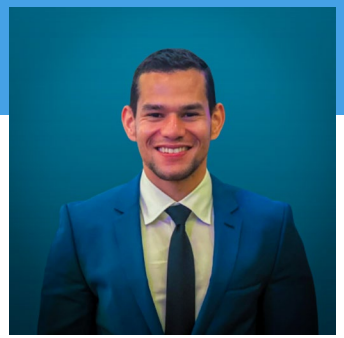
Methodologies, Workloads, and Tools for Processing-in-Memory: Enabling the Adoption of Data-Centric Architectures

Geraldo F. Oliveira

(https://geraldofojunior.github.io/)

Onur Mutlu

Brief Self Introduction



- **Geraldo F. Oliveira**
  - Researcher @ SAFARI Research Group since November 2017
  - Soon, I will defend my PhD thesis, advised by Onur Mutlu
  - https://geraldofojunior.github.io/
  - geraldofojunior@gmail.com (best way to reach me)
  - https://safari.ethz.ch

- **Research in:**
  - Computer architecture, computer systems, hardware security
  - Memory and storage systems
  - Hardware security, safety, predictability
  - Fault tolerance
  - Hardware/software cooperation
  - …

Outline

1. Introduction
2. Identifying Memory Bottlenecks
   - Methodology Overview
   - Application Profiling
   - Locality-Based Clustering
   - Memory Bottleneck Analysis
   - DAMOV Benchmark Suite
3. Enabling Complex Operations using DRAM
   - SIMDRAM Framework
   - System Integration
   - Evaluation

Outline

1. Introduction

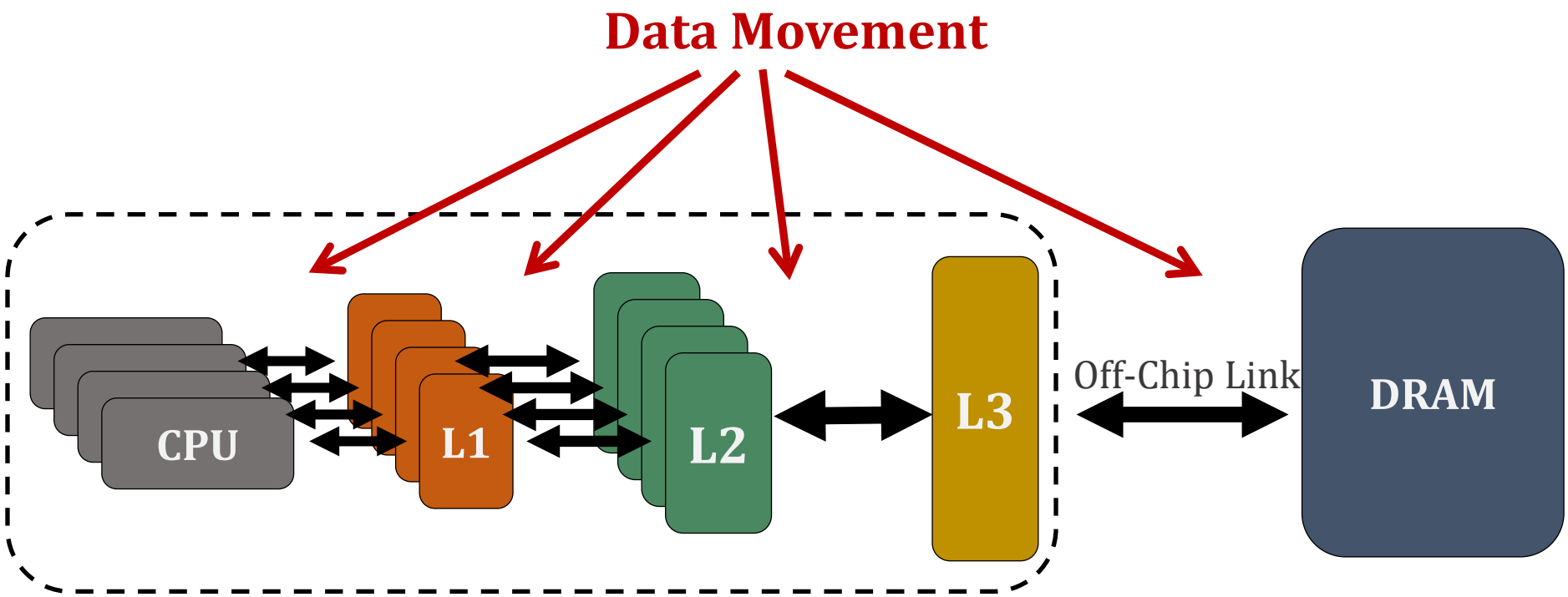
2. Identifying Memory Bottlenecks

- Methodology Overview
- Application Profiling
- Locality-Based Clustering
- Memory Bottleneck Analysis
- DAMOV Benchmark Suite

3. Enabling Complex Operations using DRAM

- SIMDRAM Framework
- System Integration
- Evaluation

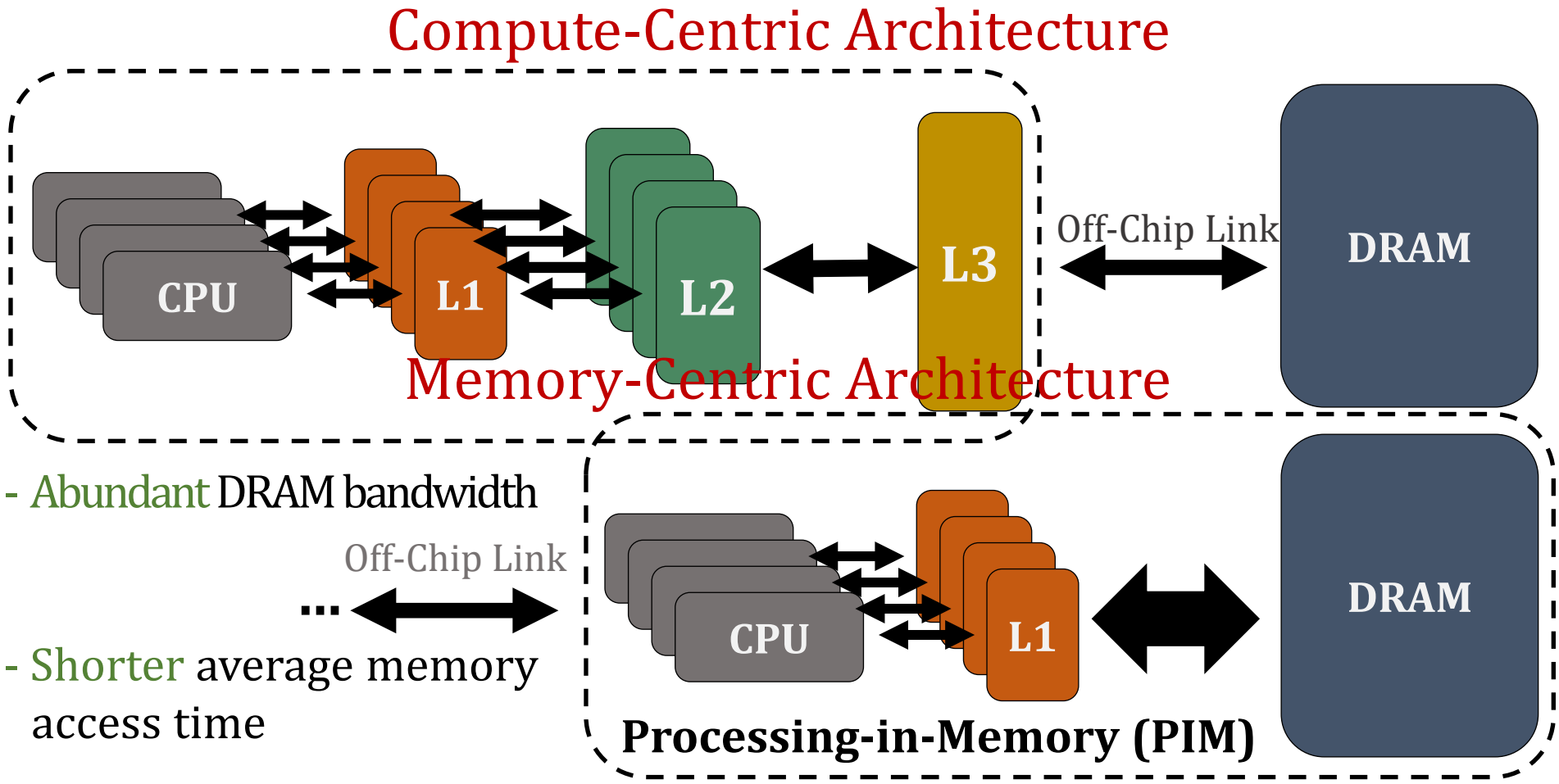
Data Movement Bottlenecks (1/2)



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)

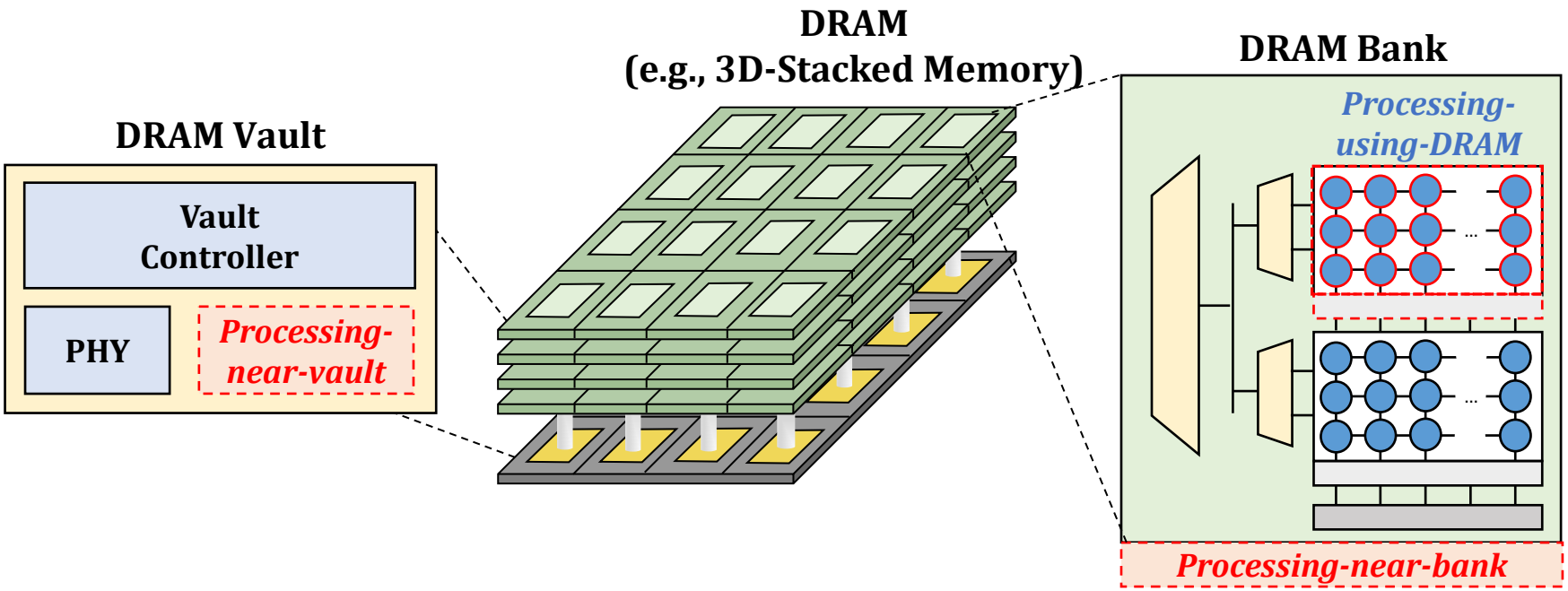


Processing-in-Memory: Taxonomy

Two main approaches for Processing-in-Memory:

1 **Processing-*near*-Memory**: PIM logic is added to the same die as memory or to the logic layer of 3D-stacked memory

2 **Processing-*using*-Memory**: uses the operational principles of memory cells to perform computation



Processing-in-Memory: Challenges

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

To fully support PIM systems, we need to develop:

1. **Workload characterization methodologies and benchmark suites targeting PIM architectures**
2. **Frameworks that can facilitate the implementation of complex operations and algorithms using PIM primitives**
3. **Compiler support and compiler optimizations targeting PIM architectures**
4. **Operating system support for PIM-aware virtual memory, memory management, data allocation and mapping**
5. **Efficient data coherence and consistency mechanisms**

In this Work

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

To fully supportt PIM systems, we need to develop:

1. **Workload characterization methodologies and benchmark suites targeting PIM architectures**
2. **Frameworks that can facilitate the implementation of complex operations and algorithms using PIM primitives**
3. **Compiler support and compiler optimizations targeting PIM architectures**
4. **Operating system support for PIM-aware virtual memory, memory management, data allocation and mapping**
5. **Efficient data coherence and consistency mechanisms**

Outline

1. Introduction

2. Identifying Memory Bottlenecks

- Methodology Overview
- Application Profiling
- Locality-Based Clustering
- Memory Bottleneck Analysis
- DAMOV Benchmark Suite

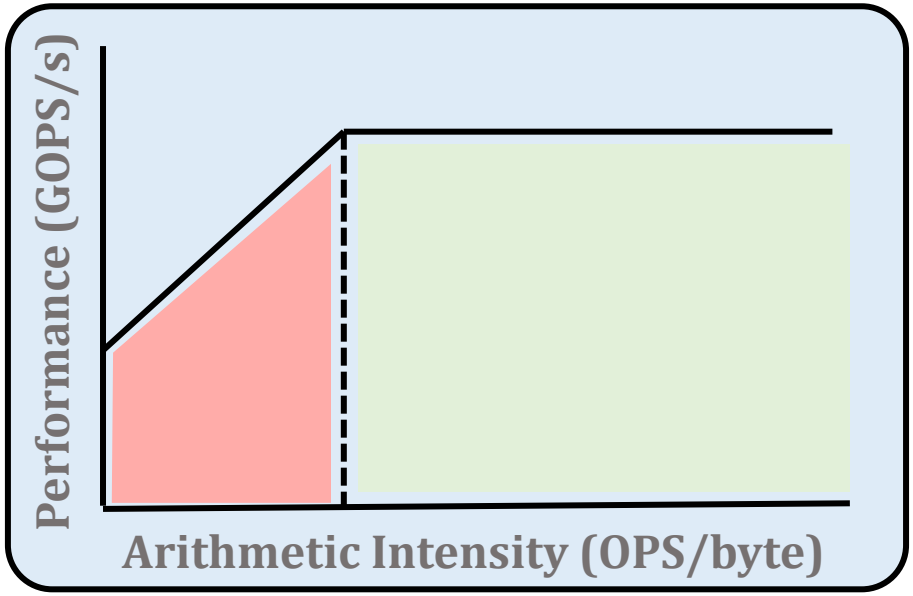
3. Enabling Complex Operations using DRAM

- SIMDRAM Framework
- System Integration
- Evaluation

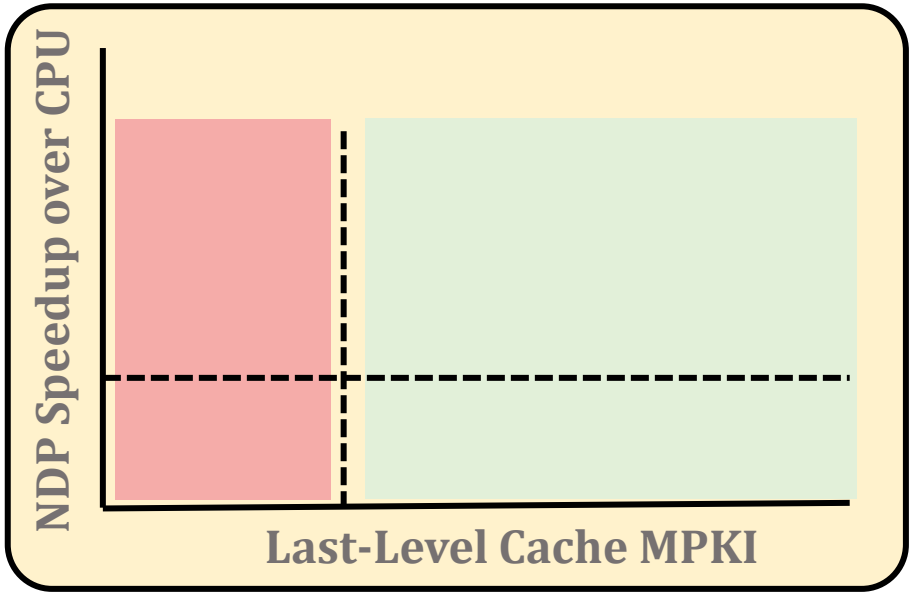
Identifying Memory Bottlenecks

- 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. Introduction

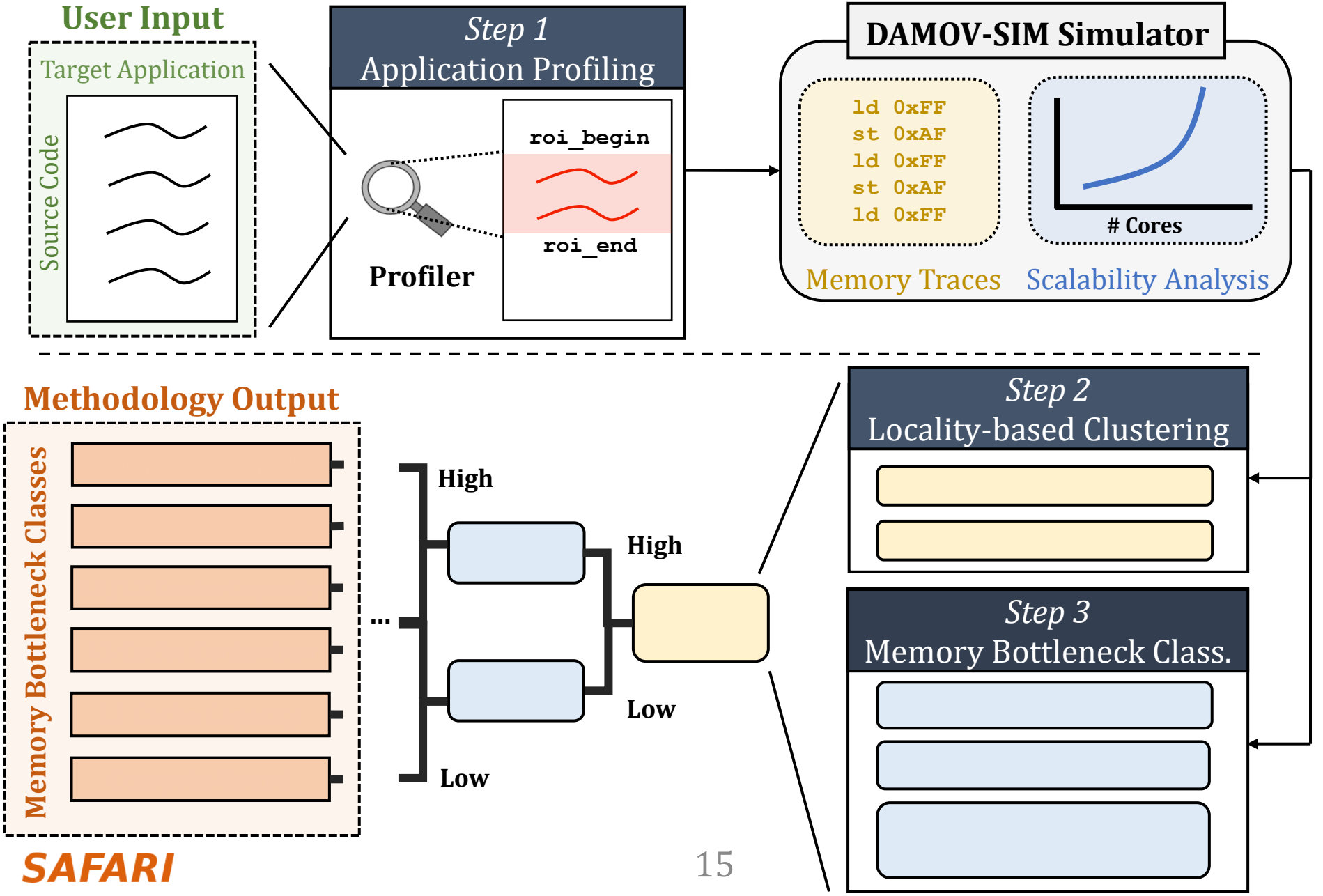
2. Identifying Memory Bottlenecks

- **Methodology Overview**
- Application Profiling
- Locality-Based Clustering
- Memory Bottleneck Analysis
- DAMOV Benchmark Suite

3. Enabling Complex Operations using DRAM

- SIMDRAM Framework
- System Integration
- Evaluation

Methodology Overview



Outline

1. Introduction

2. Identifying Memory Bottlenecks

- Methodology Overview
- **Application Profiling**
- Locality-Based Clustering
- Memory Bottleneck Analysis
- DAMOV Benchmark Suite

3. Enabling Complex Operations using DRAM

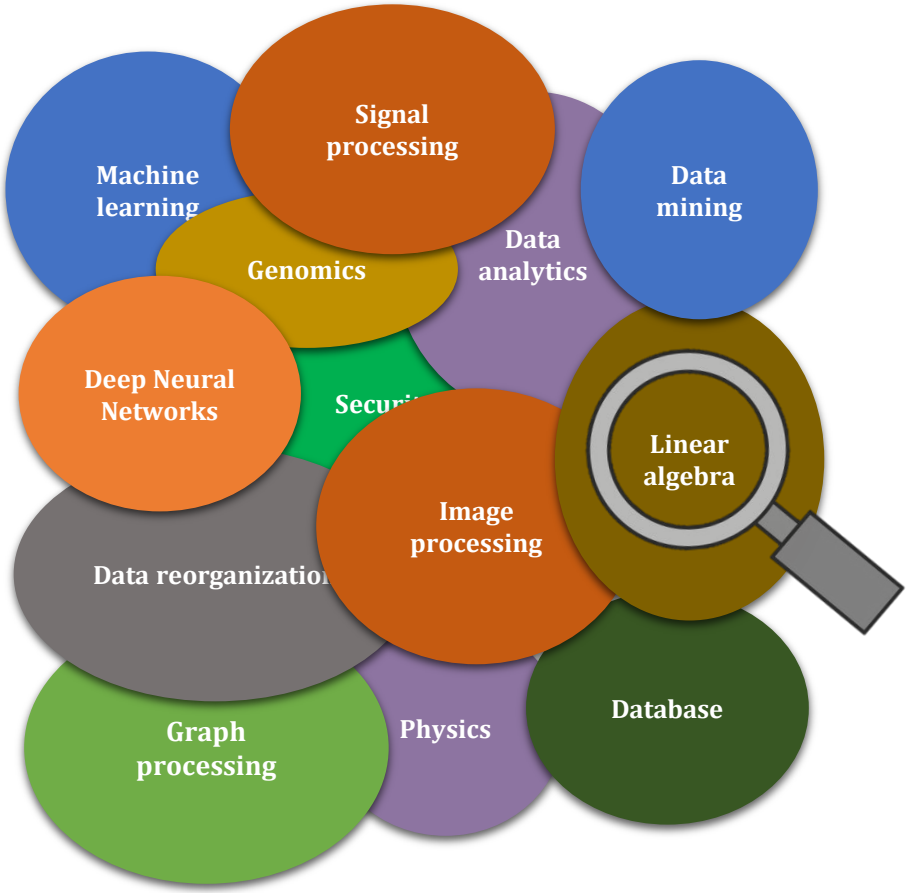
- SIMDRAM Framework
- System Integration
- Evaluation

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

...



Outline

1. Introduction

2. Identifying Memory Bottlenecks

- Methodology Overview
- Application Profiling
- **Locality-Based Clustering**
- Memory Bottleneck Analysis
- DAMOV Benchmark Suite

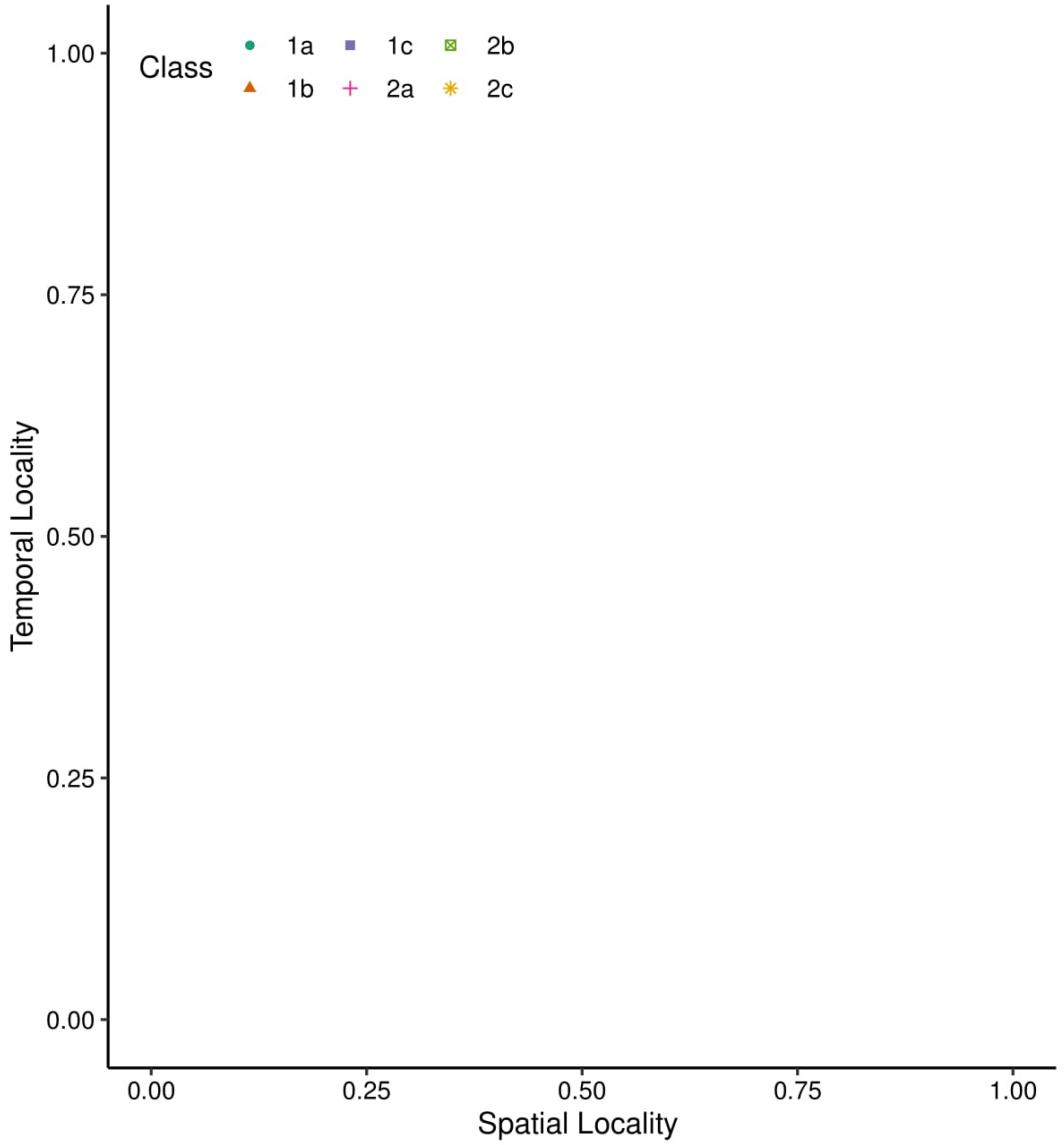
3. Enabling Complex Operations using DRAM

- SIMDRAM Framework
- System Integration
- Evaluation

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)

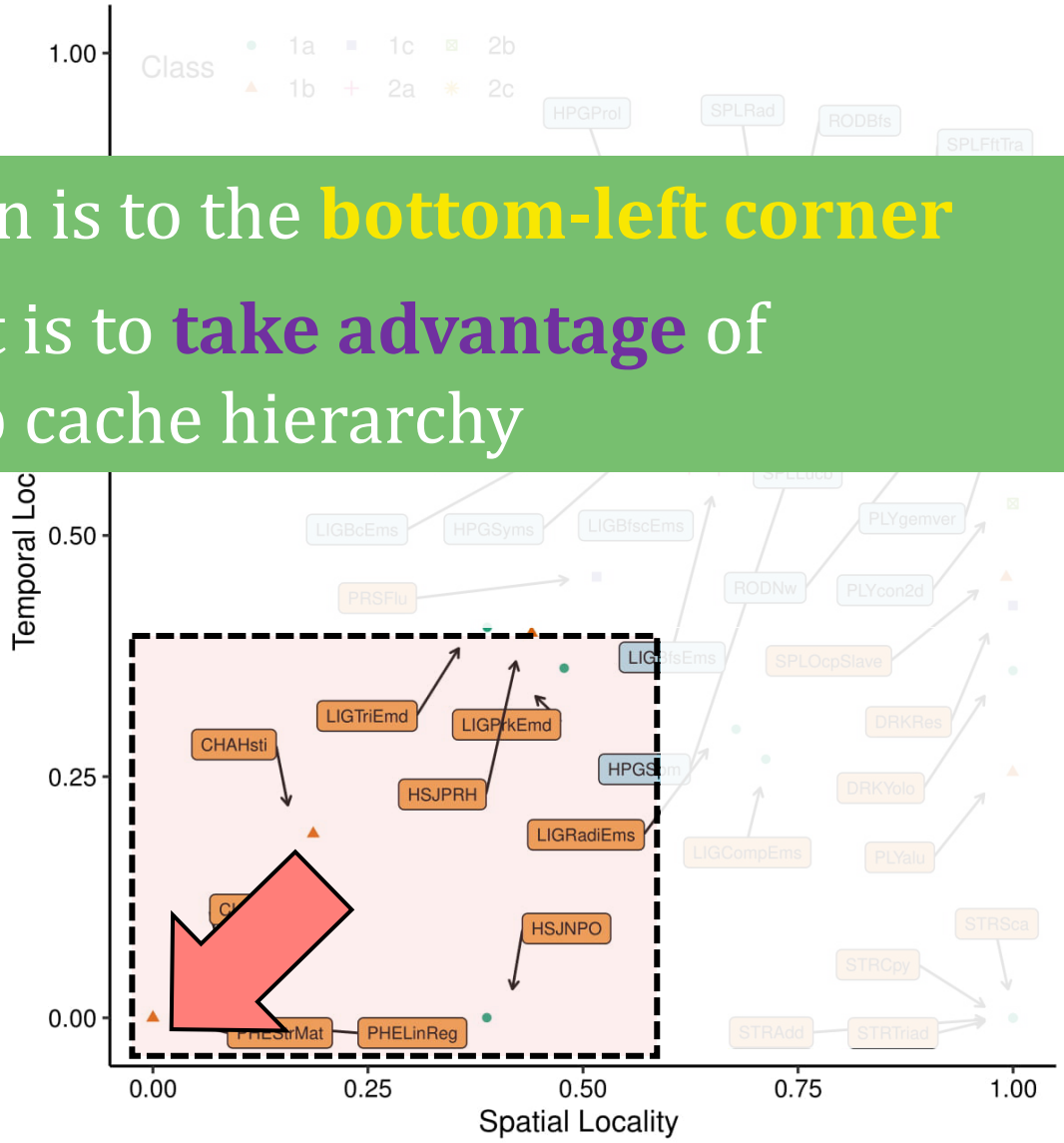


Step 2: Locality-Based Clustering

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



2. High locality applications (in blue)

Outline

1. Introduction

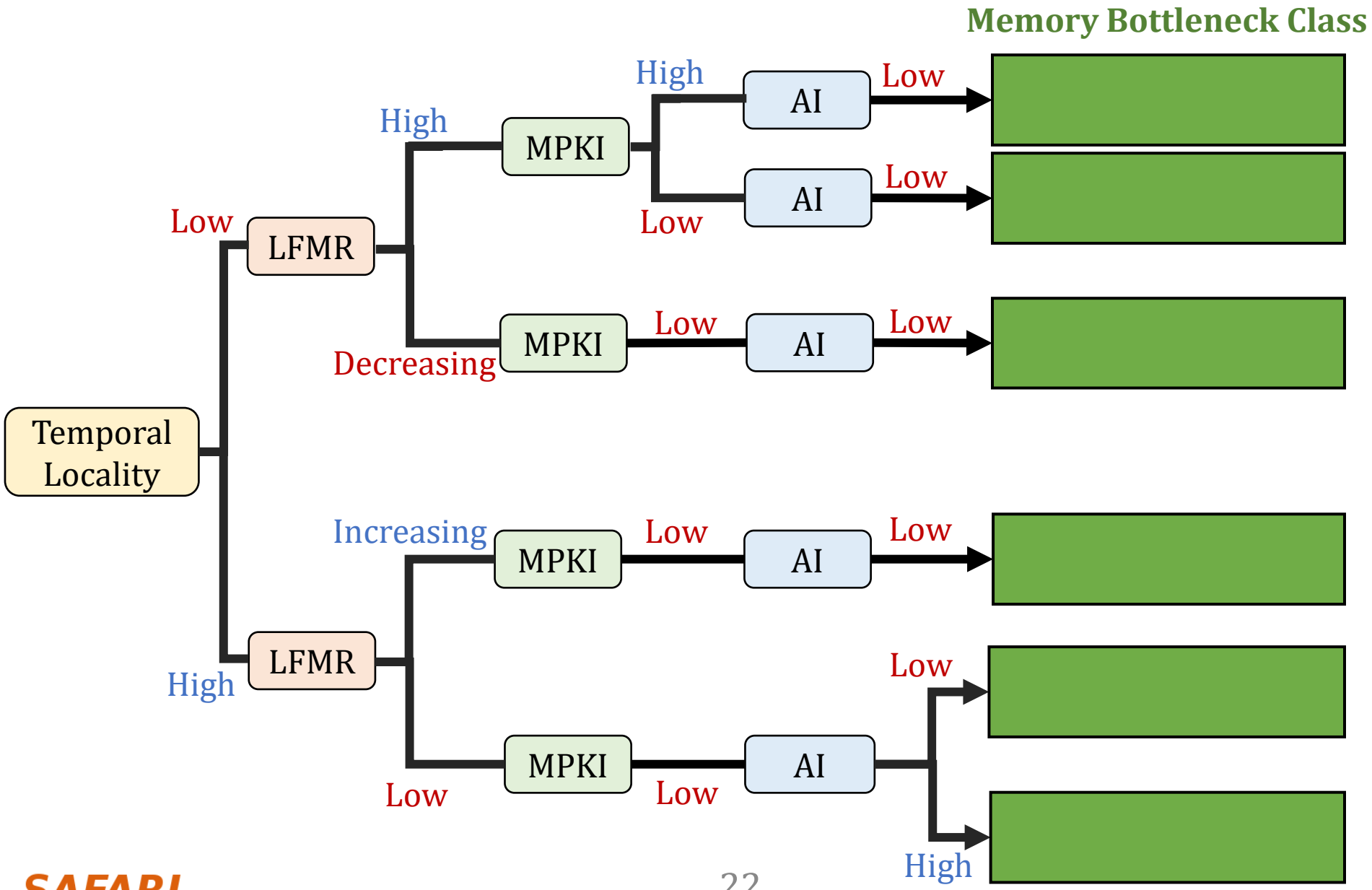
2. Identifying Memory Bottlenecks

- Methodology Overview
- Application Profiling
- Locality-Based Clustering
- **Memory Bottleneck Analysis**
- DAMOV Benchmark Suite

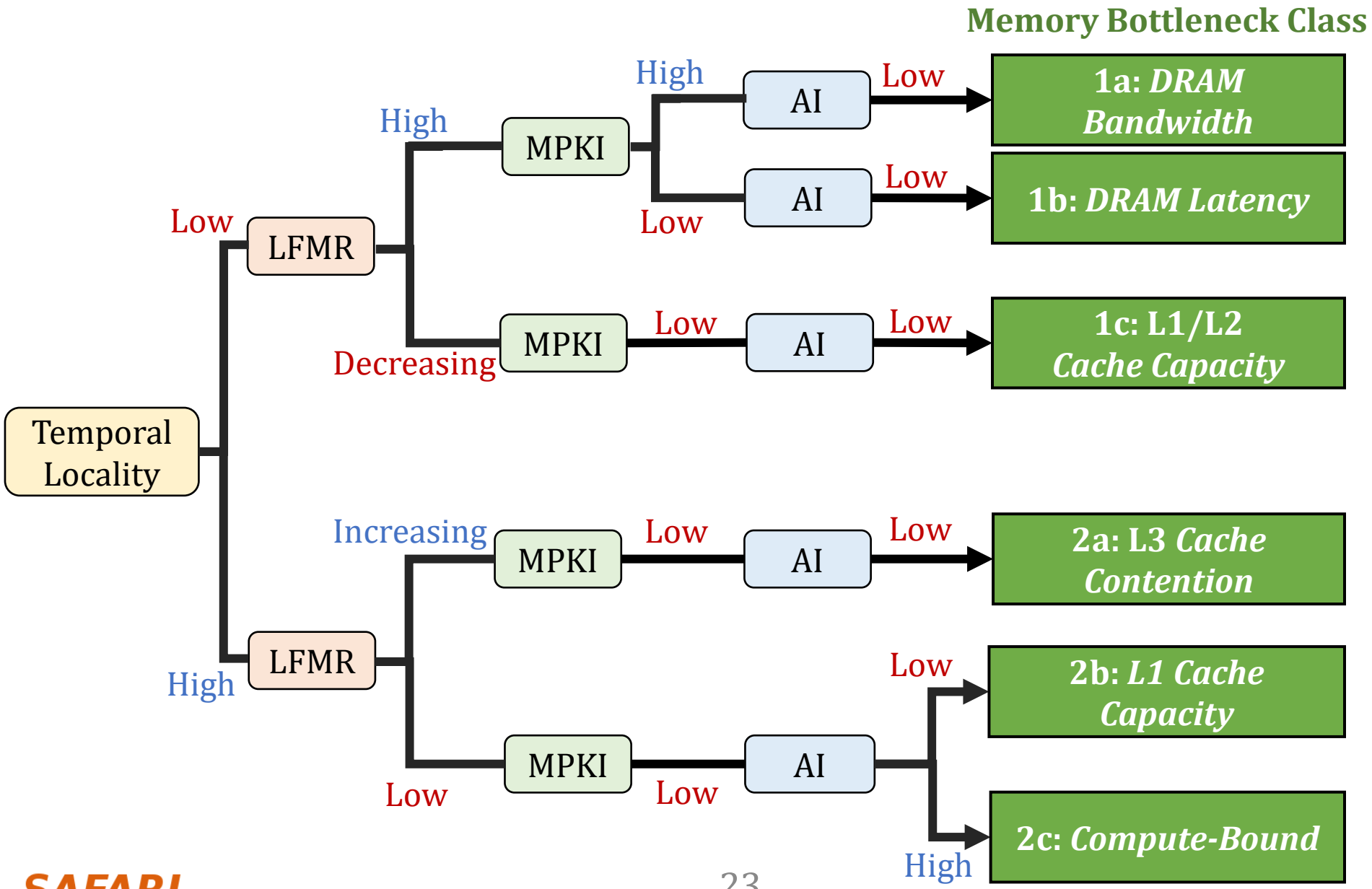
3. Enabling Complex Operations using DRAM

- SIMDRAM Framework
- System Integration
- Evaluation

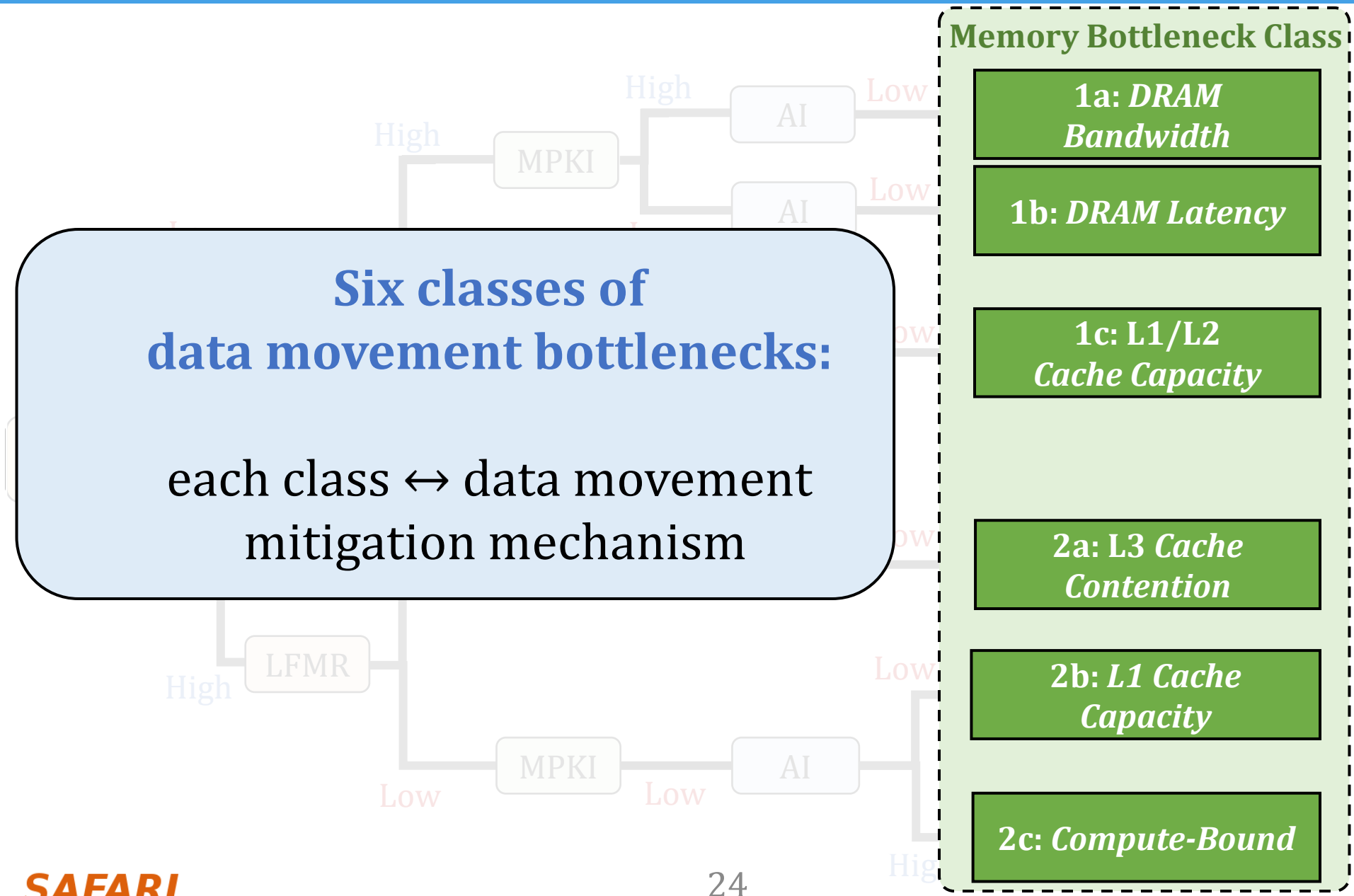
Step 3: Memory Bottleneck Analysis



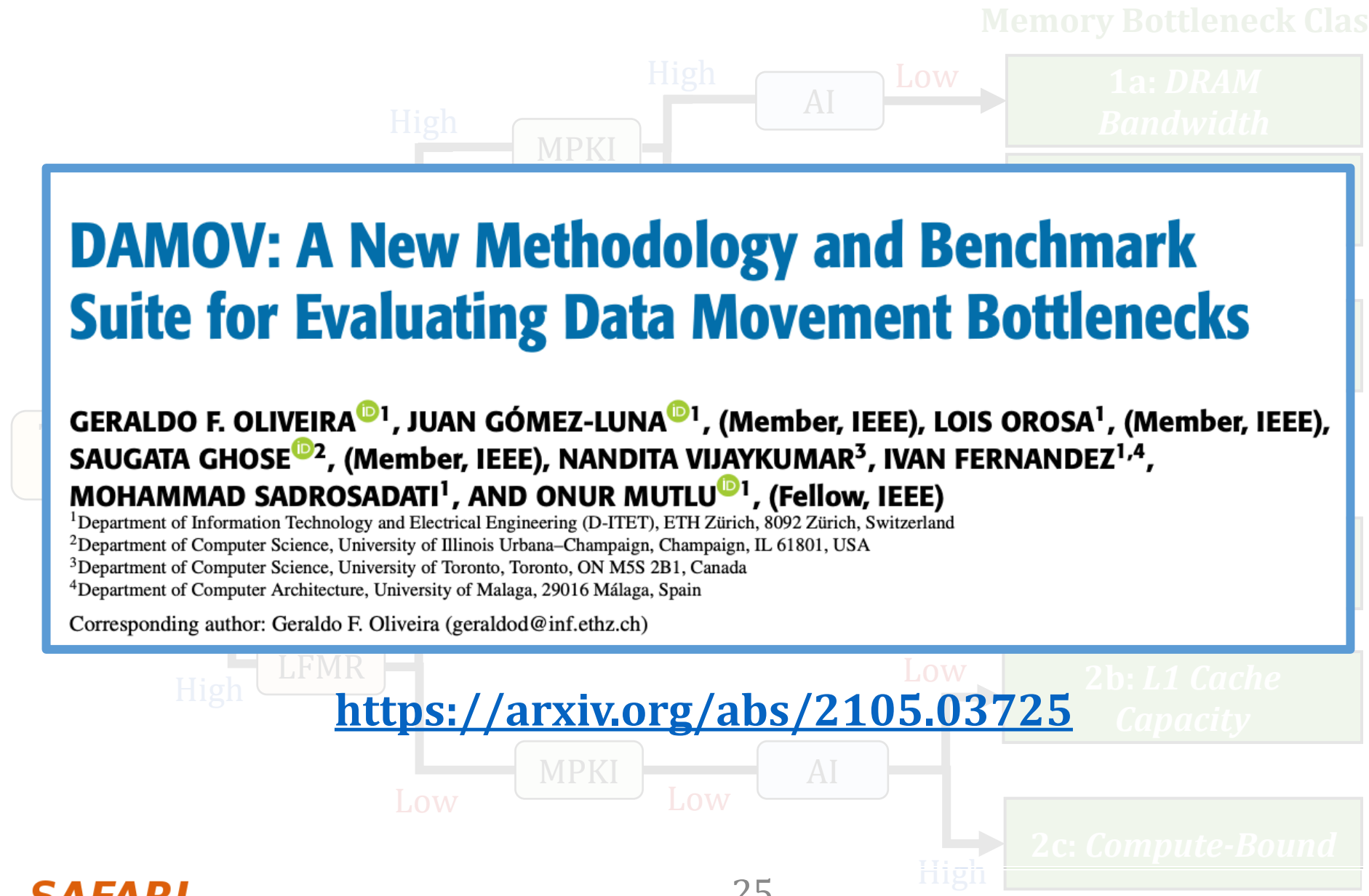
Step 3: Memory Bottleneck Analysis



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], (Member, IEEE), LOIS OROSA[1], (Member, IEEE), SAUGATA GHOSE[2], (Member, IEEE), NANDITA VIJAYKUMAR[3], IVAN FERNANDEZ[1,4], MOHAMMAD SADROSADATI[1], AND ONUR MUTLU[1], (Fellow, IEEE)

[1]Department of Information Technology and Electrical Engineering (D-ITET), ETH Zürich, 8092 Zürich, Switzerland
[2]Department of Computer Science, University of Illinois Urbana–Champaign, Champaign, IL 61801, USA
[3]Department of Computer Science, University of Toronto, Toronto, ON M5S 2B1, Canada
[4]Department of Computer Architecture, University of Malaga, 29016 Málaga, Spain

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

https://arxiv.org/abs/2105.03725

Outline

1. Introduction

2. Identifying Memory Bottlenecks

- Methodology Overview
- Application Profiling
- Locality-Based Clustering
- Memory Bottleneck Analysis
- **DAMOV Benchmark Suite**

3. Enabling Complex Operations using DRAM

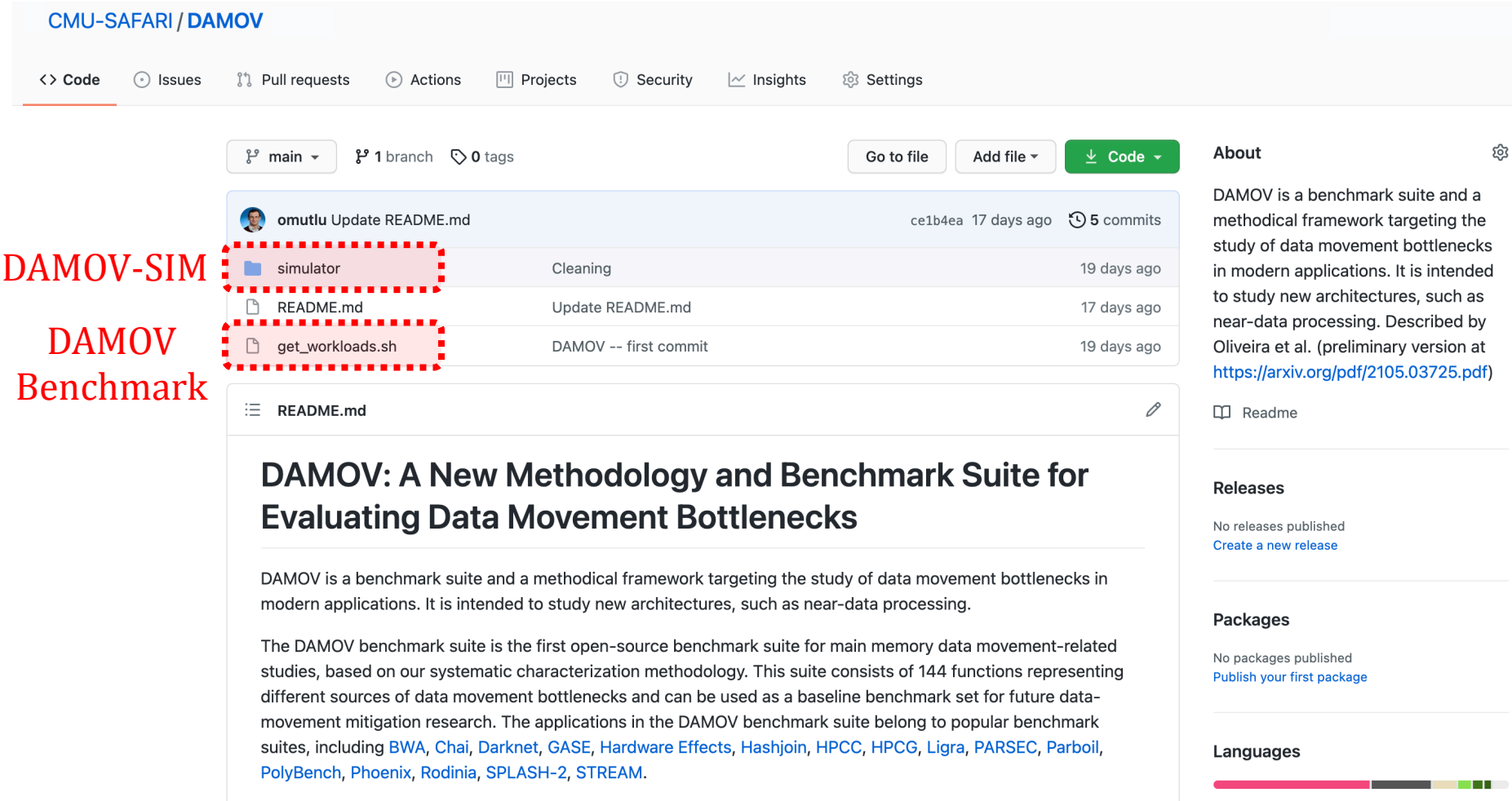
- SIMDRAM Framework
- System Integration
- Evaluation

DAMOV is Open-Source

- We open-source our benchmark suite and our toolchain

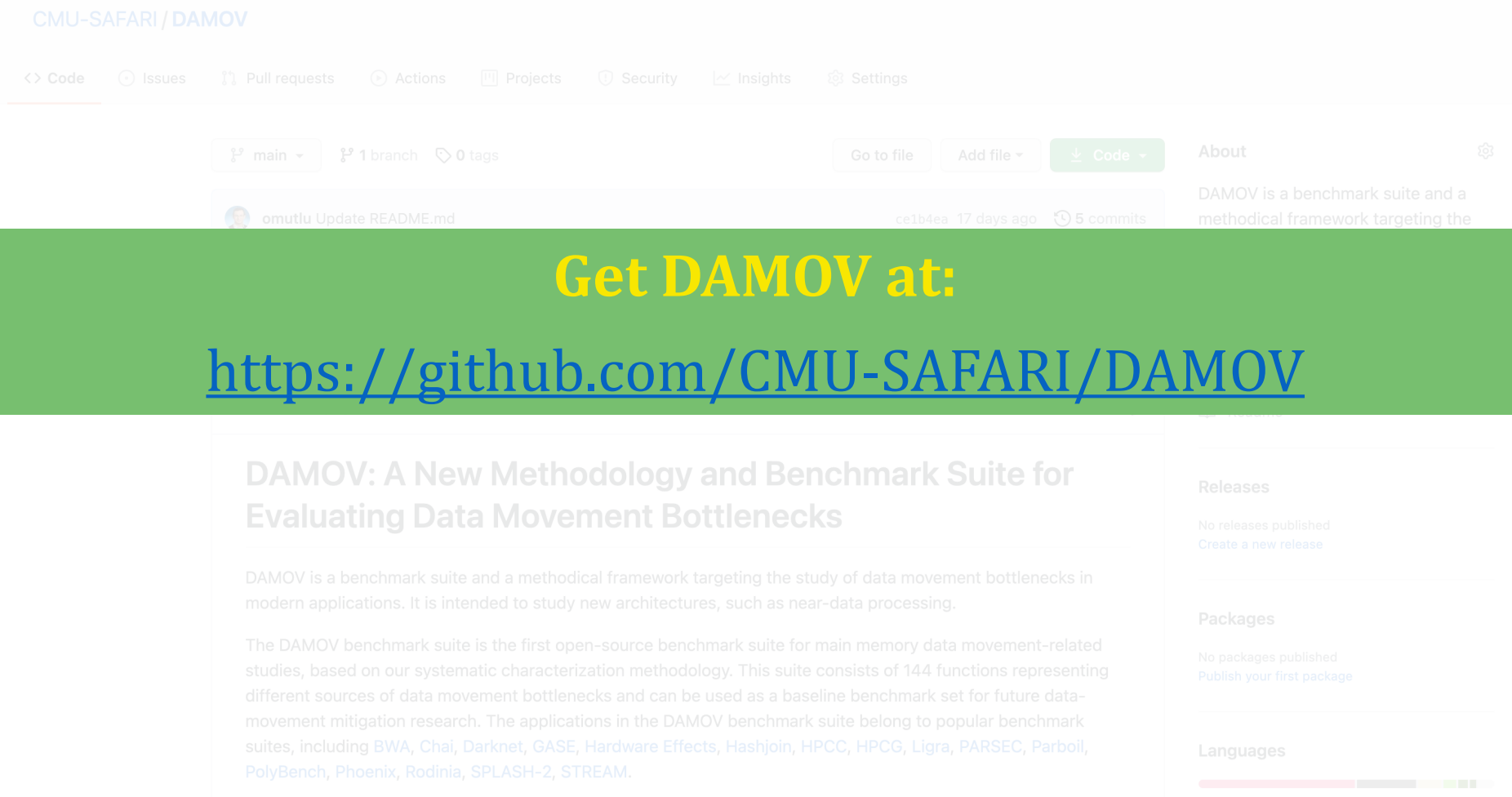
DAMOV-SIM

DAMOV Benchmark



DAMOV is Open-Source

- We open-source our benchmark suite and our toolchain



Get DAMOV at:

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

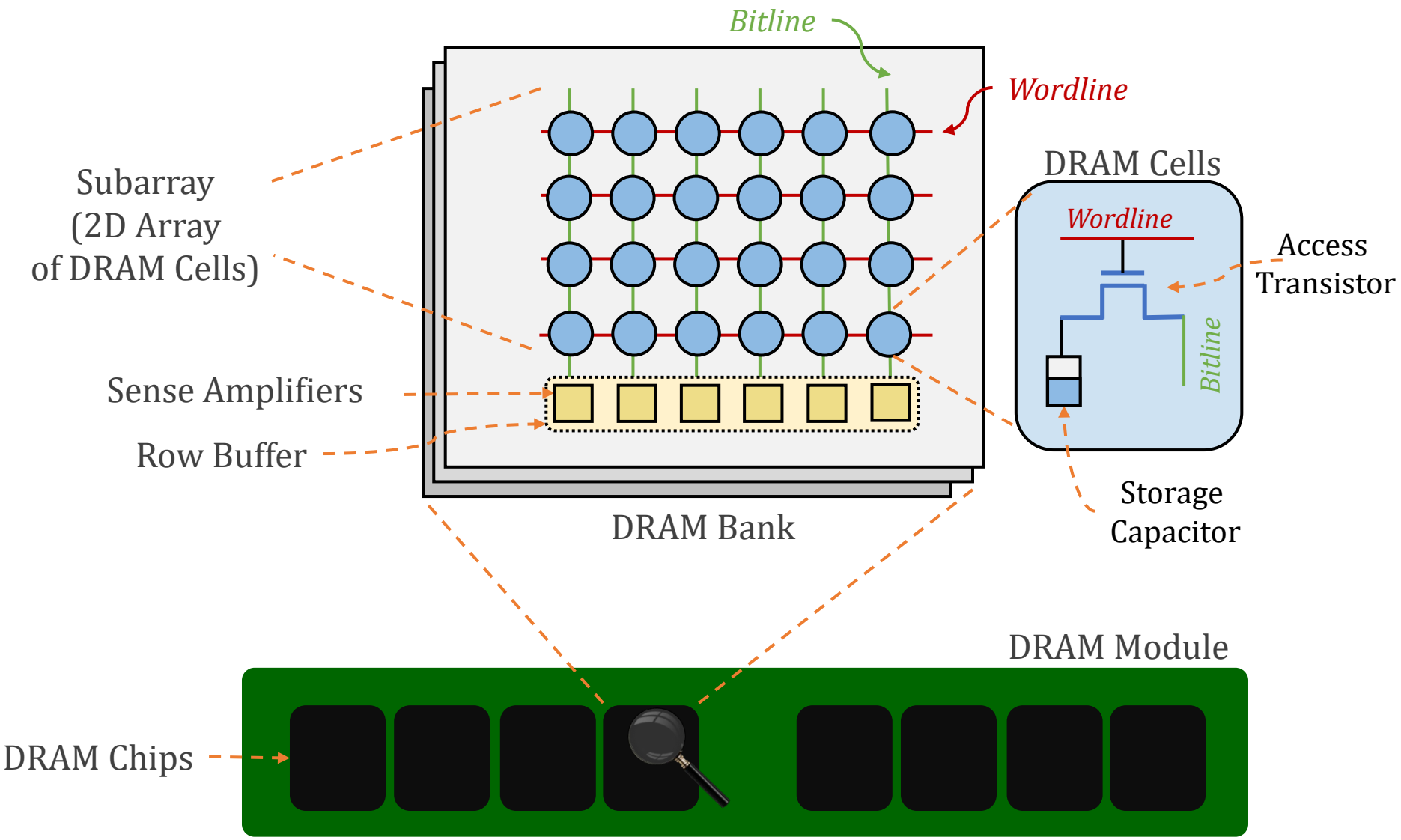
Outline

1. Introduction

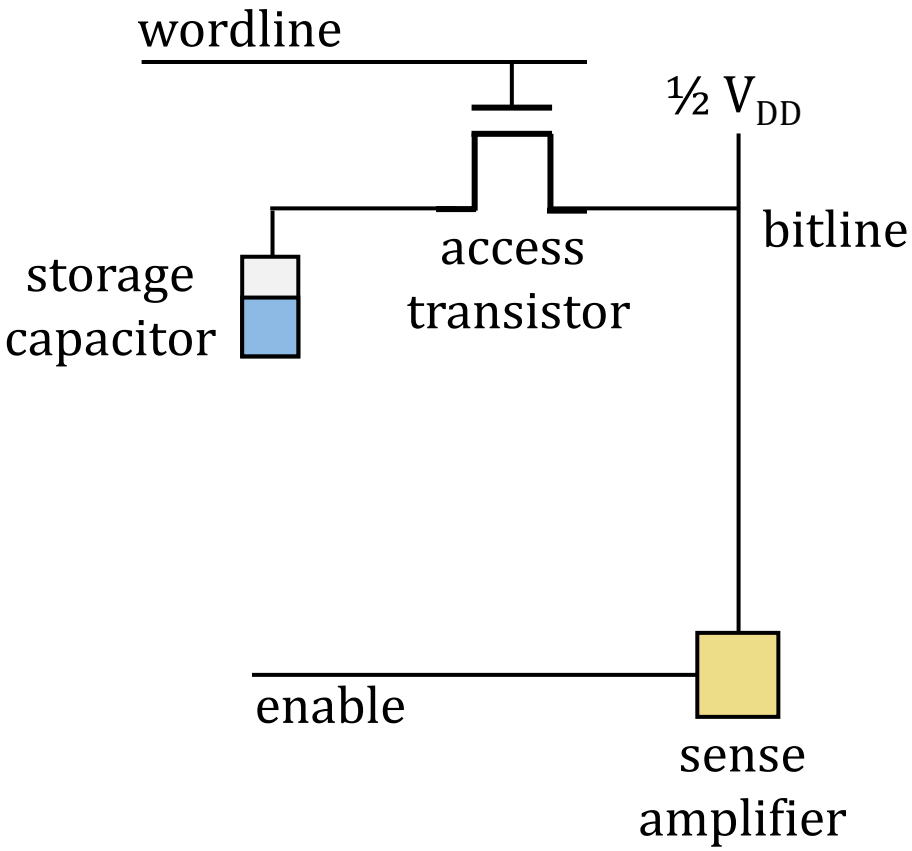
2. Identifying Memory Bottlenecks
   - Methodology Overview
   - Application Profiling
   - Locality-Based Clustering
   - Memory Bottleneck Analysis
   - DAMOV Benchmark Suite

3. Enabling Complex Operations using DRAM
   - SIMDRAM Framework
   - System Integration
   - Evaluation

Inside a DRAM Chip



DRAM Cell Operation

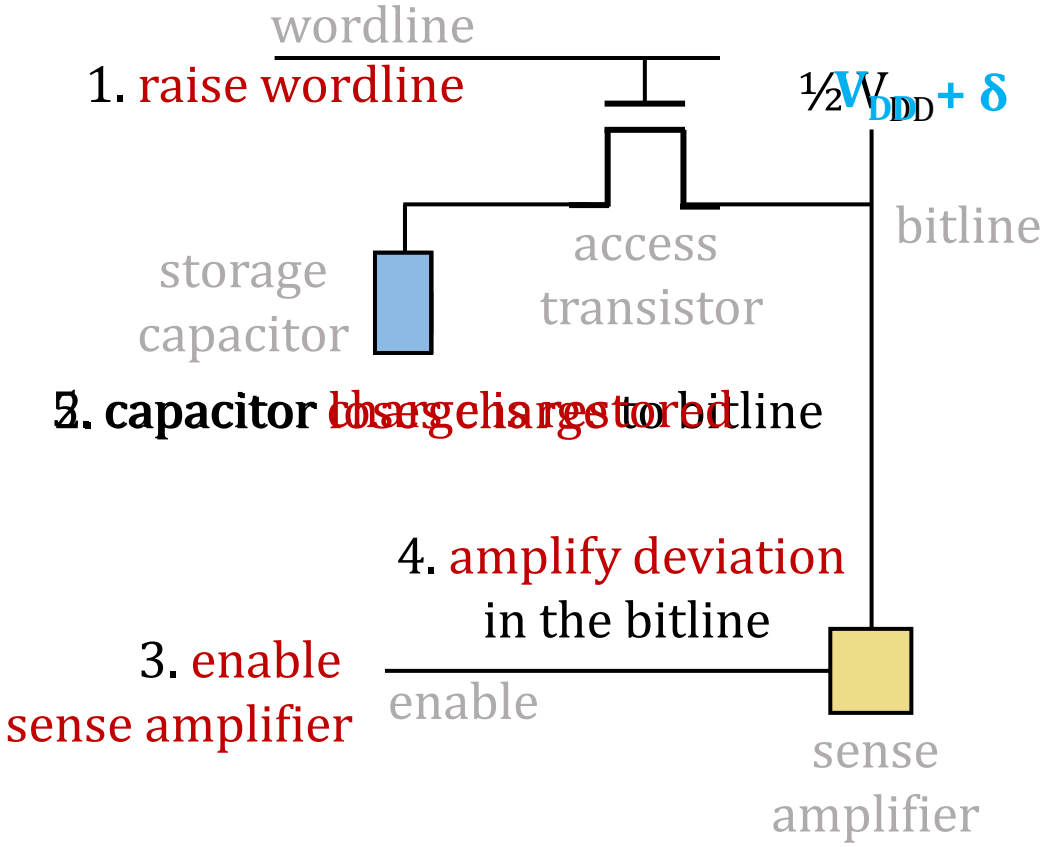


1. ACTIVATE (ACT)

2. READ/WRITE

3. PRECHARGE (PRE)

DRAM Cell Operation (1/3)

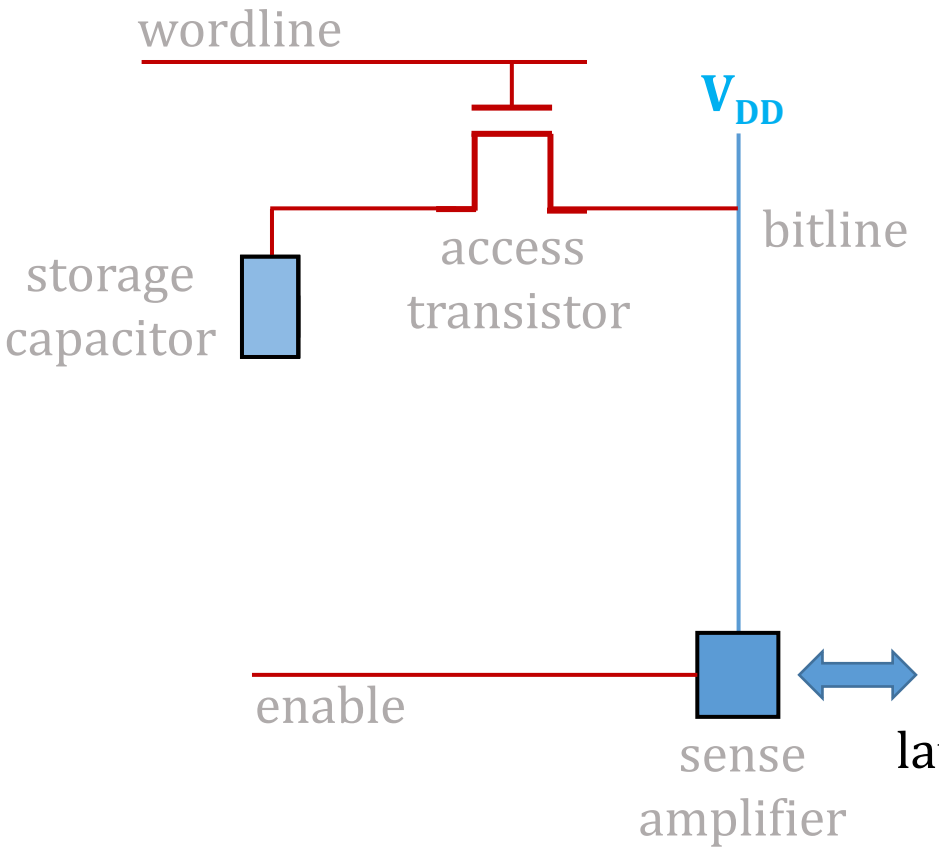


1. ACTIVATE (ACT)

2. READ/WRITE

3. PRECHARGE (PRE)

DRAM Cell Operation (2/3)



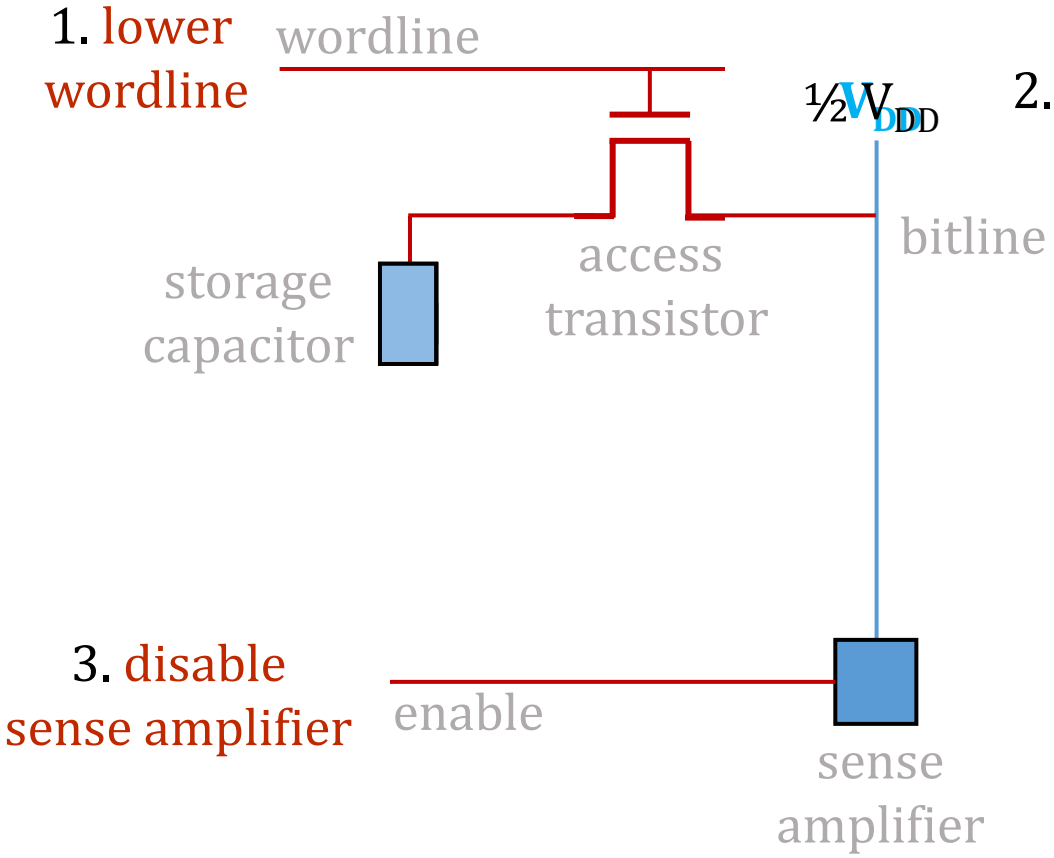
1. ACTIVATE (ACT)

2. READ/WRITE

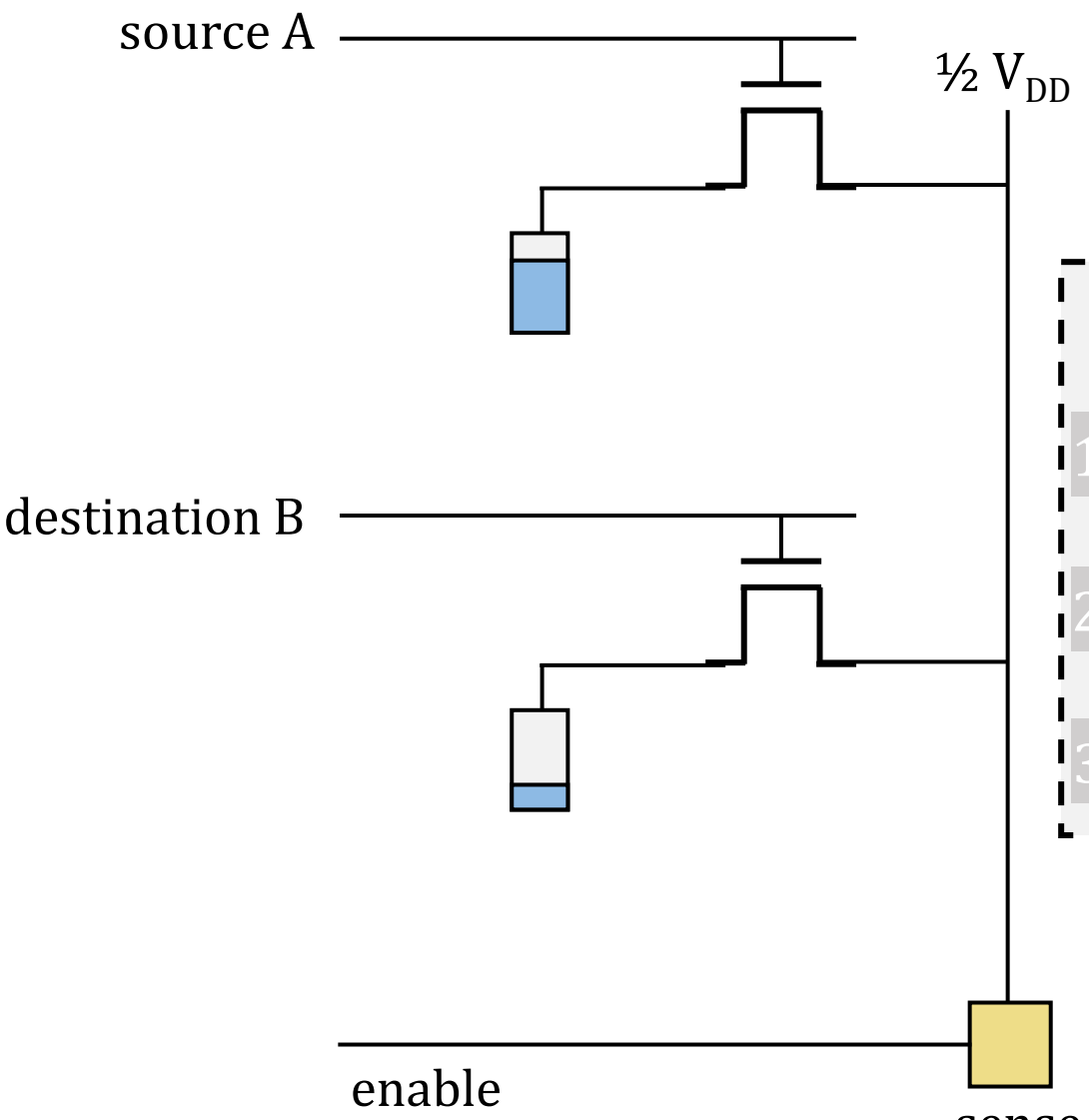
3. PRECHARGE (PRE)

read/write charge latched in sense amplifier

DRAM Cell Operation (3/3)



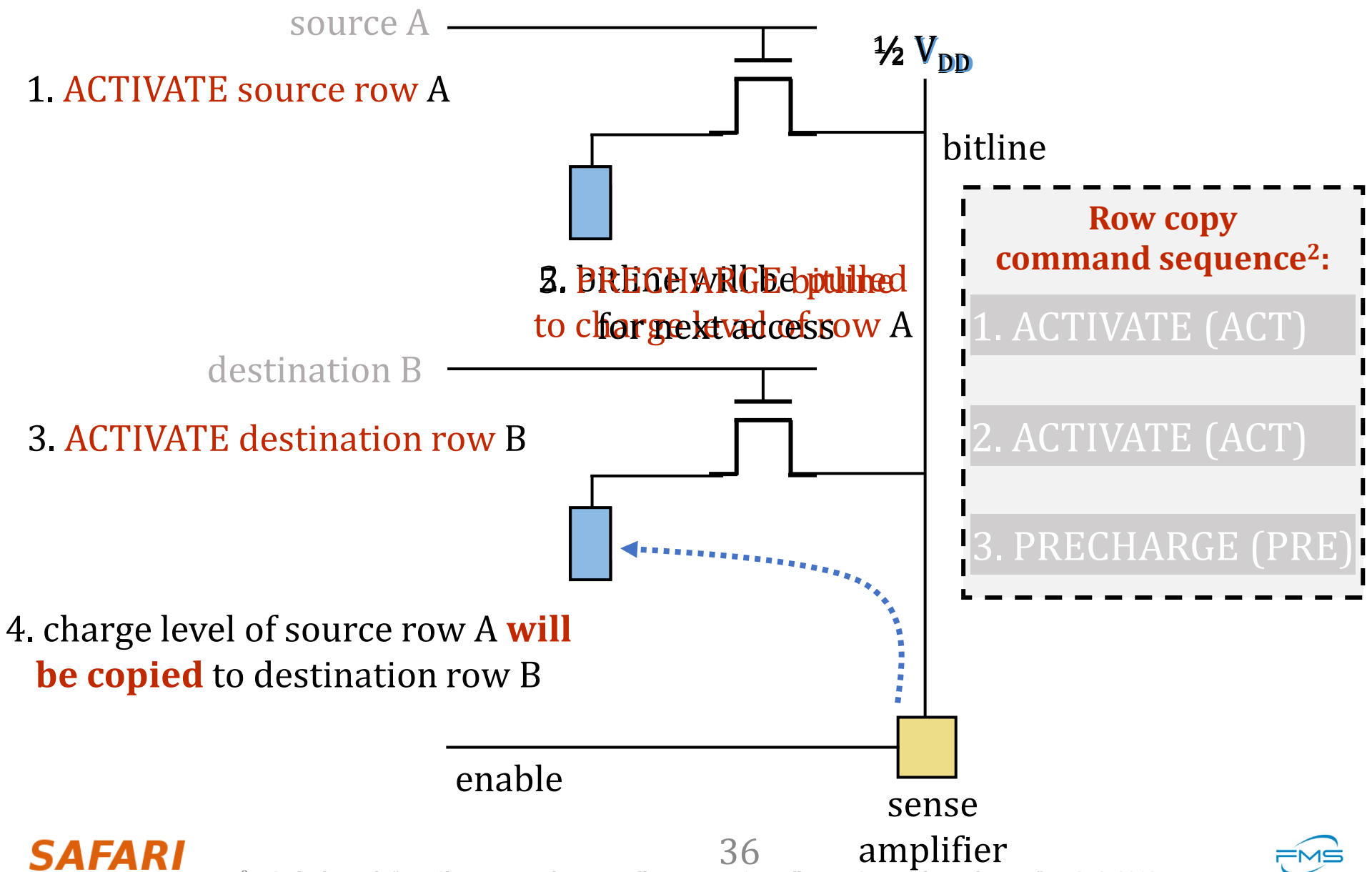
RowClone: In-DRAM Row Copy (1/2)



Row copy command sequence[2]:

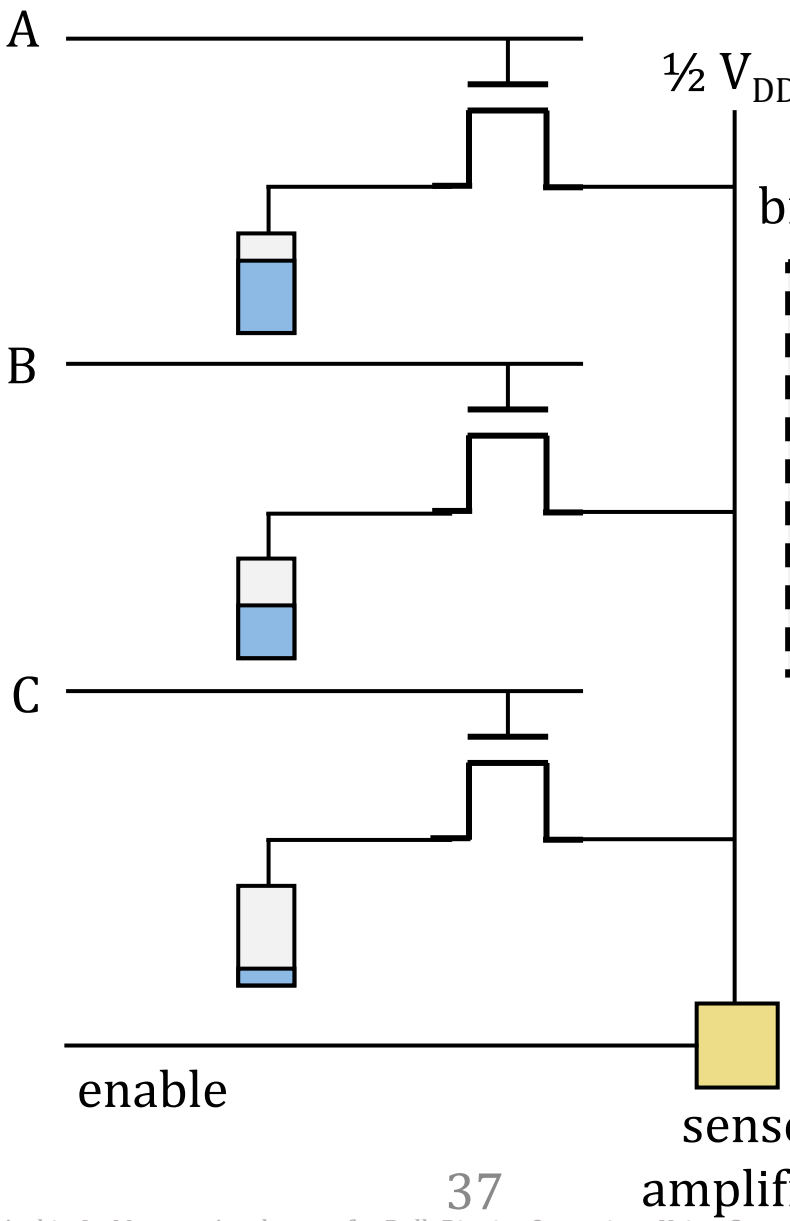
1. ACTIVATE (ACT)
2. ACTIVATE (ACT)
3. PRECHARGE (PRE)

RowClone: In-DRAM Row Copy (2/2)



[2] V. Seshadri et al., "*RowClone: Fast and Energy-Efficient In-DRAM Bulk Data Copy and Initialization*", MICRO, 2013

Triple-Row Activation: Majority Function



Majority function command sequence[3]:

1. ACTIVATE (ACT)
2. PRECHARGE (PRE)

[3] V. Seshadri et al., "*Ambit: In-Memory Accelerator for Bulk Bitwise Operations Using Commodity DRAM Technology*", MICRO, 2017

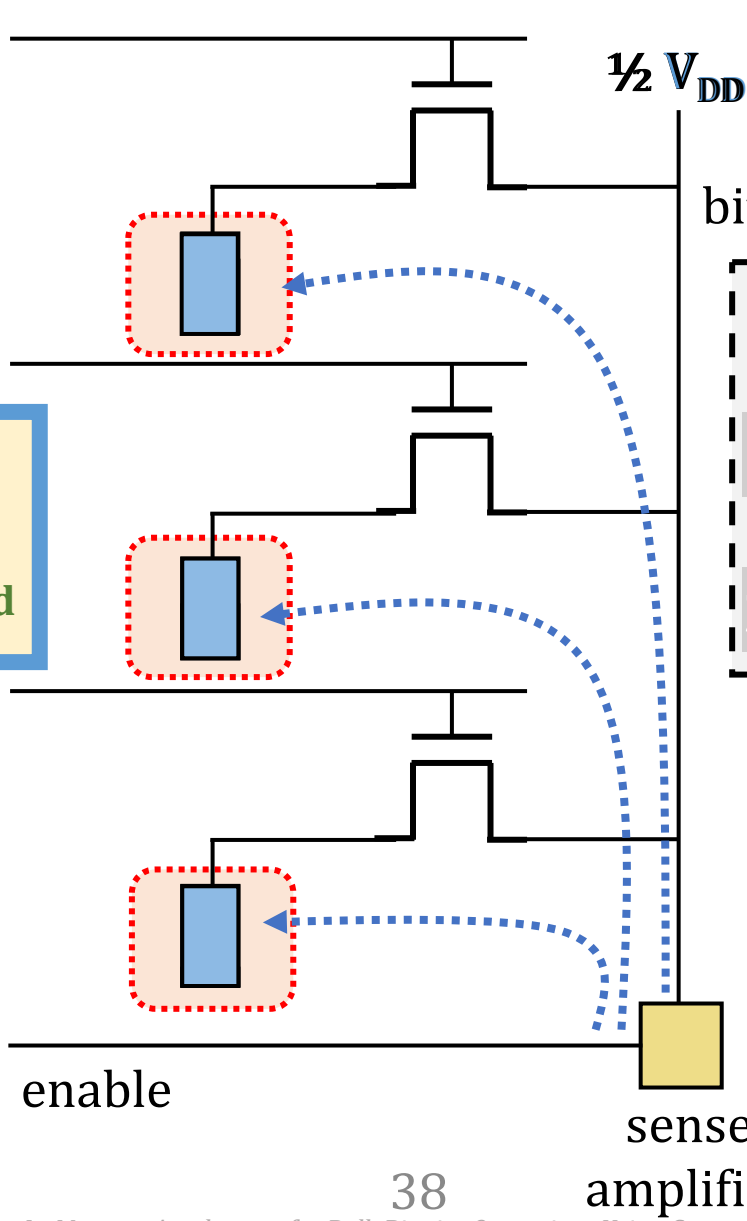
Triple-Row Activation: Majority Function

1. ACTIVATE three rows simultaneously
→ **triple-row activation**

MAJ(A, B, C) =
MAJ(V_{dd}, V_{dd}, 0) = V_{dd}

3. values in cells A, B, C will be overwritten with the majority output

4. PRECHARGE bitline for next access

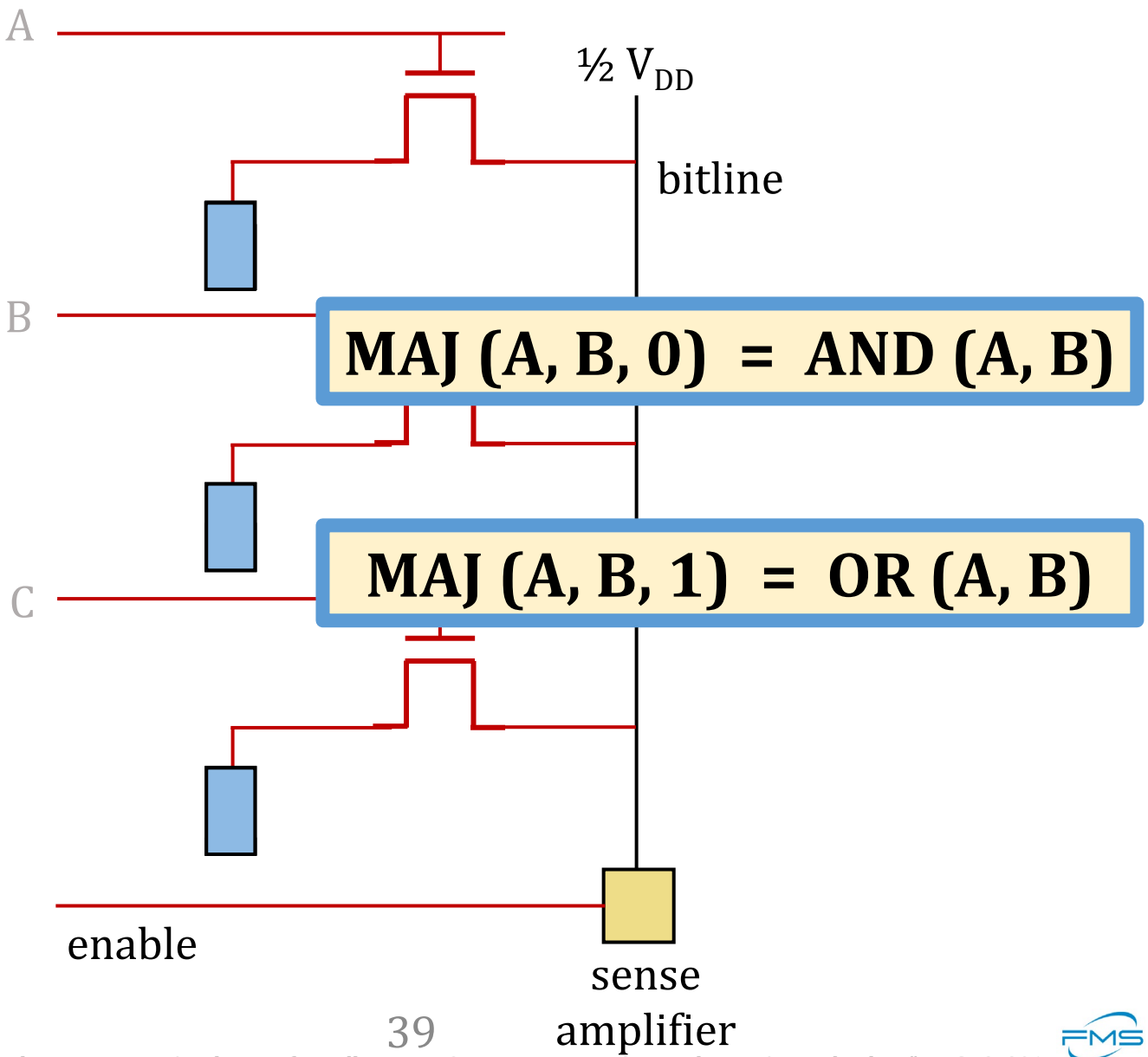


Majority function command sequence[3]:

1. ACTIVATE (ACT)
2. PRECHARGE (PRE)

[3] V. Seshadri et al., "*Ambit: In-Memory Accelerator for Bulk Bitwise Operations Using Commodity DRAM Technology*", MICRO, 2017

Ambit: In-DRAM Bulk Bitwise AND/OR



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

The Capability of COTS DRAM Chips

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

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

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

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

The Capability of COTS DRAM Chips

We demonstrate that COTS DRAM chips:

1

2

3

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

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

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

ETH Zürich

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

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

A widely used approach for PuD is to perform bulk bitwise operations, i.e., bitwise operations on large bit vectors. To perform bulk bitwise operations using DRAM, prior works propose modifications to the DRAM circuitry [29–31, 33, 35, 36, 43, 44, 46, 48–58]. Recent works [38, 41, 42, 45] experimentally demonstrate the feasibility of executing data copy & initializa-

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

PuM: Prior Works

- DRAM and other memory technologies that are capable of performing **computation using memory**

Shortcomings:

- Support **only basic** operations (e.g., Boolean operations, addition)
  - Not widely applicable
- Support a **limited** set of operations
  - Lack the flexibility to support new operations
- Require **significant changes** to the DRAM
  - Costly (e.g., area, power)

PuM: Prior Works

- DRAM and other memory technologies that are capable of performing **computation using memory**

Shortcomings:

- Support **only basic** operations (e.g., Boolean operations, addition)

Need a framework that aids general adoption of PuM, by:

- Efficiently implementing complex operations

- Providing flexibility to support new operations

- Costly (e.g., area, power)

Our Goal

Goal: Design a PuM framework that

- Efficiently implements complex operations
- Provides the flexibility to support new desired operations
- Minimally changes the DRAM architecture

Outline

1. Introduction

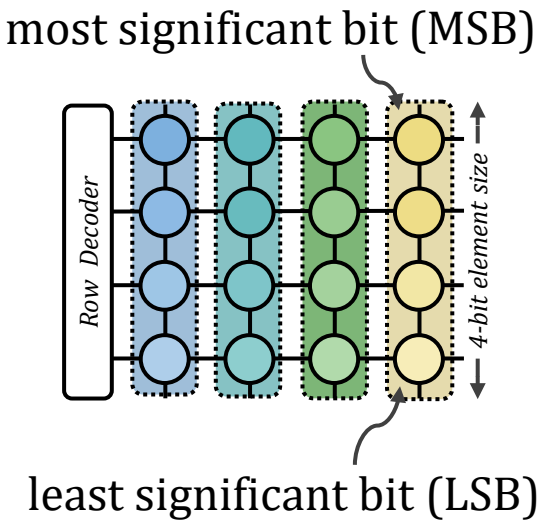
2. Identifying Memory Bottlenecks
   - Methodology Overview
   - Application Profiling
   - Locality-Based Clustering
   - Memory Bottleneck Analysis
   - DAMOV Benchmark Suite

3. Enabling Complex Operations using DRAM
   - **SIMDRAM Framework**
   - System Integration
   - Evaluation

SIMDRAM: PuM Substrate

- SIMDRAM framework is built around a DRAM substrate that enables two techniques:

(1) Vertical data layout

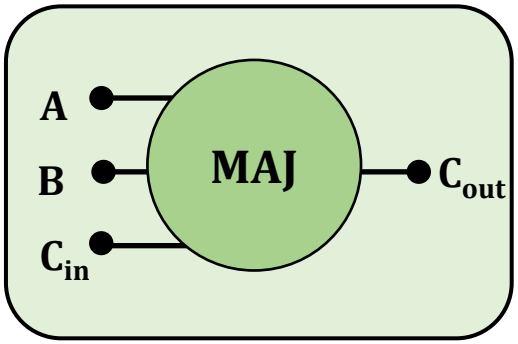


Pros compared to the conventional horizontal layout:

- Implicit shift operation
- Massive parallelism

(2) Majority-based computation

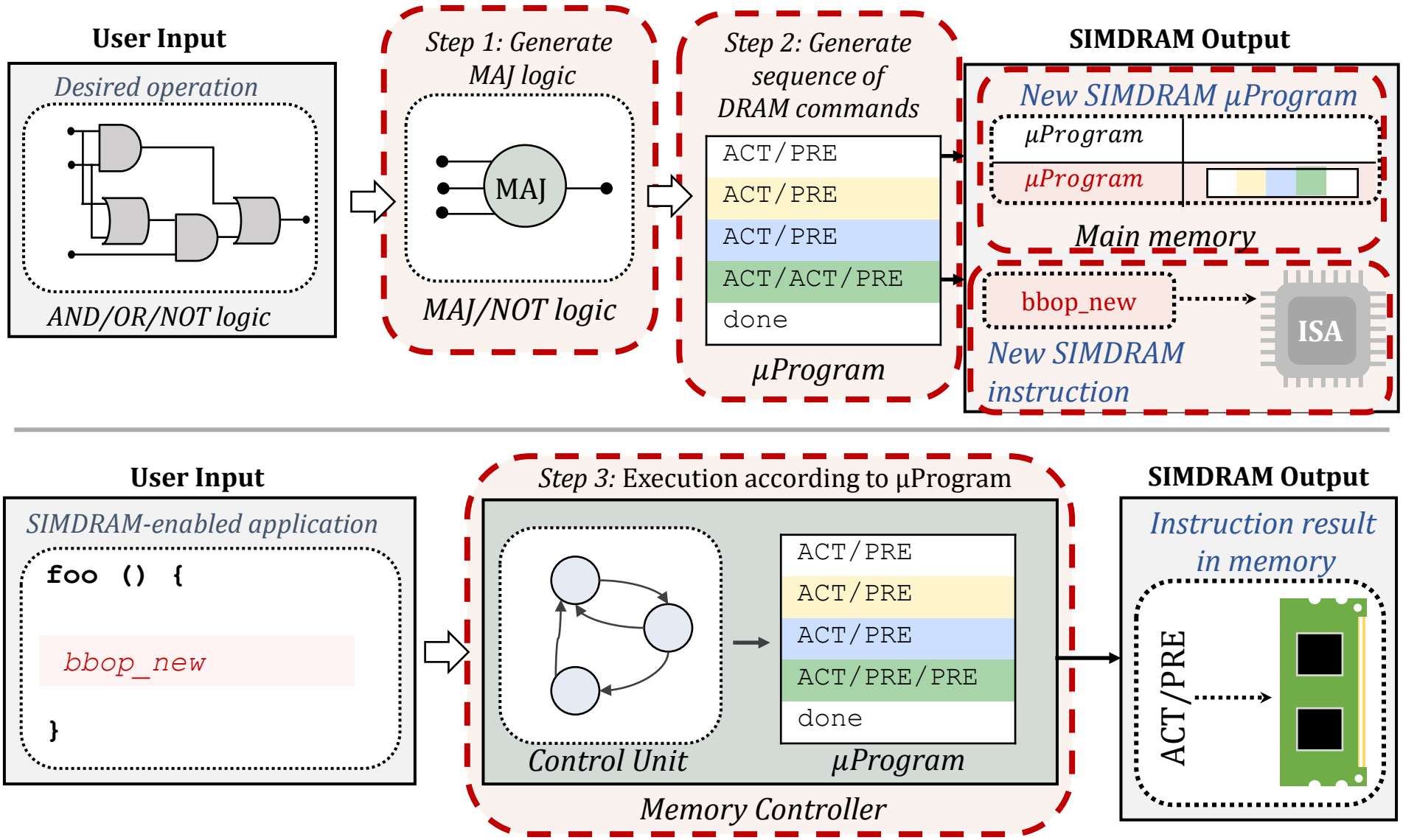
$C_{out} = AB + AC_{in} + BC_{in}$



Pros compared to AND/OR/NOT-based computation:

- Higher performance
- Higher throughput
- Lower energy consumption

SIMDRAM Framework



Outline

1. Introduction

2. Identifying Memory Bottlenecks
   - Methodology Overview
   - Application Profiling
   - Locality-Based Clustering
   - Memory Bottleneck Analysis
   - DAMOV Benchmark Suite

3. Enabling Complex Operations using DRAM
   - SIMDRAM Framework
   - **System Integration**
   - Evaluation

System Integration

- Efficiently transposing data
- Programming interface
- Handling page faults, address translation, coherence, and interrupts
- Handling limited subarray size
- Security implications
- Limitations of our framework

More in the Paper

SIMDRAM: An End-to-End Framework for Bit-Serial SIMD Computing in DRAM

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

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

Handling page faults, address translation, coherence, and interrupts

Handling limited subarray size

Security implications

Limitations of our framework

Outline

1. Introduction

2. Identifying Memory Bottlenecks
   - Methodology Overview
   - Application Profiling
   - Locality-Based Clustering
   - Memory Bottleneck Analysis
   - DAMOV Benchmark Suite

3. Enabling Complex Operations using DRAM
   - SIMDRAM Framework
   - System Integration
   - **Evaluation**

Methodology: Experimental Setup

- **Simulator:** gem5

- **Baselines:**
  - A multi-core CPU (Intel Skylake)
  - A high-end GPU (NVidia Titan V)
  - Ambit: a state-of-the-art in-memory computing mechanism

- **Evaluated SIMDRAM configurations** (all using a DDR4 device):
  - **1-bank:** SIMDRAM exploits 65'536 SIMD lanes (an 8 kB row buffer)
  - **4-banks:** SIMDRAM exploits 262'144 SIMD lanes
  - **16-banks:** SIMDRAM exploits 1'048'576 SIMD lanes

Methodology: Workloads

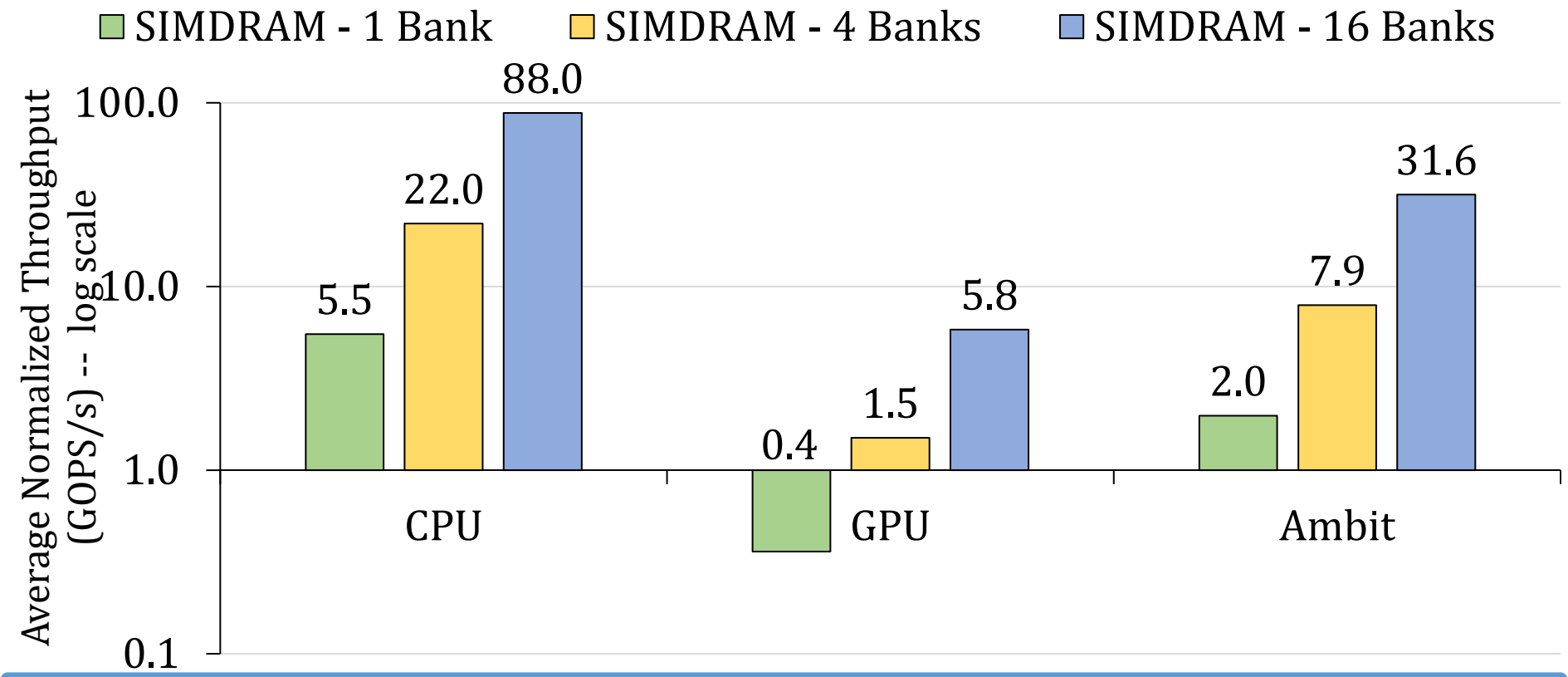
Evaluated:

- 16 complex in-DRAM operations:
  - Absolute
  - Addition/Subtraction
  - BitCount Reduction
  - Equality/ Greater/Greater Equal
  - Predication
  - ReLU
  - AND-/OR-/XOR-
  - Division/Multiplication

- 7 real-world applications
  - BitWeaving (databases)
  - TPH-H (databases)
  - kNN (machine learning)
  - LeNET (Neural Networks)
  - VGG-13/VGG-16 (Neural Networks)
  - brightness (graphics)

Throughput Analysis

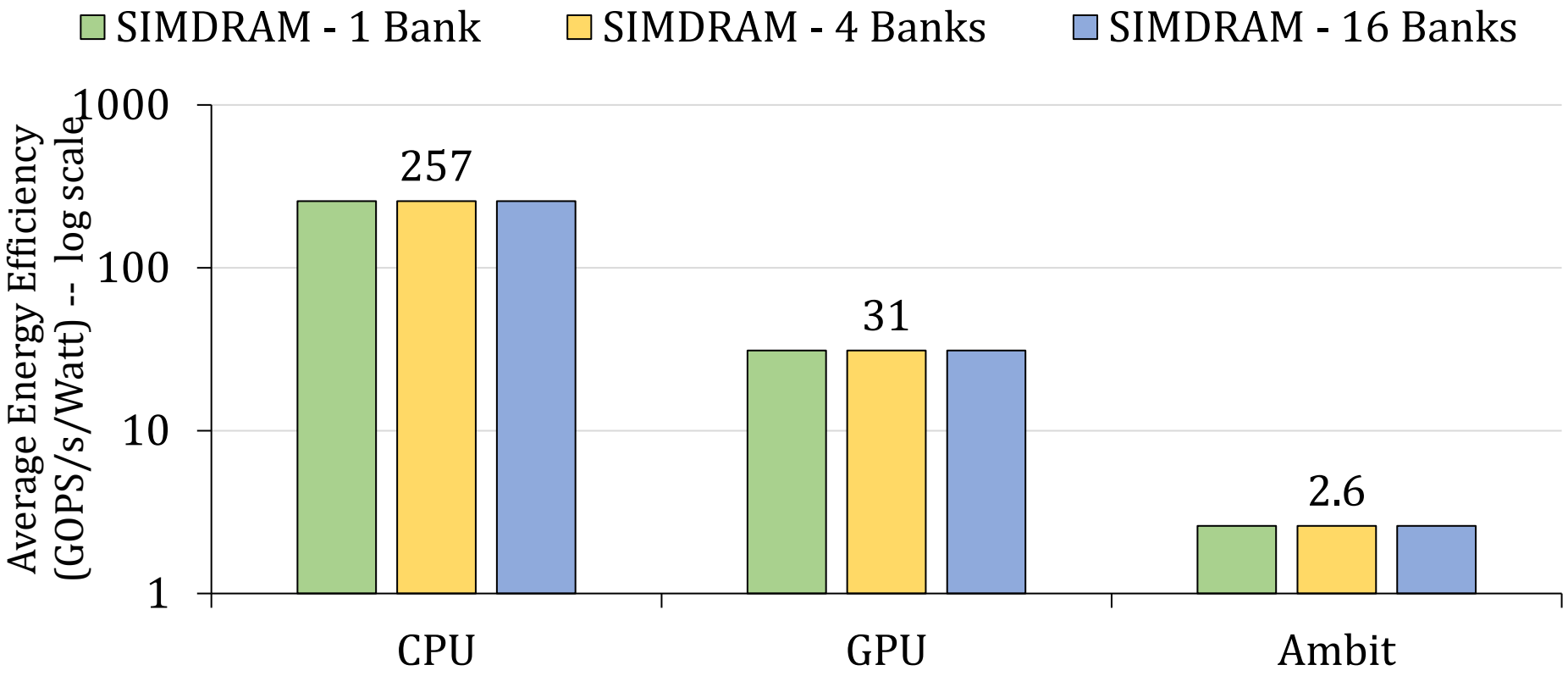
Average normalized throughput across all 16 SIMDRAM operations



SIMDRAM significantly outperforms all state-of-the-art baselines for a wide range of operations

Energy Analysis

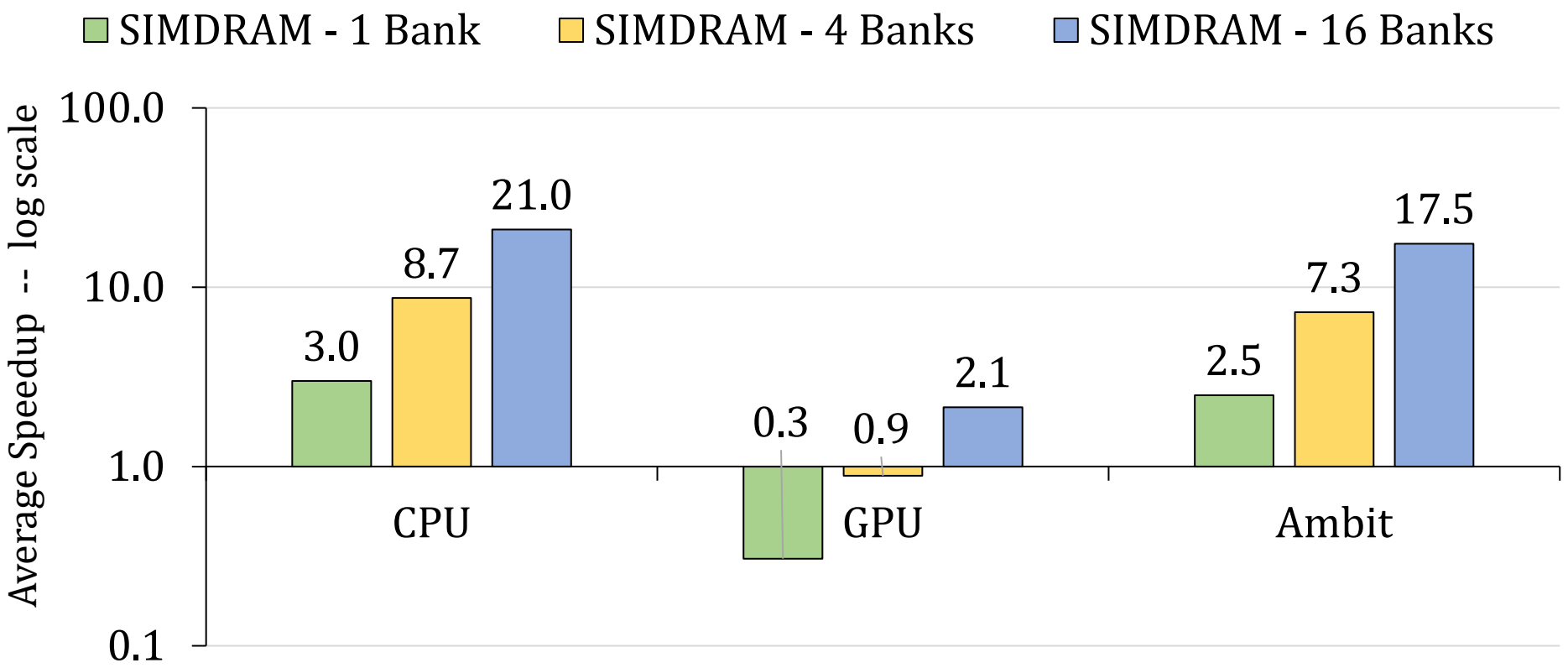
Average normalized energy efficiency across all 16 SIMDRAM operations



SIMDRAM is more energy-efficient than all state-of-the-art baselines for a wide range of operations

Real-World Application

Average speedup across 7 real-world applications



SIMDRAM effectively and efficiently accelerates many commonly-used real-world applications

In this Work

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

To fully supportt PIM systems, we need to develop:

1. Workload characterization methodologies and benchmark suites targeting PIM architectures
2. Frameworks that can facilitate the implementation of complex operations and algorithms using PIM primitives
3. **Compiler support and compiler optimizations targeting PIM architectures**
4. Operating system support for PIM-aware virtual memory, memory management, data allocation and mapping
5. Efficient data coherence and consistency mechanisms

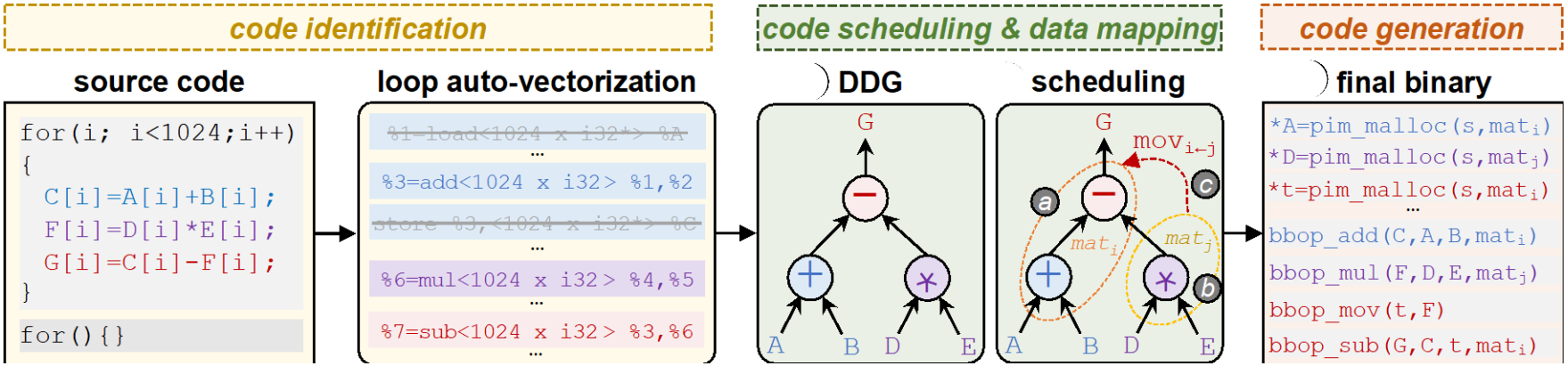
MIMDRAM: Programmer-Transparent PuM

- MIMDRAM: a hardware/software co-designed PuM

Goal

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

Three new LLVM-based passes targeting PUD execution



MIMDRAM: Programmer-Transparent PuM

- MIMDRAM: a hardware/software co-designed PuM

Transparently:
extract SIMD parallelism from an application, and

2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)

MIMDRAM: An End-to-End Processing-using-DRAM System for Energy-Efficient and Programmer-Transparent MIMD Computing

Geraldo F. Oliveira† Ataberk Olgun† A. Giray Yaglikci† Nisa Bostanci†
Juan Gómez-Luna† Saugata Ghose‡ Onur Mutlu†

† ETH Zürich ‡ University of Illinois at Urbana-Champaign

https://arxiv.org/abs/2402.19080

In this Work

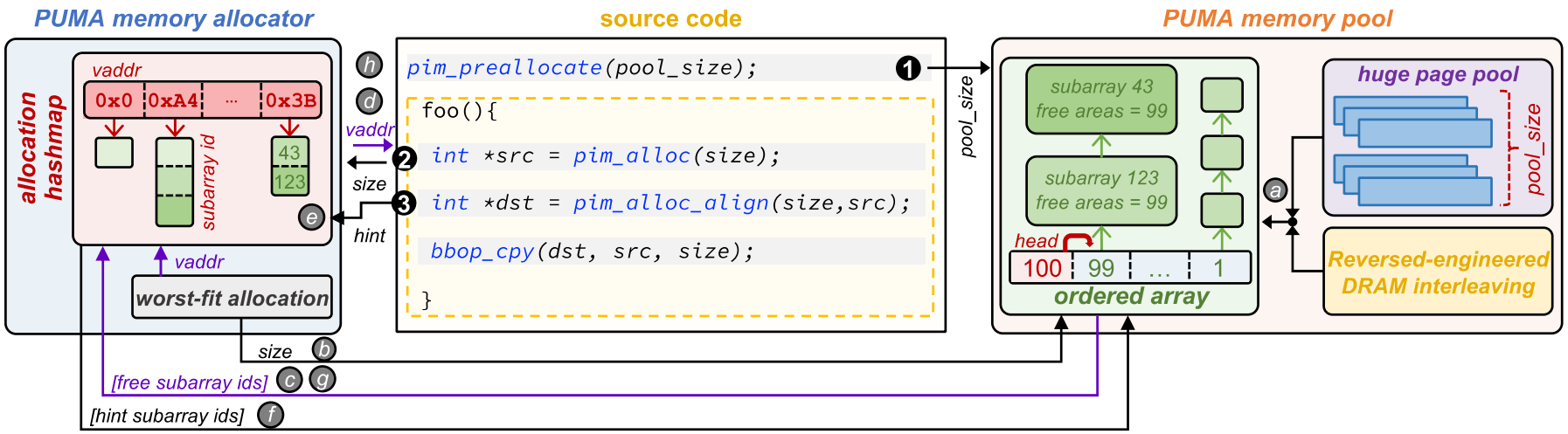
The lack of tools and system support for PIM architectures limit the adoption of PIM system

To fully supportt PIM systems, we need to develop:

1. Workload characterization methodologies and benchmark suites targeting PIM architectures
2. Frameworks that can facilitate the implementation of complex operations and algorithms using PIM primitives
3. Compiler support and compiler optimizations targeting PIM architectures
4. **Operating system support for PIM-aware virtual memory, memory management, data allocation and mapping**
5. Efficient data coherence and consistency mechanisms

PUMA: Low-Cost Data Allocation & Alignment

- PUMA: a flexible memory allocation mechanism that
  - allows programmers to have control over physical memory allocation
  - enables PUD execution from the operating system viewpoint



PUMA: Low-Cost Data Allocation & Alignment

- PUMA: a flexible memory allocation mechanism that
  - allows programmers to have control over physical memory allocation
  - e

PUMA: Efficient and Low-Cost Memory Allocation and Alignment Support for Processing-Using-Memory Architectures

Geraldo F. Oliveira Emanuele G. Esposito Juan Gómez-Luna Onur Mutlu

ETH Zürich

1. Motivation & Problem

Processing-in-memory (PIM) [1–12] is a promising paradigm that aims to alleviate the ever-growing cost of moving data back and forth between computing (e.g., CPU, GPU, accelerators) and memory (e.g., caches, main memory, storage) elements. In PIM architectures, computation is done by adding logic units *near* memory arrays, i.e., processing-near-

We observe that (i) independently of the allocation size for input operands, using `malloc` and `posix_memalign` memory allocators results in 0% of the operations being executed in the PUD substrate due to data misalignment; and (ii) for large-enough allocation sizes (e.g., 32 Kb), *only* up 60% of the PUD operations that use `huge pages`-based memory allocation can successfully be executed in DRAM.

https://arxiv.org/pdf/2403.04539

In this Work

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

To fully supportt PIM systems, we need to develop:

1. **Workload characterization methodologies and benchmark suites targeting PIM architectures**
2. **Frameworks that can facilitate the implementation of complex operations and algorithms using PIM primitives**
3. **Compiler support and compiler optimizations targeting PIM architectures**
4. **Operating system support for PIM-aware virtual memory, memory management, data allocation and mapping**
5. **Efficient data coherence and consistency mechanisms**

PIM Review and Open Problems

A Modern Primer on Processing in Memory

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

SAFARI Research Group

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

Onur Mutlu, Saugata Ghose, Juan Gomez-Luna, and Rachata Ausavarungnirun,
"A Modern Primer on Processing in Memory"
Invited Book Chapter in ***Emerging Computing: From Devices to Systems - Looking Beyond Moore and Von Neumann***, Springer, 2023

Methodologies, Workloads, and Tools for Processing-in-Memory: Enabling the Adoption of Data-Centric Architectures

Geraldo F. Oliveira

(https://geraldofojunior.github.io/)

Onur Mutlu