

SAFARI

PiDRAM

A Framework for End-to-end Integration of Processing-using-memory

P&S Processing-in-Memory Tutorial

09.11.21

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

Executive Summary

Motivation: Recent works propose **in-DRAM computation primitives** with great potential to improve performance and energy consumption of computing systems

Problem: These works are developed in **limited environments** (e.g., simulators, characterization platforms) where many parts of the system are ignored

- The challenges in integrating these primitives into a system **cannot be fully explored** in these environments

Goal: Develop a **flexible** platform to explore **end-to-end** implementations of current and future processing-in-memory (PuM) techniques

Key idea: To build an **FPGA-based infrastructure** that supports **in-DRAM operations** and has **system support**

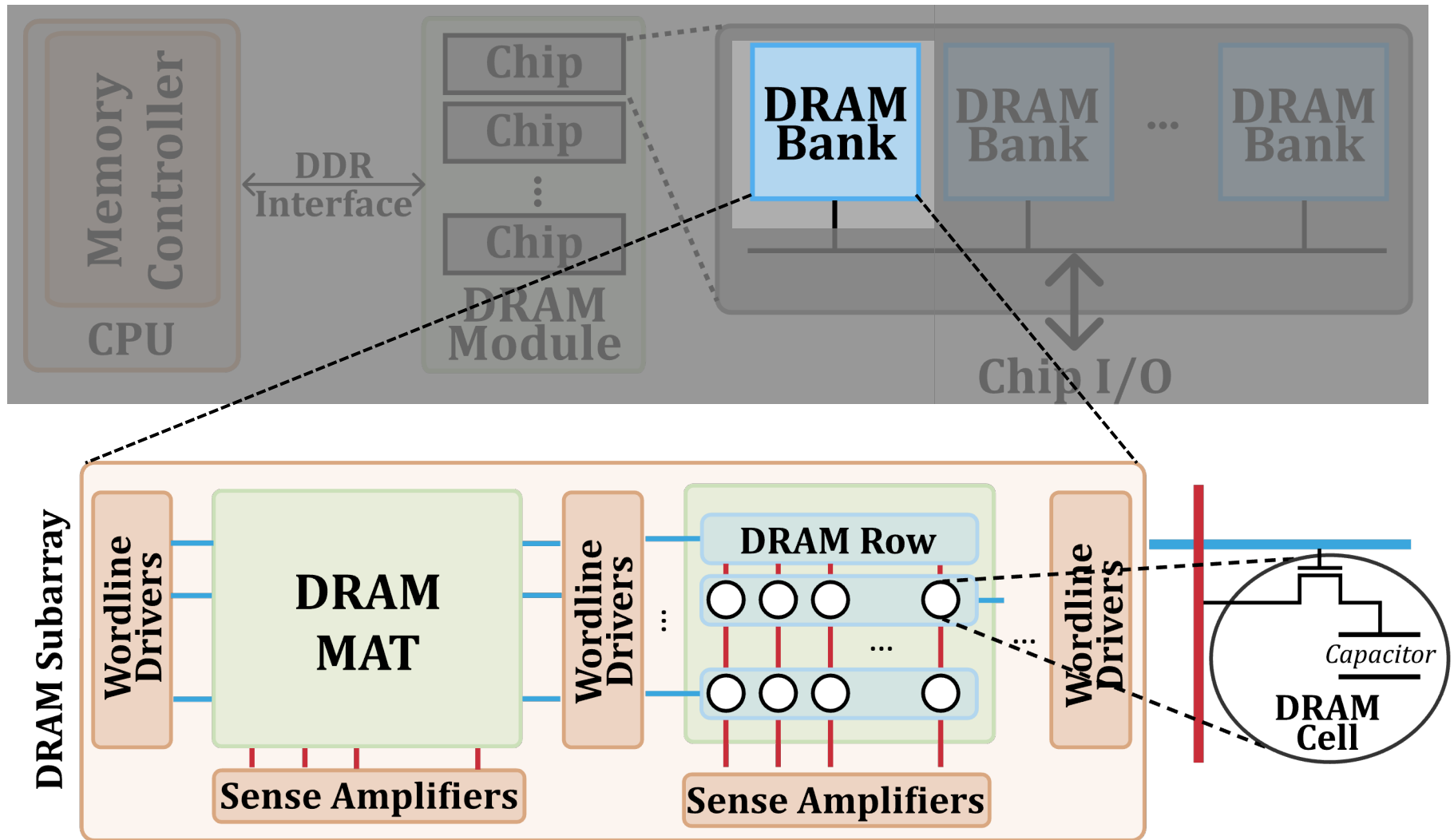
Outline

- Background
 - DRAM Organization
 - Processing-using-Memory
 - Rocket Chip SoC Generator
- Overview of PiDRAM
 - Hardware & Software Components
 - Prototype
- Case Study #1 - RowClone
 - Challenges
 - Allocation Mechanism
 - Memory Coherence
 - Evaluation
- Installing and Using PiDRAM

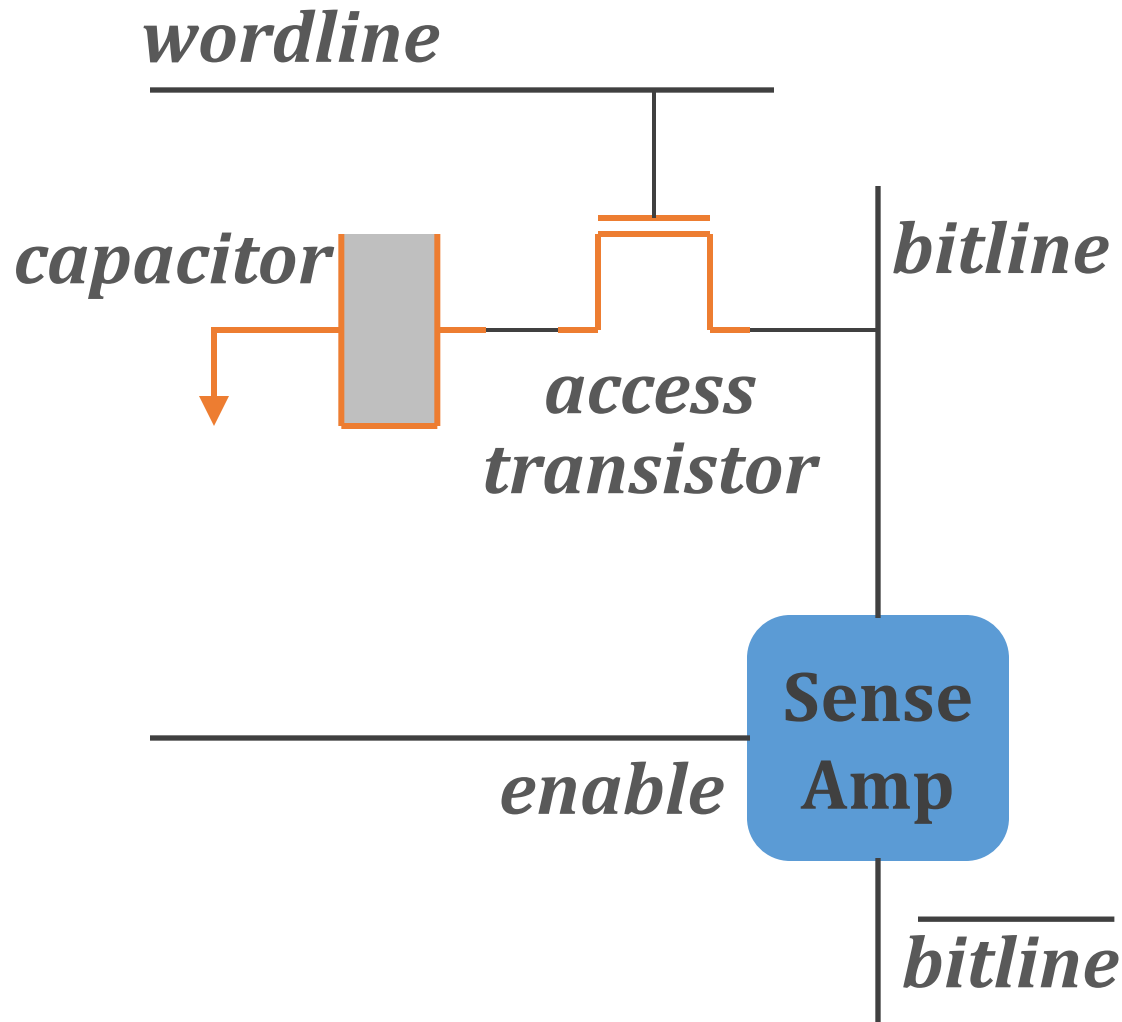
Outline

- **Background**
 - DRAM Organization
 - Processing-using-Memory
 - Rocket Chip SoC Generator
- **Overview of PiDRAM**
 - Hardware & Software Components
 - Prototype
- **Case Study #1 - RowClone**
 - Challenges
 - Allocation Mechanism
 - Memory Coherence
 - Evaluation
- **Installing and Using PiDRAM**

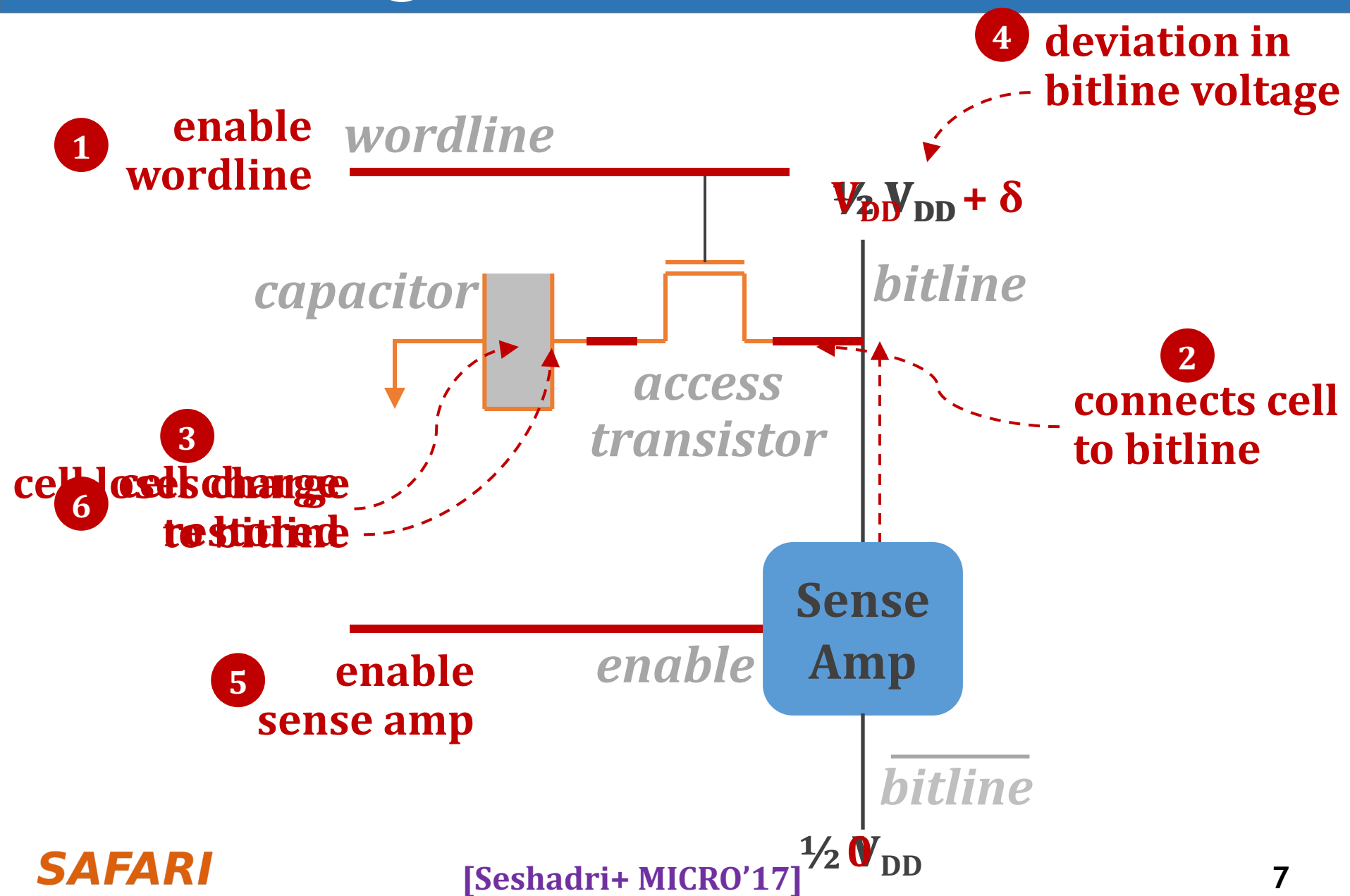
DRAM Organization



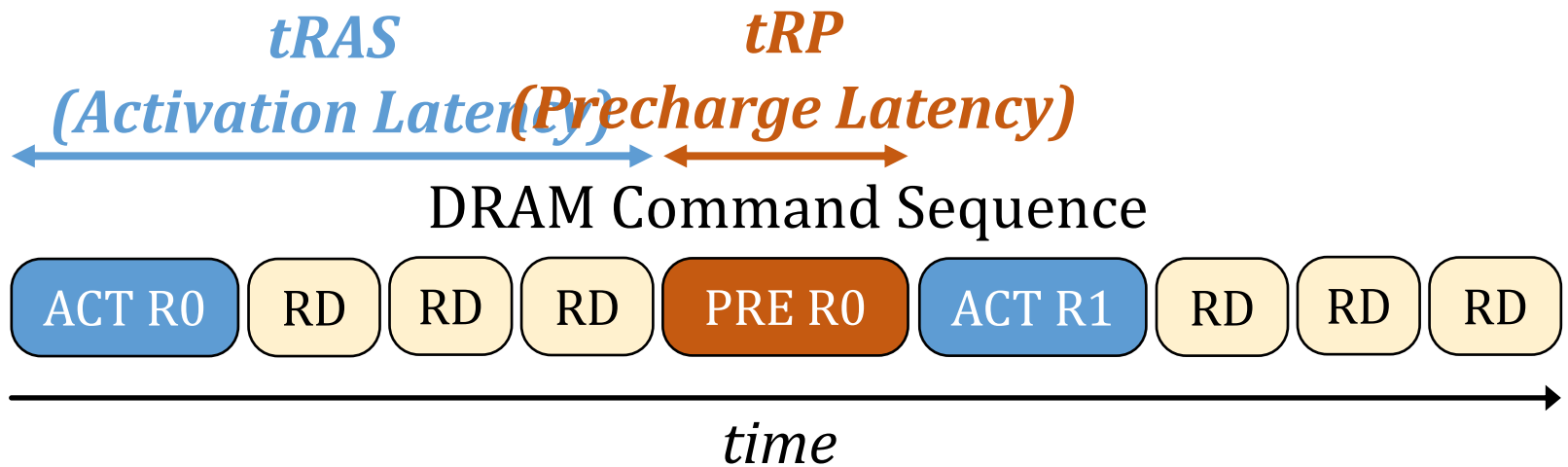
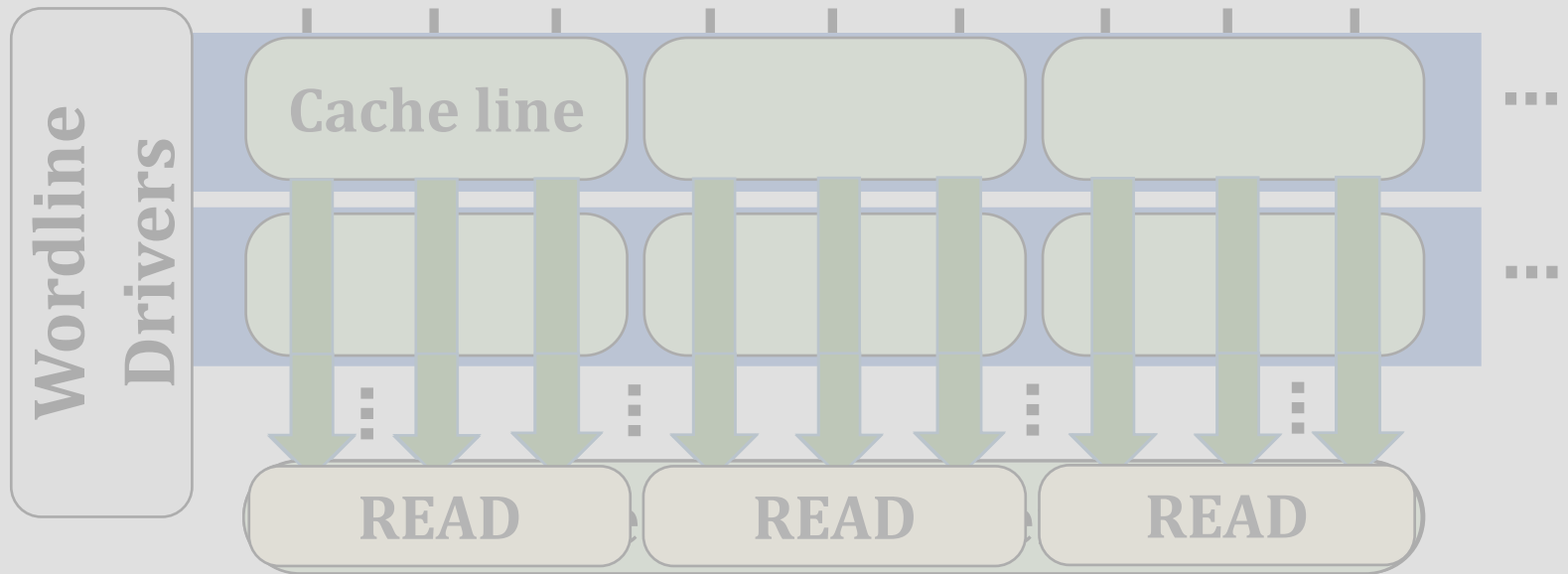
Accessing a DRAM Cell



Accessing a DRAM Cell



DRAM Operation



Outline

- **Background**
 - DRAM Organization
 - **Processing-using-Memory**
 - Rocket Chip SoC Generator
- **Overview of PiDRAM**
 - Hardware & Software Components
 - Prototype
- **Case Study #1 - RowClone**
 - Challenges
 - Allocation Mechanism
 - Memory Coherence
 - Evaluation
- **Installing and Using PiDRAM**

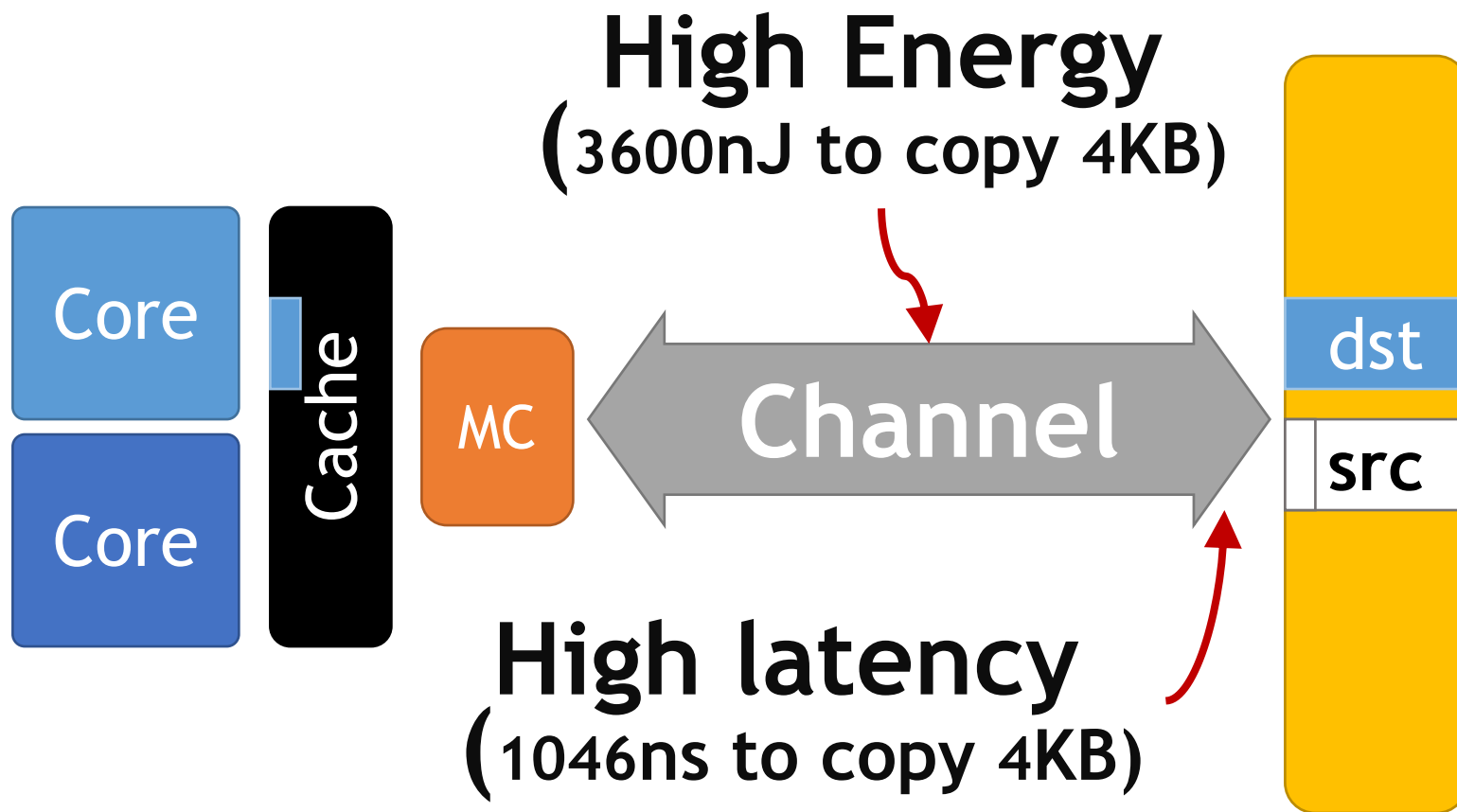
Processing Using Memory (PuM)

- Take advantage of operational principles of memory to perform **bulk data movement and computation in memory**
 - Can **exploit internal connectivity** to move data
 - Can **exploit analog computation capability**
- Examples: RowClone, In-DRAM AND/OR, D-RaNGe, ...
 - [RowClone: Fast and Efficient In-DRAM Copy and Initialization of Bulk Data \(Seshadri et al., MICRO 2013\)](#)
 - ["Ambit: In-Memory Accelerator for Bulk Bitwise Operations Using Commodity DRAM Technology" \(Seshadri et al., MICRO 2017\)](#)
 - ["D-RaNGe: Using Commodity DRAM Devices to Generate True Random Numbers with Low Latency and High Throughput" \(Kim et al., HPCA 2019\)](#)
 - ...

Processing Using Memory (PuM)

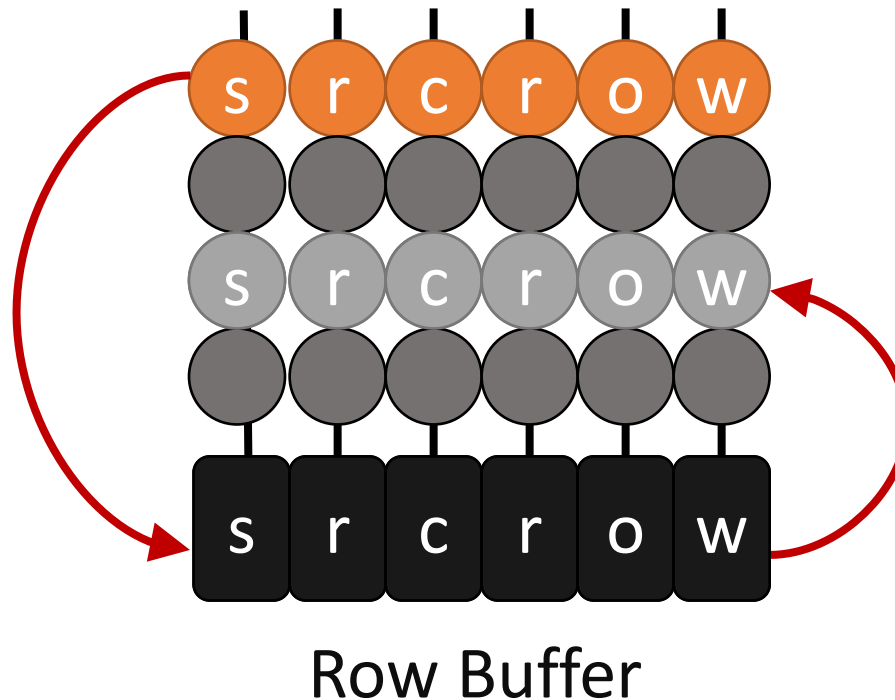
- Take advantage of operational principles of memory to perform **bulk data movement and computation in memory**
 - Can **exploit internal connectivity** to move data
 - Can **exploit analog computation capability**
- Examples: RowClone, In-DRAM AND/OR, D-RaNGe, ...
 - **RowClone: Fast and Efficient In-DRAM Copy and Initialization of Bulk Data** (Seshadri et al., MICRO 2013)
 - **"Ambit: In-Memory Accelerator for Bulk Bitwise Operations Using Commodity DRAM Technology"** (Seshadri et al., MICRO 2017)
 - **"D-RaNGe: Using Commodity DRAM Devices to Generate True Random Numbers with Low Latency and High Throughput"** (Kim et al., HPCA 2019)
 - ...

RowClone



RowClone: Can we do it in DRAM?

RowClone-FPM: Mechanism

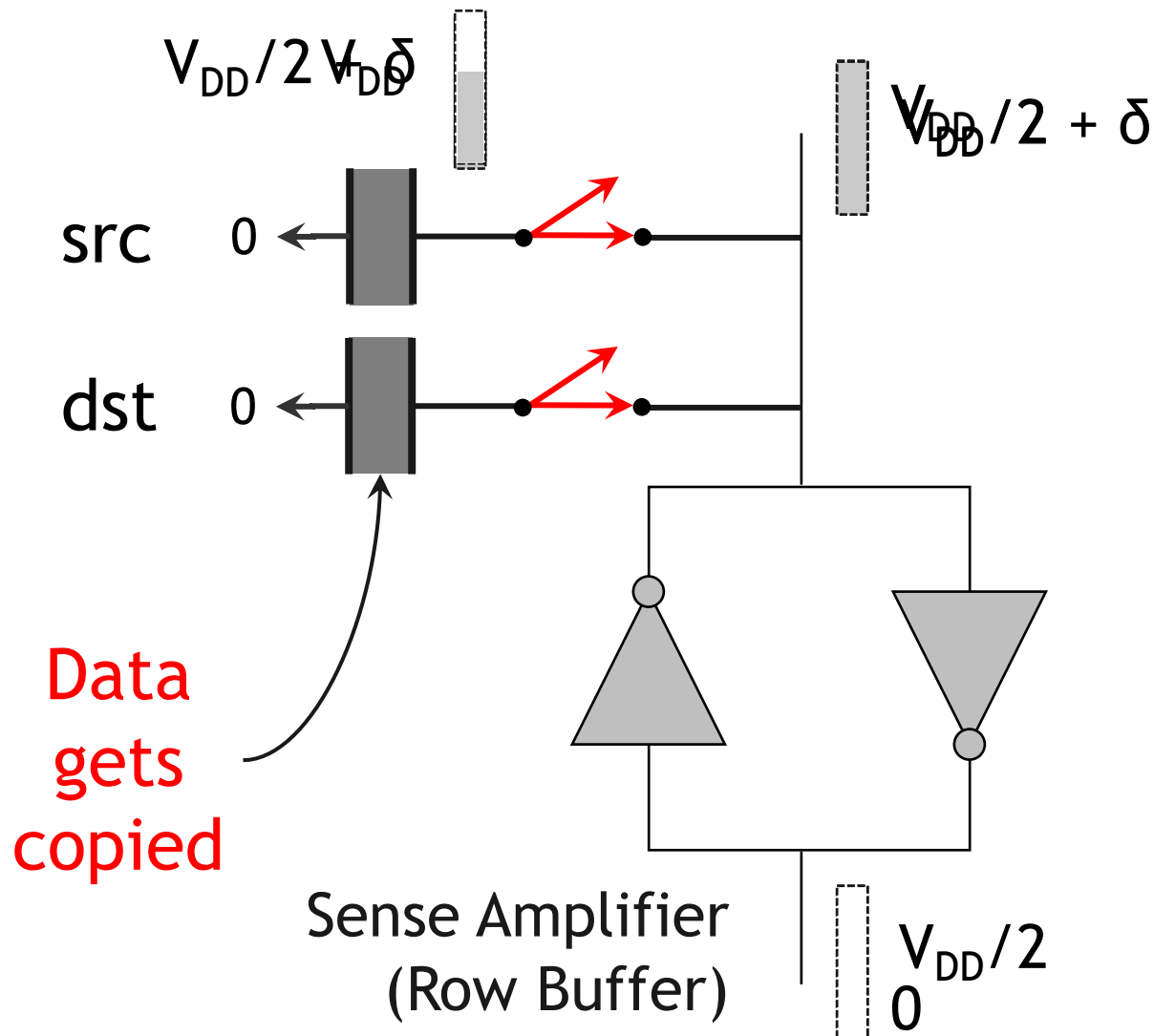


1. Source row to row buffer



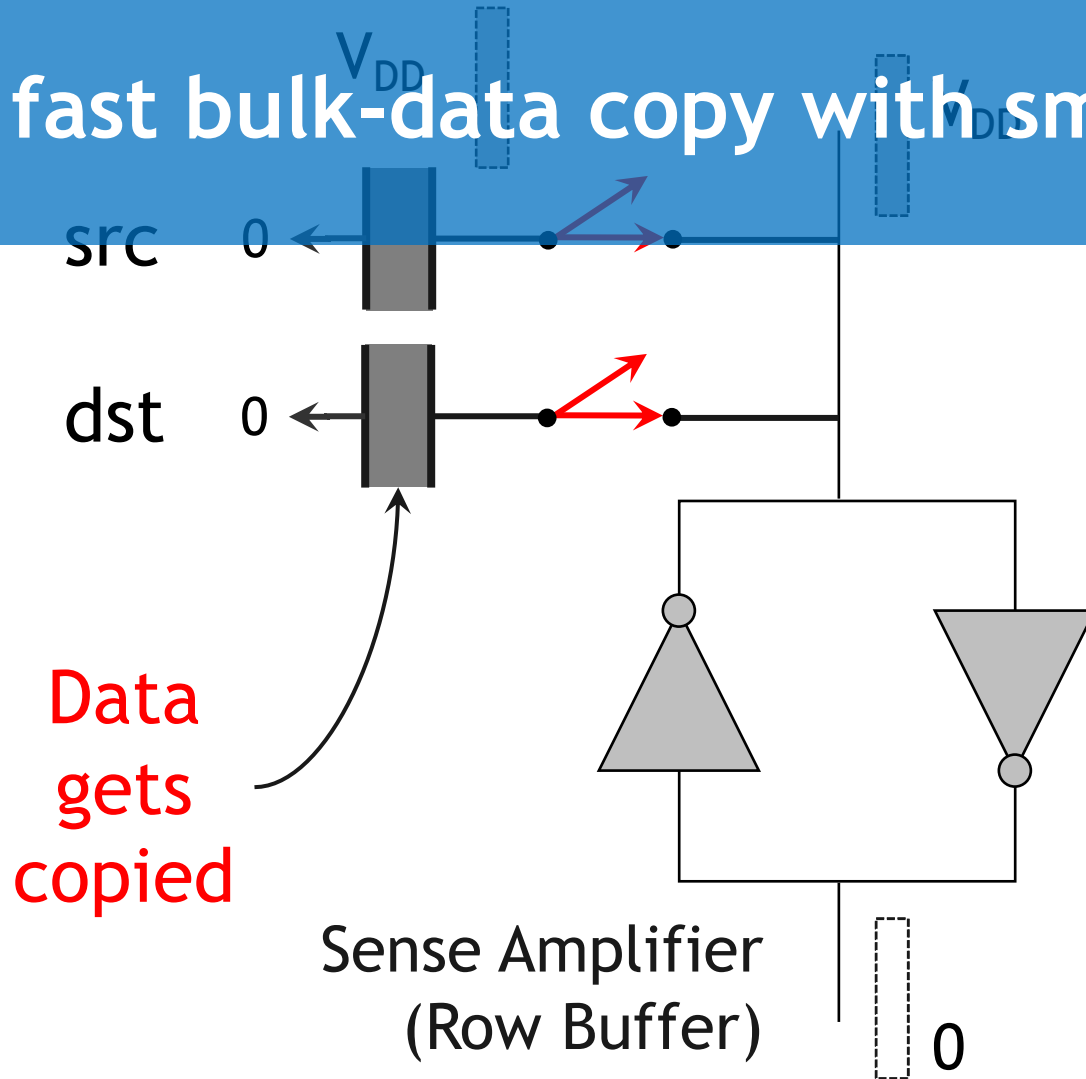
2. Row buffer to destination row

RowClone-FPM: Bitline Operation (I)



RowClone-FPM: Bitline Operation (II)

Enable fast bulk-data copy with small overhead



In-DRAM Bitwise AND/OR

- We can support in-DRAM COPY, ZERO, AND, OR, NOT, MAJ
- At low cost
- Using inherent analog computation capability of DRAM
 - Idea: activating multiple rows performs computation
- 30-60X performance and energy improvement

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

["Fast Bulk Bitwise AND and OR in DRAM"](#)

[IEEE Computer Architecture Letters](#) (CAL), April 2015.

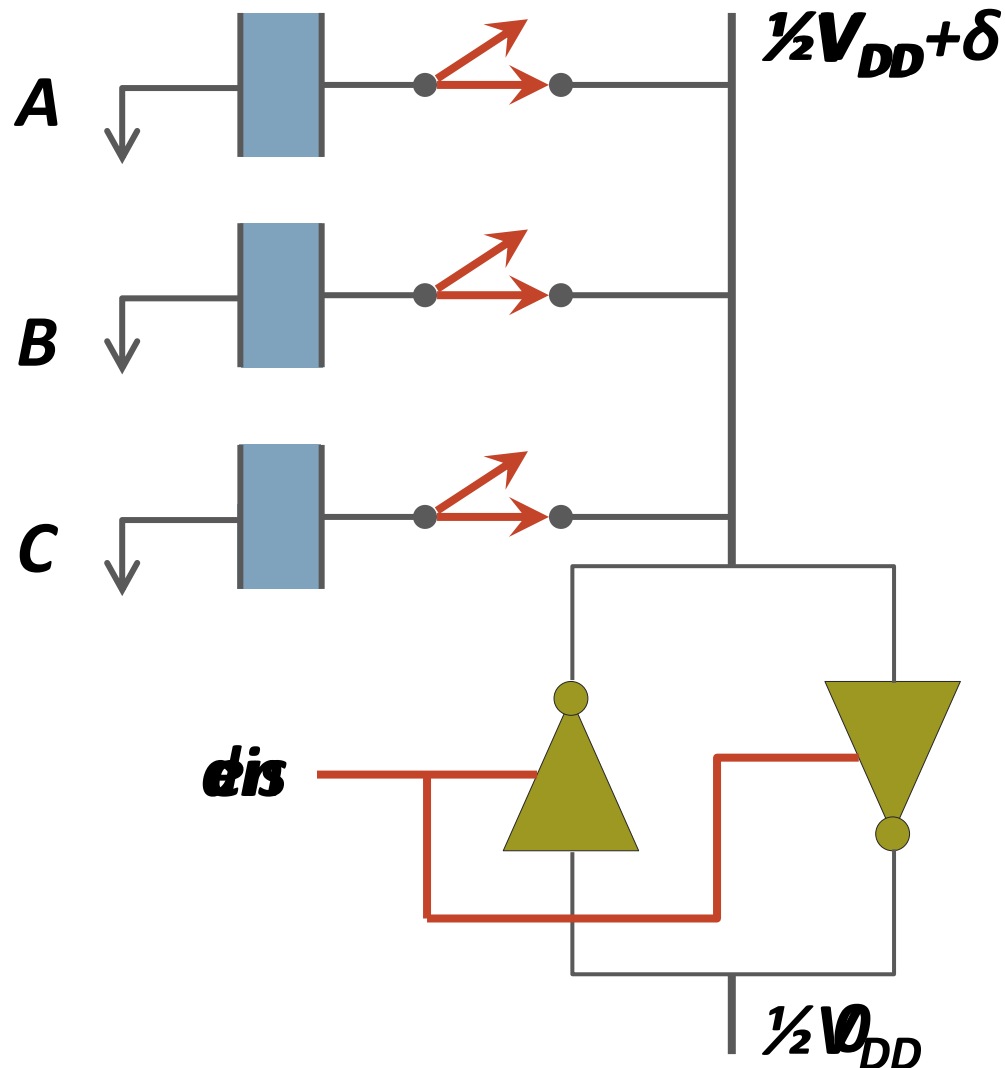
Fast Bulk Bitwise AND and OR in DRAM

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

*Carnegie Mellon University

†Intel Pittsburgh

In-DRAM AND/OR: Triple Row Activation



Final State
 $AB + BC + AC$

$C(A + B) +$
 $\sim C(AB)$

Processing-using-Memory in Real DRAM Chips

ComputeDRAM

Demonstrates RowClone and AND/OR in real chips

- **Violate** DRAM timing parameters: tRAS, tRP
 - Induce undefined behavior

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

Fei Gao

feig@princeton.edu

Department of Electrical Engineering
Princeton University

Georgios Tziantzioulis

georgios.tziantzioulis@princeton.edu

Department of Electrical Engineering
Princeton University

David Wentzlaff

wentzlaf@princeton.edu

Department of Electrical Engineering
Princeton University

RowClone & Bitwise Ops in Real DRAM Chips

MICRO-52, October 12–16, 2019, Columbus, OH, USA

Gao et al.

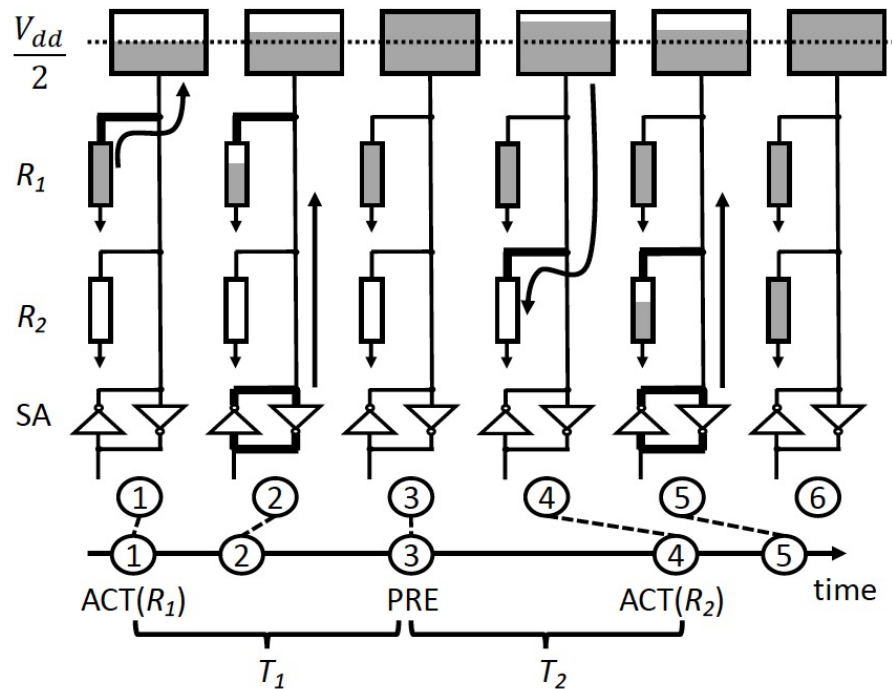


Figure 4: Timeline for a single bit of a column in a row copy operation. The data in R_1 is loaded to the bit-line, and overwrites R_2 .

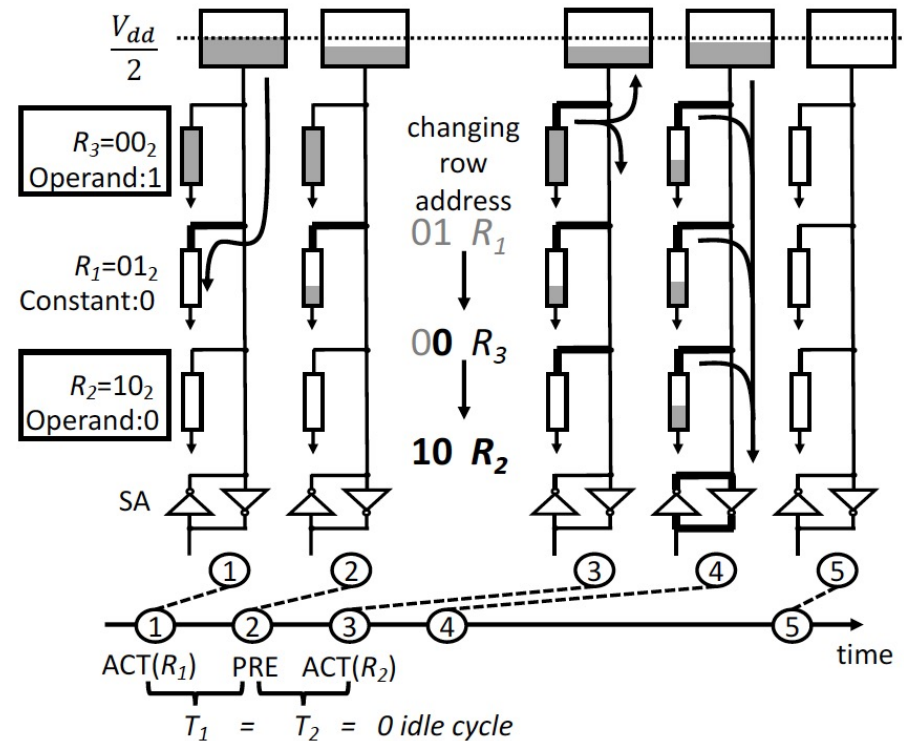
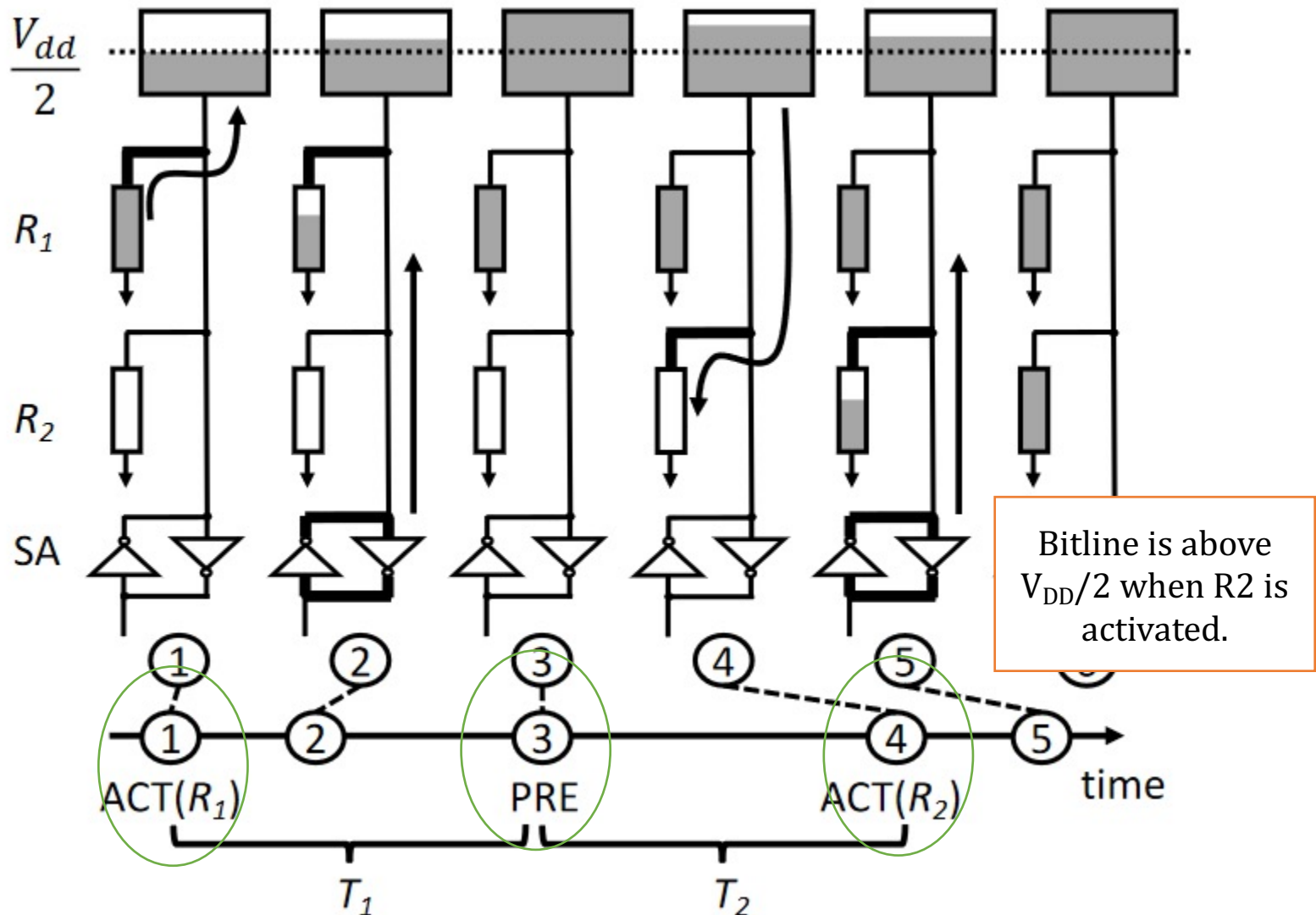
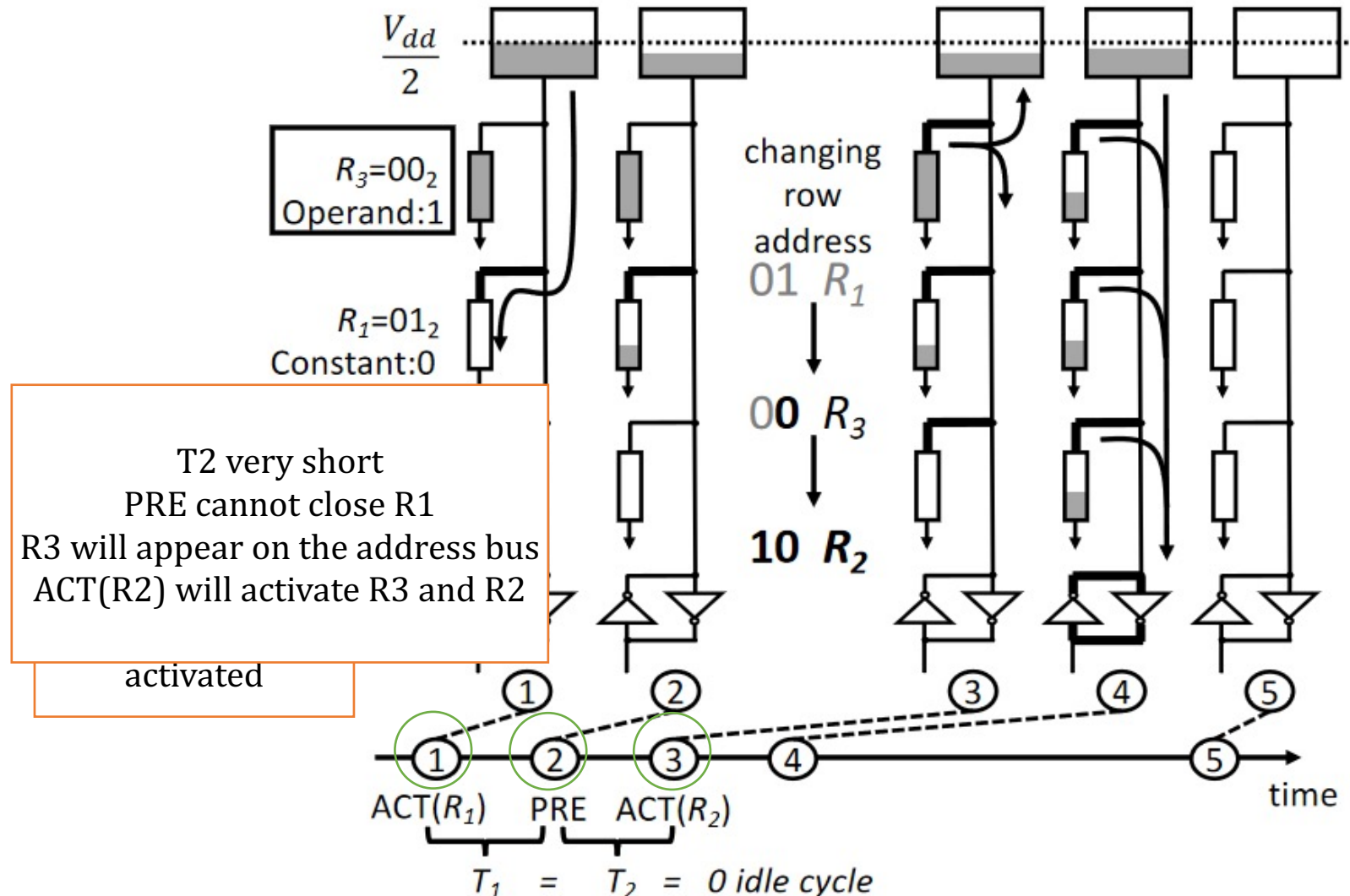


Figure 5: Logical AND in ComputeDRAM. R_1 is loaded with constant zero, and R_2 and R_3 store operands (0 and 1). The result ($0 = 1 \wedge 0$) is finally set in all three rows.

Row Copy in ComputeDRAM

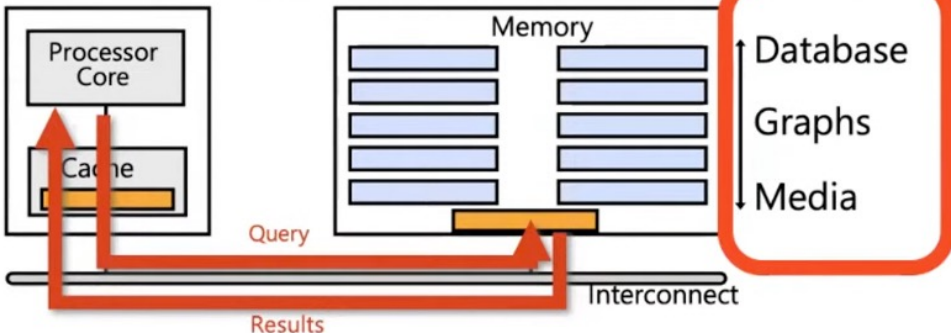


Bitwise AND in ComputeDRAM



Lecture on PuM

Goal: Processing Inside Memory



- Many questions ... How do we design the:
 - ❑ compute-capable memory & controllers?
 - ❑ processor chip and in-memory units?
 - ❑ software and hardware interfaces?
 - ❑ system software, compilers, languages?
 - ❑ algorithms and theoretical foundations?

ETH ZURICH D-ITET

Computer Architecture - Lecture 6: Processing using Memory (Fall 2021)

796 views • Streamed live on 15 Oct 2021 • Computer Architectur...

SHOW MORE

Onur Mutlu Lectures
19.8K subscribers

SUBSCRIBED

SHOW CHAT REPLAY

All From your search Computer Architecture

All of Google's Data Center Workbooks (2013):

Computer Architecture - Lecture 3: Memory Systems: Trends,...

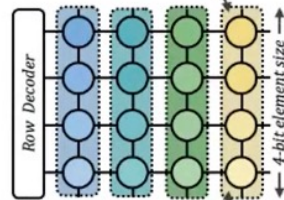
SIMDRAM Live Seminar

SIMDRAM: PuM Substrate

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

(1) Vertical data layout

most significant bit (MSB)



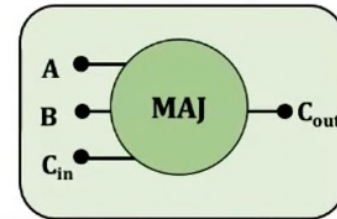
least significant bit (LSB)

Pros compared to the conventional horizontal layout:

- Implicit shift operation
- Massive parallelism

(2) Majority-based computation

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



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

- Higher performance
- Higher throughput
- Lower energy consumption



Nastaran Haji...

SAFARI

SAFARI Live Seminar - Data-Centric & Data-Aware Frameworks for Fundamentally Efficient Data Handling

375 views • Streamed live on 27 Oct 2021 • Title: Data-Centric and...

SHOW MORE

21 0 Share Save ...



Onur Mutlu Lectures
19.8K subscribers

SUBSCRIBED



SHOW CHAT REPLAY

All

From your search

Computers

Comp



Onur Mutlu - Future Computing

Outline

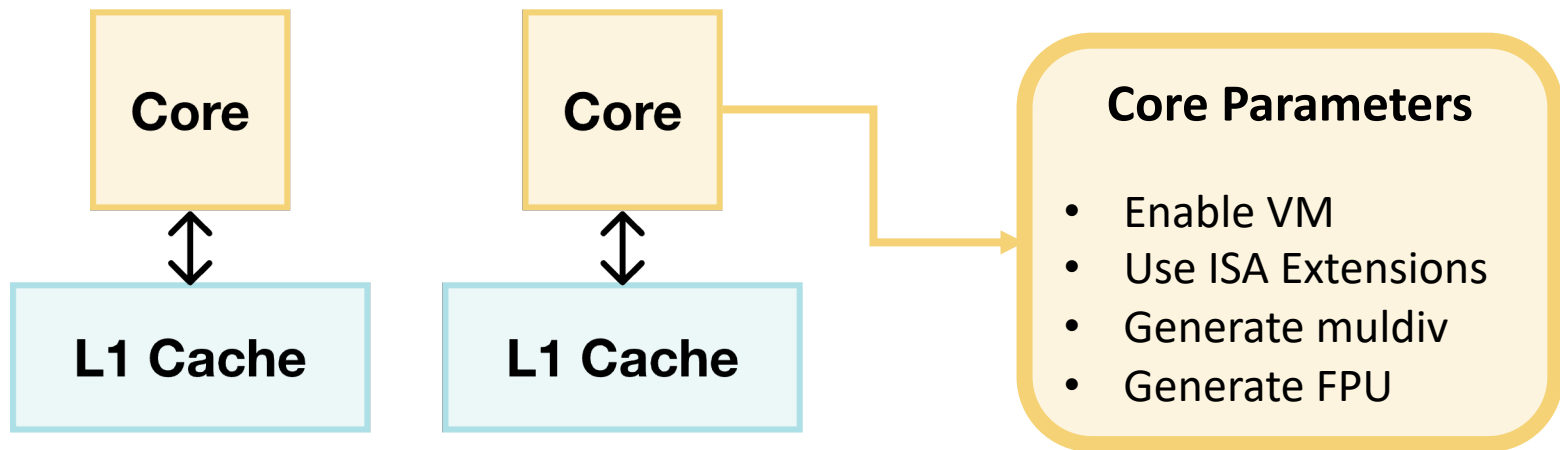
- **Background**
 - DRAM Organization
 - Processing-using-Memory
 - **Rocket Chip SoC Generator**
- **Overview of PiDRAM**
 - Hardware & Software Components
 - Prototype
- **Case Study #1 - RowClone**
 - Challenges
 - Allocation Mechanism
 - Memory Coherence
 - Evaluation
- **Installing and Using PiDRAM**

Rocket Chip

Open-source SoC design *generator*

Composed of many SoC component *generators*

- Generator: Chisel/Scala code that builds hardware



Outputs synthesizable Verilog RTL

Outline

- Background
 - DRAM Organization
 - Processing-using-Memory
 - Rocket Chip SoC Generator
- **Overview of PiDRAM**
 - Hardware & Software Components
 - Prototype
- Case Study #1 - RowClone
 - Challenges
 - Allocation Mechanism
 - Memory Coherence
 - Evaluation
- Installing and Using PiDRAM

PiDRAM

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

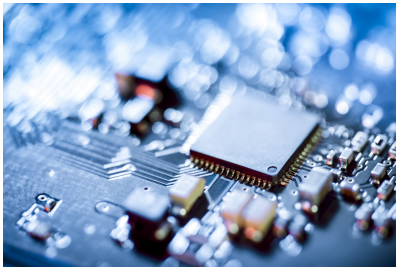
- Enable rapid integration via key components

PiDRAM

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

- Enable rapid integration via key components

Hardware



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

Software



- 1 Extensible Software Library
- 2 Custom Supervisor Software

PiDRAM: Memory Controller (I)

Current real PuM techniques **require** issuing DRAM command sequences with **violated** timings

- Extensible memory controller facilitates implementation of such DRAM command sequences through modular design
- New command sequences require only designing new state machines

PiDRAM: Memory Controller (II)

Memory controller employs three submodules to further ease developer effort

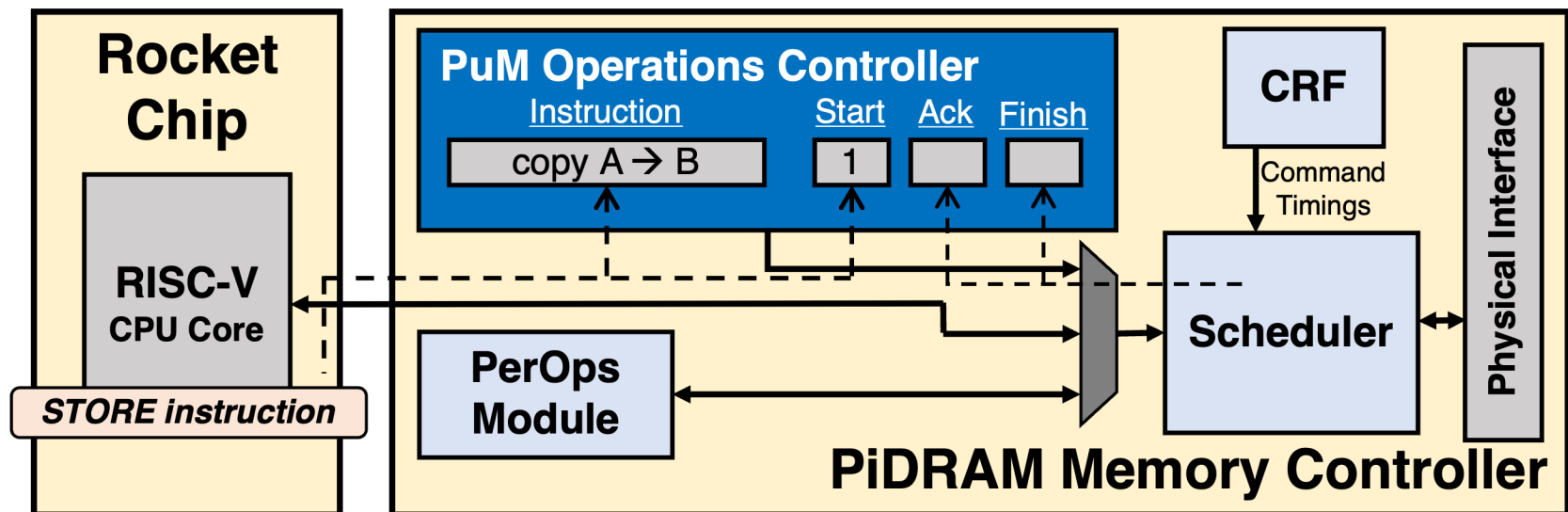
- ➊ **Periodic Operations Module (PerOps Module):**
DRAM periodic refresh and bus maintenance operations
- ➋ **Scheduler:** DRAM RD/WR operations, open-bank policy
- ➌ **Configuration Register File (CRF):**
Store timing information, accessed using LD/ST instructions from the CPU

PiDRAM: PuM Controller (I)

PuM Operations Controller (POC)

Provide ISA-transparent control for PuM operations

- Connected as a memory-mapped module
 - Hierarchically, resides within the memory controller
- Simple Interface: Offload 128-bit instructions



PiDRAM: PuM Controller (II)

Currently implements five instructions

RowClone

Reserved	Copy	Dst. Row	Src. Row
127	69 68 64 63	32 31	0

Activation Failure

Reserved	RLRD	Unused	Cache Block
127	69 68 64 63	32 31	0

Read Random Number (RN)

Reserved	RRN	Unused
127	69 68 64 63	0

Read RN Buffer Size

Reserved	RRNS	Unused
127	69 68 64 63	0

Write to configuration regs.

Reserved	WCR	Address	Data
127	69 68 64 63	32 31	0

Reserved bits for other commands

- Instruction size is configurable

PiDRAM: Software Library

Pumolib: Expose PuM operations to the user while abstracting the hardware implementation details

```
static inline void copy_row(char *source, char *target)
{
    volatile uint64_t *ptr    = (uint64_t*) IMOC_INST_UPPER;
    uint64_t imo_op           = 0x1;
    uint32_t source_row_addr  = (uint32_t) source;
    uint64_t target_row_addr  = (uint32_t) target;
    uint64_t inst_lower       = source_row_addr |
                                (target_row_addr << 32);
    uint64_t inst_upper       = imo_op << (IMO_OP_OFS);

    *ptr = inst_upper;
    *(ptr+1) = inst_lower;
    *(ptr+2) = (uint64_t) 0x1;
    while(*(ptr+2) != 0x2);
}
```

} Prepare PuM instruction

} Offload the instruction
and block until ACK'd

PiDRAM: Custom Supervisor SW

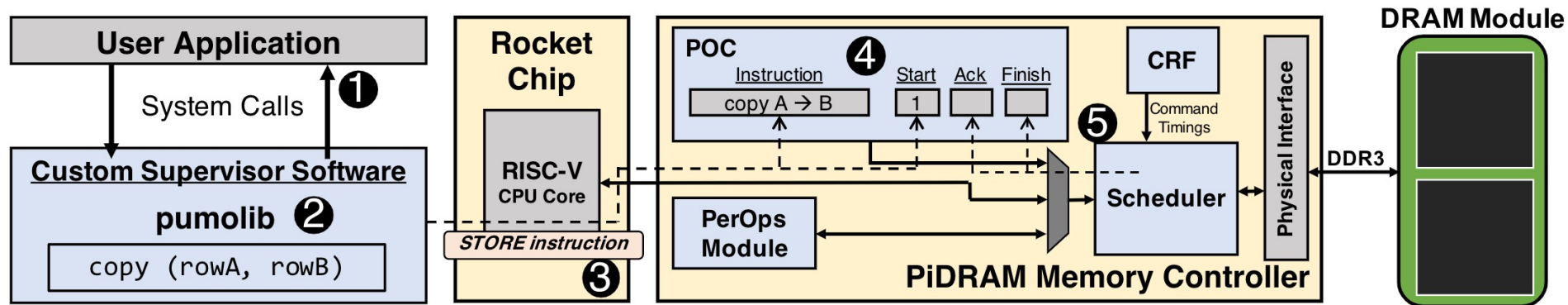
Simple, easy-to-hack OS to integrate PuM techniques end-to-end:

- Virtual memory management
- Memory allocation

OS based on RISC-V proxy kernel

Future work: Integrate pumolib into Linux

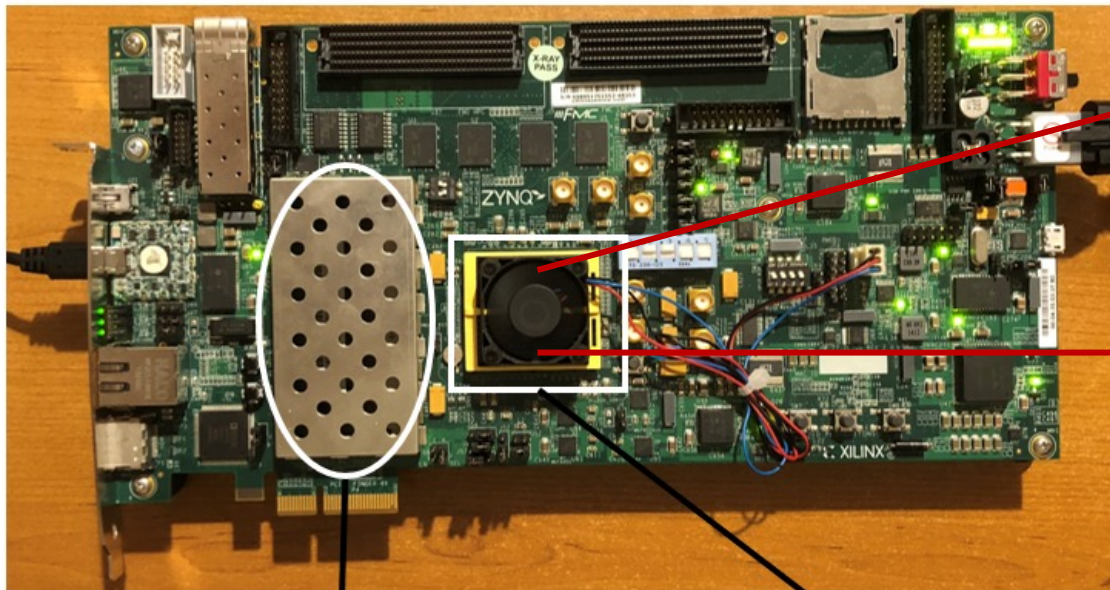
PiDRAM Workflow



- 1- User application interfaces with the OS via system calls
- 2- OS uses PuM Operations Library (pumolib) to convey operation related information to the hardware using
- 3- STORE instructions that target the memory mapped registers of the PuM Operations Controller (POC)
- 4- POC oversees the execution of a PuM operation (e.g., RowClone, bulk bitwise operations)
- 5- Scheduler arbitrates between regular (load, store) and PuM operations and issues DRAM commands with custom timings

PiDRAM FPGA Prototype

Xilinx ZC706



Single core RISC-V CPU @ 50MHz
in-order, single-issue
16KB 4-way L1 D\$
4KB I\$

Compute-Enabled DIMM

RISC-V System

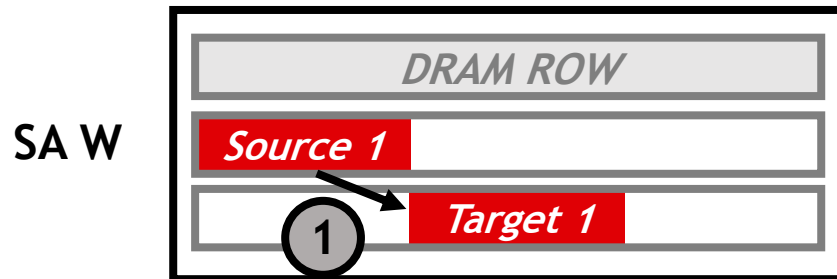
Outline

- Background
 - DRAM Organization
 - Processing-using-Memory
 - Rocket Chip SoC Generator
- Overview of PiDRAM
 - Hardware & Software Components
 - Prototype
- **Case Study #1 - RowClone**
 - **Challenges**
 - Allocation Mechanism
 - Memory Coherence
 - Evaluation
- Installing and Using PiDRAM

RowClone-FPM Challenges (I)

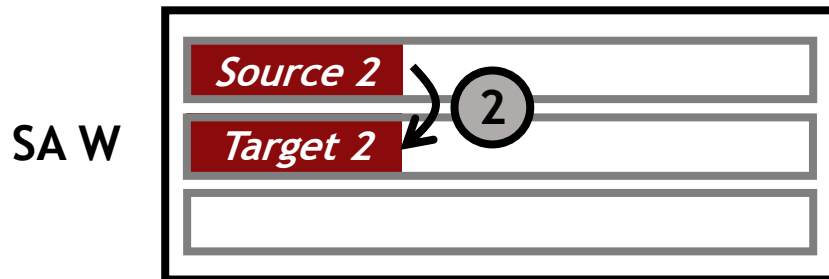
RowClone-FPM has memory mapping requirements

BANK X



1-) Alignment: Operands must be placed at the same offset to their respective DRAM rows

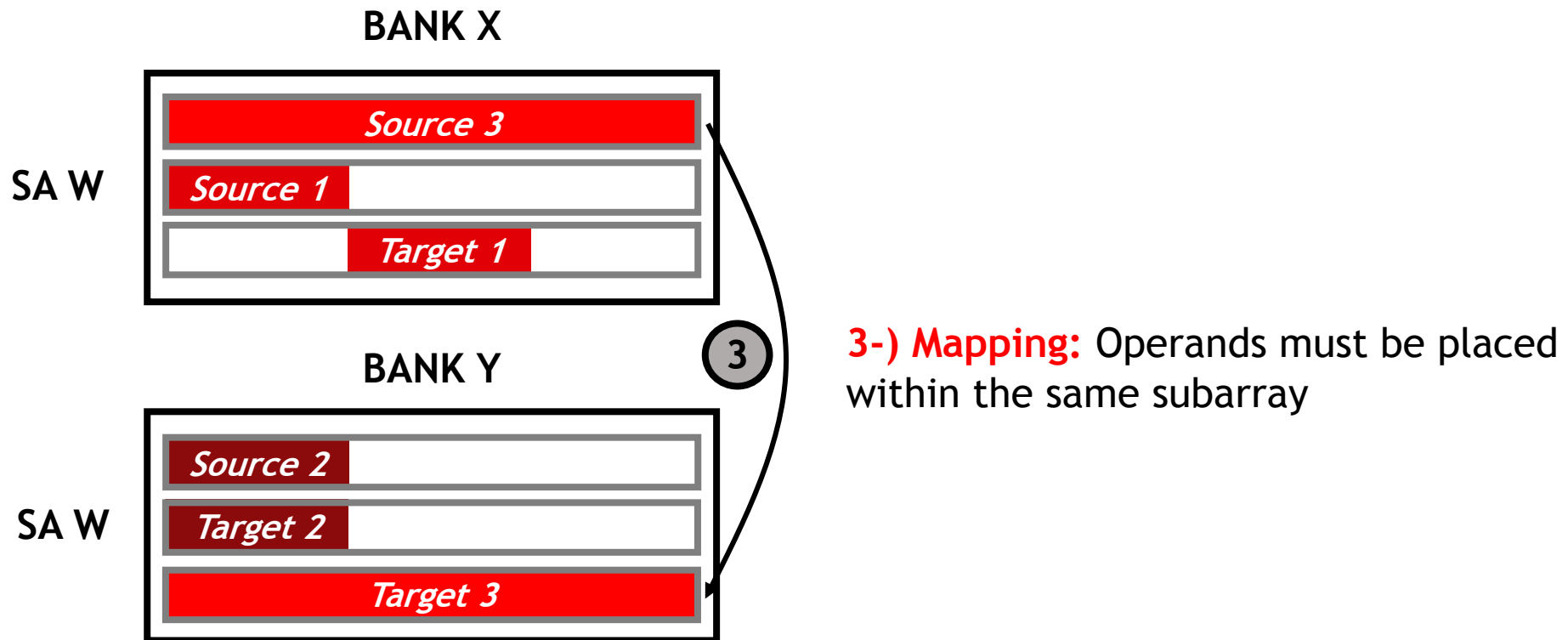
BANK Y



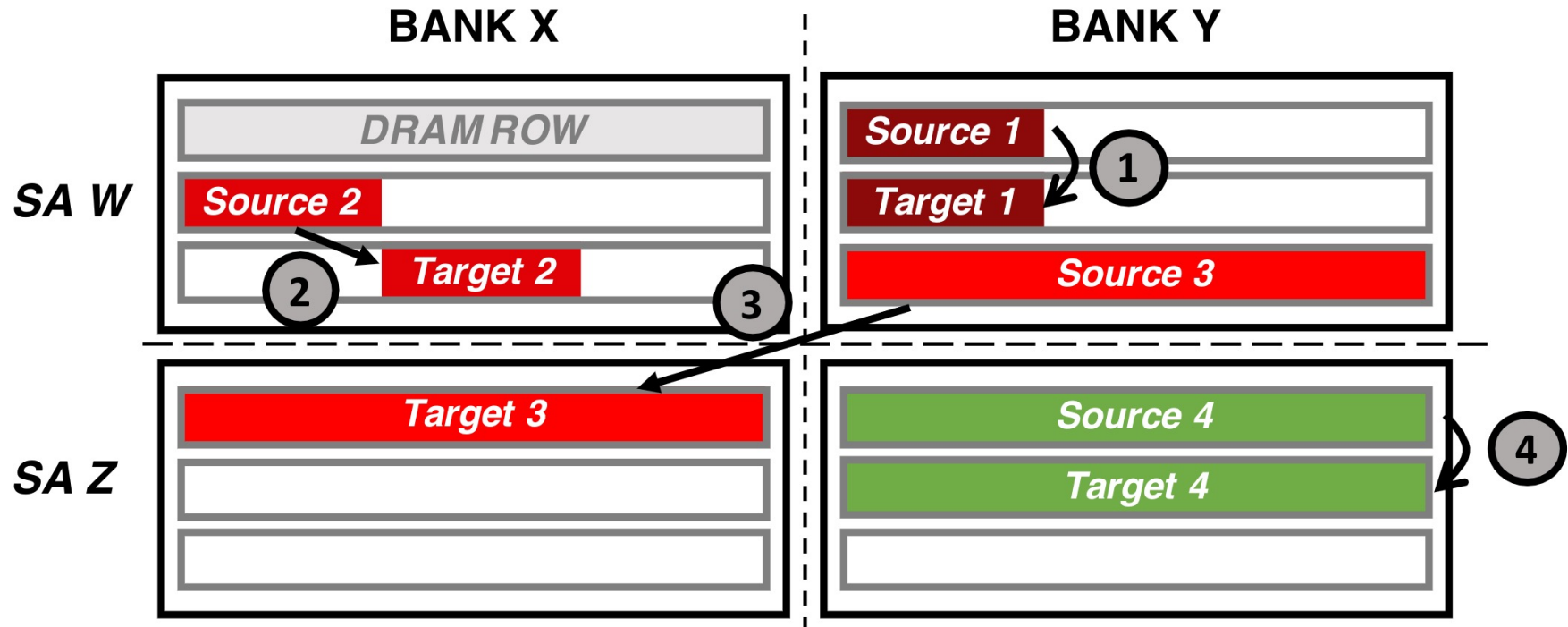
2-) Granularity: Operands must occupy whole DRAM rows

RowClone-FPM Challenges (II)

RowClone-FPM has
memory mapping requirements



RowClone-FPM Challenges (III)



(4) Satisfies all three requirements

Outline

- Background
 - DRAM Organization
 - Processing-using-Memory
 - Rocket Chip SoC Generator
- Overview of PiDRAM
 - Hardware & Software Components
 - Prototype
- **Case Study #1 - RowClone**
 - Challenges
 - **Allocation Mechanism**
 - Memory Coherence
 - Evaluation
- Installing and Using PiDRAM

Data Mapping (I)

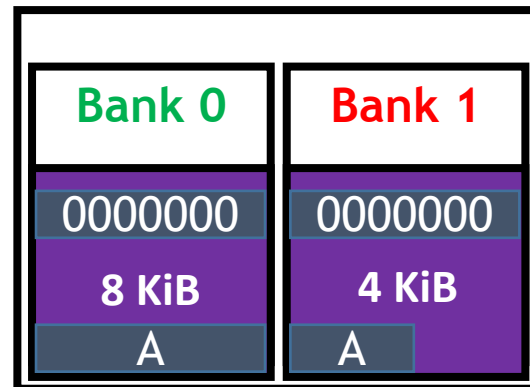
New memory allocation mechanism to satisfy these three requirements: *alloc_align()*

```
void* array = alloc_align(int size);
```

Optimize physical address allocation to *array* for *size* byte large copy operations so that RowClone can be used most effectively

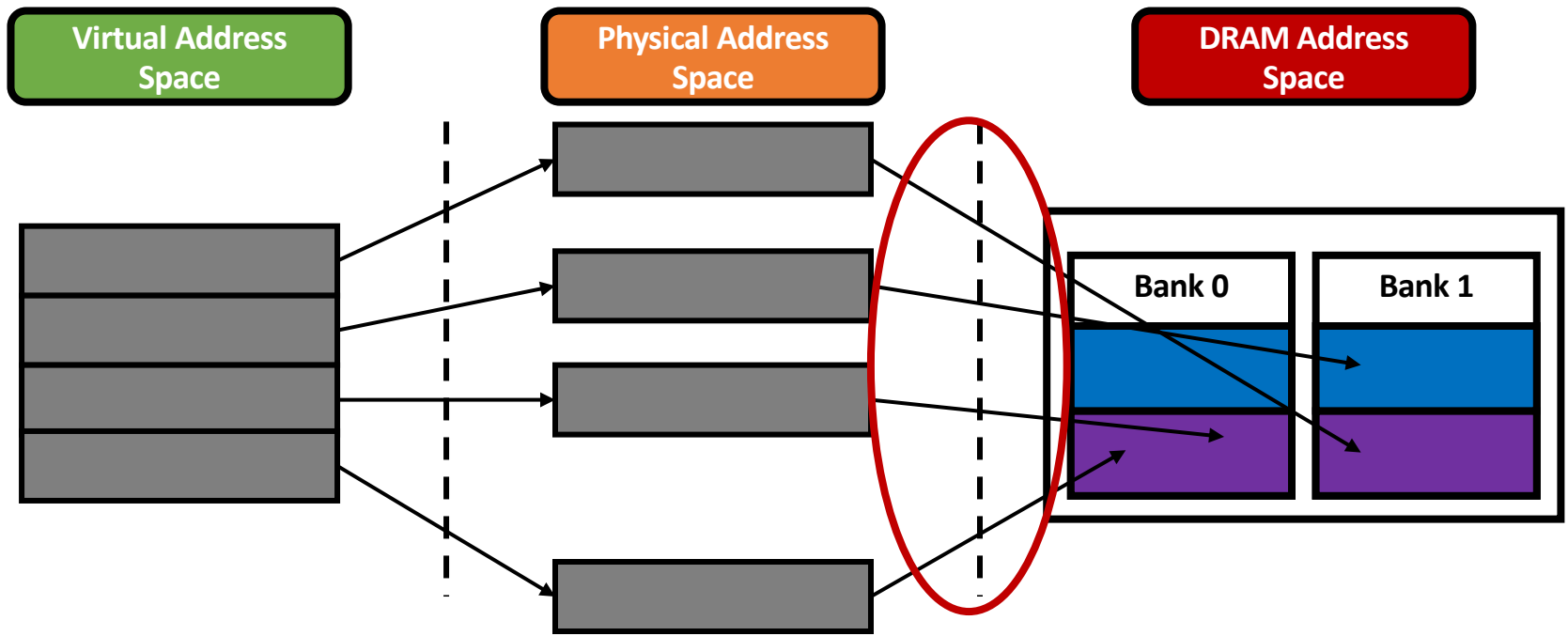
Example (Row = 8 KiB)

```
char *A =  
alloc_align(A, 4096*3);
```



We can at least
copy 8 KiB
using RowClone

Data Mapping (II)



OS has full control over VA \rightarrow PA translation

No control over PA \rightarrow DRAM address mapping

Idea: If we can control VA \rightarrow PA and we know the PA \rightarrow DRAM mappings, we can implement *alloc_align*

Alloc_align (I)

```
void* array = alloc_align(int size);
```

How to lay out an array onto DRAM?

- I. Distribute the array over multiple banks while occupying rows as fully as possible
- II. Fallback to `malloc()`; for remaining data

Assumptions:

- (i) We know the **DDR** address mapping in our system
 - Reverse-engineer: (i) Check for RowClone, (ii) do Rowhammer
- (ii) We know which **DRAM** rows are in the same subarray
 - Characterize pairs of rows for RowClone success rate

Alloc_align (II)

Other versions of alloc_align can be implemented to align multiple arrays in DRAM (for RowClone-Copy)

```
alloc_align(int size, int id);
```

id = operand RowClone identifier

Operands with the same id are placed into the same subarray

PiDRAM Address Mapping

Our Configuration:

SODIMM: MT8JTF12864HZ-1G6G1

of Rows = 16K

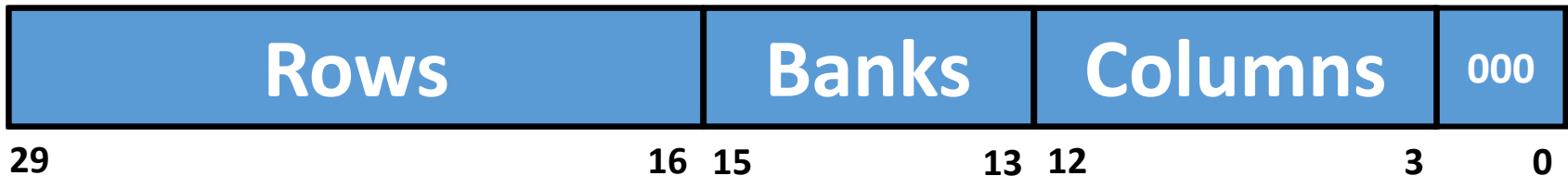
of Banks = 8

of Columns = 1K

Row Size = 8 KB

Total Size = 1 GB

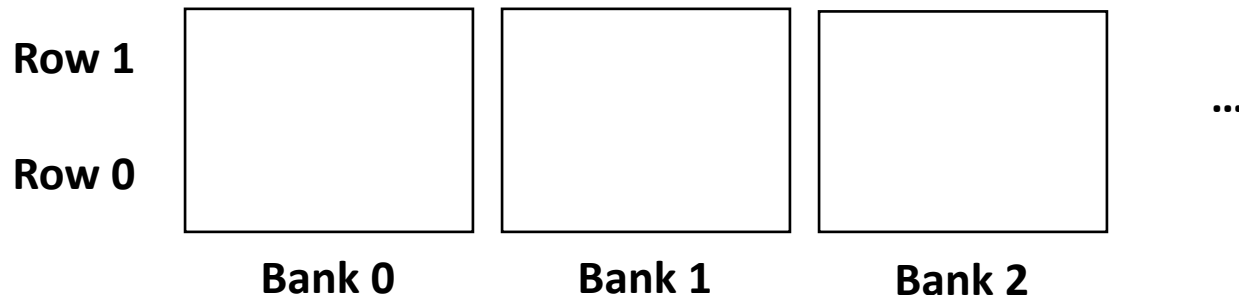
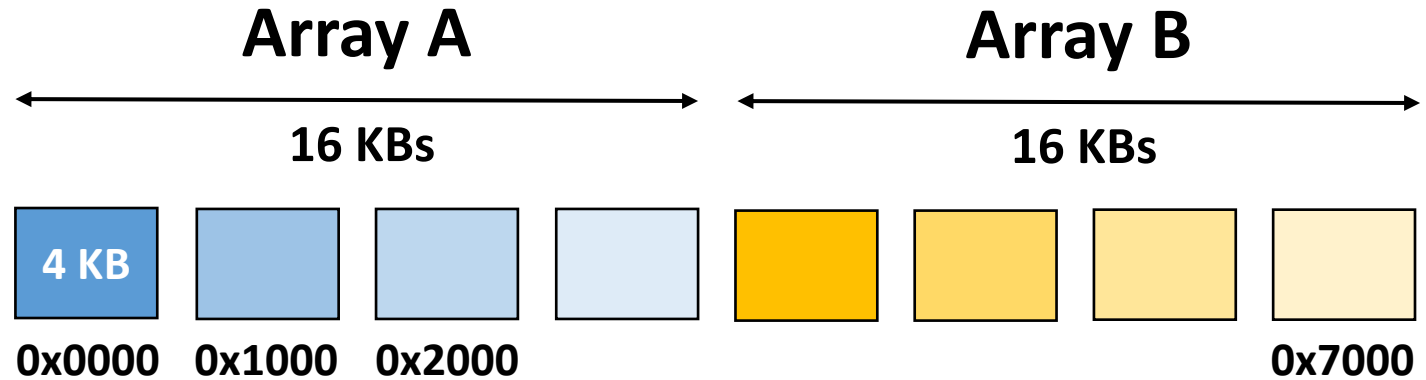
Physical to DRAM address mapping (configurable)



Alloc_align Example

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

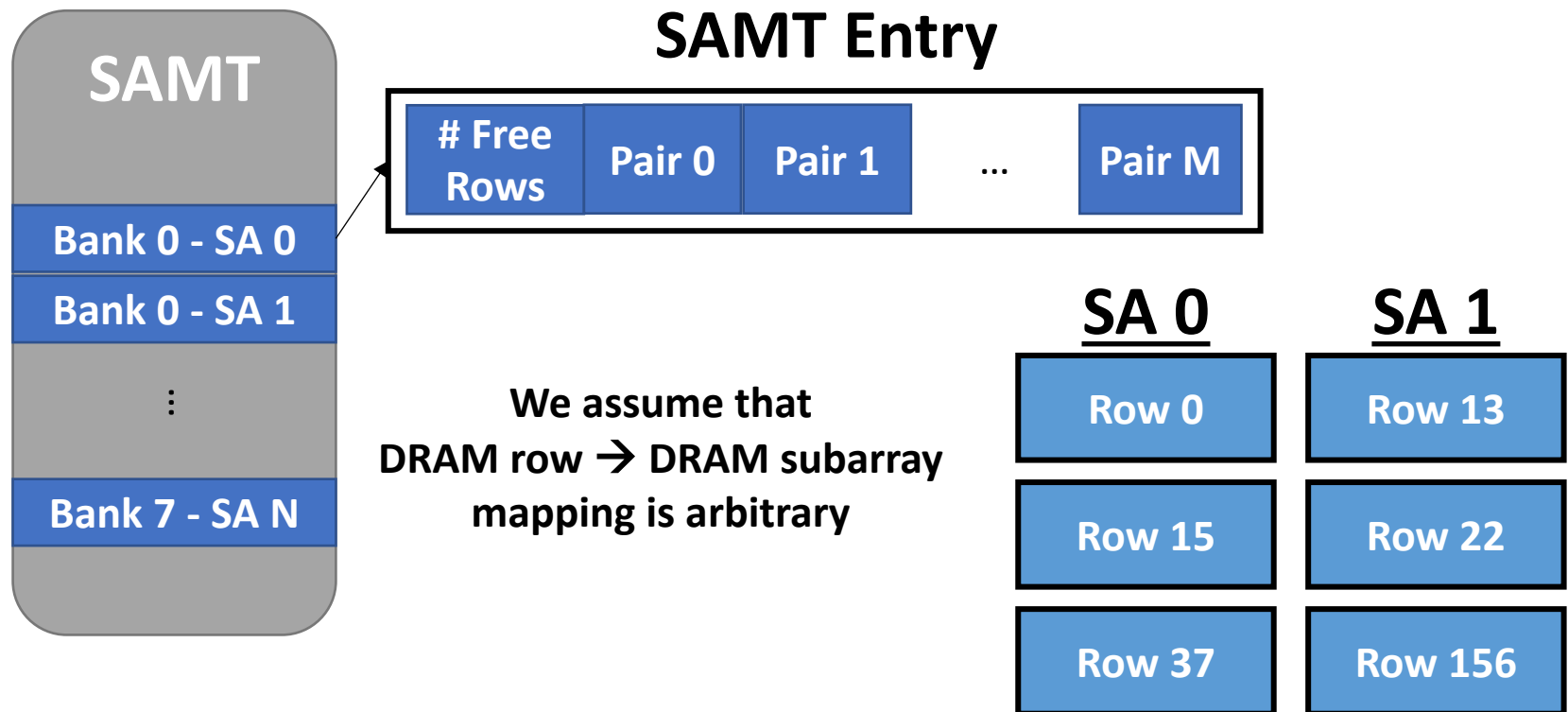
```
B = alloc_align(16*1024, 0);
```



Alloc_align - Structures (I)

SubArray Mapping Table (SAMT)

A table that collects DRAM rows that map to the same subarray under common entries as physical page address pairs



Alloc_align - Structures (II)

Reserve *one* address pair in every SAMT entry for initialization



SIM, Subarray Id Map

Mapping from page numbers to subarray indices

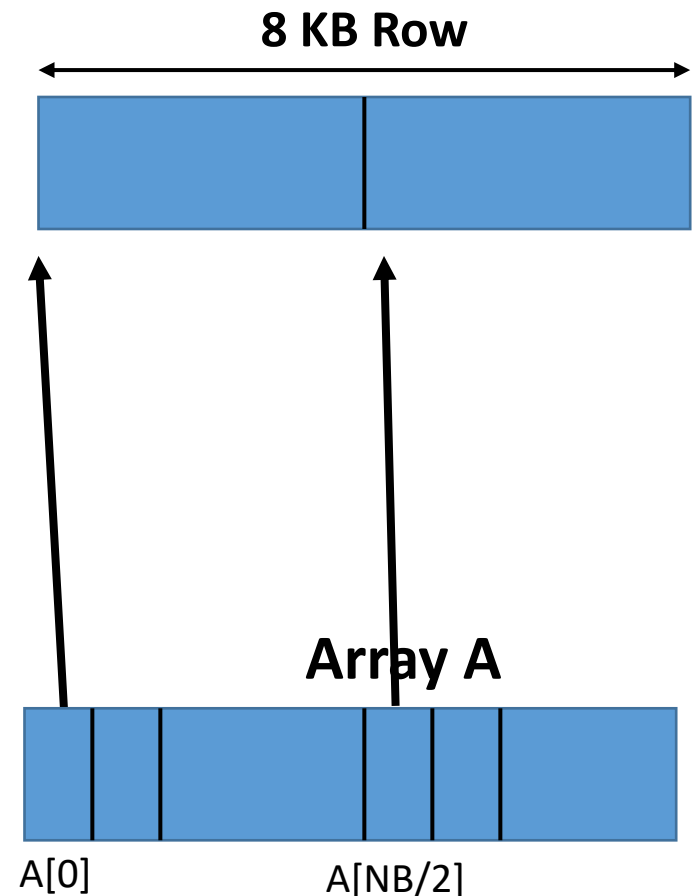
Physical Page Number	Subarray ID (SAMT idx)
PPN 0	0
PPN 1	0
...	...
PPN 256	1

Alloc_align - Structures (III)

How to use SMT

alloc_align(int size):

1. Find out how many banks to utilize
 - Spread allocation across many banks
2. Split array into 4 KB blocks
 - 4 KB == 1 page
 - NB = # of 4 KB blocks in A
3. For all *blocks* $A[i]$ where $i < NB/2$:
 1. Pick a bank.
 2. Select a SAMT entry (SAMTE) with ≥ 1 free address pairs
 3. Select one pair from SAMTE
 4. Assign physical pages in the pair to virtual pages $A[i]$ and $A[i+NB/2]$

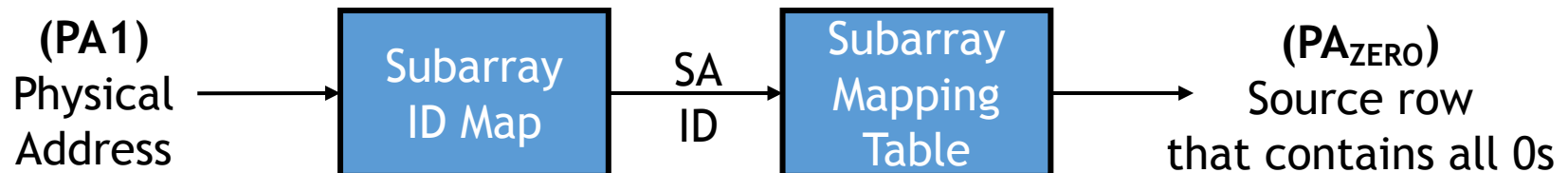


Alloc_align - Structures (IV)

RowClone-Initialize (RCI)

rci(char* A, int N) → Initialize A with N 0s.

1. Split A into 4KB blocks
2. For all *blocks* A[i] where $i < NB/2$:
 1. Translate from A[i] to PA1 → use as **destination** address
 2. Access **SIM** with PA1 to obtain its **subarray id**
 3. Access SAMT to get the address of the *all-zero* row, PA_{zero}
 4. Perform RowClone from PA_{zero} to PA1
 - This will automatically copy zeros to A[i+NB/2] too



Outline

- Background
 - DRAM Organization
 - Processing-using-Memory
 - Rocket Chip SoC Generator
- Overview of PiDRAM
 - Hardware & Software Components
 - Prototype
- **Case Study #1 - RowClone**
 - Challenges
 - Allocation Mechanism
 - **Memory Coherence**
 - Evaluation
- Installing and Using PiDRAM

Memory Coherence (I)

RowClone, AMBIT operates on data in DRAM

Up-to-date data may be in caches → coherency

Implement CLFLUSH in RISC-V rocket

Pros:

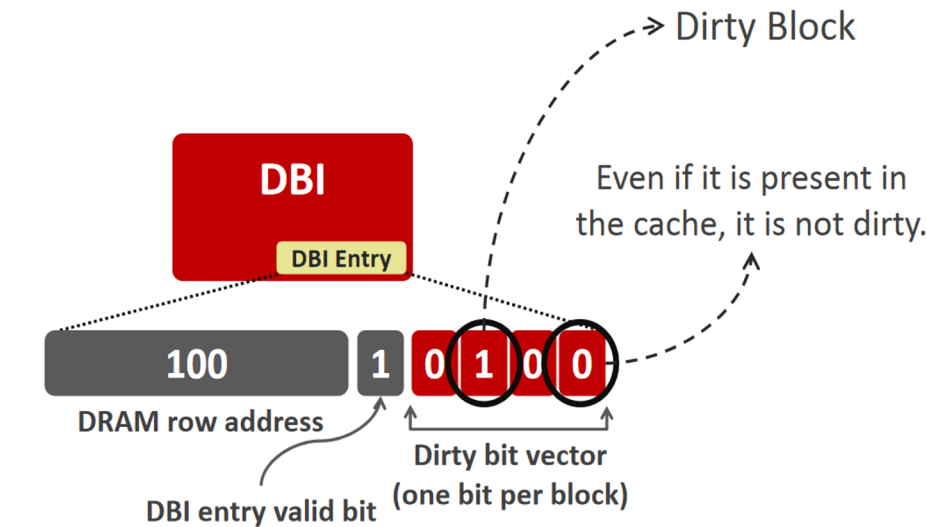
- **Realistic**, supported in contemporary architectures
- Reads and writes can hit in the cache.
Flush cache lines prior to in-DRAM operations

Cons:

- Instruction overhead: One instruction per cache block

Memory Coherence (II)

Other mechanisms that can alleviate the overheads



Vivek Seshadri, Abhishek Bhowmick, Onur Mutlu, Phillip B. Gibbons,
Michael A. Kozuch, and Todd C. Mowry,

"The Dirty-Block Index"

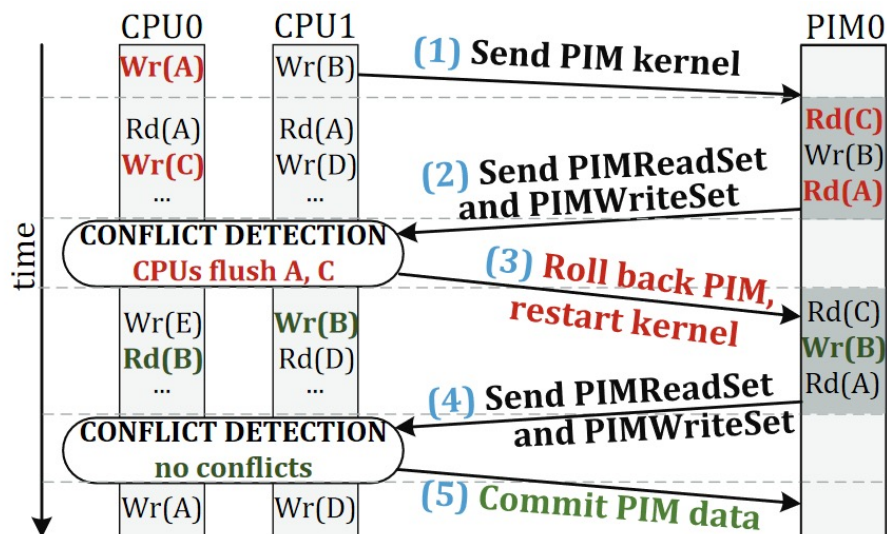
Proceedings of the [41st International Symposium on Computer Architecture \(ISCA\)](#), Minneapolis, MN, June 2014.

[[Slides \(pptx\) \(pdf\)](#)] [[Lightning Session Slides \(pptx\) \(pdf\)](#)]

[[Source Code](#)]

Memory Coherence (III)

Other mechanisms that can alleviate the overheads



Amirali Boroumand, Saugata Ghose, Minesh Patel, Hasan Hassan, Brandon Lucia, Kevin Hsieh, Krishna T. Malladi, Hongzhong Zheng, and Onur Mutlu,
["LazyPIM: An Efficient Cache Coherence Mechanism for Processing-in-Memory"](#)
[IEEE Computer Architecture Letters \(CAL\)](#), June 2016.

Outline

- Background
 - DRAM Organization
 - Processing-using-Memory
 - Rocket Chip SoC Generator
- Overview of PiDRAM
 - Hardware & Software Components
 - Prototype
- **Case Study #1 - RowClone**
 - Challenges
 - Allocation Mechanism
 - Memory Coherence
 - **Evaluation**
- Installing and Using PiDRAM

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

Test two configurations:

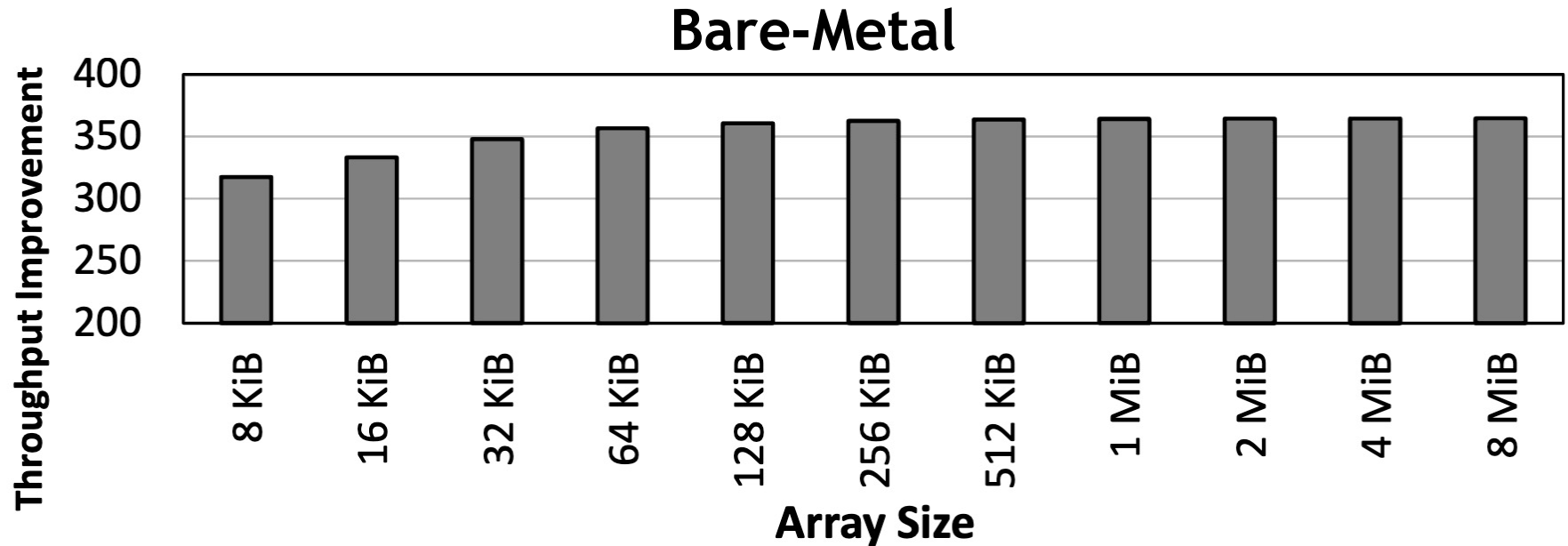
1. **Bare-Metal:** No address translation
2. **No Flush:** OS support, we assume data is always up-to-date in DRAM

Workloads:

Microbenchmarks: CPU-Copy, RowClone-Copy

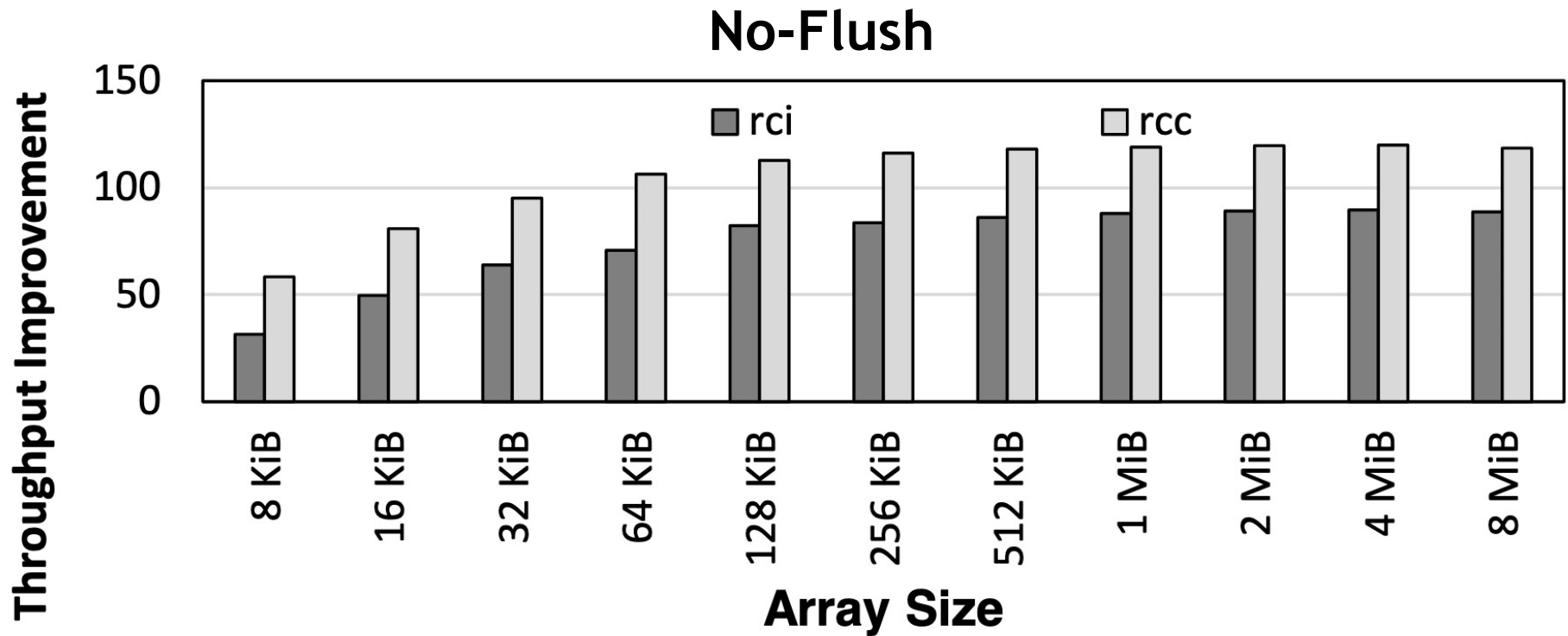
SPEC2006: libquantum

Copy Throughput Improvement (I)



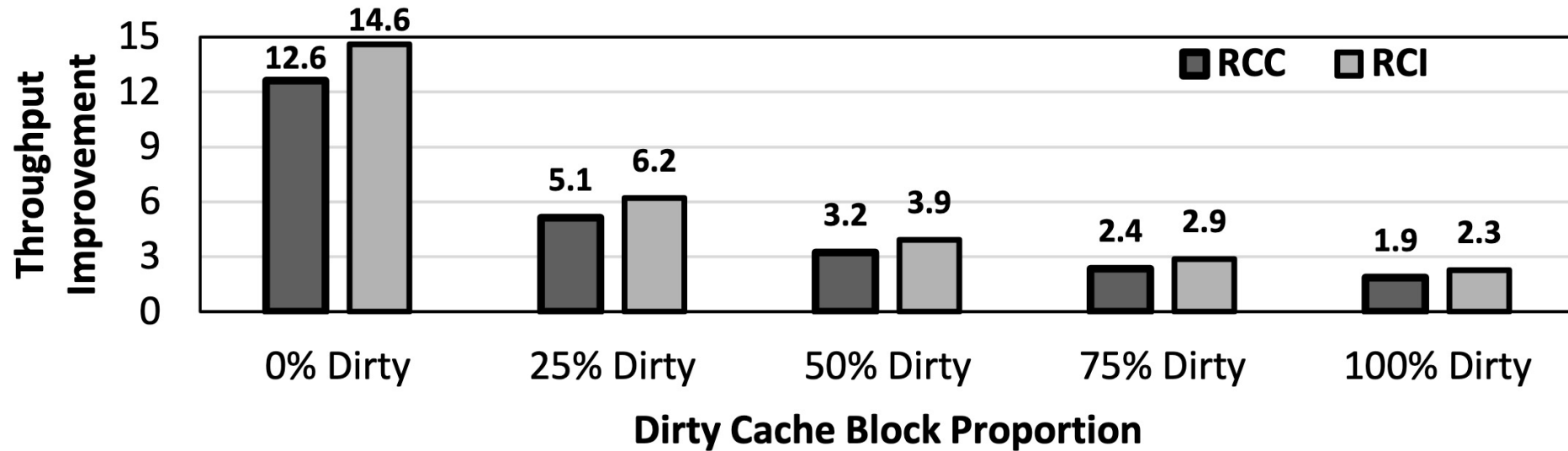
RowClone-Copy provides over 365x performance improvement over rocket CPU-Copy.

Copy Throughput Improvement (II)



RCC and RCI (with system support) improve copy throughput by 119x and 89x, respectively

CLFLUSH Overhead



CLFLUSH dramatically reduces the potential improvement

PiDRAM enables the study of better coherence mechanisms (e.g., DBI)

Evaluation - Summary

- PiDRAM can run real workloads end-to-end:
 - Replace libquantum `calloc()` with `alloc_align()` and `rci()`: 1.3% performance improvement
- RowClone-Copy can greatly improve bulk data copy performance with (119x) and without (365x) system support
- RowClone requires efficient coherency management mechanisms to achieve its potential copy throughput improvement

Potential Case Studies (I)

PiDRAM implements RowClone and D-RaNGe

AMBIT, SIMD RAM

- Memory allocation and alignment
- Memory coherence
- Transposition (SIMDRAM places data vertically)

DLPUF, QUAC-TRNG

- Memory scheduling policies
- Efficient post-processing integration

Potential Case Studies (II)

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 [102]	Bulk data-copy and initialization within DRAM	(i) <i>memory allocation and alignment mechanisms</i> that map source & destination operands of a copy operation into same DRAM subarray; (ii) <i>memory coherence</i> , i.e., source & destination operands must be up-to-date in DRAM.
D-RaNGe [67]	True random number generation using DRAM	(i) periodic generation of true random numbers; (ii) <i>memory scheduling policies</i> that minimize the interference caused by random number requests.
Ambit [99]	Bitwise operations in DRAM	(i) <i>memory allocation and alignment mechanisms</i> that map operands of a bitwise operation into same DRAM subarray; (ii) <i>memory coherence</i> , i.e., operands of the bitwise operations must be up-to-date in DRAM.
SIMDRAM [46]	Arithmetic operations in DRAM	(i) <i>memory allocation and alignment mechanisms</i> that map operands of an arithmetic operation into same DRAM subarray; (ii) <i>memory coherence</i> , i.e., operands of the arithmetic operations must be up-to-date in DRAM; (iii) <i>bit transposition</i> , i.e., operand bits must be laid out vertically in a single DRAM bitline.
DL-PUF [66]	Physical unclonable functions in DRAM	<i>memory scheduling policies</i> that minimize the interference caused by generating PUF responses.
QUAC-TRNG [89]	True random number generation using DRAM	(i) periodic generation of true random numbers; (ii) <i>memory scheduling policies</i> that minimize the interference caused by random number requests; (iii) efficient integration of the SHA-256 cryptographic hash function.

PiDRAM vs Other Platforms

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)	✓	✓	✓

PiDRAM enables end-to-end integration of PuM techniques via

- (1) Interface with real DRAM
- (2) Providing a flexible memory controller design
- (3) Open source design
- (4) Support for system software

Outline

- Background
 - DRAM Organization
 - Processing-using-Memory
 - Rocket Chip SoC Generator
- Overview of PiDRAM
 - Hardware & Software Components
 - Prototype
- Case Study #1 - RowClone
 - Challenges
 - Allocation Mechanism
 - Memory Coherence
 - Evaluation
- **Installing and Using PiDRAM**

BACKUP SLIDES

Alloc_align and RCC

