Processing-Using-Memory Systems for Bulk Bitwise Operations

June 21st 2025

Geraldo F. Oliveira
geraldofojunior@gmail.com
https://geraldofojunior.github.io

SAFARI

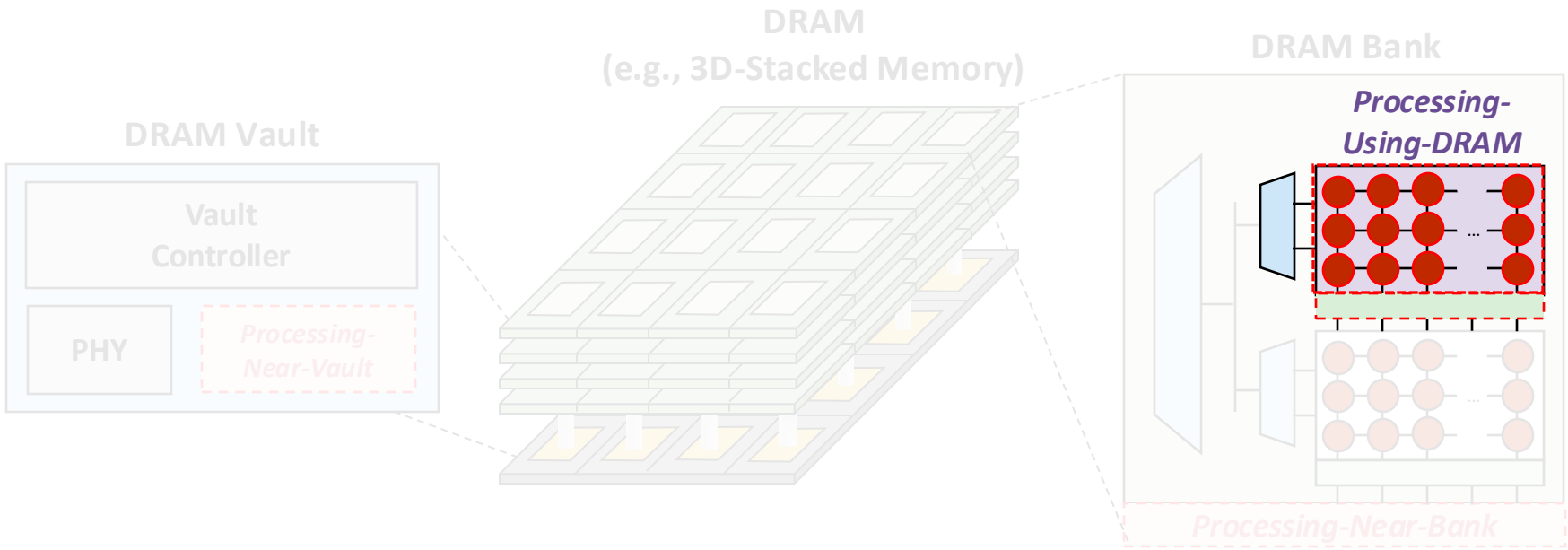
ETH zürich

Processing-in-Memory:
Overview

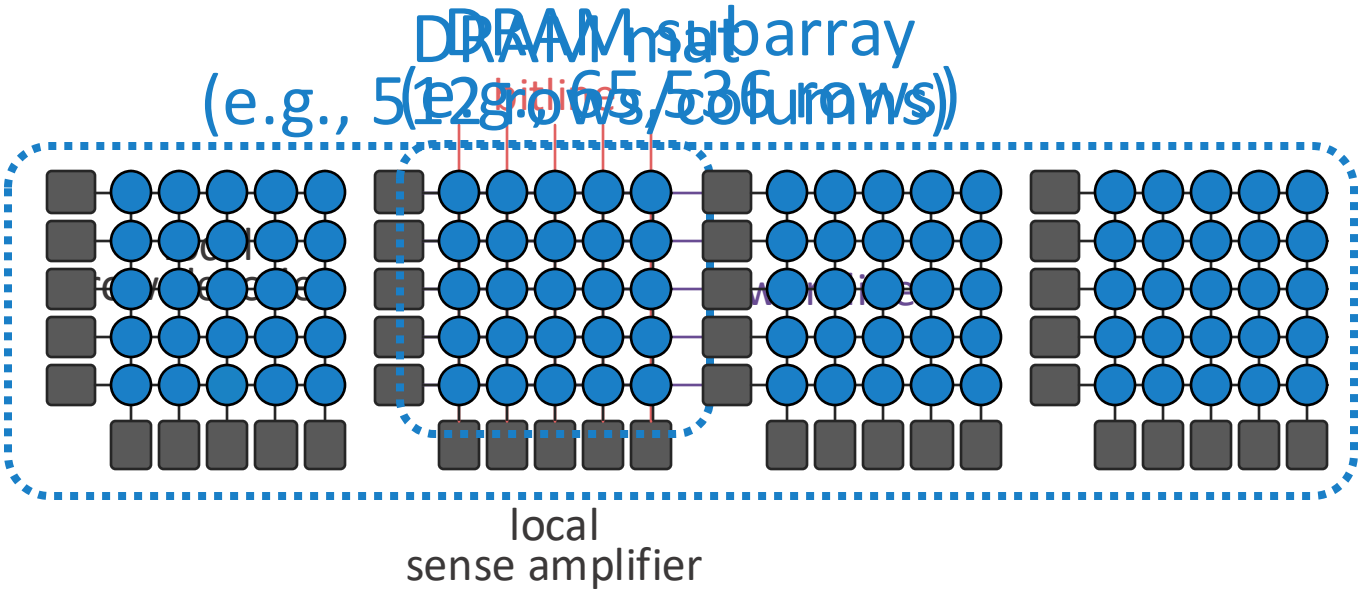
Two main approaches for Processing-in-Memory:

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

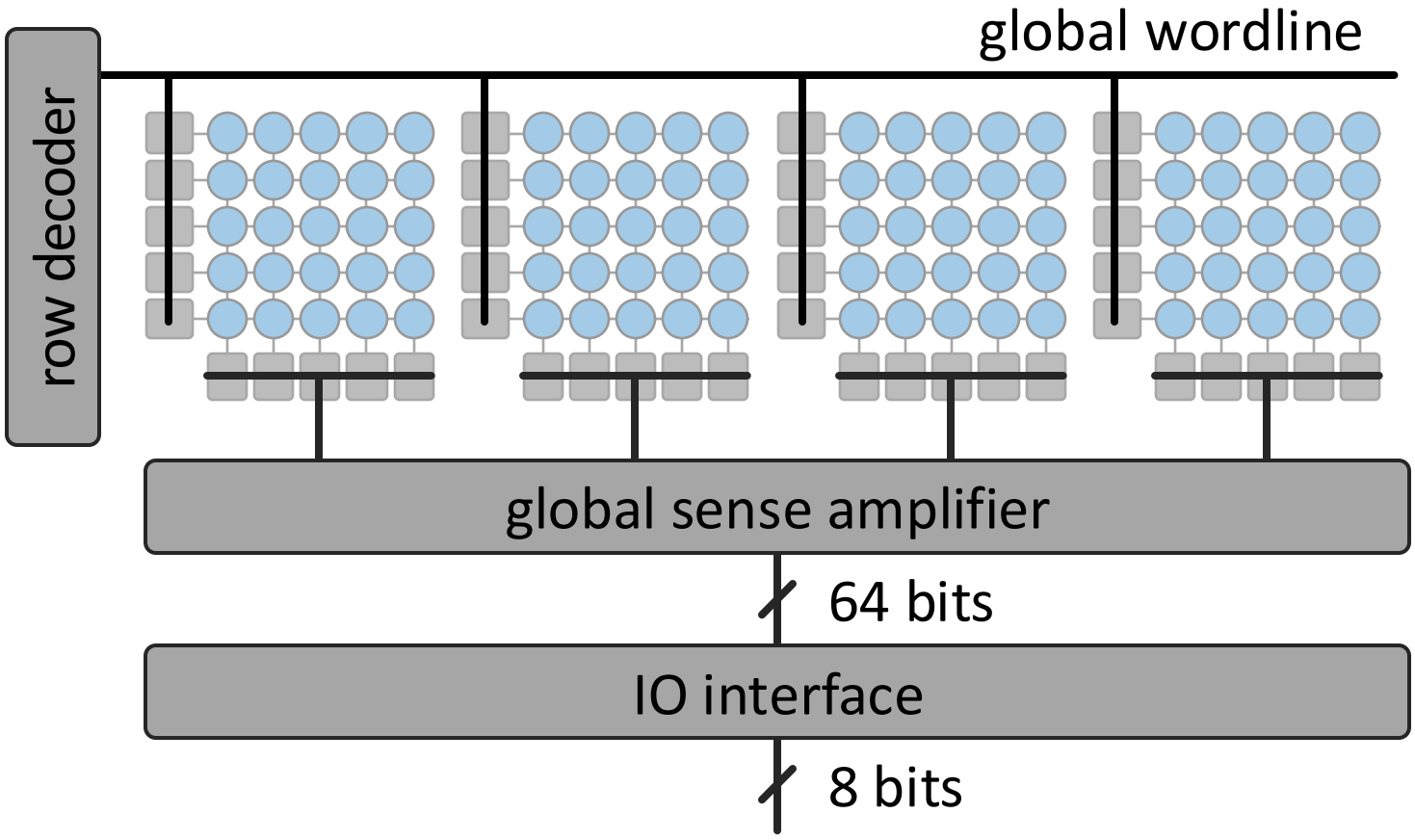
2 **Processing-Using-Memory**: uses the operational principles of memory cells to perform computation



Background:
DRAM Hierarchical Organization

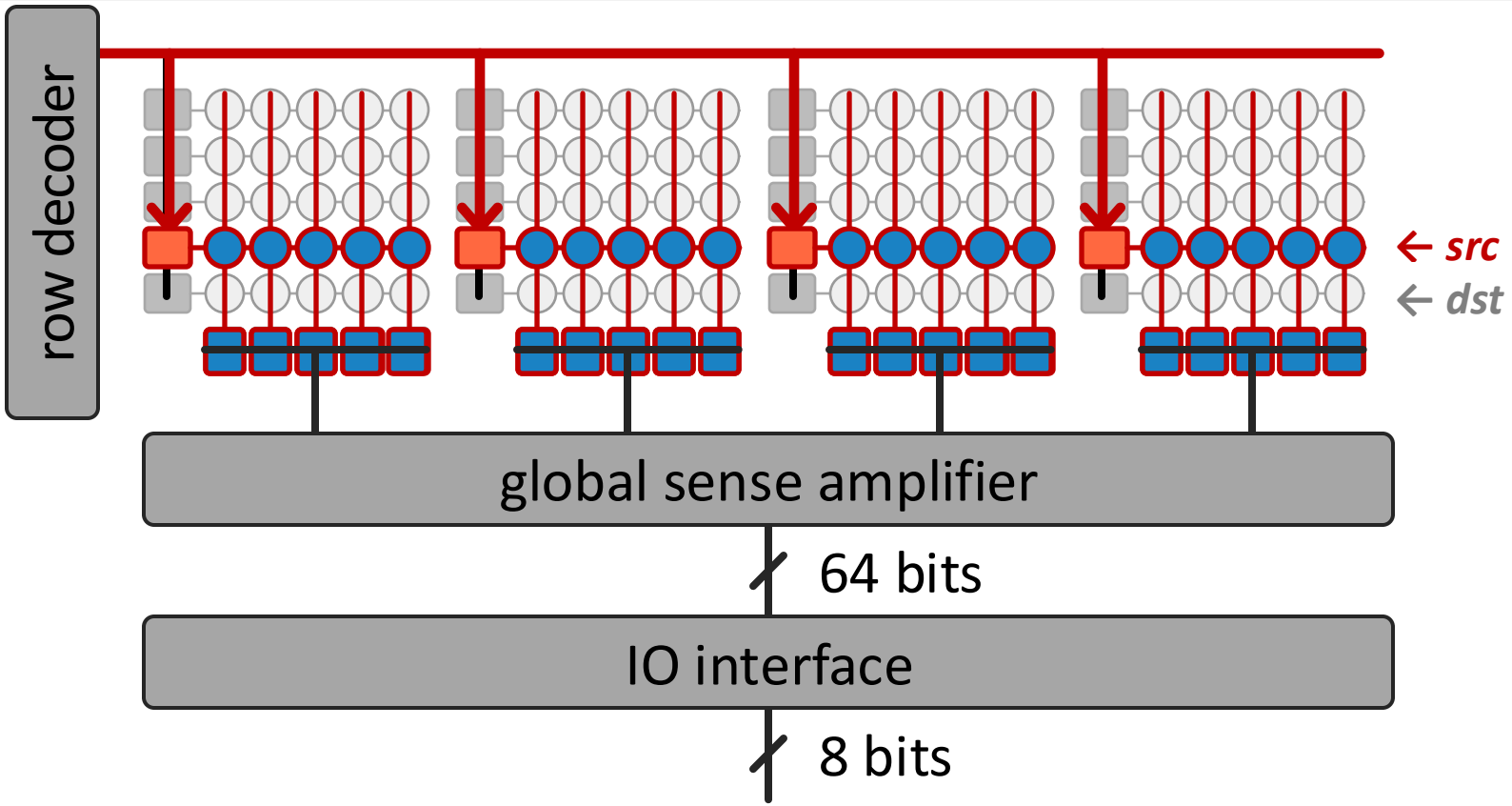


Background:
DRAM Hierarchical Organization



Background:
In-DRAM Row Copy

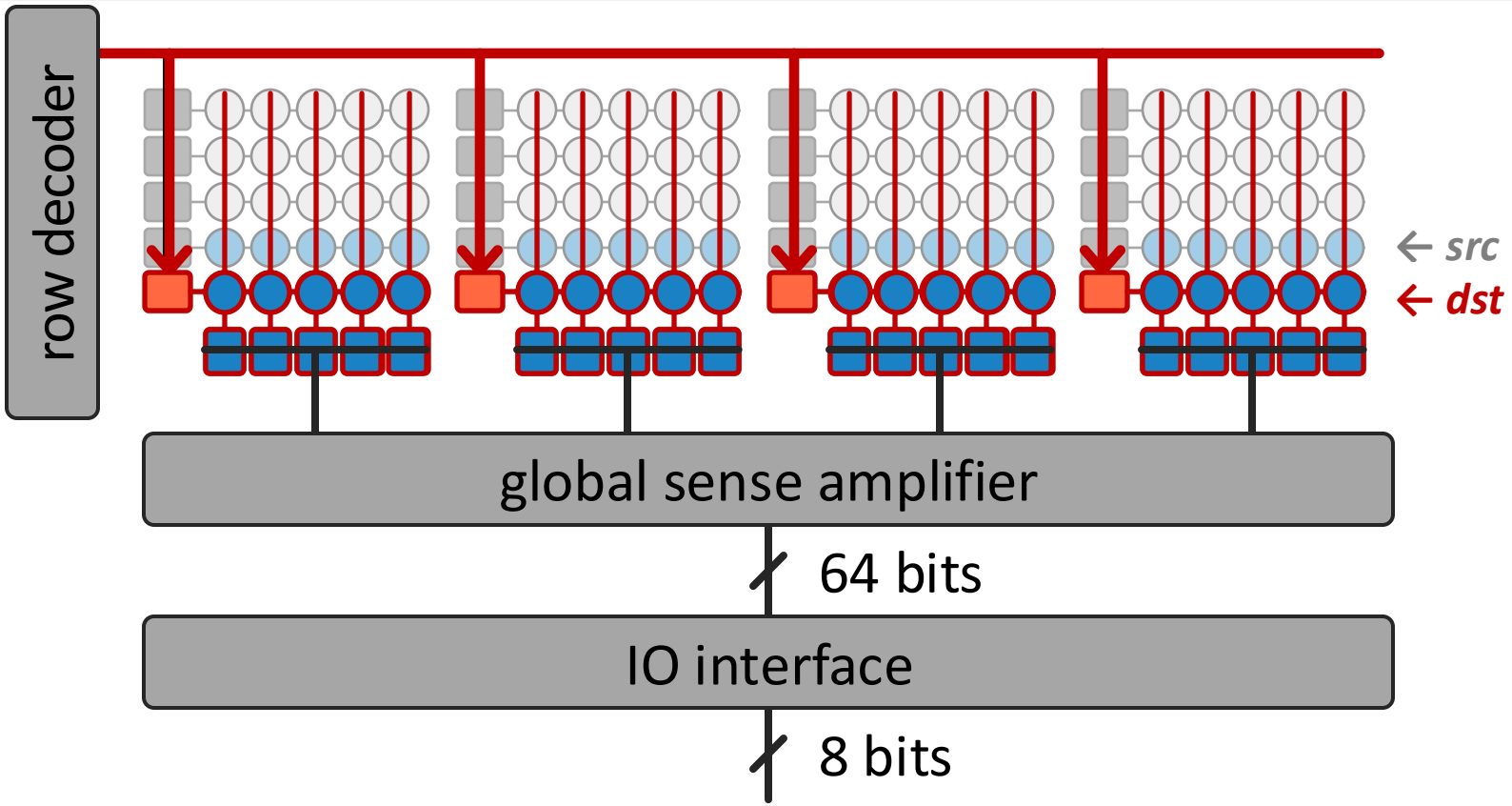
In-DRAM row copy is performed by
issuing back-to-back ACTIVATES to the DRAM



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

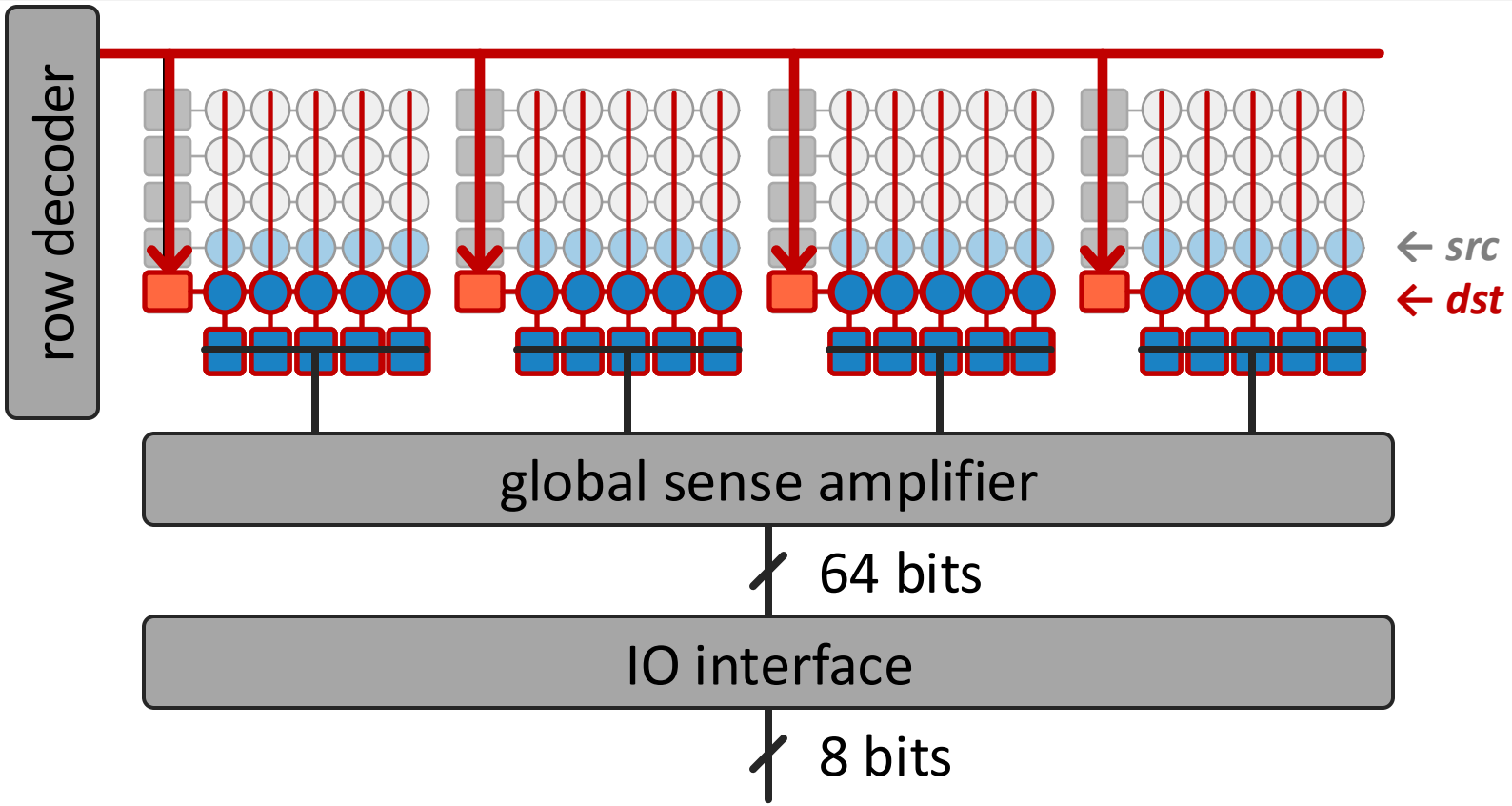


Seshadri, Vivek, et al. "RowClone: Fast and Energy-Efficient In-DRAM Bulk Data Copy and Initialization," in MICRO, 2013

Background:
In-DRAM Row Copy

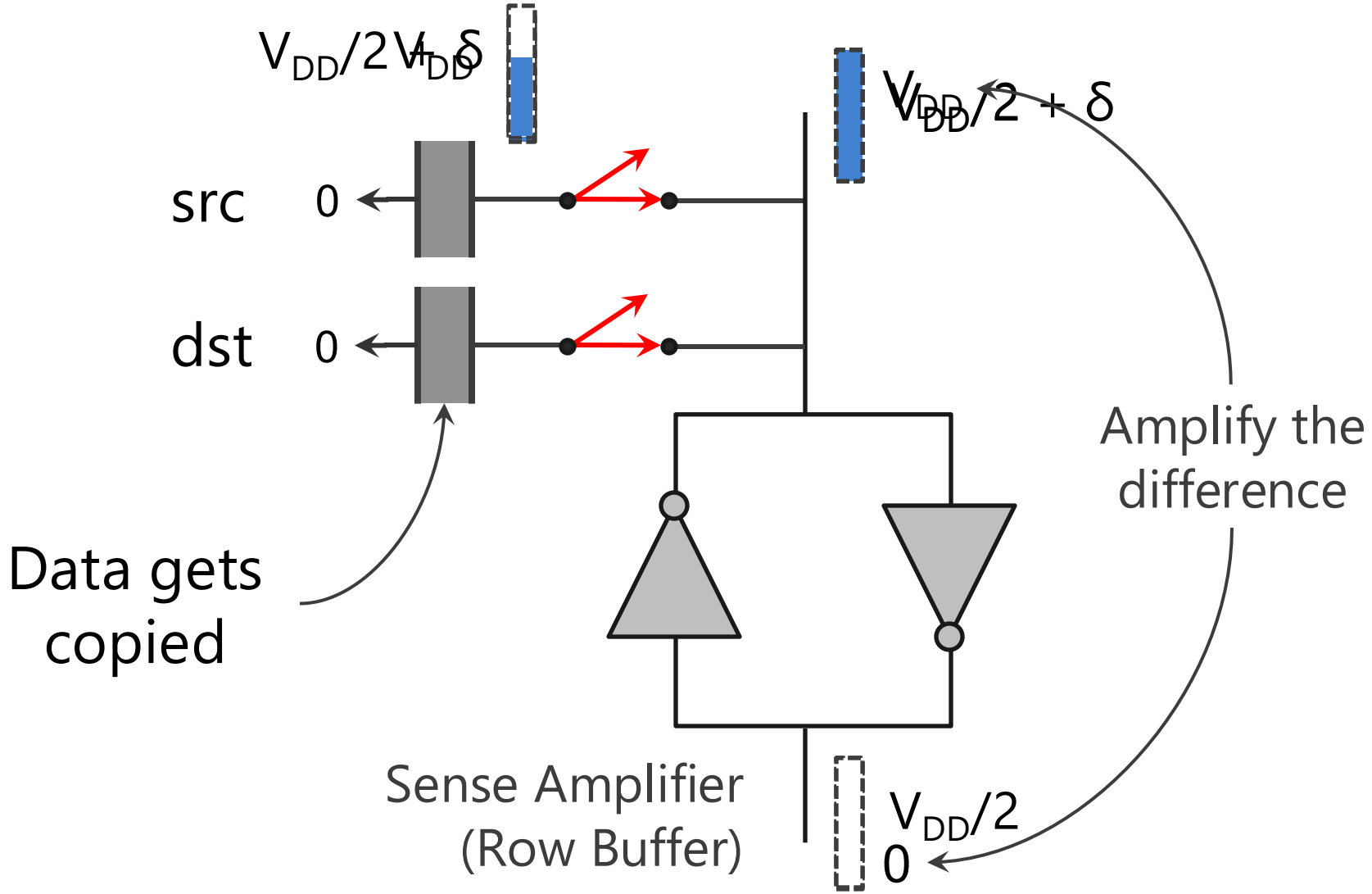
In-DRAM row copy improves performance & energy:

1046 ns, 3.6 uJ → 90 ns, 0.04 nJ

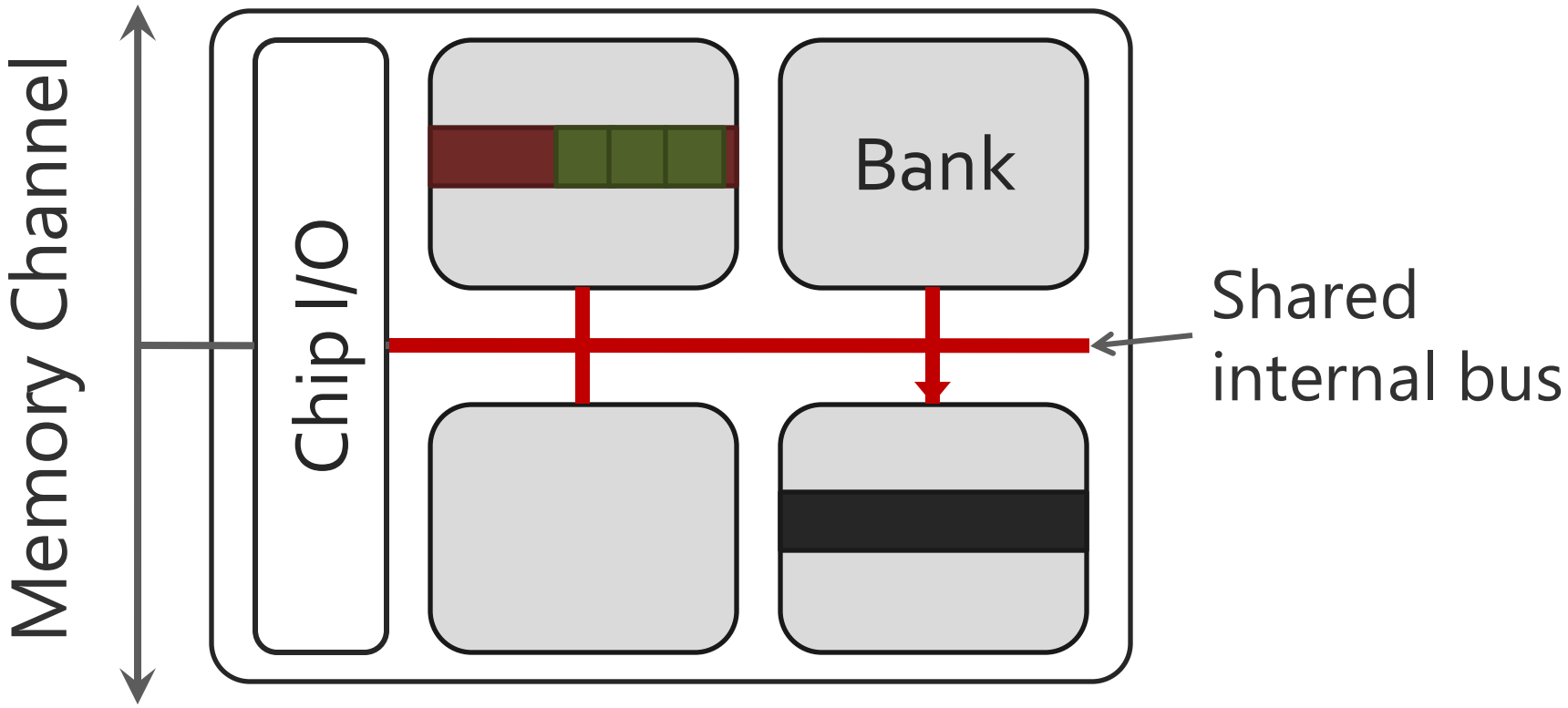


Seshadri, Vivek, et al. "**RowClone: Fast and Energy-Efficient In-DRAM Bulk Data Copy and Initialization**," in MICRO, 2013

RowClone: Intra-Subarray

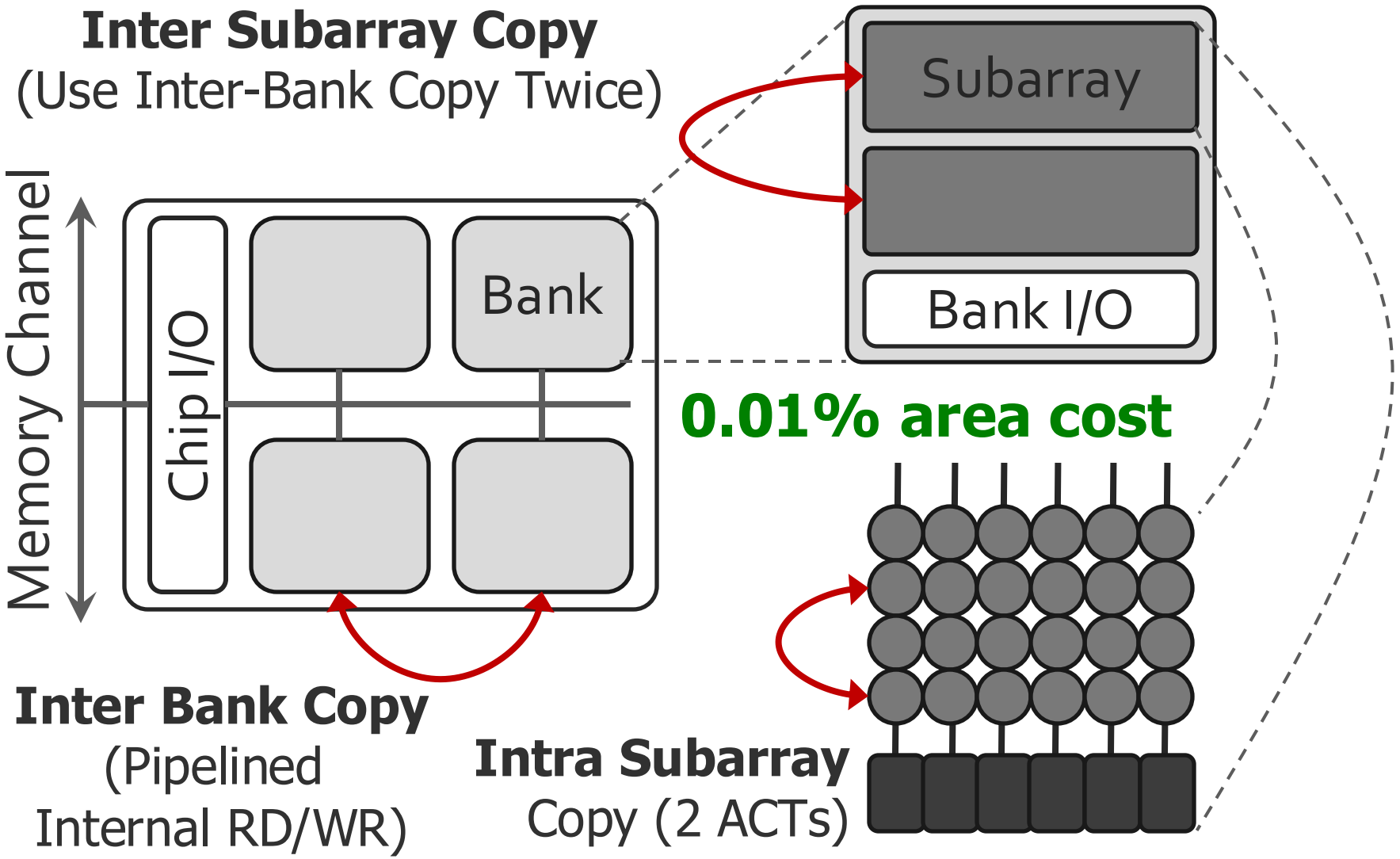


RowClone: Inter-Bank



Overlap the latency of the read and the write
1.9X latency reduction, 3.2X energy reduction

Generalized RowClone



More on RowClone

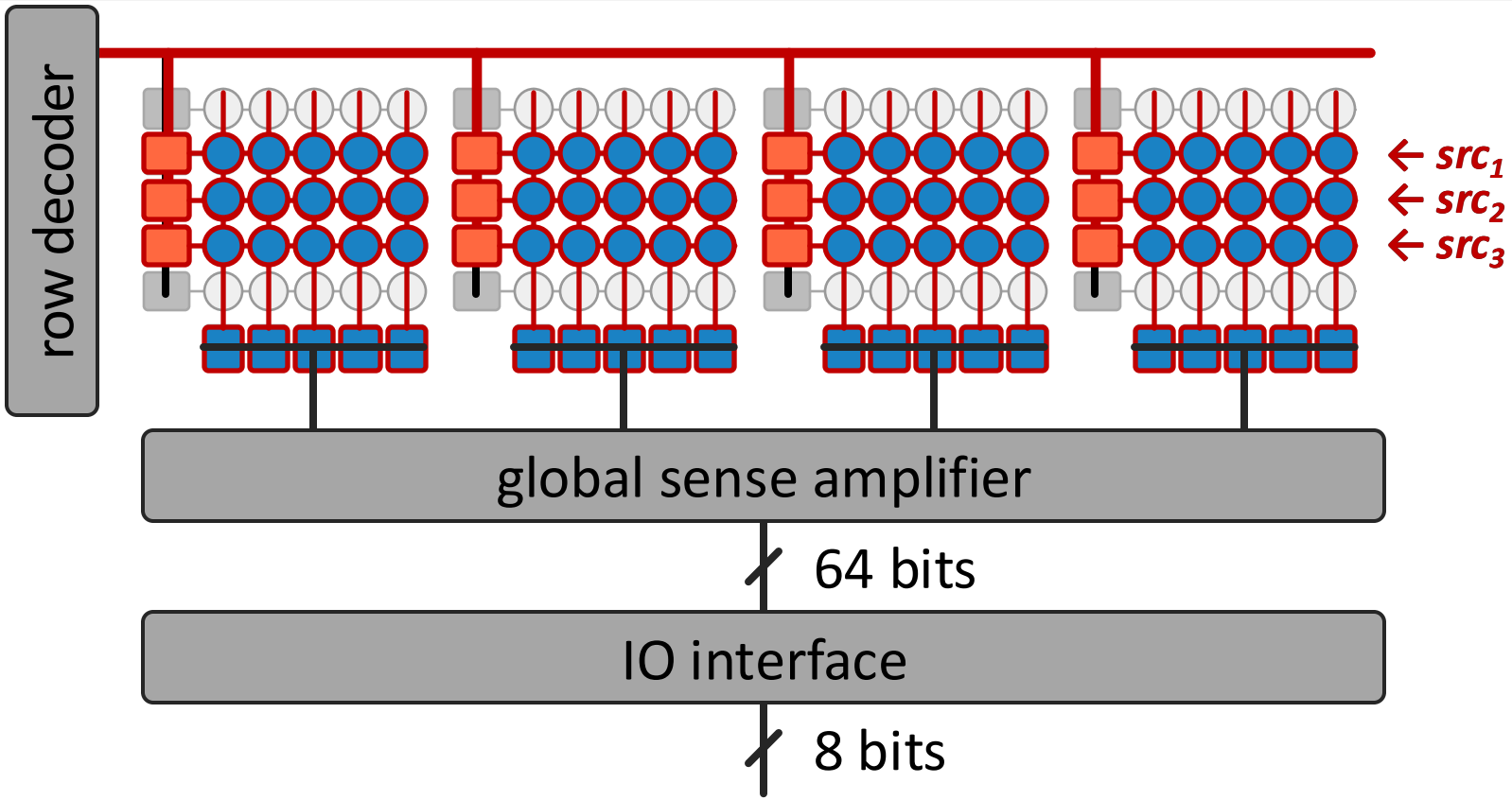
- Vivek Seshadri, Yoongu Kim, Chris Fallin, Donghyuk Lee, Rachata Ausavarungnirun, Gennady Pekhimenko, Yixin Luo, Onur Mutlu, Michael A. Kozuch, Phillip B. Gibbons, and Todd C. Mowry,
  "RowClone: Fast and Energy-Efficient In-DRAM Bulk Data Copy and Initialization"
  Proceedings of the 46th International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

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

Background:
In-DRAM Majority Operations

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

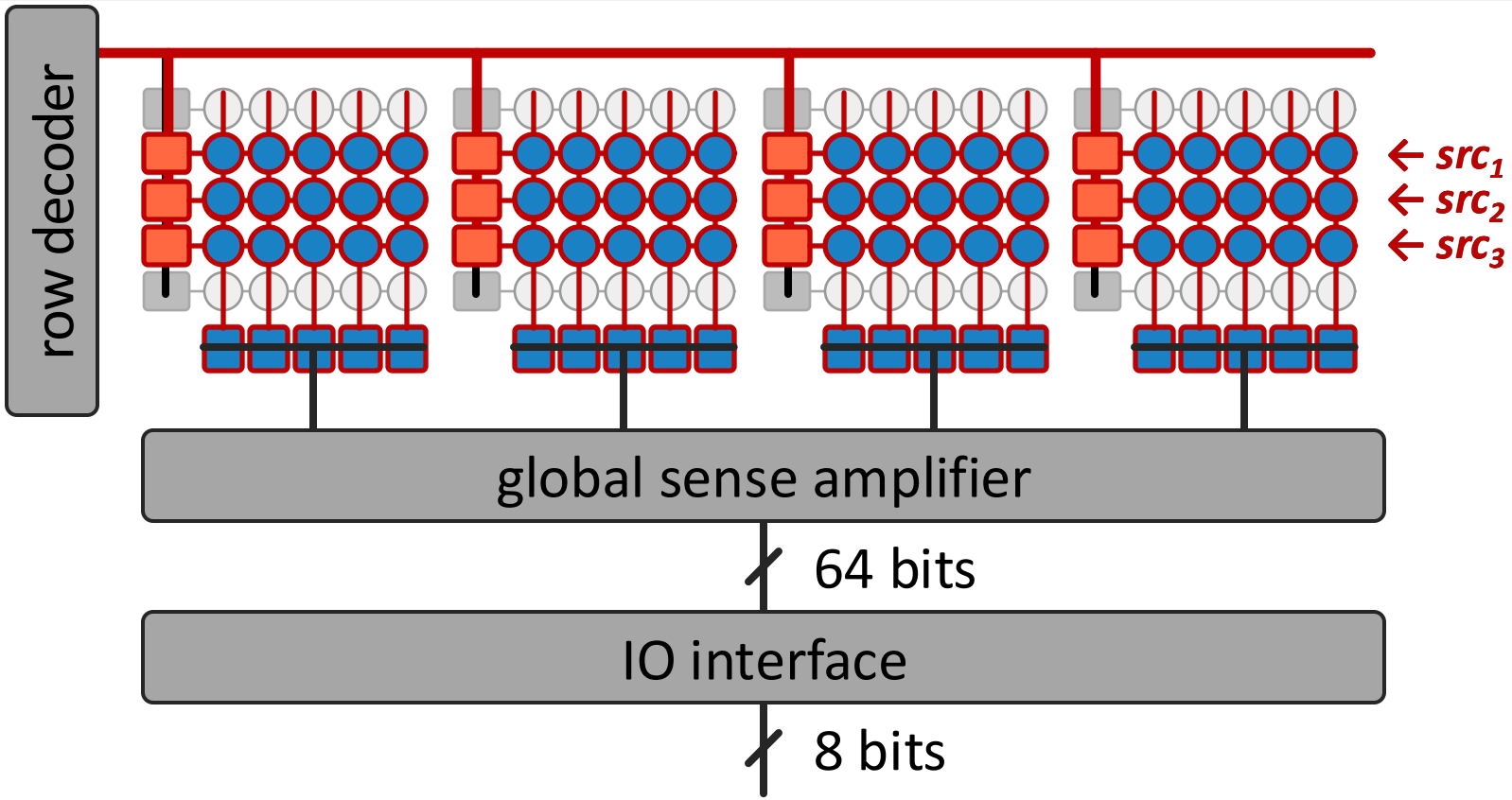


Seshadri, Vivek, et al. " **Ambit: In-Memory Accelerator for Bulk Bitwise Operations Using Commodity DRAM Technology**," in MICRO, 2017

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

In-DRAM NOT: Dual Contact Cell

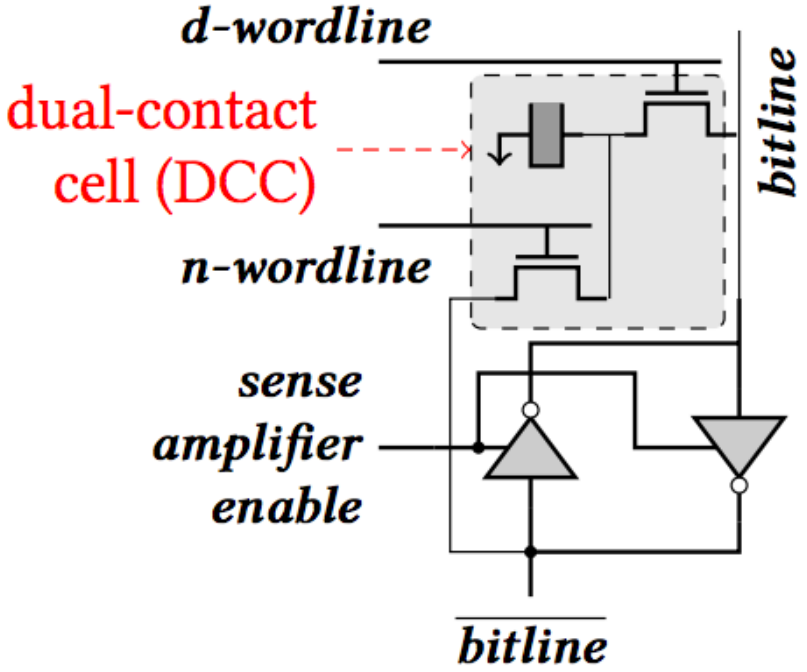


Figure 5: A dual-contact cell connected to both ends of a sense amplifier

Idea:

Feed the negated value in the sense amplifier into a special row

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

NOT in PUD: Alternative I

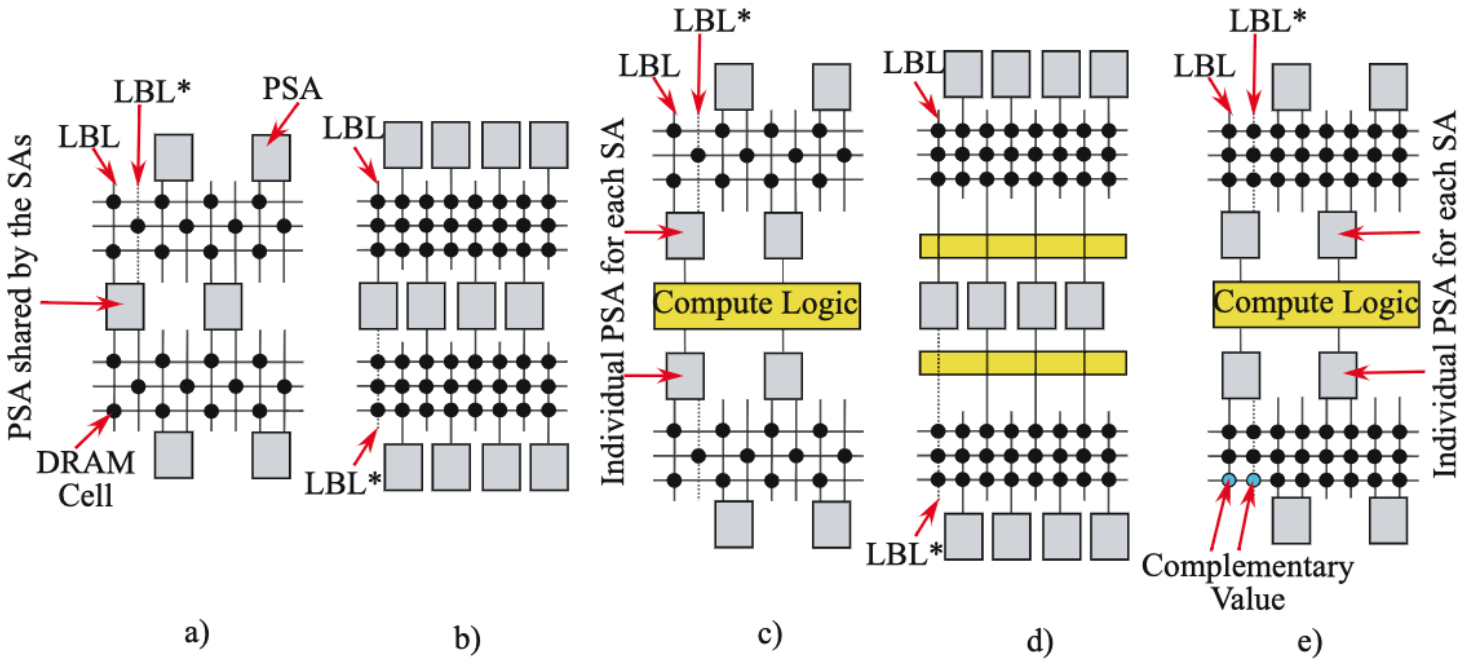


Fig. 7. Comparing DRAM BitLine array architectures from PIM perspective: a) Conventional Folded BitLine b) Conventional Open BitLine c) Folded BitLine with new compute logic d) Open BitLine with new compute logic e) Quasi BitLine (QBL) with new compute logic. QBL elimination the PSA sharing and enabling integration of large computation unit in the PSA region.

Sudarshan, Chirag, et al. "**A Critical Assessment of DRAM Architectures-Trends, Challenges and Solutions**," in SAMOS, 2022.

NOT in PUD: Alternative II

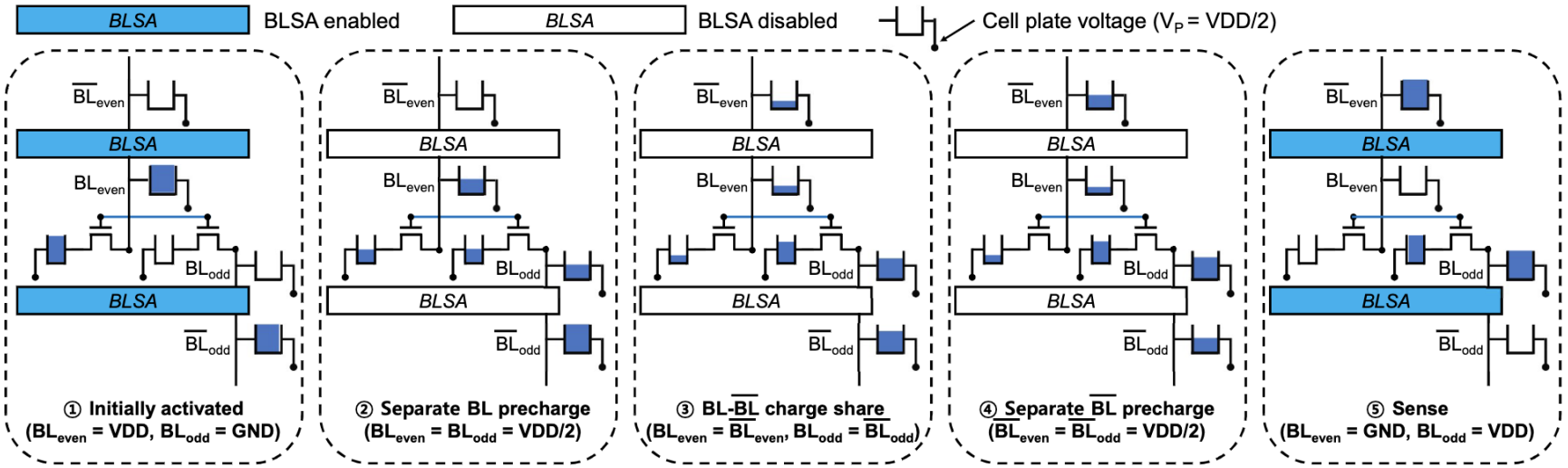
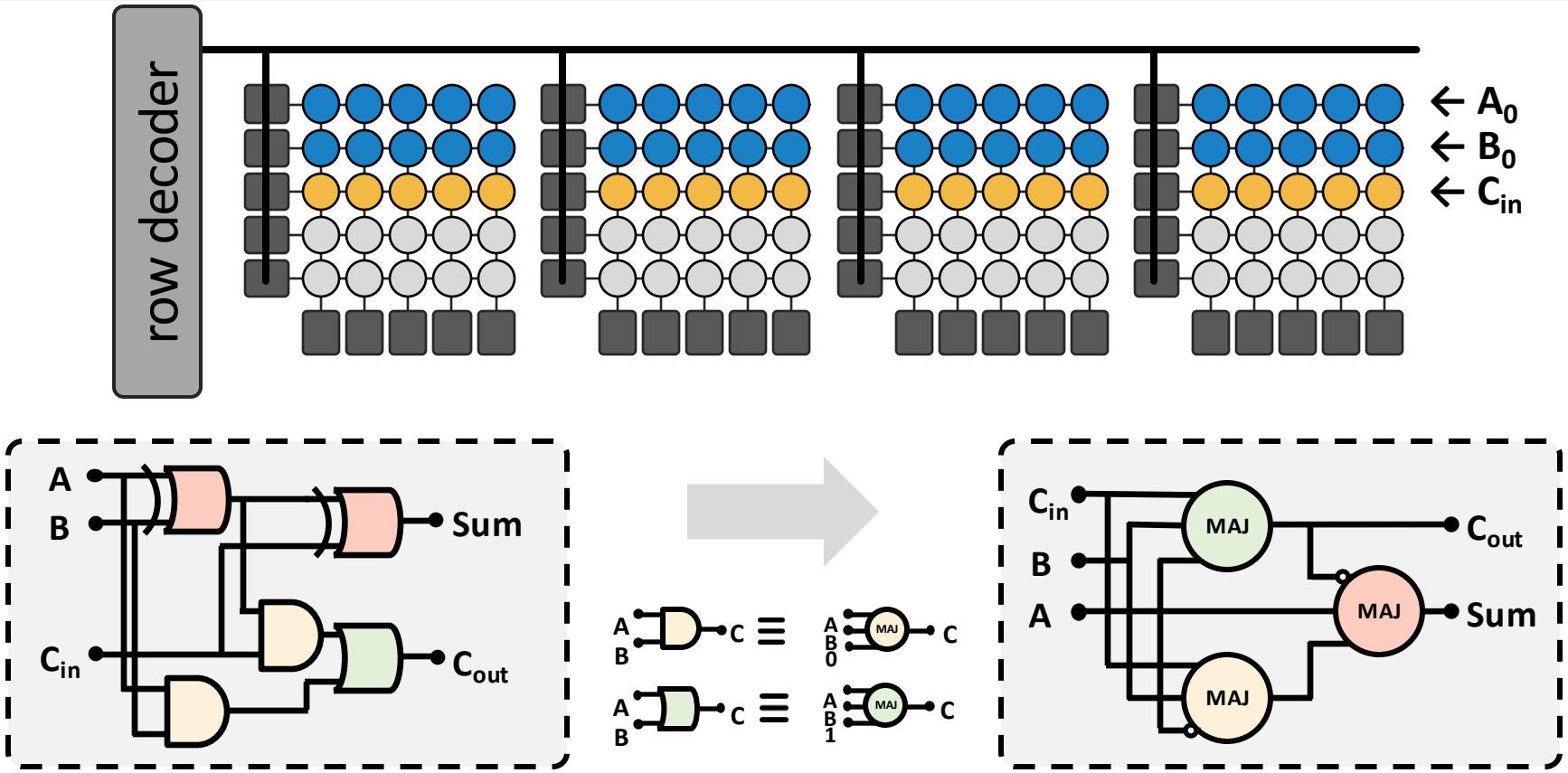


Figure 5: NOT operation sequence from initial state to inverted state

Shin, Hoon, Rihae Park, and Jae W. Lee.
"**A Processing-using-Memory Architecture for Commodity DRAM Devices with Enhanced Compatibility and Reliability**," in *ICCAD*, 2024.

Background:
Bulk Bitwise Arithmetic Operations (I)

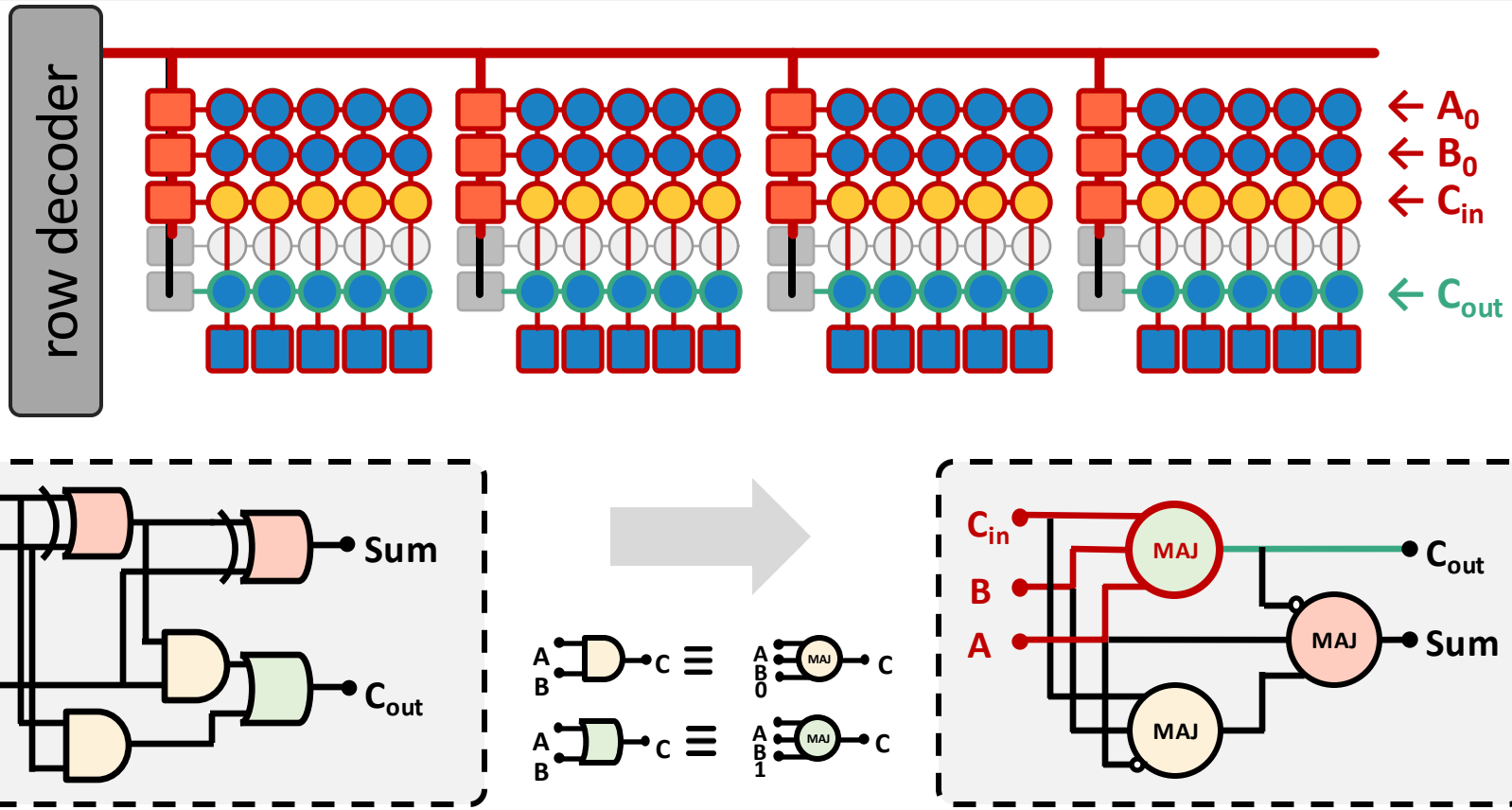
Bulk bitwise arithmetic can be performed by
orchestrating in-DRAM row copy and majority operations



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 (I)

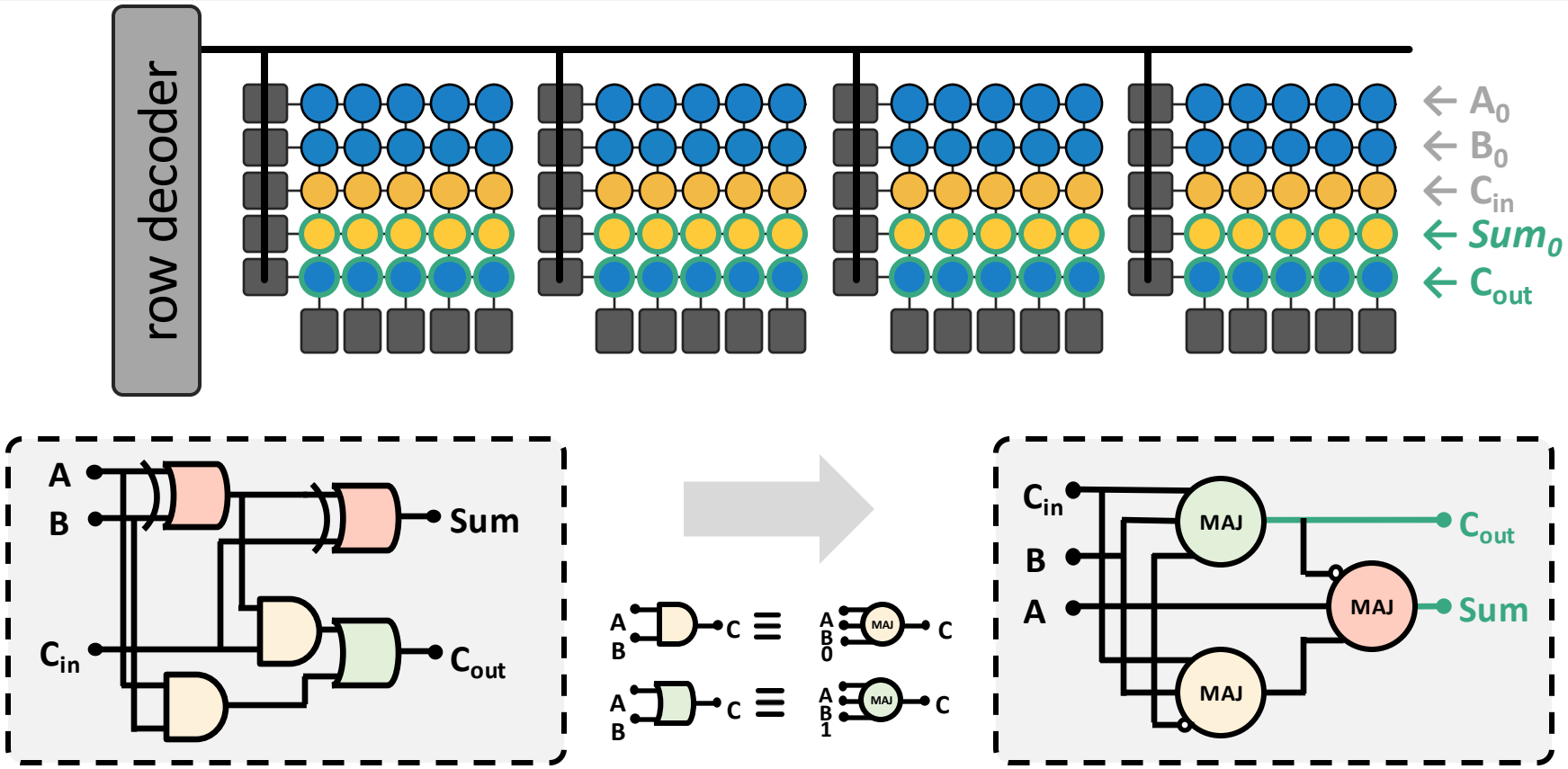
Bulk bitwise arithmetic can be performed by
orchestrating in-DRAM row copy and majority operations



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 (I)

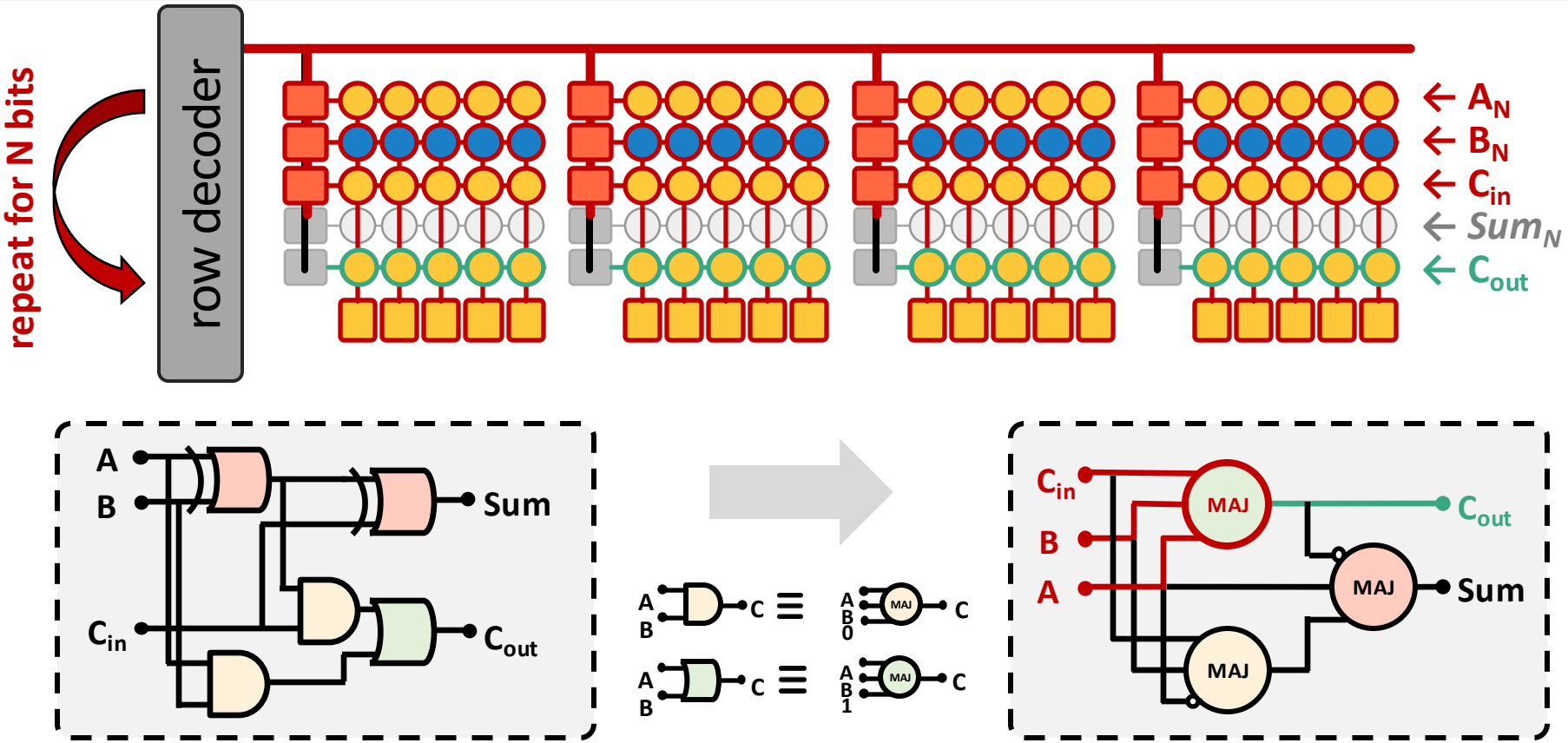
Bulk bitwise arithmetic can be performed by
orchestrating in-DRAM row copy and majority operations



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 (II)

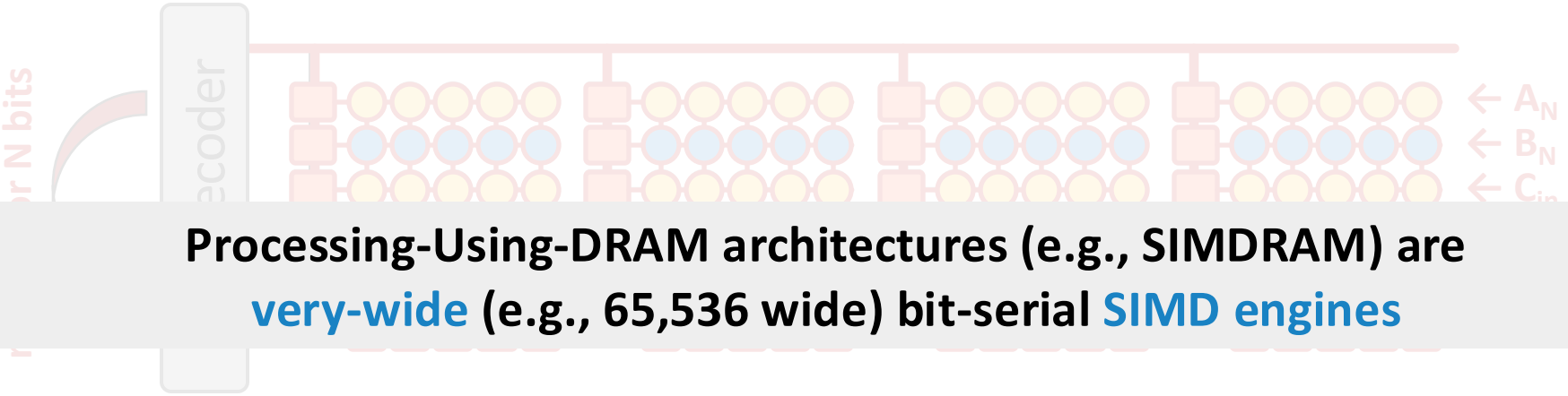
Bulk bitwise arithmetic can be performed by
orchestrating in-DRAM row copy and majority operations



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 (II)

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

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

Processing-in-Memory: Challenges

Adopting PIM as a mainstream architecture holistically and efficiently is still very challenging due to a lack of:

1. **Workload characterization methodologies and benchmark suites targeting PIM architectures**
2. **Execution paradigms that can map applications effectively onto PIM hardware resources**
3. **Compiler support and programming frameworks targeting PIM architectures**
4. **Adaptive data-aware runtime mechanisms that use the intrinsic properties of data for efficient PIM execution**

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

Processing-in-Memory: Challenges

Adopting PIM as a mainstream architecture holistically and efficiently is still very challenging due to a lack of:

1. Workload characterization methodologies and benchmark suites targeting PIM architectures
2. **Execution paradigms** that can map applications effectively onto PIM hardware resources
3. **Compiler support** and **programming frameworks** targeting PIM architectures
4. Adaptive data-aware runtime mechanisms that use the intrinsic properties of data for efficient PIM execution

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

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

Problem & Goal

Problem

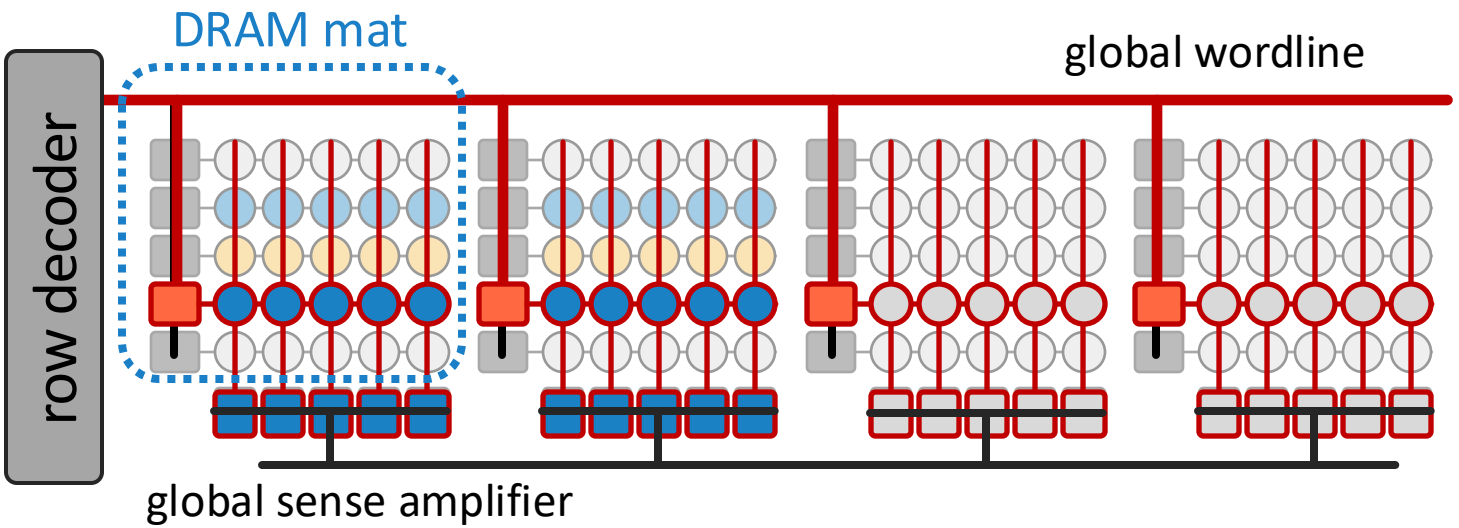
Processing-Using-DRAM's large and rigid granularity limits its applicability and efficiency for different applications

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:

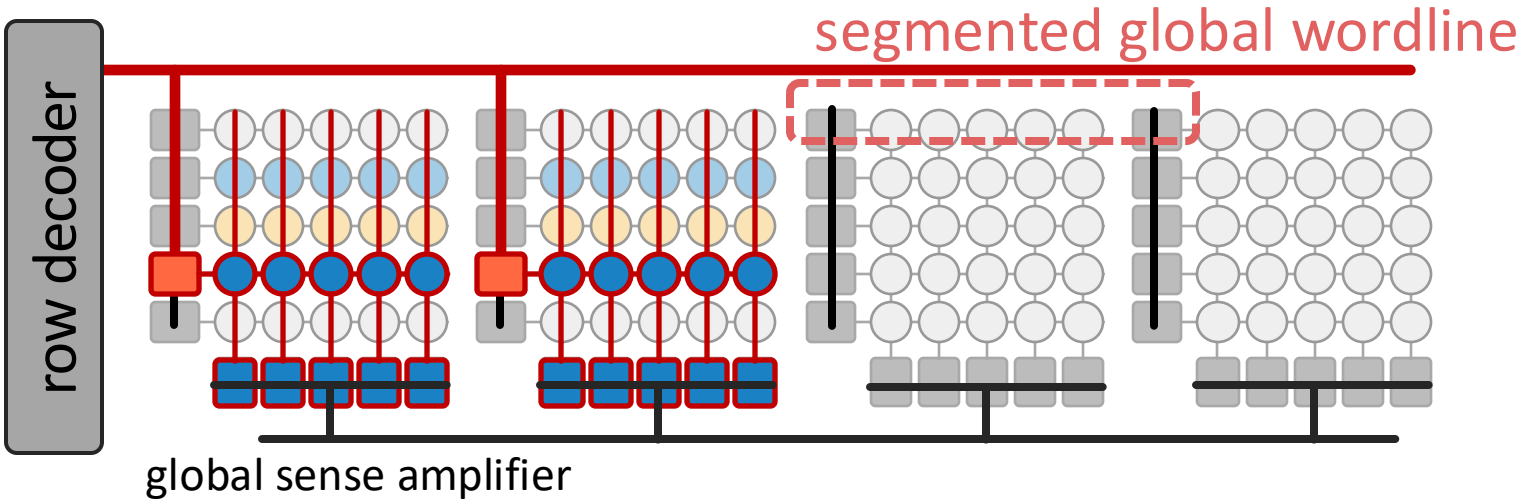
on a DRAM access, the global wordline propagates across all DRAM mats

Fine-Grained DRAM:

segments the global wordline to access individual DRAM mats

MIMDRAM:
Key Idea (II)

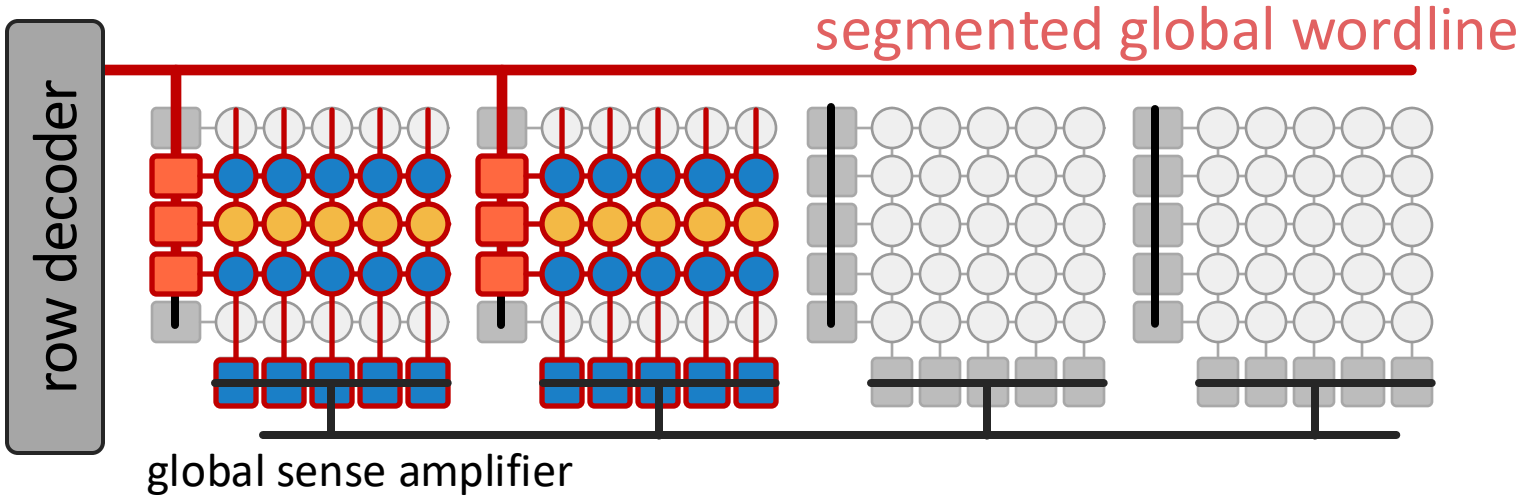
Fine-Grained DRAM:
segments the global wordline to access individual DRAM mats



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)

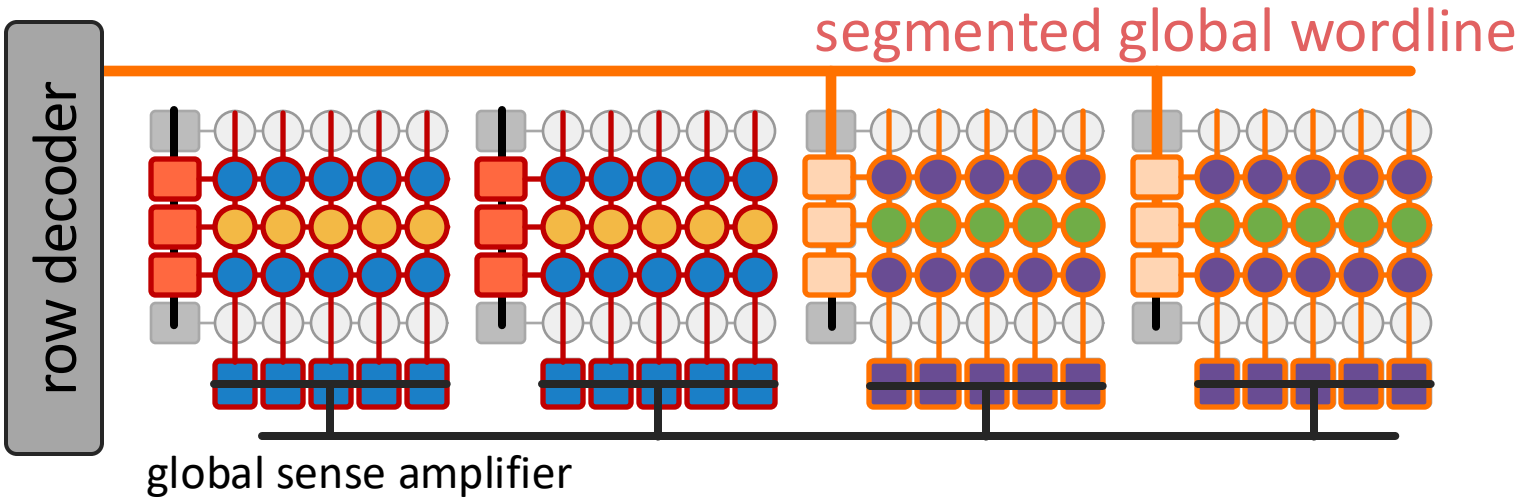


Fine-grained DRAM for processing-using-DRAM:

1 Improves SIMD utilization

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

MIMDRAM:
Key Idea (III)

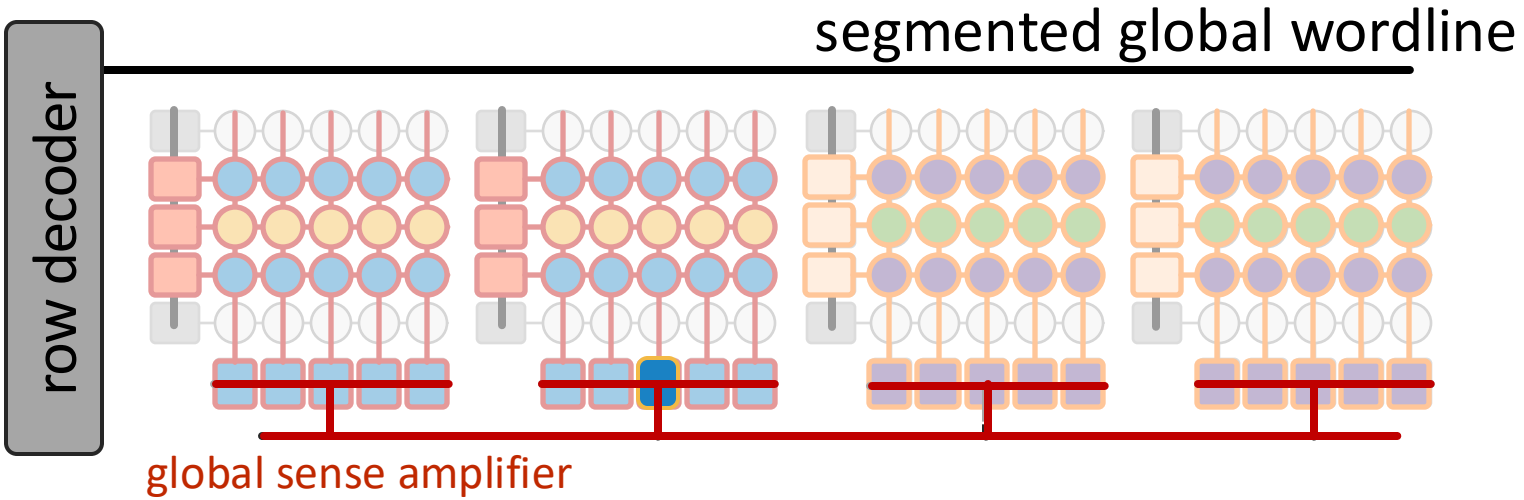


Fine-grained DRAM for processing-using-DRAM:

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:

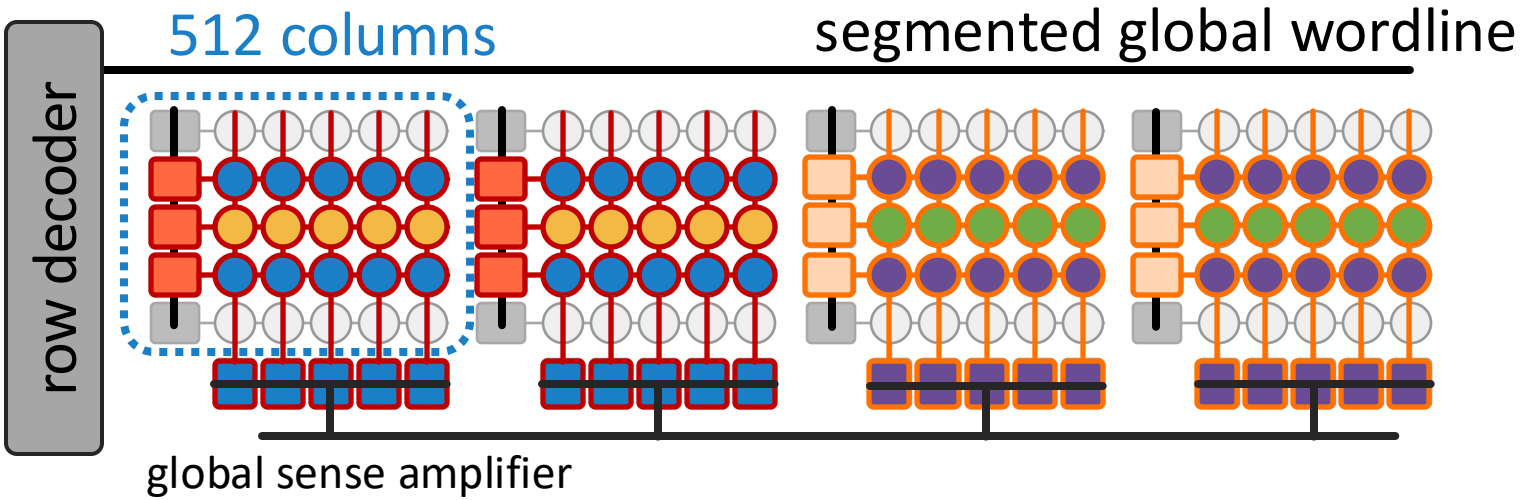
1 Improves SIMD utilization

- 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:
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:

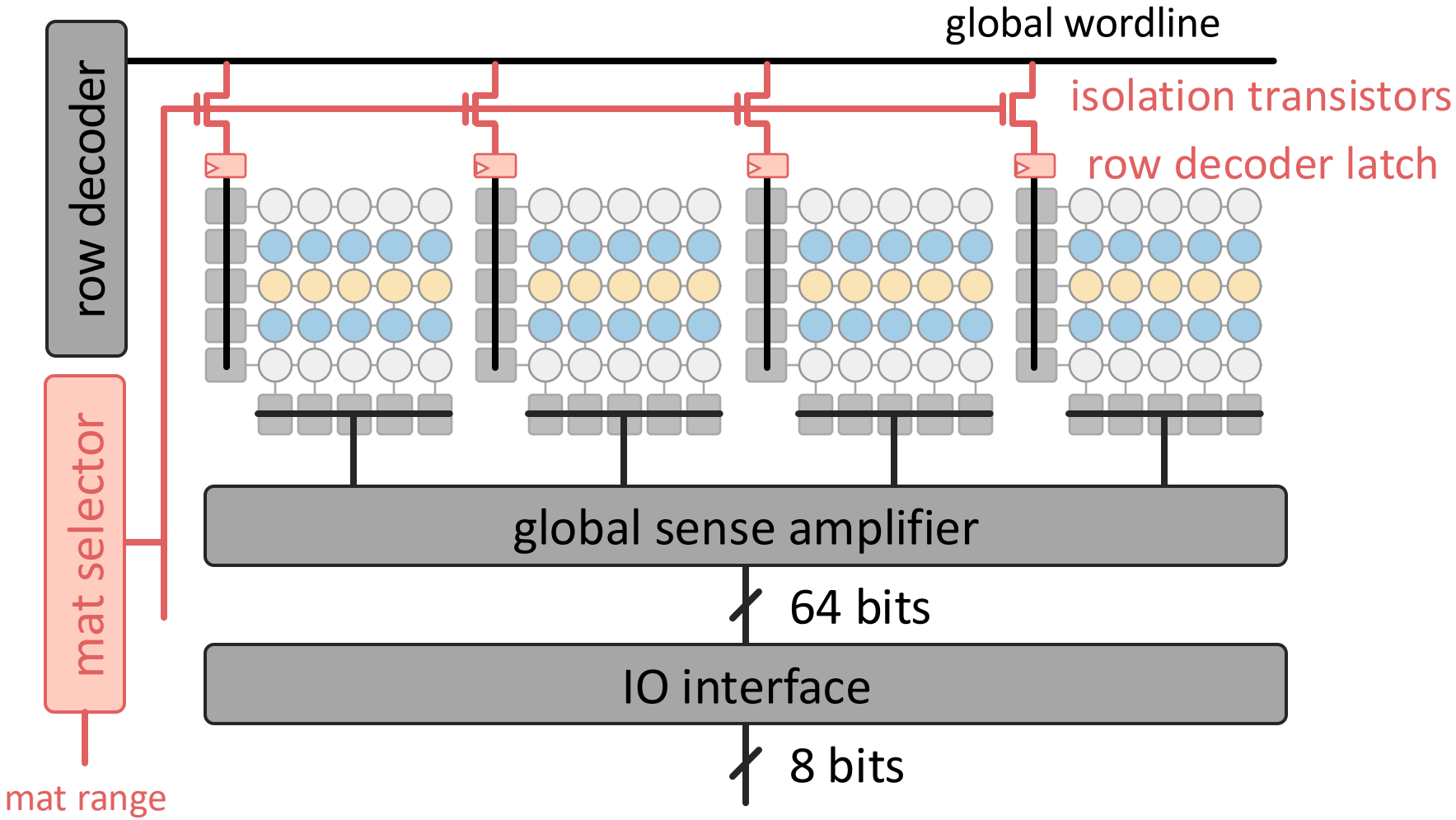
1 Hardware

- **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

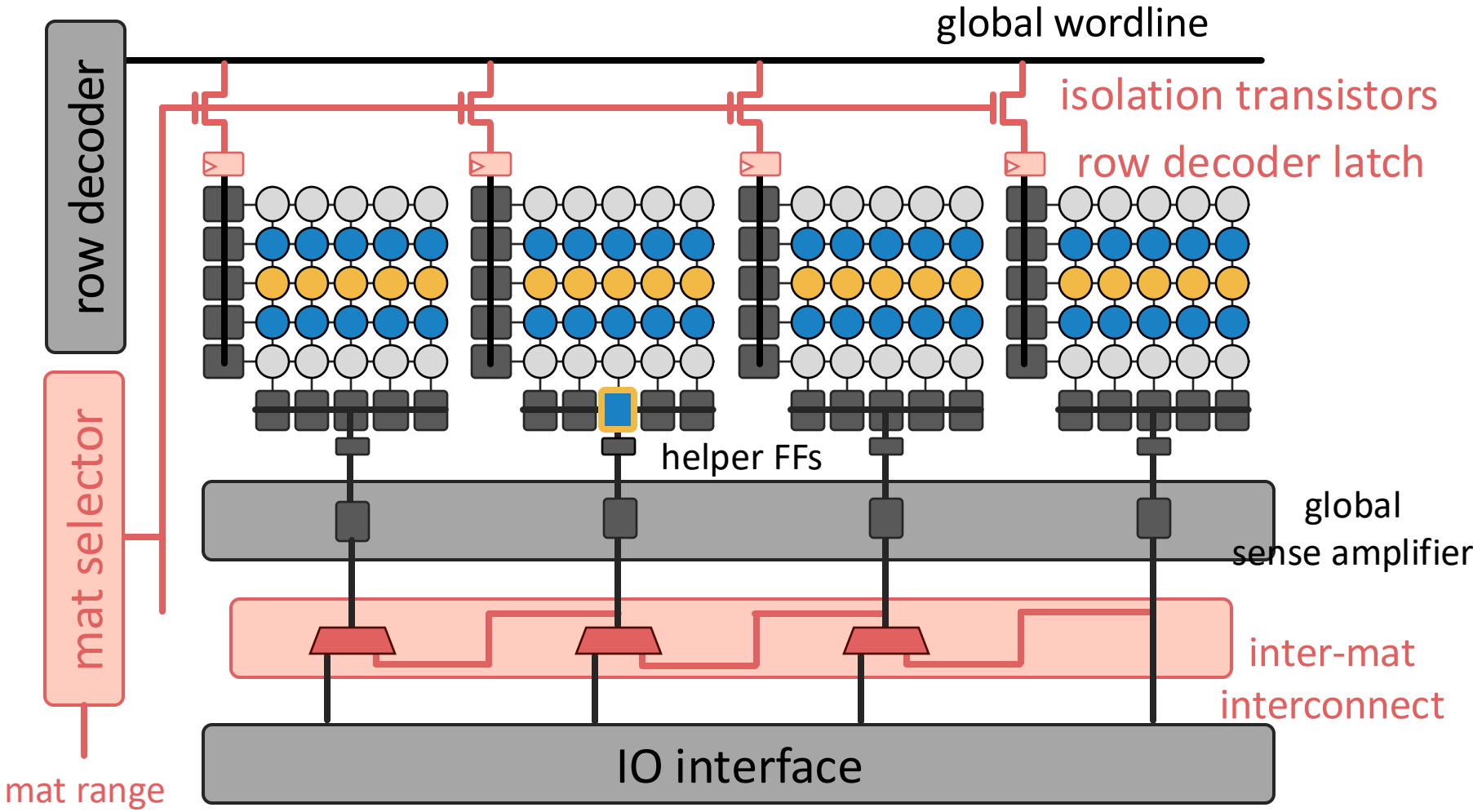
2 Software

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

MIMDRAM:
DRAM Array Modifications

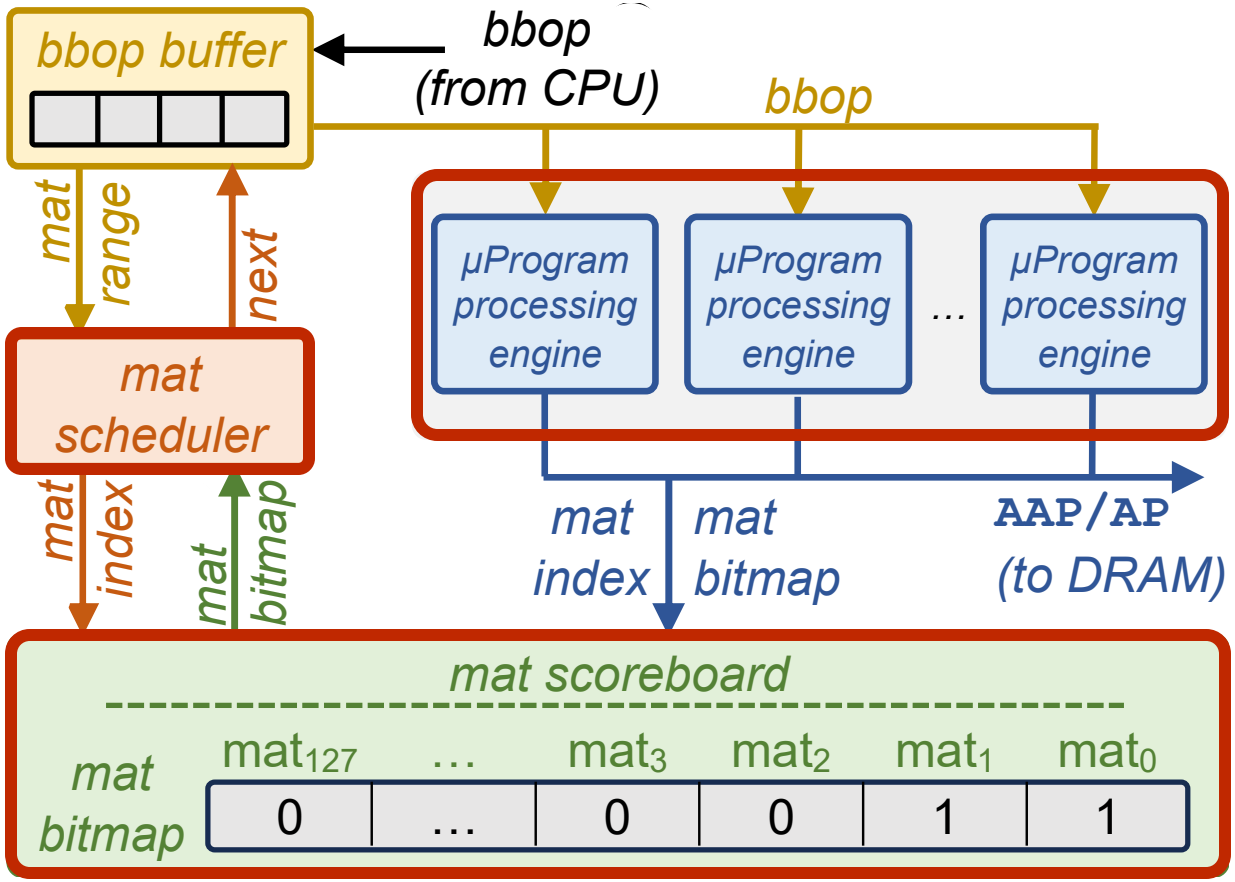


MIMDRAM:
Moving Data Across Mats



MIMDRAM:
Control Unit

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



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

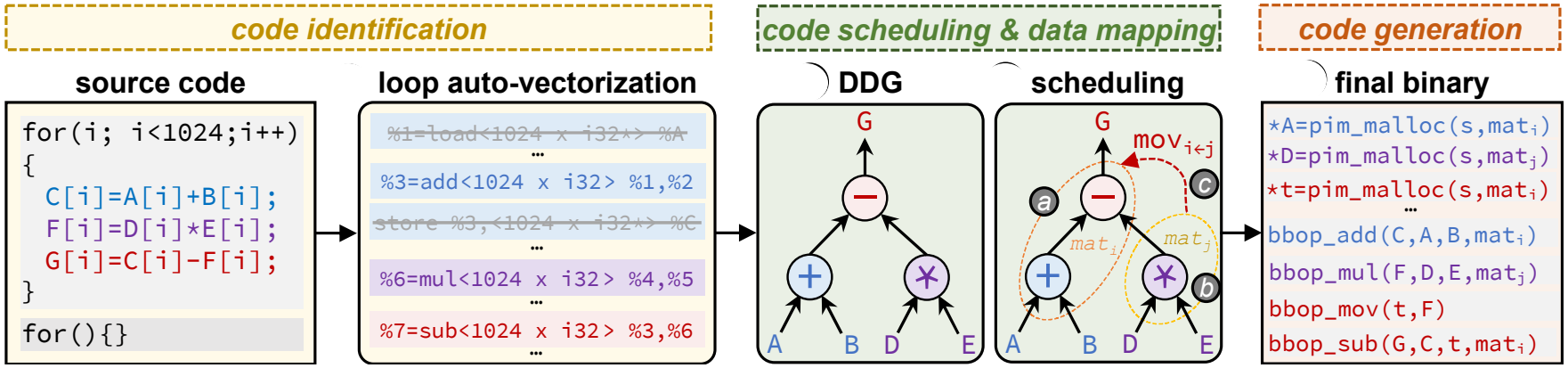
2 Software
- **new compiler support to transparently generate PUD instructions**
- system support to map and execute PUD instructions

MIMDRAM:
Compiler Support (I)

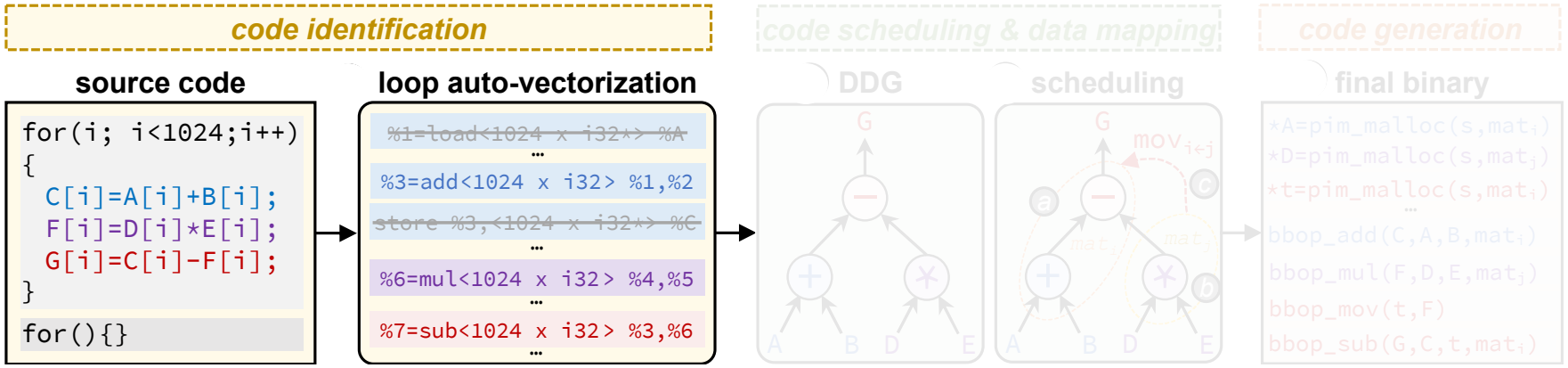
Goal

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

Three new LLVM-based passes targeting PUD execution



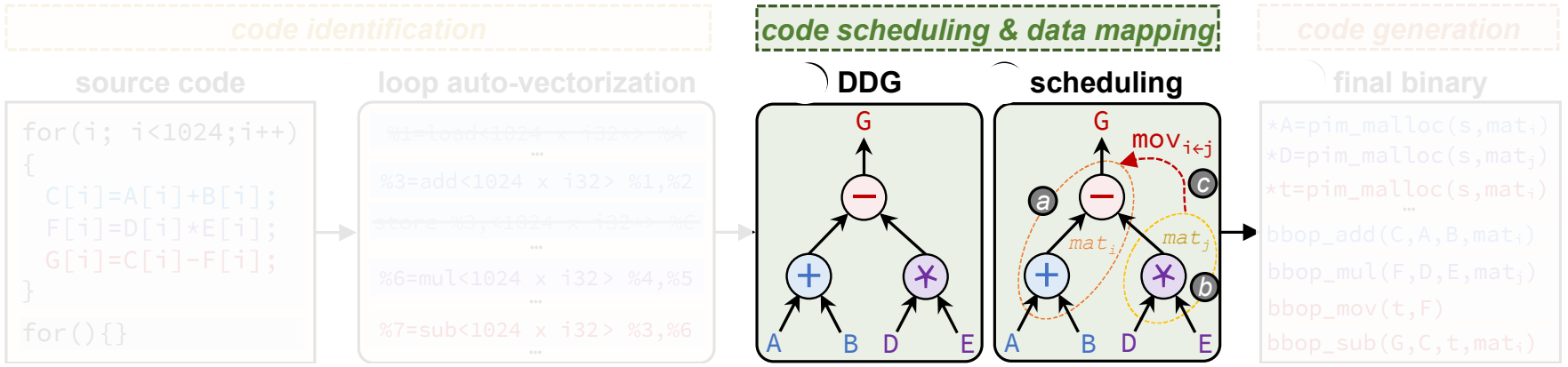
MIMDRAM:
Compiler Support (II)



Goal

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

MIMDRAM:
Compiler Support (II)



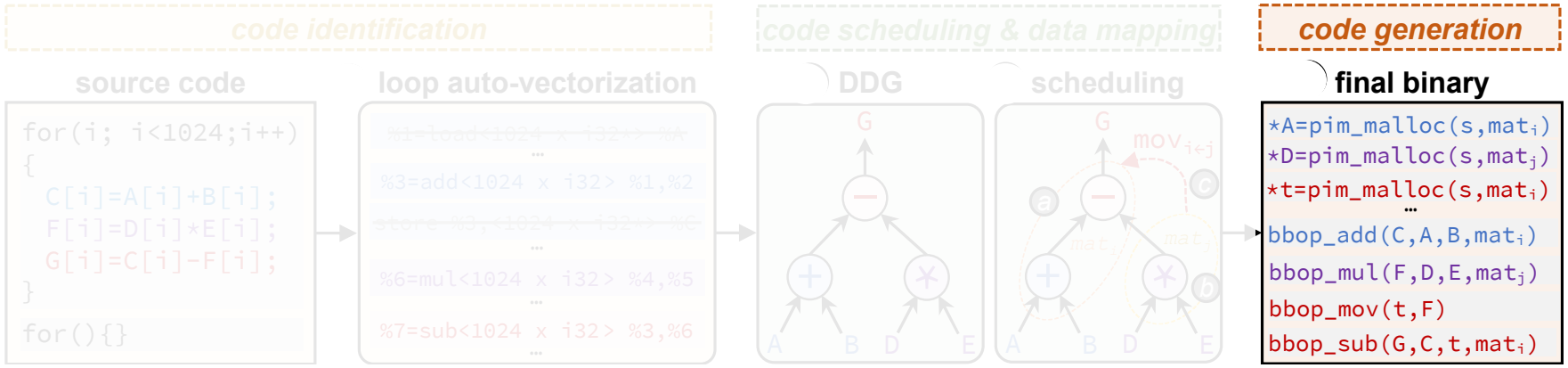
Goal

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:
System Support

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

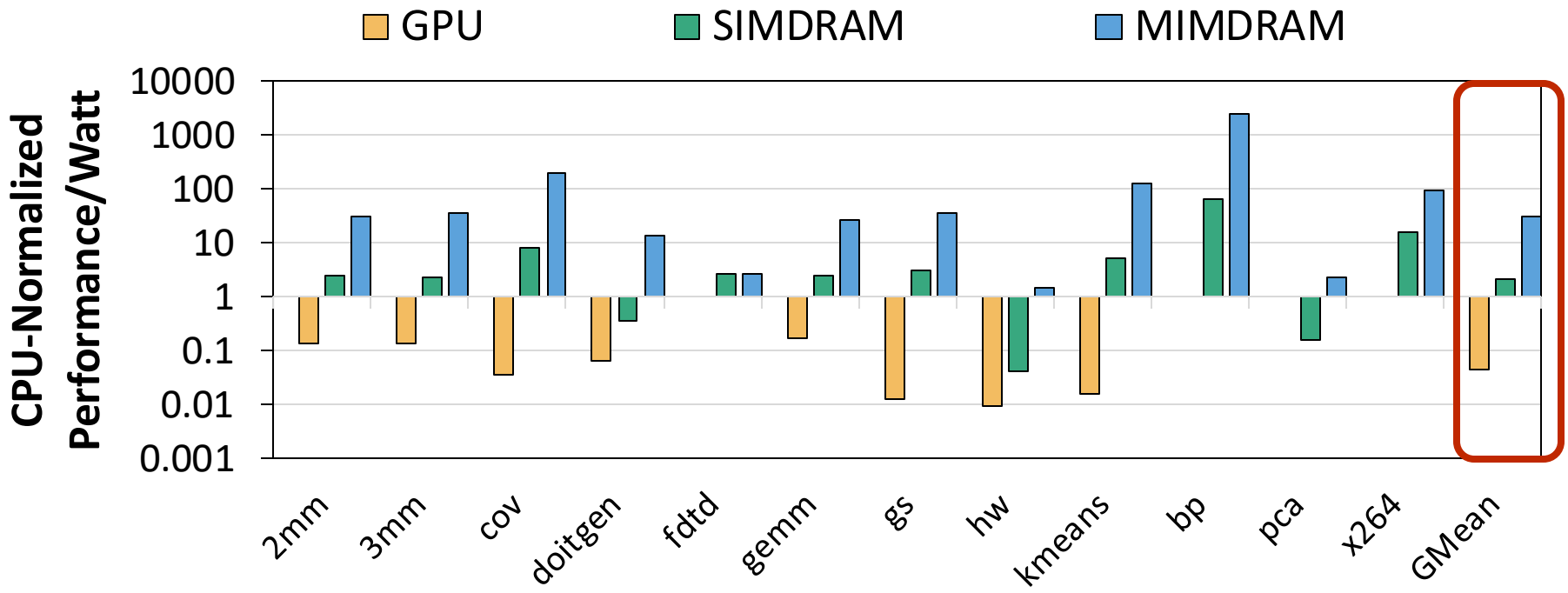
Evaluation:
Methodology Overview

- **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

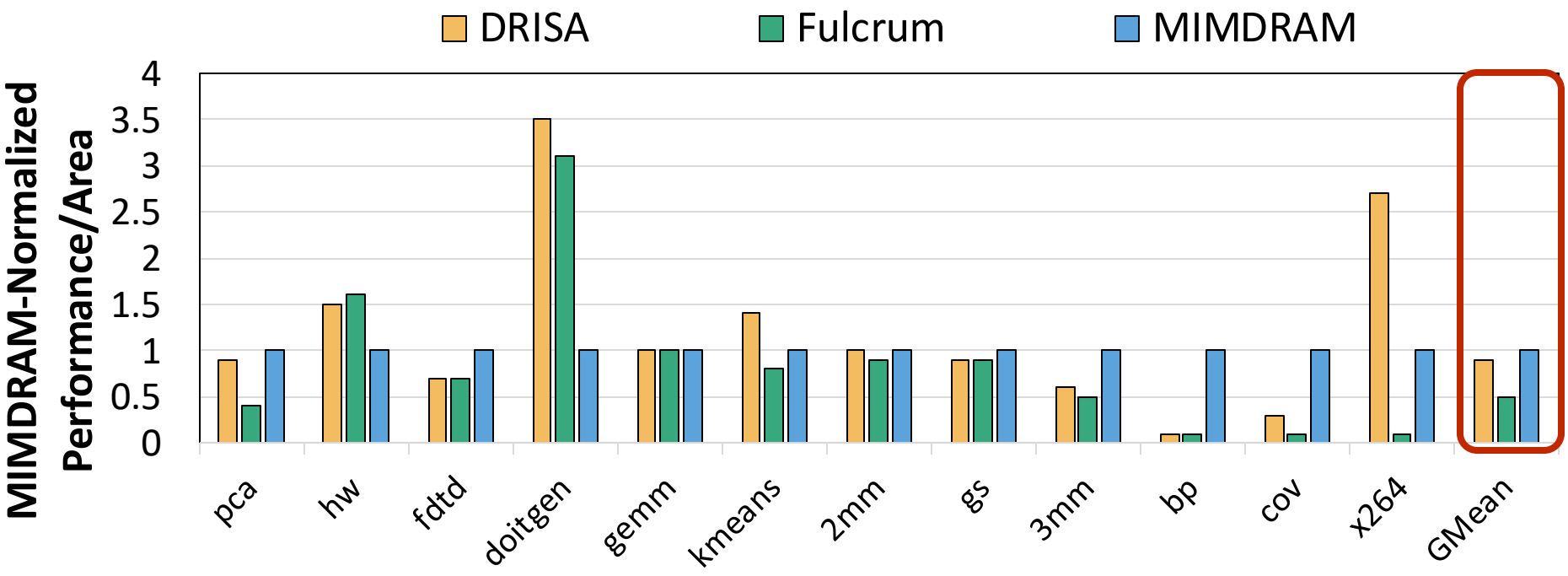
Evaluation:
Single Application Analysis – Energy Efficiency



Takeaway

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

Evaluation:
Comparison to Other PIM Architectures



Takeaway

MIMDRAM significantly improves performance/area compared to DRISA (1.18x) and Fulcrum (1.92x)

Processing-in-Memory: Challenges

Adopting PIM as a mainstream architecture holistically and efficiently is still very challenging due to a lack of:

1. Workload characterization methodologies and benchmark suites targeting PIM architectures
2. Execution paradigms that can map applications effectively onto PIM hardware resources
3. Compiler support and programming frameworks targeting PIM architectures
4. **Adaptive data-aware runtime mechanisms that use the intrinsic properties of data for efficient PIM execution**

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

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 Rigid and Static Data Representation
- leading to a subpar performance in the presence of narrow values
- opportunity 1: narrow values for PUD computation

2 Throughput-Oriented Execution with Low Latency Tolerance
- failing to reduce the latency of a single PUD operation
- opportunity 2: DRAM parallelism for latency-oriented execution

3 Scalability Challenges for High-Precision Operations
- due to the linear/quadratically scaling nature of bit-serial algorithms
- opportunity 3: alternative data representation for high-precision computation

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 Rigid and Static Data Representation
- leading to a subpar performance in the presence of narrow values
- opportunity 1: narrow values for PUD computation

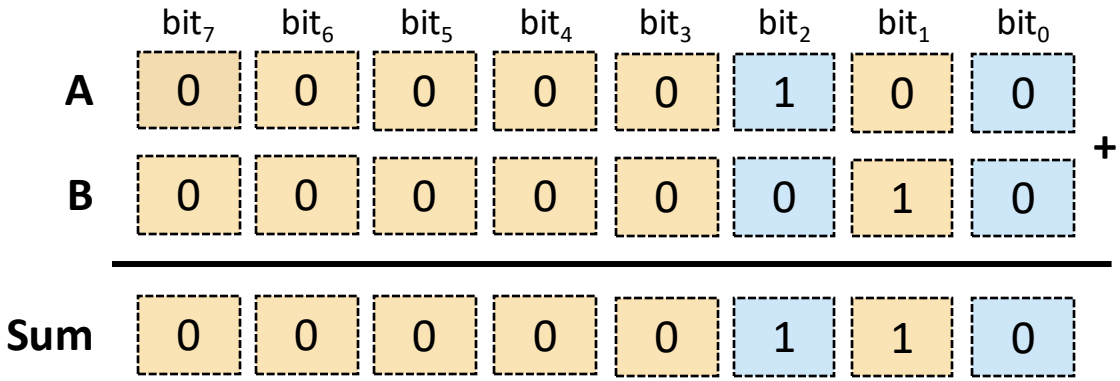
2 Throughput-Oriented Execution with Low Latency Tolerance
- failing to reduce the latency of a single PUD operation
- opportunity 2: DRAM parallelism for latency-oriented execution

3 Scalability Challenges for High-Precision Operations
- due to the linear/quadratically scaling nature of bit-serial algorithms
- opportunity 3: alternative data representation for high-precision computation

Limitations of Bit-Serial PUD Systems:
Rigid and Static Data Representation (I)

Narrow Values:

data with small dynamic range stored in large data formats



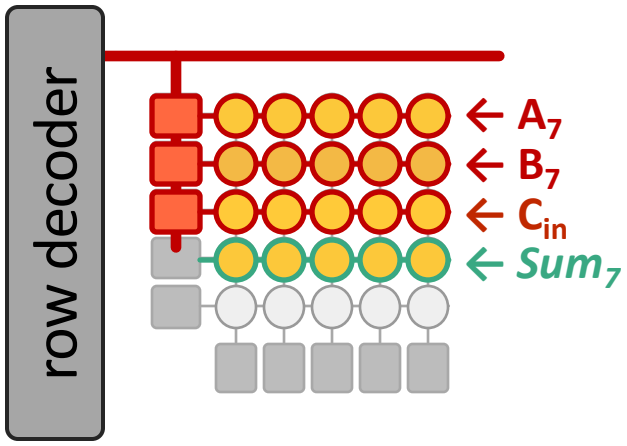
Limitations of Bit-Serial PUD Systems:
Rigid and Static Data Representation (I)

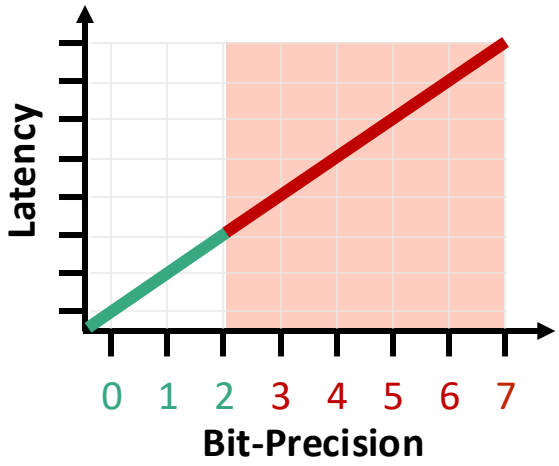
Narrow Values:

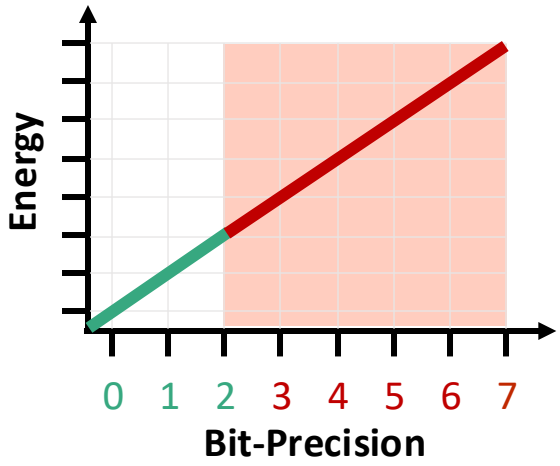
data with small dynamic range stored in large data formats

	bit_7	bit_6	bit_5	bit_4	bit_3	bit_2	bit_1	bit_0	
A	0	0	0	0	0	1	0	0	+
B	0	0	0	0	0	0	1	0	
Sum	0	0	0	0	0	1	1	0	

inconsequential bits → do not contribute to the output's dynamic range



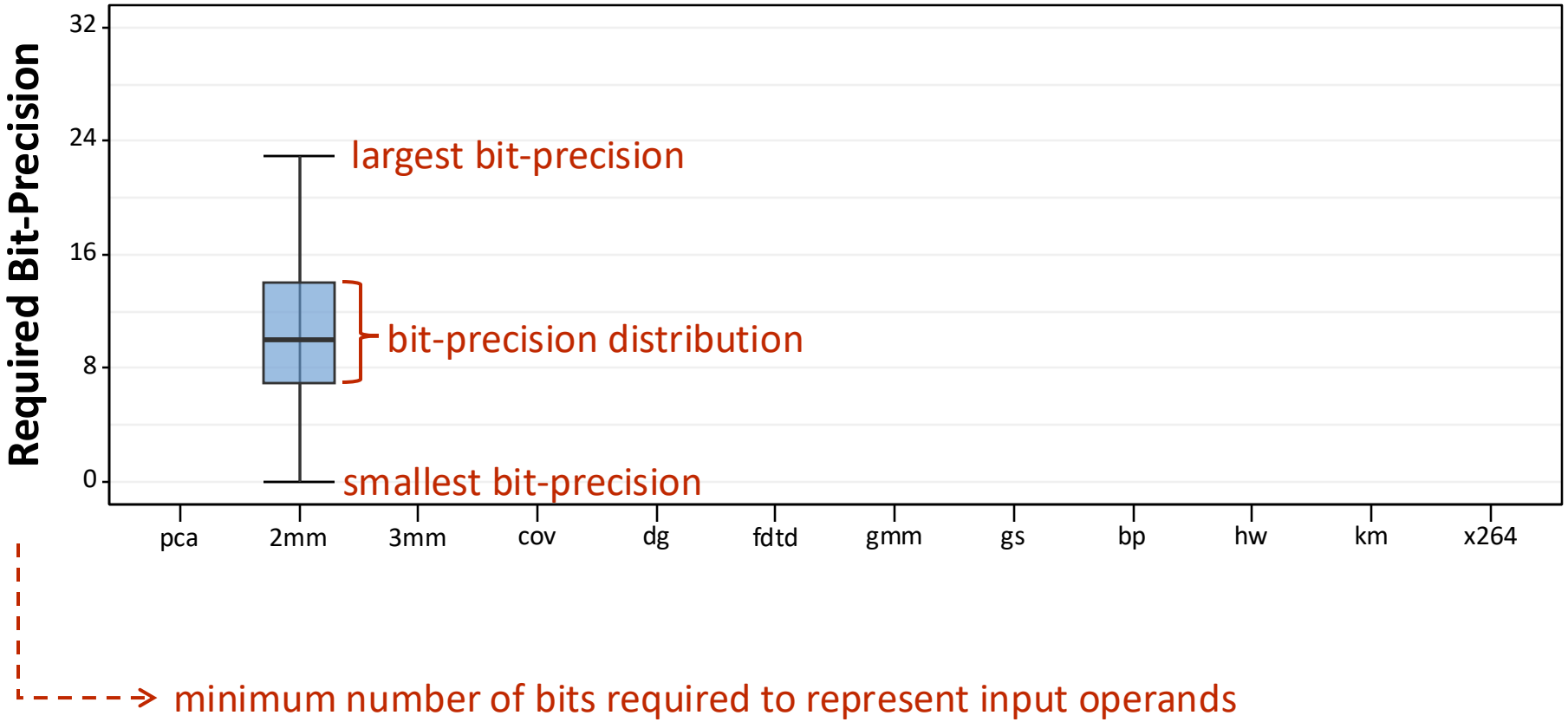




Limitations of Bit-Serial PUD Systems:
Rigid and Static Data Representation (II)

Application Analysis:

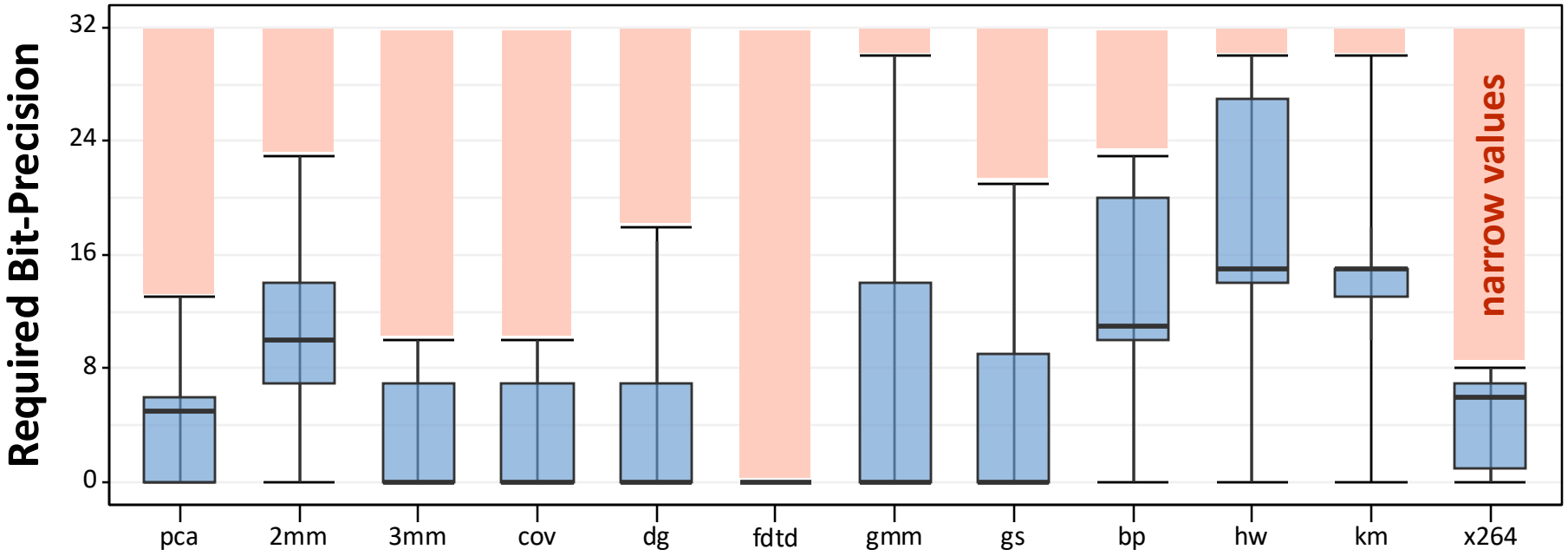
quantify the amount required bit-precision in real applications



Limitations of Bit-Serial PUD Systems:
Rigid and Static Data Representation (II)

Application Analysis:

quantify the amount required bit-precision in real applications



Applications display a significant amount of narrow values →
bit-precision can be reduced from 32 bits to 20 bits, on average

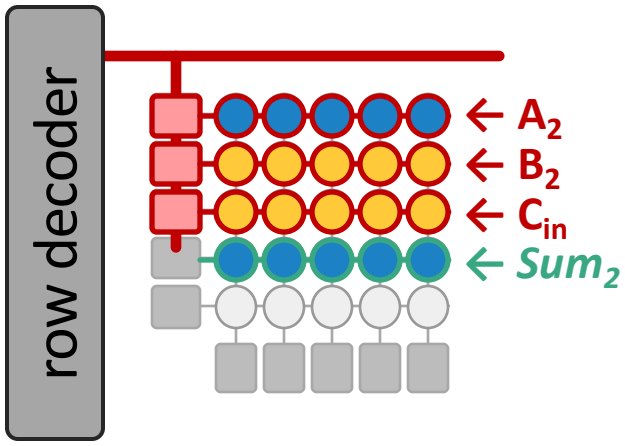
Opportunities for Bit-Serial PUD:
Narrow Values for PUD Operations

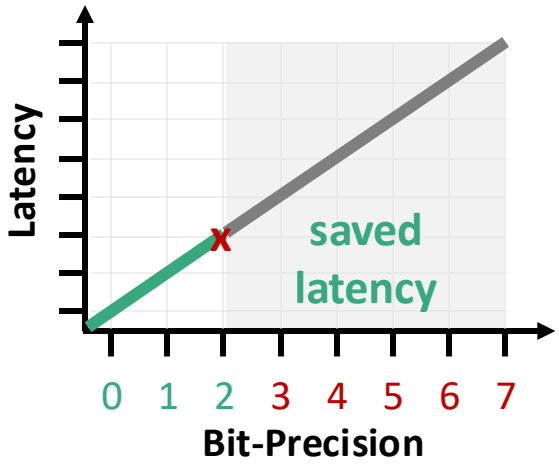
Narrow Values:

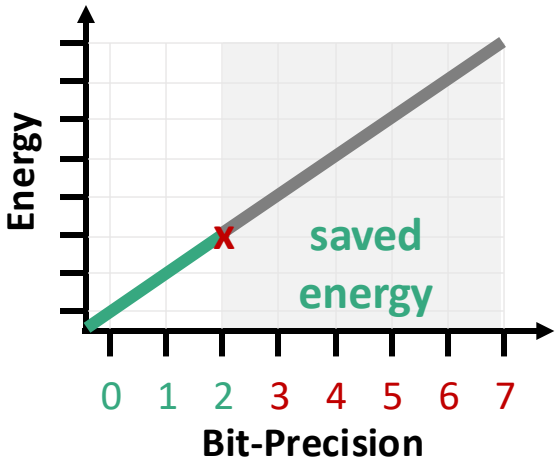
data with small dynamic range stored in large data formats

	bit_7	bit_6	bit_5	bit_4	bit_3	bit_2	bit_1	bit_0	
A	0	0	0	0	0	1	0	0	+
B	0	0	0	0	0	0	1	0	
Sum	0	0	0	0	0	1	1	0	

inconsequential bits → dynamically identify and skip during computation







Opportunities for Bit-Serial PUD: Overview

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

1 Rigid and Static Data Representation
- leading to a subpar performance in the presence of narrow values
- opportunity 1: narrow values for PUD computation

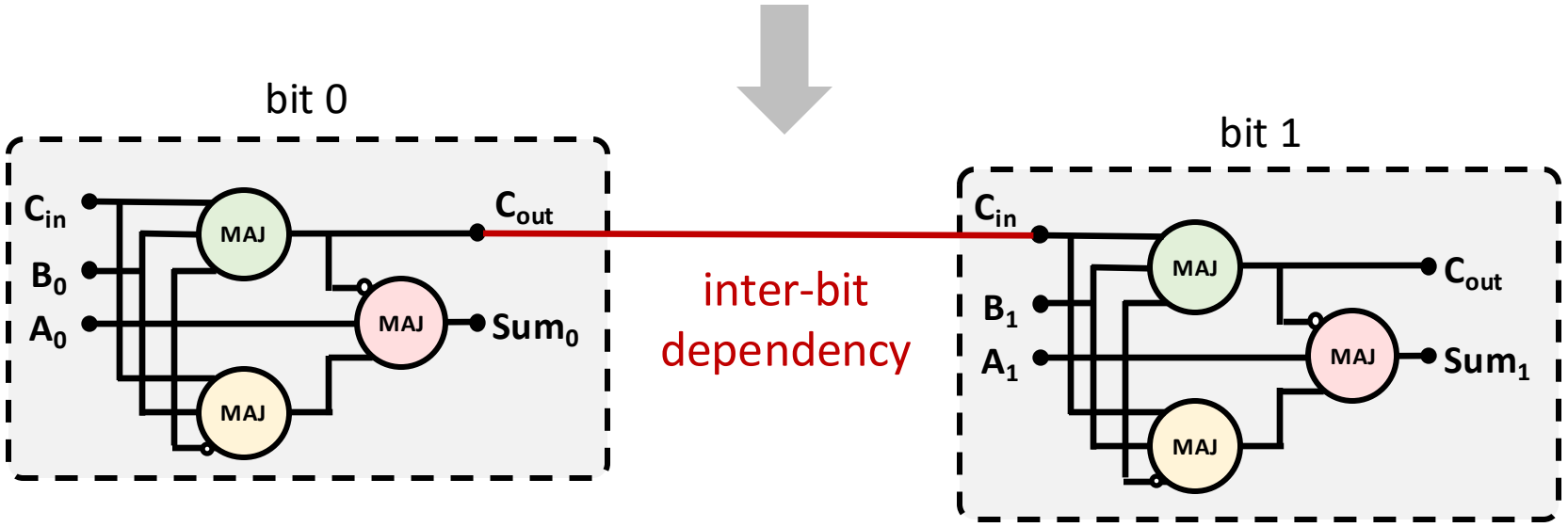
2 Throughput-Oriented Execution with Low Latency Tolerance
- failing to reduce the latency of a single PUD operation
- opportunity 2: DRAM parallelism for latency-oriented execution

3 Scalability Challenges for High-Precision Operations
- due to the linear/quadratically scaling nature of bit-serial algorithms
- opportunity 3: alternative data representation for high-precision computation

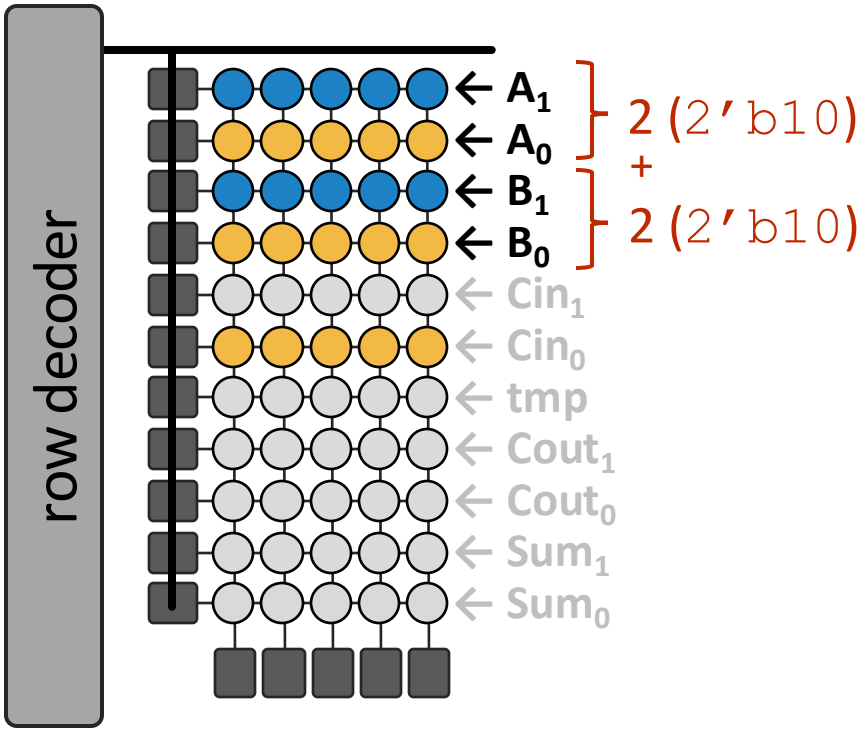
Limitations of Bit-Serial PUD Systems:
Throughput-Oriented Execution (I)

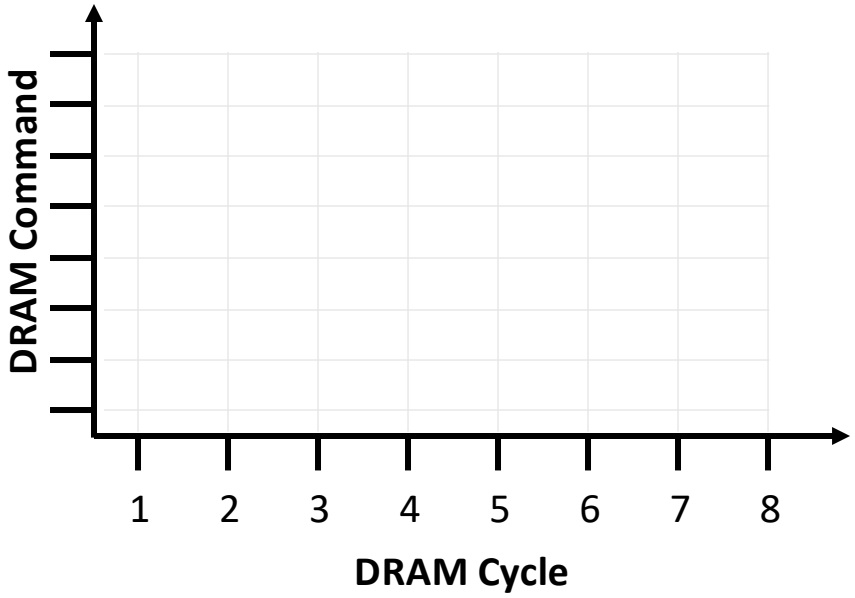
Problem

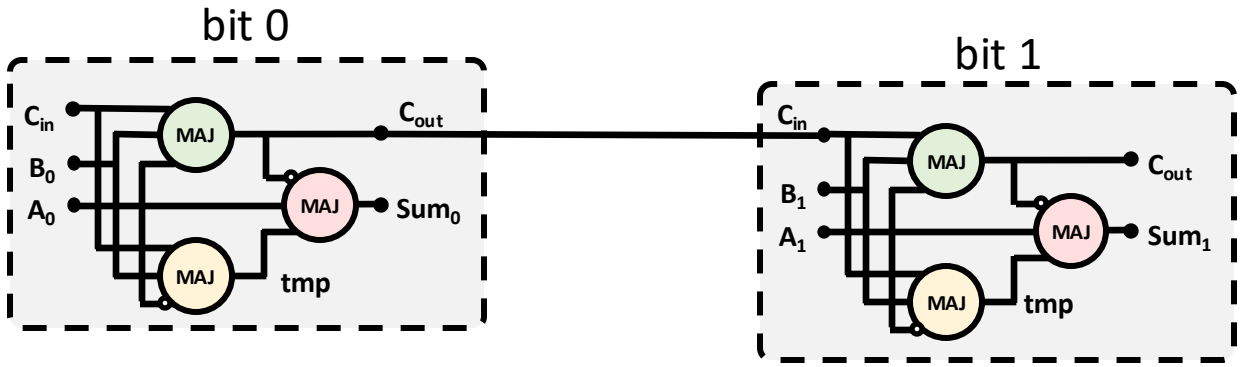
PUD operations are inherently slow
→ need to operate on each bit serially to perform an operation with a data width > 1



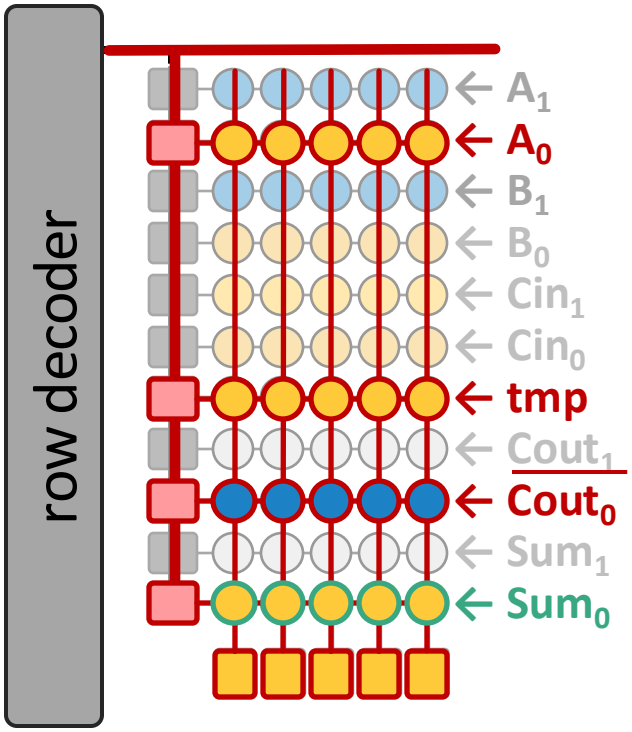
Limitations of Bit-Serial PUD Systems:
Throughput-Oriented Execution (II)

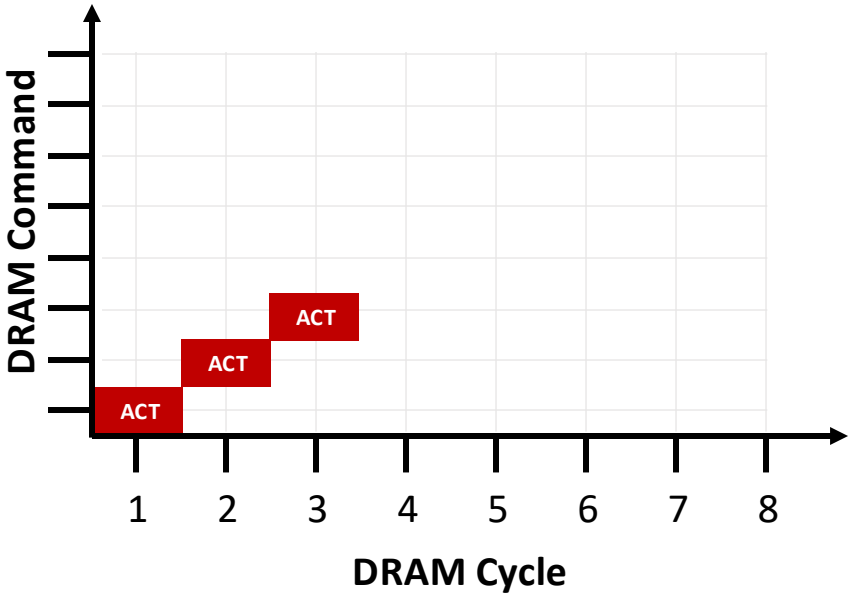


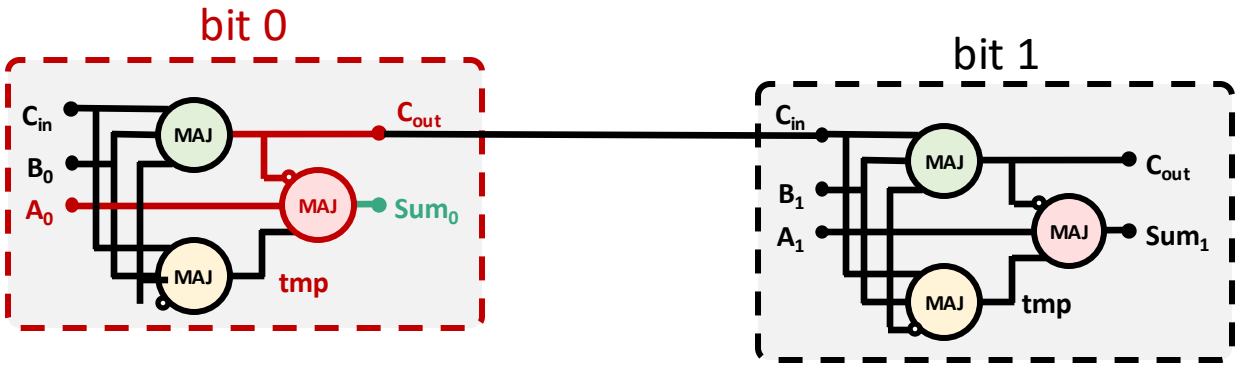




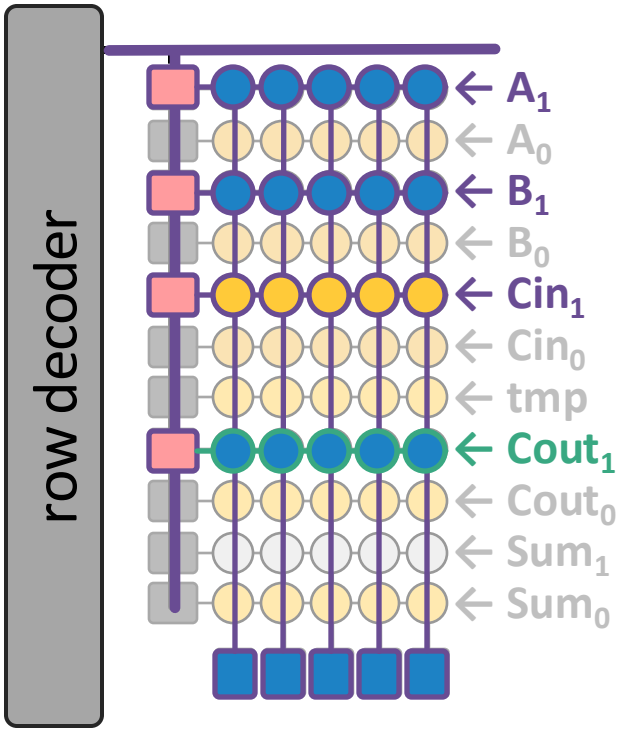
Limitations of Bit-Serial PUD Systems:
Throughput-Oriented Execution (II)

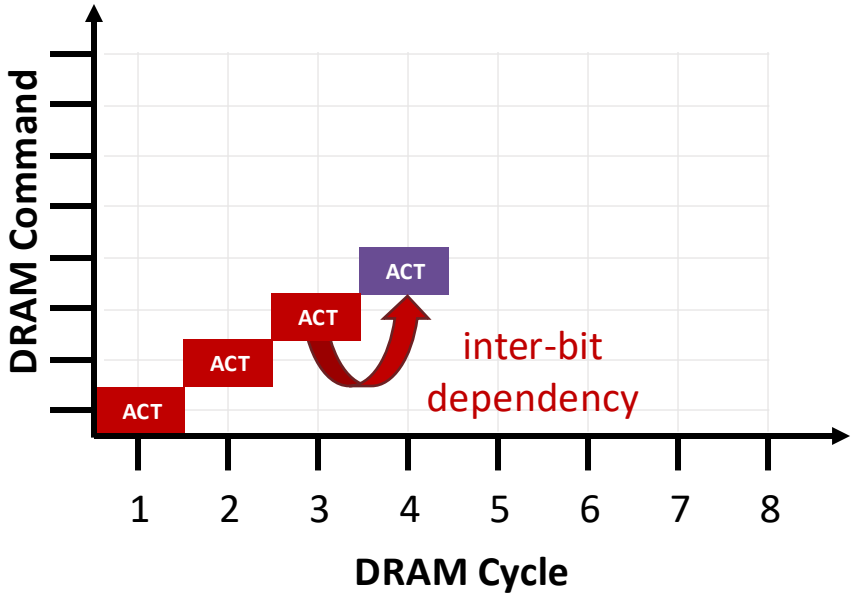


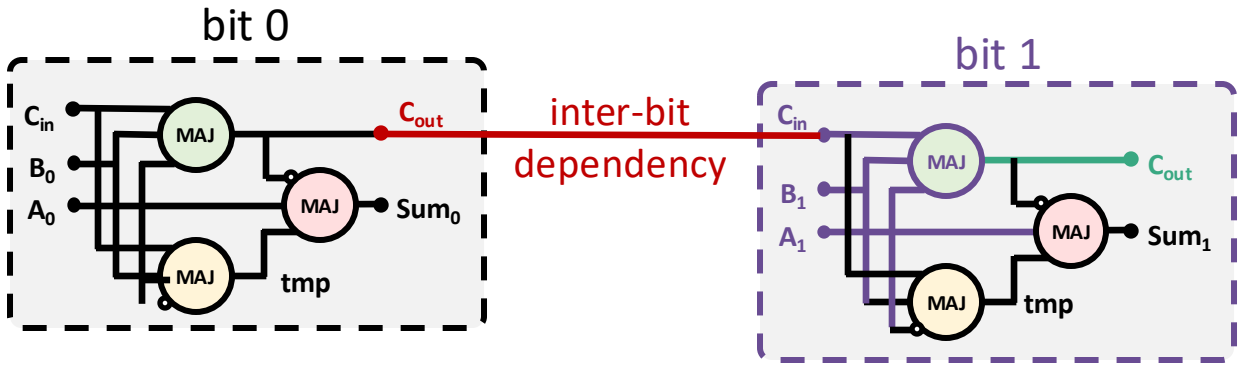




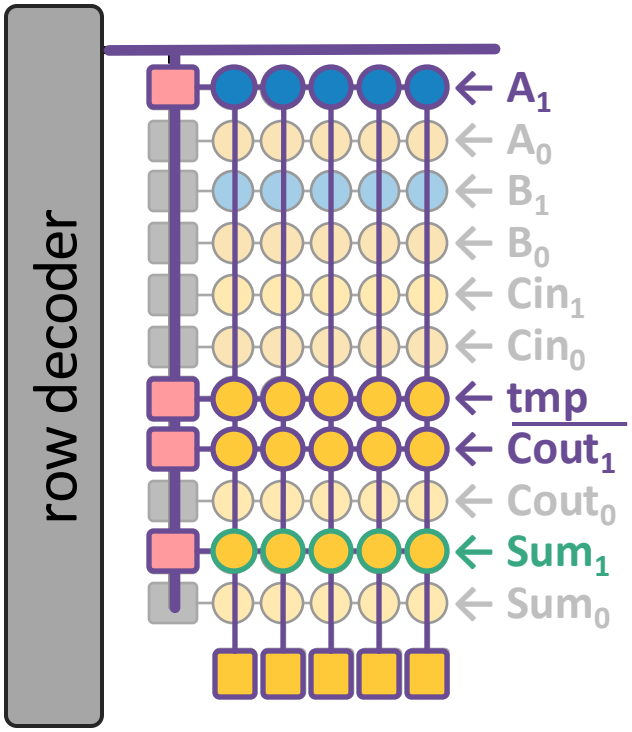
Limitations of Bit-Serial PUD Systems:
Throughput-Oriented Execution (II)

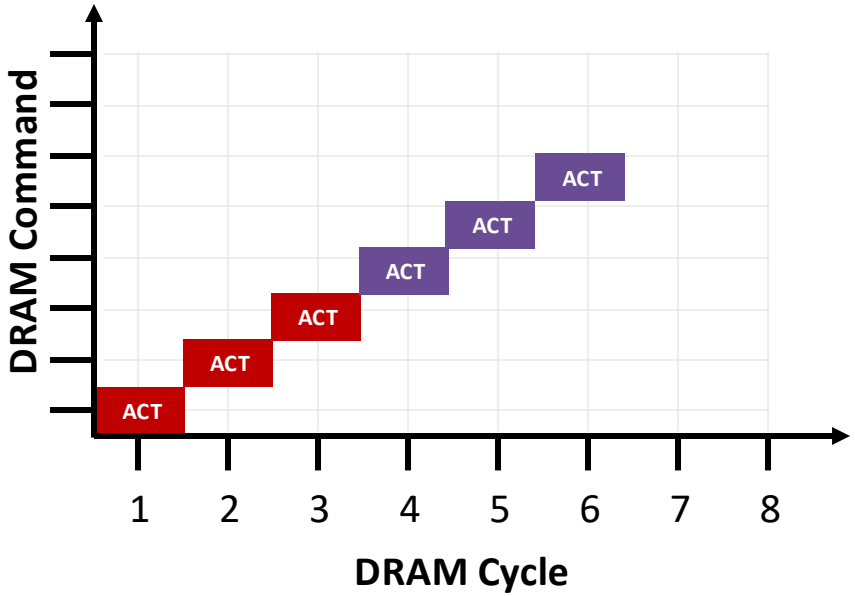


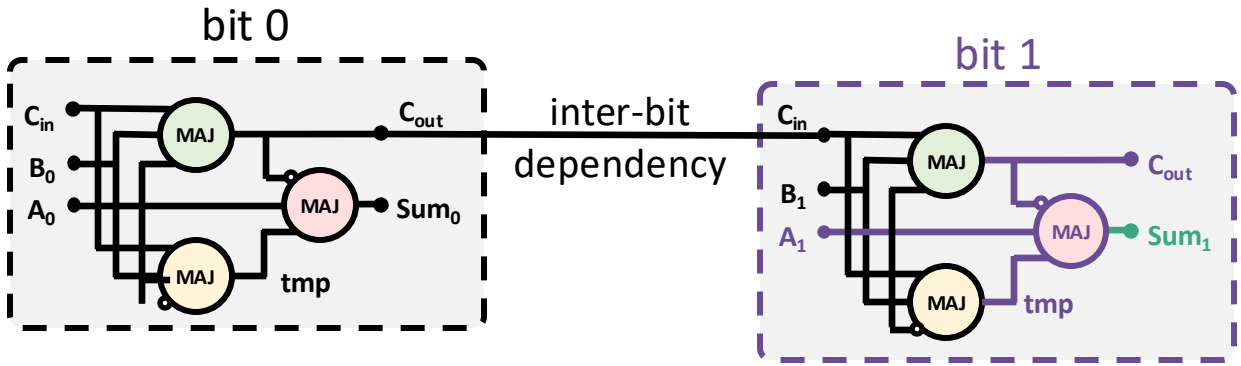




Limitations of Bit-Serial PUD Systems:
Throughput-Oriented Execution (II)



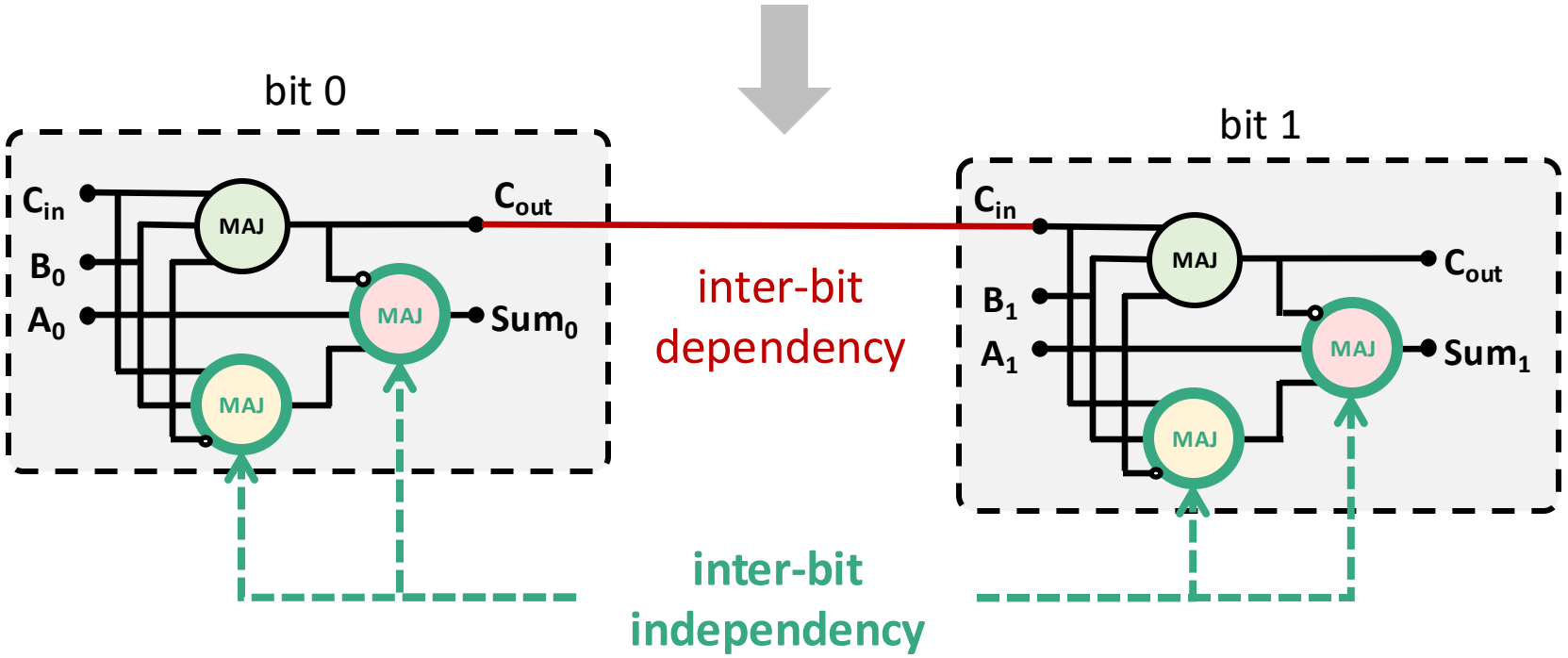




Opportunities for Bit-Serial PUD:
Latency Reduction with Subarray Parallelism (I)

Key Idea

Leverage bit-level parallelism
to execute independent PUD primitives concurrently
across different DRAM subarrays



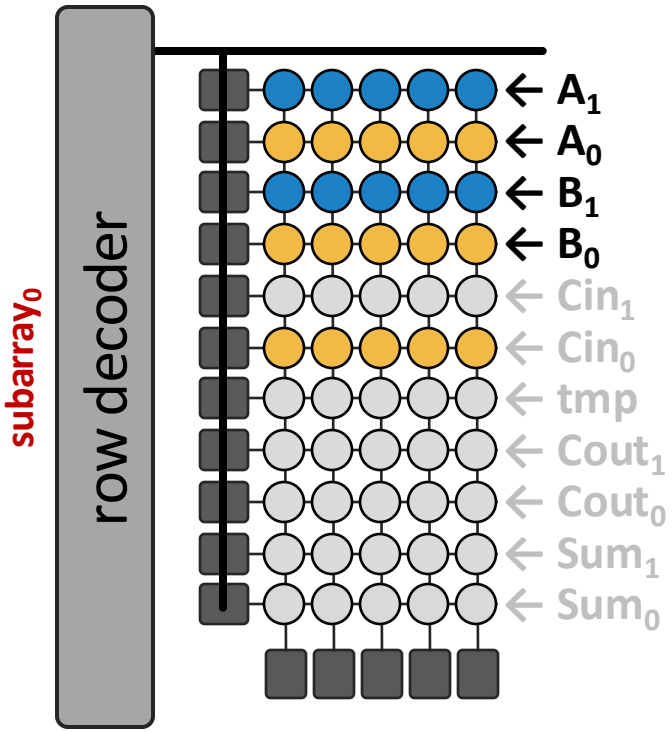
Opportunities for Bit-Serial PUD:
Latency Reduction with Subarray Parallelism (II)

Proteus' **one-bit per-subarray data mapping**

→ scatter bits across subarrays to expose bit-level parallelism

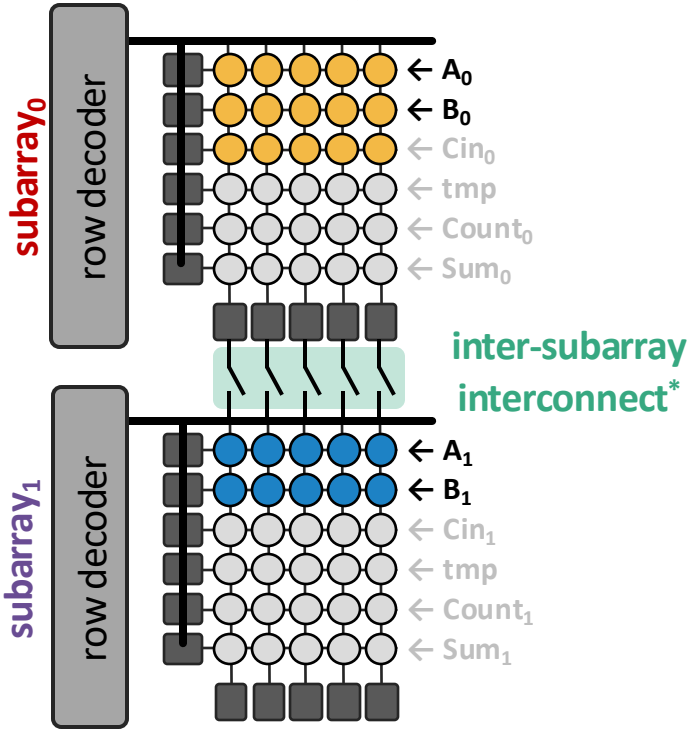
all-bits per-subarray

→ *forces serialization across bits*



one-bit per-subarray

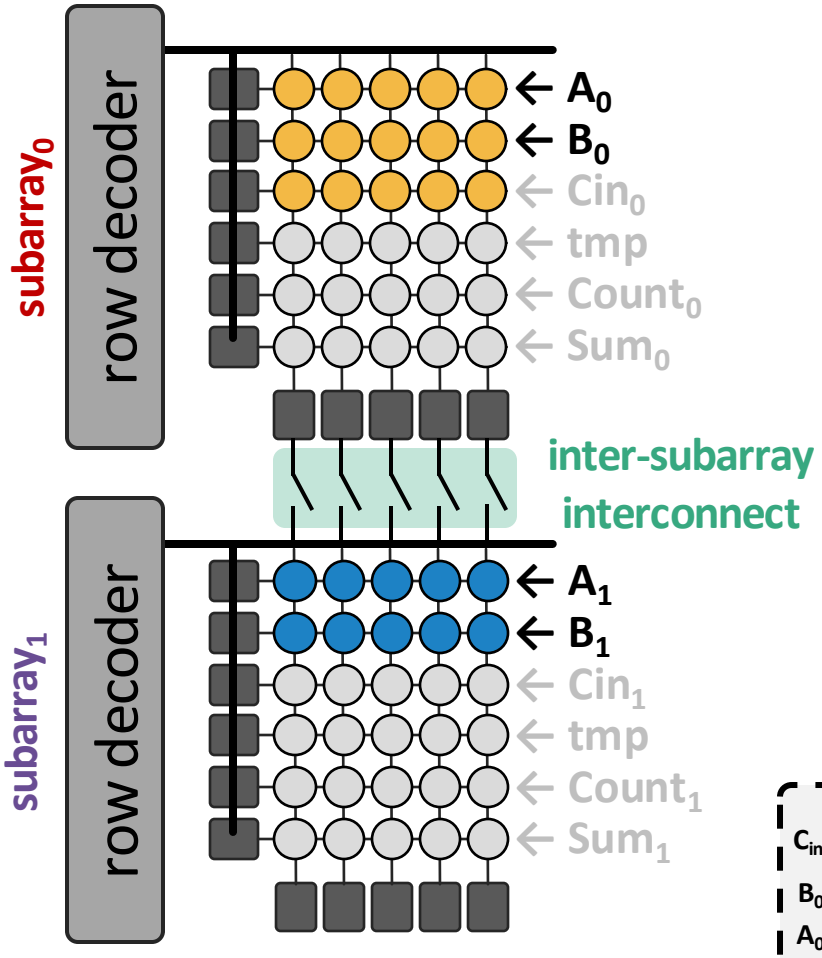
→ exposes bit-level parallelism

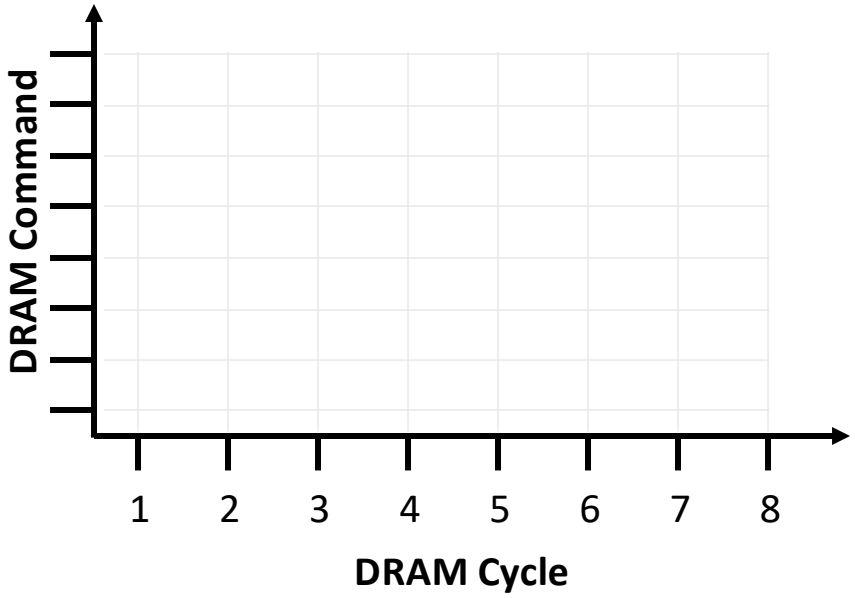


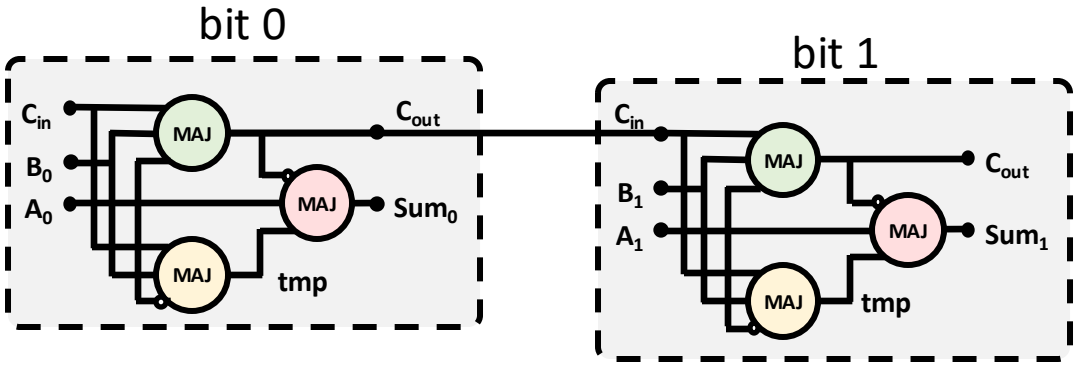
Chang, Kevin K. et al.

"Low-Cost Inter-Linked Subarrays (LISA): Enabling Fast Inter-Subarray Data Movement in DRAM," in HPCA, 2016

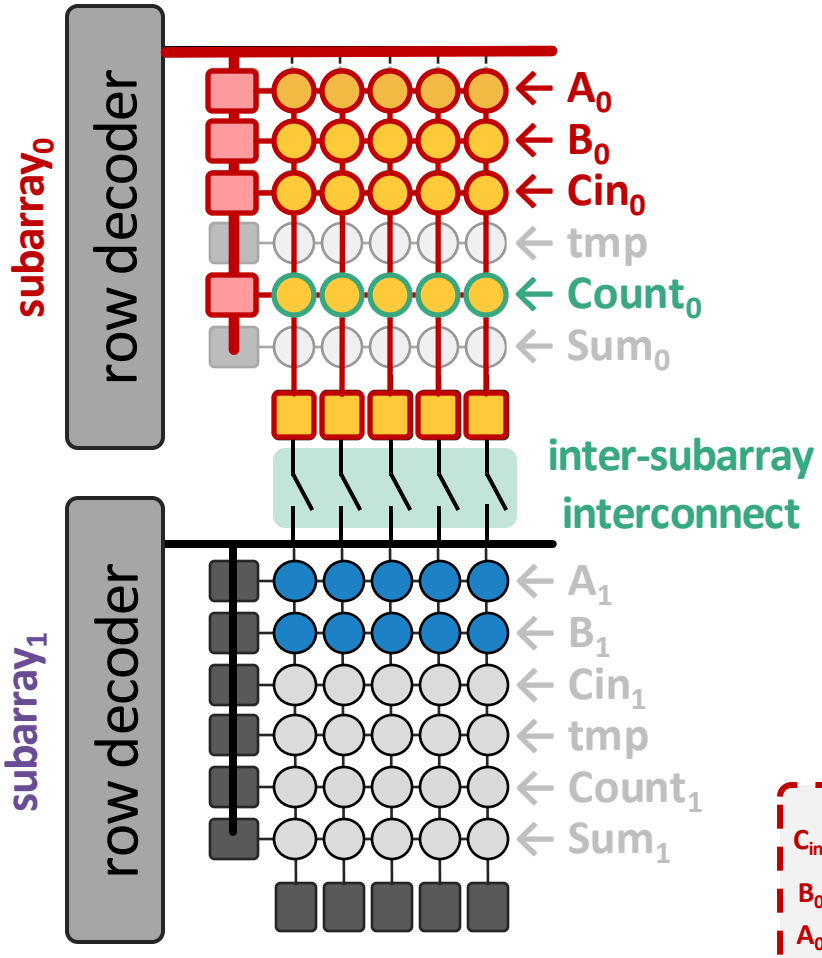
Opportunities for Bit-Serial PUD:
Latency Reduction with Subarray Parallelism (III)

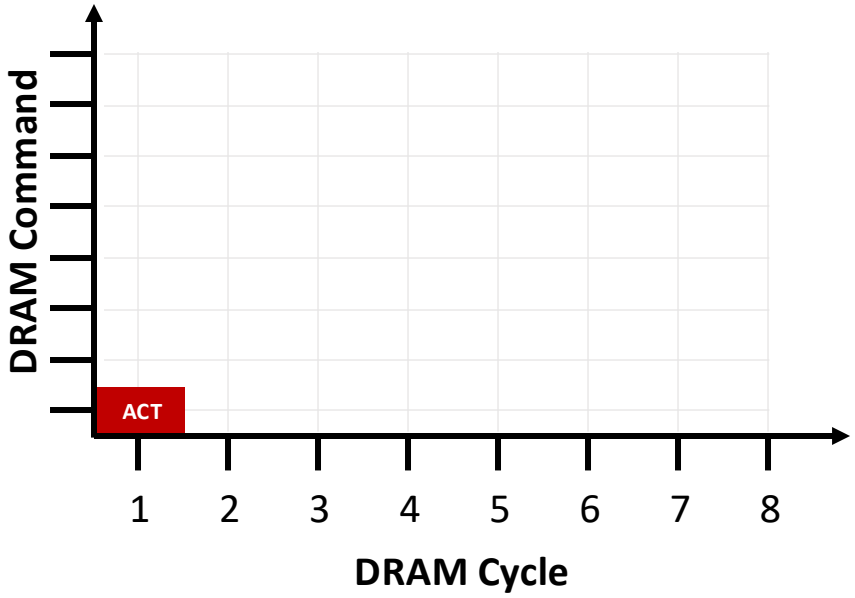


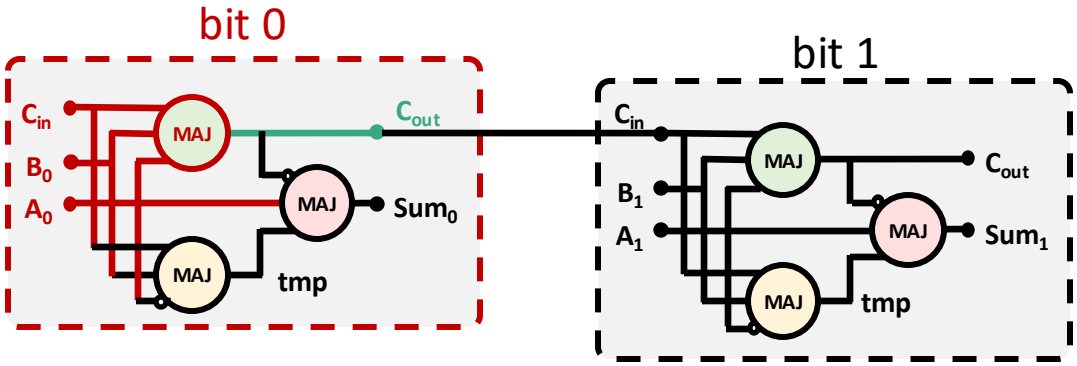




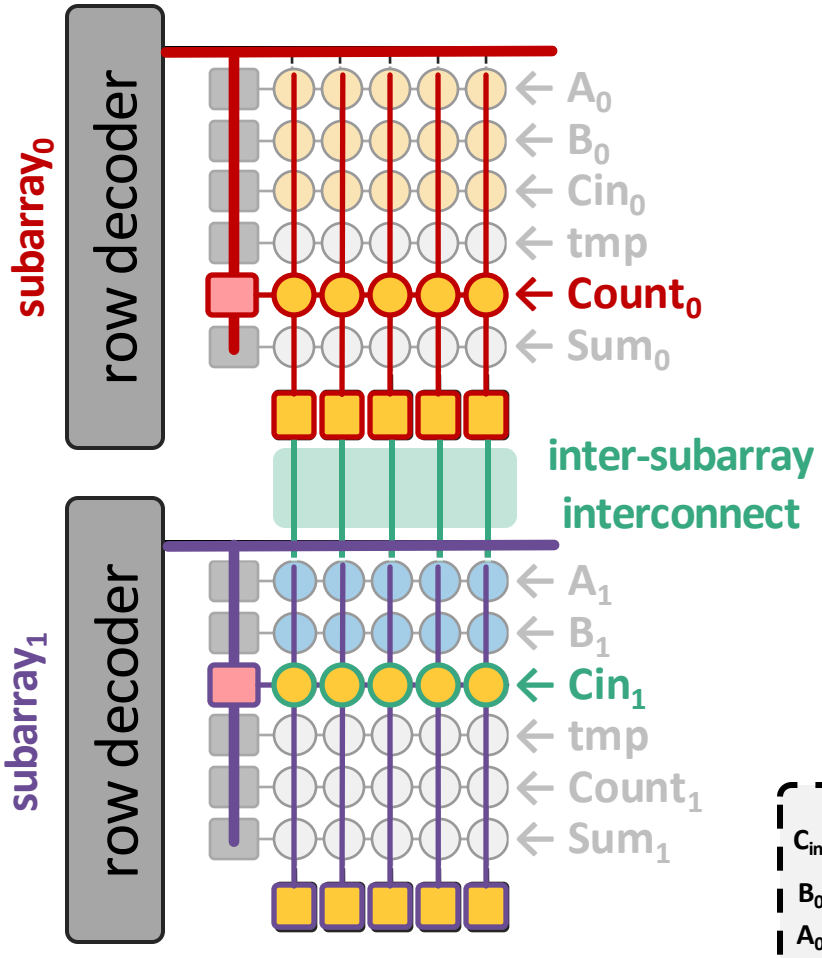
Opportunities for Bit-Serial PUD:
Latency Reduction with Subarray Parallelism (III)

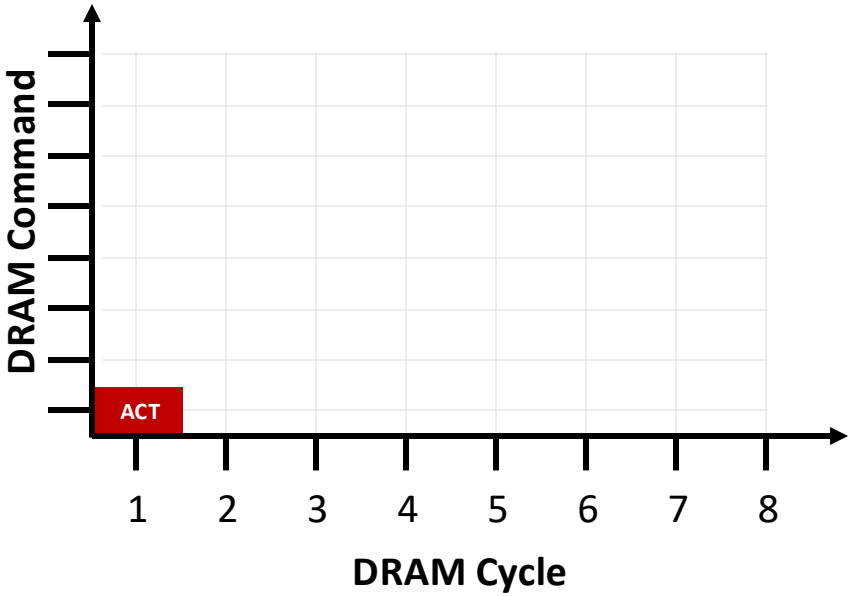


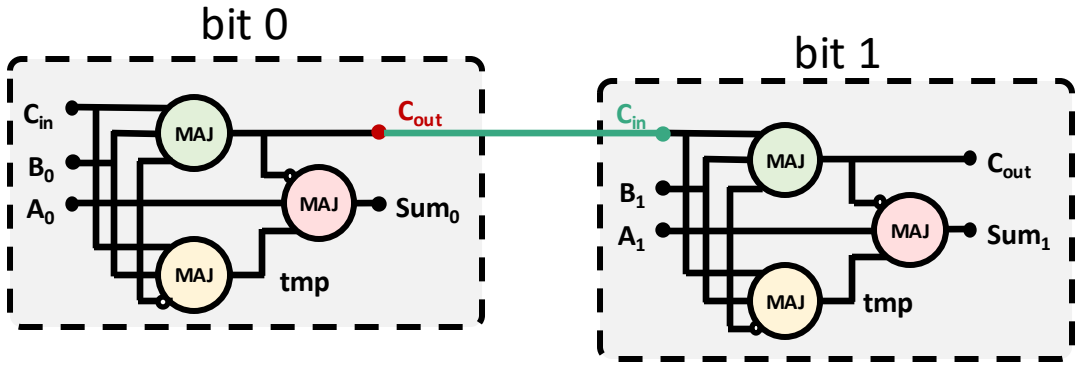




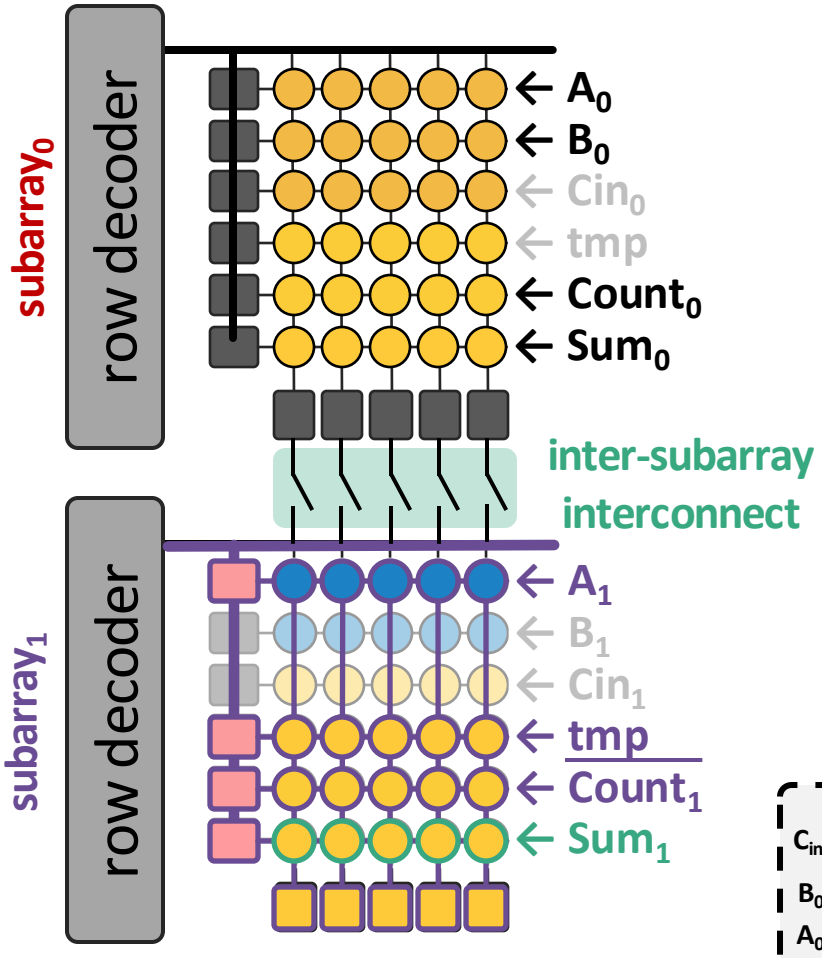
Opportunities for Bit-Serial PUD:
Latency Reduction with Subarray Parallelism (III)

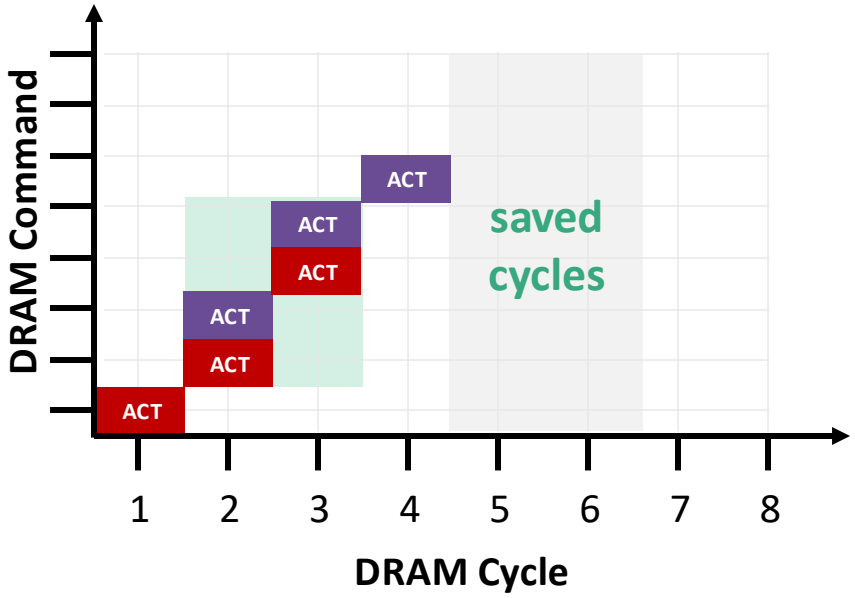


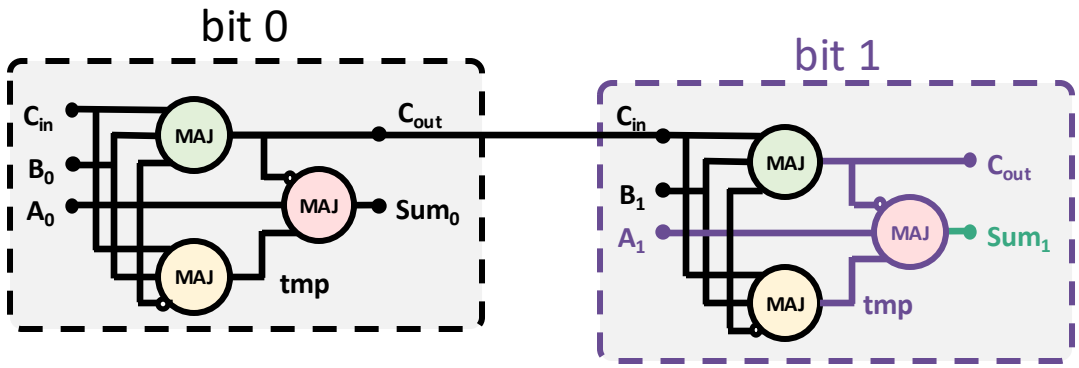




Opportunities for Bit-Serial PUD:
Latency Reduction with Subarray Parallelism (III)







Opportunities for Bit-Serial PUD: Overview

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

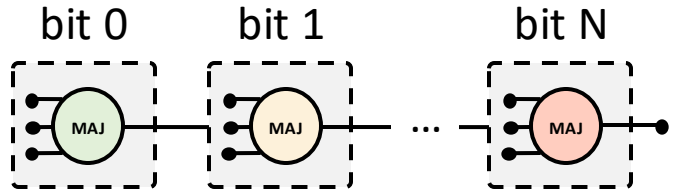
1 Rigid and Static Data Representation
- leading to a subpar performance in the presence of narrow values
- opportunity 1: narrow values for PUD computation

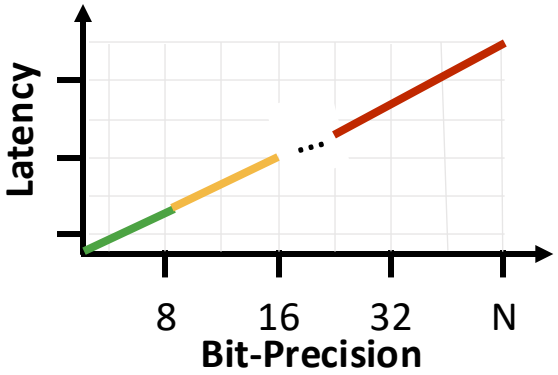
2 Throughput-Oriented Execution with Low Latency Tolerance
- failing to reduce the latency of a single PUD operation
- opportunity 2: DRAM parallelism for latency-oriented execution

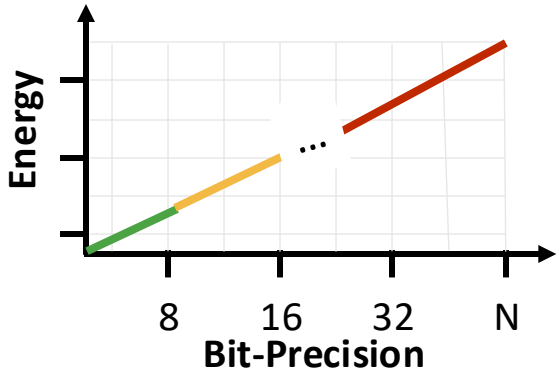
3 Scalability Challenges for High-Precision Operations
- due to the linear/quadratically scaling nature of bit-serial algorithms
- opportunity 3: alternative data representation for high-precision computation

Limitations of Bit-Serial PUD Systems:
Scalability Challenges for High-Precision

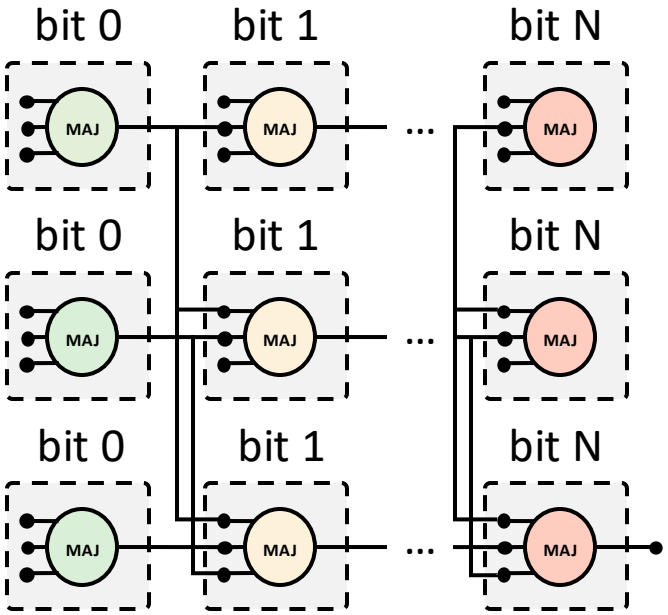
Linearly-Scaling PUD Operations (e.g., addition)

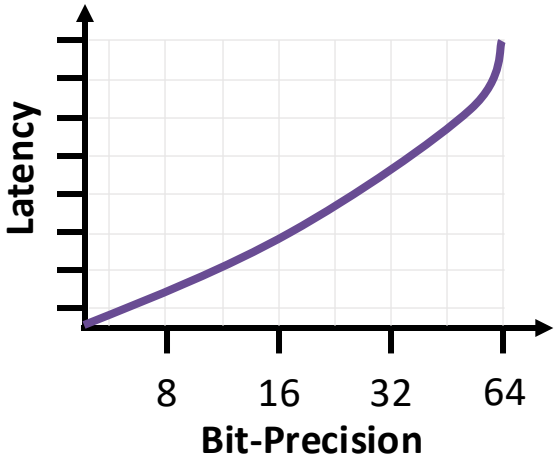


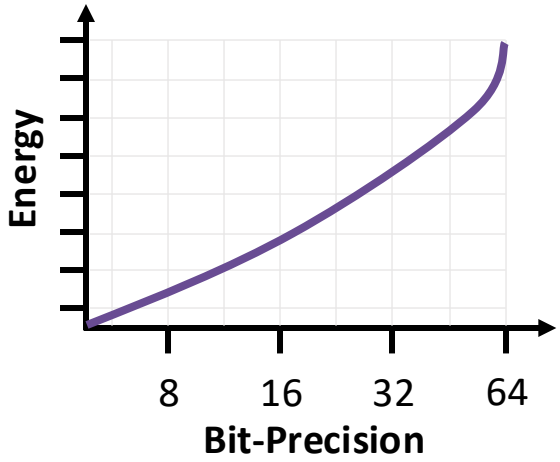




Quadratically-Scaling PUD Operations (e.g., multiplication)







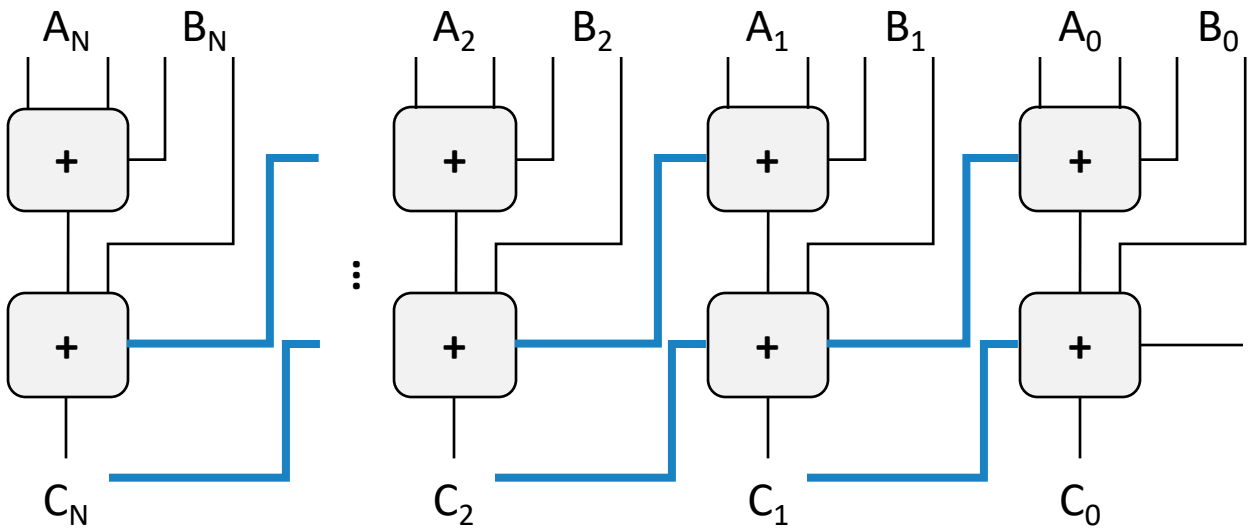
Opportunities for Bit-Serial PUD:
Alternative Data Representation for High-Precision

Key Idea

Use the redundant binary representation (RBR) for high-precision PUD operations

each bit position is represented by two bits
that can take on a value $\in \{-1, 0, 1\}$

$<0, 1, 0, -1> = 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + (-1) \times 2^0 = 4 - 1 = 3$



RBR-Based Arithmetic

1. limits carry propagation to at most two places
2. latency is independent bit-precision

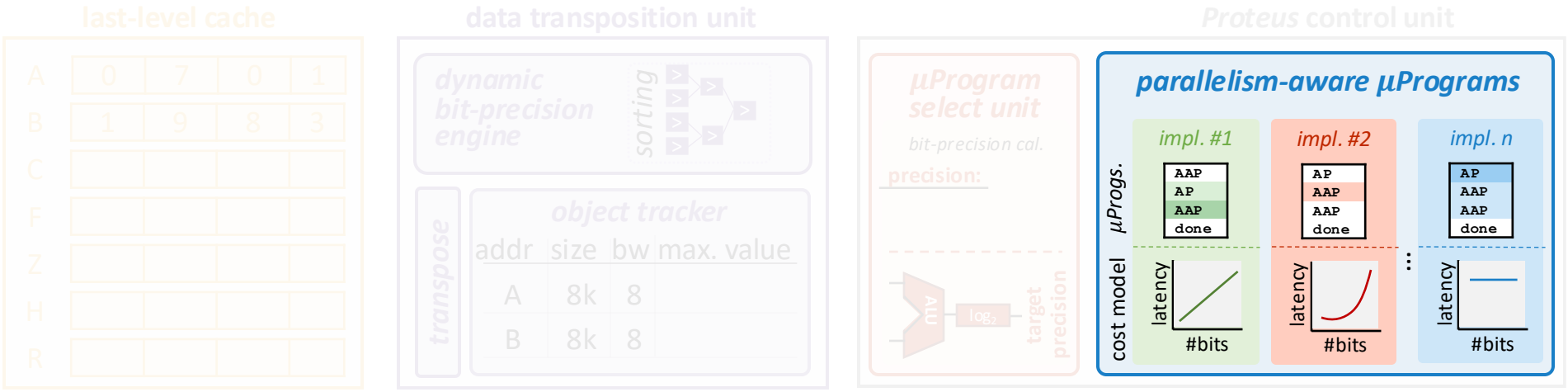
Proteus:
Design Overview (I)

***Proteus*, a data-aware hardware runtime framework that** dynamically and transparently adapts the in-DRAM implementation, bit-precision, and data format based on the input data for efficient PUD execution

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

1. **Parallelism-Aware µProgram Library**
2. **Dynamic Bit-Precision Engine**
3. **µProgram Select Unit**

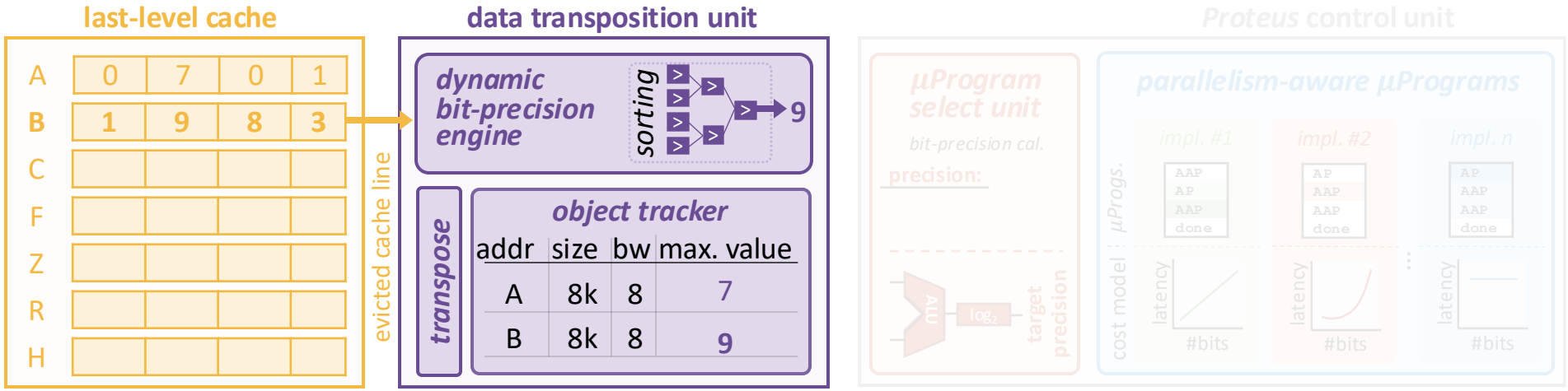
Proteus:
Design Overview (II)



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

1. **Parallelism-Aware μProgram Library**
   hand-optimized implementations of different PUD operations with different performance vs. bit-precision trade-offs
2. **Dynamic Bit-Precision Engine**
3. **μProgram Select Unit**

Proteus:
Design Overview (III)



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

1 **Parallelism-Aware μProgram Library**

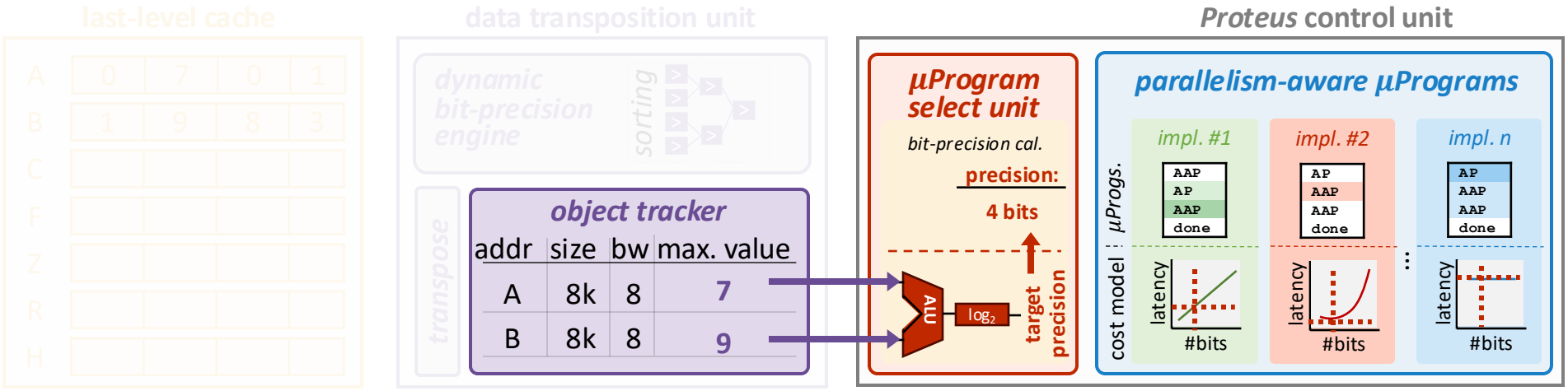
hand-optimized implementations of different PUD operations with different performance vs. bit-precision trade-offs

2 **Dynamic Bit-Precision Engine**

identifies the dynamic range of evicted cache lines belonging to the target PUD operation

3 **μProgram Select Unit**

Proteus:
Design Overview (IV)



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

1 Parallelism-Aware μProgram Library

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

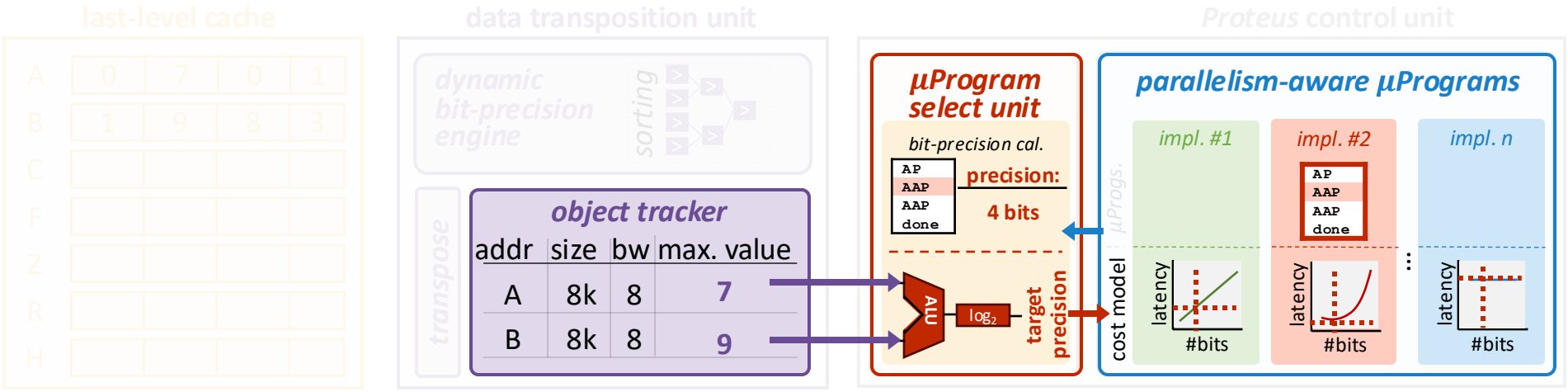
2 Dynamic Bit-Precision Engine

identifies the dynamic range of evicted cache lines belonging to the target PUD operation

3 μProgram Select Unit

identifies 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 µProgram Library

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

2 Dynamic Bit-Precision Engine

identifies the dynamic range of evicted cache lines belonging to the target PUD operation

3 µProgram Select Unit

identifies the appropriate bit-precision based on the input operations of the target PUD operation

Proteus:
More in the Paper & GitHub

- Detailed hardware description
- Implementation of bit-parallel PUD operations
- Pareto analyses for PUD operations
- Data format conversion procedure
- System integration
- *Proteus* for floating-point operations

Proteus:
More in the Paper & GitHub

- Detailed hardware description
- [illegible]
- [illegible]
- [illegible]
- System integration
- *Proteus* for floating-point operations

***Proteus*: Achieving High-Performance Processing-Using-DRAM with Dynamic Bit-Precision, Adaptive Data Representation, and Flexible Arithmetic**

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

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

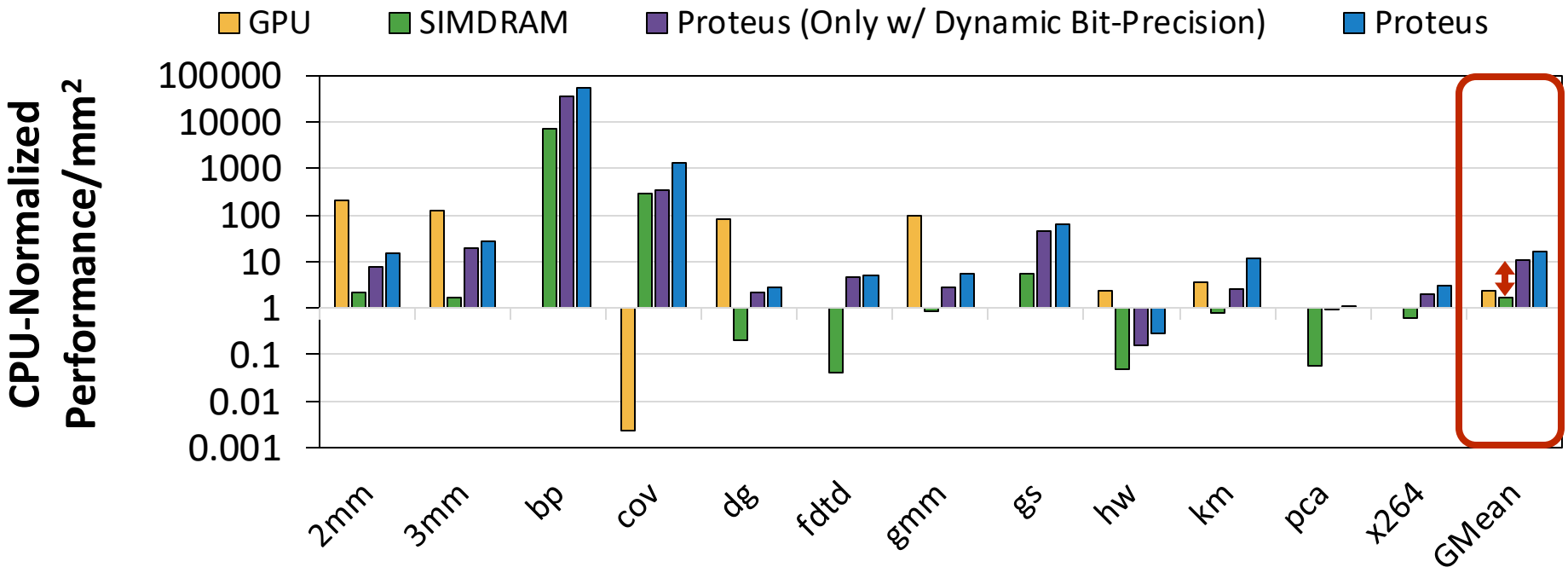
https://arxiv.org/abs/2501.17466

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

Evaluation:
Methodology Overview

- **Evaluation Setup**
  - **CPU:** Intel Skylake CPU
  - **GPU:** NVIDIA A100 GPU (with and without Tensor Cores)
  - **PUD:** SIMDRAM [Oliveira+, 2021]
  - **https://github.com/CMU-SAFARI/Proteus**
- **Workloads**
  - 12 workloads from Polybench, Rodinia, Phoenix, and SPEC2017
  - Linear algebra, data mining, video processing, machine learning
- **Metrics**
  - CPU-normalized **performance per area**
  - **Energy reduction** (compared to baseline CPU)

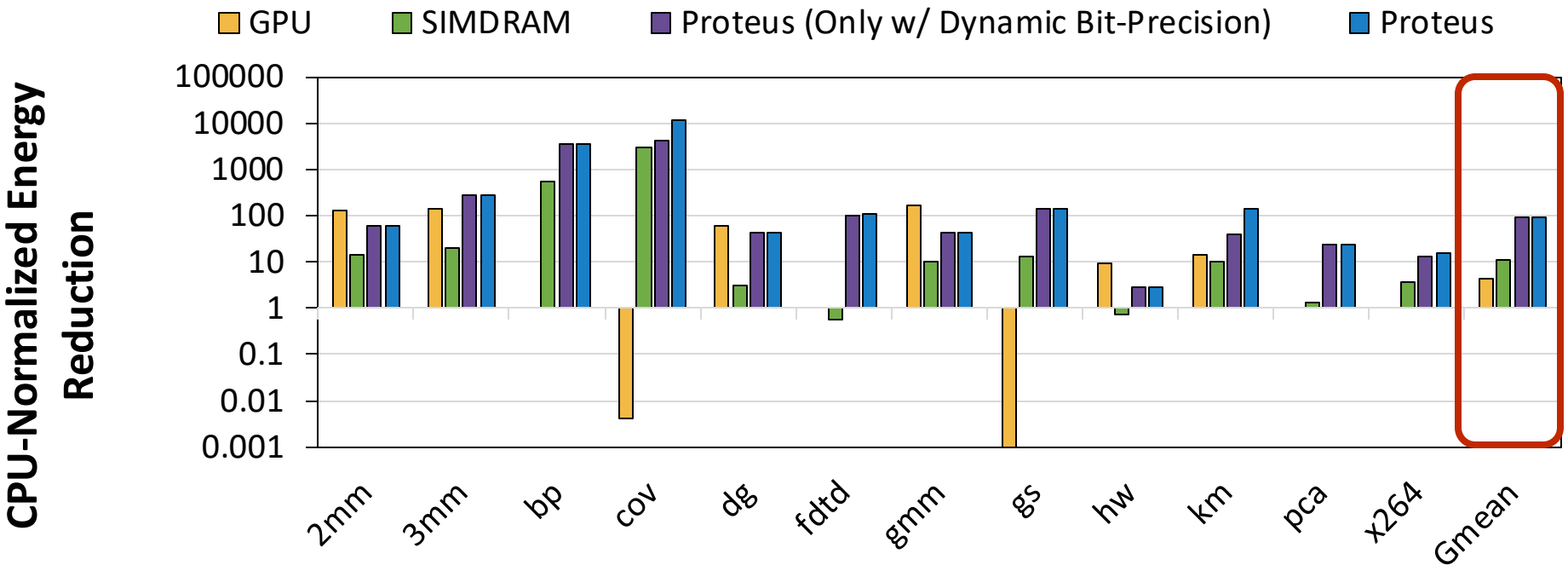
Evaluation:
Performance Analysis



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)**

Evaluation:
More in the Paper

- *Proteus* with statically-identified narrow values
- Data mapping and representation format conversion overheads
- Performance of floating-point operations
- *Proteus* versus Tensor Cores in GPUs
- Area analysis

Evaluation:
More in the Paper

- *Proteus* with statically-identified narrow values
- [illegible]
- [illegible]
- *Proteus* versus Tensor Cores in GPUs
- Area analysis

***Proteus*: Achieving High-Performance Processing-Using-DRAM with Dynamic Bit-Precision, Adaptive Data Representation, and Flexible Arithmetic**

Geraldo F. Oliveira[†] Mayank Kabra[†] Yuxin Guo[‡] Kangqi Chen[†]
A. Giray Yağlıkçı[†] Melina Soysal[†] Mohammad Sadrosadati[†]
Joaquin O. 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/abs/2501.17466

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

Processing-Using-Memory Systems for Bulk Bitwise Operations

June 21st 2025

Geraldo F. Oliveira
geraldofojunior@gmail.com
https://geraldofojunior.github.io

SAFARI

ETH zürich

2024 International Symposium on High-Performance Computer Architecture

MIMDRAM

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

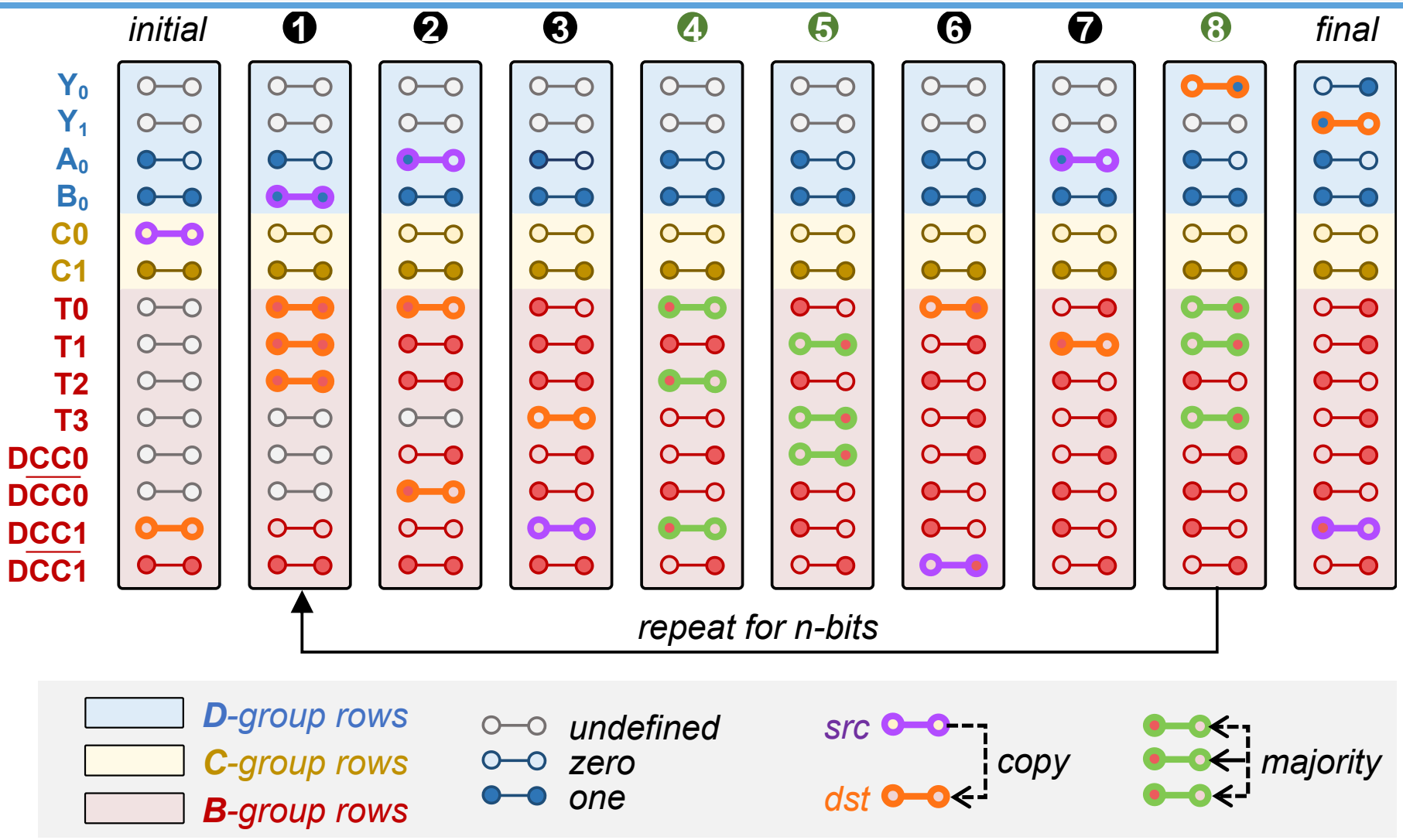
Backup Slides

Geraldo F. Oliveira

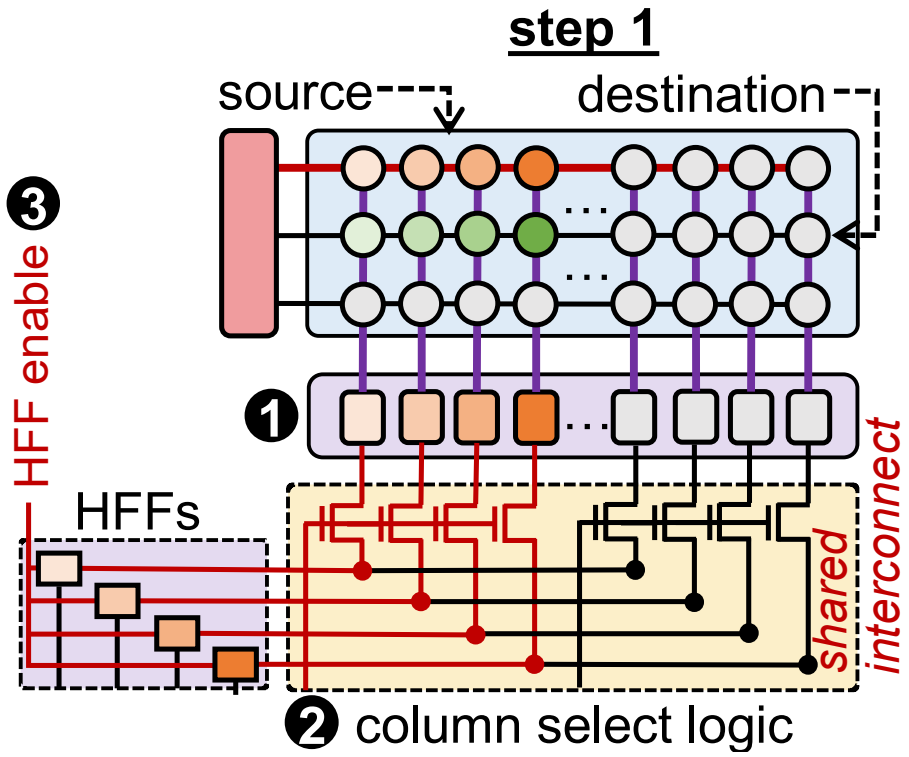
Ataberk Olgun
A. Giray Yağlıkçı
F. Nisa Bostancı
Saugata Ghose
Juan Gómez-Luna
Onur Mutlu

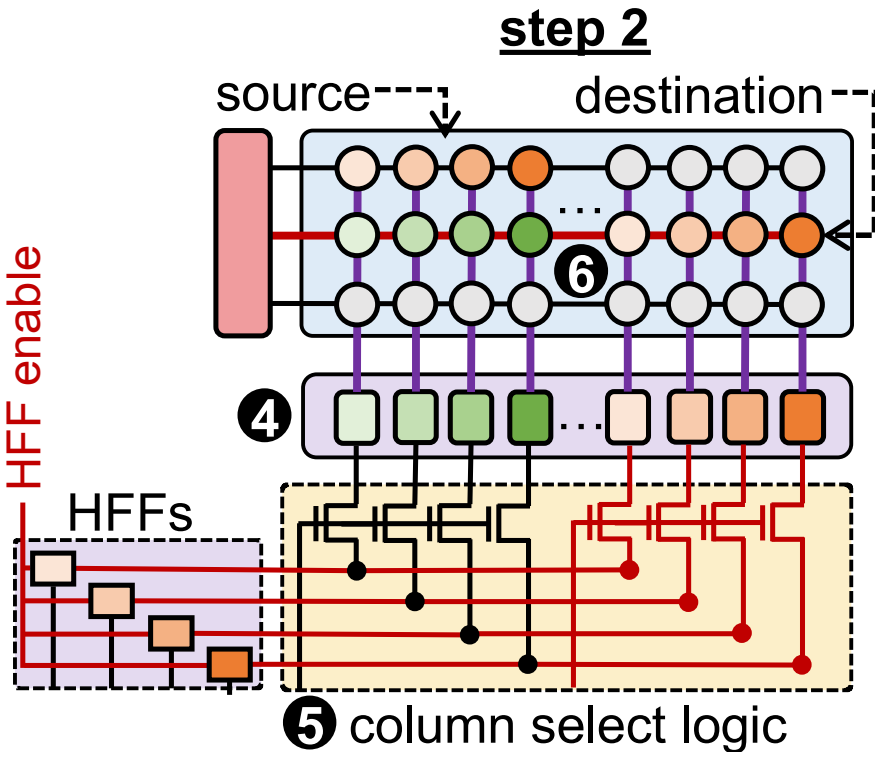
ETH zürich
UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN
SAFARI

Bit-Serial PUD Addition

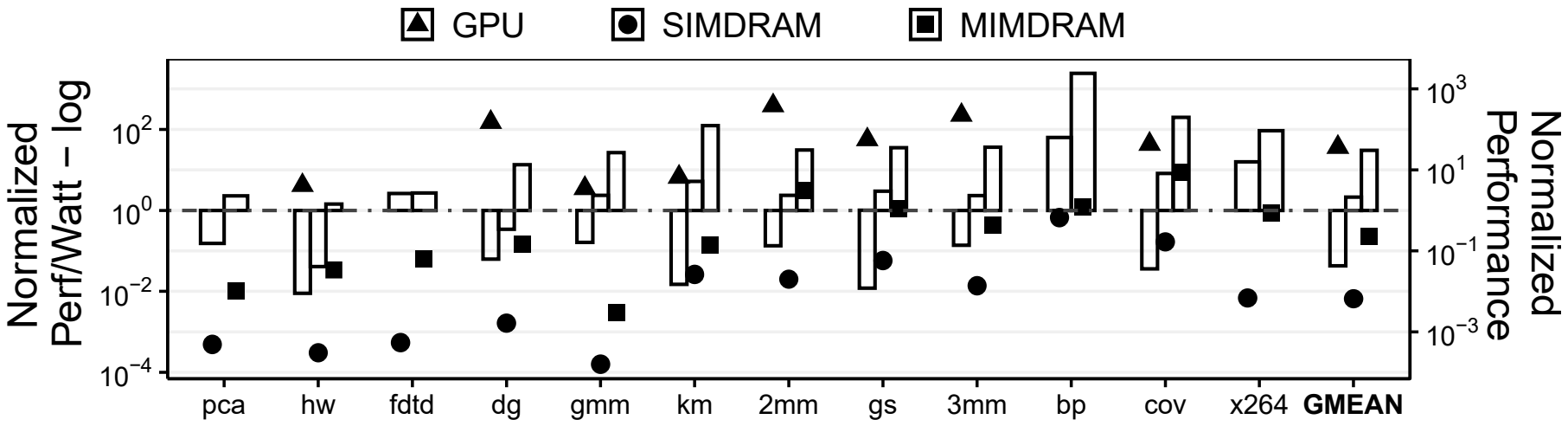


Intra-Mat Interconnect

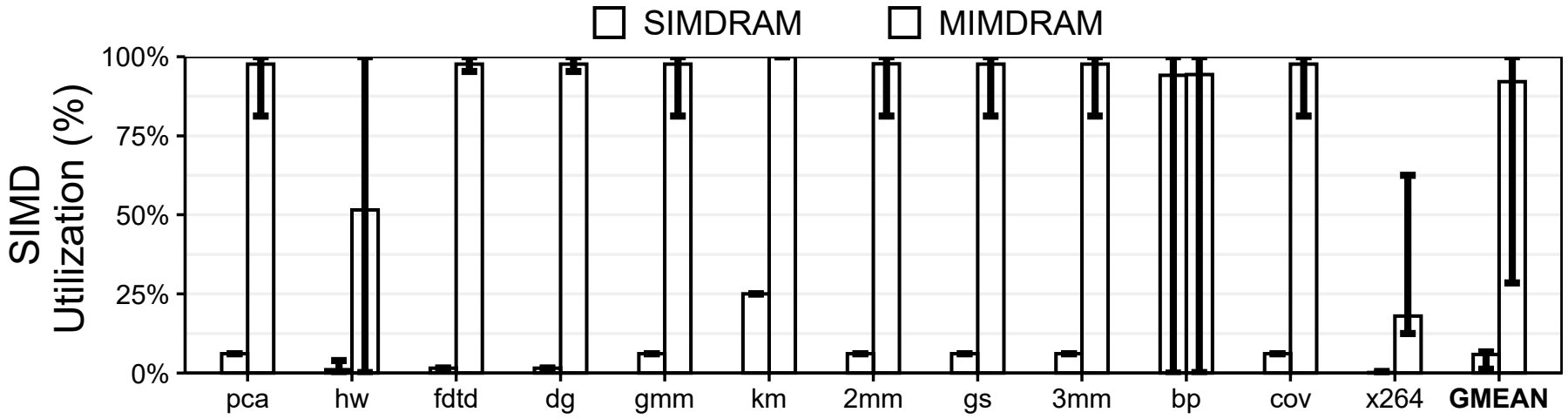




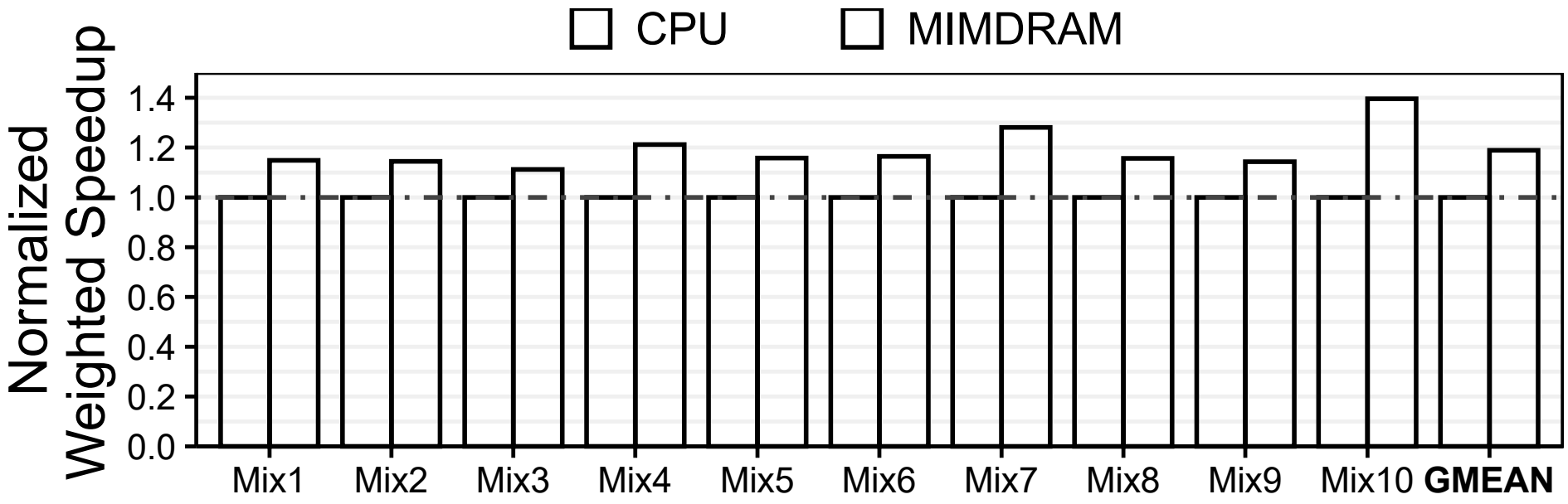
Performance for Single Applications



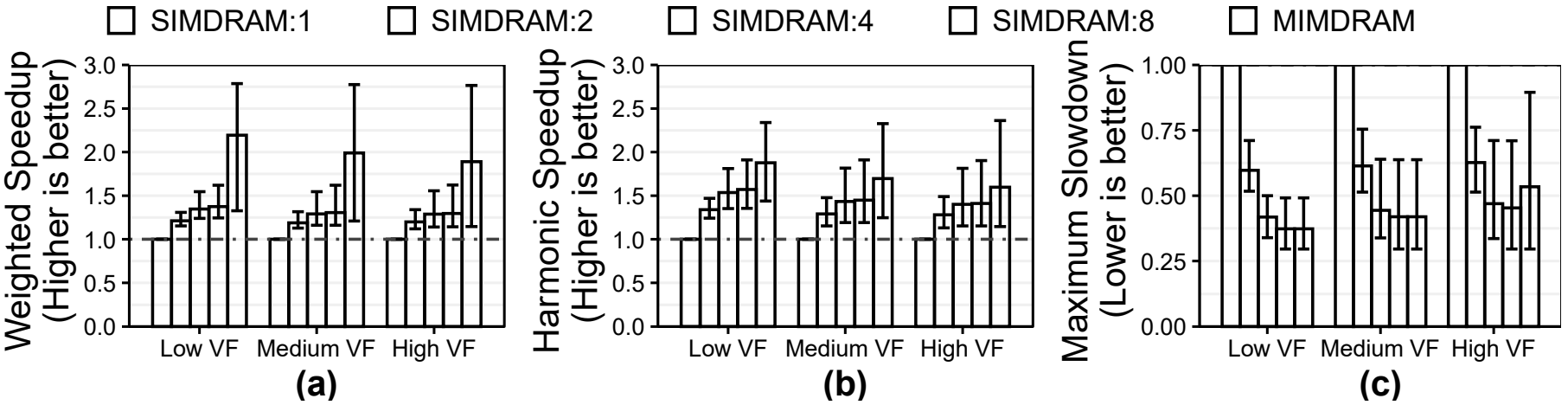
SIMD Utilization: MIMDRAM vs. SIMDRAM



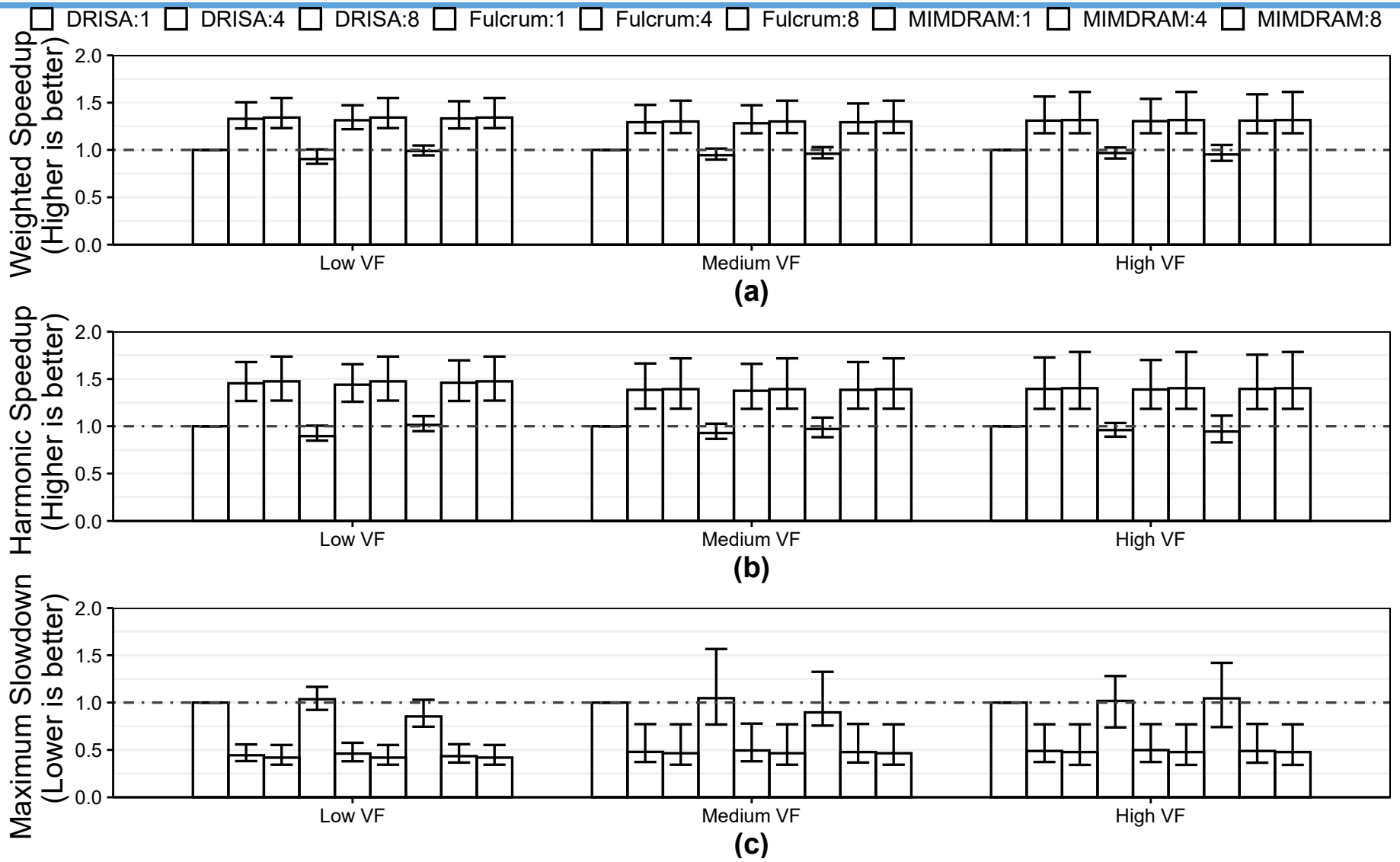
Weighted Speedup vs. CPU



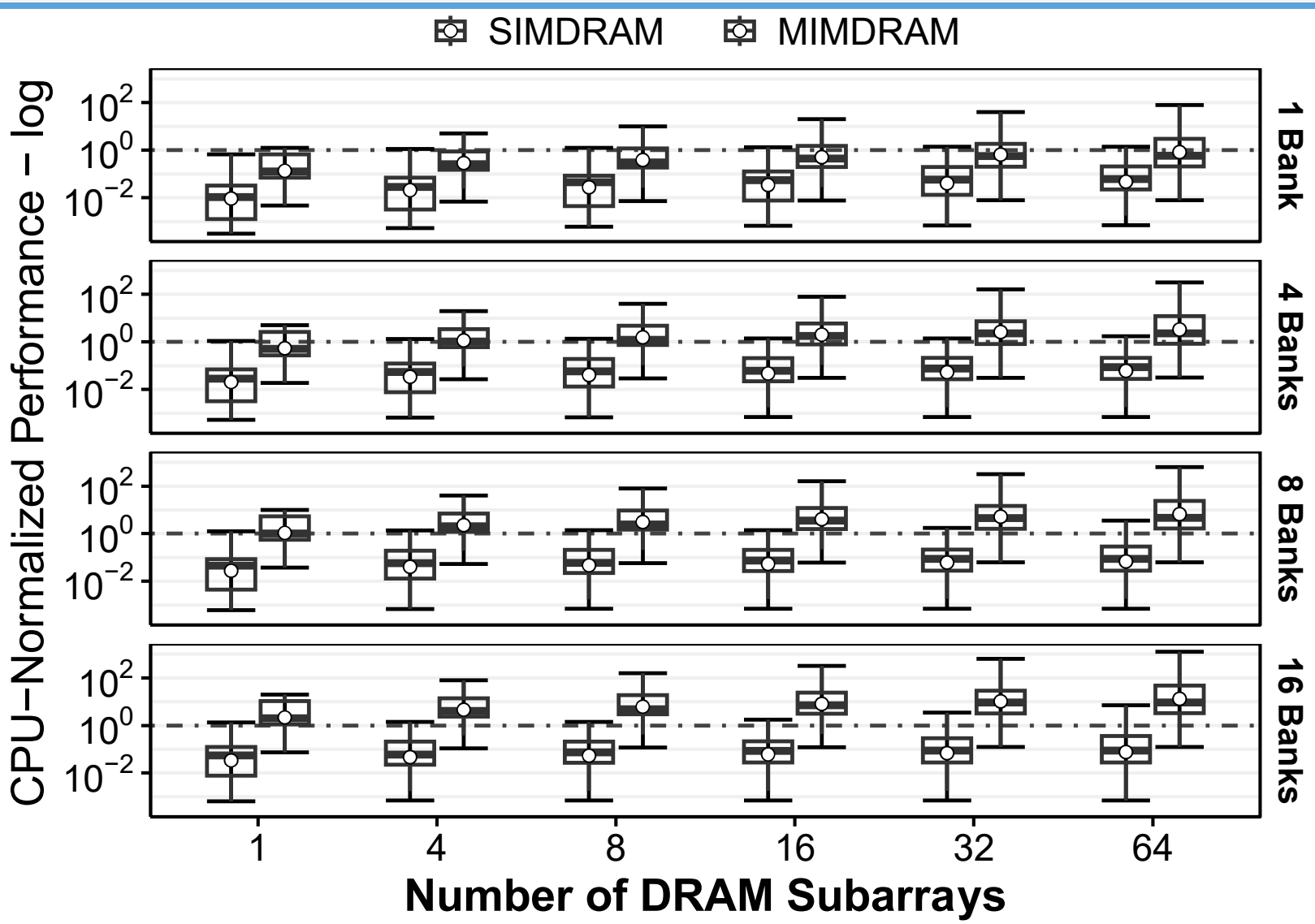
Multi-Applications: Full Data



Multi-Applications: DRISA & Fulcrum



BLP and SALP


SIMDRAM
MIMDRAM
CPU–Normalized Performance – log
10^2
10^0
10^{-2}
1 Bank
4 Banks
8 Banks
16 Banks
1
4
8
16
32
64
Number of DRAM Subarrays

System Configuration

Table 2: Evaluated system configurations.

Real Intel Skylake CPU [209]	x86 [199], 16 cores, 8-wide, out-of-order, 4 GHz; *L1 Data + Inst. Private Cache:* 256 kB, 8-way, 64 B line; *L2 Private Cache:* 2 kB, 4-way, 64 B line; *L3 Shared Cache:* 16 MB, 16-way, 64 B line; *Main Memory:* 64 GB DDR4-2133, 4 channels, 4 ranks
Real NVIDIA A100 GPU [210]	7 nm technology node; 6912 CUDA Cores; 108 streaming multiprocessors, 1.4 GHz base clock; *L2 Cache:* 40 MB L2 Cache; *Main Memory:* 40 GB HBM2 [119, 120]
Simulated SIMDRAM [101] & MIMDRAM	gem5 system emulation; x86 [199], 1-core, out-of-order, 4 GHz; *L1 Data + Inst. Cache:* 32 kB, 8-way, 64 B line; *L2 Cache:* 256 kB, 4-way, 64 B line; *Memory Controller:* 8 kB row size, FR-FCFS [215, 216] *Main Memory:* DDR4-2400, 1 channel, 8 chips, 4 rank 16 banks/rank, 16 mats/chip, 1 K rows/mat, 512 columns/mat *MIMDRAM's Setup:* 8 entries mat queue, 2 kB *bbop* buffer 8 *μProgram processing engines*, 2 kB *mat translation table*

Workload Characteristics

Table 3: Evaluated applications and their characteristics.

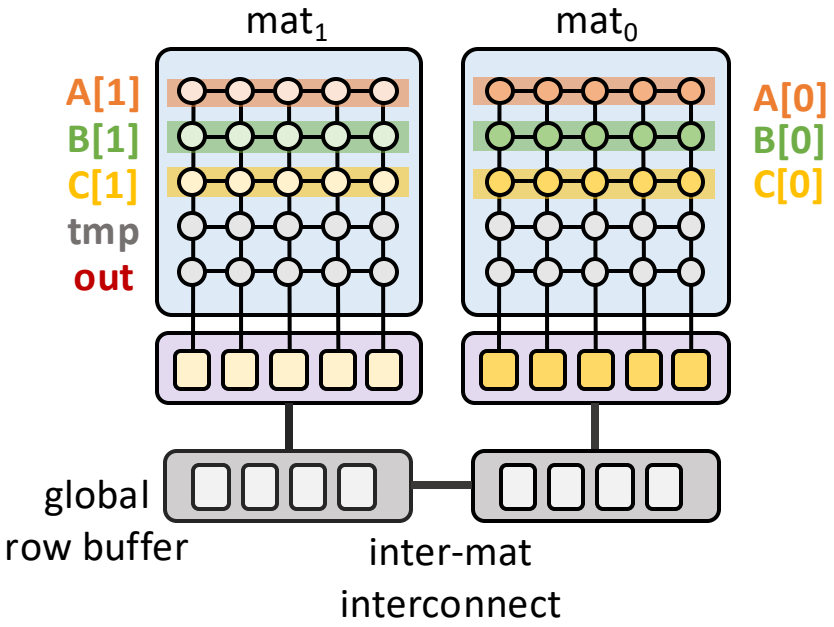
Benchmark Suite	Application (Short Name)	Dataset Size	# Vector Loops	VF {min, max}	PUD Ops.†
Phoenix [161]	‡pca (pca)	reference	2	{4000, 4000}	D, S, M, R
Polybench [162]	2mm (2mm)	NI = NJ = NK = NL = 4000	6	{4000, 4000}	M, R
	‡3mm (3mm)	NI = NJ = NK = NL = NM = 4000	7	{4000, 4000}	M, R
	covariance (cov)	N = M = 4000	2	{4000, 4000}	D, S, R
	doitgen (dg)	NQ = NR = NP = 1000	5	{1000, 1000}	M, C, R
	‡fdtd-apml (fdtd)	CZ = CYM = CXM = 1000	3	{1000, 1000}	D, M, S, A
	gemm (gmm)	NI = NJ = NK = 4000	4	{4000, 4000}	M, R
	gramschmidt (gs)	NI = NJ = 4000	5	{4000, 4000}	M, D, R
Rodinia [163]	backprop (bs)	134217729 input elm.	1	{17, 134217729}	M, R
	heartwall (hw)	reference	4	{1, 2601}	M, R
	kmeans (km)	16384 data points	2	{16384, 16384}	S, M, R
SPEC 2017 [164]	525.x64_r (x264)	reference input	2	{64, 320}	A

†: D = division, S = subtraction, M = multiplication, A = addition, R = reduction, C = copy

‡: application with independent PUD operations

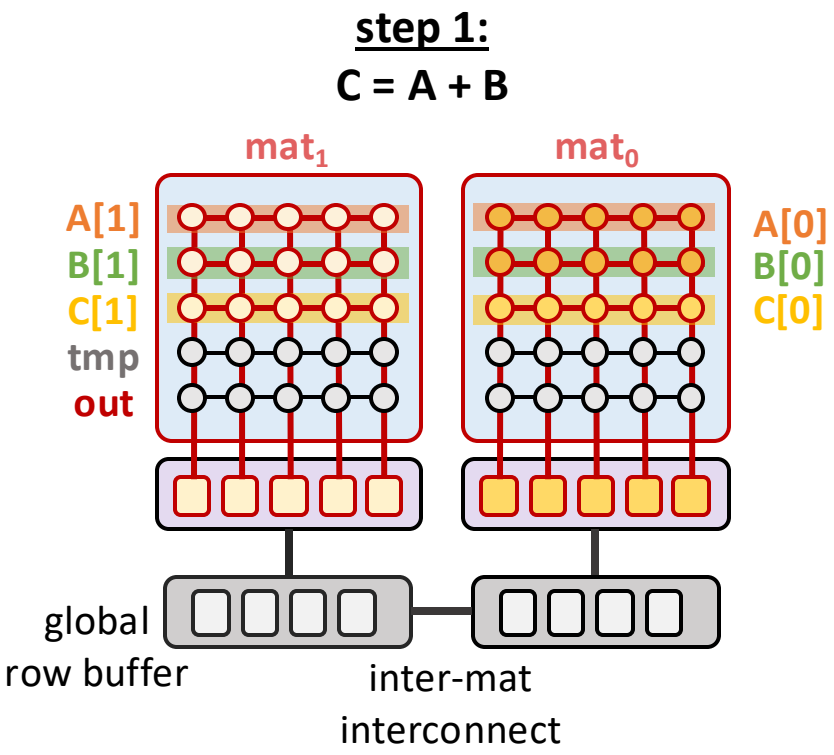
MIMDRAM:
In-DRAM Vector Reduction (I)

MIMDRAM leverages fine-grained DRAM access and inter-/intra-mat interconnects to implement in-DRAM vector reduction



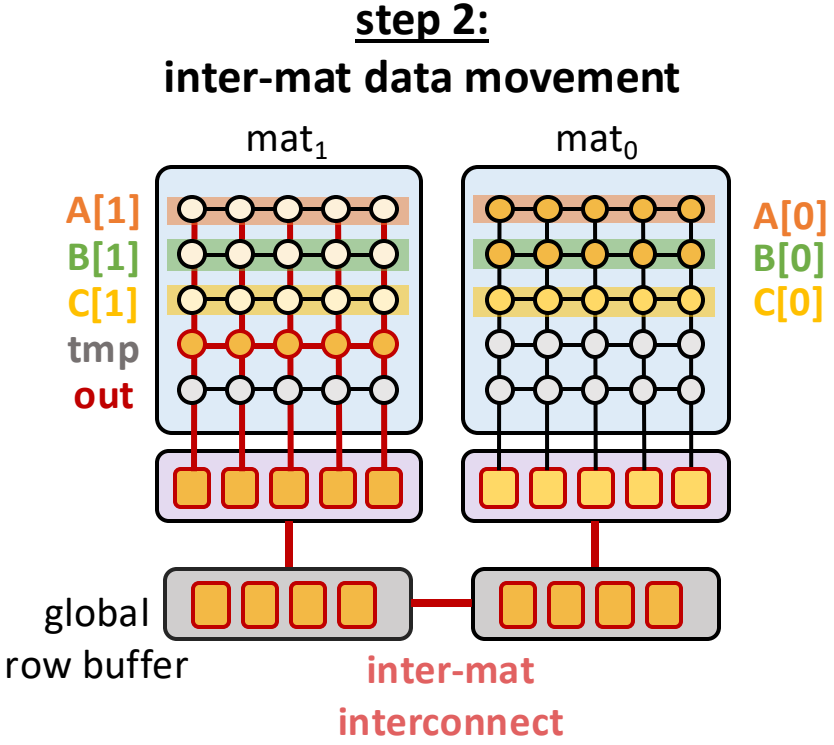
MIMDRAM:
In-DRAM Vector Reduction (II)

MIMDRAM leverages fine-grained DRAM access and inter-/intra-mat interconnects to implement in-DRAM vector reduction



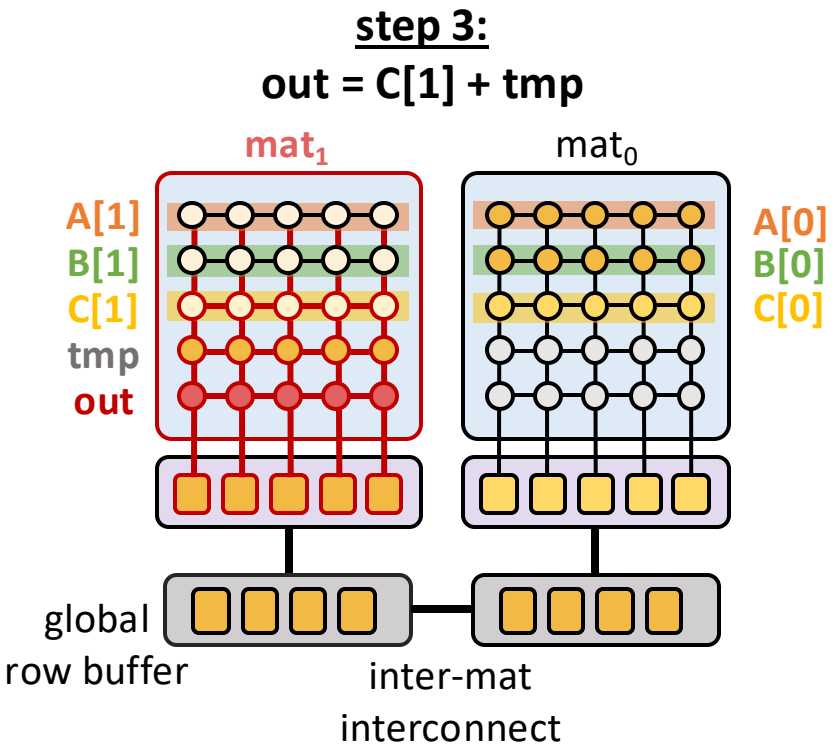
MIMDRAM:
In-DRAM Vector Reduction (III)

MIMDRAM leverages fine-grained DRAM access and inter-/intra-mat interconnects to implement in-DRAM vector reduction



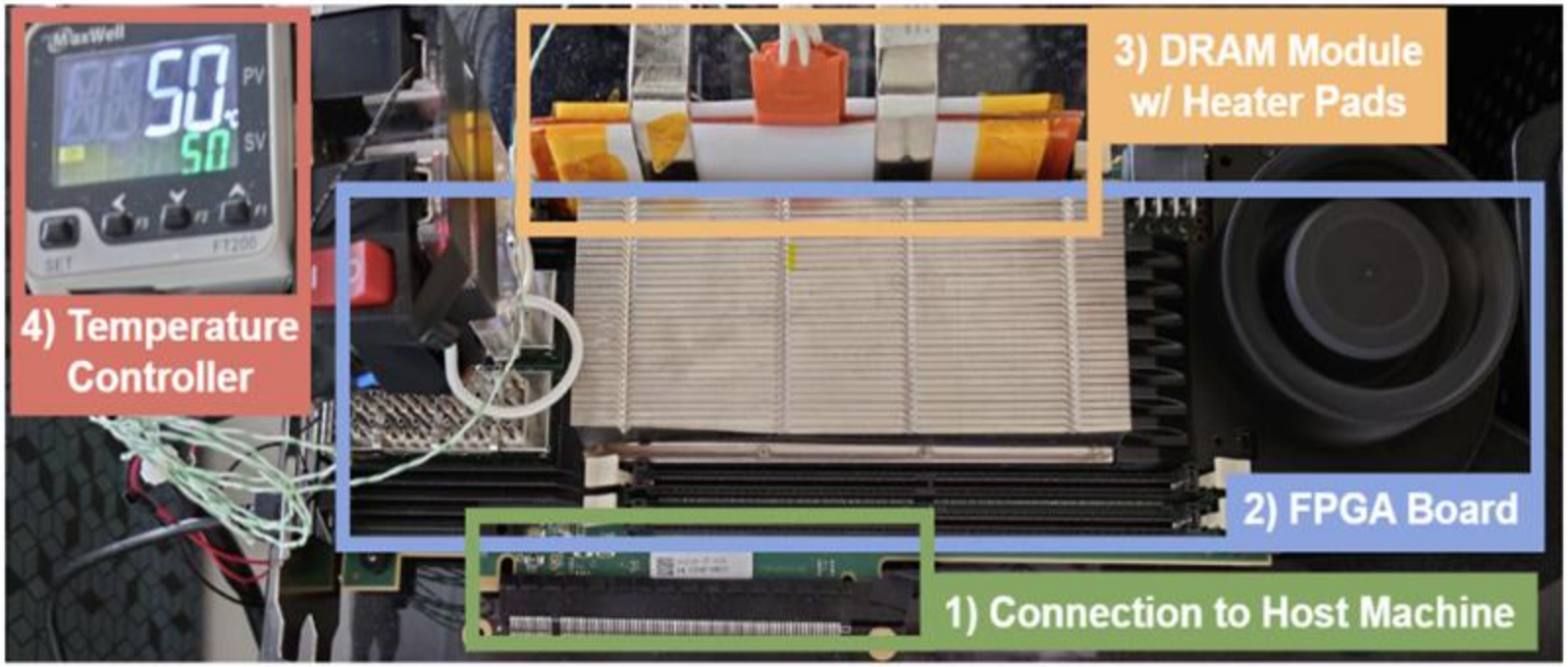
MIMDRAM:
In-DRAM Vector Reduction (IV)

MIMDRAM leverages fine-grained DRAM access and inter-/intra-mat interconnects to implement in-DRAM vector reduction



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.

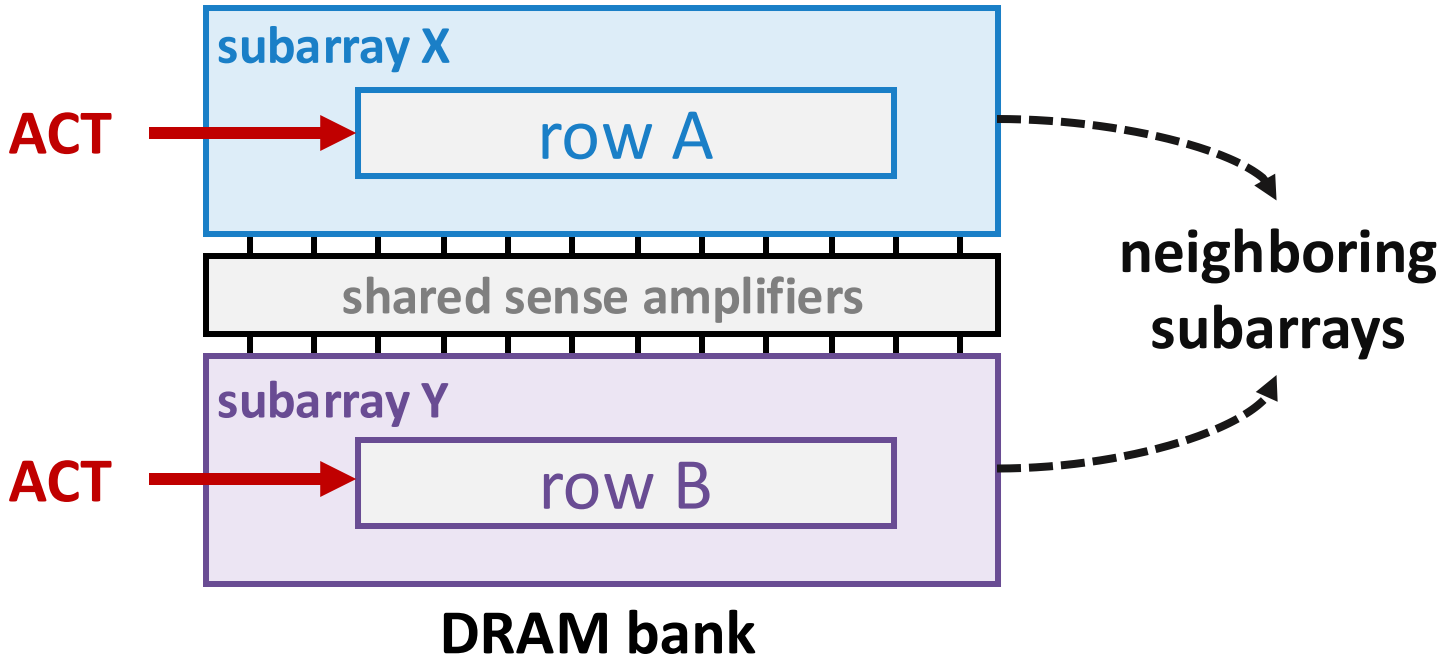
PUD in COTS DRAM:
Tested DRAM Chips

- **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



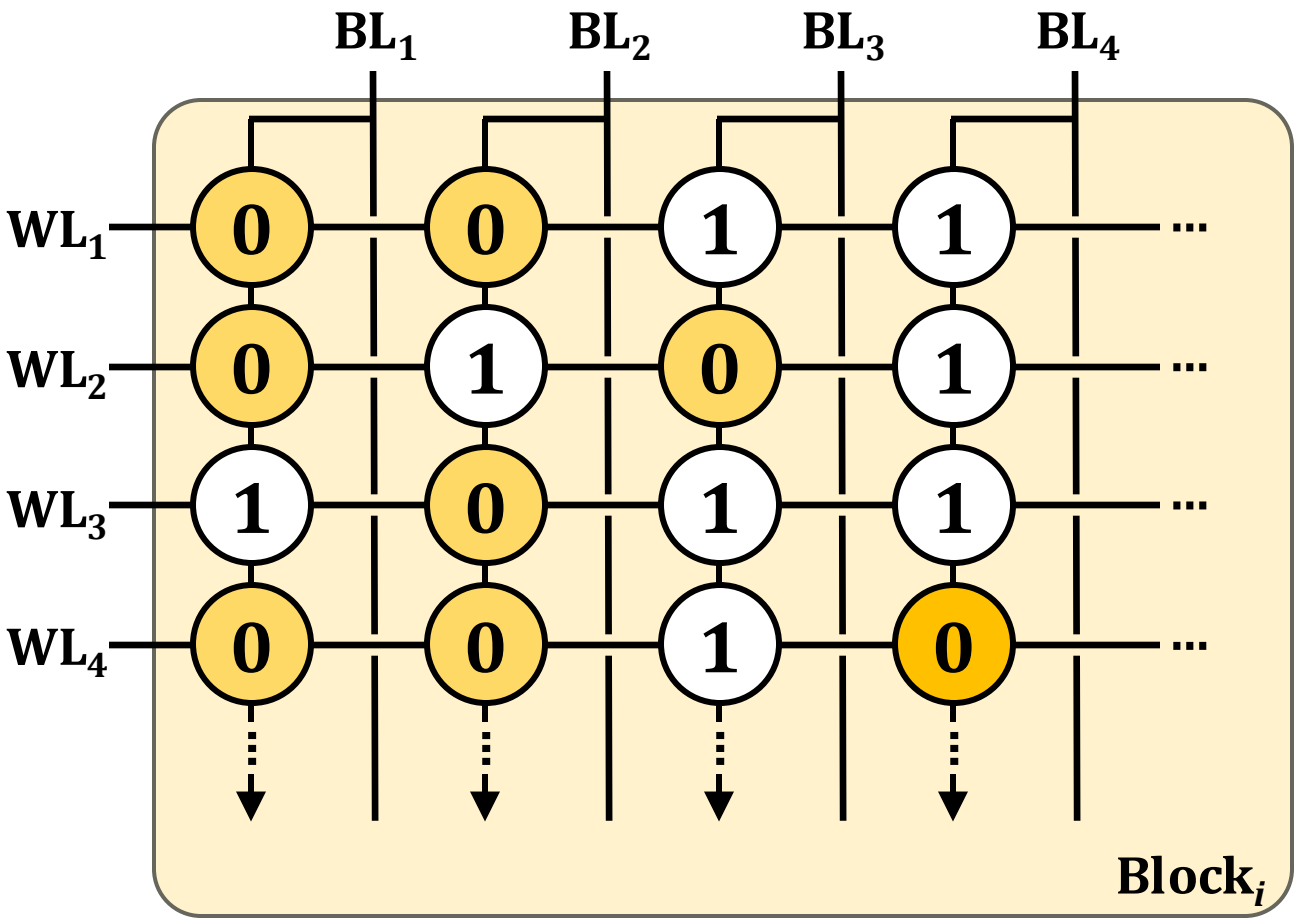
PUD in COTS DRAM:
Conclusion

- 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

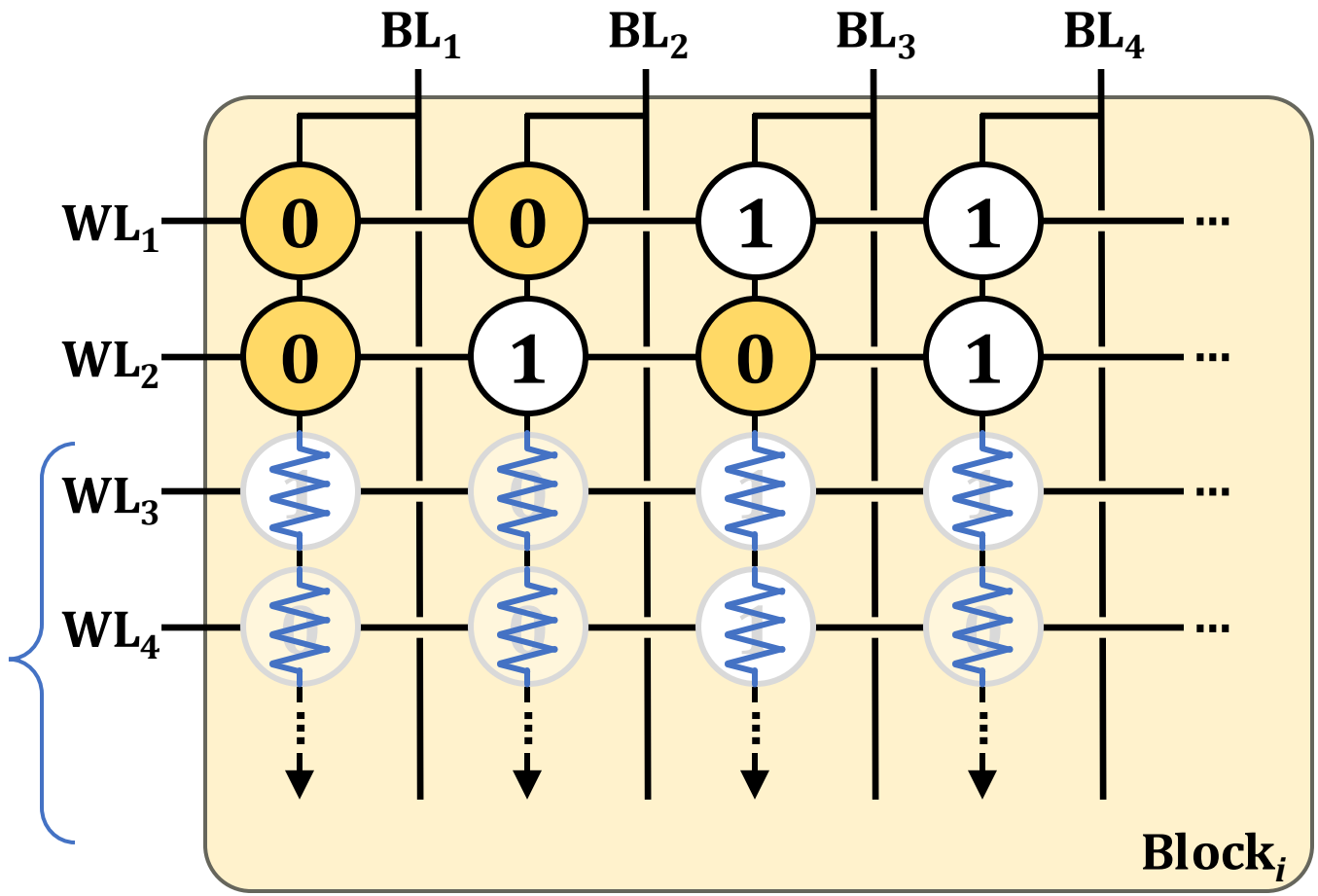
Multi-Wordline Sensing (MWS): Bitwise AND

- Intra-Block MWS:
  Simultaneously activates multiple WLs in the same block
  → Bitwise AND of the stored data in the WLs



Multi-Wordline Sensing (MWS): Bitwise AND

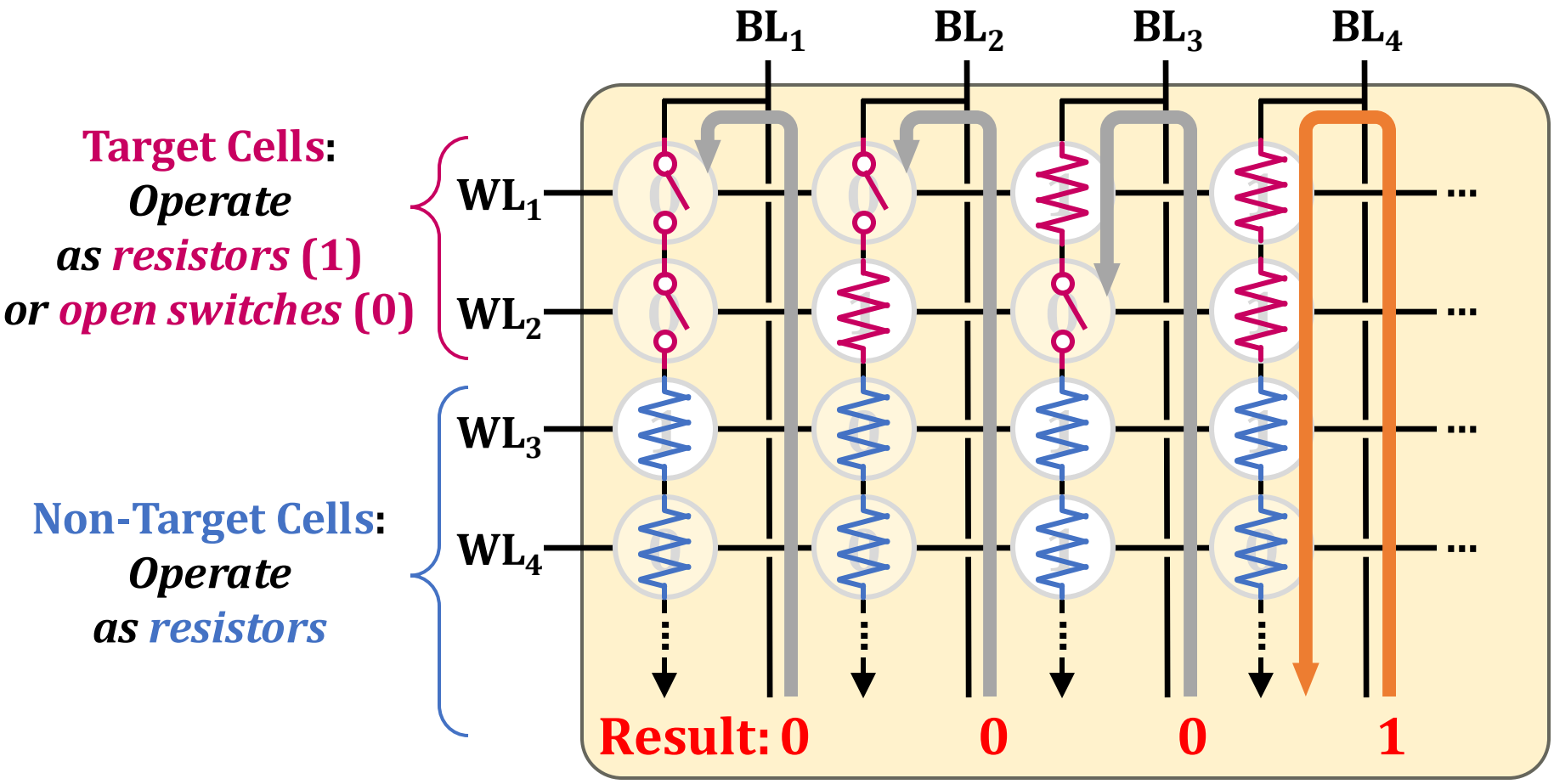
- Intra-Block MWS:
  Simultaneously activates multiple WLs in the same block
  → Bitwise AND of the stored data in the WLs



Non-Target Cells:
Operate as resistors

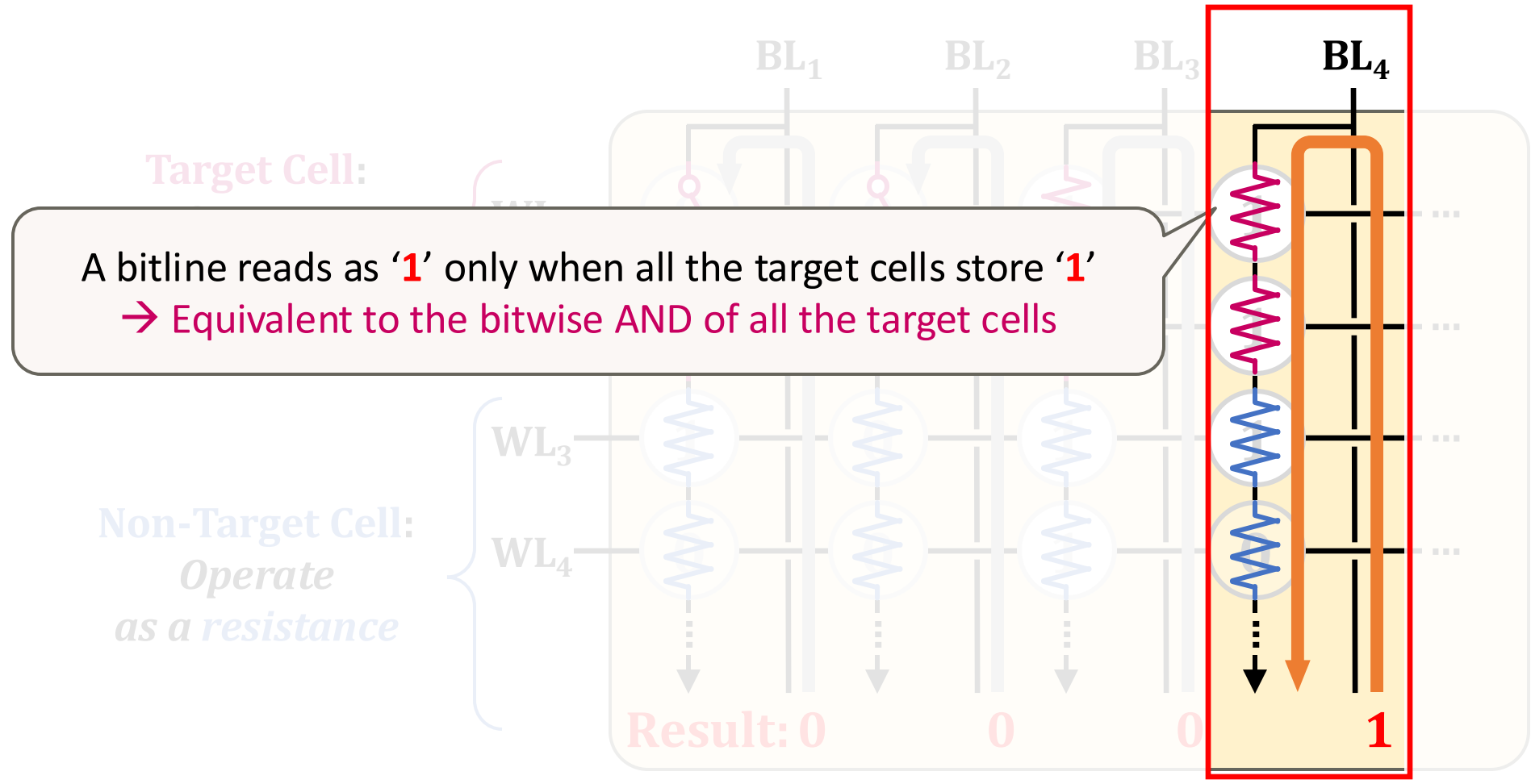
Multi-Wordline Sensing (MWS): Bitwise AND

- Intra-Block MWS:
  Simultaneously activates multiple WLs in the same block
  → Bitwise AND of the stored data in the WLs



Multi-Wordline Sensing (MWS): Bitwise AND

- Intra-Block MWS:
  Simultaneously activates multiple WLs in the same block
  → Bitwise AND of the stored data in the WLs

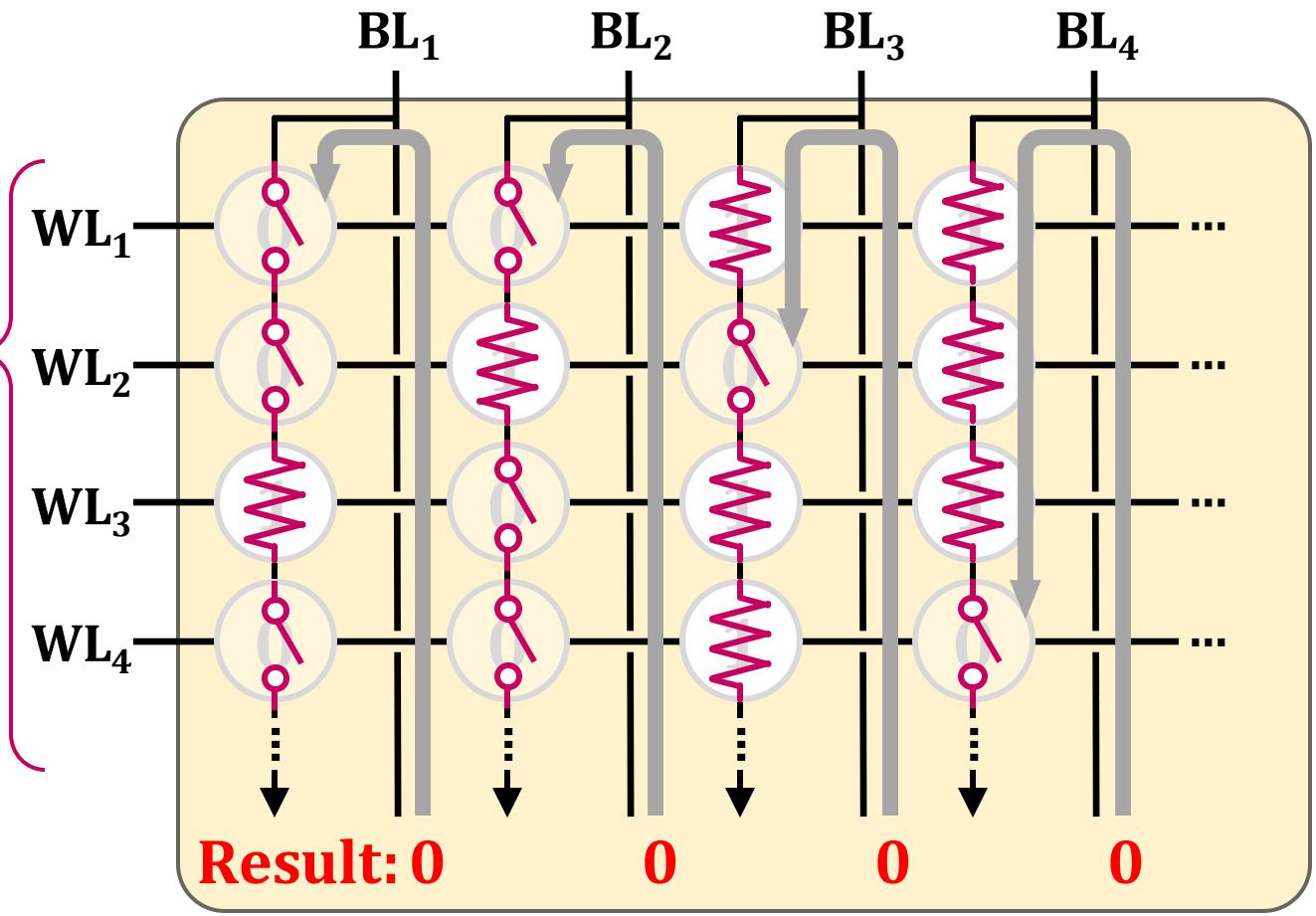


A bitline reads as '**1**' only when all the target cells store '**1**'
→ Equivalent to the bitwise AND of all the target cells

Multi-Wordline Sensing (MWS): Bitwise AND

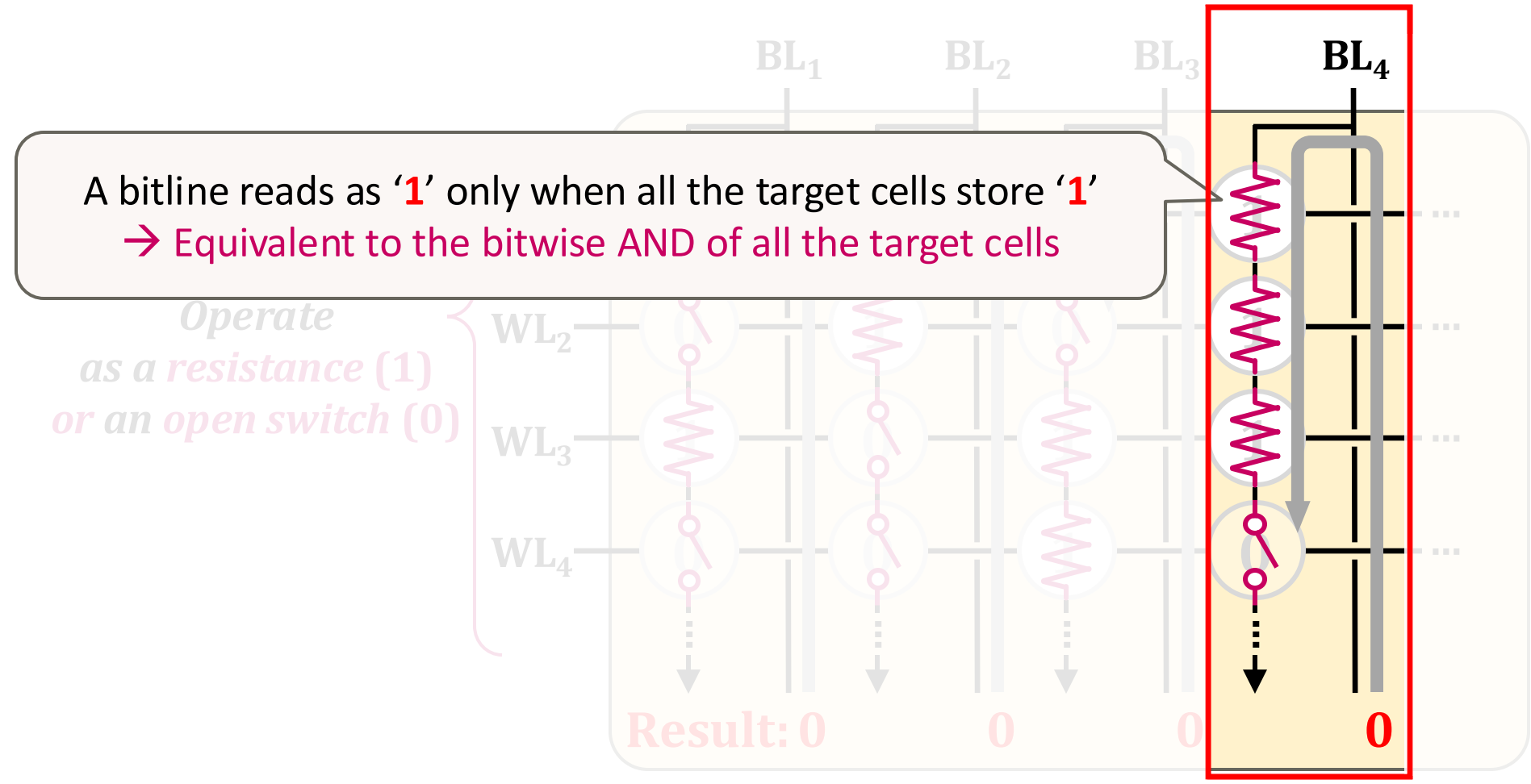
- Intra-Block MWS:
  Simultaneously activates multiple WLs in the same block
  → Bitwise AND of the stored data in the WLs

Target Cell:
Operate
as a resistance (1)
or an open switch (0)



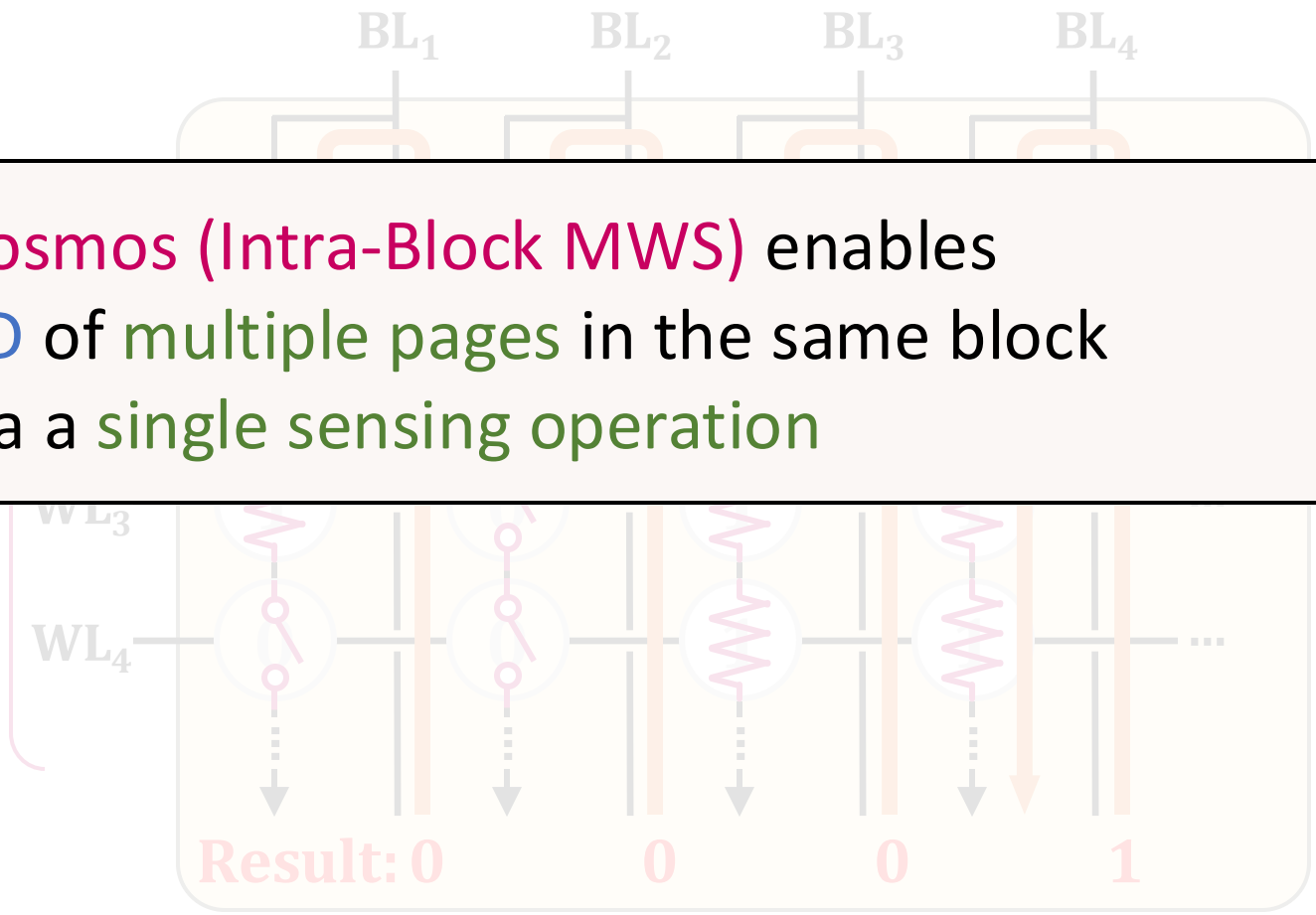
Multi-Wordline Sensing (MWS): Bitwise AND

- Intra-Block MWS:
  Simultaneously activates multiple WLs in the same block
  → Bitwise AND of the stored data in the WLs



Multi-Wordline Sensing (MWS): Bitwise AND

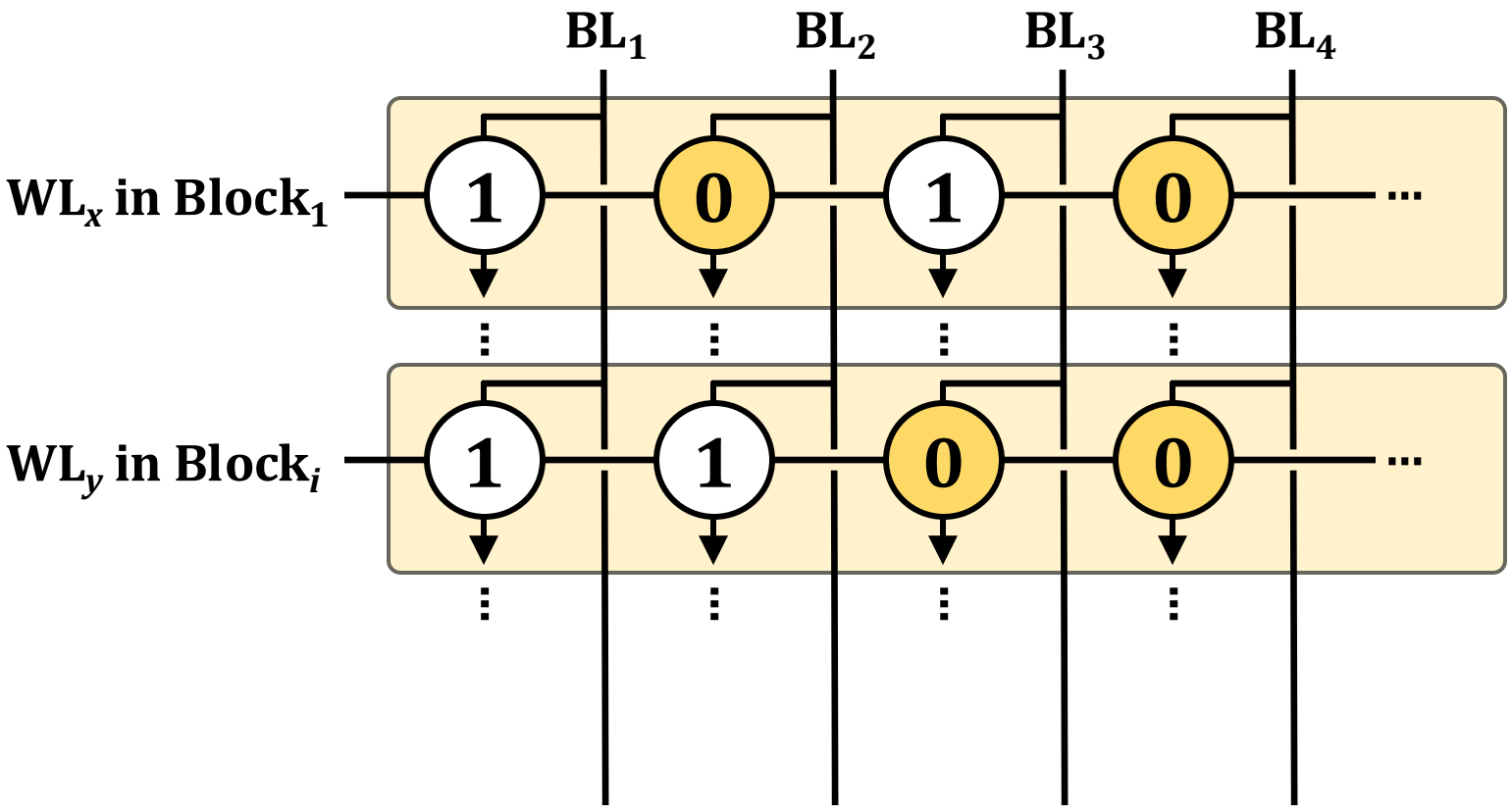
- Intra-Block MWS:
  Simultaneously activates multiple WLs in the same block
  → Bitwise AND of the stored data in the WLs



Flash-Cosmos (Intra-Block MWS) enables
bitwise AND of multiple pages in the same block
via a single sensing operation

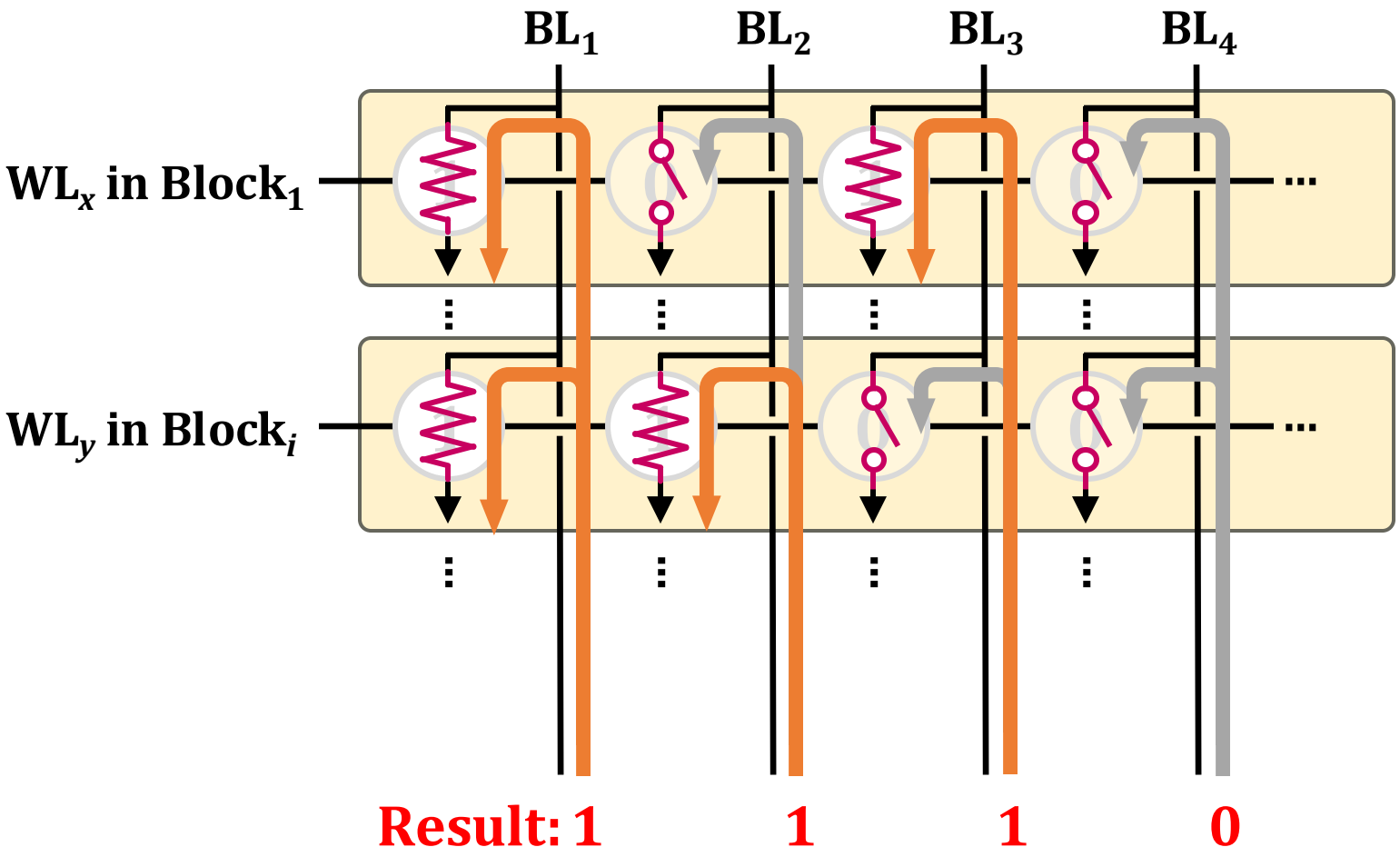
Multi-Wordline Sensing (MWS): Bitwise OR

- Inter-Block MWS:
  Simultaneously activates multiple WLs in different blocks
  → Bitwise OR of the stored data in the WLs



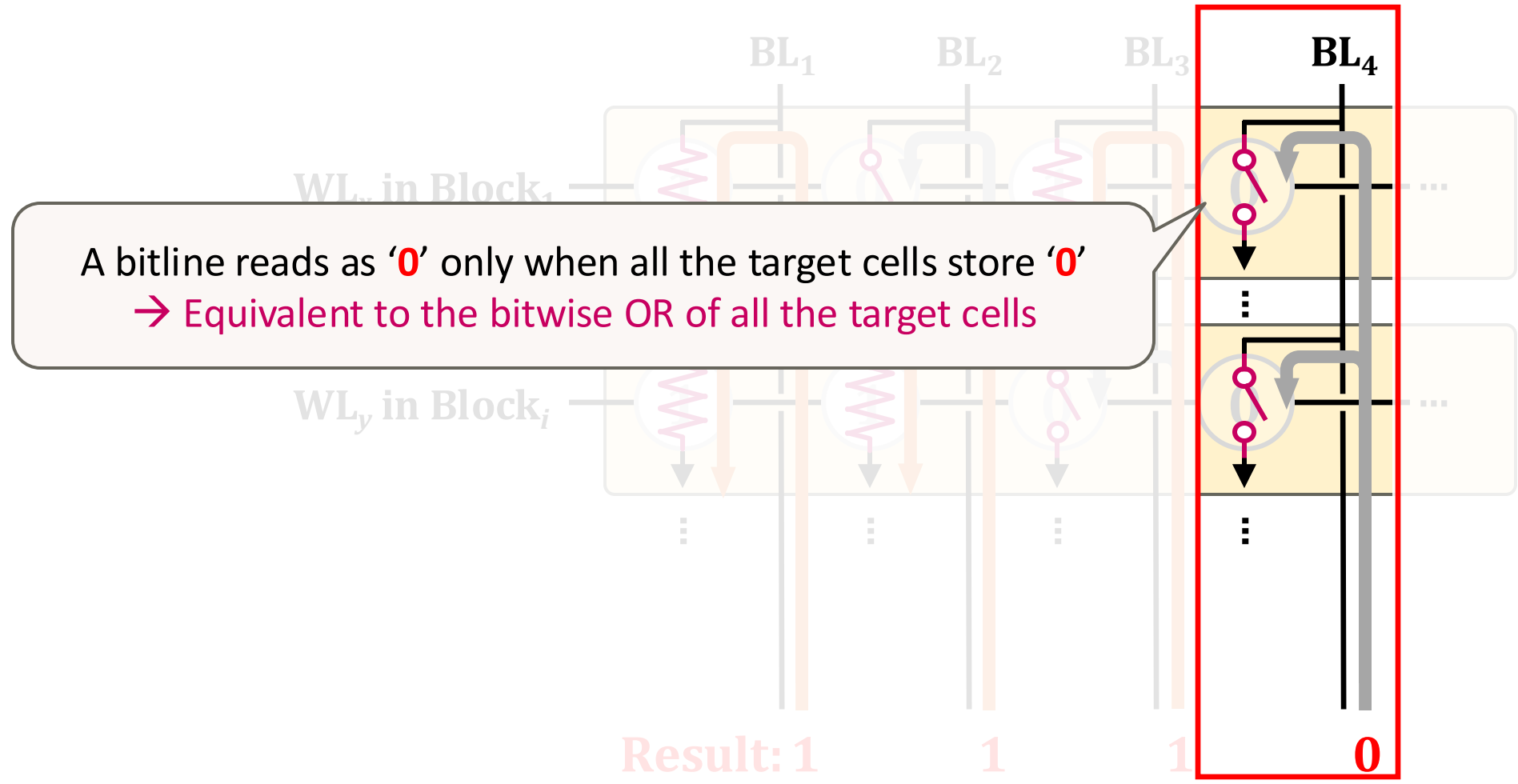
Multi-Wordline Sensing (MWS): Bitwise OR

- Inter-Block MWS:
  Simultaneously activates multiple WLs in different blocks
  → Bitwise OR of the stored data in the WLs



Multi-Wordline Sensing (MWS): Bitwise OR

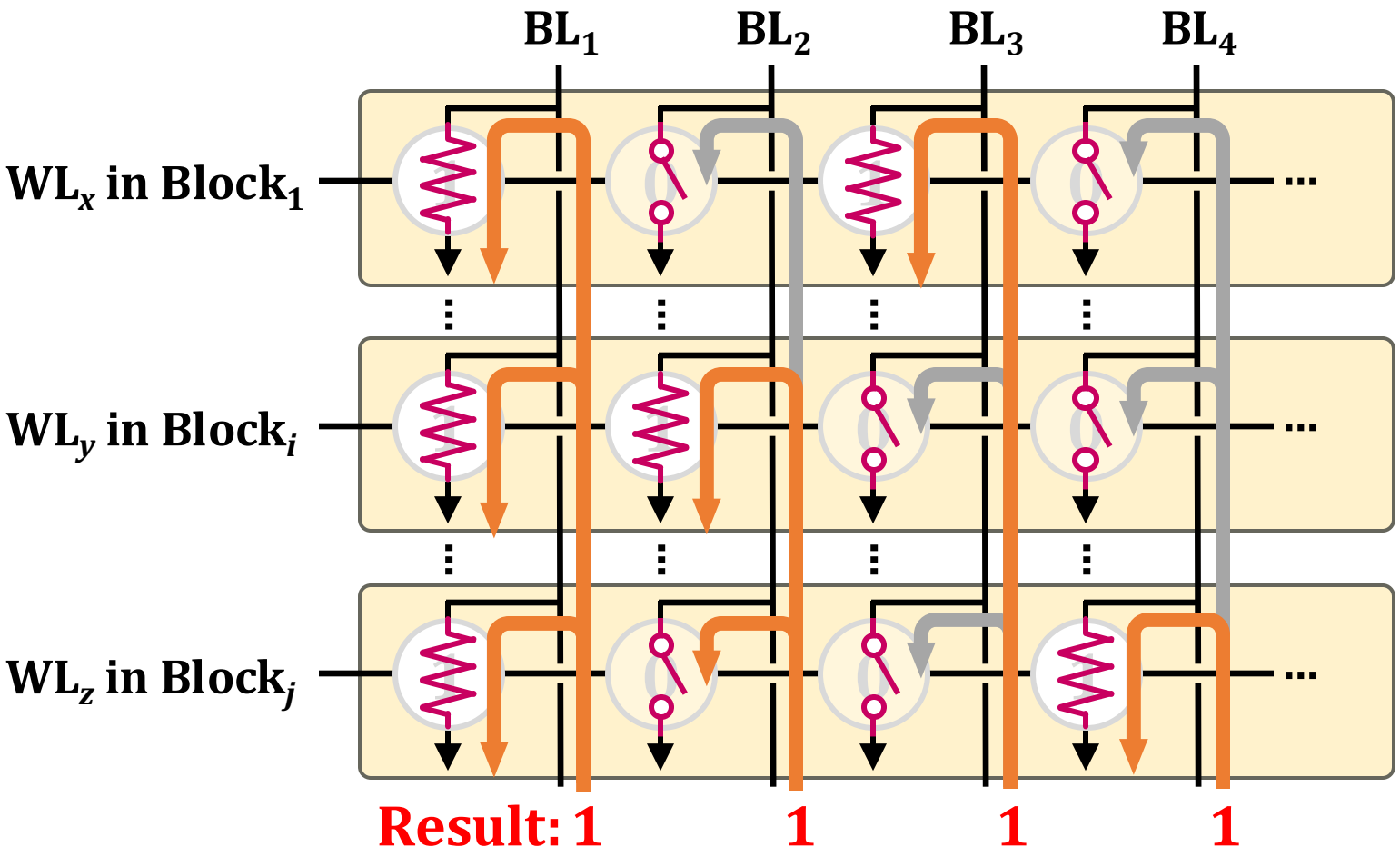
- Inter-Block MWS:
  Simultaneously activates multiple WLs in different blocks
  → Bitwise OR of the stored data in the WLs



A bitline reads as '**0**' only when all the target cells store '**0**'
→ Equivalent to the bitwise OR of all the target cells

Multi-Wordline Sensing (MWS): Bitwise OR

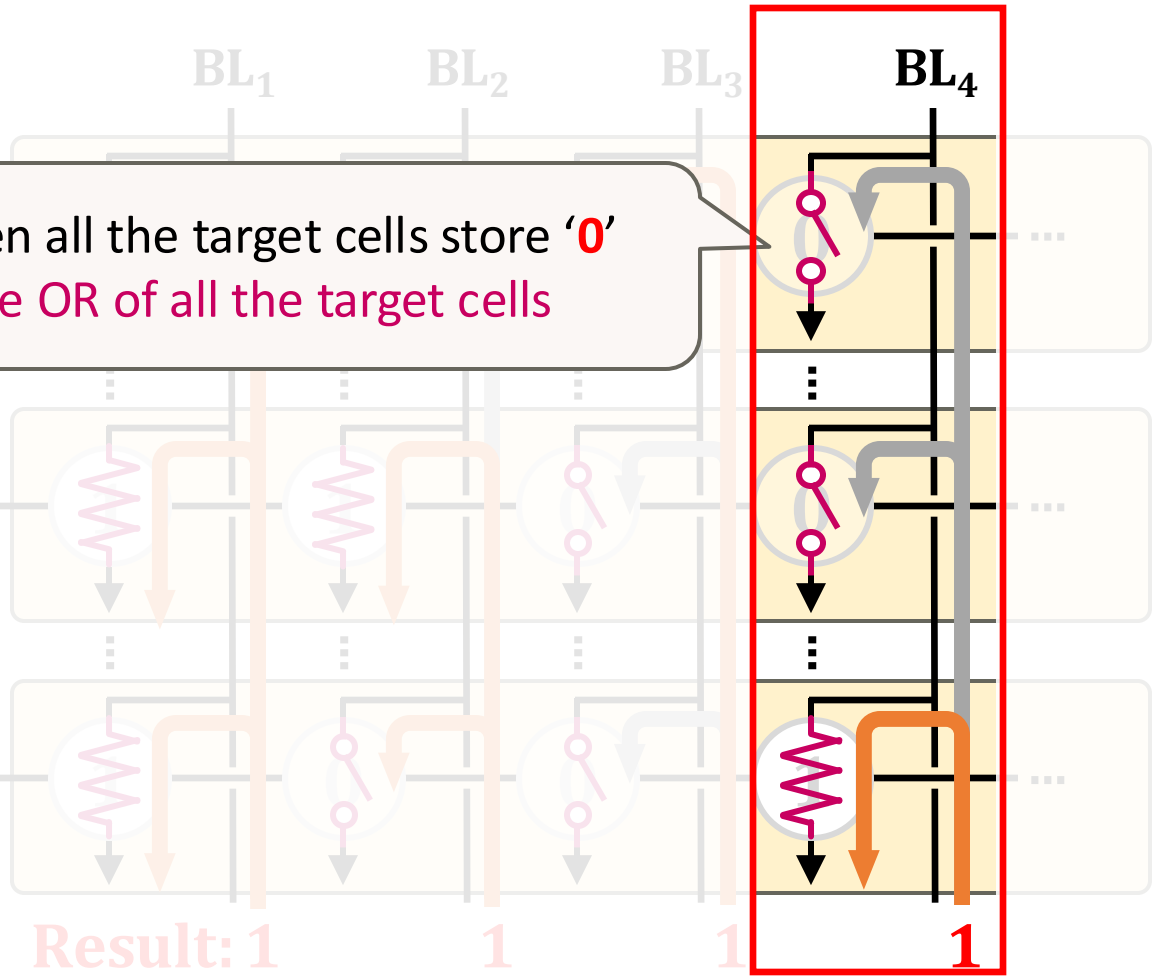
- Inter-Block MWS:
  Simultaneously activates multiple WLs in different blocks
  → Bitwise OR of the stored data in the WLs



Multi-Wordline Sensing (MWS): Bitwise OR

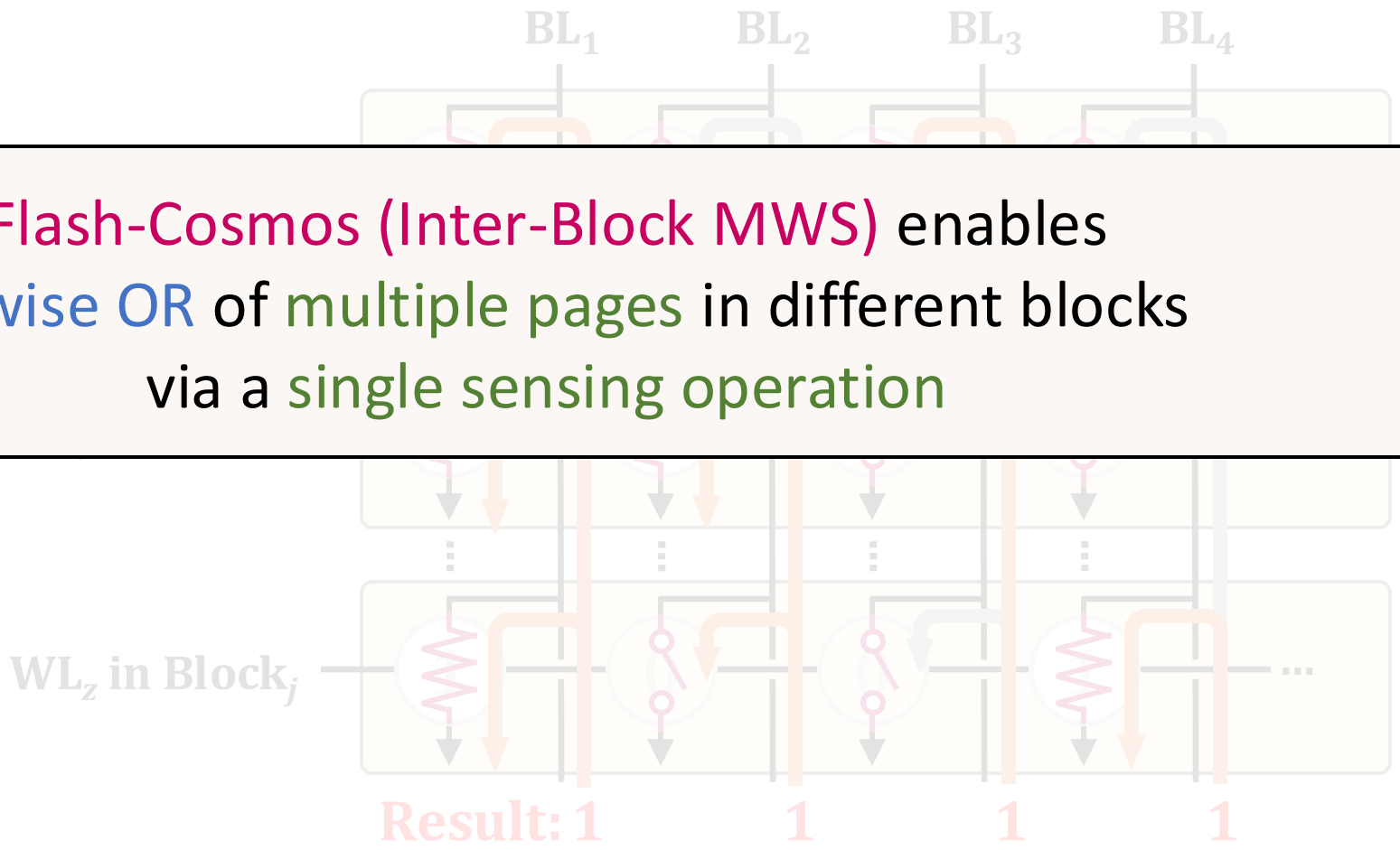
- Inter-Block MWS:
  Simultaneously activates multiple WLs in different blocks
  → Bitwise OR of the stored data in the WLs

A bitline reads as '**0**' only when all the target cells store '**0**'
→ Equivalent to the bitwise OR of all the target cells



Multi-Wordline Sensing (MWS): Bitwise OR

- Inter-Block MWS:
  Simultaneously activates multiple WLs in different blocks
  → Bitwise OR of the stored data in the WLs



Flash-Cosmos (Inter-Block MWS) enables
bitwise OR of multiple pages in different blocks
via a single sensing operation

Other Types of Bitwise Operations

Flash-Cosmos also enables
other types of bitwise operations
(NOT/NAND/NOR/XOR/XNOR)
leveraging existing features of NAND flash memory

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/abs/2209.05566.pdf