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

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 commodity DRAM based 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

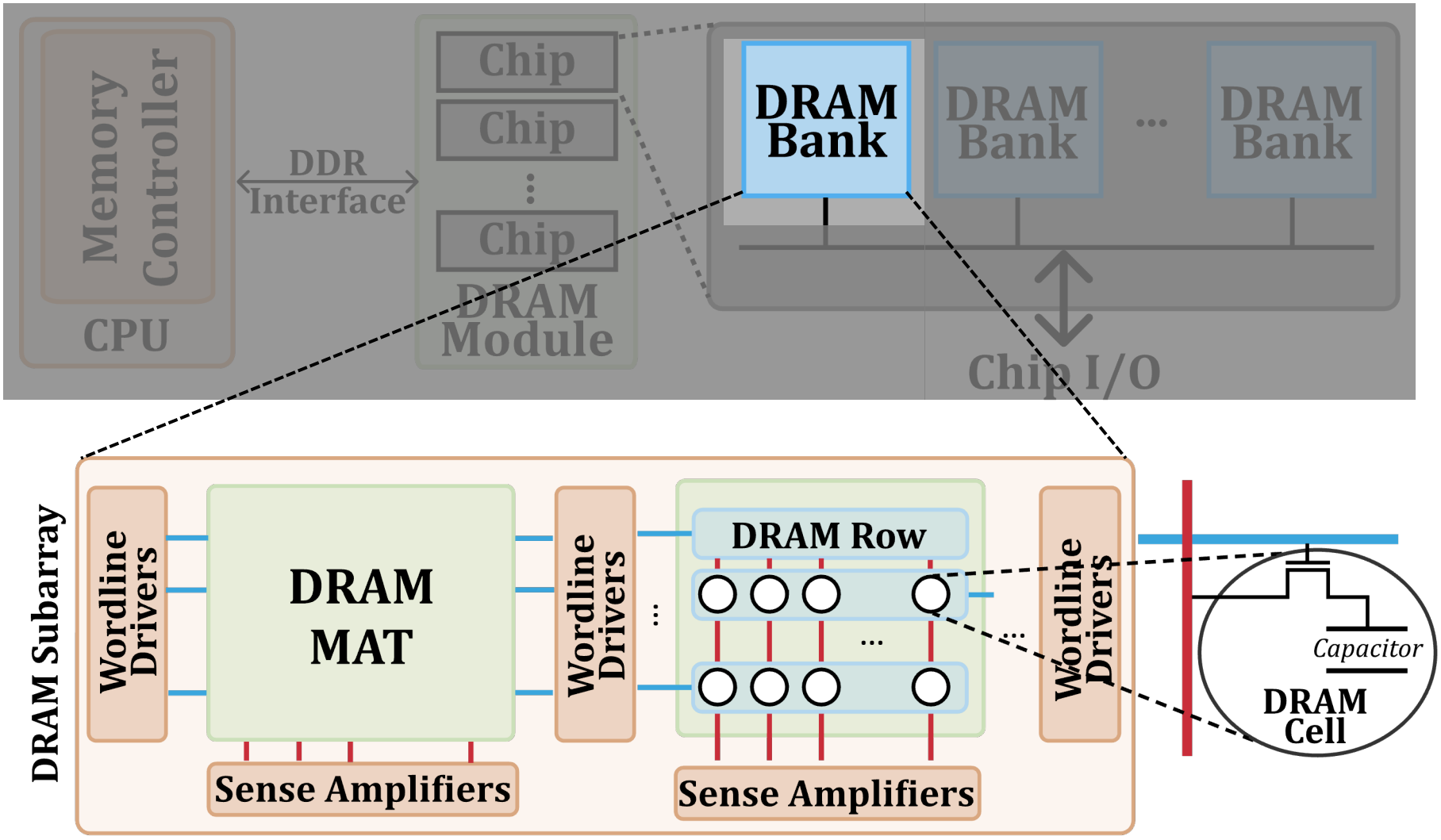
- 119x speedup for copy operations compared to CPU-copy with system support
- 198 lines of Verilog and 565 lines of C++ code over PiDRAM's flexible codebase

Case Study #2 – D-RaNGe: DRAM-based random number generation technique

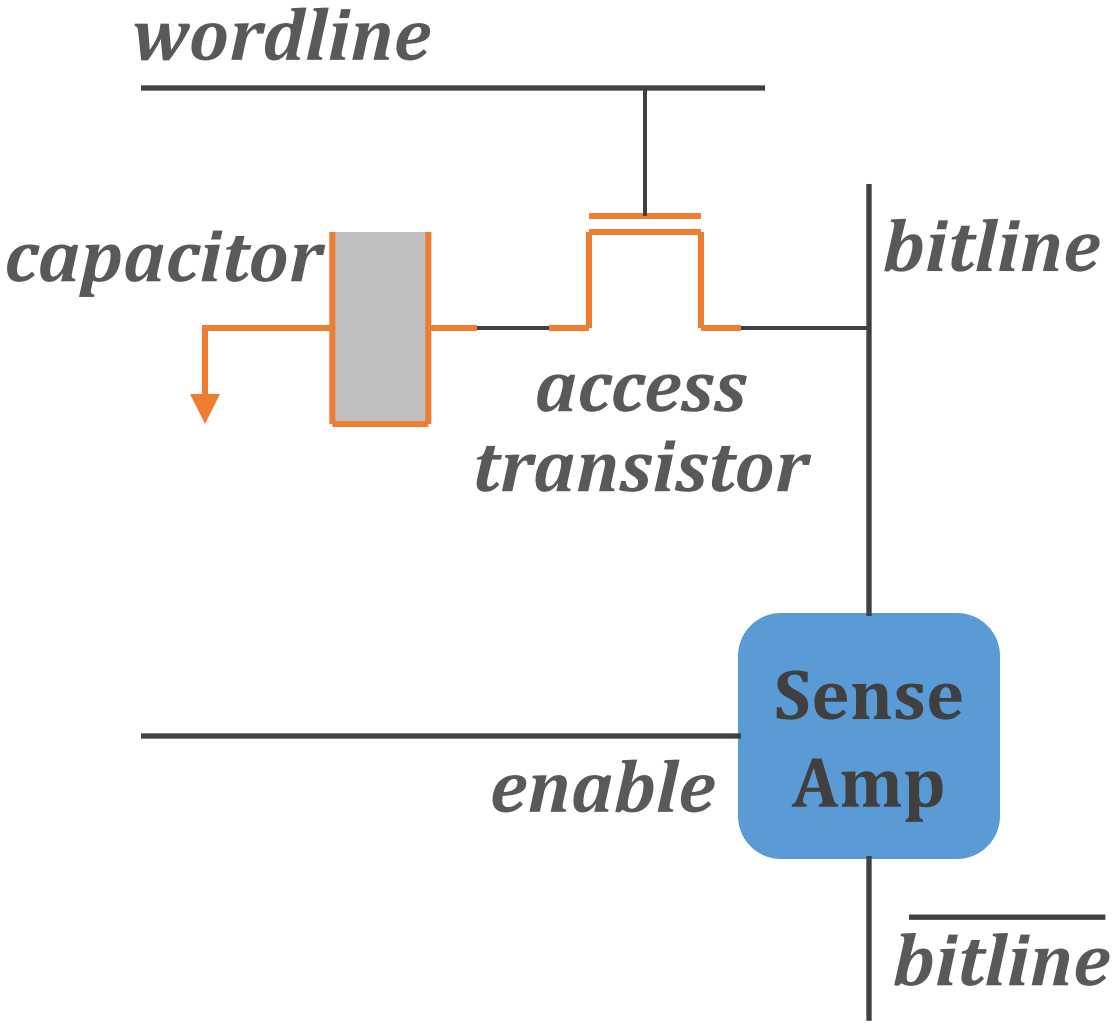
- 8.30 Mb/s true random number generator (TRNG) throughput, 220 ns TRNG latency
- 190 lines of Verilog and 78 lines of C++ code over PiDRAM's flexible codebase

Outline

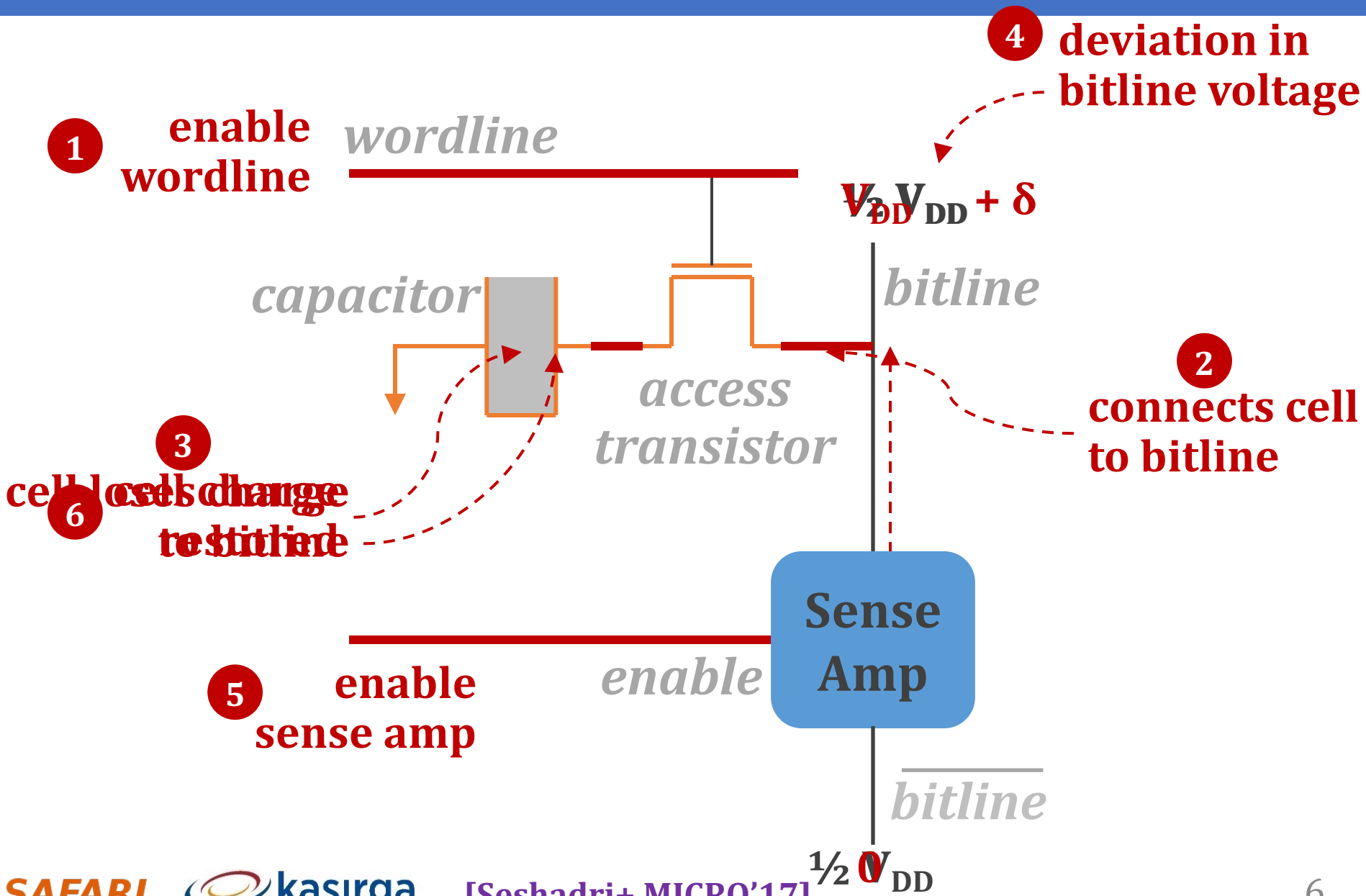
DRAM Organization



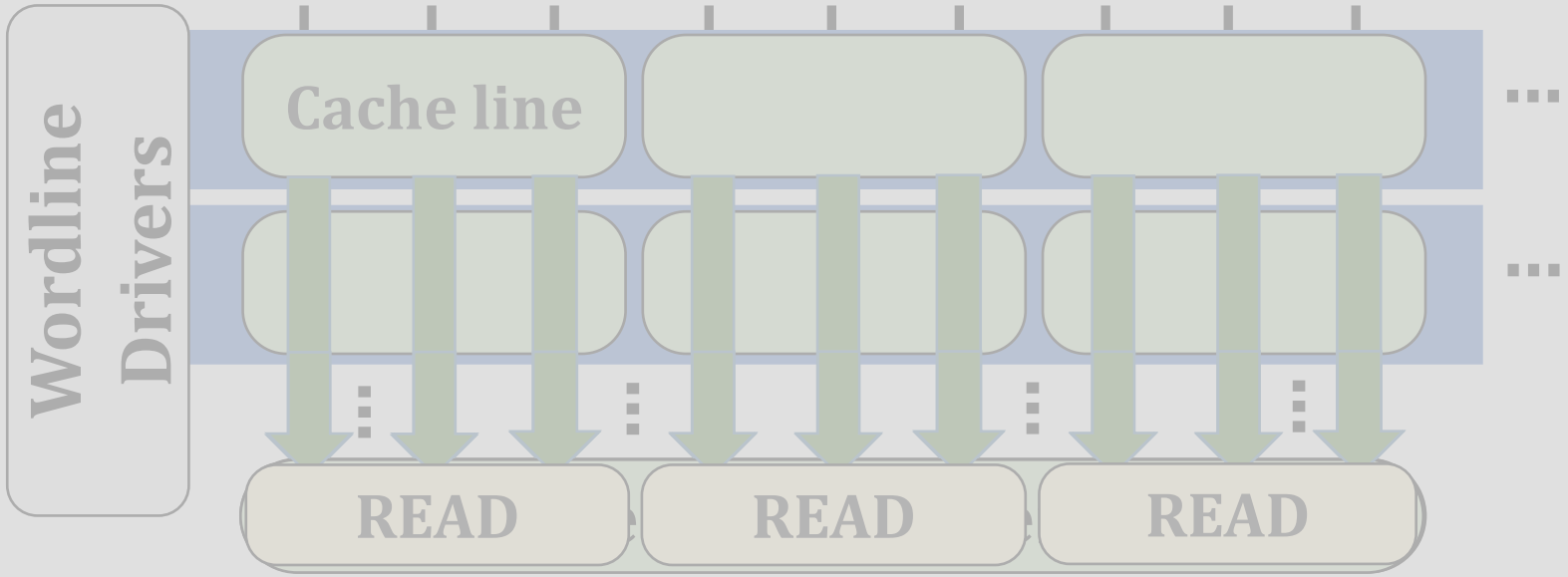
Accessing a DRAM Cell

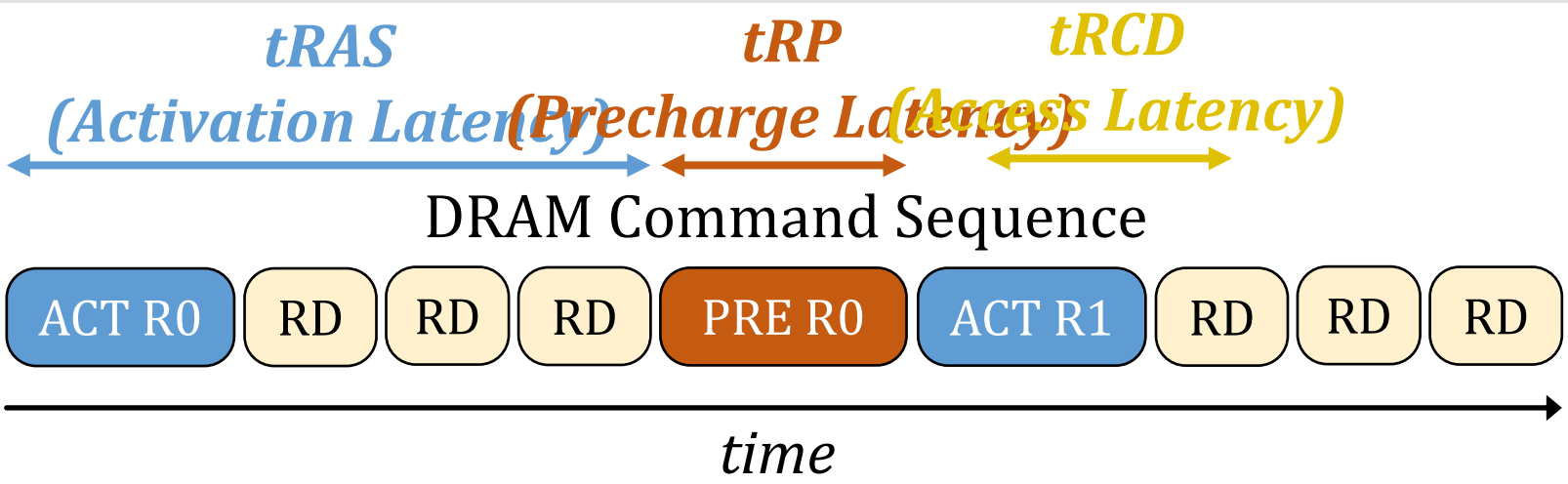


Accessing a DRAM Cell



DRAM Operation





Outline

Processing In Memory Techniques

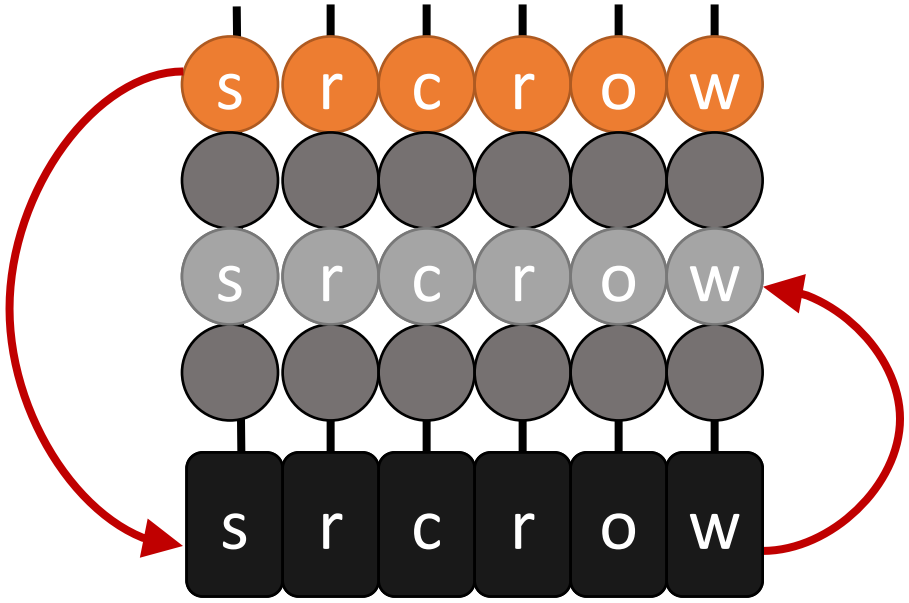
Use operational principles of commodity DRAM chips
to perform bulk data movement and computation in memory

Two relevant examples:

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

2) **D-RaNGe:** In-DRAM true random number generation (TRNG) using access latency (tRCD) failures

In-DRAM Copy: Key Idea (RowClone)



Row Buffer

1. Source row to row buffer

2. Row buffer to destination row

Outline

- Background
  - DRAM Organization and Operation
  - Commodity DRAM Based PiM Techniques
- **PiDRAM**
  - **Overview**
  - Hardware & Software Components
  - FPGA Prototype
- Case Studies
  - Case Study #1 – RowClone
- Conclusion

PiDRAM: Overview (I)

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

Identify key components shared across PiM techniques

Implement customizable key components:

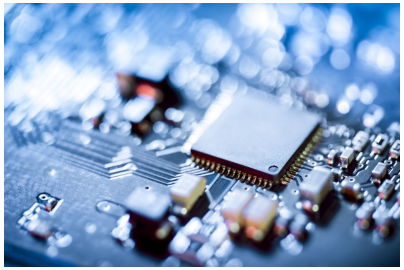
- Provide modularity, enhance extensibility of the framework

Common basis to enable system support for PiM techniques

PiDRAM: Overview (II)

Identify and develop four key hardware and software components

Hardware



1. Flexible PiM Ops. Controller
2. Easy-to-extend Memory Controller

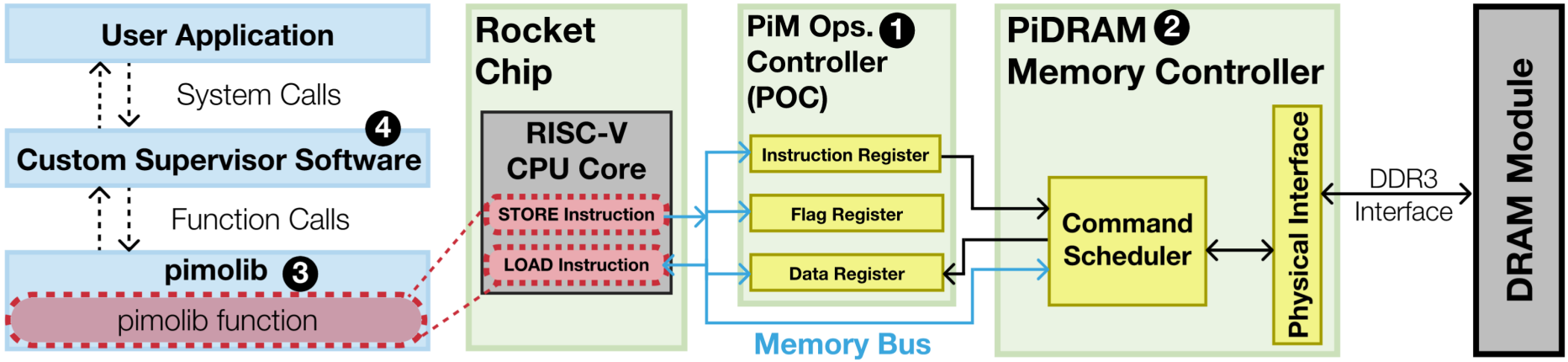
Software

3. Extensible Software Library
4. Custom Supervisor Software

PiDRAM: System Design

Key components are attached to a real computing system

- PiM Ops. Controller and PiDRAM Memory Controller is implemented within the hardware system
- Custom supervisor software runs on the hardware system
- Extensible software library is used by the supervisor software

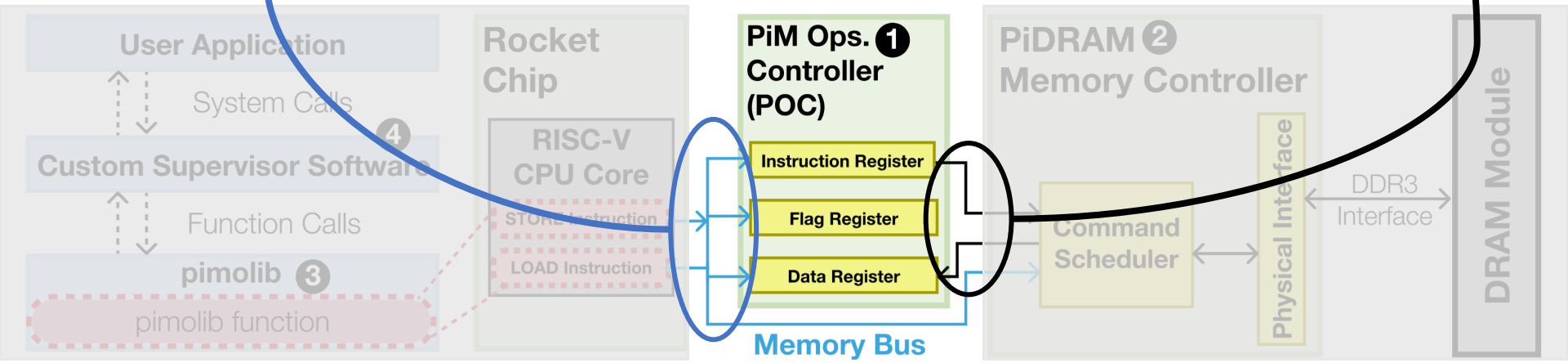


PiM Operations Controller (POC)

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

Receive instructions over memory-mapped interface
(portable to other systems with different CPU ISAs)

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



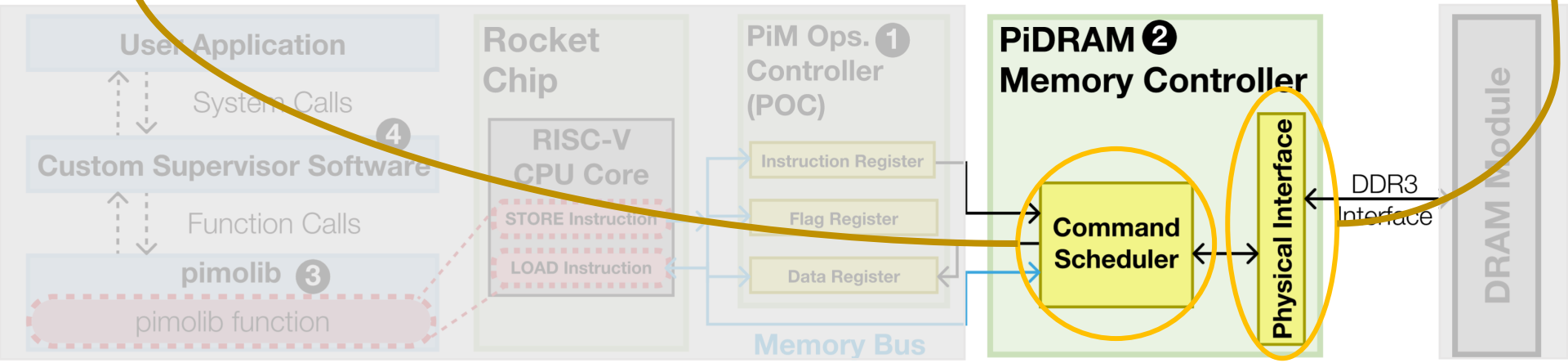
PiDRAM Memory Controller

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

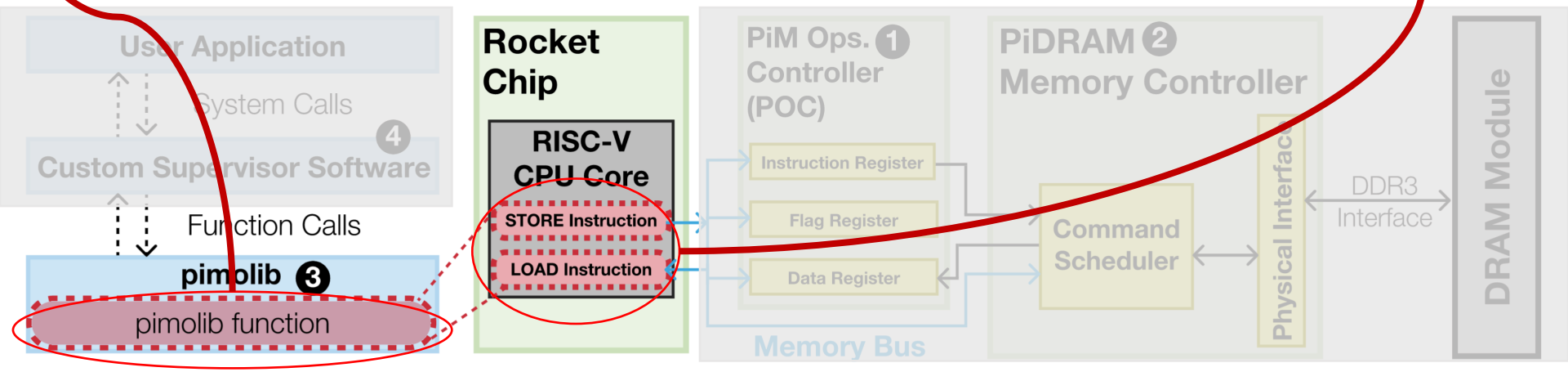
Controls the physical DDR3 interface
Receives commands from command scheduler & operates DDR3 pins



PiM Operations Library (pimolib)

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

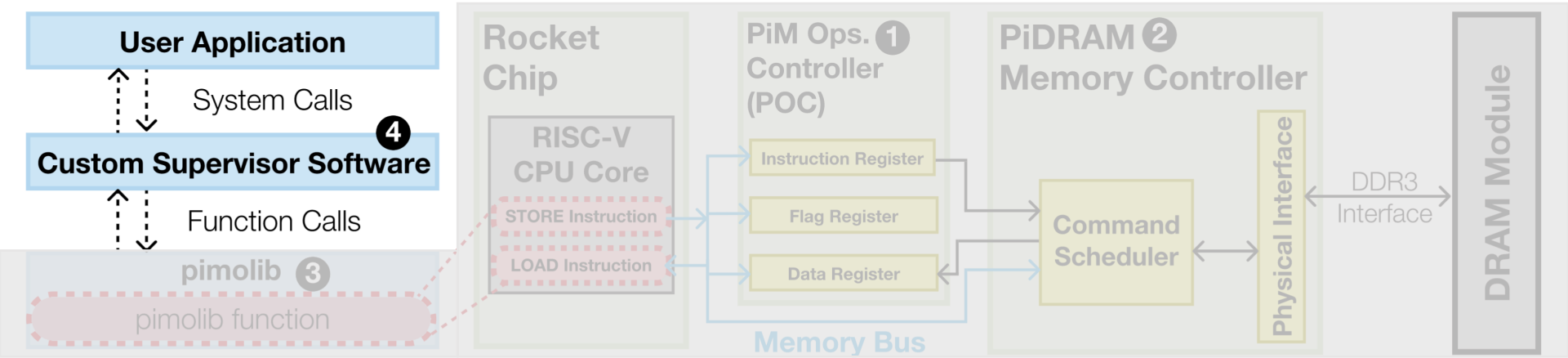
Executes LOAD & STORE requests to communicate with the POC



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)



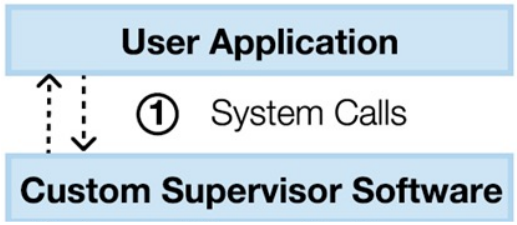
PiM Operation Execution Flow

`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



PiM Operation Execution Flow

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

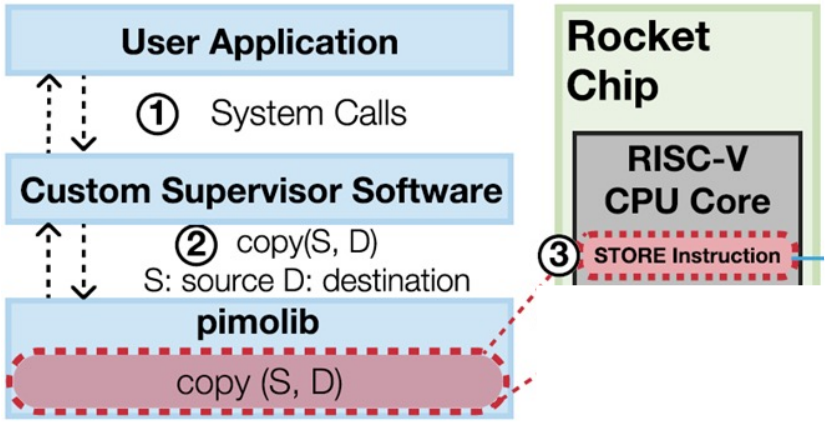
4. The first store updates the *instruction* register with `Copy(S, D)`

5. The second store sets the "Start" flag in the *flag* register

Start (S)

1

Start the execution of PiM operation

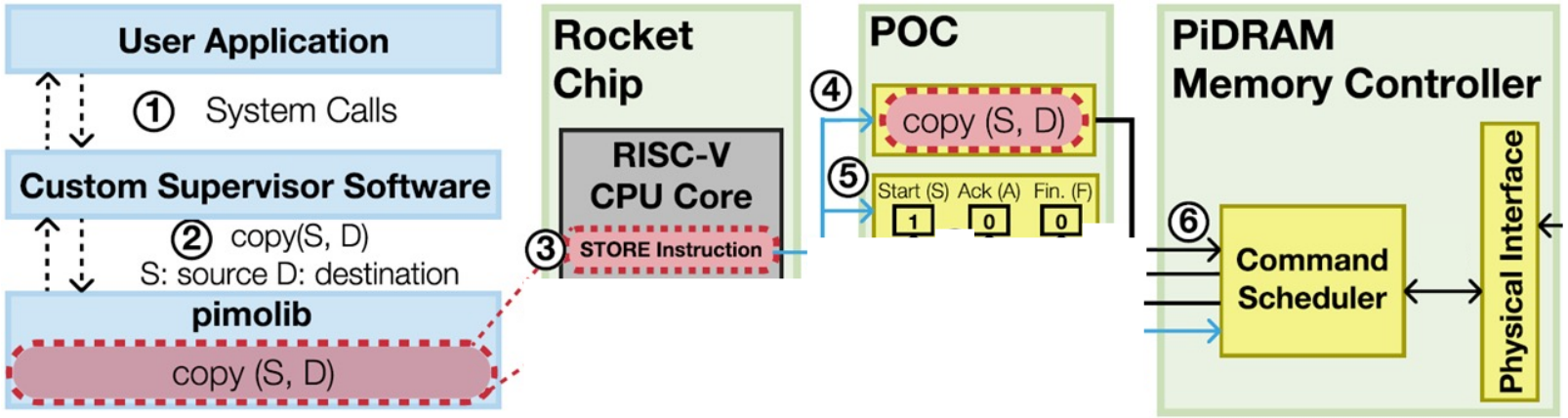


PiM Operation Execution Flow

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

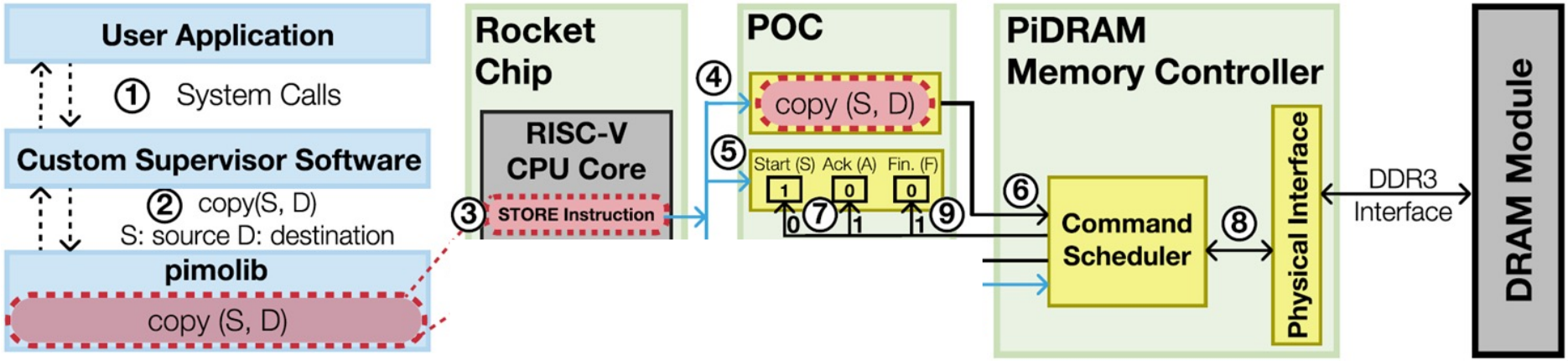


PiM Operation Execution Flow

9 The memory controller sets the "Fin." flag

10 `Copy(S, D)` periodically checks either "Ack" or "Fin." flags using LOAD instructions

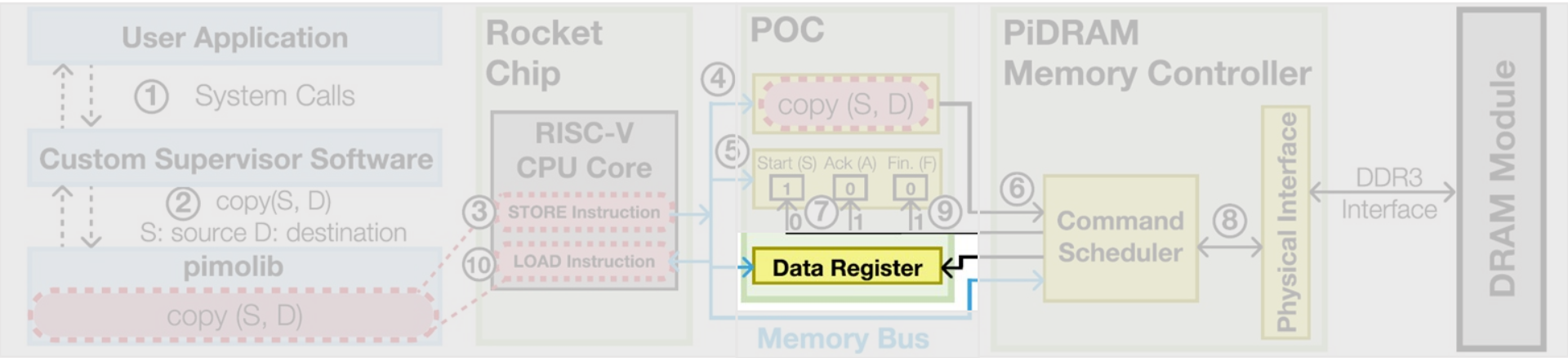
`Copy(S, D)` returns when the periodically checked flag is set



PiM Operation Execution Flow

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

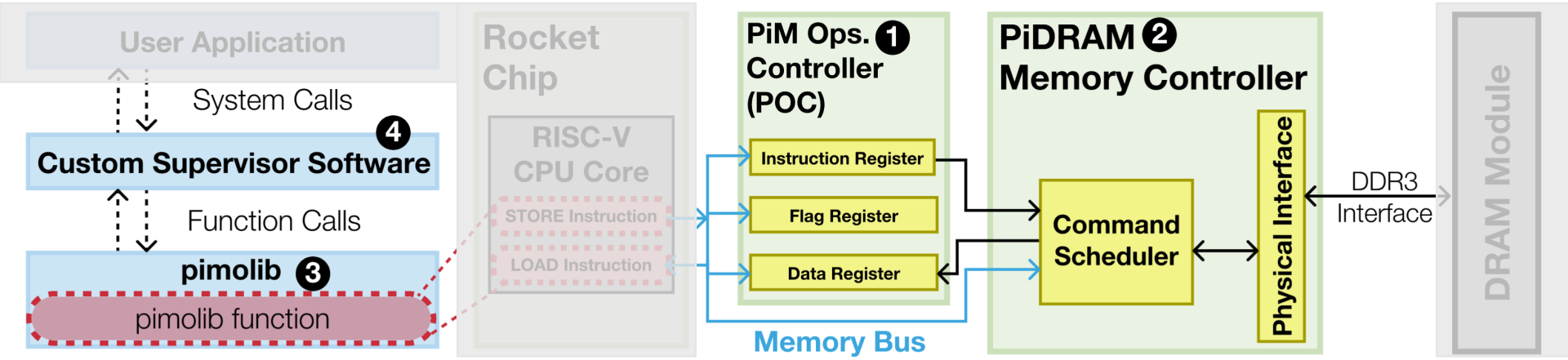
It is used to read true random numbers generated by D-RaNGe



PiDRAM Components Summary

Four key components orchestrate PiM operation execution

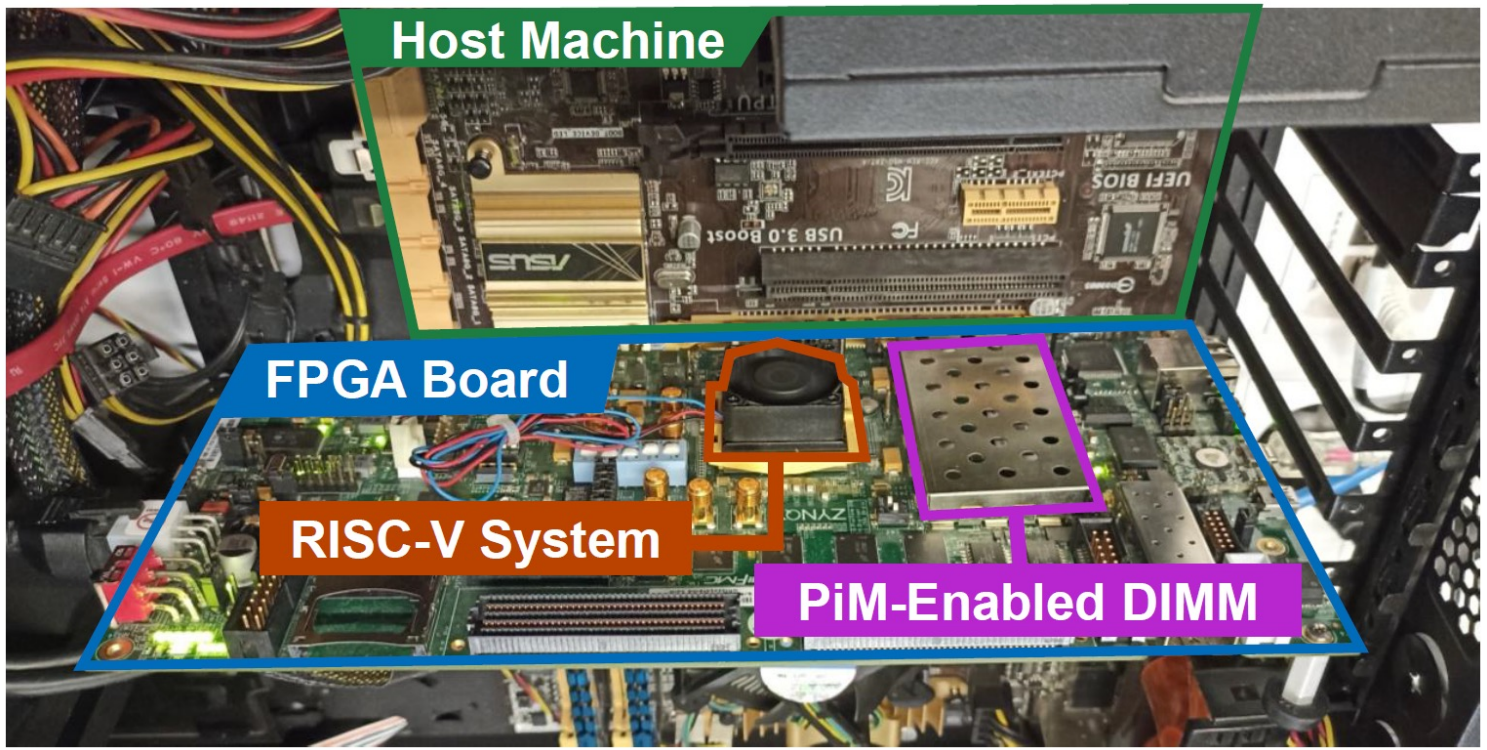
Four key components provide an extensible basis for end-to-end integration of PiM techniques



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:** Micron MT8JTF12864, 1 GiB, 8 banks



Outline

- Background
  - DRAM Organization and Operation
  - Commodity DRAM Based PiM Techniques
- PiDRAM
  - Overview
  - Hardware & Software Components
  - FPGA Prototype
- **Case Studies**
  - Case Study #1 – RowClone
- Conclusion

PiDRAM Case Studies

We conduct two case studies:

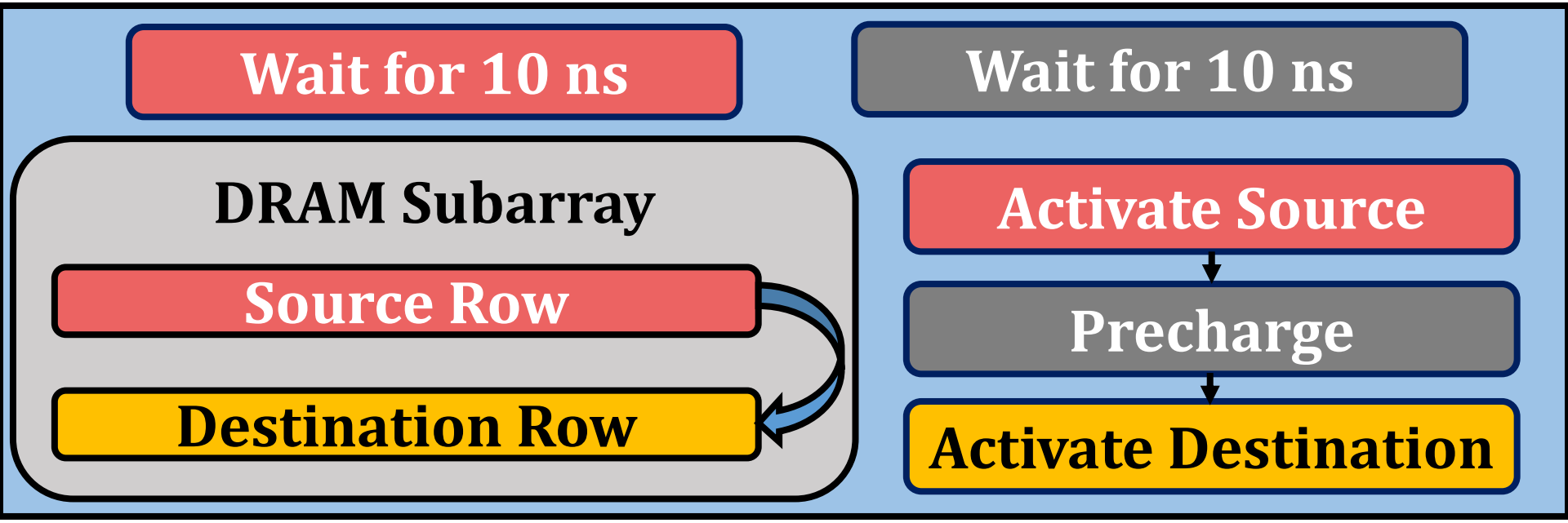
1. In-DRAM bulk data copy (RowClone)
2. In-DRAM true random number generation (D-RaNGe)

Demonstrate the flexibility and ease of use of PiDRAM

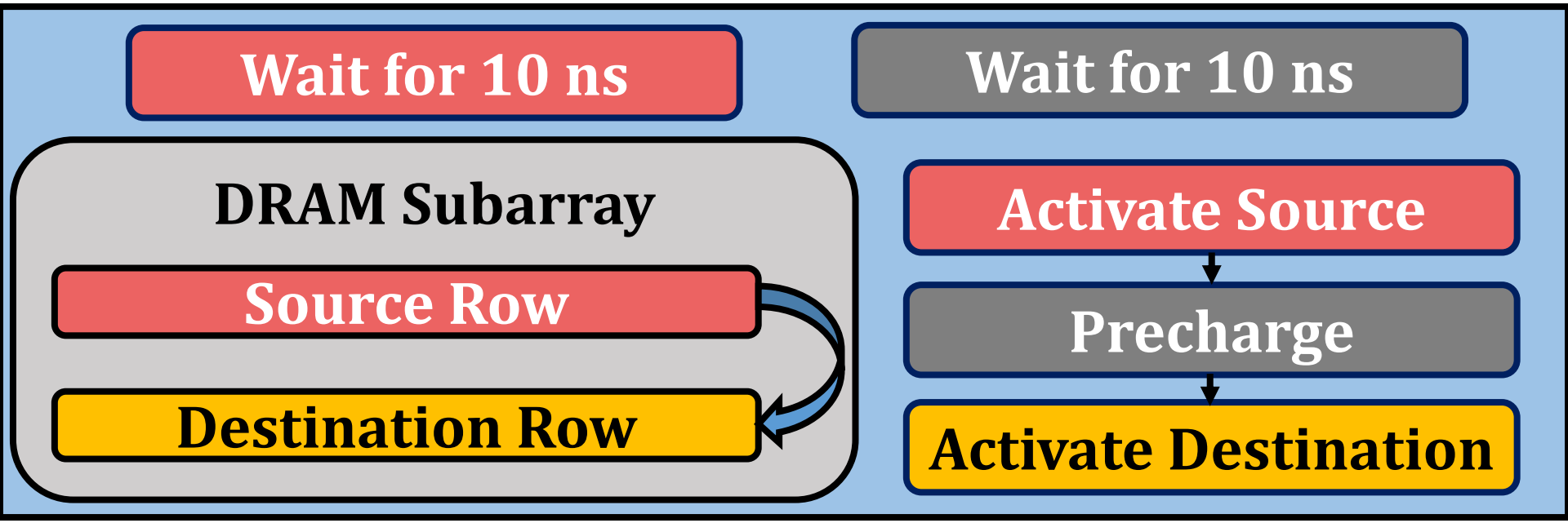
RowClone Implementation (I)

Key Idea: Perform in-DRAM copy operations by using carefully created DRAM command sequences

- ComputeDRAM [Gao+, MICRO'19] demonstrates in-DRAM copy operations in real DDR3 chips
- ACT → PRE → ACT command sequence with greatly reduced tRAS and tRP timing parameters



RowClone Implementation (II)



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

RowClone System Integration

Two challenges in integrating RowClone end-to-end in a real system

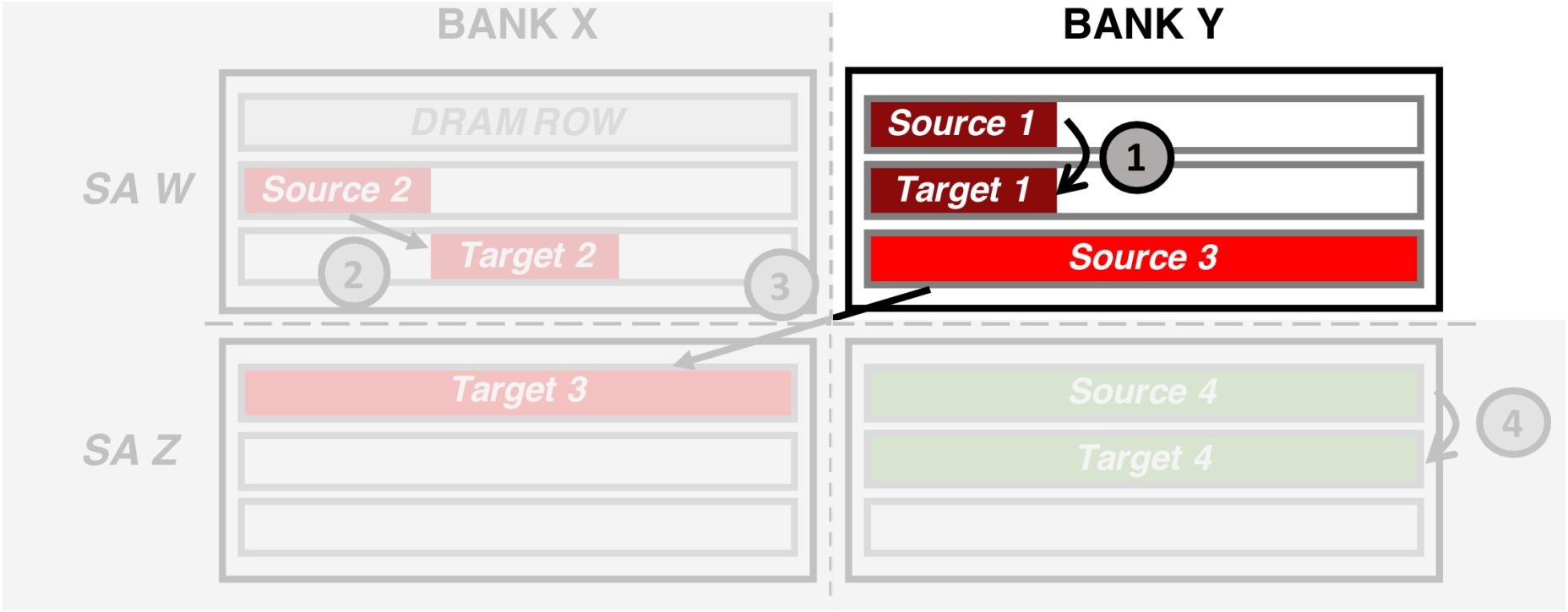
1. Memory allocation constraints (Intra-subarray operation)

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)

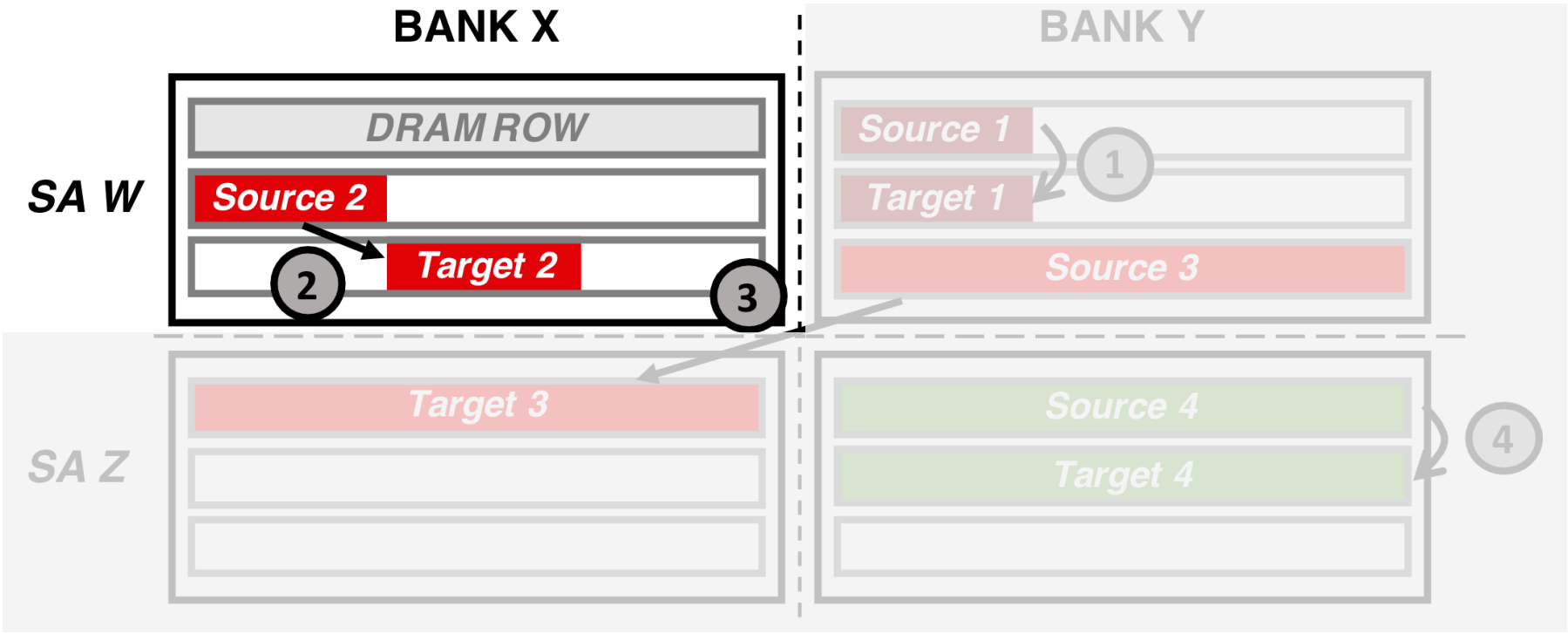
Memory allocation requirements



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

RowClone Memory Allocation (I)

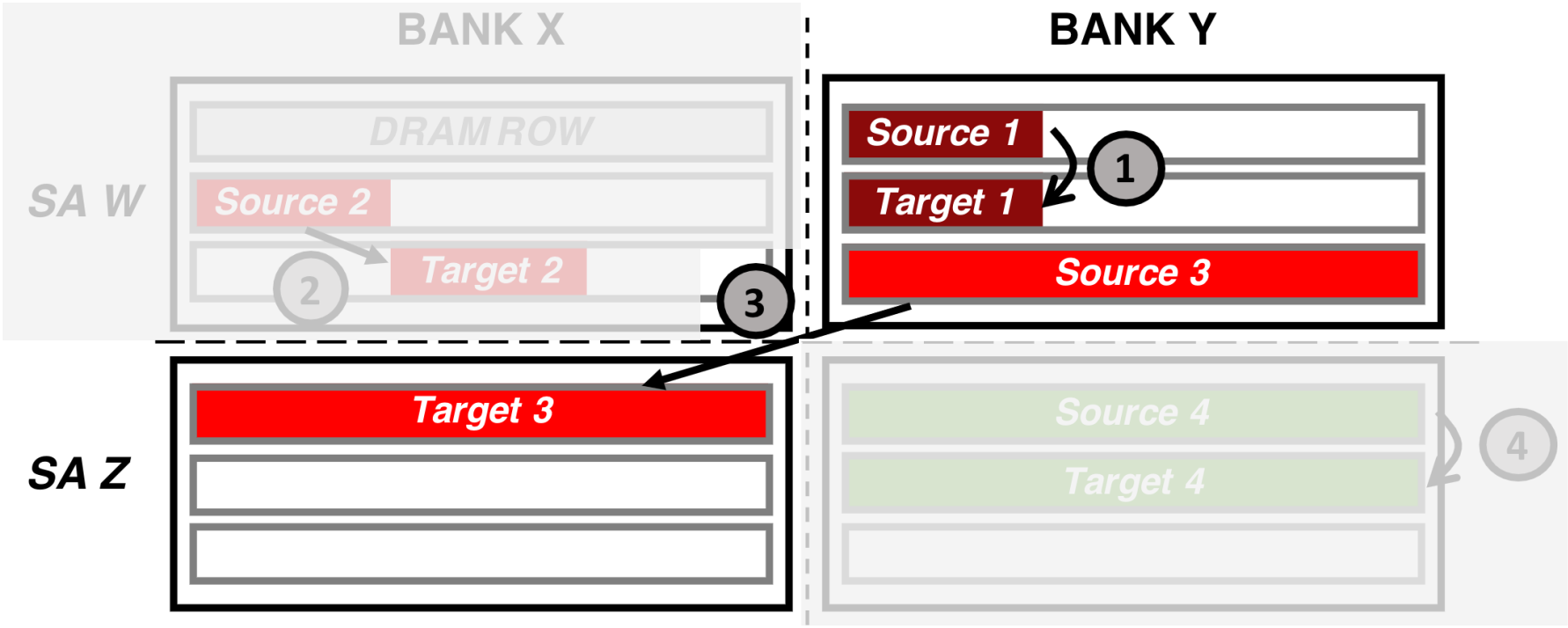
Memory allocation requirements



2 **Alignment:** Operands must be placed at the same offset

RowClone Memory Allocation (I)

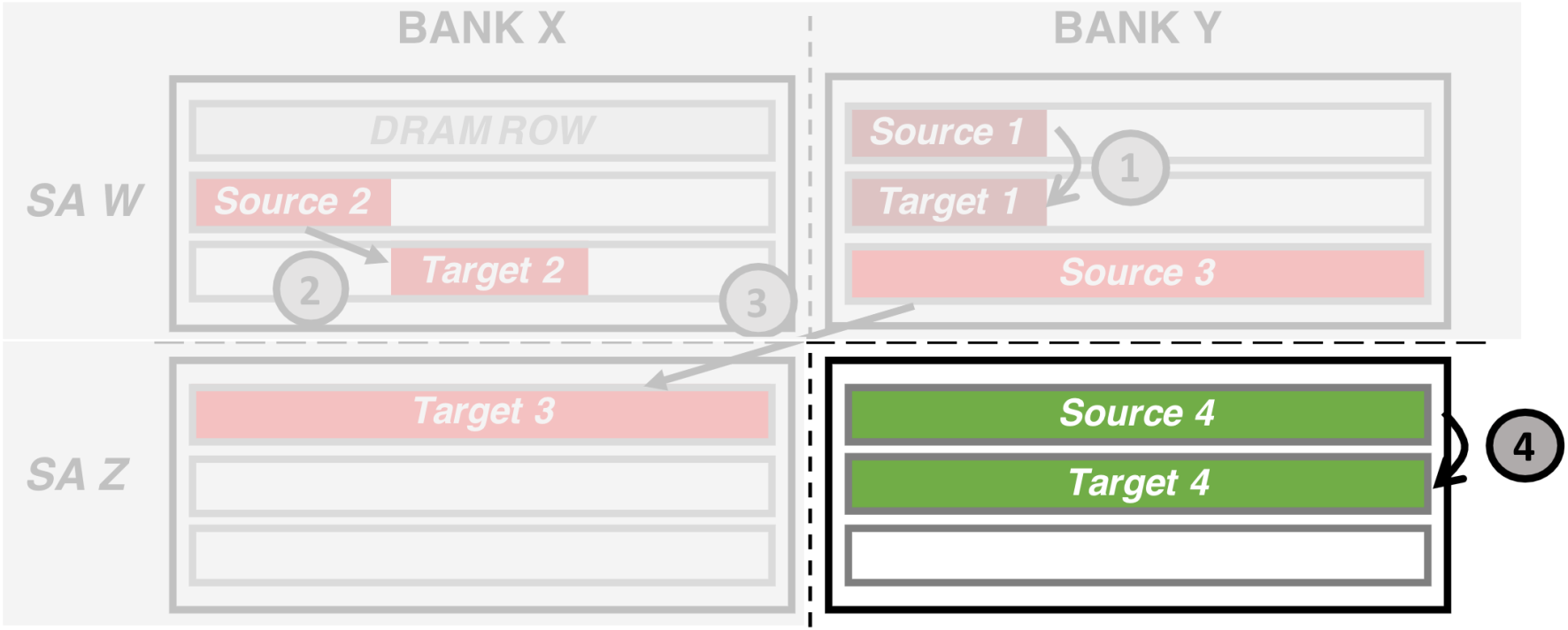
Memory allocation requirements



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)

To overcome the memory allocation challenges
implement a new memory allocation function

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

Key Mechanism: Distribute virtual pages to different banks
while mapping them to DRAM rows in the same DRAM subarray

```
alloc_align(int size, int id)
```
size: # of bytes allocated
id: allocations with the same id go to the same subarray

RowClone Memory Allocation (II)

https://arxiv.org/abs/2111.00082

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

Key Mechanism: Distribute virtual pages to different banks while mapping them to DRAM rows in the same DRAM subarray

```
alloc_align(int size, int id)
size: # of bytes allocated
id: allocations with the same id go to the same subarray
```

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

Microbenchmarks

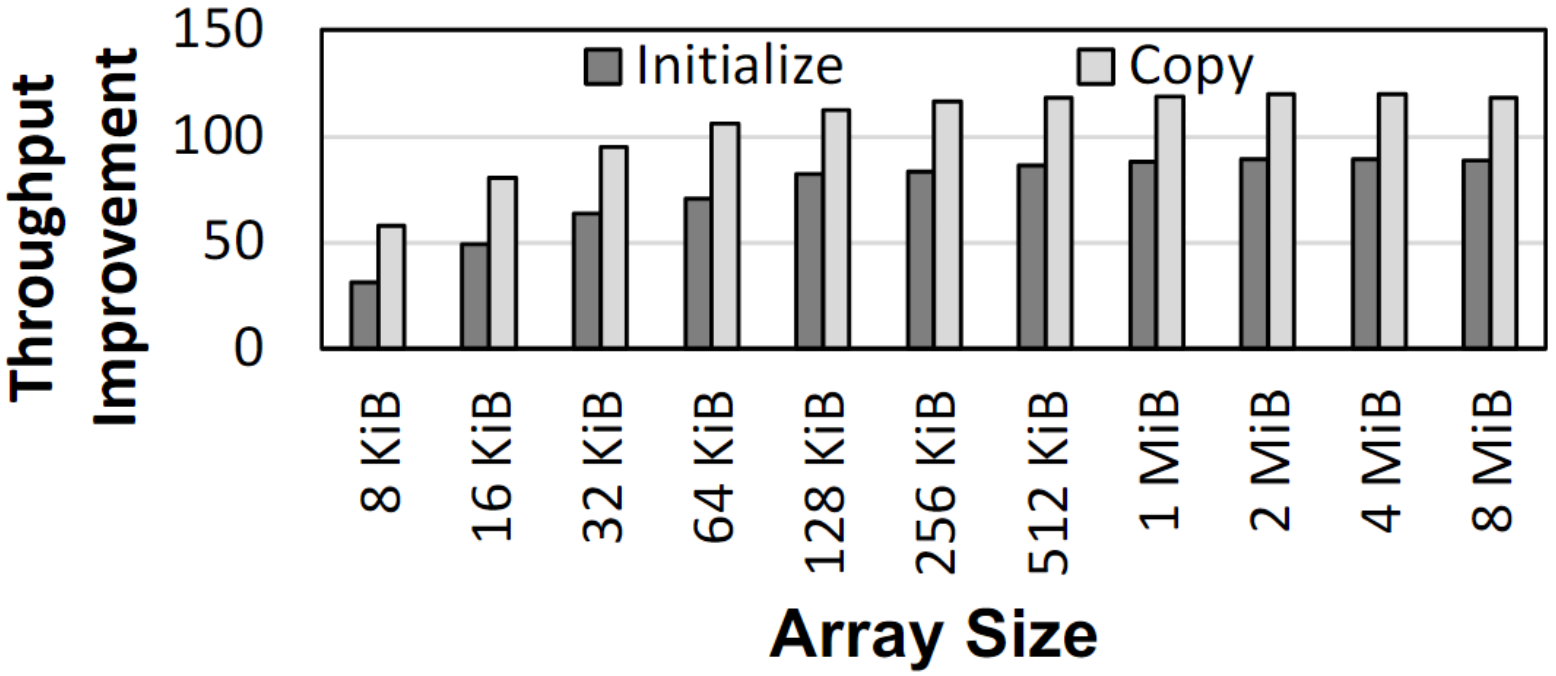
- CPU-Copy (using LOAD/STORE instructions)
- RowClone-Copy (using in-DRAM copy operations)

Copy/Initialization Heavy Workloads

- compile (initialization)
- forkbench (copy)

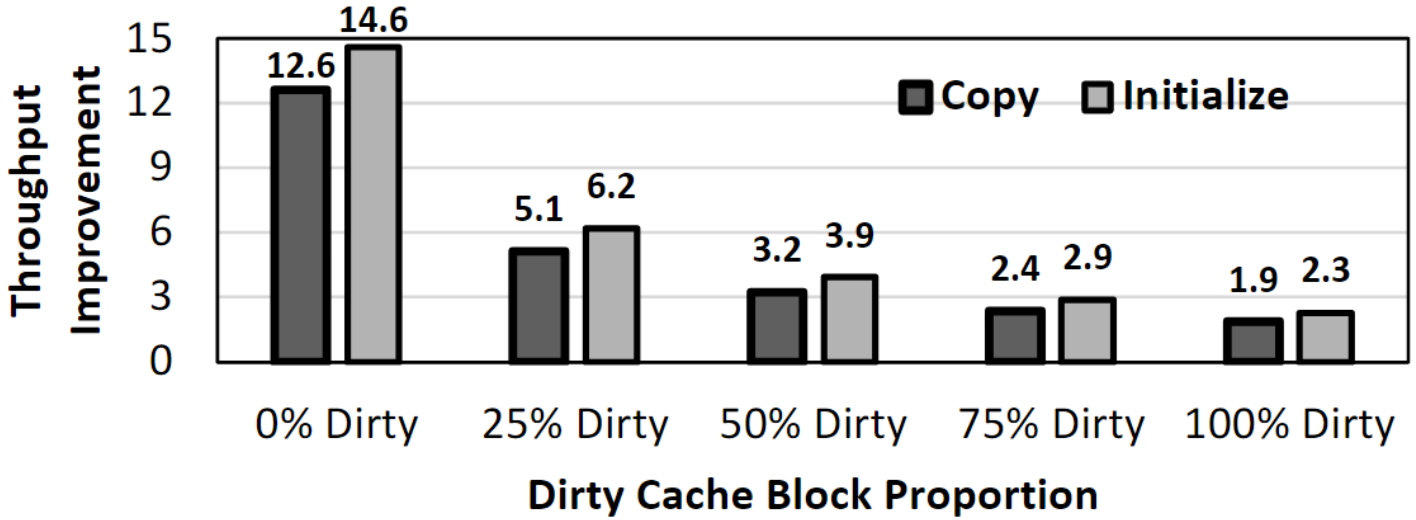
SPEC2006 libquantum: replace “calloc()” with in-DRAM initialization

Microbenchmark Copy/Initialization Throughput Improvement



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

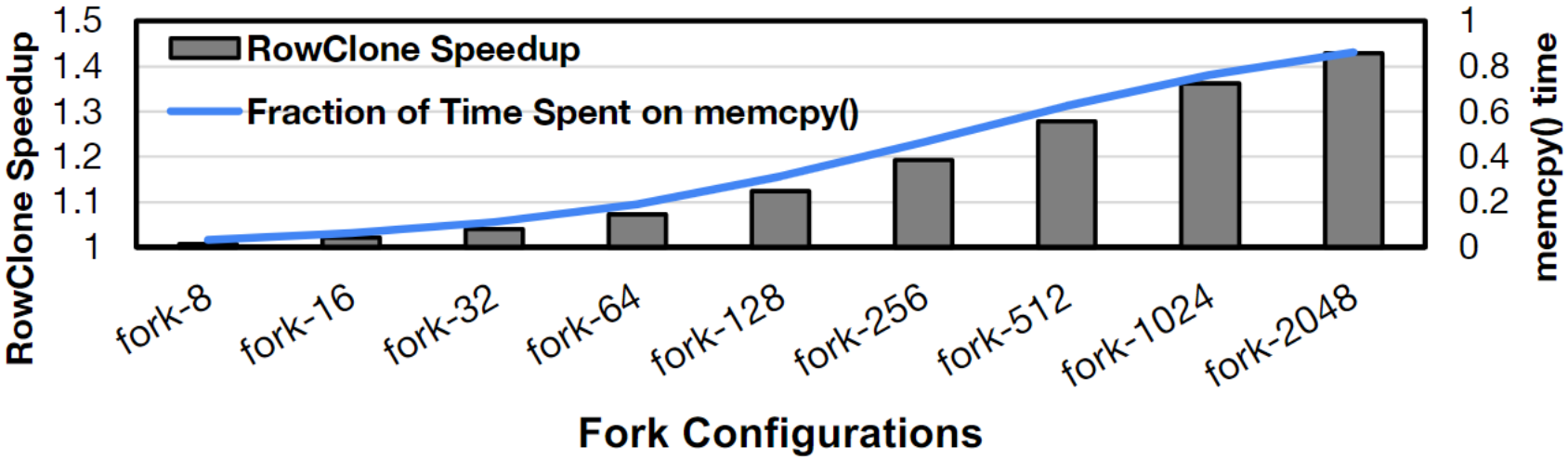
CLFLUSH Overhead



CLFLUSH dramatically reduces the potential throughput improvement

Other Workloads

forkbench (copy-heavy workload)



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
  - DRAM Organization and Operation
  - Commodity DRAM Based PiM Techniques
- PiDRAM
  - Overview
  - Hardware & Software Components
  - FPGA Prototype
- Case Studies
  - Case Study #1 – RowClone
- **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: FPGA-based framework that enables:

- System integration studies
- End-to-end evaluations

of commodity DRAM based 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

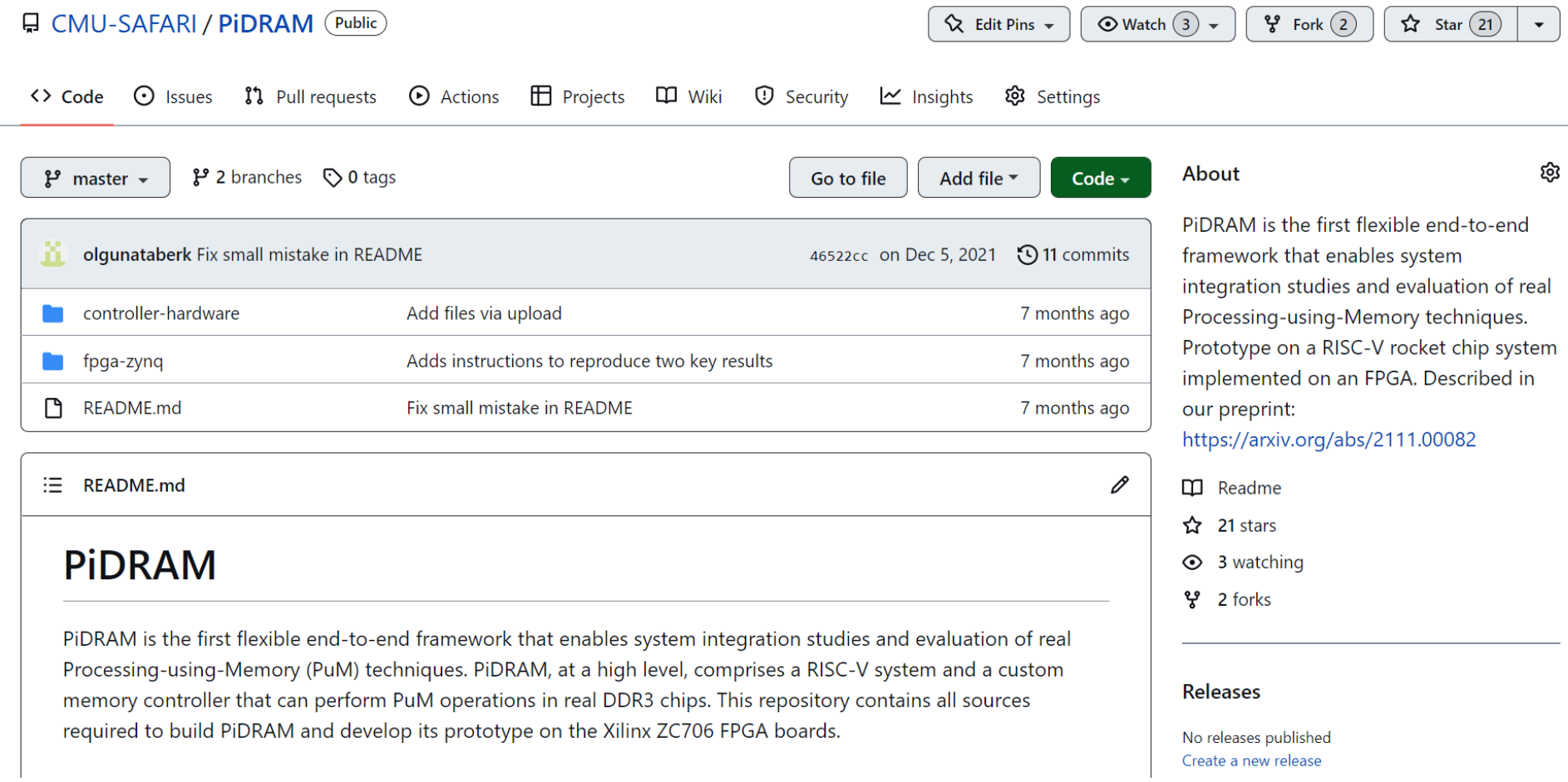
- 118.5x speedup for copy operations compared to CPU-copy with system support
- 198 lines of Verilog and 565 lines of C++ code over PiDRAM's flexible codebase

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

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

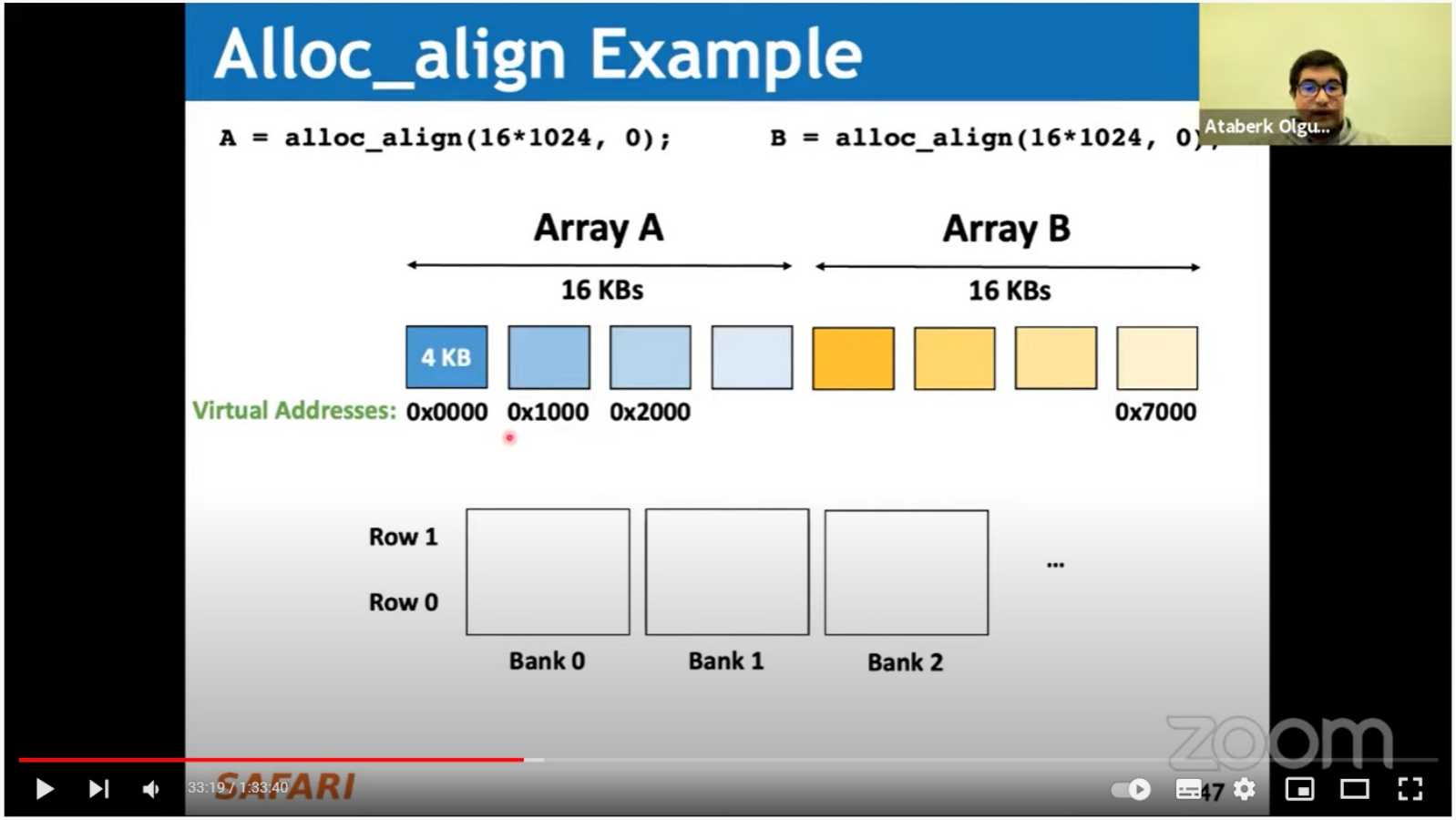
Processing-using-memory (PuM) techniques leverage the analog operation of memory cells to perform computation. Several recent works have demonstrated PuM techniques in off-the-shelf DRAM devices. Since DRAM is the dominant memory technology as main memory in current computing systems, these PuM techniques 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

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

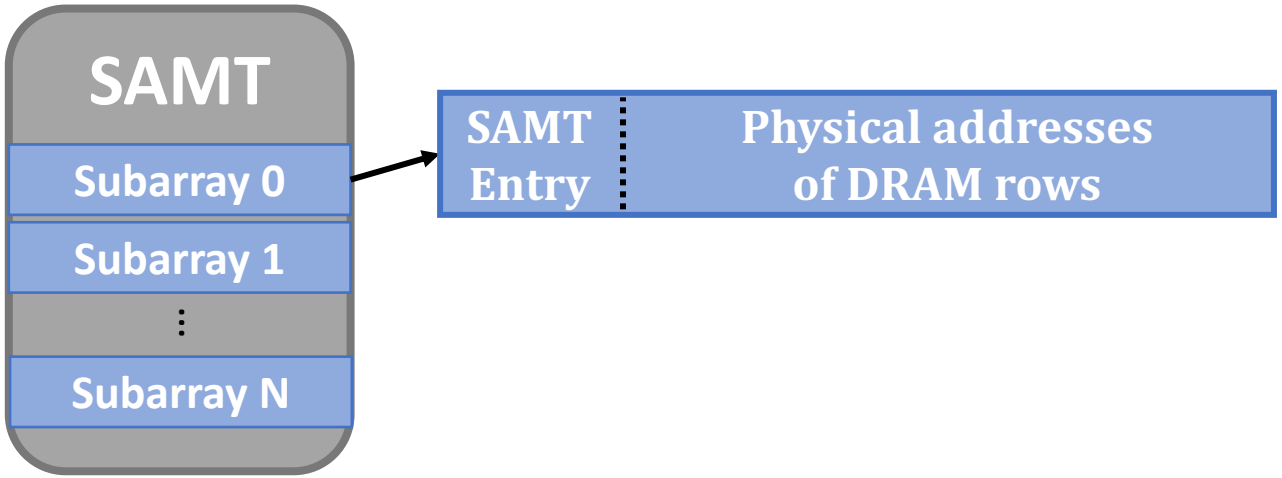
kasırga

TOBB ETÜ
University of Economics & Technology

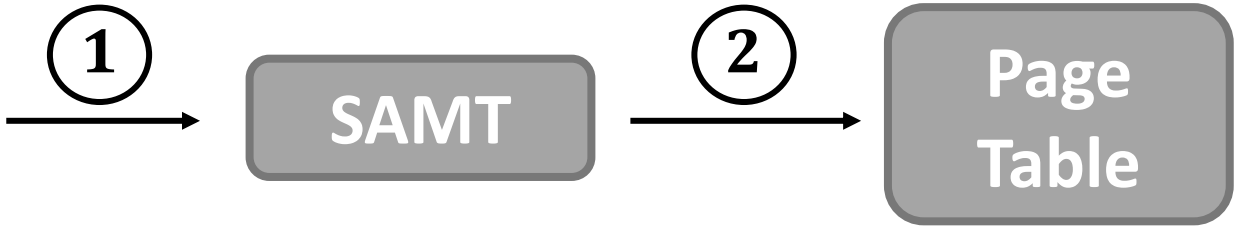
BACKUP SLIDES

alloc_align() function

To enable alloc_align(), we maintain the SubArray Mapping Table (SAMT)



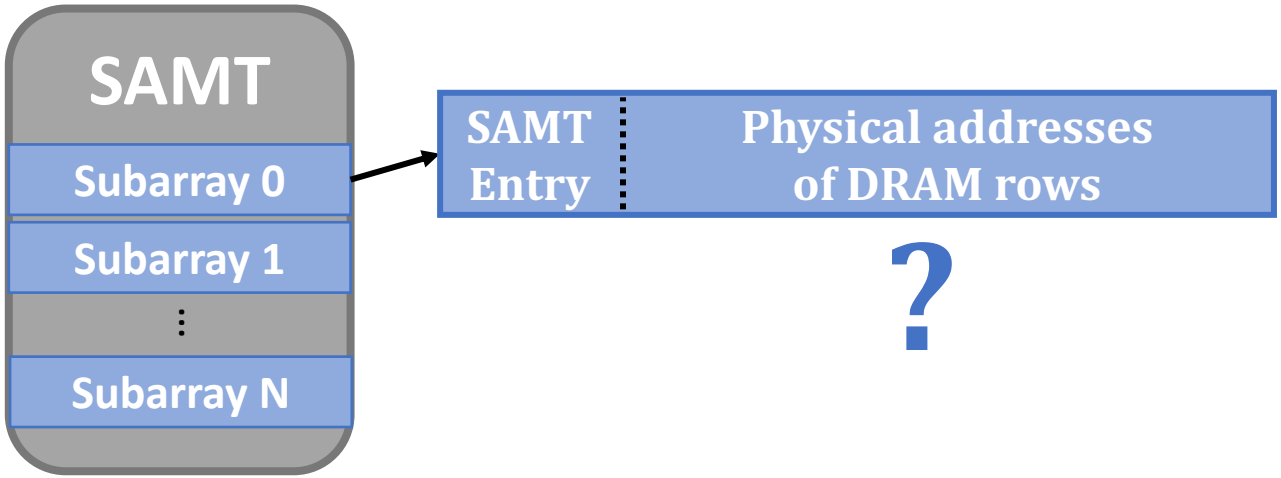

```
alloc_align(
4 KiB,
"Subarray 0")
```



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

https://arxiv.org/abs/2111.00082



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

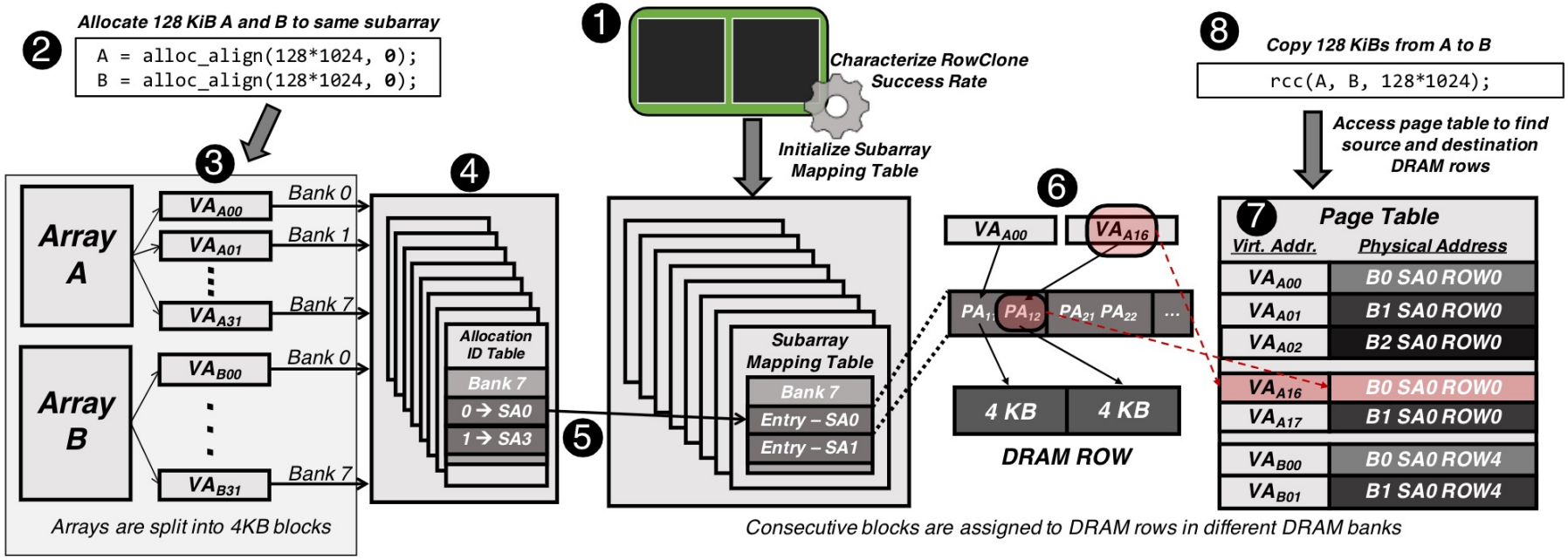
If the in-DRAM copy operation succeeds source and destination rows are in the same subarray

PiDRAM vs Others

Table 3: Comparing features of PiDRAM with related state-of-the-art hardware-software platforms

Platforms	Interface with DRAM	Flexible MC for PuM	Open-source	System software support
Silent-PIM [57]	✗	✗	✗	✓
SoftMC [48]	✓(DDR3)	✗	✓	✗
ComputeDRAM [36]	✓(DDR3)	✗	✗	✗
MEG [103]	✓(HBM)	✗	✓	✓
ZYNQ [4]	✓(DDR3/4)	✗	✗	✓
Simulators [20, 65, 84, 87]	✗	✓	✓	✓
PiDRAM (this work)	✓(DDR3)	✓	✓	✓

Alloc_align and RCC



List of PuM Techniques That Can Be Studied Using PiDRAM

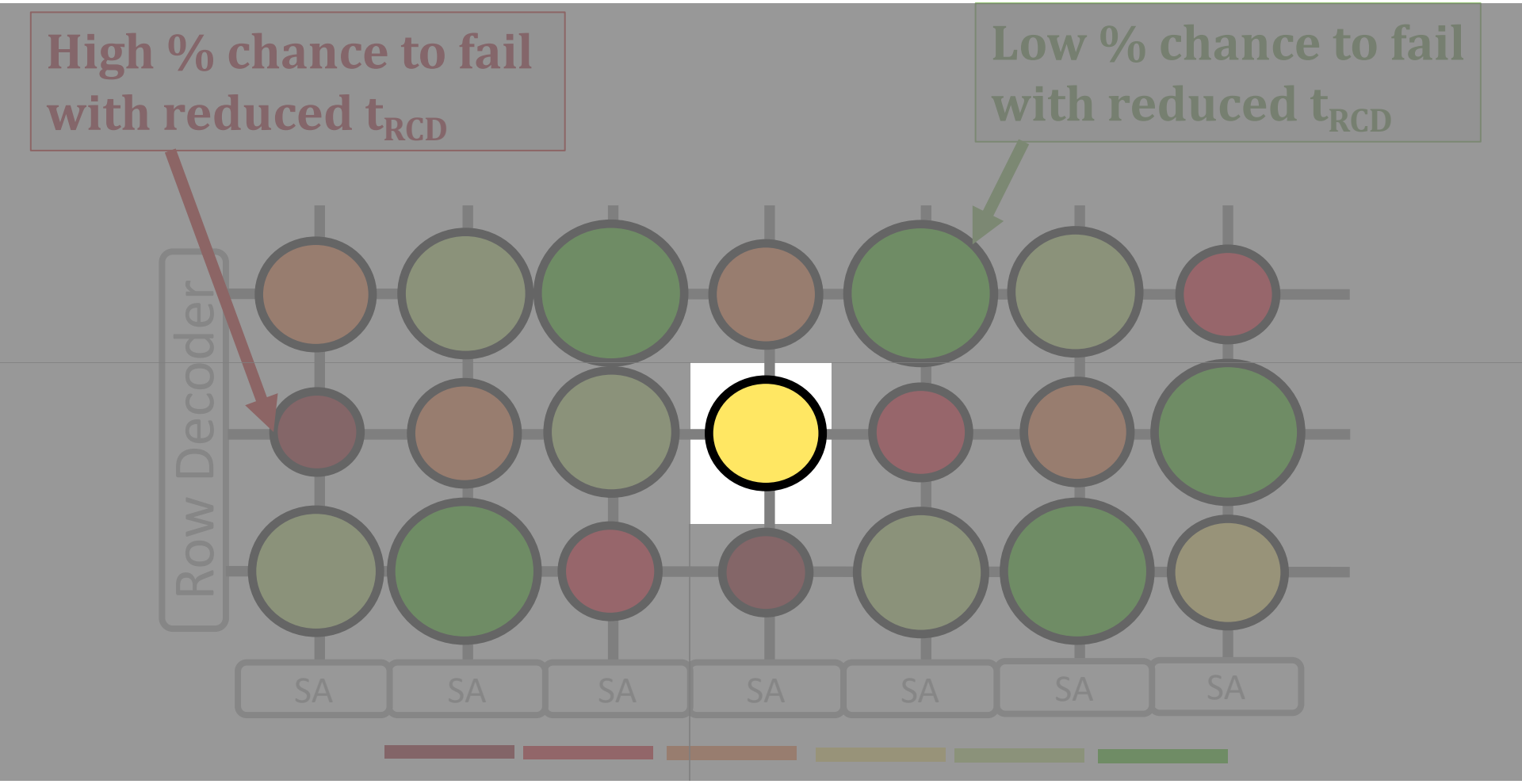
Table 1: PuM techniques that can be studied using PiDRAM. PuM techniques that we implement in this work are highlighted in bold

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.

Outline

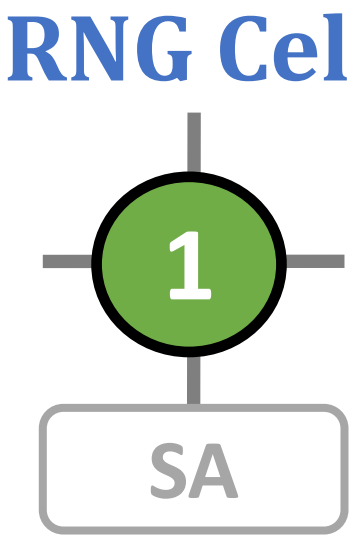
- Background
  - DRAM Organization and Operation
  - Commodity DRAM Based PiM Techniques
- PiDRAM
  - Overview
  - Hardware & Software Components
  - FPGA Prototype
- **Case Studies**
  - Case Study #1 – RowClone
  - **Case Study #2 – D-RaNGe**
- Conclusion

In-DRAM TRNG: Key Idea (D-RaNGe)



D-RaNGe Implementation

Identify four DRAM cells that fail randomly when accessed with a reduced $\mathbf{t_{RCD}}$ (**RNG Cell**) in a cache block



D-RaNGe Implementation

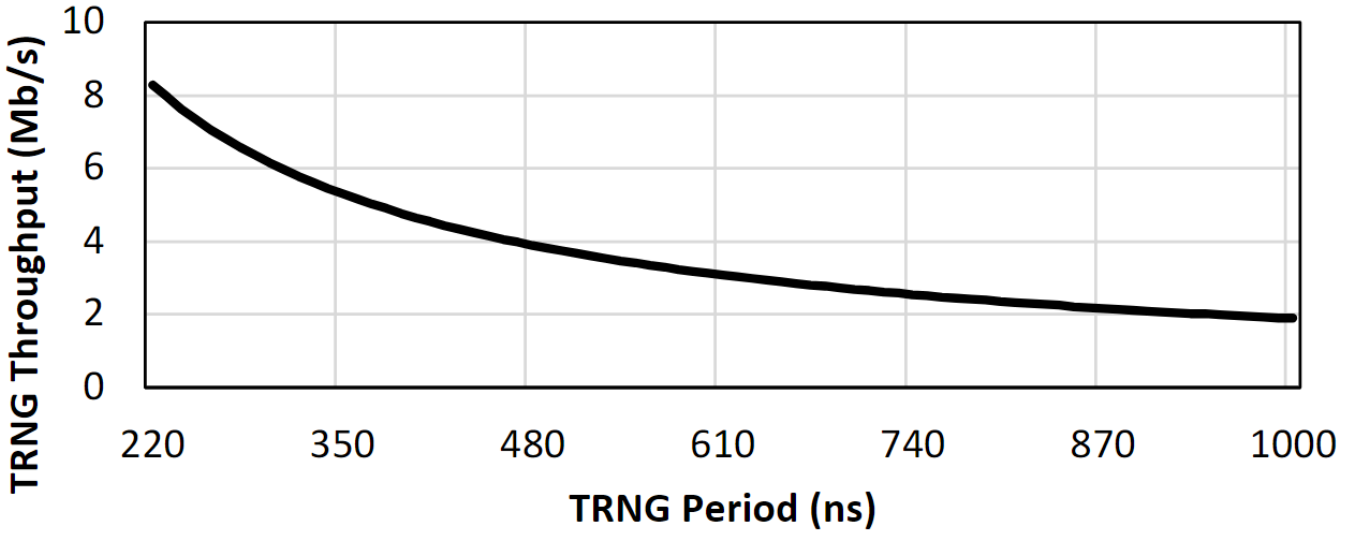
Periodically generate true random numbers
by accessing the identified cache block with reduced tRCD

- 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 D-RaNGe generates true random numbers at 8.30 Mb/s throughput