Improving DRAM Performance, Reliability, and Security by Rigorously Understanding Intrinsic DRAM Operation

Hasan Hassan

SAFARI Live Seminar

15 September 2022

ETH zürich

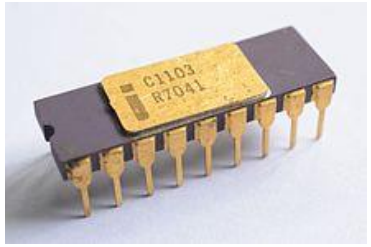
SAFARI

Dynamic Random Access Memory (DRAM)

Dr. Robert Dennard was granted the patent for one-transistor DRAM cell

[1968]

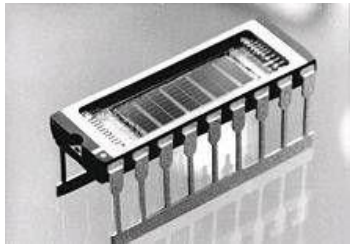
The first commercially available DRAM IC



Intel 1103 (1Kbit)

[1970]

The first SDRAM



Samsung KM48SL2000 (16 Mbit)

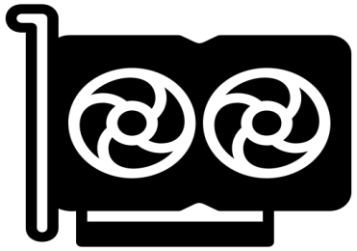
[1992]

The first DDR memory prototype

Samsung DDR SDRAM (64 Mbit)

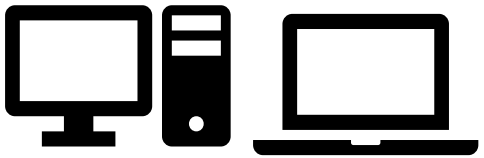
[1997]

DRAM in 2022

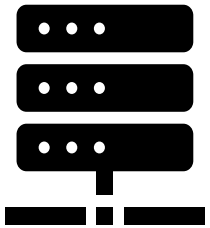


GPUs

(32Gbit)

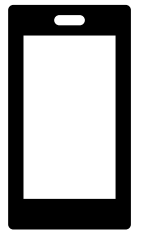


Personal Computers

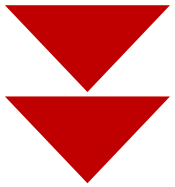


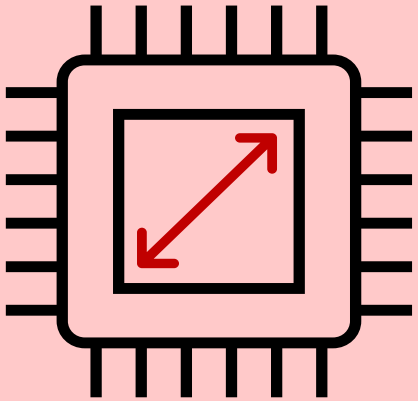
Servers

Safety-critical systems



Mobile Devices

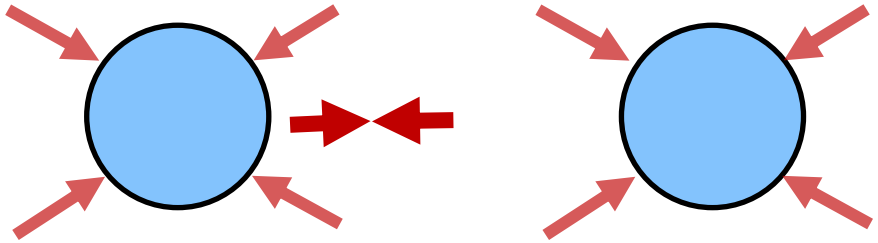




DRAM Scaling Challenges

Scaling Challenges of DRAM Technology

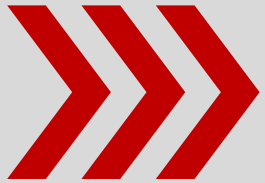
DRAM Cell



DRAM Scaling Problems

Data Integrity

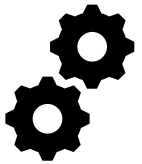
Latency



System-Level Problems

Performance

Energy Efficiency



Reliability

Security

Our Goals

To combat the **system-level implications** of the DRAM scaling challenges:

1.  Build an **infrastructure** for characterization, analysis, and understanding of real DRAM chips

2.  Enable new **mechanisms** for improving DRAM performance, energy consumption, reliability, and security

Contributions

A flexible and easy-to-use FPGA-based **DRAM characterization infrastructure**

SoftMC
HPCA'17

U-TRR
MICRO'21

A new methodology for **uncovering in-DRAM RowHammer Protection Mechanisms**

Self-Managing DRAM: Enabling autonomous and efficient in-DRAM maintenance operations

SMD
Ongoing
arXiv'22

CROW
ISCA'19

A low-cost substrate for improving DRAM performance, energy efficiency, and reliability

DRAM Background

DRAM Technology, Organization, and Operation

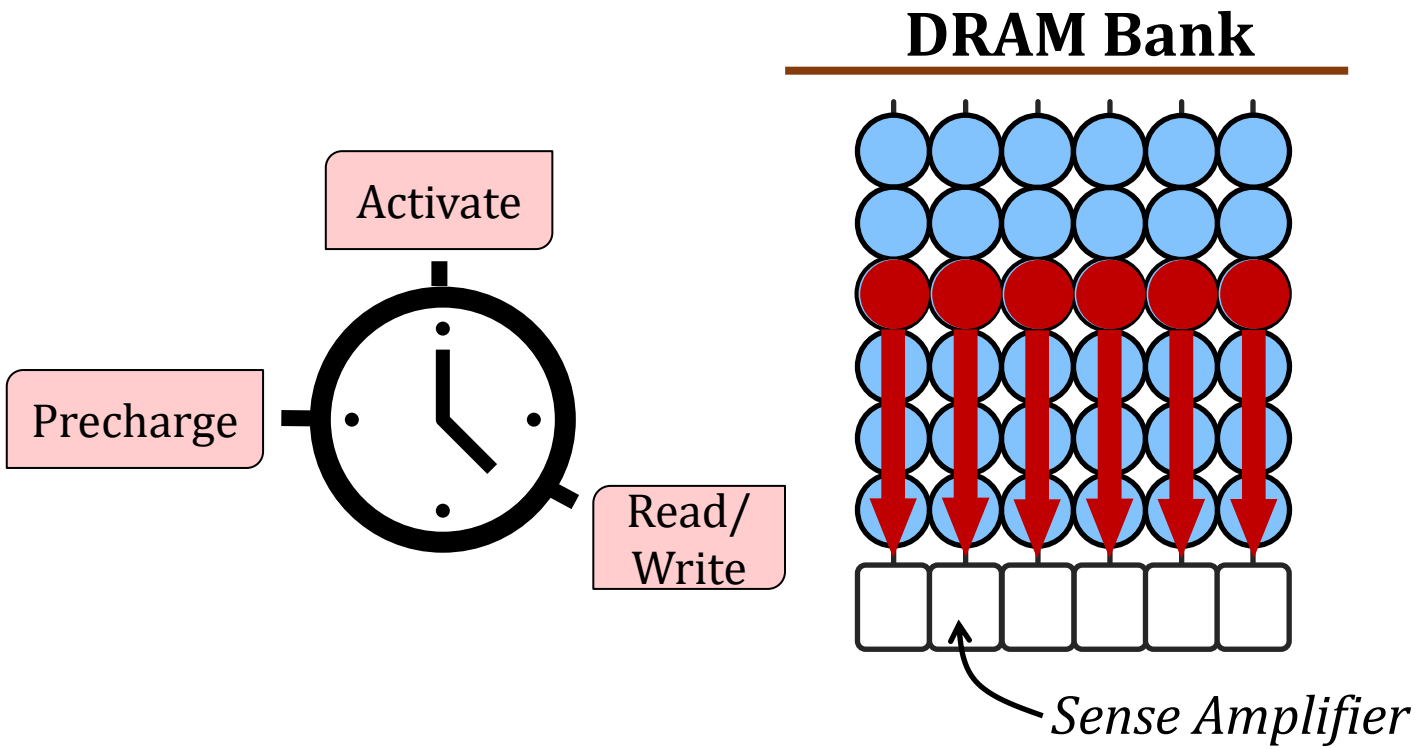
DRAM Organization

Memory Bus

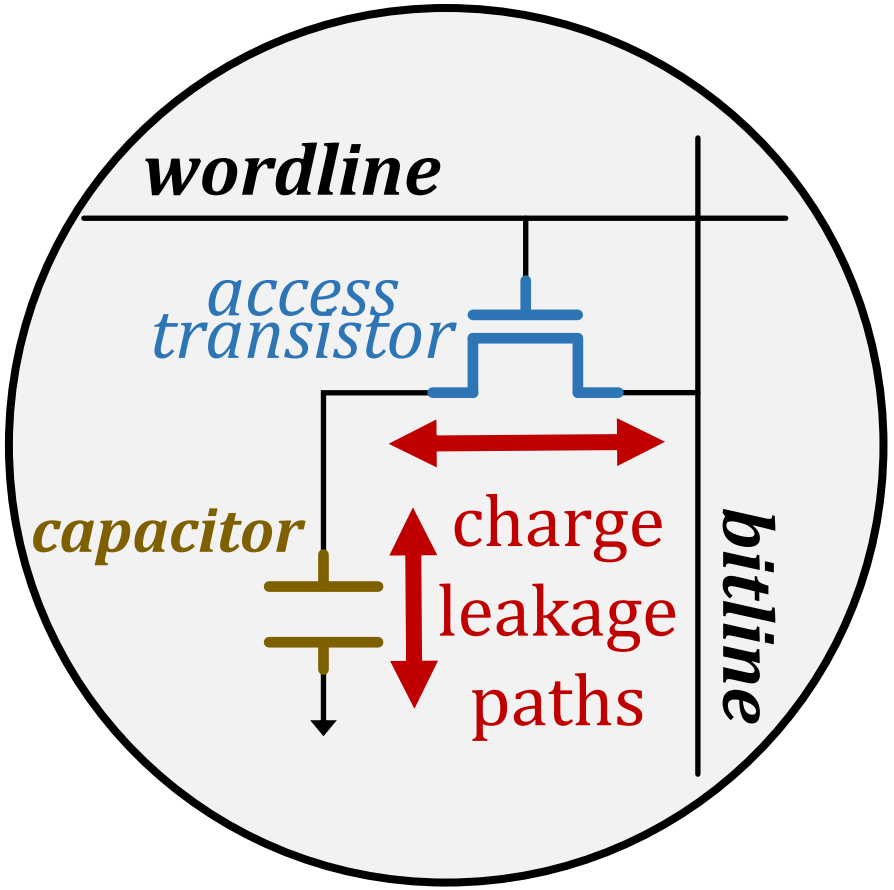
Memory Controller

CPU

DRAM Operation



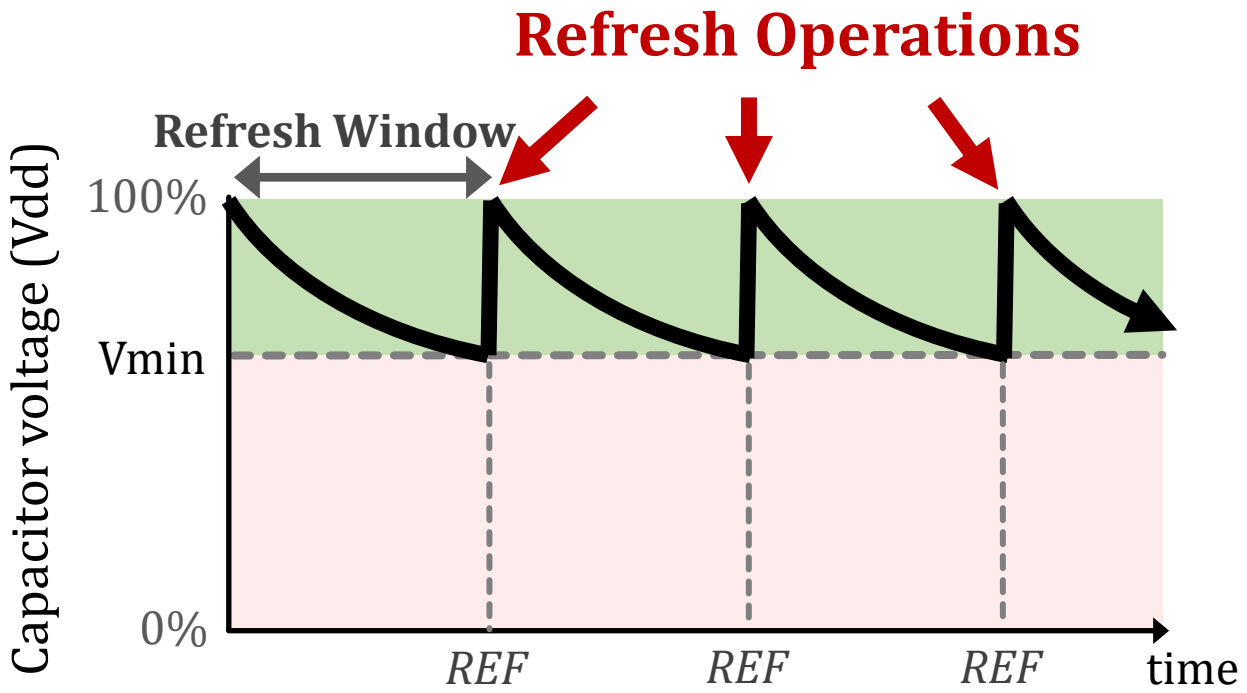
DRAM Cell



A single bit is encoded in a small (and leaky) capacitor

Stored data is **corrupted** if **too much charge leaks**

DRAM Refresh



Periodic **refresh operations** preserve stored data

Contributions

A flexible and easy-to-use FPGA-based **DRAM characterization infrastructure**

SoftMC
HPCA'17

U-TRR
MICRO'21

A new methodology for **uncovering in-DRAM RowHammer Protection Mechanisms**

Self-Managing DRAM: Enabling autonomous and efficient in-DRAM maintenance operations

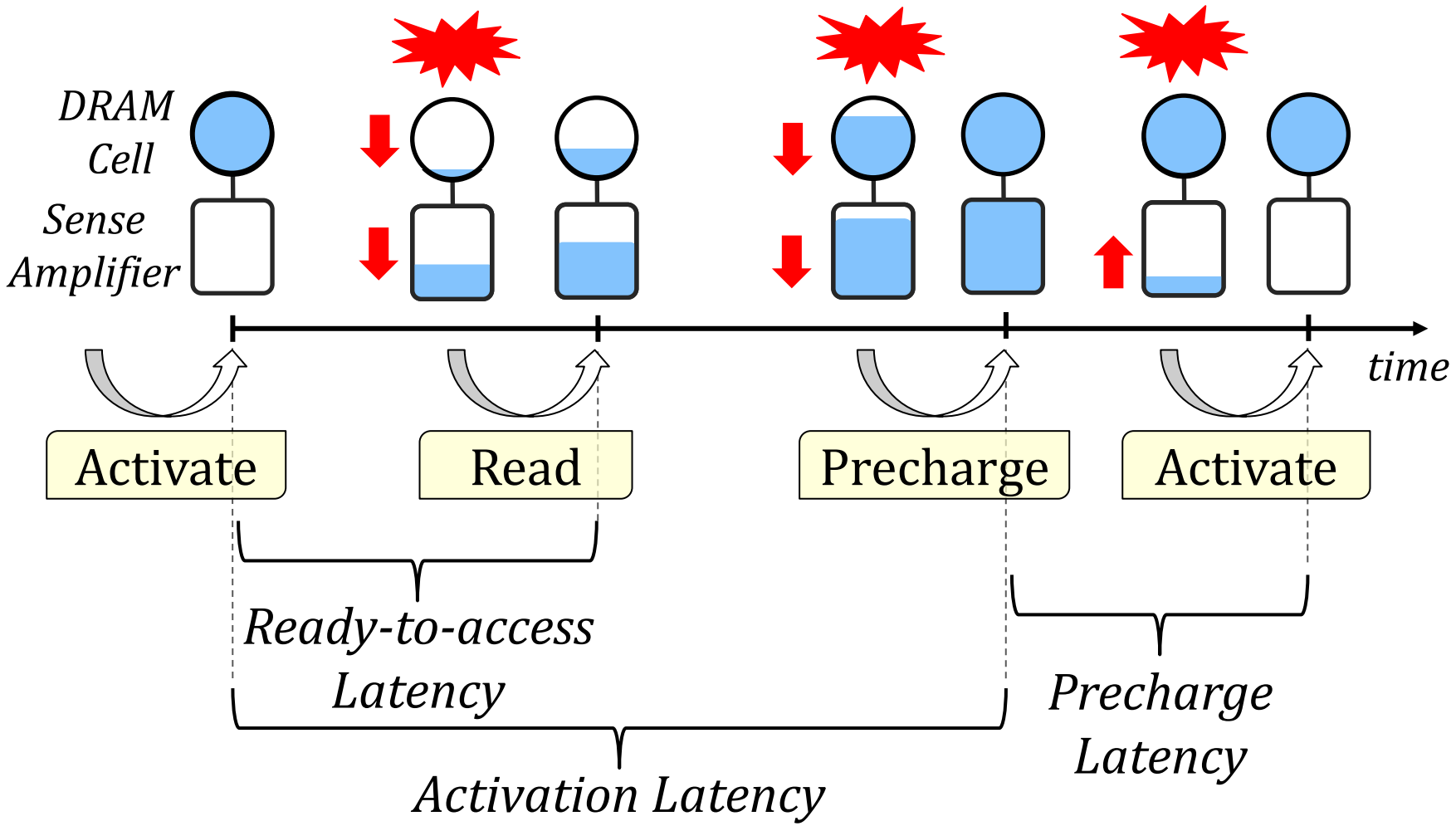
SMD
Ongoing
arXiv'22

CROW
ISCA'19

A low-cost substrate for improving DRAM performance, energy efficiency, and reliability

Reliability Effects of DRAM Timing Parameters

Many of the factors affecting DRAM **reliability** and **latency** cannot be properly modeled



Factors Affecting DRAM Reliability and Latency

DRAM timing violation

Inter-cell interference

Manufacturing process

Temperature

Voltage

●●●

We need to perform experimental studies of *real* DRAM chips

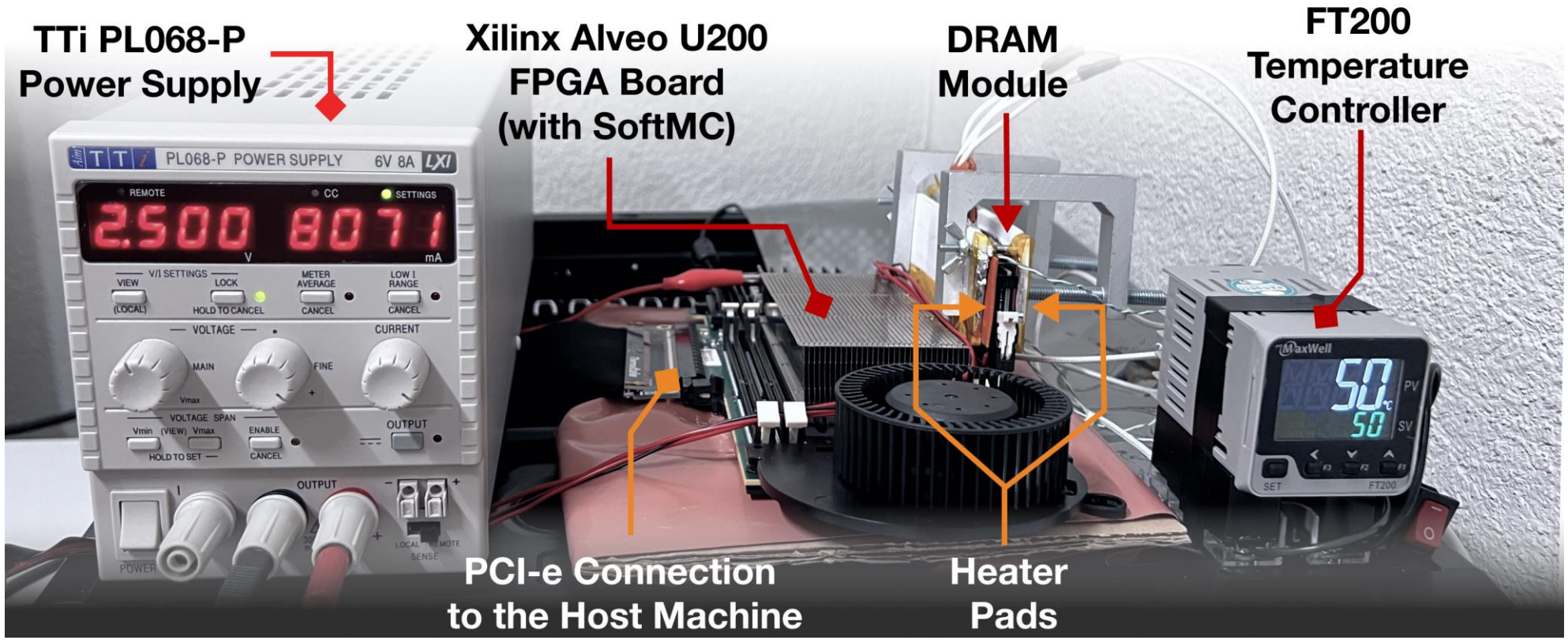
Goals of a DRAM Characterization Infrastructure

- 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*

SoftMC: High-Level View

The first publicly-available FPGA-based DRAM characterization infrastructure

Easily programmable using the SoftMC C++ API



[Yaglikci+, DSN'22]

Key Components

SoftMC API

PCIe Driver

SoftMC Hardware

SoftMC API

Writing data to DRAM:

```
InstructionSequence iseq;
iseq.insert(genACT(bank, row));
iseq.insert(genWAIT(tRCD));
iseq.insert(genWR(bank, col, data));
iseq.insert(genWAIT(tCL + tBL + tWR));
iseq.insert(genPRE(bank));
iseq.insert(genWAIT(tRP));
iseq.insert(genEND());
iseq.execute(fpga);
```

Instruction generator functions

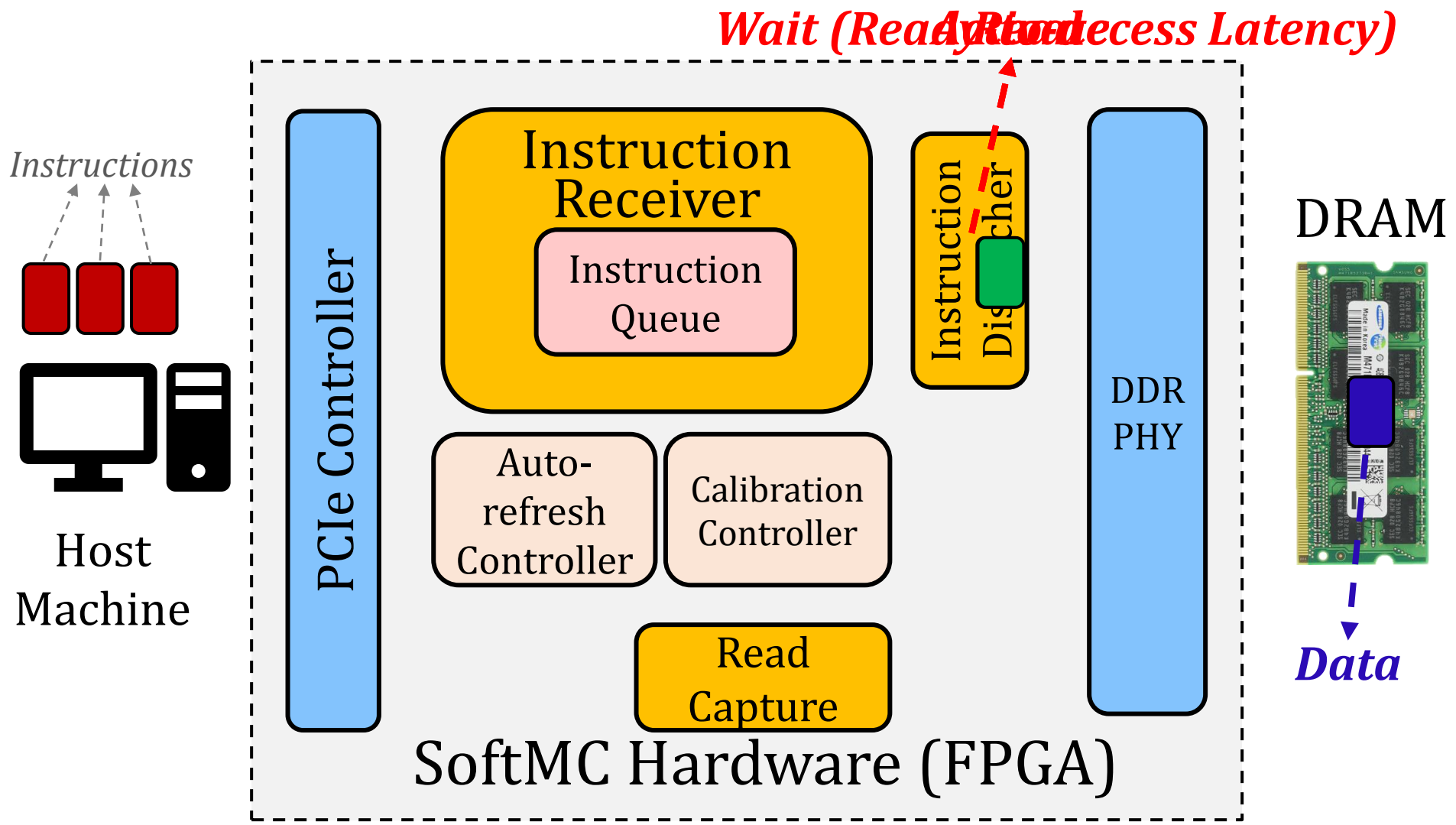
Key Components

SoftMC API

PCIe Driver

SoftMC Hardware

SoftMC Hardware

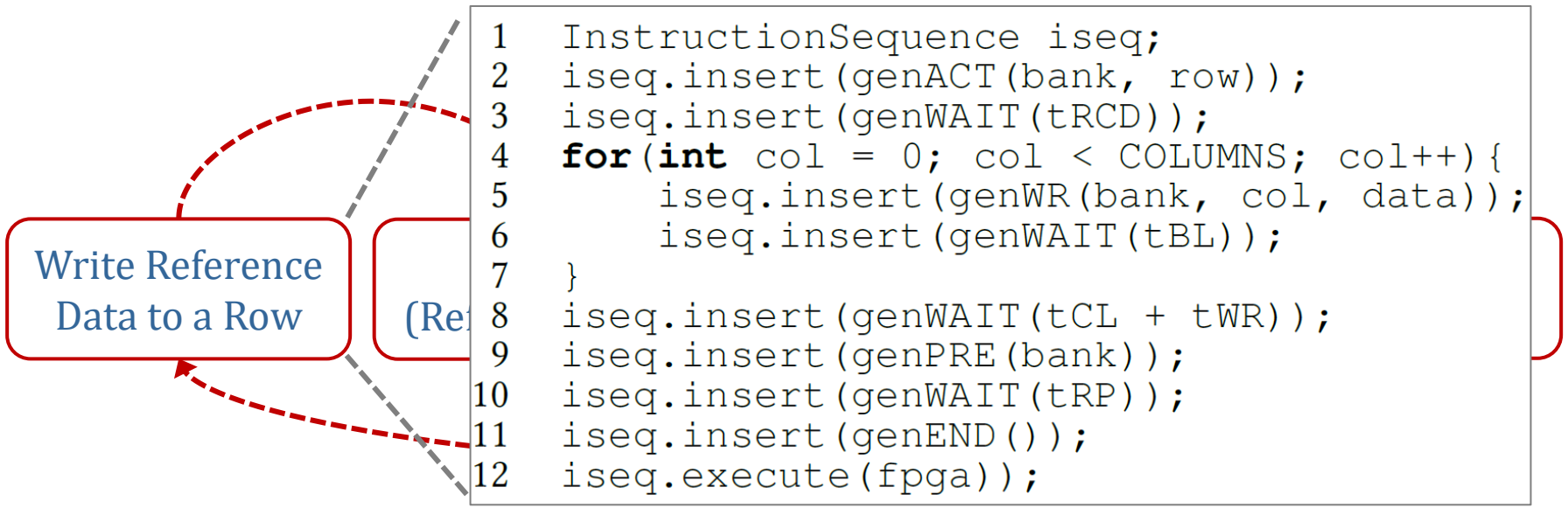


Use Cases

1. Retention Time Distribution Study

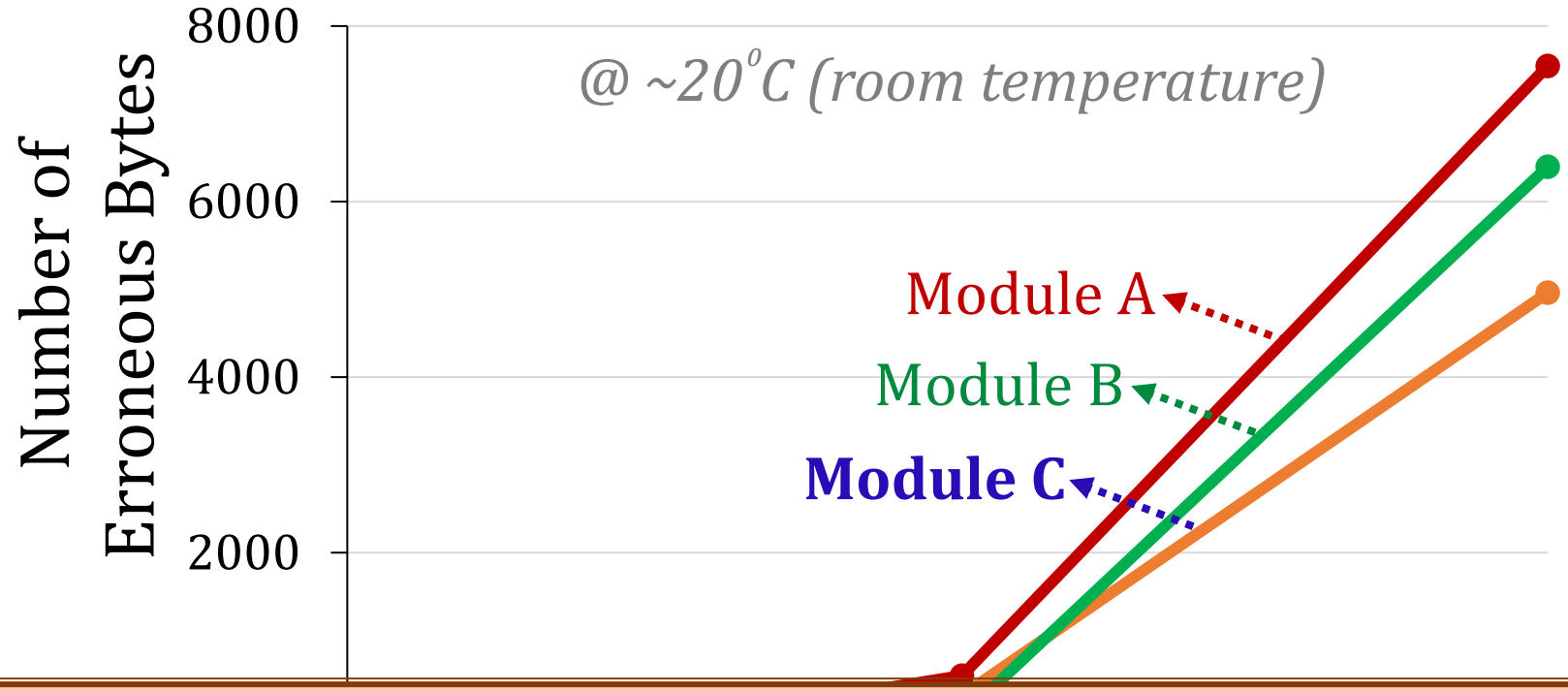
2. Evaluating the Effectiveness of New DRAM Latency Reduction Techniques

Use Case 1: Retention Time Distribution Study



Can be implemented with just ~100 lines of C++ code

Use Case 1: Results



Validates the correctness of the SoftMC Infrastructure

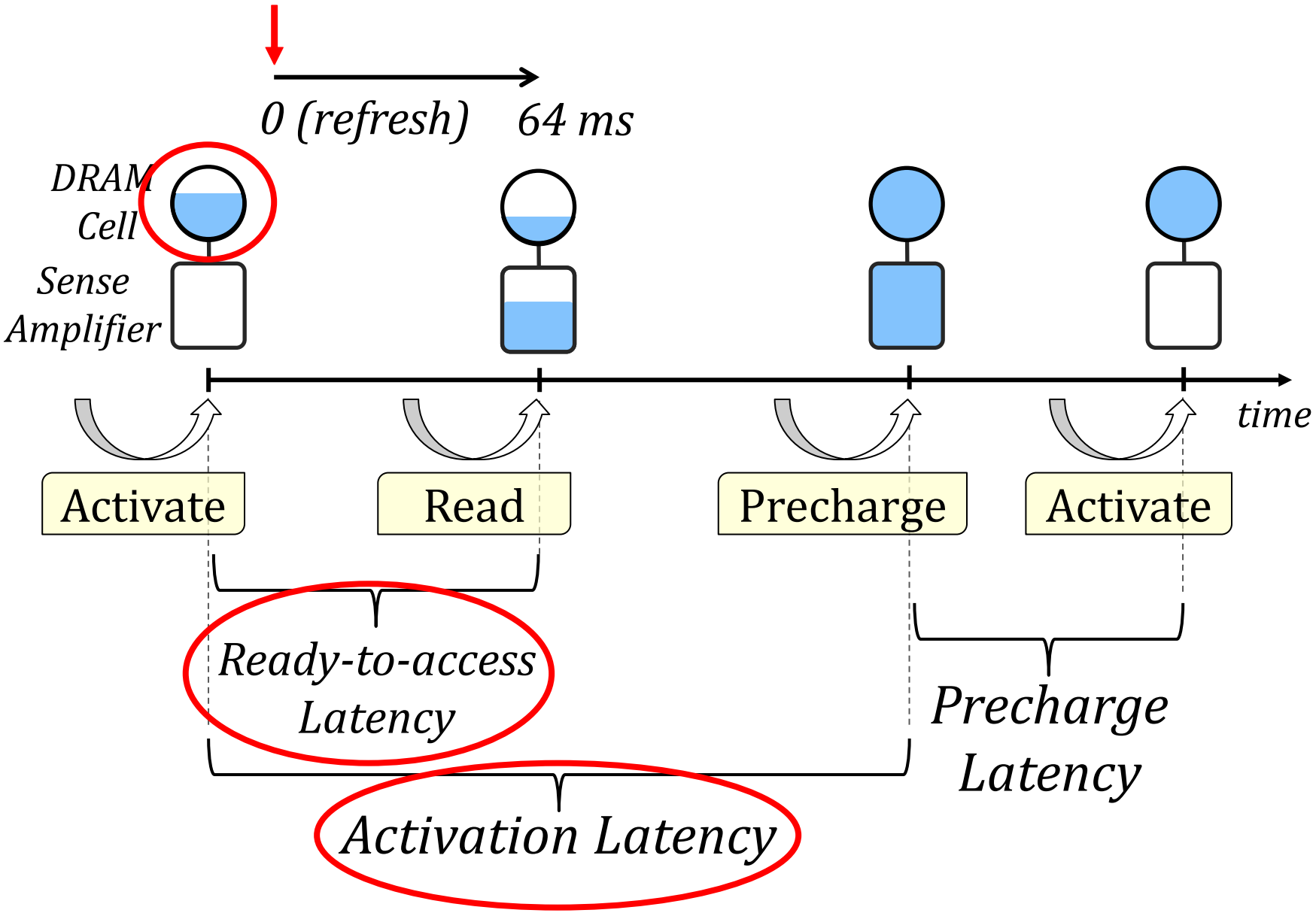
Use Case 2: Accessing Highly-Charged Cells Faster

NUAT *(Shin+, HPCA 2014)*

ChargeCache *(Hassan+, HPCA 2016)*

A highly-charged cell can be accessed with low latency

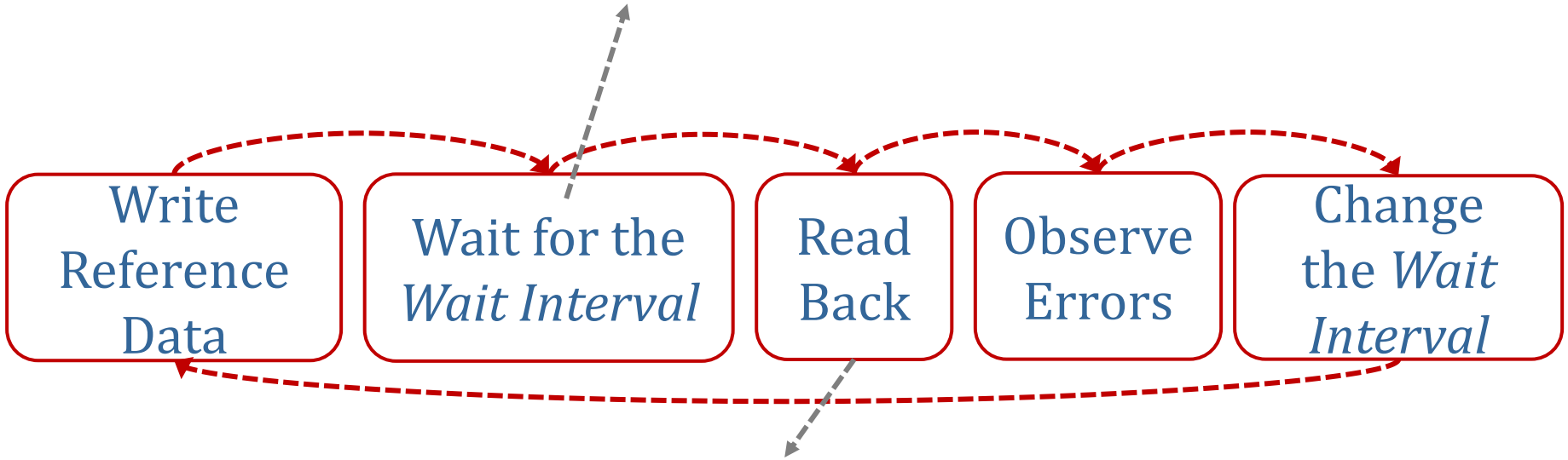
Use Case 2: How a Highly-Charged Cell Is Accessed Faster?



Use Case 2: Ready-to-Access Latency Test

Longer wait ⇒ Lower cell charge

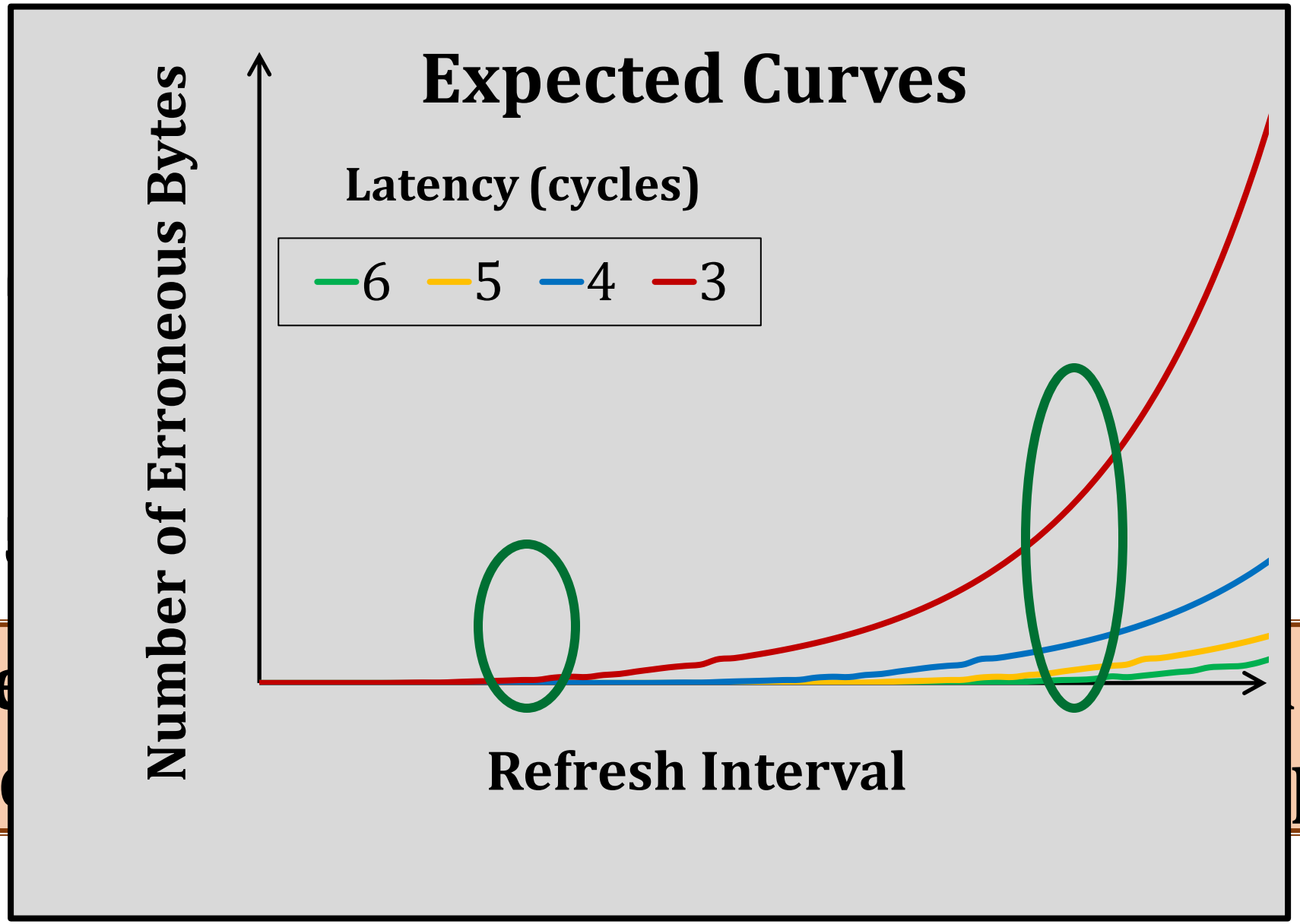
Shorter wait ⇒ Higher cell charge



With **custom** ready-to-access latency parameter

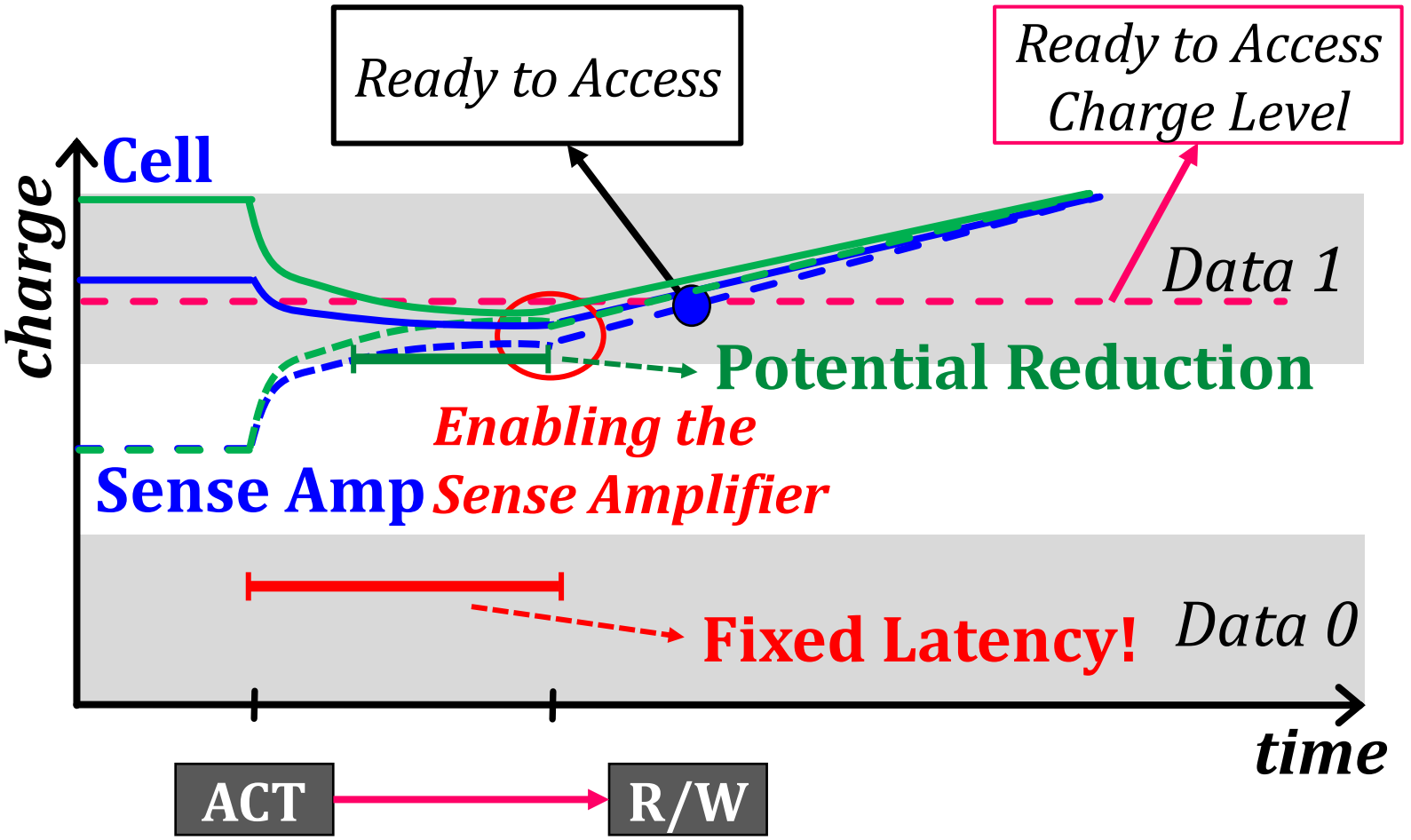
Can be implemented with just ~150 lines of C++ code

Use Case 2: Results



Use Case 2: Why Don't We See the Latency Reduction Effect?

The memory controller cannot externally control when a sense amplifier gets enabled in existing DRAM chips



Research Enabled by SoftMC (from SAFARI)

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

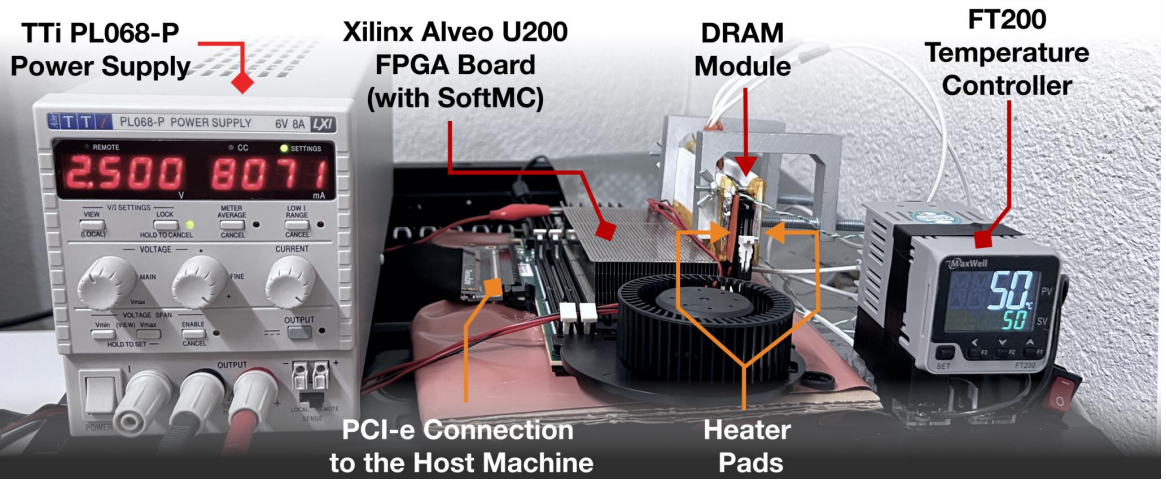
Research Enabled by SoftMC (others)

1) **[Applied Sciences'22]** Bepary+, "DRAM Retention Behavior with Accelerated Aging in Commercial Chips"
2) **[ETS'21]** Farmani+, "RHAT: Efficient RowHammer-Aware Test for Modern DRAM Modules"
3) **[HOST'20]** Talukder+, "Towards the Avoidance of Counterfeit Memory: Identifying the DRAM Origin"
4) **[MICRO'19]** Gao+, "ComputeDRAM: In-Memory Compute Using Off-the-Shelf DRAMs"
5) **[IEEE Access'19]** Talukder+, "PreLatPUF: Exploiting DRAM Latency Variations for Generating Robust Device Signatures"
6) **[ICCE'18]** Talukder+, "Exploiting DRAM Latency Variations for Generating True Random Numbers"

Summary

SoftMC

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



[Yaglikci+, DSN'22]

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

github.com/CMU-SAFARI/SoftMC

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

Contributions

A flexible and easy-to-use FPGA-based **DRAM characterization infrastructure**

SoftMC
HPCA'17

U-TRR
MICRO'21

A new methodology for **uncovering in-DRAM RowHammer Protection Mechanisms**

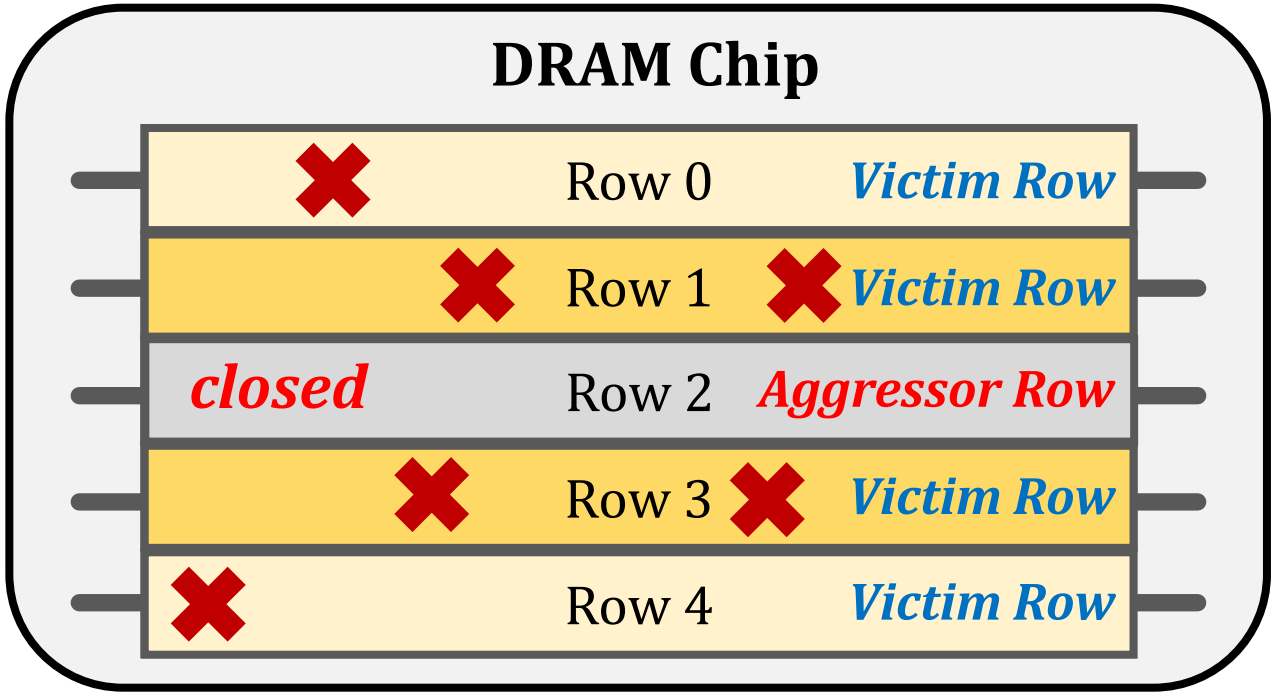
Self-Managing DRAM: Enabling autonomous and efficient in-DRAM maintenance operations

SMD
Ongoing
arXiv'22

CROW
ISCA'19

A low-cost substrate for improving DRAM performance, energy efficiency, and reliability

RowHammer

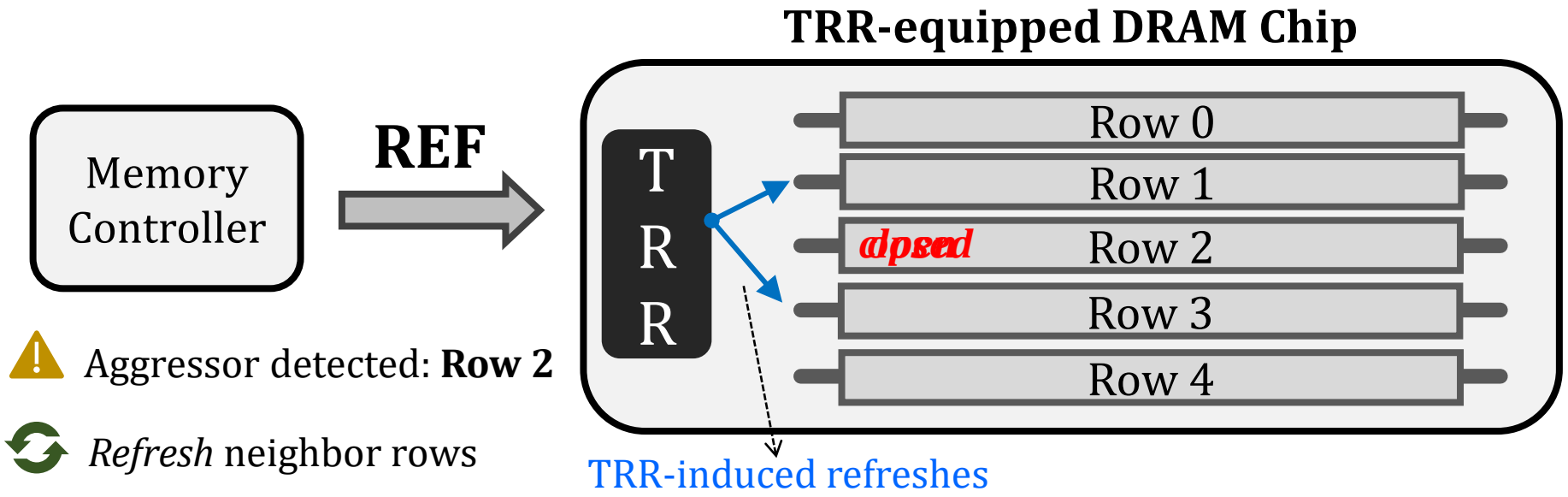


Repeatedly **opening** (activating) and **closing** (precharging) a DRAM row causes **RowHammer bit flips** in nearby cells

Target Row Refresh (TRR)

DRAM vendors equip their DRAM chips with a *proprietary* mitigation mechanisms known as **Target Row Refresh (TRR)**

Key Idea: TRR refreshes nearby rows upon detecting an aggressor row



The Problem with TRR

TRR is obscure, undocumented, and proprietary

We cannot easily study the *security properties* of TRR

Goal

Study **in-DRAM TRR** mechanisms to

1. **understand** how they operate
2. **assess** their security
3. **secure** DRAM completely against RowHammer

U-TRR: Uncovering Inner Workings of TRR

A new methodology to
uncover the **inner workings of TRR**

Key Idea:

Use data retention failures as a side channel
to detect when a row is refreshed by TRR

U-TRR: High-Level Overview

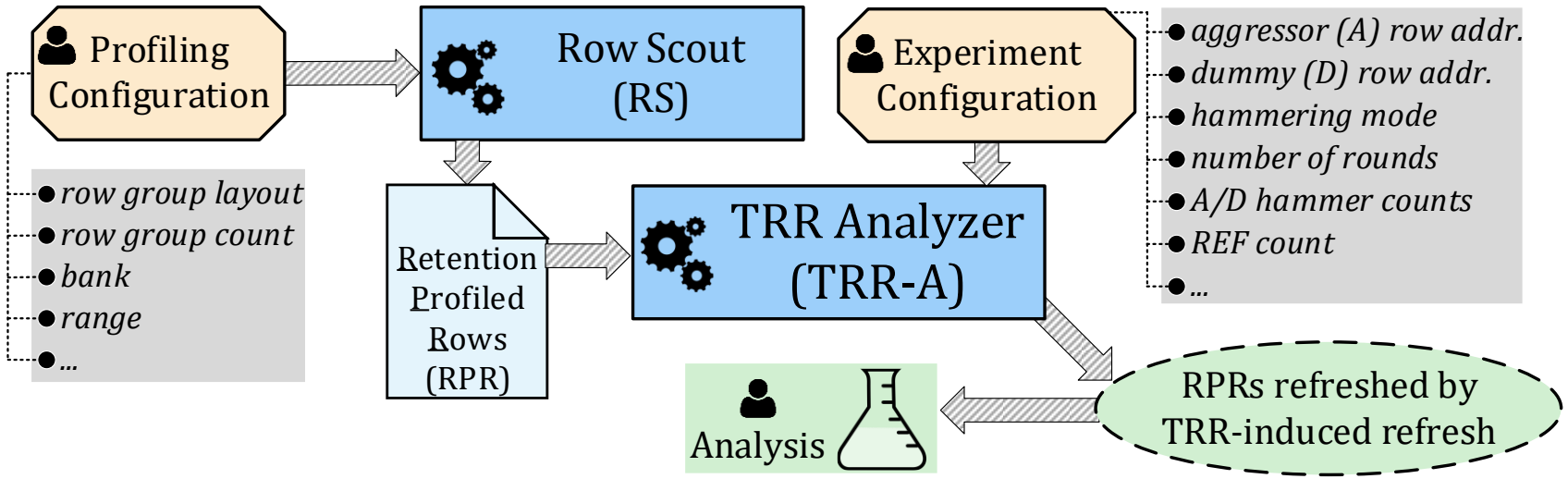
U-TRR has two main components

Row Scout (RS)

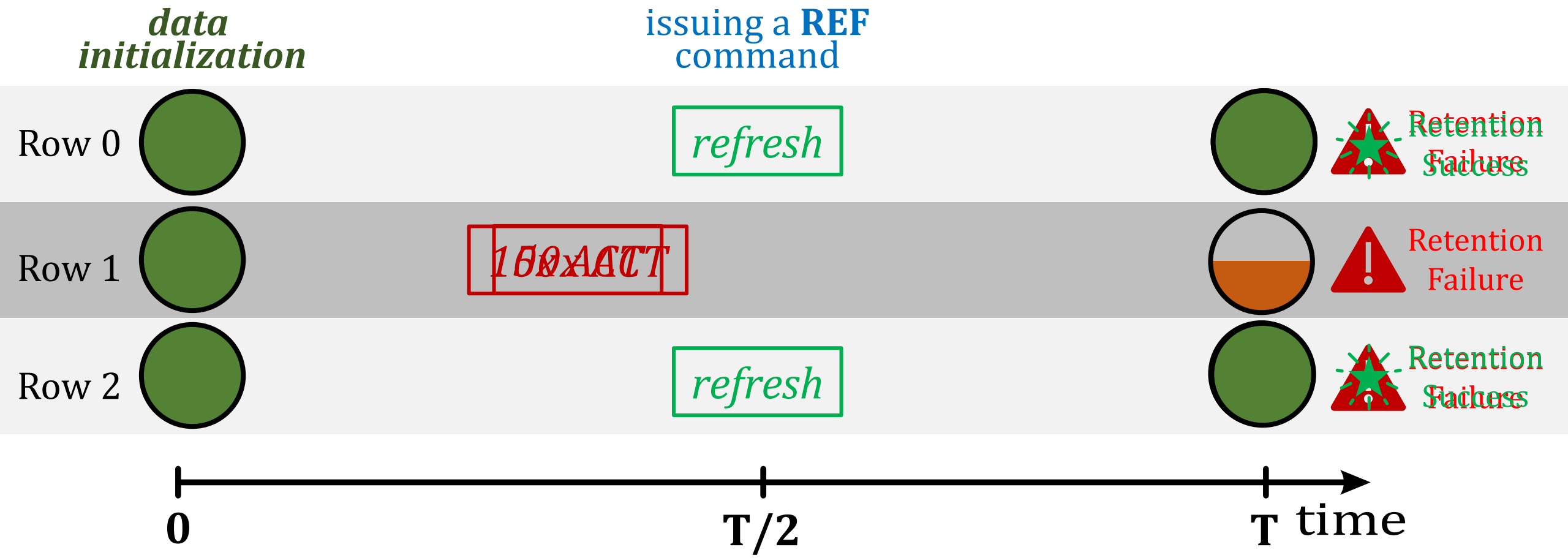
Finds a **set of DRAM rows** that meet certain requirements as needed by TRR-A and **identifies the data retention times** of these rows

TRR Analyzer (TRR-A)

Uses RS-provided rows to **distinguish between TRR-induced and regular refreshes**, and thus builds an understanding of the underlying TRR mechanism



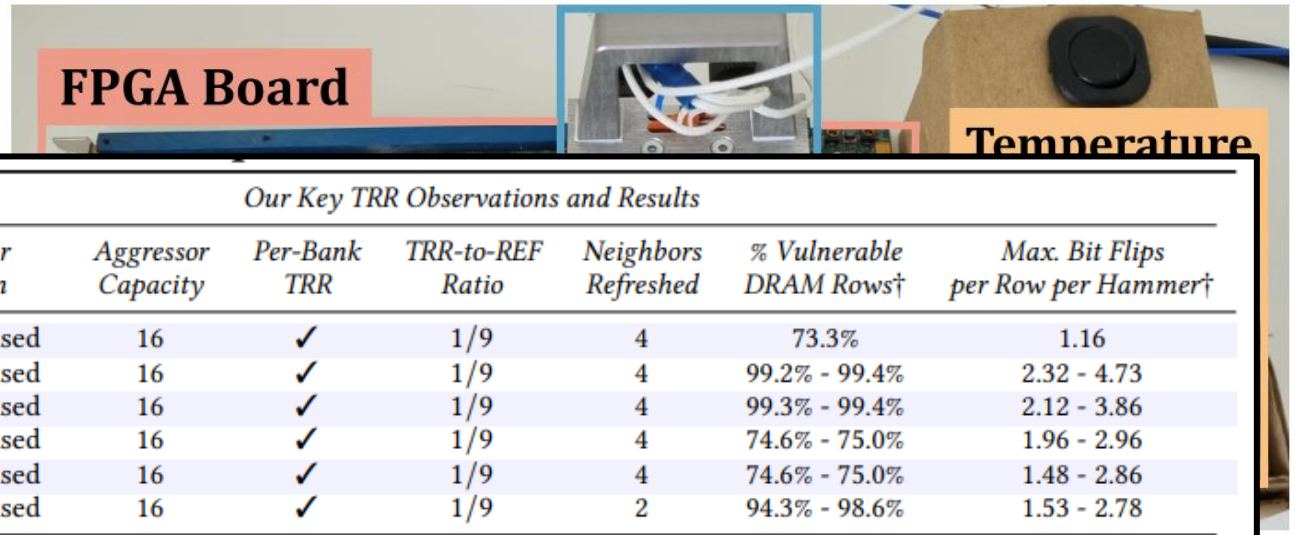
High-Level U-TRR Experiment



We perform **many** experiments to understand the operation of **different TRR mechanisms**

U-TRR: Implementation

We implement U-TRR using *SoftMC*,



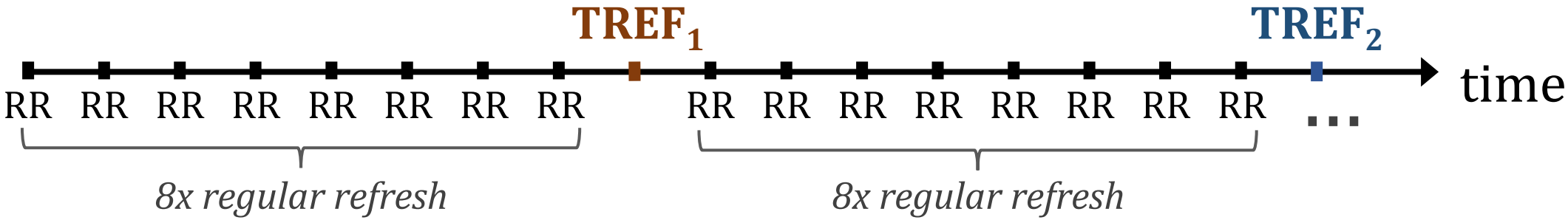
Module	Date (yy-ww)	Chip Density (Gbit)	Organization			HC_{first}†	Our Key TRR Observations and Results							
			Ranks	Banks	Pins		Version	Aggressor Detection	Aggressor Capacity	Per-Bank TRR	TRR-to-REF Ratio	Neighbors Refreshed	% Vulnerable DRAM Rows†	Max. Bit Flips per Row per Hammer†
A0	19-50	8	1	16	8	16K	A_{TRR1}	Counter-based	16	✓	1/9	4	73.3%	1.16
A1-5	19-36	8	1	8	16	13K-15K	A_{TRR1}	Counter-based	16	✓	1/9	4	99.2% - 99.4%	2.32 - 4.73
A6-7	19-45	8	1	8	16	13K-15K	A_{TRR1}	Counter-based	16	✓	1/9	4	99.3% - 99.4%	2.12 - 3.86
A8-9	20-07	8	1	16	8	12K-14K	A_{TRR1}	Counter-based	16	✓	1/9	4	74.6% - 75.0%	1.96 - 2.96
A10-12	19-51	8	1	16	8	12K-13K	A_{TRR1}	Counter-based	16	✓	1/9	4	74.6% - 75.0%	1.48 - 2.86
A13-14	20-31	8	1	8	16	11K-14K	A_{TRR2}	Counter-based	16	✓	1/9	2	94.3% - 98.6%	1.53 - 2.78
B0	18-22	4	1	16	8	44K	B_{TRR1}	Sampling-based	1	✗	1/4	2	99.9%	2.13
B1-4	20-17	4	1	16	8	159K-192K	B_{TRR1}	Sampling-based	1	✗	1/4	2	23.3% - 51.2%	0.06 - 0.11
B5-6	16-48	4	1	16	8	44K-50K	B_{TRR1}	Sampling-based	1	✗	1/4	2	99.9%	1.85 - 2.03
B7	19-06	8	2	16	8	20K	B_{TRR1}	Sampling-based	1	✗	1/4	2	99.9%	31.14
B8	18-03	4	1	16	8	43K	B_{TRR1}	Sampling-based	1	✗	1/4	2	99.9%	2.57
B9-12	19-48	8	1	16	8	42K-65K	B_{TRR2}	Sampling-based	1	✗	1/9	2	36.3% - 38.9%	16.83 - 24.26
B13-14	20-08	4	1	16	8	11K-14K	B_{TRR3}	Sampling-based	1	✓	1/2	4	99.9%	16.20 - 18.12
C0-3	16-48	4	1	16	x8	137K-194K	C_{TRR1}	Mix	Unknown	✓	1/17	2	1.0% - 23.2%	0.05 - 0.15
C4-6	17-12	8	1	16	x8	130K-150K	C_{TRR1}	Mix	Unknown	✓	1/17	2	7.8% - 12.0%	0.06 - 0.08
C7-8	20-31	8	1	8	x16	40K-44K	C_{TRR1}	Mix	Unknown	✓	1/17	2	39.8% - 41.8%	9.66 - 14.56
C9-11	20-31	8	1	8	x16	42K-53K	C_{TRR2}	Mix	Unknown	✓	1/9	2	99.7%	9.30 - 32.04
C12-14	20-46	16	1	8	x16	6K-7K	C_{TRR3}	Mix	Unknown	✓	1/8	2	99.9%	4.91 - 12.64

Table 1 in the paper provides more information about the analyzed modules

15x Vendor C DDR4 modules

Case Study: Understanding Vendor A's TRR

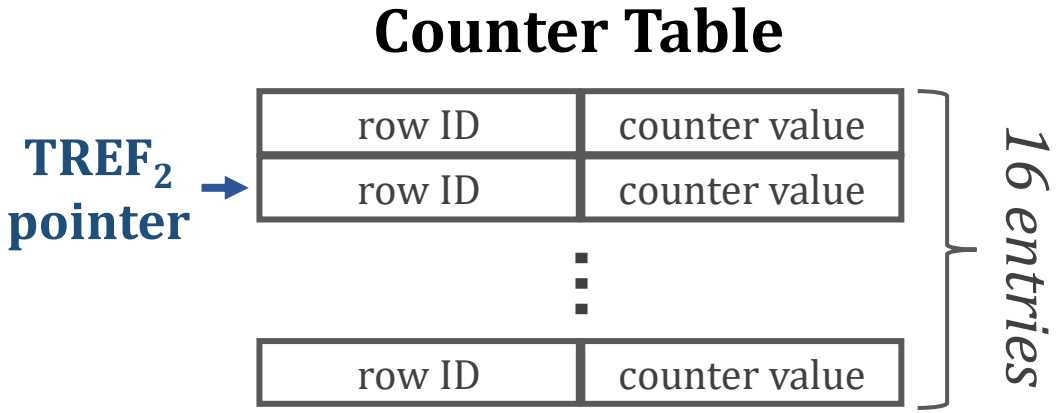
Refresh Types:
- Regular Refresh (RR)
- TRR-capable Refresh ($\mathbf{TREF_1}$ and $\mathbf{TREF_2}$)



Observation: TRR tracks potentially aggressor rows using a Counter Table

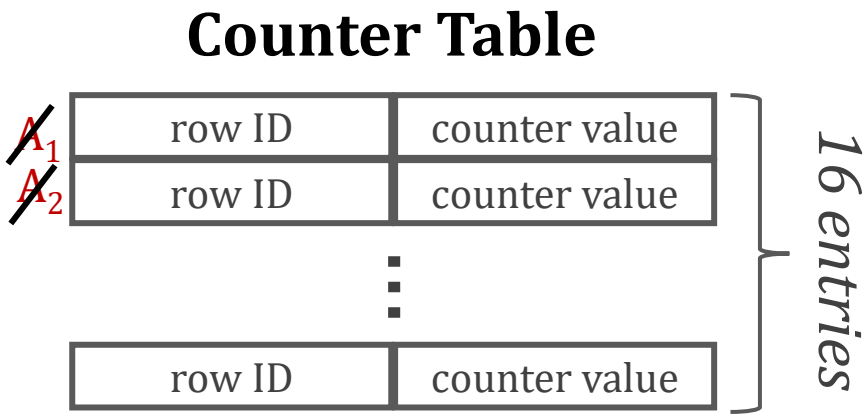
$\mathbf{TREF_1}$**:** Refreshes the victims of **row ID** with the **largest counter value**

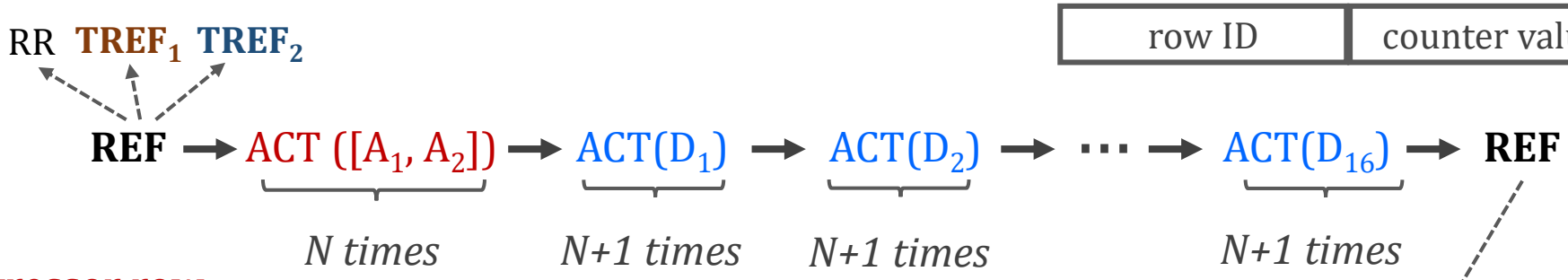
$\mathbf{TREF_2}$**:** Refreshes the victims of **row ID** that $TREF_2$ pointer refers to



Case Study: Circumventing Vendor A's TRR

Approach: Ensure an aggressor row is **discarded** from the *Counter Table* **prior** to a REF command





[A1, A2] **not** refreshed by TRR

A_i: aggressor row
D_i: dummy row

This RowHammer access pattern requires **synchronizing** accesses with REF commands

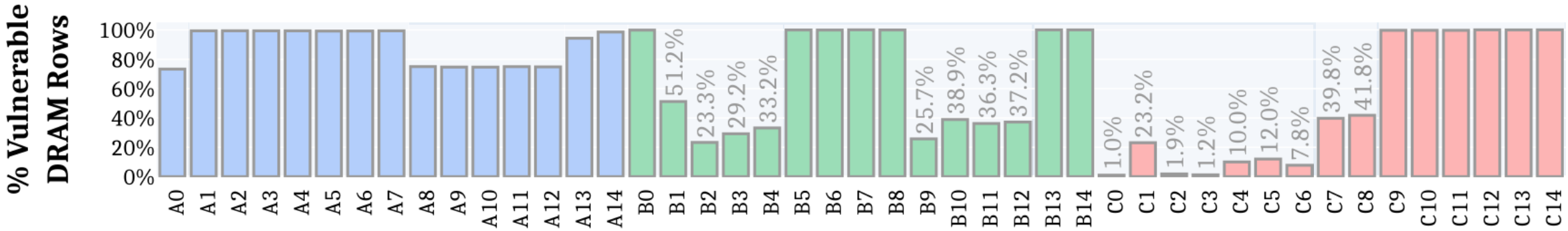
Circumventing Vendor A's TRR by discarding the actual aggressor rows from the Counter Table

Bypassing TRR with New RowHammer Access Patterns

We craft **new RowHammer access patterns** that circumvent TRR of three major DRAM vendors

On the **45** DDR4 modules we test, the new access patterns cause a large number of RowHammer bit flips

Effect on Individual Rows



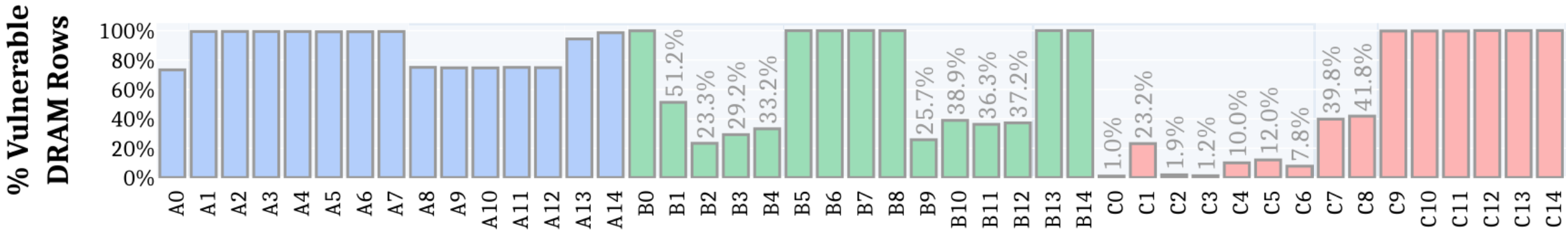
All 45 modules we tested are vulnerable to our new RowHammer access patterns

For many modules, our RowHammer access patterns cause bit flips in more than 99.9% of the rows

Why are some modules less vulnerable?

1) Fundamentally less vulnerable to RowHammer
2) Different TRR mechanisms
3) Unique row organization

Effect on Individual Rows



All 45 modules we tested are vulnerable to our new RowHammer access patterns

For many modules, our RowHammer access patterns cause bit flips in more than 99.9% of the rows

Our access patterns successfully **circumvent** the TRR implementations of **all three major DRAM vendors**

Other Observations and Results in the Paper

- More observations on the TRRs of the three vendors
- Analysis on the effectiveness of ECC against our RowHammer access patterns
- Detailed description of the crafted access patterns
- Hammers per aggressor row sensitivity analysis
- Observations and results for individual modules
- …

Module	Date (yy-ww)	Chip Density (Gbit)	Organization			HC_{first}†	Our Key TRR Observations and Results							
			Ranks	Banks	Pins		Version	Aggressor Detection	Aggressor Capacity	Per-Bank TRR	TRR-to-REF Ratio	Neighbors Refreshed	% Vulnerable DRAM Rows†	Max. Bit Flips per Row per Hammer†
A0	19-50	8	1	16	8	16K	A_{TRR1}	Counter-based	16	✓	1/9	4	73.3%	1.16
A1-5	19-36	8	1	8	16	13K-15K	A_{TRR1}	Counter-based	16	✓	1/9	4	99.2% - 99.4%	2.32 - 4.73
A6-7	19-45	8	1	8	16	13K-15K	A_{TRR1}	Counter-based	16	✓	1/9	4	99.3% - 99.4%	2.12 - 3.86
A8-9	20-07	8	1	16	8	12K-14K	A_{TRR1}	Counter-based	16	✓	1/9	4	74.6% - 75.0%	1.96 - 2.96
A10-12	19-51	8	1	16	8	12K-13K	A_{TRR1}	Counter-based	16	✓	1/9	4	74.6% - 75.0%	1.48 - 2.86
A13-14	20-31	8	1	8	16	11K-14K	A_{TRR2}	Counter-based	16	✓	1/9	2	94.3% - 98.6%	1.53 - 2.78
B0	18-22	4	1	16	8	44K	B_{TRR1}	Sampling-based	1	✗	1/4	2	99.9%	2.13
B1-4	20-17	4	1	16	8	159K-192K	B_{TRR1}	Sampling-based	1	✗	1/4	2	23.3% - 51.2%	0.06 - 0.11
B5-6	16-48	4	1	16	8	44K-50K	B_{TRR1}	Sampling-based	1	✗	1/4	2	99.9%	1.85 - 2.03
B7	19-06	8	2	16	8	20K	B_{TRR1}	Sampling-based	1	✗	1/4	2	99.9%	31.14
B8	18-03	4	1	16	8	43K	B_{TRR1}	Sampling-based	1	✗	1/4	2	99.9%	2.57
B9-12	19-48	8	1	16	8	42K-65K	B_{TRR2}	Sampling-based	1	✗	1/9	2	36.3% - 38.9%	16.83 - 24.26
B13-14	20-08	4	1	16	8	11K-14K	B_{TRR3}	Sampling-based	1	✓	1/2	4	99.9%	16.20 - 18.12
C0-3	16-48	4	1	16	x8	137K-194K	C_{TRR1}	Mix	Unknown	✓	1/17	2	1.0% - 23.2%	0.05 - 0.15
C4-6	17-12	8	1	16	x8	130K-150K	C_{TRR1}	Mix	Unknown	✓	1/17	2	7.8% - 12.0%	0.06 - 0.08
C7-8	20-31	8	1	8	x16	40K-44K	C_{TRR1}	Mix	Unknown	✓	1/17	2	39.8% - 41.8%	9.66 - 14.56
C9-11	20-31	8	1	8	x16	42K-53K	C_{TRR2}	Mix	Unknown	✓	1/9	2	99.7%	9.30 - 32.04
C12-14	20-46	16	1	8	x16	6K-7K	C_{TRR3}	Mix	Unknown	✓	1/8	2	99.9%	4.91 - 12.64

Summary

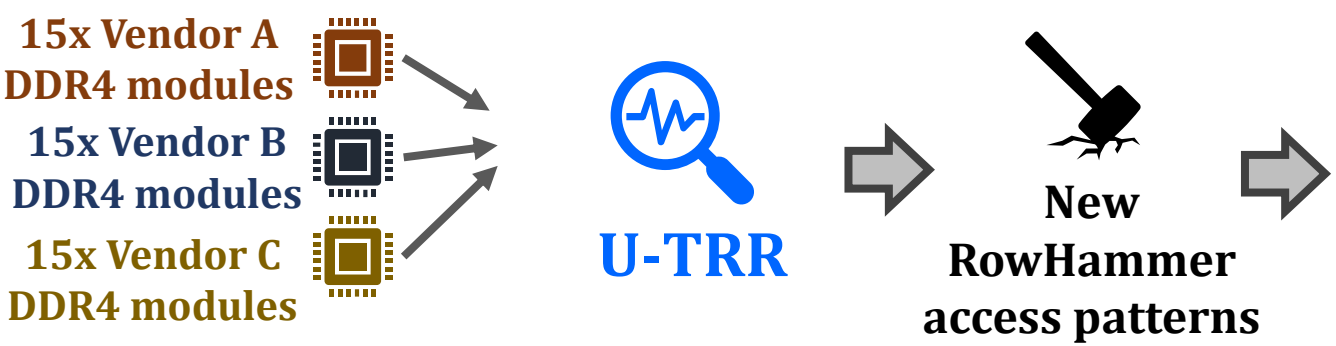
Target Row Refresh (TRR):
a set of obscure, undocumented, and proprietary RowHammer mitigation techniques

We cannot easily study the *security properties* of TRR

Is TRR fully secure? How can we validate its security guarantees?

U-TRR

A new methodology that leverages *data retention failures* to uncover the inner workings of TRR and study its security



All 45 modules we test are **vulnerable**

99.9% of rows in a DRAM bank experience **at least one RowHammer bit flip**

Up to **7** RowHammer **bit flips** in an 8-byte dataword, **making ECC ineffective**

U-TRR can facilitate the development of **new RowHammer attacks** and **more secure RowHammer protection** mechanisms

Contributions

A flexible and easy-to-use FPGA-based **DRAM characterization infrastructure**

SoftMC
HPCA'17

U-TRR
MICRO'21

A new methodology for **uncovering in-DRAM RowHammer Protection Mechanisms**

Self-Managing DRAM: Enabling autonomous and efficient in-DRAM maintenance operations

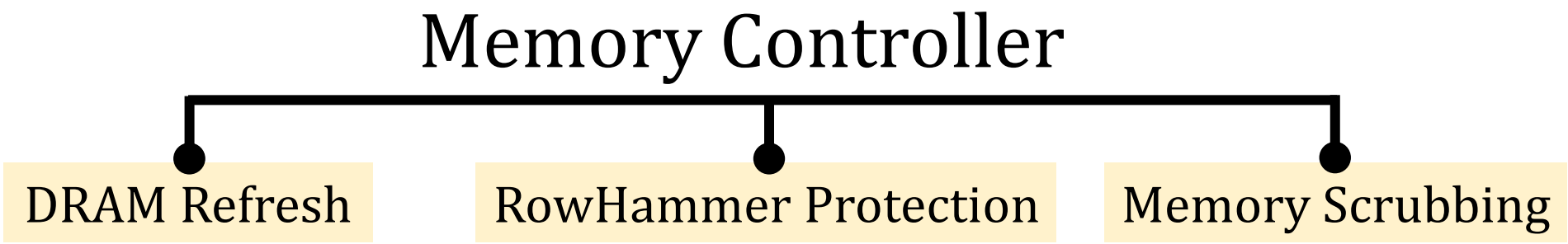
SMD
Ongoing
arXiv'22

CROW
ISCA'19

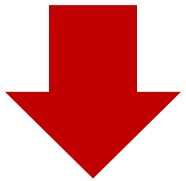
A low-cost substrate for improving DRAM performance, energy efficiency, and reliability

Problem: The Rigid DRAM Interface

The **Memory Controller** manages DRAM maintenance operations



Changes to maintenance operations are often reflected to the memory controller design, DRAM interface, and other system components



Implementing new maintenance operations (or modifying the existing ones) is difficult-to-realize

A Prime Example: New Features of DDR5

DRAM Refresh

Same Bank Refresh - simultaneously refreshes one bank in each bank group

The new **REFsb** command requires changes in DRAM interface and memory controller

Ro

Refr ow

DRA

The ne

DDR5 changes are difficult-to-implement as they were *only* possible after multiple years required to develop a new DRAM standard

Memory Scrubbing

In-DRAM Scrubbing – DDR5 uses the on-die ECC to perform periodic scrubbing

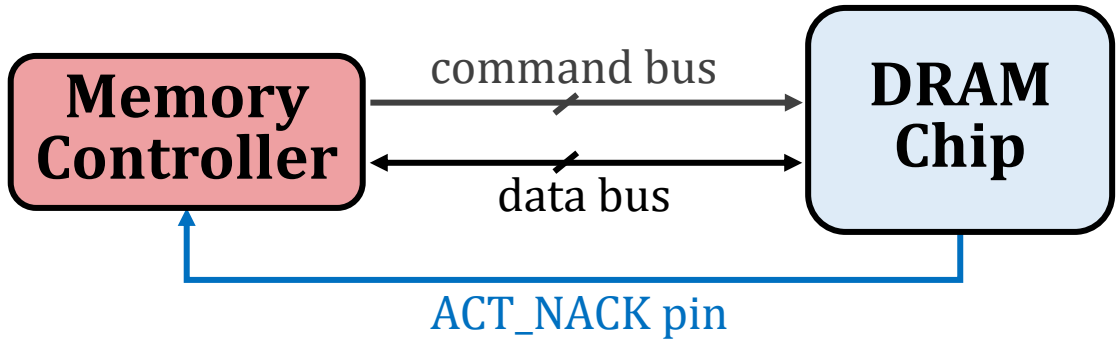
The new **scrub** command requires changes in DRAM interface and memory controller

Self-Managing DRAM (SMD)

enables autonomous in-DRAM maintenance operations

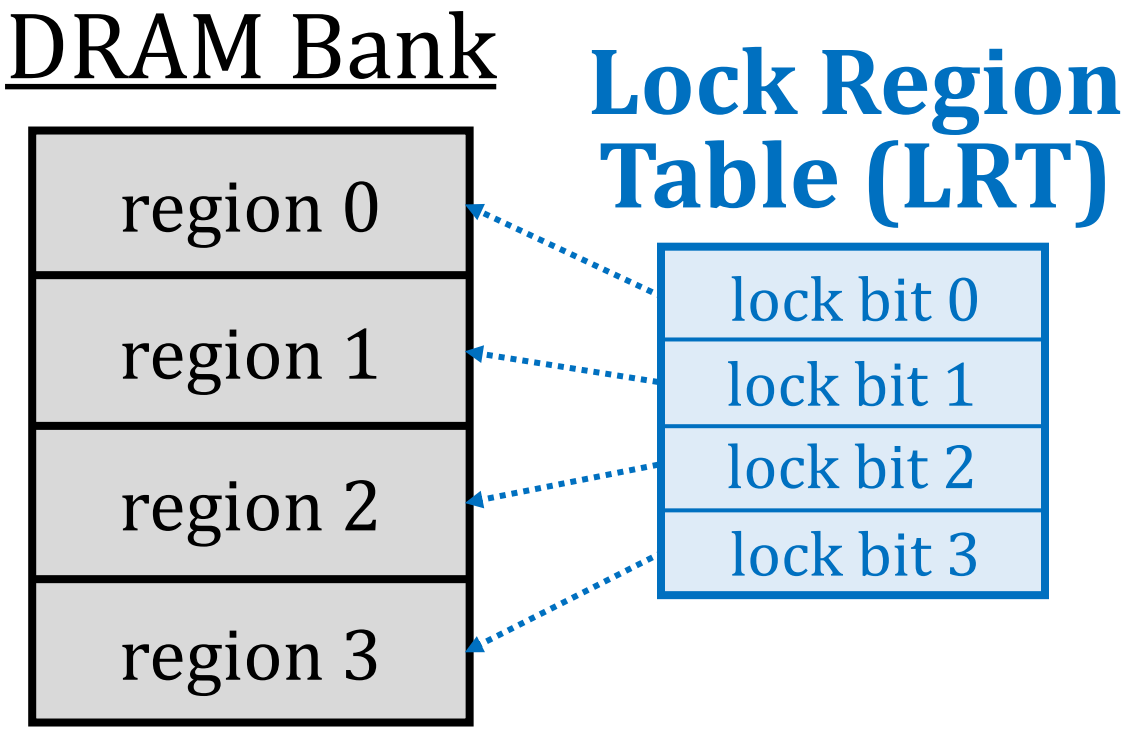
Key Idea:

Prevent the memory controller from accessing DRAM regions that are *under maintenance* by rejecting row activation (ACT) commands



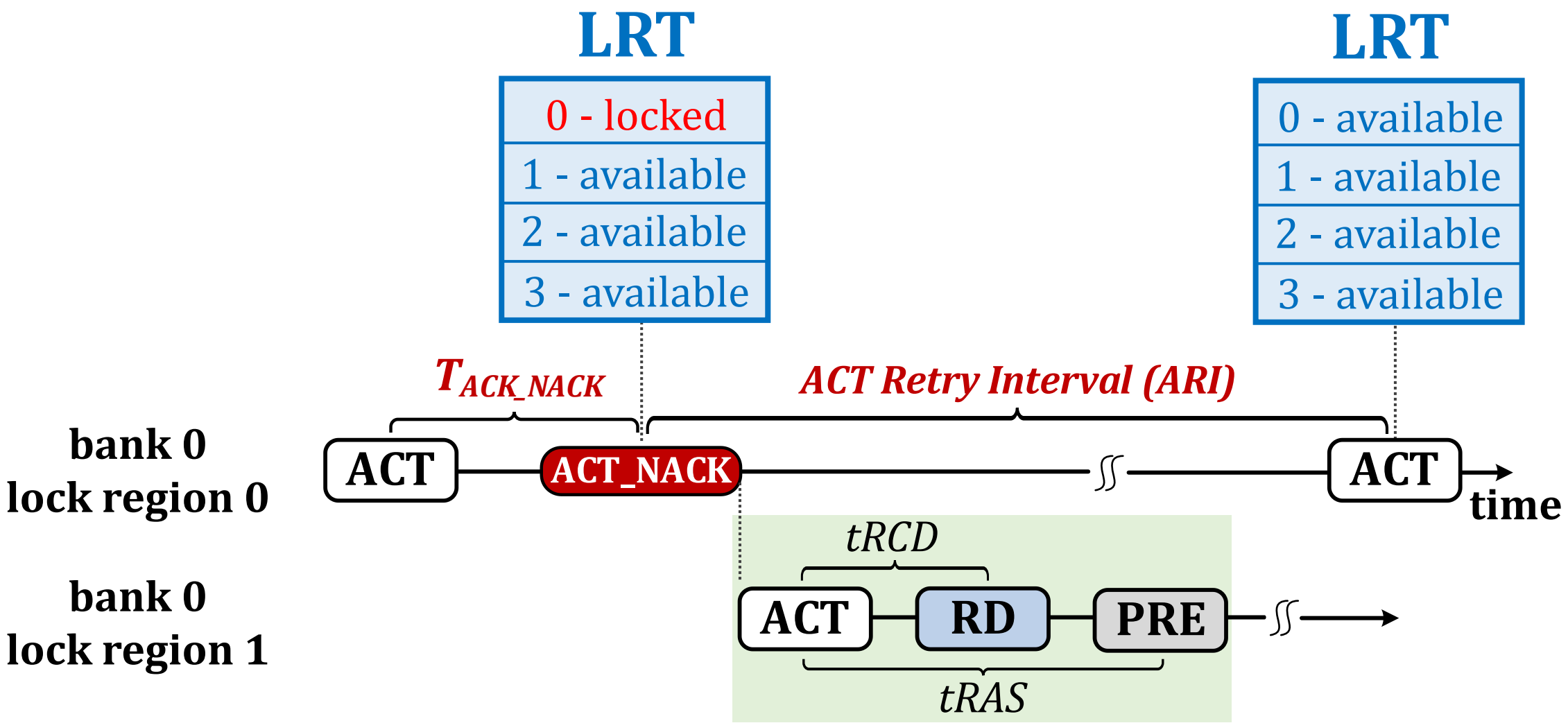
Leveraging the ability to *reject an ACT*, a maintenance operation can be implemented *completely* within a DRAM chip

SMD: DRAM Bank Architecture



- A DRAM bank is divided into configurable-size **Lock Regions**
- SMD marks regions *locked* for maintenance in the **Lock Region Table (LRT)**
- SMD rejects *any* ACT command that targets a *locked region* by sending the memory controller an **ACT_NACK** signal

SMD: High-Level Operation



SMD-Based Maintenance Mechanisms

	Fixed Rate (SMD-FR)	Variable Rate (SMD-VR)
DRAM Refresh	***uniformly*** *refreshes* ***all*** *DRAM rows with a* ***fixed*** *refresh period*	***skips*** *refreshing rows that can* ***retain their data for longer*** *than the default refresh period*

	Probabilistic (SMD-PRP)	Deterministic (SMD-DRP)
RowHammer Protection	*Performs* ***neighbor row refresh*** *with* ***a small probability*** *on every row activation*	***keeps track*** *of most* ***frequently activated*** *rows and performs* ***neighbor*** *row refresh when activation count threshold is exceeded*

	Periodic Scrubbing (SMD-MS)
Memory Scrubbing	*periodically* ***scans*** *the* ***entire*** *DRAM for errors and corrects them*

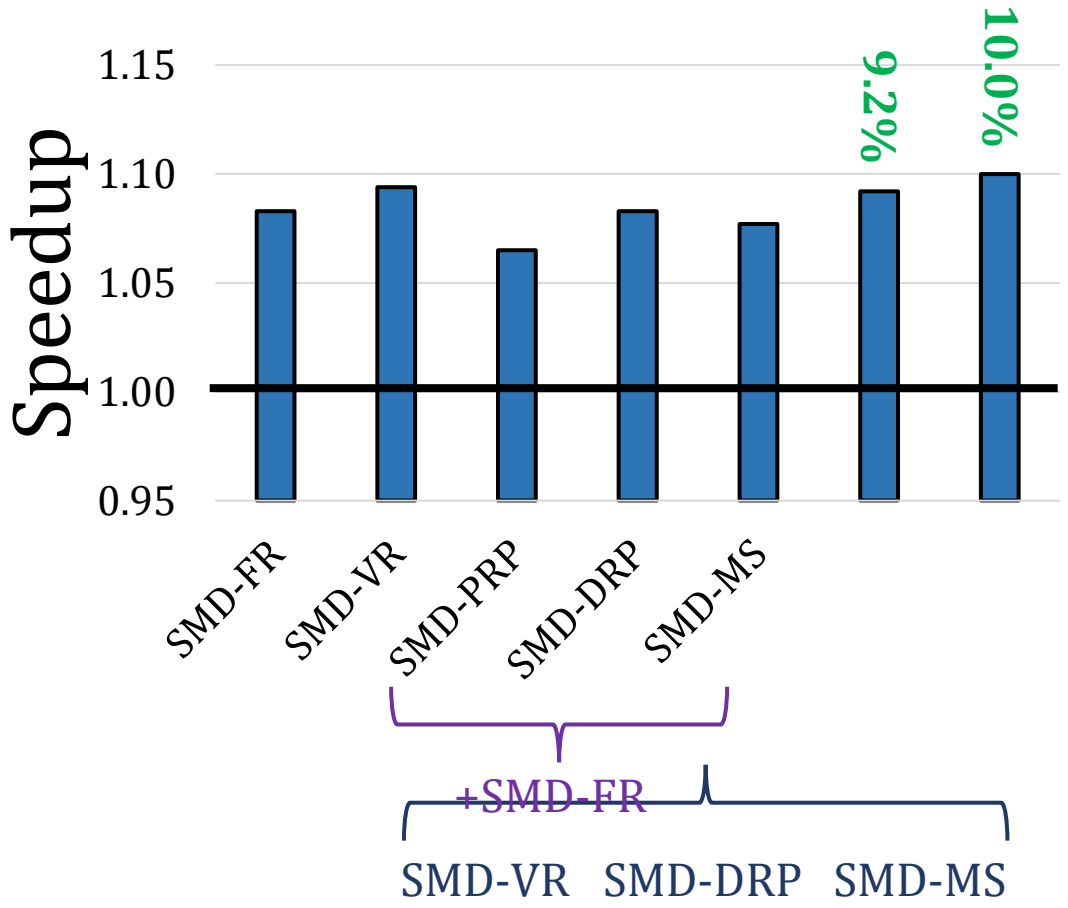
Methodology

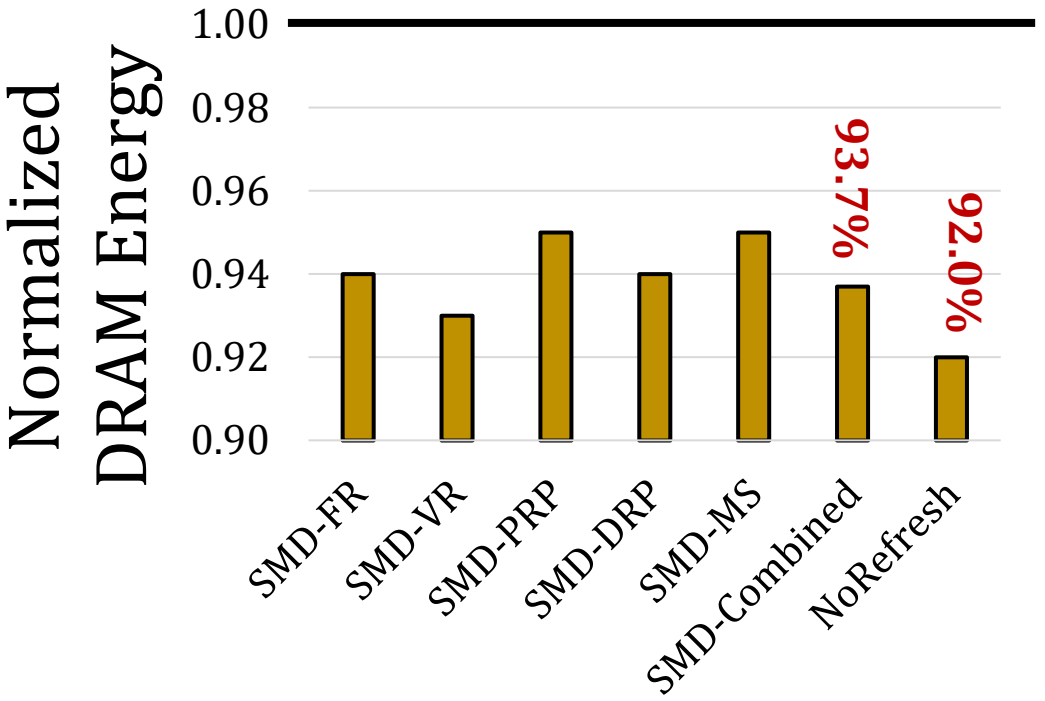
- **Simulators**
  - Ramulator [Kim+, CAL'15]
    https://github/CMU-SAFARI/ramulator
  - DRAMPower [Chandrasekar+, DSD'11]
    https://github.com/tukl-msd/DRAMPower

- **Workloads**
  - 44 single-core workloads
    SPEC CPU2006, TPC, STREAM, MediaBench
  - 60 multi-programmed four-core workloads
    By randomly choosing from single-core workloads

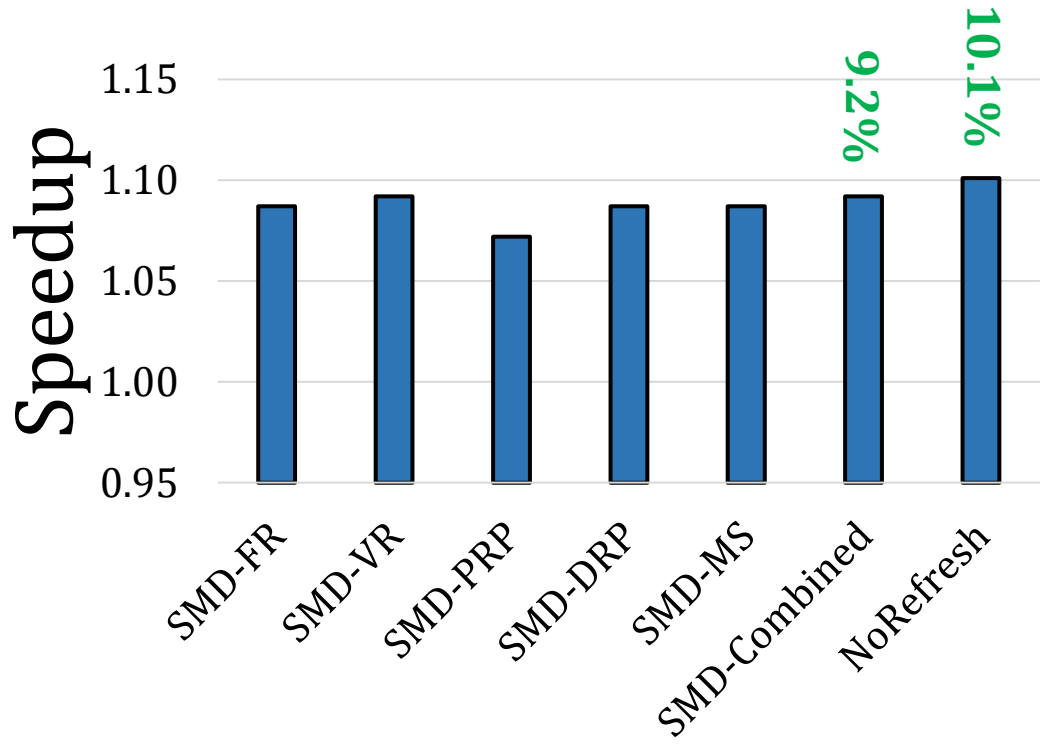
- **System Parameters**
  - 4-channel dual-rank DDR4 DRAM
  - 32ms default refresh period

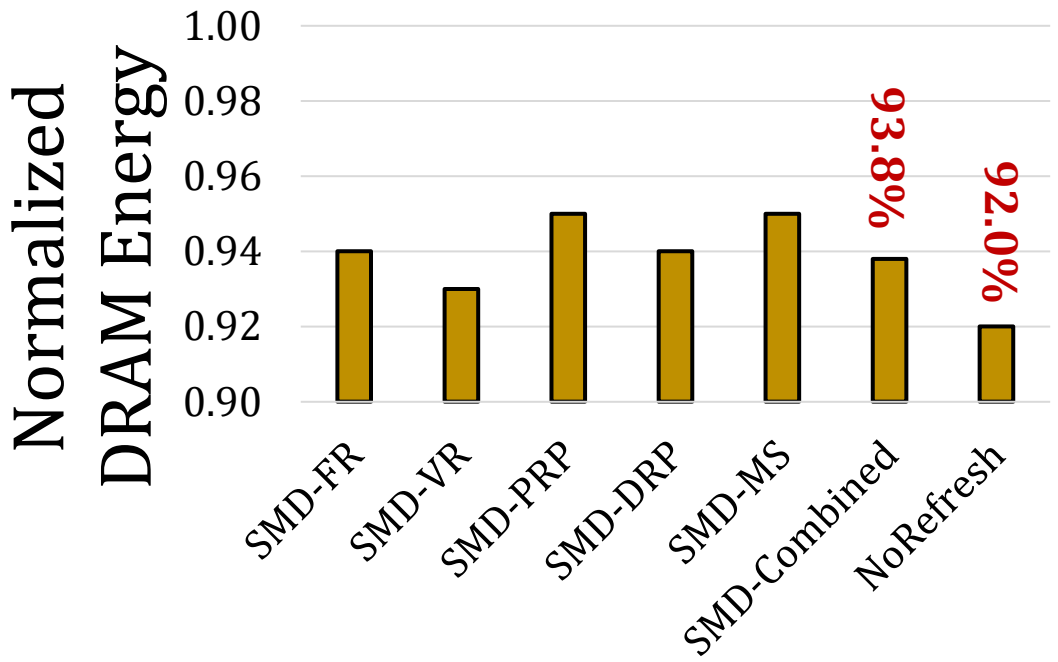
Single-Core Results





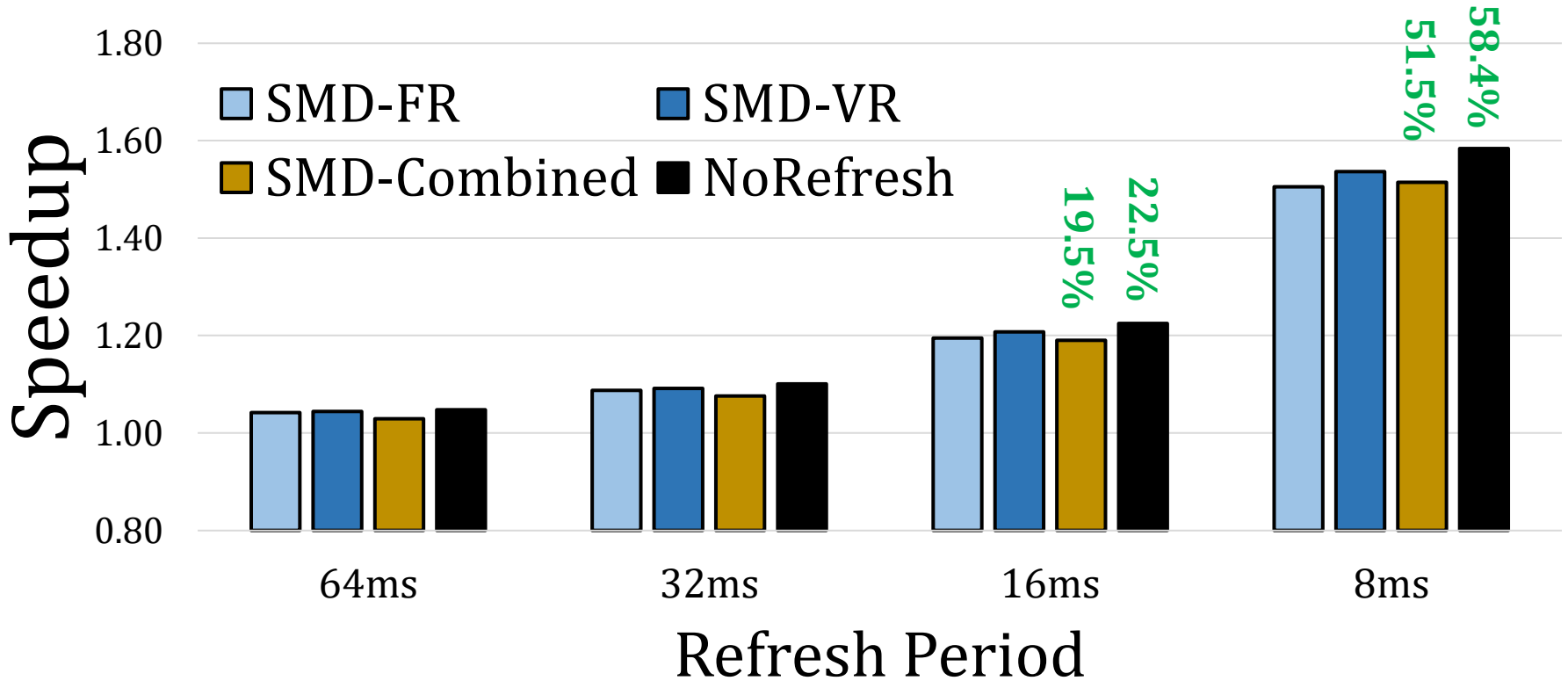
Four-Core Results





SMD-based maintenance mechanisms have **significant** *performance* and *energy efficiency* **benefits**

Sensitivity to Refresh Period



SMD-based refresh mechanisms will be even more beneficial in future DRAM chip with low refresh periods

Hardware Overhead

Interface Modifications

- A single ACT_NACK pin per DRAM chip

DRAM Chip Modifications

- Lock Region Table incurs only:
  - 32um2 **area** overhead (0.001% of a 45.4mm2 DRAM chip)
  - 0.053ns **access latency** overhead

Memory Controller Modifications

- Changes in *request scheduling* to handle ACT_NACK signals

- No further changes needed for new maintenance operations
- The memory controller no longer manages DRAM maintenance

Other Results in the Paper

- Lock region size sensitivity
- Comparison to memory controller-based RH protection
- Comparison to memory controller-based scrubbing
- SMD-DRP maximum activation threshold sensitivity
- Victim row window sensitivity

Summary

The three major DRAM maintenance operations:

- ❖Refresh
- ❖RowHammer Protection
- ❖Memory Scrubbing

Source code will be available soon:
github.com/CMU-SAFARI/SelfManagingDRAM

Implementing new **maintenance mechanisms** often requires difficult-to-realize changes

Our Goal

1. Ease the process of enabling new DRAM maintenance operations
2. Enable more efficient in-DRAM maintenance operations

Self-Managing DRAM (SMD)

Enables implementing new **in-DRAM** maintenance mechanisms
with **no further changes** in the *DRAM interface* and *memory controller*

SMD-based *refresh*, *RowHammer protection*, and *scrubbing* achieve
9.2% speedup and **6.2% lower DRAM energy** vs. conventional DRAM

Contributions

A flexible and easy-to-use FPGA-based **DRAM characterization infrastructure**

SoftMC
HPCA'17

U-TRR
MICRO'21

A new methodology for **uncovering in-DRAM RowHammer Protection Mechanisms**

Self-Managing DRAM: Enabling autonomous and efficient in-DRAM maintenance operations

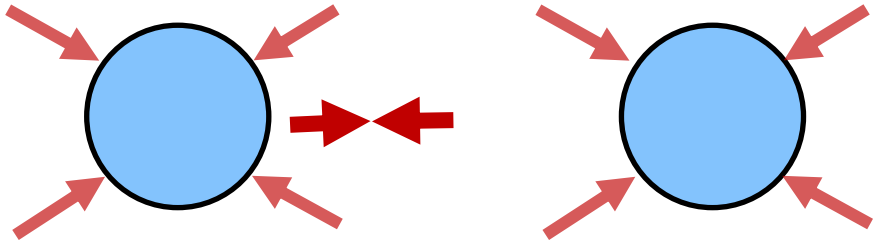
SMD
Ongoing
arXiv'22

CROW
ISCA'19

A low-cost substrate for improving DRAM performance, energy efficiency, and reliability

Scaling Challenges of DRAM Technology

DRAM Cell



DRAM Scaling Problems

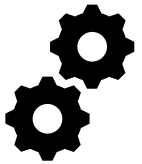
Data Integrity

Latency

System-Level Problems

Performance

Energy Efficiency



Reliability

Security

Observation

Modifying the underlying **density-optimized**

DRAM cell array structure may incur

non-negligible **area overhead**

Our Goal

1. Lower DRAM latency
2. Reduce refresh overhead
3. Improve DRAM reliability

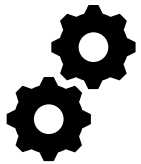
- No changes to the DRAM cell structure
- Only minimal changes to the DRAM chip

Copy-Row DRAM (CROW)

A flexible **substrate** that enables new mechanisms for improving DRAM:


Performance ✓


Energy Efficiency ✓

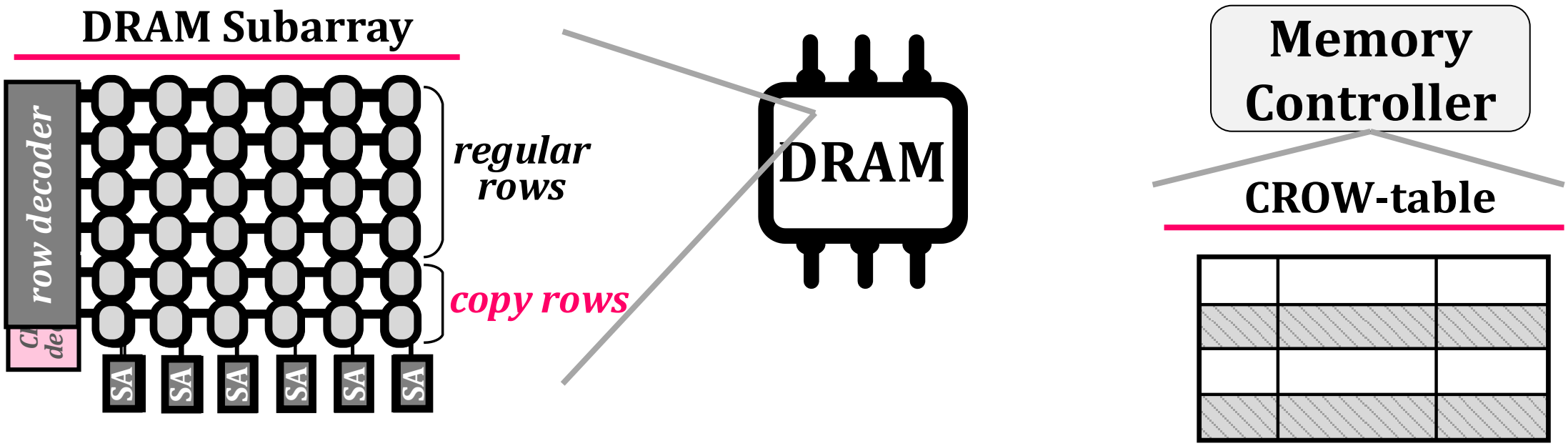

Reliability ✓


Security ✓

Key Idea:

1) efficiently **duplicate** select rows in DRAM
2) exploit **duplicated rows** in new mechanisms

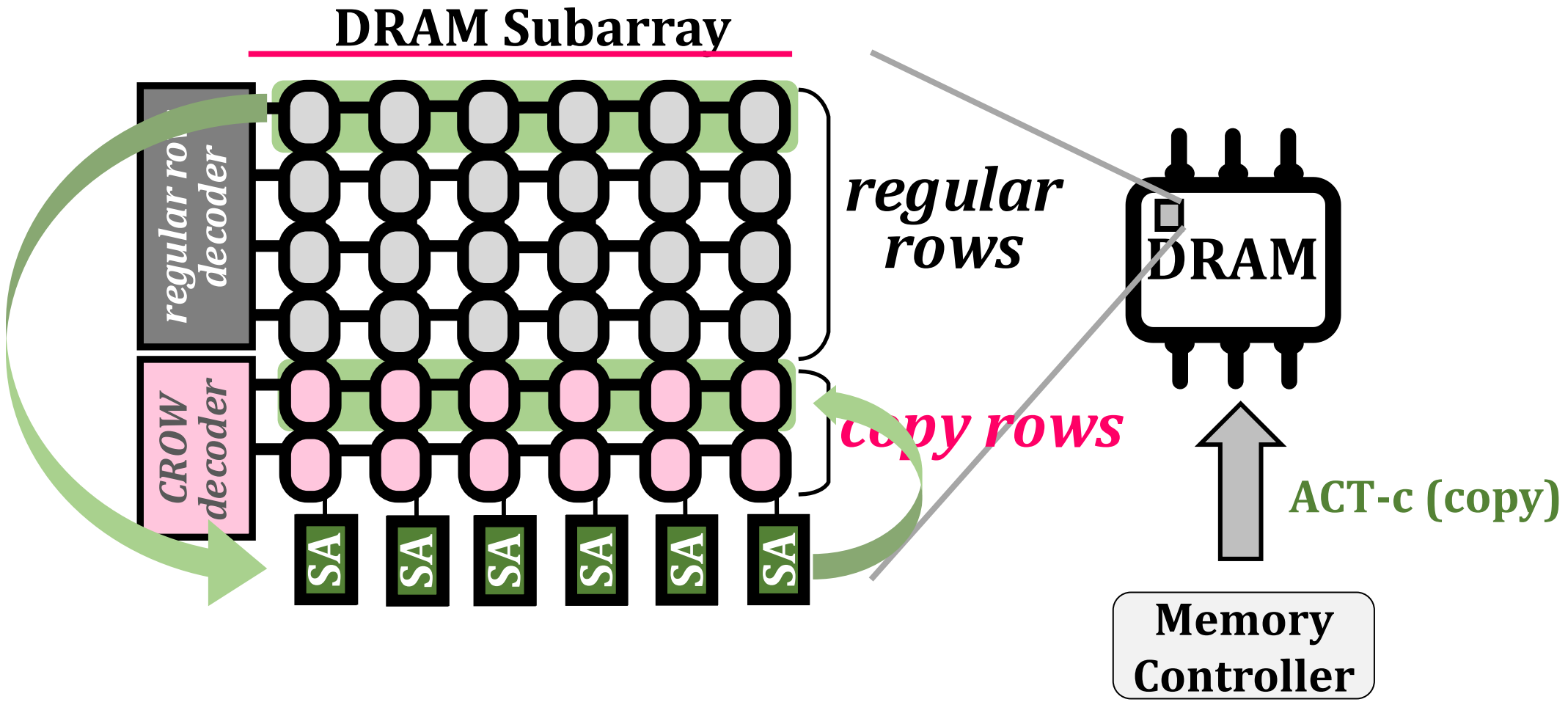
CROW: High-Level Overview



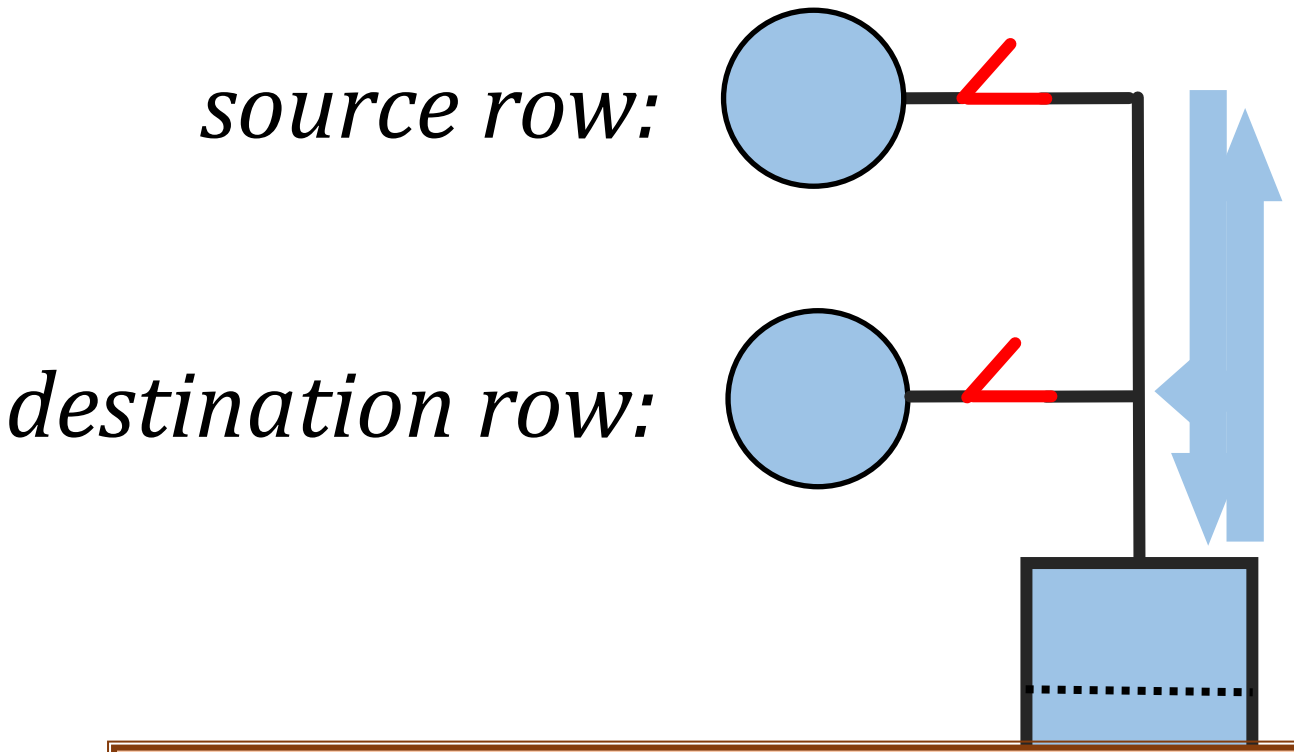
CROW enables:

1) **Row Copy:** efficiently duplicating data from a regular row to a copy row
2) **Two-row Activation:** quick access to a duplicated row

CROW Operation 1: Row Copy



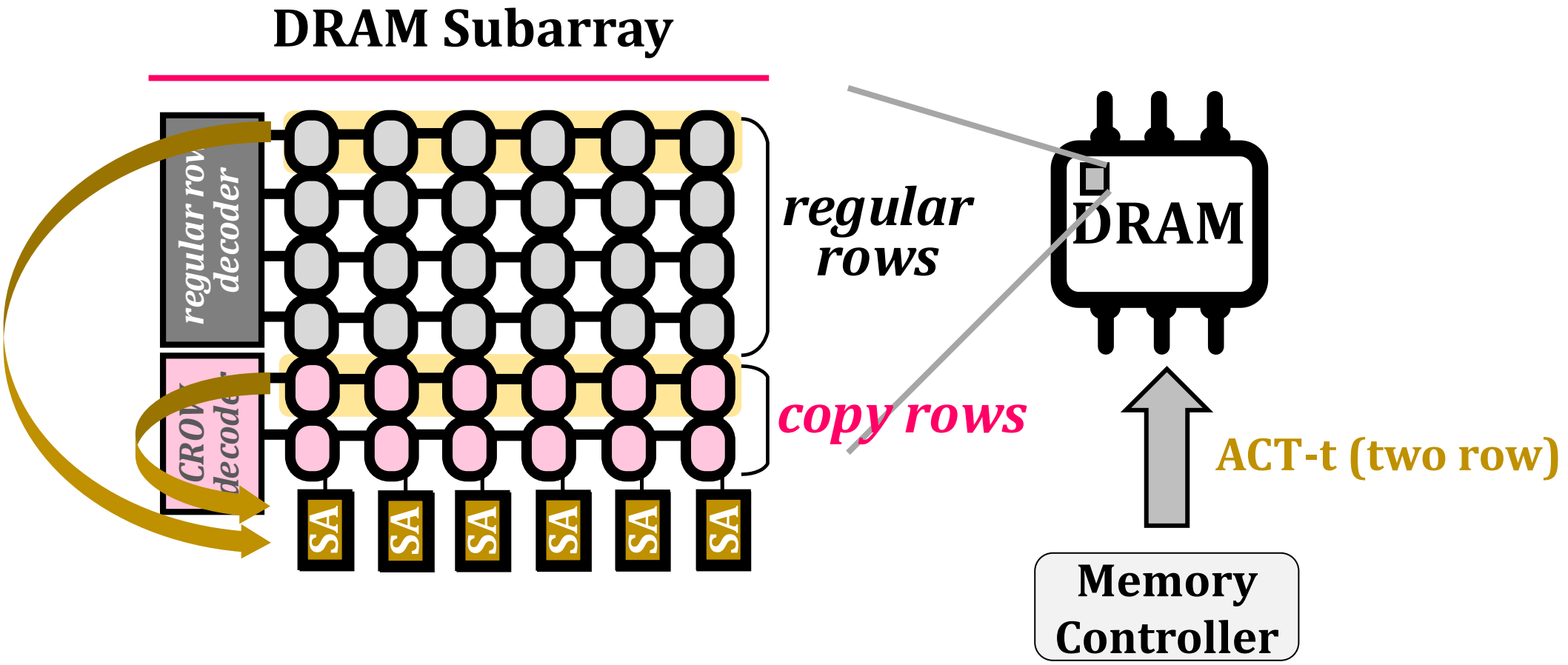
Row Copy: Steps



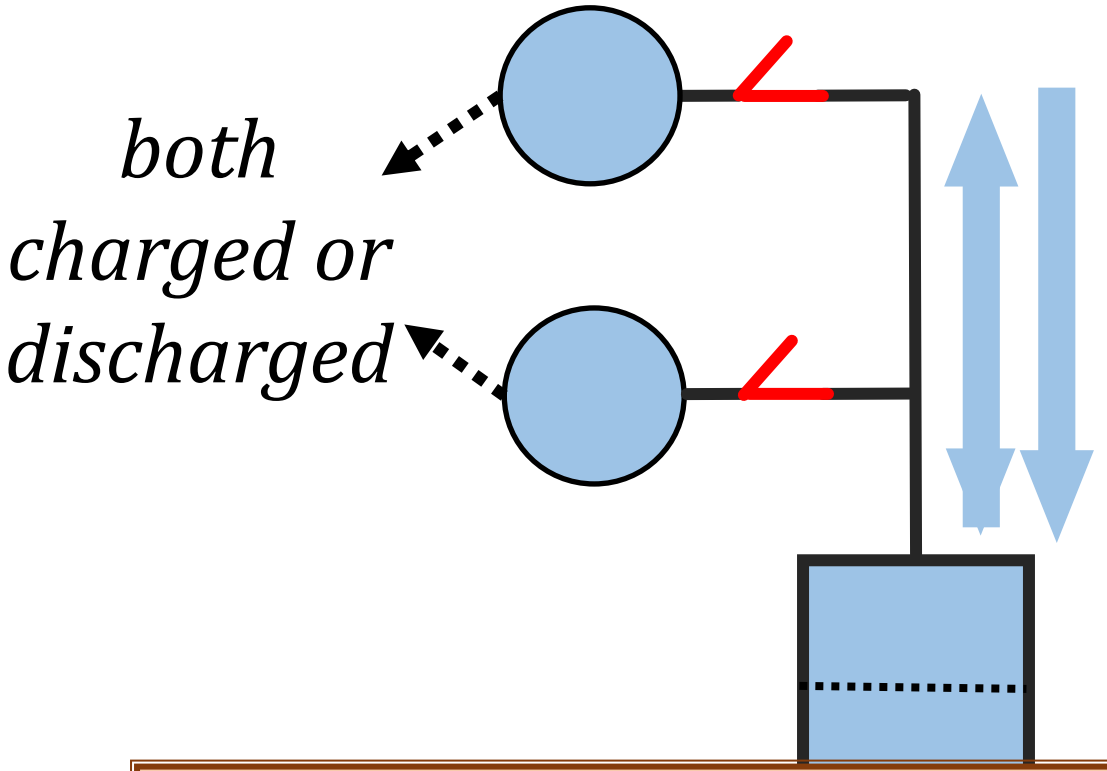
1. Activation of the source row
2. Charge sharing
3. Beginning of restoration
4. Activation of the destination row
5. Restoration of both rows

Enables quickly copying a regular row into a copy row

CROW Operation 2: Two-Row Activation



Two-Row Activation: Steps



1. Activation of two rows
2. Charge sharing **fast**
3. Restoration

Enables fast access to data that is duplicated across a regular row and a copy row

CROW-based Mechanisms

CROW-cache

CROW-ref

Mitigating RowHammer

CROW-cache

Problem: High access latency

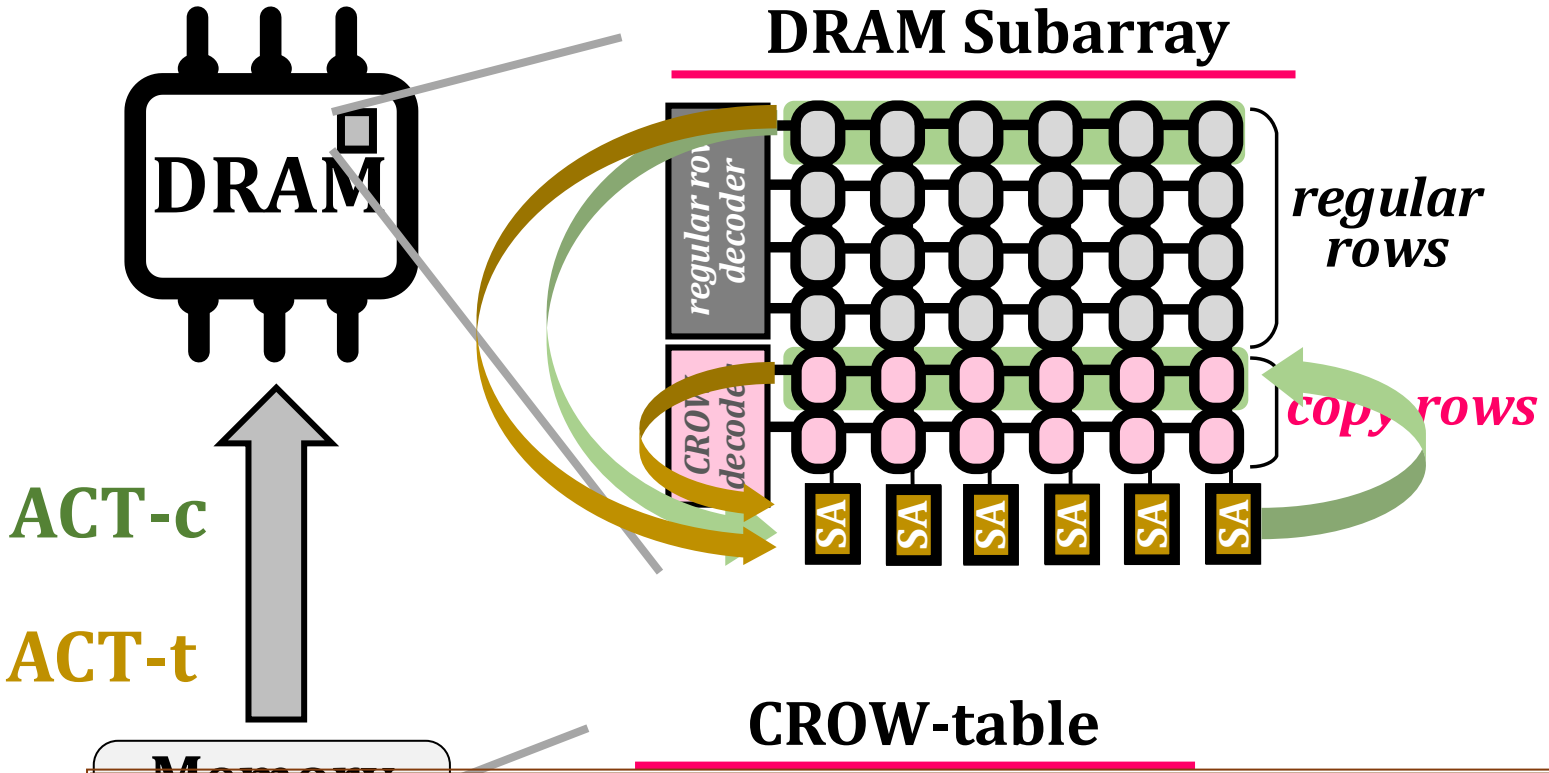
Key idea: Use copy rows to enable low-latency access to most-recently-activated regular rows in a subarray

CROW-cache combines:

- row copy → copy a newly activated regular row into a copy row
- two-row activation → activate the regular row and copy row together on the next access

Reduces activation latency by **38%**

CROW-cache Operation



Request Queue

```
load row X
[bank conflict]
load row X
```

1. CROW-table miss
2. Allocate a copy row
3. Issue ACT-c (copy)

Second activation of row X is faster

CROW-based Mechanisms

CROW-cache

CROW-ref

Mitigating RowHammer

CROW-ref

Problem: Refresh has high overheads. Weak rows lead to high refresh rate

- **weak row:** at least one of the row's cells cannot retain data correctly when refresh rate is decreased

Key idea: Safely reduce refresh rate by remapping a weak regular row to a strong copy row

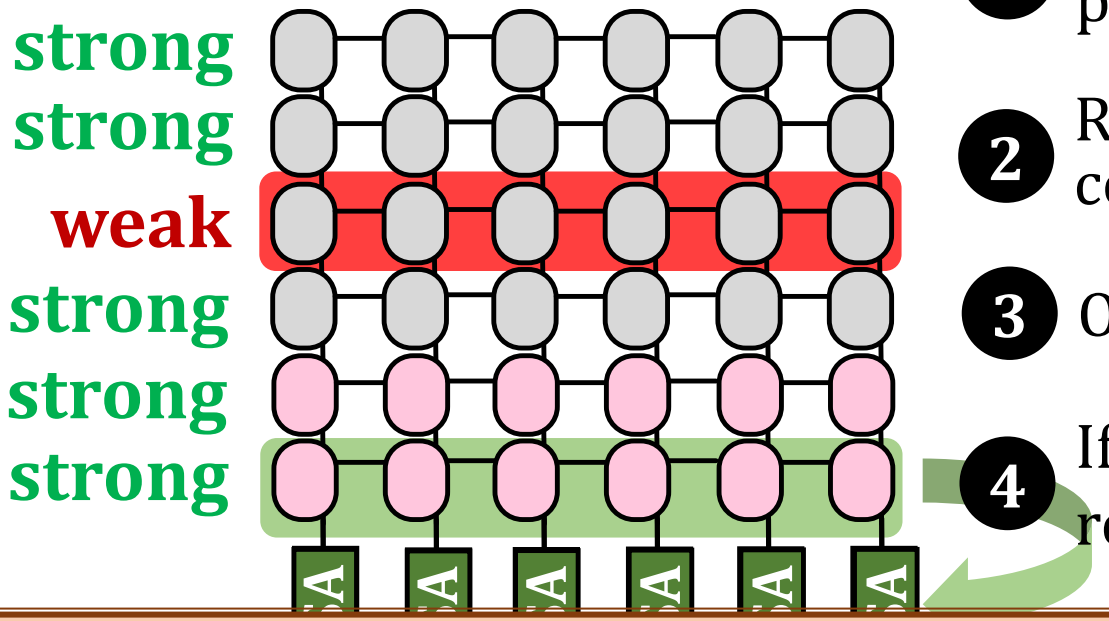
CROW-ref uses:

- row copy → copy a weak regular row to a strong copy row

CROW-ref **eliminates more than half of the refresh** requests

CROW-ref Operation

Retention Time Profiler



1. Perform retention time profiling
2. Remap weak rows to strong copy rows
3. On ACT, check the CROW-table
4. If remapped, activate a copy row

How many weak rows exist in a DRAM chip?

Identifying Weak Rows

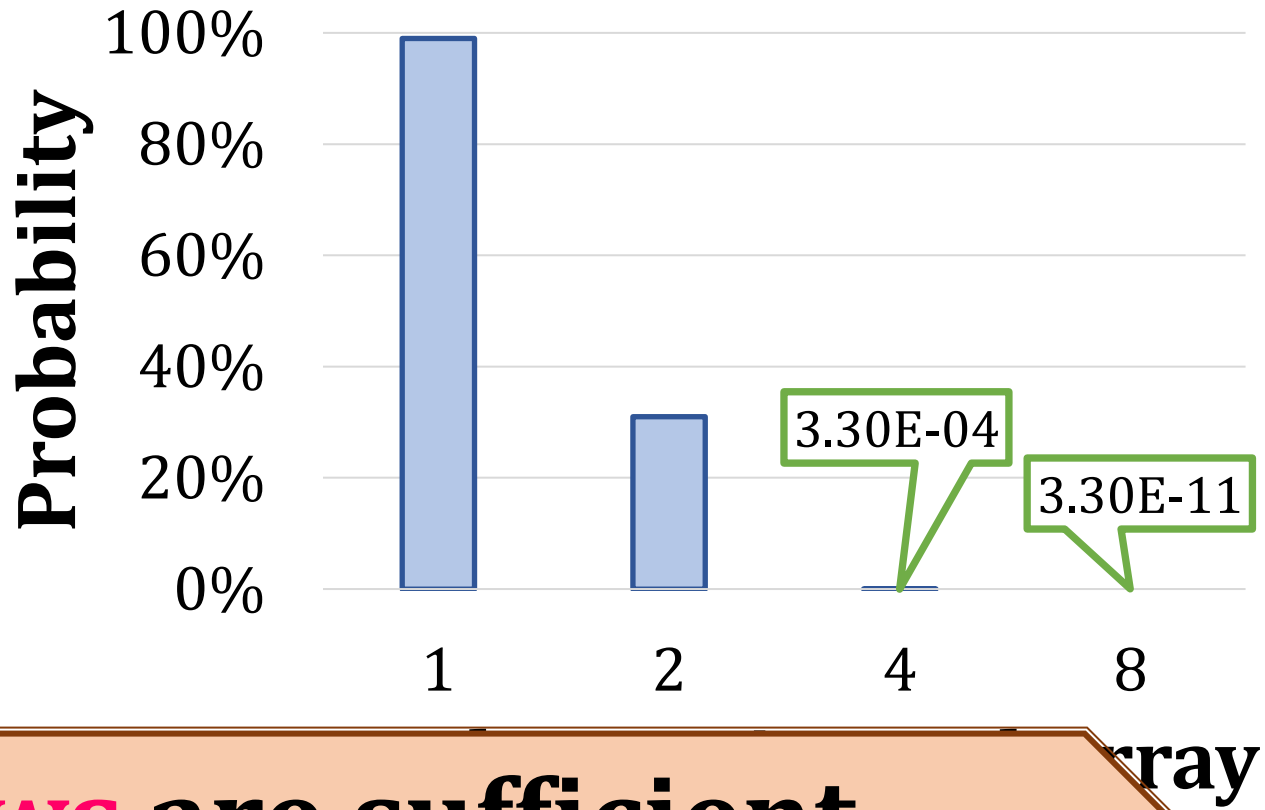
Weak cells are rare *[Liu+, ISCA'13]*

weak cell: retention < 256ms

~$1000/2^{38}$ (32 GiB) failing cells

DRAM Retention Time Profiler

- REAPER *[Patel+, ISCA'17]*
  PARBOR *[Khan+, DSN'16]*
  AVATAR *[Qureshi+, DSN'15]*



A few copy rows are sufficient to halve the refresh rate

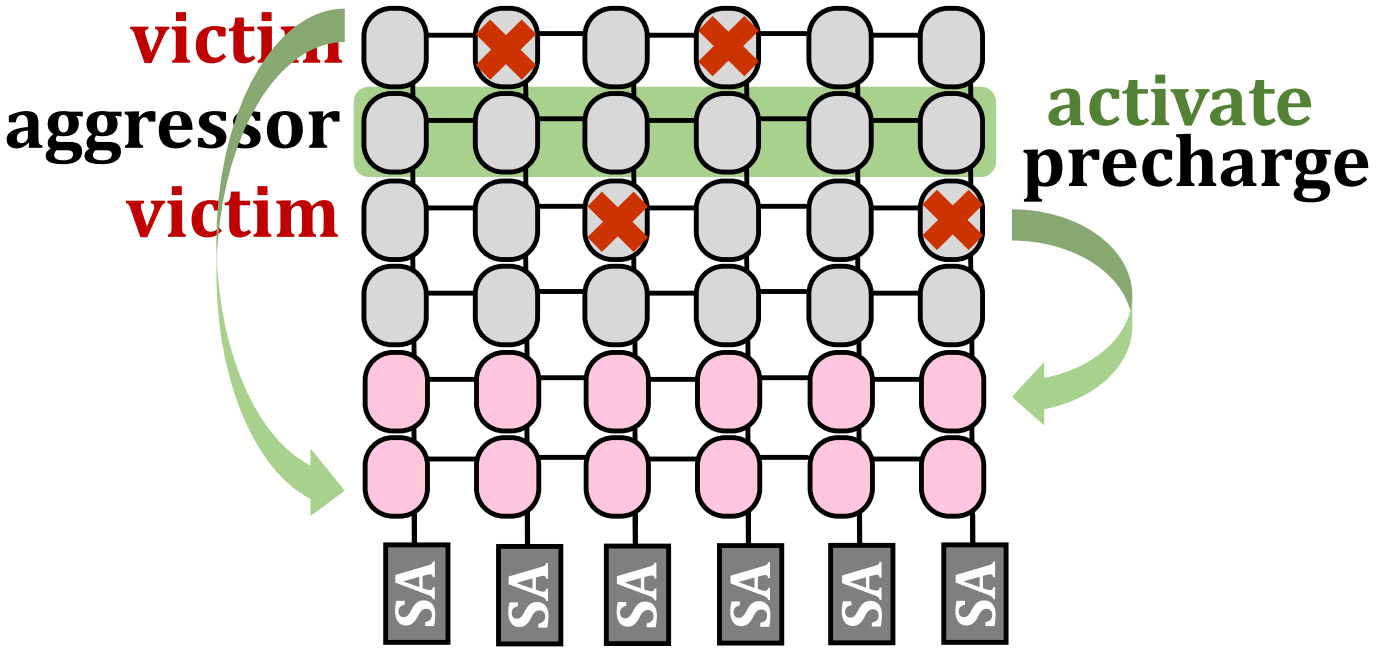
CROW-based Mechanisms

CROW-cache

CROW-ref

Mitigating RowHammer

Mitigating RowHammer



Key idea: remap victim rows to copy rows

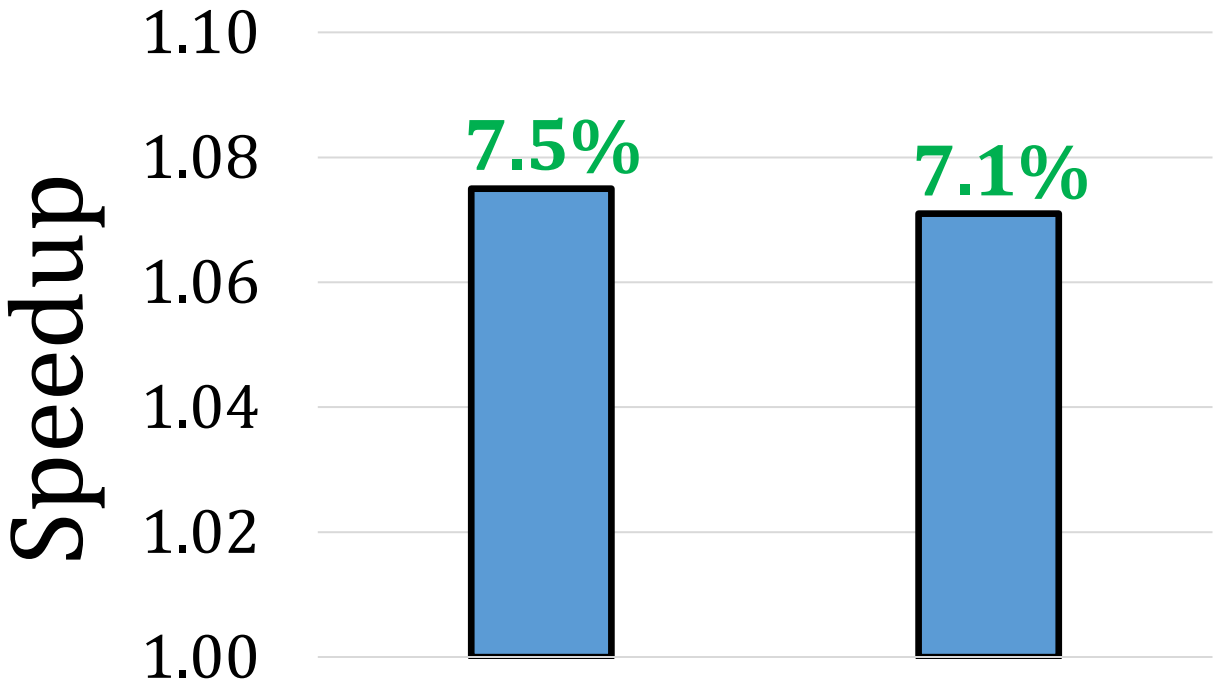
Methodology

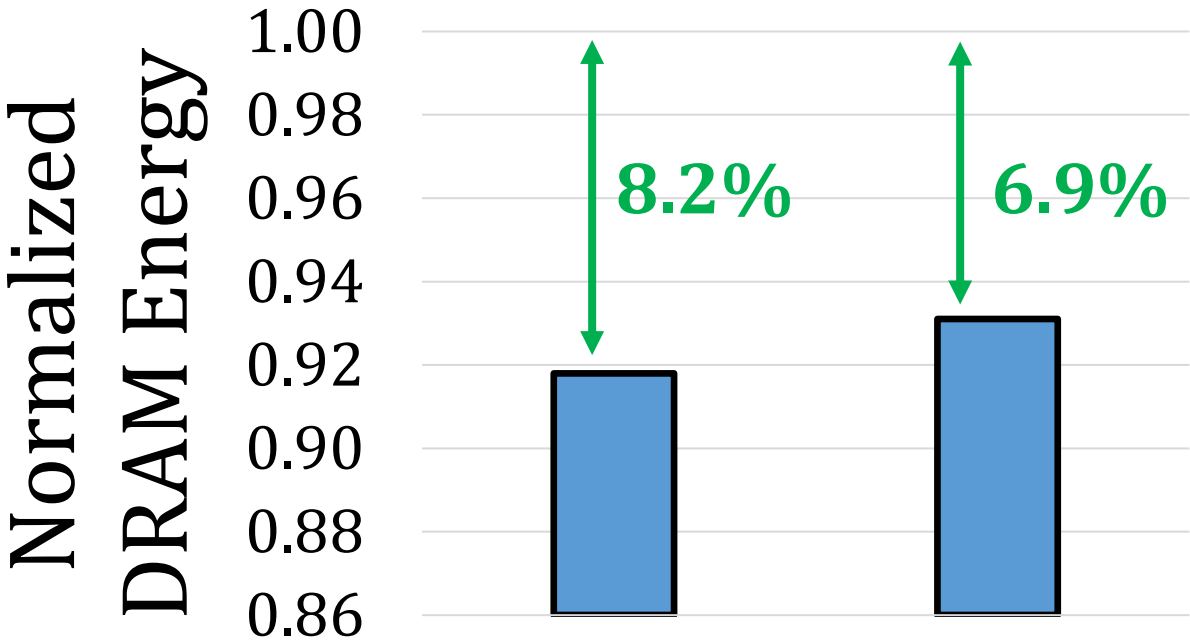
- **Simulator**
  - DRAM Simulator (Ramulator *[Kim+, CAL'15]*)
    https://github.com/CMU-SAFARI/ramulator

- **Workloads**
  - 44 single-core workloads
    - *SPEC CPU2006, TPC, STREAM, MediaBench*
  - 160 multi-programmed four-core workloads
    - *By randomly choosing from single-core workloads*
  - Execute at least 200 million representative instructions per core

- **System Parameters**
  - 1/4 core system with 8 MiB LLC
  - LPDDR4 main memory
  - 8 copy rows per 512-row subarray

CROW-cache Results

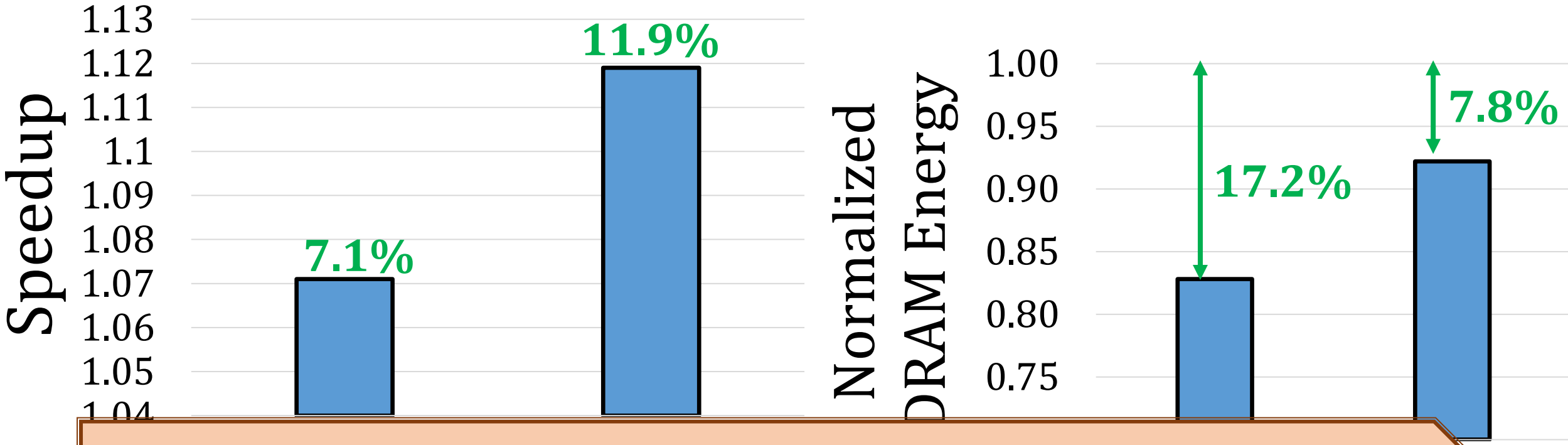




-core

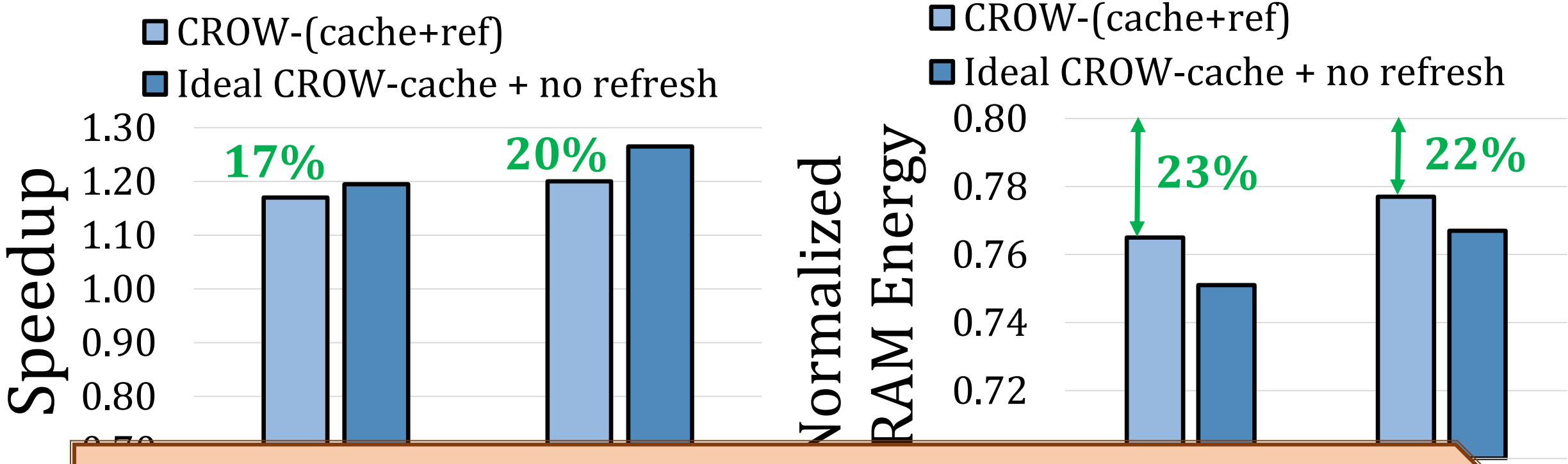
CROW-cache **improves** single-/four-core **performance** and **energy**

CROW-ref Results



CROW-ref significantly reduces the performance and energy overhead of DRAM refresh

Combining CROW-cache and CROW-ref



CROW-(cache+ref) provides more performance and DRAM energy benefits than each mechanism alone

Hardware Overhead

For 8 copy rows and 16 GiB DRAM:

- 0.5% DRAM chip area
- 1.6% DRAM capacity
- 11.3 KiB memory controller storage

CROW is a low-cost substrate

Other Results in the Paper

- Performance and energy sensitivity to:
  - Number of copy-rows per subarray
  - DRAM chip density
  - Last-level cache capacity

- CROW-cache with prefetching

- CROW-cache compared to other in-DRAM caching mechanisms:
  - TL-DRAM *[Lee+, HPCA'13]*
  - SALP *[Kim+, ISCA'12]*

Summary

Copy-Row DRAM (CROW)

- Introduces copy rows into a subarray
- The benefits of a copy row:
  - Efficiently duplicating data from regular row to a copy row
  - Quick access to a duplicated row
  - Remapping a regular row to a copy row

Source code available:
github.com/CMU-SAFARI/CROW

Use cases of CROW:

- CROW-cache & CROW-ref
- Mitigating RowHammer
- We hope CROW enables many other use cases going forward

Contributions

A flexible and easy-to-use FPGA-based **DRAM characterization infrastructure**

SoftMC
HPCA'17

U-TRR
MICRO'21

A new methodology for **uncovering in-DRAM RowHammer Protection Mechanisms**

Self-Managing DRAM: Enabling autonomous and efficient in-DRAM maintenance operations

SMD
Ongoing
arXiv'22

CROW
ISCA'19

A low-cost substrate for improving DRAM performance, energy efficiency, and reliability

Future Research Directions

- **Deeper DRAM Characterization**
  - Analyzing cell characteristics of new devices
  - Impact of aging
  - Low temperature operation
- **Extending SoftMC**
  - Supporting other DRAM and NVM standards
- **Improving RowHammer Attacks & Defenses**
  - Studying the security properties of RowHammer protection mechanisms
- **New DRAM Maintenance Mechanisms**
  - Profiling-based maintenance operations
  - Memory controller and in-DRAM processing interoperability
- **Exploiting in-DRAM Data Movement**
  - More mechanisms that exploit low-overhead in-DRAM data migration

Improving DRAM Performance, Reliability, and Security by Rigorously Understanding Intrinsic DRAM Operation

Hasan Hassan

SAFARI Live Seminar

15 September 2022

ETH zürich

SAFARI