Enabling the Adoption of Data-Centric Systems: Hardware/Software Support for Processing-Using-Memory Architectures

Memory & Storage Summit 2024 – 07 December 2024

Geraldo F. Oliveira

geraldofojunior@gmail.com

https://geraldofojunior.github.io/

SAFARI

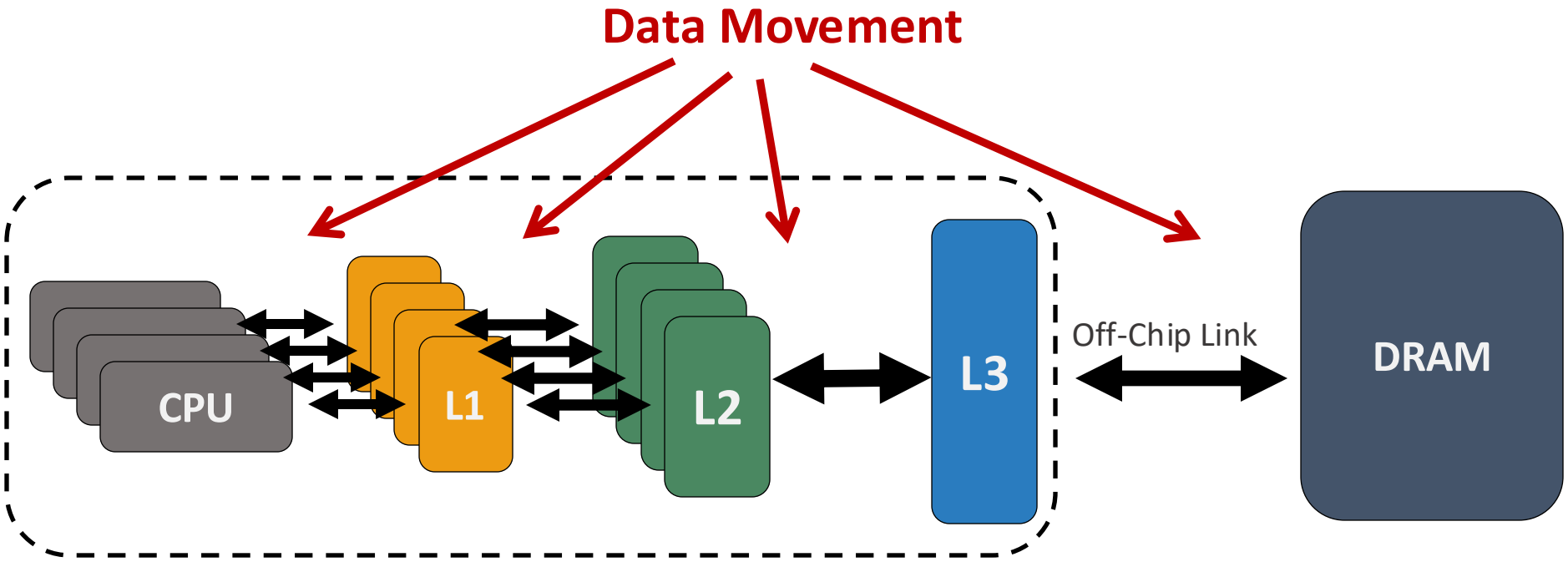
ETH zürich

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
  - …

Data Movement Bottlenecks (1/2)

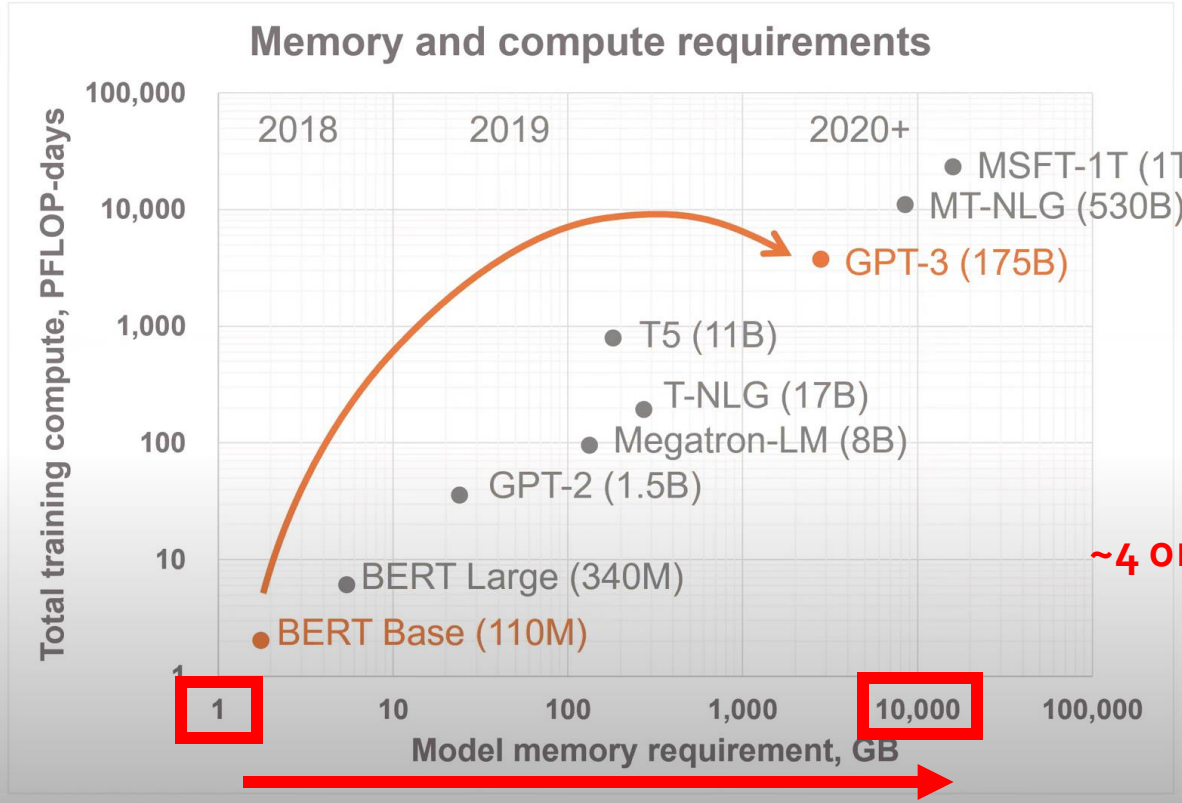


Data movement bottlenecks happen because of:

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

Huge Demand for Performance & Efficiency

Exponential Growth of Neural Networks



1800x more compute

In just **2 years**

Tomorrow, **multi-trillion** parameter models

~4 orders of magnitude increase in memory requirement in just two years!

4:43 / 1:08:15

SAFARI

https://www.youtube.com/watch?v=x2-qB0J7KHw

Data Movement Overwhelms Modern Machines

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

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

Google Workloads for Consumer Devices: Mitigating Data Movement Bottlenecks

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

Data Movement Overwhelms Modern Accelerators

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

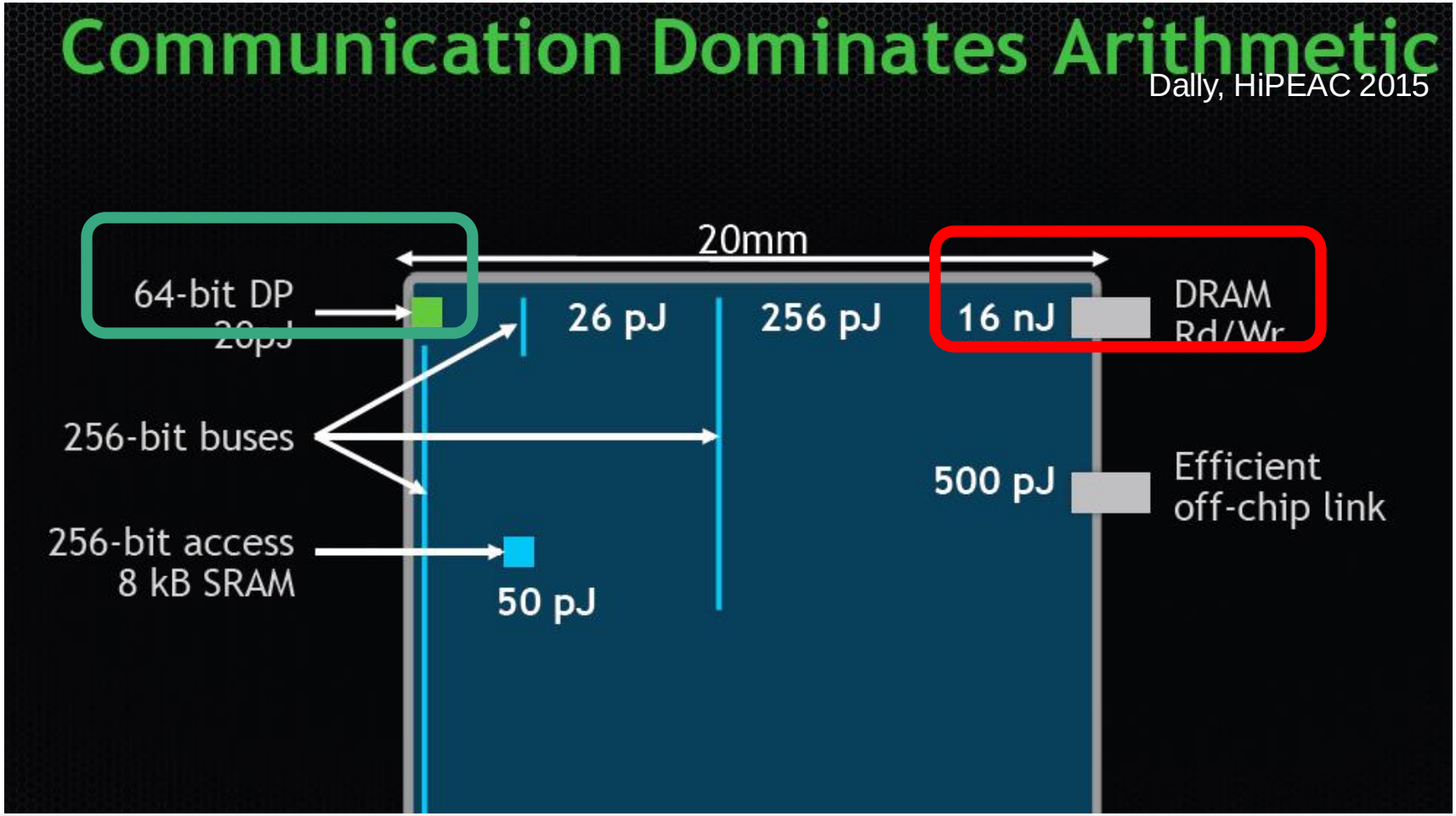
> 90% of the total system energy is spent on memory in large ML models

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

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

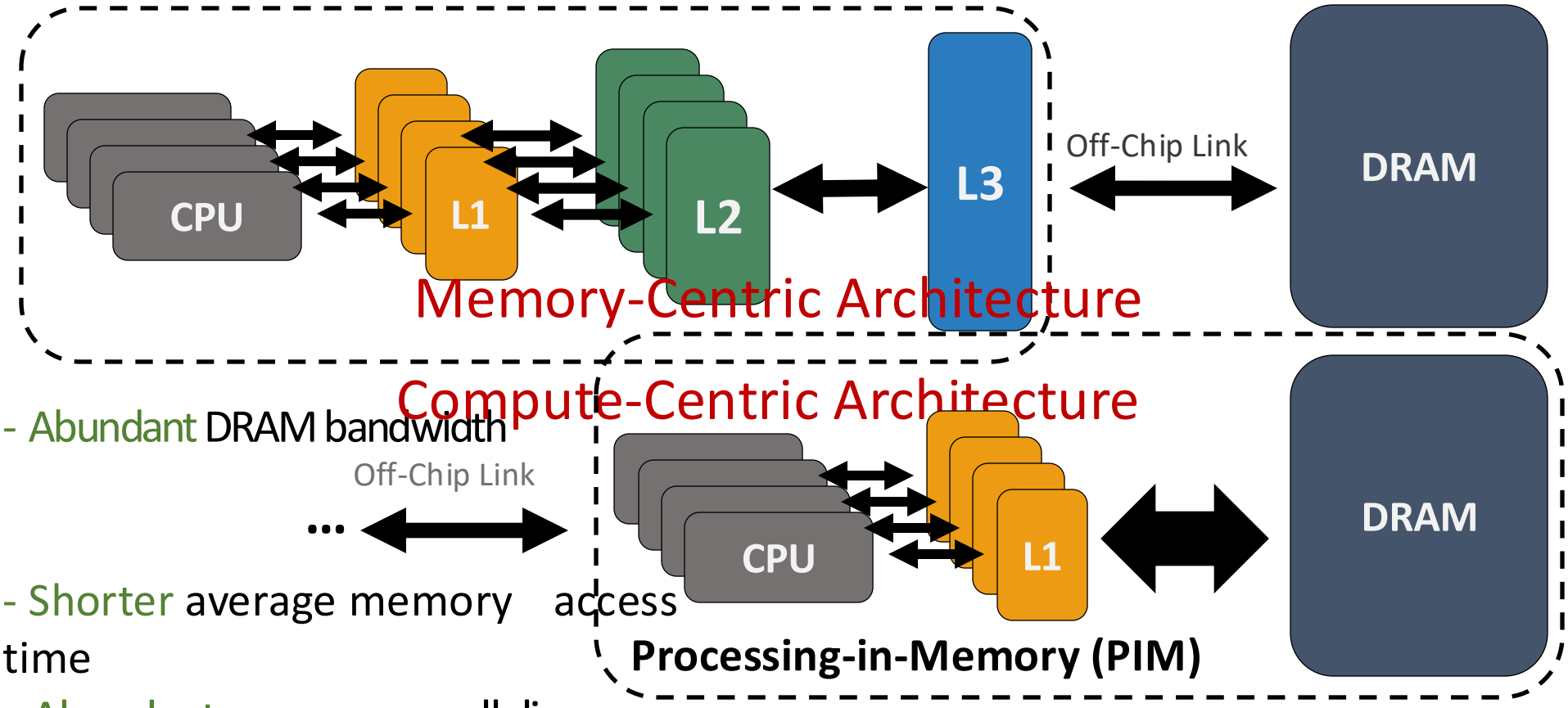
[†]*Carnegie Mellon Univ.* [◇]*Stanford Univ.* [‡]*Univ. of Illinois Urbana-Champaign* [§]*Google* [⋆]*ETH Zürich*

Data Movement vs. Computation Energy



A memory access consumes ~100-1000X the energy of a complex addition

Data Movement Bottlenecks (2/2)



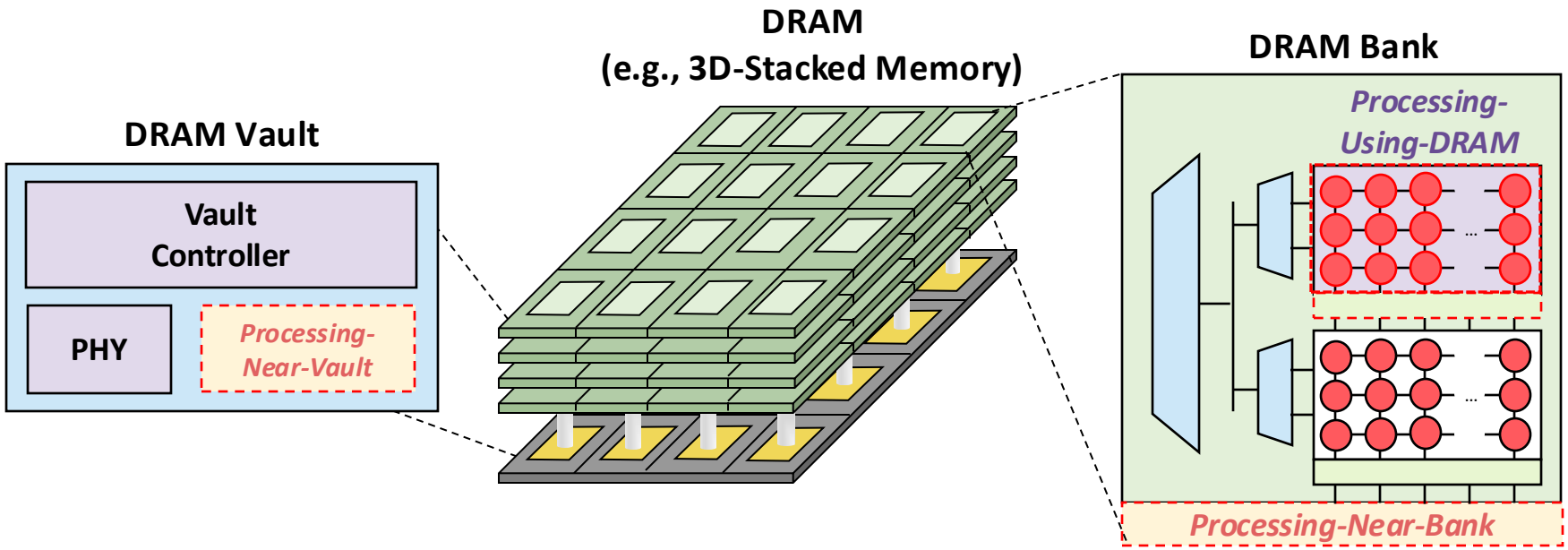
- Abundant DRAM bandwidth
- Shorter average memory access time
- Abundant per-array parallelism

Processing-in-Memory:
Overview

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

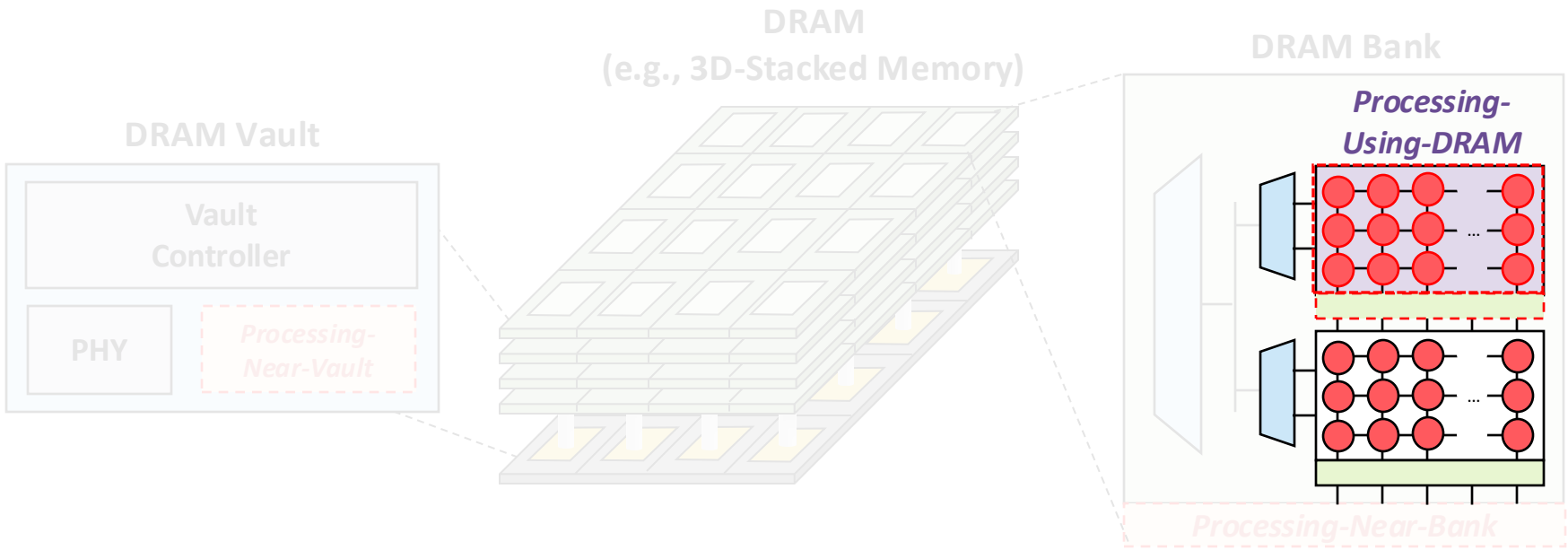


Processing-in-Memory:
Overview

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



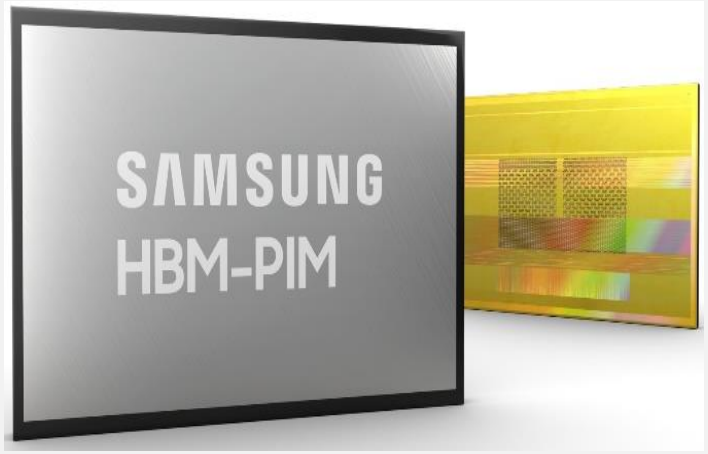
Processing-in-Memory:
Landscape of Real Systems (I)

Processing-in-Memory architectures are
now commercially available

SK hynix AiM

Near-DRAM-bank processing
for neural networks

Alibaba HB-PNM

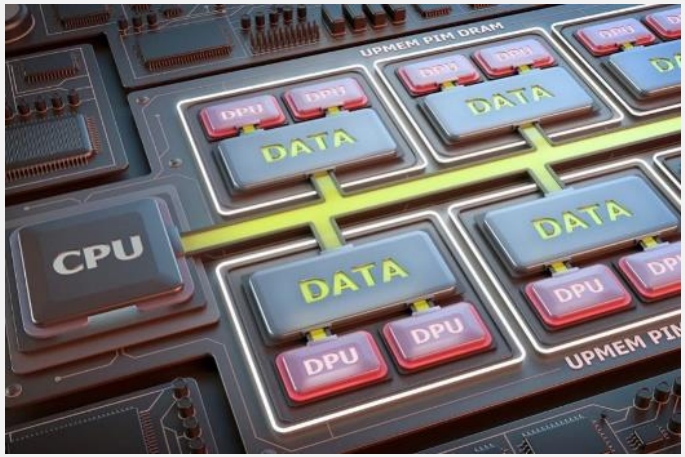


Hybrid bonding with logic
for recommendation systems

Processing-in-Memory:
Landscape of Real Systems (II)

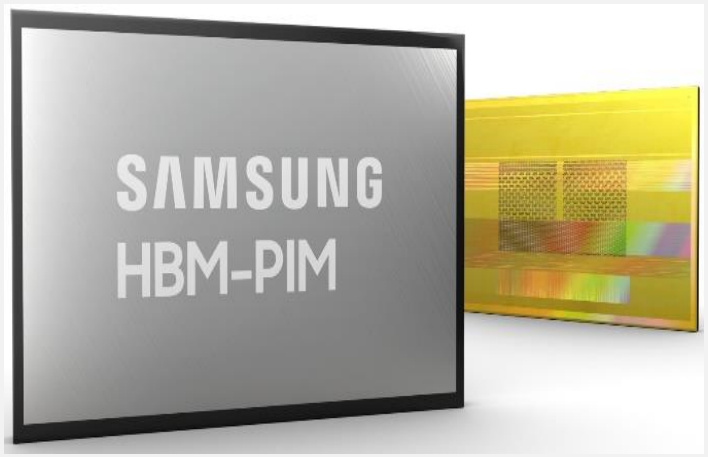
Processing-in-Memory architectures are
now commercially available

UPMEM



Near-DRAM-bank processing
for general-purpose computing

Samsung HBM-PIM

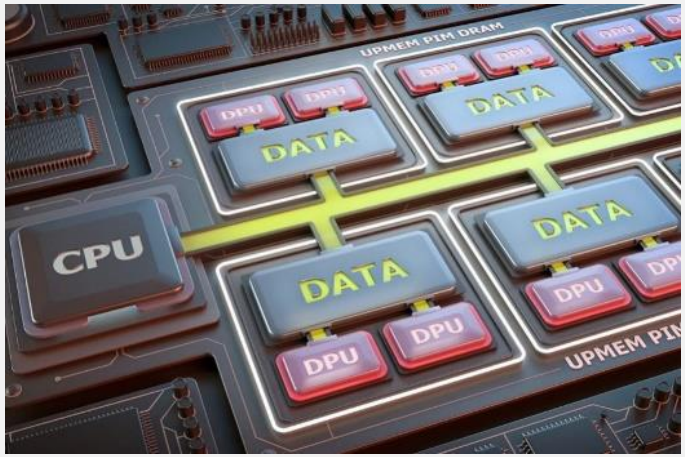


Near-DRAM-bank processing
for neural networks

Processing-in-Memory:
Landscape of Real Systems (II)

Processing-in-Memory architectures are now commercially available

UPMEM



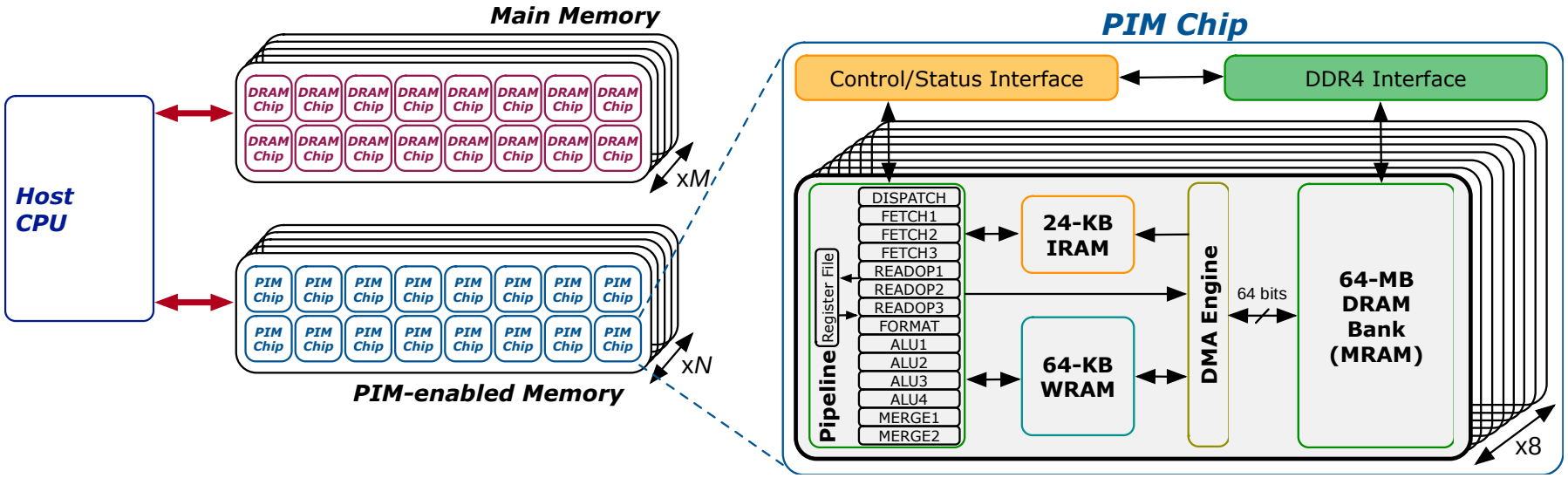
Near-DRAM-bank processing
for general-purpose computing

Samsung HBM-PIM

Near-DRAM-bank processing
for neural networks

Processing-in-Memory:
The UPMEM Architecture

UPMEM: Near-DRAM-bank processing
for general-purpose computing

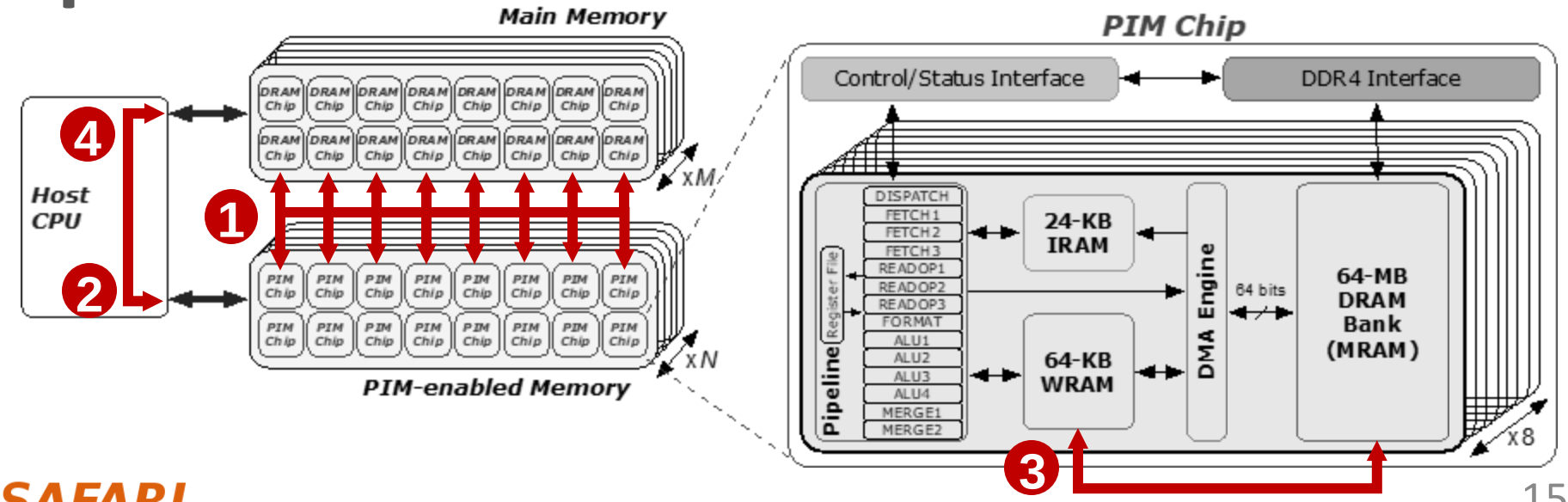


Integration of UPMEM PIM in a system follows an accelerator model

The Programmability Barrier:
Overview

Programming the UPMEM-based system requires:

1. Splitting input data and computation across PIM chips
2. Transferring input data from main memory to PIM chips
3. Manually handling caching in PIM's scratchpad memory
4. Transferring output data from PIM chips to main memory



The Programmability Barrier:
Summary

Programmer's Tasks:

Goal:

Align data | Collect parameters | Distribute parameters | Launch computation | Collect results | Manage scratchpad | Orchestrate computation | Just write my kernel

Problem

Programming the UPMEM system leads to non-trivial effort → requires knowledge of the underlying hardware and manual fine-grained data movement handling

```
    }
}
```

Our Goal

Goal

To ease programmability for the UPMEM system,
allowing a programmer to write
efficient PIM-friendly code
without the need to
explicitly manage hardware resources

SimplePIM:

A Software Framework for Productive and Efficient Processing in Memory

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

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

Jinfan Chen[1] Juan Gómez-Luna[1] Izzat El Hajj[2] Yuxin Guo[1] Onur Mutlu[1]

[1]ETH Zürich [2]American University of Beirut

SimplePIM Programming Framework: Overview

SimplePIM provides standard abstractions to build and deploy applications on PIM systems

1 **Management interface**

→ Metadata for PIM-resident arrays

2 **Communication interface**

→ Abstractions for host-PIM and PIM-PIM communication

3 **Processing interface**

→ Iterators (`map`, `reduce`, `zip`) to implement workloads

Evaluation Results:
Evaluation Methodology

- **Evaluated system**
  - UPMEM PIM system with 2,432 PIM cores with 159 GB of PIM DRAM
- **Real-world Benchmarks**
  - Vector addition
  - Reduction
  - Histogram
  - K-Means
  - Linear regression
  - Logistic regression
- Comparison to hand-optimized codes in terms of programming productivity and performance

Evaluation Results:
Productive Improvement (I)

- Example: Hand-optimized histogram with UPMEM SDK

```
... // Initialize global variables and functions for histogram
int main_kernel() {
  if (tasklet_id == 0)
    mem_reset(); // Reset the heap
  ... // Initialize variables and the histogram
  T *input_buff_A = (T*)mem_alloc(2048); // Allocate buffer in scratchpad memory

  for (unsigned int byte_index = base_tasklet; byte_index < input_size; byte_index += stride) {
    // Boundary checking
    uint32_t l_size_bytes = (byte_index + 2048 >= input_size) ? (input_size - byte_index) : 2048;
    // Load scratchpad with a DRAM block
    mram_read((const __mram_ptr void*)(mram_base_addr_A + byte_index), input_buff_A, l_size_bytes);
    // Histogram calculation
    histogram(hist, bins, input_buff_A, l_size_bytes/sizeof(uint32_t));
  }
  ...
  barrier_wait(&my_barrier); // Barrier to synchronize PIM threads
  ... // Merging histograms from different tasklets into one histo_dpu
  // Write result from scratchpad to DRAM
  if (tasklet_id == 0)
    if (bins * sizeof(uint32_t) <= 2048)
      mram_write(histo_dpu, (__mram_ptr void*)mram_base_addr_histo, bins * sizeof(uint32_t));
    else
      for (unsigned int offset = 0; offset < ((bins * sizeof(uint32_t)) >> 11); offset++) {
        mram_write(histo_dpu + (offset << 9), (__mram_ptr void*)(mram_base_addr_histo +
                  (offset << 11)), 2048);
      }
  return 0;
}
```

Evaluation Results:
Productive Improvement (II)

- Example: SimplePIM histogram

```
// Programmer-defined functions in the file "histo_filepath"
void init_func (uint32_t size, void* ptr) {
  char* casted_value_ptr = (char*) ptr;
  for (int i = 0; i < size; i++)
    casted_value_ptr[i] = 0;
}

void acc_func (void* dest, void* src) {
  *(uint32_t*)dest += *(uint32_t*)src;
}

void map_to_val_func (void* input, void* output, uint32_t* key) {
  uint32_t d = *((uint32_t*)input);
  *(uint32_t*)output = 1;
  *key = d * bins >> 12;
}

// Host side handle creation and iterator call
handle_t* handle = simple_pim_create_handle("histo_filepath", REDUCE, NULL, 0);

// Transfer (scatter) data to PIM, register as "t1"
simple_pim_array_scatter("t1", src, bins, sizeof(T), management);

// Run histogram on "t1" and produce "t2"
simple_pim_array_red("t1", "t2", sizeof(T), bins, handle, management);
```

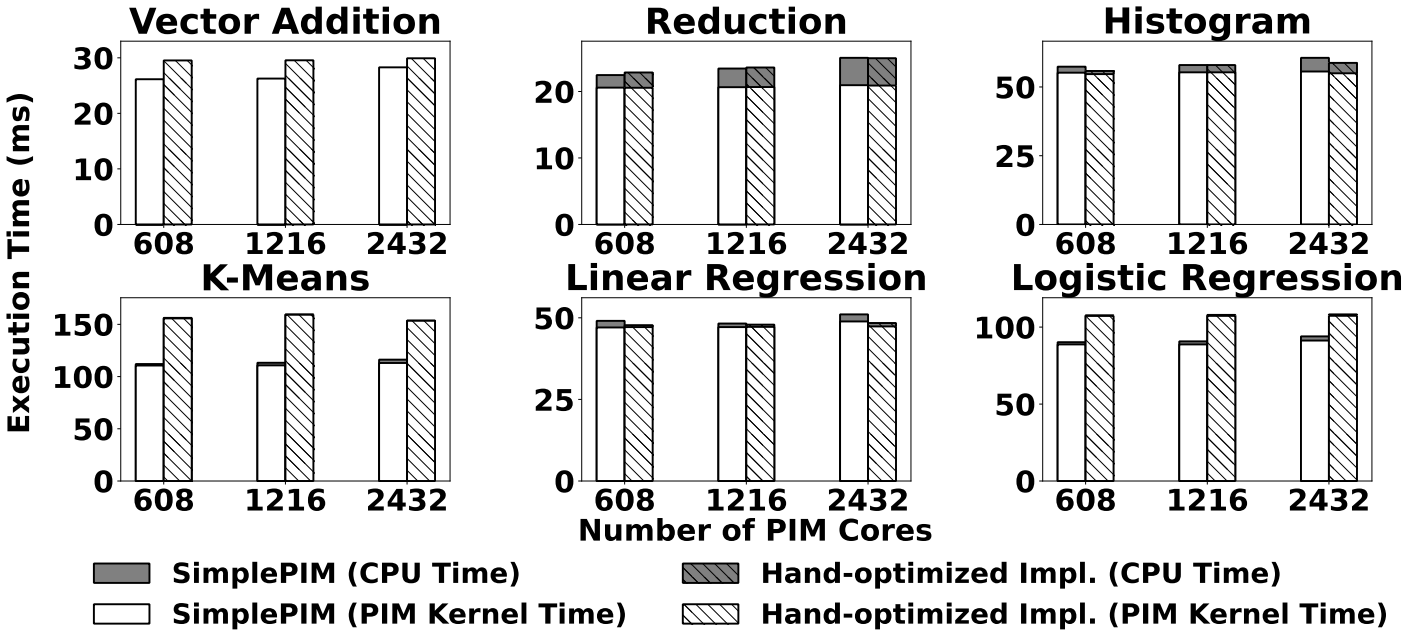
Evaluation Results:
Productive Improvement (III)

- Lines of code (LoC) reduction

	SimplePIM	Hand-optimized	LoC Reduction
Reduction	14	83	5.93×
Vector Addition	14	82	5.86×
Histogram	21	114	5.43×
Linear Regression	48	157	3.27×
Logistic Regression	59	176	2.98×
K-Means	68	206	3.03×

SimplePIM reduces the number of lines of effective code by a factor of 2.98× to 5.93×

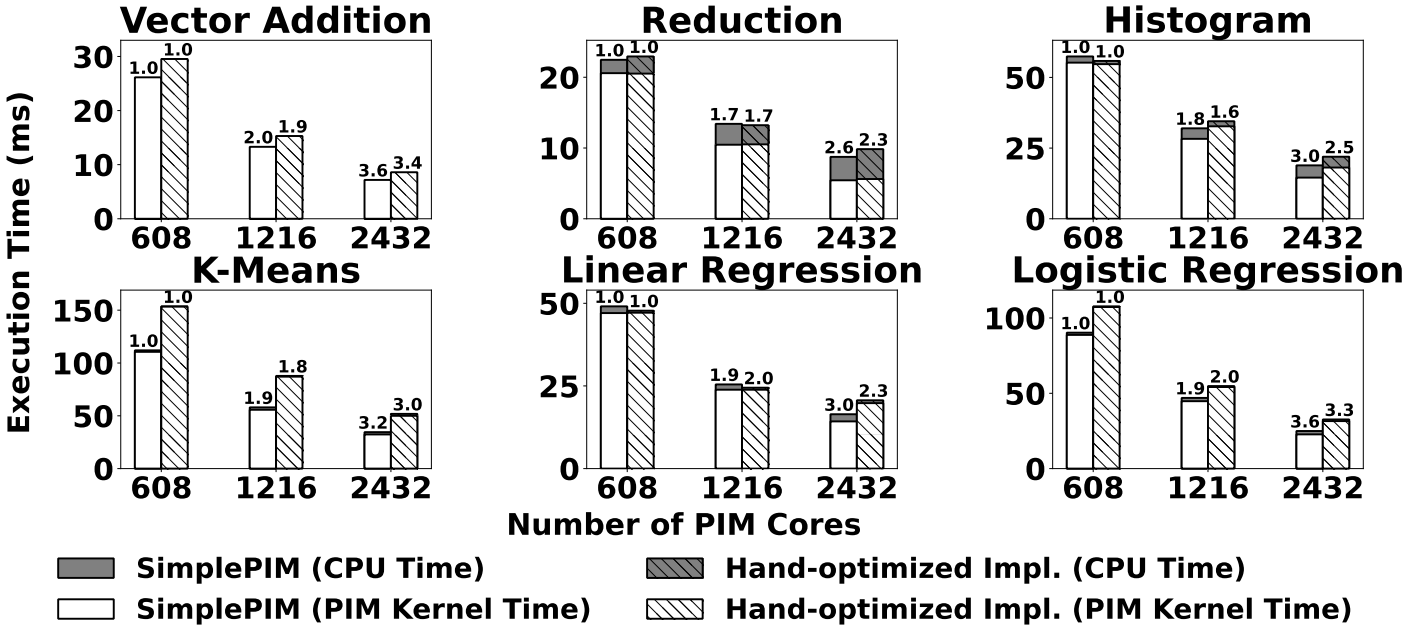
Evaluation Results:
Weak Scaling Analysis



SimplePIM achieves comparable performance for `reduction`, `histogram`, and `linear regression`

SimplePIM outperforms hand-optimized implementations for `vector addition`, `logistic regression`, and `k-means` by 10%-37%

Evaluation Results:
Strong Scaling Analysis

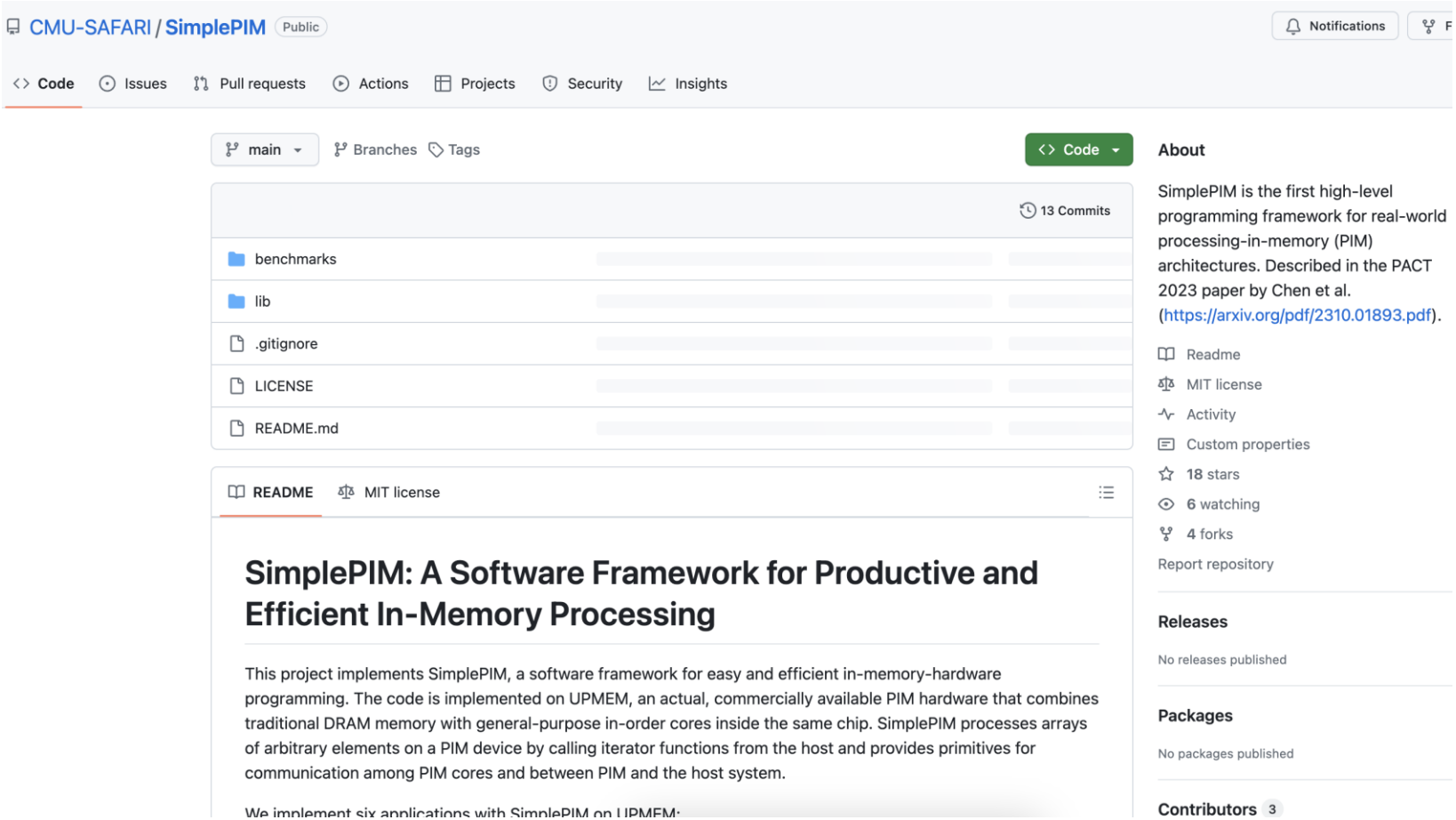


SimplePIM scales better than hand-optimized implementations for `reduction`, `histogram`, and `linear regression`

SimplePIM outperforms hand-optimized implementations for `vector addition`, `logistic regression`, and `k-means` by 15%-43%

Source Code

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



SimplePIM:

A Software Framework for Productive and Efficient Processing in Memory

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

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

Jinfan Chen[1] Juan Gómez-Luna[1] Izzat El Hajj[2] Yuxin Guo[1] Onur Mutlu[1]

[1]ETH Zürich [2]American University of Beirut

DaPPA:

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

- Geraldo F. Oliveira, Alain Kohli, David Novo, Juan Gómez-Luna, Onur Mutlu
  "DaPPA: A Data-Parallel Framework for Processing-in-Memory Architectures,"
  arXiv:2310.10168 [cs.AR]
  2nd Place ACM Student Research Competition at *the 32nd International Conference on Parallel Architectures and Compilation Techniques* (***PACT***), Vienna, Austria, October 2023.

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

Geraldo F. Oliveira⋆ Alain Kohli⋆ David Novo‡ Juan Gómez-Luna⋆ Onur Mutlu⋆
⋆*ETH Zürich* ‡*LIRMM, Univ. Montpellier, CNRS*

1. Motivation & Problem

The increasing prevalence and growing size of data in modern applications have led to high costs for computation in traditional *processor-centric computing* systems. To mitigate these costs, the *processing-in-memory* (PIM) [1–6] paradigm moves computation closer to where the data resides, reducing the need to move data between memory and the processor. Even though the concept of PIM has been first proposed in the 1960s [7, 8], real-world PIM systems have only recently

face [15, 16] that abstracts the hardware components of the UPMEM system. Using this key idea, DaPPA transforms a data-parallel pattern-based application code into the appropriate UPMEM-target code, including the required APIs for data management and code partition, which can then be compiled into a UPMEM-based binary *transparently* from the programmer. While generating UPMEM-target code, DaPPA implements several code optimizations to improve end-to-end performance.

DaPPA:
Key Idea & Overview

Key Idea

Leverage an intuitive data-parallel pattern-based interface for PIM programming

DaPPA , a Data-Parallel PIM Architecture that automatically distributes input and gathers output data, handles memory management, and parallelizes work across PIM cores

DaPPA is composed of three main components:

1. Data-Parallel Pattern APIs
2. Dataflow Programming Interface
3. Dynamic Template-Based Compilation

DaPPA:
Putting All Together

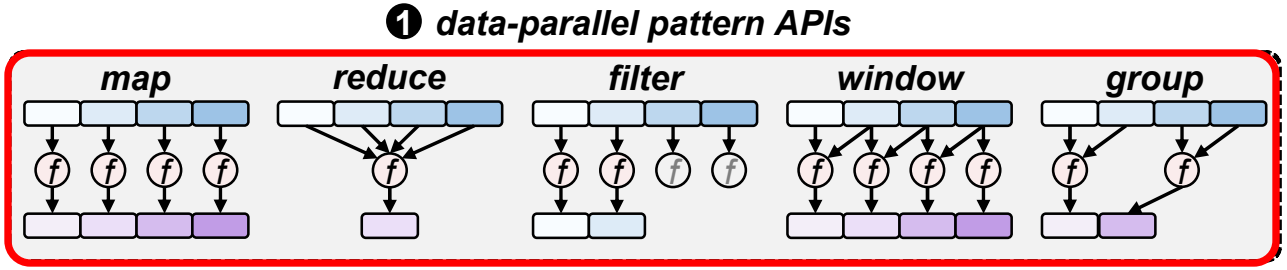
Example of DaPPA's implementation of a
vector dot product operation

reduce
$C = A_0B_0 +$
map $A_1B_1 +$
$A_2B_2 +$
A_3B_3

target computation

DaPPA:
Putting All Together

Example of DaPPA's implementation of a
vector dot product operation

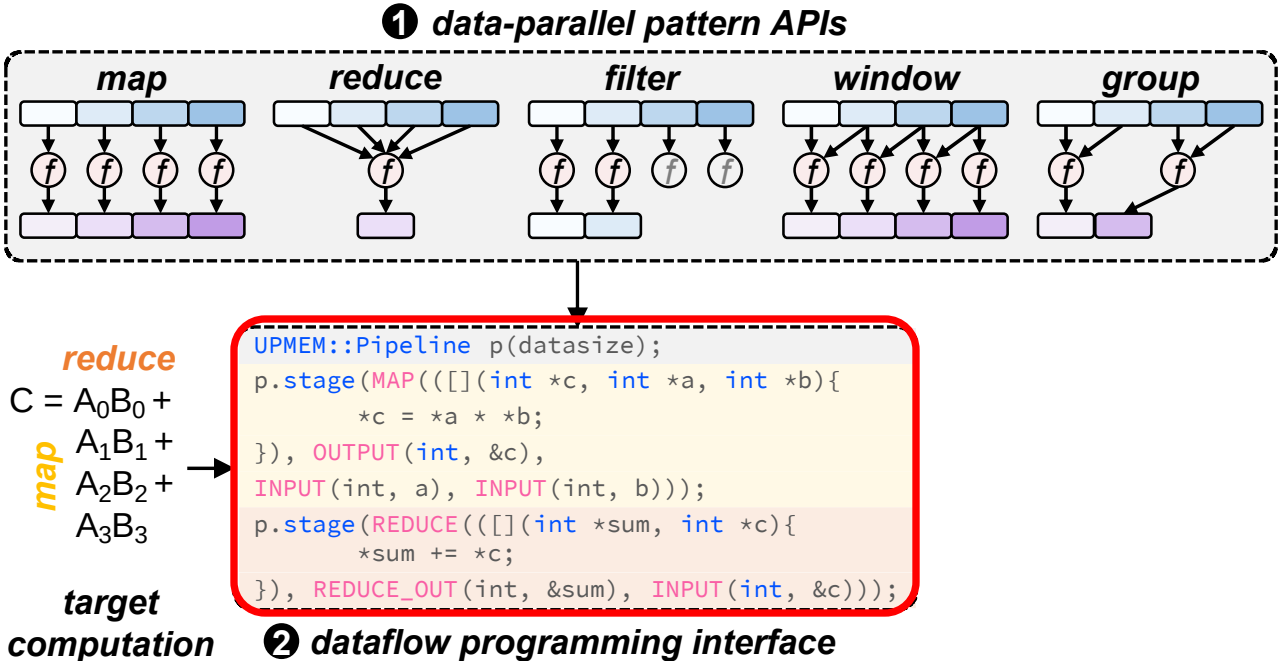


reduce
C = A_0B_0 +
map A_1B_1 +
A_2B_2 +
A_3B_3

target computation

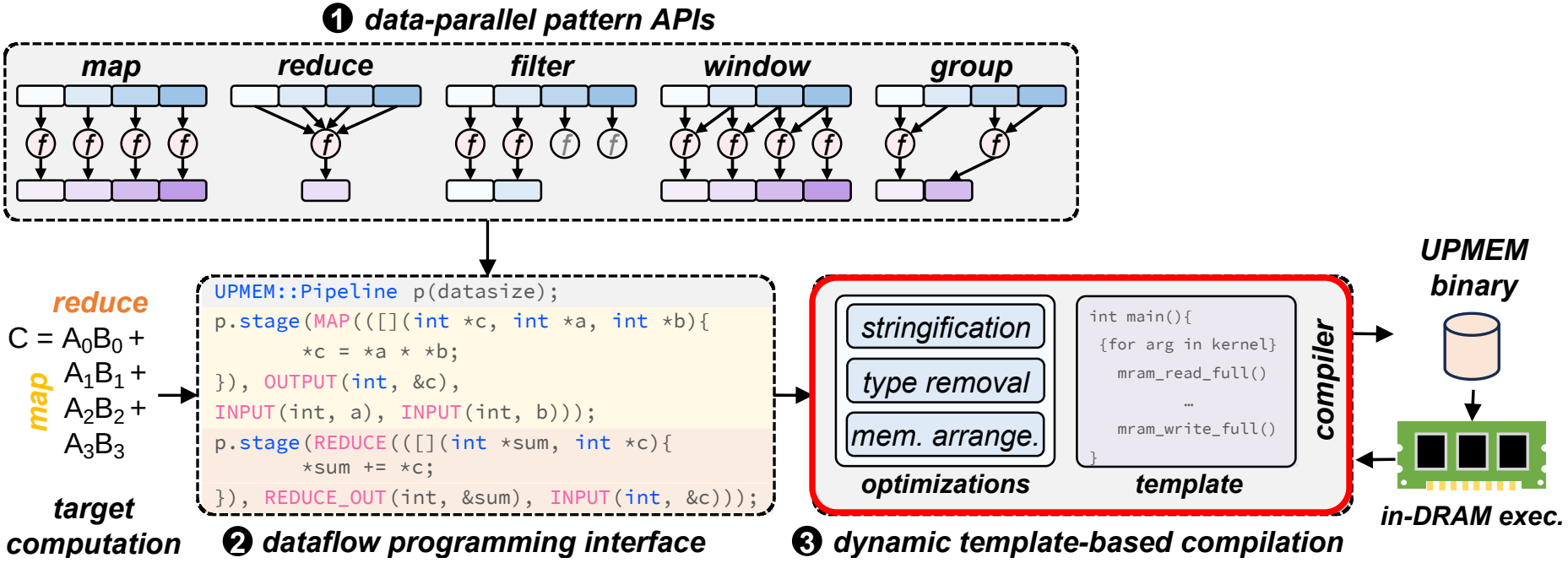
DaPPA:
Putting All Together

Example of DaPPA's implementation of a
vector dot product operation



DaPPA:
Putting All Together

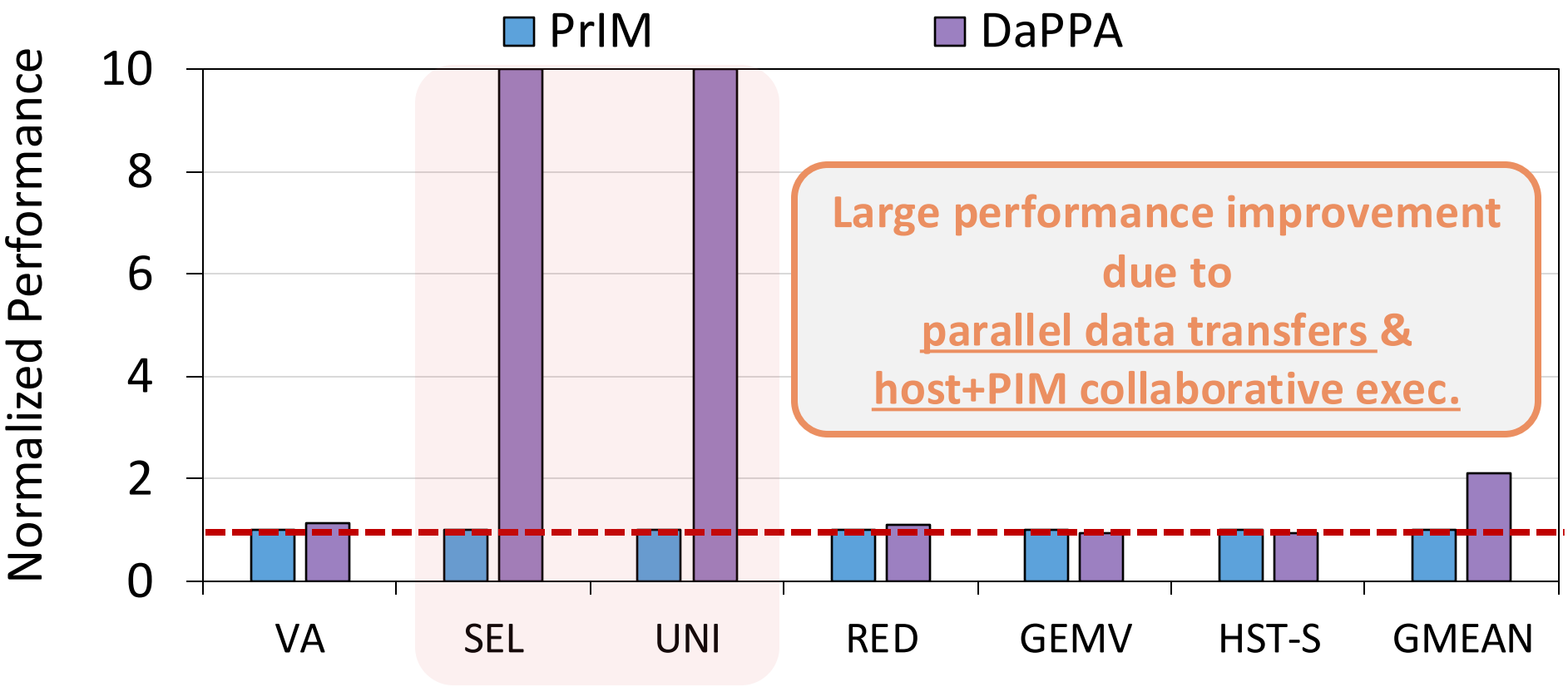
Example of DaPPA's implementation of a
vector dot product operation



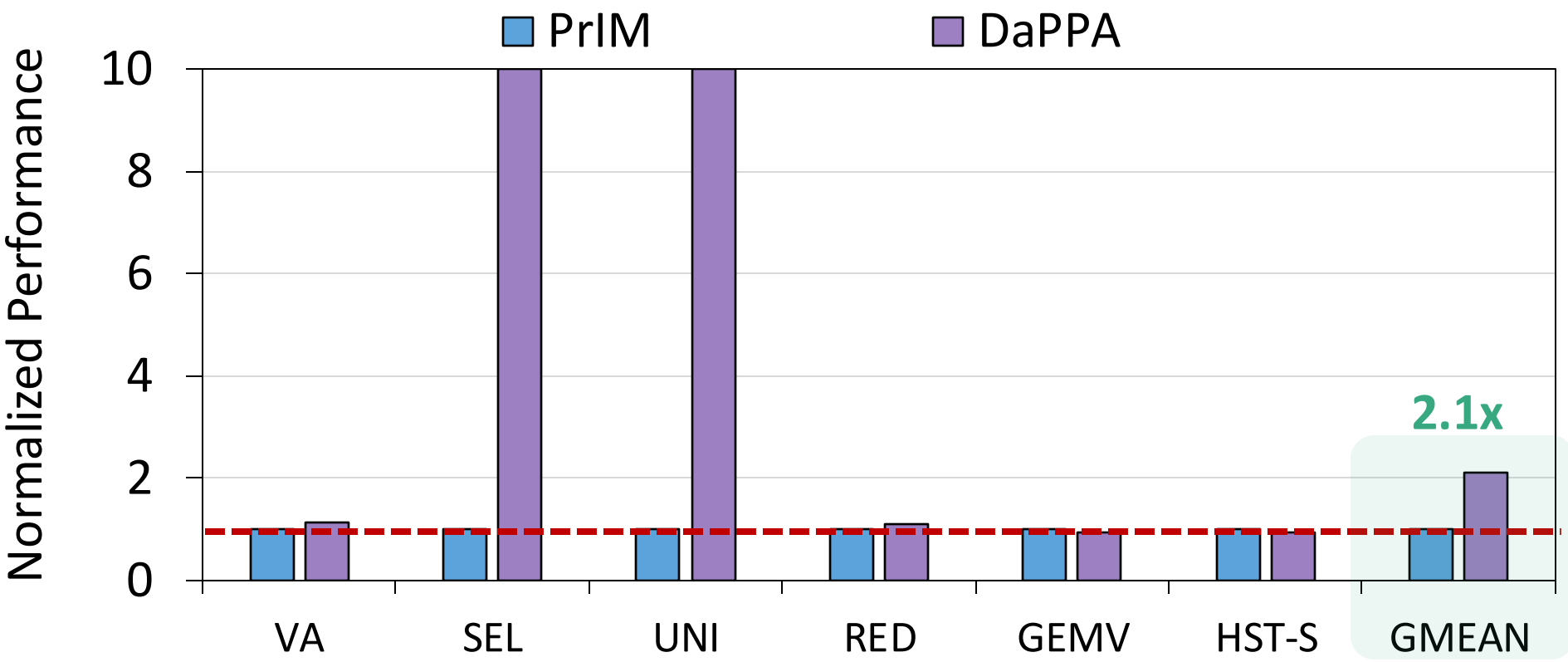
Evaluation:
Methodology Overview

- **Evaluation Setup**
  - **Host CPU:** 2-socket Intel® Xeon Silver 4110 CPU
  - **PIM Cores:** 20 UPMEM PIM DIMMs (160 GB PIM memory)
  - **2560 DPUs** in total
- **Workloads:** 6 workloads from the PrIM benchmark suite
  - Vector addition (**VA**); Select (**SEL**); Unique (**UNI**); Reduce (**RED**); General matrix-vector multiply (**GEMV**); Histogram small (**HST-S**)
- **Metrics**
  - **End-to-end execution time** (average of 10 runs)
  - **Programming complexity** (in lines of code)

Evaluation:
Performance Analysis

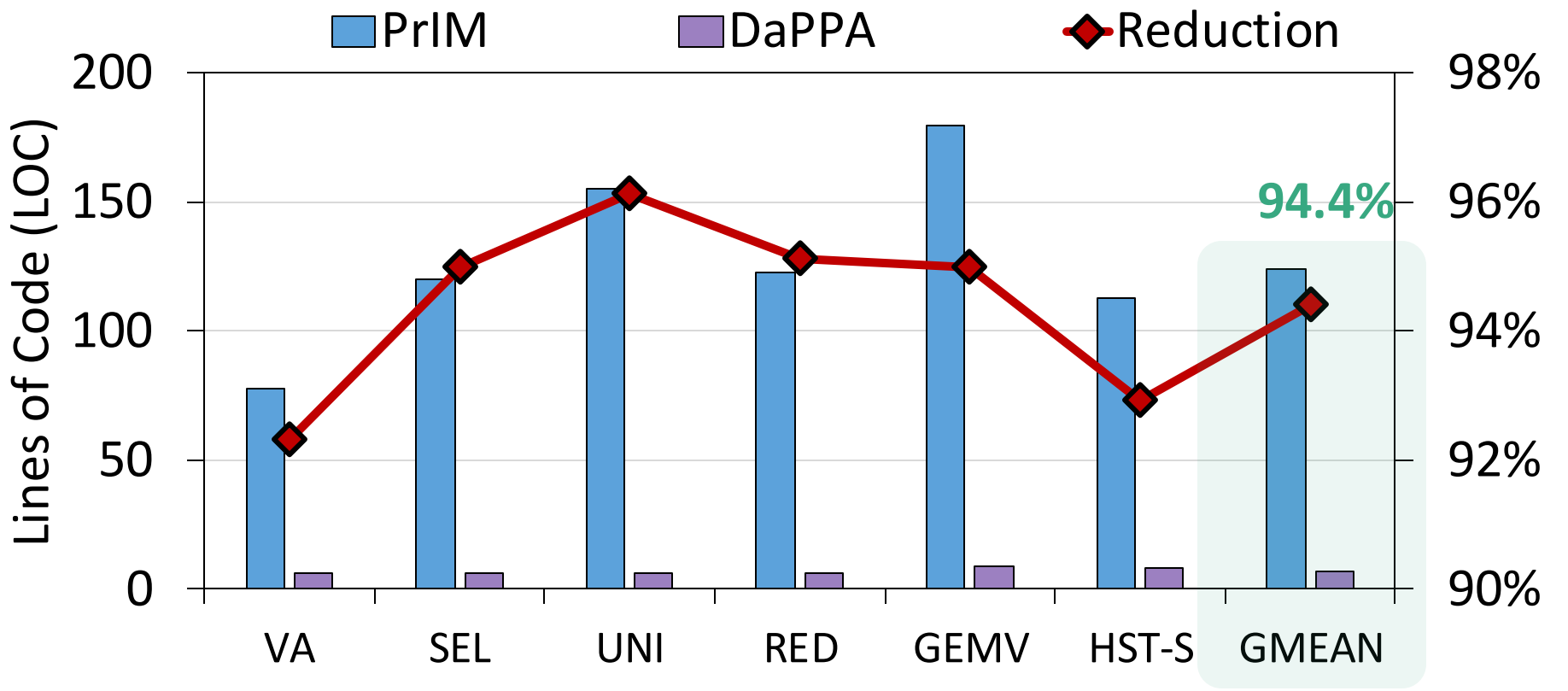


Evaluation:
Performance Analysis



DaPPA significantly improves end-to-end performance compared to hand-tuned implementations

Evaluation:
Programming Complexity Analysis



DaPPA significantly reduces programming complexity by abstracting hardware components

Evaluation:
Comparison to State-of-the-Art

SimplePIM [Chen+, PACT'23]: a framework that uses
(1) iterator functions and (2) primitives for communication
to aid PIM programmability

Compared to SimplePIM, DaPPA provides three key benefits

1. **Higher abstraction level** → The programmer does *not* need to **manually specify communication patterns** used during computation
2. **Support for more parallel patterns** → DaPPA supports **two more parallel primitives** (window and group), and allows the **mixing of parallel patterns**
3. **Further execution optimizations** → DaPPA allows using idle host resources for **collaborative execution**

DaPPA improves state-of-the-art frameworks for PIM programmability

DaPPA:

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

- Geraldo F. Oliveira, Alain Kohli, David Novo, Juan Gómez-Luna, Onur Mutlu
  "DaPPA: A Data-Parallel Framework for Processing-in-Memory Architectures,"
  arXiv:2310.10168 [cs.AR]
  2nd Place ACM Student Research Competition at *the 32nd International Conference on Parallel Architectures and Compilation Techniques* (***PACT***), Vienna, Austria, October 2023.

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

Geraldo F. Oliveira⋆ Alain Kohli⋆ David Novo‡ Juan Gómez-Luna⋆ Onur Mutlu⋆
⋆*ETH Zürich* ‡*LIRMM, Univ. Montpellier, CNRS*

1. Motivation & Problem

The increasing prevalence and growing size of data in modern applications have led to high costs for computation in traditional *processor-centric computing* systems. To mitigate these costs, the *processing-in-memory* (PIM) [1–6] paradigm moves computation closer to where the data resides, reducing the need to move data between memory and the processor. Even though the concept of PIM has been first proposed in the 1960s [7, 8], real-world PIM systems have only recently

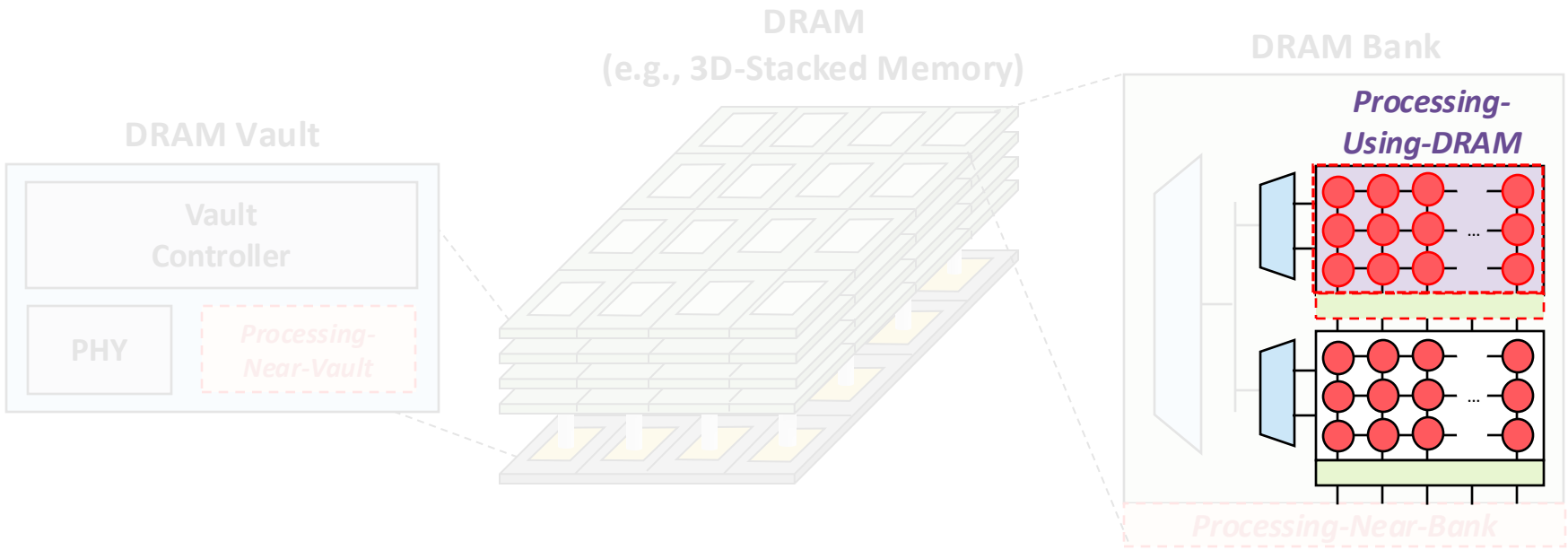
face [15, 16] that abstracts the hardware components of the UPMEM system. Using this key idea, DaPPA transforms a data-parallel pattern-based application code into the appropriate UPMEM-target code, including the required APIs for data management and code partition, which can then be compiled into a UPMEM-based binary *transparently* from the programmer. While generating UPMEM-target code, DaPPA implements several code optimizations to improve end-to-end performance.

Processing-in-Memory:
Overview

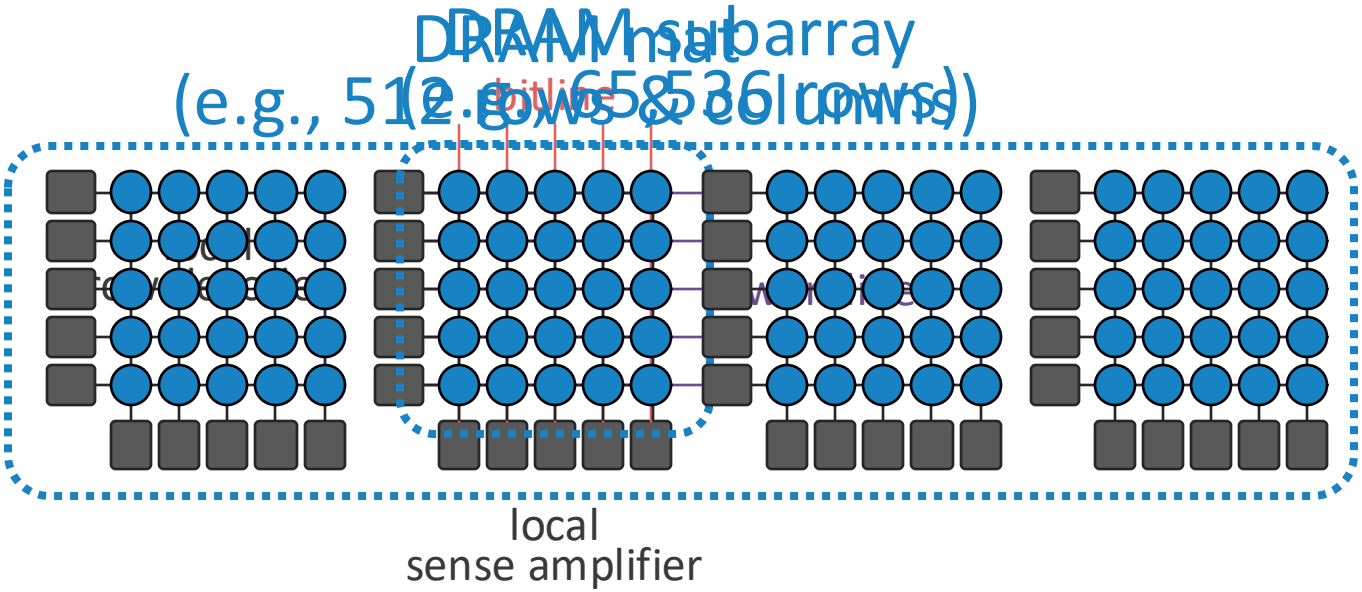
Two main approaches for Processing-in-Memory:

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

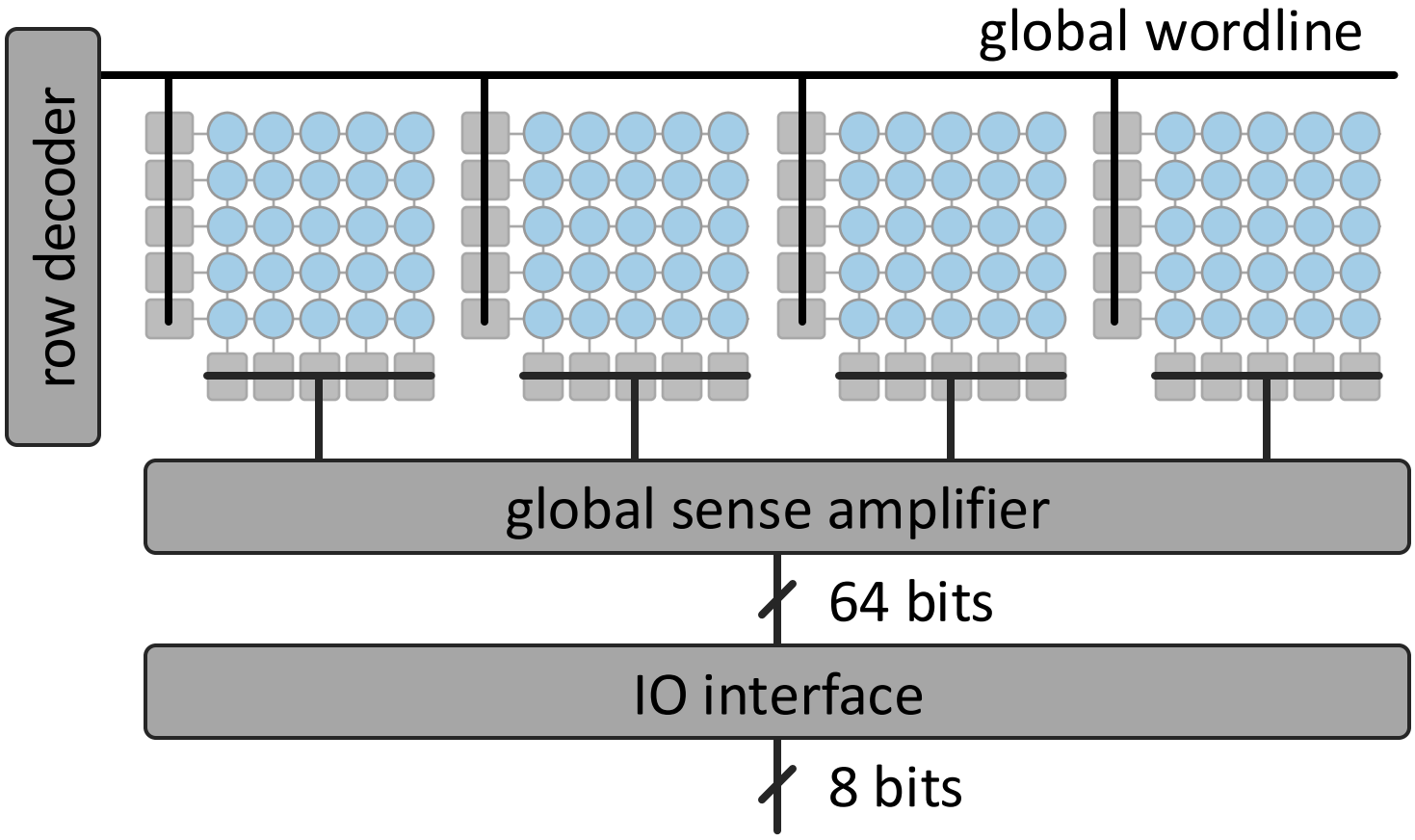
2. **Processing-Using-Memory**: uses the operational principles of memory cells to perform computation



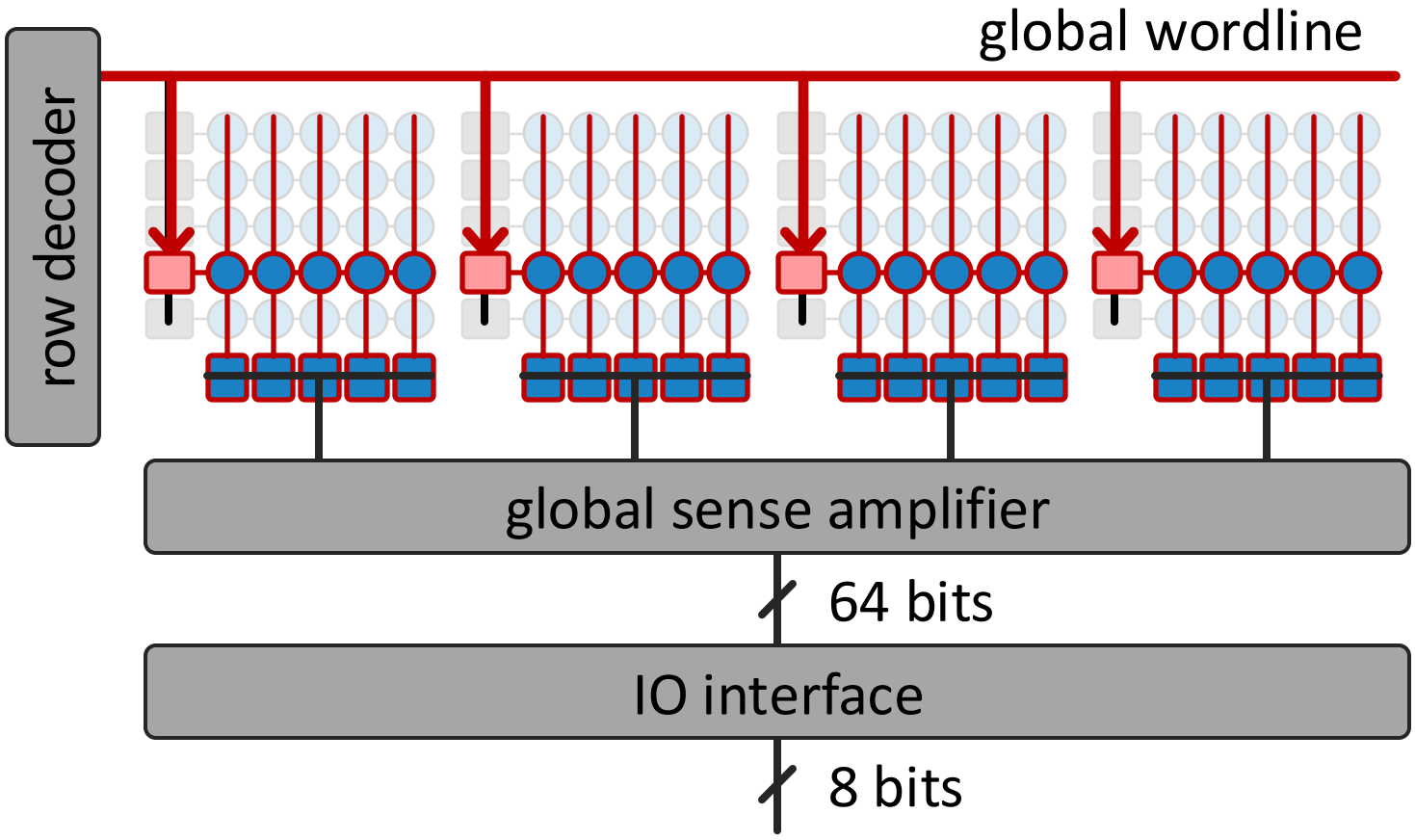
Background:
DRAM Hierarchical Organization



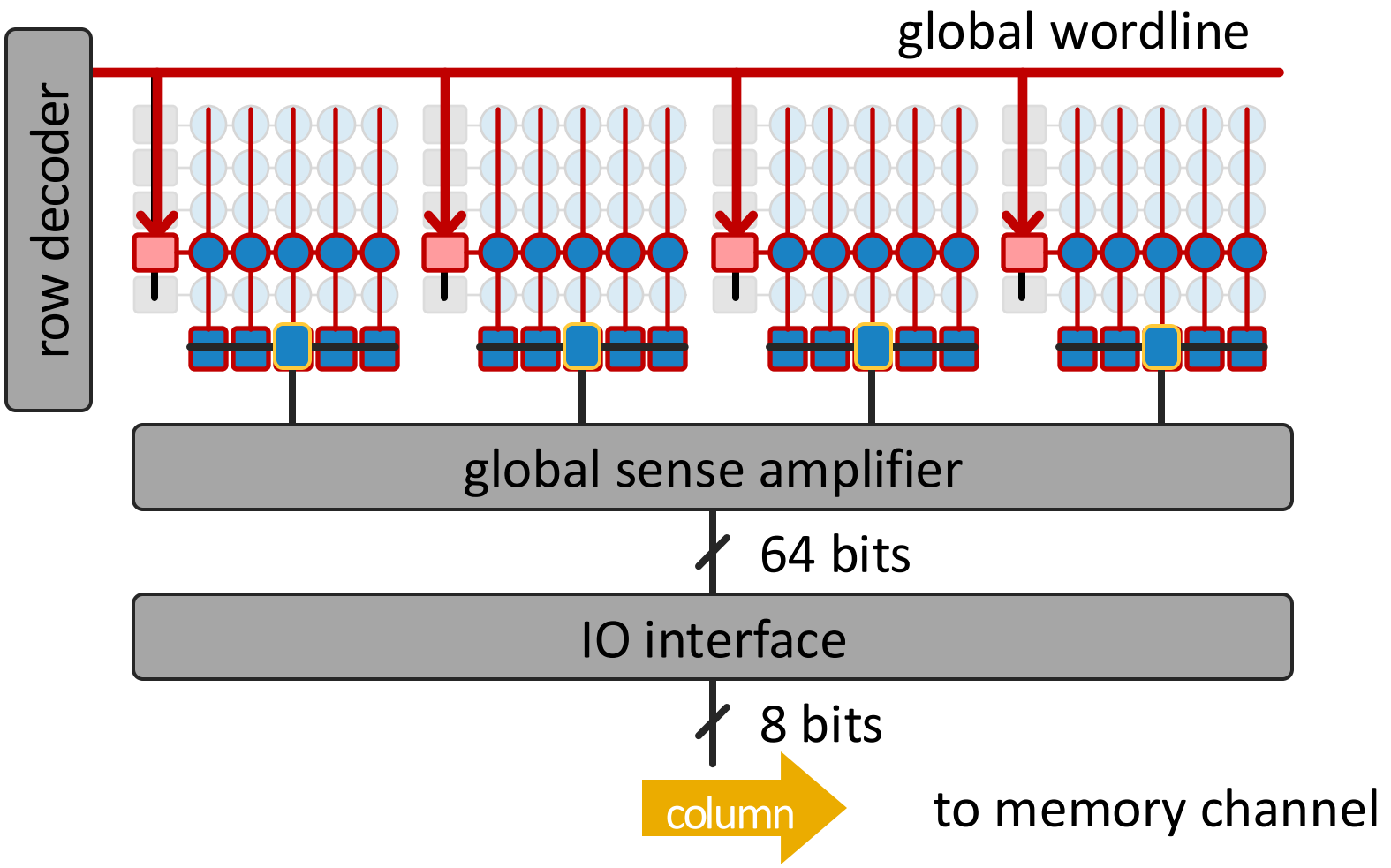
Background:
DRAM Hierarchical Organization



Background:
DRAM Operation – Row Access (ACTIVATE)

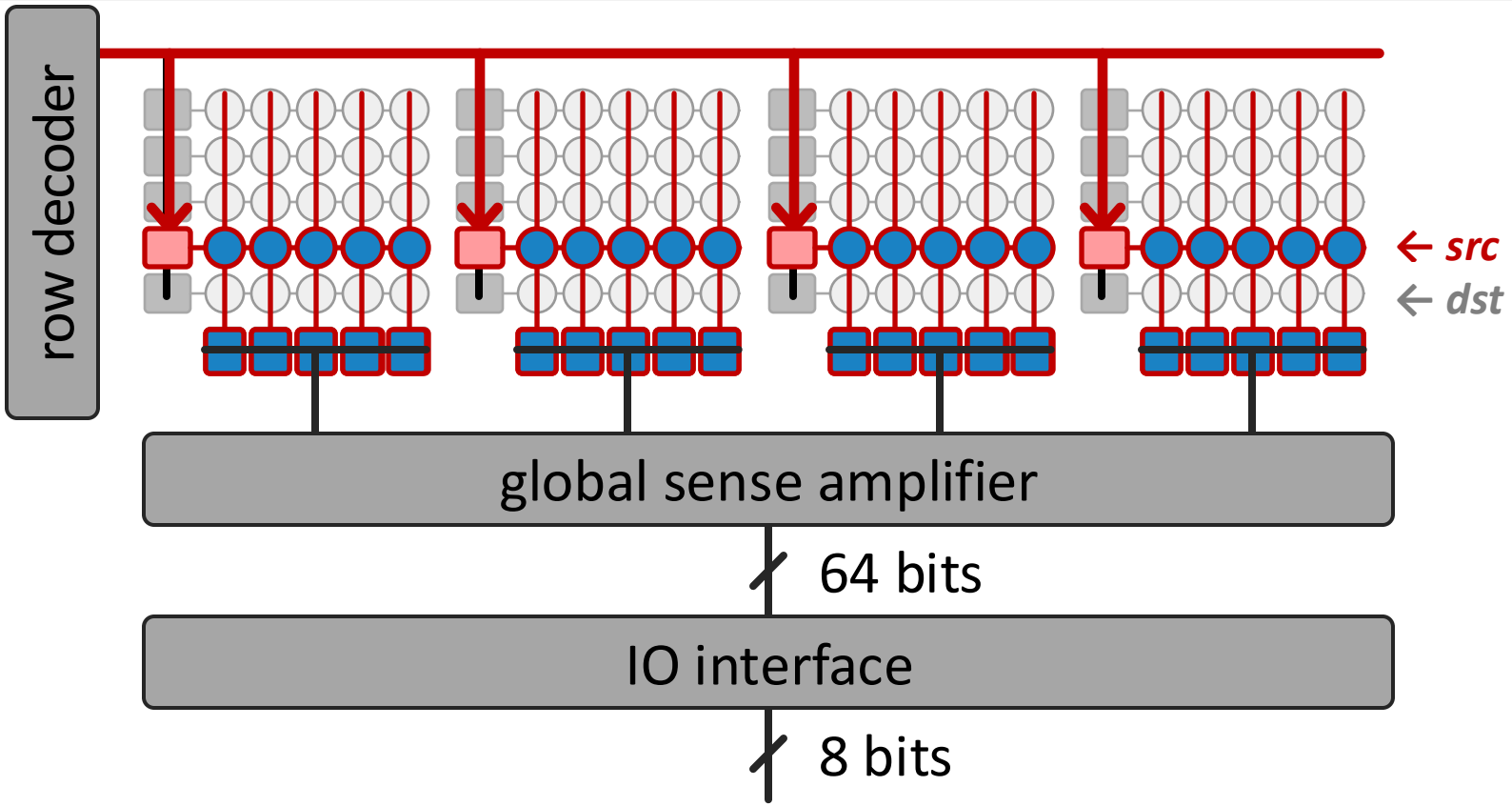


Background:
DRAM Operation – Column Access (READ)



Background:
In-DRAM Row Copy

In-DRAM row copy is performed by
issuing back-to-back ACTIVATES to the DRAM

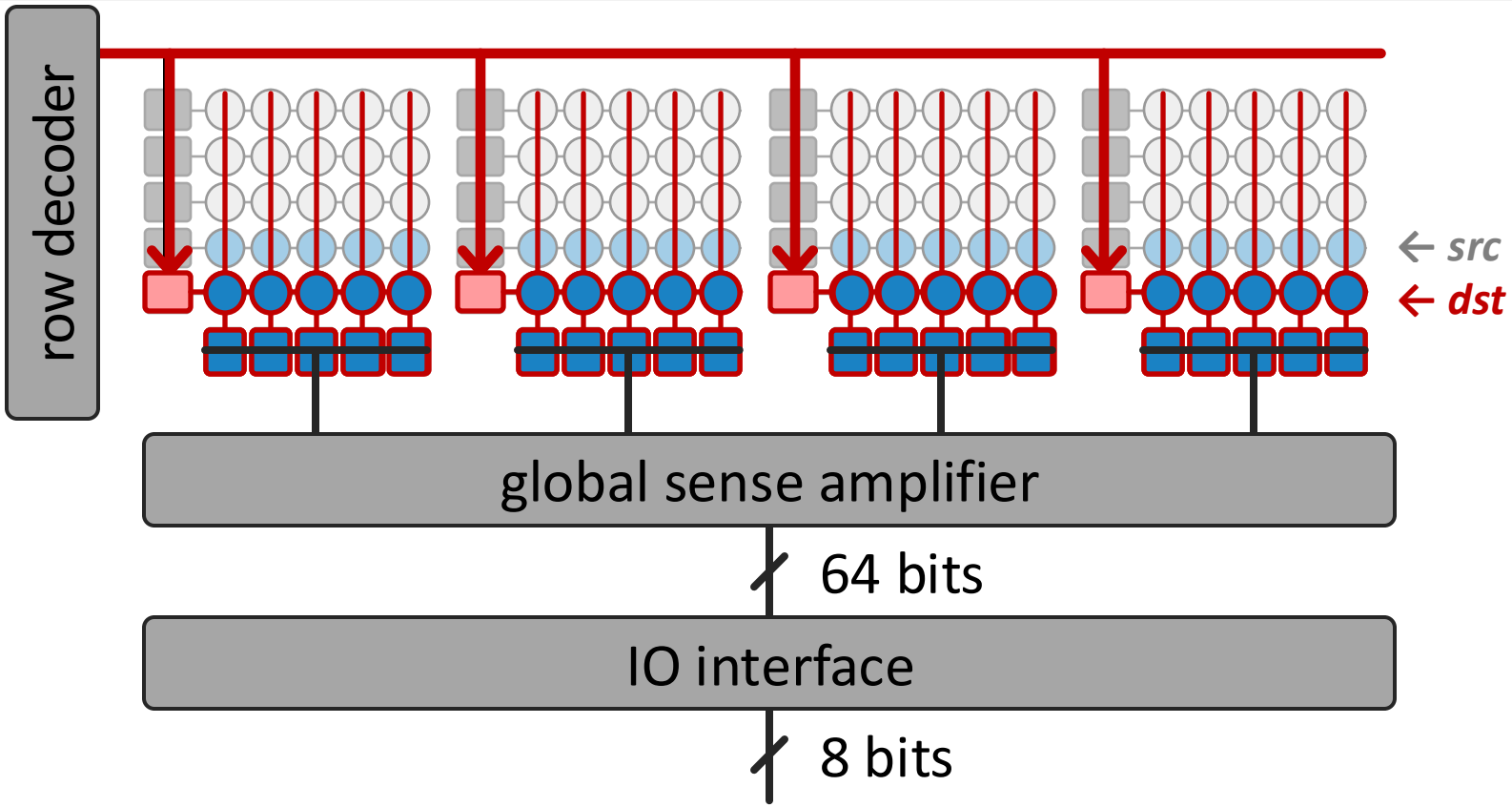


Seshadri, Vivek, et al. "RowClone: Fast and Energy-Efficient In-DRAM Bulk Data Copy and Initialization," in MICRO, 2013

Background:
In-DRAM Row Copy

In-DRAM row copy improves performance & energy:

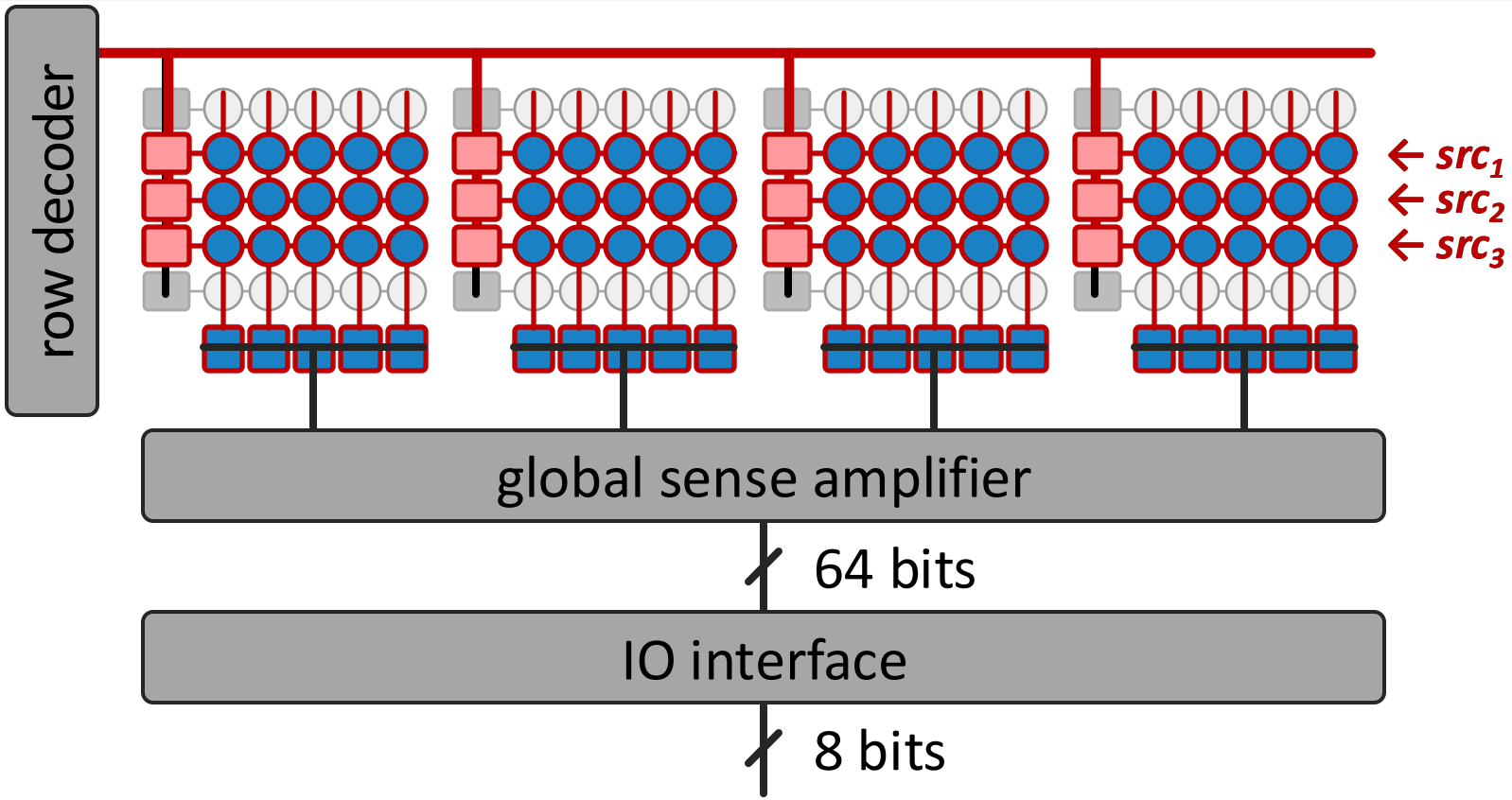
1046 ns, 3.6 uJ → 90 ns, 0.04 nJ



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

Background:
In-DRAM Majority Operations

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

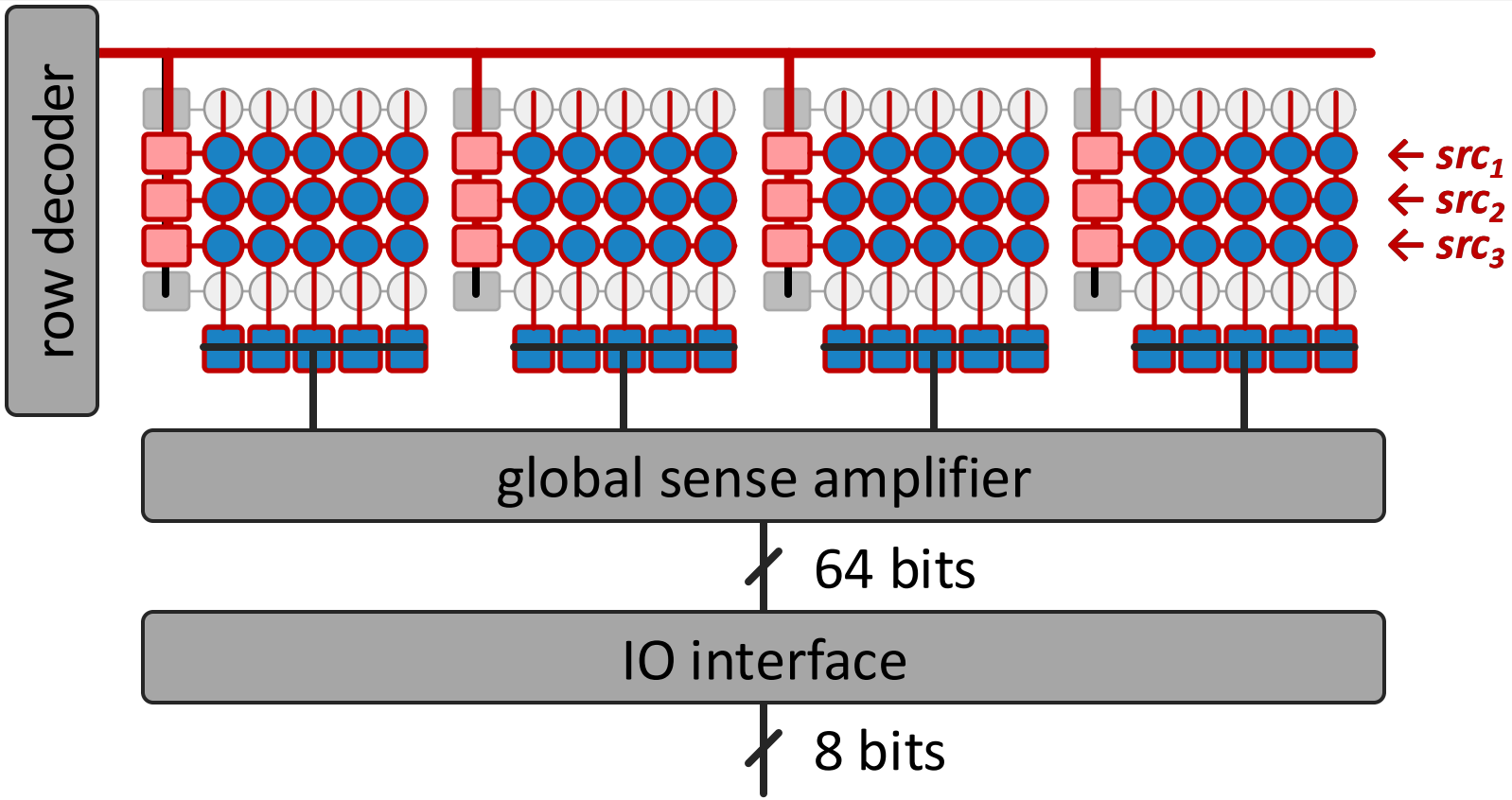


Seshadri, Vivek, et al. " **Ambit: In-Memory Accelerator for Bulk Bitwise Operations Using Commodity DRAM Technology**," in MICRO, 2017

Background:
In-DRAM Majority Operations

In-DRAM majority reduces energy:

137.9 nJ/KB → 3.2 nJ/KB



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

Background:
PUD in Commodity Off-the-Shelf (COTS) DRAM

Commodity off-the-shelf DRAM chips can
perform bulk bitwise operations without hardware modifications

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

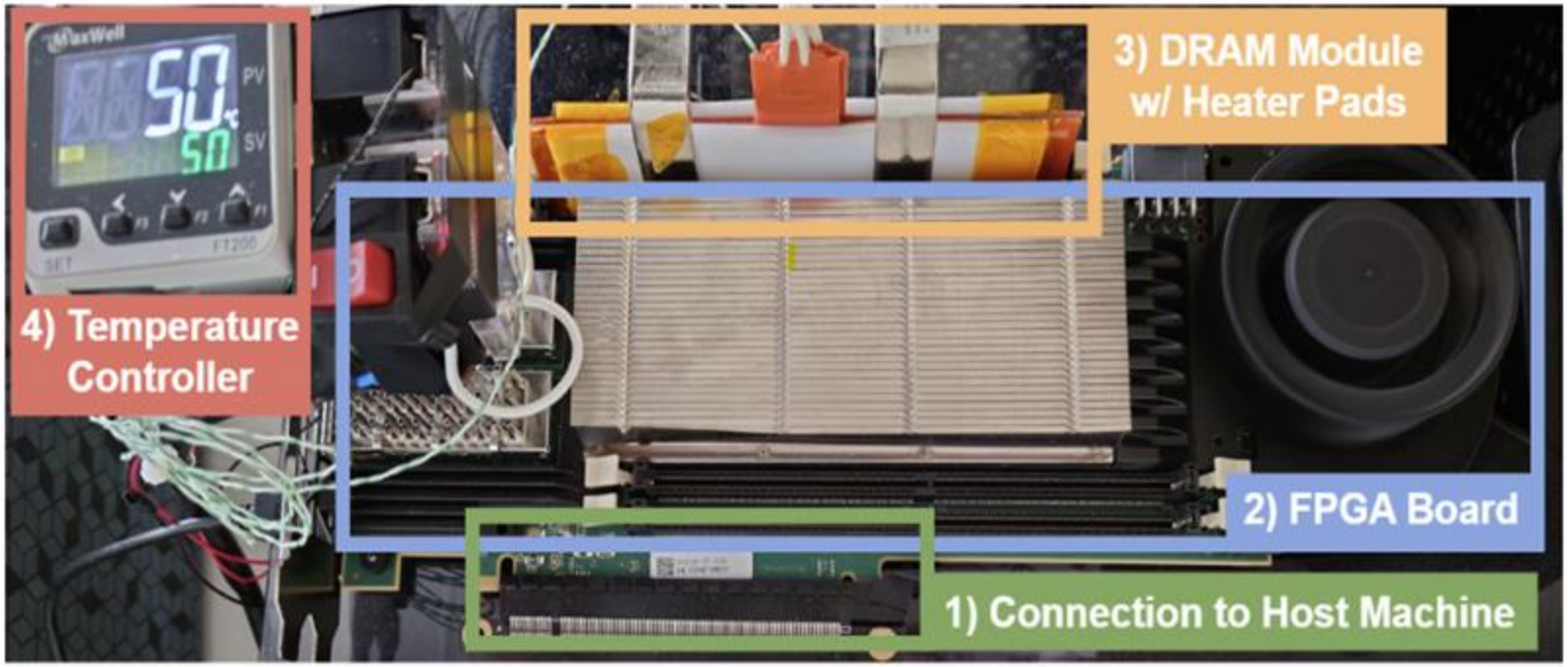
PUD in COTS DRAM:
Summary of Results

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

PUD in COTS DRAM:
Experimental Setup

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



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

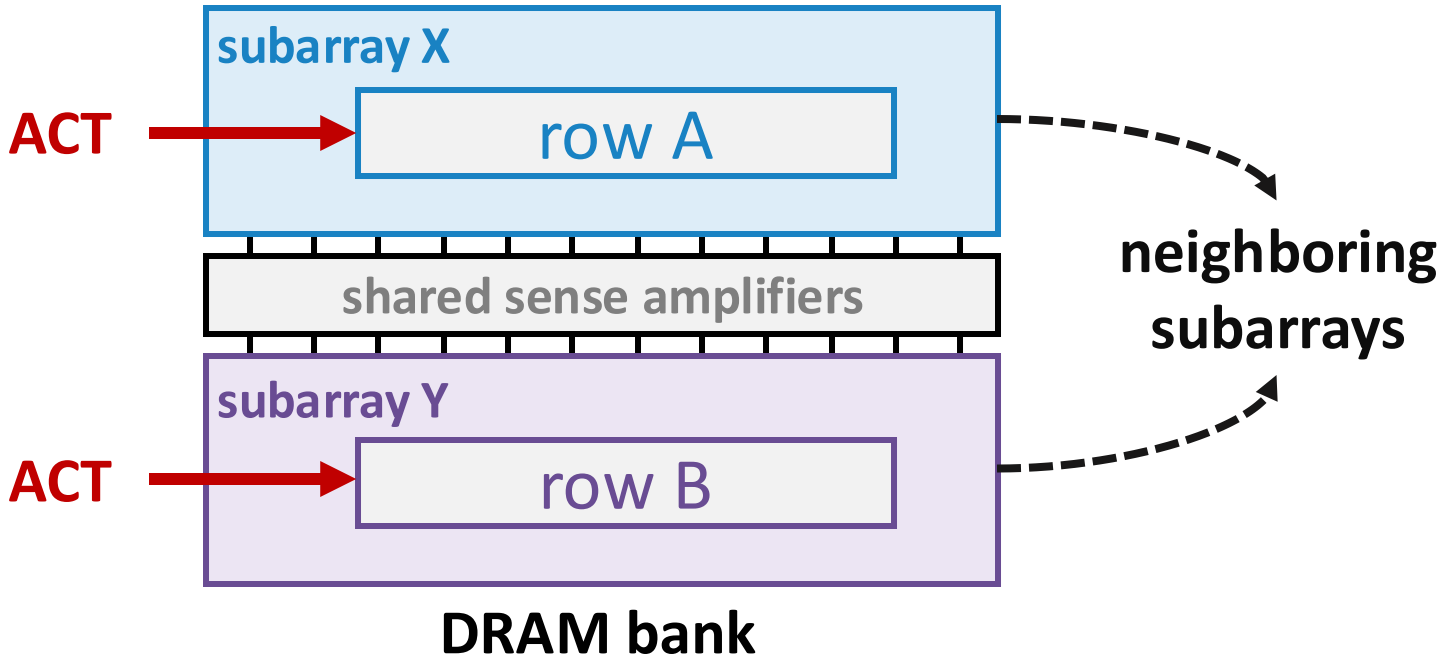
PUD in COTS DRAM:
Tested DRAM Chips

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

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

PUD in COTS DRAM:
Testing Methodology

- **Carefully sweep:**
  - **Row addresses:** Row A and Row B
  - **Timing parameters:** Between ACT → PRE and PRE → ACT



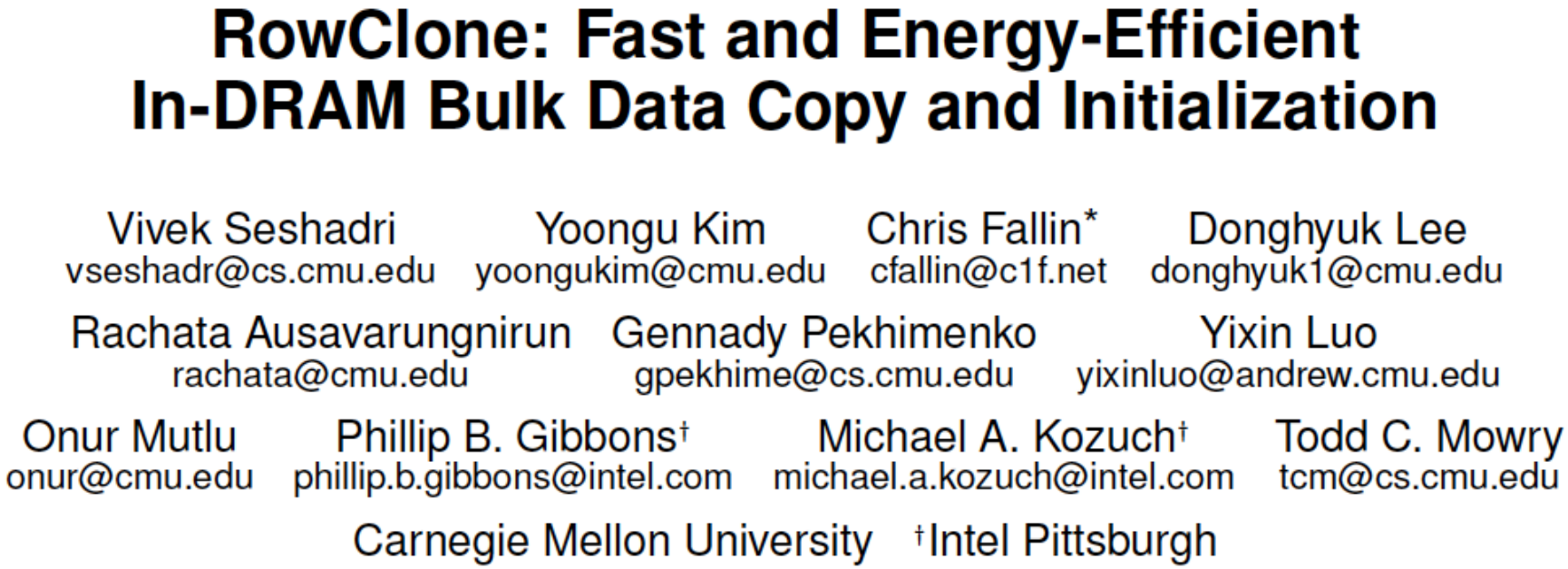
PUD in COTS DRAM:
Conclusion

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

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

More on: RowClone

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



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

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

More on: In-DRAM Bulk AND/OR

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


Fast Bulk Bitwise AND and OR in DRAM

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

*Carnegie Mellon University †Intel Pittsburgh

More on: Ambit

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

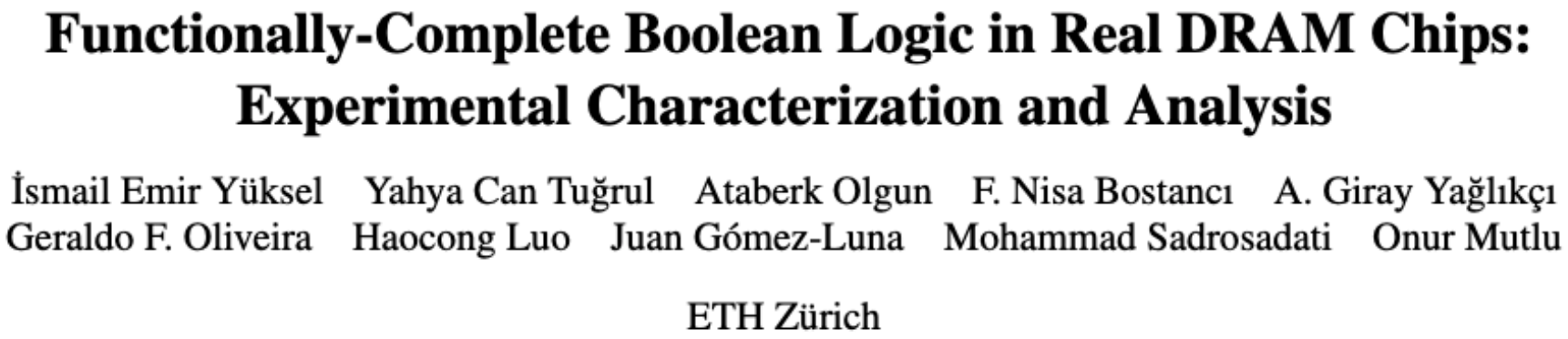
Ambit: In-Memory Accelerator for Bulk Bitwise Operations Using Commodity DRAM Technology

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

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

More on: Functionally-Complete DRAM

- Ismail Emir Yuksel, Yahya Can Tugrul, Ataberk Olgun, F. Nisa Bostanci, A. Giray Yaglikci, Geraldo F. Oliveira, Haocong Luo, Juan Gomez-Luna, Mohammad Sadrosadati, and Onur Mutlu,
  "Functionally-Complete Boolean Logic in Real DRAM Chips: Experimental Characterization and Analysis"
  *Proceedings of the 30th International Symposium on High-Performance Computer Architecture (**HPCA**)*, April 2024.



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-in-Memory: Challenges

To fully support PIM systems, we need to develop:

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

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

Processing-in-Memory:
In This Talk

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

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

Frameworks for PIM:
SIMDRAM

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

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

Our Goal

Goal

Design a PUM framework that

(1) Efficiently implements complex operations

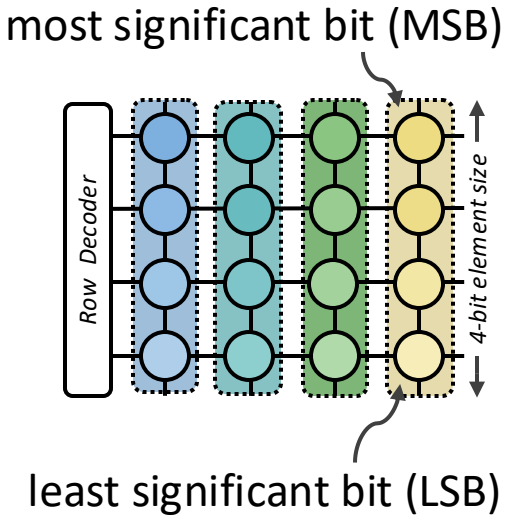
(2) Flexibility supports new PUM operations

(3) Minimally changes the DRAM architecture

SIMDRAM Framework:
PUM Substrate & Key Ideas

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

(1) Vertical data layout

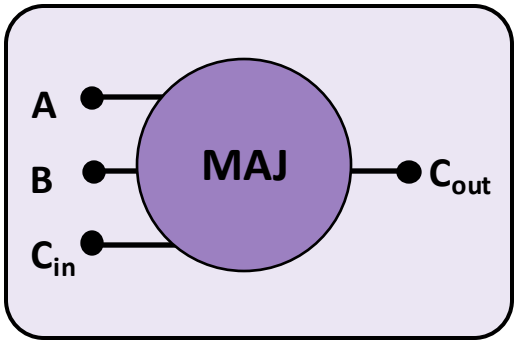


Pros compared to the conventional horizontal layout:

- Implicit shift operation
- Massive parallelism

(2) Majority-based computation

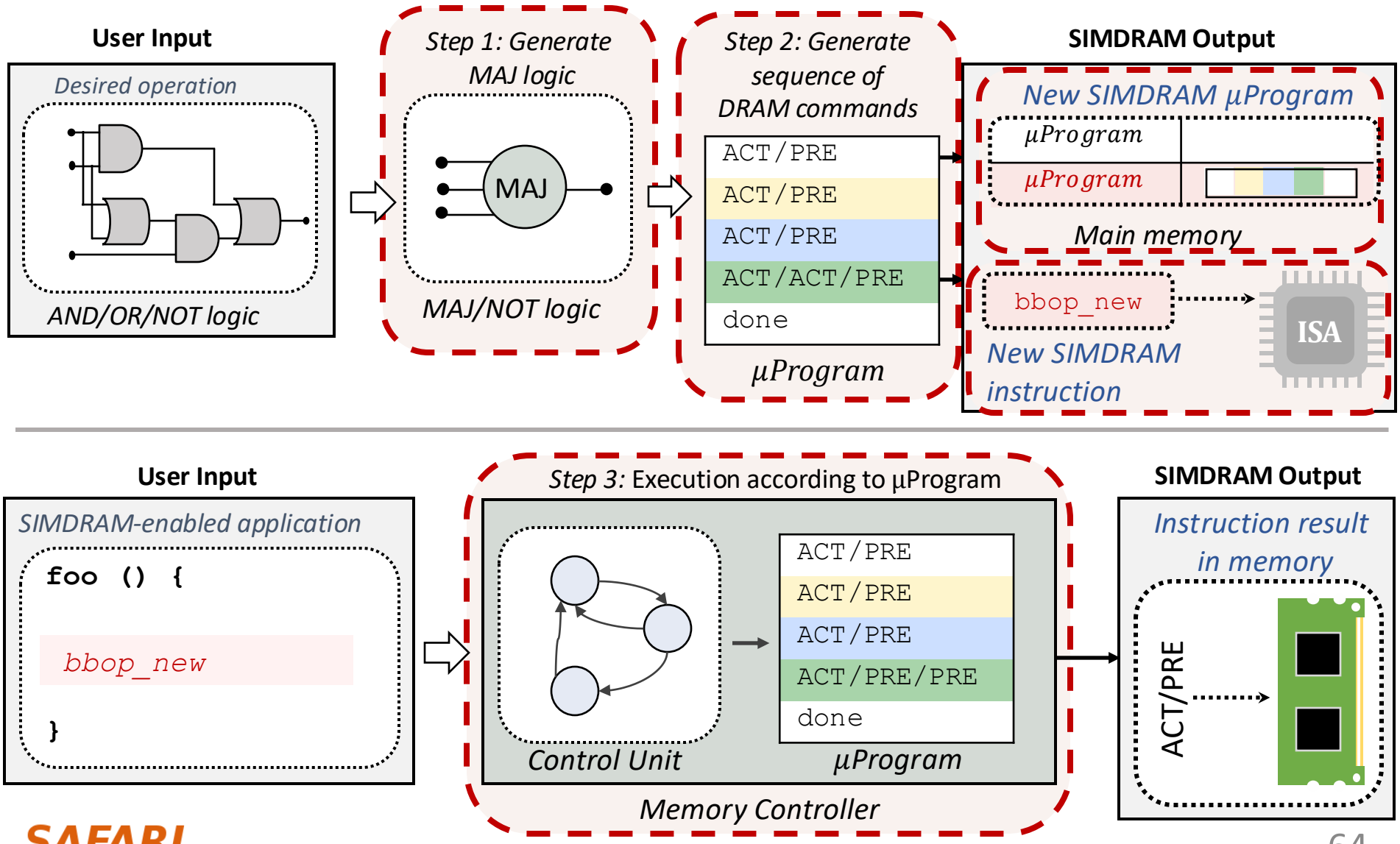
$C_{out} = AB + AC_{in} + BC_{in}$



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

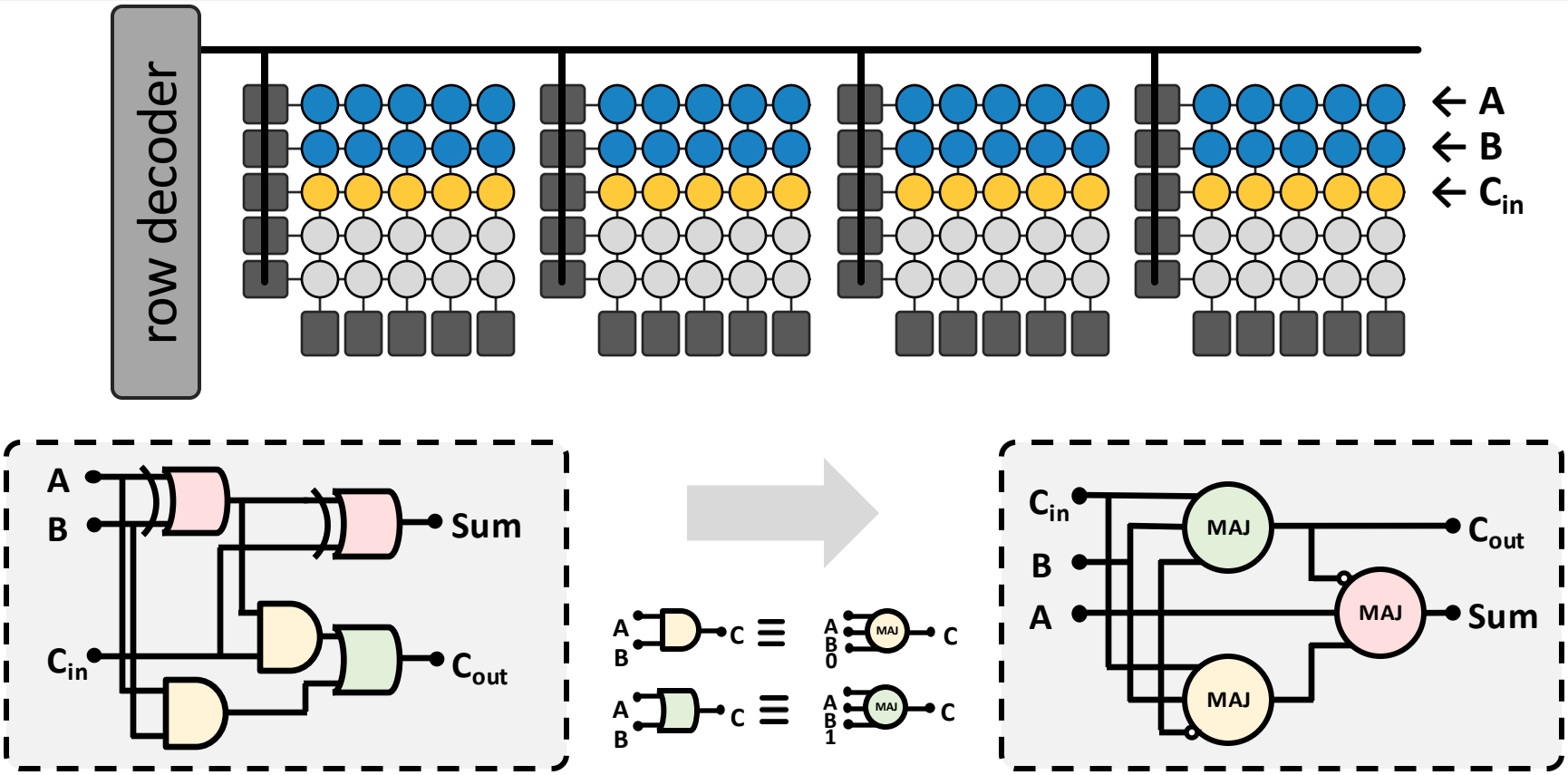
- Higher performance
- Lower energy consumption

SIMDRAM Framework: Overview



Background:
Bulk Bitwise Arithmetic Operations

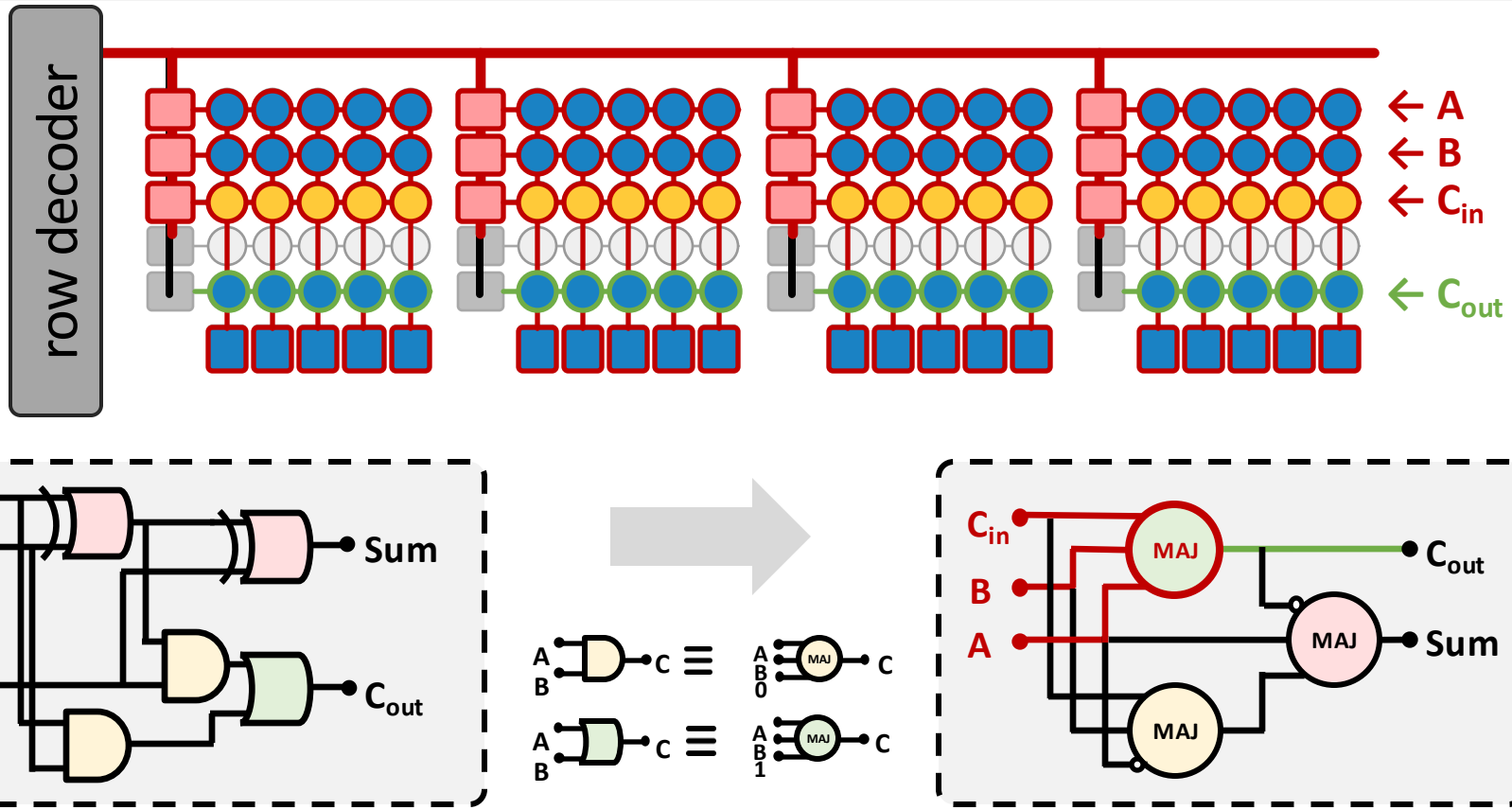
Bulk bitwise arithmetic can be performed by
orchestrating in-DRAM row copy and majority operations



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

Background:
Bulk Bitwise Arithmetic Operations

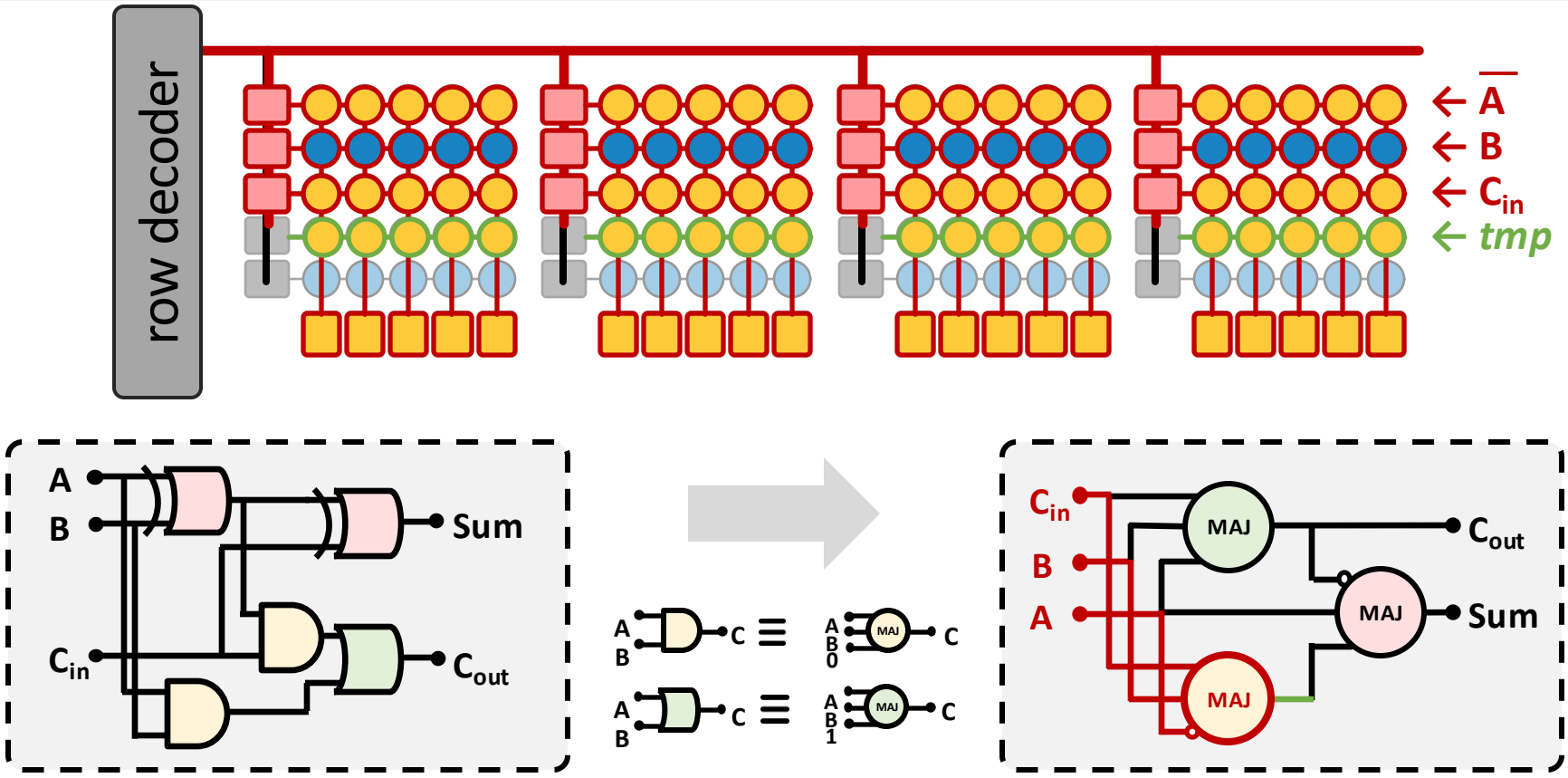
Bulk bitwise arithmetic can be performed by
orchestrating in-DRAM row copy and majority operations



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

Background:
Bulk Bitwise Arithmetic Operations

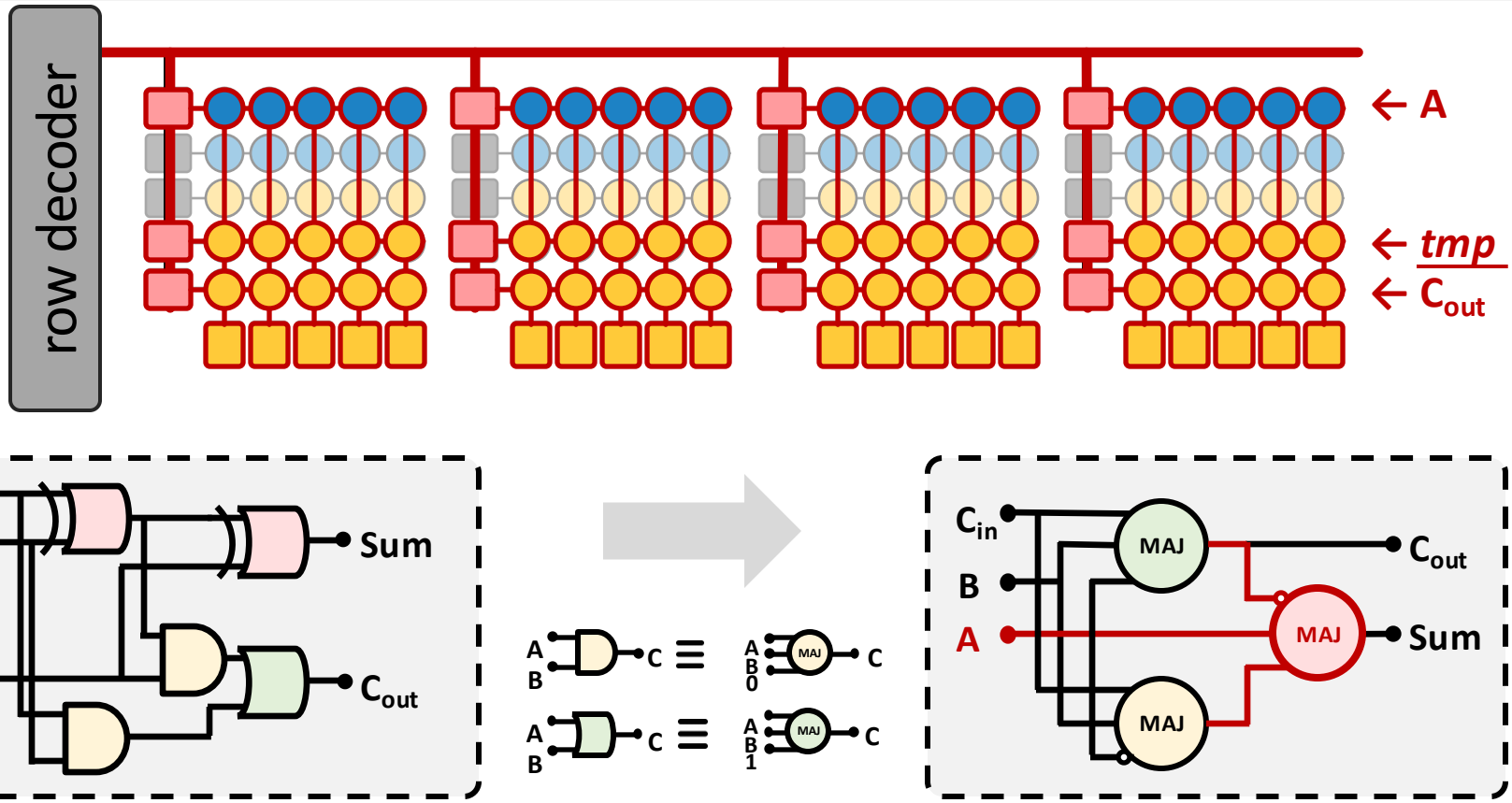
Bulk bitwise arithmetic can be performed by
orchestrating in-DRAM row copy and majority operations



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

Background:
Bulk Bitwise Arithmetic Operations

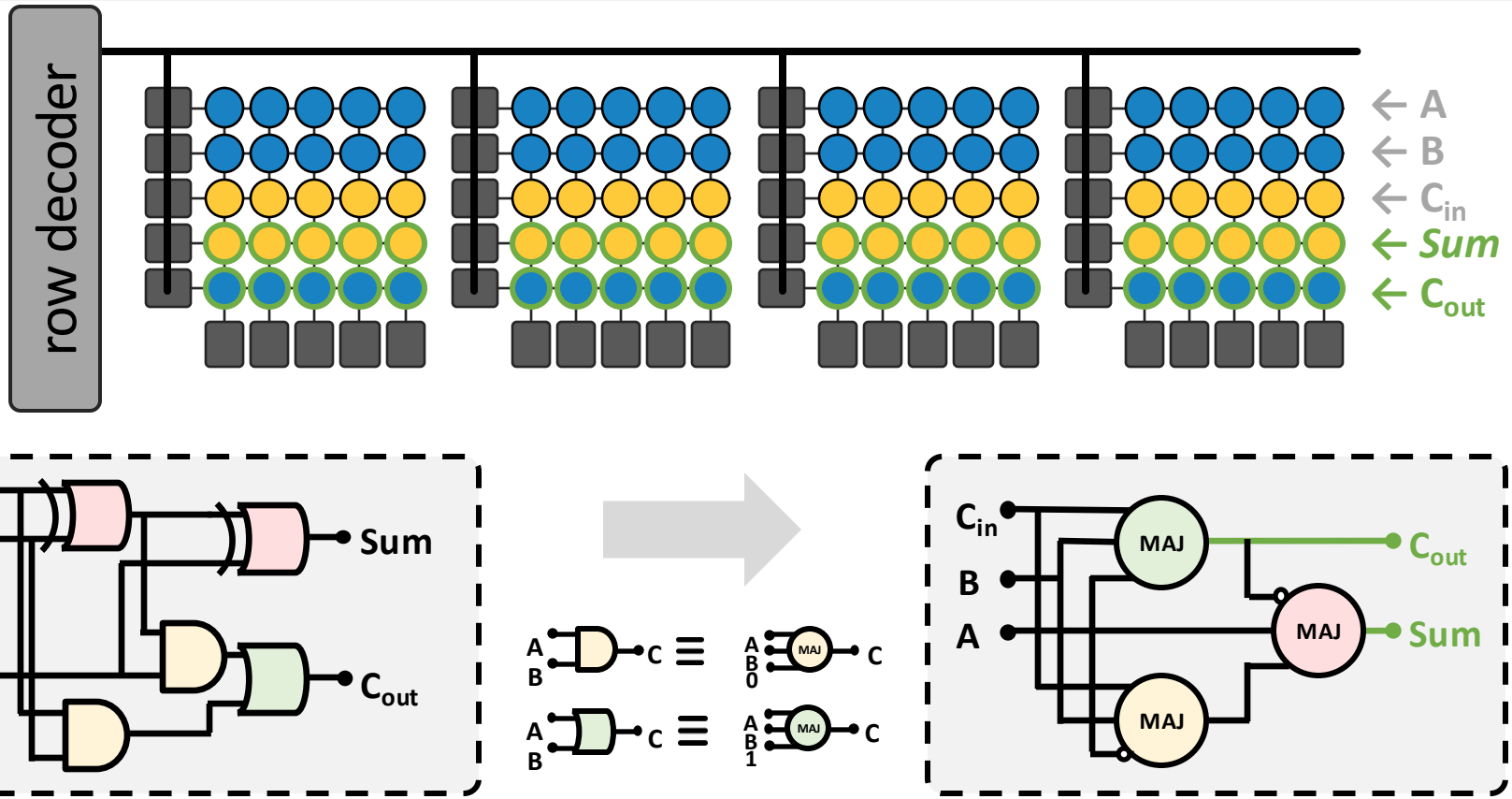
Bulk bitwise arithmetic can be performed by
orchestrating in-DRAM row copy and majority operations



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

Background:
Bulk Bitwise Arithmetic Operations

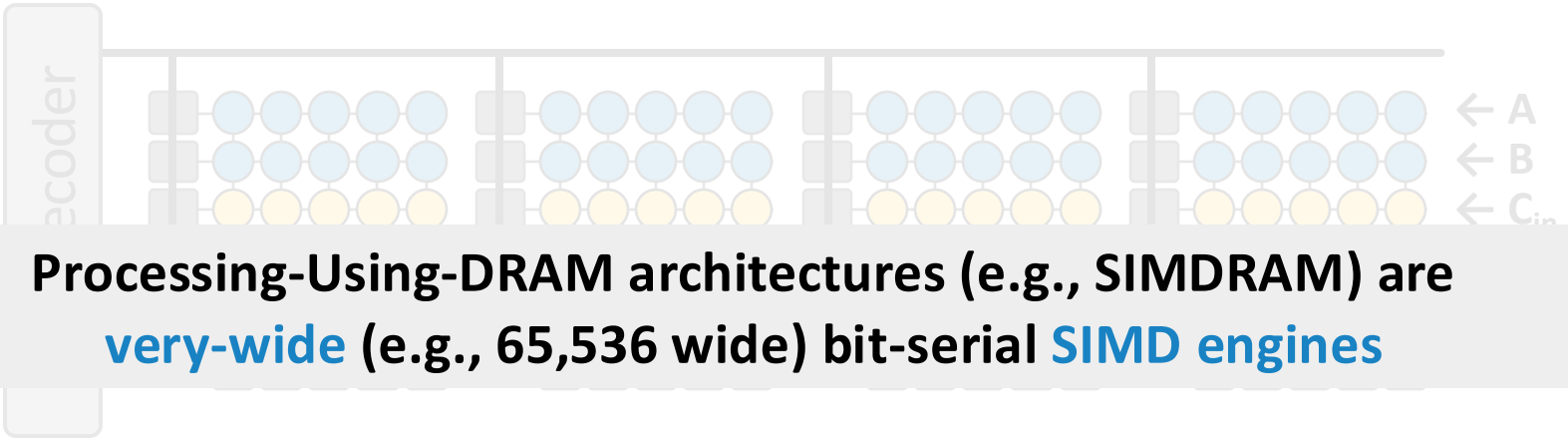
Bulk bitwise arithmetic can be performed by
orchestrating in-DRAM row copy and majority operations



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

Background:
Bulk Bitwise Arithmetic Operations

Bulk bitwise arithmetic can be performed by orchestrating in-DRAM row copy and majority operations



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

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

Evaluation:
Methodology Overview (I)

- **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

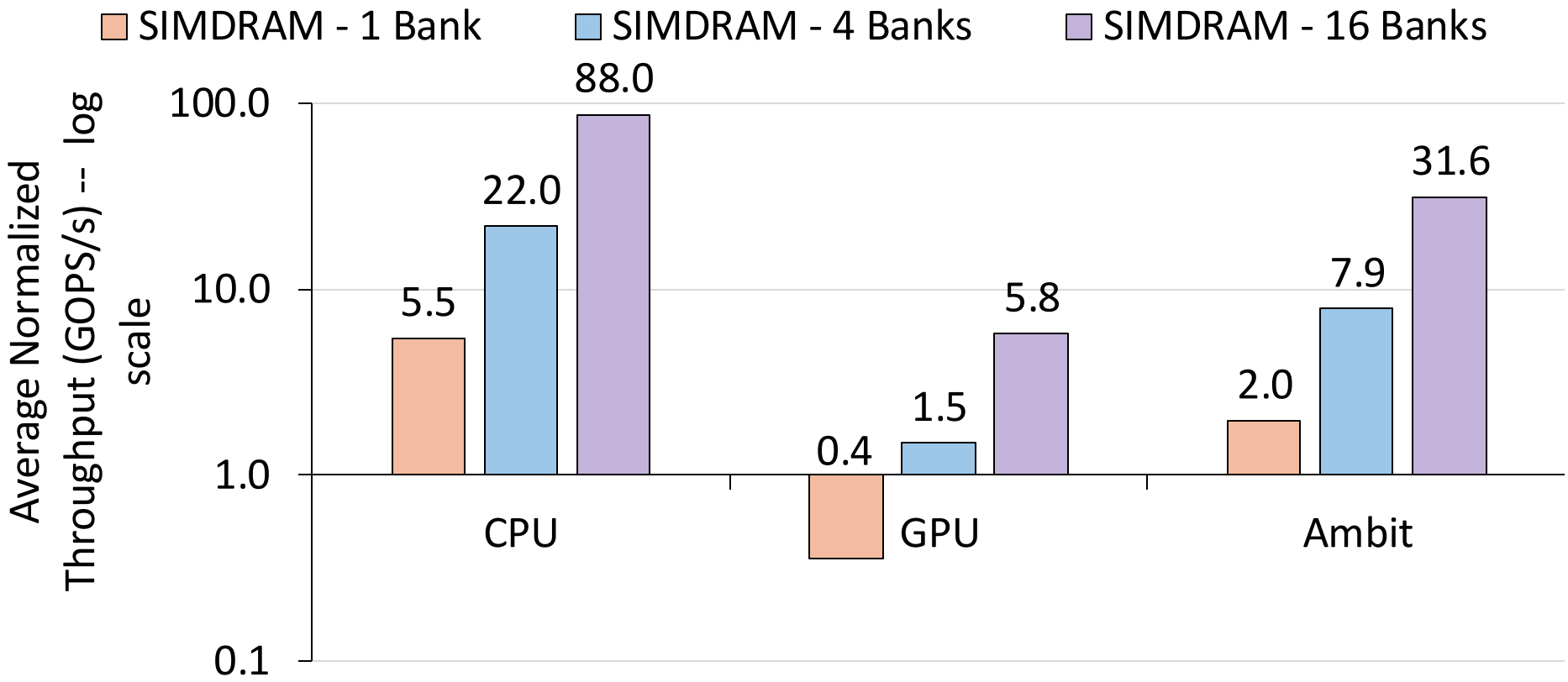
Evaluation:
Methodology Overview (II)

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

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

Evaluation:
Throughput Analysis

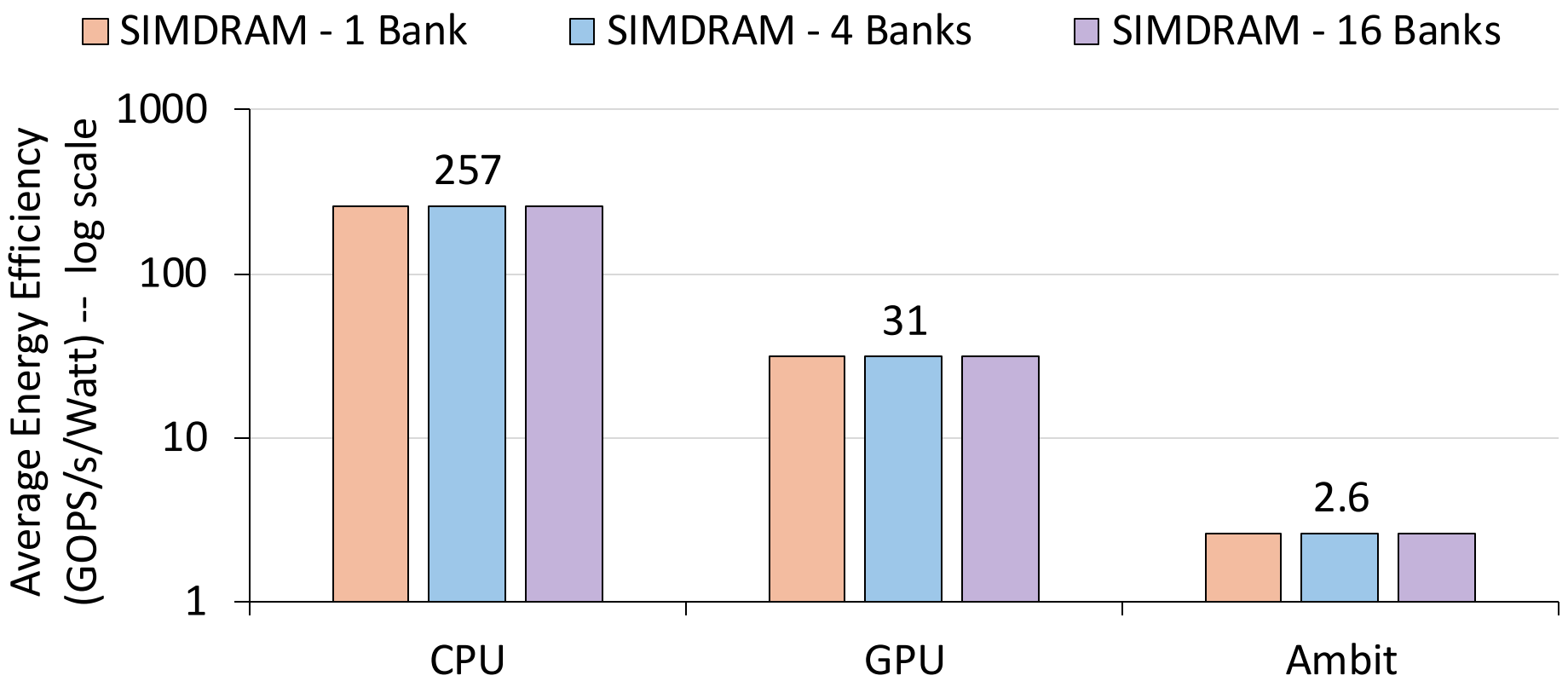
Average normalized throughput across all 16 SIMDRAM operations



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

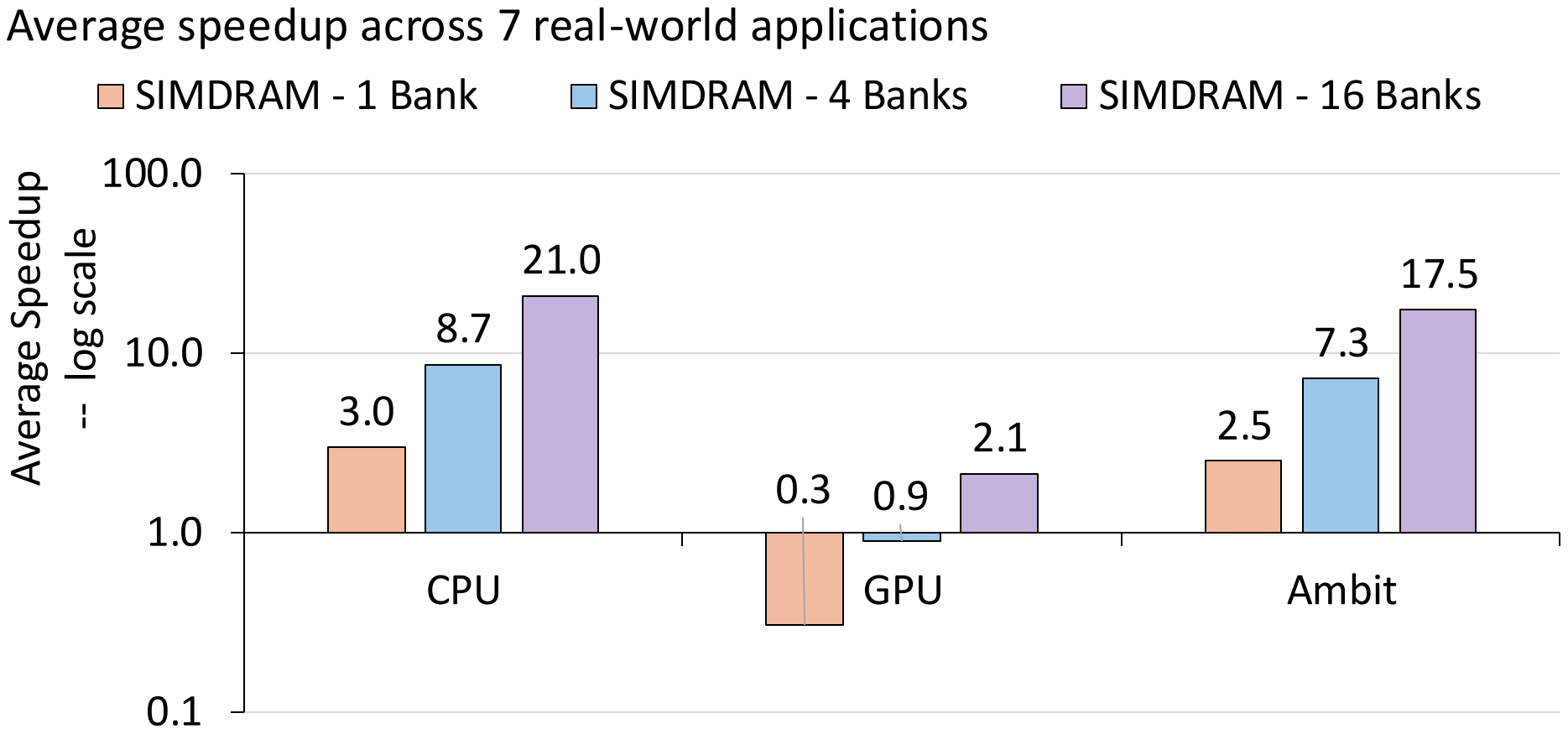
Evaluation:
Energy Analysis

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

Evaluation:
Real-World Applications



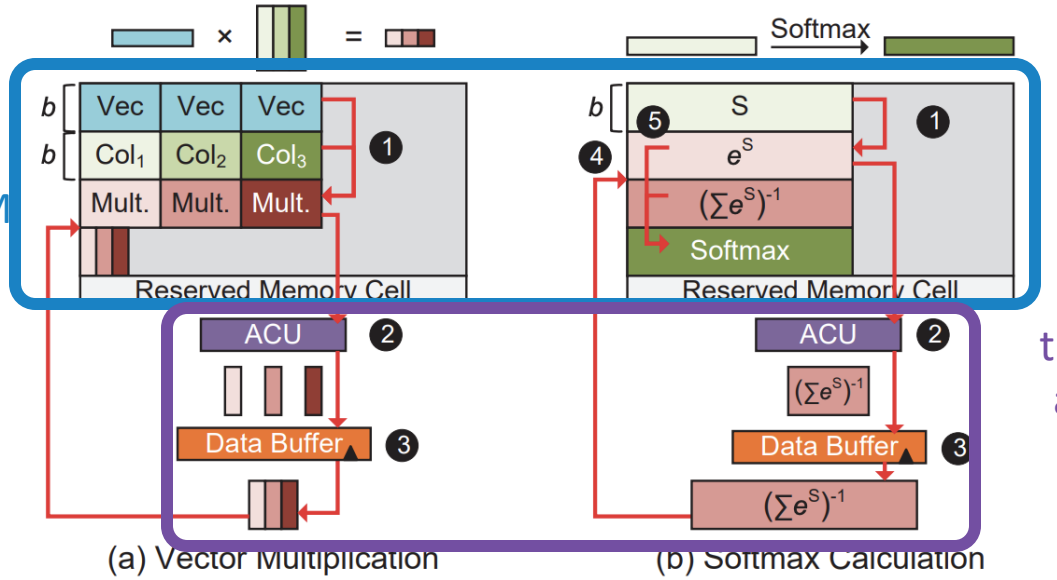
SIMDRAM effectively and efficiently accelerates many commonly-used real-world applications

SIMDRAM & HBM-PIM:
TransPIM [Zhou+, HPCA'22]

Key Idea:

to combine PUD and PND to achieve high efficiency and throughput transformer inference

point-wise vector operations are performed in DRAM cells



the vector reduction and part of softmax are offloaded to a near-DRAM bank accelerator

TransPIM is 22.1x to 114.9x faster than a GPU (Nvidia RTX 2080Ti) for RoBERTa, Pegasus, and GPT-2 models

Zhou, Minxuan, et al. "TransPIM: A Memory-based Acceleration via Software-Hardware Co-Design for Transformer," in HPCA, 2022

Frameworks for PIM: pLUTo

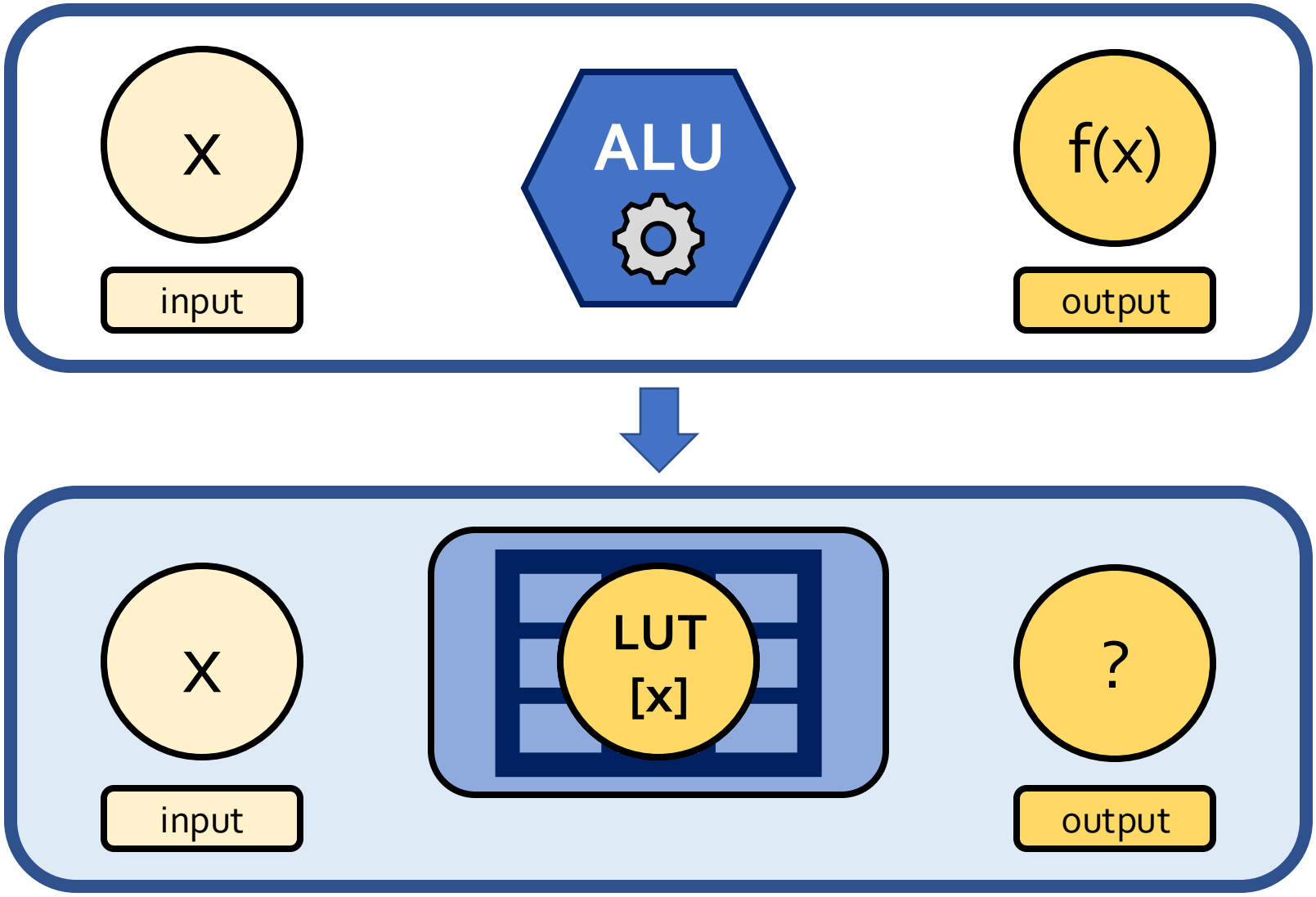
- João Dinis Ferreira, Gabriel Falcao, Juan Gómez-Luna, Mohammed Alser, Lois Orosa, Mohammad Sadrosadati, Jeremie S. Kim, **Geraldo F. Oliveira**, Taha Shahroodi, Anant Nori, and Onur Mutlu,
  "pLUTo: Enabling Massively Parallel Computation in DRAM via Lookup Tables"
  *Proceedings of the 55th International Symposium on Microarchitecture (**MICRO**)*, Chicago, IL, USA, October 2022.

pLUTo: Enabling Massively Parallel Computation in DRAM via Lookup Tables

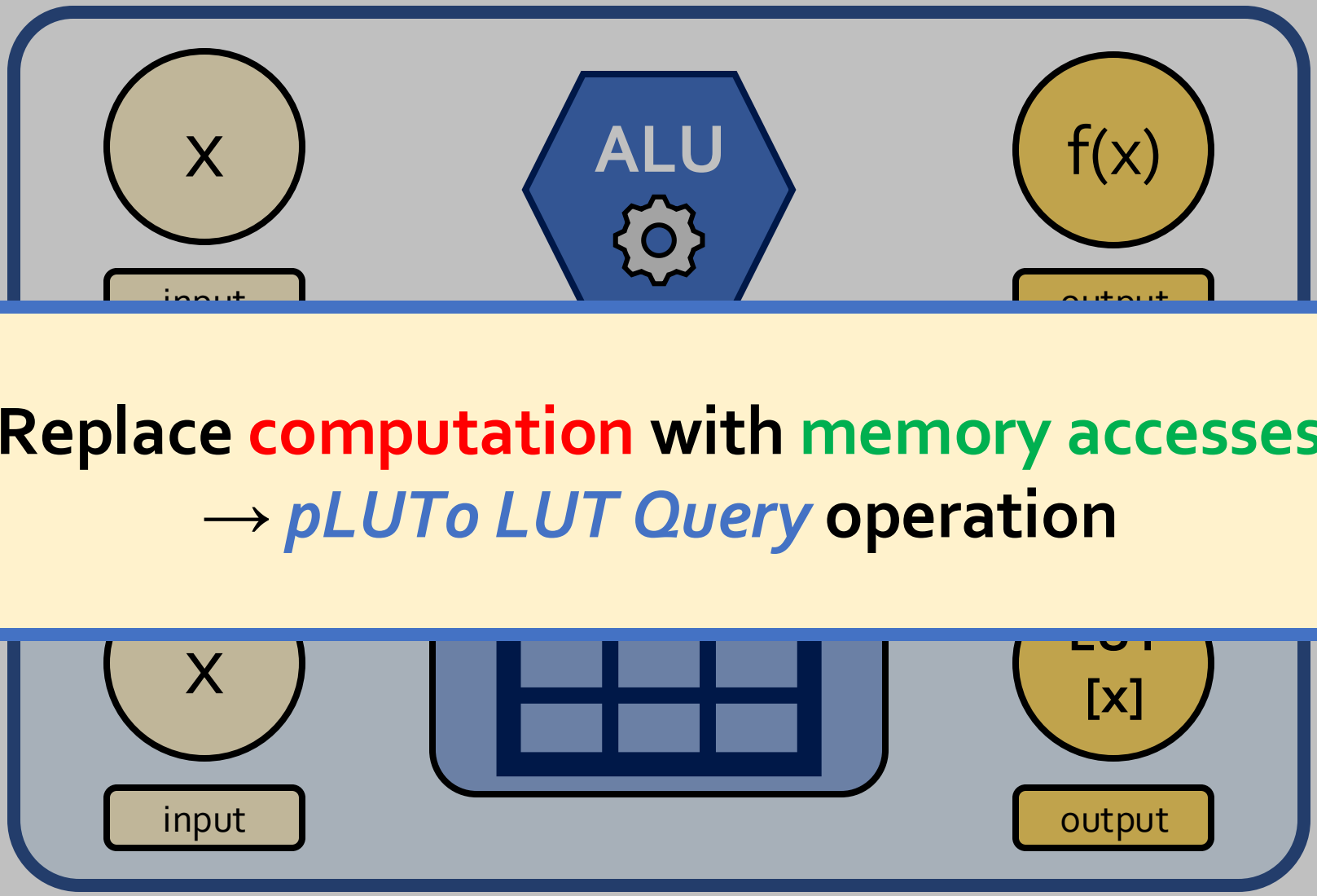
João Dinis Ferreira[§] Gabriel Falcao[†] Juan Gómez-Luna[§] Mohammed Alser[§]
Lois Orosa[§▽] Mohammad Sadrosadati[§] Jeremie S. Kim[§] Geraldo F. Oliveira[§]
Taha Shahroodi[‡] Anant Nori[⋆] Onur Mutlu[§]

[§]*ETH Zürich* [†]*IT, University of Coimbra* [▽]*Galicia Supercomputing Center* [‡]*TU Delft* [⋆]*Intel*

pLUTo: Key Idea



pLUTo: Key Idea



In-DRAM pLUTo LUT Query: Setup

LUT Query:
Return {2nd, 1st, 2nd, 4th}
prime numbers

	Prime numbers	
LUT index	i	f(i)
0	1st	2
1	2nd	3
2	3rd	5
3	4th	7

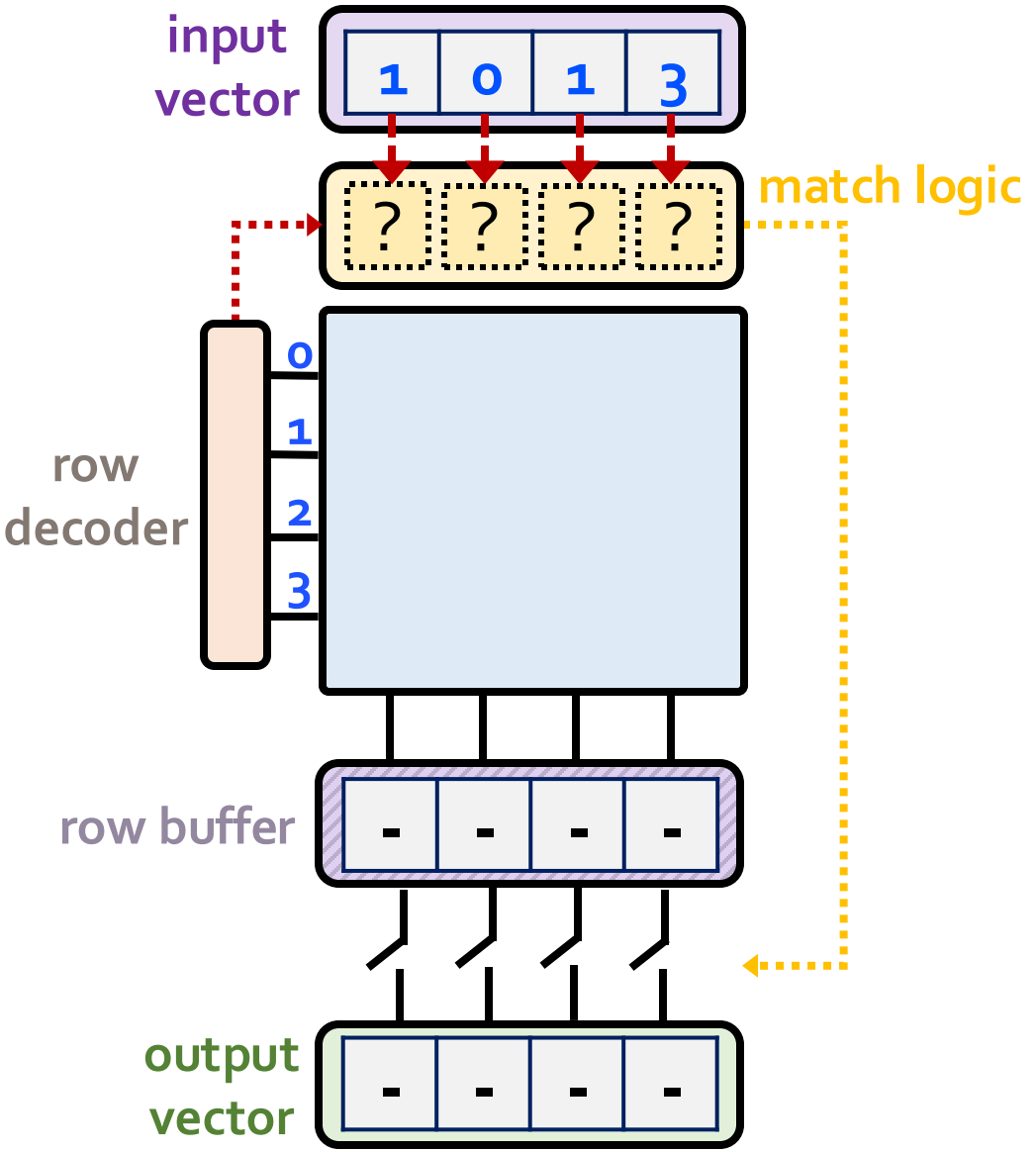
lookup table

1	0	1	3

input vector

3	2	3	7

output vector



In-DRAM pLUTo LUT Query: Setup

LUT Query:
Return {2nd, 1st, 2nd, 4th}
prime numbers

LUT index	Prime numbers i	Prime numbers f(i)
0	1st	2
1	2nd	3
2	3rd	5
3	4th	7

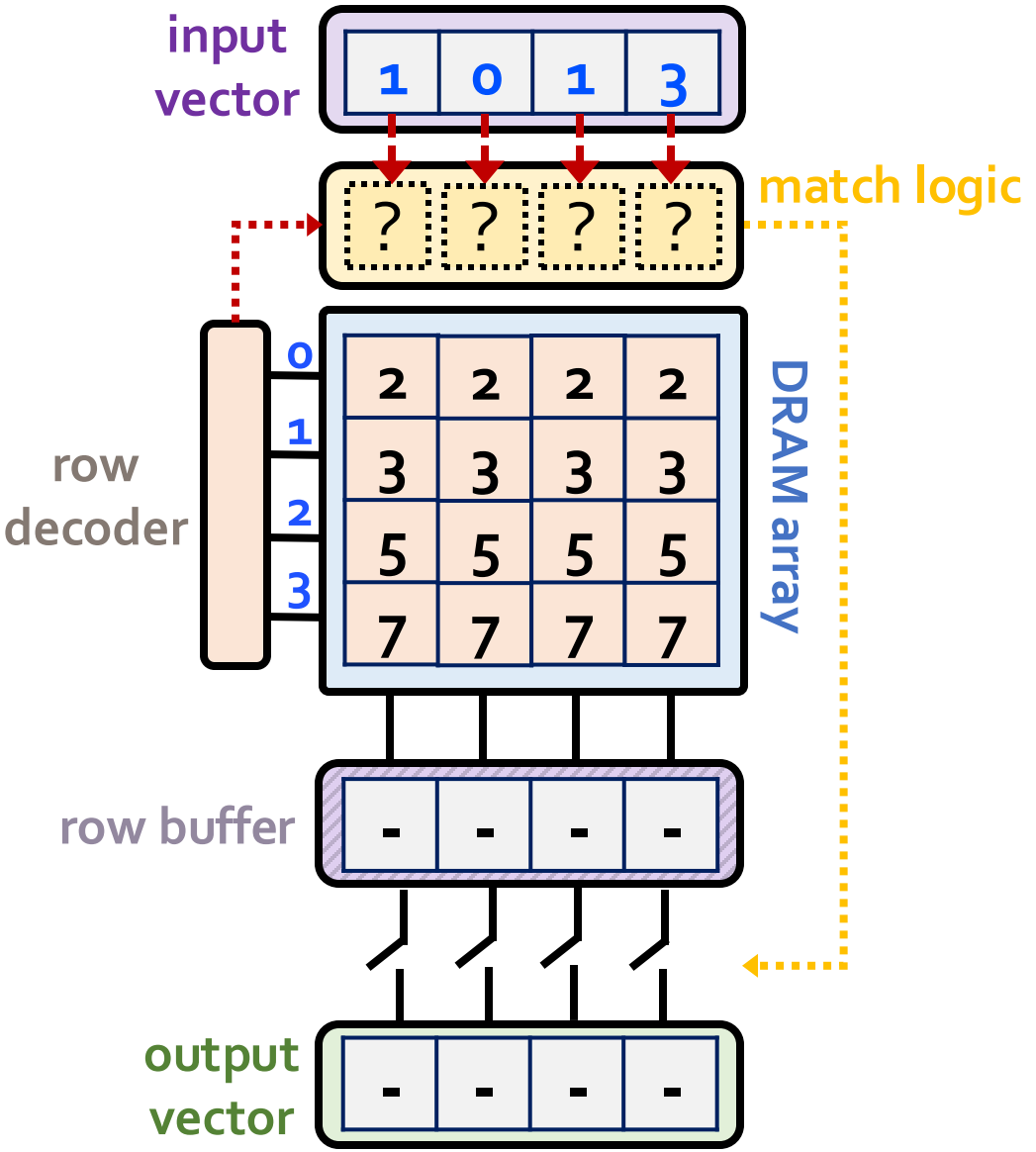
lookup table

1	0	1	3

input vector

3	2	3	7

output vector



In-DRAM pLUTo LUT Query: Step 1

LUT Query:
Return {2nd, 1st, 2nd, 4th}
prime numbers

LUT index	Prime numbers: i	Prime numbers: f(i)
0	1st	2
1	2nd	3
2	3rd	5
3	4th	7

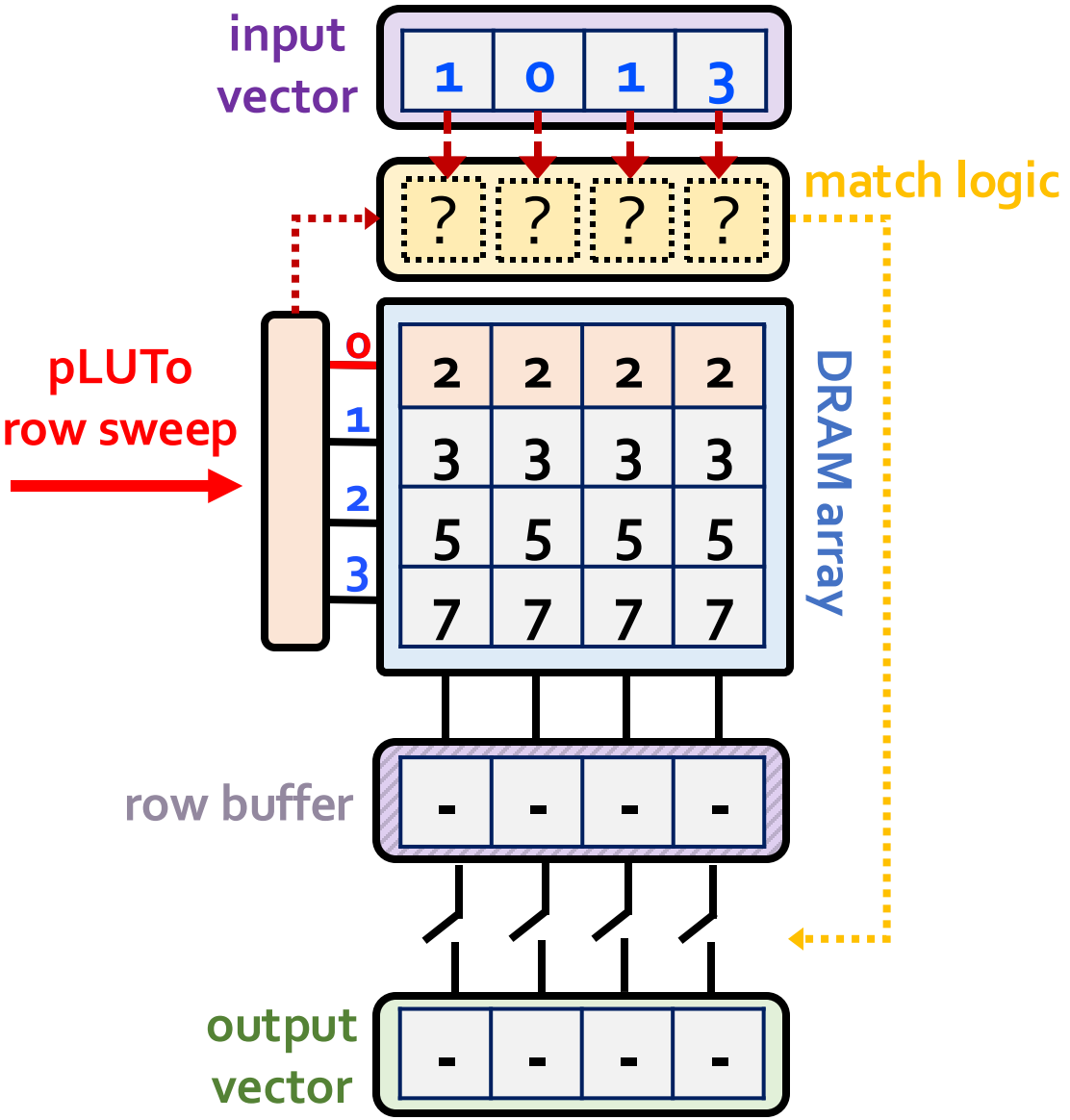
lookup table

1	0	1	3

input vector

3	2	3	7

output vector



In-DRAM pLUTo LUT Query: Step 1

LUT Query:
Return {2nd, 1st, 2nd, 4th}
prime numbers

LUT index	Prime numbers: i	Prime numbers: f(i)
0	1st	2
1	2nd	3
2	3rd	5
3	4th	7

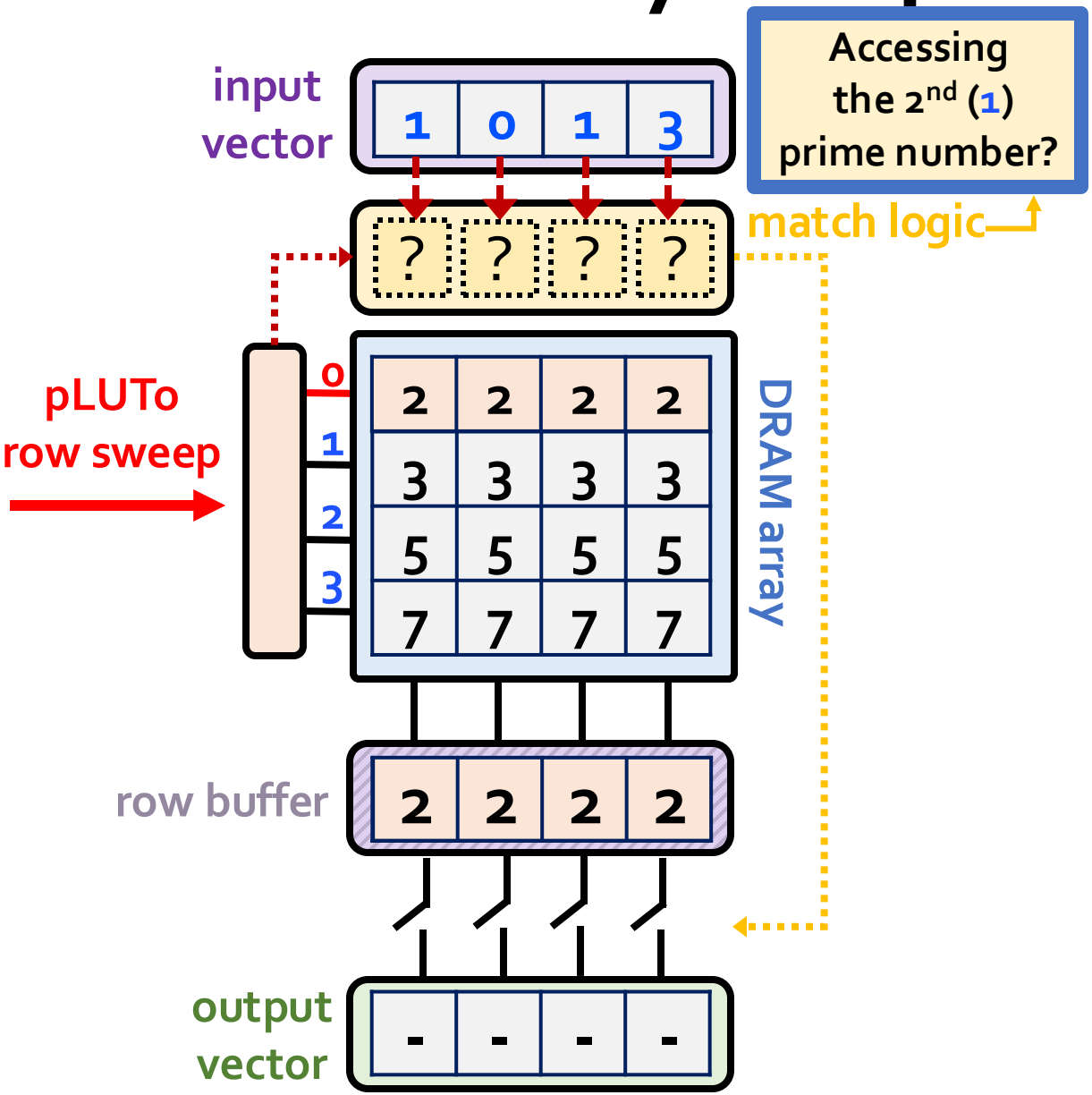
lookup table

1	0	1	3

input vector

3	2	3	7

output vector



In-DRAM pLUTo LUT Query: Step 1

LUT Query:
Return {2nd, 1st, 2nd, 4th}
prime numbers

LUT index	Prime numbers i	f(i)
0	1st	2
1	2nd	3
2	3rd	5
3	4th	7

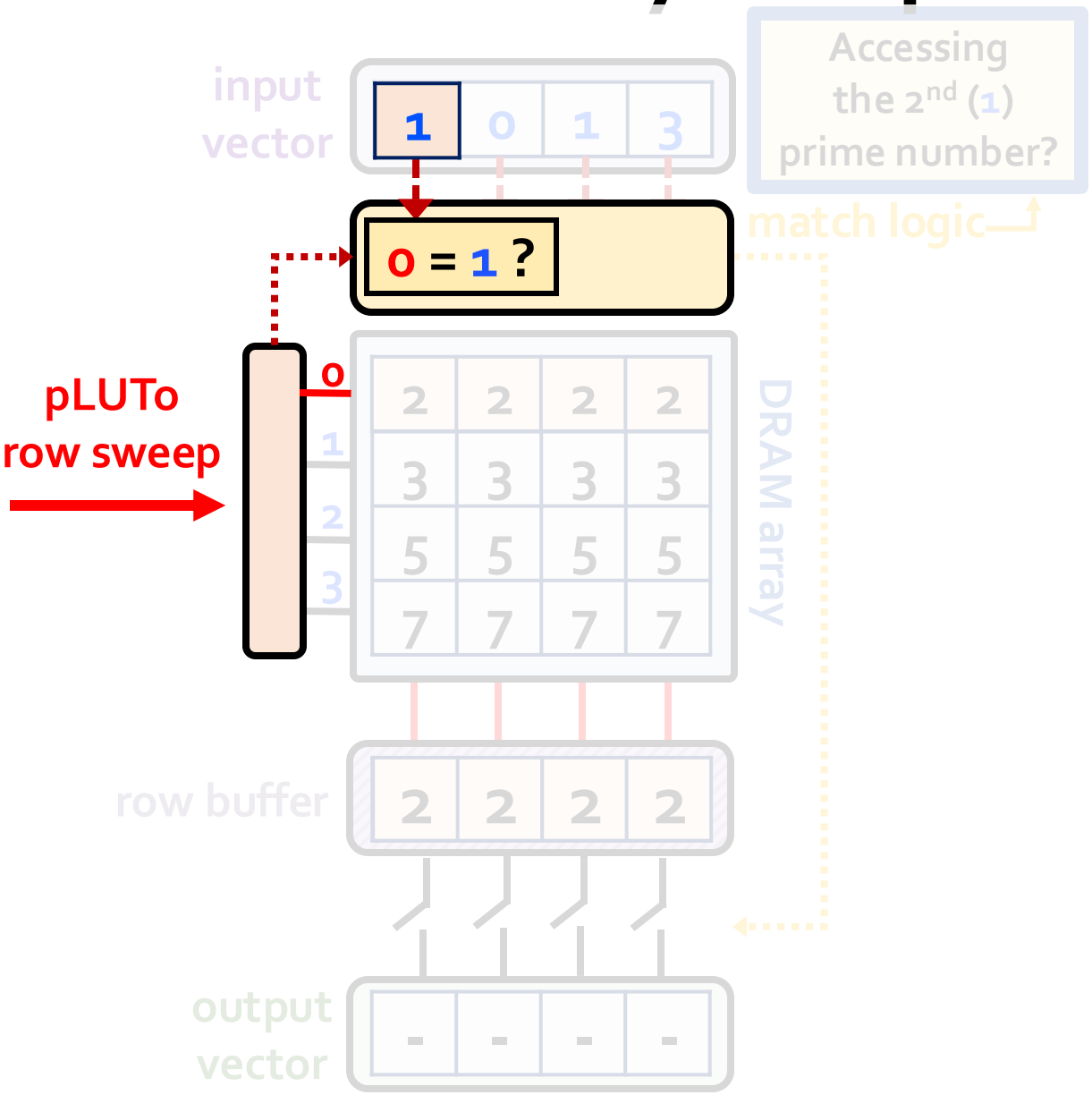
lookup table

1	0	1	3

input vector

3	2	3	7

output vector



In-DRAM pLUTo LUT Query: Step 1

LUT Query:
Return {2nd, 1st, 2nd, 4th}
prime numbers

LUT index	Prime numbers: i	Prime numbers: f(i)
0	1st	2
1	2nd	3
2	3rd	5
3	4th	7

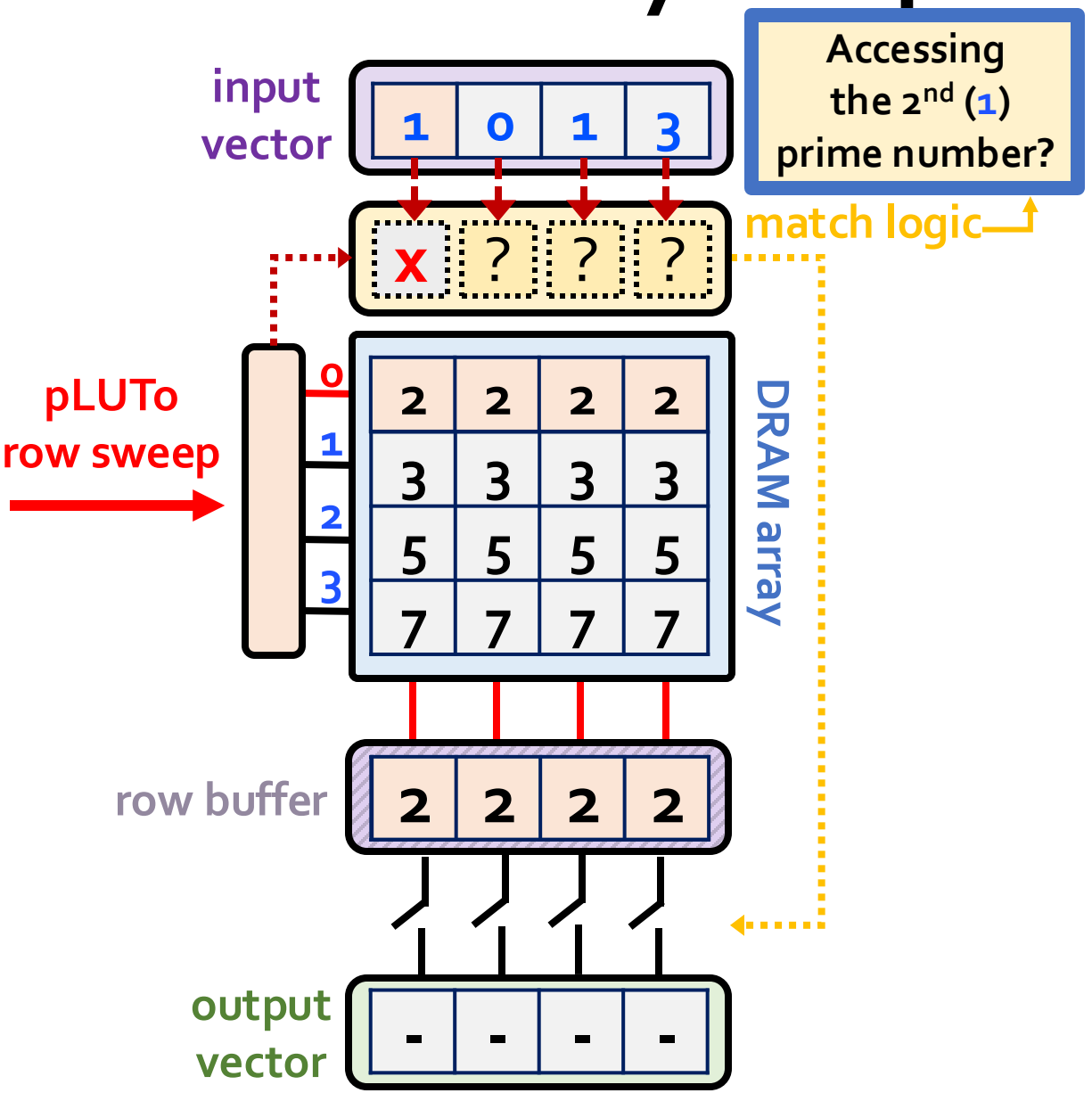
lookup table

1	0	1	3

input vector

3	2	3	7

output vector



In-DRAM pLUTo LUT Query: Step 1

LUT Query:
Return {2nd, 1st, 2nd, 4th}
prime numbers

LUT index	Prime numbers i	f(i)
0	1st	2
1	2nd	3
2	3rd	5
3	4th	7

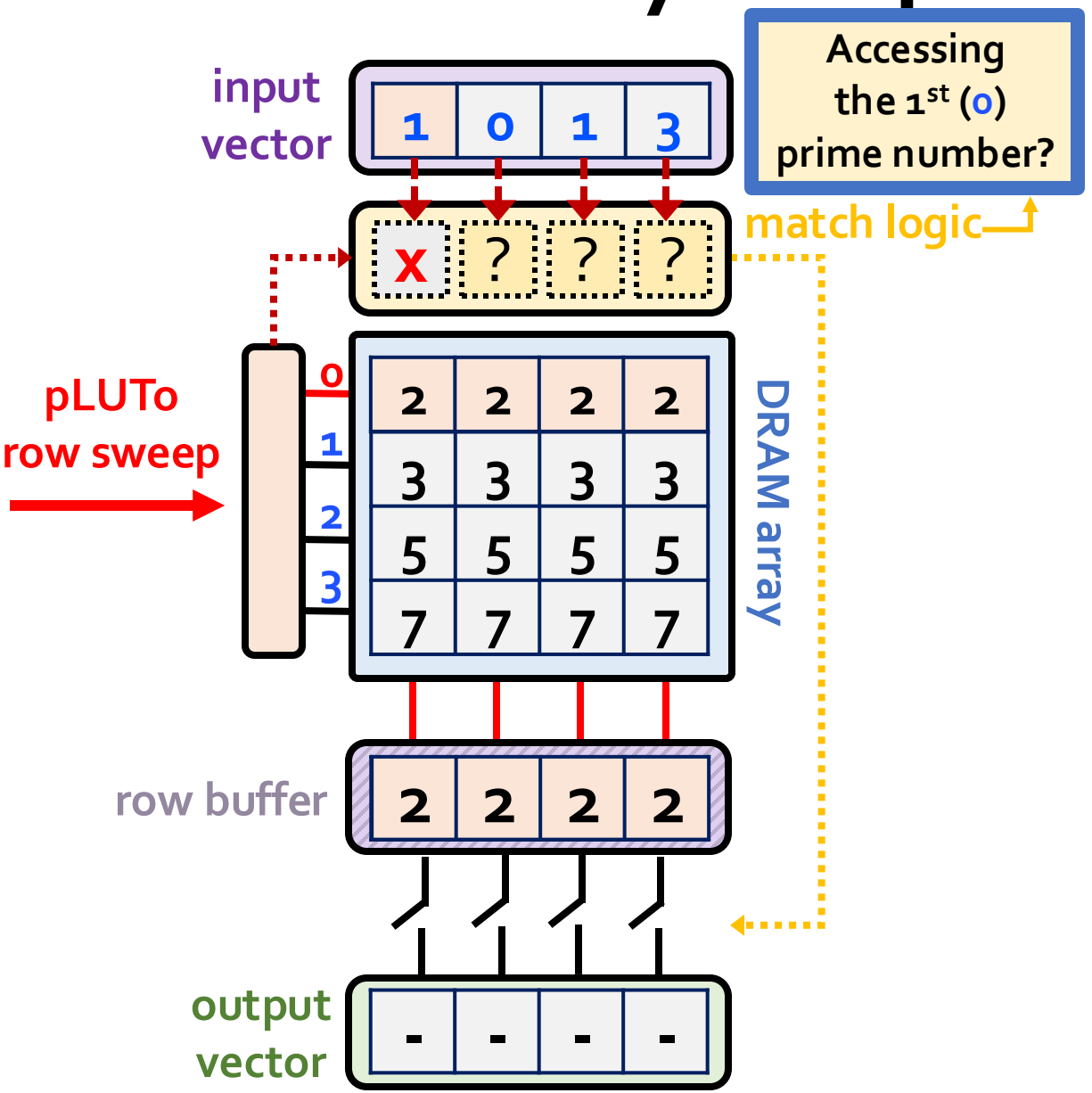
lookup table

1	0	1	3

input vector

3	2	3	7

output vector



In-DRAM pLUTo LUT Query: Step 1

LUT Query:
Return {2nd, 1st, 2nd, 4th} *prime numbers*

LUT index	Prime numbers i	f(i)
0	1st	2
1	2nd	3
2	3rd	5
3	4th	7

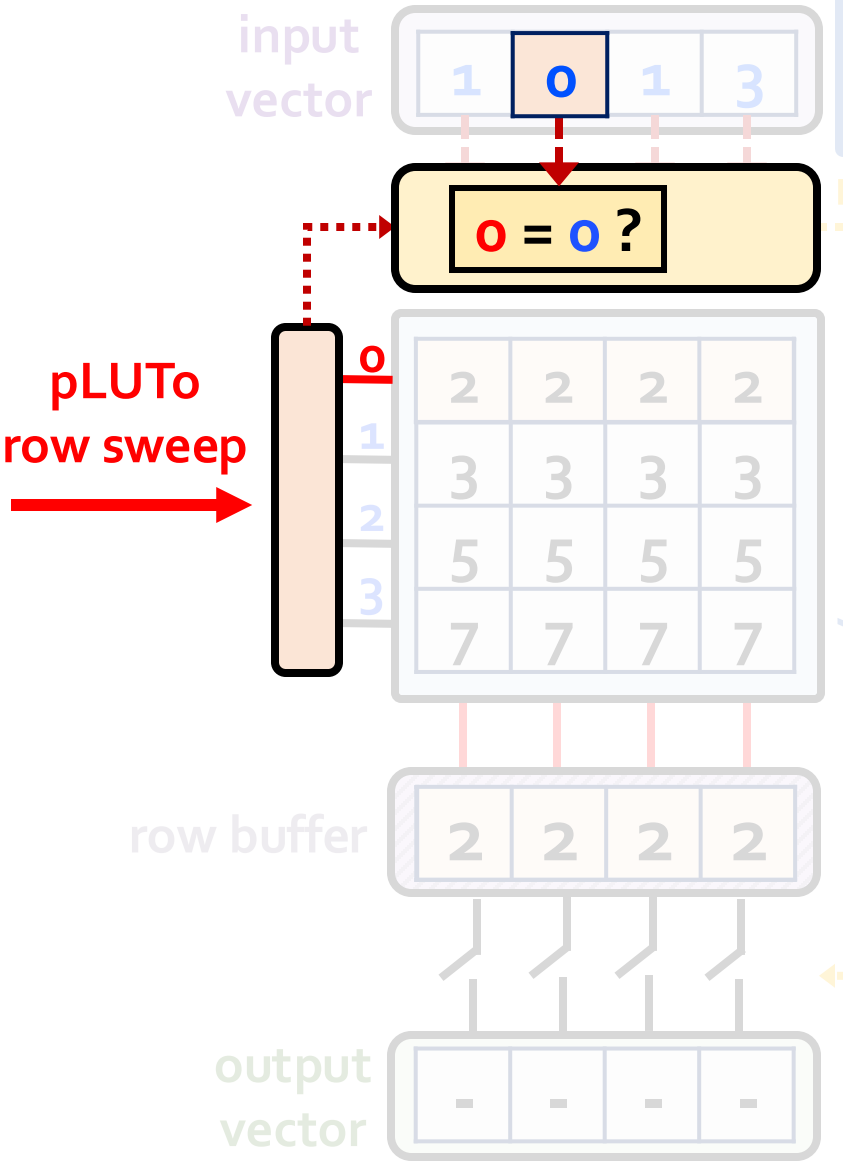
lookup table

1	0	1	3

input vector

3	2	3	7

output vector



In-DRAM pLUTo LUT Query: Step 1

LUT Query:
Return {2nd, 1st, 2nd, 4th}
prime numbers

LUT index	Prime numbers i	f(i)
0	1st	2
1	2nd	3
2	3rd	5
3	4th	7

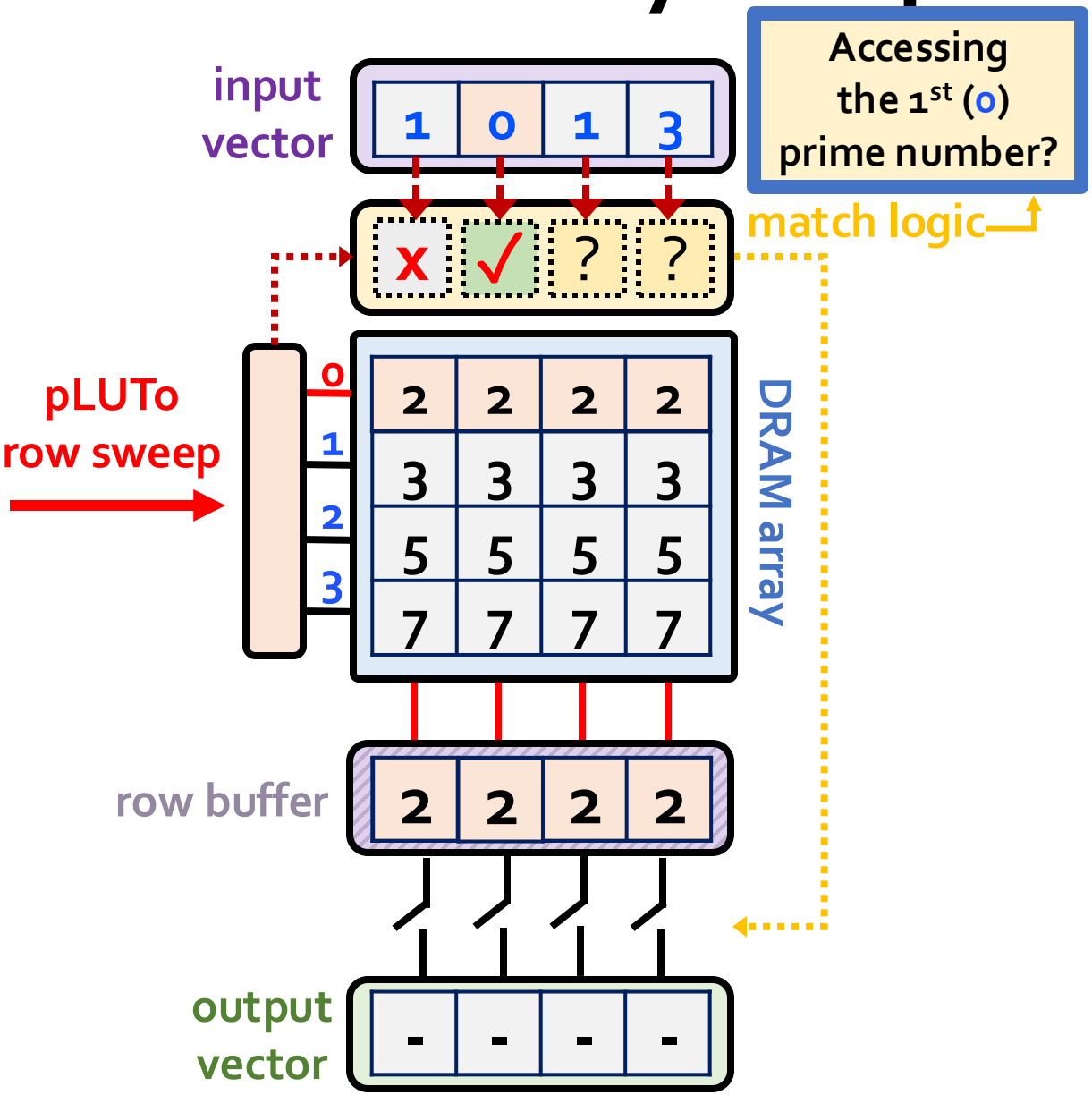
lookup table

1	0	1	3

input vector

3	2	3	7

output vector



In-DRAM pLUTo LUT Query: Step 1

LUT Query:
Return {2nd, 1st, 2nd, 4th}
prime numbers

LUT index	Prime numbers i	f(i)
0	1st	2
1	2nd	3
2	3rd	5
3	4th	7

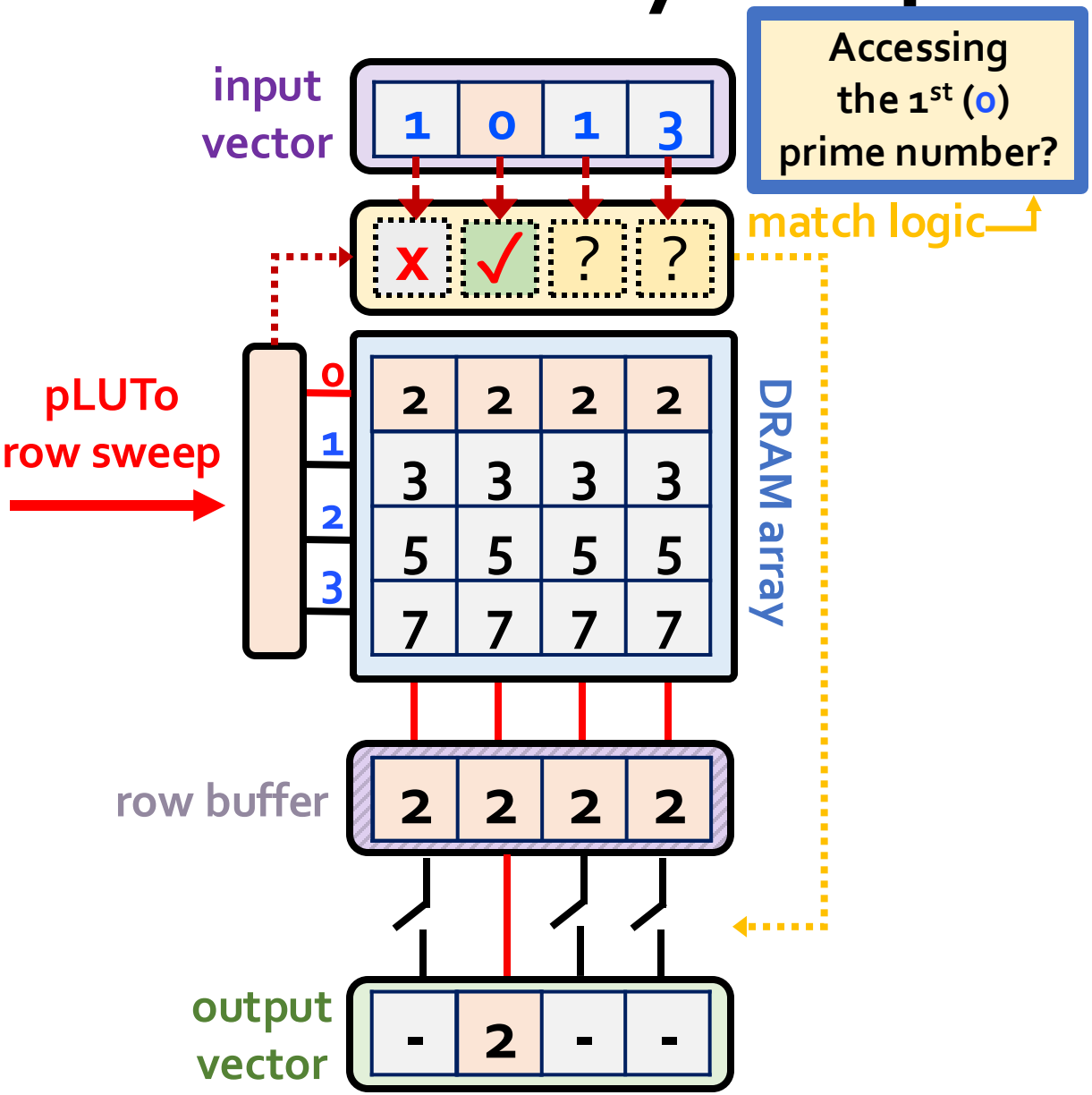
lookup table

1	0	1	3

input vector

3	2	3	7

output vector



In-DRAM pLUTo LUT Query: Step 1

LUT Query:
Return {2nd, 1st, 2nd, 4th}
prime numbers

LUT index	Prime numbers i	f(i)
0	1st	2
1	2nd	3
2	3rd	5
3	4th	7

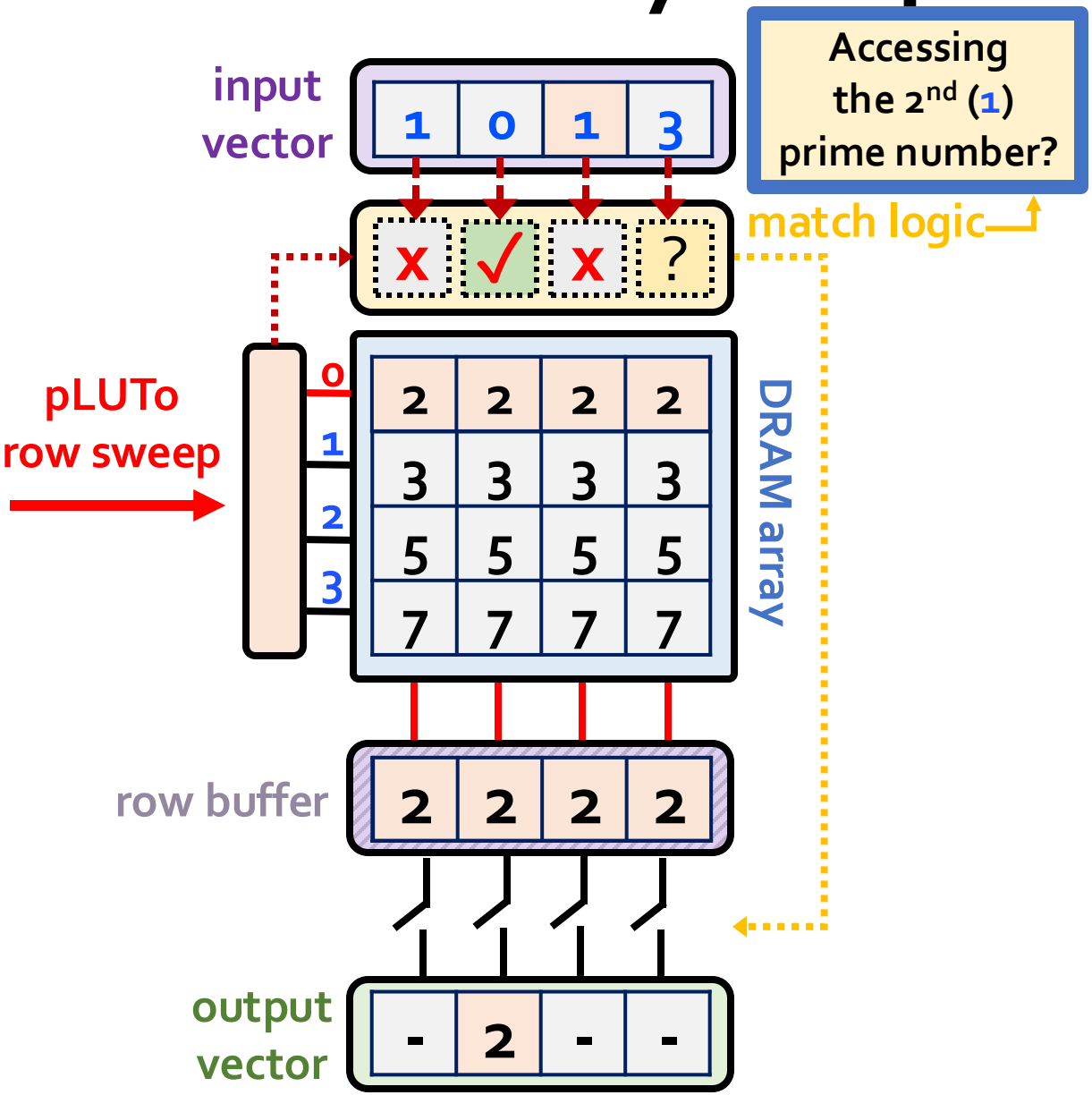
lookup table

1	0	1	3

input vector

3	2	3	7

output vector



In-DRAM pLUTo LUT Query: Step 1

LUT Query:
Return {2nd, 1st, 2nd, 4th}
prime numbers

LUT index	Prime numbers: i	Prime numbers: f(i)
0	1st	2
1	2nd	3
2	3rd	5
3	4th	7

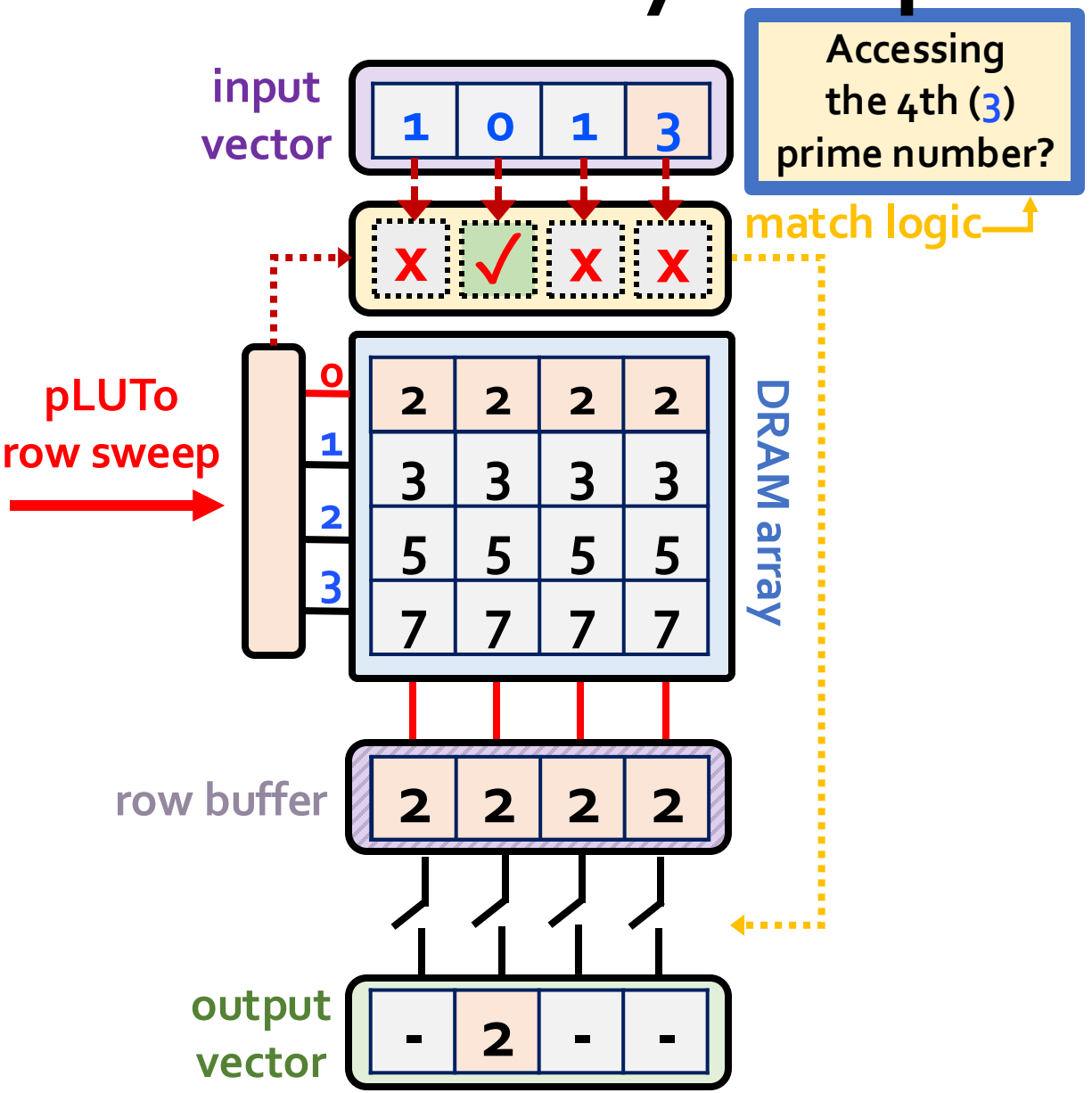
lookup table

1	0	1	3

input vector

3	2	3	7

output vector



In-DRAM pLUTo LUT Query: Step 4

LUT Query:
Return {2nd, 1st, 2nd, 4th}
prime numbers

LUT index	Prime numbers i	f(i)
0	1st	2
1	2nd	3
2	3rd	5
3	4th	7

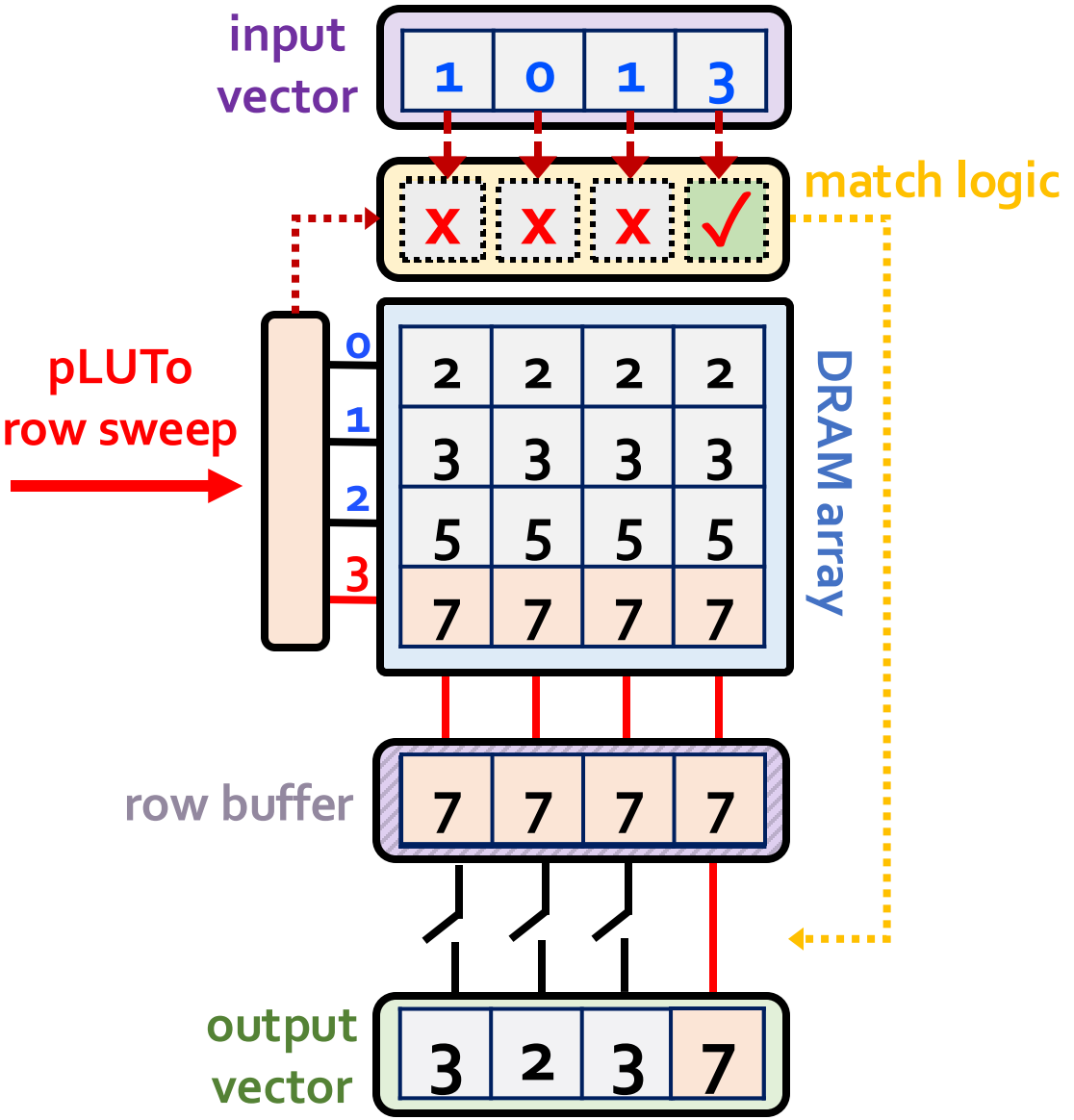
lookup table

1	0	1	3

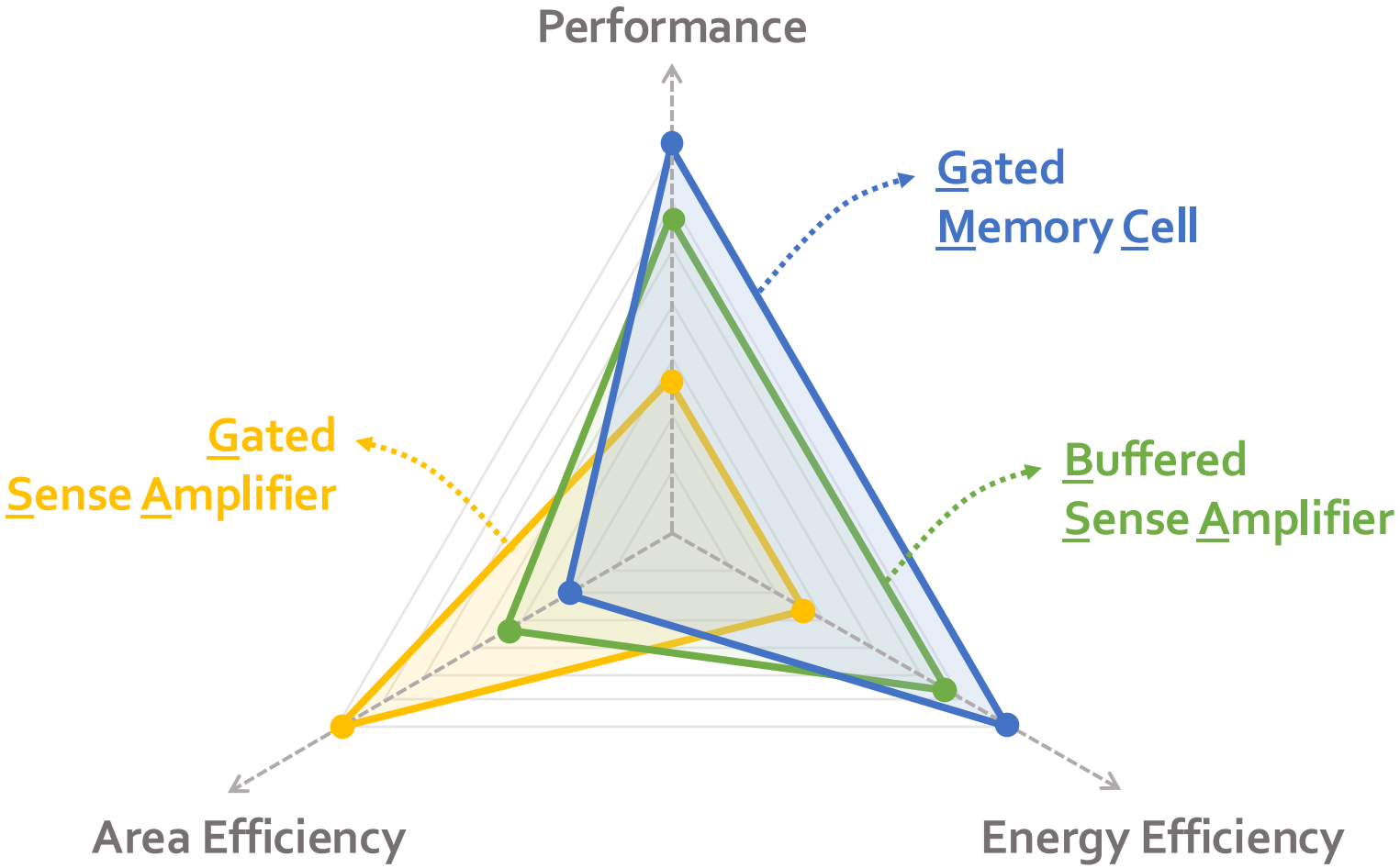
input vector

3	2	3	7

output vector



pLUTo Designs: Tradeoff Space



pLUTo designs cover a *broad design space* and provide *different* performance, energy, and area efficiency

Methodology: Experimental Setup

- **Baselines**
  - **CPU** (Intel® Xeon Gold 5118)
  - **GPU** (NVIDIA® GeForce RTX 3080 Ti)
  - **Processing-near-Memory (PnM)**
- **pLUTo**
  - In-house simulator, open-sourced
  - https://github.com/CMU-SAFARI/pLUTo

- **We evaluate:**
  - Performance
  - Energy consumption
  - Area overhead
  - Circuit-level reliability and correctness
  - …

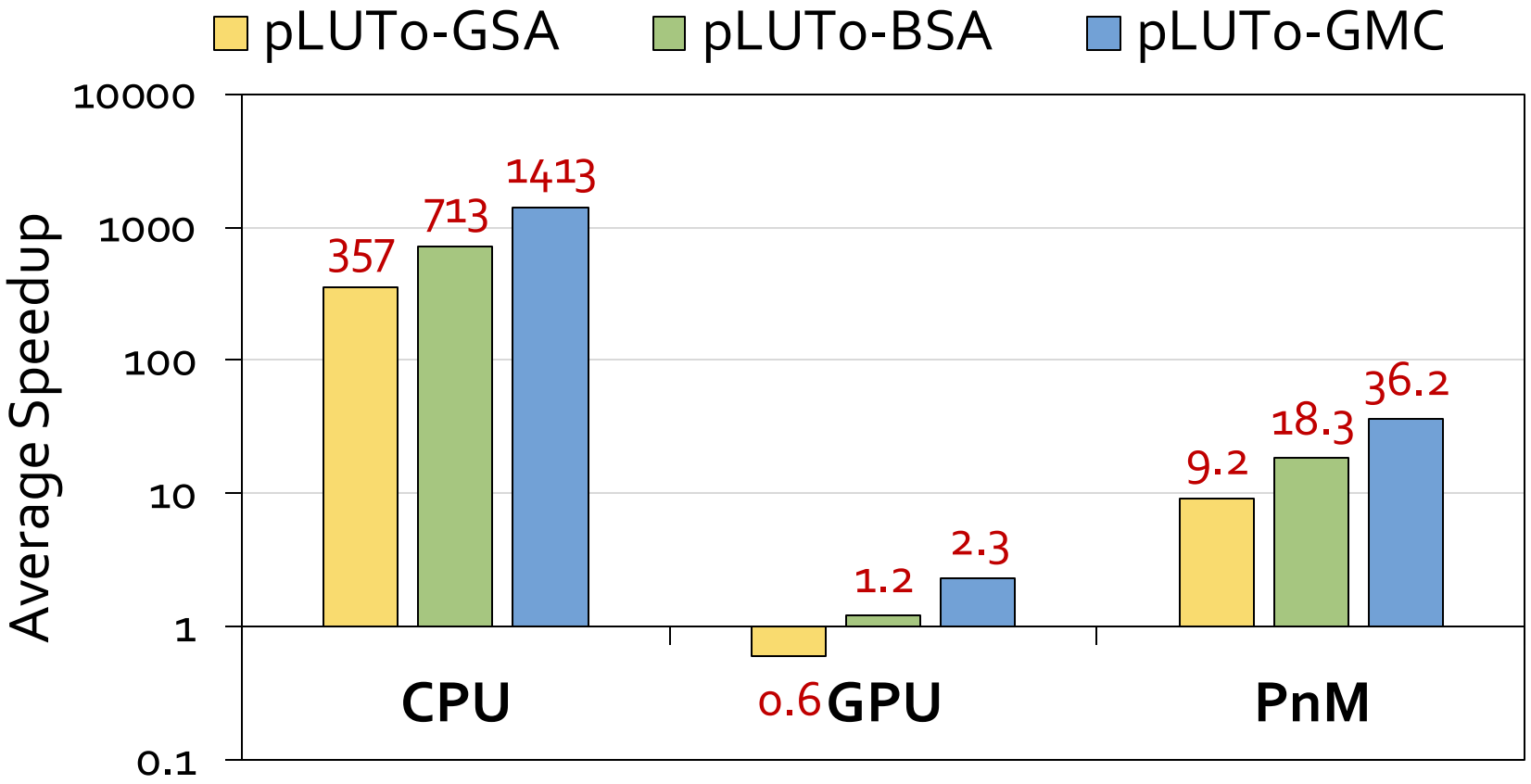
Methodology: Workloads

- **7 real-world workloads (not well supported by prior work)**
  - CRC-8/16/32
  - Salsa20
  - VMPC
  - Image Binarization
  - Color Grading

- **4 synthetic workloads (supported by prior work)**
  - Vector Addition
  - Vector Point-Wise Multiplication
  - Row-Level Bitwise Logic Operations
  - Bit Counting

Performance

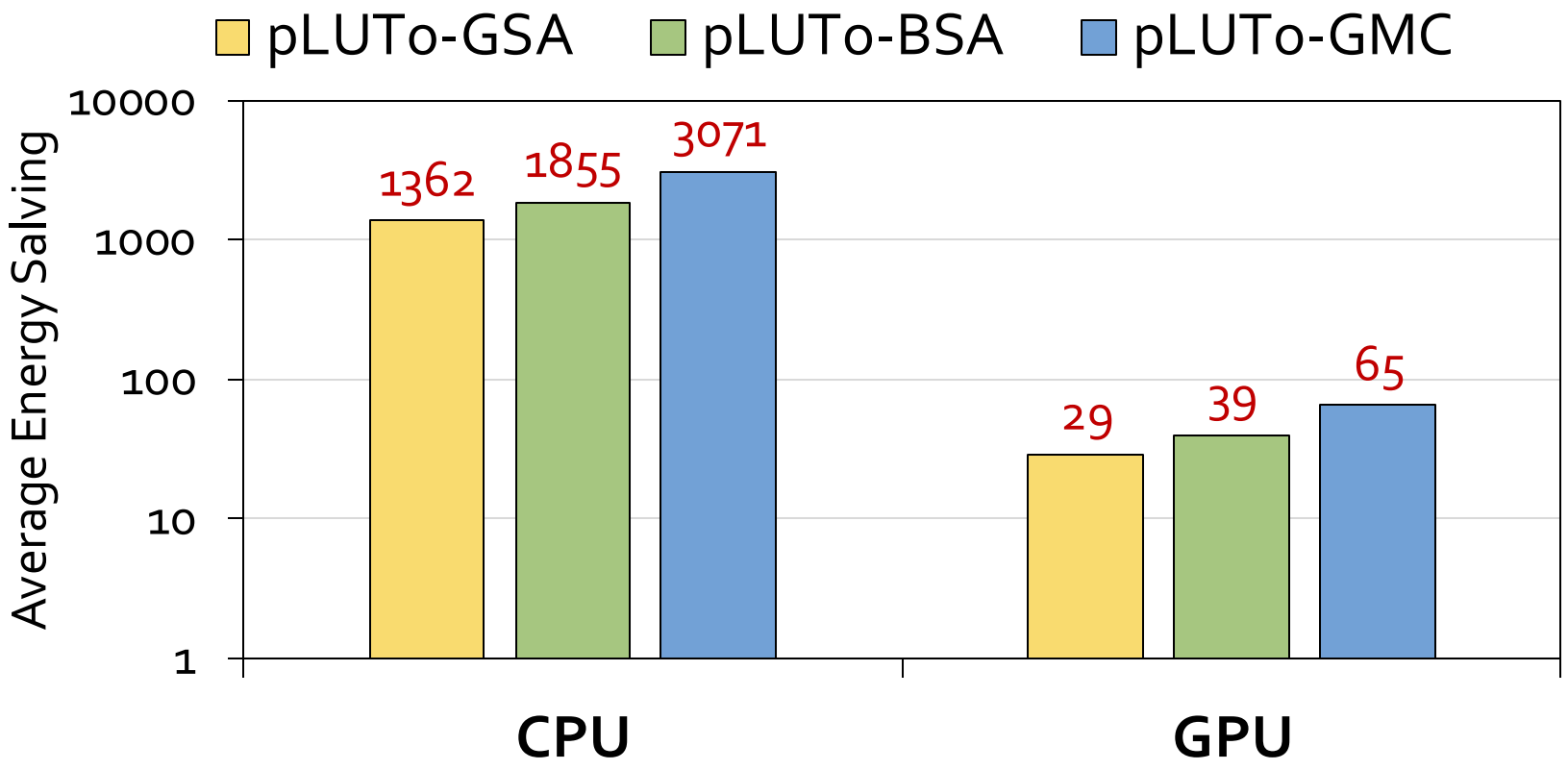
Average speedup across 7 real-world workloads



pLUTo *significantly outperforms* CPU, GPU and PnM baselines

Energy Consumption

Average energy consumption across 7 real-world workloads



pLUTo *significantly reduces energy consumption* compared to processor-centric architectures for various workloads

Processing-in-Memory:
In This Talk

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

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

Compilers & Systems for PUM: MIMDRAM

- **Geraldo F. Oliveira**, Ataberk Olgun, Abdullah Giray Yaglikci, F. Nisa Bostanci, Juan Gomez-Luna, Saugata Ghose, and Onur Mutlu, "MIMDRAM: An End-to-End Processing-Using-DRAM System for High-Throughput, Energy-Efficient and Programmer-Transparent Multiple-Instruction Multiple-Data Processing"
  *Proceedings of the 30th International Symposium on High-Performance Computer Architecture (**HPCA**)*, April 2024.

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

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

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

Limitations of PUD Systems:
Overview

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

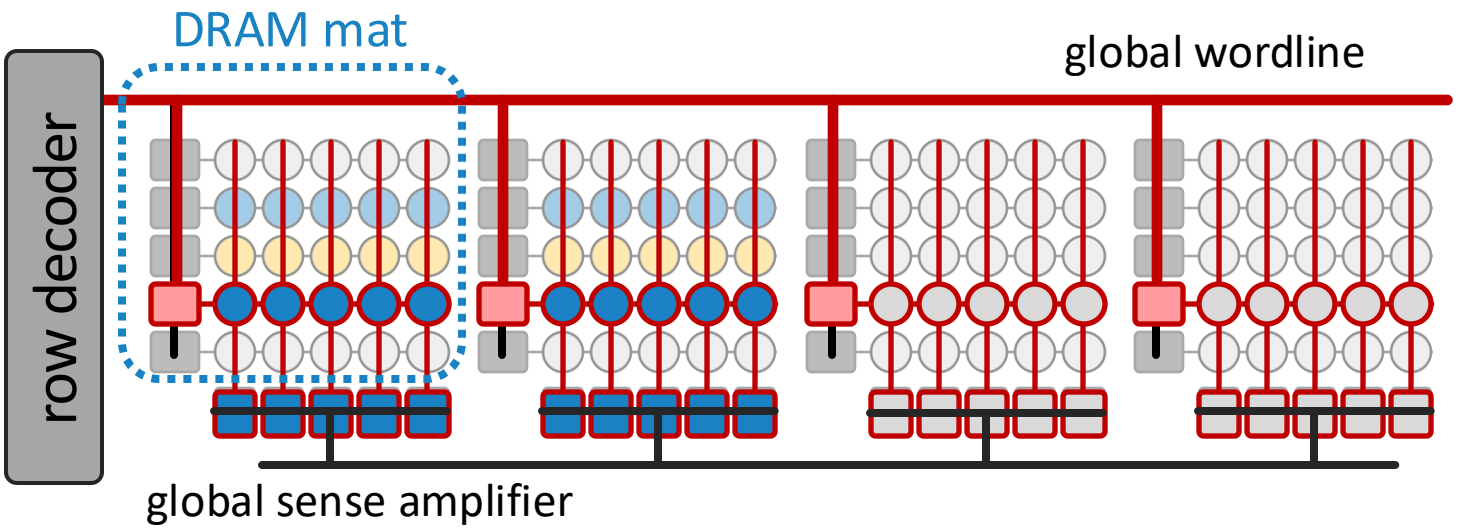
1 SIMD Underutilization
- due to data parallelism variation within and across applications
- leads to throughput and energy waste

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

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

MIMDRAM:
Key Idea (I)

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



Key Issue:

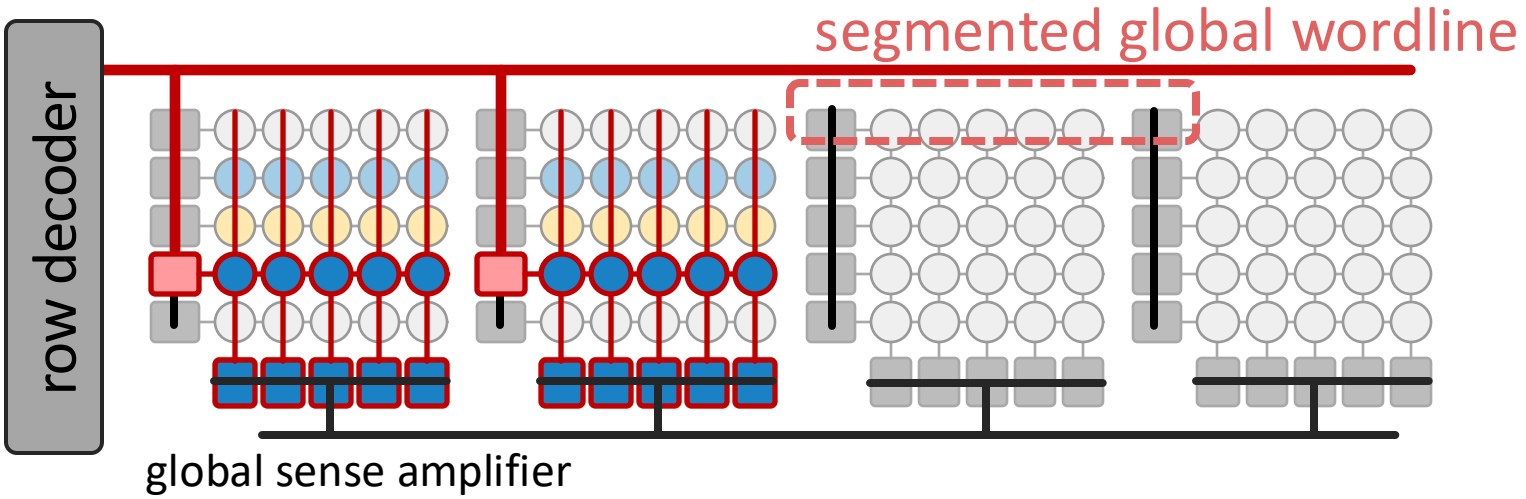
on a DRAM access, the global wordline propagates across all DRAM mats

Fine-Grained DRAM:

segments the global wordline to access individual DRAM mats

MIMDRAM:
Key Idea (II)

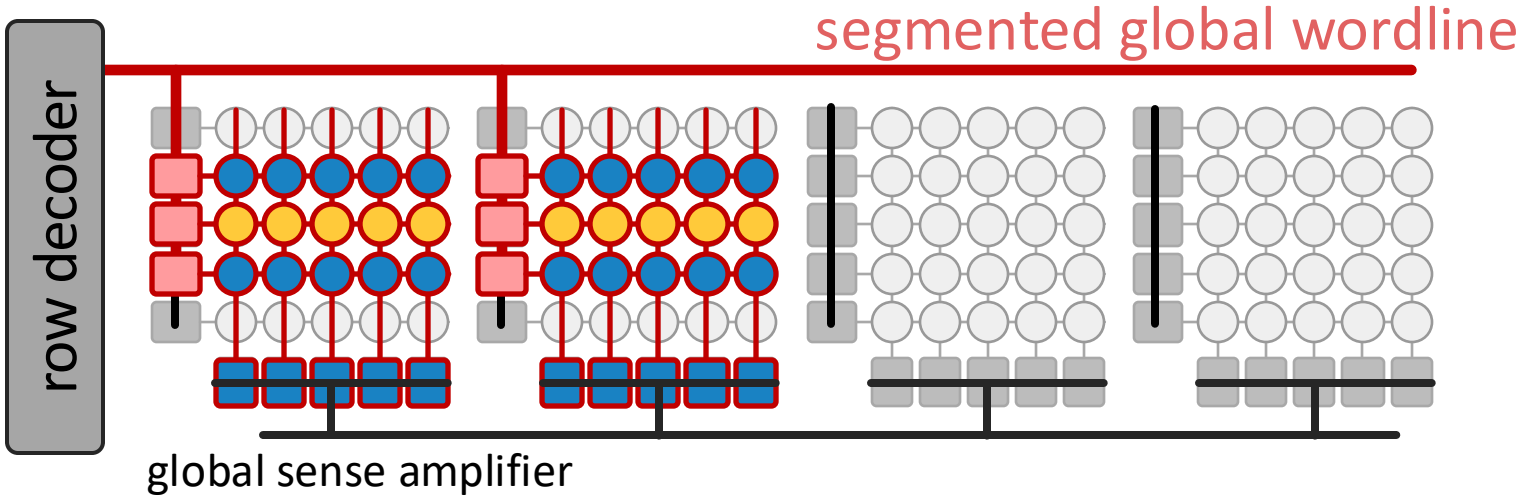
Fine-Grained DRAM:
segments the global wordline to access individual DRAM mats



Fine-grained DRAM for energy-efficient DRAM access:

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

MIMDRAM:
Key Idea (III)

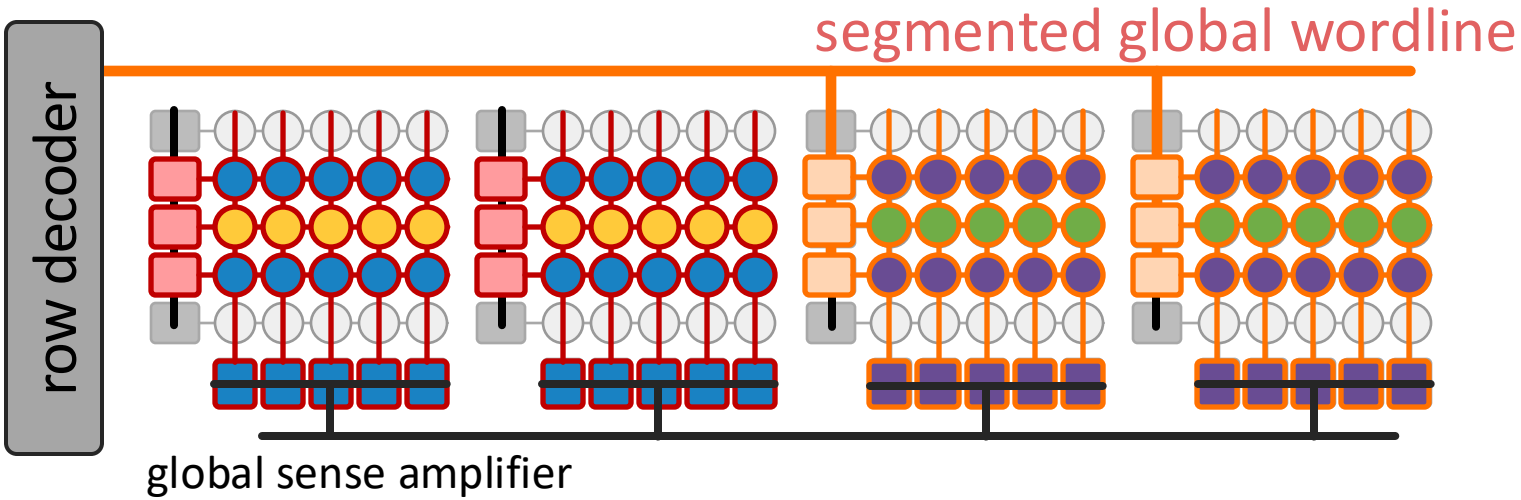


Fine-grained DRAM for processing-using-DRAM:

1 Improves SIMD utilization

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

MIMDRAM:
Key Idea (III)

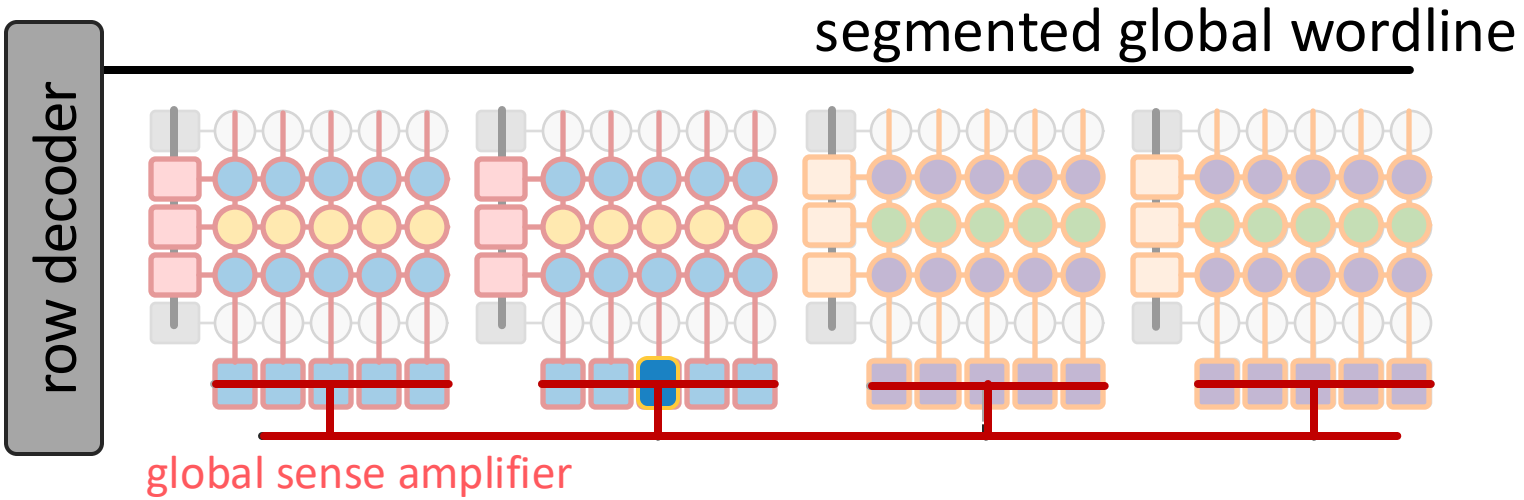


Fine-grained DRAM for processing-using-DRAM:

1 Improves SIMD utilization

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

MIMDRAM:
Key Idea (III)



Fine-grained DRAM for processing-using-DRAM:

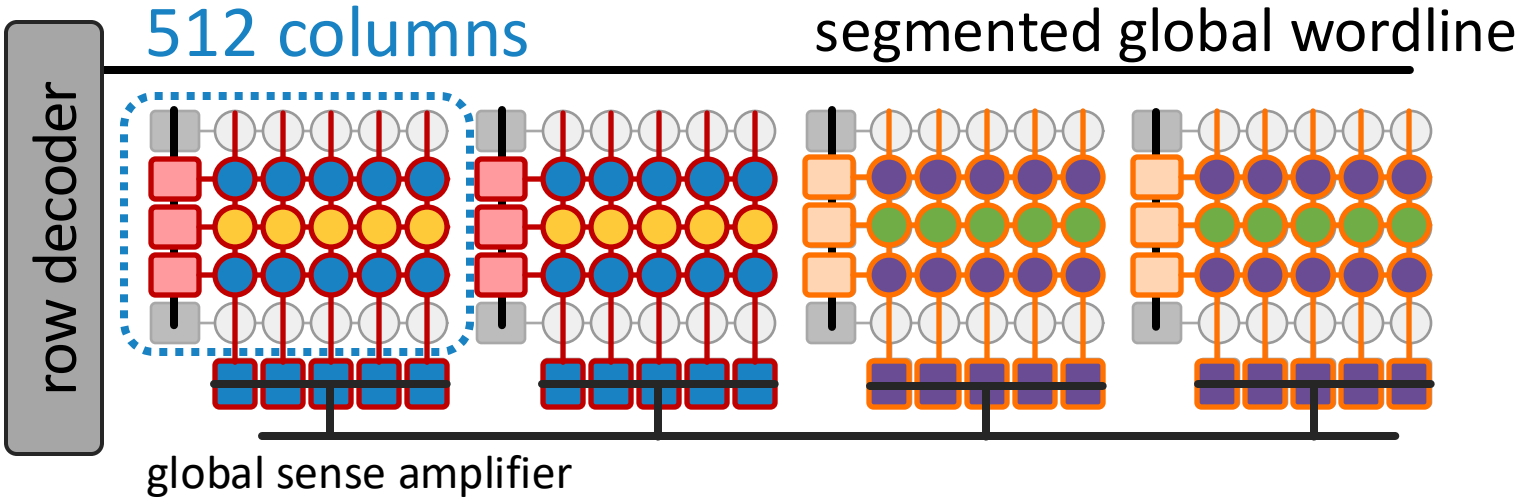
1 Improves SIMD utilization

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

2 Enables low-cost interconnects for vector reduction

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

MIMDRAM:
Key Idea (III)



Fine-grained DRAM for processing-using-DRAM:

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

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

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

MIMDRAM:
Overview

MIMDRAM is a hardware/software co-designed PUD system that enables fine-grained PUD computation at low cost and programming effort

Main components of MIMDRAM:

1 Hardware

- DRAM array modification to enable fine-grained PUD computation
- inter- and intra-mat interconnects to enable PUD vector reduction
- control unit design to orchestrate PUD execution

2 Software

- compiler support to transparently generate PUD instructions
- system support to map and execute PUD instructions

MIMDRAM:
Overview

MIMDRAM is a hardware/software co-designed PUD system that enables fine-grained PUD computation at low cost and programming effort

Main components of MIMDRAM:

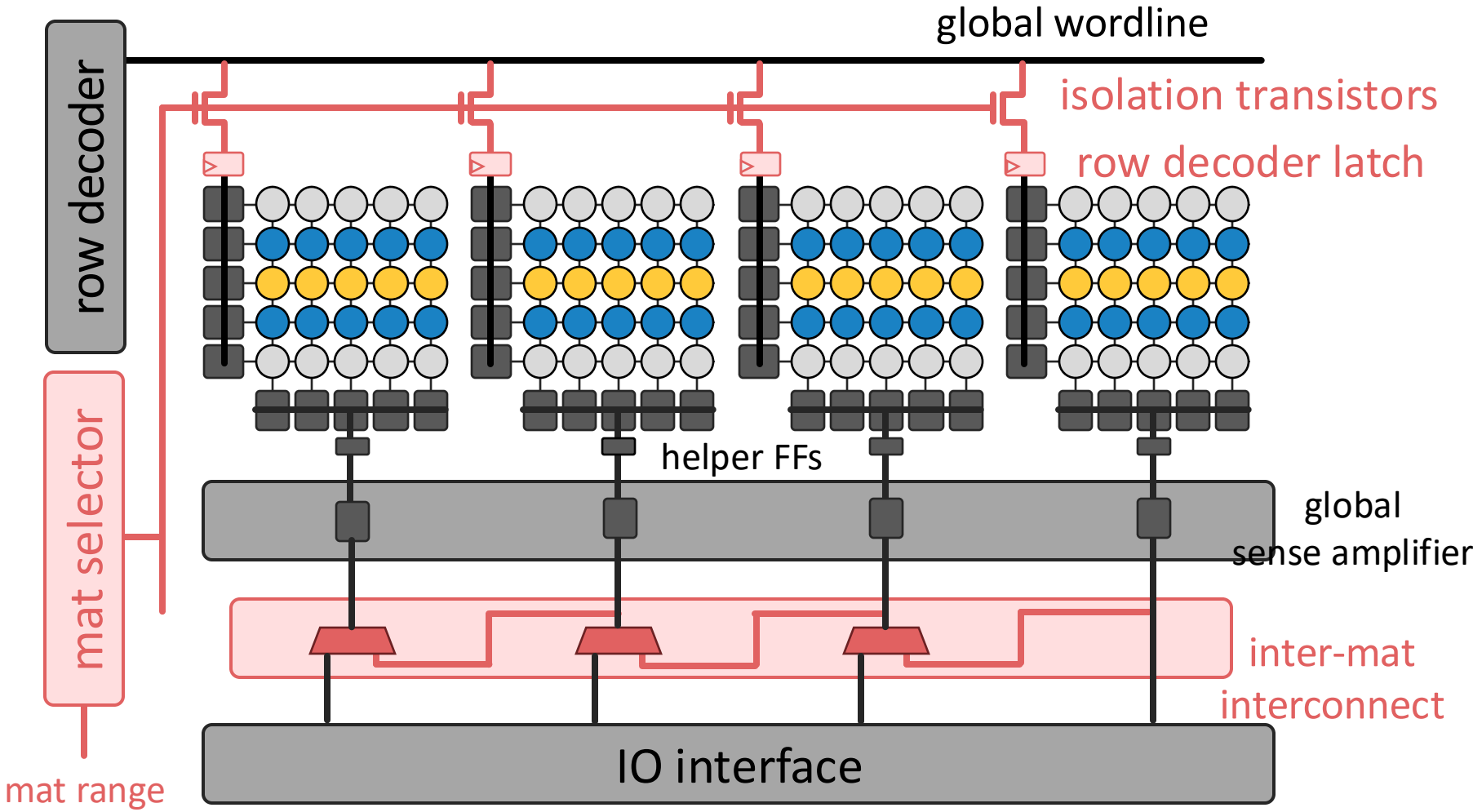
1 Hardware

- **DRAM array modification to enable fine-grained PUD computation**
- **inter- and intra-mat interconnects to enable PUD vector reduction**
- **control unit design to orchestrate PUD execution**

2 Software

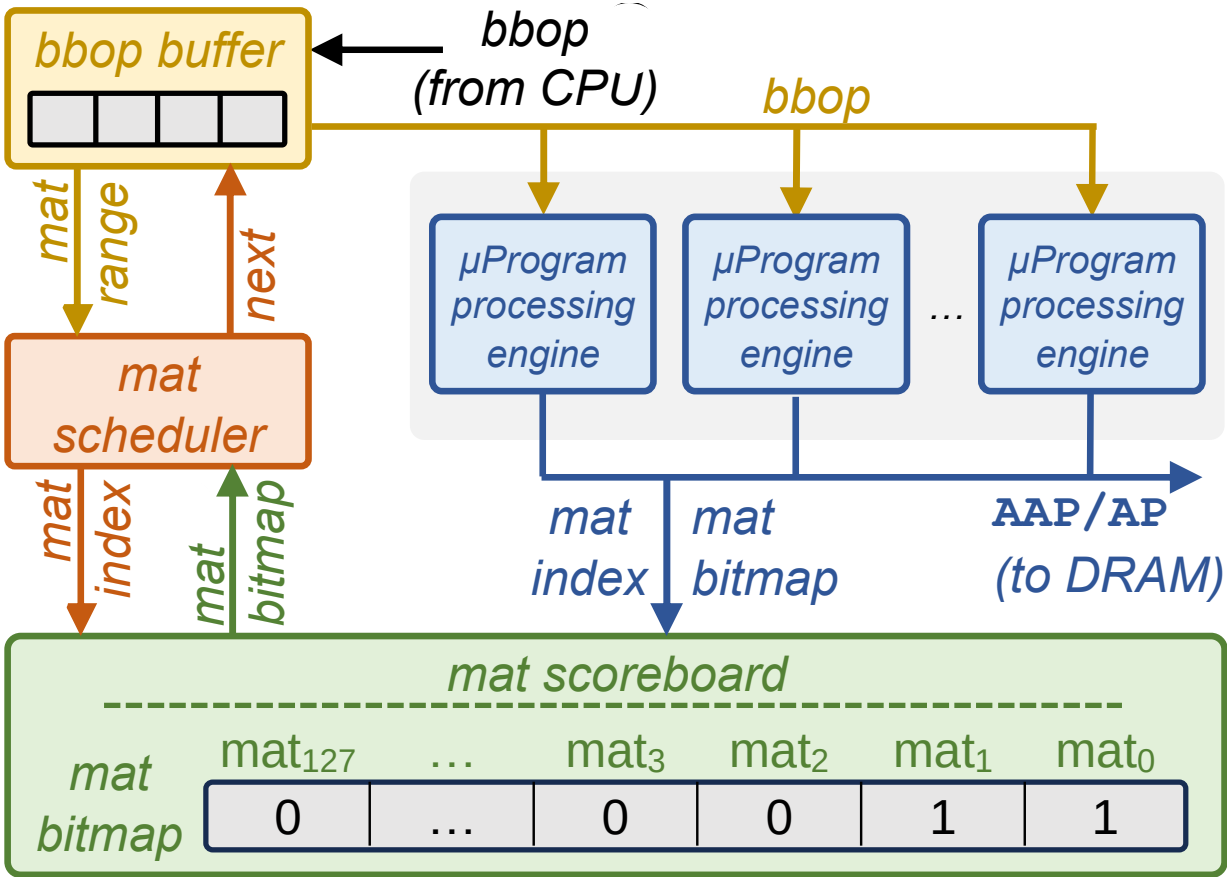
- new compiler support to transparently generate PUD instructions
- system support to map and execute PUD instructions

MIMDRAM:
Modifications to DRAM Chip



MIMDRAM:
Control Unit Design

The control unit schedules and orchestrates the execution of multiple PUD operations transparently



MIMDRAM:
More in the Paper

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

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

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

https://arxiv.org/pdf/2402.19080.pdf

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

MIMDRAM:
Overview

MIMDRAM is a hardware/software co-designed PUD system that enables fine-grained PUD computation at low cost and programming effort

Main components of MIMDRAM:

1 Hardware

- DRAM array modification to enable fine-grained PUD computation
- inter- and intra-mat interconnects to enable PUD vector reduction
- control unit design to orchestrate PUD execution

2 Software

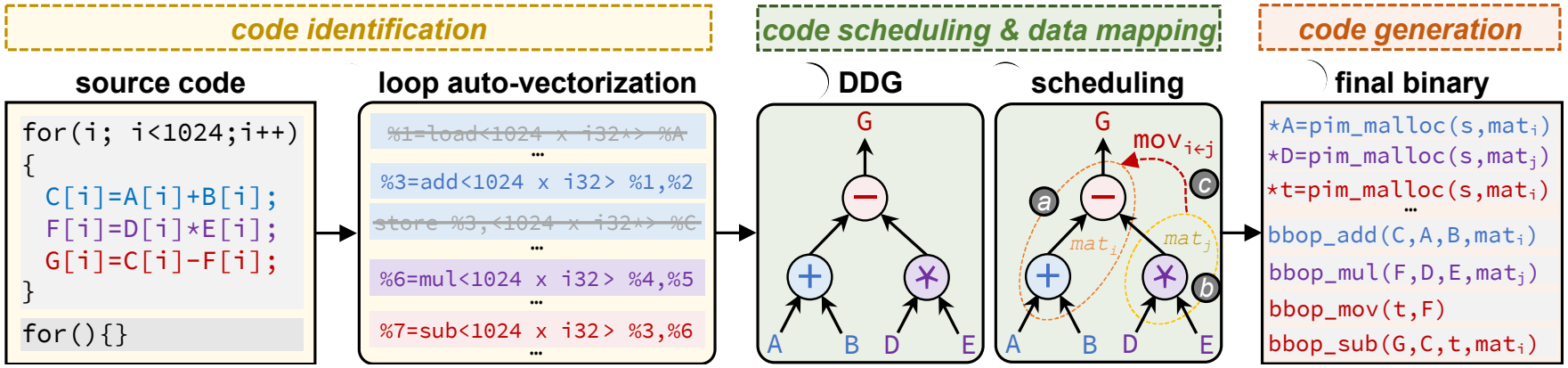
- **new compiler support to transparently generate PUD instructions**
- **system support to map and execute PUD instructions**

MIMDRAM:
Compiler Support

Goal

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

Three new LLVM-based passes targeting PUD execution

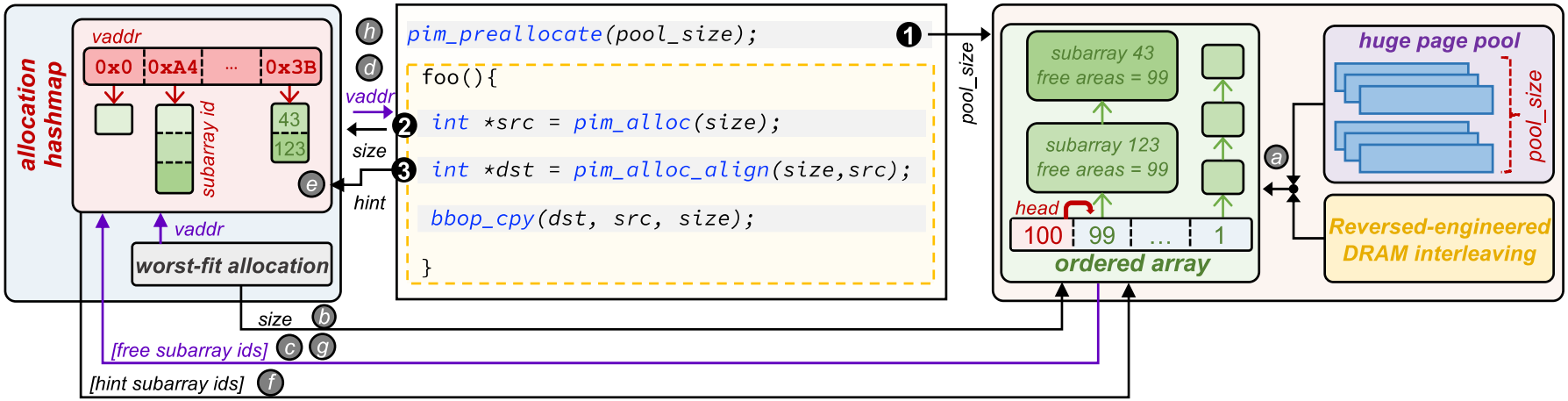


MIMDRAM:
System Support

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

MIMDRAM:
Data Allocation & Alignment

MIMDRAM's memory allocator uses a pool of huge pages and reversed-engineered DRAM interleaving information for PUD memory objects



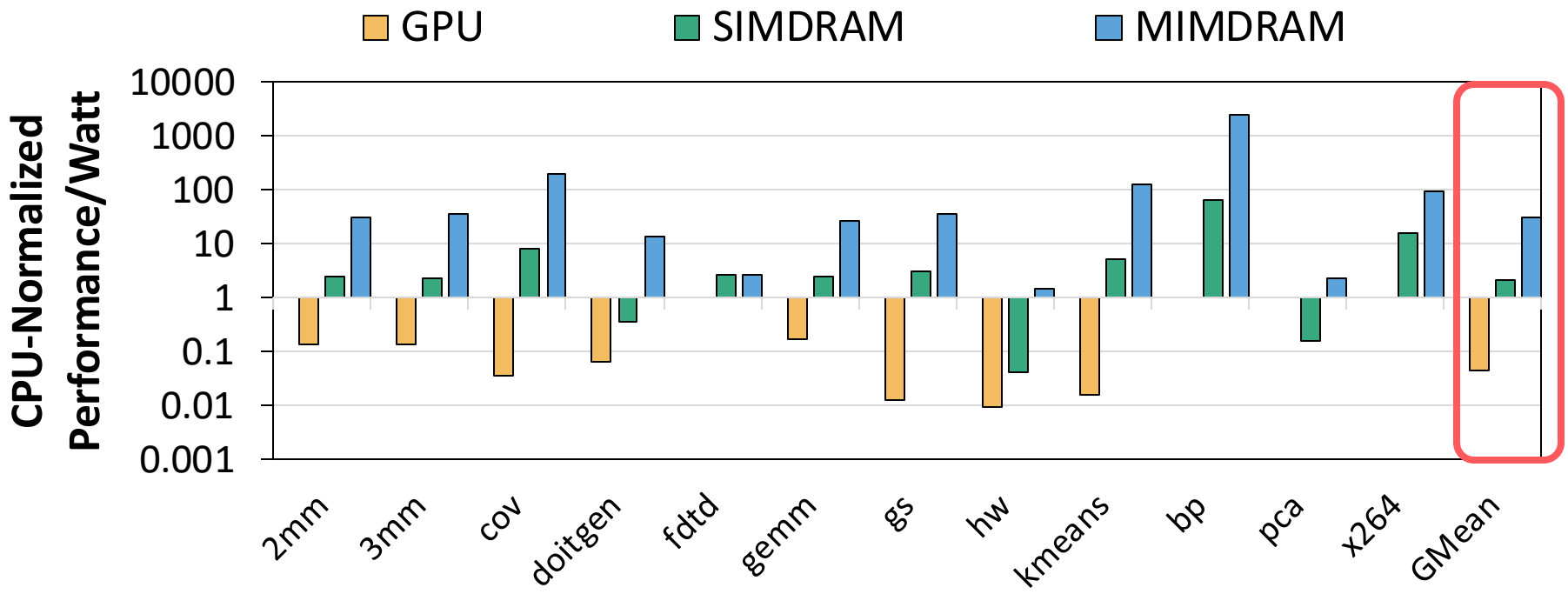
Evaluation:
Methodology Overview

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

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

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

Evaluation:
Single Application Analysis – Energy Efficiency



Takeaway

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

Evaluation:
More in the Paper

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

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

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

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

Evaluation:
More in the Paper

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

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

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

https://arxiv.org/pdf/2402.19080.pdf

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

Processing-in-Memory: Challenges

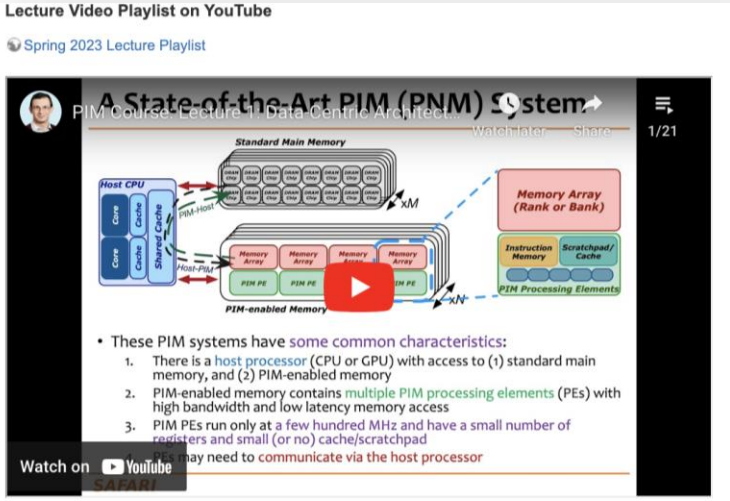
To fully support PIM systems, we need to develop:

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

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

Processing-in-Memory Course:
Fall 2024

- **Fall 2024 Edition**
  - https://safari.ethz.ch/projects_and_seminars/fall2024/doku.php?id=processing_in_memory
- **Spring 2023 Edition**
  - https://safari.ethz.ch/projects_and_seminars/spring2023/doku.php?id=processing_in_memory
- **YouTube Livestream**
  - https://www.youtube.com/playlist?list=PL5Q2soXY2Zi9DSQg7OAlsNO8dOQKxF9sl
- **Bachelor's Course:**
  - Elective at ETH Zurich
  - Introduction to PIM systems (from industry and academia)
  - Tutorial on using real PIM infrastructure
  - Potential research exploration

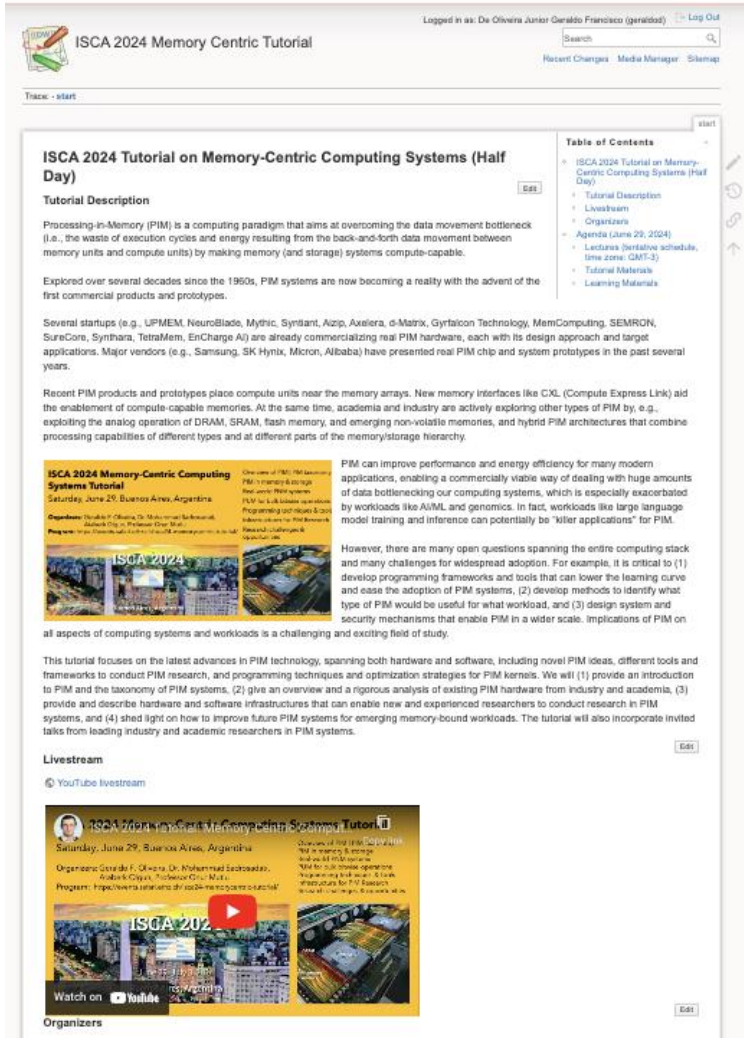


Spring 2023 Meetings/Schedule

Week	Date	Livestream	Meeting	Learning Materials	Assignments
W1	09.03 Thu.	YouTube Livestream	M1: P&S PIM Course Presentation (PDF) (PPT)	Required Materials Recommended Materials	HW 0 Out
W2	16.03 Thu.	YouTube Premiere	M2: How to Evaluate Data Movement Bottlenecks (PDF) (PPT)		
			Hands-on Project Proposals		
W3	23.03 Thu.	YouTube Premiere	M3: Real-world PIM: UPMEM PIM (PDF) (PPT)		
W4	30.03 Thu.	YouTube Premiere	M4: Real-world PIM: Microbenchmarking of UPMEM PIM (PDF) (PPT)		
W5	06.04 Thu.	YouTube Premiere	M5: Real-world PIM: Samsung HBM-PIM (PDF) (PPT)		
W6	13.04 Thu.	YouTube Premiere	M6: Real-world PIM: SK Hynix AiM (PDF) (PPT)		

PIM Tutorial:
ISCA & MICRO 2024

- ISCA 2024 Edition
  - June 29th: Lectures, hands-on labs, and invited lectures
  - https://events.safari.ethz.ch/isca24-memorycentric-tutorial/doku.php?id=start
- MICRO 2024 Edition (Upcoming)
  - November 2nd: Lectures, hands-on labs, and invited lectures
  - https://events.safari.ethz.ch/micro24-memorycentric-tutorial/doku.php?id=start
- YouTube Livestream
  - https://www.youtube.com/watch?v=KV2MXvcBgb0



Next PIM Tutorial:
PPoPP 2025

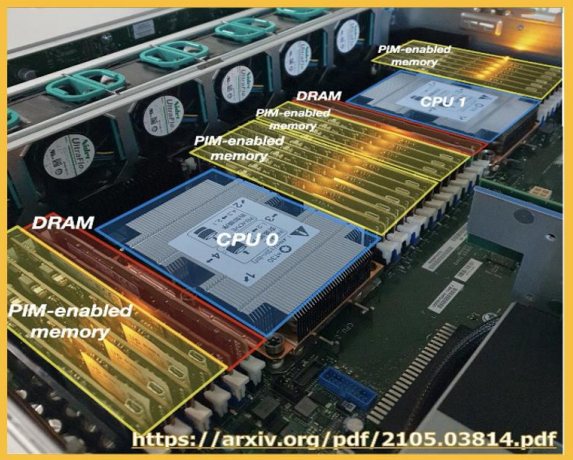
PPoPP 2025 - Tutorial on Memory-Centric Computing Systems

March 1st – March 5th, Las Vegas, Nevada, USA

Organizers: Geraldo F. Oliveira, Dr. Mohammad Sadrosadati, Ataberk Olgun, Professor Onur Mutlu

Program: https://events.safari.ethz.ch/ppopp25-memorycentric-tutorial/

- Overview of PIM | PIM taxonomy
- PIM in memory & storage
- Real-world PNM systems
- PUM for bulk bitwise operations
- Programming techniques & tools
- Infrastructures for PIM Research
- Research challenges & opportunities



https://events.safari.ethz.ch/ppopp25-memorycentric-tutorial/doku.php

1st Memory-Centric Computing Systems Workshop @ ASPLOS 2025

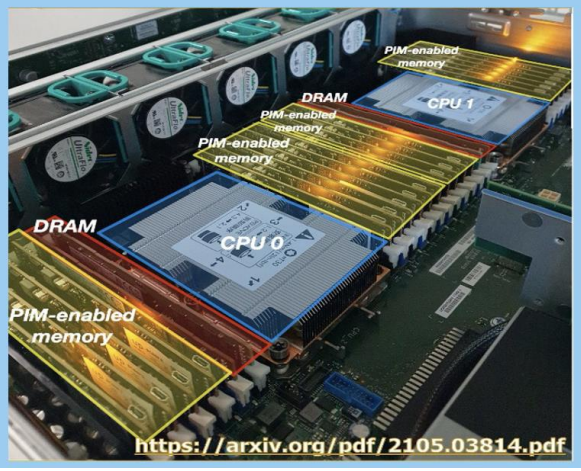
ASPLOS 2025 – 1st Workshop on Memory-Centric Computing Systems

Sunday, March 30th, Rotterdam, The Netherlands

Organizers: Geraldo F. Oliveira, Dr. Mohammad Sadrosadati, Ataberk Olgun, Professor Onur Mutlu

Program: https://events.safari.ethz.ch/asplos25-MCCSys/doku.php

- Overview of PIM | PIM taxonomy
- PIM in memory & storage
- Real-world PNM systems
- PUM for bulk bitwise operations
- Programming techniques & tools
- Infrastructures for PIM Research
- Research challenges & opportunities



https://events.safari.ethz.ch/asplos25-MCCSys/doku.php

Referenced Papers, Talks, Artifacts

- All are available at

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

https://www.youtube.com/onurmutlulectures

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

Open Source Tools: SAFARI GitHub

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

126

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

https://arxiv.org/pdf/1903.03988.pdf

Enabling the Adoption of Data-Centric Systems: Hardware/Software Support for Processing-Using-Memory Architectures

Memory & Storage Summit 2024 – 07 December 2024

Geraldo F. Oliveira

geraldofojunior@gmail.com

https://geraldofojunior.github.io/

SAFARI

ETH zürich