

Clutch

High Performance Vector-Scalar Comparison using DRAM via Chunked Temporal Coding

Daichi Tokuda

T. Kubo I. E. Yuksel A. Olgun H. Luo

T. Nagatani G. F. Oliveira A. G. Yağlıkçı M. Sadrosadati

O. Mutlu S. Takamaeda-Yamazaki



Executive Summary

Motivation: *Vector-scalar comparison* is a fundamental primitive in data-intensive workloads

- *Processing-using-DRAM (PuD)* is a promising approach to alleviate the *memory-bound bottleneck*, as it performs computations directly within the DRAM array

Problem: Existing PuD approaches *require many PuD operations* for comparison due to *the bit-serial execution model*, limiting end-to-end application-level speedup

Key Mechanism: Clutch introduces a new data representation and algorithm

- *Temporal coding* encodes values so that a comparison reduces to a *single lookup*, dramatically reducing PuD operation count
- *Divide-and-conquer approach* decomposes operands into smaller chunks, enabling a *flexible throughput-footprint tradeoff*

Key Results: Clutch achieves *12x higher throughput* and *69x higher energy efficiency* over CPU, and *2.9x/3.0x* over the bit-serial PuD

Outline

Background

Motivation: PuD for Vector-Scalar Comparison

Limitation of Prior PuD Approaches

Clutch: Lookup Table-Based Comparison

Clutch: Divide-and-Conquer Approach

Evaluation

Conclusion

Outline

Background

Motivation: PuD for Vector-Scalar Comparison

Limitation of Prior PuD Approaches

Clutch: Lookup Table-Based Comparison

Clutch: Divide-and-Conquer Approach

Evaluation

Conclusion

Vector-Scalar Comparison

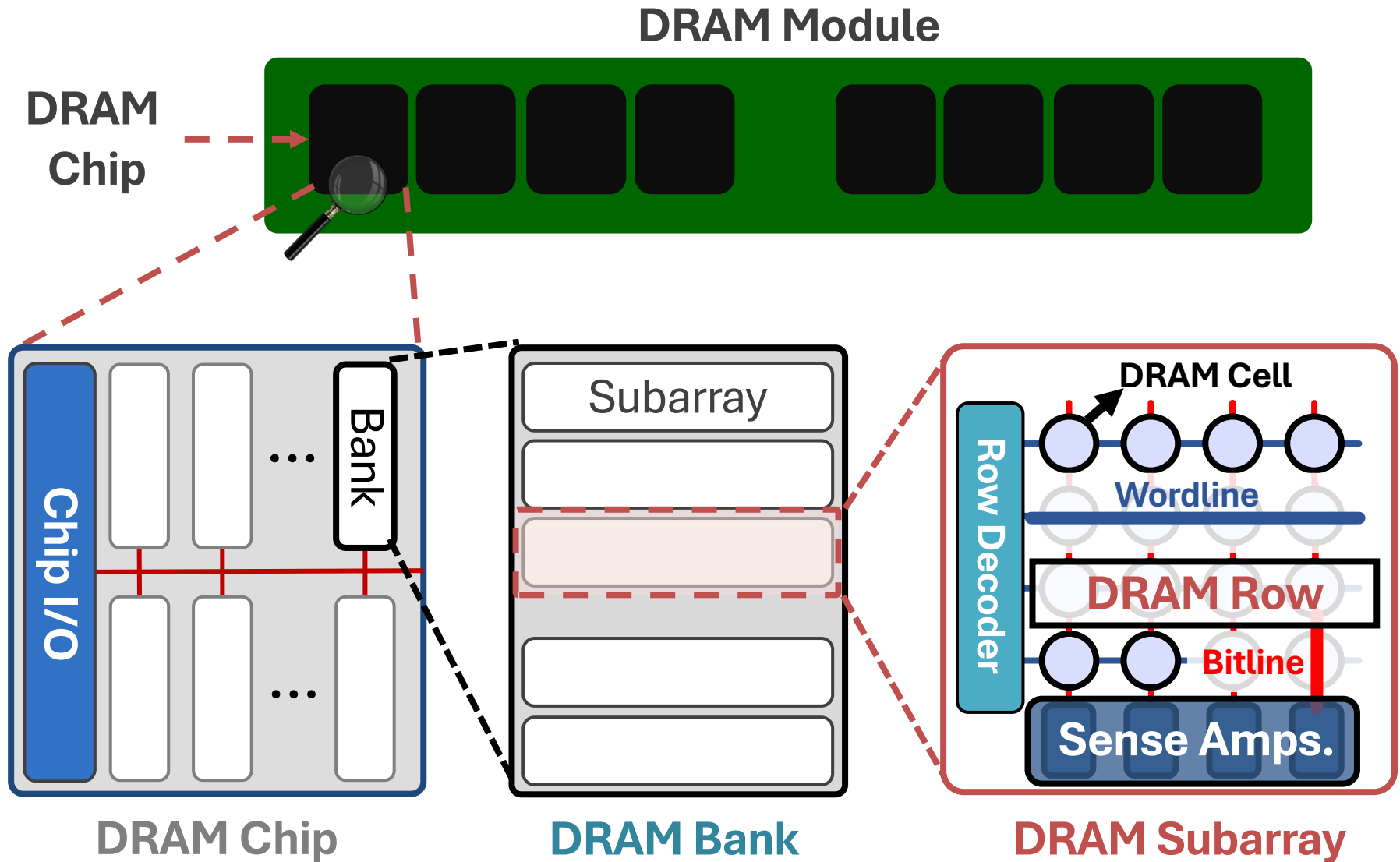
- **Vector-Scalar Comparison:** compares each element in a vector against a single scalar value
 - A fundamental primitive in data-intensive workloads, including predicate evaluation and decision tree inference
 - Due to its low computational intensity, its execution tends to be memory-bound

Vector-Scalar Comparison

```
for  $i = 0$  to  $\text{len}(B) - 1$  :
```

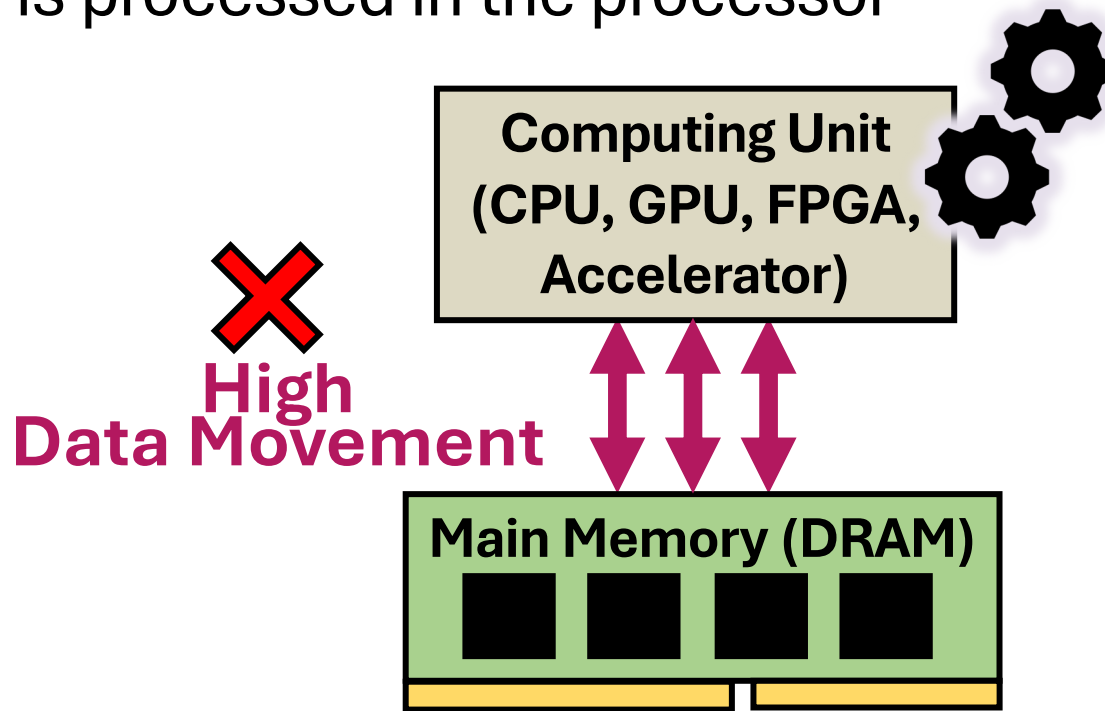
$$\underbrace{Y_i}_{\text{1-bit}} \leftarrow \underbrace{a}_{\text{n-bit}} < \underbrace{B_i}_{\text{n-bit}}$$

DRAM Organization



Data Movement Bottleneck

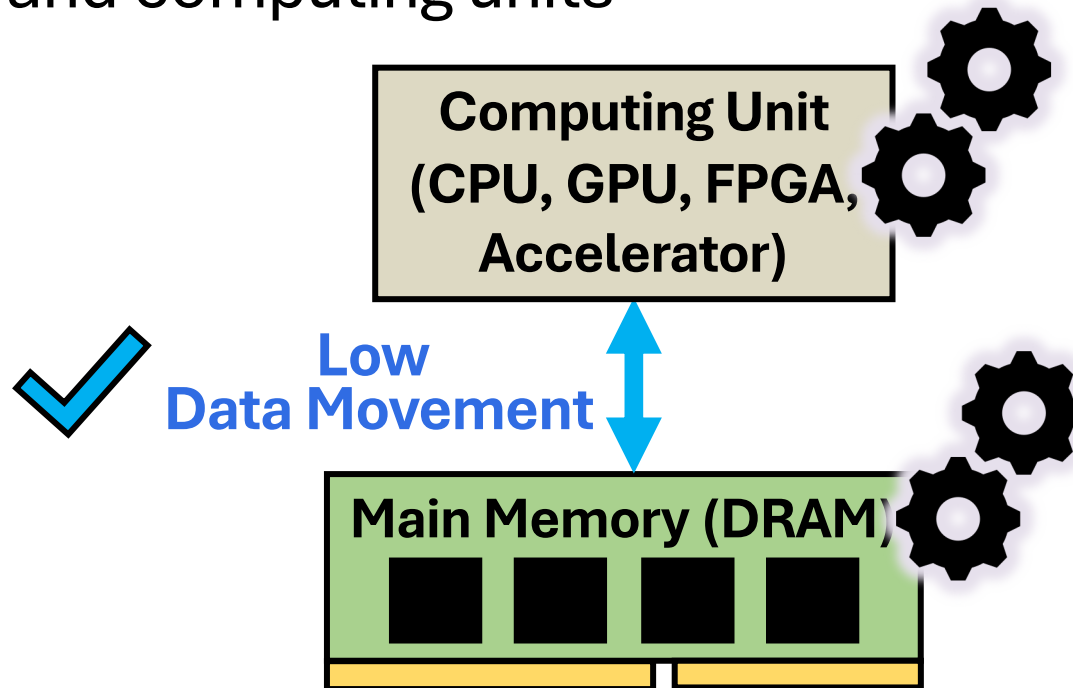
- Today's computing systems are **processor-centric**
- All data is processed in the processor



Low-computational-intensity operations are bottlenecked by **data movement**

Processing-using-DRAM (PuD)

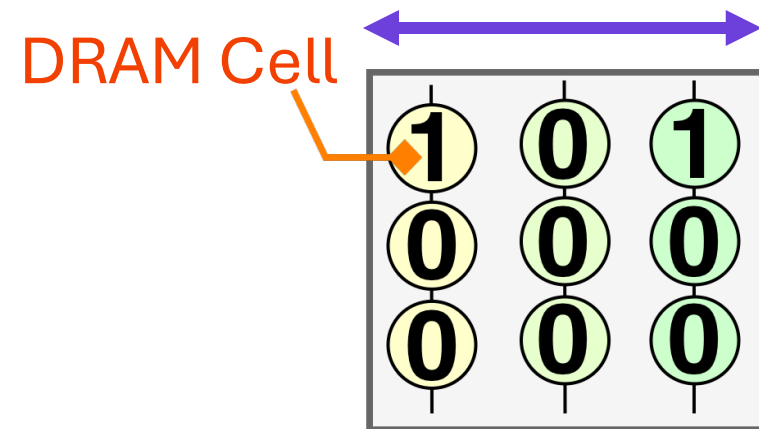
- Processing-using-DRAM (PuD) uses the **analog operational principles of DRAM cells** to perform computation
- PuD **alleviates the frequent data movement** between main memory and computing units



PuD Primitives: RowCopy

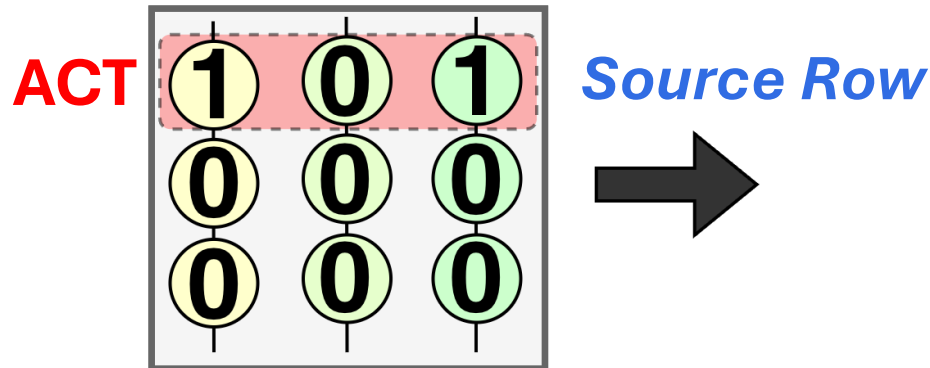
- **RowCopy**: transfers data **between rows within the same subarray** by activating two rows in quick succession, enabling **efficient bulk data movement inside DRAM**

Columns (e.g. 64K columns / bank)



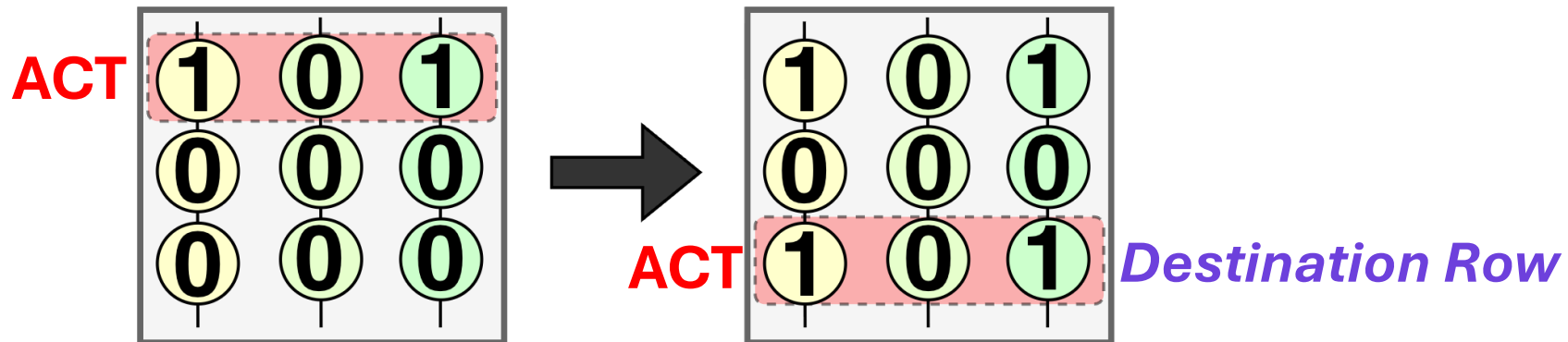
PuD Primitives: RowCopy

- **RowCopy**: transfers data **between rows within the same subarray** by activating two rows in quick succession, enabling **efficient bulk data movement inside DRAM**



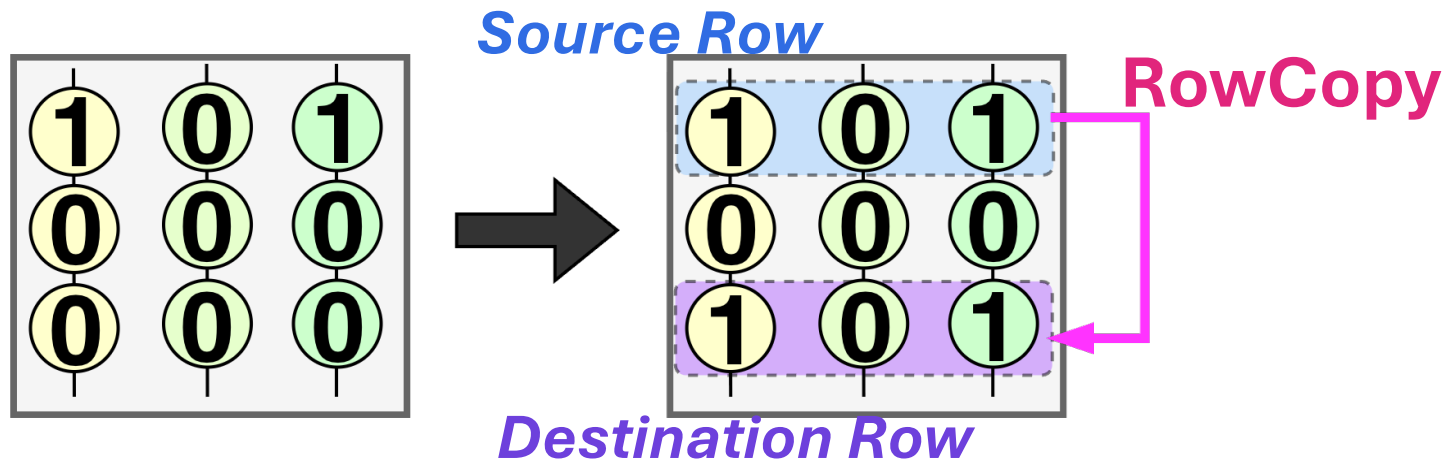
PuD Primitives: RowCopy

- **RowCopy:** transfers data **between rows within the same subarray** by activating two rows in quick succession, enabling efficient bulk data movement inside DRAM



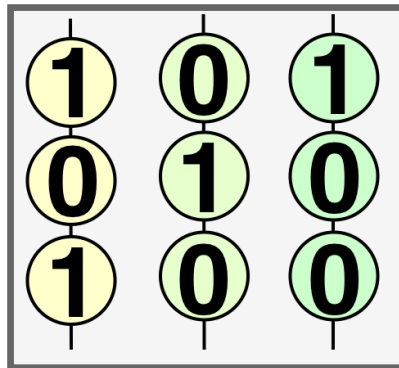
PuD Primitives: RowCopy

- **RowCopy**: transfers data **between rows within the same subarray** by activating two rows in quick succession, enabling **efficient bulk data movement inside DRAM**



PuD Primitives: MAJ3

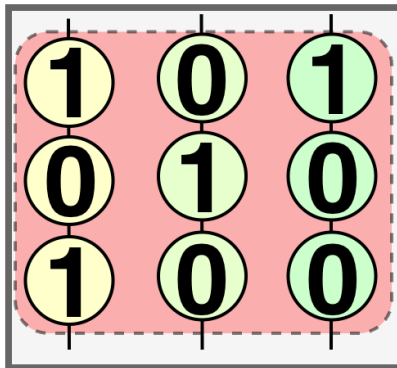
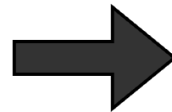
- **MAJ3**: activates **three rows simultaneously**, causing charge sharing on the bitline, such that each column independently computes **the majority of the activated cells in that column**



PuD Primitives: MAJ3

- **MAJ3**: activates **three rows simultaneously**, causing charge sharing on the bitline, such that each column independently computes **the majority of the activated cells in that column**

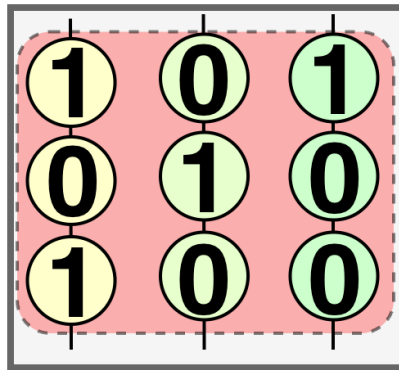
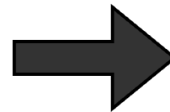
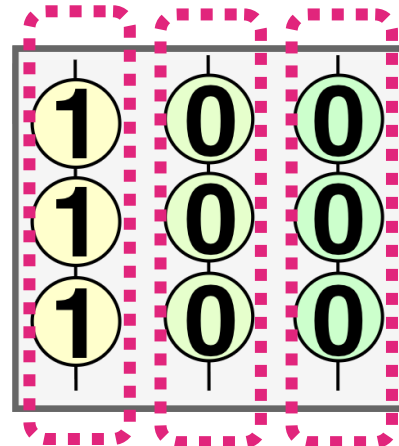
**Simultaneously
activating
three rows**



PuD Primitives: MAJ3

- **MAJ3**: activates **three rows simultaneously**, causing charge sharing on the bitline, such that each column independently computes **the majority of the activated cells in that column**

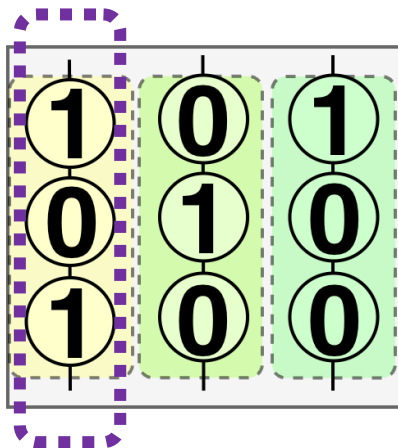
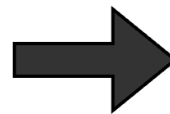
**Simultaneously
activating
three rows**



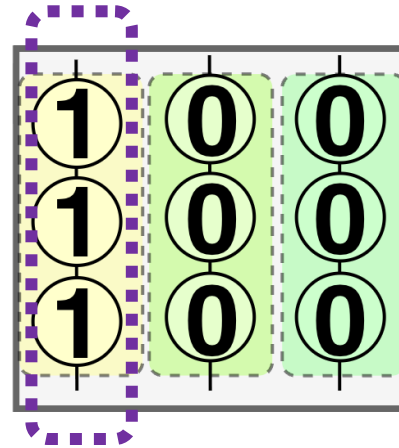
PuD Primitives: MAJ3

- **MAJ3**: activates **three rows simultaneously**, causing charge sharing on the bitline, such that each column independently computes **the majority of the activated cells in that column**

MAJ3(1, 0, 1)

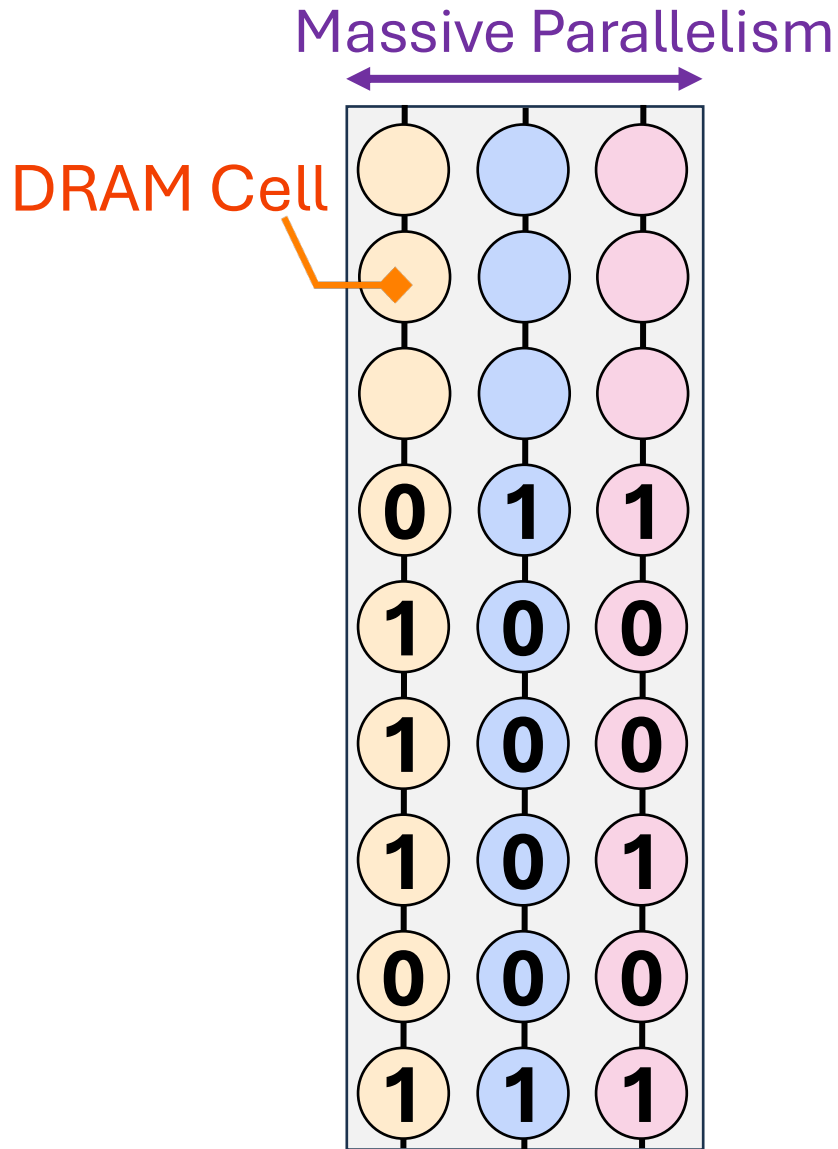


= 1

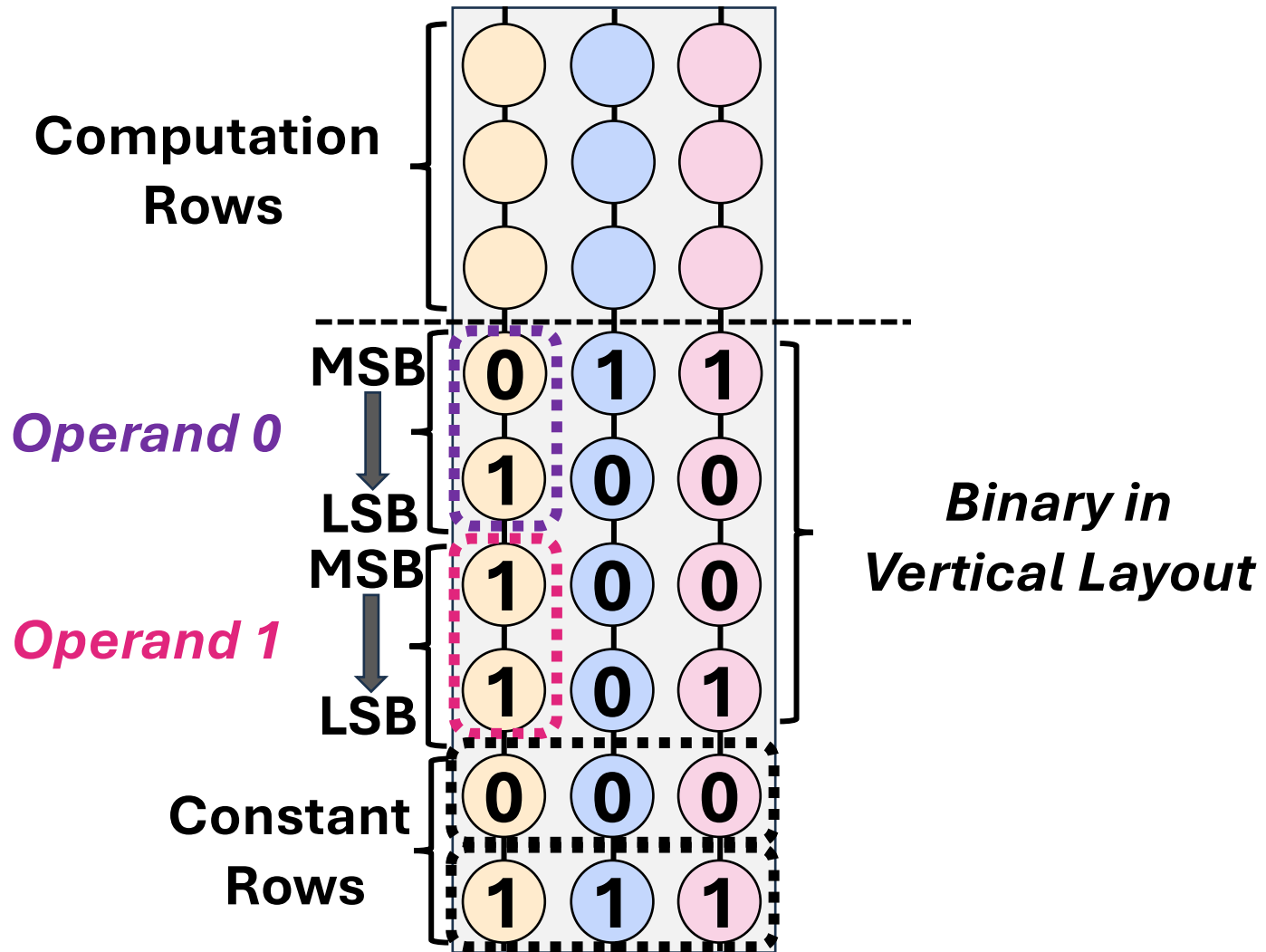


Each column
performs
MAJ3

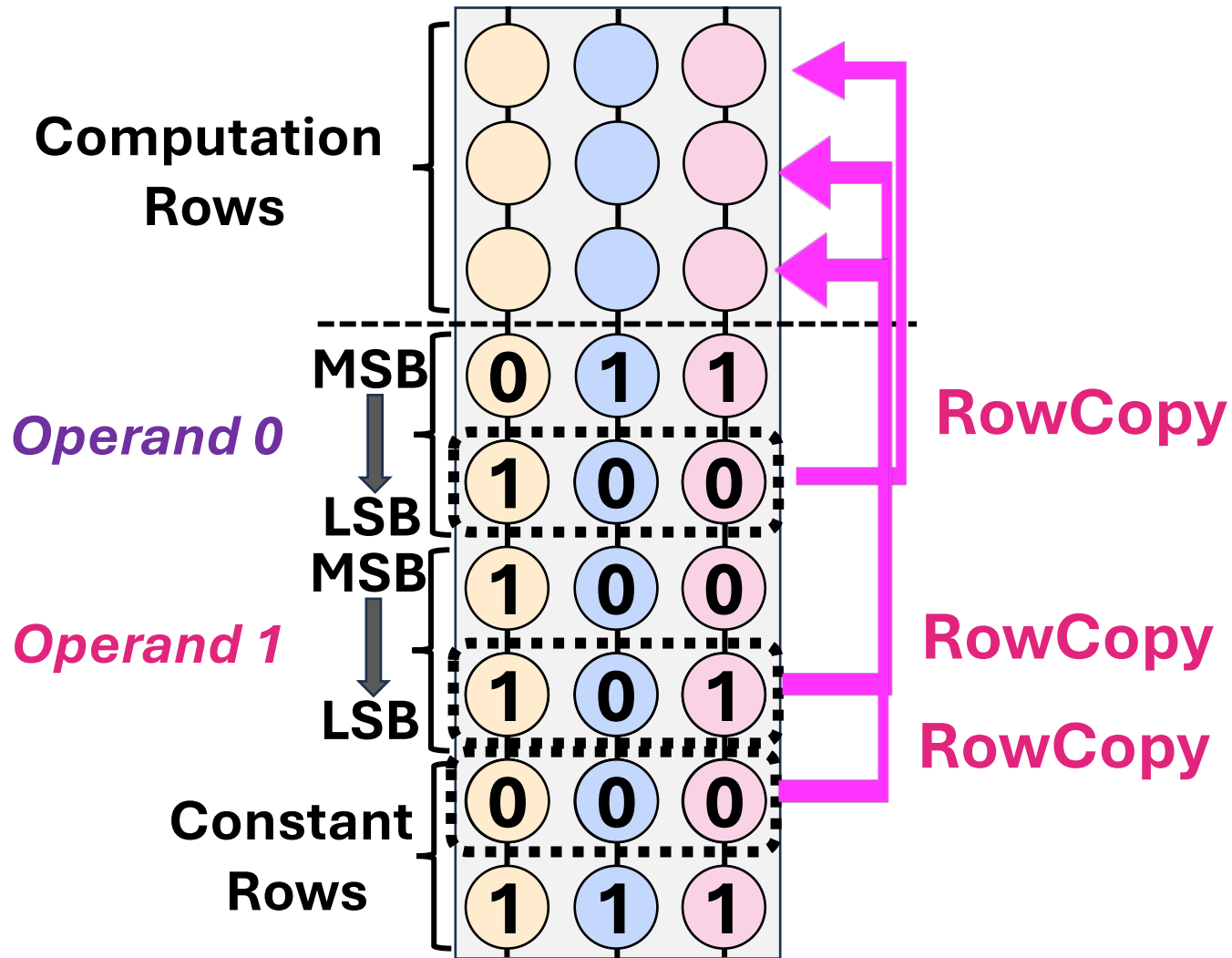
Prior PuD Arithmetic: Bit-Serial Execution



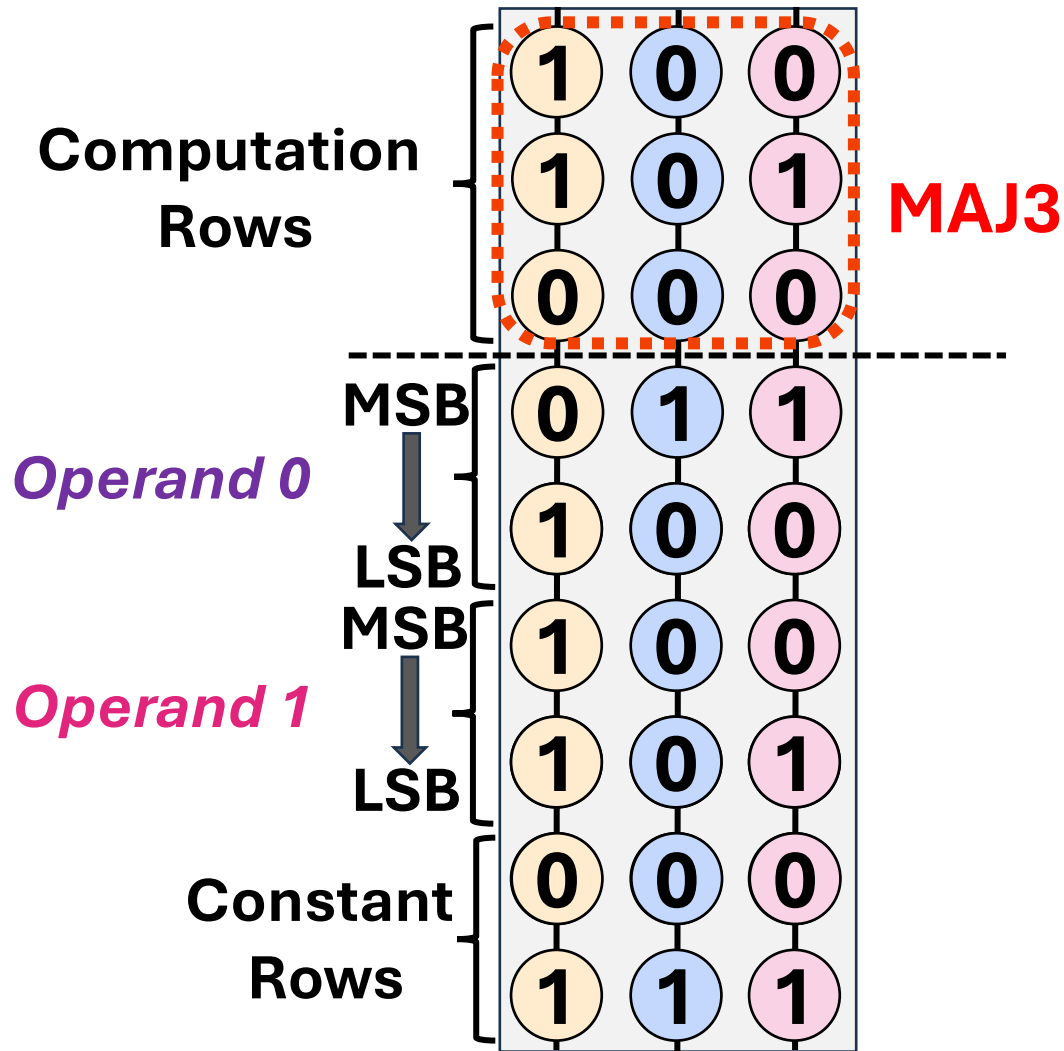
Prior PuD Arithmetic: Bit-Serial Execution



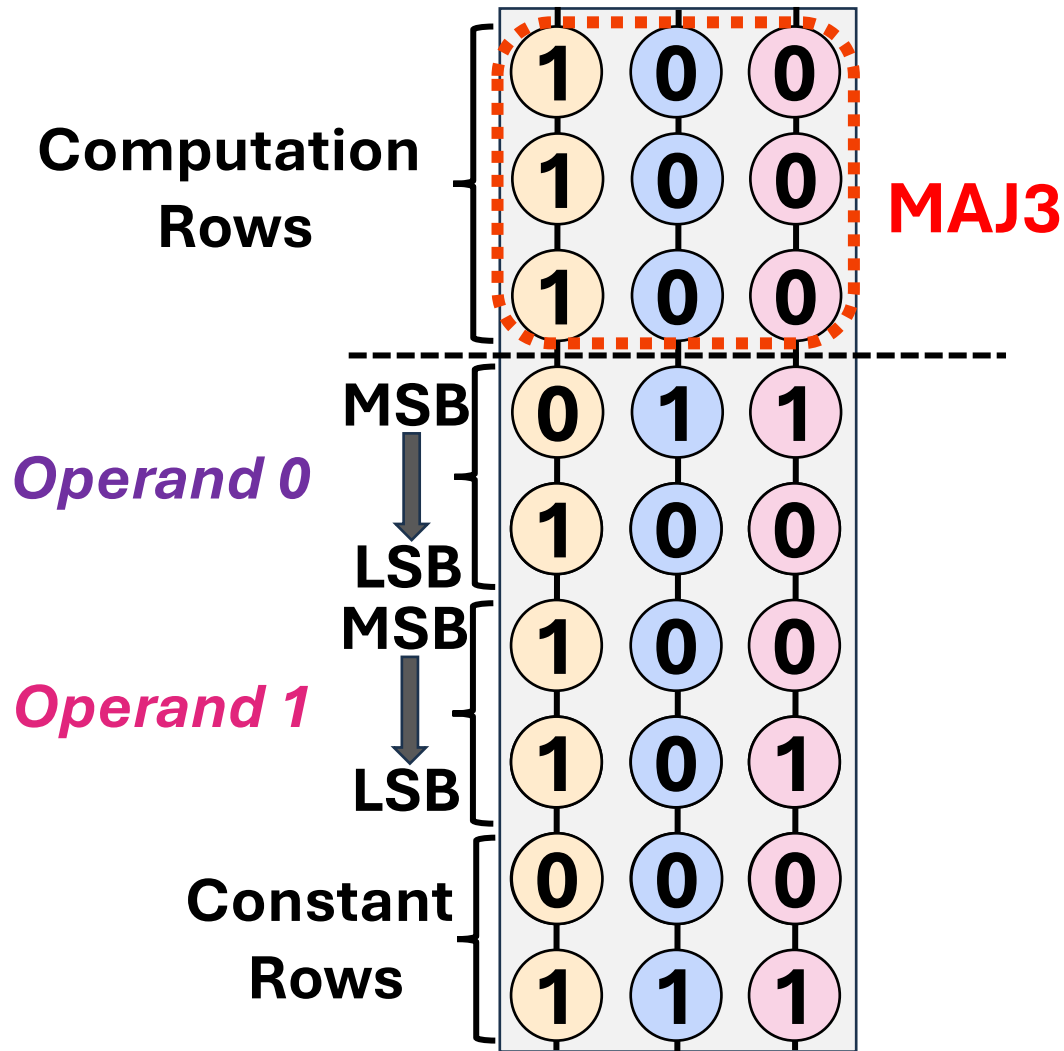
Prior PuD Arithmetic: Bit-Serial Execution



Prior PuD Arithmetic: Bit-Serial Execution



Prior PuD Arithmetic: Bit-Serial Execution



Outline

Background

Motivation: PuD for Vector-Scalar Comparison

Limitation of Prior PuD Approaches

Clutch: Lookup Table-Based Comparison

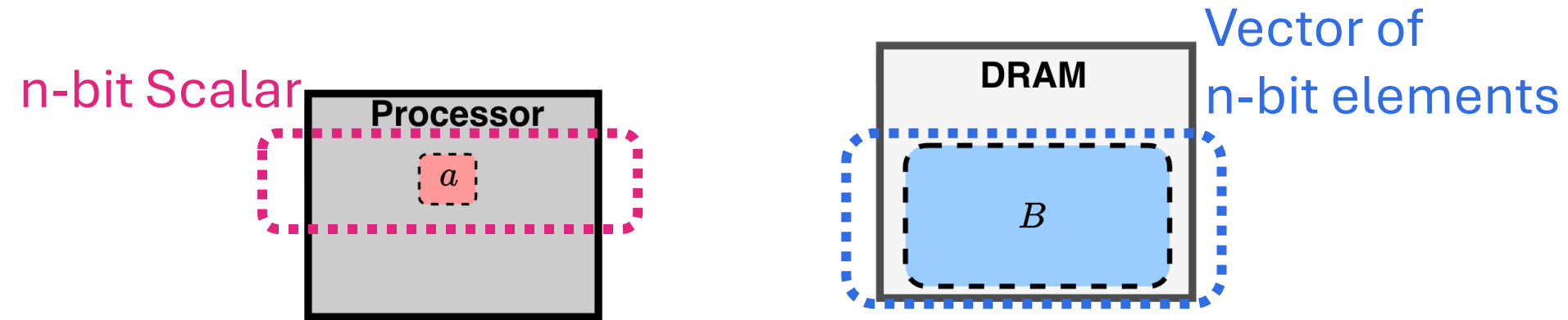
Clutch: Divide-and-Conquer Approach

Evaluation

Conclusion

Data Movement Reduction via PuD Execution

Processor-Centric Paradigm



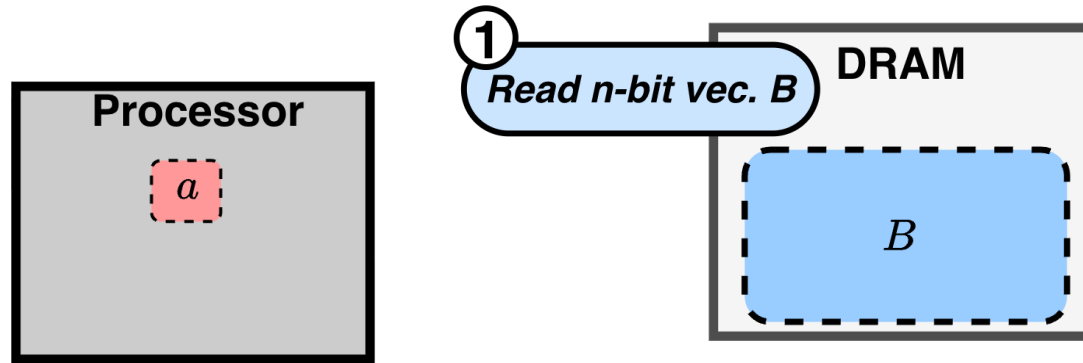
Vector-Scalar Comparison

```
for  $i = 0$  to  $\text{len}(B) - 1$ :
```

$$\underbrace{Y_i}_{\text{1-bit}} \leftarrow \underbrace{a}_{\text{n-bit}} < \underbrace{B_i}_{\text{n-bit}}$$

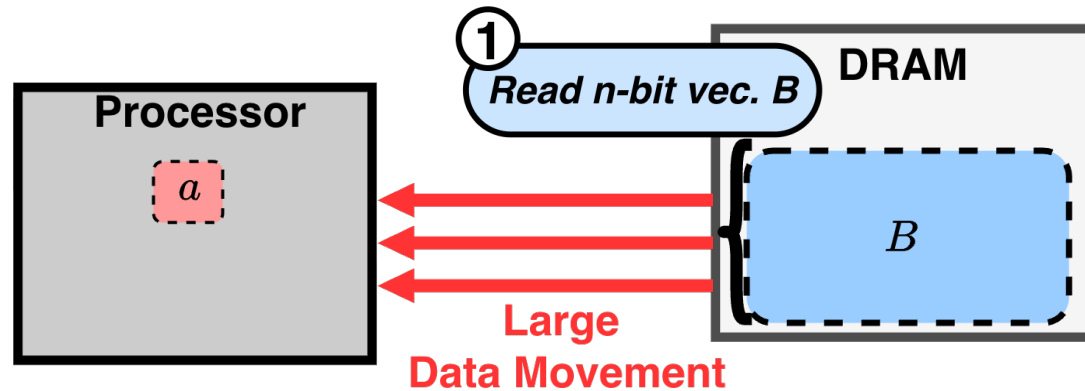
Data Movement Reduction via PuD Execution

Processor-Centric Paradigm



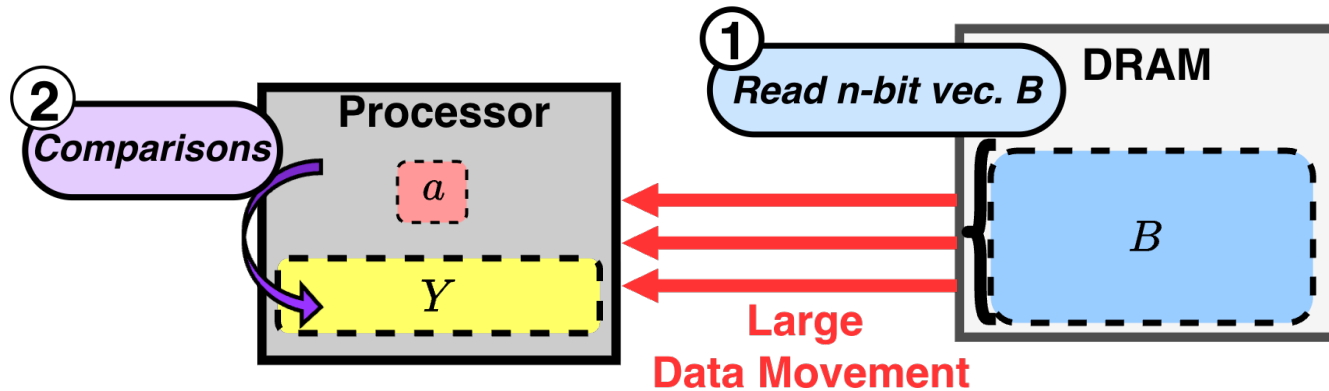
Data Movement Reduction via PuD Execution

Processor-Centric Paradigm



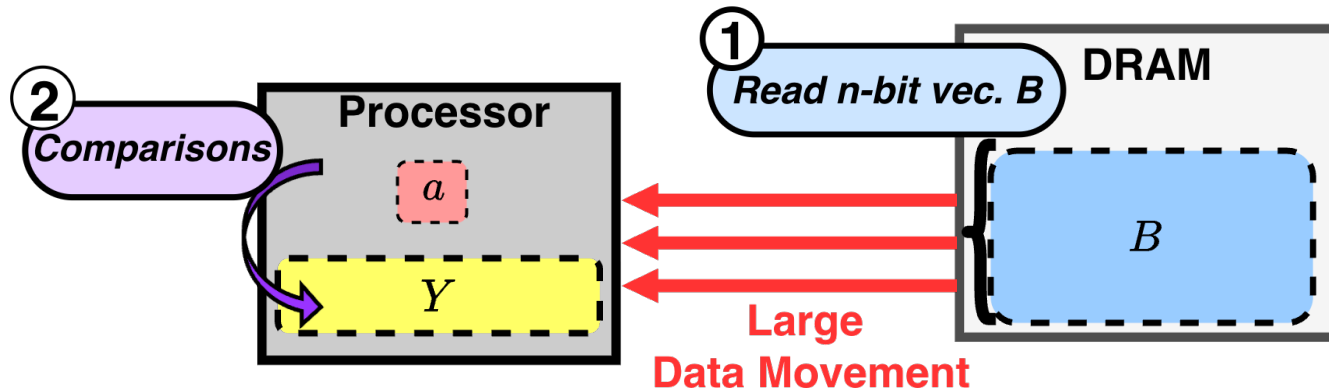
Data Movement Reduction via PuD Execution

Processor-Centric Paradigm



Data Movement Reduction via PuD Execution

Processor-Centric Paradigm



Processing-using-DRAM



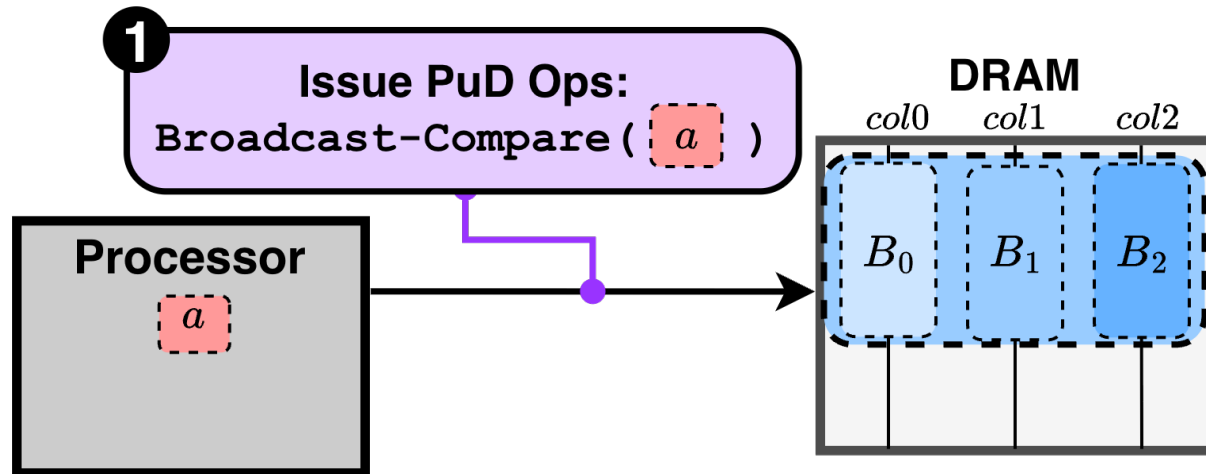
Data Movement Reduction via PuD Execution

1) The scalar value a is shared across all elements

2) Comparisons

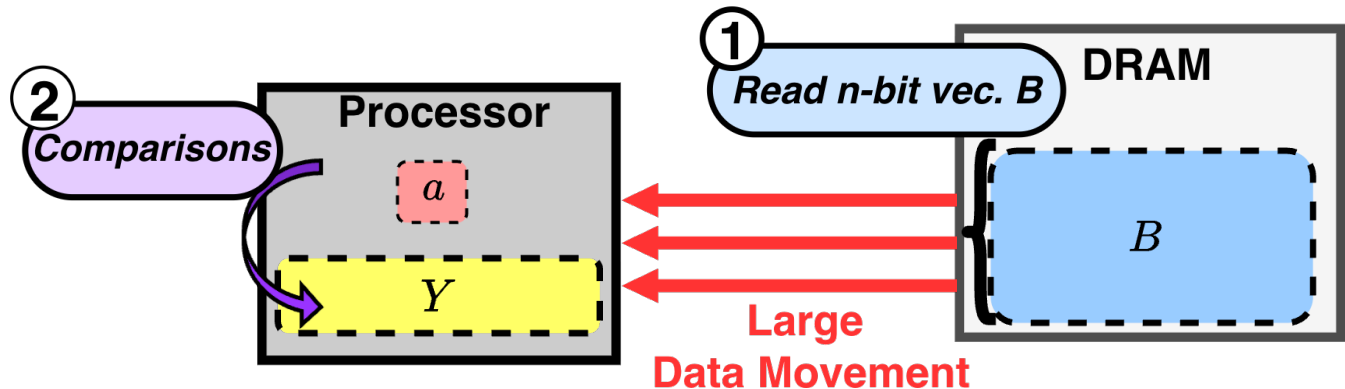
Large Data Movement

Processing-using-DRAM

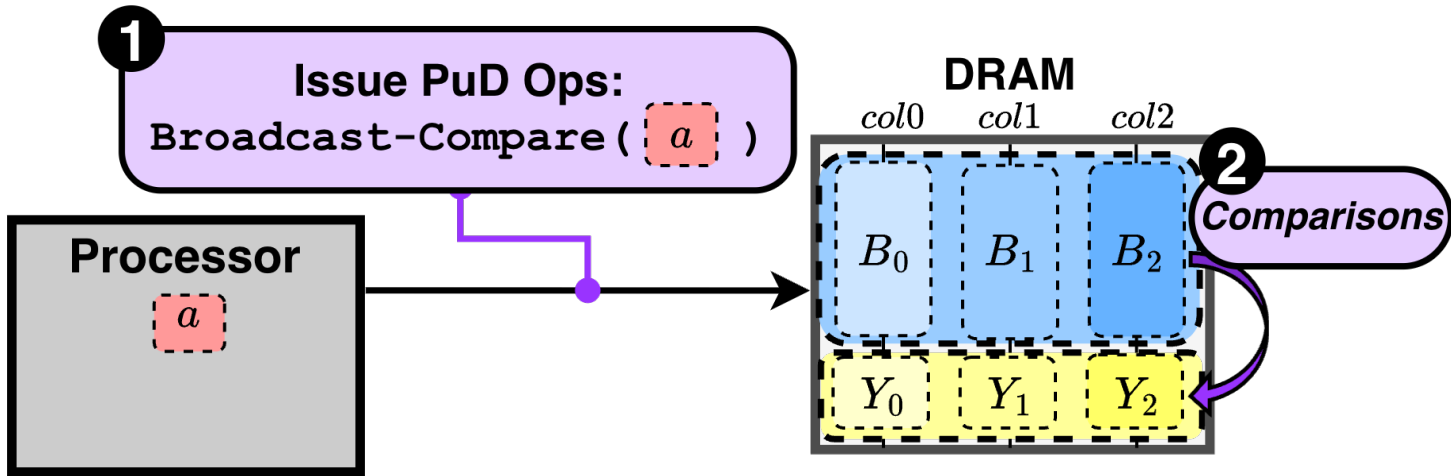


Data Movement Reduction via PuD Execution

Processor-Centric Paradigm

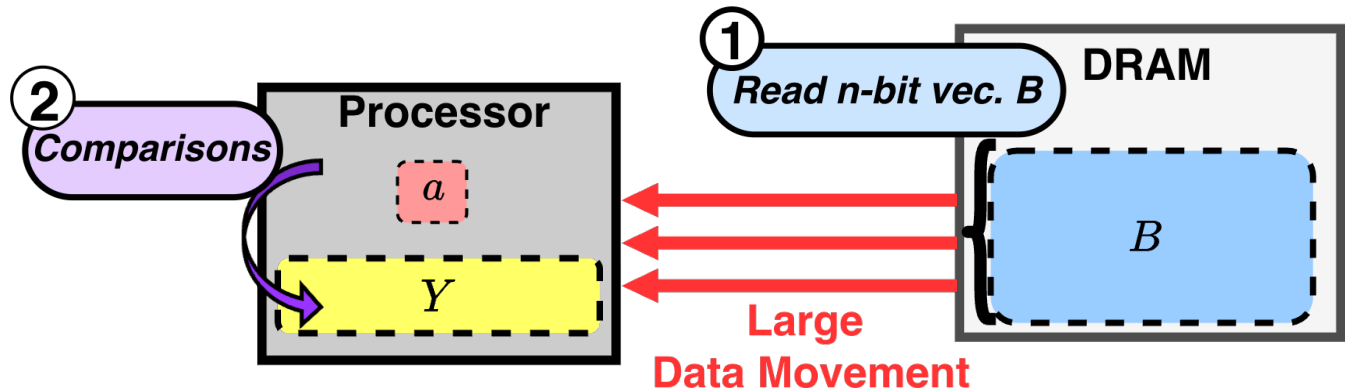


Processing-using-DRAM

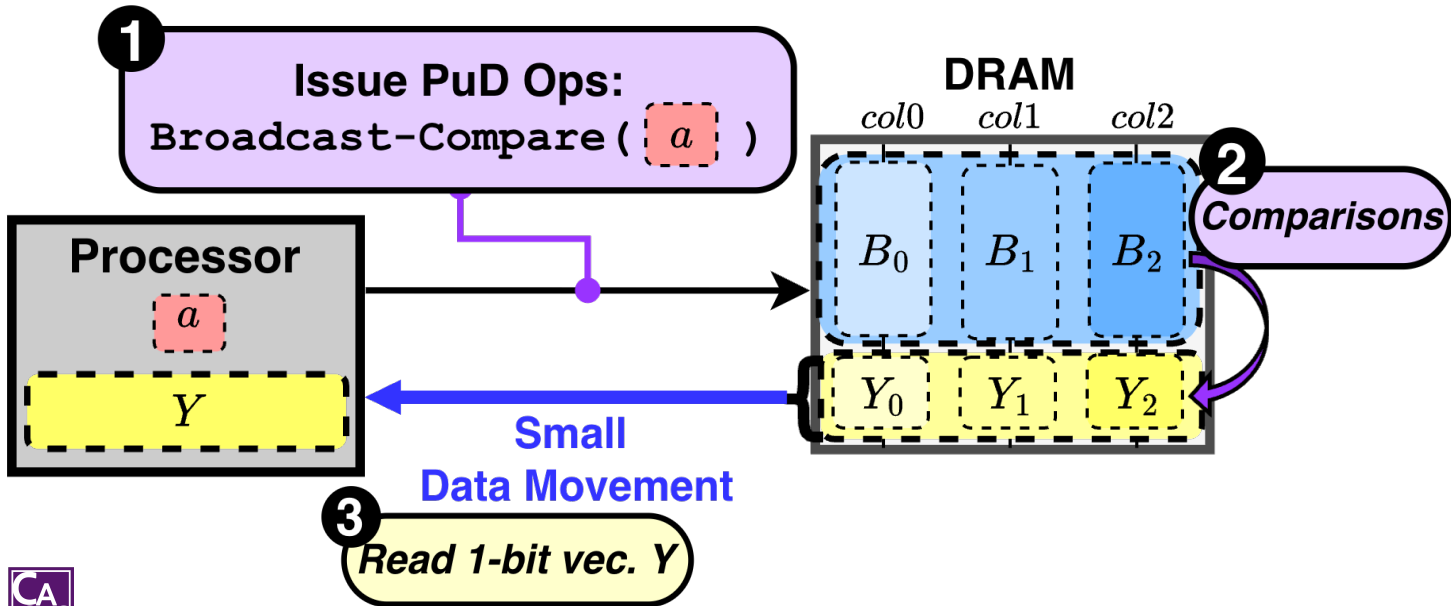


Data Movement Reduction via PuD Execution

Processor-Centric Paradigm

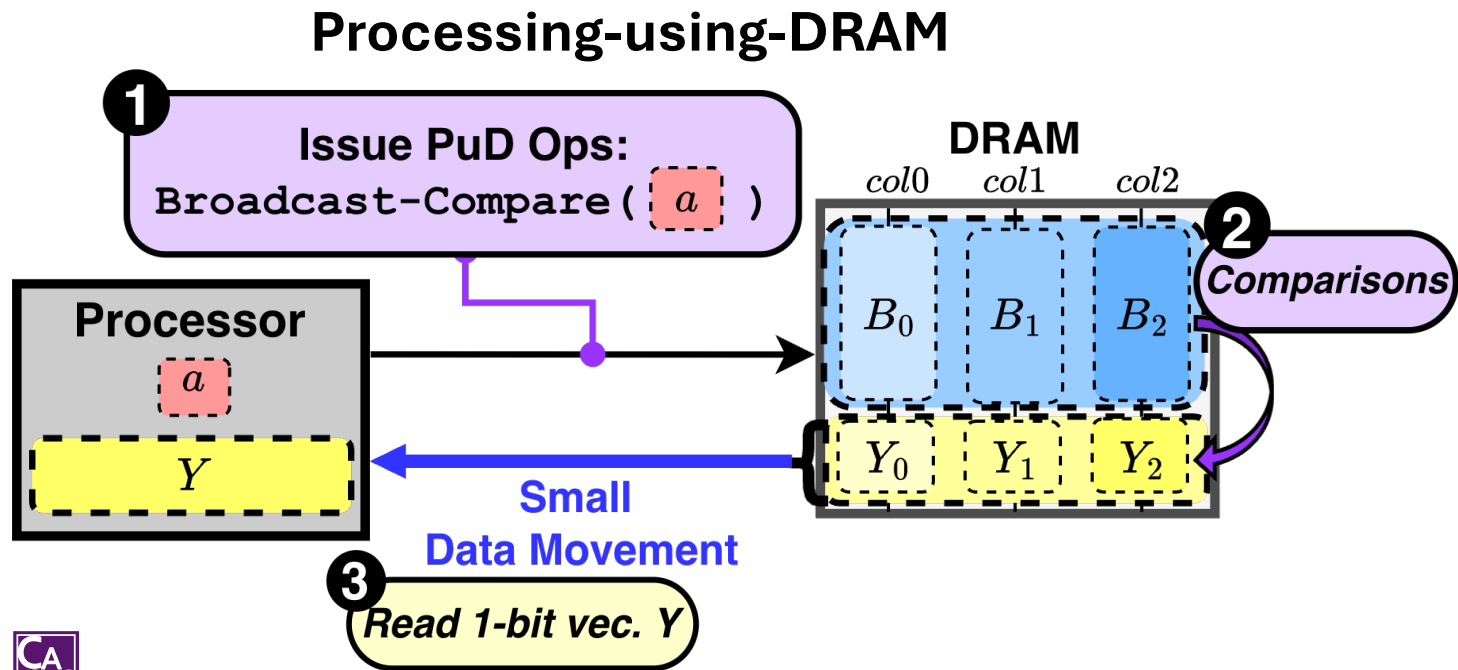


Processing-using-DRAM



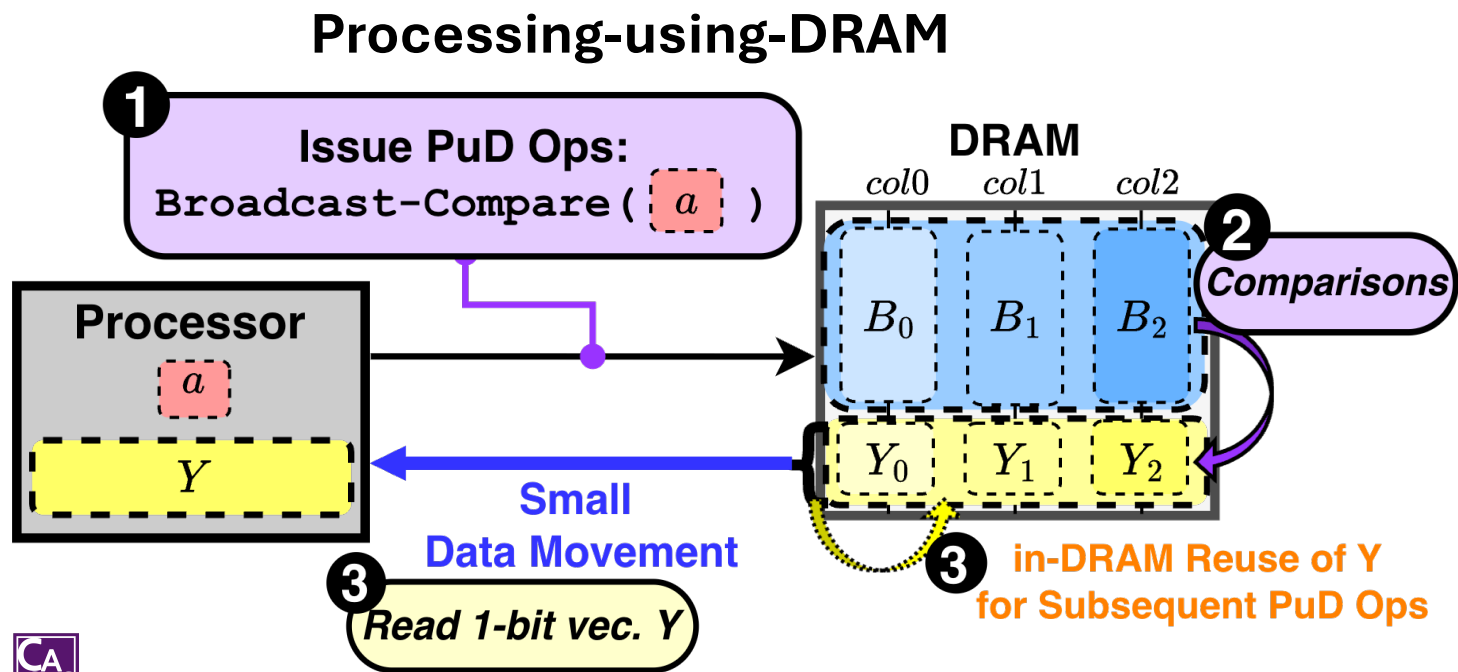
Data Movement Reduction via PuD Execution

2) The output is a **1-bit-per-element bitmap Y** , far smaller than the n -bit input vector B



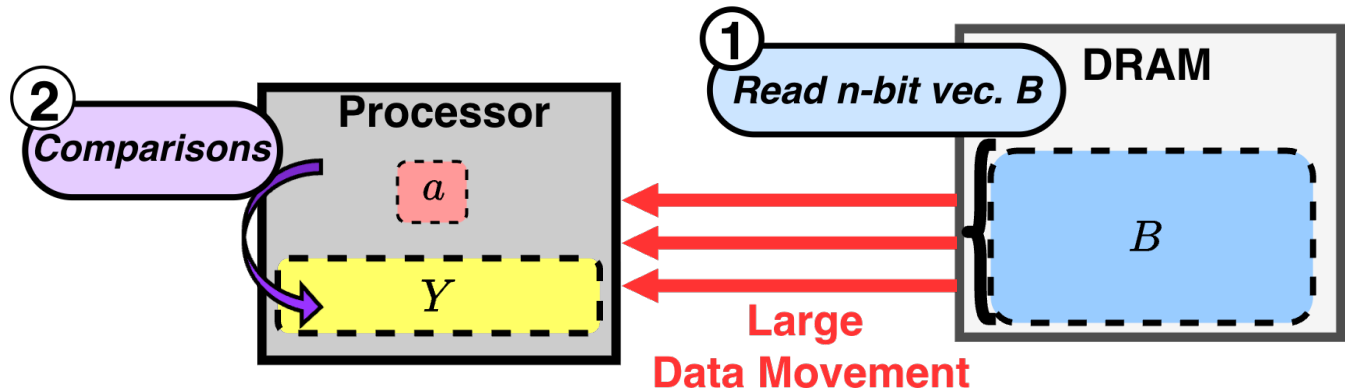
Data Movement Reduction via PuD Execution

3) The output bitmap is often combined with other bitmaps in downstream processing

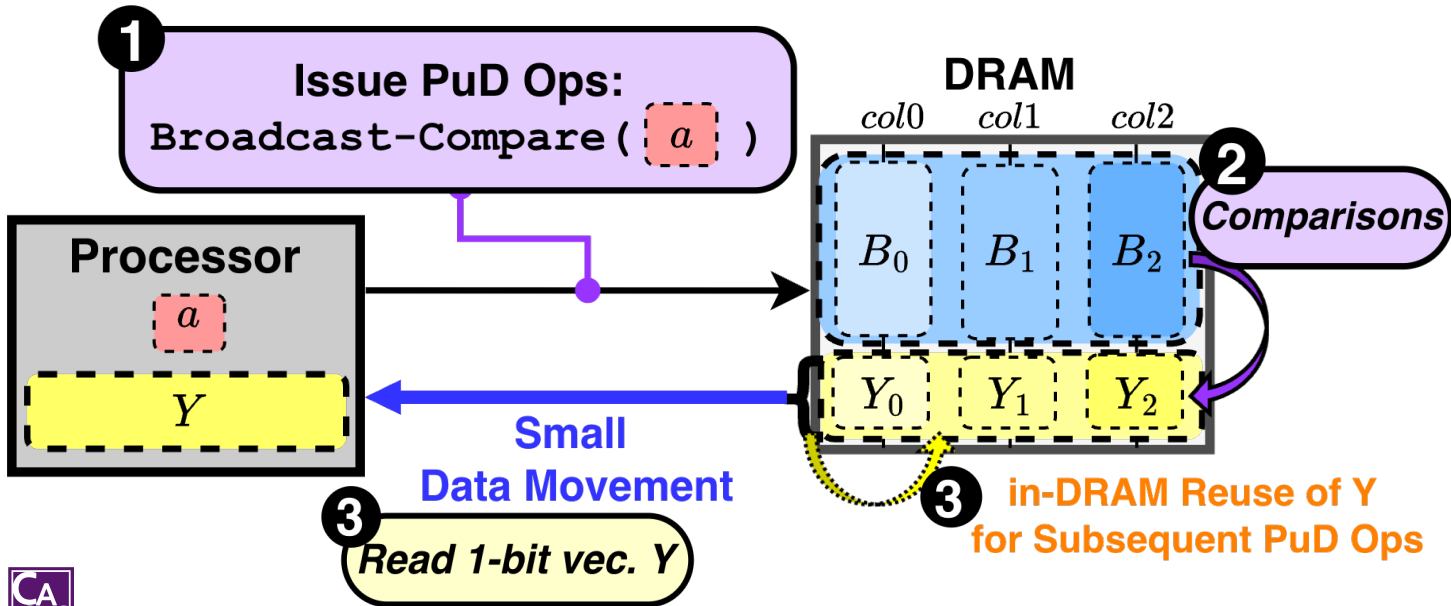


Data Movement Reduction via PuD Execution

Processor-Centric Paradigm



Processing-using-DRAM



Outline

Background

Motivation: PuD for Vector-Scalar Comparison

Limitation of Prior PuD Approaches

Clutch: Lookup Table-Based Comparison

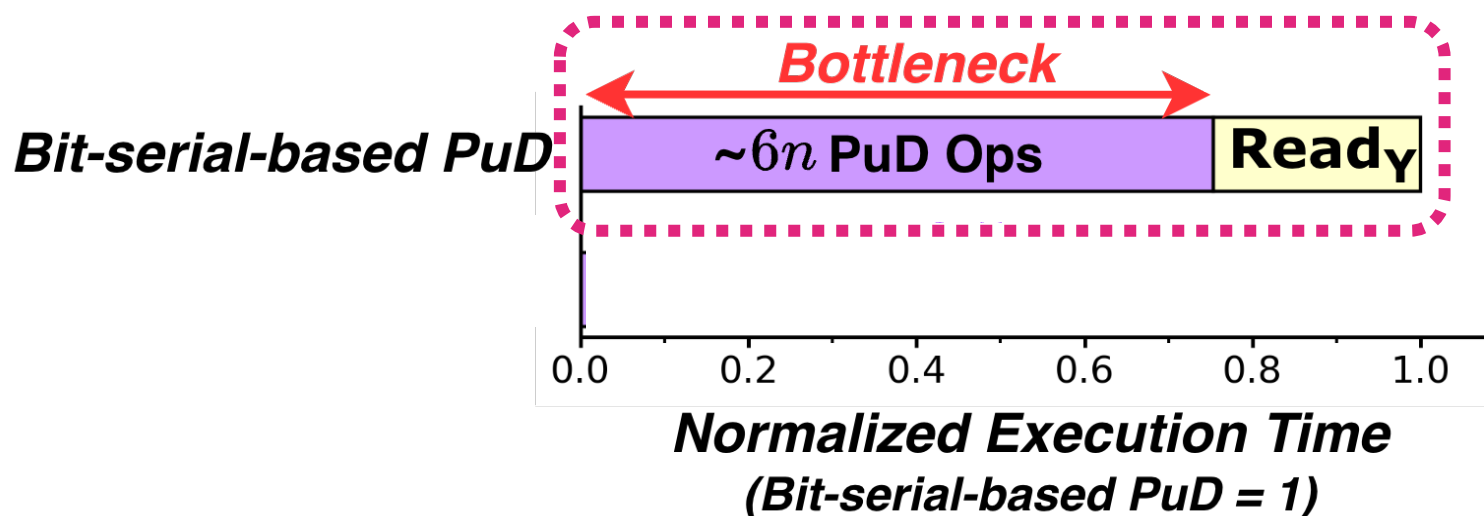
Clutch: Divide-and-Conquer Approach

Evaluation

Conclusion

Limitation of Bit-Serial PuD

- **Prior PuD¹ arithmetic** relies on **bit-serial execution**, requiring **many PuD operations** for comparison
- PuD operation latency **dominates total comparison latency**
 - accounting for up to **76%** of total latency in comparisons in our evaluation

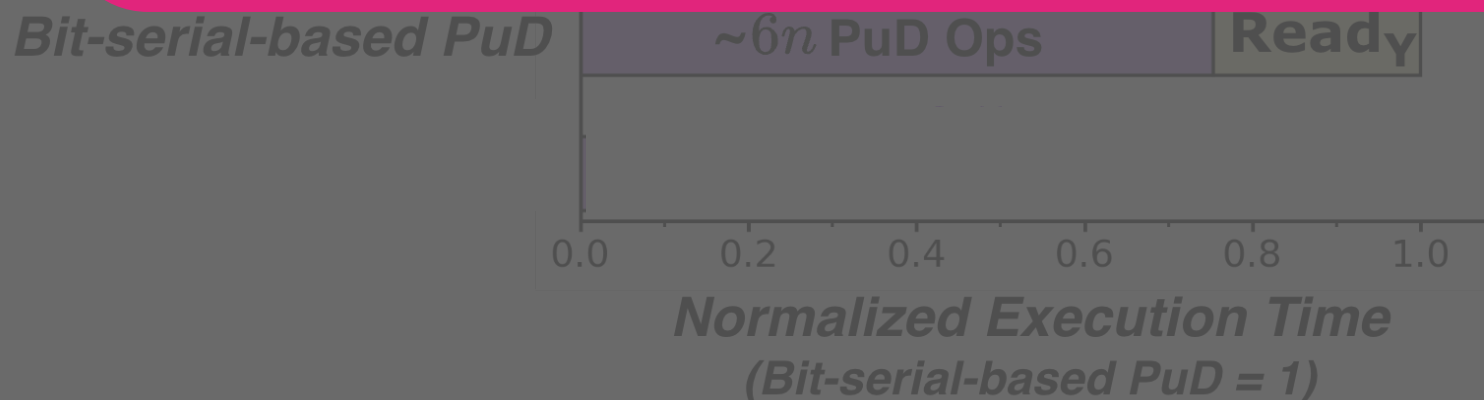


Limitation of Bit-Serial PuD

- Prior PuD¹ arithmetic relies on bit-serial execution, requiring many PuD operations for comparison
- PuD operation latency dominates total comparison latency

Key Challenge:

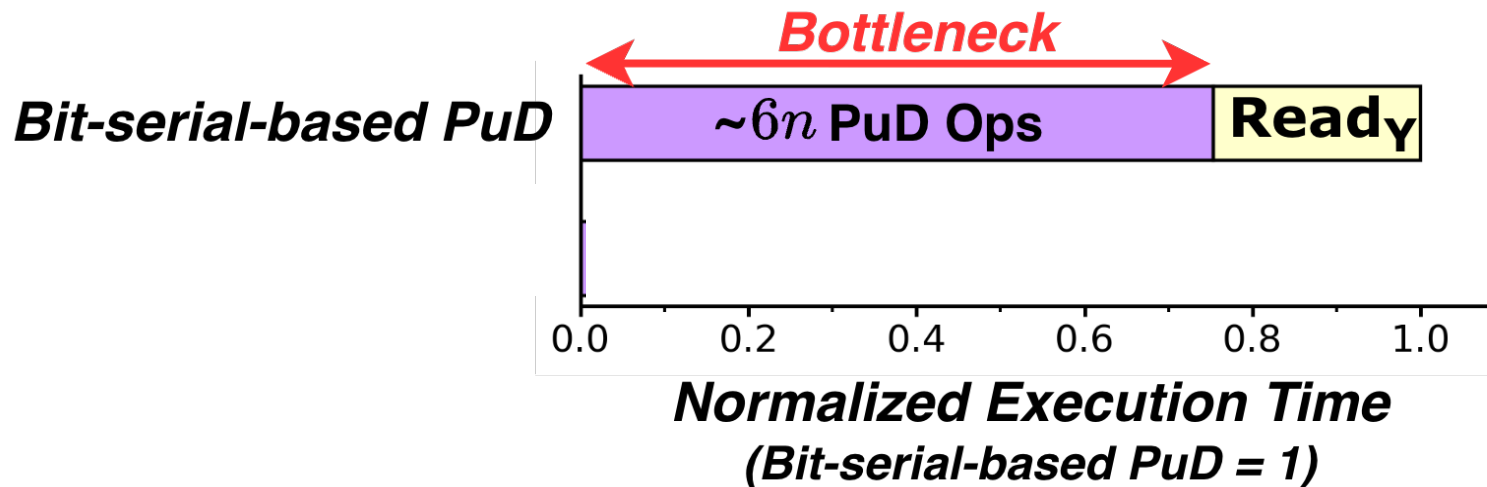
Reducing the number of PuD operations for comparison is critical to unlocking PuD's performance potential



Our Goal

Our Goal:

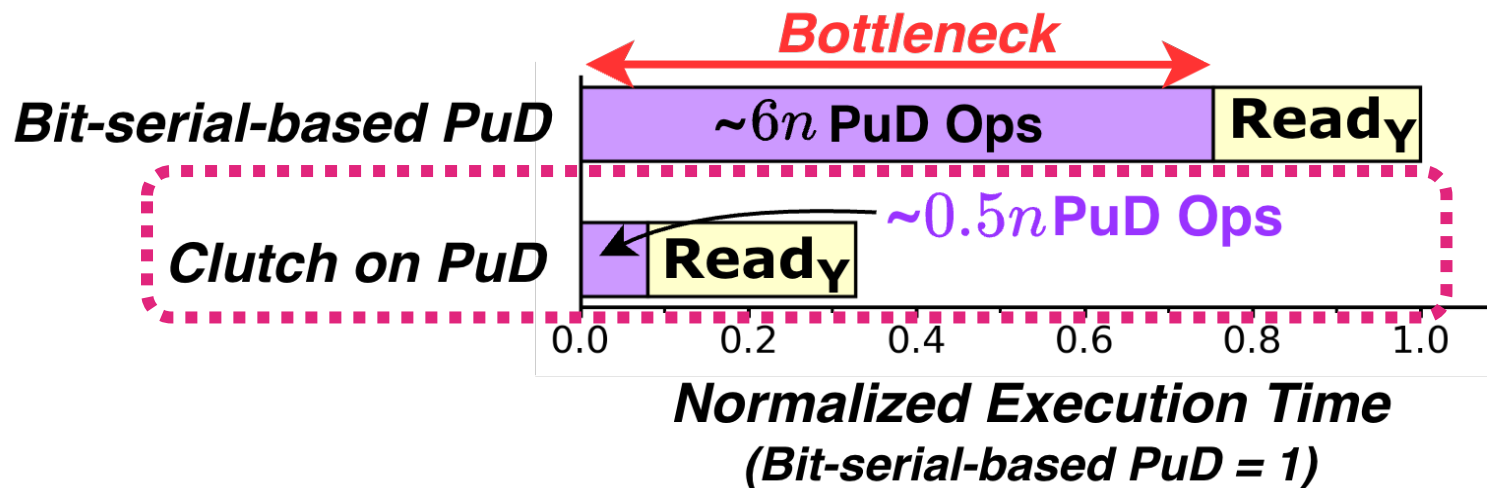
Design a comparison algorithm that significantly reduces the number of PuD operations



Our Goal

Our Goal:

Design a comparison algorithm that significantly reduces the number of PuD operations



Clutch: Overview

Clutch (Comparison Algorithm using Lookup Table with Chunked Temporal Coding)

Temporal Coding: Encodes operands so that a comparison reduces to a single lookup

Divide-and-Conquer: Decomposes operands into multiple smaller chunks, significantly reducing memory footprint

Outline

Background

Motivation: PuD for Vector-Scalar Comparison

Limitation of Prior PuD Approaches

Clutch: Lookup Table-Based Comparison

Clutch: Divide-and-Conquer Approach

Evaluation

Conclusion

Clutch: Lookup Table-Based Comparison

Key Idea: Instead of bit-serial execution, Clutch encodes vector elements **using temporal coding**

Key property: The **i -th bit** of the encoded **value b** equals the truth value of **$i < b$**

$$b = 6$$

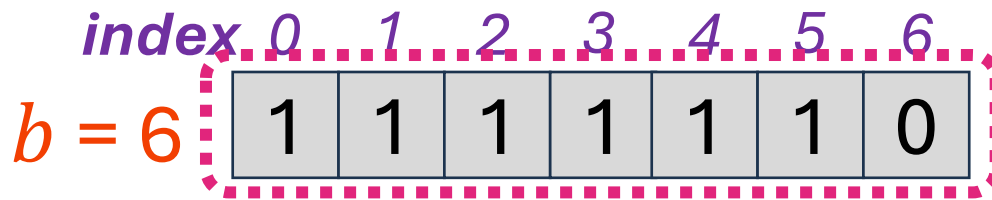
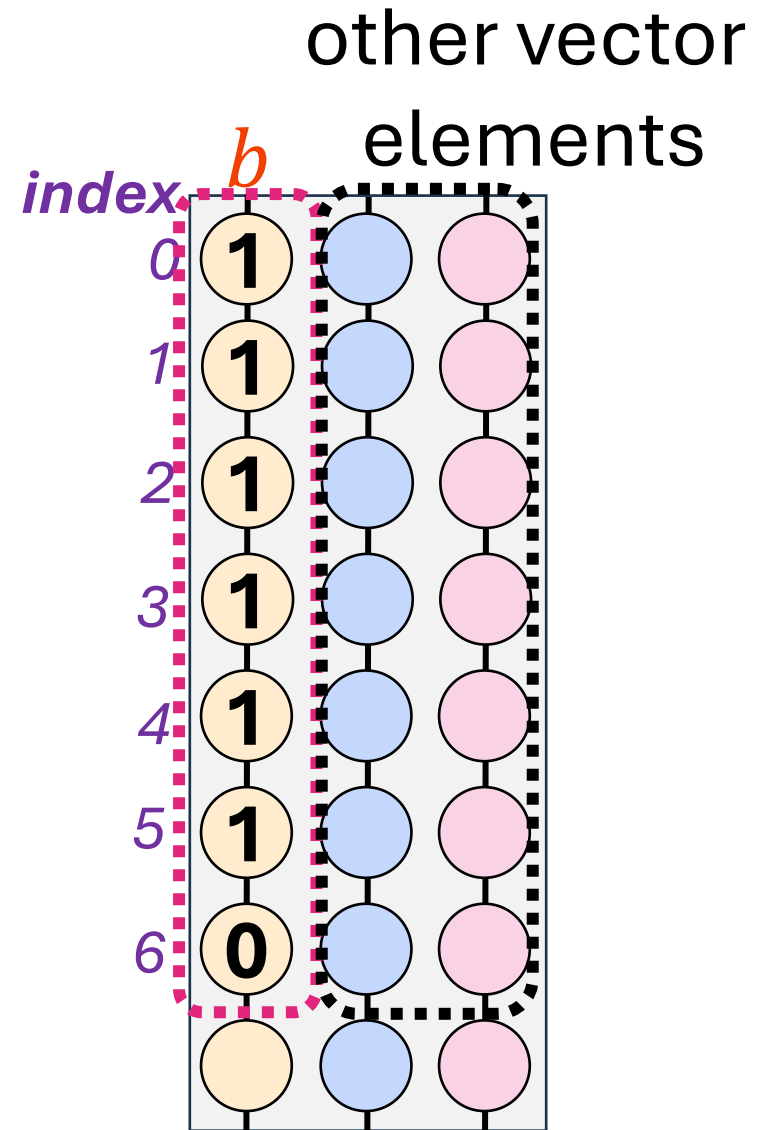
Binary

MSB			LSB
1	1	0	

Temporal

<i>index</i>	0	1	2	3	4	5	6
1	1	1	1	1	1	1	0

Clutch: Lookup Table-Based Comparison



Clutch: Lookup Table-Based Comparison

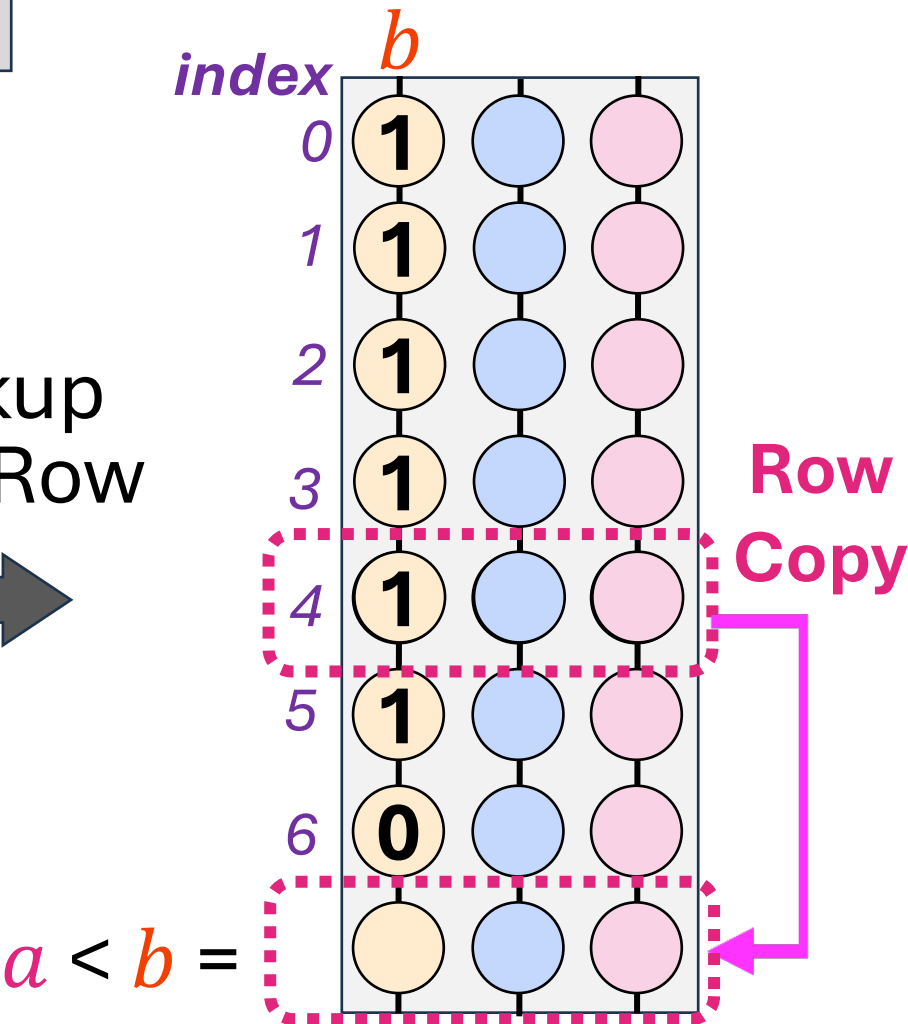
index 0 1 2 3 4 5 6

b = 6

1	1	1	1	1	1	0
---	---	---	---	---	---	---

$a = 4$
 $a < b = ??$

Lookup
a-th Row



Clutch: Lookup Table-Based Comparison

index	0	1	2	3	4	5	6
$b = 6$	1	1	1	1	1	1	0

The lookup table-based approach **significantly reduces PuD operation count**

However, a naive lookup table requires $2^n - 1$ rows for n -bit comparisons, resulting in a **prohibitive memory footprint** for large bit-widths (e.g., 16-bit or 32-bit)

Outline

Background

Motivation: PuD for Vector-Scalar Comparison

Limitation of Prior PuD Approaches

Clutch: Lookup Table-Based Comparison

Clutch: Divide-and-Conquer Approach

Evaluation

Conclusion

Clutch: Chunked Divide-and-Conquer

$$\begin{array}{c} \text{MSB} \qquad \qquad \text{LSB} \\ a = 5 \\ \begin{array}{|c|c|c|c|} \hline 0 & 1 & 0 & 1 \\ \hline \end{array} \end{array} < \begin{array}{c} \text{MSB} \qquad \qquad \text{LSB} \\ b = 6 \\ \begin{array}{|c|c|c|c|} \hline 0 & 1 & 1 & 0 \\ \hline \end{array} = ? \end{array}$$

Clutch: Chunked Divide-and-Conquer

$a = 5$

MSB LSB

0	1	0	1
---	---	---	---

Chunk1 Chunk0

$a_1 = 1$ $a_0 = 1$

<

$b = 6$

MSB LSB

0	1	1	0
---	---	---	---

Chunk1 Chunk0

$b_1 = 1$ $b_0 = 2$

= ?

<

= False

Divide

<

= True



How to Conquer?

Clutch: Divide-and-Conquer

$a = 5$

MSB LSB

0	1	0	1
---	---	---	---

$a_1 = 1$ $a_0 = 1$

$b = 6$

MSB LSB

0	1	1	0
---	---	---	---

$b_1 = 1$ $b_0 = 2$

$a < b$

=

Clutch: Divide-and-Conquer

$a = 5$

MSB LSB

0	1	0	1
---	---	---	---

$a_1 = 1$ $a_0 = 1$

$b = 6$

MSB LSB

0	1	1	0
---	---	---	---

$b_1 = 1$ $b_0 = 2$

$$a < b$$

$$= a_1 < b_1 \text{ or}$$

$$a = 3$$

0	0	1	1
---	---	---	---

$$b = 6$$

0	1	1	0
---	---	---	---

$<$

$$\Rightarrow a < b$$

Clutch: Divide-and-Conquer

$a = 5$

MSB LSB

0	1	0	1
---	---	---	---

$a_1 = 1$ $a_0 = 1$

$b = 6$

MSB LSB

0	1	1	0
---	---	---	---

$b_1 = 1$ $b_0 = 2$

$$a < b$$
$$= a_1 < b_1 \text{ or } (a_1 \leq b_1 \text{ and } a_0 < b_0)$$

$a = 5$

0	1	0	1
---	---	---	---

$b = 6$

0	1	1	0
---	---	---	---

$\leq$
and
 $<$

$\Rightarrow a < b$

Clutch: Divide-and-Conquer

$a = 5$

MSB LSB

0	1	0	1
---	---	---	---

$a_1 = 1$ $a_0 = 1$

$b = 6$

MSB LSB

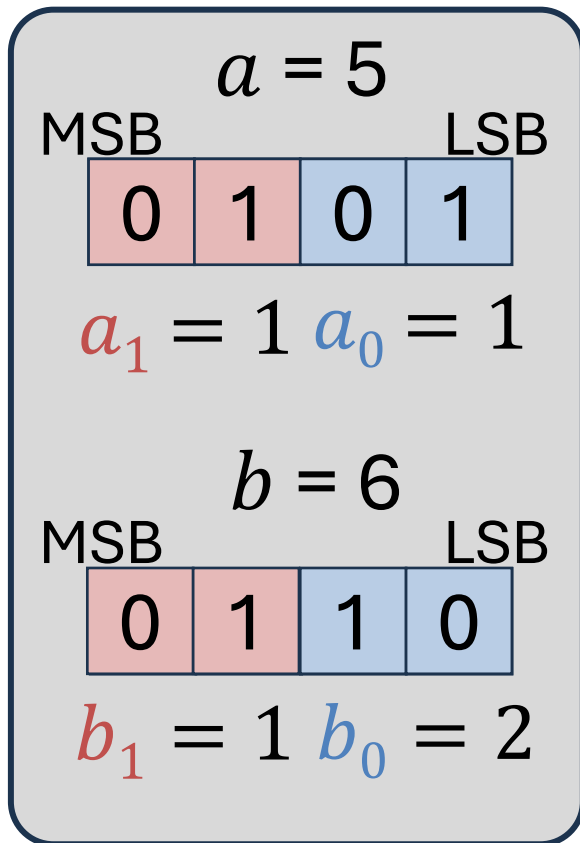
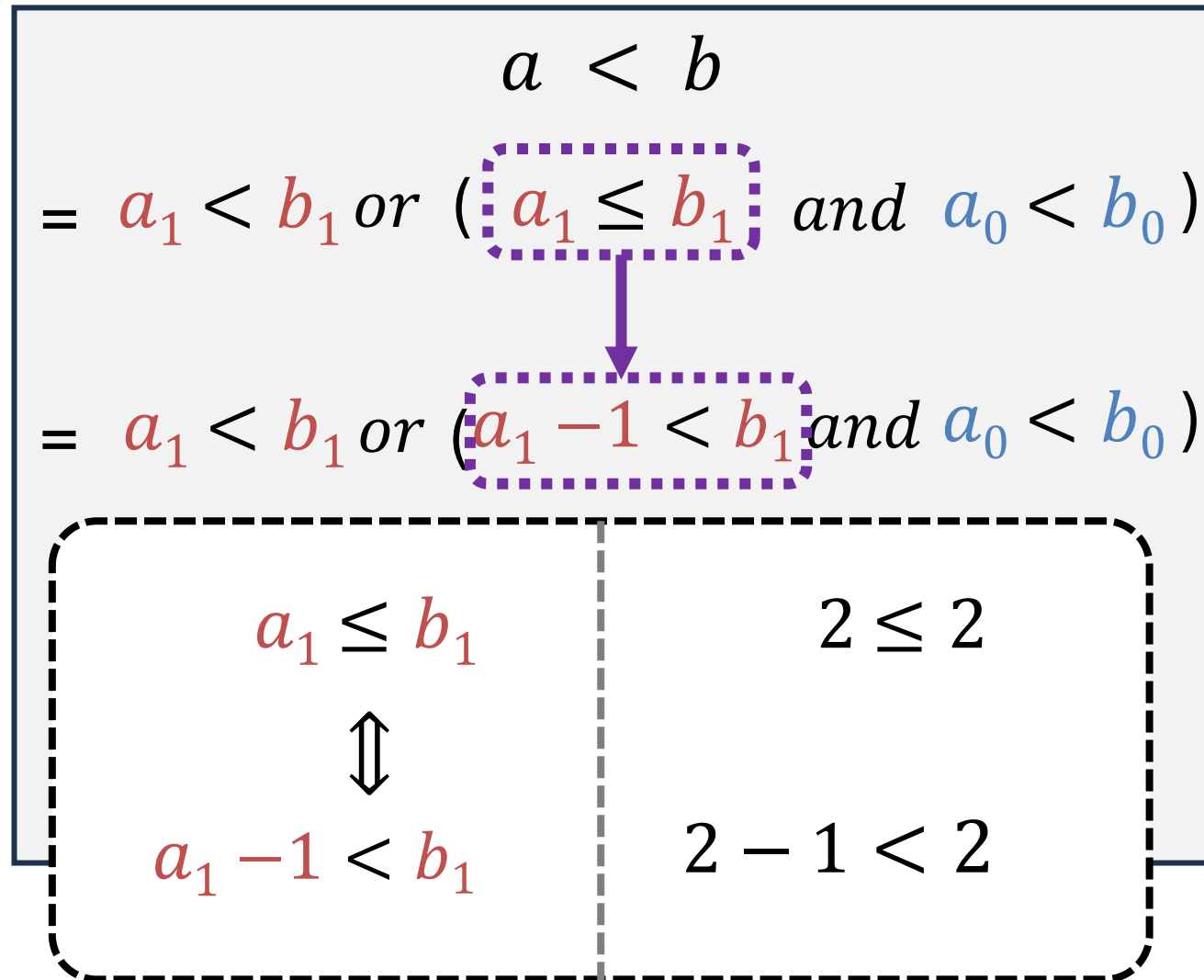
0	1	1	0
---	---	---	---

$b_1 = 1$ $b_0 = 2$

$$a < b$$

$$= a_1 < b_1 \text{ or } (a_1 \leq b_1 \text{ and } a_0 < b_0)$$

Clutch: Divide-and-Conquer



Clutch: Divide-and-Conquer

$a = 5$

MSB LSB

0	1	0	1
---	---	---	---

$a_1 = 1$ $a_0 = 1$

$b = 6$

MSB LSB

0	1	1	0
---	---	---	---

$b_1 = 1$ $b_0 = 2$

$$a < b$$

$$= a_1 < b_1 \text{ or } (a_1 \leq b_1 \text{ and } a_0 < b_0)$$

$$= a_1 < b_1 \text{ or } (a_1 - 1 < b_1 \text{ and } a_0 < b_0)$$

=

Clutch: Divide-and-Conquer

$a = 5$

MSB LSB

0	1	0	1
---	---	---	---

$a_1 = 1$ $a_0 = 1$

$b = 6$

MSB LSB

0	1	1	0
---	---	---	---

$b_1 = 1$ $b_0 = 2$

$a_1 < b_1$	$a_1 - 1 < b_1$	$a_0 < b_0$	$a < b$
True → True		X	True
False	True	True	True

$$= a_1 < b_1 \text{ or } (a_1 - 1 < b_1 \text{ and } a_0 < b_0)$$

$$= \text{MAJ3}(a_1 < b_1, a_1 - 1 < b_1, a_0 < b_0)$$

Clutch: Divide-and-Conquer

$a = 5$

MSB LSB

0	1	0	1
---	---	---	---

$a_1 = 1$ $a_0 = 1$

$b = 6$

MSB LSB

0	1	1	0
---	---	---	---

$b_1 = 1$ $b_0 = 2$

$$a < b$$

$$= a_1 < b_1 \text{ or } (a_1 \leq b_1 \text{ and } a_0 < b_0)$$

$$= a_1 < b_1 \text{ or } (a_1 - 1 < b_1 \text{ and } a_0 < b_0)$$

$$= \text{MAJ3}(a_1 < b_1, a_1 - 1 < b_1, a_0 < b_0)$$

Clutch: Lookup Table & Divide-and-Conquer

$$a < b$$
$$= \text{MAJ3}(a_1 < b_1, a_1 - 1 < b_1, a_0 < b_0)$$

$a = 5$

MSB LSB

0	1	0	1
---	---	---	---

$a_1 = 1$ $a_0 = 1$

$b = 6$

MSB LSB

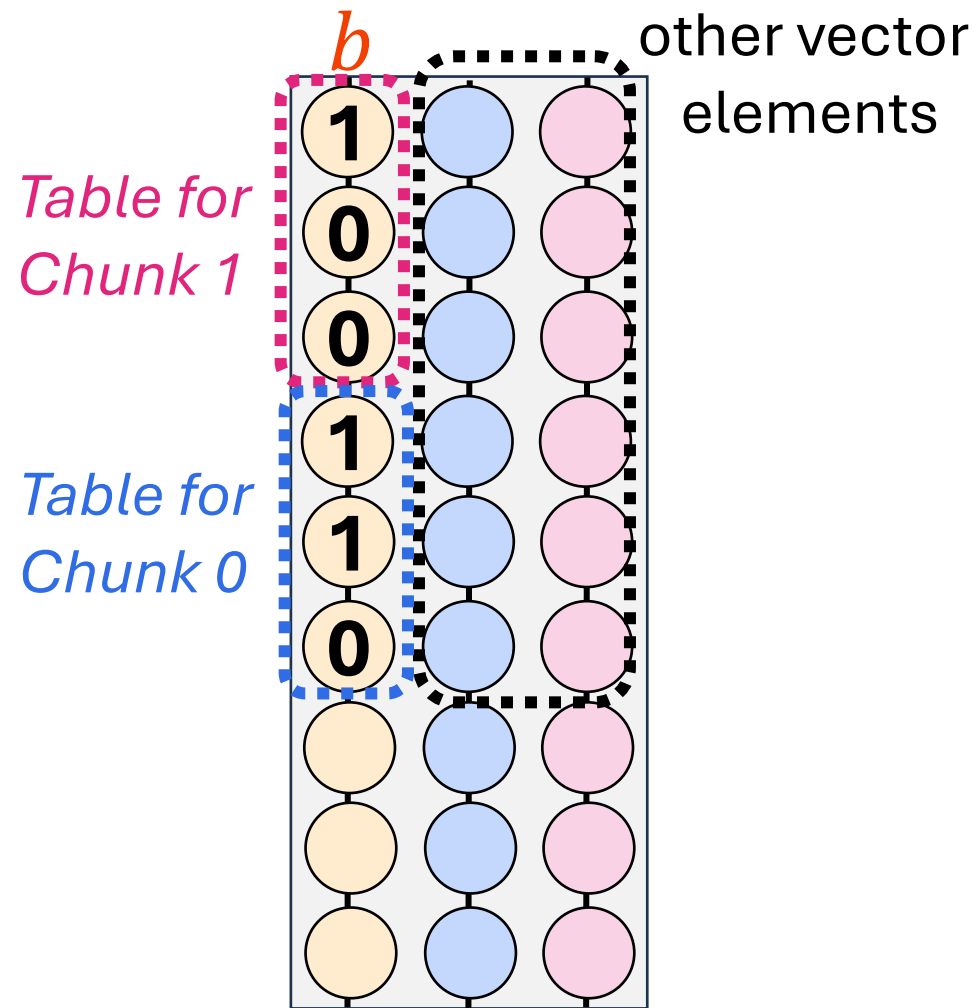
0	1	1	0
---	---	---	---

$b_1 = 1$ $b_0 = 2$

1	0	0	1	1	0
---	---	---	---	---	---

$b_1 = 1$ $b_0 = 2$

Chunked Temporal



Clutch: Lookup Table & Divide-and-Conquer

$$= \text{MAJ3}(\overbrace{a_1 < b_1}^{a_1 < b}, \overbrace{a_1 - 1 < b_1}^{a_1 < b}, \overbrace{a_0 < b_0}^{a_1 < b})$$

$a = 5$

MSB LSB

0	1	0	1
---	---	---	---

$a_1 = 1$ $a_0 = 1$

$b = 6$

MSB LSB

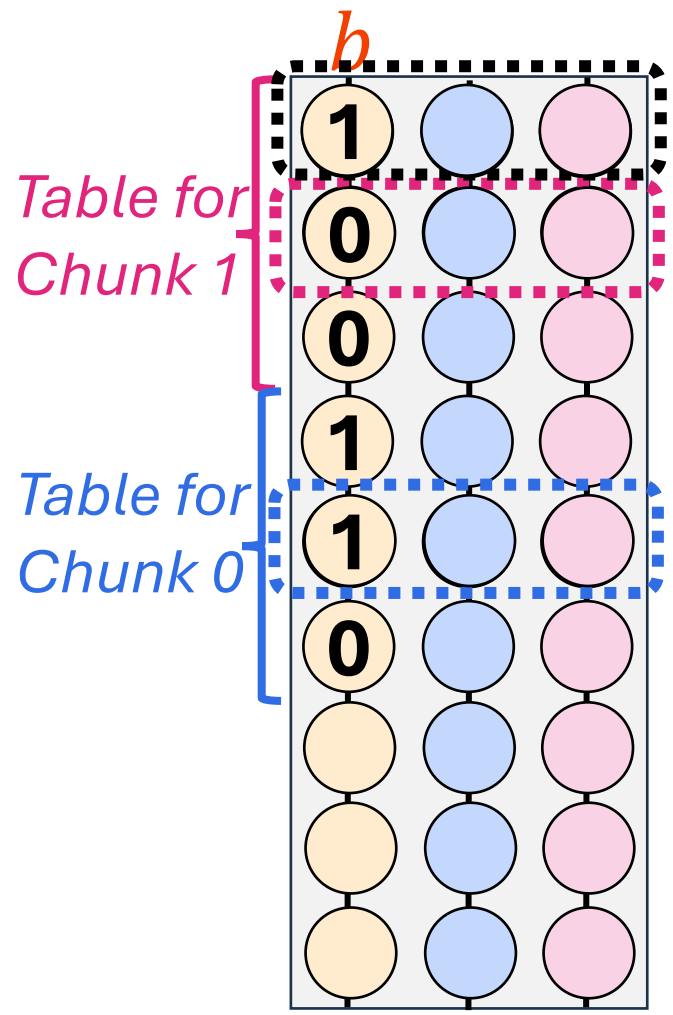
0	1	1	0
---	---	---	---

$b_1 = 1$ $b_0 = 2$

1	0	0	1	1	0
---	---	---	---	---	---

$b_1 = 1$ $b_0 = 2$

Chunked Temporal



Clutch: Lookup Table & Divide-and-Conquer

$$= \text{MAJ3}(a_1 < b_1, a_1 - 1 < b_1, a_0 < b_0)$$

MSB

$a = 5$

LSB

0

1

0

1

$a_1 = 1$

$a_0 = 1$

MSB

$b = 6$

LSB

0

1

1

0

$b_1 = 1$

$b_0 = 2$

1

0

0

1

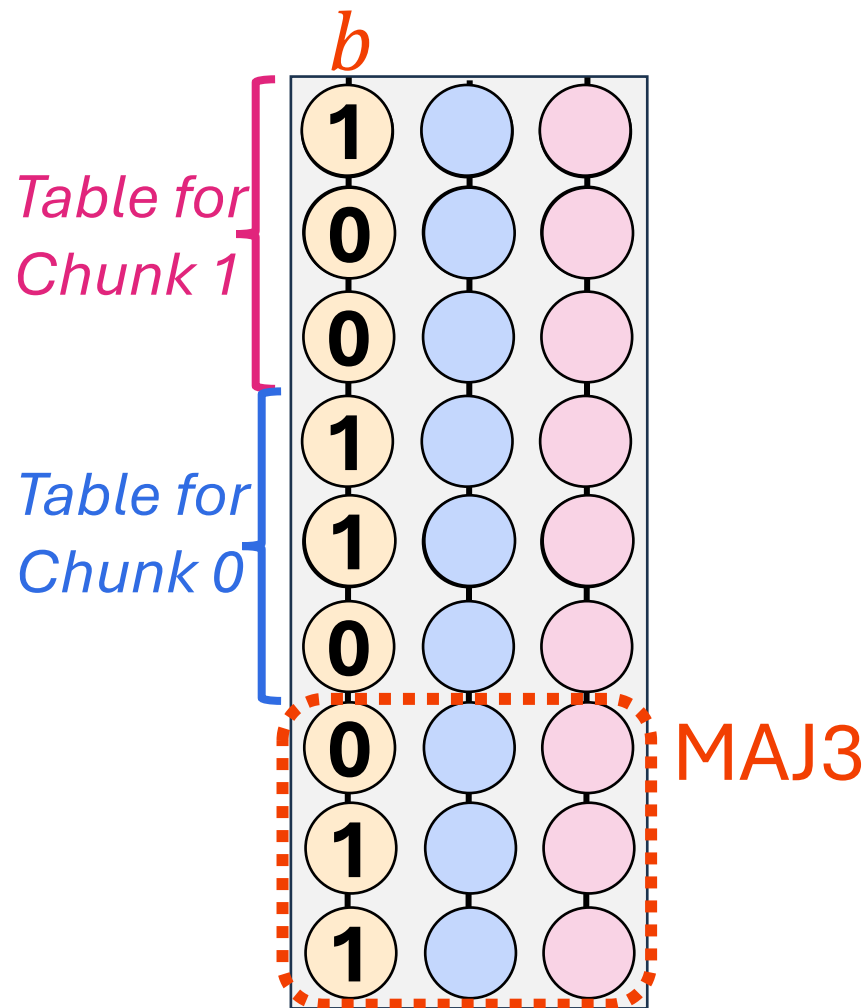
1

0

$b_1 = 1$

$b_0 = 2$

Chunked Temporal



Clutch: Lookup Table & Divide-and-Conquer

$$= \text{MAJ3}(a_1 < b_1, a_1 - 1 < b_1, a_0 < b_0)$$

MSB

$a = 5$

0

1

0

1

LSB

$a_1 = 1$

$a_0 = 1$

MSB

$b = 6$

0

1

1

0

LSB

$b_1 = 1$

$b_0 = 2$

1

0

0

1

1

0

$b_1 = 1$

$b_0 = 2$

Chunked Temporal

b

1

0

0

1

1

0

1

1

1

Table for
Chunk 1

Table for
Chunk 0

MAJ3

Results of
 $a < b$

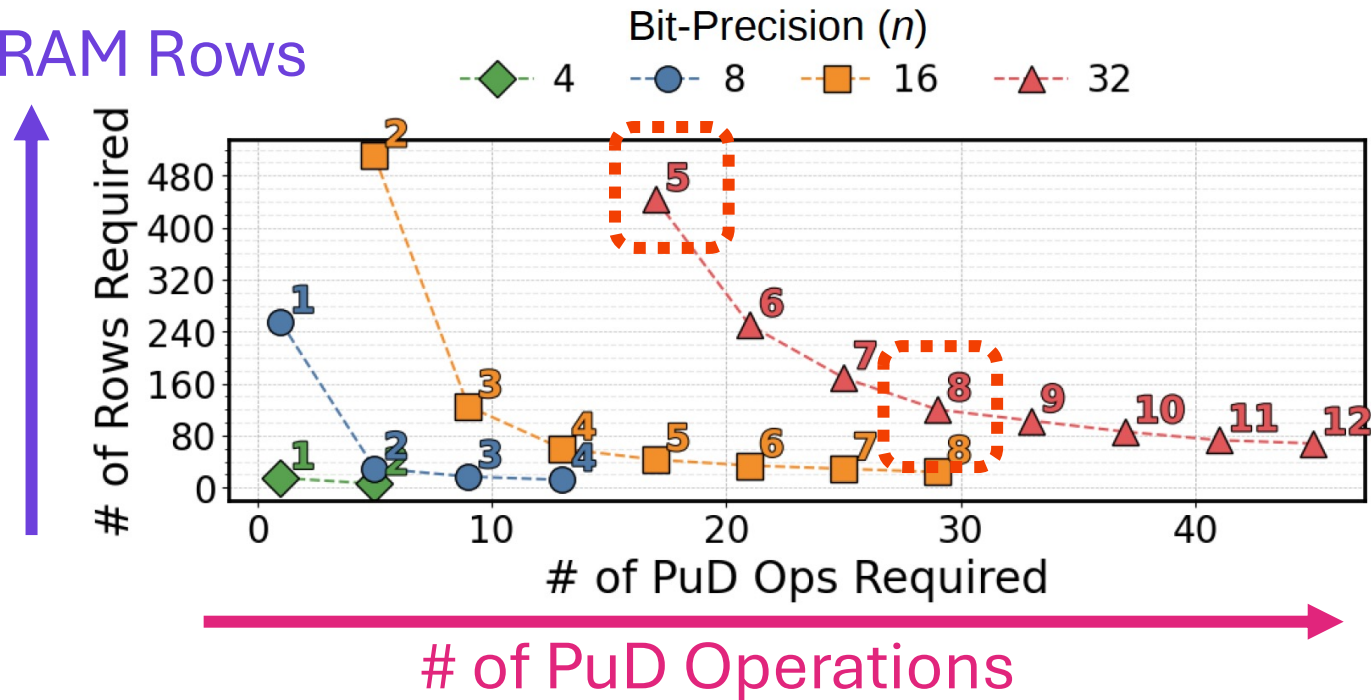
Clutch: Scalability via Chunk Count

The divide-and-conquer approach can be applied recursively from the LSB side, generalizing to any number of chunks

By adjusting the number of chunks, Clutch provides a flexible tradeoff between latency and memory footprint

Clutch: Scalability via Chunk Count

of DRAM Rows



5 Chunks

- 11x faster than bit-serial PuD with 432 rows

vs.

8 Chunks

- 6.4x faster than bit-serial PuD with 120 rows

Outline

Background

Motivation: PuD for Vector-Scalar Comparison

Limitation of Prior PuD Approaches

Clutch: Lookup Table-Based Comparison

Clutch: Divide-and-Conquer Approach

Evaluation

Conclusion

Evaluation: Methodology Overview

- Evaluation Setup
 - 4 × 16 GB DDR4, dual-channel
 - CPU: Intel Core i7-9700K
 - PuD: Unmodified PuD², SIMD³

- Workloads

- Predicate Evaluation
- Decision Tree Inference
- 8-bit, 16-bit, 32-bit

Q1 WHERE $x_0 < f_i < x_1$

Q2 WHERE ($x_0 < f_i < x_1$ AND $y_0 < f_j < y_1$)

Q3 COUNT (WHERE ($x_0 < f_i < x_1$ OR $y_0 < f_j < y_1$))

Q4 AVERAGE(f_k) FROM (WHERE $x_0 < f_i < x_1$ AND $y_0 < f_j < y_1$)

Q5 WITH avg_val = AVERAGE(f_k)
WHERE ($x_0 < f_i < x_1$ OR $y_0 < f_j < y_1$)
COUNT (WHERE avg_val < $f_l < 2 \cdot \text{avg_val}$)

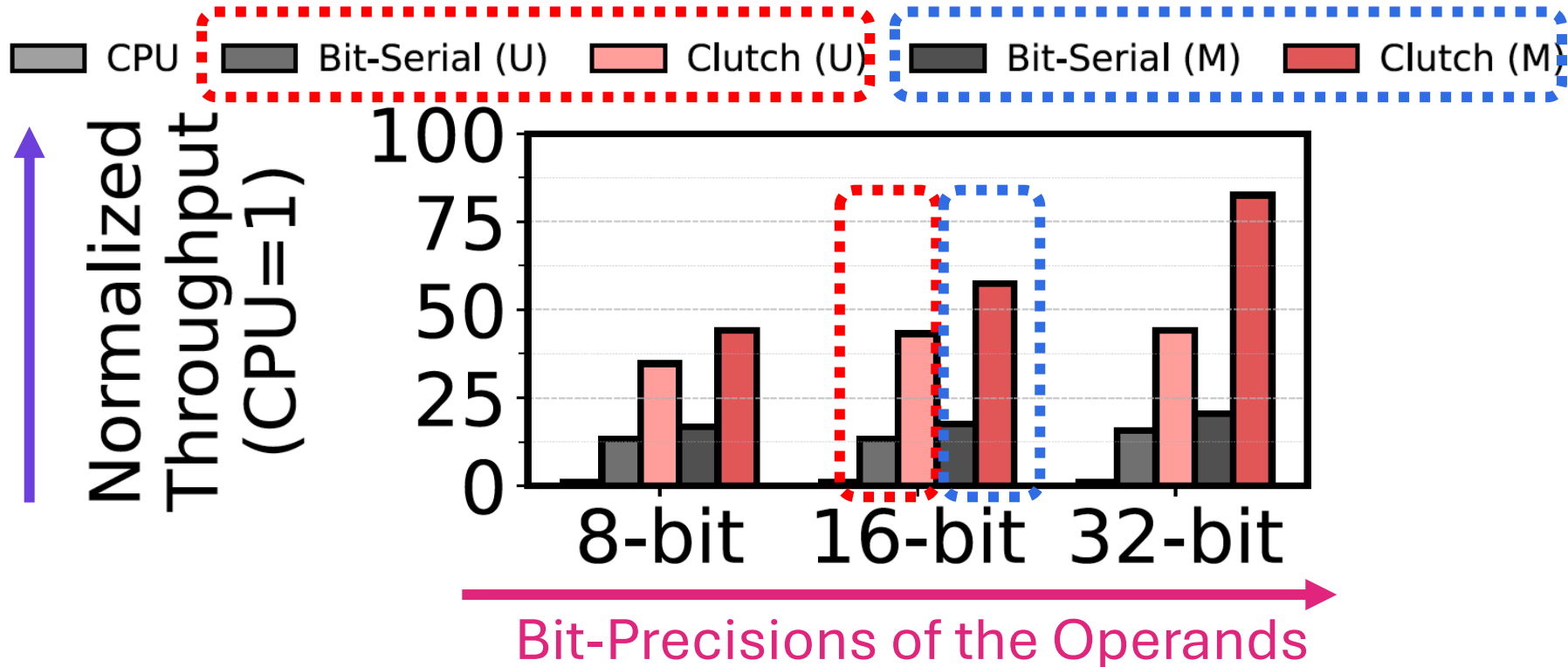
- Main Metrics
 - CPU-Normalized Throughput (CPU: BitWeaving-V⁴)
 - Real Hardware, In-House Cycle-Accurate Simulator for PuD
 - CPU-Normalized Energy Efficiency
 - Intel RAPL, CACTI

Evaluation: Throughput

Q2 WHERE ($x_0 < f_i < x_1$ AND $y_0 < f_j < y_1$)

Unmodified PuD

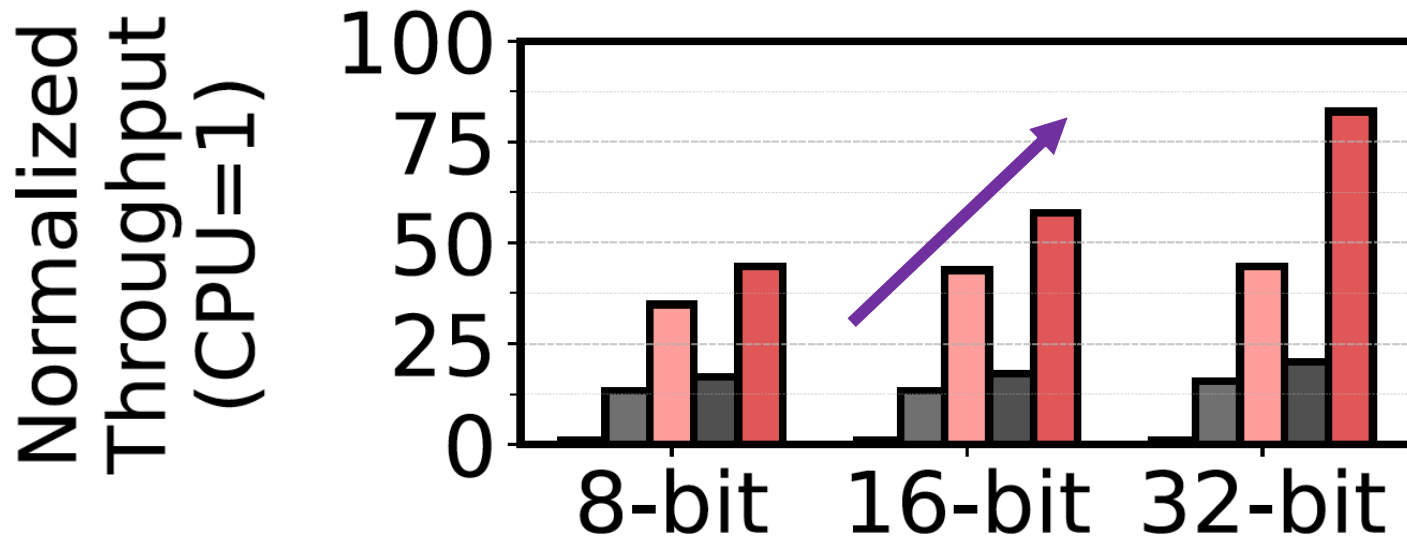
SIMDRAM



Evaluation: Throughput

Q2 WHERE ($x_0 < f_i < x_1$ AND $y_0 < f_j < y_1$)

CPU Bit-Serial (U) Clutch (U) Bit-Serial (M) Clutch (M)



Clutch provides up to **83×** (**50× on average**) higher throughput over CPU, and up to **4.0×** (**3.1× on average**) over bit-serial PuD

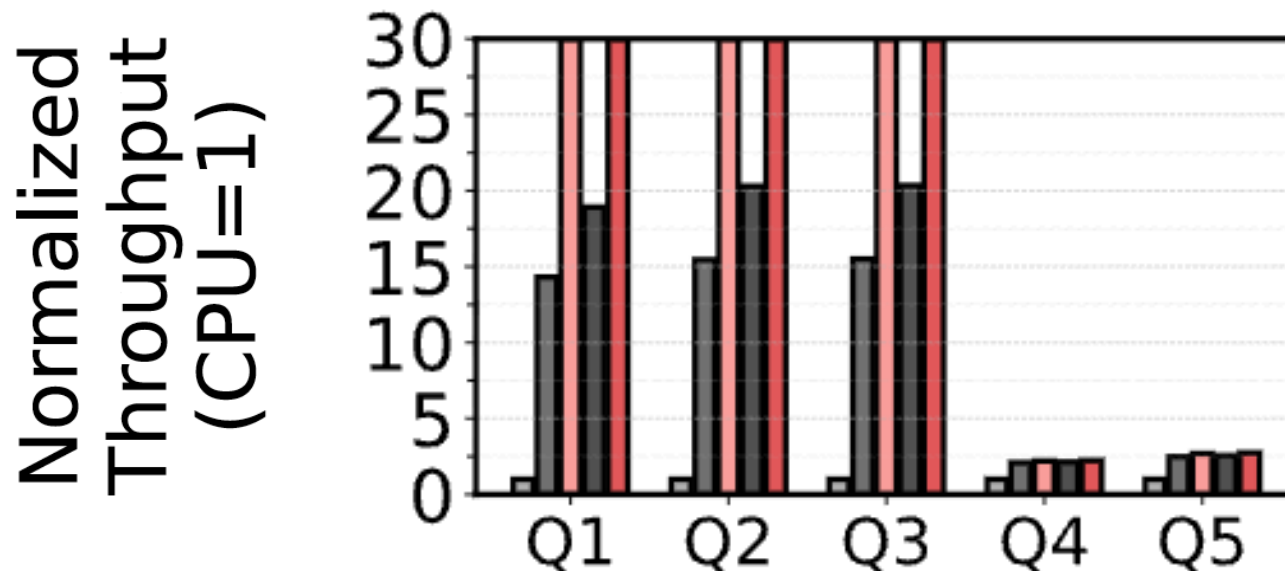
Evaluation: Throughput

Q1 WHERE $x_0 < f_i < x_1$

On average, Clutch provides **28× higher throughput** over CPU and **2.2× higher throughput** over bit-serial PuD across Q1 through Q5.

COUNT (WHERE $\text{avg_val} < f_i < 2 \cdot \text{avg_val}$)

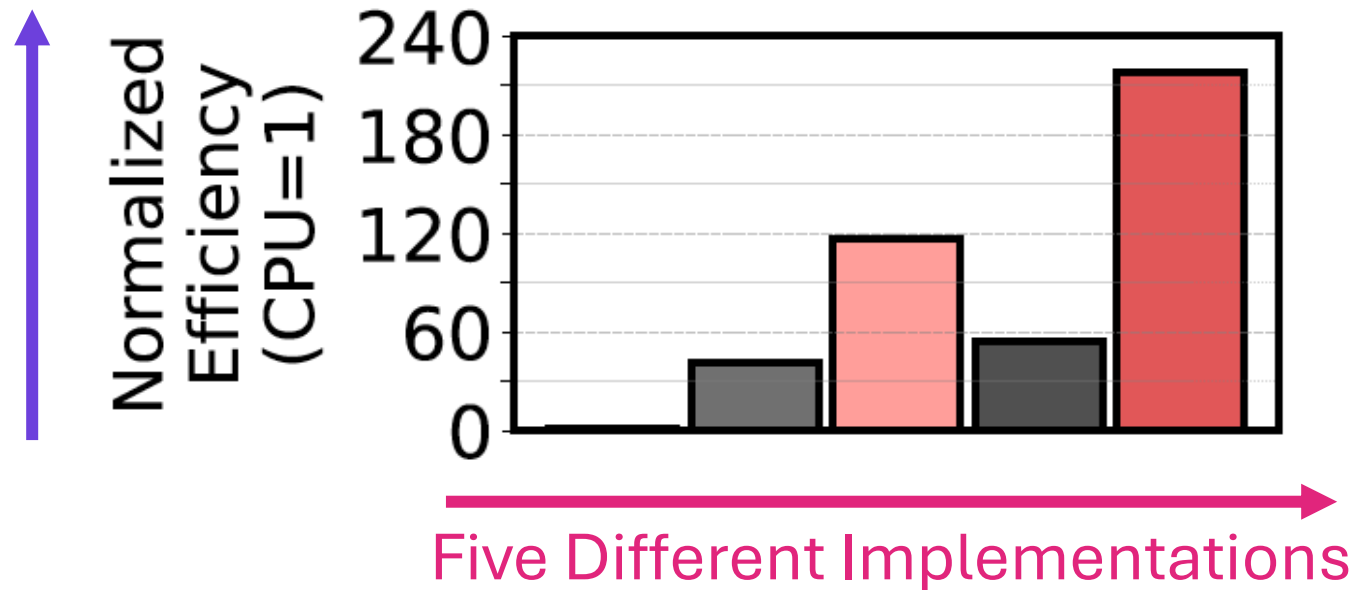
CPU Bit-Serial (U) Clutch (U) Bit-Serial (M) Clutch (M)



Evaluation: Energy Efficiency

Q2 WHERE ($x_0 < f_i < x_1$ AND $y_0 < f_j < y_1$)

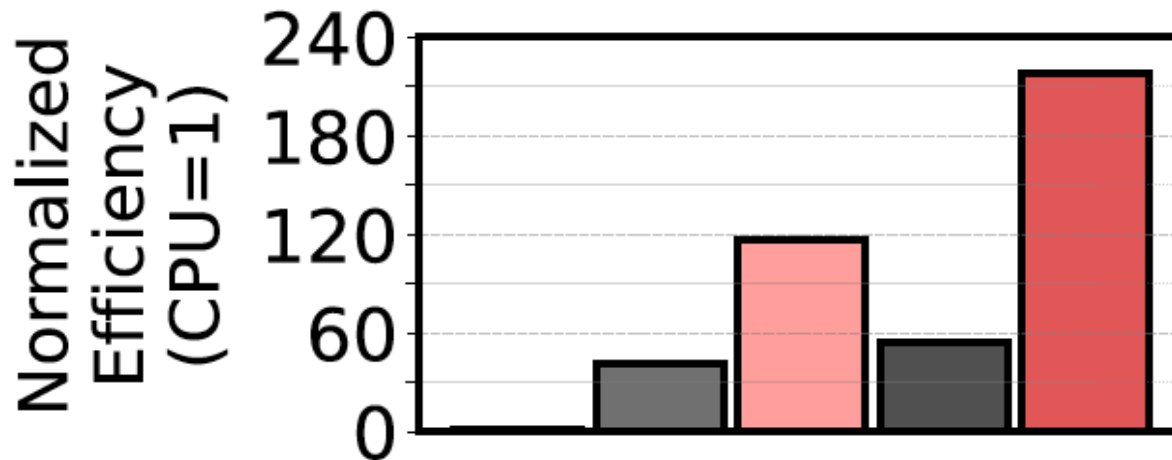
CPU Bit-Serial (U) Clutch (U) Bit-Serial (M) Clutch (M)



Evaluation: Energy Efficiency

Q2 WHERE ($x_0 < f_i < x_1$ AND $y_0 < f_j < y_1$)

CPU Bit-Serial (U) Clutch (U) Bit-Serial (M) Clutch (M)



Clutch provides up to **218×** (**135× on average**) higher energy efficiency over CPU, and up to **4.2×** (**3.1× on average**) over bit-serial PuD

Clutch: High Performance Vector-Scalar Comparison using DRAM via Chunked Temporal Coding

Daichi Tokuda
The University of Tokyo
Tokyo, Japan
tokuda-daichi@is.s.u-tokyo.ac.jp

Ataberk Olgun
ETH Zurich
Zurich, Switzerland
olgunataberk@gmail.com

Geraldo F. Oliveira
ETH Zurich
Zurich, Switzerland
geraldofjunior@gmail.com

Tatsuya Kubo
The University of Tokyo / RIKEN
Tokyo, Japan
tatsuya.kubo@is.s.u-tokyo.ac.jp

Haocong Luo
ETH Zurich
Zurich, Switzerland
richardluo723@gmail.com

Abdullah Giray Yağlıkçı
CISPA
Saarbrücken, Germany
giray.yaglikci@safari.ethz.ch

Onur Mutlu
ETH Zurich
Zurich, Switzerland
omutlu@gmail.com

Shinya Takamaeda-Yamazaki
The University of Tokyo / RIKEN
Tokyo, Japan
shinya@is.s.u-tokyo.ac.jp

Ismail Emir Yuksel
ETH Zurich
Zurich, Switzerland
ismailemryksel@gmail.com

Tomoya Nagatani
The University of Tokyo
Tokyo, Japan
t-nagatani@is.s.u-tokyo.ac.jp

Mohammad Sadrosadati
ETH Zürich
Zurich, Switzerland
m.sadr89@gmail.com

Abstract

Vector-scalar comparison is a fundamental computation primitive that compares each element in a vector against a single scalar value. It is widely used in a broad range of data-intensive workloads from databases to machine learning. Due to its low computational intensity, the execution of this operation tends to be memory-bound, especially for large vectors, thereby limiting the utilization of compute resources. Processing-using-DRAM (PuD) is an emerging comput-

lookup-based comparisons, where comparing against a scalar input simply involves accessing the corresponding DRAM row. Second, Clutch leverages our key insight that a divide-and-conquer approach enables scalable lookup-based comparisons without incurring a prohibitive memory footprint at high bit-precision. Specifically, Clutch partitions the operand into multiple multi-bit chunks which can be compared independently using compact lookup tables, and merges per-chunk results through a procedure designed

<https://arxiv.org/abs/2606.22812>

More in the Paper

- **Decision Tree Inference on PuD**
 - **Novel mapping on Decision Tree inference to PuD**, including detailed algorithm design
 - **End-to-end throughput** and energy efficiency evaluation
- **More Application-Level Evaluation**
 - Detailed evaluation setups
 - End-to-end evaluation on GPU-based systems
- **Details on Overhead Analysis**
 - Analysis on throughput–memory footprint tradeoff across different chunk configurations
 - Analysis on data conversion overhead for temporal coding

Outline

Background

Motivation: PuD for Vector-Scalar Comparison

Limitation of Prior PuD Approaches

Clutch: Lookup Table-Based Comparison

Clutch: Divide-and-Conquer Approach

Evaluation

Conclusion

Conclusion

We present **Clutch**, a PuD-oriented algorithm and data representation for accelerating vector-scalar comparison, **combining temporal coding** and **divide-and-conquer** approach

Clutch achieves **12× higher throughput** and **69× better energy efficiency** over optimized CPU execution, and **2.9×/3.0×** over the bit-serial PuD approach

Clutch demonstrates that DRAM can serve as **a high-performance computing substrate** for comparison-intensive workloads

Clutch

High Performance Vector-Scalar Comparison using DRAM via Chunked Temporal Coding

Daichi Tokuda

T. Kubo I. E. Yuksel A. Olgun H. Luo

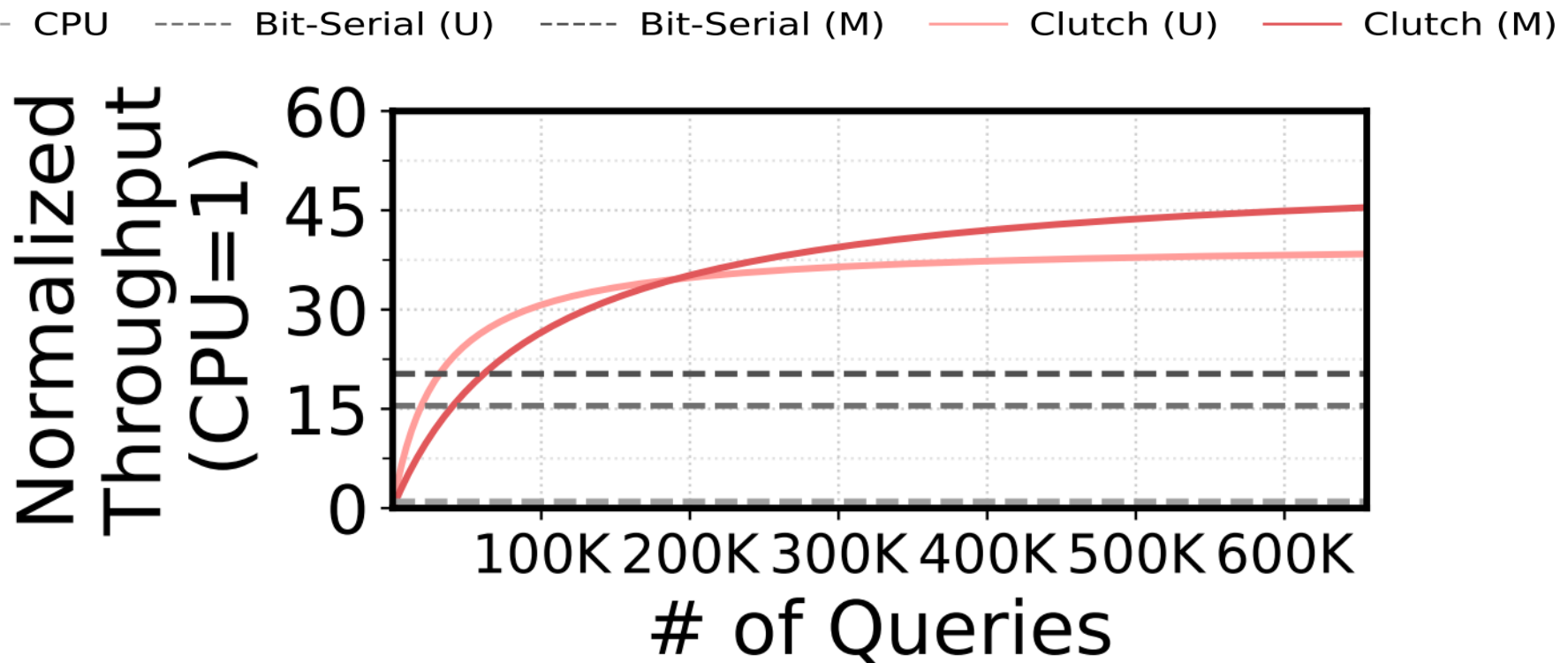
T. Nagatani G. F. Oliveira A. G. Yağlıkçı M. Sadrosadati

O. Mutlu S. Takamaeda-Yamazaki



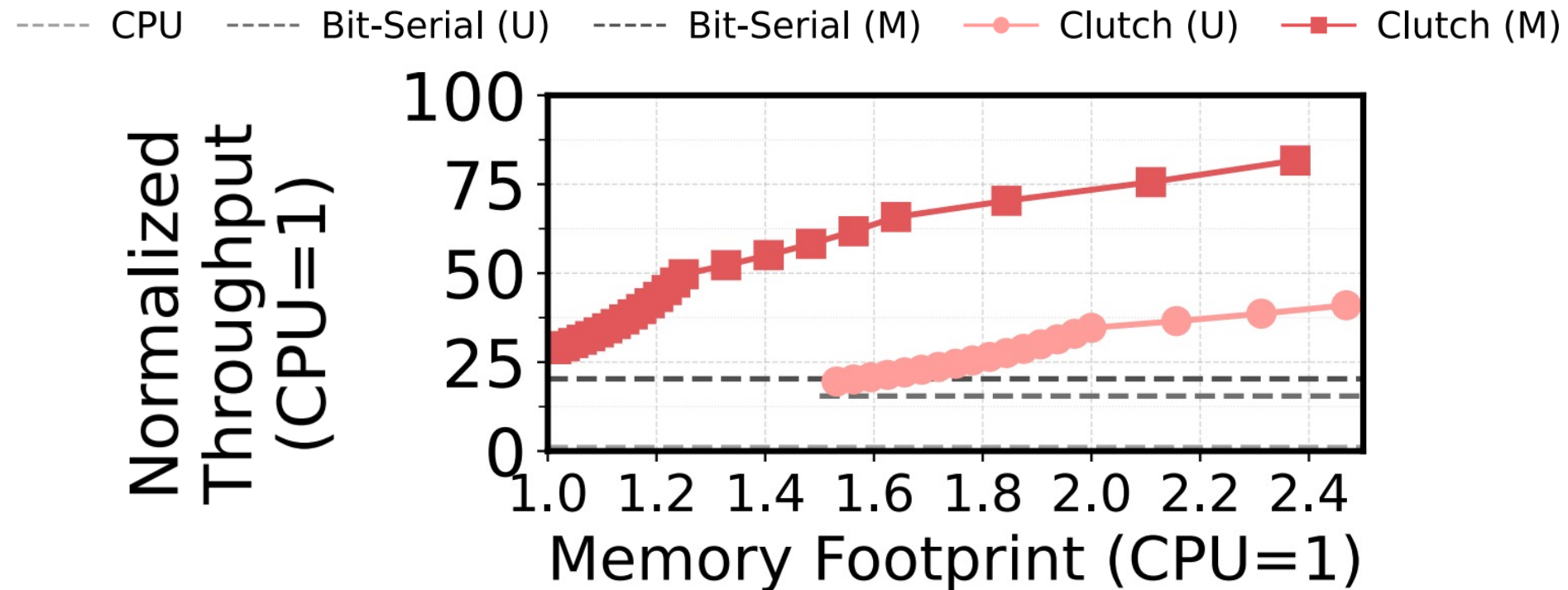
Backup Slides

Data Conversion Overhead



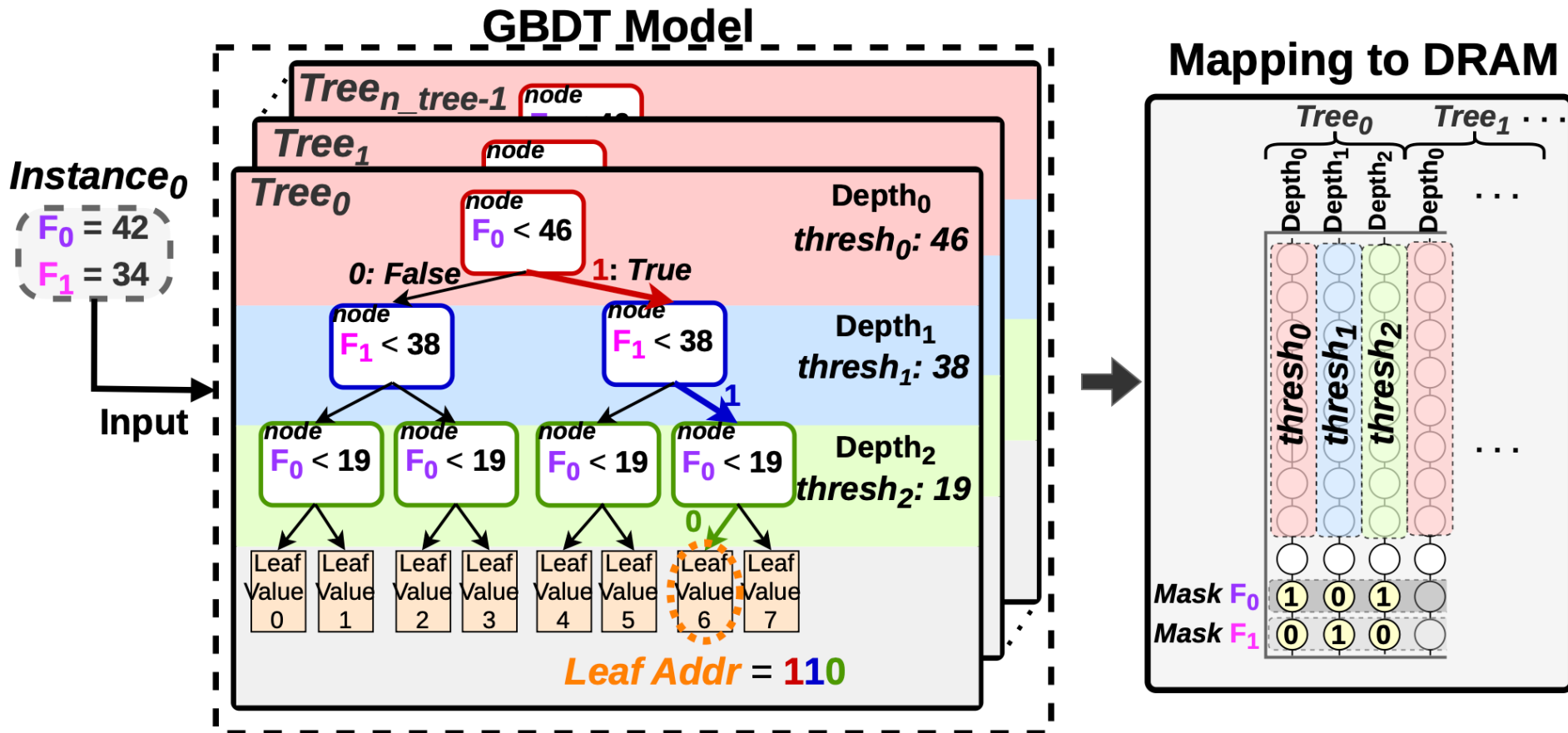
Clutch surpasses CPU throughput after **~2K queries**, and bit-serial PuD after **~50K–60K queries**

Tradeoff: Throughput vs. Memory Footprint

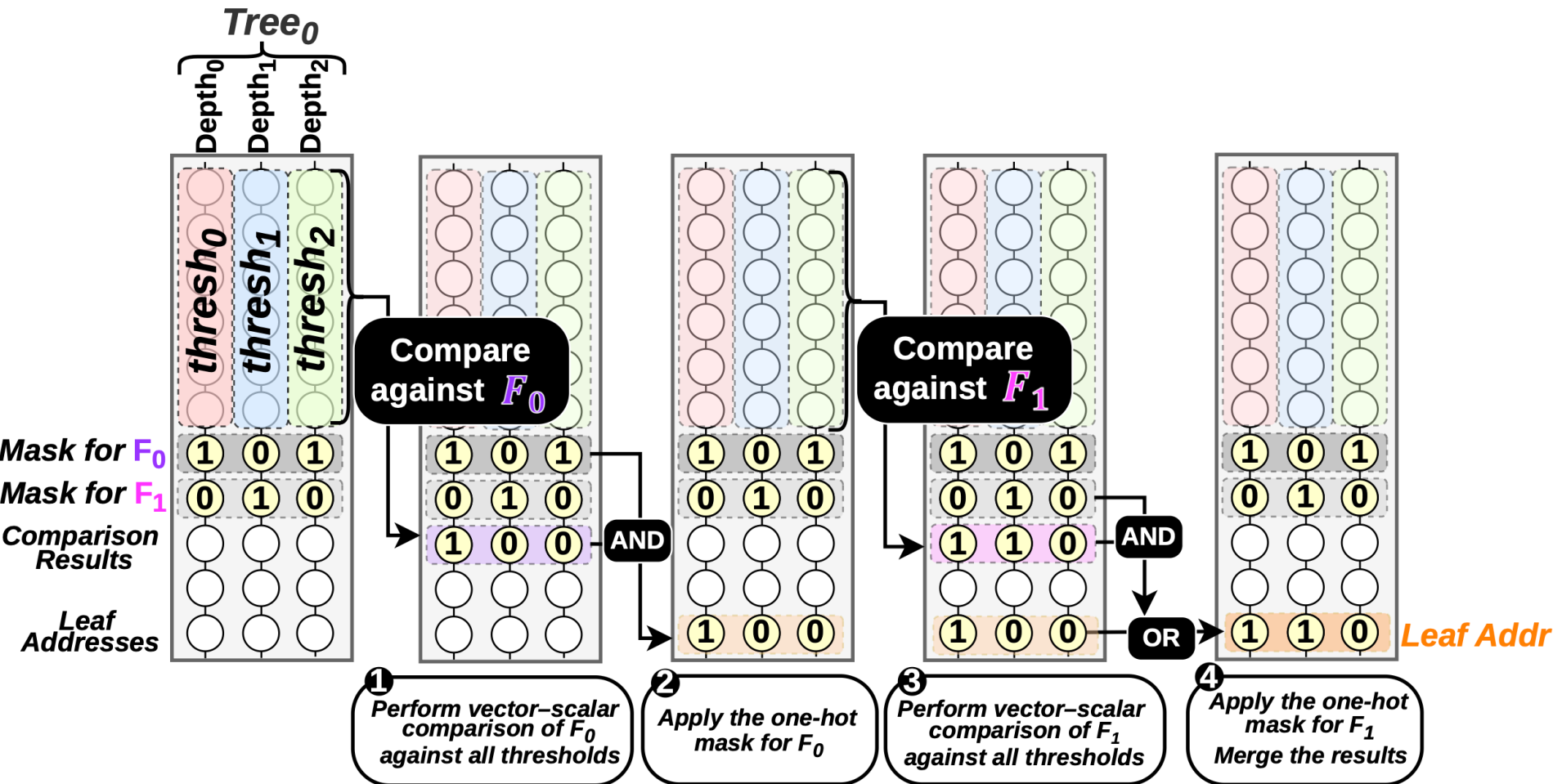


Within **1.5× memory footprint** of CPU: up to **58× (44× on average)** higher throughput over CPU

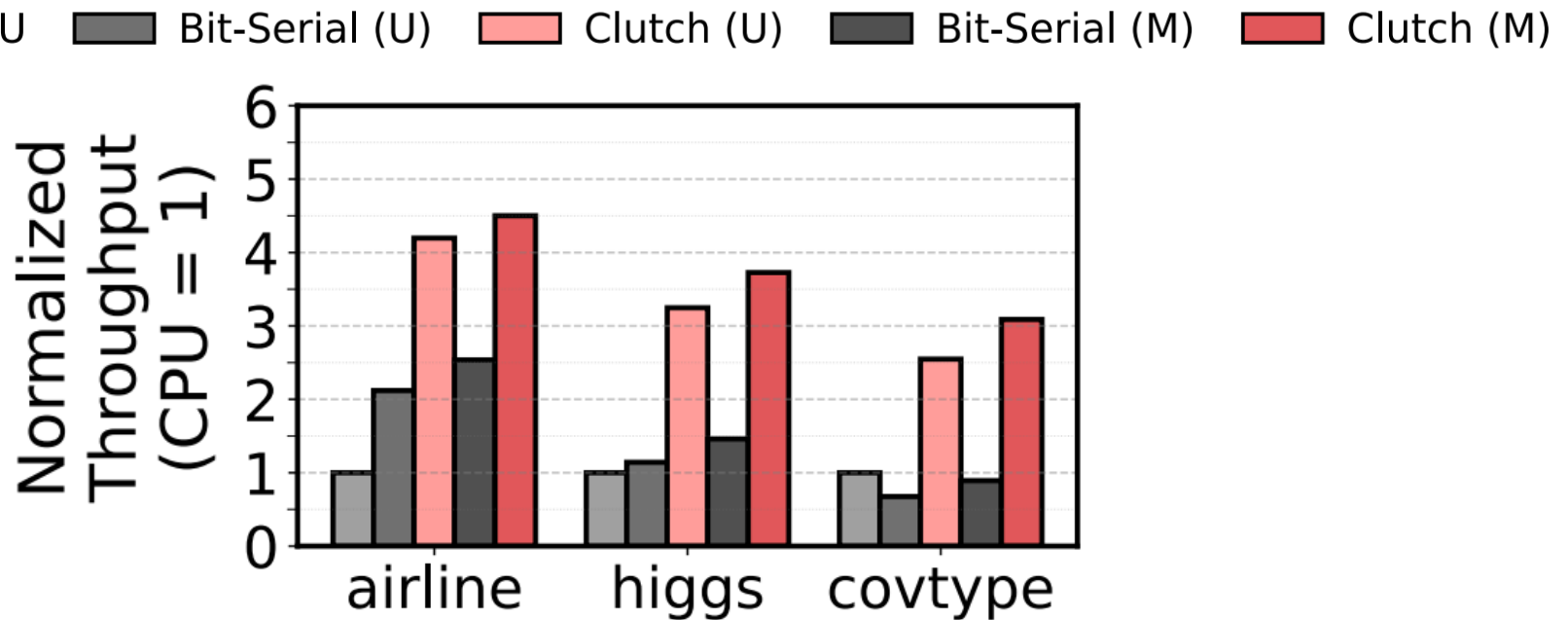
Decision Trees



Decision Tree Inference on PuD



Results: Decision Tree Inference



Clutch provides up to **4.5×** (**3.5× on average**) higher throughput over CPU, and up to **3.8×** (**2.2× on average**) over bit-serial PuD

Applications of Vector–Scalar Comparison

- Decision Tree Inference
 - Delivers state-of-the-art accuracy on **tabular data** with **low inference cost**
 - Particularly important for **edge AI applications**, including anomaly detection, remote sensing, and IoT analytics
- Predicate Evaluation
 - A fundamental operation in database query processing
 - Filters irrelevant data early, reducing data movement and **accelerating downstream operations**