

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

Giray Yağlıkçı

Melina Soysal

Mohammad Sadr

Joaquin O. Bueno

Saugata Ghose

Juan Gómez-Luna

Onur Mutlu

Executive Summary

Problem: Processing-Using-DRAM (PUD) suffers from three limitations caused by naive employing a bit-serial execution model

- Rigid and static data representation leading to unnecessary computation
- Throughput-oriented execution leading to limited latency tolerance
- Scalability challenges for high-precision operations

Goal: Design a PUD system that overcomes the three limitations caused by naively employing a bit-serial execution model

Key Mechanism: *Proteus*, a data-aware runtime PUD framework that

1. dynamically adjust the bit-precision
 2. chooses and uses the most appropriate data representation and arithmetic algorithm implementation
- for a given PUD operation

Key Results: *Proteus* achieves

- **17x (90.3x), 7.3x (21x), and 10.2x (8.1x)** the performance per mm² (energy reduction) of a high-end CPU, GPU, and state-of-the-art PUD system
- Small area cost to a DRAM chip (**1.6%**) and CPU die (**0.03%**)

Outline

1 Introduction

2 Background

3 PUD's Limitations & Key Ideas

4 *Proteus*

5 Evaluation

6 Conclusion

Outline

1 Introduction

2 Background

3 PUD's Limitations & Key Ideas

4 *Proteus*

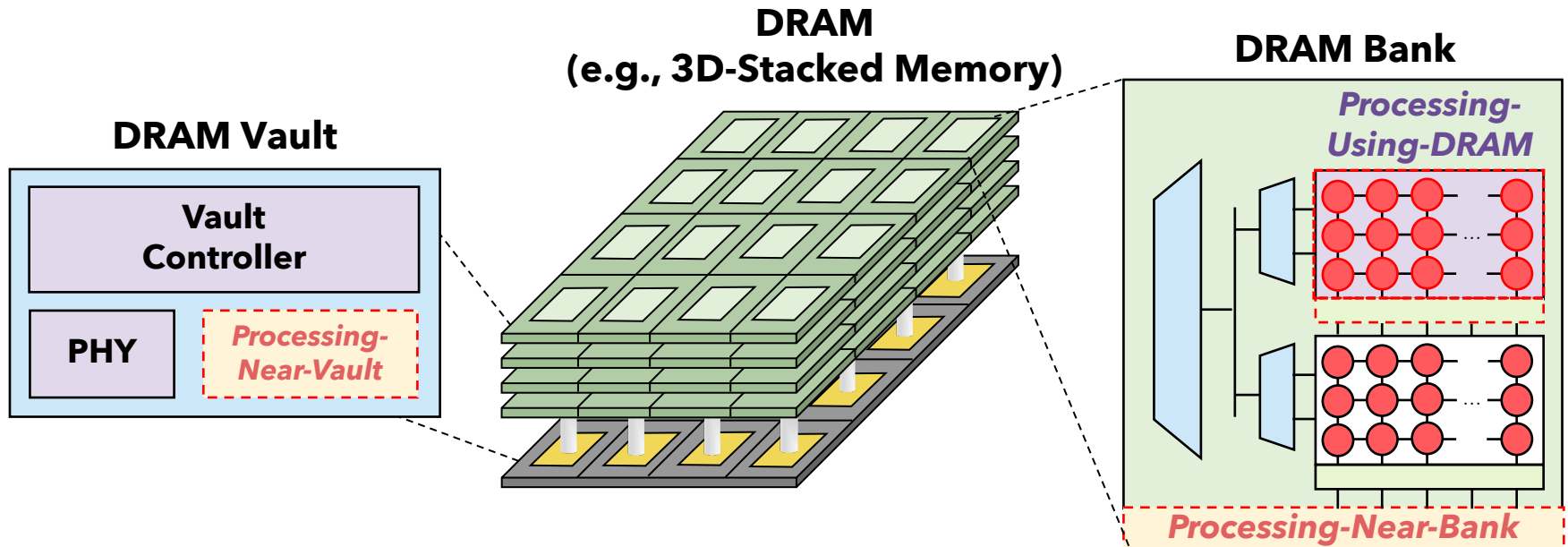
5 Evaluation

6 Conclusion

Processing-in-Memory: Overview (I)

Two main approaches for Processing-in-Memory:

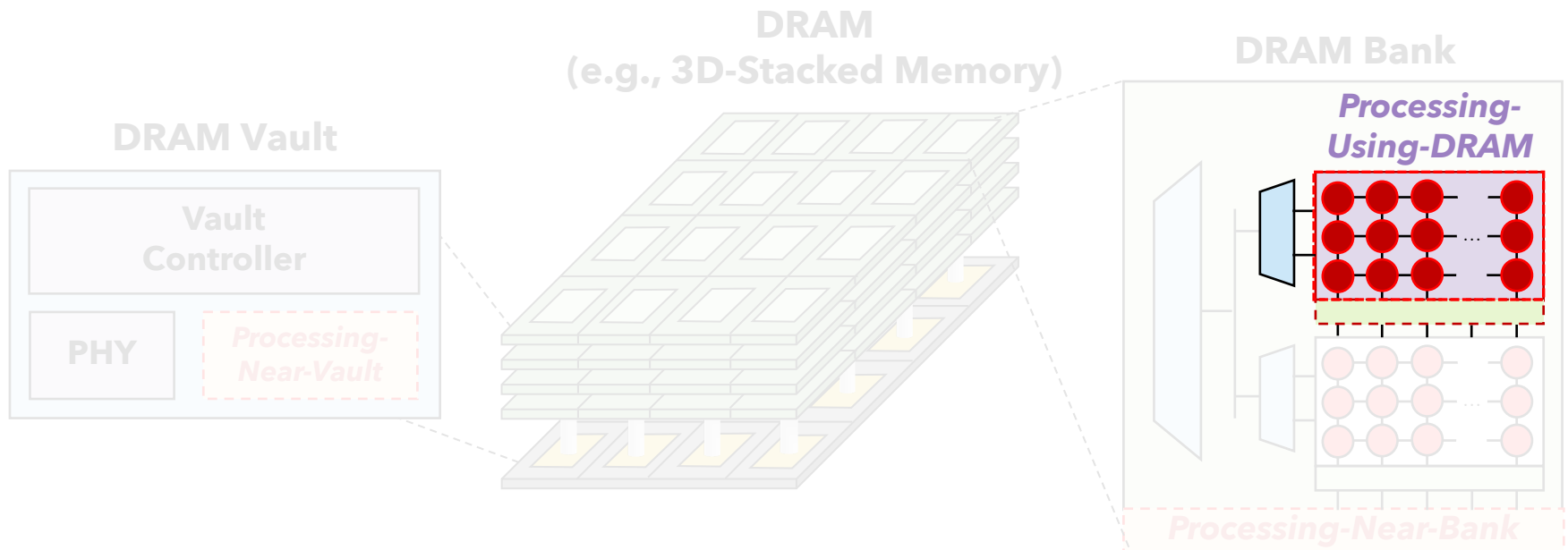
- 1 Processing-Near-Memory:** PIM logic is added to the same die as memory or to the logic layer of 3D-stacked memory
- 2 Processing-Using-Memory:** uses the operational principles of memory cells to perform computation



Processing-in-Memory: Overview (II)

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



Outline

1 Introduction

2 Background

3 PUD's Limitations & Key Ideas

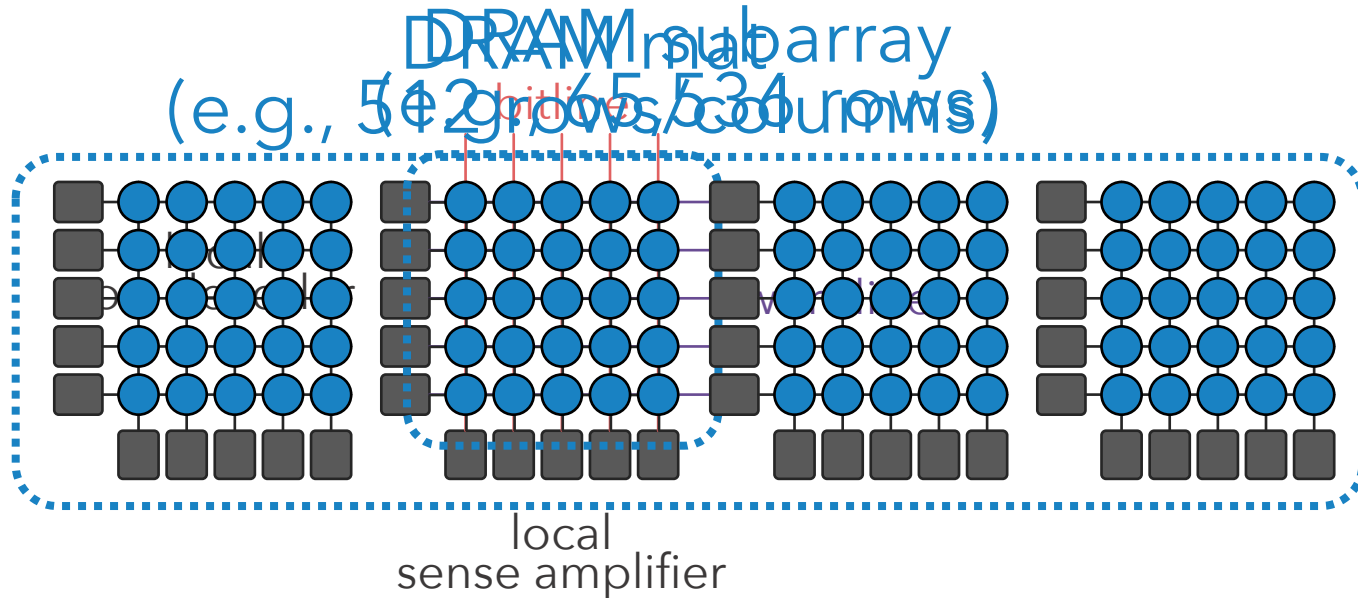
4 *Proteus*

5 Evaluation

6 Conclusion

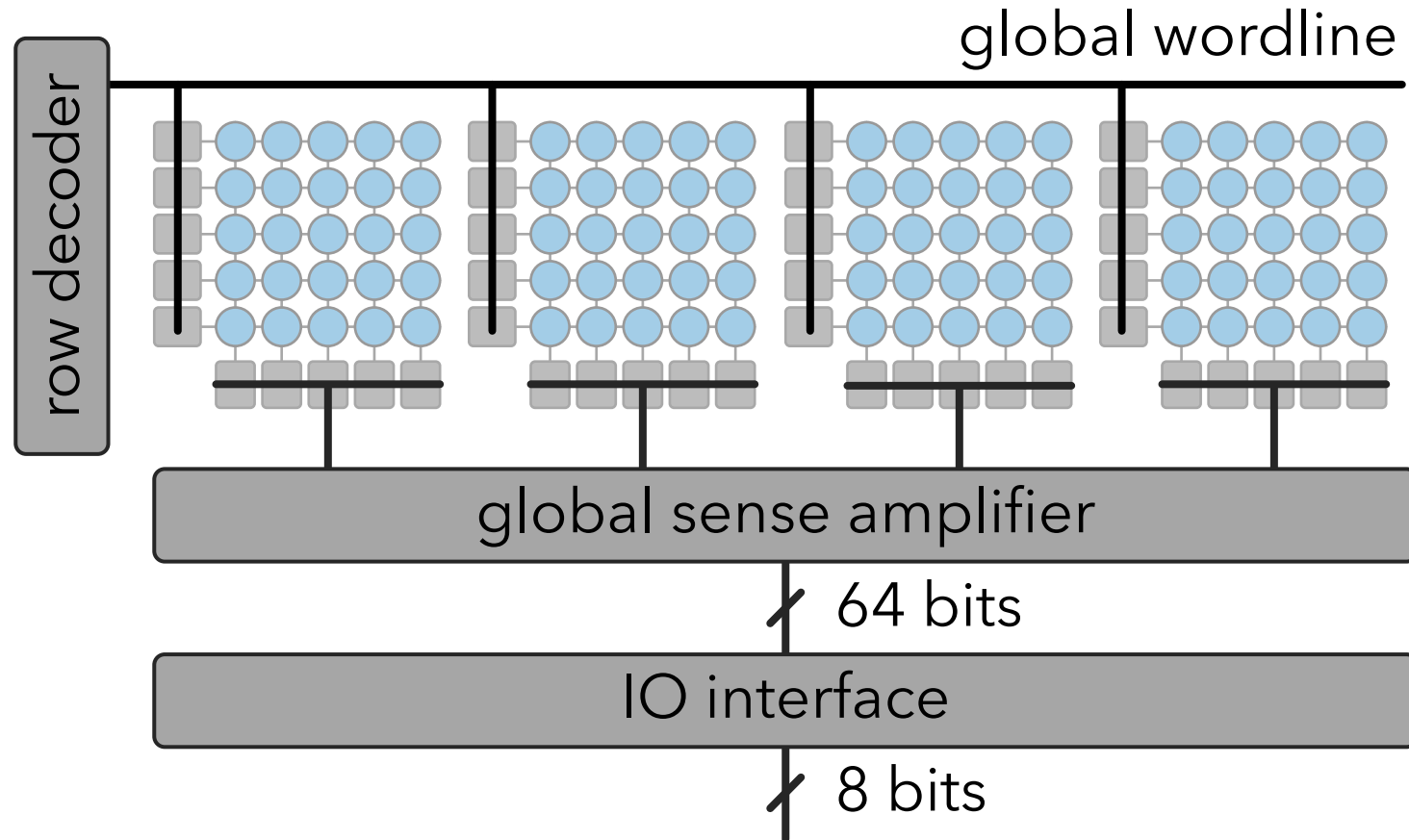
Background:

DRAM Hierarchical Organization (I)



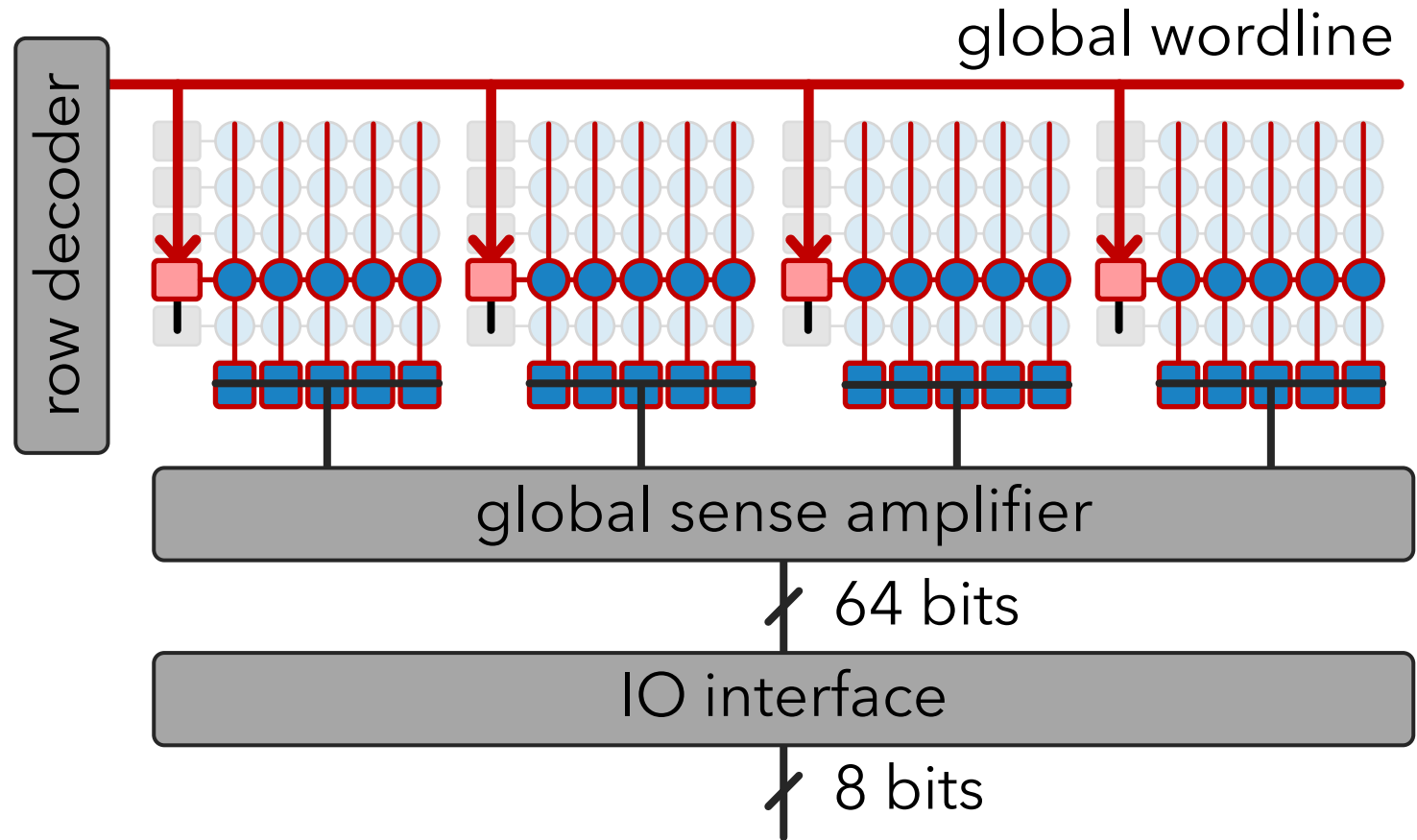
Background:

DRAM Hierarchical Organization (II)



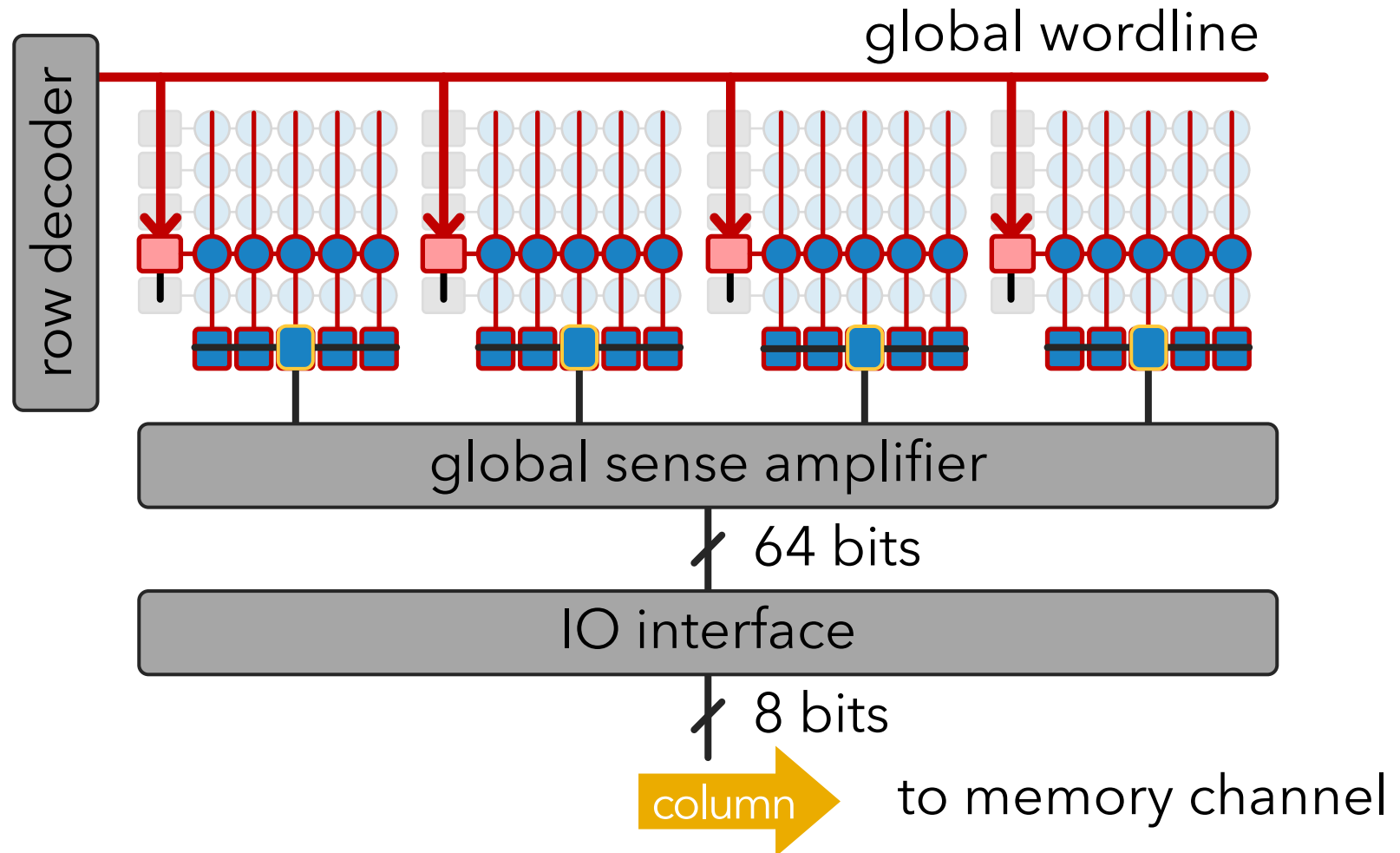
Background:

DRAM Operation - Row Access (ACTIVATE)



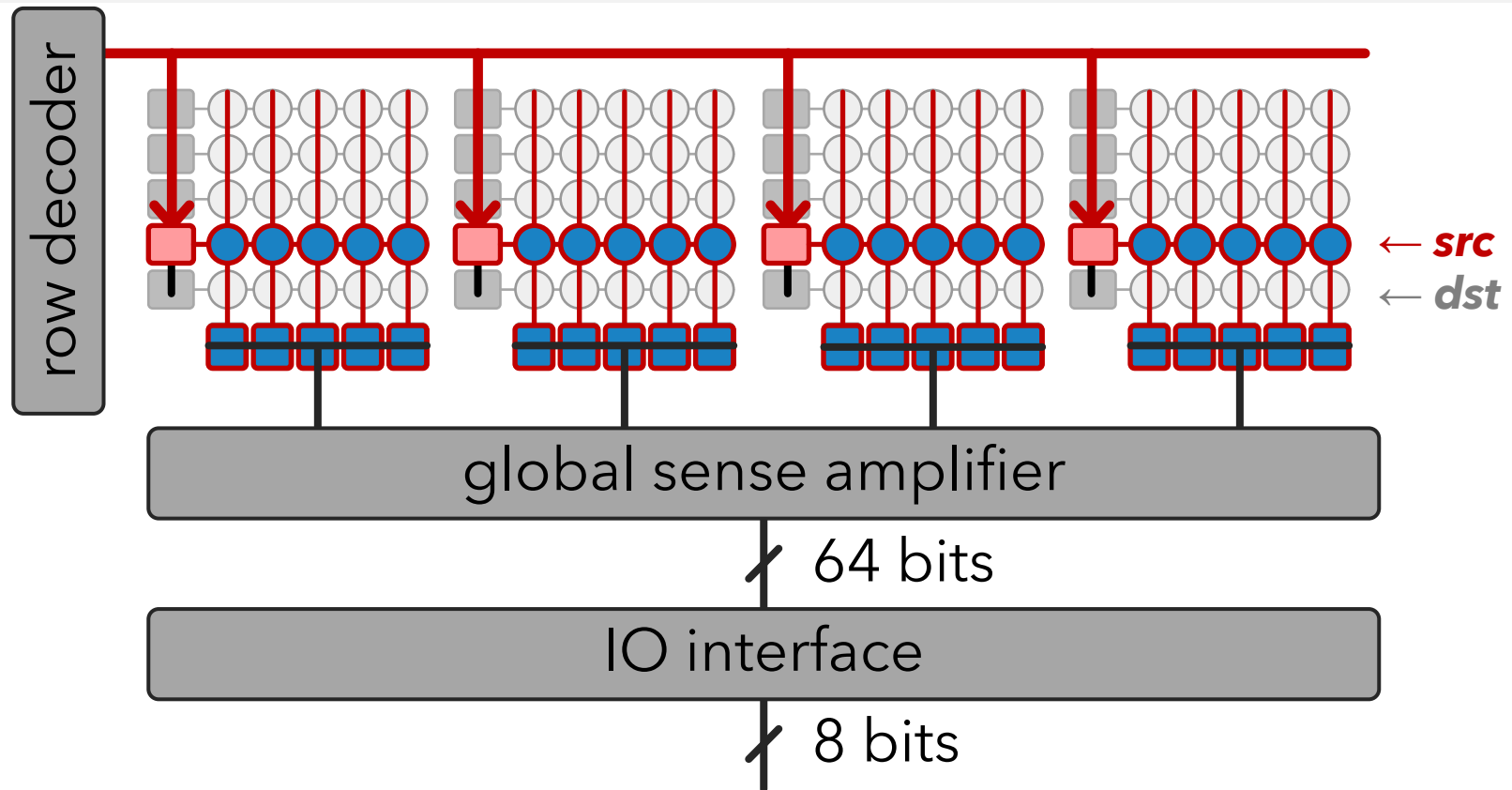
Background:

DRAM Operation - Column Access (READ)



Background: In-DRAM Row Copy

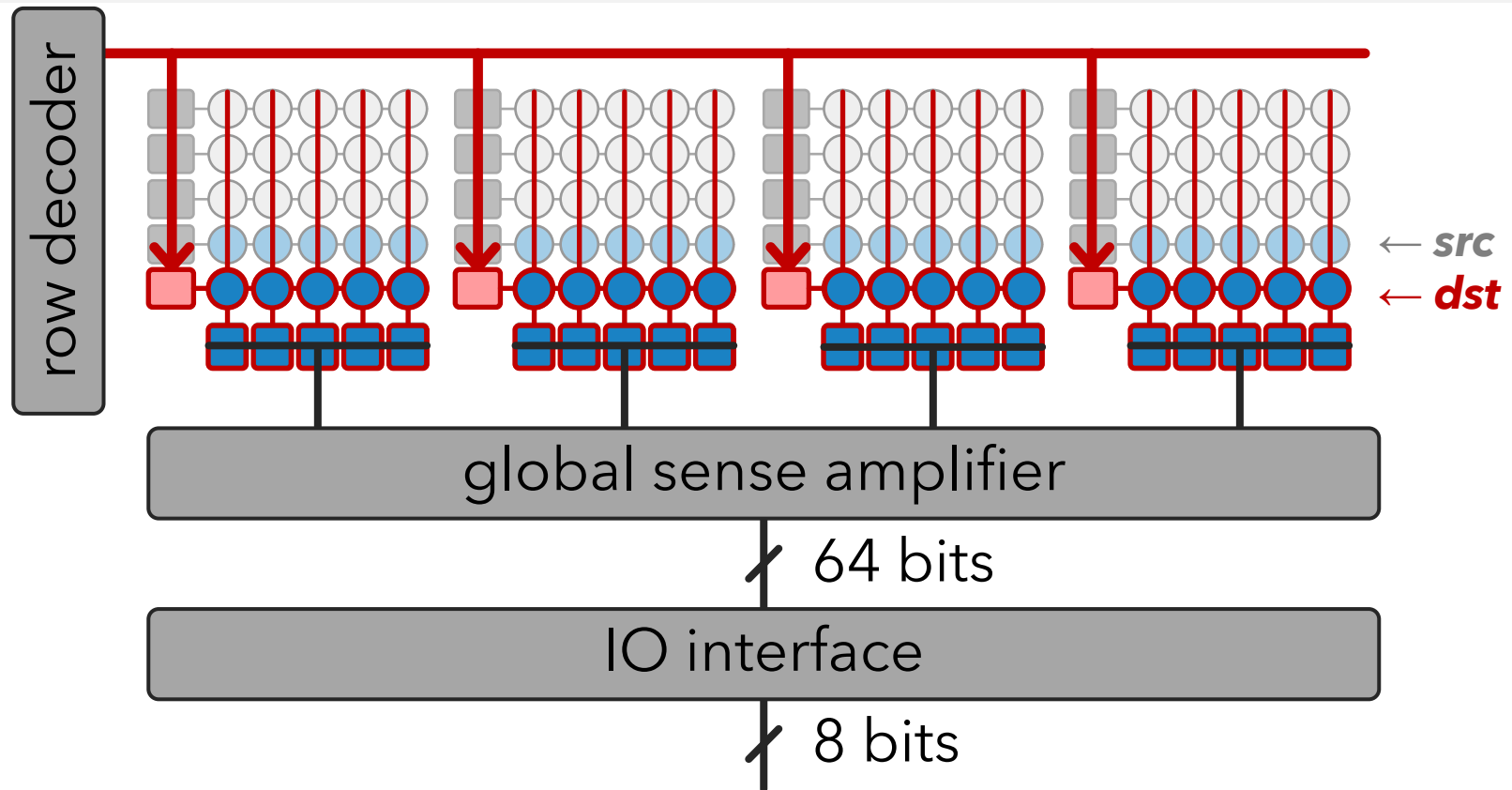
In-DRAM row copy is performed by
issuing back-to-back **ACTIVATES** to the DRAM



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

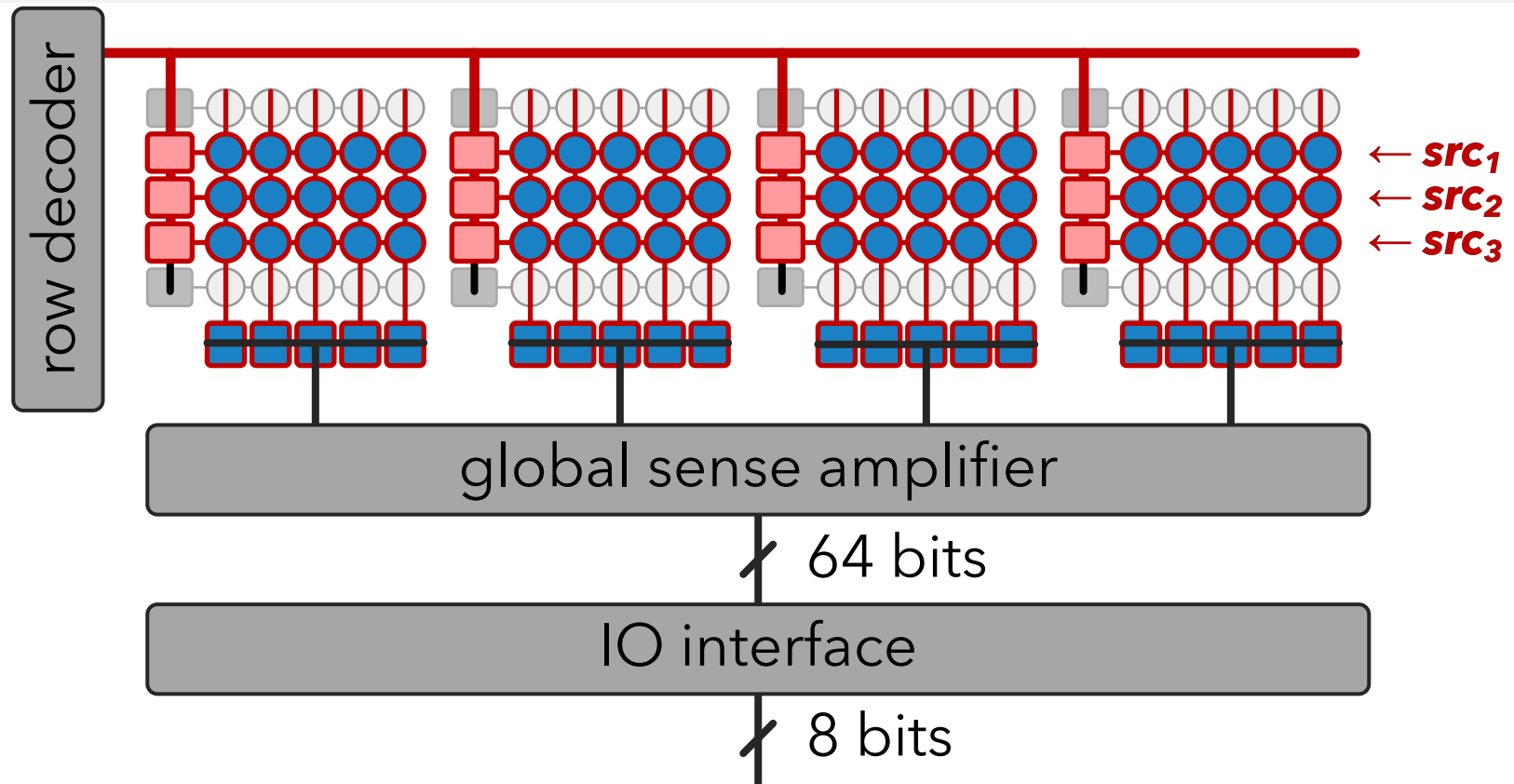


Seshadri, Vivek, et al. "RowClone: Fast and Energy-Efficient In-DRAM Bulk Data Copy and Initialization," in MICRO, 2013

Background:

In-DRAM Majority Operation

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

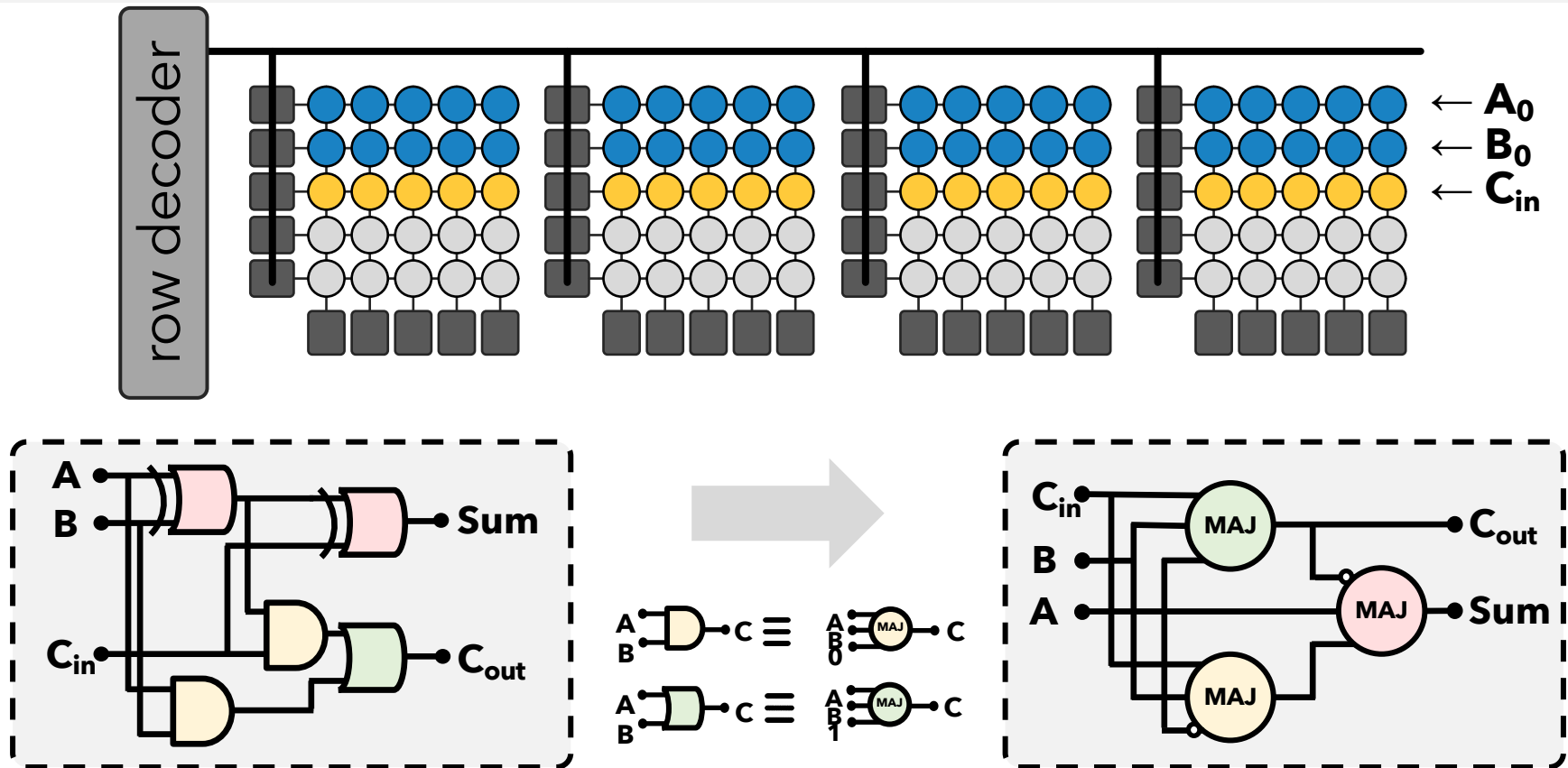


Seshadri, Vivek, et al. " **Ambit: In-Memory Accelerator for Bulk Bitwise Operations Using Commodity DRAM Technology**," in MICRO, 2017

Background:

Bulk Bitwise Arithmetic Operations (I)

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



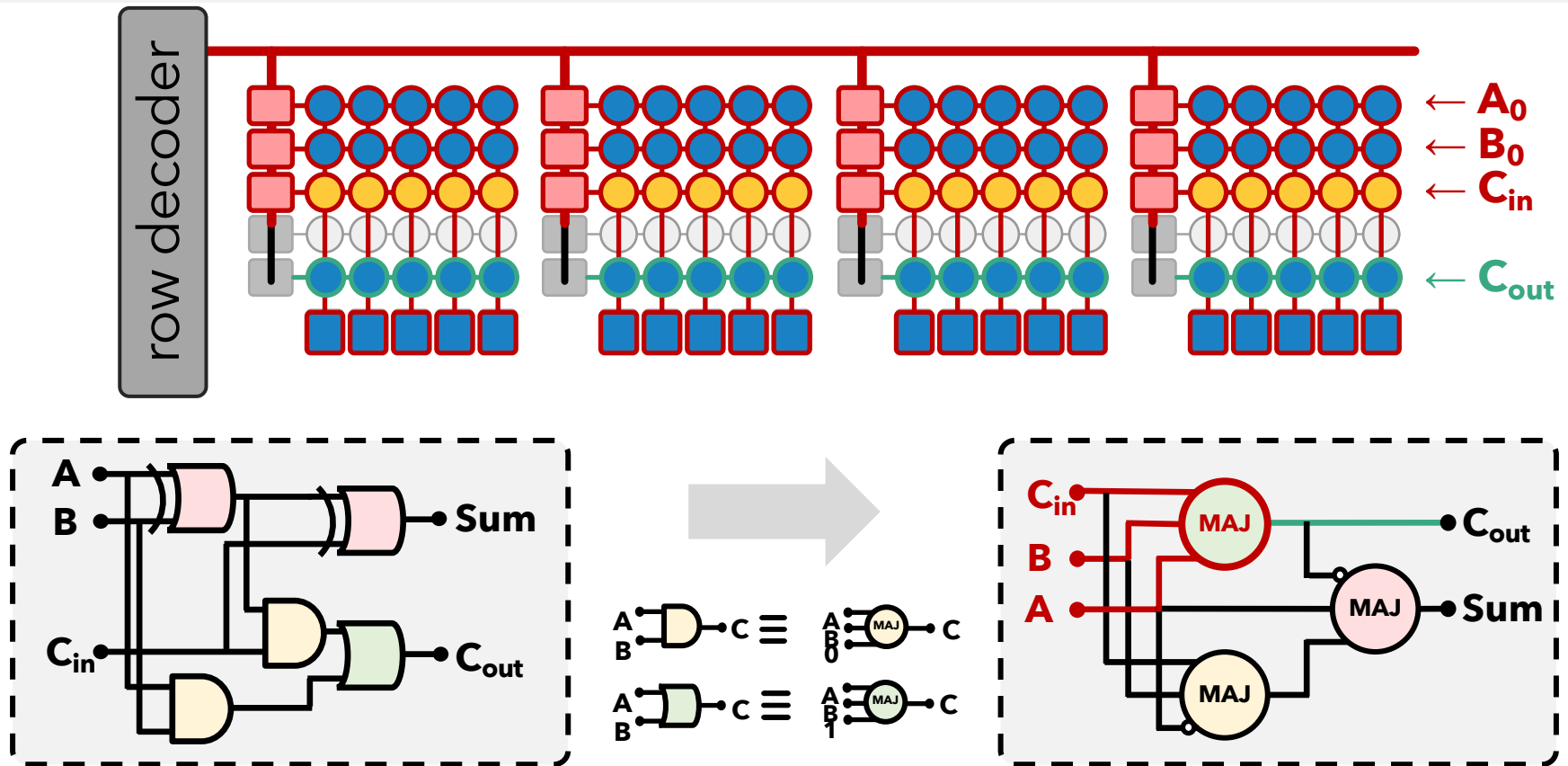
Oliveira, Geraldo F., et al.

"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



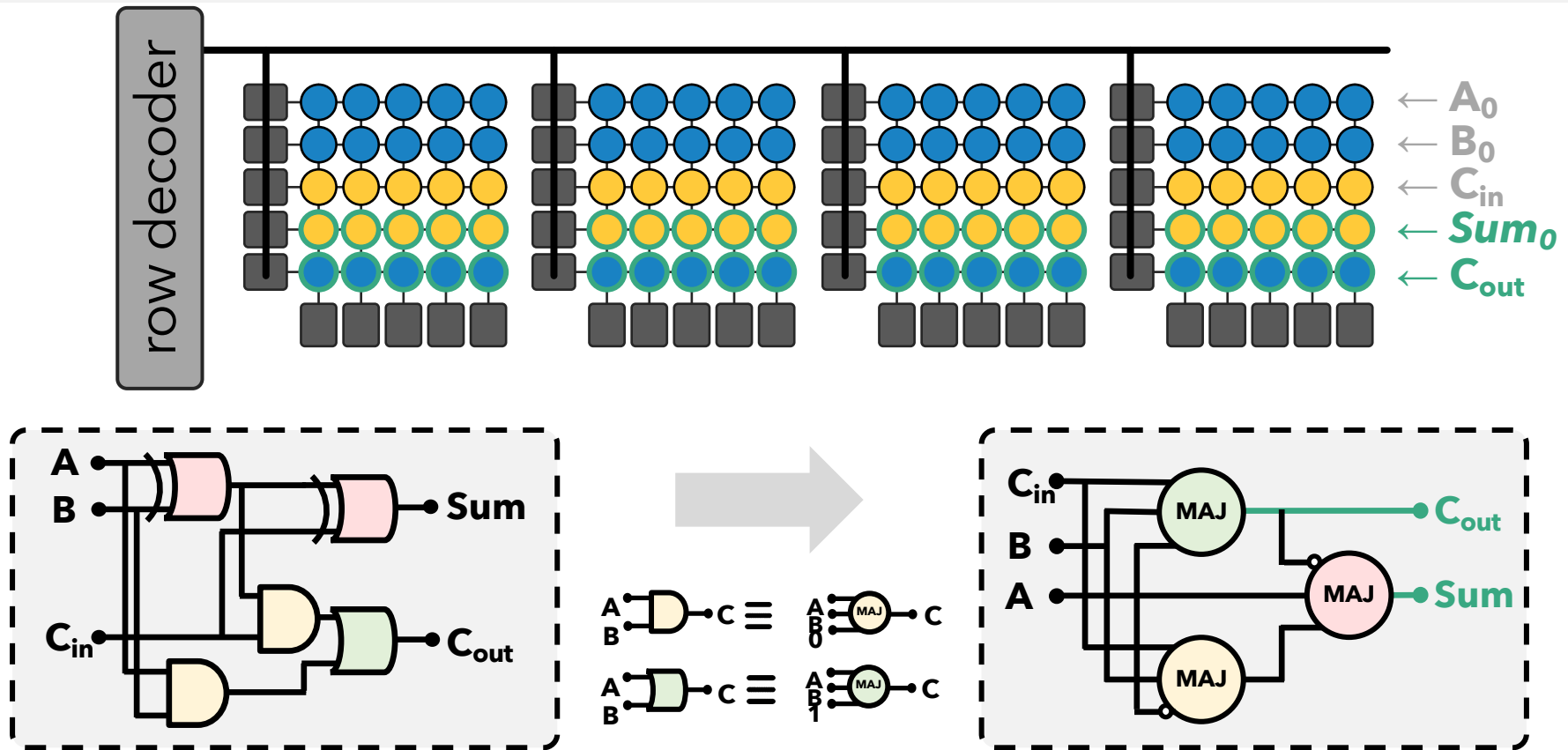
Oliveira, Geraldo F., et al.

"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



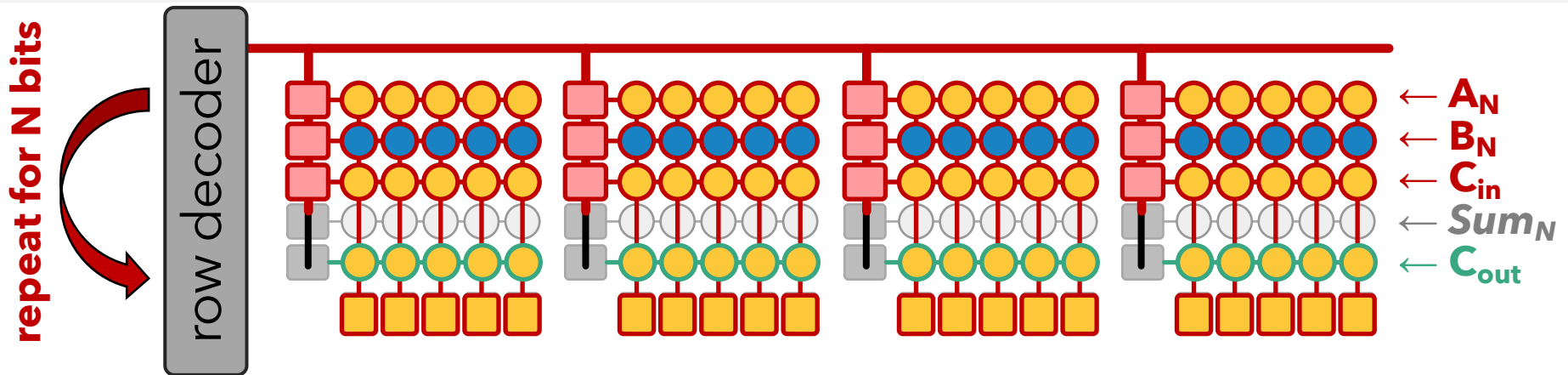
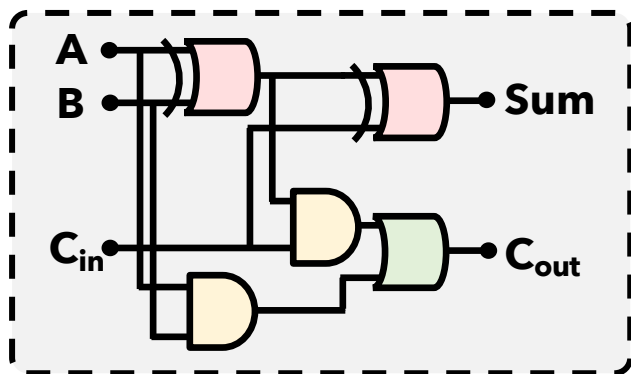
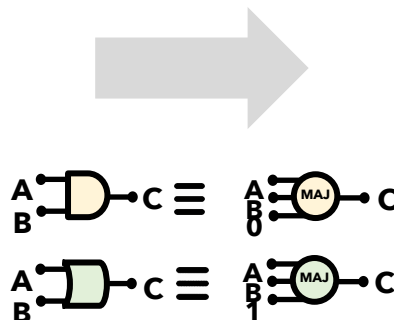
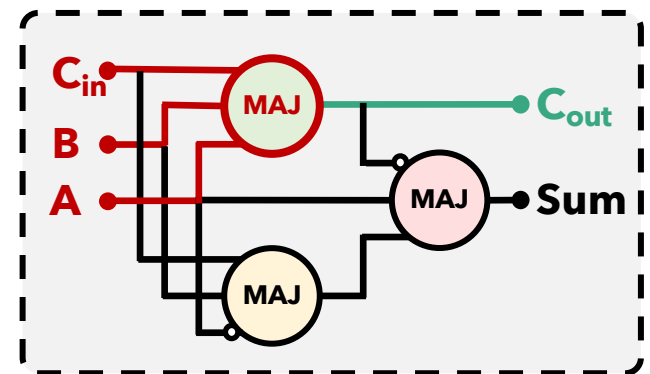
Oliveira, Geraldo F., et al.

"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



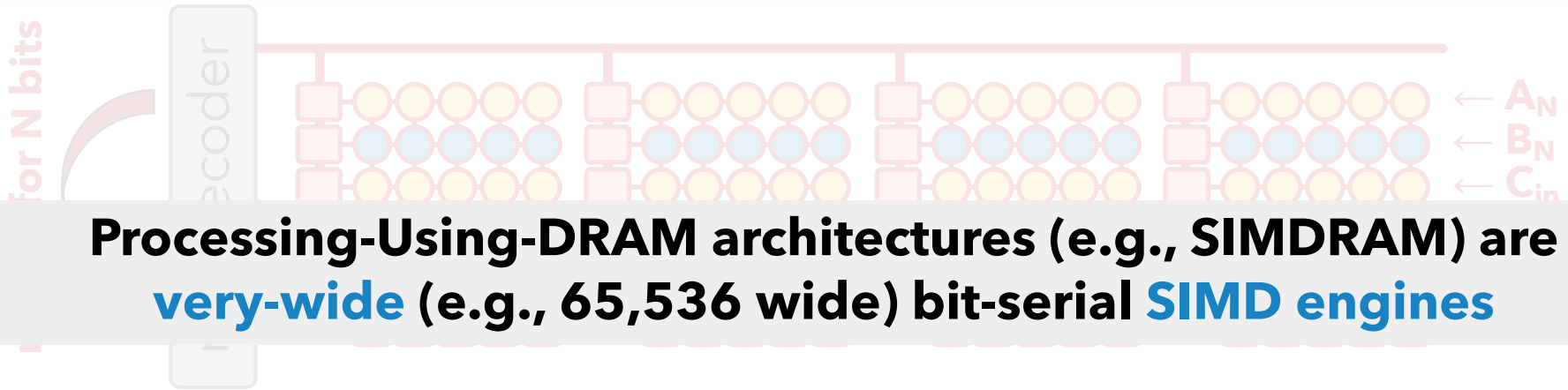
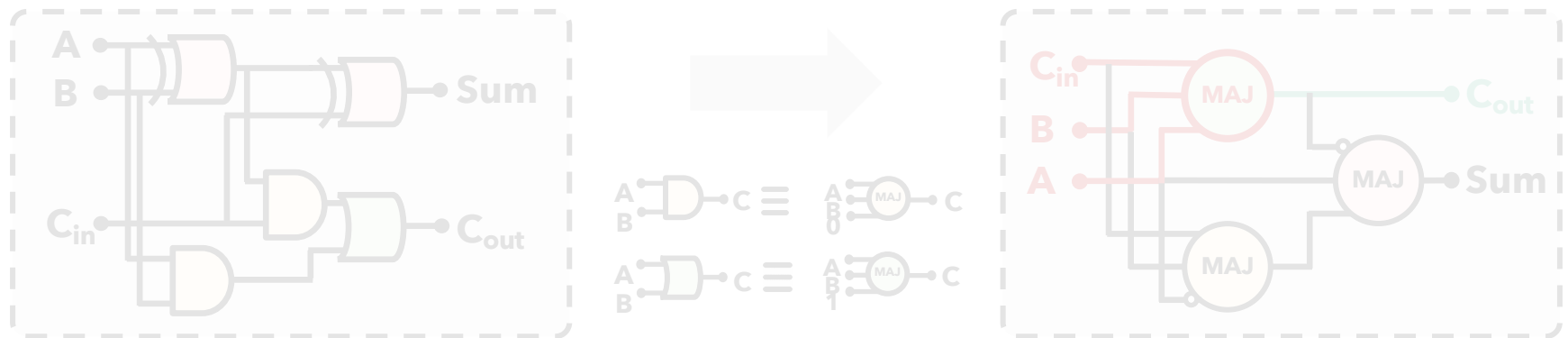
Oliveira, Geraldo F., et al.

"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



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

1 Introduction

2 Background

3 PUD's Limitations & Key Ideas

4 *Proteus*

5 Evaluation

6 Conclusion

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:

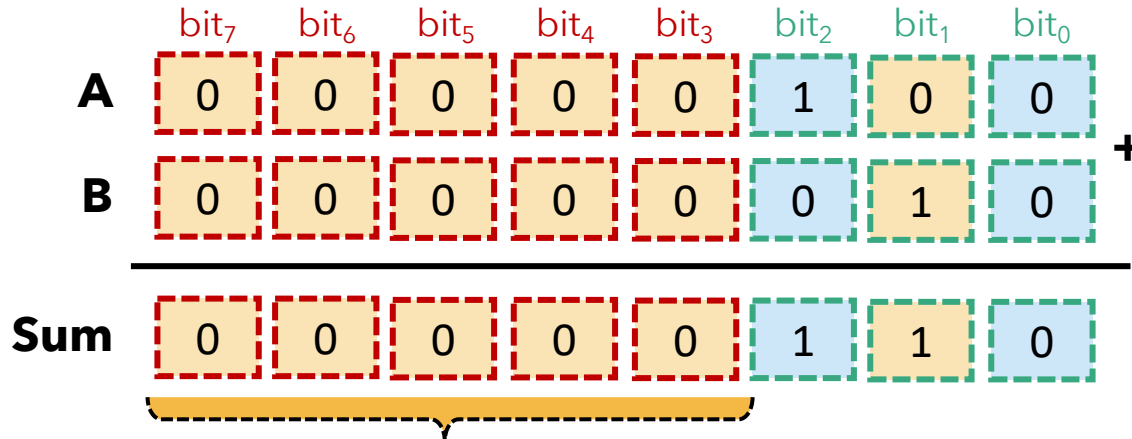
data with small dynamic range stored in large data formats

	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	

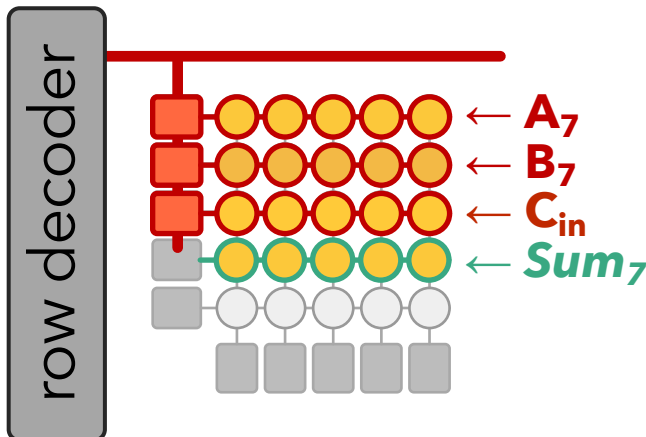
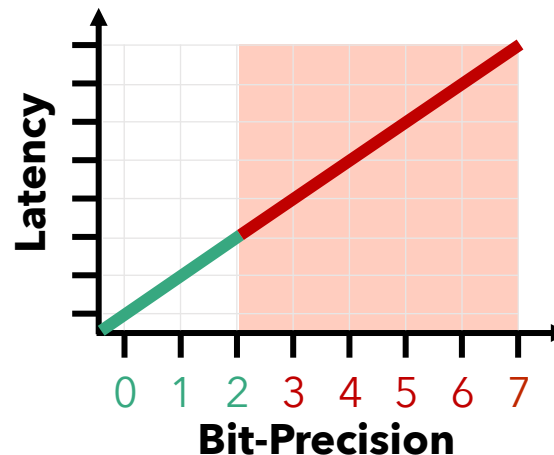
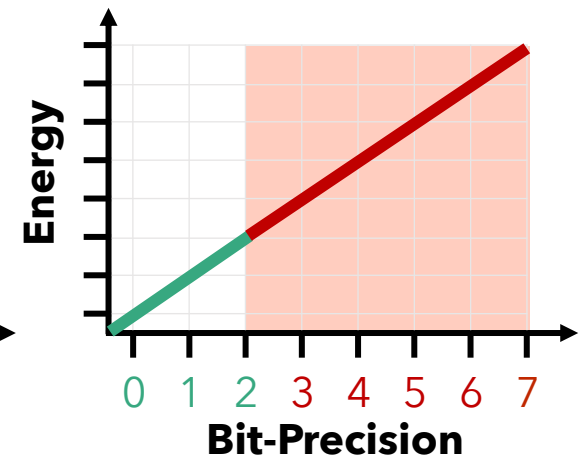
Limitations of Bit-Serial PUD Systems: Rigid and Static Data Representation (I)

Narrow Values:

data with small dynamic range stored in large data formats



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



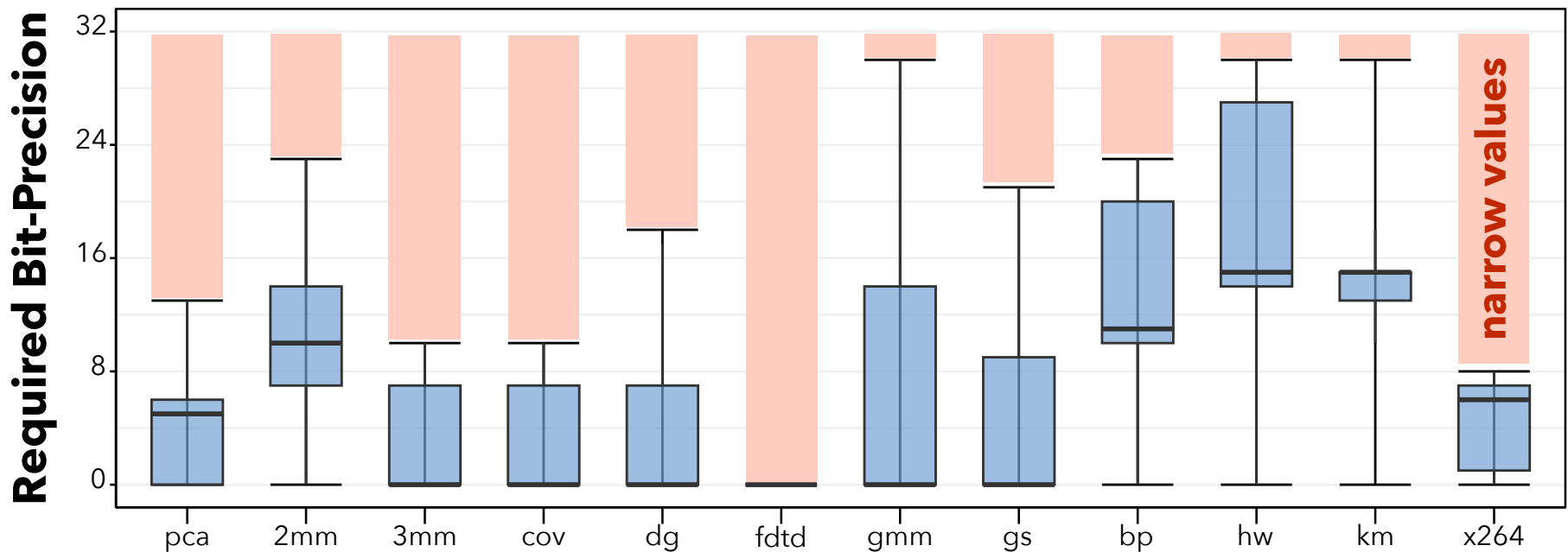
-----> minimum number of bits required to represent input operands

Limitations of Bit-Serial PUD Systems:

Rigid and Static Data Representation (II)

Application Analysis:

quantify the amount **required bit-precision** in real applications



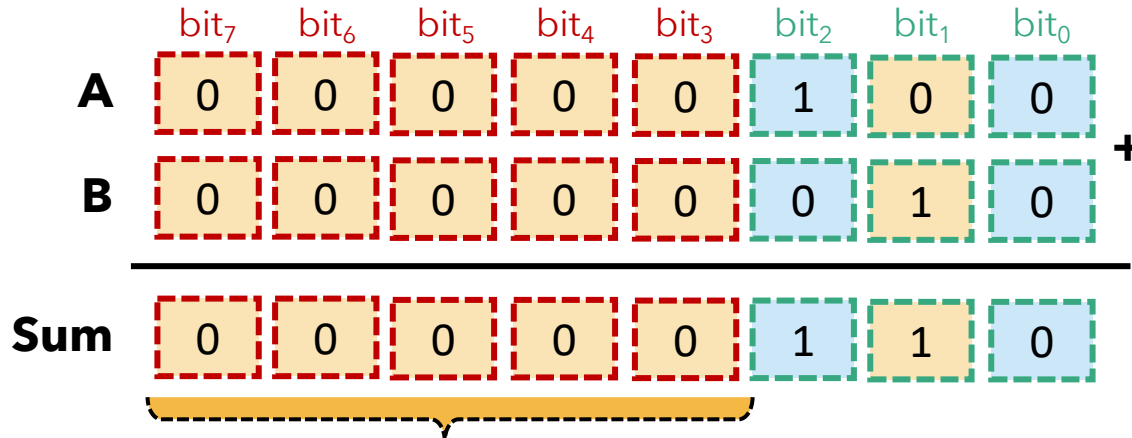
Applications display a significant amount of narrow values → bit-precision can be reduced from 32 bits to 20 bits, on average

Opportunities for Bit-Serial PUD:

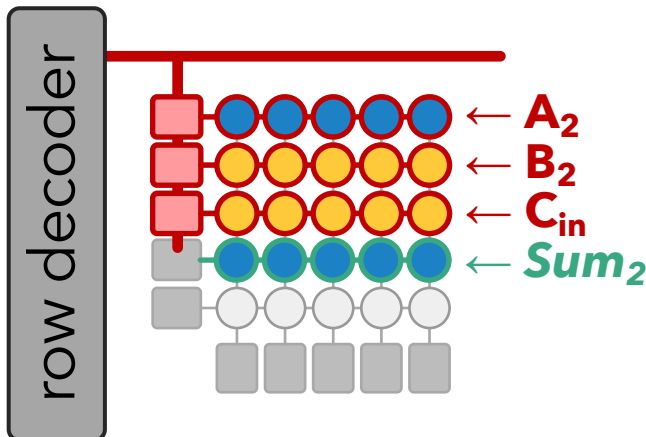
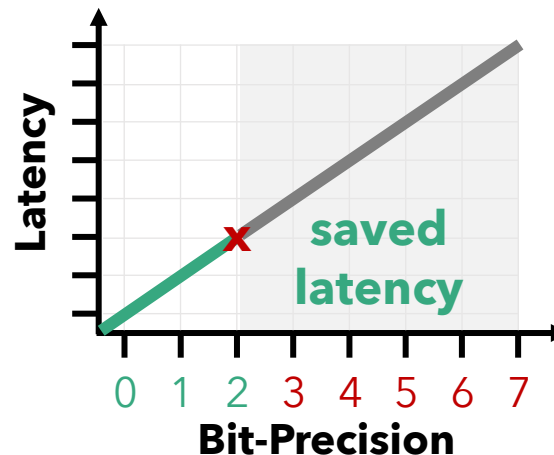
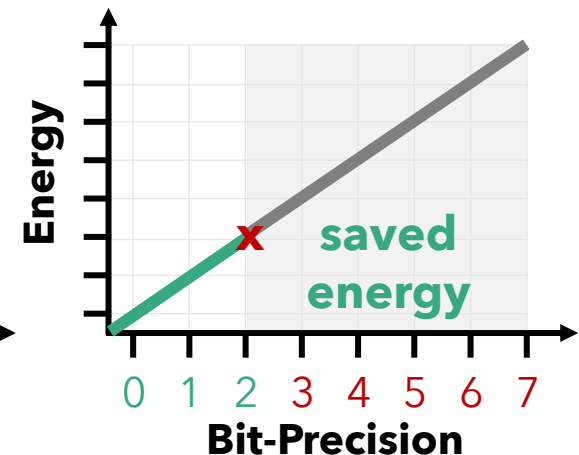
Narrow Values for PUD Operations

Narrow Values:

data with small dynamic range stored in large data formats



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

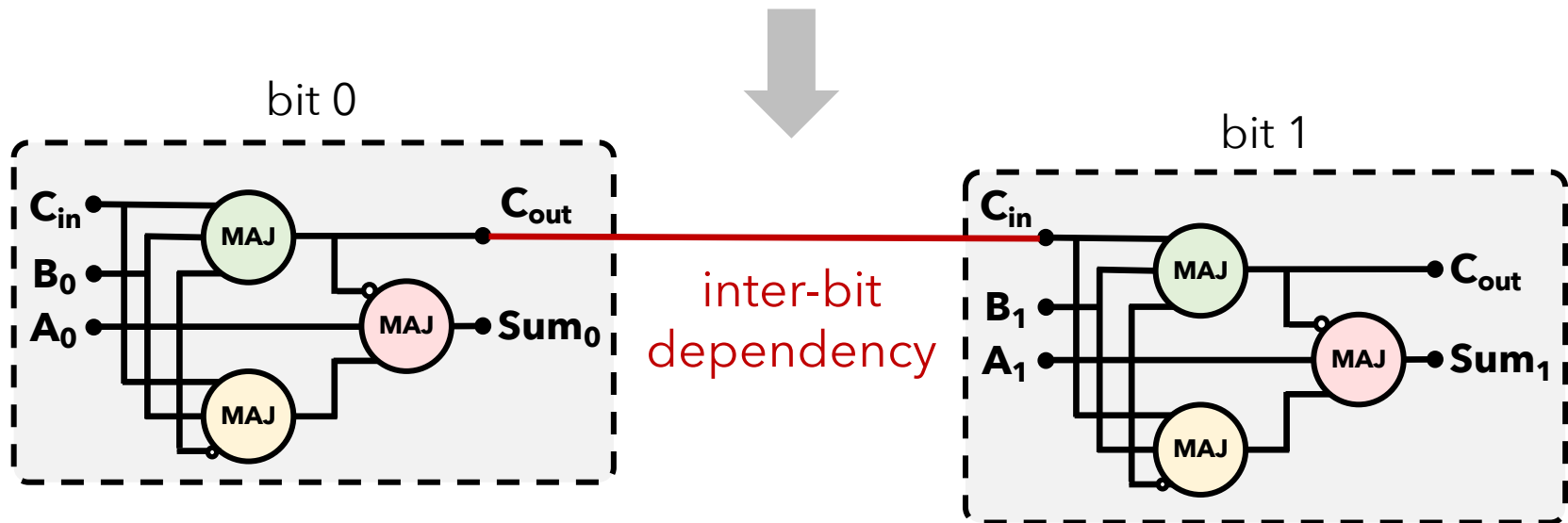
3 Scalability Challenges for High-Precision Operations

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

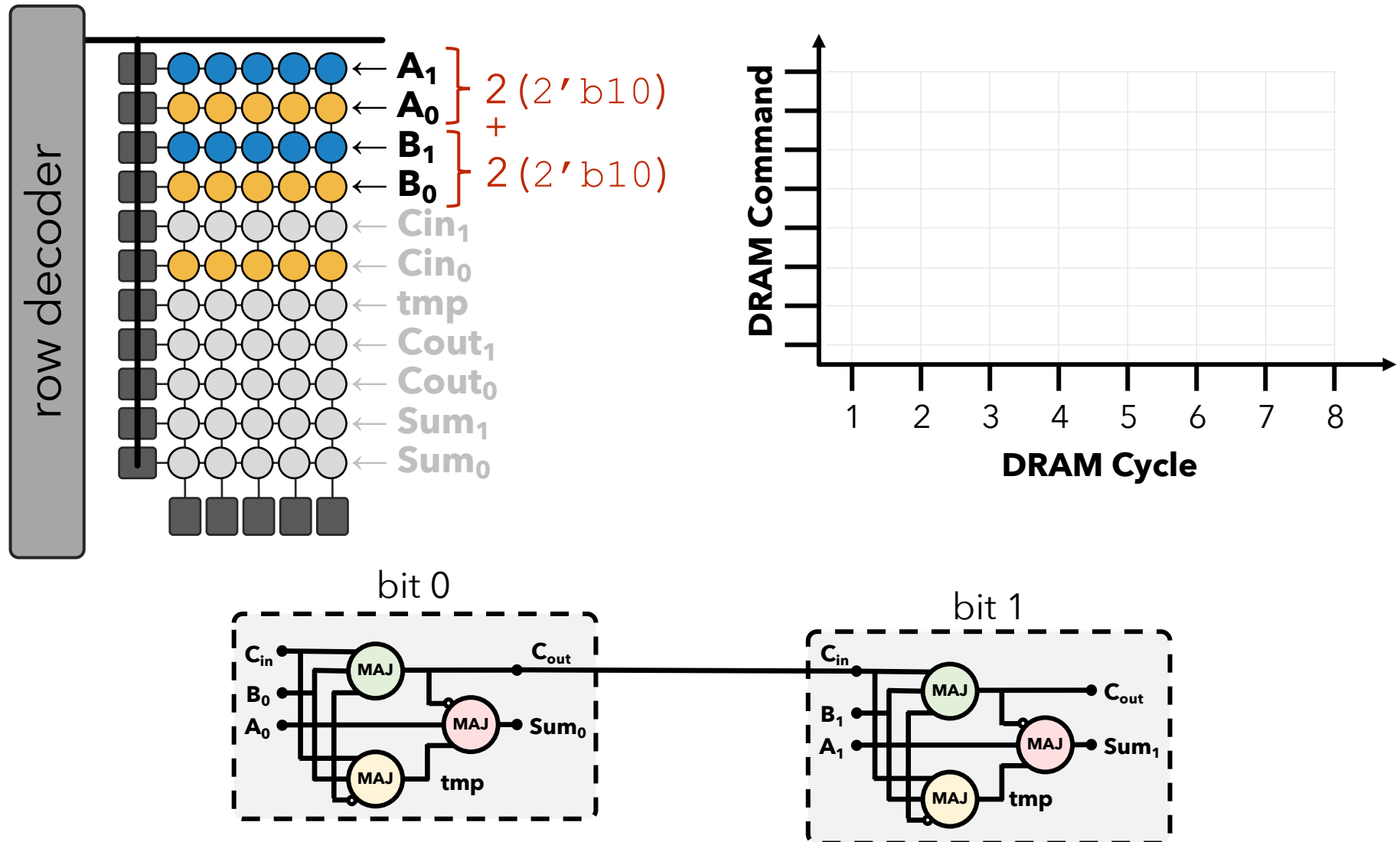
Limitations of Bit-Serial PUD Systems: Throughput-Oriented Execution (I)

Problem

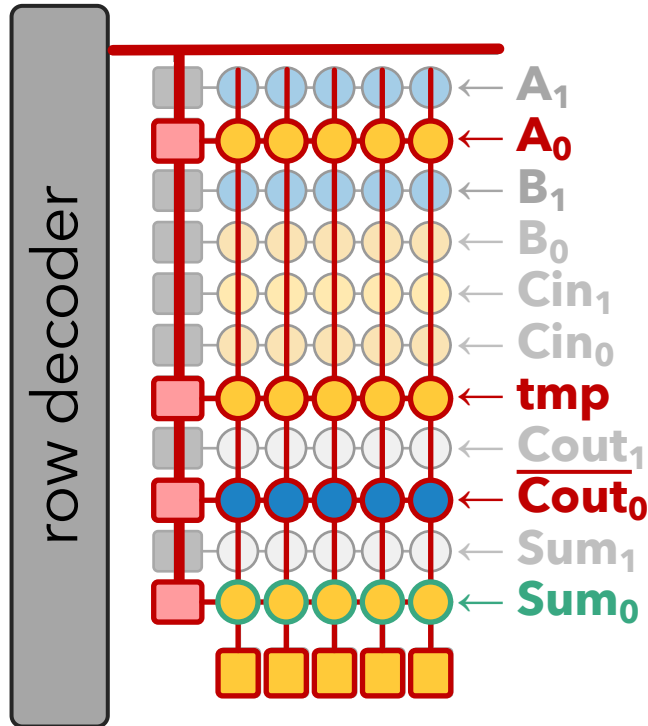
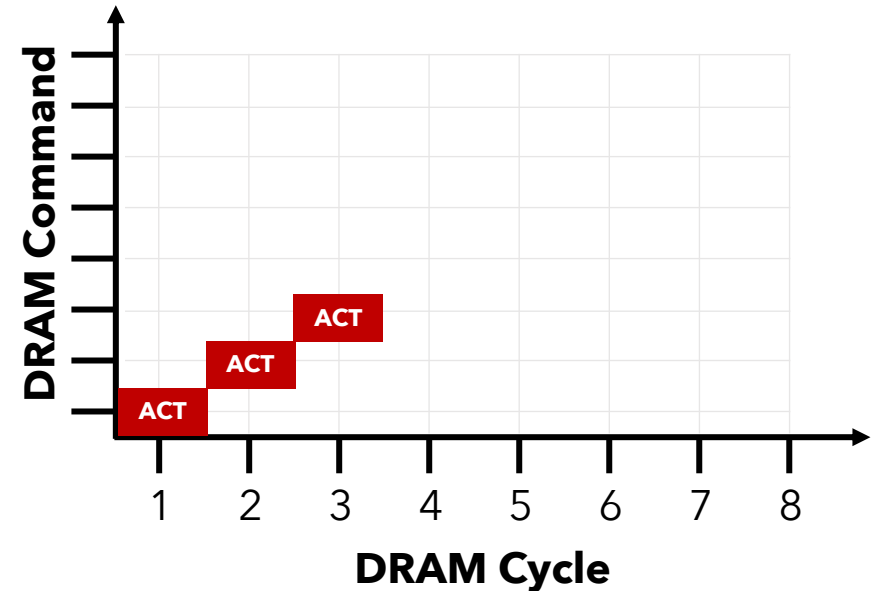
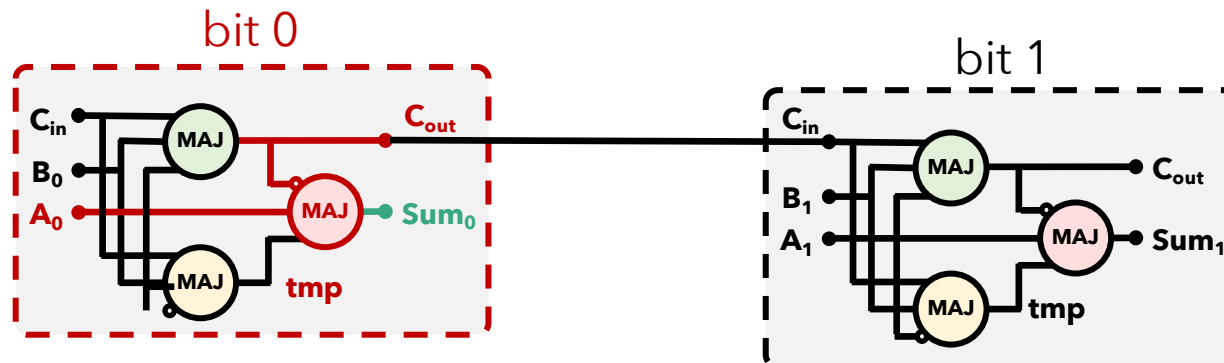
PUD operations are inherently slow
→ need to operate on each bit serially to perform an operation with a data width > 1



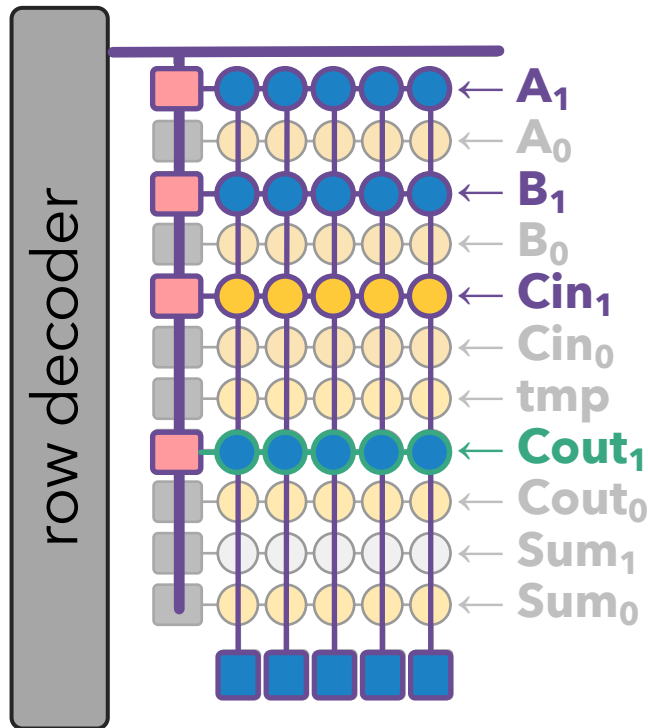
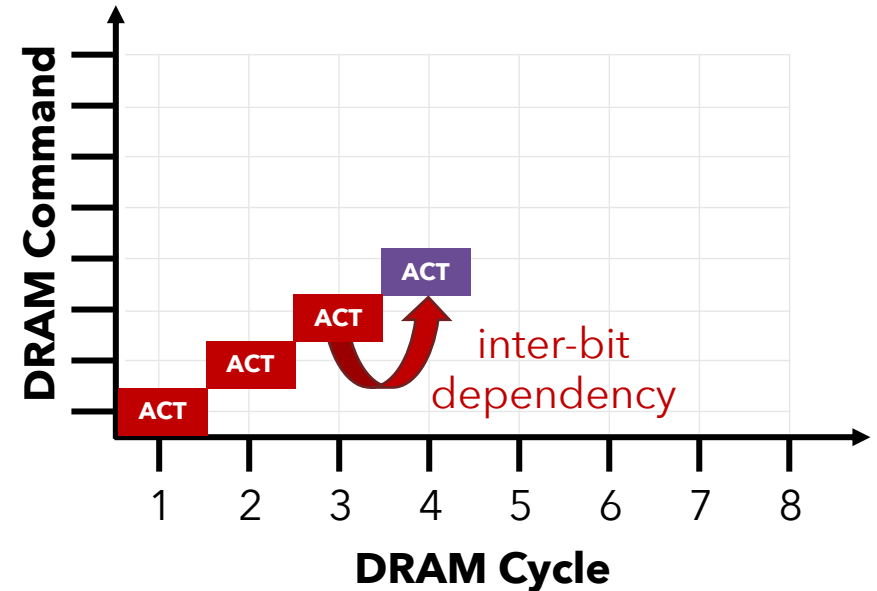
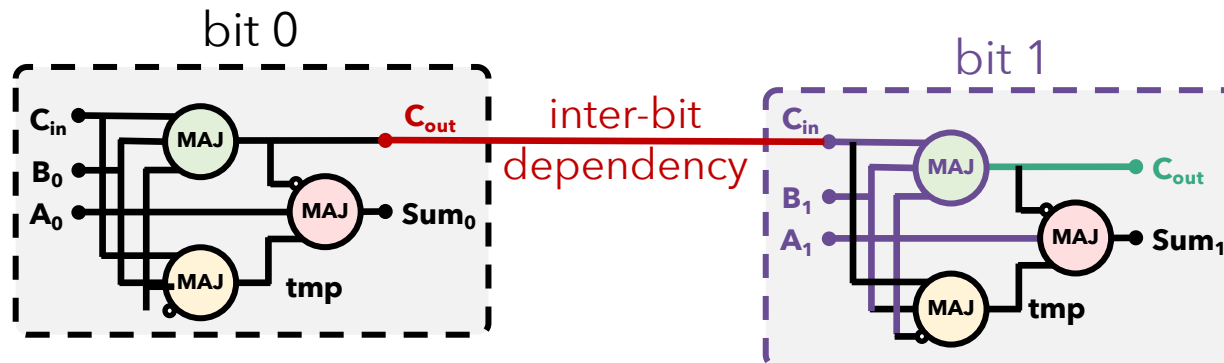
Limitations of Bit-Serial PUD Systems: Throughput-Oriented Execution (II)



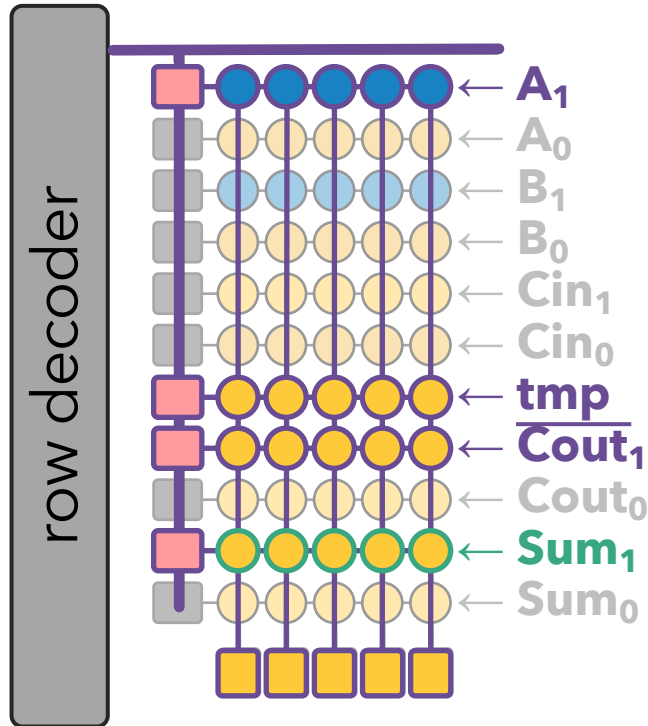
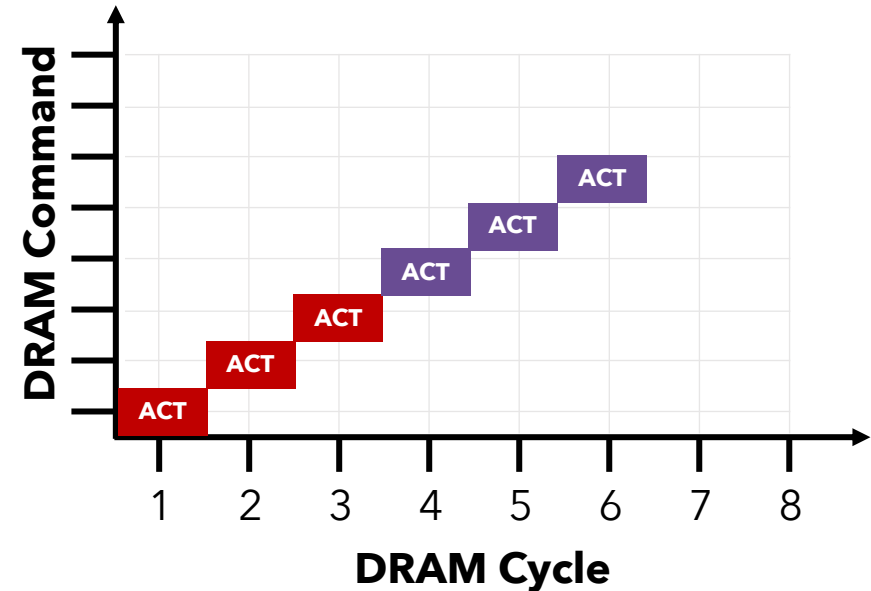
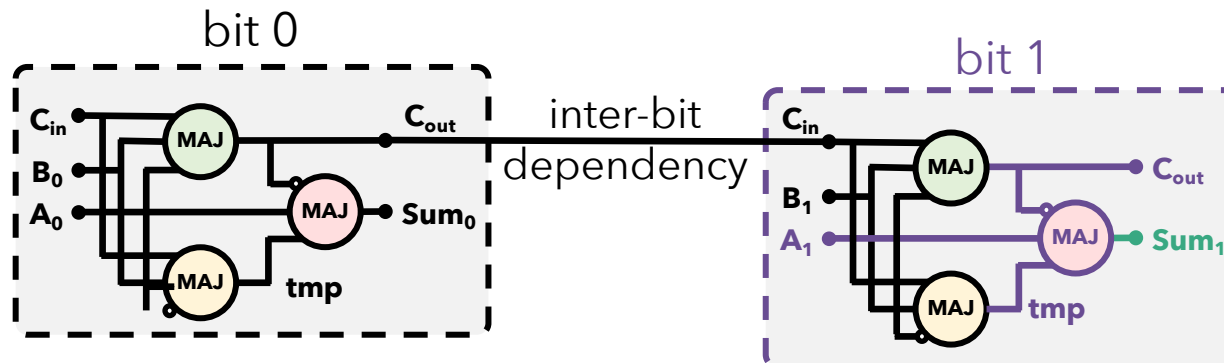
Limitations of Bit-Serial PUD Systems: Throughput-Oriented Execution (II)



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



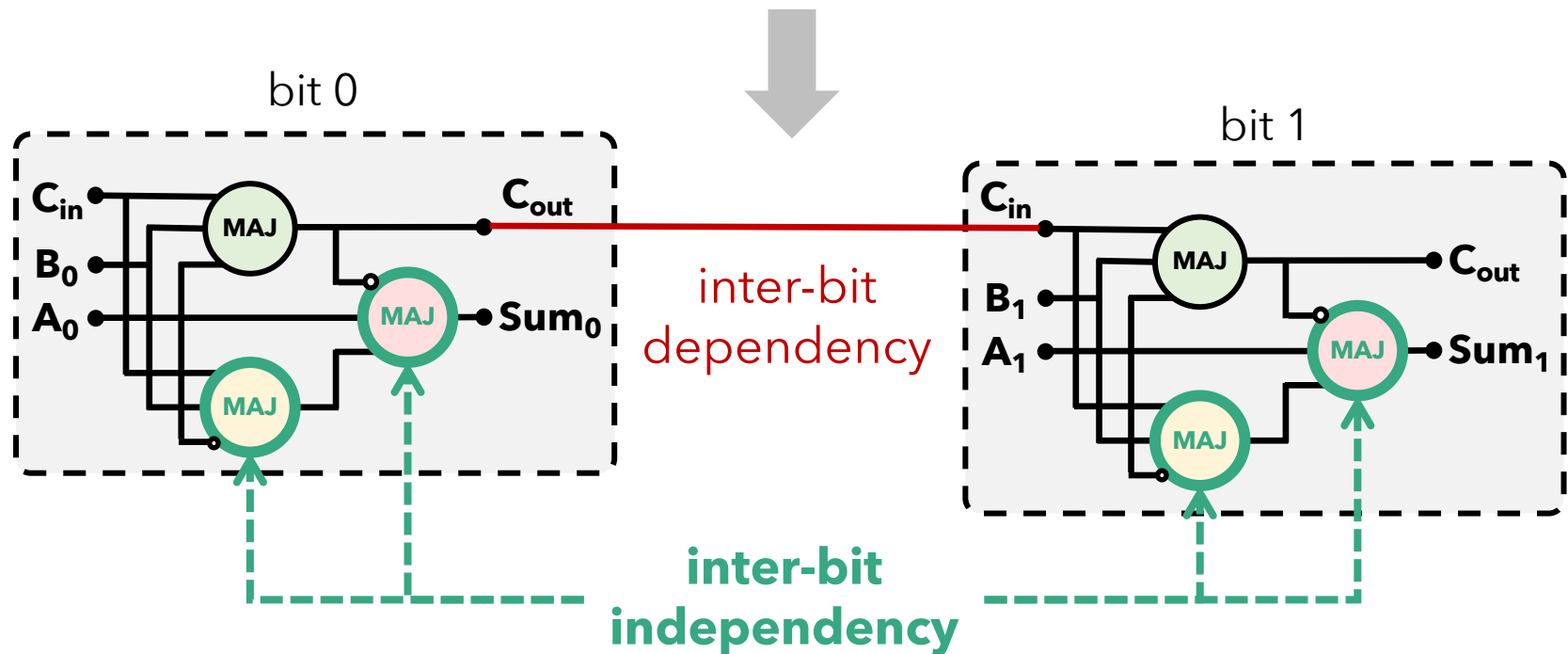
Limitations of Bit-Serial PUD Systems: Throughput-Oriented Execution (II)



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



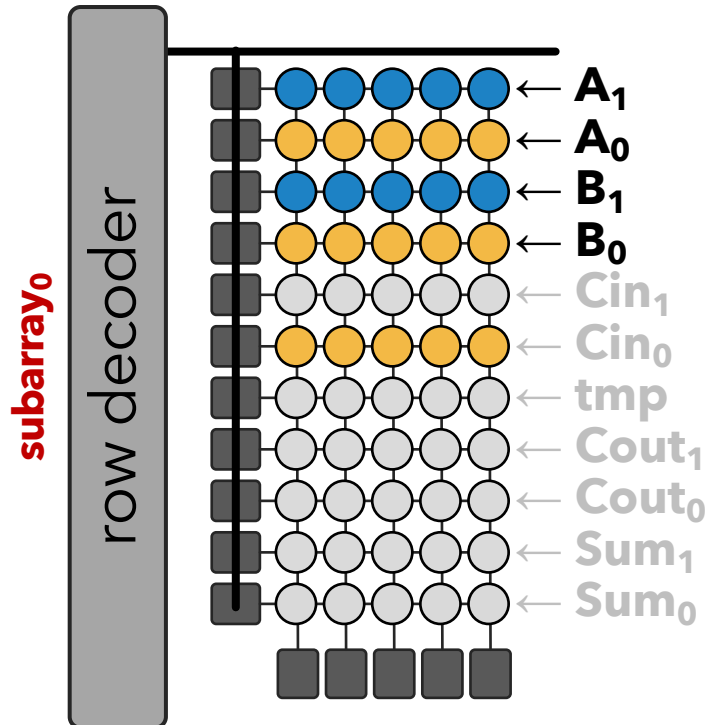
Opportunities for Bit-Serial PUD: Latency Reduction with Subarray Parallelism (II)

Proteus' one-bit per-subarray data mapping

→ scatter bits across subarrays to expose bit-level parallelism

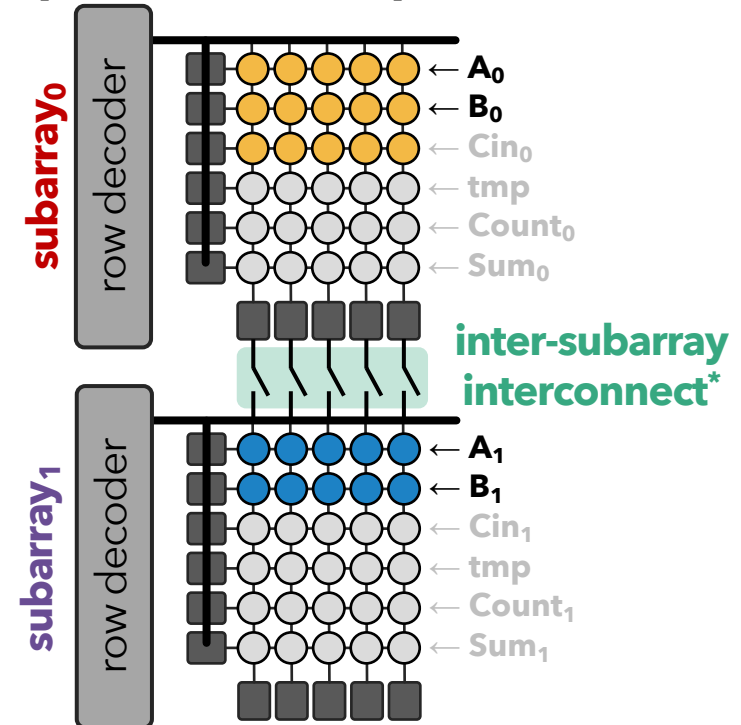
all-bits per-subarray

→ forces serialization across bits



one-bit per-subarray

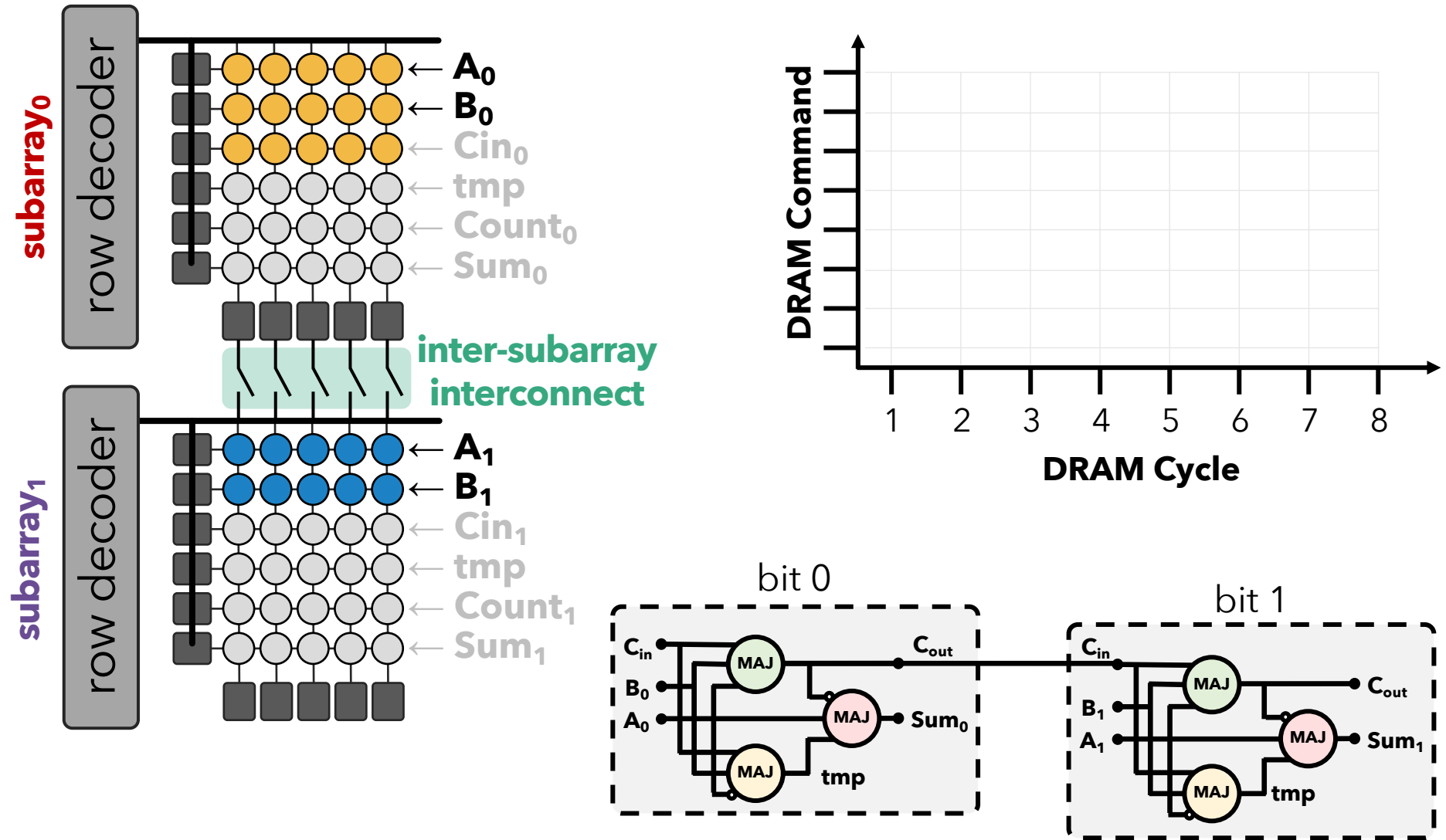
→ exposes bit-level parallelism



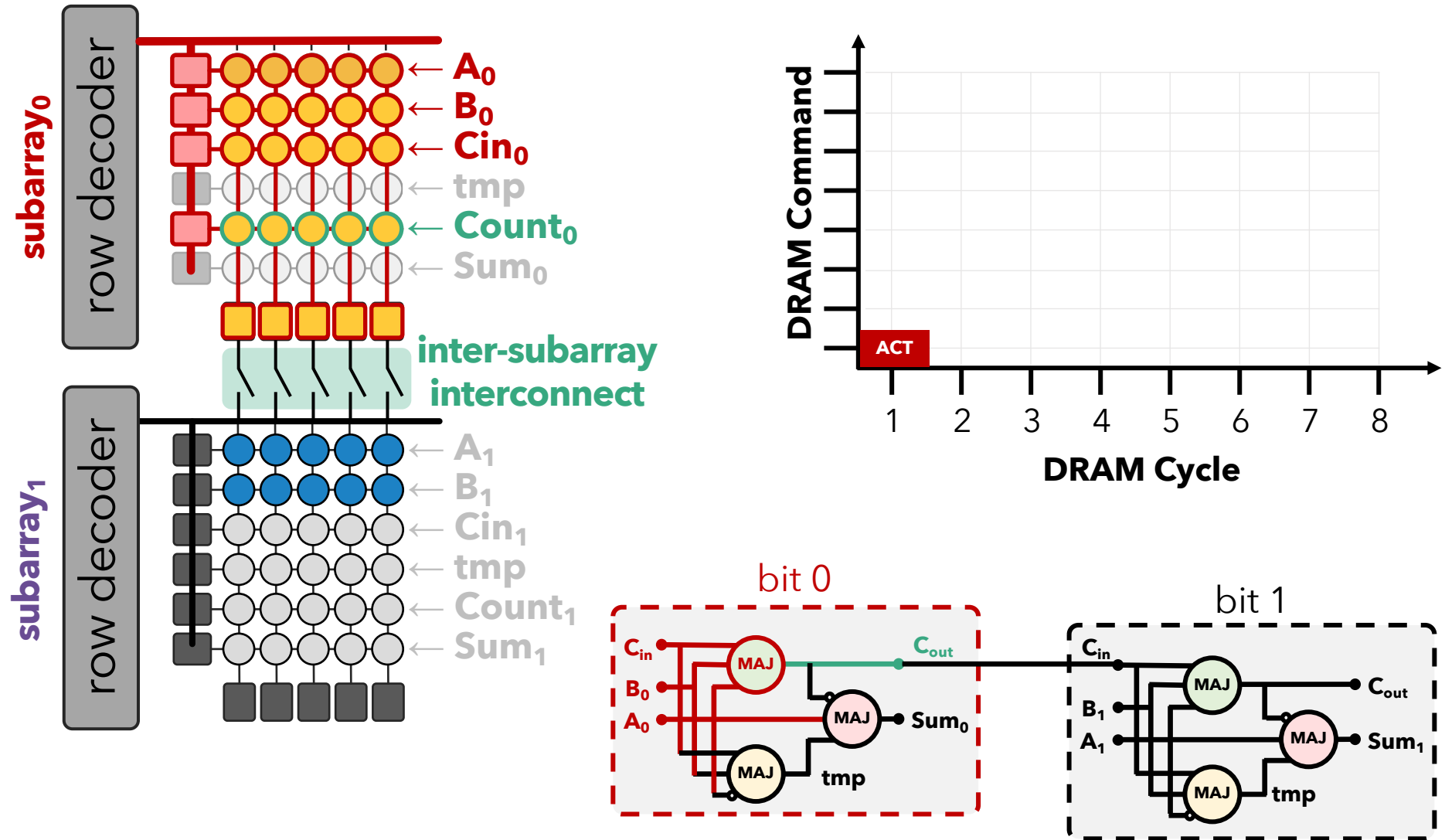
Chang, Kevin K. et al.

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

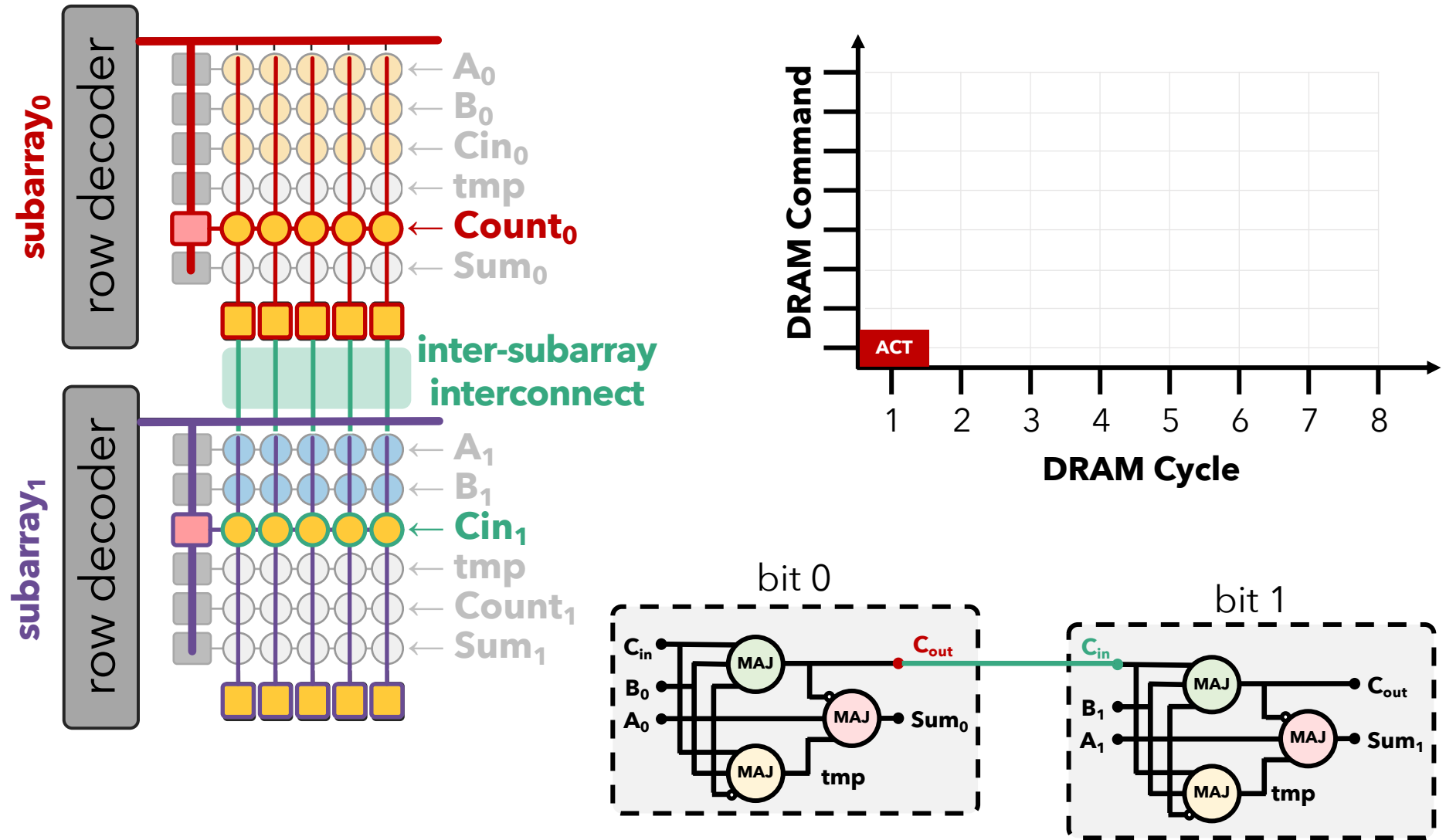
Opportunities for Bit-Serial PUD: Latency Reduction with Subarray Parallelism (III)



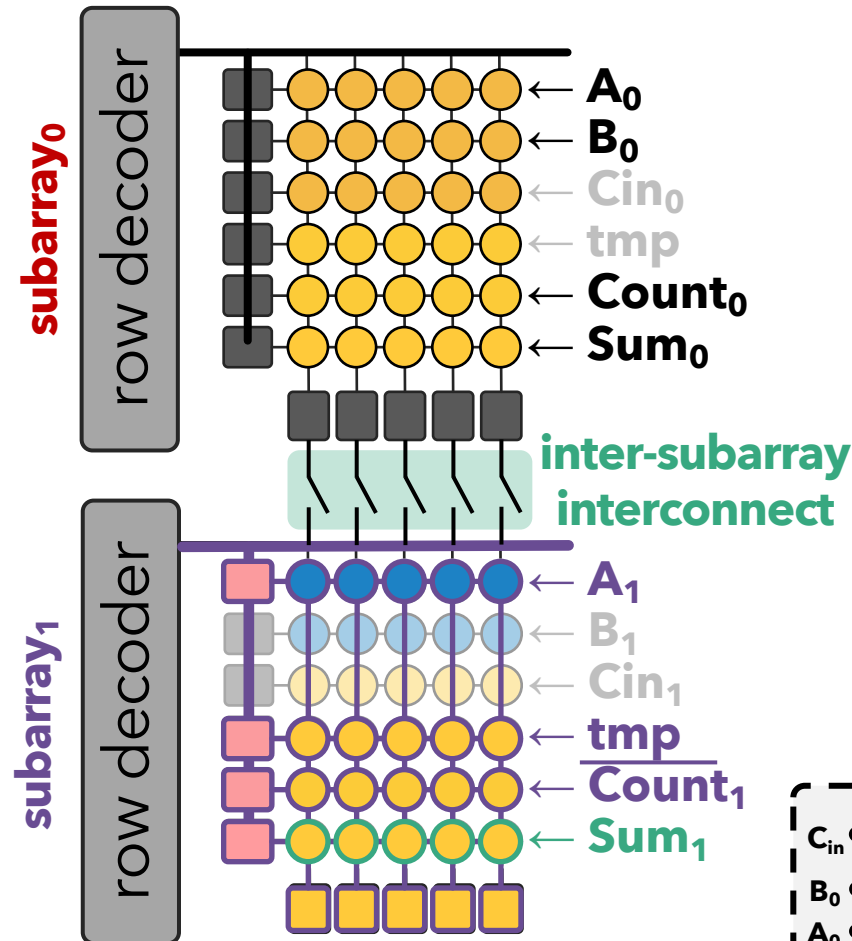
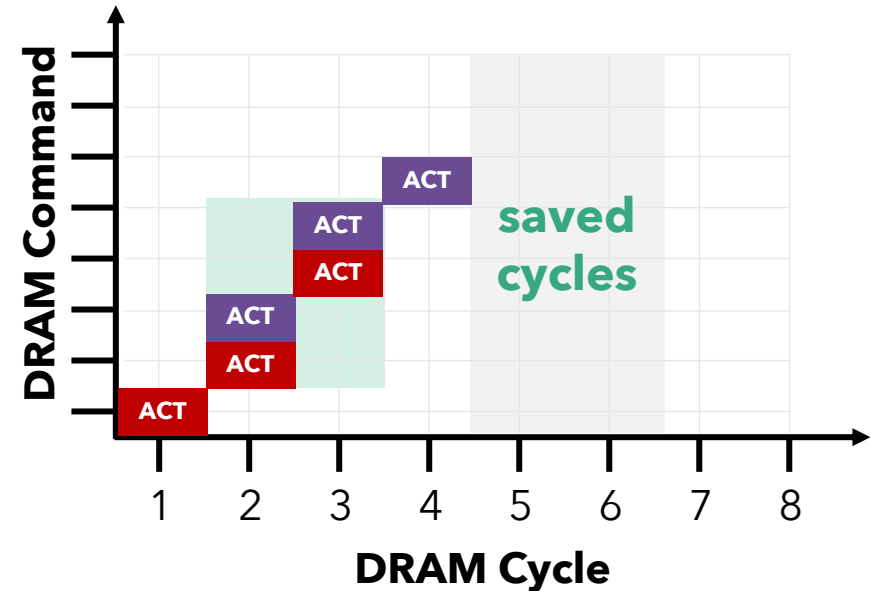
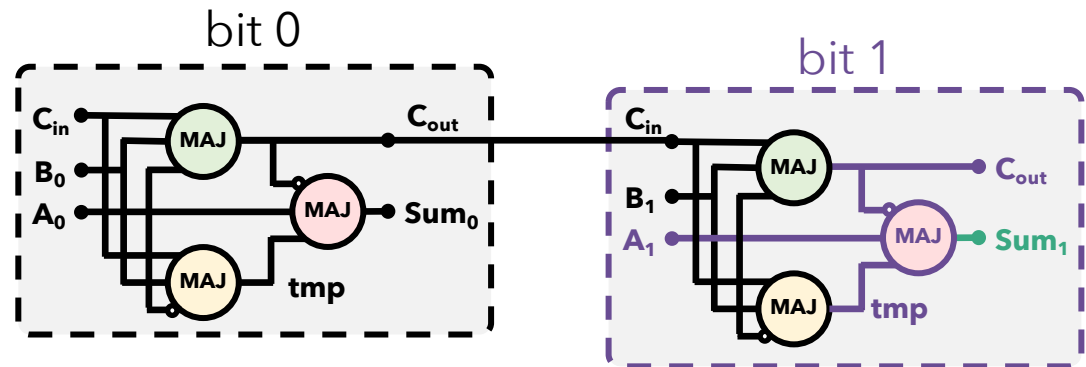
Opportunities for Bit-Serial PUD: Latency Reduction with Subarray Parallelism (III)



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

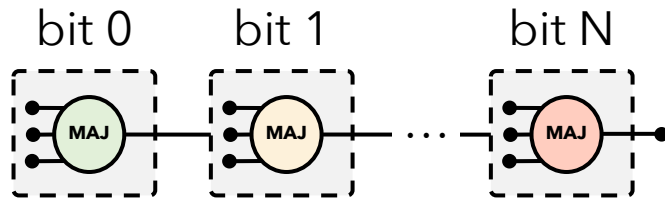
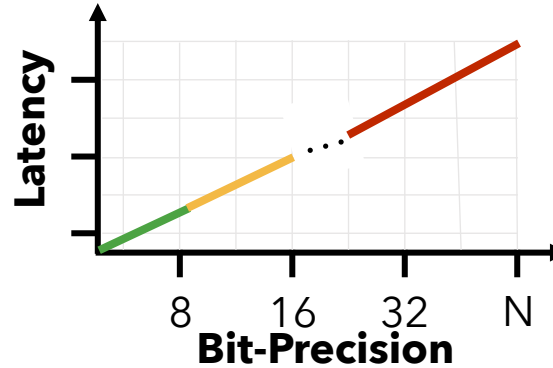
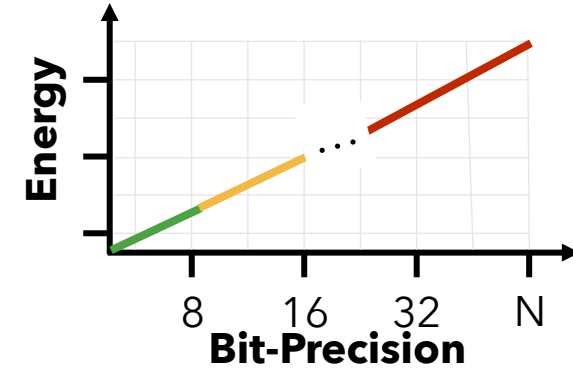
- failing to reduce the latency of a single PUD operation
- opportunity 2: DRAM parallelism for latency-oriented execution

3 Scalability Challenges for High-Precision Operations

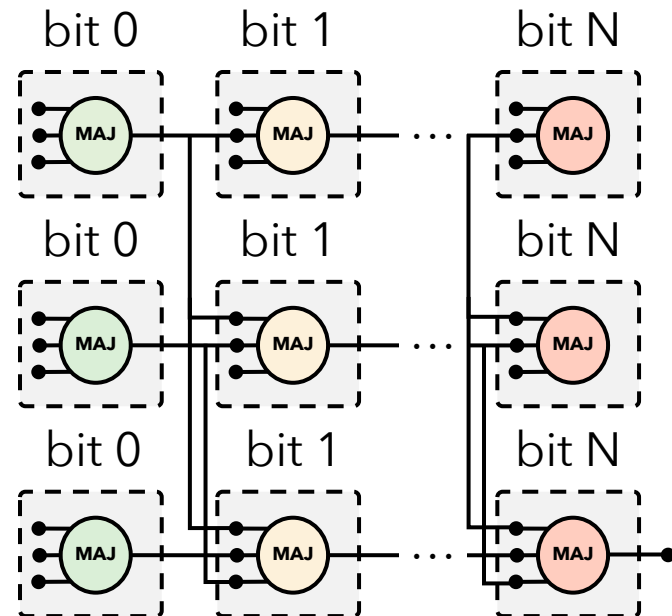
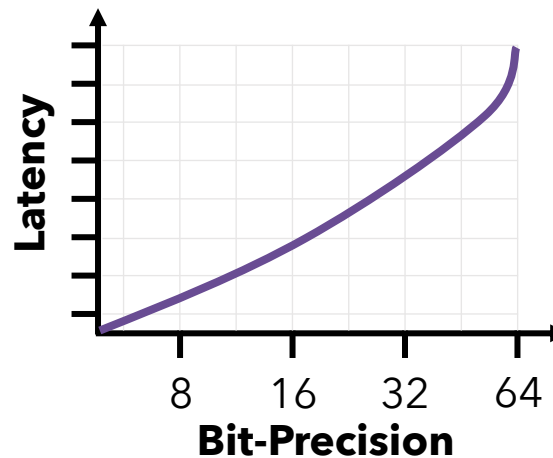
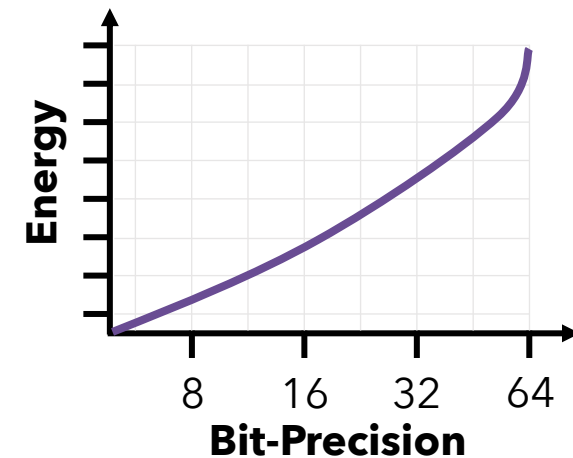
- due to the linear/quadratically scaling nature of bit-serial algorithms
- opportunity 3: alternative data representation for high-precision computation

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)



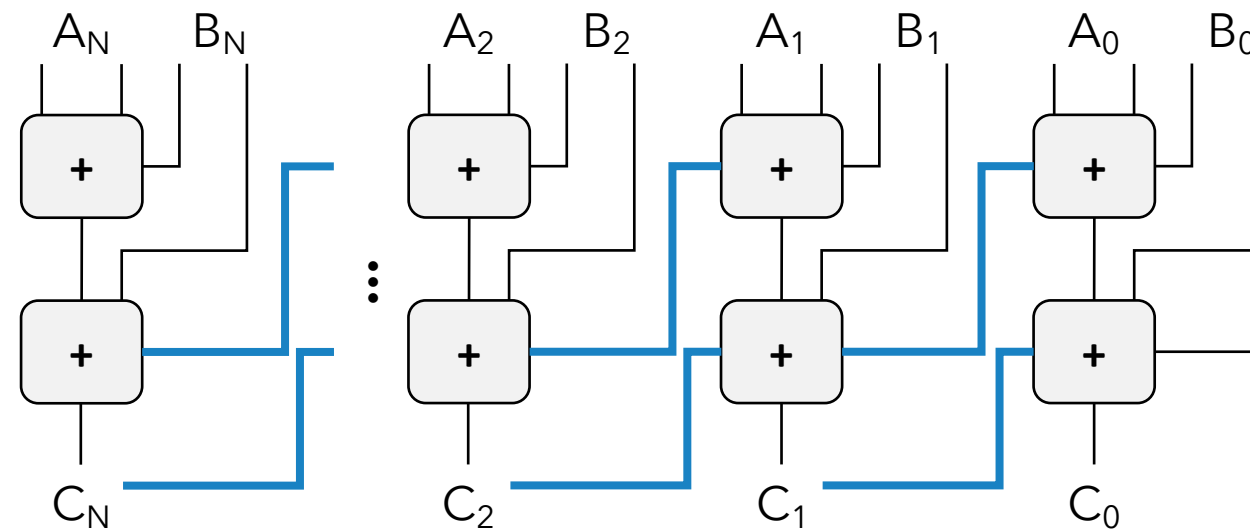
Opportunities for Bit-Serial PUD: Alternative Data Representation for High-Precision

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 $\in \{-1, 0, 1\}$

$$\langle 0, \mathbf{1}, 0, \mathbf{-1} \rangle = 0 \times 2^3 + \mathbf{1} \times \mathbf{2^2} + 0 \times 2^1 + \mathbf{(-1)} \times \mathbf{2^0} = \mathbf{4} - \mathbf{1} = \mathbf{3}$$



RBR-Based Arithmetic

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

Outline

1 Introduction

2 Background

3 PUD's Limitations & Key Ideas

4 *Proteus*

5 Evaluation

6 Conclusion

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 μ Program Library

2 Dynamic Bit-Precision Engine

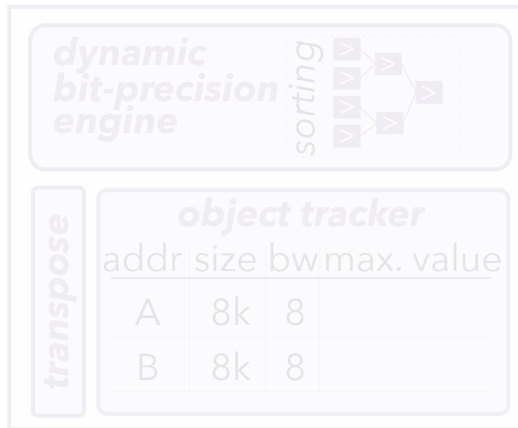
3 μ Program Select Unit

Proteus: Design Overview (II)

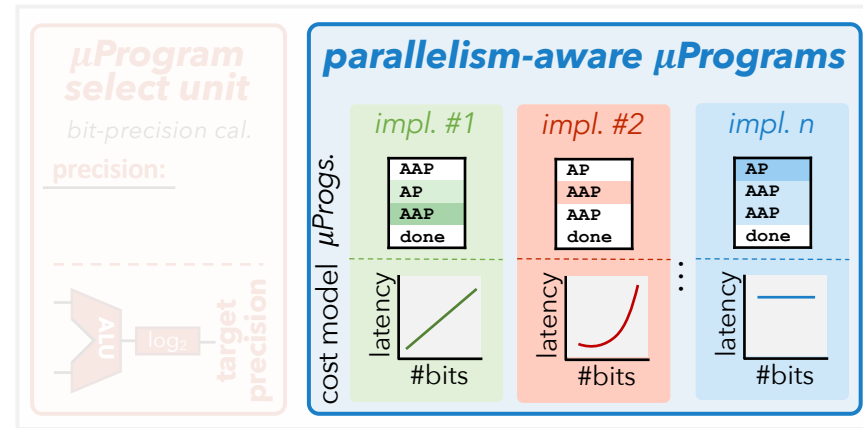
last-level cache

A	0	7	0	1
B	1	9	8	3
C				
F				
Z				
H				
R				

data transposition unit



Proteus control unit



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

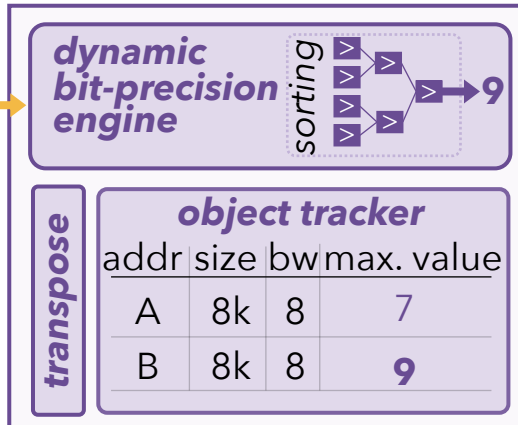
3 μProgram Select Unit

Proteus: Design Overview (III)

last-level cache

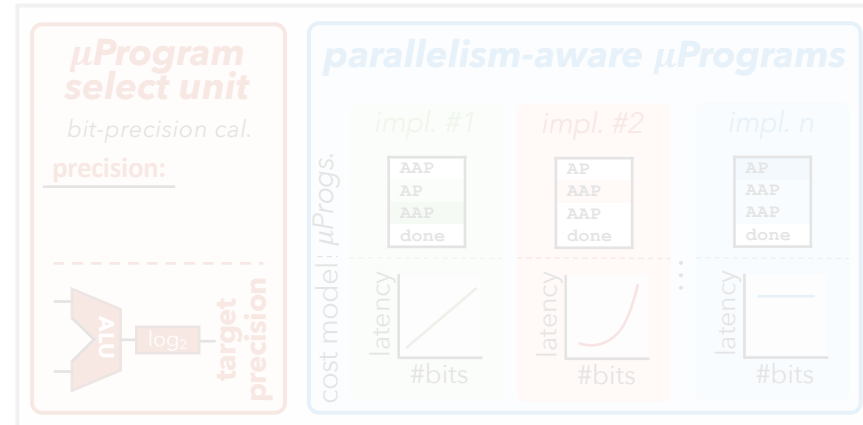
A	0	7	0	1
B	1	9	8	3
C				
F				
Z				
R				
H				

data transposition unit



evicted cache line

Proteus control unit



Proteus is composed of three main components:

1 Parallelism-Aware μProgram Library

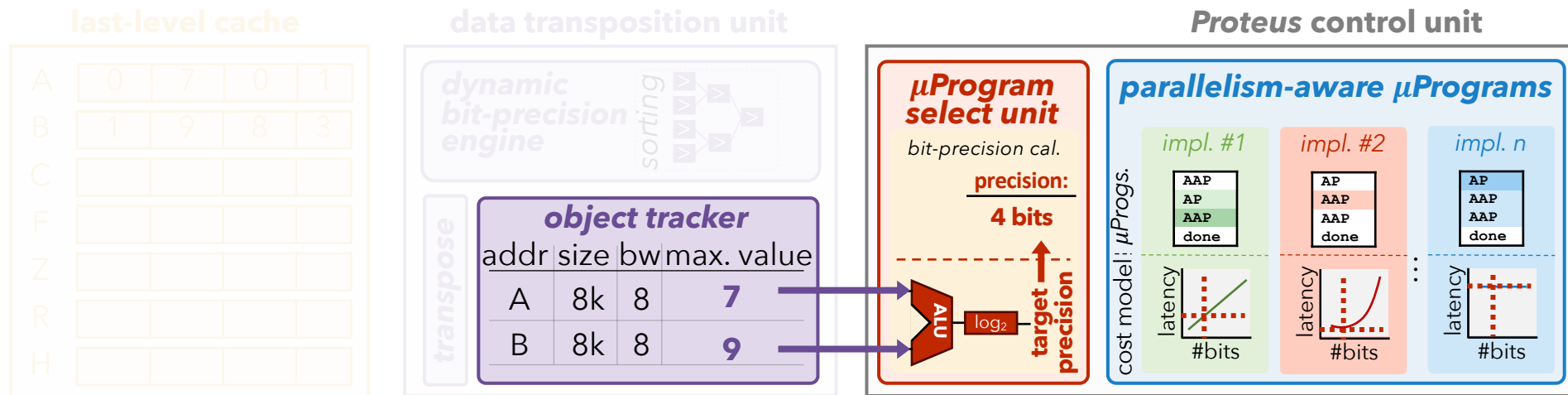
hand-optimized implementations of different PUD operations with different performance vs. bit-precision trade-offs

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 μ Program Library

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

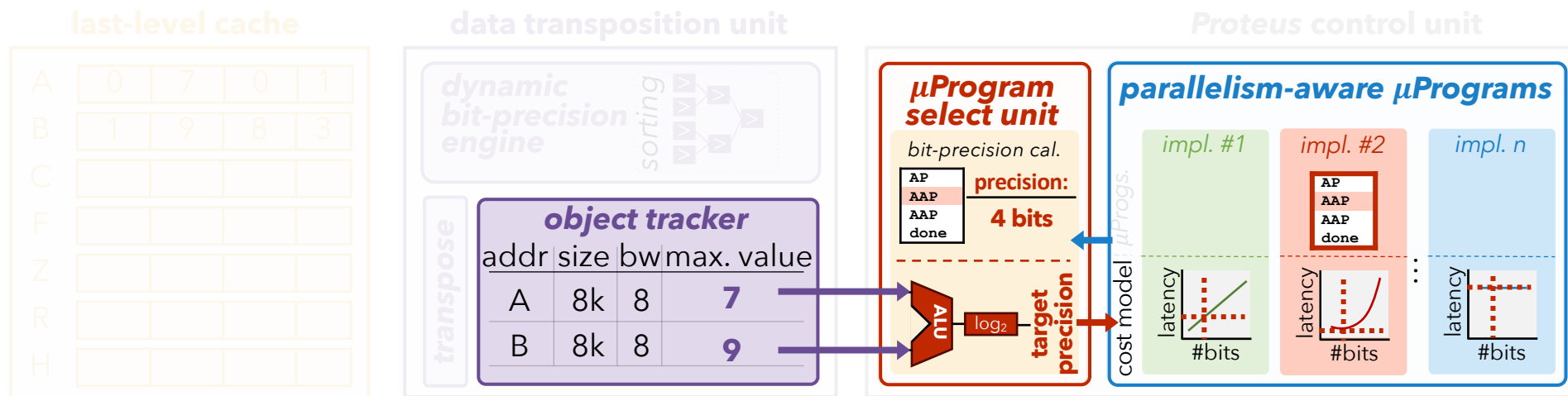
2 Dynamic Bit-Precision Engine

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

3 μ Program 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

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>

- System integration

- Proteus for floating-point operations

Outline

1 Introduction

2 Background

3 PUD's Limitations & Key Ideas

4 *Proteus*

5 **Evaluation**

6 Conclusion

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>

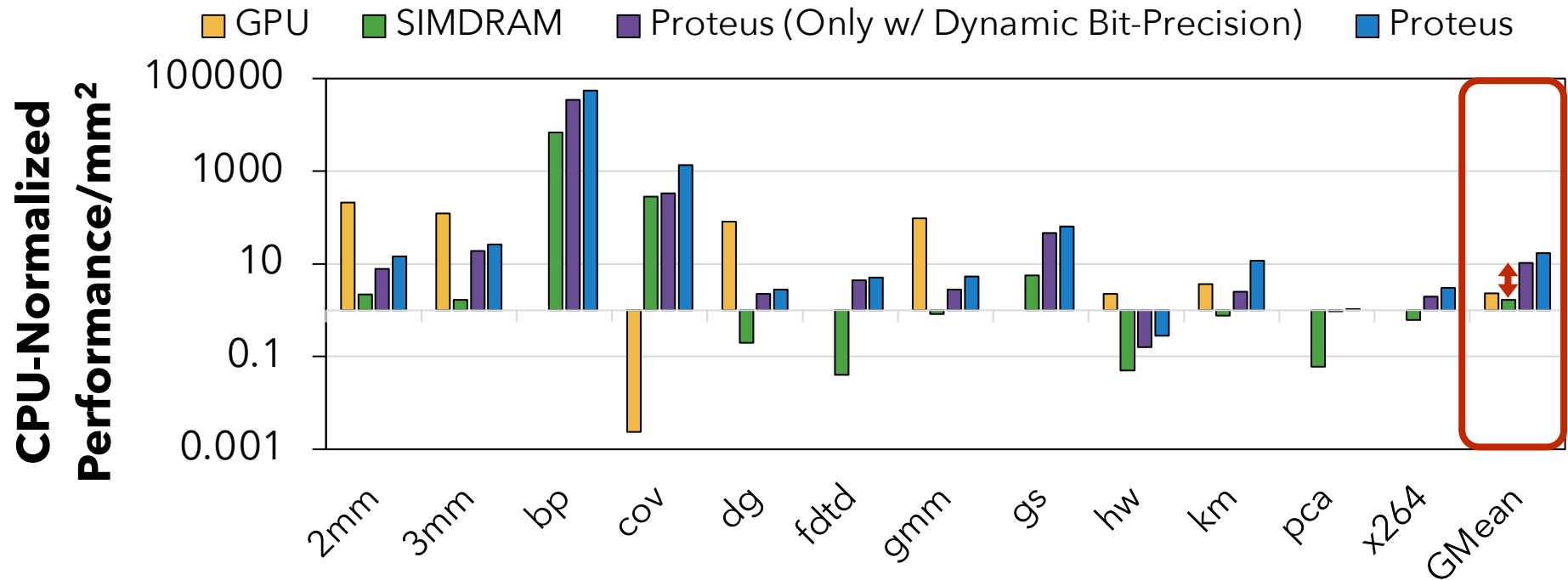
- **Workloads**

- 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)

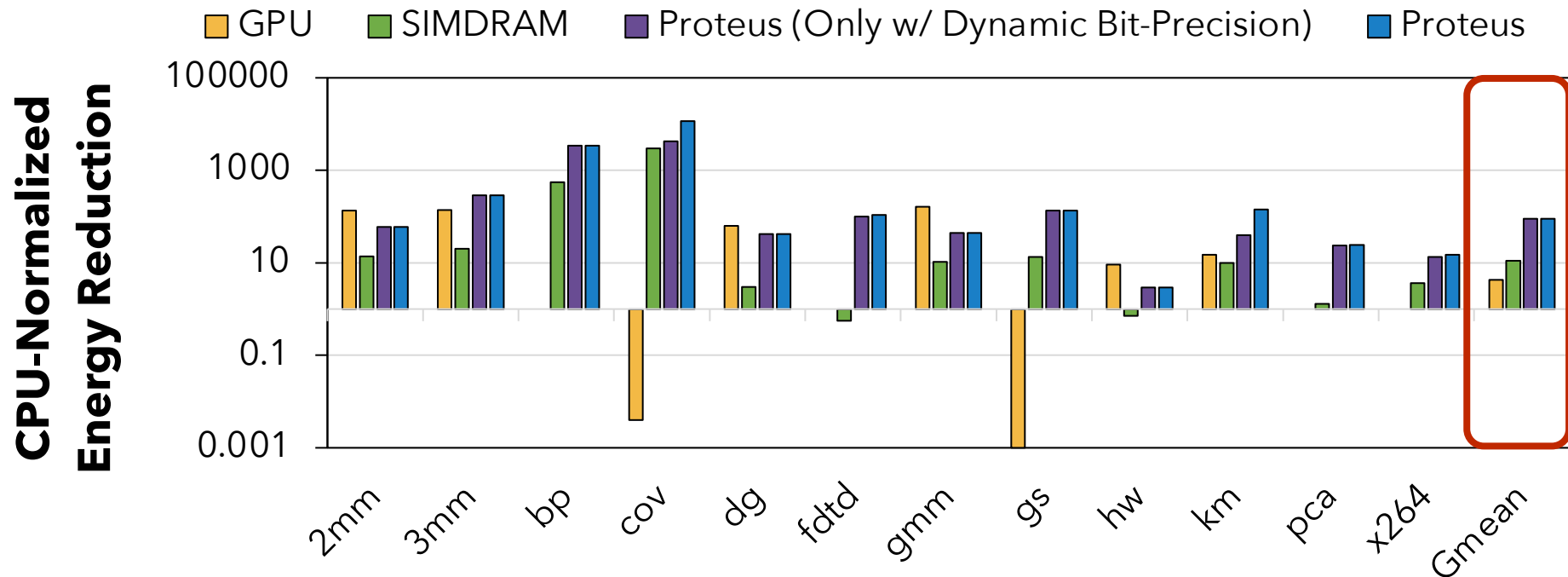
Evaluation: Performance Analysis



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

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>

Outline

1 Introduction

2 Background

3 PUD's Limitations & Key Ideas

4 *Proteus*

5 Evaluation

6 Conclusion

Conclusion

**We introduce *Proteus*,
a data-aware hardware runtime PUD framework that**

that dynamically adjusts the bit-precision and
chooses and uses the most appropriate data representation and
arithmetic algorithm implementation for a given PUD operation

Our extensive evaluation shows that *Proteus* achieves

17x (90.3x), 7.3x (21x), and 10.2x (8.1x) the performance per mm²
(energy reduction) of a CPU, GPU, and state-of-the-art PUD system,
small area cost to a DRAM chip (**1.6%**) and CPU die (**0.03%**)

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

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

Giray Yağlıkçı

Melina Soysal

Mohammad Sadr

Joaquin O. Bueno

Saugata Ghose

Juan Gómez-Luna

Onur Mutlu

SAFARI
ETH zürich



Proteus

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

Backup Slides

Geraldo F. Oliveira

Mayank Kabra

Yuxin Guo

Kangqi Chen

Giray Yağlıkçı

Melina Soysal

Mohammad Sadr

Joaquin O. Bueno

Saugata Ghose

Juan Gómez-Luna

Onur Mutlu

Cost Model Logic

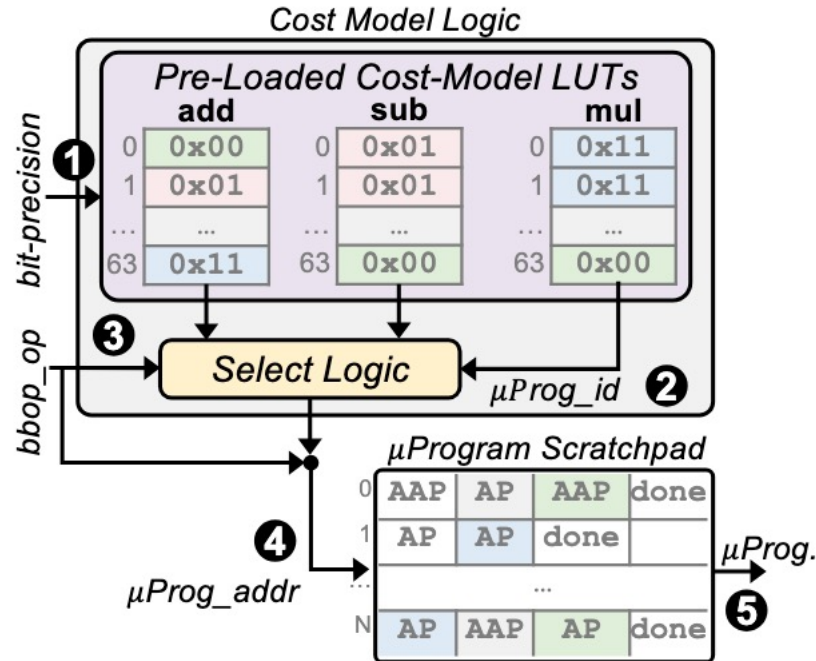


Figure 8: Proteus' Cost Model Logic.

We estimate, using CACTI [248], that the access latency and energy per access of the 64 B SRAM array (used in our Cost Model Logic) is of **0.07 ns** (i.e., less than 1 CPU cycle) and **0.000 04 nJ**

Pareto Analysis: Addition

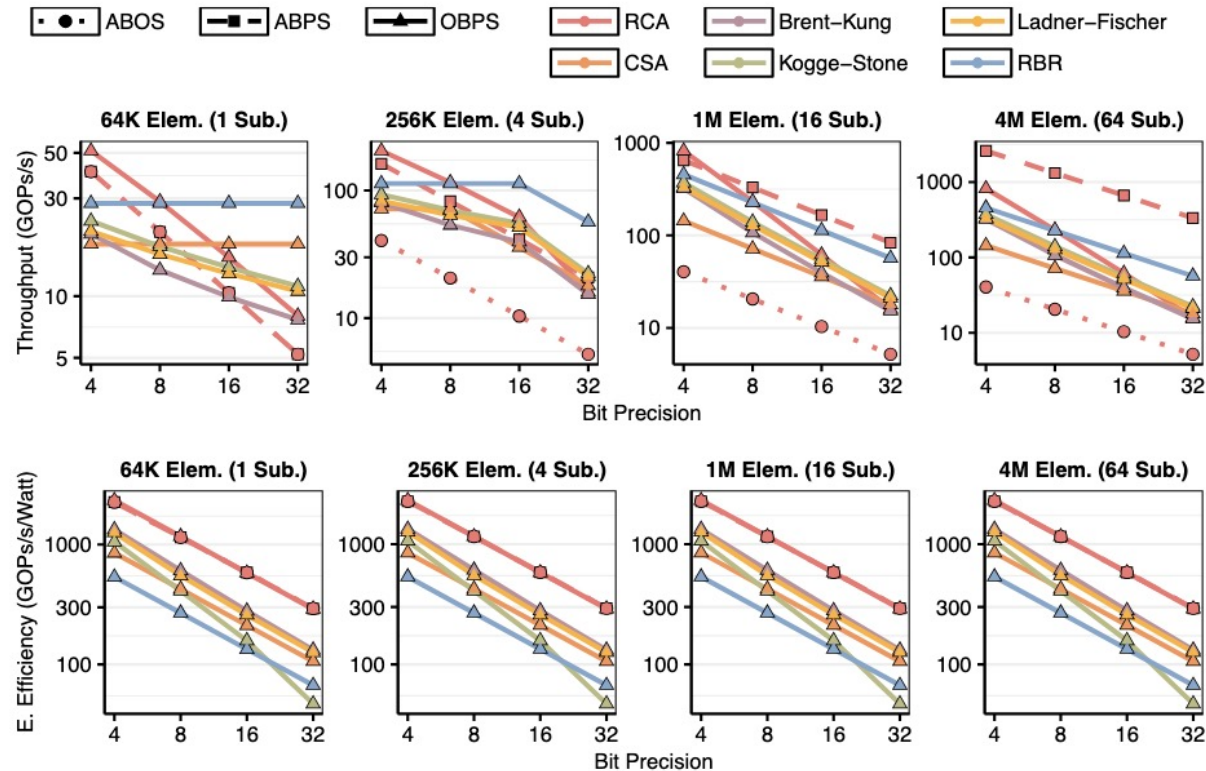


Figure 9: Pareto analysis for throughput (top) and energy efficiency (bottom) for PUD addition operations. Dotted lines represent ABOS; dashed lines represent ABPS; straight lines represent OBPS data mapping.

Pareto Analysis: Multiplication

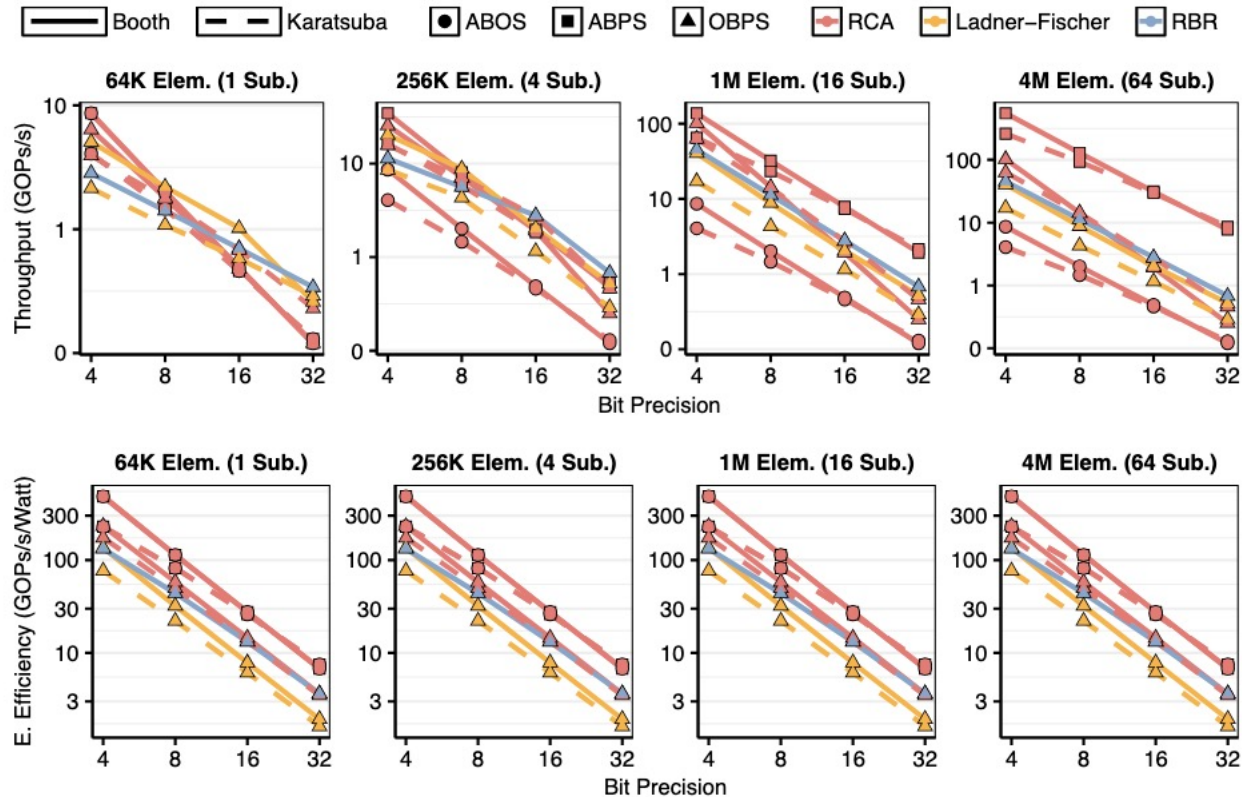


Figure 10: Pareto analysis for throughput (top) and energy efficiency (bottom) for multiplication. Straight lines represent the Booth's multiplication method [249]; dashed lines represent the Karatsuba [250] multiplication method.

Calculating Bit-Precision

```
bbop_add(tmp, A, B, 8k, 8, 1); // tmp  $\leftarrow$  A + B  
bbop_mul(D, tmp, C, 8k, 8, 1); // D  $\leftarrow$  tmp  $\times$  C
```

Assume that the maximum value of A, B, and C are 3, 6, and 2, respectively. In this case, the *μ Program Select Unit* (i) computes the bit-precision for the addition operation as $\lceil \log_2(3 + 6) \rceil = 4 \text{ bits}$; (ii) updates the *Object Tracker* entry of tmp with the maximum value of the addition operation (i.e., 9); (iii) computes the bit-precision for the multiplication operation as $\lceil \log_2(9 \times 2) \rceil = 5 \text{ bits}$ using an n -bit scalar ALU; (iv) updates the *Object Tracker* entry of D with the maximum value of the multiplication (i.e., 18).

Data Format Conversion

Table 1: Conversion from two's complement to RBR.

Input X → two's complement →	2 0 0 1 0	-1 1 1 1 1	-7 1 0 0 1
Steps to convert two's to RBR ↓	Output ↓		
Extract most-significant bit (MSB)	0	1	1
buffer1: broadcast MSB to all subarrays	0 0 0 0	1 1 1 1	1 1 1 1
buffer2: NOT(buffer1)	1 1 1 1	0 0 0 0	0 0 0 0
X + 1	1 1 1 0	0 0 0 1	0 1 1 1
$X- = \text{buffer1} \& (X + 1)$	0 0 0 0	0 0 0 1	0 1 1 1
$X+ = \text{buffer2} \& (X)$	0 0 1 0	0 0 0 0	0 0 0 0

Proteus for Floating-Point

First, the *Dynamic Bit-Precision Engine* identifies the sign, exponent, and mantissa bits in floating-point numbers stored in the evicted cache lines of PUD memory objects. Second, the *Dynamic Bit-Precision Engine* updates the *Object Tracker* with the maximum exponent and the mantissa values for the PUD memory object, in case the identified maximum values are smaller than the values stored in the current evicted cache line. This process is analogous to the execution flow described in §4.2 for integer operands, with the addition of the fields for the maximum exponent and maximum mantissa values in the *Object Tracker*. Third, *Proteus* performs the target in-memory floating-point computation by issuing [113] (i) bit-serial subtraction (addition) PUD operations to calculate the resulting exponents, followed by (ii) bit-serial addition (multiplication) PUD operations to calculate the resulting mantissas for a vector addition (multiplication) instruction [113]. Note that the bit-serial PUD operations involved in (i)–(ii) are vector-to-vector PUD operations. As such, *Proteus* can leverage the maximum exponent and mantissa values stored in the *Object Tracker* to set the required bit-precision for such bit-serial operations, following the same approach described in §5.4.

Evaluation System Configuration

Table 2: Evaluated system configurations.

Intel Comet Lake CPU [268] (Real System)	x86 [238], 16 cores, 8-wide, out-of-order, 3.8 GHz; <i>L1 Data + Inst. Private Cache:</i> 256 kB, 8-way, 64 B line; <i>L2 Private Cache:</i> 2 MB, 4-way, 64 B line; <i>L3 Shared Cache:</i> 16 MB, 16-way, 64 B line; <i>Main Memory:</i> 64 GB DDR4-2133, 4 channels, 4 ranks
NVIDIA A100 GPU [261] (Real System)	7 nm technology node; 826 mm ² die area [261]; 6912 CUDA cores; 432 tensor cores, 108 streaming multiprocessors, 1.4 GHz base clock; <i>L2 Cache:</i> 40 MB L2 Cache; <i>Main Memory:</i> 40 GB HBM2 [192, 269]
SIMDRAM [143] & Proteus (Simulated)	gem5-based in-house simulator [242, 264]; x86 [238]; 1 out-of-order core @ 4 GHz (<i>only</i> for instruction offloading); <i>L1 Data + Inst. Cache:</i> 32 kB, 8-way, 64 B line; <i>L2 Cache:</i> 256 kB, 4-way, 64 B line; <i>Memory Controller:</i> 8 kB row size, FR-FCFS [270, 271] <i>Main Memory:</i> DDR5-5200 [272], 1 channel, 1 rank, 16 banks

Evaluated Workloads

Benchmark Suite	Application (Short Name)	Peak GPU Util. (%)	Total Mem. Footprint (GB)	Bit-Precision {min, max}	PUD Instrs. [†]
Phoenix [171]	pca (pca)	–	1.91	{8, 8}	D, S, M, R
Polybench [169]	2mm (2mm)	98	4.77	{13, 25}	M, R
	3mm (3mm)	100	26.7	{12, 12}	M, R
	covariance (cov)	100	7.63	{23, 23}	D, S, R
	doitgen (dg)	92	33.08	{10, 11}	M, C, R
	fdtd-apml (fdtd)	–	36.01	{11, 13}	D, M, S, A
	gemm (gmm)	98	22.89	{12, 24}	M, R
	gramschmidt (gs)	66	22.89	{12, 13}	M, D, R
Rodinia [172]	backprop (bp)	–	22.50	{13, 13}	M, R
	heartwall (hw)	48	0.03	{17, 17}	M, R
	kmeans (km)	36	1.23	{17, 17}	S, M, R
SPEC 2017 [170]	525.x264_r (x264)	–	0.15	{1, 8}	A, R

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

Is Static Profiling Good Enough?

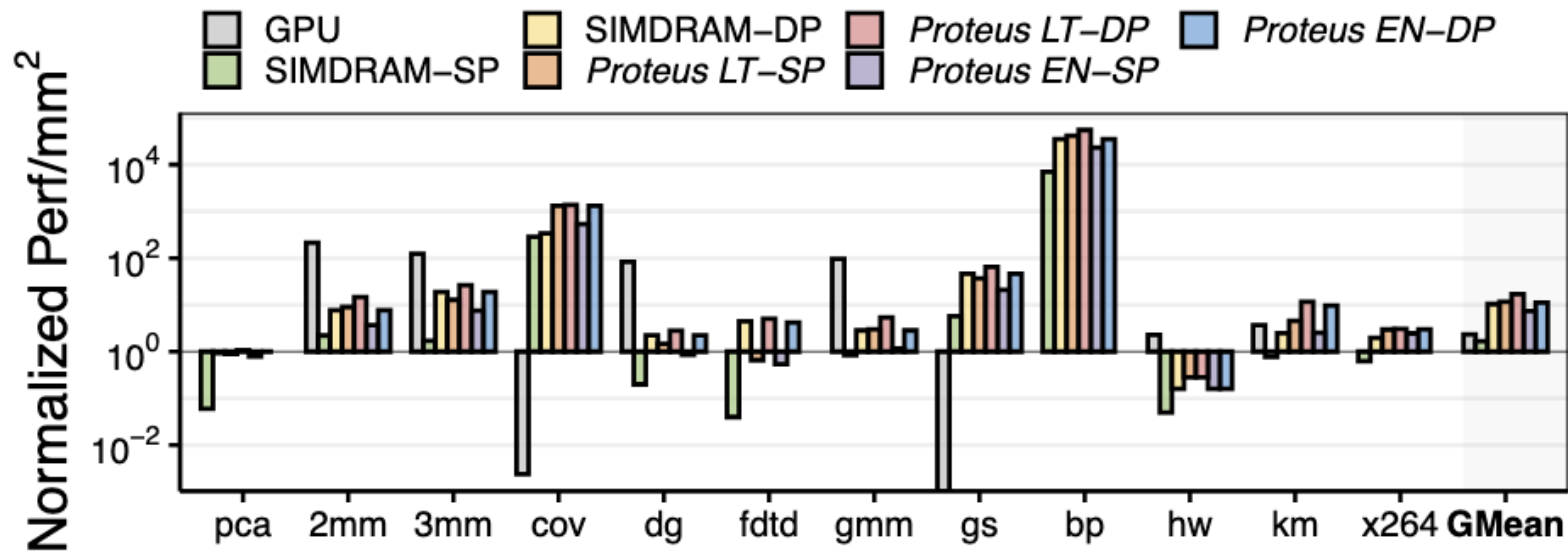
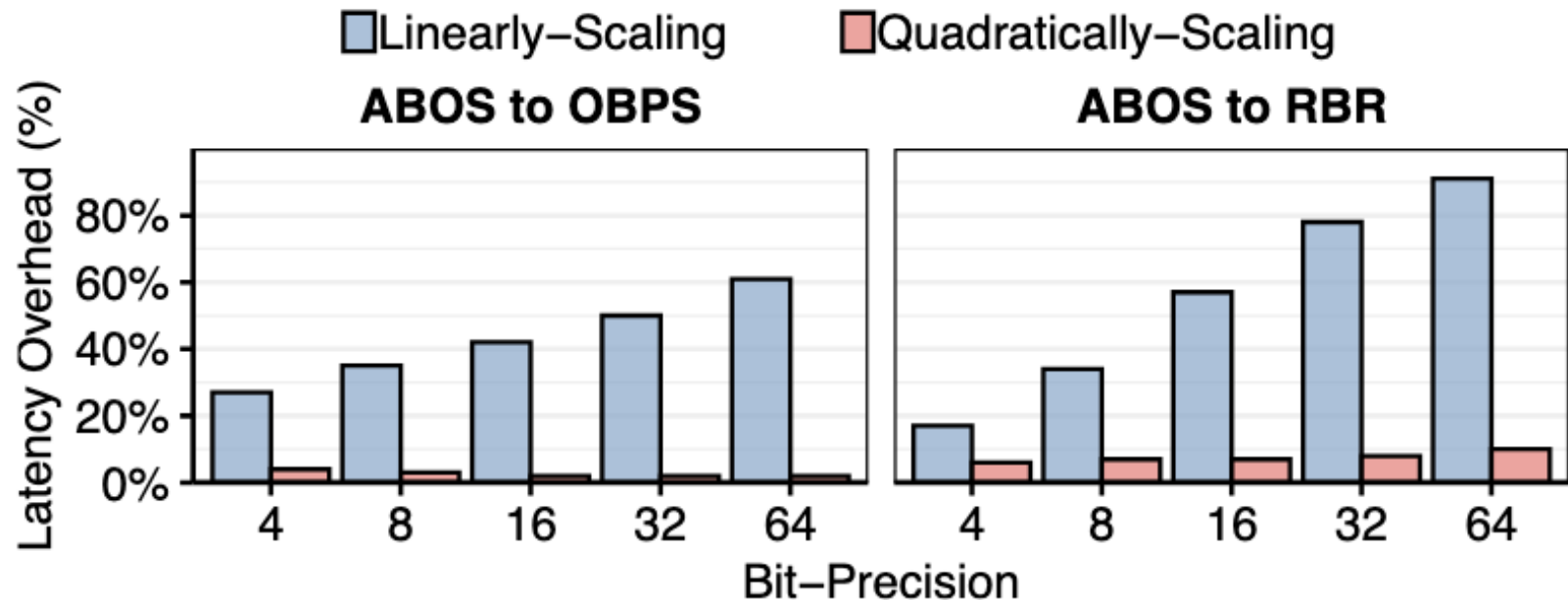


Figure 11: CPU-normalized performance per mm² for twelve real-world applications. Phoenix [171] and SPEC2017 [170] do not provide GPU implementations of pca and x264.

Proteus' Dynamic Bit-Precision Engine further **increases performance by 46%**, over Proteus with static bit-precision.

This happens because for statically profiled bit-precision, we must round the bit-precision up to the nearest power-of-two.

Data Conversion Overhead



Proteus vs. Tensor Cores

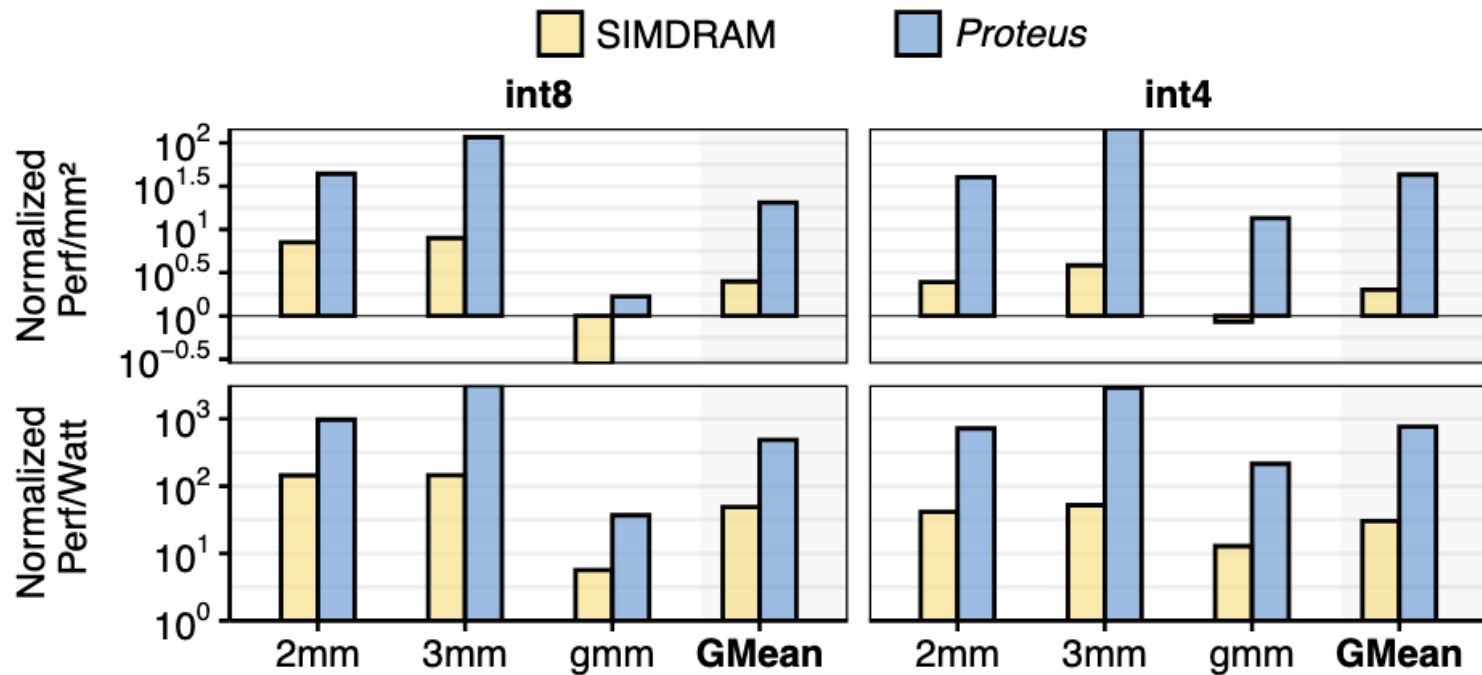


Figure 14: Performance per mm² (top) and performance per Watt (bottom) of GEMM-intensive real-world applications using int8 and int4, normalized to the same metric measured on 432 NVIDIA A100 tensor cores.

PUD and RowHammer

PuDHammer: Experimental Analysis of Read Disturbance Effects of Processing-using-DRAM in Real DRAM Chips

İsmail Emir Yüksel, Akash Sood, Ataberk Olgun, Oğuzhan Canpolat, Haocong Luo,
F. Nisa Bostancı, Mohammad Sadrosadati, A. Giray Yağlıkçı, Onur Mutlu

ETH Zürich
Zürich, Switzerland