PiDRAM

An FPGA-based Framework
for End-to-end Evaluation
of Processing-in-DRAM Techniques

Ataberk Olgun

Juan Gomez Luna   Konstantinos Kanellopoulos   Behzad Salami

Hasan Hassan   Oğuz Ergin   Onur Mutlu

SAFARI

ETHzürich

kasırga

TOBB ETÜ
University of Economics & Technology

Executive Summary

Motivation: Commodity DRAM based PiM techniques improve the performance and energy efficiency of computing systems at no additional DRAM hardware cost

Problem: Challenges of integrating these PiM techniques into real systems are not solved General-purpose computing systems, special-purpose testing platforms, and system simulators *cannot* be used to efficiently study system integration challenges

Goal: Design and implement a flexible framework that can be used to:
- solve system integration challenges
- analyze trade-offs of end-to-end implementations of commodity DRAM-based-PiM techniques

Key idea: PiDRAM, an FPGA-based framework that enables:
- system integration studies
- end-to-end evaluations of PIM techniques using real unmodified DRAM chips

Evaluation: End-to-end integration of two PiM techniques on PiDRAM's FPGA prototype

Case Study #1 – RowClone: In-DRAM bulk data copy operations
- 119x speedup for copy operations compared to CPU-copy with system support
- 198 lines of Verilog and 565 lines of C++ code over PiDRAM's flexible codebase

Case Study #2 – D-RaNGe: DRAM-based random number generation technique
- 8.30 Mb/s true random number generator (TRNG) throughput, 220 ns TRNG latency
- 190 lines of Verilog and 78 lines of C++ code over PiDRAM's flexible codebase

SAFARI ⬭**kasırga** **PiDRAM:** https://github.com/CMU-SAFARI/PiDRAM 2

Outline

Background

 Commodity DRAM Based PiM Techniques

PiDRAM

 Overview

 Hardware & Software Components

 FPGA Prototype

Case Studies

 Case Study #1 – RowClone
 Case Study #2 – D-RaNGe

Conclusion

Processing-in-Memory Techniques

Use operational principles of memory
to perform bulk data movement and computation

Commodity DRAM chips can already perform:

[Gao+, MICRO'19]-[Gao+, MICRO'22]

1) Row-copy: In-DRAM bulk data copy
   (or initialization) at DRAM row granularity

(e.g., [Kim+, HPCA'19]-[Olgun+, ISCA'21])

2) True random number generation

(e.g., [Kim+, HPCA'18])

3) Physical uncloneable functions

[Gao+, MICRO'19]-[Gao+, MICRO'22]

4) Majority operation

Row-Copy: Key Idea (RowClone)
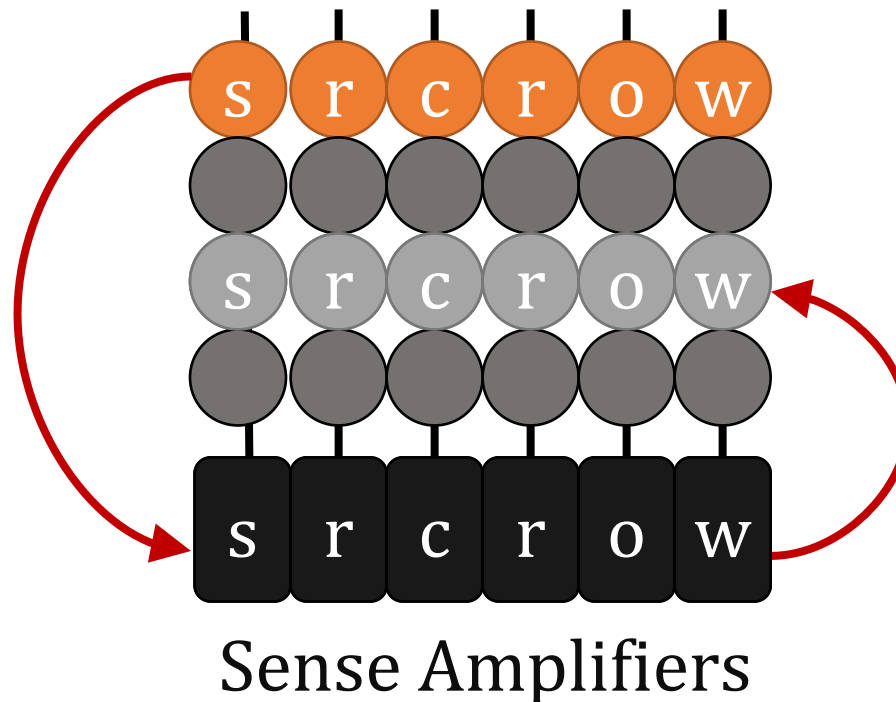
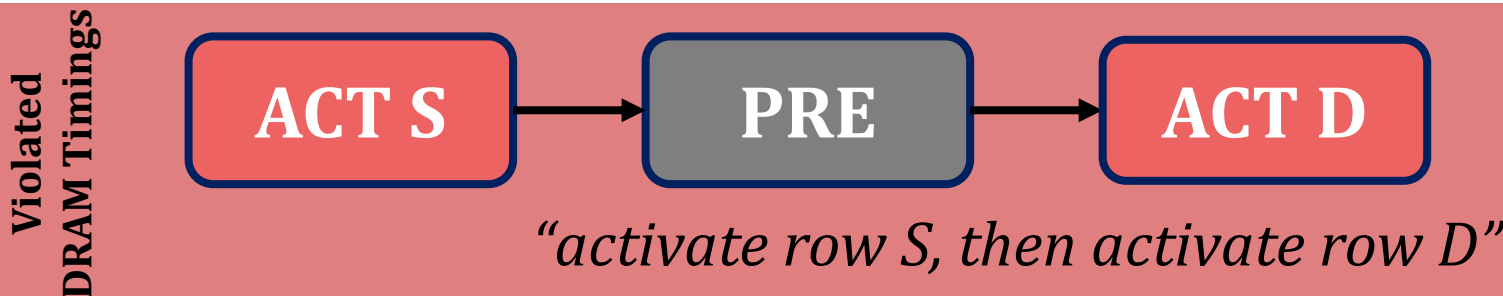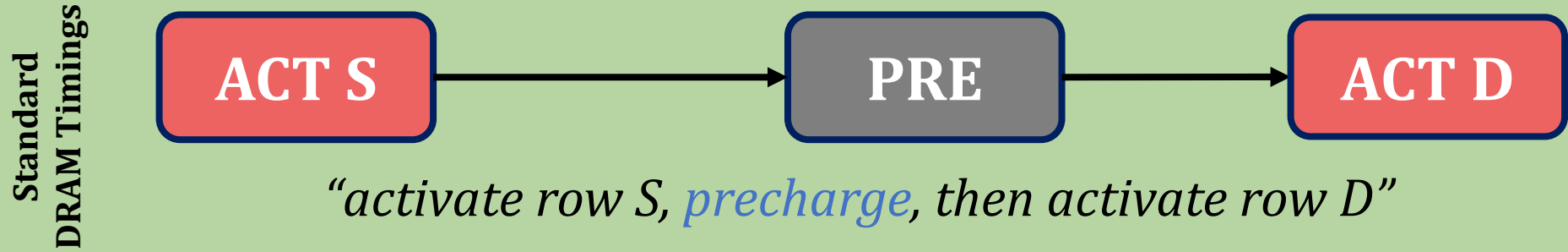Sense Amplifiers

✓ **1. Source row to sense amplifiers**
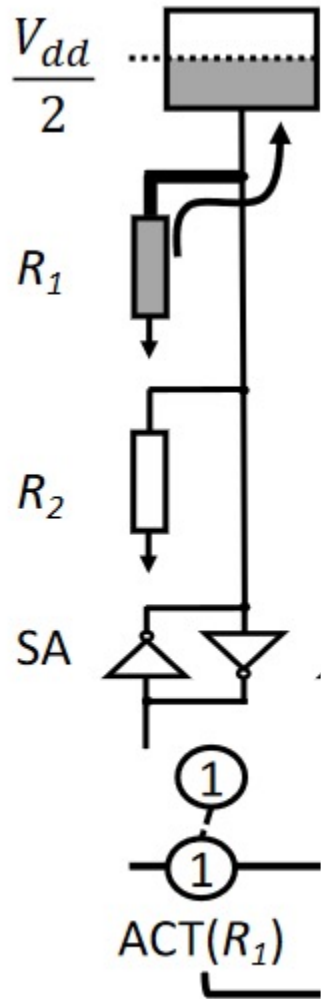
? **2. Sense amplifiers to destination row**

RowClone in Real DRAM Chips

Key Idea: Use carefully created DRAM command sequences

- ACT → PRE → ACT command sequence
  with greatly reduced DRAM timing parameters

- ComputeDRAM **[Gao+, MICRO'19]** demonstrates
  in-DRAM copy operations in real DDR3 chips

Standard DRAM Timings

ACT S → PRE → ACT D

"activate row S, precharge, then activate row D"

Violated DRAM Timings

ACT S → PRE → ACT D

"activate row S, then activate row D"

Row-copy in ComputeDRAM

More in ComputeDRAM

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

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

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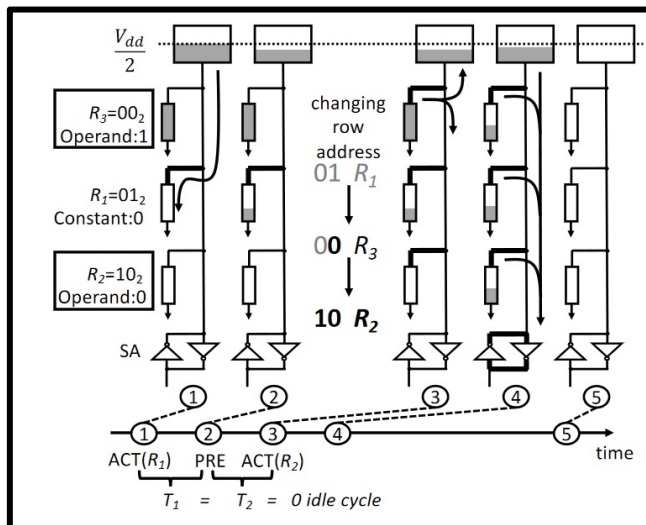georgios.tziantzioulis@princeton.edu
Department of Electrical Engineering
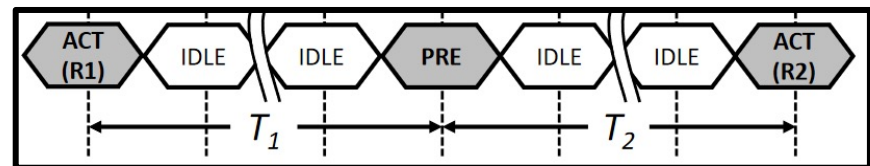Princeton University

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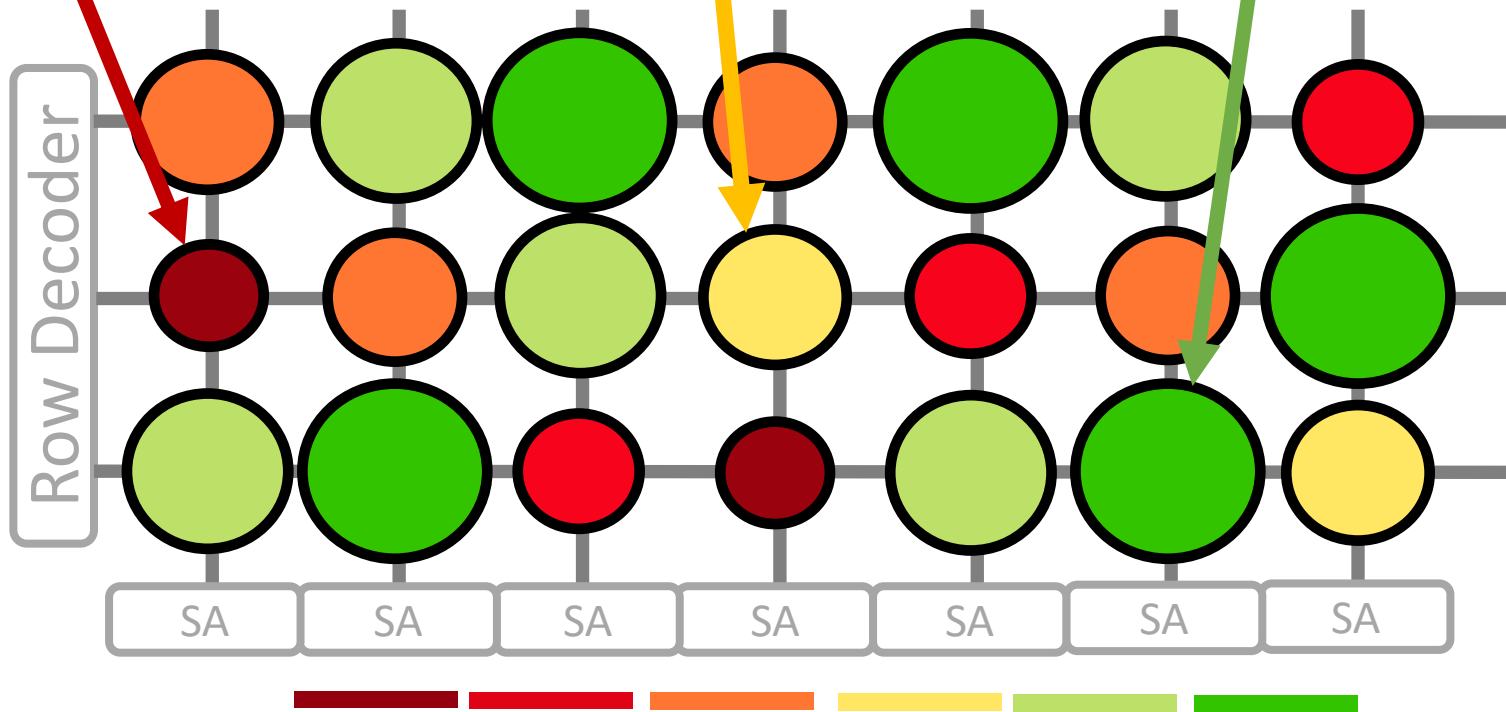
Majority Function

Row-copy/Majority Characterization

32 DDR3 Modules
~256 DRAM Chips

8

In-DRAM TRNG: Key Idea (D-RaNGe)
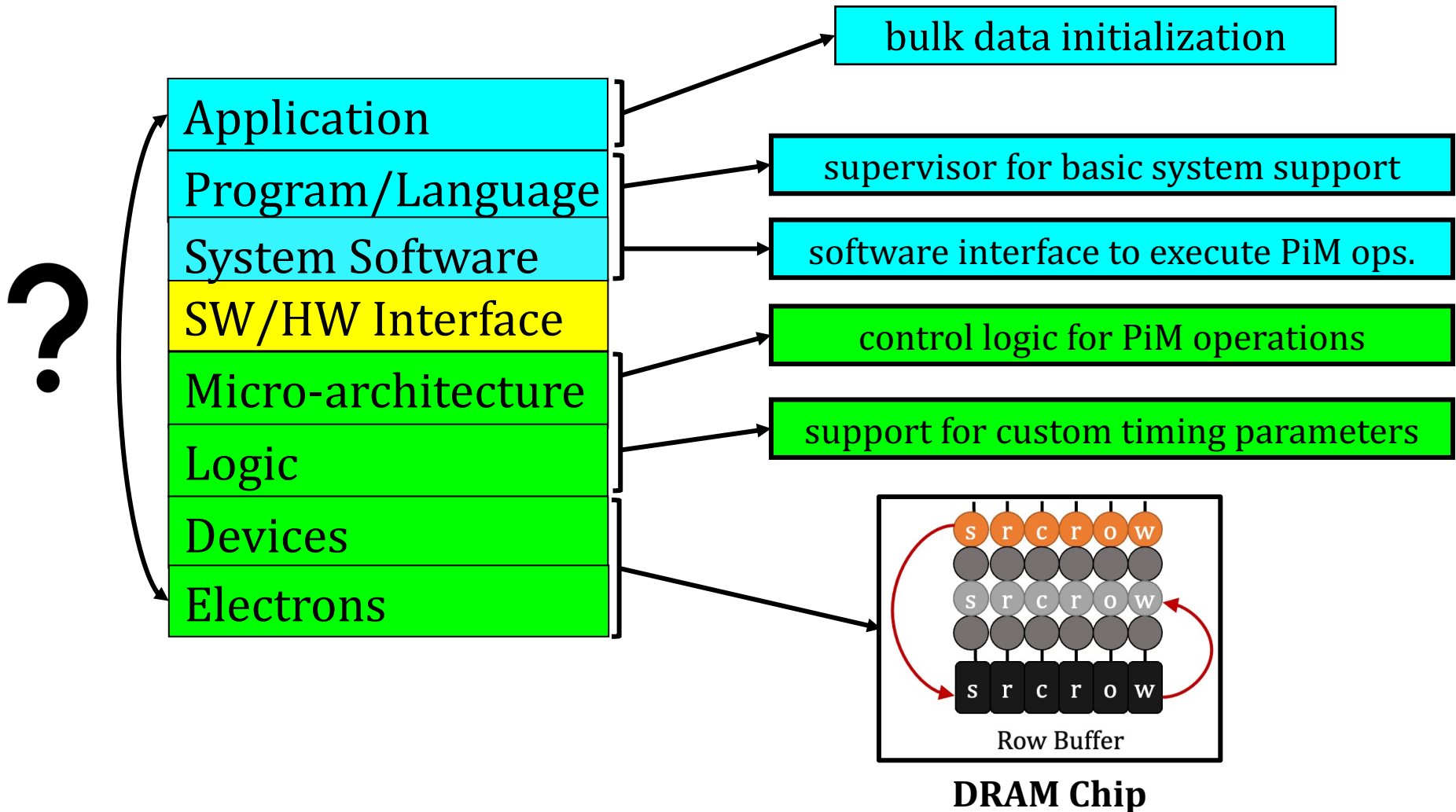
High % chance to fail with reduced access latency

50% chance to fail

Low % chance to fail with reduced access latency

Row Decoder

SA SA SA SA SA SA SA

Commodity DRAM chips can *already* perform D-RaNGe

SAFARI   kasırga   [Kim+ HPCA'19]

9

System Support for PiM

?

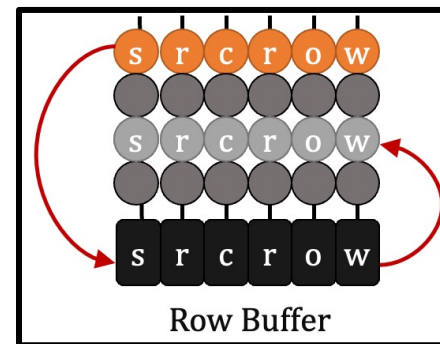| Application |
|---|
| Program/Language |
| System Software |
| SW/HW Interface |
| Micro-architecture |
| Logic |
| Devices |
| Electrons |

bulk data initialization

supervisor for basic system support

software interface to execute PiM ops.

control logic for PiM operations

support for custom timing parameters

s r c r o w
s r c r o w
s r c r o w
Row Buffer

DRAM Chip

SAFARI kasırga

PiDRAM

bulk data initialization

Bridge the "system gap"
with customizable
HW/SW components

supervisor for basic system support

software interface to execute PiM ops.

control logic for PiM operations

in doing so,
allow users to

support for custom timing parameters

rapidly implement PiM techniques,
solve system integration challenges,
analyze end-to-end implementations

Row Buffer

DRAM Chip

PiDRAM: Key Components

Control PiM operation

Memory controller for custom timing parameters

PiM Operations Controller

PiDRAM Memory Controller

Hardware

PiM Operations Library

Custom Supervisor Software

Software

Interface for Applications

Supervisor software for basic system support
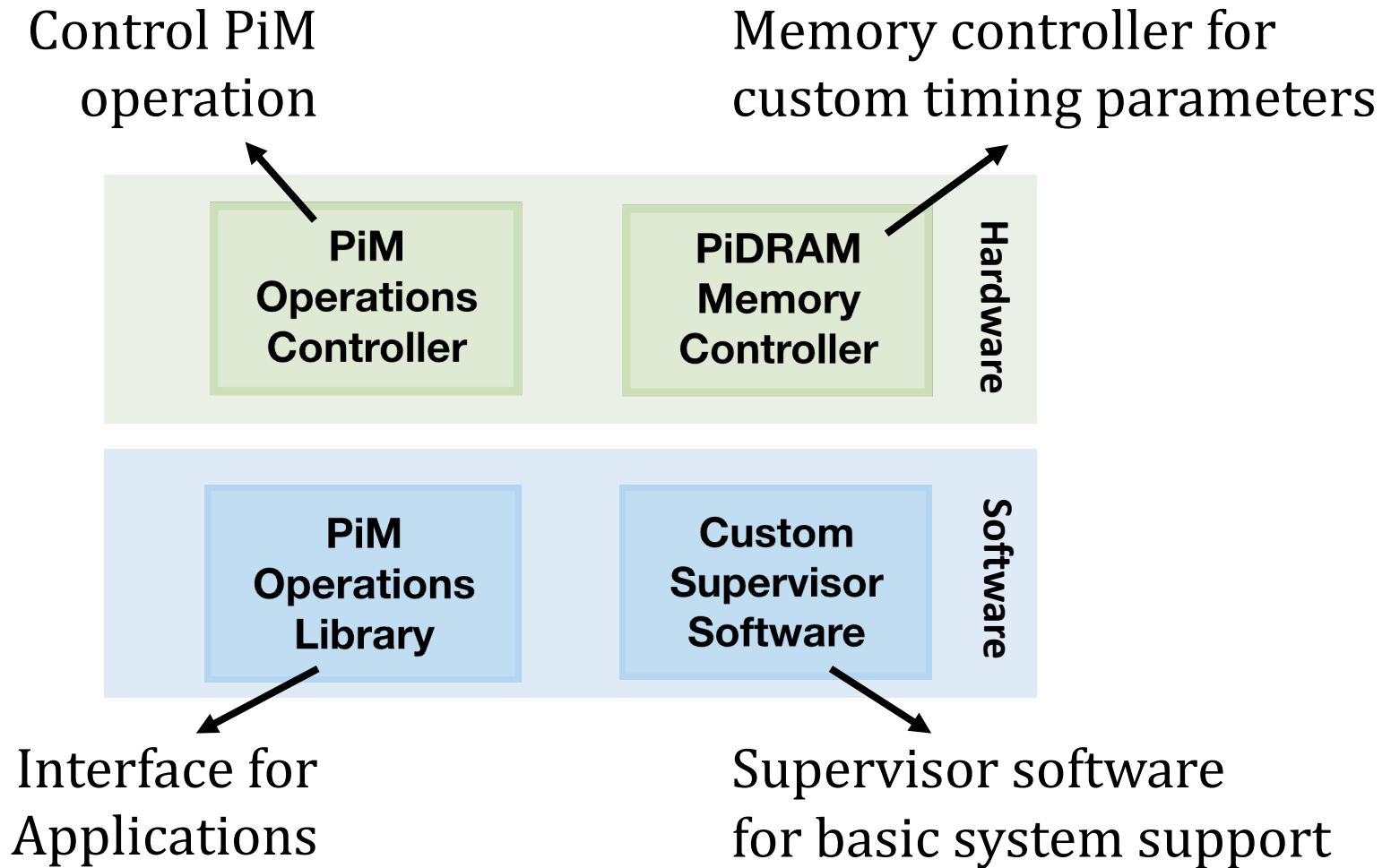
PiDRAM: System Design

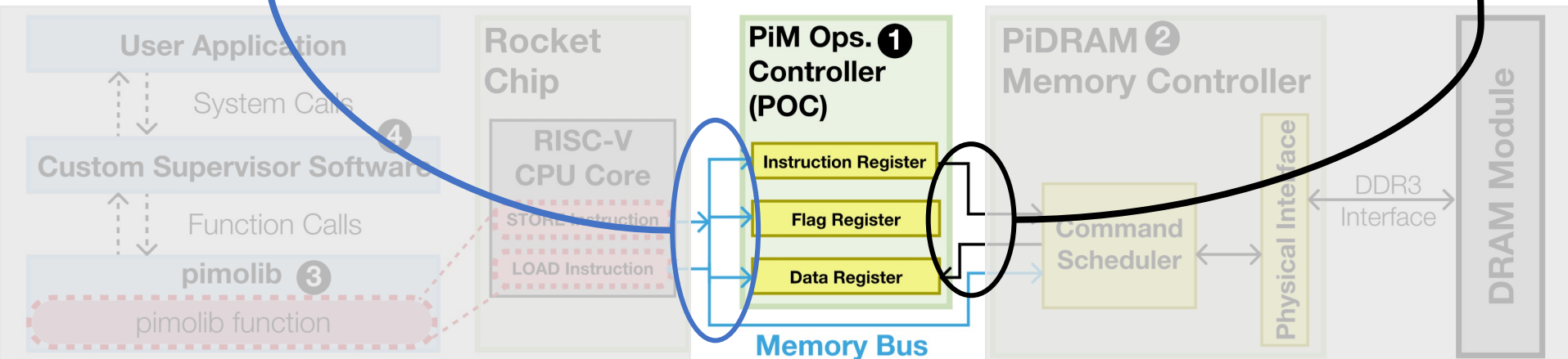Key components attached to a real computing system
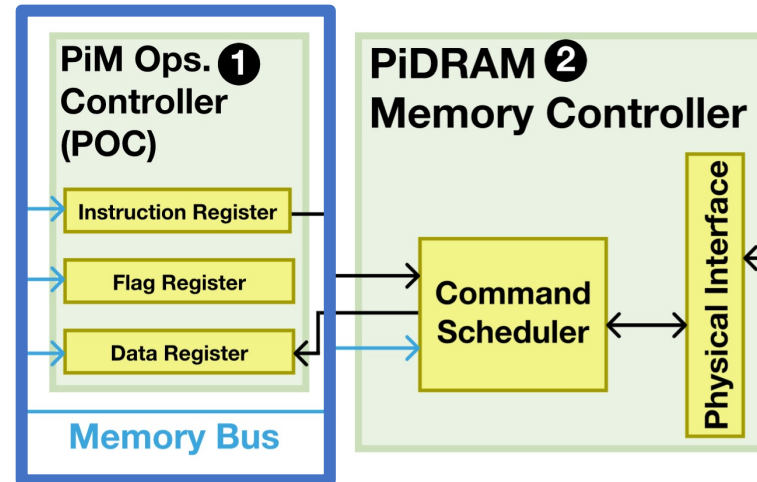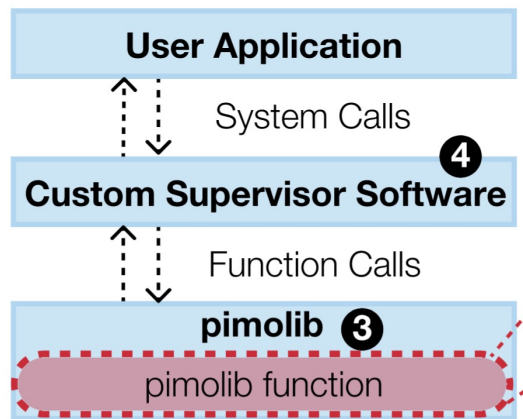
PiM Operations Controller (POC)

Decode & execute PiDRAM instructions (e.g., in-DRAM copy)

Receive instructions over memory-mapped interface

Simple interface to the PiDRAM memory controller
(i) send request, (ii) wait until completion, (iii) read results

PiDRAM Memory Controller

Perform PiM operations by violating DRAM timing parameters

Support conventional memory operations (e.g., LOAD/STORE)
One state machine per operation (e.g., LOAD/STORE, in-DRAM copy)

Easily replicate a state machine to implement a new operation

Controls the physical DDR3 interface
Receives commands from command scheduler & operates DDR3 pins
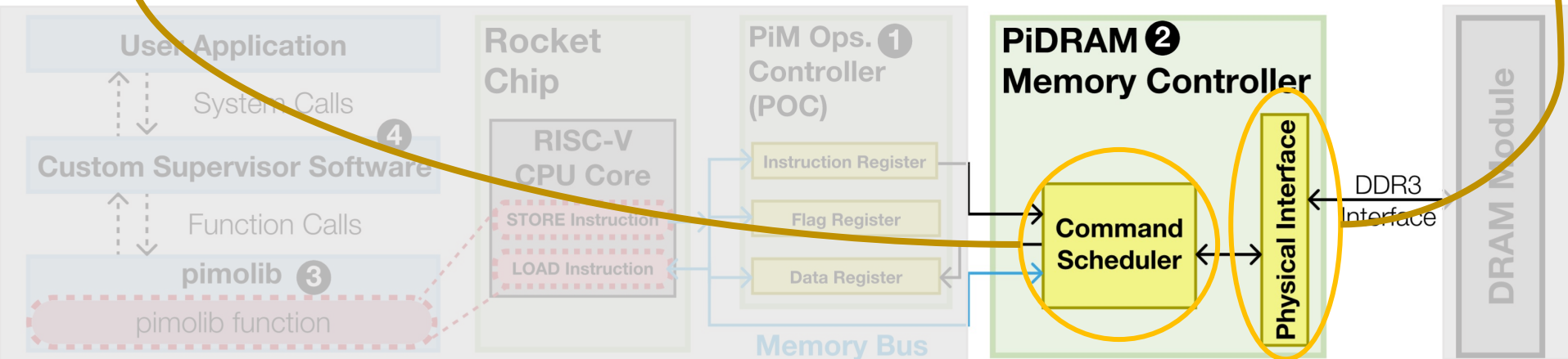
PiM Operations Library (pimolib)

Contains customizable functions that interface with the POC
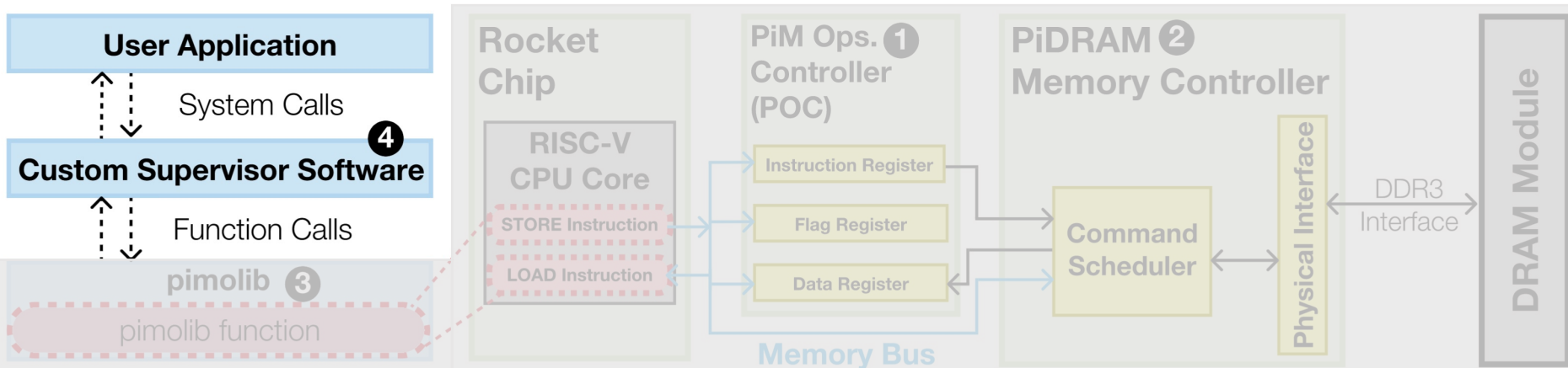**Software interface for performing PiM operations**

Executes LOAD & STORE requests to communicate with the POC
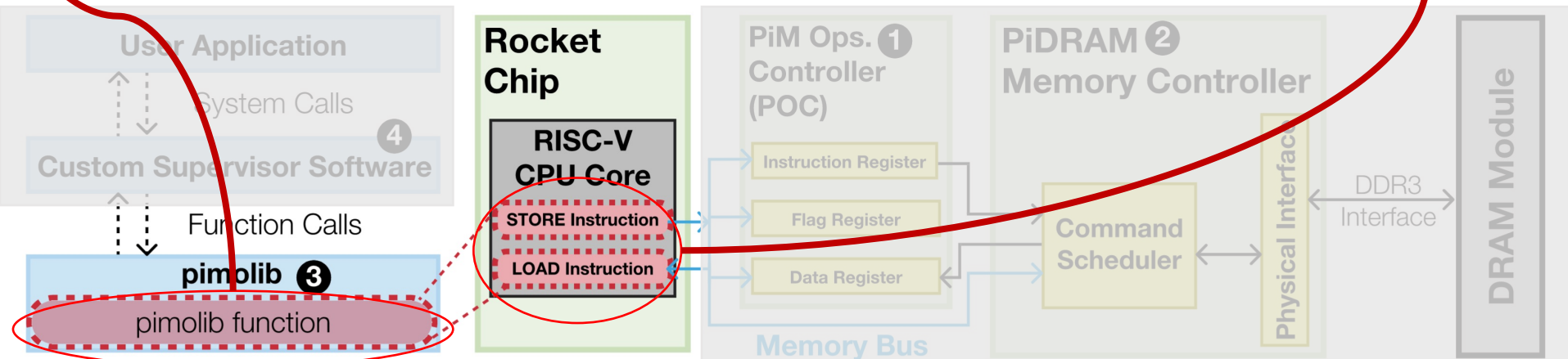
Custom Supervisor Software

Exposes PiM operations to the user application via system calls

Contains the necessary OS primitives to develop end-to-end PiM techniques (e.g., memory management and allocation for RowClone)
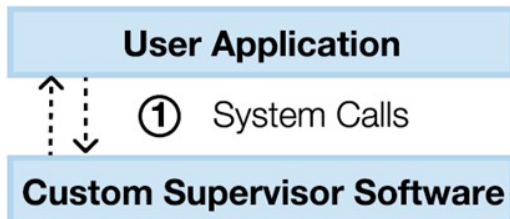
PiM Operation Execution Flow

`Copy()` function called by the user to perform a RowClone-Copy operation in DRAM

1 **Application makes a system call: `Copy(A, B, N bytes)`**

2 **Custom Supervisor Software calls the `Copy()` pimolib function**

`Copy (S, D)`

`S:` source DRAM row
`D:` destination DRAM row

User Application

① System Calls
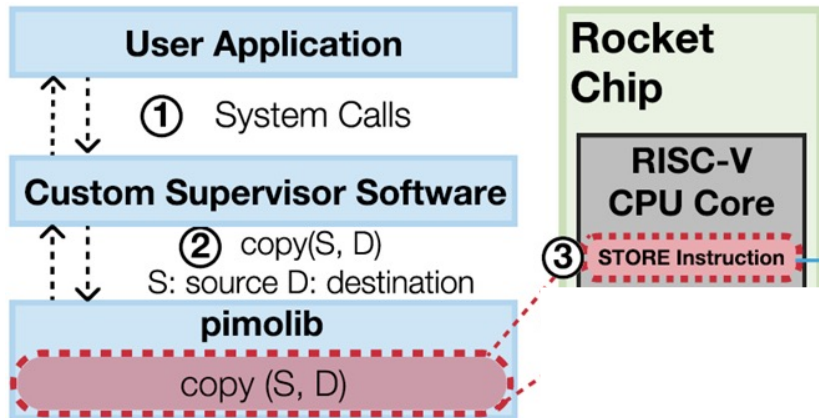
Custom Supervisor Software

PiM Operation Execution Flow

3 `Copy(S, D)` executes two *store* instructions in the CPU

4 The first store updates the *instruction* register with `Copy(S, D)`

5 The second store sets the "Start" flag in the *flag* register
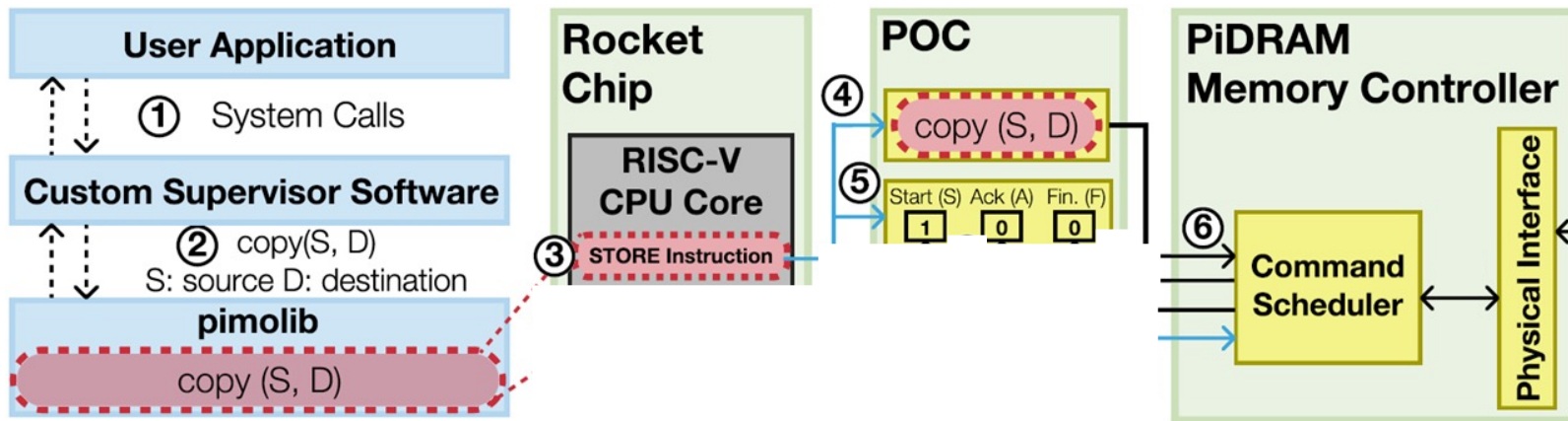
Start (S)

1 Start the execution of PiM operation

SAFARI   kasırga

PiM Operation Execution Flow

6 POC instructs the memory controller to perform RowClone

7 POC resets the "Start" flag, and sets the "Ack" flag

8 PiDRAM memory controller issues commands
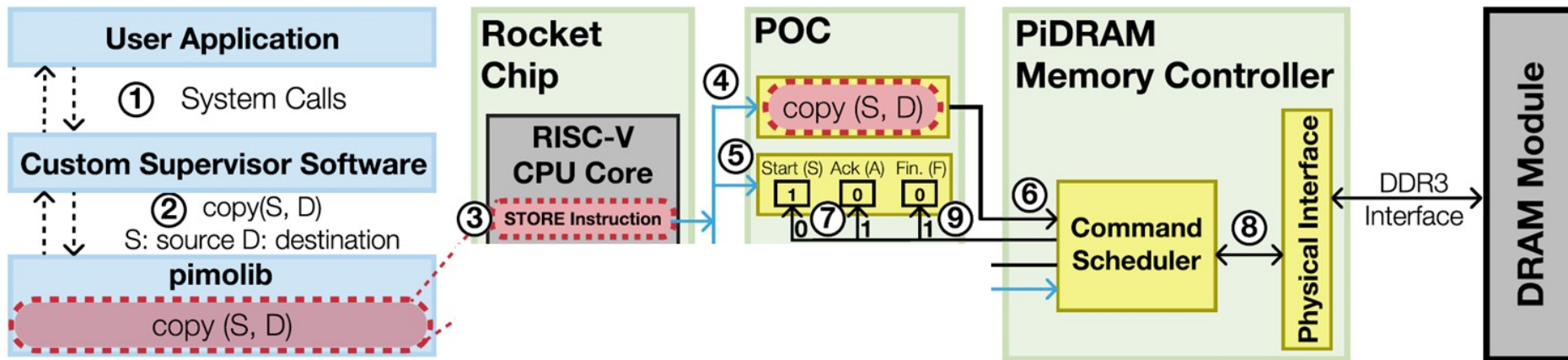with violated timing parameters to the DDR3 module

PiM Operation Execution Flow

9 The memory controller sets the "Fin." flag

10 `Copy(S, D)` periodically checks either "Ack" or "Fin." flags using LOAD instructions
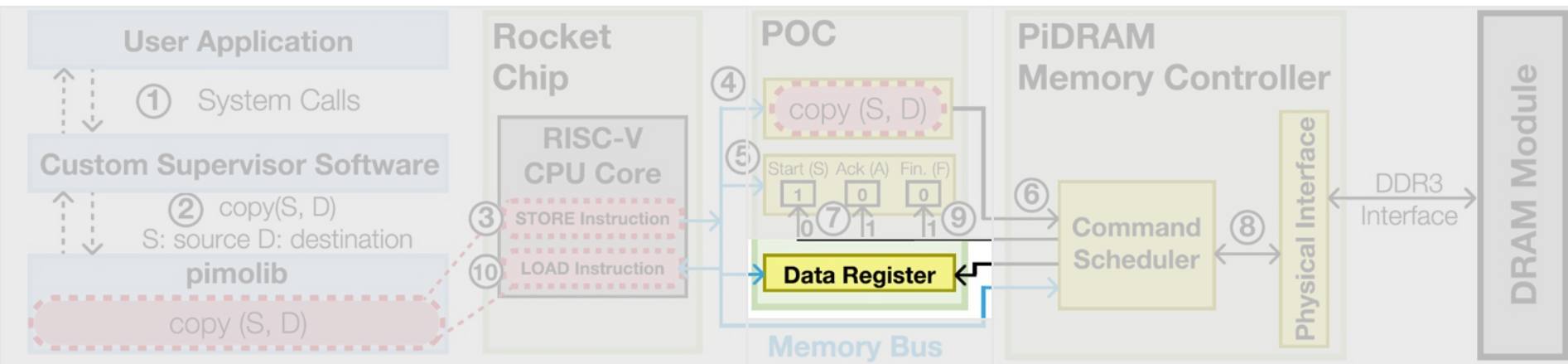
`Copy(S, D)` returns when the periodically checked flag is set

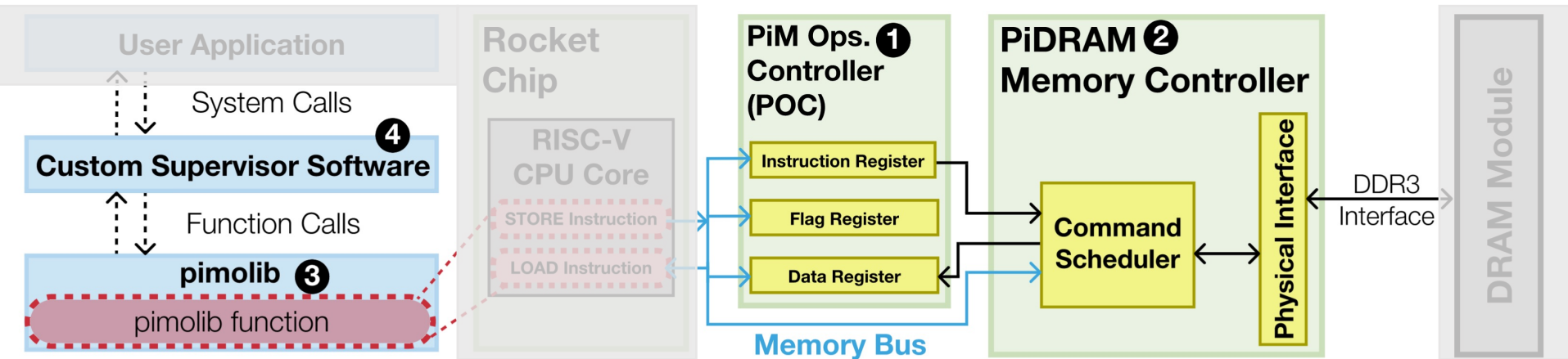# PiM Operation Execution Flow

> **Data Register is not used in RowClone operations because the result is stored *in memory***

> **It is used to read true random numbers generated by D-RaNGe**
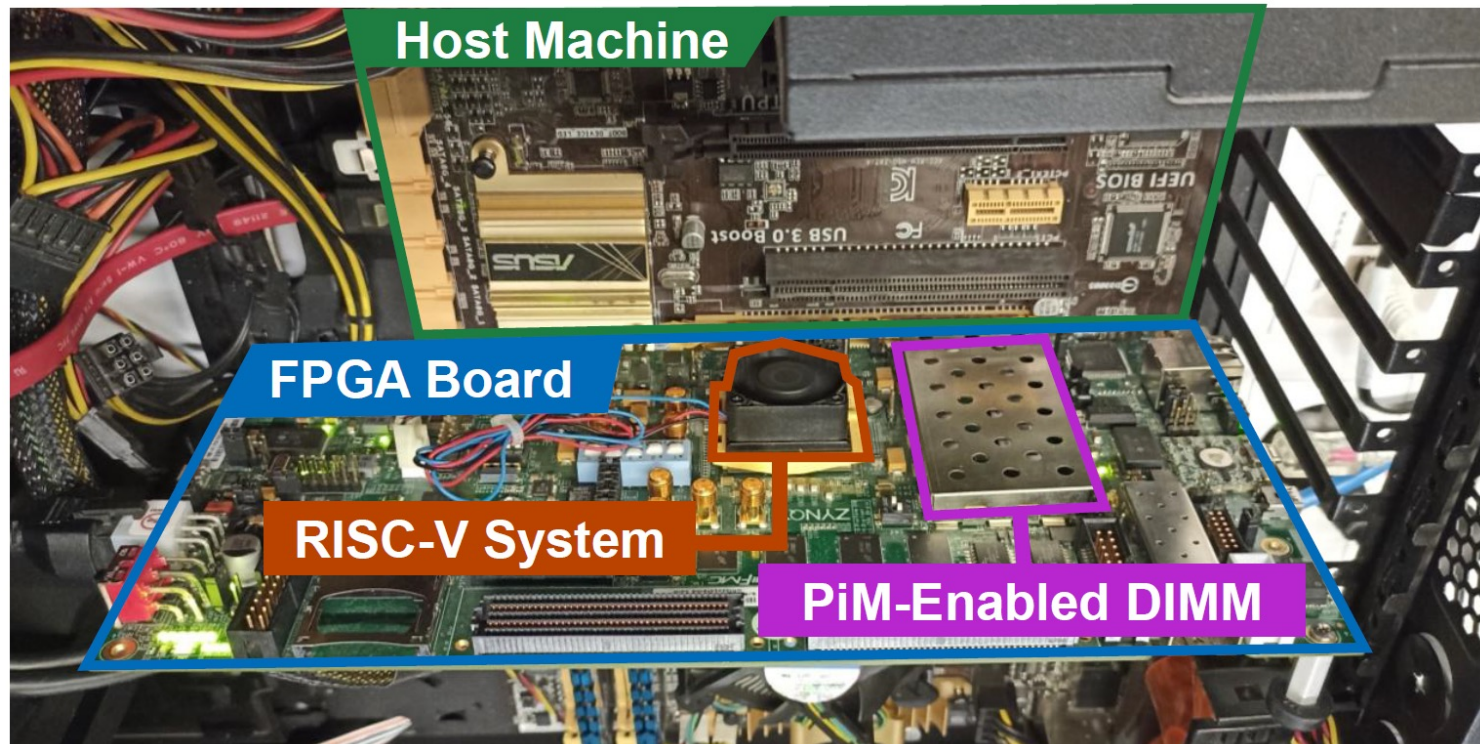
PiDRAM Components Summary

Four key components orchestrate PiM operation execution

PiDRAM's FPGA Prototype

Full system prototype on Xilinx ZC706 FPGA board

- **RISC-V System:** In-order, pipelined RISC-V Rocket CPU core, L1D/I$, TLB
- **PiM-Enabled DIMM (Commodity):** Micron MT8JTF12864, 1 GiB, 8 banks

Outline

Background

    Commodity DRAM Based PiM Techniques

PiDRAM

    Overview

    Hardware & Software Components

    FPGA Prototype

Case Studies

    Case Study #1 – RowClone

    Case Study #2 – D-RaNGe

Conclusion

RowClone Implementation

Standard DRAM Timings

[ACT S] → [PRE] → [ACT D]

"activate row S, precharge, then activate row D"

Violated DRAM Timings

[ACT S] → [PRE] → [ACT D]

"activate row S, then activate row D"

① Extend the PiDRAM memory controller
to support the DRAM command sequence

② Expose the operation to pimolib
by implementing the `copy()` PiDRAM instruction

Only 198 lines of Verilog code

SAFARI   kasırga

RowClone System Integration
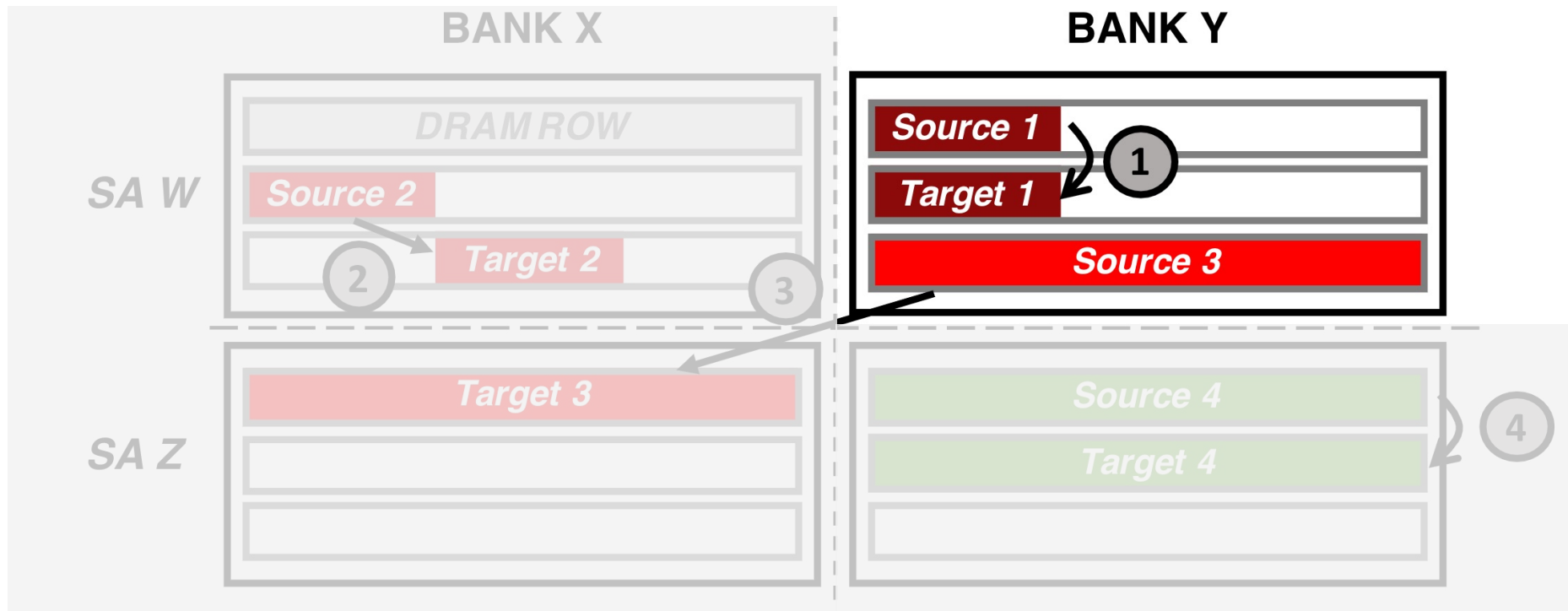
Identify two challenges in end-to-end RowClone

> **①** Memory allocation (intra-subarray operation)

② Memory coherency (computation in DRAM)

Implement CLFLUSH instruction in the RISC-V CPU
Evict a cache block from the CPU caches to the DRAM module

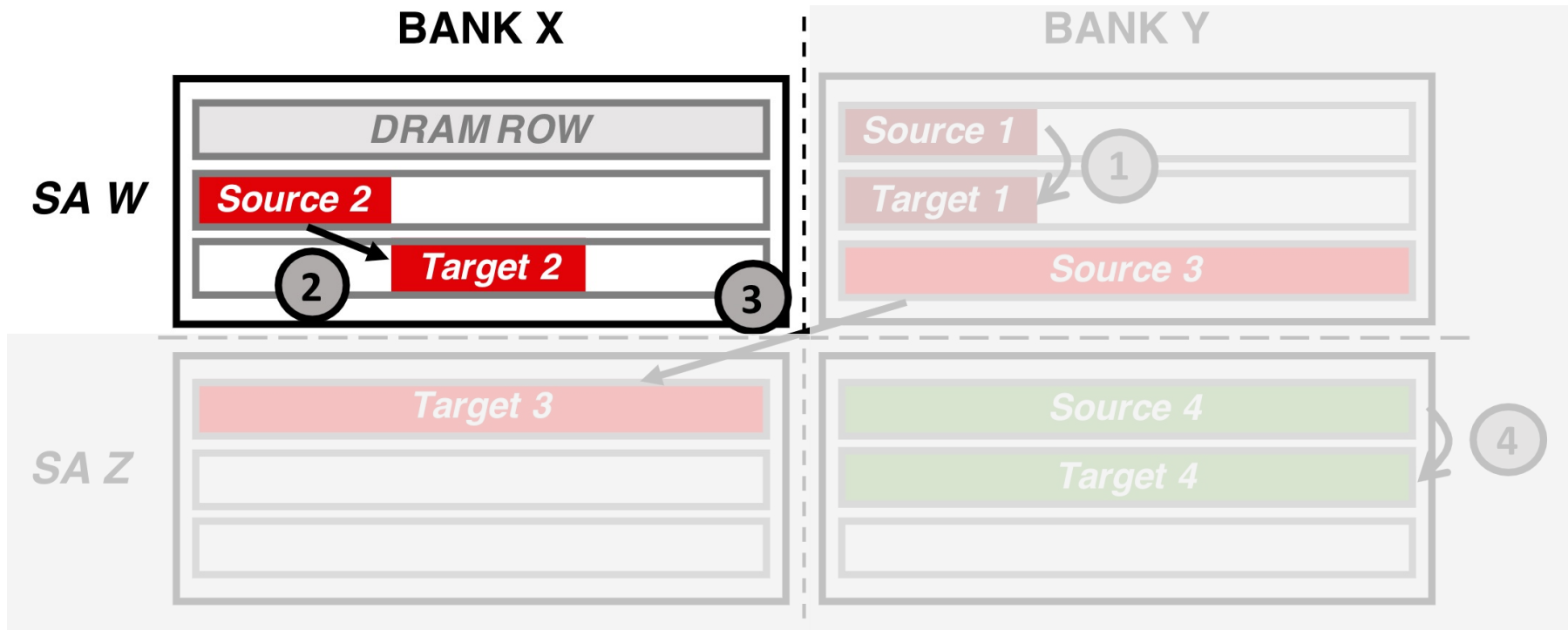# RowClone Memory Allocation (I)

Memory allocation requirements

BANK X

BANK Y

SA W

DRAM ROW

Source 2

Target 2

Source 1

Target 1

Source 3

SA Z

Target 3

Source 4

Target 4

(1) **Granularity: Operands must occupy DRAM rows fully**

RowClone Memory Allocation (I)

Memory allocation requirements

BANK X

SA W

DRAM ROW

Source 2

② Target 2 ③

BANK Y

Source 1 ①
Target 1

Source 3

SA Z
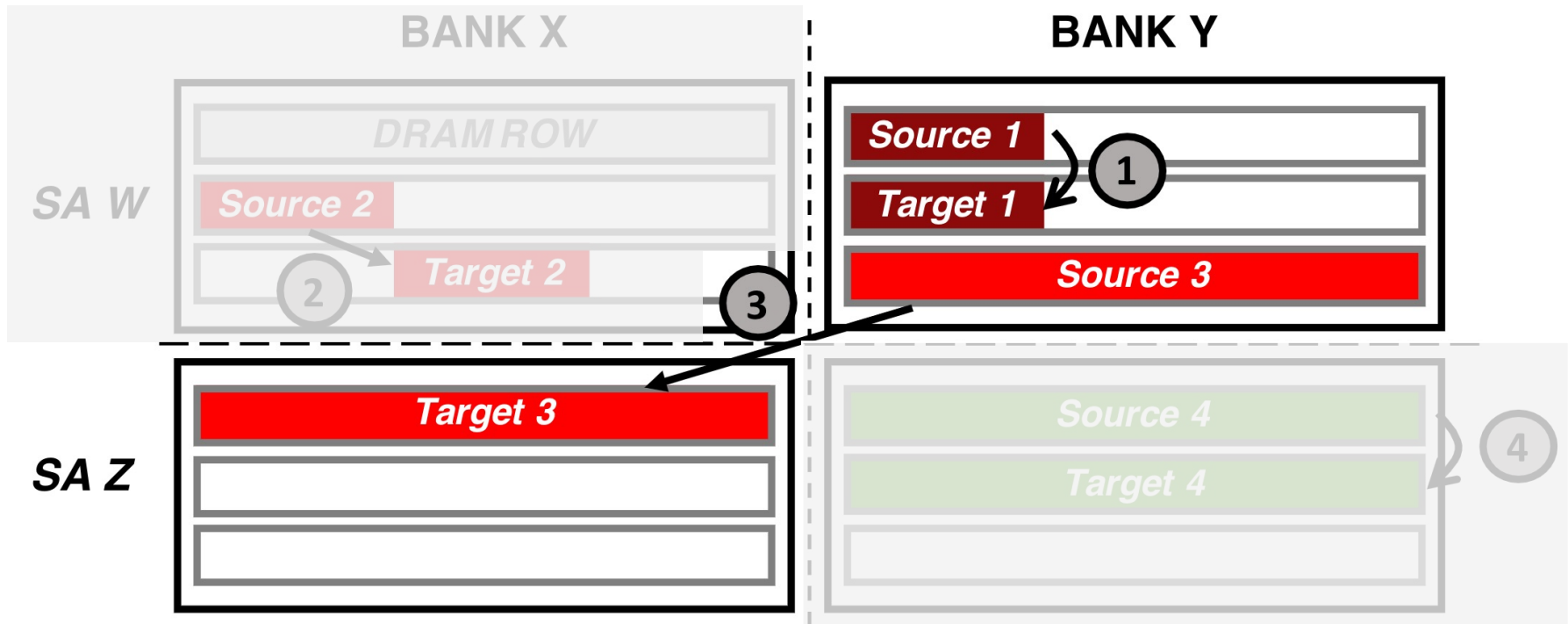
Target 3

Source 4 ④
Target 4

---

② **Alignment:** Operands must be placed at the same offset
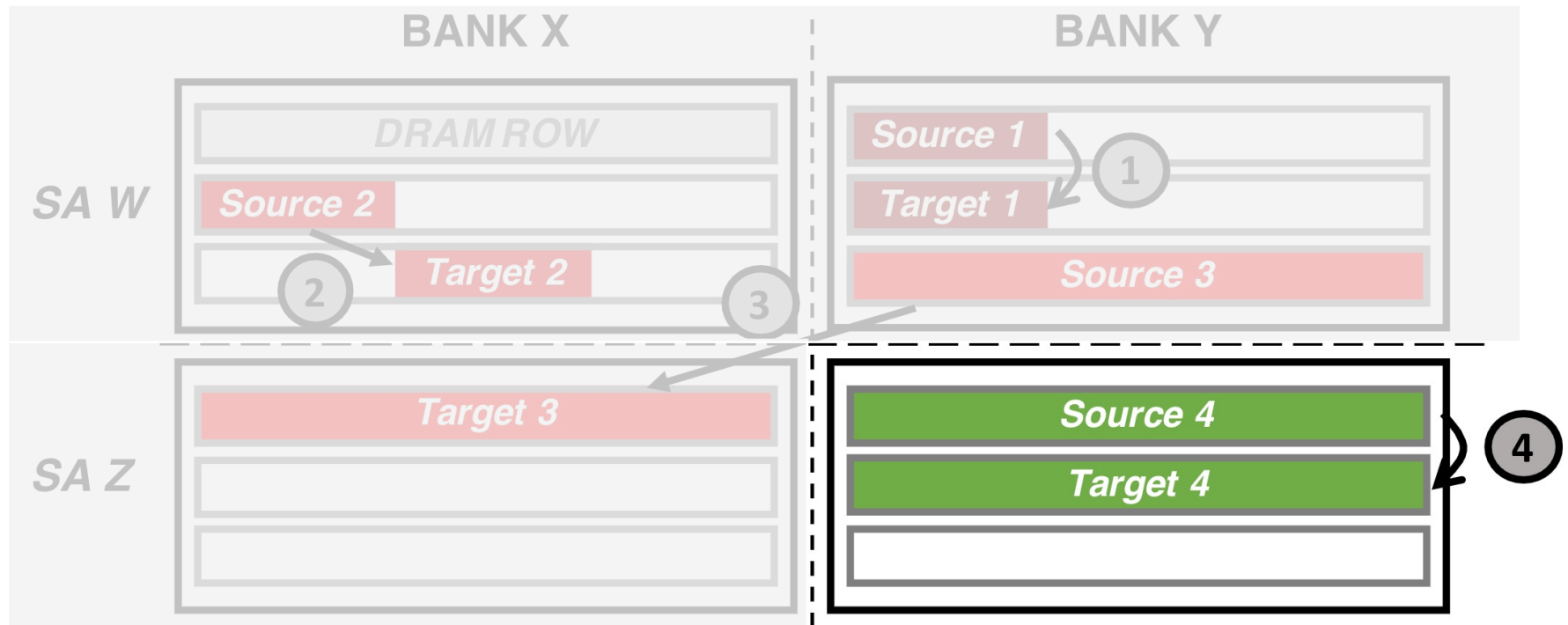
RowClone Memory Allocation (I)

Memory allocation requirements

| 3 | Mapping: Operands must be placed in the same subarray |

RowClone Memory Allocation (I)

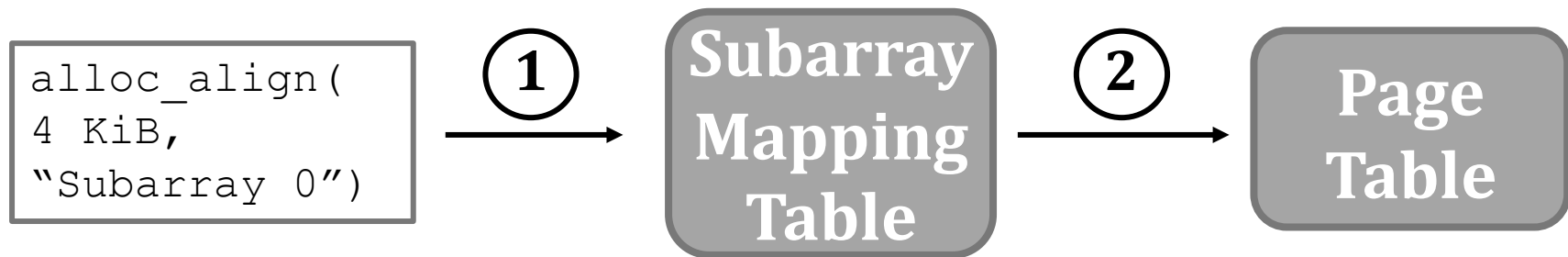Memory allocation requirements

| 4 | Satisfies all three requirements |

RowClone Memory Allocation (II)

Implement a new memory allocation function
to overcome the memory allocation challenges

Goal: Allocate virtual memory pages that are
mapped to the same DRAM subarray and aligned with each other

> virtual_address **= alloc_align(**int **size,** int **id)**
> **size:** # of bytes allocated
> **id:** allocations with the same id go to the same subarray

```
alloc_align(
4 KiB,
"Subarray 0")
```

① → **Subarray Mapping Table** **②** → **Page Table**

① Get physical address pointing to a DRAM row in subarray 0

② Update the page table to map virtual address to subarray 0

RowClone Memory Allocation (II)

Implement a new memory allocation function

https://arxiv.org/abs/2111.00082

Goal: Allocate virtual memory pages that are
mapped to the same DRAM subarray and aligned with each other

> virtual_address **= alloc_align(**int **size,** int **id)**
> **size**: # of bytes allocated
> **id**: allocations with the same id go to the same subarray

```
alloc_align(
4 KiB,
"Subarray 0")
```

① → **Subarray Mapping Table** → ② → **Page Table**

① Get physical address pointing to a DRAM row in subarray 0

② Update the page table to map virtual address to subarray 0

Evaluation: Methodology
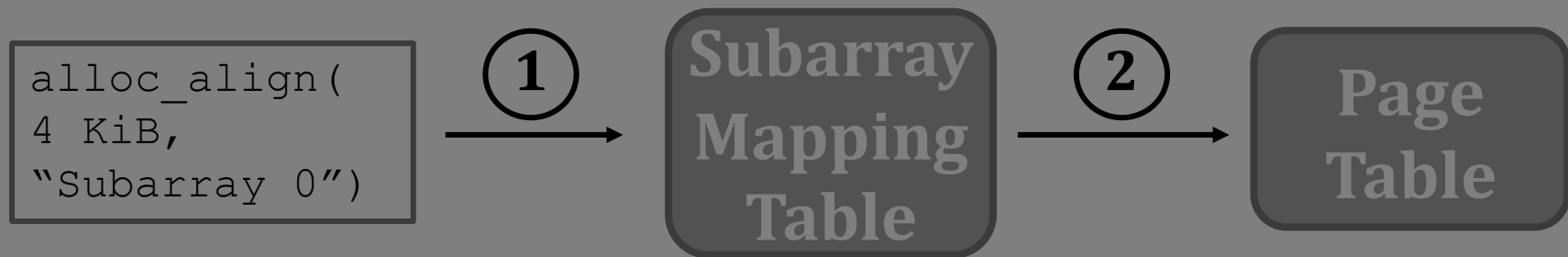
Table 2: PiDRAM system configuration

CPU: 50 MHz; in-order Rocket core [16]; **TLB** 4 entries DTLB; LRU policy	
L1 Data Cache: 16 KiB, 4-way; 64 B line; random replacement policy	
DRAM Memory: 1 GiB DDR3; 800MT/s; single rank; 8 KiB row size	

in-DRAM copy/initialization granularity

Microbenchmarks

CPU-Copy (using LOAD/STORE instructions)

RowClone-Copy (using in-DRAM copy operations) with and without CLFLUSH

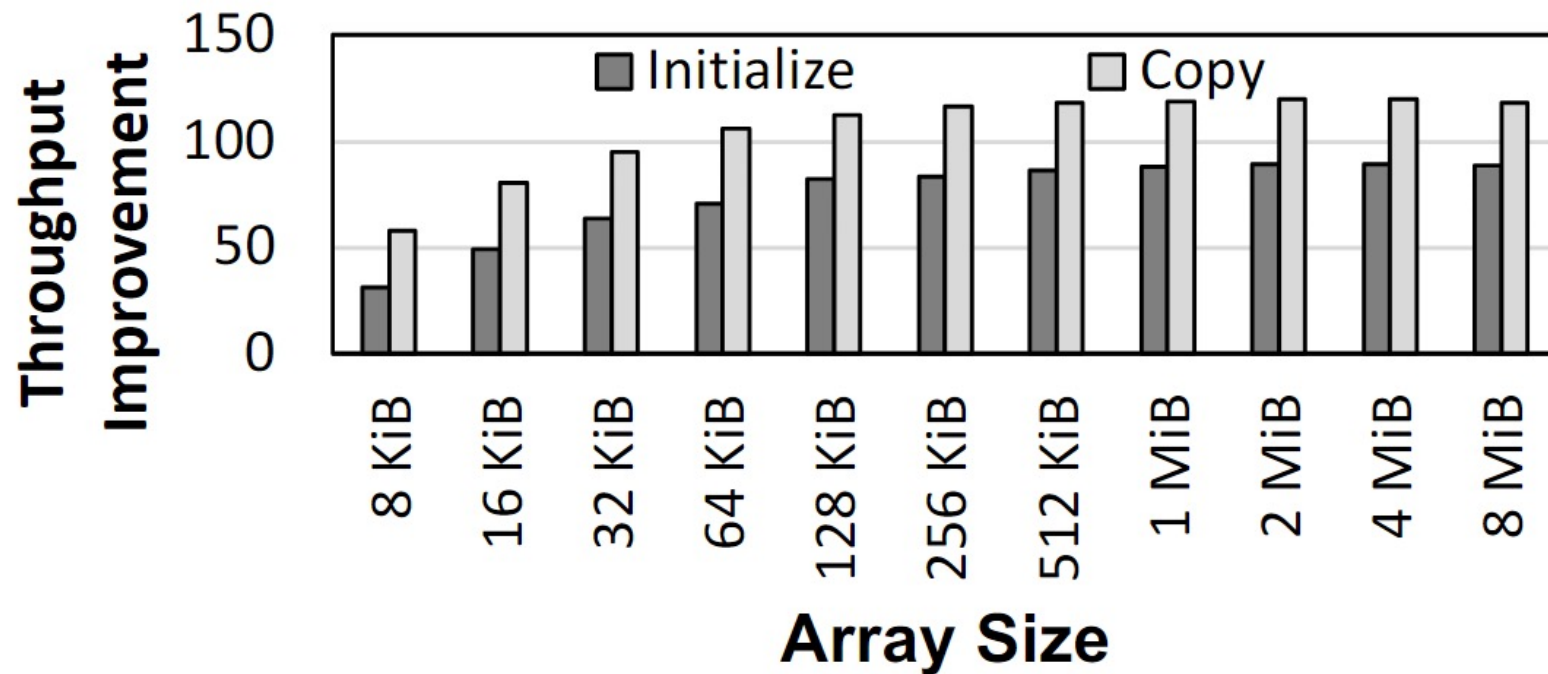**Copy/Initialization Heavy Workloads**
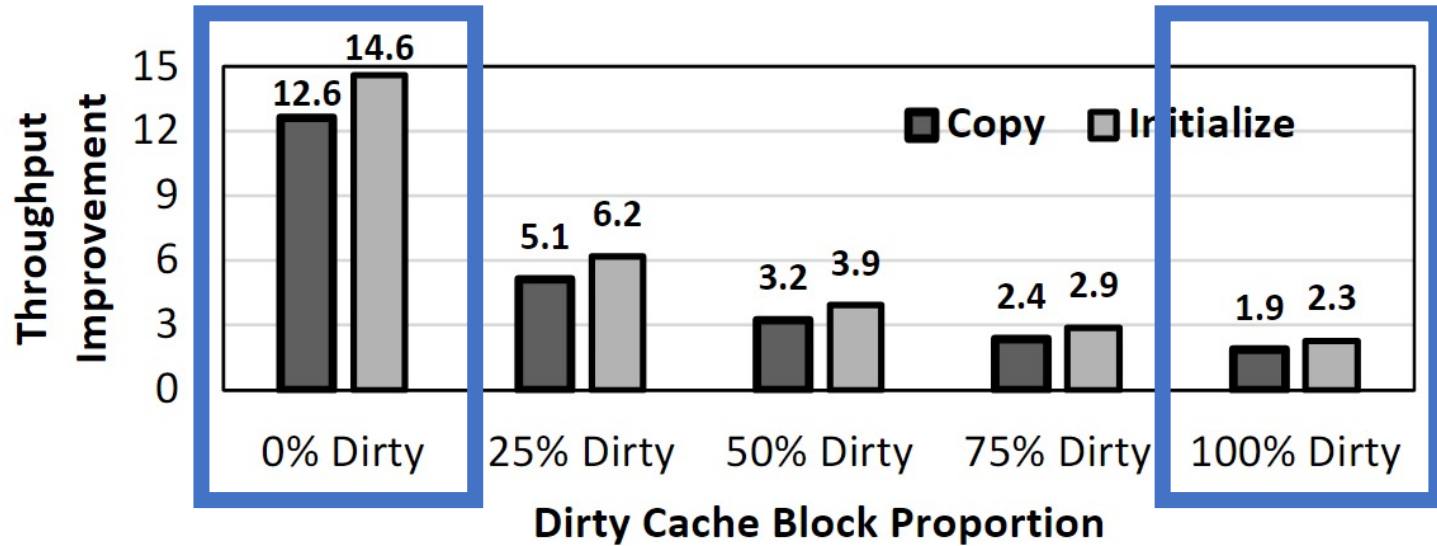
forkbench (copy)

compile (initialization)

SPEC2006 libquantum: replace "calloc()" with in-DRAM initialization

SAFARI   kasırga

Microbenchmark Copy/Initialization Throughput Improvement

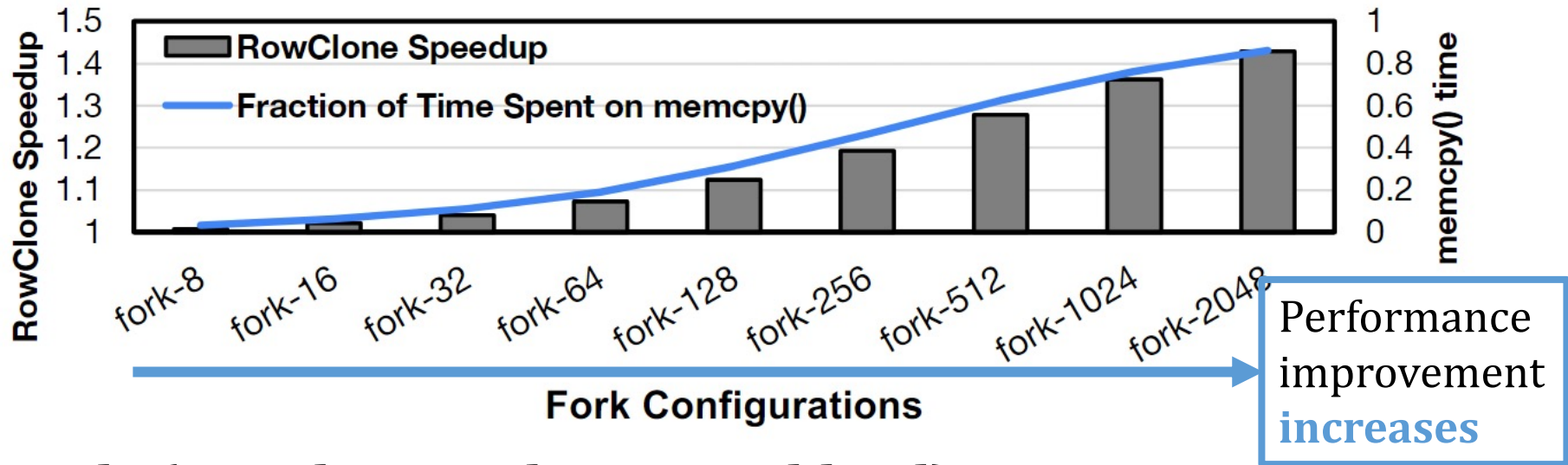**In-DRAM Copy and Initialization improve throughput by 119x and 89x, respectively**

CLFLUSH Overhead

**CLFLUSH dramatically reduces
the potential throughput improvement**

Other Workloads

forkbench (copy-heavy workload)

Performance improvement **increases**

compile (initialization-heavy workload)

- 9% execution time reduction by in-DRAM initialization
  - 17% of compile's execution time is spent on initialization

SPEC2006 libquantum

- 1.3% end-to-end execution time reduction
  - 2.3% of libquantum's time is spent on initialization
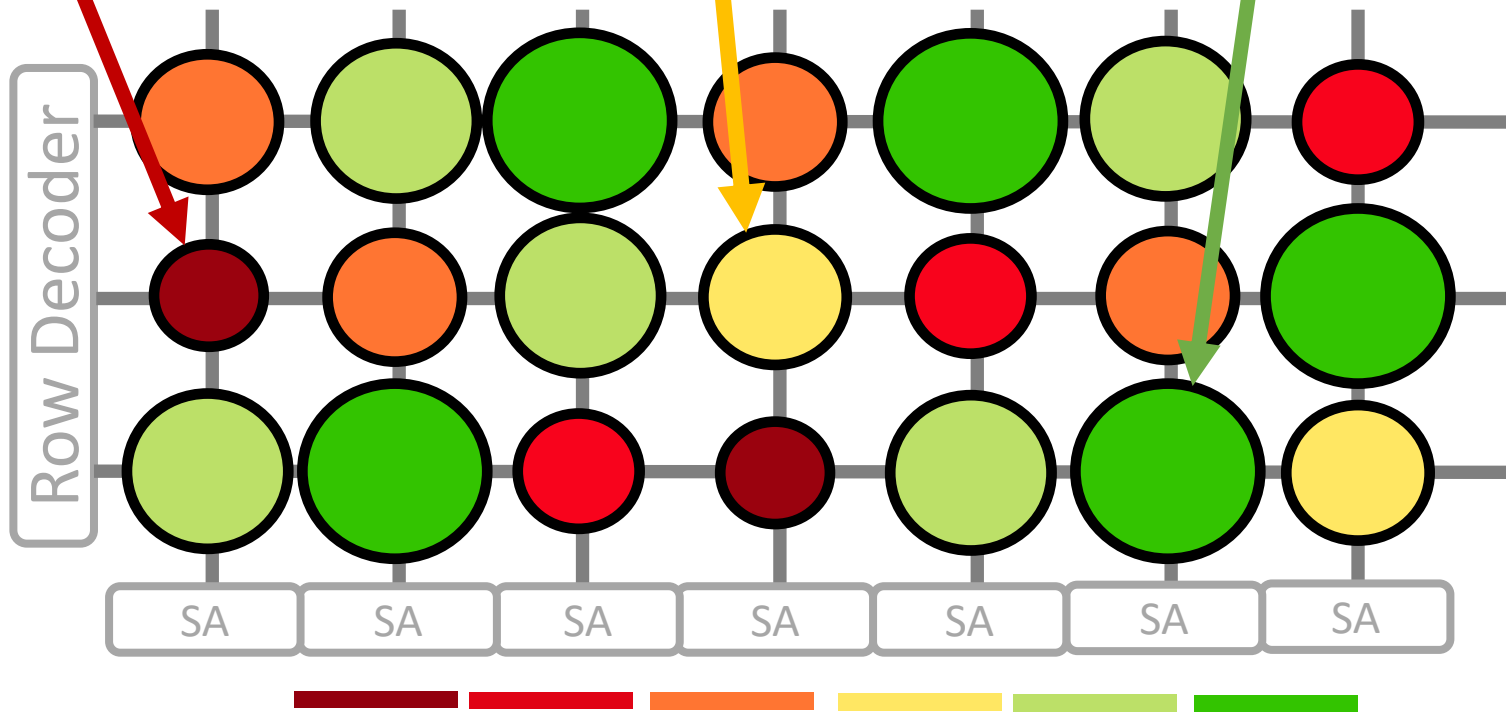
Outline

Background

Recall: D-RaNGe Key Idea

High % chance to fail with reduced access latency

50% chance to fail

Low % chance to fail with reduced access latency
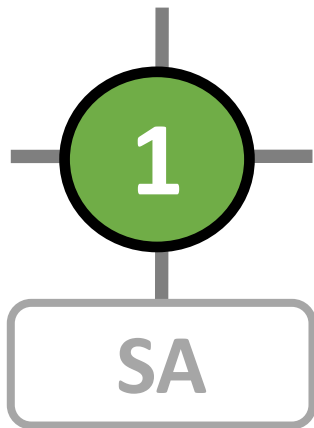
Row Decoder

SA   SA   SA   SA   SA   SA   SA

Commodity DRAM chips can *already* perform D-RaNGe

D-RaNGe Implementation

Identify four DRAM cells that fail randomly in a cache block

RNG Cell

1

SA

SAFARI kasırga **[Kim+ HPCA'19]**

D-RaNGe Implementation

Periodically generate true random numbers
by accessing the identified cache block

- Reduce access latency

- 1 KiB random number buffer in POC

- Programmers read random numbers from the
  data register using the `rand_dram()` function call

> **190 lines of Verilog code**
> **74 lines of C++ code**

Evaluation

Methodology: Microbenchmark that reads true random numbers
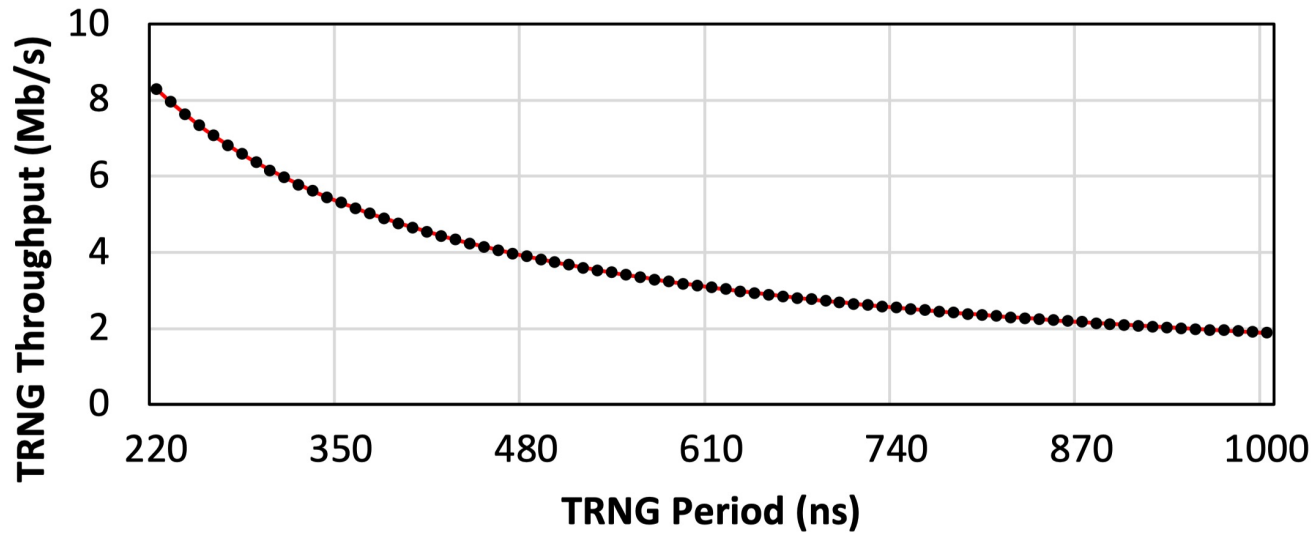
PiDRAM's D-RaNGe generates true random numbers at 8.30 Mb/s throughput

Outline

Background

    Commodity DRAM Based PiM Techniques

PiDRAM

    Overview

    Hardware & Software Components

    FPGA Prototype

Case Studies

    Case Study #1 – RowClone

    Case Study #2 – D-RaNGe

Conclusion

Executive Summary

Motivation: Commodity DRAM based PiM techniques improve the performance and energy efficiency of computing systems at no additional DRAM hardware cost

Problem: Challenges of integrating these PiM techniques into real systems are not solved General-purpose computing systems, special-purpose testing platforms, and system simulators *cannot* be used to efficiently study system integration challenges

Goal: Design and implement a flexible framework that can be used to:
- solve system integration challenges
- analyze trade-offs of end-to-end implementations of commodity DRAM-based-PiM techniques

Key idea: PiDRAM, an FPGA-based framework that enables:
- system integration studies
- end-to-end evaluations of PIM techniques using real unmodified DRAM chips

Evaluation: End-to-end integration of two PiM techniques on PiDRAM's FPGA prototype

Case Study #1 – RowClone: In-DRAM bulk data copy operations
- 119x speedup for copy operations compared to CPU-copy with system support
- 198 lines of Verilog and 565 lines of C++ code over PiDRAM's flexible codebase

Case Study #2 – D-RaNGe: DRAM-based random number generation technique
- 8.30 Mb/s true random number generator (TRNG) throughput, 220 ns TRNG latency
- 190 lines of Verilog and 74 lines of C++ code over PiDRAM's flexible codebase

PiDRAM is Open Source

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

Extended Version on ArXiv

https://arxiv.org/abs/2111.00082

Computer Science > Hardware Architecture

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.

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 2021 Show more

PiDRAM
An FPGA-based Framework
for End-to-end Evaluation
of Processing-in-DRAM Techniques

Ataberk Olgun

Juan Gomez Luna   Konstantinos Kanellopoulos   Behzad Salami

Hasan Hassan   Oğuz Ergin   Onur Mutlu

SAFARI

ETH *zürich*

kasırga

TOBB ETÜ
University of Economics & Technology

BACKUP SLIDES

Accessing a DRAM Cell

wordline

capacitor

bitline

access transistor

Sense Amp

enable

50

Accessing a DRAM Cell

(1) enable wordline

wordline

(4) deviation in bitline voltage

V_{DD} $V_{DD} + \delta$

bitline

capacitor

access transistor

(2) connects cell to bitline

(3) **(6)** cell loses charge cell loses charge to bitline to bitline

Sense Amp

(5) enable sense amp

enable

SAFARI **kasırga** **[Seshadri+ MICRO'17]**
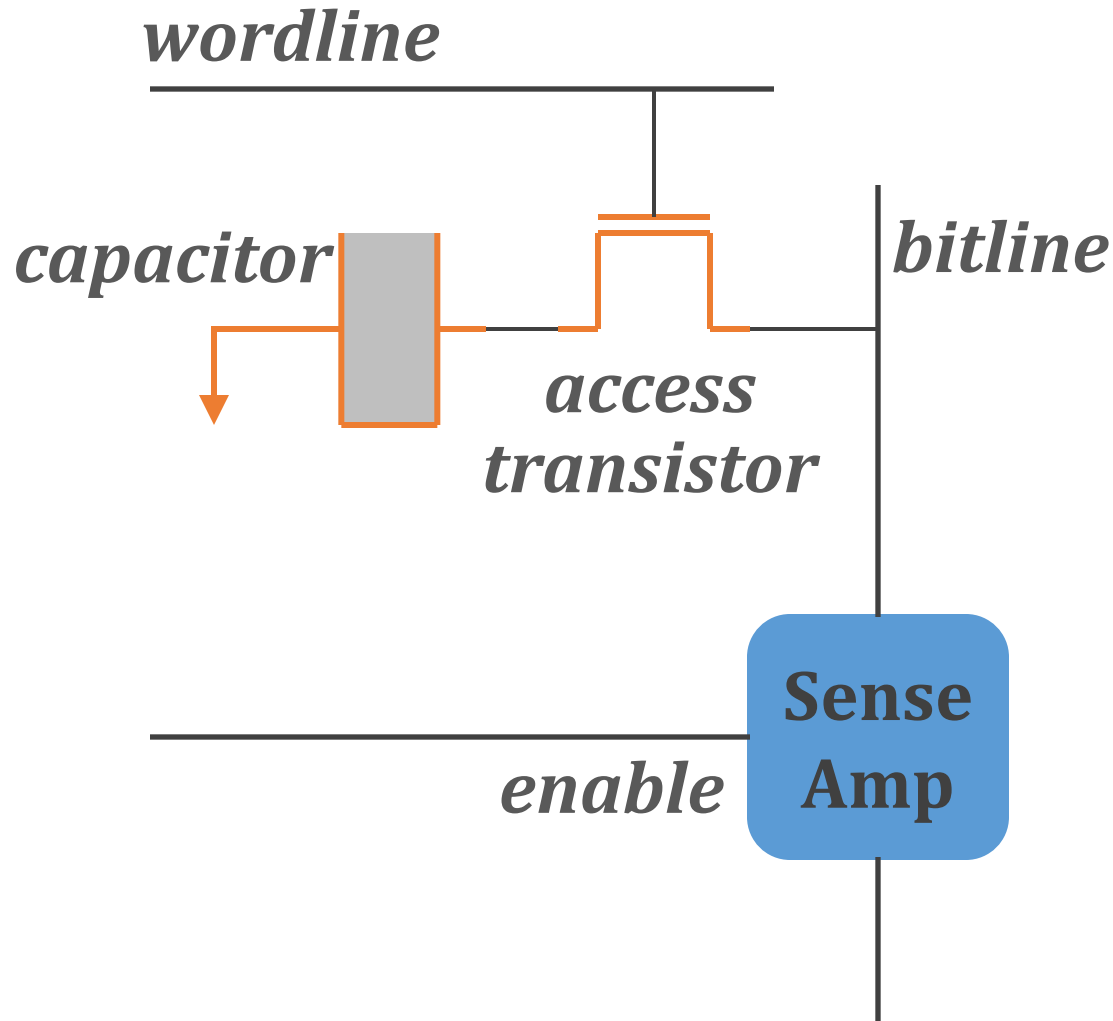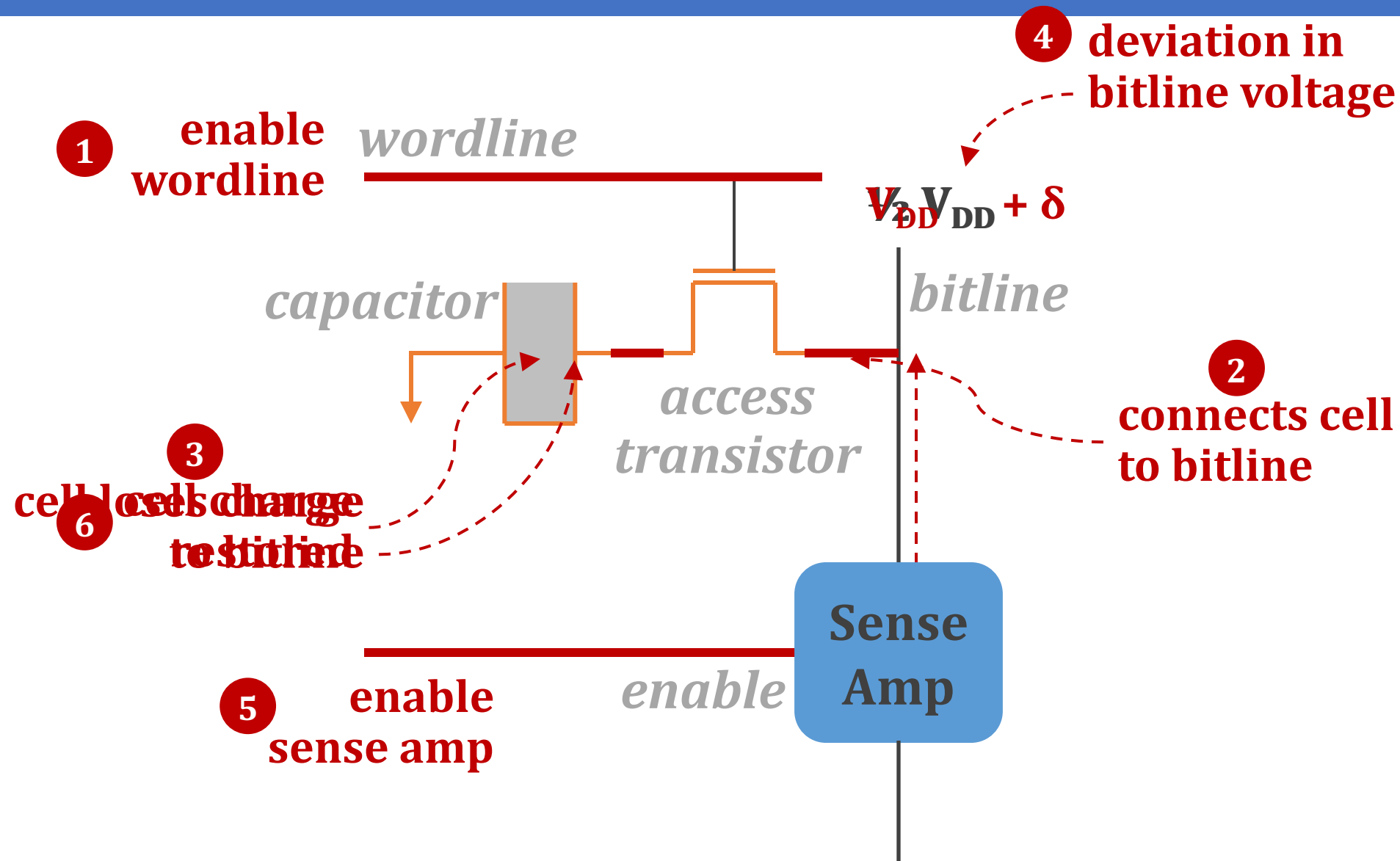
`alloc_align()` function

SubArray Mapping Table (SAMT) enables `alloc_align()`

SAMT

| Subarray 0 |
| Subarray 1 |
| ⋮ |
| Subarray N |

| SAMT Entry | ⋮ | Physical addresses of DRAM rows |

```
alloc_align(
4 KiB,
"Subarray 0")
```

① → **SAMT** → **②** → **Page Table**
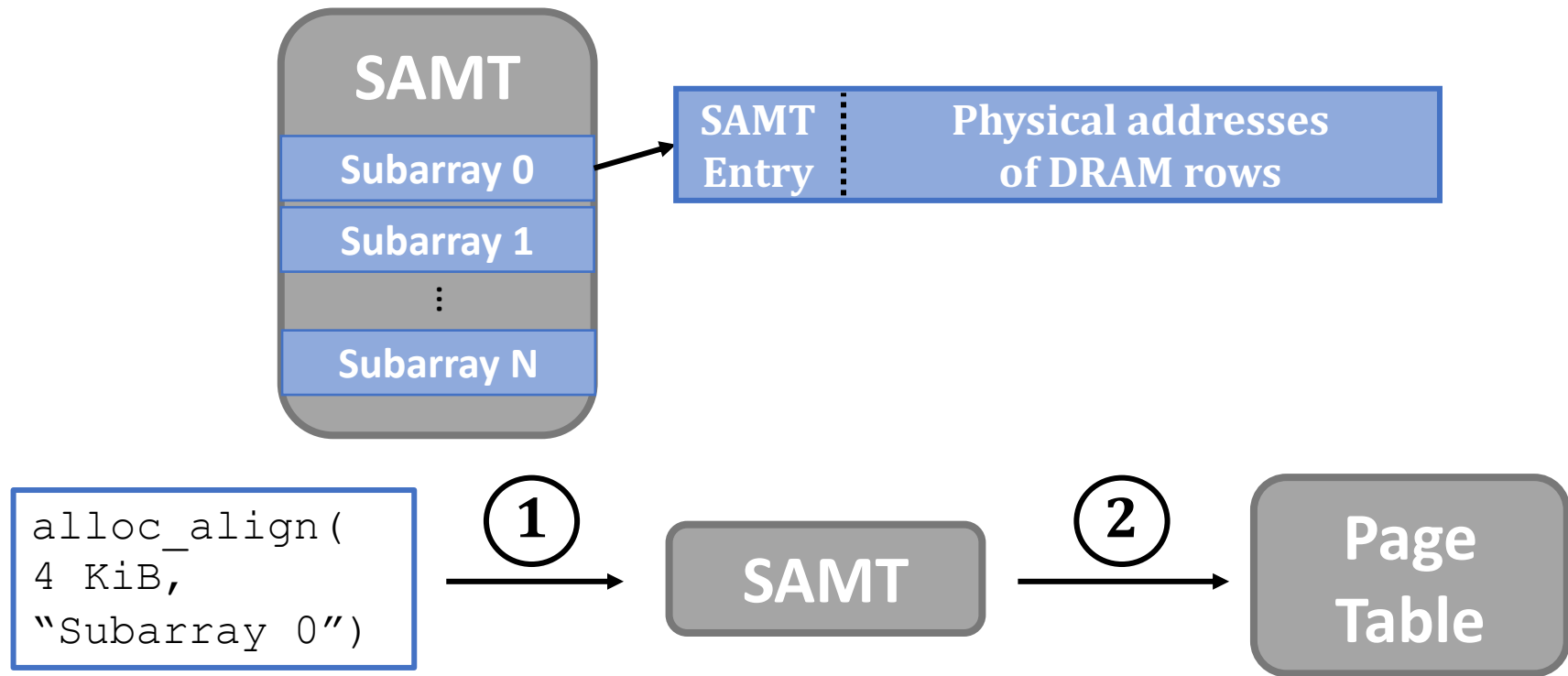
① Retrieve a physical address pointing to a DRAM row in subarray 0

② Update the page table to map programmer-allocated address to subarray 0

Initializing SAMT

SAMT

| Subarray 0 |
| Subarray 1 |
| ... |
| Subarray N |

SAMT Entry : Physical addresses of DRAM rows

?
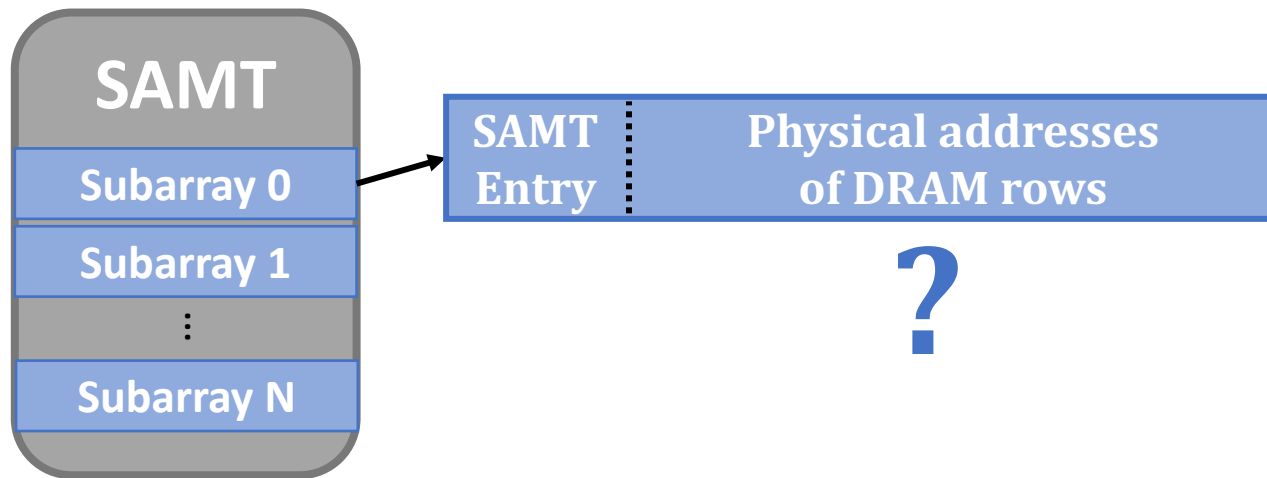
Perform in-DRAM copy using every DRAM row address as source and destination rows

If the in-DRAM copy operation succeeds source and destination rows are in the same subarray

② *Allocate 128 KiB A and B to same subarray*

```
A = alloc_align(128*1024, 0);
B = alloc_align(128*1024, 0);
```

① Characterize RowClone Success Rate

Initialize Subarray Mapping Table

⑧ *Copy 128 KiBs from A to B*

```
rcc(A, B, 128*1024);
```

Access page table to find source and destination DRAM rows

③

VA_A00	Bank 0
VA_A01	Bank 1
VA_A31	Bank 7

Array A

Array B

VA_B00	Bank 0
VA_B31	Bank 7

Arrays are split into 4KB blocks

④

Allocation ID Table

| Bank 7 |
| 0 → SA0 |
| 1 → SA3 |

⑤

Subarray Mapping Table

| Bank 7 |
| Entry – SA0 |
| Entry – SA1 |

⑥

VA_A00   VA_A16

| PA_11 | PA_12 | PA_21 | PA_22 | ... |

| 4 KB | 4 KB |

DRAM ROW

Consecutive blocks are assigned to DRAM rows in different DRAM banks

⑦

Page Table

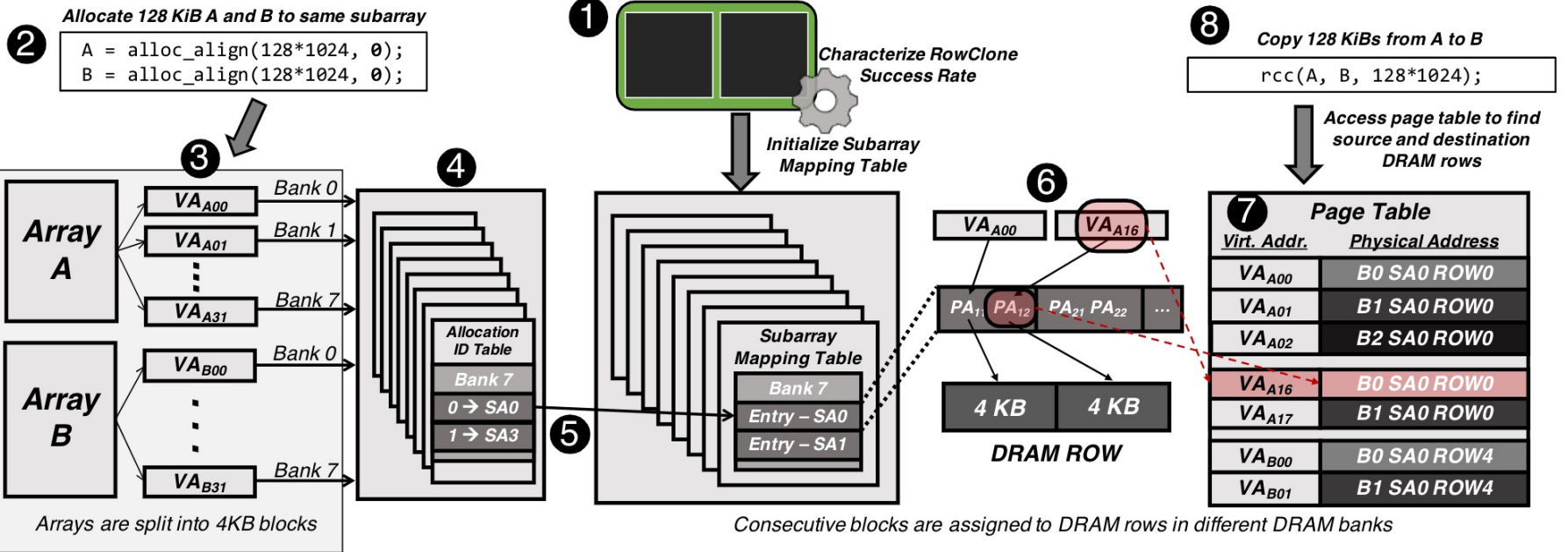| Virt. Addr.	Physical Address
VA_A00	B0 SA0 ROW0
VA_A01	B1 SA0 ROW0
VA_A02	B2 SA0 ROW0
VA_A16	B0 SA0 ROW0
VA_A17	B1 SA0 ROW0
VA_B00	B0 SA0 ROW4
VA_B01	B1 SA0 ROW4

Table 1: PuM techniques that can be studied using PiDRAM. PuM techniques that we implement in this work are highlighted in bold

| PuM Technique	Description	Integration Challenges
RowClone [91]	Bulk data-copy and initialization within DRAM	(i) *memory allocation and alignment mechanisms* that map source & destination operands of a copy operation into same DRAM subarray; (ii) *memory coherence*, i.e., source & destination operands must be up-to-date in DRAM.
D-RaNGe [62]	True random number generation using DRAM	(i) periodic generation of true random numbers; (ii) *memory scheduling policies* that minimize the interference caused by random number requests.
Ambit [89]	Bitwise operations in DRAM	(i) *memory allocation and alignment mechanisms* that map operands of a bitwise operation into same DRAM subarray; (ii) *memory coherence*, i.e., operands of the bitwise operations must be up-to-date in DRAM.
SIMDRAM [43]	Arithmetic operations in DRAM	(i) *memory allocation and alignment mechanisms* that map operands of an arithmetic operation into same DRAM subarray; (ii) *memory coherence*, i.e., operands of the arithmetic operations must be up-to-date in DRAM; (iii) *bit transposition*, i.e., operand bits must be laid out vertically in a single DRAM bitline.
DL-PUF [61]	Physical unclonable functions in DRAM	*memory scheduling policies* that minimize the interference caused by generating PUF responses.
QUAC-TRNG [82]	True random number generation using DRAM	(i) periodic generation of true random numbers; (ii) *memory scheduling policies* that minimize the interference caused by random number requests; (iii) efficient integration of the SHA-256 cryptographic hash function.
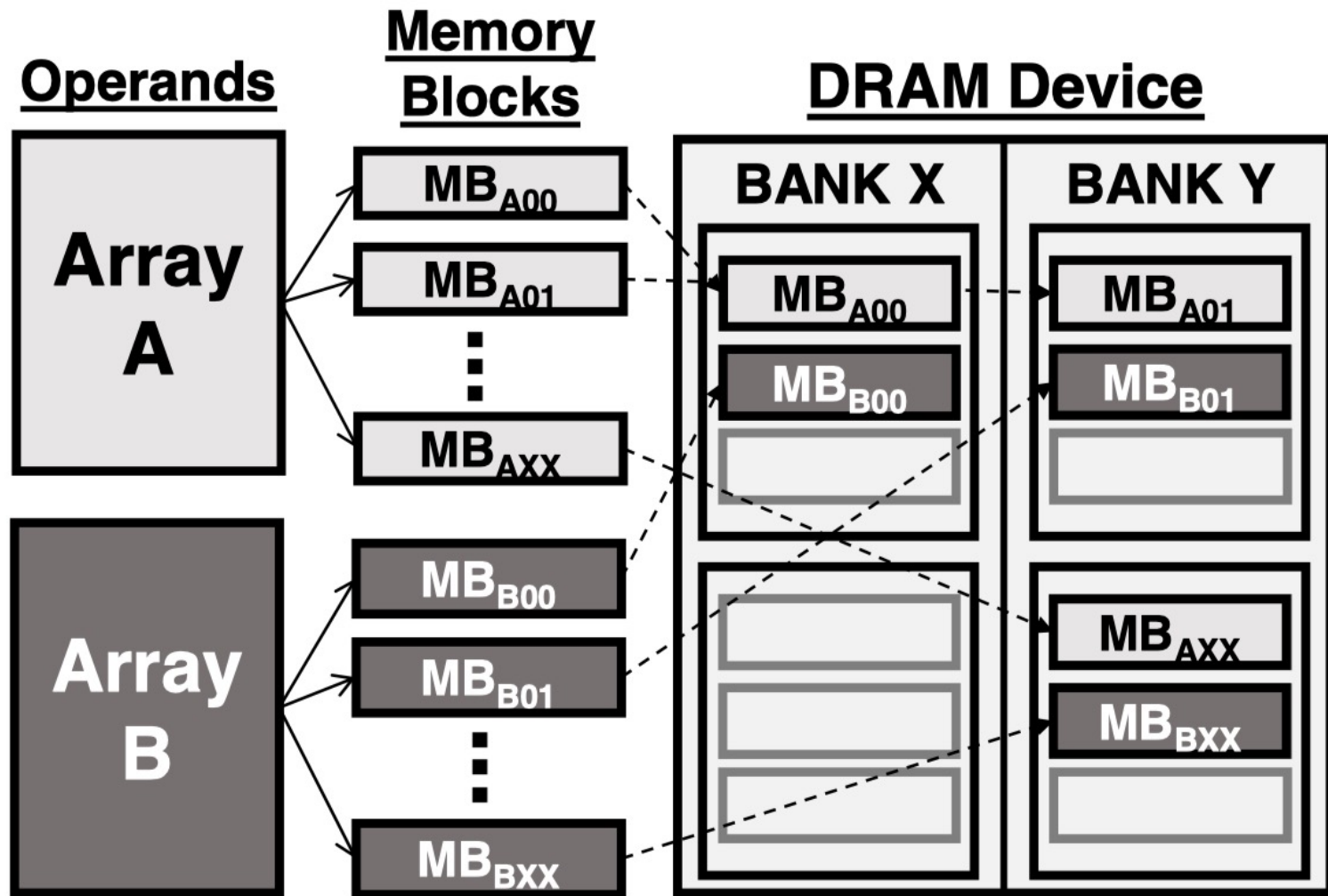
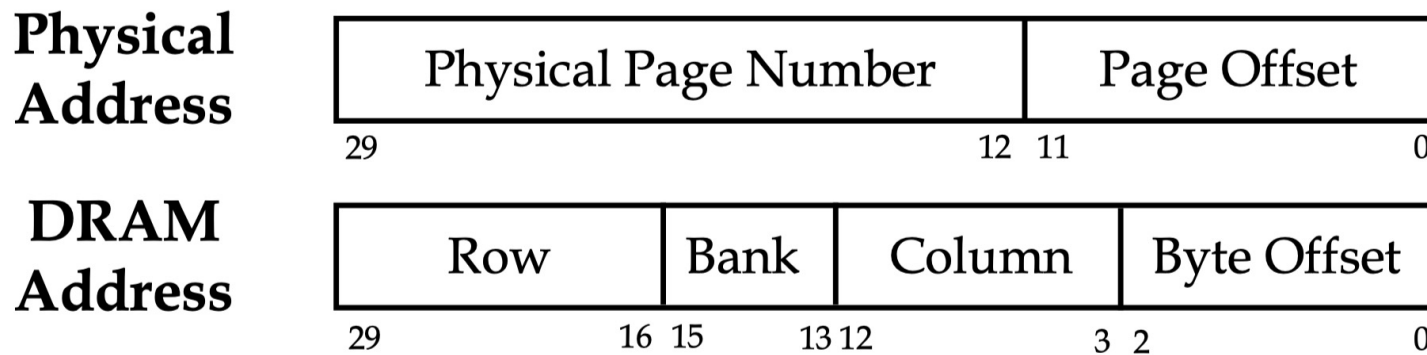Figure 6: Overview of our memory allocation mechanism

Figure 8: Physical address to DRAM address mapping in PiDRAM. Byte offset is used to address the byte in the DRAM burst.
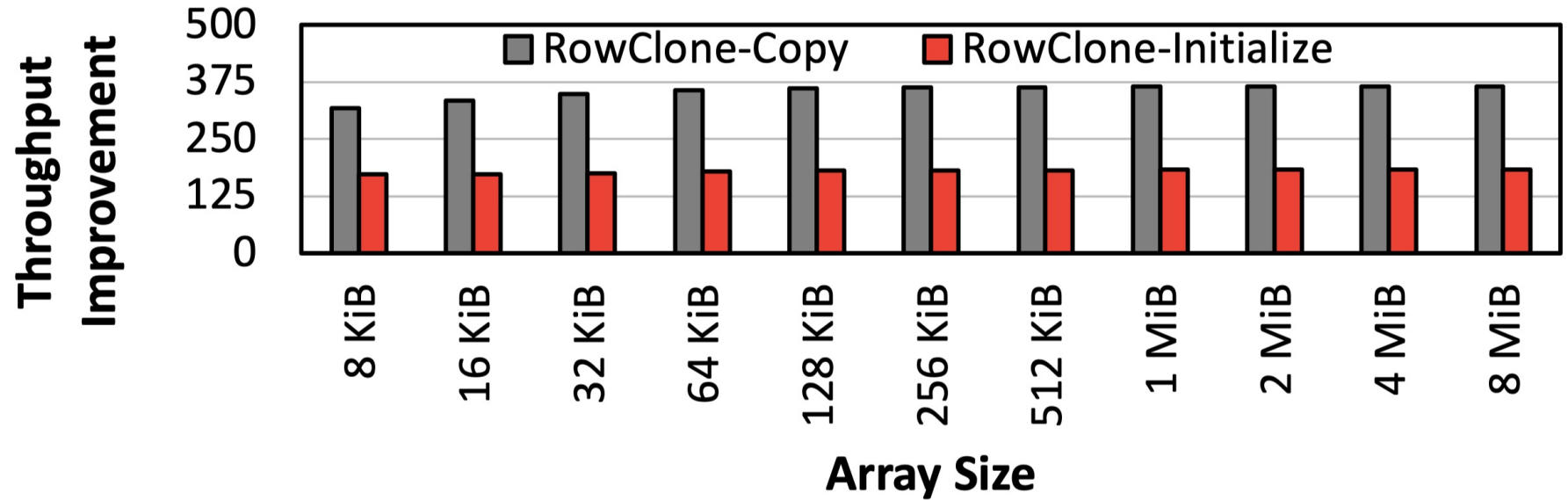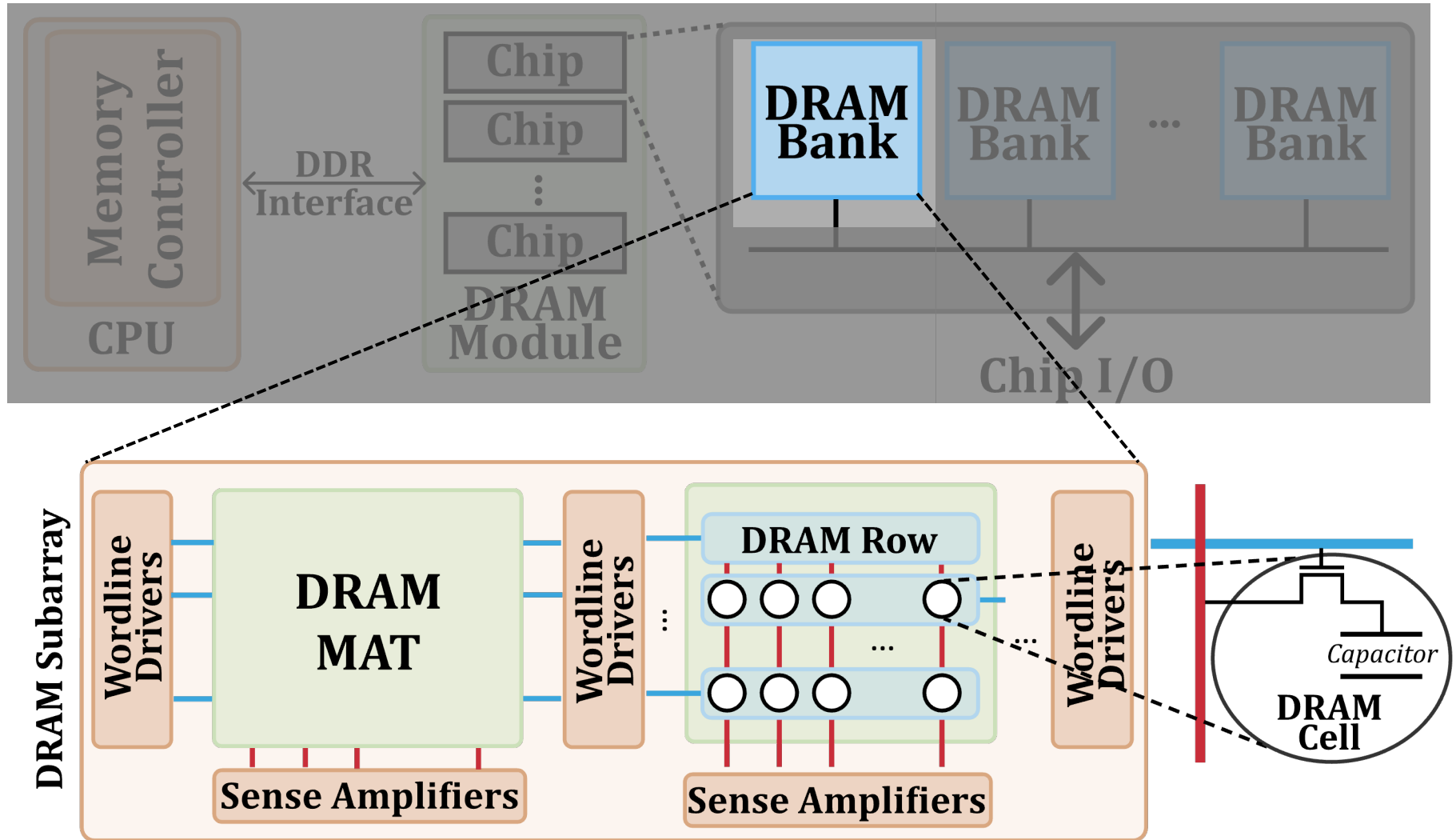
Figure 9: RowClone-Copy and RowClone-Initialize over traditional CPU-copy and -initialization for the Bare-Metal configuration

Table 4: Comparison of PiDRAM with related state-of-the-art prototyping and evaluation platforms

Platforms	Interface with real DRAM chips	Flexible MC for PuM	System software support	Open-source
Silent-PIM [78]	✗	✗	✓	✗
SoftMC [60]	✓(DDR3)	✗	✗	✓
ComputeDRAM [44]	✓(DDR3)	✗	✗	✗
MEG [174]	✓(HBM)	✗	✓	✓
PiMulator [119]	✗	✓	✗	✓
Commercial platforms (e.g., ZYNQ [166])	✓(DDR3/4)	✗	✓	✗
Simulators [18, 35, 90, 132, 140, 169, 170, 175]	✗	✓	✓(potentially)	✓
PiDRAM (this work)	✓(DDR3)	✓	✓	✓

DRAM Organization

[Olgun+ ISCA'21]

DRAM Operation

(Activation Latency)   (Access Latency)
(Precharge Latency)

DRAM Command Sequence

| ACT R0 | RD | RD | RD | PRE R0 | ACT R1 | RD | RD | RD |

time

[Kim+ HPCA'19]