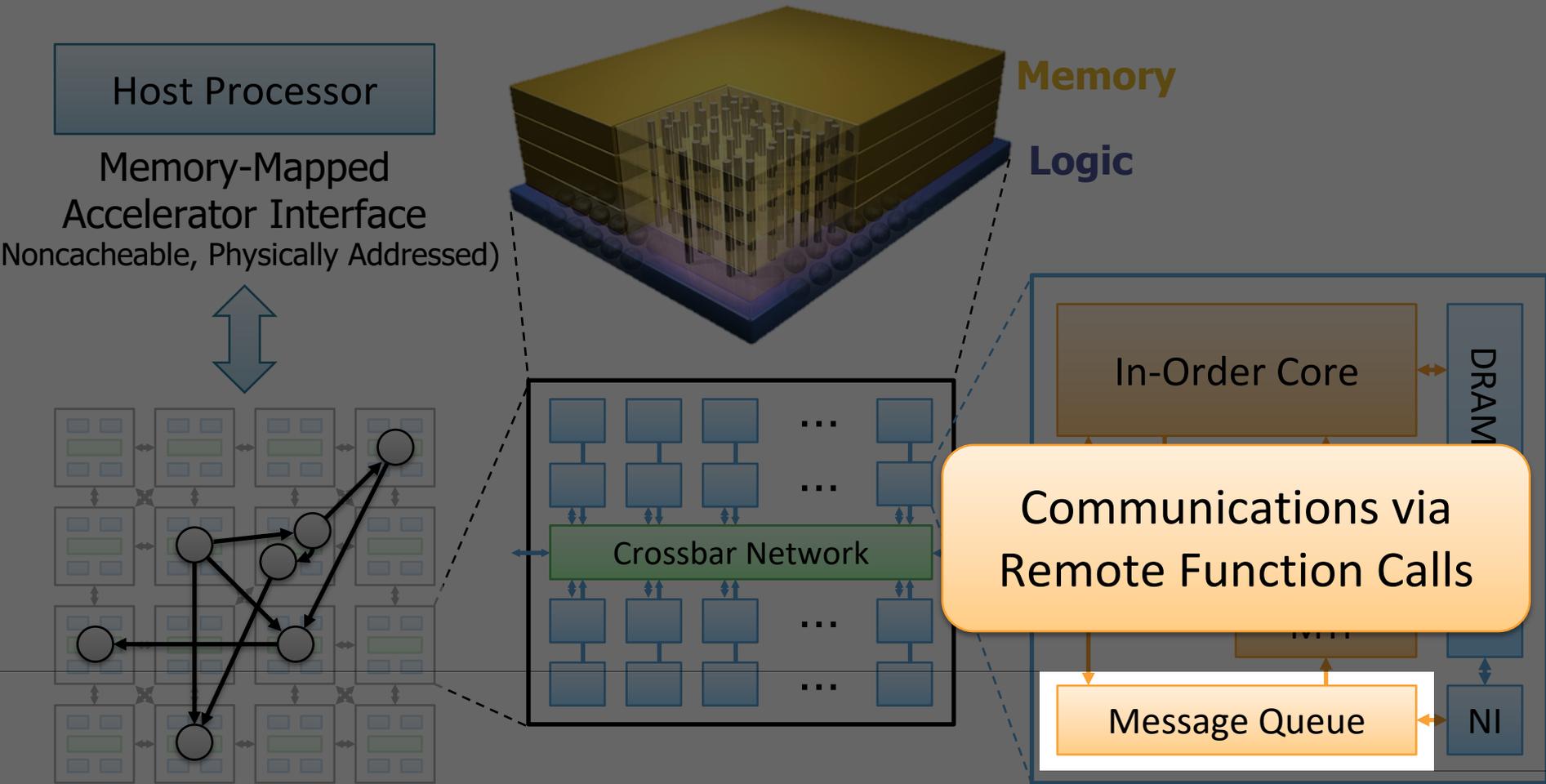
2. Little amount of computation

Tesseract System for Graph Processing

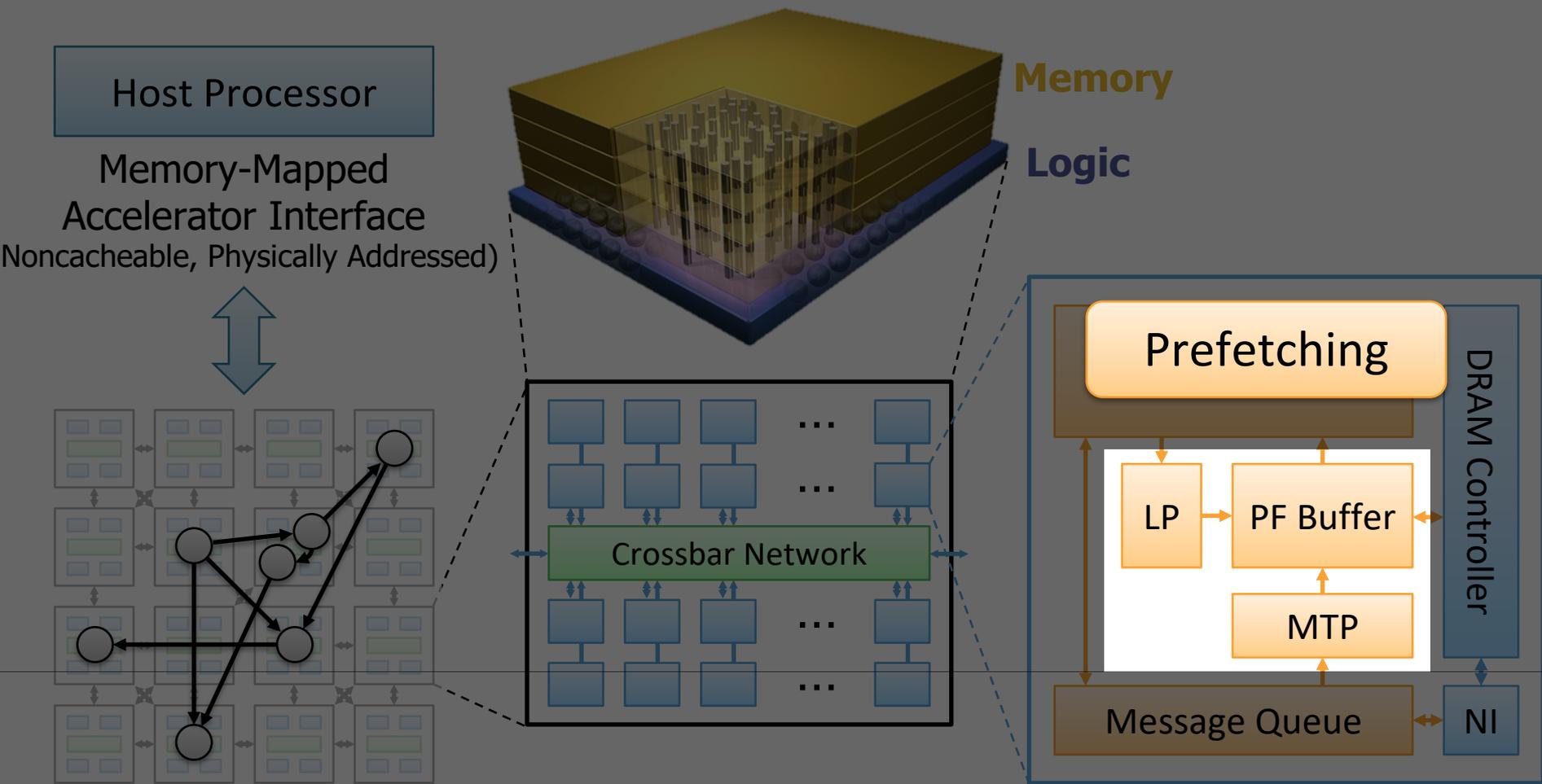
Interconnected set of 3D-stacked memory+logic chips with simple cores



Tesseract System for Graph Processing

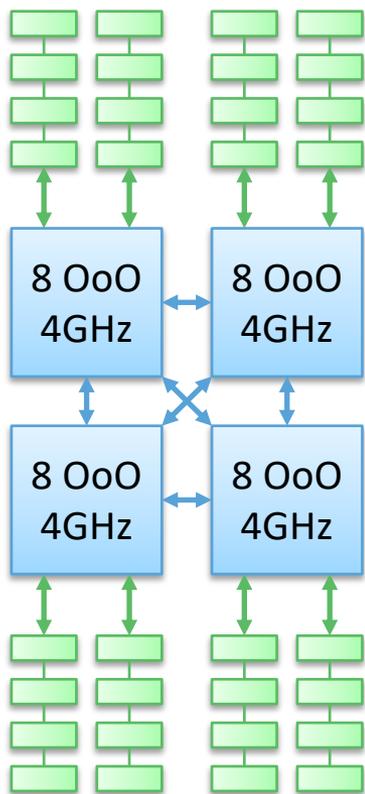


Tesseract System for Graph Processing



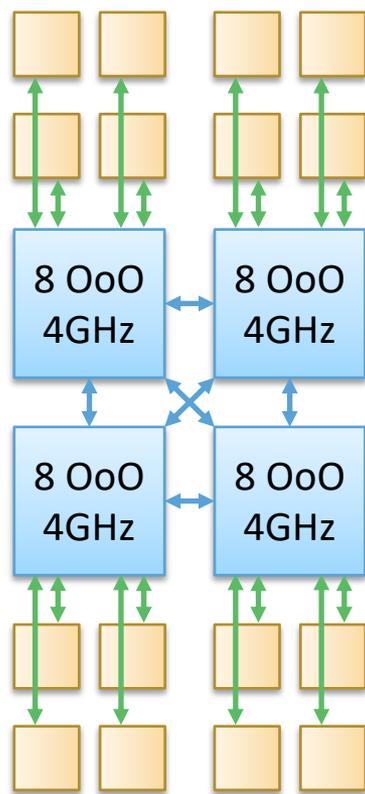
Evaluated Systems

DDR3-OoO



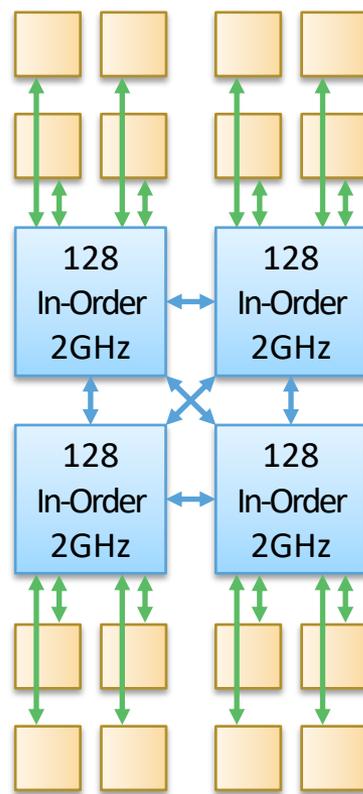
102.4GB/s

HMC-OoO



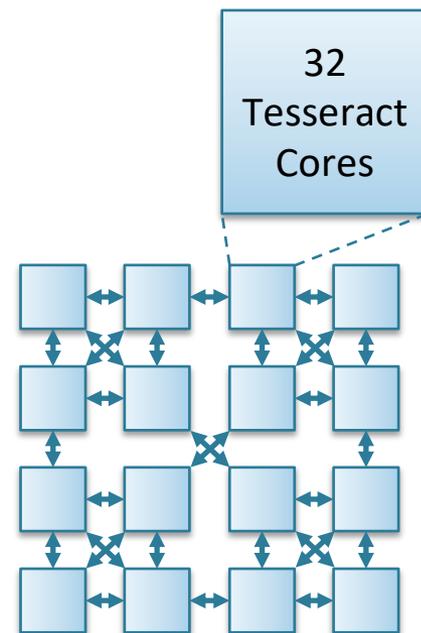
640GB/s

HMC-MC



640GB/s

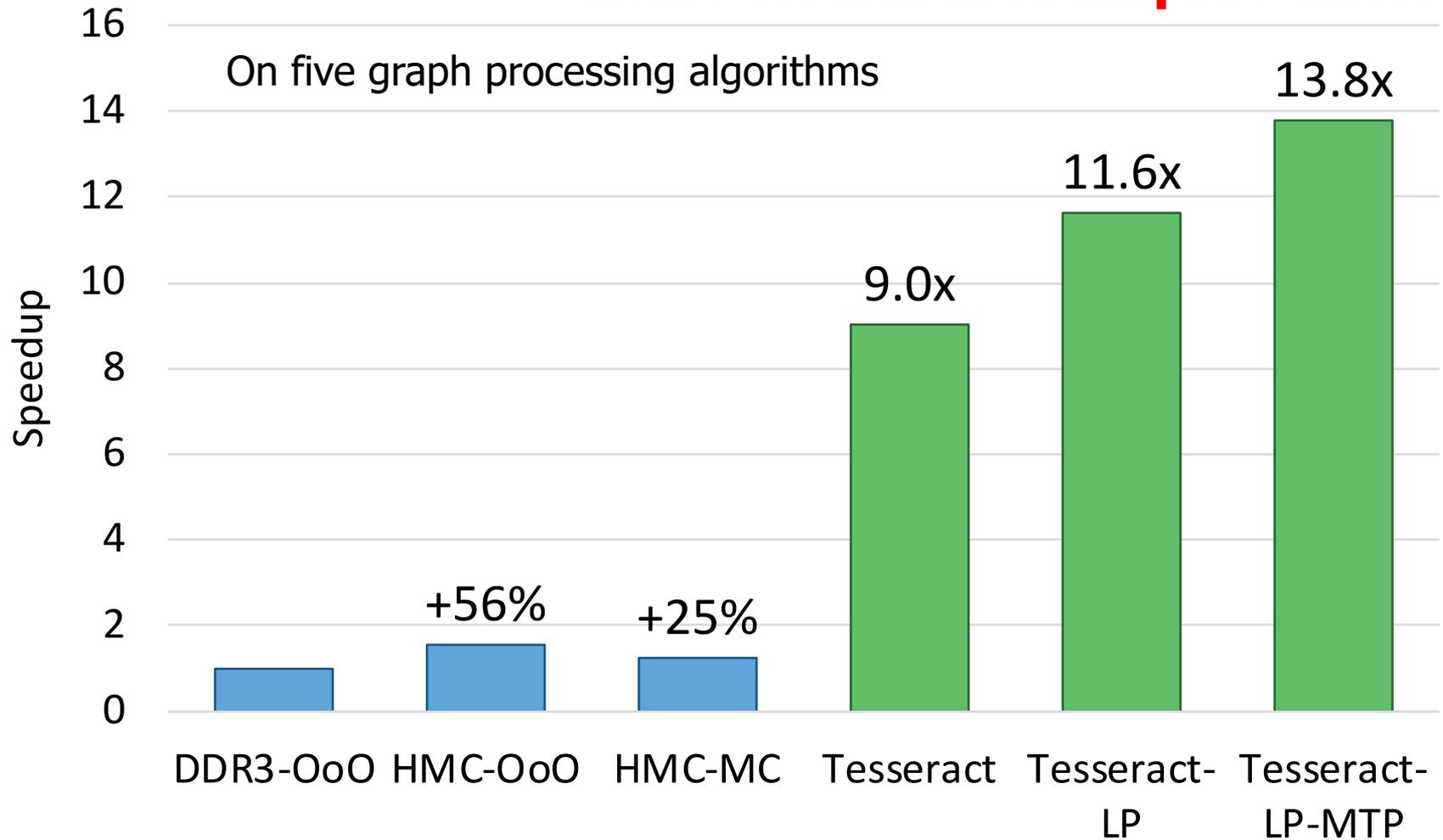
Tesseract



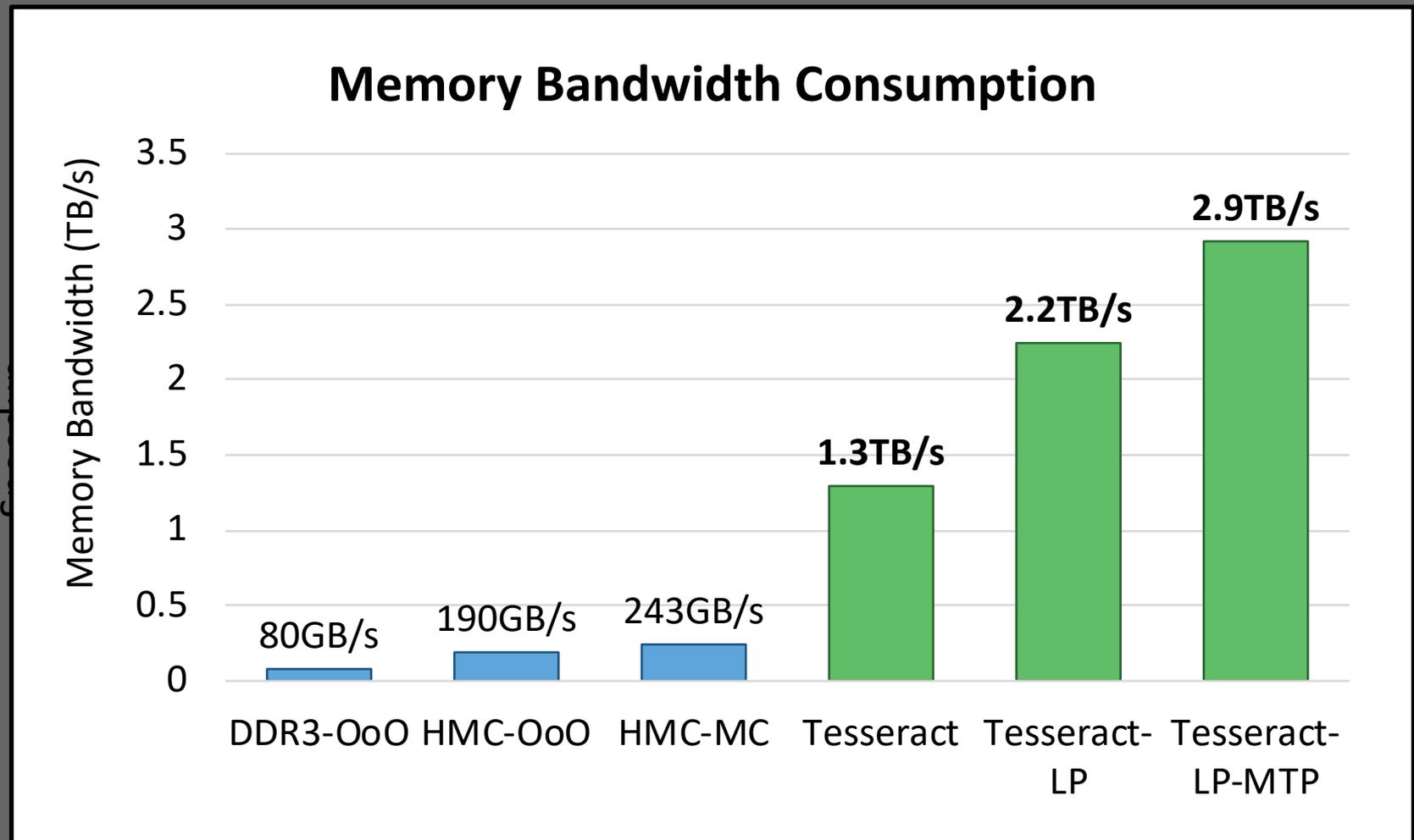
8TB/s

Tesseract Graph Processing Performance

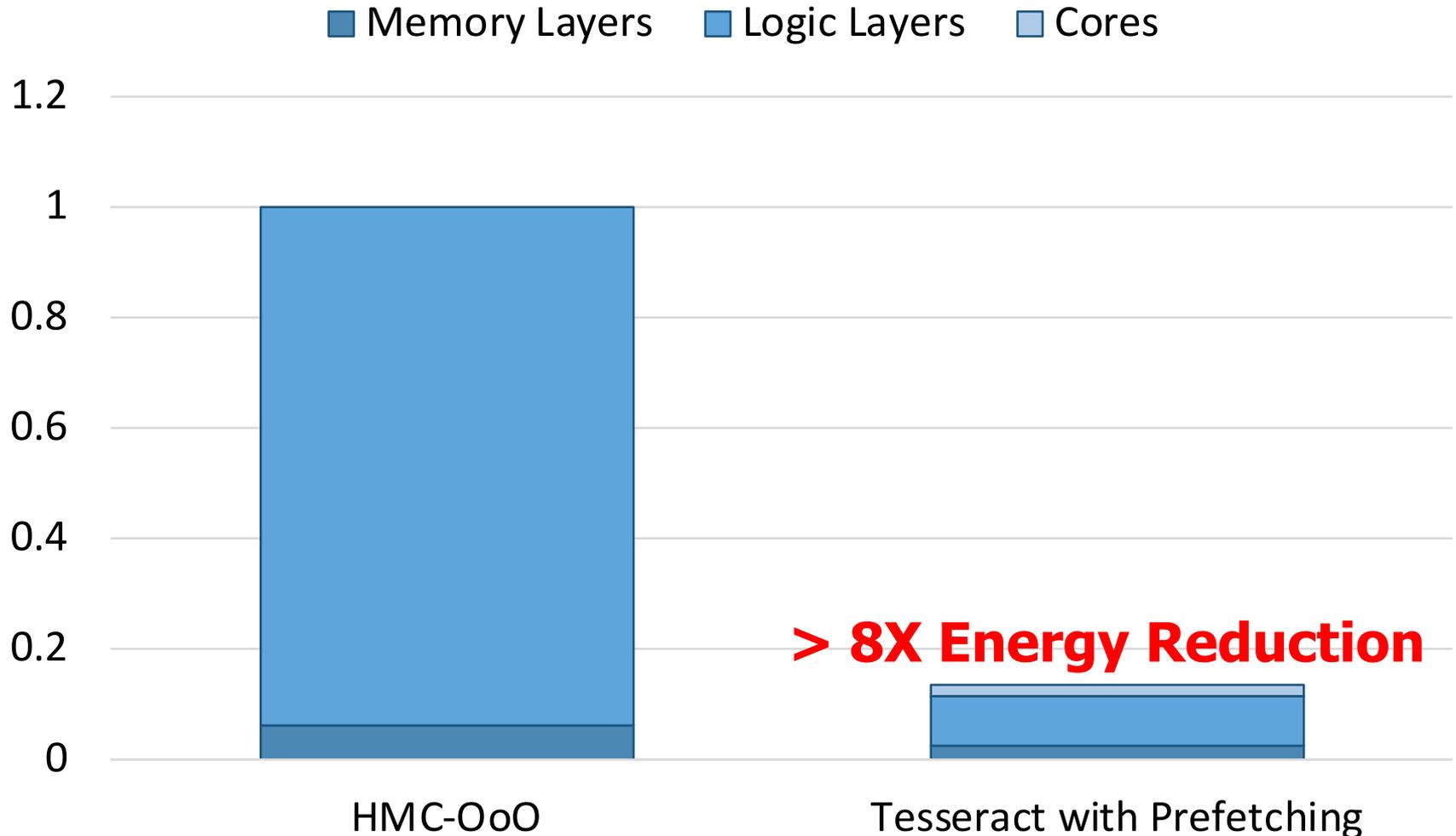
>13X Performance Improvement



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 \(pdf\)\]](#) [\[Lightning Session Slides \(pdf\)\]](#)

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

Two Key Questions in 3D-Stacked PIM

- What are the performance and energy benefits of using 3D-stacked memory as a coarse-grained accelerator?
 - By changing the entire system
 - By performing simple function offloading

- What is the minimal processing-in-memory support we can provide?
 - With minimal changes to system and programming

PIM on Mobile Devices

- 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.

Google Workloads for Consumer Devices: Mitigating Data Movement Bottlenecks

Amirali Boroumand¹

Saugata Ghose¹

Youngsok Kim²

Rachata Ausavarungnirun¹

Eric Shiu³

Rahul Thakur³

Daehyun Kim^{4,3}

Aki Kuusela³

Allan Knies³

Parthasarathy Ranganathan³

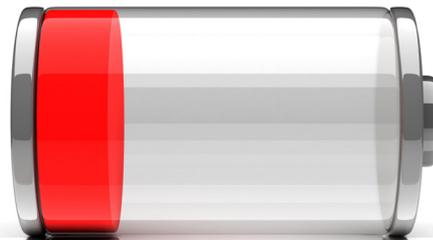
Onur Mutlu^{5,1}

Consumer Devices



Consumer devices are everywhere!

**Energy consumption is
a first-class concern in consumer devices**



Popular Google Consumer Workloads



Chrome

Google's web browser



TensorFlow Mobile

Google's machine learning framework

VP9



Video Playback

Google's **video codec**

VP9

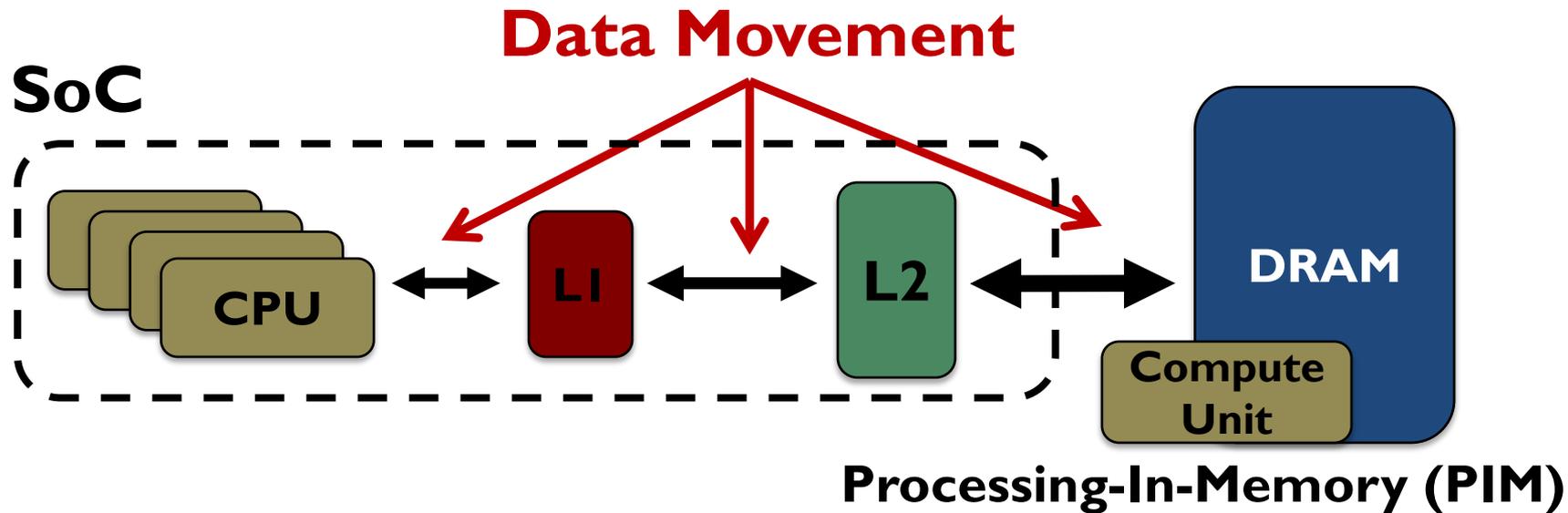


Video Capture

Google's **video codec**

Energy Cost of Data Movement

1st key observation: **62.7%** of the total system energy is spent on **data movement**



Potential solution: move computation **close to data**

Challenge: limited area and energy budget

Using PIM to Reduce Data Movement

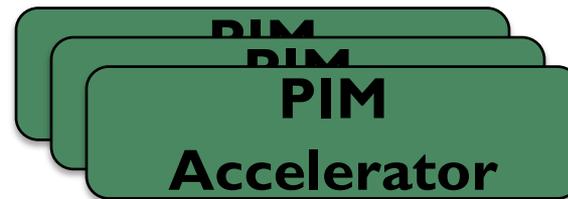
2nd key observation: a significant fraction of the **data movement** often comes from **simple functions**

We can design lightweight logic to implement these simple functions in **memory**

Small embedded
low-power core



Small fixed-function
accelerators



Offloading to PIM logic reduces energy and improves performance, on average, by 55.4% and 54.2%

Workload Analysis



Chrome

Google's web browser



TensorFlow Mobile

Google's machine learning
framework

VP9



Video Playback

Google's **video codec**

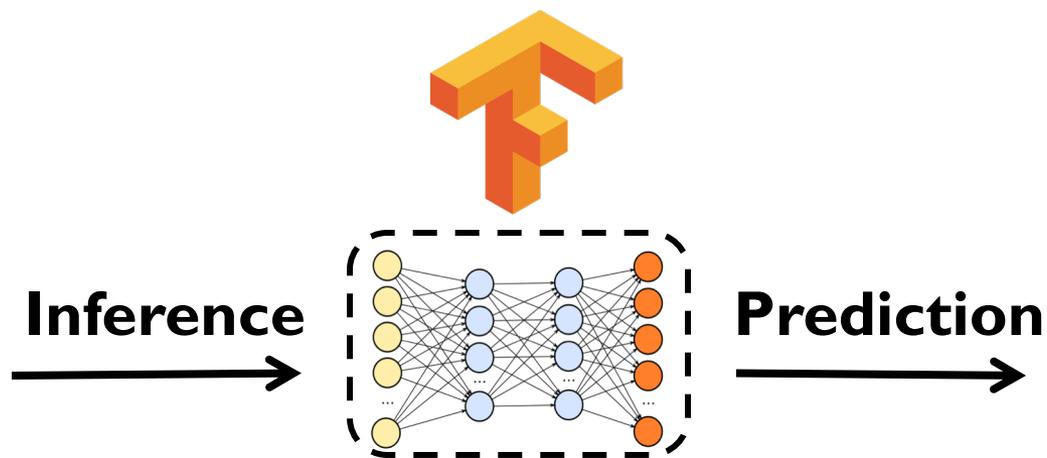
VP9



Video Capture

Google's **video codec**

TensorFlow Mobile



57.3% of the inference energy is spent on data movement



54.4% of the data movement energy comes from packing/unpacking and quantization

Packing



Reorders elements of matrices to minimize **cache misses** during **matrix multiplication**



Up to **40%** of the inference **energy** and **31%** of inference **execution time**



Packing's **data movement** accounts for up to **35.3%** of the inference **energy**

A simple **data reorganization** process that requires **simple arithmetic**

Quantization



Converts 32-bit floating point to 8-bit integers to improve inference execution time and energy consumption



Up to **16.8%** of the inference **energy** and **16.1%** of inference **execution time**

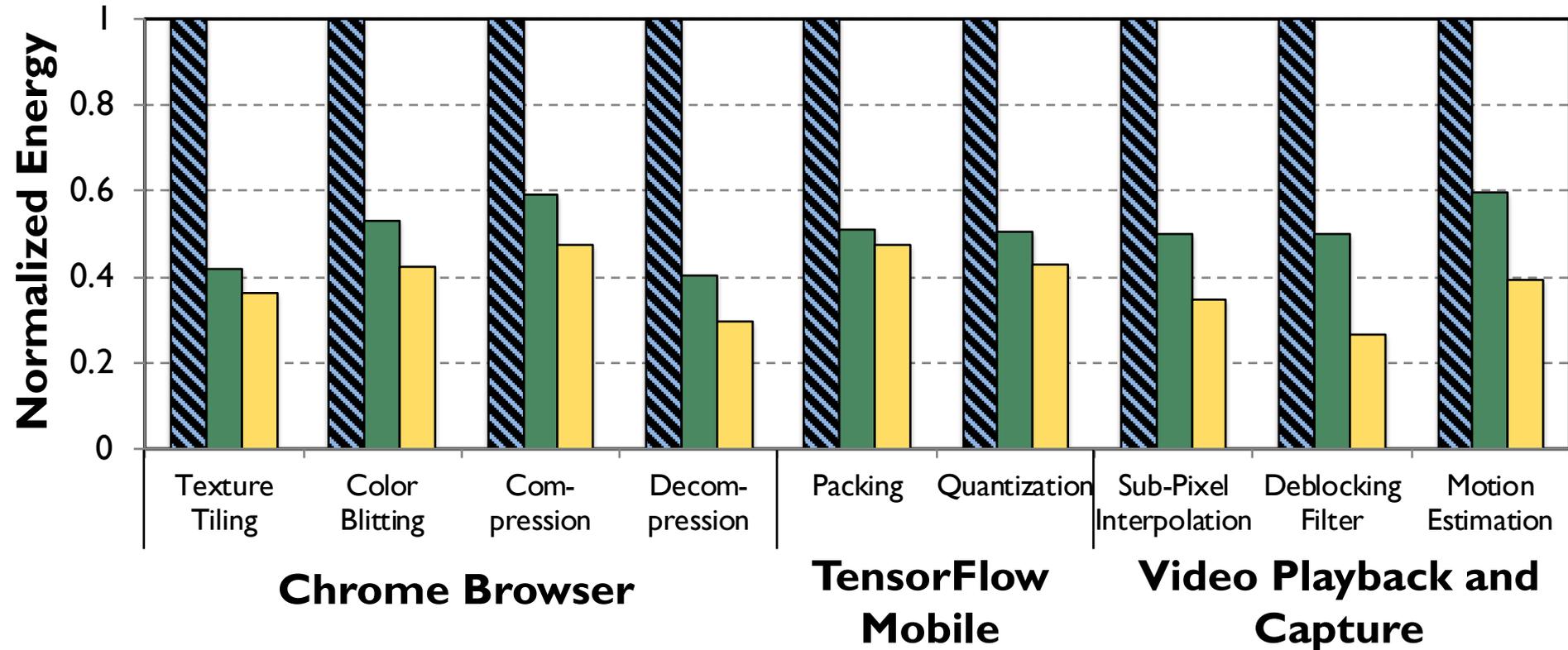


Majority of **quantization** energy comes from **data movement**

A simple **data conversion** operation that requires **shift, addition, and multiplication** operations

Normalized Energy

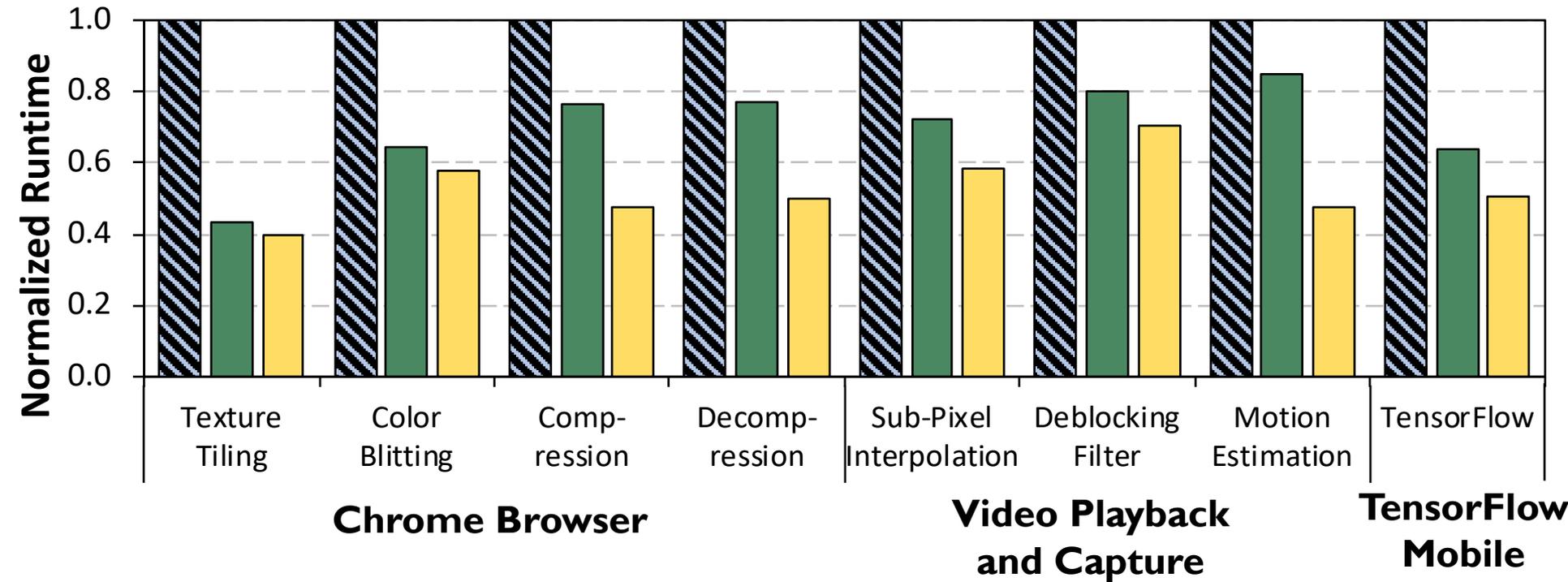
■ CPU-Only ■ PIM-Core ■ PIM-Acc



PIM core and **PIM accelerator** reduce **energy consumption** on average by **49.1%** and **55.4%**

Normalized Runtime

▨ CPU-Only █ PIM-Core █ PIM-Acc



Offloading these kernels to **PIM core** and **PIM accelerator** improves **performance** on average by **44.6%** and **54.2%**

More on PIM for Mobile Devices

- 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 Boroumand¹

Saugata Ghose¹

Youngsok Kim²

Rachata Ausavarungnirun¹

Eric Shiu³

Rahul Thakur³

Daehyun Kim^{4,3}

Aki Kuusela³

Allan Knies³

Parthasarathy Ranganathan³

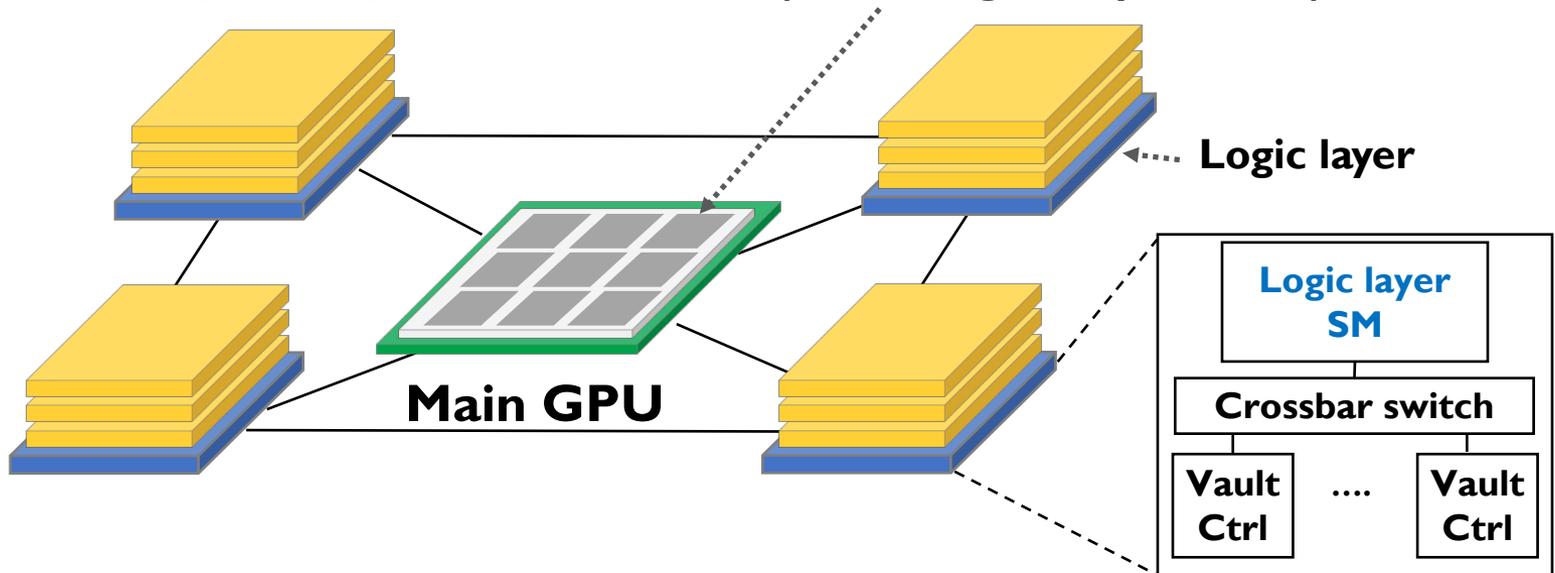
Onur Mutlu^{5,1}

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;
```

**3D-stacked memory
(memory stack)**

SM (Streaming Multiprocessor)



Accelerating GPU Execution with PIM (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

Accelerating GPU Execution with PIM (II)

- Ashutosh Pattnaik, Xulong Tang, Adwait Jog, Onur Kayiran, Asit K. Mishra, Mahmut T. Kandemir, Onur Mutlu, and Chita R. Das, **"Scheduling Techniques for GPU Architectures with Processing-In-Memory Capabilities"**
Proceedings of the 25th International Conference on Parallel Architectures and Compilation Techniques (PACT), Haifa, Israel, September 2016.

Scheduling Techniques for GPU Architectures with Processing-In-Memory Capabilities

Ashutosh Pattnaik¹ Xulong Tang¹ Adwait Jog² Onur Kayiran³
Asit K. Mishra⁴ Mahmut T. Kandemir¹ Onur Mutlu^{5,6} Chita R. Das¹

¹Pennsylvania State University ²College of William and Mary
³Advanced Micro Devices, Inc. ⁴Intel Labs ⁵ETH Zürich ⁶Carnegie Mellon University

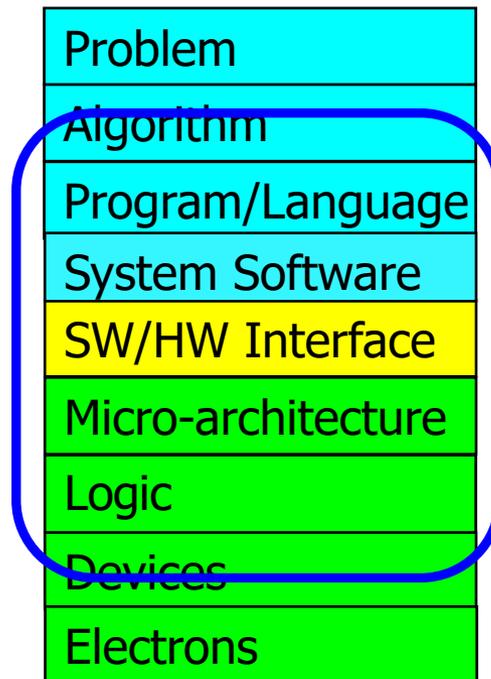
Eliminating the Adoption Barriers

How to Enable Adoption of Processing in Memory

Barriers to Adoption of PIM

1. Functionality of and applications for PIM
2. Ease of programming (interfaces and compiler/HW support)
3. System support: coherence & virtual memory
4. Runtime systems for adaptive scheduling, data mapping, access/sharing control
5. Infrastructures to assess benefits and feasibility

We Need to Revisit the Entire Stack



Open Problems: PIM Adoption

Enabling the Adoption of Processing-in-Memory: Challenges, Mechanisms, Future Research Directions

SAUGATA GHOSE, KEVIN HSIEH, AMIRALI BOROUMAND,
RACHATA AUSAVARUNGNIRUN

Carnegie Mellon University

ONUR MUTLU

ETH Zürich and Carnegie Mellon University

Saugata Ghose, Kevin Hsieh, Amirali Boroumand, Rachata Ausavarungnirun, Onur Mutlu,
**"Enabling the Adoption of Processing-in-Memory: Challenges, Mechanisms,
Future Research Directions"**

Invited Book Chapter, to appear in 2018.

[[Preliminary arxiv.org version](https://arxiv.org/abs/1802.00320)]

PIM Review and Open Problems

Processing Data Where It Makes Sense: Enabling In-Memory Computation

Onur Mutlu^{a,b}, Saugata Ghose^b, Juan Gómez-Luna^a, Rachata Ausavarungnirun^{b,c}

^a*ETH Zürich*

^b*Carnegie Mellon University*

^c*King Mongkut's University of Technology North Bangkok*

Onur Mutlu, Saugata Ghose, Juan Gomez-Luna, and Rachata Ausavarungnirun,
["Processing Data Where It Makes Sense: Enabling In-Memory Computation"](#)

Invited paper in [Microprocessors and Microsystems \(MICPRO\)](#), June 2019.
[\[arXiv version\]](#)

Computing Architectures with Minimal Data Movement

Corollaries: Architectures Today ...

- Architectures are **terrible at dealing with data**
 - Designed to mainly store and move data vs. to compute
 - They are **processor-centric** as opposed to **data-centric**
- Architectures are **terrible at taking advantage of vast amounts of data** (and metadata) available to them
 - Designed to make simple decisions, ignoring lots of data
 - They make **human-driven decisions** vs. **data-driven** decisions
- Architectures are **terrible at knowing and exploiting different properties of application data**
 - Designed to treat all data as the same
 - They make **component-aware decisions** vs. **data-aware**

Exploiting Data to Design Intelligent 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?**

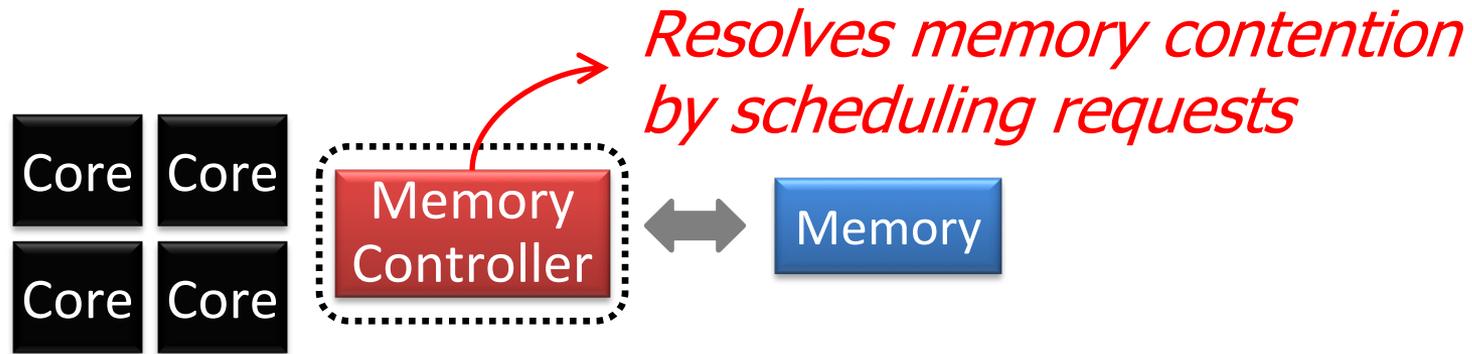
An Intelligent Architecture

- 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

How do we start?

Self-Optimizing Memory Controllers

Memory Controller



How to schedule requests to maximize system performance?

Why are Memory 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 Memory Timing Constraints

Latency	Symbol	DRAM cycles	Latency	Symbol	DRAM cycles
Precharge	t_{RP}	11	Activate to read/write	t_{RCD}	11
Read column address strobe	CL	11	Write column address strobe	CWL	8
Additive	AL	0	Activate to activate	t_{RC}	39
Activate to precharge	t_{RAS}	28	Read to precharge	t_{RTP}	6
Burst length	t_{BL}	4	Column address strobe to column address strobe	t_{CCD}	4
Activate to activate (different bank)	t_{RRD}	6	Four activate windows	t_{FAW}	24
Write to read	t_{WTR}	6	Write recovery	t_{WR}	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.

Many Memory Timing Constraints

- 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.

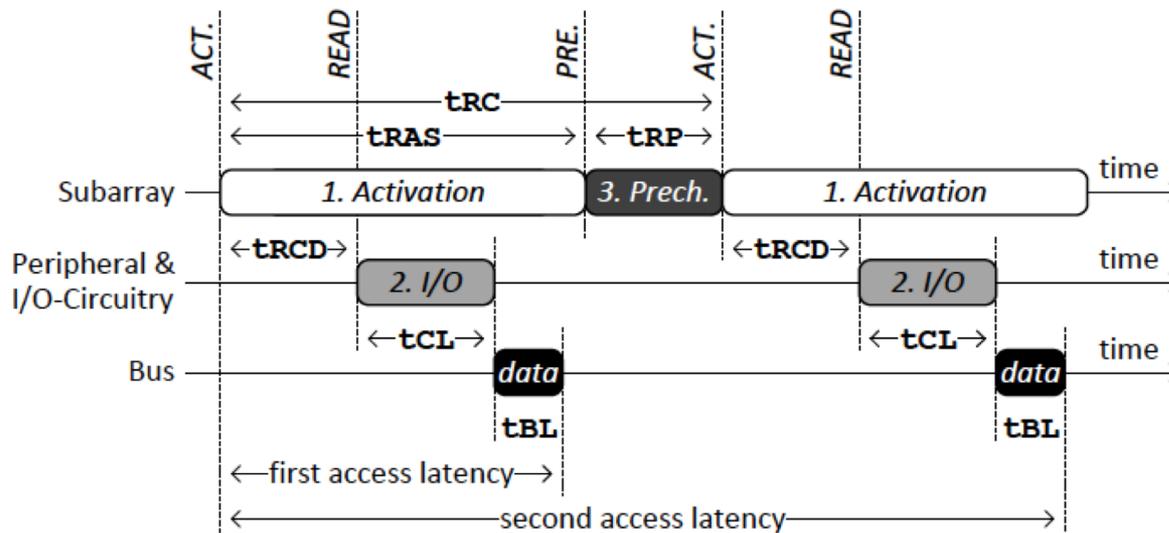
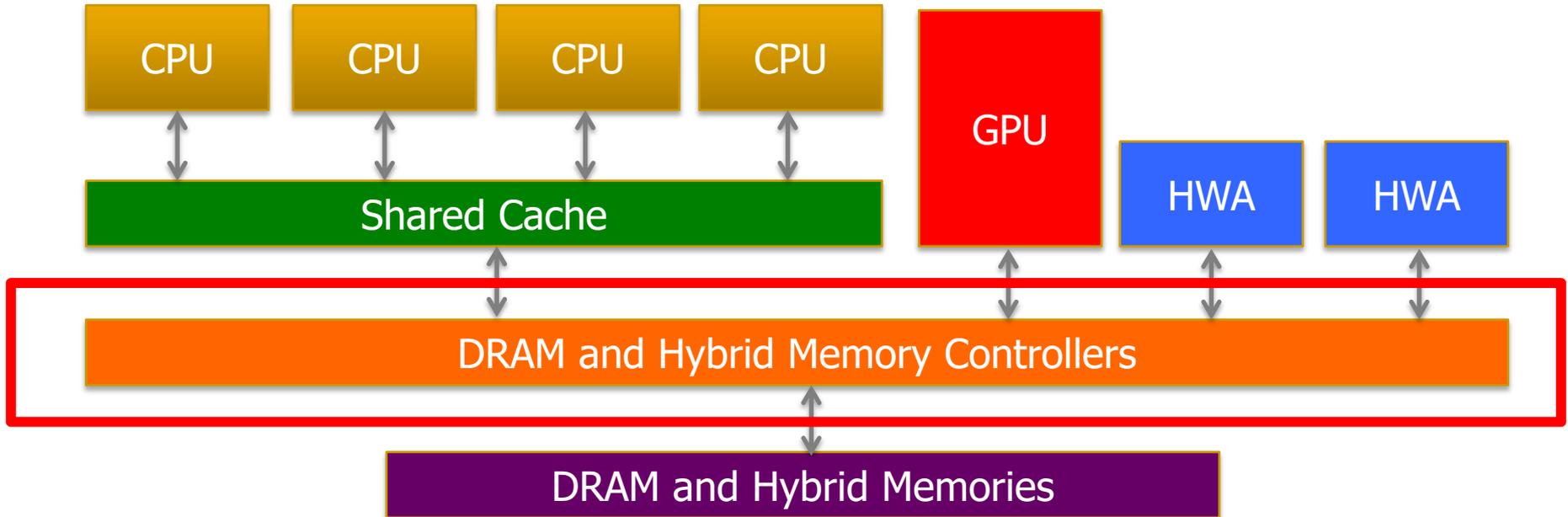


Figure 5. Three Phases of DRAM Access

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

Phase	Commands	Name	Value
1	ACT → READ	t_{RCD}	15ns
	ACT → WRITE		
	ACT → PRE	t_{RAS}	37.5ns
2	READ → data	t_{CL}	15ns
	WRITE → data	t_{CWL}	11.25ns
	data burst	t_{BL}	7.5ns
3	PRE → ACT	t_{RP}	15ns
1 & 3	ACT → ACT	t_{RC} ($t_{RAS}+t_{RP}$)	52.5ns

Memory Controller Design Is Becoming More Difficult



- Heterogeneous agents: CPUs, GPUs, and HWAs
- Main memory interference between CPUs, GPUs, HWAs
- Many timing constraints for various memory types
- Many goals at the same time: performance, fairness, QoS, energy efficiency, ...

Reality and Dream

- Reality: It 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

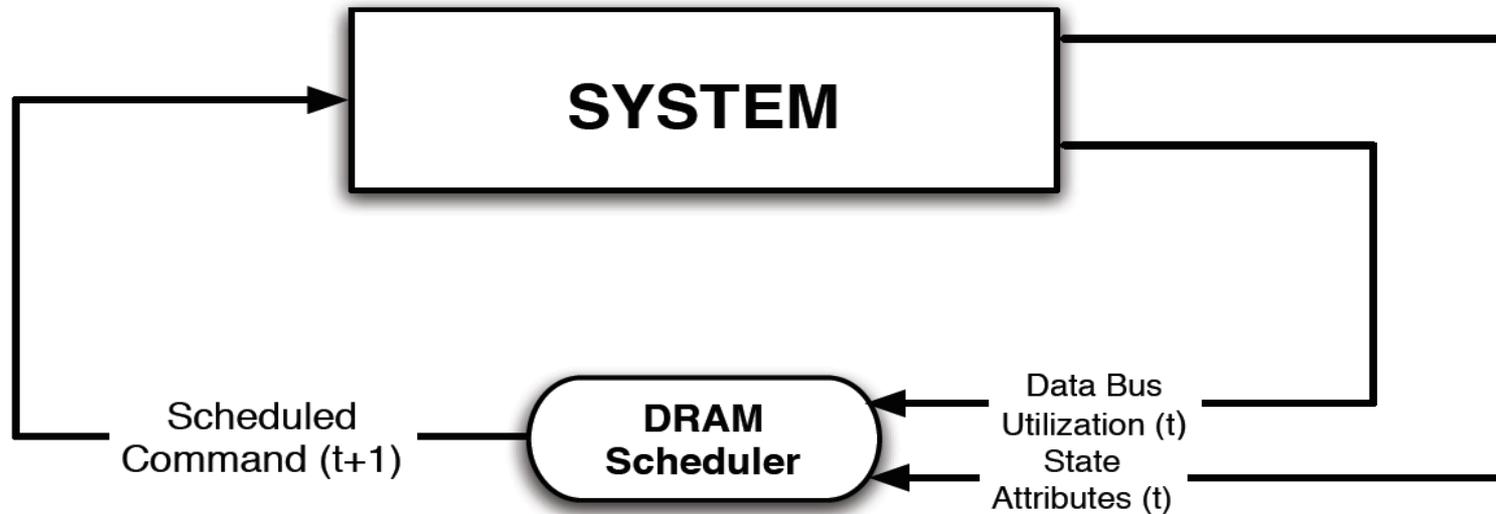


Goal: Learn to choose actions to maximize $r_0 + \gamma r_1 + \gamma^2 r_2 + \dots$ ($0 \leq \gamma < 1$)

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 $\langle \text{state}, \text{action} \rangle$ 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.

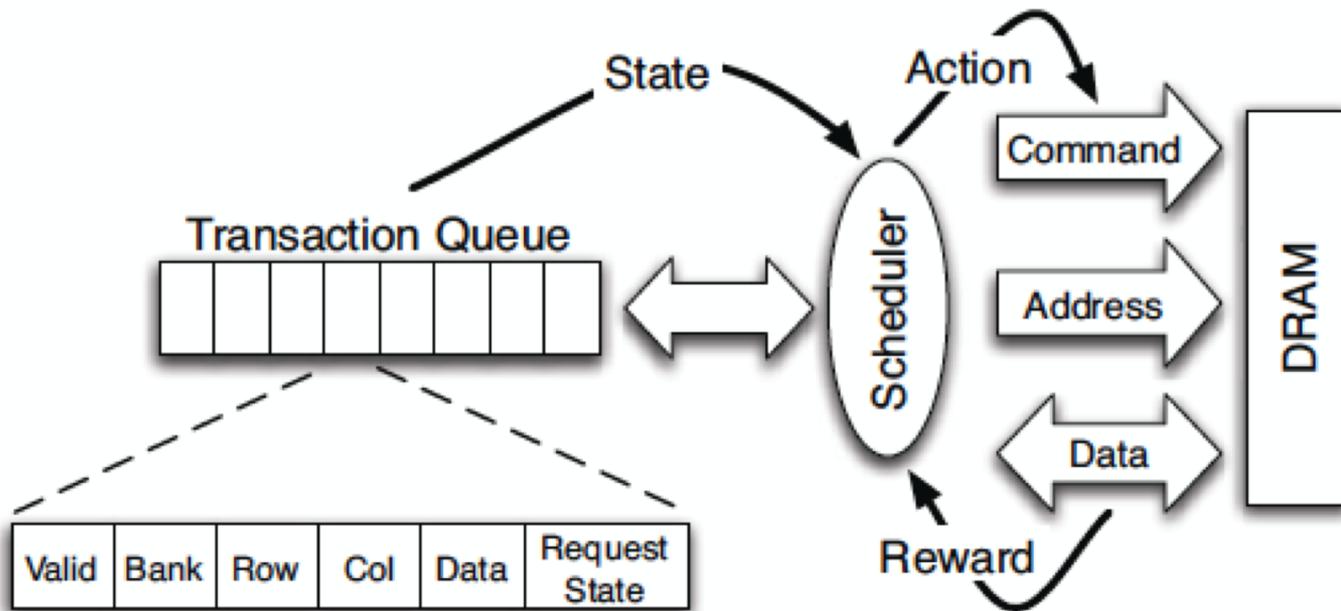


Figure 4: High-level overview of an RL-based scheduler.

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

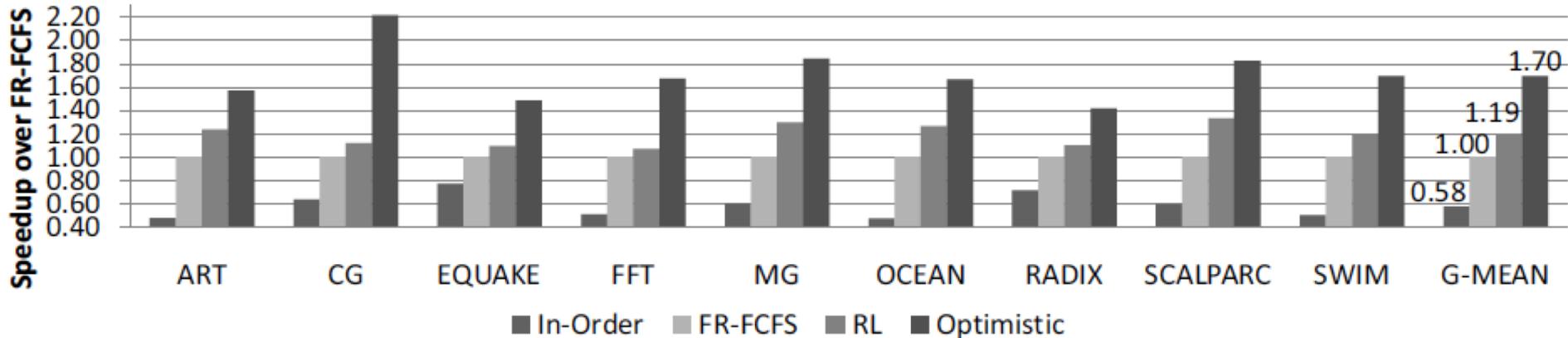


Figure 7: Performance comparison of in-order, FR-FCFS, RL-based, and optimistic memory controllers

Large, robust performance improvements over many human-designed policies

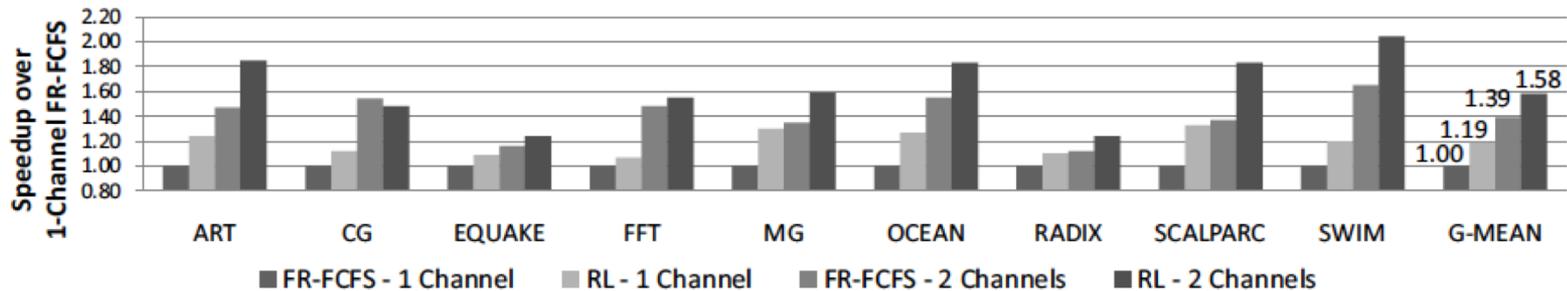


Figure 15: Performance comparison of FR-FCFS and RL-based memory controllers on systems with 6.4GB/s and 12.8GB/s peak DRAM bandwidth

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

- 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 Mutlu² José F. Martínez¹ Rich Caruana¹

¹Cornell University, Ithaca, NY 14850 USA

²Microsoft Research, Redmond, WA 98052 USA

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 (Data-Driven)

Computing Architectures

Corollaries: Architectures Today ...

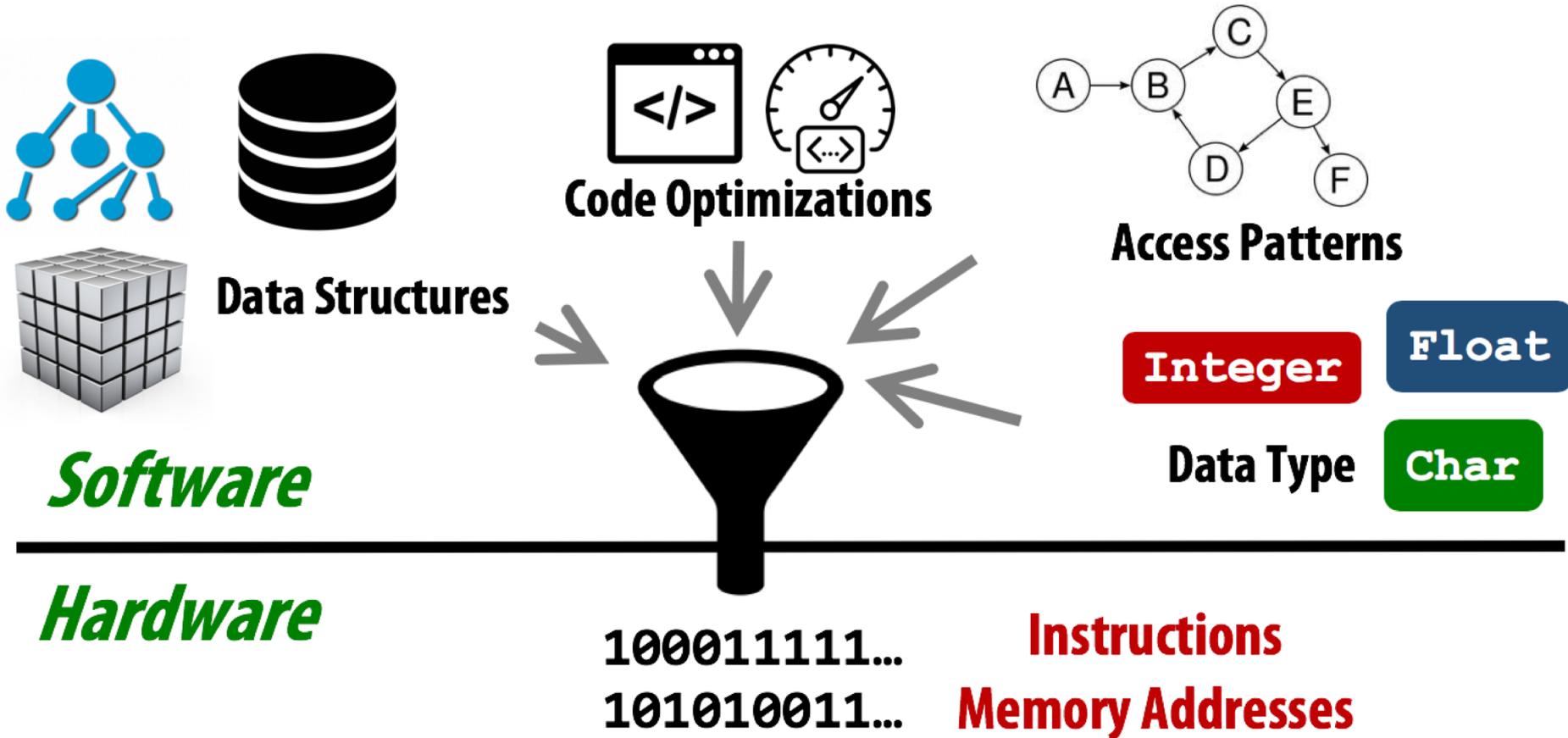
- Architectures are **terrible at dealing with data**
 - Designed to mainly store and move data vs. to compute
 - They are **processor-centric** as opposed to **data-centric**
- Architectures are **terrible at taking advantage of vast amounts of data** (and metadata) available to them
 - Designed to make simple decisions, ignoring lots of data
 - They make **human-driven decisions** vs. **data-driven** decisions
- Architectures are **terrible at knowing and exploiting different properties of application data**
 - Designed to treat all data as the same
 - They make **component-aware decisions** vs. **data-aware**

Data-Aware Architectures

- A data-aware architecture understands what it can do with and to each piece of data
- It makes use of different properties of data to improve performance, efficiency and other metrics
 - Compressibility
 - Approximability
 - Locality
 - Sparsity
 - Criticality for Computation X
 - Access Semantics
 - ...

One Problem: Limited Interfaces

Higher-level information is not visible to HW

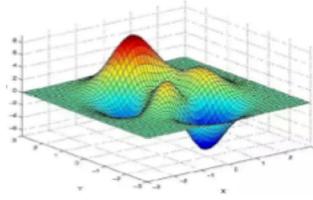
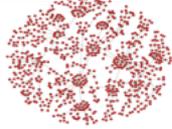


A Solution: More Expressive Interfaces

Performance

Functionality

Software



**ISA
Virtual Memory**

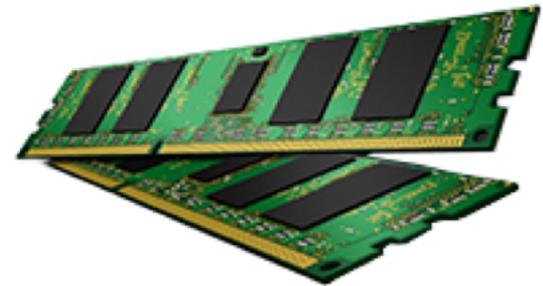
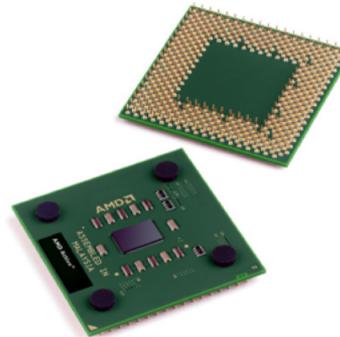
**Higher-level
Program
Semantics**

**Expressive
Memory
"XMem"**

Hardware



wiseGEEK



Expressive (Memory) Interfaces

- Nandita Vijaykumar, Abhilasha Jain, Diptesh Majumdar, Kevin Hsieh, Gennady Pekhimenko, Eiman Ebrahimi, Nastaran Hajinazar, Phillip B. Gibbons and Onur Mutlu, **"A Case for Richer Cross-layer Abstractions: Bridging the Semantic Gap with Expressive Memory"**
Proceedings of the 45th International Symposium on Computer Architecture (ISCA), Los Angeles, CA, USA, June 2018.
[[Slides \(pptx\)](#)] [[pdf](#)] [[Lightning Talk Slides \(pptx\)](#)] [[pdf](#)]
[[Lightning Talk Video](#)]

A Case for Richer Cross-layer Abstractions: Bridging the Semantic Gap with Expressive Memory

Nandita Vijaykumar^{†§} Abhilasha Jain[†] Diptesh Majumdar[†] Kevin Hsieh[†] Gennady Pekhimenko[‡]
Eiman Ebrahimi[Ⓝ] Nastaran Hajinazar[†] Phillip B. Gibbons[†] Onur Mutlu^{§†}

[†]Carnegie Mellon University

[‡]University of Toronto

[Ⓝ]NVIDIA

[†]Simon Fraser University

[§]ETH Zürich

An Example: Heterogeneous-Reliability Memory

- Yixin Luo, Sriram Govindan, Bikash Sharma, Mark Santaniello, Justin Meza, Aman Kansal, Jie Liu, Badriddine Khessib, Kushagra Vaid, and Onur Mutlu, **"Characterizing Application Memory Error Vulnerability to Optimize Data Center Cost via Heterogeneous-Reliability Memory"** *Proceedings of the 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, Atlanta, GA, June 2014. [[Summary](#)] [[Slides \(pptx\)](#)] [[pdf](#)] [[Coverage on ZDNet](#)]

Characterizing Application Memory Error Vulnerability to Optimize Datacenter Cost via Heterogeneous-Reliability Memory

Yixin Luo Sriram Govindan* Bikash Sharma* Mark Santaniello* Justin Meza
Aman Kansal* Jie Liu* Badriddine Khessib* Kushagra Vaid* Onur Mutlu

Carnegie Mellon University, yixinluo@cs.cmu.edu, {meza, onur}@cmu.edu

*Microsoft Corporation, {srgovin, bsharma, marksan, kansal, jie.liu, bk Hessib, kvaid}@microsoft.com

Exploiting Memory Error Tolerance with Hybrid Memory Systems

Vulnerable data

Tolerant data

Reliable memory

Low-cost memory

On Microsoft's Web Search workload

Reduces server hardware **cost** by **4.7 %**

Achieves single server **availability** target of **99.90 %**

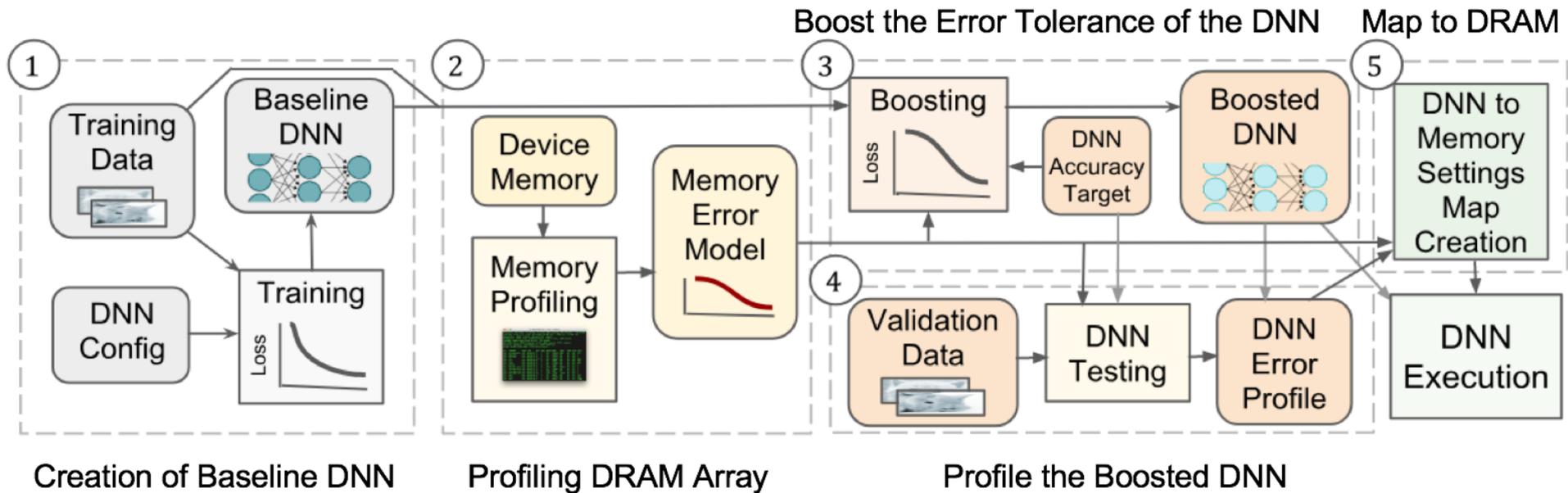
Heterogeneous-Reliability Memory [DSN 2014]

Another Example: EDEN

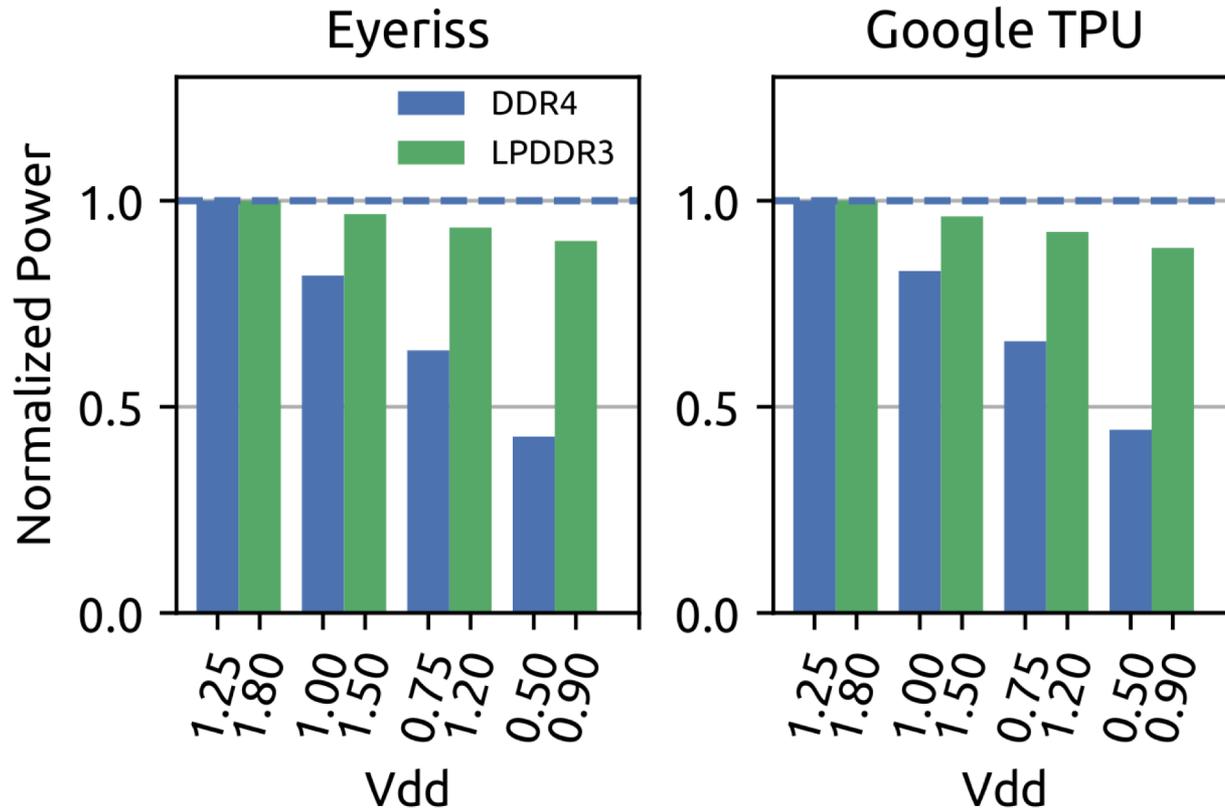
- Deep Neural Network evaluation is very DRAM-intensive (for large networks)
 1. Some data and layers in DNNs are very tolerant to errors
 2. We can reduce DRAM latency and voltage on such data and layers (intermediate feature maps and weights)
 3. While still achieving a user-specified DNN accuracy target

Data-aware management of DRAM latency and voltage

EDEN Flow



EDEN Power, Performance, Accuracy



- ~15-20% power savings, 8% perf improvement, <1% accuracy loss

Data-Aware
(Expressive)

Computing Architectures

Recap: Corollaries: Architectures Today

- Architectures are **terrible at dealing with data**
 - ❑ Designed to mainly store and move data vs. to compute
 - ❑ They are **processor-centric as opposed to data-centric**
- Architectures are **terrible at taking advantage of vast amounts of data** (and metadata) available to them
 - ❑ Designed to make simple decisions, ignoring lots of data
 - ❑ They make **human-driven decisions vs. data-driven decisions**
- Architectures are **terrible at knowing and exploiting different properties of application data**
 - ❑ Designed to treat all data as the same
 - ❑ They make **component-aware decisions vs. data-aware**

Architectures for Intelligent Machines

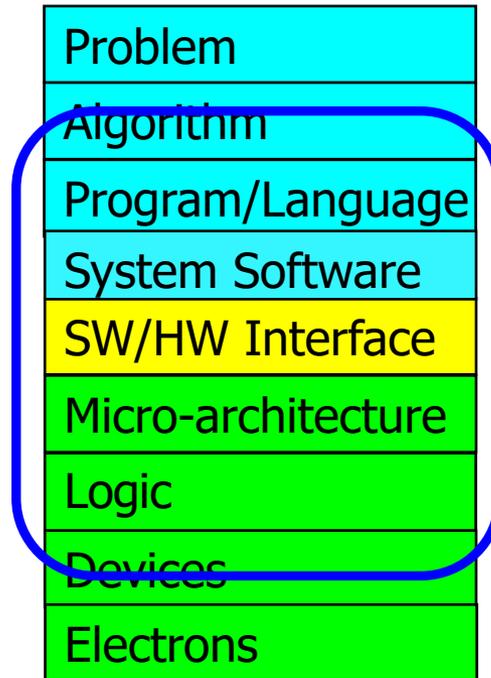
Data-centric

Data-driven

Data-aware



We Need to Revisit the Entire Stack



(Intelligent) Architectures for Intelligent Machines

Onur Mutlu

omutlu@gmail.com

<https://people.inf.ethz.ch/omutlu>

15 April 2019

Mubadala-SRC AI Hardware Systems Forum Keynote Talk

SAFARI

ETH zürich

Carnegie Mellon

Concluding Remarks

- It is time to design **principled system architectures** to solve the **data handling** problem
- **Design complete systems to be truly balanced, high-performance, and energy-efficient** → intelligent architectures
- **Data-centric, data-driven, data-aware**
- **This** can
 - Lead to **orders-of-magnitude** improvements
 - **Enable new applications & computing platforms**
 - **Enable better understanding of nature**
 - ...

Backup Slides

More on Processing in Memory

Performance: In-DRAM Bitwise Operations

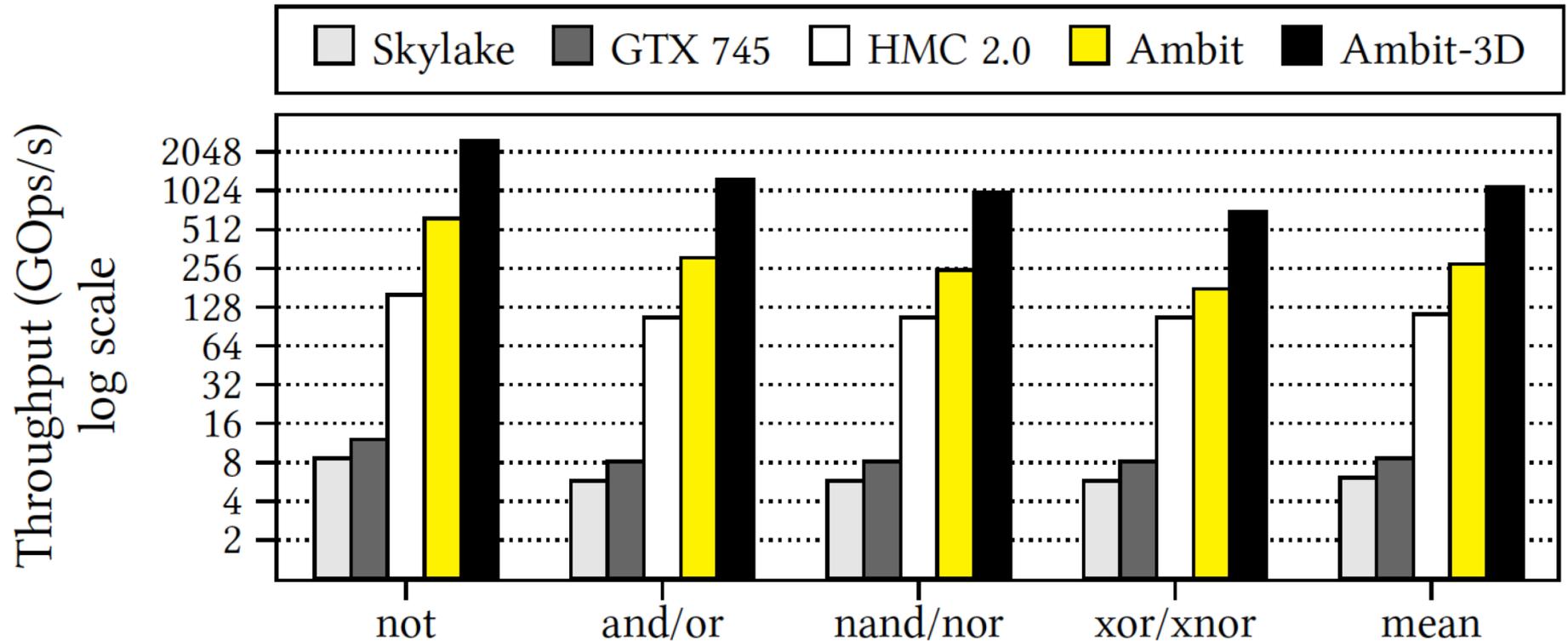


Figure 9: Throughput of bitwise operations on various systems.

Seshadri+, "Ambit: In-Memory Accelerator for Bulk Bitwise Operations using Commodity DRAM Technology," MICRO 2017.

Energy of In-DRAM Bitwise Operations

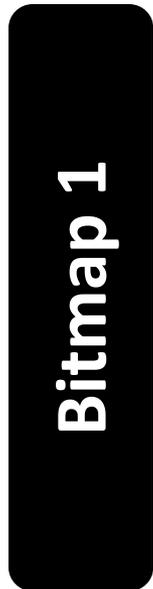
	Design	not	and/or	nand/nor	xor/xnor
DRAM &	DDR3	93.7	137.9	137.9	137.9
Channel Energy	Ambit	1.6	3.2	4.0	5.5
(nJ/KB)	(↓)	59.5X	43.9X	35.1X	25.1X

Table 3: Energy of bitwise operations. (↓) indicates energy reduction of Ambit over the traditional DDR3-based design.

Example Data Structure: Bitmap Index

- Alternative to B-tree and its variants
- Efficient for performing *range queries* and *joins*
- **Many bitwise operations to perform a query**

age < 18 18 < age < 25 25 < age < 60 age > 60



More on In-DRAM Bulk AND/OR

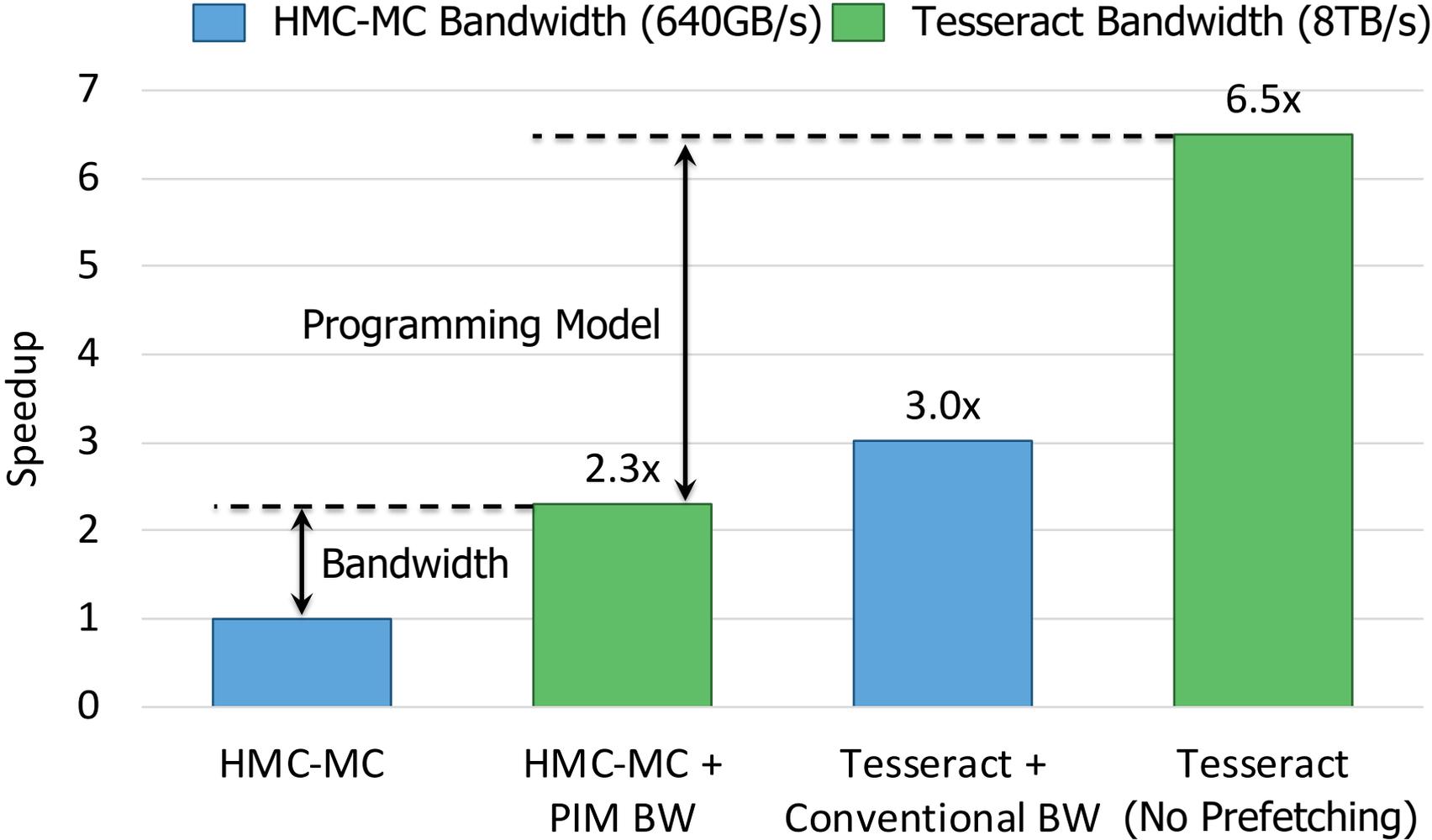
- Vivek Seshadri, Kevin Hsieh, Amirali Boroumand, Donghyuk Lee, Michael A. Kozuch, Onur Mutlu, Phillip B. Gibbons, and Todd C. Mowry,
"Fast Bulk Bitwise AND and OR in DRAM"
IEEE Computer Architecture Letters (***CAL***), April 2015.

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

Effect of Bandwidth & Programming Model



Google Workloads for Consumer Devices: Mitigating Data Movement Bottlenecks

Amirali Boroumand

Saugata Ghose, Youngsok Kim, Rachata Ausavarungnirun,
Eric Shiu, Rahul Thakur, Daehyun Kim, Aki Kuusela,
Allan Knies, Parthasarathy Ranganathan, Onur Mutlu

SAFARI

Carnegie Mellon

Google



SEOUL
NATIONAL
UNIVERSITY

ETH zürich

Quantization



Converts 32-bit floating point to 8-bit integers to improve inference execution time and energy consumption

Based on our analysis, we conclude that:

- Both functions are good candidates for **PIM execution**
- It is **feasible** to implement them in **PIM logic**

inference execution time

A simple **data conversion** operation that requires **shift, addition, and multiplication** operations

Evaluation Methodology

- **System Configuration (gem5 Simulator)**
 - **SoC:** 4 OoO cores, 8-wide issue, 64 kB L1 cache, 2MB L2 cache
 - **PIM Core:** 1 core per vault, 1-wide issue, 4-wide SIMD, 32kB L1 cache
 - **3D-Stacked Memory:** 2GB cube, 16 vaults per cube
 - Internal Bandwidth: 256GB/S
 - Off-Chip Channel Bandwidth: 32 GB/s
 - **Baseline Memory:** LPDDR3, 2GB, FR-FCFS scheduler
- **We study each target in isolation and emulate each separately and run them in our simulator**

Google Workloads for Consumer Devices: Mitigating Data Movement Bottlenecks

Amirali Boroumand

Saugata Ghose, Youngsok Kim, Rachata Ausavarungnirun,
Eric Shiu, Rahul Thakur, Daehyun Kim, Aki Kuusela,
Allan Knies, Parthasarathy Ranganathan, Onur Mutlu

ASPLOS 2018

SAFARI

Carnegie Mellon

Google



SEOUL
NATIONAL
UNIVERSITY

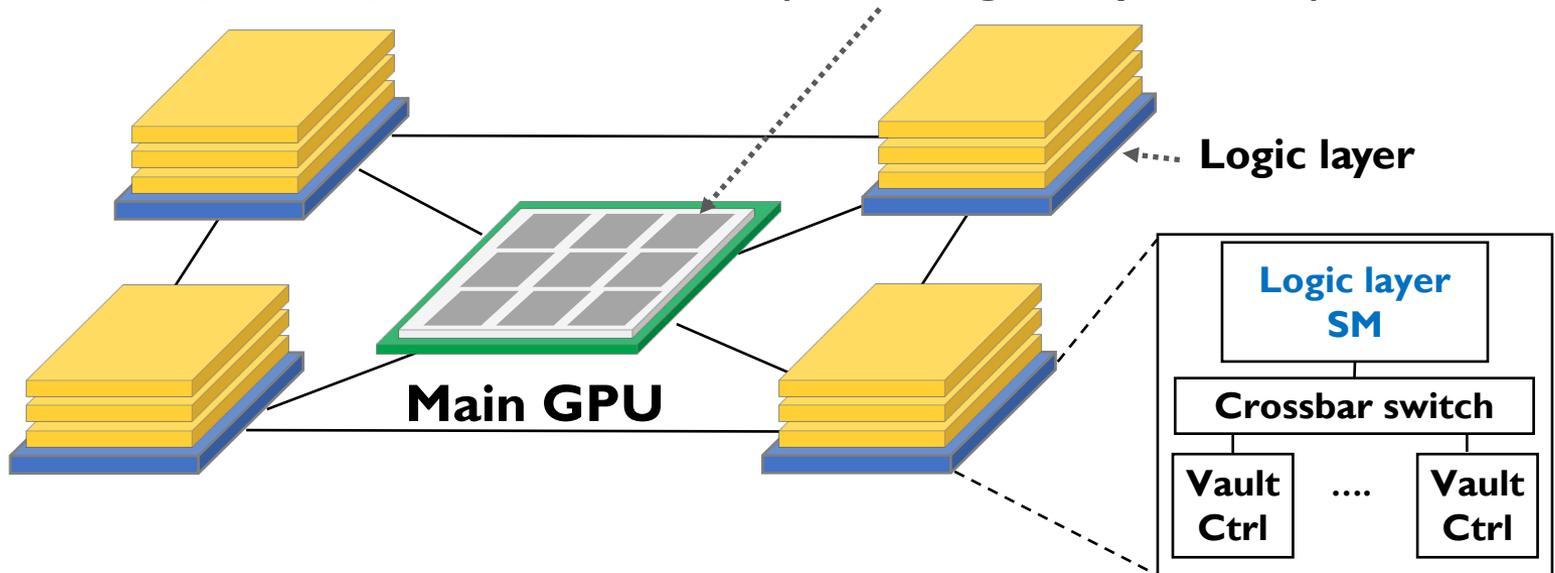
ETH zürich

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;
```

**3D-stacked memory
(memory stack)**

SM (Streaming Multiprocessor)



Accelerating GPU Execution with PIM (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

Accelerating GPU Execution with PIM (II)

- Ashutosh Pattnaik, Xulong Tang, Adwait Jog, Onur Kayiran, Asit K. Mishra, Mahmut T. Kandemir, Onur Mutlu, and Chita R. Das, **"Scheduling Techniques for GPU Architectures with Processing-In-Memory Capabilities"**
Proceedings of the 25th International Conference on Parallel Architectures and Compilation Techniques (PACT), Haifa, Israel, September 2016.

Scheduling Techniques for GPU Architectures with Processing-In-Memory Capabilities

Ashutosh Pattnaik¹ Xulong Tang¹ Adwait Jog² Onur Kayiran³
Asit K. Mishra⁴ Mahmut T. Kandemir¹ Onur Mutlu^{5,6} Chita R. Das¹

¹Pennsylvania State University ²College of William and Mary
³Advanced Micro Devices, Inc. ⁴Intel Labs ⁵ETH Zürich ⁶Carnegie Mellon University

Accelerating Linked Data Structures

- 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*

Accelerating Dependent Cache Misses

- Milad Hashemi, Khubaib, Eiman Ebrahimi, Onur Mutlu, and Yale N. Patt, **"Accelerating Dependent Cache Misses with an Enhanced Memory Controller"**

Proceedings of the 43rd International Symposium on Computer Architecture (ISCA), Seoul, South Korea, June 2016.

[[Slides \(pptx\)](#) ([pdf](#))]

[[Lightning Session Slides \(pptx\)](#) ([pdf](#))]

Accelerating Dependent Cache Misses with an Enhanced Memory Controller

Milad Hashemi*, Khubaib†, Eiman Ebrahimi‡, Onur Mutlu§, Yale N. Patt*

*The University of Texas at Austin †Apple ‡NVIDIA §ETH Zürich & Carnegie Mellon University

Two Key Questions in 3D-Stacked PIM

- What are the performance and energy benefits of using 3D-stacked memory as a coarse-grained accelerator?
 - By changing the entire system
 - By performing simple function offloading
- What is the minimal processing-in-memory support we can provide?
 - With minimal changes to system and programming

PIM-Enabled Instructions

- 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

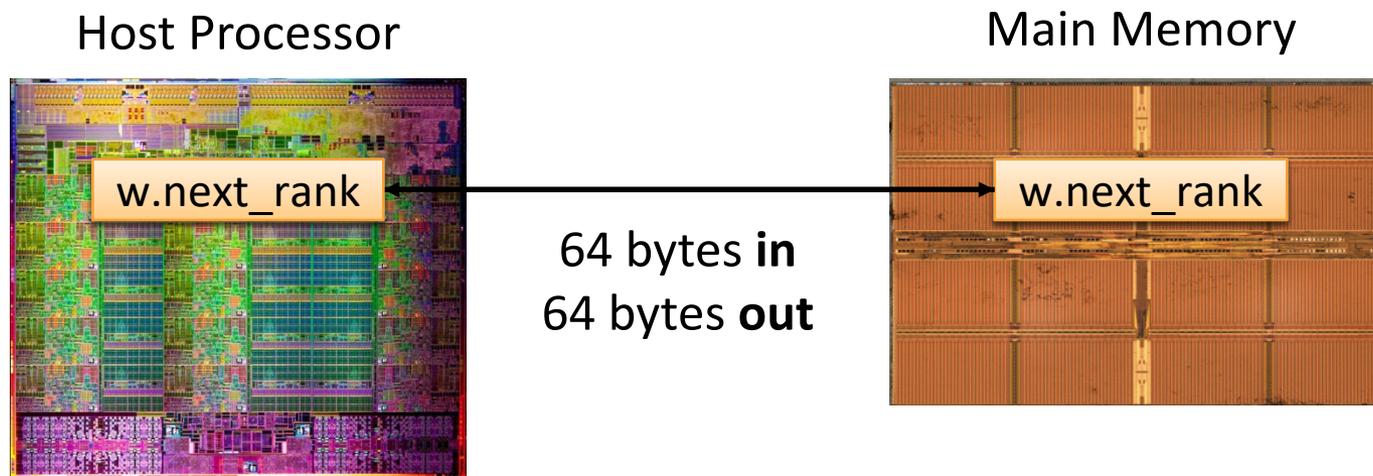
[†]Carnegie Mellon University

PEI: PIM-Enabled Instructions (Ideas)

- **Goal:** Develop mechanisms to get the most out of near-data processing with **minimal cost, minimal changes to the system, no changes to the programming model**
- **Key Idea 1:** Expose each PIM operation as a **cache-coherent, virtually-addressed host processor instruction** (called PEI) that operates on **only a single cache block**
 - e.g., `__pim_add(&w.next_rank, value) → pim.add r1, (r2)`
 - No changes sequential execution/programming model
 - No changes to virtual memory
 - Minimal changes to cache coherence
 - No need for data mapping: Each PEI restricted to a single memory module
- **Key Idea 2:** **Dynamically decide where to execute a PEI** (i.e., the host processor or PIM accelerator) based on simple locality characteristics and simple hardware predictors
 - Execute each operation at the location that provides the best performance

Simple PIM Operations as ISA Extensions (II)

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



Conventional Architecture

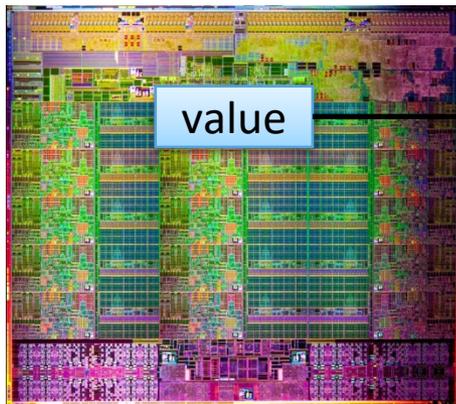
Simple PIM Operations as ISA Extensions (III)

```
for (v: graph.vertices) {  
  value = weight * v.rank;  
  for (w: v.successors) {  
    __pim_add(&w.next_rank, value);  
  }  
}
```

pim.add r1, (r2)

__pim_add(&w.next_rank, value);

Host Processor



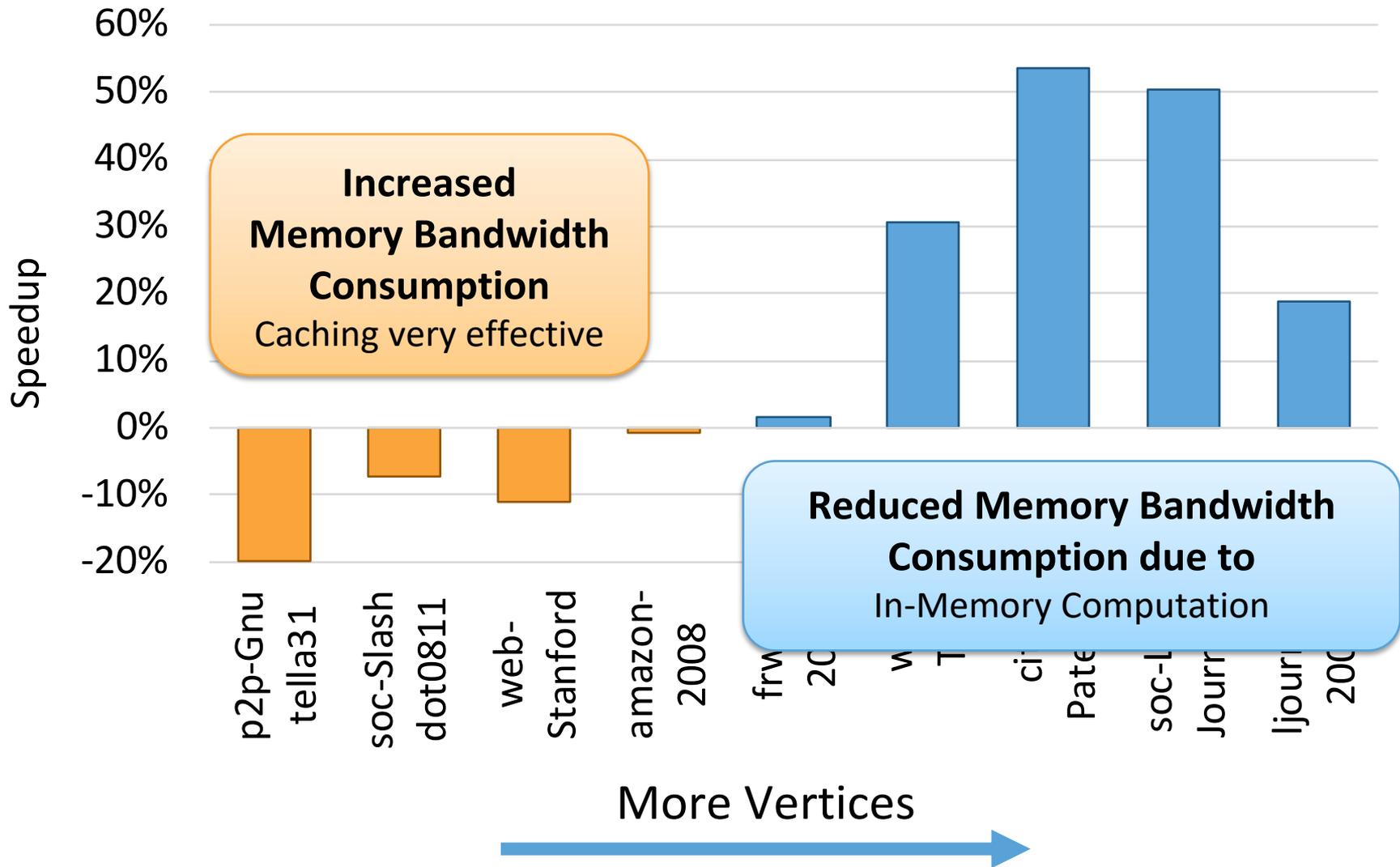
Main Memory



8 bytes in
0 bytes out

In-Memory Addition

Always Executing in Memory? Not A Good Idea



PEI: PIM-Enabled Instructions (Example)

```
for (v: graph.vertices) {  
    value = weight * v.rank;  
    for (w: v.successors) {  
        __pim_add(&w.next_rank, value);  
    }  
}
```

pim.add r1, (r2)

__pim_add(&w.next_rank, value);

pfence

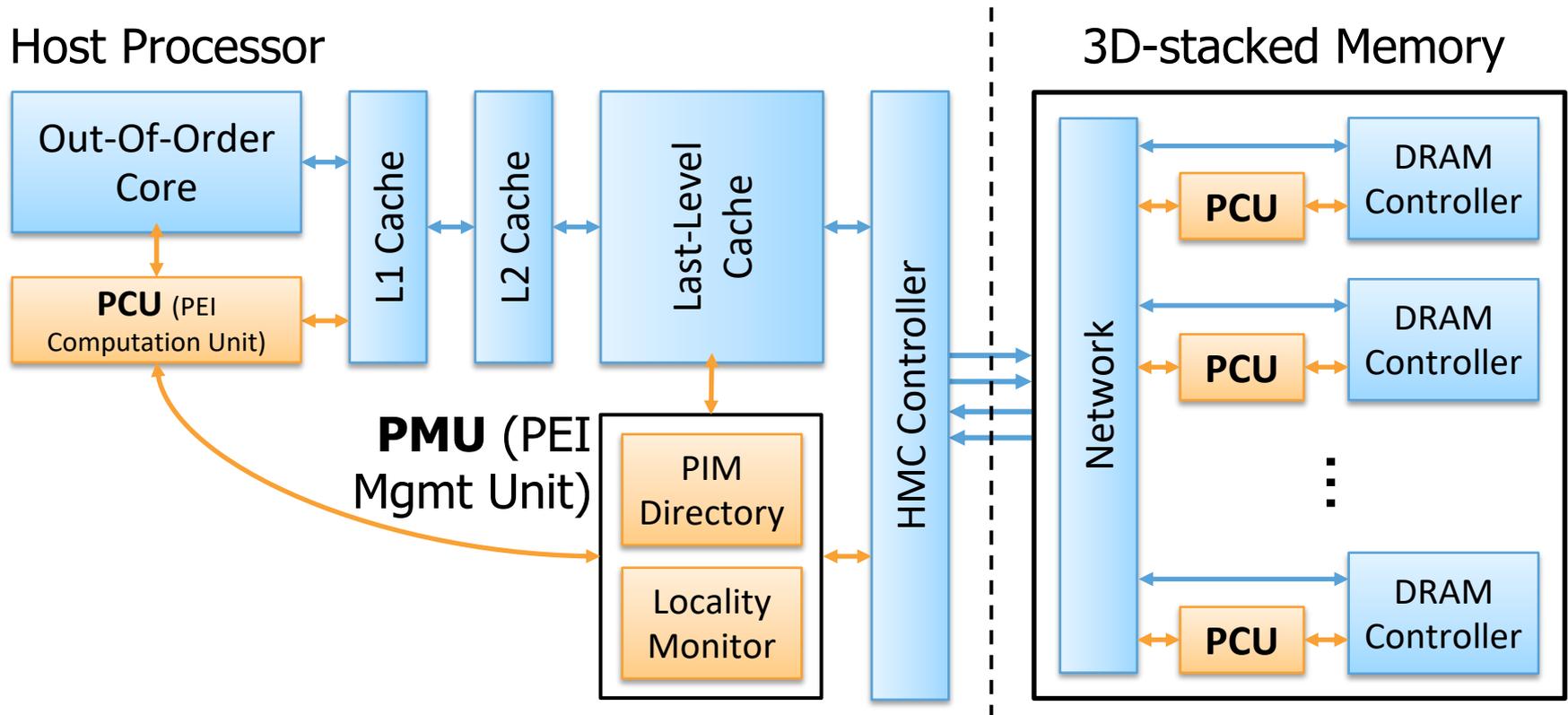
pfence();

Table 1: Summary of Supported PIM Operations

Operation	R	W	Input	Output	Applications
8-byte integer increment	O	O	0 bytes	0 bytes	AT
8-byte integer min	O	O	8 bytes	0 bytes	BFS, SP, WCC
Floating-point add	O	O	8 bytes	0 bytes	PR
Hash table probing	O	X	8 bytes	9 bytes	HJ
Histogram bin index	O	X	1 byte	16 bytes	HG, RP
Euclidean distance	O	X	64 bytes	4 bytes	SC
Dot product	O	X	32 bytes	8 bytes	SVM

- Executed either in memory or in the processor: dynamic decision
 - Low-cost locality monitoring for a single instruction
- Cache-coherent, virtually-addressed, single cache block only
- Atomic between different PEIs
- *Not* atomic with normal instructions (use *pfence* for ordering)

Example (Abstract) PEI uArchitecture



Example PEI uArchitecture

PEI: Initial Evaluation Results

- Initial evaluations with **10 emerging data-intensive workloads**
 - Large-scale graph processing
 - In-memory data analytics
 - Machine learning and data mining
 - Three input sets (small, medium, large) for each workload to analyze the impact of data locality
- Pin-based cycle-level x86-64 simulation
- **Performance Improvement and Energy Reduction:**
 - 47% average speedup with large input data sets
 - 32% speedup with small input data sets
 - 25% avg. energy reduction in a single node with large input data sets

Table 2: Baseline Simulation Configuration

Component	Configuration
Core	16 out-of-order cores, 4 GHz, 4-issue
L1 I/D-Cache	Private, 32 KB, 4/8-way, 64 B blocks, 16 MSHRs
L2 Cache	Private, 256 KB, 8-way, 64 B blocks, 16 MSHRs
L3 Cache	Shared, 16 MB, 16-way, 64 B blocks, 64 MSHRs
On-Chip Network	Crossbar, 2 GHz, 144-bit links
Main Memory	32 GB, 8 HMCs, daisy-chain (80 GB/s full-duplex)
HMC	4 GB, 16 vaults, 256 DRAM banks [20]
– DRAM	FR-FCFS, tCL = tRCD = tRP = 13.75 ns [27]
– Vertical Links	64 TSVs per vault with 2 Gb/s signaling rate [23]

Simpler PIM: PIM-Enabled Instructions

- 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

Automatic Code and Data Mapping

- 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

Automatic Offloading of Critical Code

- Milad Hashemi, Khubaib, Eiman Ebrahimi, Onur Mutlu, and Yale N. Patt, **"Accelerating Dependent Cache Misses with an Enhanced Memory Controller"**
Proceedings of the 43rd International Symposium on Computer Architecture (ISCA), Seoul, South Korea, June 2016.
[[Slides \(pptx\)](#)] [[pdf](#)]
[[Lightning Session Slides \(pptx\)](#)] [[pdf](#)]

Accelerating Dependent Cache Misses with an Enhanced Memory Controller

Milad Hashemi*, Khubaib†, Eiman Ebrahimi‡, Onur Mutlu§, Yale N. Patt*

*The University of Texas at Austin †Apple ‡NVIDIA §ETH Zürich & Carnegie Mellon University

Automatic Offloading of Prefetch Mechanisms

- Milad Hashemi, Onur Mutlu, and Yale N. Patt,
"Continuous Runahead: Transparent Hardware Acceleration for Memory Intensive Workloads"
Proceedings of the 49th International Symposium on Microarchitecture (MICRO), Taipei, Taiwan, October 2016.
[[Slides \(pptx\)](#)] [[pdf](#)] [[Lightning Session Slides \(pdf\)](#)] [[Poster \(pptx\)](#)] [[pdf](#)]

Continuous Runahead: Transparent Hardware Acceleration for Memory Intensive Workloads

Milad Hashemi*, Onur Mutlu[§], Yale N. Patt*

**The University of Texas at Austin* [§]*ETH Zürich*

Efficient Automatic Data Coherence Support

- 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^{‡†}

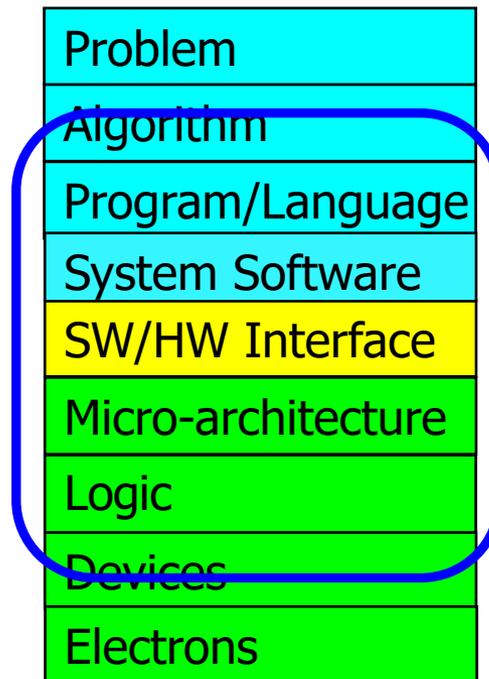
[†] *Carnegie Mellon University* ^{*} *Samsung Semiconductor, Inc.* [§] *TOBB ETÜ* [‡] *ETH Zürich*

How to Enable Adoption of Processing in Memory

Barriers to Adoption of PIM

1. Functionality of and applications for PIM
2. Ease of programming (interfaces and compiler/HW support)
3. System support: coherence & virtual memory
4. Runtime systems for adaptive scheduling, data mapping, access/sharing control
5. Infrastructures to assess benefits and feasibility

We Need to Revisit the Entire Stack



Open Problems: PIM Adoption

Enabling the Adoption of Processing-in-Memory: Challenges, Mechanisms, Future Research Directions

SAUGATA GHOSE, KEVIN HSIEH, AMIRALI BOROUMAND,
RACHATA AUSAVARUNGNIRUN

Carnegie Mellon University

ONUR MUTLU

ETH Zürich and Carnegie Mellon University

Saugata Ghose, Kevin Hsieh, Amirali Boroumand, Rachata Ausavarungnirun, Onur Mutlu,
**"Enabling the Adoption of Processing-in-Memory: Challenges, Mechanisms,
Future Research Directions"**

Invited Book Chapter, to appear in 2018.

[[Preliminary arxiv.org version](https://arxiv.org/abs/1802.00320)]

Open Problems: PIM Adoption

Processing Data Where It Makes Sense: Enabling In-Memory Computation

Onur Mutlu^{a,b}, Saugata Ghose^b, Juan Gómez-Luna^a, Rachata Ausavarungnirun^{b,c}

^a*ETH Zürich*

^b*Carnegie Mellon University*

^c*King Mongkut's University of Technology North Bangkok*

Onur Mutlu, Saugata Ghose, Juan Gomez-Luna, and Rachata Ausavarungnirun,
**"Processing Data Where It Makes Sense: Enabling In-Memory
Computation"**

Invited paper in Microprocessors and Microsystems (MICPRO), June 2019.

[arXiv version]

Readings, Videos, Reference Materials

Accelerated Memory Course (~6.5 hours)

■ ACACES 2018

- ❑ Memory Systems and Memory-Centric Computing Systems
- ❑ Taught by Onur Mutlu July 9-13, 2018
- ❑ ~6.5 hours of lectures

■ Website for the Course including Videos, Slides, Papers

- ❑ <https://people.inf.ethz.ch/omutlu/acaces2018.html>
- ❑ <https://www.youtube.com/playlist?list=PL5Q2soXY2Zi-HXxomthrpDpMJm05P6J9x>

■ All Papers are at:

- ❑ <https://people.inf.ethz.ch/omutlu/projects.htm>
- ❑ Final lecture notes and readings (for all topics)

Reference Overview Paper I

Processing Data Where It Makes Sense: Enabling In-Memory Computation

Onur Mutlu^{a,b}, Saugata Ghose^b, Juan Gómez-Luna^a, Rachata Ausavarungnirun^{b,c}

^a*ETH Zürich*

^b*Carnegie Mellon University*

^c*King Mongkut's University of Technology North Bangkok*

Onur Mutlu, Saugata Ghose, Juan Gomez-Luna, and Rachata Ausavarungnirun,
["Processing Data Where It Makes Sense: Enabling In-Memory Computation"](#)

Invited paper in [Microprocessors and Microsystems \(MICPRO\)](#), June 2019.
[\[arXiv version\]](#)

Reference Overview Paper II

Enabling the Adoption of Processing-in-Memory: Challenges, Mechanisms, Future Research Directions

SAUGATA GHOSE, KEVIN HSIEH, AMIRALI BOROUMAND,
RACHATA AUSAVARUNGNIRUN

Carnegie Mellon University

ONUR MUTLU

ETH Zürich and Carnegie Mellon University

Saugata Ghose, Kevin Hsieh, Amirali Boroumand, Rachata Ausavarungnirun, Onur Mutlu,
**"Enabling the Adoption of Processing-in-Memory: Challenges, Mechanisms,
Future Research Directions"**

Invited Book Chapter, to appear in 2018.

[[Preliminary arxiv.org version](https://arxiv.org/abs/1802.00320)]

Reference Overview Paper III

- Onur Mutlu and Lavanya Subramanian,
"Research Problems and Opportunities in Memory Systems"
Invited Article in Supercomputing Frontiers and Innovations (SUPERFRI), 2014/2015.

Research Problems and Opportunities in Memory Systems

Onur Mutlu¹, Lavanya Subramanian¹

Reference Overview Paper IV

- Onur Mutlu,
"The RowHammer Problem and Other Issues We May Face as Memory Becomes Denser"
Invited Paper in Proceedings of the Design, Automation, and Test in Europe Conference (DATE), Lausanne, Switzerland, March 2017.
[[Slides \(pptx\)](#) ([pdf](#))]

The RowHammer Problem and Other Issues We May Face as Memory Becomes Denser

Onur Mutlu
ETH Zürich
onur.mutlu@inf.ethz.ch
<https://people.inf.ethz.ch/omutlu>

Reference Overview Paper V

- Onur Mutlu,

- "Memory Scaling: A Systems Architecture Perspective"**

- Technical talk at MemCon 2013 (**MEMCON**), Santa Clara, CA, August 2013. [[Slides \(pptx\)](#)] [[pdf](#)]*
[[Video](#)] [[Coverage on StorageSearch](#)]

Memory Scaling: A Systems Architecture Perspective

Onur Mutlu

Carnegie Mellon University

onur@cmu.edu

<http://users.ece.cmu.edu/~omutlu/>

https://people.inf.ethz.ch/omutlu/pub/memory-scaling_memcon13.pdf



Proceedings of the IEEE, Sept. 2017

Error Characterization, Mitigation, and Recovery in Flash-Memory-Based Solid-State Drives

This paper reviews the most recent advances in solid-state drive (SSD) error characterization, mitigation, and data recovery techniques to improve both SSD's reliability and lifetime.

By YU CAI, SAUGATA GHOSE, ERICH F. HARATSCH, YIXIN LUO, AND ONUR MUTLU

Related Videos and Course Materials (I)

- **Undergraduate Computer Architecture Course Lecture Videos (2015, 2014, 2013)**
- **Undergraduate Computer Architecture Course Materials (2015, 2014, 2013)**

- **Graduate Computer Architecture Course Lecture Videos (2018, 2017, 2015, 2013)**
- **Graduate Computer Architecture Course Materials (2018, 2017, 2015, 2013)**

- **Parallel Computer Architecture Course Materials (Lecture Videos)**

Related Videos and Course Materials (II)

- **Freshman Digital Circuits and Computer Architecture Course Lecture Videos (2018, 2017)**
- **Freshman Digital Circuits and Computer Architecture Course Materials (2018)**

- **Memory Systems Short Course Materials (Lecture Video on Main Memory and DRAM Basics)**

Some Open Source Tools (I)

- Rowhammer – Program to Induce RowHammer Errors
 - <https://github.com/CMU-SAFARI/rowhammer>
- Ramulator – Fast and Extensible DRAM Simulator
 - <https://github.com/CMU-SAFARI/ramulator>
- MemSim – Simple Memory Simulator
 - <https://github.com/CMU-SAFARI/memsim>
- NOCulator – Flexible Network-on-Chip Simulator
 - <https://github.com/CMU-SAFARI/NOCulator>
- SoftMC – FPGA-Based DRAM Testing Infrastructure
 - <https://github.com/CMU-SAFARI/SoftMC>
- Other open-source software from my group
 - <https://github.com/CMU-SAFARI/>
 - <http://www.ece.cmu.edu/~safari/tools.html>

Some Open Source Tools (II)

- MQSim – A Fast Modern SSD Simulator
 - <https://github.com/CMU-SAFARI/MQSim>
- Mosaic – GPU Simulator Supporting Concurrent Applications
 - <https://github.com/CMU-SAFARI/Mosaic>
- IMPICA – Processing in 3D-Stacked Memory Simulator
 - <https://github.com/CMU-SAFARI/IMPICA>
- SMLA – Detailed 3D-Stacked Memory Simulator
 - <https://github.com/CMU-SAFARI/SMLA>
- HWASim – Simulator for Heterogeneous CPU-HWA Systems
 - <https://github.com/CMU-SAFARI/HWASim>
- Other open-source software from my group
 - <https://github.com/CMU-SAFARI/>
 - <http://www.ece.cmu.edu/~safari/tools.html>

More Open Source Tools (III)

- A lot more open-source software from my group
 - <https://github.com/CMU-SAFARI/>
 - <http://www.ece.cmu.edu/~safari/tools.html>

SAFARI 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.

ETH Zurich and Carnegi... <http://www.ece.cmu.ed...> omutlu@gmail.com

Repositories 30 People 27 Teams 1 Projects 0 Settings

Search repositories... Type: All Language: All Customize pinned repositories [New](#)

MQSim

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 implementations, steady-state SSD conditions, and the full end-to-end latency of requests in modern SSDs. It is described in detail in the FAST 2018 paper by A...

● C++ ★ 14 🍴 14 🏢 MIT Updated 8 days ago

Top languages

- C++
- C
- C#
- AGS Script
- Verilog

Most used topics [Manage](#)

- dram
- reliability

Referenced Papers

- All are available at

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

<http://scholar.google.com/citations?user=7XyGUGkAAAAJ&hl=en>

<https://people.inf.ethz.ch/omutlu/acaces2018.html>