Computer Architecture

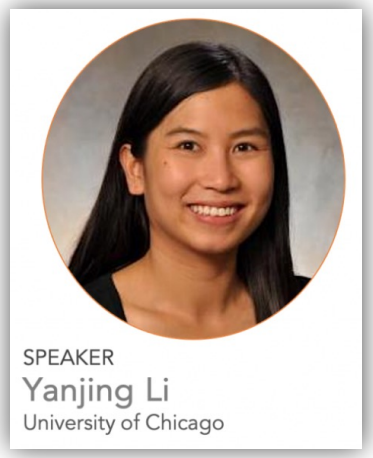
Lecture 5: Processing using Memory

Geraldo F. Oliveira
Prof. Onur Mutlu
ETH Zürich

Fall 2023

12 October 2023

SAFARI Live Seminar: Yanjing Li – 17.10



- **SAFARI Live Seminar**: Yanjing Li, Oct. 17 2023
  - Assistant Professor in the Department of Computer Science at the University of Chicago

- **Title:** How does one bit-flip corrupt an entire deep neural network, and what to do about it
  - In-depth resilience study targeting DNN workloads and hardware failures of deep learning accelerator systems
  - Lightweight yet highly effective techniques to mitigate hardware failures in deep learning accelerator systems.

- **Where:** Livestream on YouTube at 6:00 p.m.
  - https://www.youtube.com/watch?v=ZHBMbSPcddo

Sub-Agenda: In-Memory Computation

- Major Trends Affecting Main Memory
- **The Need for Intelligent Memory Controllers**
  - **Bottom Up: Push from Circuits and Devices**
  - **Top Down: Pull from Systems and Applications**
- Processing in Memory: Two Directions
  - Processing using Memory
  - Processing near Memory
- How to Enable Adoption of Processing in Memory
- Conclusion

Three Key Systems Trends

1. Data access is a major bottleneck

- Applications are increasingly data hungry

2. Energy consumption is a key limiter

3. Data movement energy dominates compute

- Especially true for off-chip to on-chip movement

Observation and Opportunity

- High latency and high energy caused by data movement
  - Long, energy-hungry interconnects
  - Energy-hungry electrical interfaces
  - Movement of large amounts of data

- Opportunity: Minimize data movement by performing computation directly (near) where the data resides
  - Processing in memory (PIM)
  - In-memory computation/processing
  - Near-data processing (NDP)
  - General concept applicable to any data storage & movement unit (caches, SSDs, main memory, network, controllers)

Four Key Issues in Future Platforms

- Fundamentally Secure/Reliable/Safe Architectures
- Fundamentally Energy-Efficient Architectures
  - Memory-centric (Data-centric) Architectures
- Fundamentally Low-Latency Architectures
- Architectures for AI/ML, Genomics, Medicine, Health

Challenge and Opportunity for Future

High Performance,

Energy Efficient,

Sustainable

(All at the Same Time)

The Problem

Data access is the major performance and energy bottleneck

Our current

design principles

cause great energy waste

(and great performance loss)

The Problem

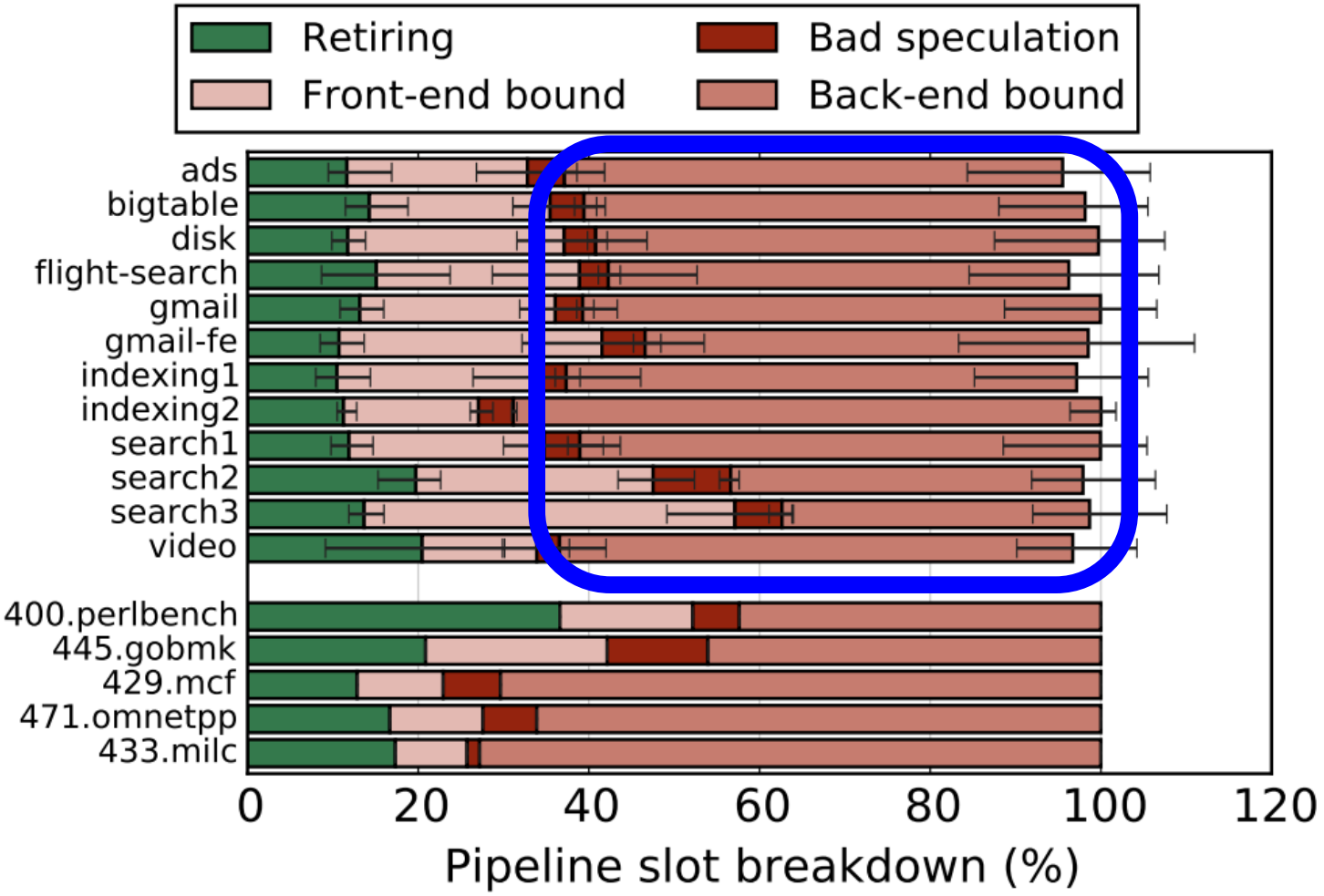
Processing of data

is performed

far away from the data

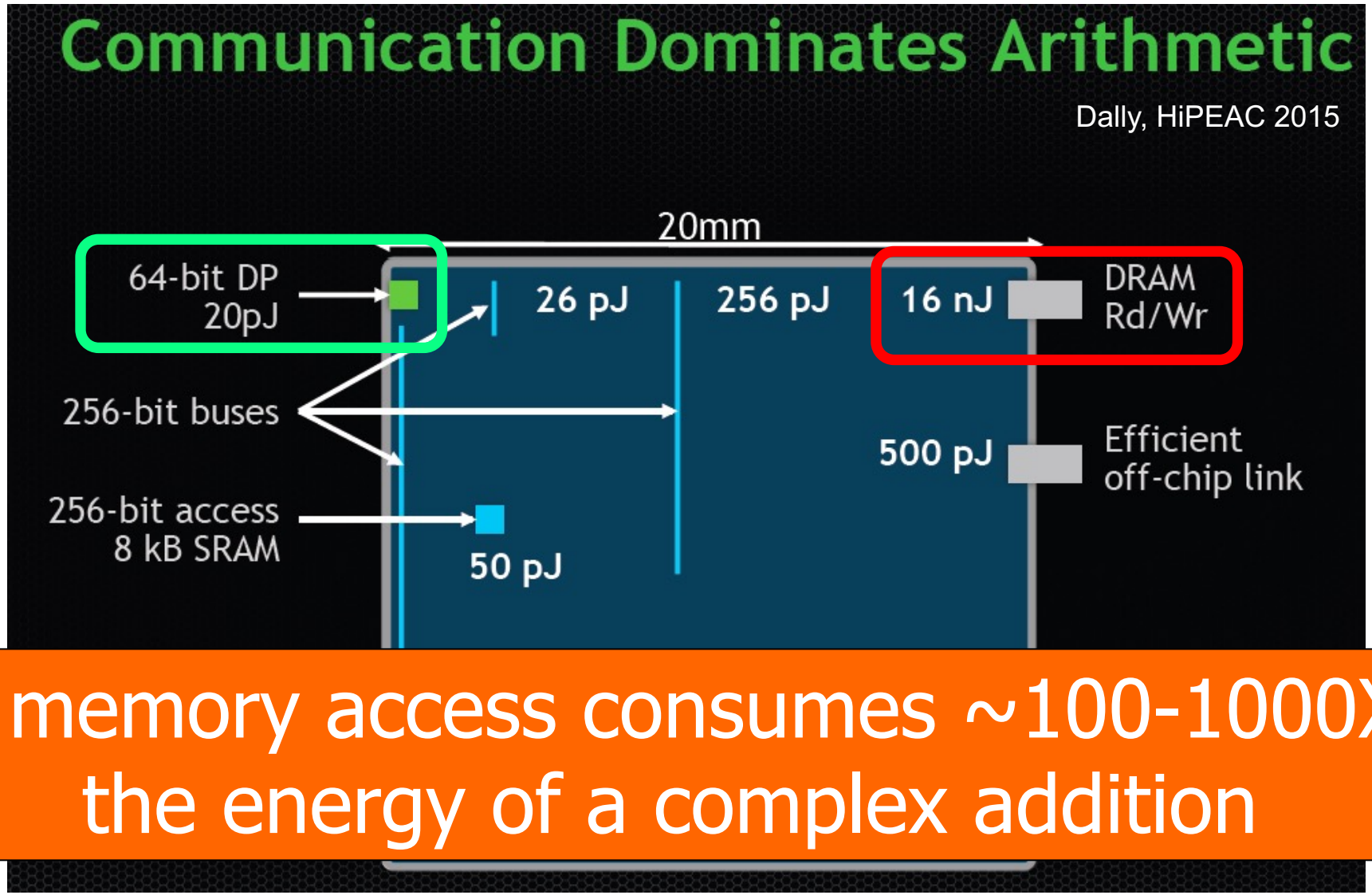
The Performance Perspective (Today)

- All of Google's Data Center Workloads (2015):



Kanev+, "Profiling a Warehouse-Scale Computer," ISCA 2015.

Data Movement vs. Computation Energy



A memory access consumes ~100-1000X the energy of a complex addition

Energy Waste in Mobile Devices

- Amirali Boroumand, Saugata Ghose, Youngsok Kim, Rachata Ausavarungnirun, Eric Shiu, Rahul Thakur, Daehyun Kim, Aki Kuusela, Allan Knies, Parthasarathy Ranganathan, and Onur Mutlu, **"Google Workloads for Consumer Devices: Mitigating Data Movement Bottlenecks"** *Proceedings of the 23rd International Conference on Architectural Support for Programming Languages and Operating Systems (**ASPLOS**)*, Williamsburg, VA, USA, March 2018.

62.7% of the total system energy is spent on data movement

Google Workloads for Consumer Devices: Mitigating Data Movement Bottlenecks

Amirali Boroumand[1] Saugata Ghose[1] Youngsok Kim[2]
Rachata Ausavarungnirun[1] Eric Shiu[3] Rahul Thakur[3] Daehyun Kim[4,3]
Aki Kuusela[3] Allan Knies[3] Parthasarathy Ranganathan[3] Onur Mutlu[5,1]

We Need to Think Differently from the Past Approaches

We Need A Paradigm Shift To …

- Enable computation with minimal data movement
- Compute where it makes sense (where data resides)
- Make computing architectures more data-centric

Processing Data Where It Makes Sense

PIM Review and Open Problems

A Modern Primer on Processing in Memory

Onur Mutlu[a,b], Saugata Ghose[b,c], Juan Gómez-Luna[a], Rachata Ausavarungnirun[d]

SAFARI Research Group

[a] *ETH Zürich*
[b] *Carnegie Mellon University*
[c] *University of Illinois at Urbana-Champaign*
[d] *King Mongkut's University of Technology North Bangkok*

Onur Mutlu, Saugata Ghose, Juan Gomez-Luna, and Rachata Ausavarungnirun,
"A Modern Primer on Processing in Memory"
Invited Book Chapter in ***Emerging Computing: From Devices to Systems - Looking Beyond Moore and Von Neumann***, Springer, to be published in 2021.

https://arxiv.org/pdf/1903.03988.pdf

A Modern Primer on Processing in Memory

Onur Mutlu[a,b], Saugata Ghose[b,c], Juan Gómez-Luna[a], Rachata Ausavarungnirun[d]

SAFARI Research Group

[a]*ETH Zürich*
[b]*Carnegie Mellon University*
[c]*University of Illinois at Urbana-Champaign*
[d]*King Mongkut's University of Technology North Bangkok*

Abstract

Modern computing systems are overwhelmingly designed to move data to computation. This design choice goes directly against at least three key trends in computing that cause performance, scalability and energy bottlenecks: (1) data access is a key bottleneck as many important applications are increasingly data-intensive, and memory bandwidth and energy do not scale well, (2) energy consumption is a key limiter in almost all computing platforms, especially server and mobile systems, (3) data movement, especially off-chip to on-chip, is very expensive in terms of bandwidth, energy and latency, much more so than computation. These trends are especially severely-felt in the data-intensive server and energy-constrained mobile systems of today.

At the same time, conventional memory technology is facing many technology scaling challenges in terms of reliability, energy, and performance. As a result, memory system architects are open to organizing memory in different ways and making it more intelligent, at the expense of higher cost. The emergence of 3D-stacked memory plus logic, the adoption of error correcting codes inside the latest DRAM chips, proliferation of different main memory standards and chips, specialized for different purposes (e.g., graphics, low-power, high bandwidth, low latency), and the necessity of designing new solutions to serious reliability and security issues, such as the RowHammer phenomenon, are an evidence of this trend.

This chapter discusses recent research that aims to practically enable computation close to data, an approach we call *processing-in-memory* (PIM). PIM places computation mechanisms in or near where the data is stored (i.e., inside the memory chips, in the logic layer of 3D-stacked memory, or in the memory controllers), so that data movement between the computation units and memory is reduced or eliminated. While the general idea of PIM is not new, we discuss motivating trends in applications as well as memory circuits/technology that greatly exacerbate the need for enabling it in modern computing systems. We examine at least two promising new approaches to designing PIM systems to accelerate important data-intensive applications: (1) *processing using memory* by exploiting analog operational properties of DRAM chips to perform massively-parallel operations in memory, with low-cost changes, (2) *processing near memory* by exploiting 3D-stacked memory technology design to provide high memory bandwidth and low memory latency to in-memory logic. In both approaches, we describe and tackle relevant cross-layer research, design, and adoption challenges in devices, architecture, systems, and programming models. Our focus is on the development of in-memory processing designs that can be adopted in real computing platforms at low cost. We conclude by discussing work on solving key challenges to the practical adoption of PIM.

Keywords: memory systems, data movement, main memory, processing-in-memory, near-data processing, computation-in-memory, processing using memory, processing near memory, 3D-stacked memory, non-volatile memory, energy efficiency, high-performance computing, computer architecture, computing paradigm, emerging technologies, memory scaling, technology scaling, dependable systems, robust systems, hardware security, system security, latency, low-latency computing

Contents

1. Introduction

Main memory, built using the Dynamic Random Access Memory (DRAM) technology, is a major component in nearly all computing systems, including servers, cloud platforms, mobile/embedded devices, and sensor systems. Across all of these systems, the data working set sizes of modern applications are rapidly growing, while the need for fast analysis of such data is increasing. Thus, main memory is becoming an increasingly significant bottleneck across a wide variety of computing systems and applications [1–26]. Alleviating the main memory bottleneck requires the memory capacity, energy, cost, and performance to all scale in an efficient manner across technology generations. Unfortunately, it has become increasingly difficult in recent years, especially the past decade, to scale all of these dimensions [1, 2, 27–59], and thus the main memory bottleneck has been worsening.

A major reason for the main memory bottleneck is the high energy and latency cost associated with *data movement*. In modern computers, to perform any operation on data that resides in main memory, the processor must retrieve the data from main memory. This requires the memory controller to issue commands to a DRAM module across a relatively slow and power-hungry off-chip bus (known as the *memory channel*). The DRAM module sends the requested data across the memory channel, after which the data is placed in the caches and registers. The CPU can perform computation on the data once the data is in its registers. Data movement from the DRAM to the CPU incurs long latency and consumes a significant amount of energy [7–9, 60–64]. These costs are often exacerbated by the fact that much of the data brought into the caches is *not reused* by the CPU [62, 63, 65, 66], providing little benefit in return for the high latency and energy cost.

The cost of data movement is a fundamental issue with the *processor-centric* nature of contemporary computer systems. The CPU is considered to be the master in the system, and computation is performed only in the processor (and accelerators). In contrast, data storage and communication units, including the main memory, are treated as unintelligent workers that are incapable of computation. As a result of this processor-centric design paradigm, data moves a lot in the system between the computation units and communication/storage units so that computation can be done on it. With the increasingly *data-centric* nature of contemporary and emerging applications, the processor-centric design paradigm leads to great inefficiency in performance, energy and cost. For example, most of the real estate within a single compute

Two PIM Approaches

5.2. Two Approaches: Processing Using Memory (PUM) vs. Processing Near Memory (PNM)

Many recent works take advantage of the memory technology innovations that we discuss in Section 5.1 to enable and implement PIM. We find that these works generally take one of two approaches, which are categorized in Table 1: (1) *processing using memory* or (2) *processing near memory*. We briefly describe each approach here. Sections 6 and 7 will provide example approaches and more detail for both.

Table 1: Summary of enabling technologies for the two approaches to PIM used by recent works. Adapted from [341] and extended.

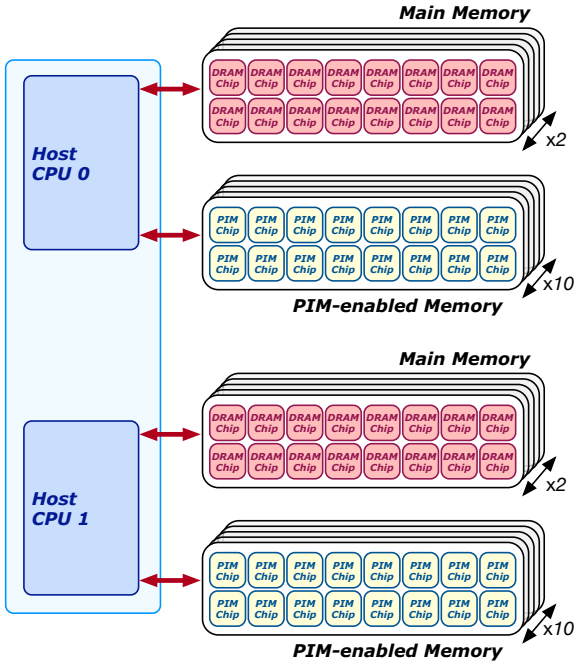
Approach	Example Enabling Technologies
Processing Using Memory	SRAM DRAM Phase-change memory (PCM) Magnetic RAM (MRAM) Resistive RAM (RRAM)/memristors
Processing Near Memory	Logic layers in 3D-stacked memory Silicon interposers Logic in memory controllers Logic in memory chips (e.g., near bank) Logic in memory modules Logic near caches Logic near/in storage devices

Onur Mutlu, Saugata Ghose, Juan Gomez-Luna, and Rachata Ausavarungnirun,
"A Modern Primer on Processing in Memory"
Invited Book Chapter in ***Emerging Computing: From Devices to Systems - Looking Beyond Moore and Von Neumann***,
Springer, to be published in 2021.
[Tutorial Video on "Memory-Centric Computing Systems" (1 hour 51 minutes)]

Processing in Memory: Two Approaches

1. Processing near Memory
2. Processing using Memory

2,560-DPU Processing-in-Memory System



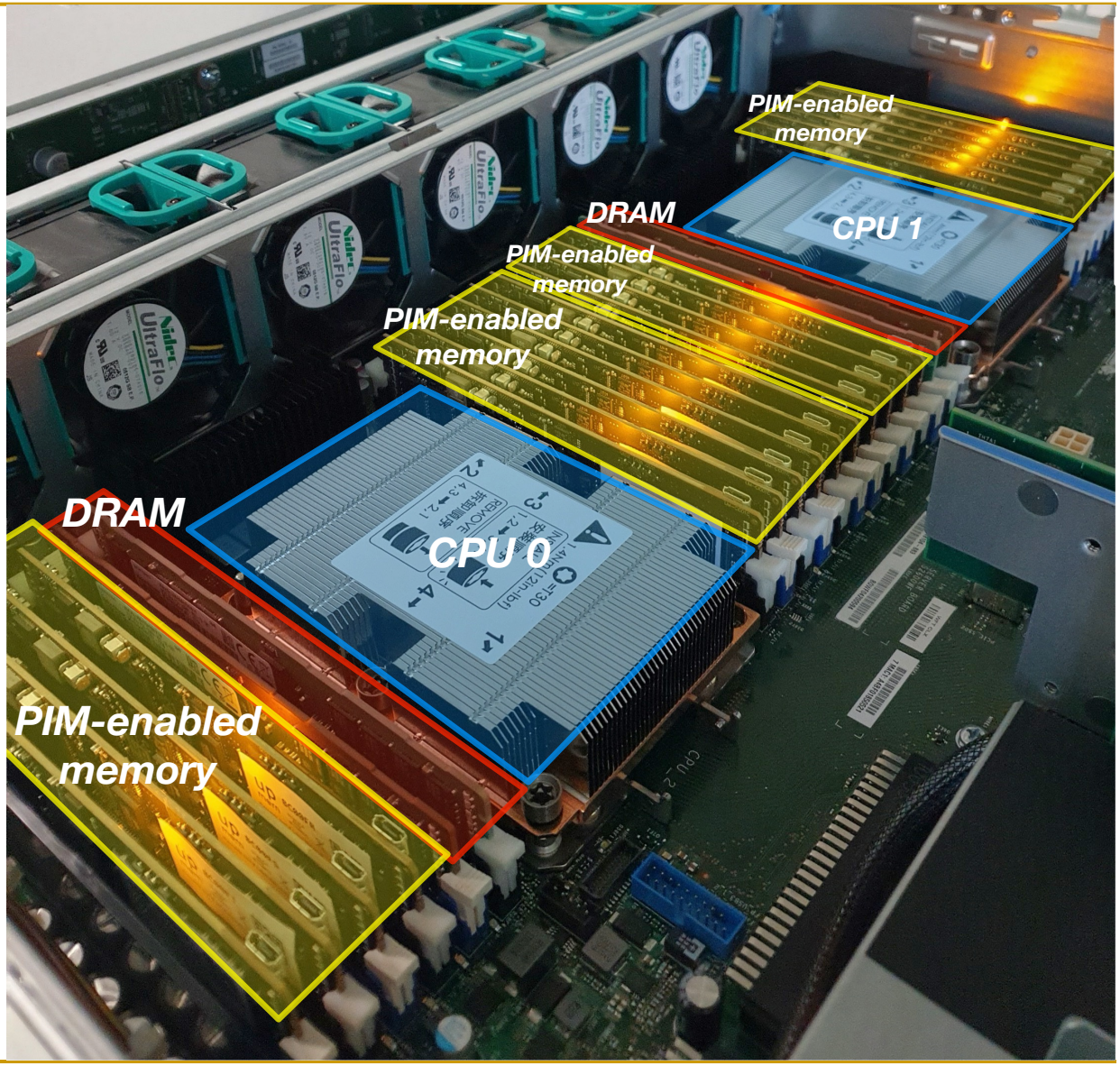
Benchmarking a New Paradigm: An Experimental Analysis of a Real Processing-in-Memory Architecture

JUAN GÓMEZ-LUNA, ETH Zürich, Switzerland
IZZAT EL HAJJ, American University of Beirut, Lebanon
IVAN FERNANDEZ, ETH Zürich, Switzerland and University of Malaga, Spain
CHRISTINA GIANNOULA, ETH Zürich, Switzerland and NTUA, Greece
GERALDO F. OLIVEIRA, ETH Zürich, Switzerland
ONUR MUTLU, ETH Zürich, Switzerland

Many modern workloads, such as neural networks, databases, and graph processing, are fundamentally memory-bound. For such workloads, the data movement between main memory and CPU cores imposes a significant overhead in terms of both latency and energy. A major reason is that this communication happens through a narrow bus with high latency and limited bandwidth, and the low data reuse in memory-bound workloads is insufficient to amortize the cost of main memory access. Fundamentally addressing this *data movement bottleneck* requires a paradigm where the memory system assumes an active role in computing by integrating processing capabilities. This paradigm is known as *processing-in-memory* (*PIM*).

Recent research explores different forms of PIM architectures, motivated by the emergence of new 3D-stacked memory technologies that integrate memory with a logic layer where processing elements can be easily placed. Past works evaluate these architectures in simulation or, at best, with simplified hardware prototypes. In contrast, the UPMEM company has designed and manufactured the first publicly-available real-world PIM architecture. The UPMEM PIM architecture combines traditional DRAM memory arrays with general-purpose in-order cores, called *DRAM Processing Units* (*DPUs*), integrated in the same chip.

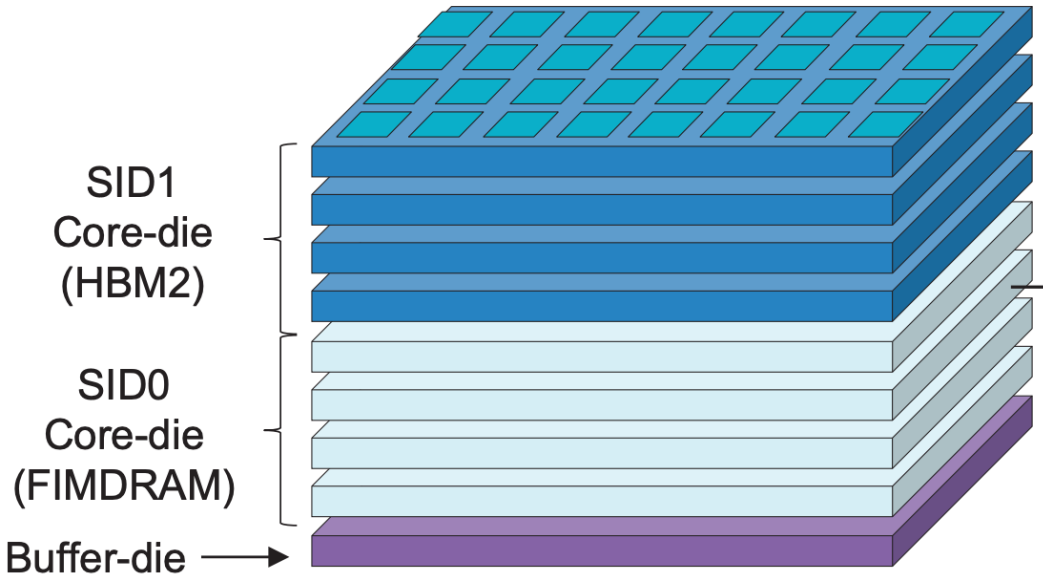
This paper provides the first comprehensive analysis of the first publicly-available real-world PIM architecture. We make two key contributions. First, we conduct an experimental characterization of the UPMEM-based PIM system using microbenchmarks to assess various architecture limits such as compute throughput and memory bandwidth, yielding new insights. Second, we present *PrIM* (*Processing-In-Memory benchmarks*), a benchmark suite of 16 workloads from different application domains (e.g., dense/sparse linear algebra, databases, data analytics, graph processing, neural networks, bioinformatics, image processing), which we identify as memory-bound. We evaluate the performance and scaling characteristics of PrIM benchmarks on the UPMEM PIM architecture, and compare their performance and energy consumption to their state-of-the-art CPU and GPU counterparts. Our extensive evaluation conducted on two real UPMEM-based PIM systems with 640 and 2,556 DPUs provides new insights about suitability of different workloads to the PIM system, programming recommendations for software designers, and suggestions and hints for hardware and architecture designers of future PIM systems.



https://arxiv.org/pdf/2105.03814.pdf

Samsung Function-in-Memory DRAM (2021)

- FIMDRAM based on HBM2



[3D Chip Structure of HBM with FIMDRAM]

Chip Specification
128DQ / 8CH / 16 banks / BL4
32 PCU blocks (1 FIM block/2 banks)
1.2 TFLOPS (4H)
FP16 ADD / Multiply (MUL) / Multiply-Accumulate (MAC) / Multiply-and- Add (MAD)

ISSCC 2021 / SESSION 25 / DRAM / 25.4

25.4 A 20nm 6GB Function-In-Memory DRAM, Based on HBM2 with a 1.2TFLOPS Programmable Computing Unit Using Bank-Level Parallelism, for Machine Learning Applications

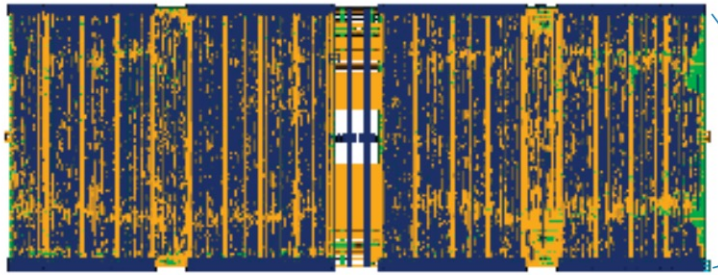
Young-Cheon Kwon[1], Suk Han Lee[1], Jaehoon Lee[1], Sang-Hyuk Kwon[1], Je Min Ryu[1], Jong-Pil Son[1], Seongil O[1], Hak-Soo Yu[1], Haesuk Lee[1], Soo Young Kim[1], Youngmin Cho[1], Jin Guk Kim[1], Jongyoon Choi[1], Hyun-Sung Shin[1], Jin Kim[1], BengSeng Phuah[1], HyoungMin Kim[1], Myeong Jun Song[1], Ahn Choi[1], Daeho Kim[1], SooYoung Kim[1], Eun-Bong Kim[1], David Wang[2], Shinhaeng Kang[1], Yuhwan Ro[3], Seungwoo Seo[3], JoonHo Song[3], Jaeyoun Youn[1], Kyomin Sohn[1], Nam Sung Kim[1]

[1]Samsung Electronics, Hwaseong, Korea
[2]Samsung Electronics, San Jose, CA
[3]Samsung Electronics, Suwon, Korea

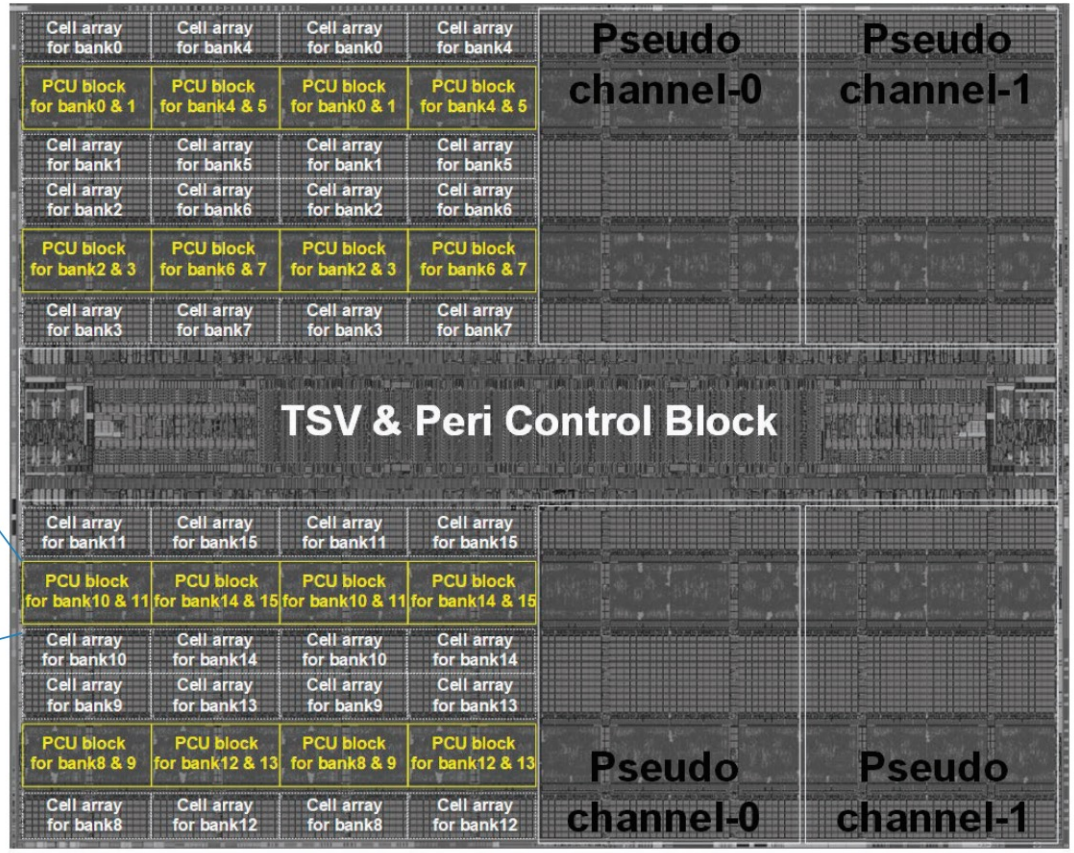
Samsung Function-in-Memory DRAM (2021)

Chip Implementation

- Mixed design methodology to implement FIMDRAM
  - Full-custom + Digital RTL



[Digital RTL design for PCU block]



ISSCC 2021 / SESSION 25 / DRAM / 25.4

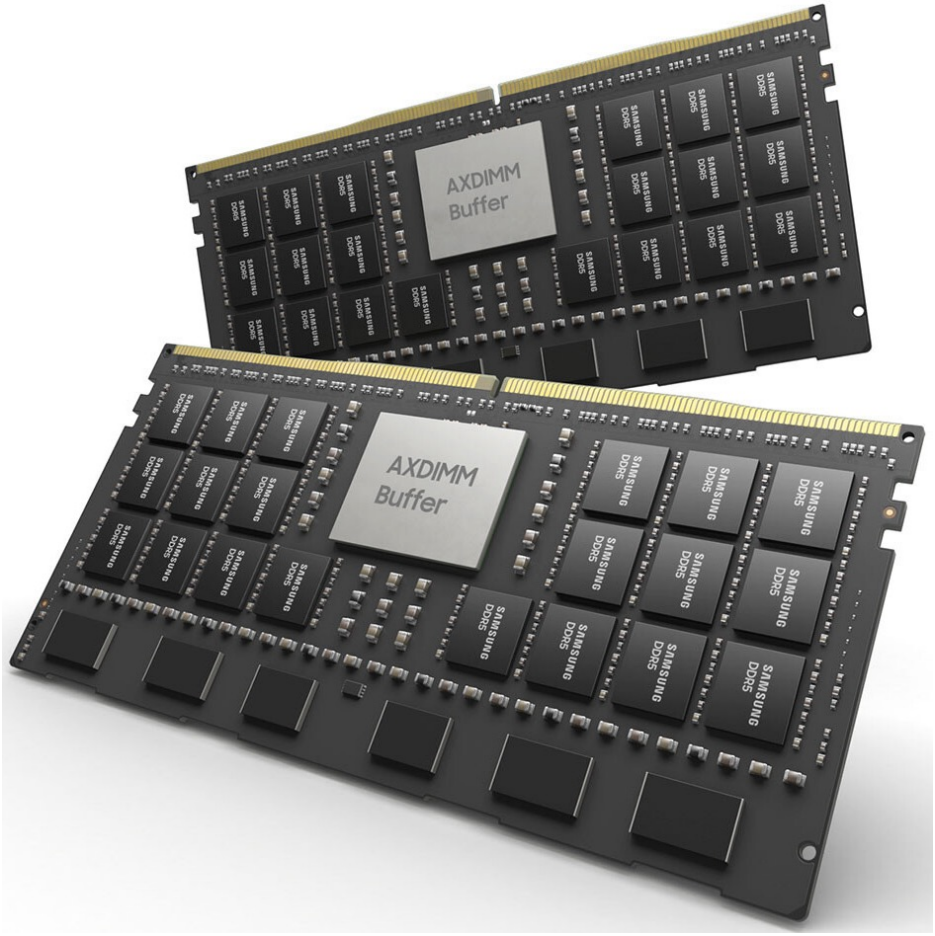
25.4 A 20nm 6GB Function-In-Memory DRAM, Based on HBM2 with a 1.2TFLOPS Programmable Computing Unit Using Bank-Level Parallelism, for Machine Learning Applications

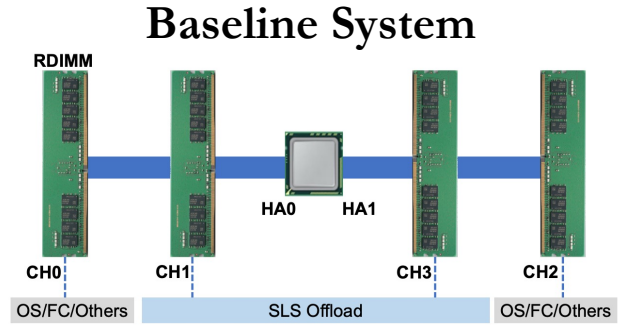
Young-Cheon Kwon[1], Suk Han Lee[1], Jaehoon Lee[1], Sang-Hyuk Kwon[1], Je Min Ryu[1], Jong-Pil Son[1], Seongil O[1], Hak-Soo Yu[1], Haesuk Lee[1], Soo Young Kim[1], Youngmin Cho[1], Jin Guk Kim[1], Jongyoon Choi[1], Hyun-Sung Shin[1], Jin Kim[1], BengSeng Phuah[1], HyoungMin Kim[1], Myeong Jun Song[1], Ahn Choi[1], Daeho Kim[1], SooYoung Kim[1], Eun-Bong Kim[1], David Wang[2], Shinhaeng Kang[1], Yuhwan Ro[3], Seungwoo Seo[3], JoonHo Song[3], Jaeyoun Youn[1], Kyomin Sohn[1], Nam Sung Kim[1]

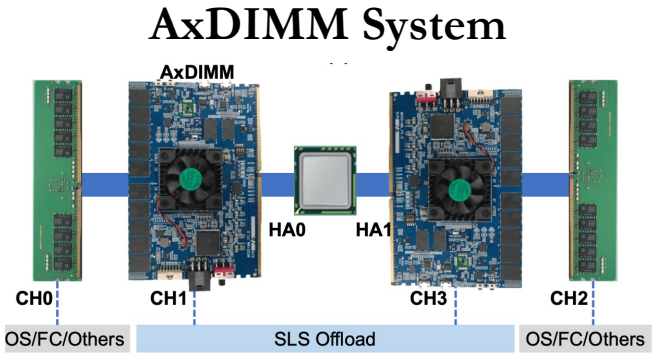
[1]Samsung Electronics, Hwaseong, Korea
[2]Samsung Electronics, San Jose, CA
[3]Samsung Electronics, Suwon, Korea

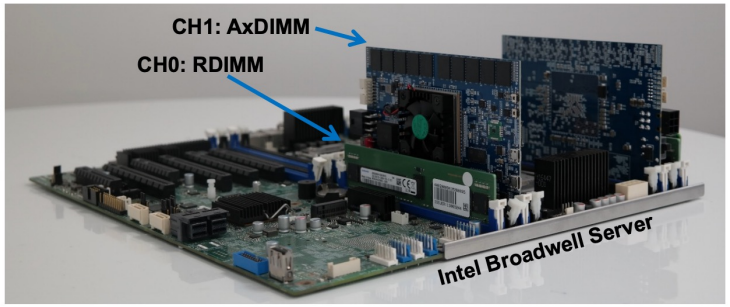
Samsung AxDIMM (2021)

- DIMM-based PIM
  - DLRM recommendation system







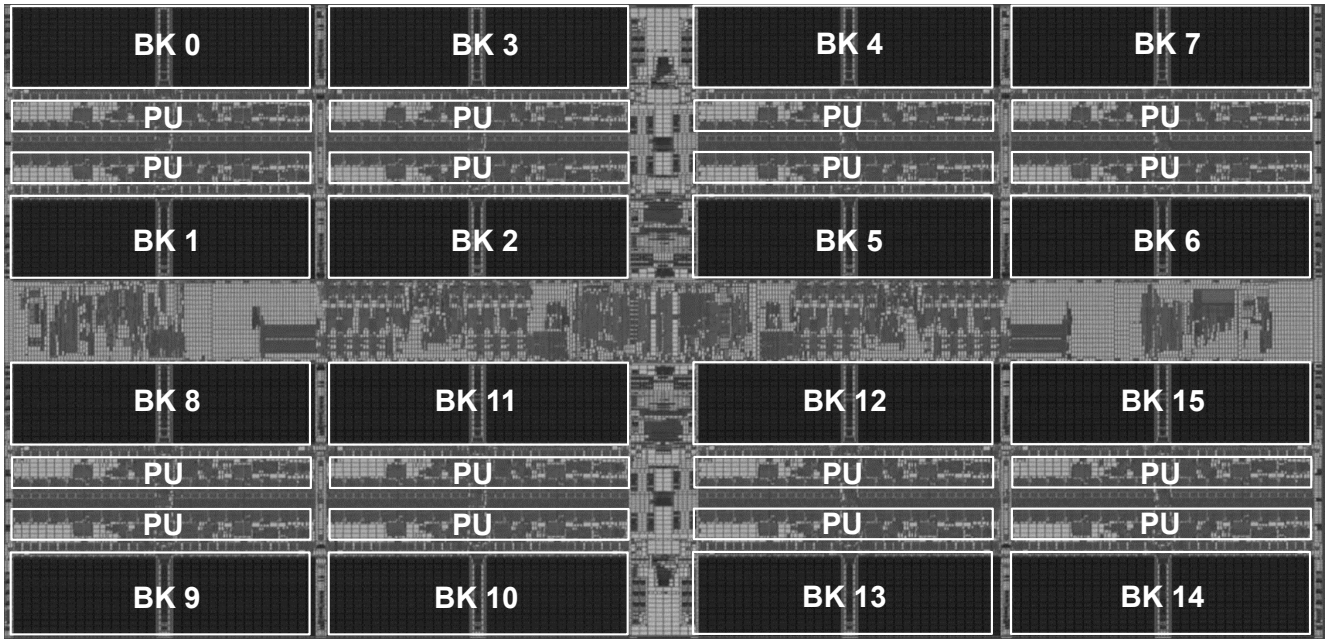


Ke et al. "Near-Memory Processing in Action: Accelerating Personalized Recommendation with AxDIMM", IEEE Micro (2021)

SK Hynix AiM: Chip Implementation (2022)

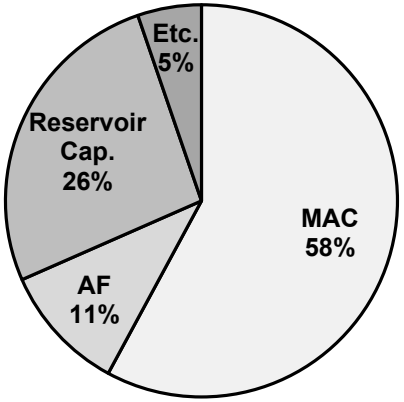
- 4 Gb AiM die with 16 processing units (PUs)

AiM Die Photograph



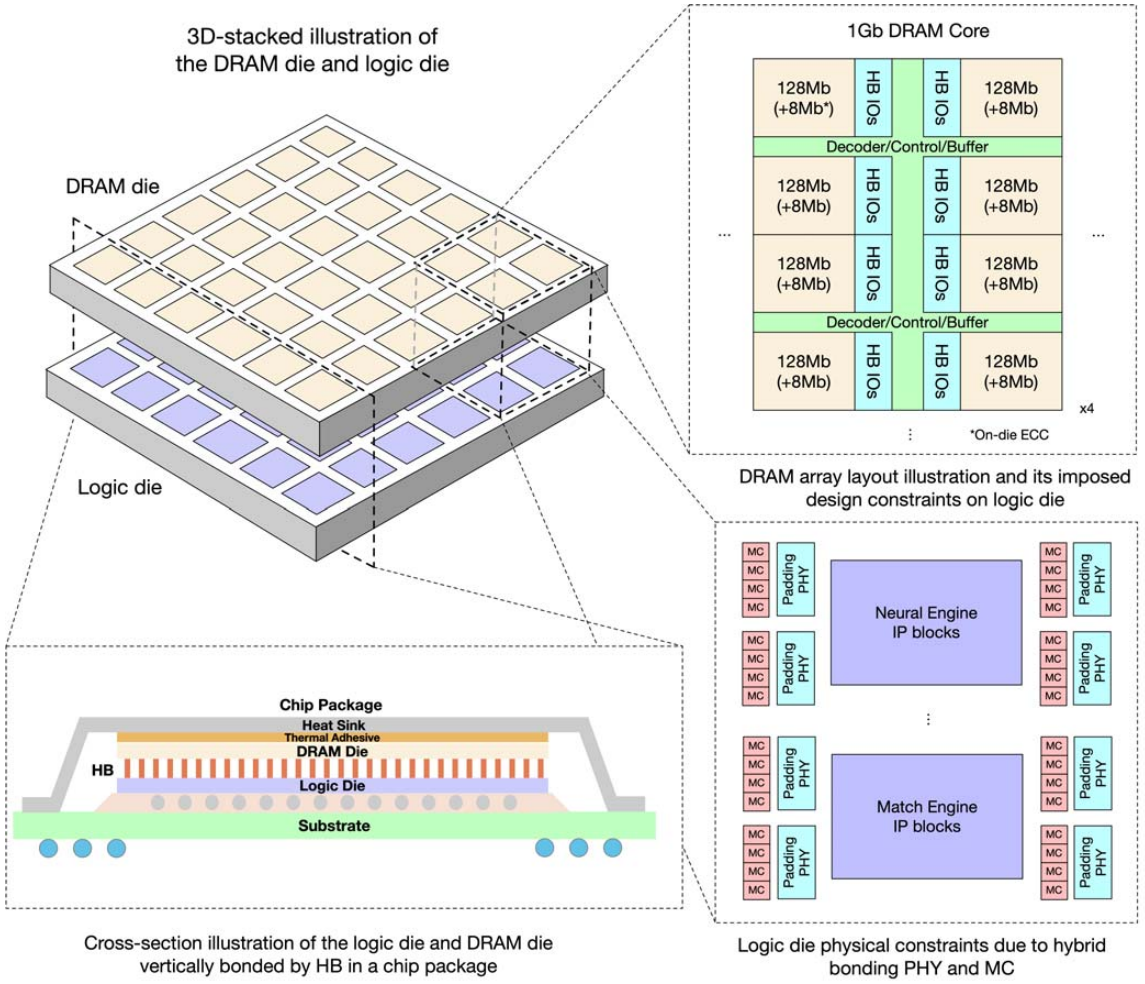
1 Process Unit (PU) Area

Total	0.19mm^2
MAC	0.11mm^2
Activation Function (AF)	0.02mm^2
Reservoir Cap.	0.05mm^2
Etc.	0.01mm^2



Alibaba HB-PNM: Overall Architecture (2022)

- 3D-stacked logic die and DRAM die vertically bonded by hybrid bonding (HB)



Processing in Memory: Two Approaches

1. Processing near Memory
2. Processing using Memory

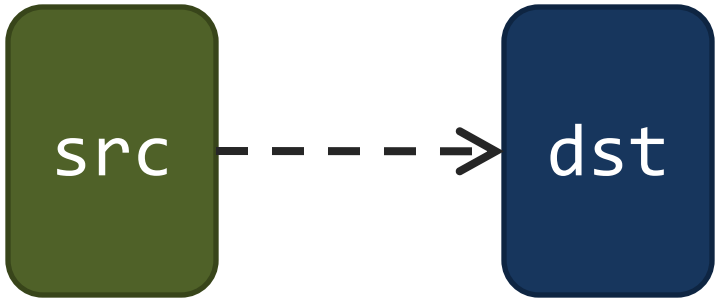
Approach 2: Processing Using Memory

- Take advantage of operational principles of memory to perform bulk data movement and computation in memory
  - Can exploit internal connectivity to move data
  - Can exploit analog computation capability
  - …

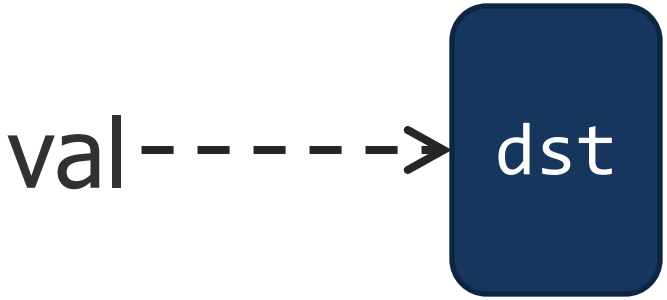
- Examples: RowClone, In-DRAM AND/OR, Gather/Scatter DRAM
  - RowClone: Fast and Efficient In-DRAM Copy and Initialization of Bulk Data (Seshadri et al., MICRO 2013)
  - Fast Bulk Bitwise AND and OR in DRAM (Seshadri et al., IEEE CAL 2015)
  - Gather-Scatter DRAM: In-DRAM Address Translation to Improve the Spatial Locality of Non-unit Strided Accesses (Seshadri et al., MICRO 2015)
  - "Ambit: In-Memory Accelerator for Bulk Bitwise Operations Using Commodity DRAM Technology" (Seshadri et al., MICRO 2017)

Starting Simple: Data Copy and Initialization

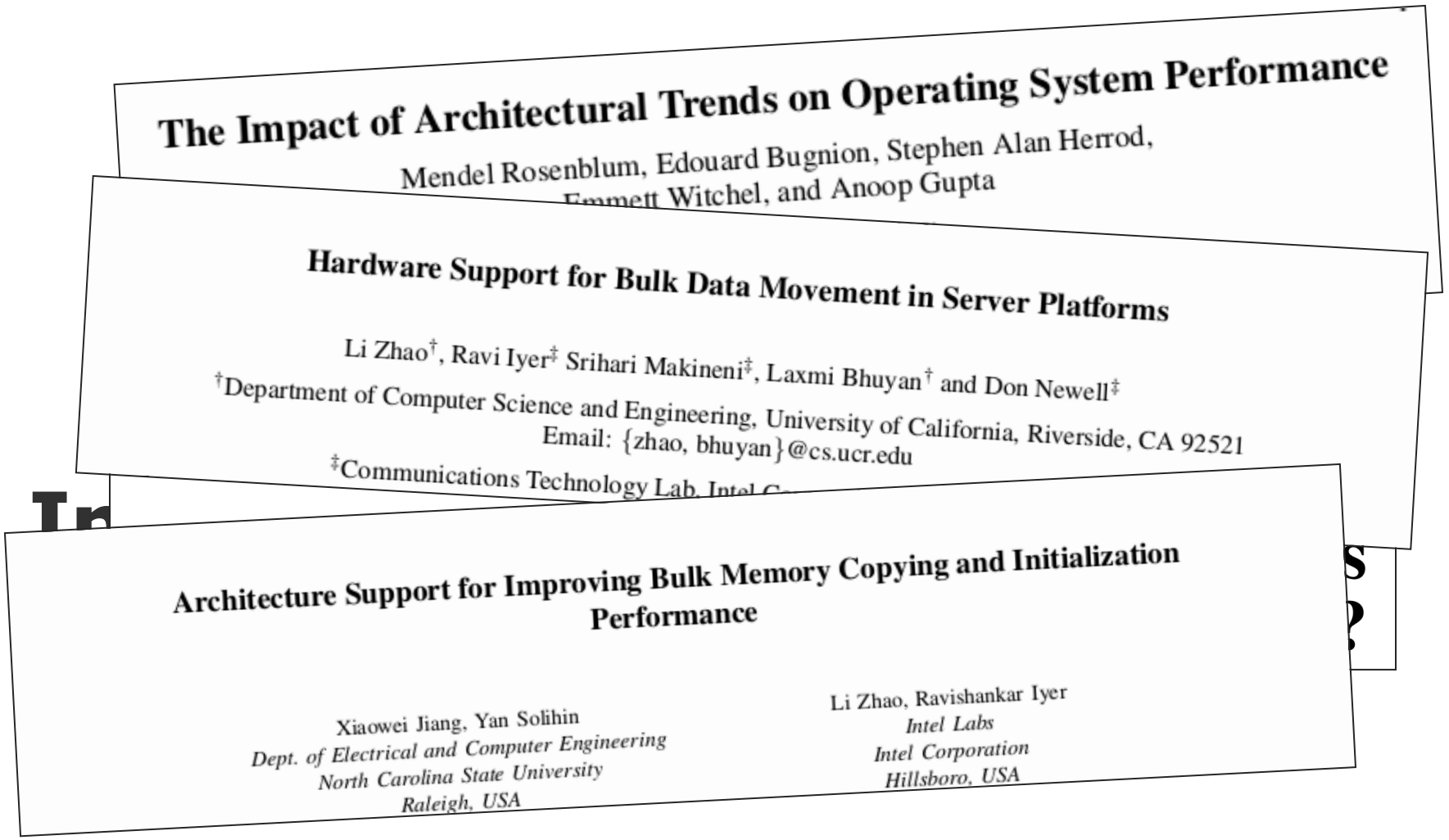
Bulk Data Copy



Bulk Data Initialization



Bulk Data Copy and Initialization



The Impact of Architectural Trends on Operating System Performance

Mendel Rosenblum, Edouard Bugnion, Stephen Alan Herrod,
Emmett Witchel, and Anoop Gupta

Hardware Support for Bulk Data Movement in Server Platforms

Li Zhao[†], Ravi Iyer[‡] Srihari Makineni[‡], Laxmi Bhuyan[†] and Don Newell[‡]

[†]Department of Computer Science and Engineering, University of California, Riverside, CA 92521
Email: {zhao, bhuyan}@cs.ucr.edu

[‡]Communications Technology Lab, Intel C...

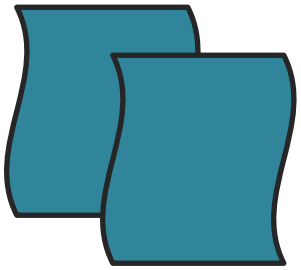
Architecture Support for Improving Bulk Memory Copying and Initialization Performance

Xiaowei Jiang, Yan Solihin
Dept. of Electrical and Computer Engineering
North Carolina State University
Raleigh, USA

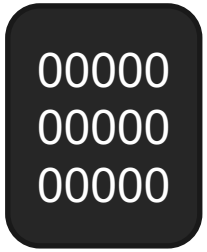
Li Zhao, Ravishankar Iyer
Intel Labs
Intel Corporation
Hillsboro, USA

Starting Simple: Data Copy and Initialization

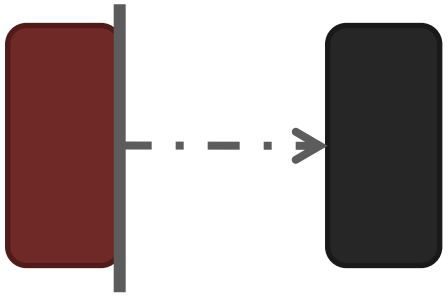
[redacted]



Forking

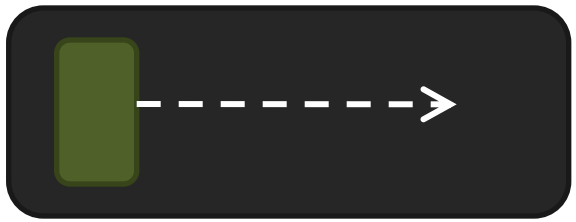


Zero initialization (e.g., security)



Checkpointing

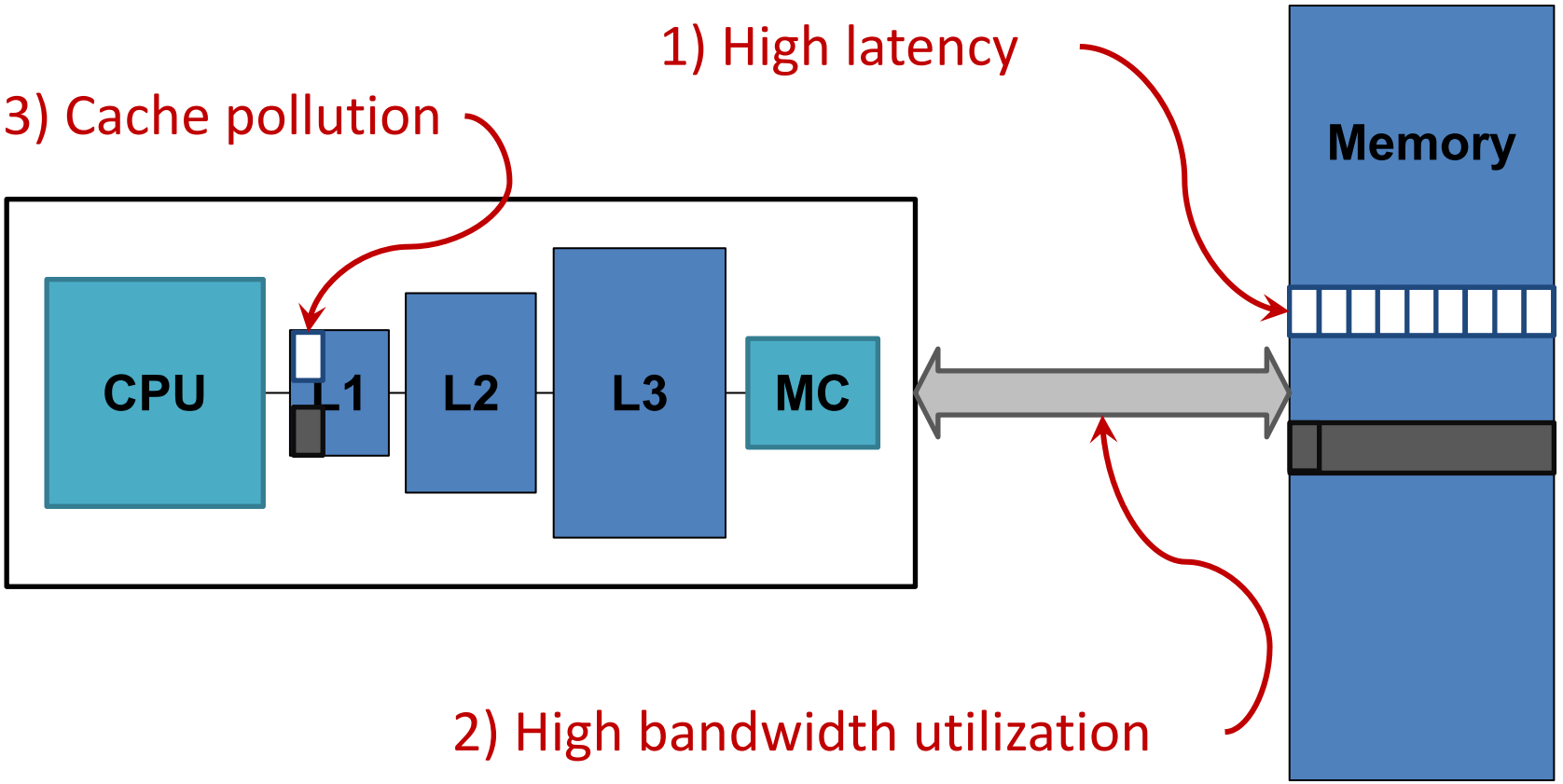
VM Cloning
Deduplication



Page Migration

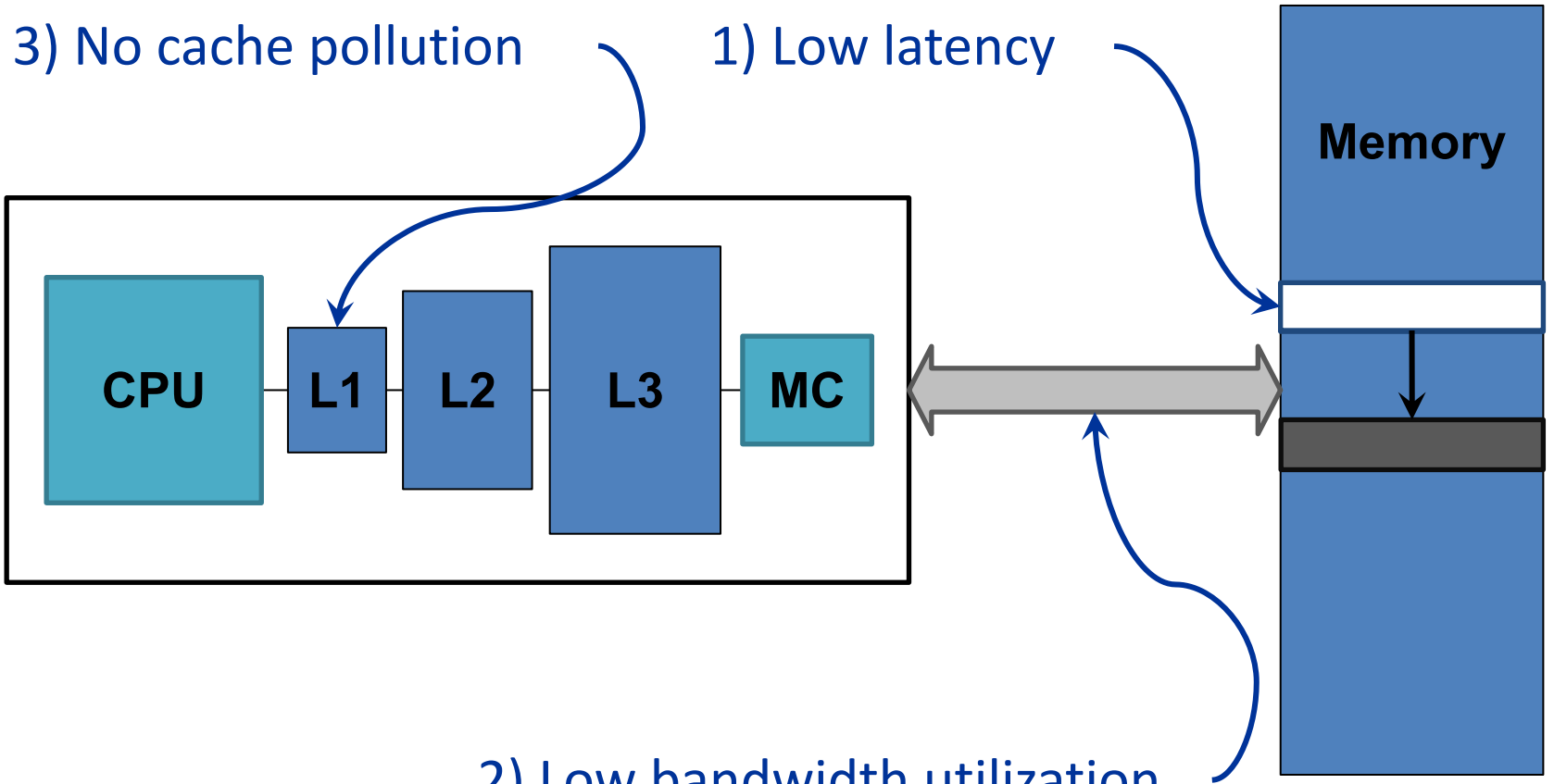
• • •
Many more

Today's Systems: Bulk Data Copy



1046ns, 3.6uJ (for 4KB page copy via DMA)

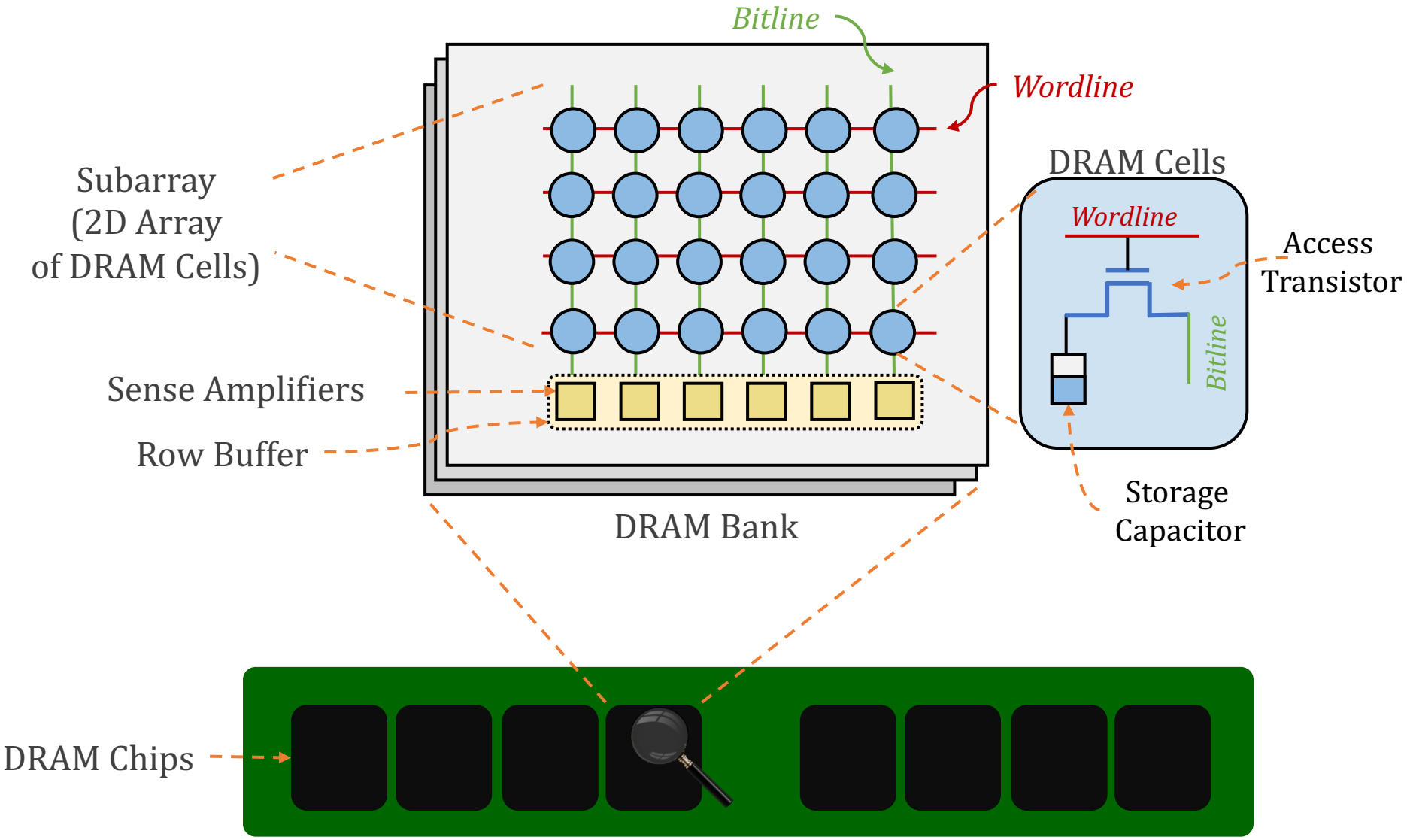
Future Systems: In-Memory Copy



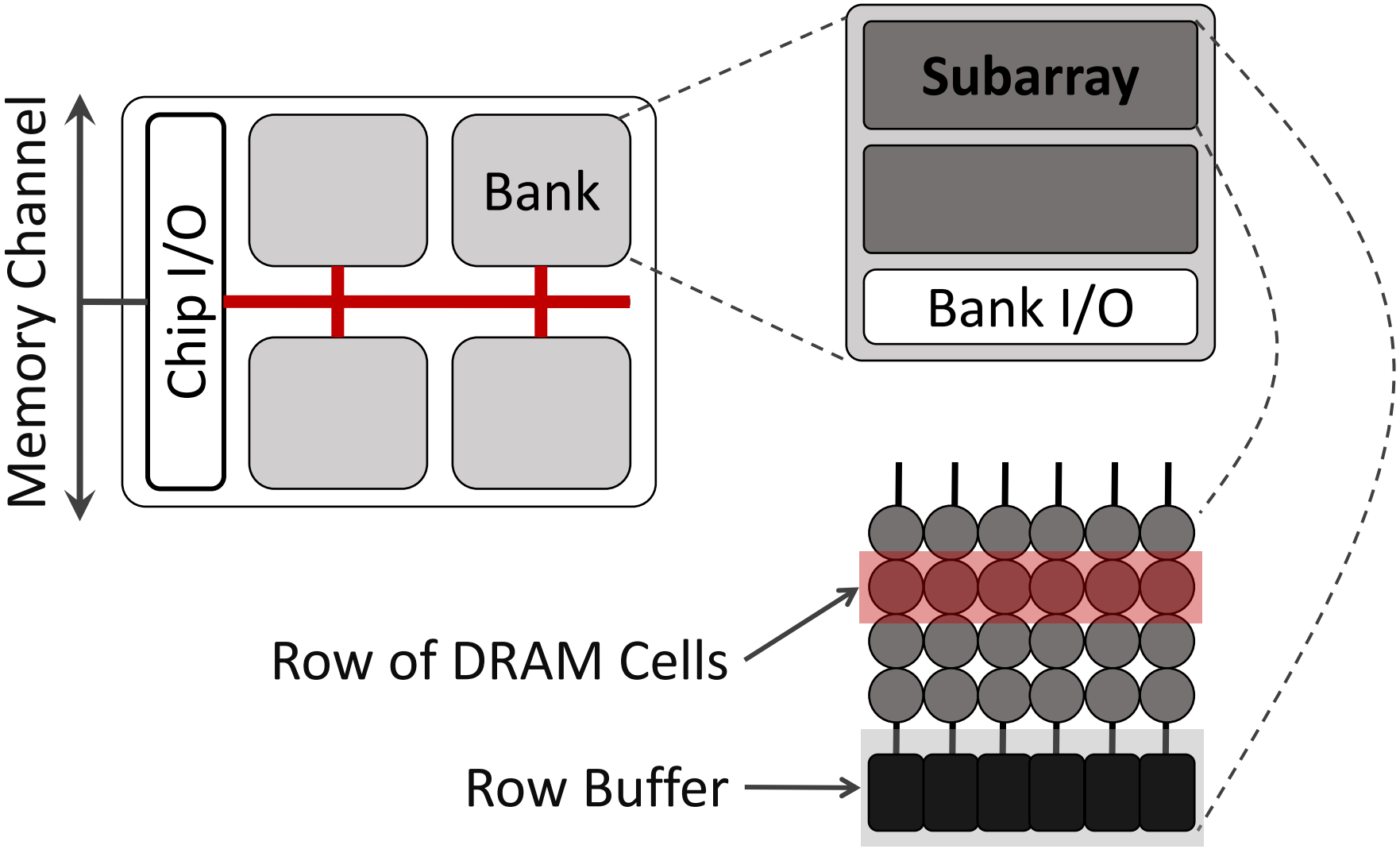
1046ns, 3.6uJ → 90ns, 0.04uJ

Brief Review:
Inside A DRAM Chip

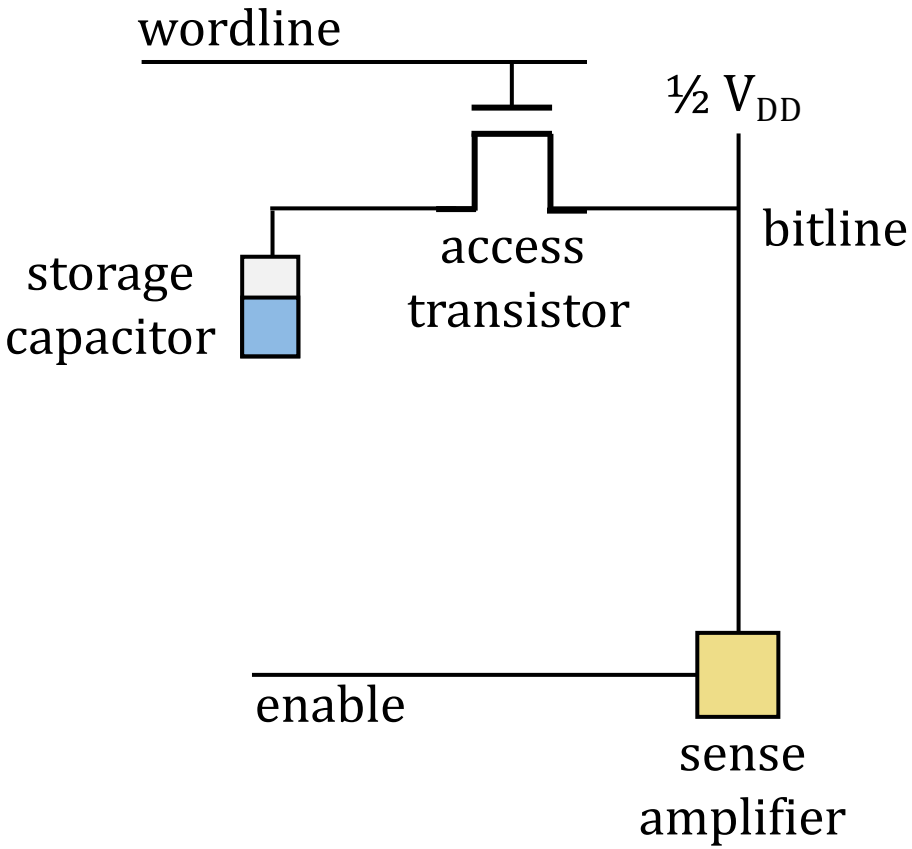
Inside a DRAM Chip



Inside a DRAM Chip: Another View



DRAM Cell Operation

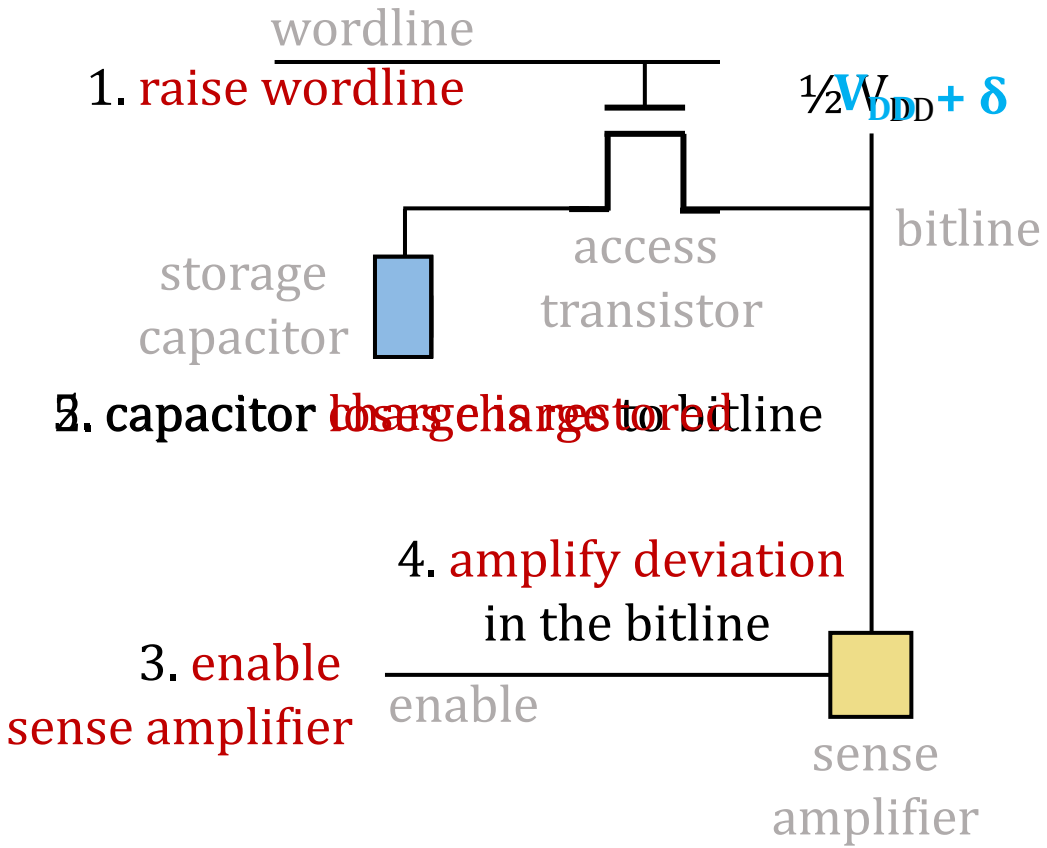


1. ACTIVATE (ACT)

2. READ/WRITE

3. PRECHARGE (PRE)

DRAM Cell Operation (1/3)

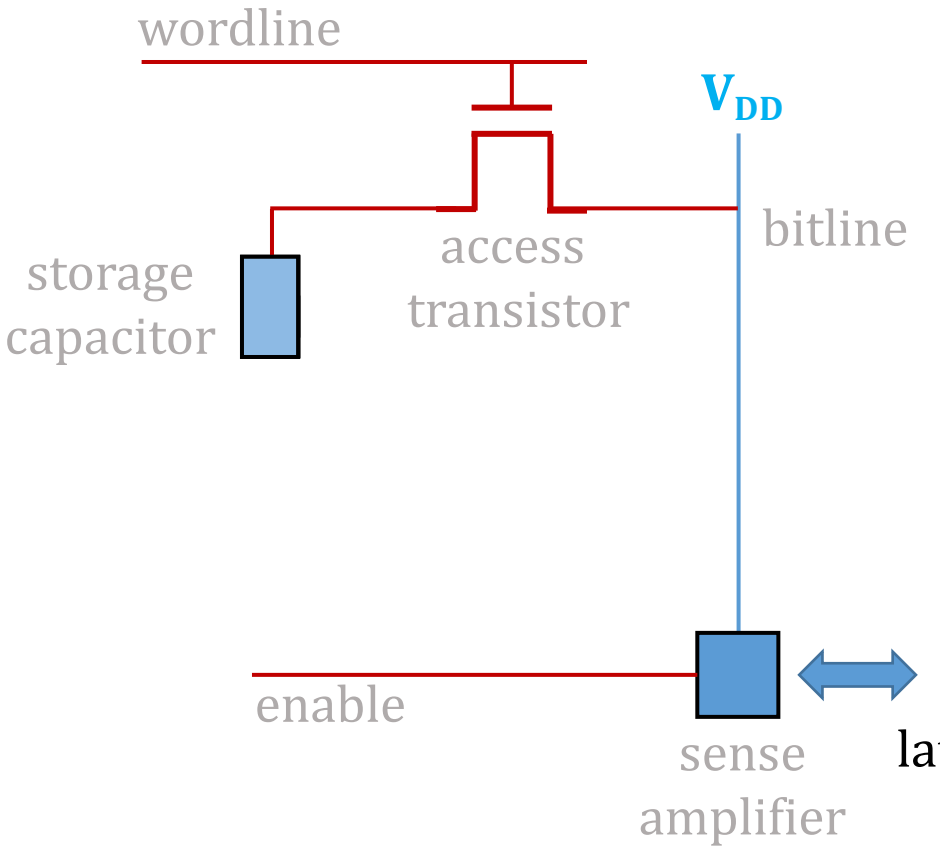


1. ACTIVATE (ACT)

2. READ/WRITE

3. PRECHARGE (PRE)

DRAM Cell Operation (2/3)



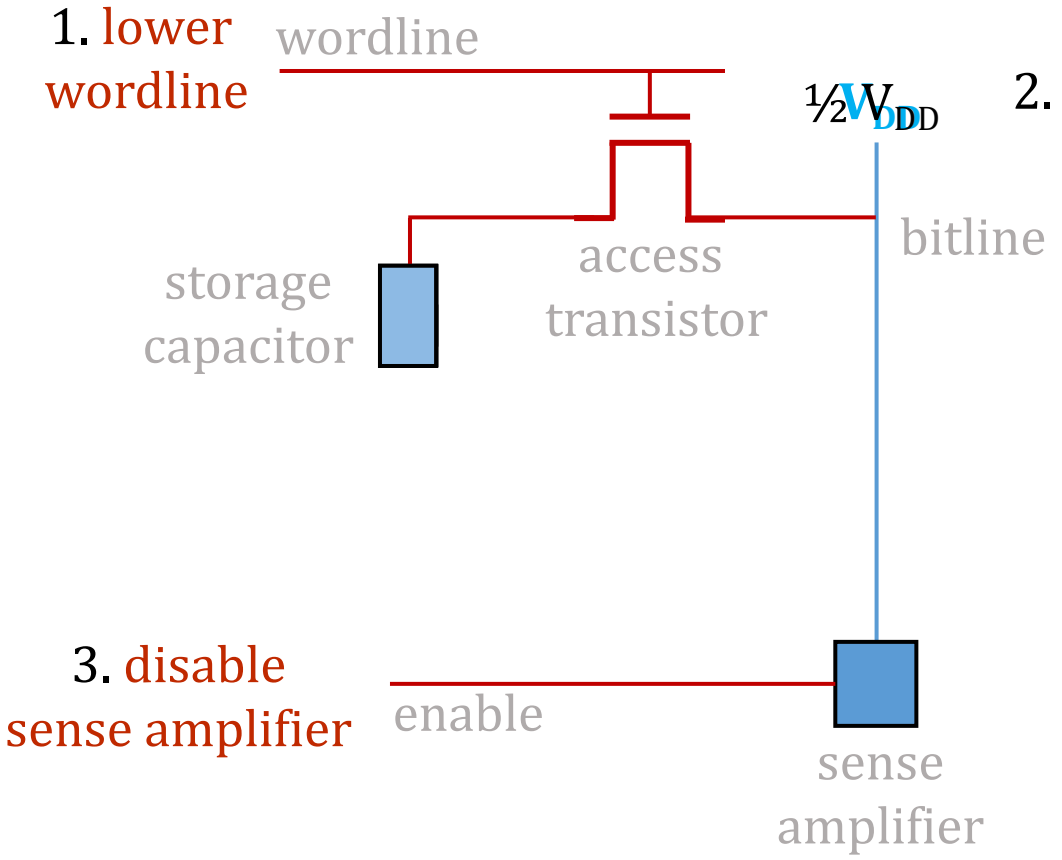
1. ACTIVATE (ACT)

2. READ/WRITE

3. PRECHARGE (PRE)

read/write charge latched in sense amplifier

DRAM Cell Operation (3/3)

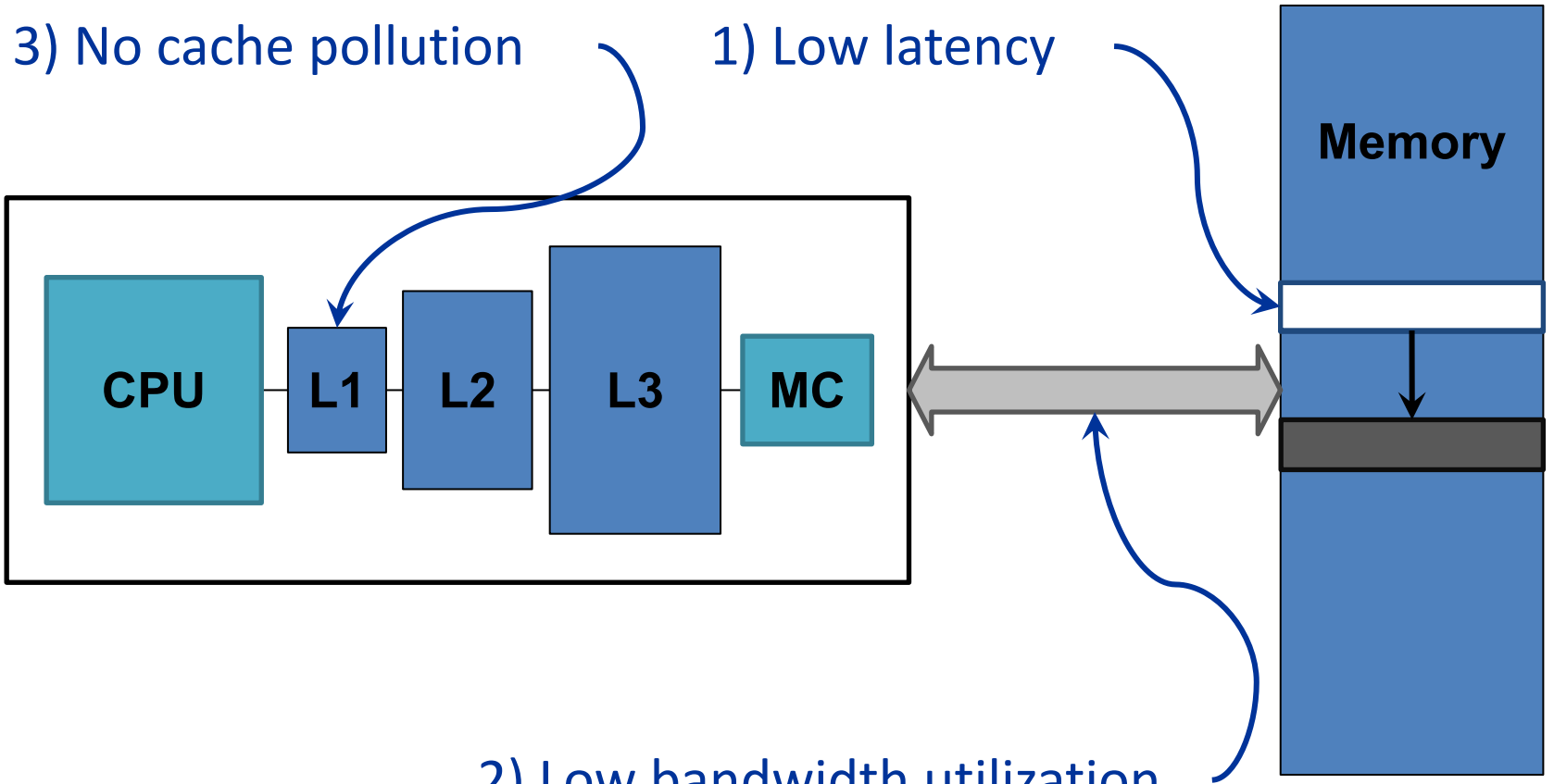


1. ACTIVATE (ACT)

2. READ/WRITE

3. PRECHARGE (PRE)

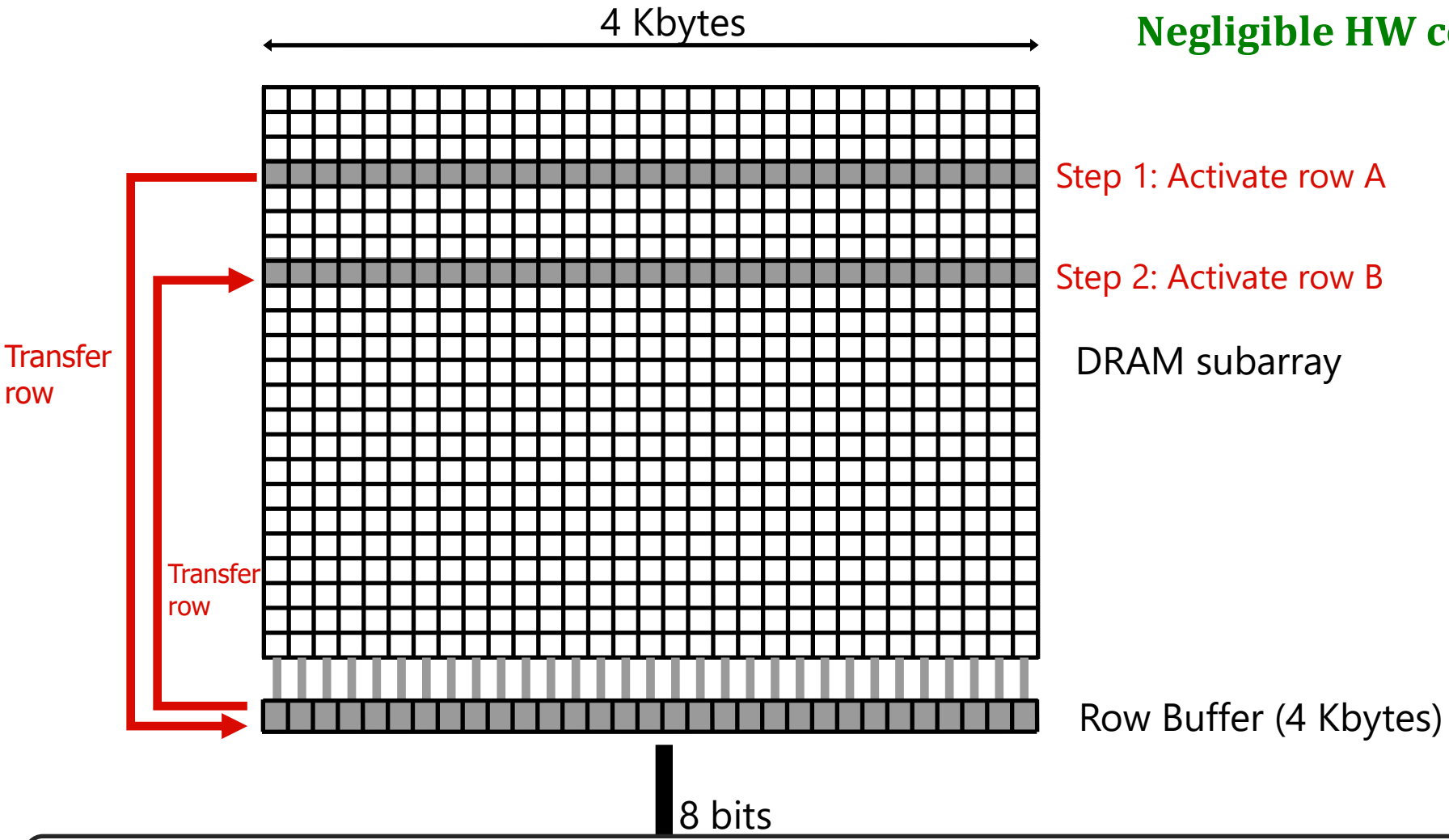
Future Systems: In-Memory Copy



1046ns, 3.6uJ → 90ns, 0.04uJ

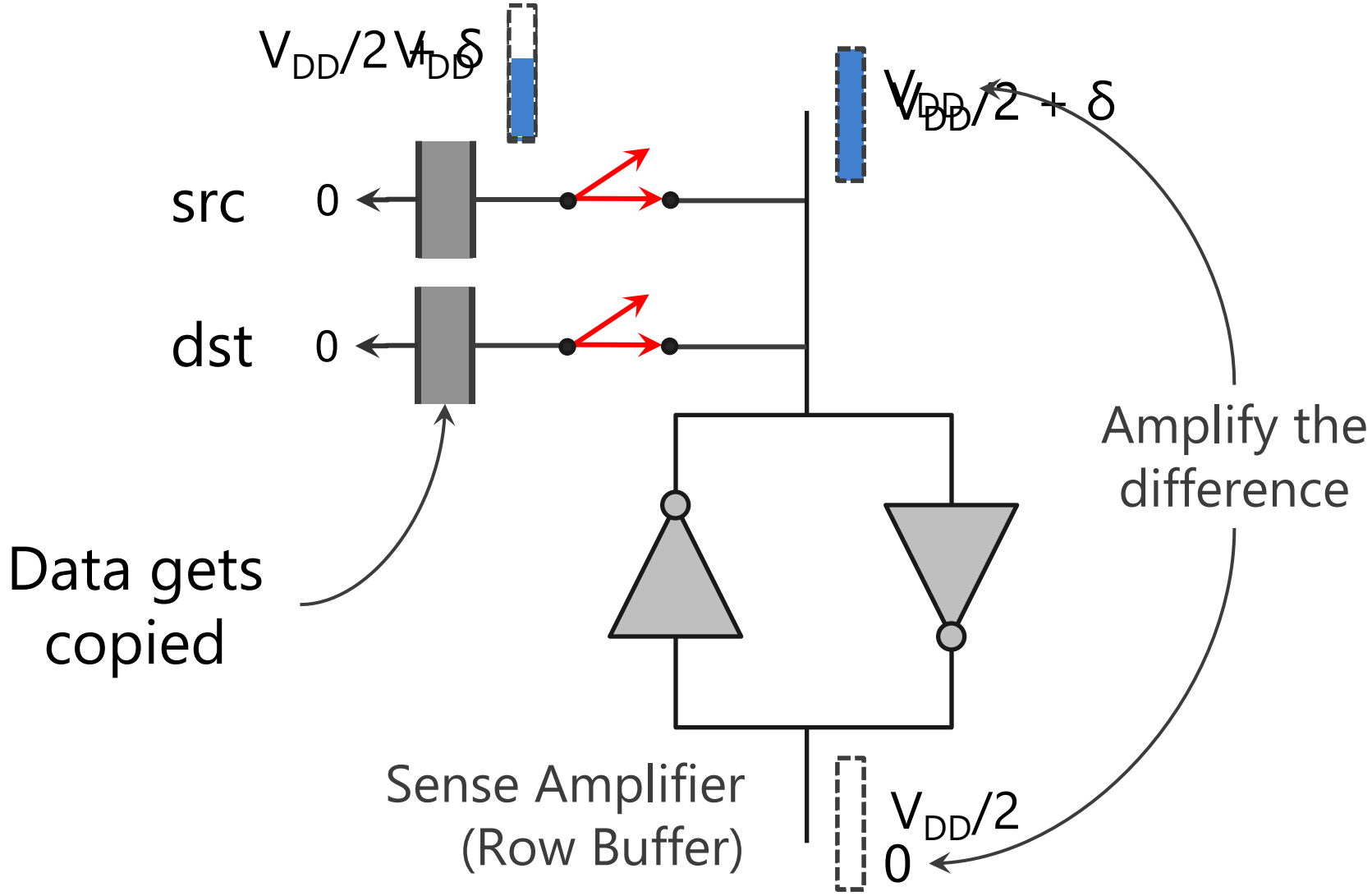
RowClone: In-DRAM Row Copy

Idea: Two consecutive ACTivates
Negligible HW cost

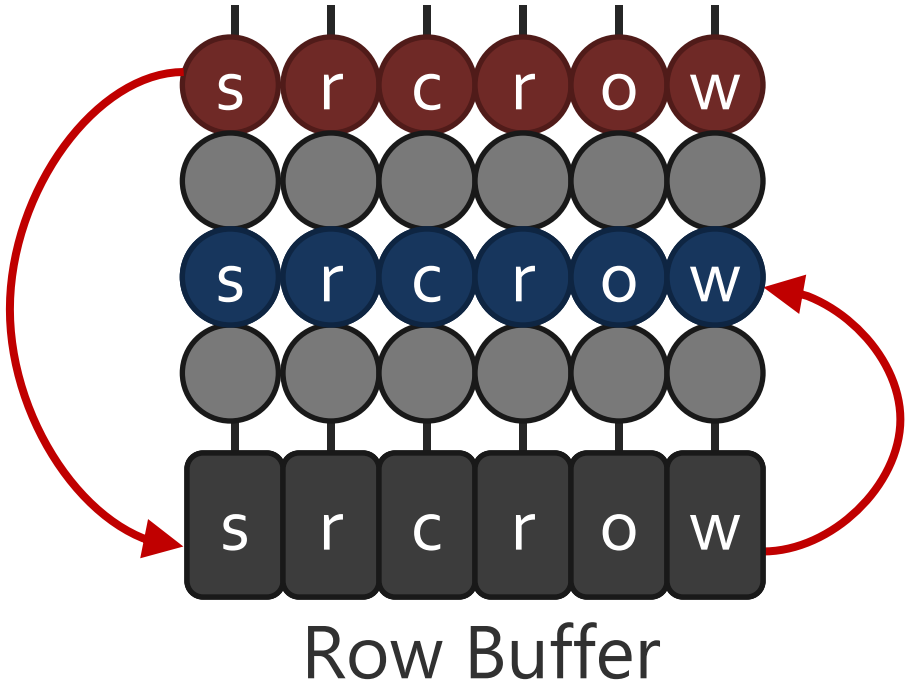


11.6X latency reduction, 74X energy reduction

RowClone: Intra-Subarray



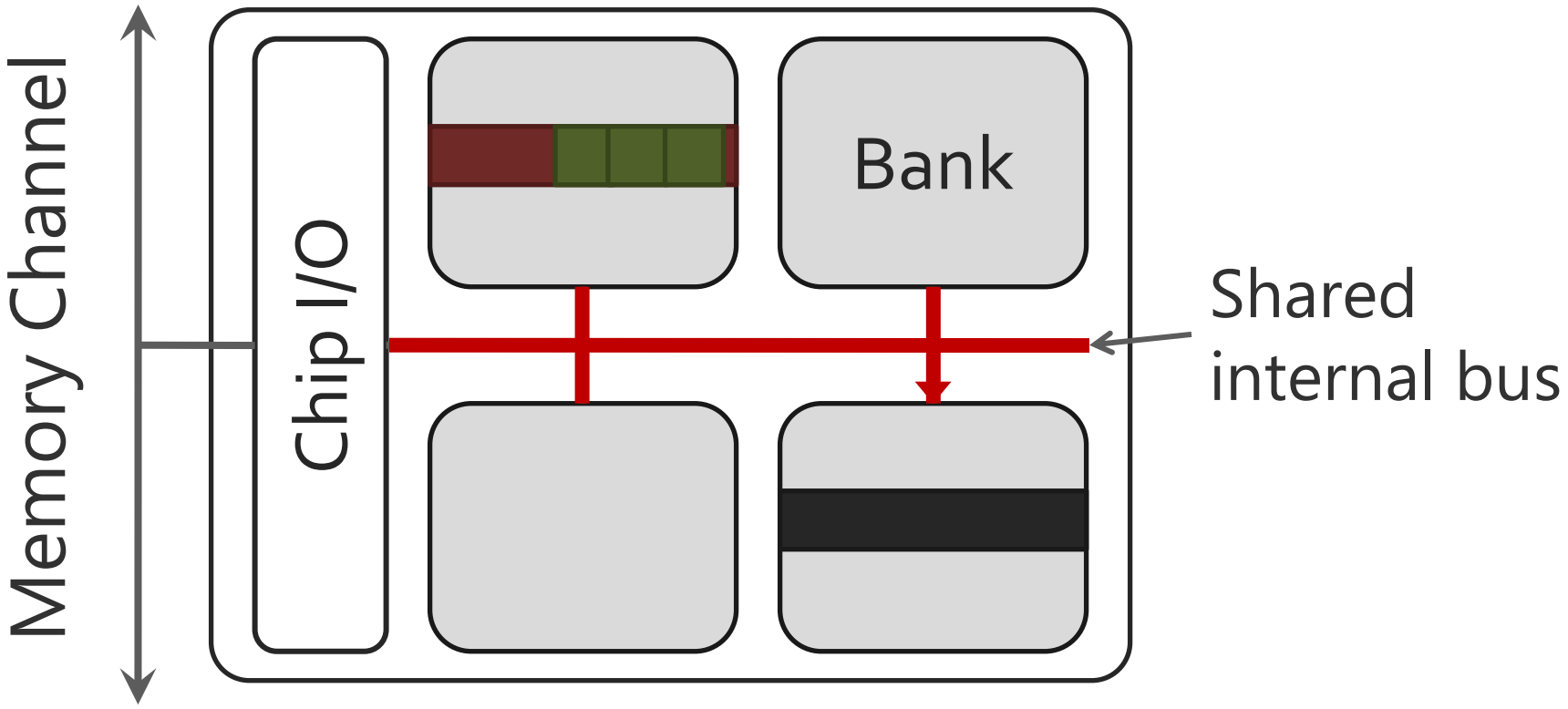
RowClone: Intra-Subarray (II)



1. **Activate** src row (copy data from src to row buffer)

2. **Activate** dst row (disconnect src from row buffer, connect dst – copy data from row buffer to dst)

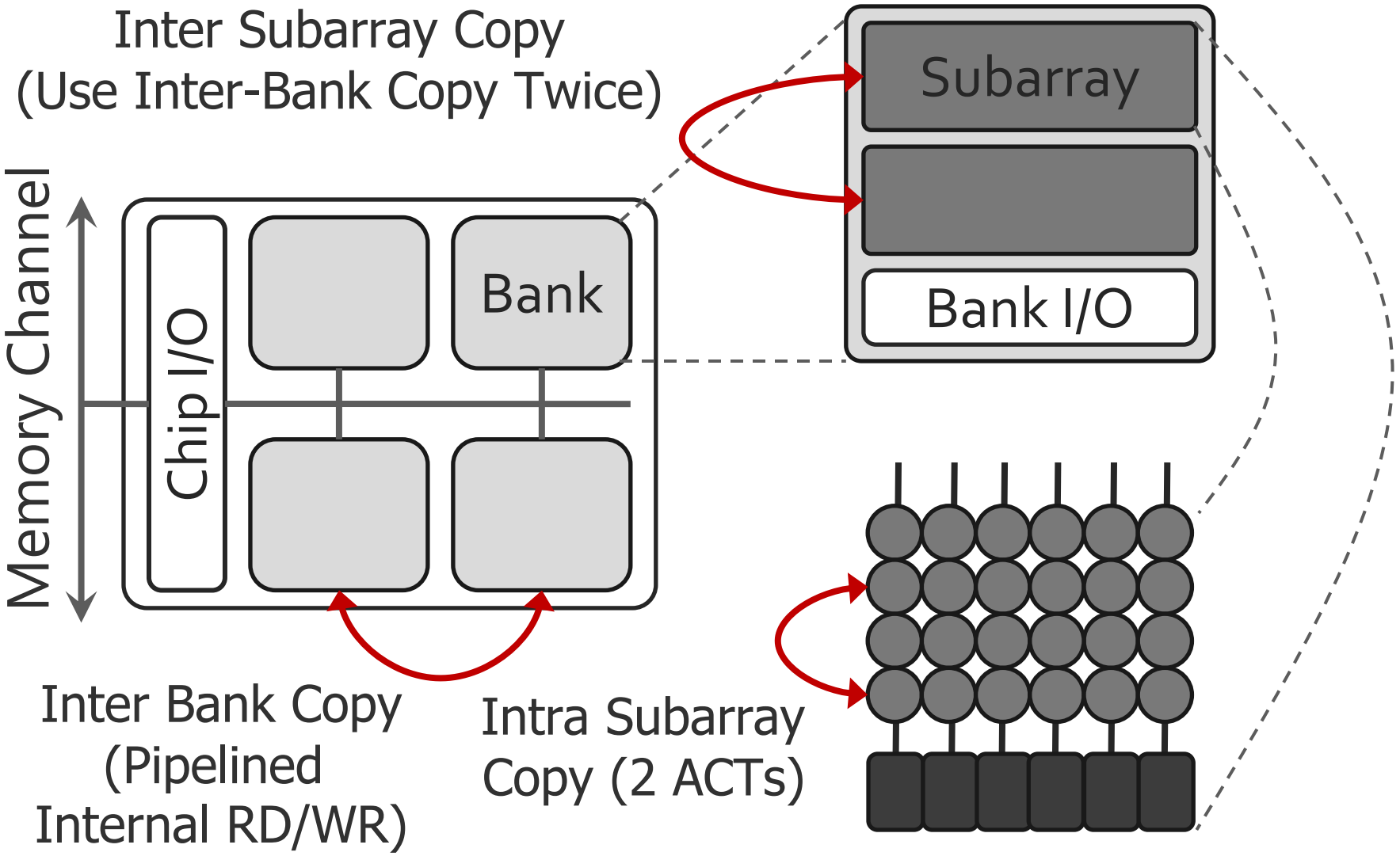
RowClone: Inter-Bank



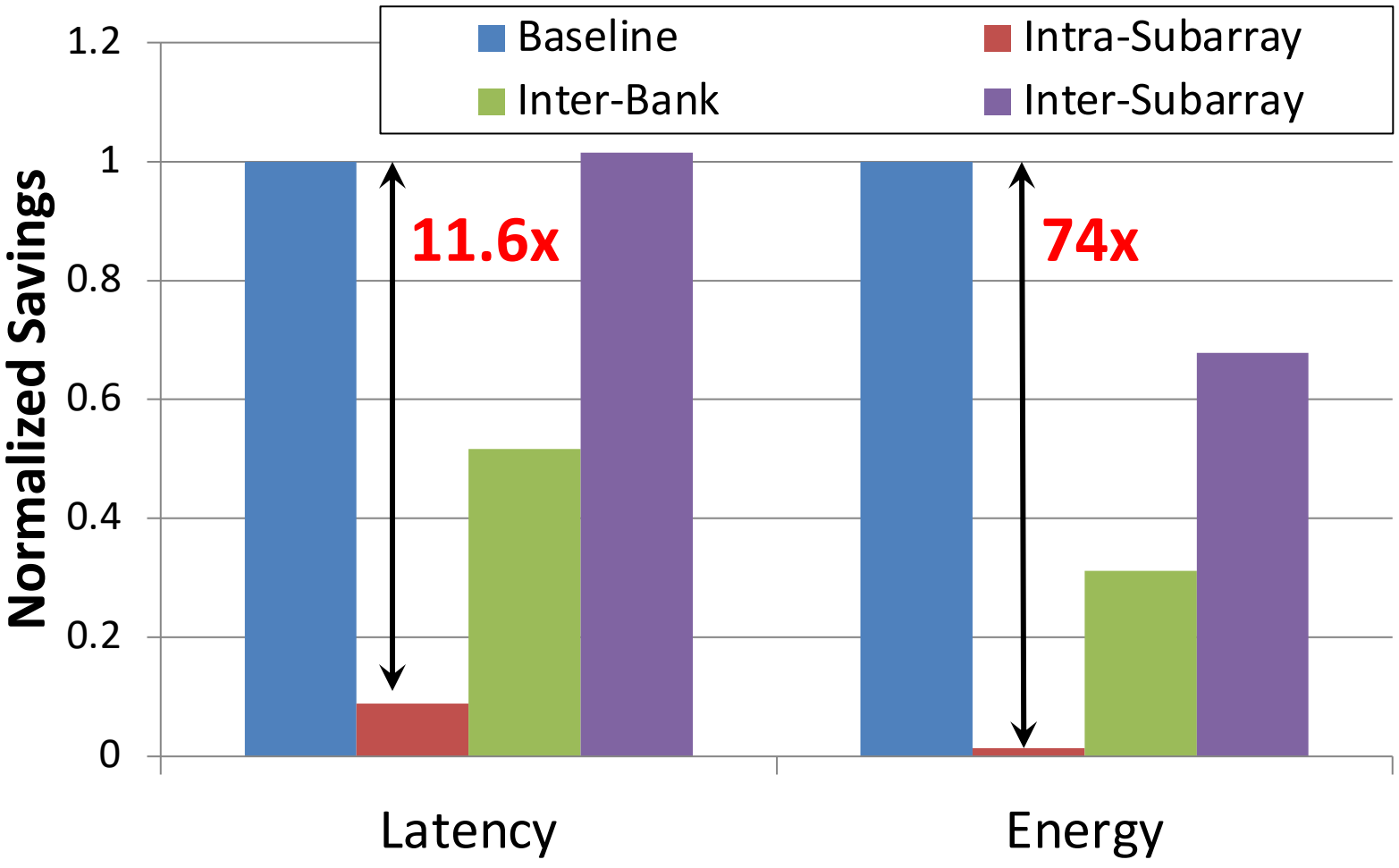
Overlap the latency of the read and the write
1.9X latency reduction, 3.2X energy reduction

Generalized RowClone

0.01% area cost

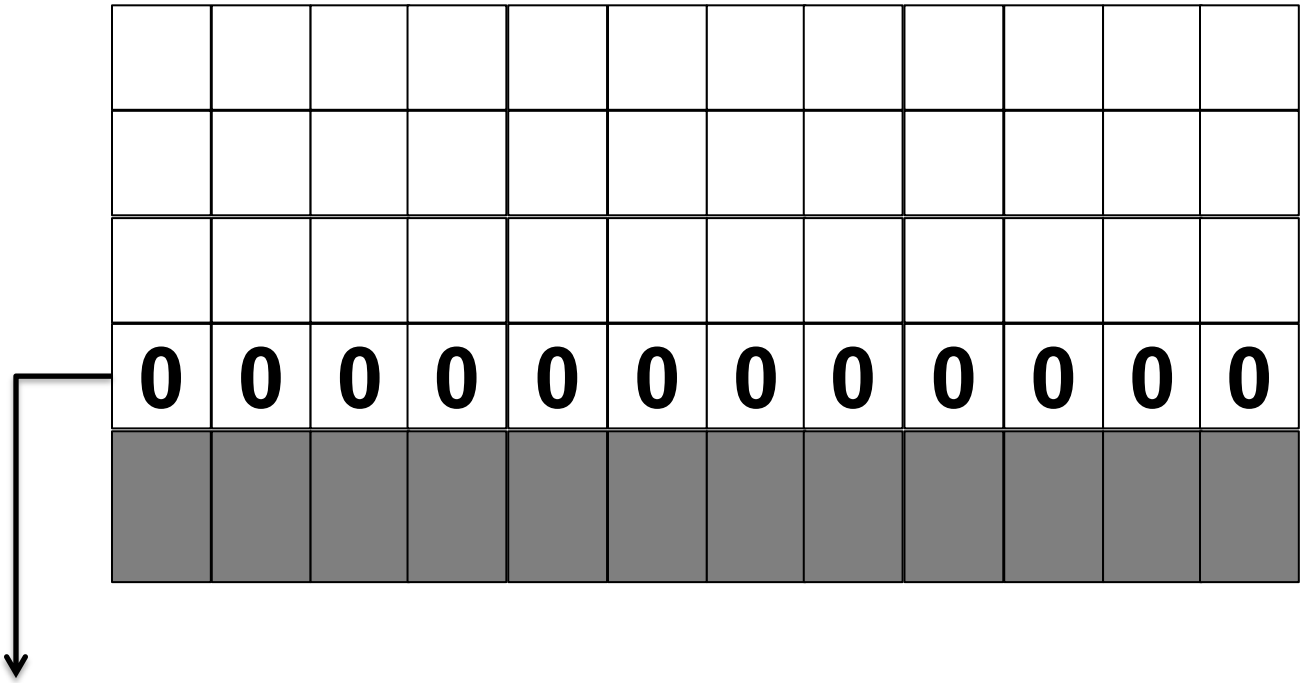


RowClone: Latency and Energy Savings



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

RowClone: Fast Row Initialization



Fix a row at Zero

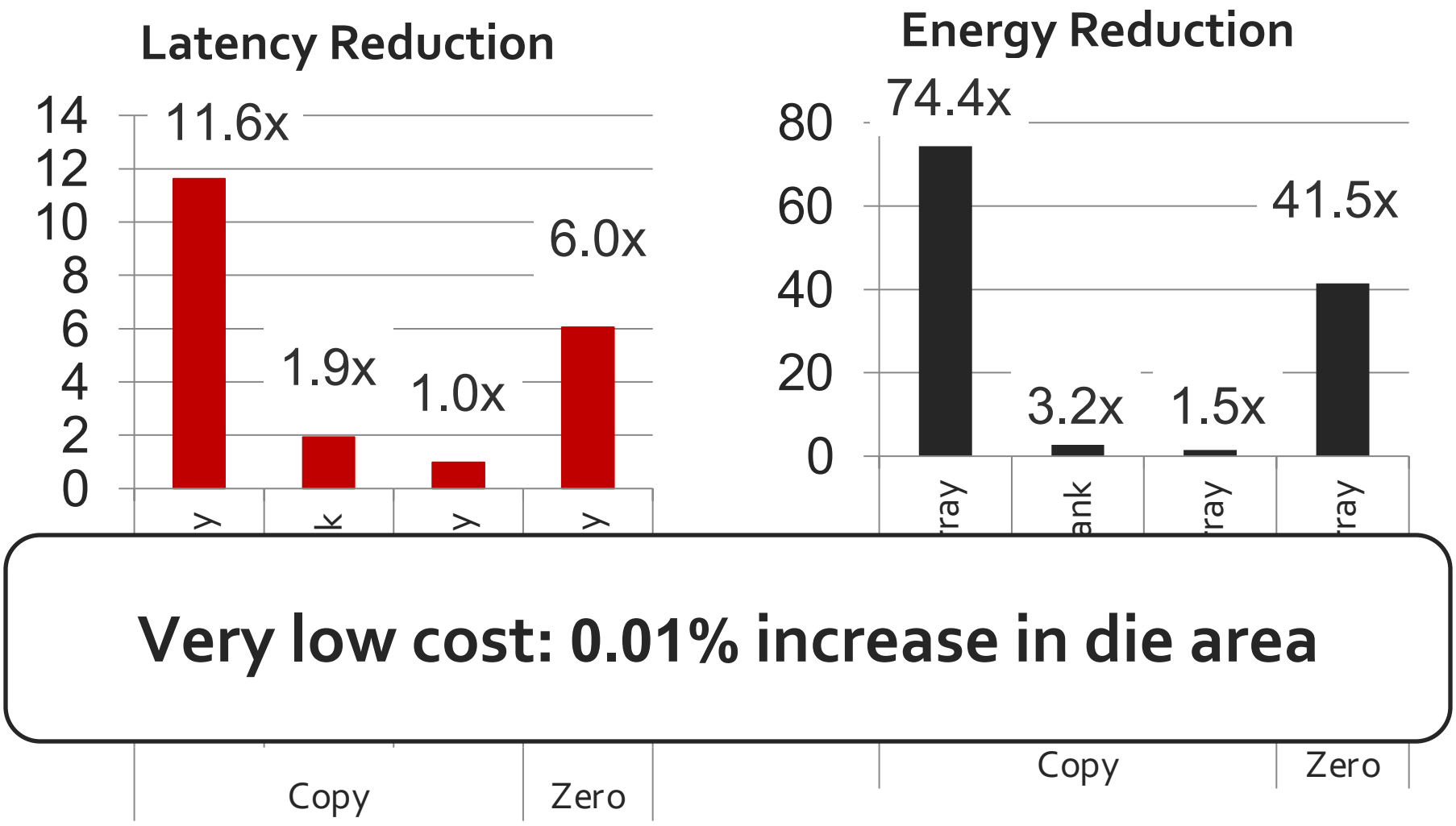
(0.5% loss in capacity)

RowClone: Bulk Initialization

- Initialization with arbitrary data
  - Initialize one row
  - Copy the data to other rows

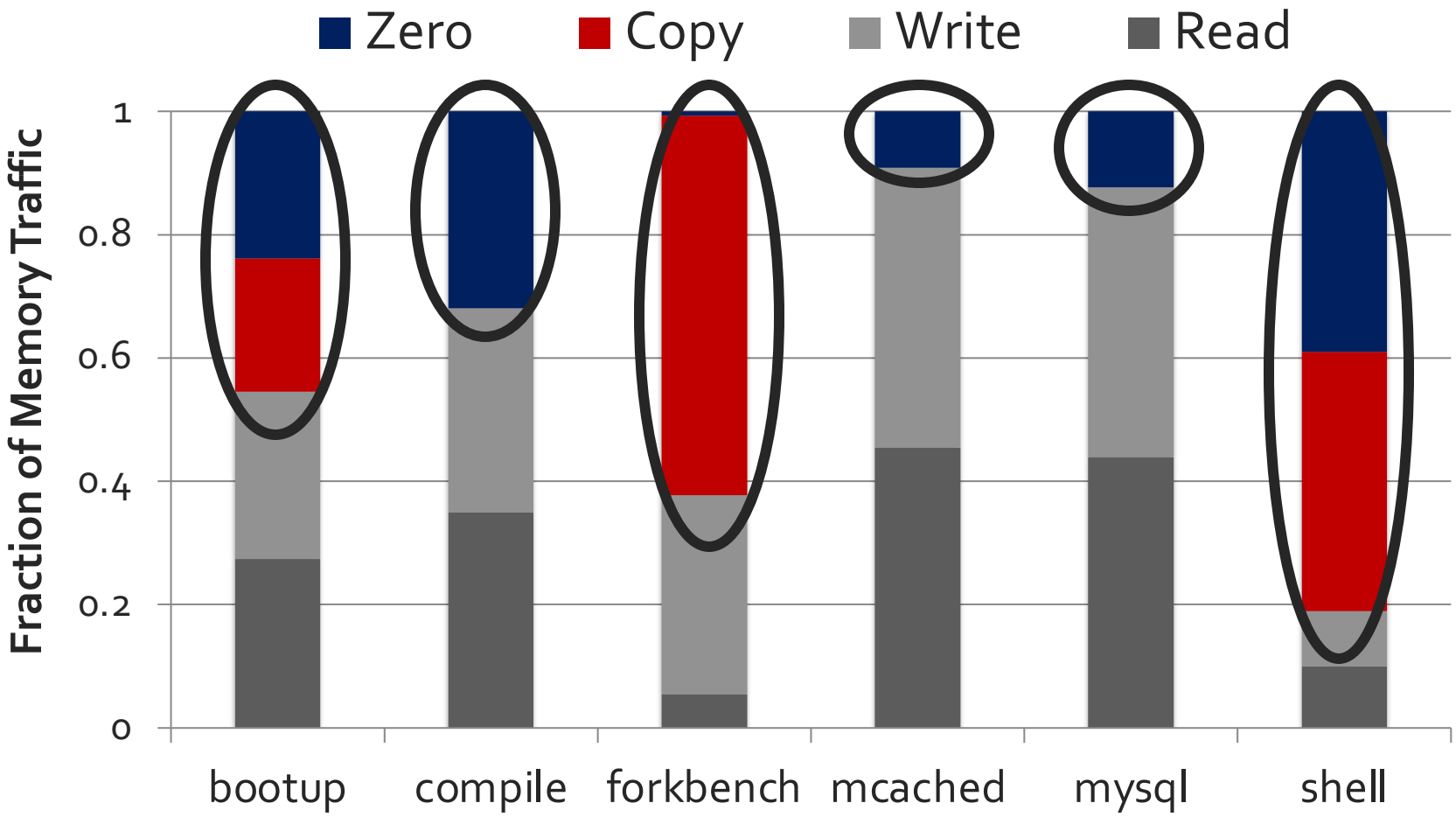
- Zero initialization (most common)
  - Reserve a row in each subarray (always zero)
  - Copy data from reserved row (FPM mode)
  - **6.0X** lower latency, **41.5X** lower DRAM energy
  - 0.2% loss in capacity

RowClone: Latency & Energy Benefits

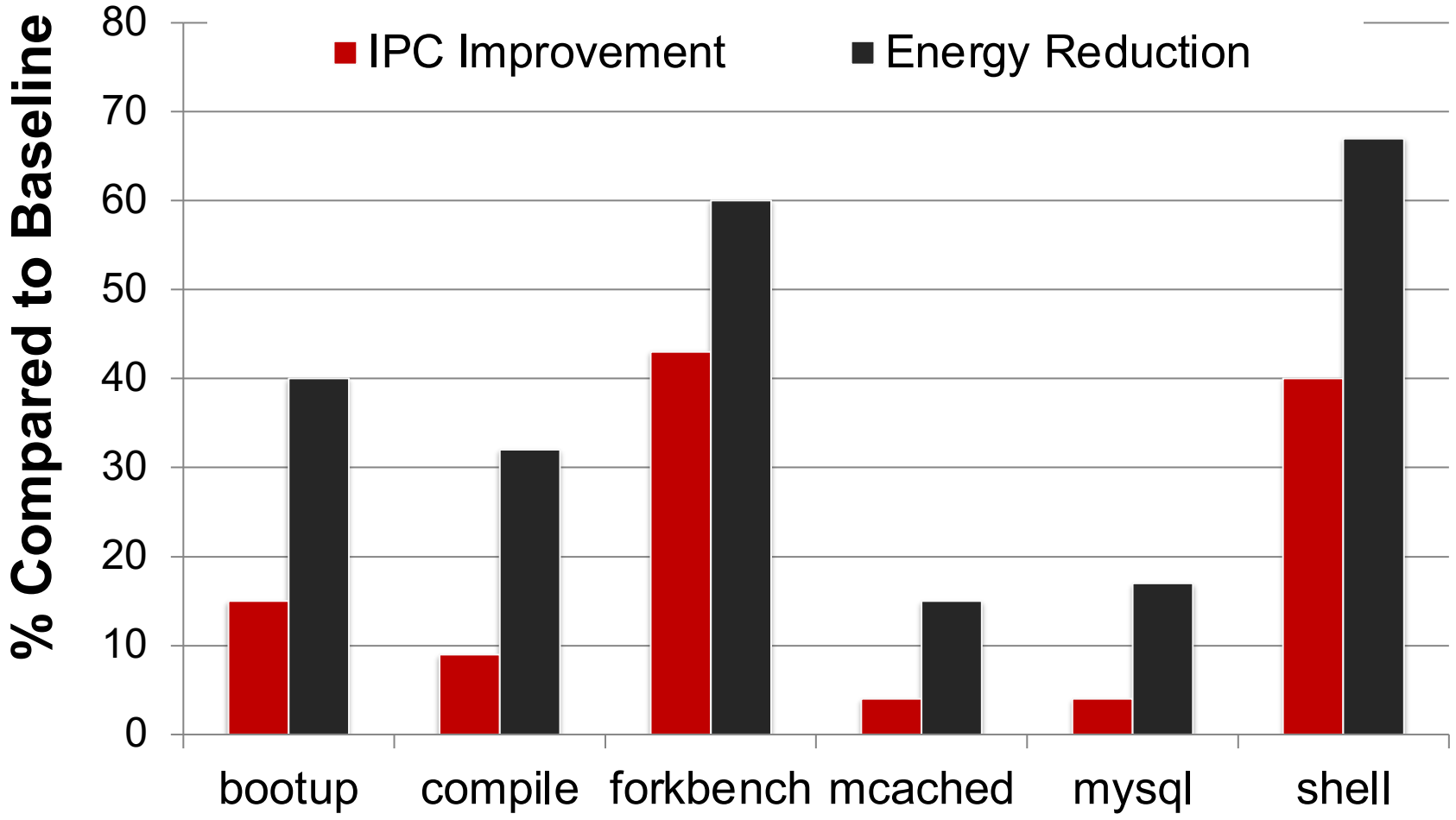


Very low cost: 0.01% increase in die area

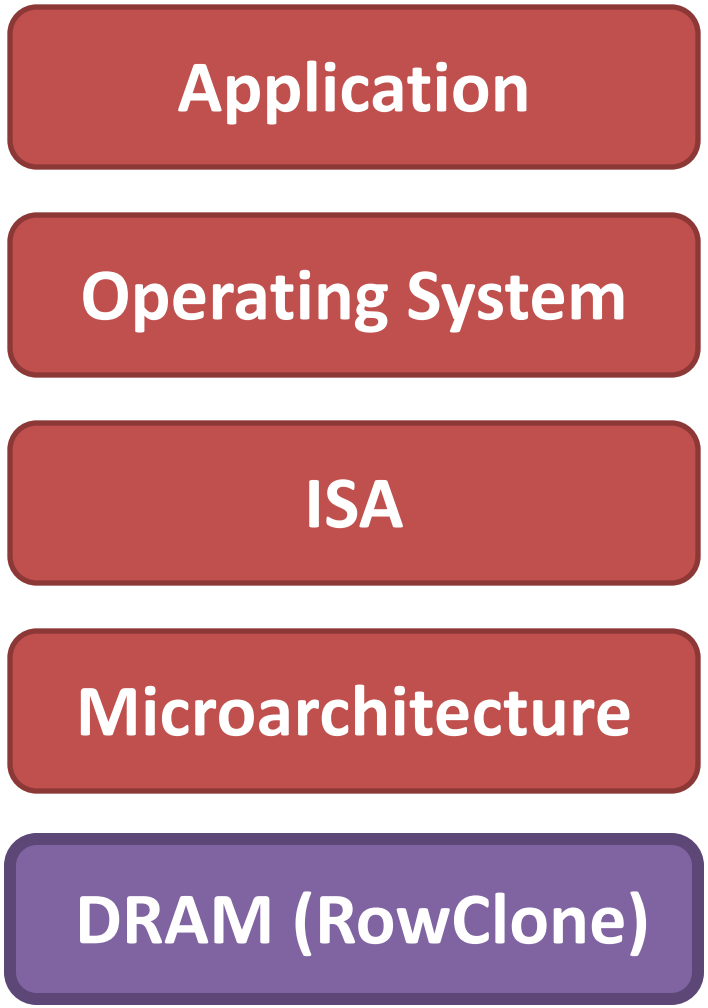
Copy and Initialization in Workloads



RowClone: Application Performance



End-to-End System Design



How to communicate occurrences of bulk copy/initialization across layers?

How to ensure cache coherence?

How to maximize latency and energy savings?

How to handle data reuse?

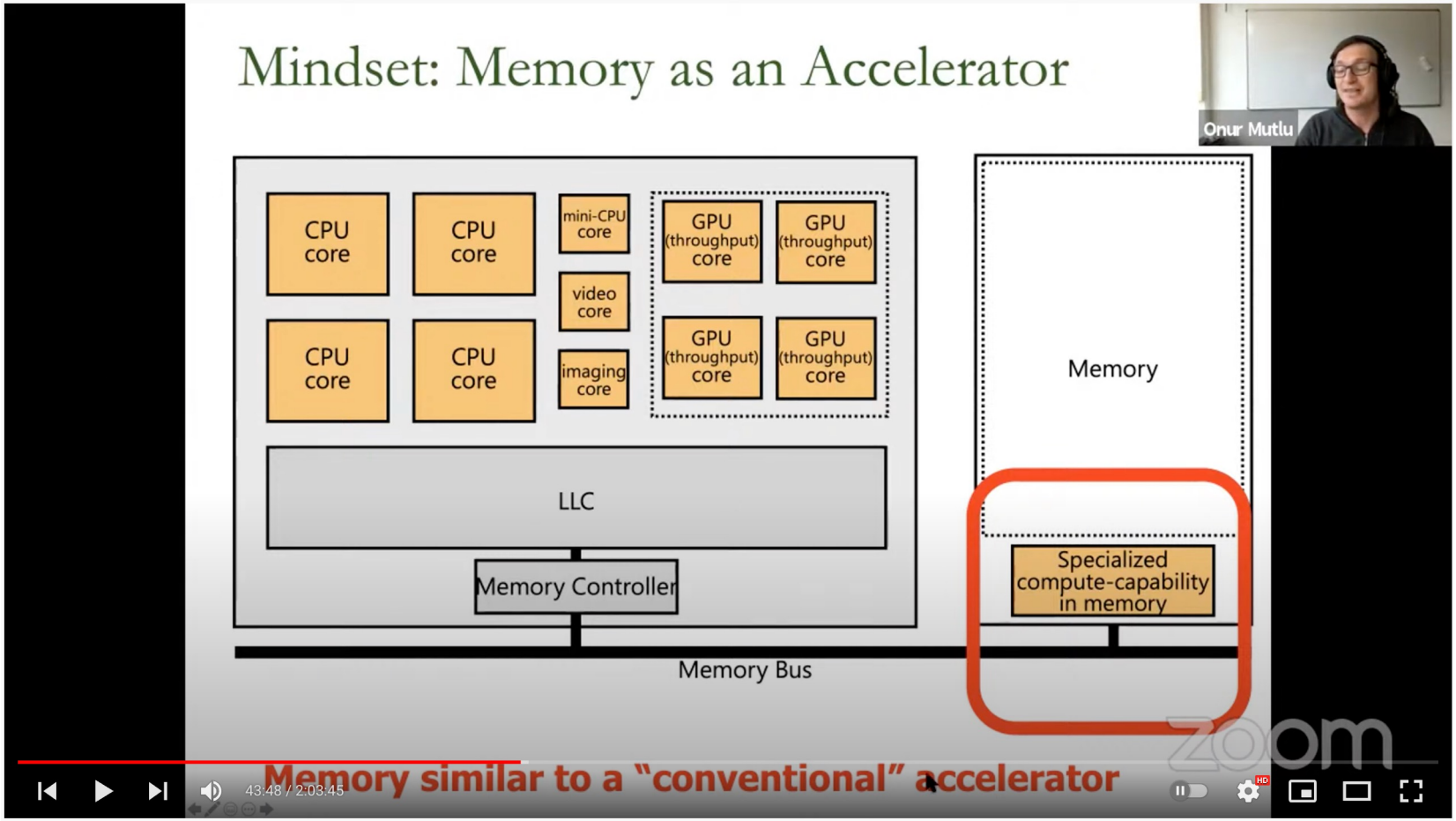
More on RowClone

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

RowClone: Fast and Energy-Efficient In-DRAM Bulk Data Copy and Initialization

Vivek Seshadri vseshadr@cs.cmu.edu
Yoongu Kim yoongukim@cmu.edu
Chris Fallin* cfallin@c1f.net
Donghyuk Lee donghyuk1@cmu.edu
Rachata Ausavarungnirun rachata@cmu.edu
Gennady Pekhimenko gpekhime@cs.cmu.edu
Yixin Luo yixinluo@andrew.cmu.edu
Onur Mutlu onur@cmu.edu
Phillip B. Gibbons† phillip.b.gibbons@intel.com
Michael A. Kozuch† michael.a.kozuch@intel.com
Todd C. Mowry tcm@cs.cmu.edu
Carnegie Mellon University †Intel Pittsburgh

Lecture on RowClone & Processing using DRAM



DEPARTMENT OF INFORMATION TECHNOLOGY AND ELECTRICAL ENGINEERING (D-ITET)

Seminar in Computer Arch. - Meeting 3: RowClone: In-Memory Data Copy and Initialization (Fall 2021)

292 views • Streamed live on Oct 7, 2021

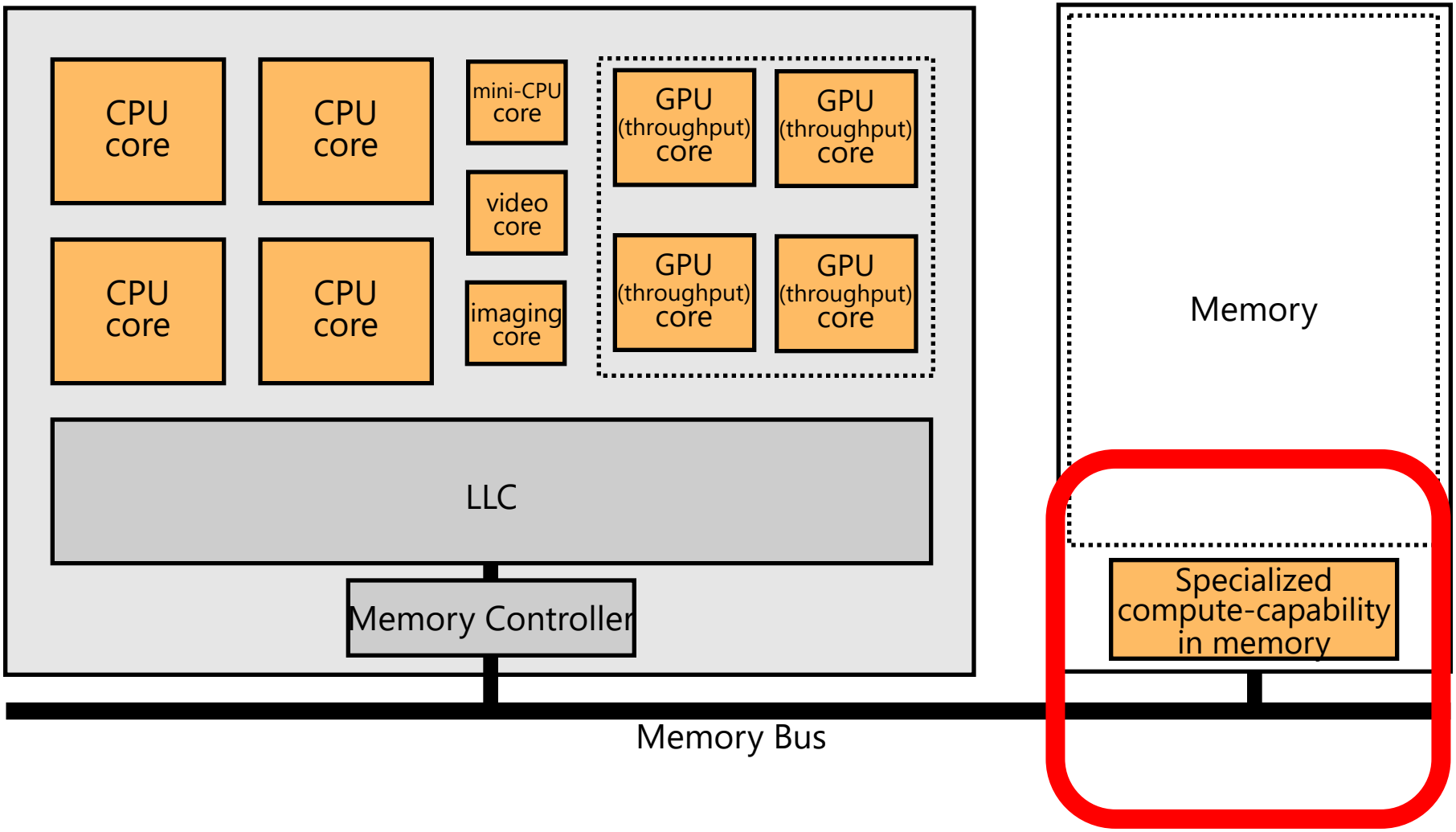
21 0 SHARE SAVE ...

Onur Mutlu Lectures
19.1K subscribers

SUBSCRIBED

https://www.youtube.com/watch?v=n6Pwg1qax_E&list=PL5Q2soXY2Zi_7UBNmC9B8Yr5JSwTG9yH4&index=4

Mindset: Memory as an Accelerator



Memory similar to a "conventional" accelerator

RowClone Strengths

Strengths of the Paper

- Simple, novel mechanism to solve an important problem
- Effective and low hardware overhead
- Intuitive idea!
- Greatly improves performance and efficiency (assuming data is mapped nicely)
- Seems like a clear win for data initialization (without mapping requirements)
- Makes software designer's life easier
  - If copies are 10x-100x cheaper, how to design software?

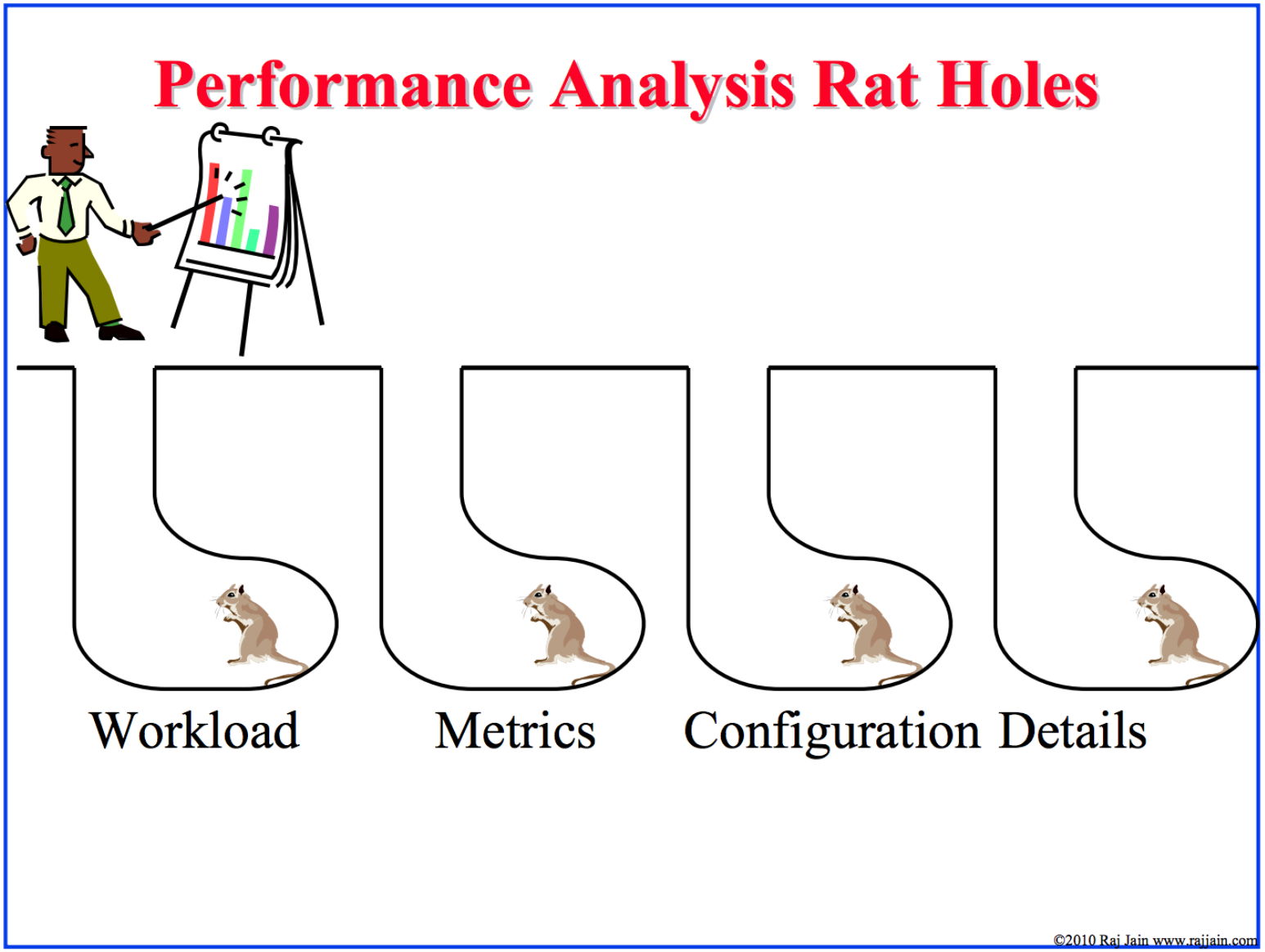
- Paper tackles many low-level and system-level issues
- Well-written, insightful paper

RowClone Weaknesses

Weaknesses

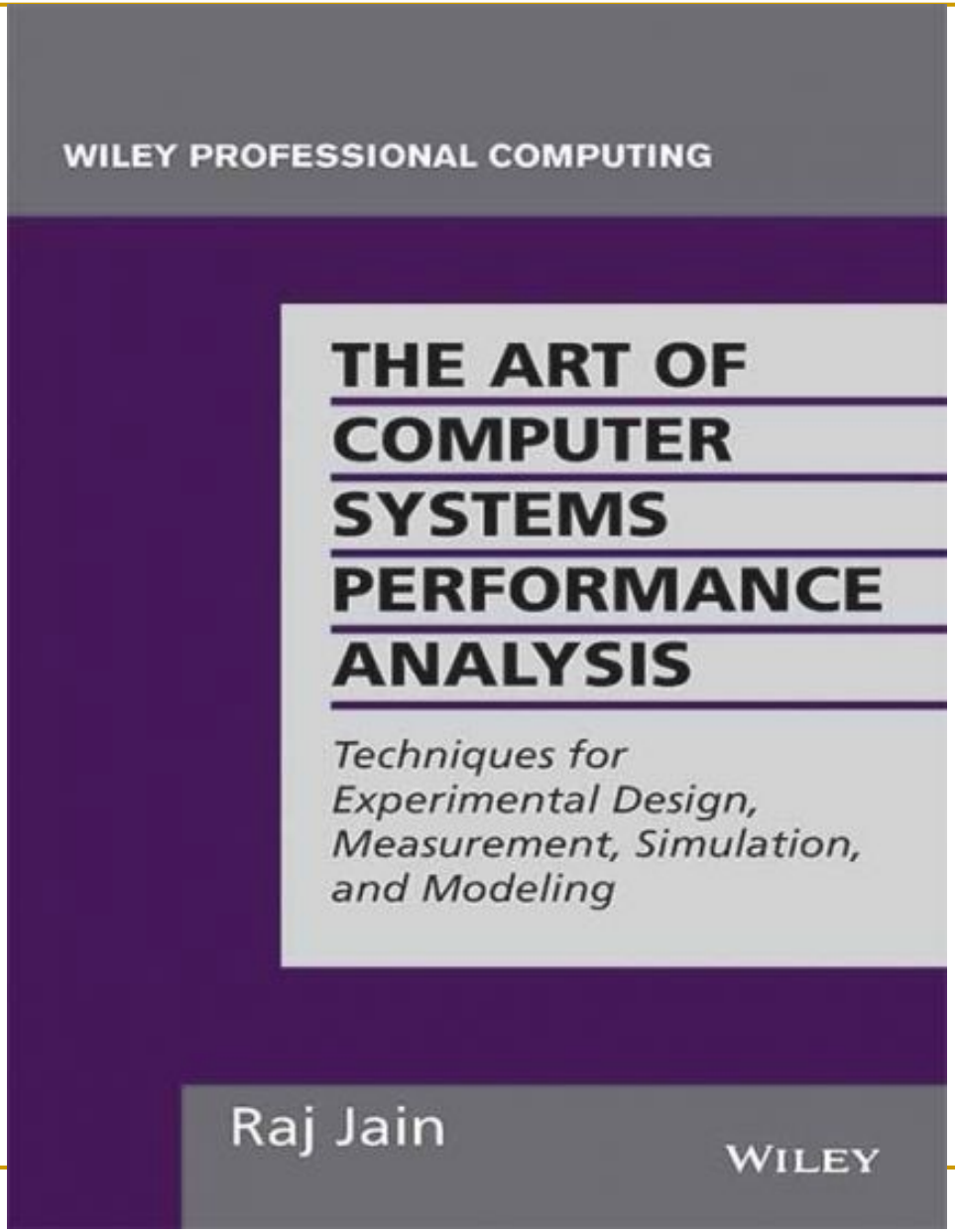
- Requires data to be mapped in the same subarray to deliver the largest benefits
  - Helps less if data movement is not within a subarray
  - Does not help if data movement is across DRAM channels
- Inter-subarray copy is very inefficient
- Causes many changes in the system stack
  - End-to-end design spans applications to circuits
  - Software-hardware cooperative solution might not always be easy to adopt
- Cache coherence and data reuse cause real overheads
- Evaluation is done solely in simulation
- Evaluation does not consider multi-chip systems
- Are these the best workloads to evaluate?

Recall: Try to Avoid Rat Holes



Source: https://www.cse.wustl.edu/~jain/iucee/ftp/k_10adp.pdf

Aside: A Recommended Book



Raj Jain, "The Art of Computer Systems Performance Analysis," Wiley, 1991.

10.8 DECISION MAKER'S GAMES

Even if the performance analysis is correctly done and presented, it may not be enough to persuade your audience—the decision makers—to follow your recommendations. The list shown in Box 10.2 is a compilation of reasons for rejection heard at various performance analysis presentations. You can use the list by presenting it immediately and pointing out that the reason for rejection is not new and that the analysis deserves more consideration. Also, the list is helpful in getting the competing proposals rejected!

There is no clear end of an analysis. Any analysis can be rejected simply on the grounds that the problem needs more analysis. This is the first reason listed in Box 10.2. The second most common reason for rejection of an analysis and for endless debate is the workload. Since workloads are always based on the past measurements, their applicability to the current or future environment can always be questioned. Actually workload is one of the four areas of discussion that lead a performance presentation into an endless debate. These "rat holes" and their relative sizes in terms of time consumed are shown in Figure 10.26. Presenting this cartoon at the beginning of a presentation helps to avoid these areas.

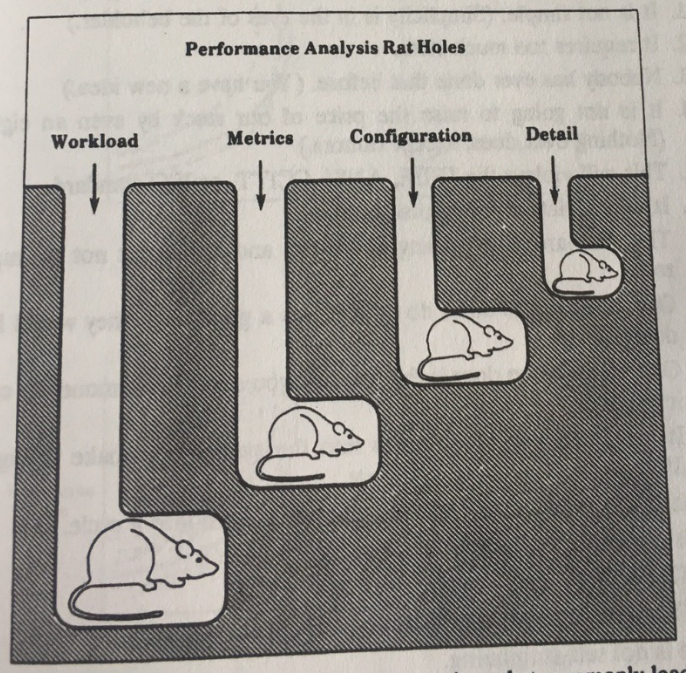


FIGURE 10.26 Four issues in performance presentations that commonly lead to endless discussion.

Raj Jain, "The Art of Computer Systems Performance Analysis," Wiley, 1991.

Improvements on RowClone

RowClone Extensions and Follow-Up Work

- Can we do faster inter-subarray copy?
  - Yes, see LISA [Chang et al., HPCA 2016]
- Can we enable data movement at smaller granularities within a bank?
  - Yes, see FIGARO [Wang et al., MICRO 2020]
- Can we do better inter-bank copy?
  - Yes, see Network-on-Memory [CAL 2020]
- Can similar ideas and DRAM properties be used to perform computation on data?
  - Yes, see Ambit [Seshadri et al., CAL 2015, MICRO 2017]
  - Yes, see SIMDRAM [Hajinazar & Oliveira, ASPLOS 2021]

LISA: Increasing Connectivity in DRAM

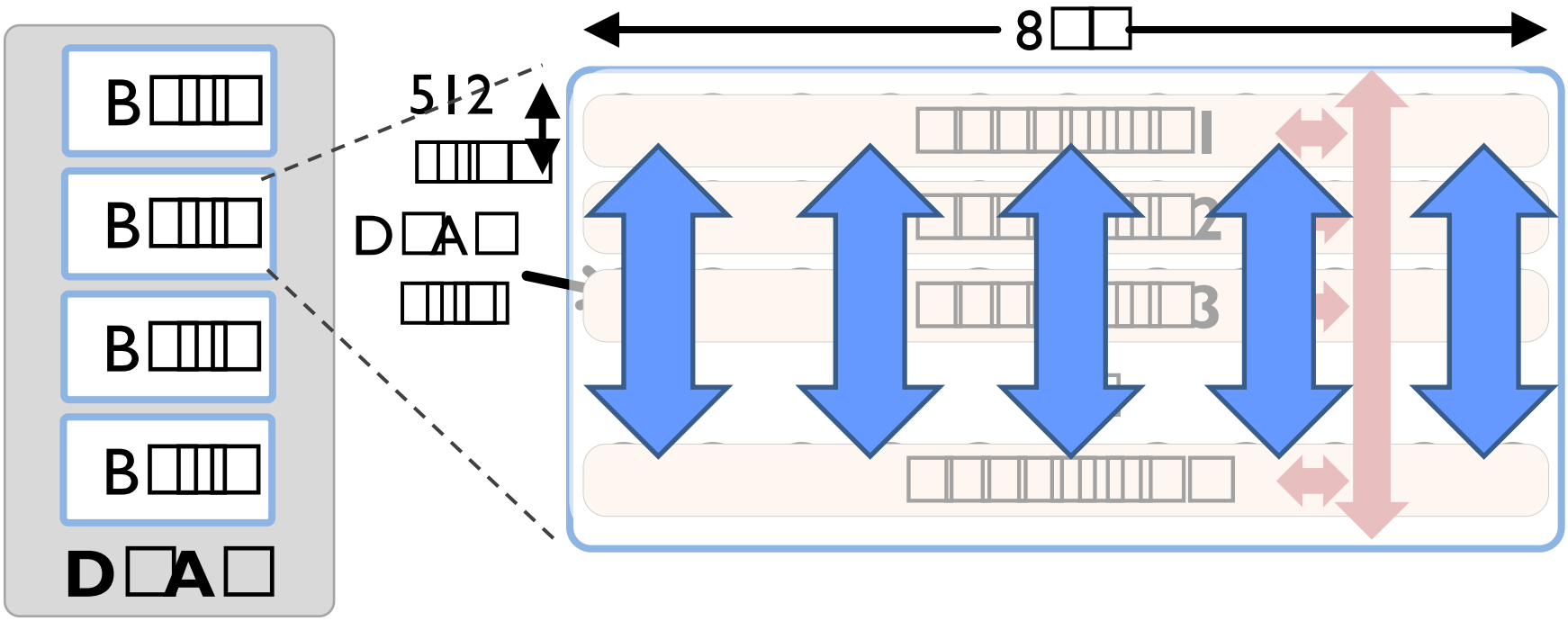
- Kevin K. Chang, Prashant J. Nair, Saugata Ghose, Donghyuk Lee, Moinuddin K. Qureshi, and Onur Mutlu,
  "Low-Cost Inter-Linked Subarrays (LISA): Enabling Fast Inter-Subarray Data Movement in DRAM"
  *Proceedings of the 22nd International Symposium on High-Performance Computer Architecture (**HPCA**)*, Barcelona, Spain, March 2016.
  [Slides (pptx) (pdf)]
  [Source Code]

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

Kevin K. Chang†, Prashant J. Nair⋆, Donghyuk Lee†, Saugata Ghose†, Moinuddin K. Qureshi⋆, and Onur Mutlu†

†*Carnegie Mellon University* ⋆*Georgia Institute of Technology*

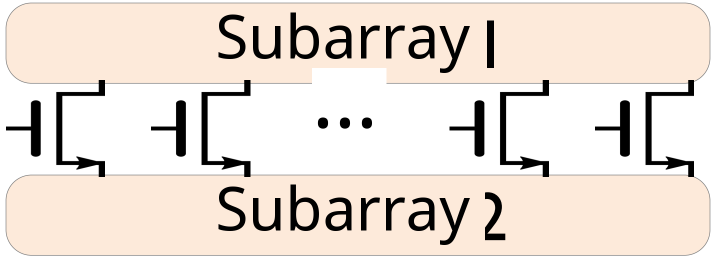
□□□□□D □□□□□□□D □A □?



Goal: Provide a new substrate to enable wide connectivity between subarrays

Key Idea and Applications

- Low-cost Inter-linked subarrays (LISA)
  - Fast bulk data movement between subarrays
  - Wide datapath via isolation transistors: 0.8% DRAM chip area



- LISA is a versatile substrate → new applications
  - Fast bulk data copy: Copy latency 1.363ms→0.148ms (9.2x)
    → 66% speedup, -55% DRAM energy
  - In-DRAM caching: Hot data access latency 48.7ns→21.5ns (2.2x)
    → 5% speedup
  - Fast precharge: Precharge latency 13.1ns→5.0ns (2.6x)
    → 8% speedup

More on LISA

- Kevin K. Chang, Prashant J. Nair, Saugata Ghose, Donghyuk Lee, Moinuddin K. Qureshi, and Onur Mutlu,
  "Low-Cost Inter-Linked Subarrays (LISA): Enabling Fast Inter-Subarray Data Movement in DRAM"
  *Proceedings of the 22nd International Symposium on High-Performance Computer Architecture (**HPCA**)*, Barcelona, Spain, March 2016.
  [Slides (pptx) (pdf)]
  [Source Code]

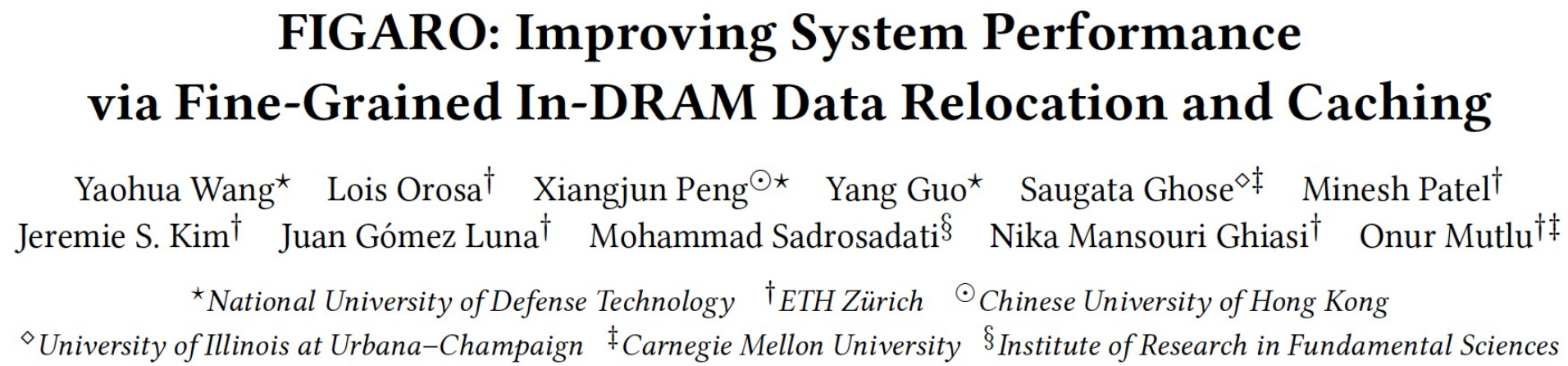
Low-Cost Inter-Linked Subarrays (LISA): Enabling Fast Inter-Subarray Data Movement in DRAM

Kevin K. Chang†, Prashant J. Nair⋆, Donghyuk Lee†, Saugata Ghose†, Moinuddin K. Qureshi⋆, and Onur Mutlu†

†*Carnegie Mellon University* ⋆*Georgia Institute of Technology*

FIGARO: Fine-Grained In-DRAM Copy

- Yaohua Wang, Lois Orosa, Xiangjun Peng, Yang Guo, Saugata Ghose, Minesh Patel, Jeremie S. Kim, Juan Gómez Luna, Mohammad Sadrosadati, Nika Mansouri Ghiasi, and Onur Mutlu,
  "FIGARO: Improving System Performance via Fine-Grained In-DRAM Data Relocation and Caching"
  *Proceedings of the 53rd International Symposium on Microarchitecture (**MICRO**)*, Virtual, October 2020.



FIGARO: Improving System Performance via Fine-Grained In-DRAM Data Relocation and Caching

Yaohua Wang[⋆] Lois Orosa[†] Xiangjun Peng[⊙⋆] Yang Guo[⋆] Saugata Ghose[◇‡] Minesh Patel[†]
Jeremie S. Kim[†] Juan Gómez Luna[†] Mohammad Sadrosadati[§] Nika Mansouri Ghiasi[†] Onur Mutlu[†‡]

[⋆]*National University of Defense Technology* [†]*ETH Zürich* [⊙]*Chinese University of Hong Kong*
[◇]*University of Illinois at Urbana–Champaign* [‡]*Carnegie Mellon University* [§]*Institute of Research in Fundamental Sciences*

Network-On-Memory: Fast Inter-Bank Copy

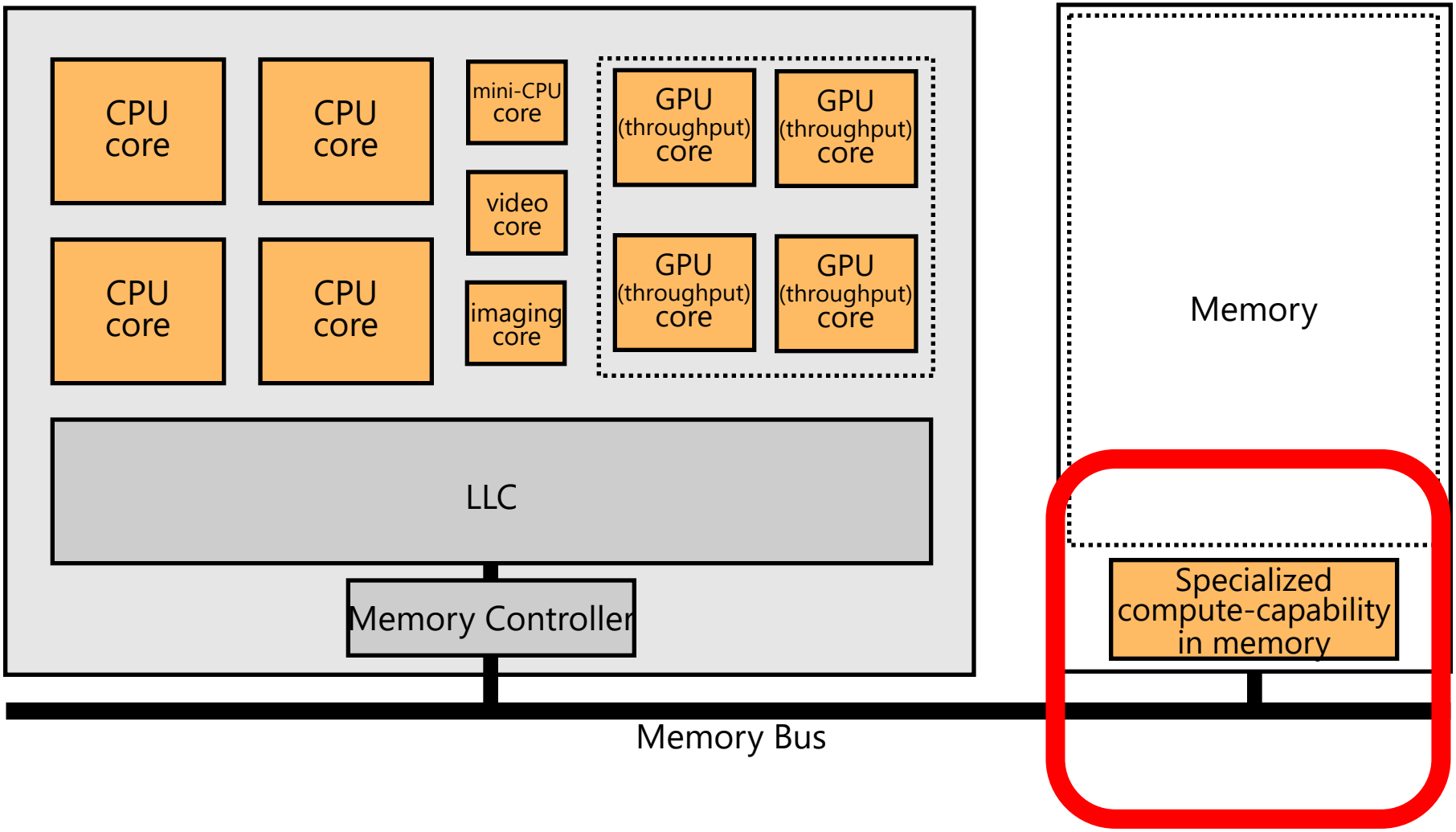
- Seyyed Hossein SeyyedAghaei Rezaei, Mehdi Modarressi, Rachata Ausavarungnirun, Mohammad Sadrosadati, Onur Mutlu, and Masoud Daneshtalab,
  "NoM: Network-on-Memory for Inter-Bank Data Transfer in Highly-Banked Memories"
  IEEE Computer Architecture Letters (***CAL***), 2020.

NoM: Network-on-Memory for Inter-bank Data Transfer in Highly-banked Memories

Seyyed Hossein SeyyedAghaei Rezaei[1] Mehdi Modarressi[1,3] Rachata Ausavarungnirun[2]
Mohammad Sadrosadati[3] Onur Mutlu[4] Masoud Daneshtalab[5]

[1]University of Tehran [2]King Mongkut's University of Technology North Bangkok [3]Institute for Research in Fundamental Sciences
[4]ETH Zürich [5]Mälardalens University

Mindset: Memory as an Accelerator



Memory similar to a "conventional" accelerator

Extensions and Follow-Up Work (II)

- Can this idea be evaluated on a real system? How?
  - Yes, see the ComputeDRAM [MICRO 2019] & PiDRAM [TACO 2022]
- Can similar ideas be used in other types of memories? Phase Change Memory? RRAM? STT-MRAM?
  - Yes, see the Pinatubo paper [DAC 2016]
- Can charge sharing properties be used for other functions?
  - Yes, see the D-RaNGe [HPCA 2019], DL-PUF [HPCA 2018], QUAC-TRNG [ISCA 2021] works on random numbers & PUFs
- Can we have more efficient solutions to
  - Cache coherence (minimize overhead)
  - Data reuse after copy and initialization

Real Processing Using Memory Prototype

- End-to-end RowClone & TRNG using off-the-shelf DRAM chips
- Idea: Violate DRAM timing parameters to mimic RowClone

PiDRAM: A Holistic End-to-end FPGA-based Framework for Processing-in-DRAM

Ataberk Olgun[§†] Juan Gómez Luna[§] Konstantinos Kanellopoulos[§] Behzad Salami[§*]
Hasan Hassan[§] Oğuz Ergin[†] Onur Mutlu[§]

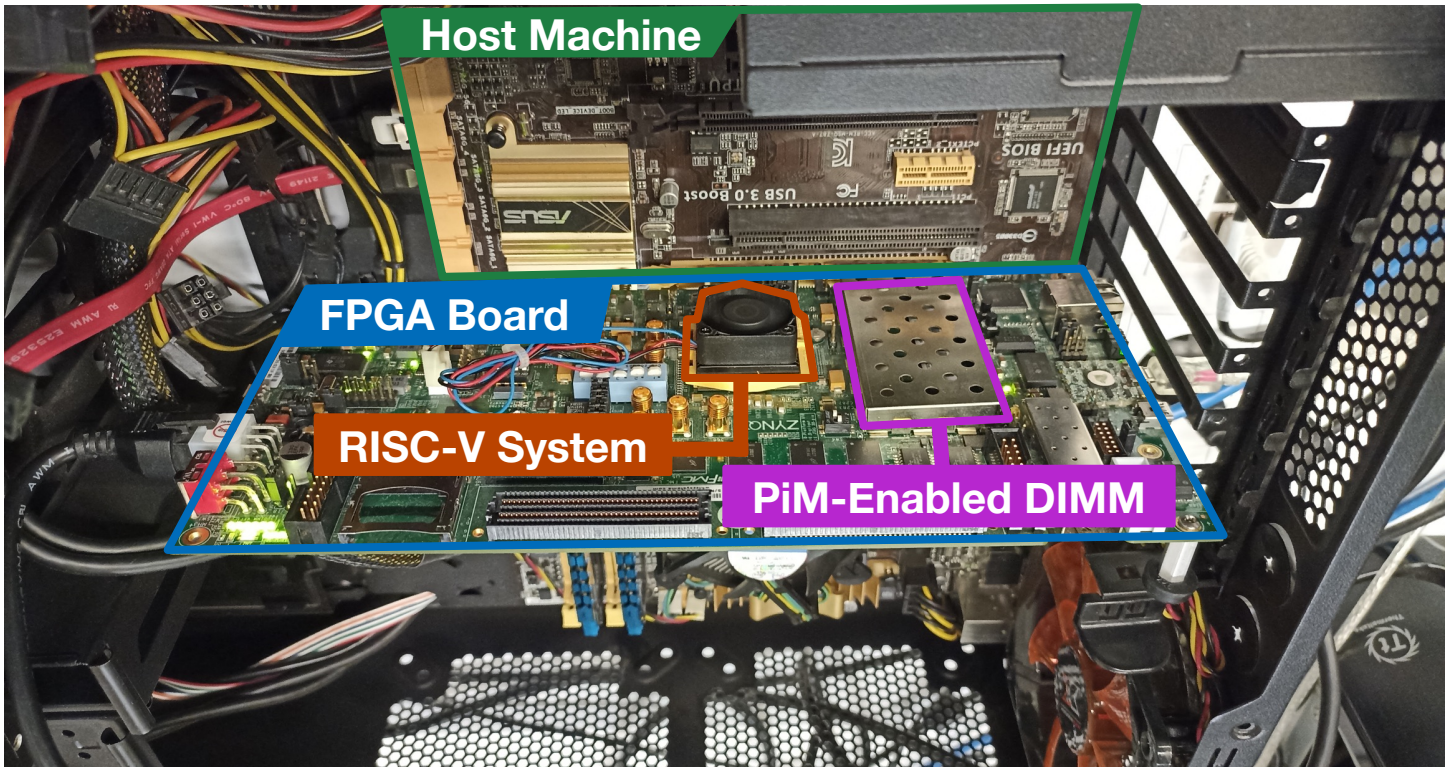
[§]ETH Zürich [†]TOBB ETÜ [*]BSC

https://arxiv.org/pdf/2111.00082.pdf

https://github.com/cmu-safari/pidram

https://www.youtube.com/watch?v=qeukNs5XI3g&t=4192s

Real Processing-using-Memory Prototype



https://arxiv.org/pdf/2111.00082.pdf

https://github.com/cmu-safari/pidram

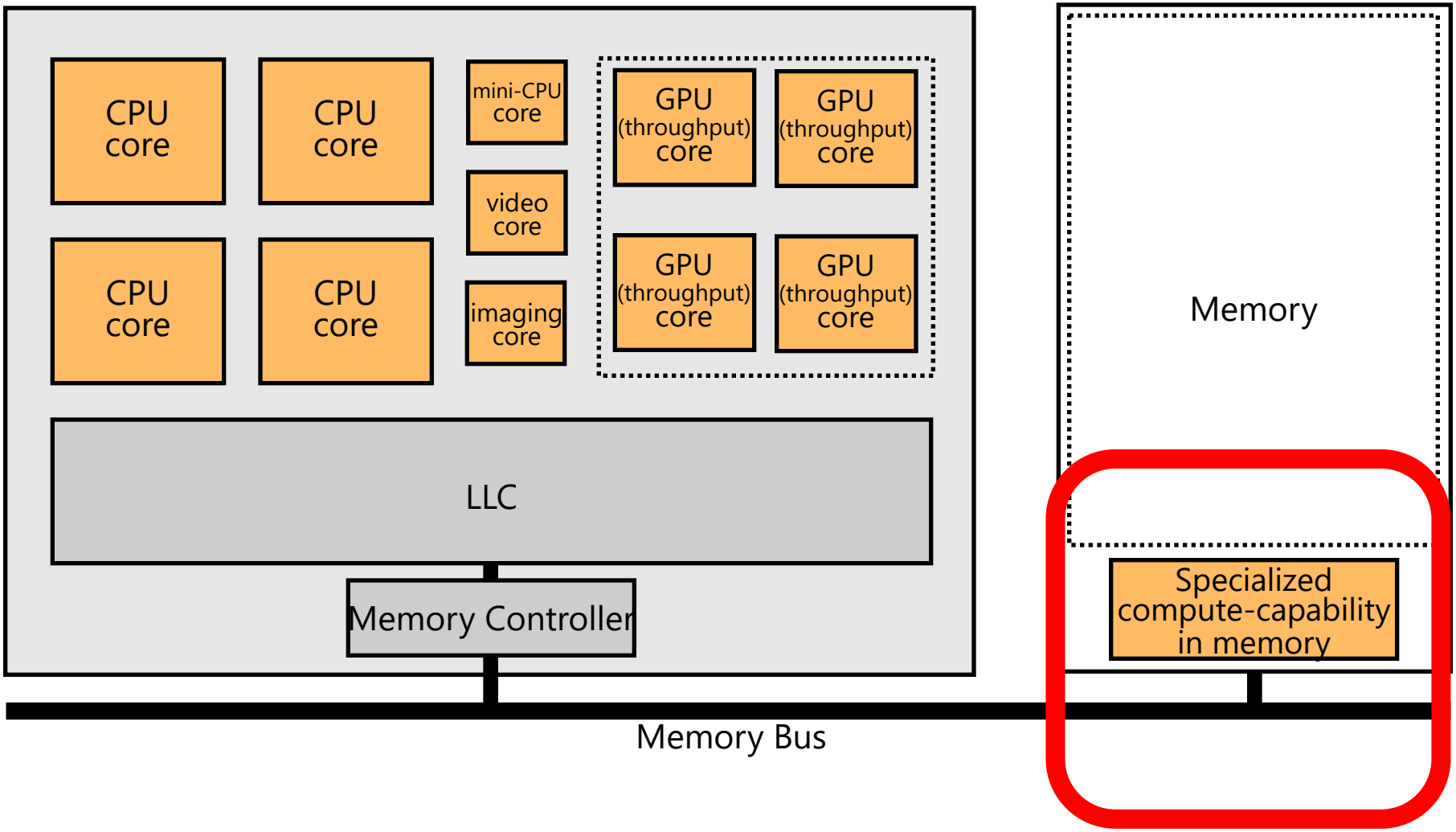
https://www.youtube.com/watch?v=qeukNs5XI3g&t=4192s

Takeaways

Key Takeaways

- A novel method to accelerate data copy and initialization
- Simple and effective
- Hardware/software cooperative
- Good potential for work building on it to extend it
  - To different granularities
  - To make things more efficient and effective
  - Many works have already built on the paper (see LISA, FIGARO, NoM, Ambit, SIMDRAM, ComputeDRAM, PiDRAM, and other works in Google Scholar)
- Easy to read and understand paper

RowClone: Memory as an Accelerator



Memory similar to a "conventional" accelerator

Mindset: Processing using DRAM

- DRAM has great capability to perform bulk data movement and computation internally with small changes
  - Can exploit internal connectivity to move data
  - Can exploit analog computation capability
  - …

- Examples: RowClone, In-DRAM AND/OR, Gather/Scatter DRAM
  - RowClone: Fast and Efficient In-DRAM Copy and Initialization of Bulk Data (Seshadri et al., MICRO 2013)
  - Fast Bulk Bitwise AND and OR in DRAM (Seshadri et al., IEEE CAL 2015)
  - Gather-Scatter DRAM: In-DRAM Address Translation to Improve the Spatial Locality of Non-unit Strided Accesses (Seshadri et al., MICRO 2015)
  - "Ambit: In-Memory Accelerator for Bulk Bitwise Operations Using Commodity DRAM Technology" (Seshadri et al., MICRO 2017)

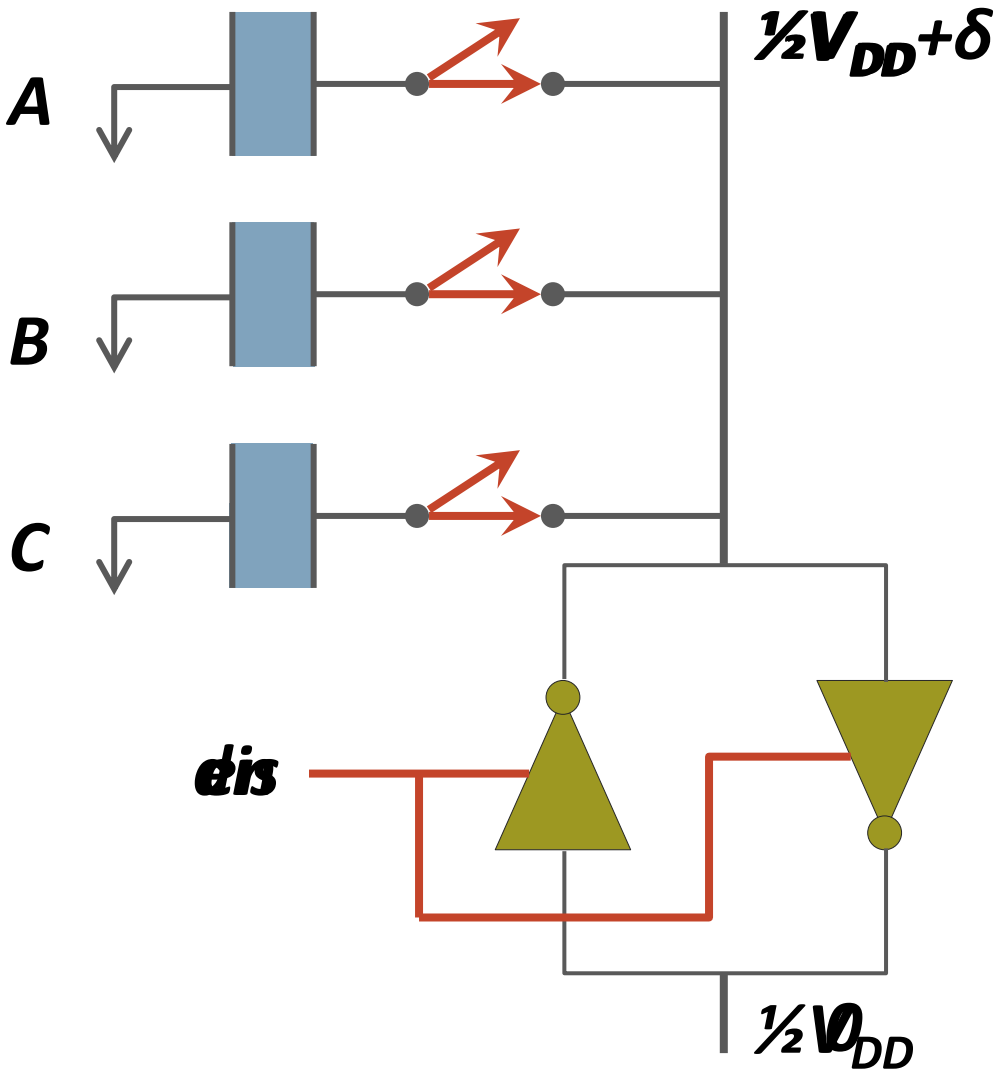
In-Memory Bulk Bitwise Operations

In-Memory Bulk Bitwise Operations

- We can support in-DRAM COPY, ZERO, AND, OR, NOT, MAJ
- At low cost
- Using inherent analog computation capability of DRAM
  - Idea: activating multiple rows performs computation
- 30-60X performance and energy improvement
  - Seshadri+, "Ambit: In-Memory Accelerator for Bulk Bitwise Operations Using Commodity DRAM Technology," MICRO 2017.

- New memory technologies enable even more opportunities
  - Memristors, resistive RAM, phase change mem, STT-MRAM, …
  - Can operate on data with minimal movement

In-DRAM AND/OR: Triple Row Activation



Final State
AB + BC + AC

C(A + B) + ~C(AB)

Seshadri+, "Fast Bulk Bitwise AND and OR in DRAM", IEEE CAL 2015.

In-DRAM Bulk Bitwise AND/OR Operation

- BULKAND A, B → C
- Semantics: Perform a bitwise AND of two rows A and B and store the result in row C

- R0 – reserved zero row, R1 – reserved one row
- D1, D2, D3 – Designated rows for triple activation

1. RowClone A into D1
2. RowClone B into D2
3. RowClone R0 into D3
4. ACTIVATE D1,D2,D3
5. RowClone Result into C

More on In-DRAM Bulk AND/OR

- Vivek Seshadri, Kevin Hsieh, Amirali Boroumand, Donghyuk Lee, Michael A. Kozuch, Onur Mutlu, Phillip B. Gibbons, and Todd C. Mowry,
  "Fast Bulk Bitwise AND and OR in DRAM"
  IEEE Computer Architecture Letters (***CAL***), April 2015.

Fast Bulk Bitwise AND and OR in DRAM

Vivek Seshadri*, Kevin Hsieh*, Amirali Boroumand*, Donghyuk Lee*,
Michael A. Kozuch†, Onur Mutlu*, Phillip B. Gibbons†, Todd C. Mowry*

*Carnegie Mellon University †Intel Pittsburgh

In-DRAM NOT: Dual Contact Cell

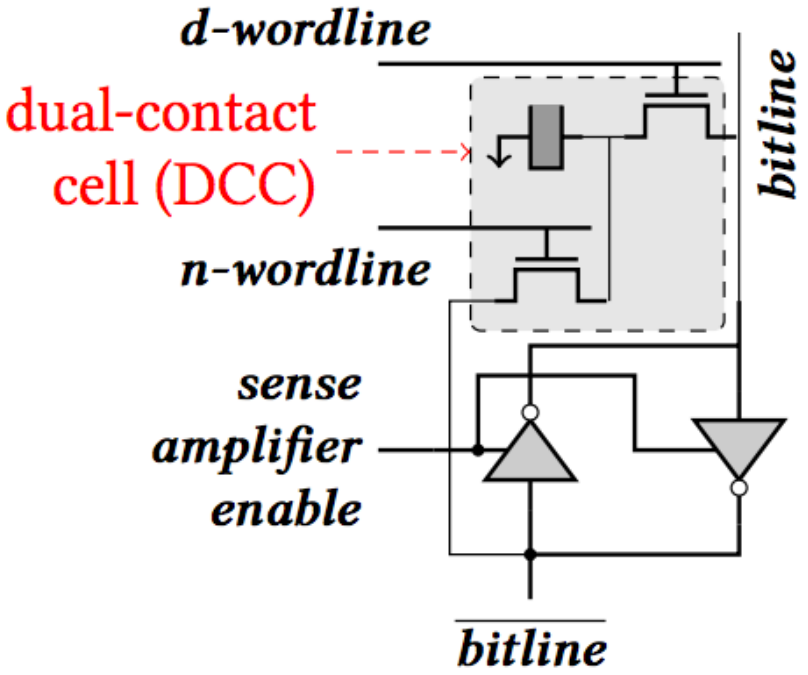


Figure 5: A dual-contact cell connected to both ends of a sense amplifier

Idea:

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

Seshadri+, "Ambit: In-Memory Accelerator for Bulk Bitwise Operations using Commodity DRAM Technology," MICRO 2017.

In-DRAM NOT Operation

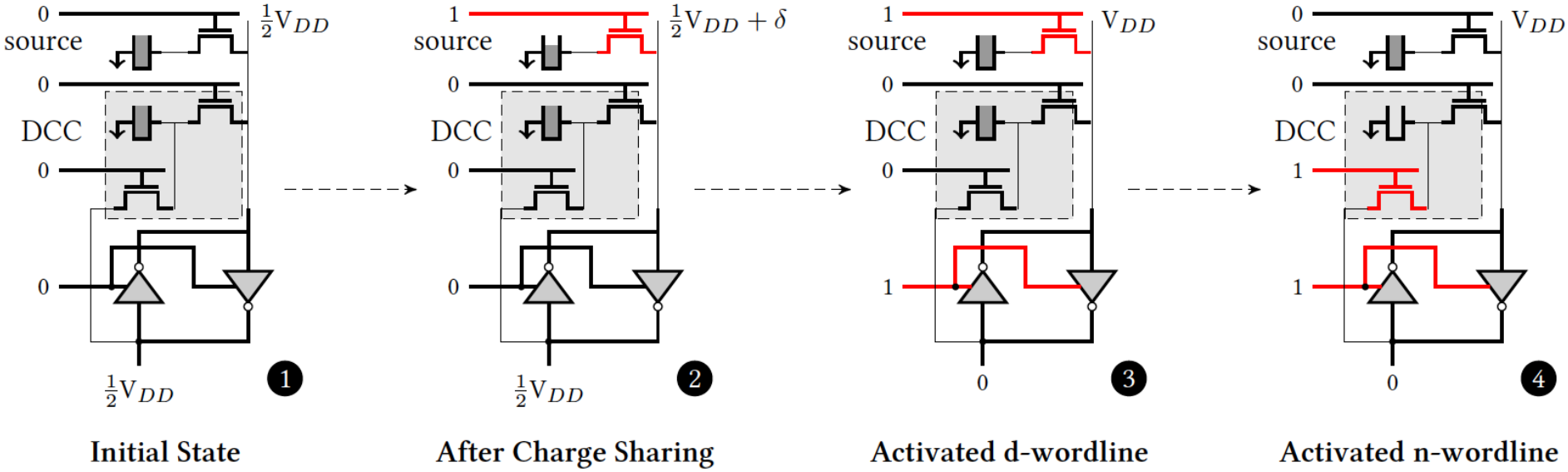


Figure 5: Bitwise NOT using a dual contact capacitor

Seshadri+, "Ambit: In-Memory Accelerator for Bulk Bitwise Operations using Commodity DRAM Technology," MICRO 2017.

Performance: In-DRAM Bitwise Operations

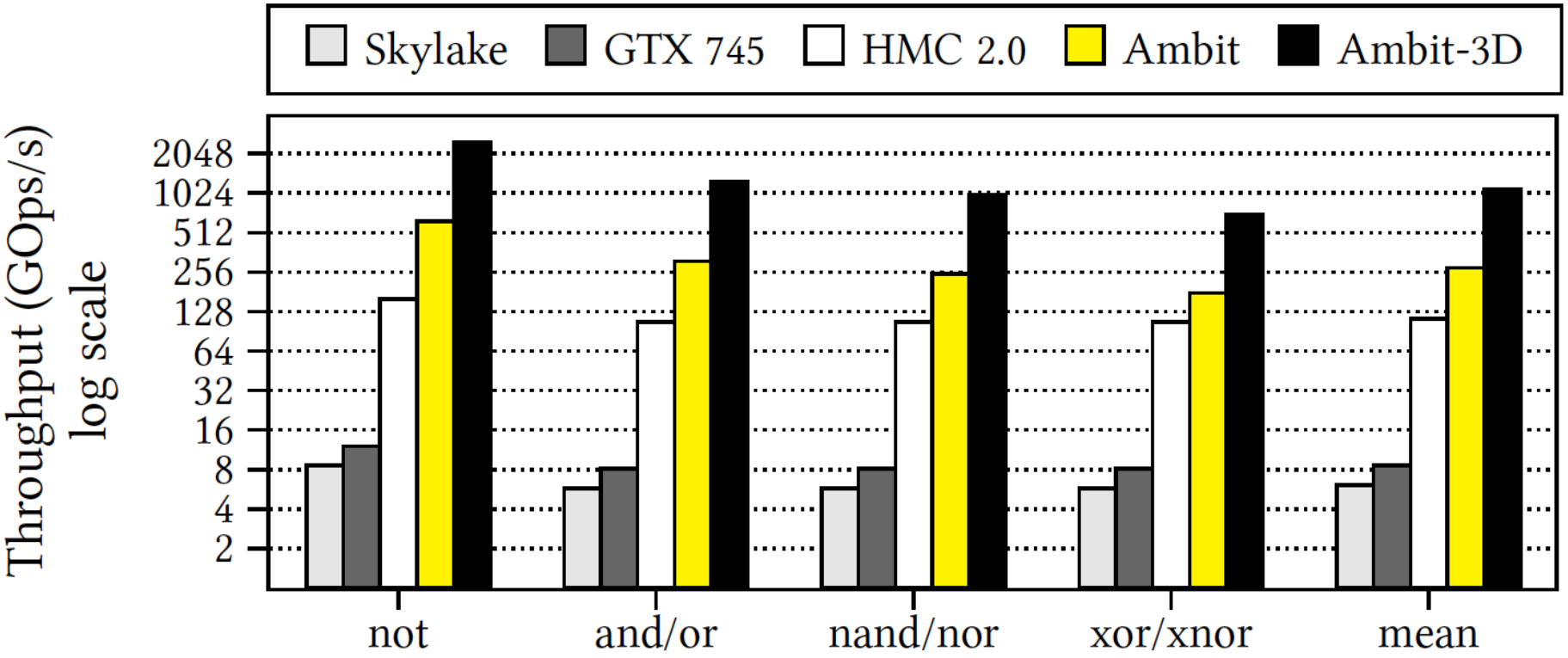


Figure 9: Throughput of bitwise operations on various systems.

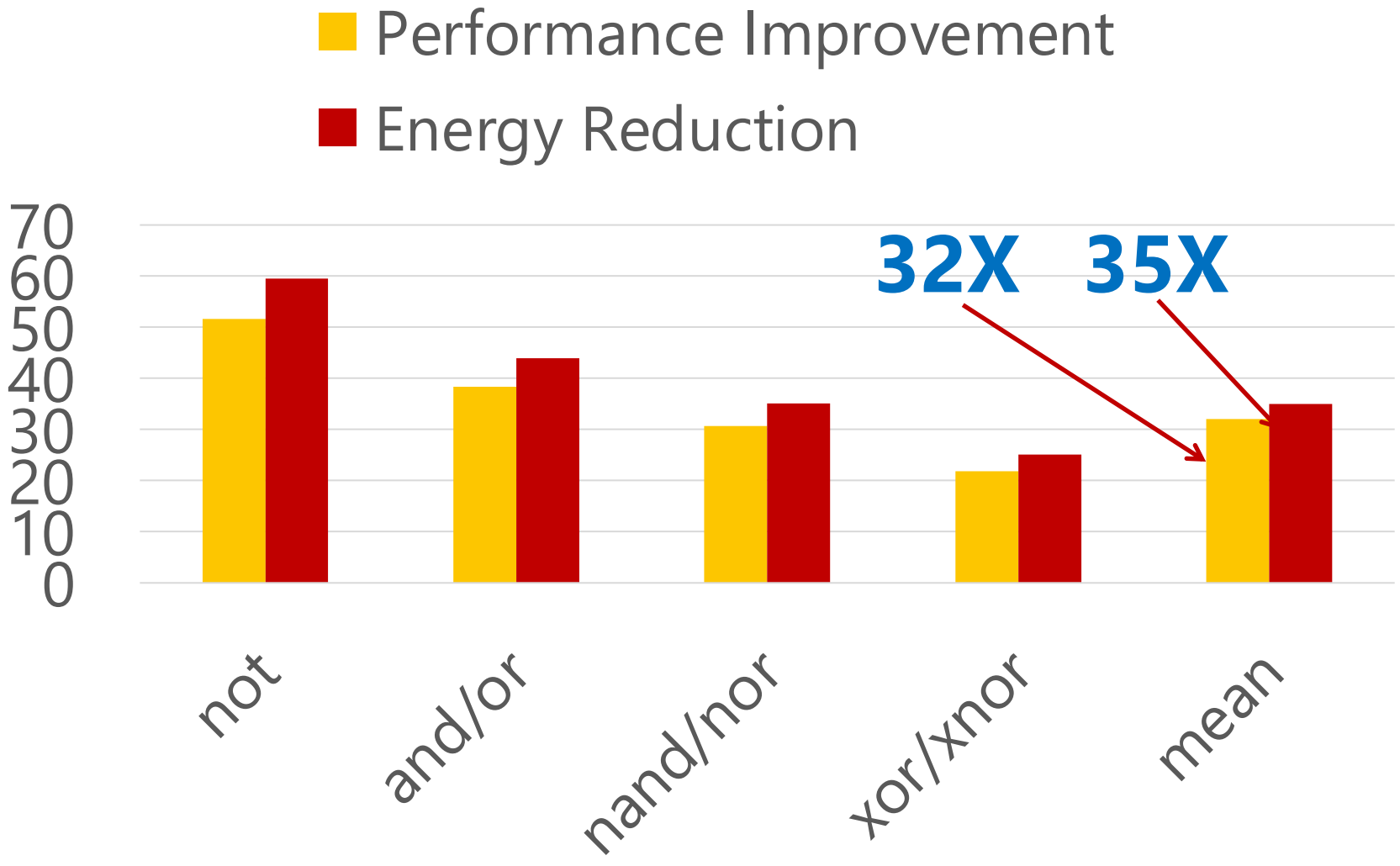
Energy of In-DRAM Bitwise Operations

	Design	not	and/or	nand/nor	xor/xnor
DRAM &	**DDR3**	93.7	137.9	137.9	137.9
Channel Energy	**Ambit**	1.6	3.2	4.0	5.5
(nJ/KB)	(↓)	59.5X	43.9X	35.1X	25.1X

Table 3: Energy of bitwise operations. (↓) indicates energy reduction of Ambit over the traditional DDR3-based design.

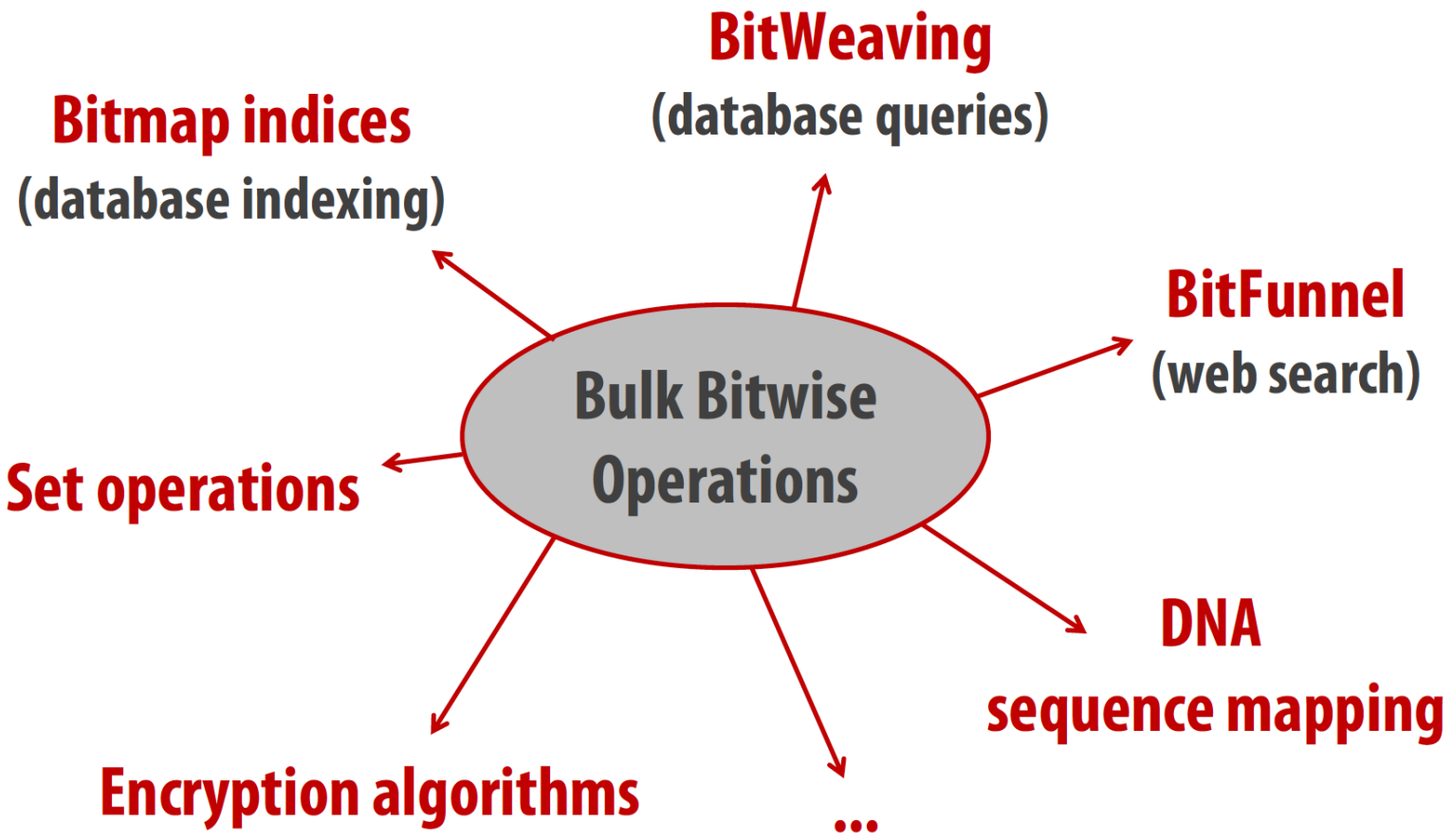
Seshadri+, "Ambit: In-Memory Accelerator for Bulk Bitwise Operations using Commodity DRAM Technology," MICRO 2017.

Ambit vs. DDR3: Performance and Energy



Seshadri+, "Ambit: In-Memory Accelerator for Bulk Bitwise Operations using Commodity DRAM Technology," MICRO 2017.

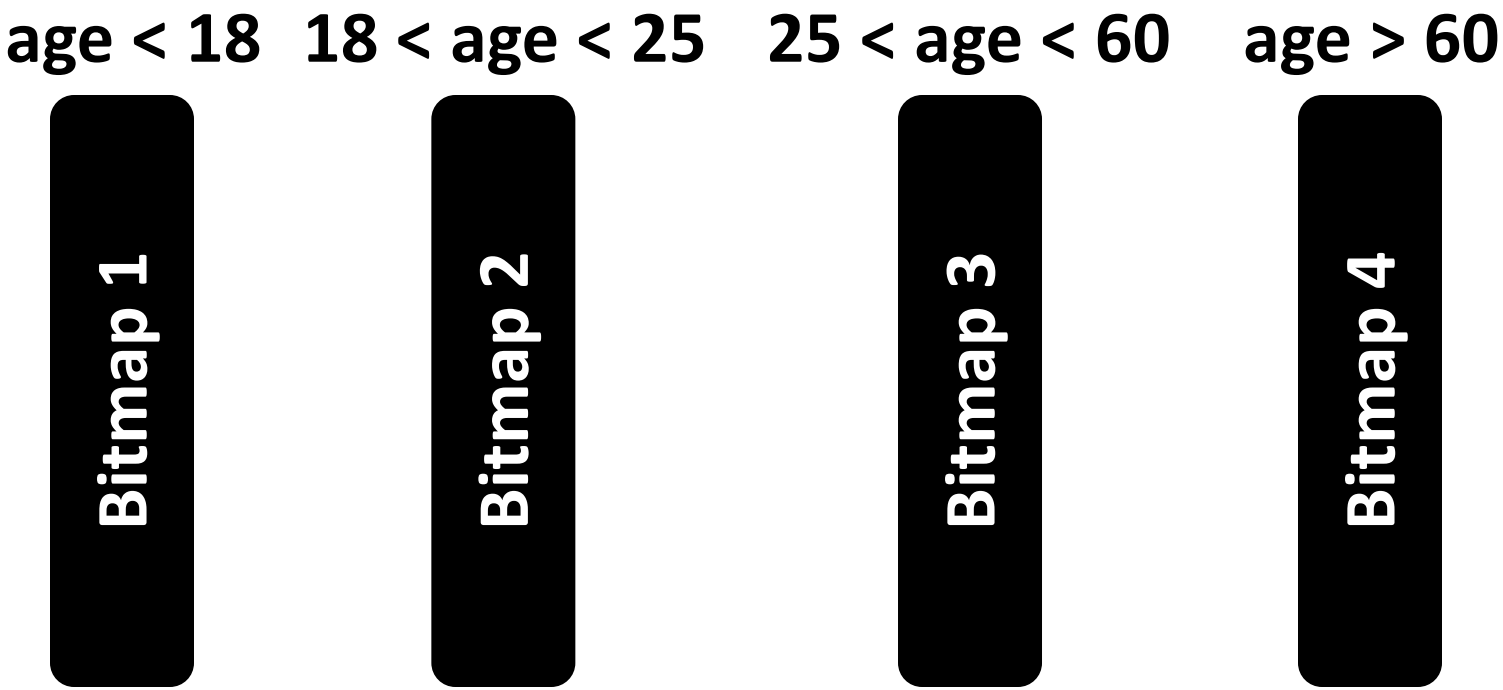
Bulk Bitwise Operations in Workloads



[1] Li and Patel, BitWeaving, SIGMOD 2013
[2] Goodwin+, BitFunnel, SIGIR 2017

Example Data Structure: Bitmap Index

- Alternative to B-tree and its variants
- Efficient for performing *range queries* and *joins*
- **Many bitwise operations to perform a query**



Performance: Bitmap Index on Ambit

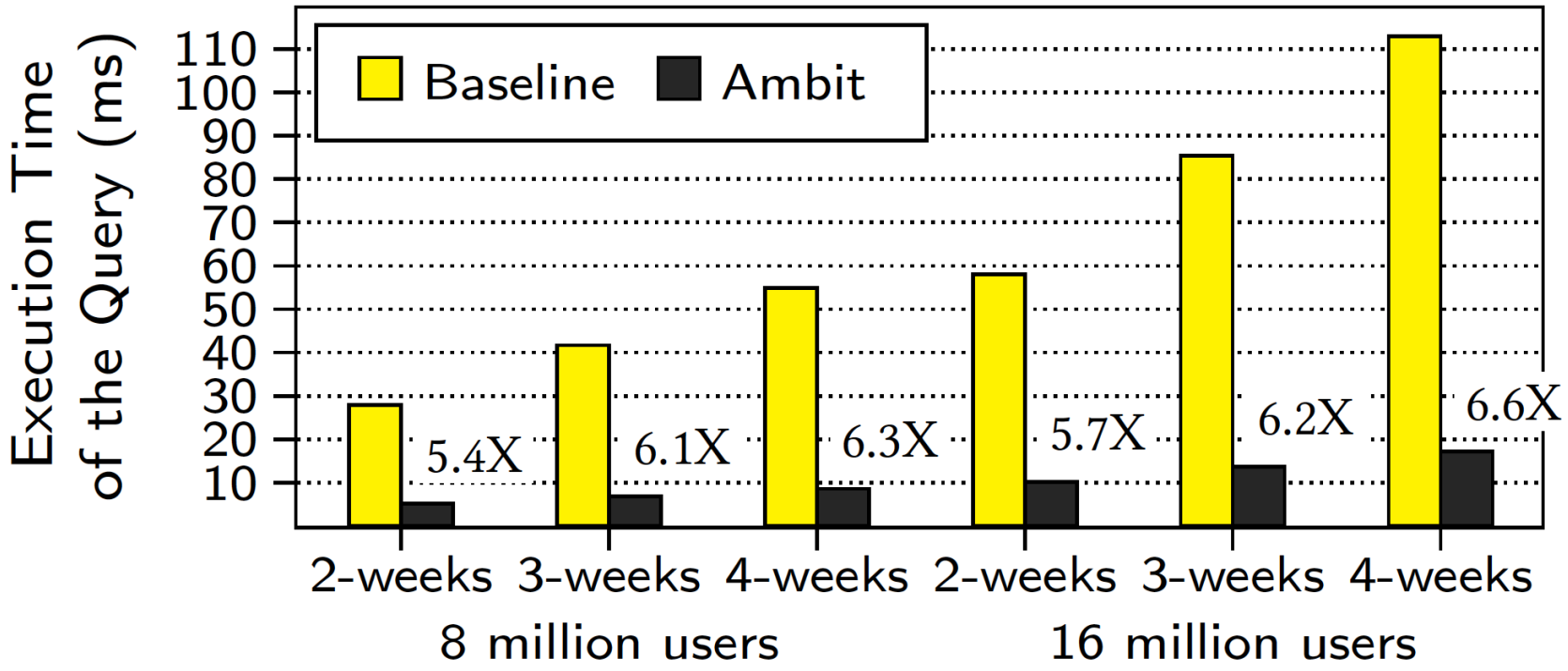


Figure 10: Bitmap index performance. The value above each bar indicates the reduction in execution time due to Ambit.

>5.4-6.6X Performance Improvement

Seshadri+, "Ambit: In-Memory Accelerator for Bulk Bitwise Operations using Commodity DRAM Technology," MICRO 2017.

Performance: BitWeaving on Ambit

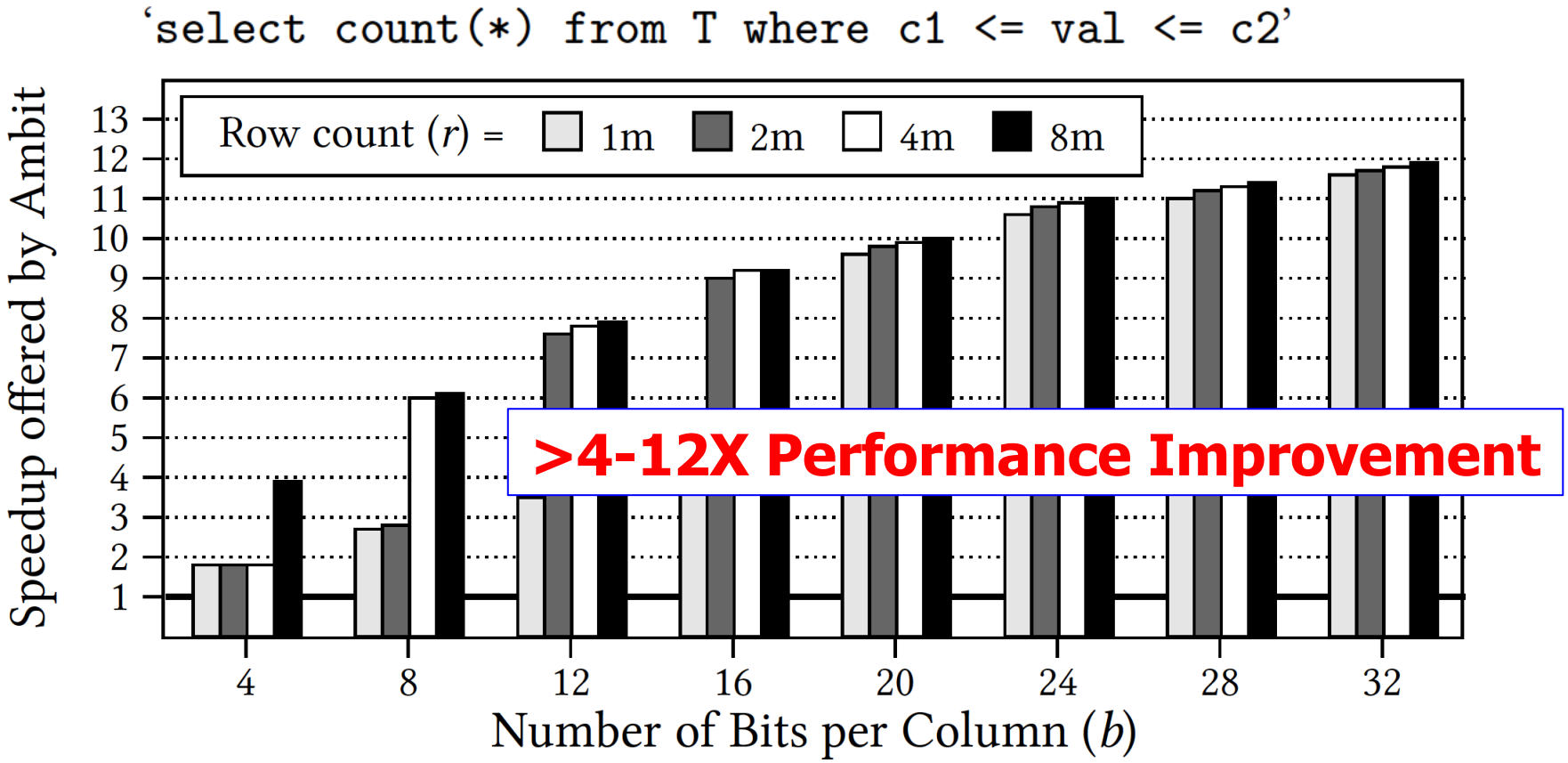


Figure 11: Speedup offered by Ambit over baseline CPU with SIMD for BitWeaving

Seshadri+, "Ambit: In-Memory Accelerator for Bulk Bitwise Operations using Commodity DRAM Technology," MICRO 2017.

More on In-DRAM Bulk AND/OR

- Vivek Seshadri, Kevin Hsieh, Amirali Boroumand, Donghyuk Lee, Michael A. Kozuch, Onur Mutlu, Phillip B. Gibbons, and Todd C. Mowry,
  "Fast Bulk Bitwise AND and OR in DRAM"
  IEEE Computer Architecture Letters (***CAL***), April 2015.

Fast Bulk Bitwise AND and OR in DRAM

Vivek Seshadri*, Kevin Hsieh*, Amirali Boroumand*, Donghyuk Lee*,
Michael A. Kozuch†, Onur Mutlu*, Phillip B. Gibbons†, Todd C. Mowry*

*Carnegie Mellon University †Intel Pittsburgh

More on In-DRAM Bitwise Operations

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

Ambit: In-Memory Accelerator for Bulk Bitwise Operations Using Commodity DRAM Technology

Vivek Seshadri[1,5] Donghyuk Lee[2,5] Thomas Mullins[3,5] Hasan Hassan[4] Amirali Boroumand[5]
Jeremie Kim[4,5] Michael A. Kozuch[3] Onur Mutlu[4,5] Phillip B. Gibbons[5] Todd C. Mowry[5]

[1]Microsoft Research India [2]NVIDIA Research [3]Intel [4]ETH Zürich [5]Carnegie Mellon University

More on In-DRAM Bulk Bitwise Execution

- Vivek Seshadri and Onur Mutlu,
  "In-DRAM Bulk Bitwise Execution Engine"
  Invited Book Chapter in Advances in Computers, to appear in 2020.
  [Preliminary arXiv version]

In-DRAM Bulk Bitwise Execution Engine

Vivek Seshadri
Microsoft Research India
visesha@microsoft.com

Onur Mutlu
ETH Zürich
onur.mutlu@inf.ethz.ch

Generalizing In-DRAM Bulk Bitwise Computing

SIMDRAM Framework

- Nastaran Hajinazar, Geraldo F. Oliveira, Sven Gregorio, Joao Dinis Ferreira, Nika Mansouri Ghiasi, Minesh Patel, Mohammed Alser, Saugata Ghose, Juan Gomez-Luna, and Onur Mutlu, **"SIMDRAM: An End-to-End Framework for Bit-Serial SIMD Computing in DRAM"** *Proceedings of the 26th International Conference on Architectural Support for Programming Languages and Operating Systems* (***ASPLOS***), Virtual, March-April 2021.
  [2-page Extended Abstract]
  [Short Talk Slides (pptx) (pdf)]
  [Talk Slides (pptx) (pdf)]
  [Short Talk Video (5 mins)]
  [Full Talk Video (27 mins)]

SIMDRAM: A Framework for Bit-Serial SIMD Processing using DRAM

*Nastaran Hajinazar[1,2] *Geraldo F. Oliveira[1] Sven Gregorio[1] João Dinis Ferreira[1]
Nika Mansouri Ghiasi[1] Minesh Patel[1] Mohammed Alser[1] Saugata Ghose[3]
Juan Gómez-Luna[1] Onur Mutlu[1]

[1]ETH Zürich [2]Simon Fraser University [3]University of Illinois at Urbana–Champaign

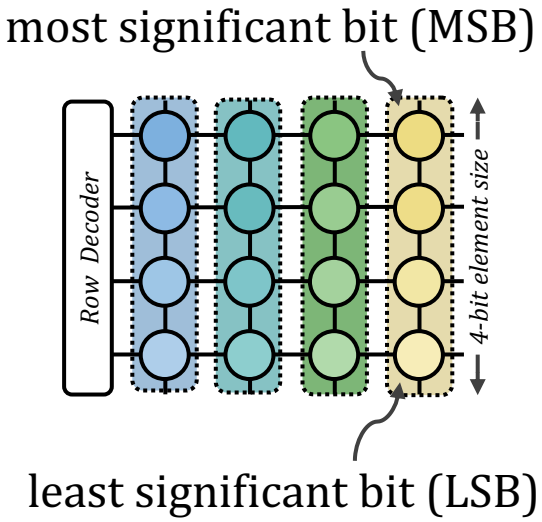
SIMDRAM Key Idea

- **SIMDRAM:** An end-to-end processing-using-DRAM framework that provides the programming interface, the ISA, and the hardware support for:
  - **Efficiently** computing **complex** operations in DRAM
  - Providing the ability to implement **arbitrary** operations as required
  - Using an **in-DRAM massively-parallel SIMD substrate** that requires **minimal** changes to DRAM architecture

SIMDRAM: PuM Substrate

- SIMDRAM framework is built around a DRAM substrate that enables two techniques:

(1) Vertical data layout

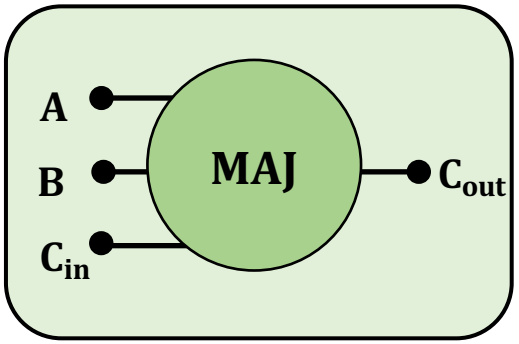


Pros compared to the conventional horizontal layout:

- Implicit shift operation
- Massive parallelism

(2) Majority-based computation

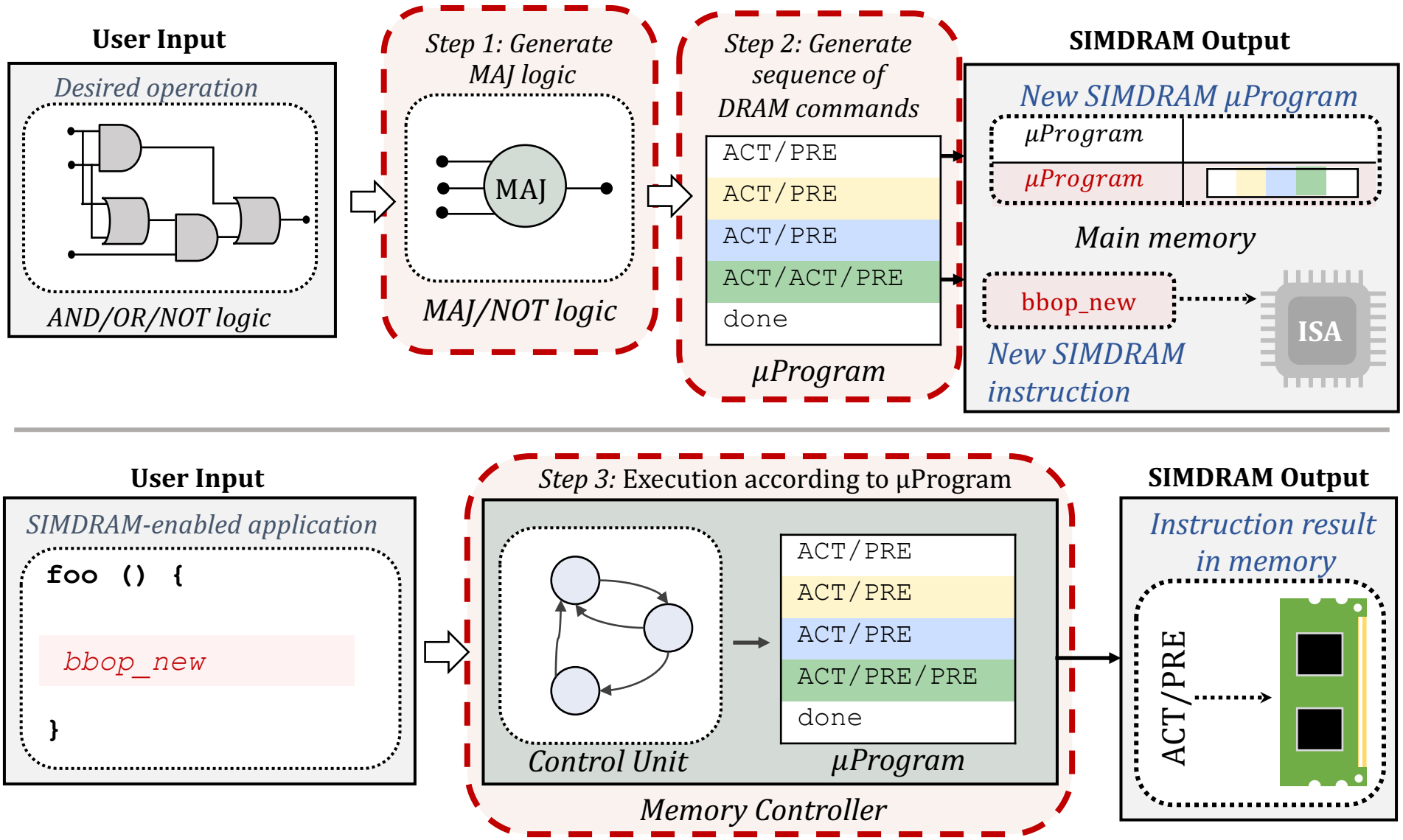
$C_{out} = AB + AC_{in} + BC_{in}$



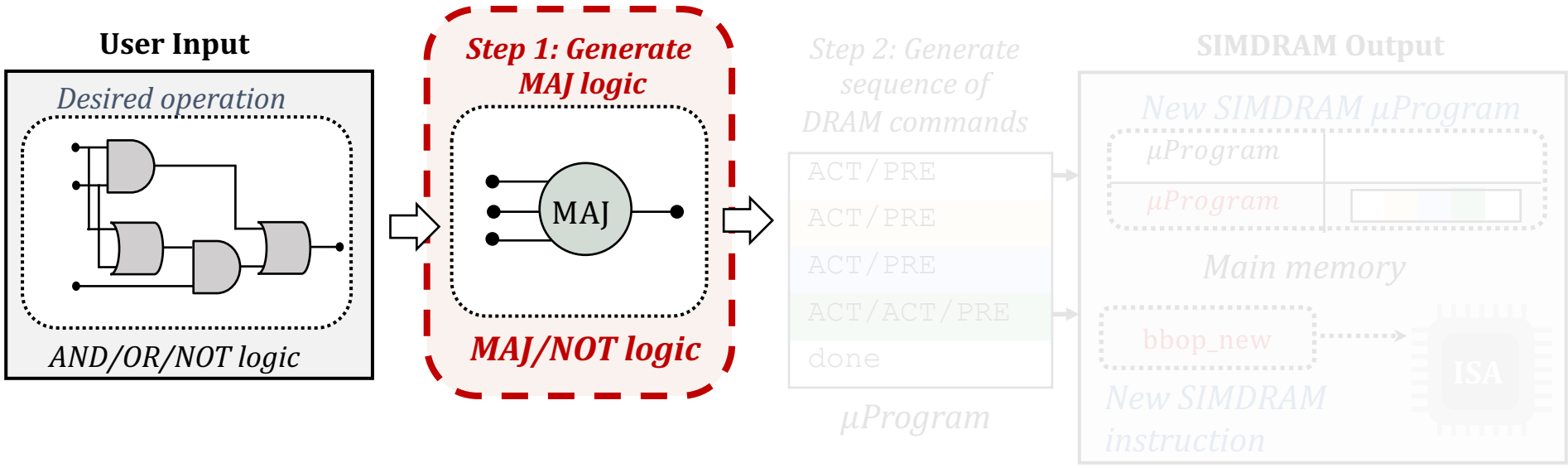
Pros compared to AND/OR/NOT-based computation:

- Higher performance
- Higher throughput
- Lower energy consumption

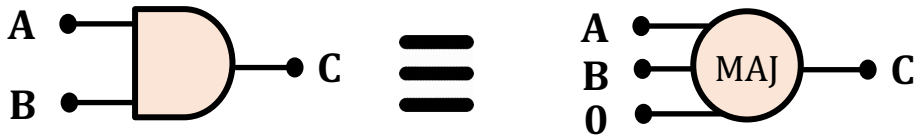
SIMDRAM Framework: Overview



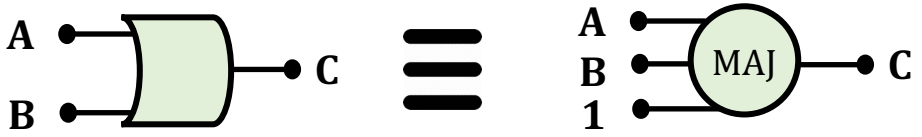
SIMDRAM Framework: Step 1



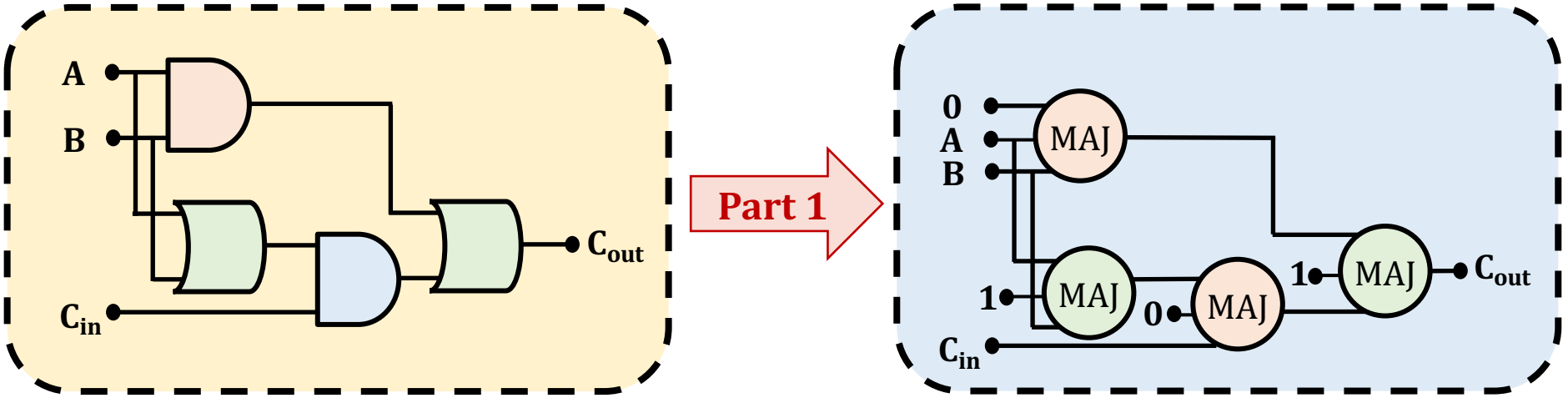
Step 1: Naïve MAJ/NOT Implementation



output is “1” only when A = B = “1”

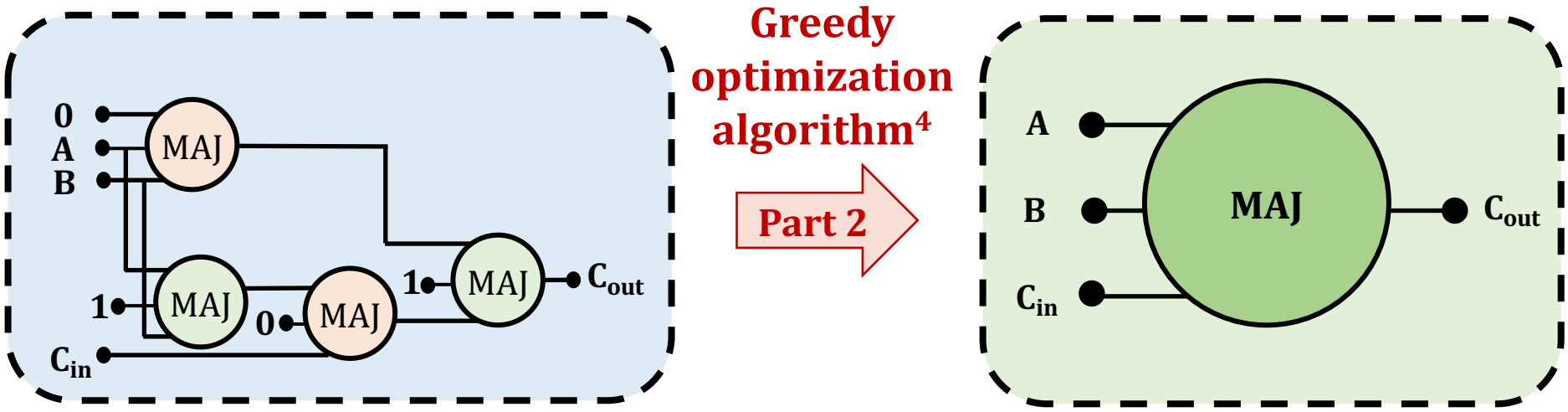


output is “0” only when A = B = “0”



Naïvely converting AND/OR/NOT-implementation to MAJ/NOT-implementation leads to an unoptimized circuit

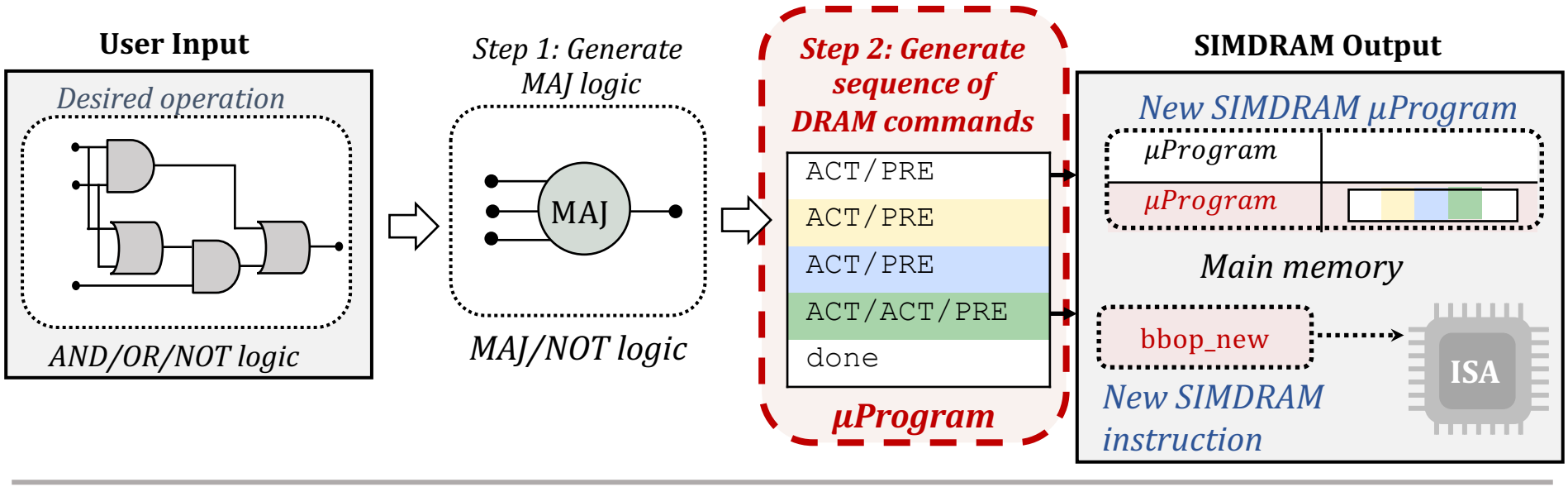
Step 1: Efficient MAJ/NOT Implementation



Step 1 generates an optimized MAJ/NOT-implementation of the desired operation

[4] L. Amarù et al, "*Majority-Inverter Graph: A Novel Data-Structure and Algorithms for Efficient Logic Optimization*", DAC, 2014.

SIMDRAM Framework: Step 2



Step 2: µProgram Generation

- **µProgram:** A series of microarchitectural operations (e.g., ACT/PRE) that SIMDRAM uses to execute SIMDRAM operation in DRAM

- **Goal of Step 2**: To generate the µProgram that executes the desired SIMDRAM operation in DRAM

Task 1: Allocate DRAM rows to the operands

Task 2: Generate µProgram

Step 2: µProgram Generation

- **µProgram:** A series of microarchitectural operations (e.g., ACT/PRE) that SIMDRAM uses to execute SIMDRAM operation in DRAM

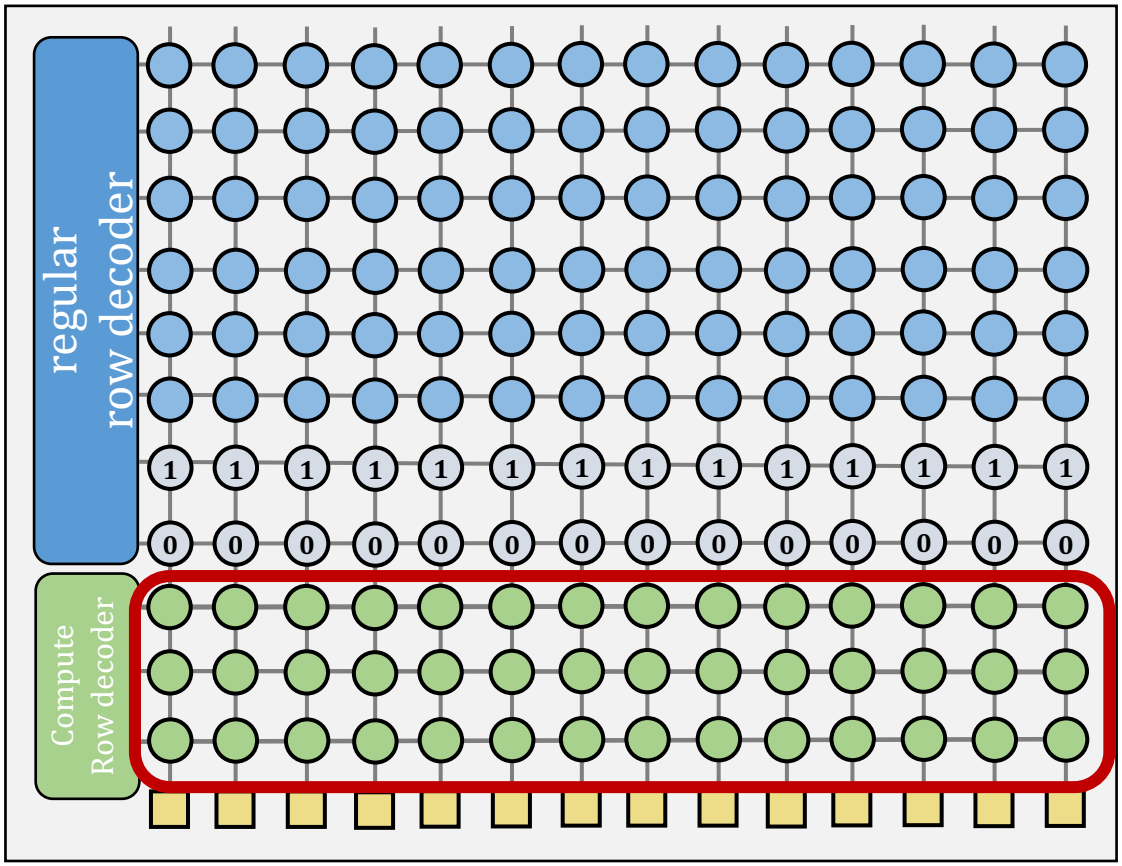
- **Goal of Step 2**: To generate the µProgram that executes the desired SIMDRAM operation in DRAM

Task 1: Allocate DRAM rows to the operands

Task 2: Generate µProgram

Task 1: Allocating DRAM Rows to Operands

- Allocation algorithm considers two constraints specific to processing-using-DRAM



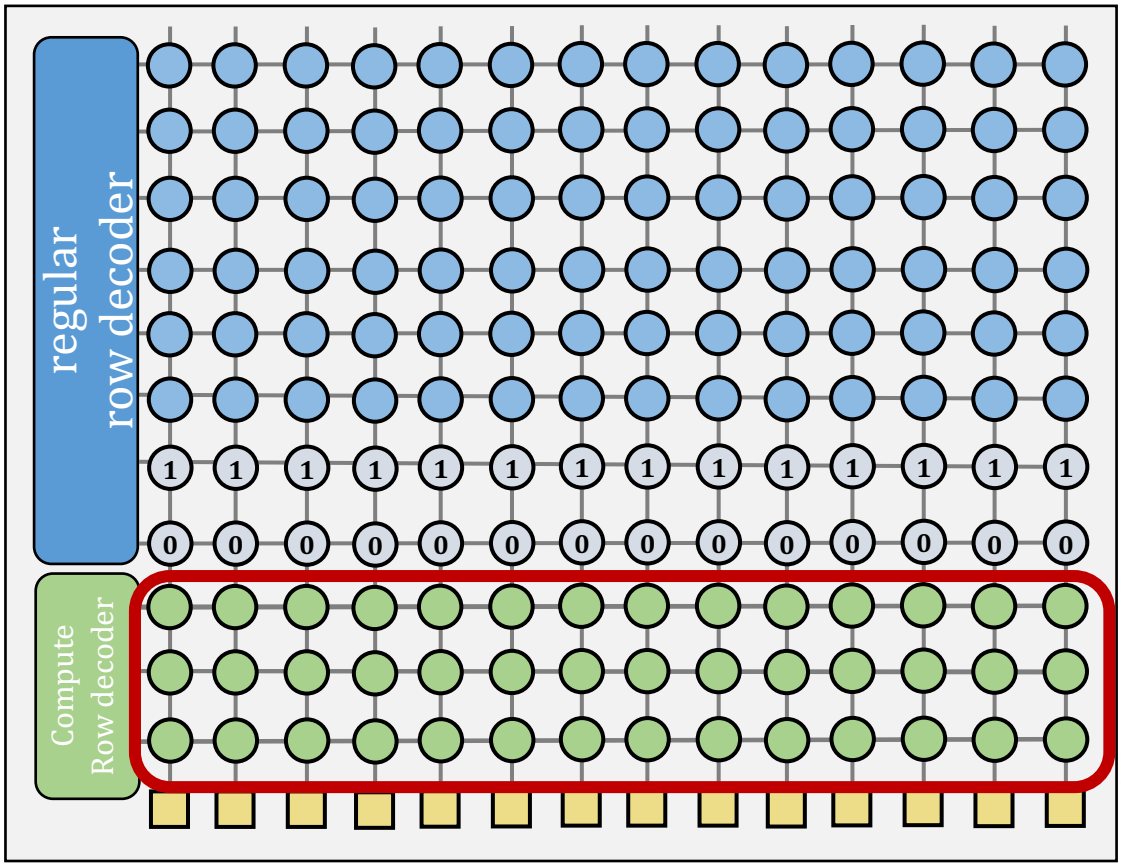
subarray organization

Constraint 1:
Limited number of rows reserved for computation

Compute rows

Task 1: Allocating DRAM Rows to Operands

- Allocation algorithm considers two constraints specific to processing-using-DRAM



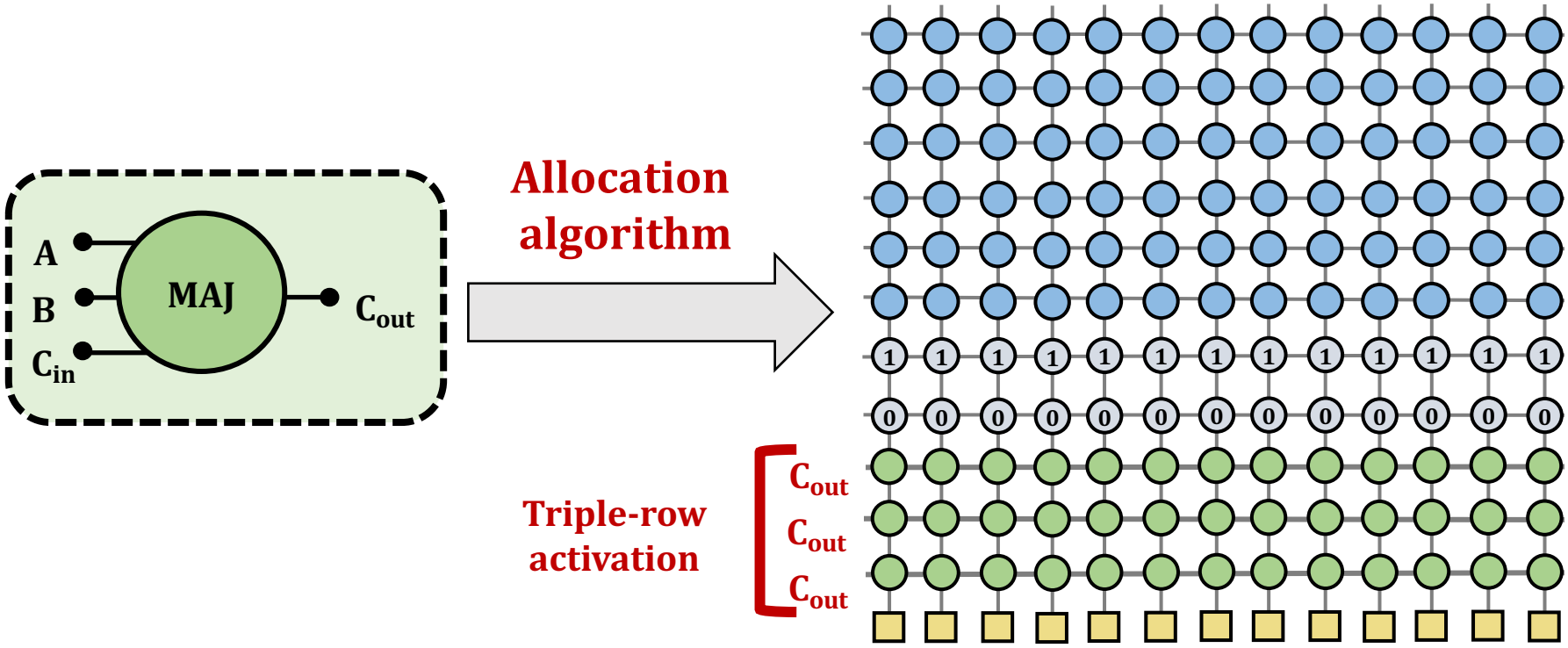
subarray organization

Constraint 2:
Destructive behavior of triple-row activation

Overwritten with MAJ output

Task 1: Allocating DRAM Rows to Operands

- Allocation algorithm:
  - Assigns as many inputs as the number of free compute rows
  - All three input rows contain the MAJ output and can be reused



Step 2: µProgram Generation

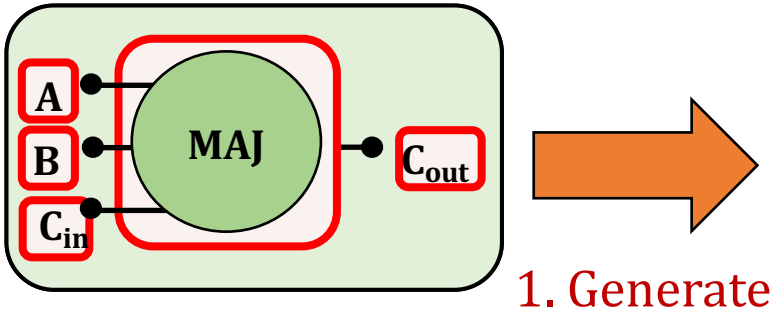
- **µProgram:** A series of microarchitectural operations (e.g., ACT/PRE) that SIMDRAM uses to execute SIMDRAM operation in DRAM

- **Goal of Step 2**: To generate the µProgram that executes the desired SIMDRAM operation in DRAM

Task 1: Allocate DRAM rows to the operands

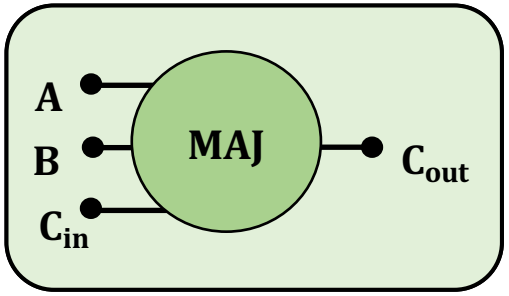
Task 2: Generate µProgram

Task 2: Generate an Initial μProgram



1. Generate μProgram

Task 2: Optimize the µProgram



1. Generate µProgram

Initial µProgram

1. Copy A to reserved row **(ACT/ACT/PRE)**

2. Copy B to reserved row **(ACT/ACT/PRE)**

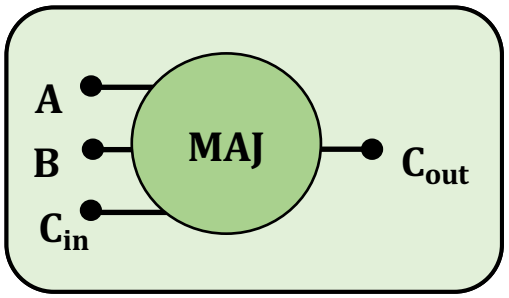
3. Copy C_{in} to reserved row **(ACT/ACT/PRE)**

4. Execute MAJ **(ACT/PRE)**

5. Copy C_{out} to destination row **(ACT/PRE)**

2. Optimize

Task 2: Optimize the µProgram



1. Generate µProgram

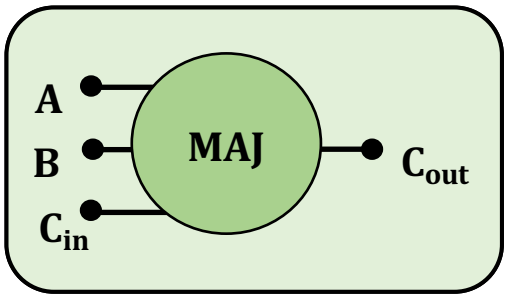
Initial µProgram

1. **Copy** A to reserved row **(ACT/ACT/PRE)**
2. **Copy** B to reserved row **(ACT/ACT/PRE)**
3. **Copy** C_{in} to reserved row **(ACT/ACT/PRE)**
4. Execute MAJ **(ACT/PRE)**
5. Copy C_{out} to destination row **(ACT/PRE)**

Coalesce row copies

2. Optimize

Task 2: Optimize the µProgram



1. Generate µProgram

Initial µProgram

1. Copy A to reserved row **(ACT/ACT/PRE)**

2. Copy B to reserved row **(ACT/ACT/PRE)**

3. Copy C_{in} to reserved row **(ACT/ACT/PRE)**

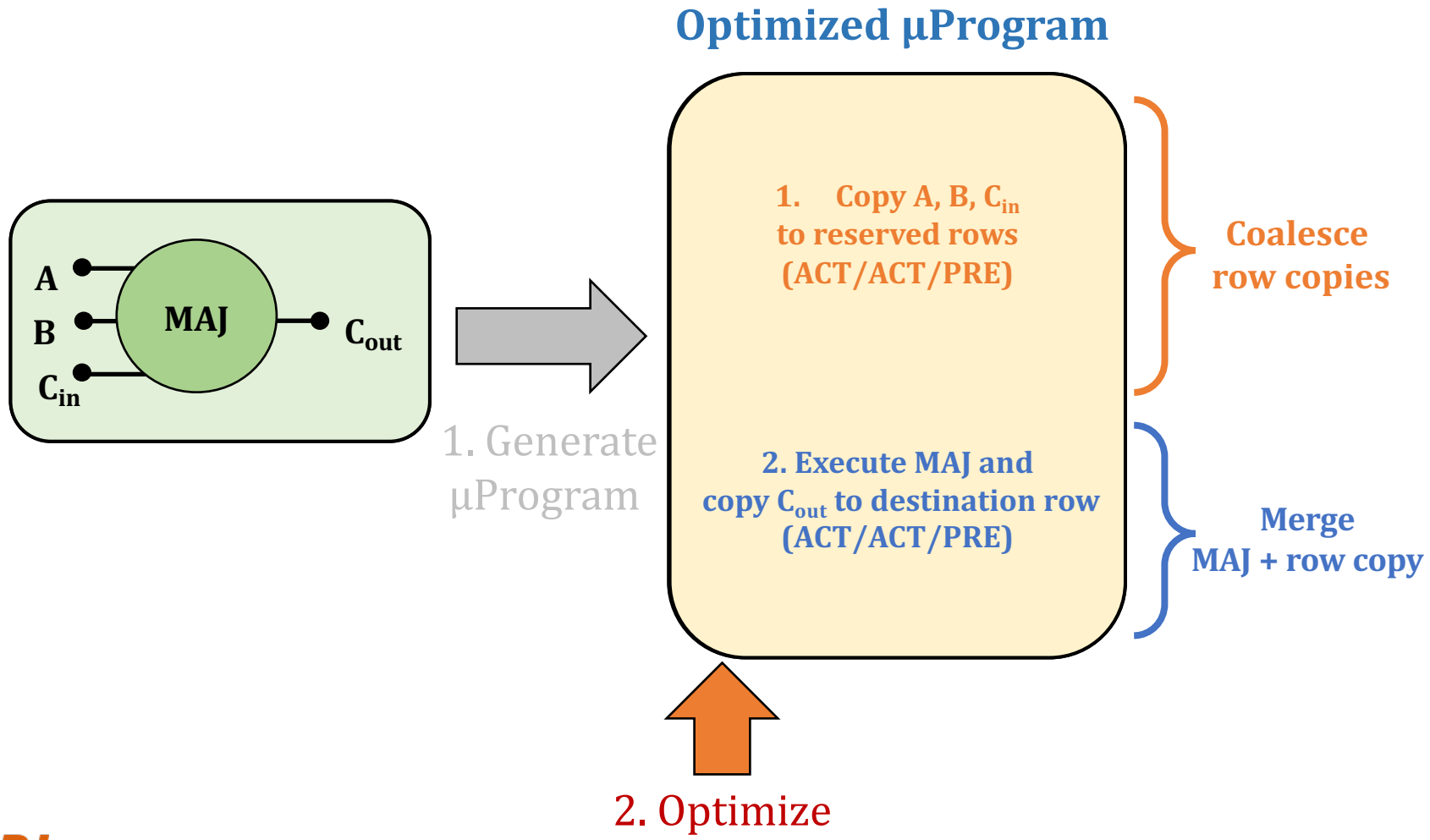
4. Execute **MAJ** **(ACT/PRE)**

5. **Copy** C_{out} to destination row **(ACT/PRE)**

Merge MAJ + row copy

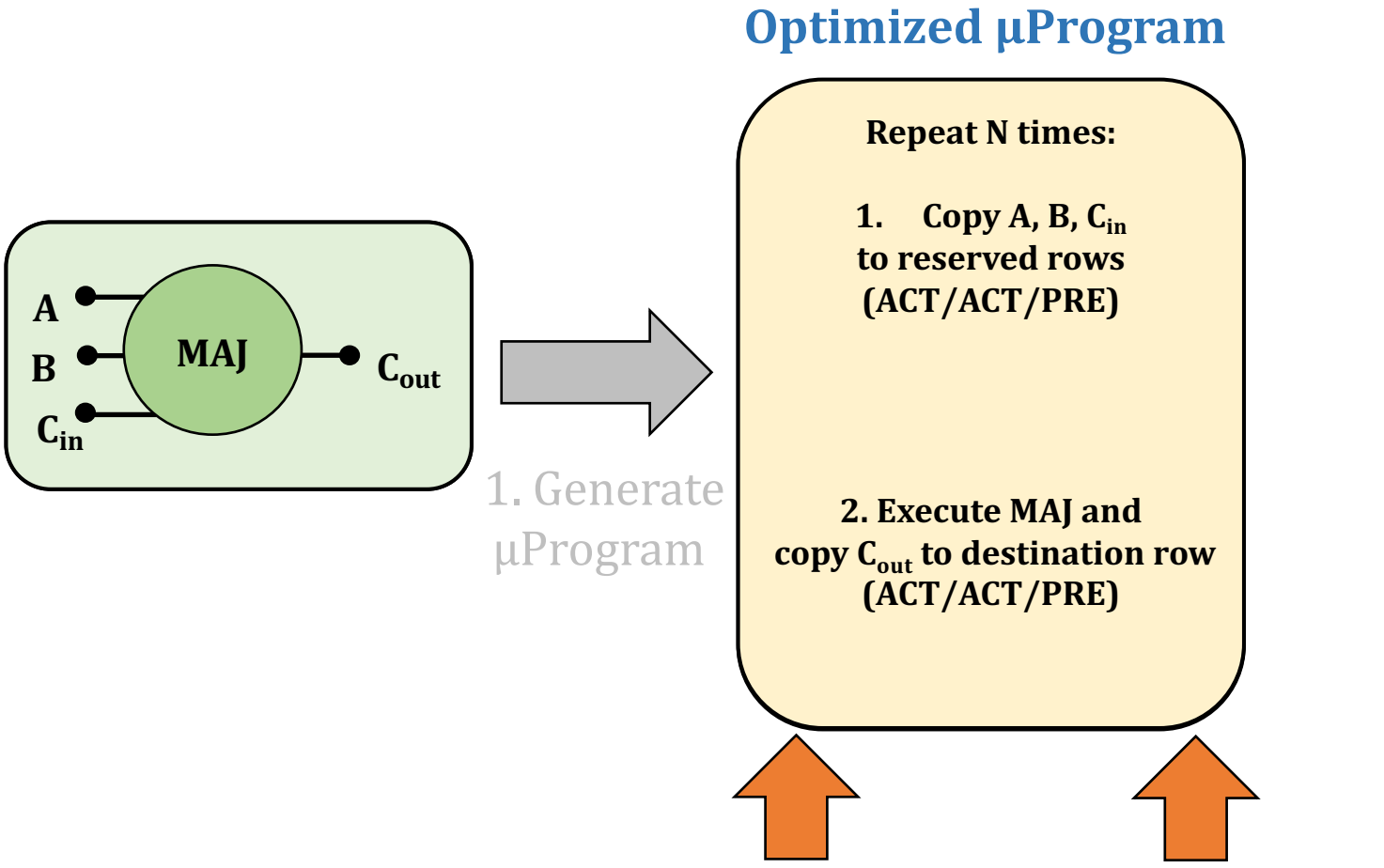
2. Optimize

Task 2: Optimize the µProgram



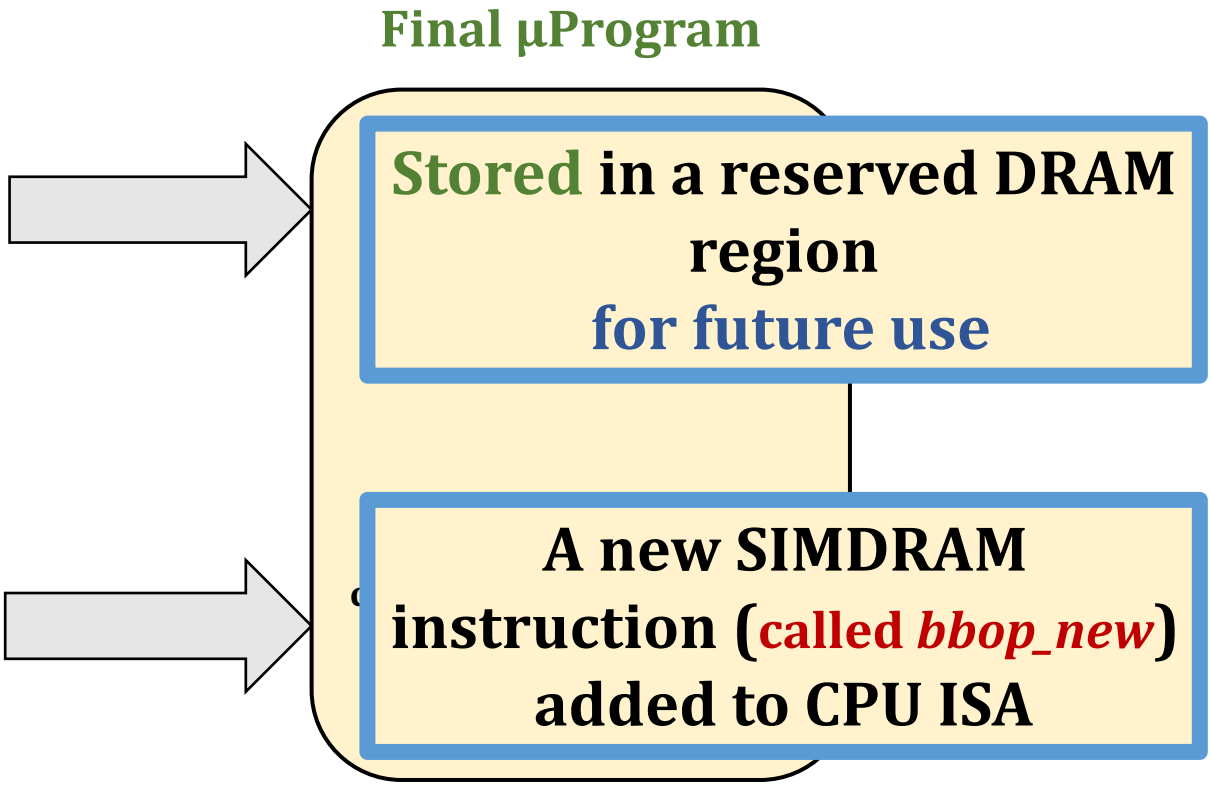
Task 2: Generate N-bit Computation

- **Final μProgram is optimized and computes the desired operation for operands of N-bit size in a bit-serial fashion**

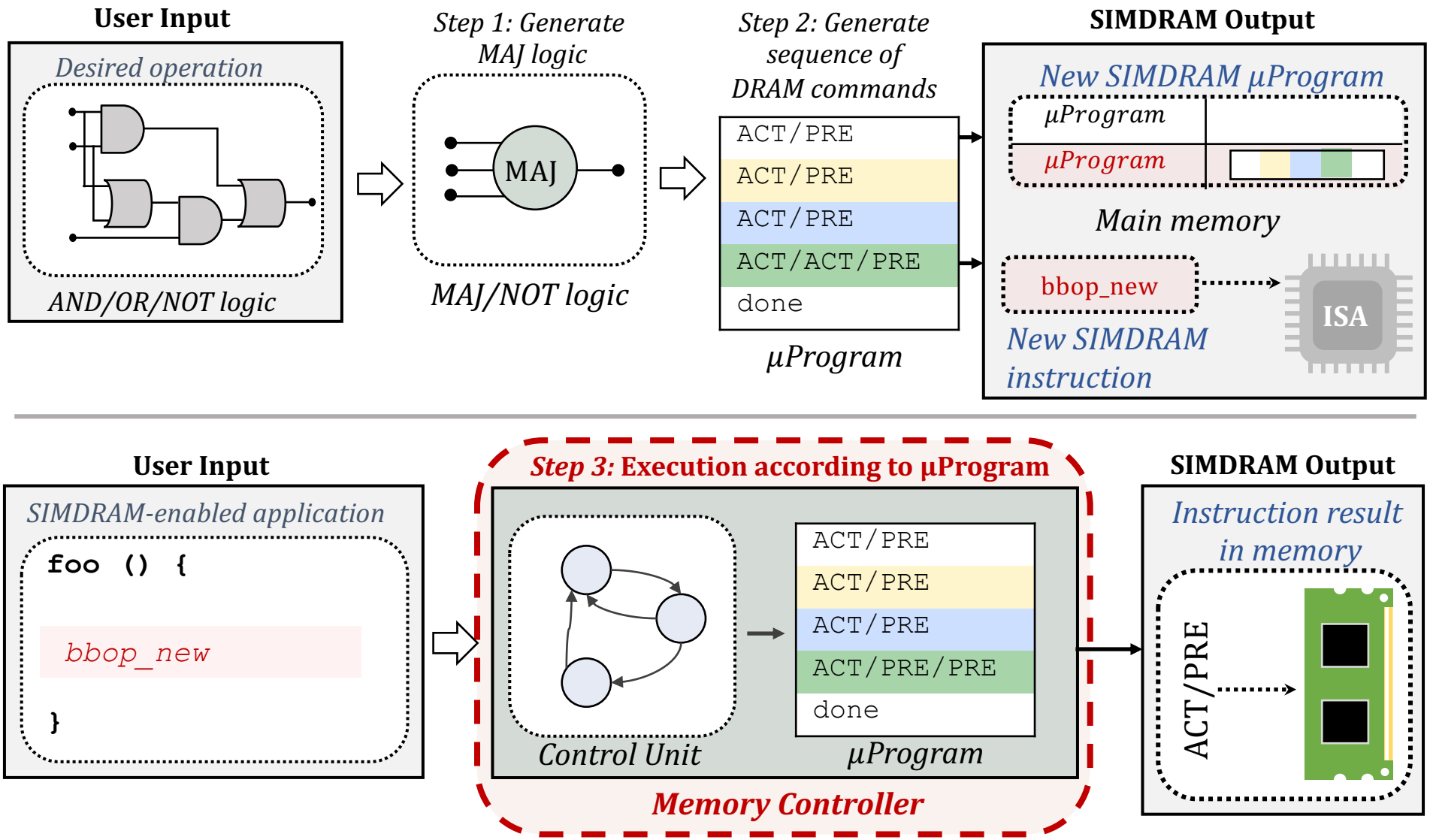


Task 2: Generate µProgram

- **Final µProgram is optimized and computes the desired operation for operands of N-bit size in a bit-serial fashion**

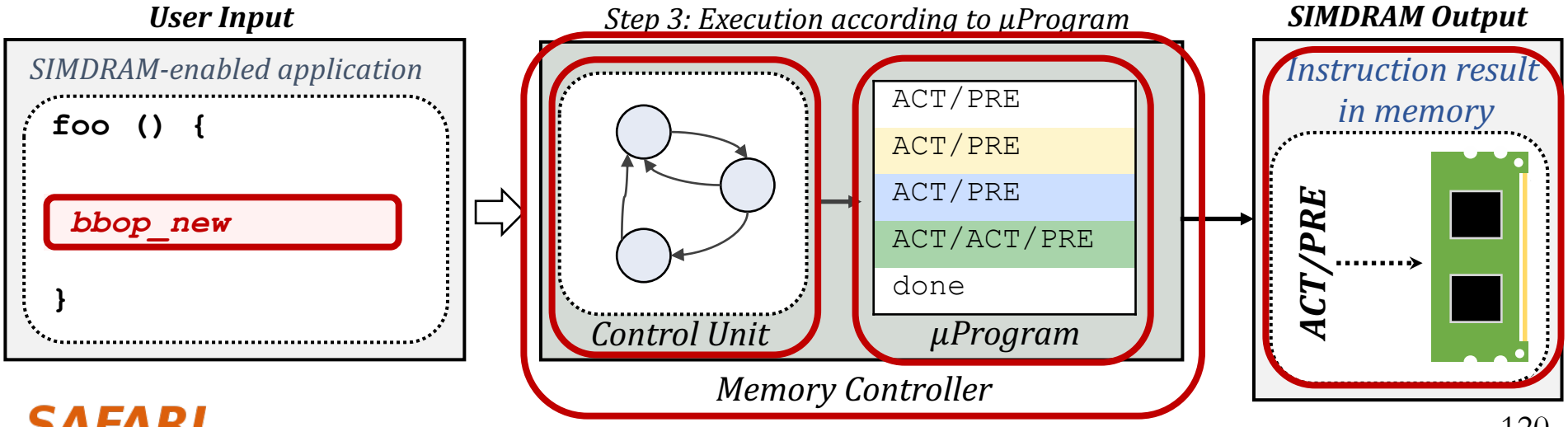


SIMDRAM Framework: Step 3



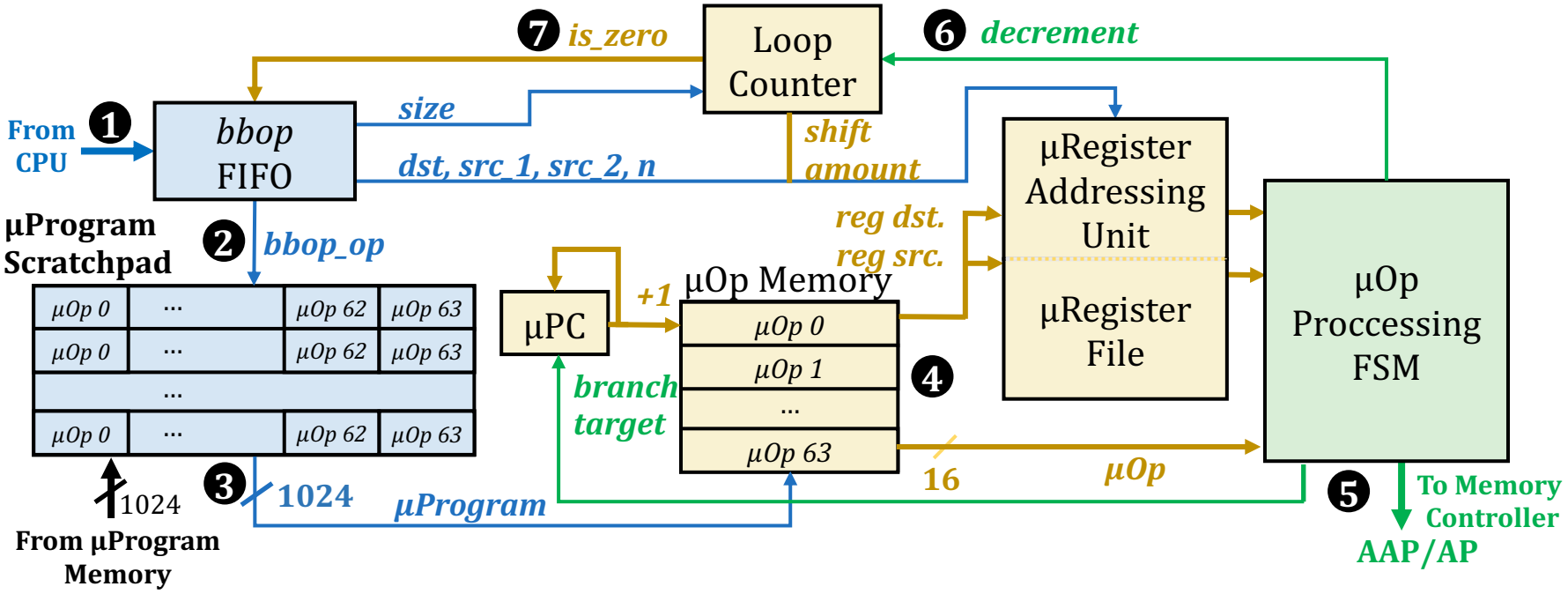
Step 3: µProgram Execution

- **SIMDRAM control unit:** handles the execution of the µProgram at runtime
- Upon receiving a **bbop instruction**, the control unit:
  1. Loads the µProgram corresponding to SIMDRAM operation
  2. Issues the sequence of DRAM commands (ACT/PRE) stored in the µProgram to SIMDRAM subarrays to perform the in-DRAM operation



More in the Paper

- Detailed reference implementation and microarchitecture of the SIMDRAM control unit



System Integration

Efficiently transposing data

Programming interface

Handling page faults, address translation, coherence, and interrupts

Handling limited subarray size

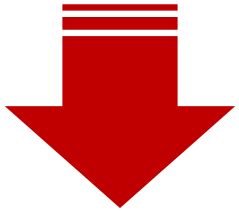
Security implications

Limitations of our framework

Transposing Data

- SIMDRAM operates on vertically-laid-out data

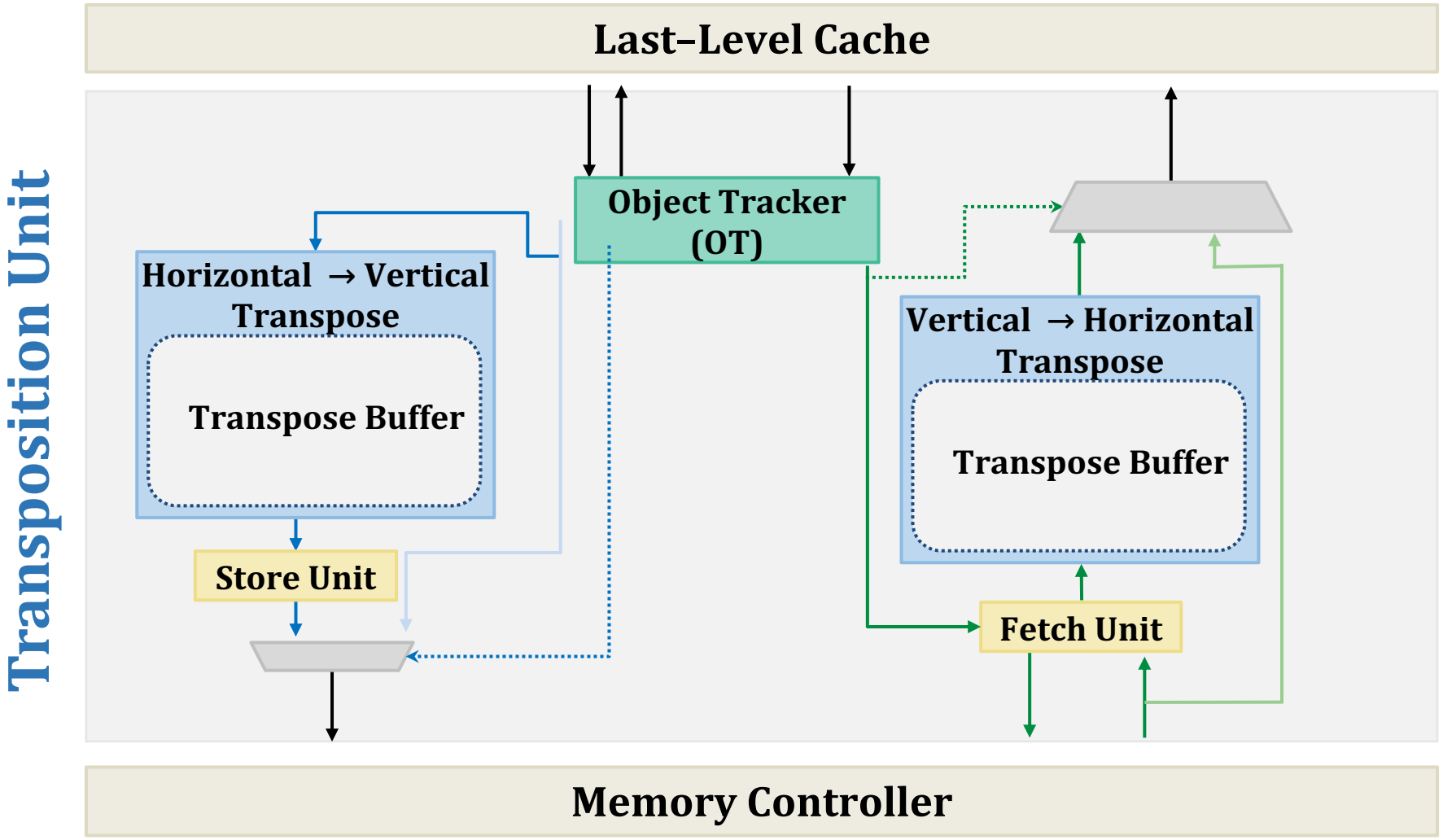
- Other system components expect data to be laid out horizontally



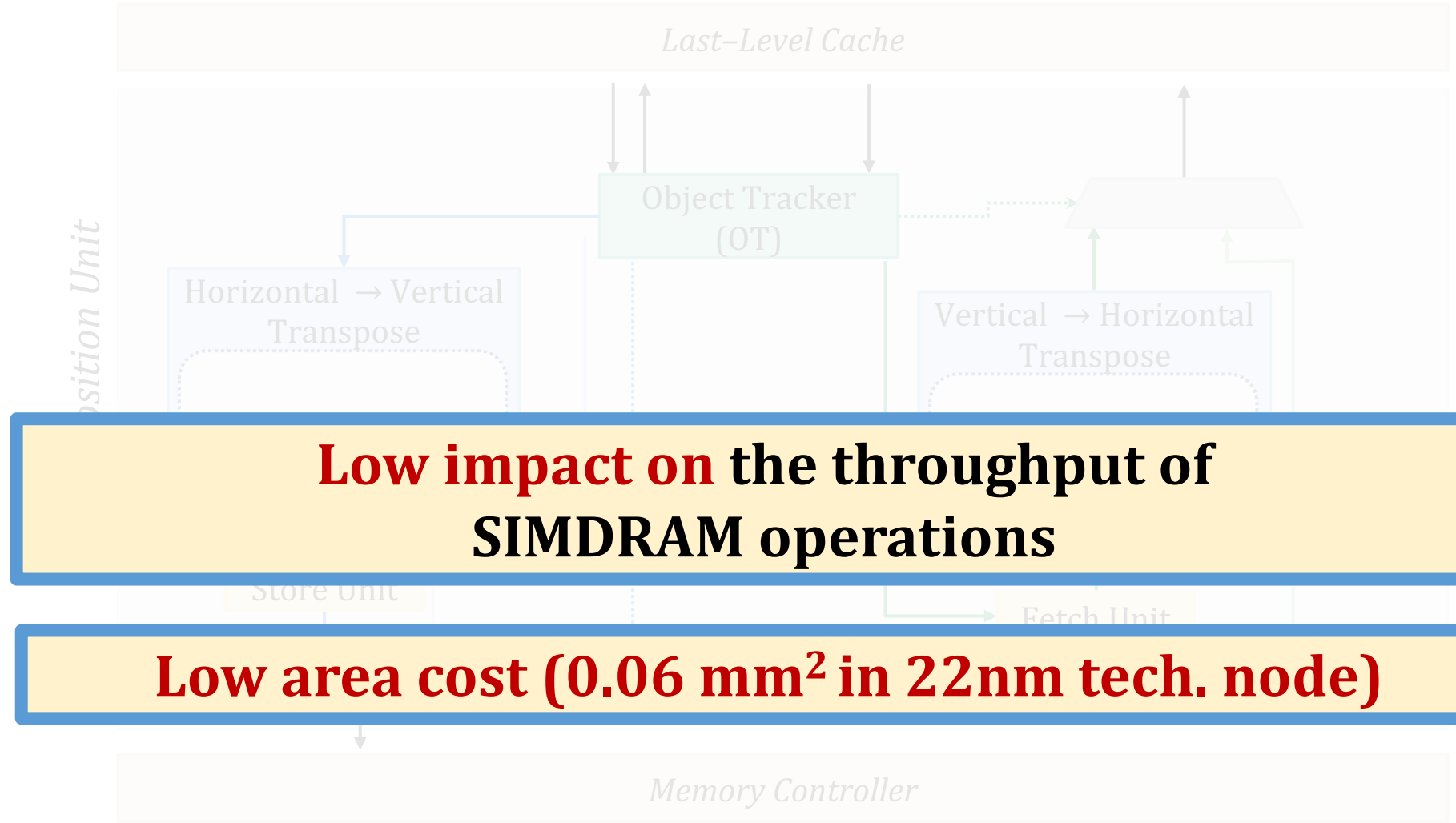
Challenging to share data between SIMDRAM and CPU

Transposition Unit

Transforms the data layout from horizontal to vertical, and vice versa



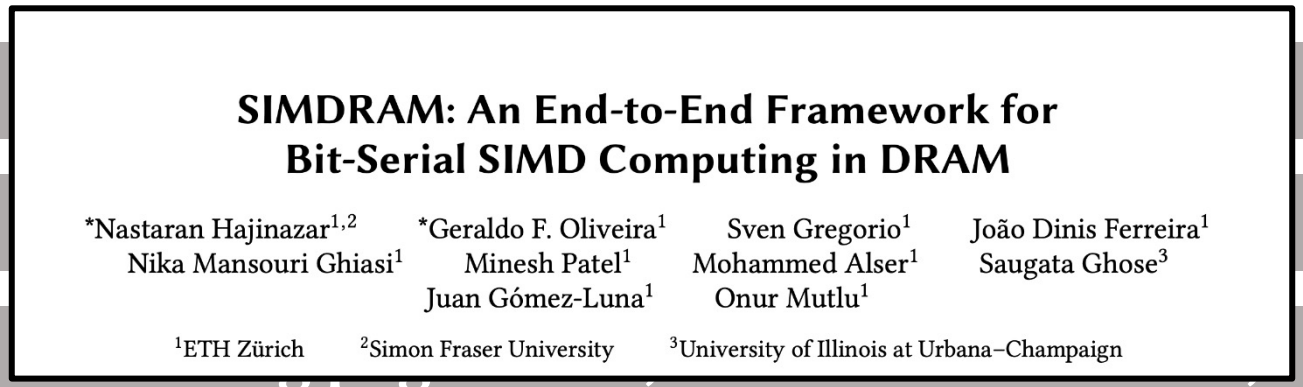
Efficiently Transposing Data



Low impact on the throughput of SIMDRAM operations

Low area cost (0.06 mm^2 in 22nm tech. node)

More in the Paper



SIMDRAM: An End-to-End Framework for Bit-Serial SIMD Computing in DRAM

*Nastaran Hajinazar[1,2] *Geraldo F. Oliveira[1] Sven Gregorio[1] João Dinis Ferreira[1]
Nika Mansouri Ghiasi[1] Minesh Patel[1] Mohammed Alser[1] Saugata Ghose[3]
Juan Gómez-Luna[1] Onur Mutlu[1]

[1]ETH Zürich [2]Simon Fraser University [3]University of Illinois at Urbana–Champaign

coherence, and interrupts

Handling limited subarray size

Security implications

Limitations of our framework

Methodology: Experimental Setup

- **Simulator:** gem5

- **Baselines:**
  - A multi-core CPU (Intel Skylake)
  - A high-end GPU (NVidia Titan V)
  - Ambit: a state-of-the-art in-memory computing mechanism

- **Evaluated SIMDRAM configurations** (all using a DDR4_2400_x64 device):
  - **1-bank:** SIMDRAM exploits 65'536 SIMD lanes (an 8 kB row buffer)
  - **4-banks:** SIMDRAM exploits 262'144 SIMD lanes
  - **16-banks:** SIMDRAM exploits 1'048'576 SIMD lanes

Methodology: Workloads

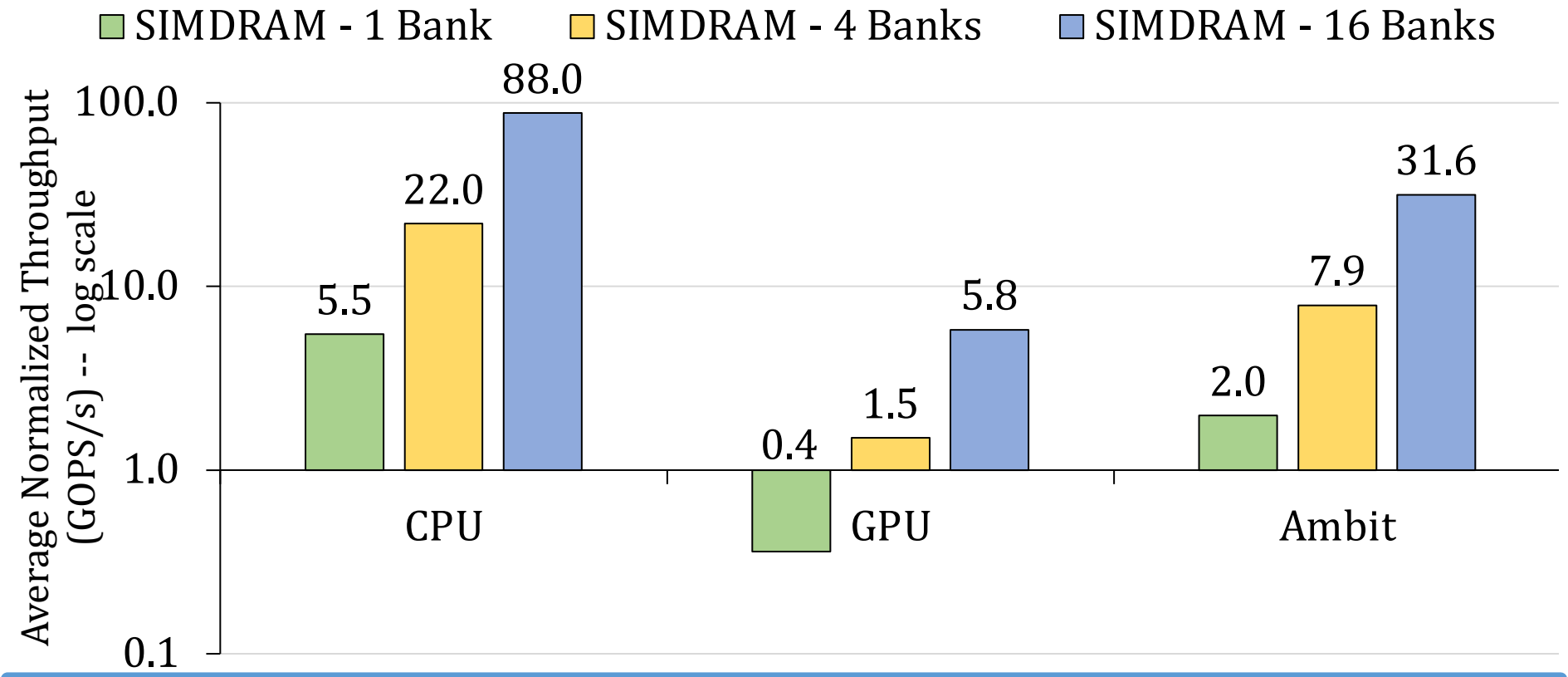
Evaluated:

- 16 complex in-DRAM operations:
  - Absolute
  - Addition/Subtraction
  - BitCount
  - Equality/ Greater/Greater Equal
  - Predication
  - ReLU
  - AND-/OR-/XOR-Reduction
  - Division/Multiplication

- 7 real-world applications
  - BitWeaving (databases)
  - TPH-H (databases)
  - kNN (machine learning)
  - LeNET (neural networks)
  - VGG-13/VGG-16 (neural networks)
  - Brightness (graphics)

Throughput Analysis

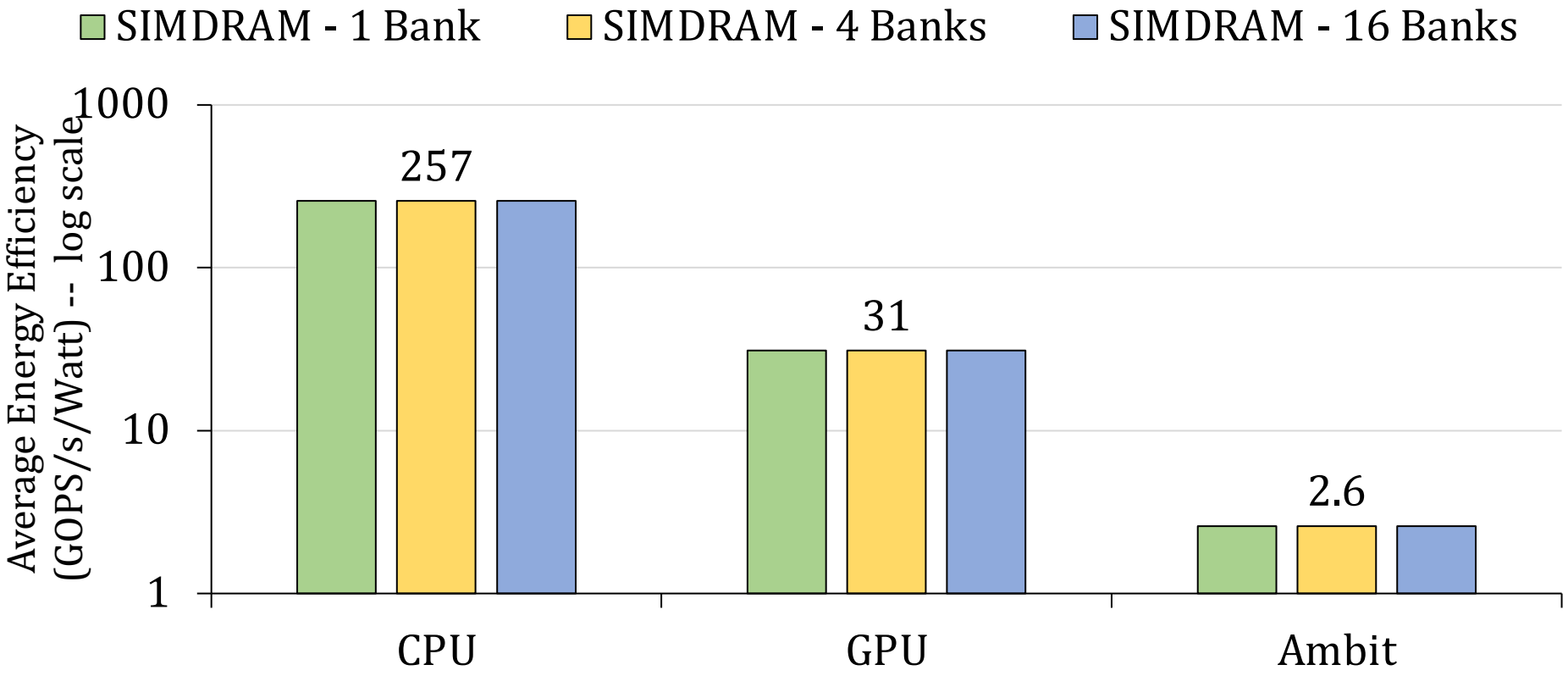
Average normalized throughput across all 16 SIMDRAM operations



SIMDRAM significantly outperforms all state-of-the-art baselines for a wide range of operations

Energy Analysis

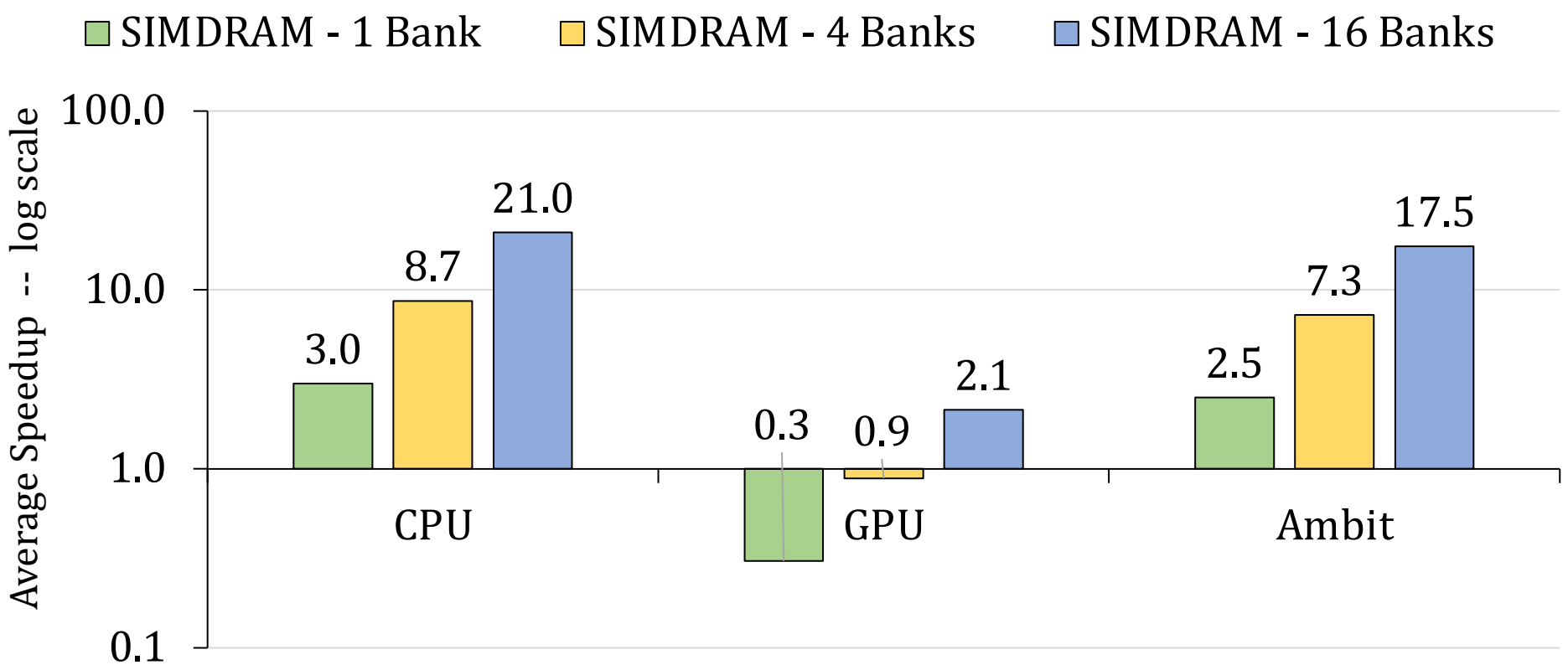
Average normalized energy efficiency across all 16 SIMDRAM operations



SIMDRAM is more energy-efficient than all state-of-the-art baselines for a wide range of operations

Real-World Applications

Average speedup across 7 real-world applications



SIMDRAM effectively and efficiently accelerates many commonly-used real-world applications

SIMDRAM Key Results

Evaluated on:

- 16 complex in-DRAM operations
- 7 commonly-used real-world applications

SIMDRAM provides:

- **88×** and **5.8×** the **throughput** of a **CPU** and a **high-end GPU**, respectively, over **16 operations**
- **257×** and **31×** the **energy efficiency** of a **CPU** and a **high-end GPU**, respectively, over **16 operations**
- **21×** and **2.1×** the **performance** of a **CPU** an a **high-end GPU**, over **seven real-world applications**

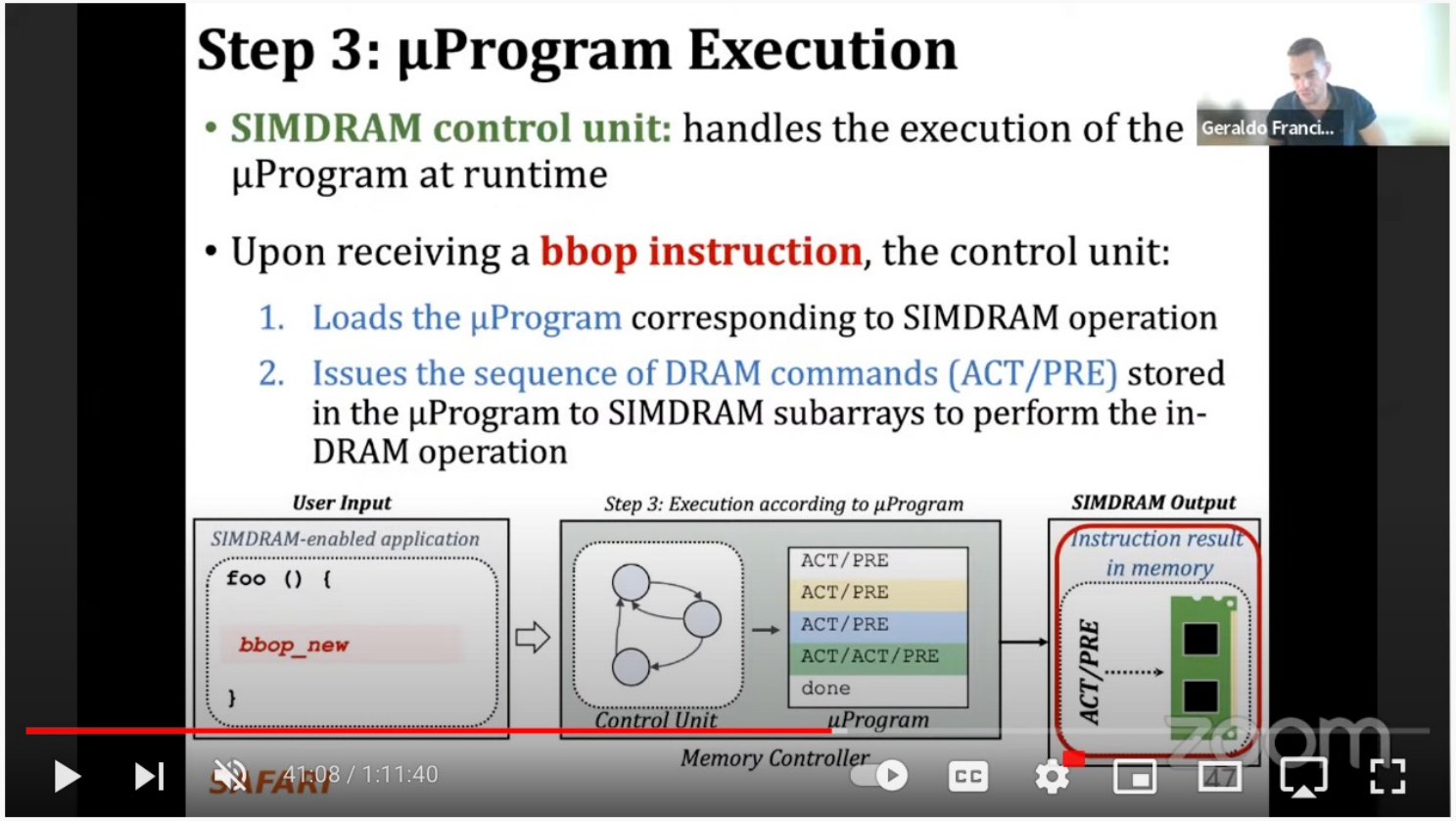
SIMDRAM Conclusion

- **SIMDRAM:**
  - Enables efficient computation of a flexible set and wide range of operations in a PuM massively parallel SIMD substrate
  - Provides the hardware, programming, and ISA support, to:
    - Address key system integration challenges
    - Allow programmers to define and employ new operations without hardware changes

SIMDRAM is a promising PuM framework

- Can **ease the adoption** of processing-using-DRAM architectures
- Improves the **performance** and **efficiency** of processing-using-memory architectures

Lecture on SIMDRAM



Processing-in-Memory Course: Lecture 13: Bit-Serial SIMD Processing using DRAM - Spring 2022

512 views • Streamed live on Jun 2, 2022 33 DISLIKE SHARE CLIP SAVE ...

Onur Mutlu Lectures
25.9K subscribers
SUBSCRIBED

https://youtu.be/pmZoAAhvkRQ

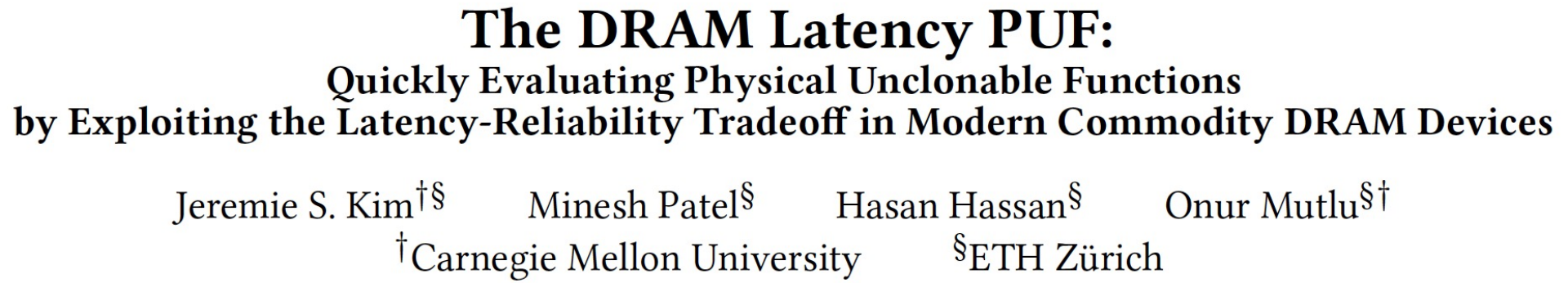
SIMDRAM: Follow-Ups

- Limitations of current substrate?
  - Computing granularity
  - Data layout conversion
  - High-latency bit-serial operations
  - Assembly-like programming model
  - Application scope
  - …

- We are working on even better processing-using-memory substrates
  - One step at a time!

In-DRAM Physical Unclonable Functions

- Jeremie S. Kim, Minesh Patel, Hasan Hassan, and Onur Mutlu,
  "The DRAM Latency PUF: Quickly Evaluating Physical Unclonable Functions by Exploiting the Latency-Reliability Tradeoff in Modern DRAM Devices"
  *Proceedings of the 24th International Symposium on High-Performance Computer Architecture (**HPCA**)*, Vienna, Austria, February 2018.
  [Lightning Talk Video]
  [Slides (pptx) (pdf)] [Lightning Session Slides (pptx) (pdf)]
  [Full Talk Lecture Video (28 minutes)]

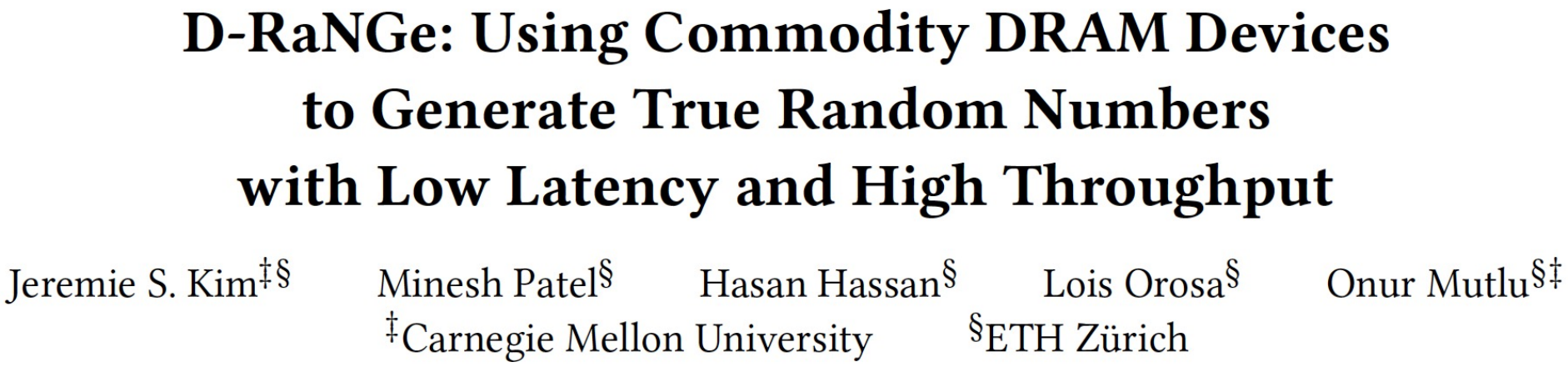


The DRAM Latency PUF:
Quickly Evaluating Physical Unclonable Functions by Exploiting the Latency-Reliability Tradeoff in Modern Commodity DRAM Devices

Jeremie S. Kim[†§] Minesh Patel[§] Hasan Hassan[§] Onur Mutlu[§†]
[†]Carnegie Mellon University [§]ETH Zürich

In-DRAM True Random Number Generation

- Jeremie S. Kim, Minesh Patel, Hasan Hassan, Lois Orosa, and Onur Mutlu,
  "D-RaNGe: Using Commodity DRAM Devices to Generate True Random Numbers with Low Latency and High Throughput"
  *Proceedings of the 25th International Symposium on High-Performance Computer Architecture (**HPCA**)*, Washington, DC, USA, February 2019.
  [Slides (pptx) (pdf)]
  [Full Talk Video (21 minutes)]
  [Full Talk Lecture Video (27 minutes)]
  Top Picks Honorable Mention by IEEE Micro.



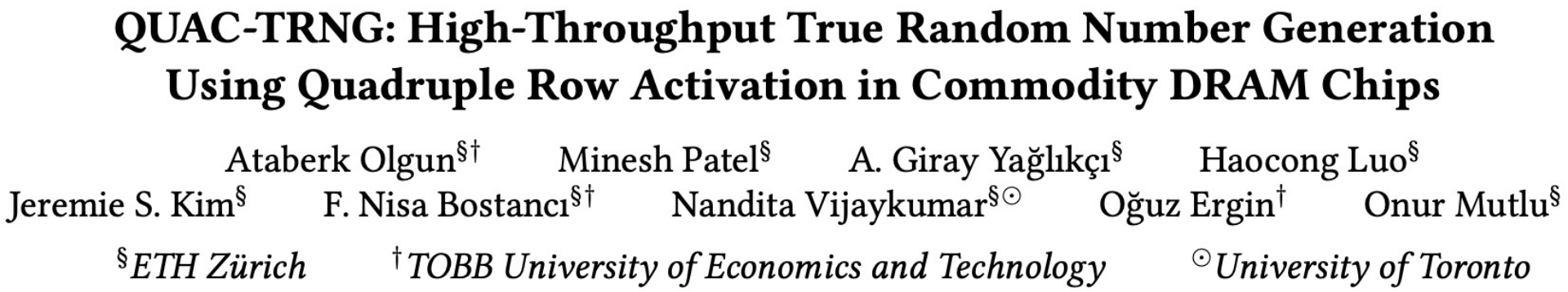
D-RaNGe: Using Commodity DRAM Devices to Generate True Random Numbers with Low Latency and High Throughput

Jeremie S. Kim[‡§] Minesh Patel[§] Hasan Hassan[§] Lois Orosa[§] Onur Mutlu[§‡]

[‡]Carnegie Mellon University [§]ETH Zürich

In-DRAM True Random Number Generation

- Ataberk Olgun, Minesh Patel, A. Giray Yaglikci, Haocong Luo, Jeremie S. Kim, F. Nisa Bostanci, Nandita Vijaykumar, Oguz Ergin, and Onur Mutlu,
  "QUAC-TRNG: High-Throughput True Random Number Generation Using Quadruple Row Activation in Commodity DRAM Chips"
  *Proceedings of the 48th International Symposium on Computer Architecture (**ISCA**)*, Virtual, June 2021.
  [Slides (pptx) (pdf)]
  [Short Talk Slides (pptx) (pdf)]
  [Talk Video (25 minutes)]
  [SAFARI Live Seminar Video (1 hr 26 mins)]



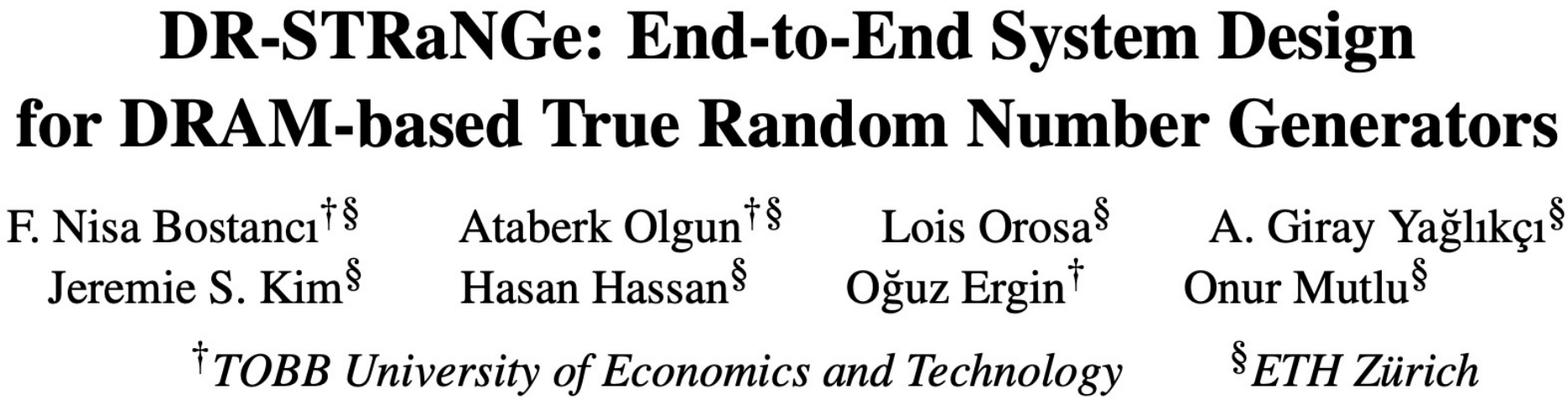
QUAC-TRNG: High-Throughput True Random Number Generation Using Quadruple Row Activation in Commodity DRAM Chips

Ataberk Olgun[§†] Minesh Patel[§] A. Giray Yağlıkçı[§] Haocong Luo[§]
Jeremie S. Kim[§] F. Nisa Bostancı[§†] Nandita Vijaykumar[§⊙] Oğuz Ergin[†] Onur Mutlu[§]

[§]*ETH Zürich* [†]*TOBB University of Economics and Technology* [⊙]*University of Toronto*

In-DRAM True Random Number Generation

- F. Nisa Bostanci, Ataberk Olgun, Lois Orosa, A. Giray Yaglikci, Jeremie S. Kim, Hasan Hassan, Oguz Ergin, and Onur Mutlu,
  "DR-STRaNGe: End-to-End System Design for DRAM-based True Random Number Generators"
  *Proceedings of the 28th International Symposium on High-Performance Computer Architecture (**HPCA**)*, Virtual, April 2022.
  [Slides (pptx) (pdf)]
  [Short Talk Slides (pptx) (pdf)]



DR-STRaNGe: End-to-End System Design for DRAM-based True Random Number Generators

F. Nisa Bostancı[†§] Ataberk Olgun[†§] Lois Orosa[§] A. Giray Yağlıkçı[§]
Jeremie S. Kim[§] Hasan Hassan[§] Oğuz Ergin[†] Onur Mutlu[§]

[†]*TOBB University of Economics and Technology* [§]*ETH Zürich*

https://arxiv.org/pdf/2201.01385.pdf

In-DRAM Lookup-Table Based Execution

João Dinis Ferreira, Gabriel Falcao, Juan Gómez-Luna, Mohammed Alser, Lois Orosa, Mohammad Sadrosadati, Jeremie S. Kim, Geraldo F. Oliveira, Taha Shahroodi, Anant Nori, and Onur Mutlu,
"pLUTo: Enabling Massively Parallel Computation in DRAM via Lookup Tables"
Proceedings of the 55th International Symposium on Microarchitecture (***MICRO***), Chicago, IL, USA, October 2022.
[Slides (pptx) (pdf)]
[Longer Lecture Slides (pptx) (pdf)]
[Lecture Video (26 minutes)]
[arXiv version]
[Source Code (Officially Artifact Evaluated with All Badges)]
Officially artifact evaluated as available, reusable and reproducible.

pLUTo: Enabling Massively Parallel Computation in DRAM via Lookup Tables

João Dinis Ferreira[§] Gabriel Falcao[†] Juan Gómez-Luna[§] Mohammed Alser[§]
Lois Orosa[§▽] Mohammad Sadrosadati[§] Jeremie S. Kim[§] Geraldo F. Oliveira[§]
Taha Shahroodi[‡] Anant Nori[⋆] Onur Mutlu[§]
[§]*ETH Zürich* [†]*IT, University of Coimbra* [▽]*Galicia Supercomputing Center* [‡]*TU Delft* [⋆]*Intel*

https://arxiv.org/pdf/2104.07699.pdf

Extending Operation Support with Lookup Tables

Limitations of Processing-using-DRAM

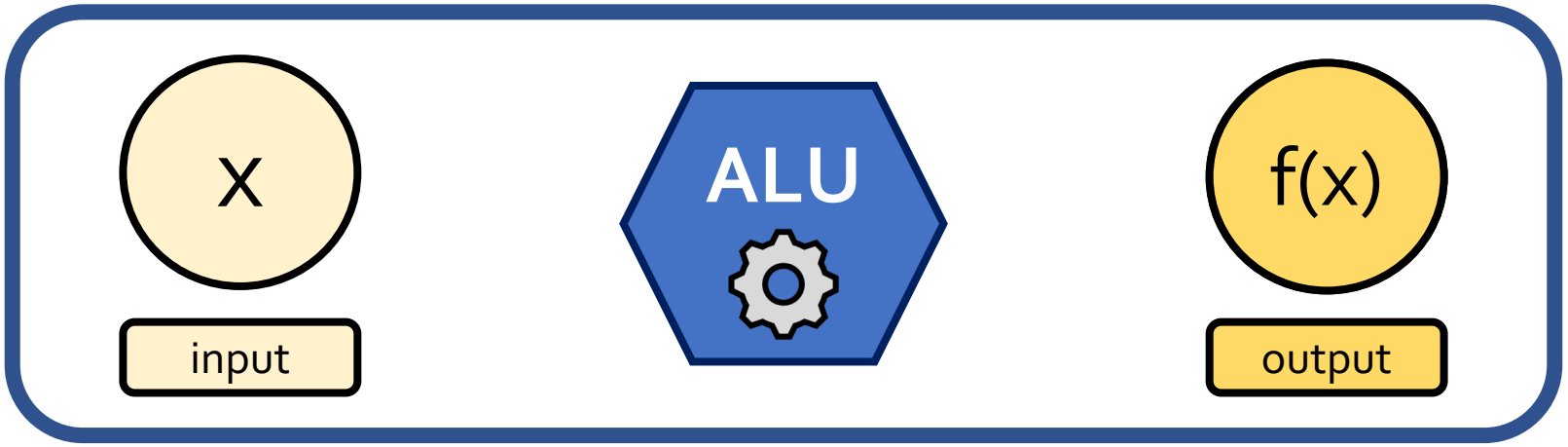
Data Movement	*RowClone, Seshadri+ 2013* *LISA, Chang+ 2013*
Bitwise Operations	*Ambit, Seshadri+ 2017*
Bit Shifting	*DRISA, Li+ 2017*
Arithmetic Operations	*SIMDRAM, Hajinazar & Oliveira+ 2021*

Existing Processing-using-DRAM architectures only support a limited range of operations

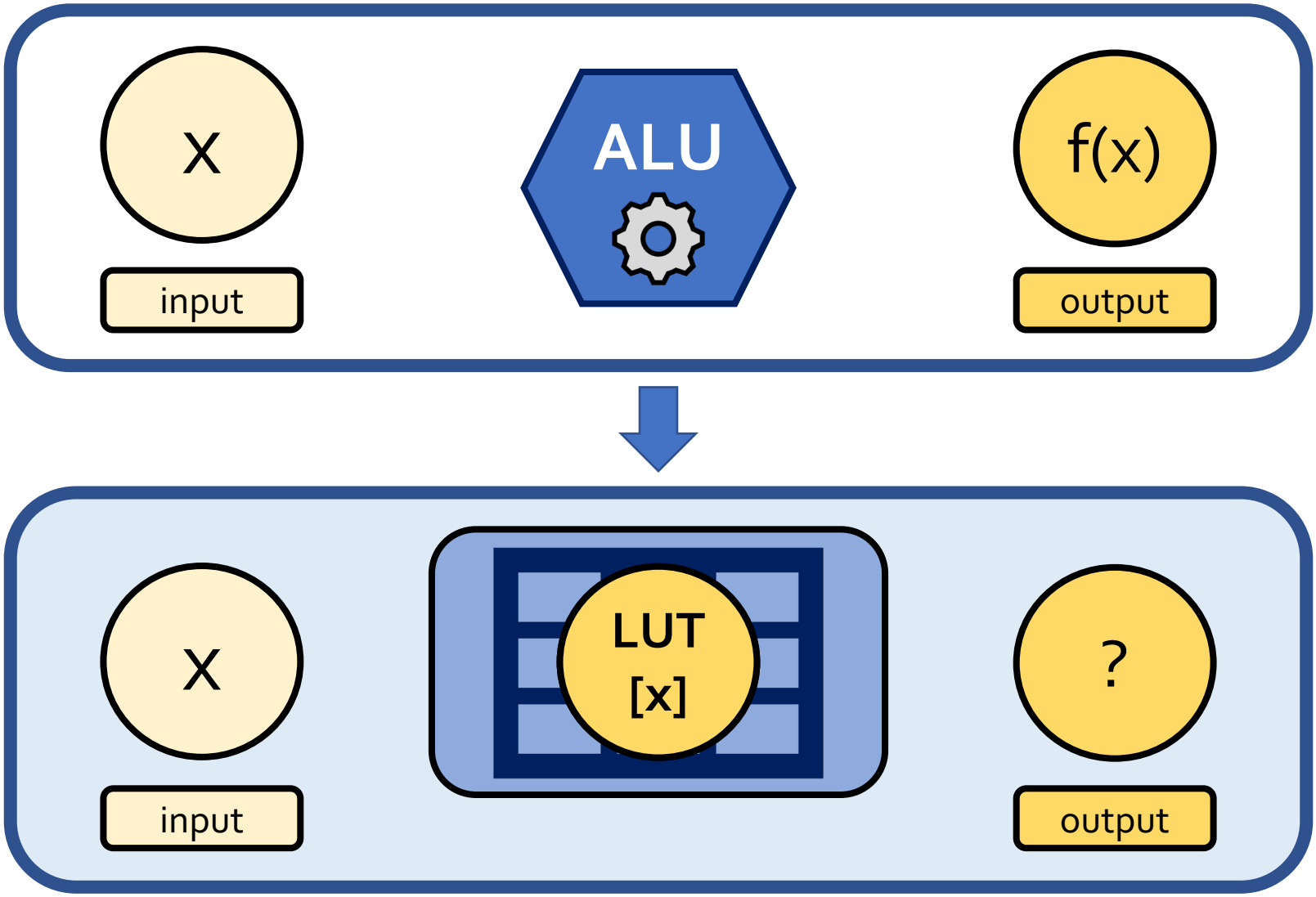
The Goal of pLUTo

Extend **Processing-using-DRAM to support the execution of** ***arbitrarily complex operations***

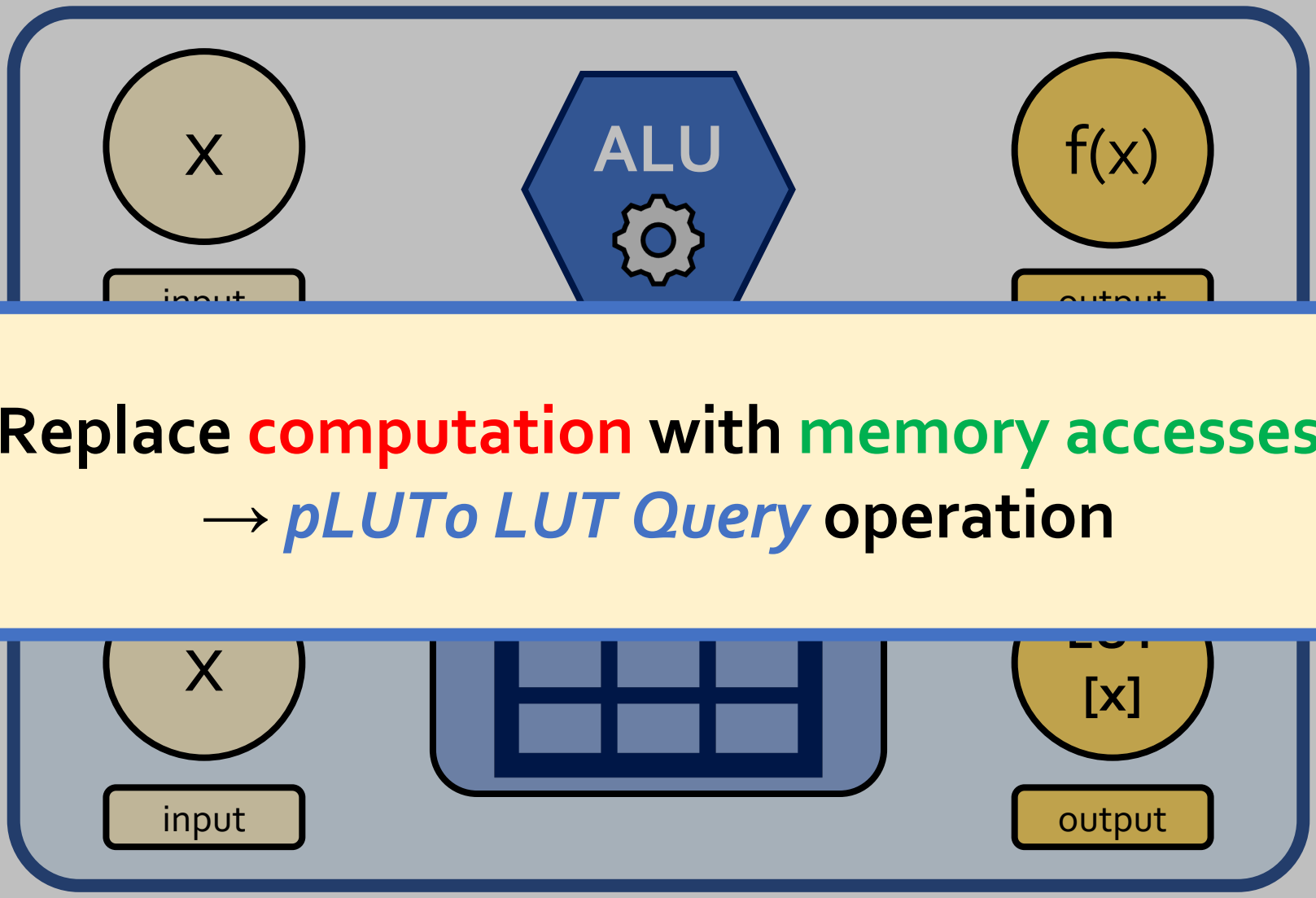
pLUTo: Key Idea



pLUTo: Key Idea



pLUTo: Key Idea



In-DRAM pLUTo LUT Query: Setup

LUT Query:
Return {2nd, 1st, 2nd, 4th}
prime numbers

LUT index	Prime numbers: i	Prime numbers: f(i)
0	1st	2
1	2nd	3
2	3rd	5
3	4th	7

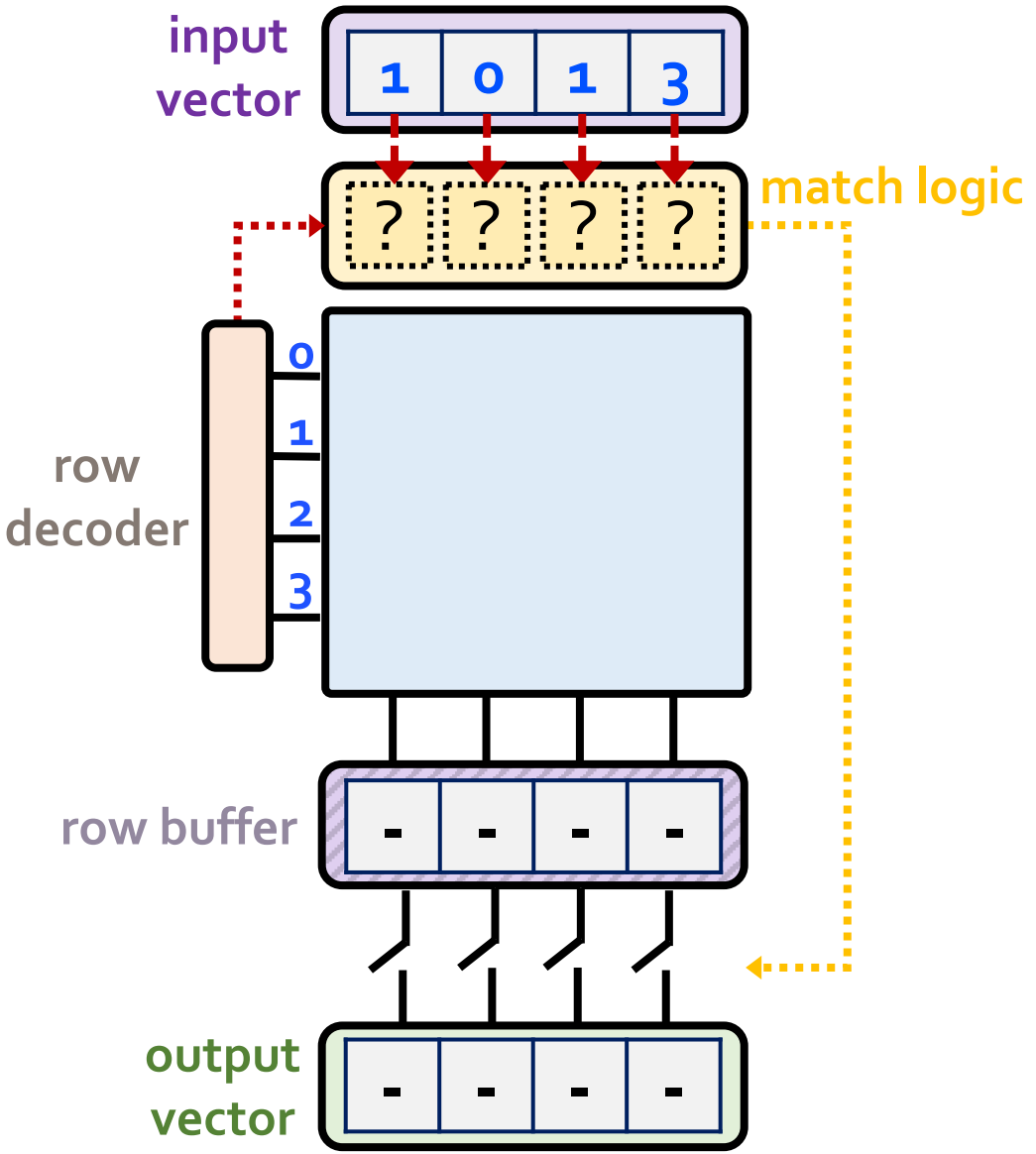
lookup table

1	0	1	3

input vector

3	2	3	7

output vector



In-DRAM pLUTo LUT Query: Setup

LUT Query:
Return {2nd, 1st, 2nd, 4th}
prime numbers

LUT index	Prime numbers: i	Prime numbers: f(i)
0	1st	2
1	2nd	3
2	3rd	5
3	4th	7

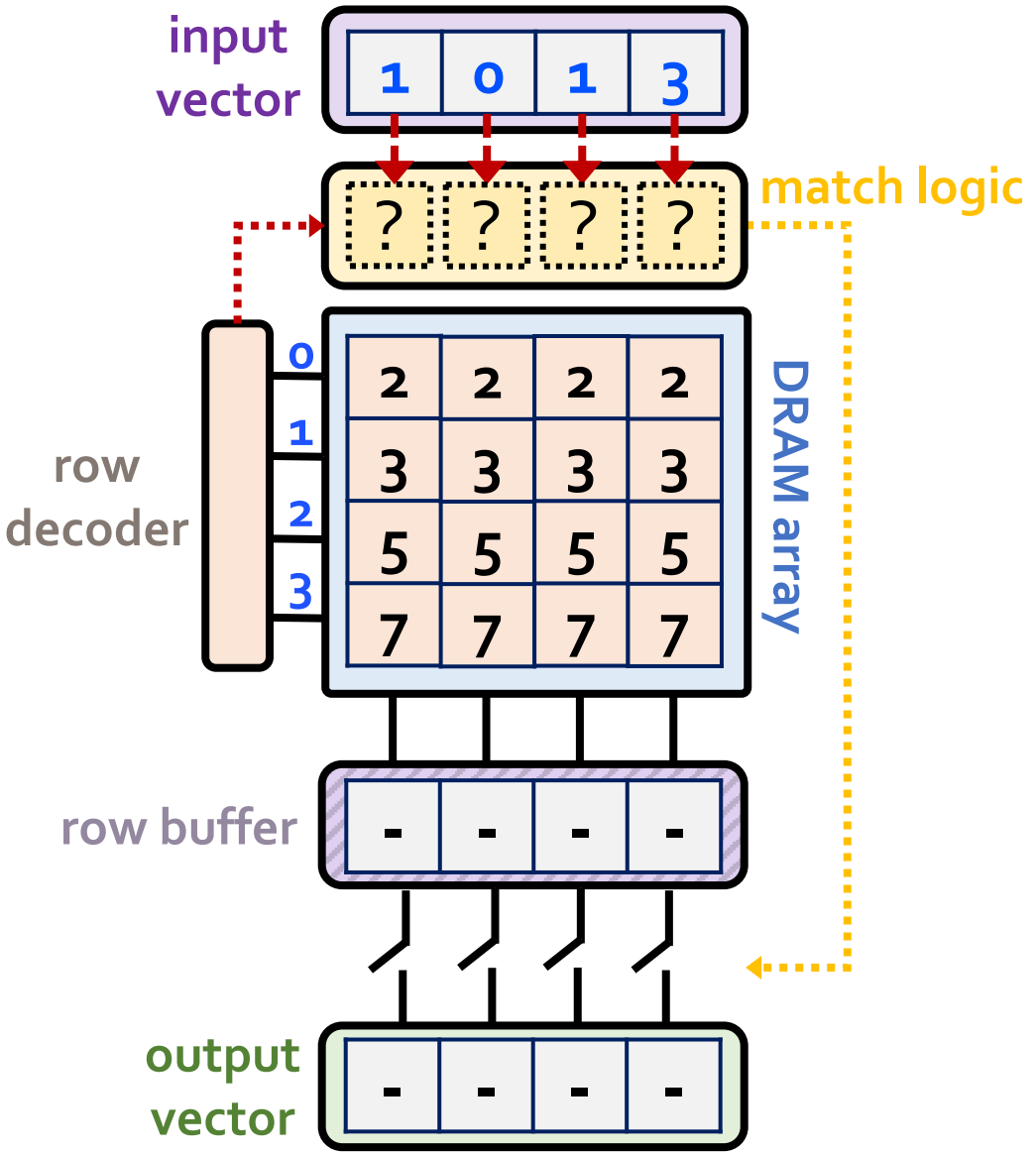
lookup table

1	0	1	3

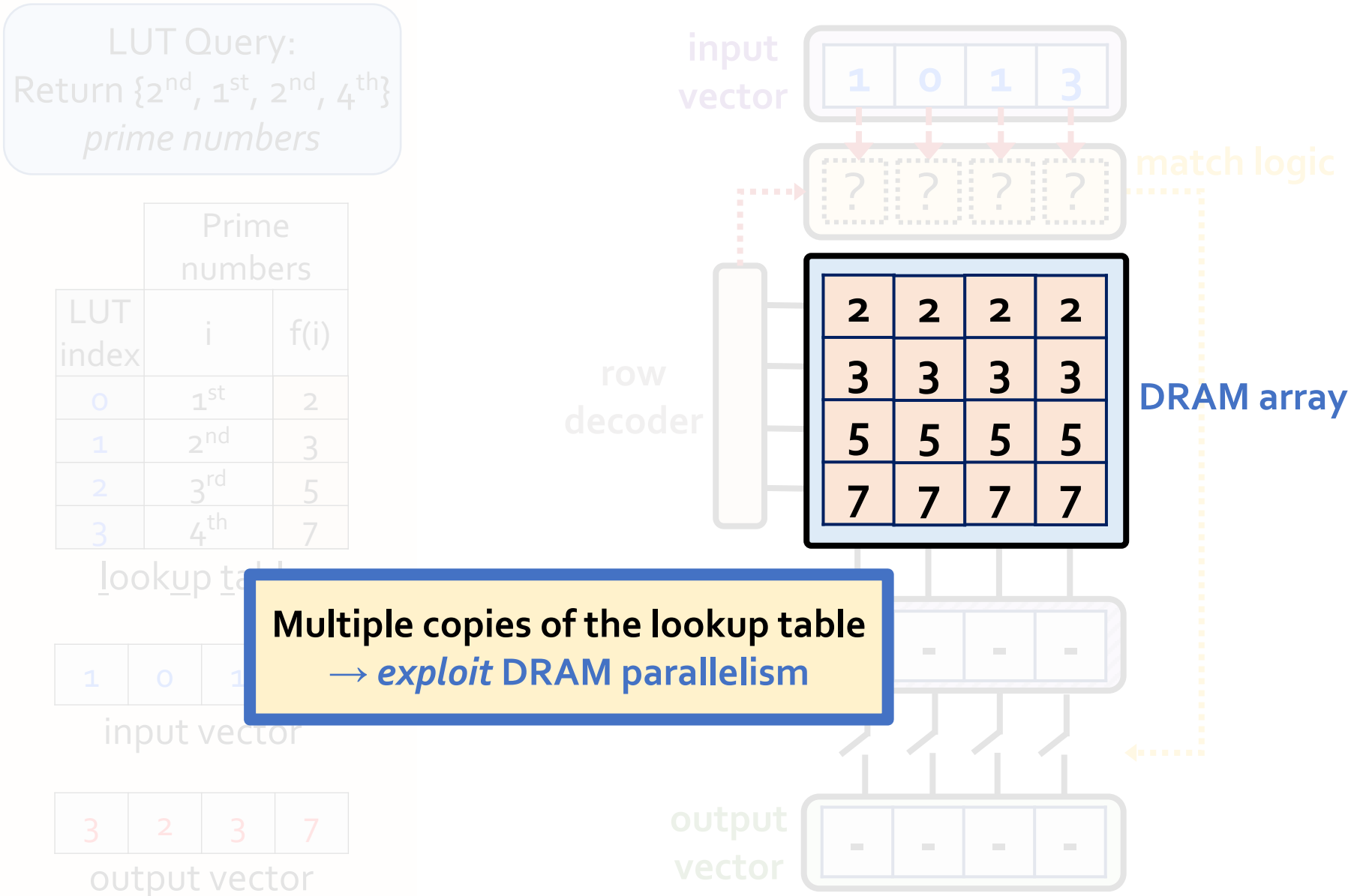
input vector

3	2	3	7

output vector



In-DRAM pLUTo LUT Query: Setup



In-DRAM pLUTo LUT Query: Setup

LUT Query:
Return {2nd, 1st, 2nd, 4th}
prime numbers

LUT index	Prime numbers: i	Prime numbers: f(i)
0	1st	2
1	2nd	3
2	3rd	5
3	4th	7

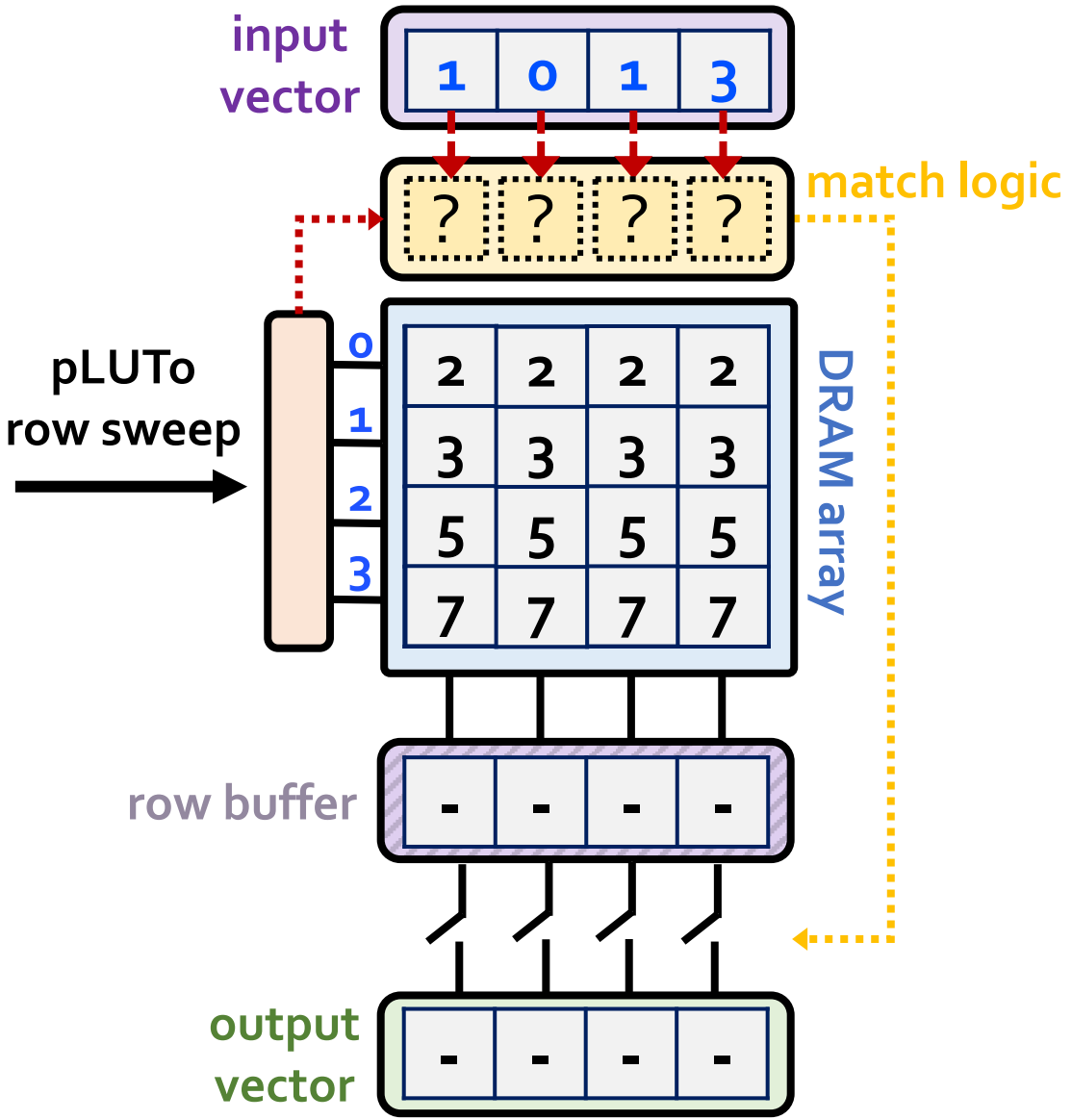
lookup table

1	0	1	3

input vector

3	2	3	7

output vector



In-DRAM pLUTo LUT Query: Step 1

LUT Query:
Return {2nd, 1st, 2nd, 4th}
prime numbers

LUT index	Prime numbers: i	Prime numbers: f(i)
0	1st	2
1	2nd	3
2	3rd	5
3	4th	7

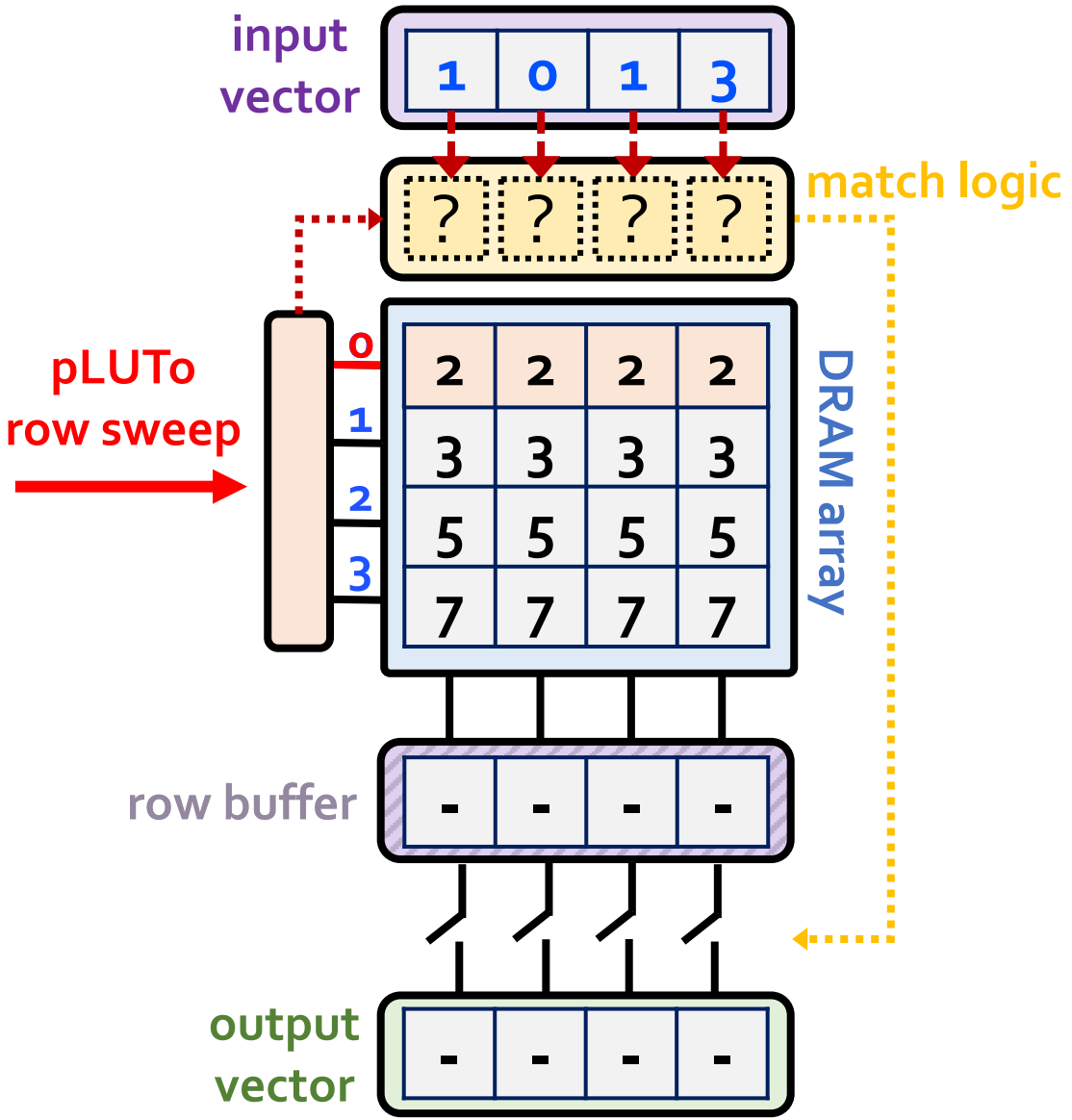
lookup table

1	0	1	3

input vector

3	2	3	7

output vector



In-DRAM pLUTo LUT Query: Step 1

LUT Query:
Return {2nd, 1st, 2nd, 4th}
prime numbers

LUT index	Prime numbers: i	Prime numbers: f(i)
0	1st	2
1	2nd	3
2	3rd	5
3	4th	7

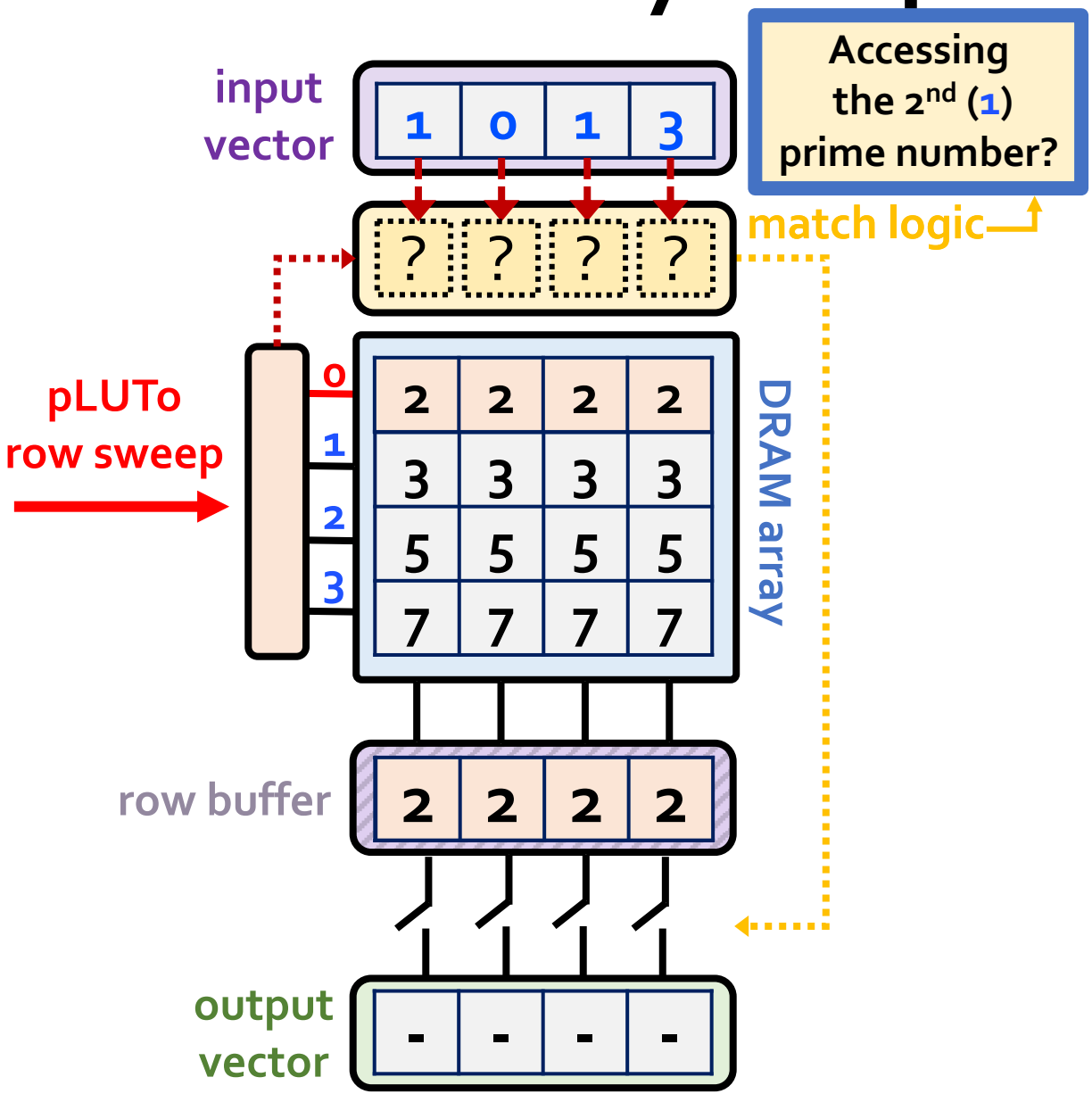
lookup table

1	0	1	3

input vector

3	2	3	7

output vector



In-DRAM pLUTo LUT Query: Step 1

LUT Query:
Return {2nd, 1st, 2nd, 4th}
prime numbers

LUT index	Prime numbers i	f(i)
0	1st	2
1	2nd	3
2	3rd	5
3	4th	7

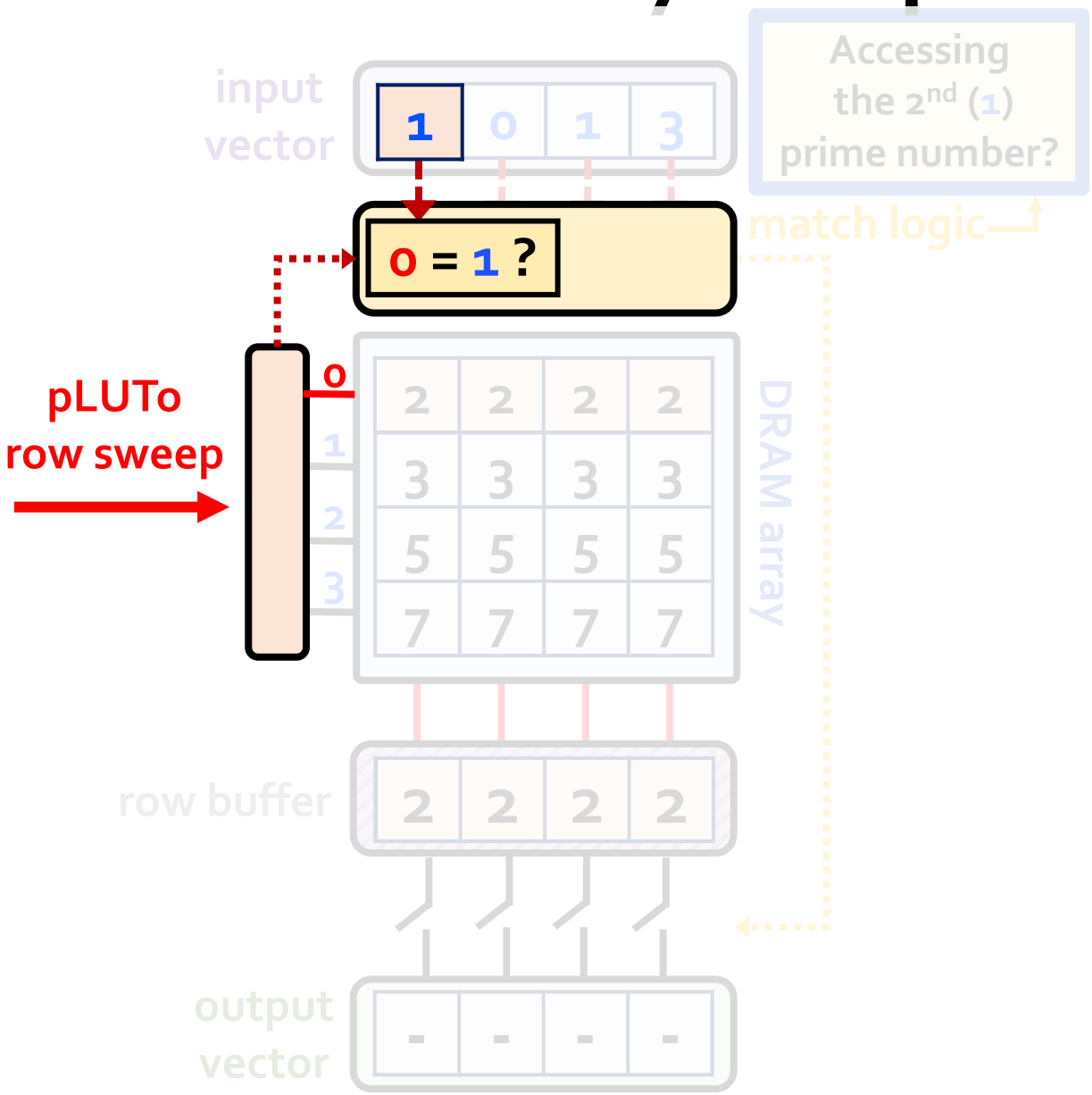
lookup table

1	0	1	3

input vector

3	2	3	7

output vector



In-DRAM pLUTo LUT Query: Step 1

LUT Query:
Return {2nd, 1st, 2nd, 4th}
prime numbers

LUT index	Prime numbers i	f(i)
0	1st	2
1	2nd	3
2	3rd	5
3	4th	7

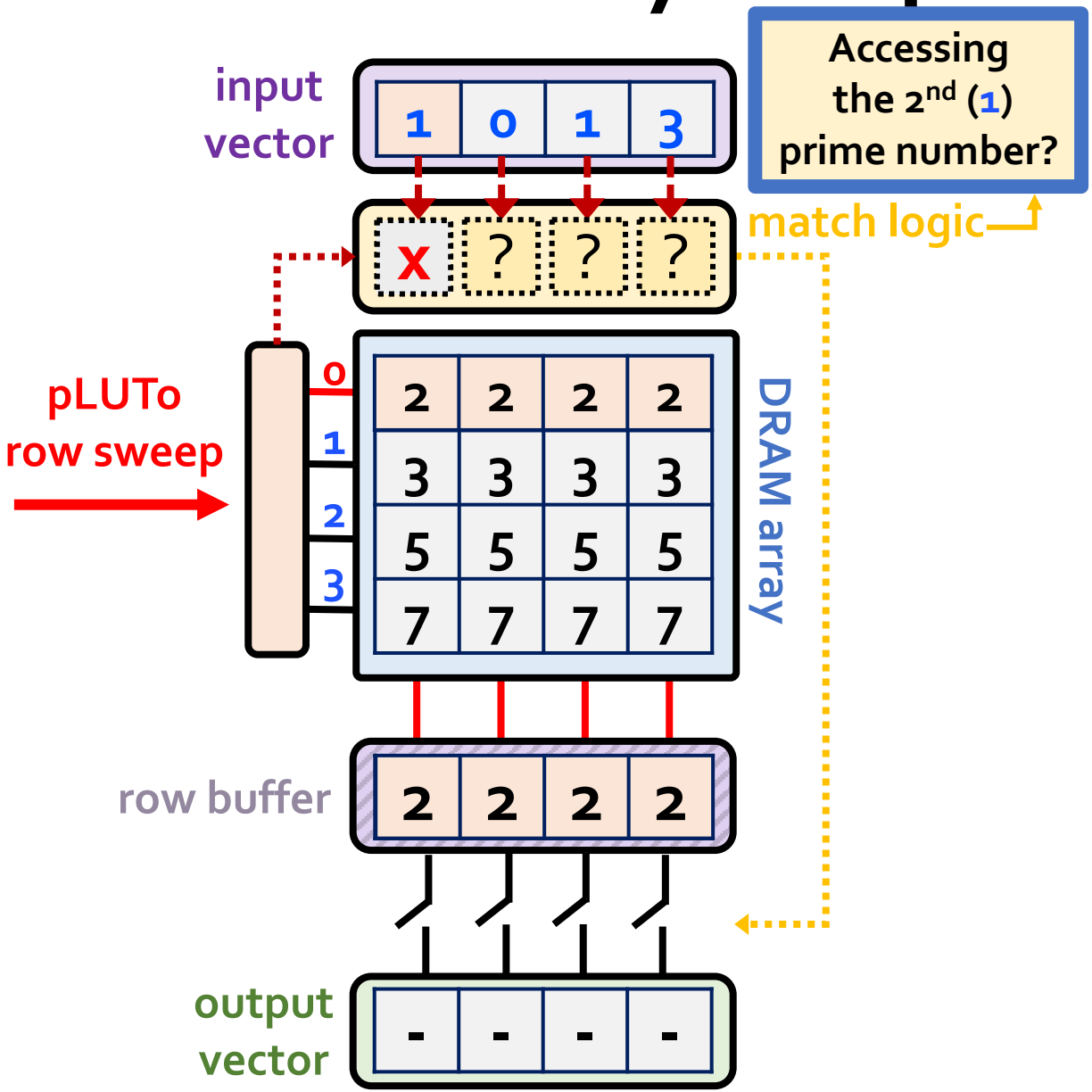
lookup table

1	0	1	3

input vector

3	2	3	7

output vector



In-DRAM pLUTo LUT Query: Step 1

LUT Query:
Return {2nd, 1st, 2nd, 4th}
prime numbers

LUT index	Prime numbers i	f(i)
0	1st	2
1	2nd	3
2	3rd	5
3	4th	7

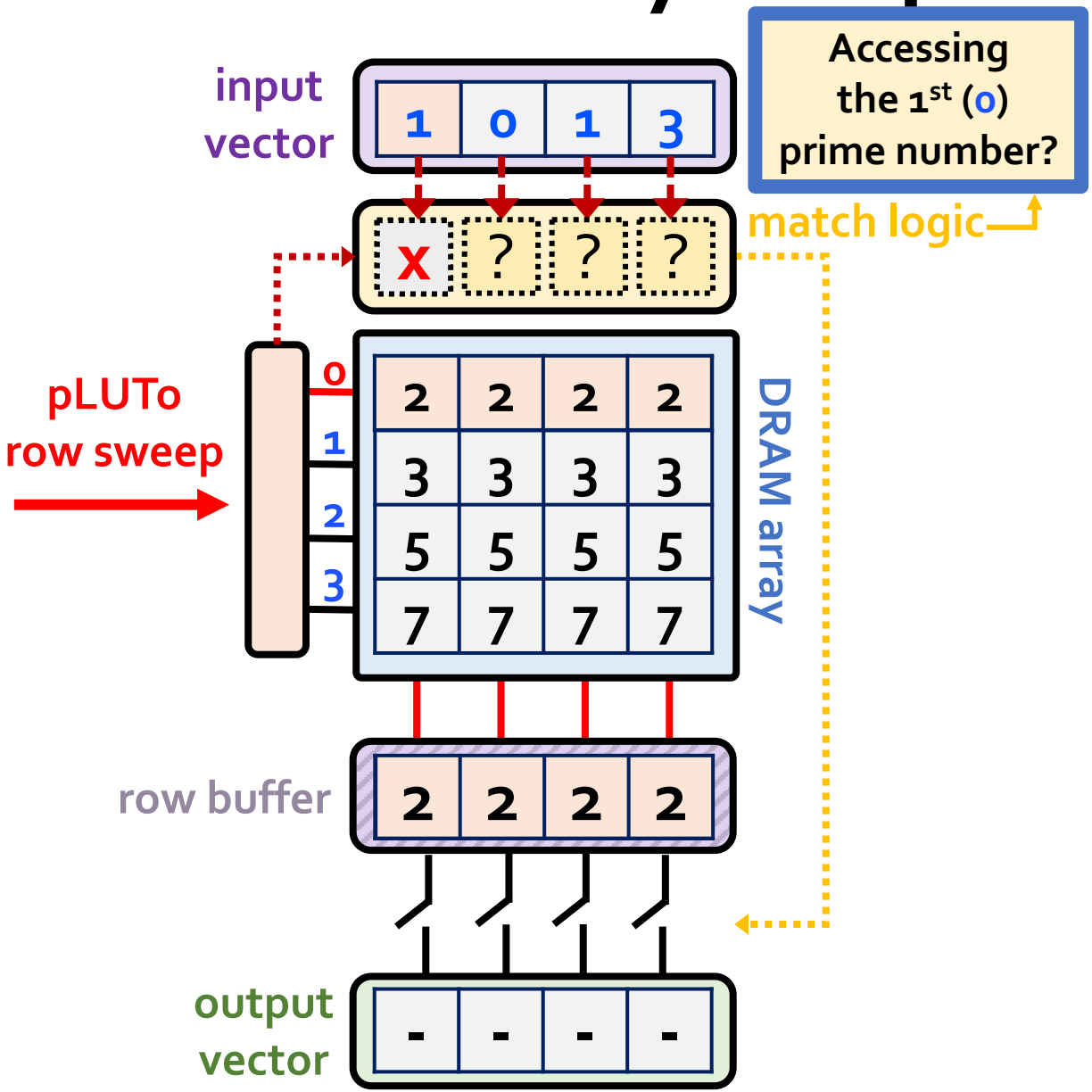
lookup table

1	0	1	3

input vector

3	2	3	7

output vector



In-DRAM pLUTo LUT Query: Step 1

LUT Query:
Return {2nd, 1st, 2nd, 4th} *prime numbers*

LUT index	i	f(i)
0	1st	2
1	2nd	3
2	3rd	5
3	4th	7

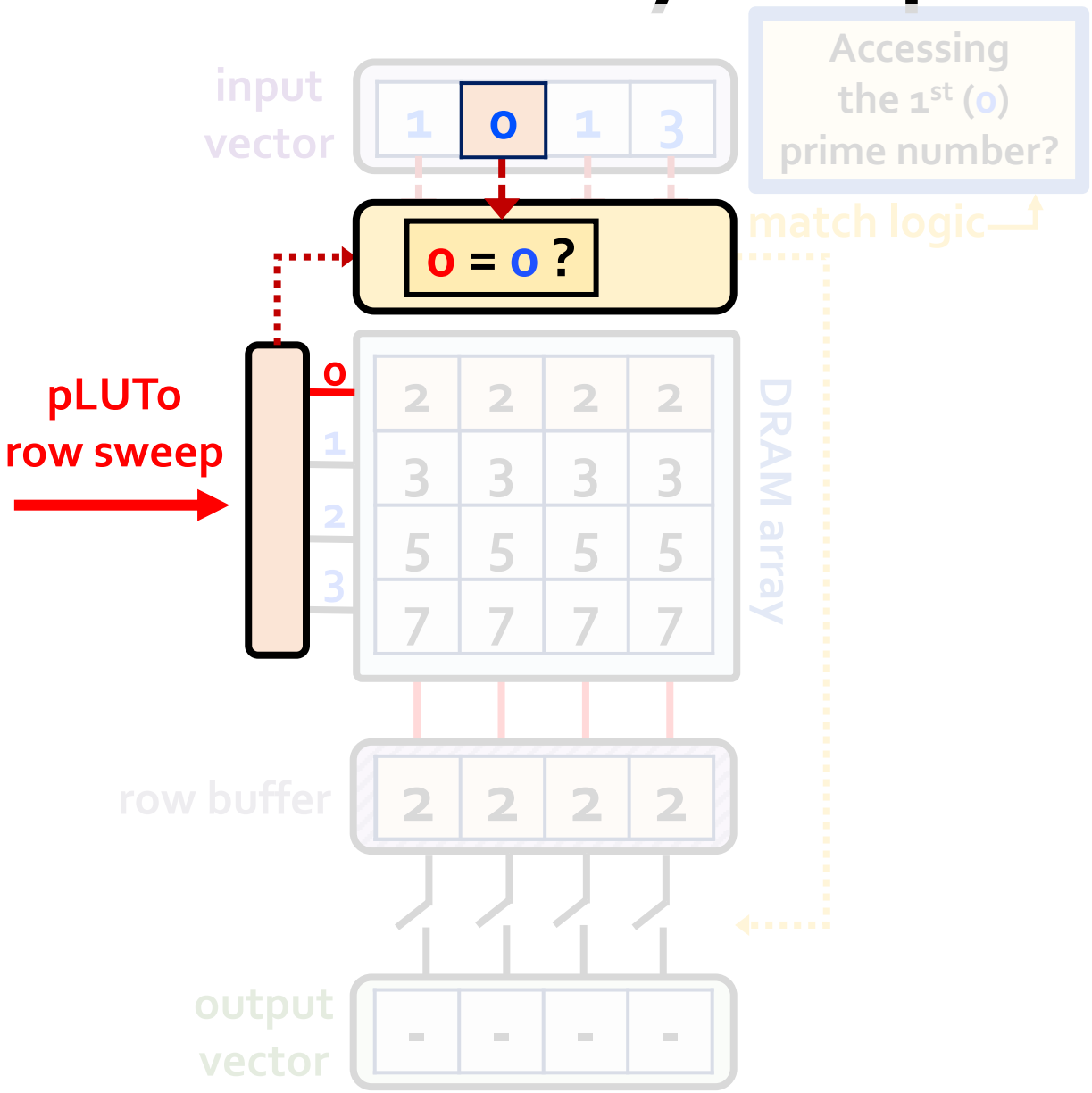
lookup table

1	0	1	3

input vector

3	2	3	7

output vector



In-DRAM pLUTo LUT Query: Step 1

LUT Query:
Return {2nd, 1st, 2nd, 4th}
prime numbers

LUT index	Prime numbers i	f(i)
0	1st	2
1	2nd	3
2	3rd	5
3	4th	7

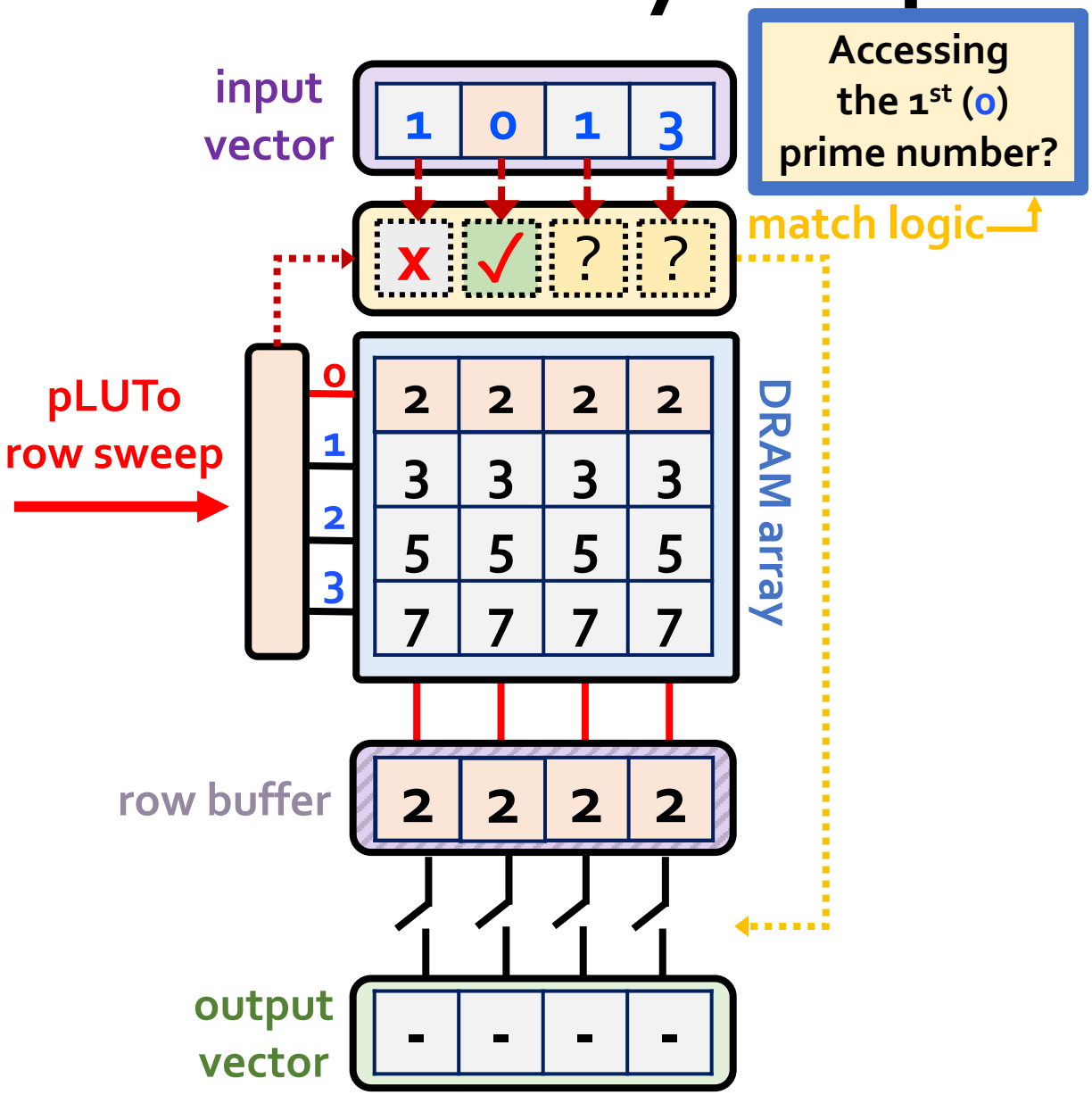
lookup table

1	0	1	3

input vector

3	2	3	7

output vector



In-DRAM pLUTo LUT Query: Step 1

LUT Query:
Return {2nd, 1st, 2nd, 4th} *prime numbers*

LUT index	Prime numbers: i	Prime numbers: f(i)
0	1st	2
1	2nd	3
2	3rd	5
3	4th	7

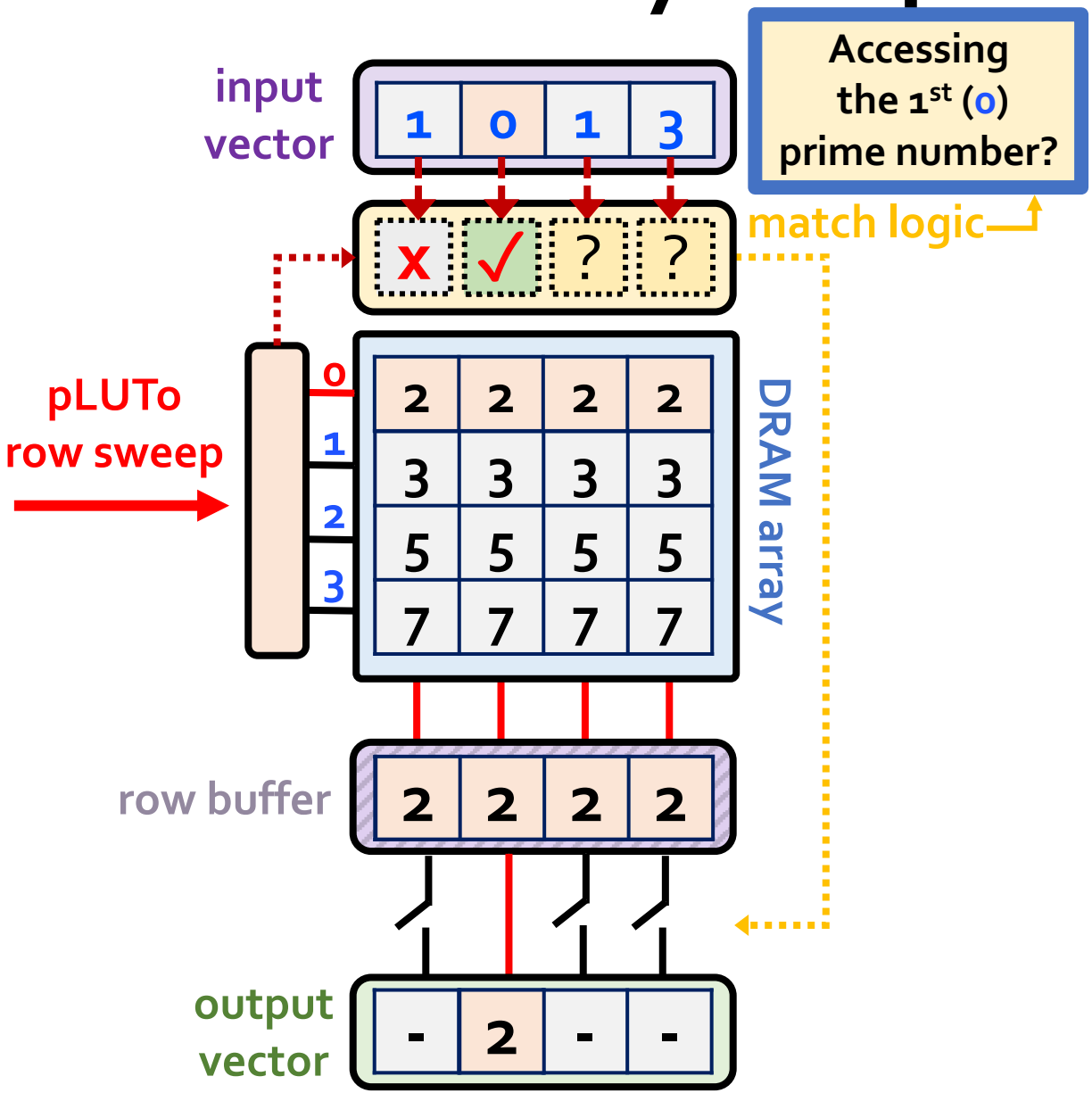
lookup table

1	0	1	3

input vector

3	2	3	7

output vector



In-DRAM pLUTo LUT Query: Step 1

LUT Query:
Return {2nd, 1st, 2nd, 4th}
prime numbers

LUT index	Prime numbers i	f(i)
0	1st	2
1	2nd	3
2	3rd	5
3	4th	7

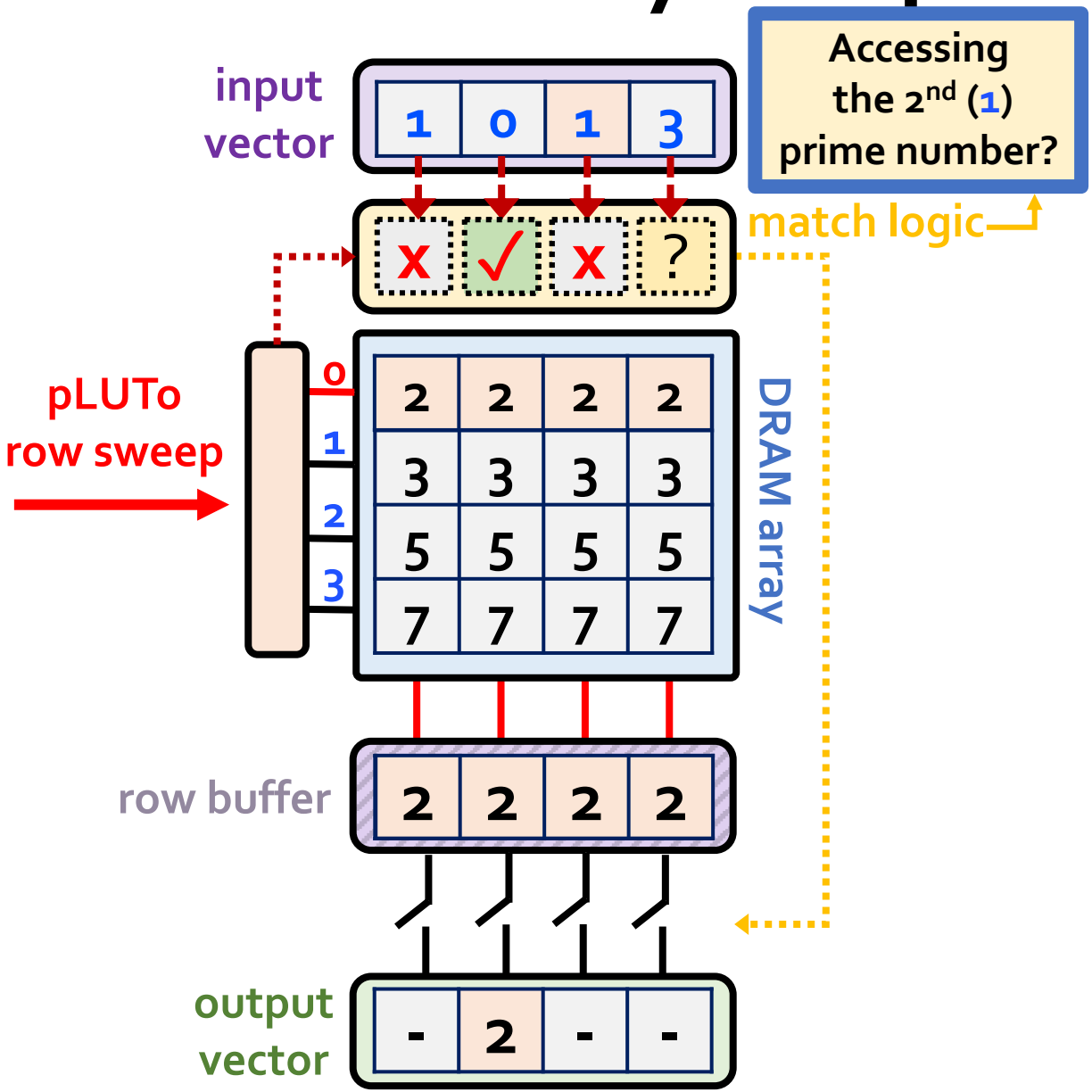
lookup table

1	0	1	3

input vector

3	2	3	7

output vector



In-DRAM pLUTo LUT Query: Step 1

LUT Query:
Return {2nd, 1st, 2nd, 4th}
prime numbers

LUT index	Prime numbers: i	Prime numbers: f(i)
0	1st	2
1	2nd	3
2	3rd	5
3	4th	7

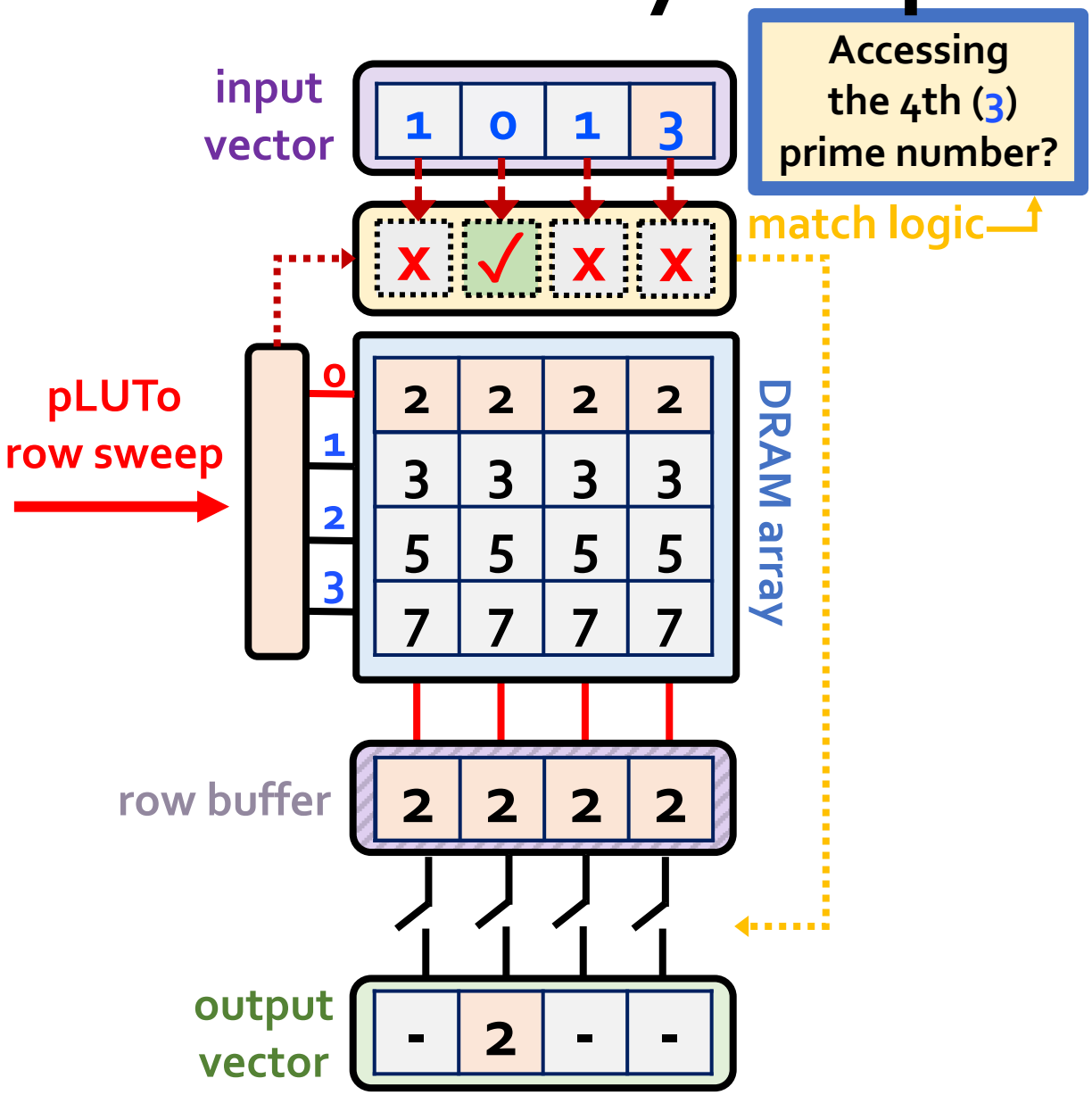
lookup table

1	0	1	3

input vector

3	2	3	7

output vector



In-DRAM pLUTo LUT Query: Step 2

LUT Query:
Return {2nd, 1st, 2nd, 4th}
prime numbers

LUT index	Prime numbers: i	Prime numbers: f(i)
0	1st	2
1	2nd	3
2	3rd	5
3	4th	7

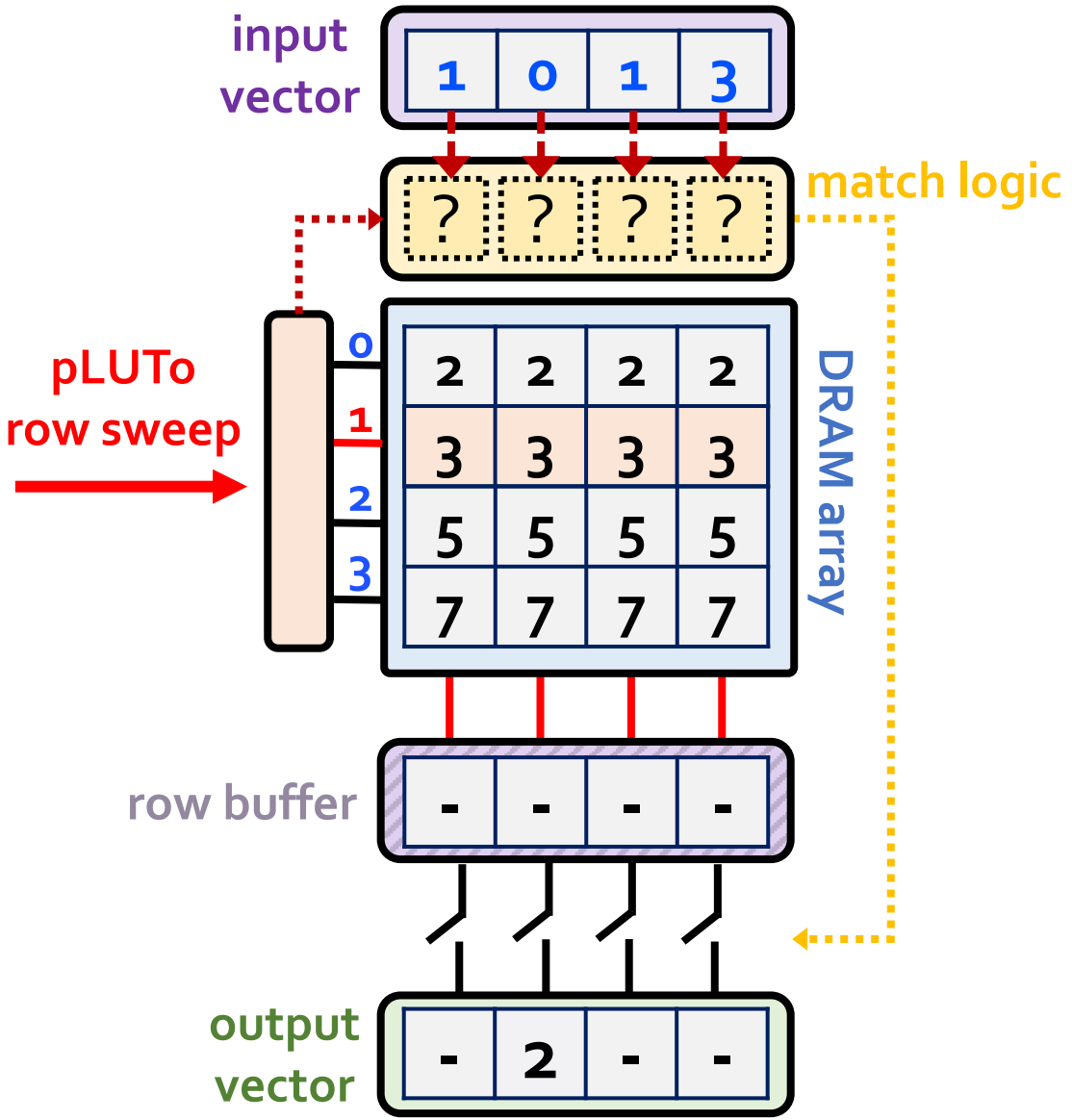
lookup table

1	0	1	3

input vector

3	2	3	7

output vector



In-DRAM pLUTo LUT Query: Step 2

LUT Query:
Return {2nd, 1st, 2nd, 4th}
prime numbers

LUT index	Prime numbers: i	Prime numbers: f(i)
0	1st	2
1	2nd	3
2	3rd	5
3	4th	7

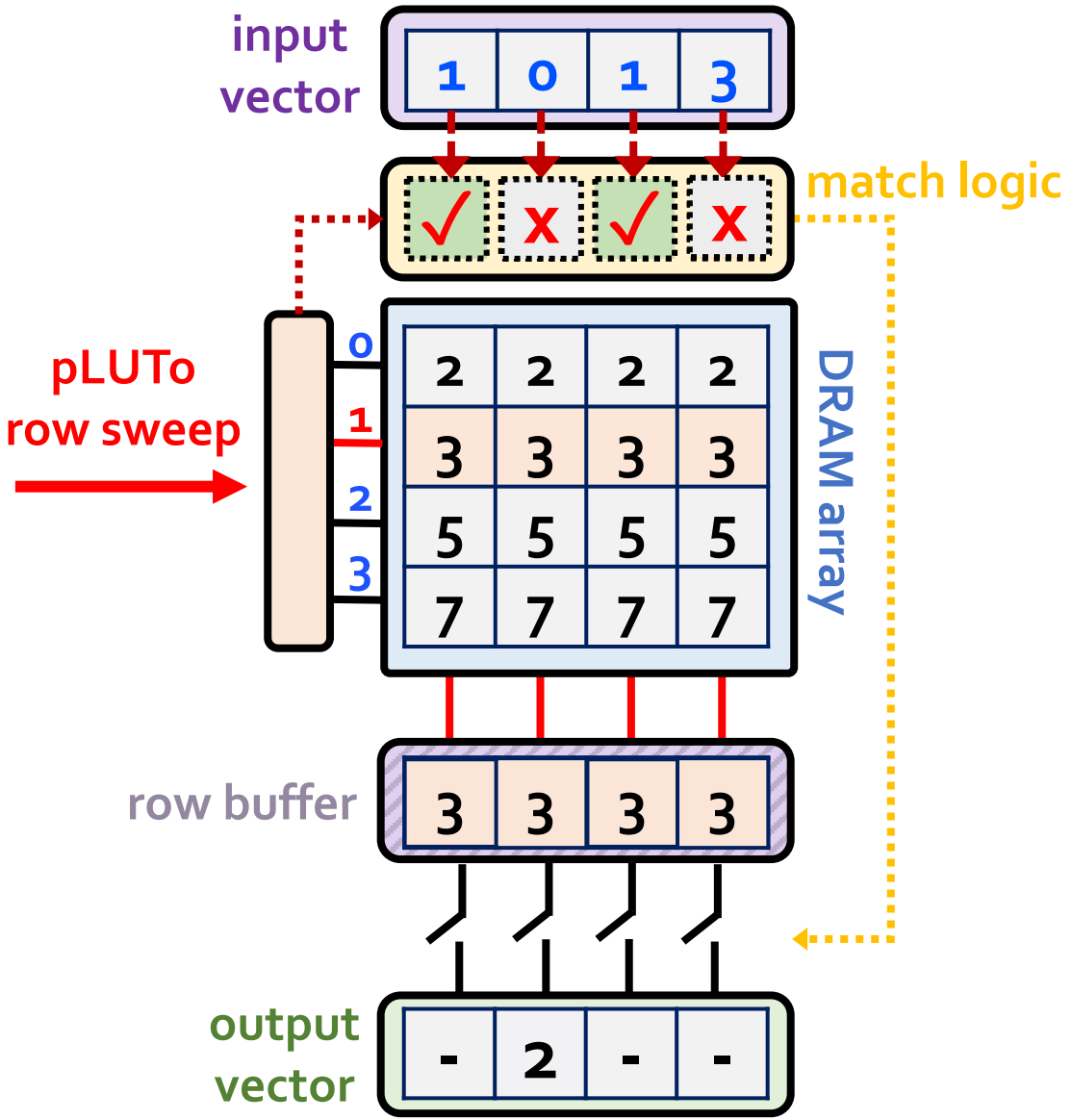
lookup table

1	0	1	3

input vector

3	2	3	7

output vector



In-DRAM pLUTo LUT Query: Step 2

LUT Query:
Return {2nd, 1st, 2nd, 4th}
prime numbers

LUT index	Prime numbers: i	Prime numbers: f(i)
0	1st	2
1	2nd	3
2	3rd	5
3	4th	7

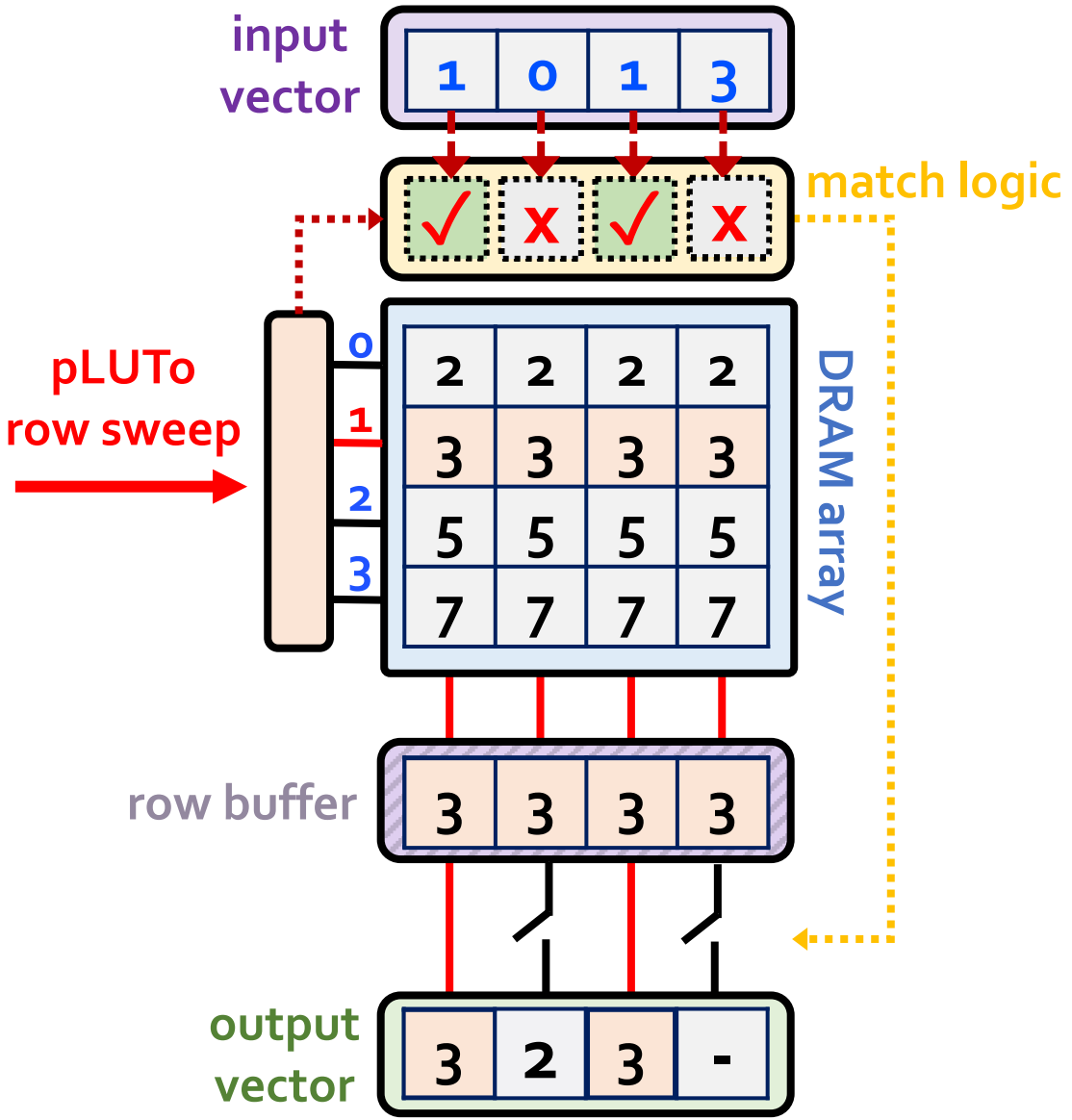
lookup table

1	0	1	3

input vector

3	2	3	7

output vector



In-DRAM pLUTo LUT Query: Step 3

LUT Query:
Return {2nd, 1st, 2nd, 4th}
prime numbers

LUT index	Prime numbers i	f(i)
0	1st	2
1	2nd	3
2	3rd	5
3	4th	7

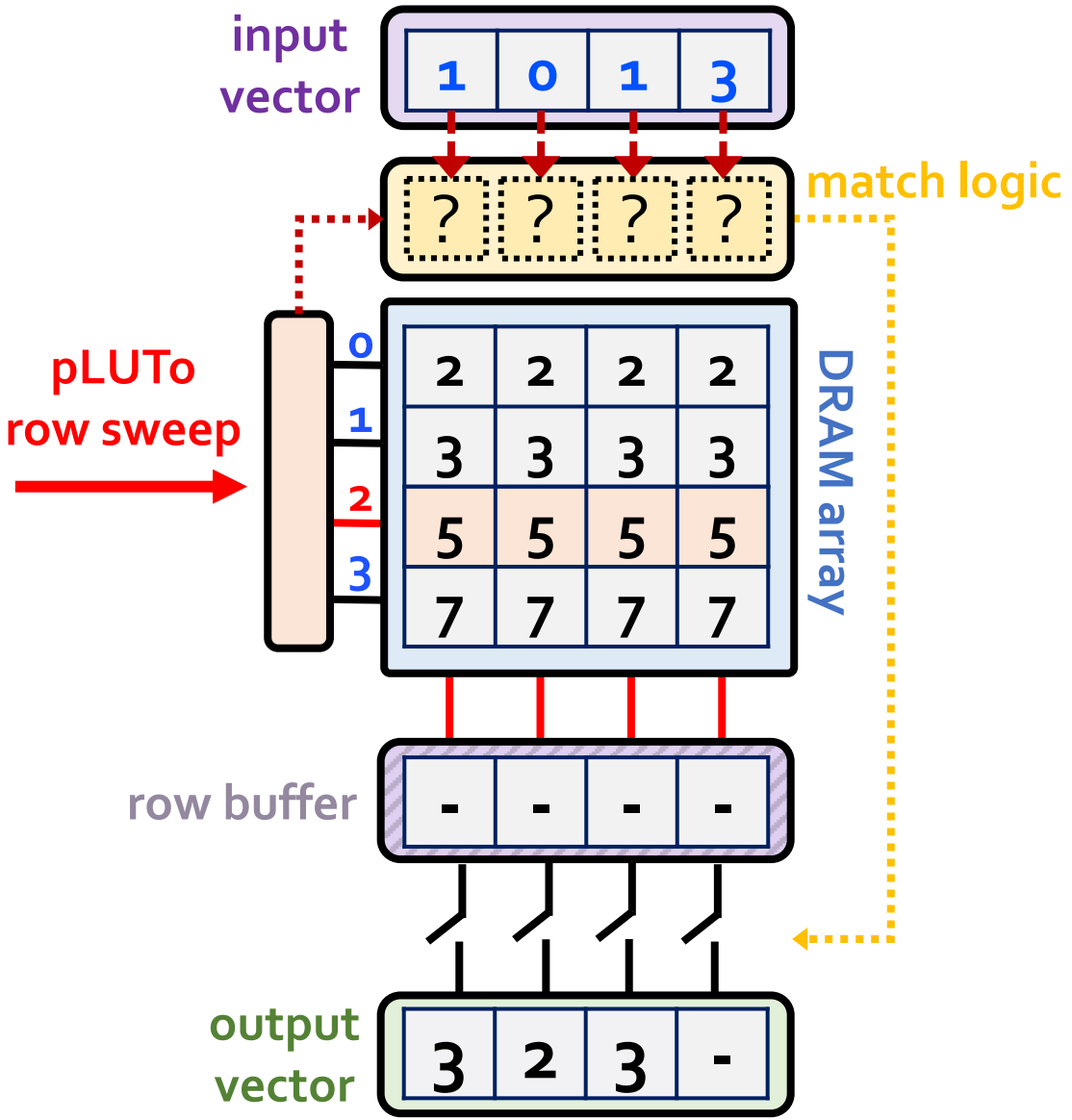
lookup table

1	0	1	3

input vector

3	2	3	7

output vector



In-DRAM pLUTo LUT Query: Step 3

LUT Query:
Return {2nd, 1st, 2nd, 4th}
prime numbers

LUT index	Prime numbers i	f(i)
0	1st	2
1	2nd	3
2	3rd	5
3	4th	7

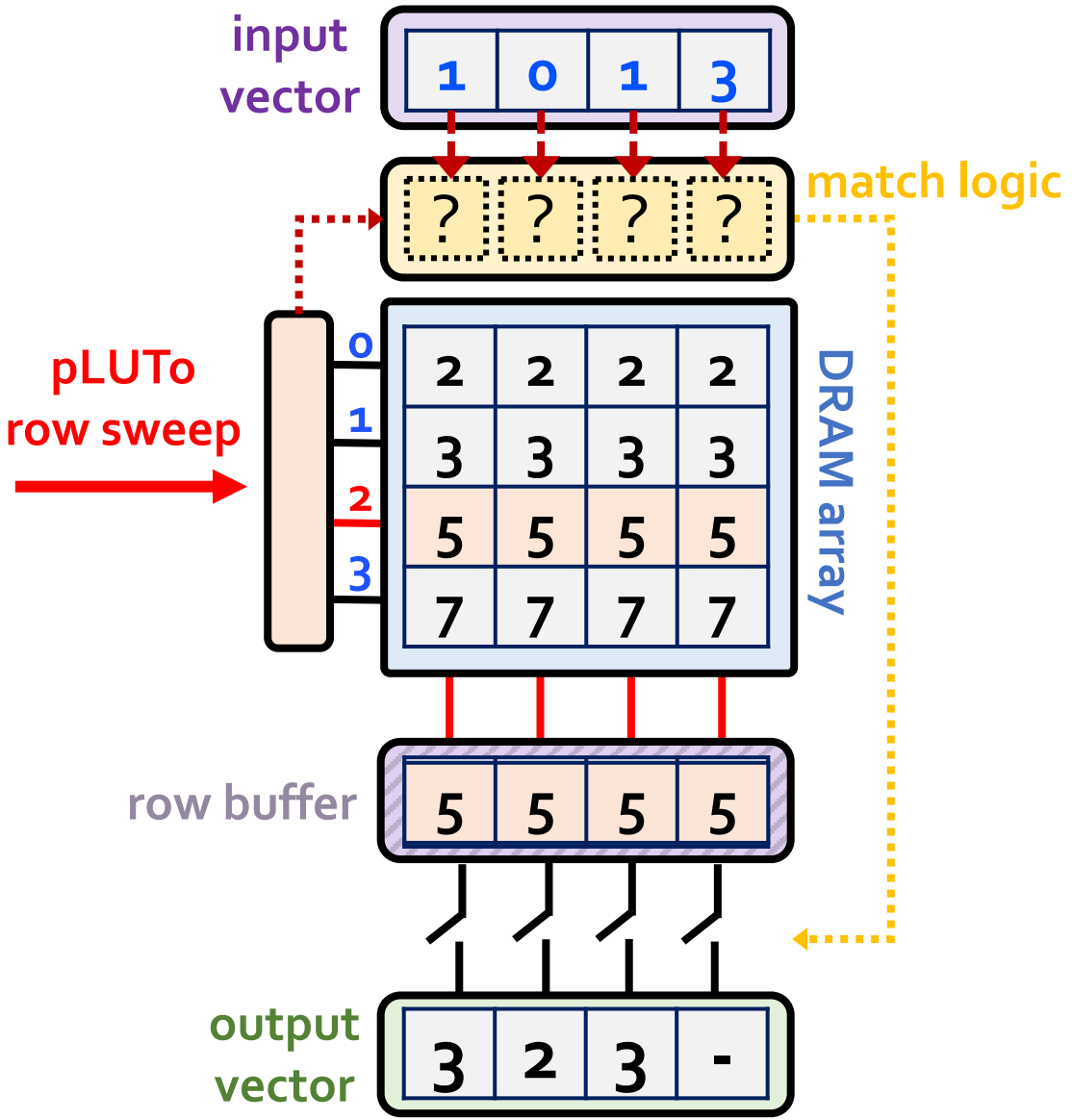
lookup table

1	0	1	3

input vector

3	2	3	7

output vector



In-DRAM pLUTo LUT Query: Step 3

LUT Query:
Return {2nd, 1st, 2nd, 4th}
prime numbers

LUT index	Prime numbers i	f(i)
0	1st	2
1	2nd	3
2	3rd	5
3	4th	7

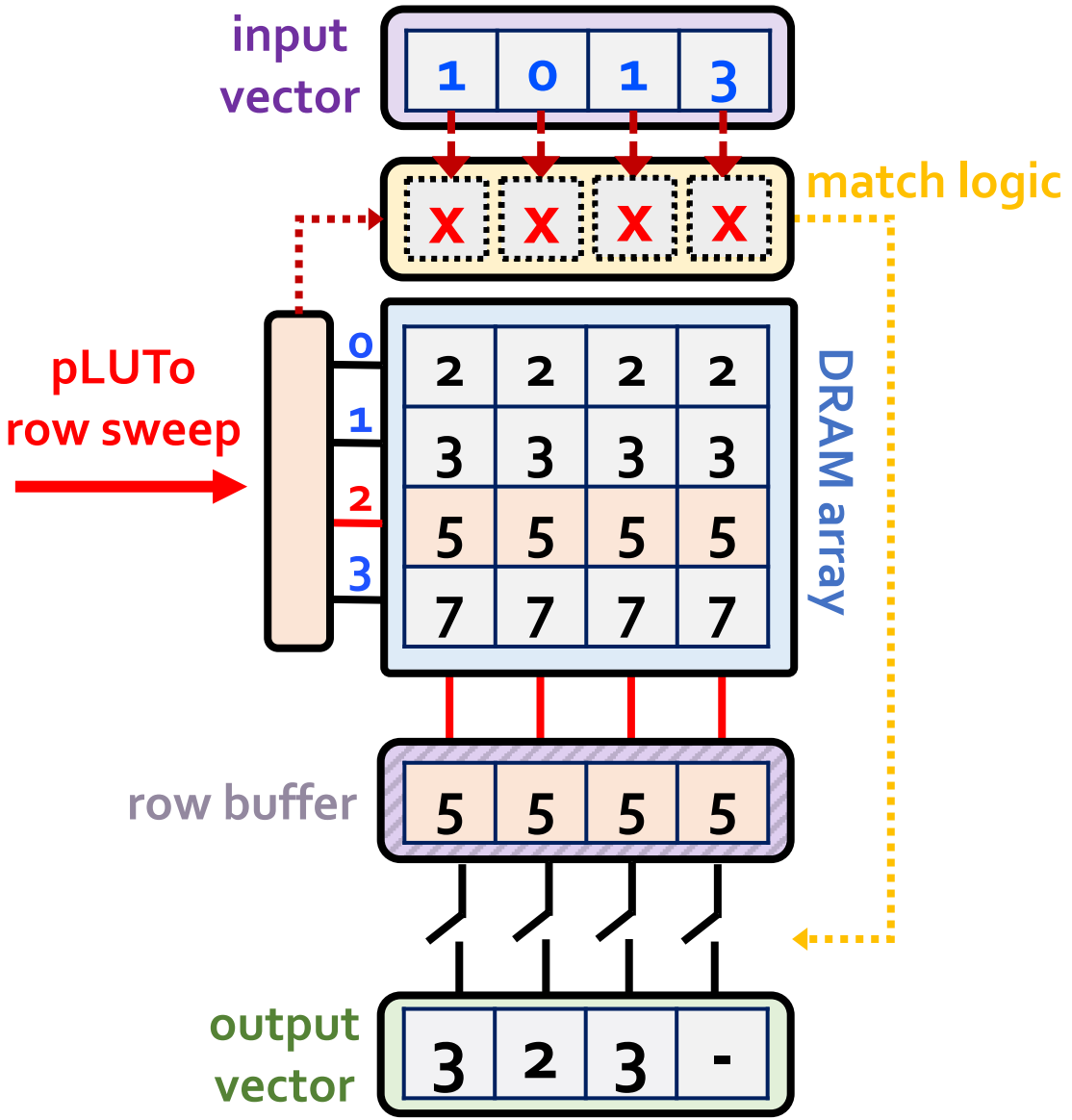
lookup table

1	0	1	3

input vector

3	2	3	7

output vector



In-DRAM pLUTo LUT Query: Step 4

LUT Query:
Return {2nd, 1st, 2nd, 4th}
prime numbers

LUT index	Prime numbers i	f(i)
0	1st	2
1	2nd	3
2	3rd	5
3	4th	7

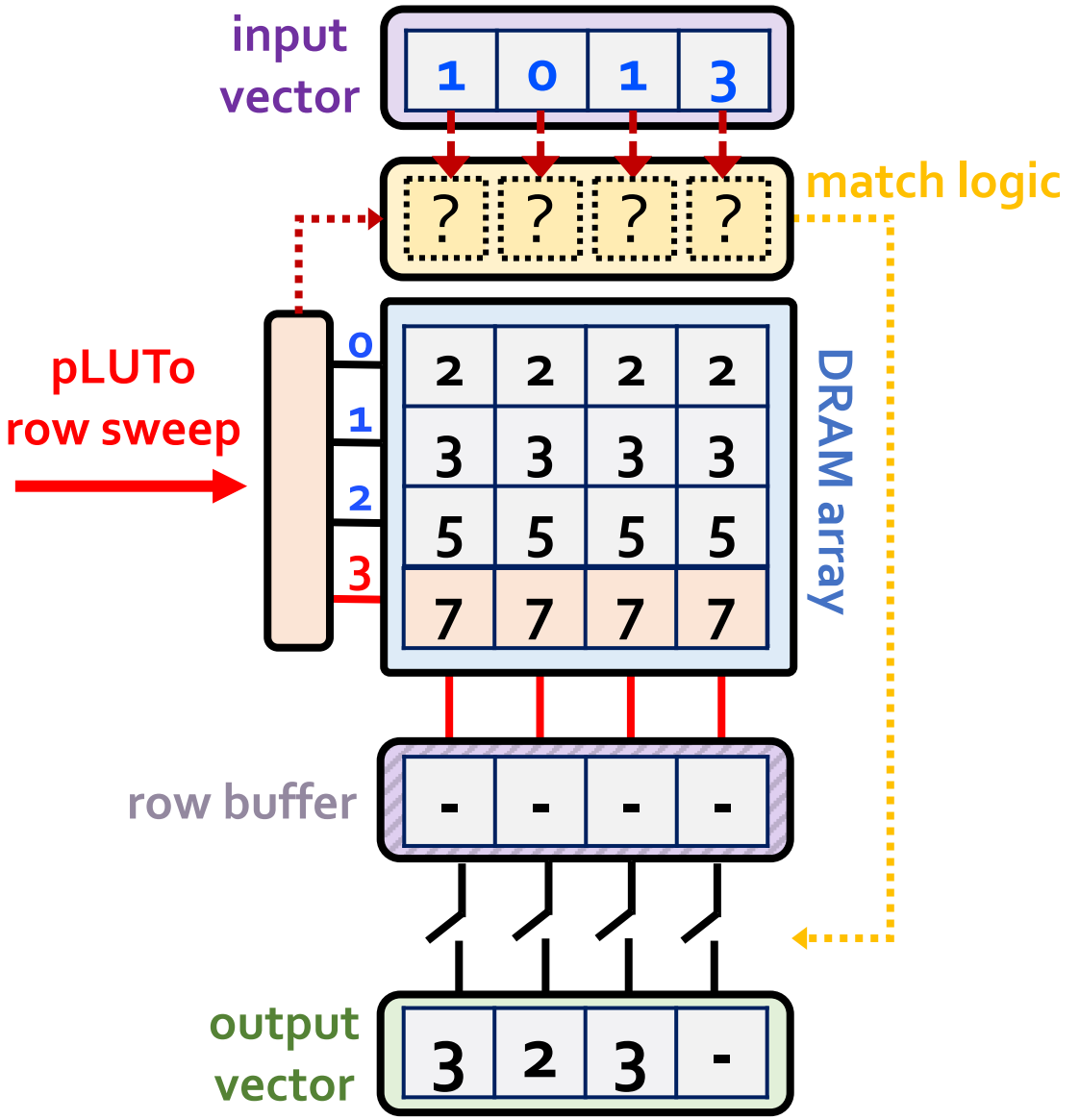
lookup table

1	0	1	3

input vector

3	2	3	7

output vector



In-DRAM pLUTo LUT Query: Step 4

LUT Query:
Return {2nd, 1st, 2nd, 4th}
prime numbers

LUT index	Prime numbers i	f(i)
0	1st	2
1	2nd	3
2	3rd	5
3	4th	7

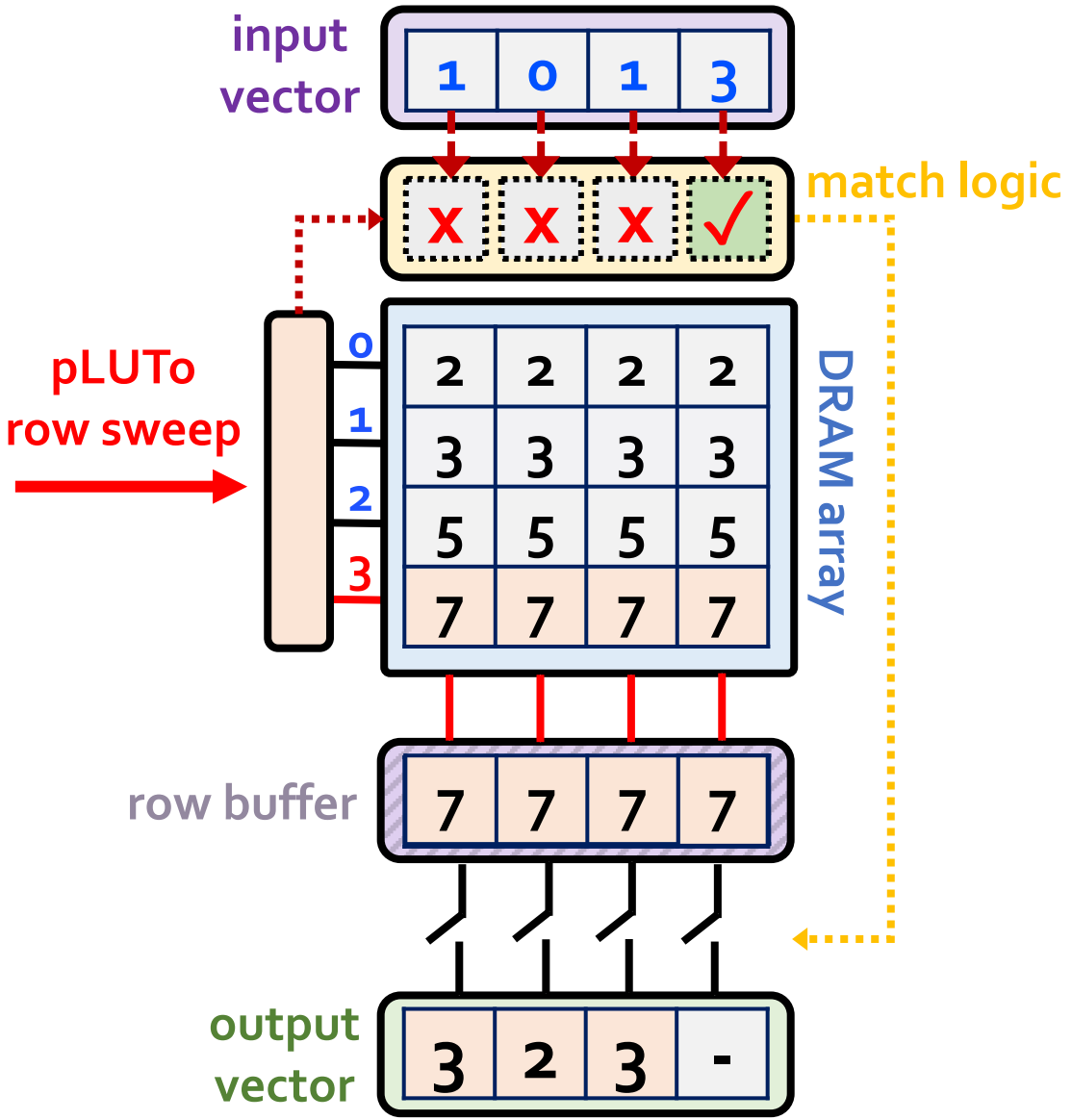
lookup table

1	0	1	3

input vector

3	2	3	7

output vector



In-DRAM pLUTo LUT Query: Step 4

LUT Query:
Return {2nd, 1st, 2nd, 4th}
prime numbers

LUT index	Prime numbers: i	Prime numbers: f(i)
0	1st	2
1	2nd	3
2	3rd	5
3	4th	7

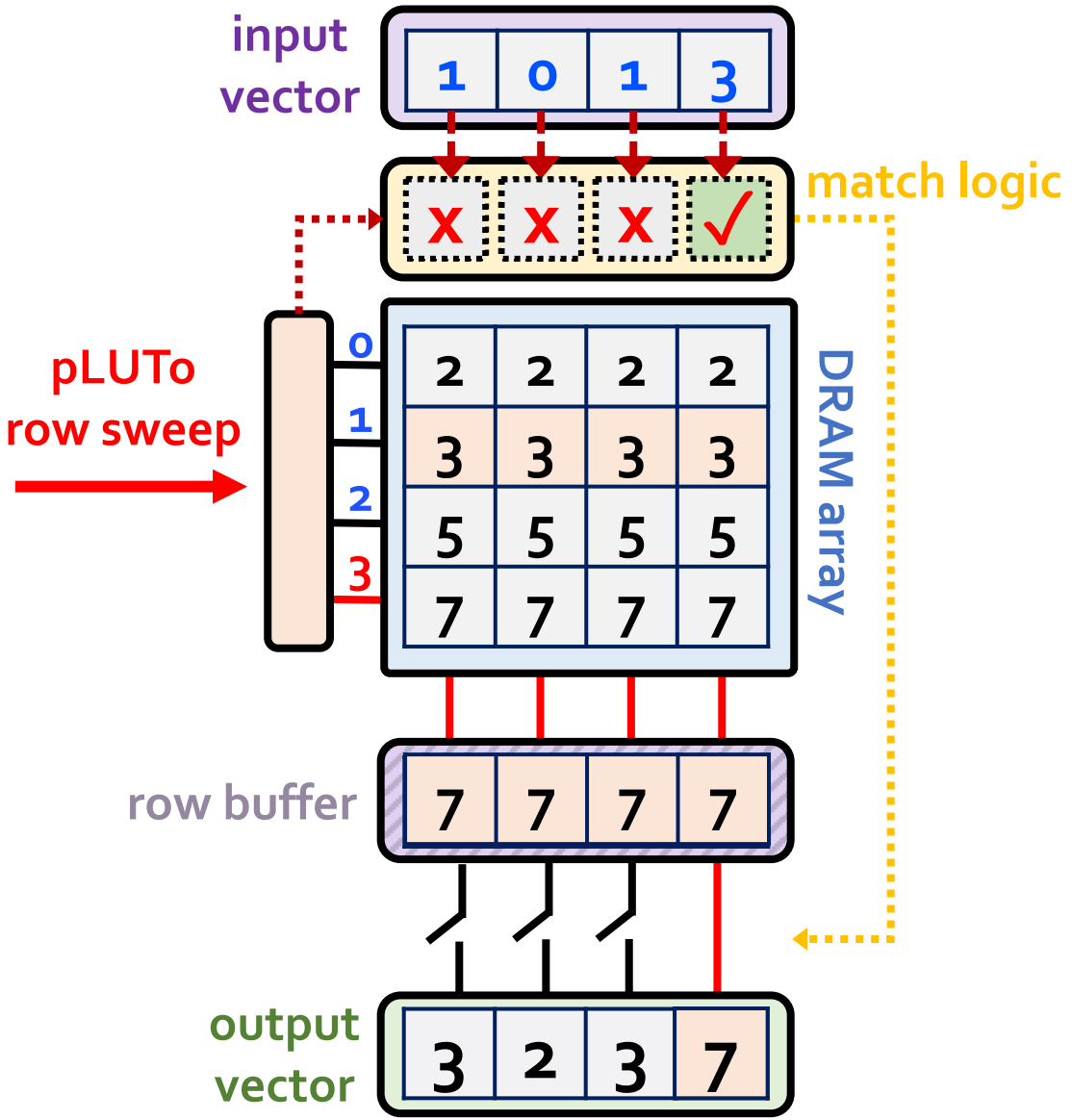
lookup table

1	0	1	3

input vector

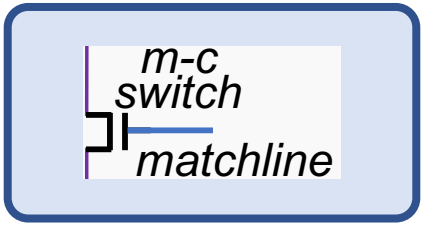
3	2	3	7

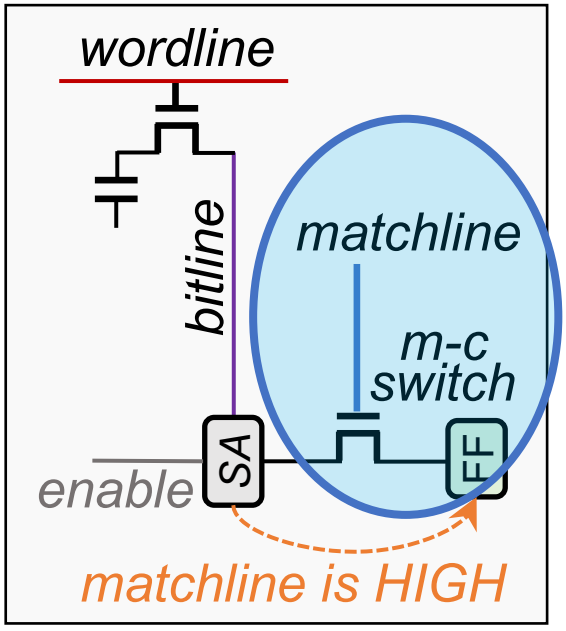
output vector



pLUTo Designs

- **Match Logic:** shared by the three pLUTo designs





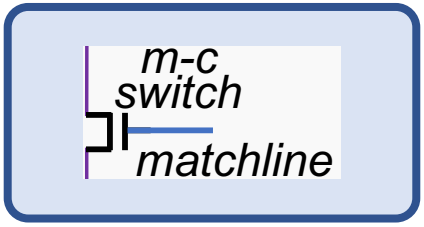
BSA
Buffered Sense Amplifier

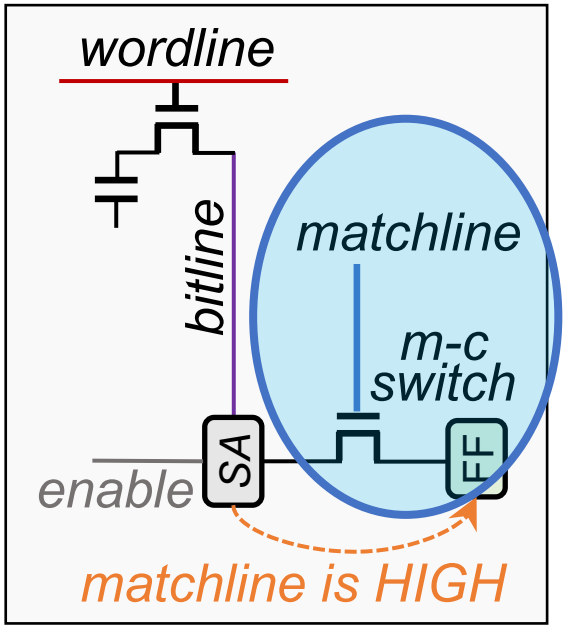
GSA
Gated Sense Amplifier

GMC
Gated Memory Cell

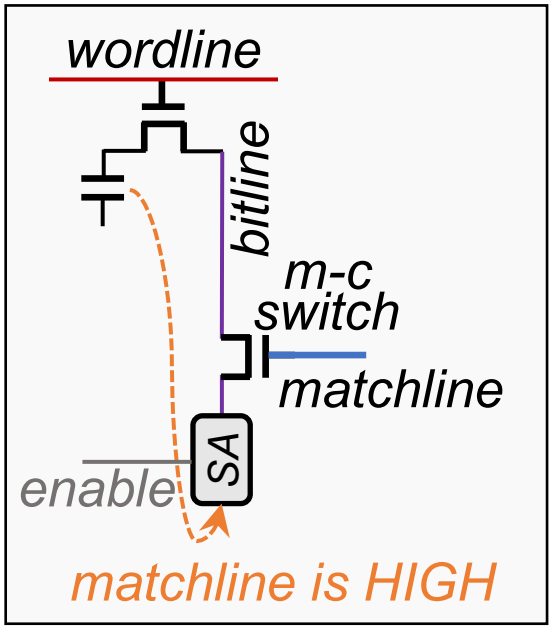
pLUTo Designs

- **Match Logic:** shared by the three pLUTo designs





BSA
Buffered Sense Amplifier

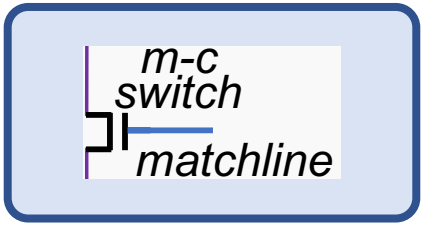


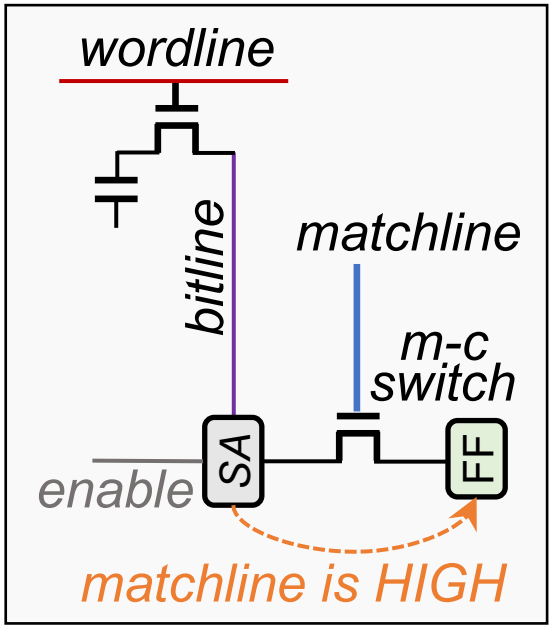
GSA
Gated Sense Amplifier

GMC
Gated Memory Cell

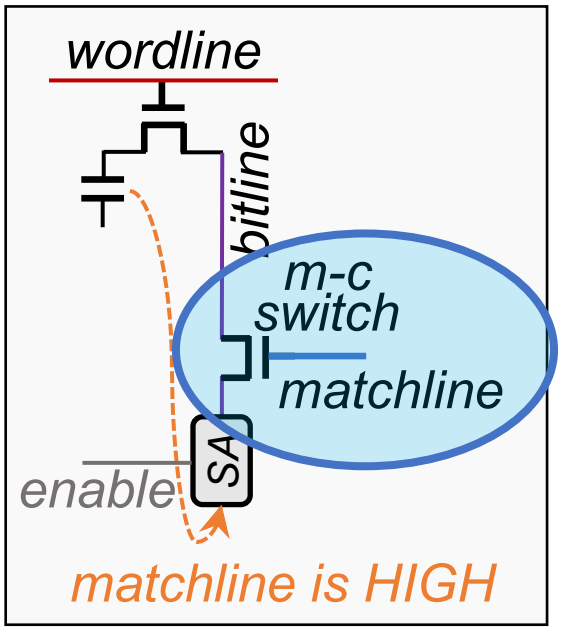
pLUTo Designs

- **Match Logic:** shared by the three pLUTo designs





BSA
Buffered Sense Amplifier

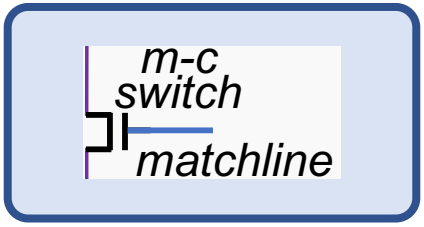


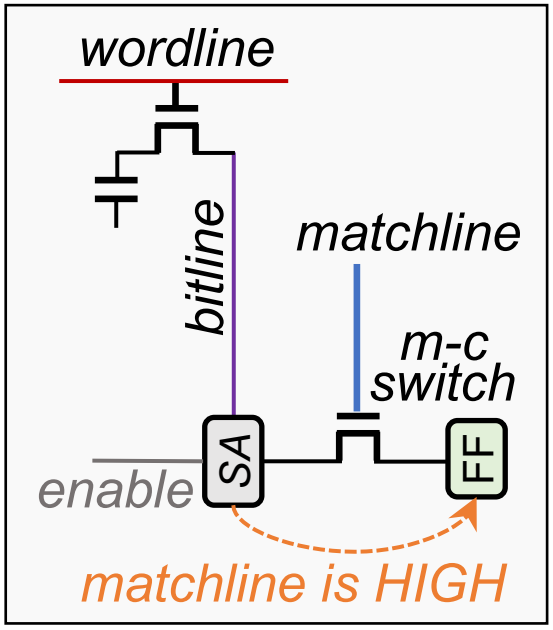
GSA
Gated Sense Amplifier

GMC
Gated Memory Cell

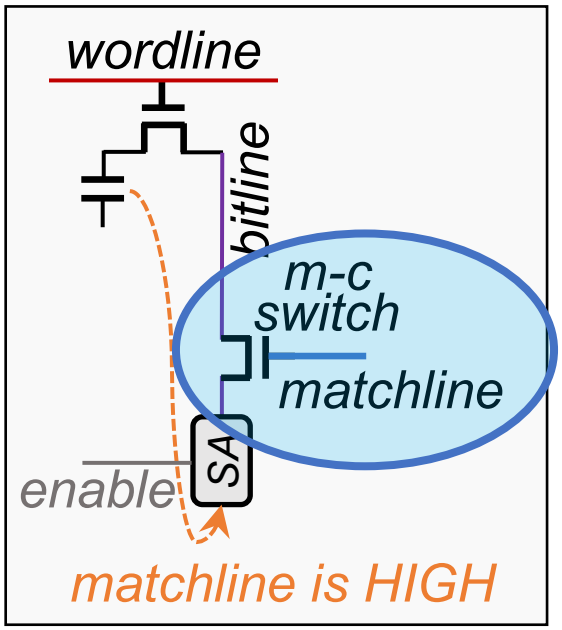
pLUTo Designs

- **Match Logic:** shared by the three pLUTo designs

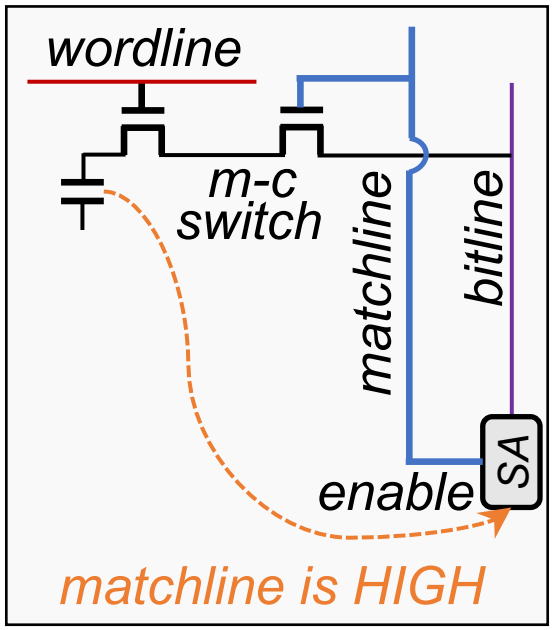




BSA
Buffered Sense Amplifier



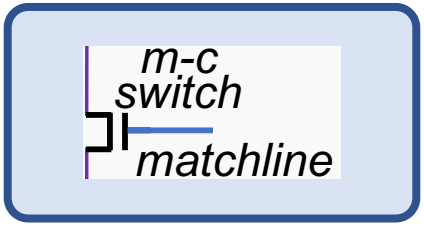
GSA
Gated Sense Amplifier

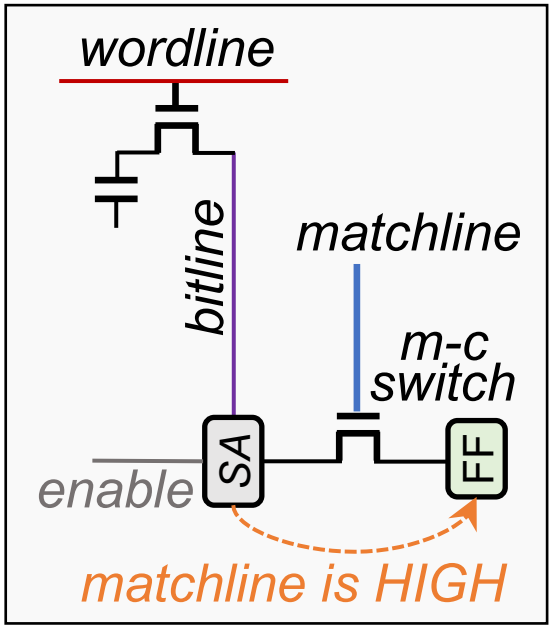


GMC
Gated Memory Cell

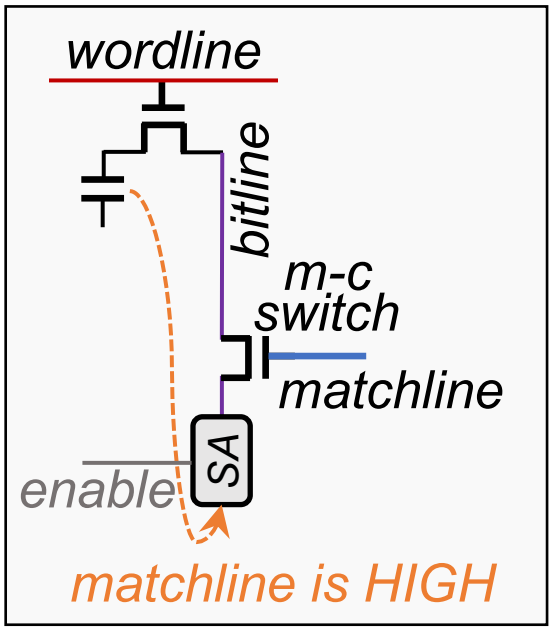
pLUTo Designs

- **Match Logic:** shared by the three pLUTo designs

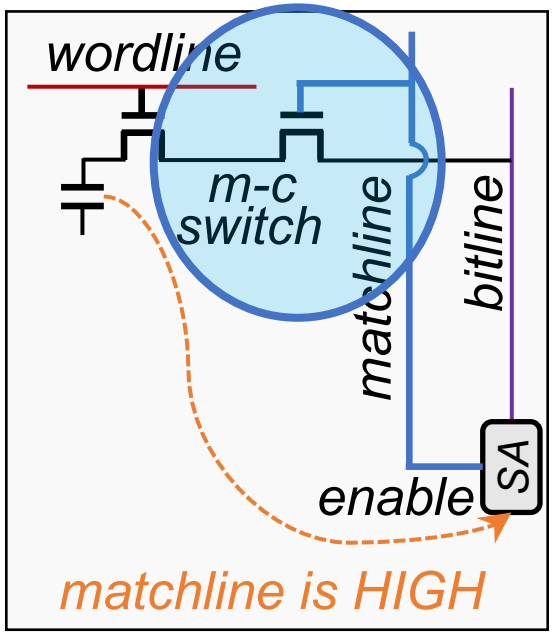




BSA
Buffered Sense Amplifier



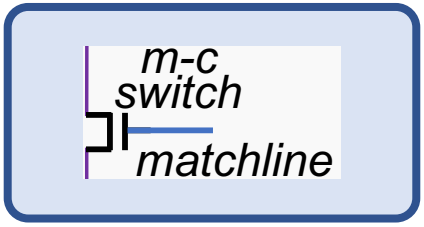
GSA
Gated Sense Amplifier

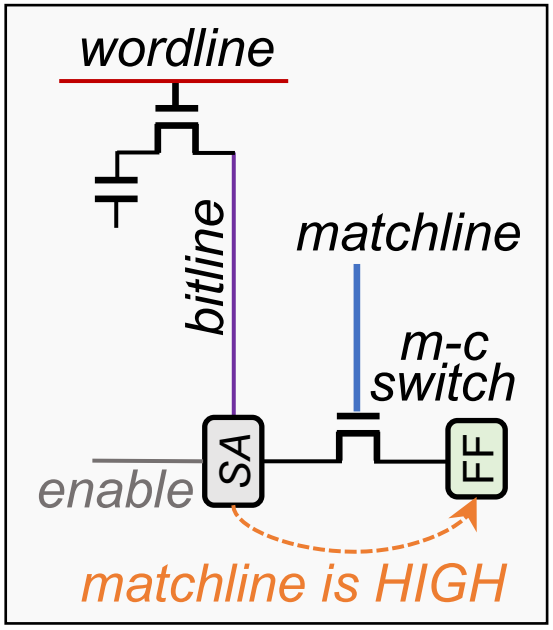


GMC
Gated Memory Cell

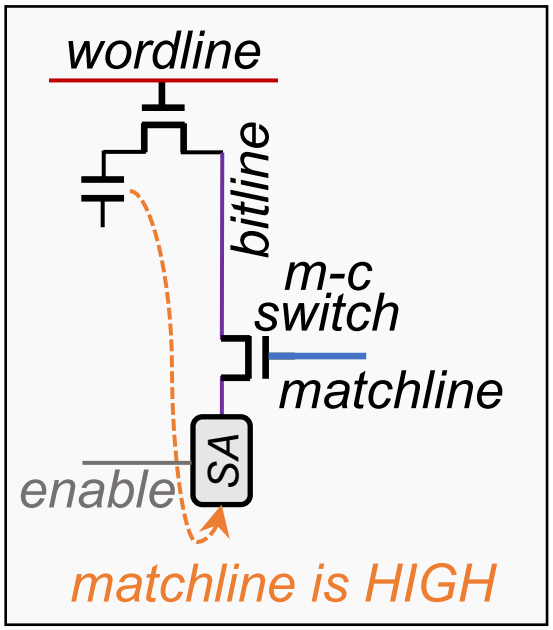
pLUTo Designs

- **Match Logic:** shared by the three pLUTo designs

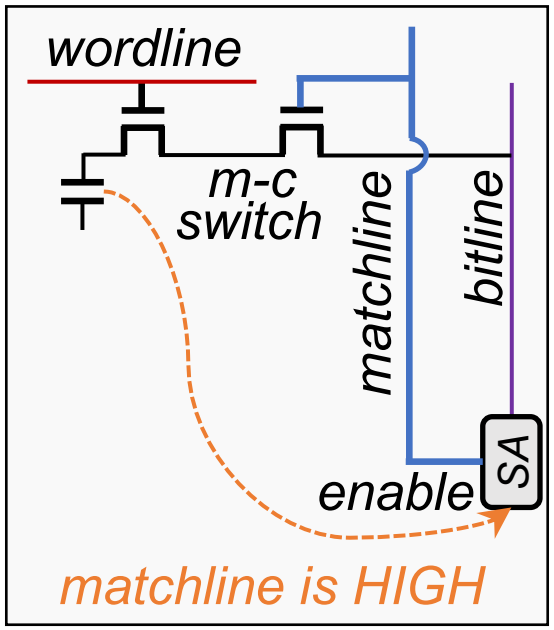




BSA
Buffered Sense Amplifier

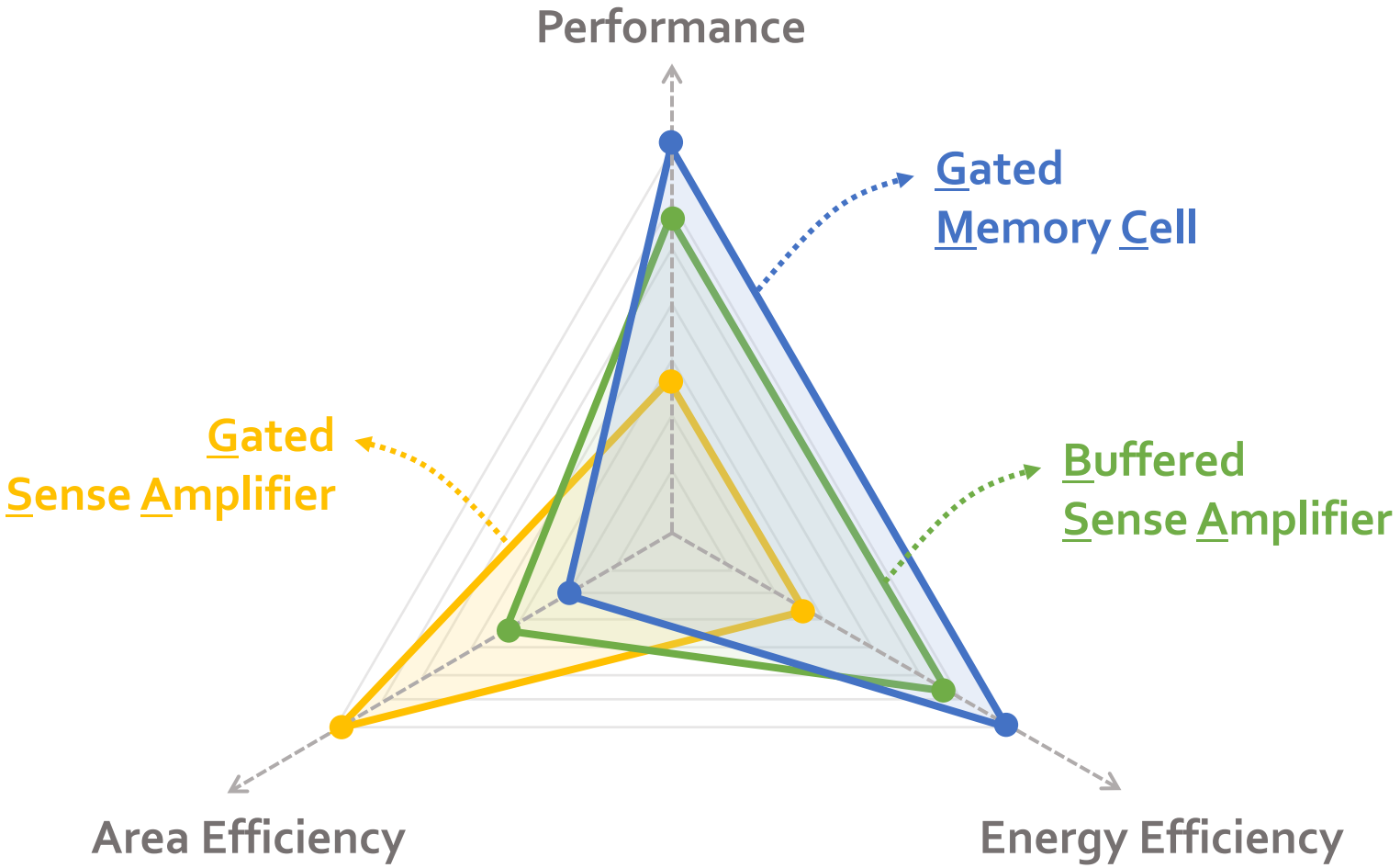


GSA
Gated Sense Amplifier



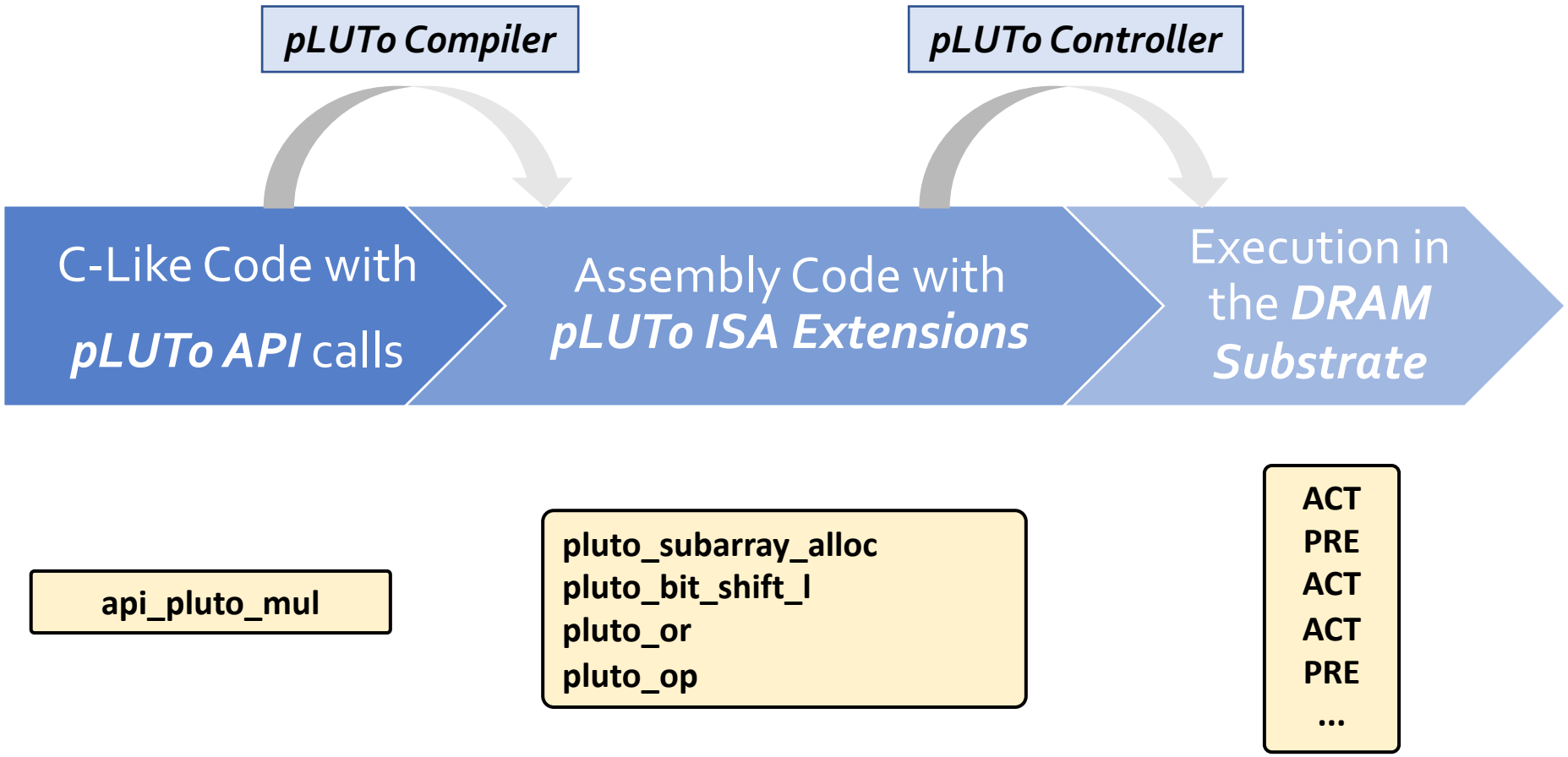
GMC
Gated Memory Cell

pLUTo Designs: Tradeoff Space

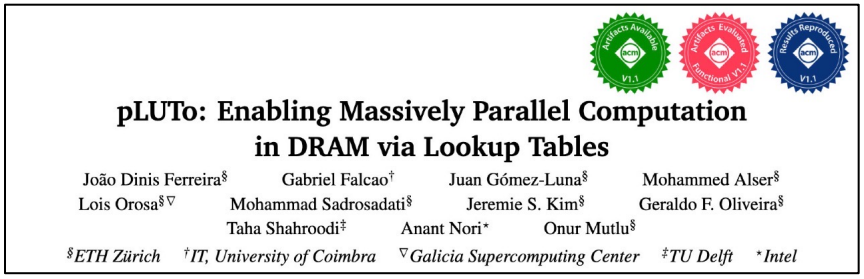


pLUTo designs cover a *broad design space* and provide *different* performance, energy, and area efficiency

System Integration



More in the Paper



pLUTo: Enabling Massively Parallel Computation in DRAM via Lookup Tables

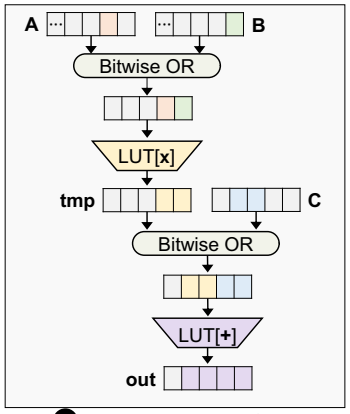
João Dinis Ferreira[§] Gabriel Falcao[†] Juan Gómez-Luna[§] Mohammed Alser[§]
Lois Orosa[§▽] Mohammad Sadrosadati[§] Jeremie S. Kim[§] Geraldo F. Oliveira[§]
Taha Shahroodi[‡] Anant Nori[⋆] Onur Mutlu[§]
[§]ETH Zürich [†]IT, University of Coimbra [▽]Galicia Supercomputing Center [‡]TU Delft [⋆]Intel

```c
uint2_t *A,*B,*C = (uint2_t *)malloc(input_size*2); // Inputs
uint4_t *out = (uint4_t *)malloc(input_size*4); // Output

// Array initialization
// ...
// Multiply-and-add loop
for (int i = 0; i < input_size; i++) {
      out[i] = A[i]*B[i] + C[i];
}
```

(a) Reference C Code

```c
// Array allocation
uint2_t *A, *B   = pluto_malloc(size=input_size, bitwidth=2);
uint2_t *C, *tmp = pluto_malloc(size=input_size, bitwidth=4);
uint4_t *out     = pluto_malloc(size=input_size, bitwidth=5);

// Multiply-and-add loop
for (int i = 0; i < input_size/row_size; i++){
    api_pluto_mul(in1 = A, in2 = B,   out = tmp, bitwidth = 2);
    api_pluto_add(in1 = C, in2 = tmp, out = out, bitwidth = 4);
}
```

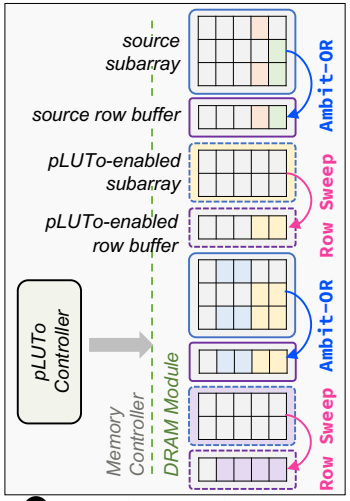
(b) pLUTo API Code



(d) Data Dependency Graph

```
# Array allocation
pluto_row_alloc $prg0, input_size, 2 # Allocate A
pluto_row_alloc $prg1, input_size, 2 # Allocate B
pluto_row_alloc $prg2, input_size, 4 # Allocate C
pluto_row_alloc $prg3, input_size, 4 # Allocate tmp
pluto_row_alloc $prg4, input_size, 5 # Allocate out

# Allocate and load LUTs
pluto_subarray_alloc $lut_rg0, "mul2_lut_file.dat"
pluto_subarray_alloc $lut_rg1, "add4_lut_file.dat"

# Allocate temporary row for OR operation
pluto_row_alloc $prg5, input_size, 8

# Multiply-and-add loop
div $r0, input_size, row_size # Initialize loop counter
LOOP:
    pluto_bit_shift_l $pgr0, 4 # Shift A 4 bits to the left
    pluto_or $prg5, $prg0, $prg1 # $prg5 <- A | B
    pluto_op $prg3, $prg5, $lut_rg0, 256, 4 # tmp <- LUT[A|B]
    pluto_bit_shift_l $pgr3, 4 # Shift tmp 4 bits to the left
    pluto_or $prg5, $prg3, $prg2 # $pgr5 <- tmp | C
    pluto_op $prg4, $prg5, $lut_rg1, 256, 8 # out <- LUT[tmp|C]
    # Update input addresses
    subi $r0, 0     # decrement loop counter
    bne $r0, LOOP   # next loop iteration
```

(c) pLUTo ISA Instructions

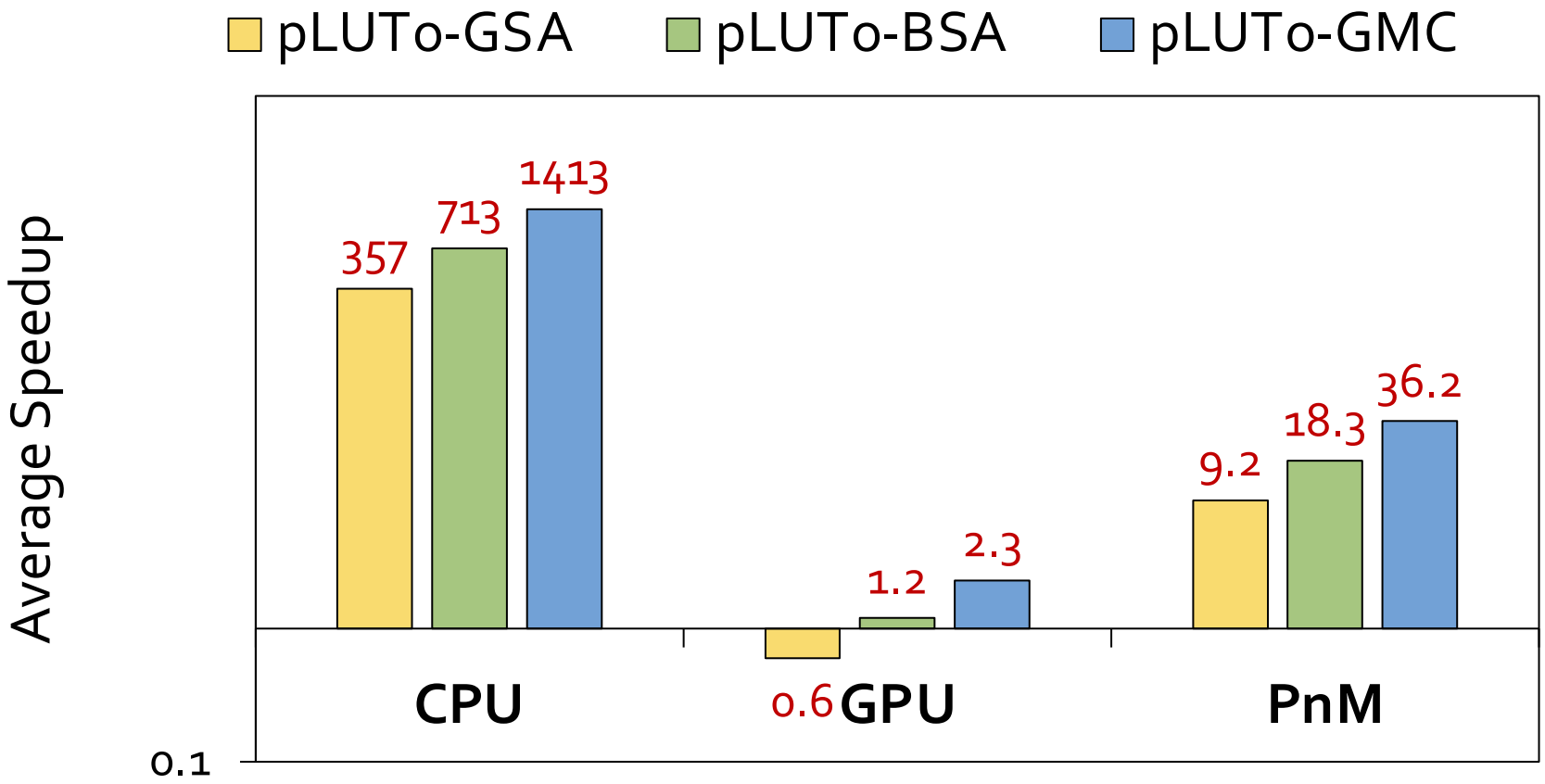


(e) pLUTo Controller & Execution

https://arxiv.org/pdf/2104.07699.pdf

Performance

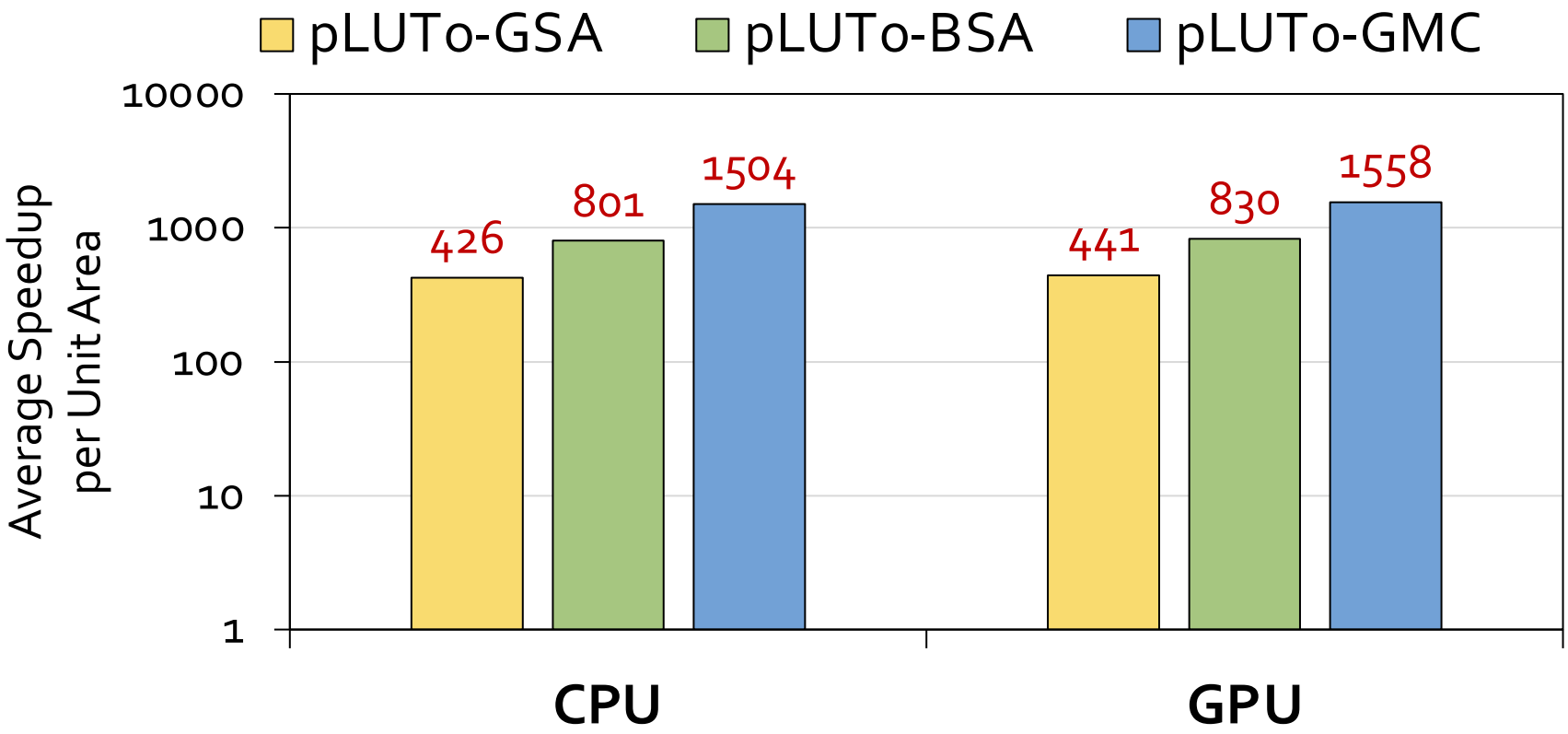
Average speedup across 7 real-world workloads



pLUTo *significantly outperforms* CPU, GPU and PnM baselines

Performance (normalized to area)

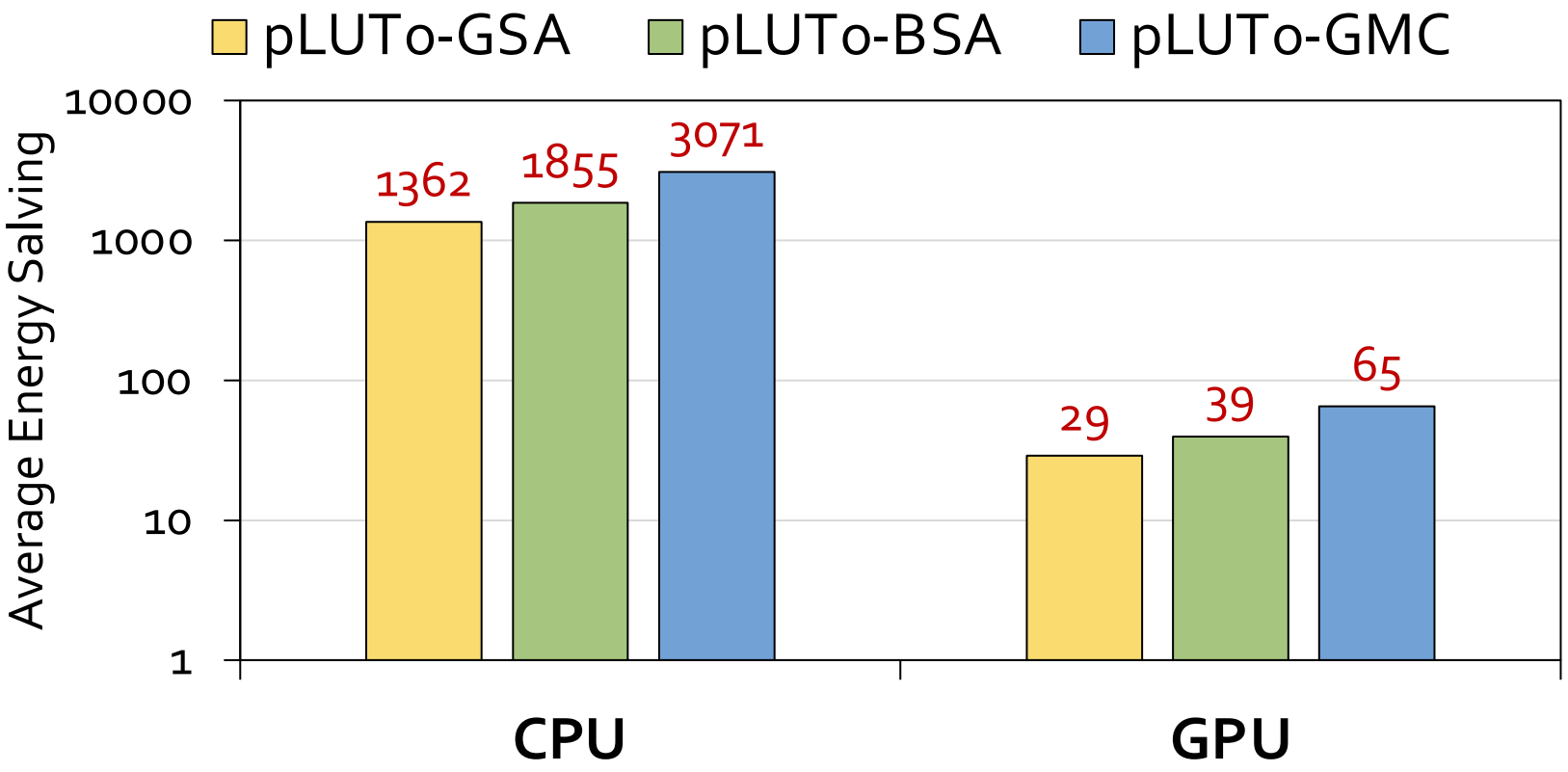
Average speedup normalized to area across 7 real-world workloads



pLUTo provides *substantially higher* performance per unit area than *both* the CPU and the GPU

Energy Consumption

Average energy consumption across 7 real-world workloads



pLUTo *significantly reduces energy consumption* compared to processor-centric architectures for various workloads

More Results in the Paper

- Comparison with FPGA
- Area Overhead Analysis
- Circuit-Level Reliability & Correctness
- Subarray-Level Parallelism
- LUT Loading Overhead
- Range of Supported Operations

pLUTo: Enabling Massively Parallel Computation in DRAM via Lookup Tables

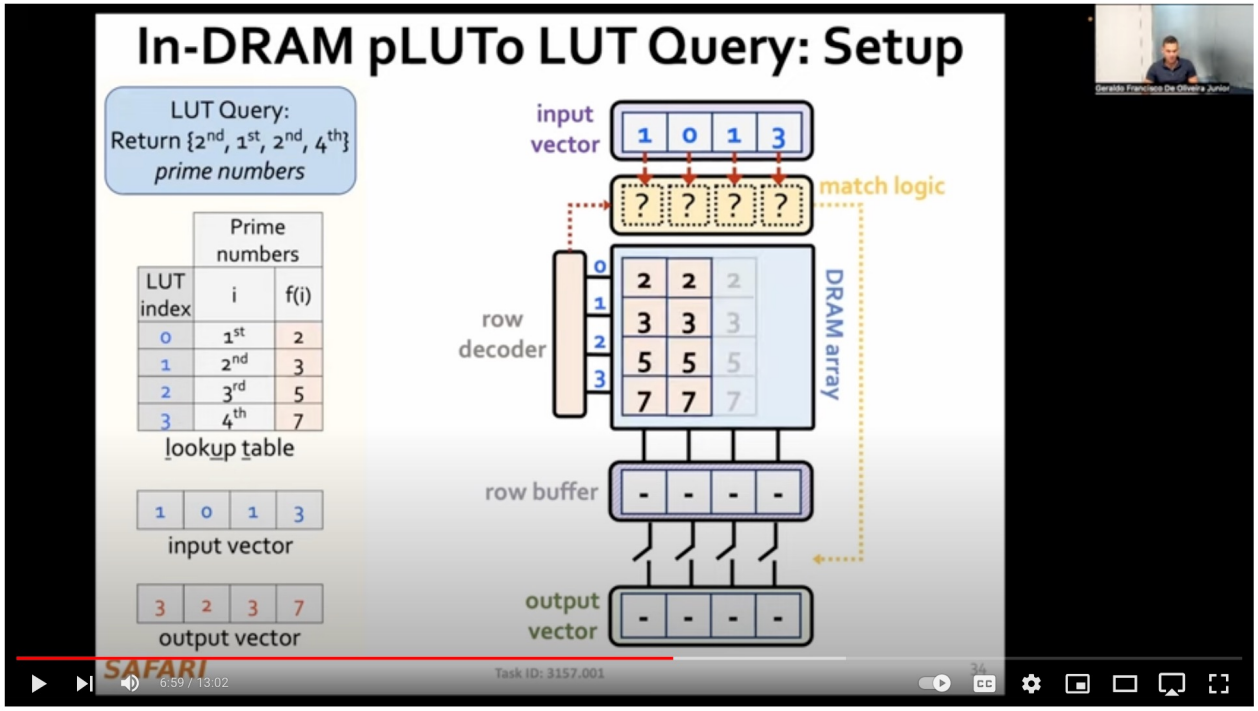
João Dinis Ferreira[§] Gabriel Falcao[†] Juan Gómez-Luna[§] Mohammed Alser[§]
Lois Orosa[§▽] Mohammad Sadrosadati[§] Jeremie S. Kim[§] Geraldo F. Oliveira[§]
Taha Shahroodi[‡] Anant Nori[⋆] Onur Mutlu[§]

[§]*ETH Zürich* [†]*IT, University of Coimbra* [▽]*Galicia Supercomputing Center* [‡]*TU Delft* [⋆]*Intel*

https://arxiv.org/pdf/2104.07699.pdf

SRC TECHCON Presentation

- Geraldo F. Oliveira
  - pLUTo: Enabling Massively Parallel Computation in DRAM via Lookup Tables
  - https://arxiv.org/pdf/2104.07699.pdf



https://youtu.be/9t1FJQ6nNw4?si=bhylWCLZde2DC7os

Processing using Memory in Off-the-Shelf DRAM

RowClone in Off-the-Shelf DRAM Chips

- Idea: Violate DRAM timing parameters to mimic RowClone

ComputeDRAM: In-Memory Compute Using Off-the-Shelf DRAMs

Fei Gao
feig@princeton.edu
Department of Electrical Engineering
Princeton University

Georgios Tziantzioulis
georgios.tziantzioulis@princeton.edu
Department of Electrical Engineering
Princeton University

David Wentzlaff
wentzlaf@princeton.edu
Department of Electrical Engineering
Princeton University

RowClone & Bitwise Ops in Real DRAM Chips

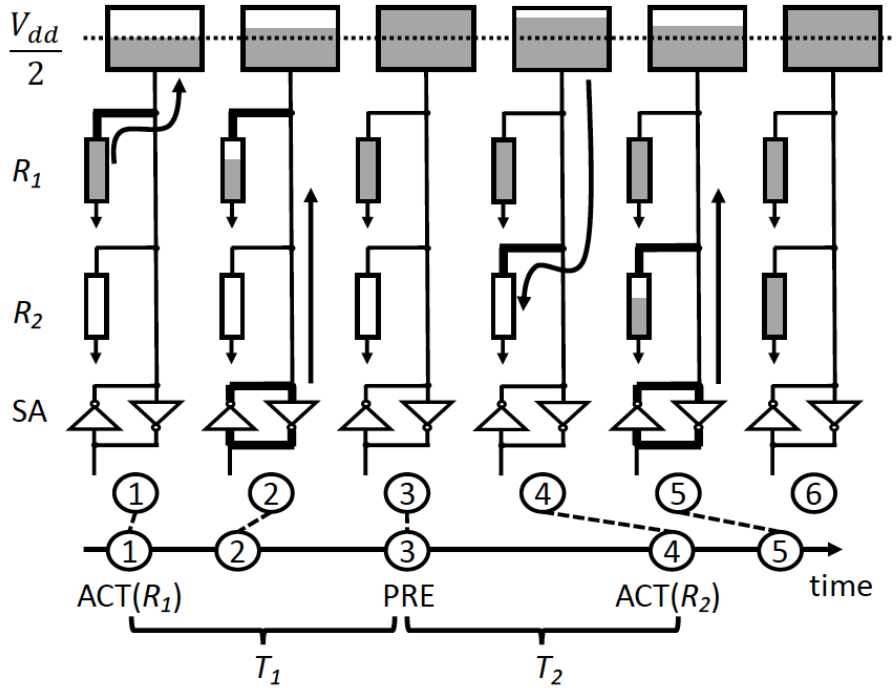


Figure 4: Timeline for a single bit of a column in a row copy operation. The data in R_1 is loaded to the bit-line, and overwrites R_2.

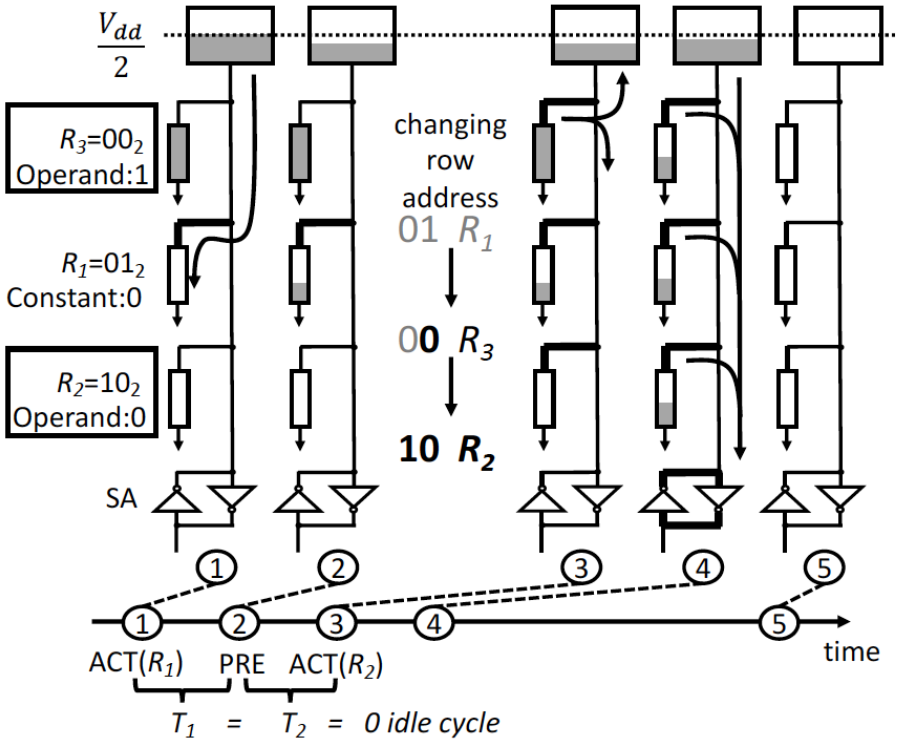
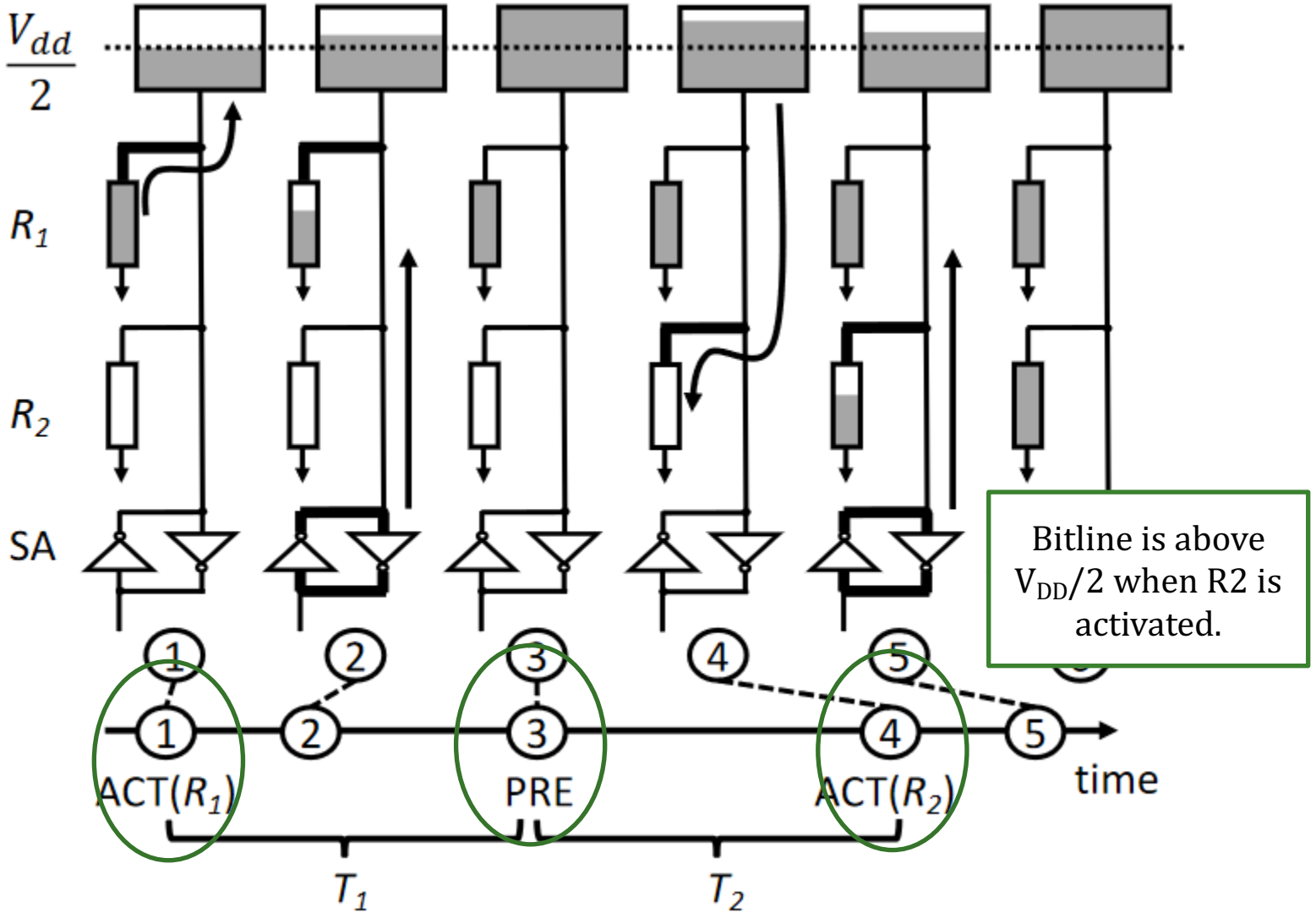
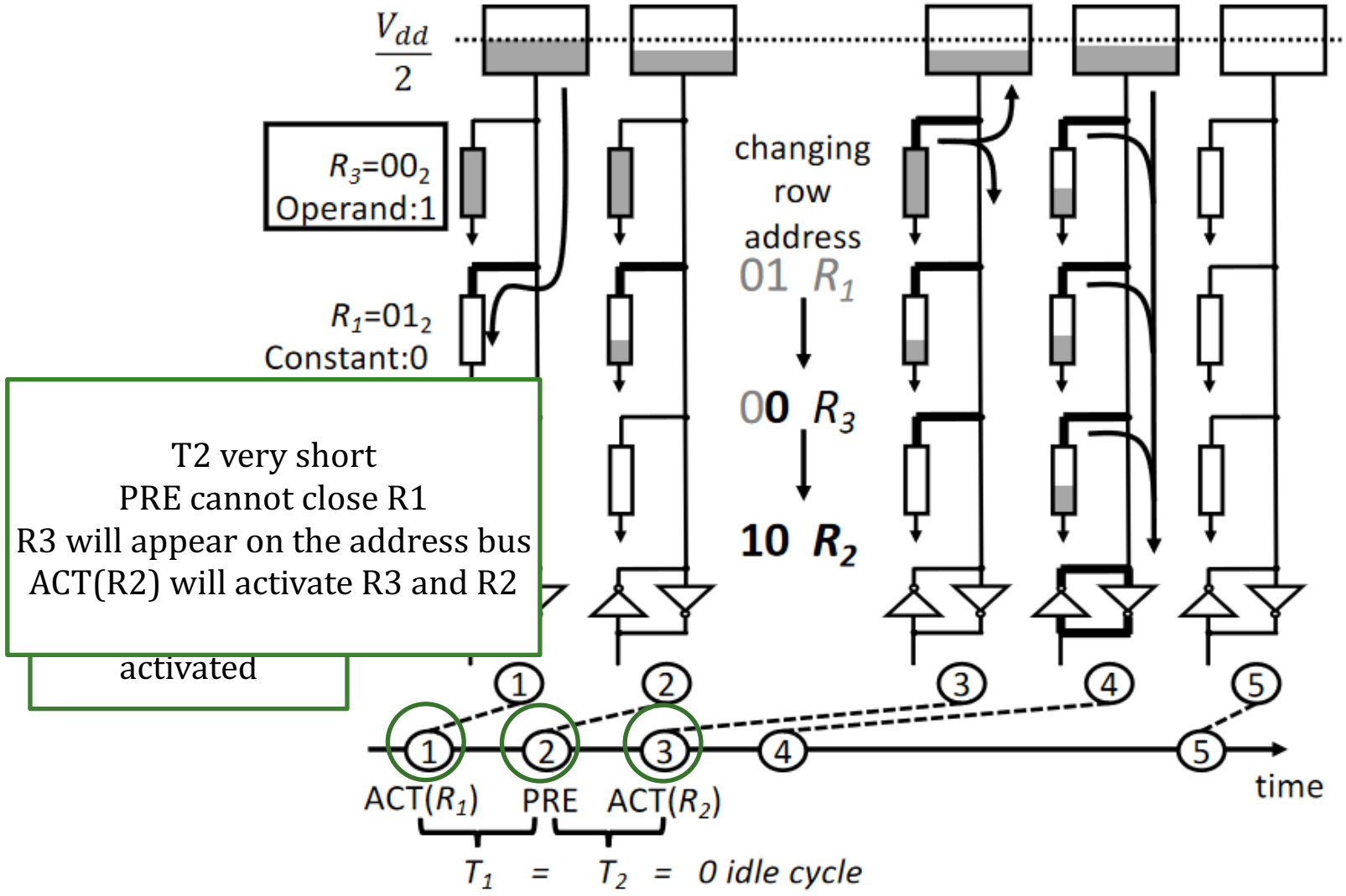


Figure 5: Logical AND in ComputeDRAM. R_1 is loaded with constant zero, and R_2 and R_3 store operands (0 and 1). The result ($0 = 1 \wedge 0$) is finally set in all three rows.

Row Copy in ComputeDRAM



Bitwise AND in ComputeDRAM



Experimental Methodology

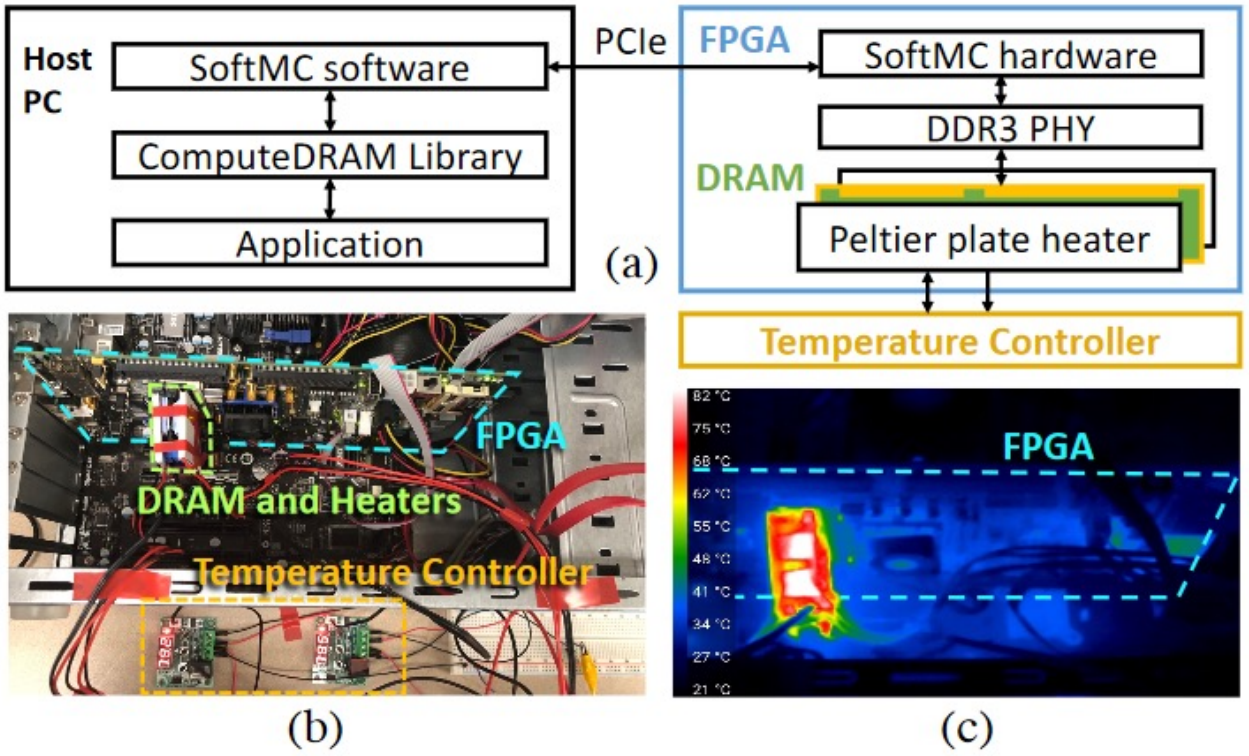


Figure 9: (a) Schematic diagram of our testing framework. (b) Picture of our testbed. (c) Thermal picture when the DRAM is heated to 80 °C.

Experimental Methodology

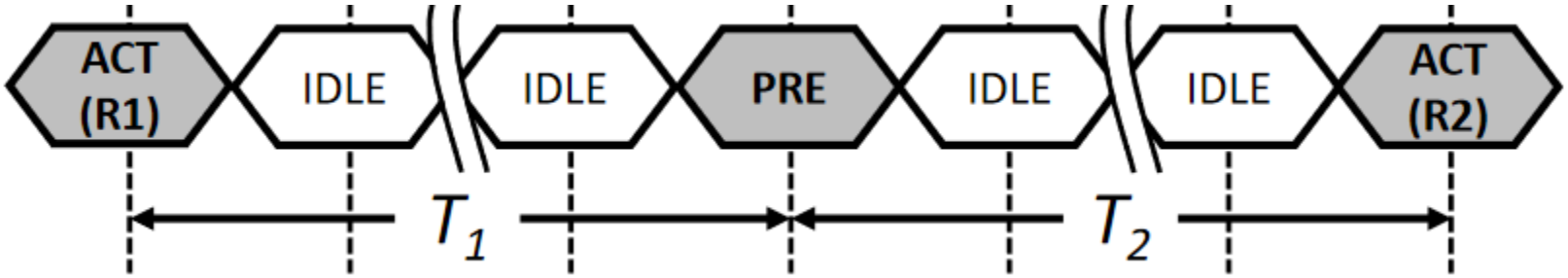
Table 1: Evaluated DRAM modules

Group ID: Vendor_Size_Freq(MHz)	Part Num	# Modules
SKhynix_2G_1333	HMT325S6BFR8C-H9	6
SKhynix_4		2
SKhynix_4		2
SKhynix_4		4
SKhynix_4		2
Samsung_4		2
Samsung_4		2
Micron_2G		2
Micron_2G		2
Elpida_2G_1333	EBJ21UE8BDS0-DJ-F	2
Nanya_4G_1333	NT4GC64B8HG0NS-CG	2
TimeTec_4G_1333	78AP10NUS2R2-4G	2
Corsair_4G_1333	CMSA8GX3M2A1333C9	2

32 DDR3 Modules
~256 DRAM Chips

Proof of Concept

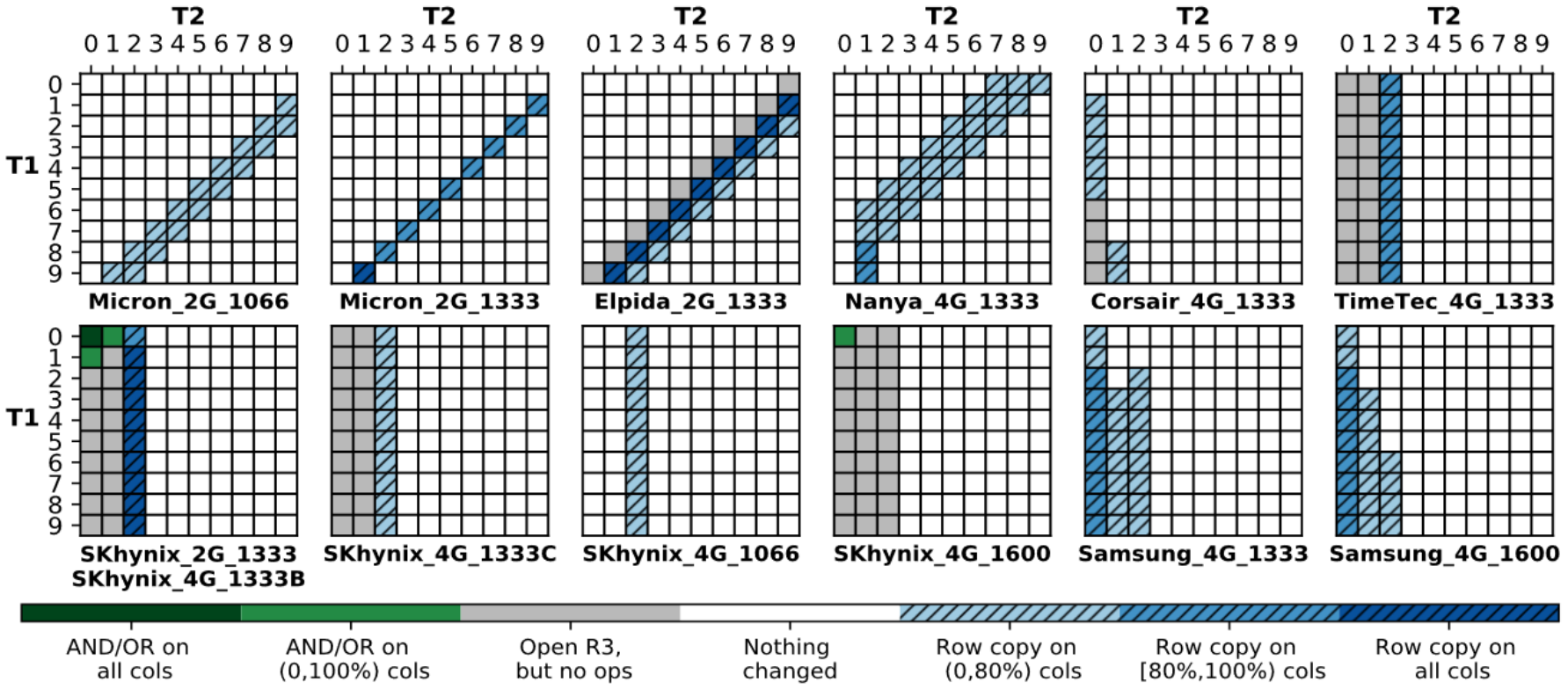
- How they test these memory modules:
  - Vary T_1 and T_2, observe what happens.



SoftMC Experiment

1. Select a random subarray
2. Fill subarray with random data
3. Issue ACT-PRE-ACTs with given T_1 & T_2
4. Read out subarray
5. Find out how many columns in a row support either operation
   - Row-wise success ratio

Proof of Concept



- Each grid represents the success ratio of operations for a specific DDR3 module.

Real Processing Using Memory Prototype

- End-to-end RowClone & TRNG using off-the-shelf DRAM chips
- Idea: Violate DRAM timing parameters to mimic RowClone

PiDRAM: A Holistic End-to-end FPGA-based Framework for Processing-in-DRAM

Ataberk Olgun[§†] Juan Gómez Luna[§] Konstantinos Kanellopoulos[§] Behzad Salami[§*]
Hasan Hassan[§] Oğuz Ergin[†] Onur Mutlu[§]

[§]ETH Zürich [†]TOBB ETÜ [*]BSC

https://arxiv.org/pdf/2111.00082.pdf

https://github.com/cmu-safari/pidram

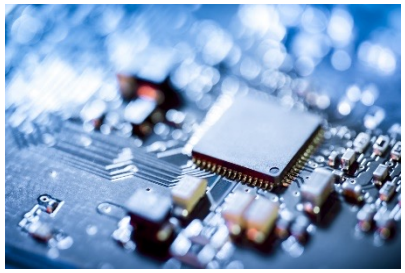
https://www.youtube.com/watch?v=qeukNs5XI3g&t=4192s

PiDRAM

Goal: Develop a **flexible** platform to explore **end-to-end** implementations of PuM techniques

- Enable rapid integration via key components

Hardware

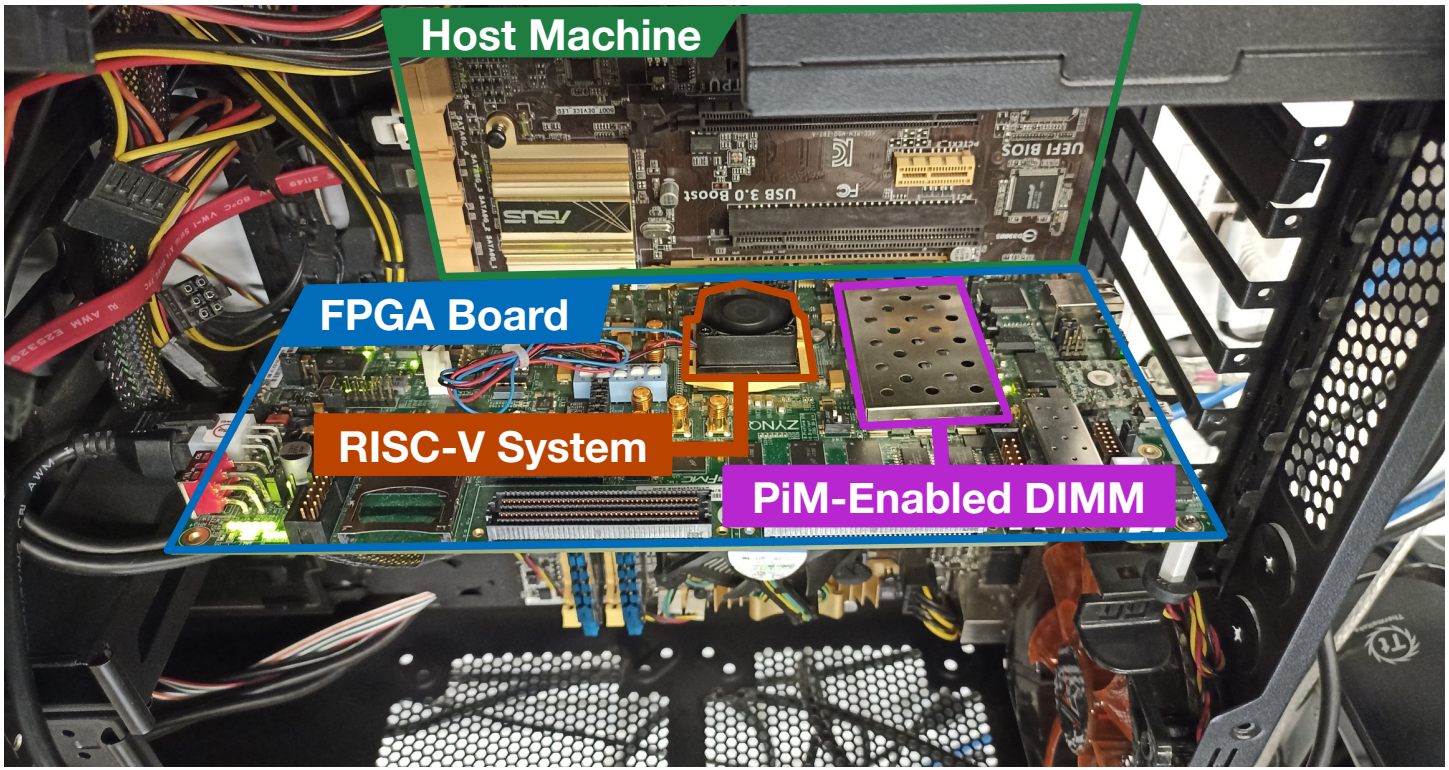


1. Easy-to-extend Memory Controller
2. ISA-transparent PuM Controller

Software

1. Extensible Software Library
2. Custom Supervisor Software

Real Processing Using Memory Prototype

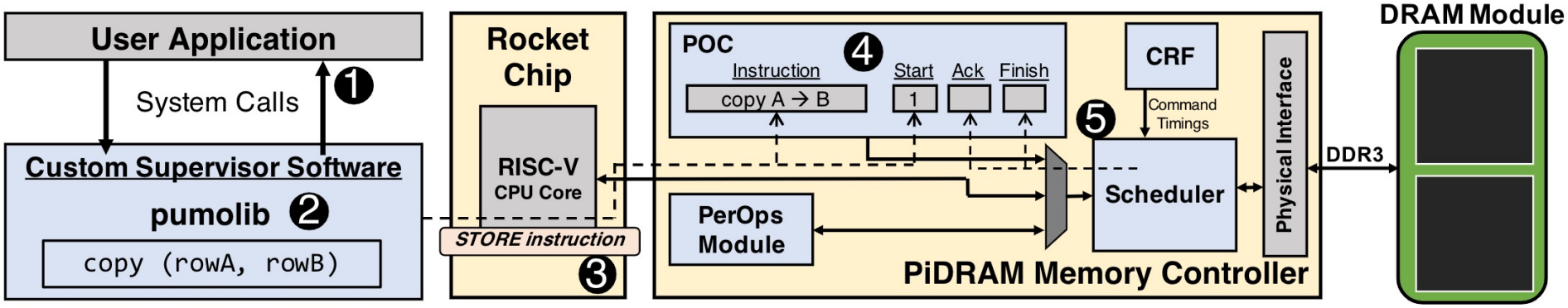


https://arxiv.org/pdf/2111.00082.pdf

https://github.com/cmu-safari/pidram

https://www.youtube.com/watch?v=qeukNs5XI3g&t=4192s

PiDRAM Workflow



1- User application interfaces with the OS via system calls

2- OS uses PuM Operations Library (pumolib) to convey operation related information to the hardware *using*

3- STORE instructions that target the memory mapped registers of the PuM Operations Controller (POC)

4- POC oversees the execution of a PuM operation (e.g., RowClone, bulk bitwise operations)

5- Scheduler arbitrates between regular (load, store) and PuM operations and issues DRAM commands with custom timings

Real Processing Using Memory Prototype

README.md

Building a PiDRAM Prototype

To build PiDRAM's prototype on Xilinx ZC706 boards, developers need to use the two sub-projects in this directory. `fpga-zynq` is a repository branched off of UCB-BAR's fpga-zynq repository. We use `fpga-zynq` to generate rocket chip designs that support end-to-end DRAM PuM execution. `controller-hardware` is where we keep the main Vivado project and Verilog sources for PiDRAM's memory controller and the top level system design.

Rebuilding Steps

1. Navigate into `fpga-zynq` and read the README file to understand the overall workflow of the repository
   - Follow the readme in `fpga-zynq/rocket-chip/riscv-tools` to install dependencies
2. Create the Verilog source of the rocket chip design using the `ZynqCopyFPGAConfig`
   - Navigate into zc706, then run `make rocket CONFIG=ZynqCopyFPGAConfig -j<number of cores>`
3. Copy the generated Verilog file (should be under zc706/src) and overwrite the same file in `controller-hardware/source/hdl/impl/rocket-chip`
4. Open the Vivado project in `controller-hardware/Vivado_Project` using Vivado 2016.2
5. Generate a bitstream
6. Copy the bitstream (system_top.bit) to `fpga-zynq/zc706`
7. Use the `./build_script.sh` to generate the new `boot.bin` under `fpga-images-zc706`, you can use this file to program the FPGA using the SD-Card
   - For details, follow the relevant instructions in `fpga-zynq/README.md`

You can run programs compiled with the RISC-V Toolchain supplied within the `fpga-zynq` repository. To install the toolchain, follow the instructions under `fpga-zynq/rocket-chip/riscv-tools`.

Generating DDR3 Controller IP sources

We cannot provide the sources for the Xilinx PHY IP we use in PiDRAM's memory controller due to licensing issues. We describe here how to regenerate them using Vivado 2016.2. First, you need to generate the IP RTL files:

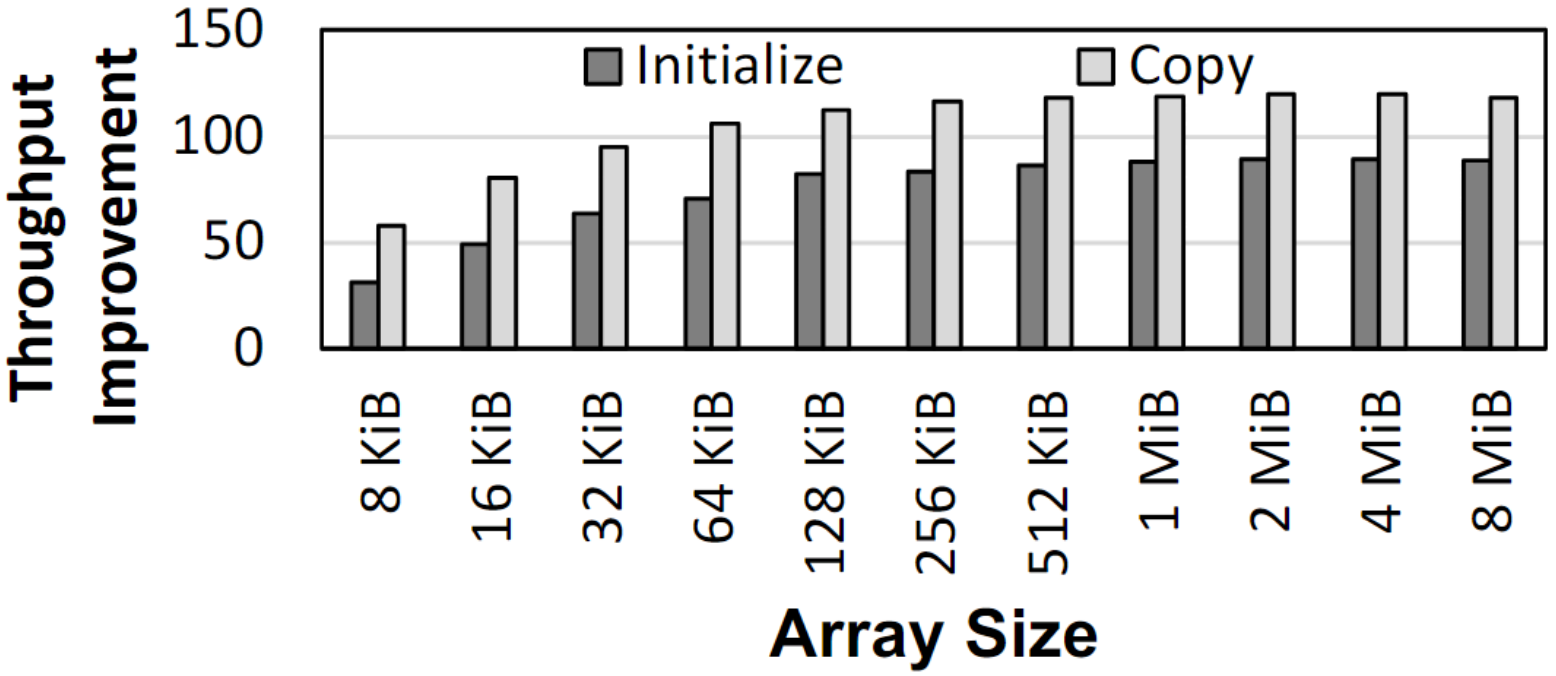
1- Open IP Catalog
2- Find "Memory Interface Generator (MIG 7 Series)" IP and double click

https://arxiv.org/pdf/2111.00082.pdf

https://github.com/cmu-safari/pidram

https://www.youtube.com/watch?v=qeukNs5XI3g&t=4192s

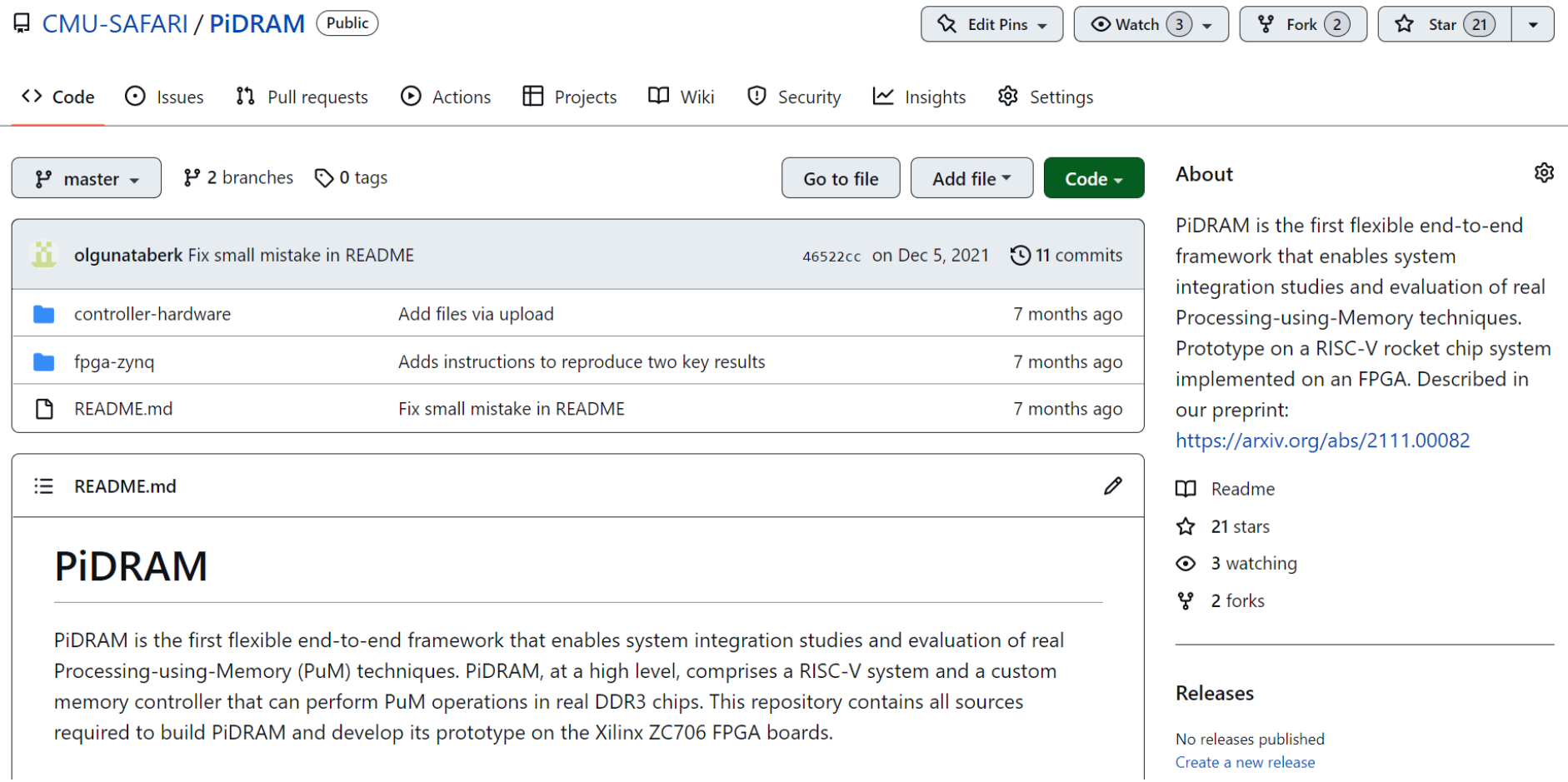
Microbenchmark Copy/Initialization Throughput



In-DRAM Copy and Initialization improve throughput by 119x and 89x

PiDRAM is Open Source

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



Extended Version on ArXiv

https://arxiv.org/abs/2111.00082

Computer Science > Hardware Architecture

[Submitted on 29 Oct 2021 (v1), last revised 19 Dec 2021 (this version, v3)]

PiDRAM: A Holistic End-to-end FPGA-based Framework for Processing-in-DRAM

Ataberk Olgun, Juan Gómez Luna, Konstantinos Kanellopoulos, Behzad Salami, Hasan Hassan, Oğuz Ergin, Onur Mutlu

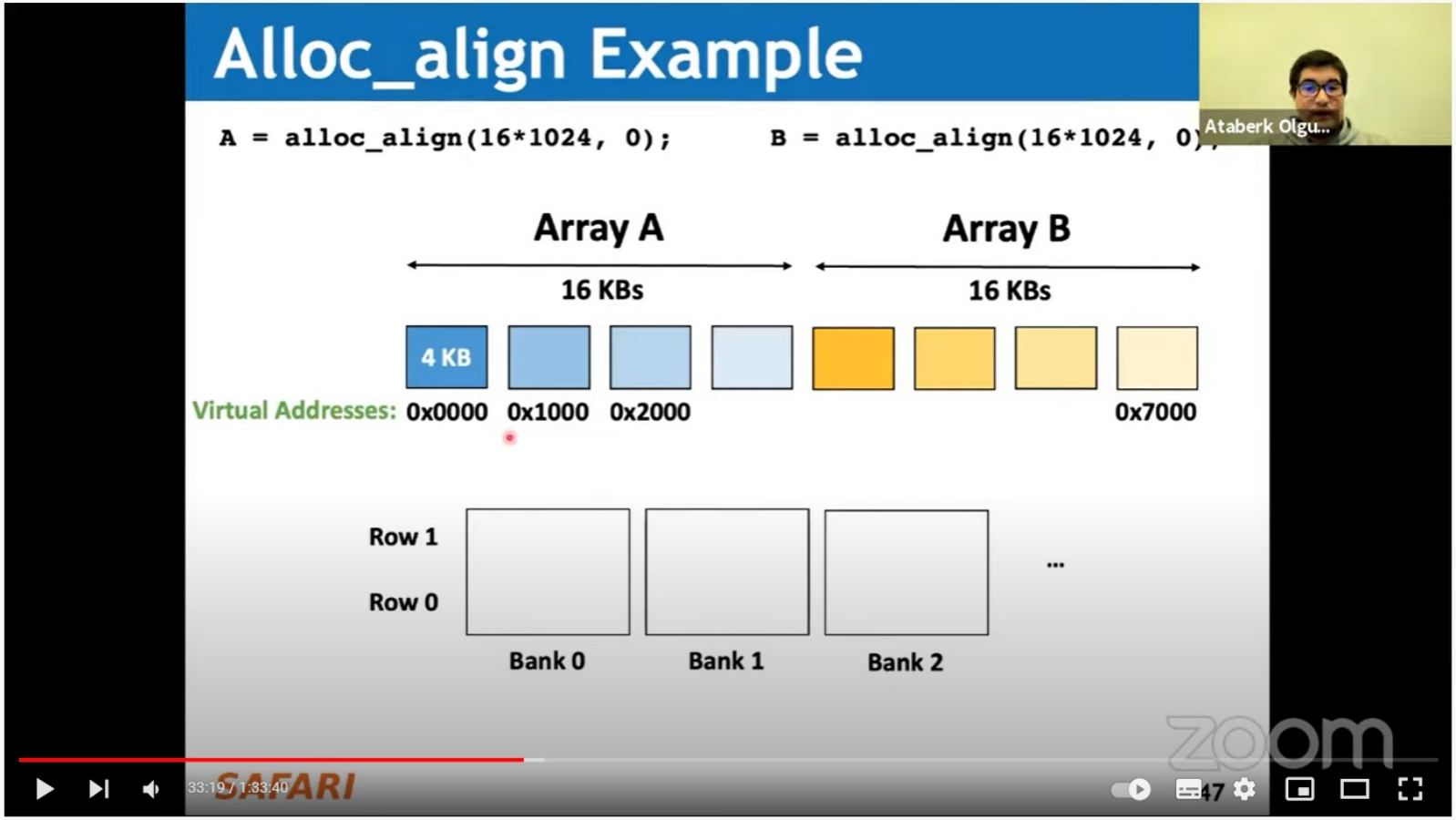
Processing-using-memory (PuM) techniques leverage the analog operation of memory cells to perform computation. Several recent works have demonstrated PuM techniques in off-the-shelf DRAM devices. Since DRAM is the dominant memory technology as main memory in current computing systems, these PuM techniques represent an opportunity for alleviating the data movement bottleneck at very low cost. However, system integration of PuM techniques imposes non-trivial challenges that are yet to be solved. Design space exploration of potential solutions to the PuM integration challenges requires appropriate tools to develop necessary hardware and software components. Unfortunately, current specialized DRAM-testing platforms, or system simulators do not provide the flexibility and/or the holistic system view that is necessary to deal with PuM integration challenges.

We design and develop PiDRAM, the first flexible end-to-end framework that enables system integration studies and evaluation of real PuM techniques. PiDRAM provides software and hardware components to rapidly integrate PuM techniques across the whole system software and hardware stack (e.g., necessary modifications in the operating system, memory controller). We implement PiDRAM on an FPGA-based platform along with an open-source RISC-V system. Using PiDRAM, we implement and evaluate two state-of-the-art PuM techniques: in-DRAM (i) copy and initialization, (ii) true random number generation. Our results show that the in-memory copy and initialization techniques can improve the performance of bulk copy operations by 12.6x and bulk initialization operations by 14.6x on a real system. Implementing the true random number generator requires only 190 lines of Verilog and 74 lines of C code using PiDRAM's software and hardware components.

Comments: 15 pages, 12 figures
Subjects: **Hardware Architecture (cs.AR)**
Cite as: arXiv:2111.00082 **[cs.AR]**
(or arXiv:2111.00082v3 **[cs.AR]** for this version)
https://doi.org/10.48550/arXiv.2111.00082

Long Talk + Tutorial on Youtube

https://youtu.be/s_z_S6FYpC8



Processing in Memory Course: Meeting 6: End-to-end Framework for Processing-using-Memory - Fall'21

615 views • Streamed live on 9 Nov 2021 • Project & Seminar, ETH Zürich, Fall 202 Show more

25 Dislike Share Download Clip Save ...

Onur Mutlu Lectures
25.7K subscribers

SUBSCRIBED

Processing using Memory in Other Memory Technologies

Pinatubo: RowClone and Bitwise Ops in PCM

Pinatubo: A Processing-in-Memory Architecture for Bulk Bitwise Operations in Emerging Non-volatile Memories

Shuangchen Li[1]*, Cong Xu[2], Qiaosha Zou[1,5], Jishen Zhao[3], Yu Lu[4], and Yuan Xie[1]

University of California, Santa Barbara[1], Hewlett Packard Labs[2]
University of California, Santa Cruz[3], Qualcomm Inc.[4], Huawei Technologies Inc.[5]
{shuangchenli, yuanxie}ece.ucsb.edu[1]

https://cseweb.ucsd.edu/~jzhao/files/Pinatubo-dac2016.pdf

Pinatubo: RowClone and Bitwise Ops in PCM

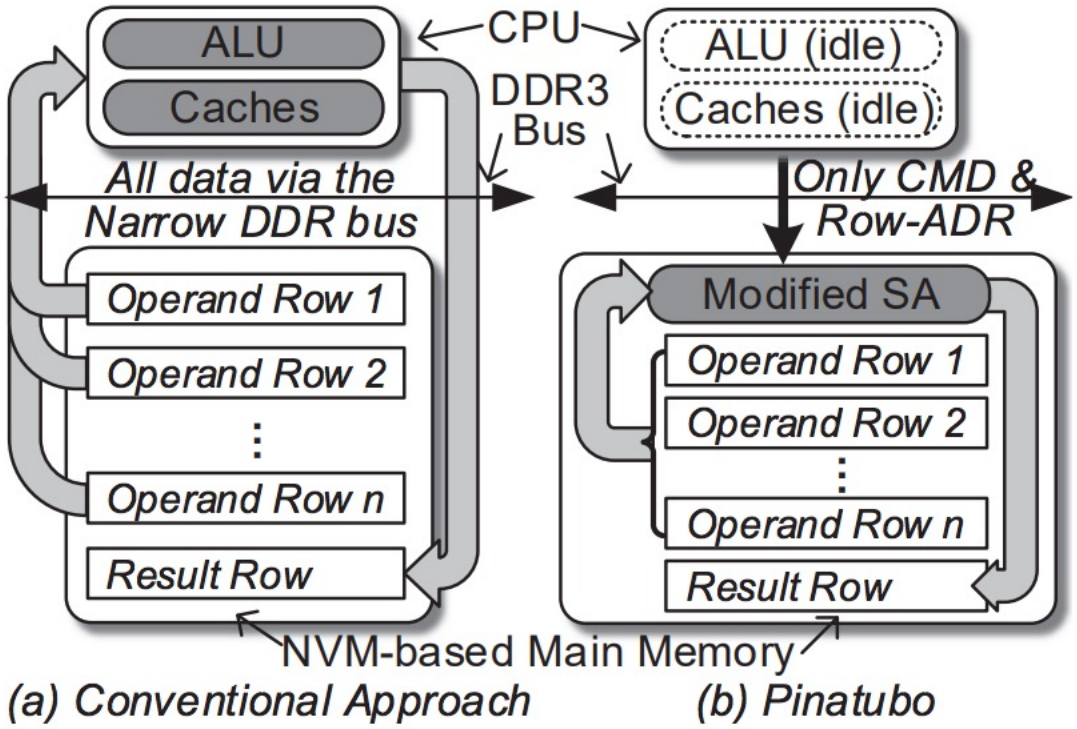


Figure 2: Overview: (a) Computing-centric approach, moving tons of data to CPU and write back. (b) The proposed Pinatubo architecture, performs n-row bitwise operations inside NVM in one step.

In-Flash Bulk Bitwise Execution

- Jisung Park, Roknoddin Azizi, Geraldo F. Oliveira, Mohammad Sadrosadati, Rakesh Nadig, David Novo, Juan Gómez-Luna, Myungsuk Kim, and Onur Mutlu,
  "Flash-Cosmos: In-Flash Bulk Bitwise Operations Using Inherent Computation Capability of NAND Flash Memory"
  *Proceedings of the 55th International Symposium on Microarchitecture (**MICRO**)*, Chicago, IL, USA, October 2022.
  [Slides (pptx) (pdf)]
  [Longer Lecture Slides (pptx) (pdf)]
  [Lecture Video (44 minutes)]
  [arXiv version]

Flash-Cosmos: In-Flash Bulk Bitwise Operations Using Inherent Computation Capability of NAND Flash Memory

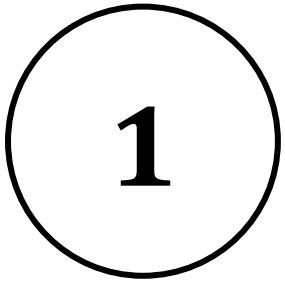
Jisung Park[§∇] Roknoddin Azizi[§] Geraldo F. Oliveira[§] Mohammad Sadrosadati[§]
Rakesh Nadig[§] David Novo[†] Juan Gómez-Luna[§] Myungsuk Kim[‡] Onur Mutlu[§]

[§]*ETH Zürich* [∇]*POSTECH* [†]*LIRMM, Univ. Montpellier, CNRS* [‡]*Kyungpook National University*

https://arxiv.org/pdf/2209.05566.pdf

NAND Flash Basics: A Flash Cell

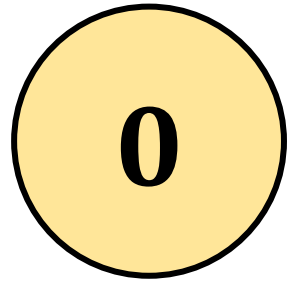
- **A flash cell stores data by adjusting the amount of charge in the cell**



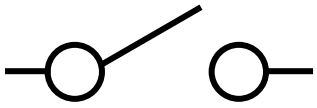
Erased Cell
(Low Charge Level)

Operates as a resistor

Activation



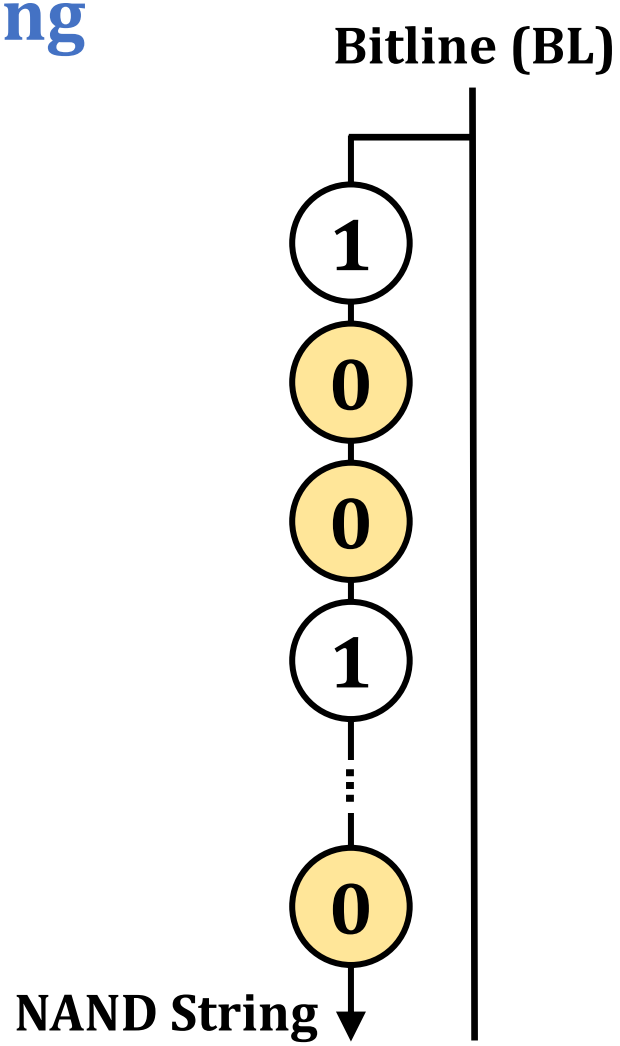
Programmed Cell
(High Charge Level)



Operates as an open switch

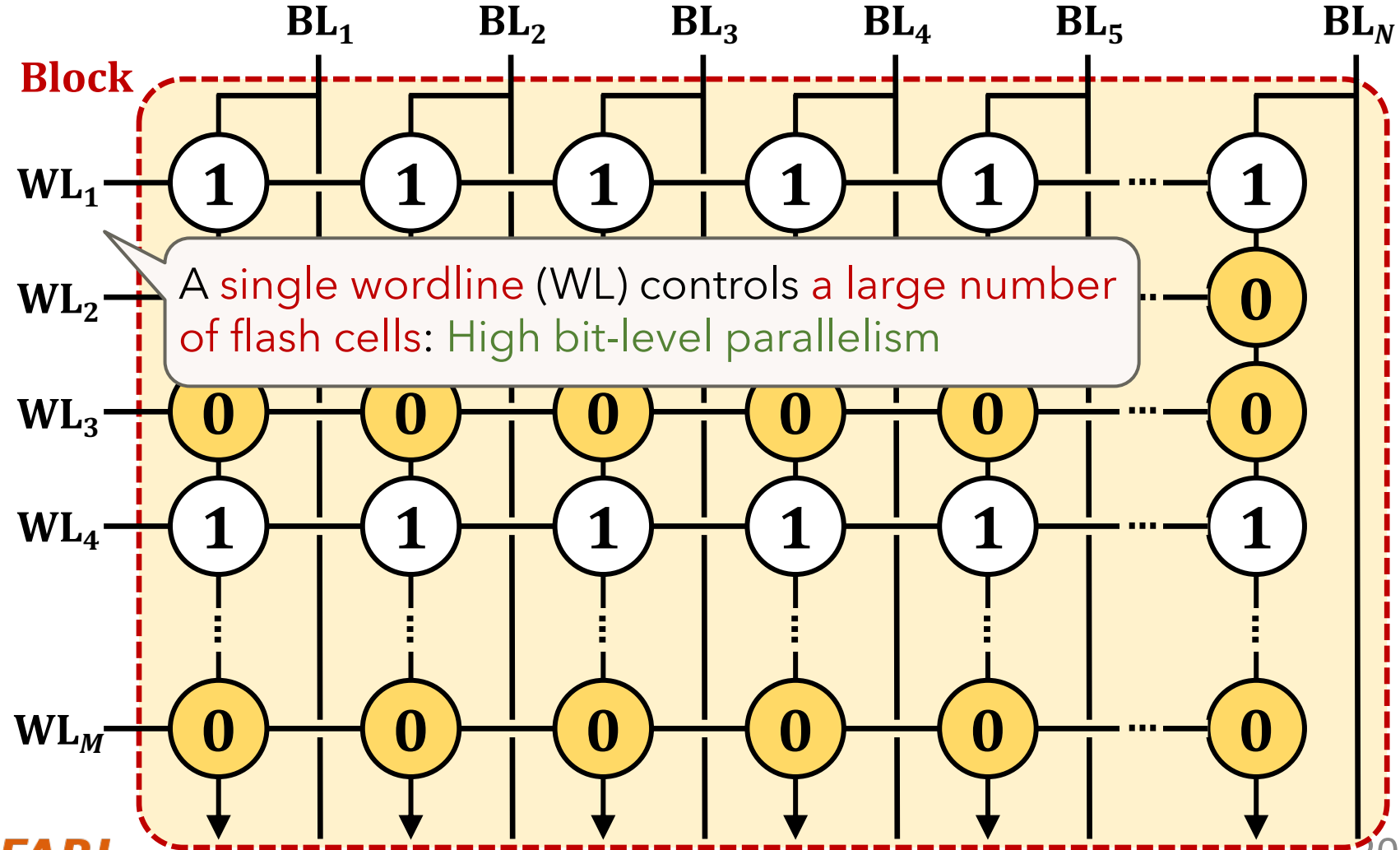
NAND Flash Basics: A NAND String

- A set of flash cells are serially connected to form a NAND String



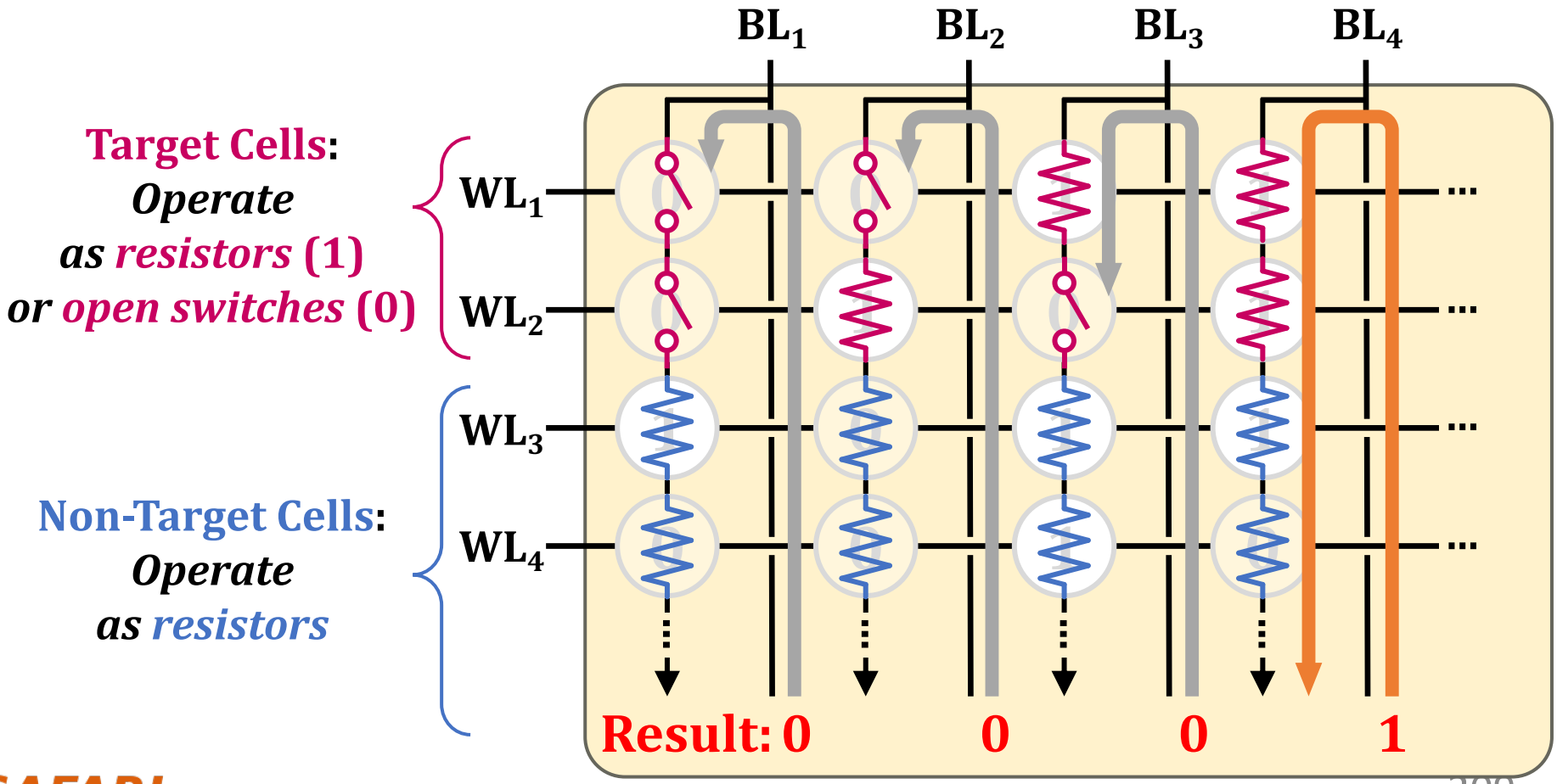
NAND Flash Basics: A NAND Flash Block

- **NAND strings connected to different bitlines comprise a NAND block**



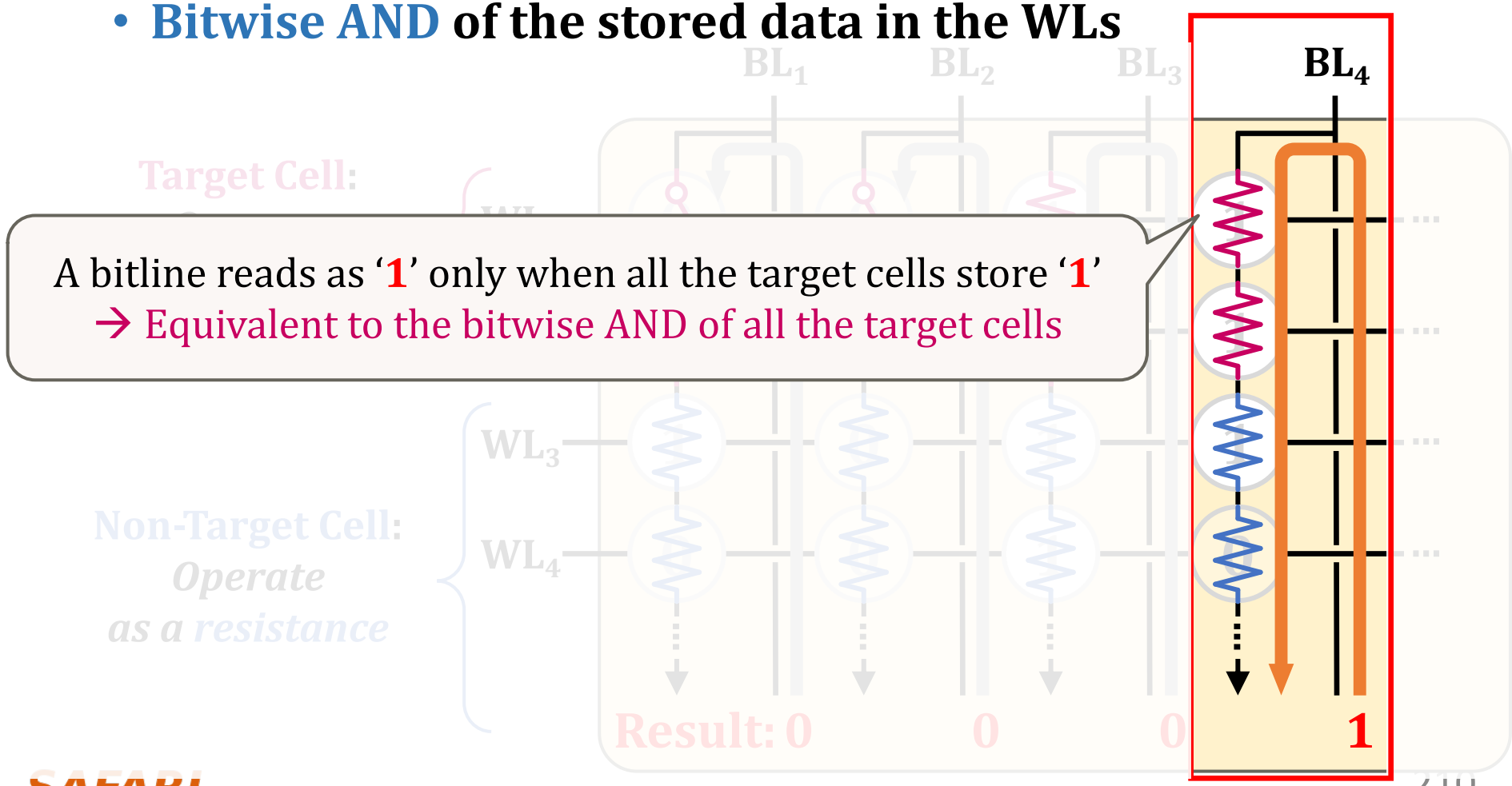
Multi-Wordline Sensing (MWS): Bitwise AND

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



Multi-Wordline Sensing (MWS): Bitwise AND

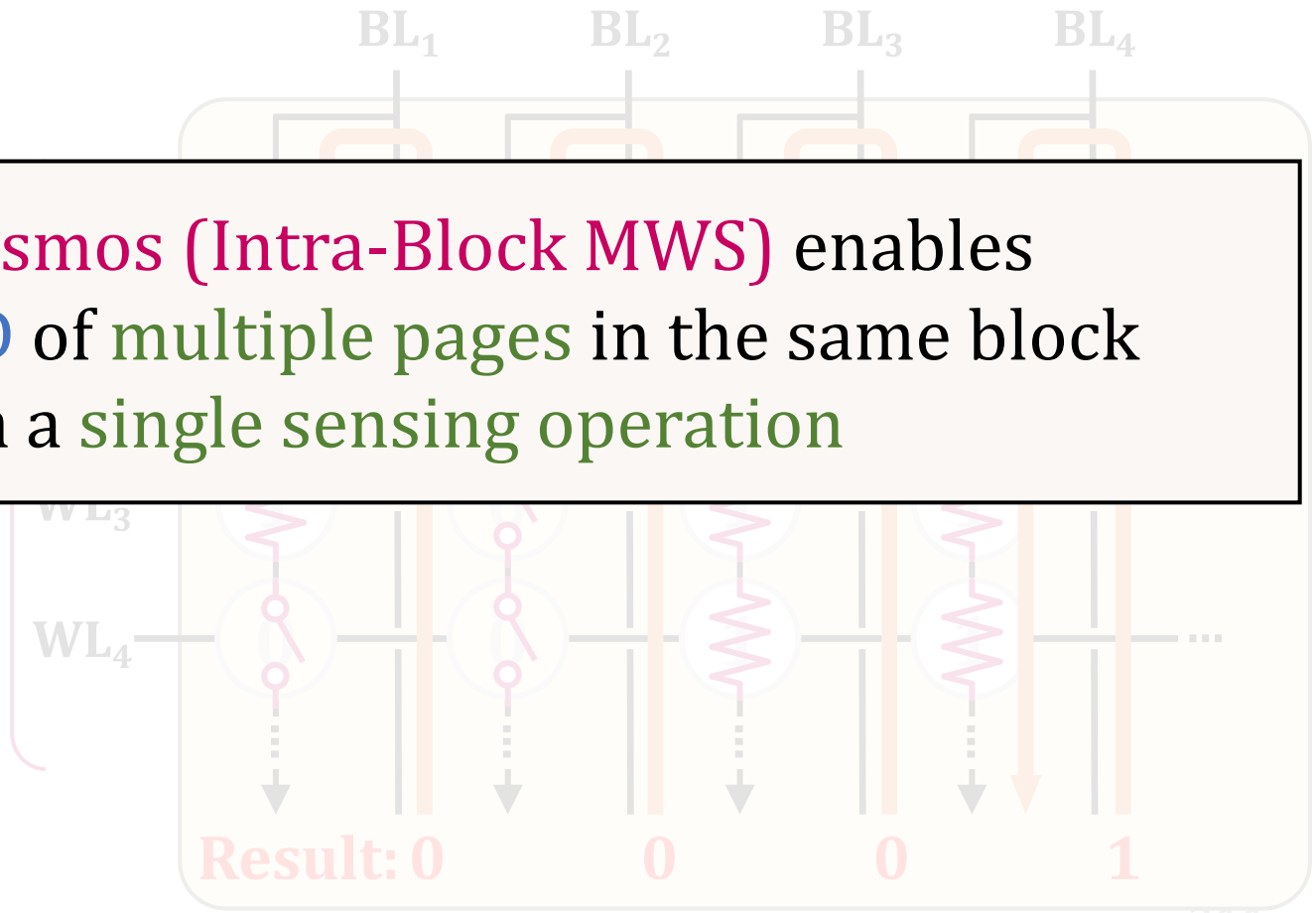
- **Intra-Block MWS: Simultaneously activates multiple WLs in the same block**
  - **Bitwise AND of the stored data in the WLs**



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

Multi-Wordline Sensing (MWS): Bitwise AND

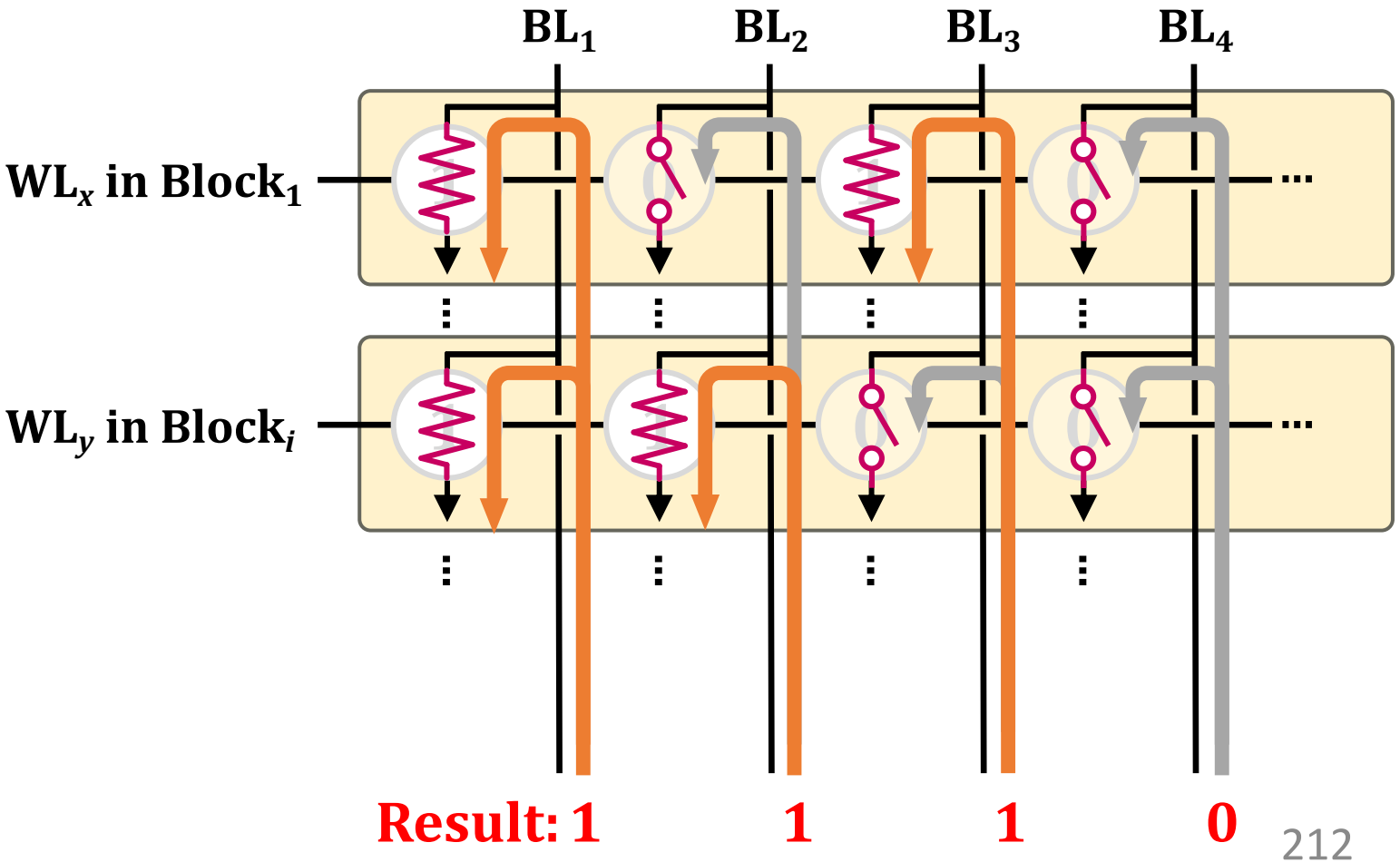
- **Intra-Block MWS**: Simultaneously activates multiple WLs in the same block
  - Bitwise AND of the stored data in the WLs



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

Multi-Wordline Sensing (MWS): Bitwise OR

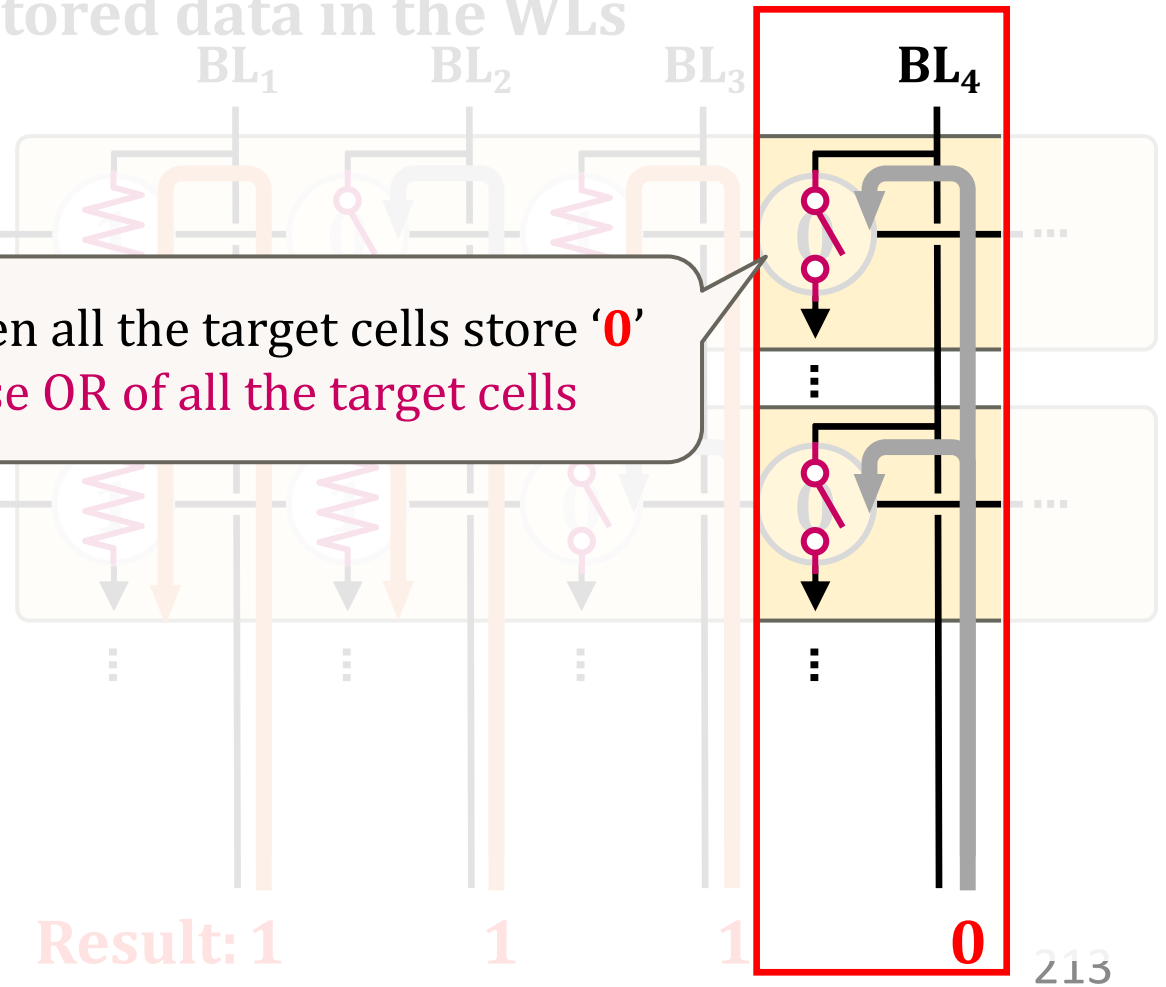
- **Inter-Block MWS**: **Simultaneously activates** **multiple WLs in different blocks**
  - **Bitwise OR** **of the stored data in the WLs**



Multi-Wordline Sensing (MWS): Bitwise OR

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

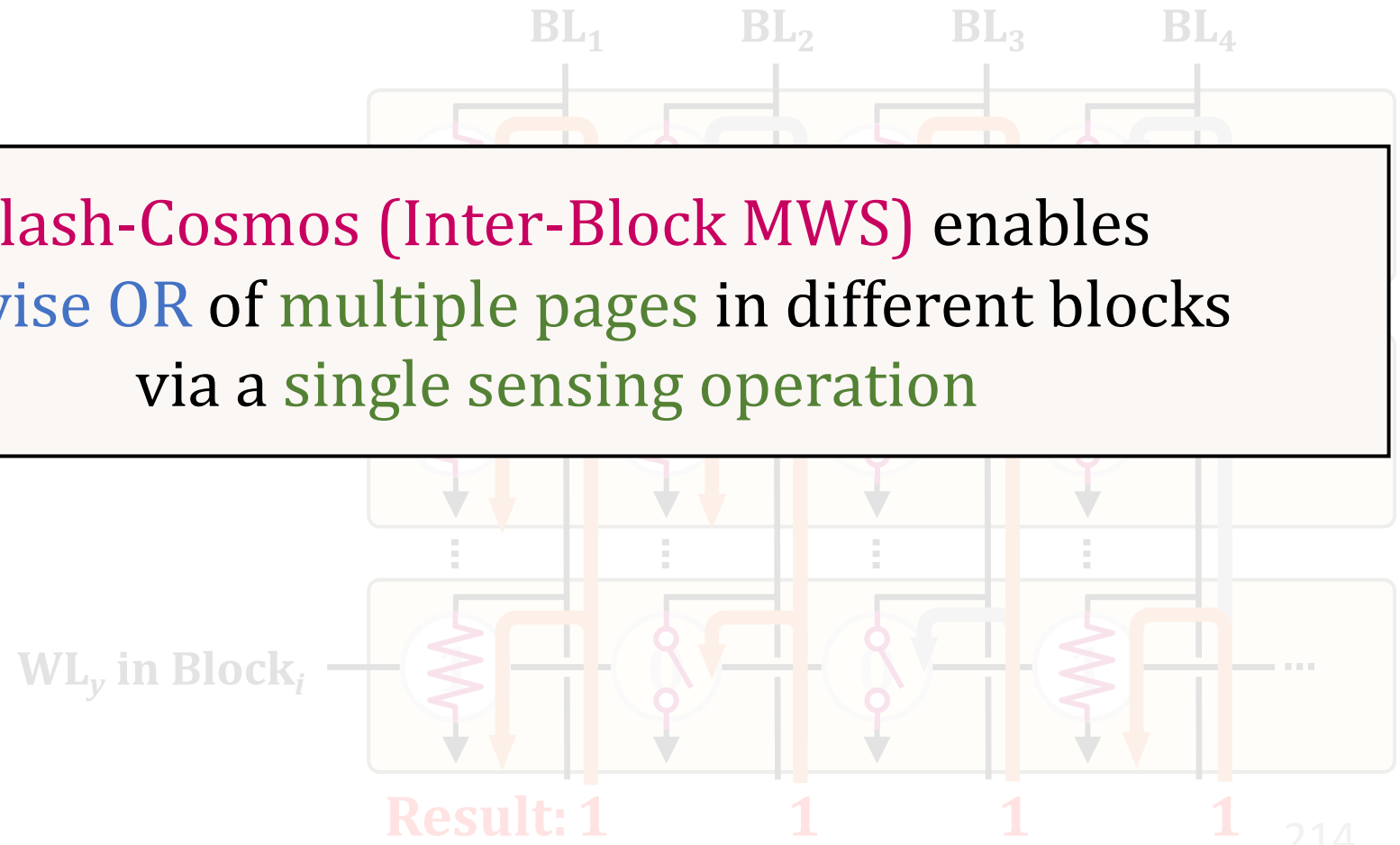
A bitline reads as '**0**' only when all the target cells store '**0**'
→ Equivalent to the bitwise OR of all the target cells



Multi-Wordline Sensing (MWS): Bitwise OR

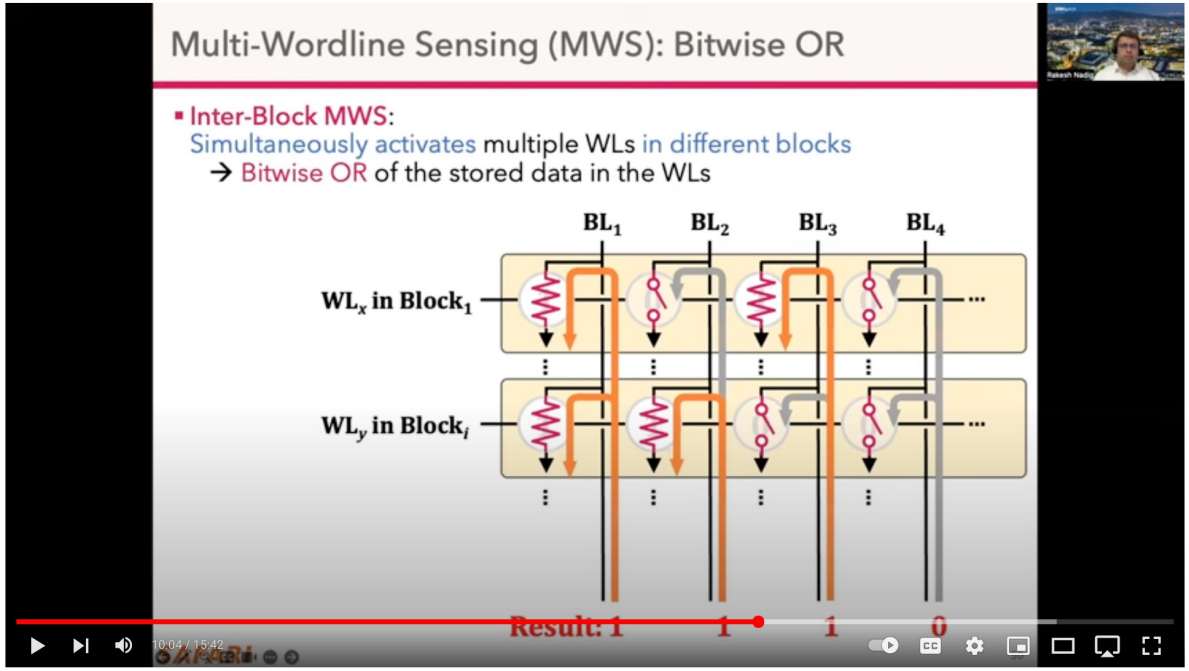
- **Inter-Block MWS**: Simultaneously activates multiple WLs in different blocks
  - **Bitwise OR** of the stored data in the WLs

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



SRC TECHCON Presentation

- Rakesh Nadig
  - Flash-Cosmos: In-Flash Bulk Bitwise Operations Using Inherent Computation Capability of NAND Flash Memory
  - https://arxiv.org/pdf/2209.05566.pdf



Flash-Cosmos: In-Flash Bulk Bitwise Operations, SRC TECHCON 2023

Onur Mutlu Lectures
35.5K subscribers
Subscribed
12
Share
Clip
Save

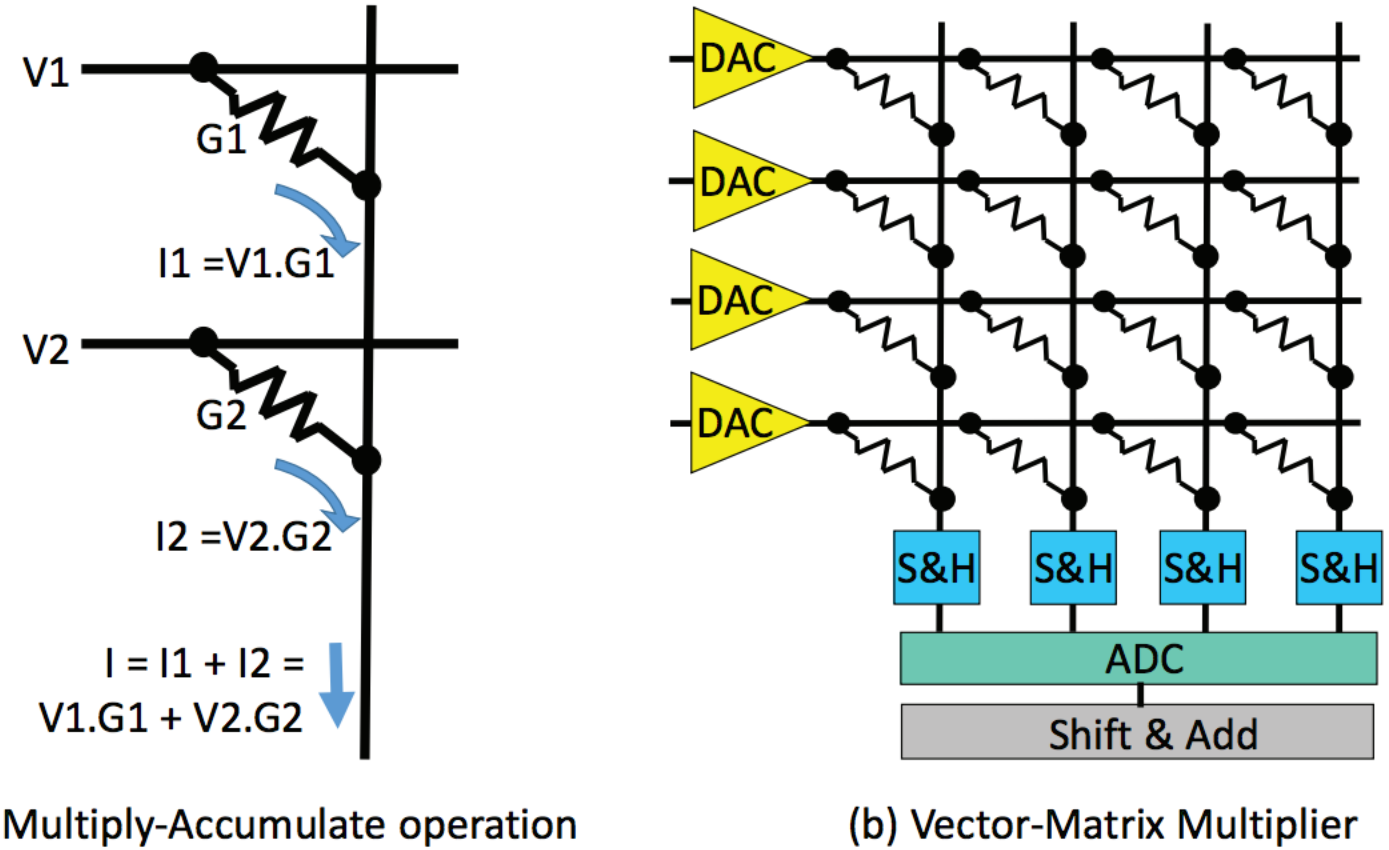
143 views 8 days ago
Flash-Cosmos: In-Flash Bulk Bitwise Operations Using Inherent Computation Capability of NAND Flash Memory
Speaker: Rakesh Nadig ...more

https://youtu.be/QWL2Rw_VhHg?si=aavfsq5UtM-vjcm8

In-Memory Crossbar Array Operations

- Some emerging NVM technologies have crossbar array structure
  - Memristors, resistive RAM, phase change mem, STT-MRAM, …
- Crossbar arrays can be used to perform dot product operations using "analog computation capability"
  - Can operate on multiple pieces of data using Kirchoff's laws
    - Bitline current is a sum of products of wordline V x (1 / cell R)
  - Computation is in analog domain inside the crossbar array
- Need peripheral circuitry for D→A and A→D conversion of inputs and outputs

Aside: In-Memory Crossbar Computation

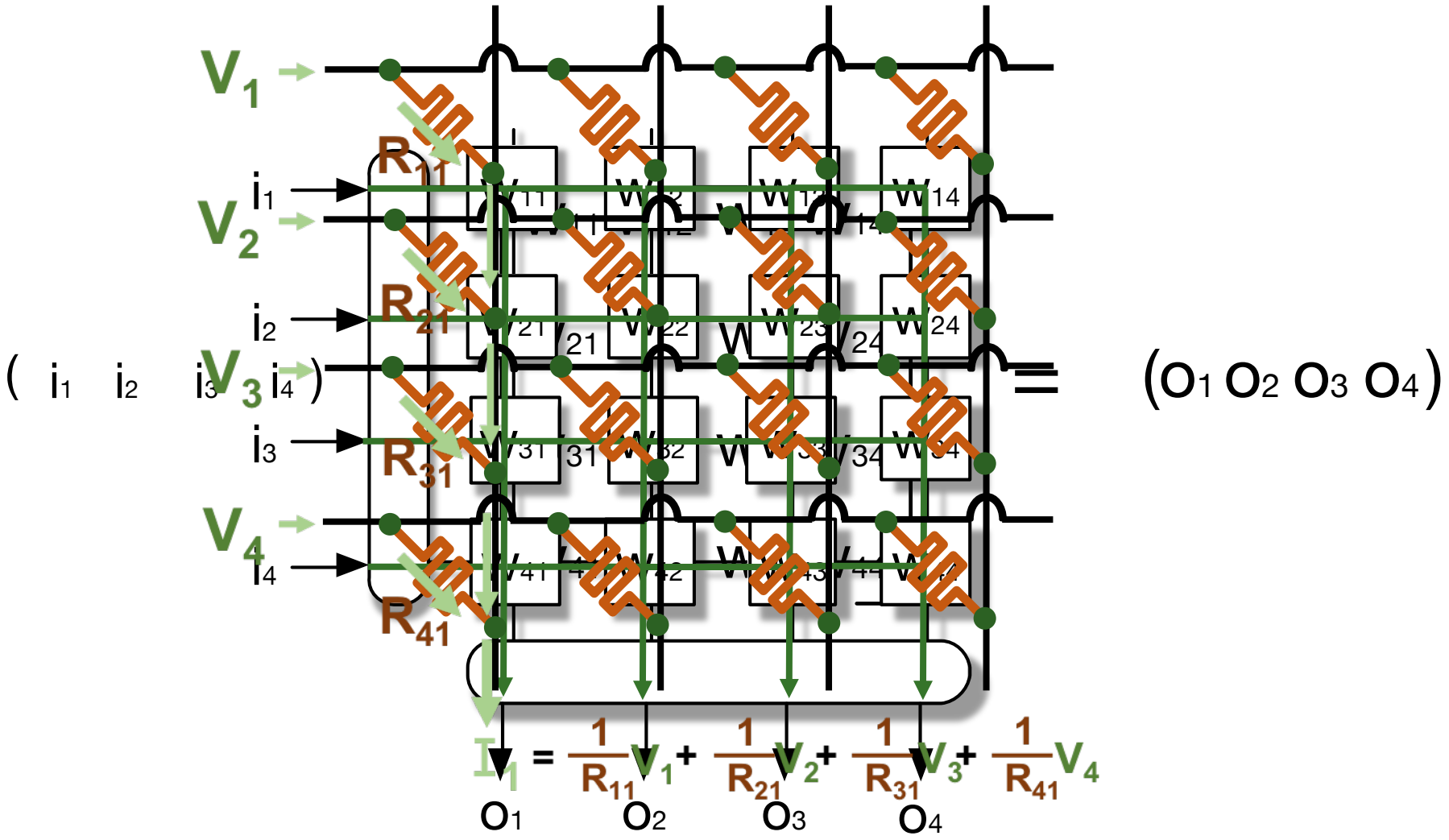


(a) Multiply-Accumulate operation

(b) Vector-Matrix Multiplier

Fig. 1. (a) Using a bitline to perform an analog sum of products operation. (b) A memristor crossbar used as a vector-matrix multiplier.

Aside: In-Memory Crossbar Computation



Readings on Processing using NVM

- Shafiee+, "ISAAC: A Convolutional Neural Network Accelerator with In-Situ Analog Arithmetic in Crossbars", ISCA 2016.
- Chi+, "PRIME: A Novel Processing-in-memory Architecture for Neural Network Computation in ReRAM-based Main Memory", ISCA 2016.
- Prezioso+, "Training and Operation of an Integrated Neuromorphic Network based on Metal-Oxide Memristors", Nature 2015
- Ambrogio+, "Equivalent-accuracy accelerated neural-network training using analogue memory", Nature 2018.

Challenge: Intelligent Memory Device

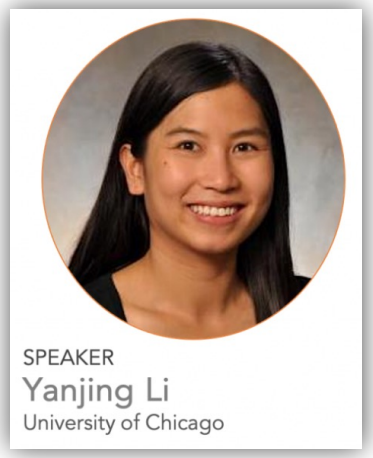
Does memory have to be dumb?

Computing Architectures with Minimal Data Movement

Processing-in-Memory: Summary

- Data movement is a major bottleneck in modern systems, leading to performance and energy drawbacks
- Processing-in-Memory: perform computation where the data resides
  - Processing near memory: add logic elements near memory
  - Processing using memory: use analog properties of memory to perform computation
- Adopting processing-in-memory requires changes to the entire stack
  - All can be solved with a change of mindset

SAFARI Live Seminar: Yanjing Li – 17.10



- **SAFARI Live Seminar**: Yanjing Li, Oct. 17 2023
  - Assistant Professor in the Department of Computer Science at the University of Chicago

- **Title:** How does one bit-flip corrupt an entire deep neural network, and what to do about it
  - In-depth resilience study targeting DNN workloads and hardware failures of deep learning accelerator systems
  - Lightweight yet highly effective techniques to mitigate hardware failures in deep learning accelerator systems.

- **Where:** Livestream on YouTube at 6 p.m.
  - https://www.youtube.com/watch?v=ZHBMbSPcddo

Computer Architecture

Lecture 5: Processing using Memory

Geraldo F. Oliveira
Prof. Onur Mutlu
ETH Zürich

Fall 2023

12 October 2023

Historical Perspective & A Detour on the Review Process

Ambit and RowClone Sound Great! No?

Some History: RowClone

RowClone: Historical Perspective

- This work is likely the first example of "minimally changing DRAM chips" to perform data movement and computation
  - Surprising that it was done as late as 2013!
- It led to a body of work on in-DRAM (and in-NVM) computation with "hopefully small" changes
- Work building on RowClone still continues
- Initially, it was dismissed by some reviewers
  - Rejected from ISCA 2013 conference

One Review (ISCA 2013 Submission)

Paper Strengths

The paper includes a well written background on DRAM organization/operation. The proposed technique is simple and elegant; it
nicely exploits key circuit-level characteristics of DRAM designs and
minimizes the changes necessary to commodity DRAM chips.

Paper Weaknesses

I am concerned on the applicability of the technique and found the
evaluation to be uncompelling in terms of motivating the work as well as
quantifying the potential benefit. Details on how to efficiently manage
the coherence between the cache hierarchy and DRAM to enable the proposed
technique are glossed over, but in my opinion are critical to the
narrative.

Another Review and Rebuttal

DETAILED COMMENTS

The paper proposes a simple and not new idea, block copy in a DRAM, and the creates a complete

Reviewer B mentions that our idea is "not new". An explicit
reference by the reviewer would be helpful here. While the
reviewer may be referring to one of the patents that we cite in
our paper (citations 2, 6, 25, 26, 27 in the paper), these patents
are at a superficial level and do *not* provide a concrete
mechanism. In contrast, we propose three concrete mechanisms and
provide details on the most important architectural and
microarchitectural modifications required at the DRAM chip, the
memory controller, and the CPU to enable a system that supports
the mechanisms. We also analyze their latency, hardware overhead,
power, and performance in detail. We are not aware of any prior
work that achieves this.

ISCA 2013 Submission

ISCA40 | **Paper #295**

onur@cmu.edu Profile | Help | Sign out

Main Edit

#268 Papers #353 (All) Search

#295 RowClone: Fast and Efficient In-DRAM Copy and Initialization of Bulk Data

COMMENT

NOTIFICATION

If selected, you will receive email when updated comments are available for this paper.

+ OTHER CONFLICTS

Rejected 1014kB Thursday 22 Nov 2012 12:11:45am EST | 0fd459a9adc6194cda028a394d2e4d929f662f32

You are an **author** of this paper.

– ABSTRACT

Many programs initialize or copy large amounts of memory data. Initialization and copying are

+ AUTHORS

V. Seshadri, Y. Kim, D. Lee, C. Fallin, R. Ausavarungnirun, G. Pekhimenko, Y. Luo, O. Mutlu,

	OveMer	Nov	WriQua	RevConAnd
Review #295A	3	4	5	3
Review #295B	4	3	4	3
Review #295C	3	4	4	3

Yet Later… in ISCA 2015…

Profiling a warehouse-scale computer

Svilen Kanev† Harvard University

Juan Pablo Darago† Universidad de Buenos Aires

Kim Hazelwood† Yahoo Labs

Parthasarathy Ranganathan Google

Tipp Moseley Google

Gu-Yeon Wei Harvard University

David Brooks Harvard University

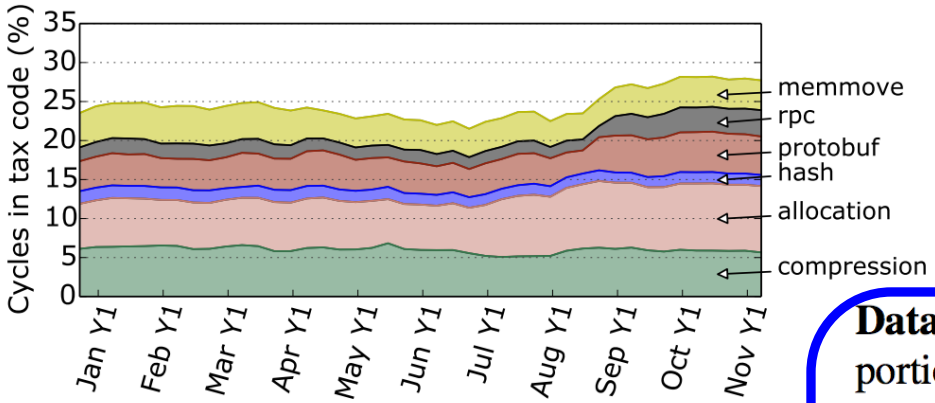


Figure 4: 22-27% of WSC cycles are spent in different cc nents of "datacenter tax".

we see common building blocks once we aggregate sampled profile data across many applications running in a datacenter. In this section, we quantify the performance impact of the **datacenter tax**, and argue that its components are prime candidates for hardware acceleration in future datacenter SoCs.

Data movement In fact, RPCs are by far not the only code portions that do data movement. We also tracked all calls to the `memcpy()` and `memmove()` library functions to estimate the amount of time spent on explicit data movement (i.e., exposed through a simple API). This is a conservative estimate because it does not track inlined or explicit copies. Just the variants of these two library functions represent 4-5% of datacenter cycles.

Recent work in performing data movement in DRAM [45] could optimize away this piece of tax.

MICRO 2013 Submission

#206 RowClone: Fast and Energy-Efficient In-DRAM Bulk Data Copy and Initialization

ATION
ceive
e for

Accepted 1947kB Friday 31 May 2013 1:48:46pm PDT |
fd8423acdd9a222280302355899340083e5a40b1

You are an **author** of this paper.

+ Abstract

Bulk data copy and initialization operations are frequently triggered by several system level operations in modern systems. Despite the fact that these operations do not require [more]

+ Authors

V. Seshadri, Y. Kim, C. Fallin, D. Lee, R. Ausavarungnirun, G. Pekhimenko, Y. Luo, O. Mutlu, P. Gibbons, M. Kozuch, T. Mowry [details]

+ Topics

	OveMer	Nov	WriQua	RevExp
Review #206A	5	4	4	4
Review #206B	4	2	4	4
Review #206C	3	4	4	4
Review #206D	3	3	4	3
Review #206E	4	3	5	3

More History: Ambit

Ambit

- First work on performing bulk bitwise operations in DRAM
  - By exploiting analog computation capability of bitlines
  - Extends and completes our IEEE CAL 2015 paper
- **Disruptive** -- spans algorithms to circuits/devices
  - Requires hardware/software cooperation for adoption
- Led to a large amount of work in similar approaches in DRAM and NVM
  - The work continues to build
- Initially, it was dismissed by many reviewers
  - Rejected from 4 conferences!

ISCA 2016: Rejected

Buddy RAM: Fast and Efficient Bulk Bitwise Operations Using DRAM

Rejected 2006kB 23 Nov 2015 11:30:23pm EST ·
7f7234da178e644380275ce12a4f539ef45c4418

You are an **author** of this paper.

▶ **Abstract**

Many data structures (e.g., database bitmap indices) rely on fast bitwise operations on large bit vectors to achieve high performance. Unfortunately, the throughput of such bulk [more]

▶ **Authors**

V. Seshadri, D. Lee, T. Mullins, A. Boroumand, J. Kim, M. Kozuch, O. Mutlu, P. Gibbons, T. Mowry [details]

▶ **Topics and Options**

	RelISC	OveMerPos	RevConAnd	Nov	WriQua
Review #171A	3	4	4	2	3
Review #171B	2	4	3	3	4
Review #171C	3	4	4	2	3
Review #171D	3	5	2	2	3
Review #171E	2	3	2	3	3

MICRO 2016: Rejected

Submission (1662kB) 10 Apr 2016 9:32:31pm EDT ·
e518c6a8916109492574858db80a6184fe61ca0c

▶ **Abstract**

Certain widely-used data structures (e.g., bitmap indices) rely on

[more]

▼ **Authors**

Vivek Seshadri (CMU)
<vseshadr@cs.cmu.ed
Donghyuk Lee (NVIDIA Re
<donghyuk1@cmu.edu
Thomas Mullins (Intel)
<thomas.p.mullins@int
Amirali Boroumand (CMU)
Jeremie Kim (CMU)
Michael A. Kozuch (Intel)
<michael.a.kozuch@in
Onur Mutlu (CMU/ETH)
<omutlu@gmail.com>
Phillip B. Gibbons (CMU)
<gibbons@cs.cmu.edu
Todd C. Mowry (CMU) <to

▶ **Topics**

Rejected · You are an **author** of this paper.

	PosRebOve	OveMer	RevExp	Nov	WriQua
Review #249A	2	2	4	3	3
Review #249B	4	4	3	3	5
Review #249C	2	3	4	2	3
Review #249D	5	5	2	3	3
Review #249E	5	5	2	2	3
Review #249F	3	3	3	3	4

HPCA 2017: Rejected

1)~significantly improves the performance of queries in applications that use bitmap indices for fast analytics, and 2)~makes bit vectors more attractive than red-black trees to represent sets. We believe Buddy can trigger programmers to redesign applications to use bitwise operations with the goal of achieving high performance and efficiency.

Rejected · You are an **author** of this paper.

	OveMer	RevExp	WriQua	ExpMet	Nov
Review #119A	1	2	3	2	2
Review #119B	4	1	4	4	3
Review #119C	4	4	4	4	4
Review #119D	3	1	4	4	3
Review #119E	3	2	5	4	4

1 Comment: *Response* (V. Seshadri)

ISCA 2017: Rejected

Rejected

Submission 19 Nov 2016 12:03:02am EST

3eea263e35e53552851cabc5225162776f809eaa

▶ **Abstract**

Bitwise operations are an important component of

[more]

▶ **Authors**

V. Seshadri, D. Lee, T. Mullins, H. Hassan, A. Boroumand, J. Kim, M. Kozuch, O. Mutlu, P. Gibbons, T. Mowry [details]

▶ **Topics and Options**

	PosRebOve	OveMer	Nov	WriQua	RevExp
Review #162A	1	2	2	4	5
Review #162B	2	2	3	3	3
Review #162C	4	4	3	4	4
Review #162D	3	3	3	4	4
Review #162E	4	4	3	4	3

Ambit Sounds Good, No?

Review from ISCA 2016

Paper summary

The paper proposes to extend DRAM to include bulk, bit-wise logical
operations directly between rows within the DRAM.

Strengths

- Very clever/novel idea.

- Great potential speedup and efficiency gains.

Weaknesses

- Probably won't ever be built. Not practical to assume DRAM manufacturers with change DRAM in this way.

Very Interesting and Novel, ….. BUT …

Comments for the authors

I found this idea very interesting and novel. In particular, while there have been many works proposing moving computation closer to
memory, I'm not aware of any work which proposes to leverage the DRAM
rows themselves to implement the computation. The benefits to this
approach are large in that no actual logic is used to implement the
logical functions. Further the operation occurs in parallel across
the whole row, a huge degree of data parallelism.

… This Will Never Get Implemented

- The biggest problem with the work is that it underestimates the
  difficulty in modifying DRAM process for benefit in only a subset of
  applications which do bulk bitwise operations. In particular, I find
  it hard to believe that the commodity DRAM industry will incorporate
  this into their standard DRAM process. DRAM process is, at this
  point, a highly optimized, extremely tuned endeavor. Adding this
  kind of functionally will have a big impact on DRAM cost. The
  performance benefit on the subset of applications isn't enough to
  justify the higher costs this will incur and this will never get
  implemented.

Another Review

Another Review from ISCA 2016

Strengths

The proposed mechanisms effectively exploit the operation of the DRAM to perform efficient bitwise operations across entire rows of the DRAM.

Weaknesses

This requires a modification to the DRAM that will only help this type of bitwise operation. It seems unlikely that something like that will be adopted.

… This Will Never Get Implemented

Comments for the authors

This paper shows that DRAM could be modified to support bitwise operations directly within the DRAM itself. The performance advantages are compelling for situations in which bulk bitwise operations matter.

However, I am not really convinced that any DRAM manufacturer would really consider modifying the DRAM in this way. It beneifts one specific type of operation, and while that is important for some applications, it is not really a general-purpose operation. It is not like the STL library would be changed to use this for its implementation of sets.

Yet Another Review

Yet Another Review from ISCA 2016

Weaknesses

The core novelty of Buddy RAM is almost all circuits-related (by exploiting sense amps). I do not find architectural innovation even though the circuits technique benefits architecturally by mitigating memory bandwidth and relieving cache resources within a subarray. The only related part is the new ISA support for bitwise operations at DRAM side and its induced issue on cache coherence.

This paper suits better to be peer-reviewed and published in a circuit conference or with a fabricated chip in ISSCC.

A Review from HPCA 2017: REJECT

Review #119A

Paper summary
Paper proposes DRAM technology changes (inverts, etc) to implement bit-wise operations directly on DRAM rows.

Overall merit
1. Reject

Post-response overall merit
Unknown

Reviewer expertise
2. I have passing familiarity with this area

Writing quality
3. Adequate

Experimental methodology
2. Poor

Novelty
2. Incremental improvement

Strengths
Seems like a new idea. Processor-in-Memory (PIM) ideas have resurged.

Weaknesses

#119 - HPCA23

* Impractical. Too many implications on ISA, DRAM design, and coherence protocols.
* Unlikely to benefit real-world computations.
* Evaluation did not consider full-program performance.

Comments for author
I am skeptical this would benefit real-world computations. I've never seen real-world program profiles with hot functions or instructions that are bit-wise operations.

On the other hand, I *have* seen system profiles that show non-trivial time zeroing pages. Suggest re-tooling your work to support page zeroing and evaluating that with a full-system simulation. Take a look at when/why the Linux kernel zeroes pages. You might be surprised at the possible impact.

A Review from ISCA 2017

Review #162A Updated 28 Jan 2017 5:16:50am EST Plain text

Post rebuttal overall merit
1. Reject

Overall merit
2. Weak reject

Novelty
2. Incremental improvement

Writing quality
4. Well-written

Reviewer expertise
5. This is my area

Paper summary
This paper proposes in-DRAM bit-wise operations by activating more than one word lines (and cells connected to the wordiness). Basically, it's a charge-based computation where the difference in charge stored cells connected to the same bit line is used for the logic operation.

Strengths
- conceptually a very interesting proposal (but practically not sure).
- consider various aspects including the interaction between

processors and RAM (although there isn't any new contribution and rather use the same proposal as prior work).

Weaknesses
- negative impact on the regularity of DRAM array design (and associated overhead evaluation seems to be very weak.
- significantly increase the testing cost

Comments to authors
This is an interesting proposal and well presented paper. However, I have some concerns regarding the evaluation (especially related to circuit level issues).

Especially, I feel that the variation related modeling and evaluation are weak as there are multiple sources of variations such as access transistors and sense-amp mismatches, minor defects in either access transistors and/or capacitor that can manifest in this particular proposed operation scenarios. That is, the authors oversimplify the variation modeling, which I believe failed to convince me this will work in practice. Also, the area overhead analysis sounds hand-waivy. I totally understand the difficulty of DRAM overhead analysis but also we must pursue more precise ways of evaluating the area impact as DRAM is very cost-sensitive.

Another Review from ISCA 2017

Review #162B Updated 1 Feb 2017 6:50:31pm EST

Plain text

Post rebuttal overall merit
2. Weak reject

Overall merit
2. Weak reject

Novelty
3. New contribution

Writing quality
3. Adequate

Reviewer expertise
3. I know the material, but am not an expert

Paper summary
This paper proposes performing bulk bit-wise operations at DRAM. They leverage analog operation of DRAM, and add some extra

circuits to do bit-wise operations at row granularity.

Strengths
The idea of handling bit-wise operations in memory is interesting.

Weaknesses
Not motivated well.
Not convinced the possible gains worth all the complexity.
Not convinced if the proposal is applicable in real world applications that do bit-wise operations on different data granularity.

Comments to authors
* The paper lacks motivation. The authors talk about how common bit-wise operations are. However, they do not provide any stat on how often these operations are being used, and more importantly, on what data granularity.

* Although bit-wise operations are common in some applications, they are not necessarily done at large granularity. For example, many applications do bit-wise operations at small 64-byte (or even smaller) entities. For such cases, this paper requires copying two whole rows to some temporary rows, and doing the operation on those rows. Please explain how you handle such cases, and what the benefits would be.

* What happens if the user does bit-wise operation on two 8-byte data, and want to store it in a third block?

* What happens if both operands are located in one row?

* The main issue with this work is that it requires flushing blocks out of caches to do the bit-wise operations. Imagine you have blocks A and B in the cache, as discussed in section 6.2.3., the proposal would flush them out of caches (not sure how?), writes

ISCA 2017 Summary

@A1 6 Mar 2017

This paper was discussed both online and at the PC meeting. Reviewers were uniformly positive about the novelty of the proposed Buddy-RAM design. However, reviewers were also concerned about the feasibility of the design. During the post-rebuttal and PC discussion, the main concerns raised were (1) the impact of process variation on the design's functional correctness;

#162 - ISCA 2017

(2) the potential reliability issues that arise due to the lack of ECC/CRC mechanisms; and (3) the impact on DRAM testing cost.

Specifically on point (1), some reviewers raised concerns about the limitations of the simulations performed to address variability: "Monte-Carlo cannot capture tail distribution of cell failures. Also Monte-Carlo cannot capture random correlated WID process variation issues (only some random uncorrelated variations)."

Given these concerns, the PC ultimately decided to reject the paper. We hope that this feedback is useful in preparing a future version of the paper.

The Reviewer Accountability Problem

Acknowledgments

We thank the reviewers of ISCA 2016/2017, MICRO 2016/2017, and HPCA 2017 for their valuable comments. We

MICRO 2017: Accepted

Accepted

Submission (1837kB) 4 Apr 2017 11:33:57pm EDT ·
7420f9f02c549bcca0dc6216a5e9887dffe0d422

Revision (1852kB) 14 Jun 2017 4:16am EDT ·
22f0d123a22cf04960928e0ac43d972b5a33a848

▼**Abstract**

Many important applications trigger bitwise operations on large bit vectors (bulk bitwise operations). In fact, recent

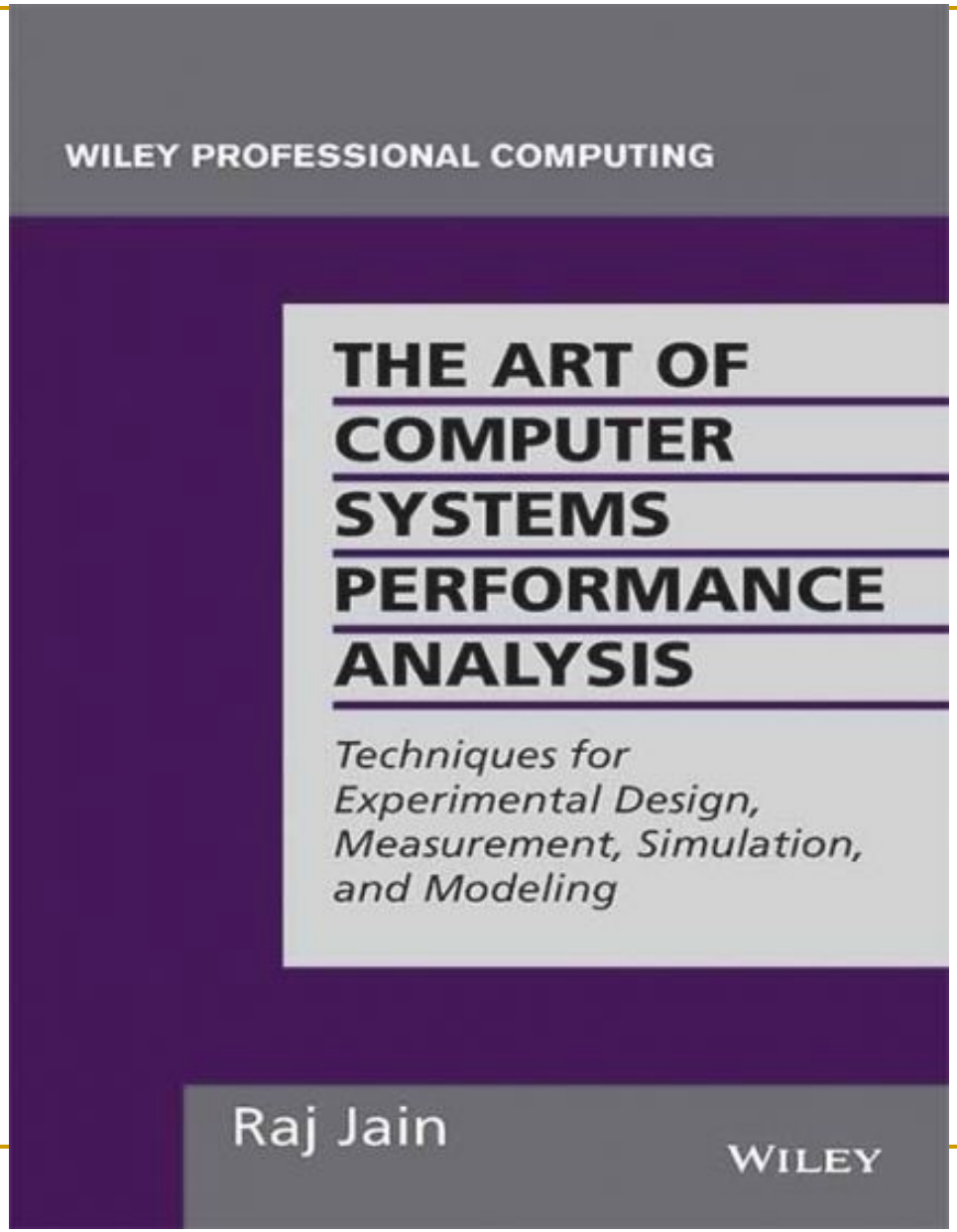
▶**Authors**

V. Seshadri, D. Lee, T. Mullins, H. Hassan, A. Boroumand, J. Kim, M. Kozuch, O. Mutlu, P. Gibbons, T. Mowry [details]

	PosResOve	OveMer	RevExp	Nov	PotImp	WriQua	ImpRev
Review #347A	4	3	5	4	3	4	2
Review #347B	3	4	5	4	3	4	3
Review #347C		2	5	3	2	4	4
Review #347D	3	4	4	4	4	4	2
Review #347E	3	3	4	3	3	3	4
Review #347F	3	2	4	3	3	4	1

1 Comment: *Rebuttal Response (V. Seshadri)*

Aside: A Recommended Book



Raj Jain, "The Art of Computer Systems Performance Analysis," Wiley, 1991.

10.8 DECISION MAKER'S GAMES

Even if the performance analysis is correctly done and presented, it may not be enough to persuade your audience—the decision makers—to follow your recommendations. The list shown in Box 10.2 is a compilation of reasons for rejection heard at various performance analysis presentations. You can use the list by presenting it immediately and pointing out that the reason for rejection is not new and that the analysis deserves more consideration. Also, the list is helpful in getting the competing proposals rejected!

There is no clear end of an analysis. Any analysis can be rejected simply on the grounds that the problem needs more analysis. This is the first reason listed in Box 10.2. The second most common reason for rejection of an analysis and for endless debate is the workload. Since workloads are always based on the past measurements, their applicability to the current or future environment can always be questioned. Actually workload is one of the four areas of discussion that lead a performance presentation into an endless debate. These "rat holes" and their relative sizes in terms of time consumed are shown in Figure 10.26. Presenting this cartoon at the beginning of a presentation helps to avoid these areas.

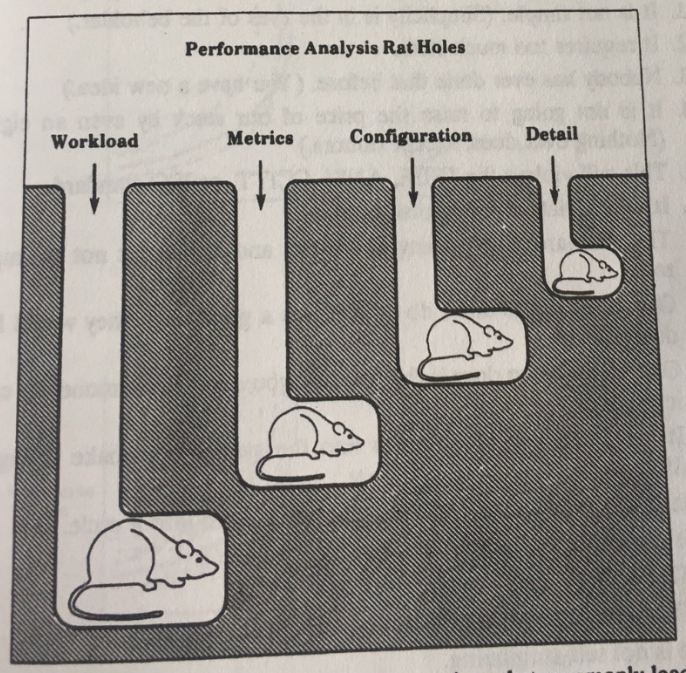
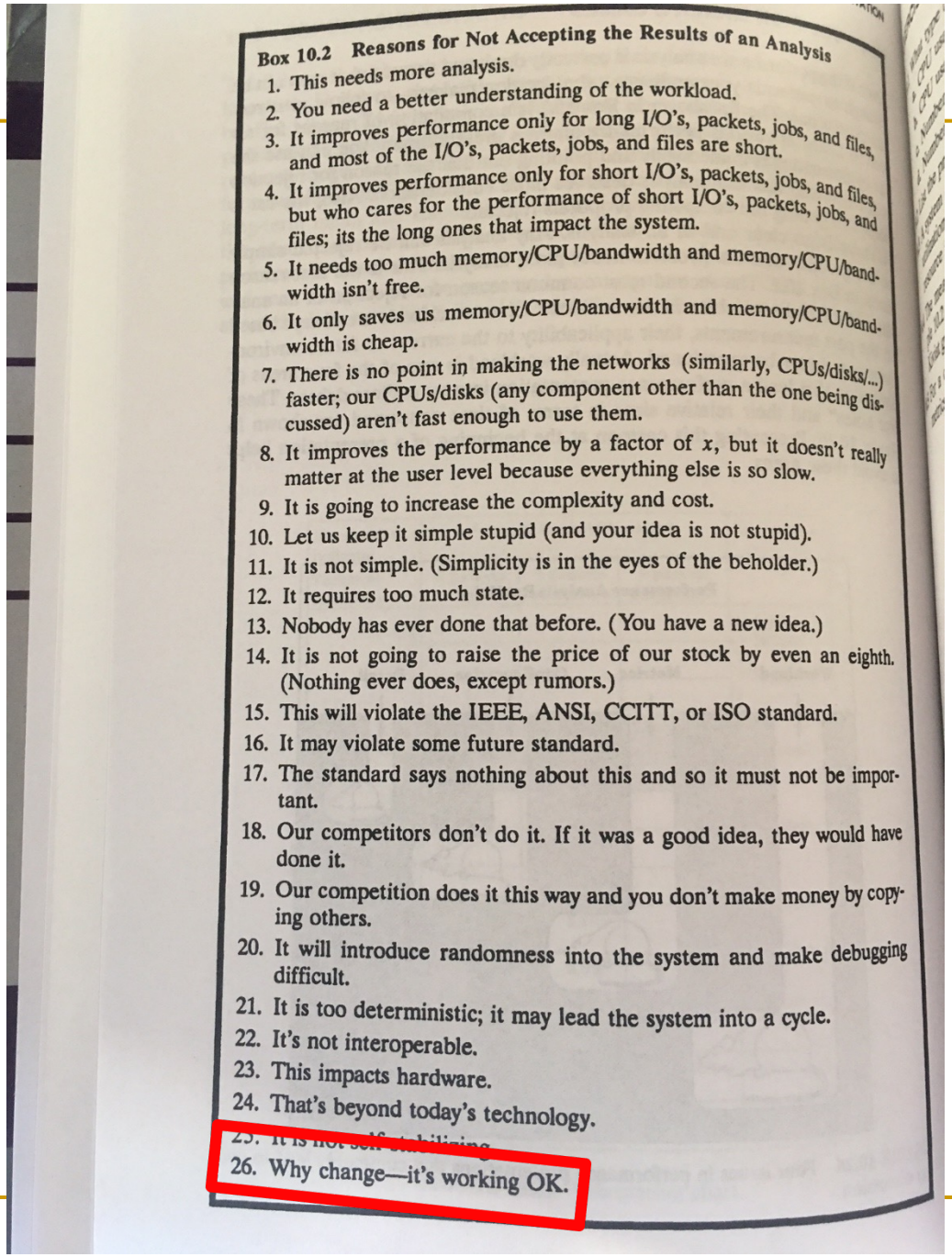


FIGURE 10.26 Four issues in performance presentations that commonly lead to endless discussion.

Raj Jain, "The Art of Computer Systems Performance Analysis," Wiley, 1991.



Box 10.2 Reasons for Not Accepting the Results of an Analysis

1. This needs more analysis.
2. You need a better understanding of the workload.
3. It improves performance only for long I/O's, packets, jobs, and files, and most of the I/O's, packets, jobs, and files are short.
4. It improves performance only for short I/O's, packets, jobs, and files, but who cares for the performance of short I/O's, packets, jobs, and files; its the long ones that impact the system.
5. It needs too much memory/CPU/bandwidth and memory/CPU/bandwidth isn't free.
6. It only saves us memory/CPU/bandwidth and memory/CPU/bandwidth is cheap.
7. There is no point in making the networks (similarly, CPUs/disks/...) faster; our CPUs/disks (any component other than the one being discussed) aren't fast enough to use them.
8. It improves the performance by a factor of x, but it doesn't really matter at the user level because everything else is so slow.
9. It is going to increase the complexity and cost.
10. Let us keep it simple stupid (and your idea is not stupid).
11. It is not simple. (Simplicity is in the eyes of the beholder.)
12. It requires too much state.
13. Nobody has ever done that before. (You have a new idea.)
14. It is not going to raise the price of our stock by even an eighth. (Nothing ever does, except rumors.)
15. This will violate the IEEE, ANSI, CCITT, or ISO standard.
16. It may violate some future standard.
17. The standard says nothing about this and so it must not be important.
18. Our competitors don't do it. If it was a good idea, they would have done it.
19. Our competition does it this way and you don't make money by copying others.
20. It will introduce randomness into the system and make debugging difficult.
21. It is too deterministic; it may lead the system into a cycle.
22. It's not interoperable.
23. This impacts hardware.
24. That's beyond today's technology.
25. It is not self-stabilizing.
26. Why change—it's working OK.

Raj Jain, "The Art of Computer Systems Performance Analysis," Wiley, 1991.

Suggestions to Reviewers

- Be fair; you do not know it all
- Be open-minded; you do not know it all
- Be accepting of diverse research methods: there is no single way of doing research or writing papers
- Be constructive, not destructive
- Enable heterogeneity, but do **not** have double standards…

Do not block or delay scientific progress for non-reasons

Suggestion to Community

We Need to Fix the Reviewer Accountability Problem

Takeaway

Main Memory Needs Intelligent Controllers

Takeaway

Research Community Needs

Accountable Reviewers

An Interview on Research and Education

- Computing Research and Education (@ ISCA 2019)
  - https://www.youtube.com/watch?v=8ffSEKZhmvo&list=PL5Q2soXY2Zi_4oP9LdL3cc8G6NIjD2Ydz

- Maurice Wilkes Award Speech (10 minutes)
  - https://www.youtube.com/watch?v=tcQ3zZ3JpuA&list=PL5Q2soXY2Zi8D_5MGV6EnXEJHnV2YFBJl&index=15

More Thoughts and Suggestions

- Onur Mutlu,
  "Some Reflections (on DRAM)"
  *Award Speech for ACM SIGARCH Maurice Wilkes Award, at the **ISCA** Awards Ceremony*, Phoenix, AZ, USA, 25 June 2019.
  [Slides (pptx) (pdf)]
  [Video of Award Acceptance Speech (Youtube; 10 minutes) (Youku; 13 minutes)]
  [Video of Interview after Award Acceptance (Youtube; 1 hour 6 minutes) (Youku; 1 hour 6 minutes)]
  [News Article on "ACM SIGARCH Maurice Wilkes Award goes to Prof. Onur Mutlu"]

- Onur Mutlu,
  "How to Build an Impactful Research Group"
  *57th Design Automation Conference Early Career Workshop (**DAC**)*, Virtual, 19 July 2020.
  [Slides (pptx) (pdf)]

Suggestion to Researchers: Principle: Passion

Follow Your Passion

(Do not get derailed by naysayers)

Suggestion to Researchers: Principle: Resilience

Be Resilient

Principle: Learning and Scholarship

Focus on

learning and scholarship

Principle: Learning and Scholarship

The quality of your work defines your impact

Principle: Work Hard

Work Hard to
Enable Your Passion

Principle: Good Mindset, Goals & Focus

You can make a
good impact
on the world

Recommended Interview on Research & Education

- **Computing Research and Education** (@ ISCA 2019)
  - https://www.youtube.com/watch?v=8ffSEKZhmvo&list=PL5Q2soXY2Zi_4oP9LdL3cc8G6NIjD2Ydz

- **Maurice Wilkes Award Speech** (10 minutes)
  - https://www.youtube.com/watch?v=tcQ3zZ3JpuA&list=PL5Q2soXY2Zi8D_5MGV6EnXEJHnV2YFBJl&index=15

- Onur Mutlu,
  "Some Reflections (on DRAM)"
  *Award Speech for ACM SIGARCH Maurice Wilkes Award, at the **ISCA** Awards Ceremony*,
  Phoenix, AZ, USA, 25 June 2019.
  [Slides (pptx) (pdf)]
  [Video of Award Acceptance Speech (Youtube; 10 minutes) (Youku; 13 minutes)]
  [Video of Interview after Award Acceptance (Youtube; 1 hour 6 minutes) (Youku; 1 hour 6 minutes)]
  [News Article on "ACM SIGARCH Maurice Wilkes Award goes to Prof. Onur Mutlu"]

Recommended Interview

https://www.youtube.com/watch?v=8ffSEKZhmvo

A Talk on Impactful Research & Education

Arch. Mentoring Workshop @ISCA'21 - Applying to Grad School & Doing Impactful Research - Onur Mutlu

1,563 views • Premiered Jun 16, 2021 74 1 SHARE SAVE

Onur Mutlu Lectures
17.2K subscribers

ANALYTICS EDIT VIDEO

Panel talk at Undergraduate Architecture Mentoring Workshop at ISCA 2021
(https://sites.google.com/wisc.edu/uar...)

https://www.youtube.com/watch?v=83tlorht7Mc&list=PL5Q2soXY2Zi8D_5MGV6EnXEJHnV2YFBJl&index=54

Suggested Reading

Richard Hamming

``You and Your Research''

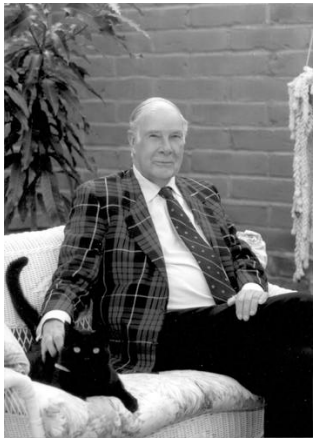
Transcription of the
Bell Communications Research Colloquium Seminar
7 March 1986

https://safari.ethz.ch/architecture/fall2021/lib/exe/fetch.php?media=youandyourresearch.pdf

Required Reading on Mindset & More

If you really want to be a first-class scientist you need to know yourself, your weaknesses, your strengths, and your bad faults, like my egotism. How can you convert a fault to an asset? How can you convert a situation where you haven't got enough manpower to move into a direction when that's exactly what you need to do? I say again that I have seen, as I studied the history, the successful scientist changed the viewpoint and what was a defect became an asset.

In summary, I claim that some of the reasons why so many people who have greatness within their grasp don't succeed are: they don't work on important problems, they don't become emotionally involved, they don't try and change what is difficult to some other situation which is easily done but is still important, and they keep giving themselves alibis why they don't. They keep saying that it is a matter of luck. I've told you how easy it is; furthermore I've told you how to reform. Therefore, go forth and become great scientists!



https://safari.ethz.ch/architecture/fall2021/lib/exe/fetch.php?media=youandyourresearch.pdf

Computer Architecture

Lecture 5: Processing using Memory

Geraldo F. Oliveira
Prof. Onur Mutlu
ETH Zürich

Fall 2023

12 October 2023