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

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

Executive Summary

Problem: Processing-Using-DRAM (PUD) suffers from three issues caused by DRAM's large and rigid access granularity

- Underutilization due to data parallelism variation in (and across) applications
- Limited computation support due to a lack of interconnects
- Challenging programming model due to a lack of compilers

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

Key Mechanism: MIMDRAM, a hardware/software co-design PUD system

- **Key idea**: leverage fine-grained DRAM for PUD operation
- **HW**: - simple changes to the DRAM array, enabling concurrent PUD operations
  - low-cost interconnects at the DRAM peripherals for data reduction
- **SW**: - compiler and OS support to generate and map PUD instructions

Key Results: MIMDRAM achieves

- **14.3x**, **30.6x**, and **6.8x** the energy efficiency of state-of-the-art PUD systems, a high-end CPU and GPU, respectively
- Small area cost to a DRAM chip (**1.11%**) and CPU die (**0.6%**)

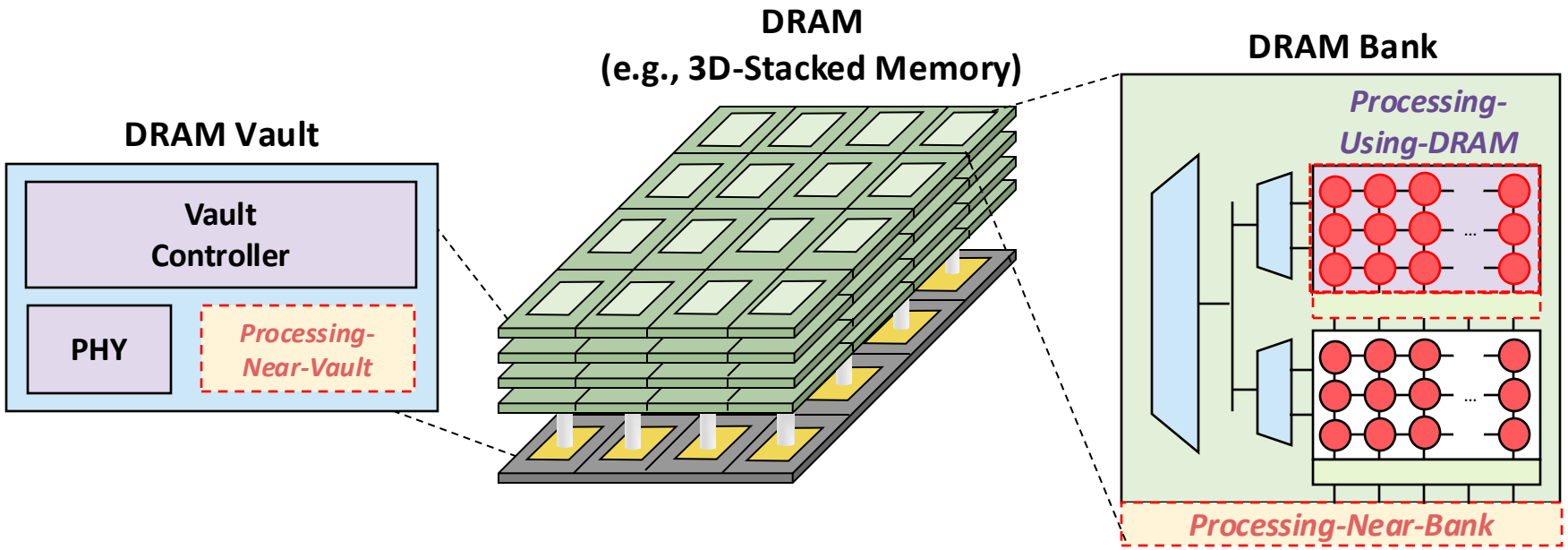
Outline

Processing-in-Memory:
Overview

Two main approaches for Processing-in-Memory:

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

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

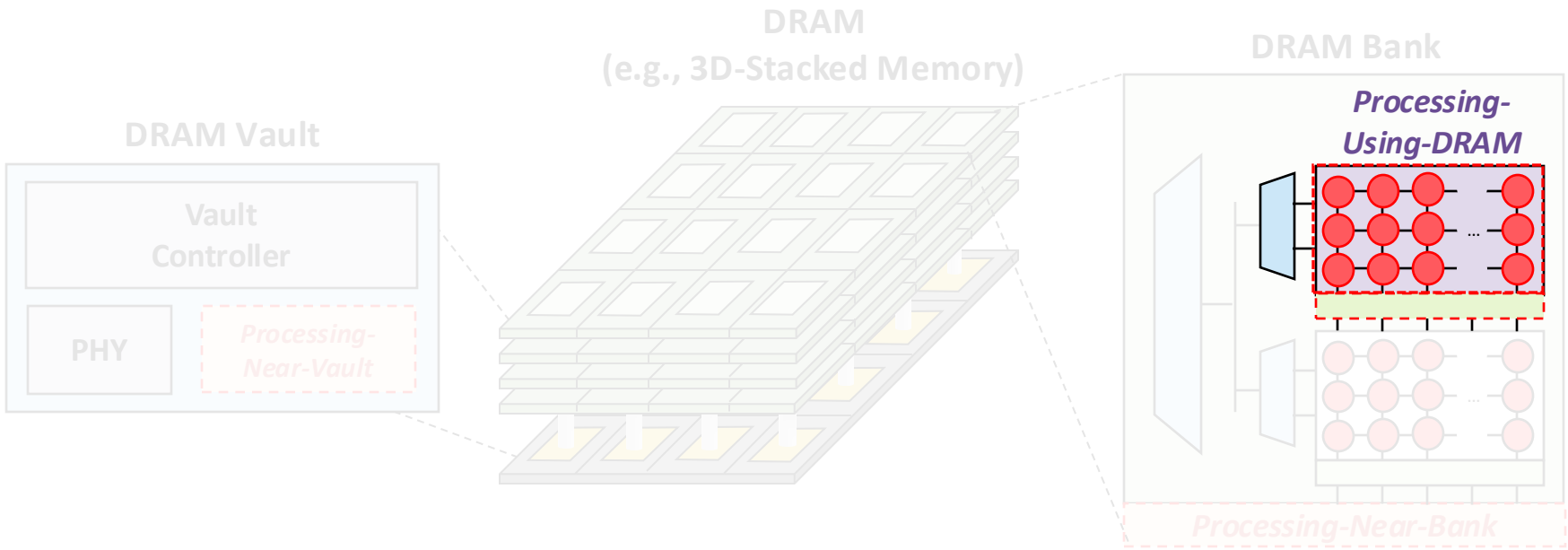


Processing-in-Memory:
Overview

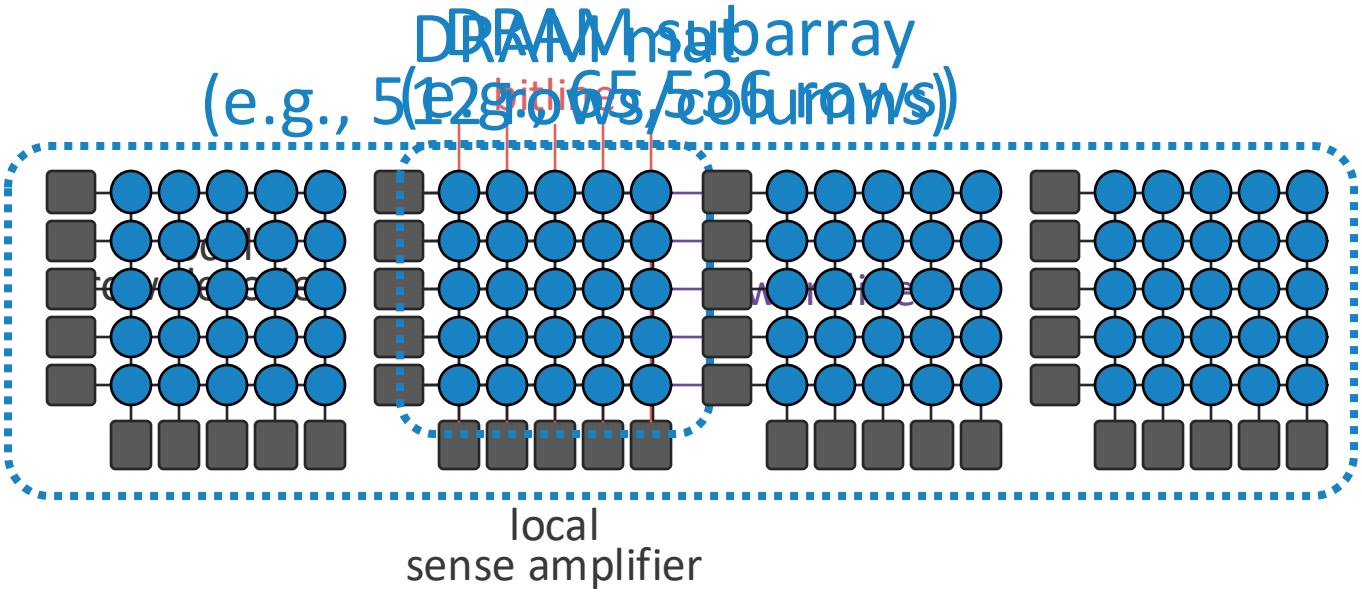
Two main approaches for Processing-in-Memory:

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

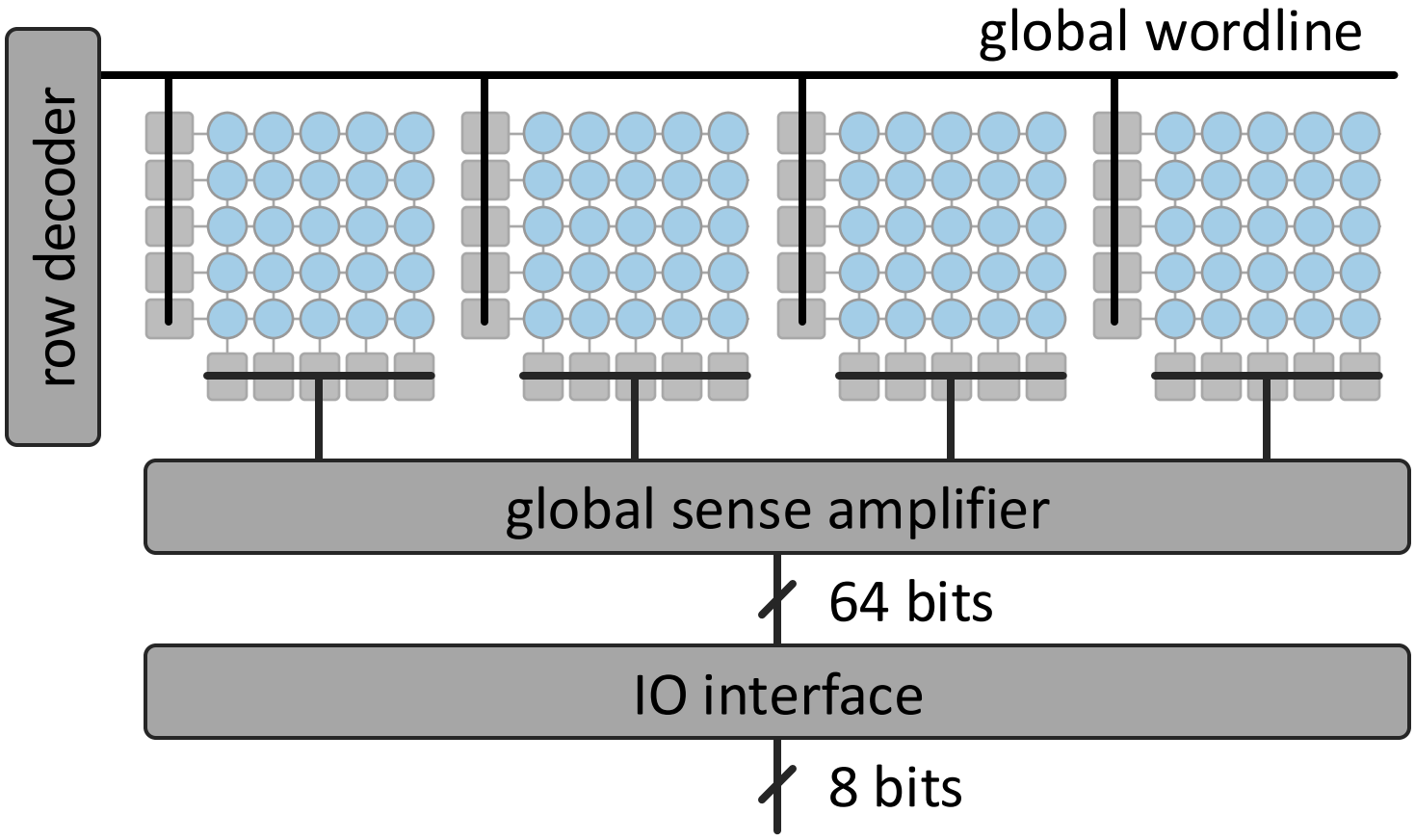
2 **Processing-Using-Memory**: uses the operational principles of memory cells to perform computation



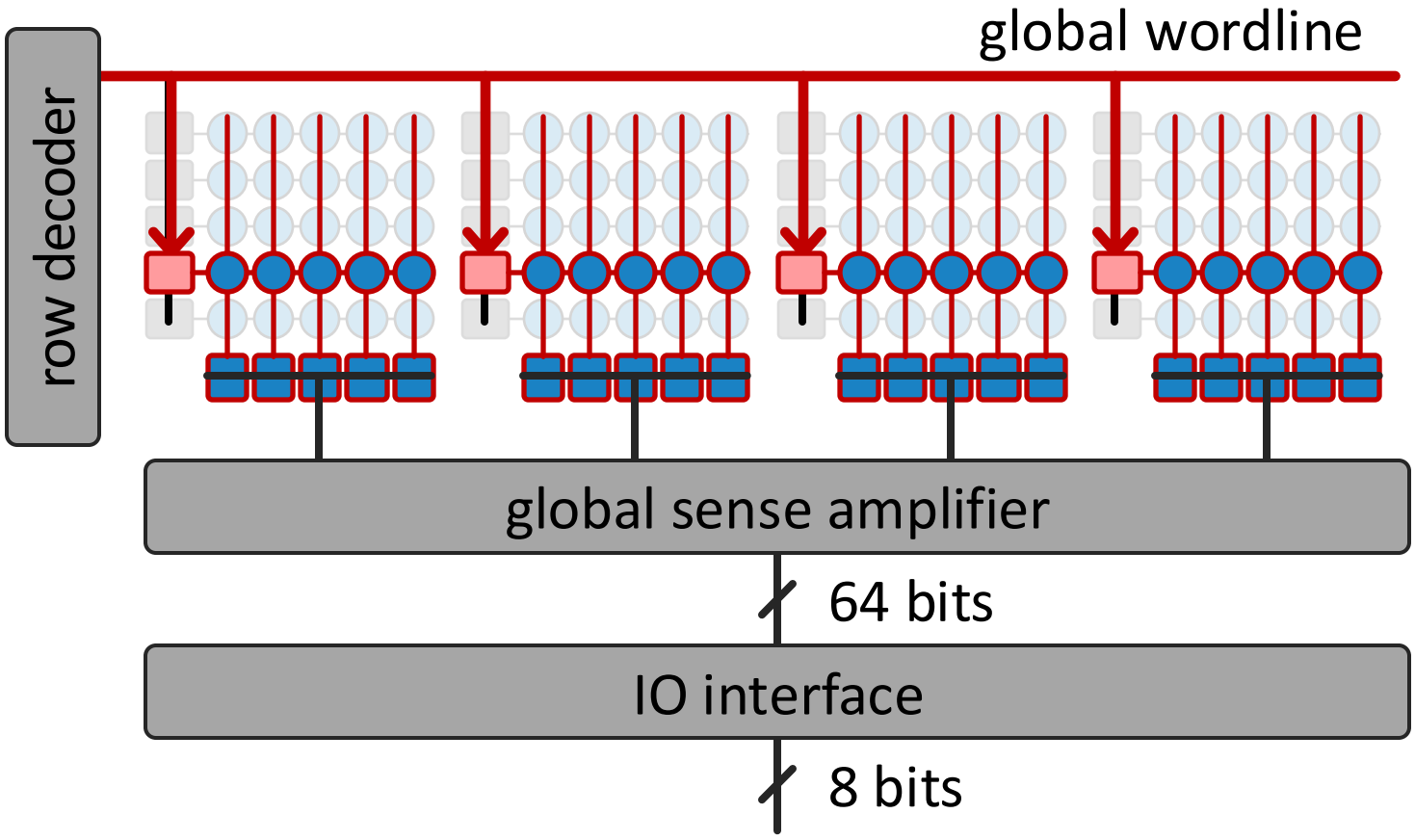
Background:
DRAM Hierarchical Organization



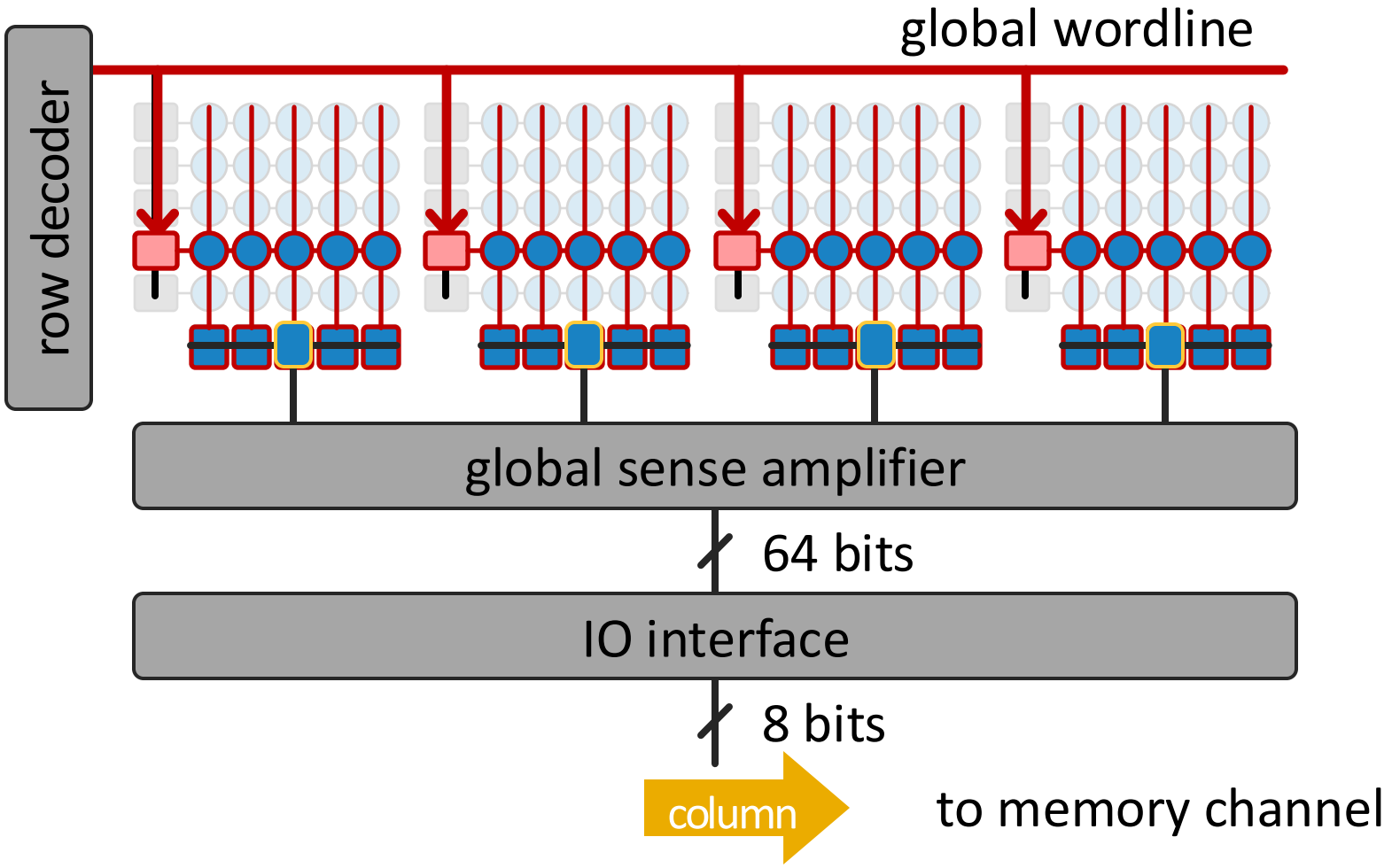
Background:
DRAM Hierarchical Organization



Background:
DRAM Operation – Row Access (ACTIVATE)

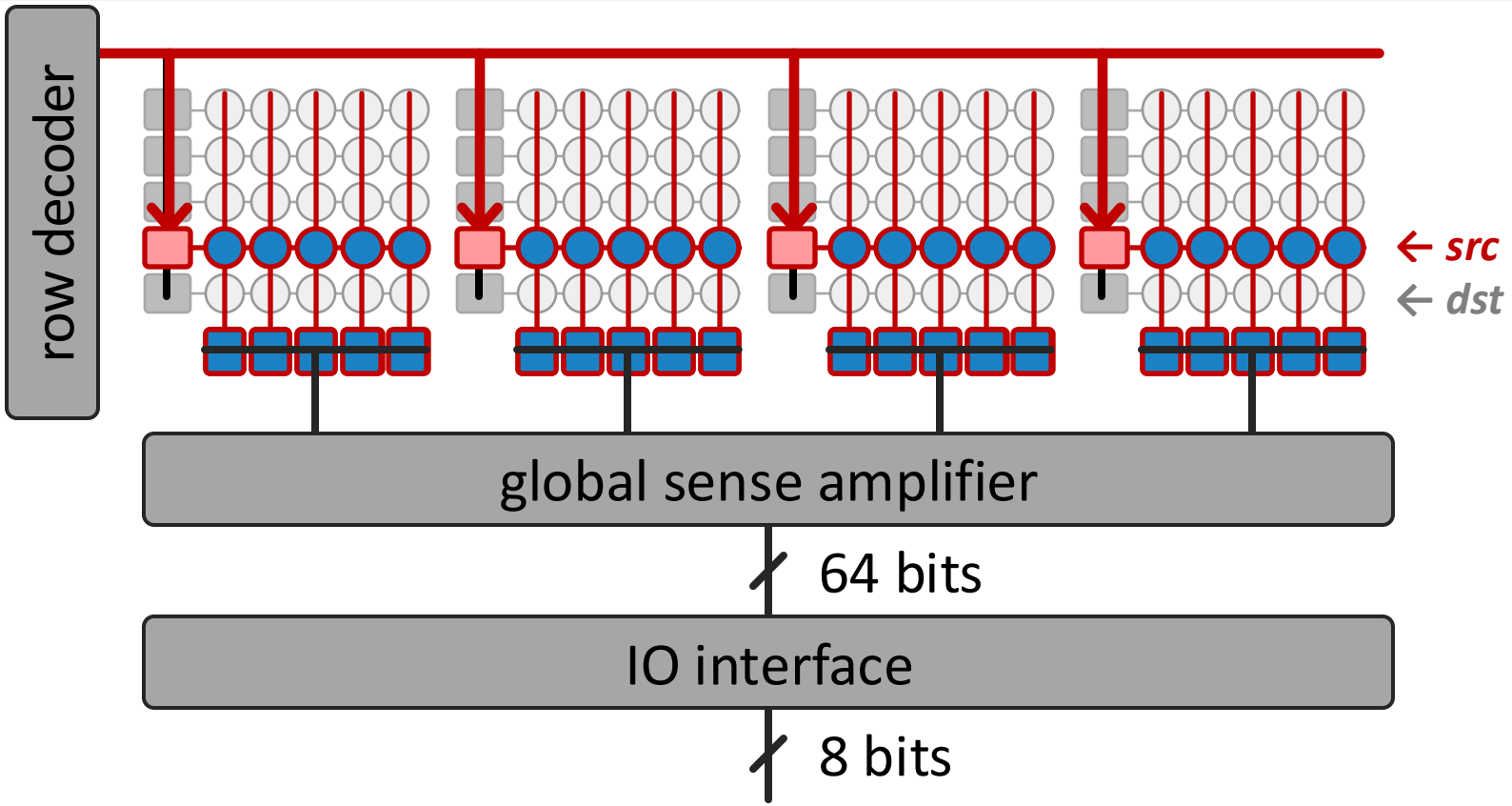


Background:
DRAM Operation – Column Access (READ)



Background:
In-DRAM Row Copy

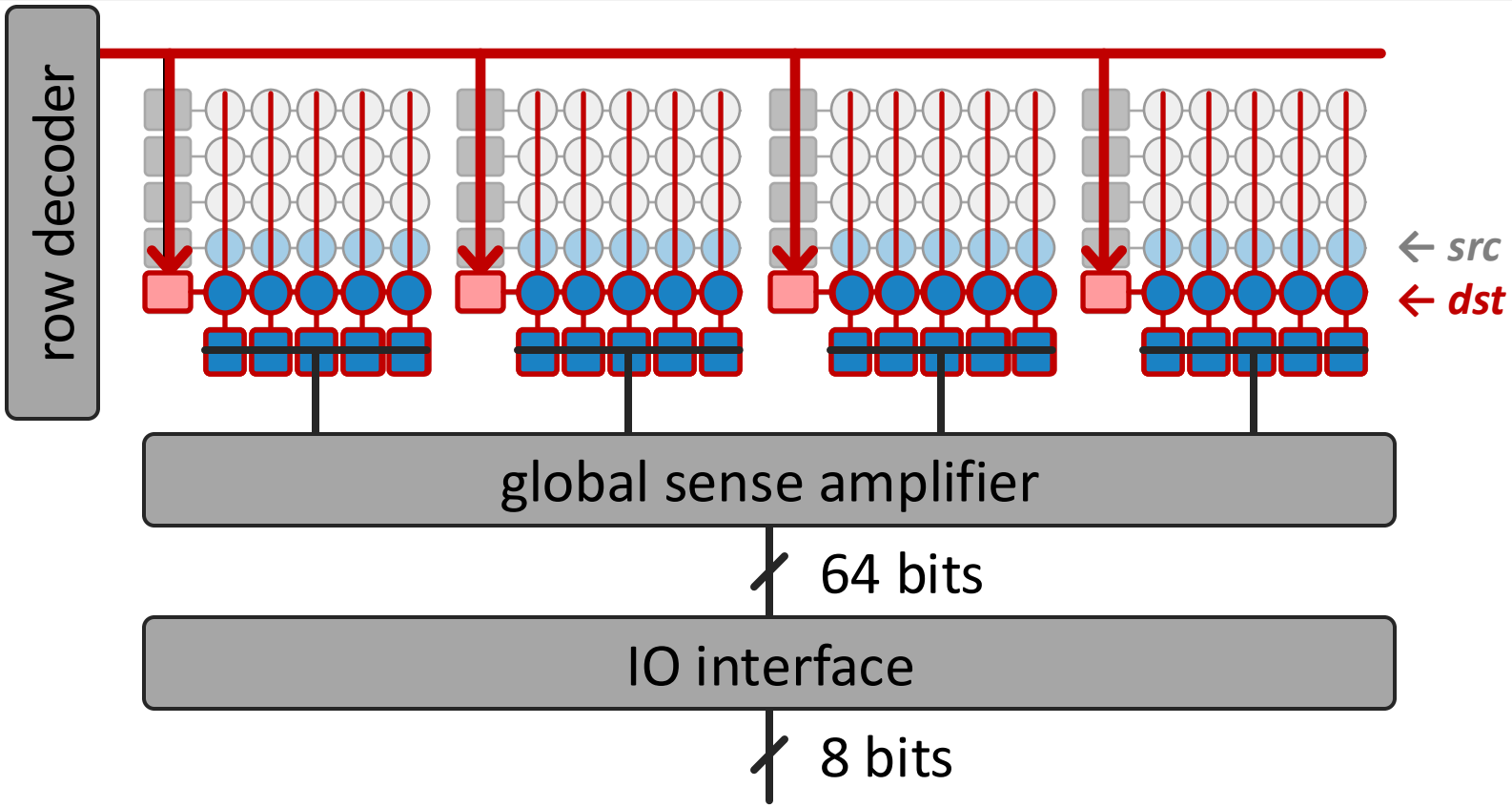
In-DRAM row copy is performed by
issuing back-to-back ACTIVATES to the DRAM



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

Background:
In-DRAM Row Copy

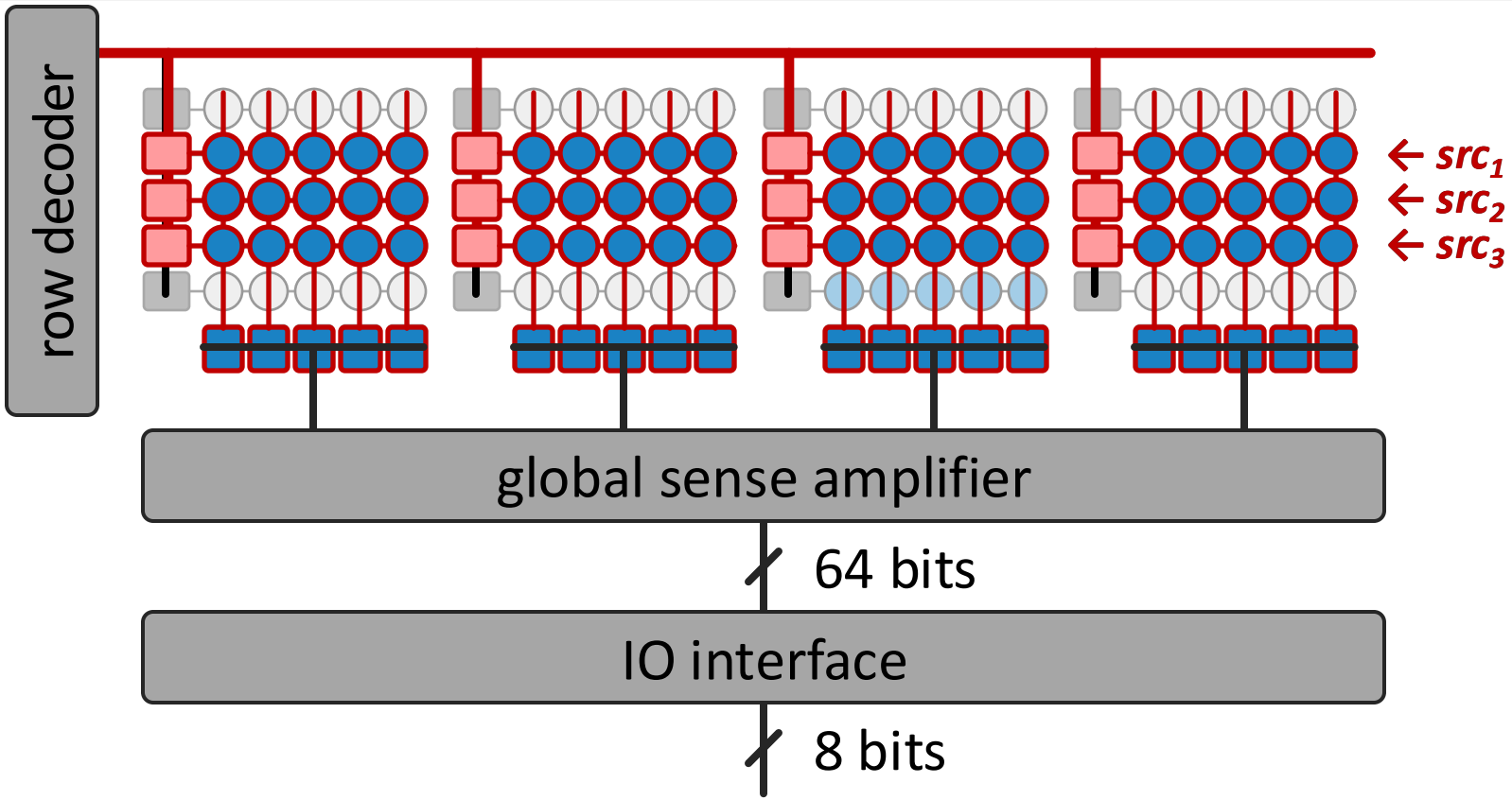
In-DRAM row copy is performed by
issuing back-to-back ACTIVATES to the DRAM



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

Background:
In-DRAM Majority Operations

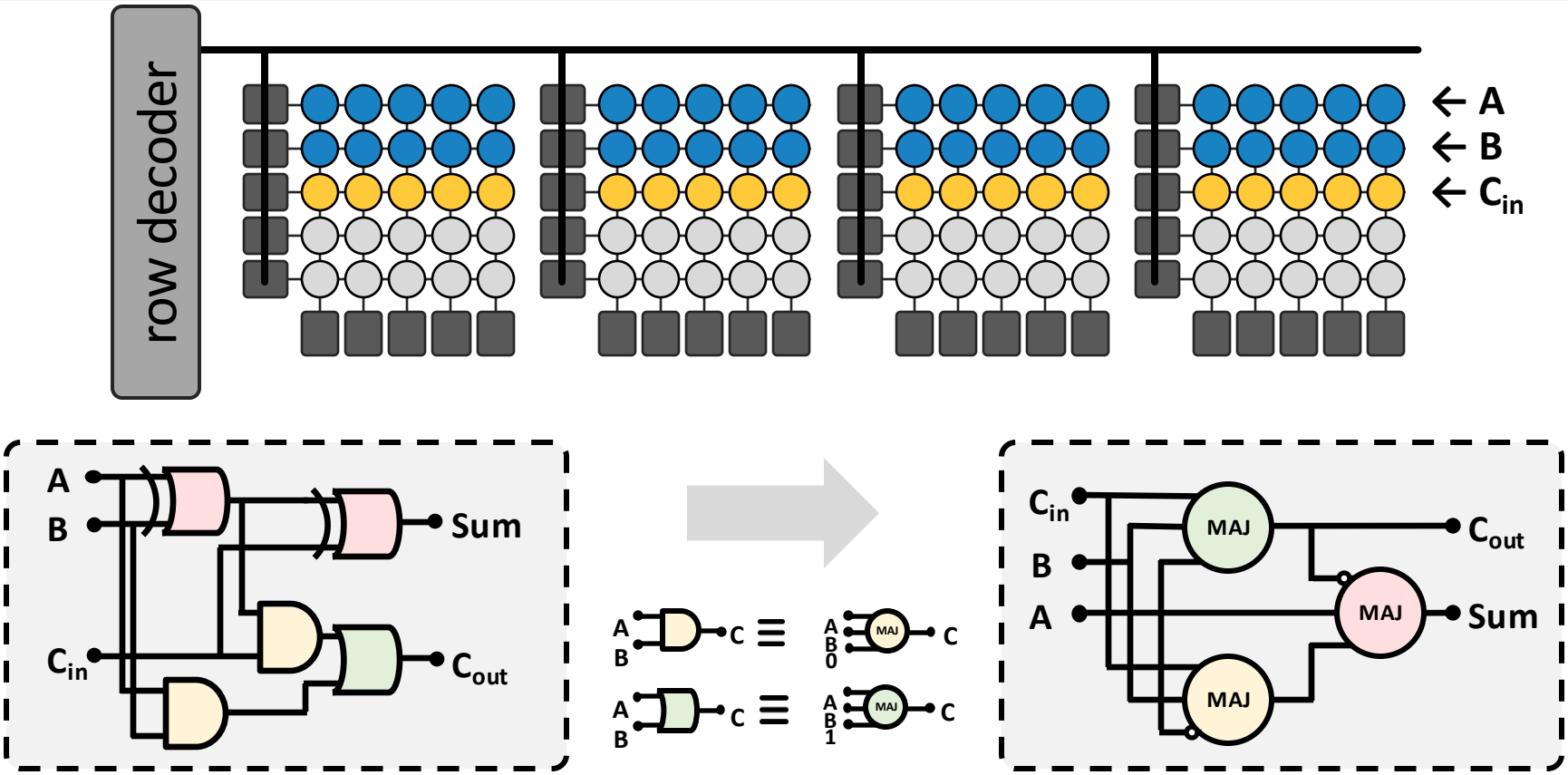
In-DRAM majority is performed by
simultaneously activating three DRAM rows



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

Background:
Bulk Bitwise Arithmetic Operations

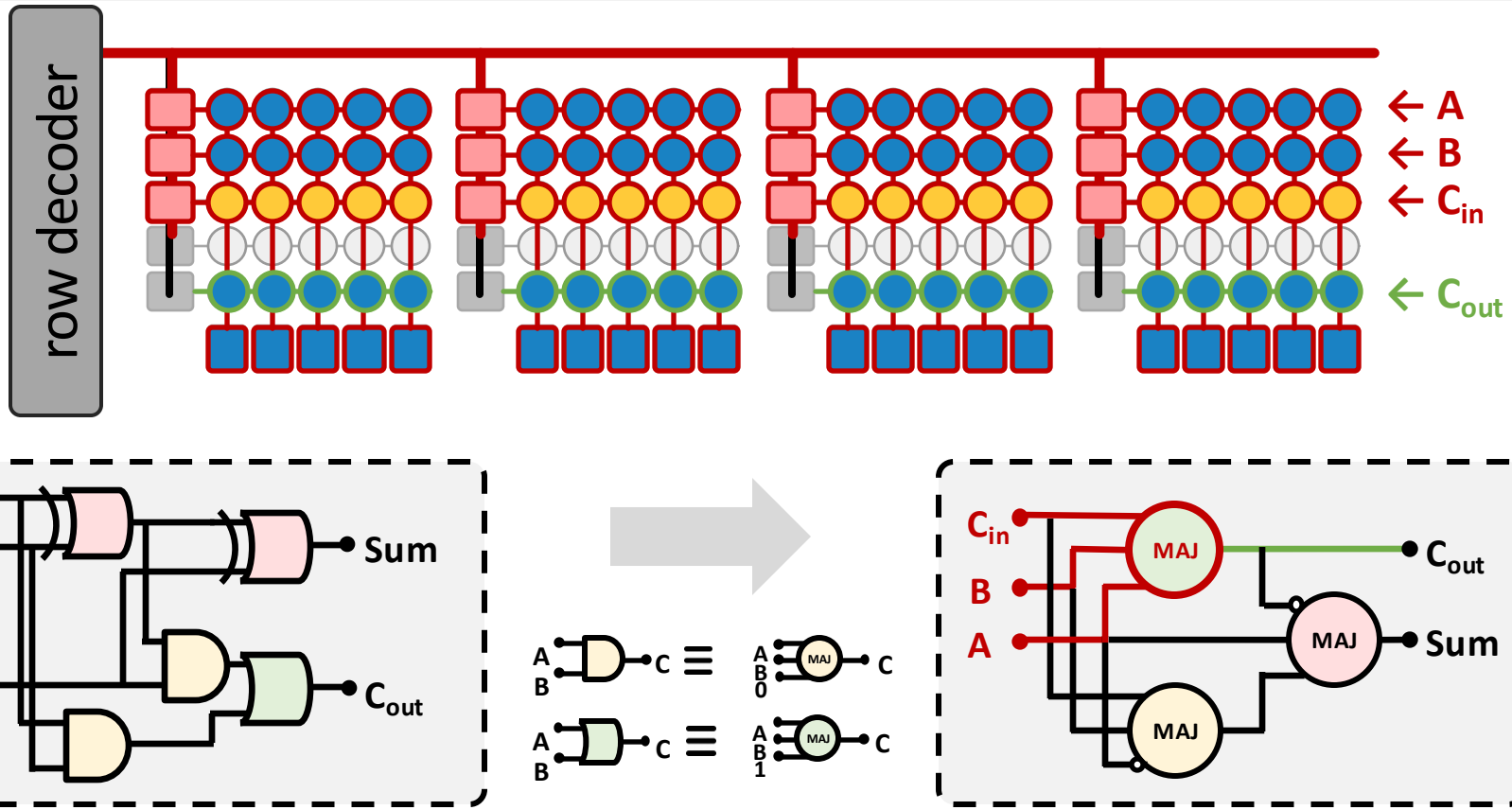
Bulk bitwise arithmetic can be performed by
orchestrating in-DRAM row copy and majority operations



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

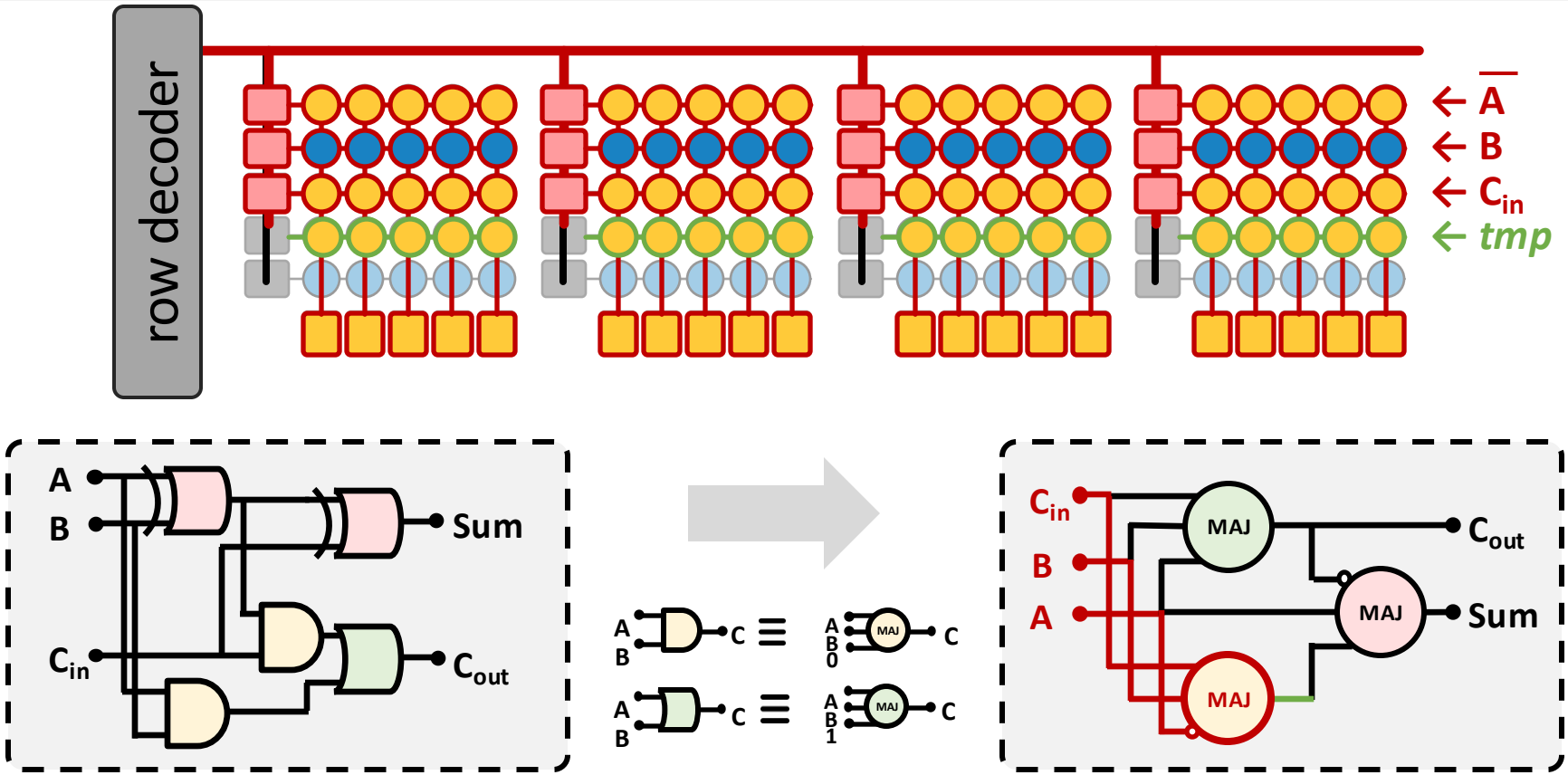
Bulk bitwise arithmetic can be performed by
orchestrating in-DRAM row copy and majority operations



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

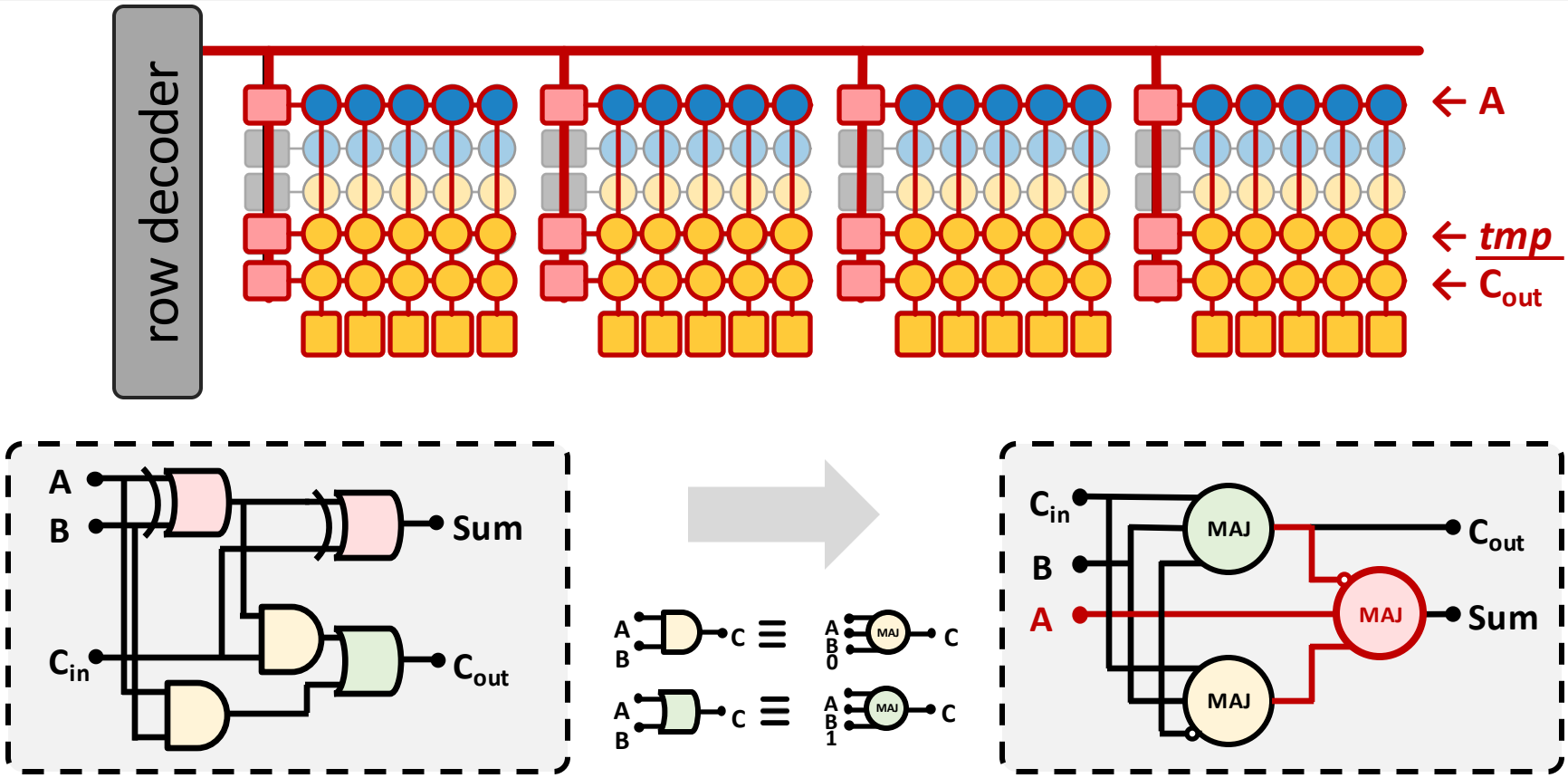
Bulk bitwise arithmetic can be performed by
orchestrating in-DRAM row copy and majority operations



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

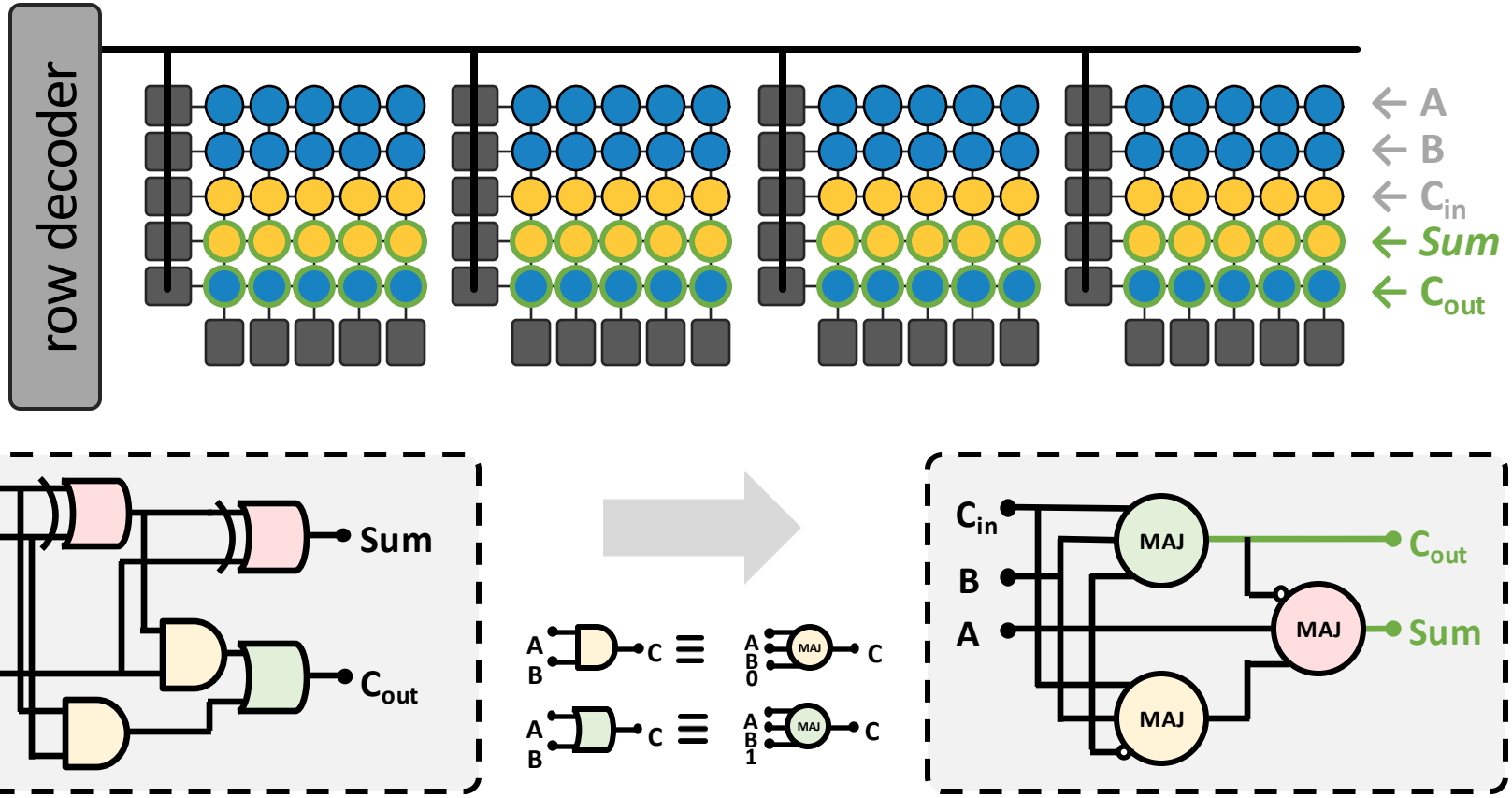
Bulk bitwise arithmetic can be performed by
orchestrating in-DRAM row copy and majority operations



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

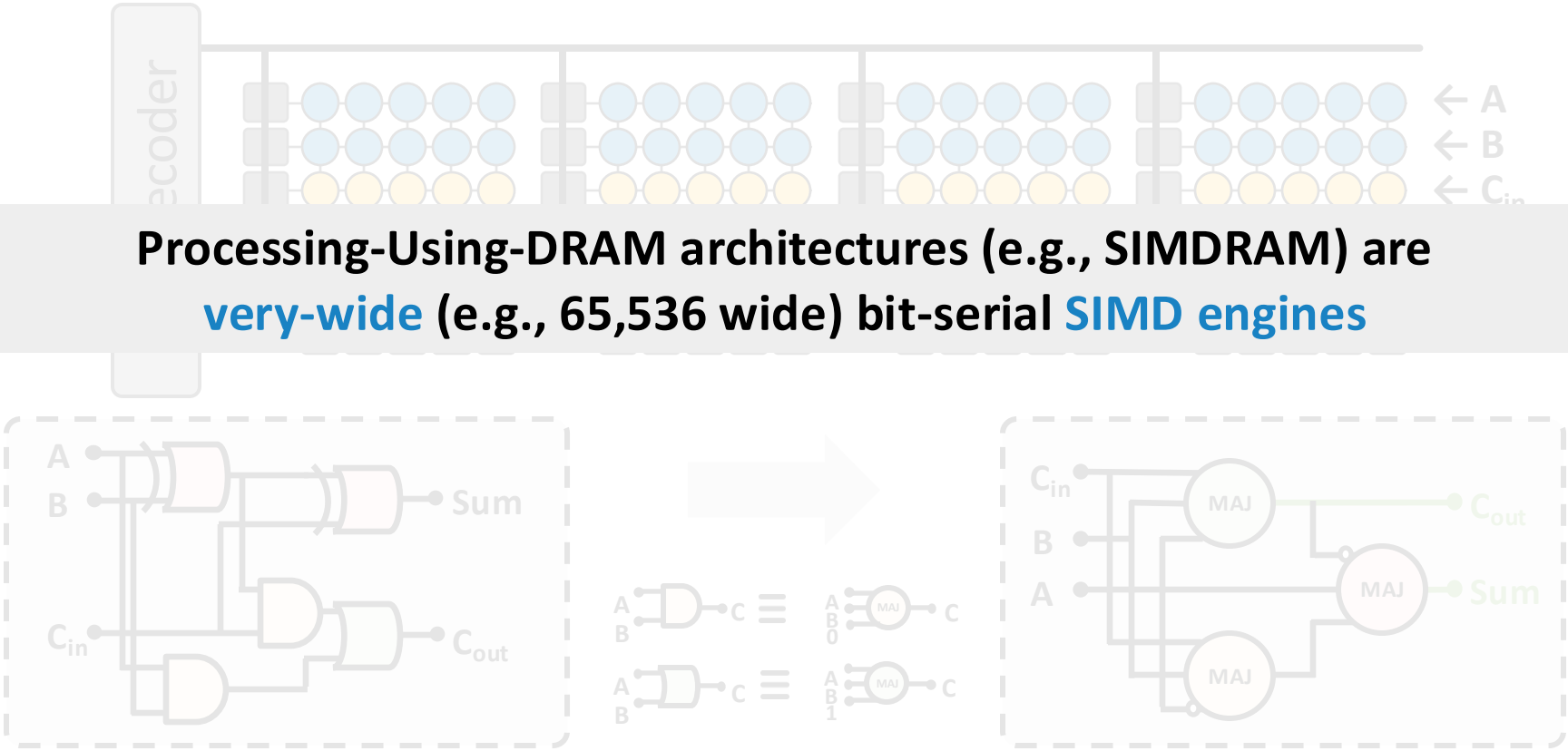
Bulk bitwise arithmetic can be performed by
orchestrating in-DRAM row copy and majority operations



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

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

Outline

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

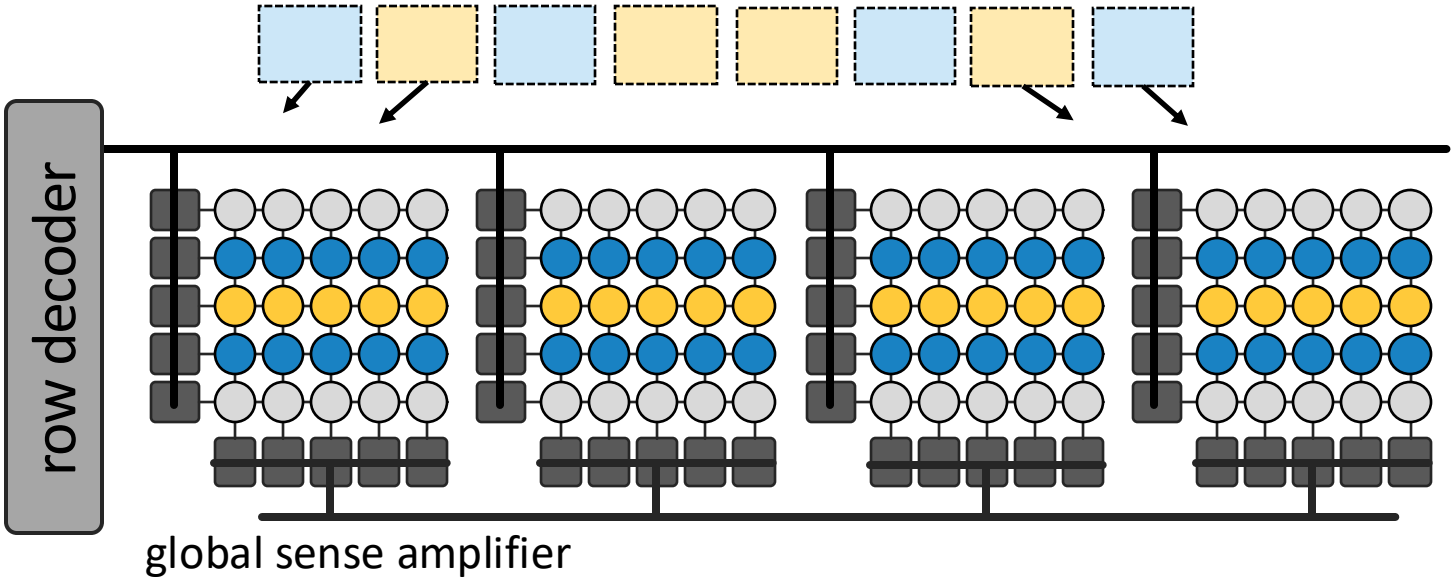
Limitations of PUD Systems:
Underutilization of SIMD Lanes (I)

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

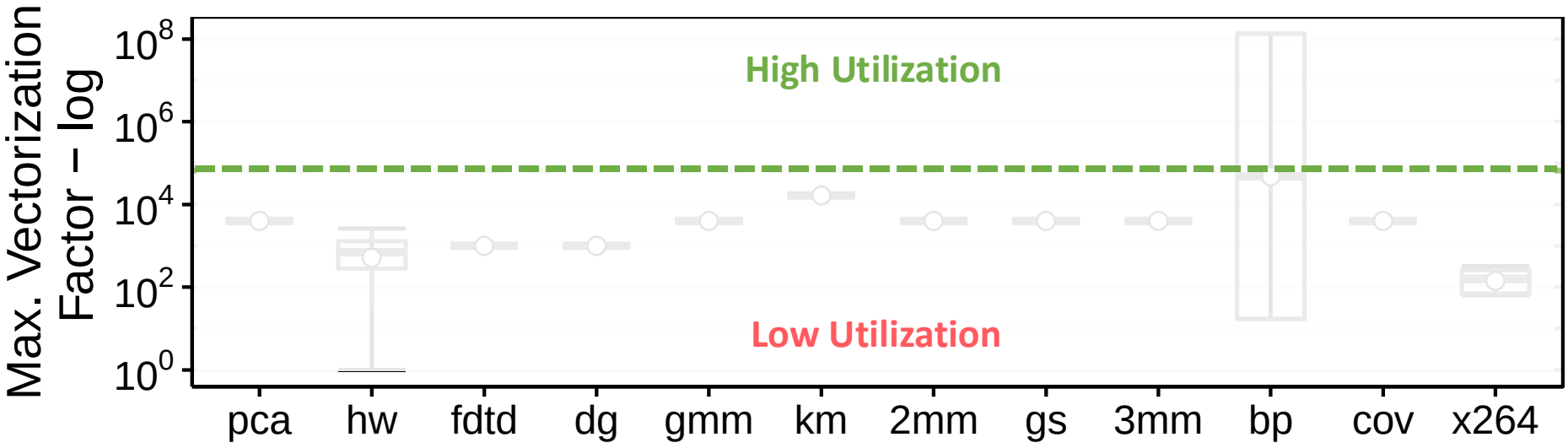


Ideal maximum vectorization factor = # DRAM columns (e.g., 65,536)

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

Application Analysis:

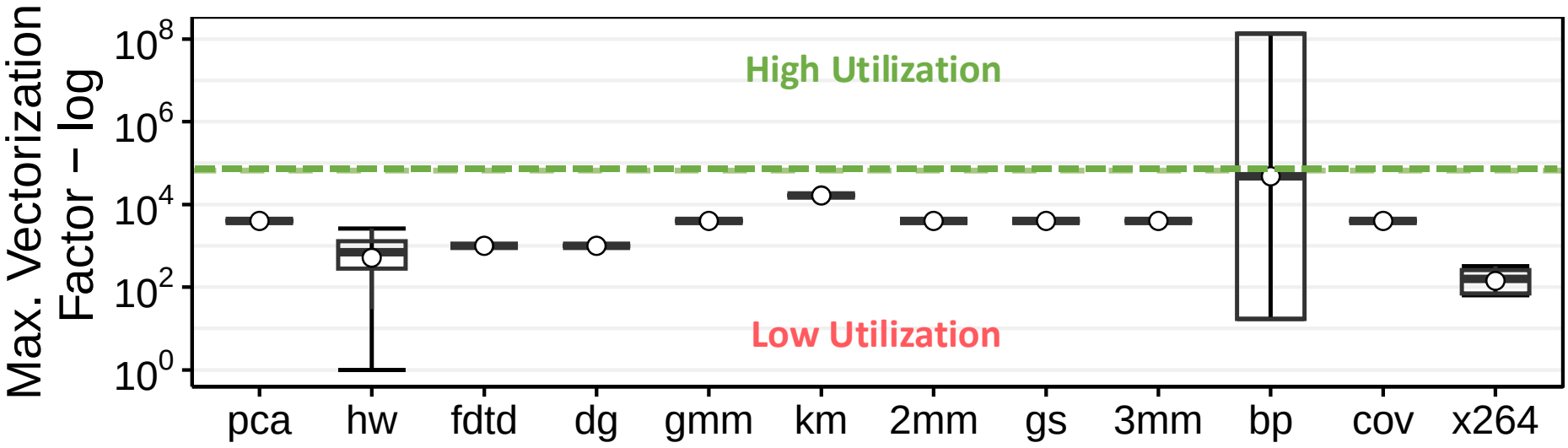
quantify the fraction of SIMD parallelism in real applications



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

Application Analysis:

quantify the fraction of SIMD parallelism in real applications



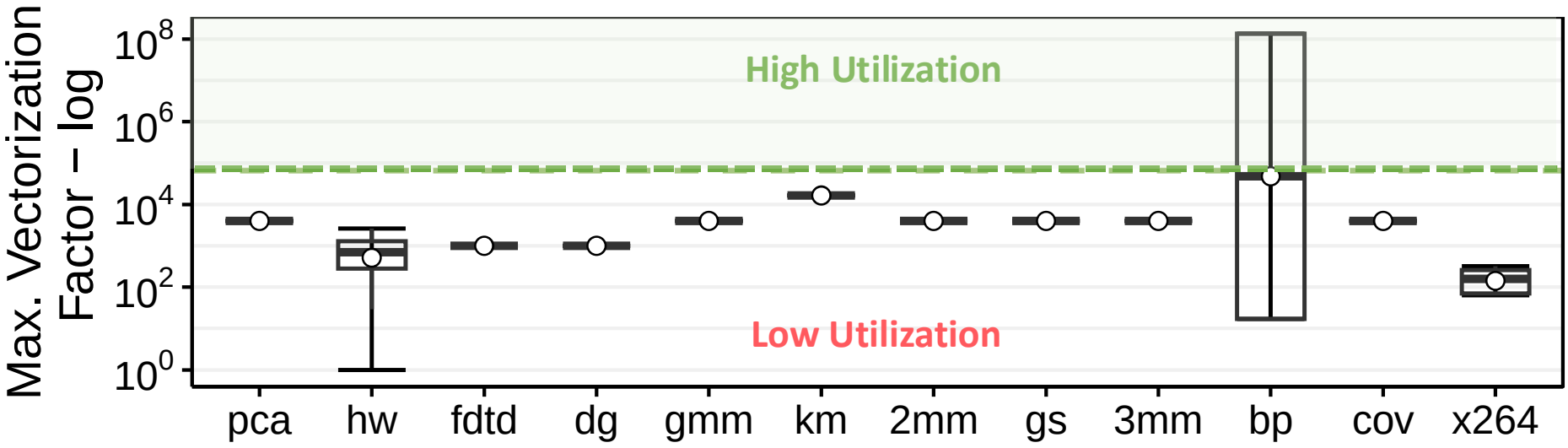
Takeaway

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

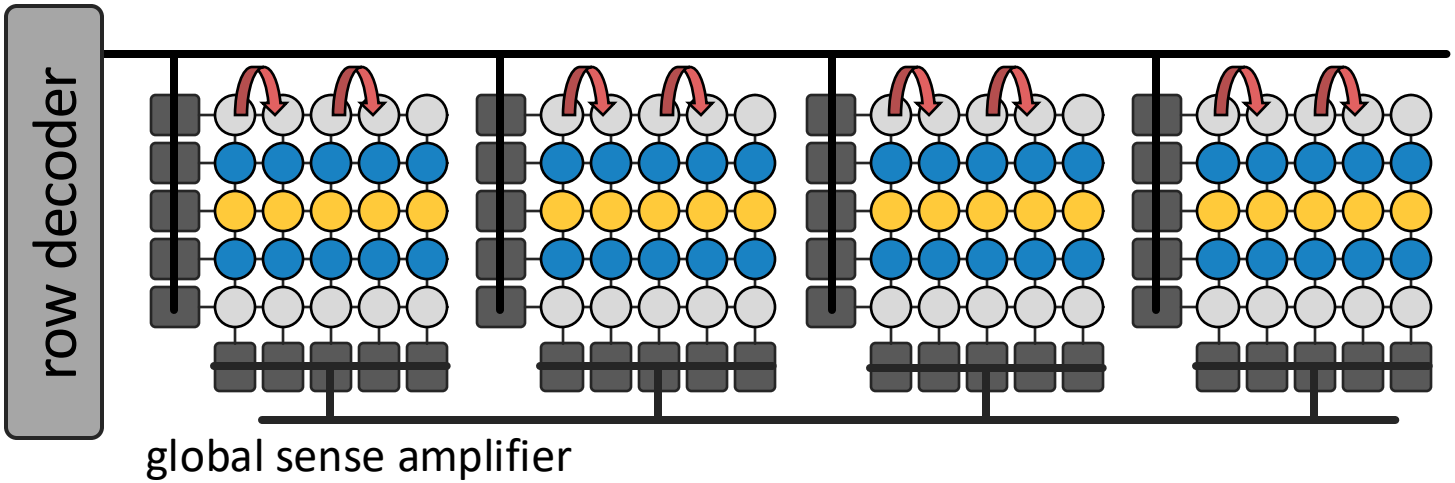
1 SIMD Underutilization
- due to data parallelism variation within and across applications
- leads to throughput and energy waste

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

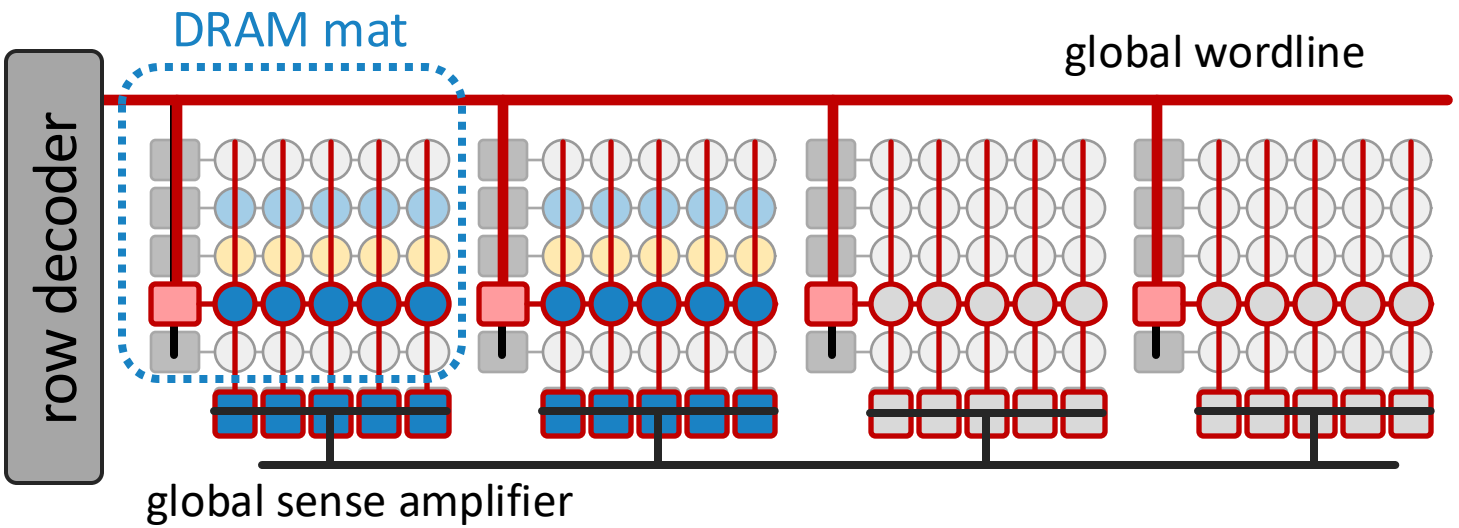
Goal

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

Outline

MIMDRAM:
Key Idea (I)

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



Key Issue:

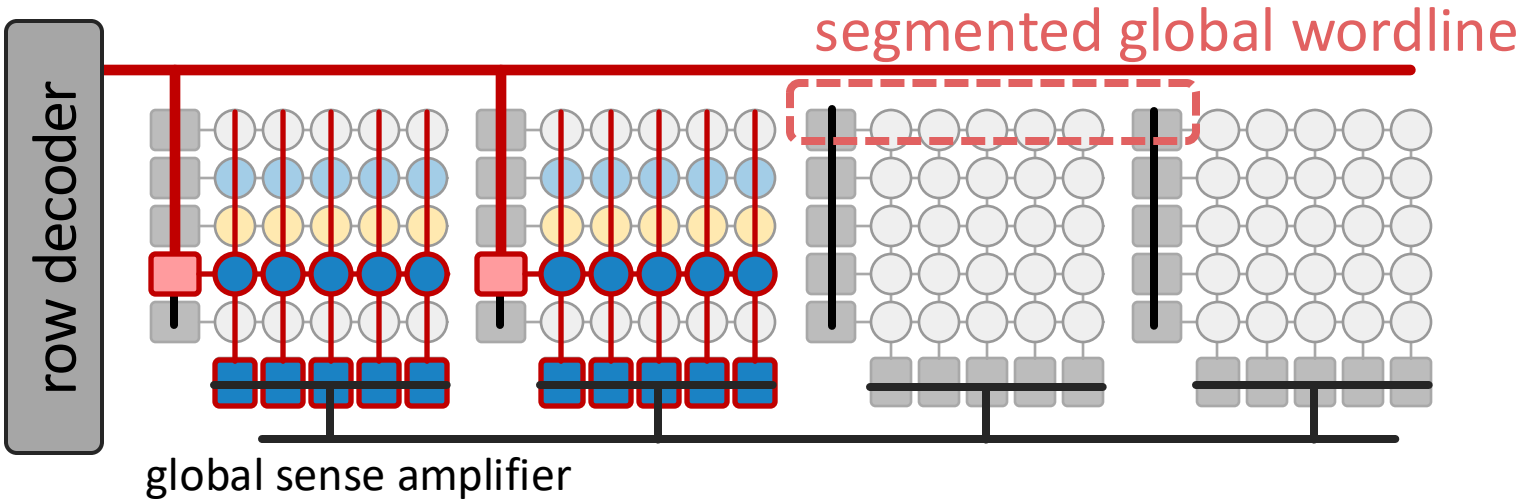
on a DRAM access, the global wordline propagates across all DRAM mats

Fine-Grained DRAM:

segments the global wordline to access individual DRAM mats

MIMDRAM:
Key Idea (II)

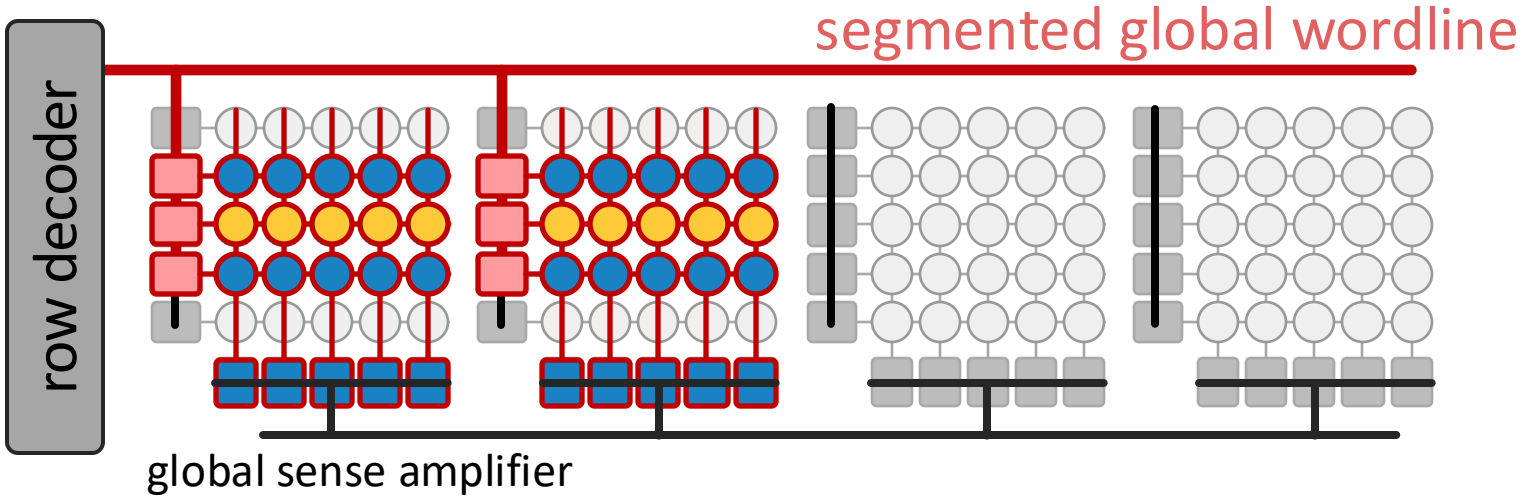
Fine-Grained DRAM:
segments the global wordline to access individual DRAM mats



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)

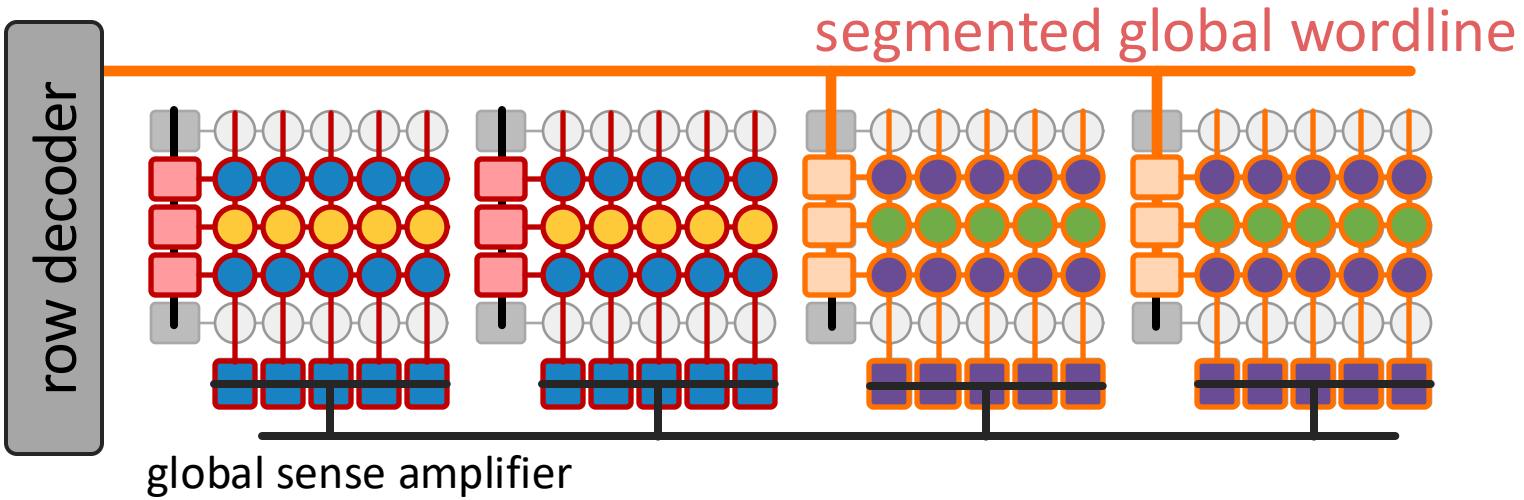


Fine-grained DRAM for processing-using-DRAM:

1 Improves SIMD utilization

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

MIMDRAM:
Key Idea (III)

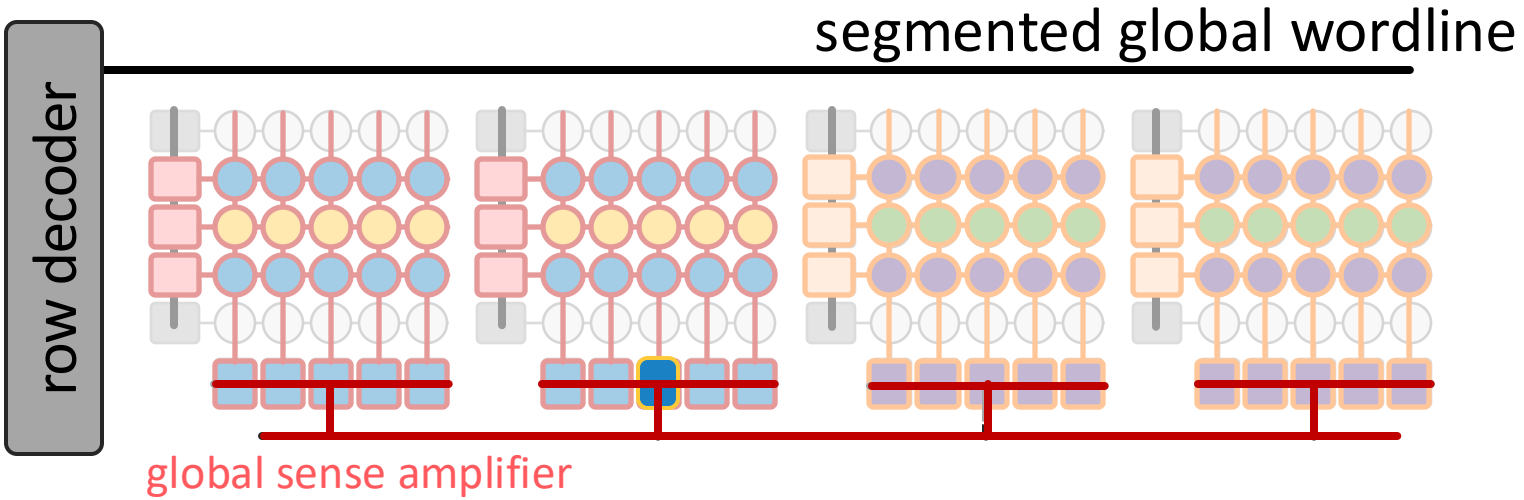


Fine-grained DRAM for processing-using-DRAM:

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

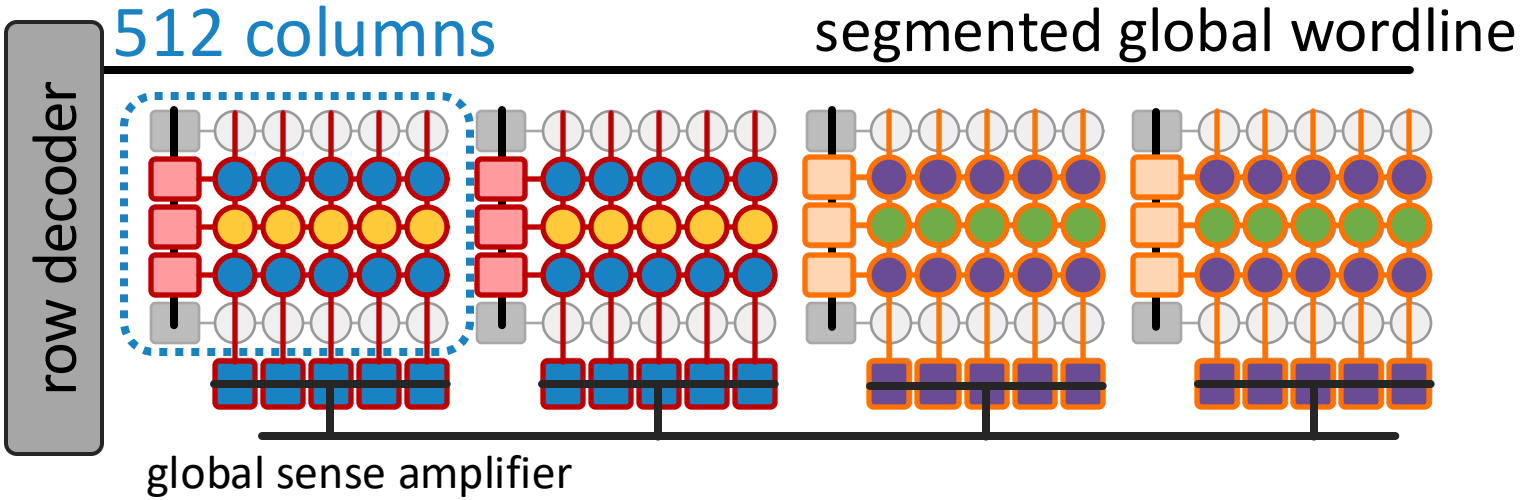
- for a single PUD operation, only access the DRAM mats with target data
- for multiple PUD operations, execute independent operations concurrently

→ 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

Outline

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:

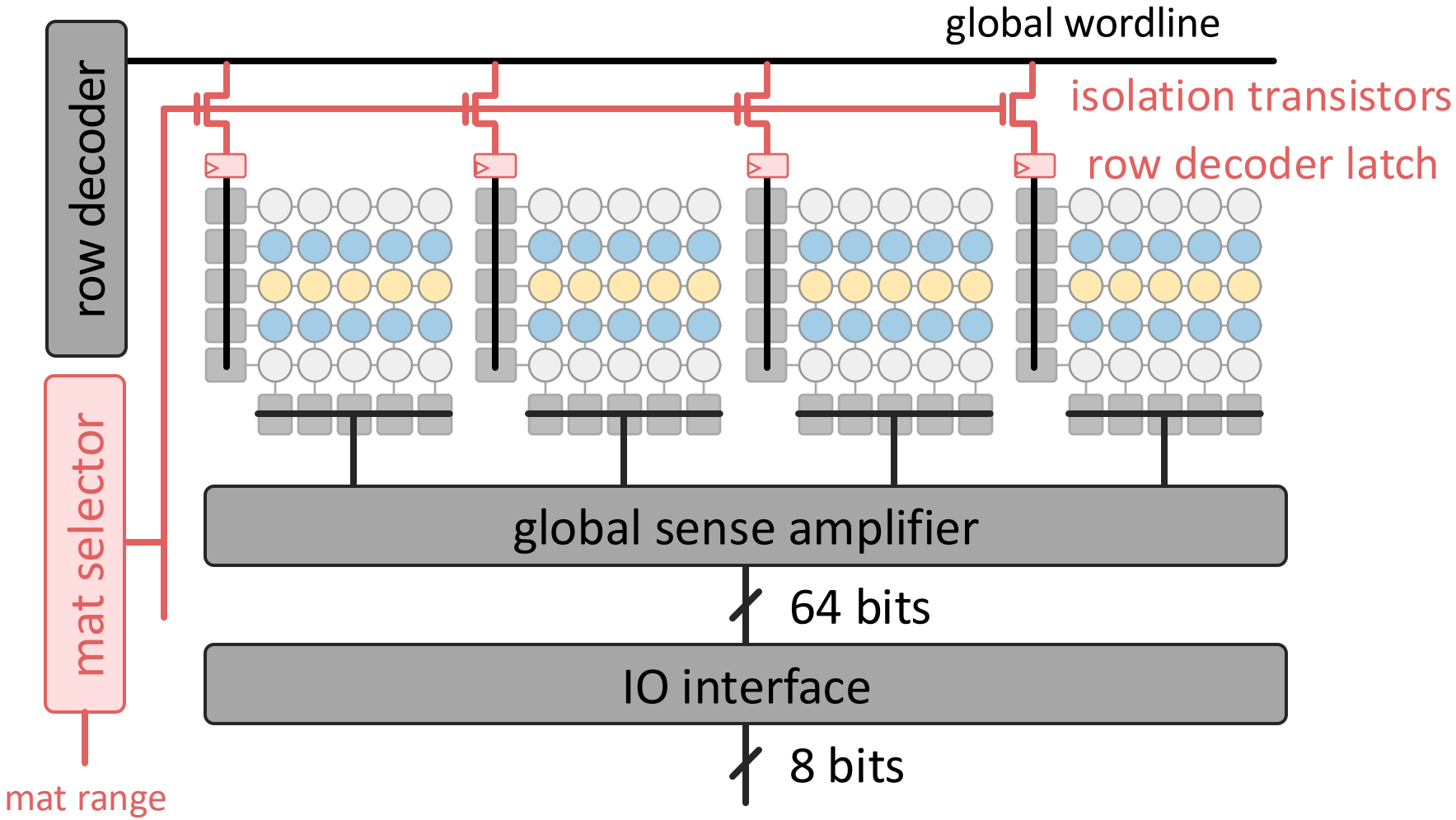
1 Hardware

- **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:
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

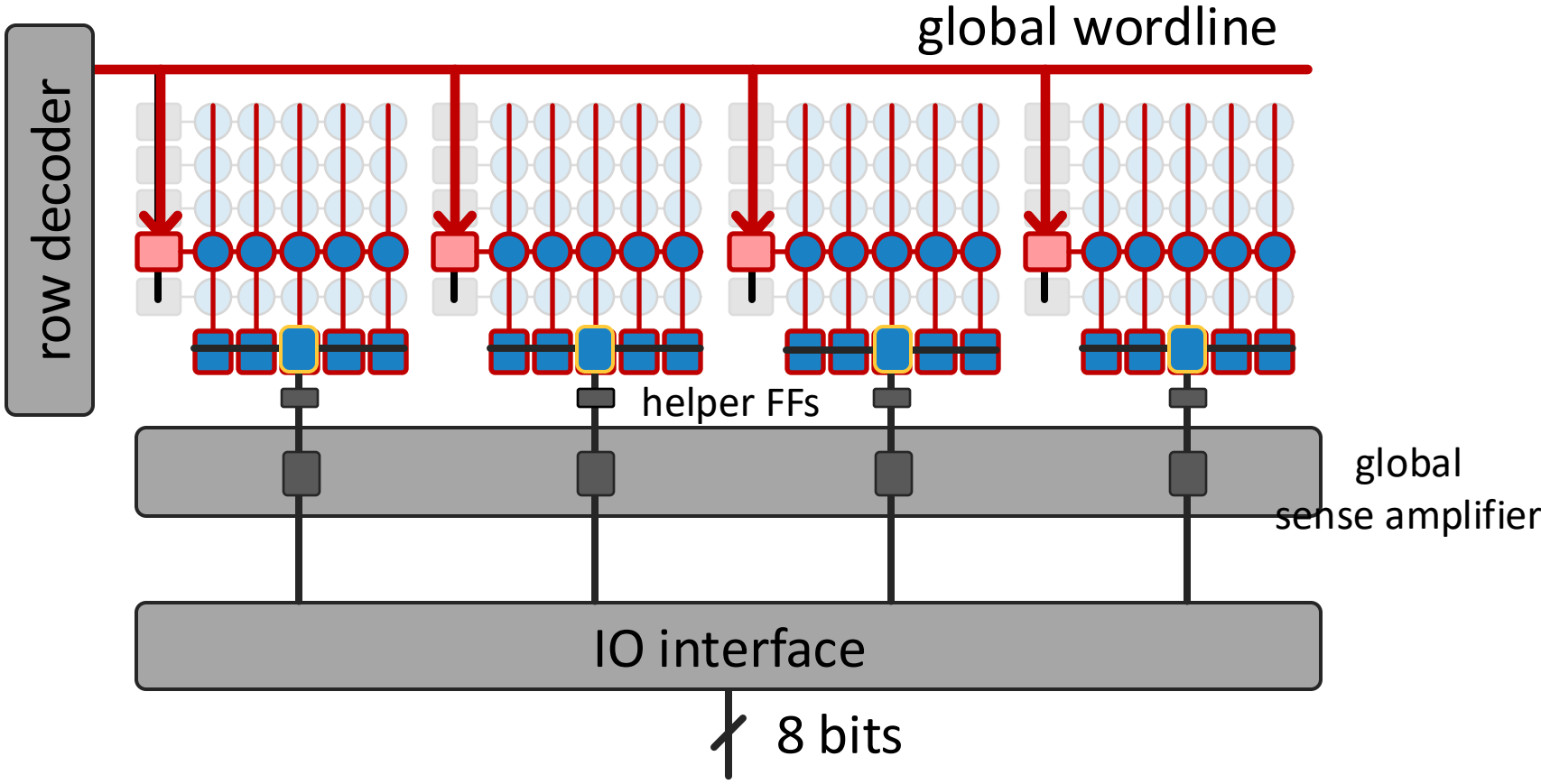
- DRAM array modification to enable fine-grained PUD computation
- **inter- and intra-mat interconnects to enable PUD vector reduction**
- control unit design to orchestrate PUD execution

2 Software

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

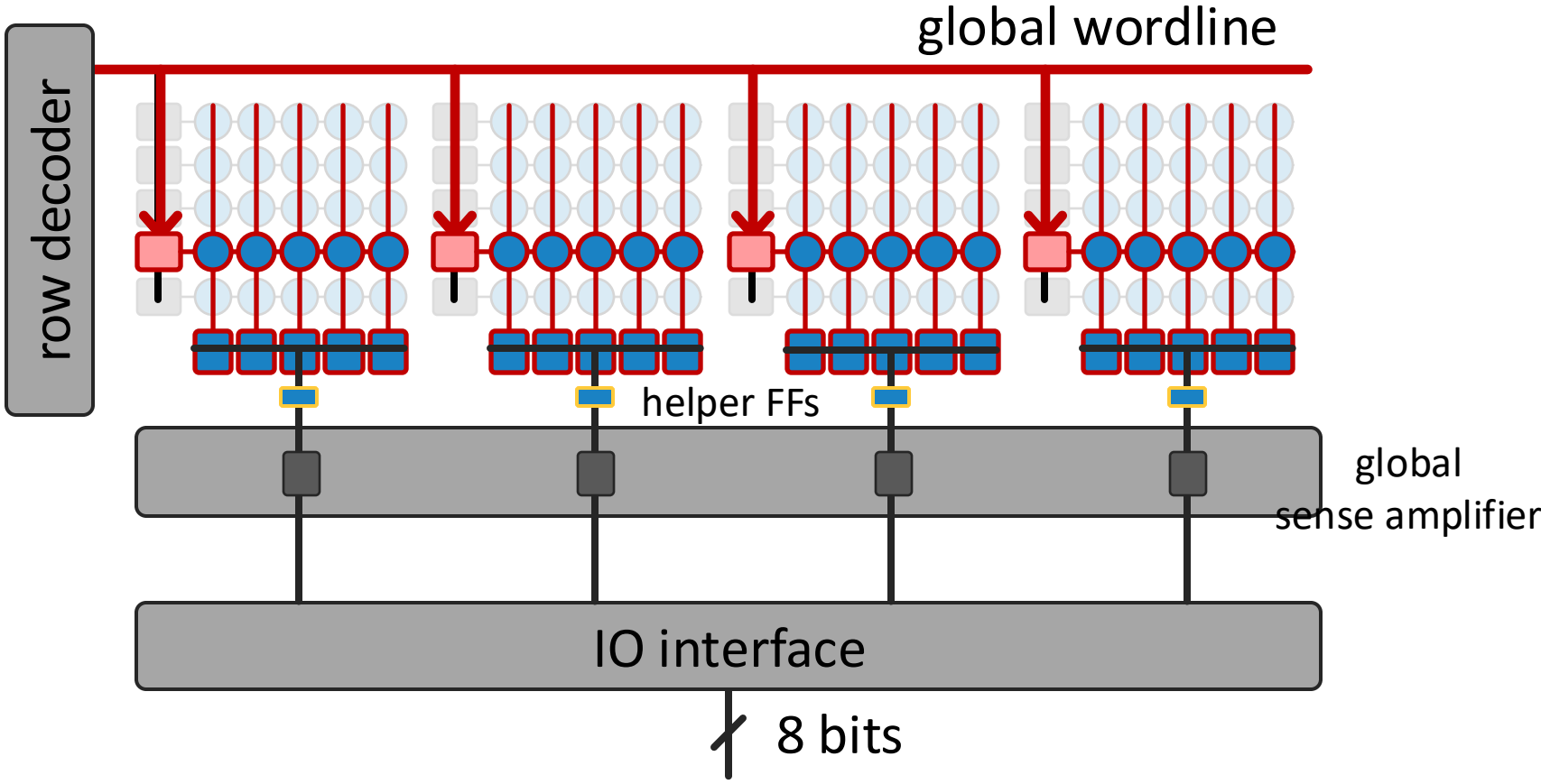
MIMDRAM:
Background on DRAM Column Access

During a column access,
data is locally and globally amplified to improve signal integrity



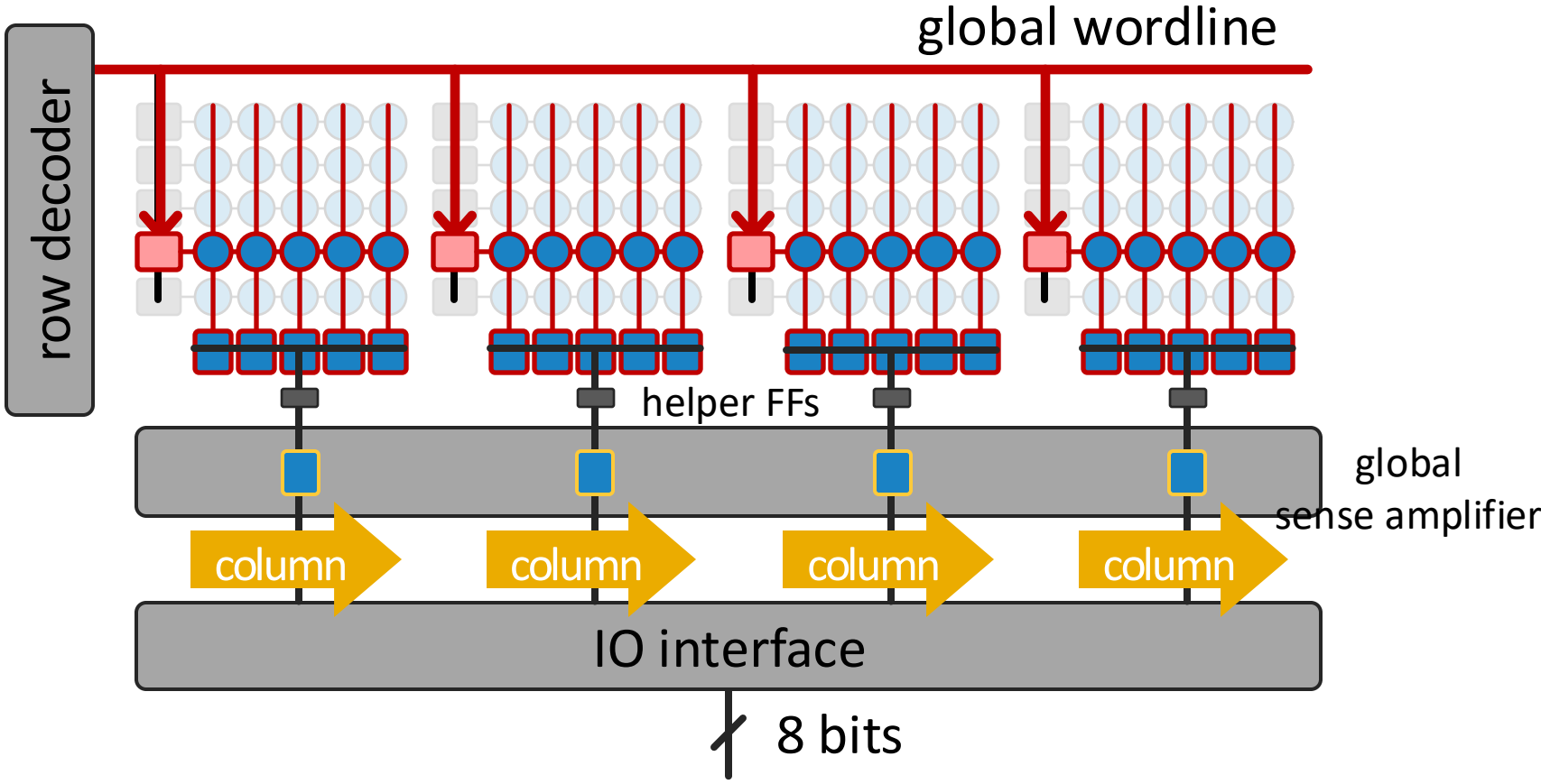
MIMDRAM:
Background on DRAM Column Access

During a column access,
data is locally and globally amplified to improve signal integrity



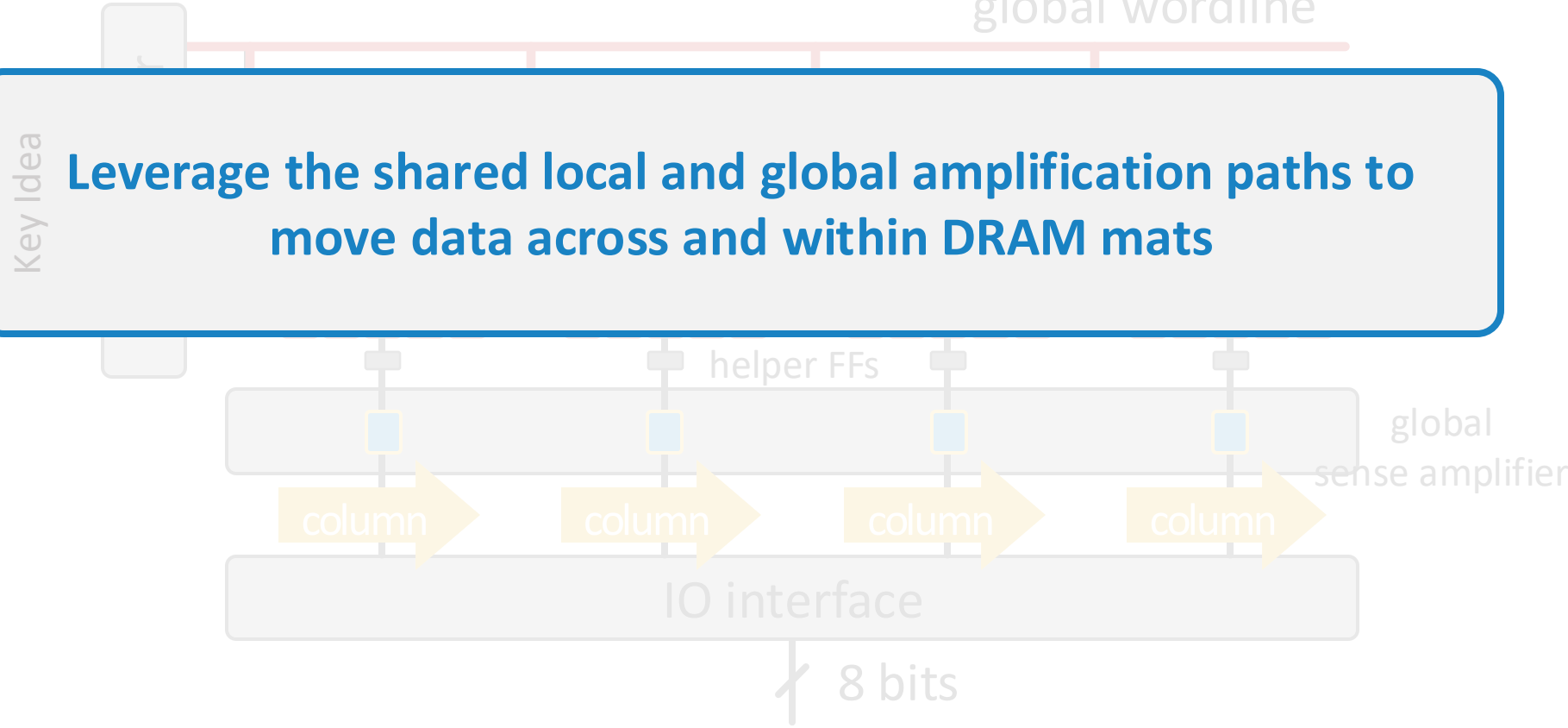
MIMDRAM:
Background on DRAM Column Access

During a column access,
data is locally and globally amplified to improve signal integrity



MIMDRAM:
Background on DRAM Column Access

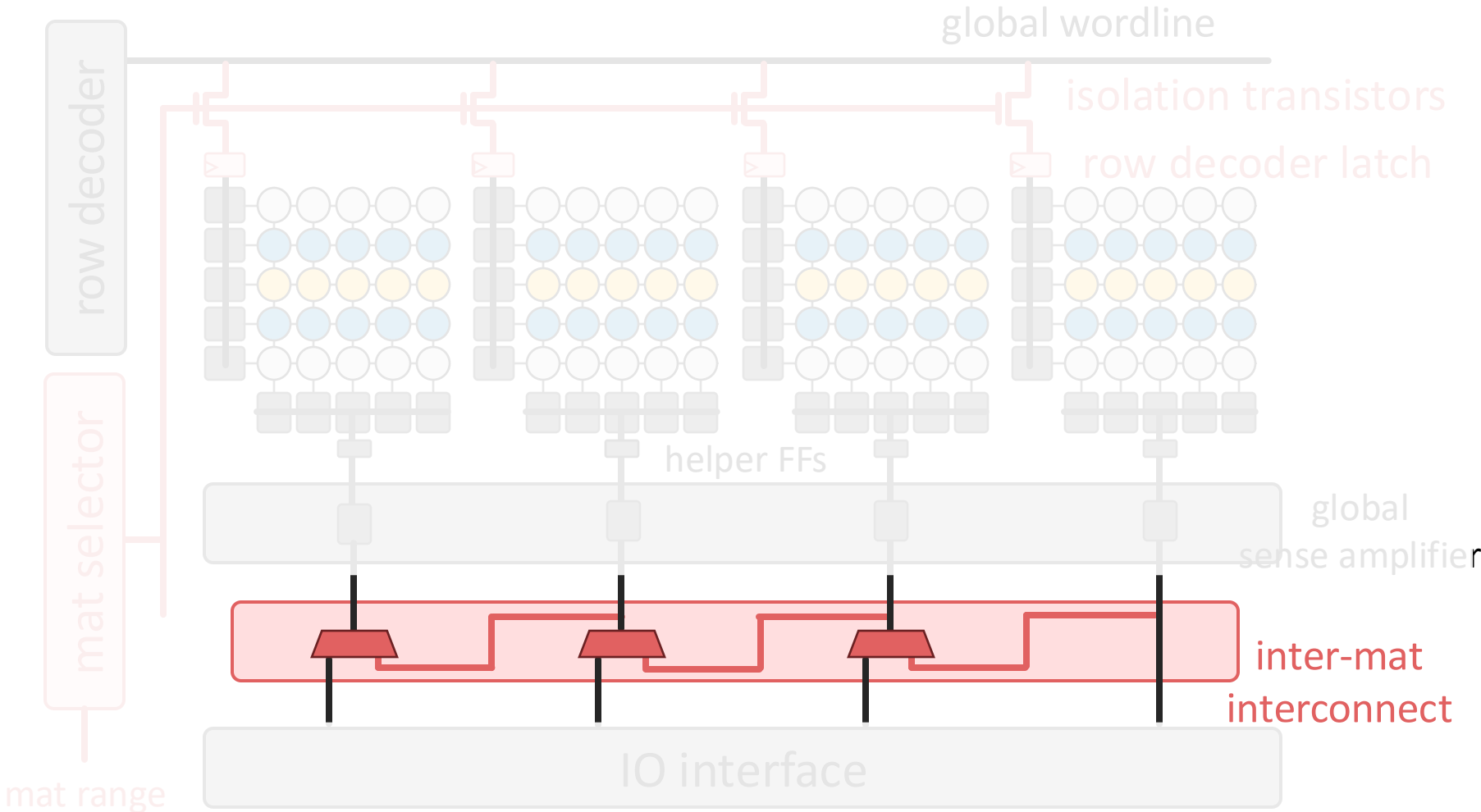
During a column access,
data is locally and globally amplified to improve signal integrity



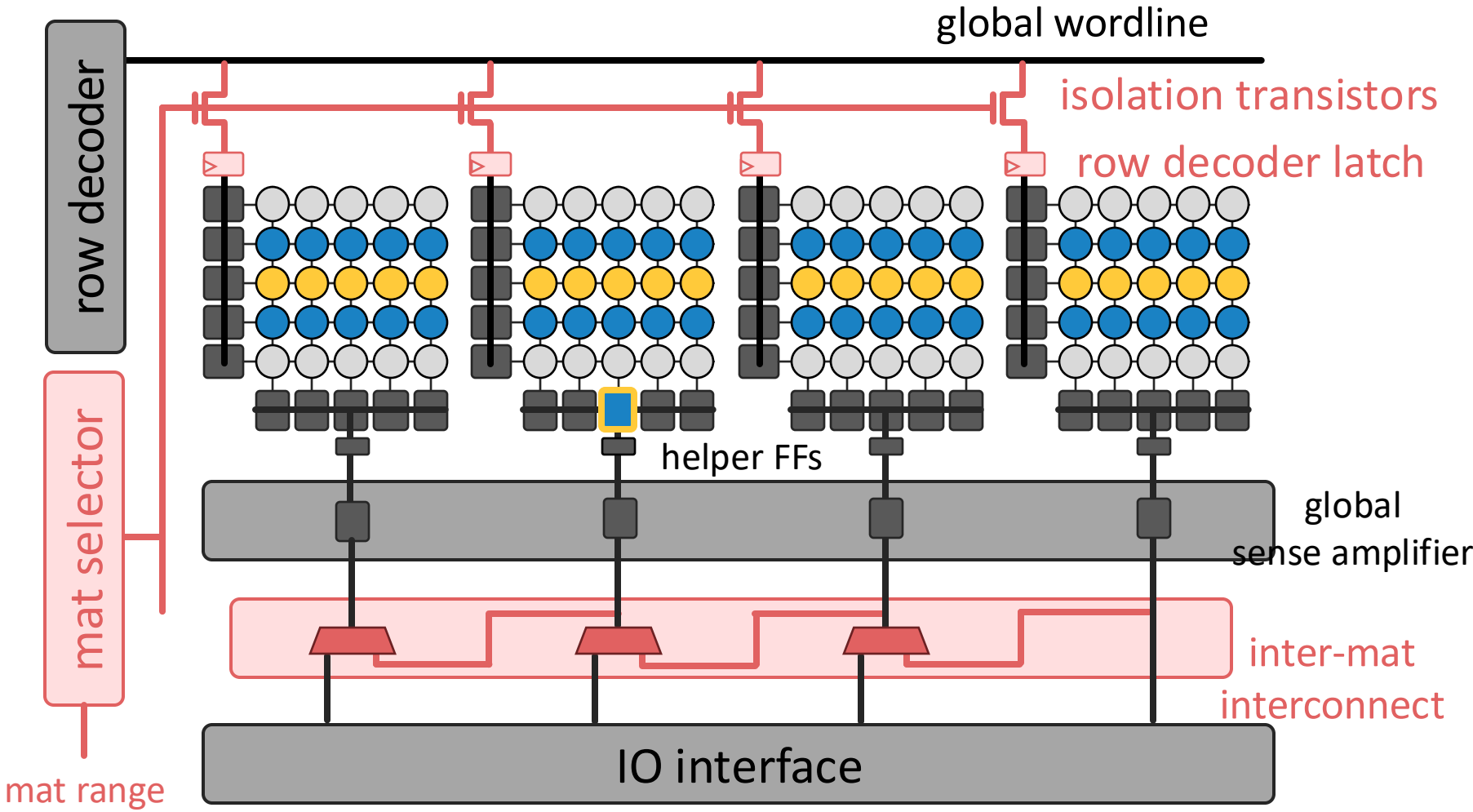
Key Idea

Leverage the shared local and global amplification paths to move data across and within DRAM mats

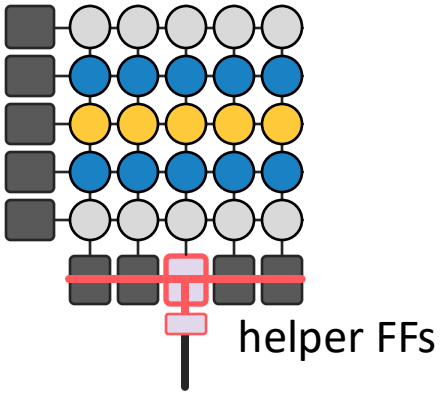
MIMDRAM:
Moving Data Across Mats



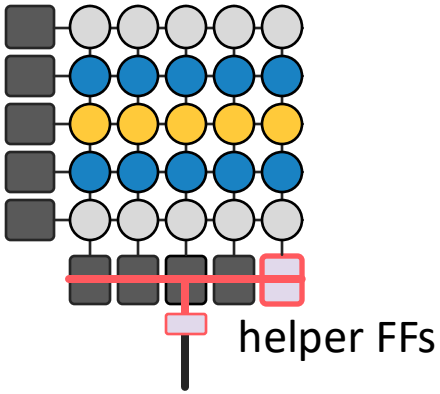
MIMDRAM:
Moving Data Across Mats



MIMDRAM:
Moving Data Within a Mat



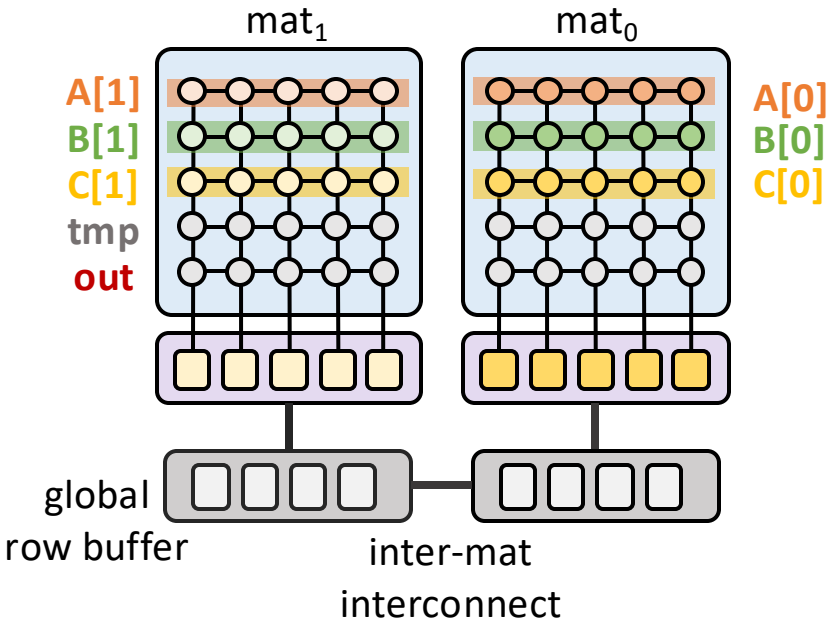
MIMDRAM:
Moving Data Within a Mat



Helper flip-flops latches and drives source column into the destination column

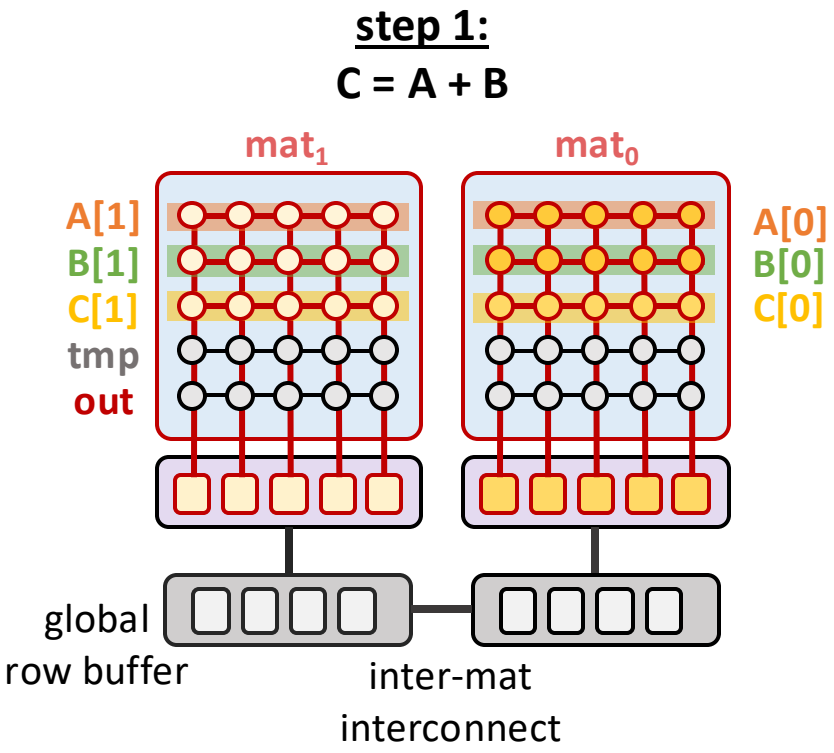
MIMDRAM:
In-DRAM Vector Reduction (I)

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



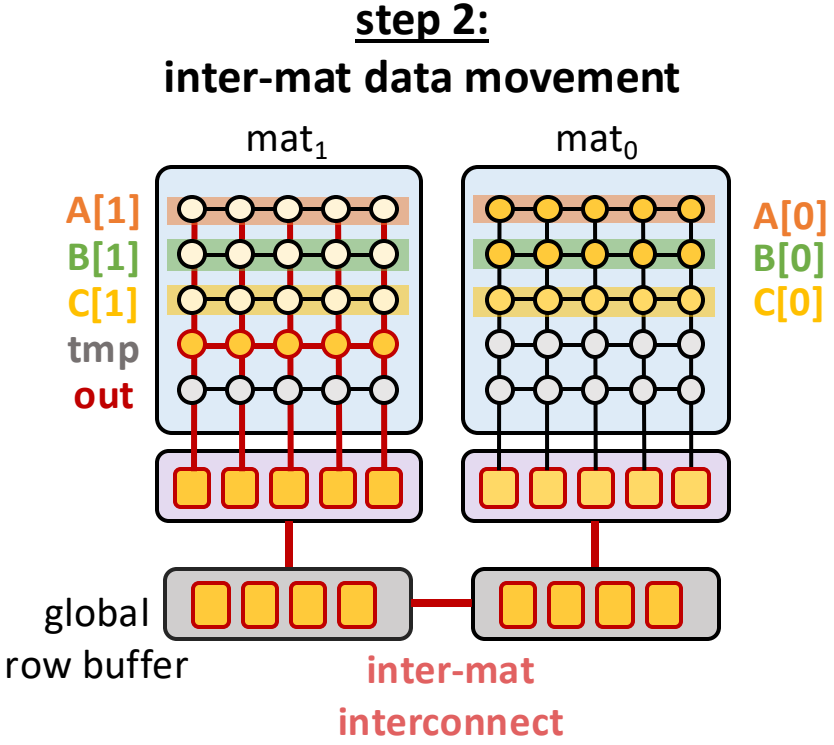
MIMDRAM:
In-DRAM Vector Reduction (II)

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



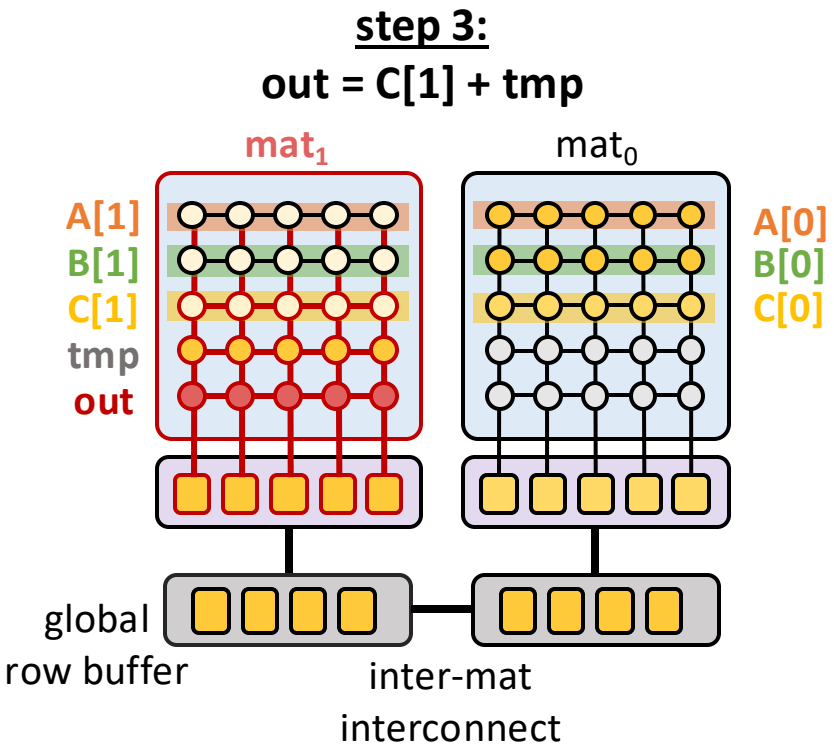
MIMDRAM:
In-DRAM Vector Reduction (III)

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



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

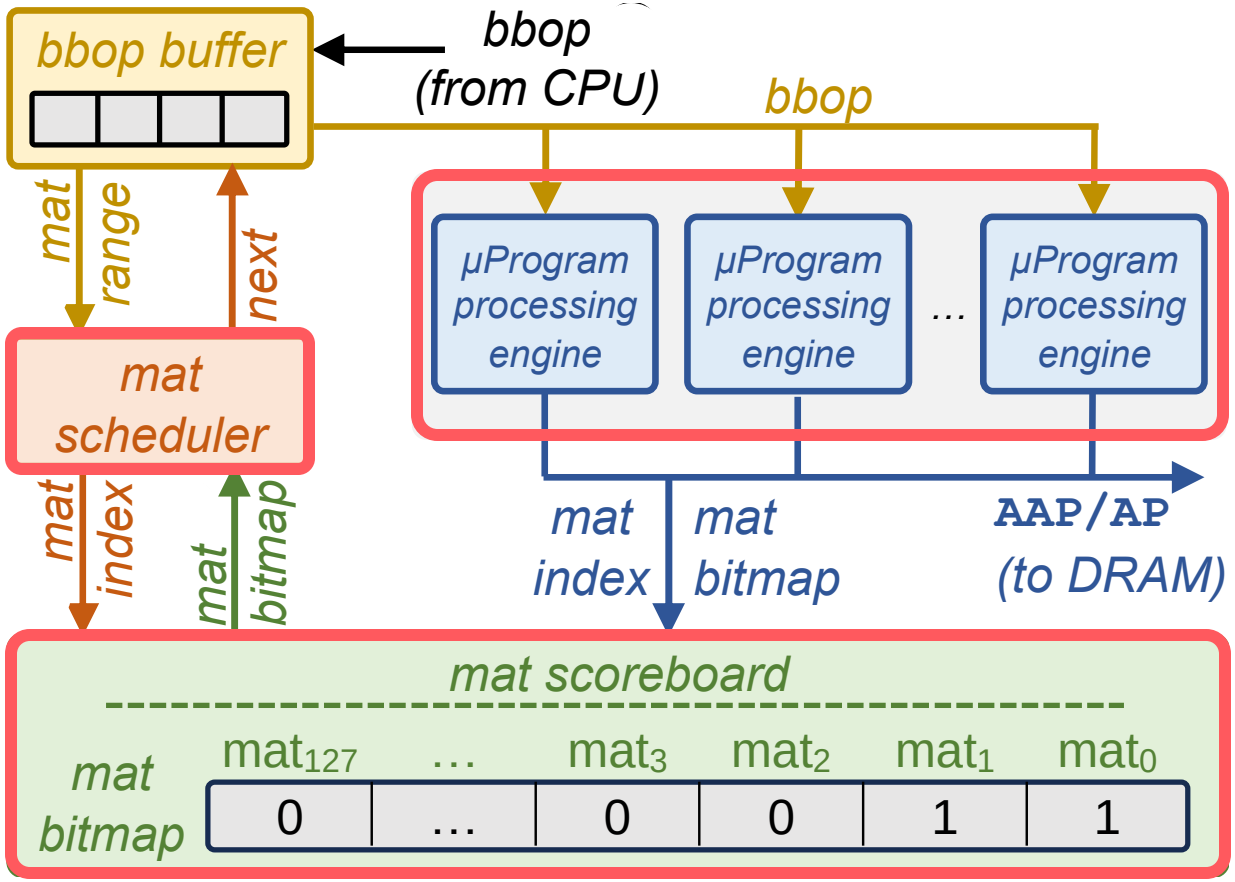
- DRAM array modification to enable fine-grained PUD computation
- inter- and intra-mat interconnects to enable PUD vector reduction
- **control unit design to orchestrate PUD execution**

2 Software

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

MIMDRAM:
Control Unit

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



Outline

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

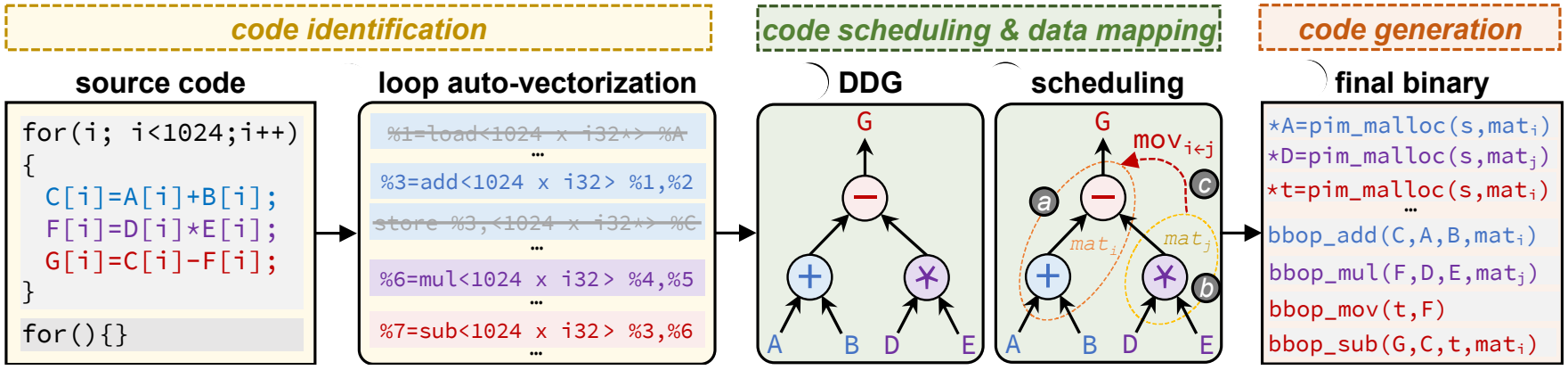
- **new compiler support to transparently generate PUD instructions**
- system support to map and execute PUD instructions

MIMDRAM:
Compiler Support (I)

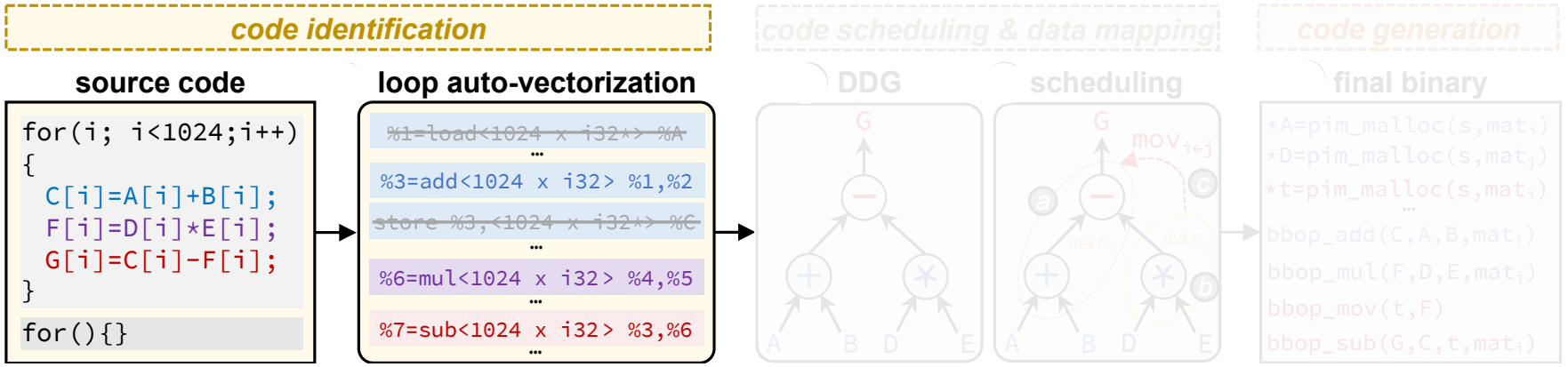
Goal

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

Three new LLVM-based passes targeting PUD execution



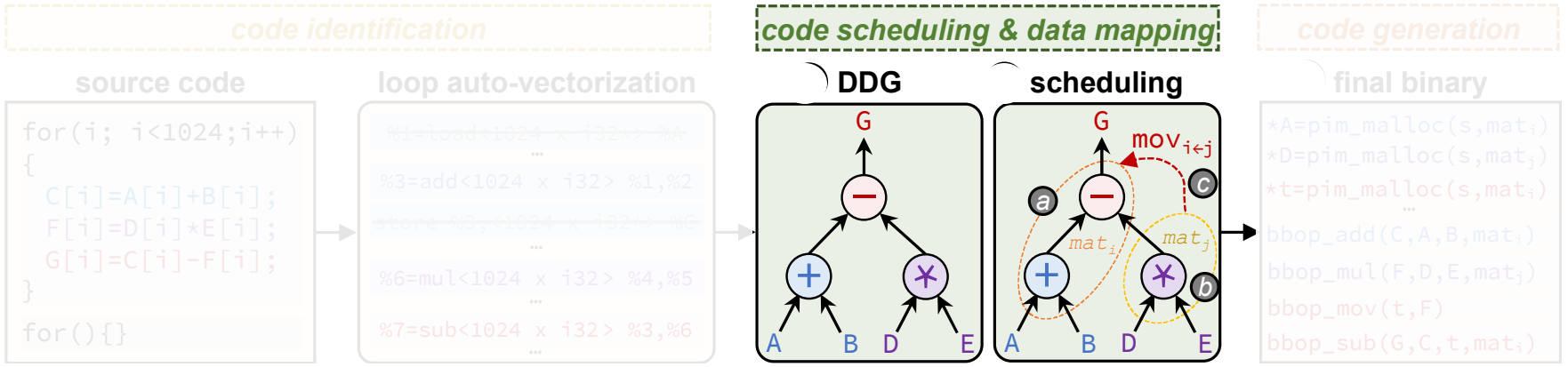
MIMDRAM:
Compiler Support (II)



Goal

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

MIMDRAM:
Compiler Support (II)



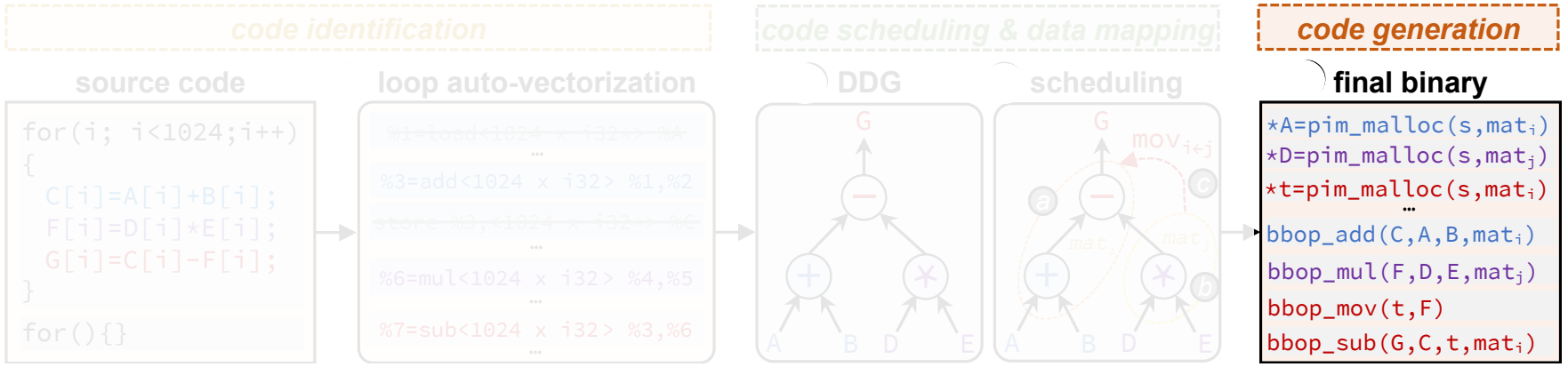
Goal

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:
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

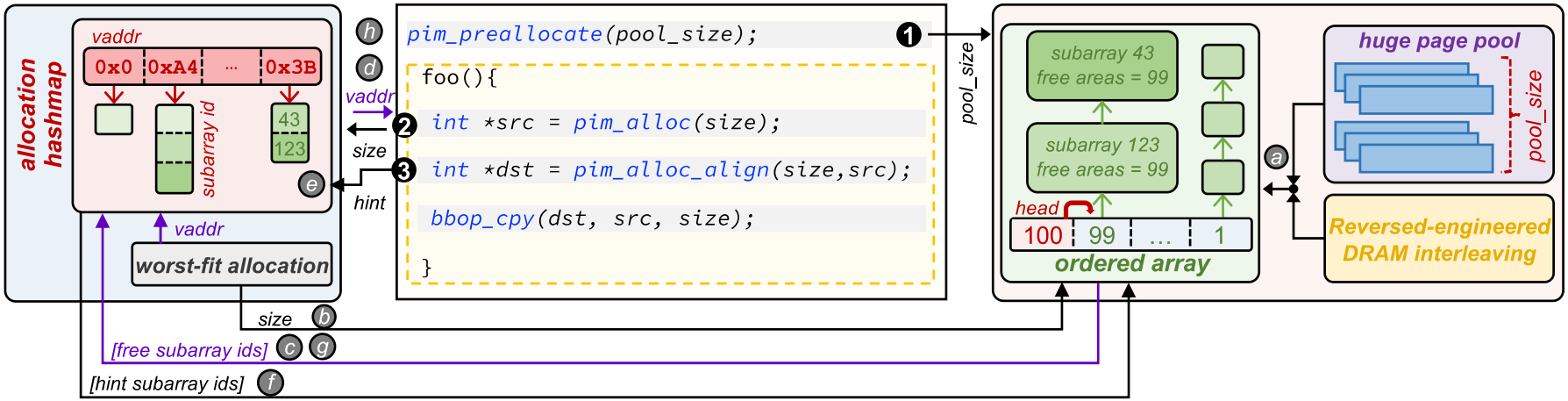
- Instruction set architecture
- Execution & data transposition
- Data coherence
- Address translation
- Data allocation & alignment
- Mat label translation

MIMDRAM:
System Support

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

MIMDRAM:
Data Allocation & Alignment

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

Outline

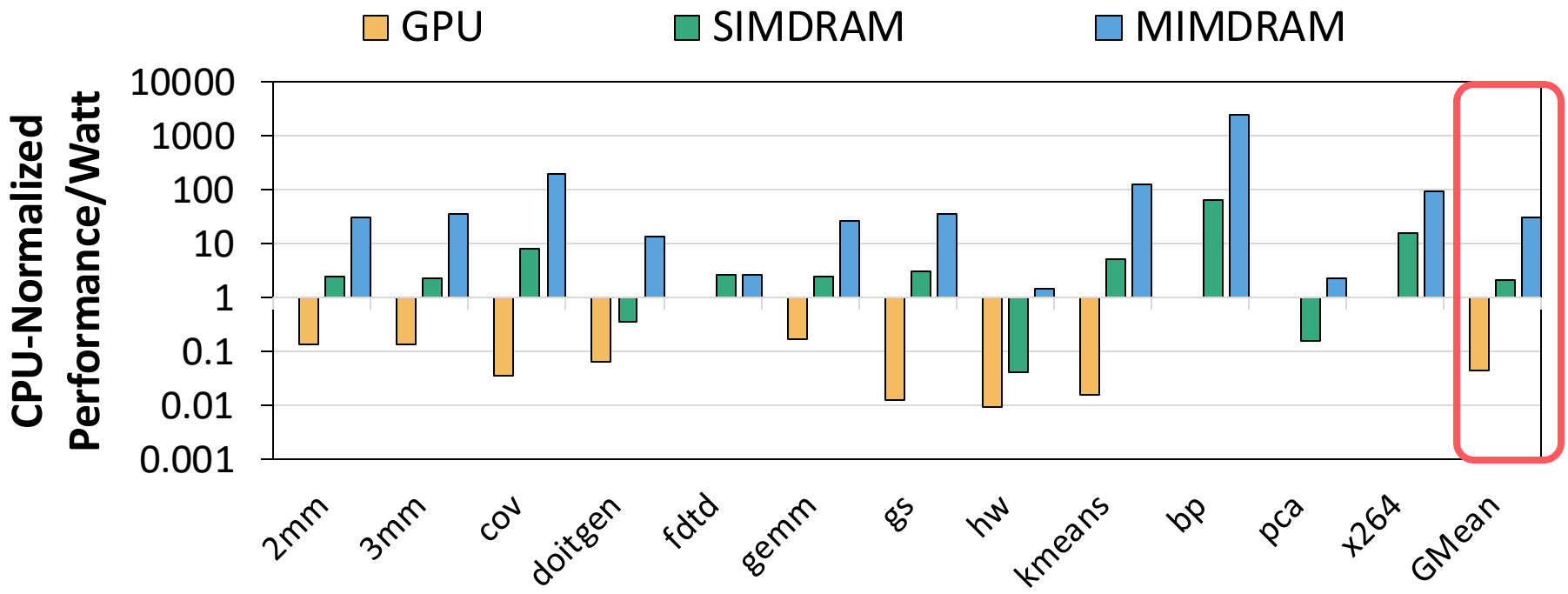
Evaluation:
Methodology Overview

- **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

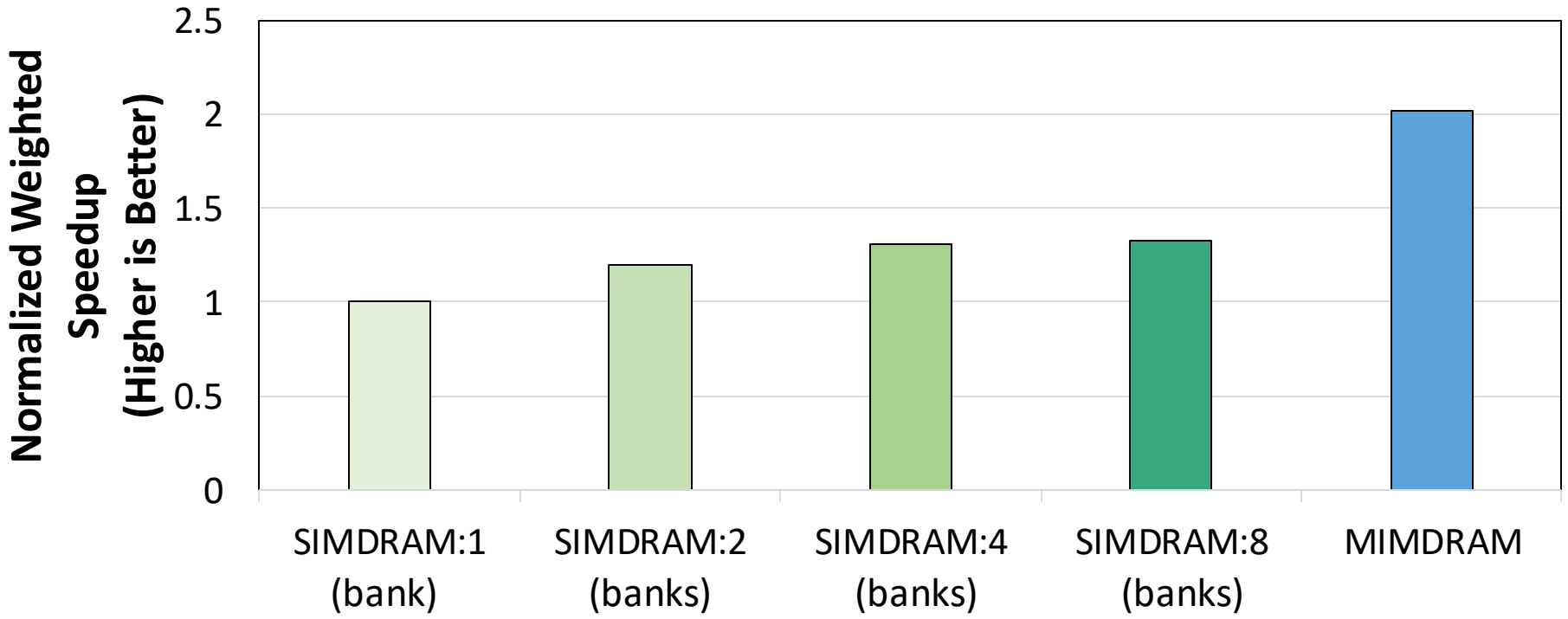
Evaluation:
Single Application Analysis – Energy Efficiency



Takeaway

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

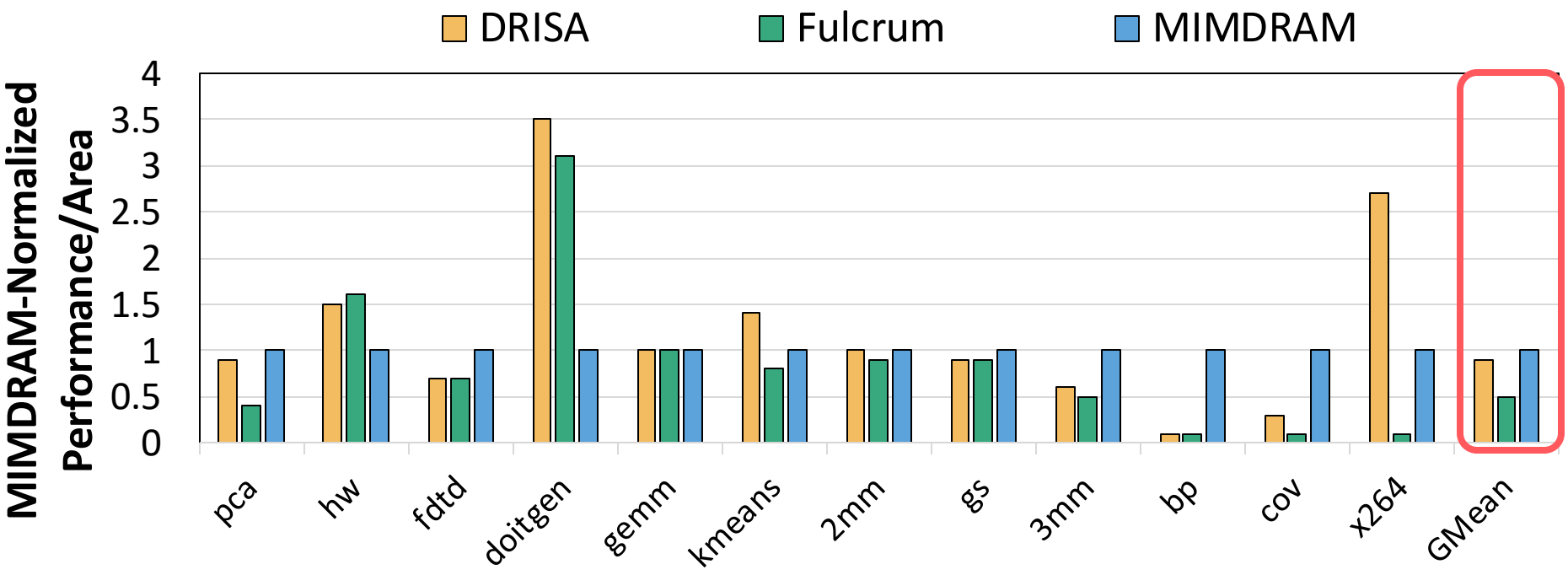
Evaluation:
Multi-Programmed Workload Analysis



Takeaway

MIMDRAM significantly improves system throughput (1.68x) compared to SIMDRAM

Evaluation:
Comparison to Other PIM Architectures



Takeaway

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

Evaluation:
More in the Paper

- **MIMDRAM with subarray and bank-level parallelism**
  - MIMDRAM provides significant performance gains compared to the baseline CPU (**13.2x**) and GPU (**2x**)
- **Comparison to DRISA and Fulcrum for multi-programmed workloads**
  - MIMDRAM achieves system throughput on par with DRISA and Fulcrum
- **MIMDRAM's SIMD utilization versus SIMDRAM**
  - MIMDRAM provides **15.6x** the utilization of SIMDRAM
- **Area analysis**
  - MIMDRAM adds small area cost to a DRAM chip (**1.11%**) and CPU die (**0.6%**)

Evaluation:
More in the Paper

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

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

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

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

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

Outline

Conclusion

We introduced MIMDRAM,
a hardware/software co-designed processing-using-DRAM system

- **Key idea**: leverage fine-grained DRAM for processing-using-DRAM operation
- **HW**: - simple changes to DRAM, enabling concurrent instruction execution
  - low-cost interconnects at the DRAM peripherals for data reduction
- **SW**: - compiler and OS support to generate and map instructions

Our evaluation demonstrates that MIMDRAM

- **significantly** improves **performance**, **energy efficiency**, and **throughput** compared to processor-centric (CPU and GPU) and memory-centric (SIMDRAM, DRISA, and Fulcrum) architectures
- incurs **small area cost** to a DRAM chip and CPU die

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

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

Full Draft

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

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

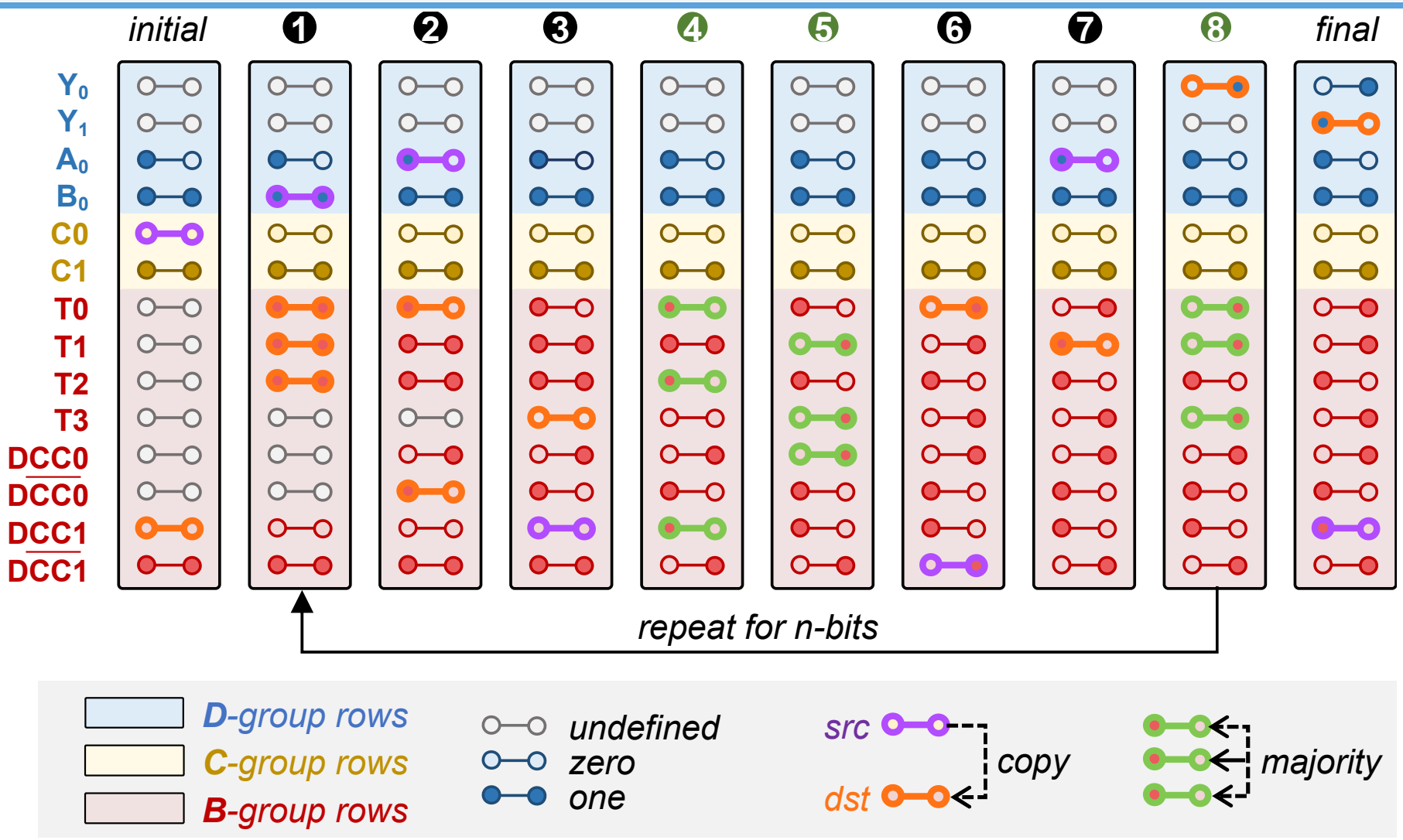
Backup Slides

Geraldo F. Oliveira

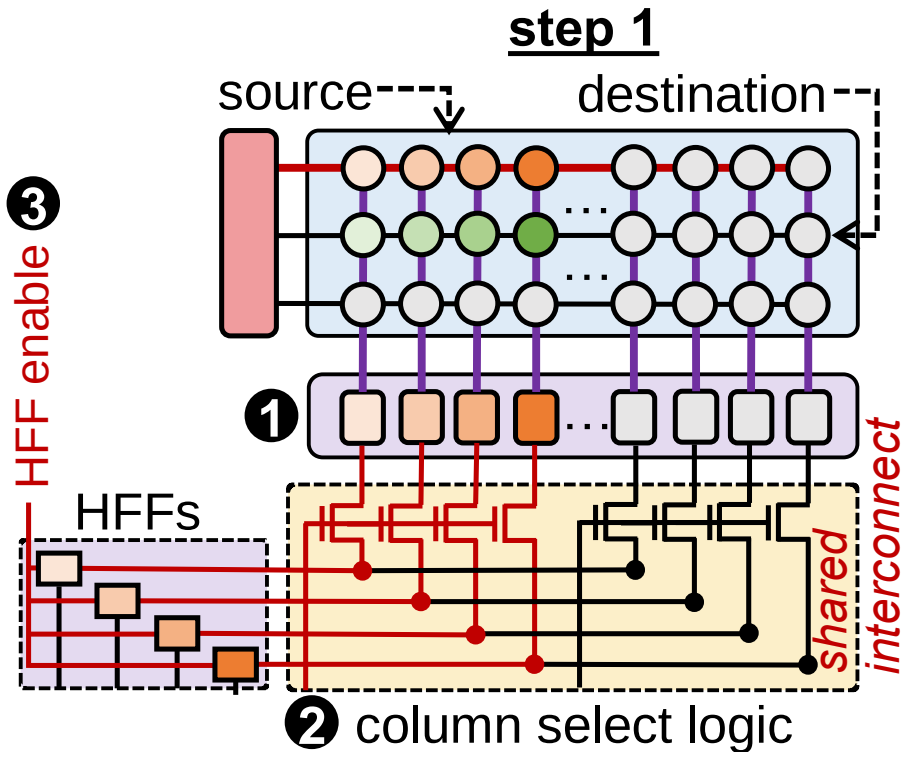
Ataberk Olgun
A. Giray Yağlıkçı
F. Nisa Bostancı
Saugata Ghose
Juan Gómez-Luna
Onur Mutlu

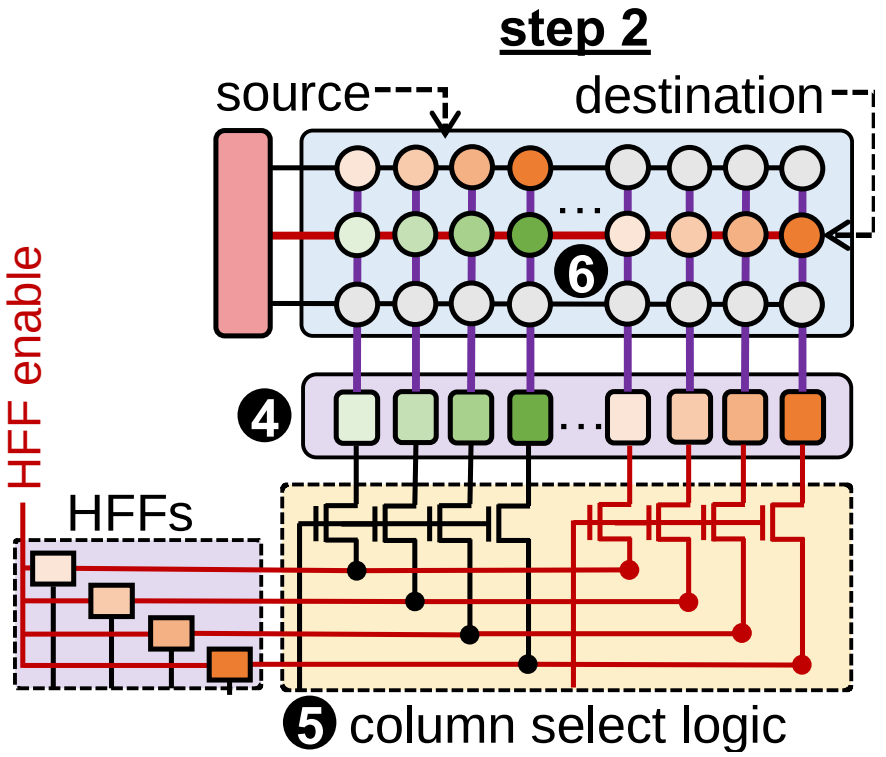
ETH zürich
UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN
SAFARI

Bit-Serial PUD Addition

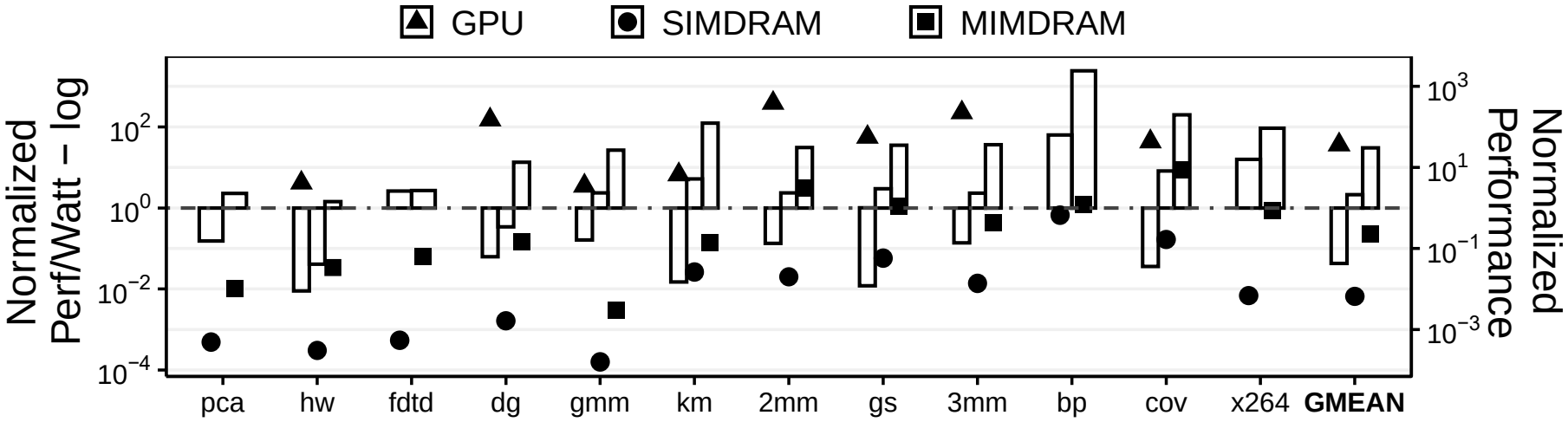


Intra-Mat Interconnect

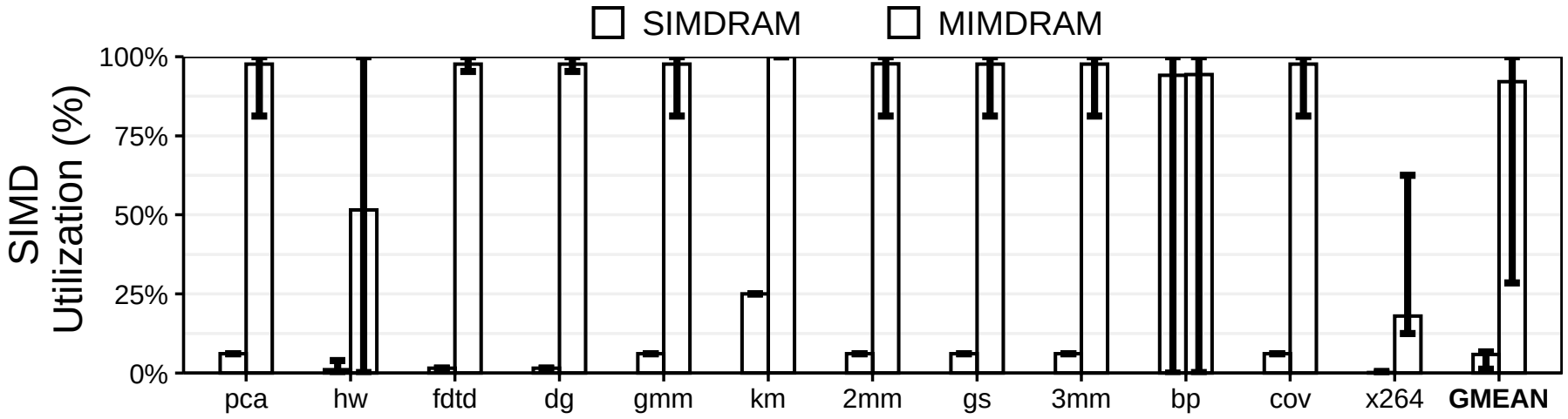




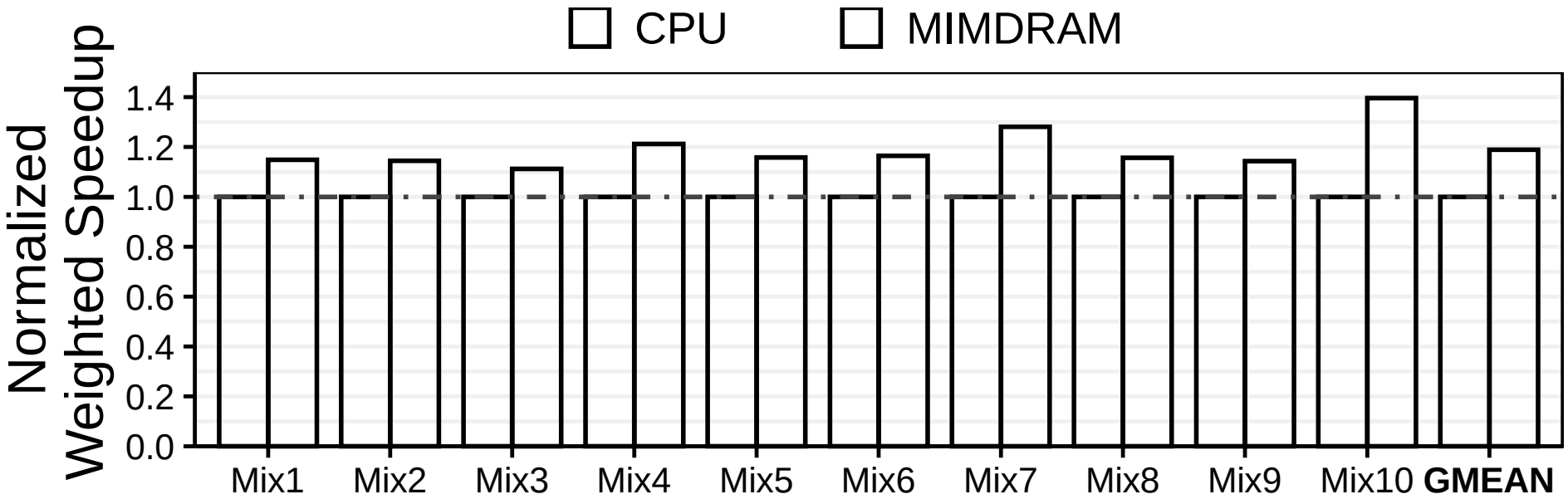
Performance for Single Applications



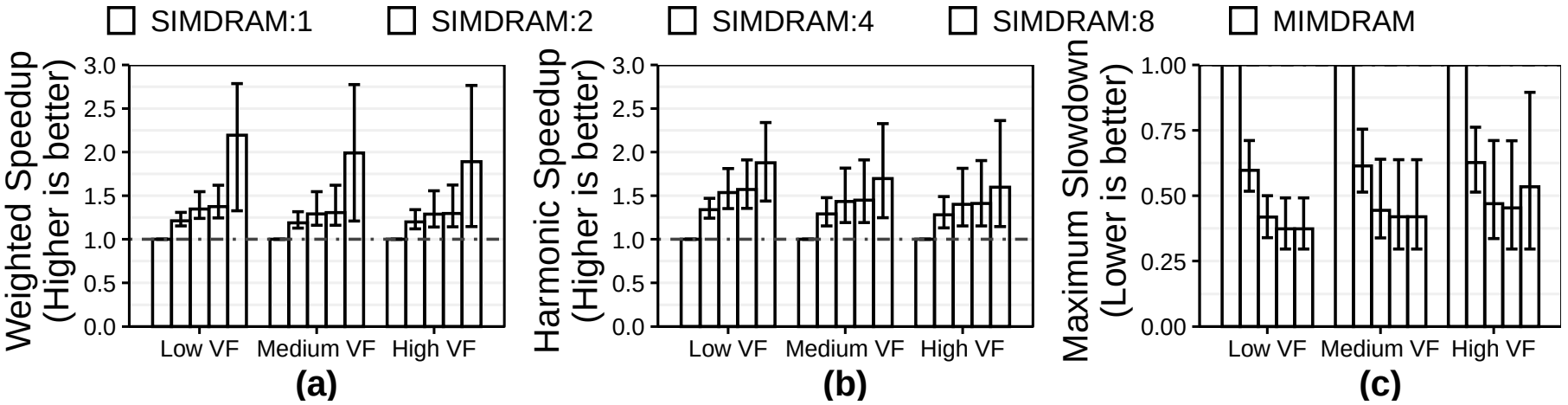
SIMD Utilization: MIMDRAM vs. SIMDRAM



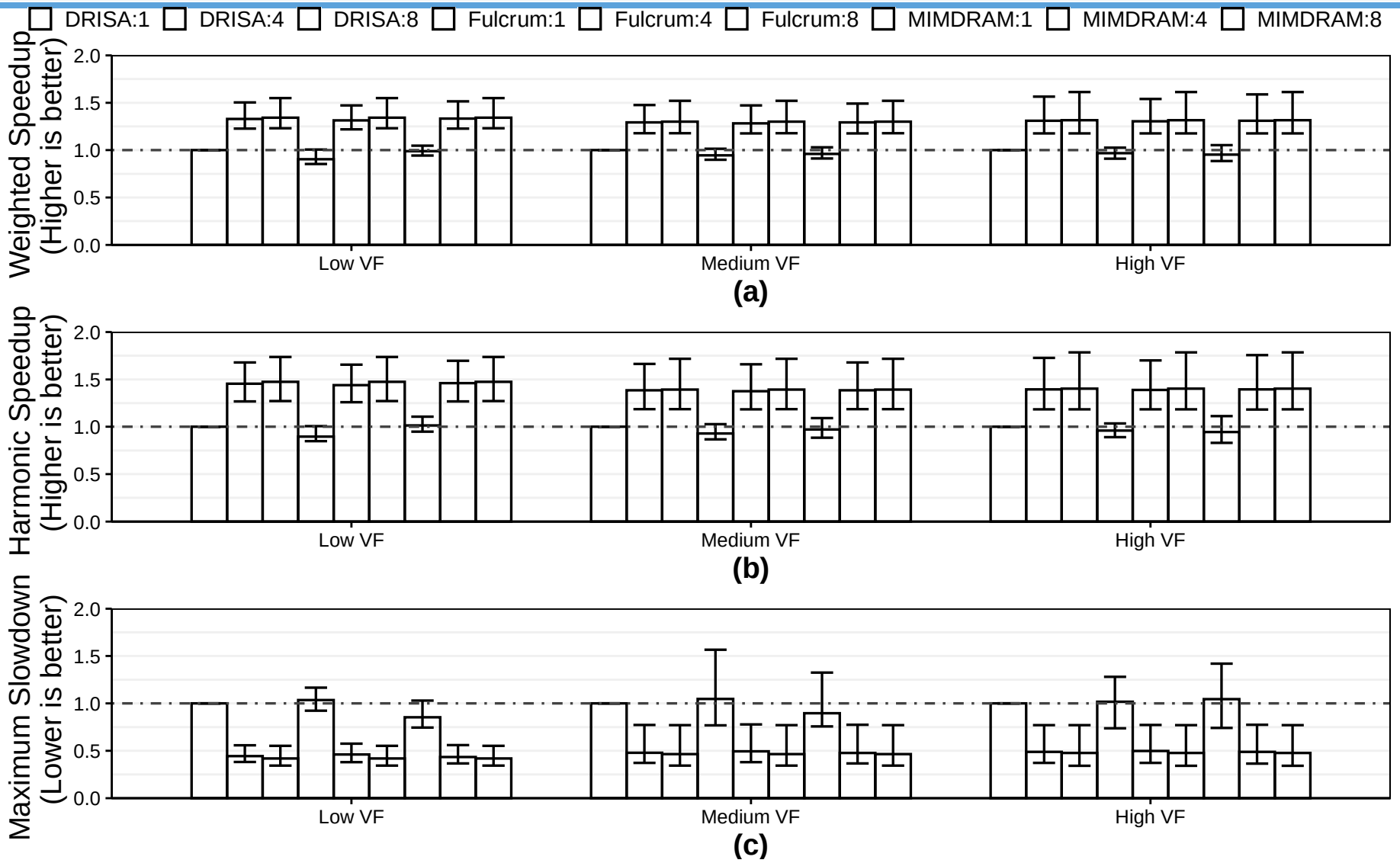
Weighted Speedup vs. CPU



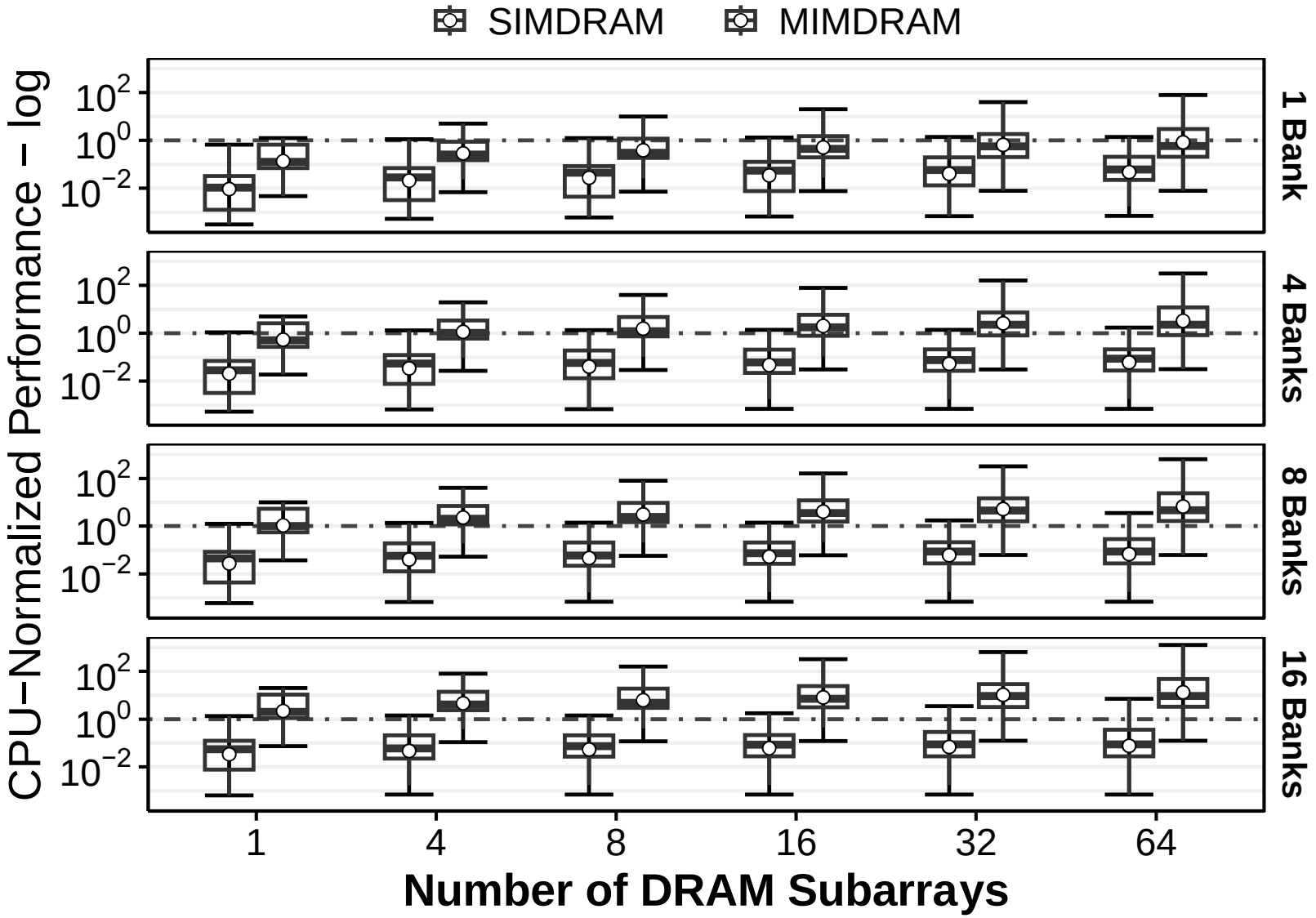
Multi-Applications: Full Data



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