

PIM Adoption Issues

How to Enable PIM Adoption?

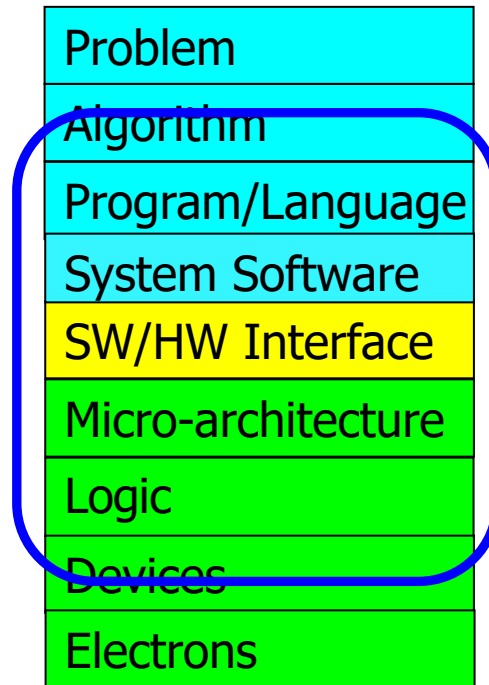
Dr. Juan Gómez Luna
Professor Onur Mutlu

Potential Barriers to Adoption of PIM

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

All can be solved with change of mindset

We Need to Revisit the Entire Stack



We can get there step by step

PIM Review and Open Problems

A Modern Primer on Processing in Memory

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

SAFARI Research Group

^a*ETH Zürich*

^b*Carnegie Mellon University*

^c*University of Illinois at Urbana-Champaign*

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

Onur Mutlu, Saugata Ghose, Juan Gomez-Luna, and Rachata Ausavarungnirun,

"A Modern Primer on Processing in Memory"

*Invited Book Chapter in **Emerging Computing: From Devices to Systems - Looking Beyond Moore and Von Neumann**, Springer.*

1 Introduction	2
2 Major Trends Affecting Main Memory	4
3 The Need for Intelligent Memory Controllers to Enhance Memory Scaling	6
4 Perils of Processor-Centric Design	9
5 Processing-in-Memory (PIM): Technology Enablers and Two Approaches	12
5.1 New Technology Enablers: 3D-Stacked Memory and Non-Volatile Memory . . .	12
5.2 Two Approaches: Processing Using Memory (PUM) vs. Processing Near Memory (PNM)	13
6 Processing Using Memory (PUM)	14
6.1 RowClone	14
6.2 Ambit	15
6.3 Gather-Scatter DRAM	17
6.4 In-DRAM Security Primitives	17
7 Processing Near Memory (PNM)	18
7.1 Tesseract: Coarse-Grained Application-Level PNM Acceleration of Graph Processing	19
7.2 Function-Level PNM Acceleration of Mobile Consumer Workloads	20
7.3 Programmer-Transparent Function-Level PNM Acceleration of GPU Applications	21
7.4 Instruction-Level PNM Acceleration with PIM-Enabled Instructions (PEI) . .	21
7.5 Function-Level PNM Acceleration of Genome Analysis Workloads	22
7.6 Application-Level PNM Acceleration of Time Series Analysis	23
8 Enabling the Adoption of PIM	24
8.1 Programming Models and Code Generation for PIM	24
8.2 PIM Runtime: Scheduling and Data Mapping	25
8.3 Memory Coherence	27
8.4 Virtual Memory Support	27
8.5 Data Structures for PIM	28
8.6 Benchmarks and Simulation Infrastructures	29
8.7 Real PIM Hardware Systems and Prototypes	30
8.8 Security Considerations	30
9 Conclusion and Future Outlook	31

Main memory, built using the Dynamic Random Access Memory (DRAM) technology, is a major component in nearly all computing systems, including servers, cloud platforms, mobile/embedded devices, and sensor systems. Across all of these systems, the data working set sizes of modern applications are rapidly growing, while the need for fast analysis of such data is increasing. Thus, main memory is becoming an increasingly significant bottleneck across a wide variety of computing systems and applications [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]. Alleviating the main memory bottleneck requires the memory capacity, energy, cost, and performance to all scale in an efficient manner across technology generations. Unfortunately, it has become increasingly difficult in recent years, especially the past decade, to scale all of these dimensions [1, 2, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49], and thus the main memory bottleneck has been worsening.

A major reason for the main memory bottleneck is the high energy and latency cost associated with *data movement*. In modern computers, to perform any operation on data that resides in main memory, the processor must retrieve the data from main memory. This requires the memory controller to issue commands to a DRAM module across a relatively slow and power-hungry off-chip bus (known as the *memory channel*). The DRAM module sends the requested data across the memory channel, after which the data is placed in the caches and registers. The CPU can perform computation on the data once the data is in its registers. Data movement from the DRAM to the CPU incurs long latency and consumes a significant amount of energy [7, 50, 51, 52, 53, 54]. These costs are often exacerbated by the fact that much of the data brought into the caches is *not reused* by the CPU [52, 53, 55, 56], providing little benefit in return for the high latency and energy cost.

The cost of data movement is a fundamental issue with the *processor-centric* nature of contemporary computer systems. The CPU is considered to be the master in the system, and computation is performed only in the processor (and accelerators). In contrast, data storage and communication units, including the main memory, are treated as unintelligent workers that are incapable of computation. As a result of this processor-centric design paradigm, data moves a lot in the system between the computation units and communication/ storage units so that computation can be done on it. With the increasingly *data-centric* nature of contemporary and emerging appli-

PIM Review and Open Problems (II)

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]*

PIM Review and Open Problems (III)

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

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

[†]*Carnegie Mellon University*

[§]*ETH Zürich*

Saugata Ghose, Amirali Boroumand, Jeremie S. Kim, Juan Gomez-Luna, and Onur Mutlu,

"Processing-in-Memory: A Workload-Driven Perspective"

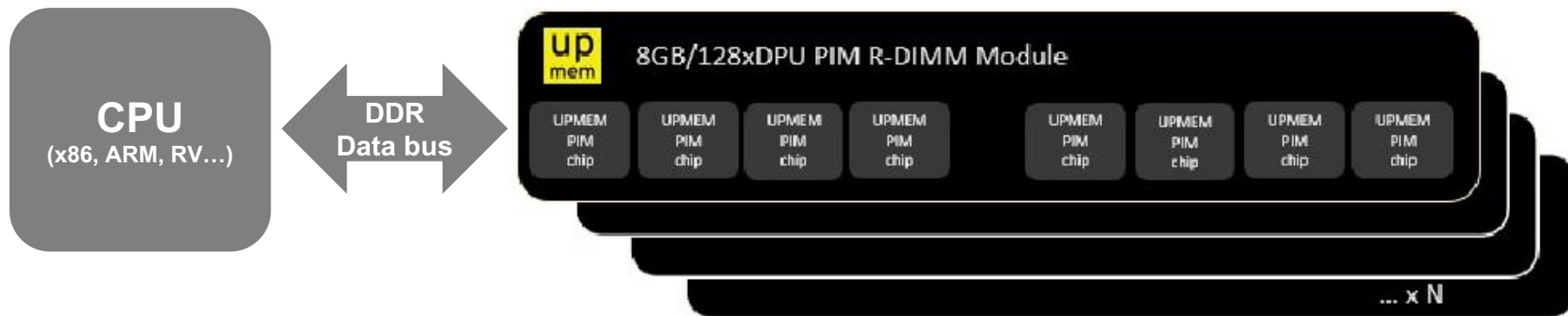
Invited Article in IBM Journal of Research & Development, Special Issue on Hardware for Artificial Intelligence, to appear in November 2019.

[Preliminary arXiv version]

Real PIM Hardware Systems and Prototypes

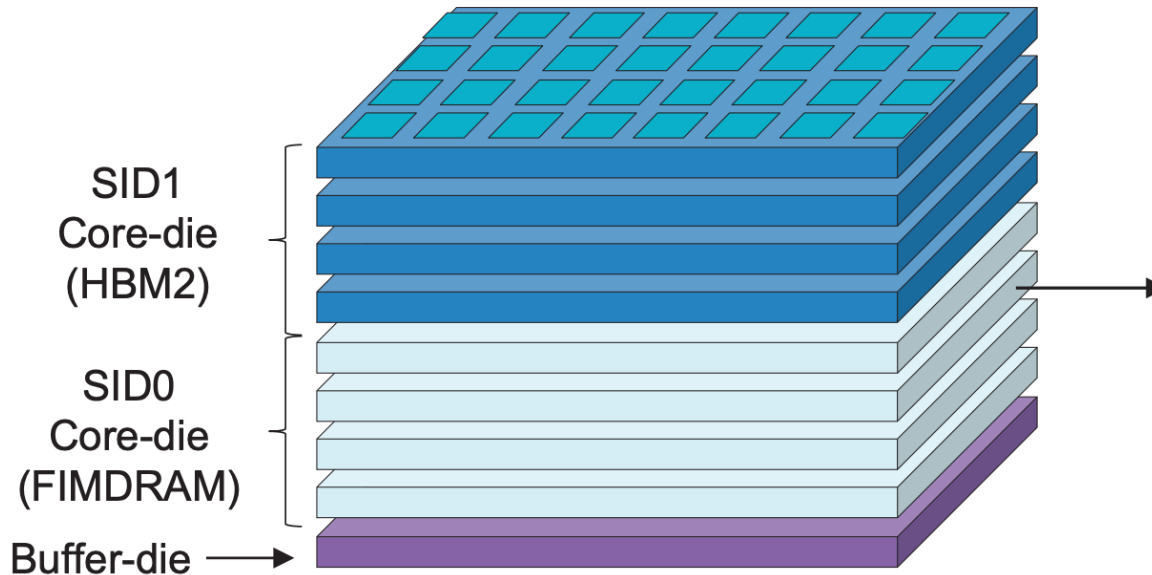
UPMEM Processing-in-DRAM Engine (2019)

- **Processing in DRAM Engine**
- Includes **standard DIMM modules**, with a **large number of DPU processors** combined with DRAM chips.
- Replaces **standard DIMMs**
 - DDR4 R-DIMM modules
 - 8GB+128 DPUs (16 PIM chips)
 - Standard 2x-nm DRAM process
 - **Large amounts of** compute & memory bandwidth



Samsung Function-in-Memory DRAM (2021)

■ FIMDRAM based on HBM2



[3D Chip Structure of HBM with FIMDRAM]

Chip Specification

128DQ / 8CH / 16 banks / BL4

32 PCU blocks (1 FIM block/2 banks)

1.2 TFLOPS (4H)

**FP16 ADD /
Multiply (MUL) /
Multiply-Accumulate (MAC) /
Multiply-and- Add (MAD)**

ISSCC 2021 / SESSION 25 / DRAM / 25.4

25.4 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

Young-Cheon Kwon¹, Suk Han Lee¹, Jaehoon Lee¹, Sang-Hyuk Kwon¹, Je Min Ryu¹, Jong-Pil Son¹, Seongil O¹, Hak-Soo Yu¹, Haesuk Lee¹, Soo Young Kim¹, Youngmin Cho¹, Jin Guk Kim¹, Jongyoon Choi¹, Hyun-Sung Shin¹, Jin Kim¹, BengSeng Phuah¹, HyoungMin Kim¹, Myeong Jun Song¹, Ahn Choi¹, Daeho Kim¹, SooYoung Kim¹, Eun-Bong Kim¹, David Wang², Shinhaeng Kang¹, Yuhwan Ro³, Seungwoo Seo³, JoonHo Song³, Jaeyoun Youn¹, Kyomin Sohn¹, Nam Sung Kim¹

¹Samsung Electronics, Hwaseong, Korea

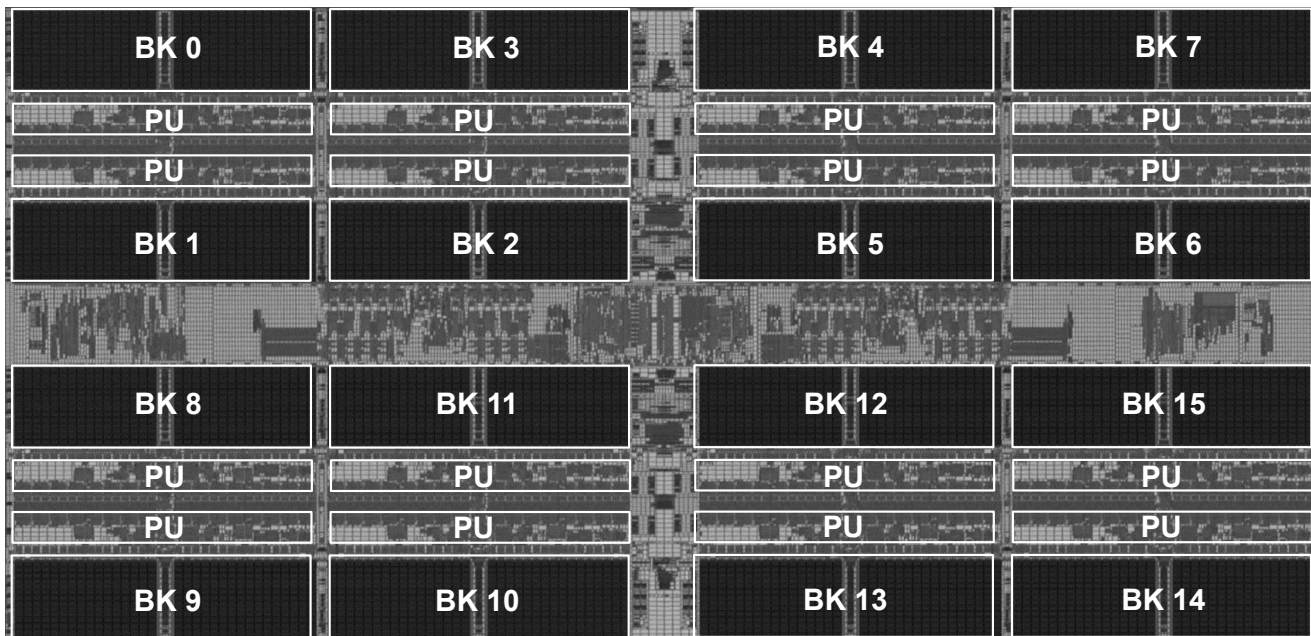
²Samsung Electronics, San Jose, CA

³Samsung Electronics, Suwon, Korea

AiM: Chip Implementation

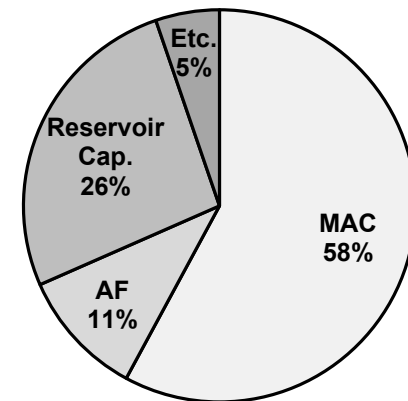
- 4 Gb AiM die with 16 processing units (PUs)

AiM Die Photograph



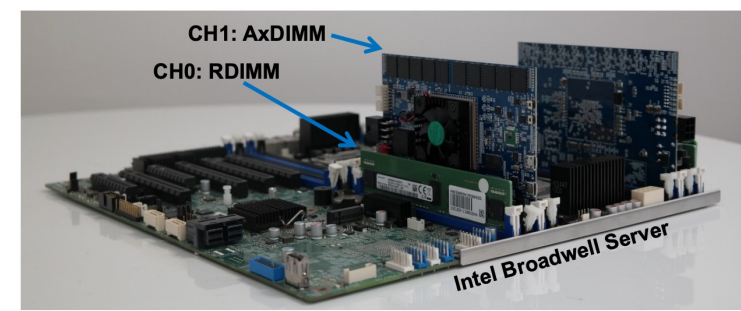
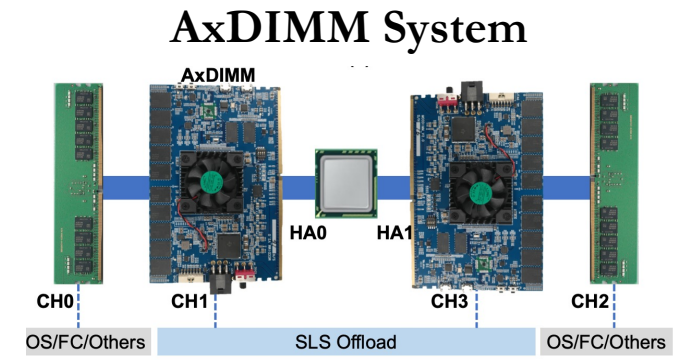
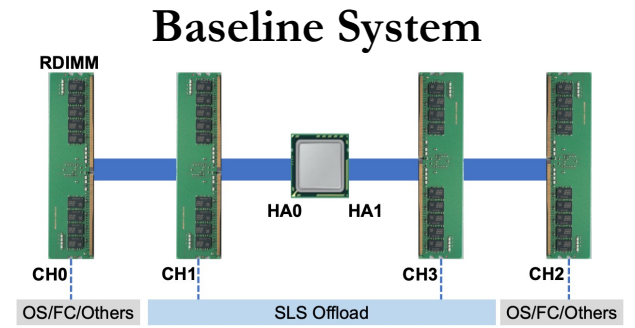
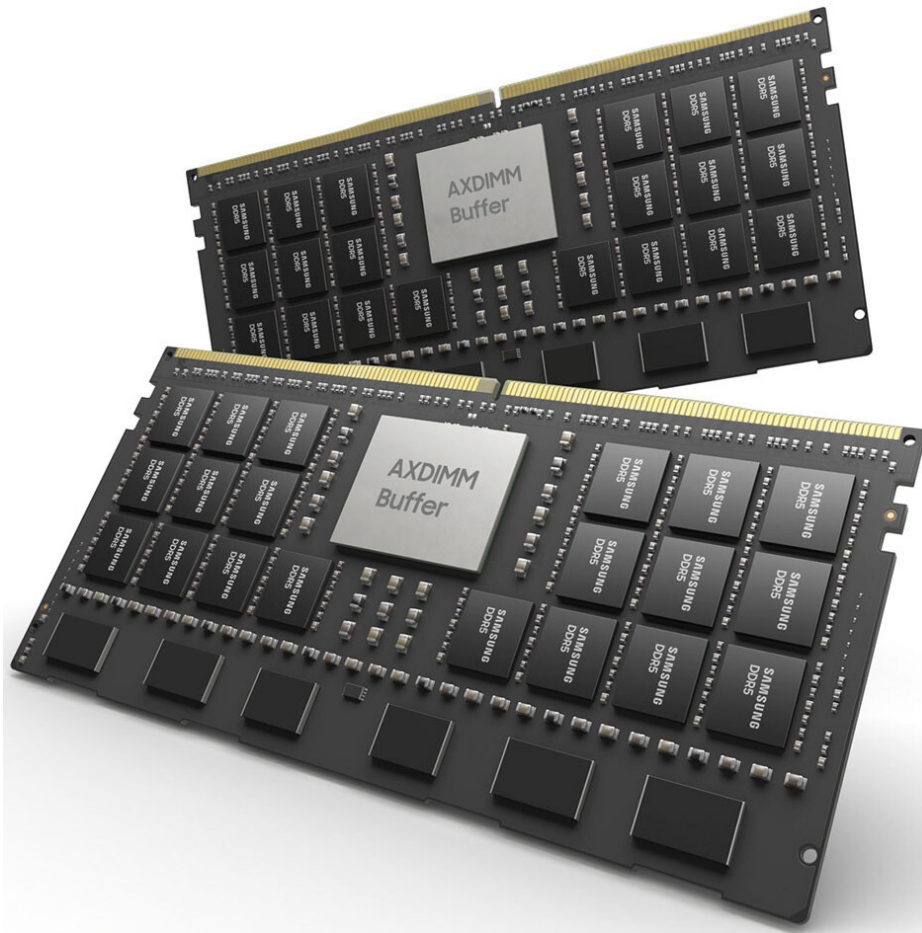
1 Process Unit (PU) Area

Total	0.19mm ²
MAC	0.11mm ²
Activation Function (AF)	0.02mm ²
Reservoir Cap.	0.05mm ²
Etc.	0.01mm ²



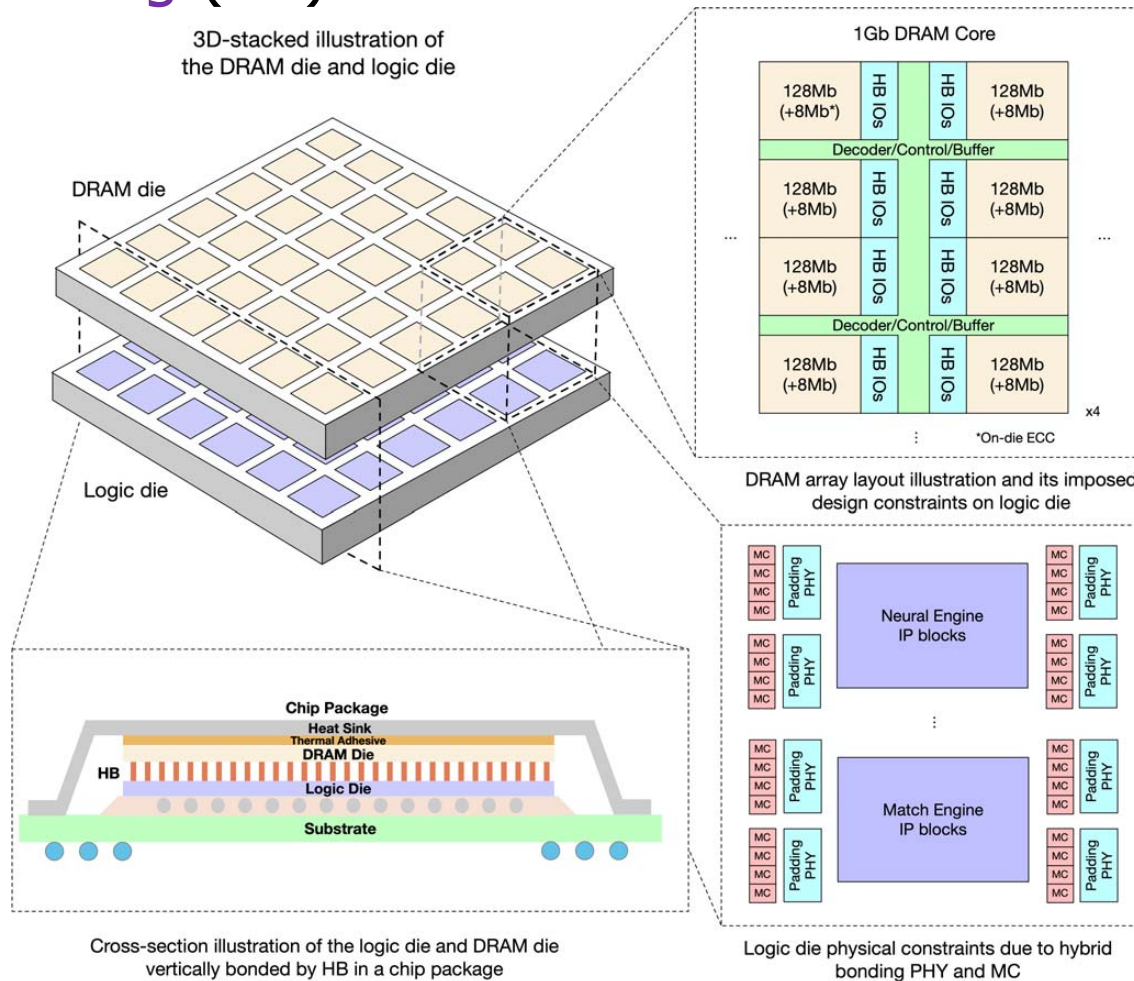
Samsung AxDIMM (2021)

- DIMM-based PIM
 - DLRM recommendation system



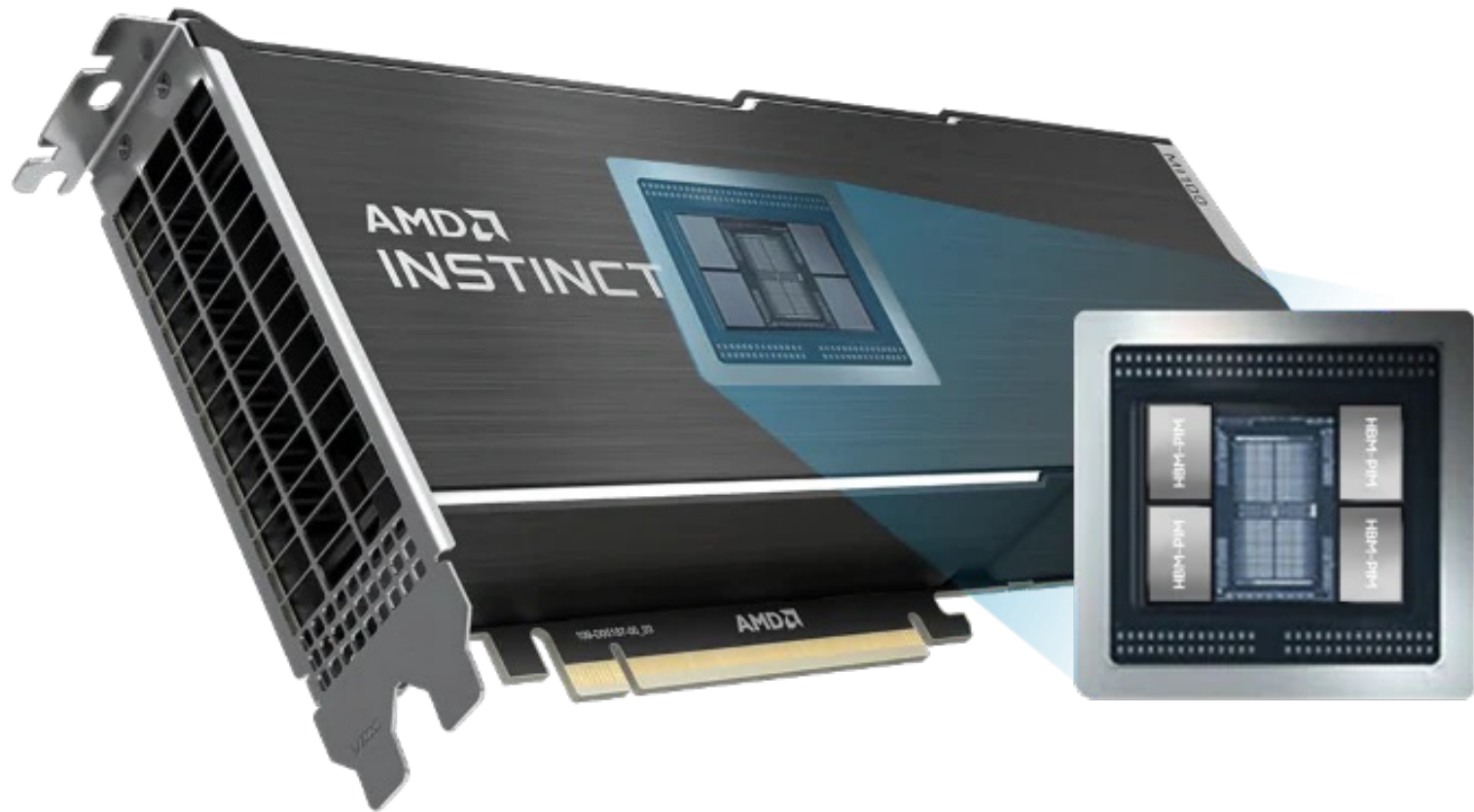
HB-PNM: Overall Architecture

- 3D-stacked logic die and DRAM die vertically bonded by hybrid bonding (HB)



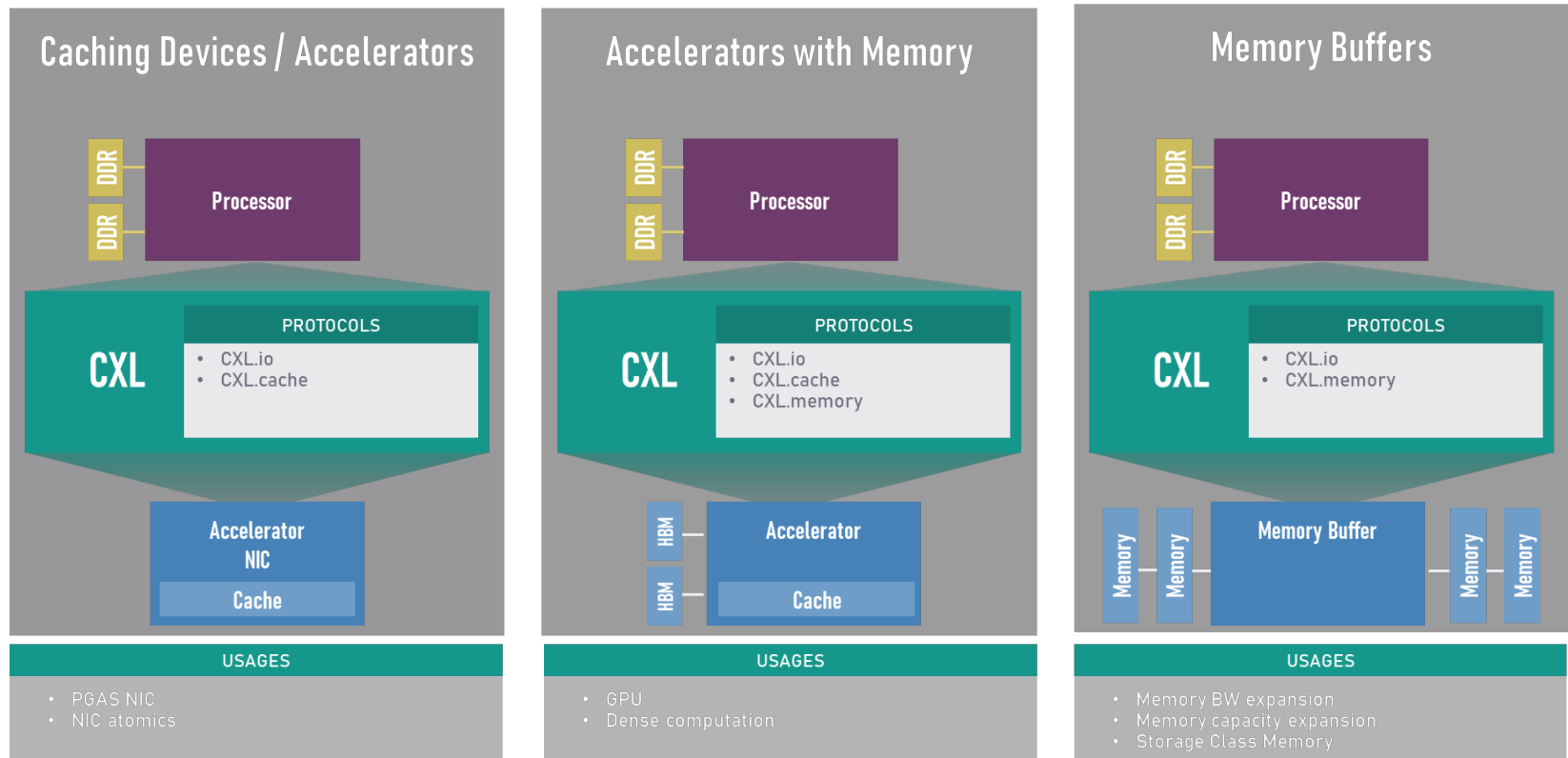
AMD GPU with HBM-PIM

- AMD Instinct Mi100 + HBM-PIM



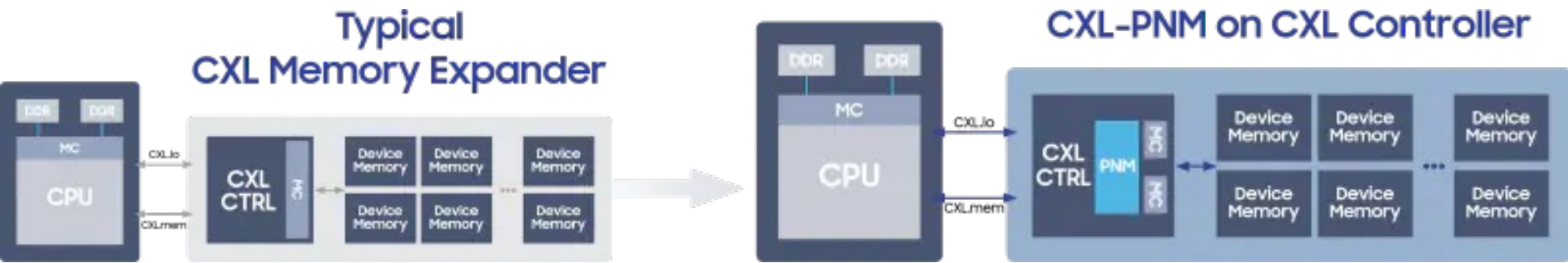
Compute Express Link (CXL)

- Compute Express Link (CXL) is an open industry standard interconnect offering **high-bandwidth, low-latency connectivity between host processor and devices** such as accelerators, memory buffers, and smart I/O devices



CXL and PIM

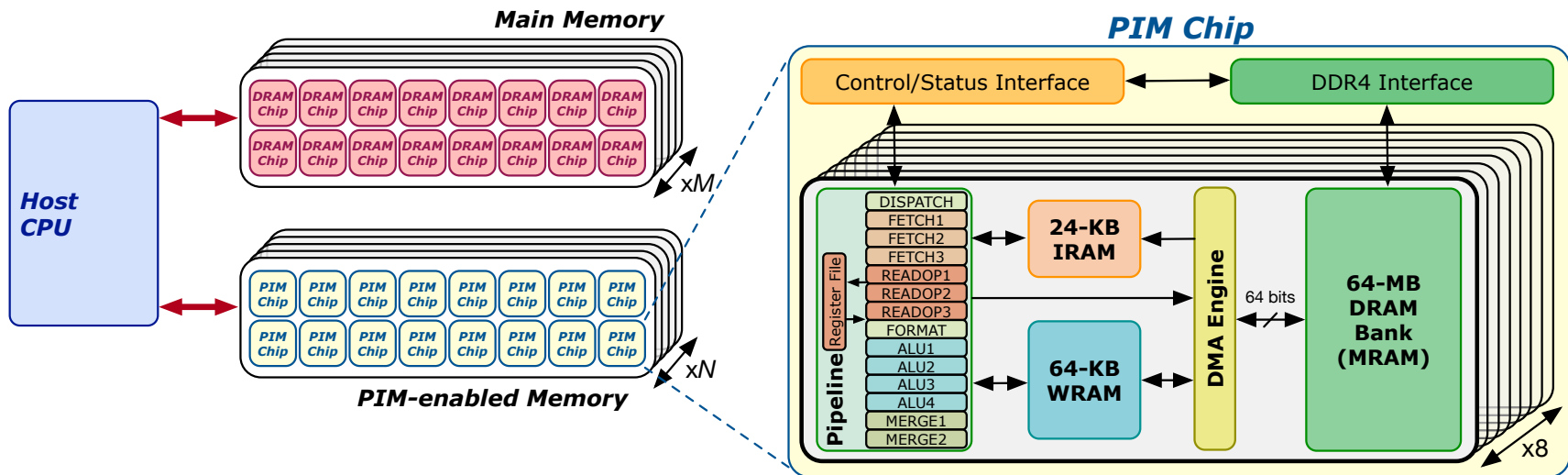
- CXL, as a high-bandwidth and low-latency interconnect, is a perfect complement for near-data processing



Programming Models and Code Generation for PIM

UPMEM System Organization

- A UPMEM DIMM contains 8 or 16 chips
 - Thus, 1 or 2 ranks of 8 chips each
- Inside each PIM chip there are:
 - 8 64MB banks per chip: Main RAM (MRAM) banks
 - 8 DRAM Processing Units (DPUs) in each chip, 64 DPUs per rank



Accelerator Model (I)

- UPMEM DIMMs coexist with conventional DIMMs
- Integration of UPMEM DIMMs in a system follows an **accelerator model**
- UPMEM DIMMs can be seen as a **loosely coupled accelerator**
 - Explicit data movement between the main processor (host CPU) and the accelerator (UPMEM)
 - Explicit kernel launch onto the UPMEM processors
- This resembles GPU computing

Accelerator Model (II)

- FIG. 6 is a flow diagram representing operations in a method of delegating a processing task to a DRAM processor according to an example embodiment

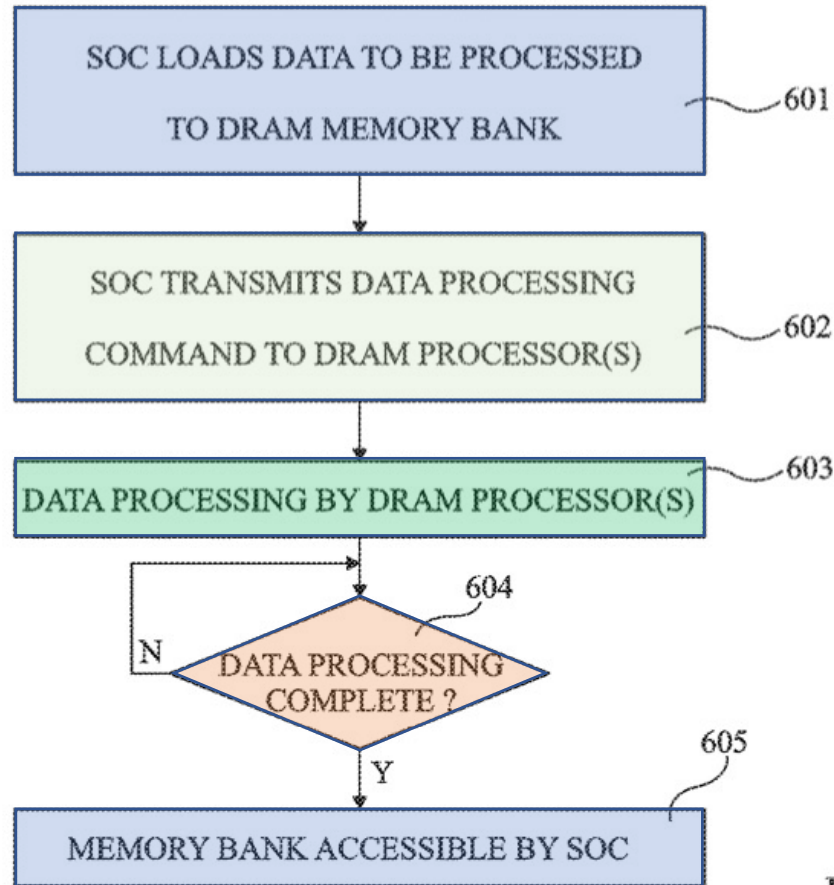
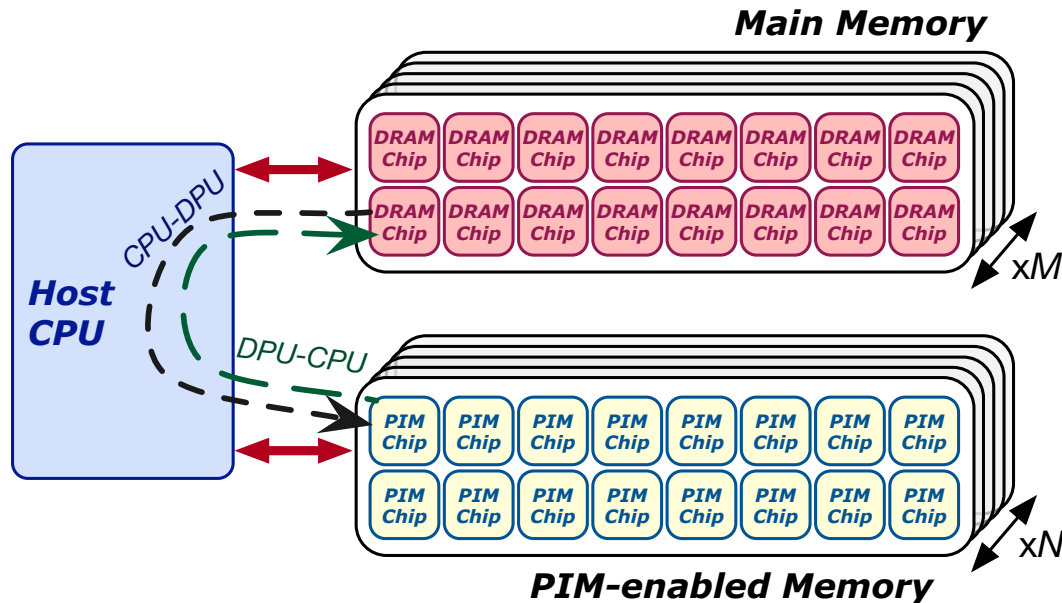


Fig 6

Inter-DPU Communication

- There is **no direct communication channel between DPUs**

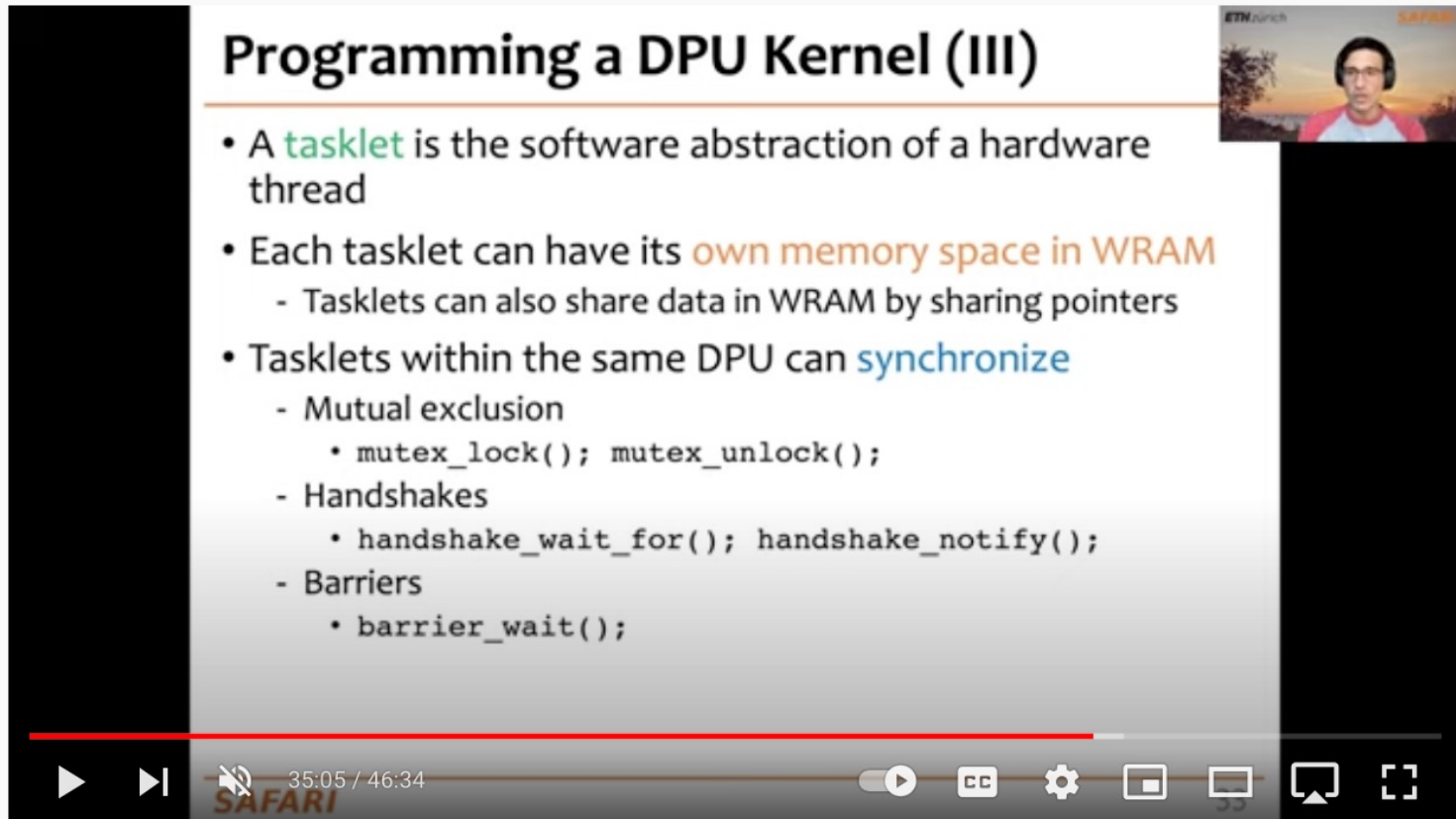


- Inter-DPU communication takes place via the host CPU using CPU-DPU and DPU-CPU transfers
- Example communication patterns:
 - Merging of partial results to obtain the final result
 - Only DPU-CPU transfers
 - Redistribution of intermediate results for further computation
 - DPU-CPU transfers and CPU-DPU transfers

Lecture on Programming UPMEM PIM

Programming a DPU Kernel (III)

- A **tasklet** is the software abstraction of a hardware thread
- Each tasklet can have its **own memory space in WRAM**
 - Tasklets can also share data in WRAM by sharing pointers
- Tasklets within the same DPU can **synchronize**
 - Mutual exclusion
 - `mutex_lock(); mutex_unlock();`
 - Handshakes
 - `handshake_wait_for(); handshake_notify();`
 - Barriers
 - `barrier_wait();`



Processing-in-Memory Course: Lecture 7: Programming PIM Architectures - Spring 2022

420 views · Premiered Apr 21, 2022

 15  DISLIKE  SHARE  CLIP  SAVE ...



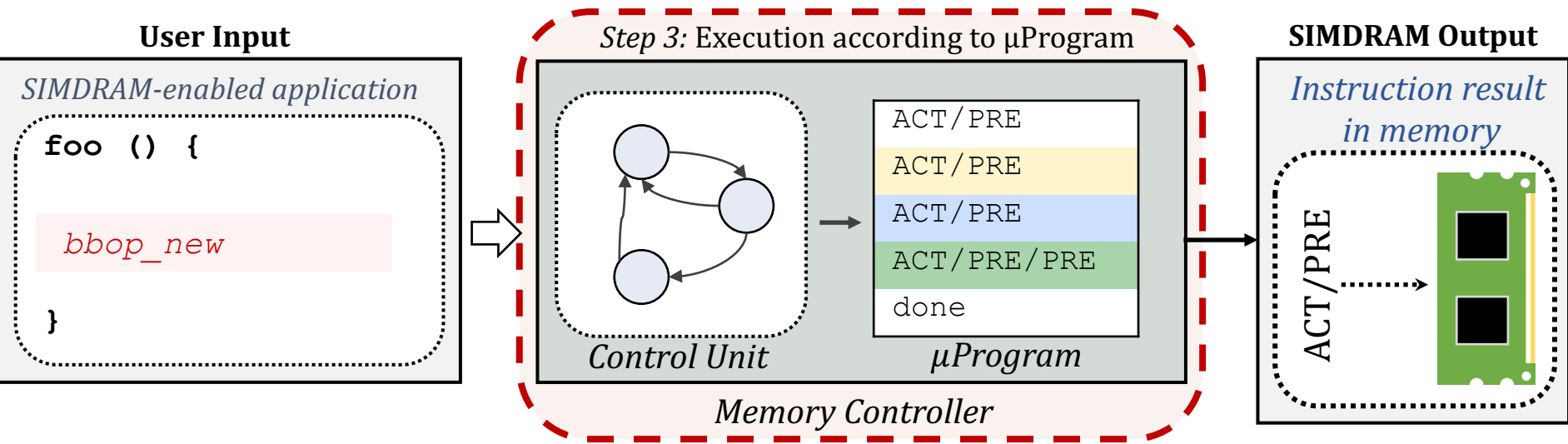
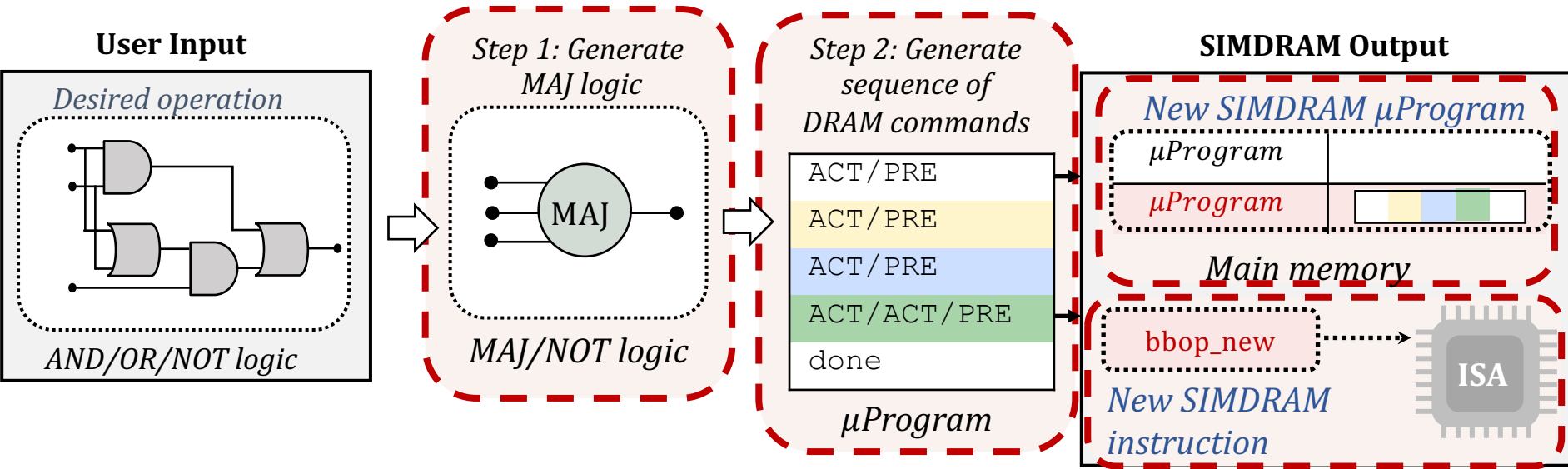
Onur Mutlu Lectures

25.9K subscribers

SUBSCRIBED



SIMDRAM Framework



Programming Interface

- Four new SIMD/DRAM ISA extensions

Type	ISA Format
Initialization	<code>bbop_trsp_init address, size, n</code>
1-Input Operation	<code>bbop_op dst, src, size, n</code>
2-Input Operation	<code>bbop_op dst, src_1, src_2, size, n</code>
Predication	<code>bbop_if_else dst, src_1, src_2, select, size, n</code>

Code Using SIMD RAM Instructions

```
1 int size = 65536;
2 int elm_size = sizeof (uint8_t);
3 uint8_t *A , *B , *C = (uint8_t *) malloc(size * elm_size);
4 uint8_t *pred = (uint8_t *) malloc(size * elm_size);
5 ...
6 for (int i = 0; i < size ; ++ i){
7     bool cond = A[i] > pred[i];
8     if (cond)
9         C [i] = A[i] + B[i];
10    else
11        C [i] = A[i] - B [i];
12 }
```

← C code for vector add/sub
with predicated execution

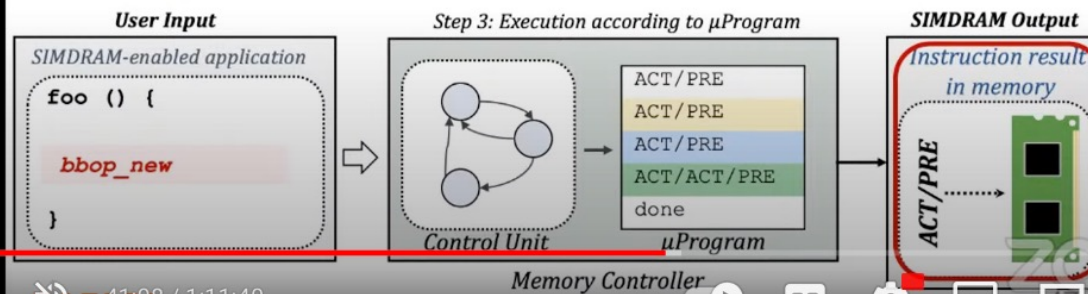
Equivalent code using
SIMDRAM operations →

```
1 int size = 65536;
2 int elm_size = sizeof(uint8_t);
3 uint8_t *A , *B , *C = (uint8_t *) malloc(size * elm_size);
4
5 bbop_trsp_init(A , size , elm_size);
6 bbop_trsp_init(B , size , elm_size);
7 bbop_trsp_init(C , size , elm_size);
8 uint8_t *pred = (uint8_t *) malloc(size * elm_size);
9 // D, E, F store intermediate data
10 uint8_t *D , *E = (uint8_t *) malloc (size * elm_size);
11 bool *F = (bool *) malloc (size * sizeof(bool));
12 ...
13 bbop_add(D , A , B , size , elm_size);
14 bbop_sub(E , A , B , size , elm_size);
15 bbop_greater(F , A , pred , size , elm_size);
16 bbop_if_else(C , D , E , F , size , elm_size);
```

Lecture on SIMD RAM

Step 3: μ Program Execution

- **SIMDRAM control unit:** handles the execution of the μ Program at runtime
- Upon receiving a **bbop instruction**, the control unit:
 1. Loads the μ Program corresponding to SIMD RAM operation
 2. Issues the sequence of DRAM commands (ACT/PRE) stored in the μ Program to SIMD RAM subarrays to perform the in-DRAM operation



Processing-in-Memory Course: Lecture 13: Bit-Serial SIMD Processing using DRAM - Spring 2022

512 views • Streamed live on Jun 2, 2022

33 DISLIKE SHARE CLIP SAVE ...



Onur Mutlu Lectures

25.9K subscribers

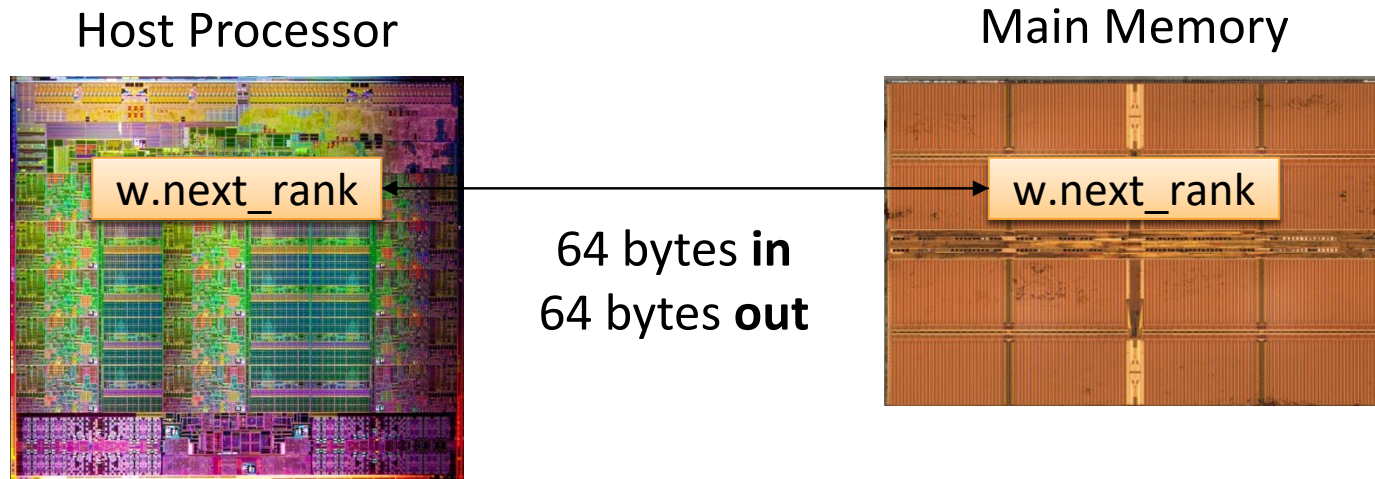
SUBSCRIBED



PIM Runtime: Scheduling and Data Mapping

Simple PIM Operations as ISA Extensions (I)

```
for (v: graph.vertices) { PageRank algorithm (Page et al. 1999)  
  value = weight * v.rank;  
  for (w: v.successors) {  
    w.next_rank += value;  
  }  
}
```



Conventional Architecture

Simple PIM Operations as ISA Extensions (II)

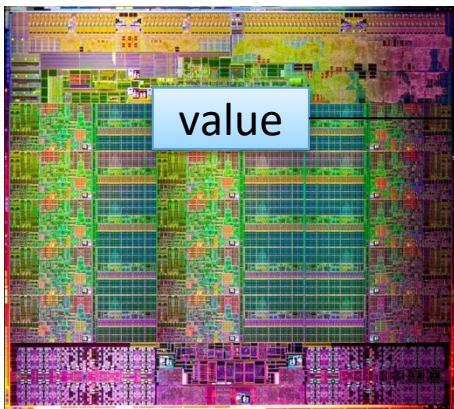
```
for (v: graph.vertices) { PageRank algorithm (Page et al. 1999)  
  value = weight * v.rank;  
  for (w: v.successors) {  
    __pim_add(&w.next_rank, value);  
  }  
}
```

pim.add r1, (r2)

__pim_add(&w.next_rank, value);

Host Processor

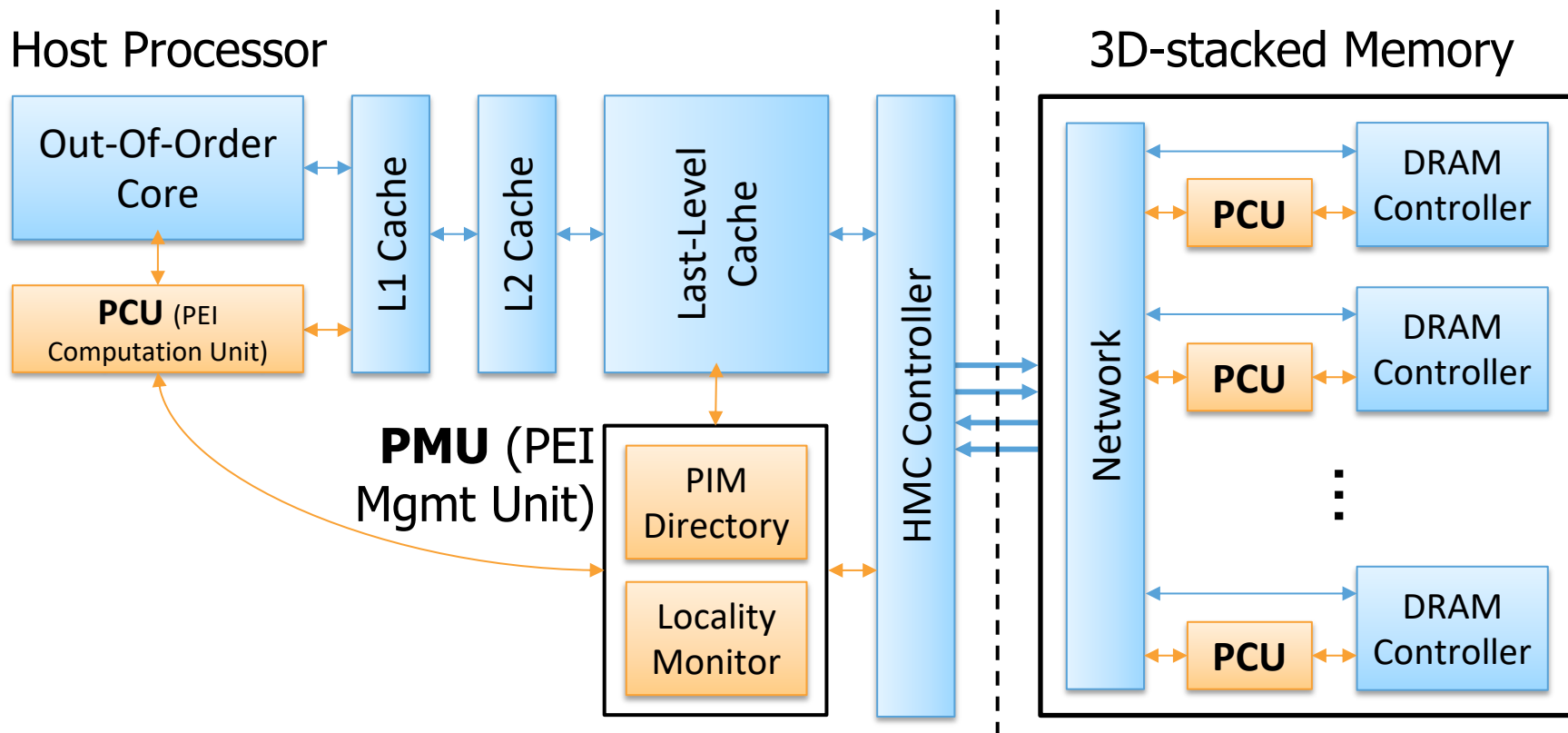
Main Memory



8 bytes in
0 bytes out

In-Memory Addition

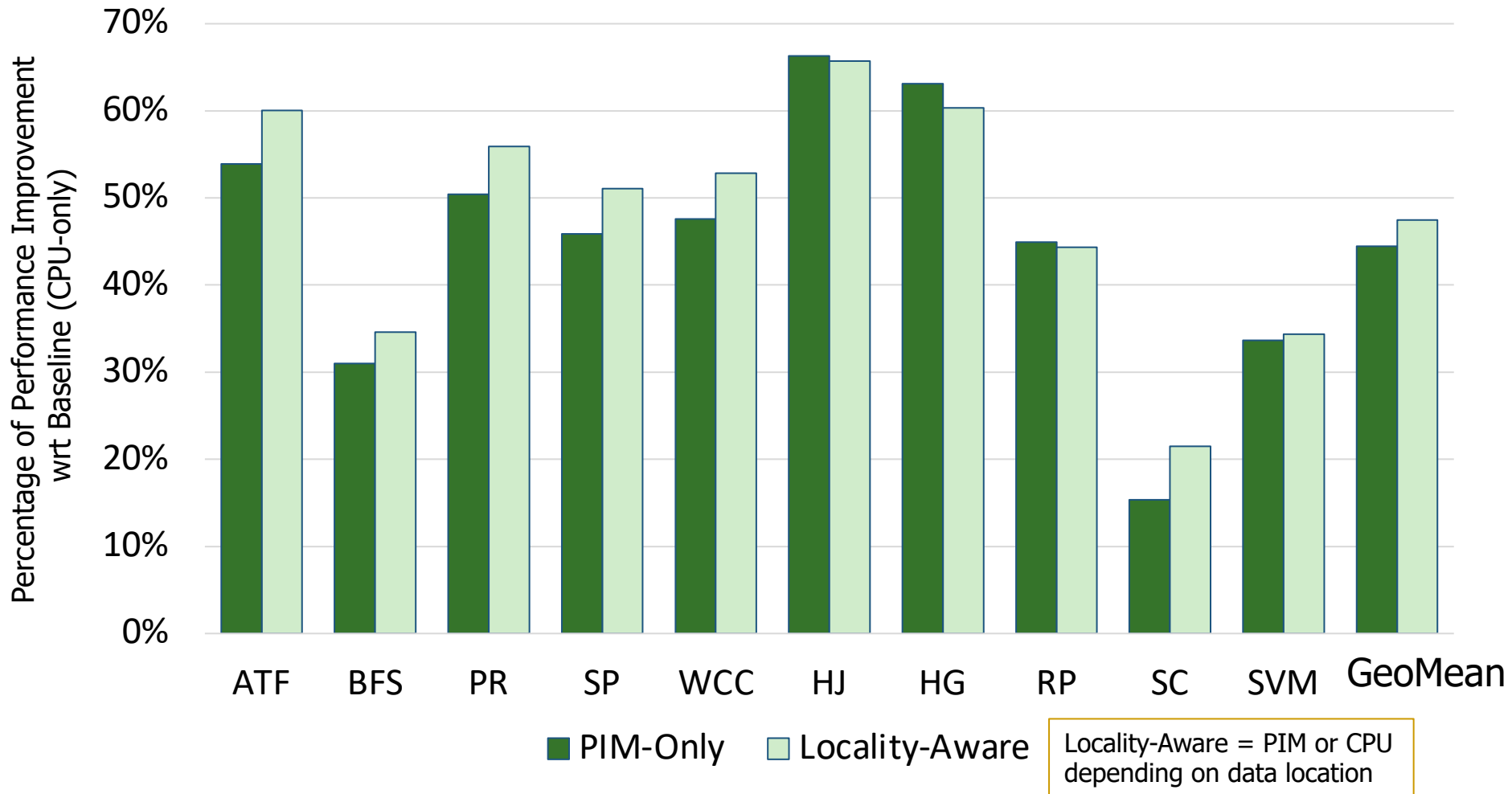
Example PEI Microarchitecture



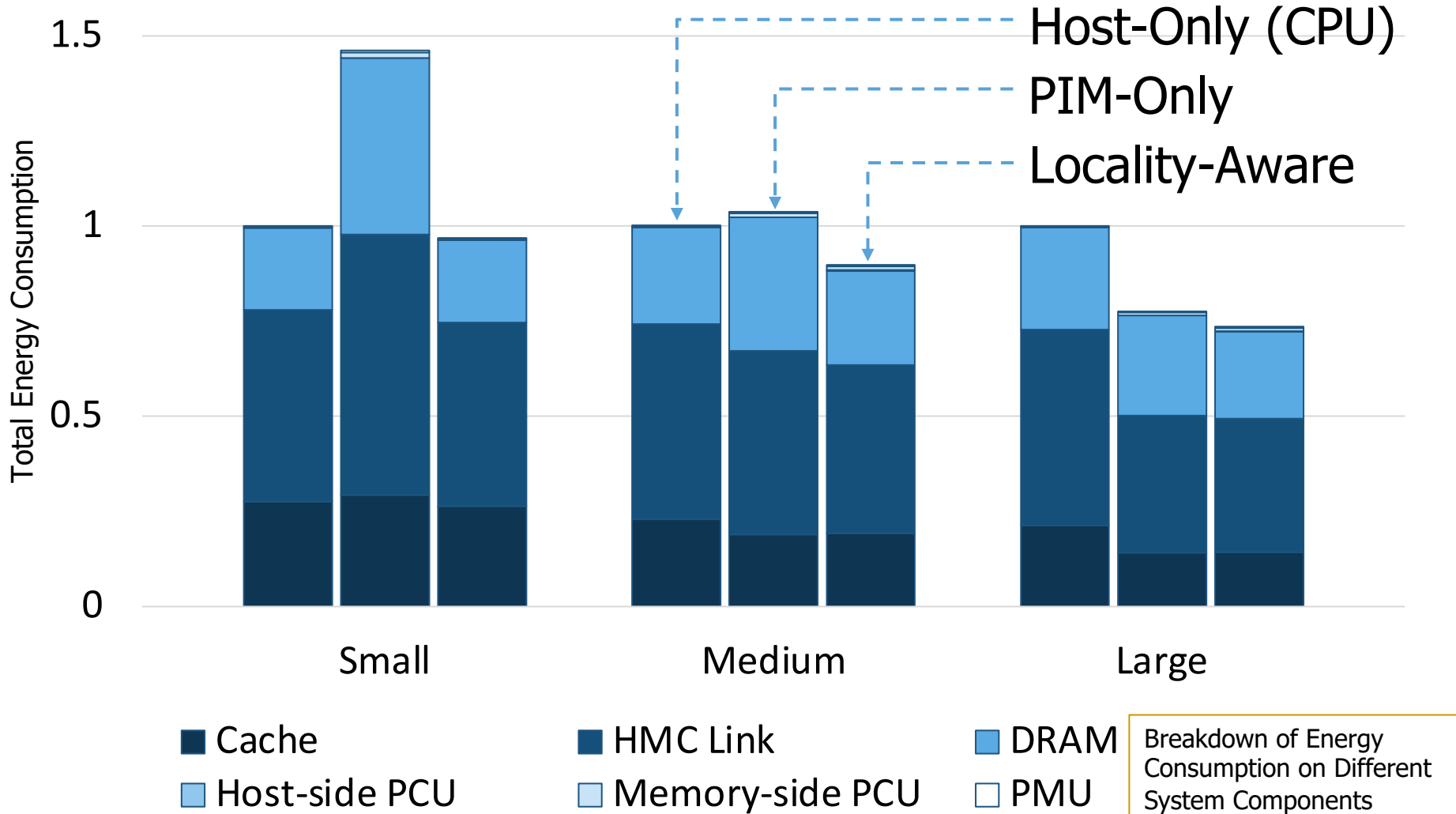
Example PEI uArchitecture

PEI Performance Delta: Large Data Sets

(Large Inputs, Baseline: CPU-Only)



PEI Energy Consumption



More on 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

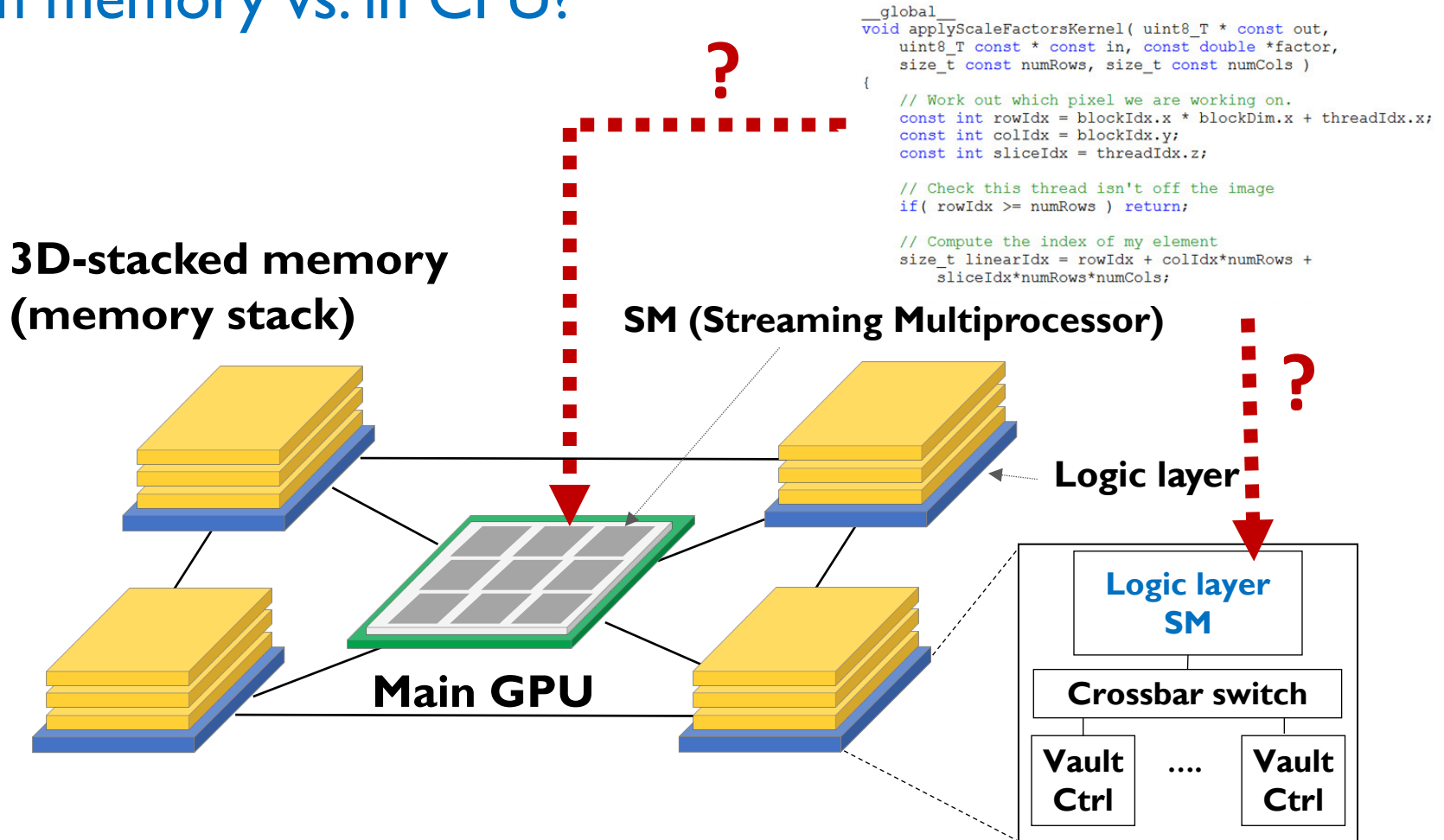
junwhan@snu.ac.kr, sungjoo.yoo@gmail.com, onur@cmu.edu, kchoi@snu.ac.kr

Seoul National University

[†]Carnegie Mellon University

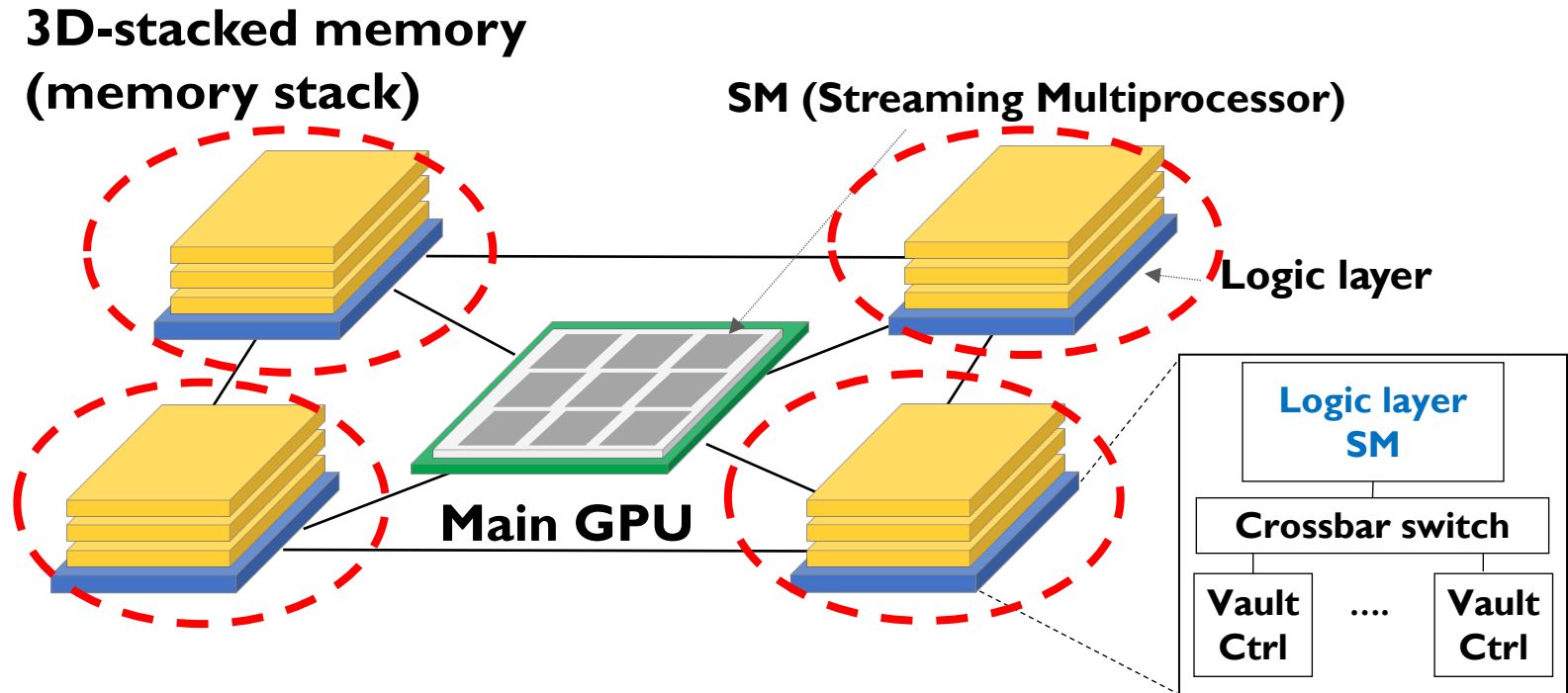
Key Challenge 1: Code Mapping

- **Challenge 1: Which operations should be executed in memory vs. in CPU?**



Key Challenge 2: Data Mapping

- **Challenge 2:** How should data be mapped to different 3D memory stacks?



How to Do the 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

How to Schedule Code? (I)

- 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

How to Schedule Code? (II)

- 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

How to Schedule Code? (III)

- 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*

Research Questions

What are simple **mechanisms to enable and disable PIM execution**?
How can PIM execution be throttled for highest performance gains?
How should data locations and access patterns affect where/whether PIM execution should occur?

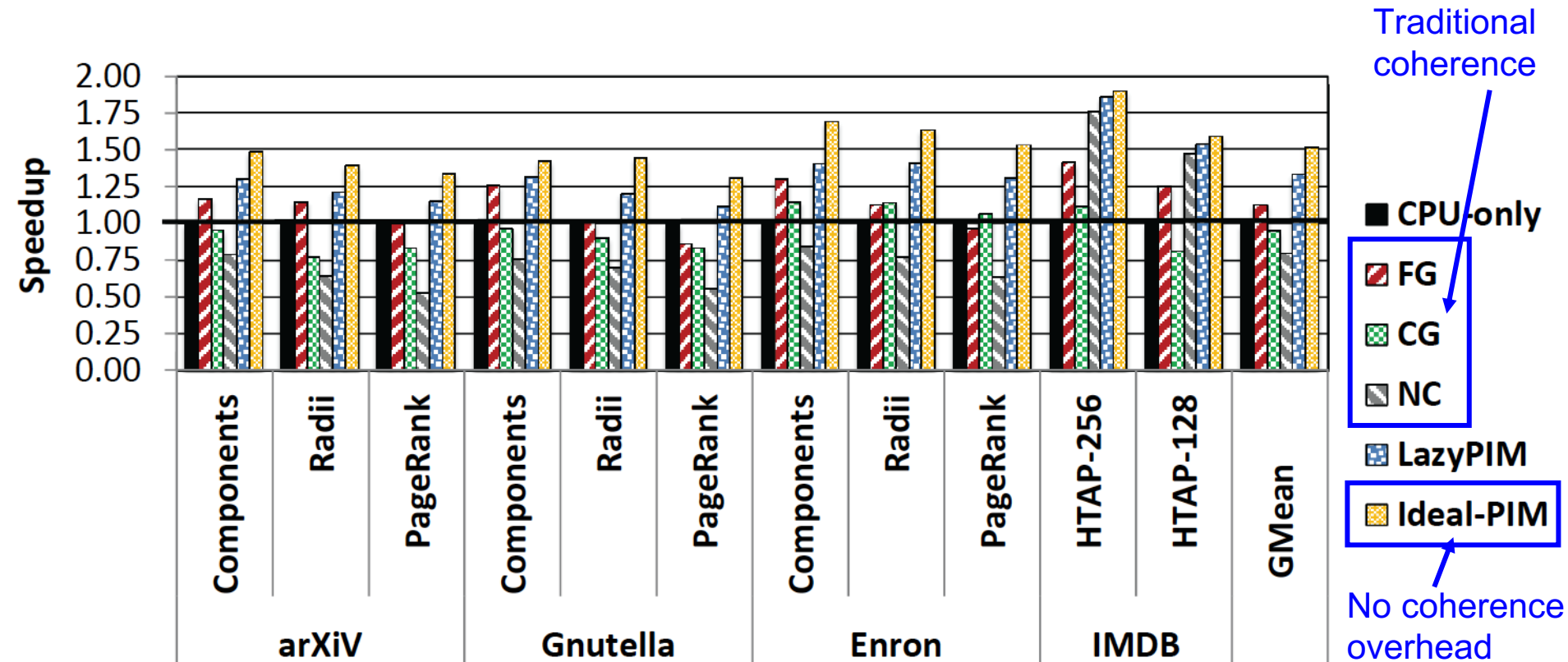
Which parts of a given application's code should be executed on PIM? What are simple **mechanisms to identify when those parts of the application code can benefit from PIM**?

What are scheduling mechanisms to **share PIM engines between multiple requesting cores** to maximize benefits obtained from PIM?

What are simple **mechanisms to manage access to a memory that serves both CPU requests and PIM requests**?

Memory Coherence

Challenge: Coherence for Hybrid CPU-PIM Apps



How to Maintain Coherence? (I)

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

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

Amirali Boroumand[†], Saugata Ghose[†], Minesh Patel[†], Hasan Hassan^{†§}, Brandon Lucia[†],
Kevin Hsieh[†], Krishna T. Malladi^{*}, Hongzhong Zheng^{*}, and Onur Mutlu^{‡†}

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

How to Maintain Coherence? (II)

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

CoNDA: Efficient Cache Coherence Support for Near-Data Accelerators

Amirali Boroumand[†]

Saugata Ghose[†]

Minesh Patel^{*}

Hasan Hassan^{*}

Brandon Lucia[†]

Rachata Ausavarungnirun^{†‡}

Kevin Hsieh[†]

Nastaran Hajinazar^{◇†}

Krishna T. Malladi[§]

Hongzhong Zheng[§]

Onur Mutlu^{*†}

[†]Carnegie Mellon University

^{*}ETH Zürich

[‡]KMUTNB

[◇]Simon Fraser University

[§]Samsung Semiconductor, Inc.

CoNDA: Efficient Cache Coherence Support for Near-Data Accelerators

Amirali Boroumand

Saugata Ghose, Minesh Patel, Hasan Hassan,
Brandon Lucia, Rachata Ausavarungnirun, Kevin Hsieh,
Nastaran Hajinazar, Krishna Malladi, Hongzhong Zheng,
Onur Mutlu

SAFARI



Carnegie Mellon



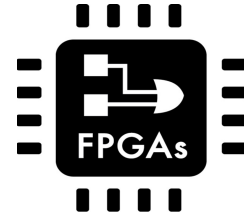
ETH zürich

Specialized Accelerators

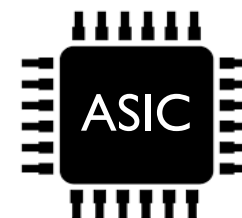
Specialized accelerators are now everywhere!



GPU

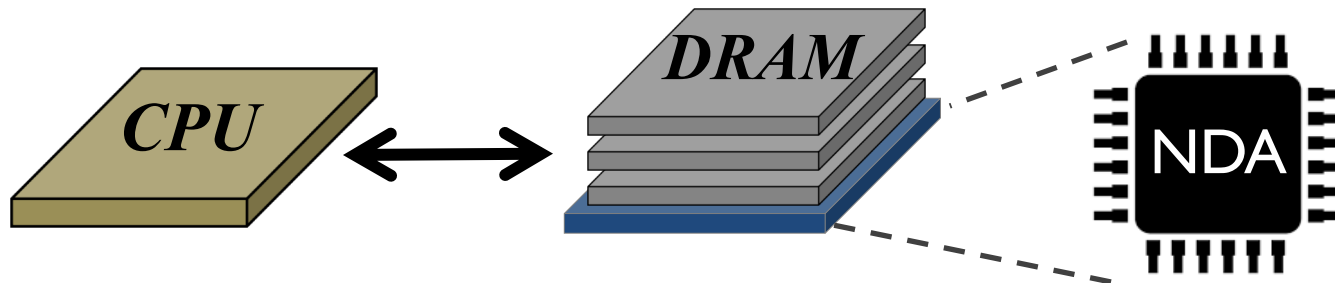


FPGA



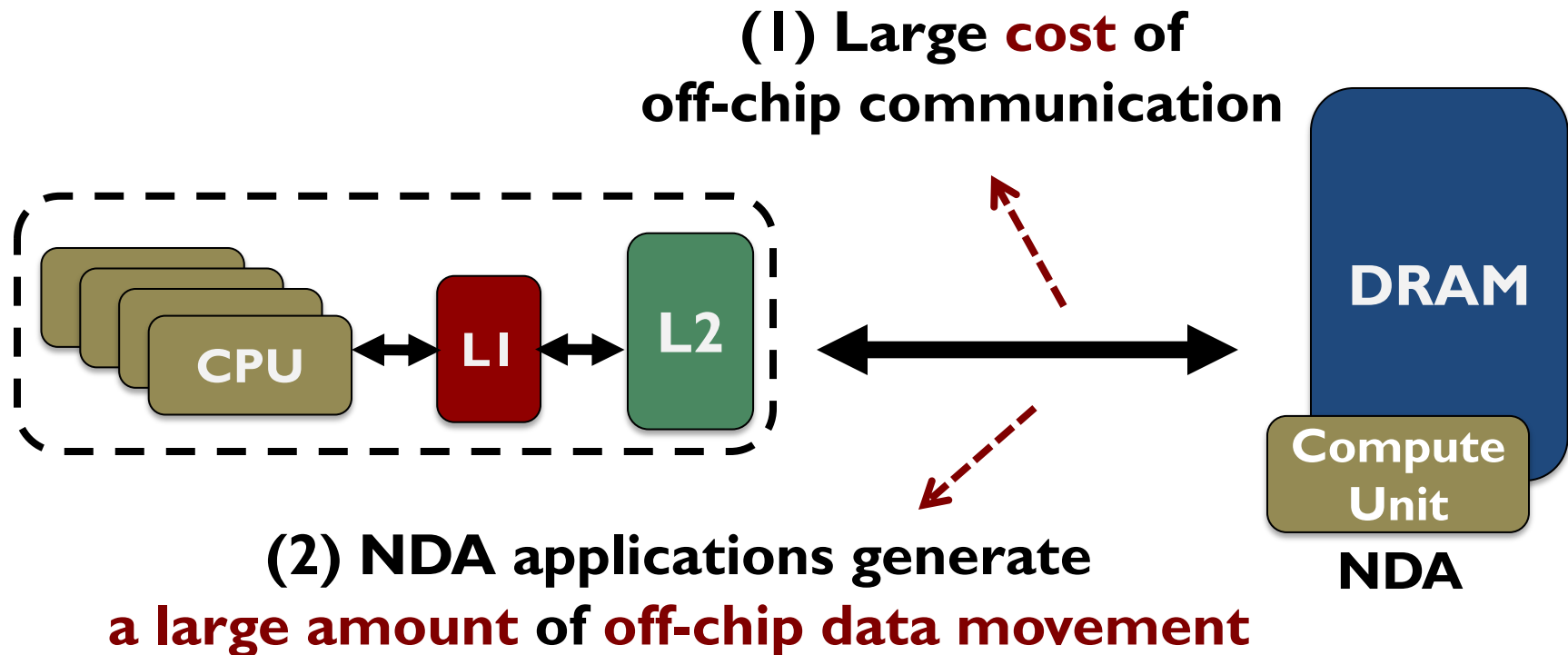
ASIC

Recent advancement in 3D-stacked technology enabled **Near-Data Accelerators (NDA)**



Coherence For NDAs

Challenge: Coherence between NDAs and CPUs



It is **impractical** to use traditional coherence protocols

Existing Coherence Mechanisms

We extensively study existing **NDA coherence mechanisms** and make **three key observations**:

1

These mechanisms **eliminate** a significant portion of **NDA's benefits**

2

The **majority of off-chip coherence traffic** generated by these mechanisms is **unnecessary**

3

Much of the **off-chip traffic** can be eliminated if the coherence mechanism has **insight** into the **memory accesses**

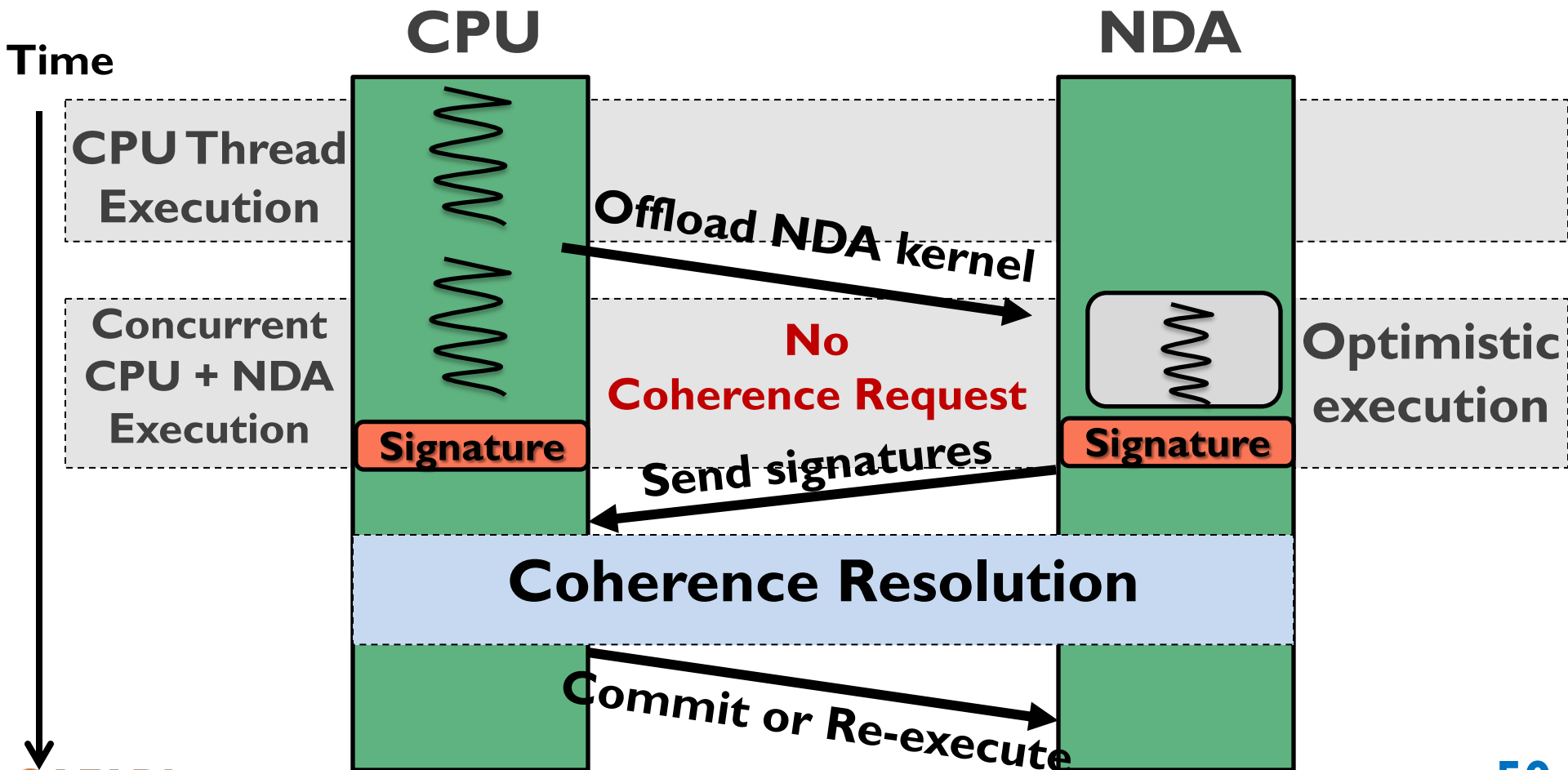
An Optimistic Approach

We find that an optimistic approach to coherence can address the challenges related to NDA coherence

- 1 Gain insights *before* any coherence checks happens
- 2 Perform *only the necessary* coherence requests

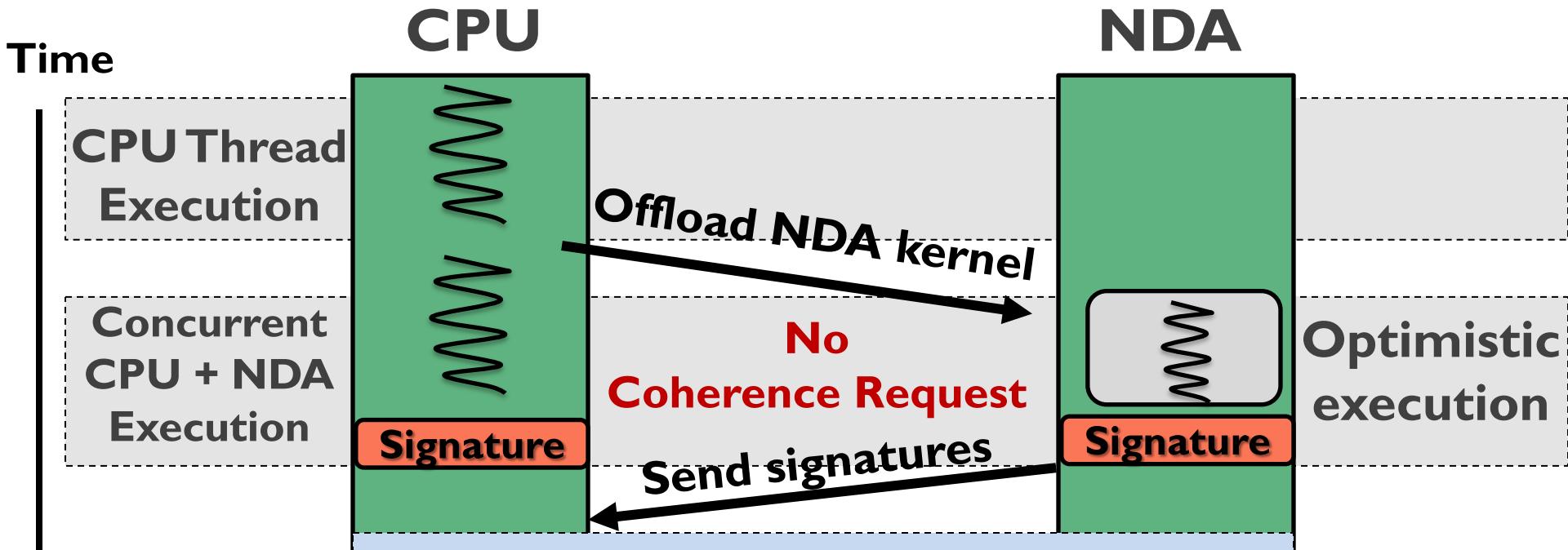
CoNDA

We propose **CoNDA**, a mechanism that uses **optimistic NDA execution** to avoid **unnecessary coherence traffic**



CoNDA

We propose **CoNDA**, a mechanism that uses **optimistic NDA execution** to avoid **unnecessary coherence traffic**



CoNDA comes within 10.4% and 4.4% of performance and energy of an ideal NDA coherence mechanism

CoNDA: Efficient Cache Coherence Support for Near-Data Accelerators

Amirali Boroumand

Saugata Ghose, Minesh Patel, Hasan Hassan,
Brandon Lucia, Rachata Ausavarungnirun, Kevin Hsieh,
Nastaran Hajinazar, Krishna Malladi, Hongzhong Zheng,
Onur Mutlu

SAFARI



Carnegie Mellon



ETH zürich

How to Maintain Coherence? (II)

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

CoNDA: Efficient Cache Coherence Support for Near-Data Accelerators

Amirali Boroumand[†]

Saugata Ghose[†]

Minesh Patel^{*}

Hasan Hassan^{*}

Brandon Lucia[†]

Rachata Ausavarungnirun^{†‡}

Kevin Hsieh[†]

Nastaran Hajinazar^{◇†}

Krishna T. Malladi[§]

Hongzhong Zheng[§]

Onur Mutlu^{*†}

[†]Carnegie Mellon University

^{*}ETH Zürich

[‡]KMUTNB

[◇]Simon Fraser University

[§]Samsung Semiconductor, Inc.

Synchronization Support

How to Support Synchronization?

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

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

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

SynCron

Efficient Synchronization Support for Near-Data-Processing Architectures



Christina Giannoula

Nandita Vijaykumar, Nikela Papadopoulou, Vasileios Karakostas
Ivan Fernandez, Juan Gómez Luna, Lois Orosa
Nectarios Koziris, Georgios Goumas, Onur Mutlu

SAFARI



ETH zürich



Executive Summary

Problem:

Synchronization support is **challenging** for NDP systems

Prior schemes are **not suitable** or **efficient** for NDP systems

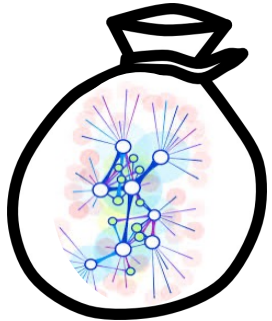
Contribution:

SynCron: the **first end-to-end** synchronization solution for NDP architectures

Key Results:

SynCron comes within **9.5%** and **6.2%** of performance and energy of an **Ideal** zero-overhead synchronization scheme

Synchronization is Necessary



Graph Analytics



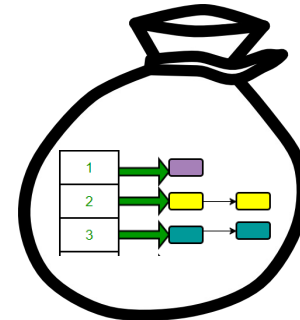
Bioinformatics



Databases



Image Processing



Concurrent Data Structures

Single Source Shortest Path (SSSP)

```
for v in Graph:
  for u in neighbors[v]:
    if distance[v] + edge_weight[v, u] < distance[u]
      lock_acquire(u)
      if distance[v] + edge_weight[v, u] < distance[u]
        distance[u] = distance[v] + edge_weight[v, u]
      lock_release(u)
```

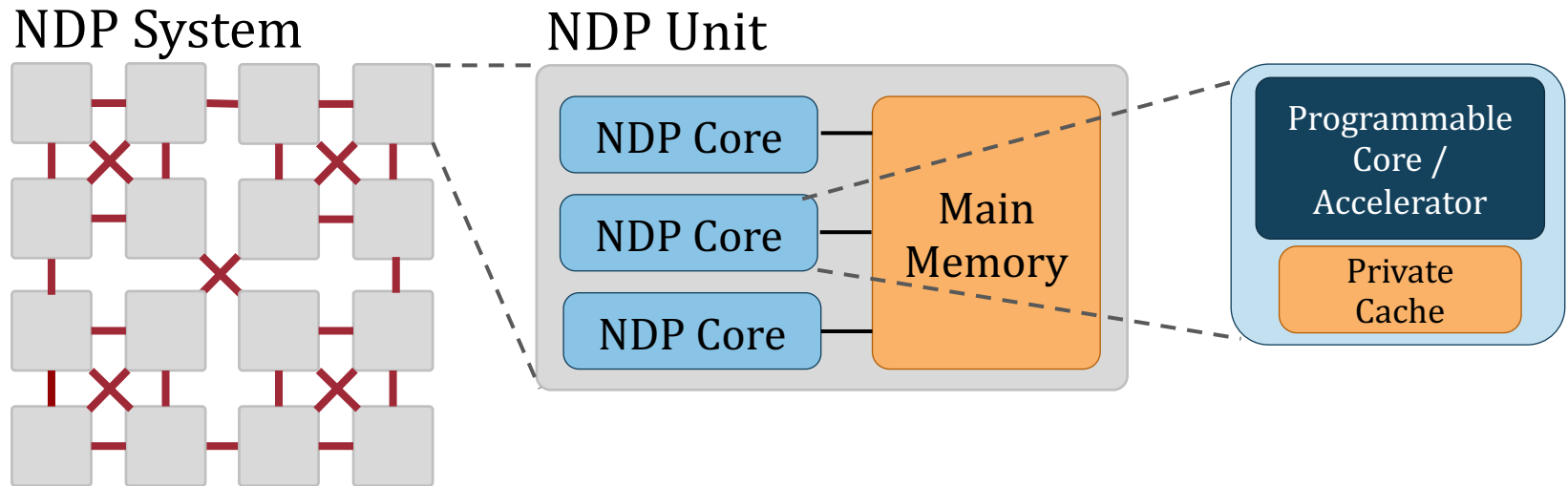
Locks



Barriers



Baseline NDP Architecture



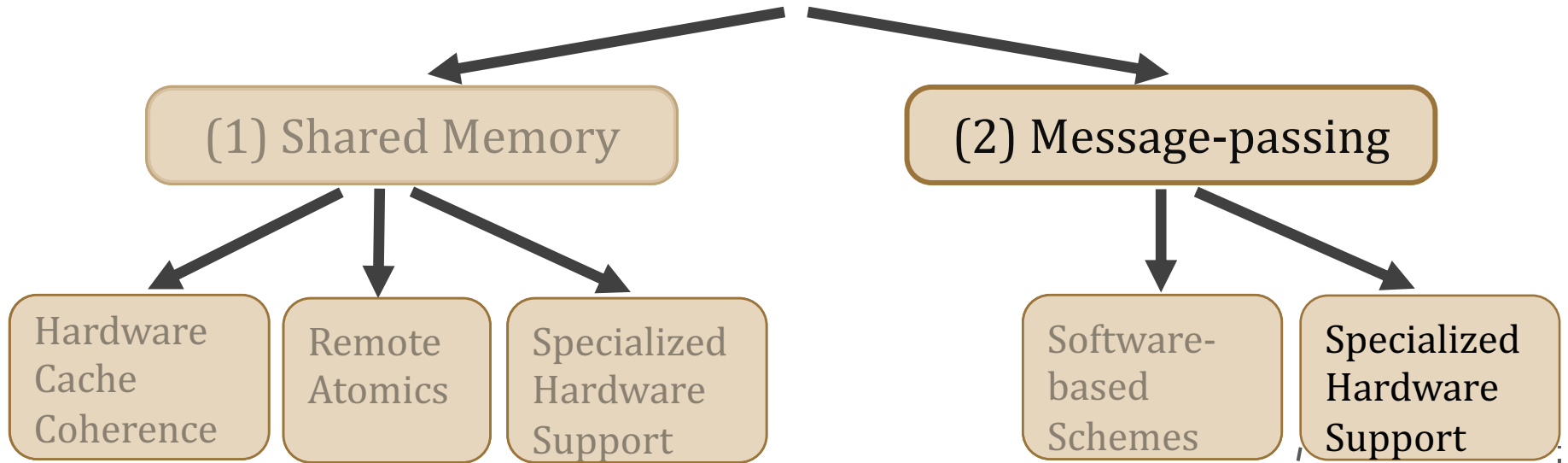
Synchronization **challenges** in NDP systems:

(1) Lack of hardware cache coherence support

(2) Expensive communication across NDP units

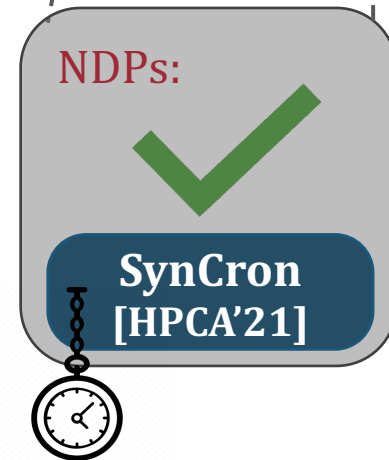
(3) Lack of a shared level of cache memory

NDP Synchronization Solution Space



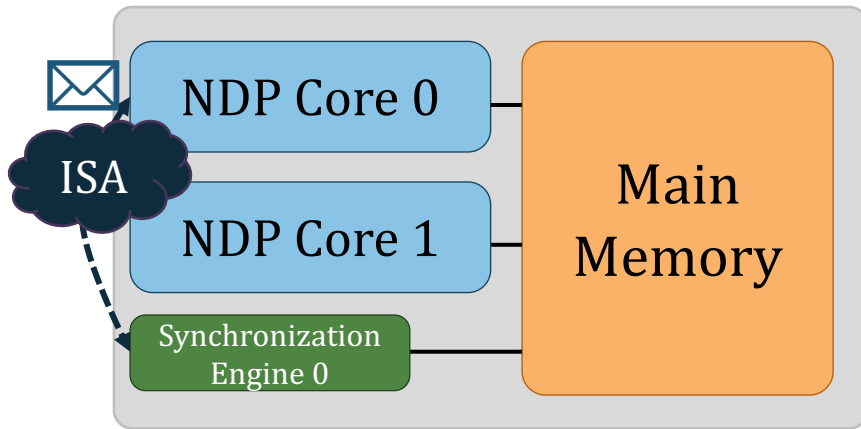
SynCron's Key Techniques:

1. **Hardware support** for synchronization acceleration
2. **Direct buffering** of synchronization variables
3. **Hierarchical** message-passing **communication**
4. Integrated hardware-only **overflow management**

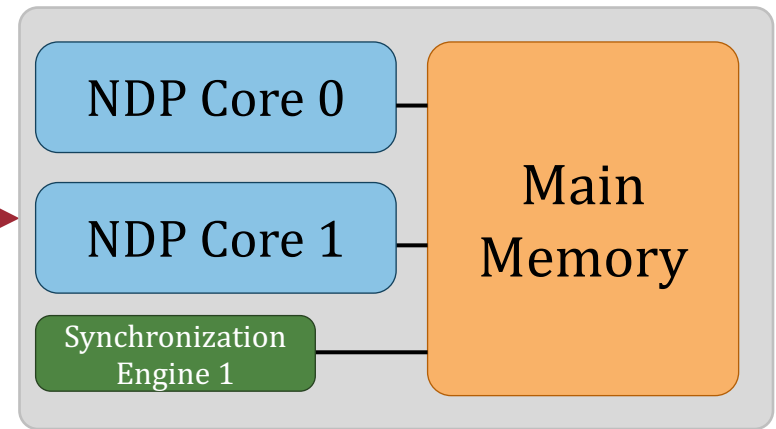


1. Hardware Synchronization Support

NDP Unit 0



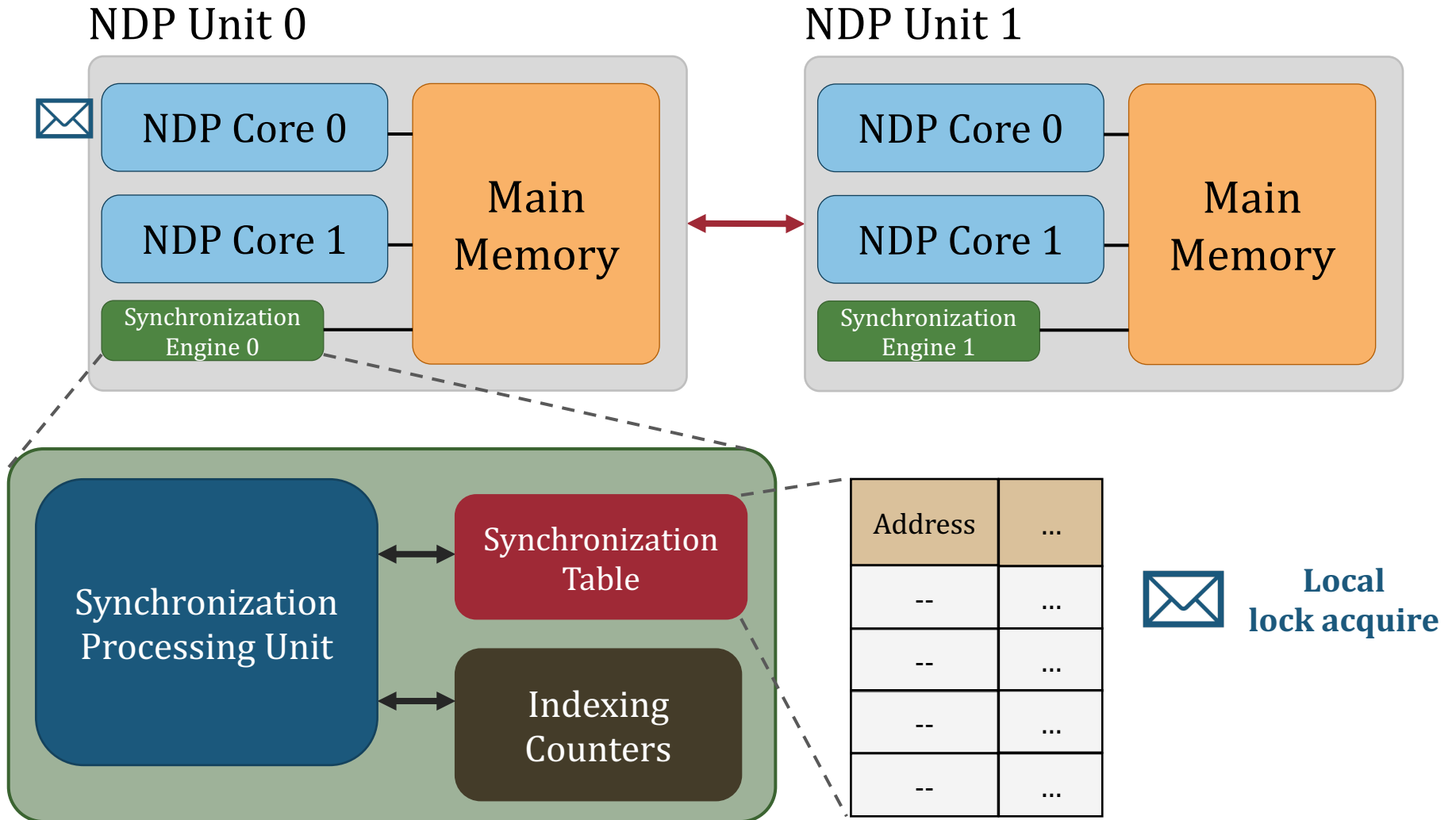
NDP Unit 1



Local
lock acquire

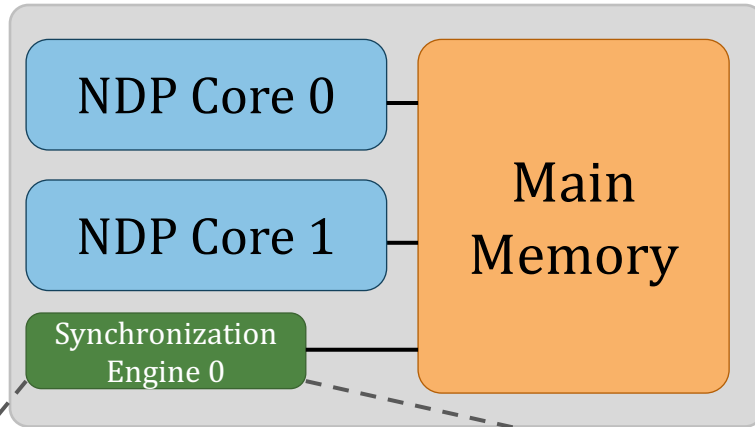
- ✓ No Complex Cache Coherence Protocols
- ✓ No Expensive Atomic Operations
- ✓ Low Hardware Cost

2. Direct Buffering of Variables

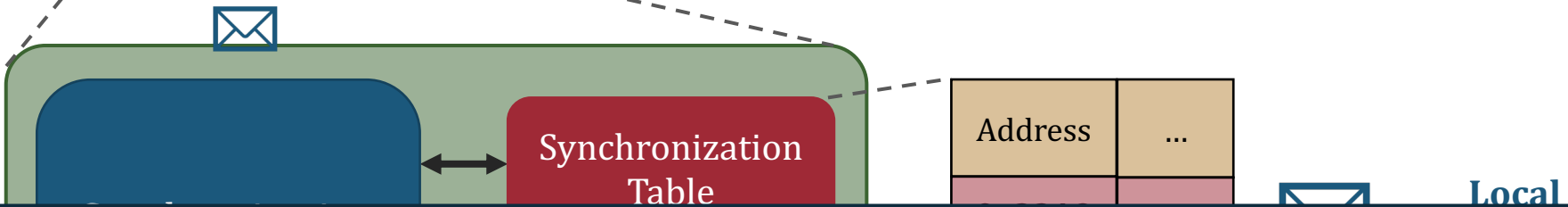
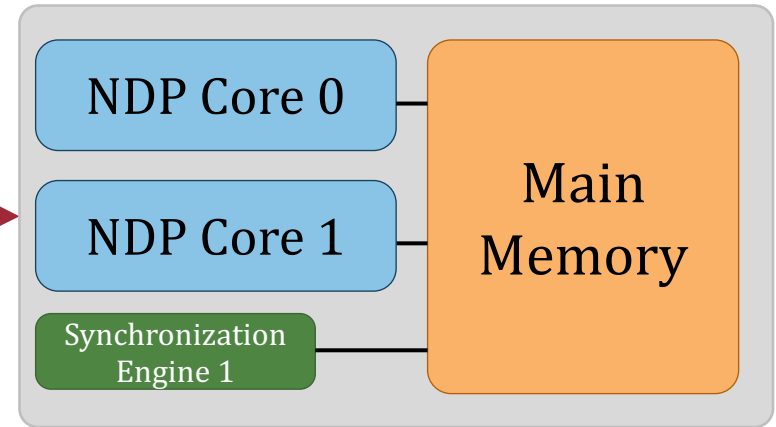


2. Direct Buffering of Variables

NDP Unit 0



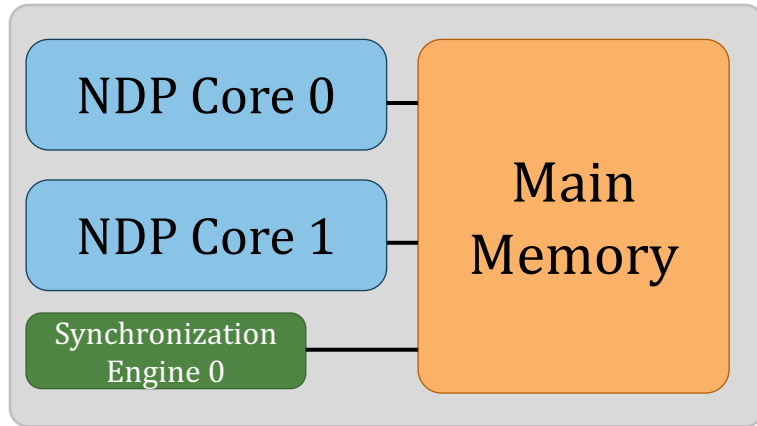
NDP Unit 1



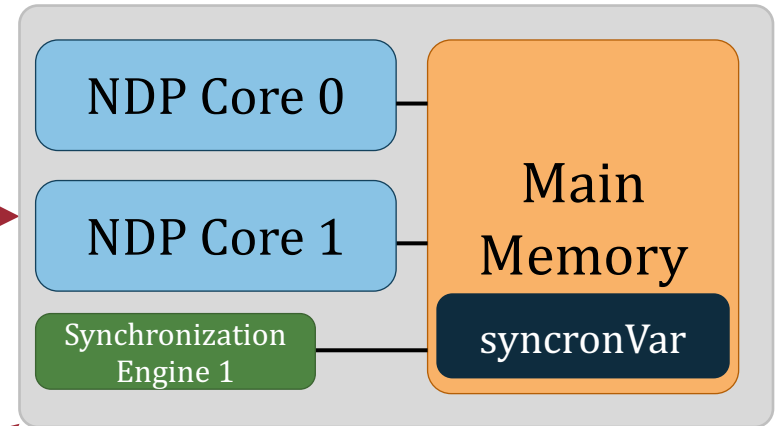
- ✓ No Costly Memory Accesses
- ✓ Low Latency

3. Hierarchical Communication

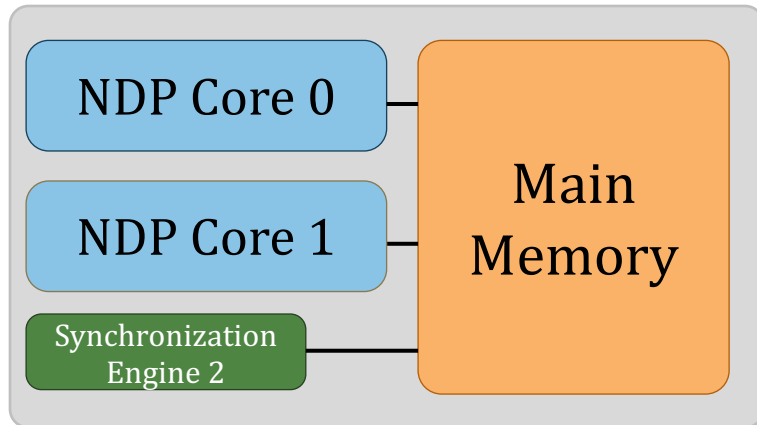
NDP Unit 0



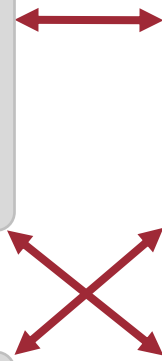
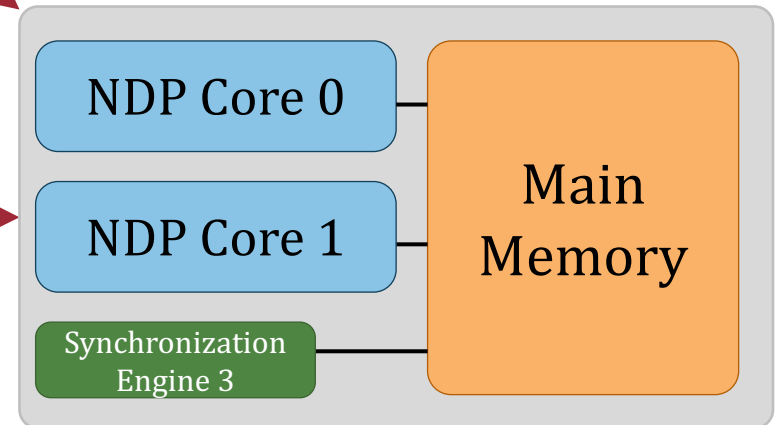
NDP Unit 1



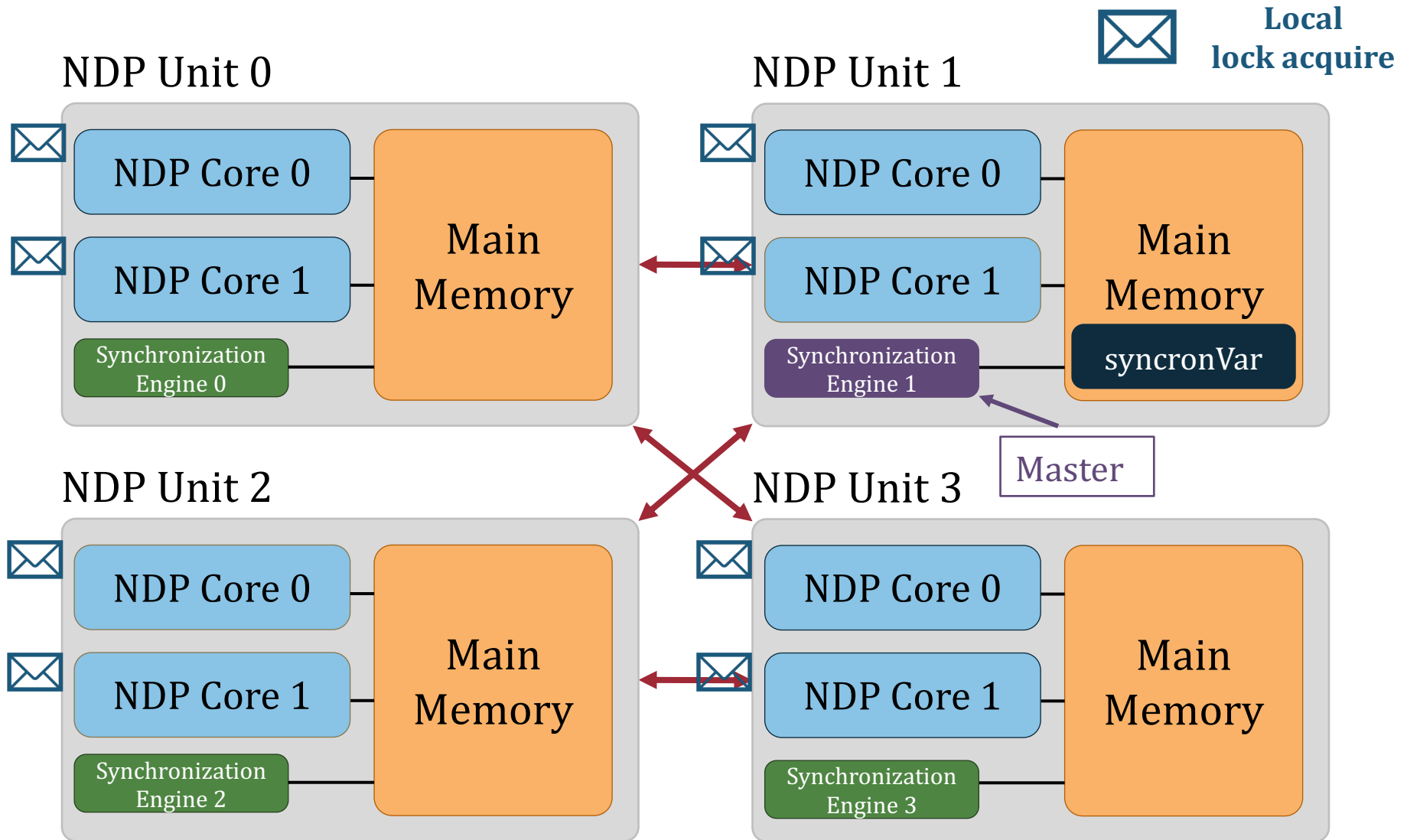
NDP Unit 2



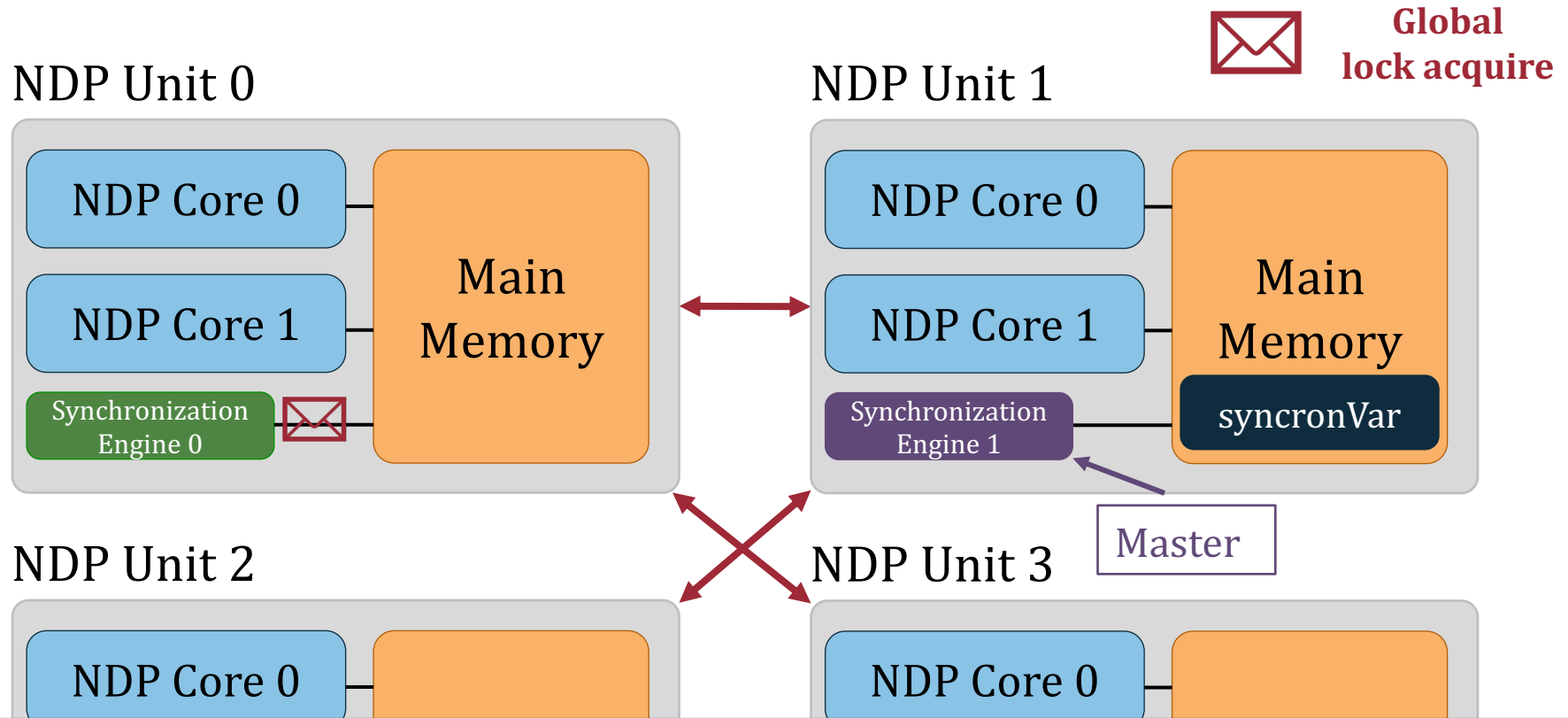
NDP Unit 3



3. Hierarchical Communication



3. Hierarchical Communication



✓ **Minimize Expensive Traffic**

SynCron

The first end-to-end synchronization solution for NDP architectures

SynCron's Benefits:

1. High System Performance
2. Low Hardware Cost

SynCron comes within 9.5% and 6.2% of performance and energy of Ideal zero-overhead synchronization

SynCron

Efficient Synchronization Support for Near-Data-Processing Architectures



Christina Giannoula

Nandita Vijaykumar, Nikela Papadopoulou, Vasileios Karakostas
Ivan Fernandez, Juan Gómez Luna, Lois Orosa
Nectarios Koziris, Georgios Goumas, Onur Mutlu

SAFARI



ETH zürich



How to Support Synchronization?

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

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

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

Lecture on Synchronization Support for PIM

1. Hardware Synchronization Support

NDP Unit 0

- NDP Core 0
- NDP Core 1
- Main Memory
- Synchronization Engine 0

NDP Unit 1

- NDP Core 0
- NDP Core 1
- Main Memory
- Synchronization Engine 1

Synchronization Table

- ✓ No Complex Cache Coherence Protocols
- ✓ No Expensive Atomic Operations
- ✓ Low Hardware Cost

19:47 / 1:04:55

Processing in Memory Course: Meeting 11: Synchronization Support for PIM Architectures - Fall'21

360 views • Streamed live on Dec 14, 2021

20 DISLIKE SHARE SAVE ...



Onur Mutlu Lectures
20.9K subscribers

SUBSCRIBED



How to Design Data Structures for PIM?

- Zhiyu Liu, Irina Calciu, Maurice Herlihy, and Onur Mutlu,
"Concurrent Data Structures for Near-Memory Computing"
Proceedings of the 29th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA), Washington, DC, USA, July 2017.
[[Slides \(pptx\)](#)] ([pdf](#))]

Concurrent Data Structures for Near-Memory Computing

Zhiyu Liu

Computer Science Department
Brown University
zhiyu.liu@brown.edu

Irina Calciu

VMware Research Group
icalciu@vmware.com

Maurice Herlihy

Computer Science Department
Brown University
mph@cs.brown.edu

Onur Mutlu

Computer Science Department
ETH Zürich
onur.mutlu@inf.ethz.ch

Virtual Memory Support

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

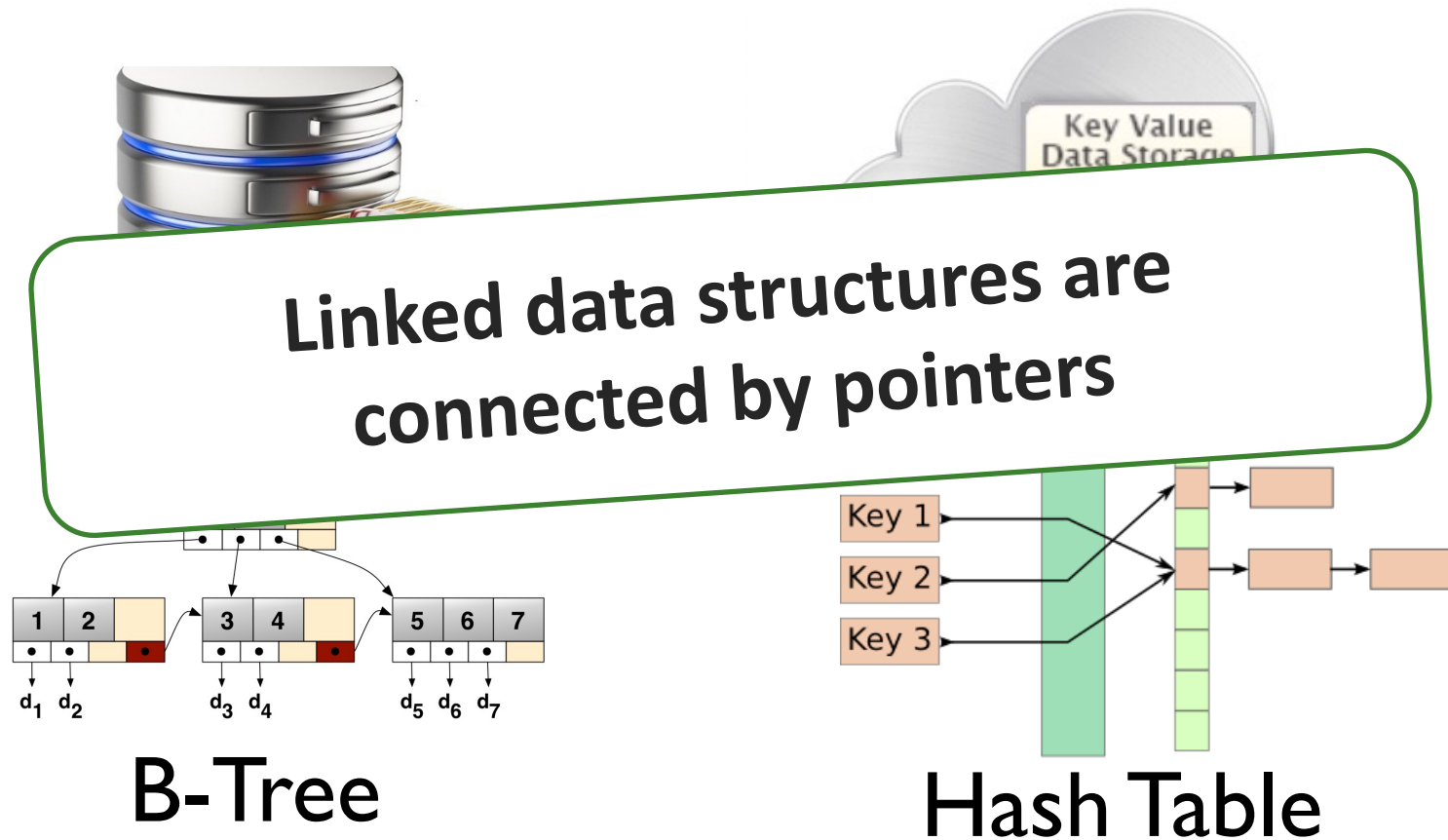
Kevin Hsieh[†] Samira Khan[‡] Nandita Vijaykumar[†]
Kevin K. Chang[†] Amirali Boroumand[†] Saugata Ghose[†] Onur Mutlu^{§†}
[†]*Carnegie Mellon University* [‡]*University of Virginia* [§]*ETH Zürich*

Executive Summary

- **Our Goal:** Accelerating pointer chasing inside main memory
- **Challenges:** Parallelism challenge and Address translation challenge
- **Our Solution:** In-Memory PoInter Chasing Accelerator (IMPICA)
 - Address-access decoupling: enabling parallelism in the accelerator with low cost
 - IMPICA page table: low cost page table in logic layer
- **Key Results:**
 - 1.2X – 1.9X speedup for pointer chasing operations, +16% database throughput
 - 6% - 41% reduction in energy consumption

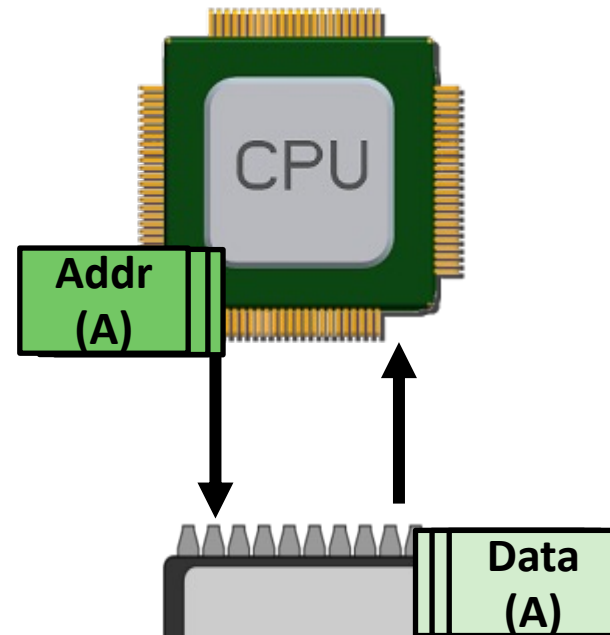
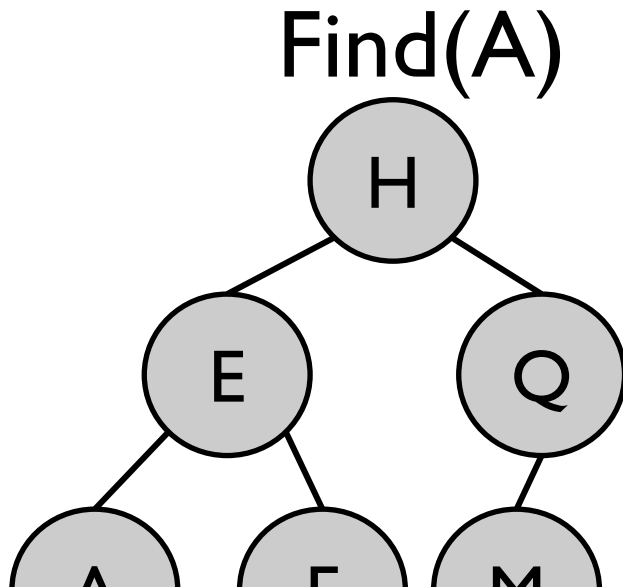
Linked Data Structures

- Linked data structures are widely used in many important applications



The Problem: Pointer Chasing

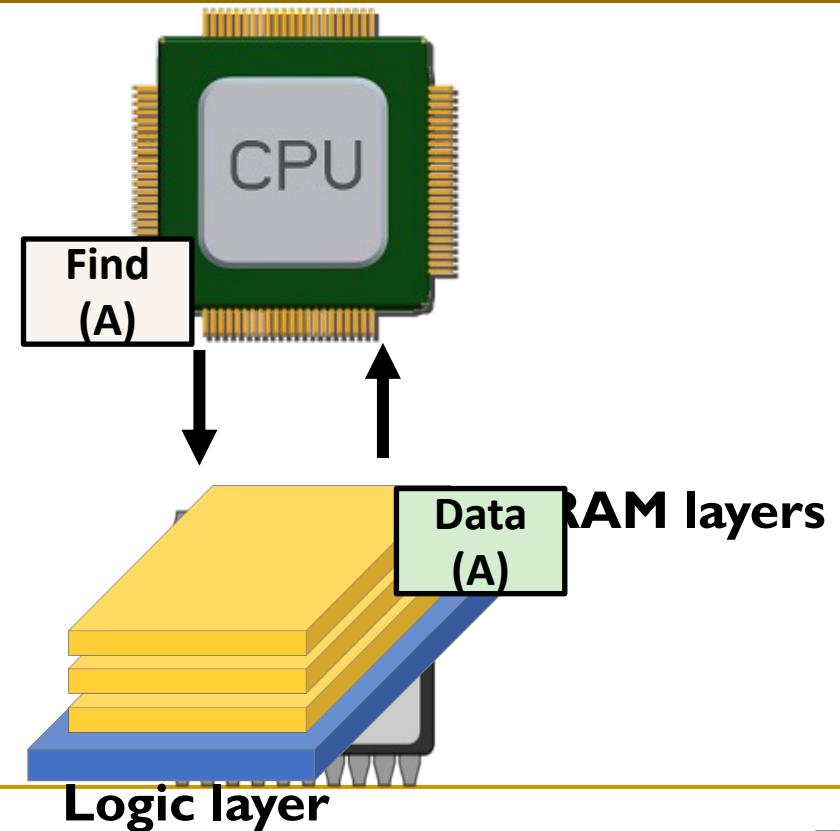
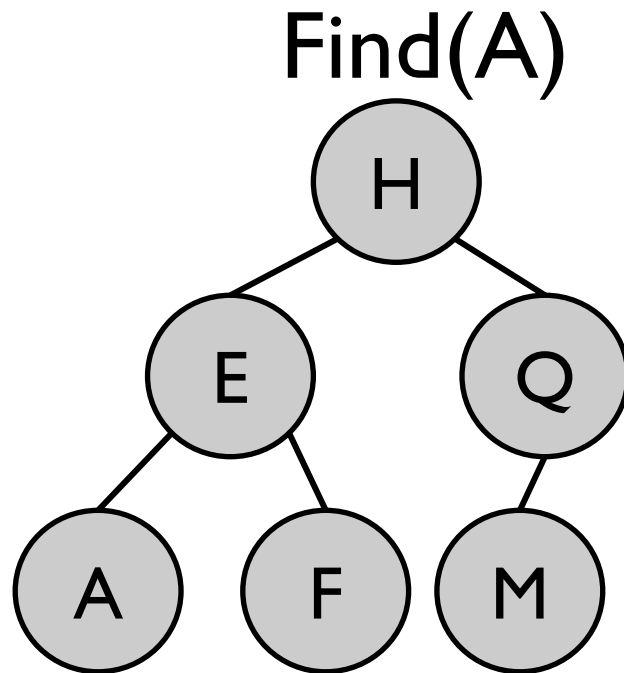
- Traversing linked data structures requires chasing pointers



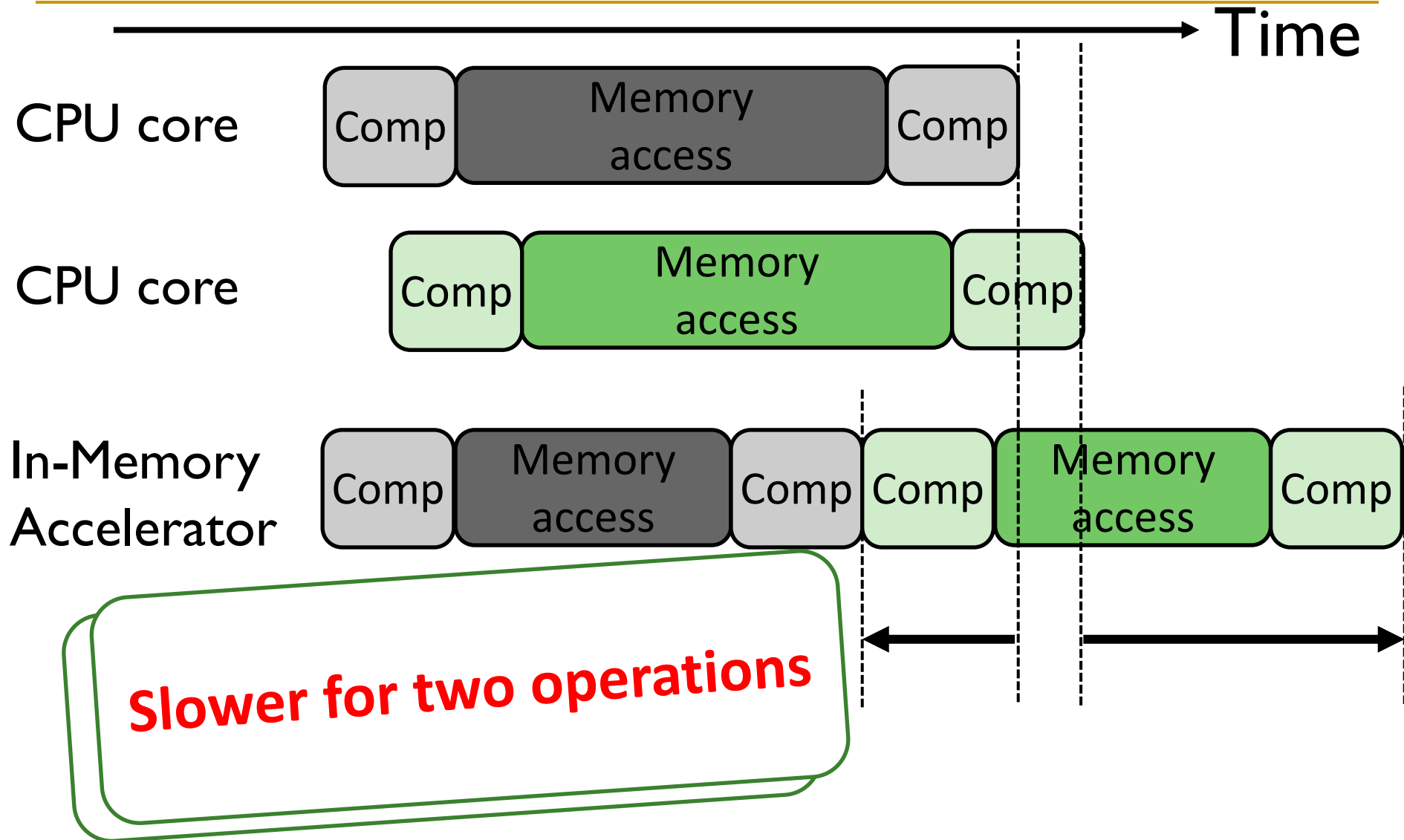
**Serialized and irregular access pattern
6X cycles per instruction in real workloads**

Our Goal

Accelerating pointer chasing inside main memory

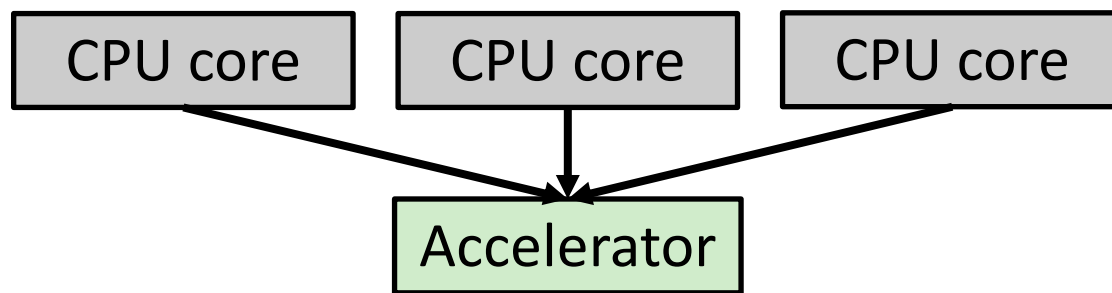


Parallelism Challenge

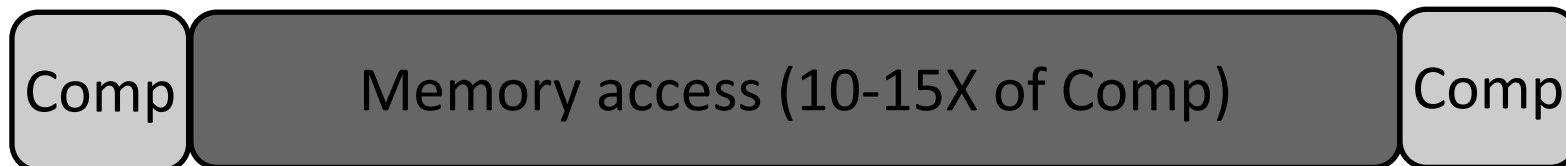


Parallelism Challenge and Opportunity

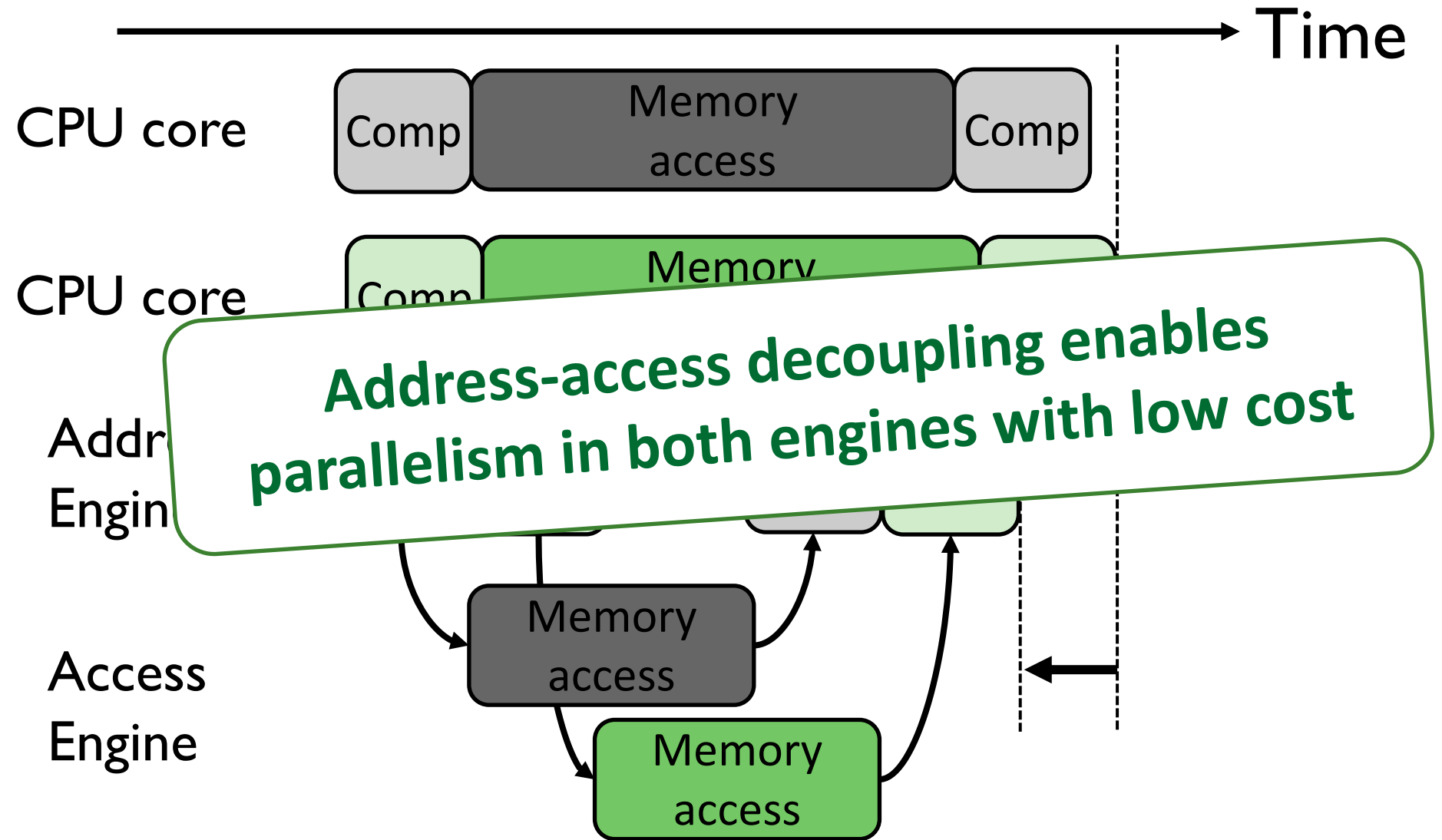
- A simple in-memory accelerator can still be **slower** than multiple CPU cores



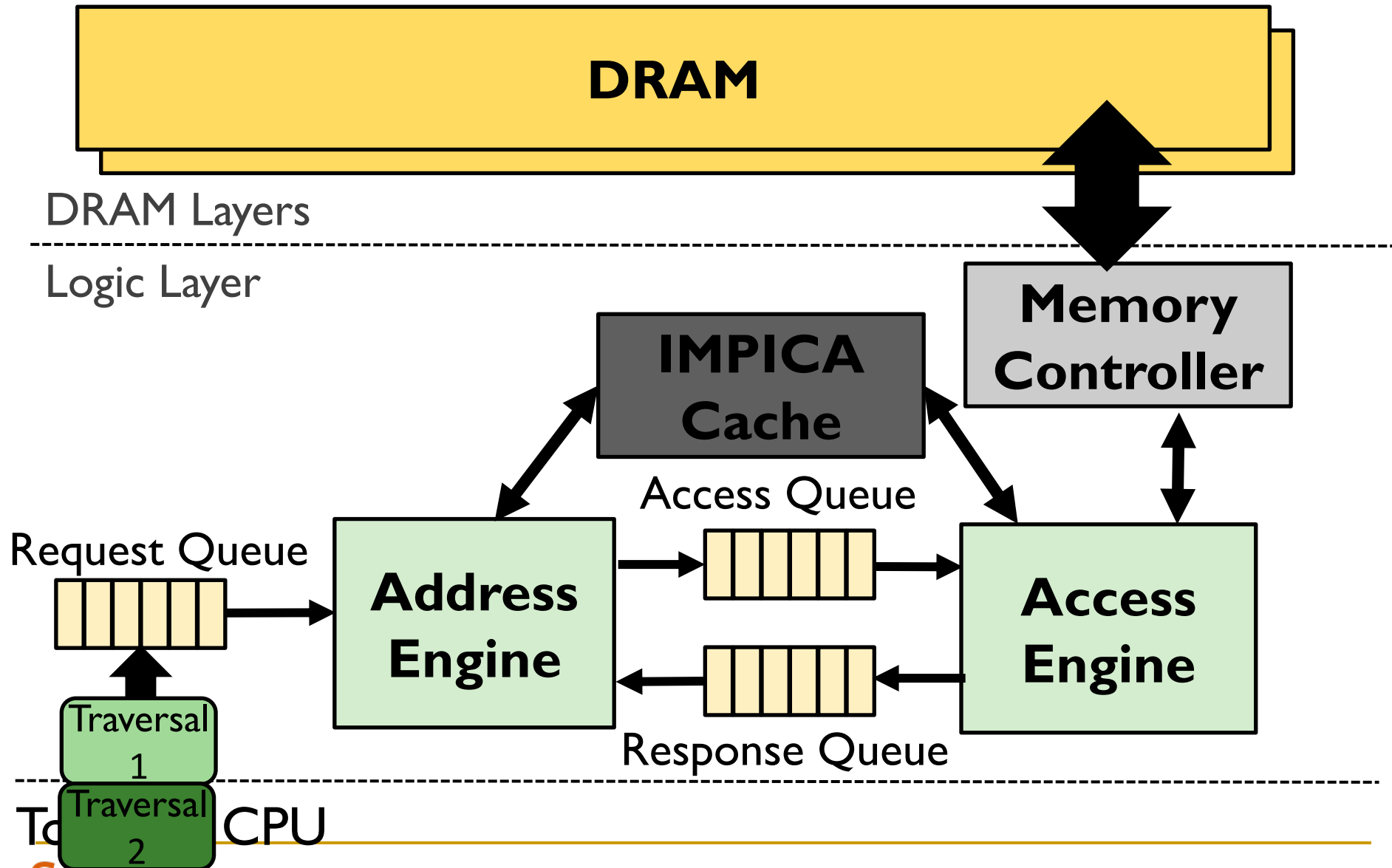
- **Opportunity:** a pointer-chasing accelerator spends a long time **waiting for memory**



Our Solution: Address-Access Decoupling



IMPICA Core Architecture

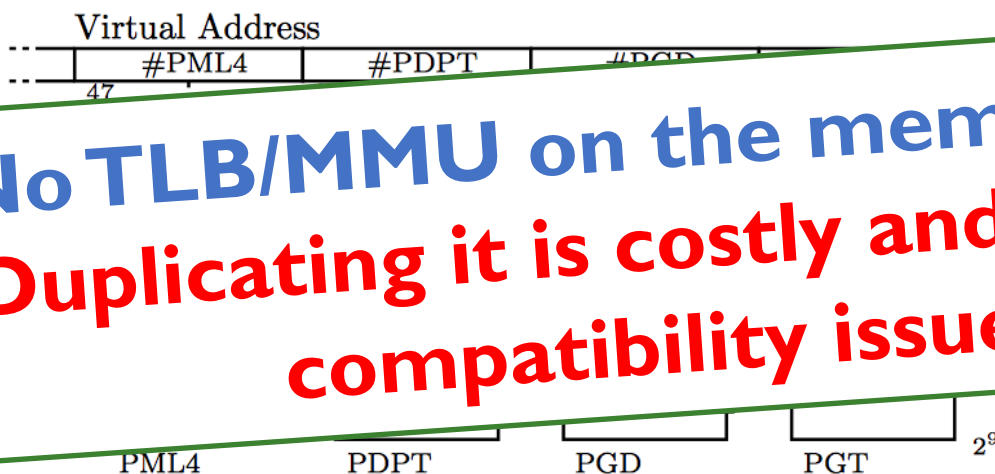


Address Translation Challenge

The page table walk requires multiple memory accesses



No TLB/MMU on the memory side
Duplicating it is costly and creates compatibility issue



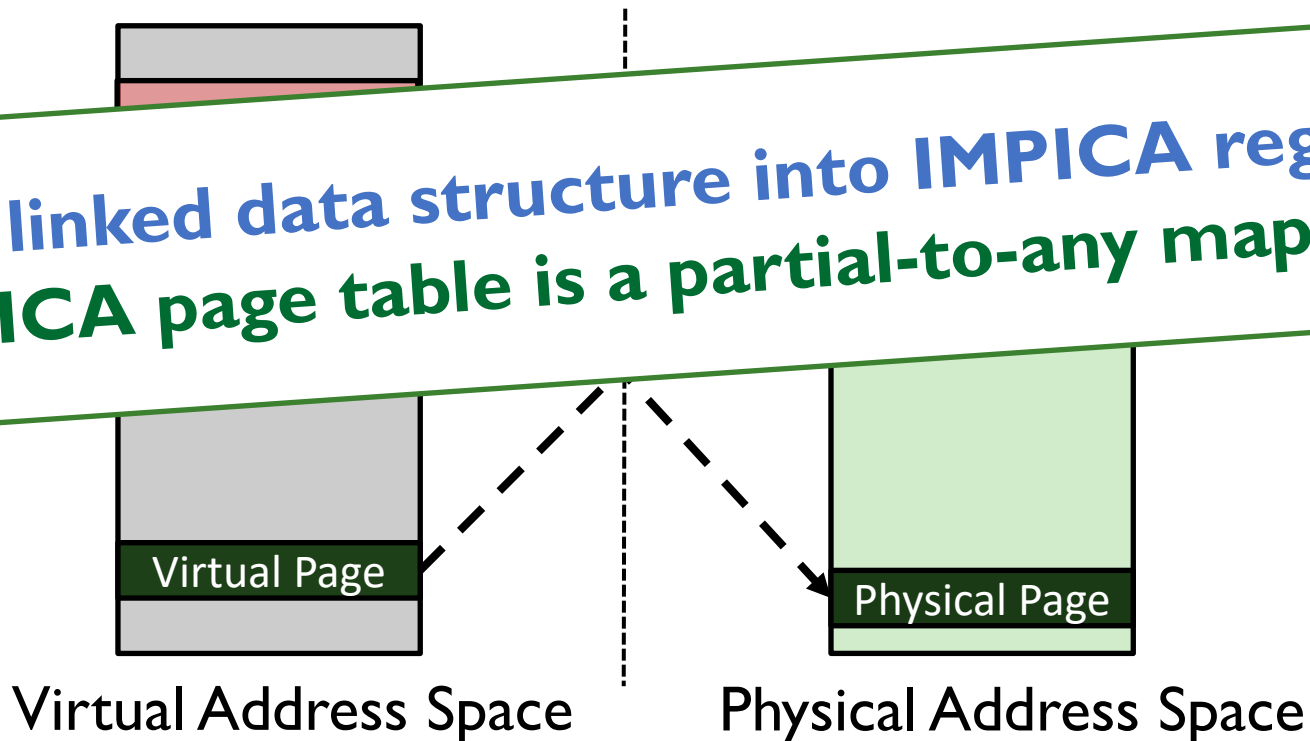
Page table walk

Our Solution: IMPICA Page Table

- Completely decouple the page table of IMPICA from the page table of the CPUs

IMPICA Page Table

Map linked data structure into IMPICA regions
IMPICA page table is a partial-to-any mapping



IMPICA Page Table: Mechanism

Virtual Address

Bit [47:4]

Bit [11:0]

**Flat page table
saves one memory access**

Region Table

+

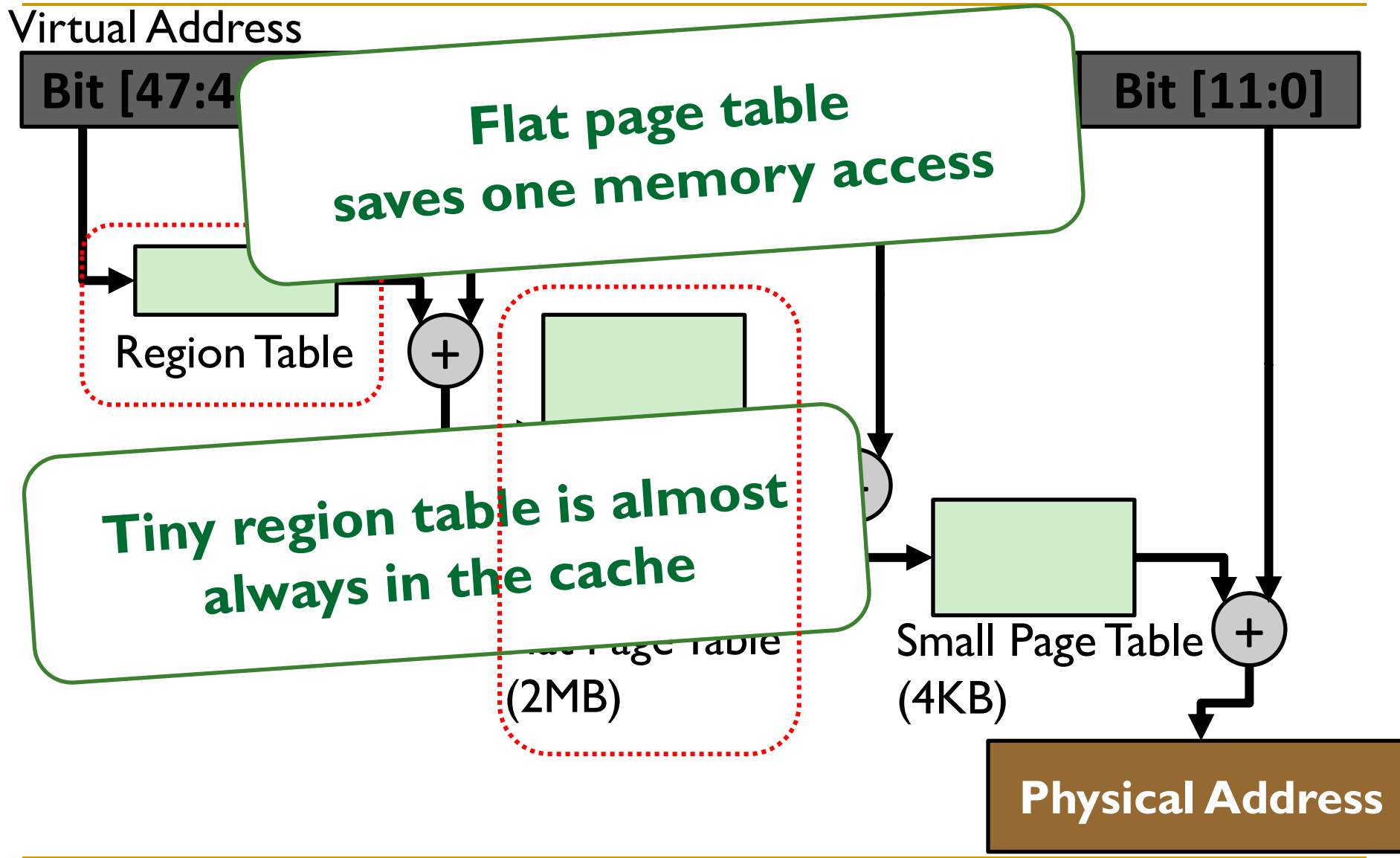
**Tiny region table is almost
always in the cache**

Region Page Table
(2MB)

Small Page Table
(4KB)

+

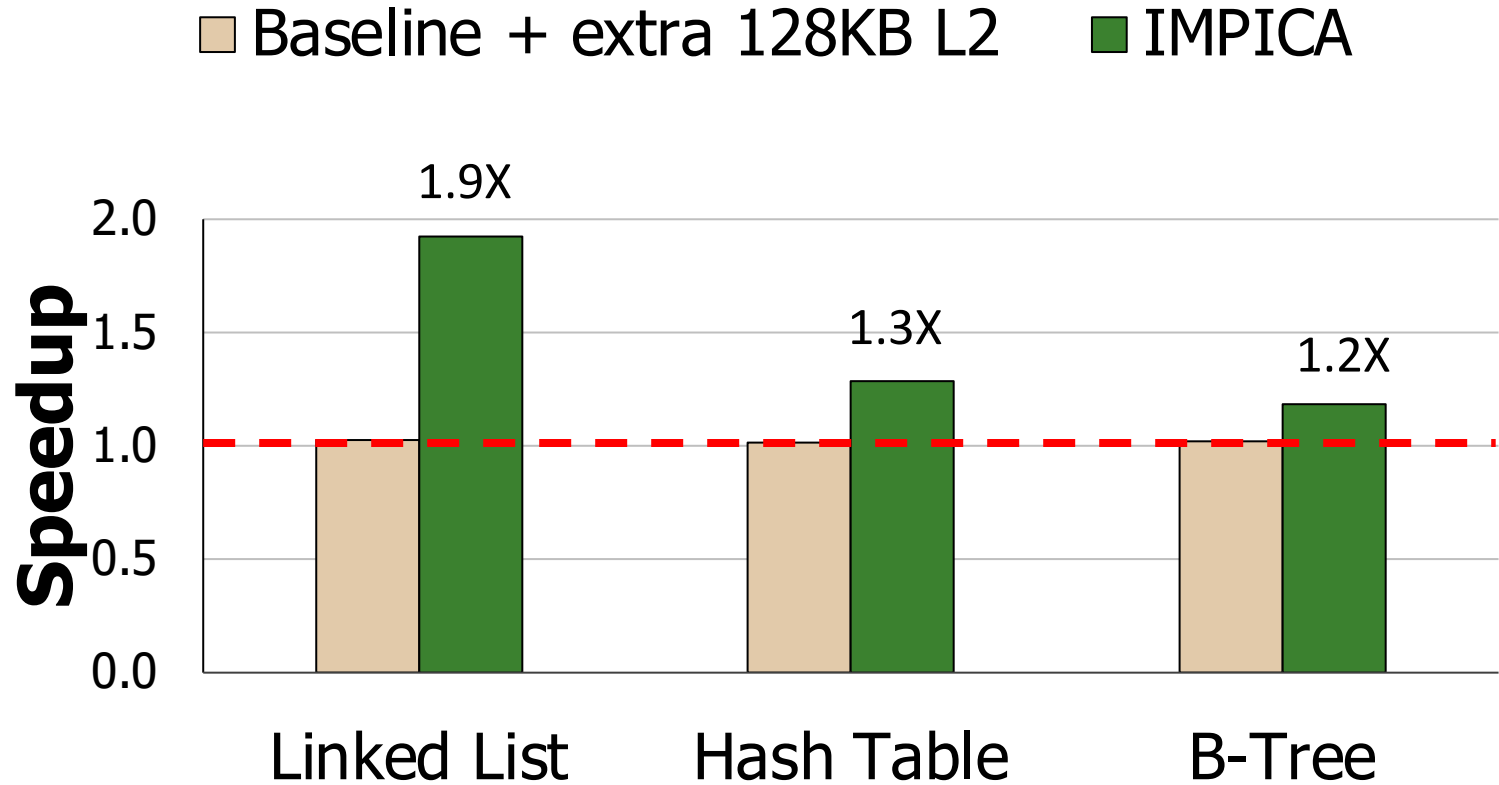
Physical Address



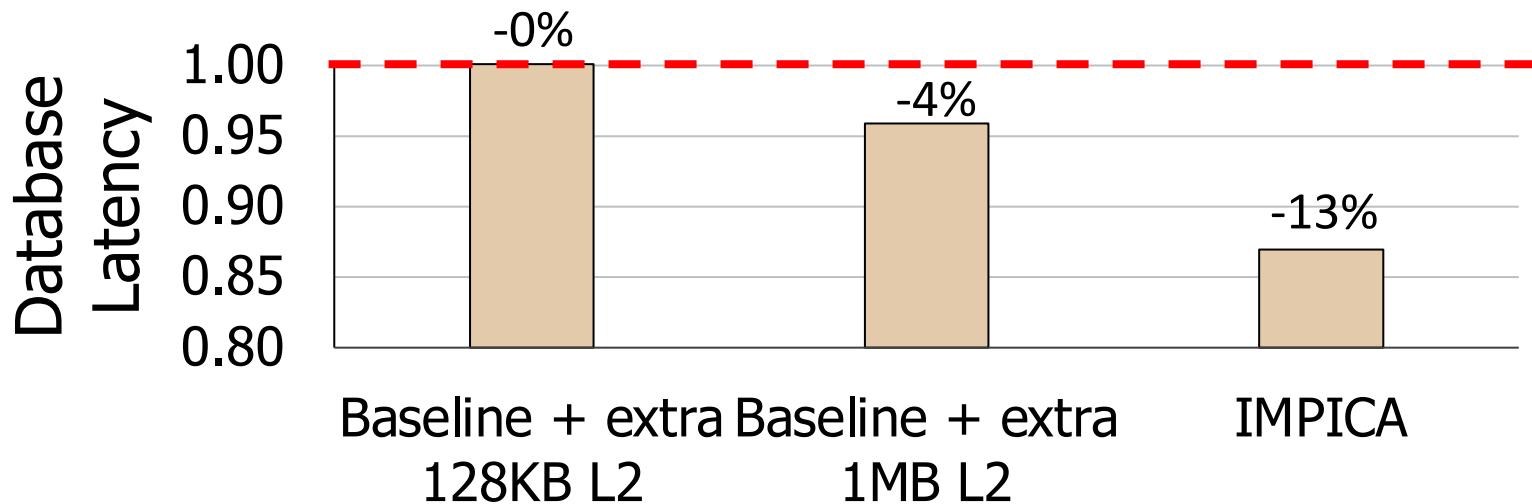
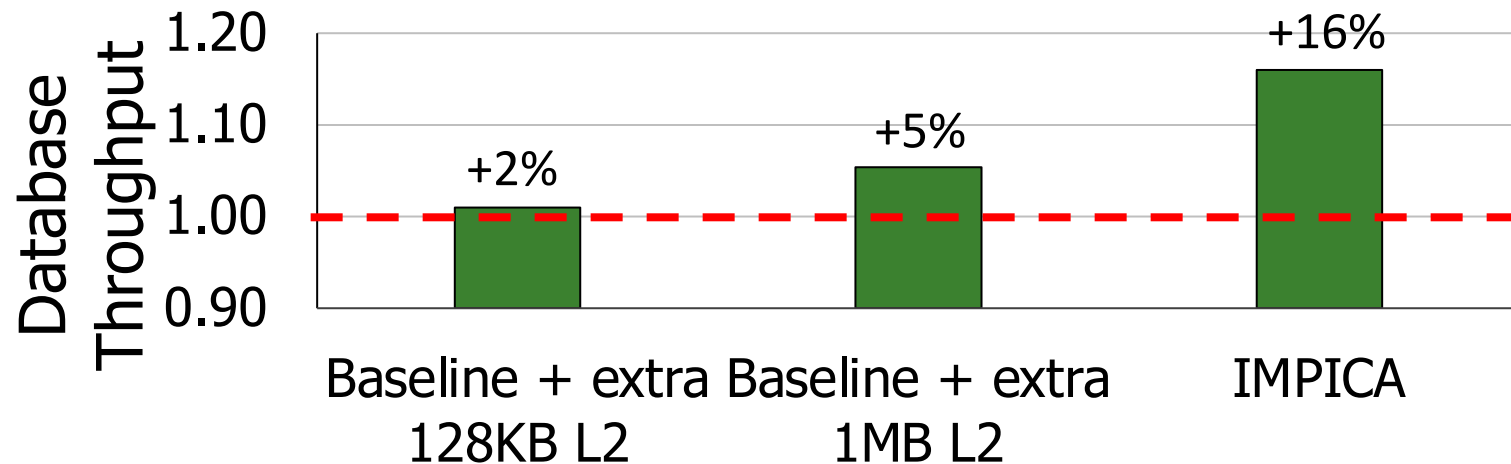
Evaluation Methodology

- Simulator: [gem5](#)
- System Configuration
 - CPU
 - 4 OoO cores, 2GHz
 - Cache: 32KB L1, 1MB L2
 - IMPICA
 - 1 core, 500MHz, 32KB Cache
 - Memory Bandwidth
 - 12.8 GB/s for CPU, 51.2 GB/s for IMPICA
- Our simulator code is open source
 - <https://github.com/CMU-SAFARI/IMPICA>

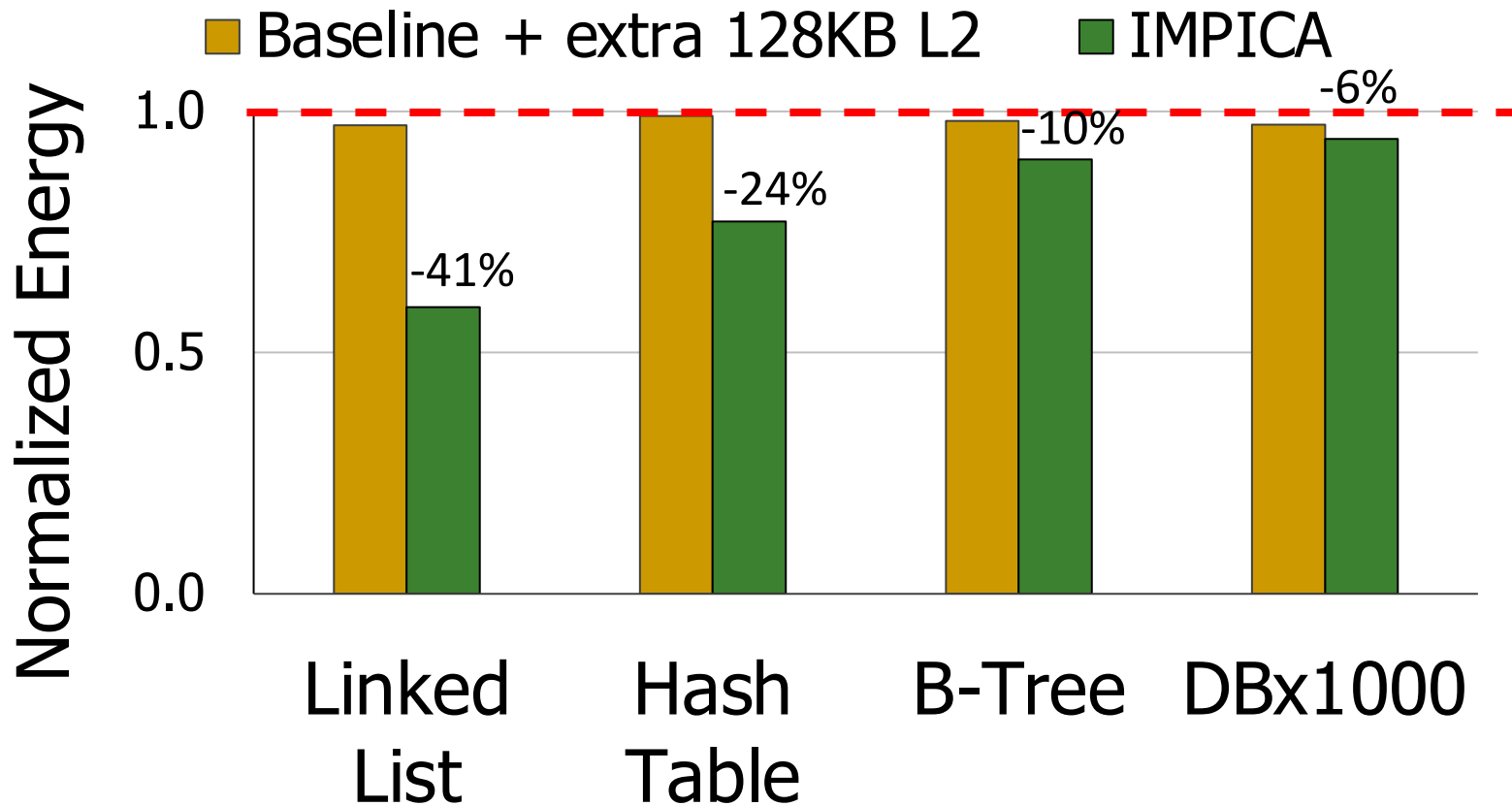
Result – Microbenchmark Performance



Result – Database Performance



System Energy Consumption



Area and Power Overhead

CPU (Cortex-A57)	5.85 mm ² per core
L2 Cache	5 mm ² per MB
Memory Controller	10 mm ²
IMPICA (+32KB cache)	0.45 mm ²

- Power overhead: average power increases by 5.6%

How to Support Virtual Memory?

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

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

Kevin Hsieh[†] Samira Khan[‡] Nandita Vijaykumar[†]

Kevin K. Chang[†] Amirali Boroumand[†] Saugata Ghose[†] Onur Mutlu^{§†}

[†]*Carnegie Mellon University* [‡]*University of Virginia* [§]*ETH Zürich*

Rethinking Virtual Memory

Nastaran Hajinazar, Pratyush Patel, Minesh Patel, Konstantinos Kanellopoulos, Saugata Ghose, Rachata Ausavarungnirun, Geraldo Francisco de Oliveira Jr., Jonathan Appavoo, Vivek Seshadri, and Onur Mutlu, "[The Virtual Block Interface: A Flexible Alternative to the Conventional Virtual Memory Framework](#)"

Proceedings of the 47th International Symposium on Computer Architecture (ISCA), Virtual, June 2020.

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

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

[[ARM Research Summit Poster \(pptx\)](#) ([pdf](#))]

[[Talk Video](#) (26 minutes)]

[[Lightning Talk Video](#) (3 minutes)]

[[Lecture Video](#) (43 minutes)]

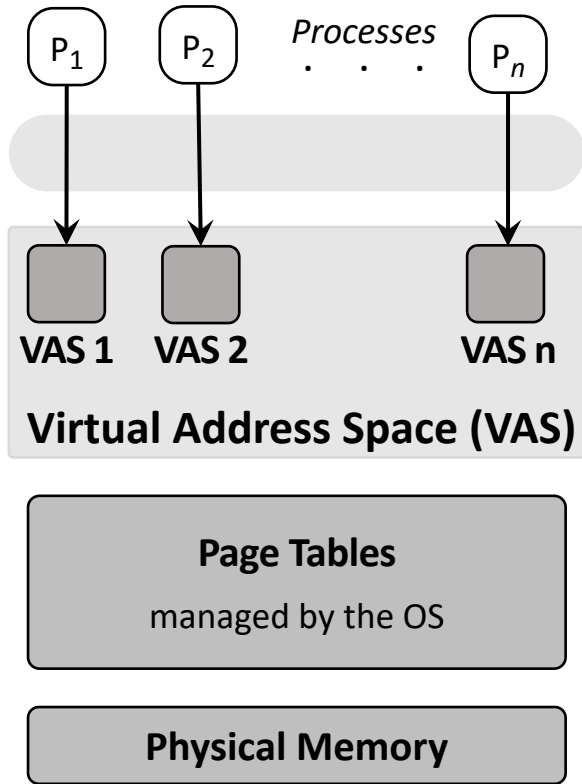
The Virtual Block Interface: A Flexible Alternative to the Conventional Virtual Memory Framework

Nastaran Hajinazar^{*†} Pratyush Patel[✕] Minesh Patel^{*} Konstantinos Kanellopoulos^{*} Saugata Ghose[‡]
Rachata Ausavarungnirun[○] Geraldo F. Oliveira^{*} Jonathan Appavoo[◇] Vivek Seshadri[▽] Onur Mutlu^{*‡}

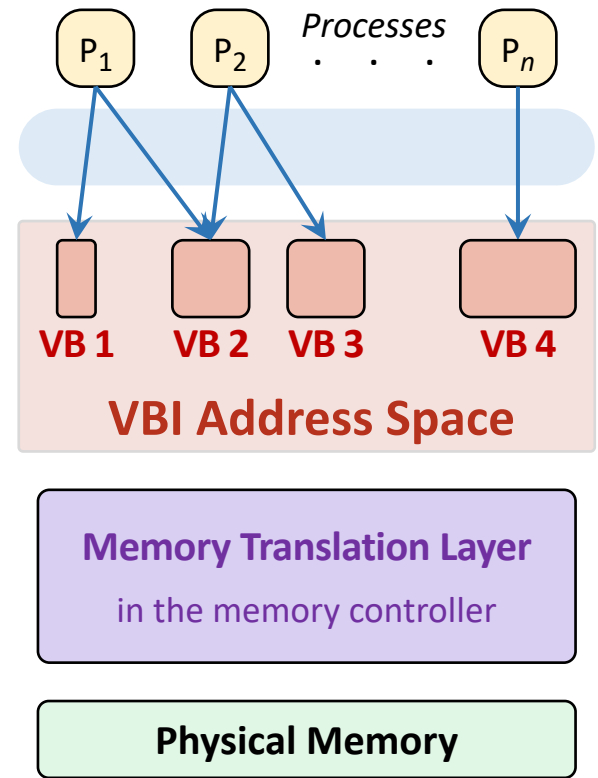
^{*}ETH Zürich [†]Simon Fraser University [✕]University of Washington [‡]Carnegie Mellon University

[○]King Mongkut's University of Technology North Bangkok [◇]Boston University [▽]Microsoft Research India

VBI: Overview



Conventional Virtual Memory

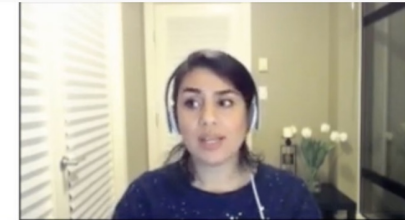


VBI

Lecture on Virtual Block Interface

Challenges

- **Three examples** of the **challenges** in adapting conventional virtual memory frameworks for increasingly-diverse systems:
 - Requiring a **rigid page table structure**
 - High address **translation overhead** in virtual machines
 - **Inefficient** heterogeneous memory **management**



ETH ZÜRICH HAUPTGEBÄUDE

Computer Architecture - Lecture 12c: The Virtual Block Interface (ETH Zürich, Fall 2020)

726 views • Oct 31, 2020

16 0 SHARE SAVE ...



Onur Mutlu Lectures
16.5K subscribers

ANALYTICS

EDIT VIDEO

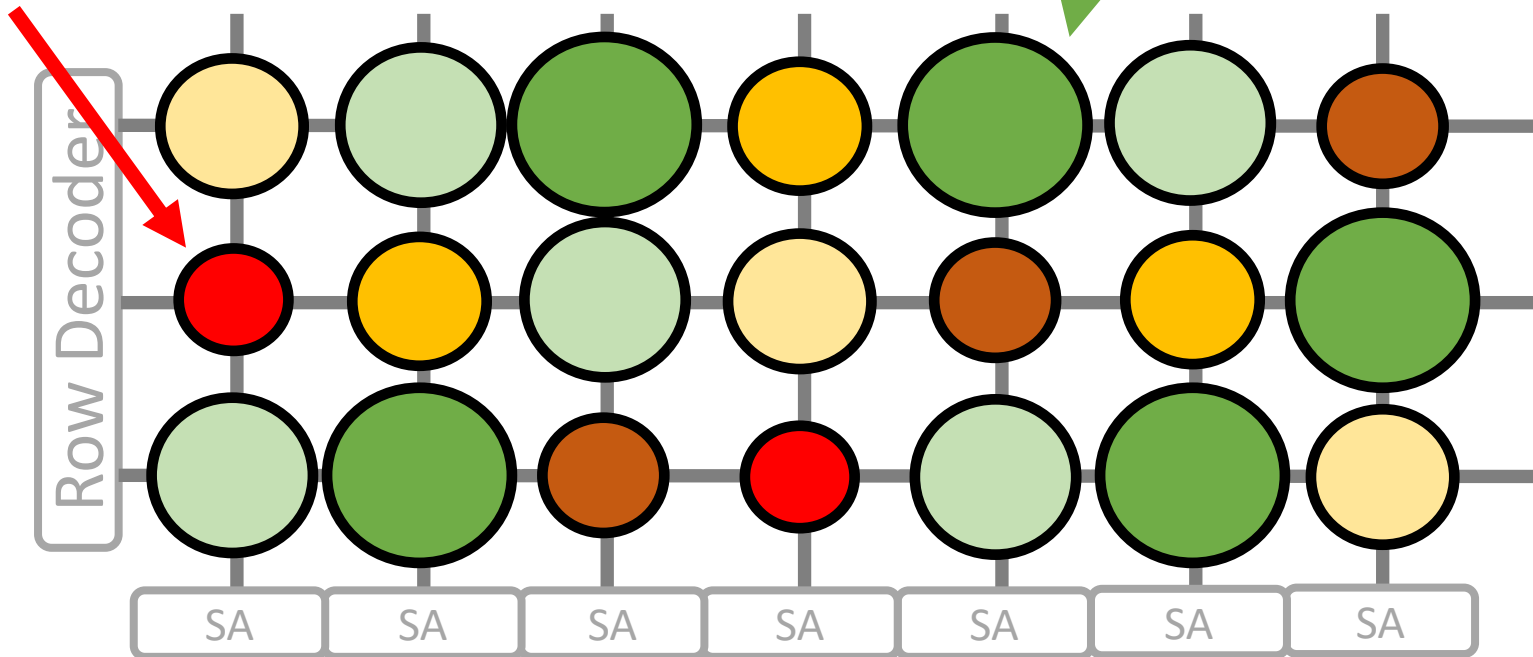
Security Considerations

DRAM Latency PUF Key Idea

- A cell's latency failure probability is inherently related to **random process variation** from manufacturing
- We can provide **repeatable and unique device signatures** using latency error patterns

**High % chance to fail
with reduced t_{RCD}**

**Low % chance to fail
with reduced t_{RCD}**



DRAM Latency Physical Unclonable Functions

- Jeremie S. Kim, Minesh Patel, Hasan Hassan, and Onur Mutlu,
["The DRAM Latency PUF: Quickly Evaluating Physical Unclonable Functions by Exploiting the Latency-Reliability Tradeoff in Modern DRAM Devices"](#)
Proceedings of the 24th International Symposium on High-Performance Computer Architecture (HPCA), Vienna, Austria, February 2018.
[[Lightning Talk Video](#)]
[[Slides \(pptx\)](#)] [[pdf](#)] [[Lightning Session Slides \(pptx\)](#)] [[pdf](#)]
[[Full Talk Lecture Video](#) (28 minutes)]

The DRAM Latency PUF:

Quickly Evaluating Physical Unclonable Functions

by Exploiting the Latency-Reliability Tradeoff in Modern Commodity DRAM Devices

Jeremie S. Kim^{†§}

Minesh Patel[§]

Hasan Hassan[§]

Onur Mutlu^{§†}

[†]Carnegie Mellon University

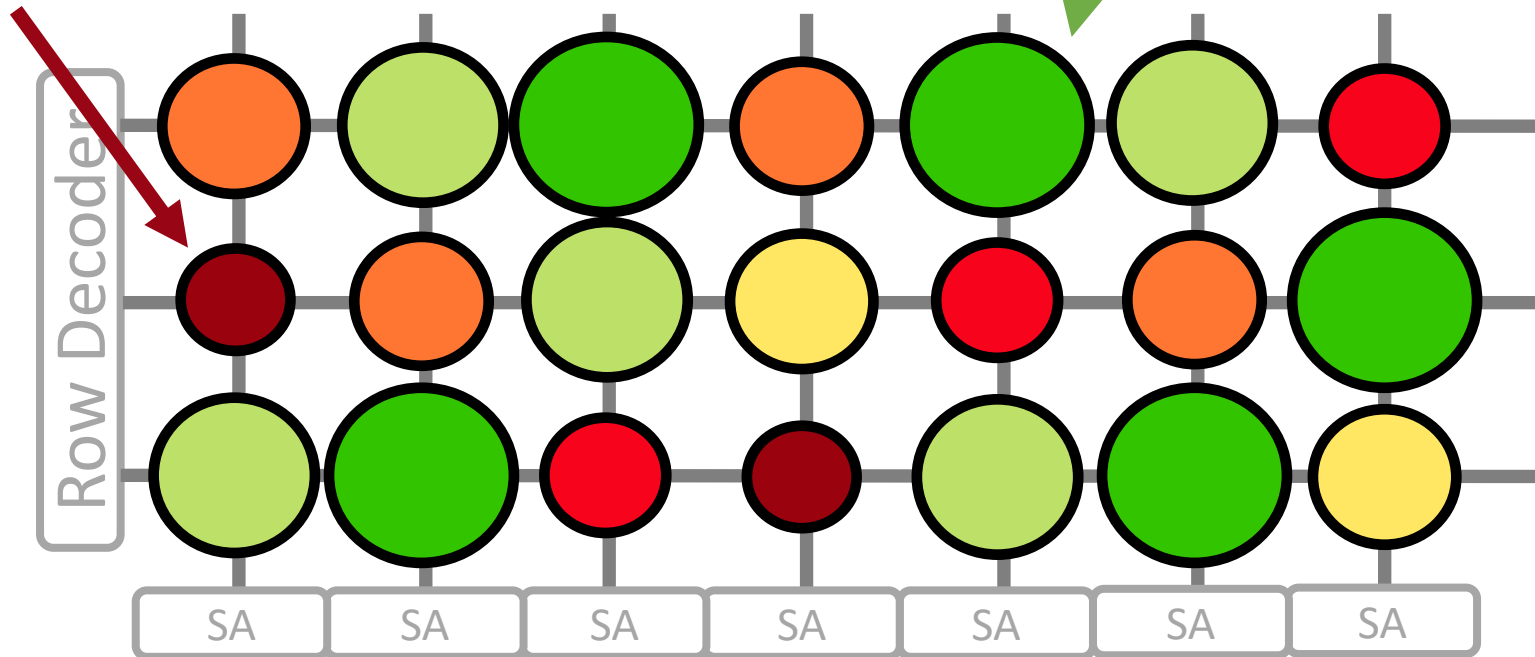
[§]ETH Zürich

D-RaNGe Key Idea

- A cell's latency failure probability is inherently related to **random process variation** from manufacturing
- We can extract **random values** by observing DRAM cells' latency failure probabilities

High % chance to fail
with reduced t_{RCD}

Low % chance to fail
with reduced t_{RCD}



DRAM Latency True Random Number Generator

- Jeremie S. Kim, Minesh Patel, Hasan Hassan, Lois Orosa, and Onur Mutlu, **"D-RaNGe: Using Commodity DRAM Devices to Generate True Random Numbers with Low Latency and High Throughput"**
Proceedings of the 25th International Symposium on High-Performance Computer Architecture (HPCA), Washington, DC, USA, February 2019.
[[Slides \(pptx\)](#)] [[pdf](#)]
[[Full Talk Video](#) (21 minutes)]
[[Full Talk Lecture Video](#) (27 minutes)]
Top Picks Honorable Mention by IEEE Micro.

D-RaNGe: Using Commodity DRAM Devices to Generate True Random Numbers with Low Latency and High Throughput

Jeremie S. Kim^{‡§}

Minesh Patel[§]

Hasan Hassan[§]

Lois Orosa[§]

Onur Mutlu^{§‡}

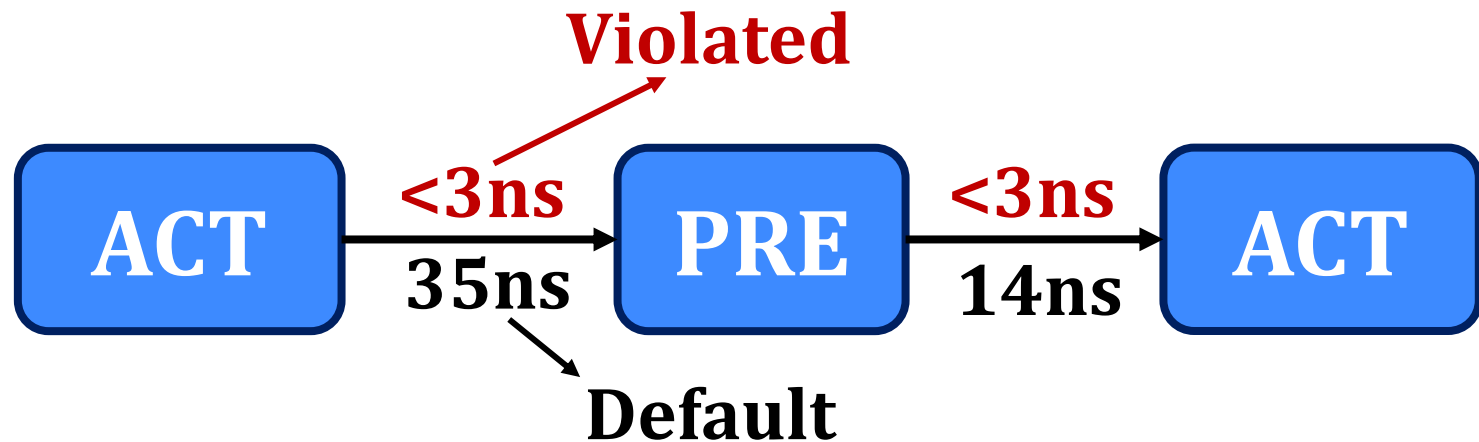
[‡]Carnegie Mellon University

[§]ETH Zürich

Quadruple Activation (QUAC)

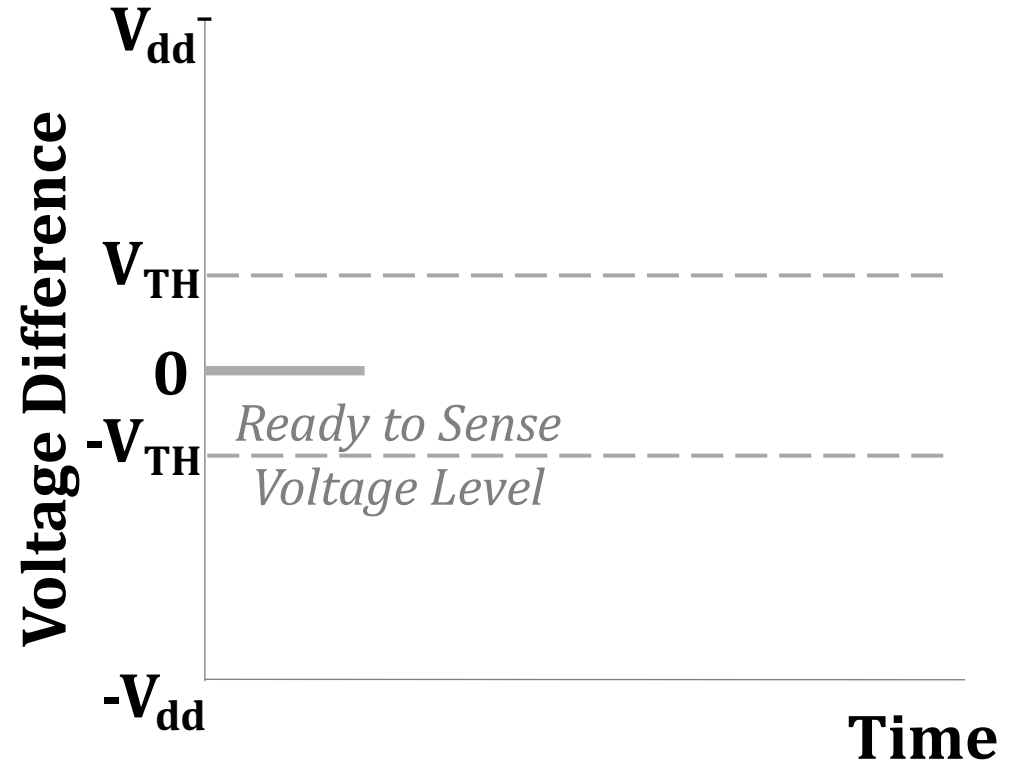
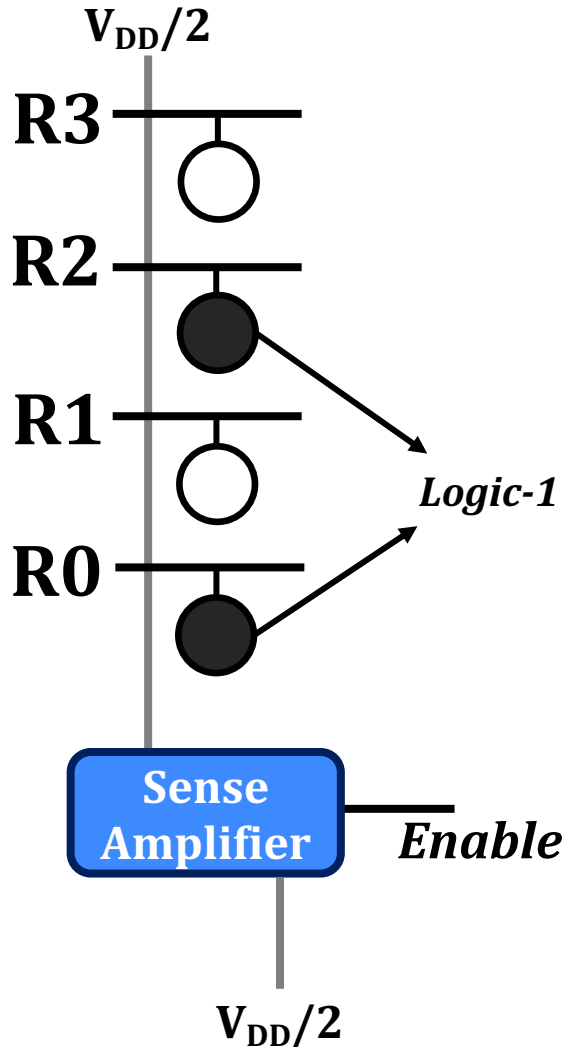
New Observation

Carefully-engineered DRAM commands can activate four rows in real DRAM chips

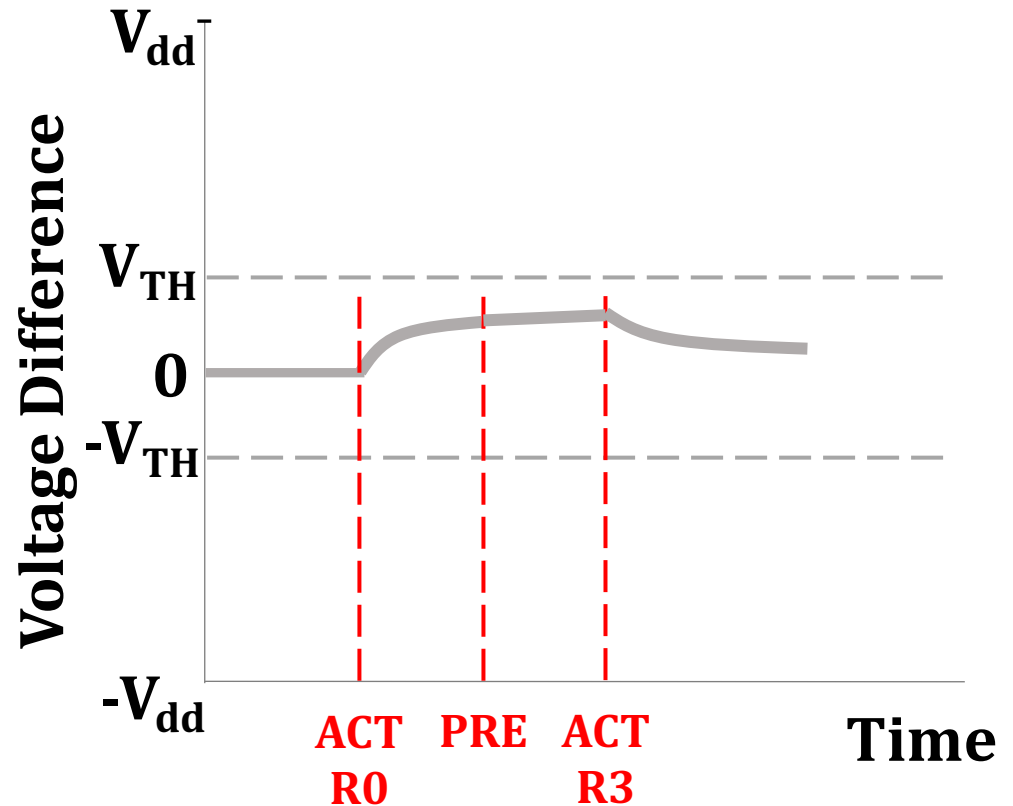
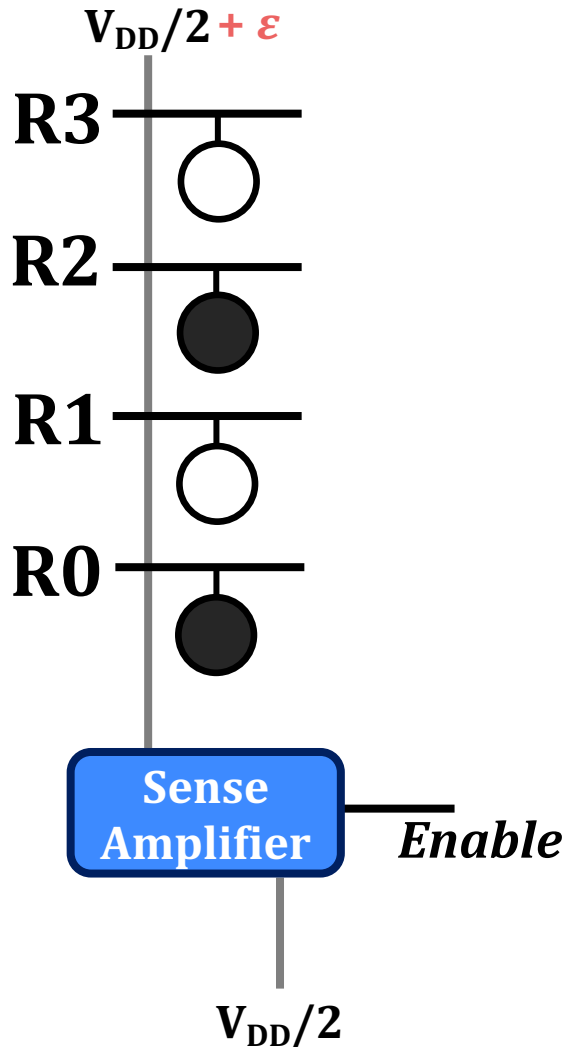


Activate four rows with two ACT commands

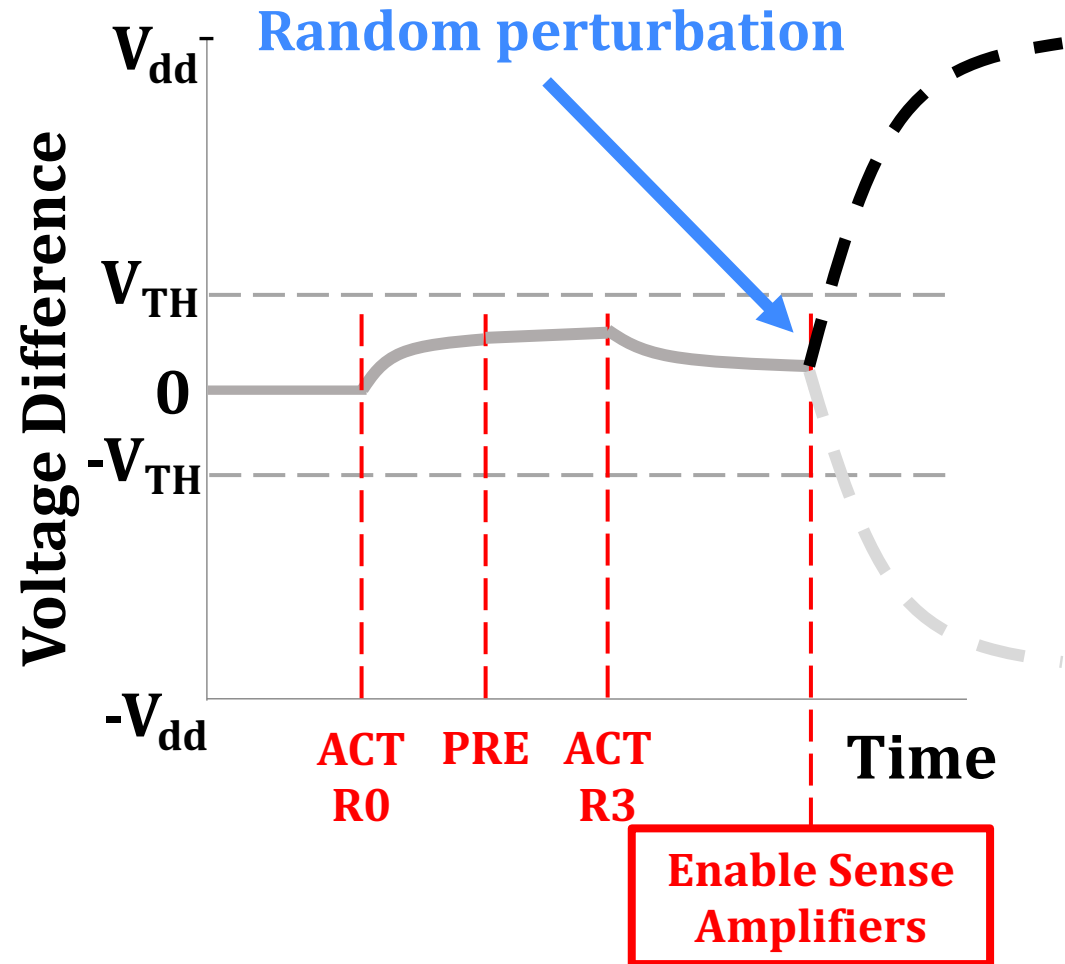
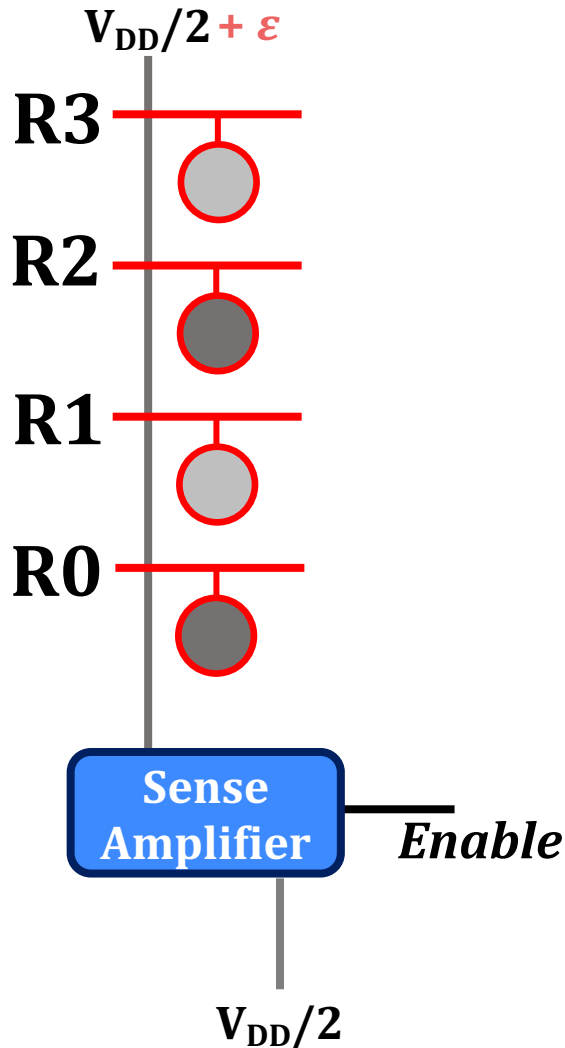
Generating Random Values via QUAC



Generating Random Values via QUAC

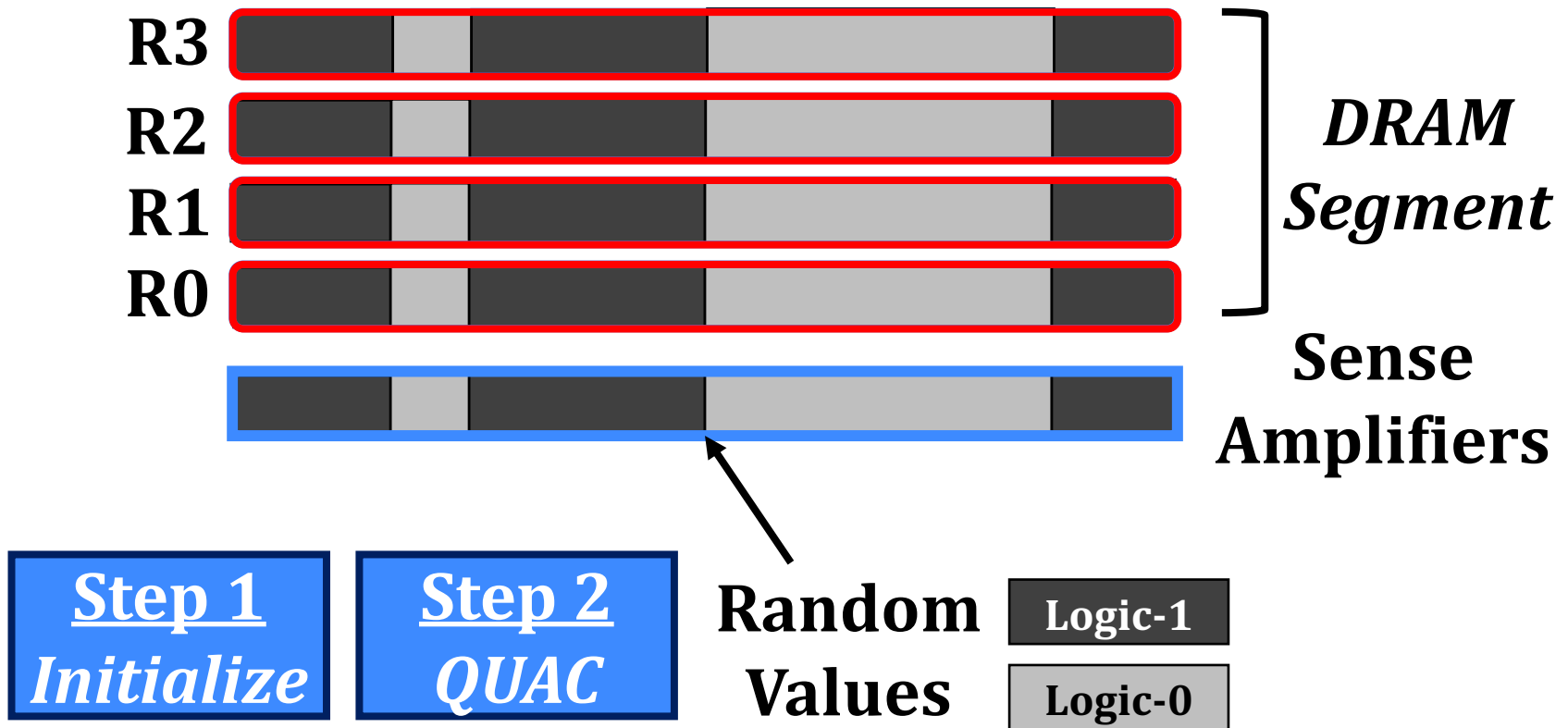


Generating Random Values via QUAC



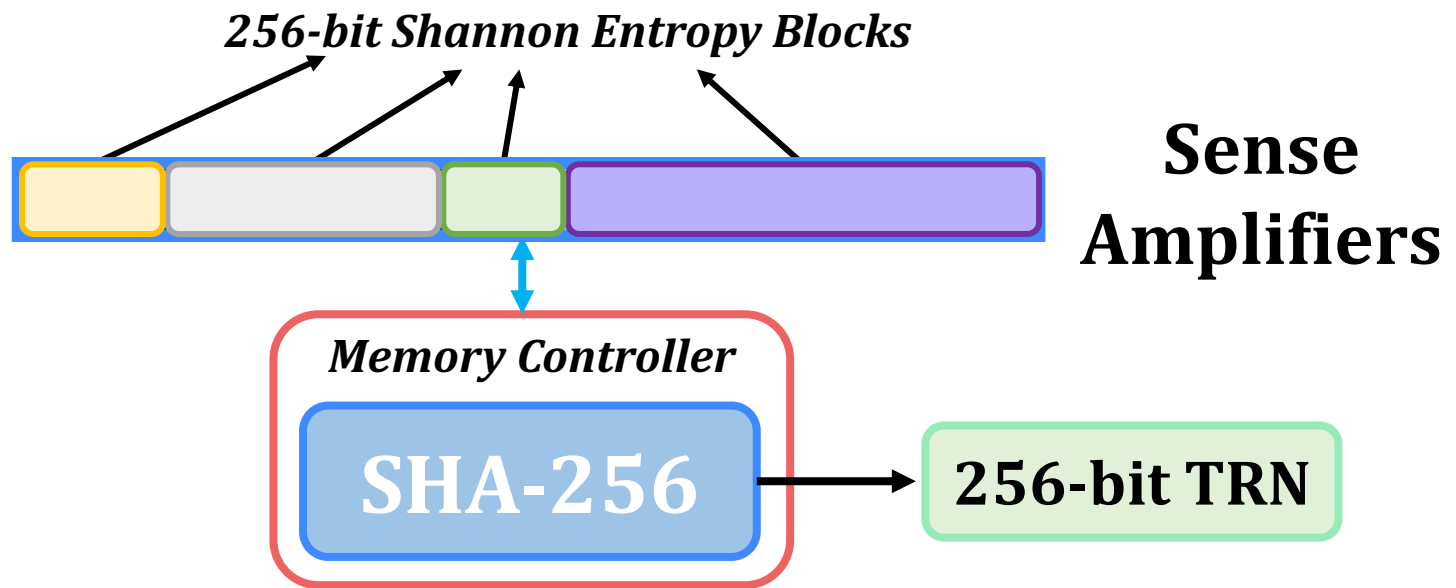
QUAC-TRNG

Key Idea: Leverage **random values** on sense amplifiers generated by **QUAC** operations as **source of entropy**



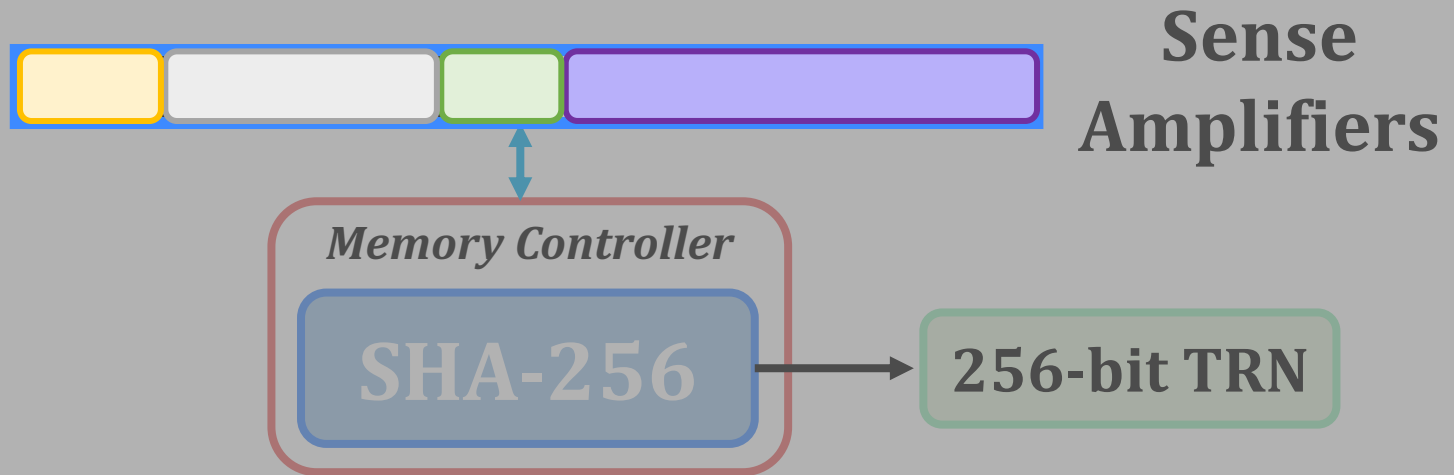
QUAC-TRNG

Key Idea: Leverage **random values** on sense amplifiers generated by **QUAC** operations as **source of entropy**



QUAC-TRNG

Key Idea: Leverage random values on sense amplifiers generated by QUAC operations as source of entropy



Generates a **256-bit random number**
for every **256-bit Shannon Entropy** block

In-DRAM True Random Number Generation

- Ataberk Olgun, Minesh Patel, A. Giray Yaglikci, Haocong Luo, Jeremie S. Kim, F. Nisa Bostanci, Nandita Vijaykumar, Oguz Ergin, and Onur Mutlu,
["QUAC-TRNG: High-Throughput True Random Number Generation Using Quadruple Row Activation in Commodity DRAM Chips"](#)
Proceedings of the [48th International Symposium on Computer Architecture \(ISCA\)](#), Virtual, June 2021.
[[Slides \(pptx\)](#)] [[pdf](#)]
[[Short Talk Slides \(pptx\)](#)] [[pdf](#)]
[[Talk Video](#) (25 minutes)]
[[SAFARI Live Seminar Video](#) (1 hr 26 mins)]

QUAC-TRNG: High-Throughput True Random Number Generation Using Quadruple Row Activation in Commodity DRAM Chips

Ataberk Olgun^{§†}

Minesh Patel[§]

A. Giray Yağlıkçı[§]

Haocong Luo[§]

Jeremie S. Kim[§]

F. Nisa Bostanci^{§†}

Nandita Vijaykumar^{§⊙}

Oğuz Ergin[†]

Onur Mutlu[§]

[§]*ETH Zürich*

[†]*TOBB University of Economics and Technology*

[⊙]*University of Toronto*

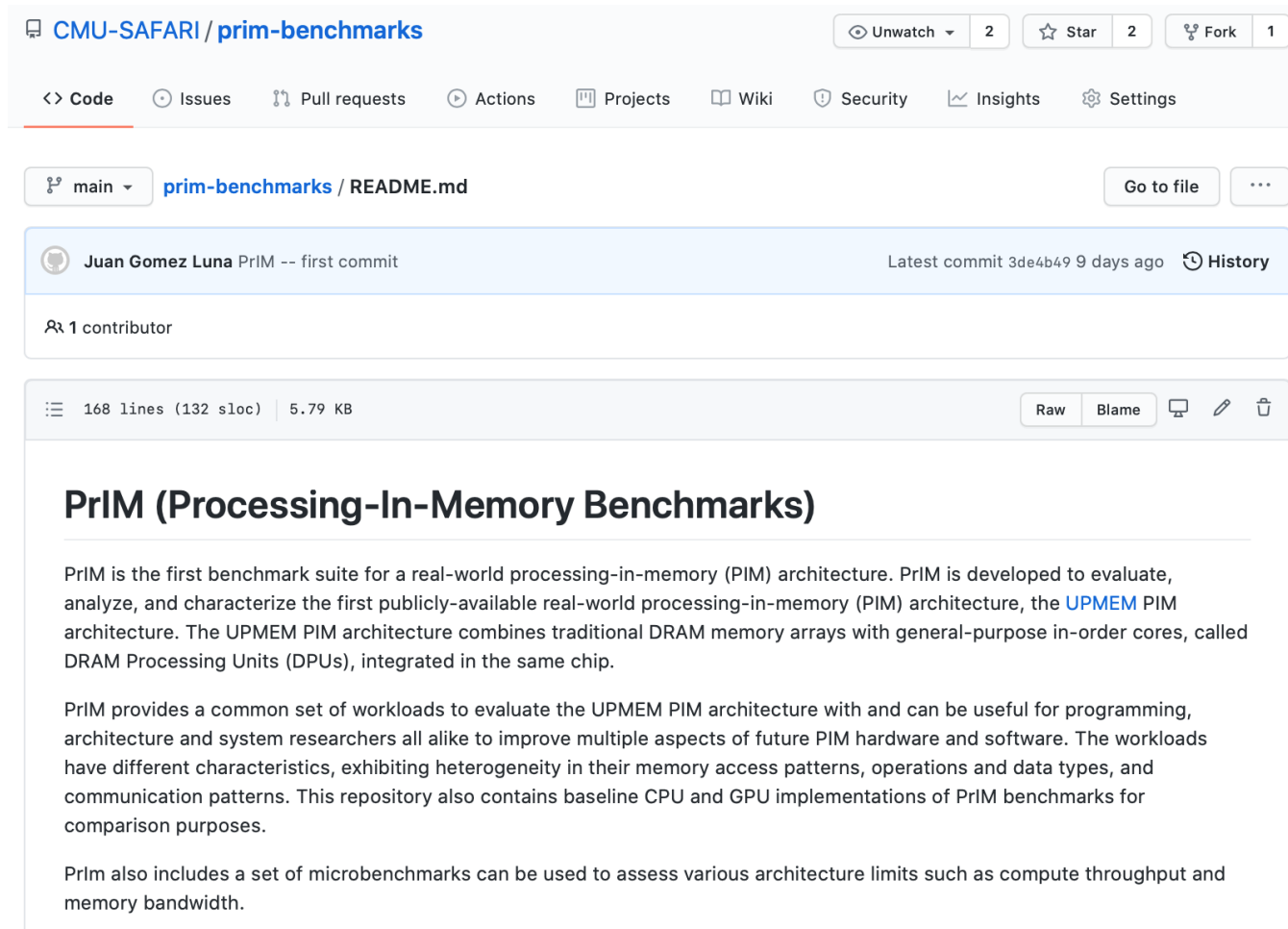
Benchmarks and Simulation Infrastructures

PrIM Benchmarks: Application Domains

Domain	Benchmark	Short name
Dense linear algebra	Vector Addition	VA
	Matrix-Vector Multiply	GEMV
Sparse linear algebra	Sparse Matrix-Vector Multiply	SpMV
Databases	Select	SEL
	Unique	UNI
Data analytics	Binary Search	BS
	Time Series Analysis	TS
Graph processing	Breadth-First Search	BFS
Neural networks	Multilayer Perceptron	MLP
Bioinformatics	Needleman-Wunsch	NW
Image processing	Image histogram (short)	HST-S
	Image histogram (large)	HST-L
Parallel primitives	Reduction	RED
	Prefix sum (scan-scan-add)	SCAN-SSA
	Prefix sum (reduce-scan-scan)	SCAN-RSS
	Matrix transposition	TRNS

PrIM Benchmarks are Open Source

- All microbenchmarks, benchmarks, and scripts
- <https://github.com/CMU-SAFARI/prim-benchmarks>



CMU-SAFARI / prim-benchmarks

Unwatch 2 Star 2 Fork 1

<> Code Issues Pull requests Actions Projects Wiki Security Insights Settings

main prim-benchmarks / README.md Go to file

Juan Gomez Luna Prim -- first commit Latest commit 3de4b49 9 days ago History

1 contributor

168 lines (132 sloc) 5.79 KB Raw Blame

PrIM (Processing-In-Memory Benchmarks)

PrIM is the first benchmark suite for a real-world processing-in-memory (PIM) architecture. PrIM is developed to evaluate, analyze, and characterize the first publicly-available real-world processing-in-memory (PIM) architecture, the [UPMEM PIM](#) architecture. The UPMEM PIM architecture combines traditional DRAM memory arrays with general-purpose in-order cores, called DRAM Processing Units (DPUs), integrated in the same chip.

PrIM provides a common set of workloads to evaluate the UPMEM PIM architecture with and can be useful for programming, architecture and system researchers all alike to improve multiple aspects of future PIM hardware and software. The workloads have different characteristics, exhibiting heterogeneity in their memory access patterns, operations and data types, and communication patterns. This repository also contains baseline CPU and GPU implementations of PrIM benchmarks for comparison purposes.

Prim also includes a set of microbenchmarks can be used to assess various architecture limits such as compute throughput and memory bandwidth.

Lecture on PrIM Benchmarks

Strong Scaling: 1 DPU (VI)

The amount of time spent on CPU-DPU and DPU-CPU transfers is low compared to the time spent on DPU execution

TRNS performs step 1 of the matrix transposition via the CPU-DPU transfer. Using small transfers (8 elements) does not exploit full CPU-DPU bandwidth

KEY OBSERVATION 13
Transferring large data chunks from/to the host CPU is preferred for input data and output results due to higher sustained CPU-DPU/DPU-CPU bandwidths.


Execution Time (ms) vs #tasklets per DPU for TRNS:

#tasklets per DPU	Execution Time (ms)
1	~14000
2	~12000
4	~11000
8	~10500
16	~10000

Processing-in-Memory Course: Lecture 8: Benchmarking and Workload Suitability on PIM - Spring 2022

213 views • Premiered Apr 28, 2022

5 DISLIKE SHARE CLIP SAVE ...

 **Onur Mutlu Lectures**
25.9K subscribers

SUBSCRIBED 

DAMOV Analysis Methodology & Workloads

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

GERALDO F. OLIVEIRA, ETH Zürich, Switzerland

JUAN GÓMEZ-LUNA, ETH Zürich, Switzerland

LOIS OROSA, ETH Zürich, Switzerland

SAUGATA GHOSE, University of Illinois at Urbana-Champaign, USA

NANDITA VIJAYKUMAR, University of Toronto, Canada

IVAN FERNANDEZ, University of Malaga, Spain & ETH Zürich, Switzerland

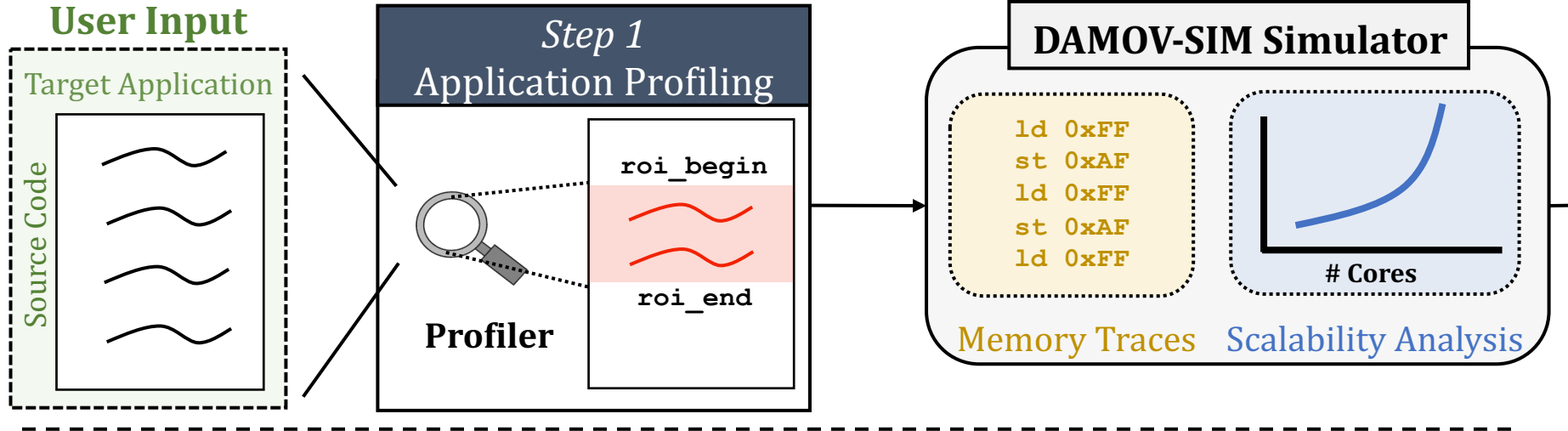
MOHAMMAD SADROSADATI, Institute for Research in Fundamental Sciences (IPM), Iran & ETH Zürich, Switzerland

ONUR MUTLU, ETH Zürich, Switzerland

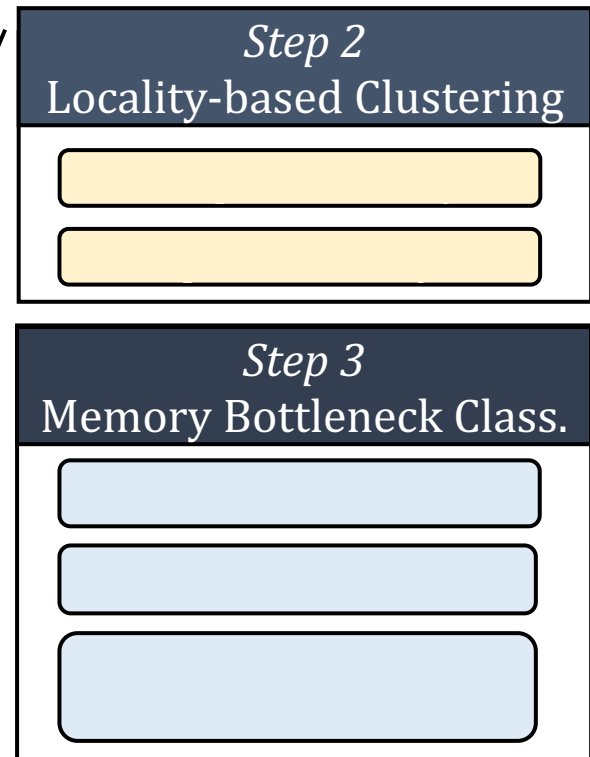
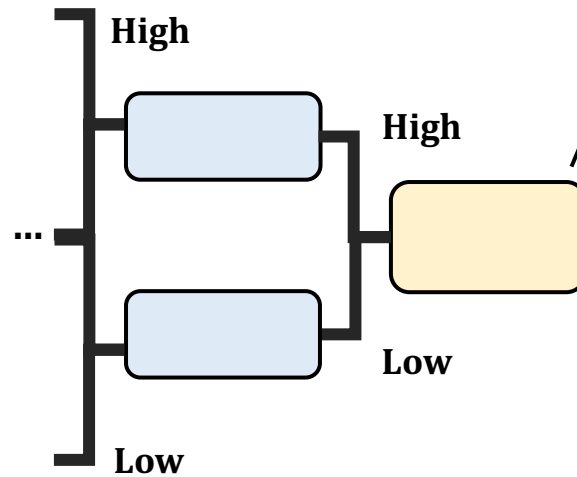
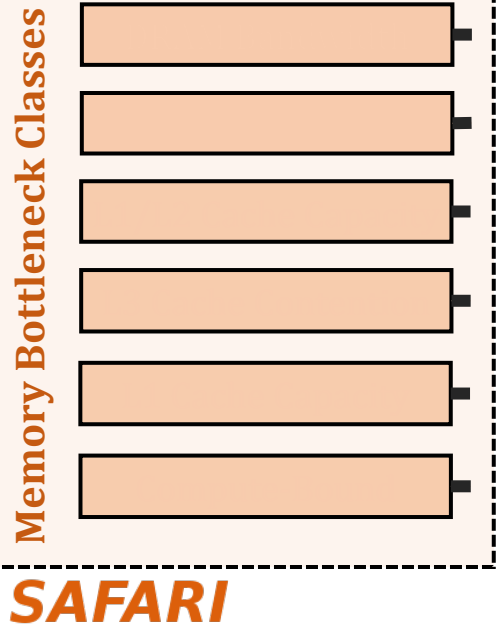
Data movement between the CPU and main memory is a first-order obstacle against improving performance, scalability, and energy efficiency in modern systems. Computer systems employ a range of techniques to reduce overheads tied to data movement, spanning from traditional mechanisms (e.g., deep multi-level cache hierarchies, aggressive hardware prefetchers) to emerging techniques such as Near-Data Processing (NDP), where some computation is moved close to memory. Prior NDP works investigate the root causes of data movement bottlenecks using different profiling methodologies and tools. However, there is still a lack of understanding about the key metrics that can identify different data movement bottlenecks and their relation to traditional and emerging data movement mitigation mechanisms. Our goal is to methodically identify potential sources of data movement over a broad set of applications and to comprehensively compare traditional compute-centric data movement mitigation techniques (e.g., caching and prefetching) to more memory-centric techniques (e.g., NDP), thereby developing a rigorous understanding of the best techniques to mitigate each source of data movement.

With this goal in mind, we perform the first large-scale characterization of a wide variety of applications, across a wide range of application domains, to identify fundamental program properties that lead to data movement to/from main memory. We develop the first systematic methodology to classify applications based on the sources contributing to data movement bottlenecks. From our large-scale characterization of 77K functions across 345 applications, we select 144 functions to form the first open-source benchmark suite (DAMOV) for main memory data movement studies. We select a diverse range of functions that (1) represent different types of data movement bottlenecks, and (2) come from a wide range of application domains. Using NDP as a case study, we identify new insights about the different data movement bottlenecks and use these insights to determine the most suitable data movement mitigation mechanism for a particular application. We open-source DAMOV and the complete source code for our new characterization methodology at <https://github.com/CMU-SAFARI/DAMOV>.

Methodology Overview



Methodology Output



More on DAMOV

- Geraldo F. Oliveira, Juan Gomez-Luna, Lois Orosa, Saugata Ghose, Nandita Vijaykumar, Ivan Fernandez, Mohammad Sadrosadati, and Onur Mutlu, [**"DAMOV: A New Methodology and Benchmark Suite for Evaluating Data Movement Bottlenecks"**](#)

*Preprint in [**arXiv**](#), 8 May 2021.*

[[**arXiv preprint**](#)]

[[**DAMOV Suite and Simulator Source Code**](#)]

[[**SAFARI Live Seminar Video**](#) (2 hrs 40 mins)]

[[**Short Talk Video**](#) (21 minutes)]

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

GERALDO F. OLIVEIRA, ETH Zürich, Switzerland

JUAN GÓMEZ-LUNA, ETH Zürich, Switzerland

LOIS OROSA, ETH Zürich, Switzerland

SAUGATA GHOSE, University of Illinois at Urbana–Champaign, USA

NANDITA VIJAYKUMAR, University of Toronto, Canada

IVAN FERNANDEZ, University of Malaga, Spain & ETH Zürich, Switzerland

MOHAMMAD SADROSADATI, ETH Zürich, Switzerland

ONUR MUTLU, ETH Zürich, Switzerland

Lecture on DAMOV

Step 3: Memory Bottleneck Classification (2/2)

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

DAMOV-SIM Simulator
Scalability Analysis
Cores
Integrated ZSim and Ramulator

Configuration 1: Host CPU System
CPU, L1, L2, L3, Off-chip link, DRAM

Configuration 2: NDP System
Off-chip link, CPU, L1, Logic Layer, DRAM

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

24:08 / 1:00:03
DAMOV-SIM: <https://github.com/CMU-SAFARI/DAMOV-SIM>

Processing-in-Memory Course: Lecture 5: How to Evaluate Data Movement Bottlenecks - Spring 2022

346 views • Streamed live on Apr 7, 2022

12 DISLIKE SHARE CLIP SAVE ...



Onur Mutlu Lectures

25.9K subscribers

SUBSCRIBED



Simulation Infrastructures for PIM

- **Ramulator** extended for PIM
 - Flexible and extensible DRAM simulator
 - Can model many different memory standards and proposals
 - Kim+, “**Ramulator: A Flexible and Extensible DRAM Simulator**”, IEEE CAL 2015.
 - <https://github.com/CMU-SAFARI/ramulator-pim>
 - <https://github.com/CMU-SAFARI/ramulator>
 - [[Source Code for Ramulator-PIM](#)]

Ramulator: A Fast and Extensible DRAM Simulator

Yoongu Kim¹ Weikun Yang^{1,2} Onur Mutlu¹
¹Carnegie Mellon University ²Peking University

Simulation Infrastructures for PIM (in SSDs)

- Arash Tavakkol, Juan Gomez-Luna, Mohammad Sadrosadati, Saugata Ghose, and Onur Mutlu,
"MQSim: A Framework for Enabling Realistic Studies of Modern Multi-Queue SSD Devices"
Proceedings of the 16th USENIX Conference on File and Storage Technologies (FAST), Oakland, CA, USA, February 2018.
[Slides (pptx)] [pdf]
[Source Code]

MQSim: A Framework for Enabling Realistic Studies of Modern Multi-Queue SSD Devices

Arash Tavakkol[†], Juan Gómez-Luna[†], Mohammad Sadrosadati[†], Saugata Ghose[‡], Onur Mutlu^{†‡}
[†]*ETH Zürich* [‡]*Carnegie Mellon University*



Funded by the Horizon 2020 Framework
Programme of the European Union
MSCA-ITN-EID

NAPEL: Near-Memory Computing Application Performance Prediction via Ensemble Learning

Gagandeep Singh, Juan Gomez-Luna, Giovanni Mariani, Geraldo F. Oliveira,
Stefano Corda, Sander Stuijk, Onur Mutlu, Henk Corporaal

56th Design Automation Conference (DAC), Las Vegas
4th-June-2019



Executive Summary

- **Motivation:** A promising paradigm to alleviate **data movement bottleneck** is *near-memory computing (NMC)*, which consists of placing compute units close to the memory subsystem
- **Problem:** Simulation times are extremely slow, imposing long run-time especially in the early-stage design space exploration
- **Goal:** A quick high-level performance and energy estimation framework for NMC architectures
- **Our contribution: NAPEL**
 - Fast and accurate performance and energy prediction for previously-unseen applications using ensemble learning
 - Use intelligent statistical techniques and micro-architecture-independent application features to minimize experimental runs
- **Evaluation**
 - NAPEL is, on average, 220x faster than state-of-the-art NMC simulator
 - Error rates (average) of 8.5% and 11.5% for performance and energy estimation

We open source Ramulator-PIM: <https://github.com/CMU-SAFARI/ramulator-pim/>

NMC Simulators

- Simulation for:
 - Design space exploration (DSE)
 - Workload suitability analysis
- NMC Simulators:
 - Sinuca, 2015
 - HMC-SIM, 2016
 - CasHMC, 2016
 - Smart Memory Cube (SMC), 2016
 - CLAPPS, 2017
 - Gem5+HMC, 2017
 - Ramulator-PIM¹, 2019

¹Ramulator-PIM: <https://github.com/CMU-SAFARI/ramulator-pim/>

NMC Simulators

- Simulation for:
 - Design space exploration (DSE)
 - Workload suitability analysis
- NMC Simulators:

Simulation of real workloads can be 10000x slower than native-execution!!!

- Gem5+HMC, 2017
- Ramulator-PIM¹, 2019

¹Ramulator-PIM: <https://github.com/CMU-SAFARI/ramulator-pim/>

NMC Simulators

- Simulation for:
 - Design space exploration (DSE)
 - Workload suitability analysis
- NMC Simulators:

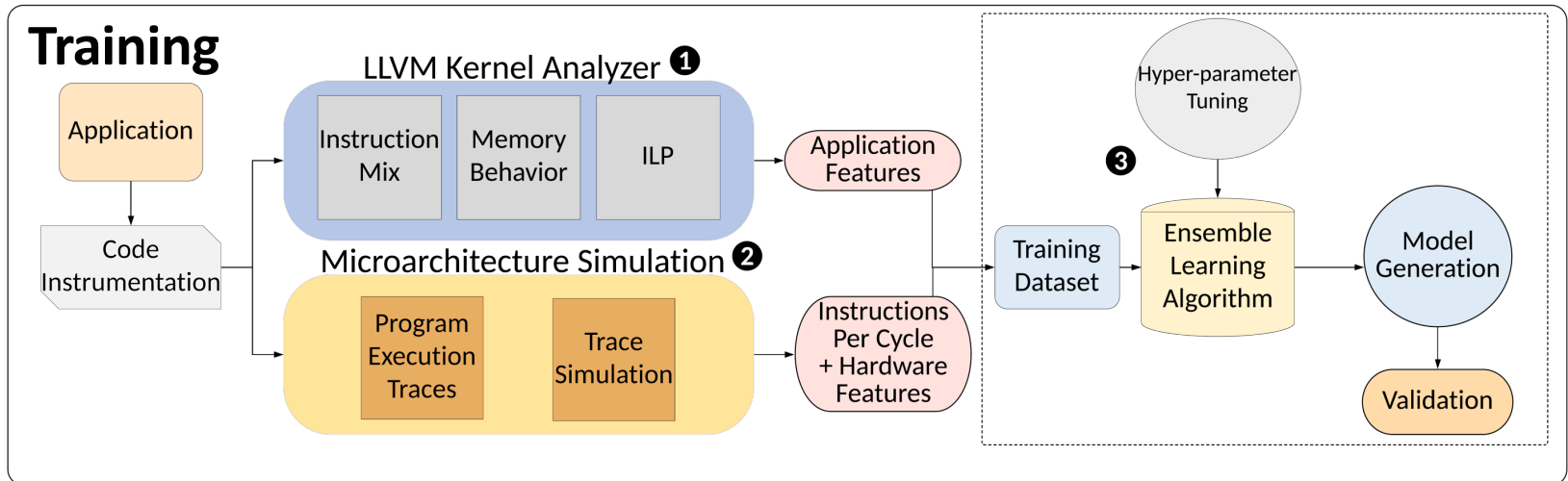
Idea: Leverage ML with statistical techniques for quick NMC performance/energy prediction

- Gem5+HMC, 2017
- Ramulator-PIM¹, 2019

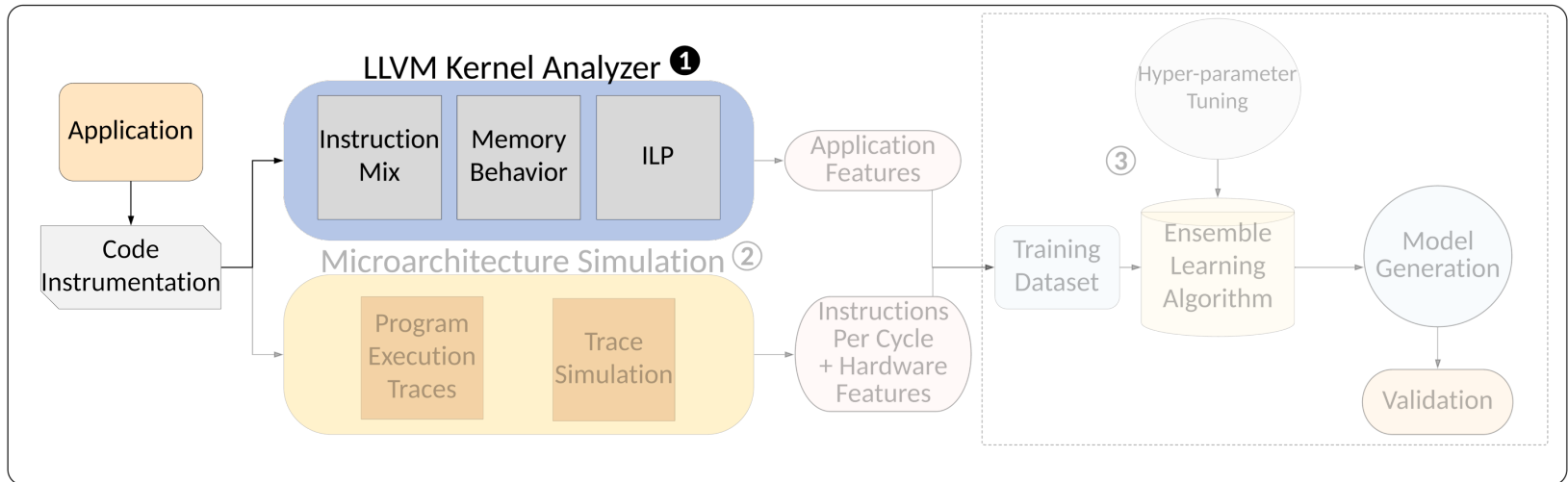
¹Ramulator-PIM: <https://github.com/CMU-SAFARI/ramulator-pim/>

NAPEL: Near-Memory Computing Application Performance Prediction via Ensemble Learning

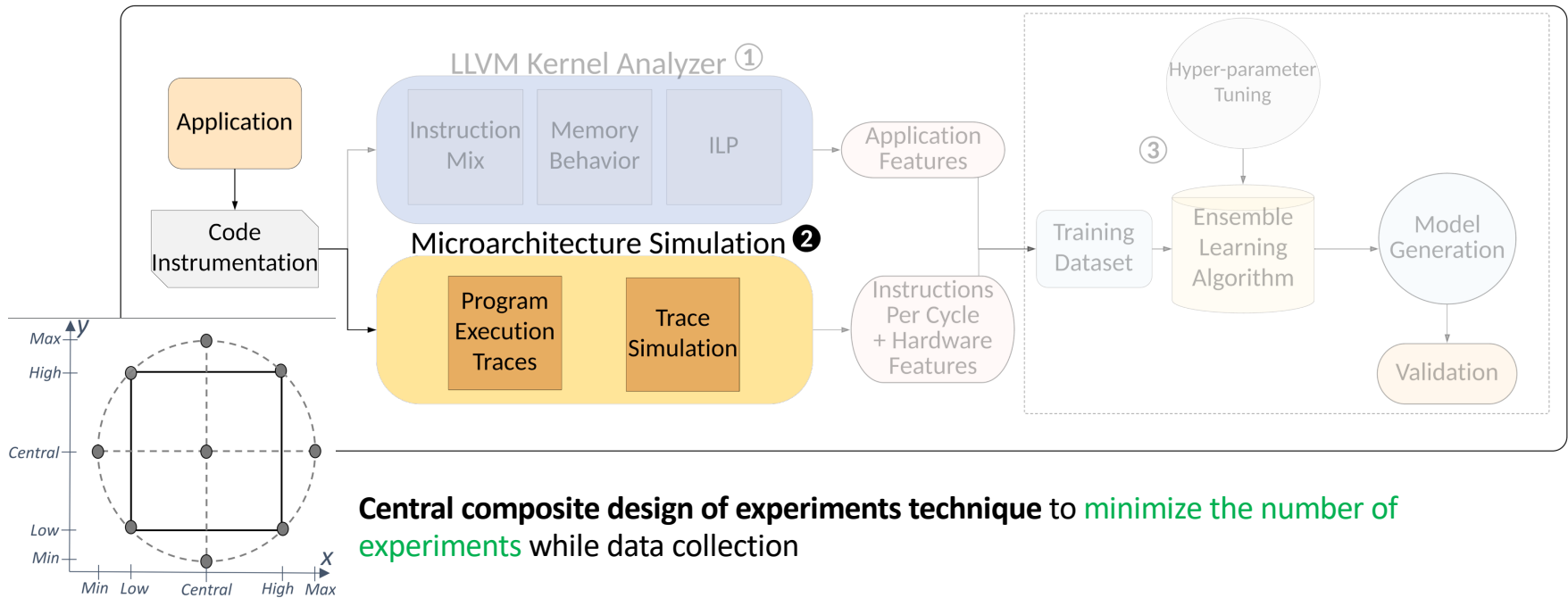
NAPEL Model



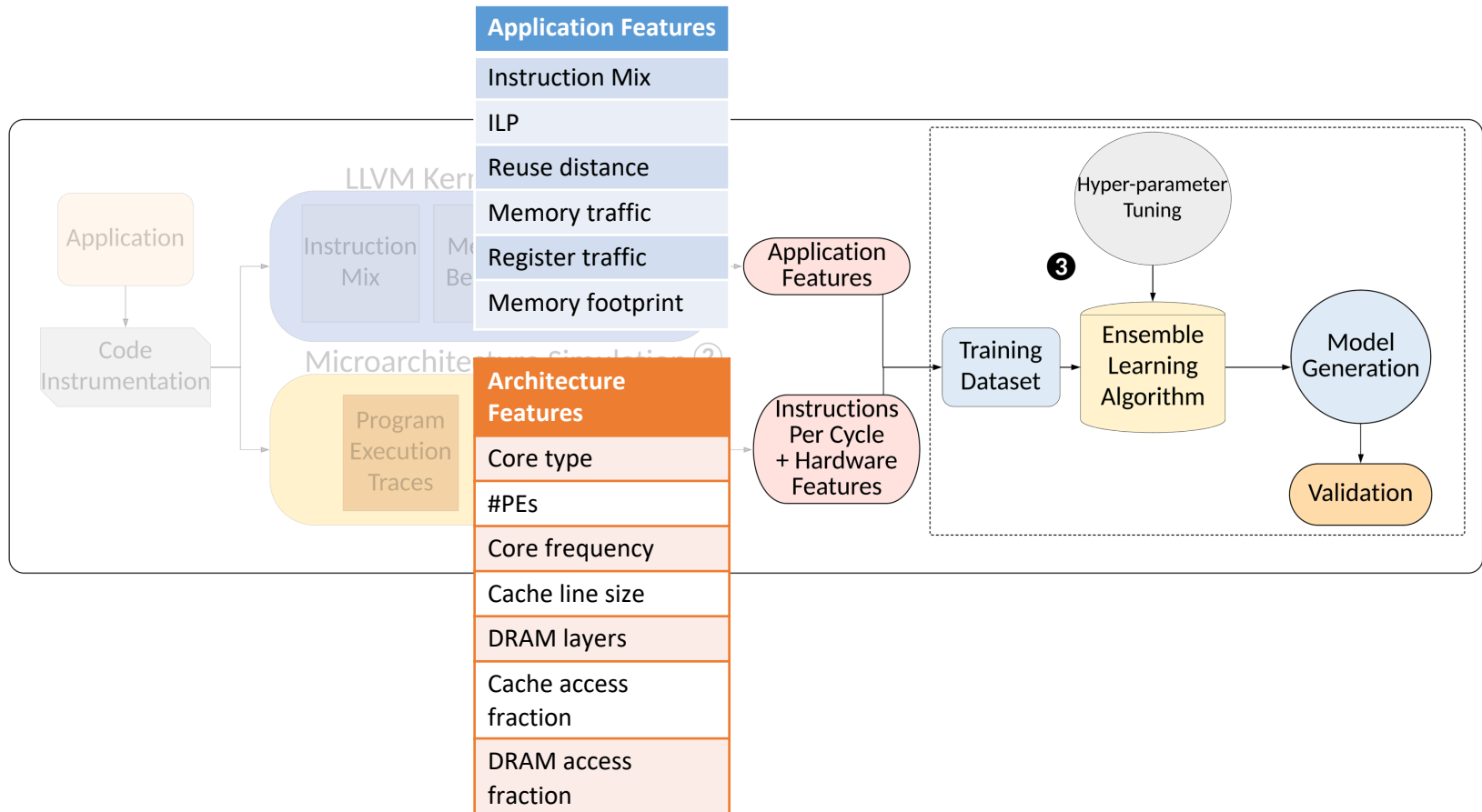
Phase 1: LLVM Analyzer



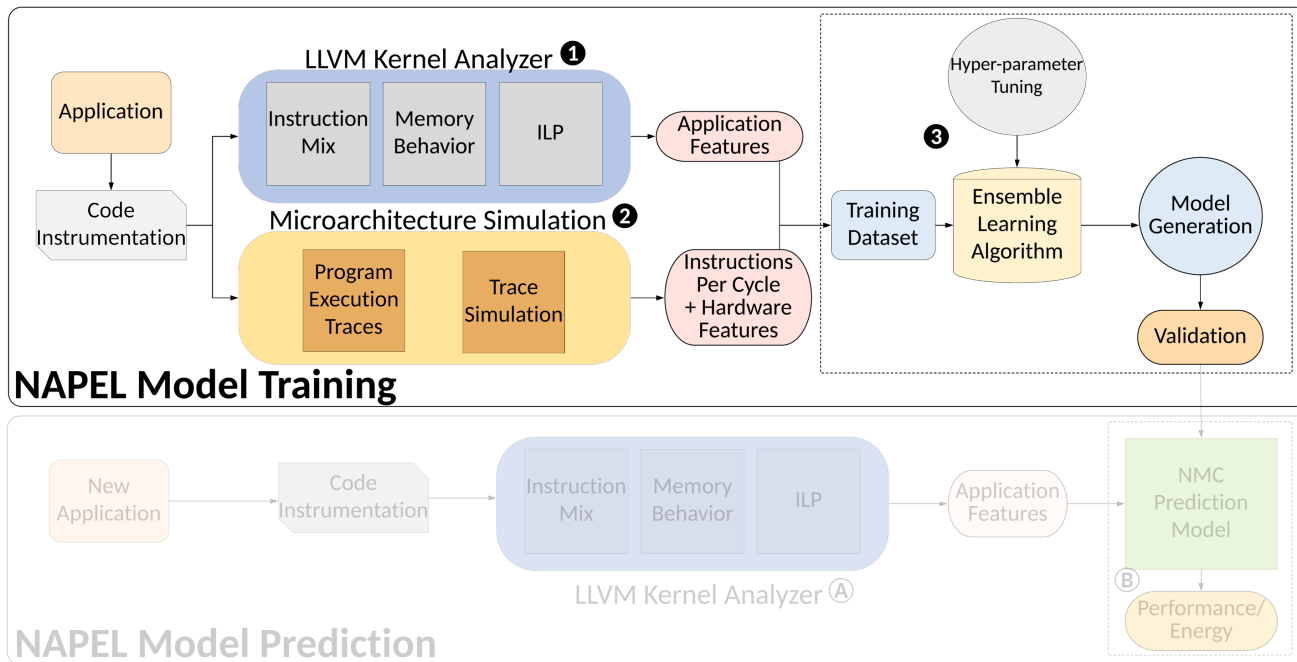
Phase 2: Microarchitecture Simulation



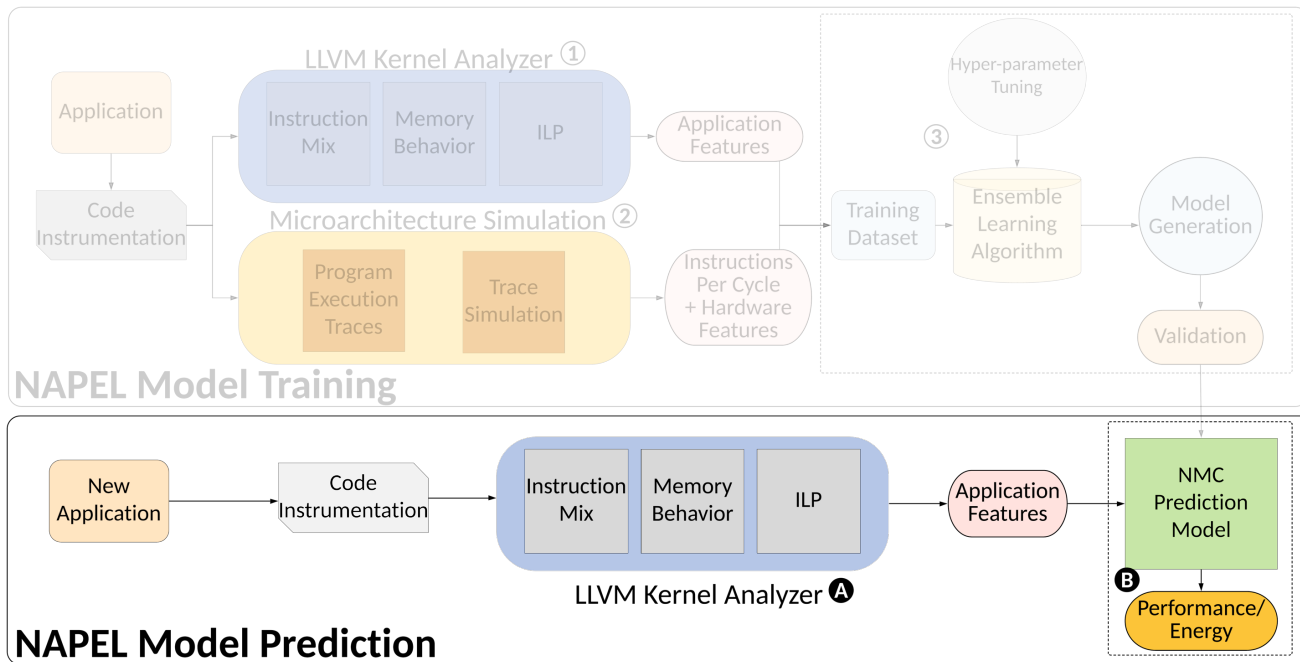
Phase 3: Ensemble ML Training



NAPEL Framework

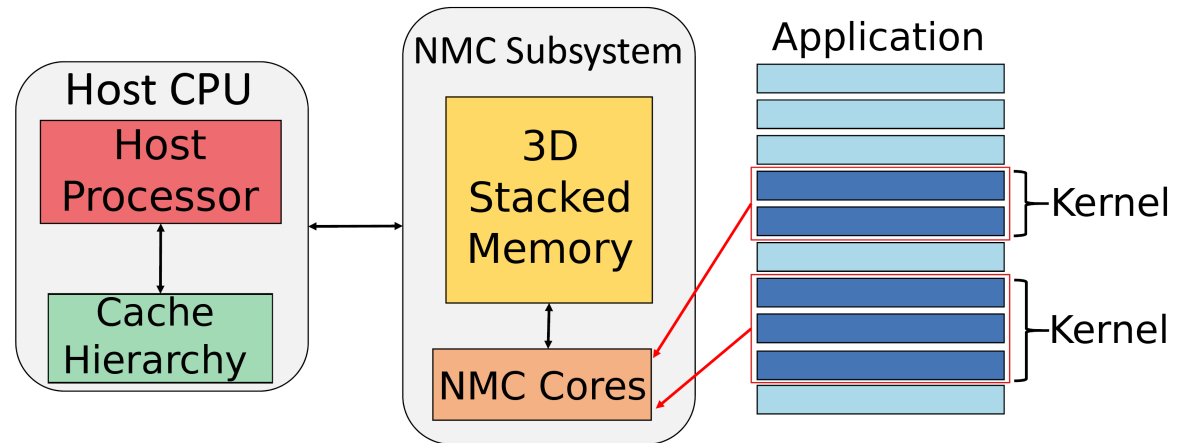


NAPEL Prediction



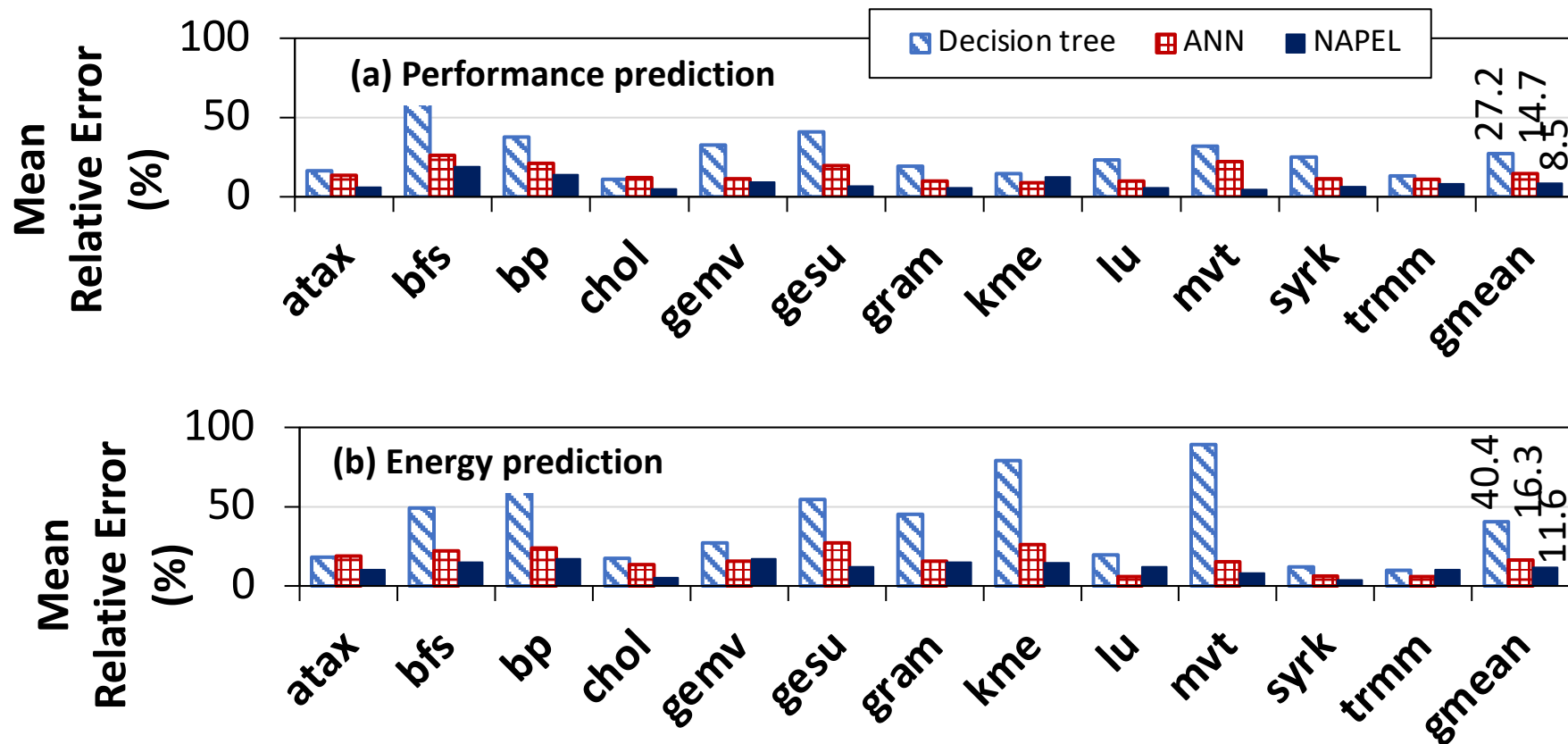
Experimental Setup

- Host System
 - **IBM POWER9**
 - Power: AMESTER
- NMC Subsystem
 - **Ramulator-PIM¹**
- Workloads
 - **PolyBench** and **Rodinia**
 - Heterogeneous workloads such as image processing, machine learning, graph processing etc.
- Accuracy in terms of mean relative error (MRE)



¹<https://github.com/CMU-SAFARI/ramulator-pim/>

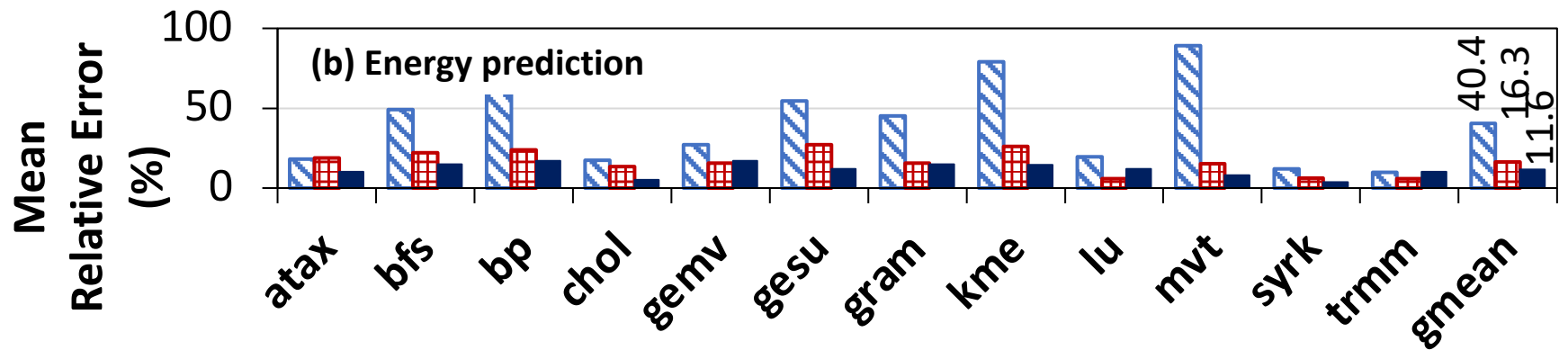
NAPEL Accuracy: Performance and Energy Estimates



NAPEL Accuracy: Performance and Energy Estimates

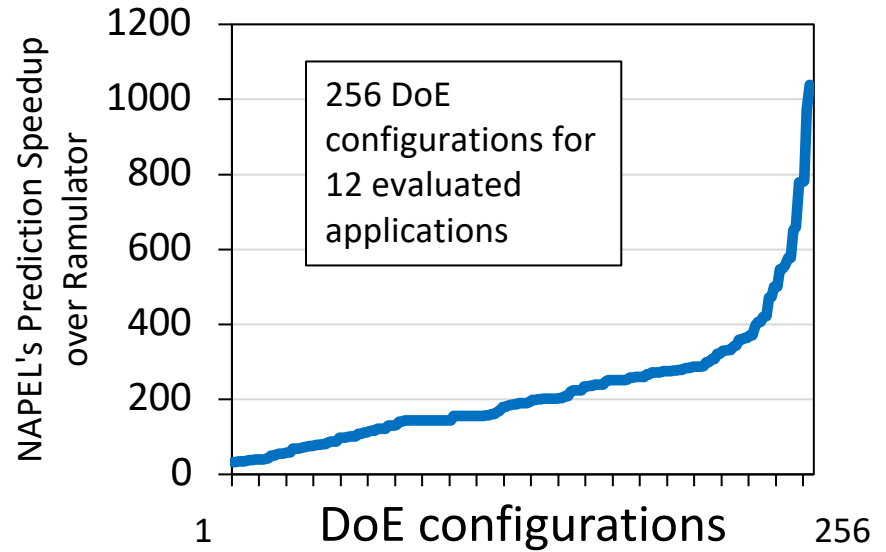


MRE of 8.5% and 11.6% for performance and energy



Speed of Evaluation

Application Name	Training/Prediction Time			
	#DoE conf.	DoE run (mins)	Train+Tune (mins)	Pred. (mins)
atax	11	522	34.9	0.49
bfs	31	1084	34.2	0.48
bp	31	1073	43.8	0.47
chol	19	741	34.9	0.49
gemv	19	741	24.4	0.51
gesu	19	731	36.1	0.51
gram	19	773	36.5	0.52
kme	31	742	36.9	0.55
lu	19	633	37.9	0.51
mvt	19	955	38.0	0.54
syrk	19	928	35.7	0.51
trmm	19	898	37.6	0.48

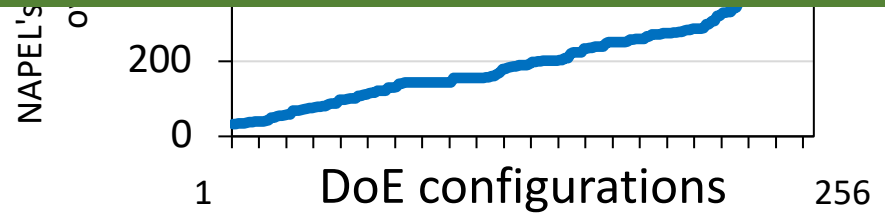
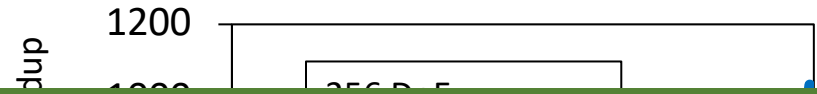


Speed of Evaluation

Application	Training/Prediction Time			
-------------	--------------------------	--	--	--

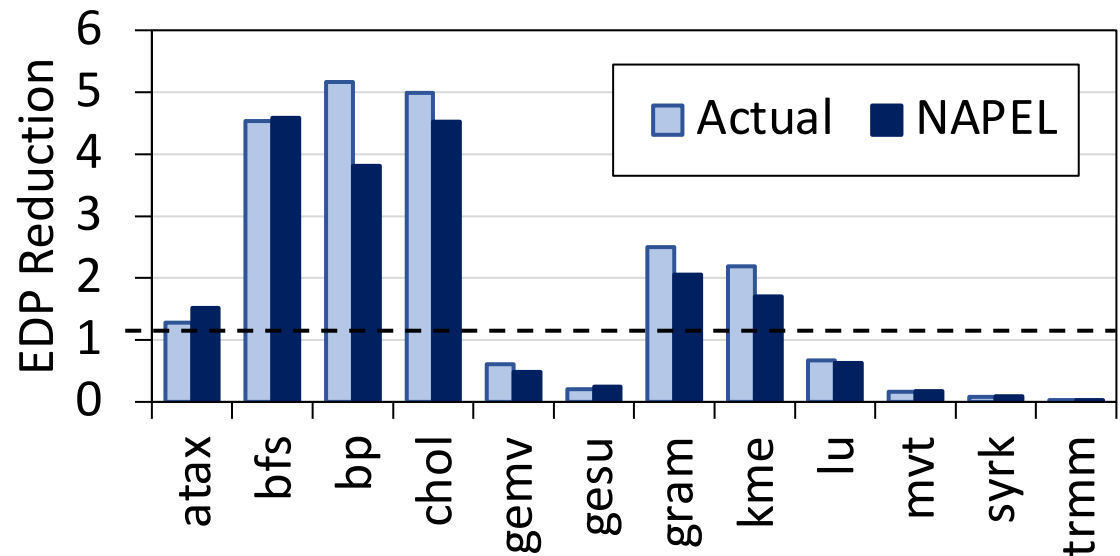
220x (up to 1039x) faster than NMC simulator

kme	31	742	36.9	0.55
lu	19	633	37.9	0.51
mvt	19	955	38.0	0.54
syrk	19	928	35.7	0.51
trmm	19	898	37.6	0.48



Use Case: NMC Suitability Analysis

- Assess the potential of offloading a workload to NMC
- **NAPEL provides accurate prediction of NMC suitability**
- MRE between 1.3% to 26.3% (average 14.1%)



Performance & Energy Models for PIM

- Gagandeep Singh, Juan Gomez-Luna, Giovanni Mariani, Geraldo F. Oliveira, Stefano Corda, Sander Stuijk, [Onur Mutlu](#), and Henk Corporaal, **"NAPEL: Near-Memory Computing Application Performance Prediction via Ensemble Learning"**
Proceedings of the [56th Design Automation Conference \(DAC\)](#), Las Vegas, NV, USA, June 2019.
[[Slides \(pptx\) \(pdf\)](#)]
[[Poster \(pptx\) \(pdf\)](#)]
[[Source Code for Ramulator-PIM](#)]

NAPEL: Near-Memory Computing Application Performance Prediction via Ensemble Learning

Gagandeep Singh^{a,c} Juan Gómez-Luna^b Giovanni Mariani^c Geraldo F. Oliveira^b
Stefano Corda^{a,c} Sander Stuijk^a Onur Mutlu^b Henk Corporaal^a
^aEindhoven University of Technology ^bETH Zürich ^cIBM Research - Zurich

Applications that Benefit from PIM

New Applications and Use Cases for PIM

- Jeremie S. Kim, Damla Senol Cali, Hongyi Xin, Donghyuk Lee, Saugata Ghose, Mohammed Alser, Hasan Hassan, Oguz Ergin, Can Alkan, and Onur Mutlu, "**GRIM-Filter: Fast Seed Location Filtering in DNA Read Mapping Using Processing-in-Memory Technologies**" *BMC Genomics*, 2018.
Proceedings of the 16th Asia Pacific Bioinformatics Conference (APBC), Yokohama, Japan, January 2018.
[arxiv.org Version \(pdf\)](#)

GRIM-Filter: Fast seed location filtering in DNA read mapping using processing-in-memory technologies

Jeremie S. Kim^{1,6*}, Damla Senol Cali¹, Hongyi Xin², Donghyuk Lee³, Saugata Ghose¹, Mohammed Alser⁴, Hasan Hassan⁶, Oguz Ergin⁵, Can Alkan^{4*} and Onur Mutlu^{6,1*}

From The Sixteenth Asia Pacific Bioinformatics Conference 2018
Yokohama, Japan. 15-17 January 2018

Genome Read In-Memory (GRIM) Filter: Fast Seed Location Filtering in DNA Read Mapping using Processing-in-Memory Technologies

Jeremie Kim,

Damla Senol, Hongyi Xin, Donghyuk Lee,
Saugata Ghose, Mohammed Alser, Hasan Hassan,
Oguz Ergin, Can Alkan, and Onur Mutlu

Carnegie Mellon



ETH zürich

Executive Summary

- **Genome Read Mapping** is a very important problem and is the first step in many types of genomic analysis
 - Could lead to improved health care, medicine, quality of life
- Read mapping is an **approximate string matching** problem
 - Find the best fit of 100 character strings into a 3 billion character dictionary
 - **Alignment** is currently the best method for determining the similarity between two strings, but is **very expensive**
- We propose an in-memory processing algorithm **GRIM-Filter** for accelerating read mapping, by reducing the number of required alignments
- We implement GRIM-Filter using **in-memory processing** within **3D-stacked memory** and show up to **3.7x speedup**.

Accelerating Approximate String Matching

- Damla Senol Cali, Gurpreet S. Kalsi, Zülal Bingöl, Can Firtina, Lavanya Subramanian, Jeremie S. Kim, Rachata Ausavarungnirun, Mohammed Alser, Juan Gomez-Luna, Amirali Boroumand, Anant Nori, Allison Scibisz, Sreenivas Subramoney, Can Alkan, Saugata Ghose, and Onur Mutlu, **["GenASM: A High-Performance, Low-Power Approximate String Matching Acceleration Framework for Genome Sequence Analysis"](#)**

Proceedings of the [53rd International Symposium on Microarchitecture \(MICRO\)](#), Virtual, October 2020.

[[Lightning Talk Video](#) (1.5 minutes)]

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

[[Talk Video](#) (18 minutes)]

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

GenASM: A High-Performance, Low-Power Approximate String Matching Acceleration Framework for Genome Sequence Analysis

Damla Senol Cali[†]✕ Gurpreet S. Kalsi[✕] Zülal Bingöl[∇] Can Firtina[◇] Lavanya Subramanian[‡] Jeremie S. Kim[◇]†
Rachata Ausavarungnirun[○] Mohammed Alser[◇] Juan Gomez-Luna[◇] Amirali Boroumand[†] Anant Nori[✕]
Allison Scibisz[†] Sreenivas Subramoney[✕] Can Alkan[∇] Saugata Ghose^{*†} Onur Mutlu[◇]†∇

[†]Carnegie Mellon University [✕]Processor Architecture Research Lab, Intel Labs [∇]Bilkent University [◇]ETH Zürich
[‡]Facebook [○]King Mongkut's University of Technology North Bangkok ^{*}University of Illinois at Urbana-Champaign

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

Accelerating Climate Modeling

- Gagandeep Singh, Dionysios Diamantopoulos, Christoph Hagleitner, Juan Gómez-Luna, Sander Stuijk, Onur Mutlu, and Henk Corporaal,

"NERO: A Near High-Bandwidth Memory Stencil Accelerator for Weather Prediction Modeling"

Proceedings of the 30th International Conference on Field-Programmable Logic and Applications (FPL), Gothenburg, Sweden, September 2020.

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

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

[[Talk Video](#) (23 minutes)]

Nominated for the Stamatis Vassiliadis Memorial Award.

NERO: A Near High-Bandwidth Memory Stencil Accelerator for Weather Prediction Modeling

Gagandeep Singh^{a,b,c}

Dionysios Diamantopoulos^c

Christoph Hagleitner^c

Juan Gómez-Luna^b

Sander Stuijk^a

Onur Mutlu^b

Henk Corporaal^a

^aEindhoven University of Technology

^bETH Zürich

^cIBM Research Europe, Zurich

Accelerating Time Series Analysis

- Ivan Fernandez, Ricardo Quisiant, Christina Giannoula, Mohammed Alser, Juan Gómez-Luna, Eladio Gutiérrez, Oscar Plata, and Onur Mutlu, ["NATSA: A Near-Data Processing Accelerator for Time Series Analysis"](#) *Proceedings of the 38th IEEE International Conference on Computer Design (ICCD)*, Virtual, October 2020.

NATSA: A Near-Data Processing Accelerator for Time Series Analysis

Ivan Fernandez[§]

Ricardo Quisiant[§]

Christina Giannoula[†]

Mohammed Alser[‡]

Juan Gómez-Luna[‡]

Eladio Gutiérrez[§]

Oscar Plata[§]

Onur Mutlu[‡]

[§]*University of Malaga*

[†]*National Technical University of Athens*

[‡]*ETH Zürich*

Epilogue

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.*

1 Introduction	2
2 Major Trends Affecting Main Memory	4
3 The Need for Intelligent Memory Controllers to Enhance Memory Scaling	6
4 Perils of Processor-Centric Design	9
5 Processing-in-Memory (PIM): Technology Enablers and Two Approaches	12
5.1 New Technology Enablers: 3D-Stacked Memory and Non-Volatile Memory . . .	12
5.2 Two Approaches: Processing Using Memory (PUM) vs. Processing Near Memory (PNM)	13
6 Processing Using Memory (PUM)	14
6.1 RowClone	14
6.2 Ambit	15
6.3 Gather-Scatter DRAM	17
6.4 In-DRAM Security Primitives	17
7 Processing Near Memory (PNM)	18
7.1 Tesseract: Coarse-Grained Application-Level PNM Acceleration of Graph Processing	19
7.2 Function-Level PNM Acceleration of Mobile Consumer Workloads	20
7.3 Programmer-Transparent Function-Level PNM Acceleration of GPU Applications	21
7.4 Instruction-Level PNM Acceleration with PIM-Enabled Instructions (PEI) . .	21
7.5 Function-Level PNM Acceleration of Genome Analysis Workloads	22
7.6 Application-Level PNM Acceleration of Time Series Analysis	23
8 Enabling the Adoption of PIM	24
8.1 Programming Models and Code Generation for PIM	24
8.2 PIM Runtime: Scheduling and Data Mapping	25
8.3 Memory Coherence	27
8.4 Virtual Memory Support	27
8.5 Data Structures for PIM	28
8.6 Benchmarks and Simulation Infrastructures	29
8.7 Real PIM Hardware Systems and Prototypes	30
8.8 Security Considerations	30
9 Conclusion and Future Outlook	31

Main memory, built using the Dynamic Random Access Memory (DRAM) technology, is a major component in nearly all computing systems, including servers, cloud platforms, mobile/embedded devices, and sensor systems. Across all of these systems, the data working set sizes of modern applications are rapidly growing, while the need for fast analysis of such data is increasing. Thus, main memory is becoming an increasingly significant bottleneck across a wide variety of computing systems and applications [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]. Alleviating the main memory bottleneck requires the memory capacity, energy, cost, and performance to all scale in an efficient manner across technology generations. Unfortunately, it has become increasingly difficult in recent years, especially the past decade, to scale all of these dimensions [1, 2, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49], and thus the main memory bottleneck has been worsening.

A major reason for the main memory bottleneck is the high energy and latency cost associated with *data movement*. In modern computers, to perform any operation on data that resides in main memory, the processor must retrieve the data from main memory. This requires the memory controller to issue commands to a DRAM module across a relatively slow and power-hungry off-chip bus (known as the *memory channel*). The DRAM module sends the requested data across the memory channel, after which the data is placed in the caches and registers. The CPU can perform computation on the data once the data is in its registers. Data movement from the DRAM to the CPU incurs long latency and consumes a significant amount of energy [7, 50, 51, 52, 53, 54]. These costs are often exacerbated by the fact that much of the data brought into the caches is *not reused* by the CPU [52, 53, 55, 56], providing little benefit in return for the high latency and energy cost.

The cost of data movement is a fundamental issue with the *processor-centric* nature of contemporary computer systems. The CPU is considered to be the master in the system, and computation is performed only in the processor (and accelerators). In contrast, data storage and communication units, including the main memory, are treated as unintelligent workers that are incapable of computation. As a result of this processor-centric design paradigm, data moves a lot in the system between the computation units and communication/ storage units so that computation can be done on it. With the increasingly *data-centric* nature of contemporary and emerging appli-

PIM Review and Open Problems (II)

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.
[\[arXiv version\]](#)

PIM Review and Open Problems (III)

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

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

[†]*Carnegie Mellon University*

[§]*ETH Zürich*

Saugata Ghose, Amirali Boroumand, Jeremie S. Kim, Juan Gomez-Luna, and Onur Mutlu,

"Processing-in-Memory: A Workload-Driven Perspective"

Invited Article in IBM Journal of Research & Development, Special Issue on Hardware for Artificial Intelligence, to appear in November 2019.

[Preliminary arXiv version]

Fundamentally Energy-Efficient (Data-Centric)

Computing Architectures

Fundamentally High-Performance (Data-Centric) Computing Architectures

Computing Architectures with Minimal Data Movement

A Tutorial on Memory-Centric Systems

- Onur Mutlu,

"Memory-Centric Computing Systems"

Invited Tutorial at *66th International Electron Devices Meeting (IEDM)*, Virtual, 12 December 2020.

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

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

[[Tutorial Video](#) (1 hour 51 minutes)]

[[Executive Summary Video](#) (2 minutes)]

[[Abstract and Bio](#)]

[[Related Keynote Paper from VLSI-DAT 2020](#)]

[[Related Review Paper on Processing in Memory](#)]

<https://www.youtube.com/watch?v=H3sEaINPBOE>

Memory-Centric Computing Systems



Onur Mutlu

omutlu@gmail.com

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

12 December 2020

IEDM Tutorial

SAFARI

ETH zürich

Carnegie Mellon



0:02 / 1:51:05



IEDM 2020 Tutorial: Memory-Centric Computing Systems, Onur Mutlu, 12 December 2020

1,862 views • Dec 23, 2020

55 likes 0 comments SHARE SAVE ...

ANALYTICS

EDIT VIDEO

Onur Mutlu Lectures
15.2K subscribers


Speaker: Professor Onur Mutlu (<https://people.inf.ethz.ch/omutlu/>)

Date: December 12, 2020

Abstract and Bio: <https://iee-iedm.org/wp-content/uplo...>

Real PIM Tutorial (ISCA 2023)

- Lectures + Hands-on labs + Invited lectures



ISCA 2023 Real-World PIM Tutorial

Search

[Recent Changes](#) [Media Manager](#) [Sitemap](#)

Trace: • [start](#)

Real-world Processing-in-Memory Systems for Modern Workloads

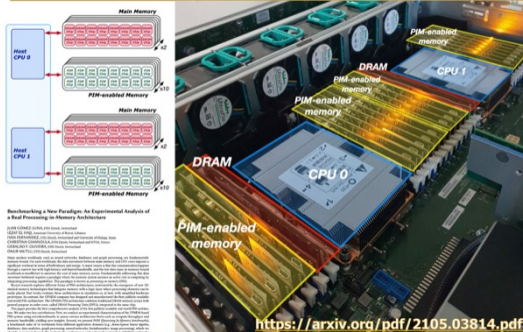
Tutorial Description

Processing-in-Memory (PIM) is a computing paradigm that aims at overcoming the data movement bottleneck (i.e., the waste of execution cycles and energy resulting from the back-and-forth data movement between memory units and compute units) by making memory compute-capable.

Explored over several decades since the 1960s, PIM systems are becoming a reality with the advent of the first commercial products and prototypes.

A number of startups (e.g., UPMEM, Neuroblade) are already commercializing real PIM hardware, each with its own design approach and target applications. Several major vendors (e.g., Samsung, SK Hynix, Alibaba) have presented real PIM chip prototypes in the last two years. Most of these architectures have in common that they place compute units near the memory arrays. This type of PIM is called processing near memory (PNM).

2,560-DPU Processing-in-Memory System



<https://arxiv.org/pdf/2105.03814.pdf>

PIM can provide large improvements in both performance and energy consumption for many modern applications, thereby enabling a commercially viable way of dealing with huge amounts of data that is bottlenecking our computing systems. Yet, it is critical to (1) study and understand the characteristics that make a workload suitable for a PIM architecture, (2) propose optimization strategies for PIM kernels, and (3) develop programming frameworks and tools that can lower the learning curve and ease the adoption of PIM.

This tutorial focuses on the latest advances in PIM technology, workload characterization for PIM, and programming and optimizing PIM kernels. We will (1) provide an introduction to PIM and taxonomy of PIM systems, (2) give an overview and a rigorous analysis of existing real-world PIM hardware, (3) conduct hand-on labs about important workloads (machine learning, sparse linear algebra, bioinformatics, etc.) using real PIM systems, and (4) shed light on how to improve future PIM systems for such workloads.

Table of Contents

- Real-world Processing-in-Memory Systems for Modern Workloads
 - Tutorial Description
 - Organizers
 - Agenda (June 18, 2023)
 - Lectures (tentative)
 - Hands-on Labs (tentative)
 - Learning Materials

Processing-in-Memory Course (Fall 2022)

- Short weekly lectures
- Hands-on projects

Livestream - P&S Data-Centric Architectures: Fundamentally...
Onur Mutlu Lectures
14 videos 631 views Updated 3 days ago
Play all Shuffle

1 PIM Course: Lecture 1: Data-Centric Architectures: Improving Performance & Energy - Fall 2022
Onur Mutlu Lectures • 1K views • 3 months ago 1:34:40

2 PIM Course: Lecture 2: How to Evaluate Data Movement Bottlenecks - Fall 2022
Onur Mutlu Lectures • 678 views • 2 months ago 59:51

3 PIM Course: Lecture 3: Real-world PIM: UPMEM PIM - Fall 2022
Onur Mutlu Lectures • 455 views • 2 months ago 31:46

4 PIM Course: Lecture 4: Real-world PIM: Microbenchmarking of UPMEM PIM - Fall 2022
Onur Mutlu Lectures • 275 views • 2 months ago 56:05

5 PIM Course: Lecture 5: Real-world PIM: Samsung HBM-PIM - Fall 2022
Onur Mutlu Lectures • 725 views • 2 months ago 1:06:09

6 PIM Course: Lecture 6: Real-world PIM: SK Hynix AiM - Fall 2022
Onur Mutlu Lectures • 1K views • 2 months ago 41:33

7 PIM Course: Lecture 7: Real-world PIM: Samsung AxDIMM - Fall 2022
Onur Mutlu Lectures • 767 views • 1 month ago 32:43

8 PIM Course: Lecture 8: Real-world PIM: Alibaba HB-PNM - Fall 2022
Onur Mutlu Lectures • 383 views • 1 month ago 41:44

9 PIM Course: Lecture 9: Programming PIM Architectures - Fall 2022
Onur Mutlu Lectures • 367 views • 1 month ago 46:43

SAFARI Project & Seminars Courses (Fall 2022)

Trace: heterogeneous_systems • processing_in_memory

Home

Courses

- SoftMC
- Ramulator
- Accelerating Genomics
- Mobile Genomics
- Processing-in-Memory
- Heterogeneous Systems
- Modern SSDs

design

Data-Centric Architectures: Fundamentally Improving Performance and Energy (227-0085-37L)

Course Description

Data movement between the memory units and the compute units of current computing systems is a major performance and energy bottleneck. From large-scale servers to mobile devices, data movement costs dominate computation costs in terms of both performance and energy consumption. For example, data movement between the main memory and the processing cores accounts for 62% of the total system energy in consumer applications. As a result, the data movement bottleneck is a huge burden that greatly limits the energy efficiency and performance of modern computing systems. This phenomenon is an undesired effect of the dichotomy between memory and the processor, which leads to the data movement bottleneck.

Many modern and important workloads such as machine learning, computational biology, graph processing, databases, video analytics, and real-time data analytics suffer greatly from the data movement bottleneck. These workloads are exemplified by irregular memory accesses, relatively low data reuse, low cache line utilization, low arithmetic intensity (i.e., ratio of operations per accessed byte), and large datasets that greatly exceed the main memory size. The computation in these workloads cannot usually compensate for the data movement costs. In order to alleviate this data movement bottleneck, we need a paradigm shift from the traditional processor-centric design, where all computation takes place in the compute units, to a more data-centric design where processing elements are placed closer to or inside where the data resides. This paradigm of computing is known as Processing-in-Memory (PIM).

This is your perfect P&S if you want to become familiar with the main PIM technologies, which represent “the next big thing” in Computer Architecture. You will work hands-on with the first real-world PIM architecture, will explore different PIM architecture designs for important workloads, and will develop tools to enable research of future PIM systems. Projects in this course span software and hardware as well as the software/hardware interface. You can potentially work on developing and optimizing new workloads for the first real-world PIM hardware or explore new PIM designs in simulators, or do something else that can forward our understanding of the PIM paradigm.

Table of Contents

- Data-Centric Architectures: Fundamentally Improving Performance and Energy (227-0085-37L)
- Course Description
- Mentors
- Lecture Video Playlist on YouTube
- Spring 2022 Meetings/Schedule
- Past Lecture Video Playlists on YouTube
- Learning Materials
- Assignments

https://safari.ethz.ch/projects_and_seminars/fall2022/doku.php?id=processing_in_memory

Processing-in-Memory Course (Spring 2023)

- Short weekly lectures
- Hands-on projects

Introducción al Procesamiento en Memoria (en Español)



PIM for Future Computing Systems (in Spanish) - Juan Gómez Luna - Lecture @ Univ. of Córdoba



Onur Mutlu Lectures

31.6K subscribers



Subscribed



16



Share



Clip



Save



753 views 11 months ago

Lecture: Enabling the Processing-in-Memory Paradigm for Future Computing Systems (Introducción al Procesamiento en Memoria)

Lecturer: Dr. Juan Gómez Luna

Date: March 16, 2022 Show more

Análisis de una Arquitectura Real de Procesamiento en Memoria (en Español)

PIM Adoption Issues

How to Enable PIM Adoption?

Dr. Juan Gómez Luna
Professor Onur Mutlu