1st Workshop on Memory-Centric Computing: Processing-Using-Memory

Geraldo F. Oliveira

https://geraldofojunior.github.io

ASPLOS 2025

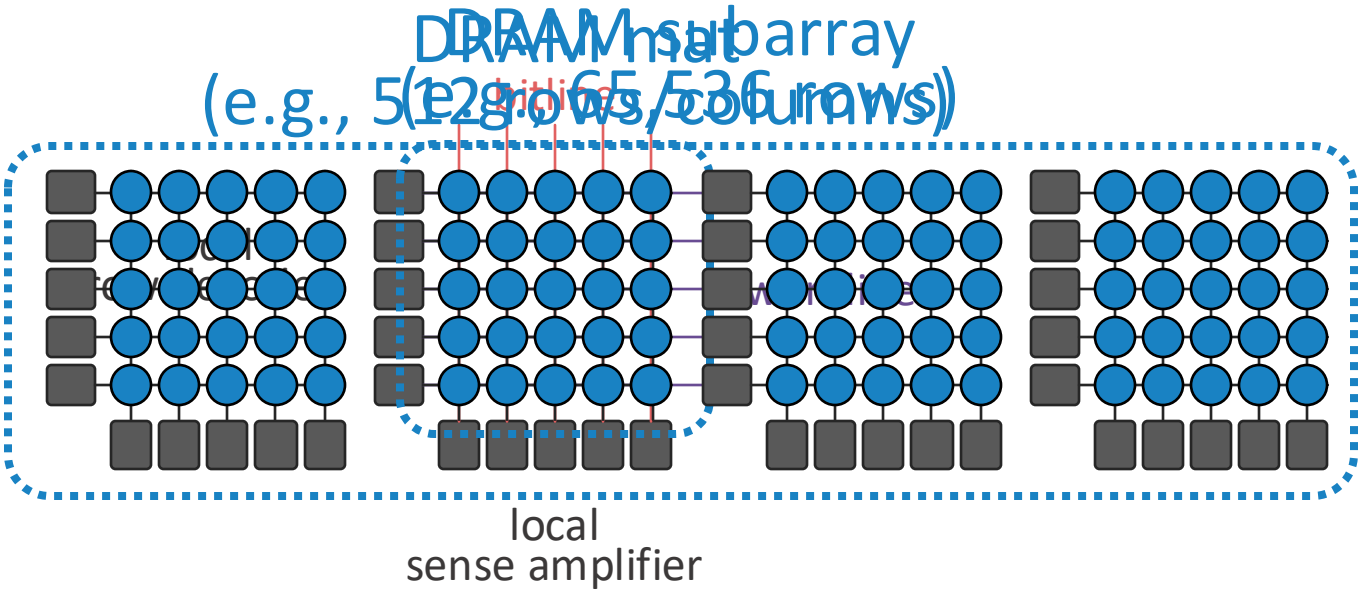
30 March 2025

SAFARI

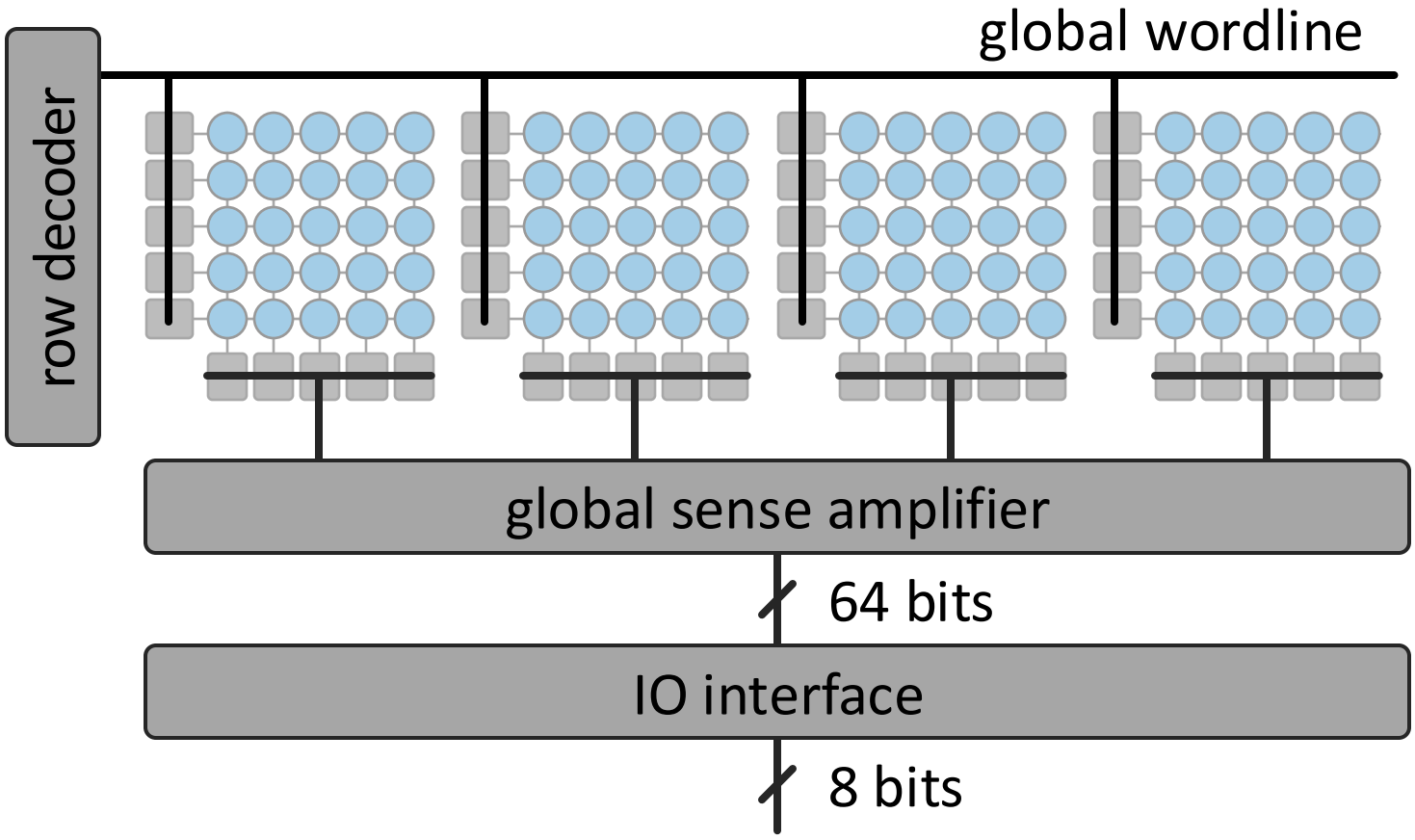
ETH zürich

Brief Review: Inside A DRAM Chip

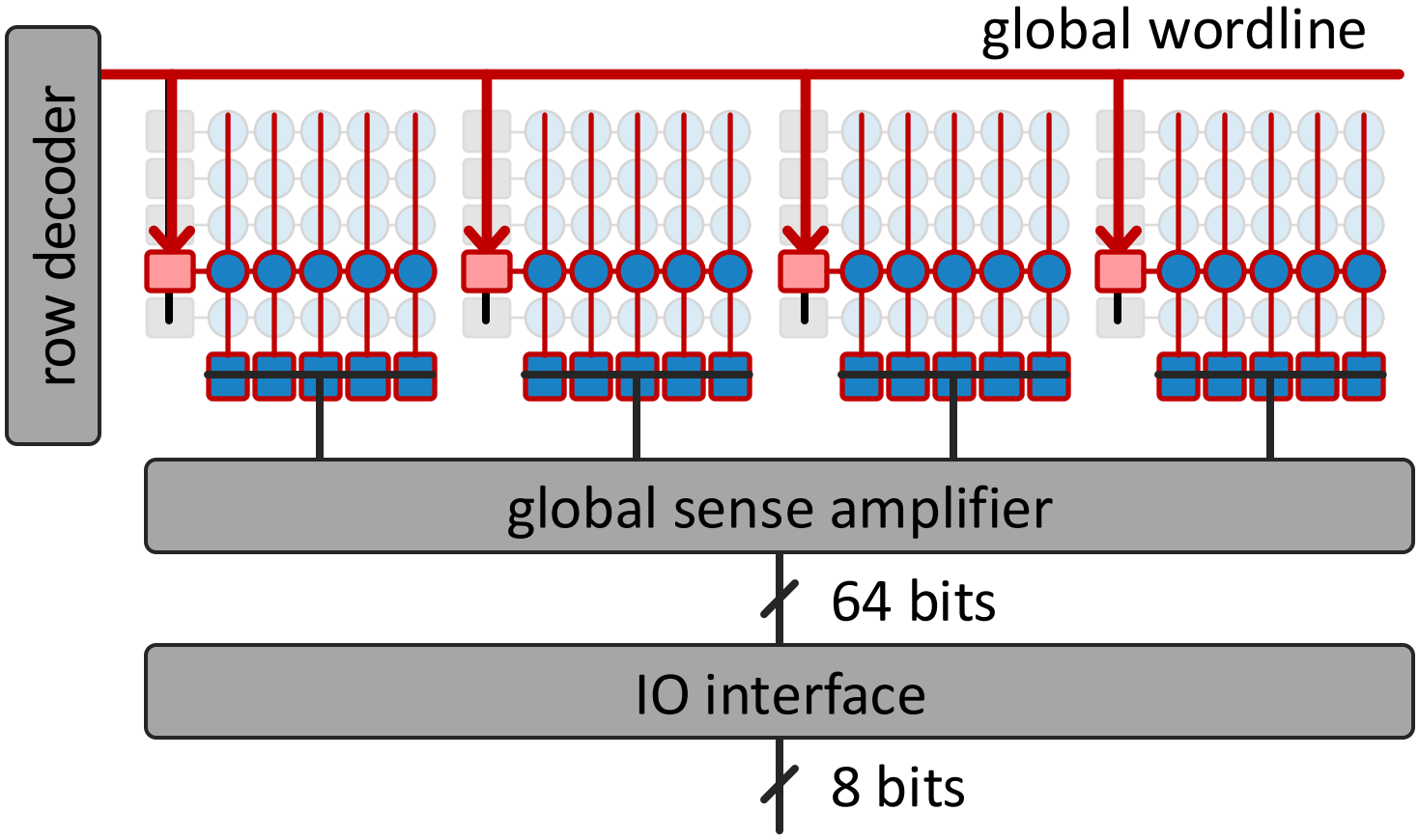
Background:
DRAM Hierarchical Organization



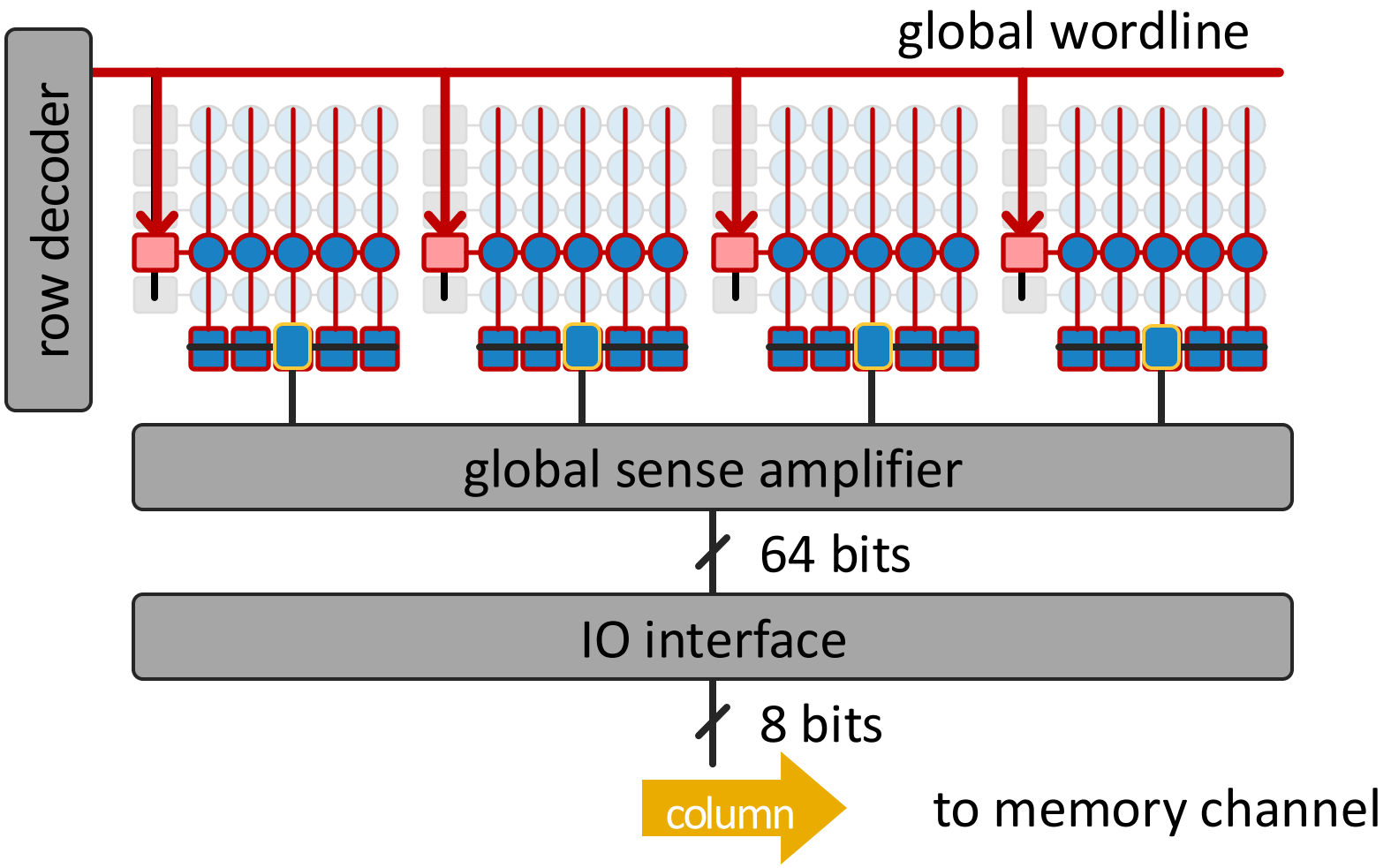
Background:
DRAM Hierarchical Organization



Background:
DRAM Operation – Row Access (ACTIVATE)

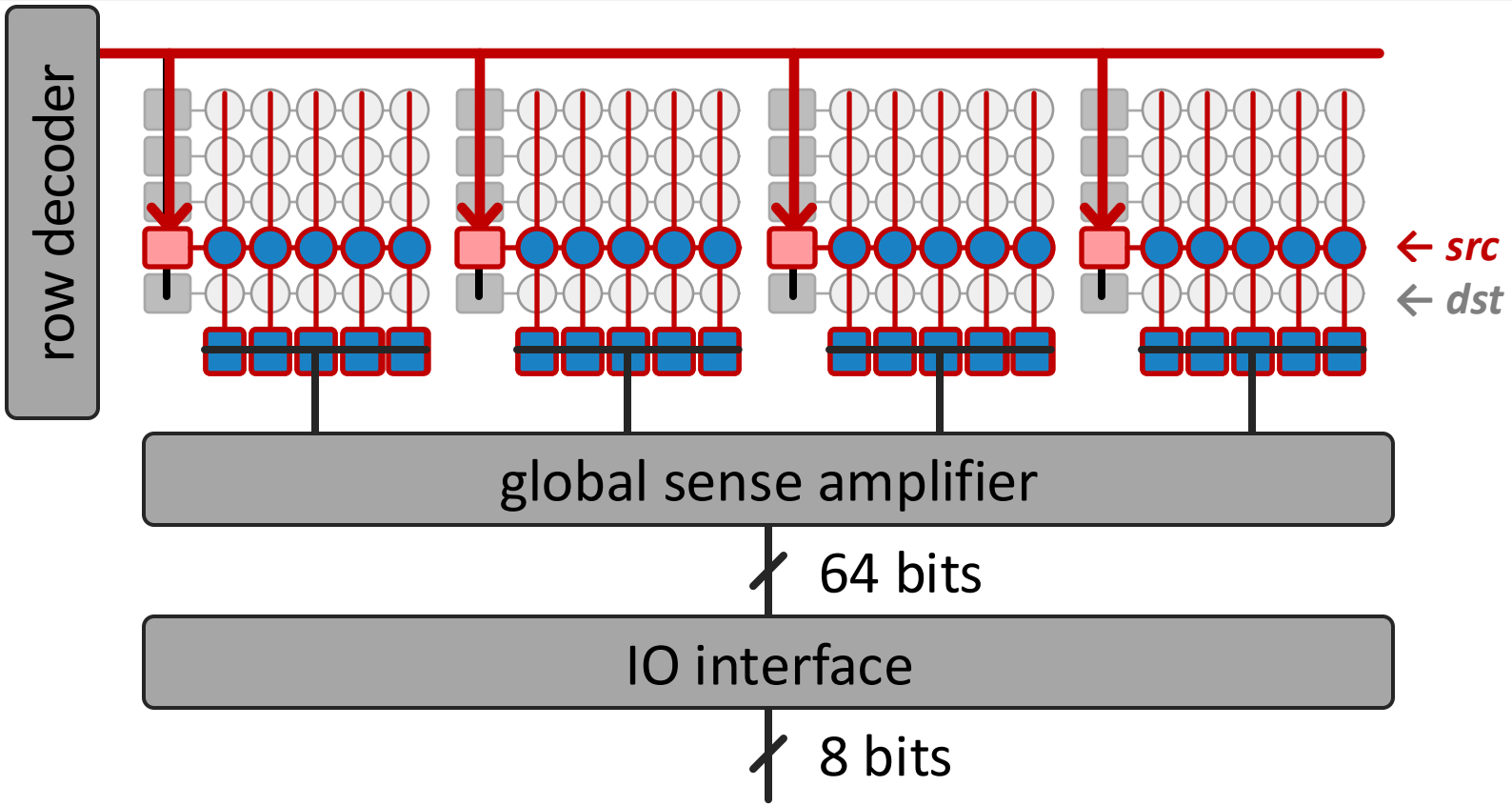


Background:
DRAM Operation – Column Access (READ)



Background:
In-DRAM Row Copy

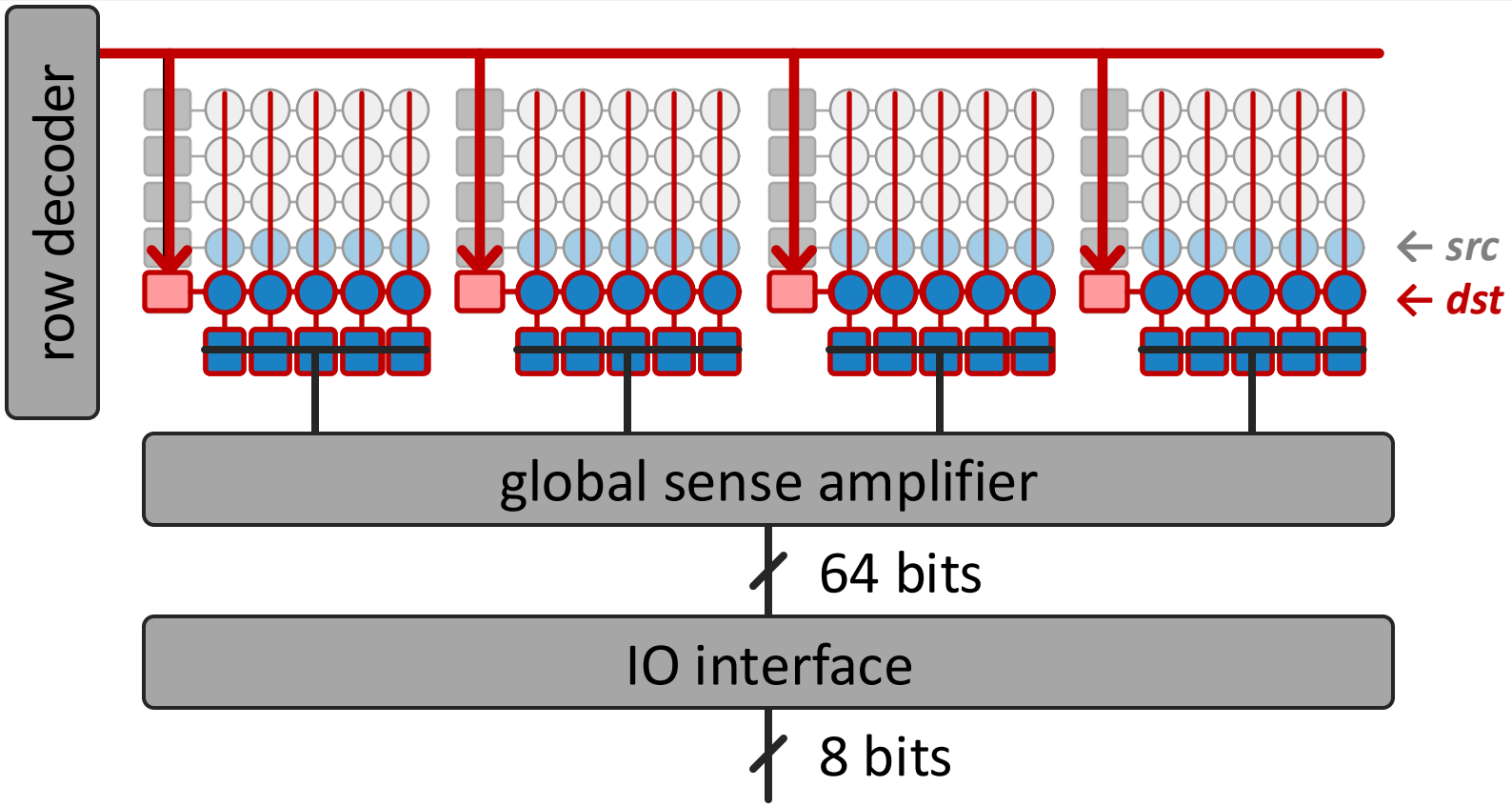
In-DRAM row copy is performed by
issuing back-to-back ACTIVATES to the DRAM



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

Background:
In-DRAM Row Copy

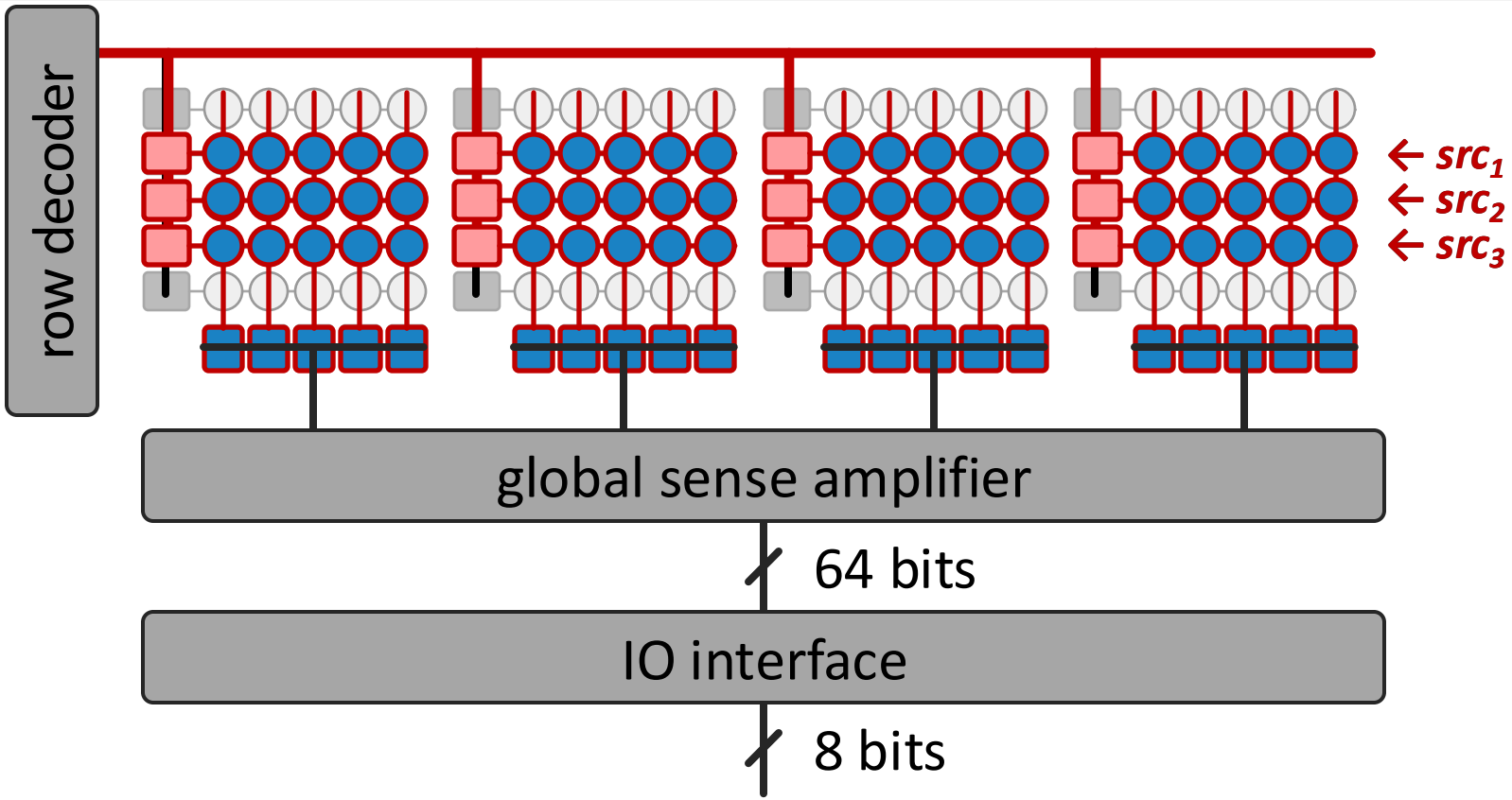
In-DRAM row copy is performed by
issuing back-to-back ACTIVATES to the DRAM



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

Background:
In-DRAM 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:
PUD in Commodity Off-the-Shelf 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

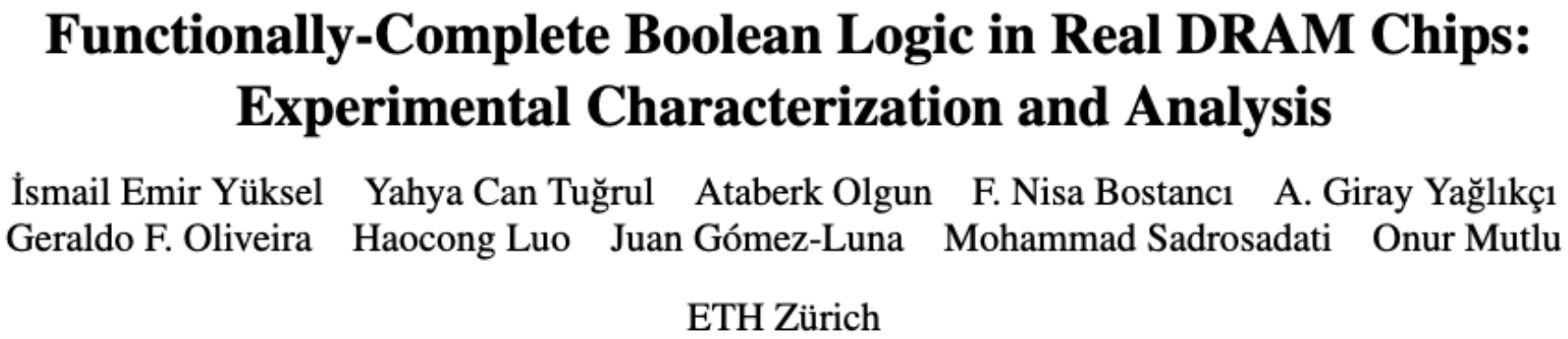
PUD in COTS DRAM:
Summary of Results

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

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.



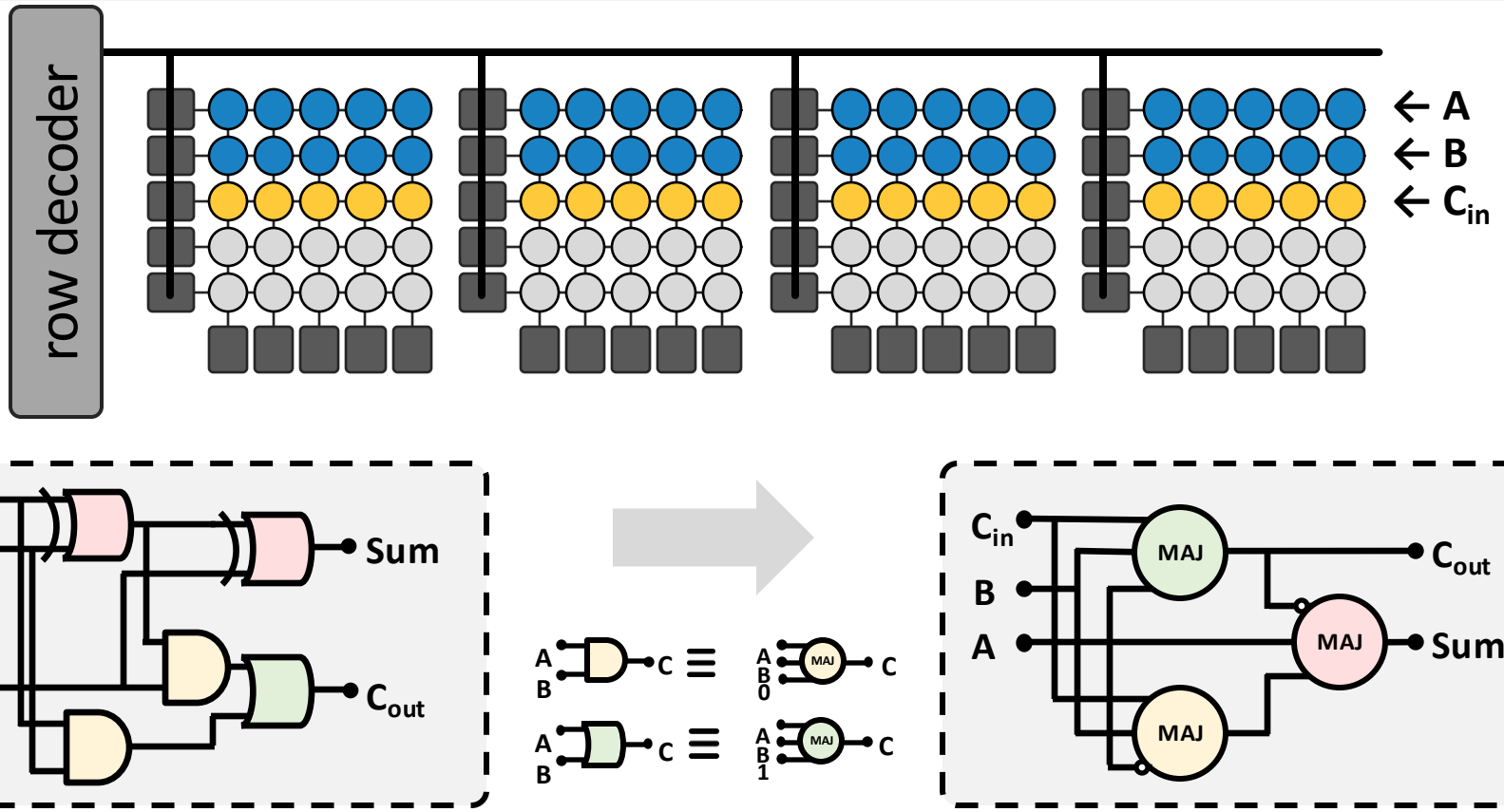
Functionally-Complete Boolean Logic in Real DRAM Chips: Experimental Characterization and Analysis

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

Background:
Bulk Bitwise Arithmetic Operations

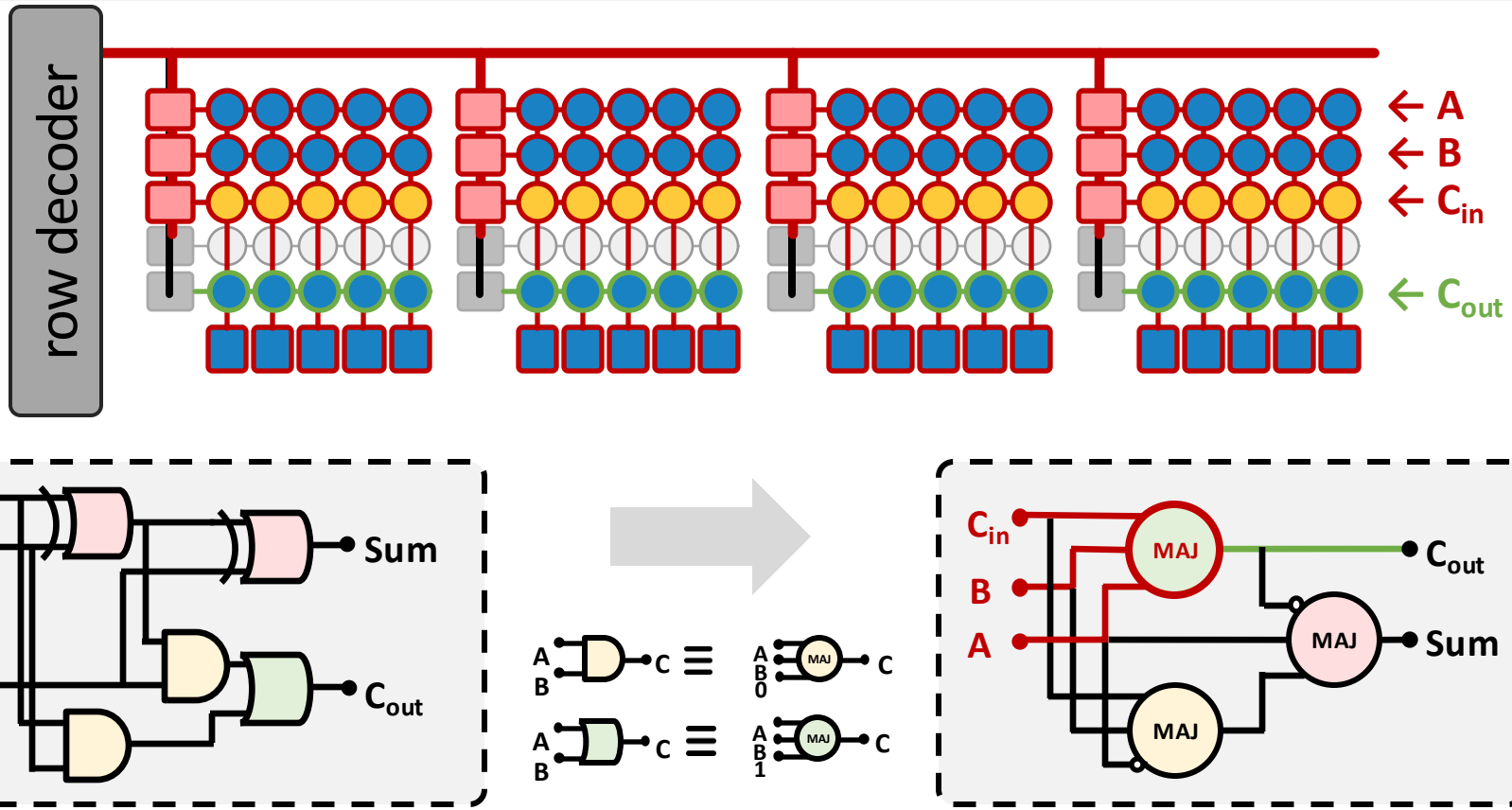
Bulk bitwise arithmetic can be performed by
orchestrating in-DRAM row copy and majority operations



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

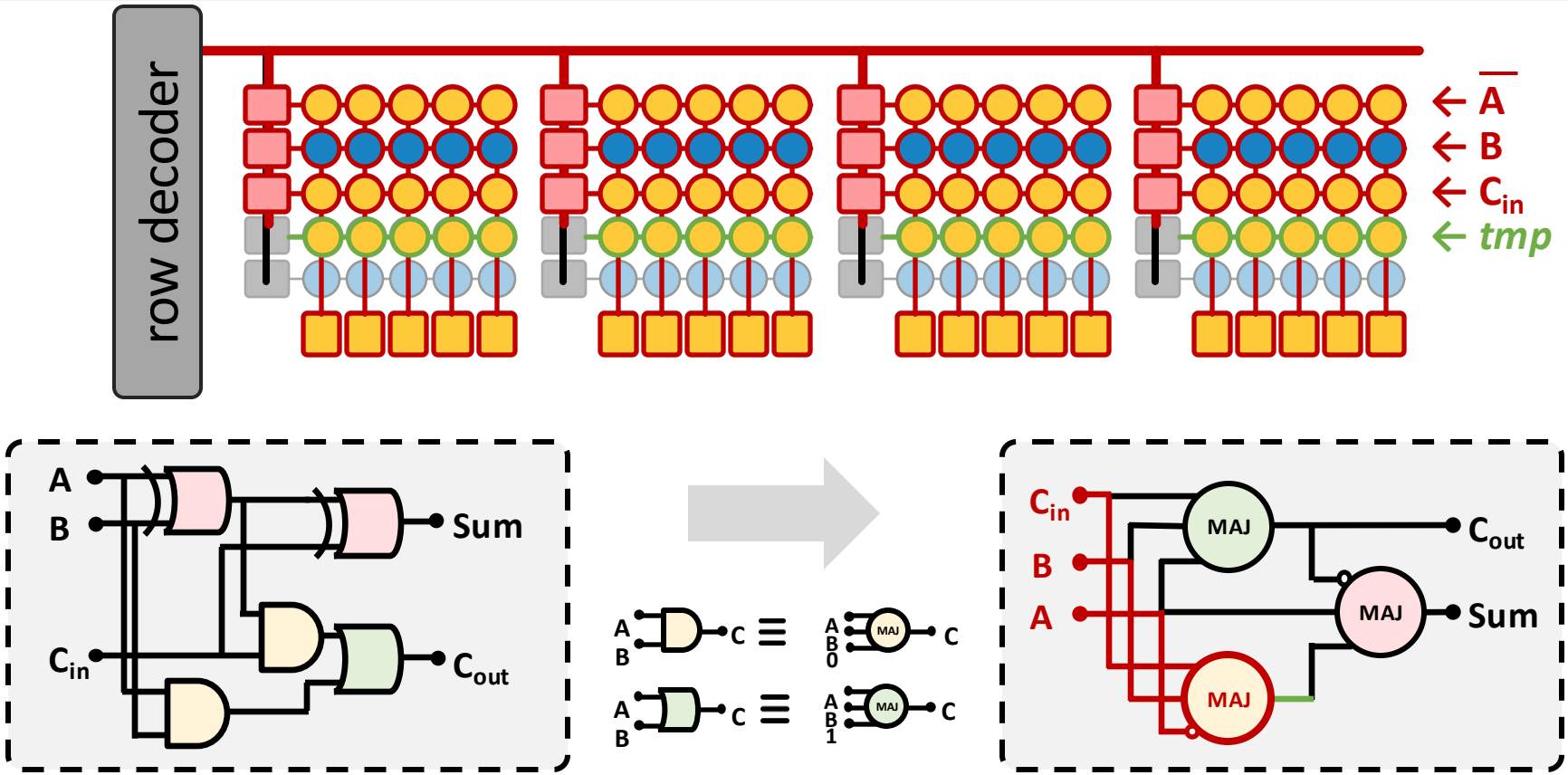
Bulk bitwise arithmetic can be performed by
orchestrating in-DRAM row copy and majority operations



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

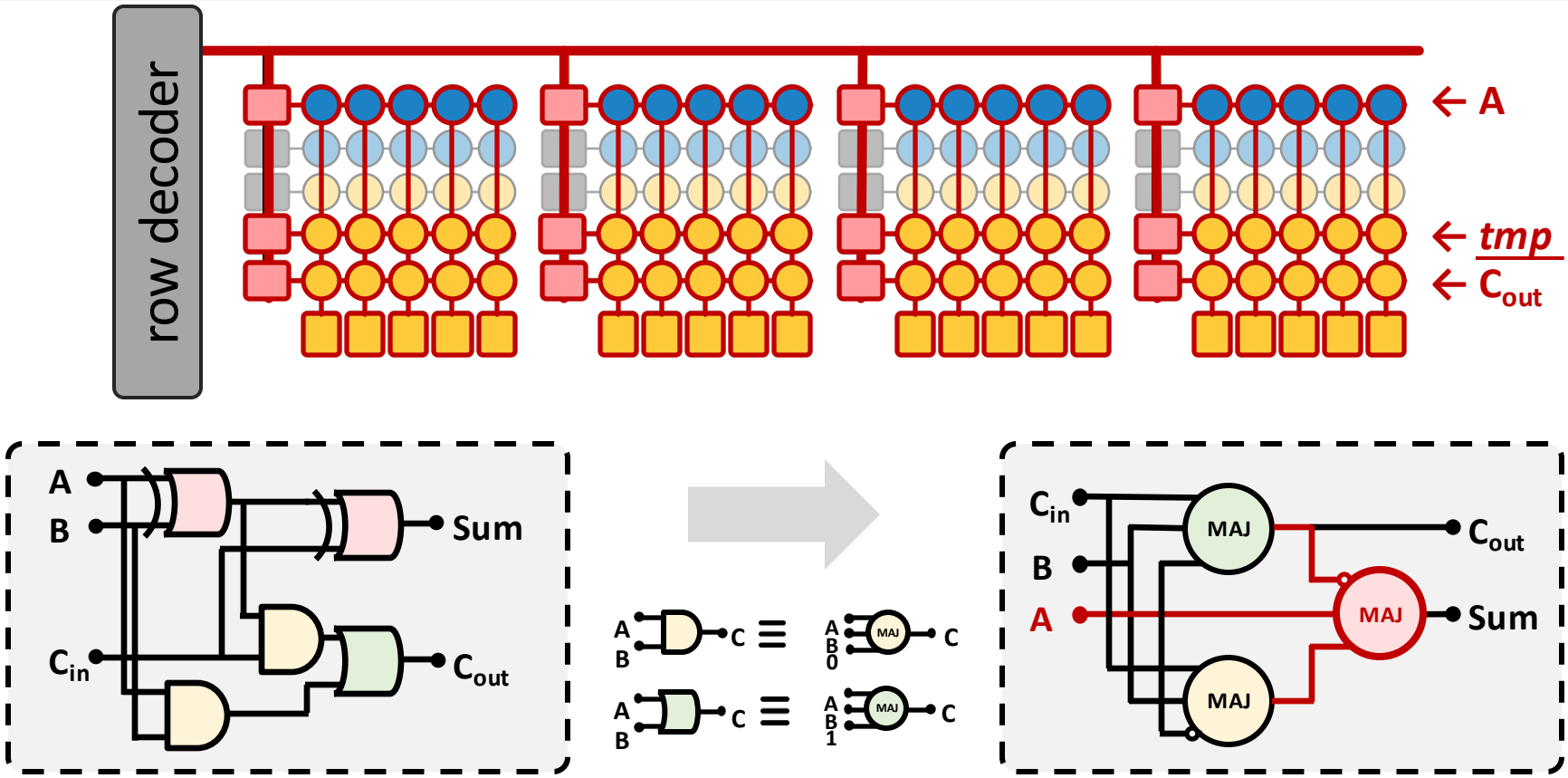
Bulk bitwise arithmetic can be performed by
orchestrating in-DRAM row copy and majority operations



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

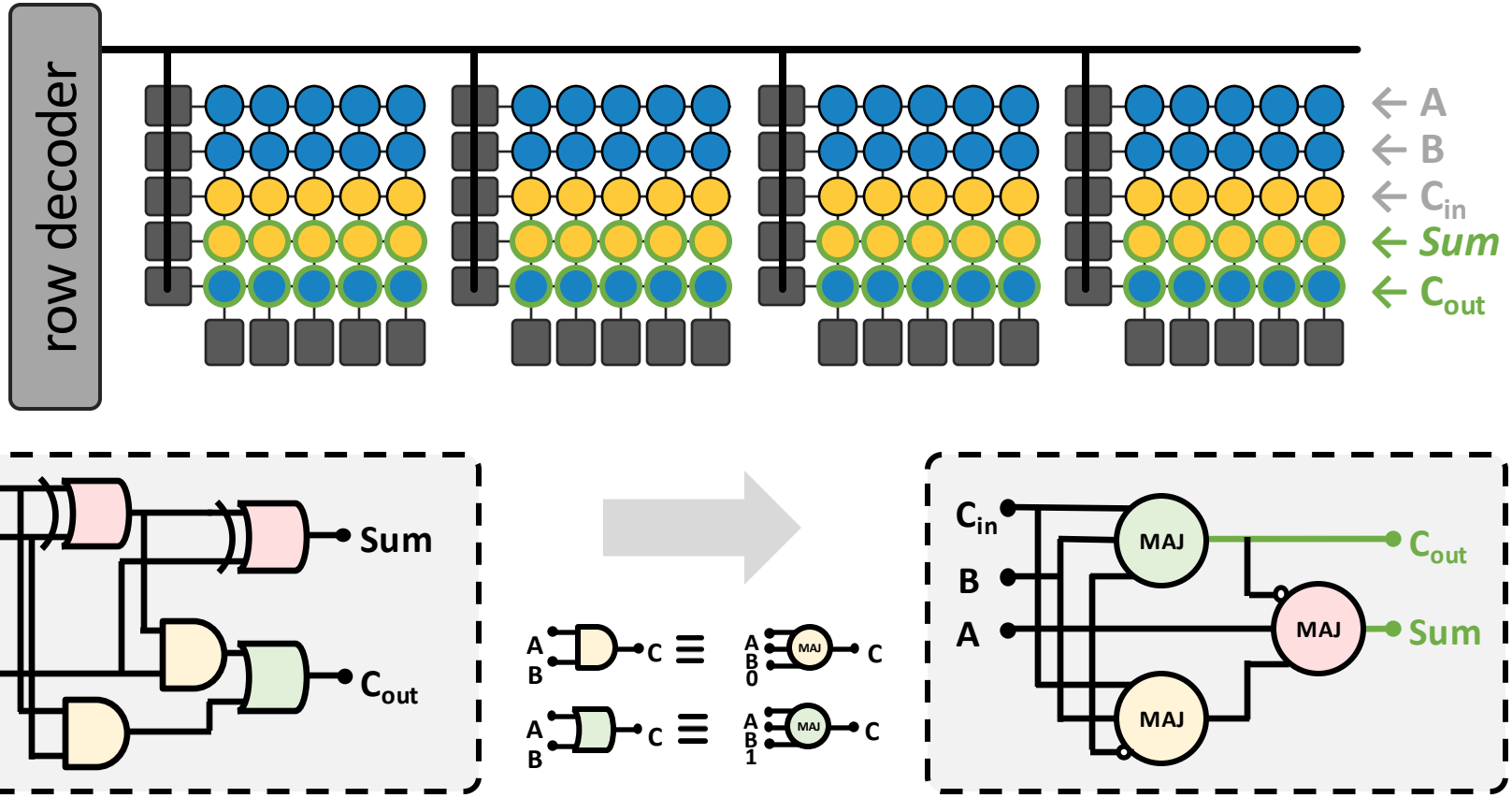
Bulk bitwise arithmetic can be performed by
orchestrating in-DRAM row copy and majority operations



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

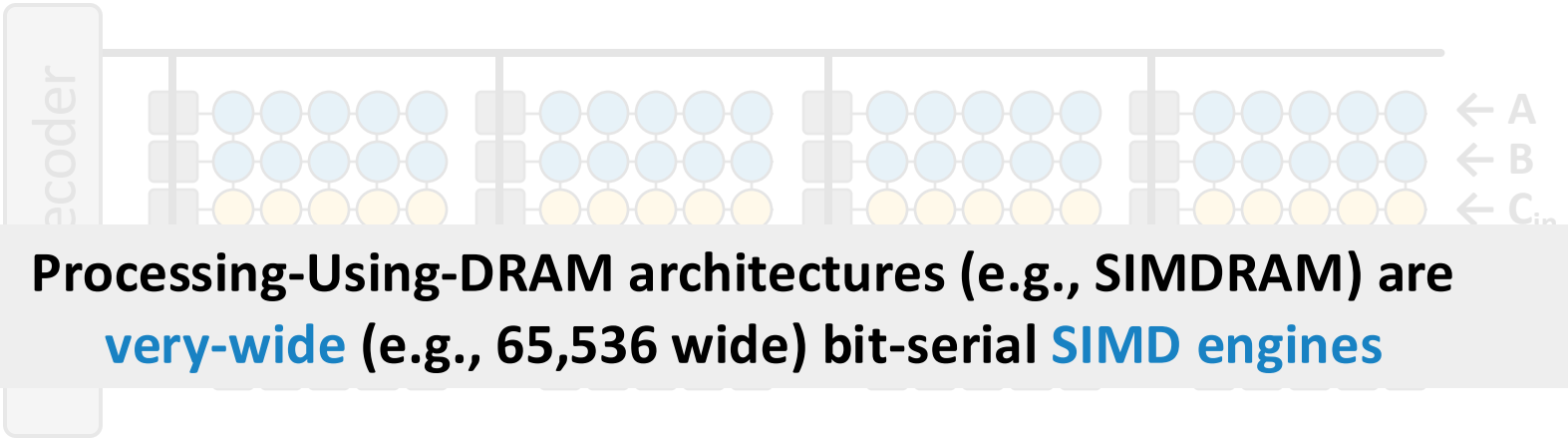
Bulk bitwise arithmetic can be performed by
orchestrating in-DRAM row copy and majority operations



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

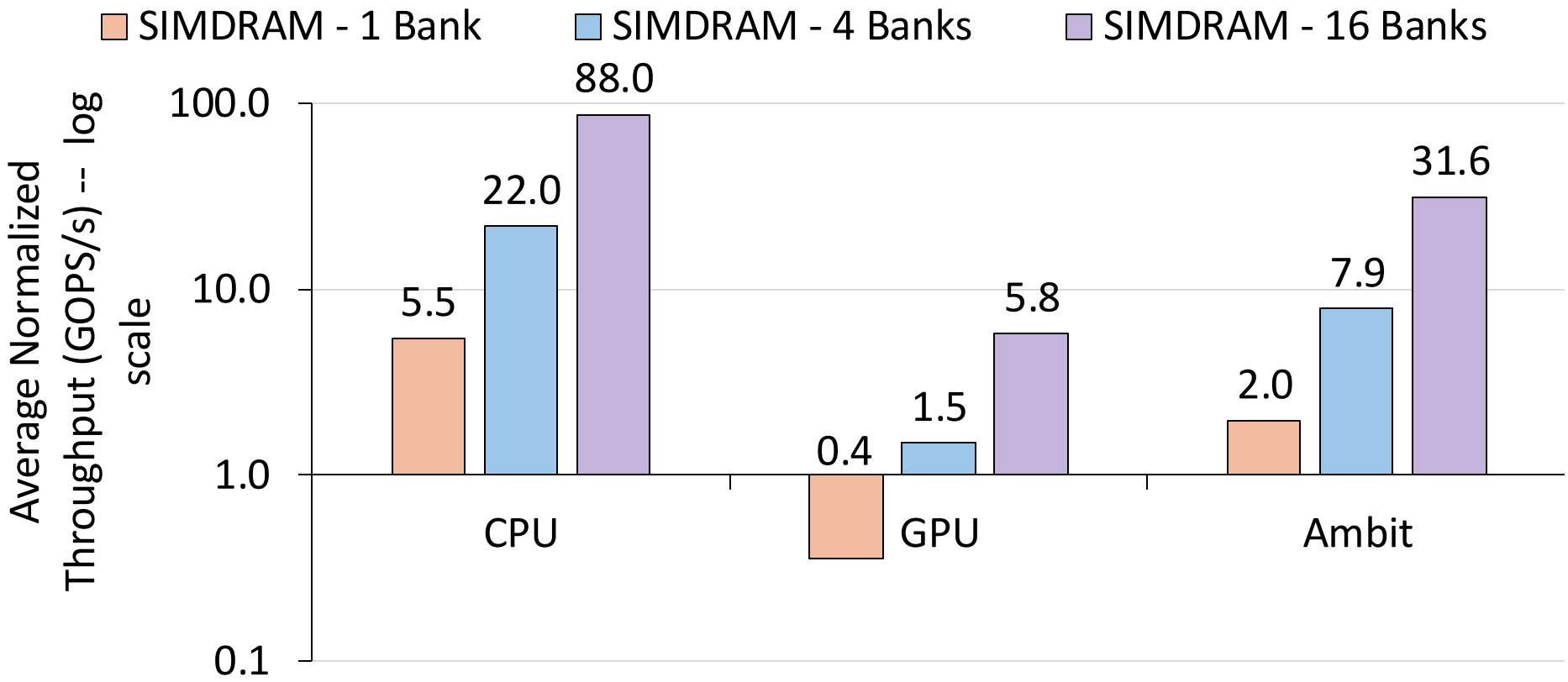
Evaluation:
Methodology Overview (II)

- **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

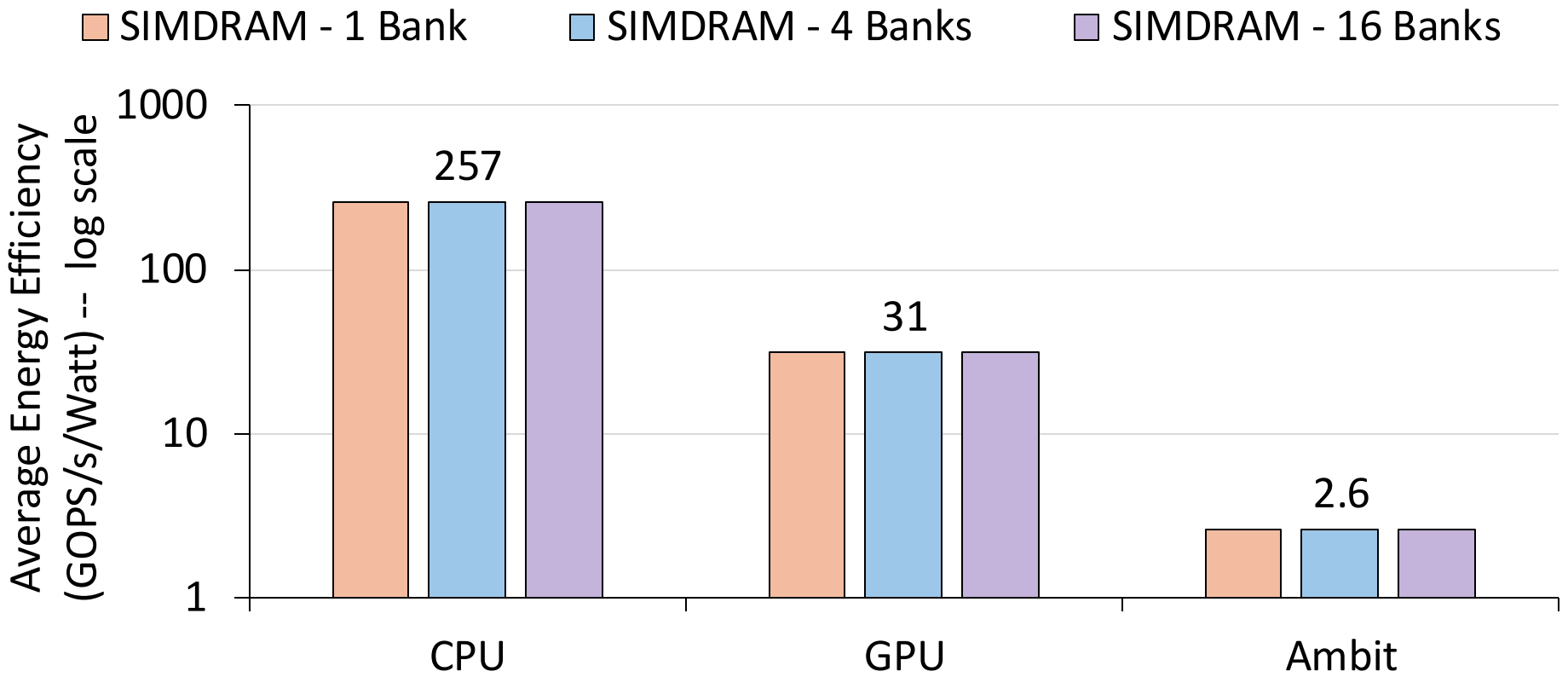
Average normalized throughput across all 16 SIMDRAM operations



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

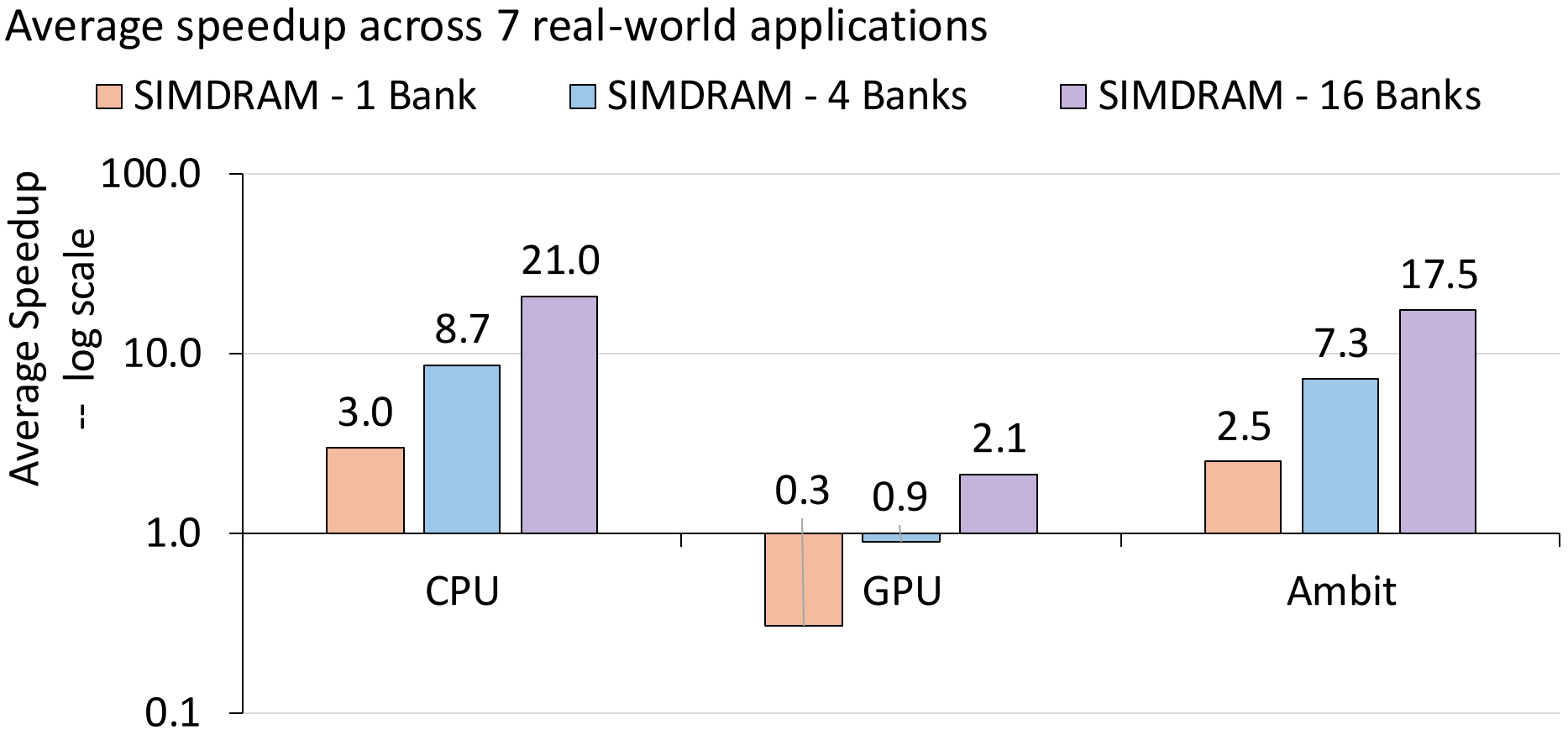
Evaluation:
Energy Analysis

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

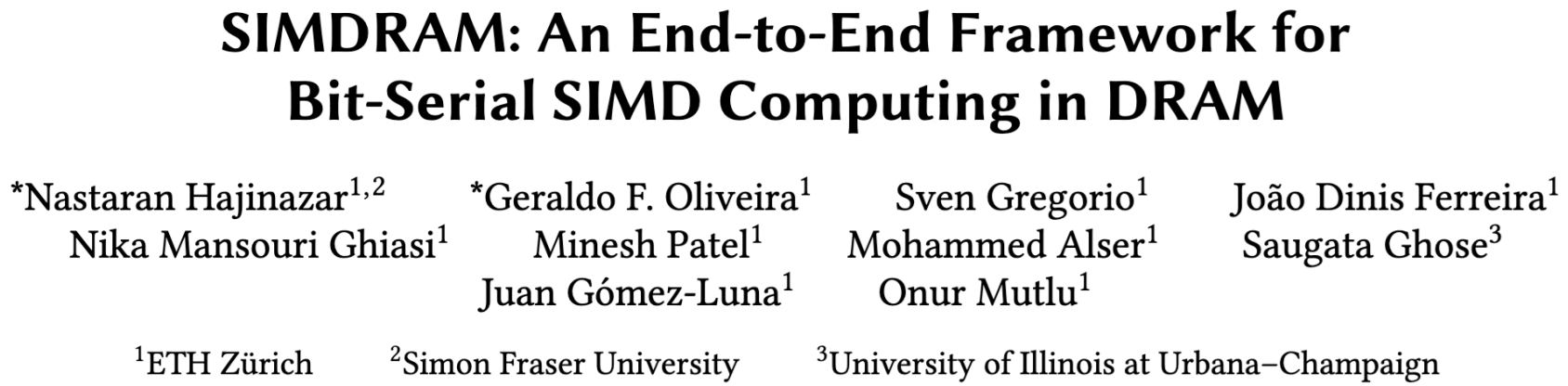
Evaluation:
Real-World Applications



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

Frameworks for PUM:
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

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

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

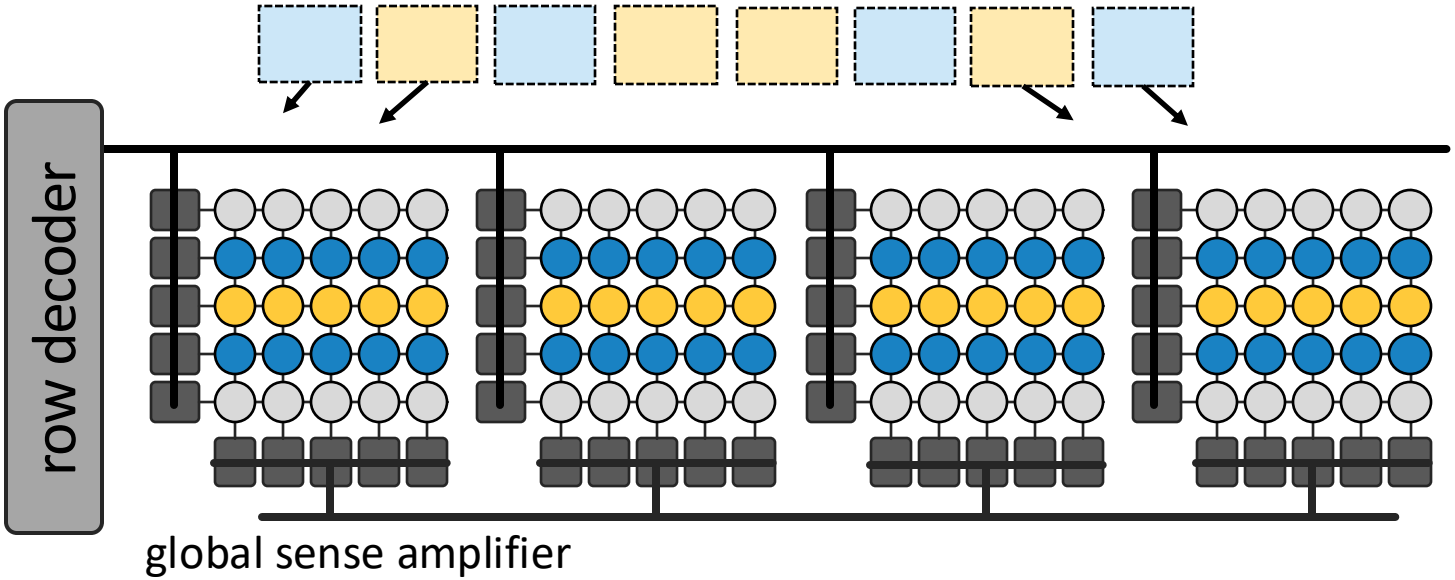
Limitations of PUD Systems:
Underutilization of SIMD Lanes (I)

Application Analysis:

quantify the fraction of SIMD parallelism in real applications

Maximum Vectorization Factor:

maximum number of scalar operands that fit into a SIMD register

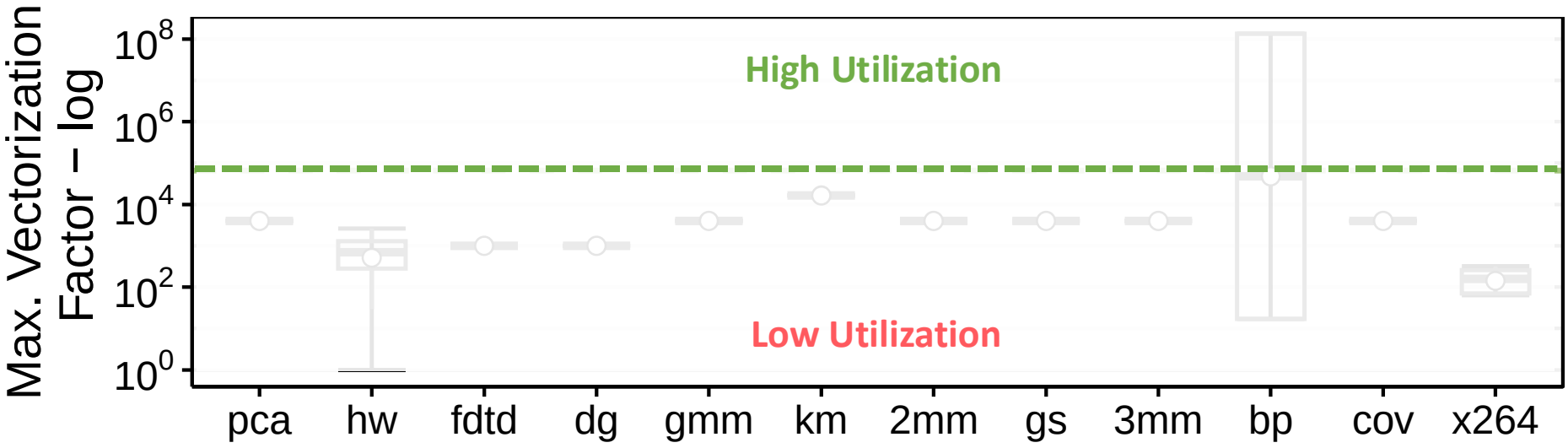


Ideal maximum vectorization factor = # DRAM columns (e.g., 65,536)

Limitations of PUD Systems:
Underutilization of SIMD Lanes (II)

Application Analysis:

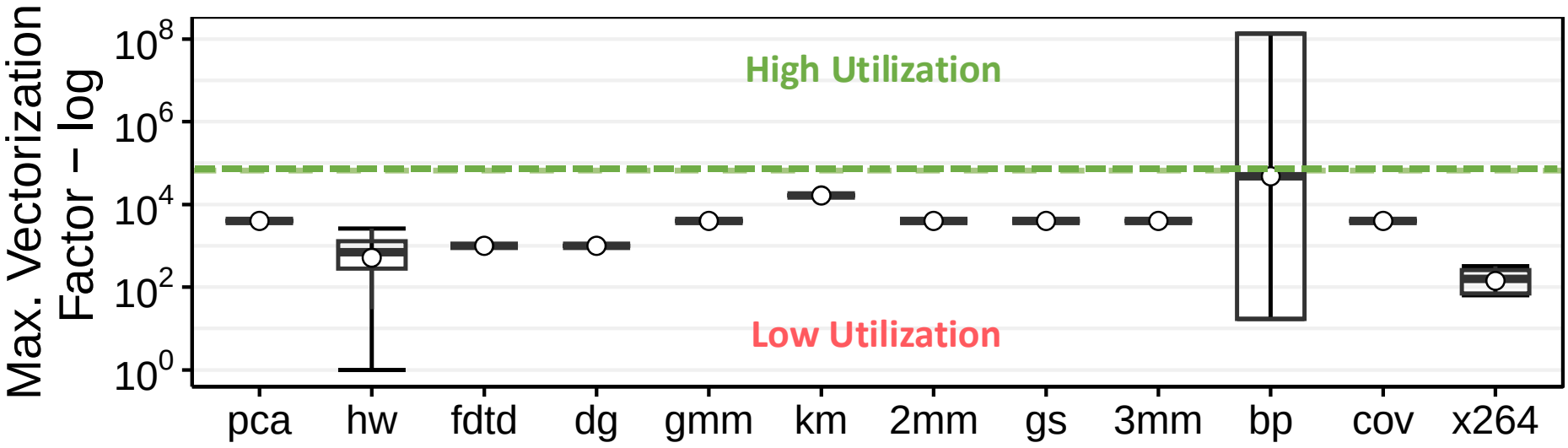
quantify the fraction of SIMD parallelism in real applications



Limitations of PUD Systems:
Underutilization of SIMD Lanes (II)

Application Analysis:

quantify the fraction of SIMD parallelism in real applications



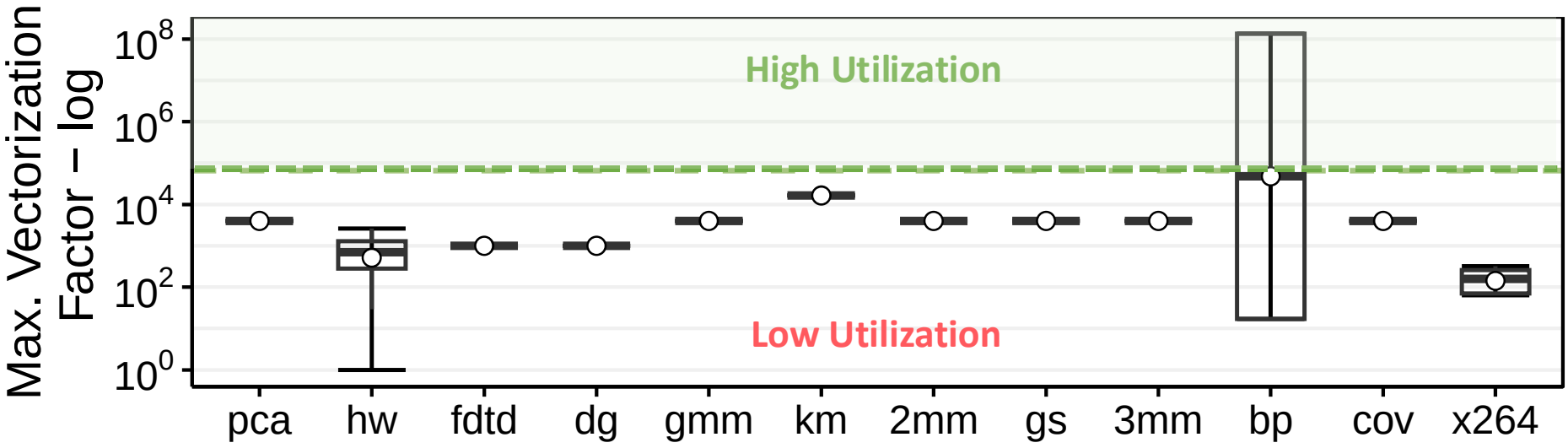
Takeaway

SIMD parallelism significantly varies within a single application and across different applications

Limitations of PUD Systems:
Underutilization of SIMD Lanes (III)

Application Analysis:

quantify the fraction of SIMD parallelism in real applications



Takeaway

A small fraction of vectorized loops have a large enough maximum vectorization factor to fully exploit the SIMD parallelism of PUD systems

Limitations of PUD Systems:
Overview

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

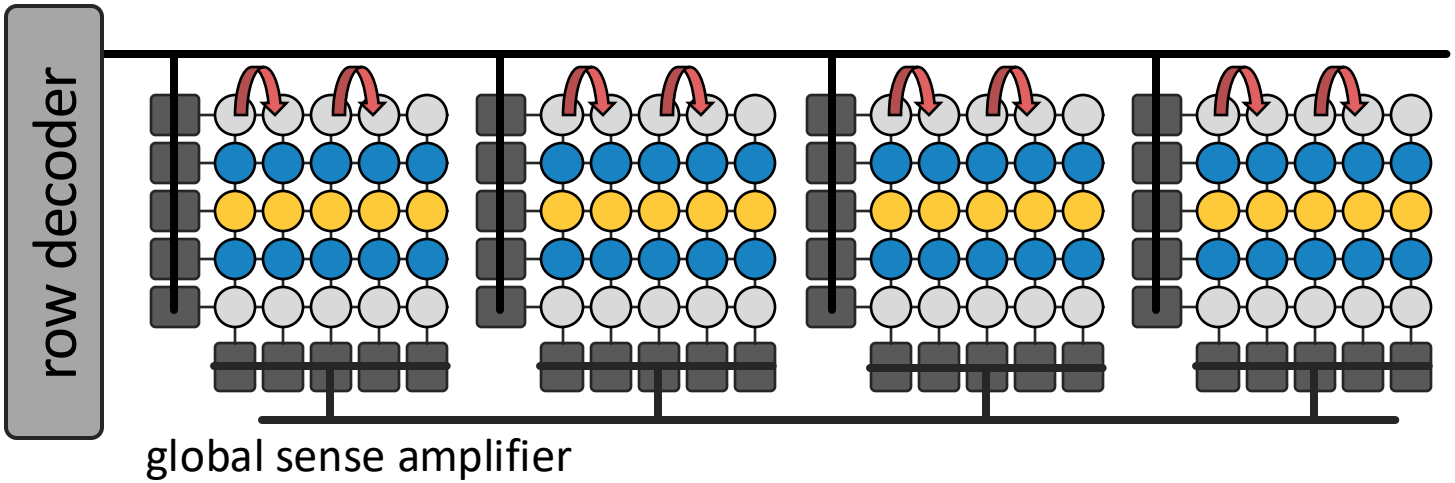
1 SIMD Underutilization
- due to data parallelism variation within and across applications
- leads to throughput and energy waste

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

Limitations of PUD Systems:
Limited Computation Support

PUD systems do not support vector reduction at low area cost since data movement is bounded to within a DRAM column



no direct communication path across columns

Takeaway

Directly connecting all DRAM columns using a custom all-to-all interconnect leads to large (i.e., 21%) area cost

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

Limitations of PUD Systems:
Challenging Programming Model

Programmer's Tasks:

Goal:

Just write my kernel

High-level code for

C[i] = (A[i] > pred[i])? A[i] + B[i] : A[i] – B[i]

```
for (int i = 0; i < size ; ++ i){
   bool cond = A[i] > pred[i];
   if (cond) C[i] = A[i] + B[i];
   else  C[i] = A[i] - B[i];
}
```

Limitations of PUD Systems:
Challenging Programming Model

Programmer's Tasks:

Map & align data structures

Goal:

Just write my kernel

High-level code for

C[i] = (A[i] > pred[i])? A[i] + B[i] : A[i] – B[i]

```
for (int i = 0; i < size ; ++ i){
   bool cond = A[i] > pred[i];
   if (cond) C[i] = A[i] + B[i];
   else  C[i] = A[i] - B[i];
}
```

Limitations of PUD Systems:
Challenging Programming Model

Programmer's Tasks:

- Map & align data structures
- Identify array boundaries

Goal:

- Just write my kernel

High-level code for

C[i] = (A[i] > pred[i])? A[i] + B[i] : A[i] – B[i]

```
for (int i = 0; i < size ; ++ i){
   bool cond = A[i] > pred[i];
   if (cond) C[i] = A[i] + B[i];
   else  C[i] = A[i] - B[i];
}
```

Limitations of PUD Systems:
Challenging Programming Model

Programmer's Tasks:

Map & align data structures	Identify array boundaries	Manually unroll loop	Map C to PUD instructions

Goal:

Just write my kernel

High-level code for

C[i] = (A[i] > pred[i])? A[i] + B[i] : A[i] – B[i]

```
for (int i = 0; i < size ; ++ i){
   bool cond = A[i] > pred[i];
   if (cond) C[i] = A[i] + B[i];
   else  C[i] = A[i] - B[i];
}
```

Limitations of PUD Systems:
Challenging Programming Model

Programmer's Tasks:

Map & align data structures	Identify array boundaries	Manually unroll loop	Map C to PUD instructions	Orchestrate data movement

Goal: Just write my kernel

High-level code for

C[i] = (A[i] > pred[i])? A[i] + B[i] : A[i] – B[i]

```
for (int i = 0; i < size ; ++ i){
   bool cond = A[i] > pred[i];
   if (cond) C[i] = A[i] + B[i];
   else  C[i] = A[i] - B[i];
}
```

Limitations of PUD Systems:
Challenging Programming Model

Programmer's Tasks:

Map & align data structures	Identify array boundaries	Manually unroll loop	Map C to PUD instructions	Orchestrate data movement	**Goal:** Just write my kernel

PUD's assembly-like code for

C[i] = (A[i] > pred[i])? A[i] + B[i] : A[i] – B[i]

```
bbop_trsp_init(A , size , elm_size);
bbop_trsp_init(B , size , elm_size);
bbop_trsp_init(C , size , elm_size);

bbop_add(D , A , B , size , elm_size);
bbop_sub(E , A , B , size , elm_size);
bbop_greater(F , A , pred , size , elm_size);
bbop_if_else(C , D , E , F , size , elm_size);
```

Problem & Goal

Problem

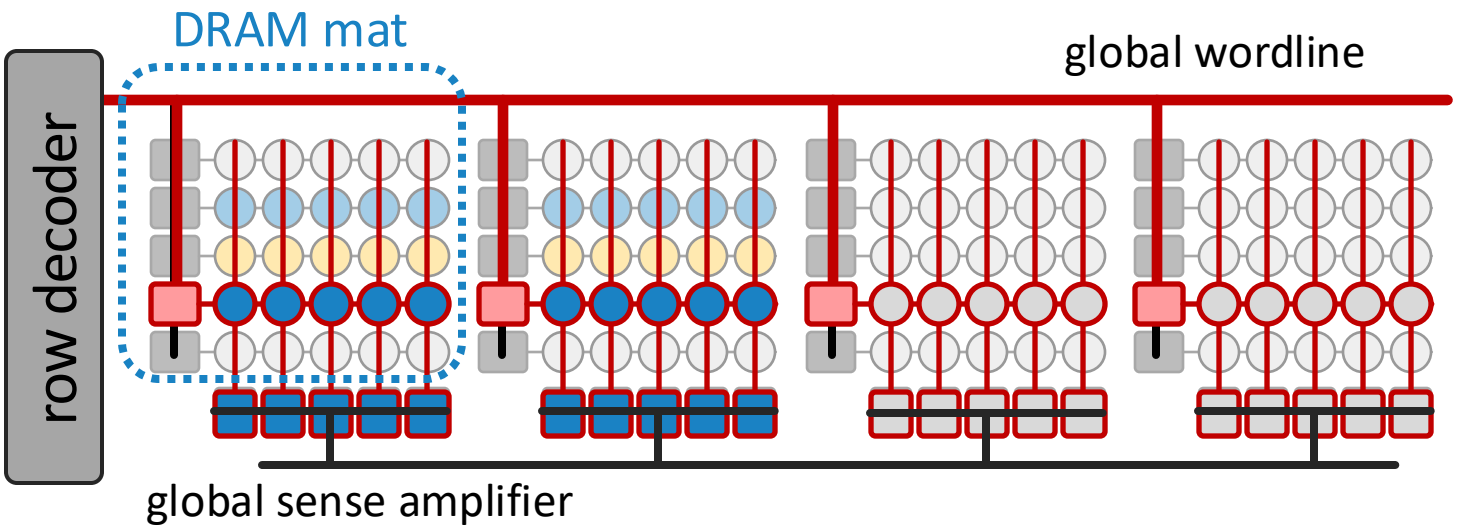
Processing-Using-DRAM's large and rigid granularity limits its applicability and efficiency for different applications

Goal

Design a flexible PUD system that overcomes the three limitations caused by large and rigid DRAM access granularity

MIMDRAM:
Key Idea (I)

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

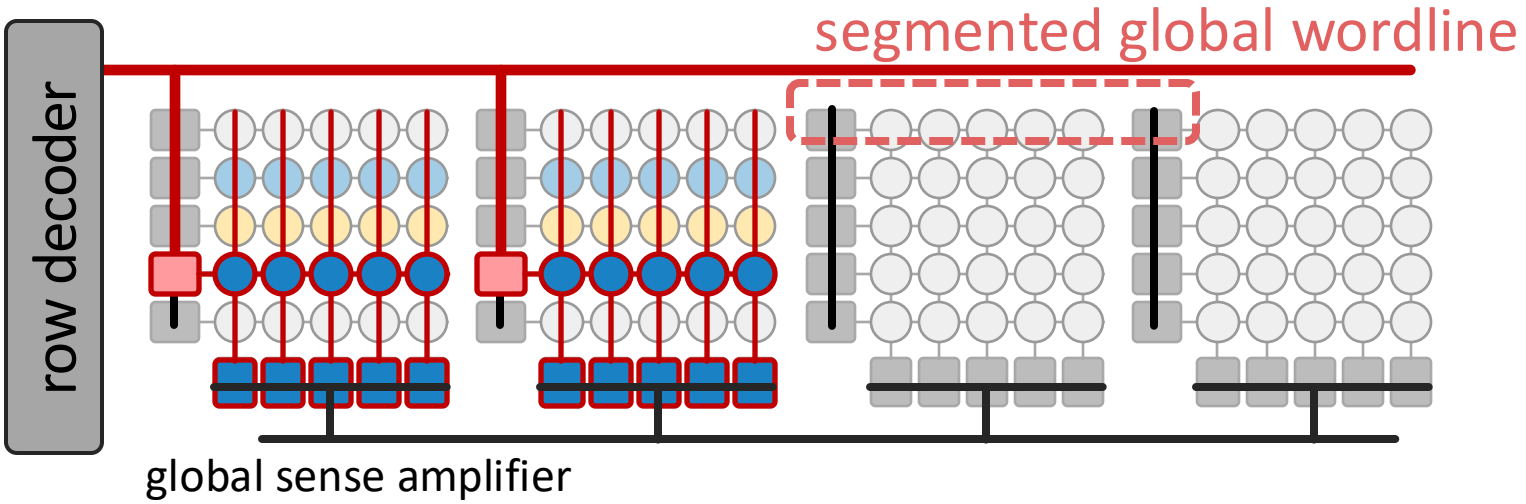


Key Issue:
on a DRAM access, the global wordline propagates across all DRAM mats

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

MIMDRAM:
Key Idea (II)

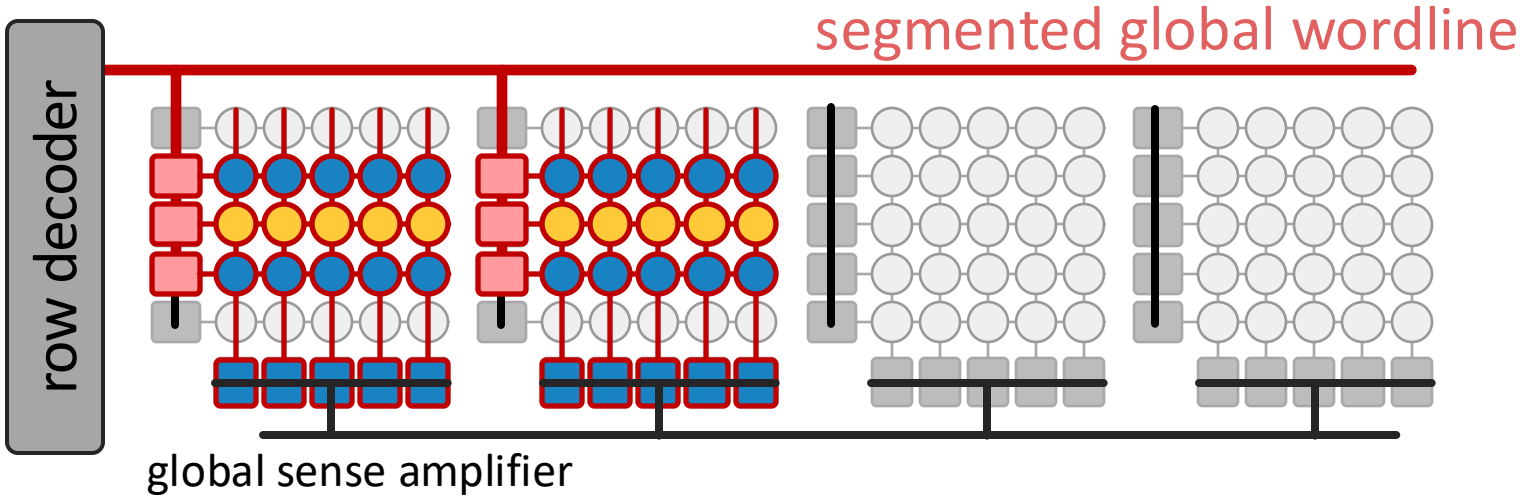
Fine-Grained DRAM:
segments the global wordline to access individual DRAM mats



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)

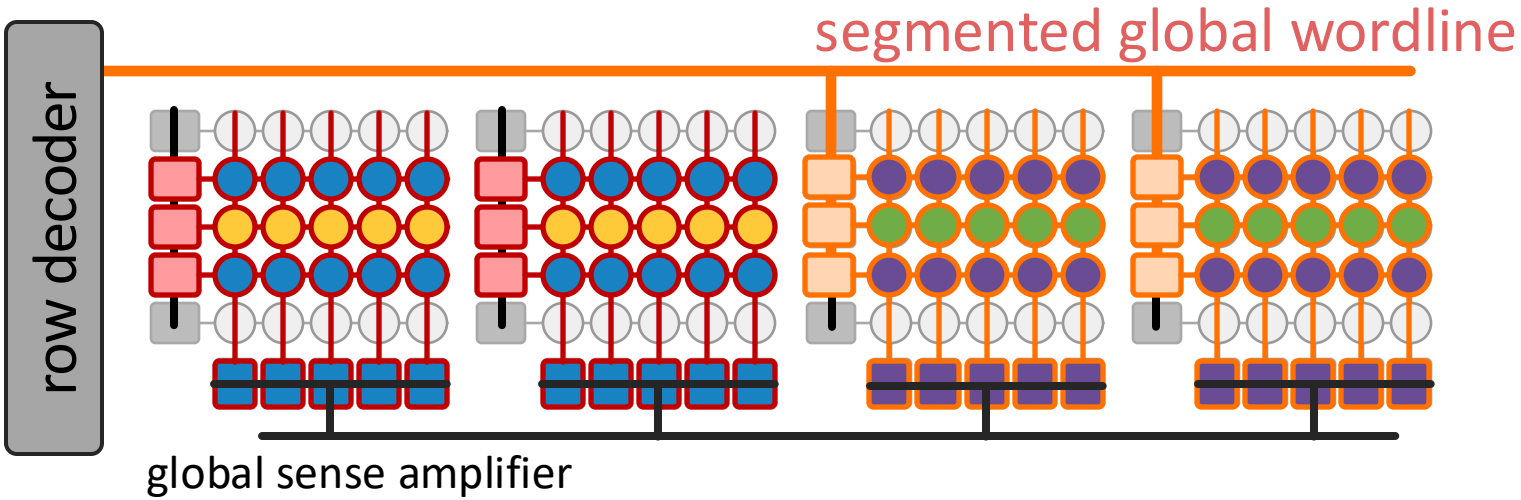


Fine-grained DRAM for processing-using-DRAM:

1 Improves SIMD utilization

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

MIMDRAM:
Key Idea (III)

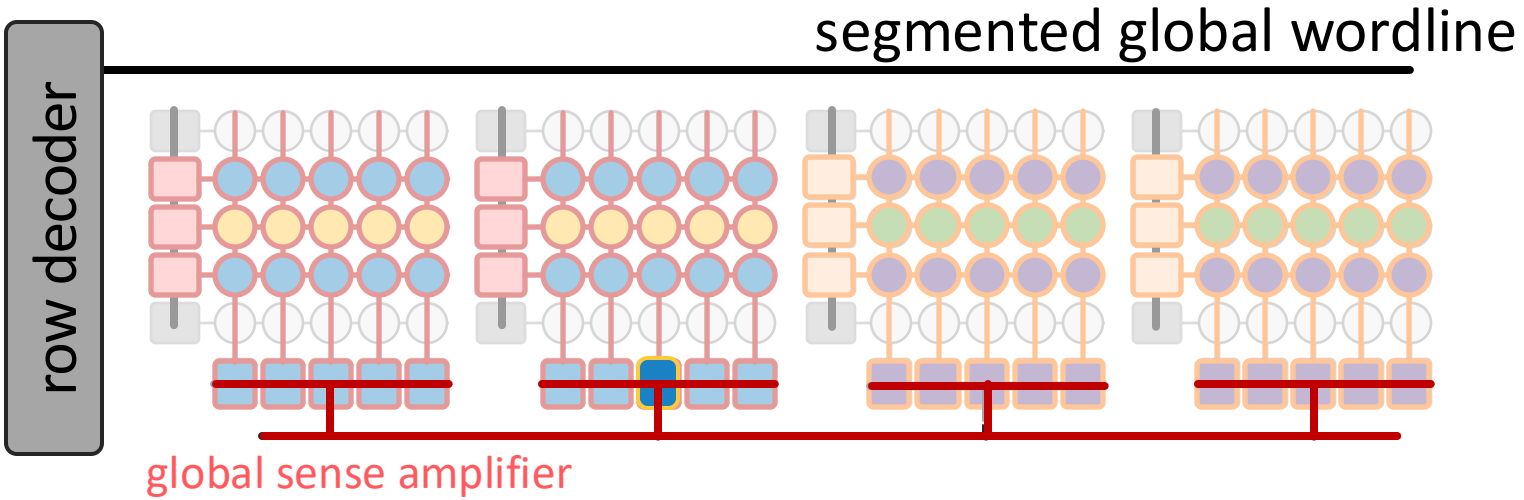


Fine-grained DRAM for processing-using-DRAM:

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:

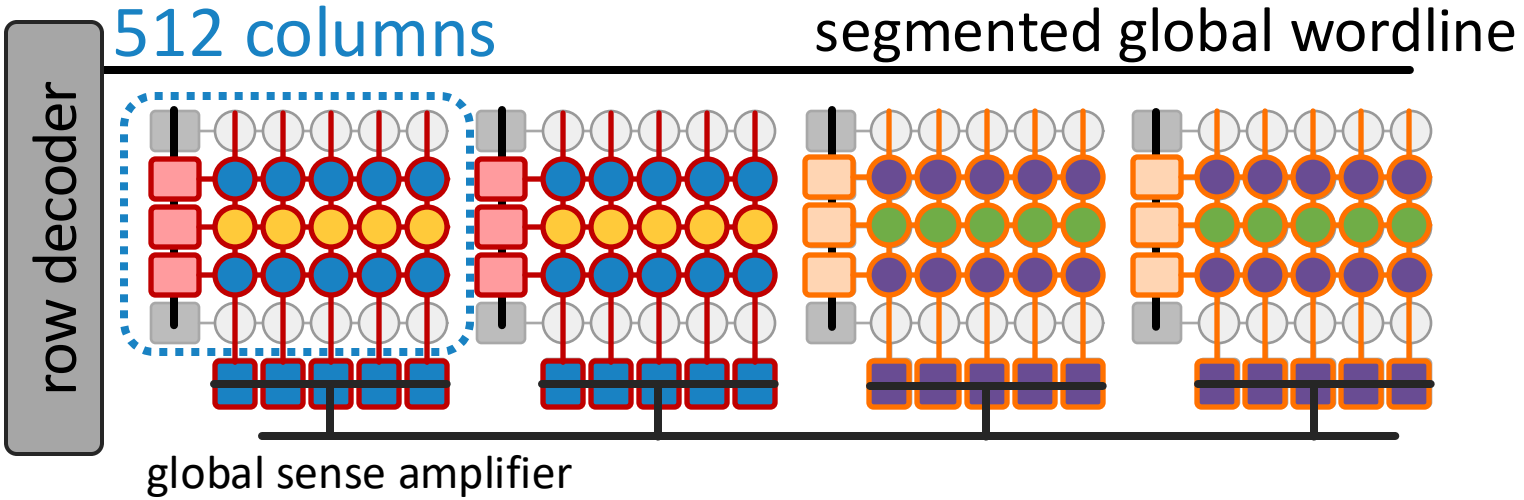
1 Improves SIMD utilization

- 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:

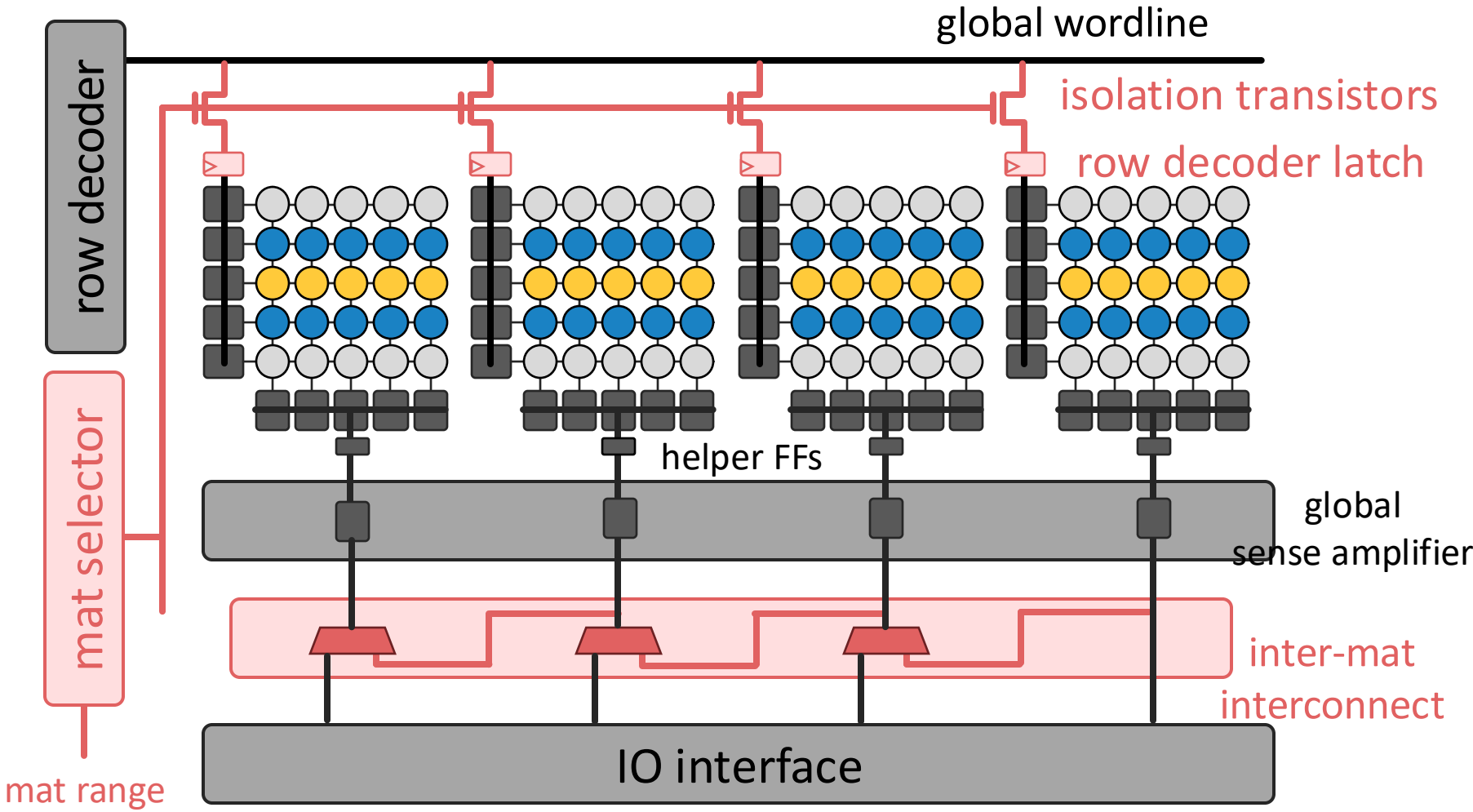
1 Hardware

- 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

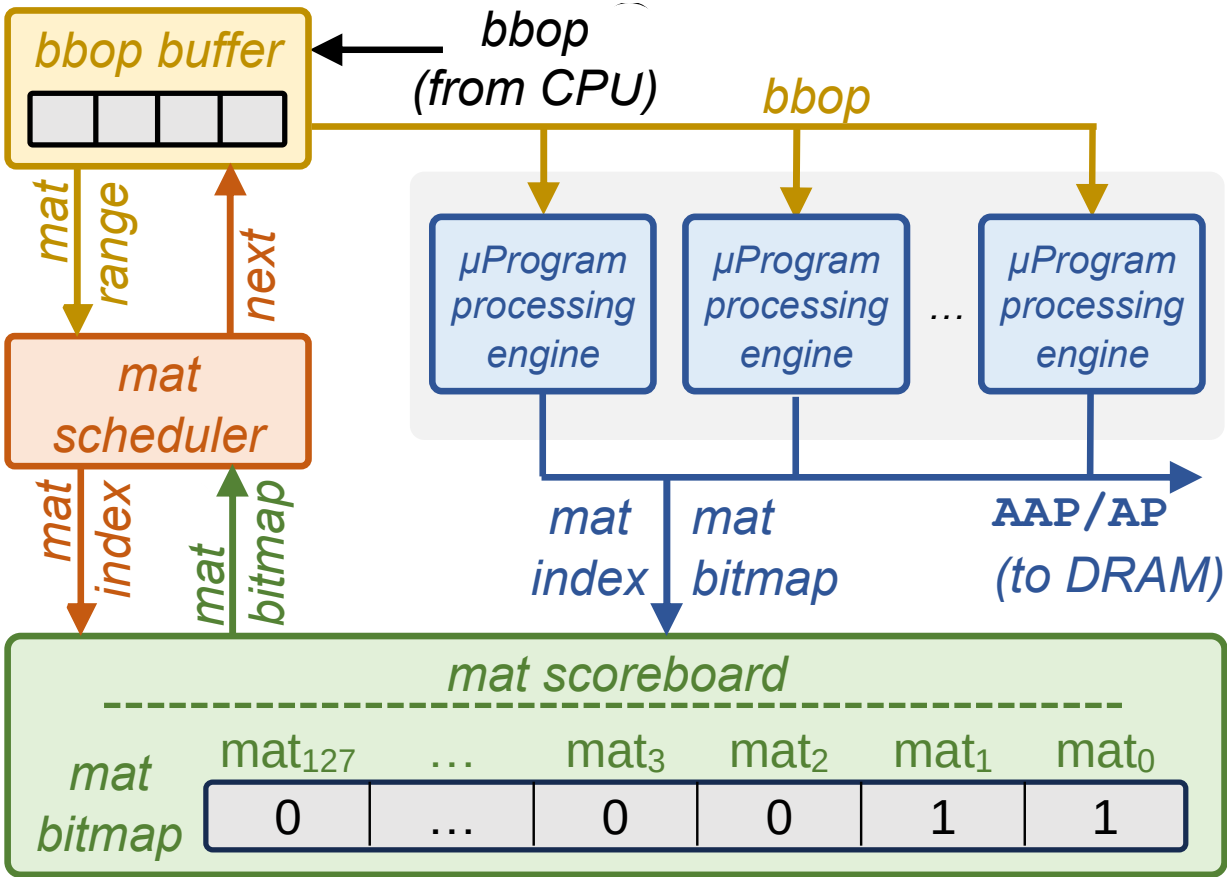
- compiler support to transparently generate PUD instructions
- system support to map and execute PUD instructions

MIMDRAM:
Modifications to DRAM Chip



MIMDRAM:
Control Unit Design

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

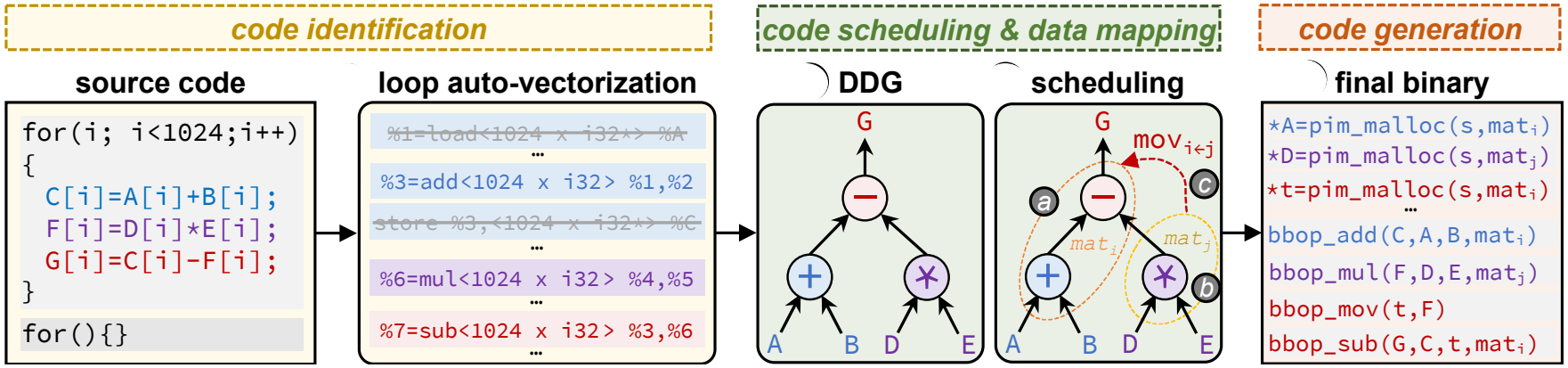


MIMDRAM:
Compiler Support

Goal

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

Three new LLVM-based passes targeting PUD execution

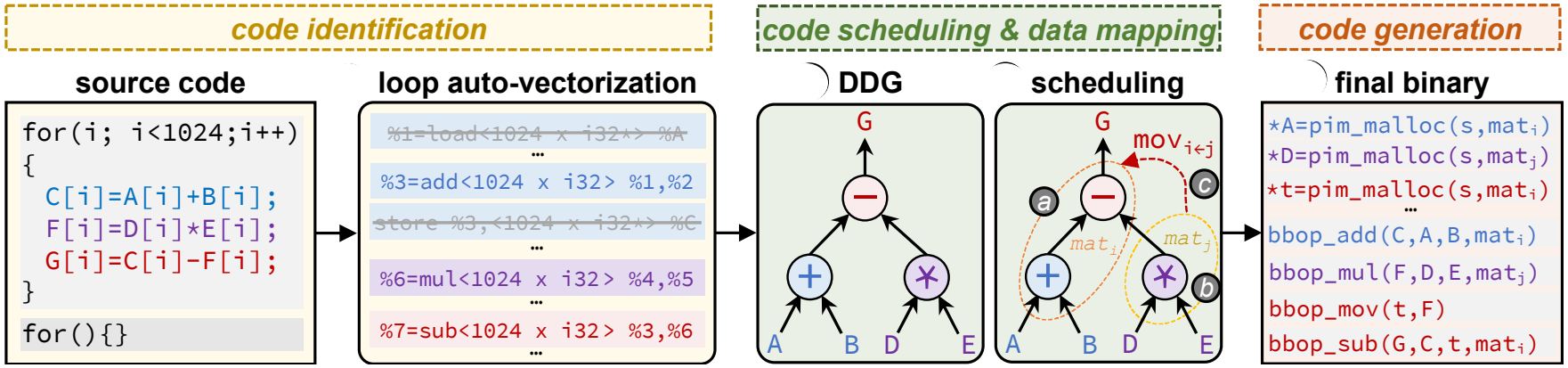


MIMDRAM:
Compiler Support (I)

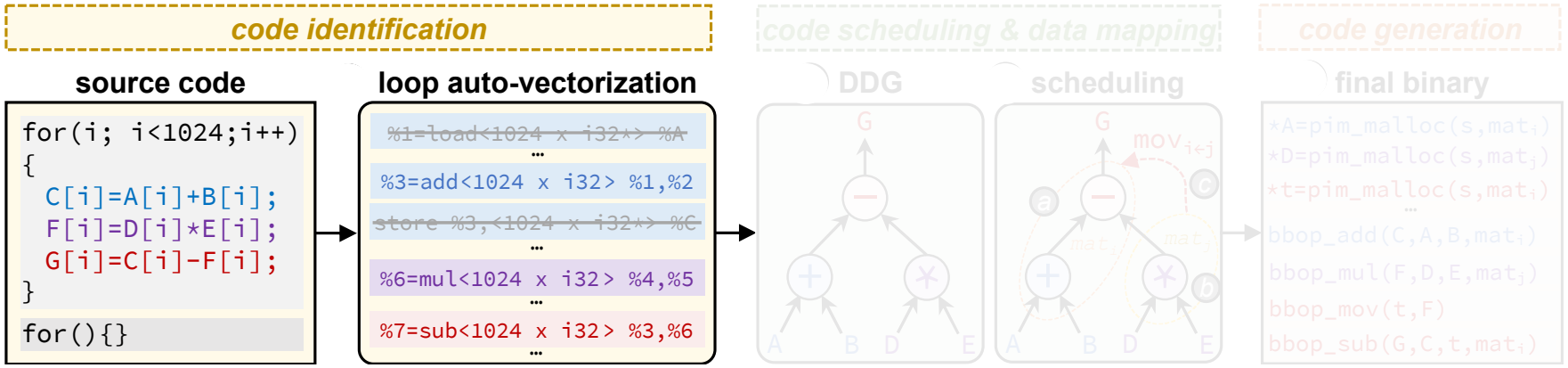
Goal

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

Three new LLVM-based passes targeting PUD execution



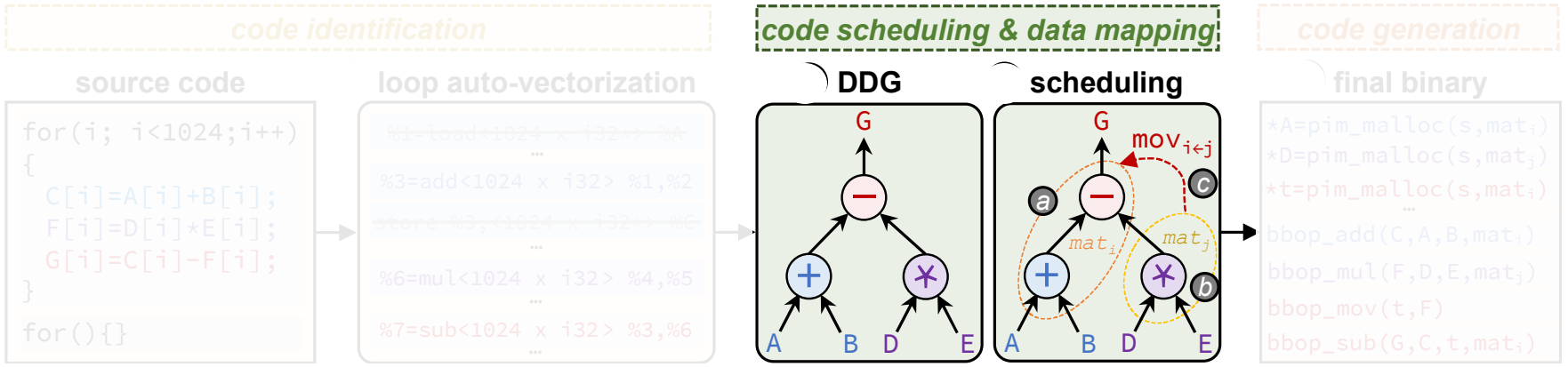
MIMDRAM:
Compiler Support (II)



Goal

Identify SIMD parallelism, generate PUD instructions, and set the appropriate vectorization factor

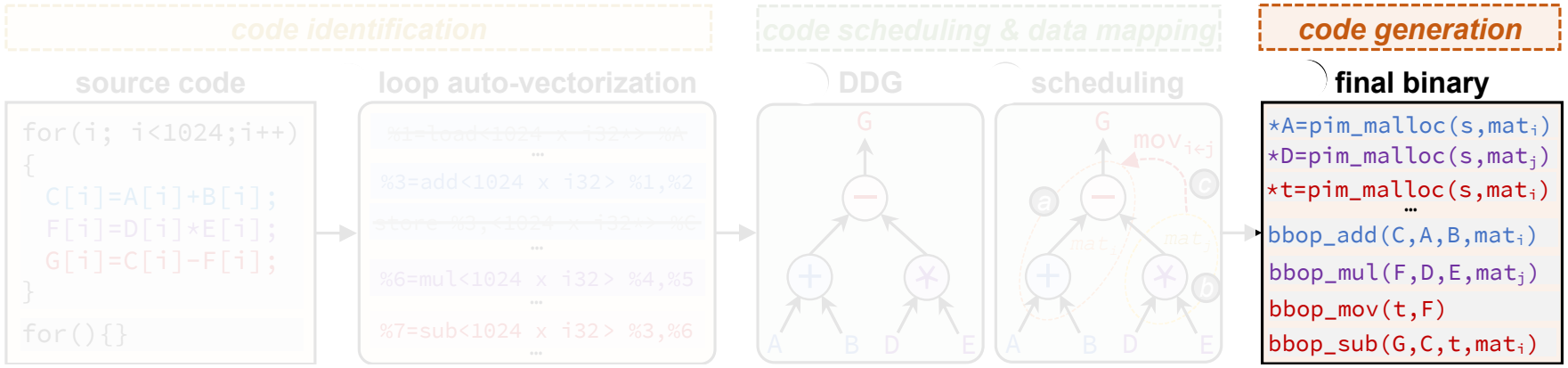
MIMDRAM:
Compiler Support (II)



Goal: Identify SIMD parallelism, generate PUD instructions, and set the appropriate vectorization factor

Goal: **Improve SIMD utilization by allowing the distribution of independent PUD instructions across DRAM mats**

MIMDRAM:
Compiler Support (III)



Goal

Identify SIMD parallelism, generate PUD instructions, and set the appropriate vectorization factor

Goal

Improve SIMD utilization by allowing the distribution of independent PUD instructions across DRAM mats

Goal

Generate the appropriate binary for data allocation and 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-side**
- 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:
System Support

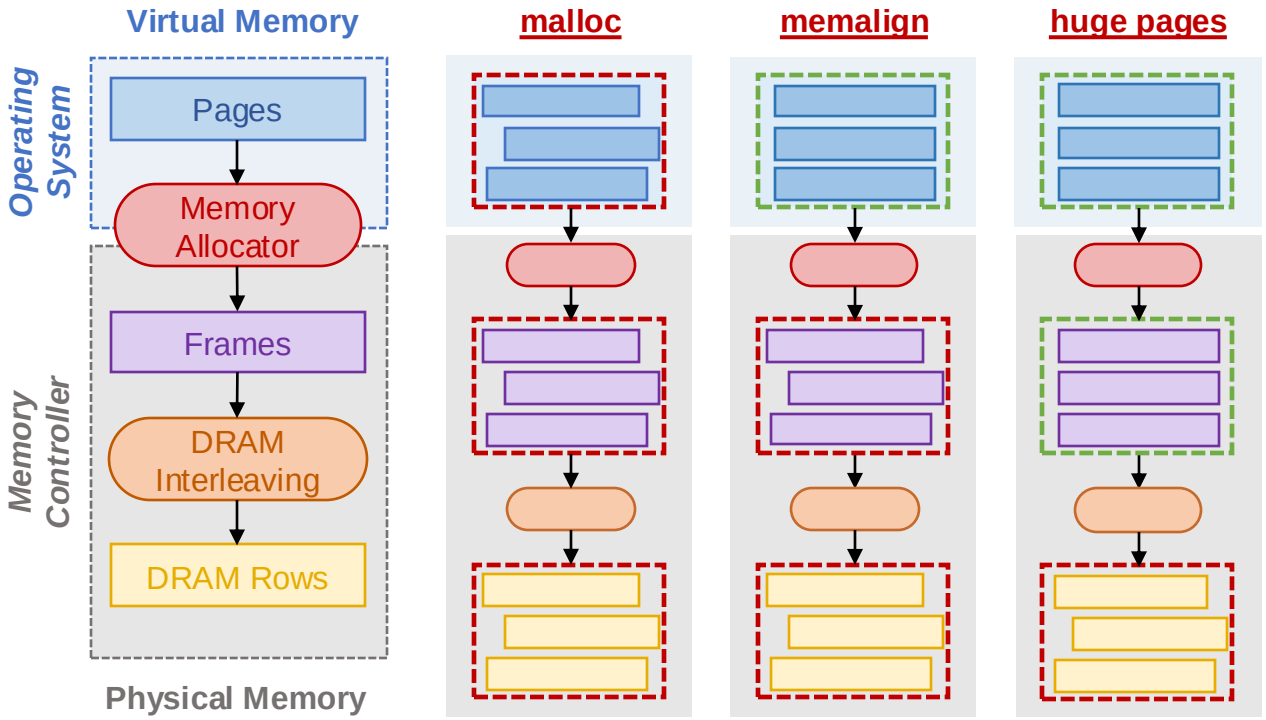
- Instruction set architecture
- Execution & data transposition
- Data coherence
- Address translation
- Data allocation & alignment
- Mat label translation

MIMDRAM:
System Support

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

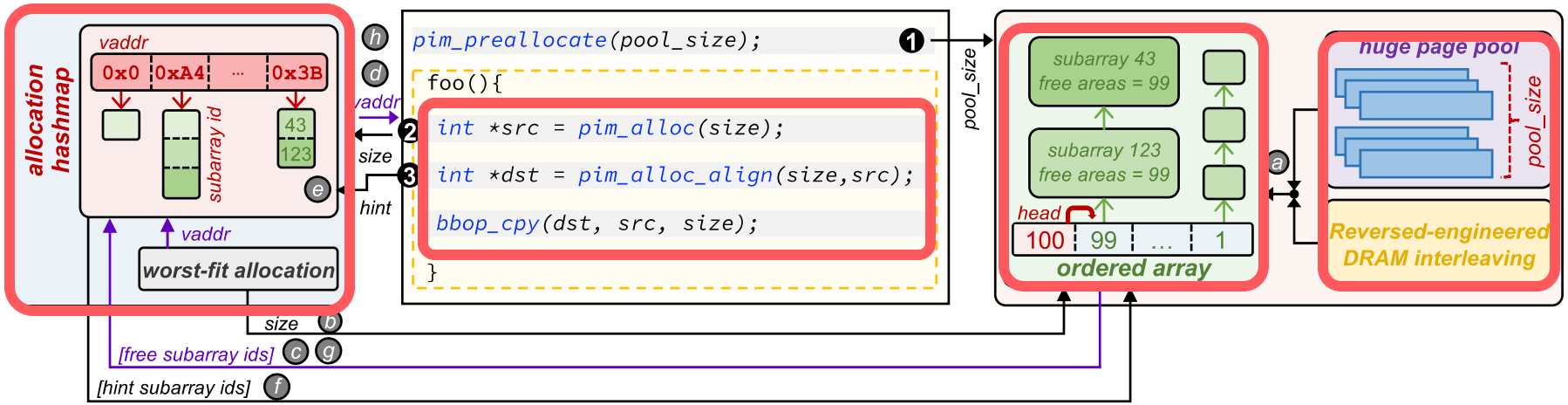
MIMDRAM:
Data Allocation & Alignment

PUD systems require OS support to guarantee that data is properly mapped and aligned within bank/subarray/mat a
→ not a natively supported operation



MIMDRAM:
Data Allocation & Alignment

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



MIMDRAM:
More in the Paper & GitHub

- Instruction set architecture
- Data allocation & alignment
- Mat label translation

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

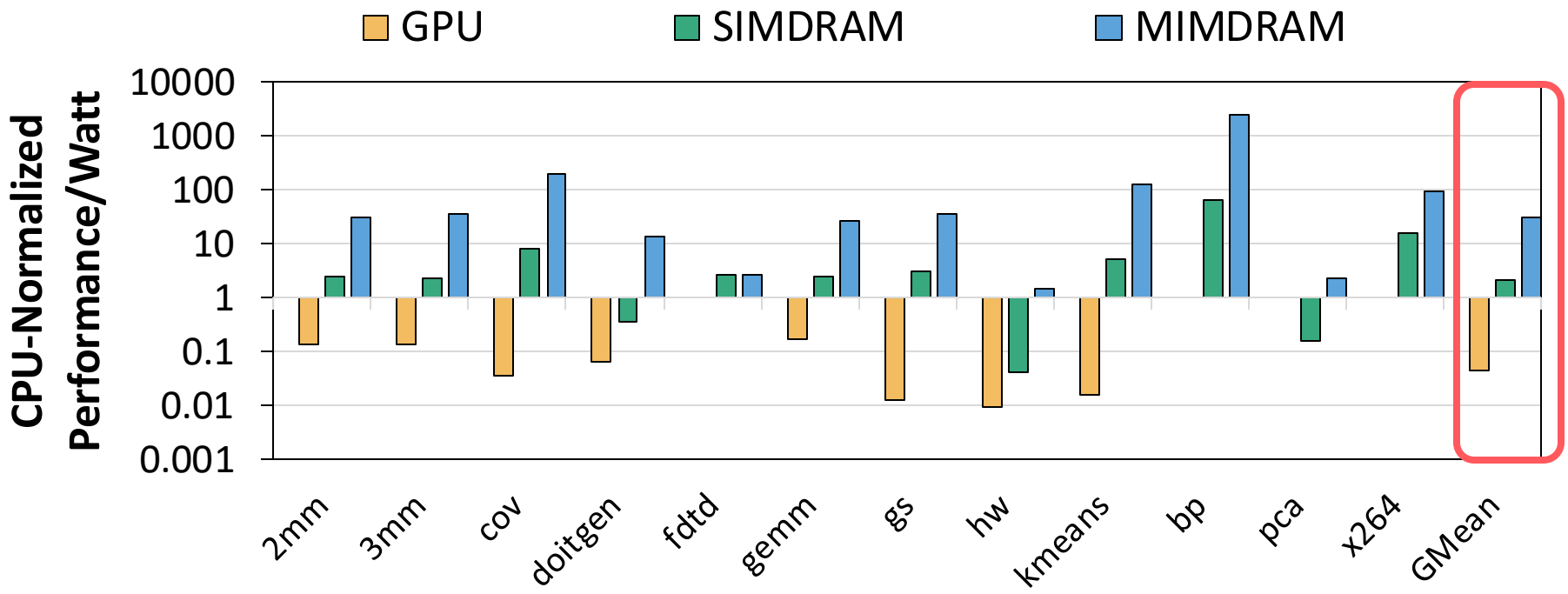
Geraldo F. Oliveira[†] Ataberk Olgun[†] Abdullah Giray Yağlıkçı[†] F. Nisa Bostancı[†]
Juan Gómez-Luna[†] Saugata Ghose[‡] Onur Mutlu[†]

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

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

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

Evaluation:
Single Application Analysis – Energy Efficiency

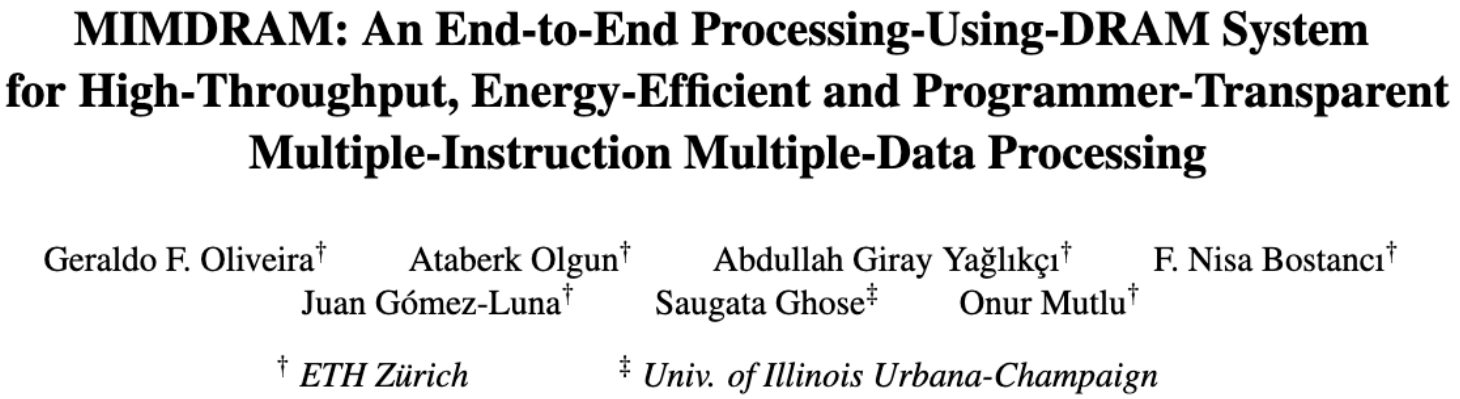


Takeaway

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

More on MIMDRAM

- Geraldo F. Oliveira, Ataberk Olgun, Abdullah Giray Yağlıkçı, F. Nisa Bostancı, Juan Gómez-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 Computing"**
  *Proceedings of the 30th International Symposium on High-Performance Computer Architecture (**HPCA**)*, Edinburgh, Scotland, March 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*

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

Limitations of PUD Systems:
Overview

PUD systems suffer from three sources of inefficiency due to the naive use of a bit-serial execution model

1 Static Data Representation
- leading to a subpar performance in the presence of *narrow values*
- opportunity 1: reduce the bit-precision of a PUD operation

2 Throughput-Oriented Execution
- failing to reduce the latency of a *single* PUD operations
- opportunity 2: use SALP to concurrently execute PUD primitives

3 High-Precision Computation
- due to the linear/quadratically scaling nature of bit-serial algorithms
- opportunity 3: use alternative data representation for high-precision PUD operations

Limitations of PUD Systems:
Overview

PUD systems suffer from three sources of inefficiency due to the naive use of a bit-serial execution model

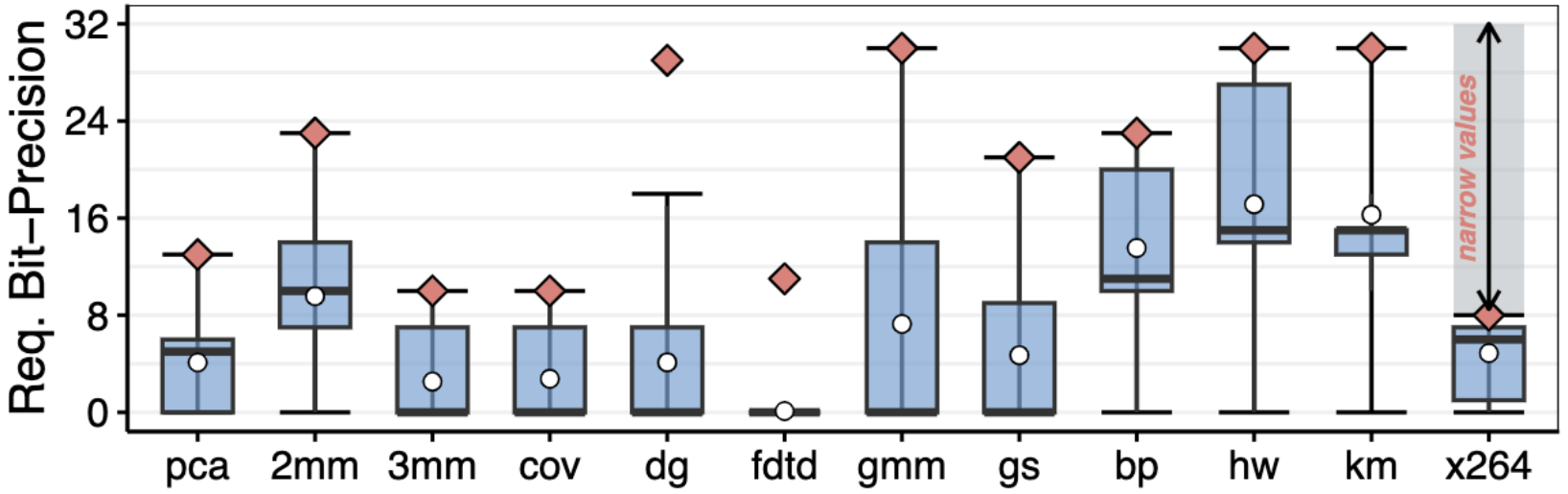
1 Static Data Representation
- leading to a subpar performance in the presence of *narrow values*
- opportunity 1: reduce the bit-precision of a PUD operation

2 Throughput-Oriented Execution
- failing to reduce the latency of a *single* PUD operations
- opportunity 2: use SALP to concurrently execute PUD primitives

3 High-Precision Computation
- due to the linear/quadratically scaling nature of bit-serial algorithms
- opportunity 3: use alternative data representation for high-precision PUD operations

Limitations of PUD Systems:
Static Data Representation (I)

Application Analysis:
quantify the required bit-precision in real applications

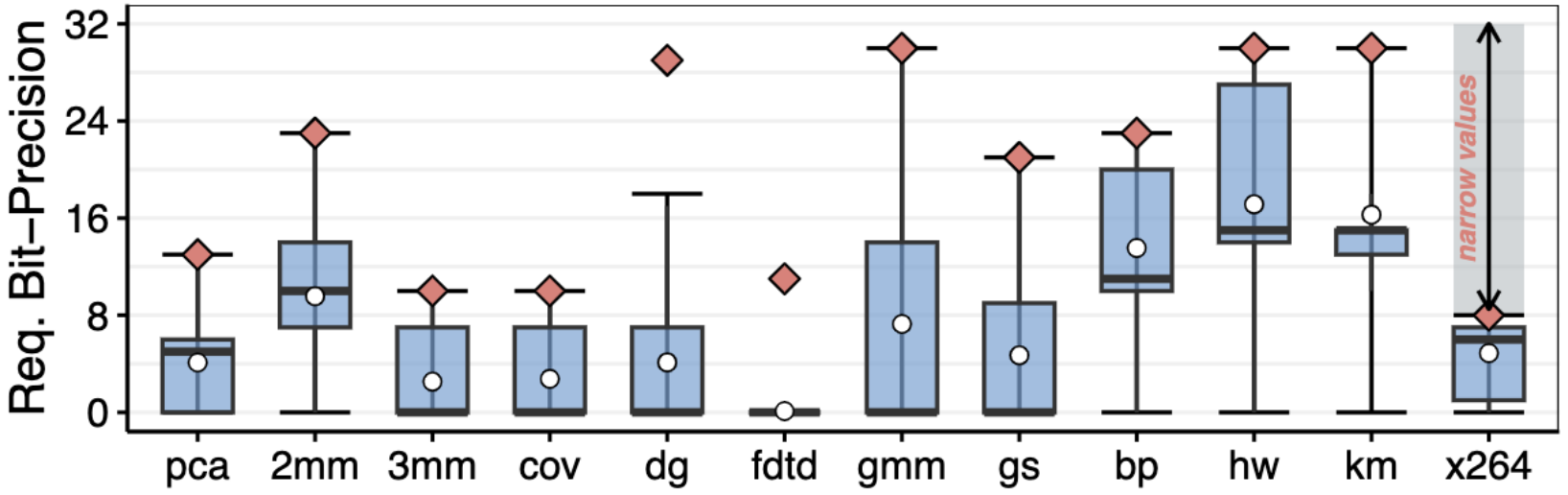


Takeaway

**Applications display a significant amount of narrow values →
bit-precision can be reduced from
32-bit to 20-bit (min. 8-bit, max. 30 bit) on average**

Limitations of PUD Systems:
Static Data Representation (II)

Application Analysis:
quantify the required bit-precision in real applications

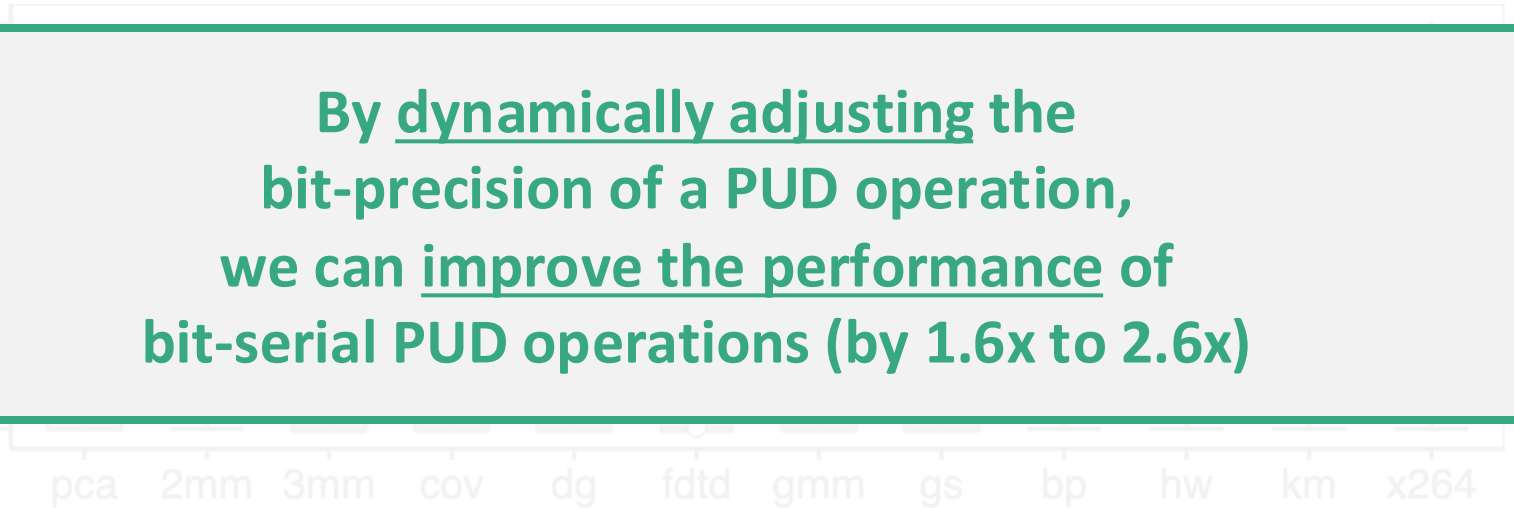


Takeaway

Bit-precision significantly varies for a given application →
indicates the need for a mechanism that can
dynamically identify the target bit-precision

Limitations of PUD Systems:
Static Data Representation (III)

Application Analysis:
quantify the required bit-precision in real applications



Key Idea

By dynamically adjusting the bit-precision of a PUD operation, we can improve the performance of bit-serial PUD operations (by 1.6x to 2.6x)

Limitations of PUD Systems:
Overview

PUD systems suffer from three sources of inefficiency due to the naive use of a bit-serial execution model

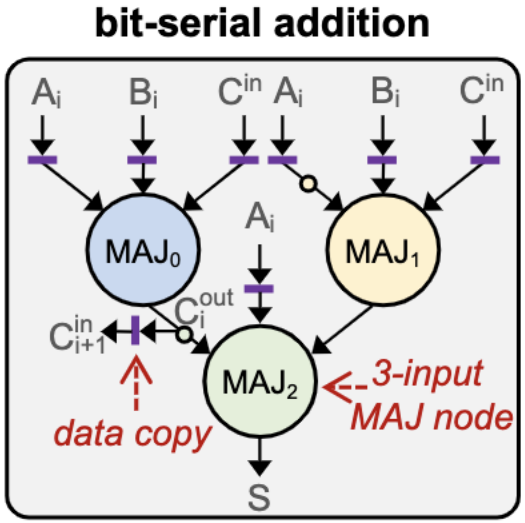
1 Static Data Representation
- leading to a subpar performance in the presence of *narrow values*
- opportunity 1: reduce the bit-precision of a PUD operation

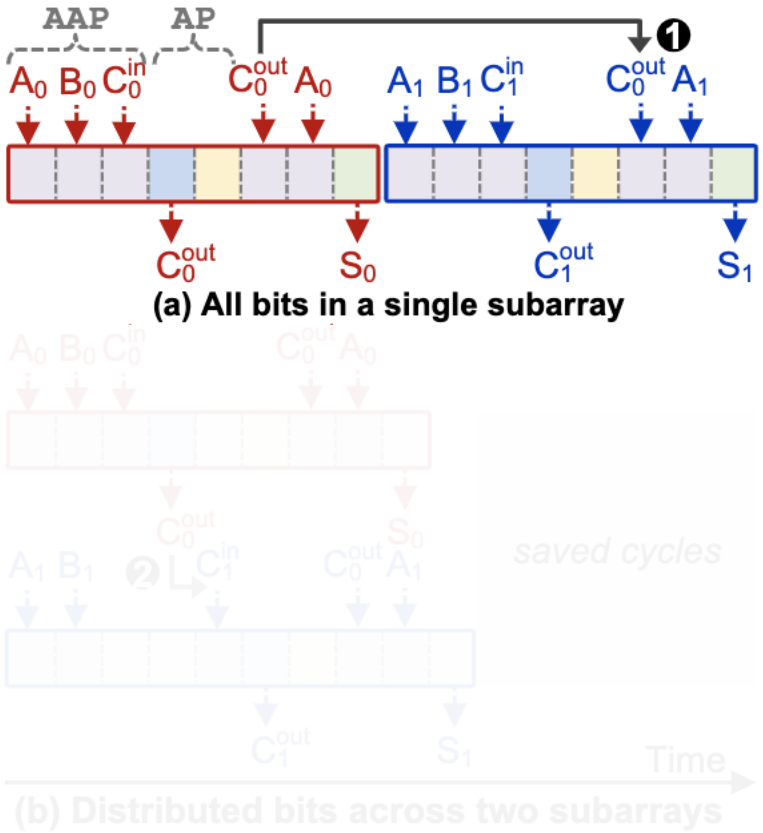
2 Throughput-Oriented Execution
- failing to reduce the latency of a *single* PUD operations
- opportunity 2: use SALP to concurrently execute PUD primitives

3 High-Precision Computation
- due to the linear/quadratically scaling nature of bit-serial algorithms
- opportunity 3: use alternative data representation for high-precision PUD operations

Limitations of PUD Systems:
Throughput-Oriented Execution

PUD architectures favor a throughput-oriented execution, as DRAM parallelism can partially hide the latency of DRAM ACTs



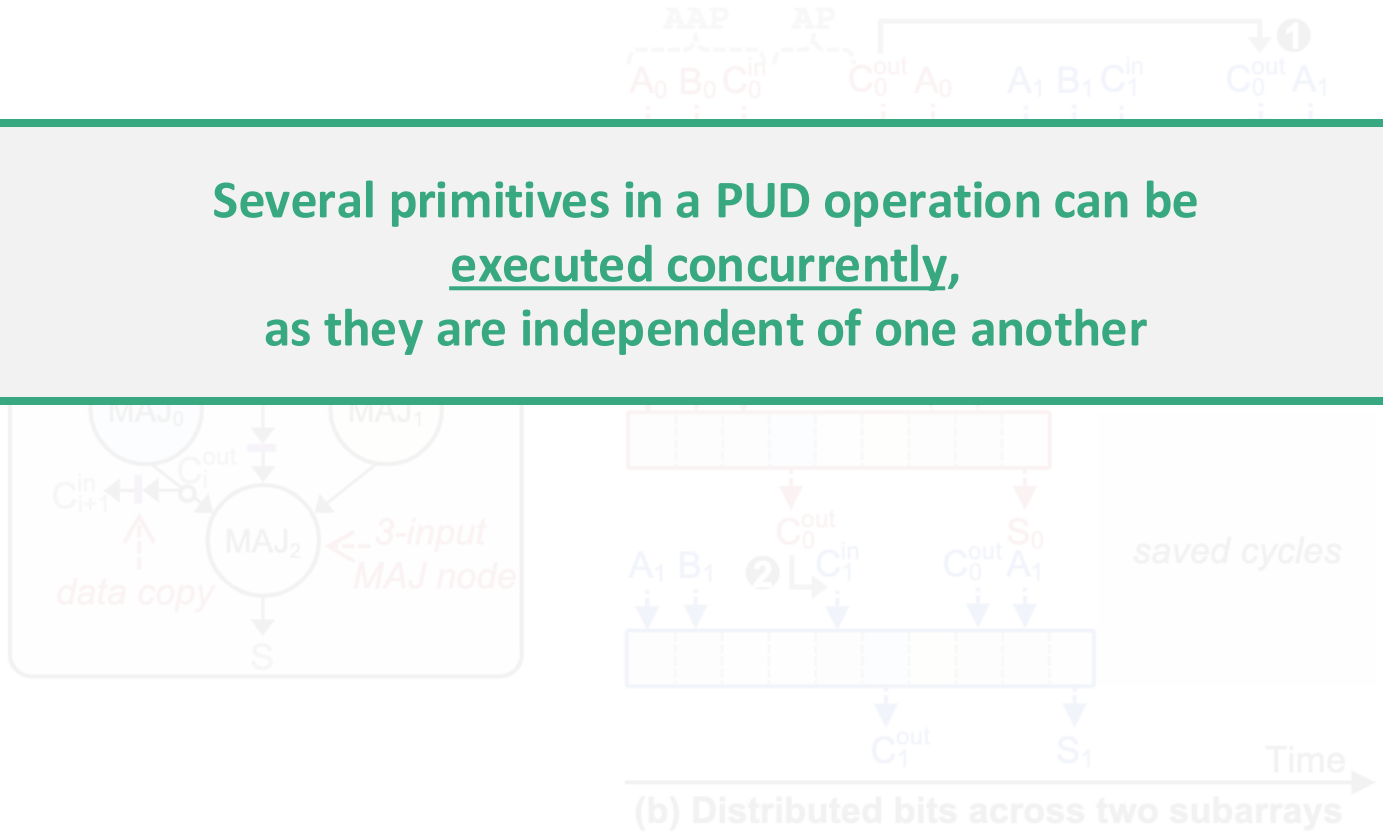


Limitations of PUD Systems:
Throughput-Oriented Execution

PUD architectures favor a throughput-oriented execution, as DRAM parallelism can partially hide the latency of DRAM ACTs

Key Idea

Several primitives in a PUD operation can be executed concurrently, as they are independent of one another



Limitations of PUD Systems:
Overview

PUD systems suffer from three sources of inefficiency due to the naive use of a bit-serial execution model

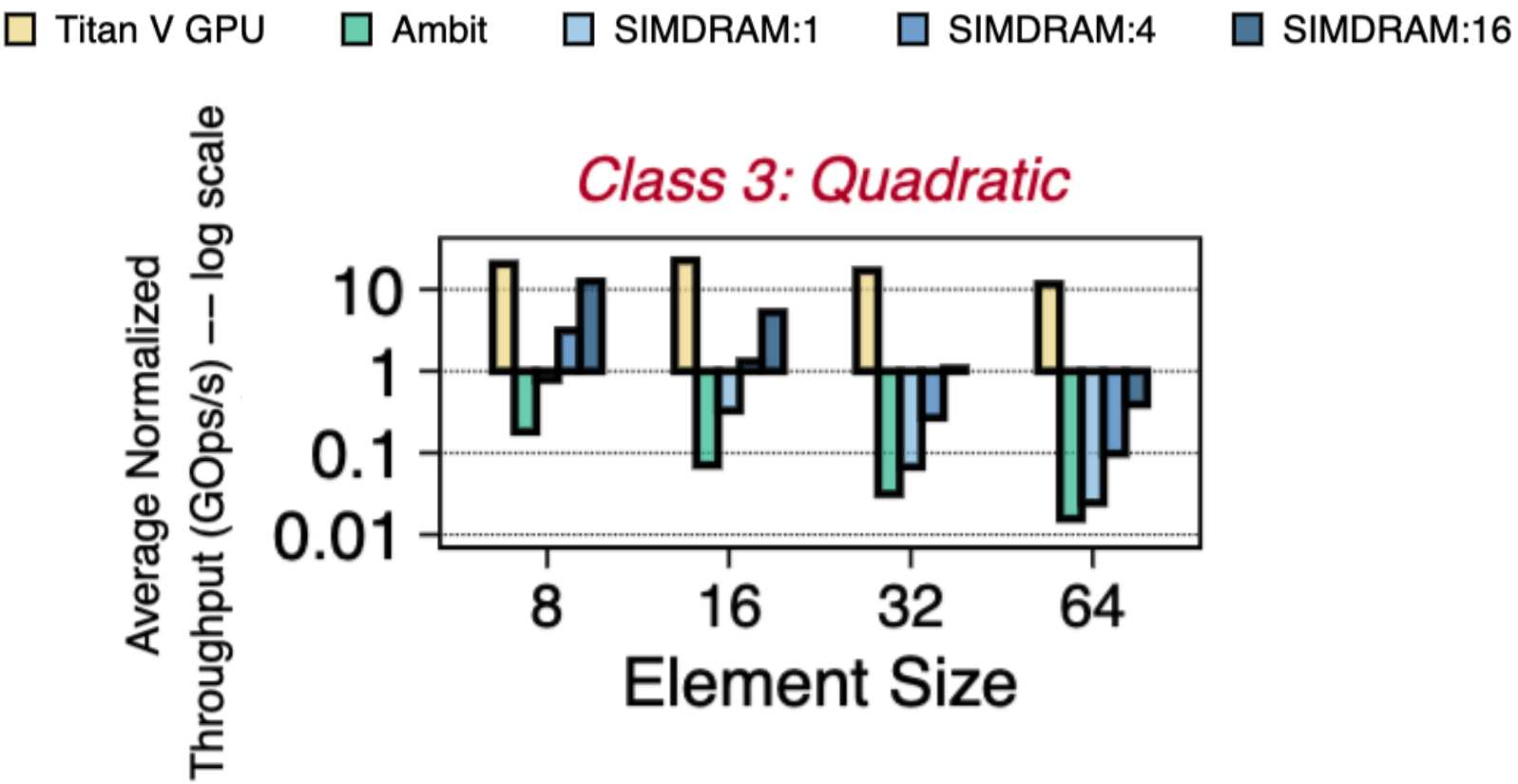
1 Static Data Representation
- leading to a subpar performance in the presence of *narrow values*
- opportunity 1: reduce the bit-precision of a PUD operation

2 Throughput-Oriented Execution
- failing to reduce the latency of a *single* PUD operations
- opportunity 2: use SALP to concurrently execute PUD primitives

3 High-Precision Computation
- due to the linear/quadratically scaling nature of bit-serial algorithms
- opportunity 3: use alternative data representation for high-precision PUD operations

Limitations of PUD Systems:
High-Precision Computation

For operations that require high precision,
PUD architectures suffer from high latency



Limitations of PUD Systems:
High-Precision Computation

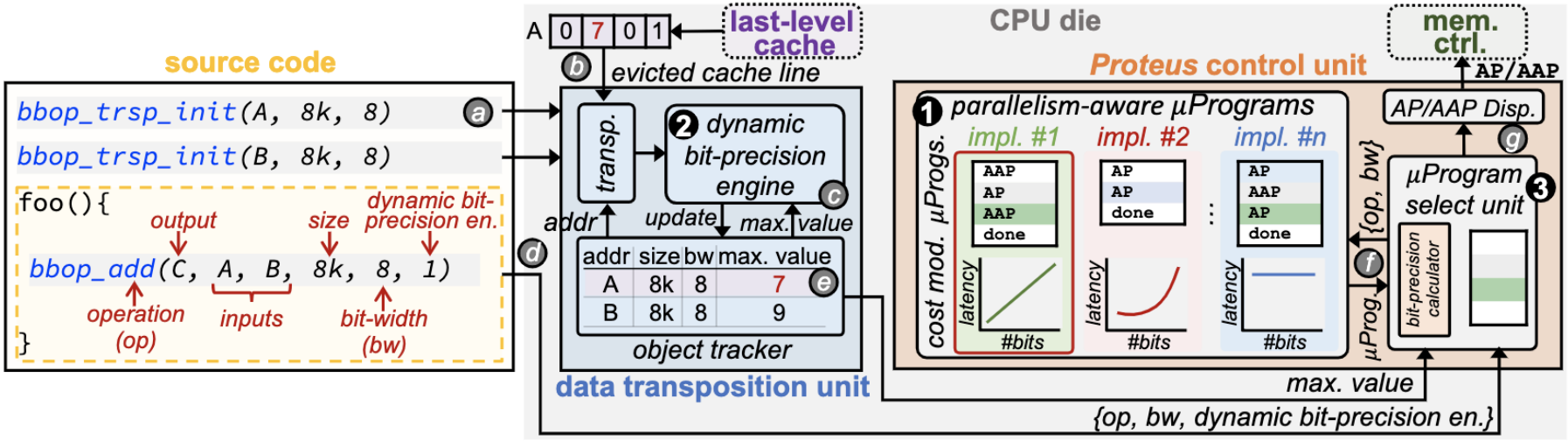
For operations that require high precision,
PUD architectures suffer from high latency

Key Idea

Use a positional number system representation, i.e., the redundant binary representation (RBR) for high-precision PUD operations:

(1) the operation no longer needs to propagate carry bits
(2) the operation latency is independent of the data precision

Proteus: Design Overview (I)



Proteus is composed of three main components:

1 Parallelism-Aware uProgram Library

hand-optimized implementations of different PUD operations with different performance vs. bit-precision trade-offs

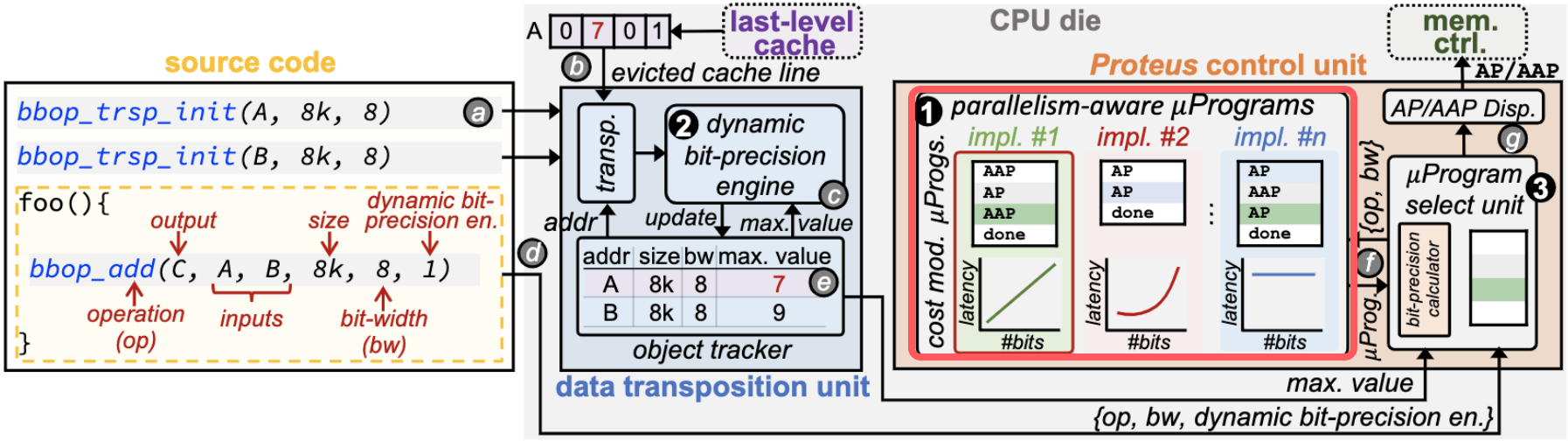
2 Dynamic Bit-Precision Engine

aims to identify the dynamic range of memory objects associated with a PUD operation

3 uProgram Select Unit

aims to identify the appropriate bit-precision based on the input operations of the target PUD operation

Proteus:
Design Overview (II)



***Proteus* is composed of three main components:**

1 Parallelism-Aware uProgram Library

hand-optimized implementations of different PUD operations with different performance vs. bit-precision trade-offs

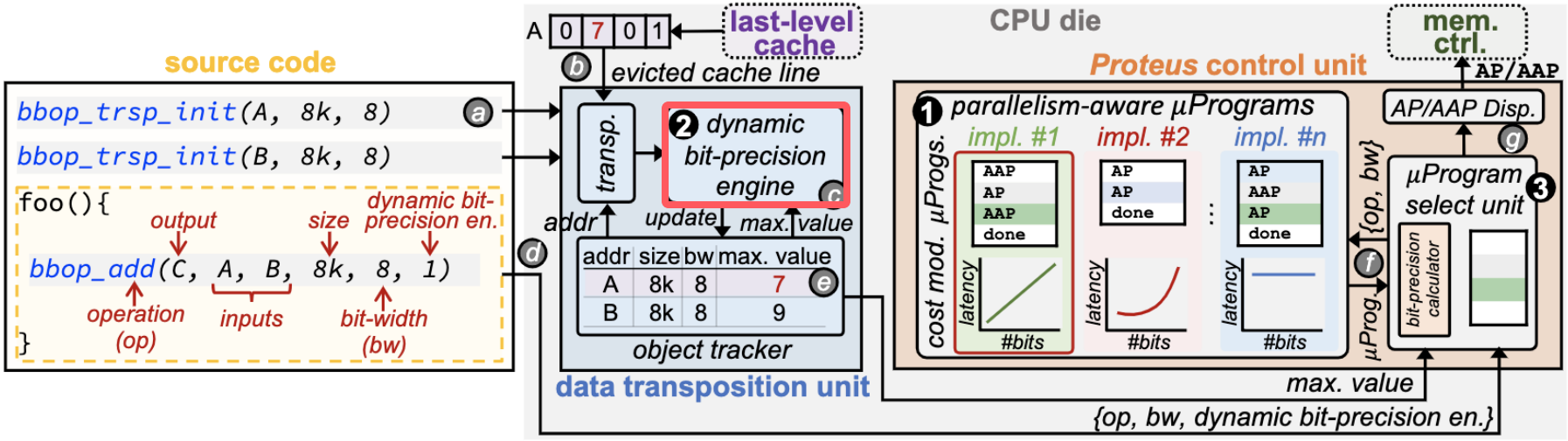
2 Dynamic Bit-Precision Engine

aims to identify the dynamic range of memory objects associated with a PUD operation

3 uProgram Select Unit

aims to identify the appropriate bit-precision based on the input operations of the target PUD operation

Proteus: Design Overview (III)



Proteus is composed of three main components:

1 Parallelism-Aware uProgram Library

hand-optimized implementations of different PUD operations with different performance vs. bit-precision trade-offs

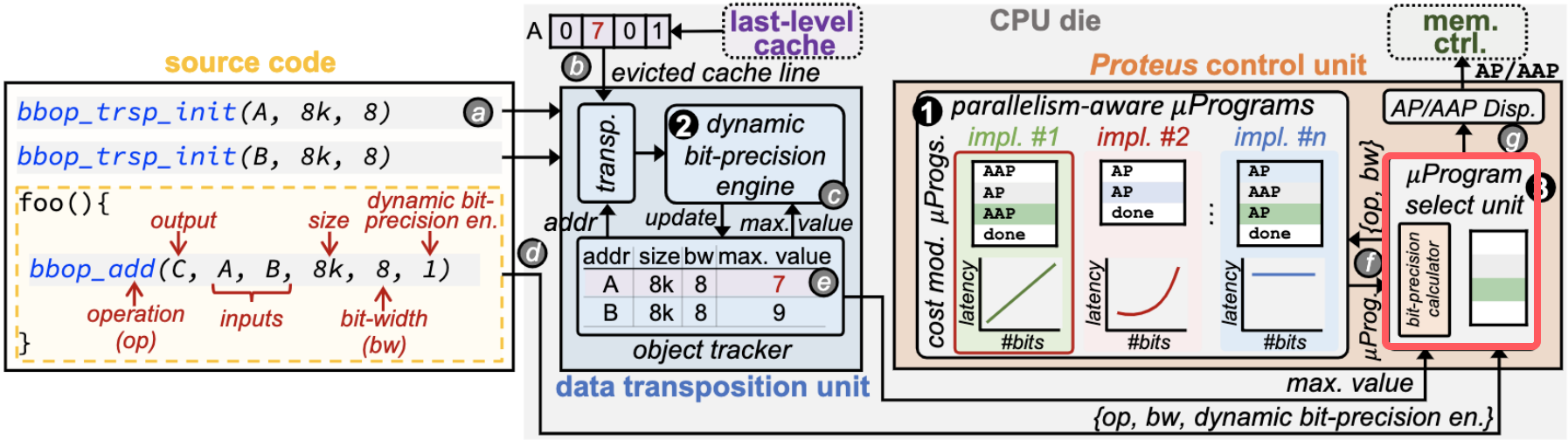
2 Dynamic Bit-Precision Engine

aims to identify the dynamic range of memory objects associated with a PUD operation

3 uProgram Select Unit

aims to identify the appropriate bit-precision based on the input operations of the target PUD operation

Proteus:
Design Overview (IV)



***Proteus* is composed of three main components:**

1 Parallelism-Aware uProgram Library

hand-optimized implementations of different PUD operations with different performance vs. bit-precision trade-offs

2 Dynamic Bit-Precision Engine

aims to identify the dynamic range of memory objects associated with a PUD operation

3 uProgram Select Unit

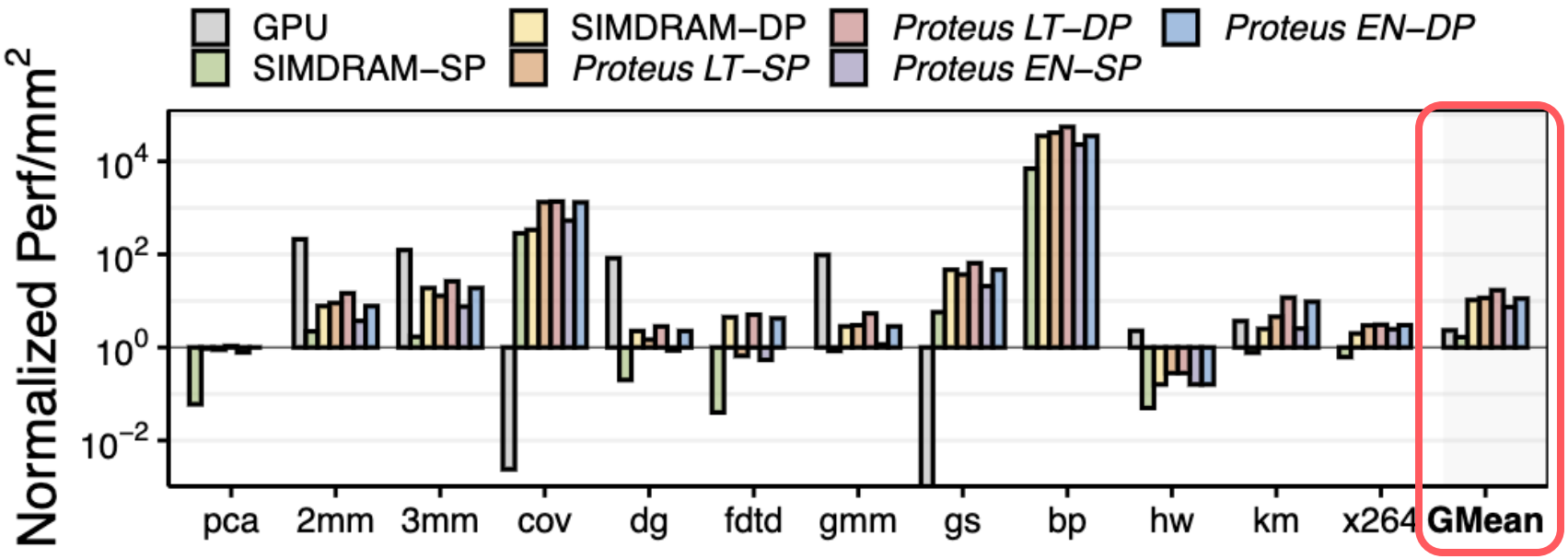
aims to identify the appropriate bit-precision based on the input operations of the target PUD operation

Evaluation:
Methodology Overview

- **Evaluation Setup**
  - **CPU:** Intel Skylake CPU
  - **GPU:** NVIDIA A100 GPU (with and without Tensor Cores)
  - **PUD:** SIMDRAM [Oliveira+, 2021]

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

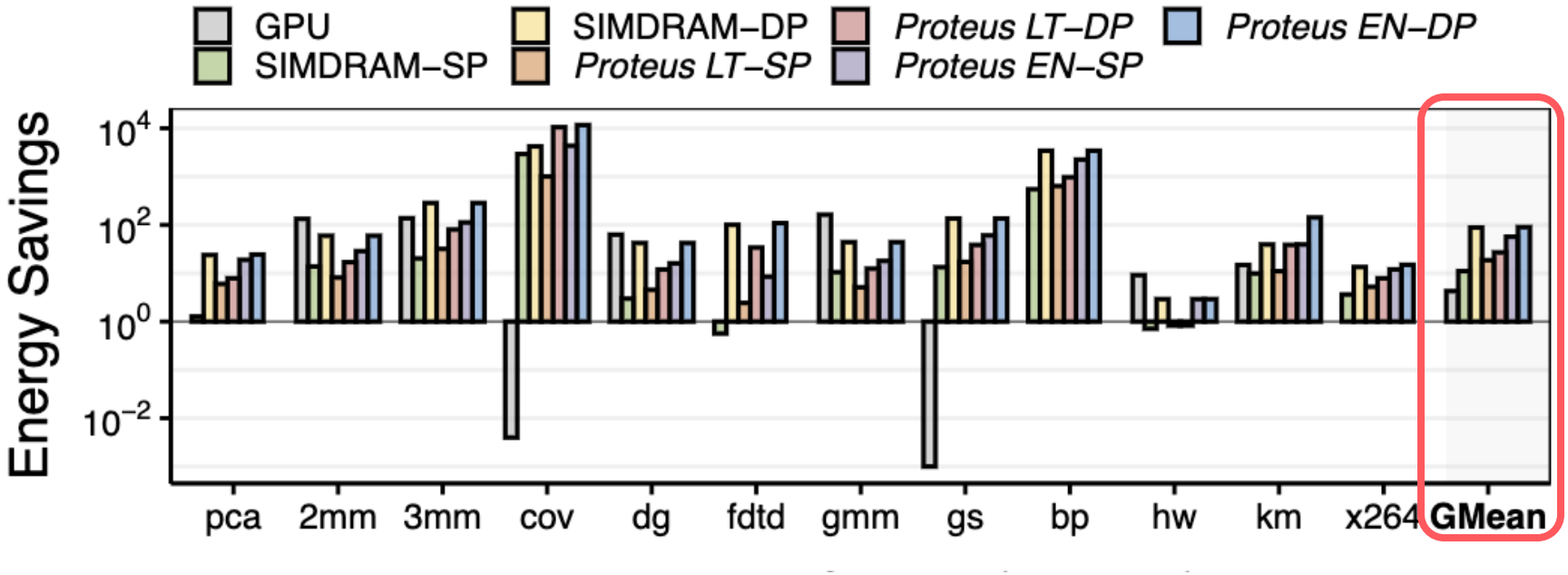
Evaluation:
Performance Analysis



Takeaway

Proteus **significantly improves performance/mm² compared to CPU (17x), GPU (7.3x), and SIMDRAM (10.2x)**

Evaluation:
Energy Analysis



Takeaway

Proteus **significantly improves energy-efficiency compared to CPU (90.3x), GPU (21x), and SIMDRAM (8.1x)**

More on Proteus

- Geraldo F. Oliveira, Mayank Kabra, Yuxin Guo, Kangqi Chen, A. Giray Yağlıkçı, Melina Soysal, Mohammad Sadrosadati, Joaquin Olivares Bueno, Saugata Ghose, Juan Gómez-Luna, Onur Mutlu "***Proteus*: Achieving High-Performance Processing-Using-DRAM via Dynamic Precision Bit-Serial Arithmetic"** *arXiv, 2025.*

Proteus: Achieving High-Performance Processing-Using-DRAM via Dynamic Precision Bit-Serial Arithmetic

Geraldo F. Oliveira[†] Mayank Kabra[†] Yuxin Guo[‡] Kangqi Chen[†] A. Giray Yağlıkçı[†]
Melina Soysal[†] Mohammad Sadrosadati[†] Joaquin Olivares Bueno[⋆]
Saugata Ghose[▽] Juan Gómez-Luna[§] Onur Mutlu[†]

[†] *ETH Zürich* [†] *Cambridge University* [⋆] *Universidad de Córdoba*
[▽] *Univ. of Illinois Urbana-Champaign* [§] *NVIDIA Research*

https://arxiv.org/pdf/2501.17466

In-DRAM Lookup-Table Based Execution

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.
[Slides (pptx) (pdf)]
[Longer Lecture Slides (pptx) (pdf)]
[Lecture Video (26 minutes)]
[arXiv version]
[Source Code (Officially Artifact Evaluated with All Badges)]
Officially artifact evaluated as available, reusable and reproducible.

pLUTo: Enabling Massively Parallel Computation in DRAM via Lookup Tables

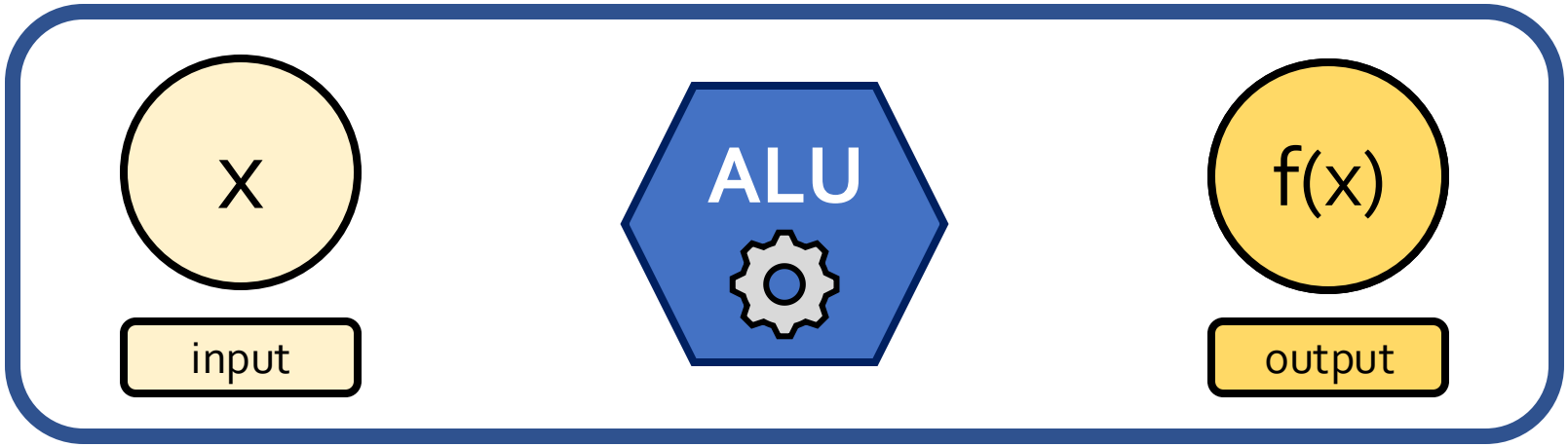
João Dinis Ferreira[§] Gabriel Falcao[†] Juan Gómez-Luna[§] Mohammed Alser[§]
Lois Orosa[§▽] Mohammad Sadrosadati[§] Jeremie S. Kim[§] Geraldo F. Oliveira[§]
Taha Shahroodi[‡] Anant Nori[⋆] Onur Mutlu[§]
[§]*ETH Zürich* [†]*IT, University of Coimbra* [▽]*Galicia Supercomputing Center* [‡]*TU Delft* [⋆]*Intel*

https://arxiv.org/pdf/2104.07699.pdf

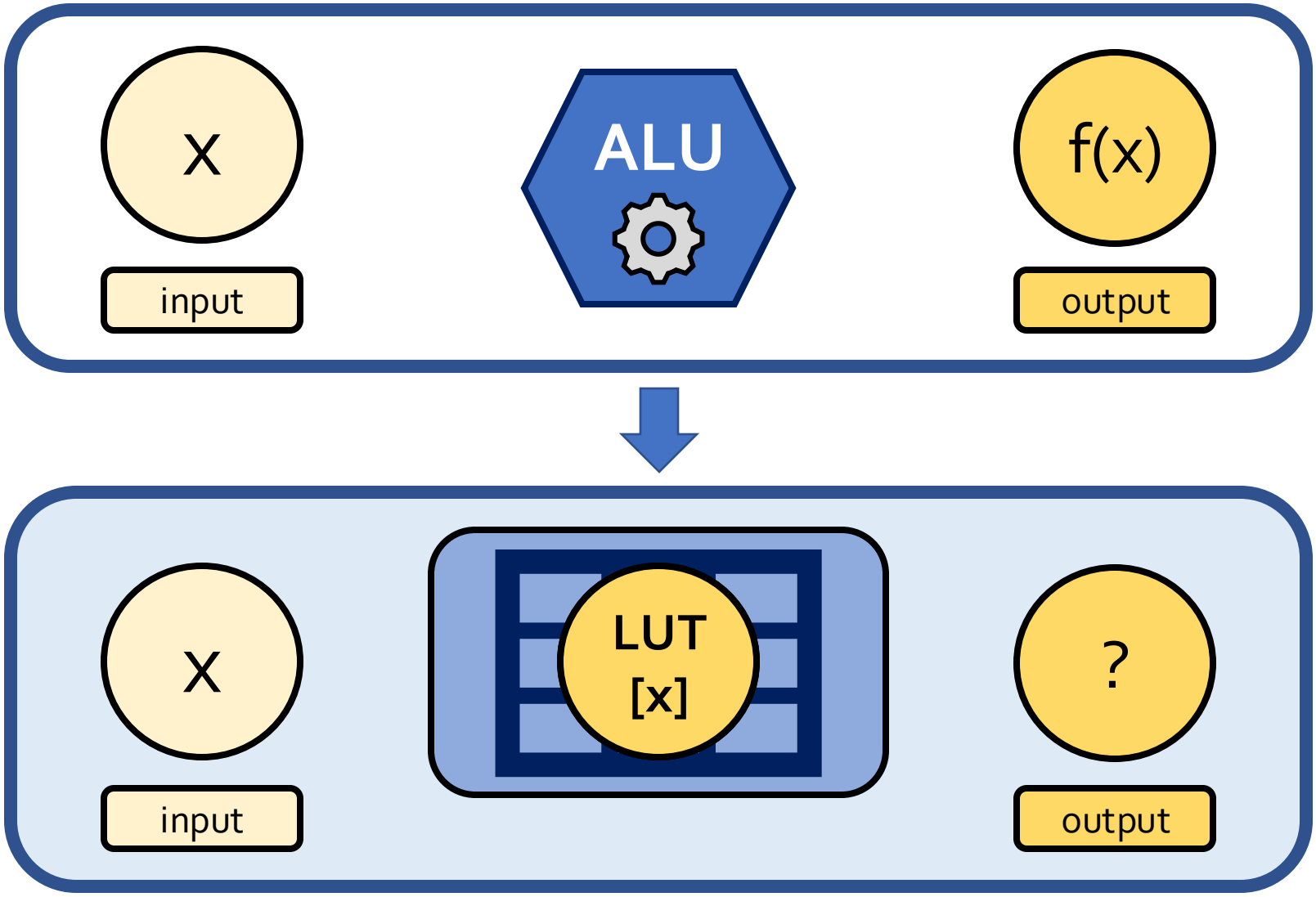
The Goal of pLUTo

Extend **Processing-using-DRAM to support the execution of** ***arbitrarily complex operations***

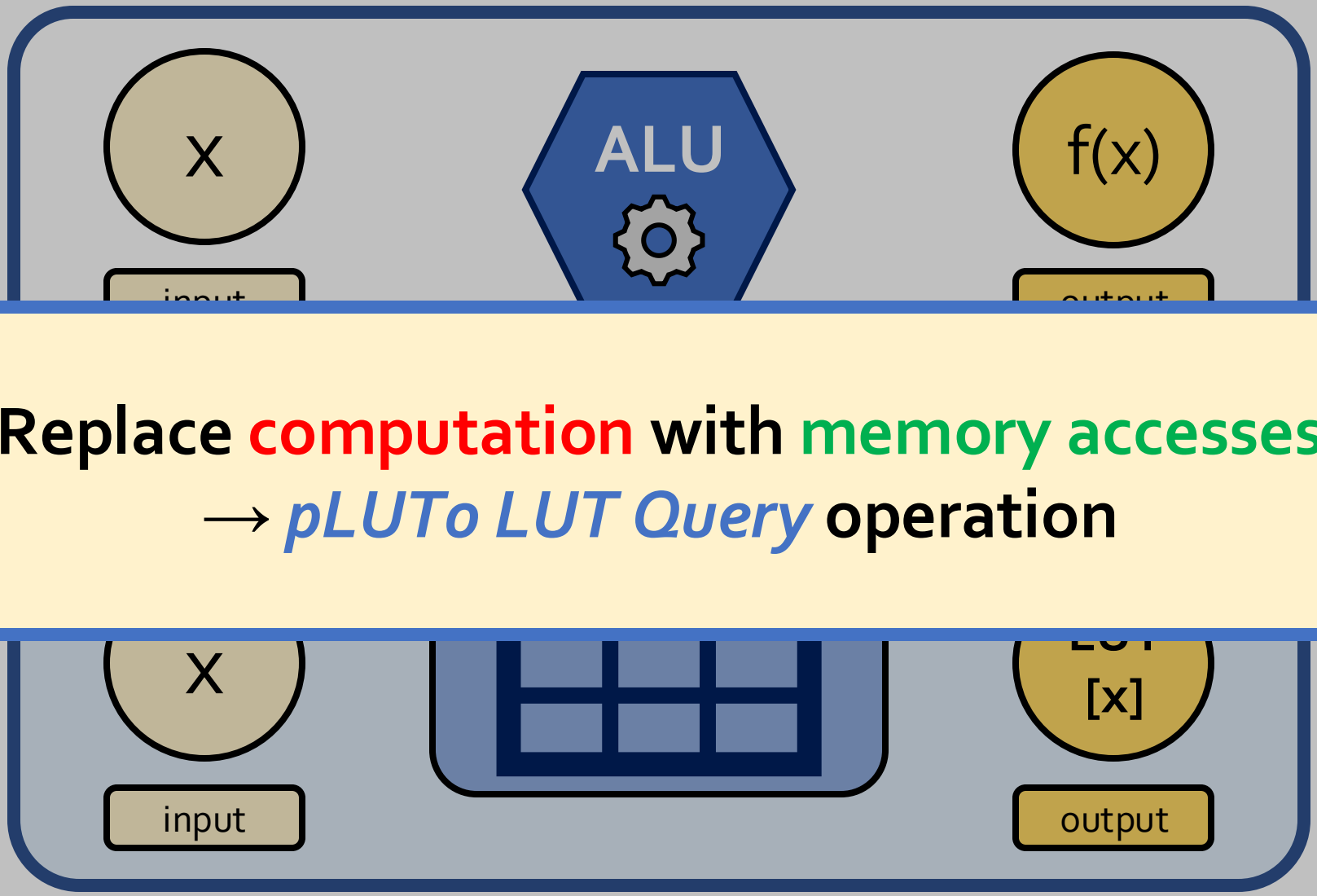
pLUTo: Key Idea



pLUTo: Key Idea



pLUTo: Key Idea



Replace computation with memory accesses
→ *pLUTo LUT Query* operation

In-DRAM pLUTo LUT Query: Setup

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

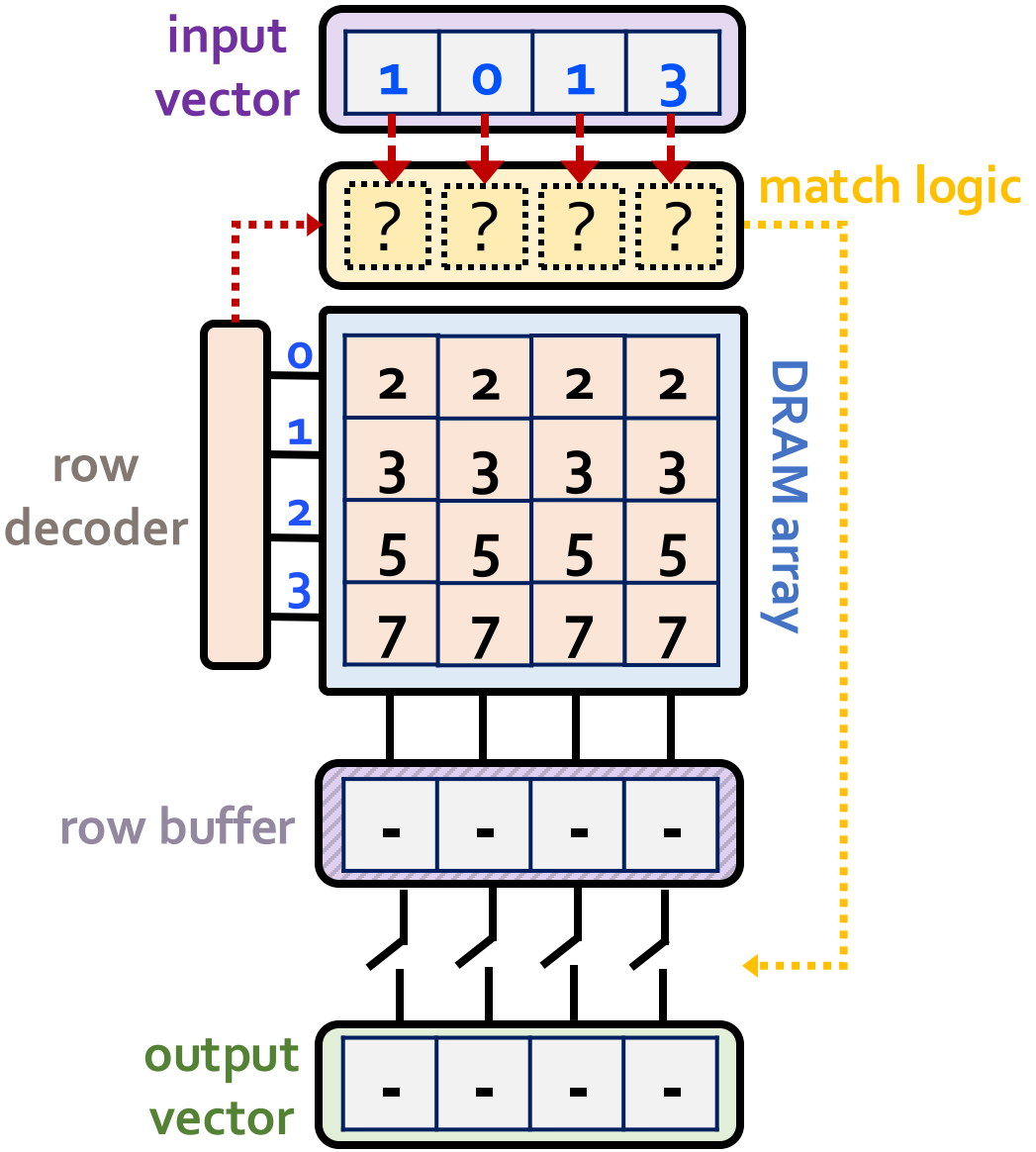
lookup table

1	0	1	3

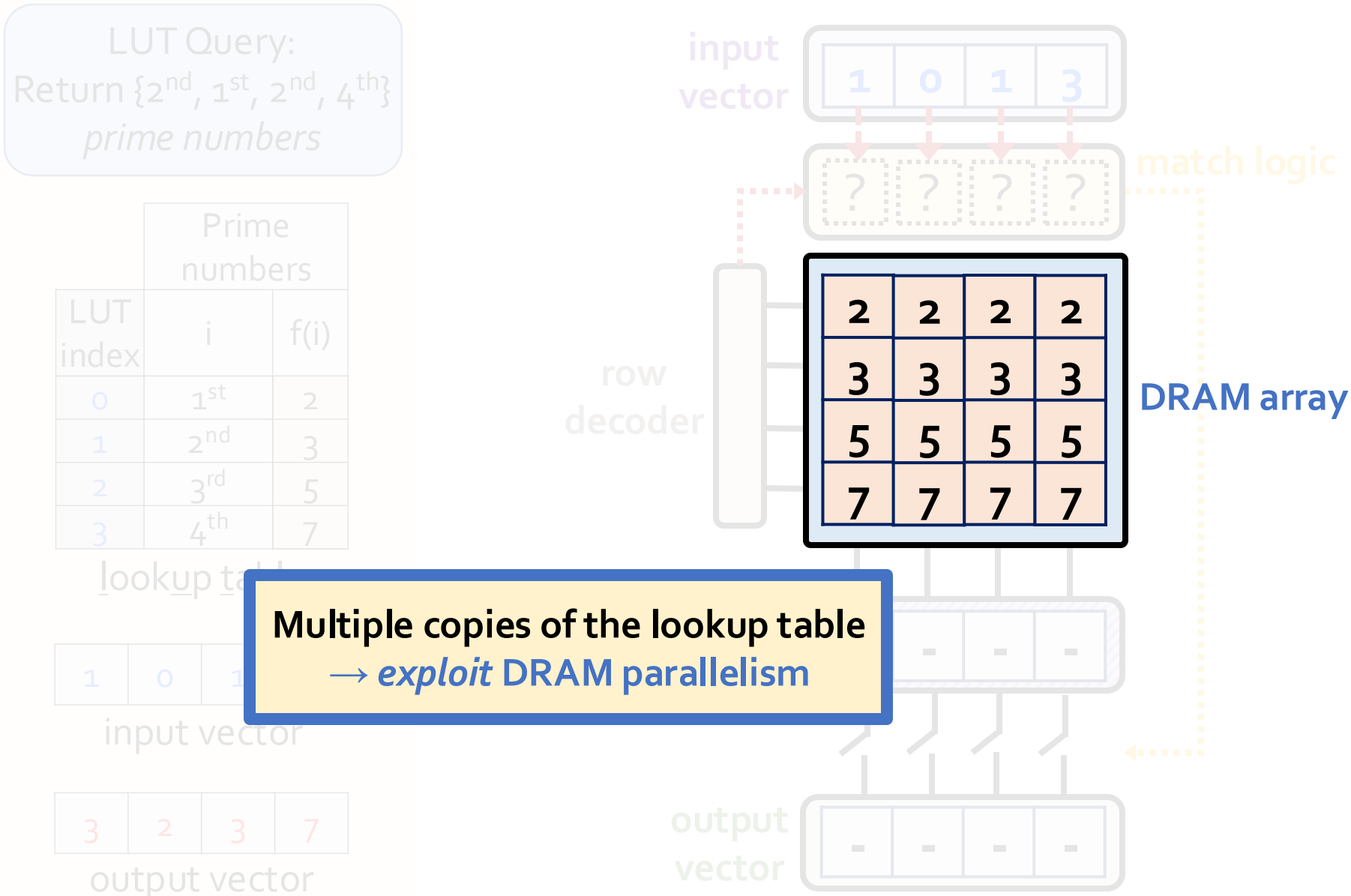
input vector

3	2	3	7

output vector



In-DRAM pLUTo LUT Query: Setup



Multiple copies of the lookup table
→ ***exploit*** **DRAM parallelism**

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

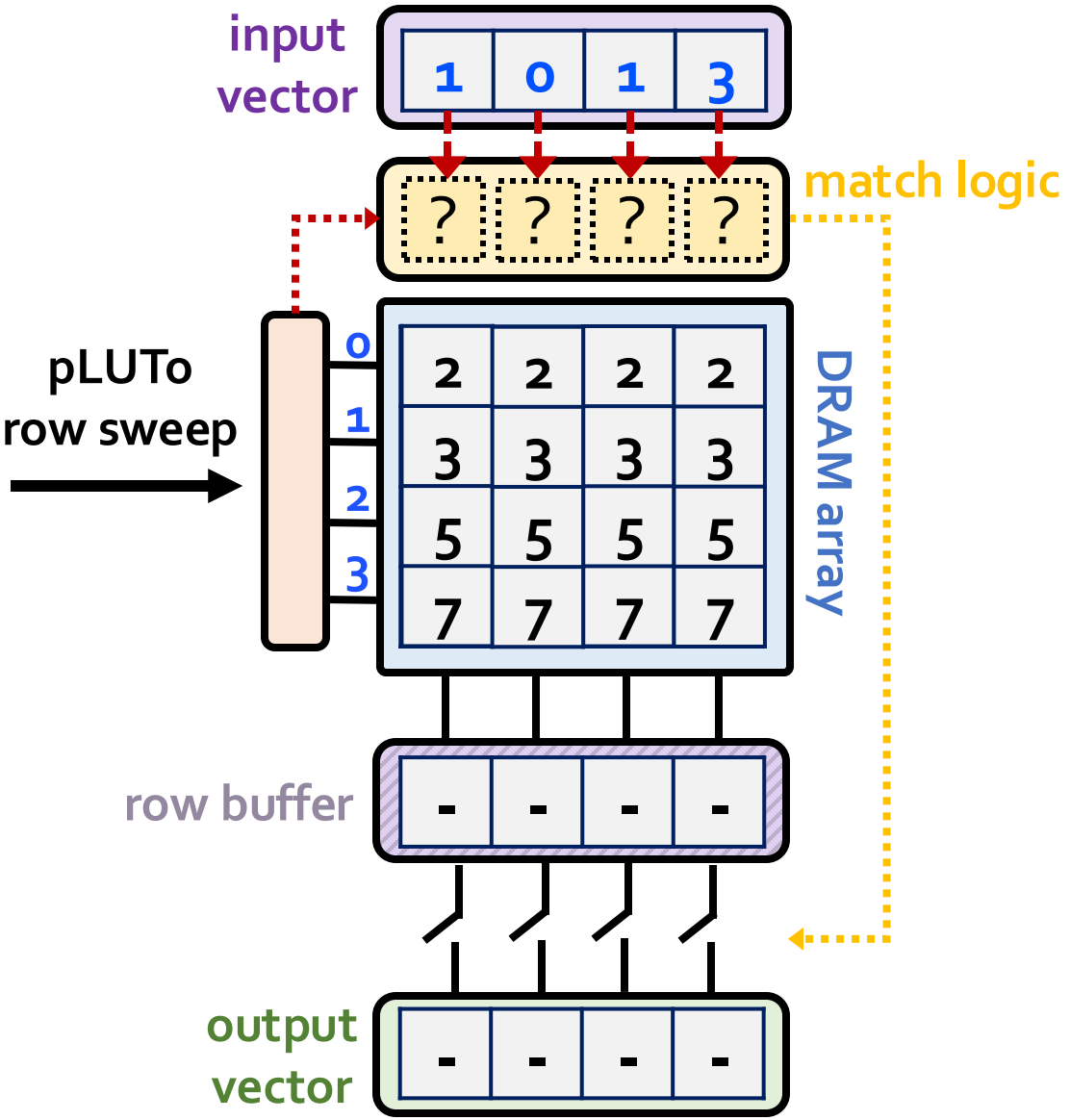
lookup table

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	f(i)
0	1st	2
1	2nd	3
2	3rd	5
3	4th	7

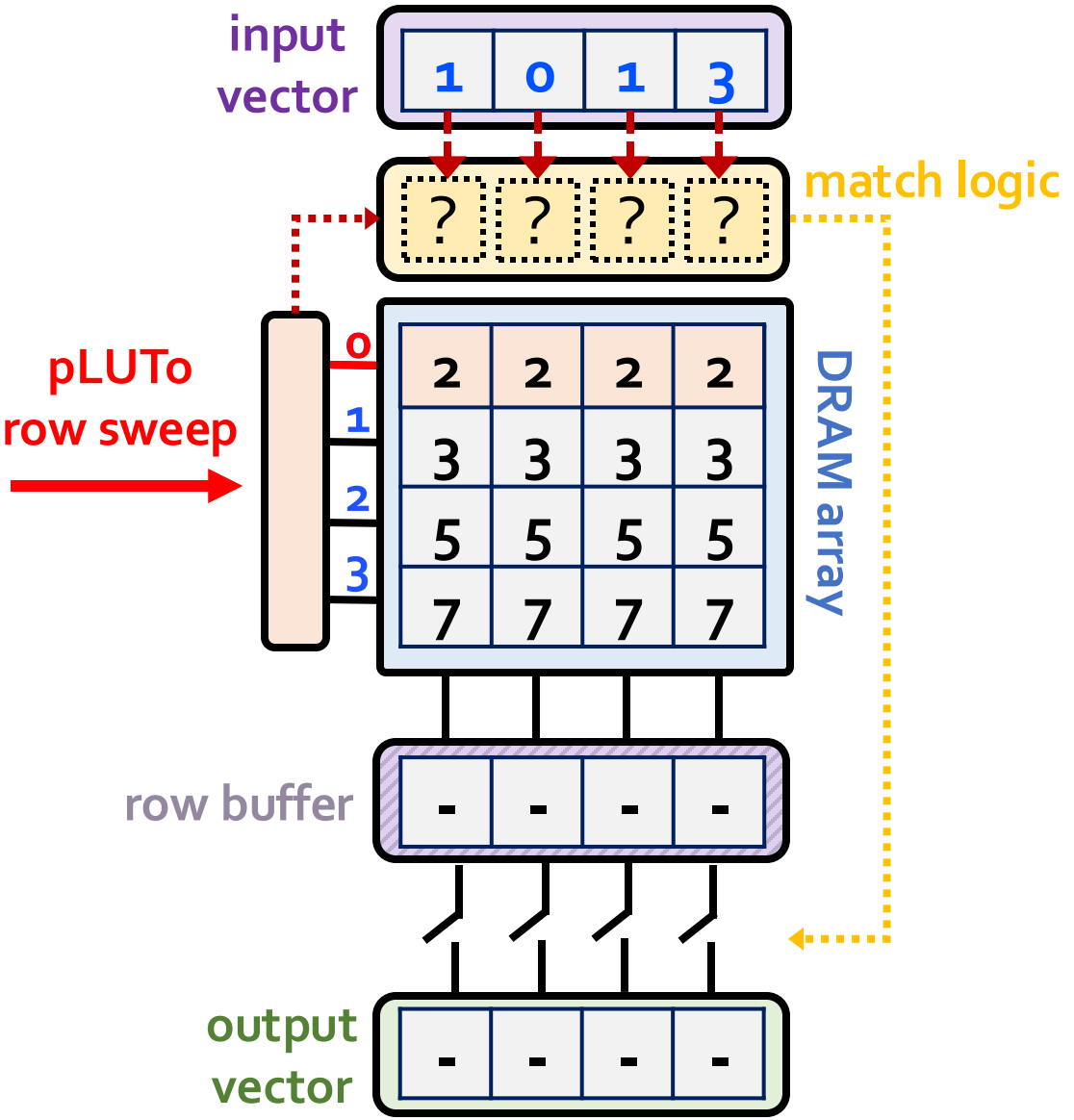
lookup table

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

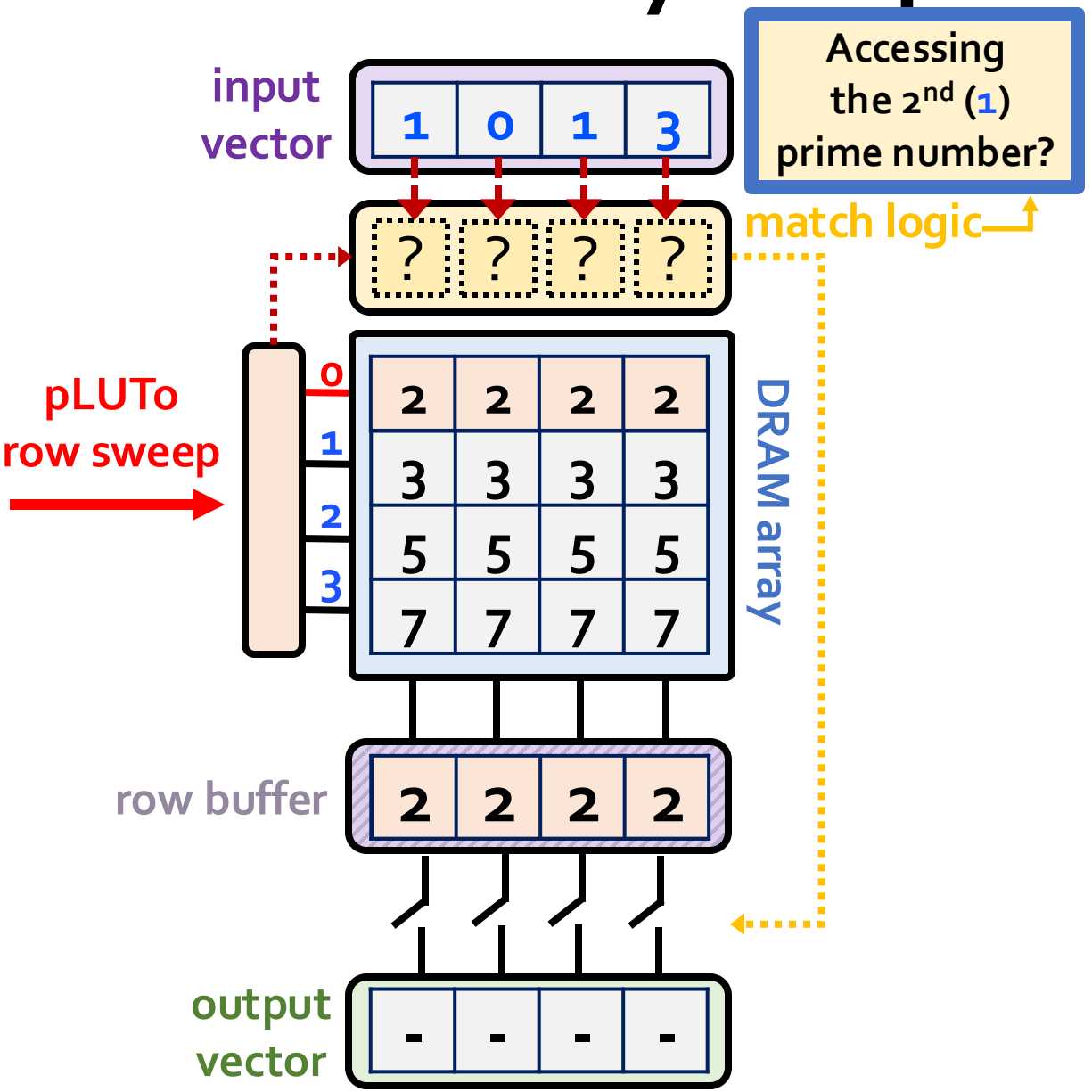
lookup table

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

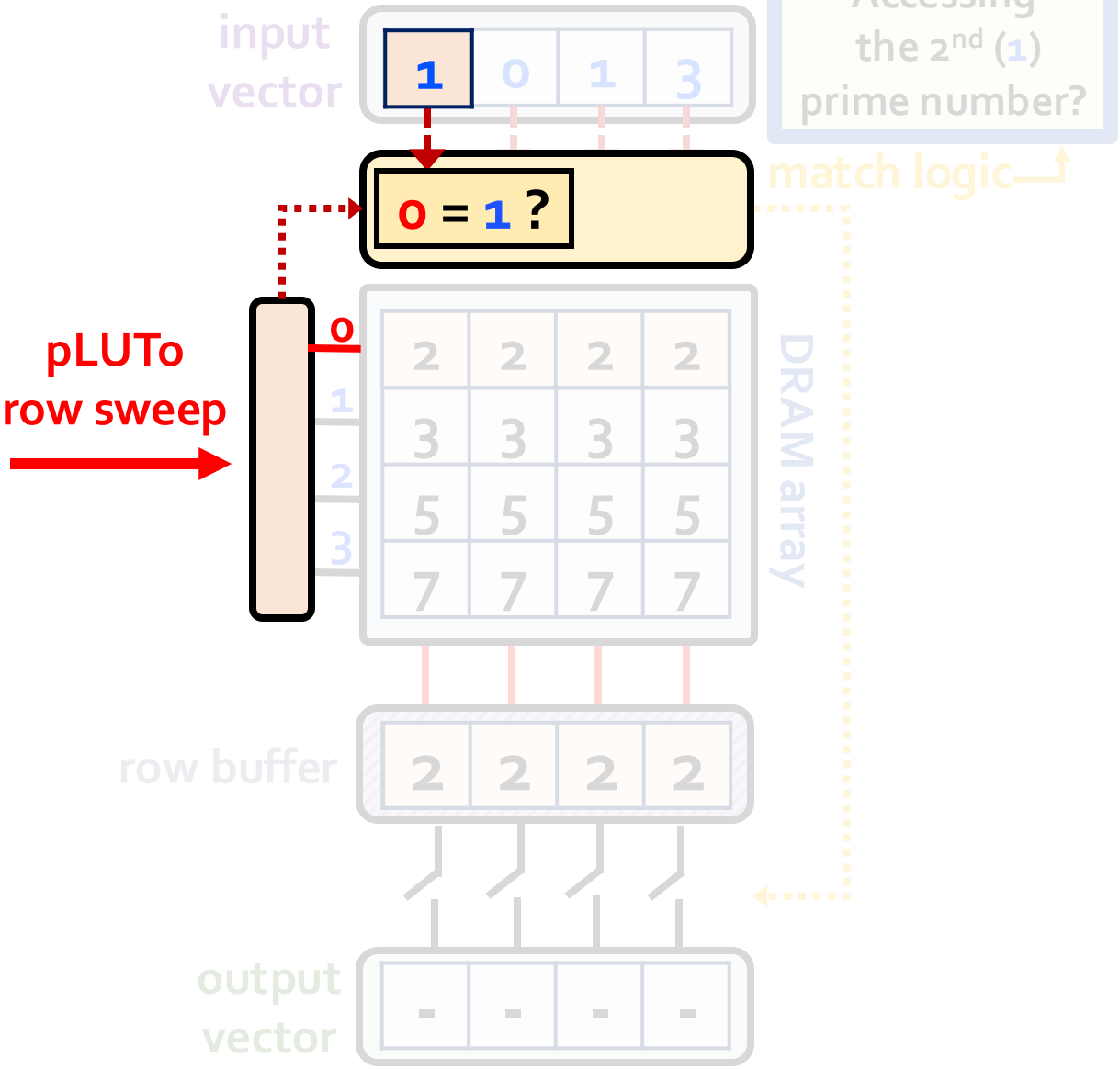
lookup table

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

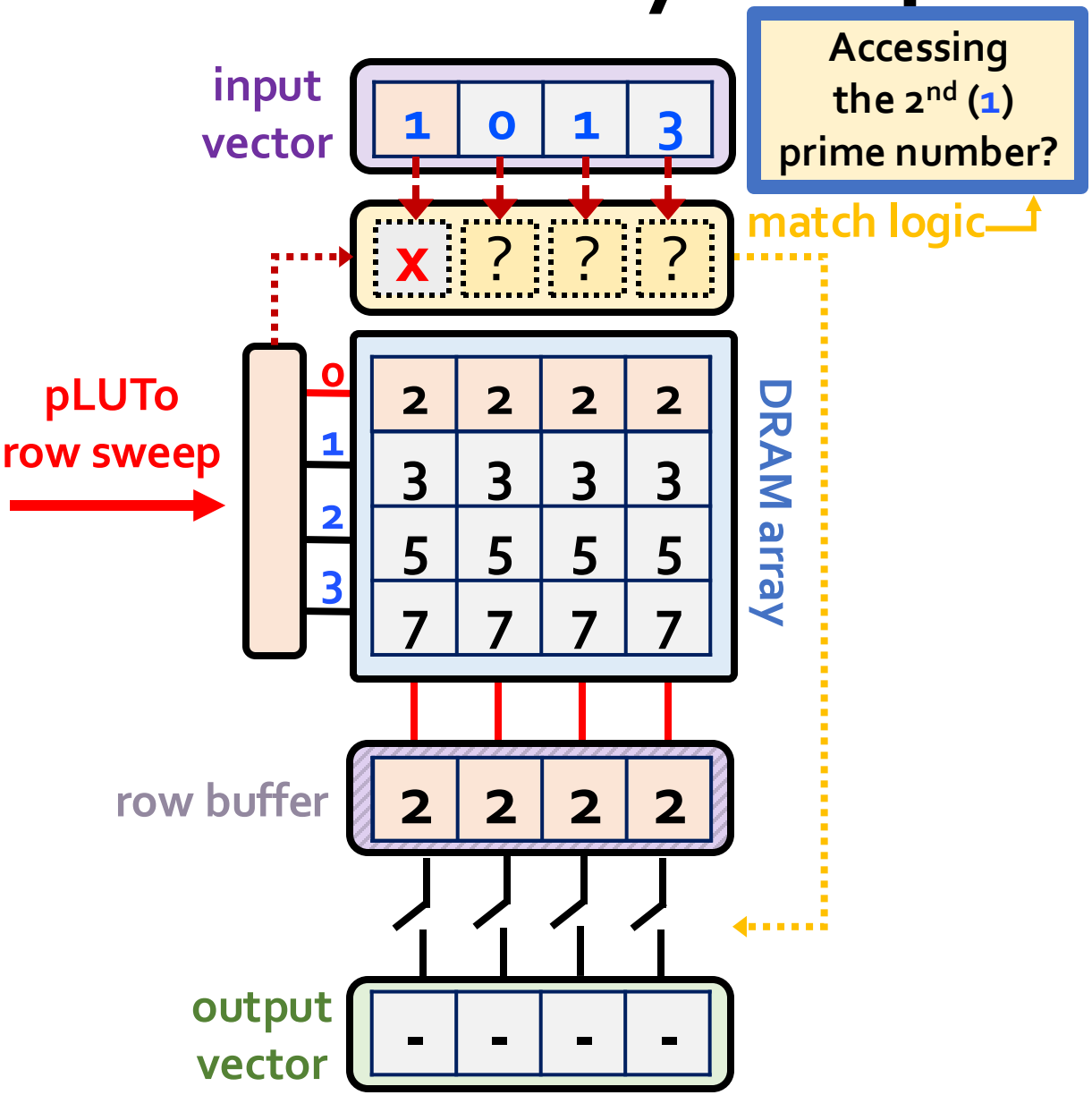
lookup table

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

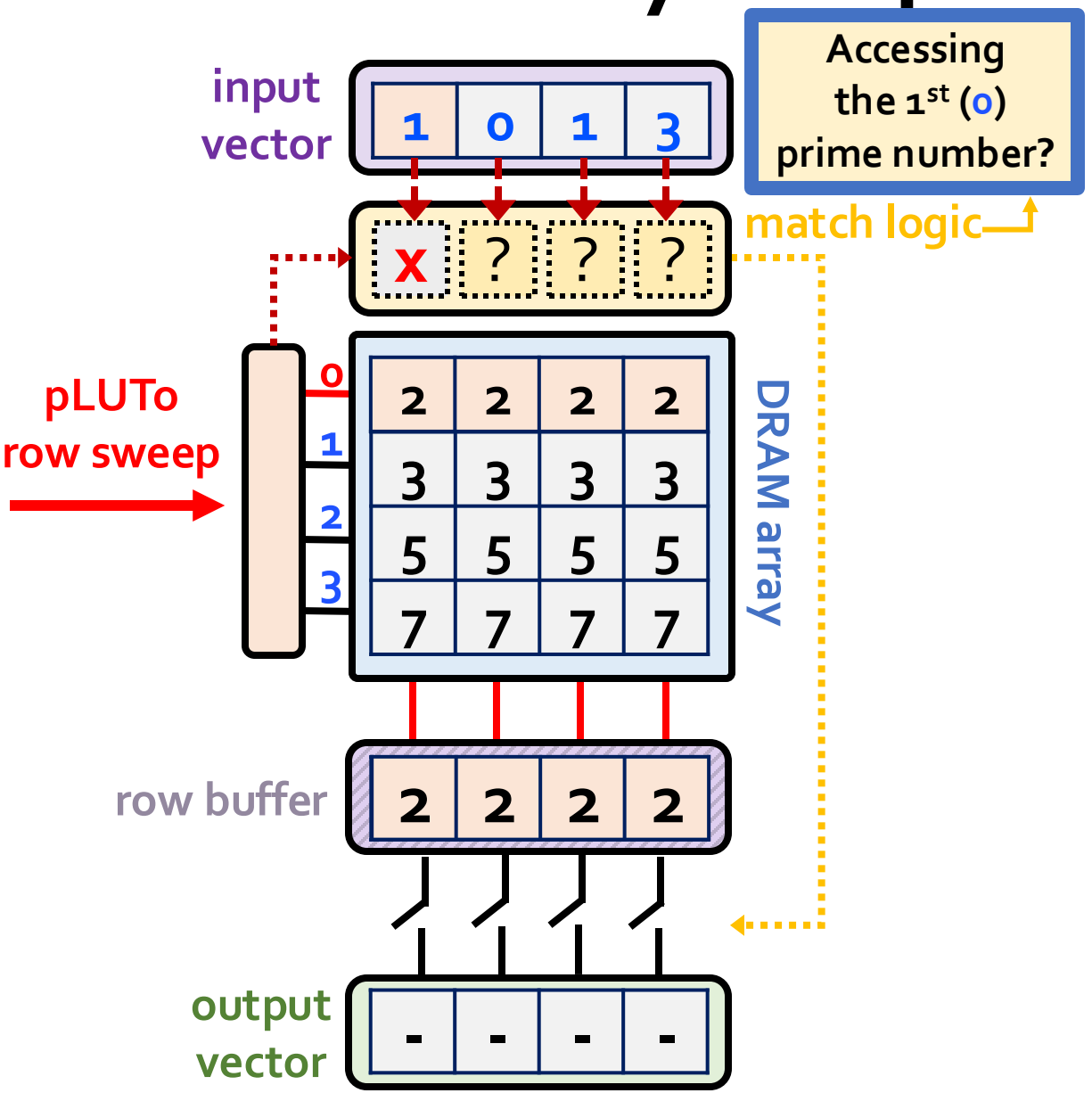
lookup table

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

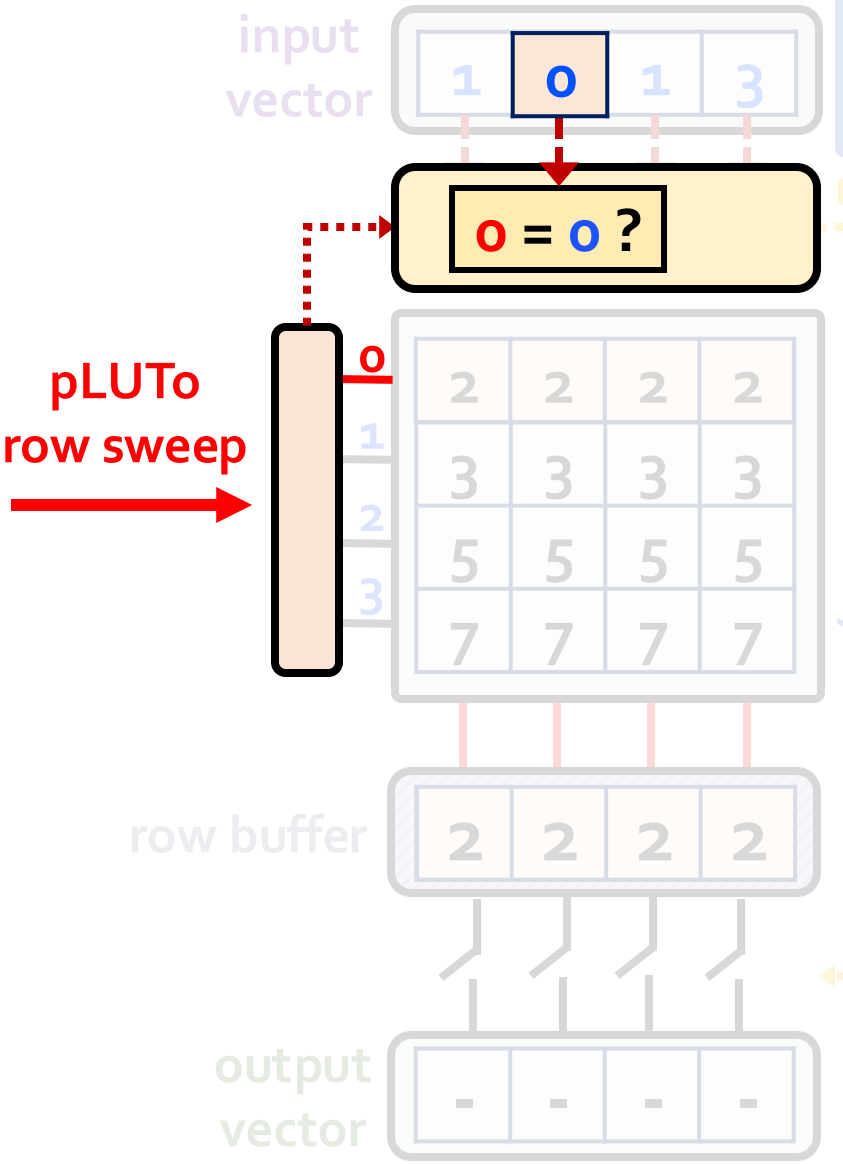
lookup table

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

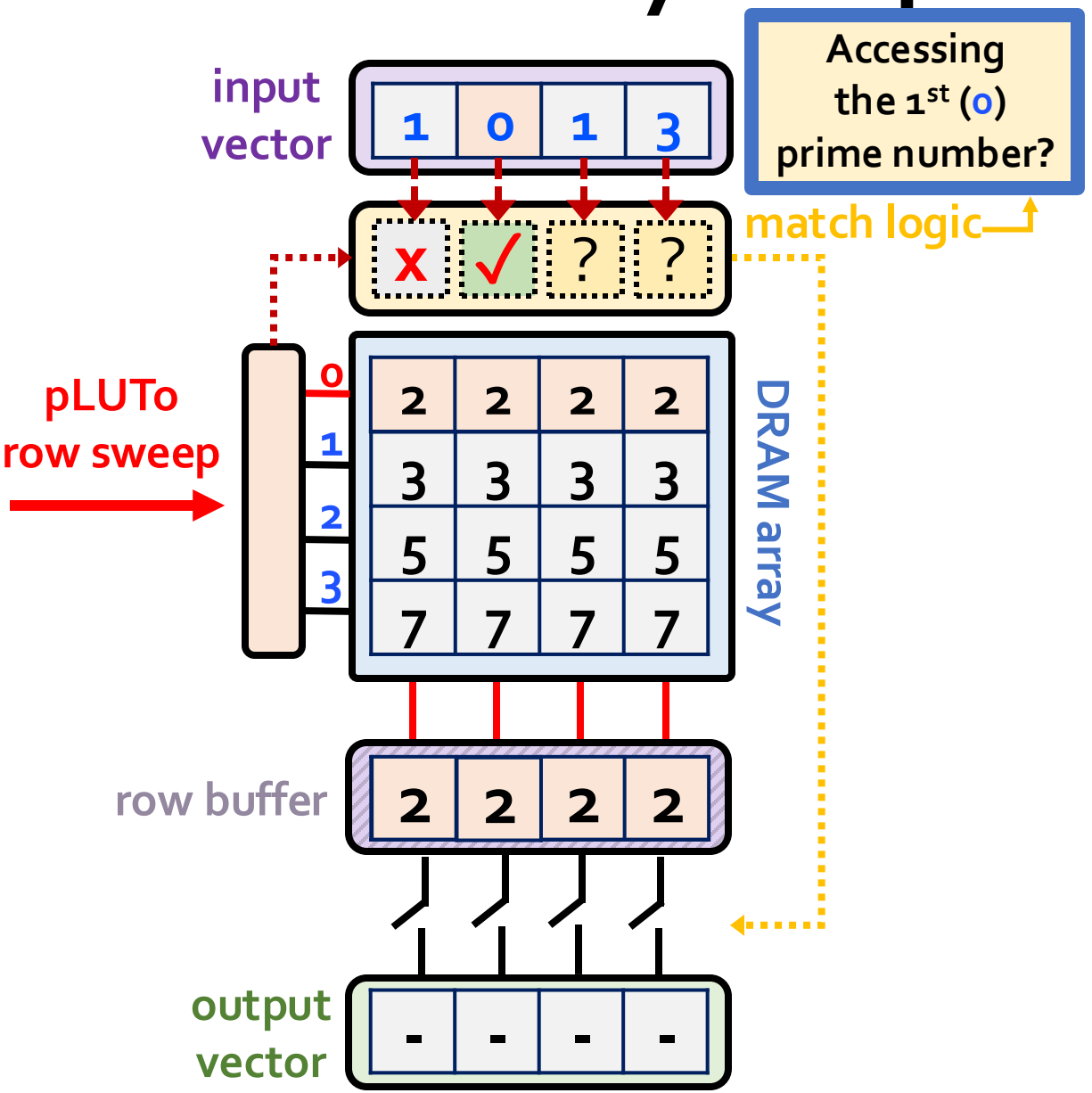
lookup table

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

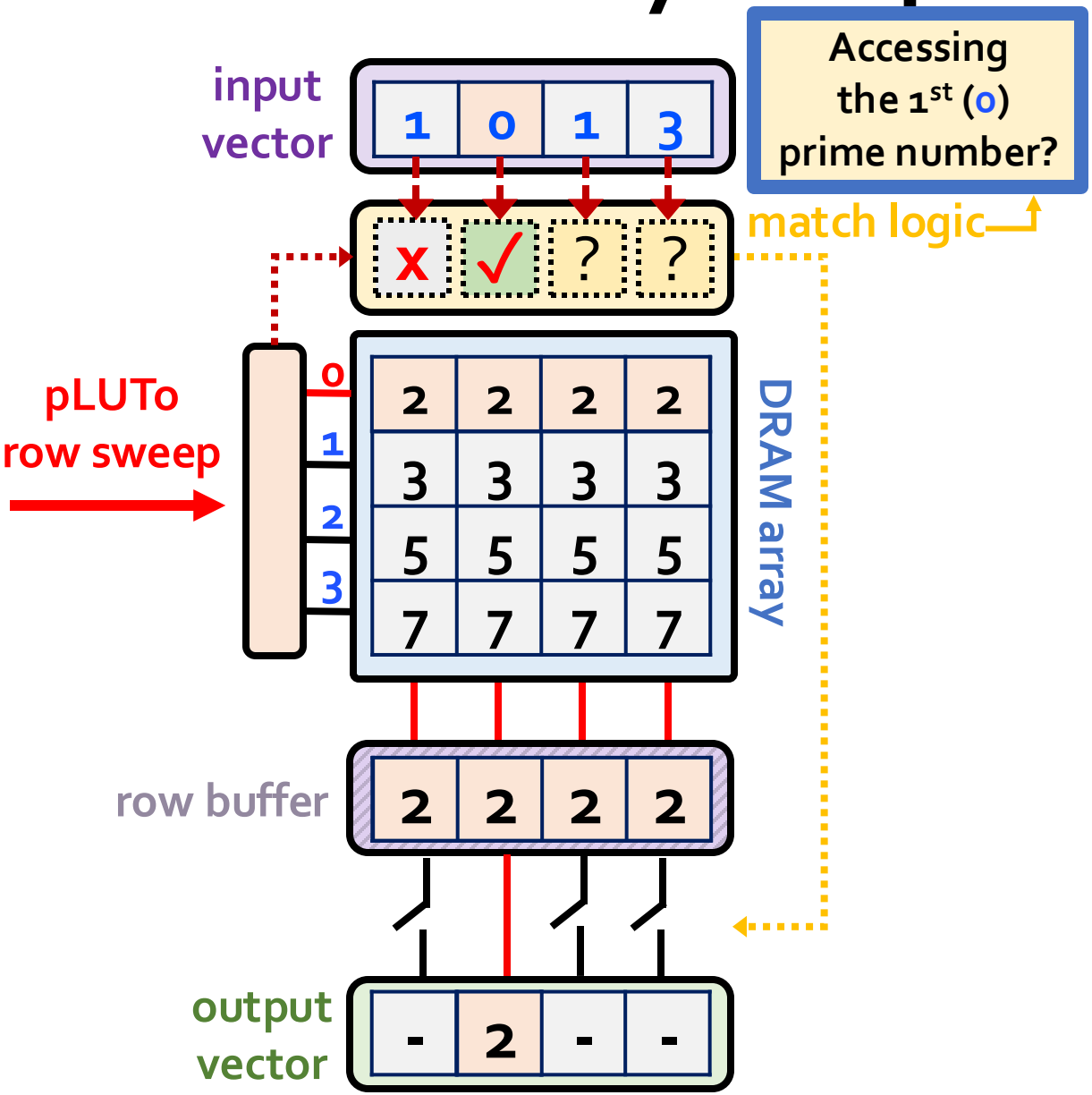
lookup table

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

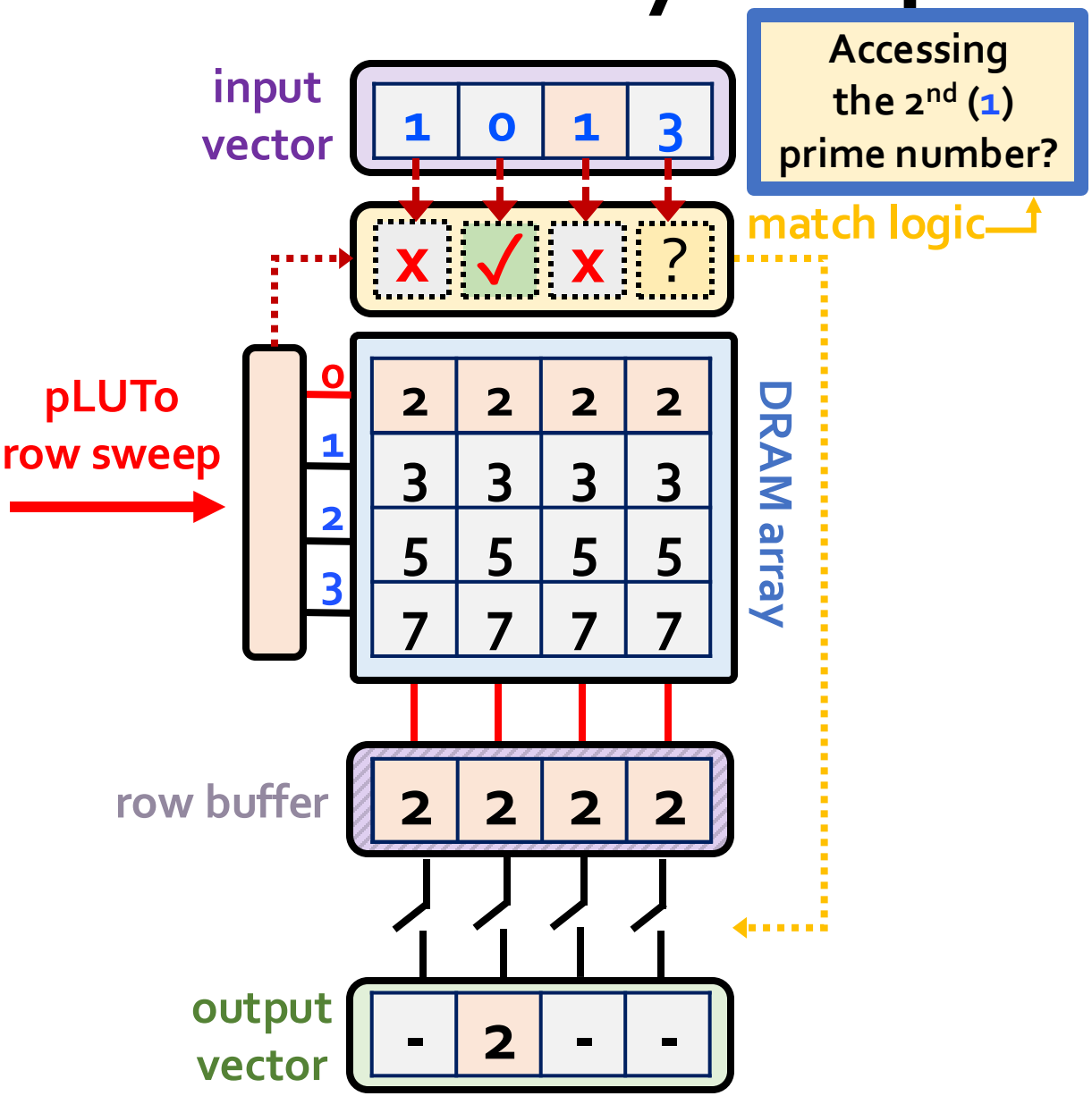
lookup table

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

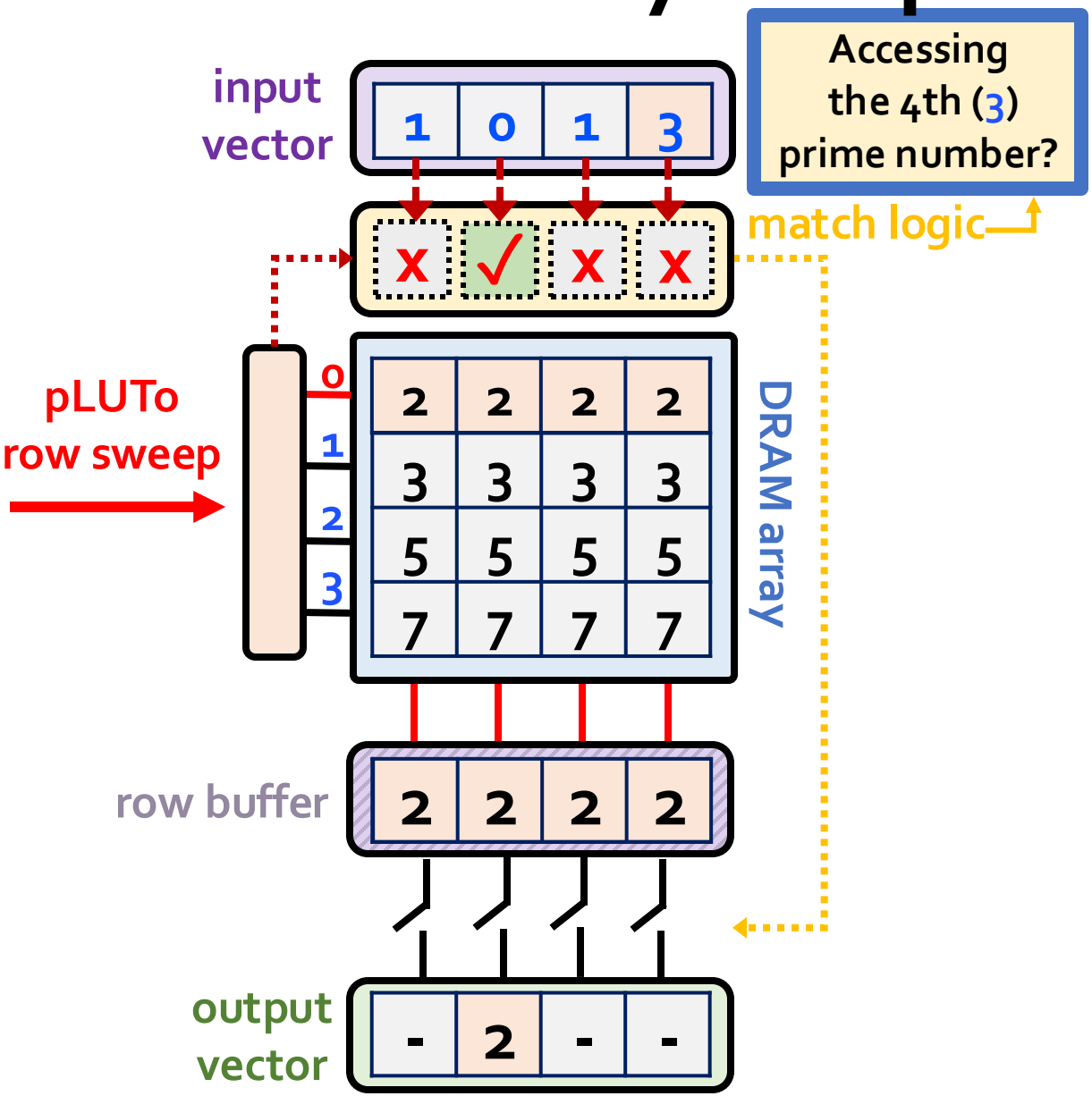
lookup table

1	0	1	3

input vector

3	2	3	7

output vector



In-DRAM pLUTo LUT Query: Step 2

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

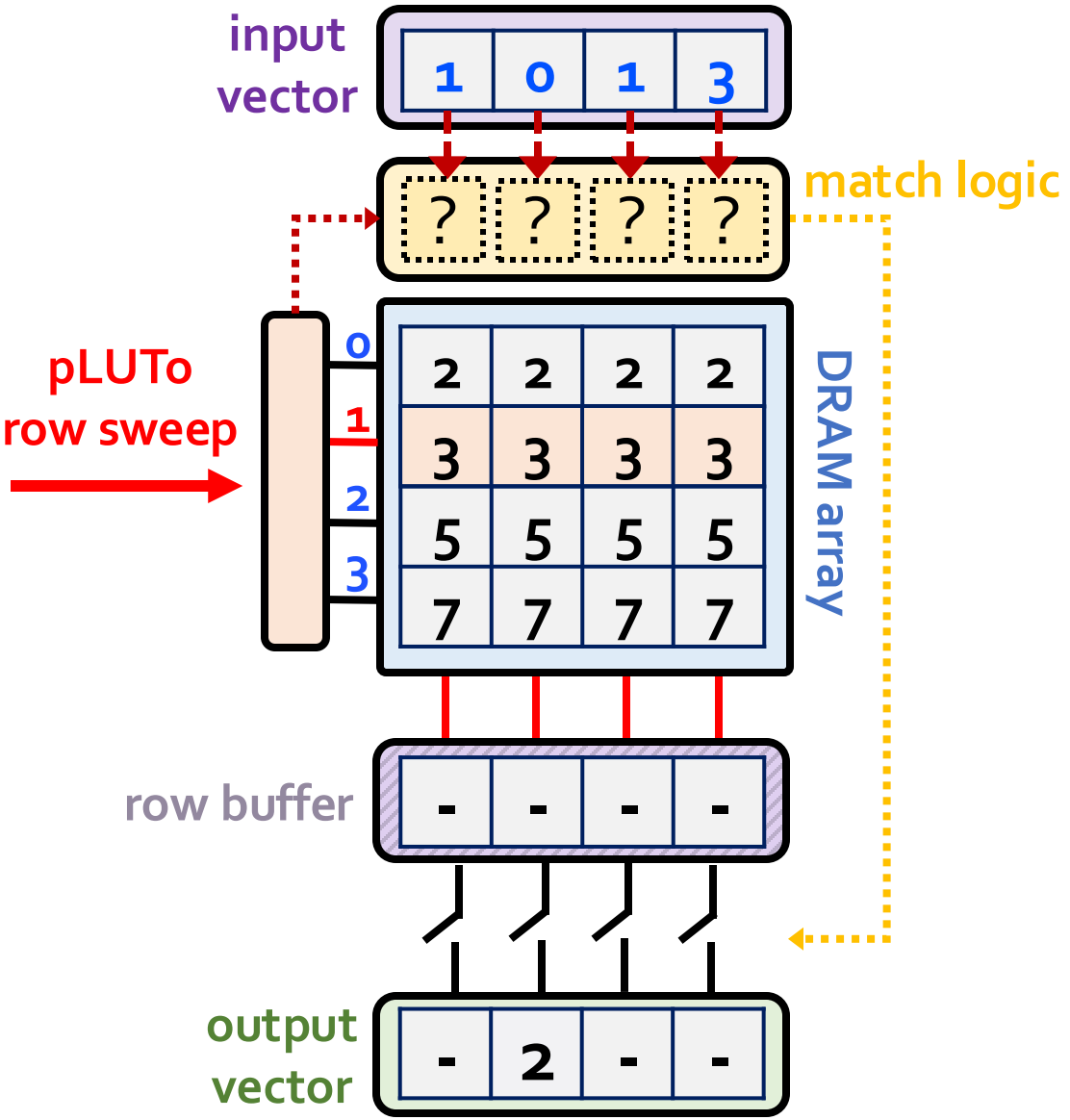
lookup table

1	0	1	3

input vector

3	2	3	7

output vector



In-DRAM pLUTo LUT Query: Step 2

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

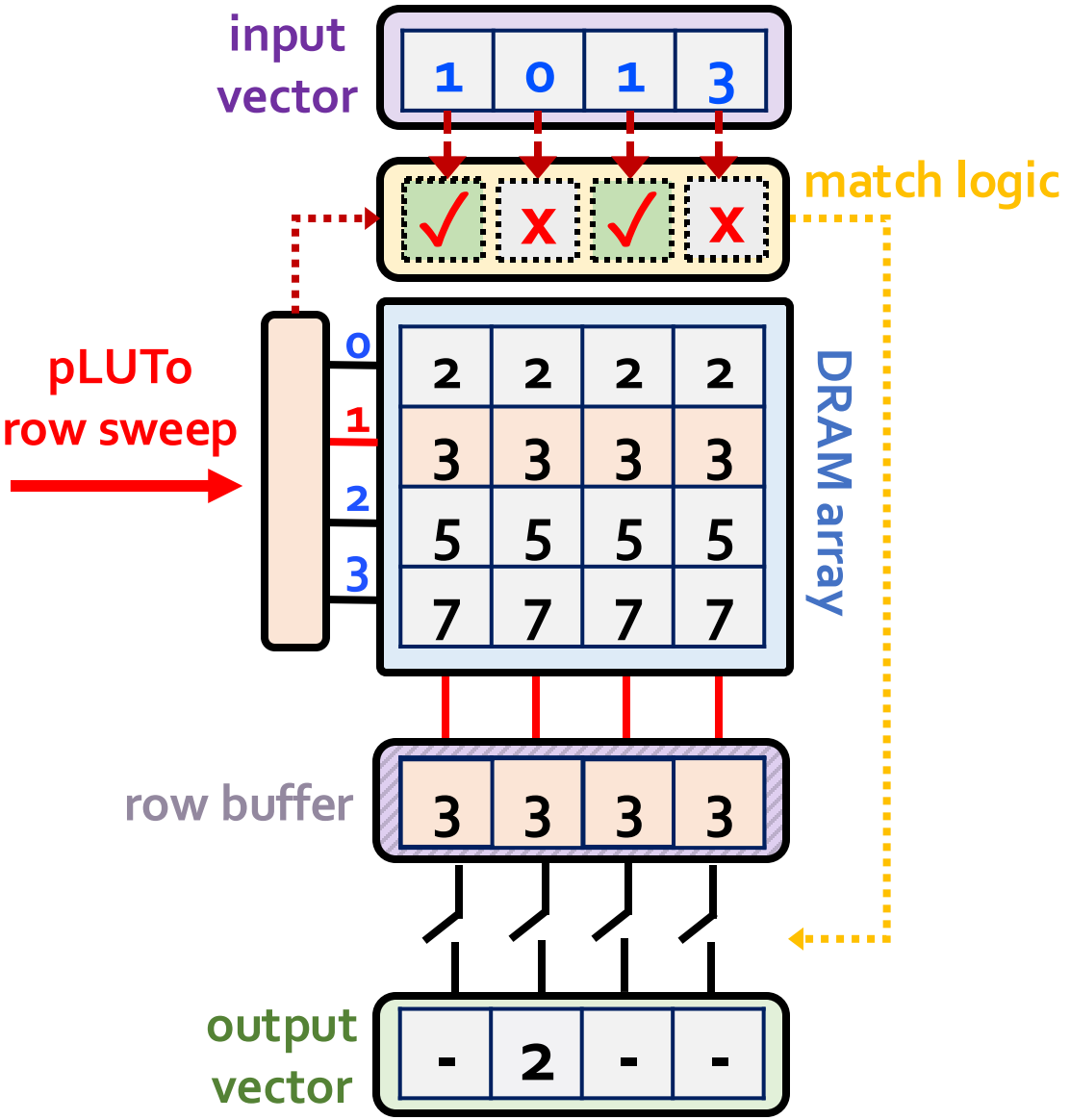
lookup table

1	0	1	3

input vector

3	2	3	7

output vector



In-DRAM pLUTo LUT Query: Step 2

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

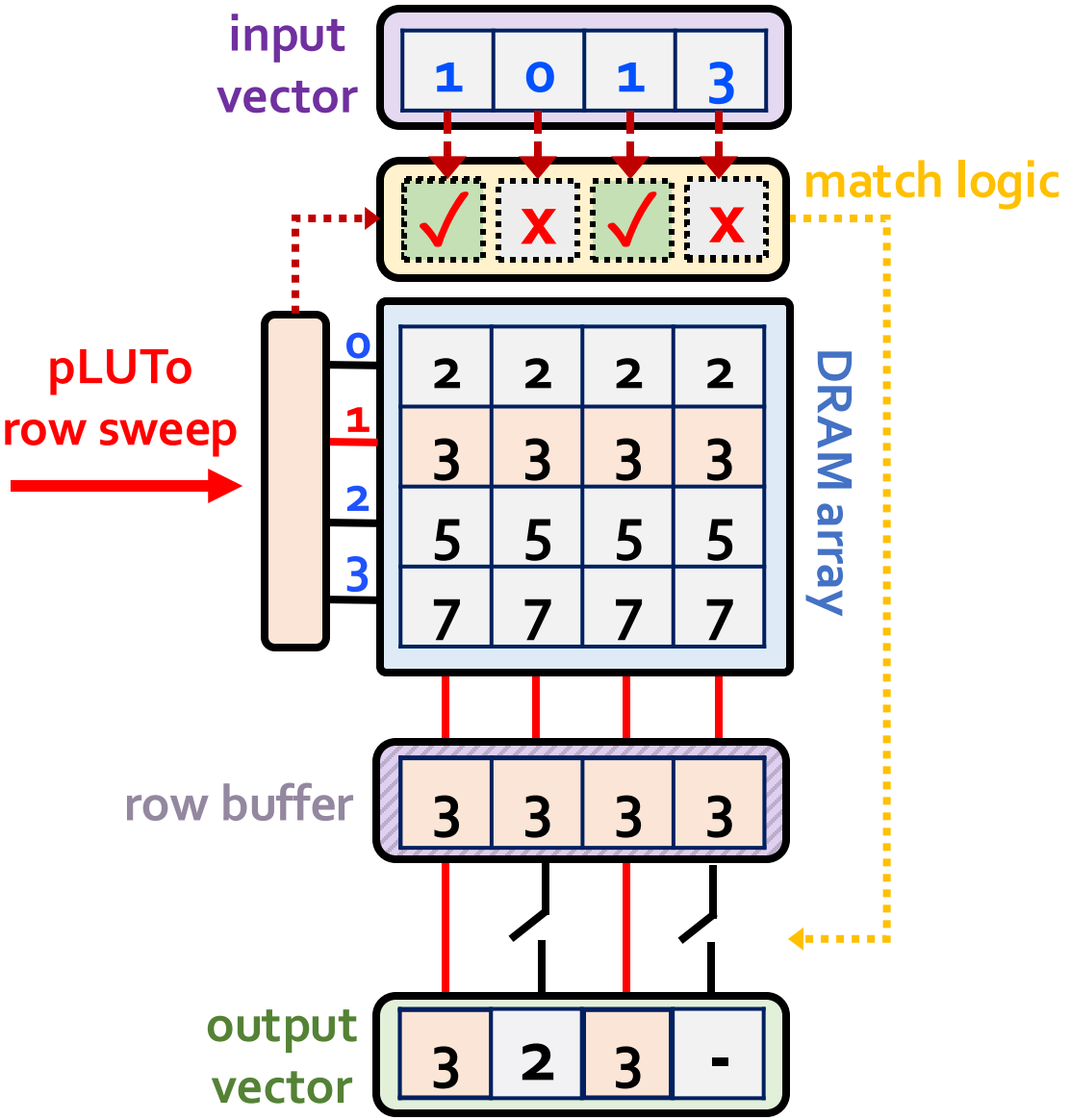
lookup table

1	0	1	3

input vector

3	2	3	7

output vector



In-DRAM pLUTo LUT Query: Step 3

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

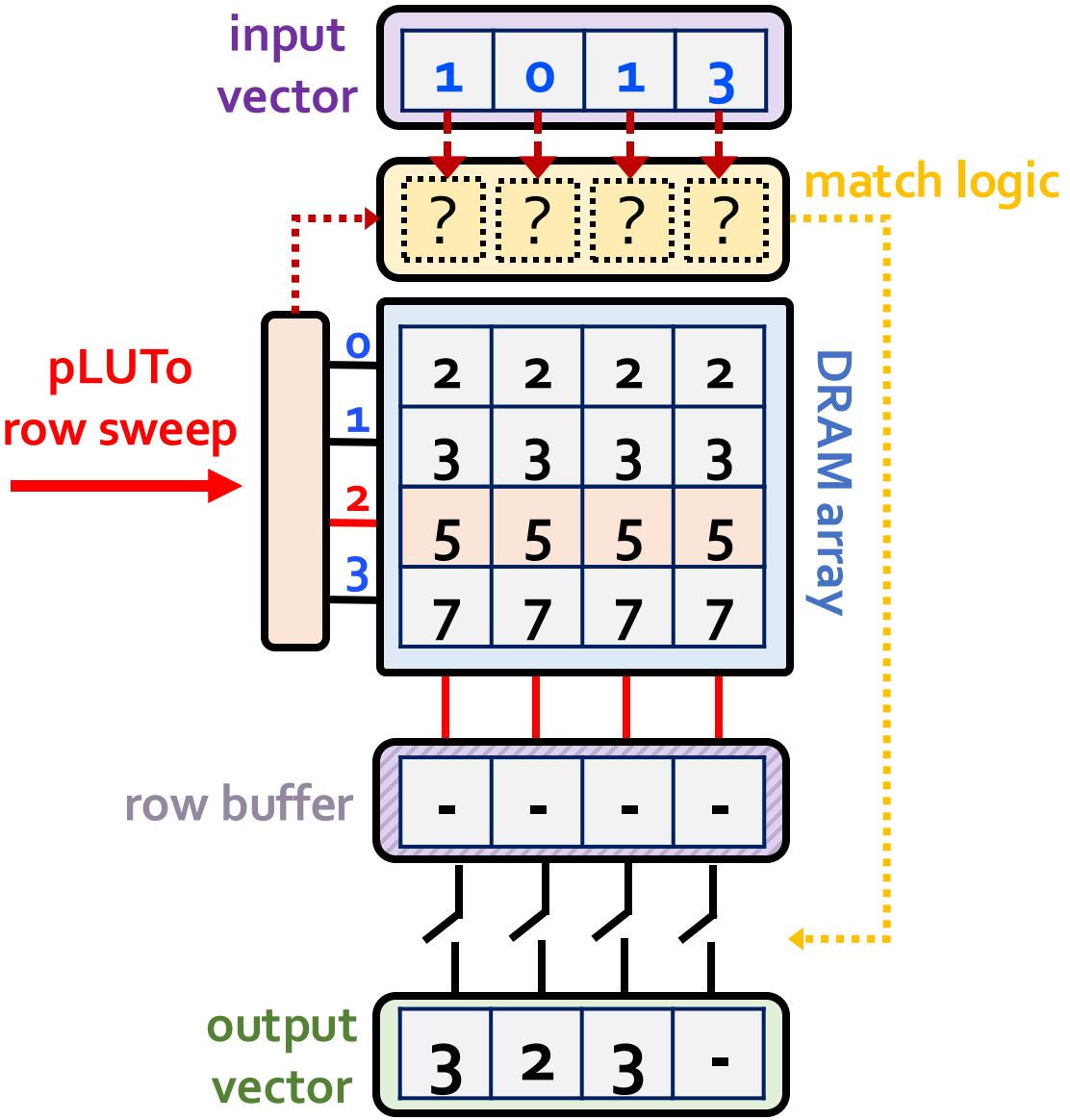
lookup table

1	0	1	3

input vector

3	2	3	7

output vector



In-DRAM pLUTo LUT Query: Step 3

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

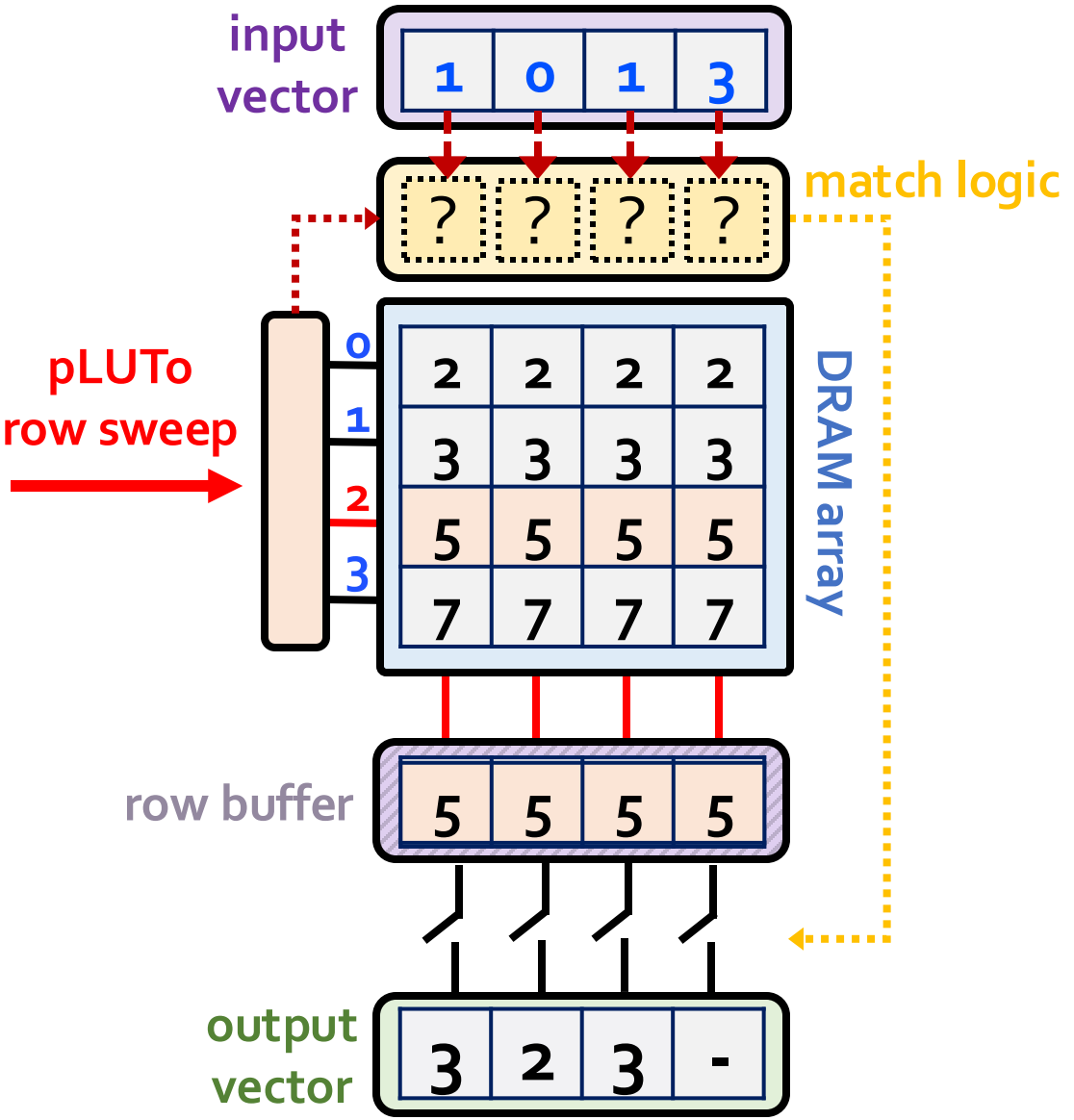
lookup table

1	0	1	3

input vector

3	2	3	7

output vector



In-DRAM pLUTo LUT Query: Step 3

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

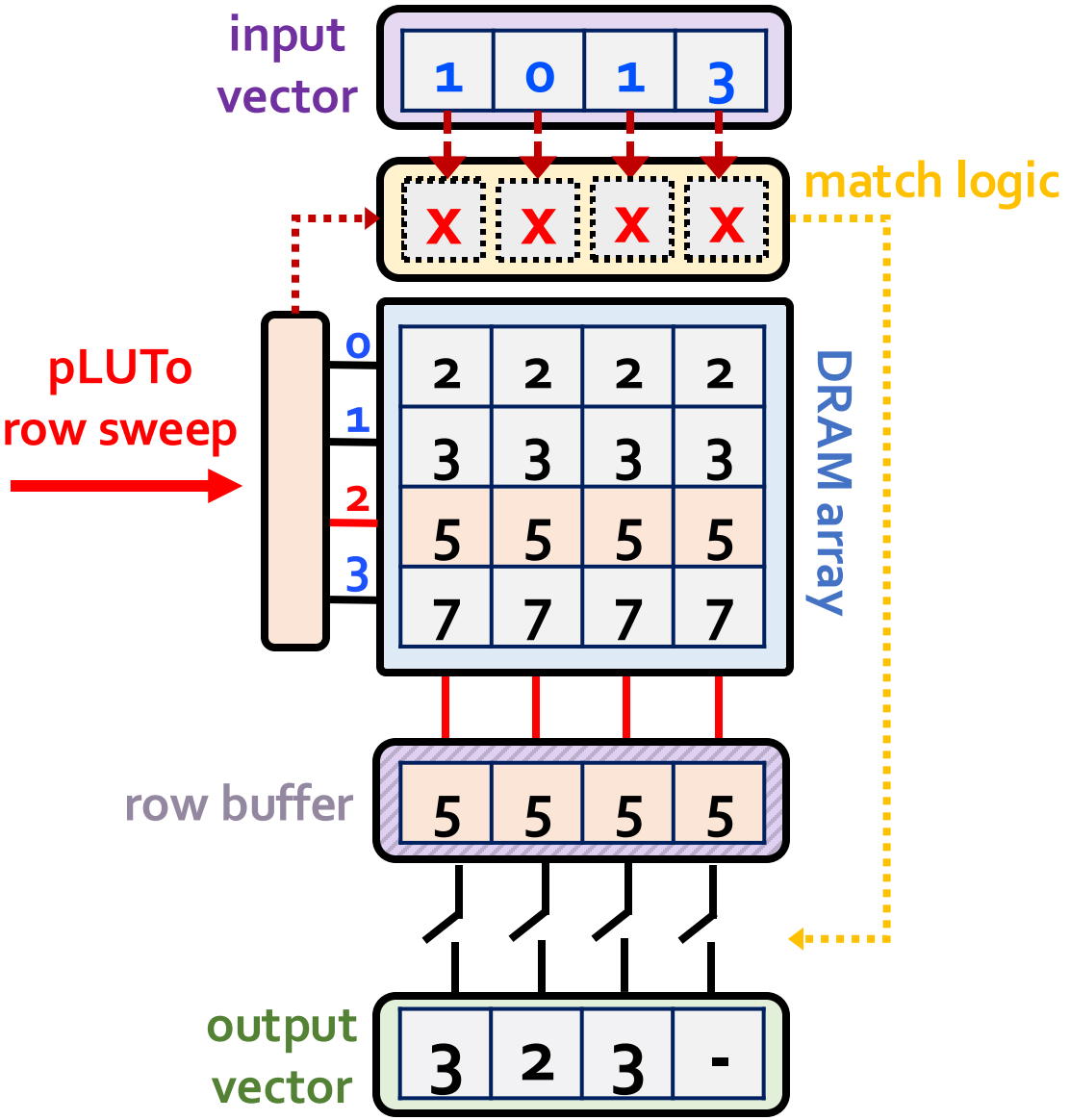
lookup table

1	0	1	3

input vector

3	2	3	7

output vector



In-DRAM pLUTo LUT Query: Step 4

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

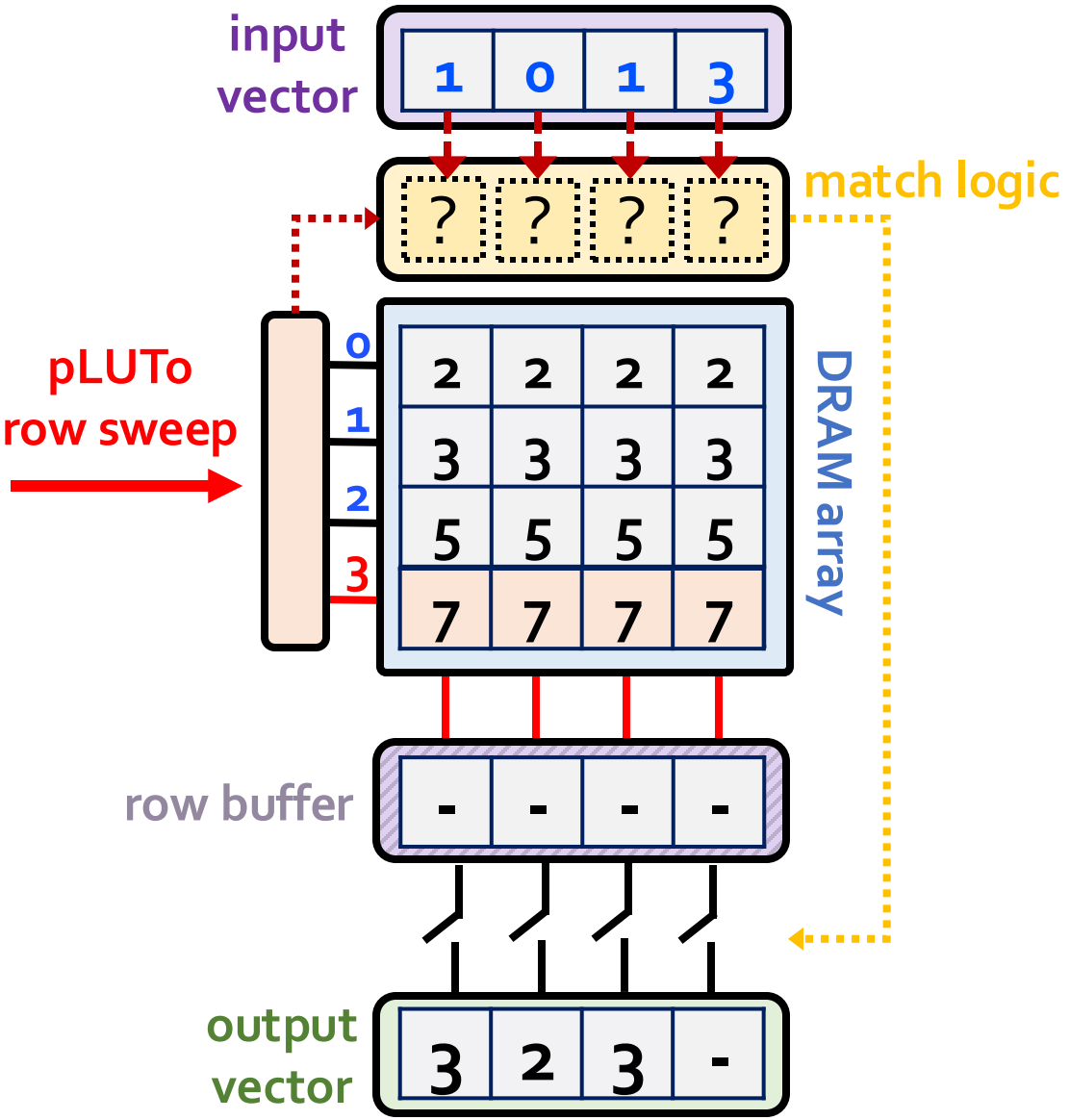
lookup table

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

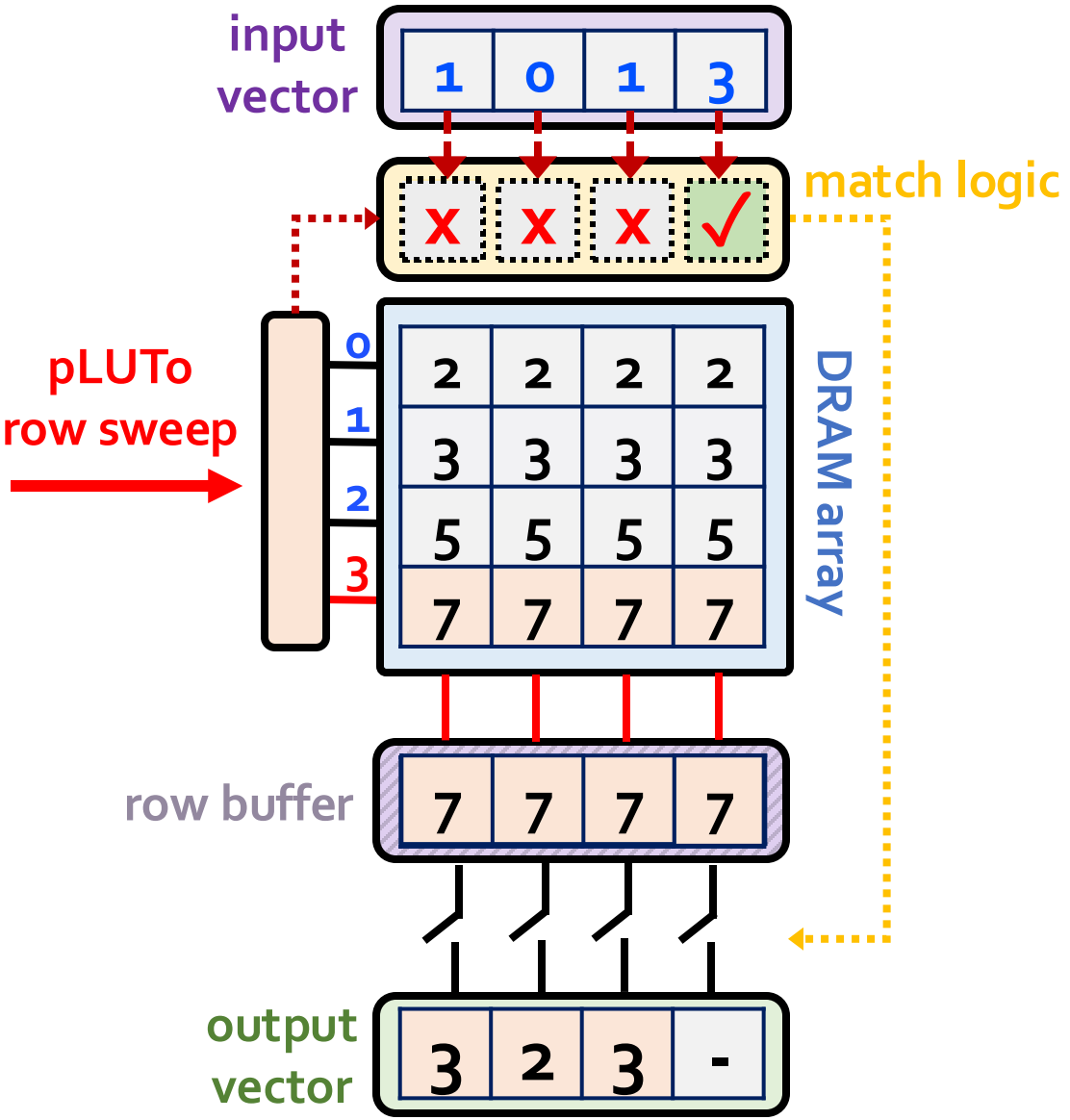
lookup table

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

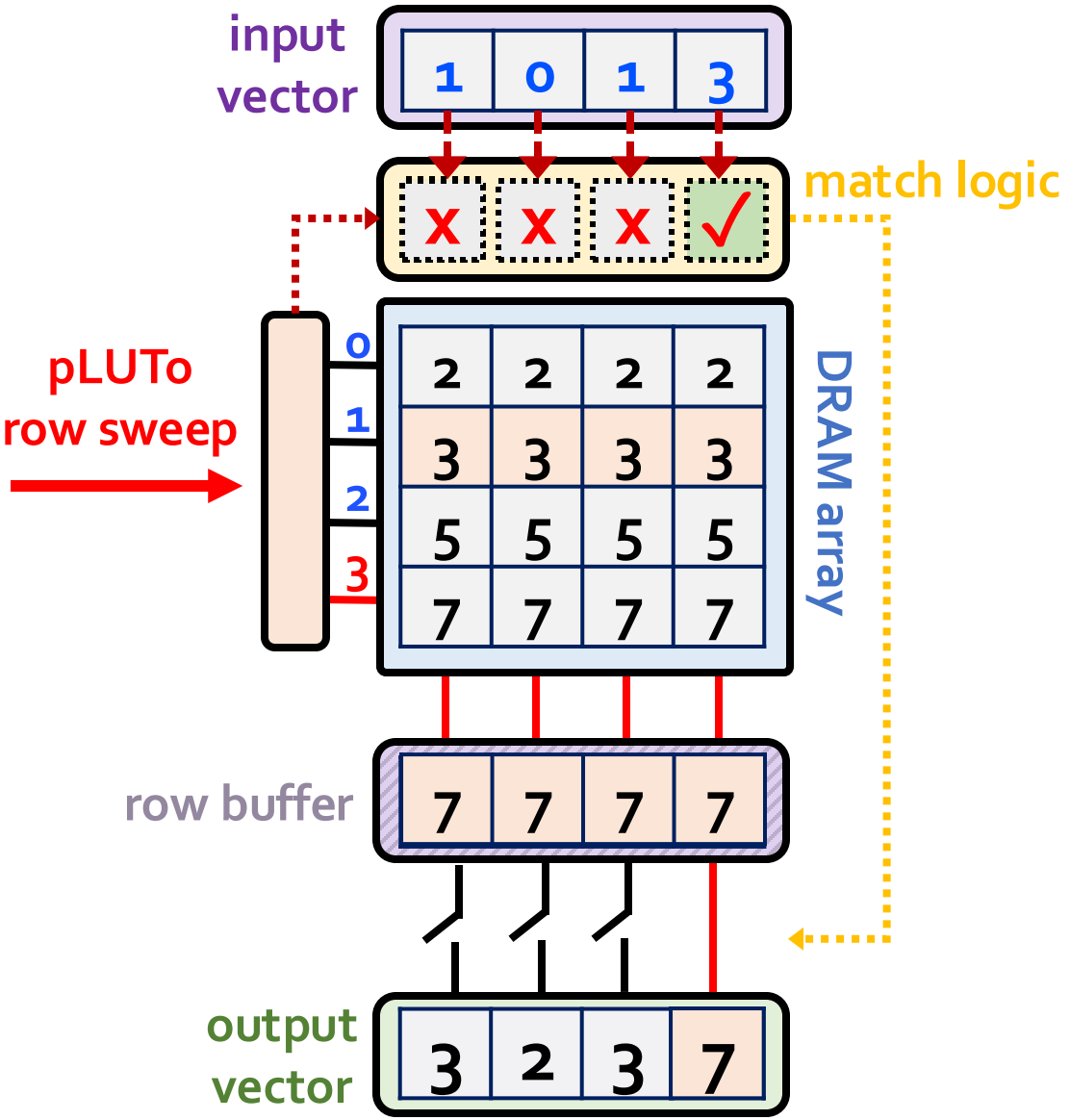
lookup table

1	0	1	3

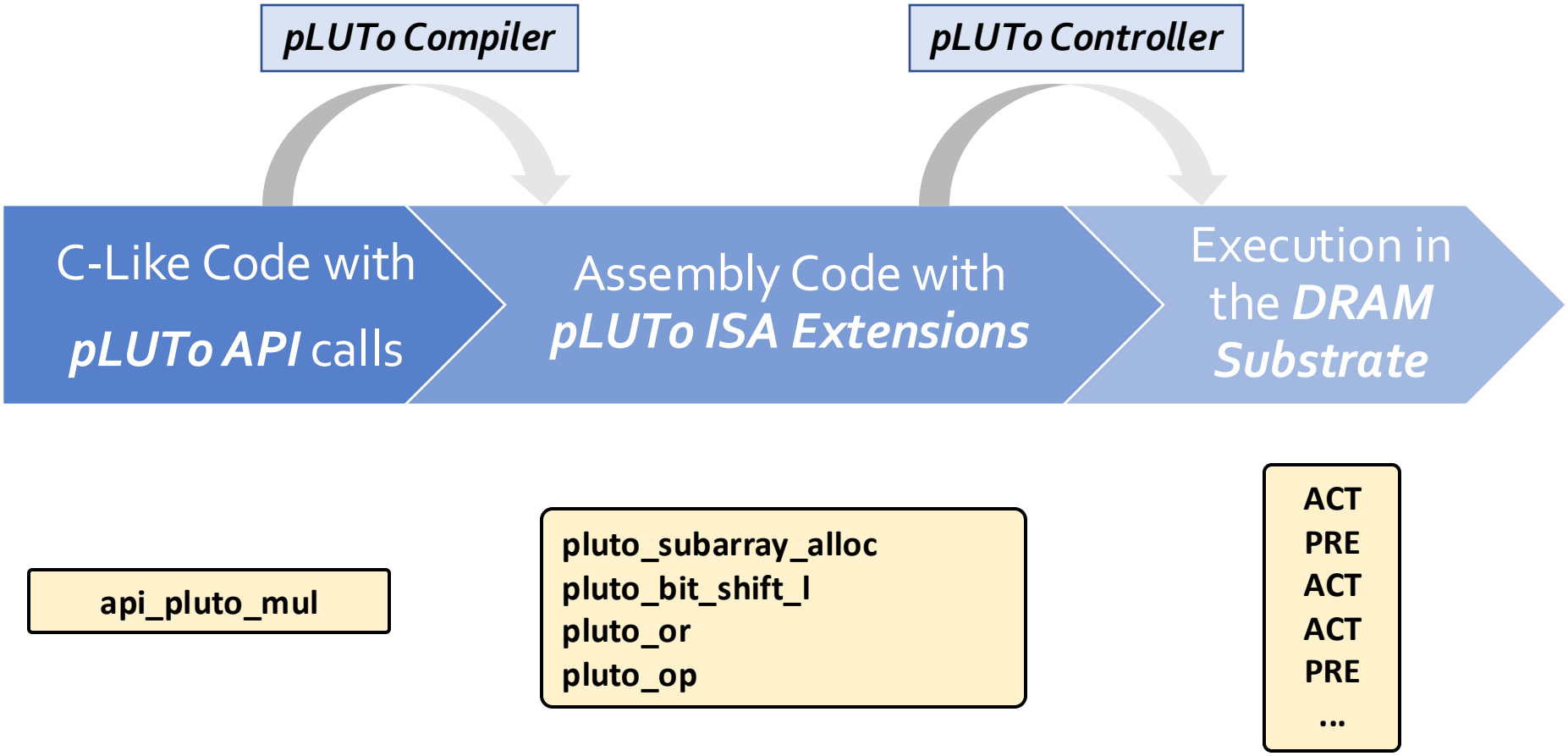
input vector

3	2	3	7

output vector

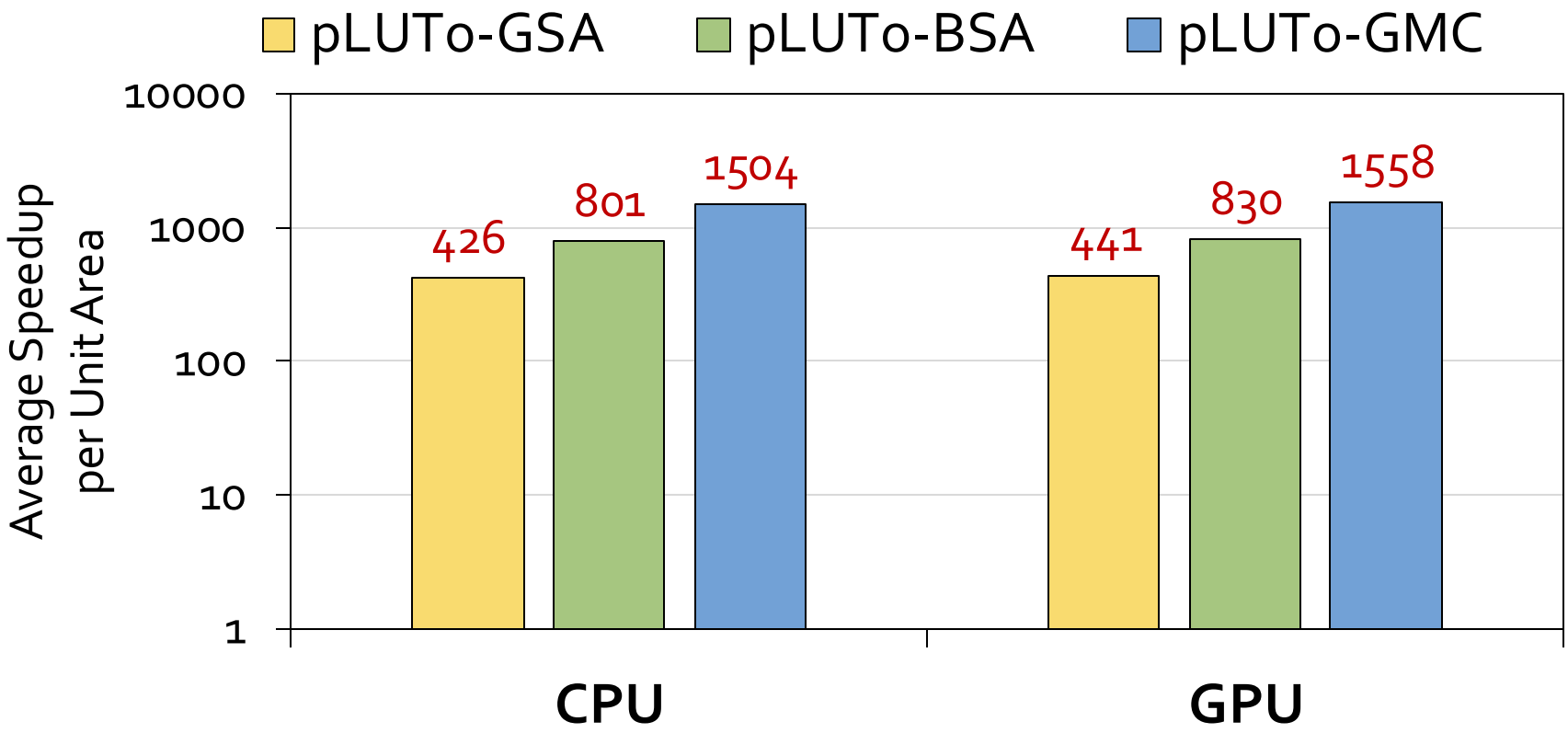


System Integration



Performance (normalized to area)

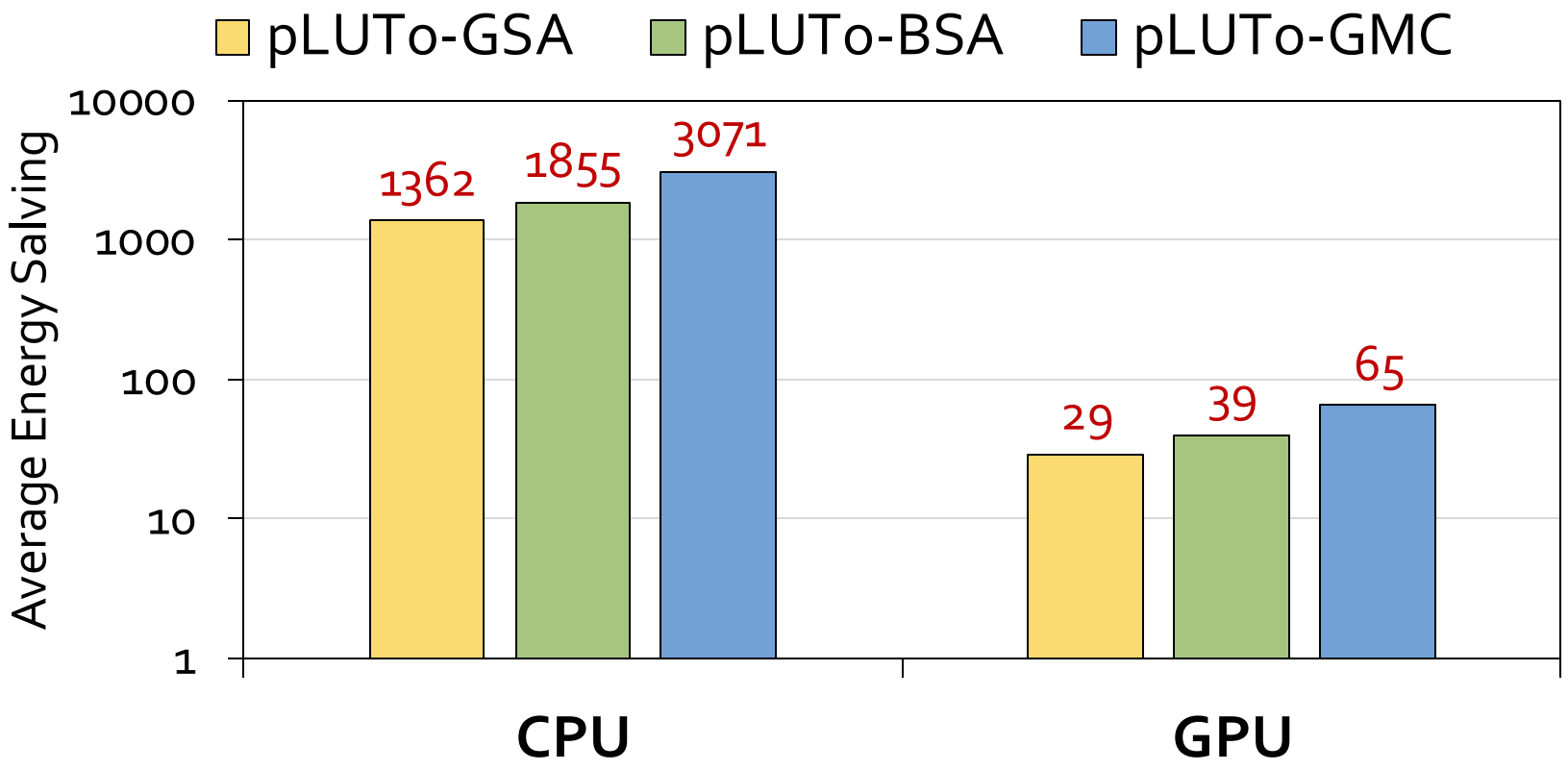
Average speedup normalized to area across 7 real-world workloads



pLUTo provides *substantially higher* performance per unit area than *both* the CPU and the GPU

Energy Consumption

Average energy consumption across 7 real-world workloads



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

More Results in the Paper

- Comparison with FPGA
- Area Overhead Analysis
- Circuit-Level Reliability & Correctness
- Subarray-Level Parallelism
- LUT Loading Overhead
- Range of Supported Operations

pLUTo: Enabling Massively Parallel Computation in DRAM via Lookup Tables

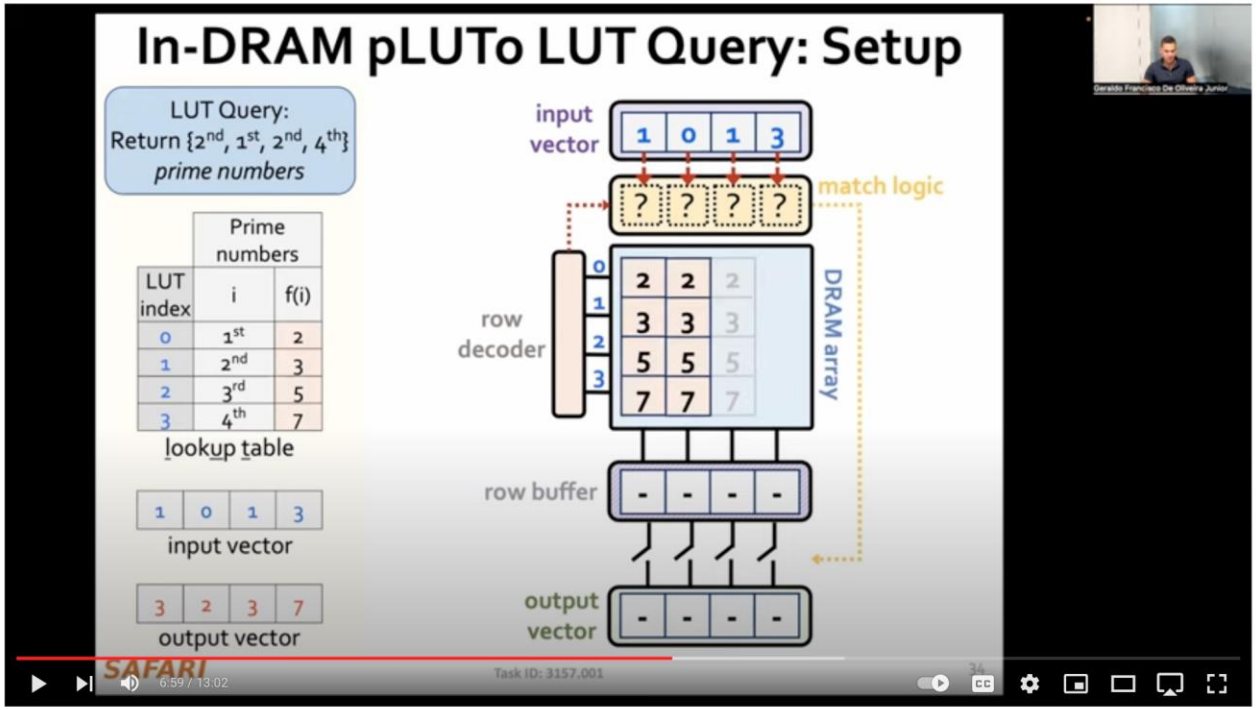
João Dinis Ferreira[§] Gabriel Falcao[†] Juan Gómez-Luna[§] Mohammed Alser[§]
Lois Orosa[§▽] Mohammad Sadrosadati[§] Jeremie S. Kim[§] Geraldo F. Oliveira[§]
Taha Shahroodi[‡] Anant Nori[⋆] Onur Mutlu[§]

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

https://arxiv.org/pdf/2104.07699.pdf

SRC TECHCON Presentation

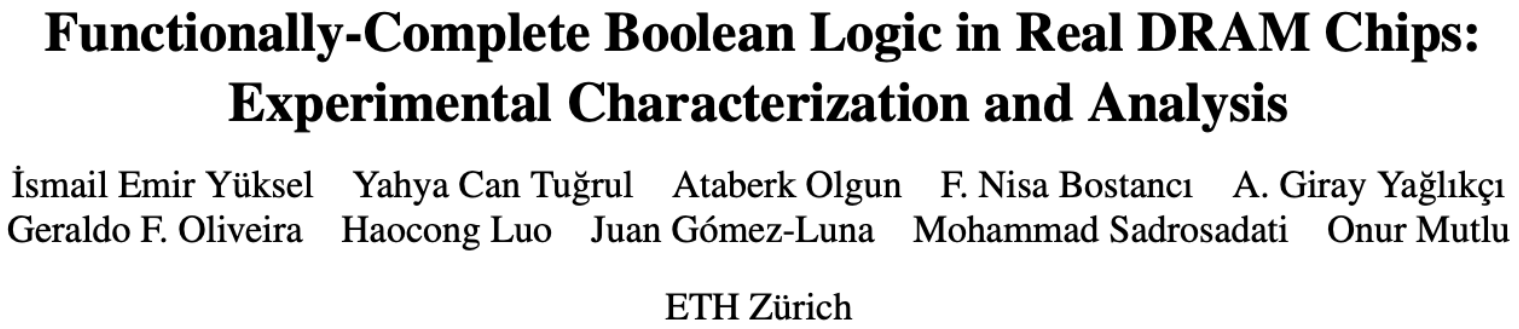
- Geraldo F. Oliveira
  - pLUTo: Enabling Massively Parallel Computation in DRAM via Lookup Tables
  - https://arxiv.org/pdf/2104.07699.pdf



https://youtu.be/9t1FJQ6nNw4?si=bhylWCLZde2DC7os

Bulk Bitwise Operations in Real DRAM Chips

- Ismail Emir Yüksel, Yahya Can Tugrul Ataberk Olgun, F. Nisa Bostancı, A. Giray Yaglıkçı, Geraldo F. Oliveira, Haocong Luo, Juan Gómez-Luna, Mohammad Sadrosadati, 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**)*, Edinburgh, Scotland, March 2024.



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

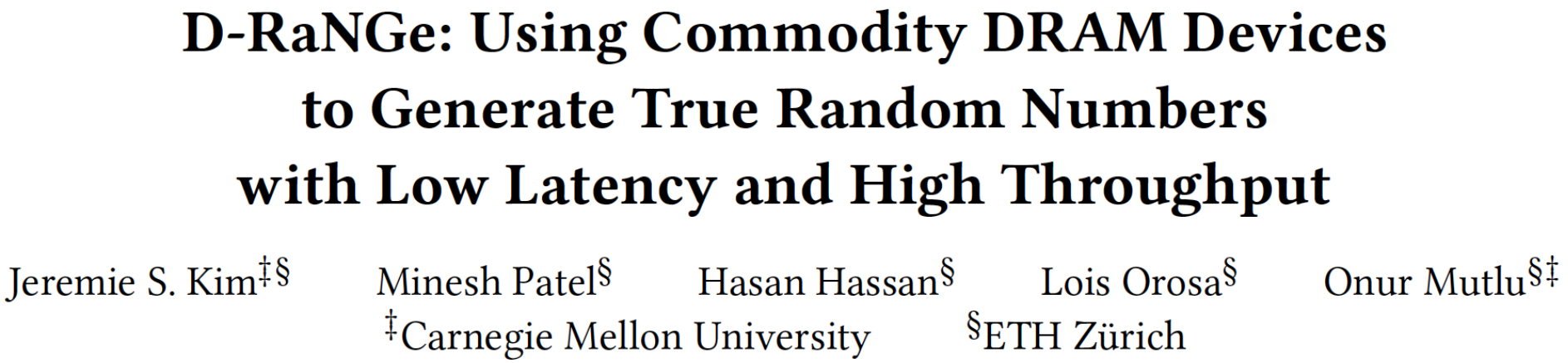
İsmail Emir Yüksel Yahya Can Tuğrul Ataberk Olgun F. Nisa Bostancı A. Giray Yağlıkçı
Geraldo F. Oliveira Haocong Luo Juan Gómez-Luna Mohammad Sadrosadati Onur Mutlu

ETH Zürich

https://arxiv.org/pdf/2402.18736

In-DRAM True Random Number Generation

- 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

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 Bostancı[§†] Nandita Vijaykumar[§⊙] Oğuz Ergin[†] Onur Mutlu[§]

[§]*ETH Zürich* [†]*TOBB University of Economics and Technology* [⊙]*University of Toronto*

In-DRAM True Random Number Generation

- F. Nisa Bostanci, Ataberk Olgun, Lois Orosa, A. Giray Yaglikci, Jeremie S. Kim, Hasan Hassan, Oguz Ergin, and Onur Mutlu,
  "DR-STRaNGe: End-to-End System Design for DRAM-based True Random Number Generators"
  *Proceedings of the 28th International Symposium on High-Performance Computer Architecture (**HPCA**)*, Virtual, April 2022.
  [Slides (pptx) (pdf)]
  [Short Talk Slides (pptx) (pdf)]

DR-STRaNGe: End-to-End System Design for DRAM-based True Random Number Generators

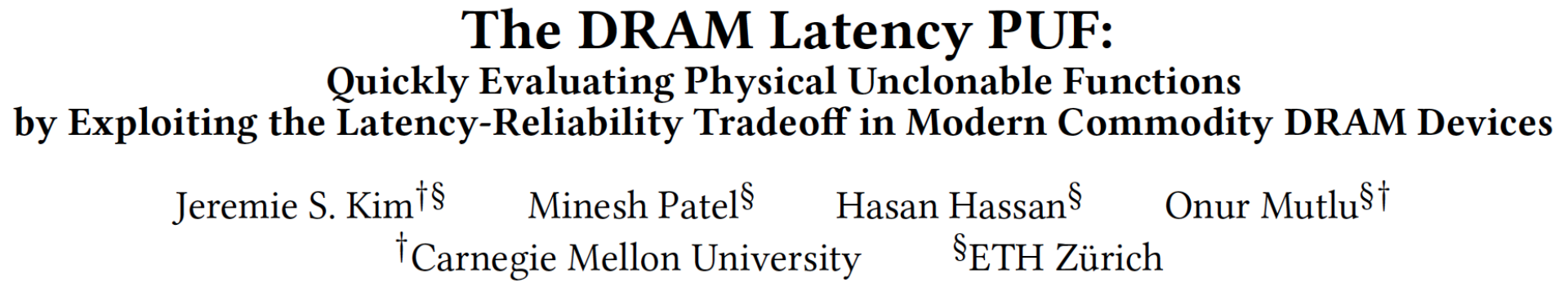
F. Nisa Bostancı[†§] Ataberk Olgun[†§] Lois Orosa[§] A. Giray Yağlıkçı[§]
Jeremie S. Kim[§] Hasan Hassan[§] Oğuz Ergin[†] Onur Mutlu[§]

[†]*TOBB University of Economics and Technology* [§]*ETH Zürich*

https://arxiv.org/pdf/2201.01385.pdf

In-DRAM 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

In-Flash Bulk Bitwise Execution

- Jisung Park, Roknoddin Azizi, Geraldo F. Oliveira, Mohammad Sadrosadati, Rakesh Nadig, David Novo, Juan Gómez-Luna, Myungsuk Kim, and Onur Mutlu,
  "Flash-Cosmos: In-Flash Bulk Bitwise Operations Using Inherent Computation Capability of NAND Flash Memory"
  Proceedings of the 55th International Symposium on Microarchitecture (***MICRO***), Chicago, IL, USA, October 2022.
  [Slides (pptx) (pdf)]
  [Longer Lecture Slides (pptx) (pdf)]
  [Lecture Video (44 minutes)]
  [arXiv version]

Flash-Cosmos: In-Flash Bulk Bitwise Operations Using Inherent Computation Capability of NAND Flash Memory

Jisung Park[§∇] Roknoddin Azizi[§] Geraldo F. Oliveira[§] Mohammad Sadrosadati[§]
Rakesh Nadig[§] David Novo[†] Juan Gómez-Luna[§] Myungsuk Kim[‡] Onur Mutlu[§]

[§]*ETH Zürich* [∇]*POSTECH* [†]*LIRMM, Univ. Montpellier, CNRS* [‡]*Kyungpook National University*

https://arxiv.org/pdf/2209.05566.pdf

1st Workshop on Memory-Centric Computing: Processing-Using-Memory

Geraldo F. Oliveira

https://geraldofojunior.github.io

ASPLOS 2025

30 March 2025

SAFARI

ETH zürich