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

**Geraldo F. Oliveira**

Juan Gómez-Luna  Lois Orosa  Saugata Ghose

Nandita Vijaykumar  Ivan Fernandez  Mohammad Sadrosadati

Onur Mutlu

SAFARI

ETH Zürich  UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN  UNIVERSITY OF TORONTO  UNIVERSITAS MALACITANA  UNIVERSIDAD DE MÁLAGA

# Executive Summary

- **Problem:** Data movement is a major bottleneck is modern systems. However, it is **unclear** how to identify:
  - **different sources** of data movement bottlenecks
  - the **most suitable** mitigation technique (e.g., caching, prefetching, near-data processing) for a given data movement bottleneck
- **Goals:**
  1. Design a methodology to **identify** sources of data movement bottlenecks
  2. **Compare** compute- and memory-centric data movement mitigation techniques
- **Key Approach:** Perform a large-scale application characterization to identify **key metrics** that reveal the sources to data movement bottlenecks
- **Key Contributions**:
  - **Experimental characterization** of 77K functions across 345 applications
  - A **methodology** to characterize applications based on data movement bottlenecks and their relation with different data movement mitigation techniques
  - **DAMOV:** a **benchmark suite** with **144 functions** for data movement studies
  - **Four case-studies** to highlight DAMOV's applicability to open research problems

# Outline

1. Data Movement Bottlenecks
2. Methodology Overview
3. Application Profiling
4. Locality-Based Clustering
5. Memory Bottleneck Analysis
6. Case Studies

# Outline

**1. Data Movement Bottlenecks**

2. Methodology Overview

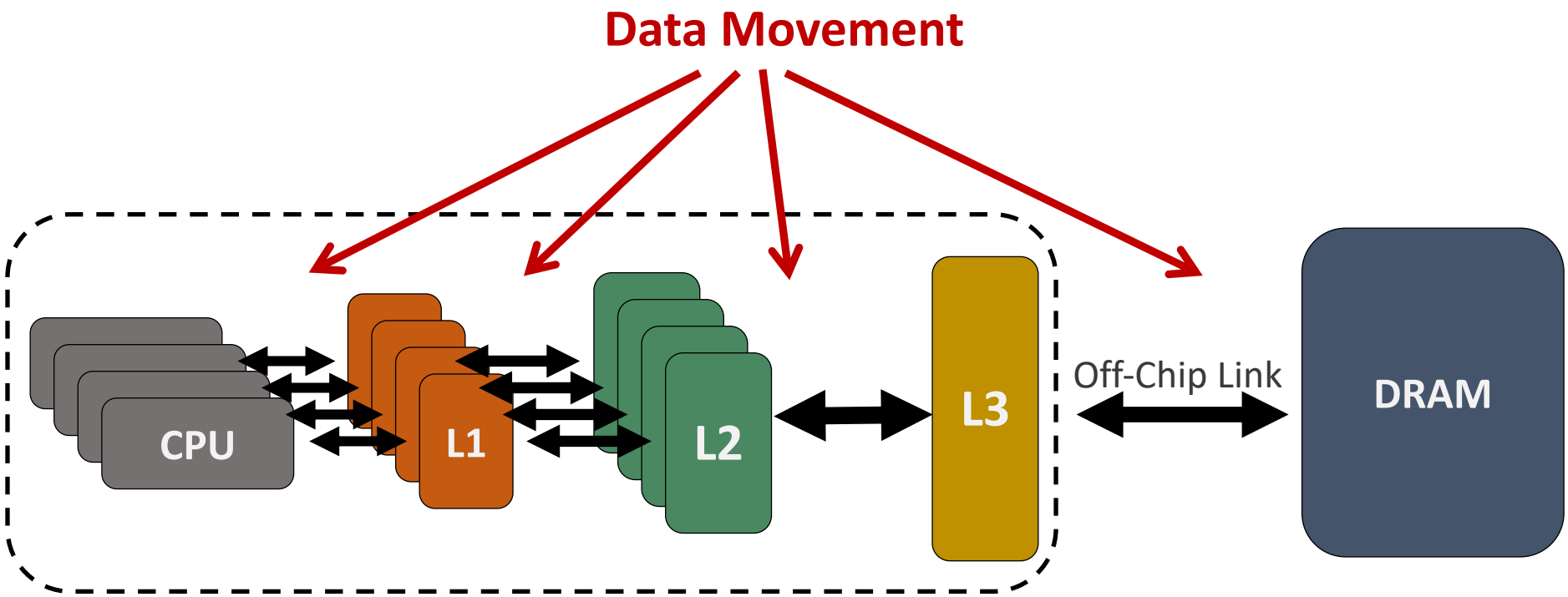
3. Application Profiling

4. Locality-Based Clustering

5. Memory Bottleneck Analysis

6. Case Studies

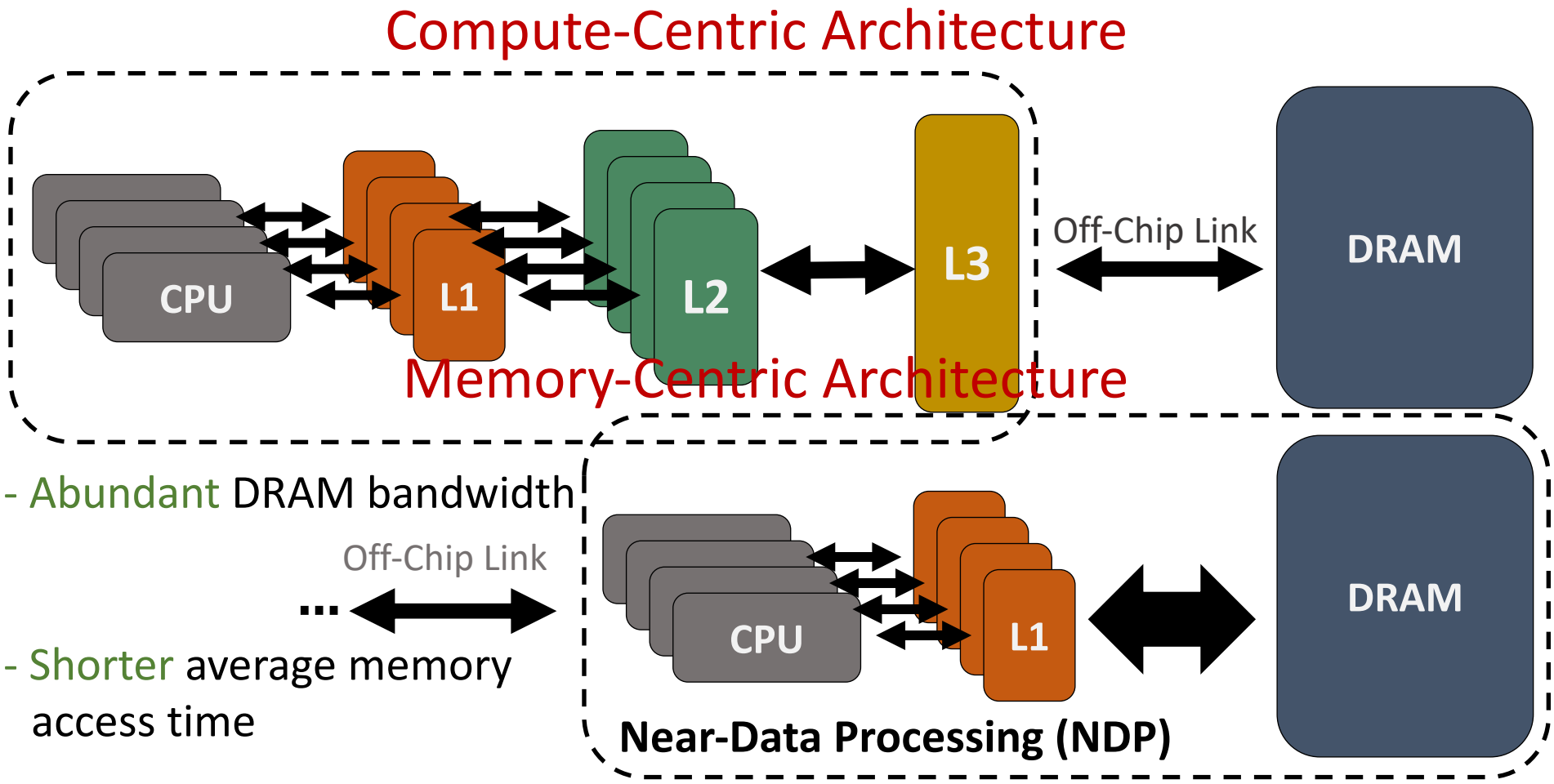
# Data Movement Bottlenecks (1/2)




Data movement bottlenecks happen because of:

- Not enough data **locality** → ineffective use of the cache hierarchy
- Not enough **memory bandwidth**
- High average **memory access time**

# Data Movement Bottlenecks (2/2)



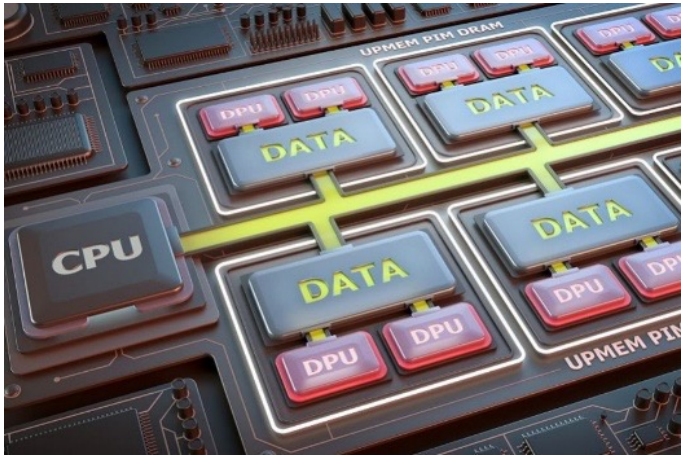
# Near-Data Processing (1/2)




The goal of Near-Data Processing (NDP) is
**to mitigate data movement**

# Near-Data Processing (2/2)

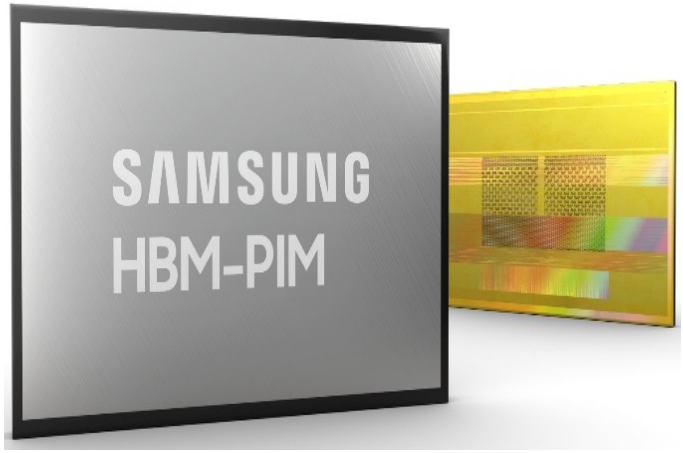
## UPMEM (2019)




Near-DRAM-banks processing
for general-purpose computing

**0.9 TOPS compute throughput[1]**

## Samsung FIMDRAM (2021)




Near-DRAM-banks processing
for neural networks

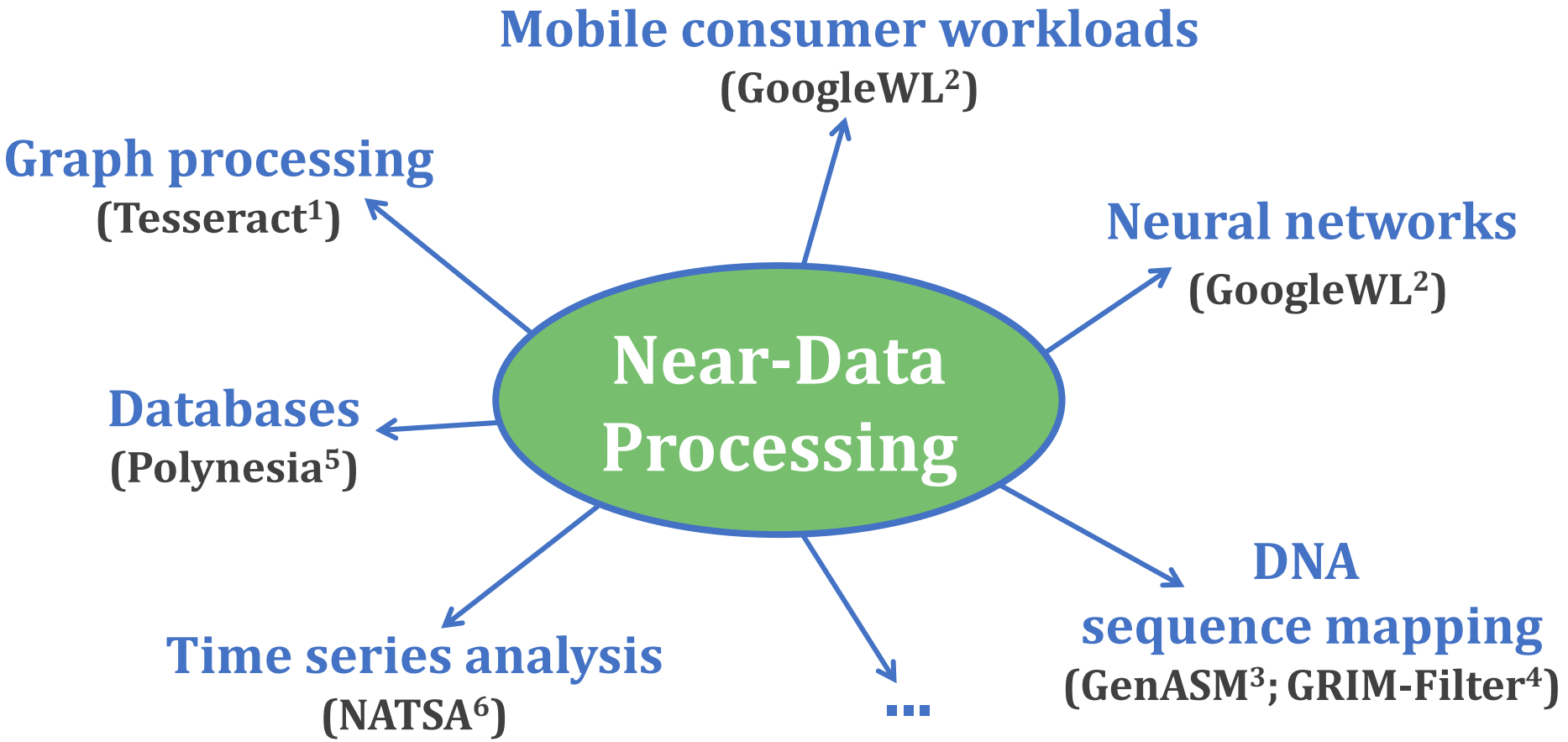
**1.2 TFLOPS compute throughput[2]**

The goal of Near-Data Processing (NDP) is
**to mitigate data movement**



[1] Devaux, "The True Processing In Memory Accelerator," HCS, 2019
[2] Kwon+, "A 20nm 6GB Function-In-Memory DRAM, Based on HBM2 with a 1.2TFLOPS Programmable Computing Unit Using Bank-Level Parallelism, for Machine Learning Applications," ISSCC, 2021

# When to Employ Near-Data Processing?





[1] Ahn+, "A Scalable Processing-in-Memory Accelerator for Parallel Graph Processing," ISCA, 2015

[2] Boroumand+, "Google Workloads for Consumer Devices: Mitigating Data Movement Bottlenecks," ASPLOS, 2018

[3] Cali+, "GenASM: A High-Performance, Low-Power Approximate String Matching Acceleration Framework for Genome Sequence Analysis," MICRO, 2020

[4] Kim+, "GRIM-Filter: Fast Seed Location Filtering in DNA Read Mapping Using Processing-in-Memory Technologies," BMC Genomics, 2018

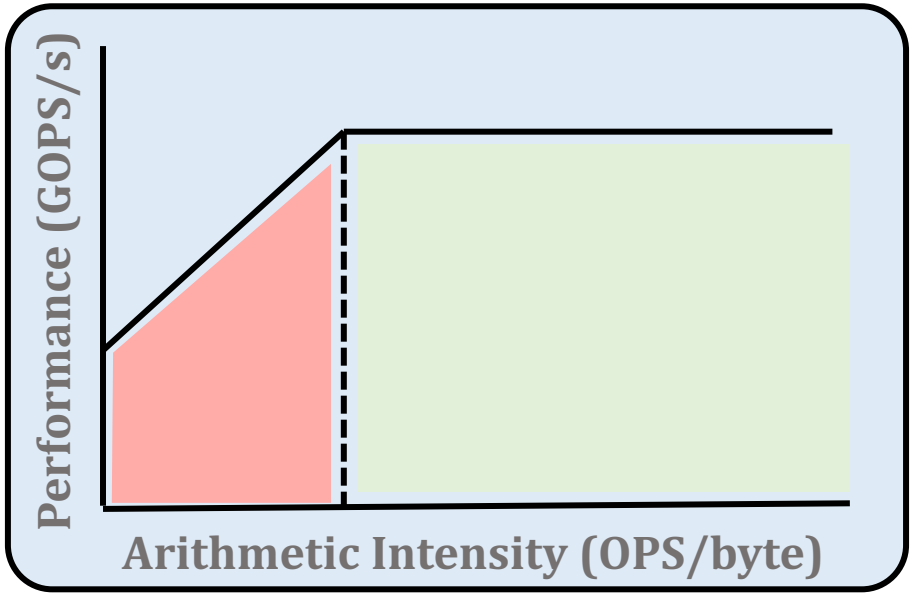
[5] Boroumand+, "Polynesia: Enabling Effective Hybrid Transactional/Analytical Databases with Specialized Hardware/Software Co-Design," arXiv:2103.00798 [cs.AR], 2021

[6] Fernandez+, "NATSA: A Near-Data Processing Accelerator for Time Series Analysis," ICCD, 2020

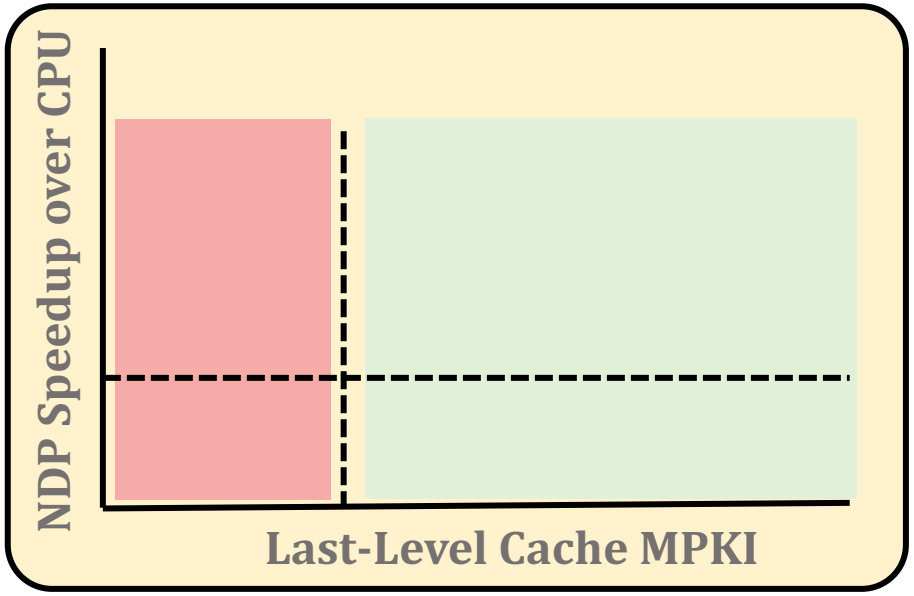
# Identifying Memory Bottlenecks

- Multiple approaches to identify applications that:
  - suffer from data movement bottlenecks
  - take advantage of NDP
- Existing approaches are not comprehensive enough

**Roofline model**

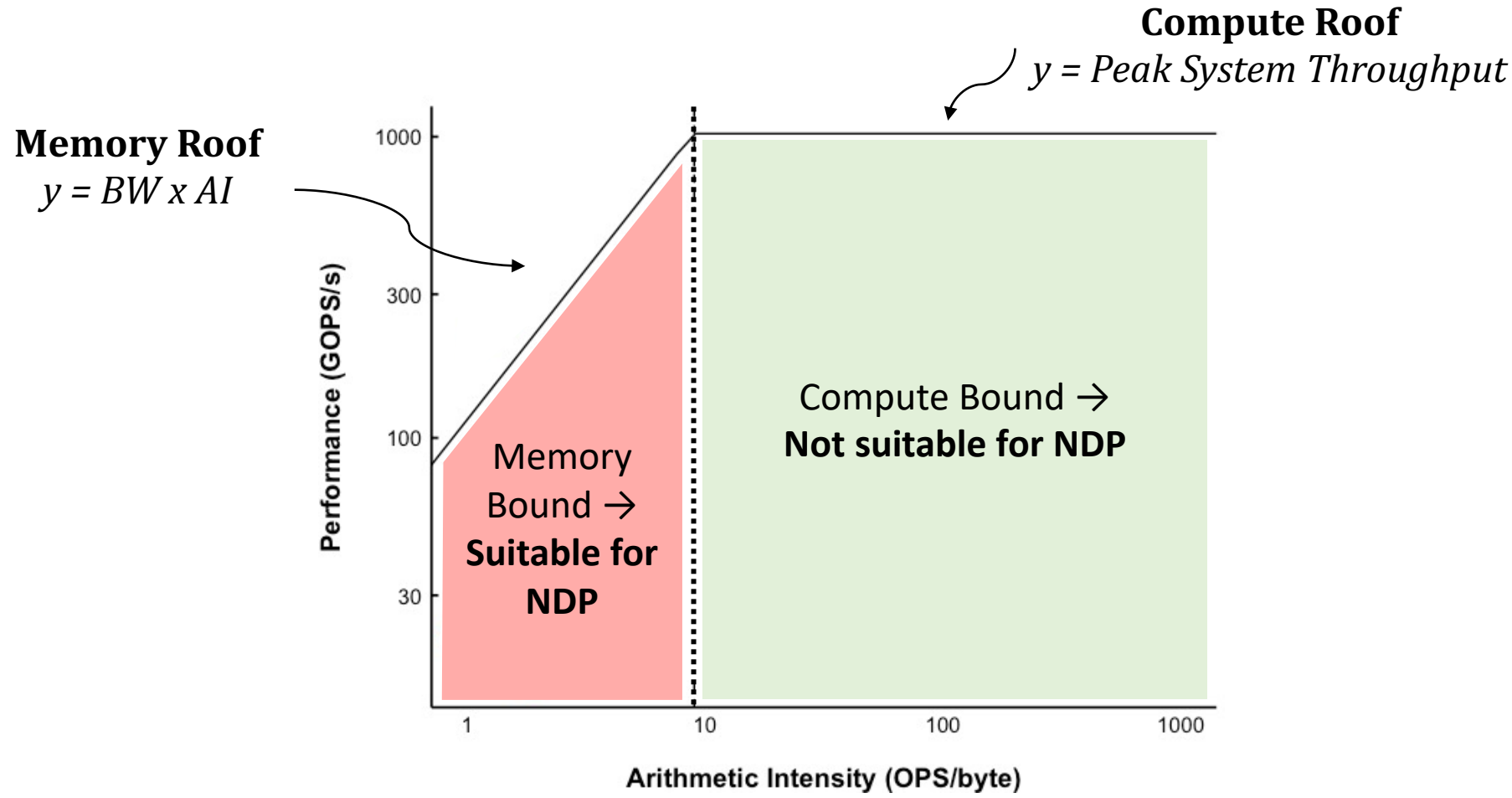



**High LLC MPKI**



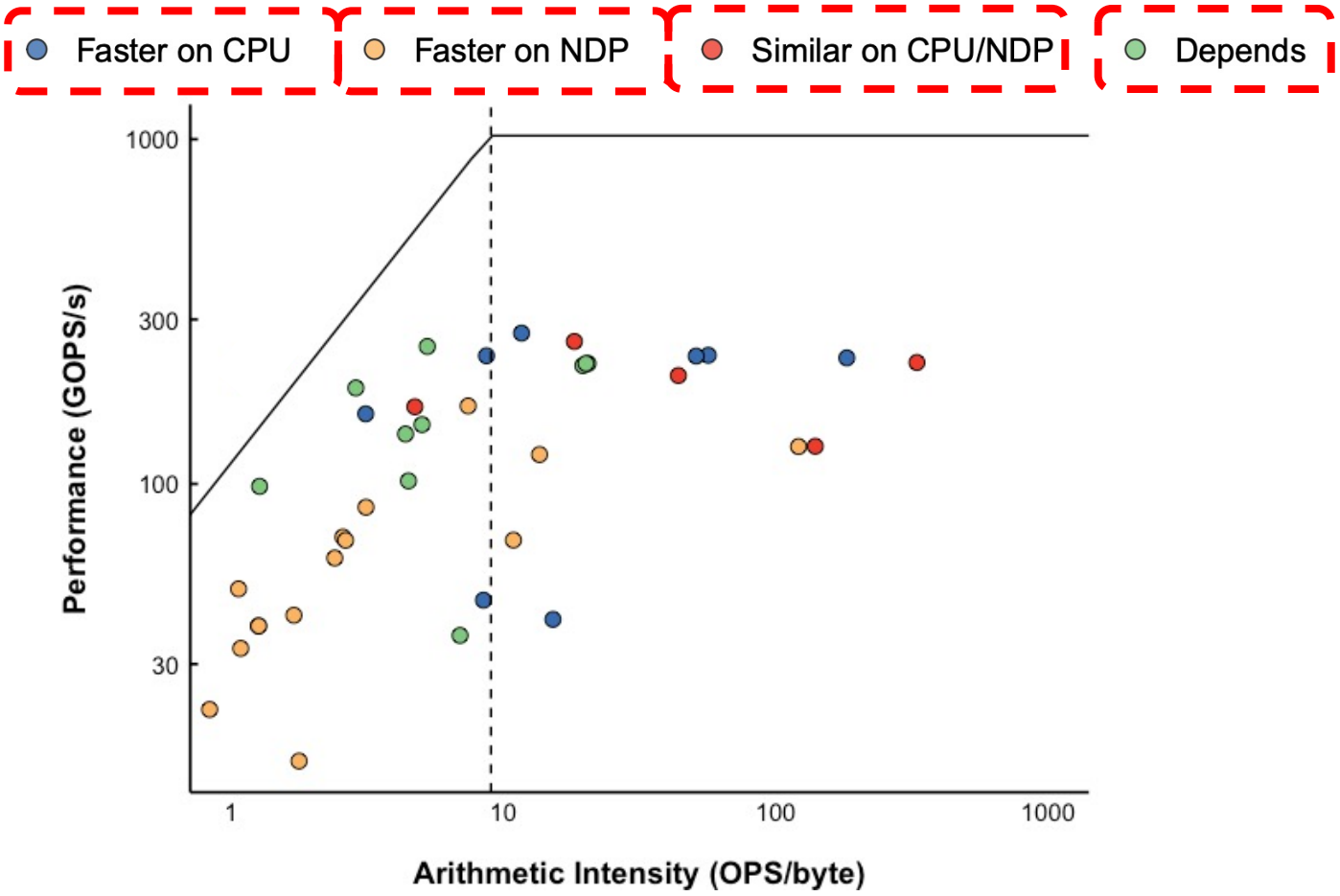
# Limitations of Prior Approaches (1/2)

- **Roofline model** → identifies when an application is *bounded* by **compute** or **memory** units



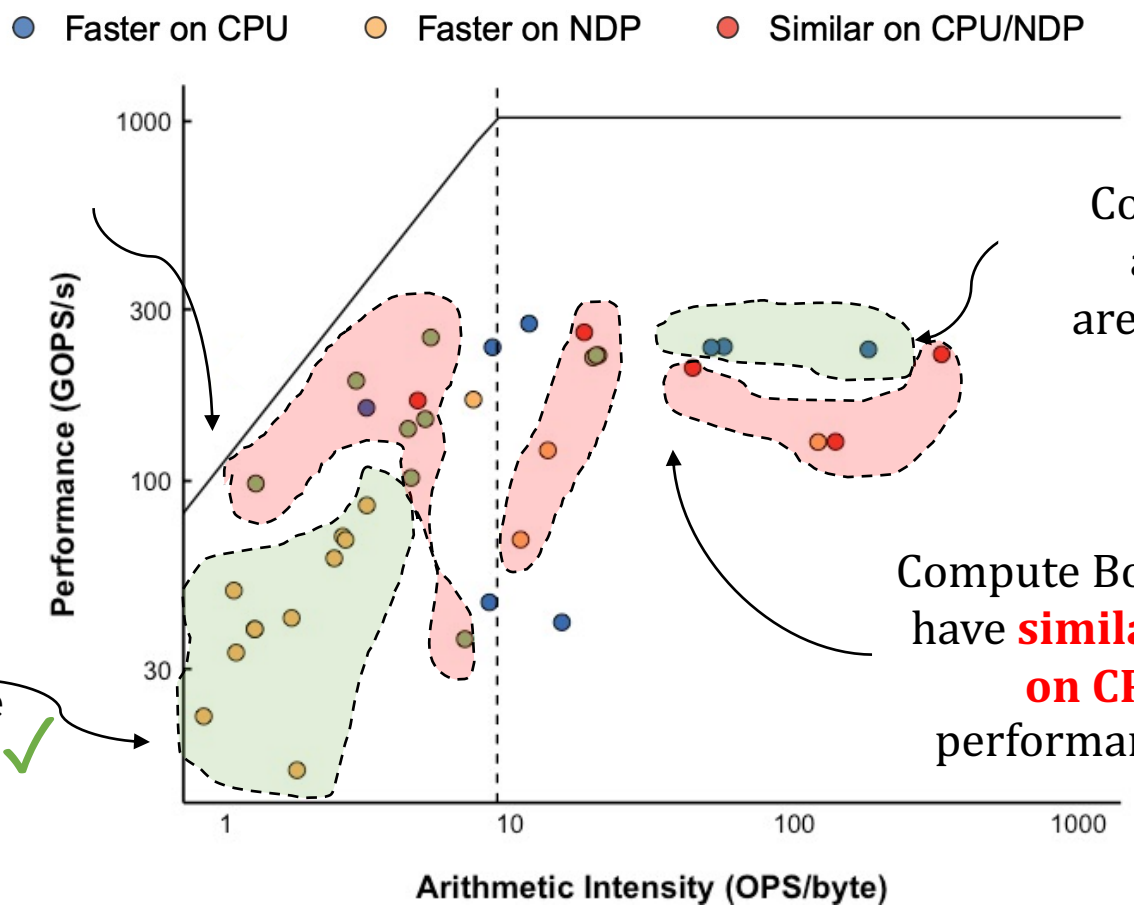
# Limitations of Prior Approaches (1/2)

- **Roofline model** → identifies when an application is *bounded* by **compute** or **memory** units



# Limitations of Prior Approaches (1/2)

- **Roofline model** → identifies when an application is *bounded* by **compute** or **memory** units




Memory Bound applications are **faster on CPU**, or performance *depends* ✗

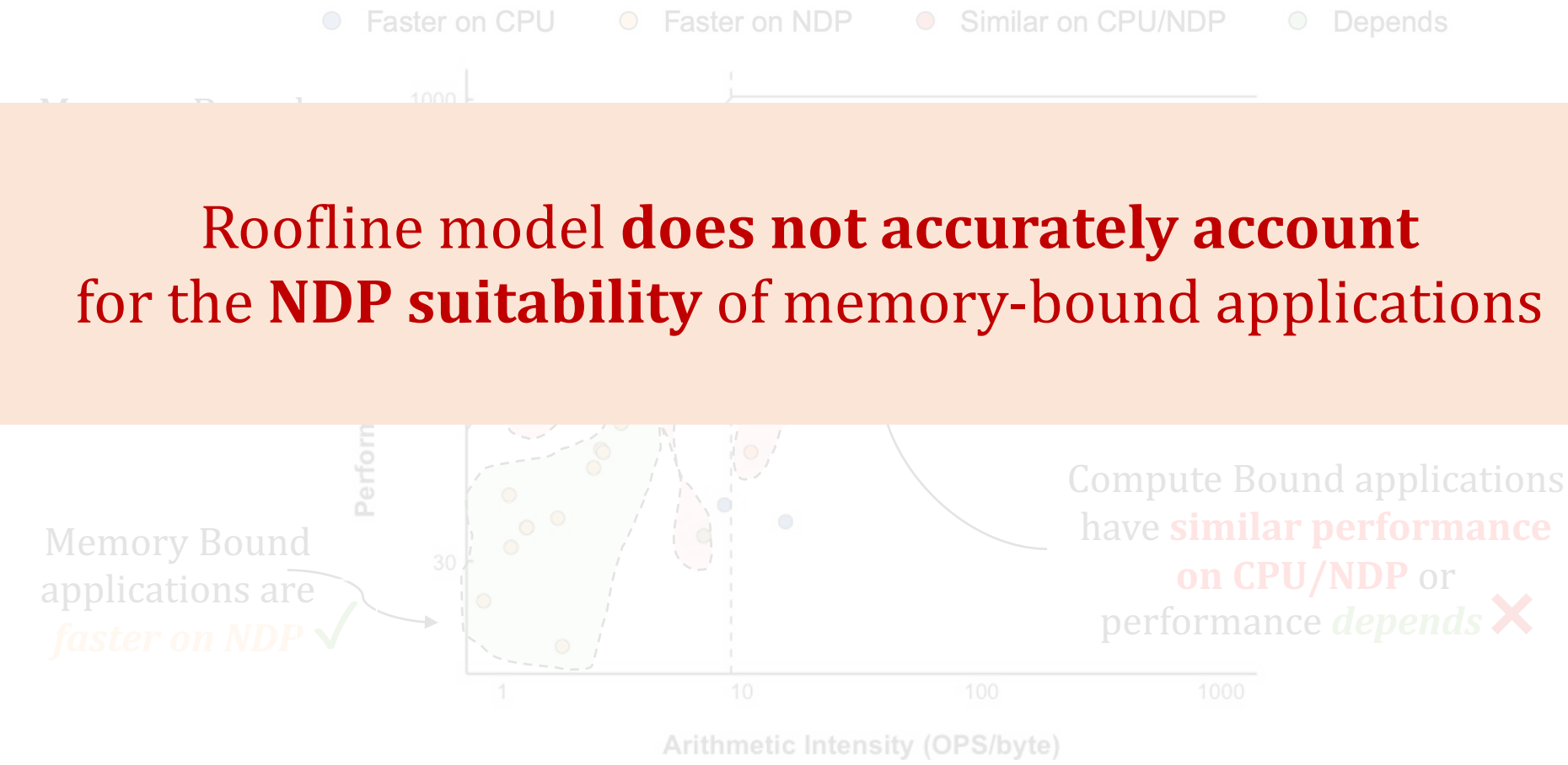
Compute Bound applications are *faster on CPU* ✓

Memory Bound applications are *faster on NDP* ✓

Compute Bound applications have **similar performance on CPU/NDP** or performance *depends* ✗

# Limitations of Prior Approaches (1/2)

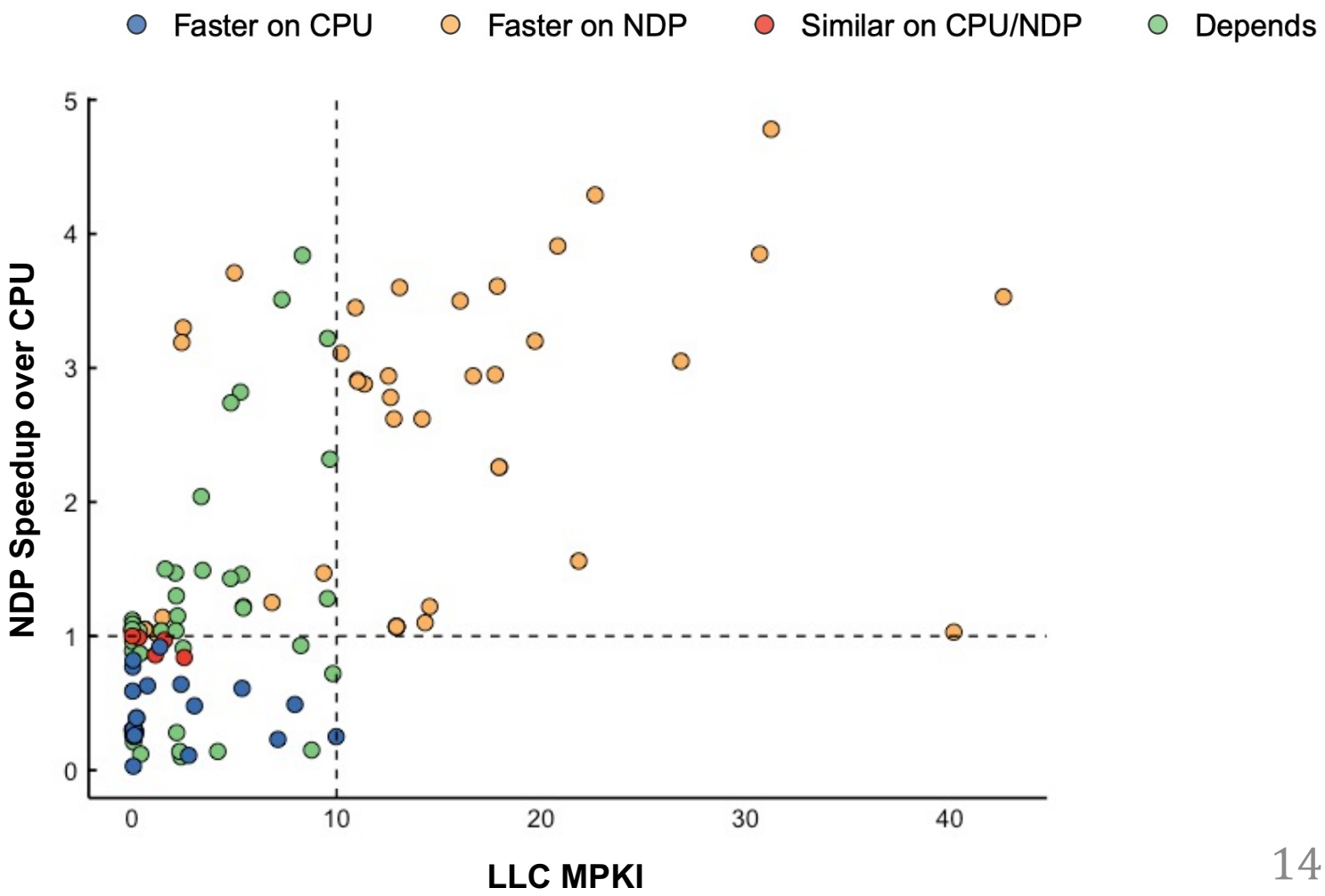
- **Roofline model** → identifies when an application is *bounded* by **compute** or **memory** units




Roofline model **does not accurately account** for the **NDP suitability** of memory-bound applications

# Limitations of Prior Approaches (2/2)

- Application with a last-level cache **MPKI > 10**
  → **memory intensive** and **benefits from NDP**

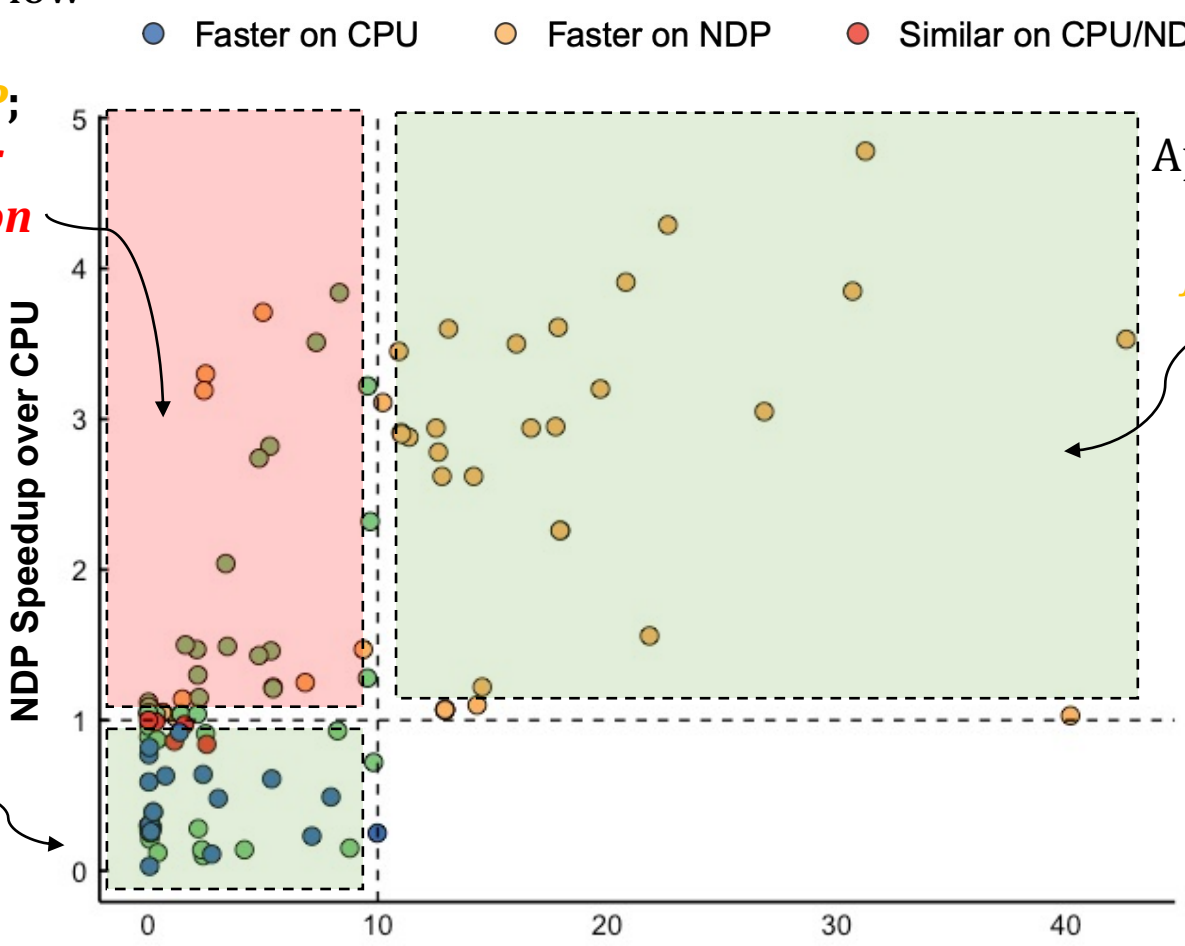


# Limitations of Prior Approaches (2/2)

- Application with a last-level cache **MPKI > 10** → **memory intensive** and **benefits from NDP**

Applications with low MPKI can be ***faster on NDP***; have ***similar performance on CPU/NDP*** or; performance can ***depends*** ✗

Applications with low MPKI are ***faster on CPU*** ✓

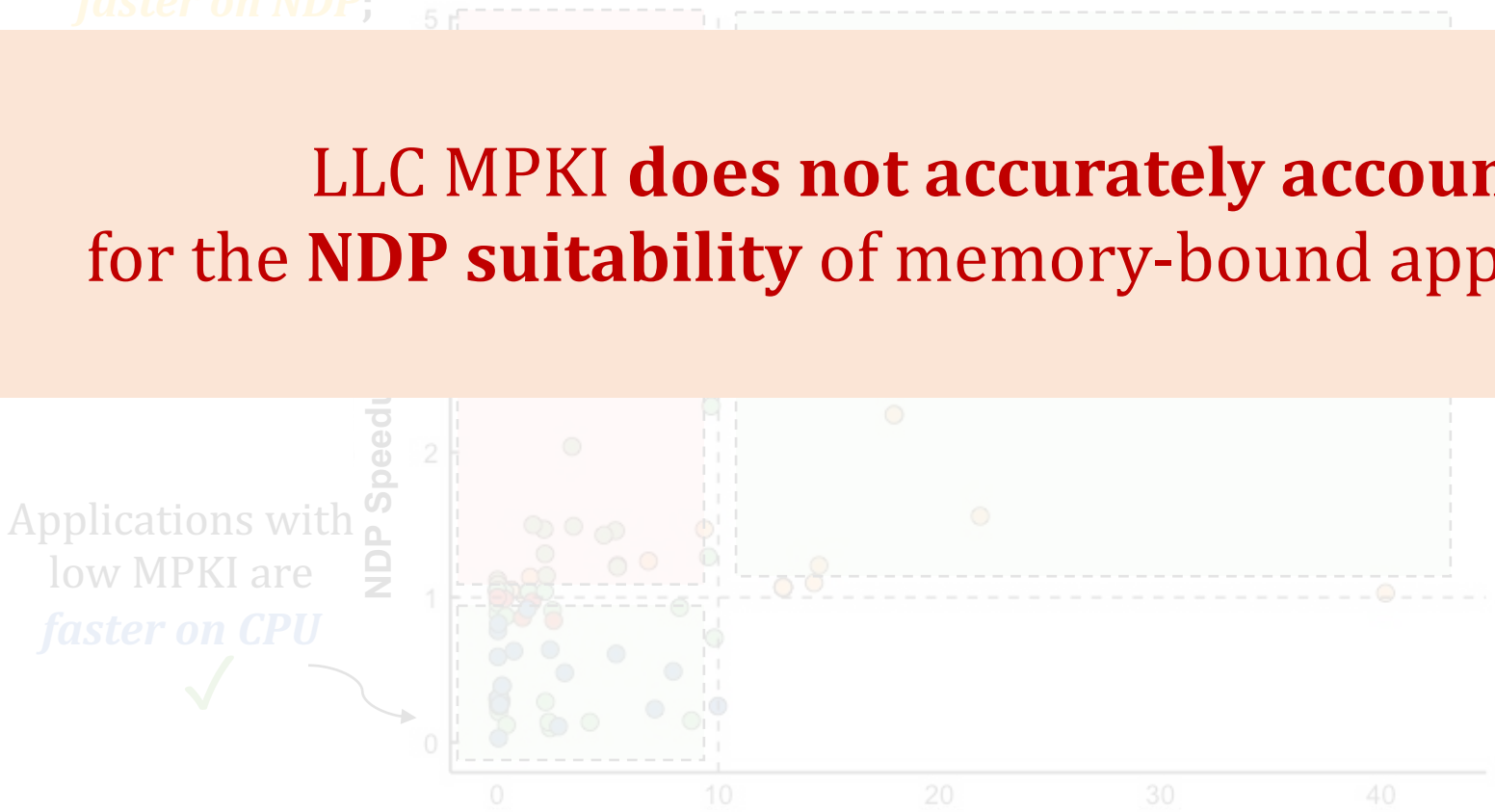



Applications with high MPKI are ***faster on NDP*** ✓

# Limitations of Prior Approaches (2/2)

- Application with a last-level cache **MPKI > 10**
  → **memory intensive** and **benefits from NDP**

Applications with low MPKI can be *faster on NDP*;



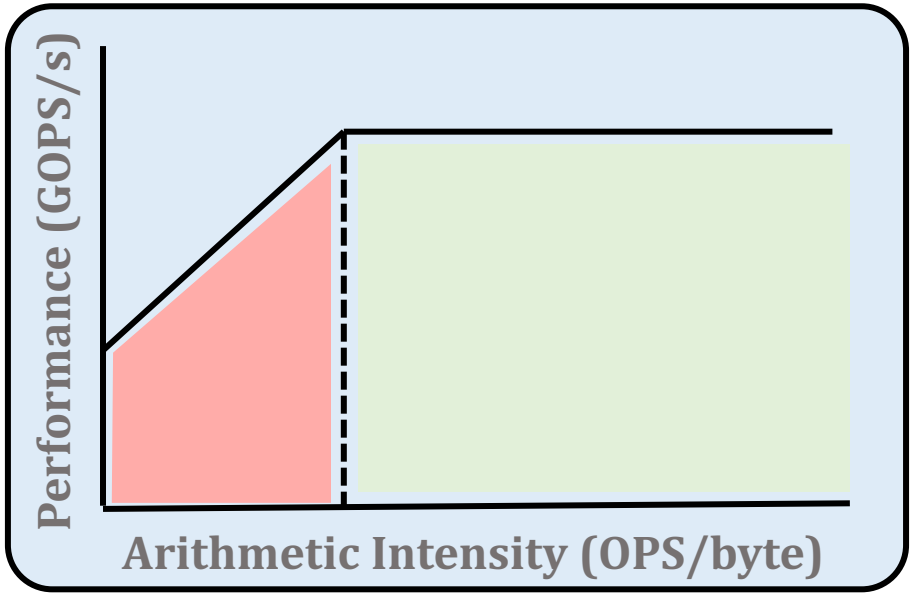

LLC MPKI **does not accurately account** for the **NDP suitability** of memory-bound applications

Applications with low MPKI are *faster on CPU* ✓

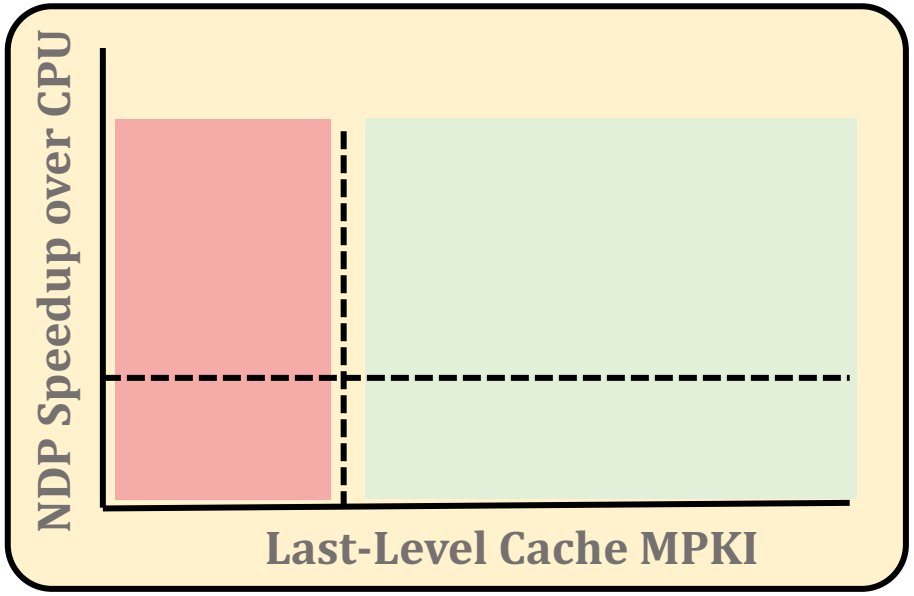
# Identifying Memory Bottlenecks

- Multiple approaches to identify applications that:
  - suffer from data movement bottlenecks
  - take advantage of NDP
- Existing approaches are not comprehensive enough

**Roofline model**




**High LLC MPKI**



# The Problem

- Multiple approaches to identify applications that:
  - suffer from data movement bottlenecks
  - take advantage of NDP

No available methodology can comprehensively:

- **identify** data movement bottlenecks
- **correlate** them with the **most suitable** data movement mitigation mechanism

# Our Goal

- **Our Goal:** develop a methodology to:
  - methodically identify sources of data movement bottlenecks
  - comprehensively compare compute- and memory-centric data movement mitigation techniques

# Outline

1. Data Movement Bottlenecks
2. **Methodology Overview**
3. Application Profiling
4. Locality-Based Clustering
5. Memory Bottleneck Analysis
6. Case Studies

# Key Approach

- New workload characterization methodology to analyze:
  - data movement bottlenecks
  - suitability of different data movement mitigation mechanisms

- Two main profiling strategies:

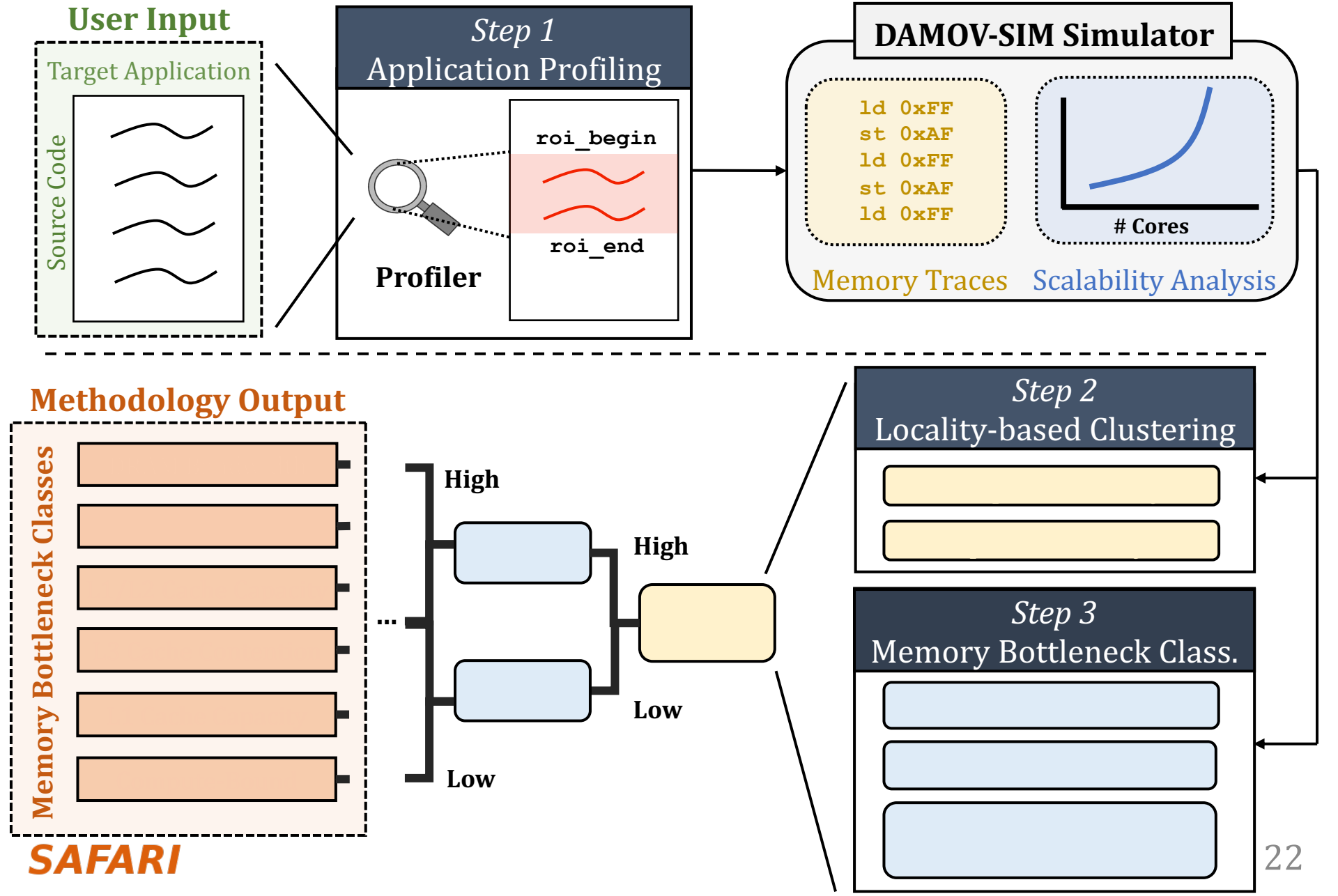
**Architecture-independent profiling:**

characterizes the memory behavior independently of the underlying hardware

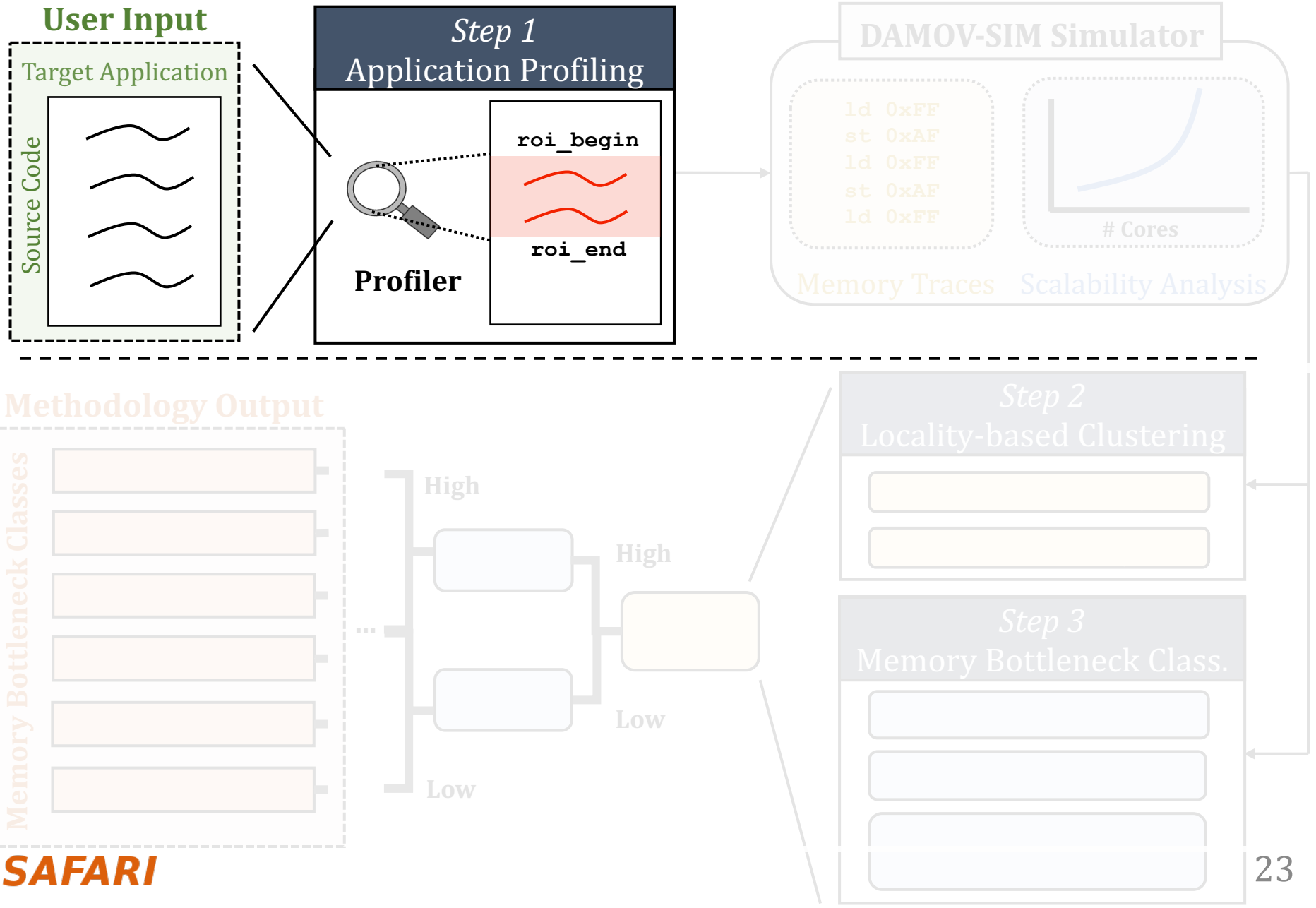
**Architecture-dependent profiling:**

evaluates the impact of the system configuration on the memory behavior

# Methodology Overview

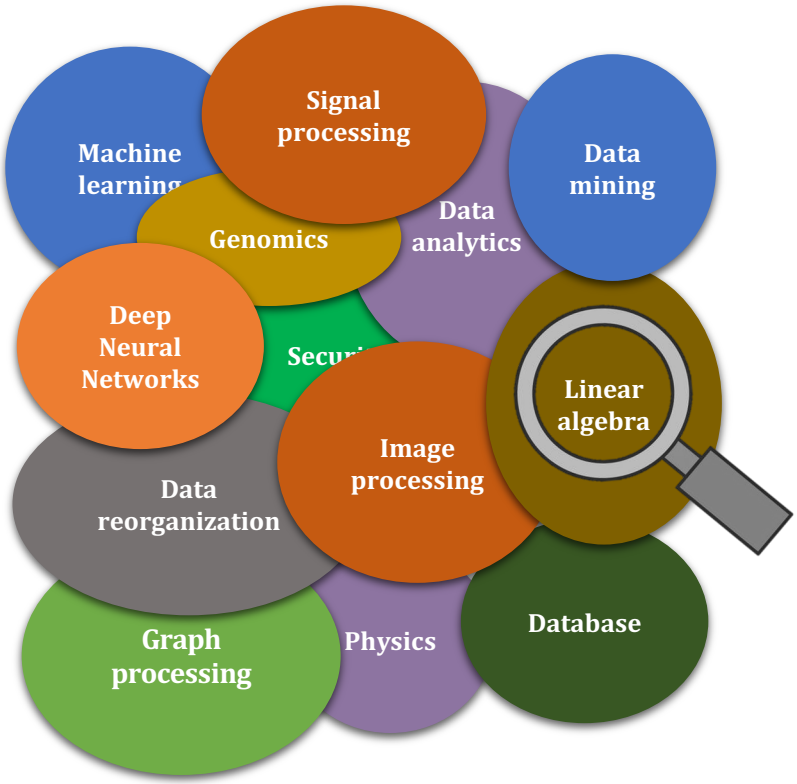


# Methodology Overview



# Step 1: Application Profiling

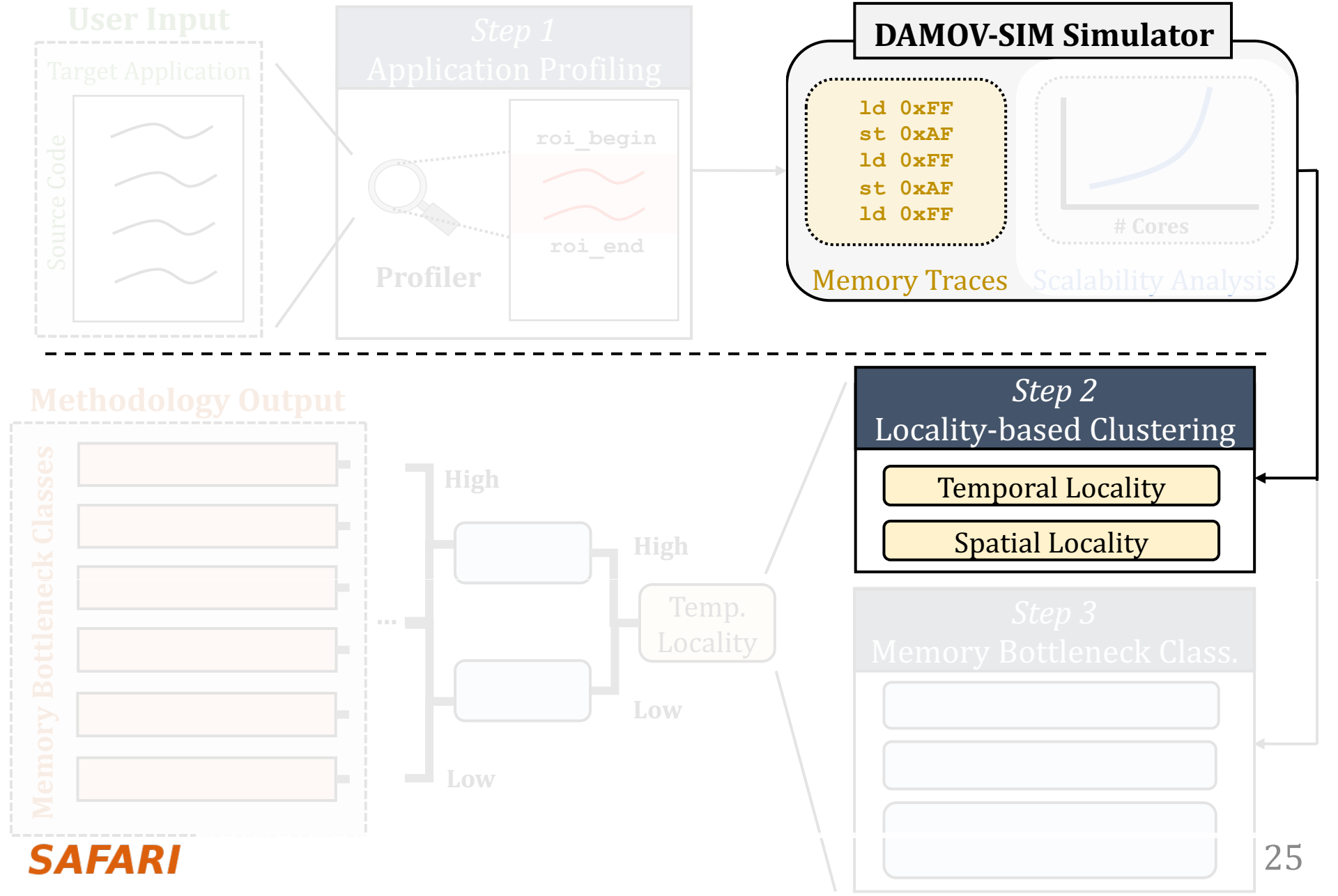
Goal: Identify **application functions** that suffer from data movement bottlenecks




Hardware Profiling Tool:
**Intel VTune**

**MemoryBound:**
CPU is stalled due to load/store

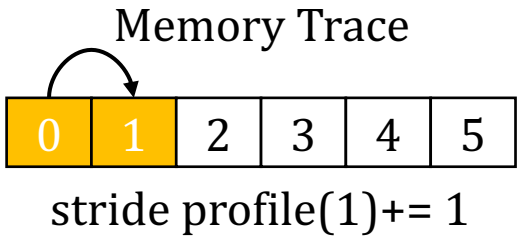
# Methodology Overview

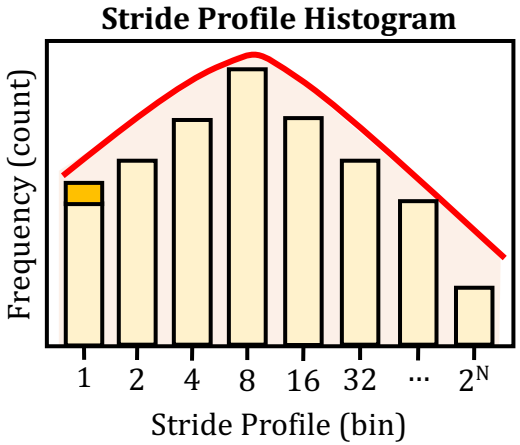


# Step 2: Locality-Based Clustering

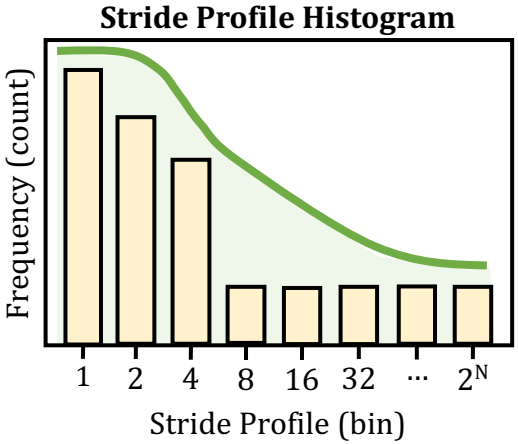
- **Goal:** analyze application's memory characteristics

## Spatial Locality[7]




**Low spatial locality**




**High spatial locality**



[7] Weinberg+, "Quantifying Locality in the Memory Access Patterns of HPC Applications," SC, 2005

# Step 2: Locality-Based Clustering

- **Goal:** analyze application's memory characteristics

## Spatial Locality[7]





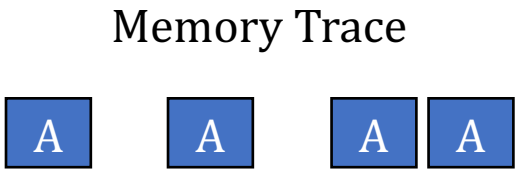


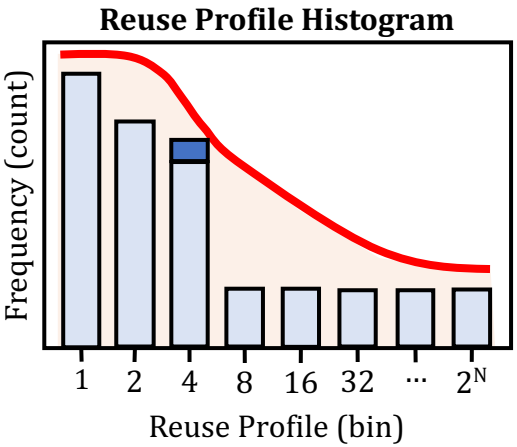
Low spatial locality



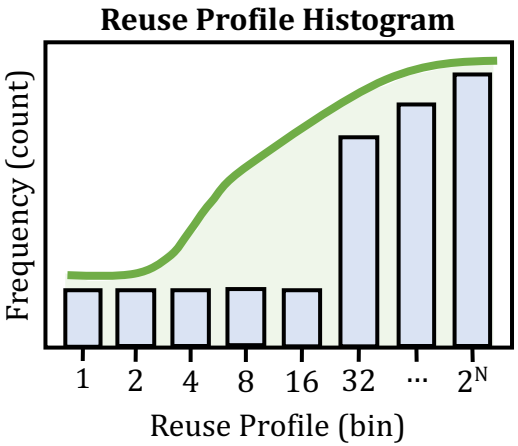

High spatial locality

## Temporal Locality[7]




Low temporal locality

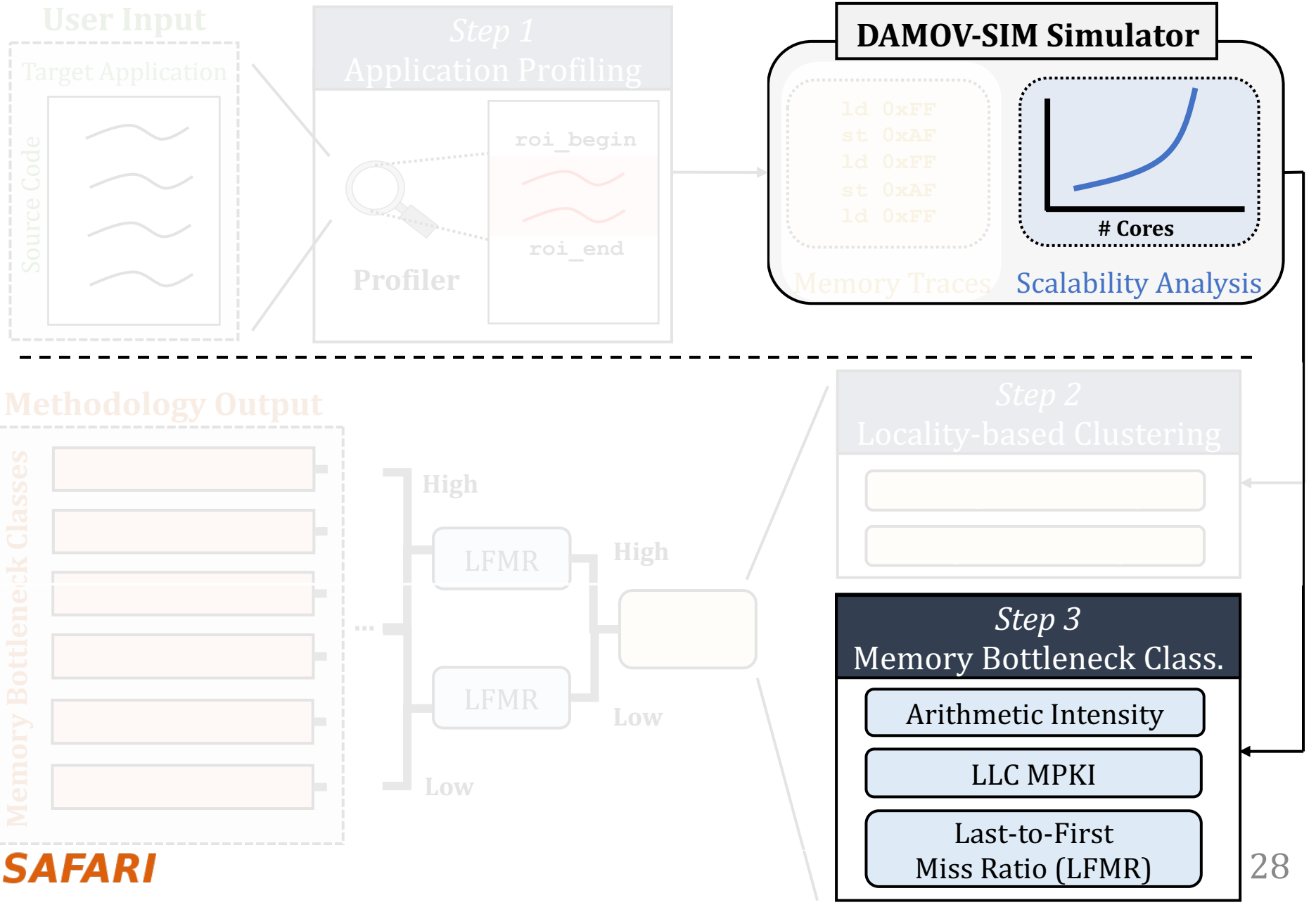



High temporal locality



[7] Weinberg+, "Quantifying Locality in the Memory Access Patterns of HPC Applications," SC, 2005

# Methodology Overview



# Step 3: Memory Bottleneck Classification (1/2)

## Arithmetic Intensity (AI)

- floating-point/arithmetic operations per L1 cache lines accessed

→ **shows computational intensity per memory request**

## LLC Misses-per-Kilo-Instructions (MPKI)

- LLC misses per one thousand instructions

→ **shows memory intensity**

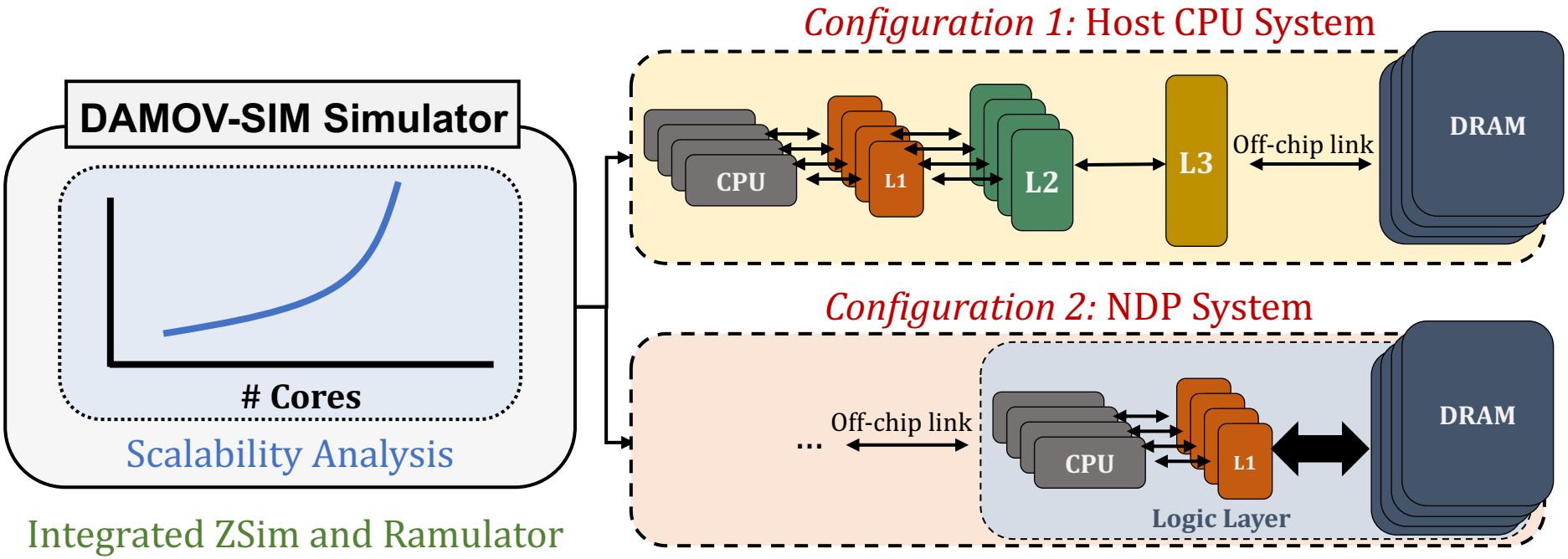
## Last-to-First Miss Ratio (LFMR)

- LLC misses per L1 misses

→ **shows if an application benefits from L2/L3 caches**

# Step 3: Memory Bottleneck Classification (2/2)

- **Goal:** identify the specific sources of data movement bottlenecks




- Scalability Analysis:
  - 1, 4, 16, 64, and 256 out-of-order/in-order host and NDP CPU cores
  - 3D-stacked memory as main memory



**DAMOV-SIM:** https://github.com/CMU-SAFARI/DAMOV

# Outline

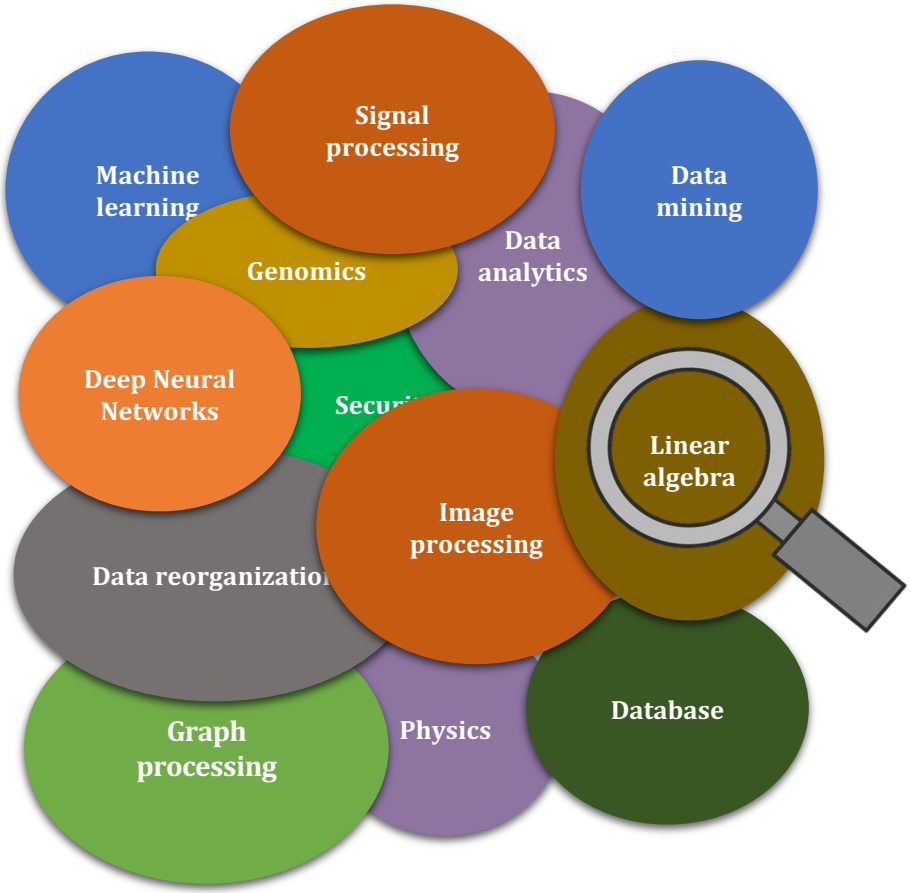
1. Data Movement Bottlenecks
2. Methodology Overview
3. **Application Profiling**
4. Locality-Based Clustering
5. Memory Bottleneck Analysis
6. Case Studies

# Step 1: Application Profiling

- We analyze 345 applications from distinct domains:

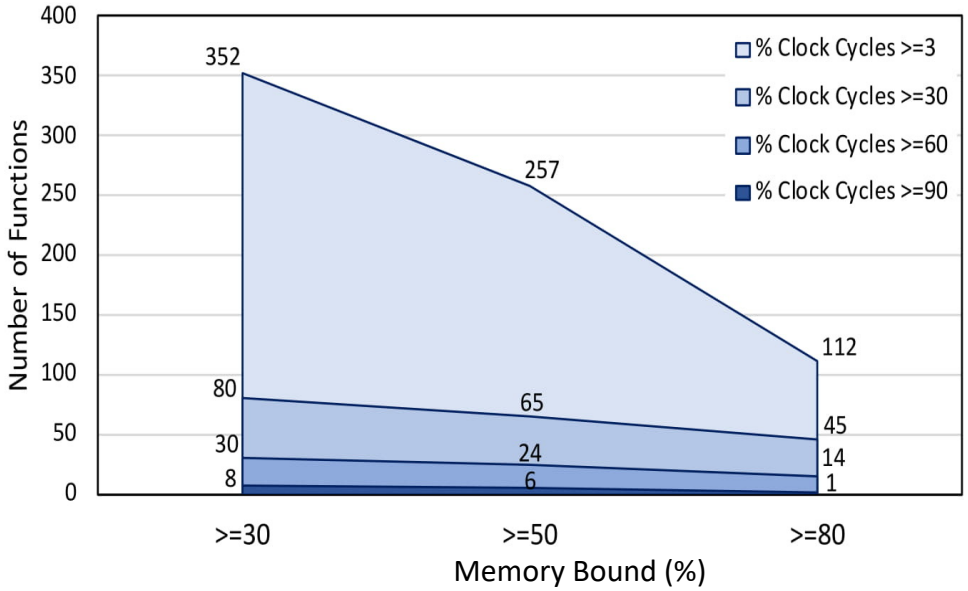
- Graph Processing
- Deep Neural Networks
- Physics
- High-Performance Computing
- Genomics
- Machine Learning
- Databases
- Data Reorganization
- Image Processing
- Map-Reduce
- Benchmarking
- Linear Algebra

...



# Memory Bound Functions

- We analyze 345 applications from distinct domains
- Selection criteria: clock cycles > **3%** and Memory Bound > **30%**




- We find 144 functions from a total of 77K functions and select:
  - 44 functions → **apply steps 2 and 3**
  - 100 functions → **validation**

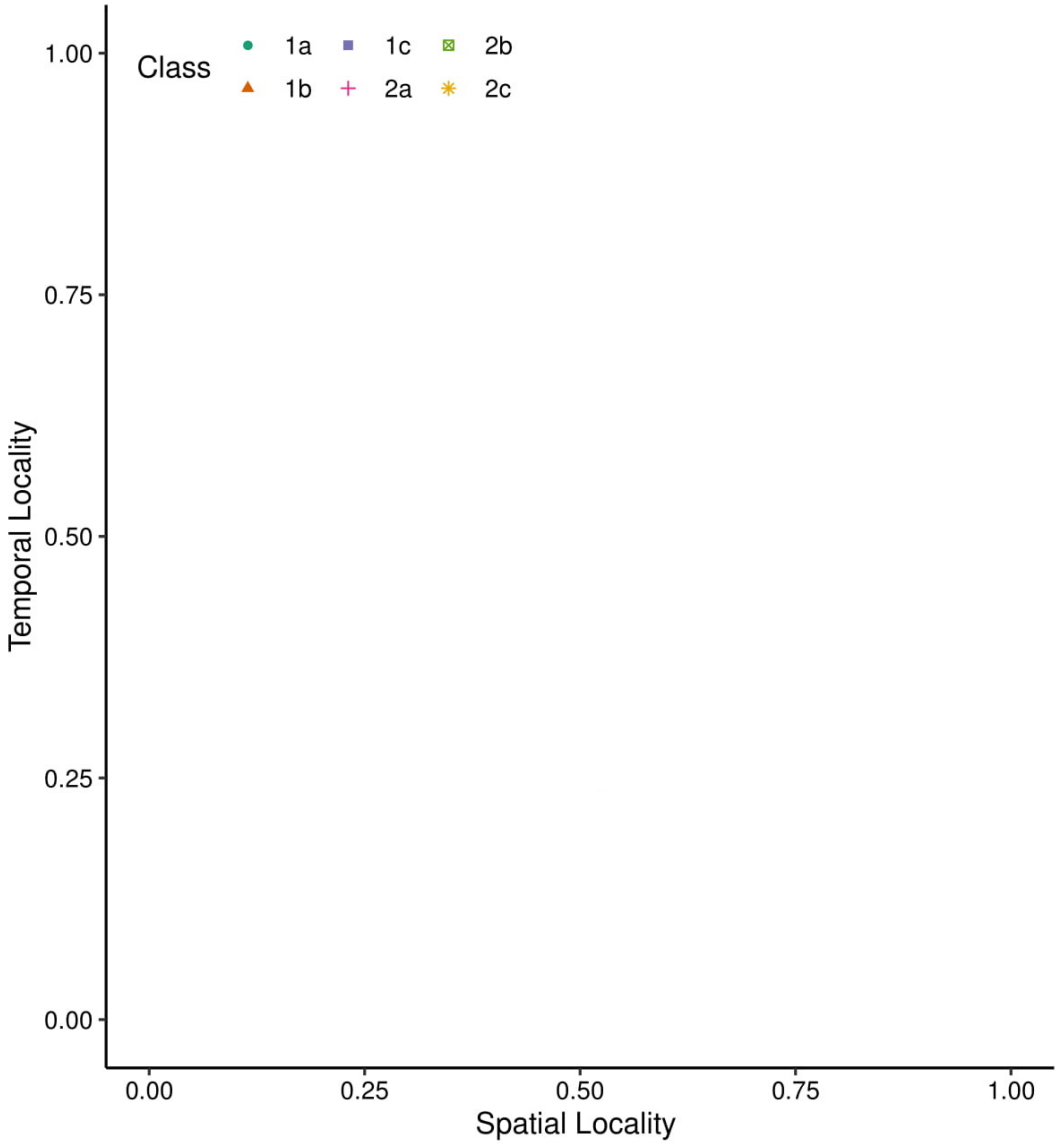
# Outline

1. Data Movement Bottlenecks
2. Methodology Overview
3. Application Profiling
4. **Locality-Based Clustering**
5. Memory Bottleneck Analysis
6. Case Studies

# Step 2: Locality-Based Clustering

We use K-means to cluster the applications across both **spatial and temporal locality**, forming two groups

1. Low locality applications (in orange)
2. High locality applications (in blue)



# Step 2: Locality-Based Clustering

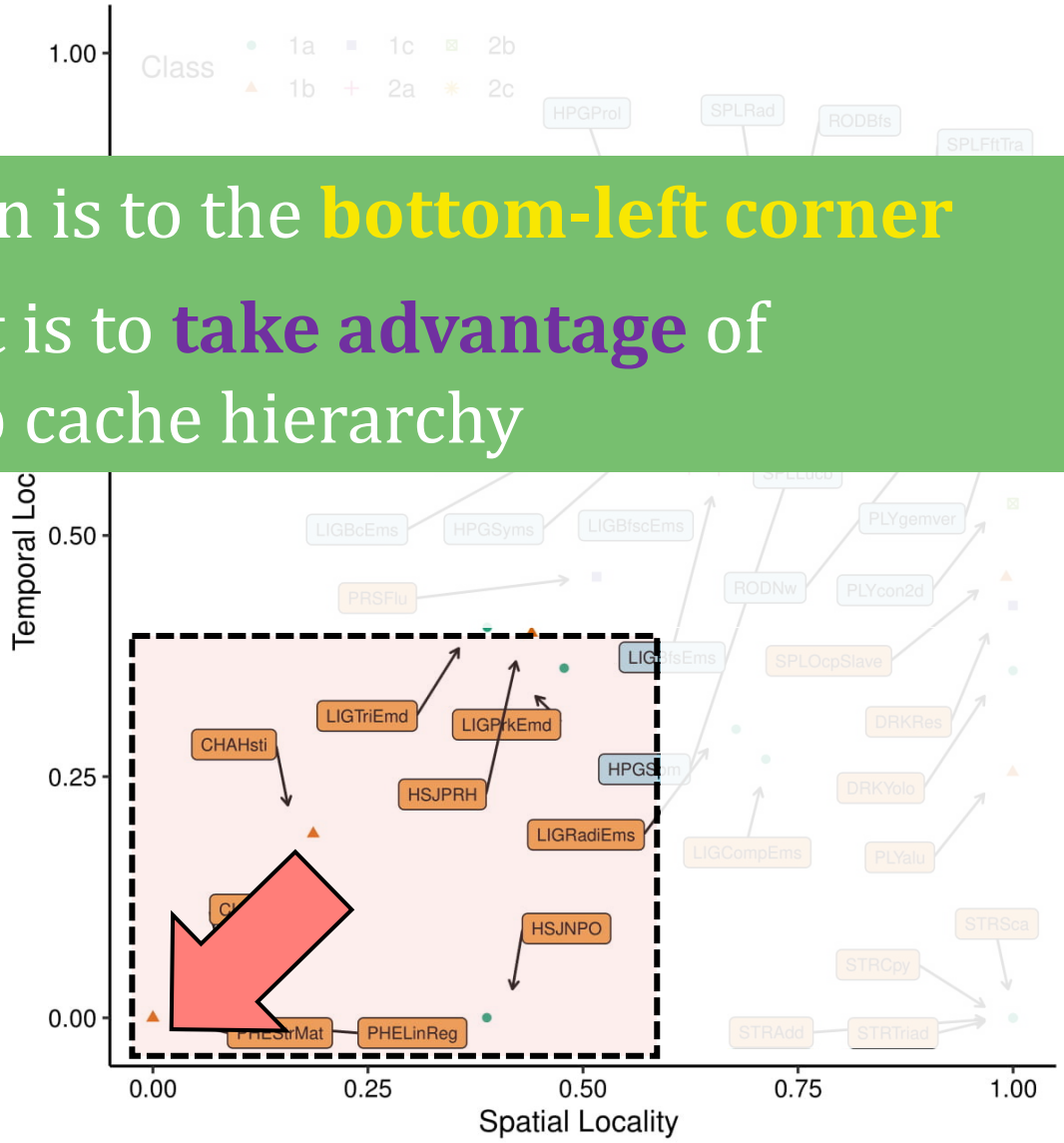
We use K-means to cluster the applications across both

The closer a function is to the **bottom-left corner**

→ less likely it is to **take advantage** of a deep cache hierarchy

applications (in orange)

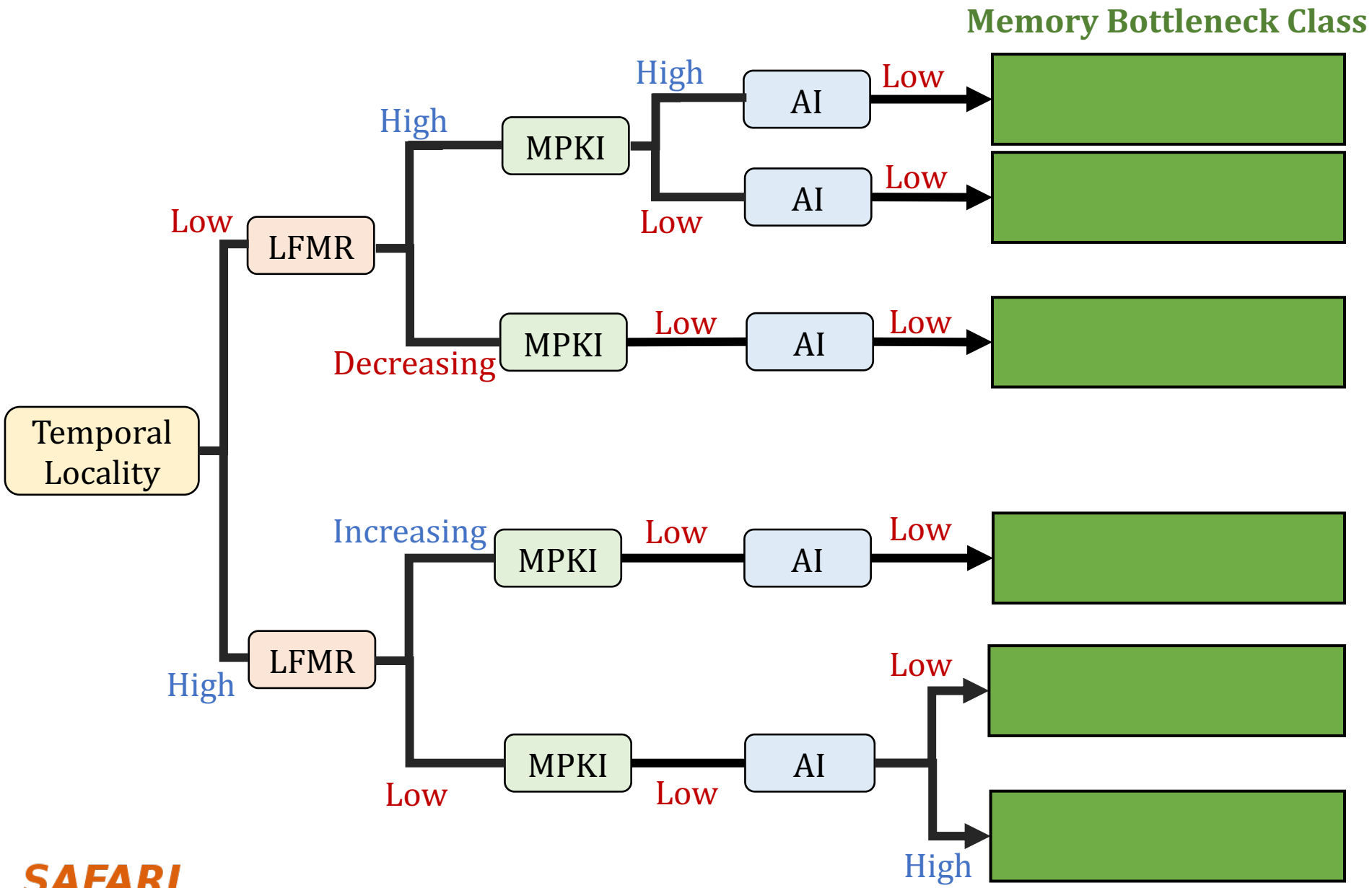
2. High locality applications (in blue)



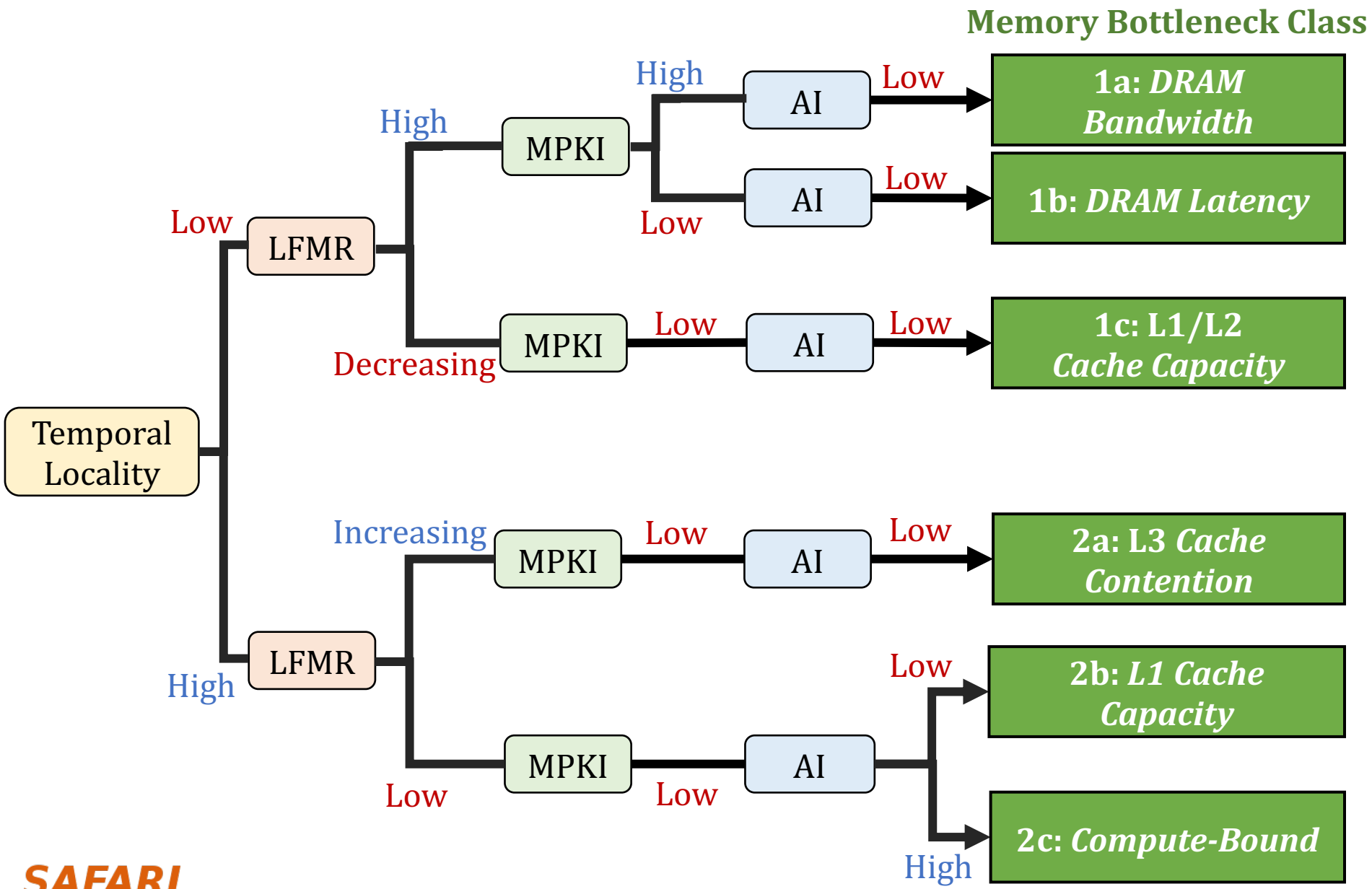
# Outline

1. Data Movement Bottlenecks
2. Methodology Overview
3. Application Profiling
4. Locality-Based Clustering
5. **Memory Bottleneck Analysis**
6. Case Studies

# Step 3: Memory Bottleneck Analysis



# Step 3: Memory Bottleneck Analysis



# Step 3: Memory Bottleneck Analysis

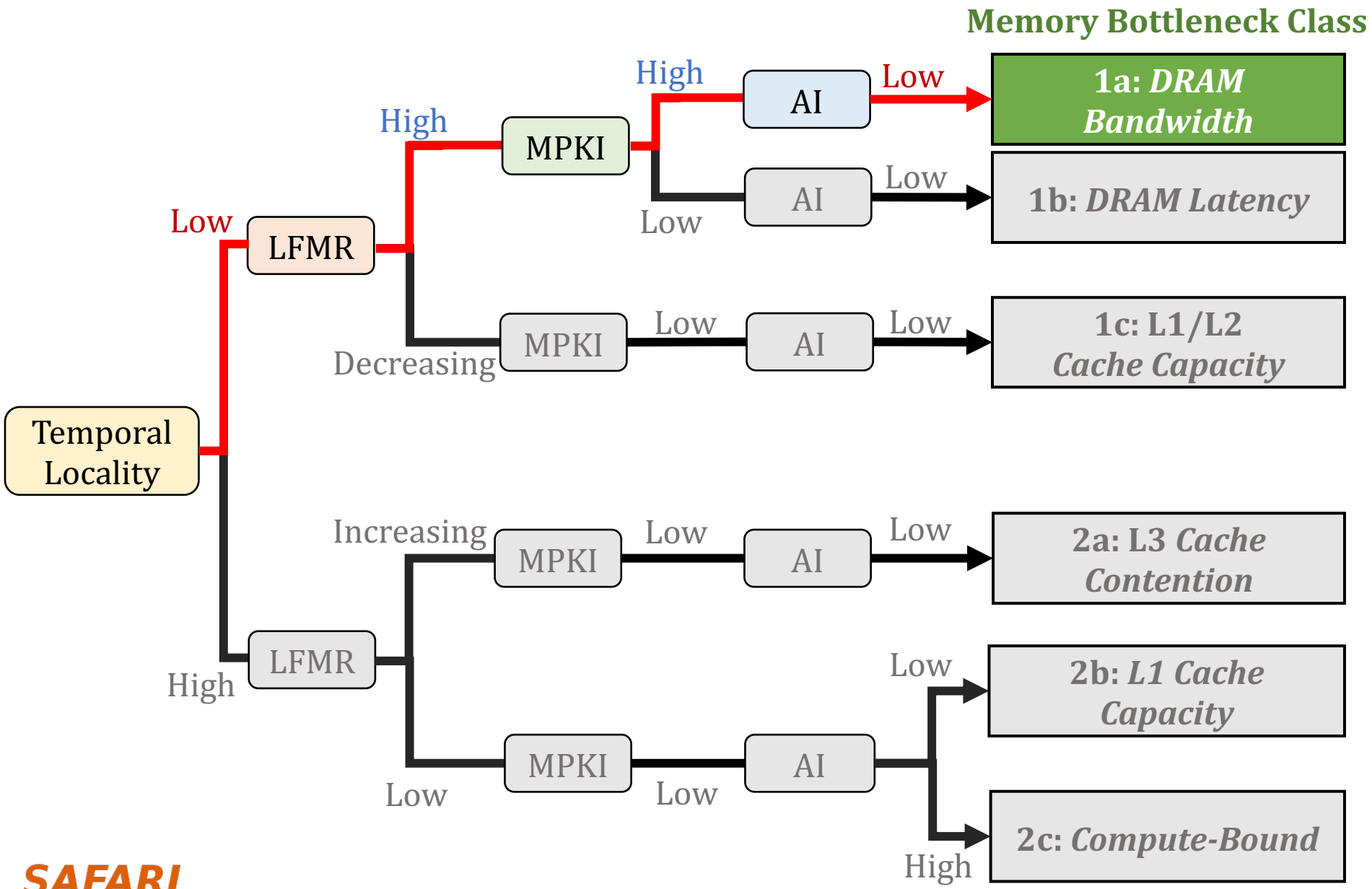
**Six classes of data movement bottlenecks:**

each class ↔ data movement mitigation mechanism

**Memory Bottleneck Class**

- **1a: *DRAM Bandwidth***
- **1b: *DRAM Latency***
- **1c: L1/L2 *Cache Capacity***
- **2a: L3 *Cache Contention***
- **2b: *L1 Cache Capacity***
- **2c: *Compute-Bound***

# Step 3: Memory Bottleneck Analysis

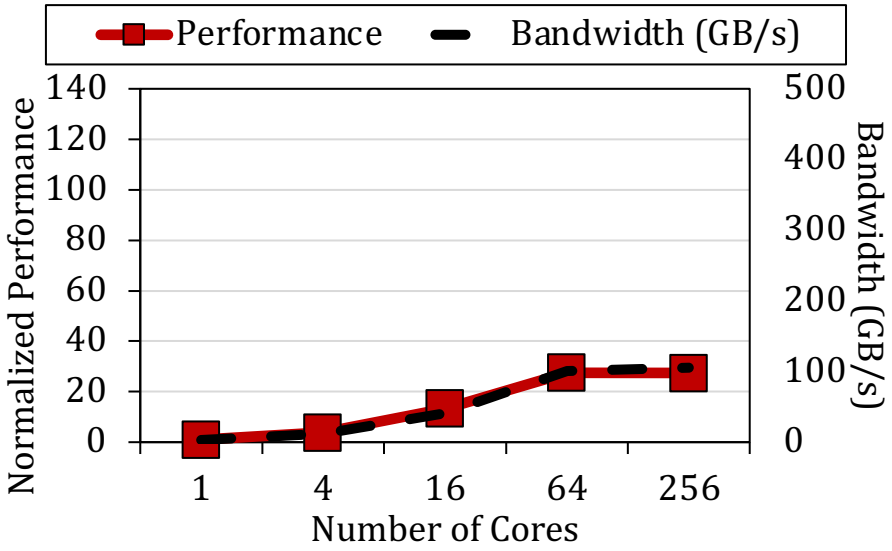


# Class 1a: DRAM Bandwidth Bound (1/2)

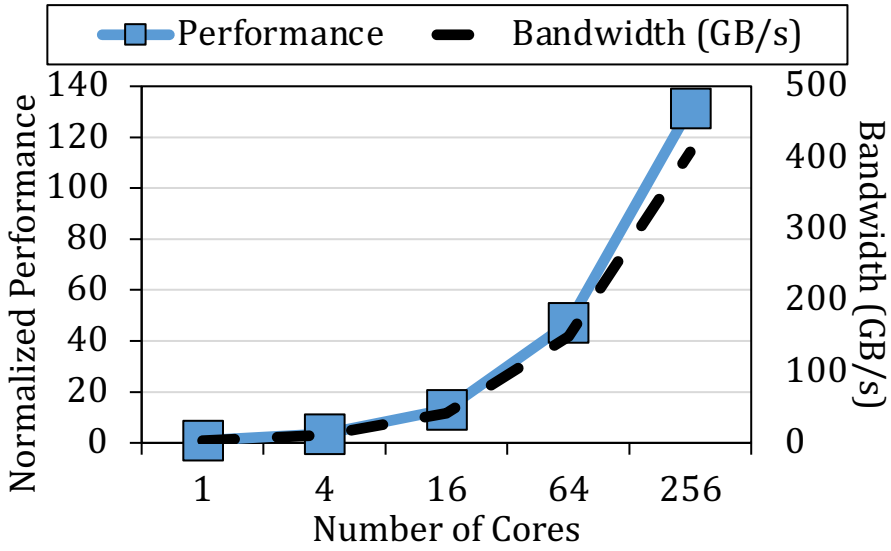
- High MPKI → high memory pressure
- Host scales well until bandwidth saturates
- NDP scales **without saturating** alongside attained bandwidth

Temp. Loc: *low*
LFMR: *high*
MPKI: *high*
AI: *low*

**Host**




**NDP**

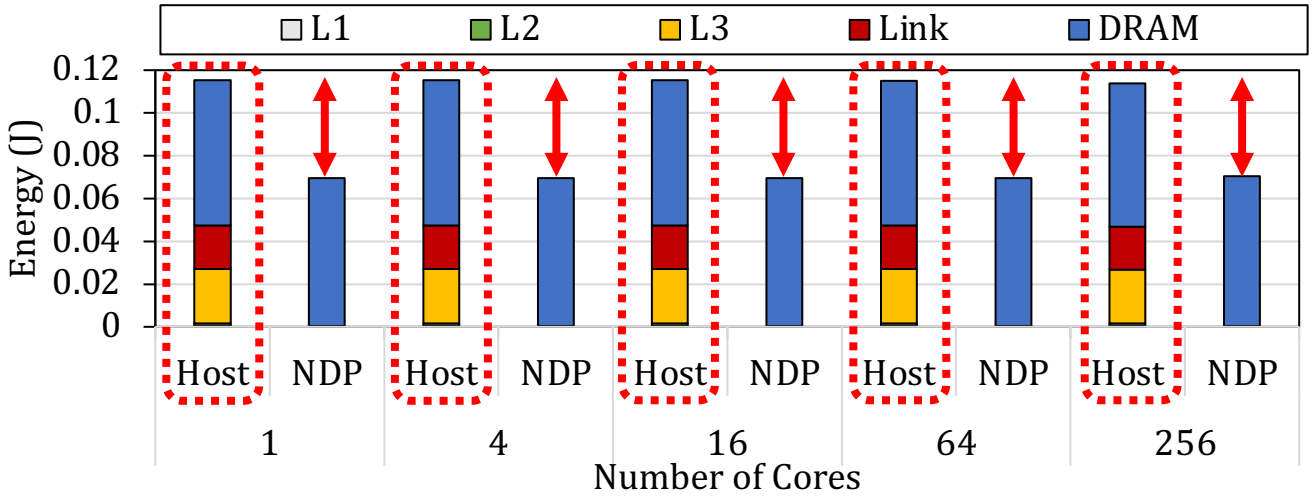



**DRAM bandwidth bound applications:**
NDP does better because of the higher internal DRAM bandwidth

# Class 1a: DRAM Bandwidth Bound (2/2)

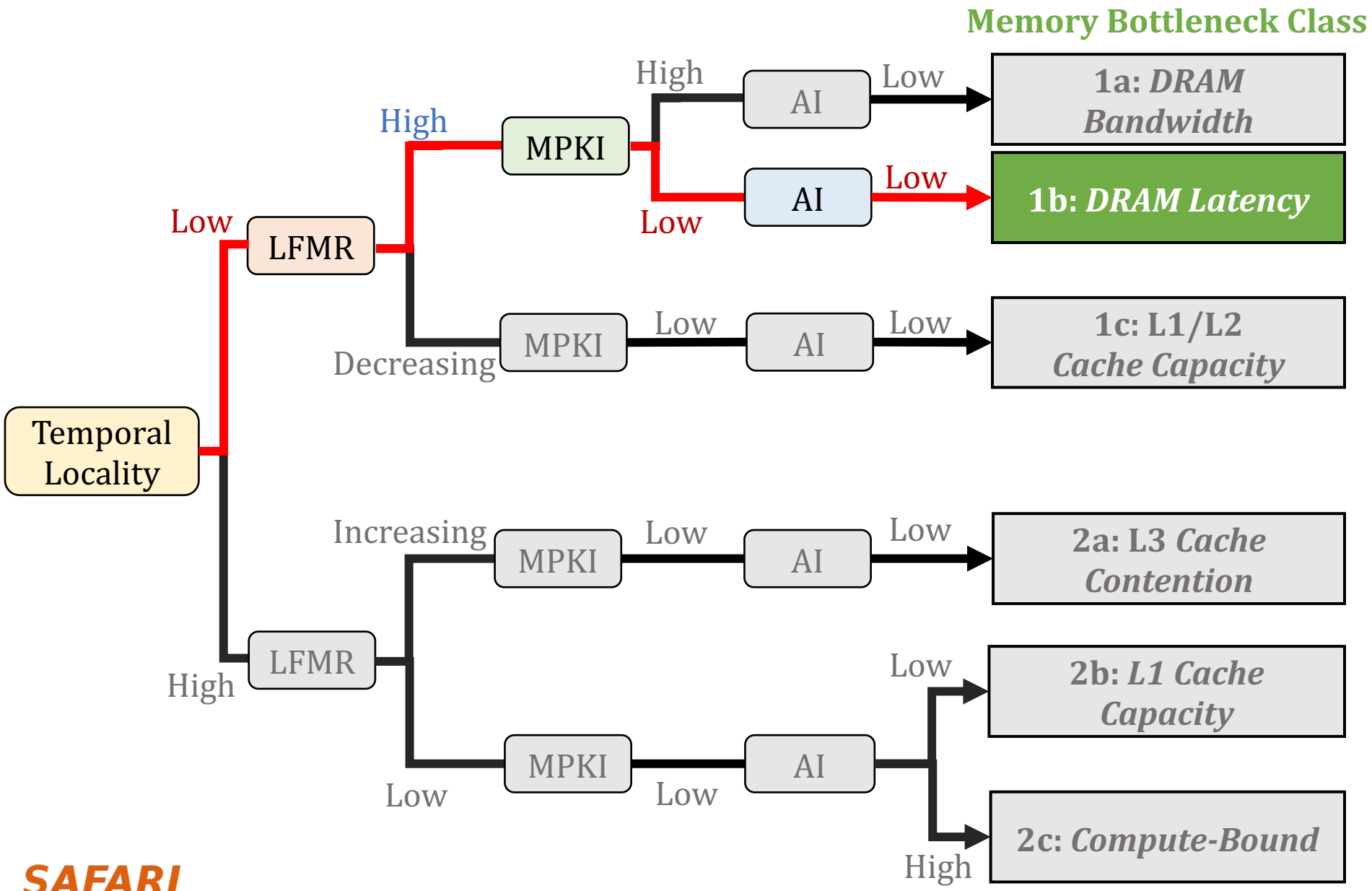
Temp. Loc: *low*
LFMR: *high*
MPKI: *high*
AI: *low*

- High LFMR → L2 and L3 caches are inefficient
- Host's energy consumption is dominated by cache look-ups and off-chip data transfers
- NDP provides **large system energy reduction** since it does not access L2, L3, and off-chip links




**DRAM bandwidth bound applications:**
NDP does better because it eliminates off-chip I/O traffic

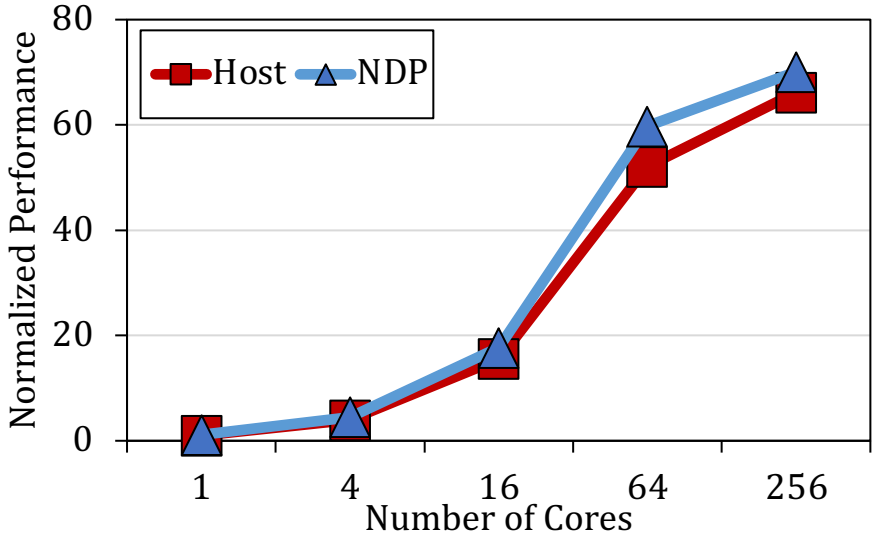
# Step 3: Memory Bottleneck Analysis

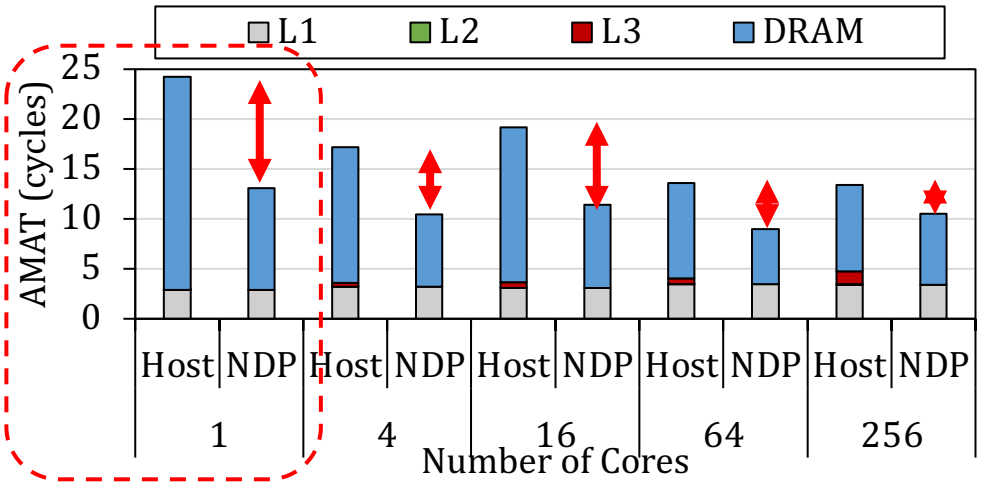


# Class 1b: DRAM Latency Bound

- High LFMR → L2 and L3 caches are inefficient
- Host scales well but NDP performance is always higher
- NDP performs better than host because of its **lower memory access latency**

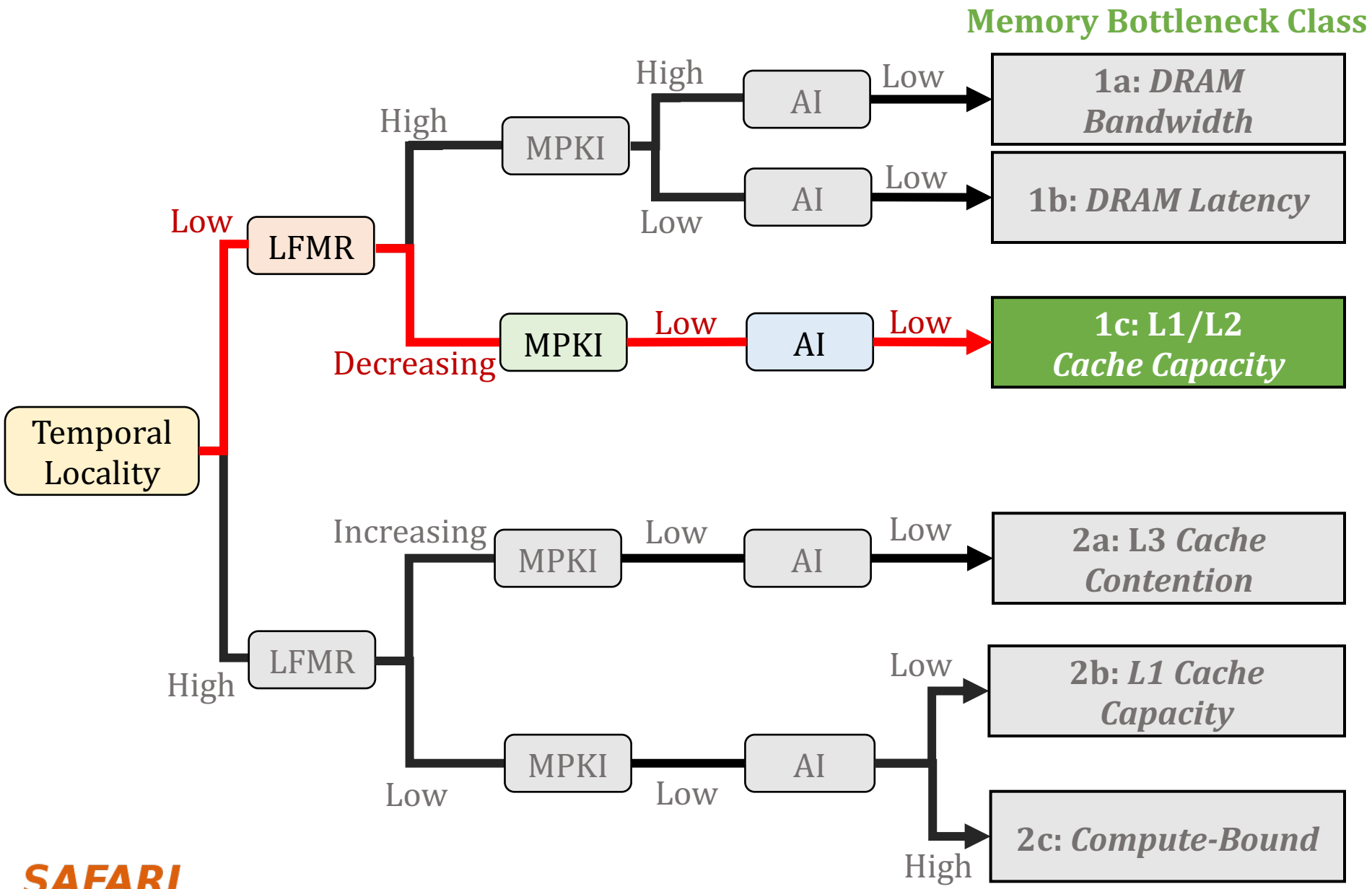
Temp. Loc: *low*
LFMR: *high*
MPKI: *low*
AI: *low*




**DRAM latency bound applications:**
host performance is hurt by the cache hierarchy and off-chip link

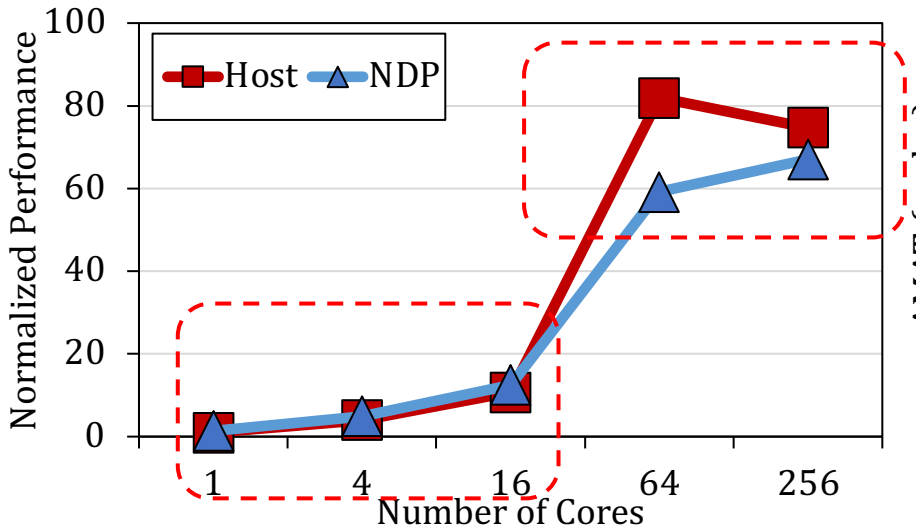
# Step 3: Memory Bottleneck Analysis

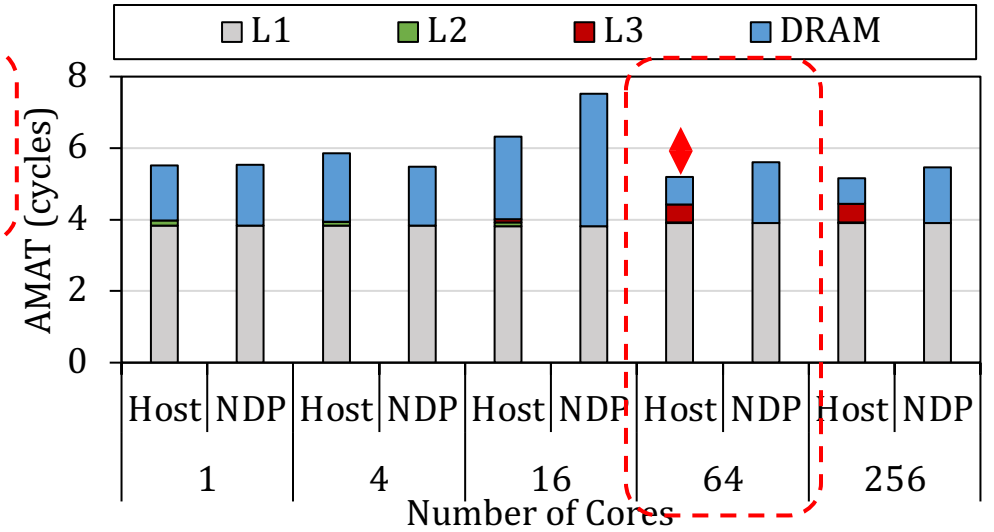


# Class 1c: L1/L2 Cache Capacity

- Decreasing LFMR → L2/L3 caches turn efficient
- NDP scales better than the host at low core counts
- Host scales better than NDP at high core counts
- Host performs better than NDP at high core counts since it **reduces memory access latency via data caching**

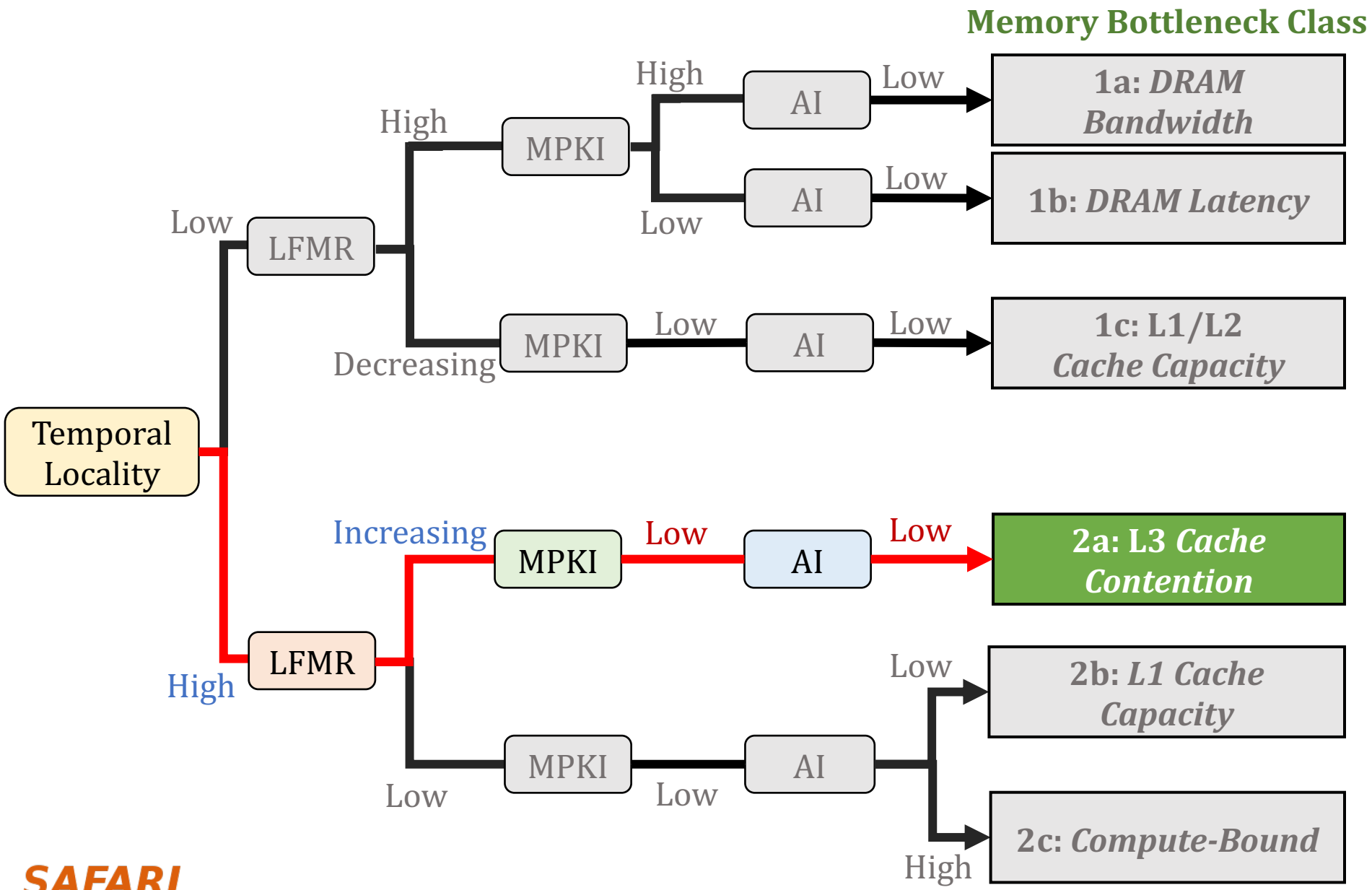
Temp. Loc: *low*
LFMR: *decreasing*
MPKI: *low*
AI: *low*




**L1/L2 cache capacity bottlenecked applications:**
NDP is higher performance when the aggregated cache size is small

# Step 3: Memory Bottleneck Analysis

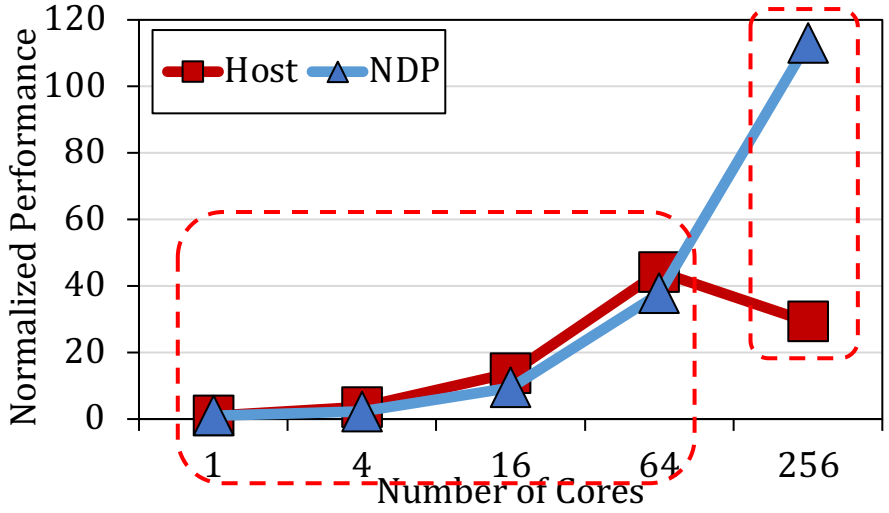


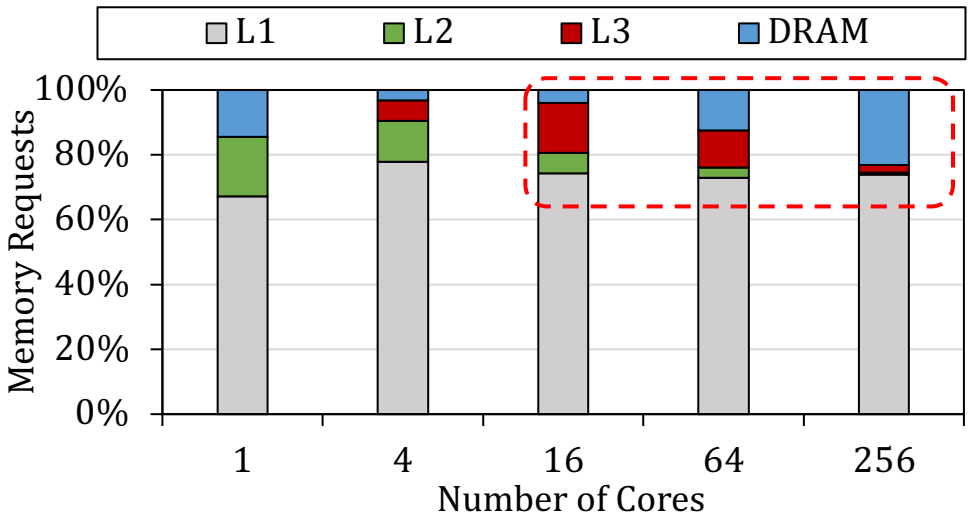
# Class 2a: L3 Cache Contention

- Increasing LFMR → L2/L3 caches turn inefficient
- Host scales better than the NDP at low core counts
- NDP scales better than host at high core counts

Temp. Loc: *high*
LFMR: *increasing*
MPKI: *low*
AI: *low*

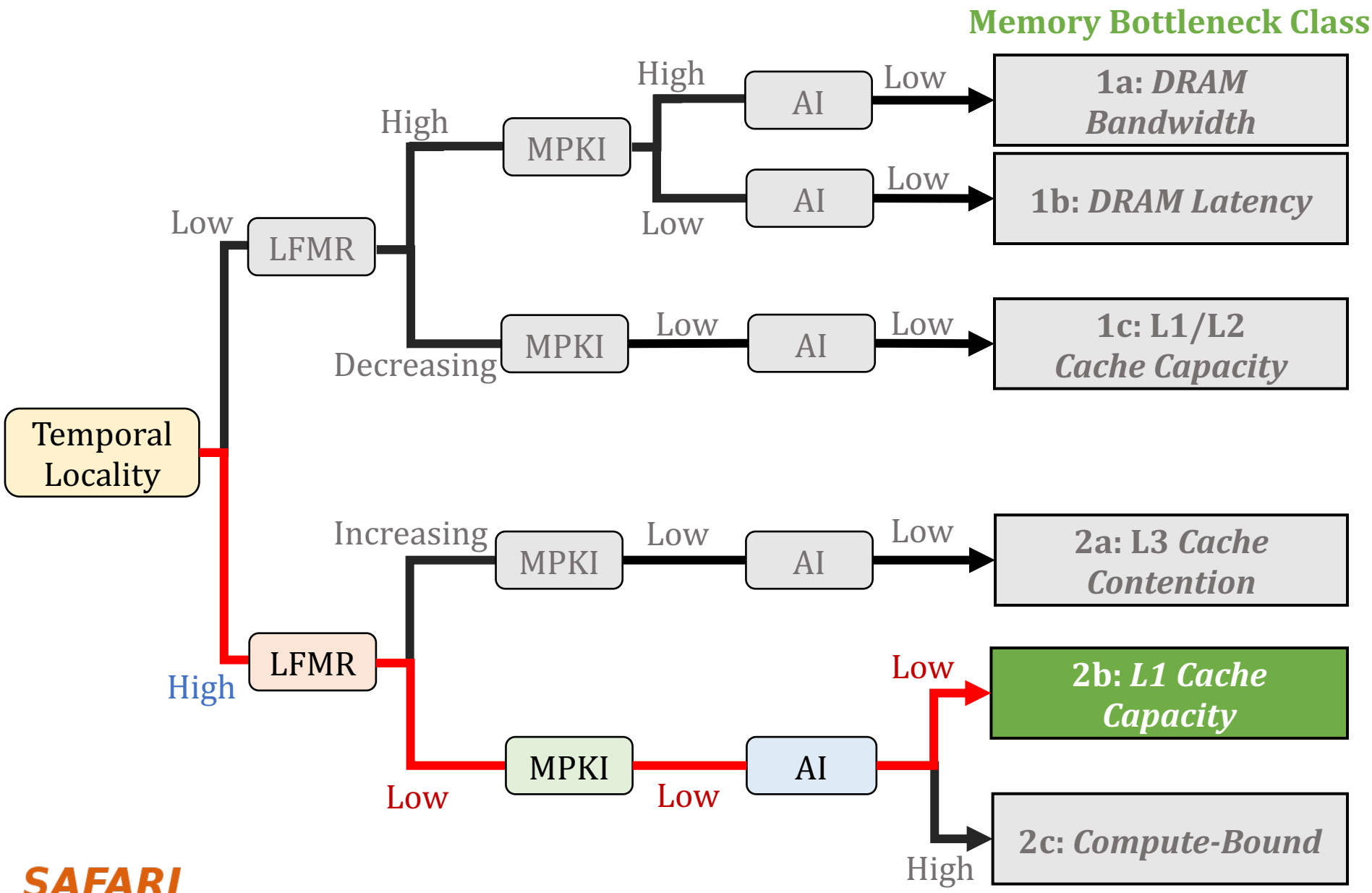
- NDP performs better than host at high core counts since it **reduces memory access latency**




**L3 cache contention bottlenecked applications:**
at high core counts, applications turn into DRAM latency-bound

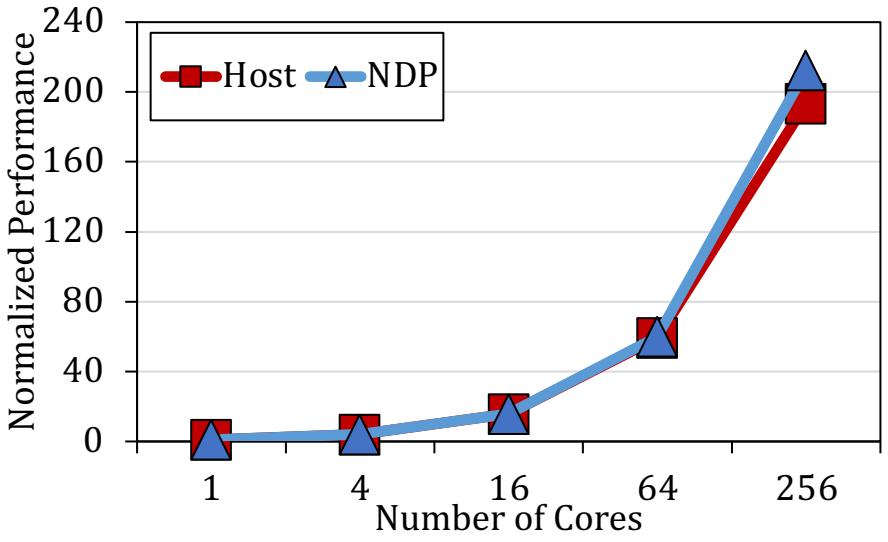
# Step 3: Memory Bottleneck Analysis

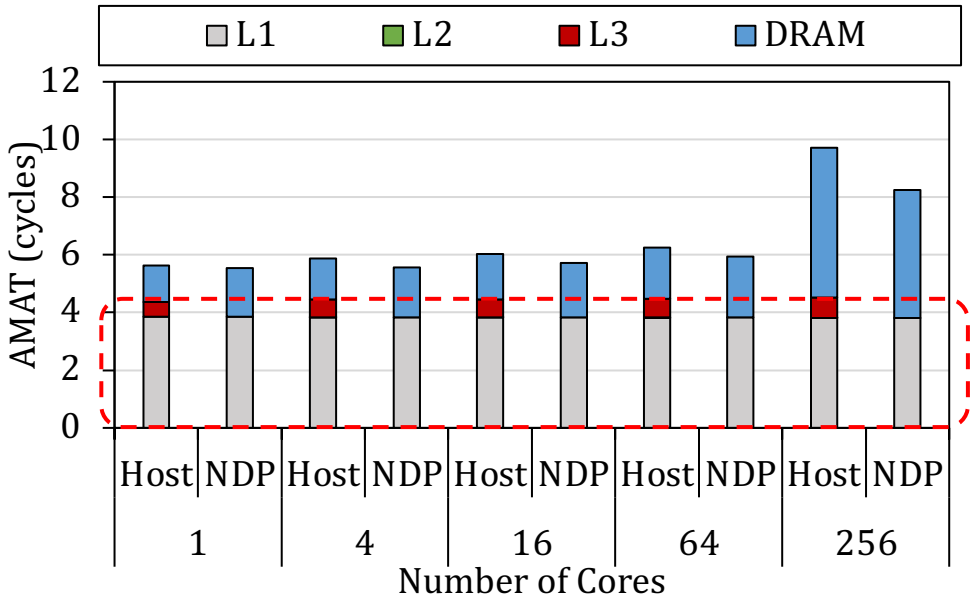


# Class 2b: L1 Cache Capacity

- Low LFMR, MPKI; high temporal locality
  → efficient L2/L3 caches, low memory intensity
- Low AI → few operations per byte
- **Host and NDP performance are similar**
  → **L1 dominates** average memory access time

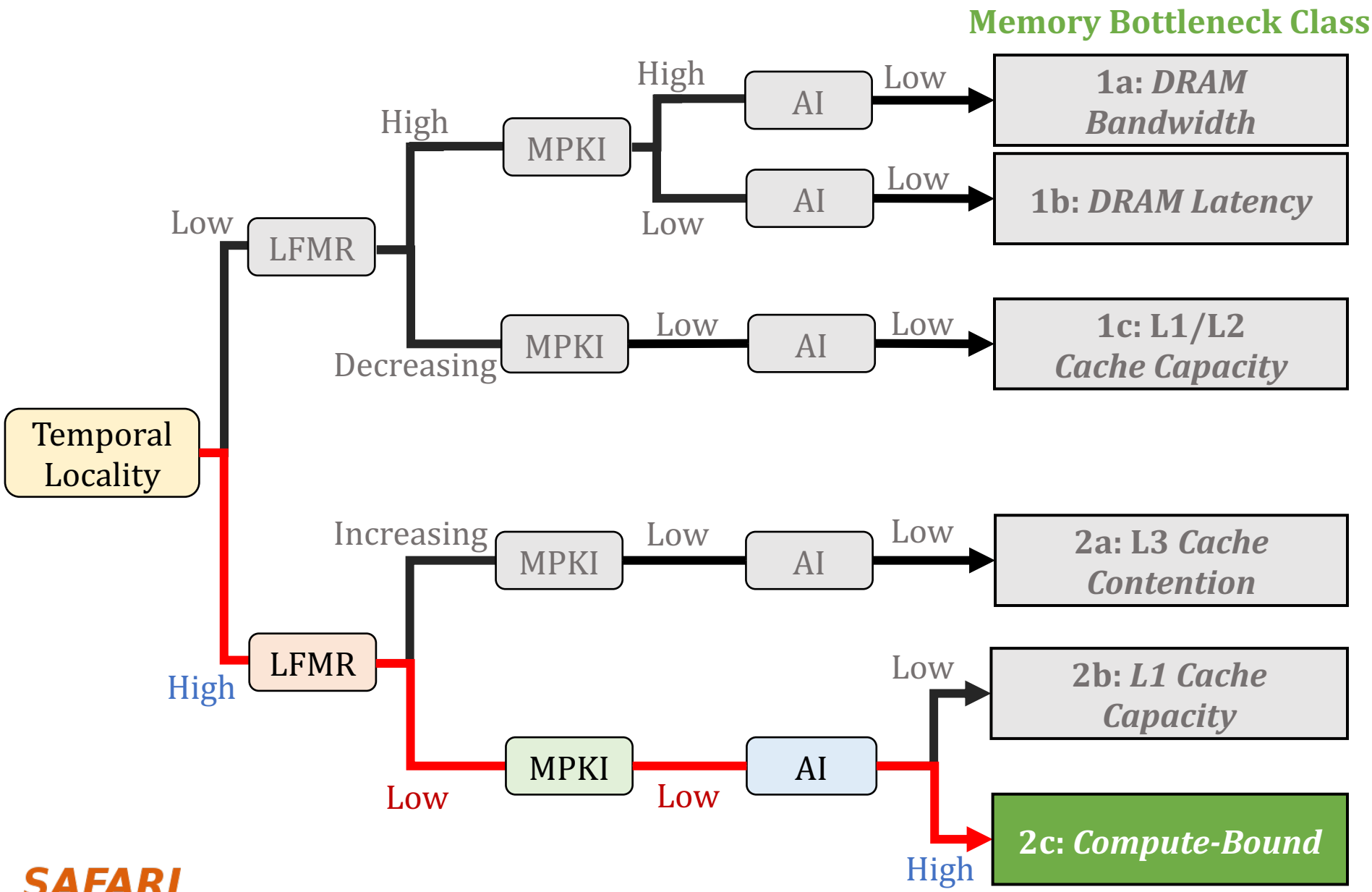
Temp. Loc: *high*
LFMR: *low*
MPKI: *low*
AI: *low*




**L1 cache capacity bottlenecked applications:**
NDP can be used to reduce the host overall SRAM area

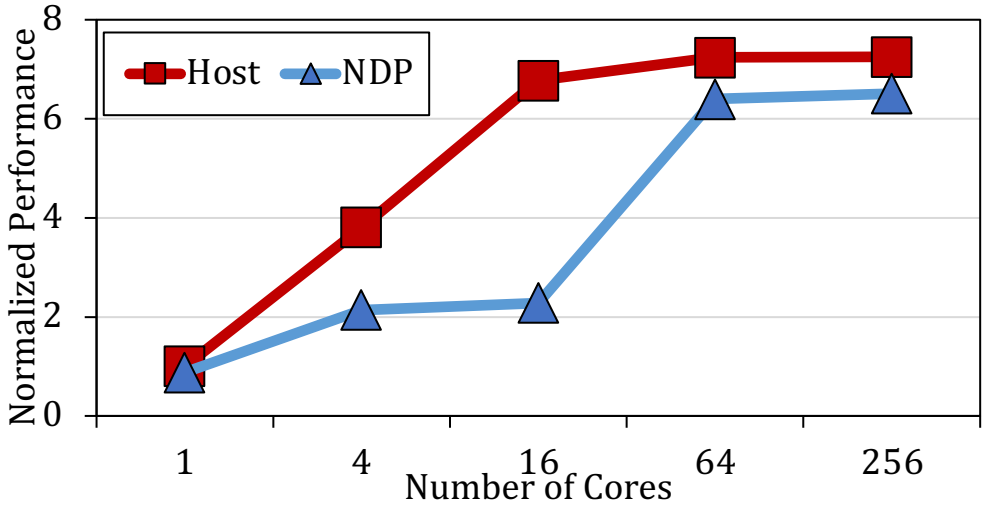
# Step 3: Memory Bottleneck Analysis



# Class 2c: Compute-Bound

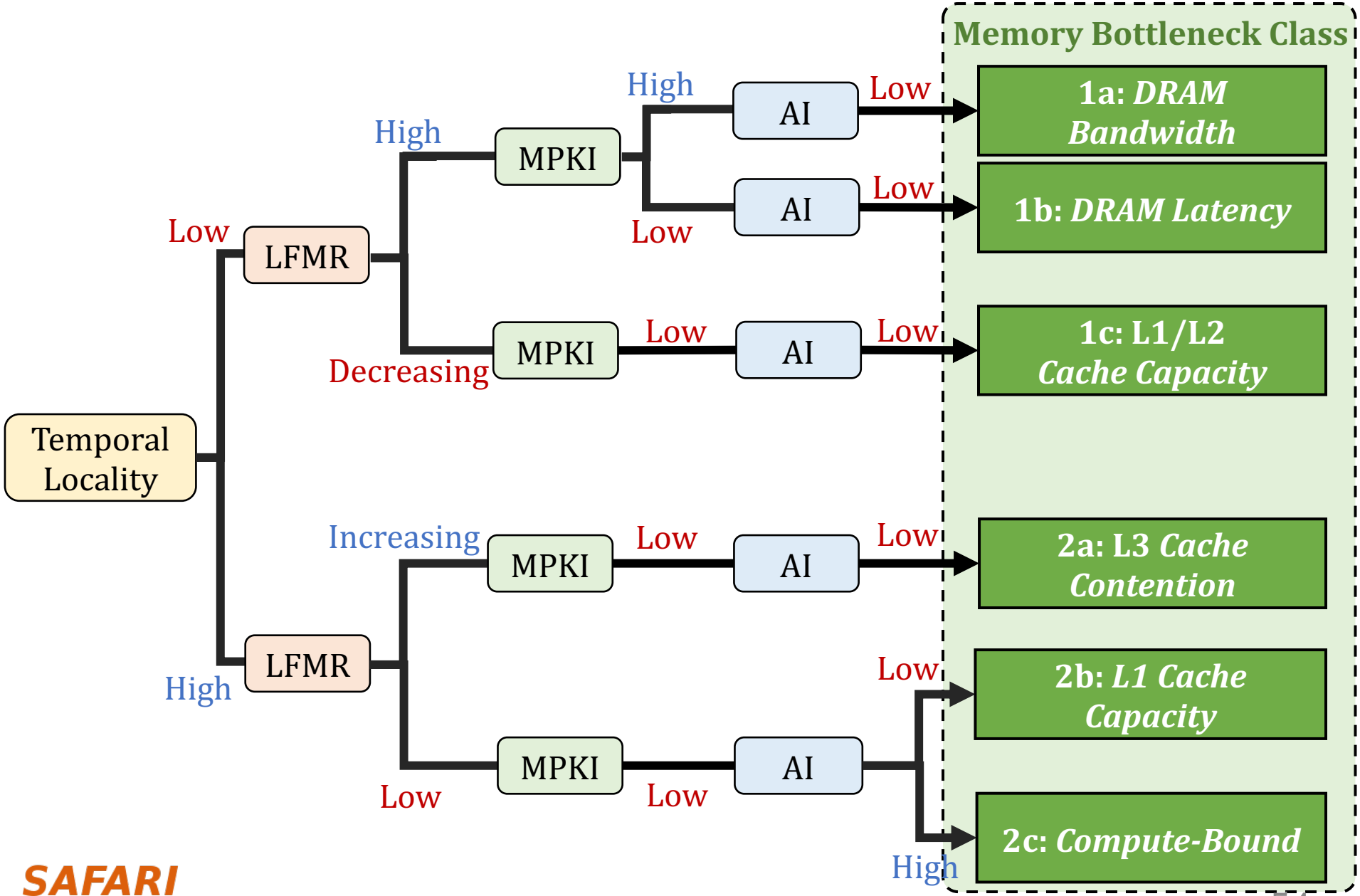
Temp. Loc: *high*
LFMR: *low*
MPKI: *low*
AI: *high*

- Low LFMR, MPKI; high temporal locality
  → efficient L2/L3 caches, low memory intensity
- High AI → many operations per byte
- Host performs better than NDP because computation dominates execution time

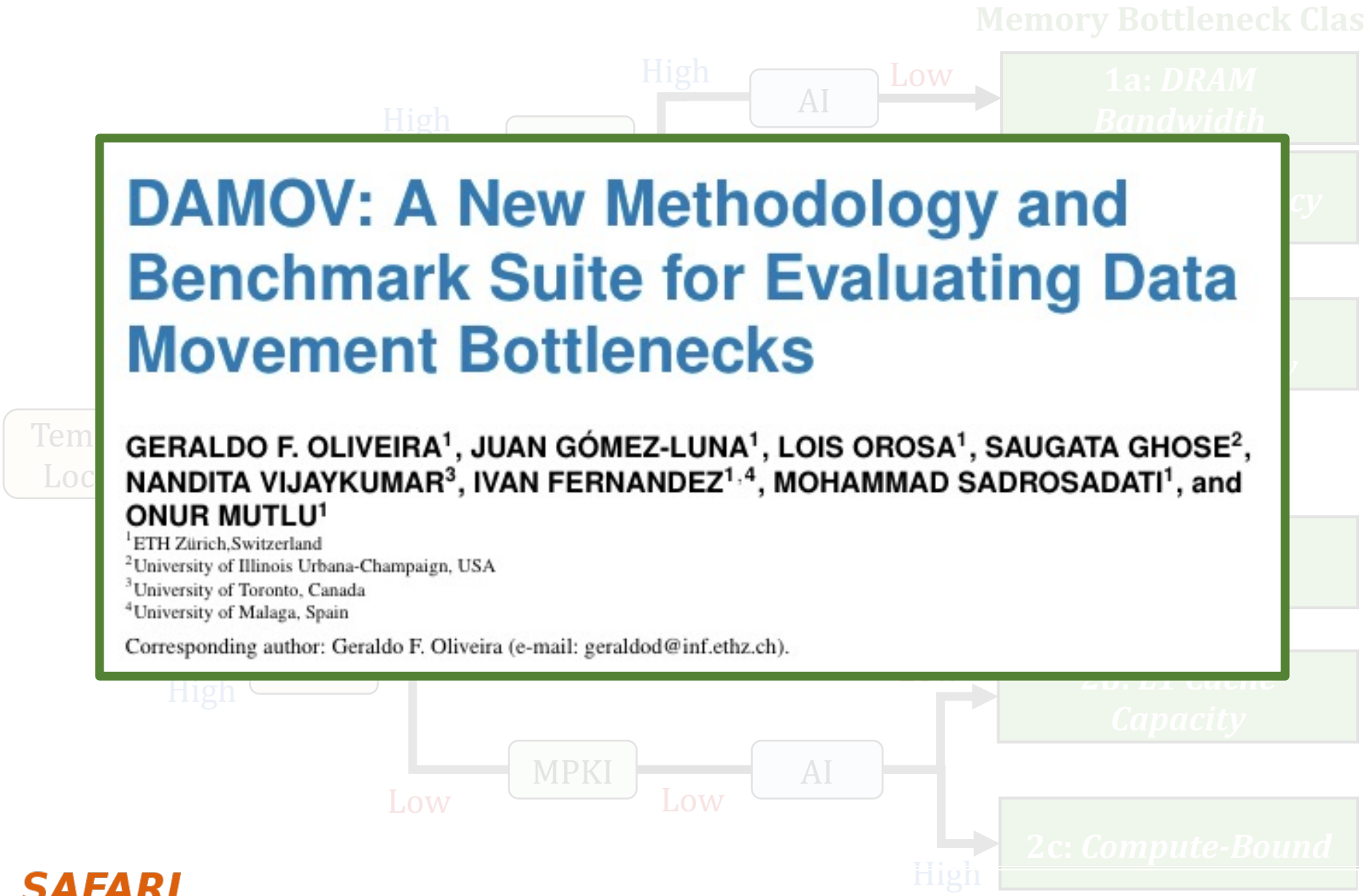



**Compute-bound applications:**
benefit highly from cache hierarchy; NDP is *not* a good fit

# Step 3: Memory Bottleneck Analysis



# Step 3: Memory Bottleneck Analysis




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

GERALDO F. OLIVEIRA[1], JUAN GÓMEZ-LUNA[1], LOIS OROSA[1], SAUGATA GHOSE[2], NANDITA VIJAYKUMAR[3], IVAN FERNANDEZ[1,4], MOHAMMAD SADROSADATI[1], and ONUR MUTLU[1]

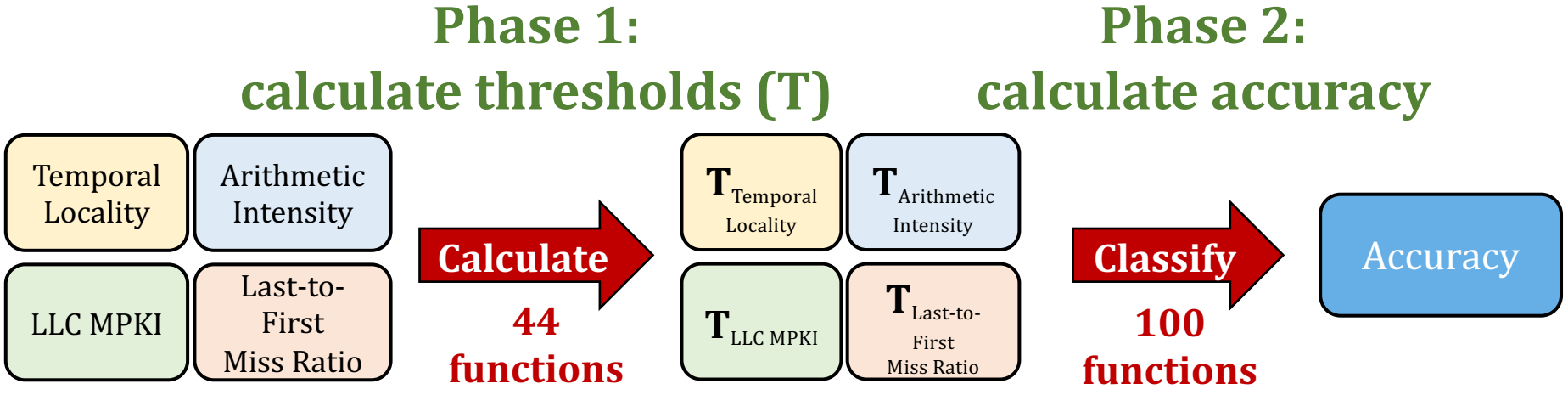
[1]ETH Zürich,Switzerland
[2]University of Illinois Urbana-Champaign, USA
[3]University of Toronto, Canada
[4]University of Malaga, Spain

Corresponding author: Geraldo F. Oliveira (e-mail: geraldod@inf.ethz.ch).

# Methodology Validation

- **Goal:** evaluate the accuracy of our workload characterization methodically on a large set of functions

- Two-phase validation:




**High accuracy:**

our methodology accurately classifies 97% of functions into one of the six memory bottleneck classes

# More in the Paper

- Effect of the last-level cache size
  - Large L3 cache size (e.g., 512 MB) can mitigate some cache contention issues
- Summary of our workload characterization methodology
  - Including workload characterization using in-order host/NDP cores
- Limitations of our methodology
- Benchmark diversity

# More in the Paper

- Effect of the last-level cache size
  - Large L3 cache size (e.g., 512 MB) can mitigate some cache
- Su… …ogy …DP
- Li…
- Benchmark diversity

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

GERALDO F. OLIVEIRA[1], JUAN GÓMEZ-LUNA[1], LOIS OROSA[1], SAUGATA GHOSE[2], NANDITA VIJAYKUMAR[3], IVAN FERNANDEZ[1,4], MOHAMMAD SADROSADATI[1], and ONUR MUTLU[1]

[1]ETH Zürich,Switzerland
[2]University of Illinois Urbana-Champaign, USA
[3]University of Toronto, Canada
[4]University of Malaga, Spain

Corresponding author: Geraldo F. Oliveira (e-mail: geraldod@inf.ethz.ch).

# Outline

1. Data Movement Bottlenecks
2. Methodology Overview
3. Application Profiling
4. Locality-Based Clustering
5. Memory Bottleneck Analysis
6. **Case Studies**

# Case Studies

- Many open questions related to NDP system designs[8]:
  - Interconnects
  - Data mapping and allocation
  - NDP core design (accelerators, general-purpose cores)
  - Offloading granularity
  - Programmability
  - Coherence
  - System integration
  - …

- **Goal:** demonstrate how DAMOV is useful to study NDP system designs

[8] Mutlu+, "A Modern Primer on Processing in Memory," Emerging Computing: From Devices to Systems - Looking Beyond Moore and Von Neumann, 2021

# Case Studies

**Load Balance and Inter-Vault Communication on NDP**

**NDP Accelerators and Our Methodology**

**Different Core Models on NDP Architectures**

**Fine-Grained NDP Offloading**

# Case Studies (1/4)

## Load Balance and Inter-Vault Communication on NDP

portion of the memory requests an NDP core issues go to remote vaults
→ **increases the memory access latency for the NDP core**

## NDP Accelerators and Our Methodology

## Different Core Models on NDP Architectures

## Fine-Grained NDP Offloading

# Case Studies (2/4)

## Load Balance and Inter-Vault Communication on NDP

## NDP Accelerators and Our Methodology

NDP accelerator is faster than compute-centric accelerator for Class 1a and 1b applications; slower for Class 2c

**→ key observations hold for other NDP architectures**

## Different Core Models on NDP Architectures

## Fine-Grained NDP Offloading

# Case Studies (3/4)

## Load Balance and Inter-Vault Communication on NDP

## NDP Accelerators and Our Methodology

## Different Core Models on NDP Architectures

using in-order cores limits performance of some applications
→ **static instruction scheduling cannot exploit memory parallelism**

## Fine-Grained NDP Offloading

# Case Studies (4/4)

**Load Balance and Inter-Vault Communication on NDP**

**NDP Accelerators and Our Methodology**

**Different Core Models on NDP Architectures**

**Fine-Grained NDP Offloading**

few basic blocks are responsible for most of LLC misses
→ **offloading such basic blocks to NDP are enough to improve performance**

# Case Studies

## Load Balance and Inter-Vault Communication on NDP

portion of the memory requests an NDP core issues go to remote vaults
**→ increases the memory access latency for the NDP core**

## NDP Accelerators and Our Methodology

NDP accelerator is faster than compute-centric accelerator for Class 1a and 1b applications; slower for Class 2c
**→ key observations hold for other NDP architectures**

## Different Core Models on NDP Architectures

using in-order cores limits performance of some applications
**→ static instruction scheduling cannot exploit memory parallelism**

## Fine-Grained NDP Offloading

few basic blocks are responsible for most of LLC misses
**→ offloading such basic blocks to NDP are enough to improve performance**

# Case Studies

## Load Balance and Inter-Vault Communication on NDP

## NDP Accelerators and Our Methodology

NDP accelerator is faster than compute-centric accelerator for Class 1a and 1b applications; slower for Class 2c

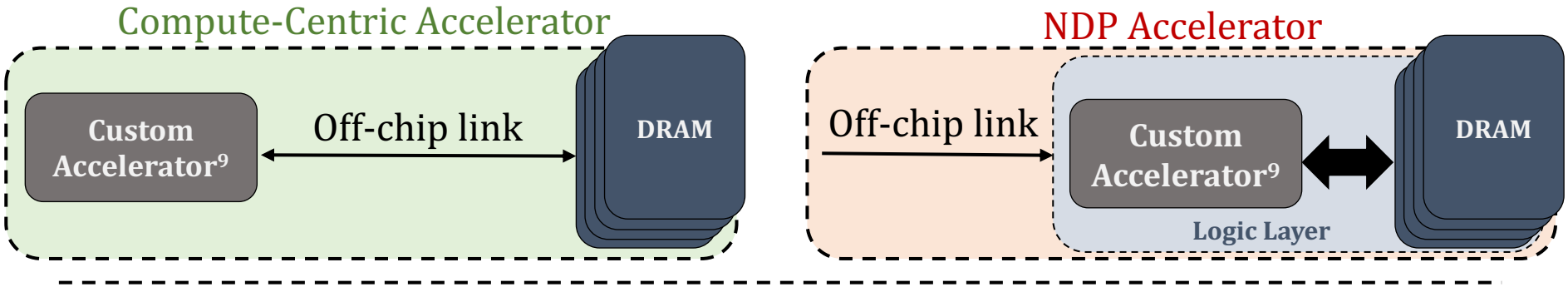
**→ key observations hold for other NDP architectures**

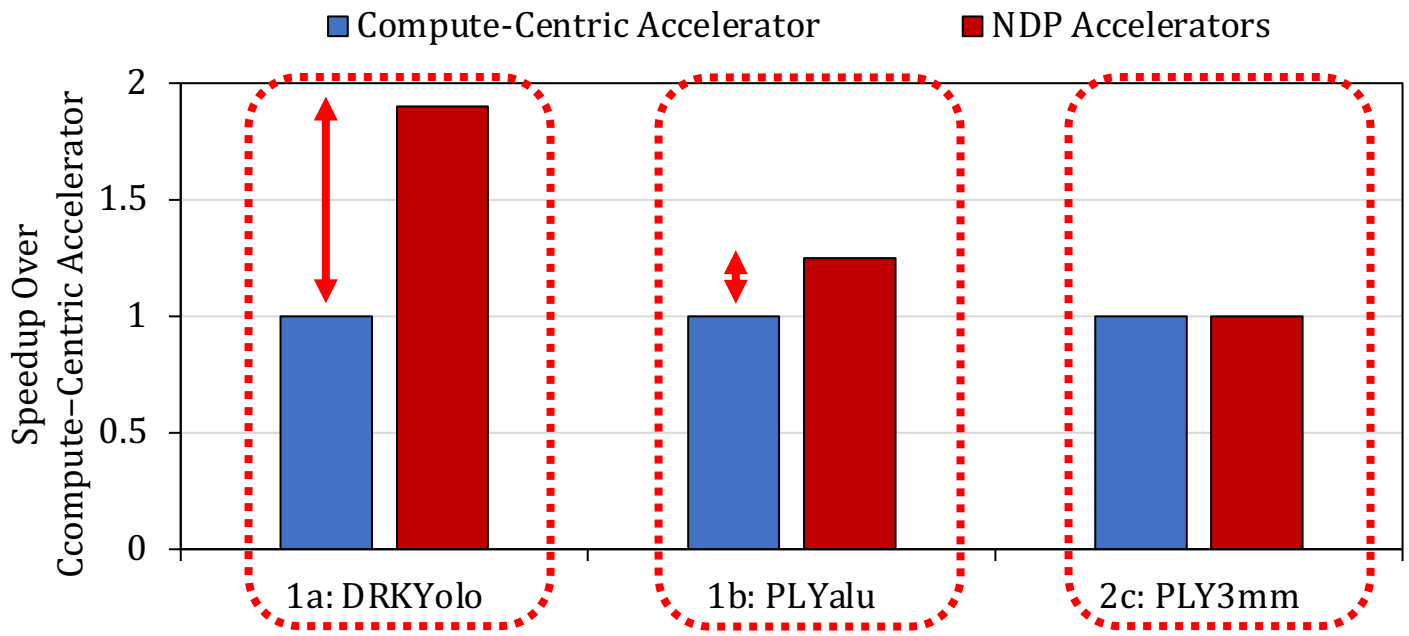
## Different Core Models on NDP Architectures

## Fine-Grained NDP Offloading

# NDP Accelerators and Our Methodology

- Goal: evaluate compute-centric versus NDP accelerators

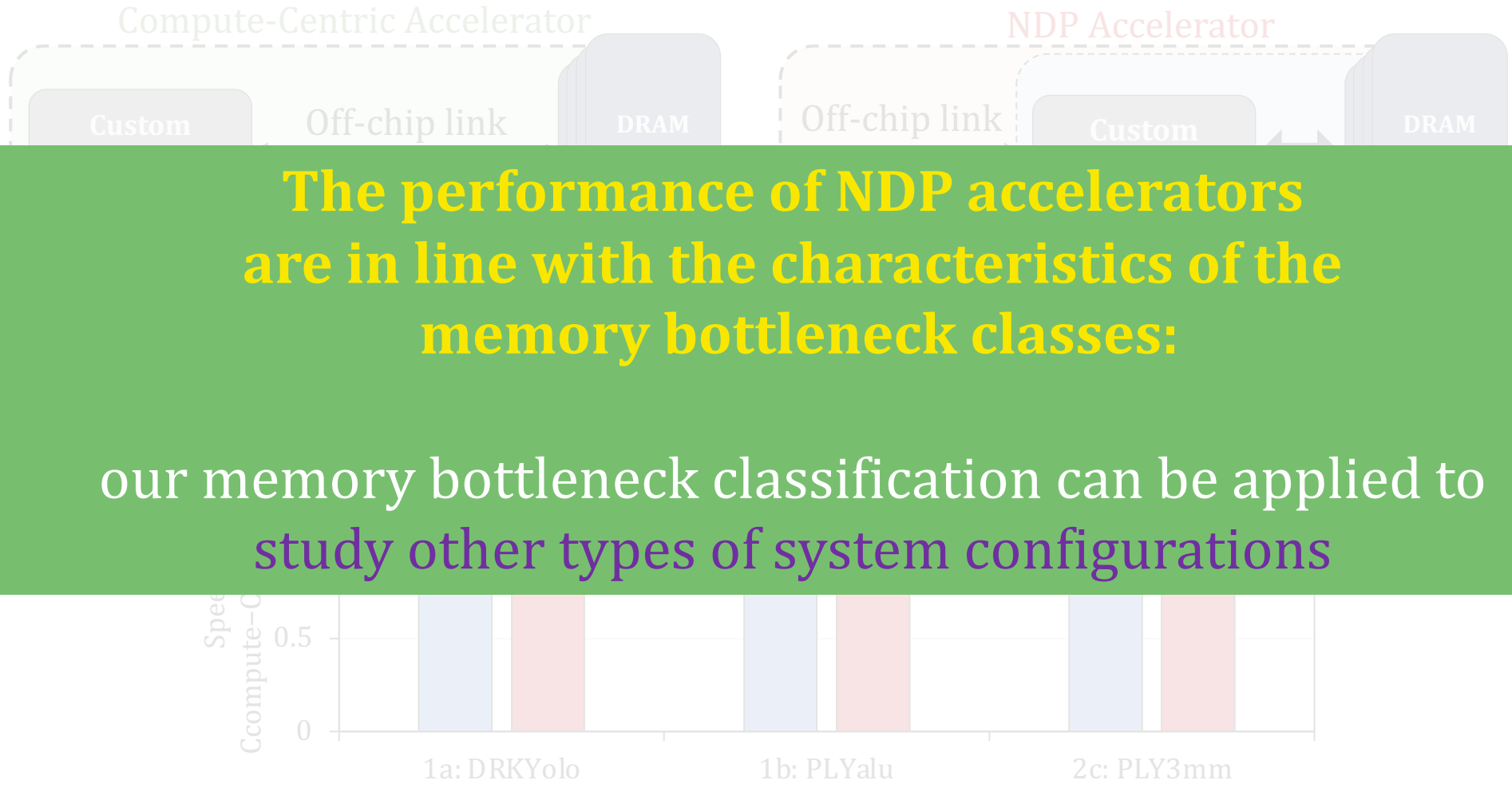



[9] Shao+, "Aladdin: A Pre-RTL, Power-Performance Accelerator Simulator Enabling Large Design Space Exploration of Customized Architectures," in ISCA, 2014

# NDP Accelerators and Our Methodology

- Goal: evaluate compute-centric versus NDP accelerators




**The performance of NDP accelerators are in line with the characteristics of the memory bottleneck classes:**

our memory bottleneck classification can be applied to study other types of system configurations

[9] Shao+, "Aladdin: A Pre-RTL, Power-Performance Accelerator Simulator Enabling Large Design Space Exploration of Customized Architectures," in ISCA, 2014

# Case Studies

## Load Balance and Inter-Vault Communication on NDP

portion of the memory requests an NDP core issues go to remote vaults
**→ increases the memory access latency for the NDP core**

## NDP Accelerators and Our Methodology

NDP accelerator is faster than compute-centric accelerator for Class 1a and 1b applications; slower for Class 2c
**→ key observations hold for other NDP architectures**

## Different Core Models on NDP Architectures

using in-order cores limits performance of some applications
**→ static instruction scheduling cannot exploit memory parallelism**

## Fine-Grained NDP Offloading

few basic blocks are responsible for most of LLC misses
**→ offloading such basic blocks to NDP are enough to improve performance**

# Case Studies

## Load Balance and Inter-Vault Communication on NDP

portion of the memory requests an NDP core issues go to remote vaults

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

GERALDO F. OLIVEIRA[1], JUAN GÓMEZ-LUNA[1], LOIS OROSA[1], SAUGATA GHOSE[2], NANDITA VIJAYKUMAR[3], IVAN FERNANDEZ[1,4], MOHAMMAD SADROSADATI[1], and ONUR MUTLU[1]

[1]ETH Zürich,Switzerland
[2]University of Illinois Urbana-Champaign, USA
[3]University of Toronto, Canada
[4]University of Malaga, Spain

Corresponding author: Geraldo F. Oliveira (e-mail: geraldod@inf.ethz.ch).

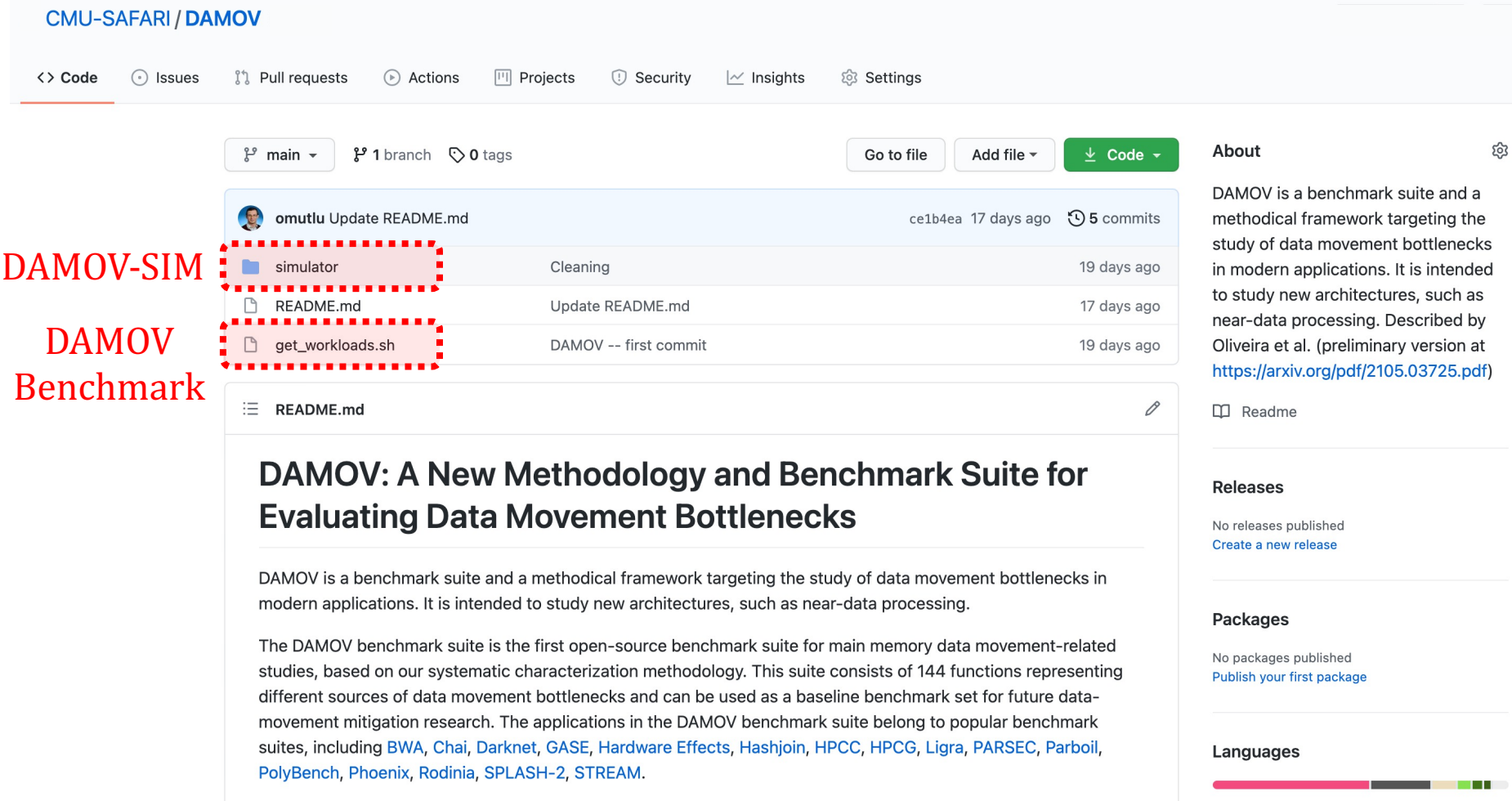
## Fine-Grained NDP Offloading

few basic blocks are responsible for most of LLC misses

→ **offloading such basic blocks to NDP are enough to improve performance**

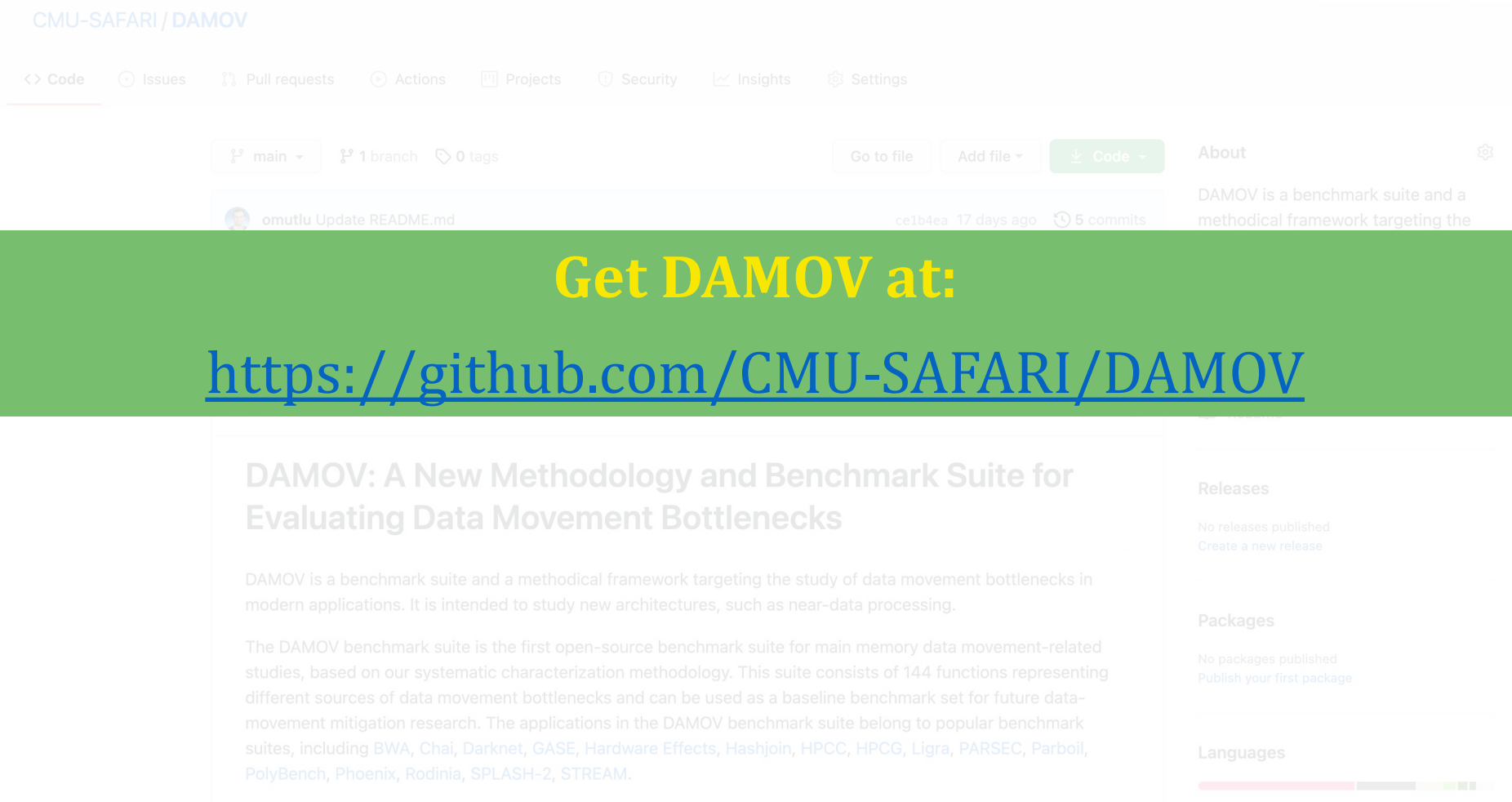
# DAMOV is Open-Source

- We open-source our benchmark suite and our toolchain



# DAMOV is Open-Source

- We open-source our benchmark suite and our toolchain




**Get DAMOV at:**

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

# Conclusion

- **Problem:** Data movement is a major bottleneck is modern systems. However, it is **unclear** how to identify:
  - **different sources** of data movement bottlenecks
  - the **most suitable** mitigation technique (e.g., caching, prefetching, near-data processing) for a given data movement bottleneck
- **Goals:**
  1. Design a methodology to **identify** sources of data movement bottlenecks
  2. **Compare** compute- and memory-centric data movement mitigation techniques
- **Key Approach:** Perform a large-scale application characterization to identify **key metrics** that reveal the sources to data movement bottlenecks
- **Key Contributions**:
  - **Experimental characterization** of 77K functions across 345 applications
  - A **methodology** to characterize applications based on data movement bottlenecks and their relation with different data movement mitigation techniques
  - **DAMOV:** a **benchmark suite** with **144 functions** for data movement studies
  - **Four case-studies** to highlight DAMOV's applicability to open research problems



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

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

**Geraldo F. Oliveira**

Juan Gómez-Luna  Lois Orosa  Saugata Ghose

Nandita Vijaykumar  Ivan Fernandez  Mohammad Sadrosadati

Onur Mutlu

SAFARI

ETH Zürich  UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN  UNIVERSITY OF TORONTO  UNIVERSITAS MALACITANA  UNIVERSIDAD DE MÁLAGA

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

## DAMOV Benchmark Suite and DAMOV-SIM Tutorial

SAFARI

ETH Zürich

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

UNIVERSITY OF TORONTO

UNIVERSITAS MALACITANA

UNIVERSIDAD DE MÁLAGA

# Outline

- Downloading and Installing
- Compiling the Workloads
- Installing DAMOV-SIM
- Generating Configuration Files
- Executing Simulations and Collecting Statistics

# Outline

- **Downloading and Installing**
- Compiling the Workloads
- Installing DAMOV-SIM
- Generating Configuration Files
- Executing Simulations and Collecting Statistics

# Downloading and Installing

- Step 1: Clone DAMOV's GitHub repository

```
$ git clone https://github.com/CMU-SAFARI/DAMOV.git
$ cd DAMOV/
$ ls
get_workloads.sh  LICENSE  README.md  simulator
```

- Step 2: Download the workloads

```
$ ./get_workloads.sh
Downloading DAMOV workloads -- Part 1
Cloning into 'damovworkloadspart1'...
…
bwa  chai-cpu  Darknet  GASE-master  hardware-effects  hpcc  hpcg
ligra  multicore-hashjoins-0.1  parboil  parsec-3.0  phoenix
PolyBench-ACC  rodinia_3.1  STREAM  zsim_hooks.h
```

# Outline

- Downloading and Installing
- **Compiling the Workloads**
- Installing DAMOV-SIM
- Generating Configuration Files
- Executing Simulations and Collecting Statistics

# Compiling the Workloads

- *Note 1:* Each workload might have its own set of dependencies

- Step 3: Compile applications with `compile.py`

```
$ cd workloads/STREAM/
$ python compile.py
$ ls
stream_add  stream.c  stream_copy  stream_scale  stream_triad
compile.py
```

- *Note 2:* `compile.py` generates a single binary per memory-bound function

# Outline

- Downloading and Installing
- Compiling the Workloads
- **Installing DAMOV-SIM**
- Generating Configuration Files
- Executing Simulations and Collecting Statistics

# Installing DAMOV-SIM

- Step 4: Install DAMOV-SIM

```
$ cd simulator/
$ sudo ./scripts/setup.sh
$ ./scripts/compile.sh
…
scons: done building targets.
```

- To execute a simulation:

```
$ ./build/opt/zsim configuration_file
```

# Outline

- Downloading and Installing
- Compiling the Workloads
- Installing DAMOV-SIM
- **Generating Configuration Files**
- Executing Simulations and Collecting Statistics

# Generating Configuration Files (1/4)

- DAMOV-SIM simulates different system configurations
  - *Host:* a host CPU with private L1/L2 and shared L3 caches → **fixed LLC size**
  - *Host with prefetcher:* same as *Host* with a stream prefetcher
  - *Host NUCA:* a host CPU with private L1/L2 and shared L3 caches organized in a 2D mesh network → **LLC size increases as a factor of the core count**
  - *NDP:* an NDP system with private L1 cache
- We provide template configuration files under `simulator/templates`

| | in-order | out-out-order | accelerator |
|---|---|---|---|
| Host | template_host_inorder.cfg | template_host_ooo.cfg | template_host_accelerator.cfg |
| Host with prefetcher | template_host_prefetch_inorder.cfg | template_host_prefetch_ooo.cfg | template_host_prefetch_accelerator.cfg |
| Host with NUCA | template_host_nuca_inorder.cfg | template_host_nuca.cfg | N/A |
| NDP | template_pim_inorder.cfg | template_pim_ooo.cfg | template_pim_accelerator.cfg |

# Generating Configuration Files (2/4)

- Step 5.1: Create a command file
  - Command file format = {benchmark, application, function, binary path, inputs}

```
$ cd simulator/
$ cat command_files/stream_cf
stream,Add,Add,PIM_ROOT/STREAM/stream_add THREADS
stream,Copy,Copy,PIM_ROOT/STREAM/stream_copy THREADS
stream,Scale,Scale,PIM_ROOT/STREAM/stream_scale THREADS
stream,Triad,Triad,PIM_ROOT/STREAM/stream_triad THREADS
```

# Generating Configuration Files (3/4)

- Step 5.2: Generate configuration files using `generate_config_files.py`

```
$ python scripts/generate_config_files.py command_files/stream_cfg
$ ls config_files/
host_accelerator  host_inorder  host_ooo  pim_accelerator
pim_inorder  pim_ooo

$ ls config_files/host_ooo/
no_prefetch  prefetch

$ ls config_files/host_ooo/no_prefetch/
stream

$ ls config_files/host_ooo/no_prefetch/stream
1  16  256  4  64

$ ls config_files/host_ooo/no_prefetch/stream/4/
Add_Add.cfg  Copy_Copy.cfg  Scale_Scale.cfg  Triad_Triad.cfg
```

# Generating Configuration Files (4/4)

- Step 5.3: Verify the configuration file

```
$ cat config_files/host_ooo/no_prefetch/stream/4/Add_Add.cfg

// This system is similar to a 6-core, 2.4GHz Westmere with 10
Niagara-like cores attached to the L3
sys = {
    lineSize = 64;
    frequency = 2400;

    cores = {
        core = {
            type = "OOO";
            cores = 4;
            icache = "l1i";
            dcache = "l1d";
        };
    };
…
```

type = {OOO, Timing, Accelerator}

# Generating Configuration Files (4/4)

- Step 5.3: Verify the configuration file

```
$ cat config_files/host_ooo/no_prefetch/stream/4/Add_Add.cfg
…
    caches = {
        l1d = {
            caches = 4;
            size = 32768;
            array = {
                type = "SetAssoc";
                ways = 8;
            };
            latency = 4;
        };

        l1i = { … };
        l2  = { … };
        l3  = { … };
        mem = {
            type = "Ramulator";
            ramulatorConfig = "ramulator-configs/HMC-config.cfg";
            latency = 1;
        };};
…
```

Define cache size and organization

Use Ramulator as the memory controller, and HMC as the main memory

# Generating Configuration Files (4/4)

- Step 5.3: Verify the configuration file

```
$ cat config_files/host_ooo/no_prefetch/stream/4/Add_Add.cfg
…
sim = {
    pimMode = false;
    stats = "zsim_stats/host_ooo/no_prefetch/4/stream_Add_Add";
    phaseLength = 1000;
    maxOffloadInstrs = 1000000000L;
    maxTotalInstrs = 1000000000L;
    …
};

process0 = {
    command = "/home/safari/DAMOV/workloads//STREAM/stream_add 4";
    startFastForwarded = True;
};
```

pimMode enables host (*pimMode = false*) or NDP (*pimMode = true*) execution

Simulation termination condition

Command that will be simulated

# Outline

- Downloading and Installing
- Compiling the Workloads
- Installing DAMOV-SIM
- Generating Configuration Files
- **Executing Simulations and Collecting Statistics**

# Executing a Simulation

- Step 6: Run STREAM Add in a host CPU with 4 cores

```
$ ./build/opt/zsim config_files/host_ooo/no_prefetch/stream/4/Add_Add.cfg
```

- Step 7: Run STREAM Add in an NDP system with 4 cores

```
$ ./build/opt/zsim config_files/pim_ooo/stream/4/Add_Add.cfg
```

- Current DAMOV-SIM limitation: no support for concurrent execution on host and NDP cores

# Collecting Statistics (1/2)

- Step 8.1: Check the statistics stored under `zsim_stats/`

```
$ ls zsim_stats/host_ooo/no_prefetch/4/
stream_Add_Add.dramRequestsPerPhase  stream_Add_Add.out.cfg
stream_Add_Add.ramulator.stats  stream_Add_Add.zsim.out

$ ls zsim_stats/pim_ooo/4/
stream_Add_Add.dramRequestsPerPhase  stream_Add_Add.out.cfg
stream_Add_Add.ramulator.stats  stream_Add_Add.zsim.out
```

# Collecting Statistics (2/2)

- Step 8.2: Filter key metrics to compare host and NDP execution using `get_stats_per_app.py`

```
$ python scripts/get_stats_per_app.py
zsim_stats/host_ooo/no_prefetch/4/stream_Add_Add.zsim.out
 ------------------ Summary ----------------------
Instructions: 1000000939
Cycles: 450320568
IPC: 2.22064238247
L3 Miss Rate (%): 99.9992276064
L2 Miss Rate (%): 100.0
L1 Miss Rate (%): 73.5630799769
L3 MPKI: 23.433444996
LFMR: 0.999993129532

$ python scripts/get_stats_per_app.py
zsim_stats/pim_ooo/4/stream_Add_Add.zsim.out
 ------------------ Summary ----------------------
Instructions: 1000005774
Cycles: 280045288
IPC: 3.57087162988
…
```

NDP speedup over CPU
= 3.57/2.22 = 1.6x

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

## DAMOV Benchmark Suite and DAMOV-SIM Tutorial

SAFARI

ETH Zürich

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

UNIVERSITY OF TORONTO

UNIVERSITAS MALACITANA

UNIVERSIDAD DE MÁLAGA