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 kasırga

ETH zürich 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

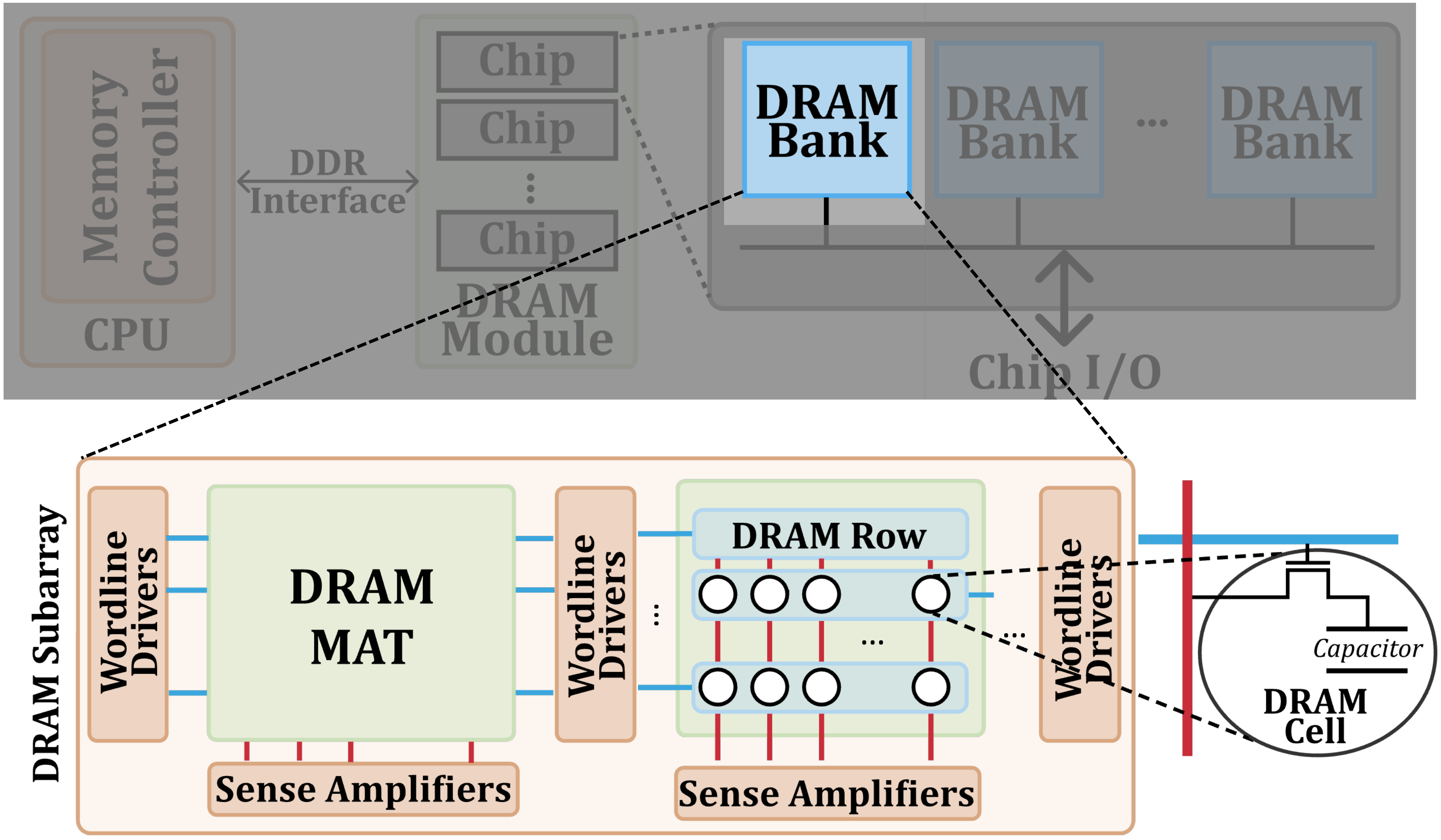
Case Study #2 – D-RaNGe: DRAM-based random number generation technique

- 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

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

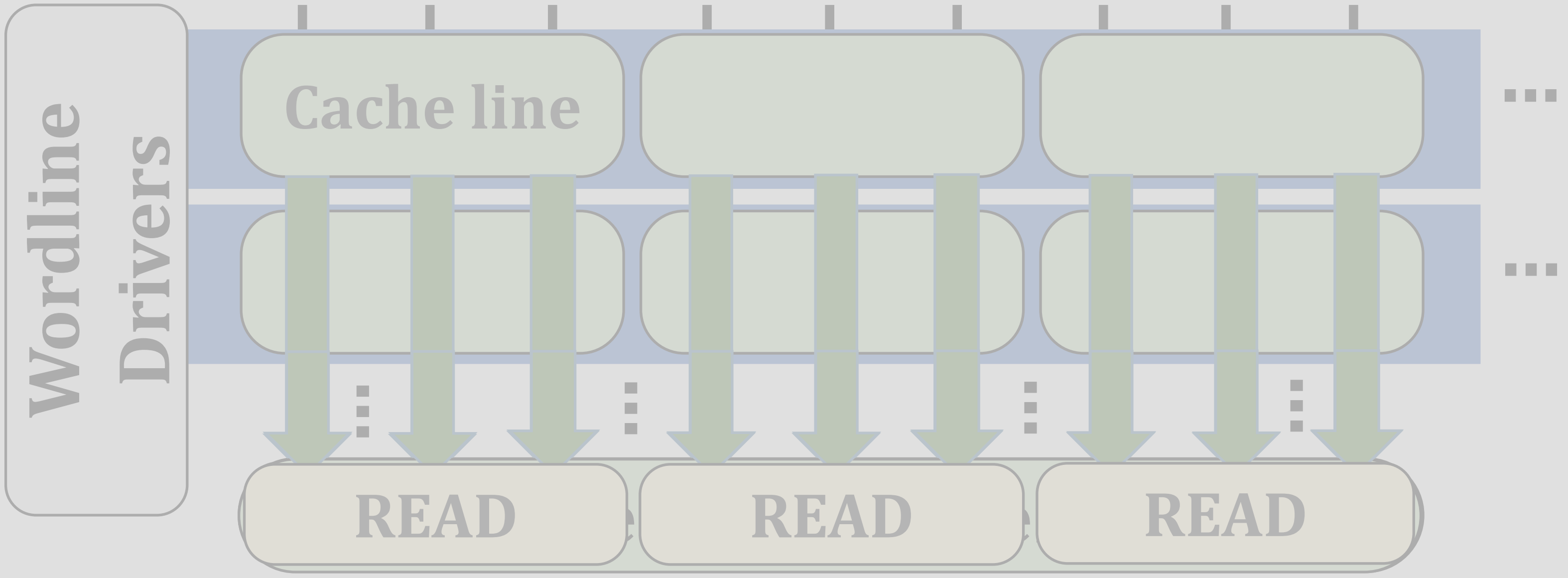
Outline

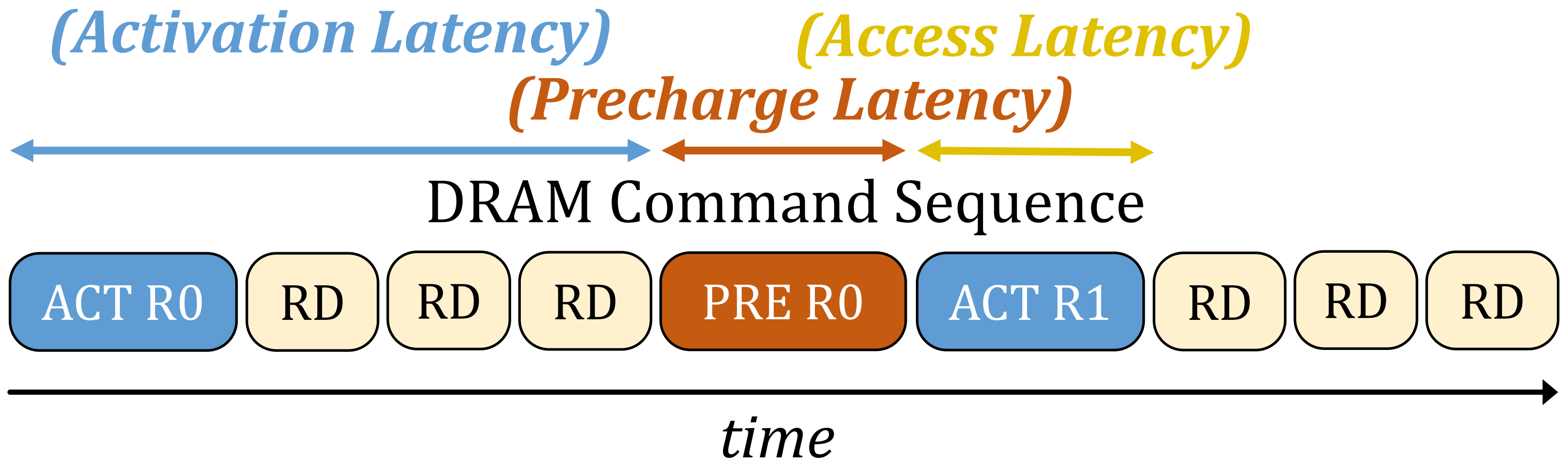
DRAM Organization



[Olgun+ ISCA'21]

DRAM Operation





[Kim+ HPCA'19]

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

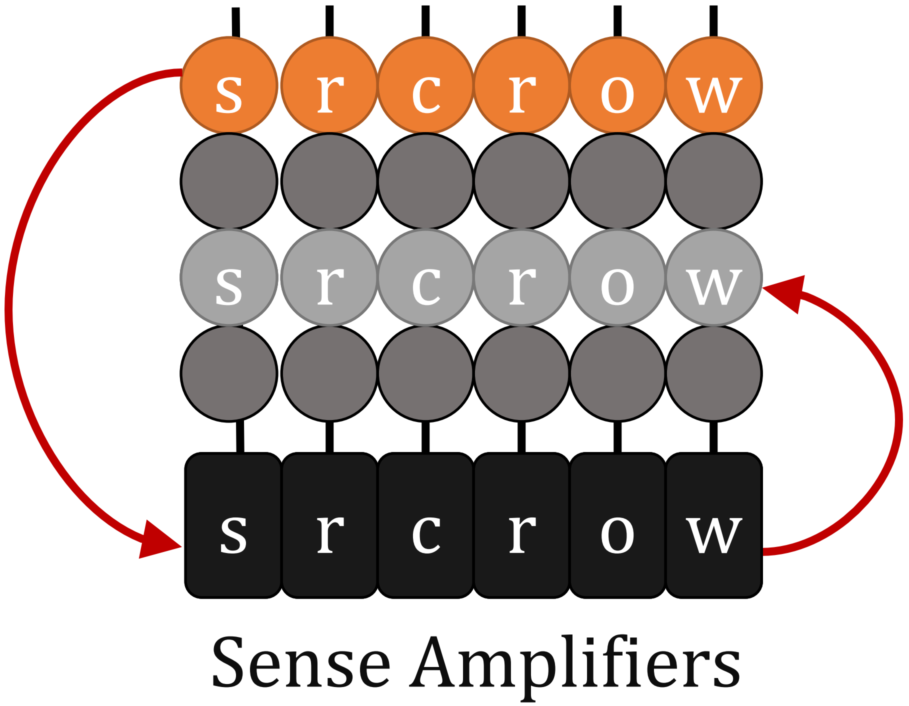
(e.g., [Kim+, HPCA'18])

3) Physical uncloneable functions

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

4) Majority operation

Row-Copy: Key Idea (RowClone)



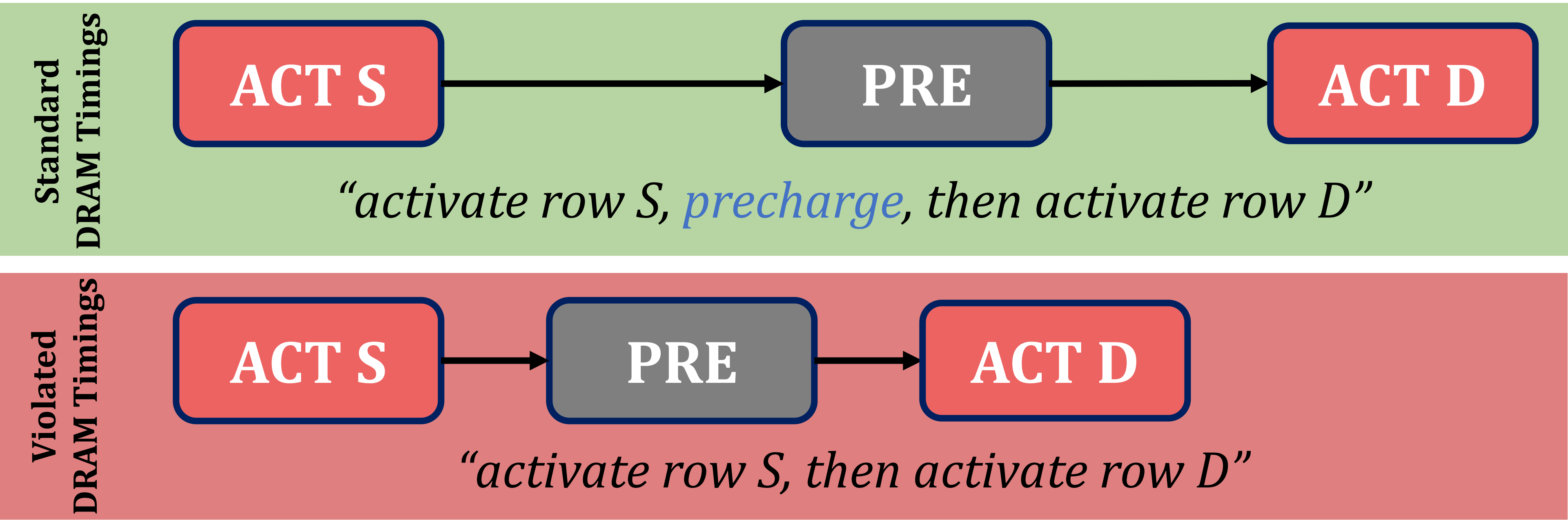
✓ 1. Source row to sense amplifiers

? 2. Sense amplifiers to destination row

RowClone in Real DRAM Chips

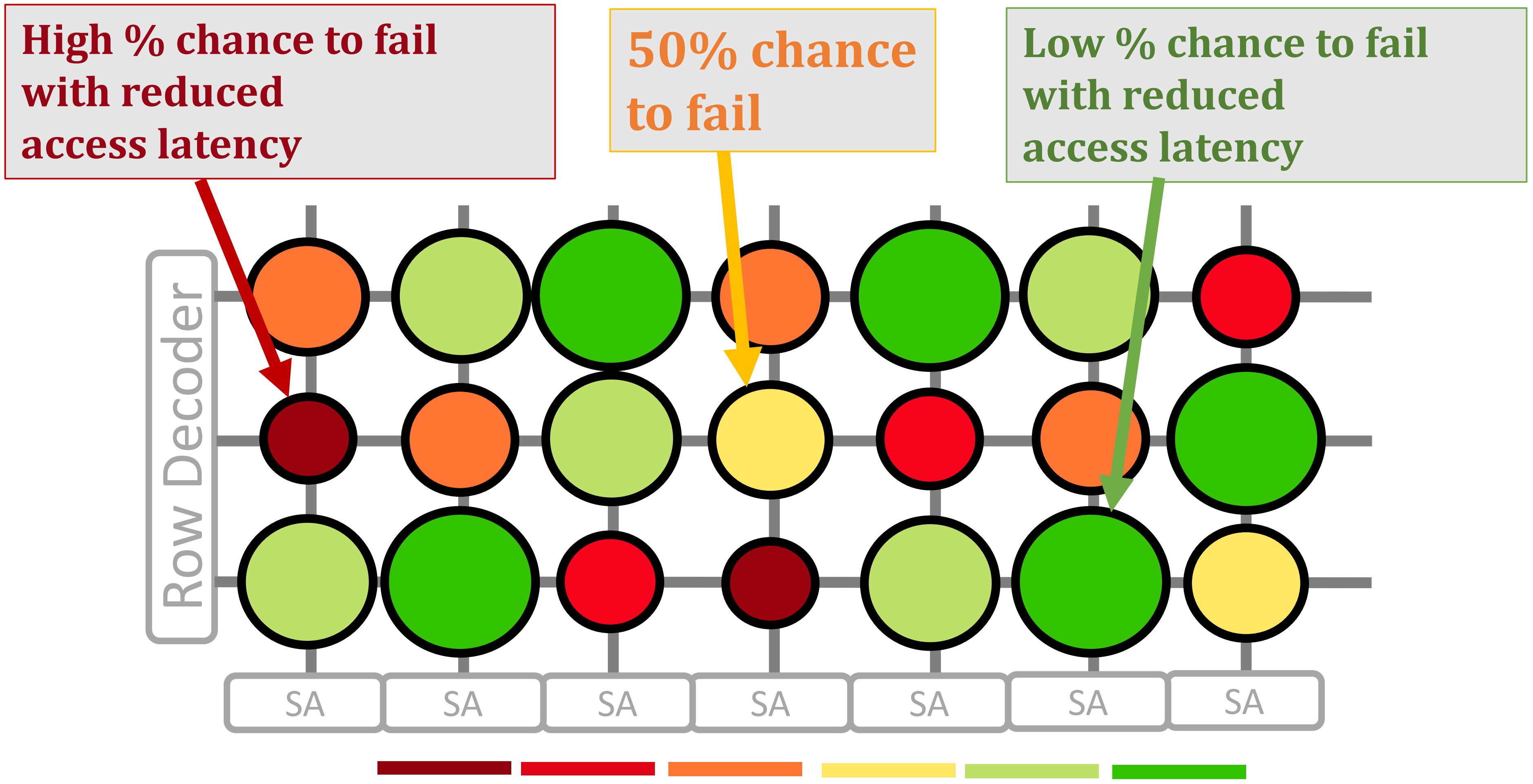
Key Idea: Use carefully created DRAM command sequences

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



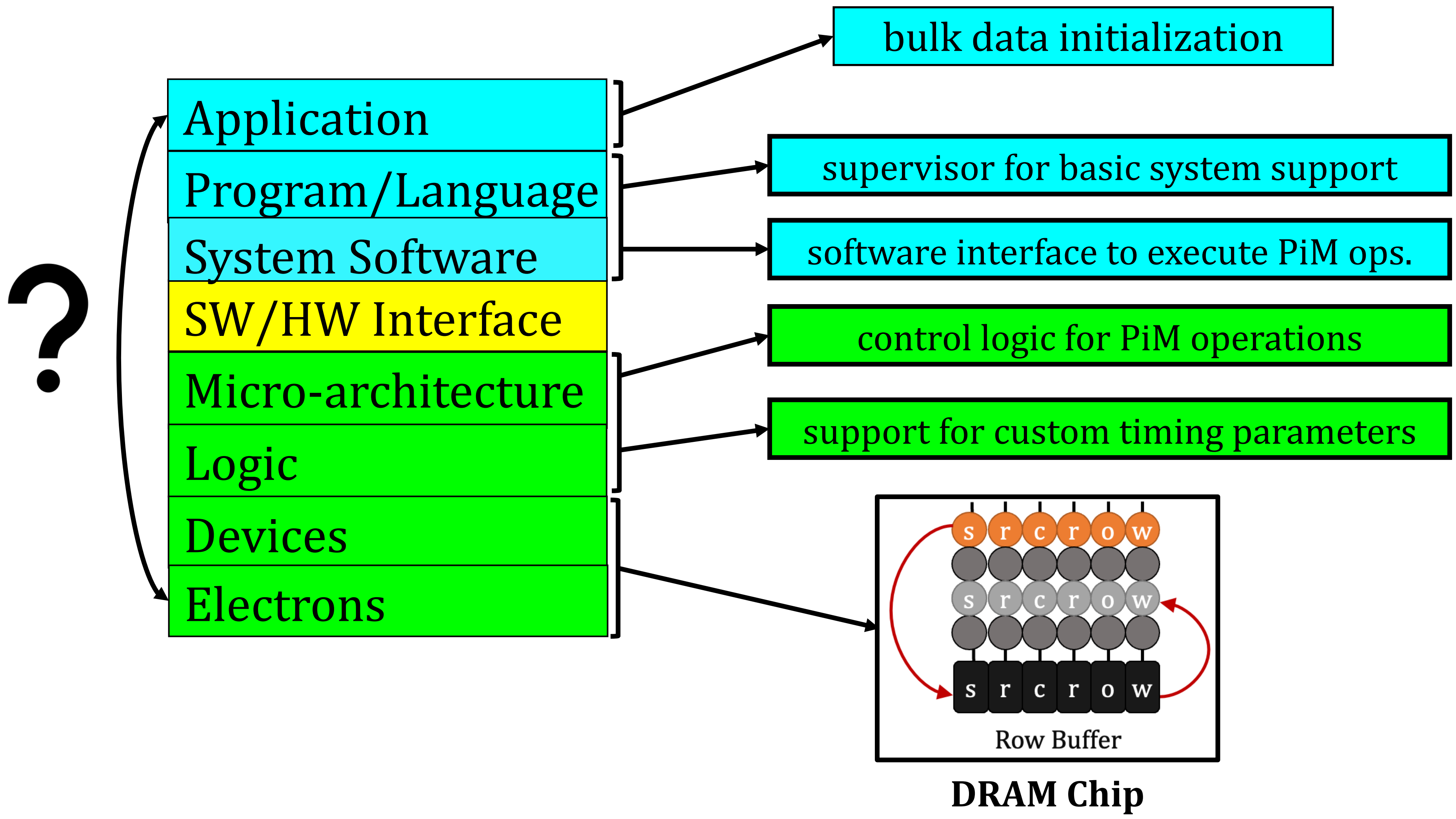
Standard DRAM Timings
ACT S
PRE
ACT D
"activate row S, precharge, then activate row D"
Violated DRAM Timings
ACT S
PRE
ACT D
"activate row S, then activate row D"

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

System Support for PiM



PiDRAM

bulk data initialization

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

supervisor for basic system support

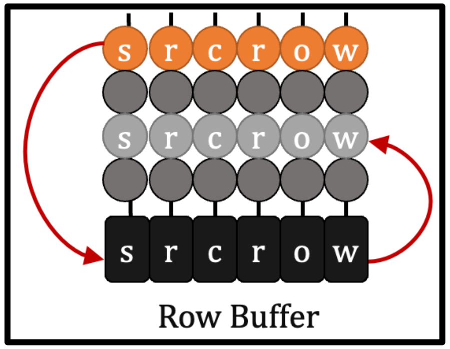
software interface to execute PiM ops.

control logic for PiM operations

in doing so, allow users to

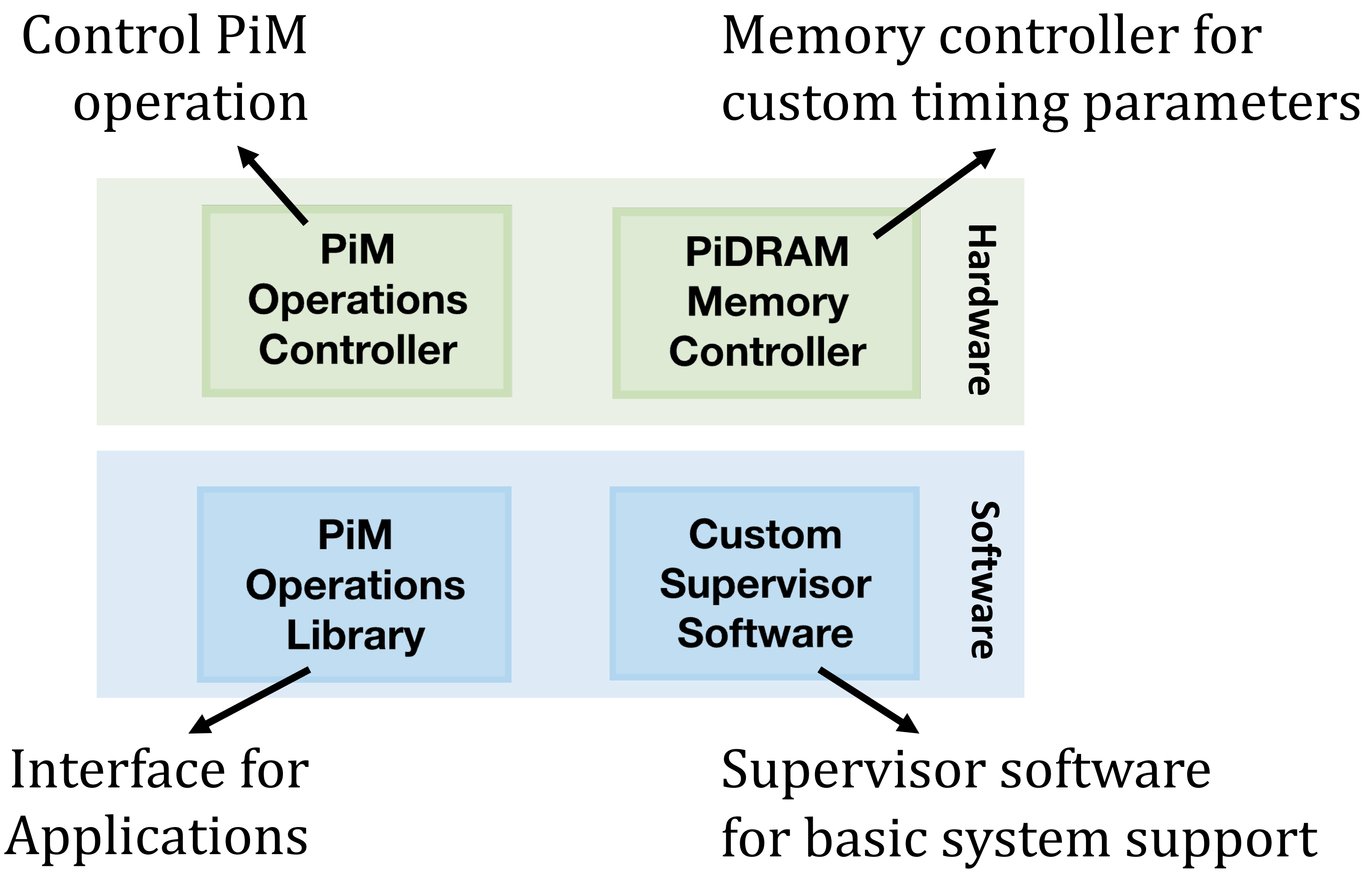
support for custom timing parameters

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



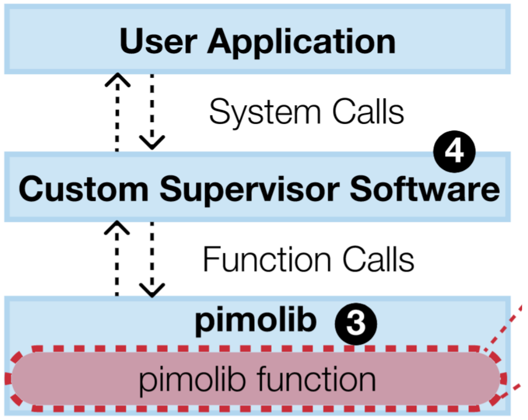
DRAM Chip

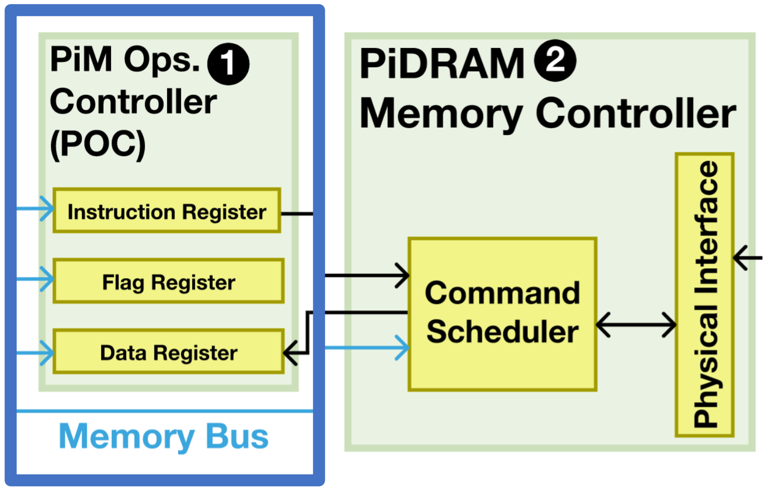
PiDRAM: Key Components


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



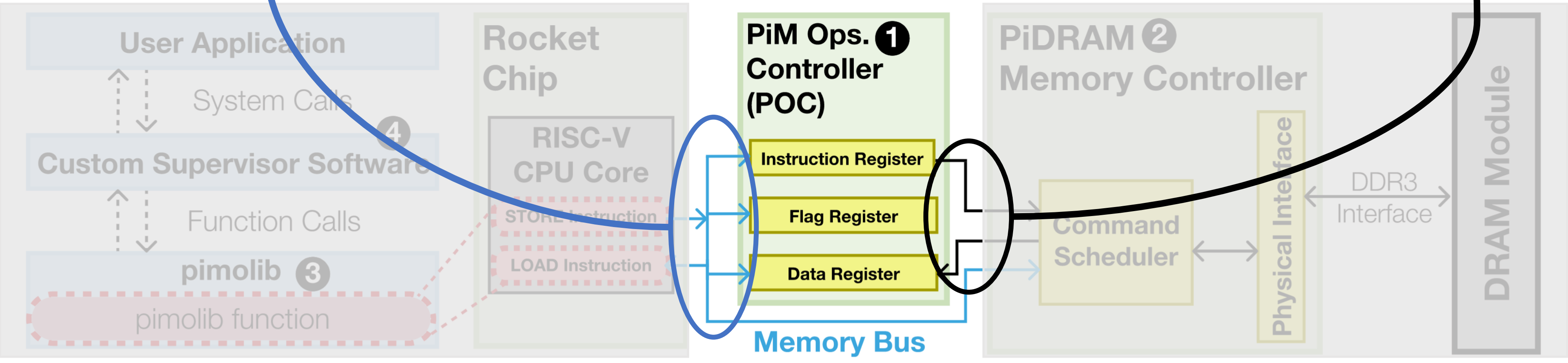


PiM Operations Controller (POC)

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



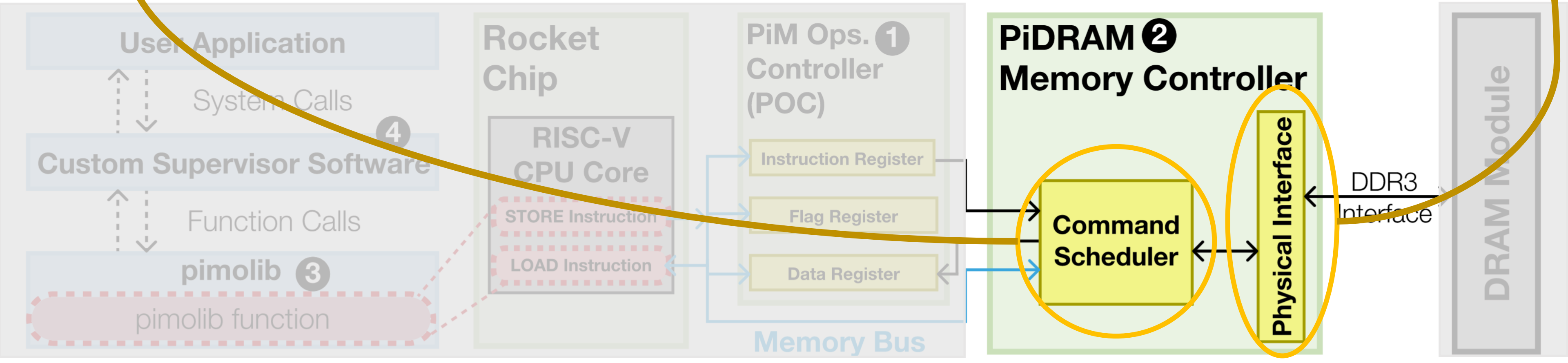
PiDRAM Memory Controller

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

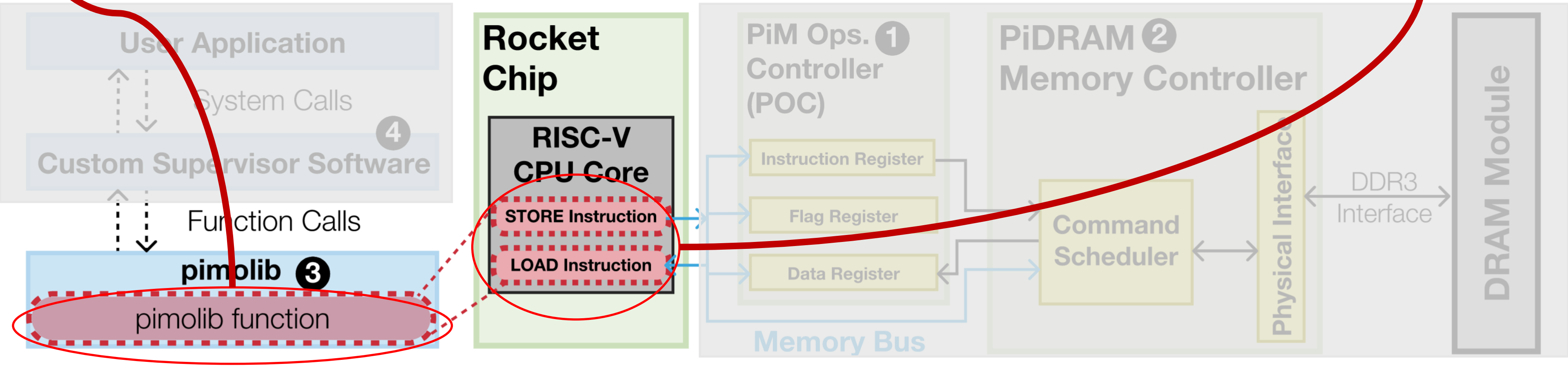
Controls the physical DDR3 interface
Receives commands from command scheduler & operates DDR3 pins



PiM Operations Library (pimolib)

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

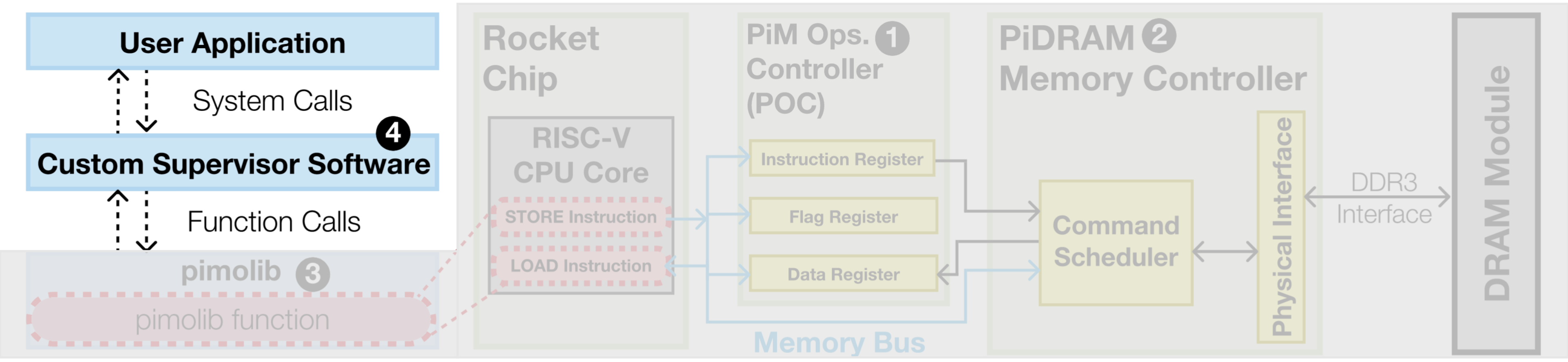
Executes LOAD & STORE requests to communicate with the POC



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)



SAFARI

PiM Operation Execution Flow

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

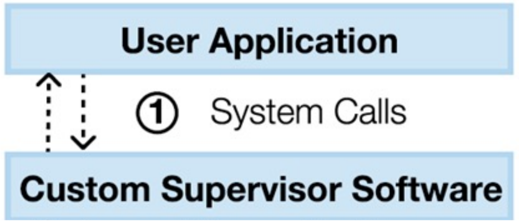
1 Application makes a system call: `Copy(A, B, N bytes)`

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

`Copy (S, D)`

`S`: source DRAM row

`D`: destination DRAM row



SAFARI

PiM Operation Execution Flow

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

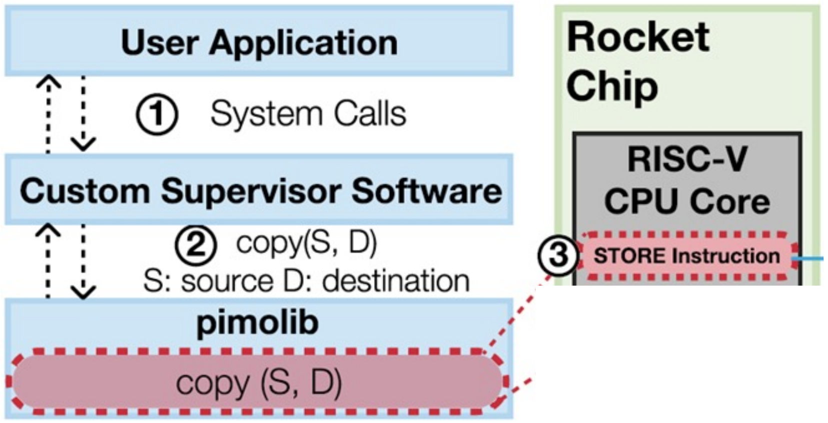
4 **The first store updates the *instruction* register with** `Copy(S, D)`

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

Start (S)

1

Start the execution of PiM operation

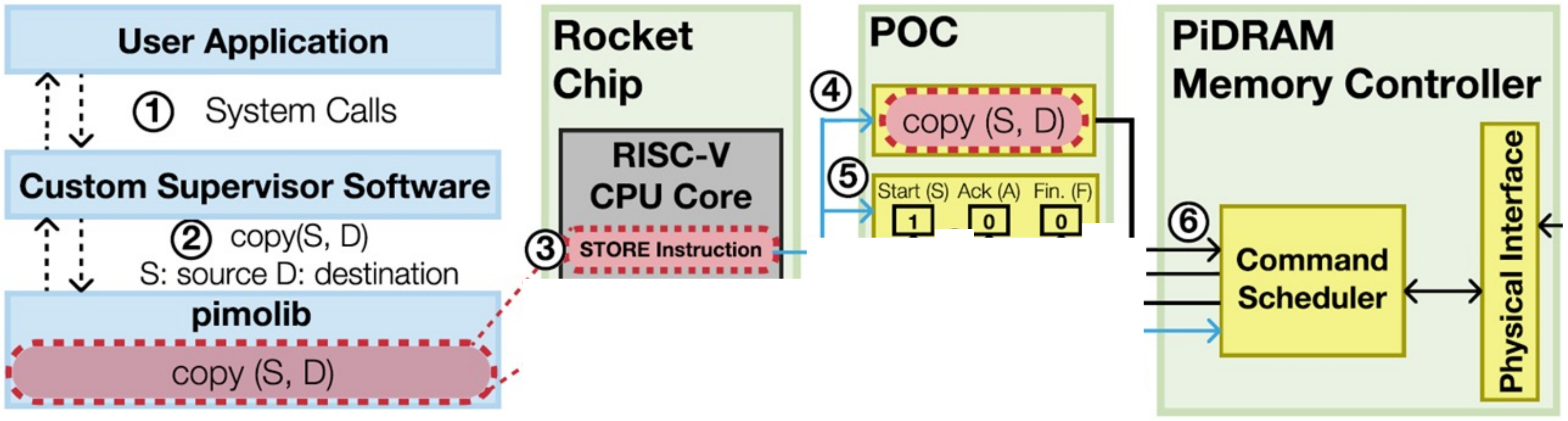


PiM Operation Execution Flow

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**

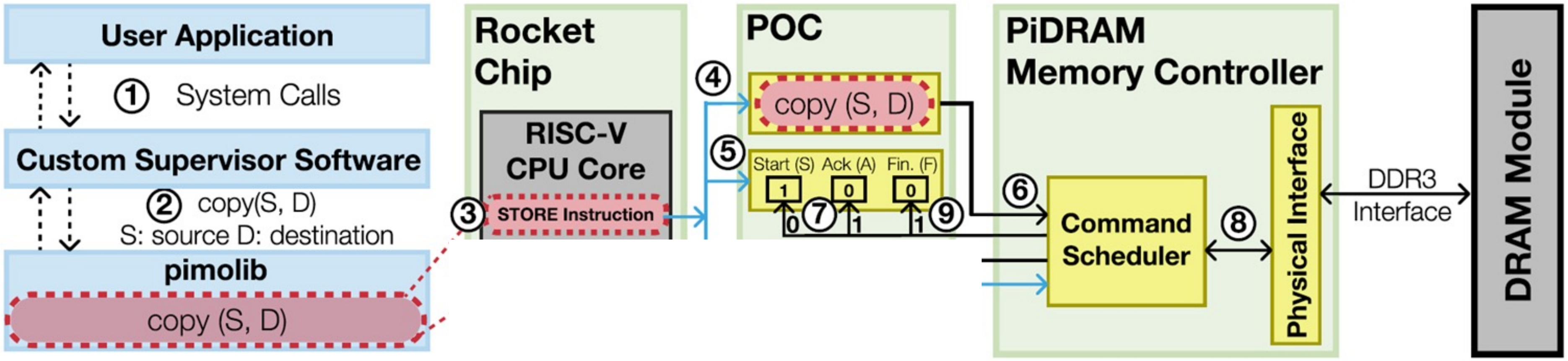


PiM Operation Execution Flow

9 The memory controller sets the "Fin." flag

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

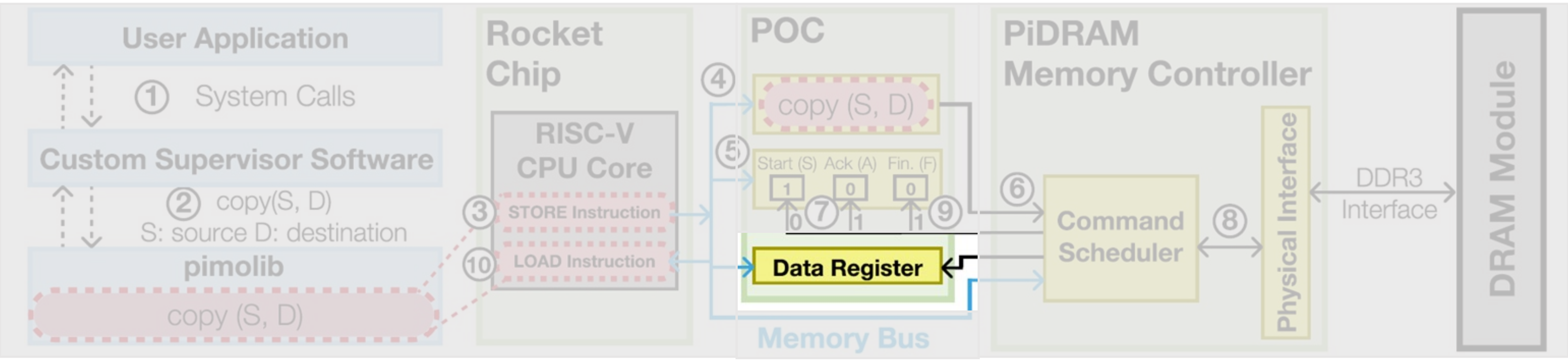
`Copy(S, D)` returns when the periodically checked flag is set



PiM Operation Execution Flow

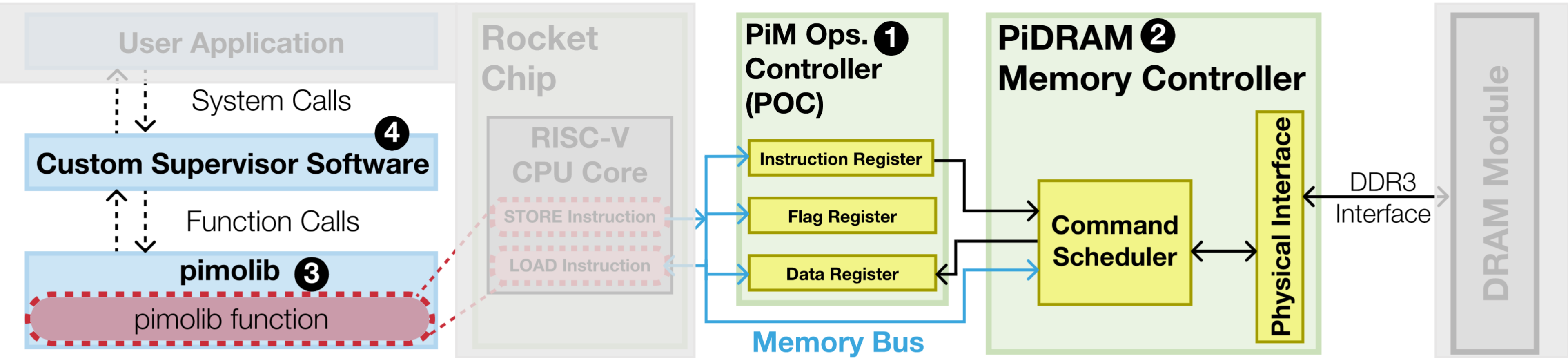
Data Register is not used in RowClone operations because the result is stored *in memory*

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



PiDRAM Components Summary

Four key components orchestrate PiM operation execution

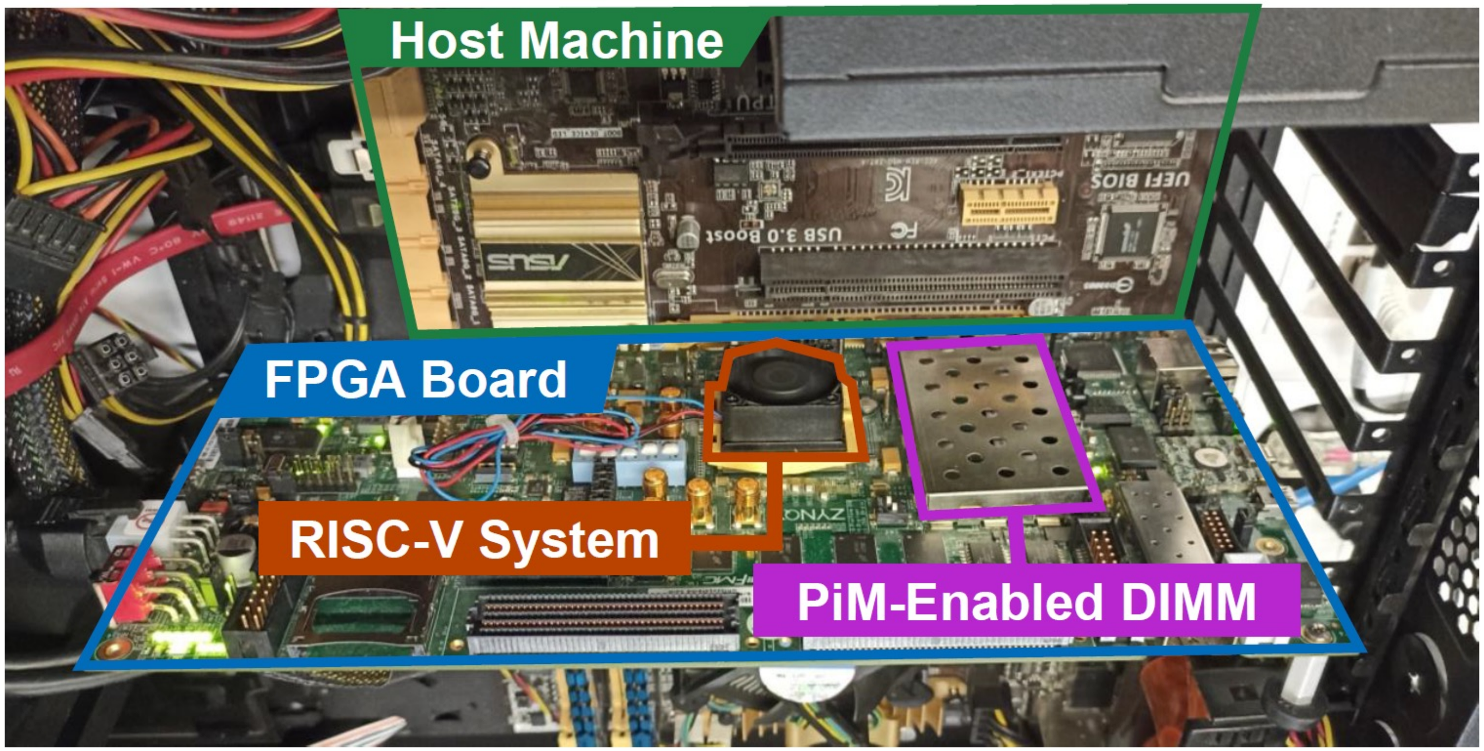


SAFARI

PiDRAM's FPGA Prototype

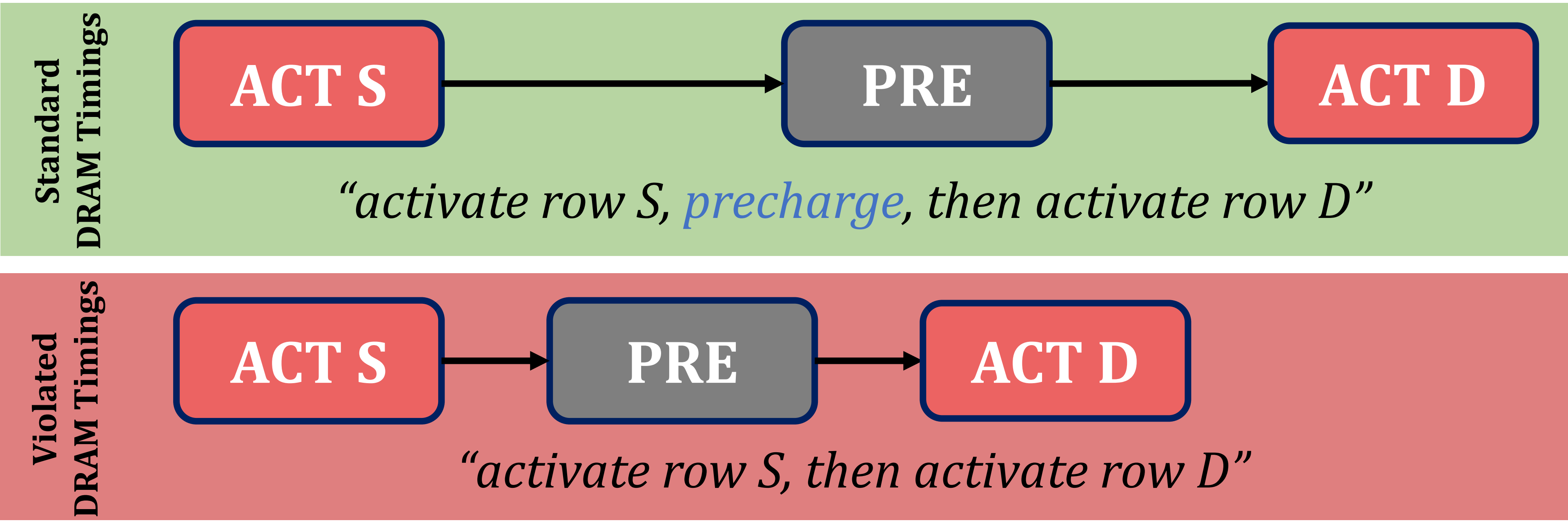
Full system prototype on Xilinx ZC706 FPGA board

- **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

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

RowClone System Integration

Identify two challenges in end-to-end RowClone

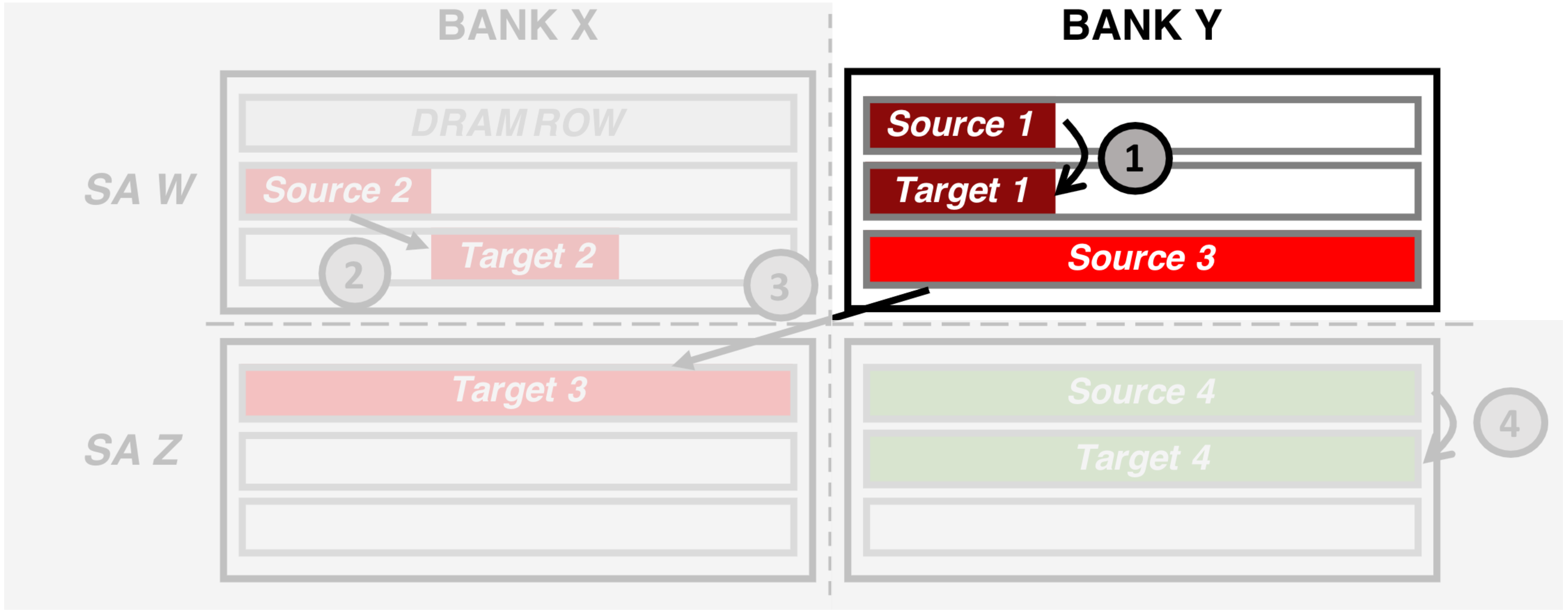
1. Memory allocation (intra-subarray operation)

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)

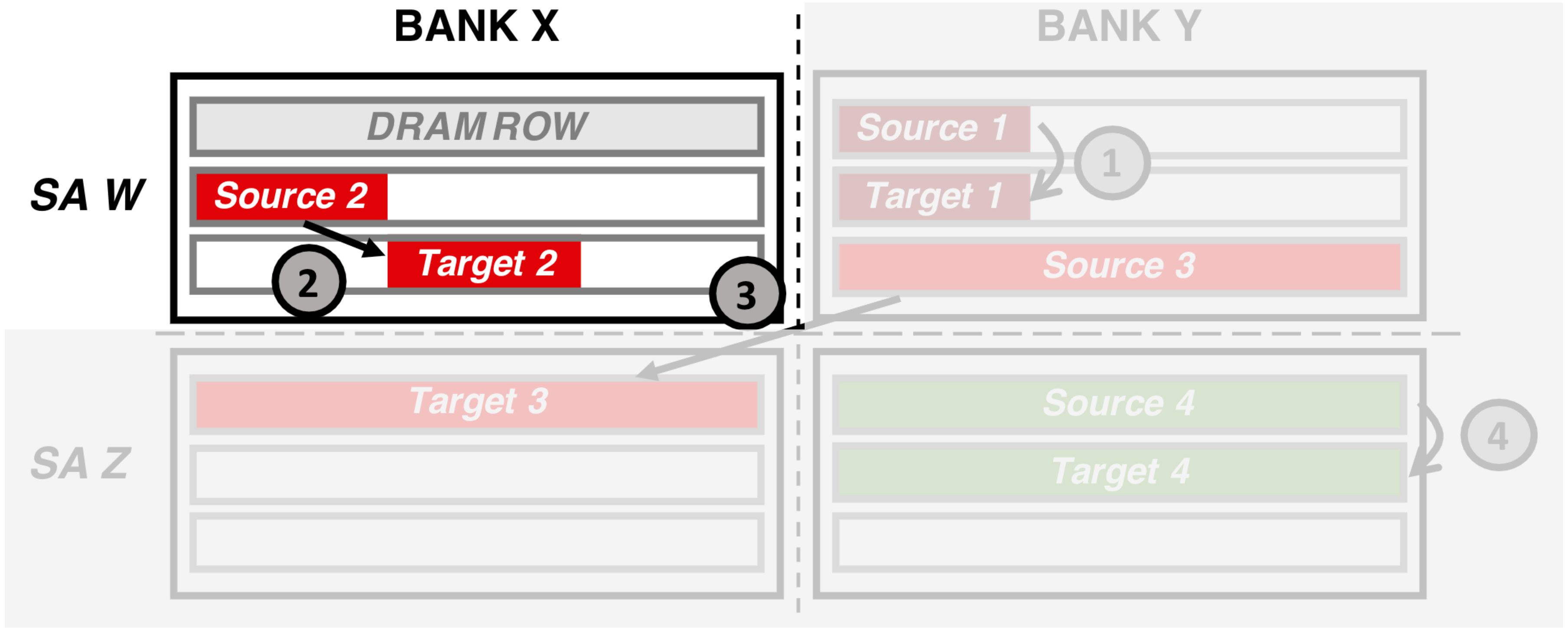
Memory allocation requirements



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

RowClone Memory Allocation (I)

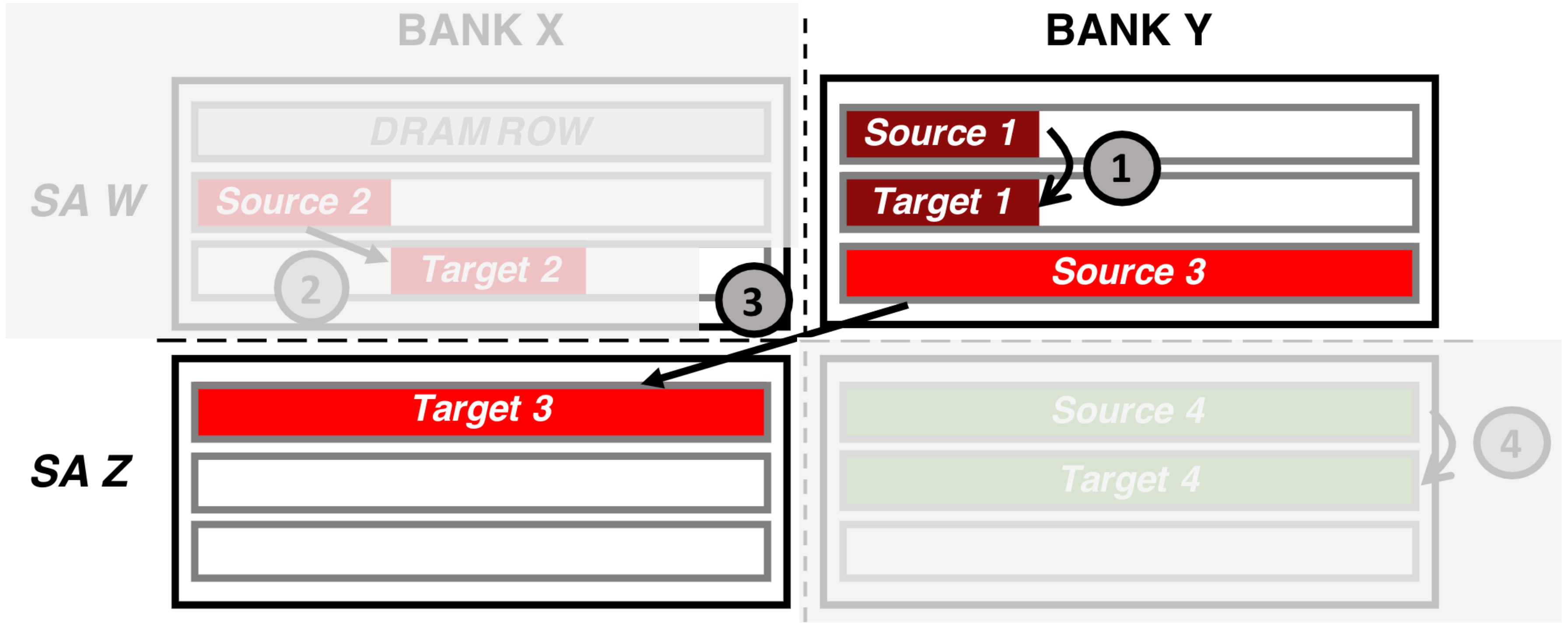
Memory allocation requirements



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

RowClone Memory Allocation (I)

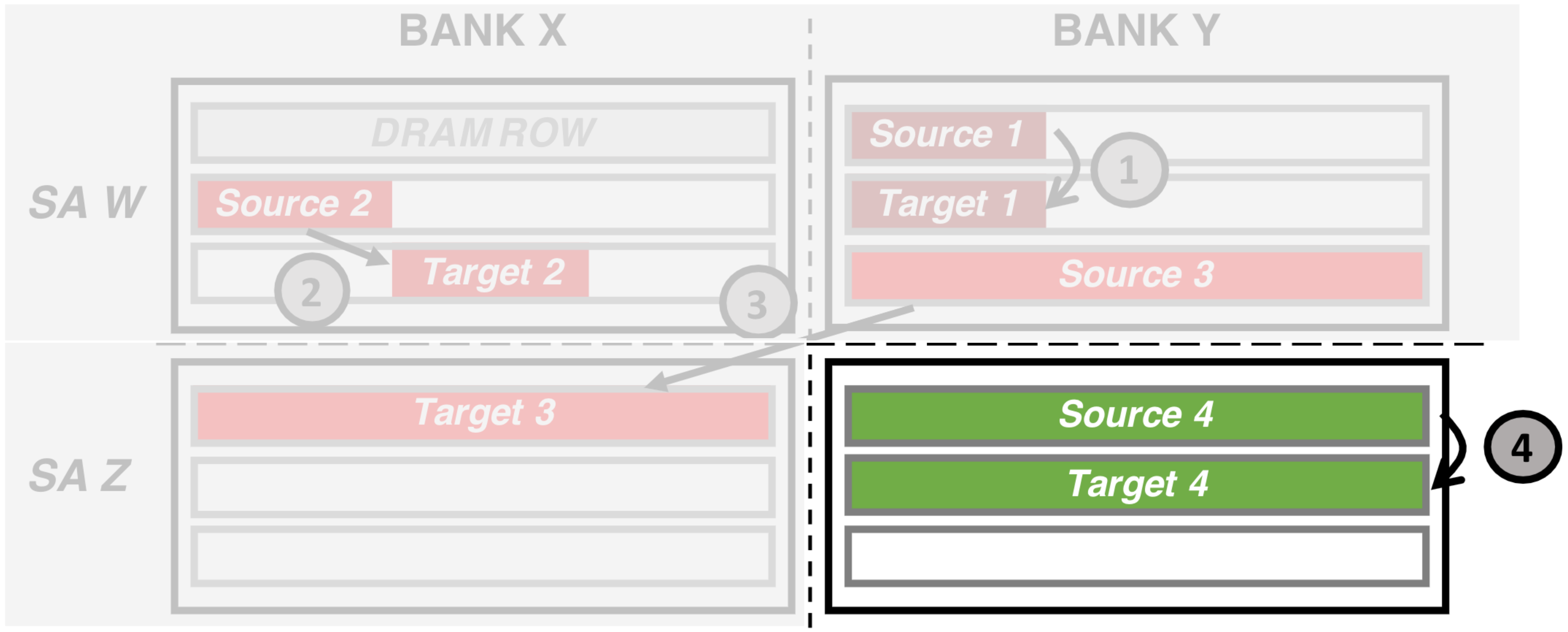
Memory allocation requirements



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

RowClone Memory Allocation (I)

Memory allocation requirements



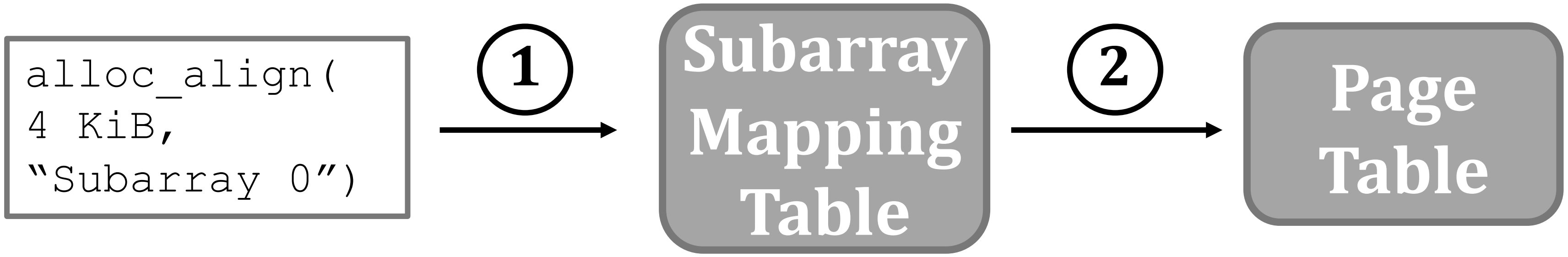
4 **Satisfies all three requirements**

RowClone Memory Allocation (II)

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



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)

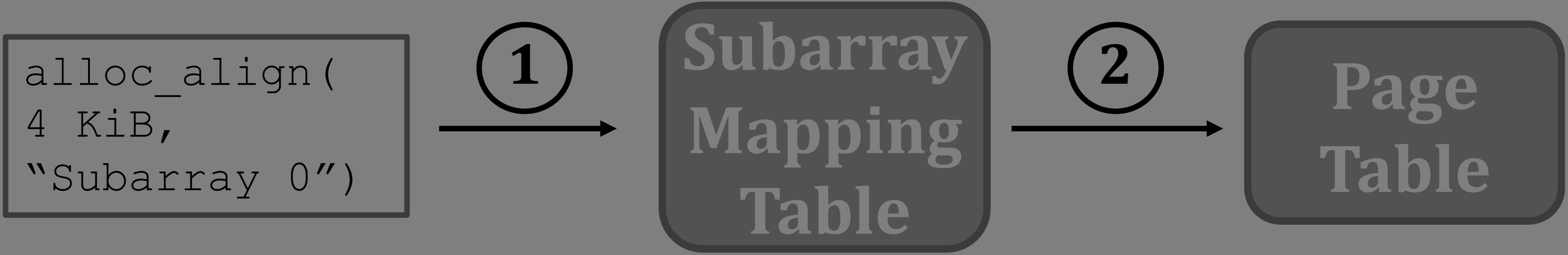
Implement a new memory allocation function

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

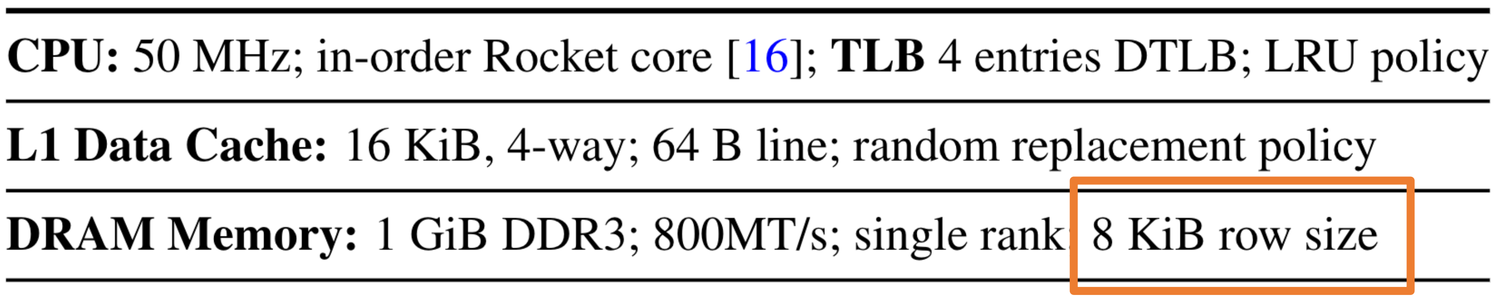
`id`: allocations with the same id go to the same subarray



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

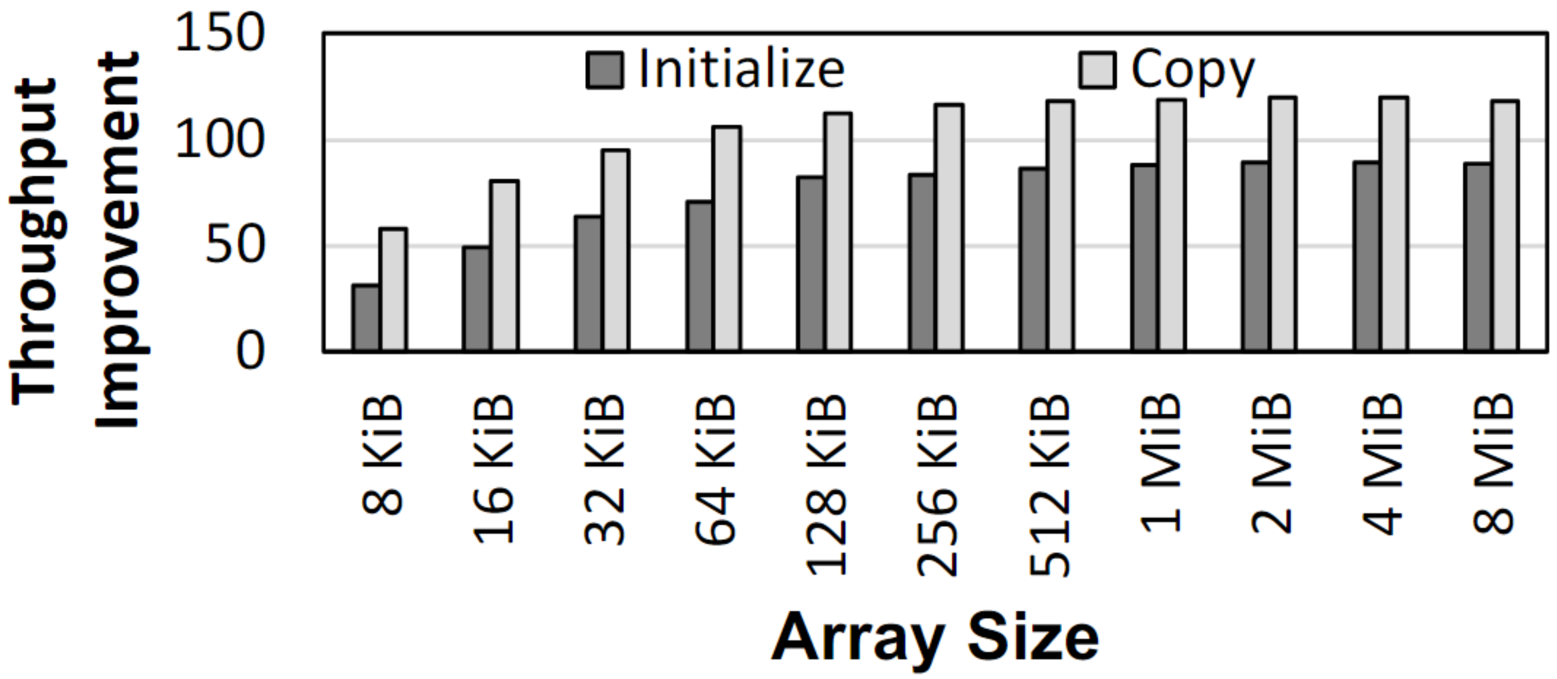
Copy/Initialization Heavy Workloads

forkbench (copy)

compile (initialization)

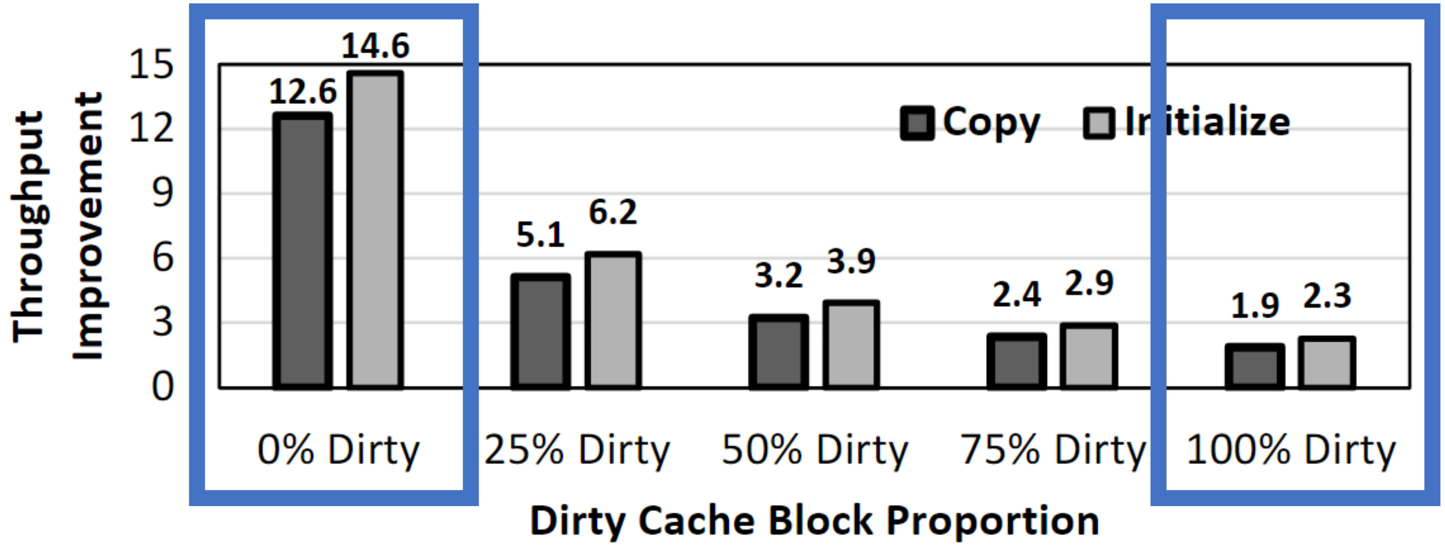
SPEC2006 libquantum: replace “calloc()” with in-DRAM initialization

Microbenchmark Copy/Initialization Throughput Improvement



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

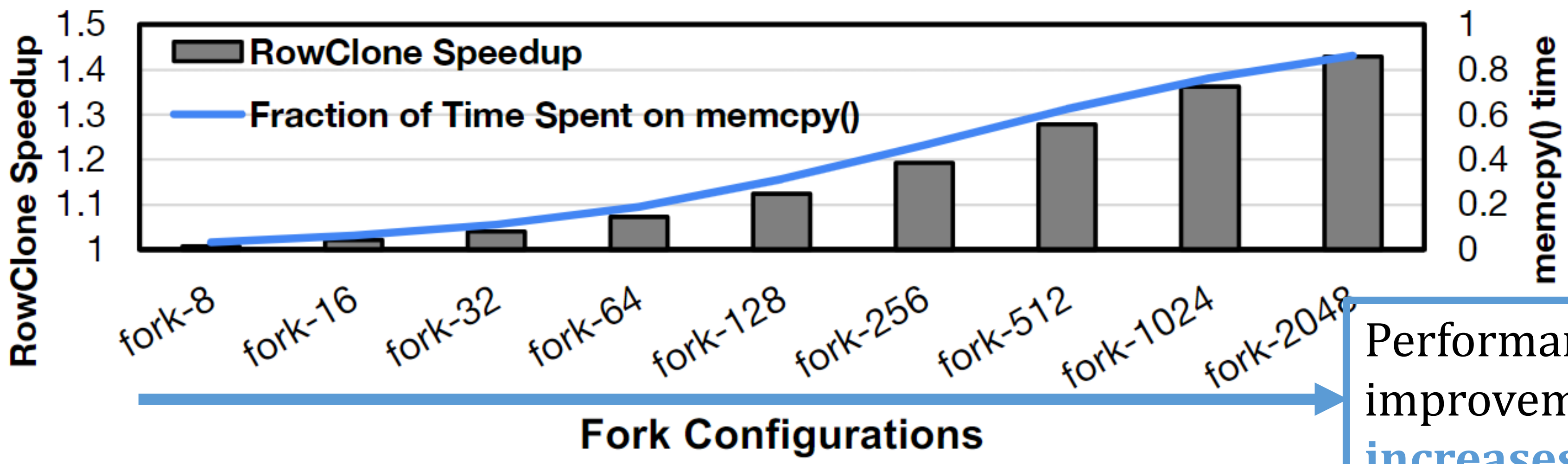
CLFLUSH Overhead



CLFLUSH dramatically reduces the potential throughput improvement

Other Workloads

forkbench (copy-heavy workload)



Performance improvement **increases**

compile (initialization-heavy workload)

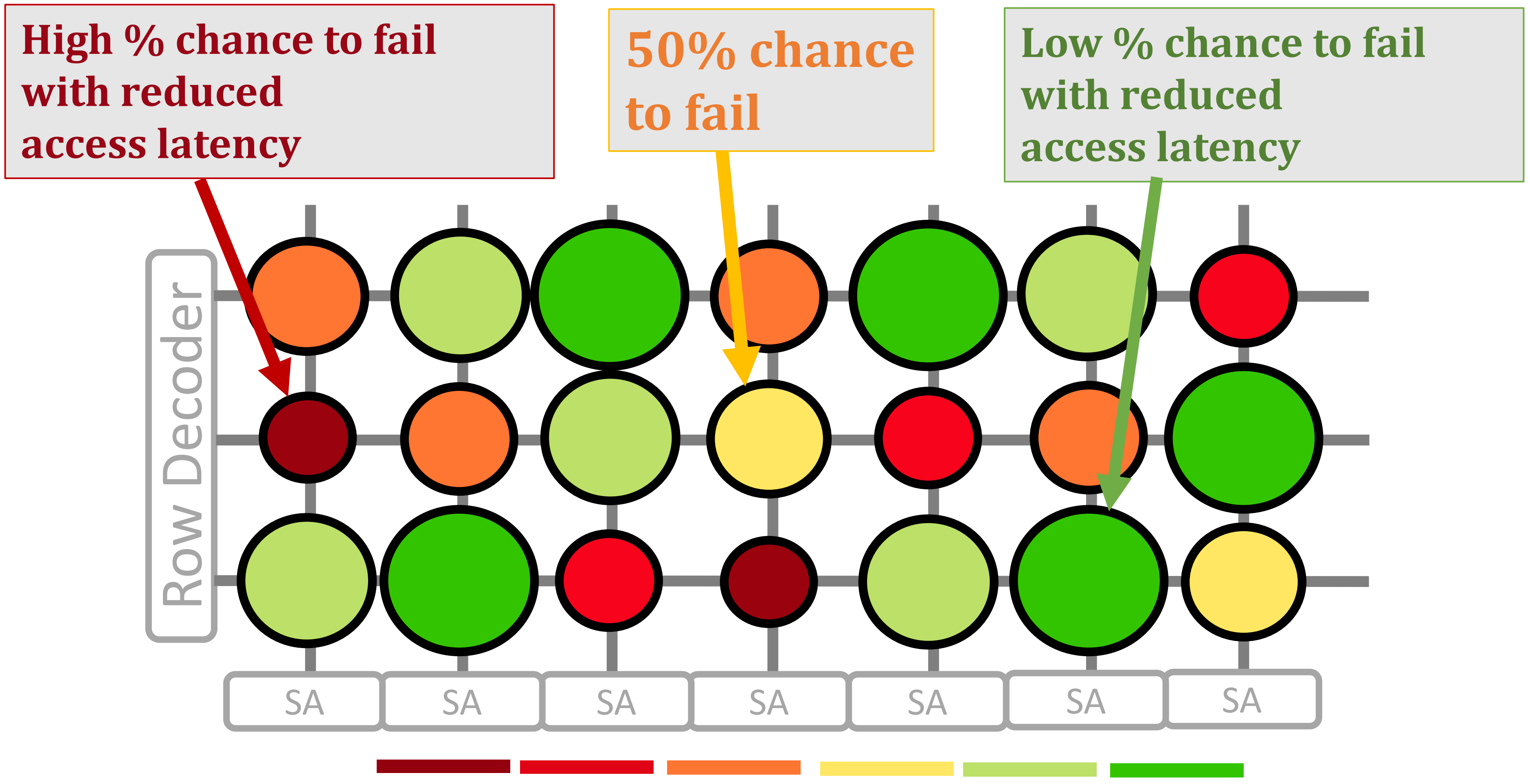
- 9% execution time reduction by in-DRAM initialization
  - 17% of compile's execution time is spent on initialization

SPEC2006 libquantum

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

Outline

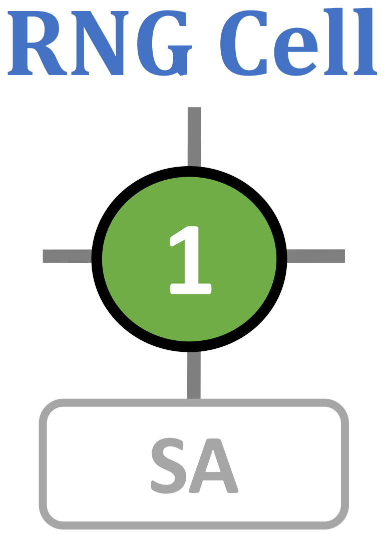
Recall: D-RaNGe Key Idea


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



D-RaNGe Implementation

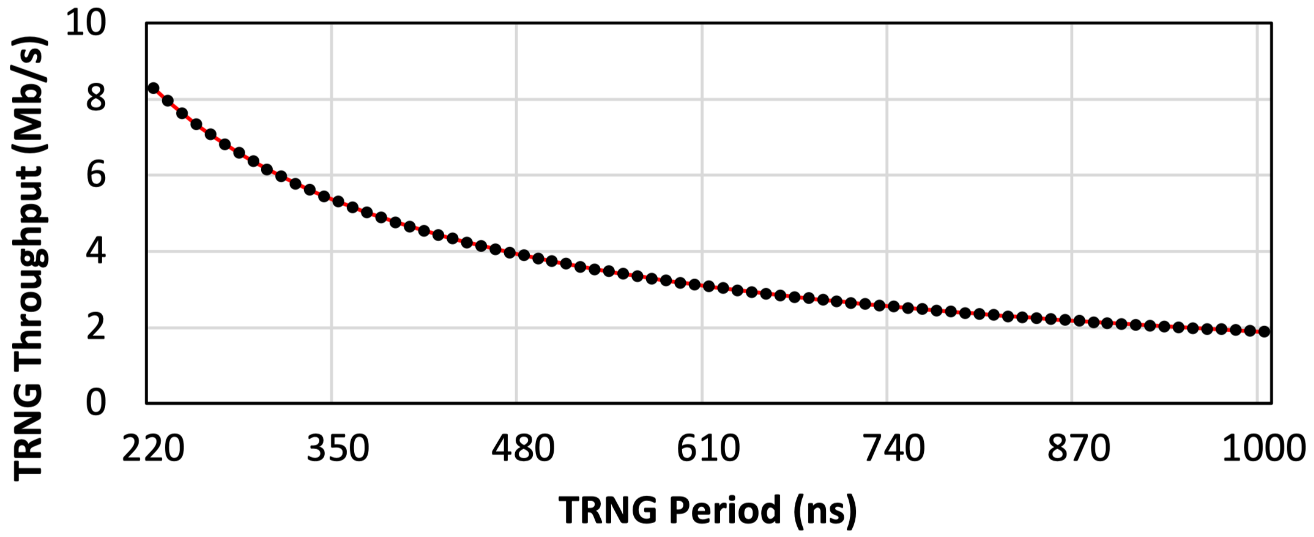
Periodically generate true random numbers by accessing the identified cache block

- 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

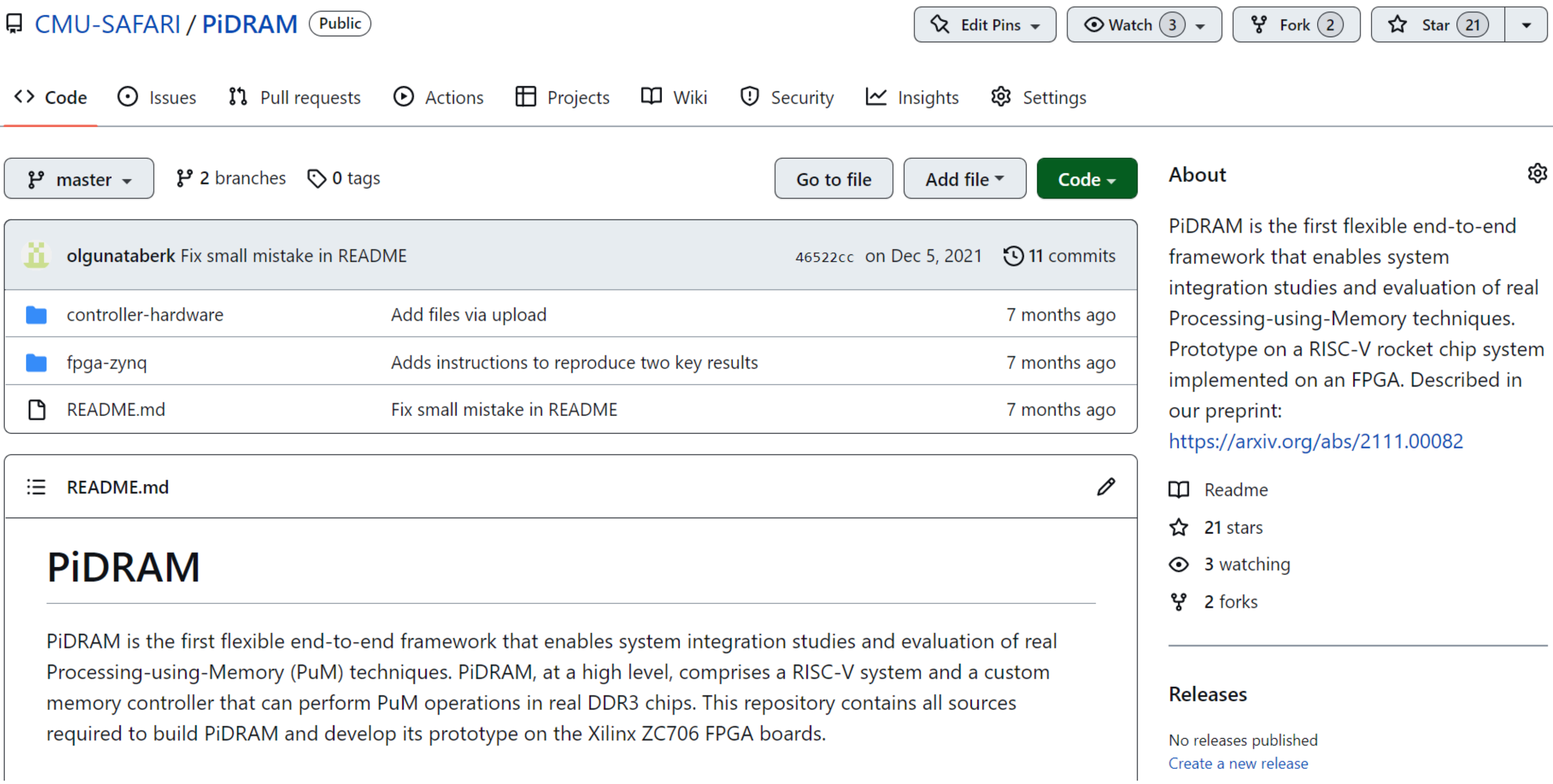
Evaluation: End-to-end integration of two PiM techniques on PiDRAM's FPGA prototype

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



CMU-SAFARI / PiDRAM Public

Edit Pins · Watch 3 · Fork 2 · Star 21

Code · Issues · Pull requests · Actions · Projects · Wiki · Security · Insights · Settings

master · 2 branches · 0 tags · Go to file · Add file · Code

olgunataberk Fix small mistake in README · 46522cc on Dec 5, 2021 · 11 commits

controller-hardware	Add files via upload	7 months ago
fpga-zynq	Adds instructions to reproduce two key results	7 months ago
README.md	Fix small mistake in README	7 months ago

README.md

PiDRAM

PiDRAM is the first flexible end-to-end framework that enables system integration studies and evaluation of real Processing-using-Memory (PuM) techniques. PiDRAM, at a high level, comprises a RISC-V system and a custom memory controller that can perform PuM operations in real DDR3 chips. This repository contains all sources required to build PiDRAM and develop its prototype on the Xilinx ZC706 FPGA boards.

About

PiDRAM is the first flexible end-to-end framework that enables system integration studies and evaluation of real Processing-using-Memory techniques. Prototype on a RISC-V rocket chip system implemented on an FPGA. Described in our preprint: https://arxiv.org/abs/2111.00082

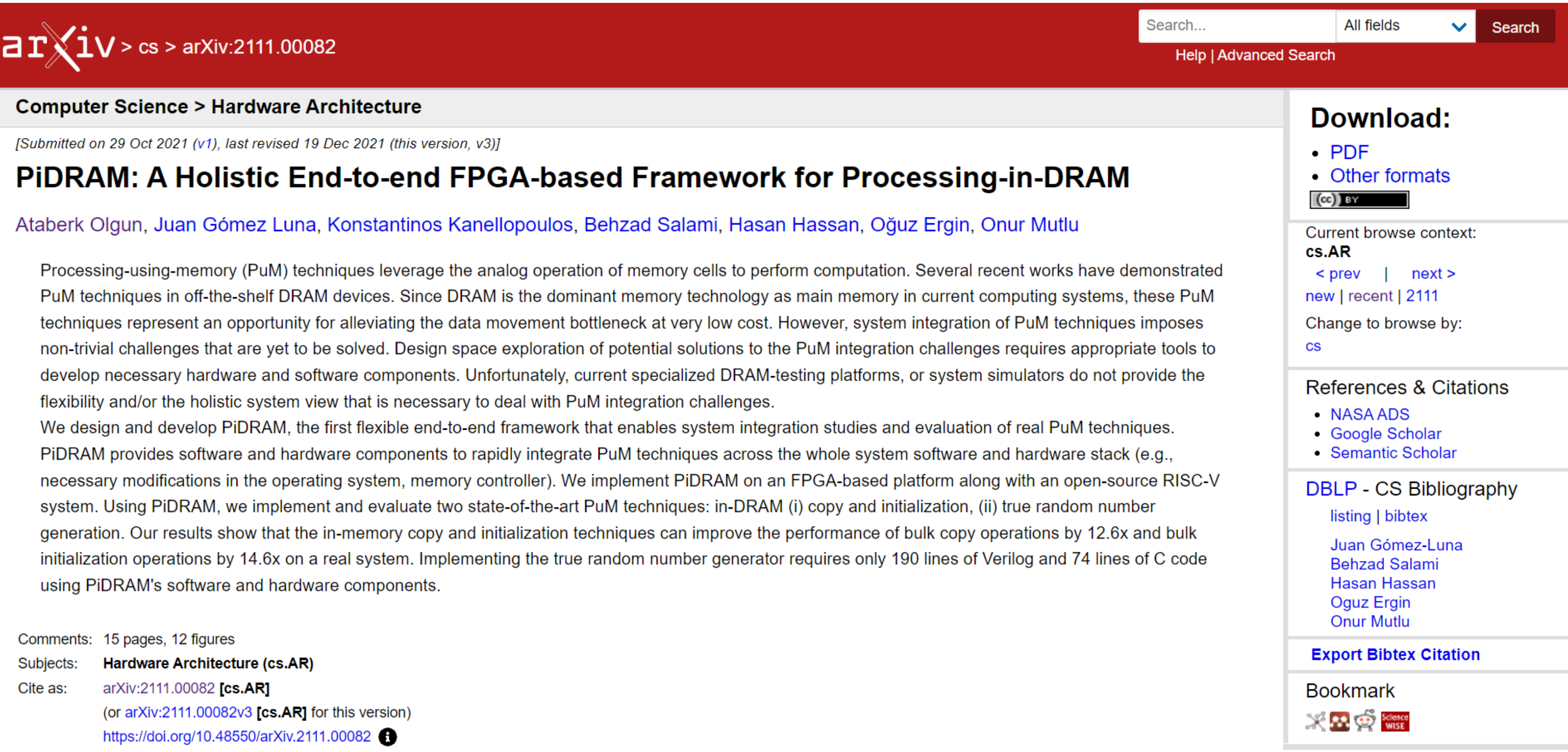
Readme
21 stars
3 watching
2 forks

Releases

No releases published
Create a new release

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

Download:
- PDF
- Other formats

Current browse context:
cs.AR
< prev | next >
new | recent | 2111
Change to browse by:
cs

References & Citations
- NASA ADS
- Google Scholar
- Semantic Scholar

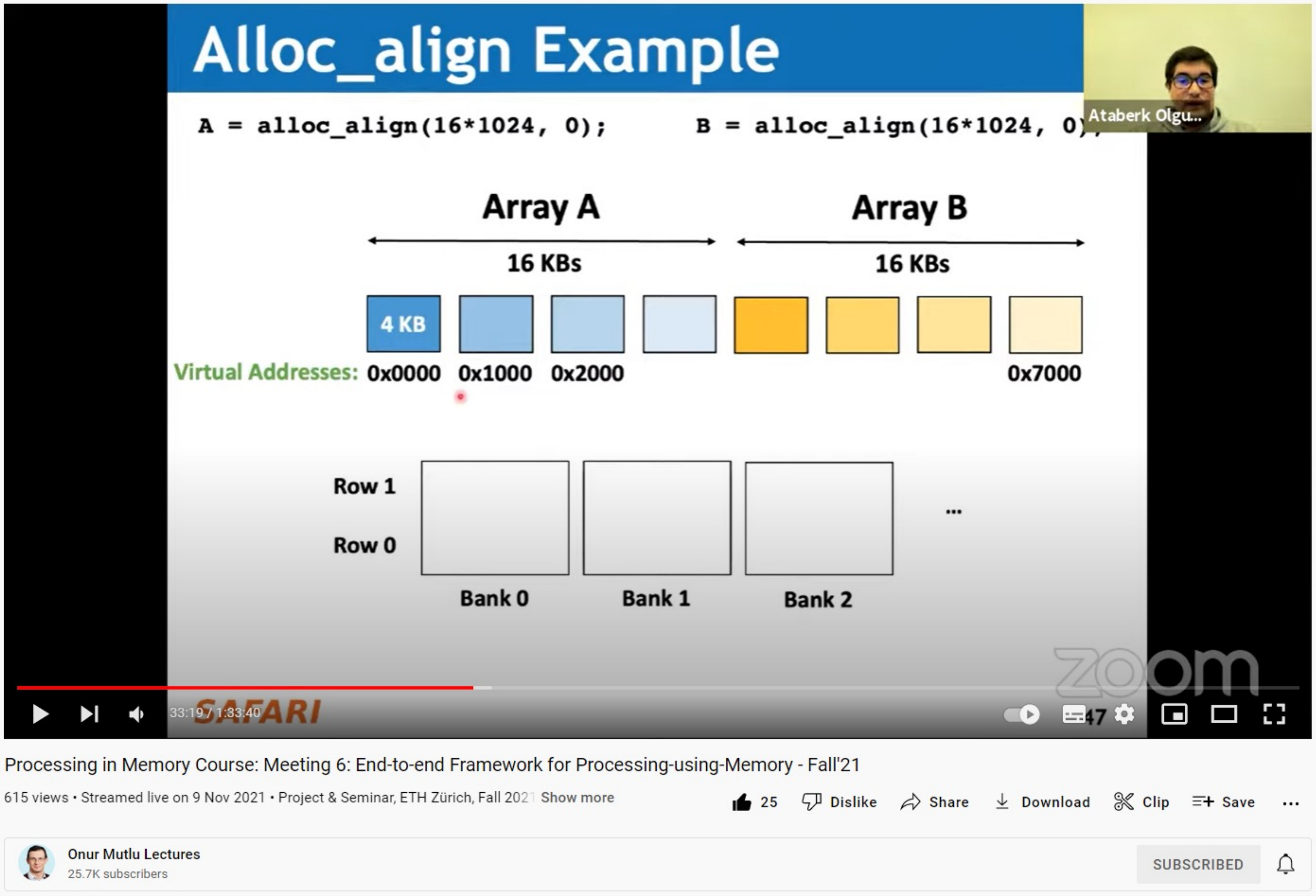
DBLP - CS Bibliography
listing | bibtex
Juan Gómez-Luna
Behzad Salami
Hasan Hassan
Oguz Ergin
Onur Mutlu

Export Bibtex Citation

Bookmark

Long Talk + Tutorial on Youtube

https://youtu.be/s_z_S6FYpC8



Alloc_align Example

```
A = alloc_align(16*1024, 0);        B = alloc_align(16*1024, 0);
```

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 kasırga

ETH zürich TOBB ETÜ University of Economics & Technology

DRAM Bender An Extensible and Versatile FPGA-based Infrastructure to Easily Test State-of-the-art DRAM Chips

Ataberk Olgun Hasan Hassan A. Giray Yaglikci
Yahya Can Tugrul Lois Orosa Haocong Luo
Minesh Patel Oguz Ergin Onur Mutlu

SAFARI ETH zürich

Factors Affecting DRAM Reliability and Latency

DRAM timing violation *Inter-cell interference* *Manufacturing process* *Temperature* *Voltage* •••

Factors affecting DRAM reliability and latency
cannot be properly **modeled** in simulation or analytically

We need to perform experimental studies
of real DRAM chips

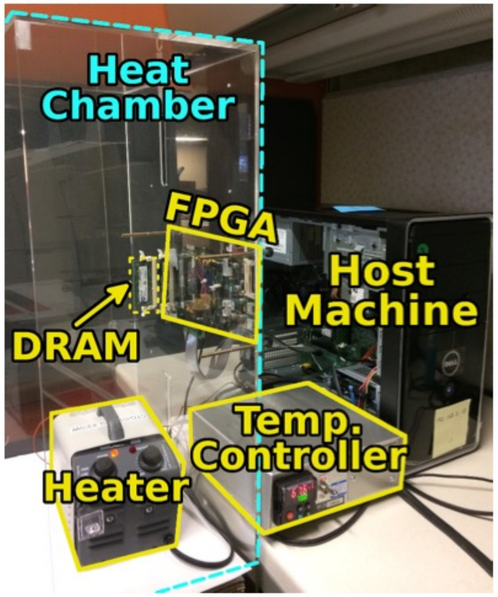
DRAM Testing Infrastructure

Allow experimental studies of real DRAM chips

Open-source FPGA-based testing infrastructure

- Publicly-available: Start using today
- Relatively low cost: An FPGA board + DRAM modules

SoftMC



Litex Tester

Limitations of Existing Infrastructure

Testing Infrastructure	**Interface (IF) Restrictions**	**Ease of Use**	**Extensibility**
SoftMC [134]	Data IF	✗	✗
LiteX RowHammer Tester (LRT) [17]	Command & Data IF	✗	✓
DRAM Bender (this work)	**No Restrictions**	✓	✓

Impose restrictions on the DDR4 interface.
Restrictions limit various characterization experiments.

Difficult to set up (based on discontinued HW/SW)
and use (require developing HW)

Monolithic hardware design
makes extensions (new standards, prototypes) relatively difficult

DRAM Bender: Design Goals

- Flexibility
  - Ability to test any DRAM operation
  - Ability to test any combination of DRAM operations and custom timing parameters

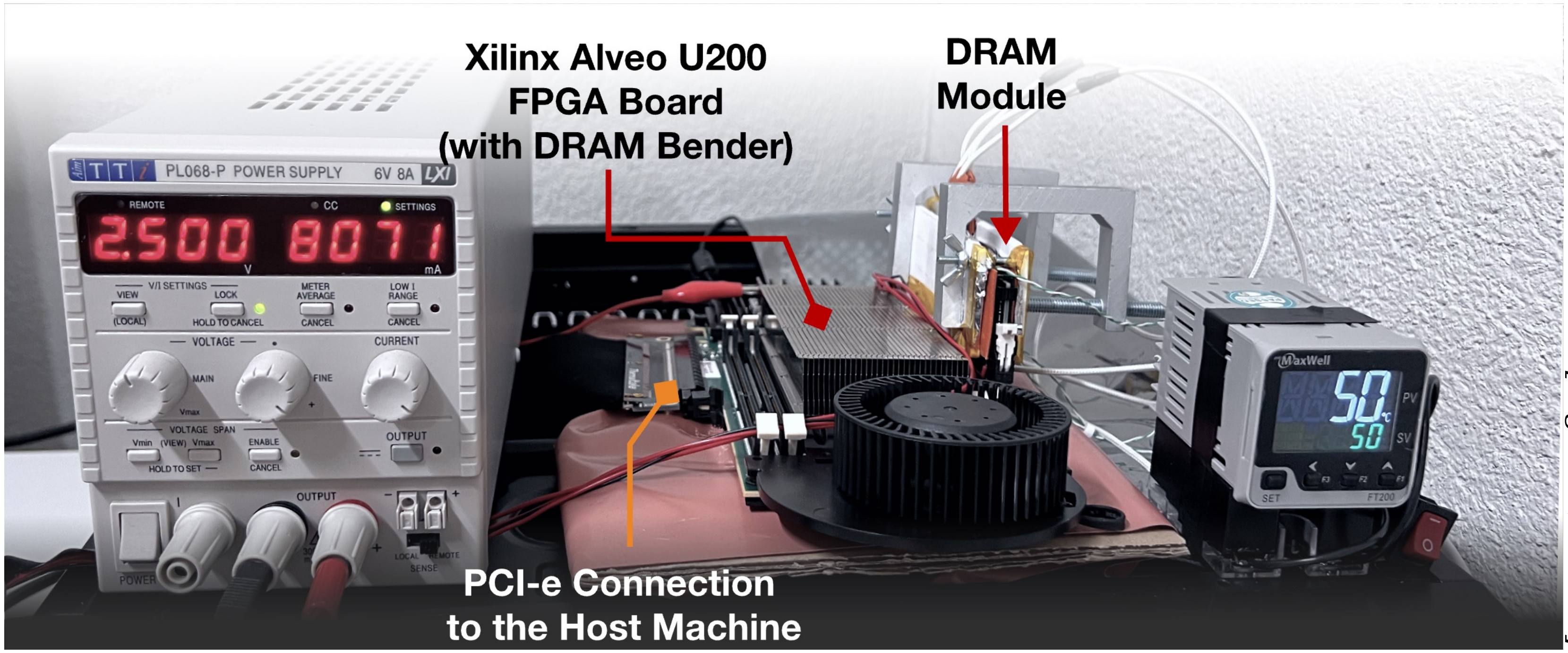
- Ease of use
  - Simple programming interface (C++)
  - Minimal programming effort and time
  - Accessible to a wide range of users
    - *who may lack experience in hardware design*

- Extensibility
  - Modular design
  - Well-defined interfaces between hardware modules

DRAM Bender: Overview

Publicly-available FPGA-based
DDR4/3 (and HBM2) characterization infrastructure

Easily programmable using the DRAM Bender C++ API



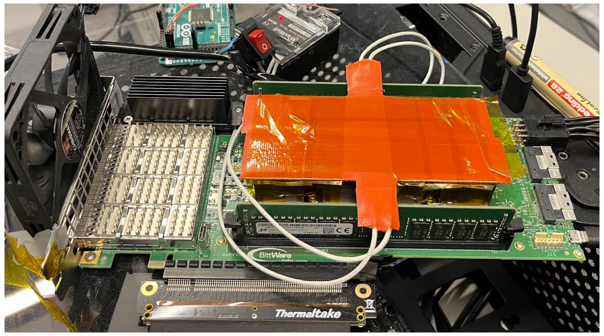
[Yaglikci+, DSN'22]

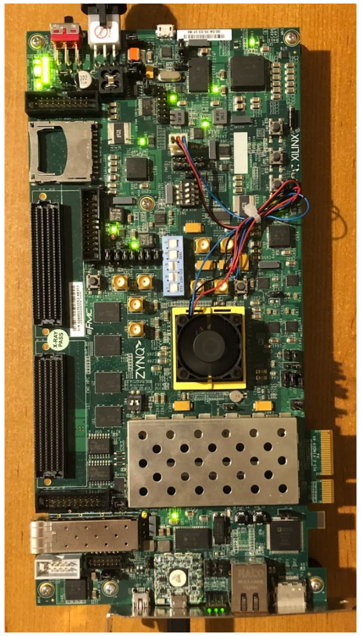
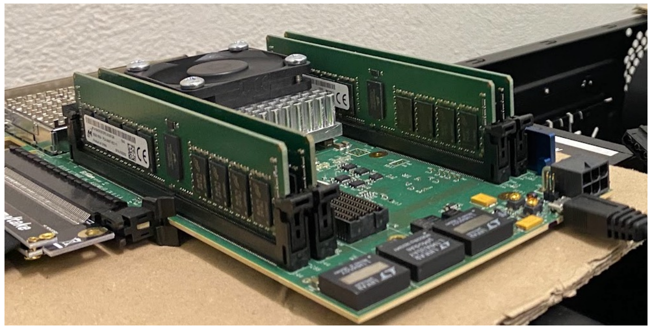
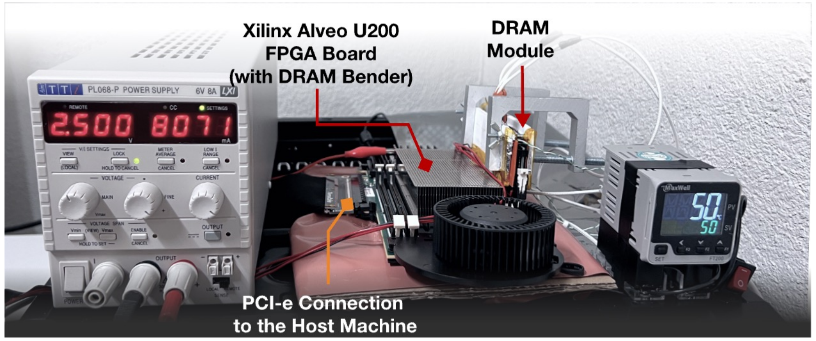
SAFARI

DRAM Bender: Prototypes

Testing Infrastructure	Protocol Support	FPGA Support
SoftMC [134]	DDR3	One Prototype
LiteX RowHammer Tester (LRT) [17]	DDR3/4, LPDDR4	Two Prototypes
DRAM Bender (this work)	**DDR3/DDR4**	**Five Prototypes**

Five out of the box FPGA-based prototypes



DRAM Bender is Flexible

1. RowHammer: Interleaving Pattern of Activations
   - Interleaving pattern significantly affects the number of RowHammer bitflips

2. RowHammer: Random Data Patterns
   - Use 512-bit random data patterns
   - Uncover more bitflips than 8-bit SoftMC random patterns

3. In-DRAM Bitwise Operations
   - Demonstrate in-DRAM bitwise AND/OR capability in real DDR4 chips

DRAM Bender is flexible:
supports many different types of experiments

DRAM Bender is Easy to Use

Easily programmable using the DRAM Bender C++ API

3. In-DRAM Bitwise Operations

```
p.appendACT(BANK, false, R1, false, N);
p.appendPRE(BANK, false, false, M);
p.appendACT(BANK, false, R2, false);
```

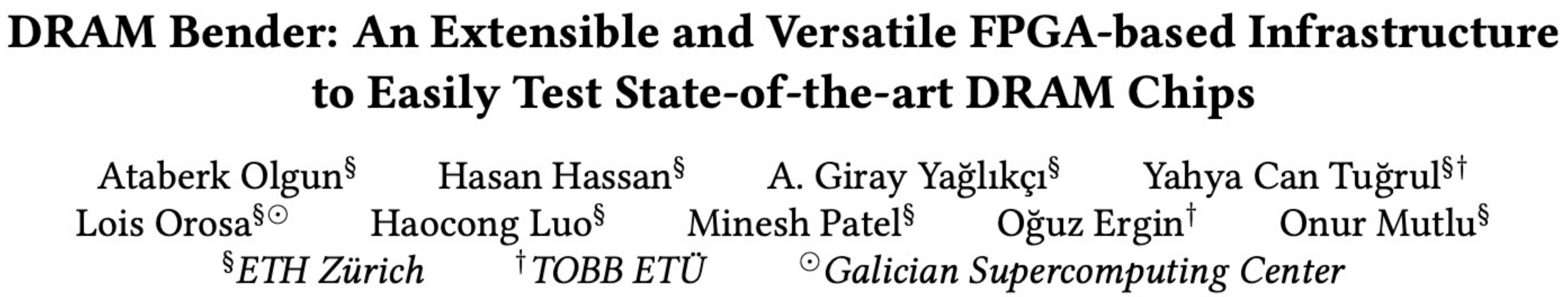
Listing 3: DRAM Bender code segment to perform a bitwise majority operation

Easy to devise new experiments to uncover new insights

More in the paper (II)

- DRAM Bender design details
  - DRAM Bender instruction set architecture
  - Hardware & software modules
  - Prototype design
  - Temperature controller setup
- DRAM Bender application programming interface
- Detailed results for three case studies
- Future work & improvements

More in the paper (II)



DRAM Bender: An Extensible and Versatile FPGA-based Infrastructure to Easily Test State-of-the-art DRAM Chips

Ataberk Olgun[§] Hasan Hassan[§] A. Giray Yağlıkçı[§] Yahya Can Tuğrul[§†]
Lois Orosa[§⊙] Haocong Luo[§] Minesh Patel[§] Oğuz Ergin[†] Onur Mutlu[§]
[§]*ETH Zürich* [†]*TOBB ETÜ* [⊙]*Galician Supercomputing Center*

https://arxiv.org/abs/2211.05838

Research DRAM Bender Enabled

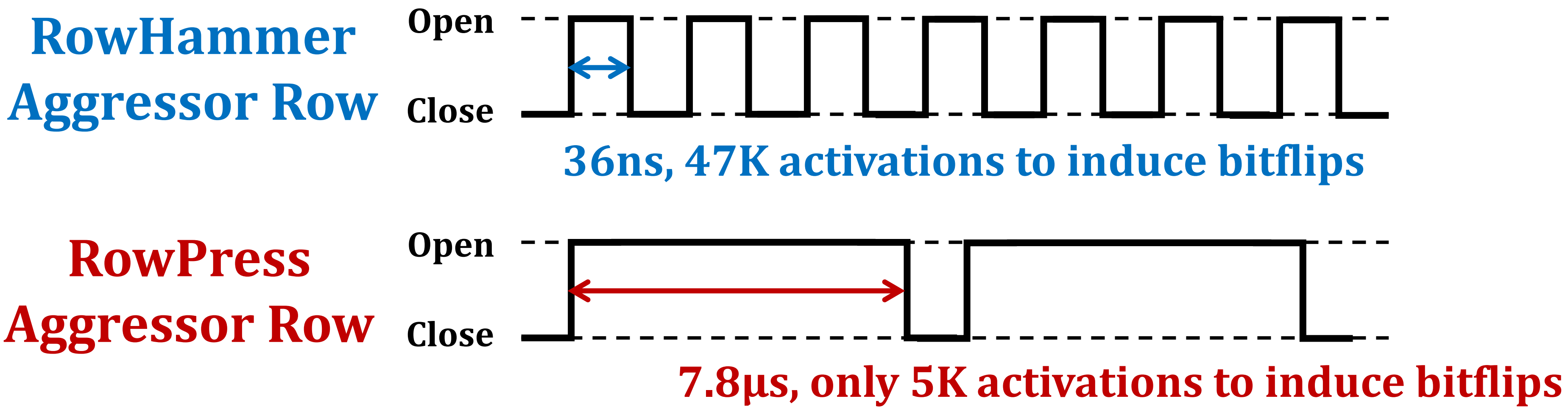
1) **[ISCA'23]** Luo+, "RowPress: Amplifying Read Disturbance in Modern DRAM Chips"
2) **[DSN'23 Disrupt]** Olgun+, "An Experimental Analysis of RowHammer on HBM2 DRAM Chips"
3) **[arXiv Preprint, 2023]** Orosa+, "SpyHammer: Using RowHammer to Remotely Spy on Temperature"
4) **[MICRO'22]** Yaglikci+, "HIRA: Hidden Row Activation for Reducing Refresh Latency of Off-the-Shelf DRAM Chips"
5) **[DSN'22]** Yaglikci+, "Understanding RowHammer Under Reduced Wordline Voltage: An Experimental Study Using Real DRAM Devices"
6) **[MICRO'21]** Orosa+, "A Deeper Look into RowHammer's Sensitivities: Experimental Analysis of Real DRAM Chips and Implications on Future Attacks and Defenses"
7) **[MICRO'21]** Hassan+, "Uncovering In-DRAM RowHammer Protection Mechanisms: A New Methodology, Custom RowHammer Patterns, and Implications"
8) **[ISCA'21]** Olgun+, "QUAC-TRNG: High-Throughput True Random Number Generation Using Quadruple Row Activation in Commodity DRAM Chips"
9) **[ISCA'21]** Orosa+, "CODIC: A Low-Cost Substrate for Enabling Custom In-DRAM Functionalities and Optimizations"
10) **[ISCA'20]** Kim+, "Revisiting RowHammer: An Experimental Analysis of Modern Devices and Mitigation Techniques"
11) **[S&P'20]** Frigo+, "TRRespass: Exploiting the Many Sides of Target Row Refresh"
12) **[HPCA'19]** Kim+, "D-RaNGe: Using Commodity DRAM Devices to Generate True Random Numbers with Low Latency and High Throughput"
13) **[MICRO'19]** Koppula+, "EDEN: Enabling Energy-Efficient, High-Performance Deep Neural Network Inference Using Approximate DRAM"
14) **[SIGMETRICS'18]** Ghose+, "What Your DRAM Power Models Are Not Telling You: Lessons from a Detailed Experimental Study"
15) **[SIGMETRICS'17]** Chang+, "Understanding Reduced-Voltage Operation in Modern DRAM Devices: Experimental Characterization, Analysis, and Mechanisms"
16) **[MICRO'17]** Khan+, "Detecting and Mitigating Data-Dependent DRAM Failures by Exploiting Current Memory Content"
17) **[SIGMETRICS'16]** Chang+, "Understanding Latency Variation in Modern DRAM Chips: Experimental Characterization, Analysis, and Optimization"

A Highlight: RowPress

Keeping a DRAM row **open for a long time** causes bitflips in adjacent rows

These bitflips do **NOT** require many row activations

Only one activation is enough in some cases!



RowHammer Aggressor Row
Open
Close
36ns, 47K activations to induce bitflips
RowPress Aggressor Row
Open
Close
7.8µs, only 5K activations to induce bitflips

RowPress Results & Source Code

RowPress: Amplifying Read Disturbance in Modern DRAM Chips

Haocong Luo Ataberk Olgun A. Giray Yağlıkçı Yahya Can Tuğrul Steve Rhyner
Meryem Banu Cavlak Joël Lindegger Mohammad Sadrosadati Onur Mutlu

ETH Zürich

Fully open source and artifact evaluated

- ➢ https://github.com/CMU-SAFARI/RowPress

RowPress [ISCA 2023]

- Haocong Luo, Ataberk Olgun, Giray Yaglikci, Yahya Can Tugrul, Steve Rhyner, M. Banu Cavlak, Joel Lindegger, Mohammad Sadrosadati, and Onur Mutlu,
  "RowPress: Amplifying Read Disturbance in Modern DRAM Chips"
  Proceedings of the 50th International Symposium on Computer Architecture (***ISCA***), Orlando, FL, USA, June 2023.
  [Slides (pptx) (pdf)]
  [Lightning Talk Slides (pptx) (pdf)]
  [Lightning Talk Video (3 minutes)]
  [RowPress Source Code and Datasets (Officially Artifact Evaluated with All Badges)]
  Officially artifact evaluated as available, reusable and reproducible.
  Best artifact award at ISCA 2023.

RowPress: Amplifying Read-Disturbance in Modern DRAM Chips

Haocong Luo   Ataberk Olgun   A. Giray Yağlıkçı   Yahya Can Tuğrul   Steve Rhyner
Meryem Banu Cavlak   Joël Lindegger   Mohammad Sadrosadati   Onur Mutlu
ETH Zürich

More Research DRAM Bender Enabled

18) **[DRAMSec'23]** Lang+, "BLASTER: Characterizing the Blast Radius of Rowhammer"

19) **[Applied Sciences'22]** Bepary+, "DRAM Retention Behavior with Accelerated Aging in Commercial Chips"

20) **[ETS'21]** Farmani+, "RHAT: Efficient RowHammer-Aware Test for Modern DRAM Modules"

21) **[HOST'20]** Talukder+, "Towards the Avoidance of Counterfeit Memory: Identifying the DRAM Origin"

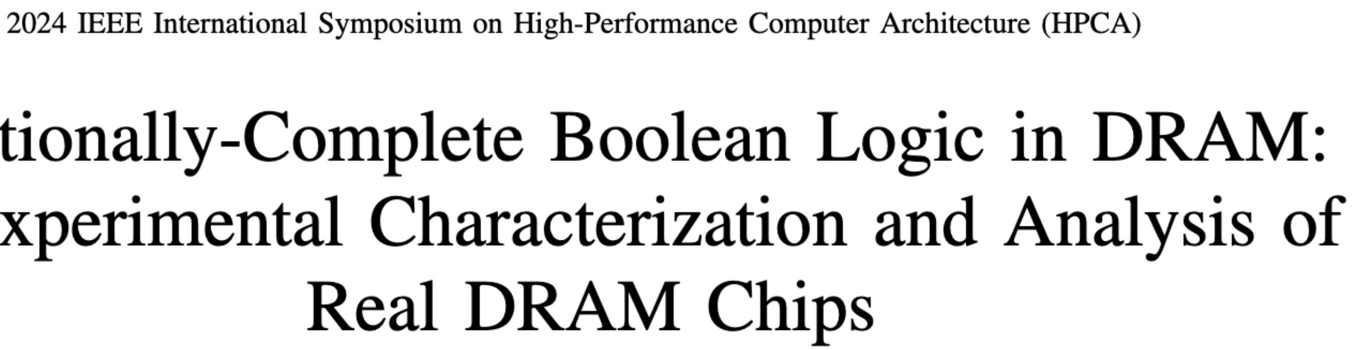
22) **[MICRO'19]** Gao+, "ComputeDRAM: In-Memory Compute Using Off-the-Shelf DRAMs"

23) **[IEEE Access'19]** Talukder+, "PreLatPUF: Exploiting DRAM Latency Variations for Generating Robust Device Signatures"

24) **[ICCE'18]** Talukder+, "Exploiting DRAM Latency Variations for Generating True Random Numbers"

Functionally-Complete Real PUM Prototype

- Ismail Emir Yuksel, Yahya Can Tuğrul, Ataberk Olgun, Nisa Bostanci, A. Giray Yaglikci, Geraldo F. Oliveira, Haocong Luo, Juan Gómez-Luna, Mohammad Sadrosadati, and Onur Mutlu, **"Functionally-Complete Boolean Logic in DRAM: An Experimental Characterization and Analysis of Real DRAM Chips,"** *Proceedings of the 30th International Symposium on High-Performance Computer Architecture* (***HPCA***), Edinburgh, Scotland, March 2024.


2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)

Functionally-Complete Boolean Logic in DRAM: An Experimental Characterization and Analysis of Real DRAM Chips

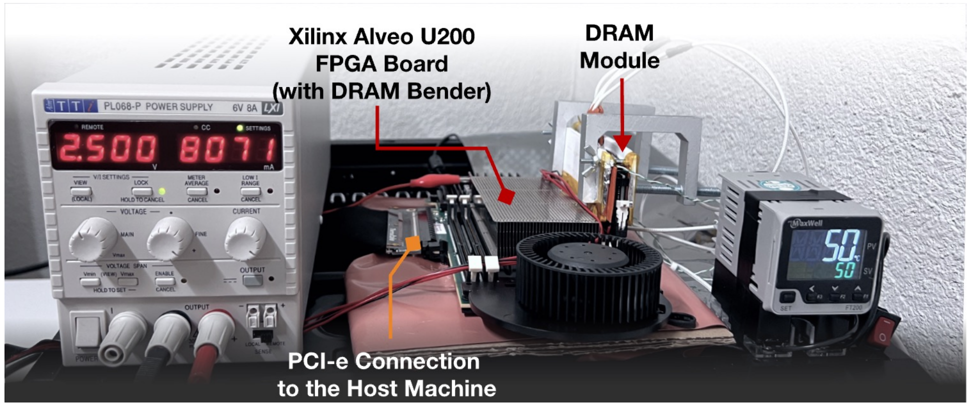
Summary

DRAM Bender

The first **publicly-available** DDR4 characterization infrastructure

- Flexible and Easy to Use
- **Source code** available:

github.com/CMU-SAFARI/DRAMBender

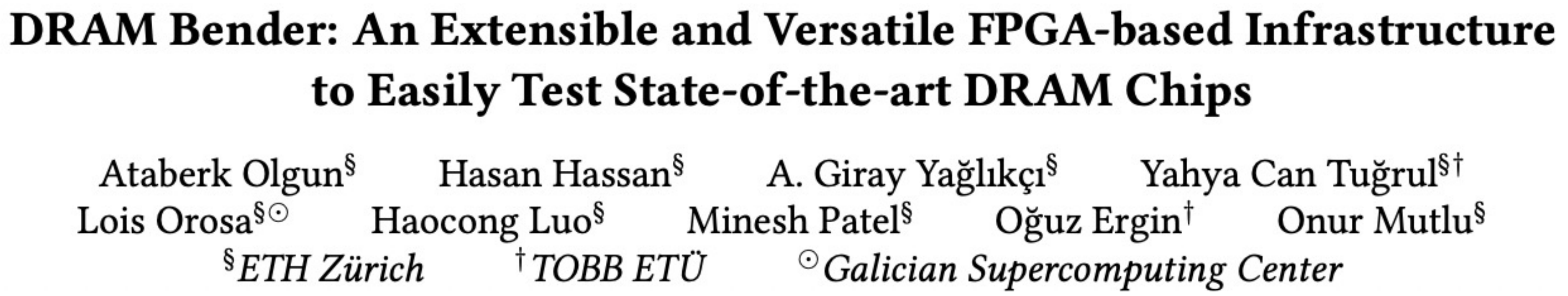

Xilinx Alveo U200 FPGA Board (with DRAM Bender)
DRAM Module
PCI-e Connection to the Host Machine

[Yaglikci+, DSN'22]

DRAM Bender enables many studies, ideas, and methodologies in the design of future memory systems

DRAM Bender

- Ataberk Olgun, Hasan Hassan, A Giray Yağlıkçı, Yahya Can Tuğrul, Lois Orosa, Haocong Luo, Minesh Patel, Oğuz Ergin, and Onur Mutlu,
  "DRAM Bender: An Extensible and Versatile FPGA-based Infrastructure to Easily Test State-of-the-art DRAM Chips"
  IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (***TCAD***), 2023.
  [Extended arXiv version]
  [DRAM Bender Source Code]
  [DRAM Bender Tutorial Video (43 minutes)]



DRAM Bender: An Extensible and Versatile FPGA-based Infrastructure to Easily Test State-of-the-art DRAM Chips

Ataberk Olgun[§] Hasan Hassan[§] A. Giray Yağlıkçı[§] Yahya Can Tuğrul[§†]
Lois Orosa[§⊙] Haocong Luo[§] Minesh Patel[§] Oğuz Ergin[†] Onur Mutlu[§]
[§]*ETH Zürich* [†]*TOBB ETÜ* [⊙]*Galician Supercomputing Center*

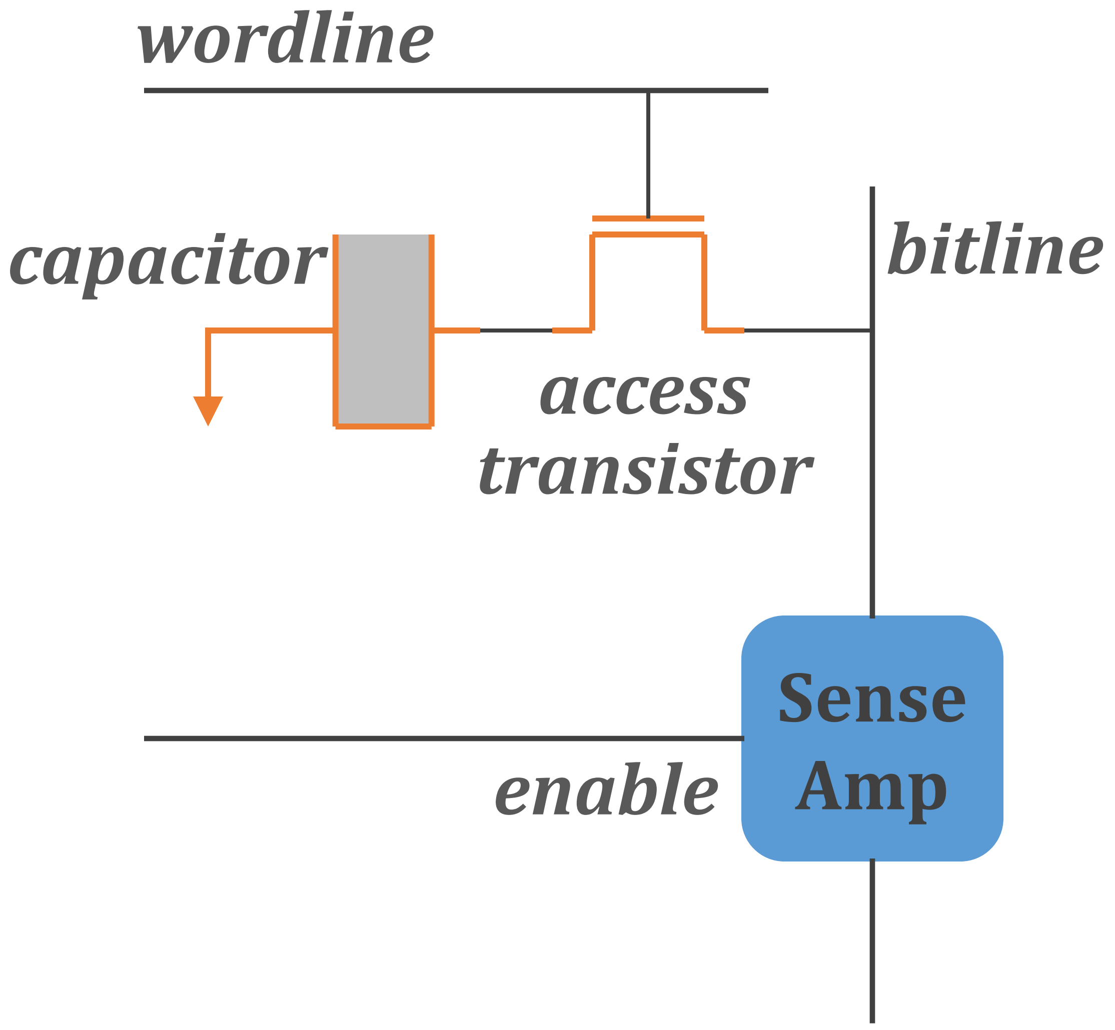
DRAM Bender
An Extensible and Versatile FPGA-based Infrastructure to Easily Test State-of-the-art DRAM Chips

Ataberk Olgun Hasan Hassan A. Giray Yaglikci
Yahya Can Tugrul Lois Orosa Haocong Luo
Minesh Patel Oguz Ergin Onur Mutlu

SAFARI ETH zürich

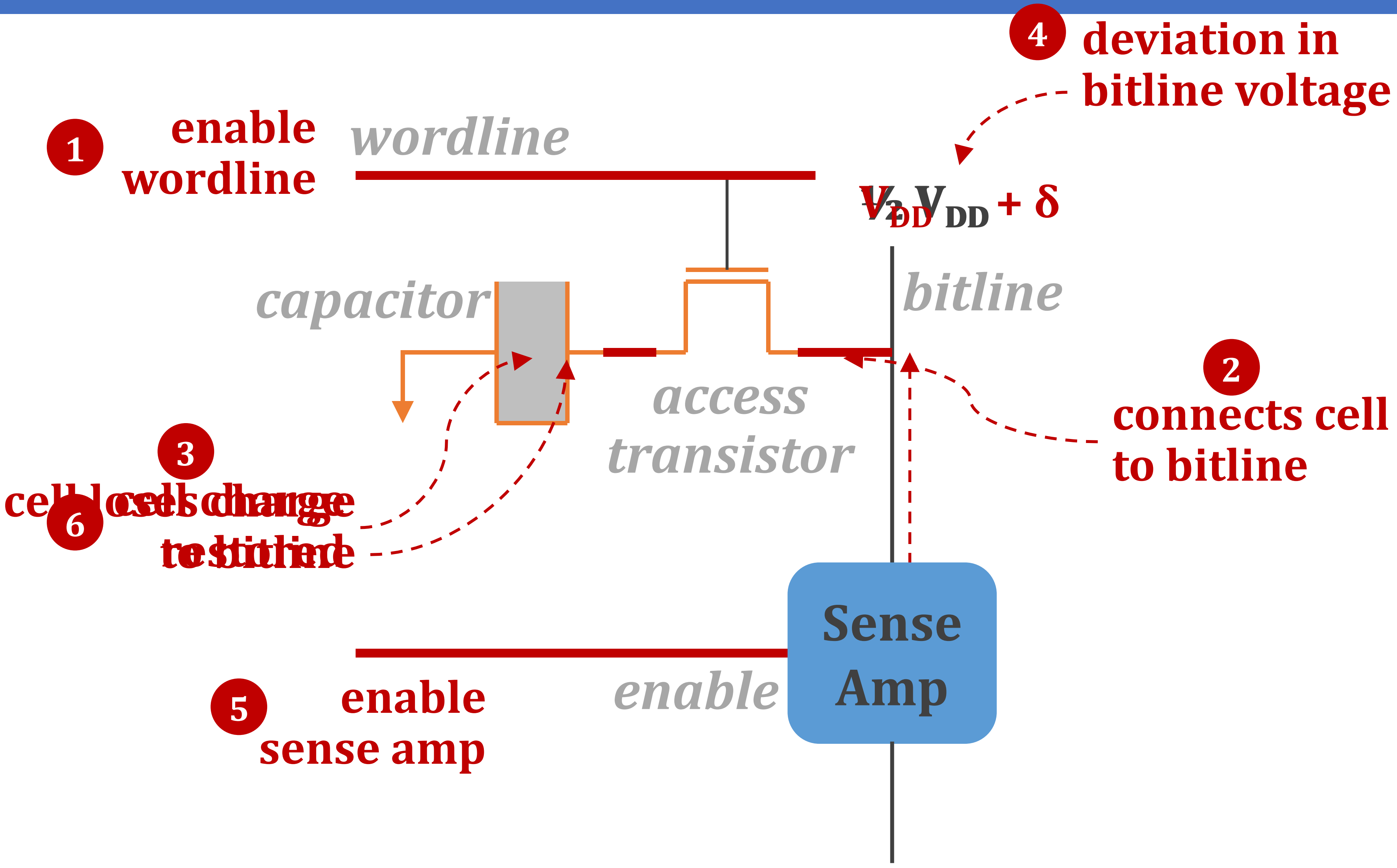
BACKUP SLIDES

Accessing a DRAM Cell



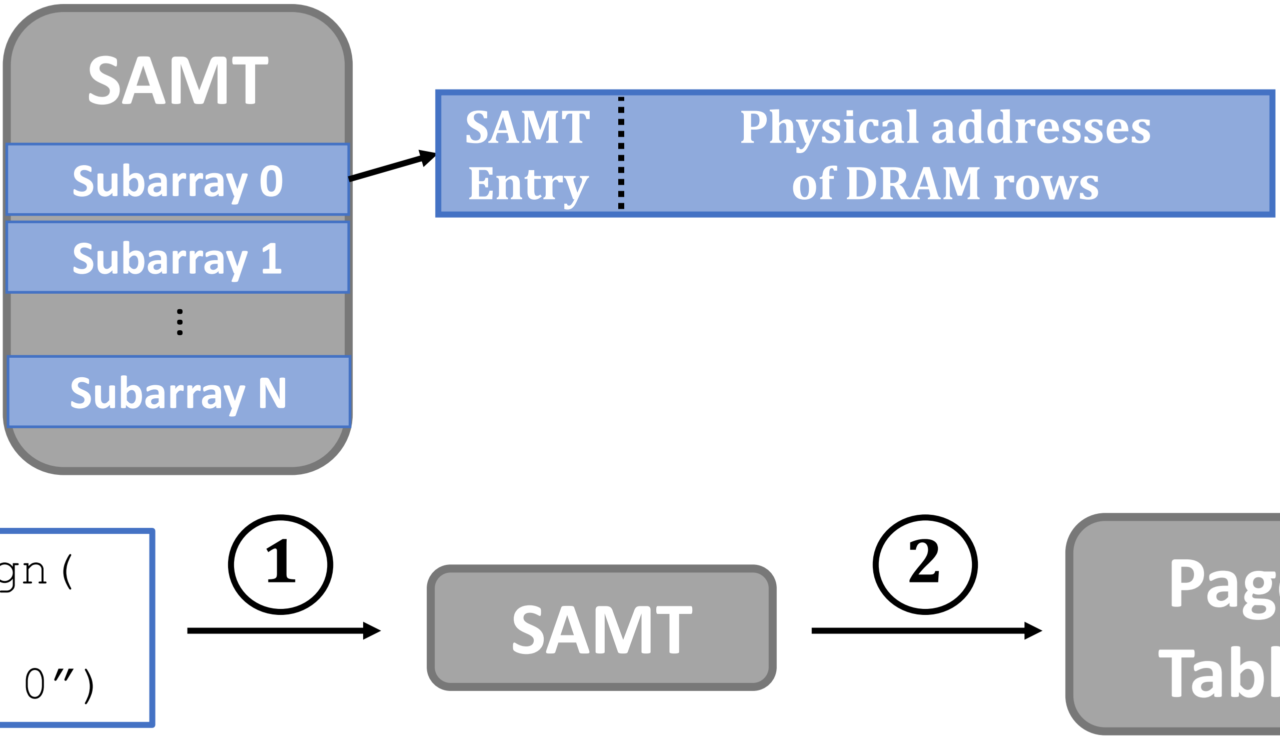
[Seshadri+ MICRO'17]

Accessing a DRAM Cell



alloc_align() function

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



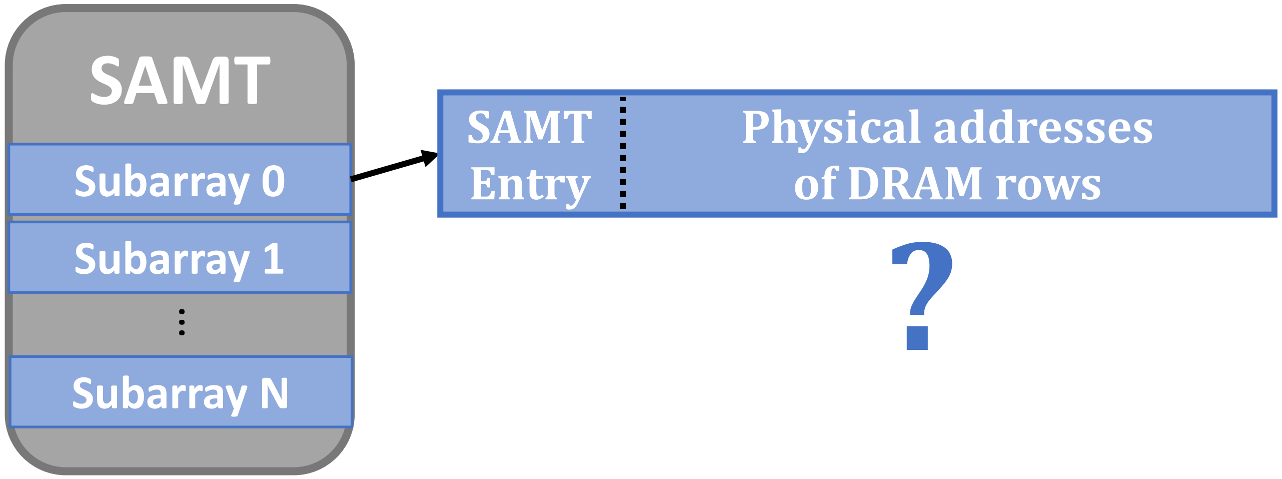
SAMT
Subarray 0
Subarray 1
⋮
Subarray N
SAMT Entry
Physical addresses of DRAM rows
alloc_align(4 KiB, "Subarray 0")
1
SAMT
2
Page Table

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

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

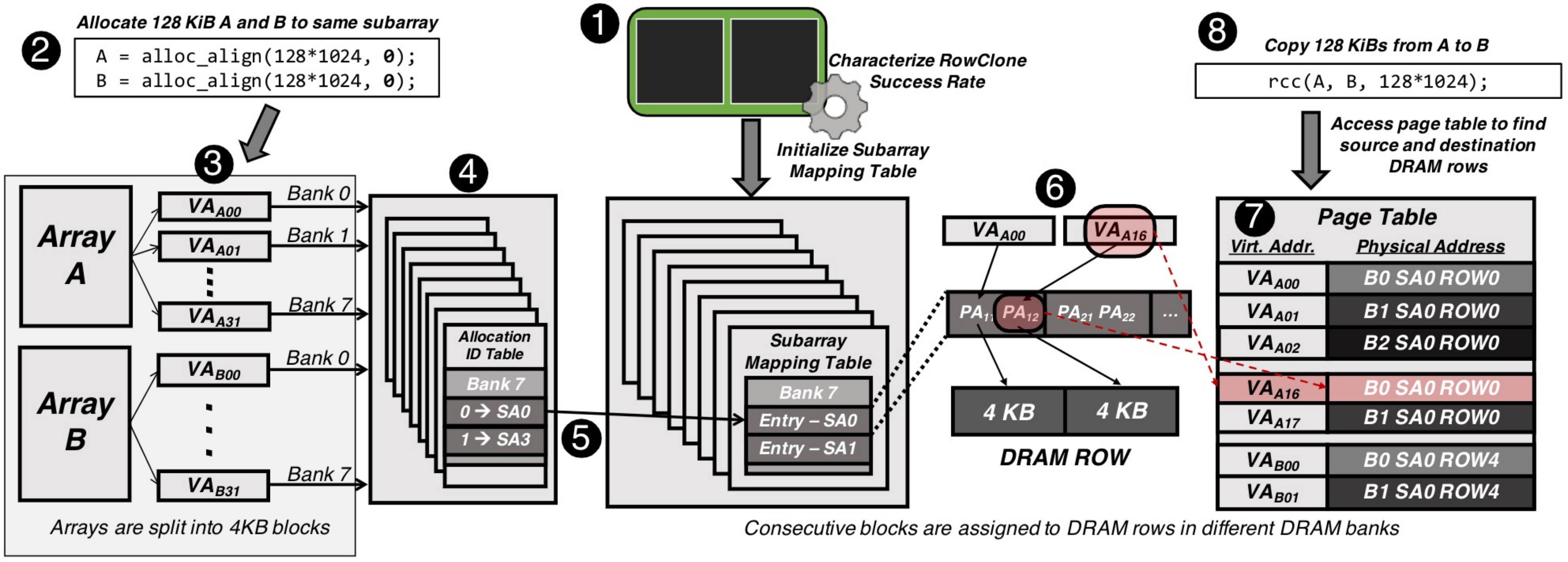
2 Update the page table to map programmer-allocated address to subarray 0

Initializing SAMT



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);
3
Array A
VA A00
VA A01
VA A31
Bank 0
Bank 1
Bank 7
Array B
VA B00
VA B31
Bank 0
Bank 7
Arrays are split into 4KB blocks
4
Allocation ID Table
Bank 7
0 → SA0
1 → SA3
5
1
Characterize RowClone Success Rate
Initialize Subarray Mapping Table
Subarray Mapping Table
Bank 7
Entry – SA0
Entry – SA1
6
VA A00
VA A16
PA11 PA12 PA21 PA22 ...
4 KB
4 KB
DRAM ROW
Consecutive blocks are assigned to DRAM rows in different DRAM banks
8
Copy 128 KiBs from A to B
rcc(A, B, 128*1024);
Access page table to find source and destination DRAM rows
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.

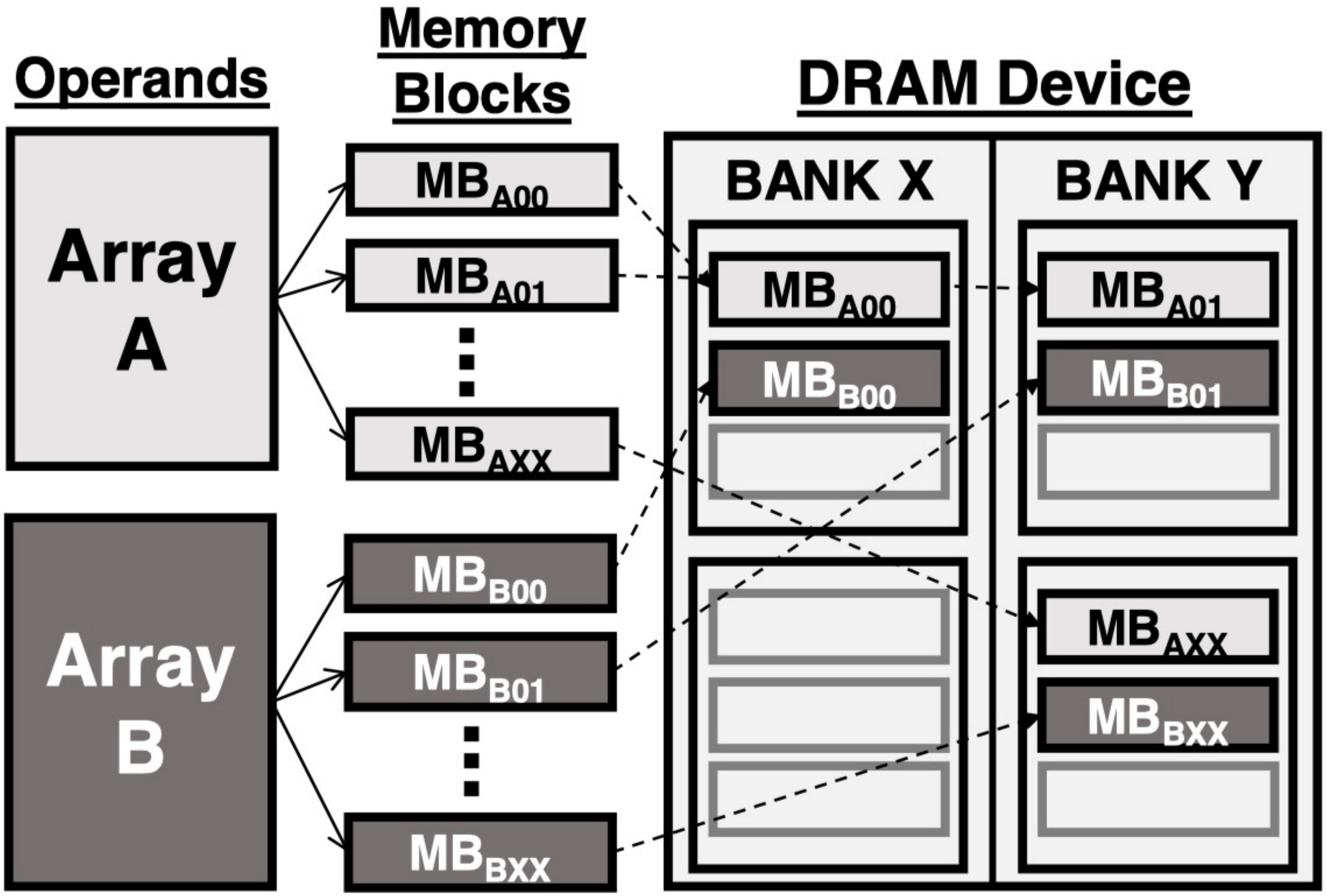


Figure 6: Overview of our memory allocation mechanism

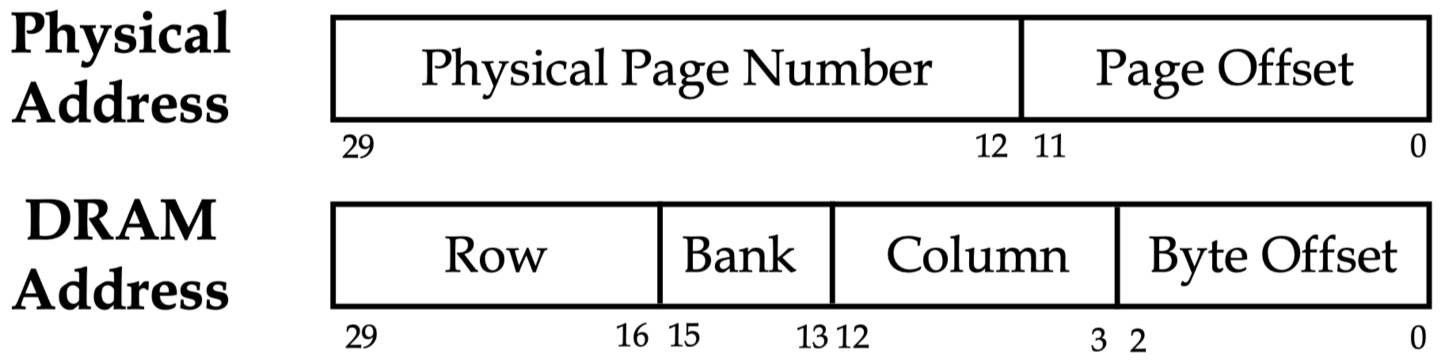


Figure 8: Physical address to DRAM address mapping in PiDRAM. Byte offset is used to address the byte in the DRAM burst.

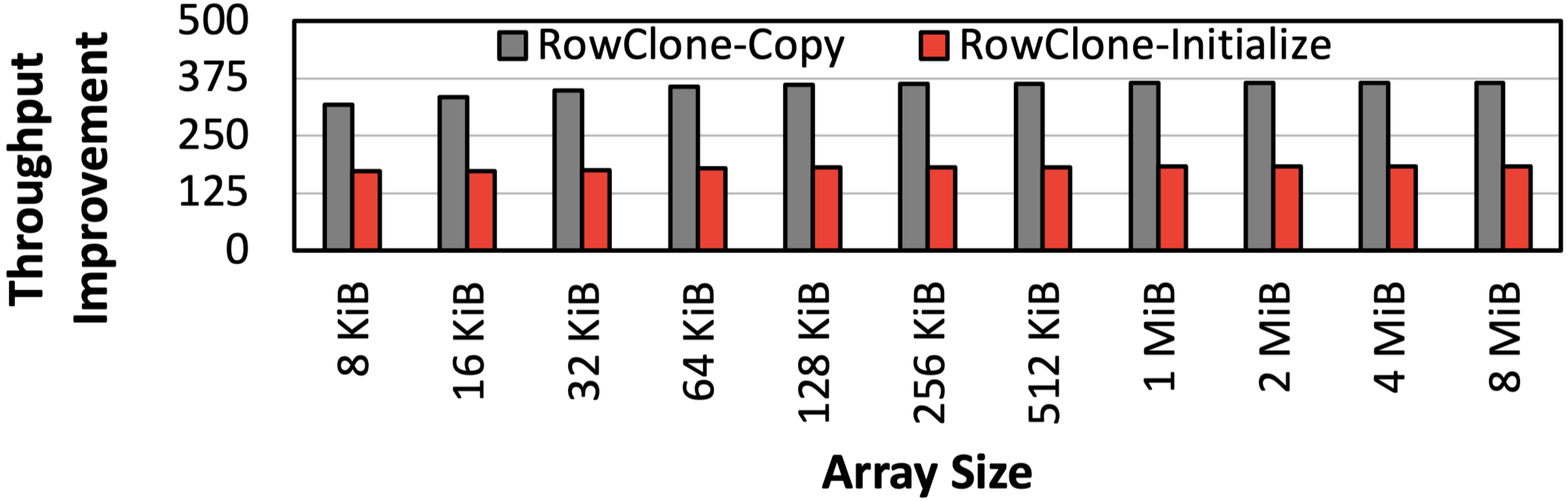
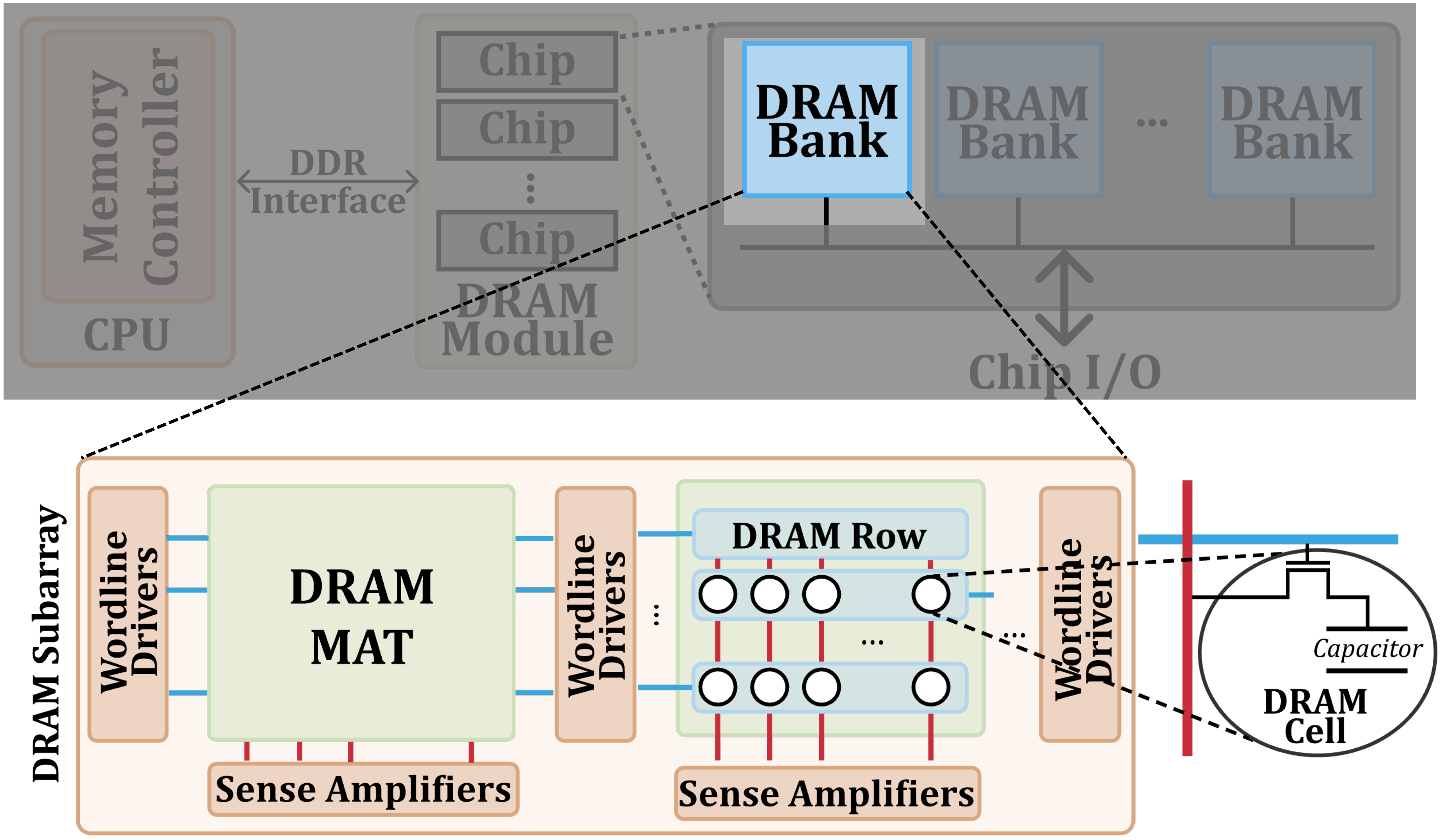


Figure 9: RowClone-Copy and RowClone-Initialize over traditional CPU-copy and -initialization for the Bare-Metal configuration

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

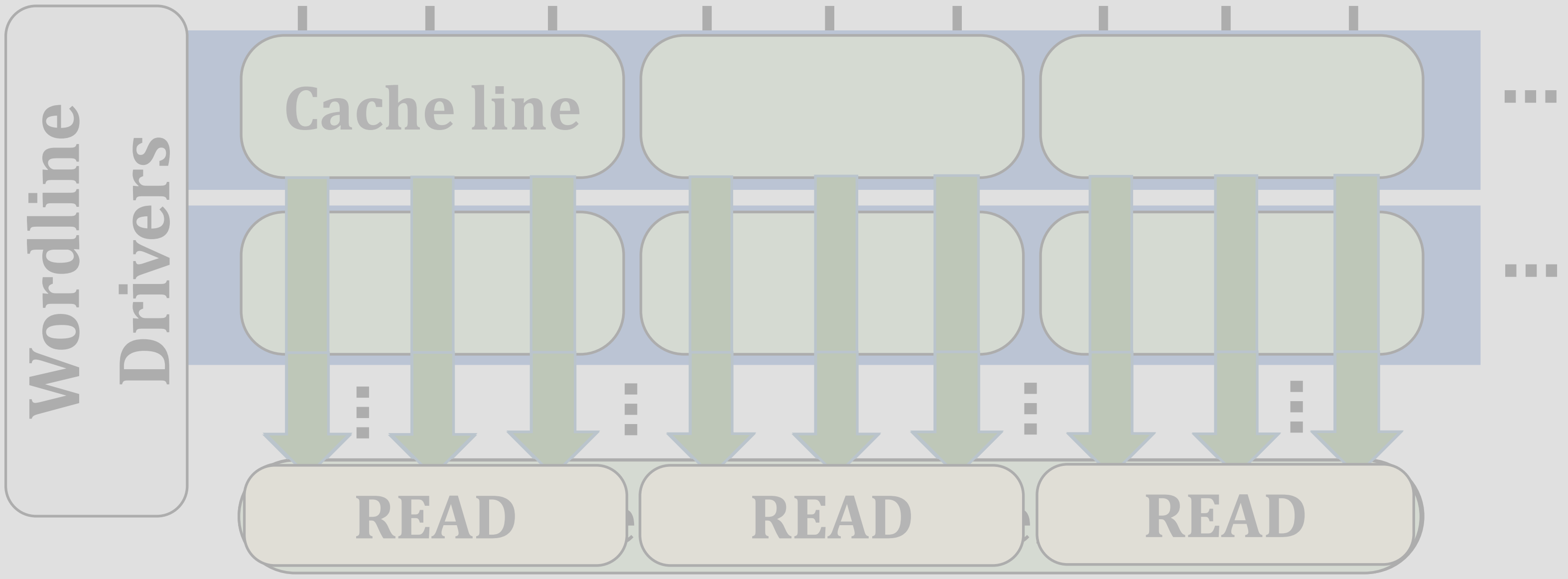
Platforms	**Interface with real DRAM chips**	**Flexible MC for PuM**	**System software support**	**Open-source**
Silent-PIM [78]	✗	✗	✓	✗
SoftMC [60]	✓(DDR3)	✗	✗	✓
ComputeDRAM [44]	✓(DDR3)	✗	✗	✗
MEG [174]	✓(HBM)	✗	✓	✓
PiMulator [119]	✗	✓	✗	✓
Commercial platforms (e.g., ZYNQ [166])	✓(DDR3/4)	✗	✓	✗
Simulators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

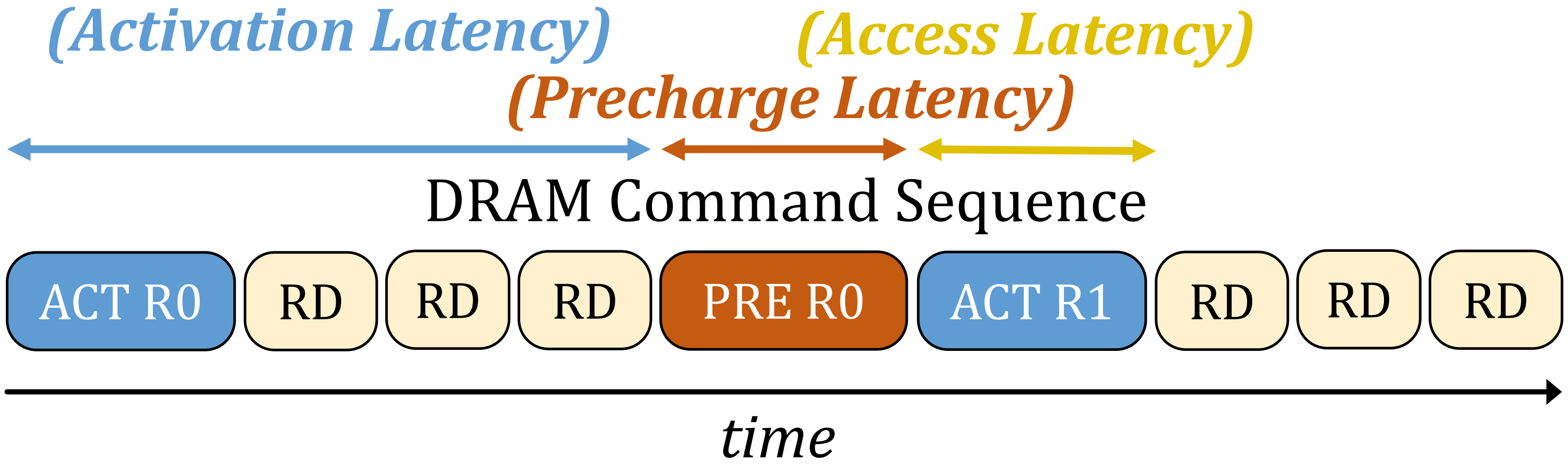
DRAM Organization



[Olgun+ ISCA'21]

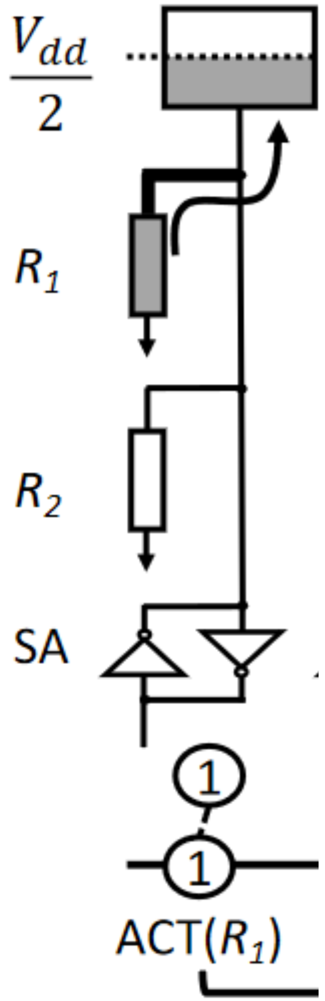
DRAM Operation





[Kim+ HPCA'19]

Row-copy in ComputeDRAM



More in ComputeDRAM

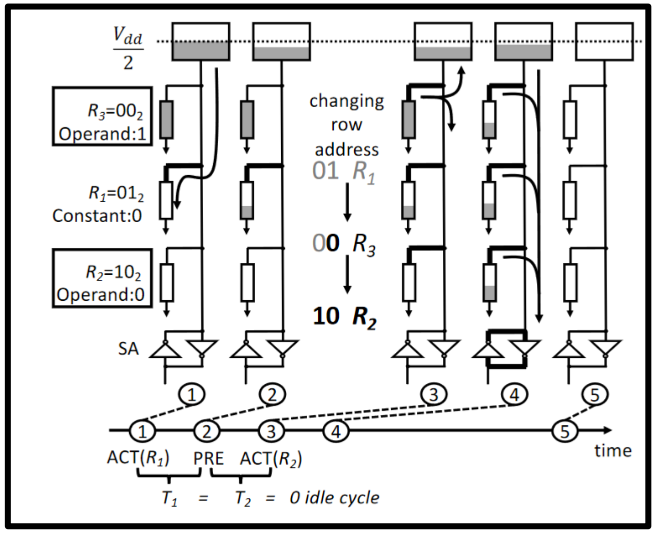
ComputeDRAM: In-Memory Compute Using Off-the-Shelf DRAMs

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

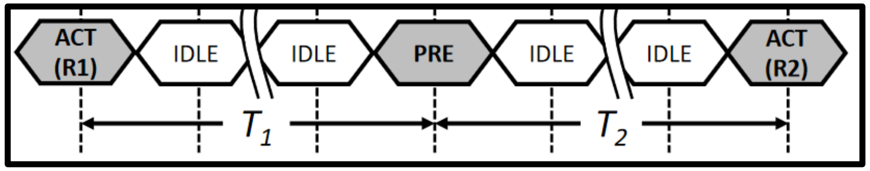
Georgios Tziantzioulis
georgios.tziantzioulis@princeton.edu
Department of Electrical Engineering
Princeton University

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

Majority Function



Row-copy/Majority Characterization



32 DDR3 Modules
~256 DRAM Chips