2025 International Conference on Supercomputing (ICS)

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 UNIVERSITY OF CAMBRIDGE UNIVERSIDAD Đ CÓRDOBA UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN NVIDIA

Executive Summary

Problem: Processing-Using-DRAM (PUD) suffers from three limitations caused 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, and based on that,
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 mm^2 (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

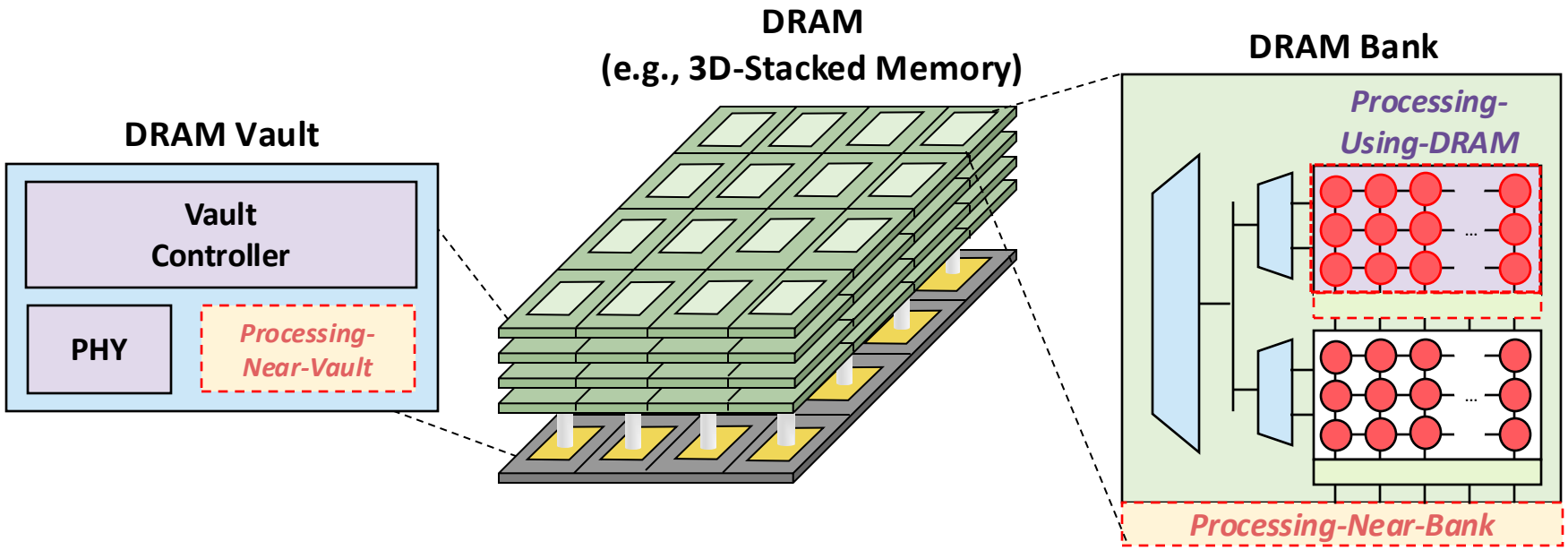
Outline

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



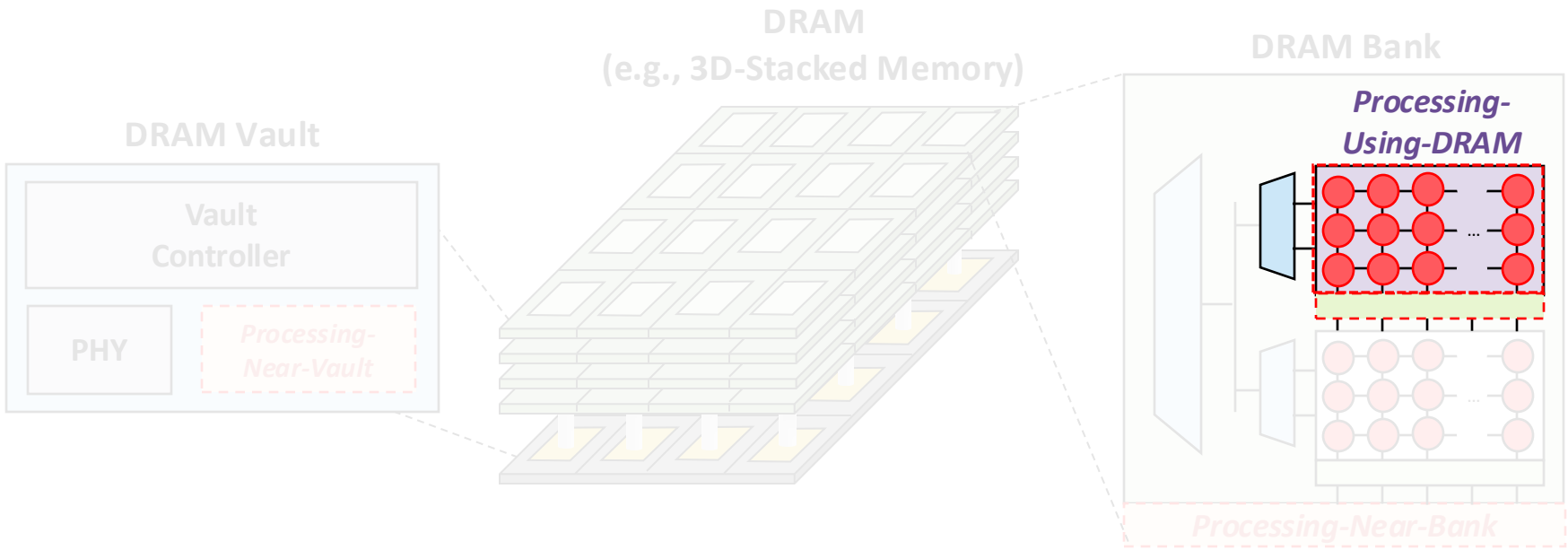
Outline

Processing-in-Memory:
Overview

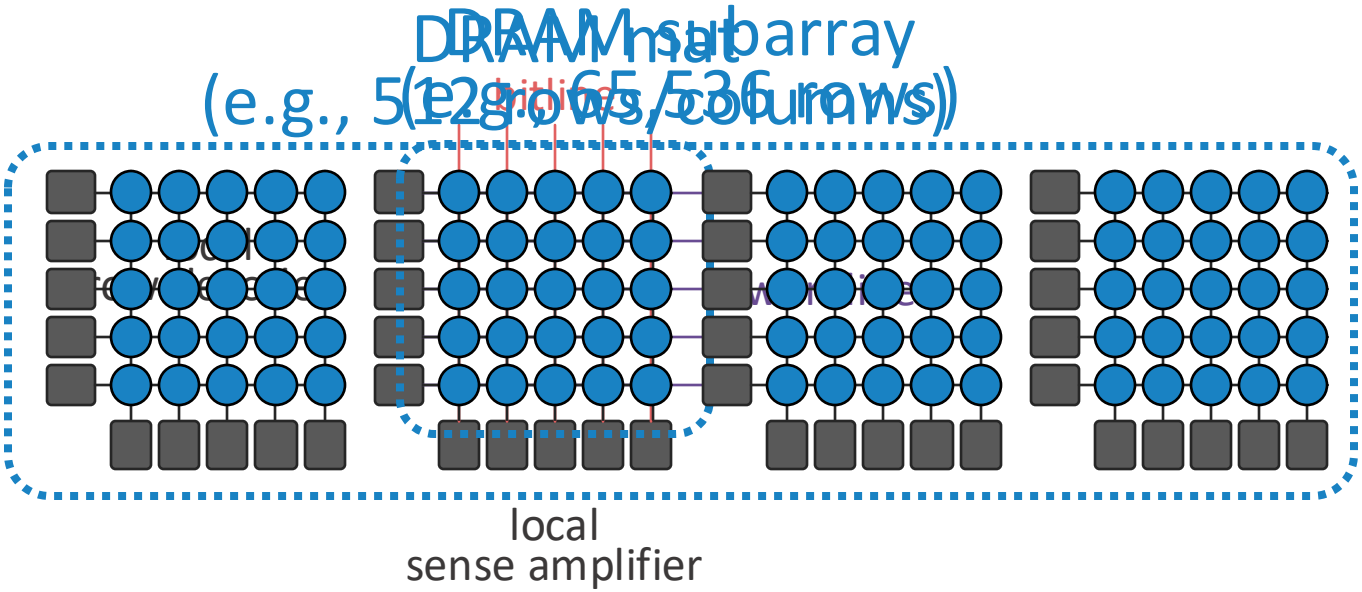
Two main approaches for Processing-in-Memory:

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

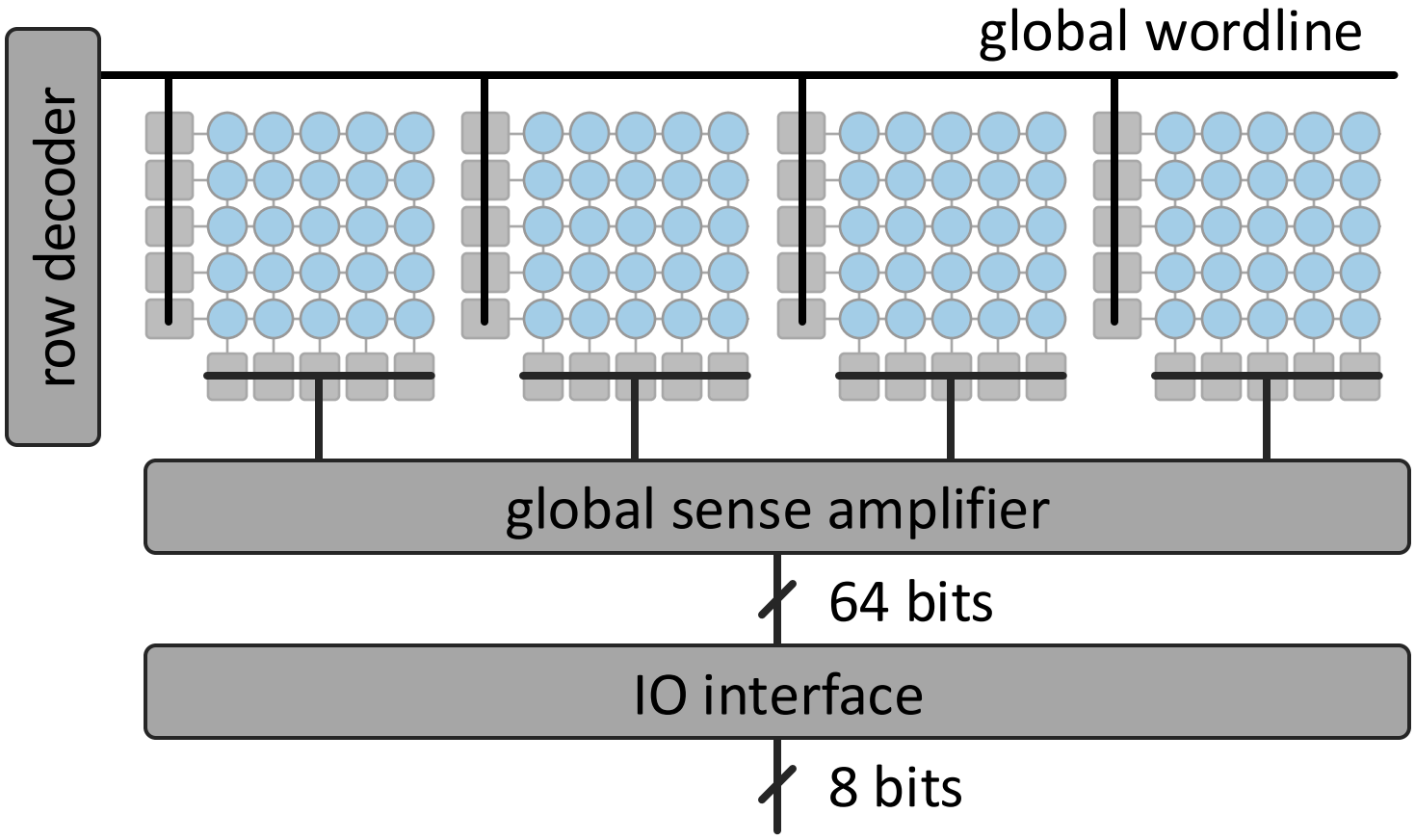
2. **Processing-Using-Memory**: uses the operational principles of memory cells to perform computation



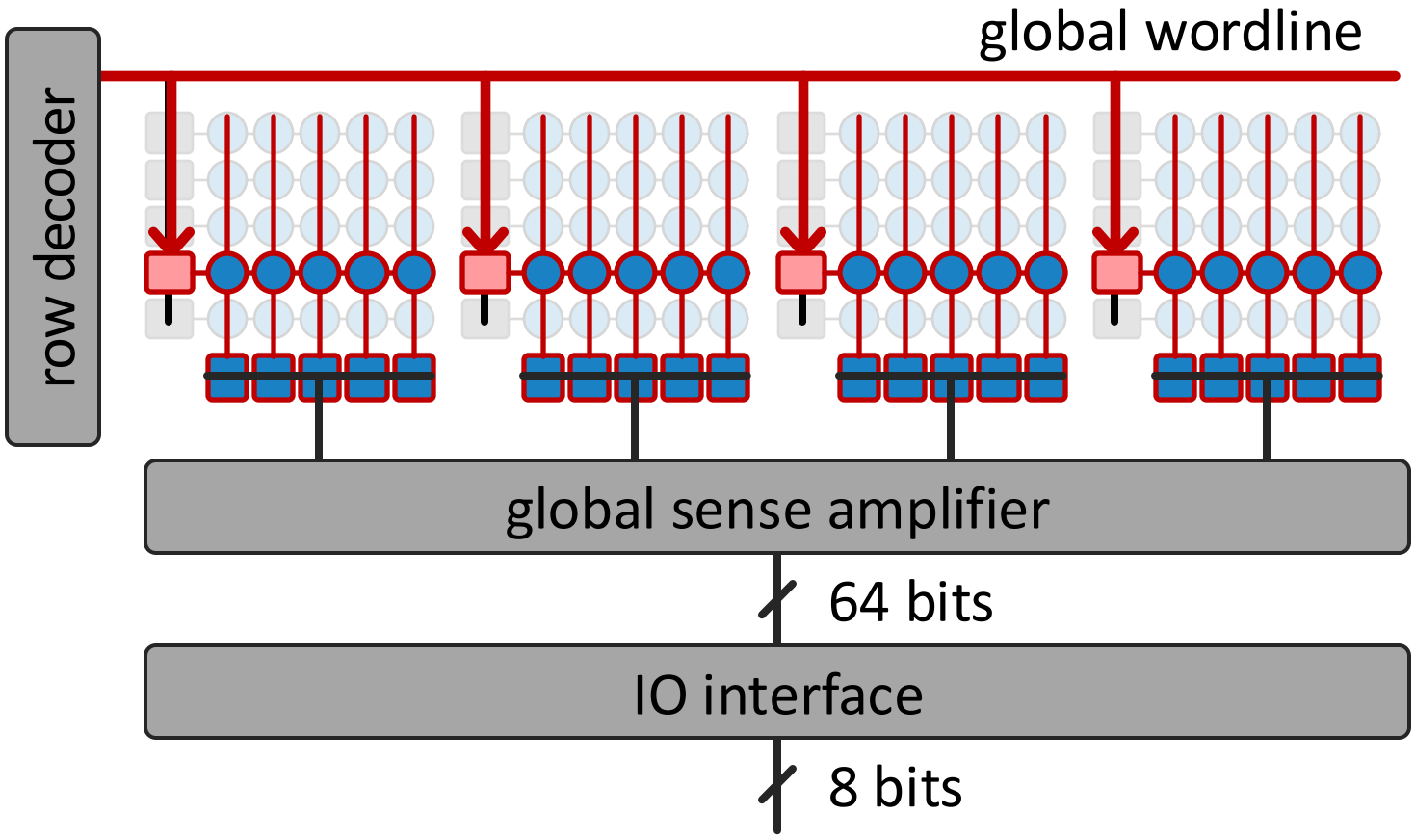
Background:
DRAM Hierarchical Organization



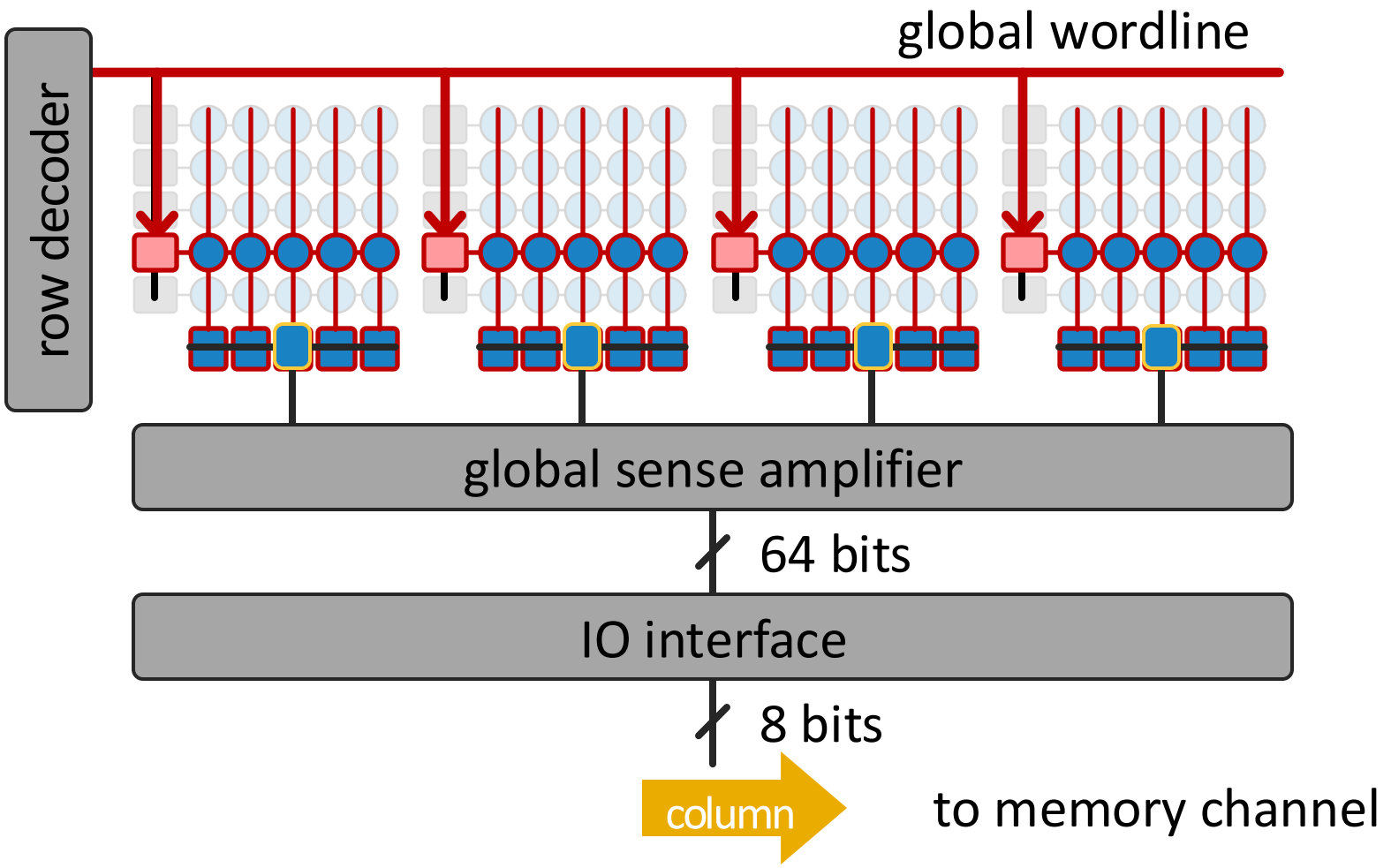
Background:
DRAM Hierarchical Organization



Background:
DRAM Operation – Row Access (ACTIVATE)

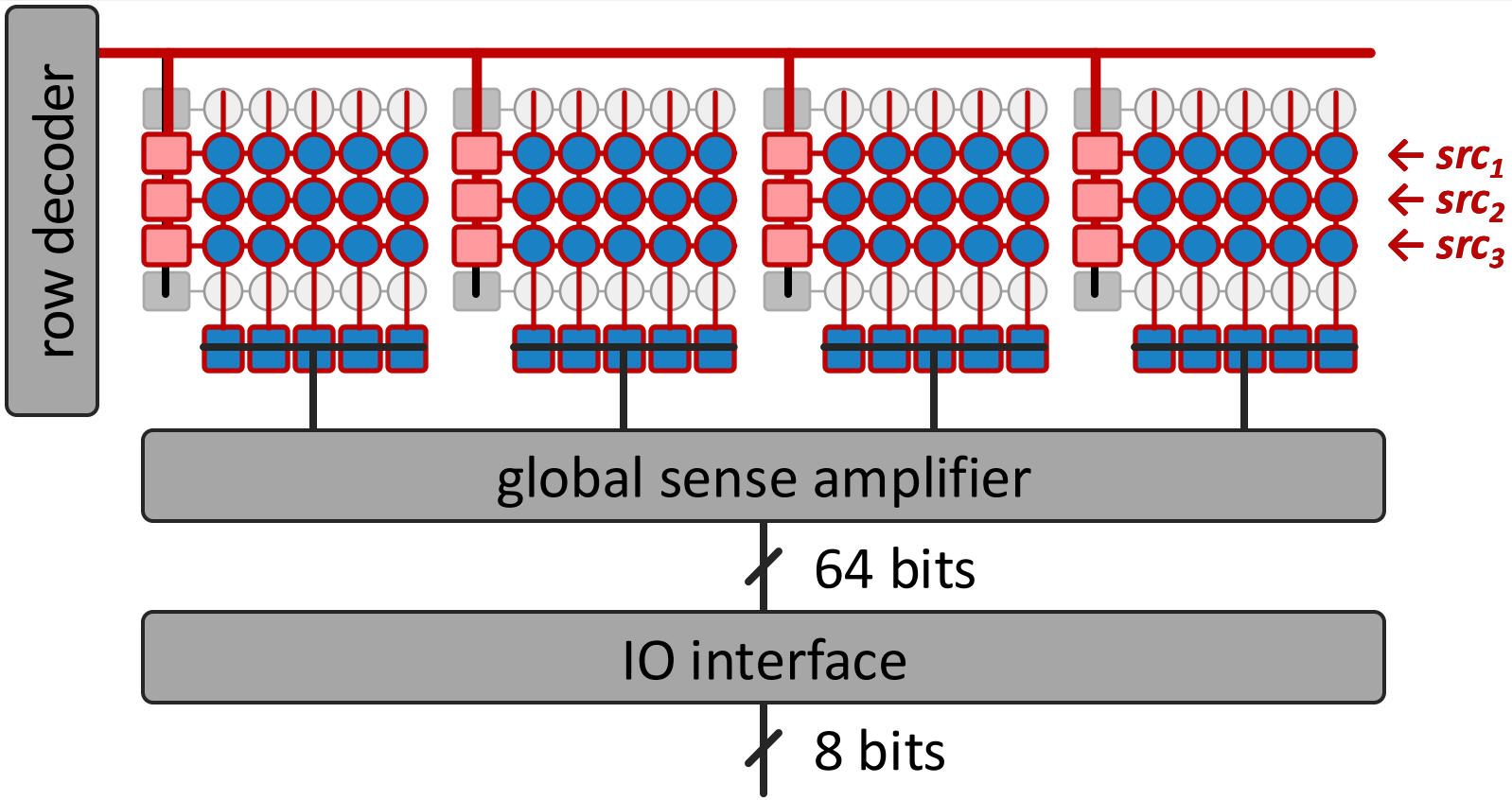


Background:
DRAM Operation – Column Access (READ)



Background:
In-DRAM Majority Operations

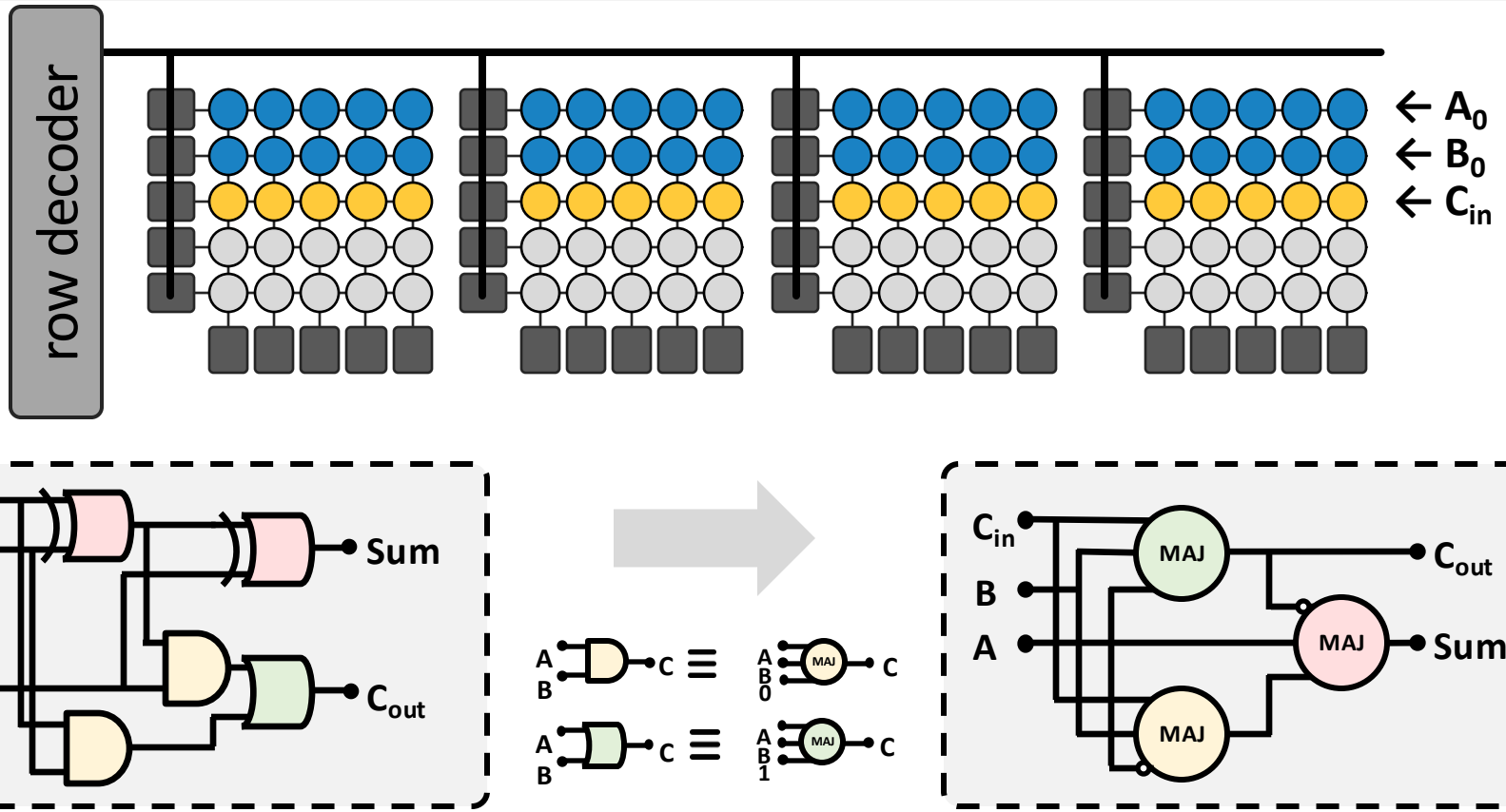
In-DRAM majority is performed by
simultaneously activating three DRAM rows



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

Background:
Bulk Bitwise Arithmetic Operations

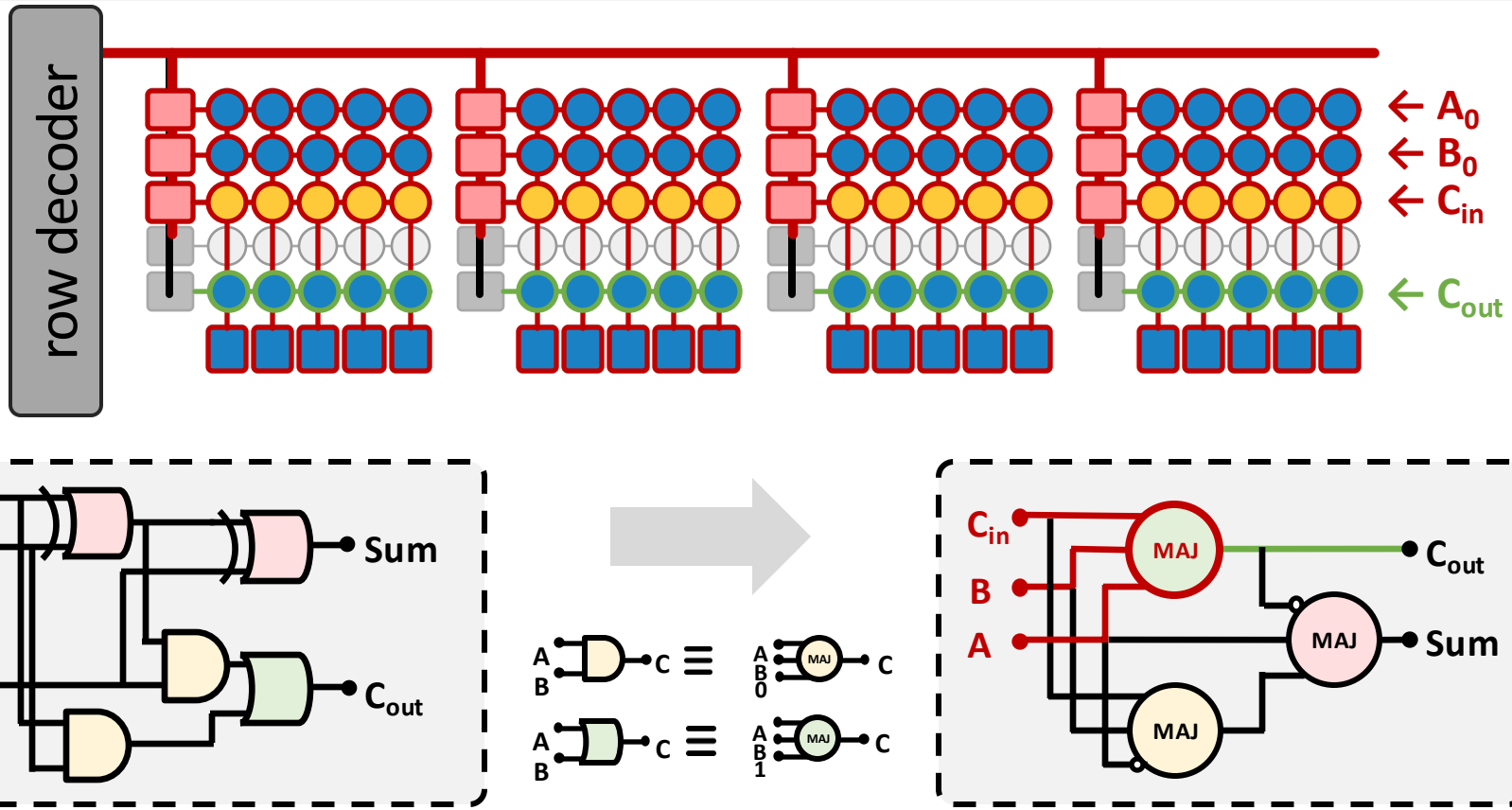
Bulk bitwise arithmetic can be performed by
orchestrating in-DRAM row copy and majority operations



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

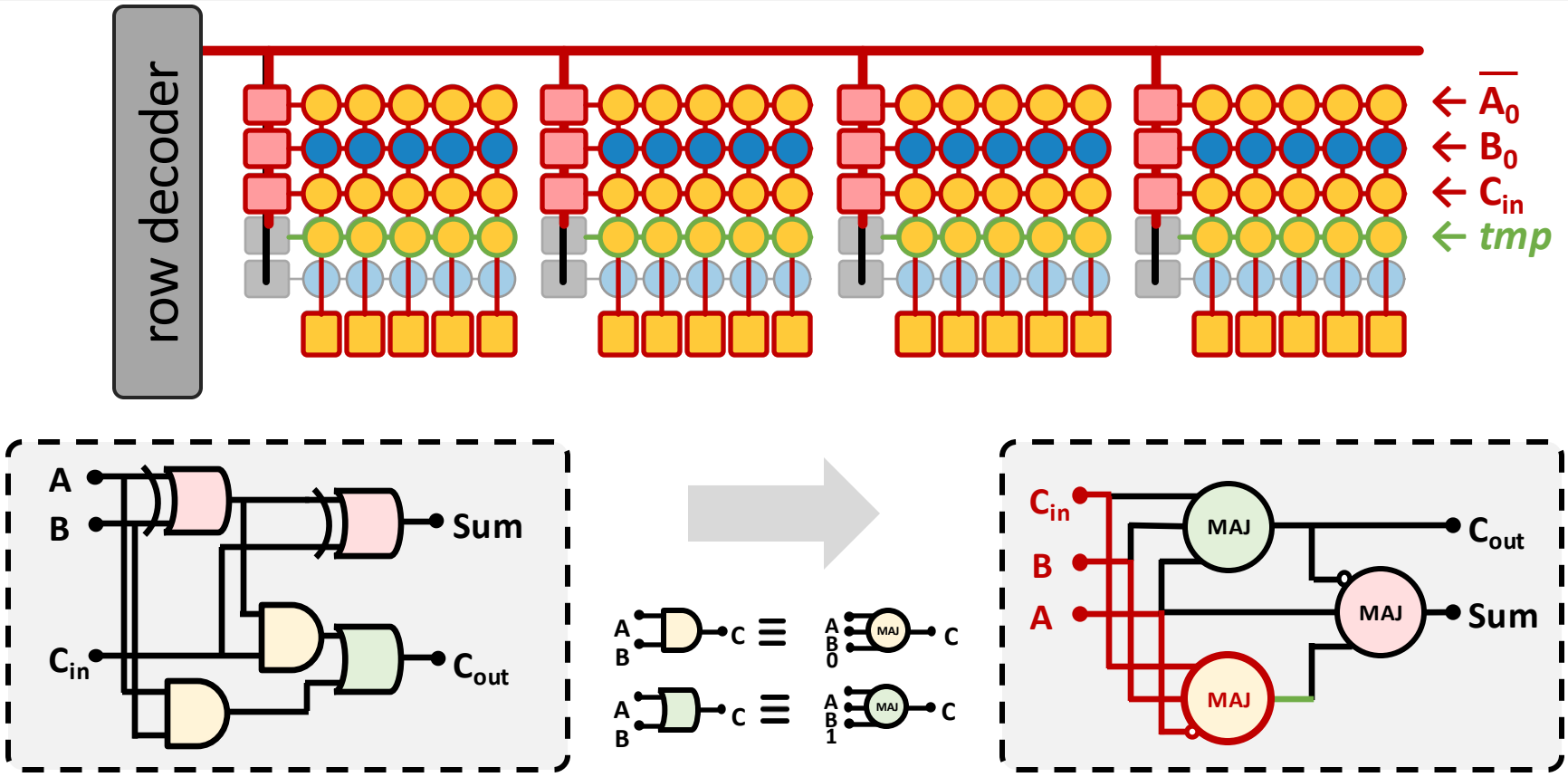
Bulk bitwise arithmetic can be performed by
orchestrating in-DRAM row copy and majority operations



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

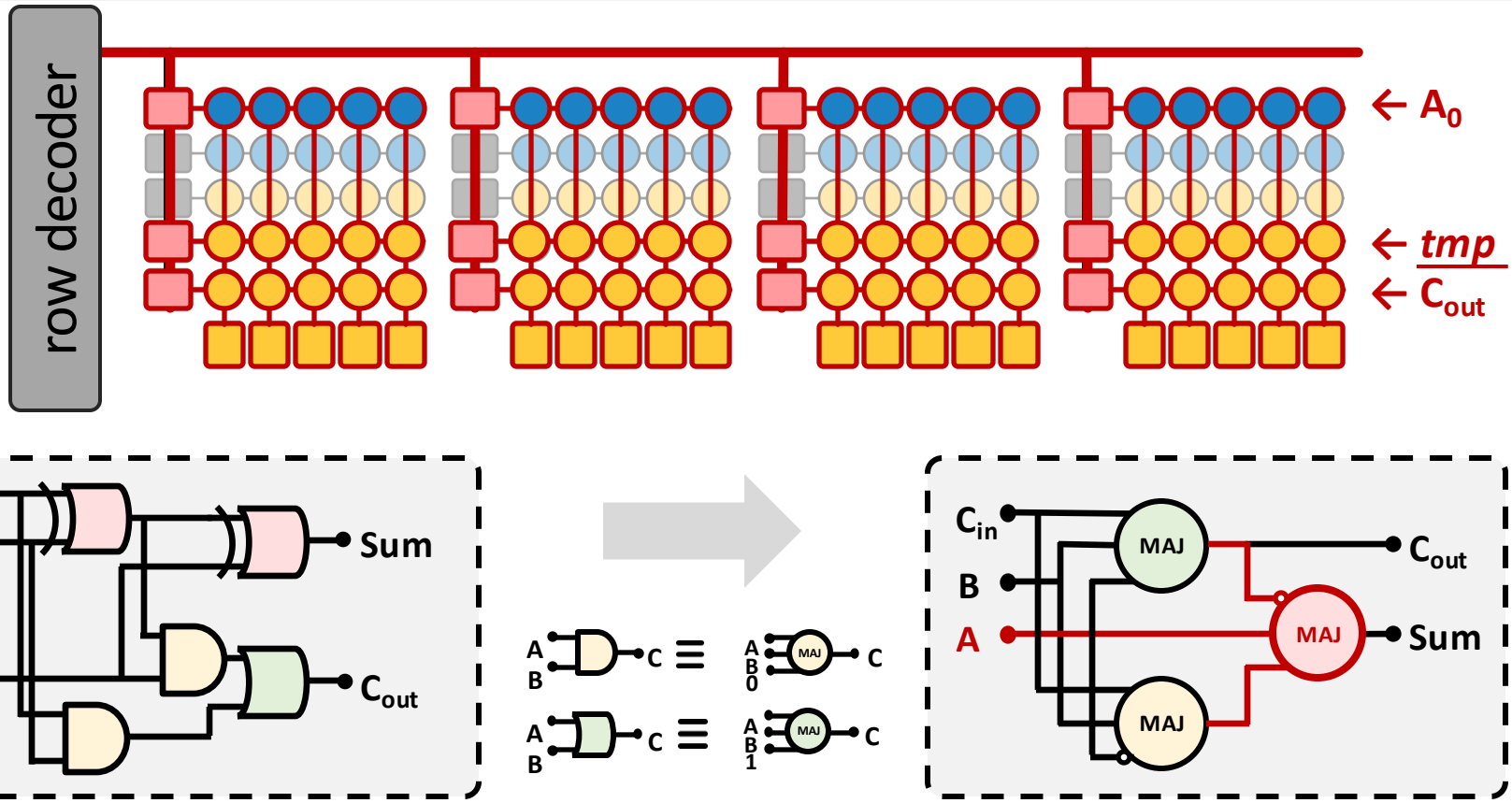
Bulk bitwise arithmetic can be performed by
orchestrating in-DRAM row copy and majority operations



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

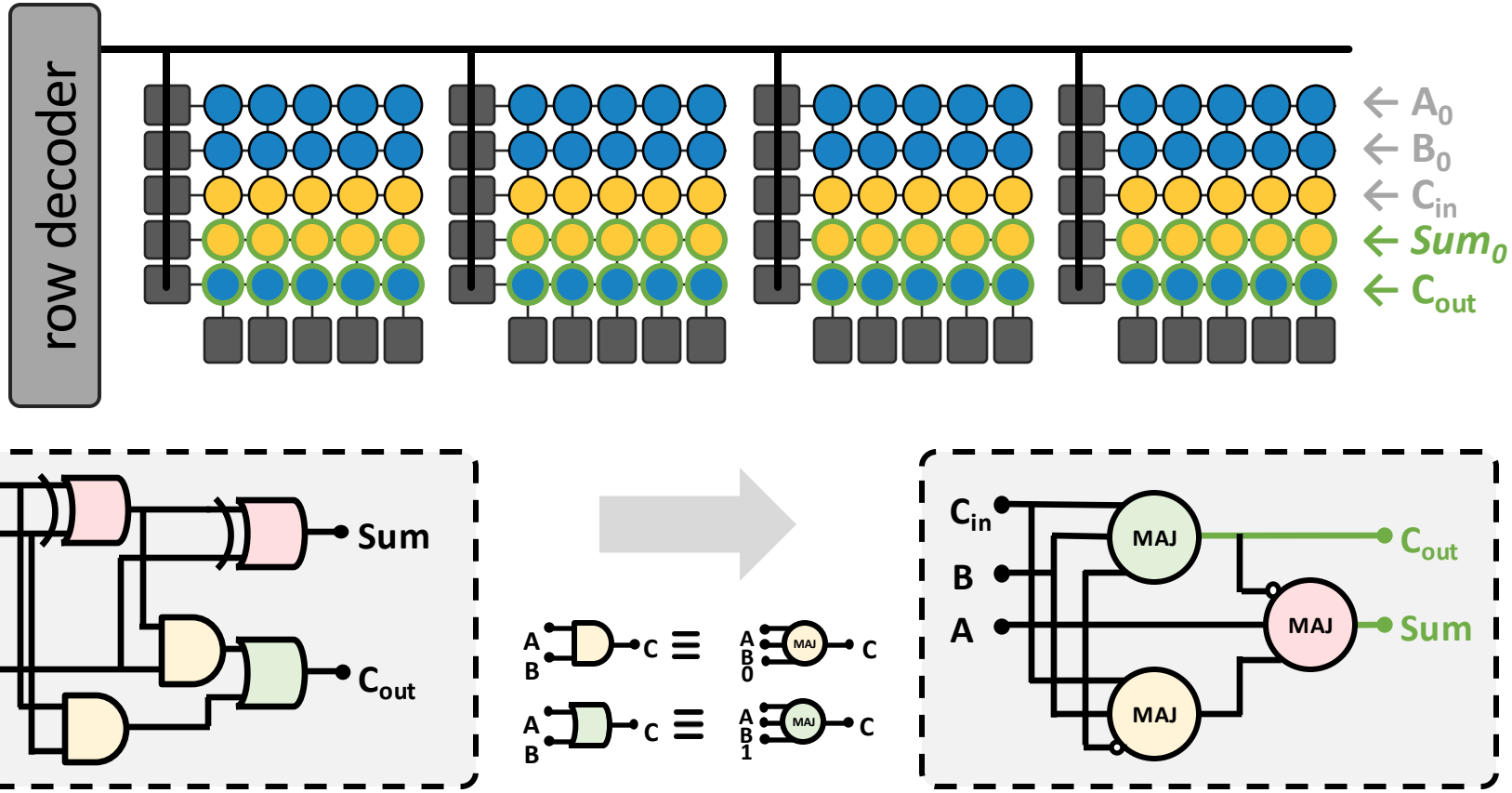
Bulk bitwise arithmetic can be performed by
orchestrating in-DRAM row copy and majority operations



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

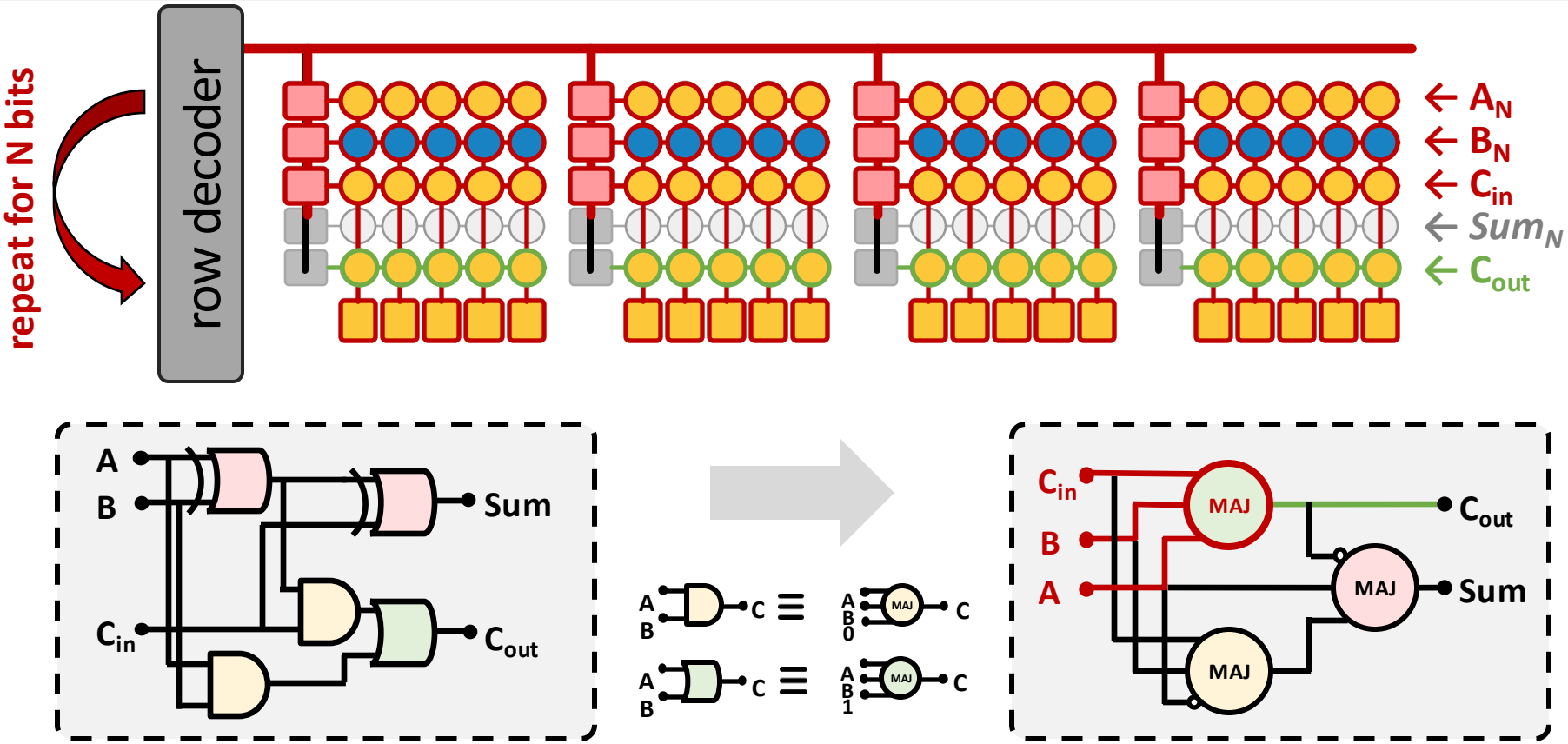
Bulk bitwise arithmetic can be performed by
orchestrating in-DRAM row copy and majority operations



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

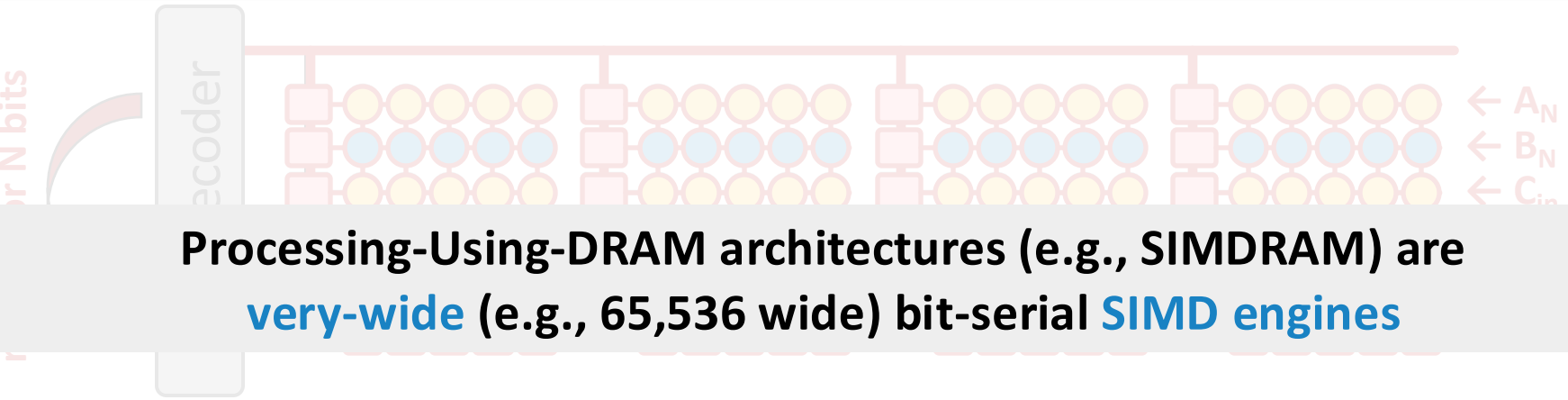
Bulk bitwise arithmetic can be performed by
orchestrating in-DRAM row copy and majority operations



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 Bit-Serial 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

2 Throughput-Oriented Execution with Low Latency Tolerance
- failing to reduce the latency of a single PUD operation

3 Scalability Challenges for High-Precision Operations
- due to the linear/quadratically scaling nature of bit-serial algorithms

Limitations of Bit-Serial 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

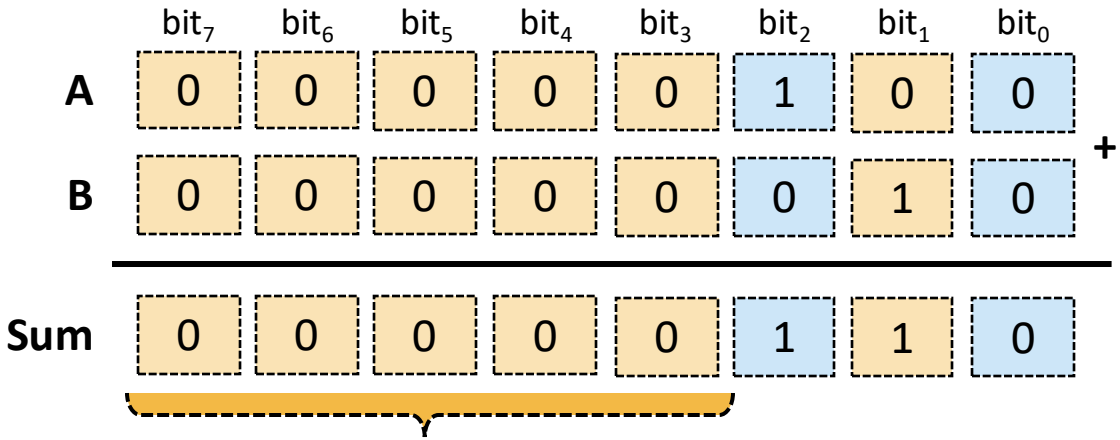
2 Throughput-Oriented Execution with Low Latency Tolerance
- failing to reduce the latency of a single PUD operation

3 Scalability Challenges for High-Precision Operations
- due to the linear/quadratically scaling nature of bit-serial algorithms

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 → increased latency and energy

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

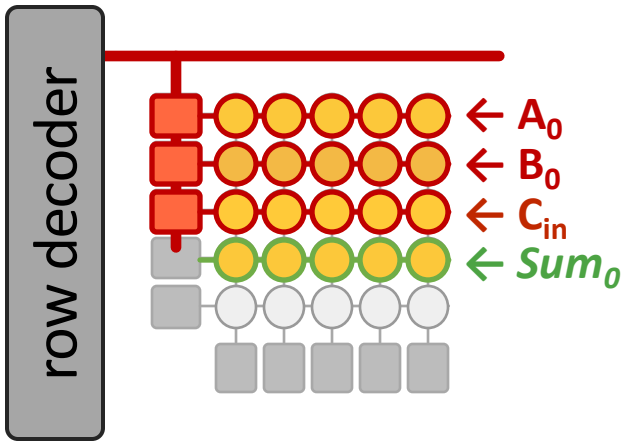
Narrow Values:

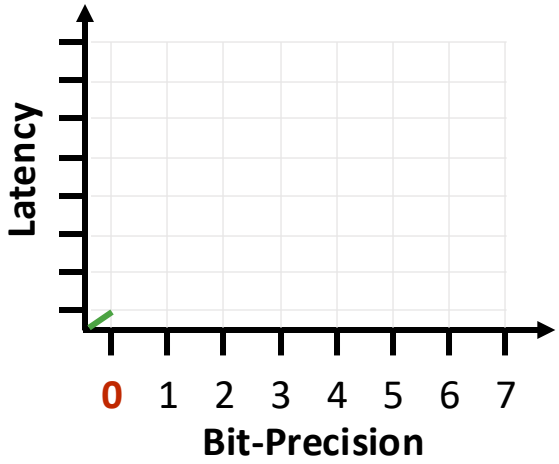
data with small dynamic range stored in large data formats

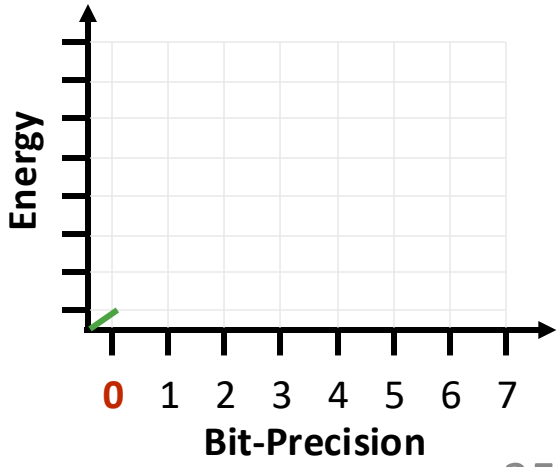
	bit_7	bit_6	bit_5	bit_4	bit_3	bit_2	bit_1	bit_0
A	0	0	0	0	0	1	0	0
B	0	0	0	0	0	0	1	0
Sum	0	0	0	0	0	1	1	0

+

inconsequential bits → increased latency and energy







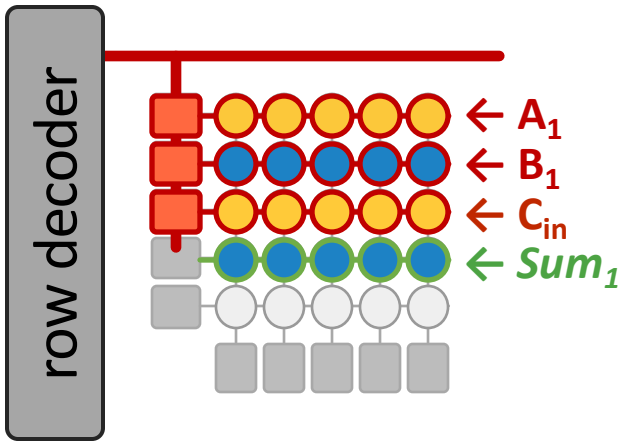
Limitations of Bit-Serial PUD Systems:
Rigid and Static Data Representation (I)

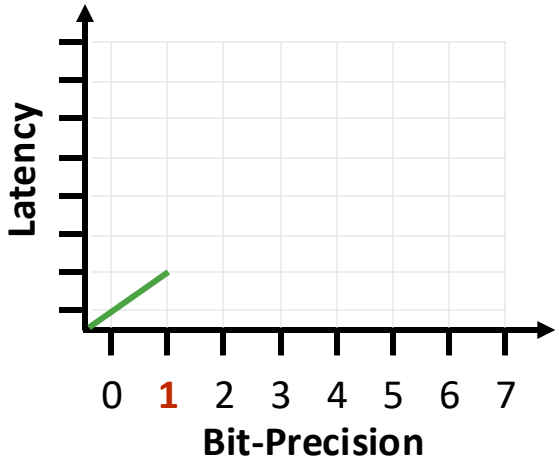
Narrow Values:

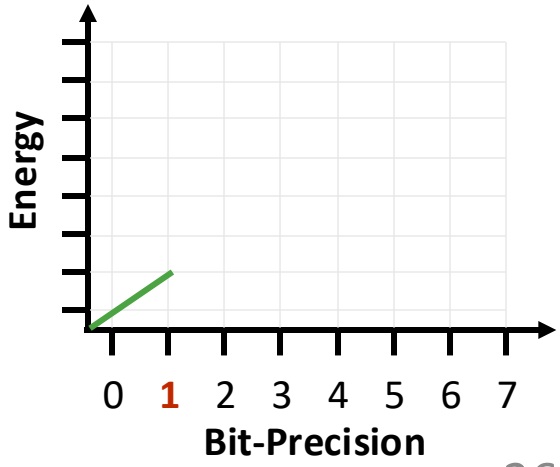
data with small dynamic range stored in large data formats

	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 → increased latency and energy







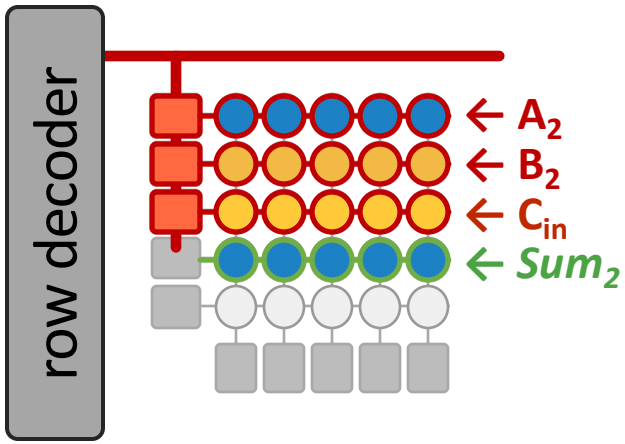
Limitations of Bit-Serial PUD Systems:
Rigid and Static Data Representation (I)

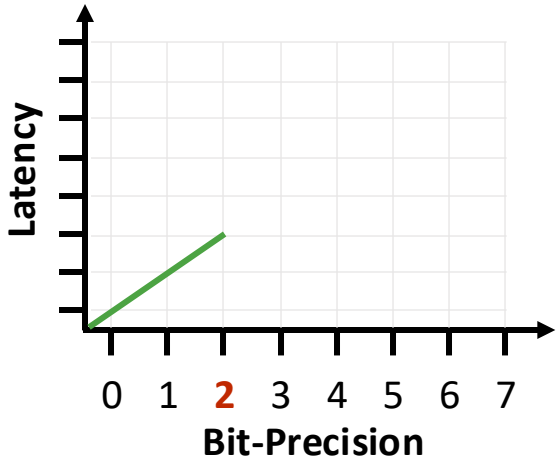
Narrow Values:

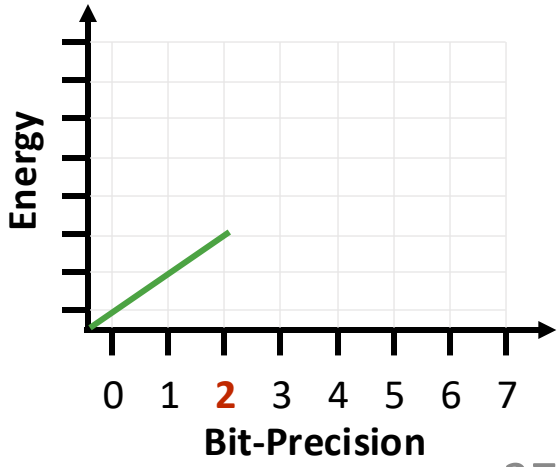
data with small dynamic range stored in large data formats

	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 → increased latency and energy







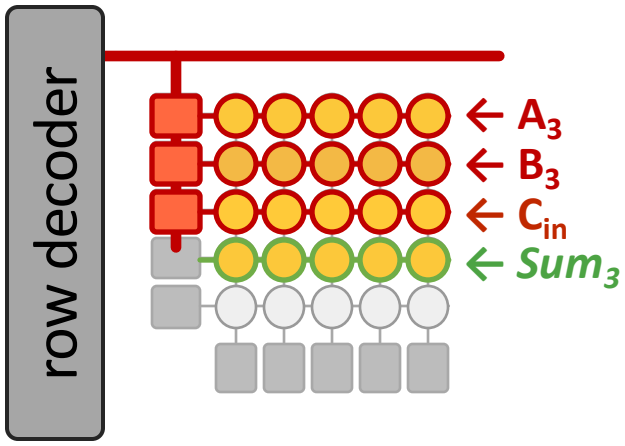
Limitations of Bit-Serial PUD Systems:
Rigid and Static Data Representation (I)

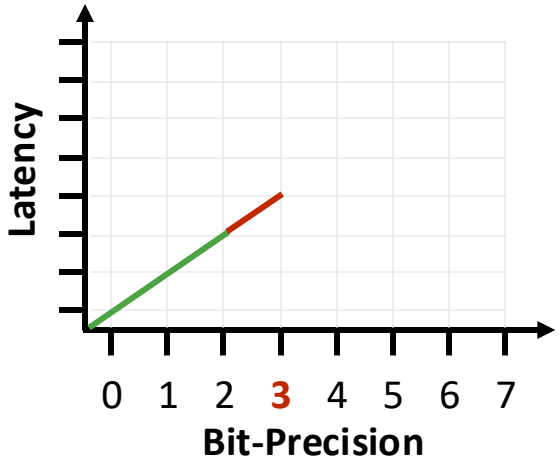
Narrow Values:

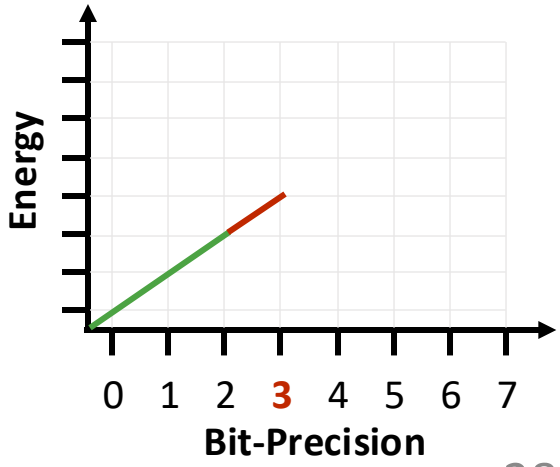
data with small dynamic range stored in large data formats

	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 → wasted latency and energy







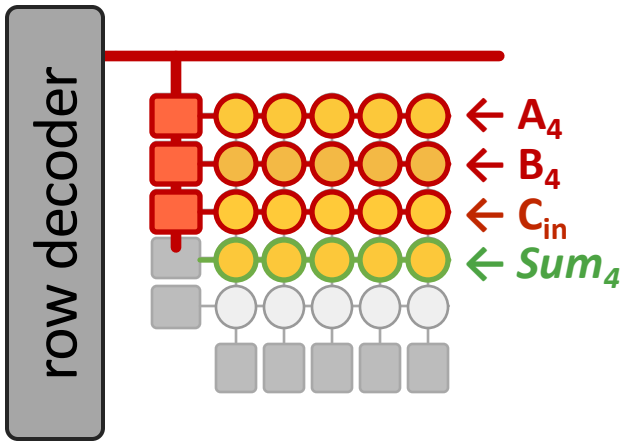
Limitations of Bit-Serial PUD Systems:
Rigid and Static Data Representation (I)

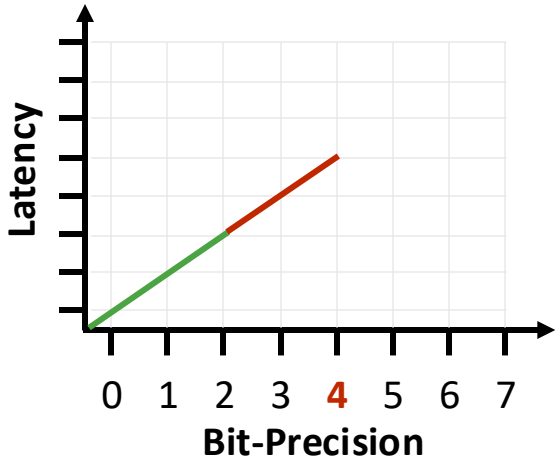
Narrow Values:

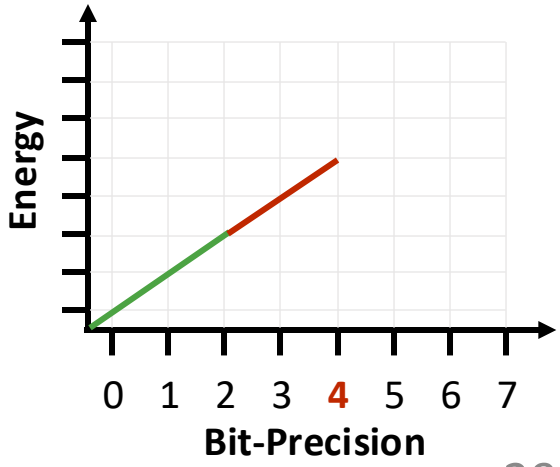
data with small dynamic range stored in large data formats

	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 → wasted latency and energy







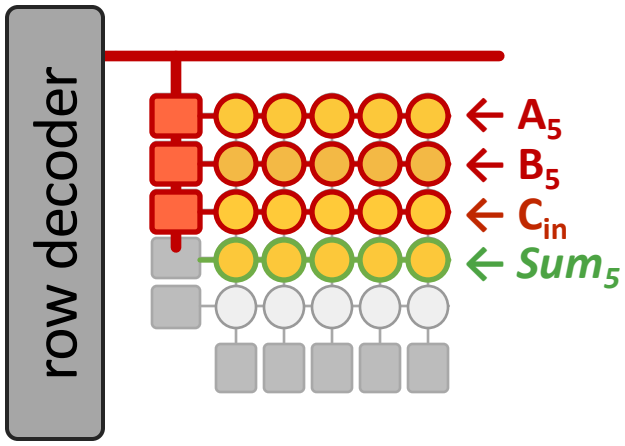
Limitations of Bit-Serial PUD Systems:
Rigid and Static Data Representation (I)

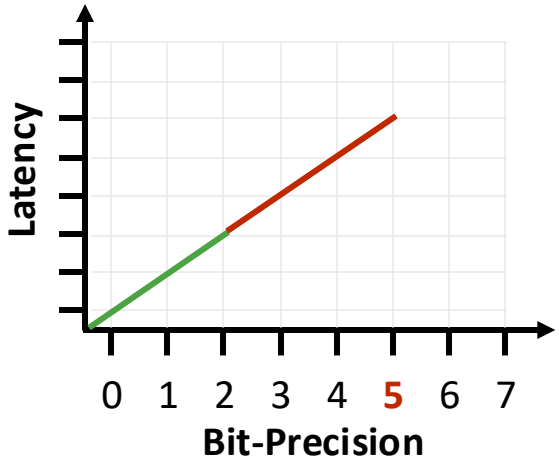
Narrow Values:

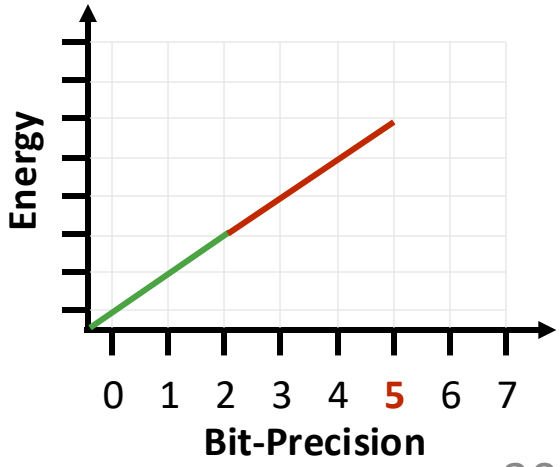
data with small dynamic range stored in large data formats

	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 → wasted latency and energy







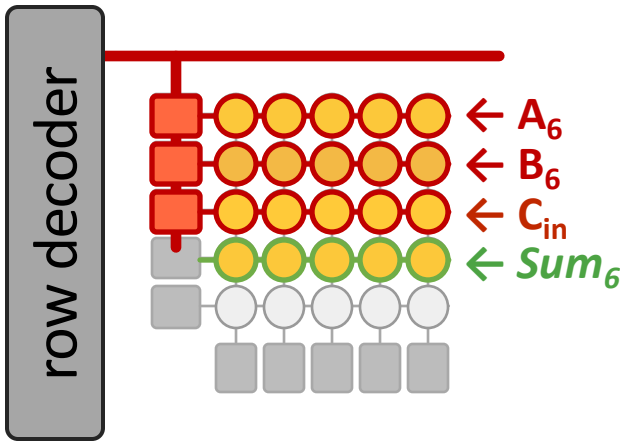
Limitations of Bit-Serial PUD Systems:
Rigid and Static Data Representation (I)

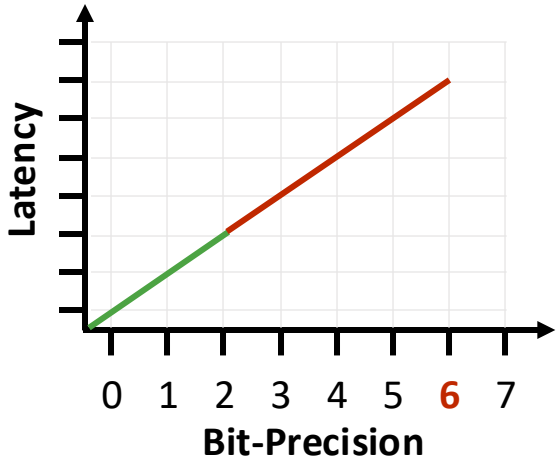
Narrow Values:

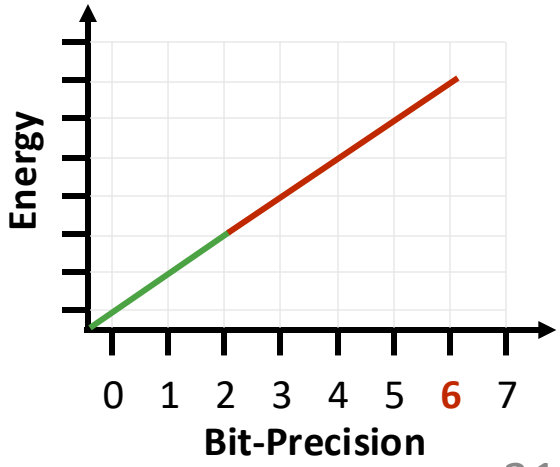
data with small dynamic range stored in large data formats

	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 → wasted latency and energy







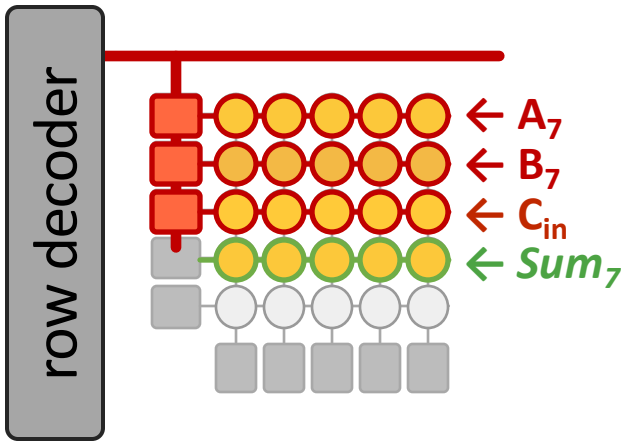
Limitations of Bit-Serial PUD Systems:
Rigid and Static Data Representation (I)

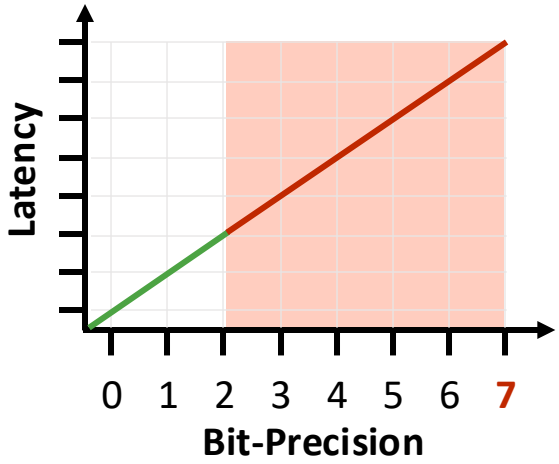
Narrow Values:

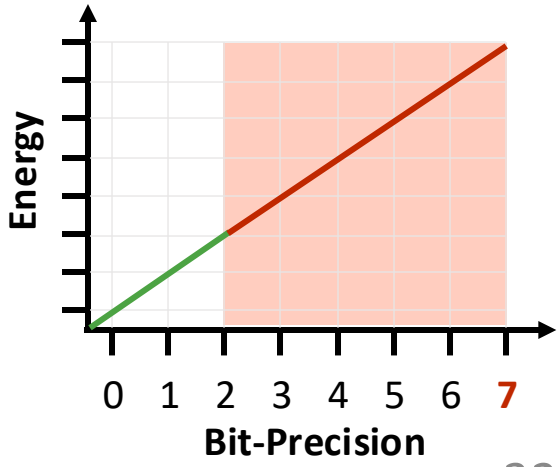
data with small dynamic range stored in large data formats

	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 → wasted latency and energy



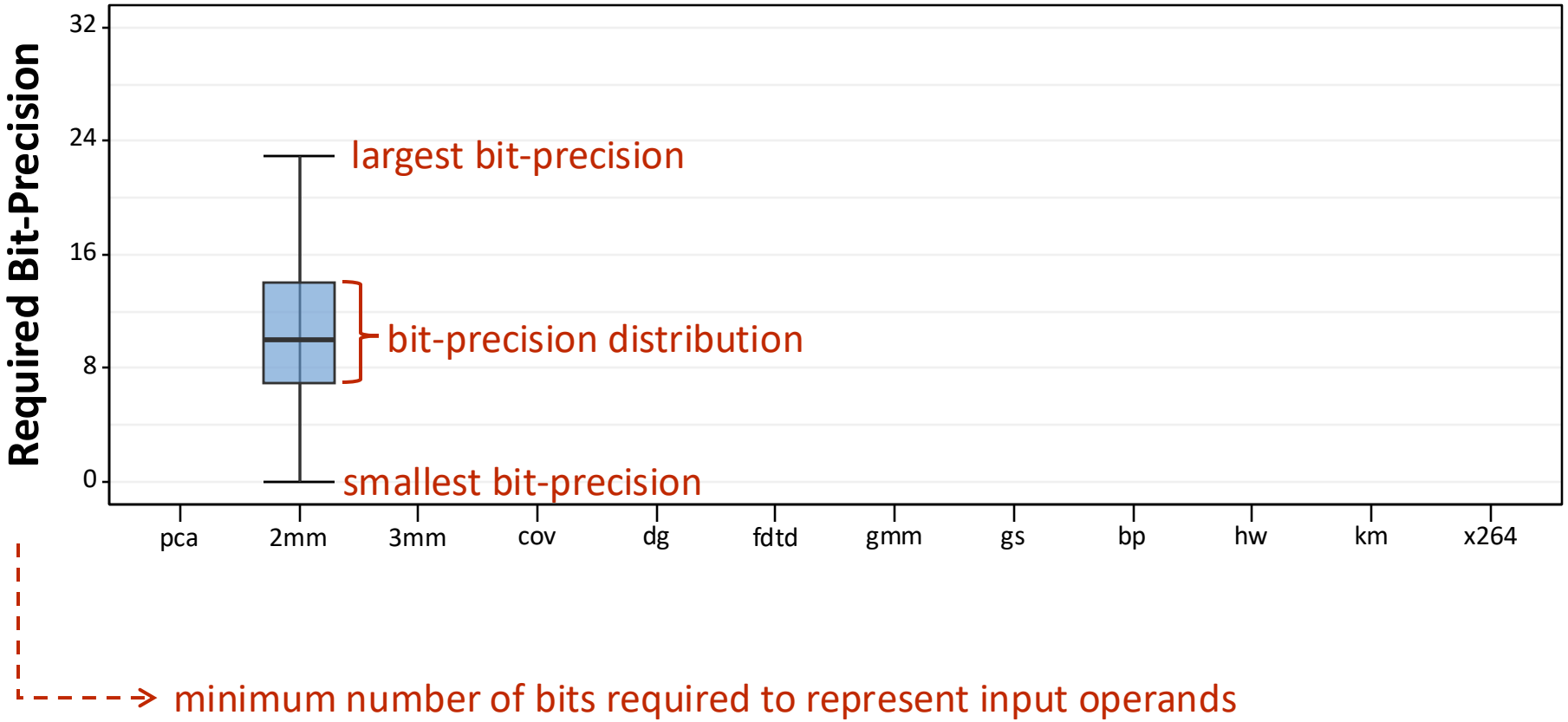




Limitations of Bit-Serial PUD Systems:
Rigid and Static Data Representation (II)

Application Analysis:

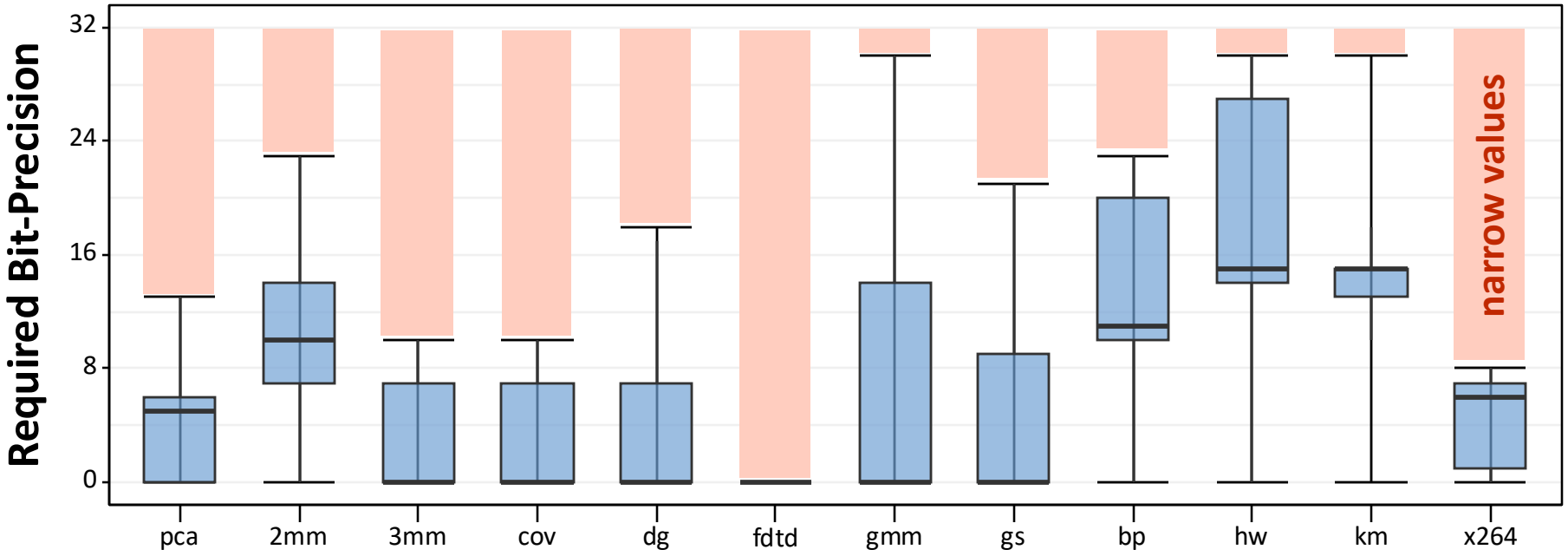
quantify the amount required bit-precision in real applications



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

Limitations of Bit-Serial 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

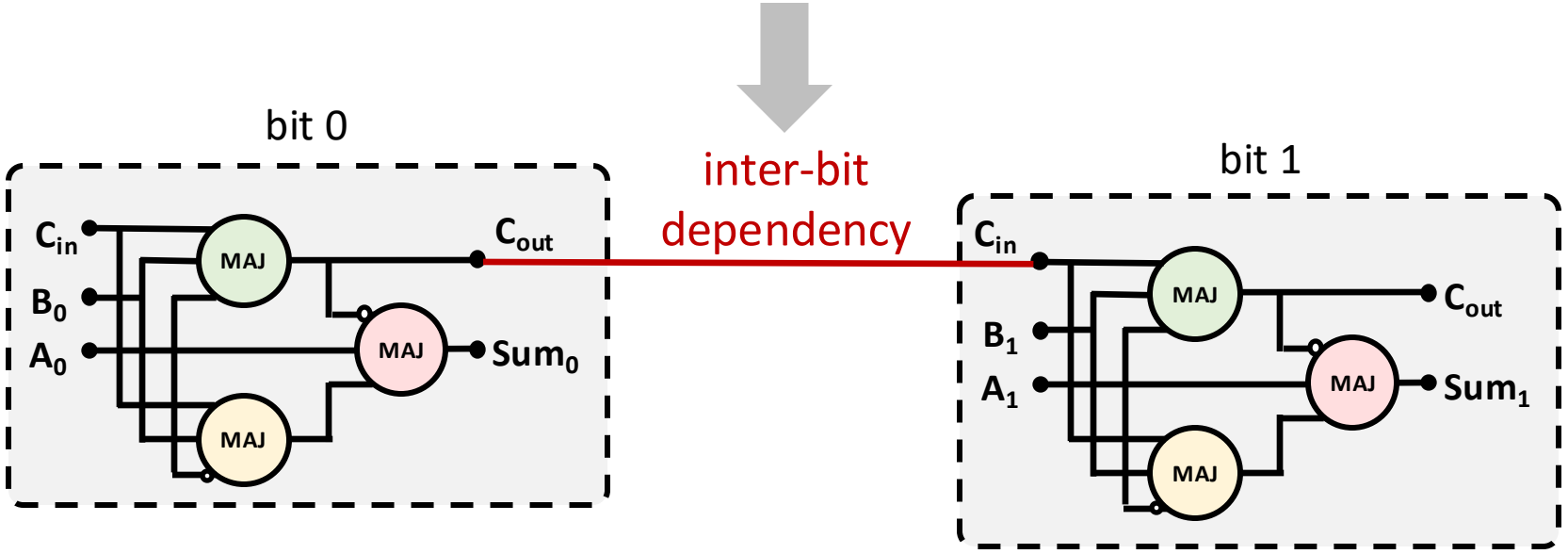
2 **Throughput-Oriented Execution with Low Latency Tolerance**
- failing to reduce the latency of a single PUD operation

3 **Scalability Challenges for High-Precision Operations**
- due to the linear/quadratically scaling nature of bit-serial algorithms

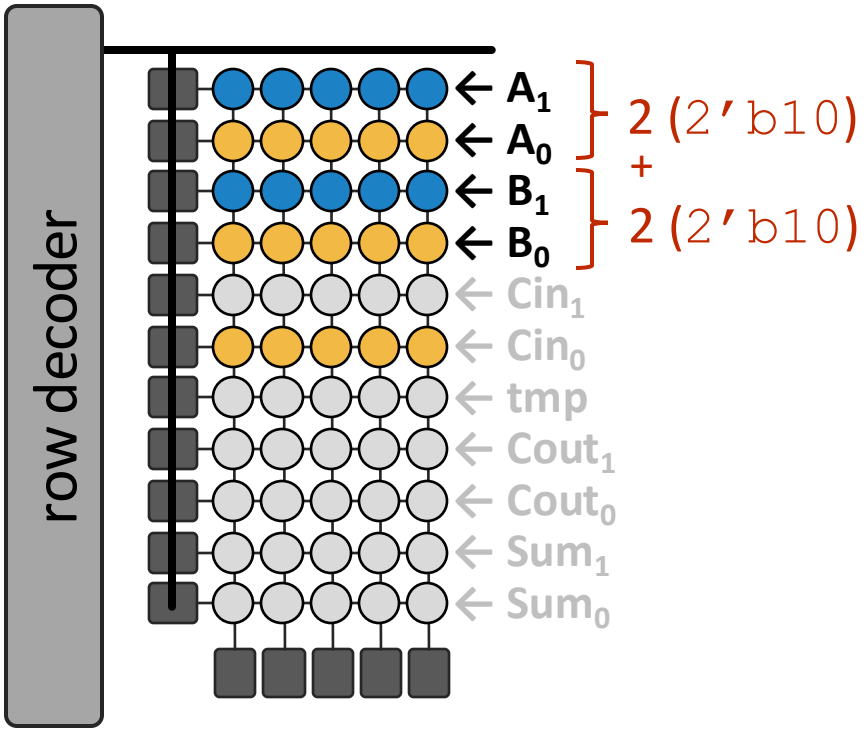
Limitations of Bit-Serial PUD Systems:
Throughput Oriented Execution (I)

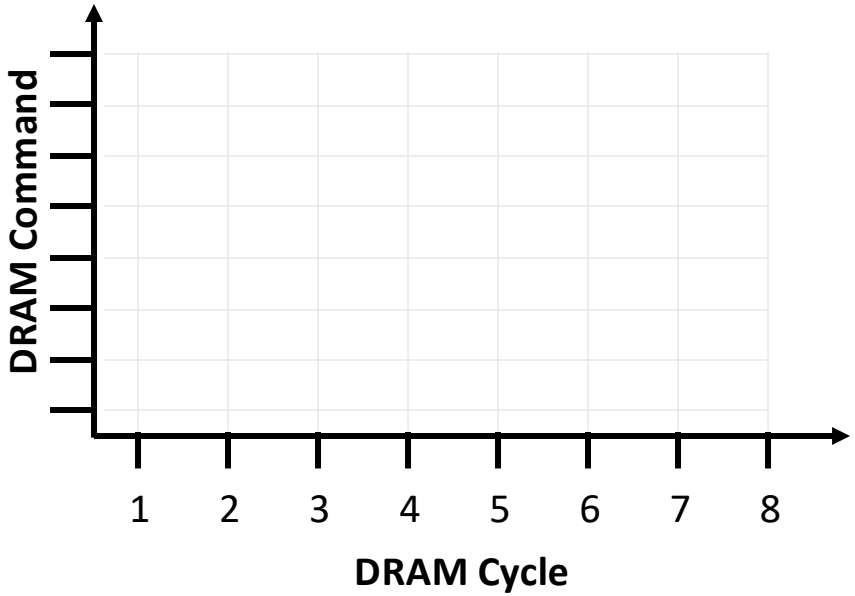
Problem

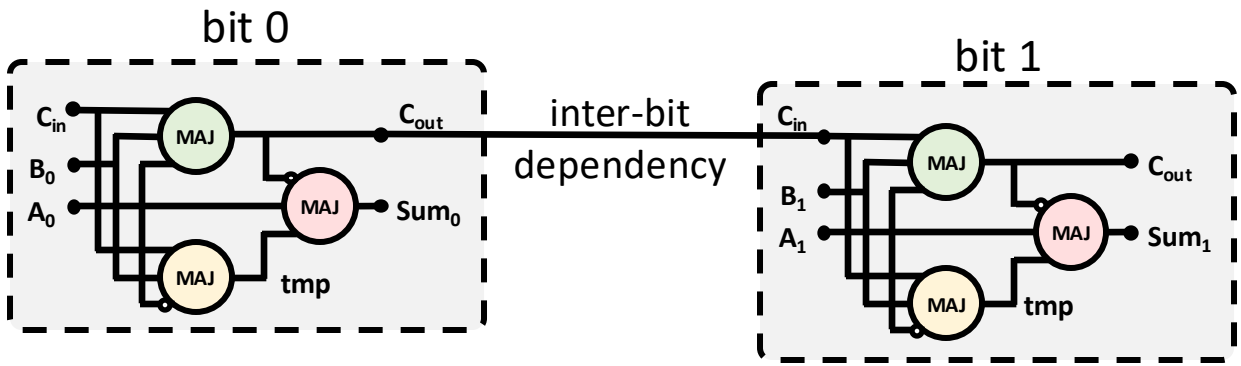
PUD operations are inherently slow
→ need to operate on each bit serially to perform an operation with a data width > 1



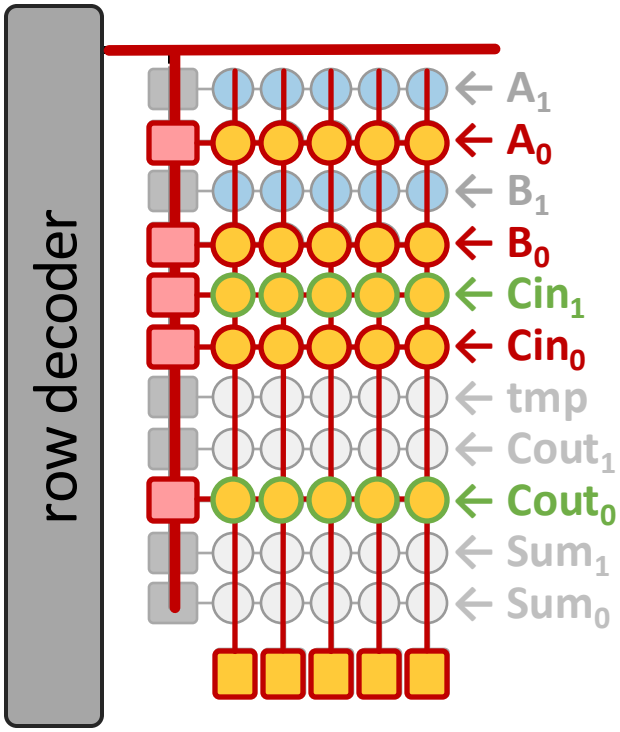
Limitations of Bit-Serial PUD Systems:
Throughput Oriented Execution (II)

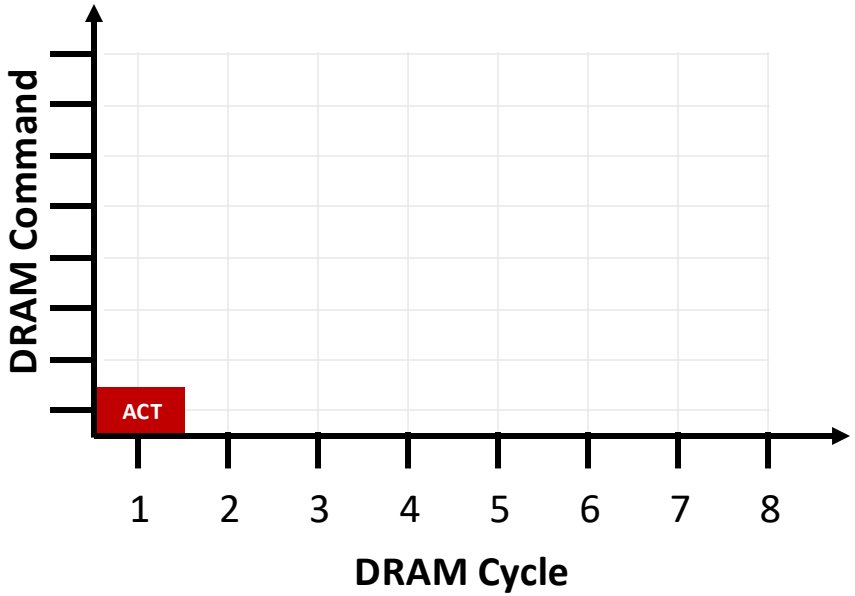


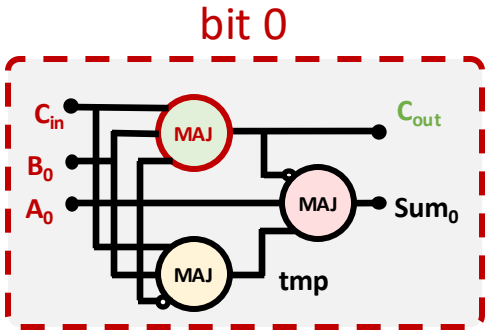


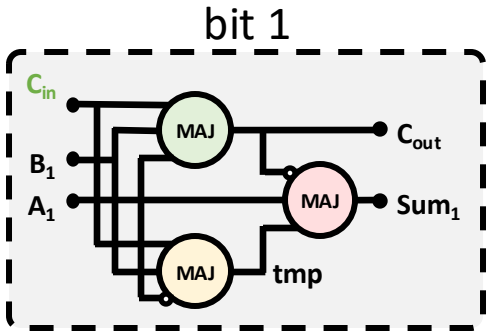


Limitations of Bit-Serial PUD Systems:
Throughput Oriented Execution (II)

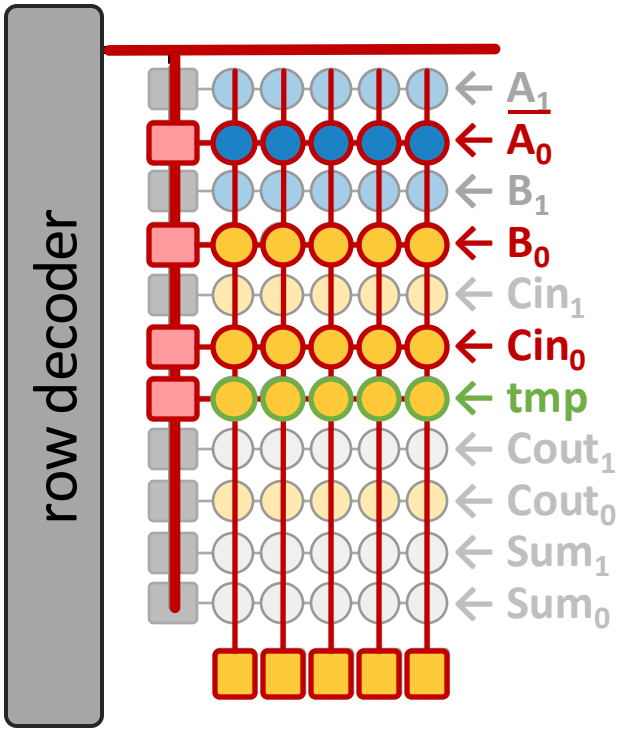


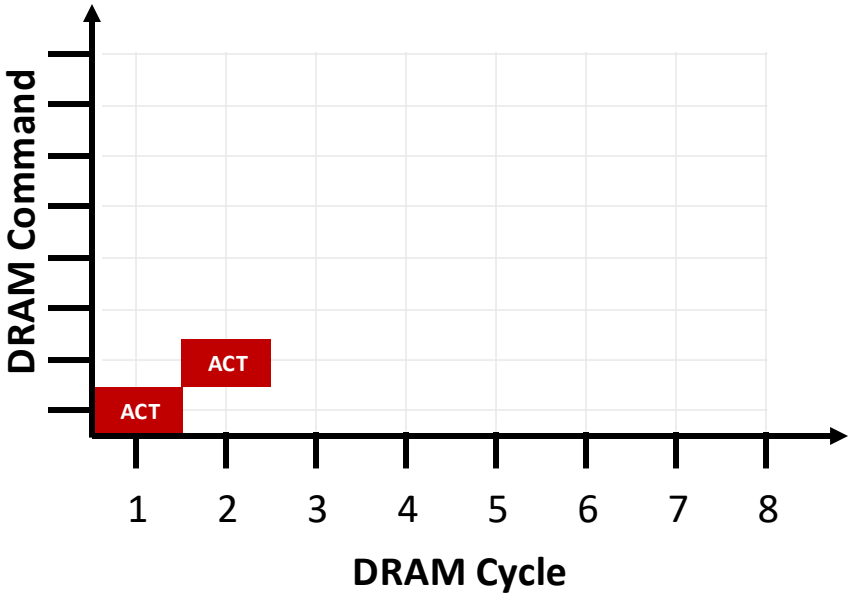


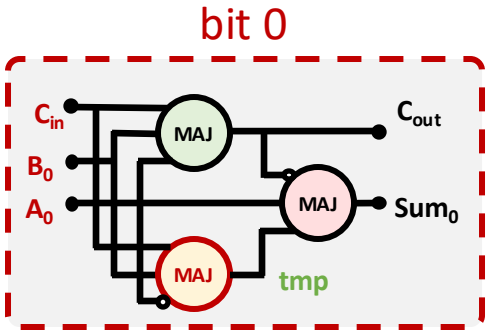


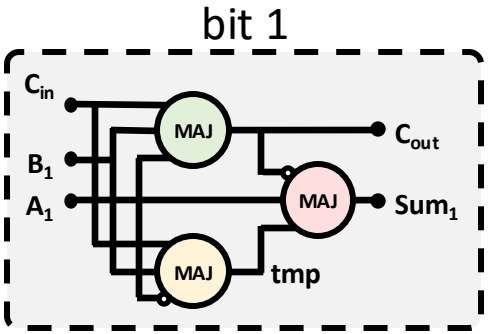


Limitations of Bit-Serial PUD Systems:
Throughput Oriented Execution (II)

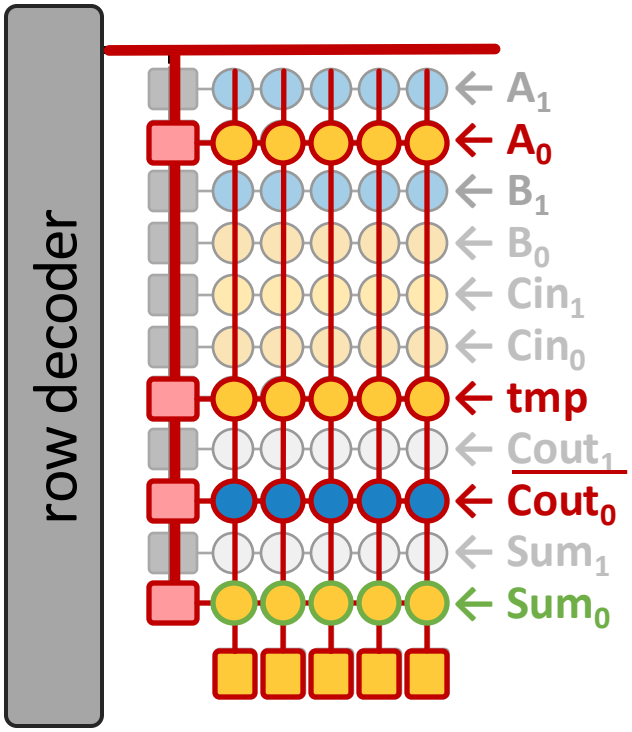


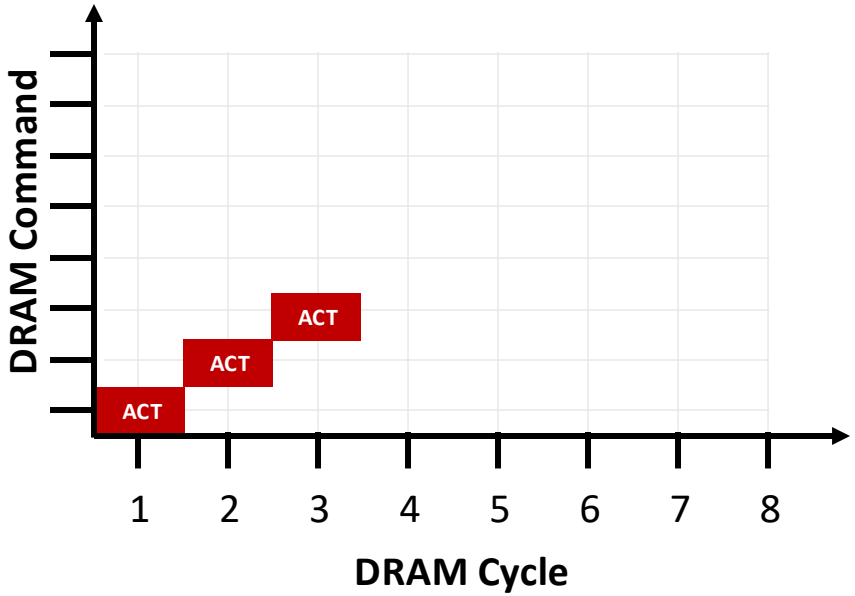


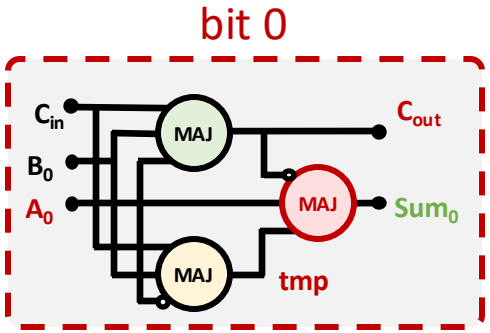


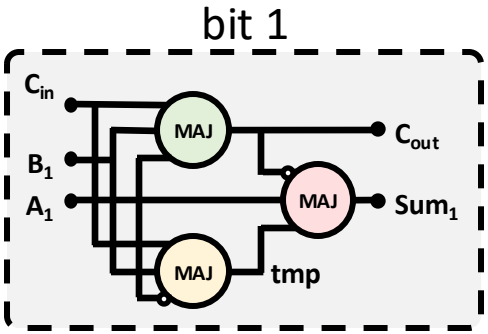


Limitations of Bit-Serial PUD Systems:
Throughput Oriented Execution (II)

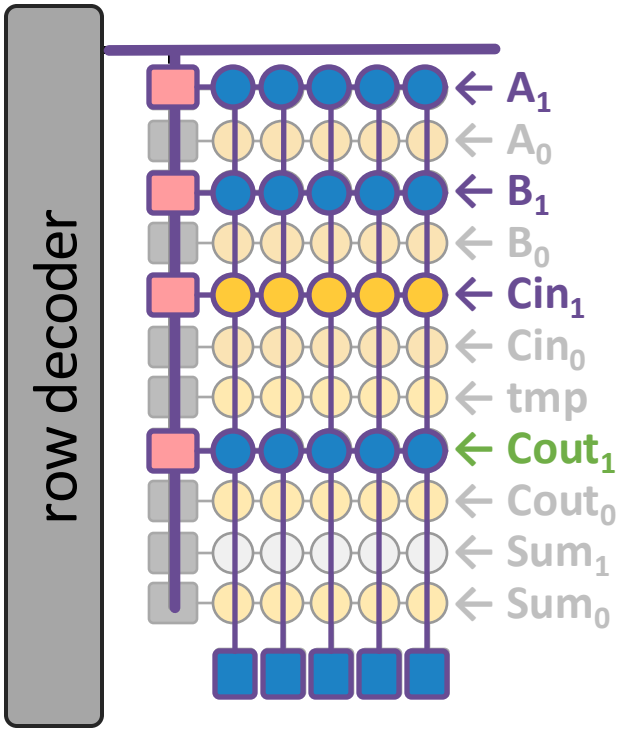


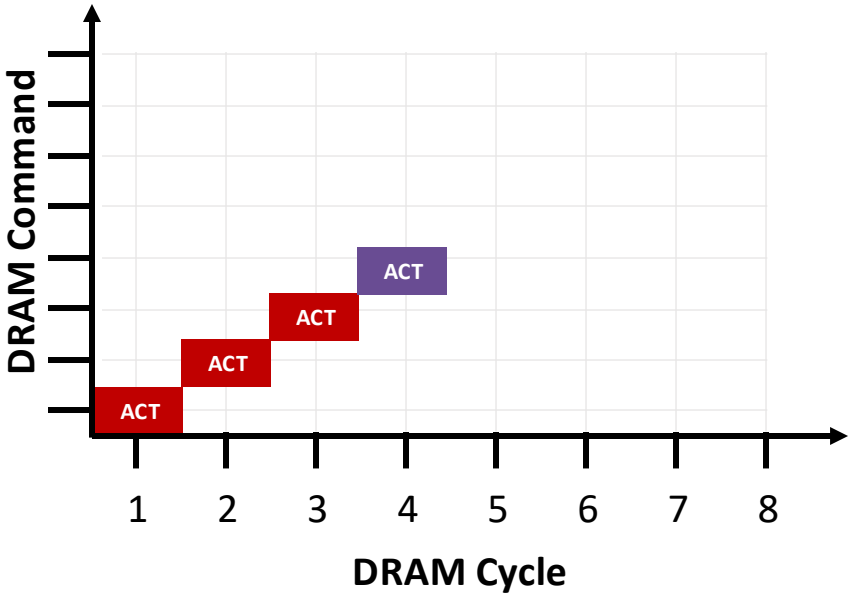


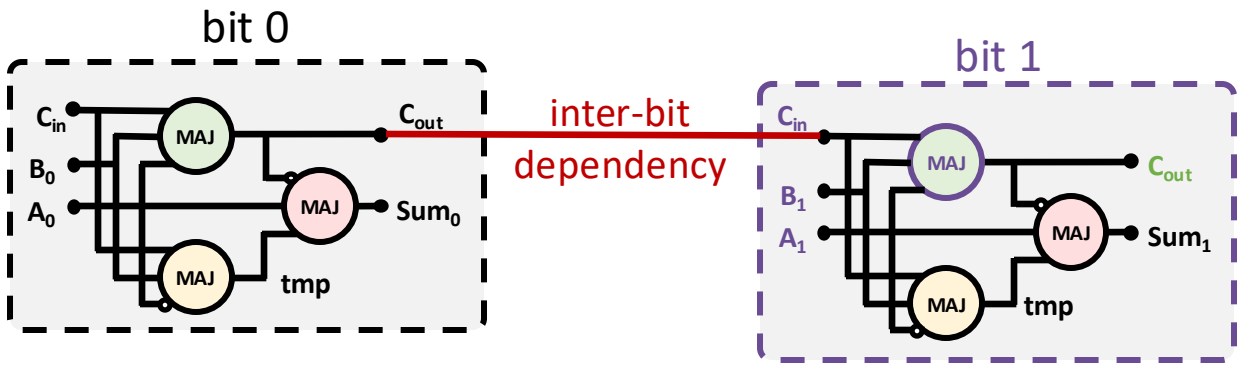




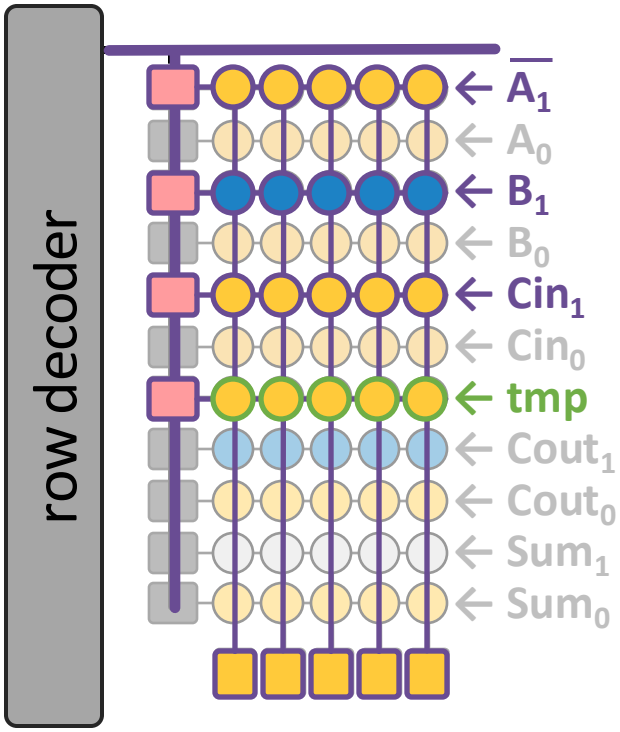
Limitations of Bit-Serial PUD Systems:
Throughput Oriented Execution (II)

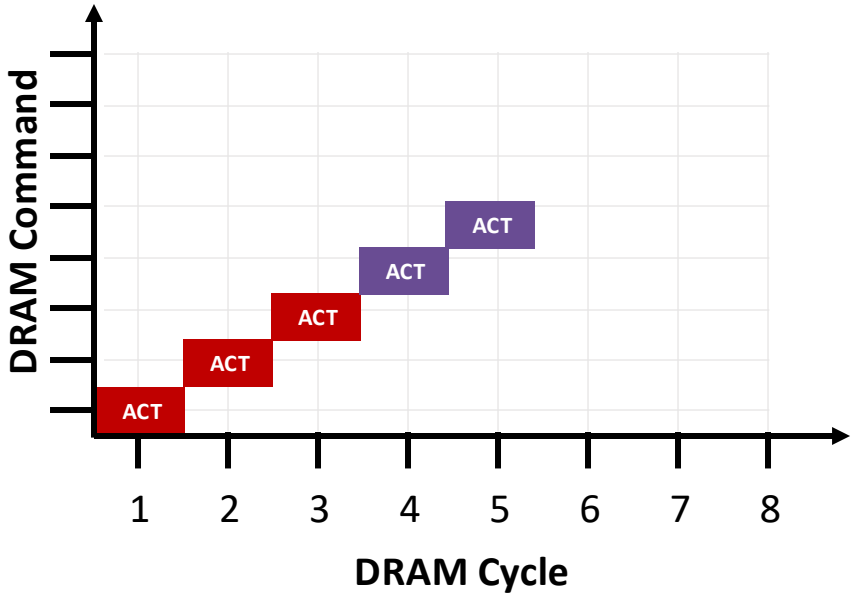


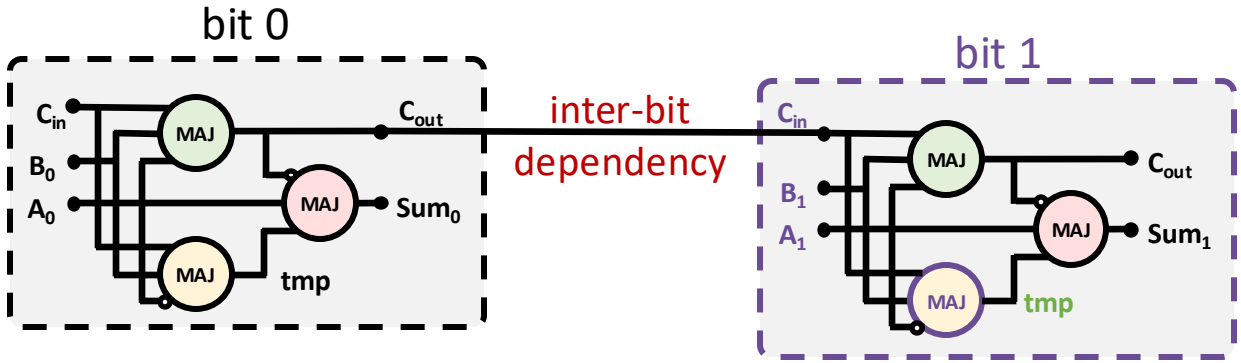




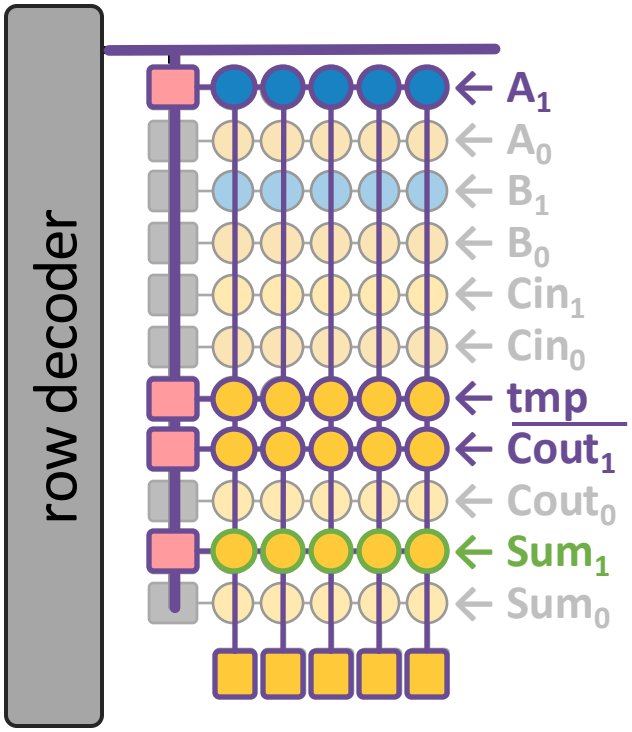
Limitations of Bit-Serial PUD Systems:
Throughput Oriented Execution (II)

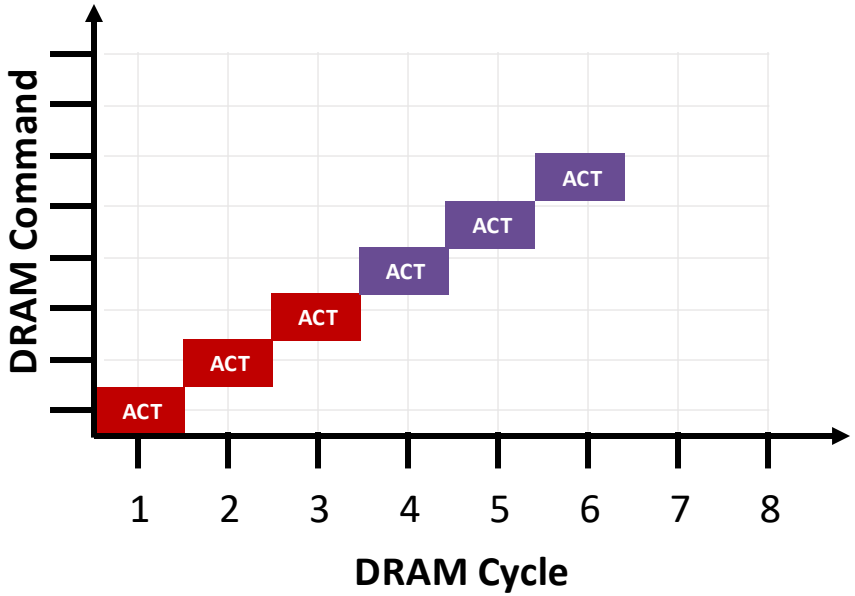


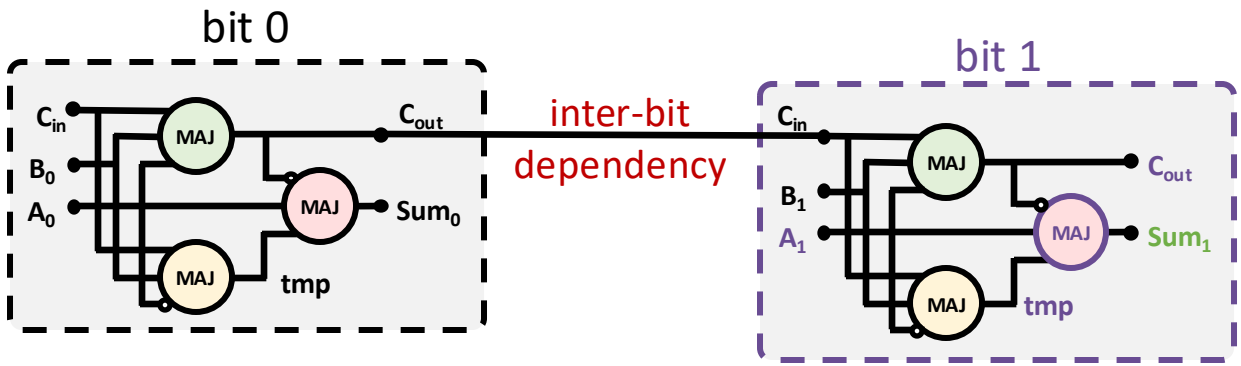




Limitations of Bit-Serial PUD Systems:
Throughput Oriented Execution (II)







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

Problem

Inter-bit dependency forces the serialization of PUD primitives when bits of a data word are placed within the same DRAM subarray

Limitations of Bit-Serial PUD Systems: Overview

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

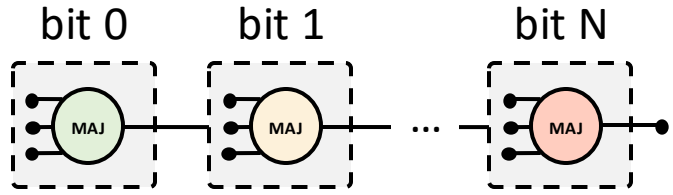
1 Rigid and Static Data Representation
- leading to a subpar performance in the presence of narrow values

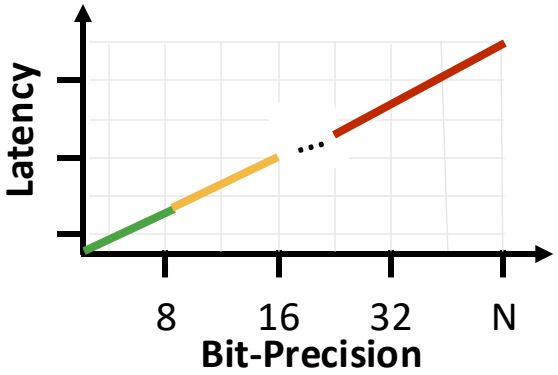
2 Throughput-Oriented Execution with Low Latency Tolerance
- failing to reduce the latency of a single PUD operation

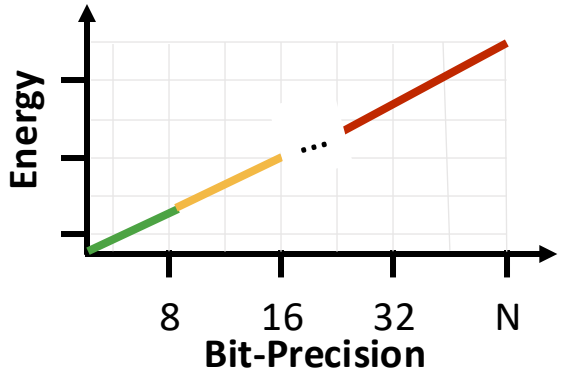
3 Scalability Challenges for High-Precision Operations
- due to the linear/quadratically scaling nature of bit-serial algorithms

Limitations of Bit-Serial PUD Systems:
Scalability Challenges (I)

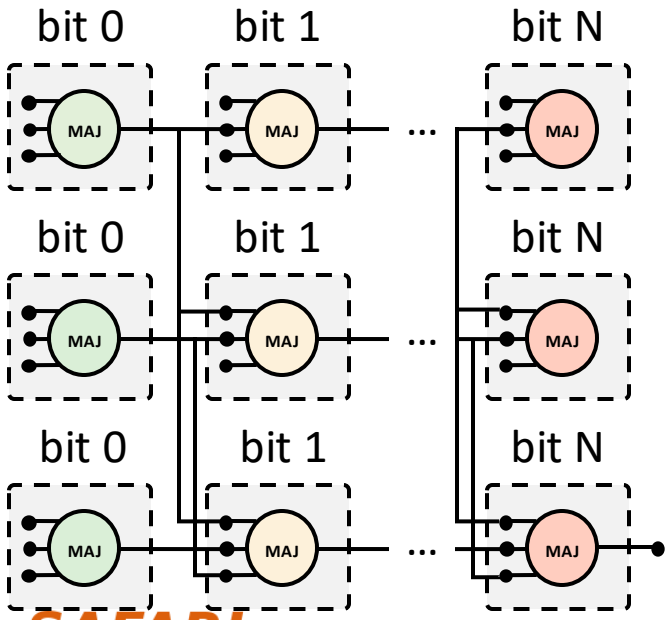
Linearly-Scaling PUD Operations (e.g., addition)

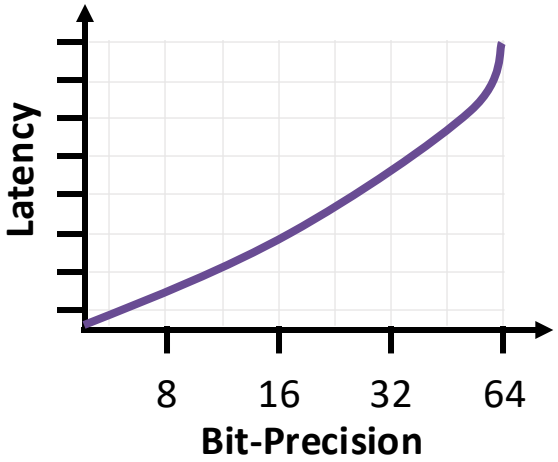


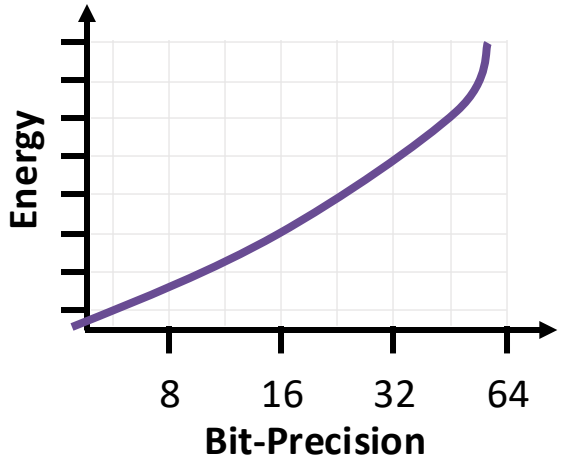




Quadratically-Scaling PUD Operations (e.g., multiplication)







Problem & Goal

Problem

Naively employing a bit-serial execution model for Processing-Using-DRAM architectures limits its applicability and efficiency for different applications

Goal

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

Outline

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

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

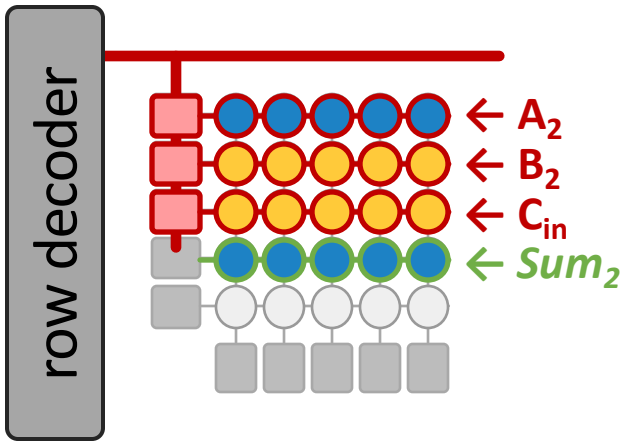
Opportunities for Bit-Serial PUD:
Narrow Values for PUD Operations

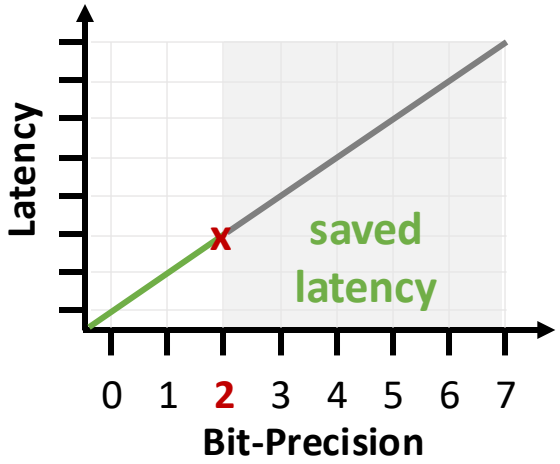
Narrow Values:

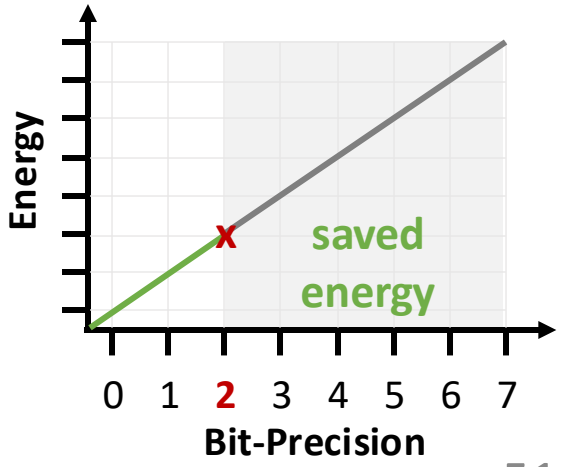
data with small dynamic range stored in large data formats

	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







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

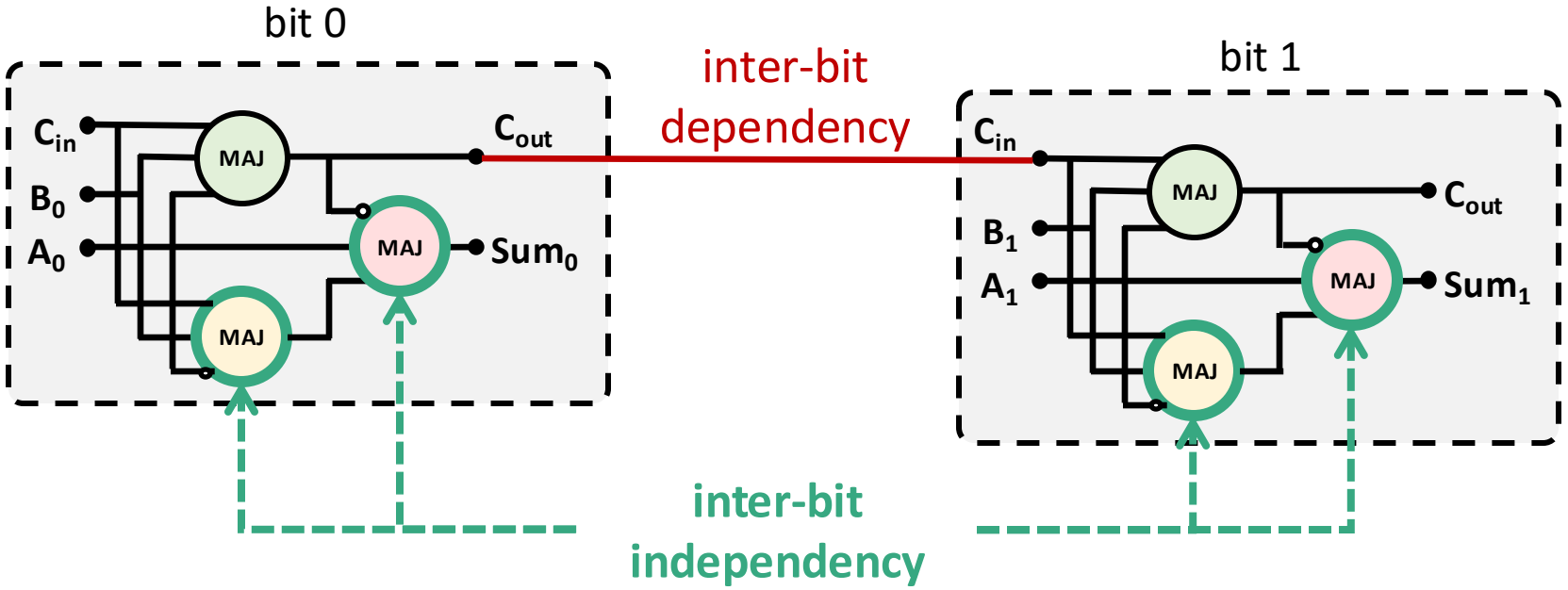
2 Throughput-Oriented Execution with Low Latency Tolerance
- failing to reduce the latency of a single PUD operation
- opportunity 2: DRAM parallelism for latency-oriented execution

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

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

Key Idea

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

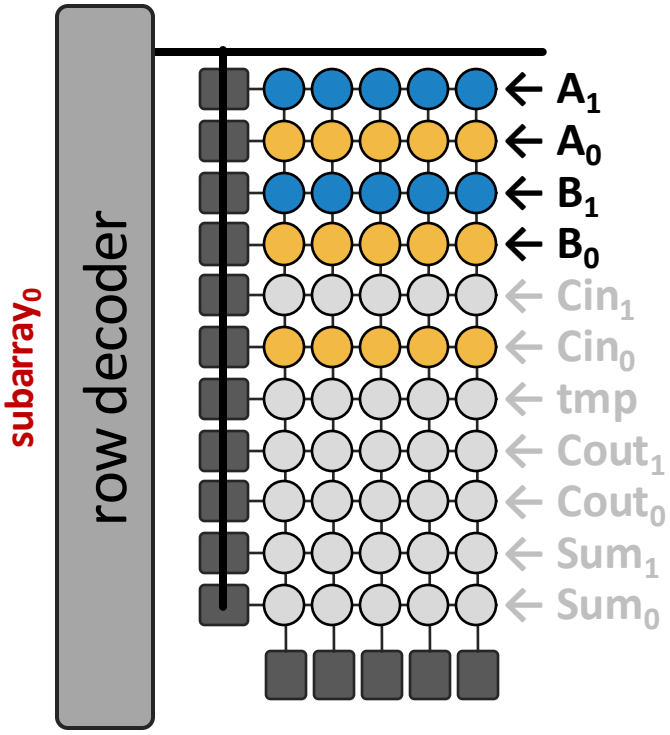


Opportunities for Bit-Serial PUD:
Latency Reduction with Subarray Parallelism

Proteus' ***one-bit per-subarray data mapping*** → **scatter bits of a data word across DRAM subarrays to expose bit-level parallelism**

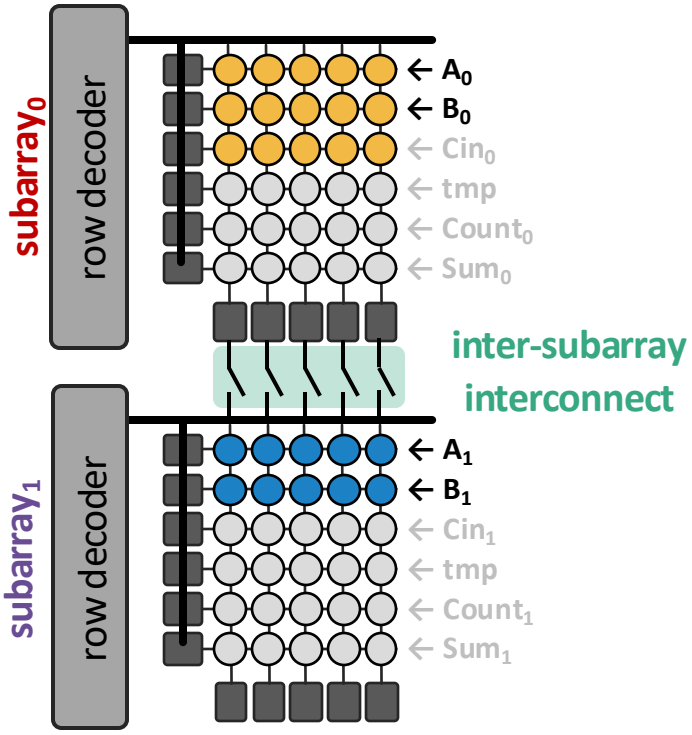
all-bits per-subarray

→ ***forces serialization across bits***

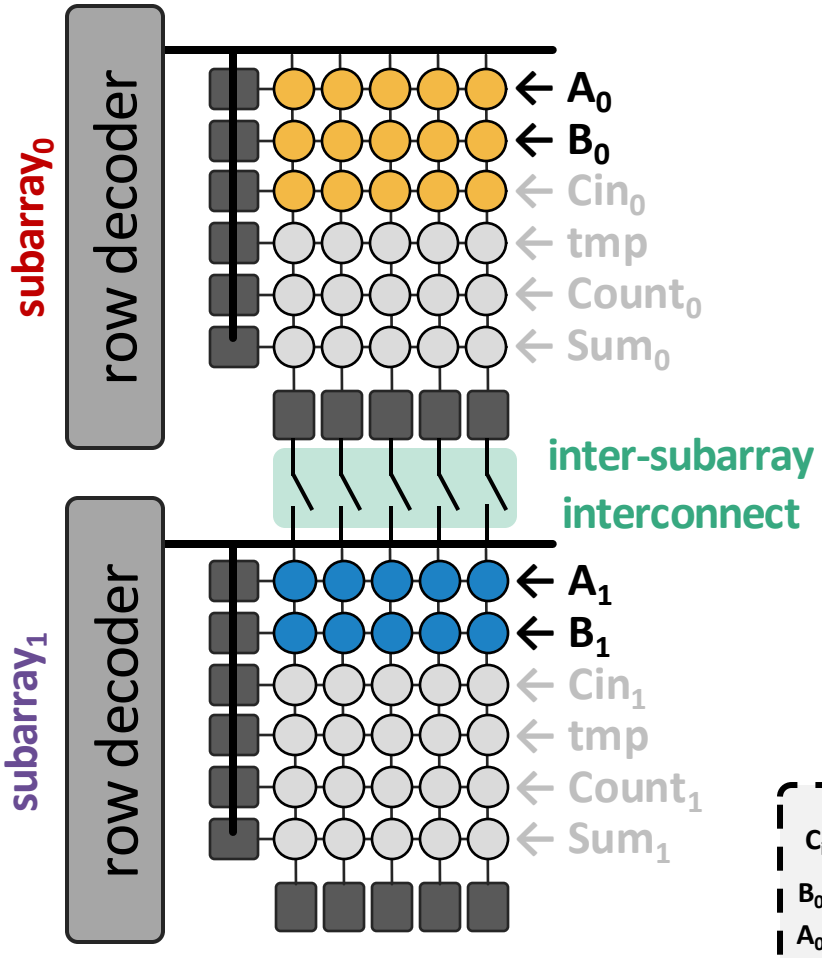


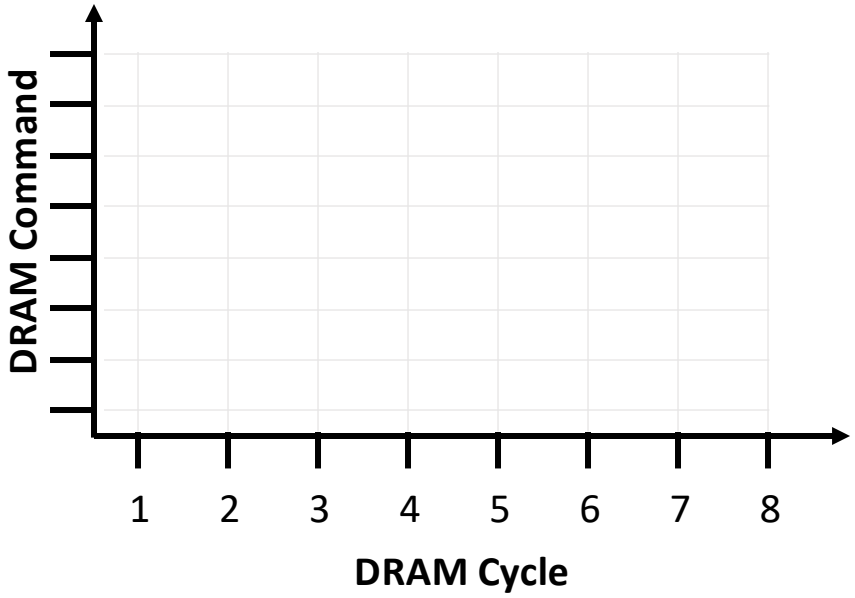
one-bit per-subarray

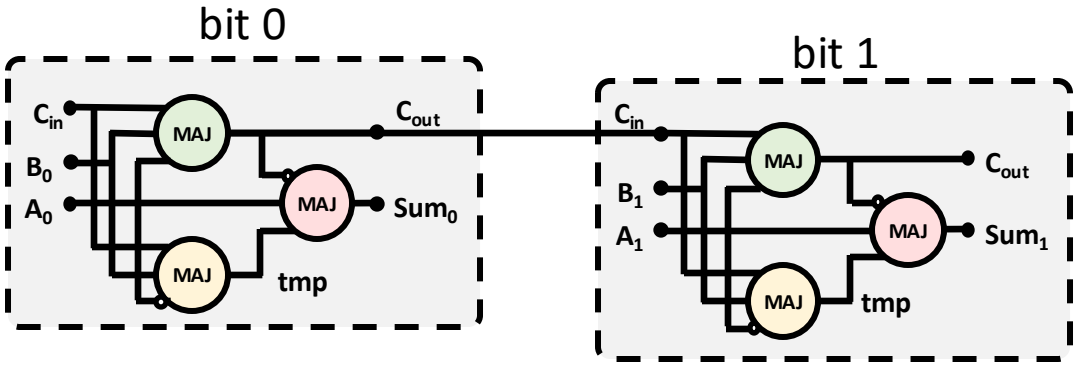
→ **exposes bit-level parallelism**



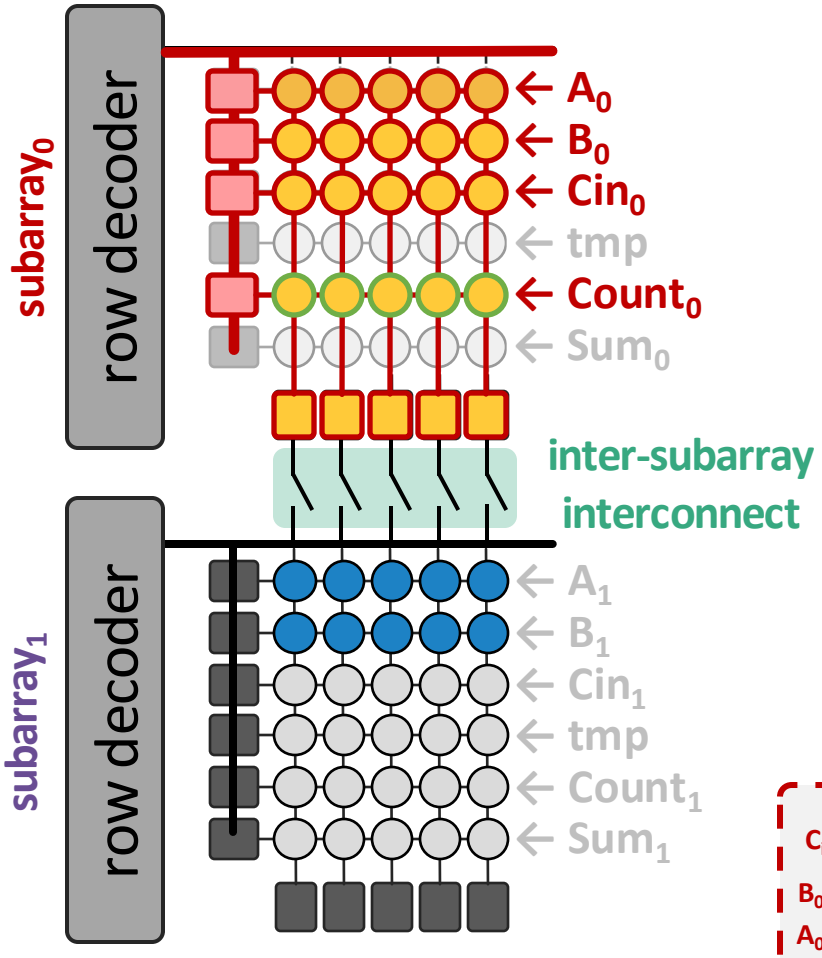
Opportunities for Bit-Serial PUD:
Latency Reduction with Subarray Parallelism

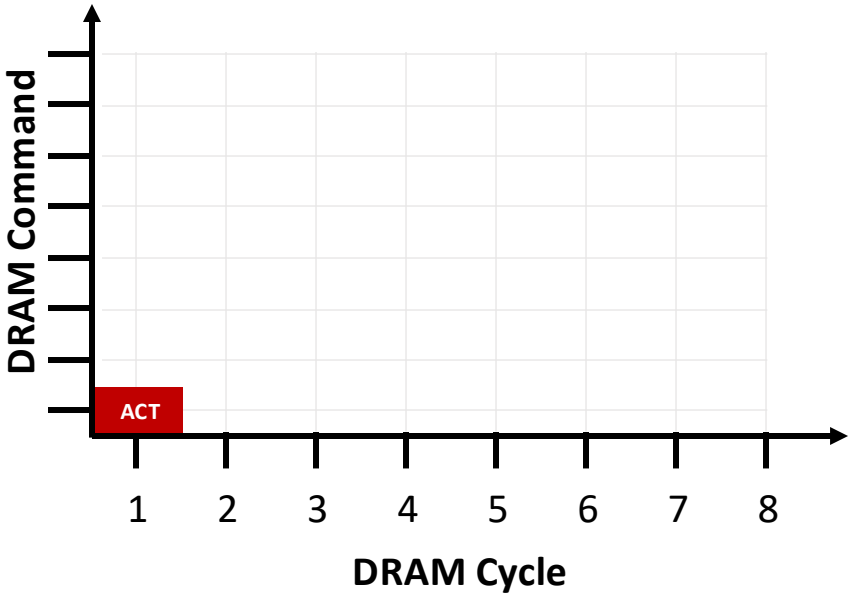


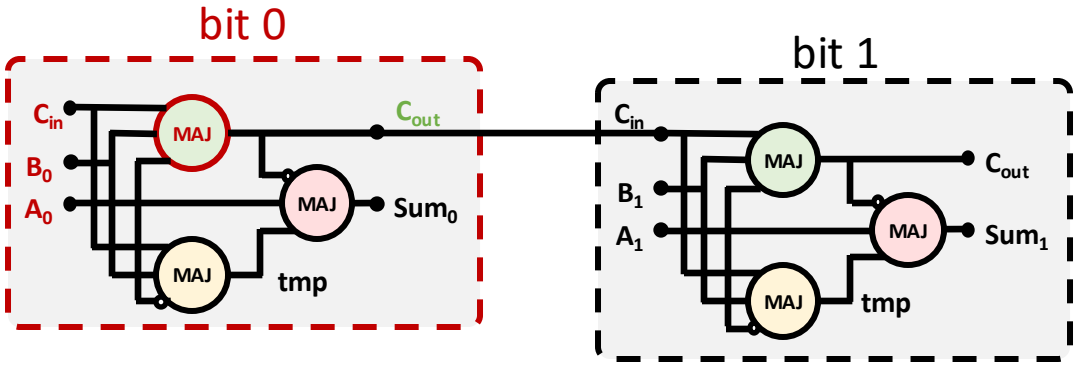




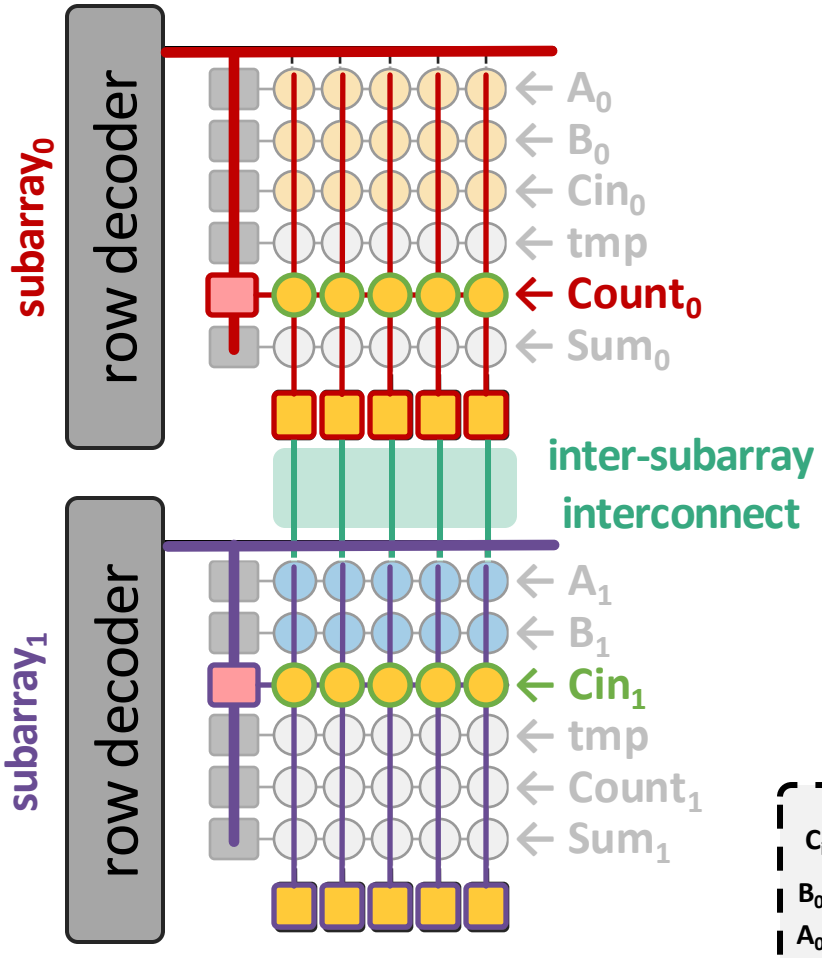
Opportunities for Bit-Serial PUD:
Latency Reduction with Subarray Parallelism

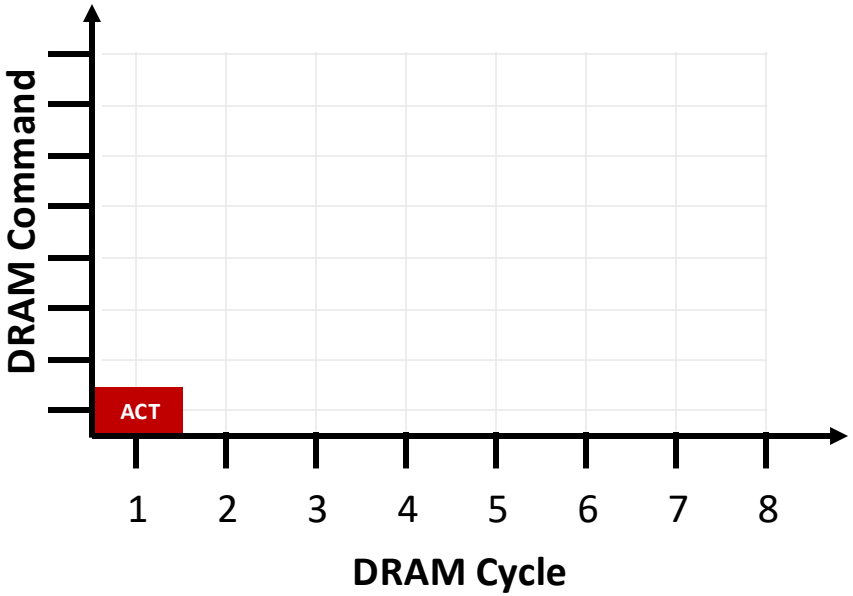


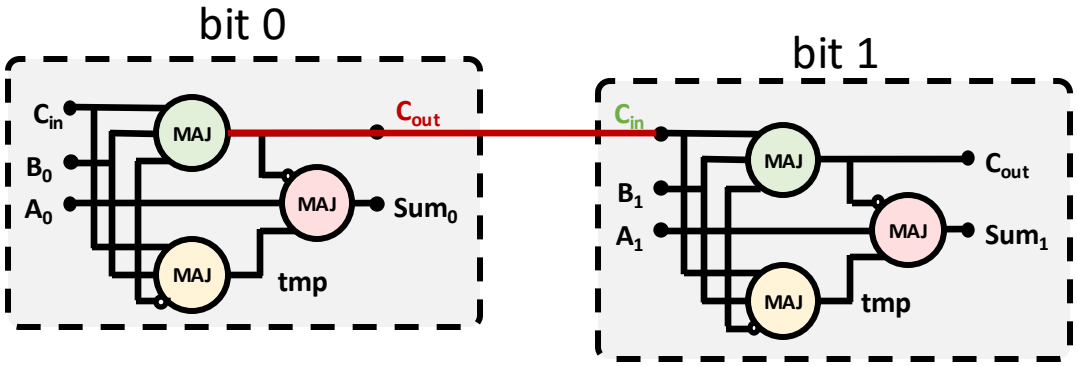




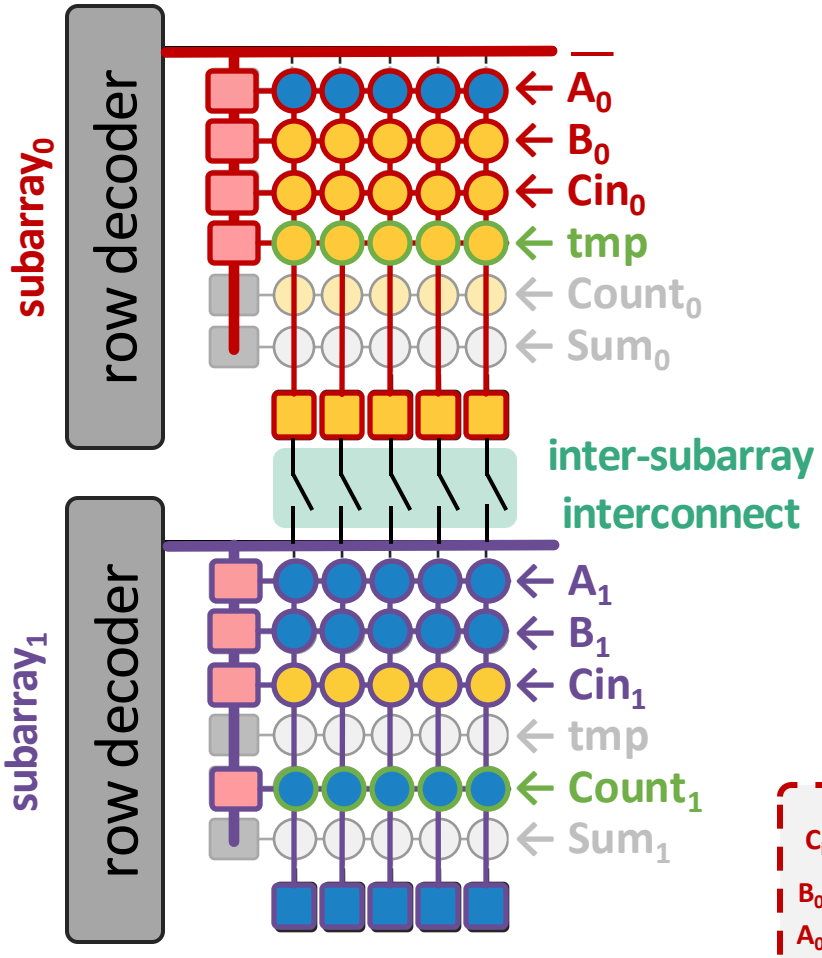
Opportunities for Bit-Serial PUD:
Latency Reduction with Subarray Parallelism

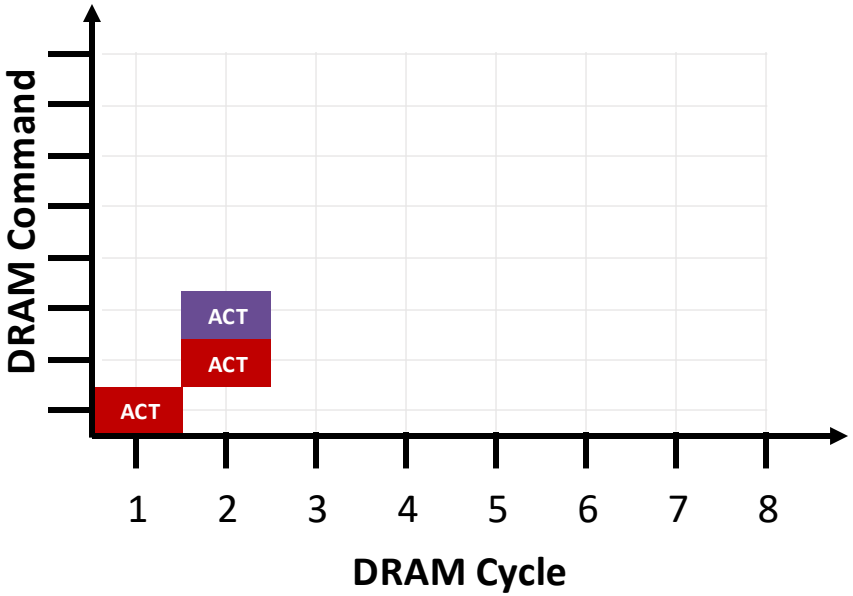


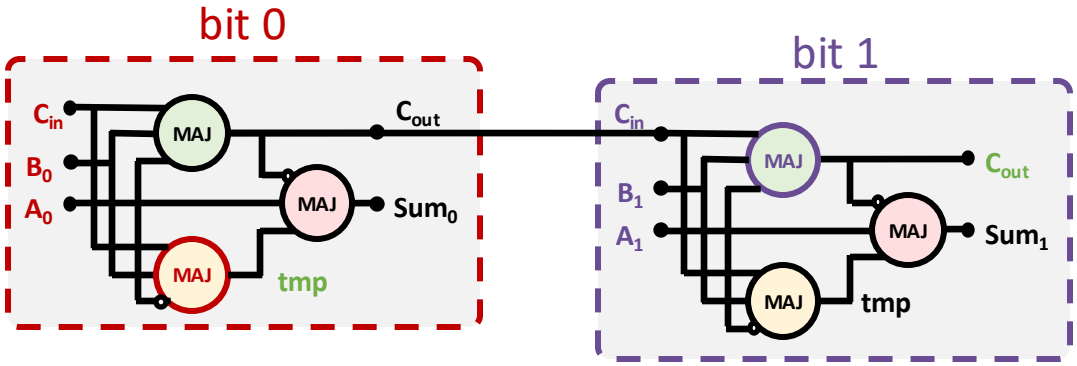




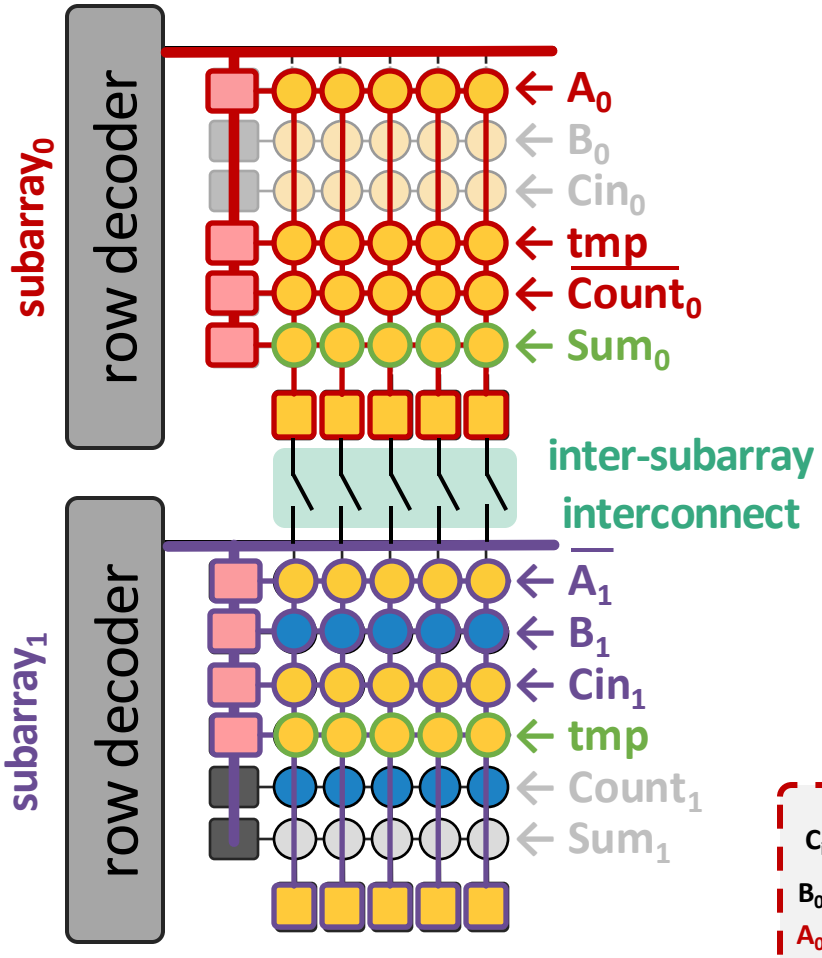
Opportunities for Bit-Serial PUD:
Latency Reduction with Subarray Parallelism

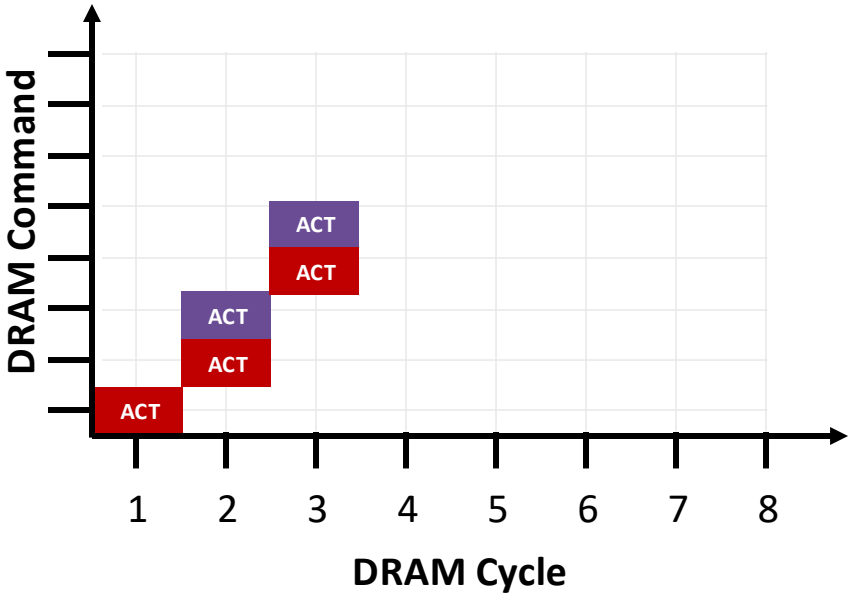


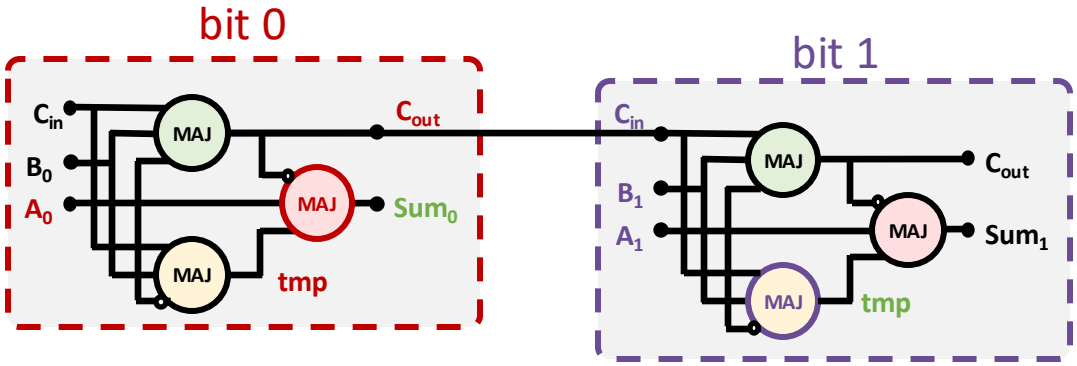




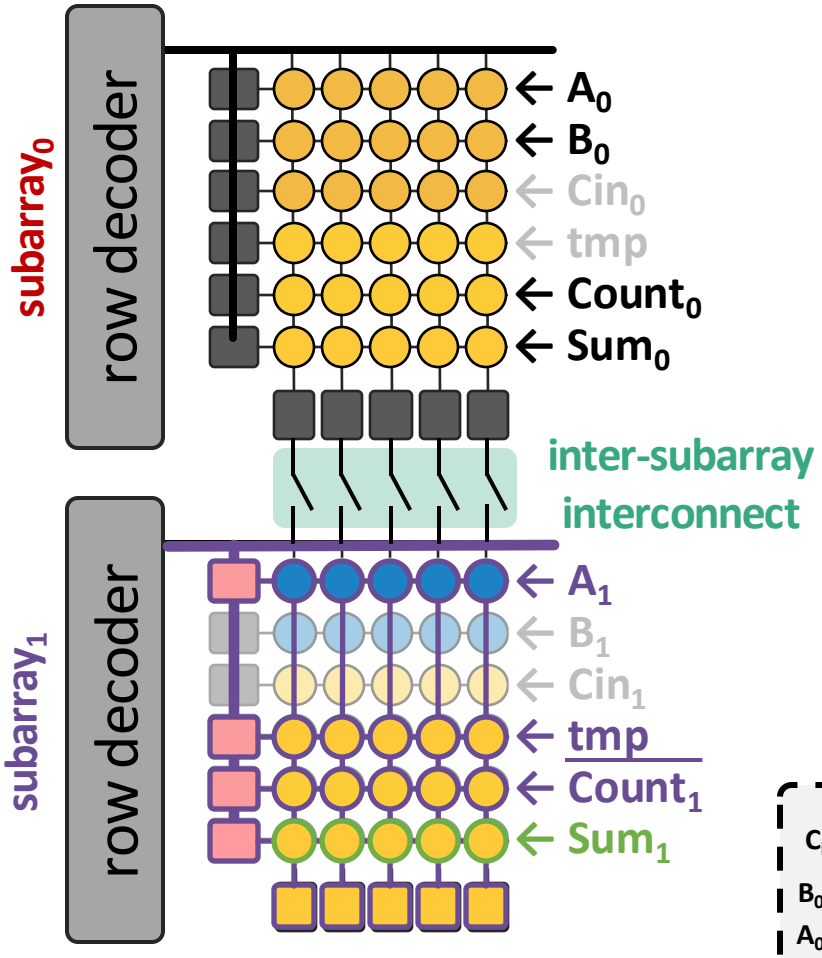
Opportunities for Bit-Serial PUD:
Latency Reduction with Subarray Parallelism

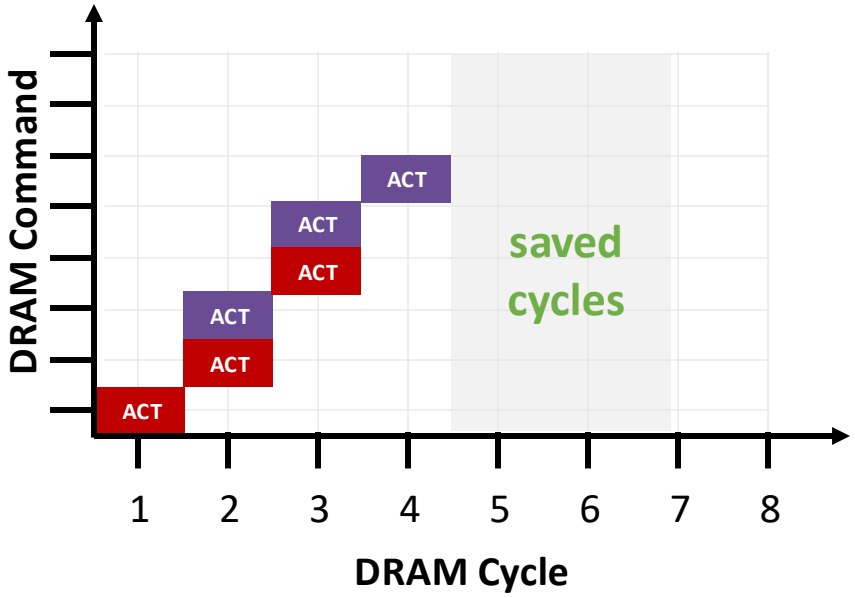


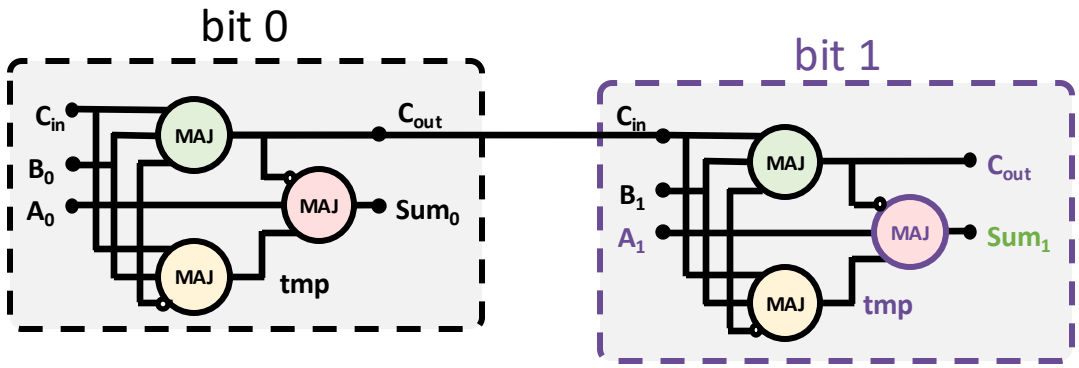




Opportunities for Bit-Serial PUD:
Latency Reduction with Subarray Parallelism







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

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

Key Idea

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

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

Outline

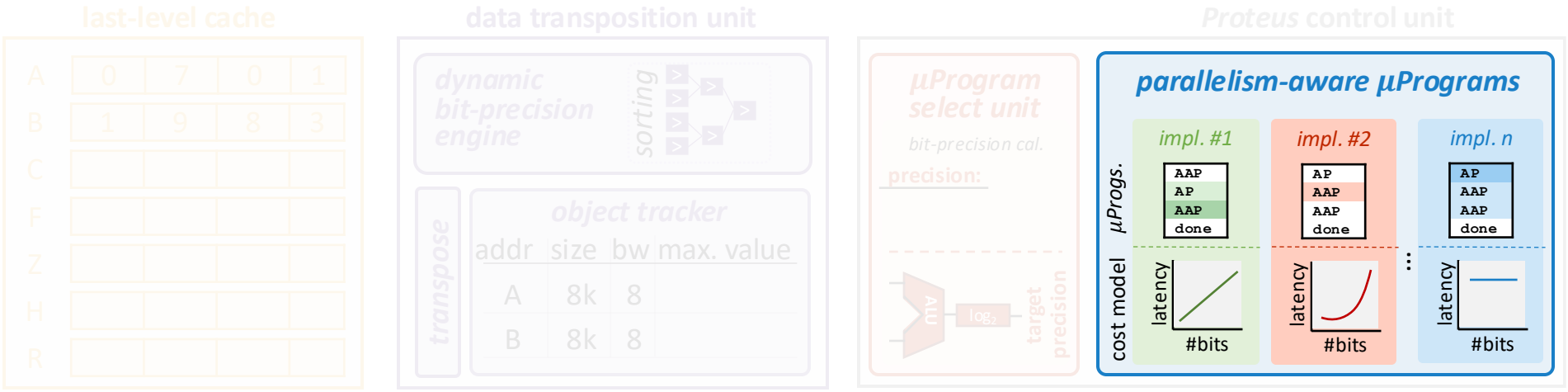
Proteus:
Design Overview (I)

Proteus, a data-aware runtime mechanism 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**

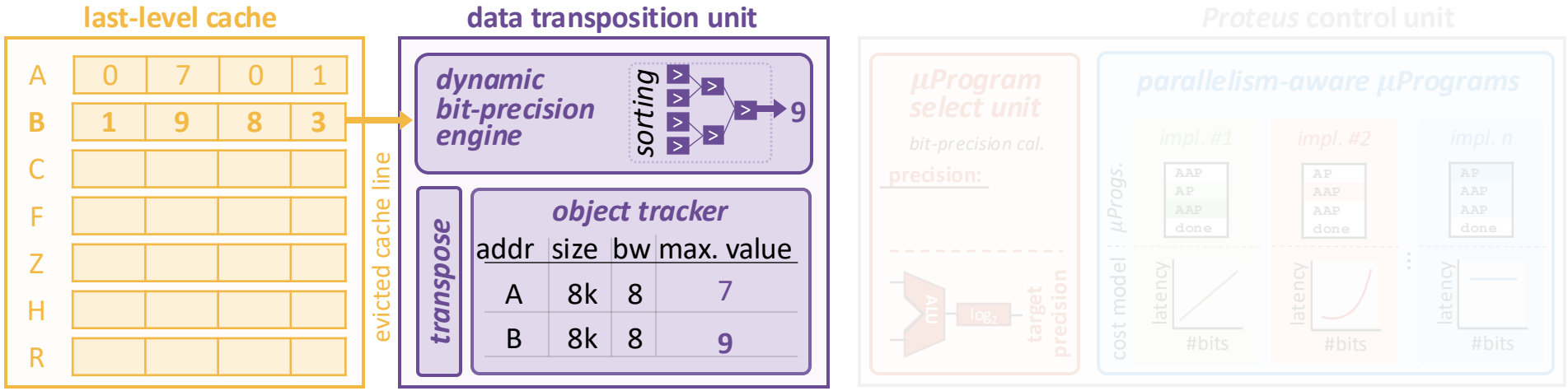
Proteus:
Design Overview (II)



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

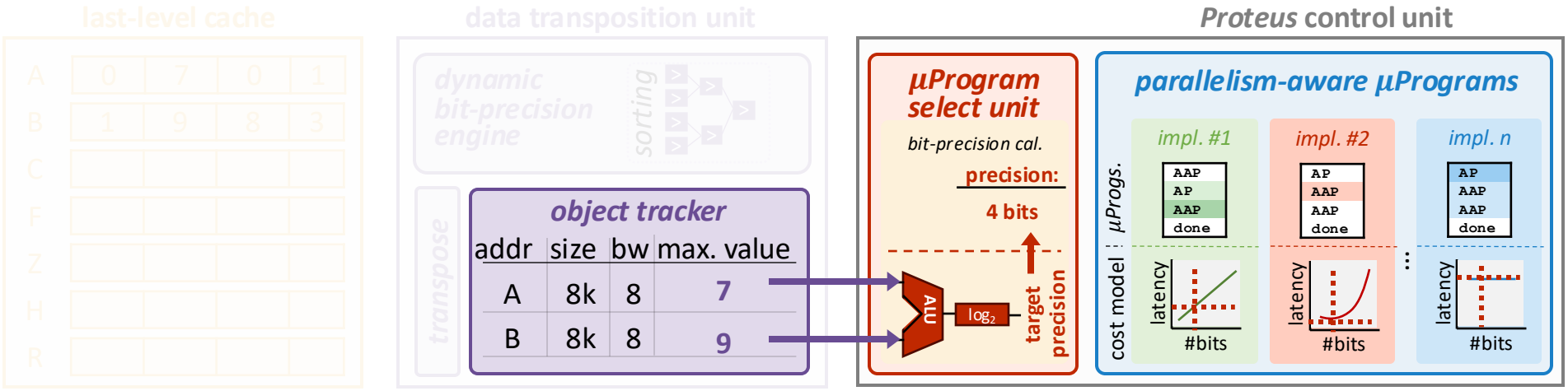
hand-optimized implementations of different PUD operations with different performance vs. bit-precision trade-offs

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

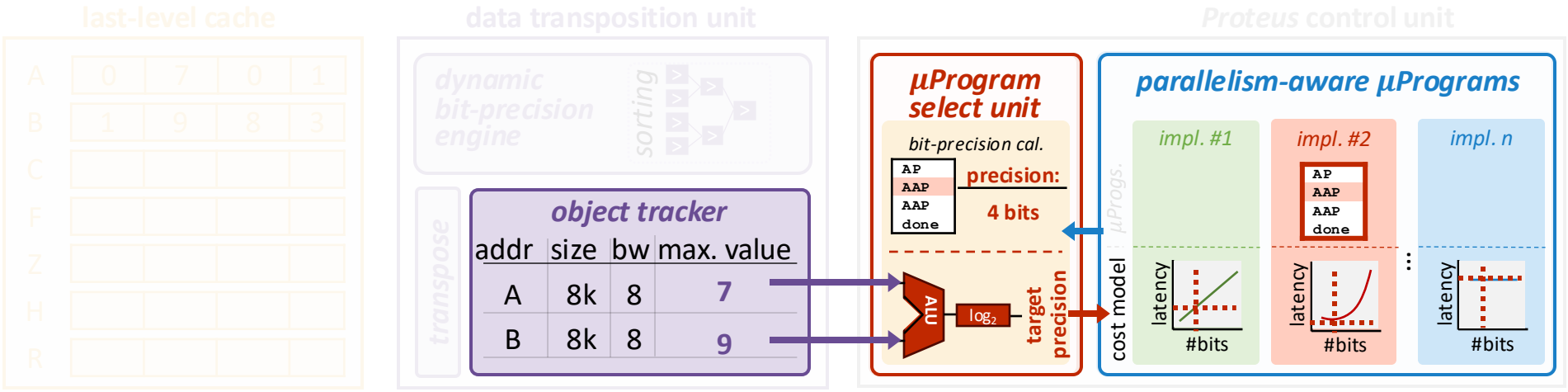
2 Dynamic Bit-Precision Engine

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

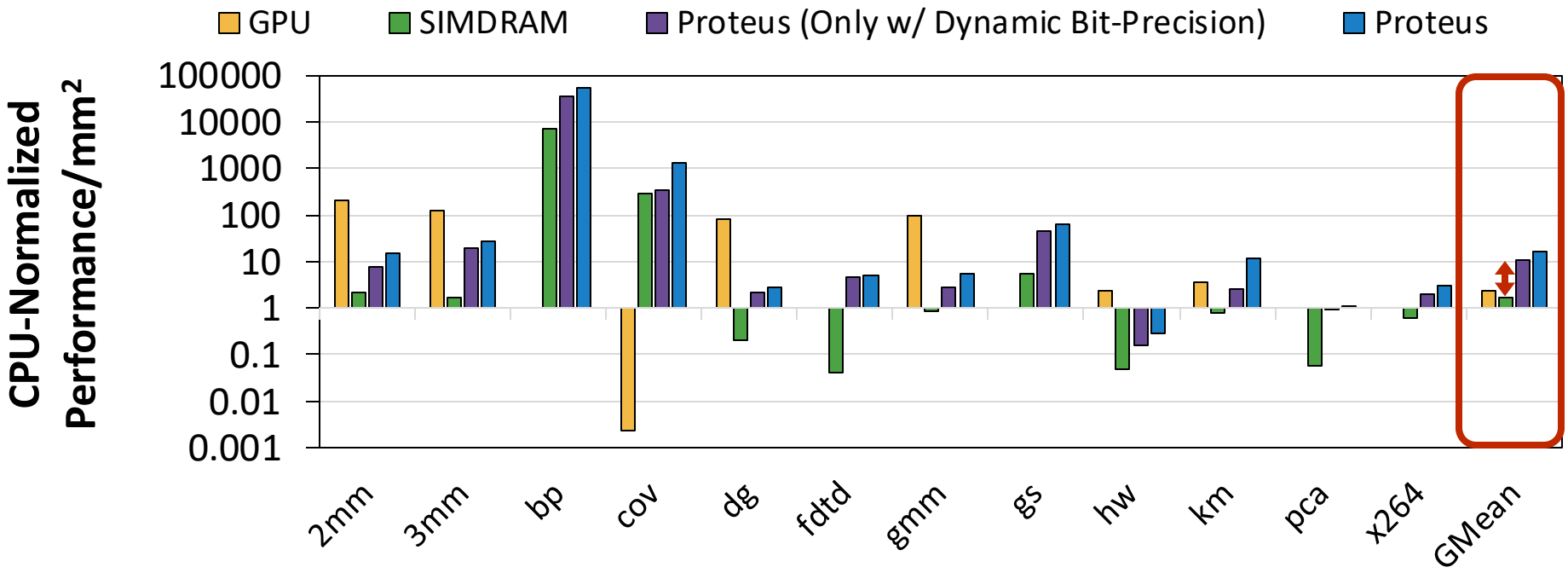
identifies the appropriate bit-precision based on the input operations of the target PUD operation

Outline

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)

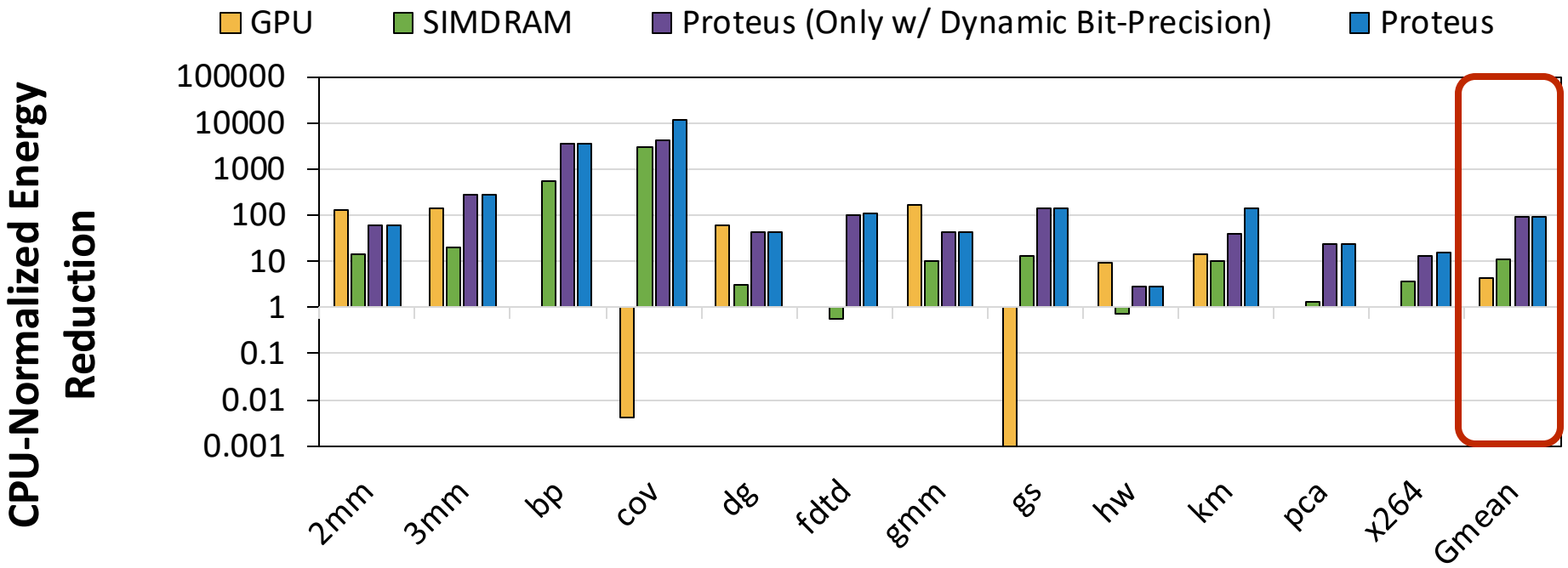
Evaluation:
Performance Analysis



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)**

Outline

Conclusion

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

that dynamically adjusts the bit-precision and, based on that, 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 mm^2 (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

2025 International Conference on Supercomputing (ICS)

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 UNIVERSITY OF CAMBRIDGE UNIVERSIDAD Đ CÓRDOBA UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN NVIDIA