

# Scalable and Low Overhead DRAM Read Disturbance Mitigation

**Abdullah Giray Yağlıkçı**  
(on behalf of Ataberk Olgun)  
The Future of Memory and Storage  
August 6, 2024



**SAFARI**

**ETH zürich**

# Brief Self Introduction



- Abdullah Giray Yaglikci
  - Researcher @ SAFARI Research Group since August 2016
    - ETH Zurich (Feb 2018 – ongoing)
    - Intel Labs (Aug 2017 – Feb 2018)
    - Carnegie Mellon University (Aug 2016 – Aug 2017)
  - Defended my PhD thesis, advised by Onur Mutlu, in April 2024
  - <https://agyaglikci.github.io/>
  - [agirayyaglikci@gmail.com](mailto:agirayyaglikci@gmail.com) (Best way to reach me)
  - <https://safari.ethz.ch>
- **Research interests:**
  - Computer architecture, hardware security
  - Memory and storage systems
  - Hardware security, safety, reliability, performance, availability, fairness, energy efficiency
  - Hardware/software cooperation
  - ...

# Papers in This Talk

- Ataberk Olgun, Yahya Can Tugrul, Nisa Bostanci, Ismail Emir Yuksel, Haocong Luo, Steve Rhyner, Abdullah Giray Yaglikci, Geraldo F. Oliveira, and Onur Mutlu,  
**"ABACuS: All-Bank Activation Counters for Scalable and Low Overhead RowHammer Mitigation"** *Proceedings of the 33rd USENIX Security Symposium (USENIX Security)*, Philadelphia, PA, USA, August 2024.  
[arXiv version] [ABACuS Source Code (Officially Artifact Evaluated with All Badges)]  
*Officially artifact evaluated as available, functional, and reproduced.*
- F. Nisa Bostanci, Ismail Emir Yuksel, Ataberk Olgun, Konstantinos Kanellopoulos, Yahya Can Tugrul, A. Giray Yaglikci, Mohammad Sadrosadati, and Onur Mutlu,  
**"CoMeT: Count-Min-Sketch-based Row Tracking to Mitigate RowHammer at Low Cost"** *Proceedings of the 30th International Symposium on High-Performance Computer Architecture (HPCA)*, April 2024.  
[Slides (pptx) (pdf)] [arXiv version]  
[CoMeT Source Code (Officially Artifact Evaluated with All Badges)]  
*Officially artifact evaluated as available, reviewed and reproducible.*

# Memory & Generative AI (I)

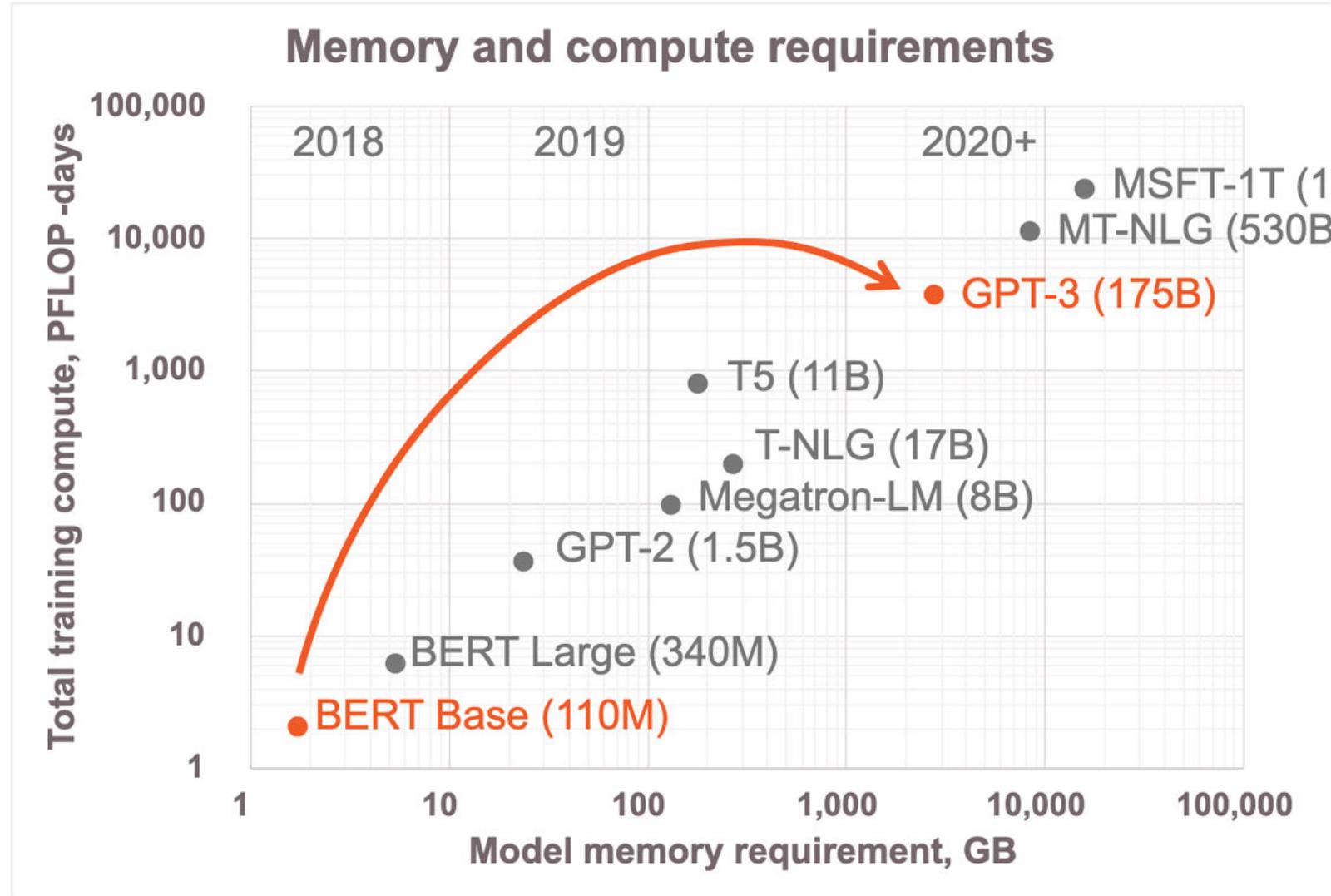
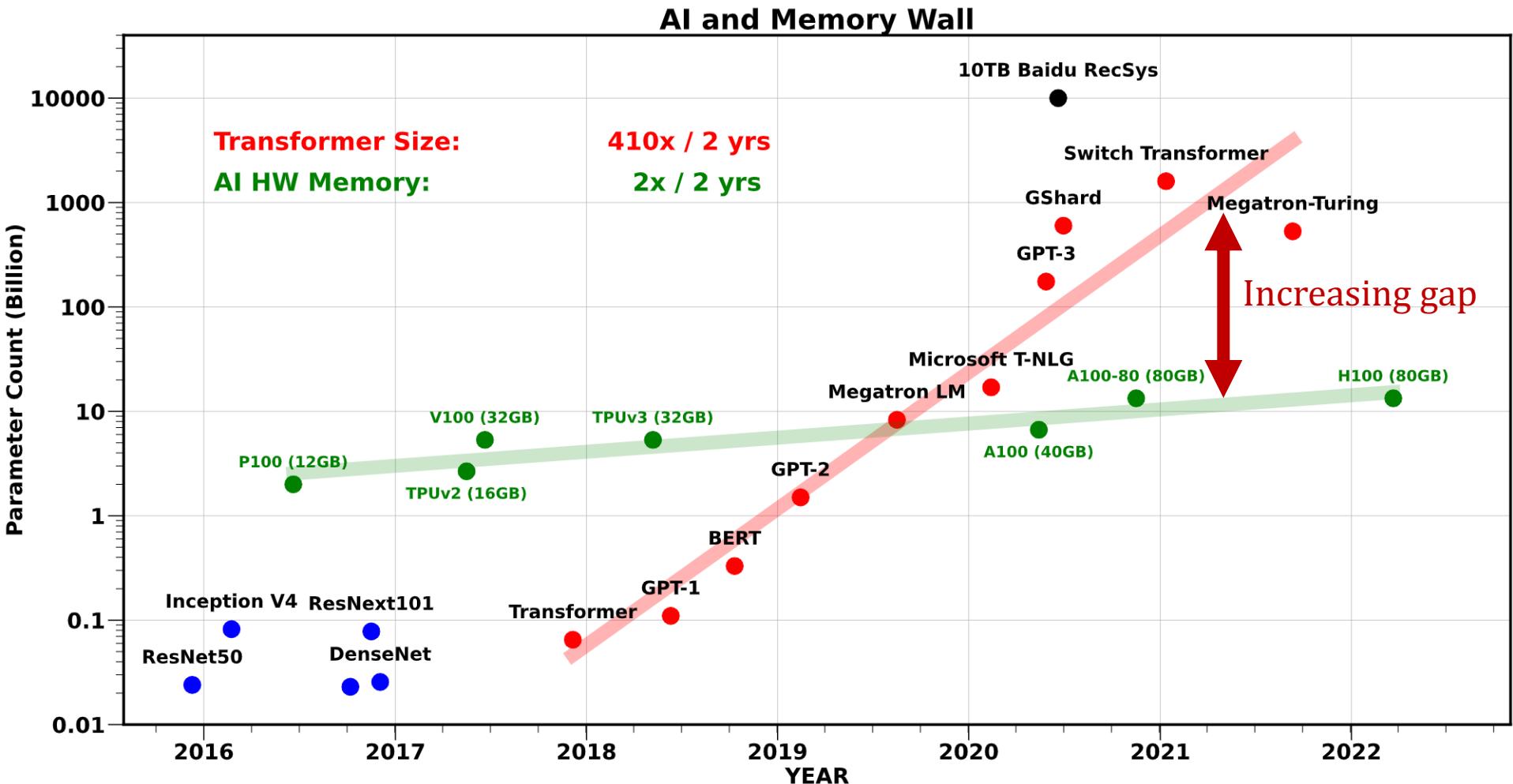


Image Source: <https://www.cerebras.net/blog/cerebras-architecture-deep-dive-first-look-inside-the-hw/sw-co-design-for-deep-learning>

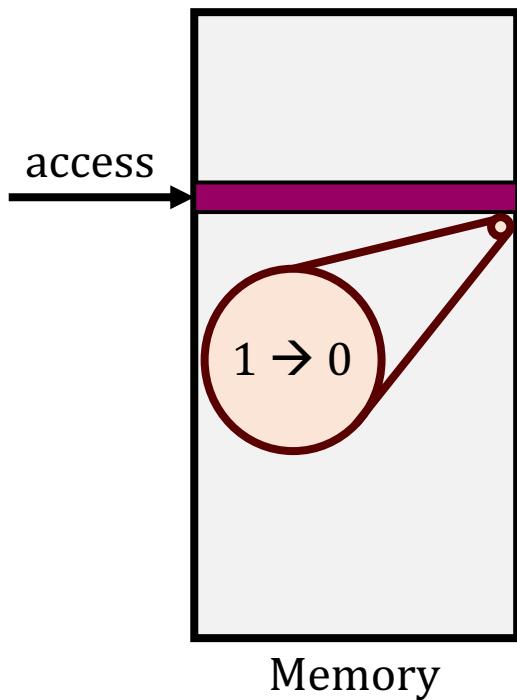
# Memory & Generative AI (II)



Gholami A, Yao Z, Kim S, Mahoney MW, Keutzer K. AI and Memory Wall. RiseLab Medium Blog Post,  
University of California Berkeley, 2021, March 29.

# Memory Isolation

- A memory access should **not have unintended side effects** on data stored in **other addresses**
- A fundamental property for **robustness** (safety, security, and reliability)

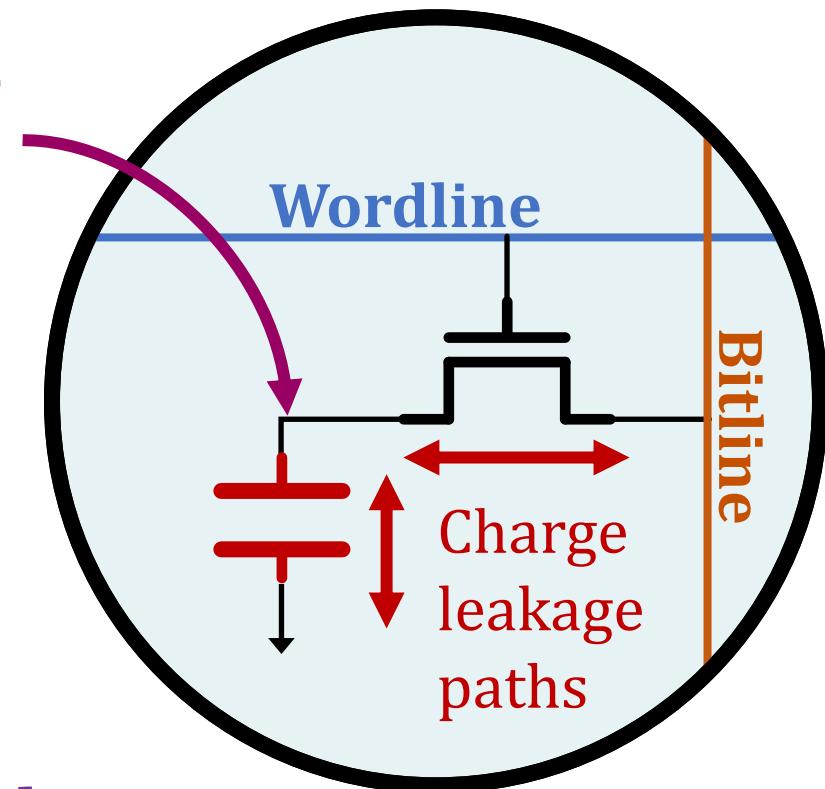


-  **Data loss or corruption**
-  Compromise **application correctness**
-  **Leak** private information
-  **Take over** a computer

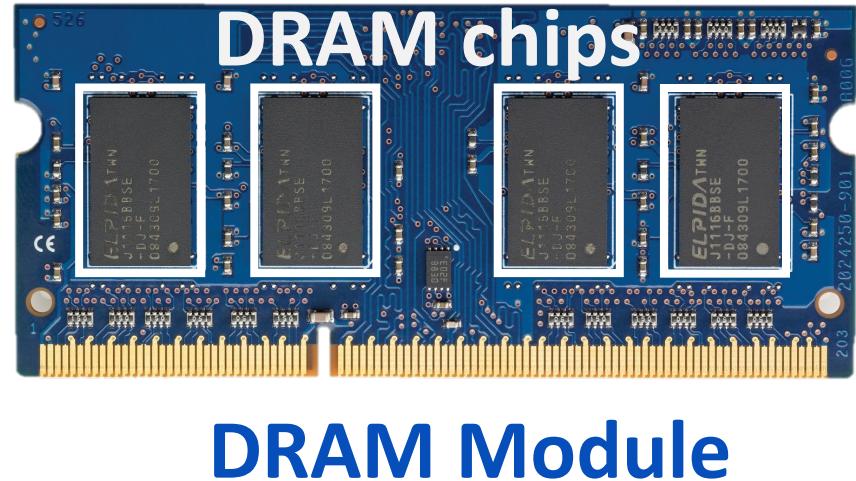
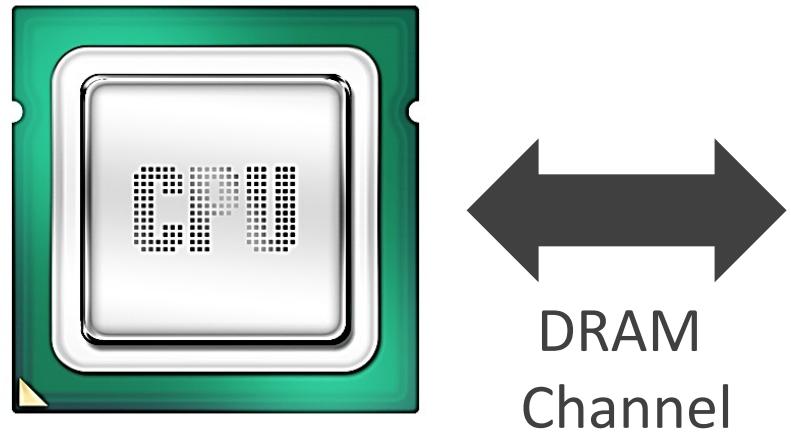
Memory isolation is **difficult in modern memory chips**

# Read Disturbance in Modern Memory Chips

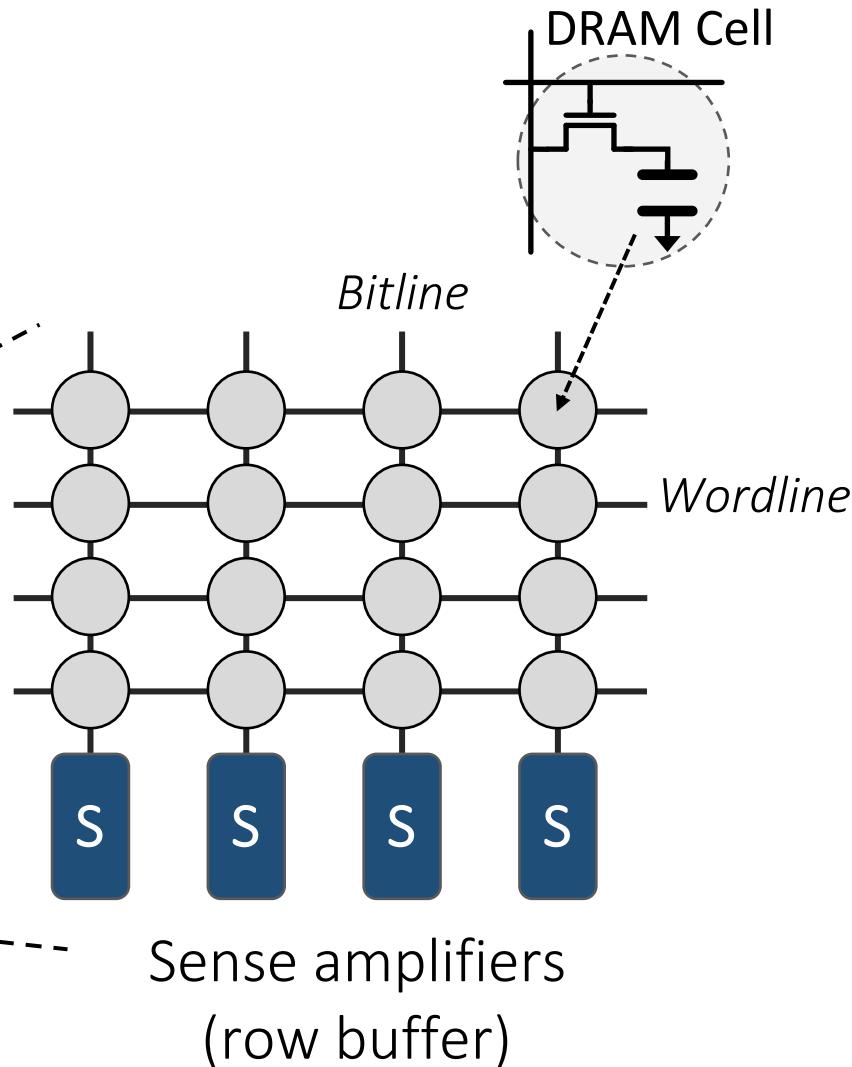
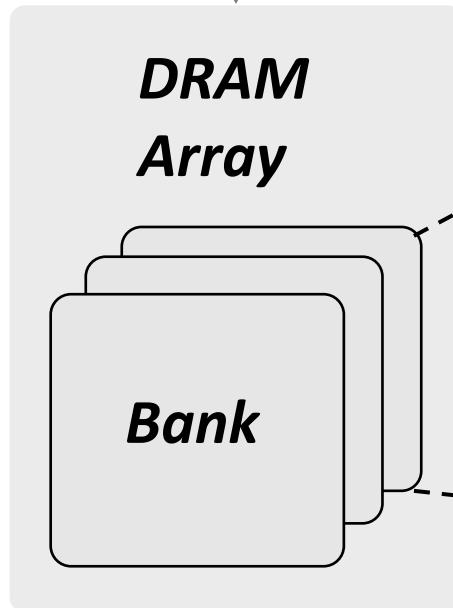
- Prevalent memory technology:  
Dynamic Random Access Memory (DRAM)
- DRAM stores **data** in the form of  
**electrical charge** on a capacitor
- DRAM **leaks charge** over time  
and needs **periodic refresh**
- **DRAM Read Disturbance:**  
Accessing a DRAM cell disturbs  
other **physically nearby cells**  
and exacerbates their **charge leakage**



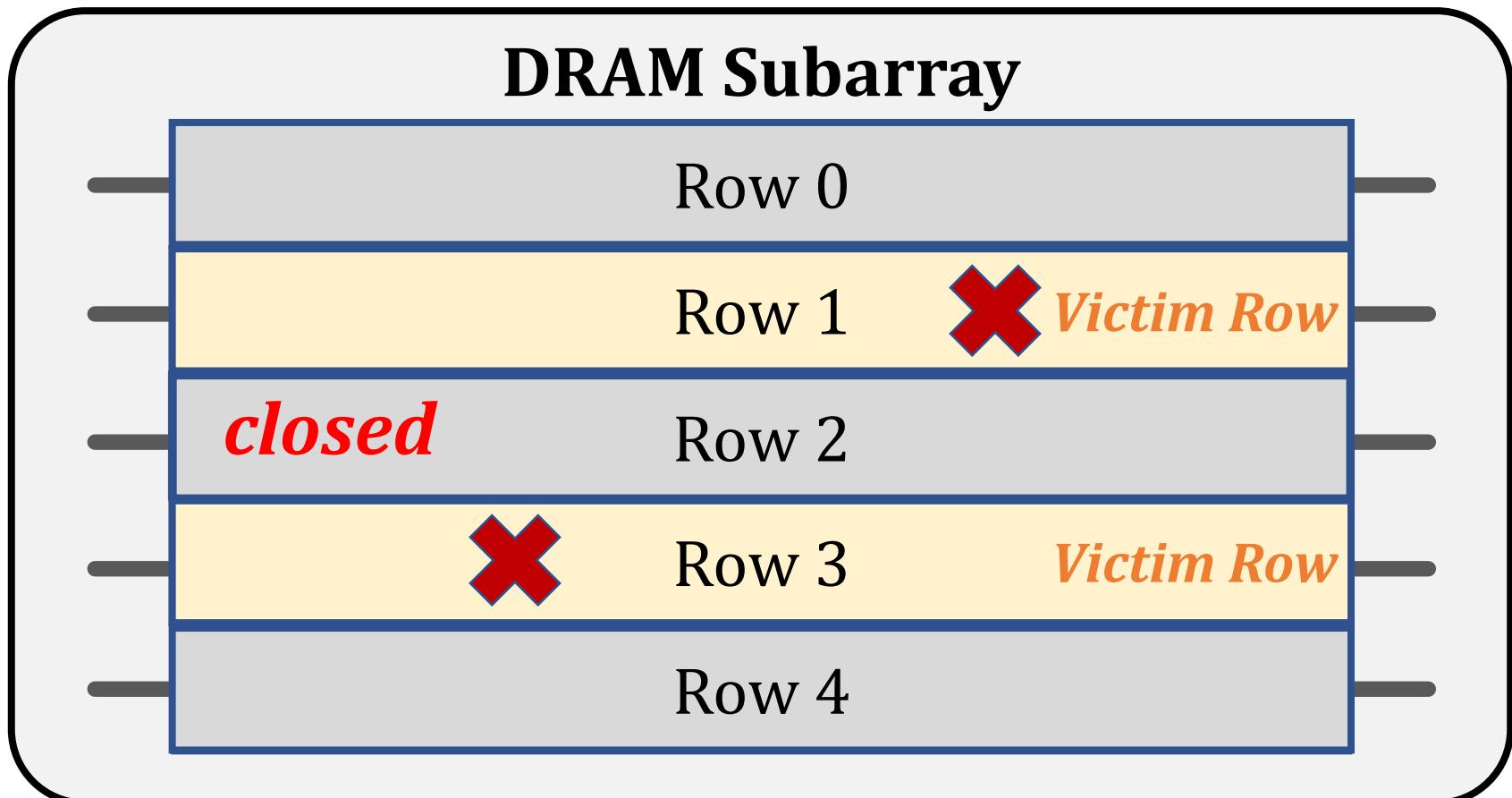
# High-Level DRAM Organization



# DRAM Chip Internals



# RowHammer: An Example of DRAM Read Disturbance



Repeatedly **opening** (activating) and **closing** (precharging) a DRAM row causes **RowHammer bitflips** in nearby cells and breaks **memory isolation**

# One Can Take Over an Otherwise-Secure System

## Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors

*Abstract. Memory isolation is a key property of a reliable and secure computing system — an access to one memory address should not have unintended side effects on data stored in other addresses. However, as DRAM process technology*

[Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors](#)  
(Kim et al., ISCA 2014)

## Project Zero

[Exploiting the DRAM rowhammer bug to gain kernel privileges](#) (Seaborn, 2015)

News and updates from the Project Zero team at Google

Induce bit flips in page table entries (PTEs).

Gain write access to its own page table,  
and hence gain read-write access to all of physical memory.

Exploiting the DRAM rowhammer bug to gain kernel privileges

# More Security Implications (I)

**"We can gain unrestricted access to systems of website visitors."**

www.iaik.tugraz.at ■

Not there yet, but ...



ROOT privileges for web apps!



29

Daniel Gruss (@lavados), Clémentine Maurice (@BloodyTangerine),  
December 28, 2015 — 32c3, Hamburg, Germany

Rowhammer.js: A Remote Software-Induced Fault Attack in JavaScript (DIMVA'16)

# More Security Implications (II)

**“Can gain control of a smart phone deterministically”**

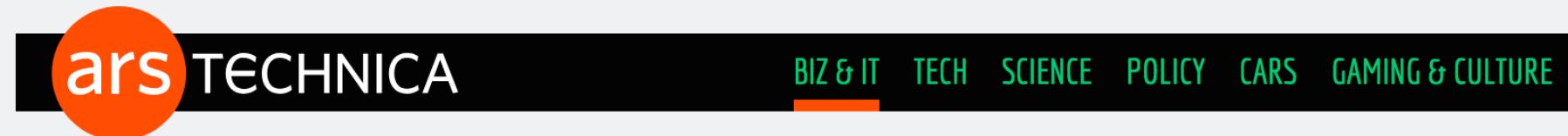


Drammer: Deterministic Rowhammer Attacks on Mobile Platforms, CCS'16

Source: <https://fossbytes.com/drammer-rowhammer-attack-android-root-devices/>

# More Security Implications (III)

- Using an integrated GPU in a mobile system to remotely escalate privilege via the WebGL interface. **IEEE S&P 2018**



"GRAND PWNING UNIT" —

## Drive-by Rowhammer attack uses GPU to compromise an Android phone

JavaScript based GLitch pwns browsers by flipping bits inside memory chips.

## Grand Pwning Unit: Accelerating Microarchitectural Attacks with the GPU

Pietro Frigo  
Vrije Universiteit  
Amsterdam  
p.frigo@vu.nl

Cristiano Giuffrida  
Vrije Universiteit  
Amsterdam  
giuffrida@cs.vu.nl

Herbert Bos  
Vrije Universiteit  
Amsterdam  
herbertb@cs.vu.nl

Kaveh Razavi  
Vrije Universiteit  
Amsterdam  
kaveh@cs.vu.nl

# More Security Implications (IV)

- Rowhammer over RDMA (I)

**ars** TECHNICA

BIZ & IT TECH SCIENCE POLICY CARS GAMING & CULTURE

THROWHAMMER —

## Packets over a LAN are all it takes to trigger serious Rowhammer bit flips

The bar for exploiting potentially serious DDR weakness keeps getting lower.

DAN GOODIN - 5/10/2018, 5:26 PM

### Throwhammer: Rowhammer Attacks over the Network and Defenses

Andrei Tatar  
*VU Amsterdam*

Radhesh Krishnan  
*VU Amsterdam*

Herbert Bos  
*VU Amsterdam*

Elias Athanasopoulos  
*University of Cyprus*

Kaveh Razavi  
*VU Amsterdam*

Cristiano Giuffrida  
*VU Amsterdam*

# More Security Implications (V)

- Rowhammer over RDMA (II)



Nethammer—Exploiting DRAM Rowhammer Bug Through Network Requests

## **Nethammer: Inducing Rowhammer Faults through Network Requests**

Moritz Lipp  
Graz University of Technology

Daniel Gruss  
Graz University of Technology

Misiker Tadesse Aga  
University of Michigan

Clémentine Maurice  
Univ Rennes, CNRS, IRISA

Lukas Lamster  
Graz University of Technology



Michael Schwarz  
Graz University of Technology

Lukas Raab  
Graz University of Technology

# More Security Implications (VI)

## JackHammer: Efficient Rowhammer on Heterogeneous FPGA-CPU Platforms

Zane Weissman<sup>1</sup>, Thore Tiemann<sup>2</sup>, Daniel Moghimi<sup>1</sup>, Evan Custodio<sup>3</sup>, Thomas Eisenbarth<sup>2</sup> and Berk Sunar<sup>1</sup>

<sup>1</sup> Worcester Polytechnic Institute, MA, USA

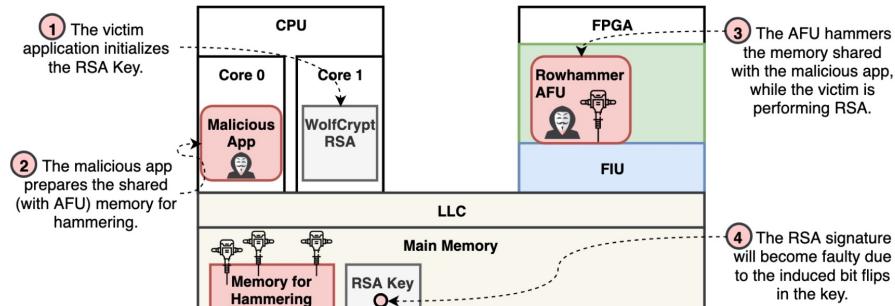
[zweissman@wpi.edu](mailto:zweissman@wpi.edu), [amoghimi@wpi.edu](mailto:amoghimi@wpi.edu), [sunar@wpi.edu](mailto:sunar@wpi.edu)

<sup>2</sup> University of Lübeck, Lübeck, Germany

[thore.tiemann@student.uni-luebeck.de](mailto:thore.tiemann@student.uni-luebeck.de), [thomas.eisenbarth@uni-luebeck.de](mailto:thomas.eisenbarth@uni-luebeck.de)

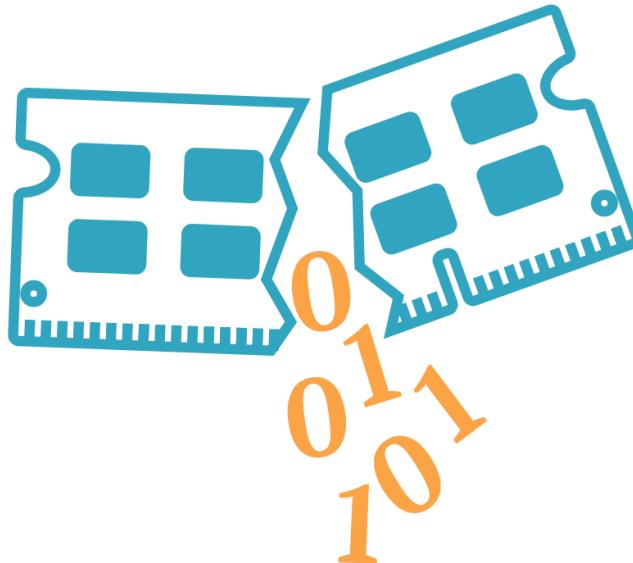
<sup>3</sup> Intel Corporation, Hudson, MA, USA

[evan.custodio@intel.com](mailto:evan.custodio@intel.com)



An **FPGA-based** RowHammer attack recovering **private keys** twice as fast compared to **CPU-based** attacks

# More Security Implications (VII)



## RAMBleed

**RAMBleed: Reading Bits in Memory Without Accessing Them**

Andrew Kwong

*University of Michigan*

[ankwong@umich.edu](mailto:ankwong@umich.edu)

Daniel Genkin

*University of Michigan*

[genkin@umich.edu](mailto:genkin@umich.edu)

Daniel Gruss

*Graz University of Technology*

[daniel.gruss@iaik.tugraz.at](mailto:daniel.gruss@iaik.tugraz.at)

Yuval Yarom

*University of Adelaide and Data61*

[yval@cs.adelaide.edu.au](mailto:yval@cs.adelaide.edu.au)

# More Security Implications (VIII)

## Terminal Brain Damage: Exposing the Graceless Degradation in Deep Neural Networks Under Hardware Fault Attacks

Sanghyun Hong, Pietro Frigo<sup>†</sup>, Yiğitcan Kaya, Cristiano Giuffrida<sup>†</sup>, Tudor Dumitraş

*University of Maryland, College Park*

*†Vrije Universiteit Amsterdam*



### A Single Bit-flip Can Cause Terminal Brain Damage to DNNs

*One specific bit-flip in a DNN's representation leads to accuracy drop over 90%*

Our research found that a specific bit-flip in a DNN's bitwise representation can cause the accuracy loss up to 90%, and the DNN has 40-50% parameters, on average, that can lead to the accuracy drop over 10% when individually subjected to such single bitwise corruptions...

[Read More](#)

# More Security Implications (IX)

## DeepHammer: Depleting the Intelligence of Deep Neural Networks through Targeted Chain of Bit Flips

Fan Yao

*University of Central Florida*

[fan.yao@ucf.edu](mailto:fan.yao@ucf.edu)

Adnan Siraj Rakin

*Arizona State University*

[asrakin@asu.edu](mailto:asrakin@asu.edu)

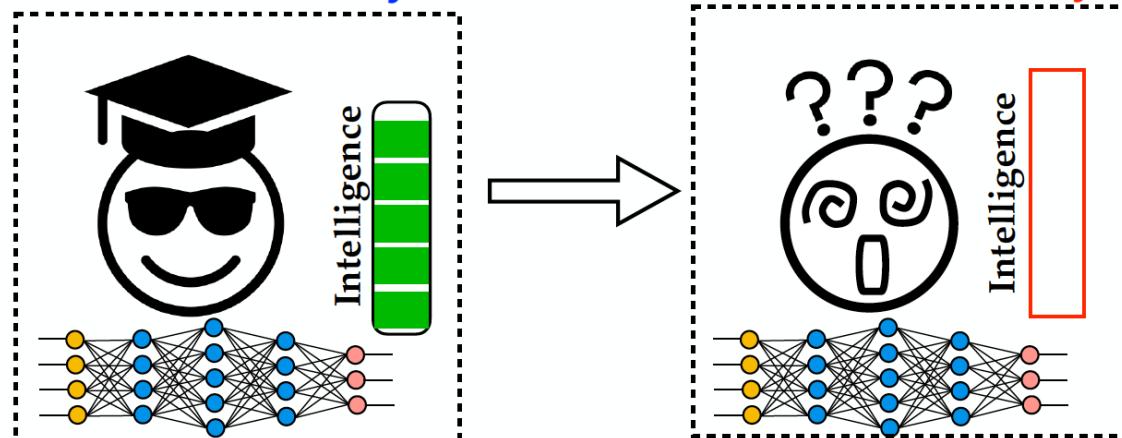
Deliang Fan

*Arizona State University*  
[dfan@asu.edu](mailto:dfan@asu.edu)

Degrade the inference accuracy to the level of Random Guess

Example: ResNet-20 for CIFAR-10, 10 output classes

Before attack, **Accuracy: 90.2%** After attack, **Accuracy: ~10% (1/10)**



# More Security Implications (X)

## HAMMERSCOPE: Observing DRAM Power Consumption Using Rowhammer

Yaakov Cohen\*

Ben-Gurion University of the Negev  
and Intel  
Beer-Sheva, Israel  
yaakoc@post.bgu.ac.il

Daniel Genkin

Georgia Institute of Technology  
Atlanta, Georgia, USA  
genkin@gatech.edu

Kevin Sam Tharayil\*

Georgia Institute of Technology  
Atlanta, Georgia, USA  
kevinsam@gatech.edu

Arie Haenel\*

Jerusalem College of Technology  
and Intel  
Jerusalem, Israel  
arie.haenel@jct.ac.il

Angelos D. Keromytis

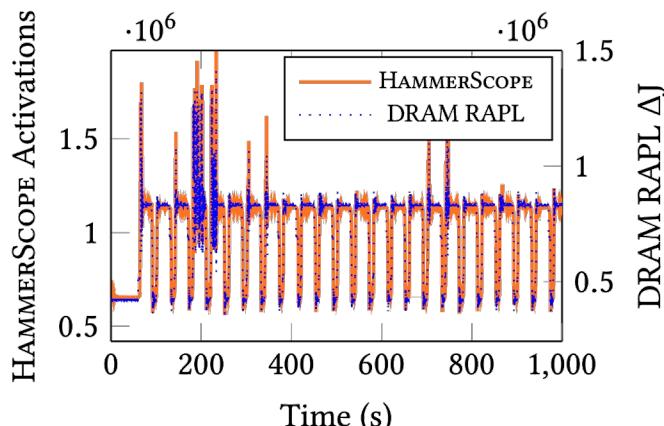
Georgia Institute of Technology  
Atlanta, Georgia, USA  
angelos@gatech.edu

Yossi Oren

Ben-Gurion University of the Negev  
and Intel  
Beer-Sheva, Israel  
yos@bgu.ac.il

Yuval Yarom

University of Adelaide  
Adelaide, Australia  
yval@cs.adelaide.edu.au



HammerScope is a **software-based power analysis** method using **RowHammer** as a side channel

# More Security Implications (XI)

- **Exploiting Correcting Codes: On the Effectiveness of ECC Memory Against Rowhammer Attacks.** Cojocar, L.; Razavi, K.; Giuffrida, C.; and Bos, H. In *S&P*, May 2019 *Best Practical Paper Award, Pwnie Award Nomination for Most Innovative Research* [[Paper](#)] [[Slides](#)]

## Exploiting Correcting Codes: On the Effectiveness of ECC Memory Against Rowhammer Attacks

Lucian Cojocar, Kaveh Razavi, Cristiano Giuffrida, Herbert Bos  
Vrije Universiteit Amsterdam

*Thus, many believed that Rowhammer on ECC memory, even if plausible in theory, is simply impractical. This paper shows this to be false: while harder, Rowhammer attacks are still a realistic threat even to modern ECC-equipped systems.*

# More Security Implications (XII)

Hasan Hassan, Yahya Can Tugrul, Jeremie S. Kim, Victor van der Veen, Kaveh Razavi, and Onur Mutlu,  
["Uncovering In-DRAM RowHammer Protection Mechanisms: A New Methodology, Custom RowHammer Patterns, and Implications,"](#) *MICRO*, 2021. [[Slides \(pptx\)](#) [\(pdf\)](#)] [[Short Talk Slides \(pptx\)](#) [\(pdf\)](#)] [[Lightning Talk Slides \(pptx\)](#) [\(pdf\)](#)] [[Full Talk](#) (25 mins)] [[Lightning Talk](#) (1.5 mins)]

## Uncovering In-DRAM RowHammer Protection Mechanisms: A New Methodology, Custom RowHammer Patterns, and Implications

Hasan Hassan<sup>†</sup>

<sup>†</sup>*ETH Zürich*

Yahya Can Tuğrul<sup>†‡</sup>

Kaveh Razavi<sup>†</sup>

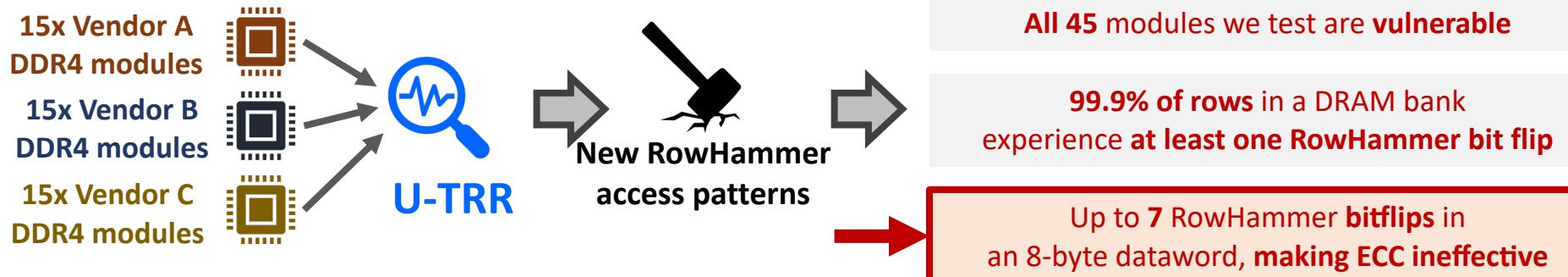
<sup>‡</sup>*TOBB University of Economics & Technology*

Jeremie S. Kim<sup>†</sup>

Onur Mutlu<sup>†</sup>

Victor van der Veen<sup>σ</sup>

<sup>σ</sup>*Qualcomm Technologies Inc.*



TRR does not provide security against RowHammer

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

# A RowHammer Survey:

Onur Mutlu, Ataberk Olgun, and A. Giray Yaglikci, "Fundamentally Understanding and Solving RowHammer" Invited Special Session Paper at the 28th Asia and South Pacific Design Automation Conference (ASP-DAC), Tokyo, Japan, January 2023.

[[arXiv version](#)]

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

[[Talk Video](#) (26 minutes)]

## Fundamentally Understanding and Solving RowHammer

Onur Mutlu

[onur.mutlu@safari.ethz.ch](mailto:onur.mutlu@safari.ethz.ch)

ETH Zürich

Zürich, Switzerland

Ataberk Olgun

[ataberk.olgun@safari.ethz.ch](mailto:ataberk.olgun@safari.ethz.ch)

ETH Zürich

Zürich, Switzerland

A. Giray Yağlıkçı

[giray.yaglikci@safari.ethz.ch](mailto:giray.yaglikci@safari.ethz.ch)

ETH Zürich

Zürich, Switzerland

<https://arxiv.org/pdf/2211.07613.pdf>

# Breaking Samsung's Best Practice

- Salman Qazi and Daniel Moghimi, “[SoothSayer: Bypassing DSAC Mitigation by Predicting Counter Replacement](#),” DRAMSec, 2024

## SoothSayer: Bypassing DSAC Mitigation by Predicting Counter Replacement

Salman Qazi  
Google

Daniel Moghimi  
Google

**Abstract**—In-DRAM Stochastic and Approximate Counting (DSAC) is a recently published algorithm that aims to mitigate Rowhammer at low cost. Existing in-DRAM counter-based schemes keep track of row activations and issue Targeted Row Refresh (TRR) upon detecting a concerning pattern. However, due to insufficiency of the tracking ability they are vulnerable to attacks utilizing decoy rows. DSAC claims to improve upon existing TRR mitigation by filtering out decoy-row accesses, so they cannot saturate the limited number of counters available for detecting Rowhammer, promising a reliable mitigation without the area cost of deterministic and provable schemes such as per-row activation counting (PRAC).

In this paper, we analyze DSAC and discover some gaps that make it vulnerable to Rowhammer and Rowpress attacks.

The main focus of this work is a novel attack named SoothSayer that targets the counter replacement policy in DSAC by cloning the random number generator. We describe and simulate

literature such as Graphene [18] (implemented in the memory controller) and ProTRR [17] (implemented in DRAM) that utilize frequent item counting schemes. These can account for all Rowhammer activity if the threshold is sufficiently large and enough counters are provided. As the Rowhammer threshold decreases, the number of counters required for a correct implementation increases. According to the authors of DSAC, who are affiliated with a memory vendor, the number of counters used in these implementations are unacceptably large for a memory vendor to implement within their designs. To avoid this cost, deployed counter-based mitigations employ fewer counters than necessary and are often probabilistic. Due to this limitation, recent Rowhammer techniques [2], [8], [13] have managed to bypass TRR with decoy DRAM

# RowHammer in DDR5

Patrick Jattke; Max Wipfli; Flavien Solt; Michele Marazzi; Matej Bölcskei; and Kaveh Razavi, [\*\*“ZenHammer: Rowhammer Attacks on AMD Zen-based Platforms,”\*\*](#) in *USENIX Security*, 2024.  
[\[Paper\]](#) [\[URL\]](#)

## ZENHAMMER: Rowhammer Attacks on AMD Zen-based Platforms

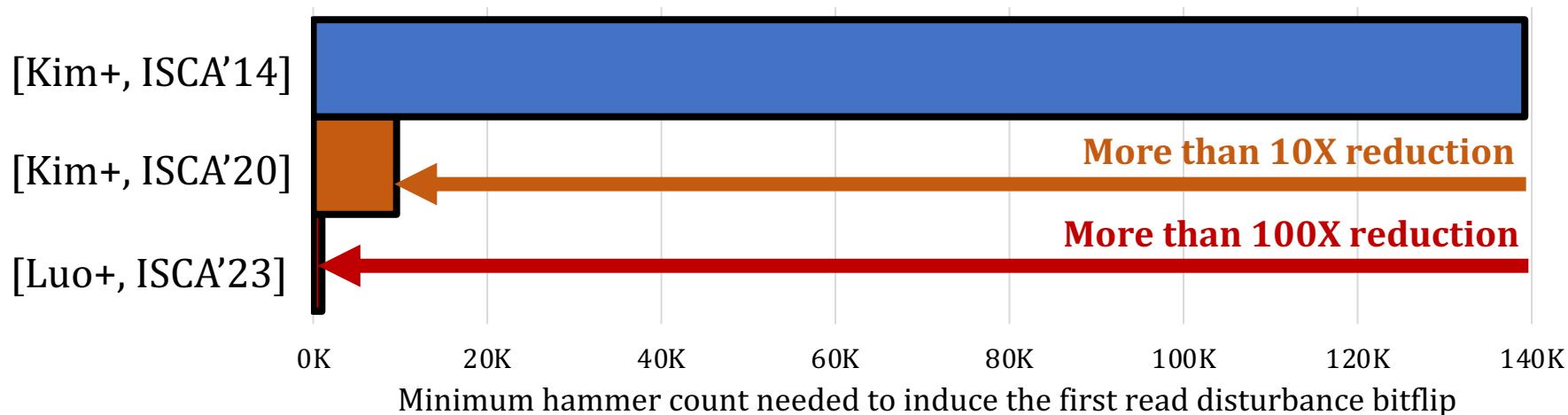
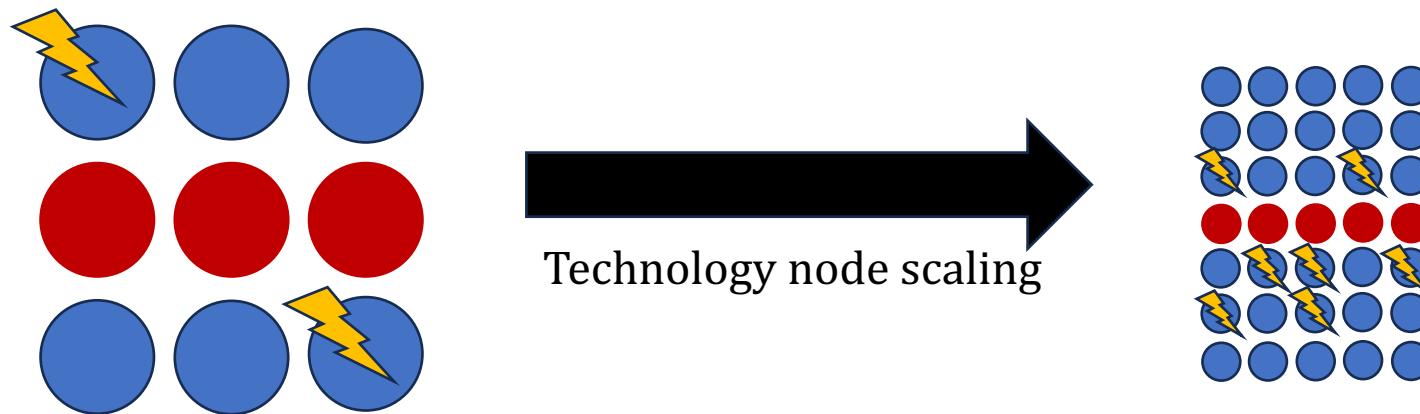
Patrick Jattke<sup>†</sup> Max Wipfli<sup>†</sup> Flavien Solt Michele Marazzi Matej Bölcskei Kaveh Razavi

*ETH Zurich*

<sup>†</sup>Equal contribution first authors

*We found bit flips on only 1 of 10 tested devices (S1), suggesting that the changes in DDR5 such as improved Rowhammer mitigations, on-die error correction code (ECC), and a higher refresh rate (32 ms) make it harder to trigger bit flips.*

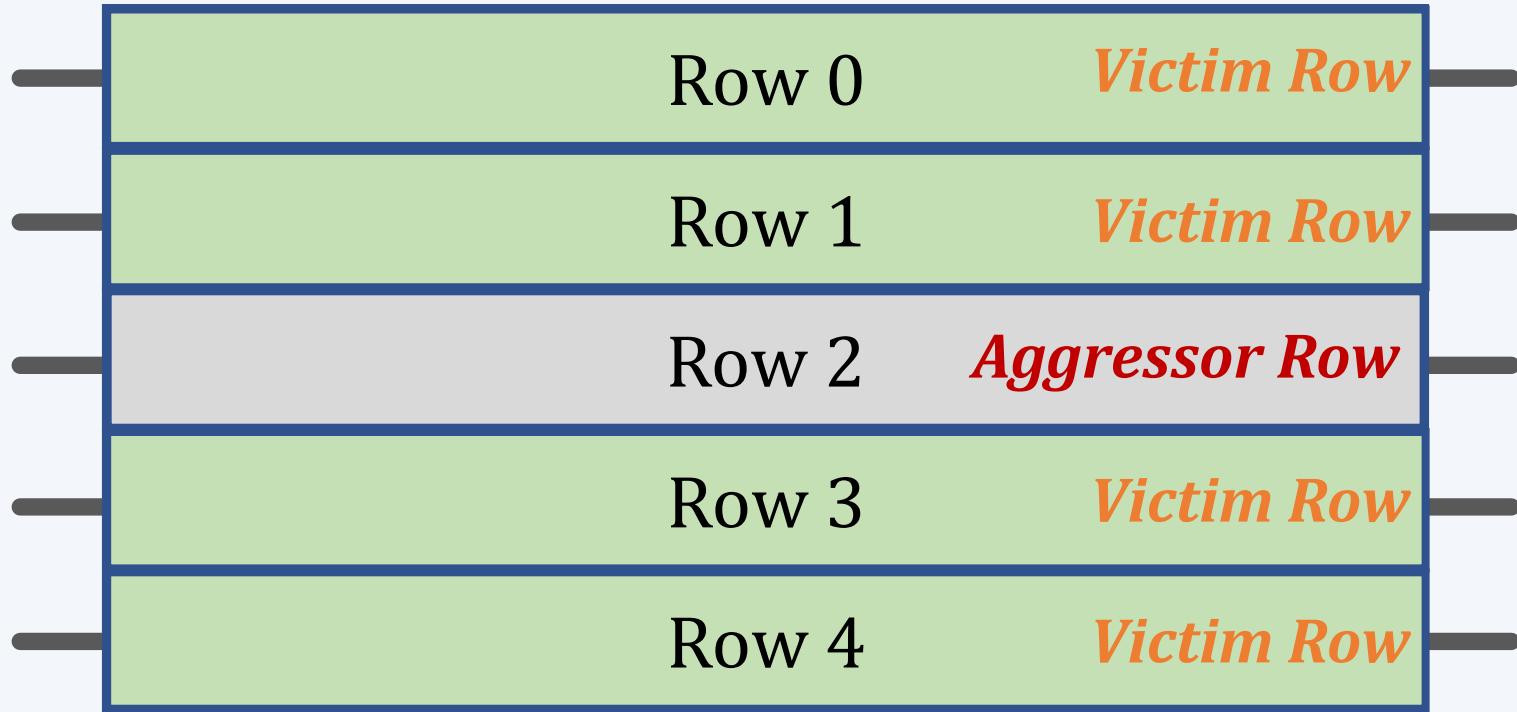
# DRAM Read Disturbance: A Critical Challenge



DRAM cells become **increasingly more vulnerable to read disturbance**

# Existing RowHammer Mitigations (I)

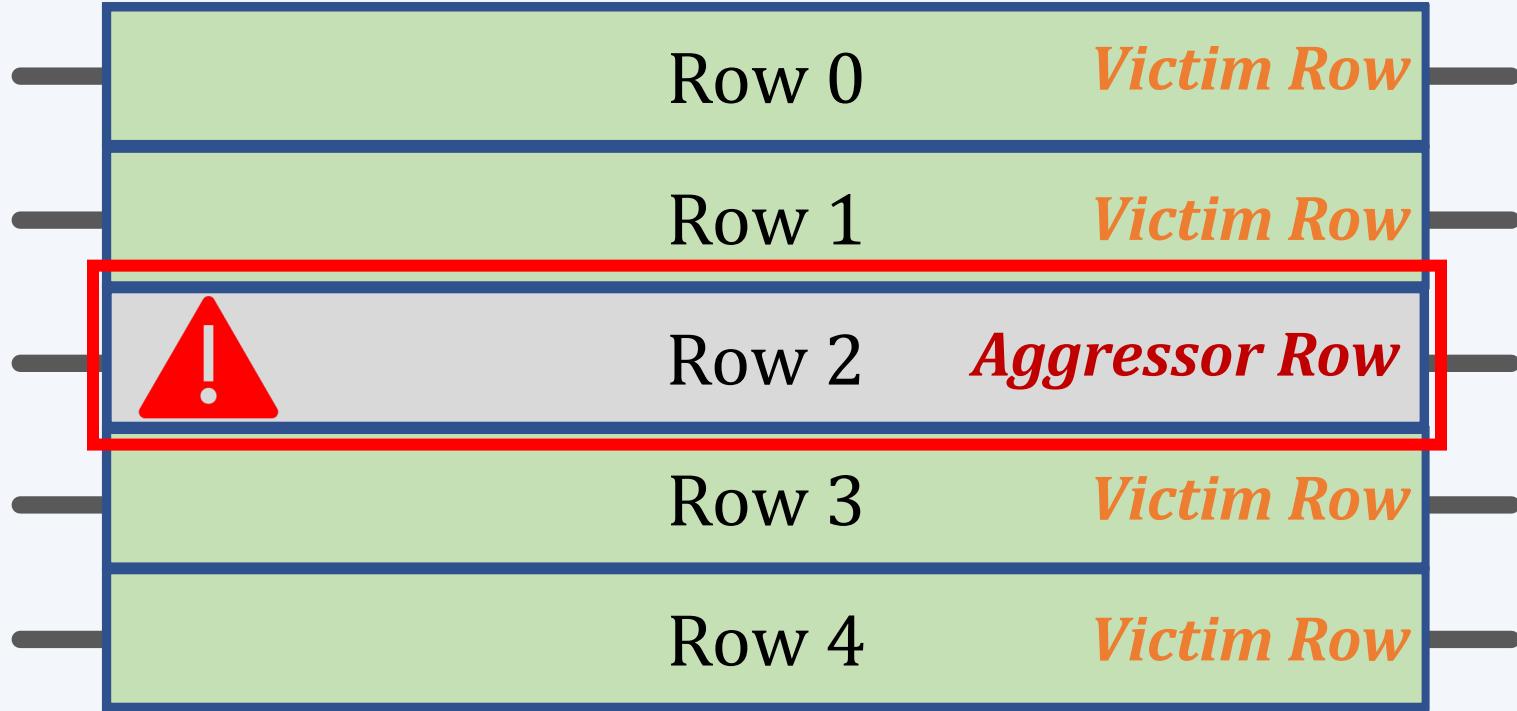
## DRAM Subarray



Refreshing potential victim rows  
mitigates read disturbance bitflips

# Existing RowHammer Mitigations (II)

## DRAM Subarray



Mitigation techniques track DRAM row activation counts (of aggressor rows) to preventively refresh potential victim rows

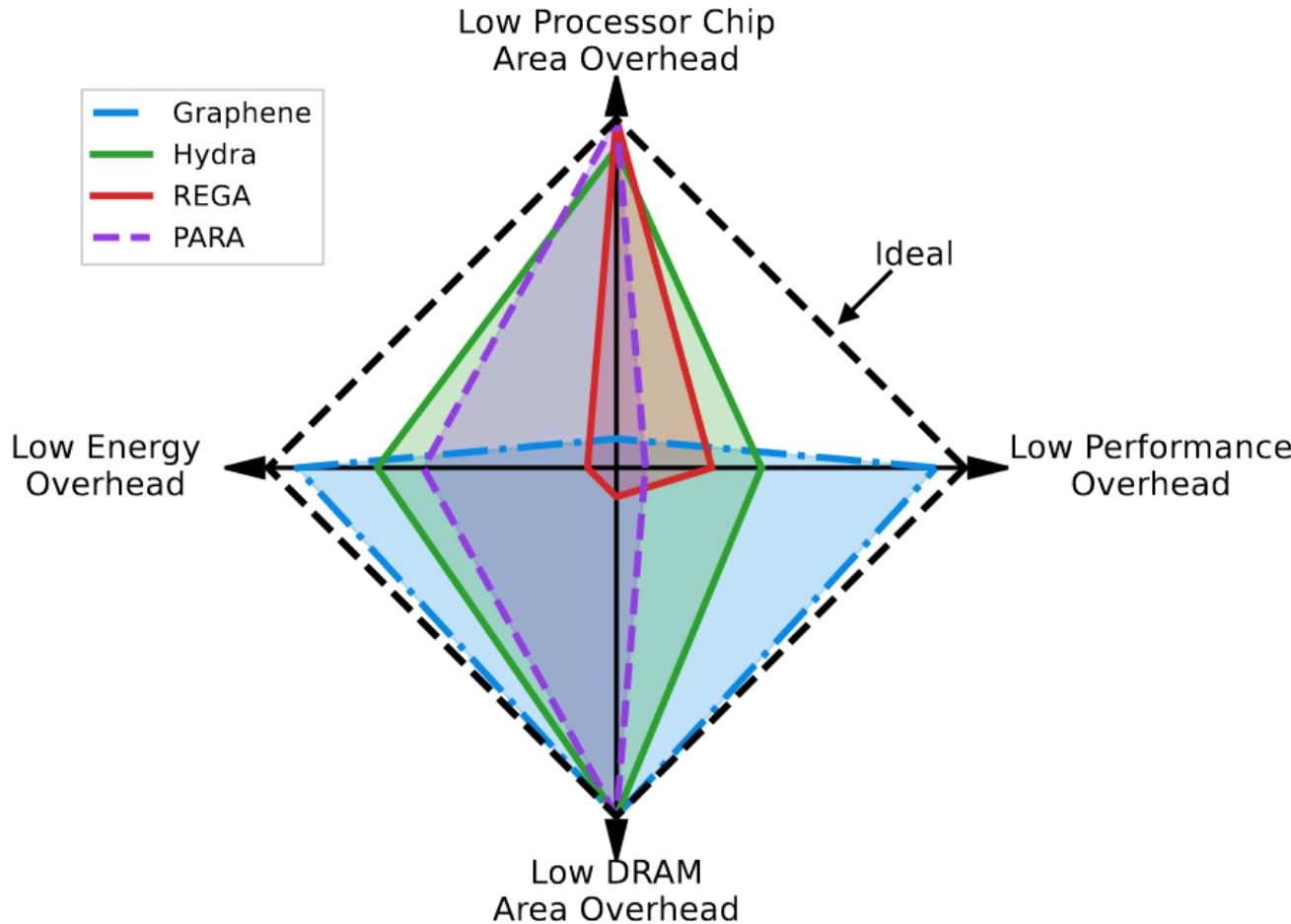
# Existing RowHammer Mitigations (III)

## Overview of Preventive Refresh-Based Mitigation Techniques

<p>DRAM</p> <p>One ACT counter per DRAM row</p>	<p>Processor Chip</p> <p>One ACT counter per aggressor row</p> <p>CAM-based implementation</p>	<p>Processor Chip</p> <p>Shared Counters</p> <p>&lt;1 counter per DRAM row</p>
<p>Performance &amp; Energy Costs</p> <p>VERY LOW</p> <p>Many DRAM rows (e.g., 128K per bank)</p>	<p>LOW</p> <p># of aggressor rows increases significantly</p>	<p>HIGH</p> <p>High DRAM bandwidth consumption</p>
<p>Area Cost</p> <p>VERY HIGH</p>	<p>HIGH</p>	<p>LOW</p>

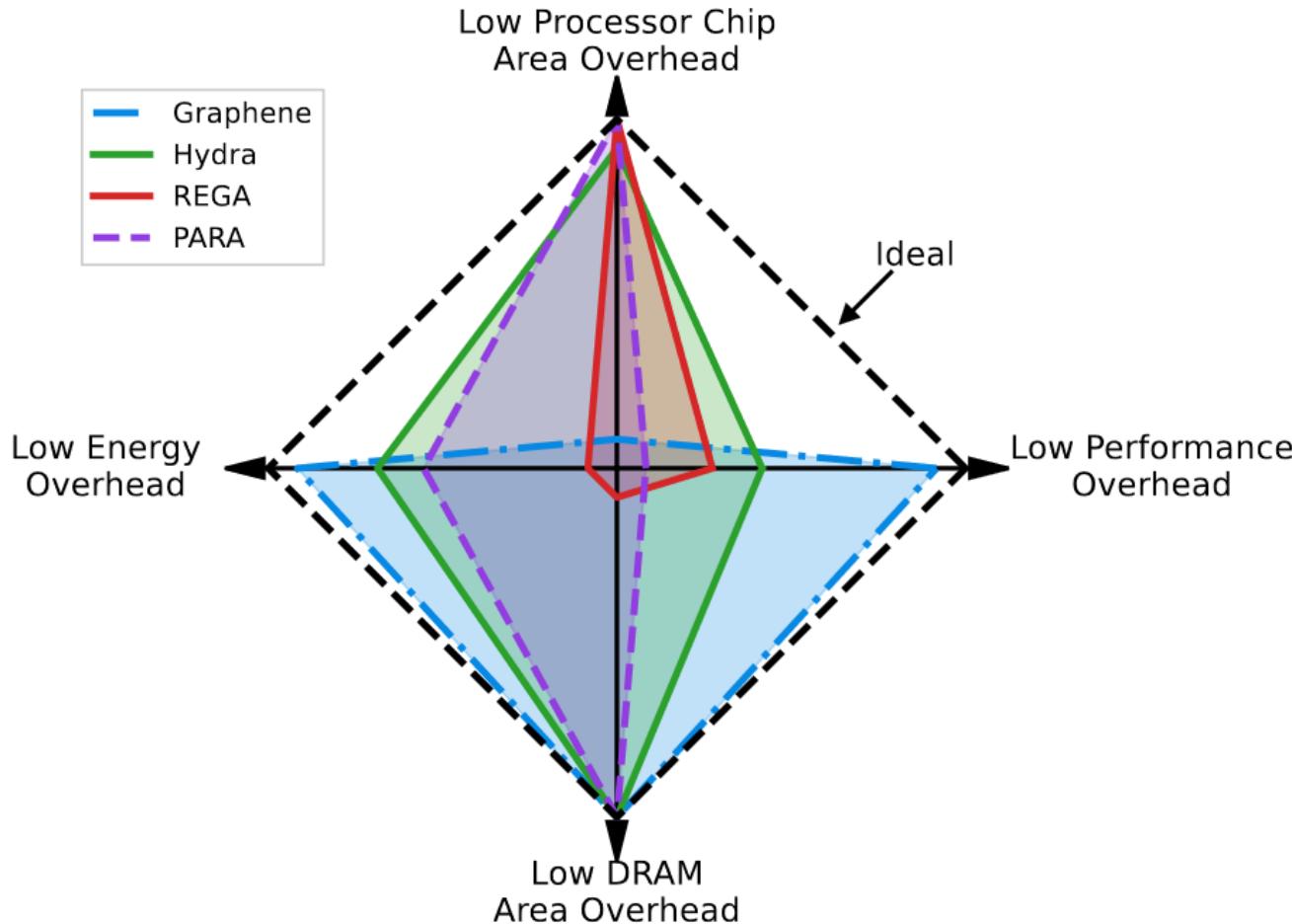
# Existing RowHammer Mitigations (IV)

## Overhead Trade-Off of State-of-the-Art Mitigation Techniques



# Existing RowHammer Mitigations (IV)

## Overhead Trade-Off of State-of-the-Art Mitigation Techniques



No existing mitigation technique prevents RowHammer bitflips  
at low area, performance and energy costs

# Our Goal

Prevent RowHammer bitflips  
with low area, performance, and energy overheads  
in **highly vulnerable DRAM-based systems**  
(e.g., a RowHammer threshold of 125)

# ABACuS: All-Bank Activation Counters

- Ataberk Olgun, Yahya Can Tugrul, Nisa Bostanci, Ismail Emir Yuksel, Haocong Luo, Steve Rhyner, Abdullah Giray Yaglikci, Geraldo F. Oliveira, and Onur Mutlu, **"ABACuS: All-Bank Activation Counters for Scalable and Low Overhead RowHammer Mitigation"** *To appear in Proceedings of the 33rd USENIX Security Symposium (USENIX Security)*, Philadelphia, PA, USA, August 2024.  
[\[arXiv version\]](#) [\[ABACuS Source Code\]](#)

## **ABACuS: All-Bank Activation Counters for Scalable and Low Overhead RowHammer Mitigation**

Ataberk Olgun      Yahya Can Tugrul      Nisa Bostanci      Ismail Emir Yuksel  
Haocong Luo    Steve Rhyner    Abdullah Giray Yaglikci    Geraldo F. Oliveira    Onur Mutlu

ETH Zurich

will be presented next week  
in USENIX Security 2024



# ABACuS

## All-Bank Activation Counters for Scalable and Low Overhead RowHammer Mitigation

Ataberk Olgun

İsmail Emir Yüksel

A. Giray Yağlıkçı

Yahya Can Tuğrul

Haocong Luo

Geraldo F. Oliveira

F. Nisa Bostancı

Steve Rhyner

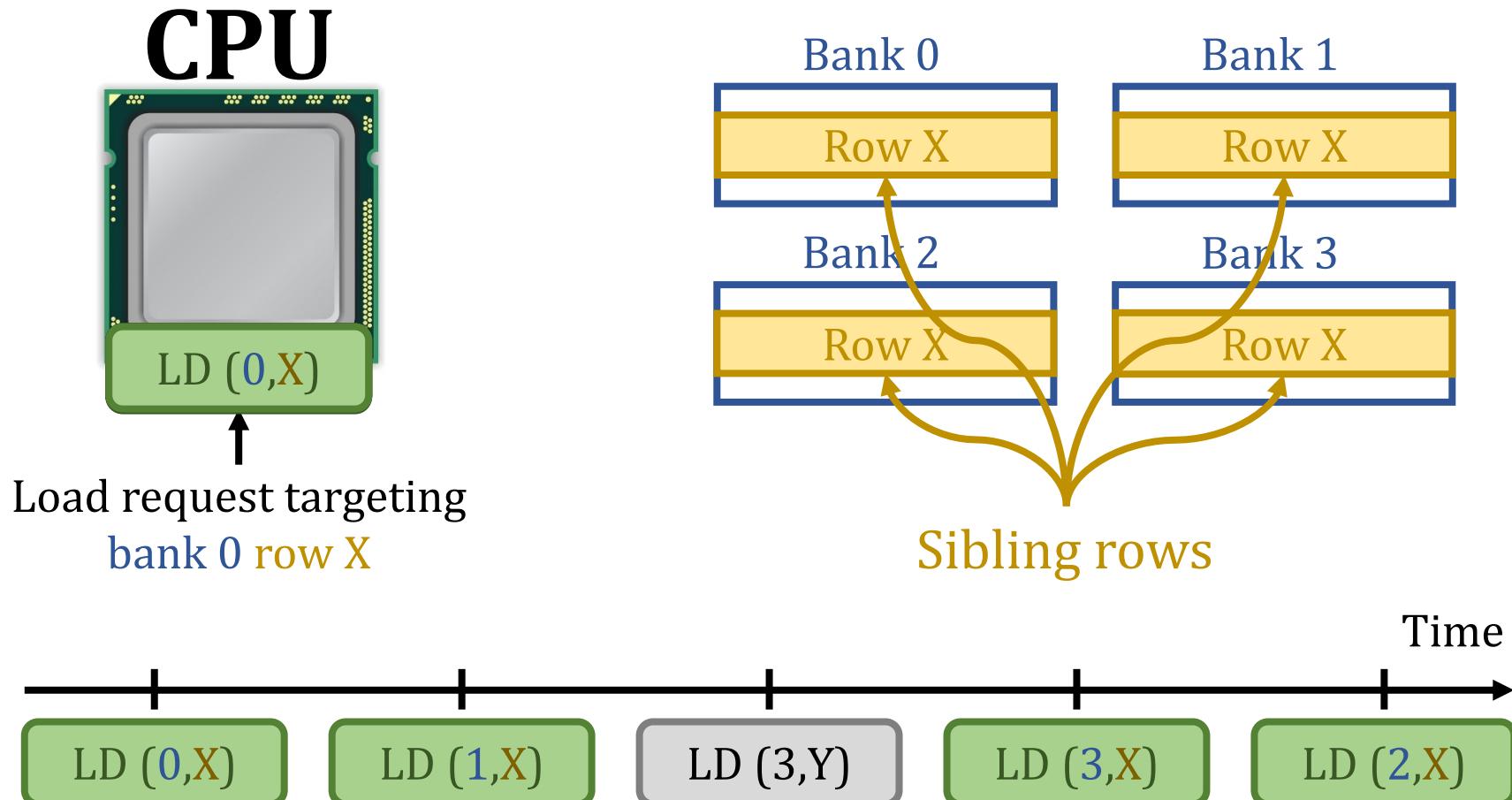
Onur Mutlu

**SAFARI**

**ETH** zürich

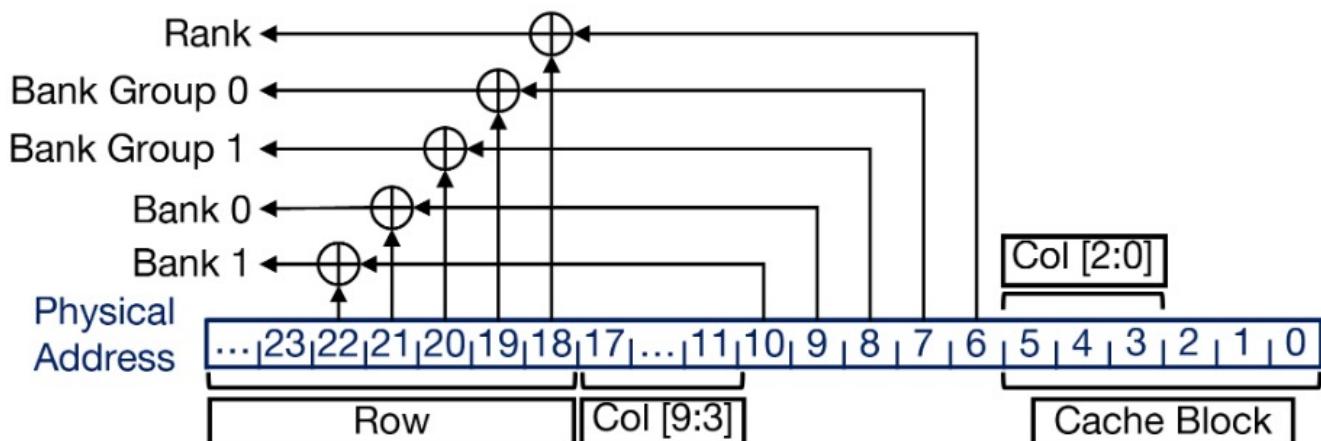
# Key Observation

Many workloads access the **same row address** in different **banks** at **around the same time**

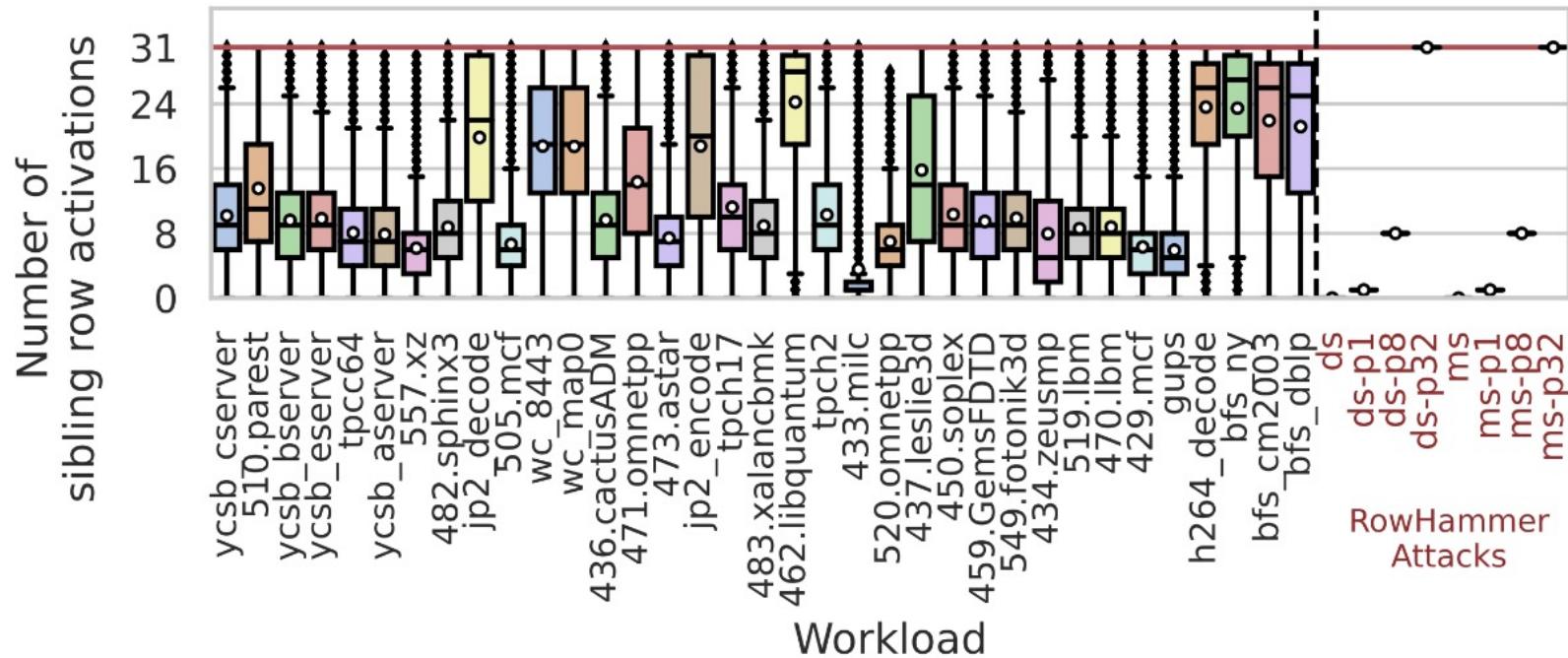


# Explanation for the Key Observation

- 1 Programs exhibit **spatial locality**: access **cache blocks** around each other at around the same time
- 2 Address mappings distribute **consecutive** cache blocks to **different banks** to leverage **bank-level parallelism** (using the **same row ID**)



# Motivational Analysis



Gathered from 128 row activation windows in a 32-bank system

Workloads access **the same row address**  
in **different DRAM banks** at around **the same time**

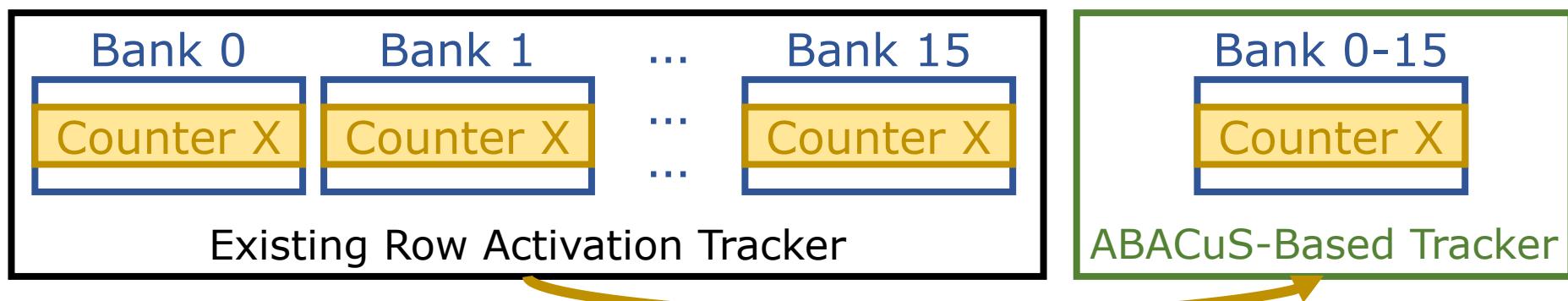
# Problem

No mitigation mechanism leverages  
the spatial locality across banks

Area overhead linearly increases  
with the number of banks

# Key Idea

Sibling rows can share one activation counter to reduce the number of counters by a factor of the bank count

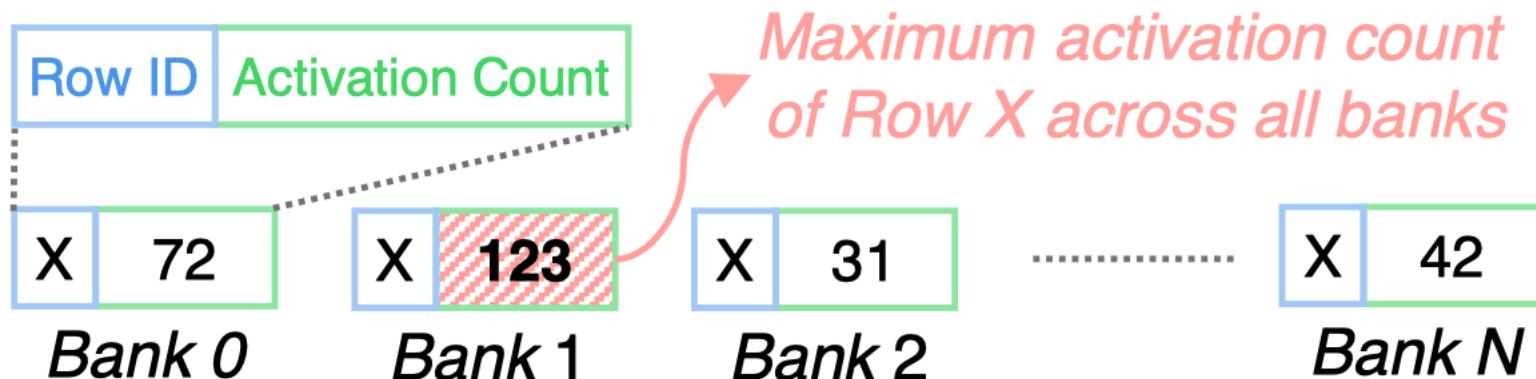


16x reduction in number of counters

**Sibling rows:** Rows with the same ID across all banks

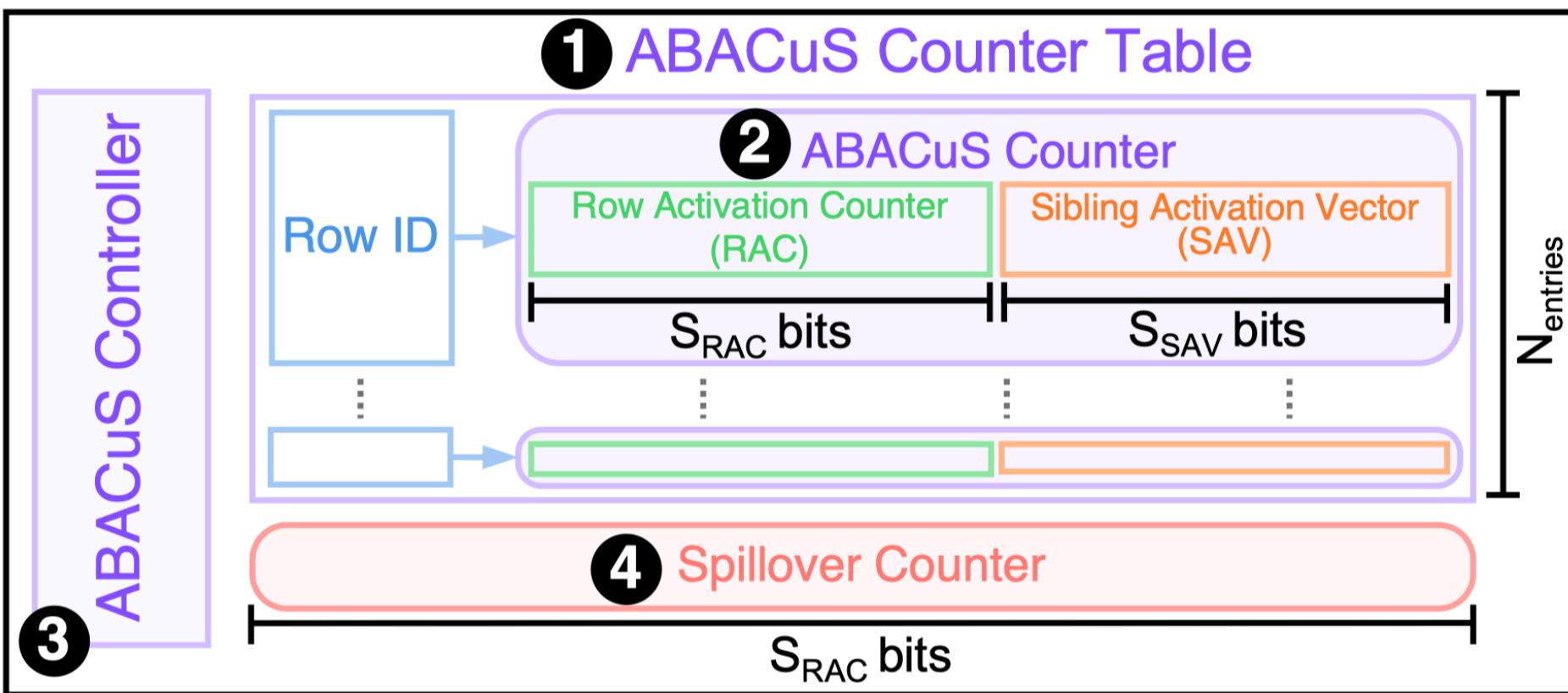
# ABAxCuS: High-level Overview

**Key Mechanism:** Track the **maximum** (worst) activation count of sibling rows using **one counter**

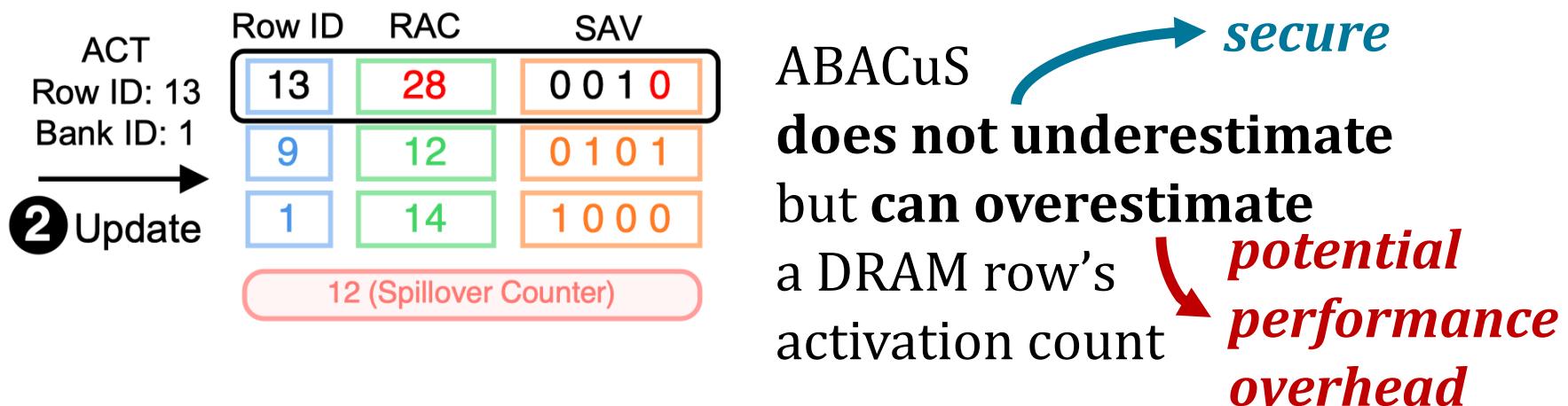
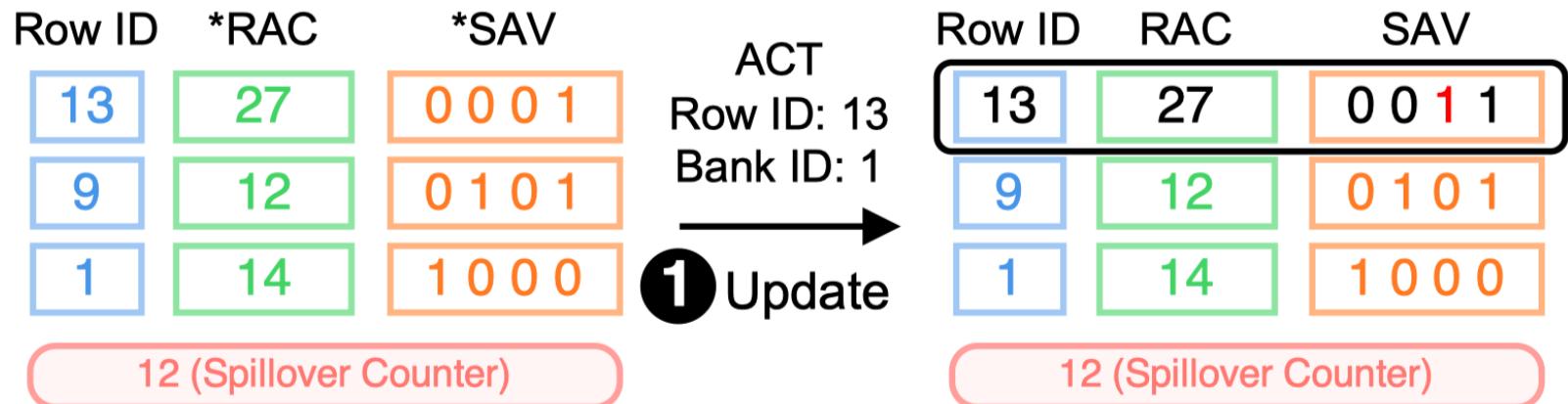


# ABACuS's Integration with Misra-Gries Algorithm (I)

## Memory Controller



# ABACuS's Integration with Misra-Gries Algorithm (II)



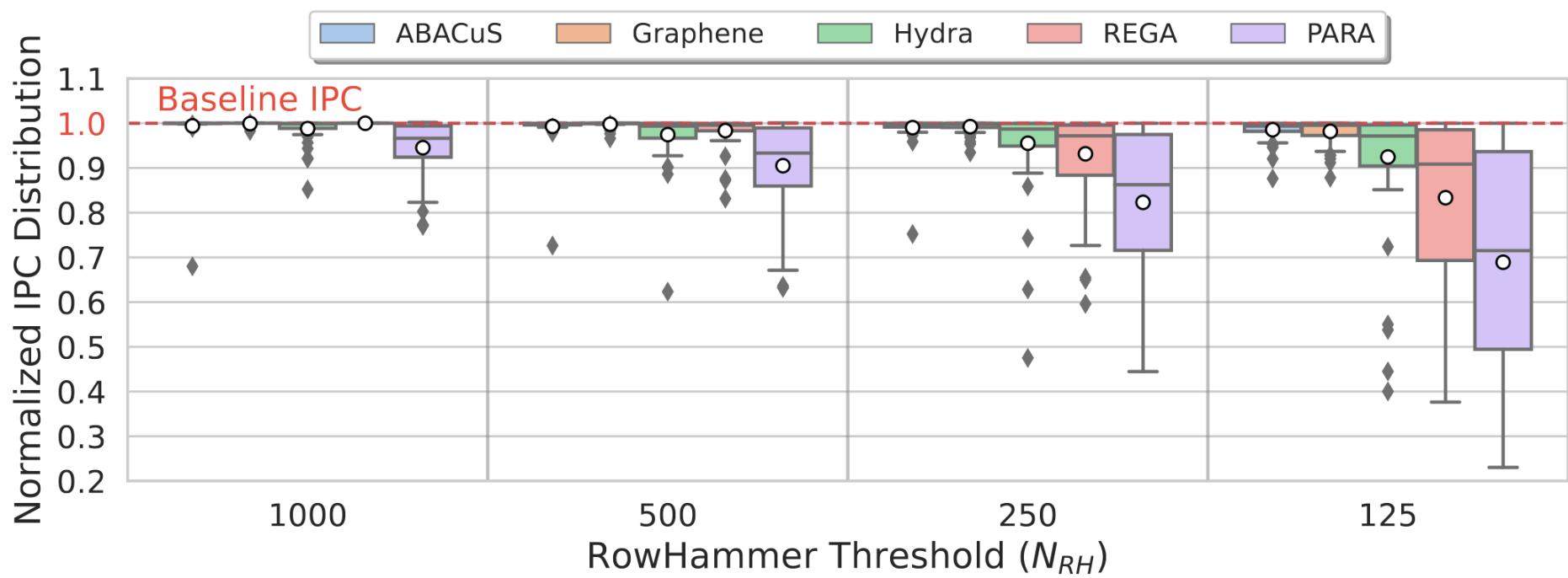
# Evaluation

## System Configuration

<b>Processor</b>	1 or 8 cores, 3.6GHz clock frequency, 4-wide issue, 128-entry instruction window
<b>DRAM</b>	DDR4, 1 channel, 2 rank/channel, 4 bank groups, 4 banks/bank group, 128K rows/bank, 3200 MT/s
<b>Memory Ctrl.</b>	64-entry read and write requests queues, Scheduling policy: FR-FCFS [181, 182] with a column cap of 16 [183], Address mapping: MOP [166, 168] 45 ns $tRC$ , 7.9 $\mu$ s $tREFI$ , 64 ms $tREFW$ 64 ms ABACuS reset period
<b>Last-Level Cache</b>	2 MiB per core

- **Workloads:** 62 1- & 8-core workloads
- Four different very low nRH values: 1000, 500, 250, 125
- Four state-of-the-art mitigation mechanisms

# Performance Comparison



Lower overhead than all evaluated state-of-the-art mechanisms (except Graphene)

# More in the Paper

- More motivational analysis
- Multi-core performance & energy results
- Performance under adversarial workloads
  - Alternative ABACuS design
- Performance & energy sensitivity to:
  - Blast radius
  - Number of ABACuS counters
  - Number of banks
- Circuit area, latency, energy, and power
- Security proof

# ABACuS Summary

**Goal:** Prevent RowHammer bitflips at low performance, energy, and area cost especially at very low RowHammer thresholds (e.g., 125 aggressor row activations induce a bitflip)

**Key Observation:** Many workloads access the same row address in different DRAM banks at around the same time

**Key Idea:** Use one counter to track the activation count of many rows with the same address across all DRAM banks

**Key Results:** At very low RowHammer thresholds, ABACuS:

- Induces small system performance and DRAM energy overhead
- Outperforms the state-of-the-art mitigation (Hydra)
- Takes up 22.7X smaller chip area than state-of-the-art (Graphene)

# Extended Version on arXiv

<https://arxiv.org/pdf/2310.09977.pdf>

arXiv > cs > arXiv:2310.09977

Search... All fields Help | Advanced Search

Computer Science > Cryptography and Security

[Submitted on 15 Oct 2023]

## ABACuS: All-Bank Activation Counters for Scalable and Low Overhead RowHammer Mitigation

Ataberk Olgun, Yahya Can Tugrul, Nisa Bostancı, Ismail Emir Yuksel, Haocong Luo, Steve Rhyner, Abdullah Giray Yaglikci, Geraldo F. Oliveira, Onur Mutlu

We introduce ABACuS, a new low-cost hardware-counter-based RowHammer mitigation technique that performance-, energy-, and area-efficiently scales with worsening RowHammer vulnerability. We observe that both benign workloads and RowHammer attacks tend to access DRAM rows with the same row address in multiple DRAM banks at around the same time. Based on this observation, ABACuS's key idea is to use a single shared row activation counter to track activations to the rows with the same row address in all DRAM banks. Unlike state-of-the-art RowHammer mitigation mechanisms that implement a separate row activation counter for each DRAM bank, ABACuS implements fewer counters (e.g., only one) to track an equal number of aggressor rows.

Our evaluations show that ABACuS securely prevents RowHammer bitflips at low performance/energy overhead and low area cost. We compare ABACuS to four state-of-the-art mitigation mechanisms. At a near-future RowHammer threshold of 1000, ABACuS incurs only 0.58% (0.77%) performance and 1.66% (2.12%) DRAM energy overheads, averaged across 62 single-core (8-core) workloads, requiring only 9.47 KiB of storage per DRAM rank. At the RowHammer threshold of 1000, the best prior low-area-cost mitigation mechanism incurs 1.80% higher average performance overhead than ABACuS, while ABACuS requires 2.50X smaller chip area to implement. At a future RowHammer threshold of 125, ABACuS performs very similarly to (within 0.38% of the performance of) the best prior performance- and energy-efficient RowHammer mitigation mechanism while requiring 22.72X smaller chip area. ABACuS is freely and openly available at [this https URL](https://arxiv.org/pdf/2310.09977.pdf).

Access Paper:

- Download PDF
- PostScript
- Other Formats

(cc) BY

Current browse context: cs.CR  
< prev | next >  
new | recent | 2310

Change to browse by:  
cs  
cs.AR

References & Citations  
[NASA ADS](#)  
[Google Scholar](#)  
[Semantic Scholar](#)

Export BibTeX Citation

Bookmark

# ABACuS is Open Source

<https://github.com/CMU-SAFARI/ABACuS>

The screenshot shows the GitHub interface for the ABACuS repository. At the top, there's a navigation bar with links for Code, Issues, Pull requests, Actions, Projects, Security, Insights, and Settings. Below the navigation bar, the repository name 'ABACuS' is displayed along with its status as 'Public'. On the right side of the header, there are buttons for Edit Pins, Unwatch (with 4 notifications), Fork (with 0 forks), and Starred (with 3 stars). The main content area shows a list of commits from the 'main' branch. Each commit includes the author's icon, the author's name, the commit message, the commit hash, the time of the commit, and the number of commits. The commits listed are:

- olgunataberk add verilog sources and update readme ef1c89c yesterday 8 commits
- abacus\_cacti add abacus cacti sources yesterday
- abacus\_verilog add verilog sources and update readme yesterday
- configs/ABACUS Initial commit 3 days ago
- ext Initial commit 3 days ago
- scripts Initial commit 3 days ago
- src Initial commit 3 days ago
- .gitignore Initial commit 3 days ago
- CMakeLists.txt Initial commit 3 days ago
- Doxmyfile Initial commit 3 days ago

On the right side of the commits, there's an 'About' section with a detailed description of the RowHammer mitigation mechanism, a link to the USENIX Security'24 paper, and various repository statistics like Readme, MIT license, Activity, stars, watching, forks, and a 'Report repository' link.

# CoMeT: Count-Min-Sketch-based Row Tracking

- F. Nisa Bostancı, Ismail Emir Yuksel, Ataberk Olgun, Konstantinos Kanellopoulos, Yahya Can Tugrul, A. Giray Yaglikci, Mohammad Sadrosadati, and Onur Mutlu,

**"CoMeT: Count-Min-Sketch-based Row Tracking to Mitigate RowHammer at Low Cost"** *Proceedings of the 30th International Symposium on High-Performance Computer Architecture (HPCA)*, April 2024.

[[Slides \(pptx\)](#) [\(pdf\)](#)] [[arXiv version](#)]

[[CoMeT Source Code](#) ([Officially Artifact Evaluated with All Badges](#))]

***Officially artifact evaluated as available, reviewed and reproducible.***



## CoMeT: Count-Min-Sketch-based Row Tracking to Mitigate RowHammer at Low Cost

F. Nisa Bostancı  
Yahya Can Tuğrul

İsmail Emir Yüksel  
A. Giray Yağlıkçı

Ataberk Olgun  
Mohammad Sadrosadati

Konstantinos Kanellopoulos  
Onur Mutlu

ETH Zürich



# Count-Min-Sketch-based Row Tracking to Mitigate RowHammer at Low Cost

**F. Nisa Bostancı**

I. E. Yüksel A. Olgun K. Kanellopoulos Y. C. Tuğrul

A. G. Yağlıkçı M. Sadrosadati O. Mutlu

<https://github.com/CMU-SAFARI/CoMeT>

**SAFARI**

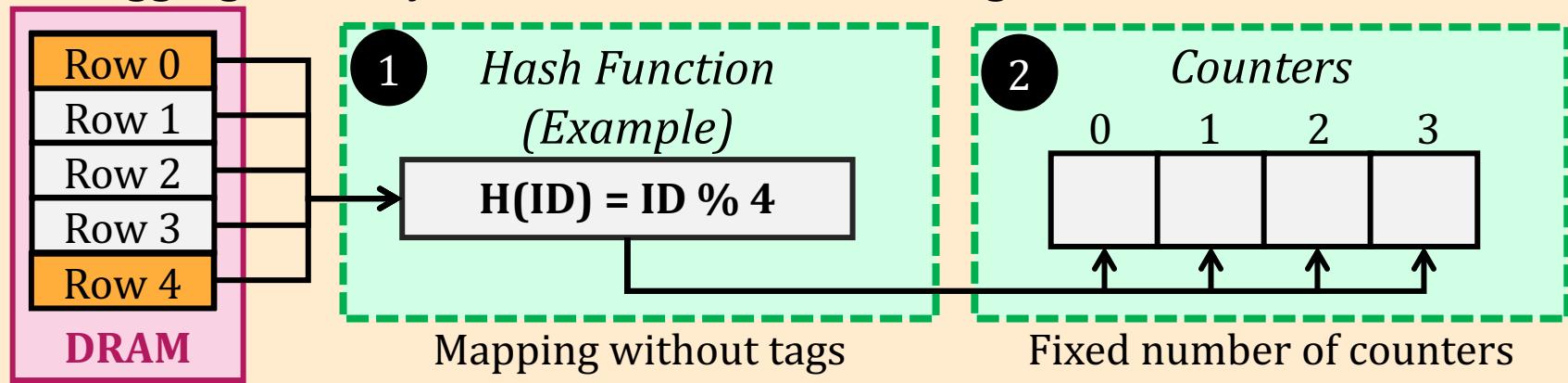
**ETH zürich**

# Key Observation

## Hash-based counters are low-cost:

1. can be implemented with low-cost structures and
2. can aggregate many rows' activation counts together

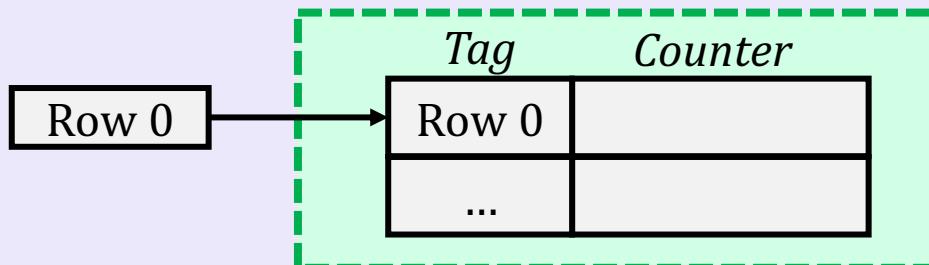
LOW COST



## Tag-based counters are highly accurate:

Each one tracks one row's activation count

HIGH  
ACCURACY



# Key Idea

1

Use **low-cost** and **scalable** hash-based counters to track most DRAM rows' activations with **low area overhead**

2

Use **highly accurate** tag-based counters to track only a small set of DRAM rows to achieve **low performance overhead**

# CoMeT Overview

## Counter Table (CT):

- Maps each DRAM row to a group of low-cost hash-based counters as uniquely as possible by employing the Count-Min Sketch technique
- Triggers a preventive refresh to an aggressor row's victim rows when the aggressor's counter group reaches an activation threshold

Tracks DRAM row activations at low area cost

## Recent Aggressor Table (RAT):

- Allocates highly accurate per-DRAM-row counters for *only* a small set of DRAM rows that are activated many times

Reduces performance penalties by increasing tracking accuracy

# More Operational Details for CoMeT



## CoMeT: Count-Min-Sketch-based Row Tracking to Mitigate RowHammer at Low Cost

F. Nisa Bostancı  
Yahya Can Tuğrul

İsmail Emir Yüksel  
A. Giray Yağlıkçı

Ataberk Olgun  
Mohammad Sadrosadati

Konstantinos Kanellopoulos  
Onur Mutlu

ETH Zürich

*DRAM chips are increasingly more vulnerable to read-disturbance phenomena (e.g., RowHammer and RowPress), where repeatedly accessing DRAM rows causes bitflips in nearby rows due to DRAM density scaling. Under low RowHammer thresholds, existing RowHammer mitigations either incur high area overheads or degrade performance significantly.*

*We propose a new RowHammer mitigation mechanism, CoMeT, that prevents RowHammer bitflips with low area, performance, and energy costs in DRAM-based systems at very*

### 1. Introduction

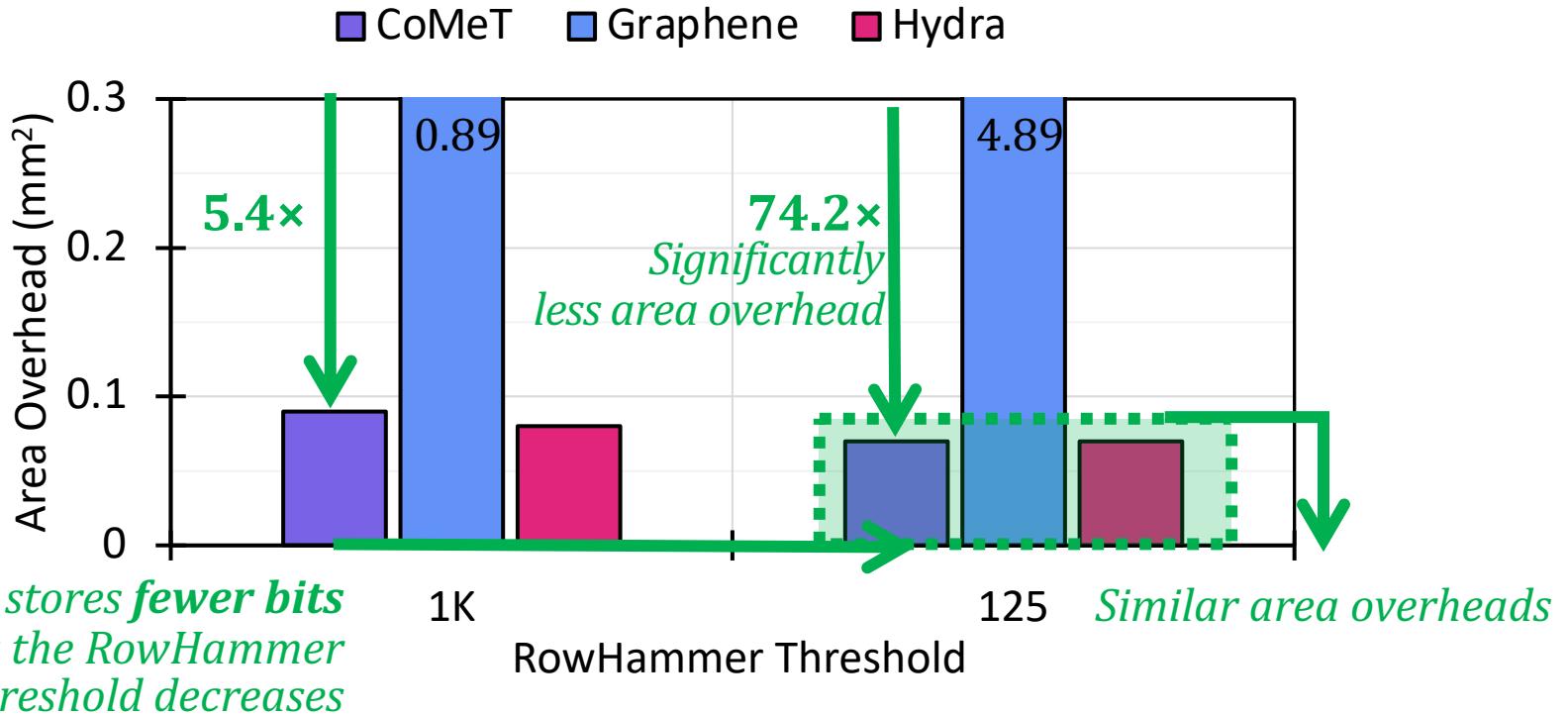
DRAM chips are susceptible to read-disturbance where repeatedly accessing a DRAM row (i.e., *an aggressor row*) can cause bitflips in physically nearby rows (i.e., *victim rows*) [1–13]. RowHammer is a type of read-disturbance phenomenon that is caused by repeatedly opening and closing (i.e., *hammering*) DRAM rows. Modern DRAM chips become more vulnerable to RowHammer as DRAM technology node size becomes smaller [1, 2, 4, 14–19]: the minimum number of row activations needed to cause a bitflip (i.e., *RowHammer threshold*)

<https://arxiv.org/abs/2402.18769>

<https://github.com/CMU-SAFARI/CoMeT>

# Hardware Complexity

- Storage and area overhead analysis: CACTI
- Dual-rank area overhead comparison:



CoMeT incurs a significantly less area overhead than Graphene and a similar area overhead to Hydra

# Evaluation Methodology

- **Performance and energy consumption evaluation:** cycle-level simulations using [Ramulator](#) [Kim+, CAL 2015] and [DRAMPower](#) [Chandrasekar+, DATE 2013]

- **System Configuration:**

**Processor**

1 or 8 cores, 3.6GHz clock frequency,  
4-wide issue, 128-entry instruction window

**DRAM**

DDR4, 1 channel, 2 rank/channel, 4 bank groups,  
4 banks/bank group, 128K rows/bank

**Memory Ctrl.**

64-entry read and write requests queues,  
Scheduling policy: FR-FCFS with a column cap of 16  
Last-Level Cache 8 MiB (single-core), 16 MiB (8-core)

**CoMeT**

*Counter Table:* 4 hash functions 512 counters per hash  
*Recent Aggressor Table:* 128 entries

- **Comparison Points:** 4 state-of-the-art RowHammer mitigations

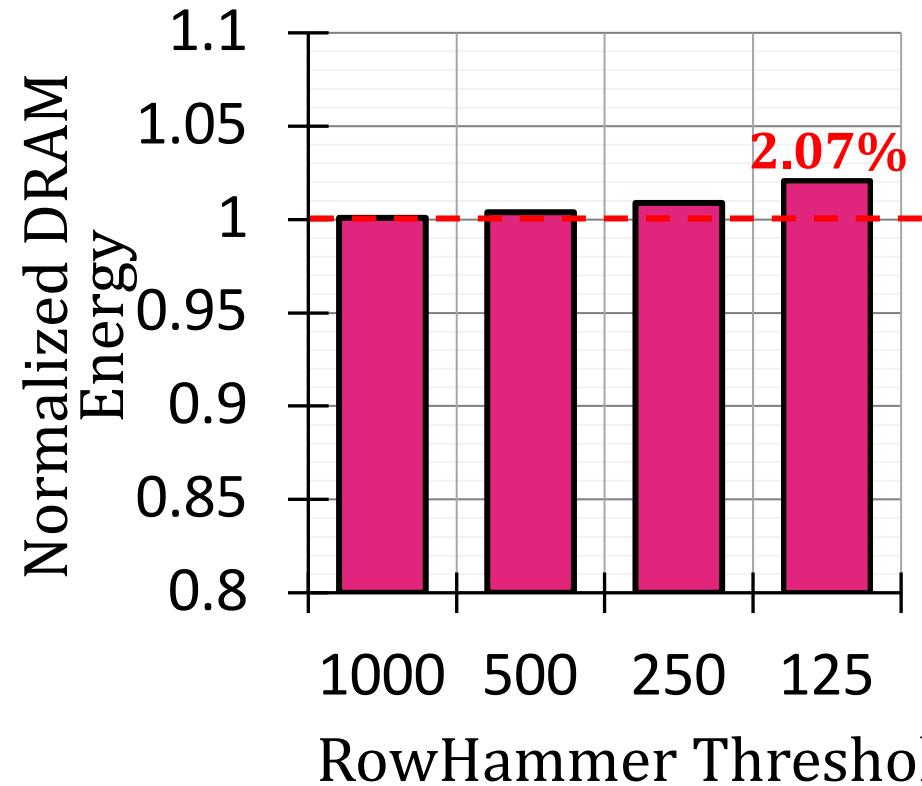
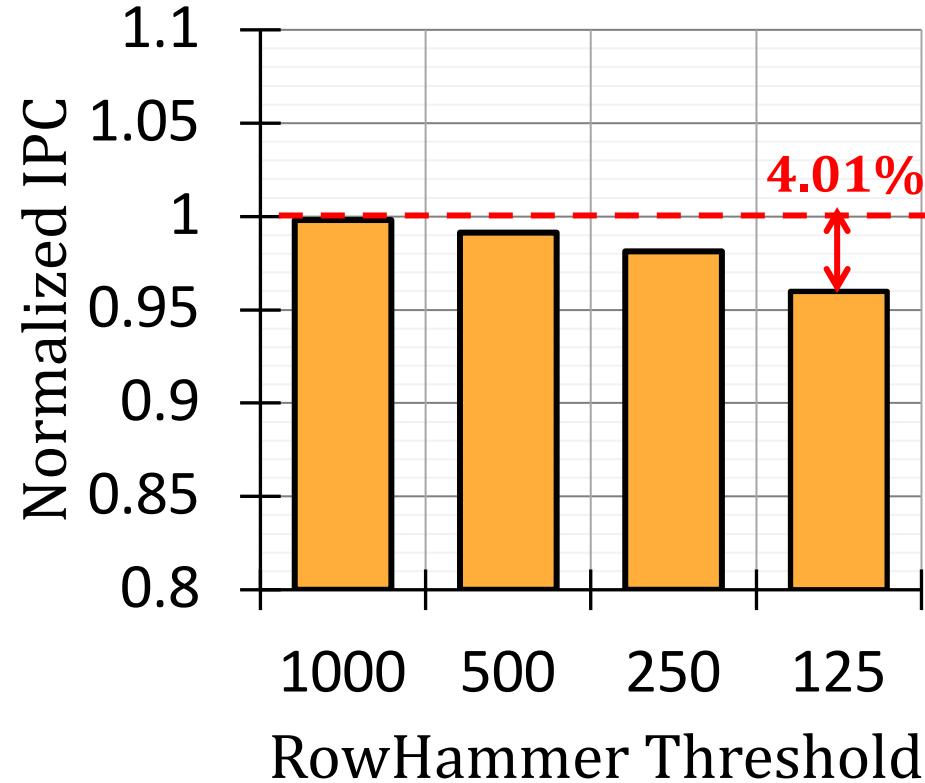
- Graphene (best performing), Hydra (area-optimized best performing),  
Low Processor Chip Area Cost: REGA, PARA

- **Workloads:** 61 single-core applications and 56 8-core workload mixes

- SPEC CPU2006, SPEC CPU2017, TPC, MediaBench, YCSB

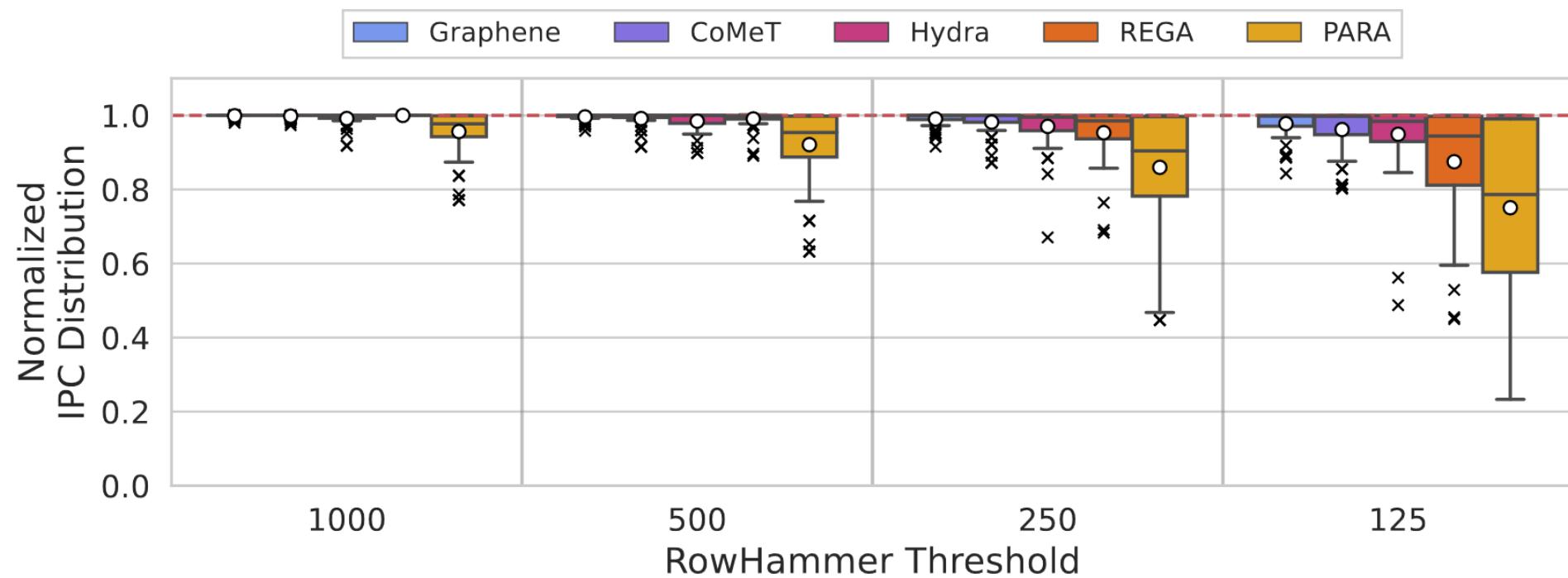
# Performance and DRAM Energy

- Average performance and DRAM energy overheads of CoMeT across **single-core applications**



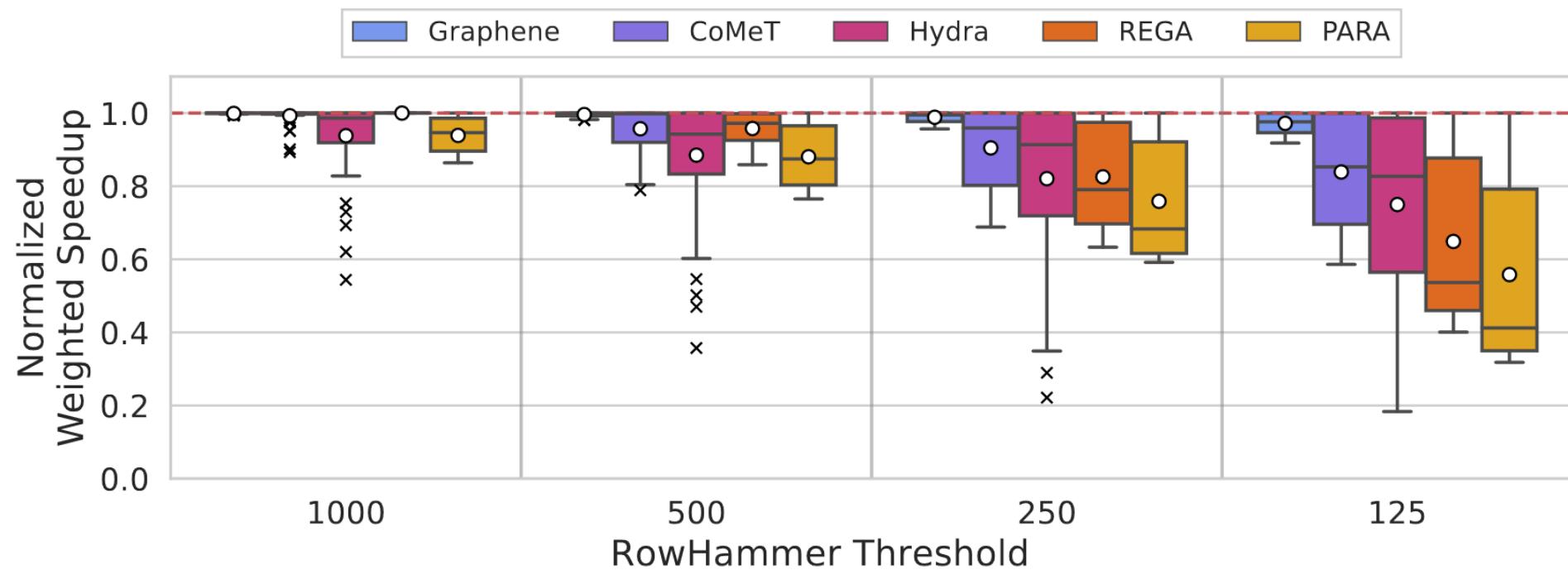
CoMeT prevents bitflips with **very small average performance and DRAM energy overheads** compared to a baseline system with *no* RowHammer mitigation

# Performance Comparison: Single-Core Applications



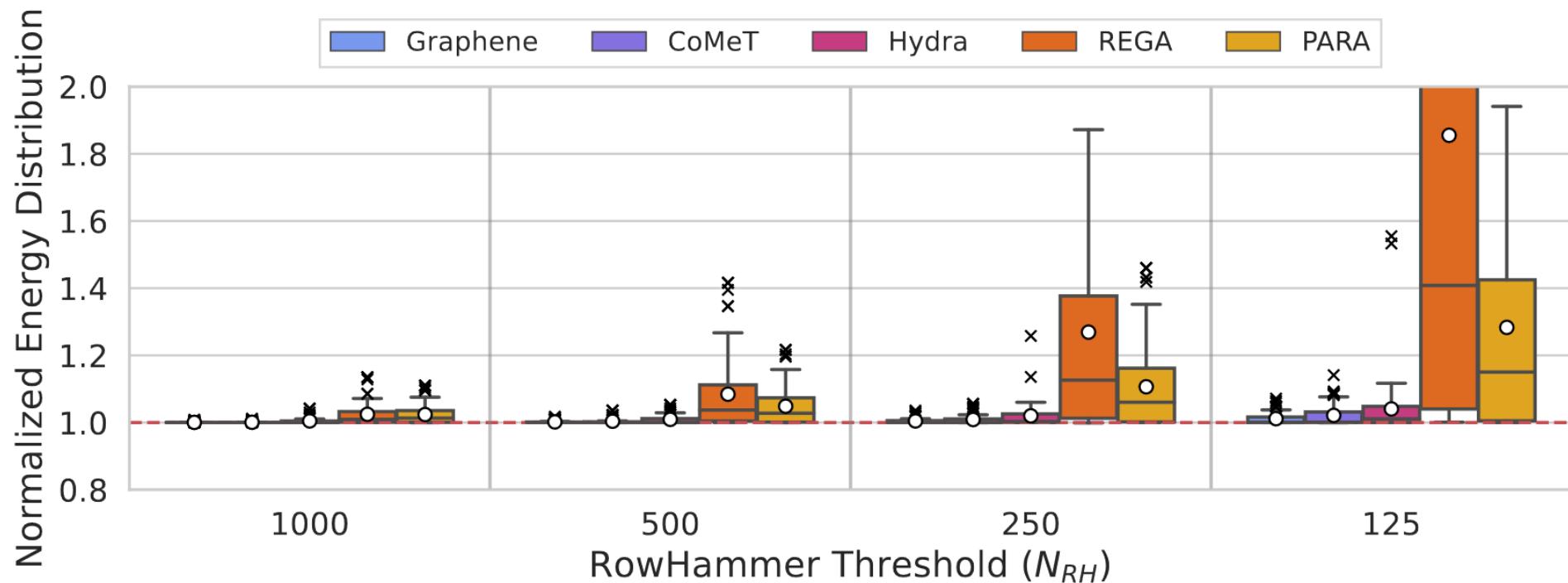
CoMeT incurs a small performance overhead ( $\leq 1.75\%$ ) over Graphene and outperforms Hydra (by up to 39.1%) at all RowHammer thresholds

# Performance Comparison: 8-Core Workloads



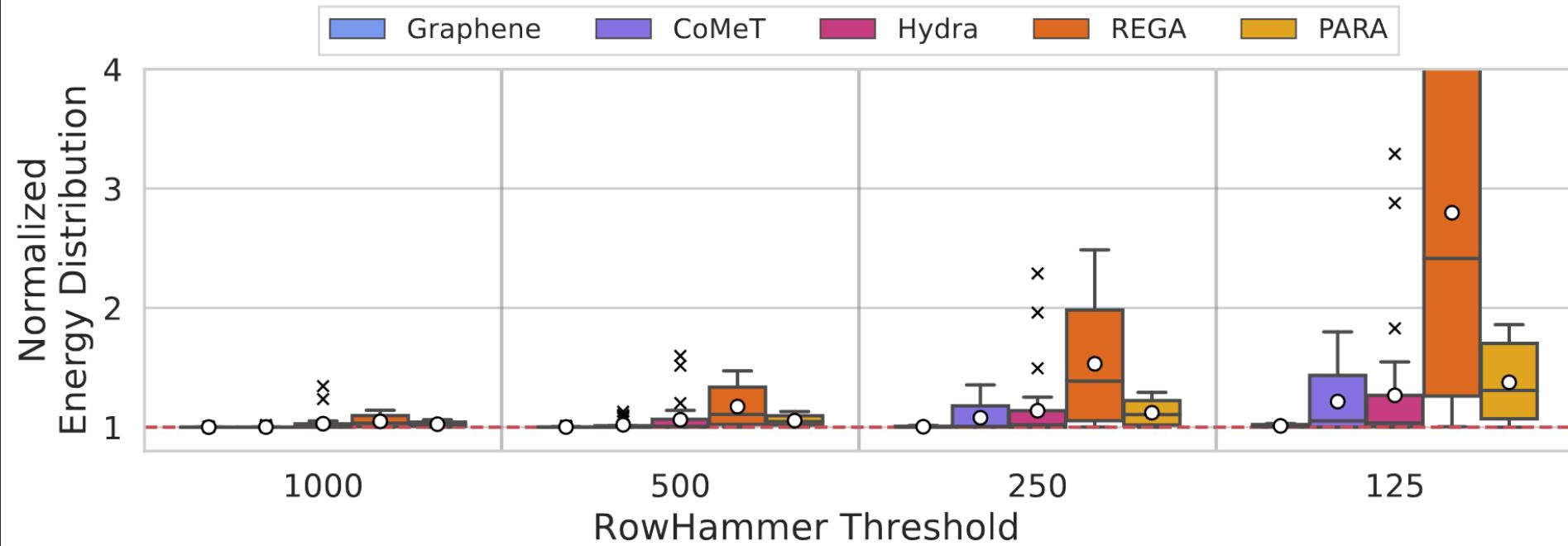
Trends are similar to single-core application evaluation trends

# DRAM Energy Comparison: Single-Core Applications



CoMeT incurs a small DRAM energy overhead (<1%) over Graphene  
and consumes less DRAM energy than Hydra

# DRAM Energy Comparison: 8-Core Workloads



Multicore energy trends are similar to single core energy trends

# More in the Paper

- Security Analysis of CoMeT
- Sensitivity Analysis
  - Counter Table Configurations
  - Recent Aggressor Table Configurations
  - Counter Reset Period and Preventive Refresh Threshold Values
- CoMeT's Performance under Adversarial Workloads
- Comparison against Throttling-Based Mitigation Techniques
- CoMeT's Performance at High RowHammer Thresholds

# More in the Paper

- Security Analysis of CoMeT



## CoMeT: Count-Min-Sketch-based Row Tracking to Mitigate RowHammer at Low Cost

F. Nisa Bostancı      İsmail Emir Yüksel      Ataberk Olgun      Konstantinos Kanellopoulos  
Yahya Can Tuğrul      A. Giray Yağlıkçı      Mohammad Sadrosadati      Onur Mutlu

ETH Zürich

*DRAM chips are increasingly more vulnerable to read-disturbance phenomena (e.g., RowHammer and RowPress), where repeatedly accessing DRAM rows causes bitflips in nearby rows due to DRAM density scaling. Under low RowHammer thresholds, existing RowHammer mitigations either incur high area overheads or degrade performance significantly.*

*We propose a new RowHammer mitigation mechanism, CoMeT, that prevents RowHammer bitflips with low area, performance, and energy costs in DRAM-based systems at very*

### 1. Introduction

DRAM chips are susceptible to read-disturbance where repeatedly accessing a DRAM row (i.e., *an aggressor row*) can cause bitflips in physically nearby rows (i.e., *victim rows*) [1–13]. RowHammer is a type of read-disturbance phenomenon that is caused by repeatedly opening and closing (i.e., *hammering*) DRAM rows. Modern DRAM chips become more vulnerable to RowHammer as DRAM technology node size becomes smaller [1, 2, 4, 14–19]: the minimum number of row activations needed to cause a bitflip (i.e., *RowHammer threshold*)

- Performance Evaluation: <https://arxiv.org/abs/2402.18769> Thresholds

# Conclusion

**Goal:** Prevent RowHammer bitflips with low area, performance, and energy overheads in highly RowHammer-vulnerable DRAM-based systems

**Key Idea:** Use **low-cost** and **scalable hash-based counters** to accurately track DRAM rows

## CoMeT:

- tracks most DRAM rows with **scalable hash-based counters** by employing the **Count-Min-Sketch technique** to achieve a low area cost
- tracks only a small set of DRAM rows that are activated **many times** with **highly accurate per-DRAM-row activation counters** to reduce performance penalties

**Evaluation:** CoMeT achieves a good trade-off between area, performance and energy costs

- incurs **significantly less area overhead (74.2 $\times$ )** compared to the state-of-the-art technique
- **outperforms** the state-of-the-art technique (by up to **39.1%**)

<https://github.com/CMU-SAFARI/CoMeT>

# CoMeT is Open Source and Artifact Evaluated



[CMU-SAFARI / CoMeT](#) Public

Code Issues Pull requests Actions Projects Security Insights

master 1 Branch 0 Tags Go to file Code About

olgunataberk	Update README.md	ff039ed · 3 months ago	28 Commits
configs/ArtifactEvaluation	initial commit	3 months ago	
ext	add ext files	3 months ago	
scripts/artifact	clean stale results	3 months ago	
src	remove unnecessary files	3 months ago	
.gitignore	add scripts for fetching CPU traces and generating Slurm jobs	3 months ago	
CMakeLists.txt	initial commit	3 months ago	
Doxyfile	initial commit	3 months ago	
LICENSE	update README.md and LICENSE	3 months ago	
README.md	Update README.md	3 months ago	

About

CoMeT is a new low-cost RowHammer mitigation that uses Count-Min Sketch-based aggressor row tracking

- Readme
- MIT license
- Activity
- Custom properties
- 6 stars
- 7 watching
- 0 forks

Report repository

Releases

No releases published

<https://github.com/CMU-SAFARI/CoMeT>

# Scalable and Low Overhead DRAM Read Disturbance Mitigation

**Abdullah Giray Yağlıkçı**  
(on behalf of Ataberk Olgun)  
The Future of Memory and Storage  
August 6, 2024



**SAFARI**

**ETH zürich**

# Scalable and Low Overhead DRAM Read Disturbance Mitigation

## Backup Slides

**Abdullah Giray Yağlıkçı**

(on behalf of Ataberk Olgun)

The Future of Memory and Storage

August 6, 2024



**SAFARI**

**ETH zürich**



# ABACuS

## All-Bank Activation Counters for Scalable and Low Overhead RowHammer Mitigation

Ataberk Olgun

Yahya Can Tuğrul

F. Nisa Bostancı

İsmail Emir Yüksel

Haocong Luo

Steve Rhyner

A. Giray Yağlıkçı

Geraldo F. Oliveira

Onur Mutlu

**SAFARI**

**ETH** zürich

# ABACuS Summary

**Problem:** As DRAM becomes more vulnerable to **read disturbance**, existing RowHammer mitigation techniques either prevent bitflips

- at **high area overheads** or
- with prohibitively large performance and energy overheads

**Goal:** Prevent RowHammer bitflips at **low performance, energy, and area cost** especially at **very low RowHammer thresholds** (e.g., 125 aggressor row activations induce a bitflip)

**Key Observation:** Many workloads access the **same row address** in different DRAM banks at **around the same time**

**Key Idea:** Use **one counter** to track the activation count of many **rows** with the **same address** across all DRAM banks

**Key Results:** At very low RowHammer thresholds, ABACuS:

- Induces **small system performance and DRAM energy overhead**
- **Outperforms** the state-of-the-art mitigation (Hydra)
- Takes up **22.7X smaller chip area** than state-of-the-art (Graphene)

# Outline

1. Background & Motivation

---

2. ABACuS: Key Idea and Mechanism

---

3. Evaluation

---

4. Conclusion

# Outline

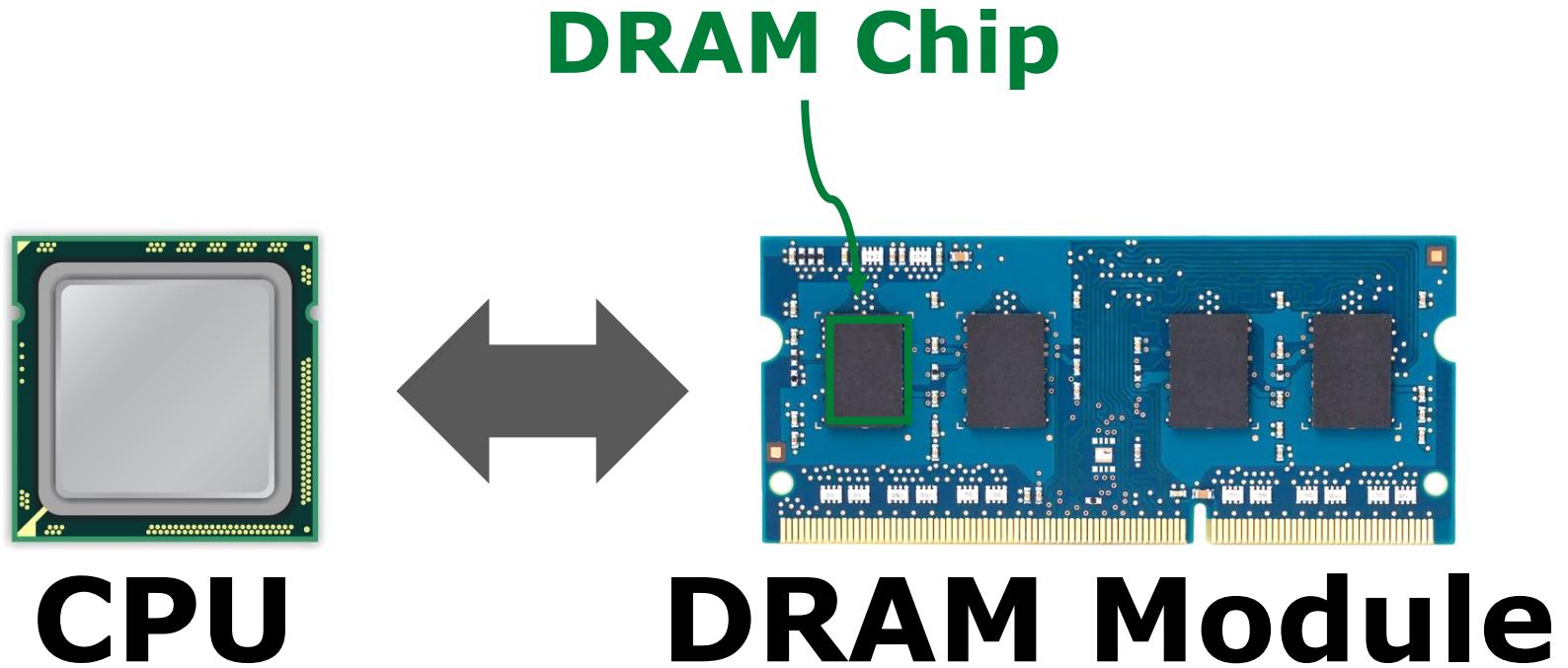
1. Background & Motivation

2. ABACuS: Key Idea and Mechanism

3. Evaluation

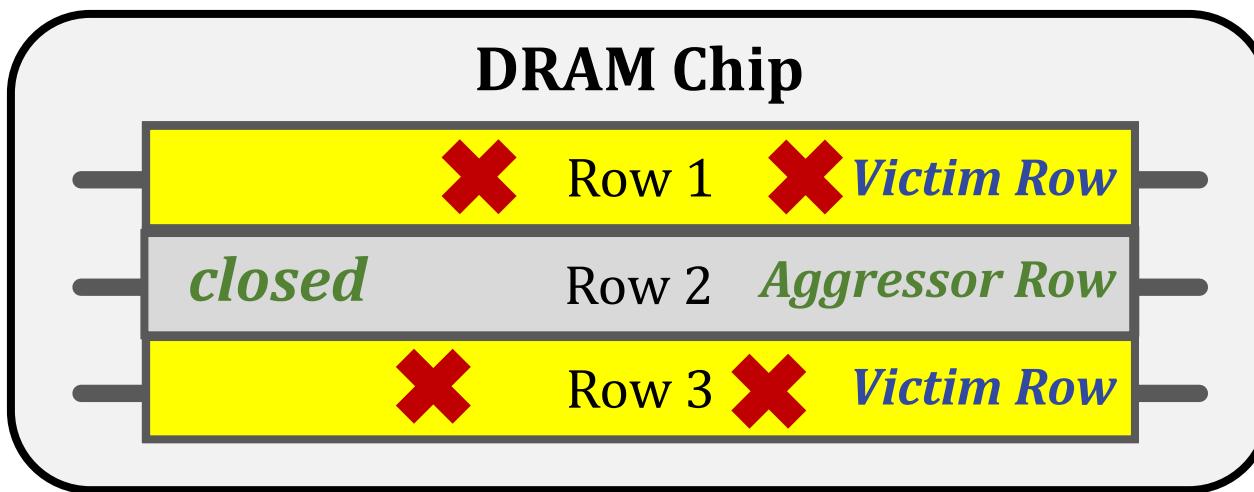
4. Conclusion

# DRAM Read Disturbance (I)



# DRAM Read Disturbance (II)

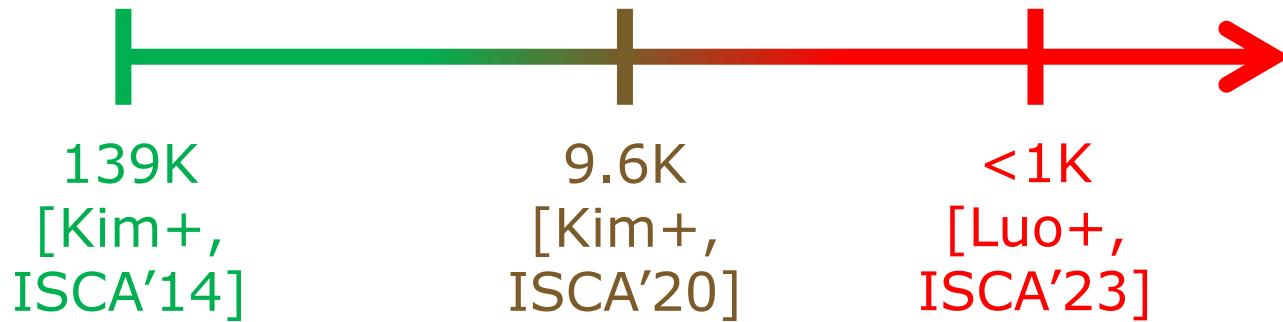
- Read disturbance in DRAM breaks memory isolation
- **Prominent example: RowHammer**



Repeatedly opening (activating) and closing a DRAM row many times causes RowHammer bitflips in adjacent rows

# Read Disturbance Worsens

- Read disturbance bitflips occur at much smaller activation counts
  - More than 100x decrease in less than a decade



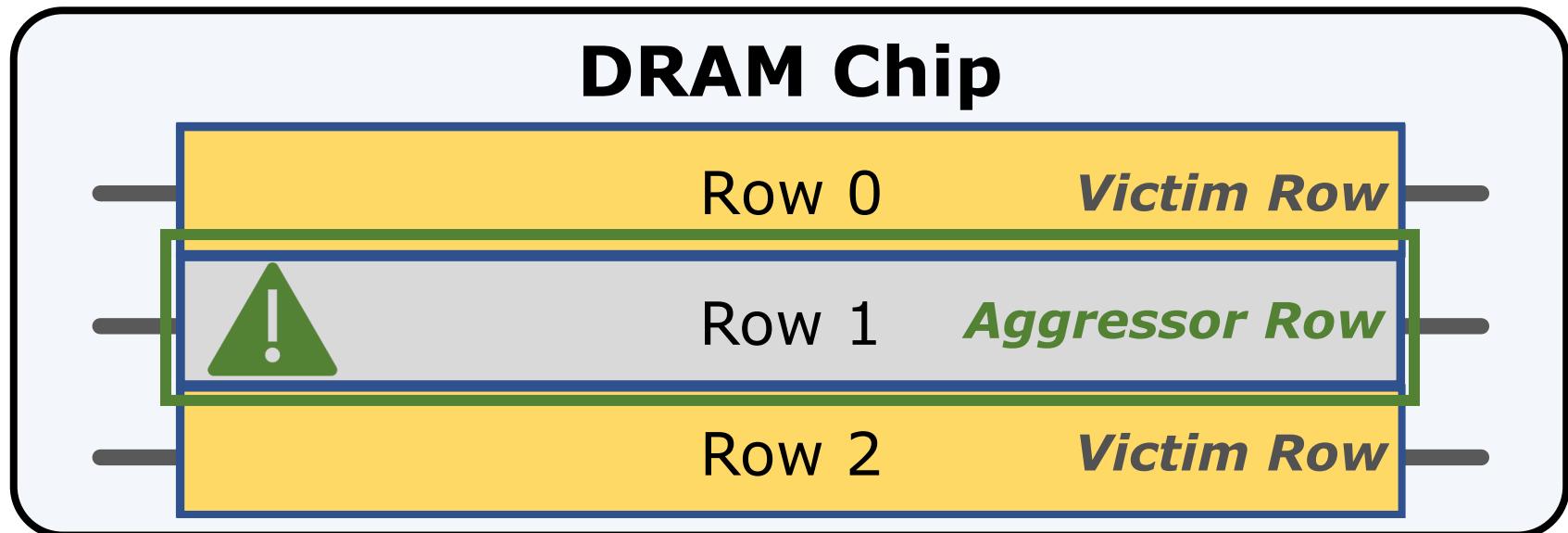
Mitigation techniques against read disturbance attacks need to be **effective** and **efficient** for highly vulnerable systems

# Read Disturbance Mitigation Approaches

There are many ways to mitigate RowHammer bitflips

- More robust DRAM chips and/or error-correcting codes
- Increased refresh rate
- Physical isolation
- Preventive refresh
- Proactive throttling

# Preventive Refresh

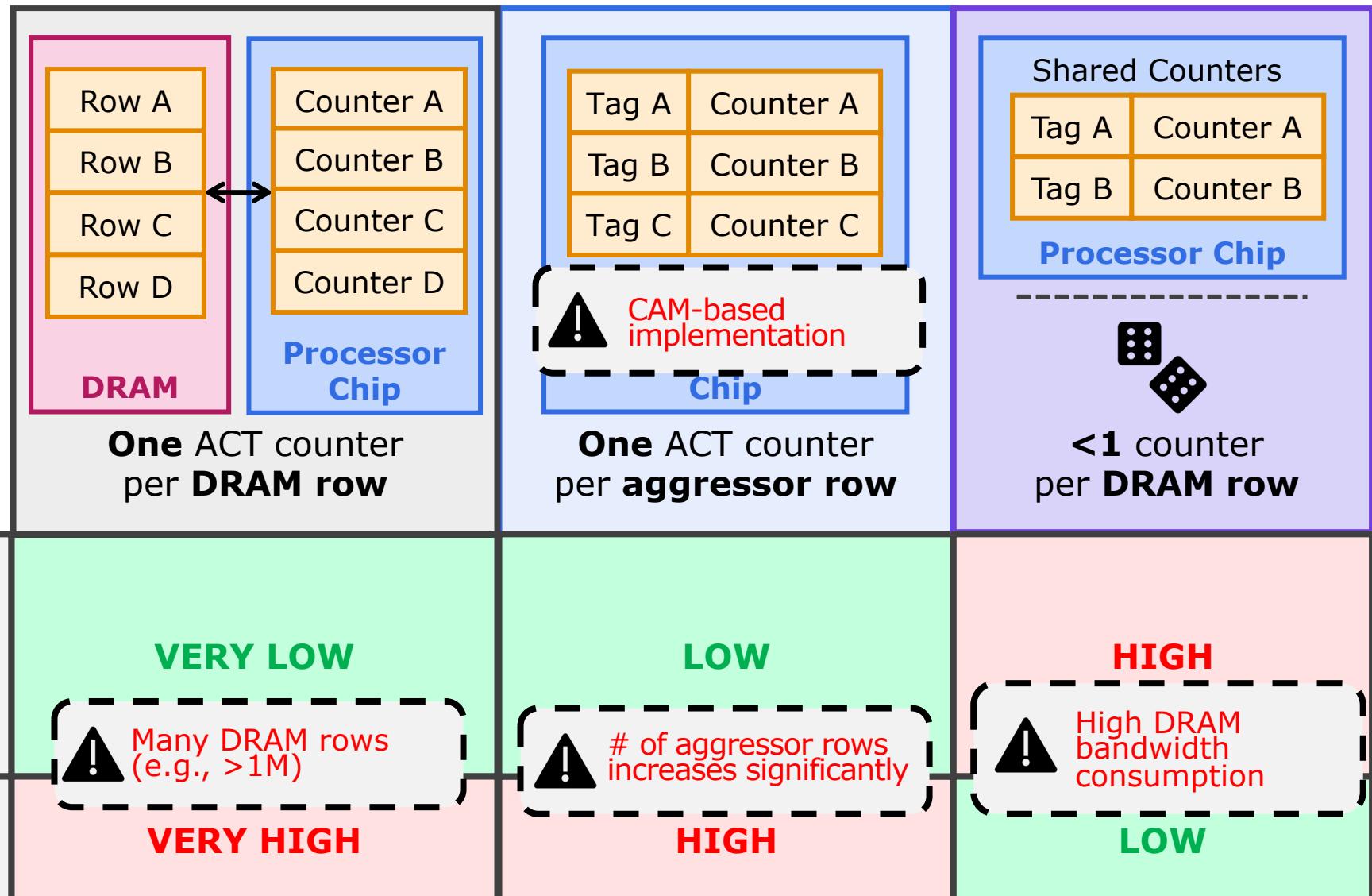


Refreshing potential victim rows  
mitigates read disturbance bitflips



Requires aggressor row activation count  
estimation or tracking

# Preventive-Refresh-Based Mitigations



# Problem & Goal

## Problem

No existing mitigation technique prevents RowHammer bitflips  
at low area, performance and energy costs

## Goal

Prevent RowHammer bitflips  
at low performance, energy, and area cost  
especially at **very low** RowHammer thresholds  
(e.g., 125 aggressor row activations induce a bitflip)

# Outline

1. Background & Motivation

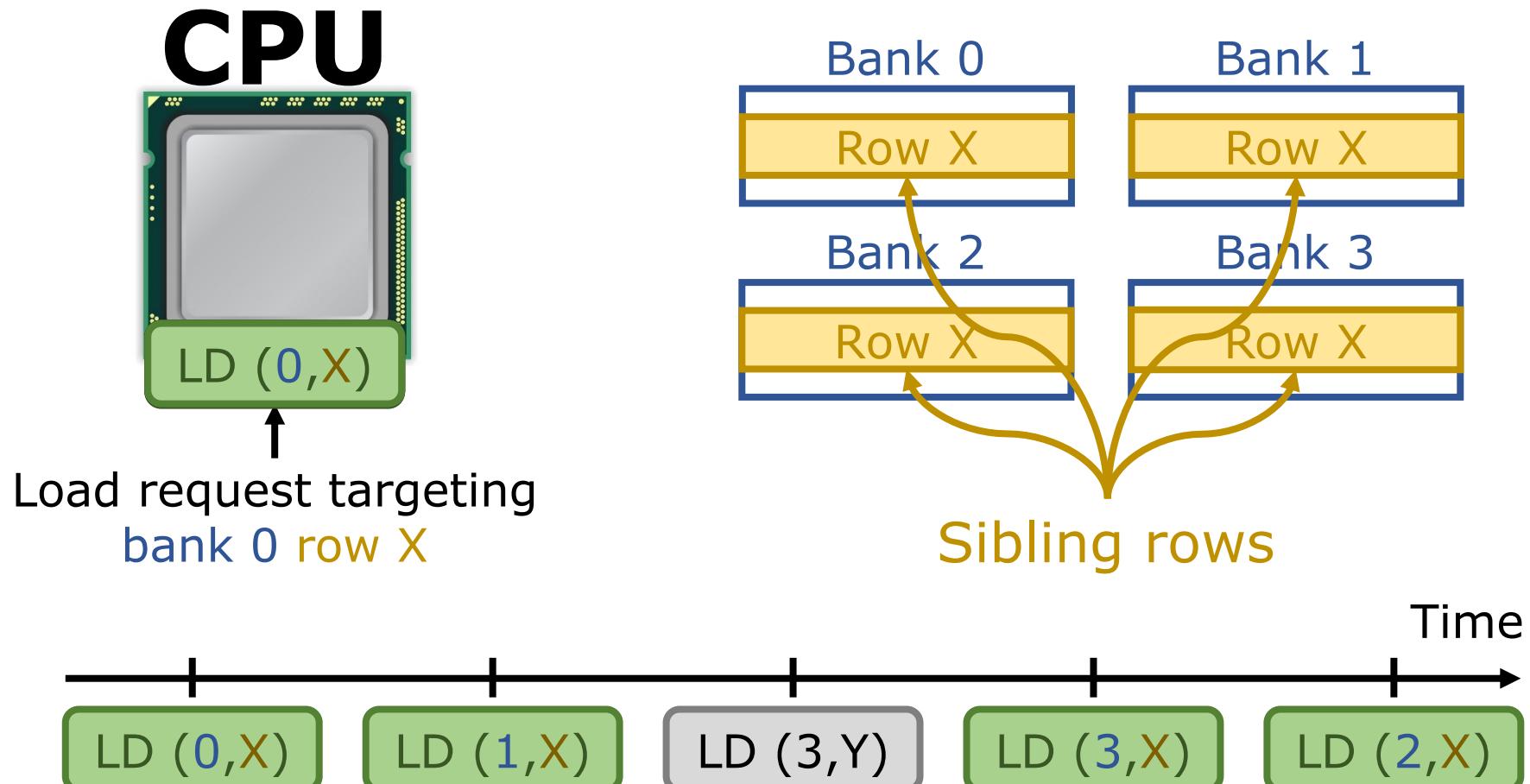
2. ABACuS: Key Idea and Mechanism

3. Evaluation

4. Conclusion

# Key Observation

Many workloads access the **same row address** in different **banks** at **around the same time**



# Explanation for the Key Observation

1

## Spatial locality in memory accesses

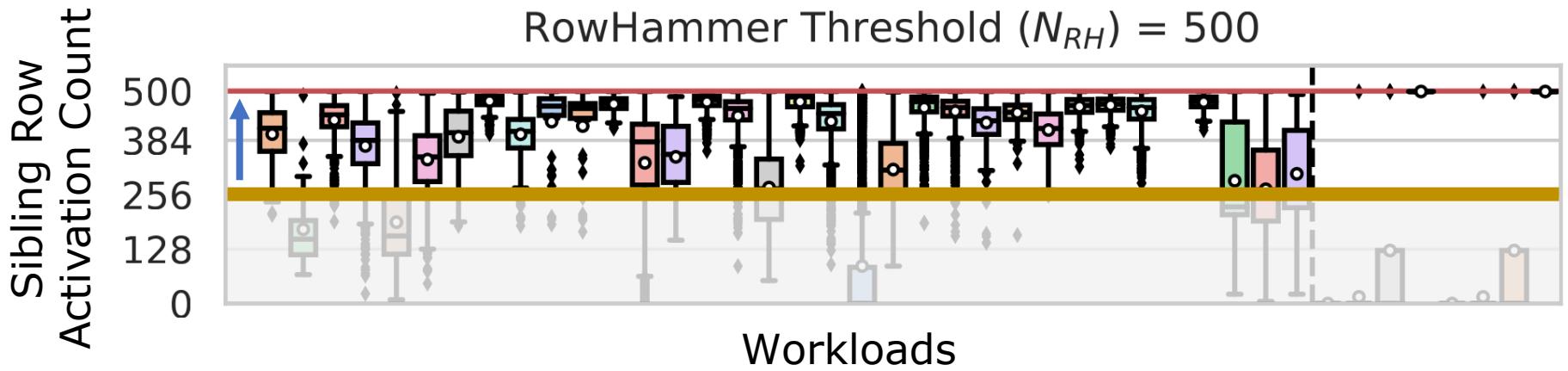
- A program tends to access neighboring cache blocks at around the same time
- e.g., a streaming access to an array

2

## Modern physical → DRAM address mappings

- Place neighboring cache blocks into different banks, but into the same row
- Leverage DRAM bank-level parallelism for higher-throughput DRAM access

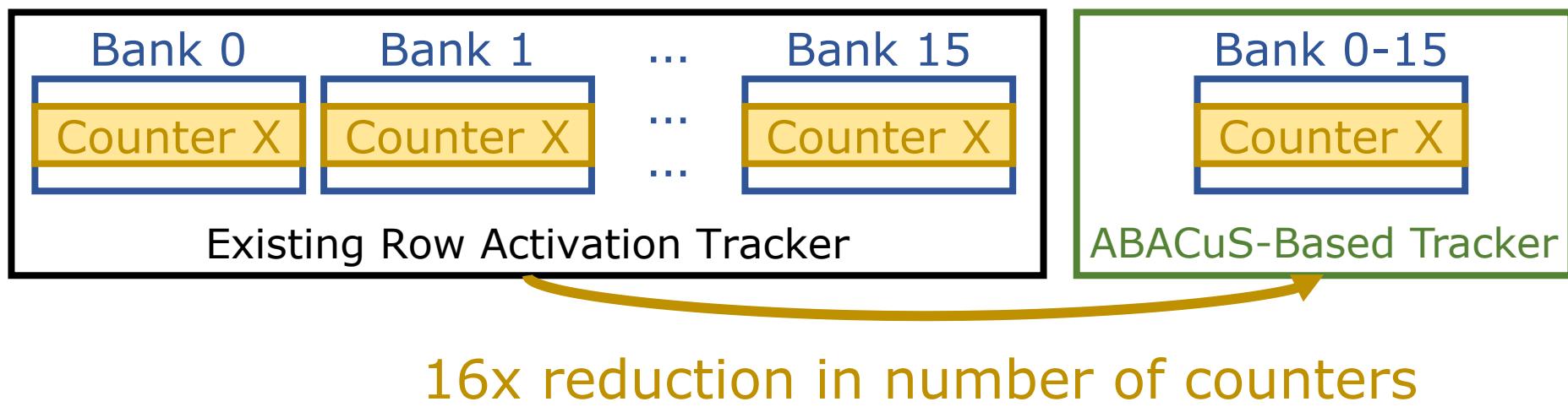
# Sibling Row Activation Count



If a row is activated RowHammer threshold ( $N_{RH}$ ) times its siblings are likely activated more than  $N_{RH}/2$  times

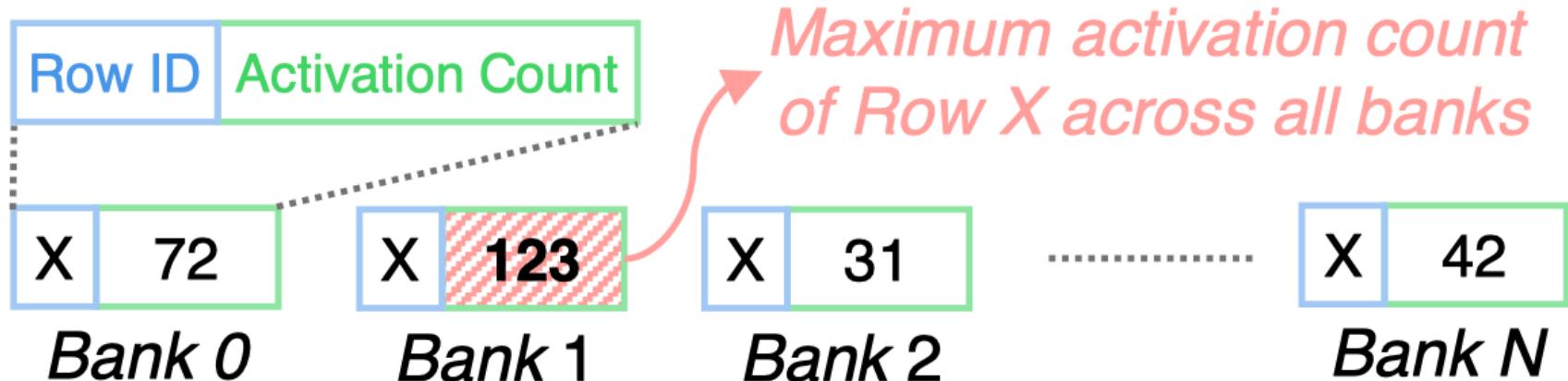
# Key Idea

- There are **many** (e.g., 16) banks in a DRAM chip
  - Newer DRAM standards (DDR5) have more (32) banks
  - # of activation counters **linearly increases** with # of banks
- Sibling rows **have similar activation counts**
- Have **one counter** for all **siblings**
  - Reduce the number of counters by a **factor of the number of banks**



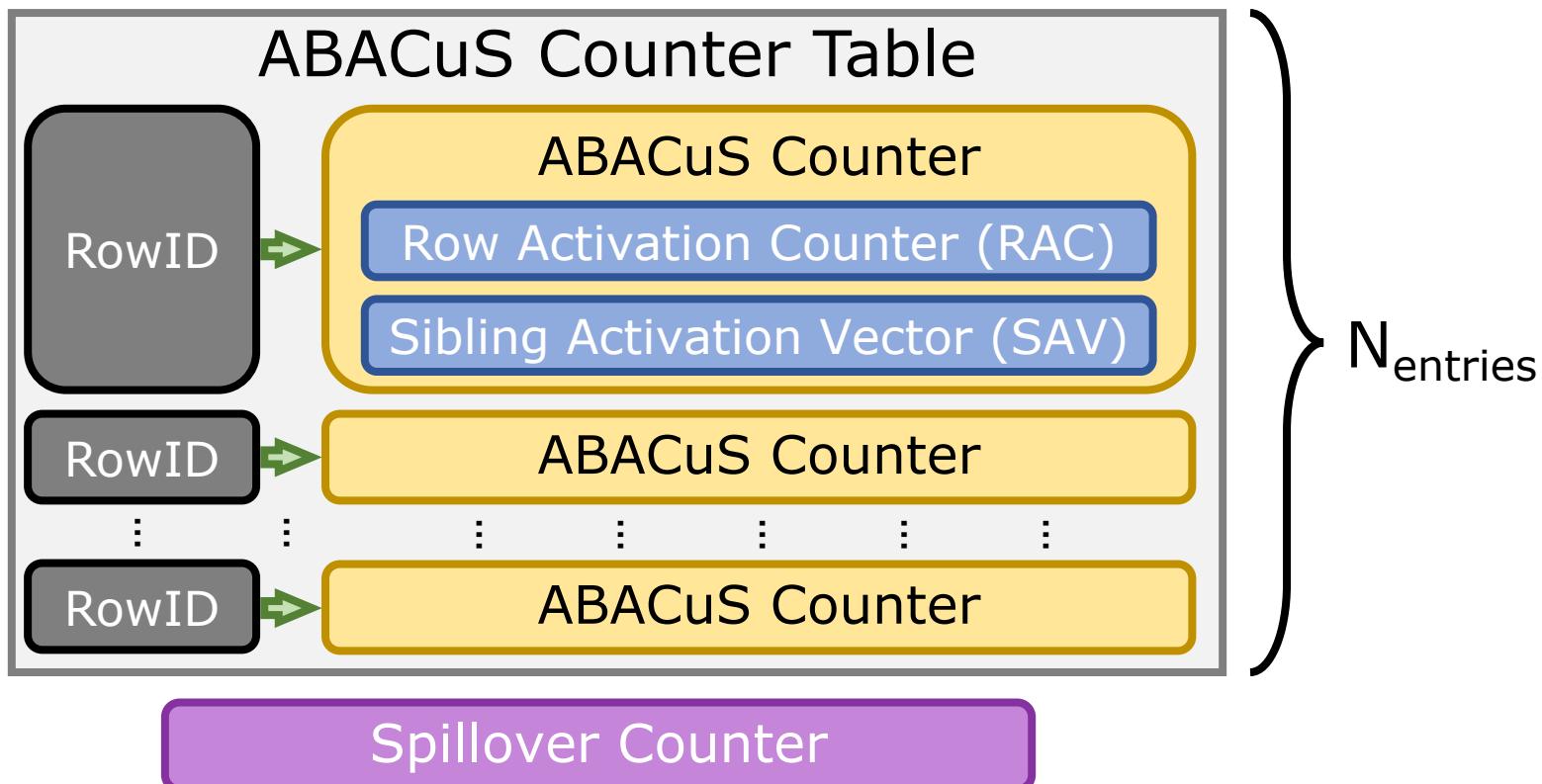
# ABAxCuS: Overview

Track the **maximum** (worst) activation count across all sibling rows using one counter

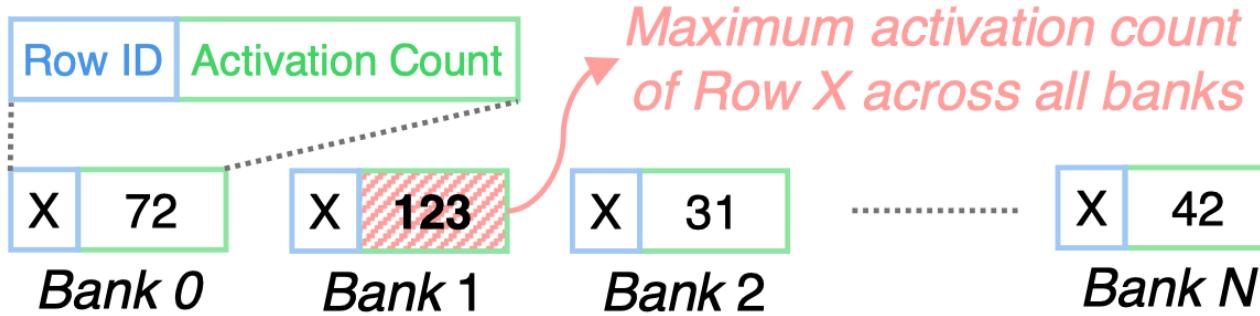


# Key Components

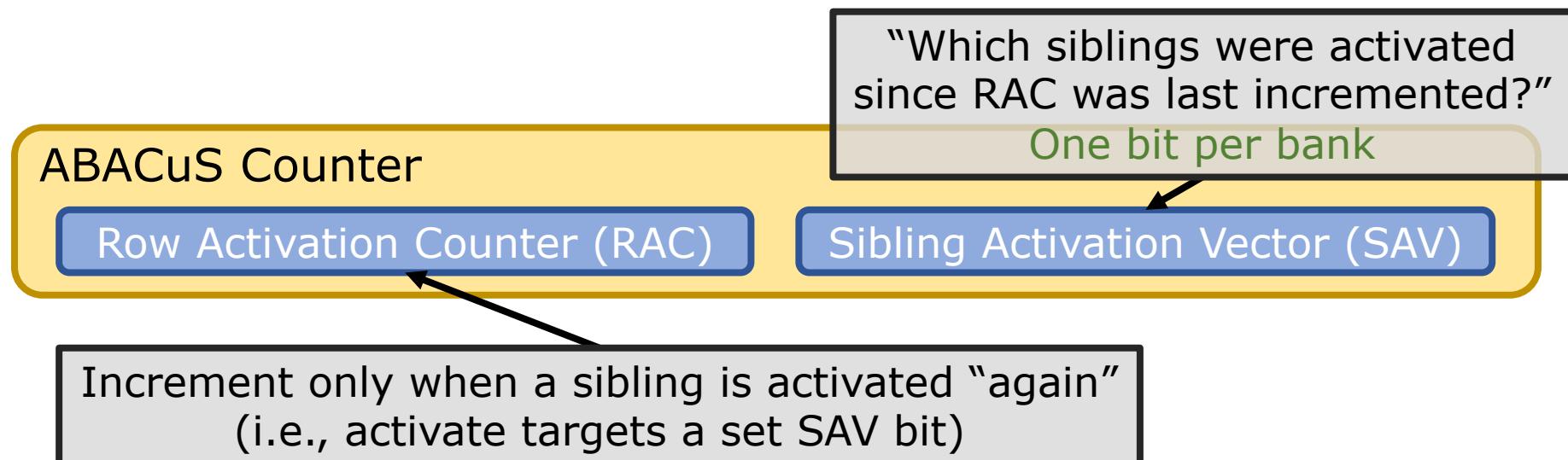
- Adopt a frequent item counting algorithm
  - Area-efficient, fewer counters to track more DRAM rows
  - ABACuS is compatible with other counting methods



# Operation



- The RAC always stores the maximum activation count
  - Store small additional information in SAV



# Outline

1. Background & Motivation

2. ABACuS: Key Idea and Mechanism

3. Evaluation

4. Conclusion

# Evaluation Methodology

- **Performance and energy consumption evaluation:**

Cycle-level simulations using **Ramulator** [Kim+, CAL 2015] and **DRAMPower** [Chandrasekar+, DATE 2013]

- **System Configuration:**

**Processor** 1 or 8 cores, 3.6GHz clock frequency,

4-wide issue, 128-entry instruction window

**DRAM** DDR4, 1 channel, 2 rank/channel, 4 bank groups,  
4 banks/bank group, 128K rows/bank, 3200 MT/s

**Memory Ctrl.** 64-entry read and write requests queues,

Scheduling policy: FR-FCFS with a column cap of 16

Last-Level Cache 2 MiB (single-core), 16 MiB (8-core)

- **Comparison Points:** 4 state-of-the-art RowHammer mitigations

- Graphene (best performing), Hydra (area-optimized best performing),  
Low Processor Chip Area Cost: REGA, PARA

- **Workloads:** 62 1- & 8-core workloads

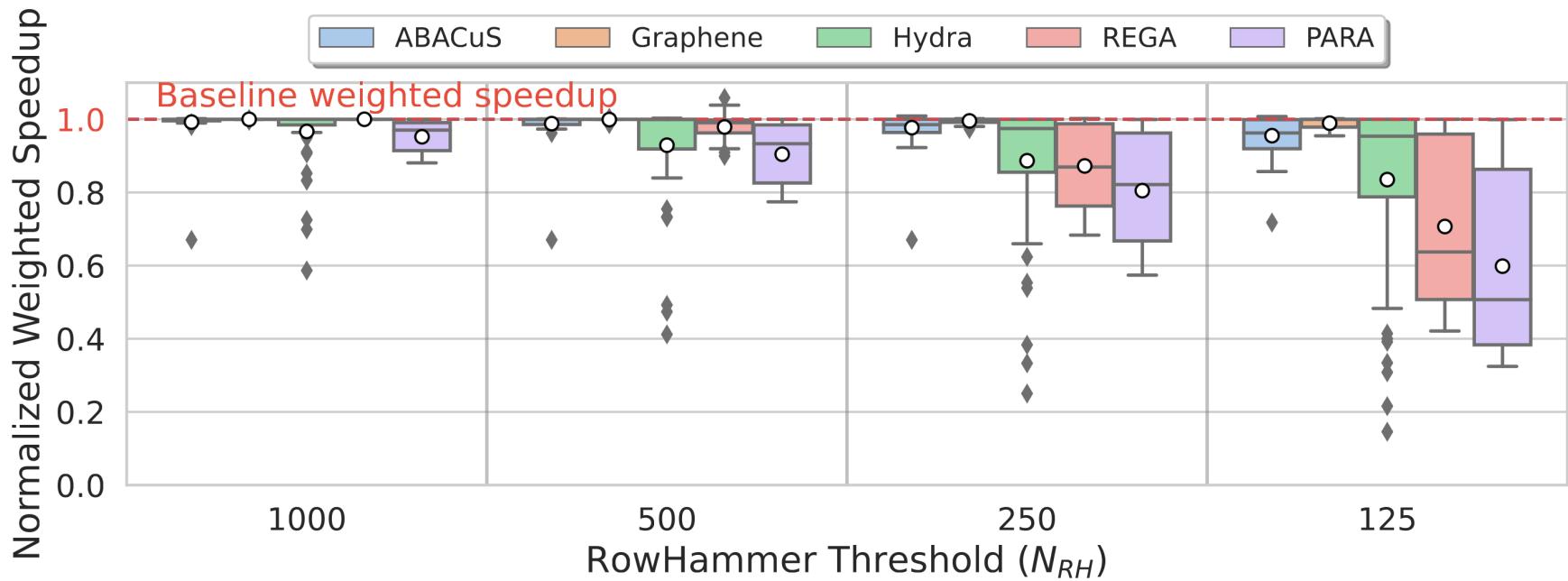
- SPEC CPU2006, SPEC CPU2017, TPC, MediaBench, YCSB

# Single-Core Performance and Energy

Small 0.6% average performance overhead at  $nRH = 1000$   
1.5% at  $nRH = 125$

ENERGY

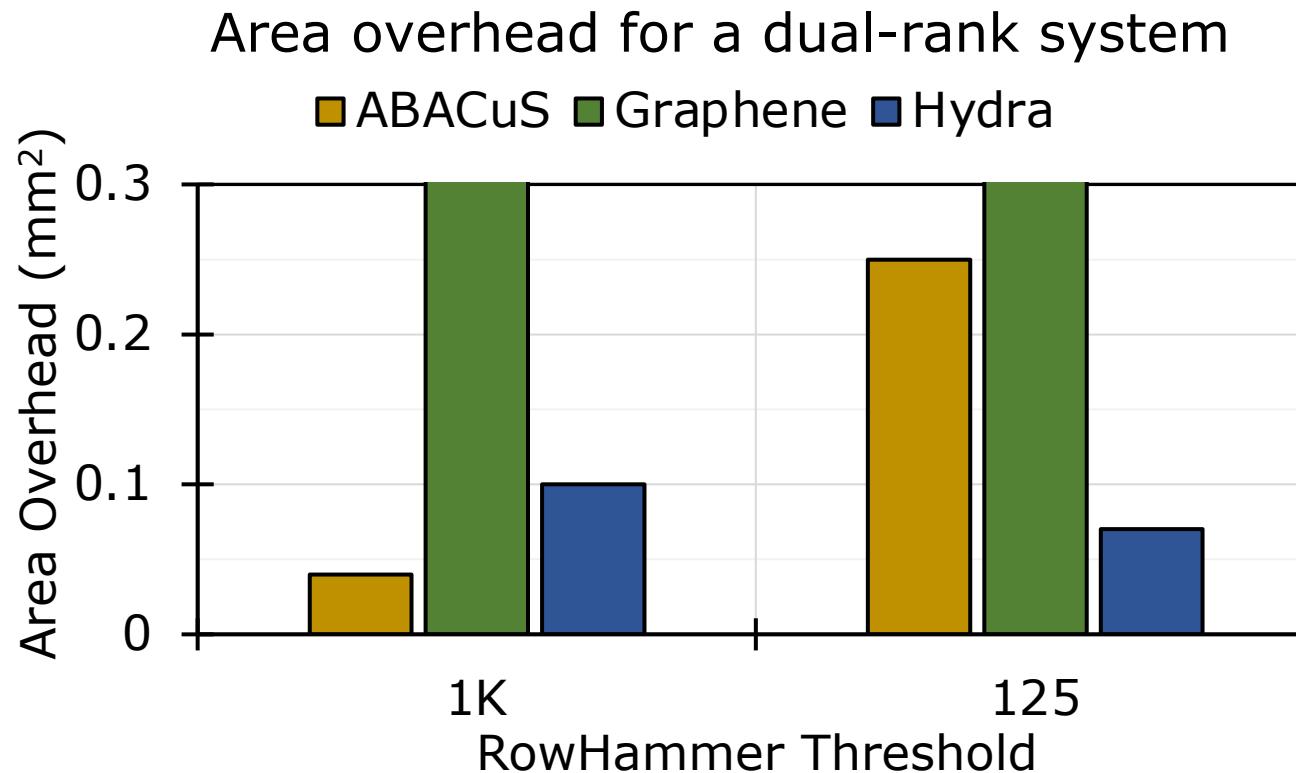
# Multi-Core Performance



# Storage and Area Overhead

ABACuS takes up **18.93 KiB** at **1K** and **151.41 KiB** at **125 RowHammer threshold**

Area overhead analysis: **CACTI**



# More in the Paper

- More motivational analysis
- Multi-core performance & energy results
- Performance under adversarial workloads
  - Alternative ABACuS design
- Performance & energy sensitivity to:
  - Blast radius
  - Number of ABACuS counters
  - Number of banks
- Circuit area, latency, energy, and power
- Security proof

# Outline

1. Background & Motivation

2. ABACuS: Key Idea and Mechanism

3. Evaluation

4. Conclusion

# ABACuS Summary

**Goal:** Prevent RowHammer bitflips at **low performance, energy, and area cost** especially at **very low RowHammer thresholds** (e.g., 125 aggressor row activations induce a bitflip)

**Key Observation:** Many workloads access the **same row address** in different DRAM banks at **around the same time**

**Key Idea:** Use **one counter** to track the activation count of many **rows** with the **same address** across all DRAM banks

**Key Results:** At very low RowHammer thresholds, ABACuS:

- Induces **small system performance and DRAM energy overhead**
- **Outperforms** the state-of-the-art mitigation (Hydra)
- Takes up **22.7X smaller chip area** than state-of-the-art (Graphene)

# Extended Version on arXiv

<https://arxiv.org/pdf/2310.09977.pdf>

arXiv > cs > arXiv:2310.09977

Search... All fields Help | Advanced Search

Computer Science > Cryptography and Security

[Submitted on 15 Oct 2023]

## ABACuS: All-Bank Activation Counters for Scalable and Low Overhead RowHammer Mitigation

Ataberk Olgun, Yahya Can Tugrul, Nisa Bostancı, Ismail Emir Yuksel, Haocong Luo, Steve Rhyner, Abdullah Giray Yaglikci, Geraldo F. Oliveira, Onur Mutlu

We introduce ABACuS, a new low-cost hardware-counter-based RowHammer mitigation technique that performance-, energy-, and area-efficiently scales with worsening RowHammer vulnerability. We observe that both benign workloads and RowHammer attacks tend to access DRAM rows with the same row address in multiple DRAM banks at around the same time. Based on this observation, ABACuS's key idea is to use a single shared row activation counter to track activations to the rows with the same row address in all DRAM banks. Unlike state-of-the-art RowHammer mitigation mechanisms that implement a separate row activation counter for each DRAM bank, ABACuS implements fewer counters (e.g., only one) to track an equal number of aggressor rows.

Our evaluations show that ABACuS securely prevents RowHammer bitflips at low performance/energy overhead and low area cost. We compare ABACuS to four state-of-the-art mitigation mechanisms. At a near-future RowHammer threshold of 1000, ABACuS incurs only 0.58% (0.77%) performance and 1.66% (2.12%) DRAM energy overheads, averaged across 62 single-core (8-core) workloads, requiring only 9.47 KiB of storage per DRAM rank. At the RowHammer threshold of 1000, the best prior low-area-cost mitigation mechanism incurs 1.80% higher average performance overhead than ABACuS, while ABACuS requires 2.50X smaller chip area to implement. At a future RowHammer threshold of 125, ABACuS performs very similarly to (within 0.38% of the performance of) the best prior performance- and energy-efficient RowHammer mitigation mechanism while requiring 22.72X smaller chip area. ABACuS is freely and openly available at [this https URL](https://arxiv.org/pdf/2310.09977.pdf).

Access Paper:

- Download PDF
- PostScript
- Other Formats

(cc) BY

Current browse context: cs.CR  
< prev | next >  
new | recent | 2310

Change to browse by:  
cs  
cs.AR

References & Citations  
[NASA ADS](#)  
[Google Scholar](#)  
[Semantic Scholar](#)

Export BibTeX Citation

Bookmark

# ABACuS is Open Source

<https://github.com/CMU-SAFARI/ABACuS>

The screenshot shows the GitHub interface for the ABACuS repository. At the top, there's a navigation bar with links for Code, Issues, Pull requests, Actions, Projects, Security, Insights, and Settings. Below the navigation bar, the repository name 'ABACuS' is displayed along with its status as 'Public'. On the right side of the header, there are buttons for Edit Pins, Unwatch (with 4 notifications), Fork (with 0 forks), and Starred (with 3 stars). The main content area shows a list of commits from the 'main' branch. Each commit includes the author's icon, the author's name, the commit message, the commit hash, the time of the commit, and the number of commits. The commits listed are:

- olgunataberk add verilog sources and update readme ef1c89c yesterday 8 commits
- abacus\_cacti add abacus cacti sources yesterday
- abacus\_verilog add verilog sources and update readme yesterday
- configs/ABACUS Initial commit 3 days ago
- ext Initial commit 3 days ago
- scripts Initial commit 3 days ago
- src Initial commit 3 days ago
- .gitignore Initial commit 3 days ago
- CMakeLists.txt Initial commit 3 days ago
- Doxmyfile Initial commit 3 days ago

On the right side of the commit list, there's an 'About' section with a detailed description of the RowHammer mitigation mechanism, a link to the USENIX Security'24 paper, and various repository statistics like Readme, MIT license, Activity, stars, watching, forks, and a 'Report repository' link.

# **SAFARI**



# ABACuS

## All-Bank Activation Counters for Scalable and Low Overhead RowHammer Mitigation

Ataberk Olgun

Yahya Can Tuğrul

F. Nisa Bostancı

İsmail Emir Yüksel

Haocong Luo

Steve Rhyner

A. Giray Yağlıkçı

Geraldo F. Oliveira

Onur Mutlu

**SAFARI**

**ETH** zürich



# Count-Min-Sketch-based Row Tracking to Mitigate RowHammer at Low Cost

**F. Nisa Bostancı**

I. E. Yüksel A. Olgun K. Kanellopoulos Y. C. Tuğrul  
A. G. Yağlıkçı M. Sadrosadati O. Mutlu

<https://github.com/CMU-SAFARI/CoMeT>

**SAFARI**

**ETH** zürich

# Executive Summary

**Problem:** As DRAM becomes more vulnerable to read disturbance, existing RowHammer mitigation techniques either prevent bitflips

- (1) at low performance cost but with **high area overheads** or
- (2) at low area cost but **with prohibitively large performance and energy overheads**

**Goal:** Prevent RowHammer bitflips **with low area, performance, and energy overheads** in highly RowHammer-vulnerable DRAM-based systems

**Key Idea:** Use **low-cost** and **scalable hash-based counters** to accurately track DRAM rows

**CoMeT:**

- tracks most DRAM rows with **scalable hash-based counters** by employing **the Count-Min-Sketch technique** to achieve a low area cost
- tracks only **a small set of DRAM rows that are activated many times** with **highly accurate per-DRAM-row activation counters** to reduce performance penalties

**Evaluation:** CoMeT achieves a good trade-off between area, performance and energy costs

- **incurs significantly less area overhead (74.2 $\times$ )** compared to the state-of-the-art technique
- **outperforms** the state-of-the-art technique (by up to **39.1%**)

<https://github.com/CMU-SAFARI/CoMeT>

# Outline

Background and Problem

Goal and Key Idea

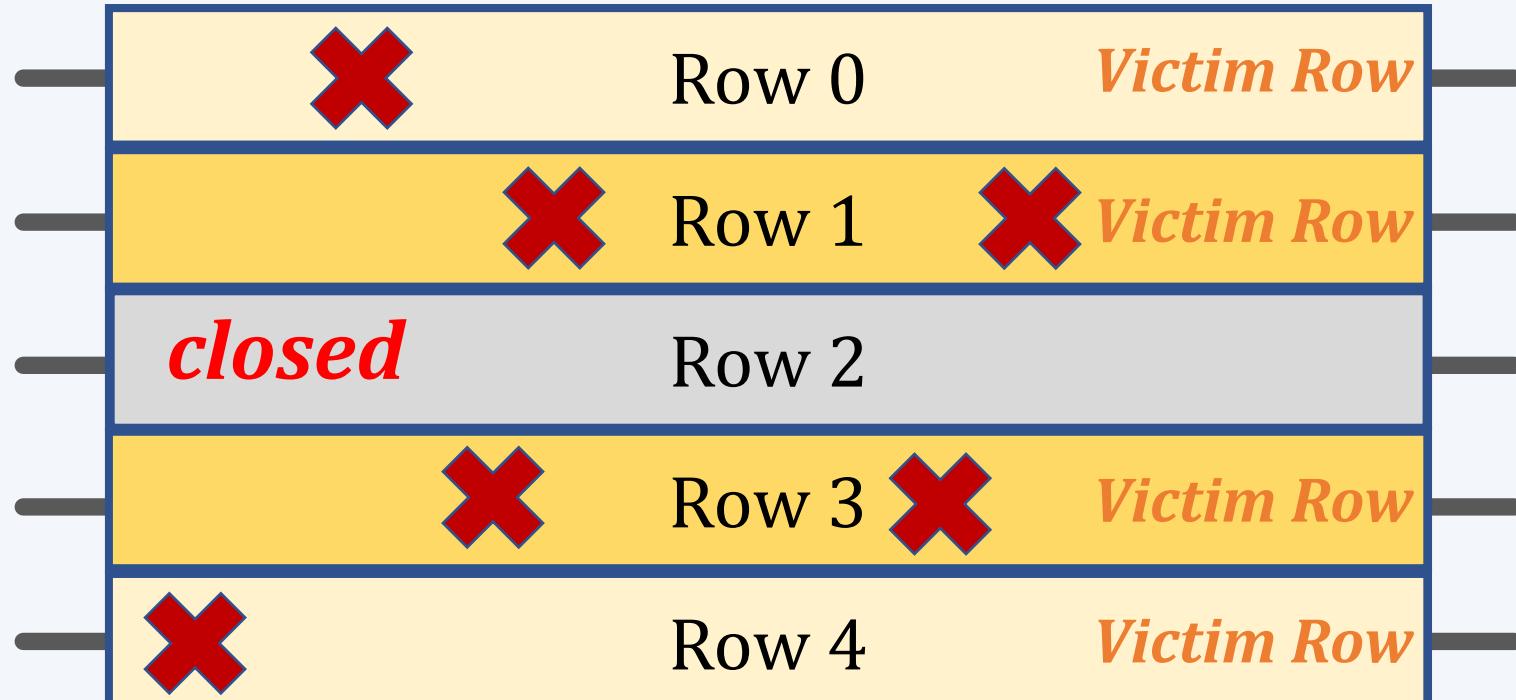
CoMeT: Count-Min-Sketch-based Row Tracking

Evaluation

Conclusion

# Read Disturbance Vulnerabilities (I)

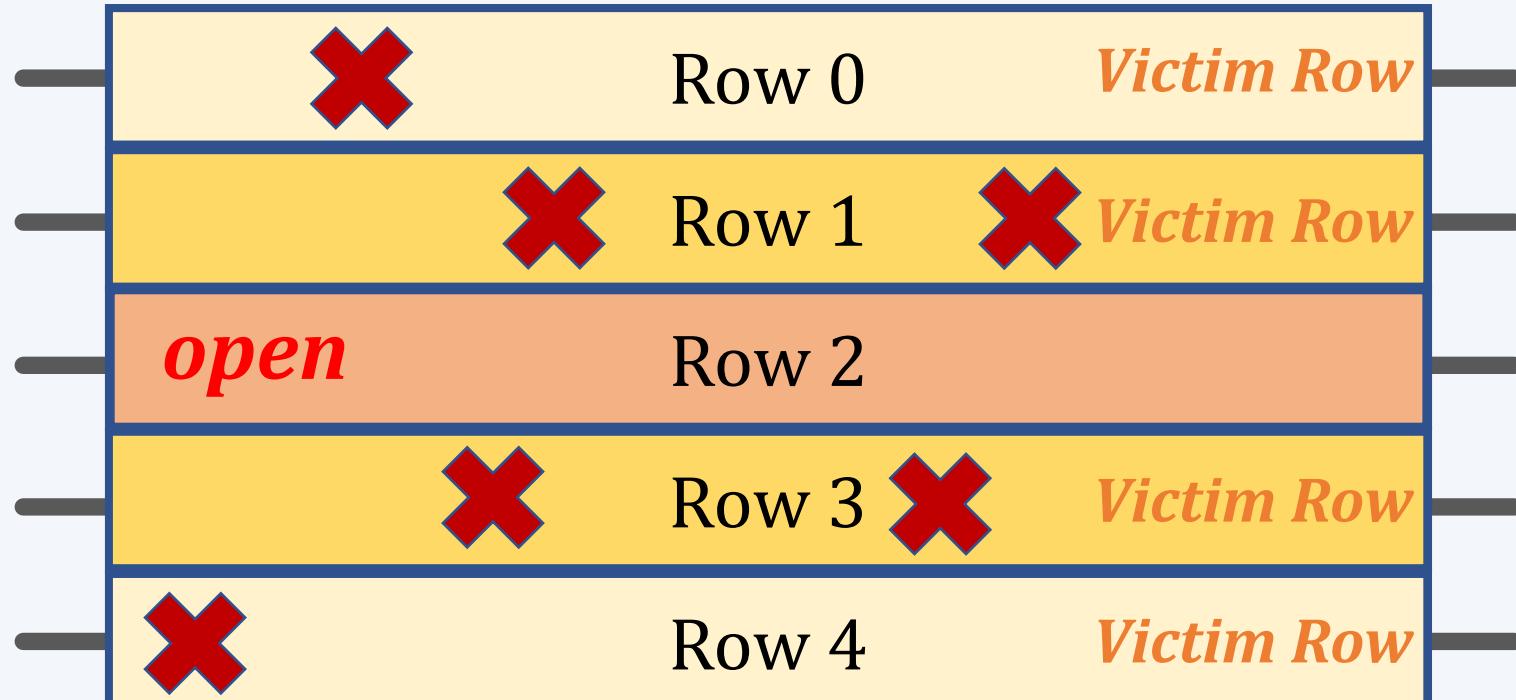
DRAM Subarray



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

# Read Disturbance Vulnerabilities (I)

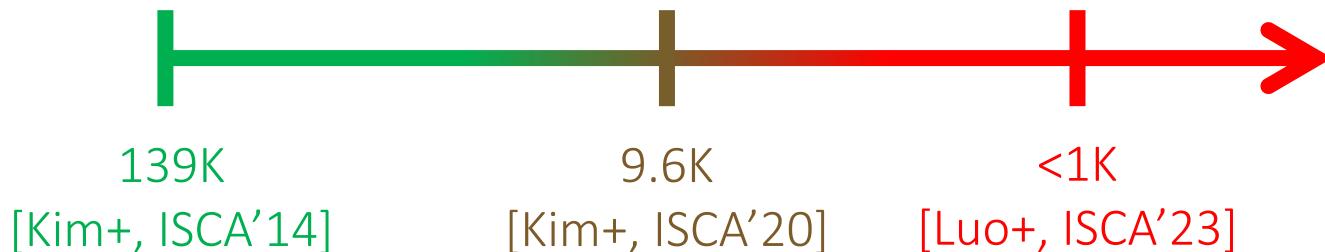
## DRAM Subarray



The minimum number of activations that causes a bitflip  
is called **the RowHammer threshold**

# Read Disturbance Vulnerabilities (II)

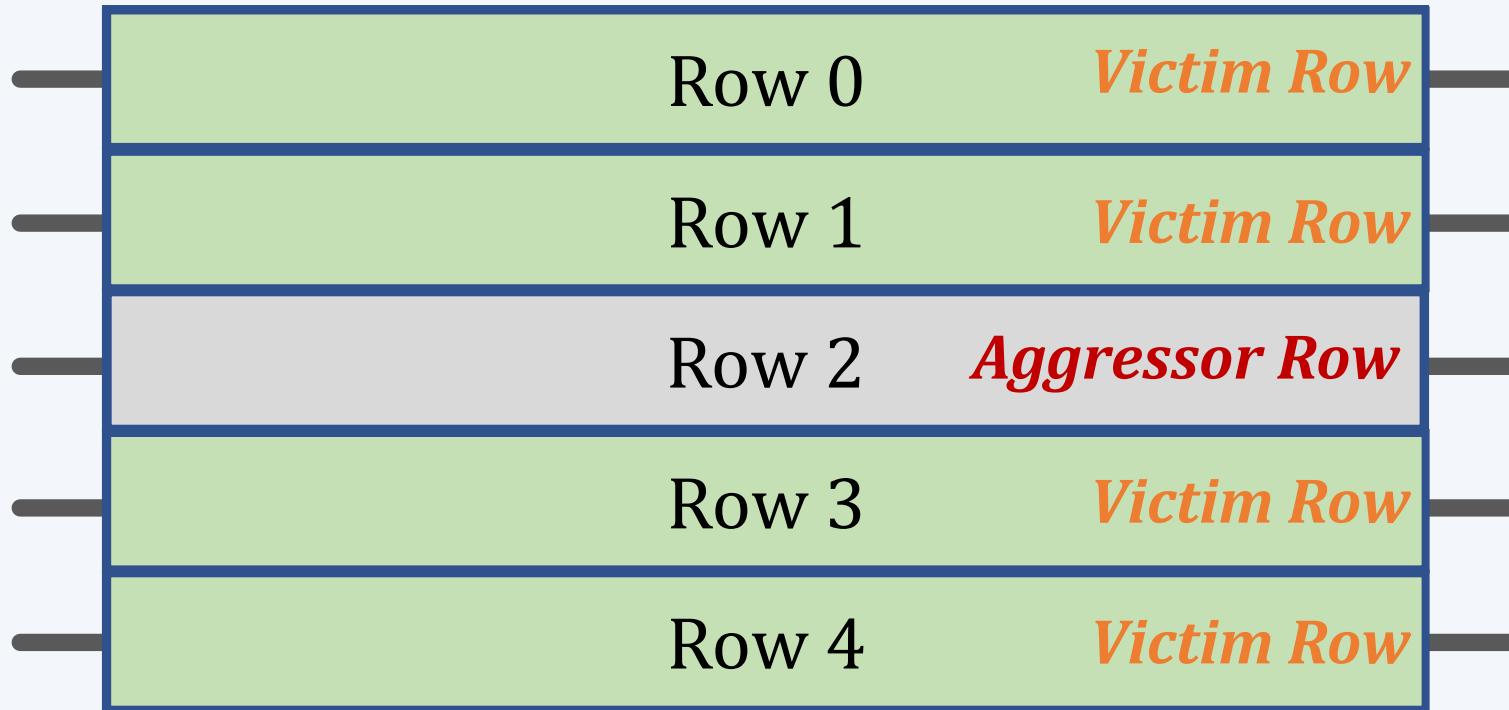
- DRAM chips are more vulnerable to read disturbance today
- Read disturbance bitflips occur at much lower activation counts  
(more than two orders of magnitude decrease in less than a decade):



Mitigation techniques against read disturbance attacks need to be **effective** and **efficient** for highly vulnerable systems

# Existing RowHammer Mitigations (I): Preventive Refresh

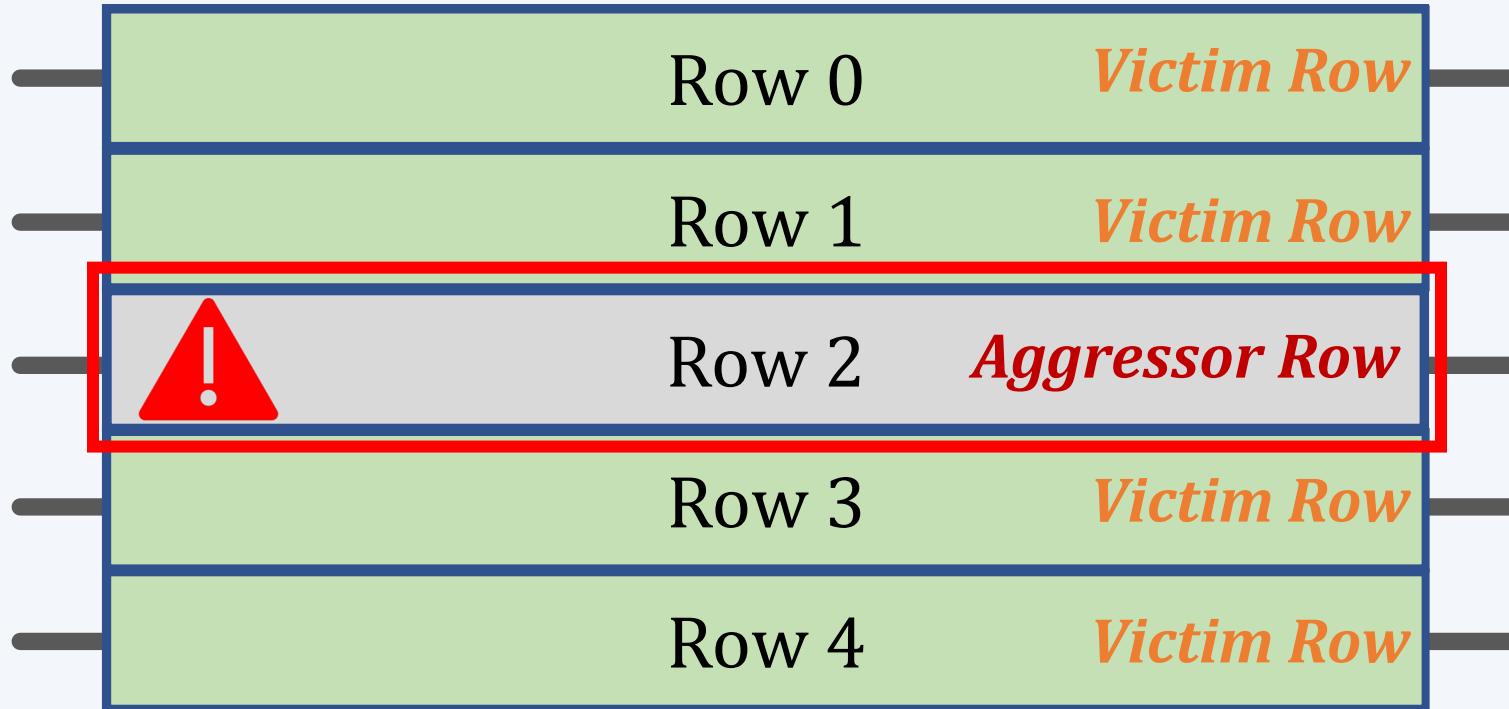
DRAM Subarray



Refreshing potential victim rows  
mitigates read disturbance bitflips

# Existing RowHammer Mitigations (II): DRAM Row Activation Tracking

## DRAM Subarray

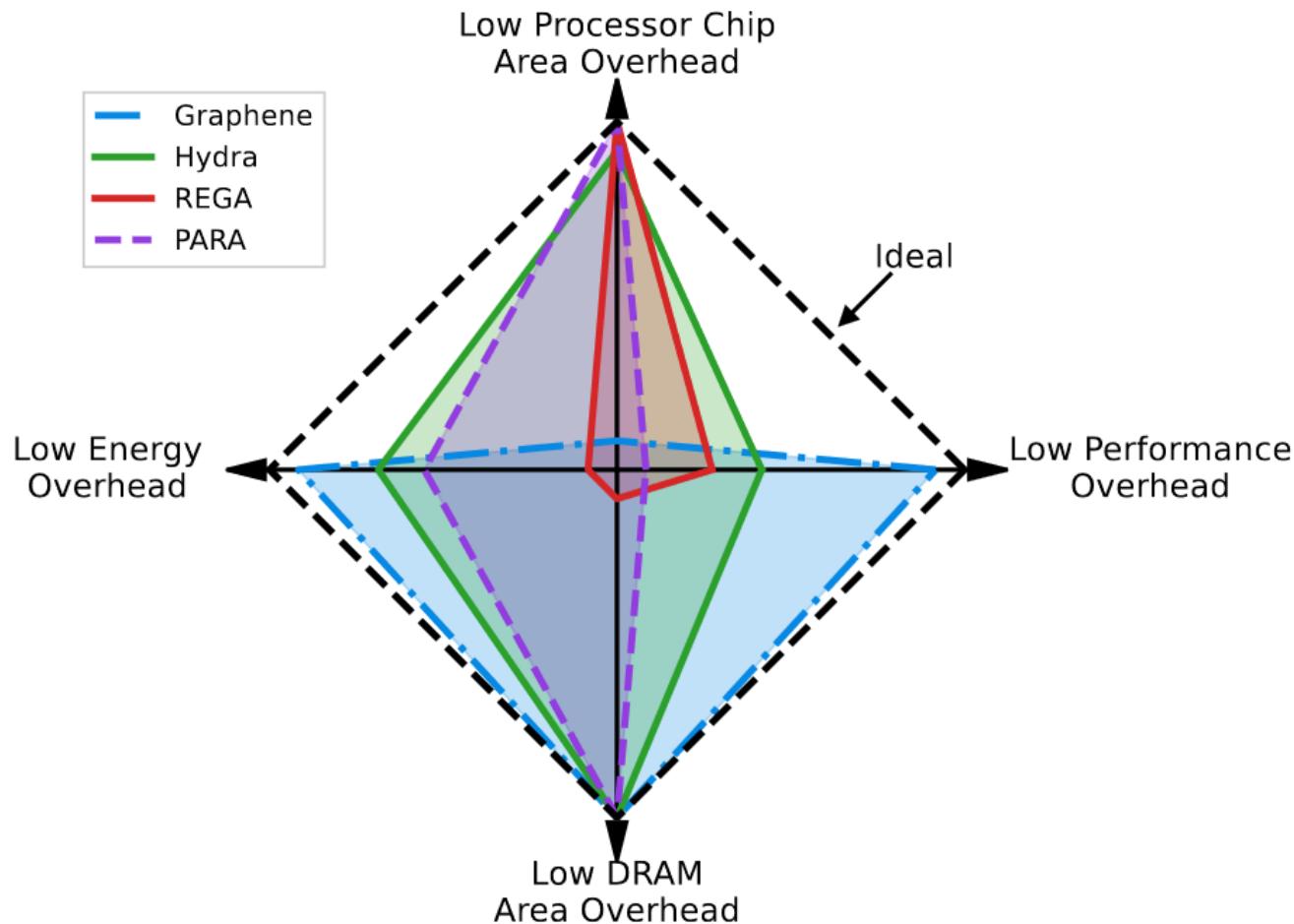


Mitigation techniques track DRAM row activation counts  
(of aggressor rows) to preventively refresh potential victim rows

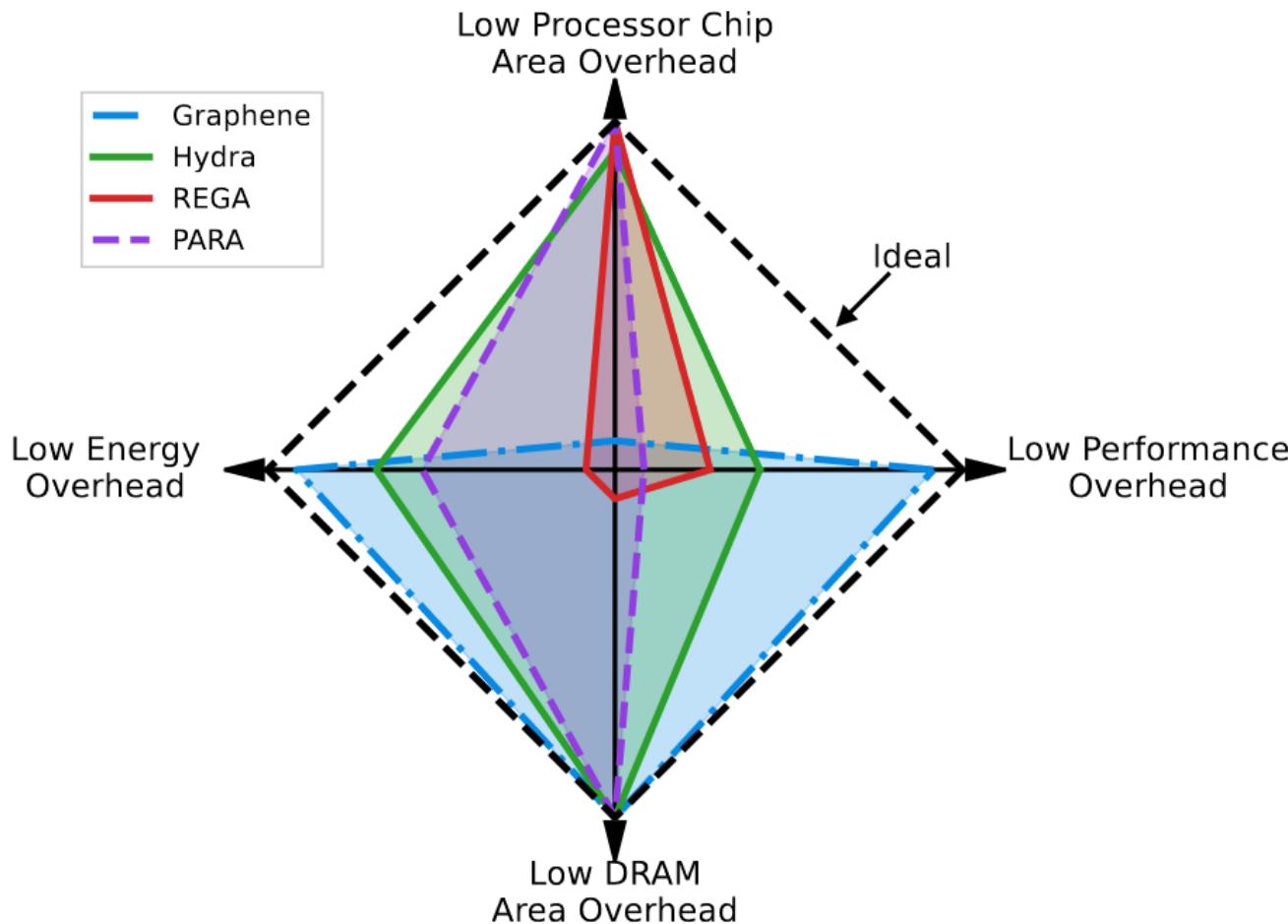
# Existing RowHammer Mitigations (III): Overview of Preventive Refresh-Based Mitigation Techniques

<p>DRAM</p> <p>One ACT counter per DRAM row</p>	<p>CAM-based implementation</p> <p>Processor Chip</p> <p>One ACT counter per aggressor row</p>	<p>Processor Chip</p> <p>&lt;1 counter per DRAM row</p>
Performance & Energy Costs  VERY LOW  Many DRAM rows (e.g., 128K per bank)	LOW  # of aggressor rows increases significantly	HIGH  High DRAM bandwidth consumption
Area Cost  VERY HIGH	HIGH	LOW

# Existing RowHammer Mitigations (IV): Overhead Trade-Off of State-of-the-Art Mitigation Techniques



# Existing RowHammer Mitigations (IV): Overhead Trade-Off of State-of-the-Art Mitigation Techniques



No existing mitigation technique prevents RowHammer bitflips  
at low area, performance and energy costs

# Outline

Background and Problem

Goal and Key Idea

CoMeT: Count-Min-Sketch-based Row Tracking

Evaluation

Conclusion

# Our Goal

---

Prevent RowHammer bitflips  
with low area, performance, and energy overheads  
in highly RowHammer-vulnerable DRAM-based systems  
(e.g., a RowHammer threshold of 125)

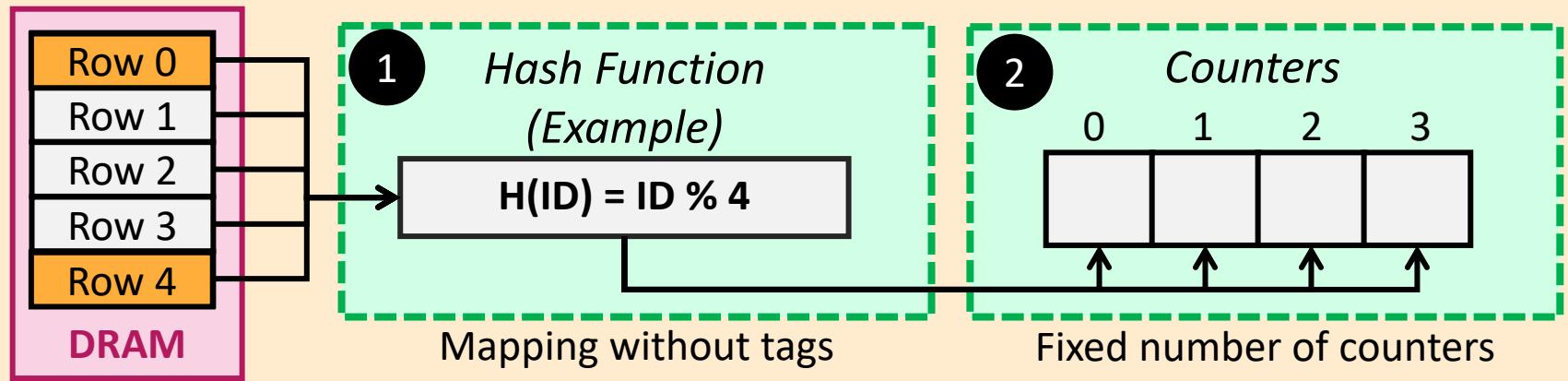
---

# Key Observation

Hash-based counters are low-cost:

1. can be implemented with low-cost structures and
2. can aggregate many rows' activation counts together

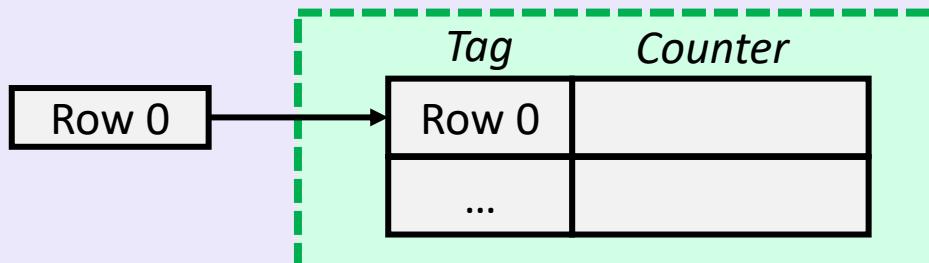
**LOW COST**



Tag-based counters are highly accurate:

Each one tracks one row's activation count

**HIGH  
ACCURACY**



# Key Idea

1

Use low-cost and scalable hash-based counters to track most DRAM rows' activations with low area overhead

2

Use highly accurate tag-based counters to track only a small set of DRAM rows to achieve low performance overhead

# Outline

Background and Problem

Goal and Key Idea

CoMeT: Count-Min-Sketch-based Row Tracking

Evaluation

Conclusion

# CoMeT Overview

## Counter Table (CT):

- Maps each DRAM row to a group of low-cost hash-based counters as uniquely as possible by employing the Count-Min Sketch technique
- Triggers a preventive refresh to an aggressor row's victim rows when the aggressor's counter group reaches an activation threshold

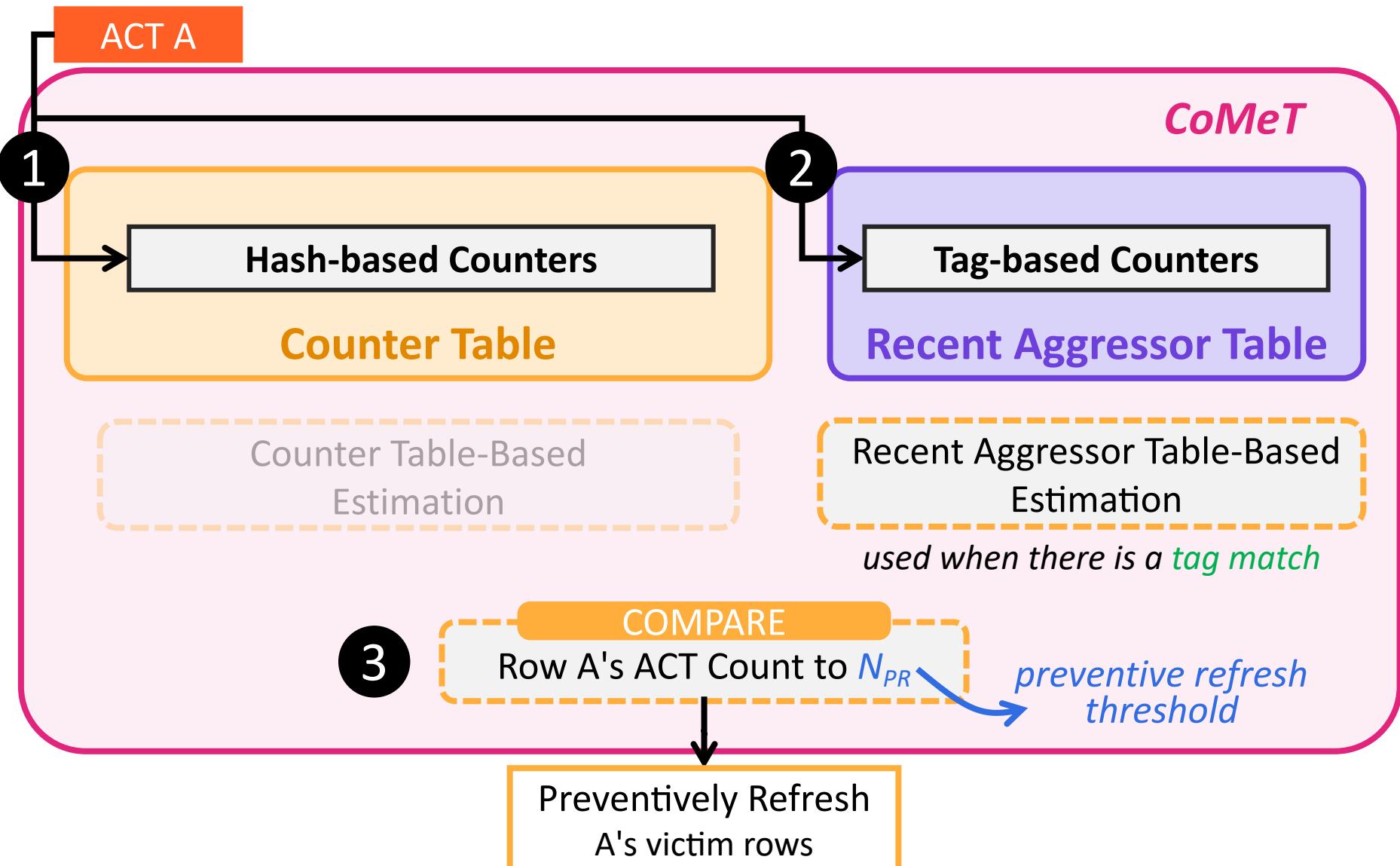
Tracks DRAM row activations at low area cost

## Recent Aggressor Table (RAT):

- Allocates highly accurate per-DRAM-row counters for *only* a small set of DRAM rows that are activated many times

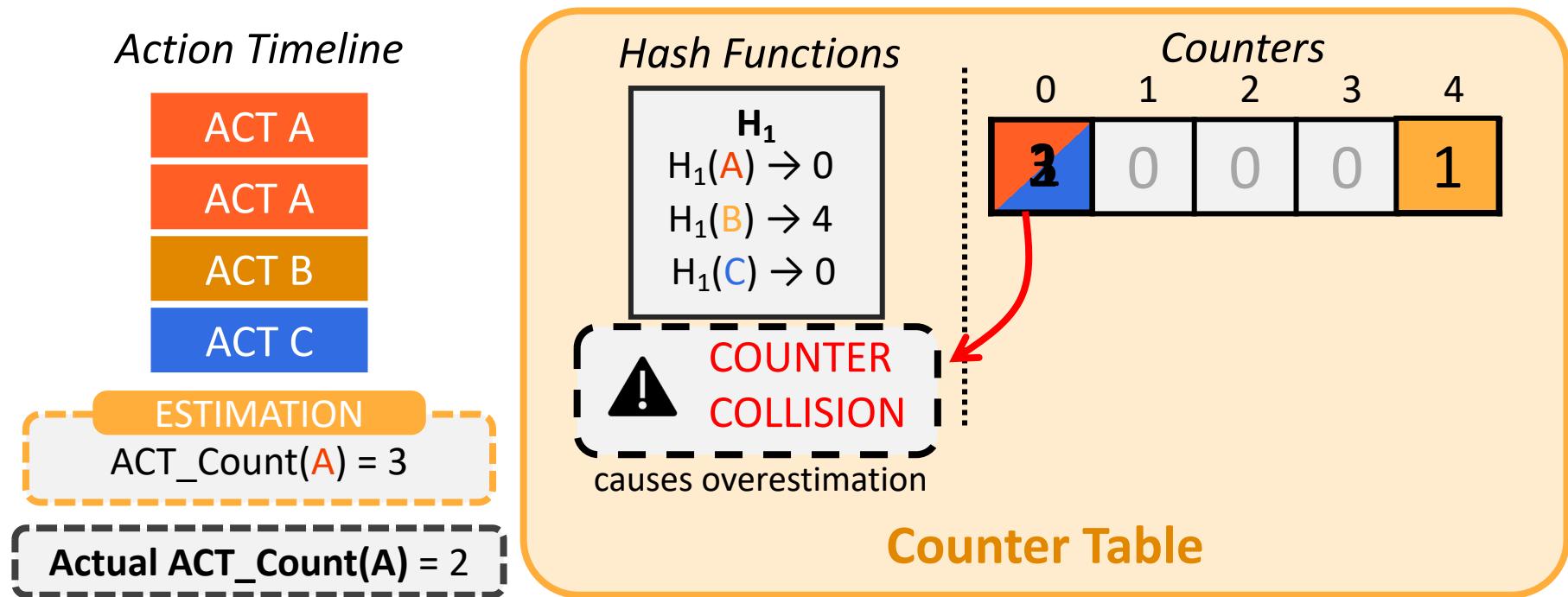
Reduces performance penalties by increasing tracking accuracy

# Operation of CoMeT



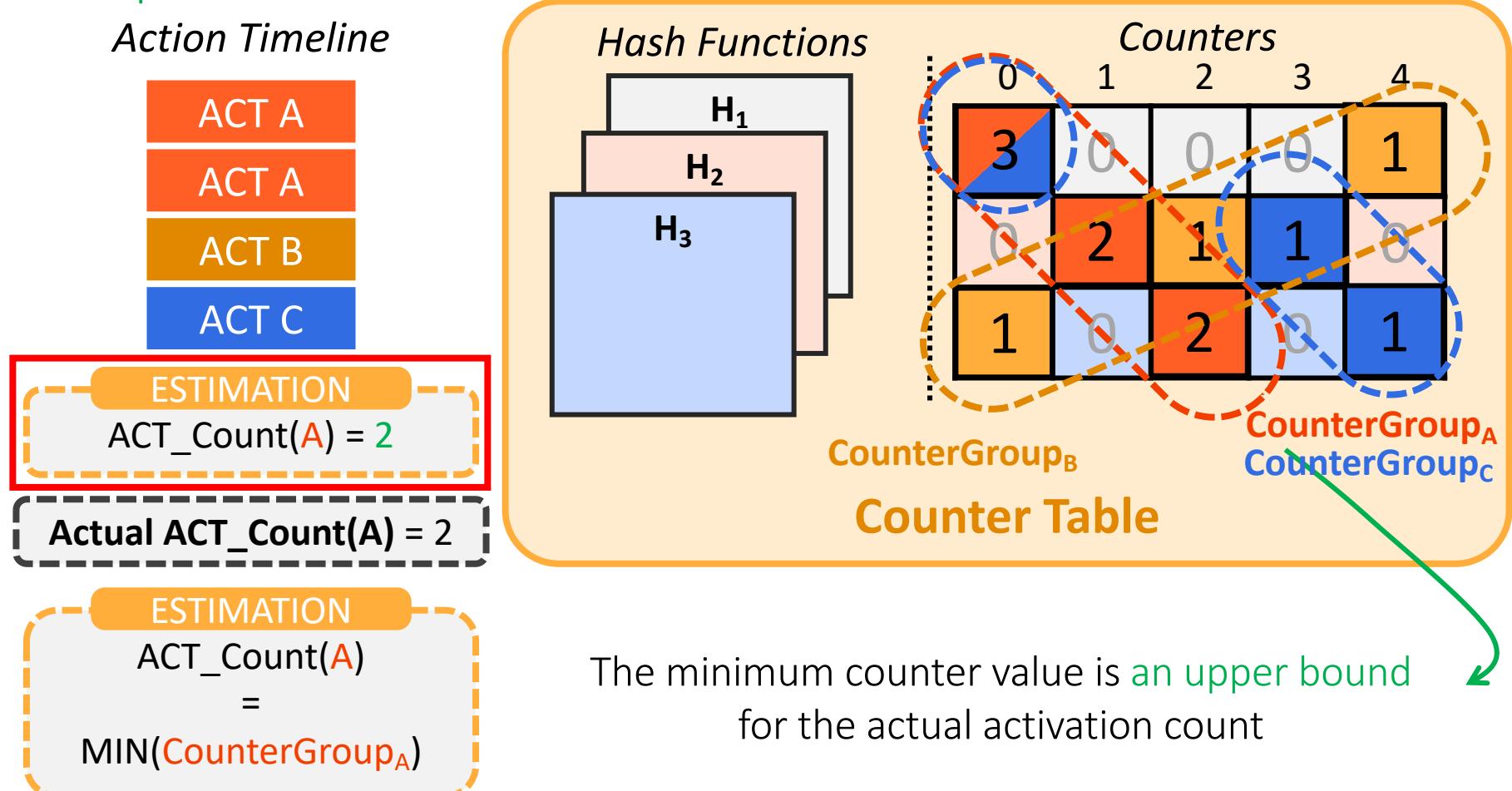
# Counter Table (CT): Count-Min-Sketch-based Row Tracking

- Count-Min Sketch: A hash-based frequent item counting technique



# Counter Table (CT): Count-Min-Sketch-based Row Tracking

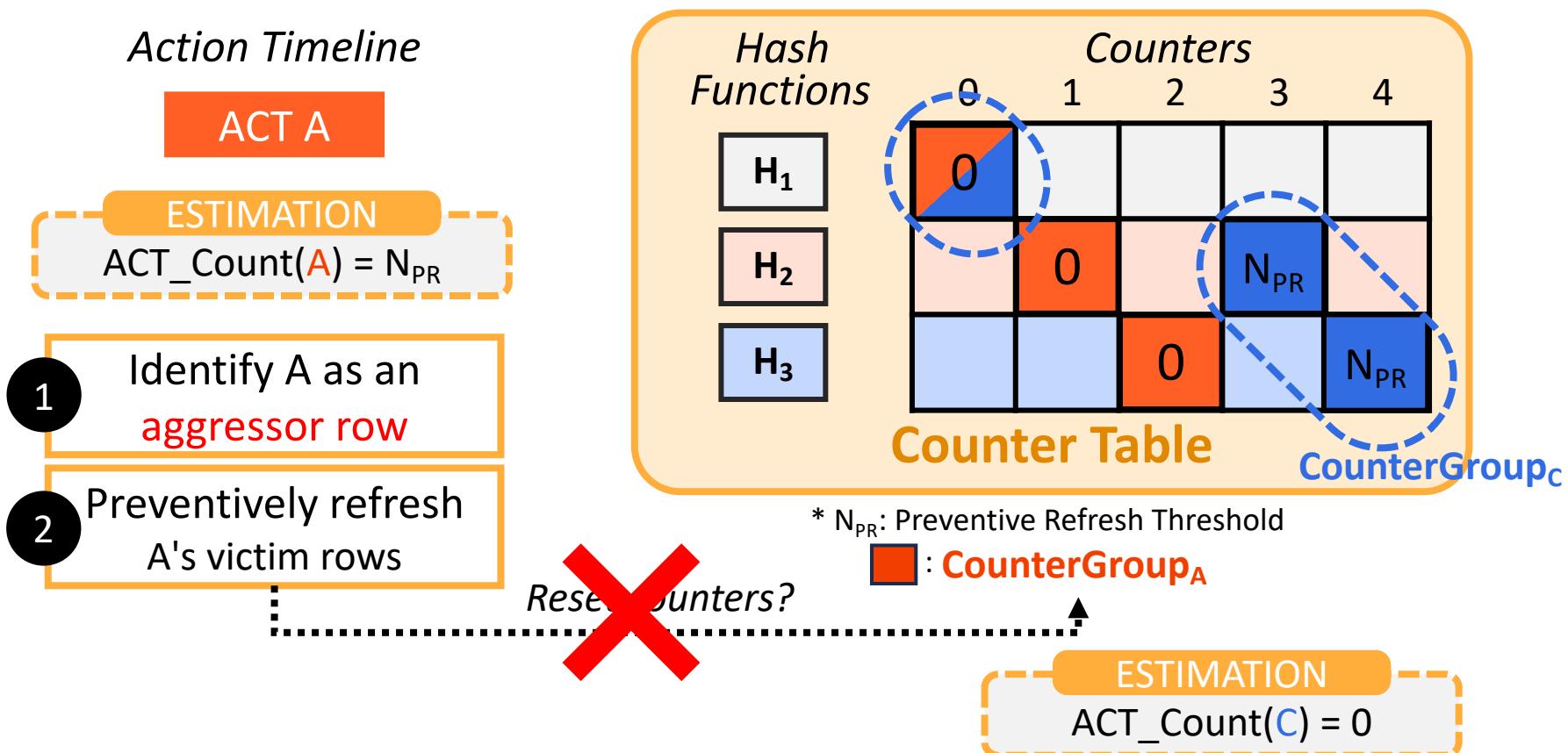
- To avoid overestimations, Counter Table implements multiple hash functions



The minimum counter value is an upper bound  
for the actual activation count

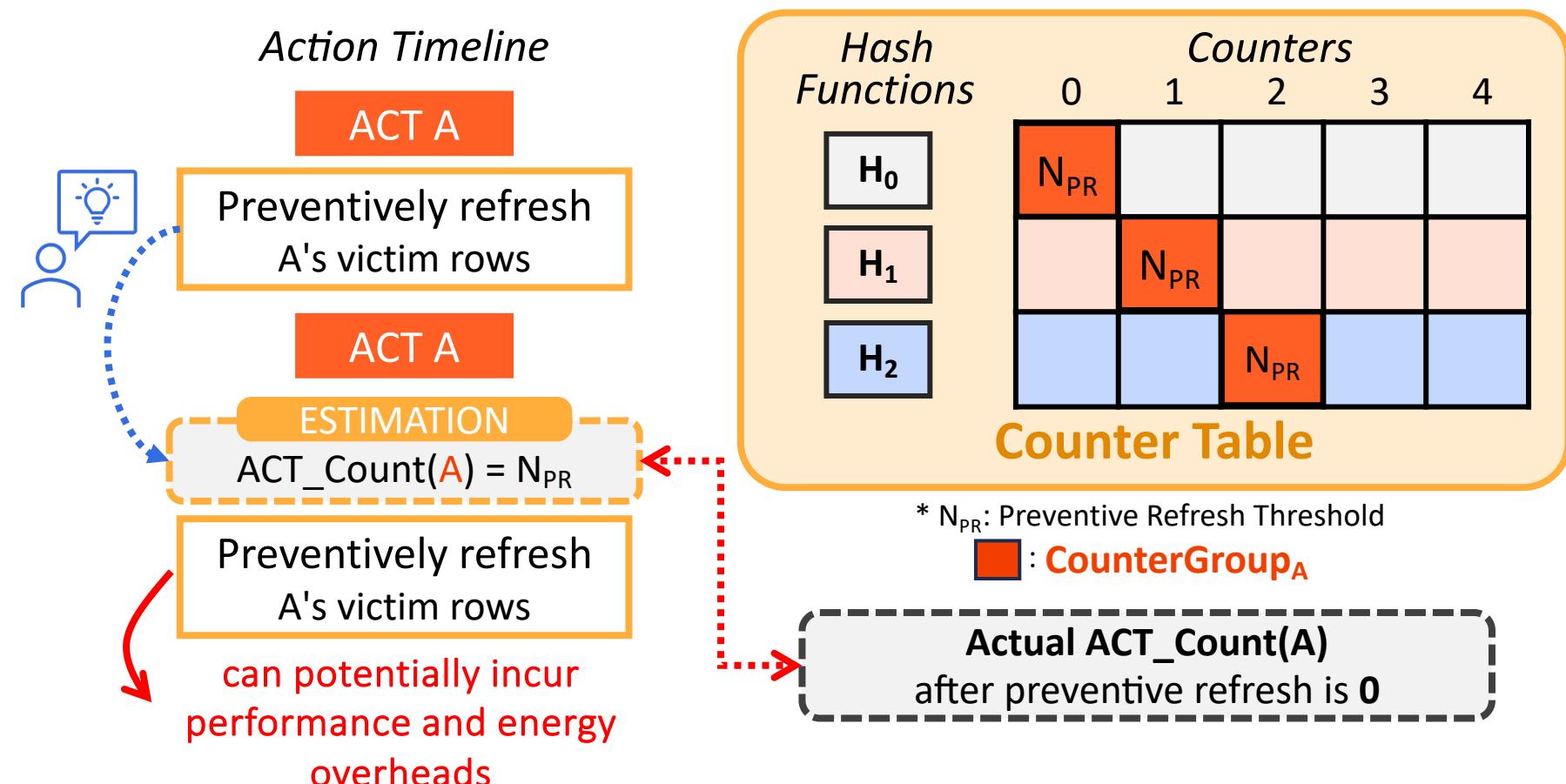
# Counter Table (CT): Identifying Aggressor Rows

- CoMeT sets a *preventive refresh threshold ( $N_{PR}$ )* to timely refresh an aggressor row's victim rows to prevent bitflips



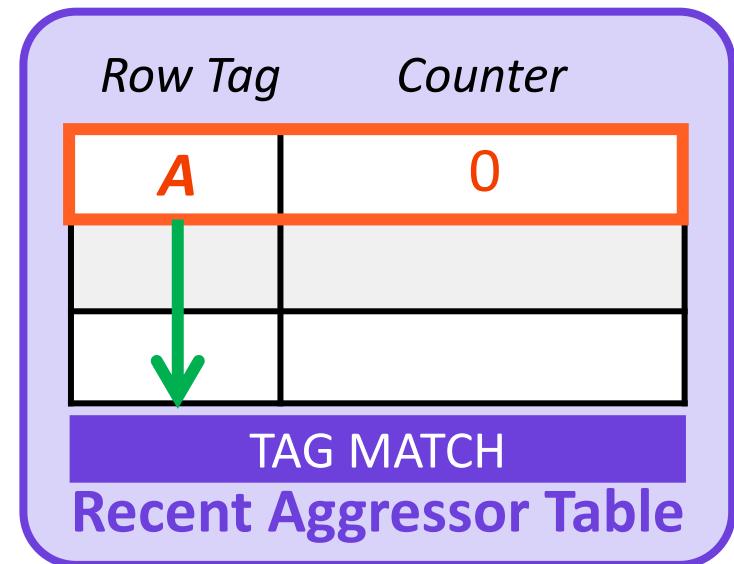
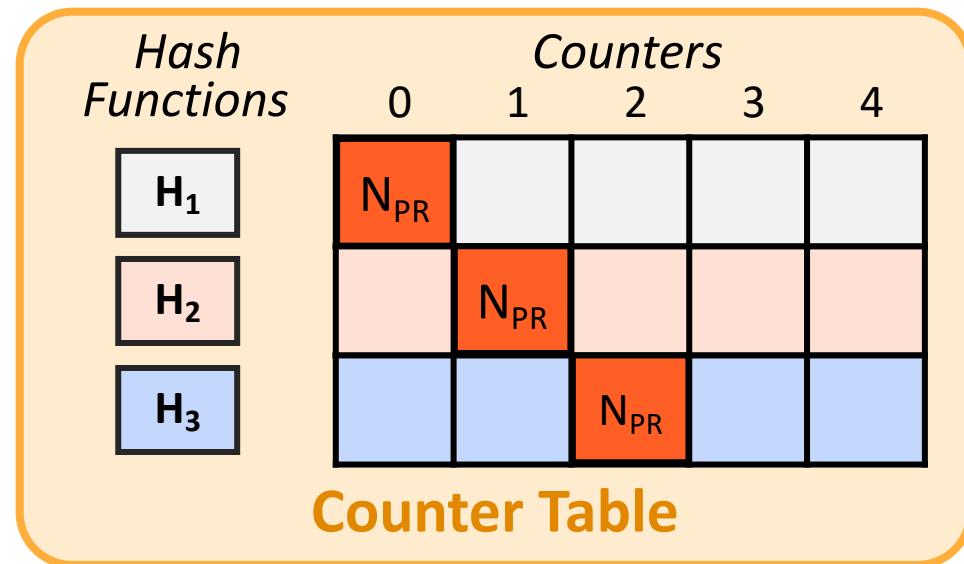
# Counter Table (CT): Counter Saturation

- CoMeT does not reset any counter in CT after preventive refresh
  - CT counters **saturate at  $N_{PR}$**



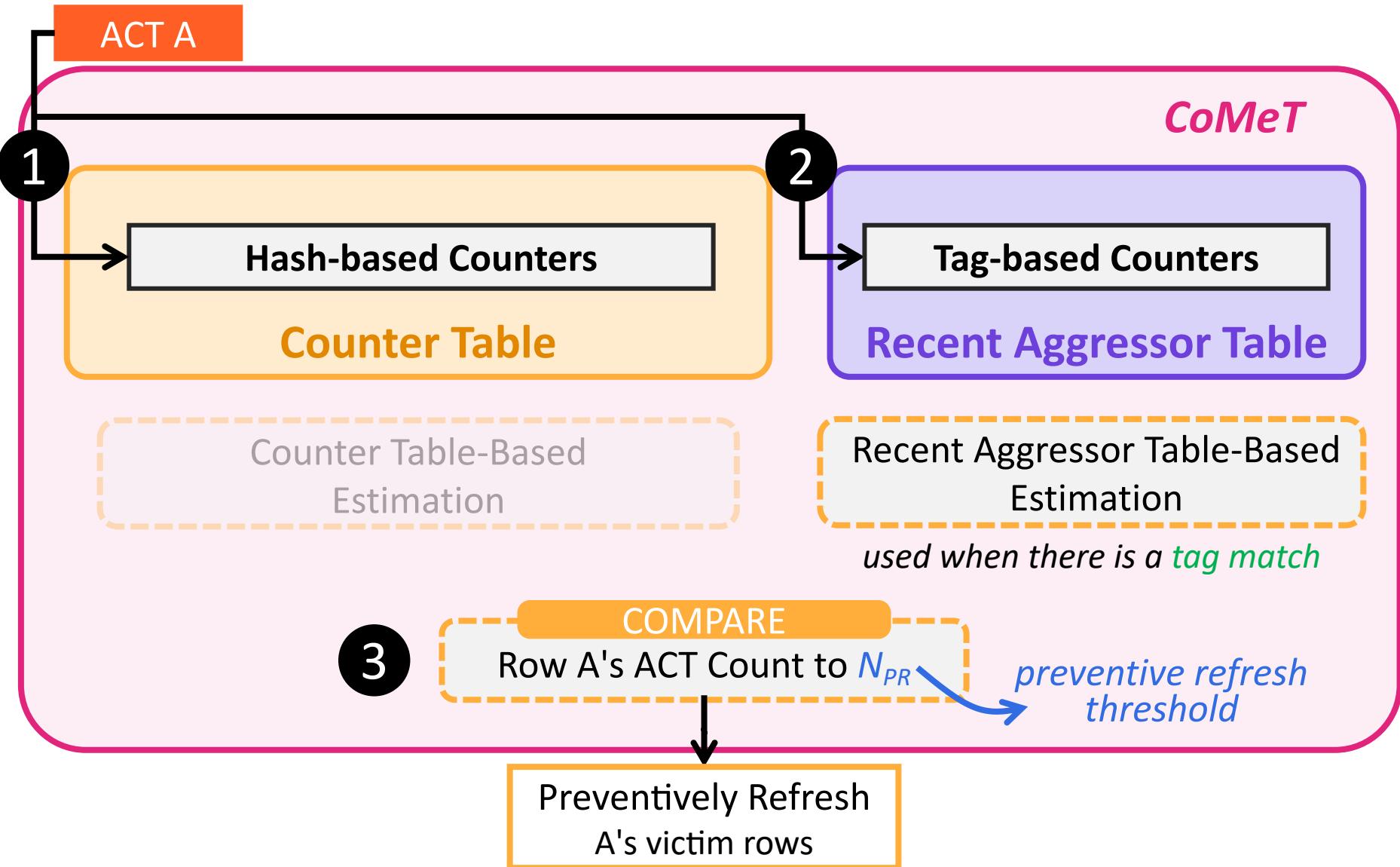
# Recent Aggressor Table

- Allocates per-DRAM-row counters for **aggressor rows** to accurately estimate their activation counts **after preventive refreshes**
  - Implemented for *only* **a small set** of DRAM rows to maintain a low area cost



If a DRAM row has a **Recent Aggressor Table entry**, CoMeT estimates its activation count **100% accurately**

# Operation of CoMeT



# More Operational Details for CoMeT

- Counter update policy
- Periodic counter reset mechanism
- Recent Aggressor Table eviction policy
- Early preventive refresh at coarse granularity
- Determining the preventive refresh threshold

# More Operational Details for CoMeT



## CoMeT: Count-Min-Sketch-based Row Tracking to Mitigate RowHammer at Low Cost

F. Nisa Bostancı      İsmail Emir Yüksel      Ataberk Olgun      Konstantinos Kanellopoulos  
Yahya Can Tuğrul      A. Giray Yağlıkçı      Mohammad Sadrosadati      Onur Mutlu

ETH Zürich

*DRAM chips are increasingly more vulnerable to read-disturbance phenomena (e.g., RowHammer and RowPress), where repeatedly accessing DRAM rows causes bitflips in nearby rows due to DRAM density scaling. Under low RowHammer thresholds, existing RowHammer mitigations either incur high area overheads or degrade performance significantly.*

*We propose a new RowHammer mitigation mechanism, CoMeT, that prevents RowHammer bitflips with low area, performance, and energy costs in DRAM-based systems at very*

### 1. Introduction

DRAM chips are susceptible to read-disturbance where repeatedly accessing a DRAM row (i.e., *an aggressor row*) can cause bitflips in physically nearby rows (i.e., *victim rows*) [1–13]. RowHammer is a type of read-disturbance phenomenon that is caused by repeatedly opening and closing (i.e., *hammering*) DRAM rows. Modern DRAM chips become more vulnerable to RowHammer as DRAM technology node size becomes smaller [1, 2, 4, 14–19]: the minimum number of row activations needed to cause a bitflip (i.e., *RowHammer threshold*)

<https://arxiv.org/abs/2402.18769>

<https://github.com/CMU-SAFARI/CoMeT>

# Outline

Background and Problem

Goal and Key Idea

CoMeT: Count-Min-Sketch-based Row Tracking

Evaluation

Conclusion

# Evaluation Methodology

- **Performance and energy consumption evaluation:** cycle-level simulations using [Ramulator](#) [Kim+, CAL 2015] and [DRAMPower](#) [Chandrasekar+, DATE 2013]

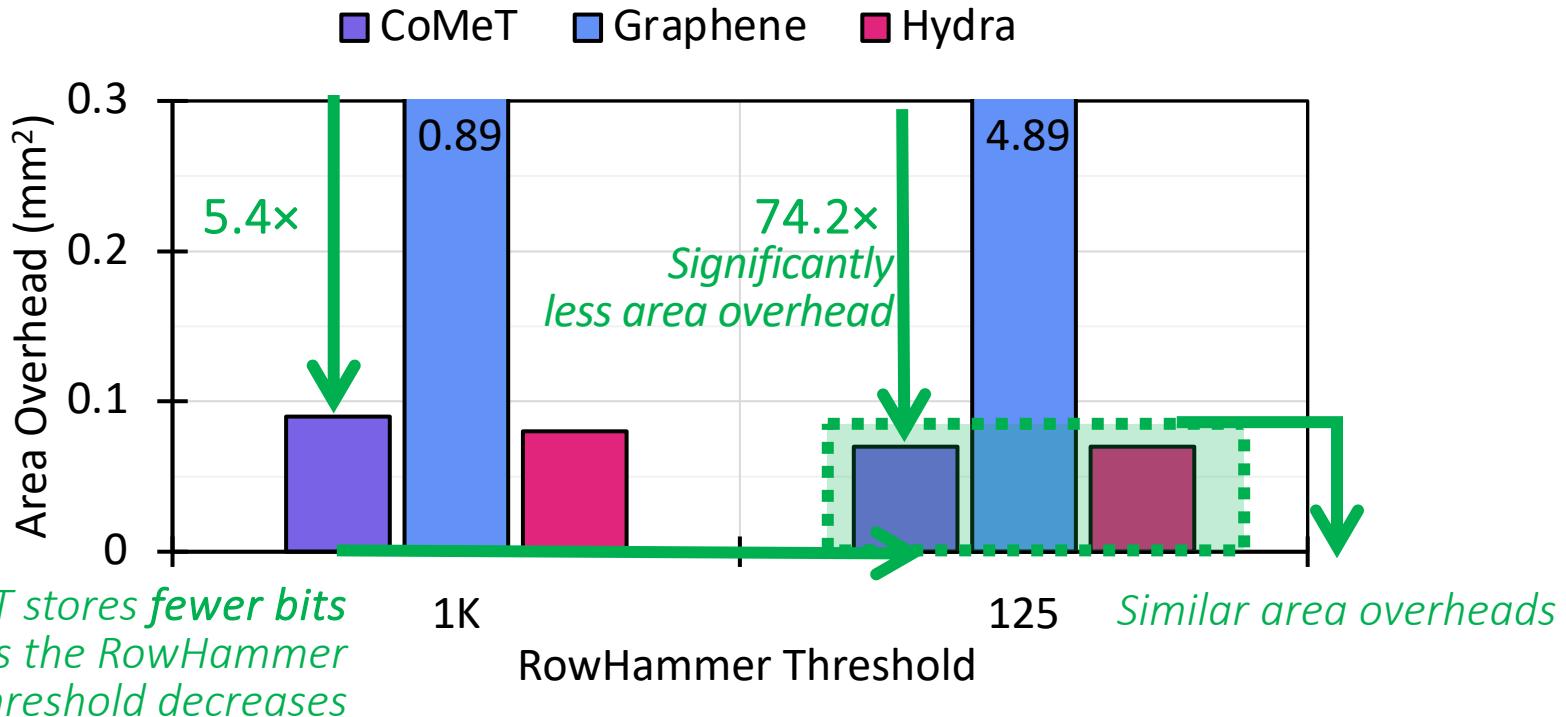
- **System Configuration:**

Processor	1 or 8 cores, 3.6GHz clock frequency, 4-wide issue, 128-entry instruction window
DRAM	DDR4, 1 channel, 2 rank/channel, 4 bank groups, 4 banks/bank group, 128K rows/bank
Memory Ctrl.	64-entry read and write requests queues, Scheduling policy: FR-FCFS with a column cap of 16 Last-Level Cache 8 MiB (single-core), 16 MiB (8-core)
CoMeT	<i>Counter Table:</i> 4 hash functions 512 counters per hash <i>Recent Aggressor Table:</i> 128 entries

- **Comparison Points:** 4 state-of-the-art RowHammer mitigations
  - Graphene (best performing), Hydra (area-optimized best performing), Low Processor Chip Area Cost: REGA, PARA
- **Workloads:** 61 single-core applications and 56 8-core workload mixes
  - SPEC CPU2006, SPEC CPU2017, TPC, MediaBench, YCSB

# Hardware Implementation

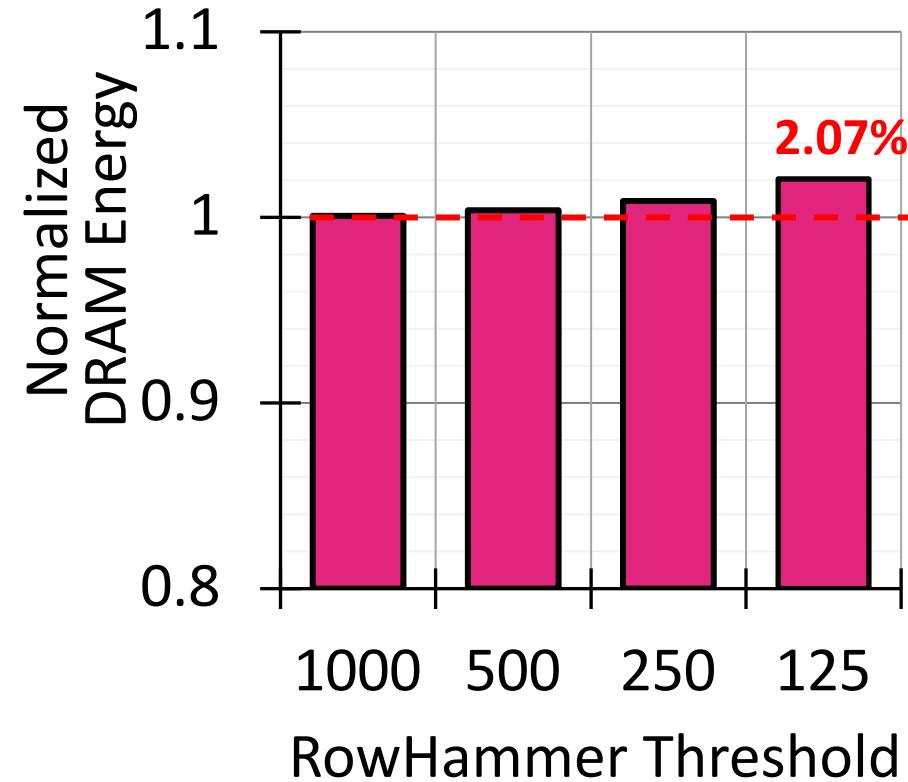
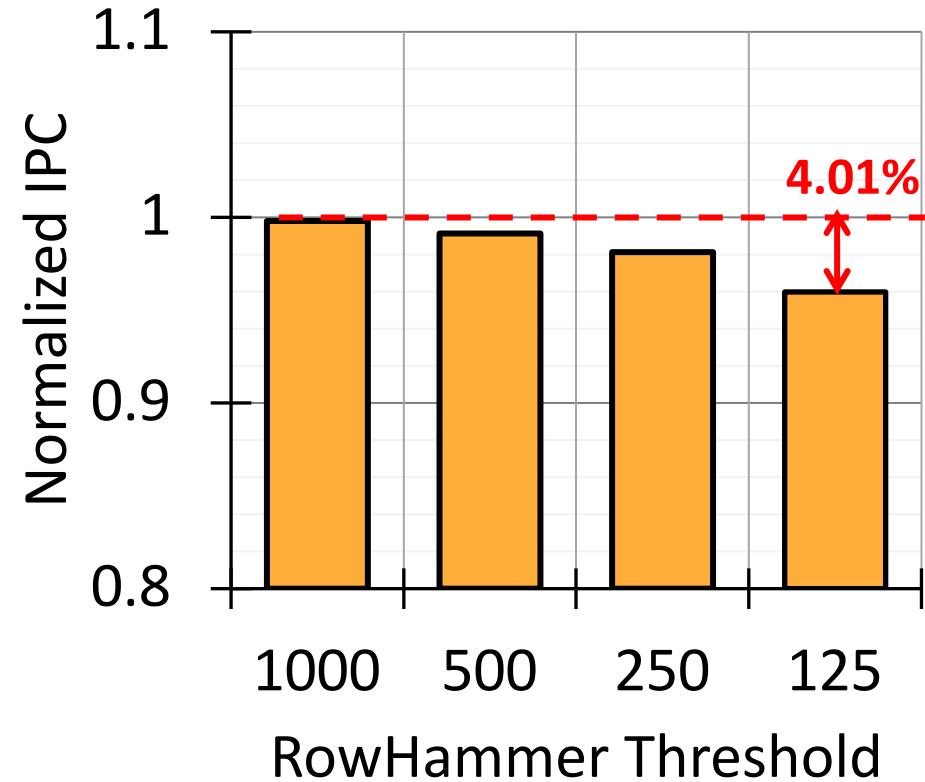
- Storage and area overhead analysis: CACTI
- Dual-rank area overhead comparison:



CoMeT incurs **a significantly less area overhead** than Graphene  
and **a similar area overhead** to Hydra

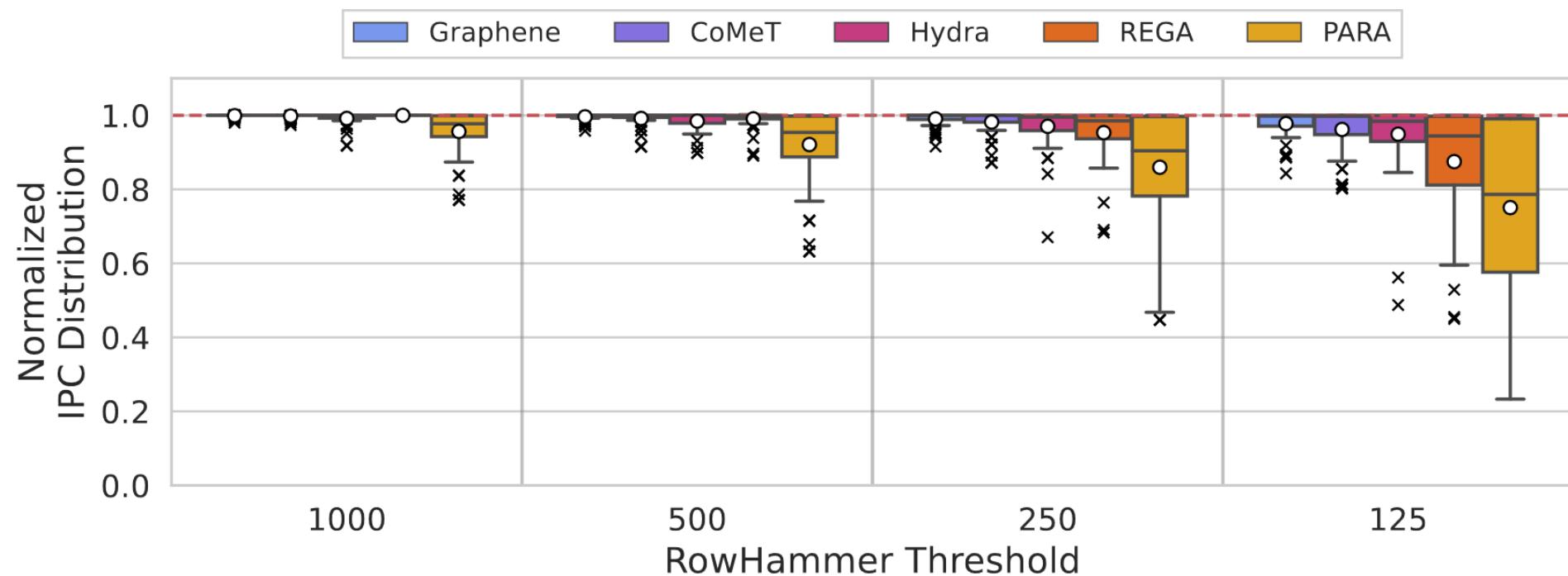
# Performance and DRAM Energy

- Average performance and DRAM energy overheads of CoMeT across single-core applications



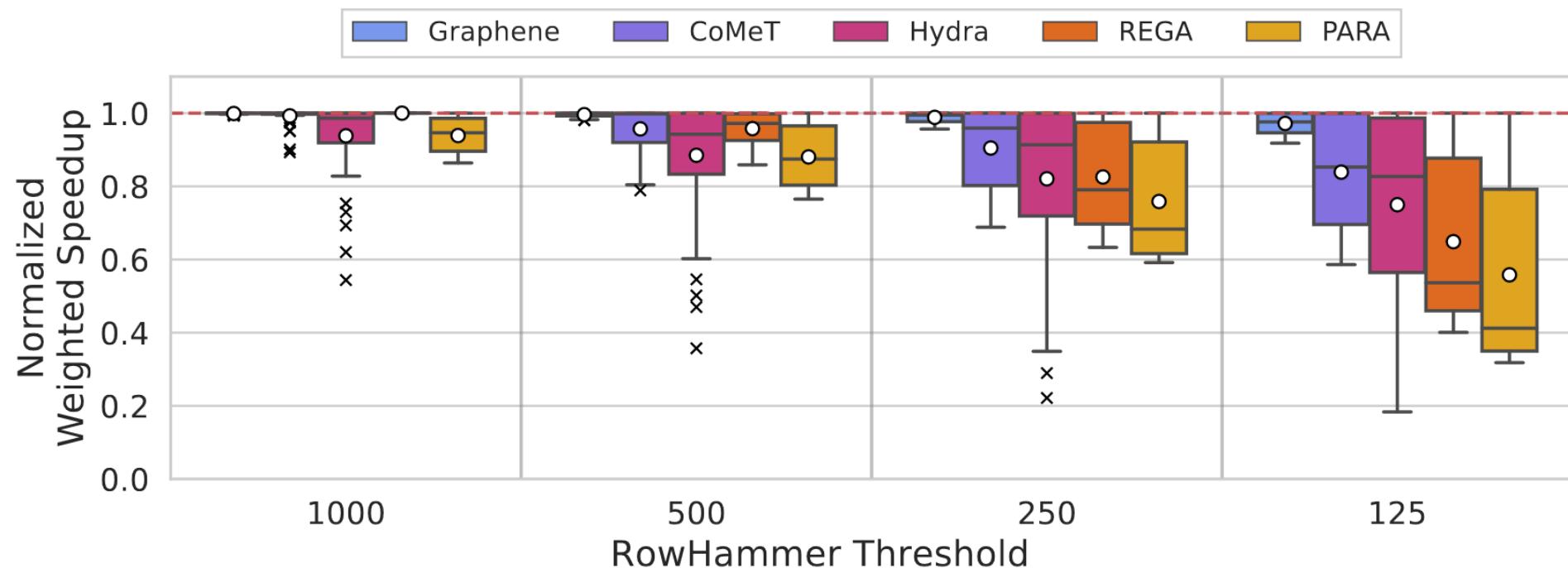
CoMeT prevents bitflips with **very small average performance and DRAM energy overheads** compared to a baseline system with *no* RowHammer mitigation

# Performance Comparison: Single-Core Applications



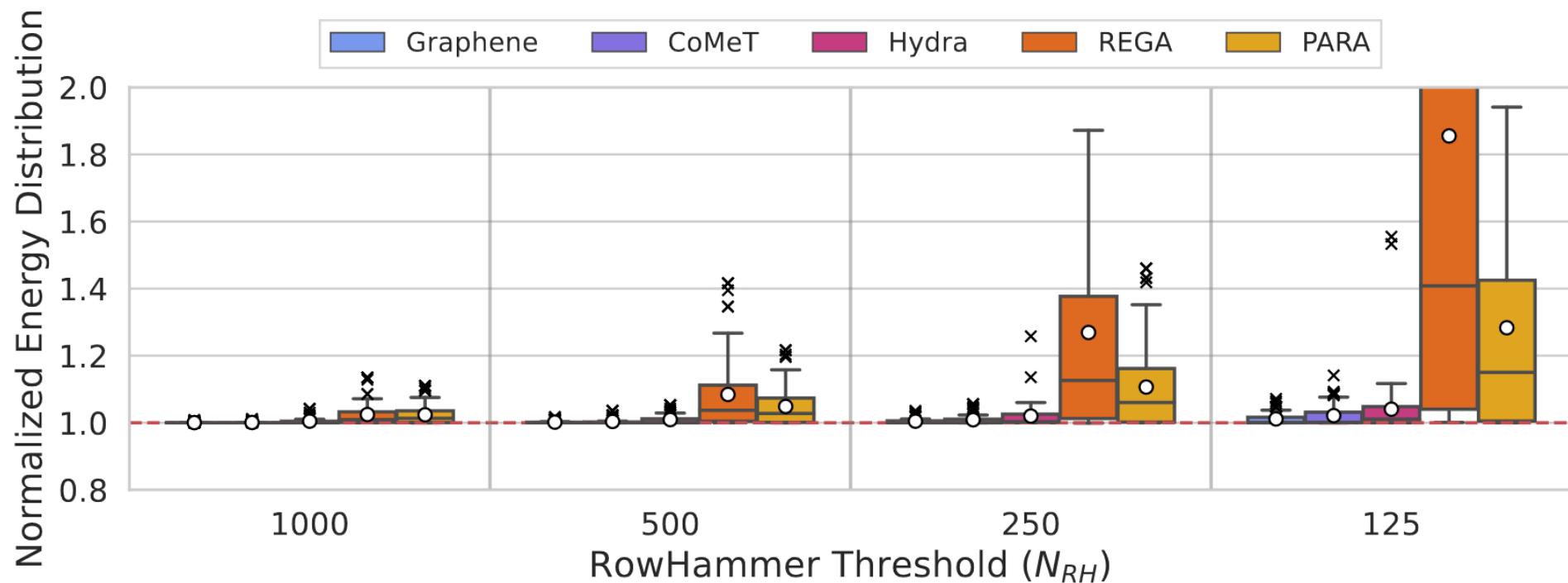
CoMeT incurs a small performance overhead ( $\leq 1.75\%$ ) over Graphene and outperforms Hydra (by up to 39.1%) at all RowHammer thresholds

# Performance Comparison: 8-Core Workloads



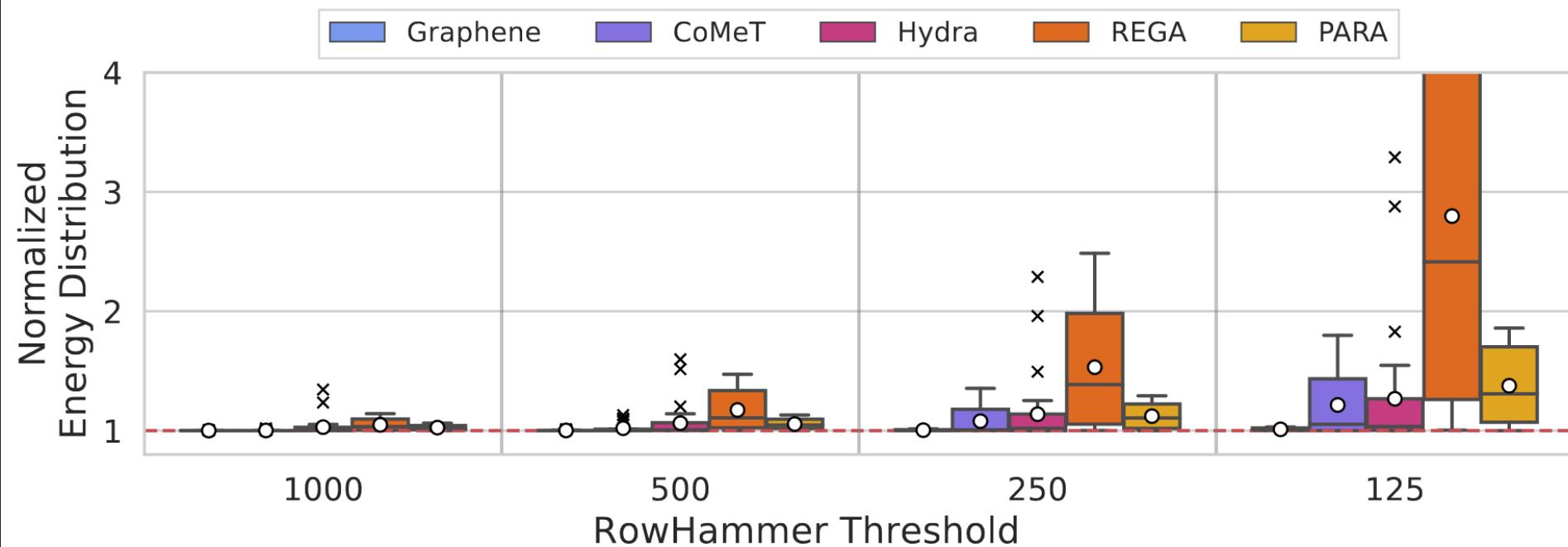
Trends are similar to single-core application evaluation trends

# DRAM Energy Comparison: Single-Core Applications



CoMeT incurs a small DRAM energy overhead (<1%) over Graphene and consumes less DRAM energy than Hydra

# DRAM Energy Comparison: 8-Core Workloads



Multicore energy trends are similar to single core energy trends

# More in the Paper

- Security Analysis of CoMeT
- Sensitivity Analysis
  - Counter Table Configurations
  - Recent Aggressor Table Configurations
  - Counter Reset Period and Preventive Refresh Threshold Values
- CoMeT's Performance under Adversarial Workloads
- Comparison against Throttling-Based Mitigation Techniques
- CoMeT's Performance at High RowHammer Thresholds

# More in the Paper

- Security Analysis of CoMeT



## CoMeT: Count-Min-Sketch-based Row Tracking to Mitigate RowHammer at Low Cost

F. Nisa Bostancı      İsmail Emir Yüksel      Ataberk Olgun      Konstantinos Kanellopoulos  
Yahya Can Tuğrul      A. Giray Yağlıkçı      Mohammad Sadrosadati      Onur Mutlu

ETH Zürich

*DRAM chips are increasingly more vulnerable to read-disturbance phenomena (e.g., RowHammer and RowPress), where repeatedly accessing DRAM rows causes bitflips in nearby rows due to DRAM density scaling. Under low RowHammer thresholds, existing RowHammer mitigations either incur high area overheads or degrade performance significantly.*

*We propose a new RowHammer mitigation mechanism, CoMeT, that prevents RowHammer bitflips with low area, performance, and energy costs in DRAM-based systems at very*

### 1. Introduction

DRAM chips are susceptible to read-disturbance where repeatedly accessing a DRAM row (i.e., *an aggressor row*) can cause bitflips in physically nearby rows (i.e., *victim rows*) [1–13]. RowHammer is a type of read-disturbance phenomenon that is caused by repeatedly opening and closing (i.e., *hammering*) DRAM rows. Modern DRAM chips become more vulnerable to RowHammer as DRAM technology node size becomes smaller [1, 2, 4, 14–19]: the minimum number of row activations needed to cause a bitflip (i.e., *RowHammer threshold*)

- Performance Evaluation: <https://arxiv.org/abs/2402.18769> Thresholds

# CoMeT is Open Source and Artifact Evaluated



CMU-SAFARI / CoMeT Public

Code Issues Pull requests Actions Projects Security Insights

master 1 Branch 0 Tags Go to file Code About

olgunataberk	Update README.md	ff039ed · 3 months ago	28 Commits
configs/ArtifactEvaluation	initial commit	3 months ago	
ext	add ext files	3 months ago	
scripts/artifact	clean stale results	3 months ago	
src	remove unnecessary files	3 months ago	
.gitignore	add scripts for fetching CPU traces and generating Slurm jobs	3 months ago	
CMakeLists.txt	initial commit	3 months ago	
Doxyfile	initial commit	3 months ago	
LICENSE	update README.md and LICENSE	3 months ago	
README.md	Update README.md	3 months ago	

About

CoMeT is a new low-cost RowHammer mitigation that uses Count-Min Sketch-based aggressor row tracking

Readme MIT license Activity Custom properties 6 stars 7 watching 0 forks Report repository

Releases No releases published

<https://github.com/CMU-SAFARI/CoMeT>

# Outline

Background and Problem

Goal and Key Idea

CoMeT: Count-Min-Sketch-based Row Tracking

Evaluation

Conclusion

# Conclusion

**Goal:** Prevent RowHammer bitflips **with low area, performance, and energy overheads** in highly RowHammer-vulnerable DRAM-based systems

**Key Idea:** Use **low-cost** and **scalable hash-based counters** to accurately track DRAM rows

**CoMeT:**

- tracks most DRAM rows with **scalable hash-based counters** by employing the **Count-Min-Sketch technique** to achieve a low area cost
- tracks only **a small set of DRAM rows that are activated many times** with **highly accurate per-DRAM-row activation counters** to reduce performance penalties

**Evaluation:** CoMeT achieves a good trade-off between area, performance and energy costs

- incurs **significantly less area overhead (74.2 $\times$ )** compared to the state-of-the-art technique
- **outperforms** the state-of-the-art technique (by up to **39.1%**)

<https://github.com/CMU-SAFARI/CoMeT>



# Count-Min-Sketch-based Row Tracking to Mitigate RowHammer at Low Cost

**F. Nisa Bostancı**

I. E. Yüksel A. Olgun K. Kanellopoulos Y. C. Tuğrul  
A. G. Yağlıkçı M. Sadrosadati O. Mutlu



*Paper Link*



*Github Repo*

**SAFARI**

**ETH zürich**



# Count-Min-Sketch-based Row Tracking to Mitigate RowHammer at Low Cost

## BACKUP SLIDES

**F. Nisa Bostancı**

I. E. Yüksel A. Olgun K. Kanellopoulos Y. C. Tuğrul  
A. G. Yağlıkçı M. Sadrosadati O. Mutlu



Paper Link

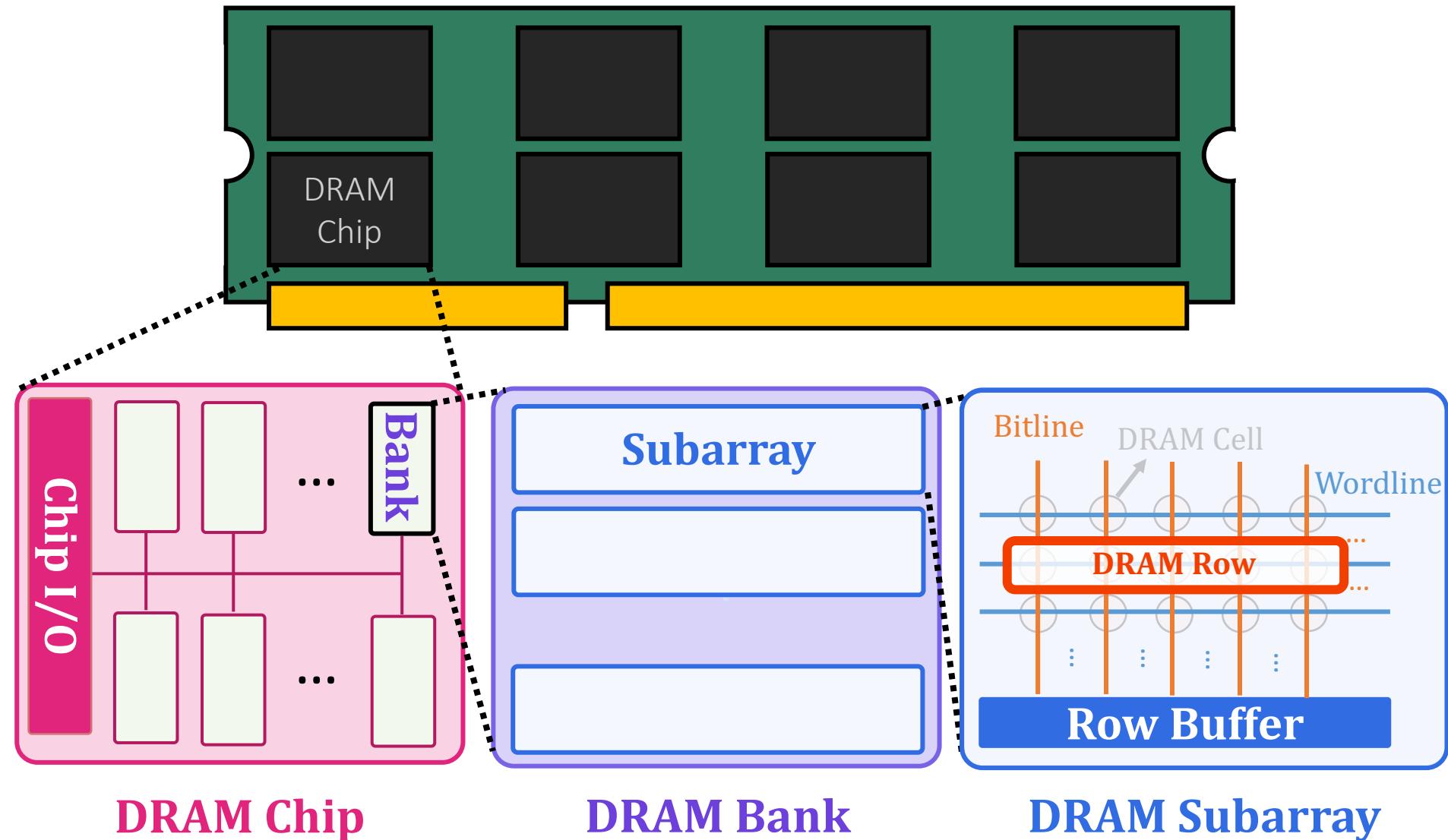


Github Repo

**SAFARI**

**ETH zürich**

# DRAM Organization



DRAM Chip

DRAM Bank

DRAM Subarray

# Read Disturbance Vulnerabilities

## DRAM Subarray

[Kim+ ISCA'20]

The minimum number of row activations needed to cause a bitflip (i.e., *RowHammer threshold* ( $N_{RH}$ )) has **reduced by more than an order of magnitude** in less than a decade

*closed*

Row 2

[Luo+ ISCA'23]

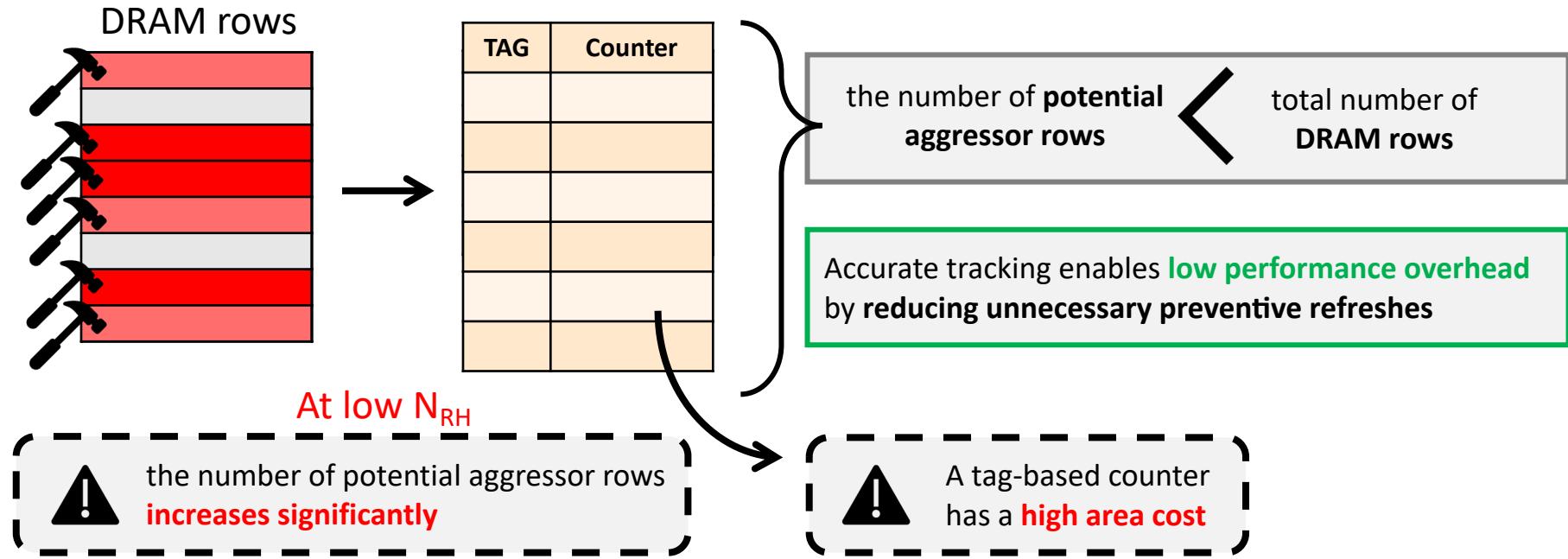
RowPress is shown to lead to bitflips with **one to two orders of magnitude fewer activations** (than RowHammer) under realistic conditions

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

[Kim+ ISCA'20]

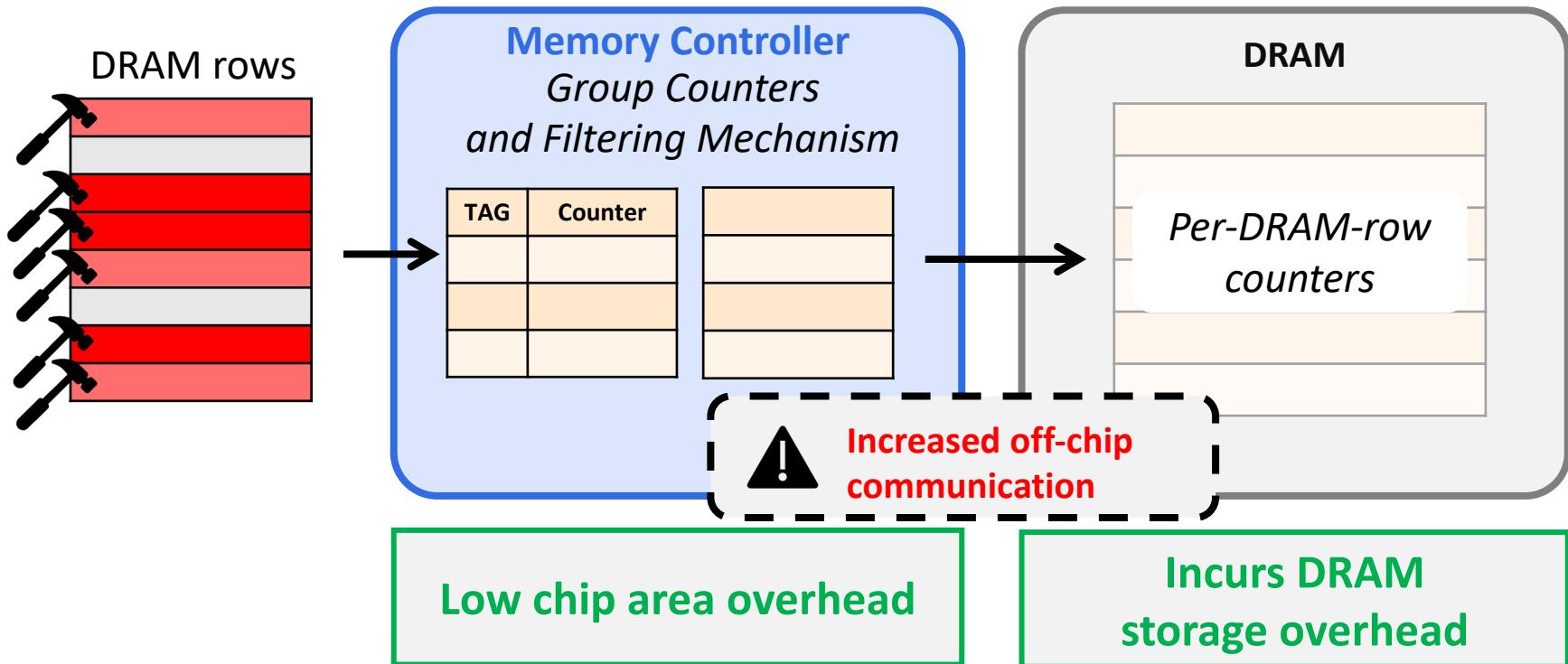
# Existing RowHammer Mitigations (II): Performance Optimized Mitigations

- Accurately tracking DRAM row activations can be done by allocating **per-row counters** to **potential aggressor rows**



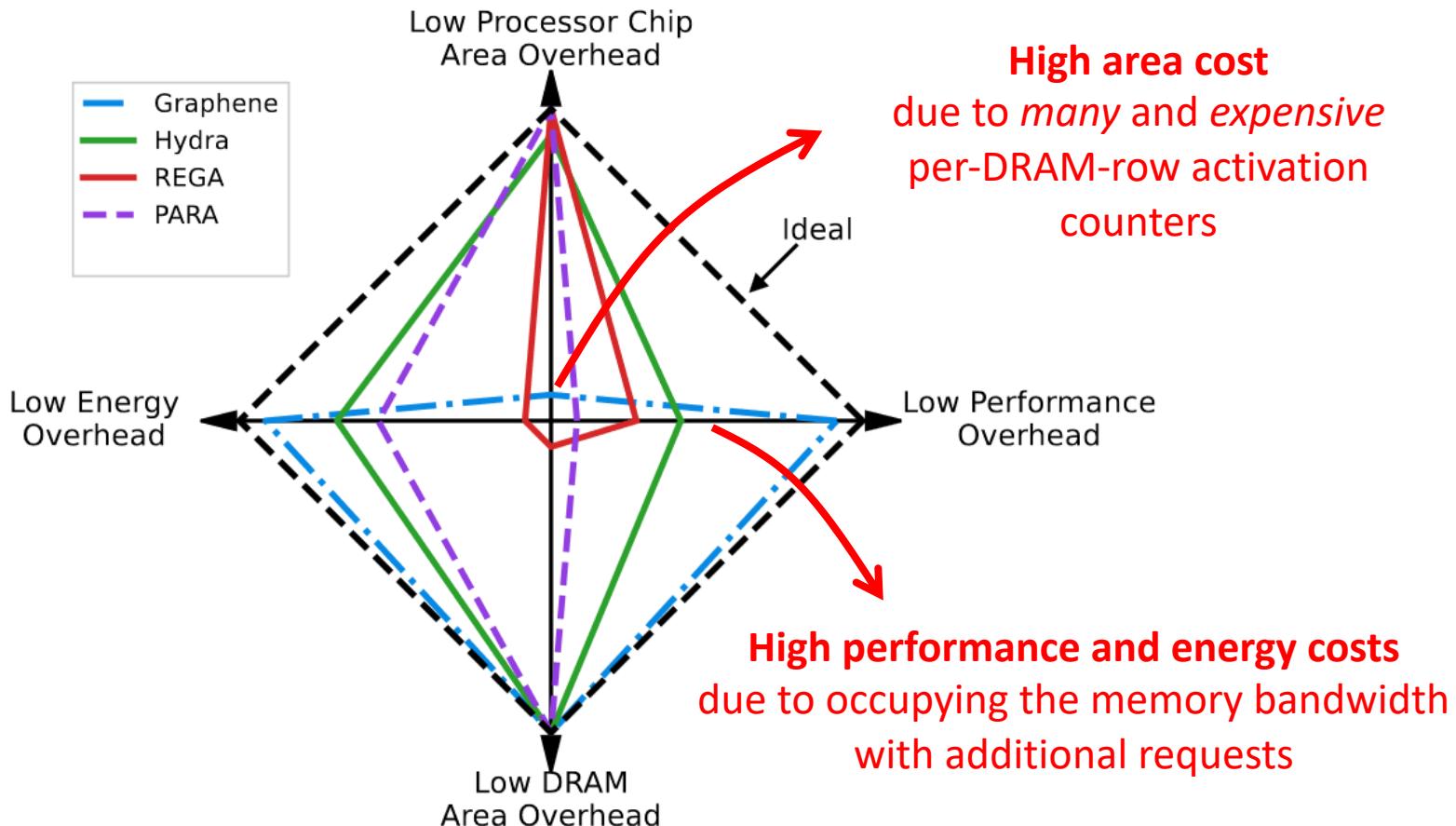
As DRAM becomes more vulnerable to read disturbance, tracking all potential aggressor rows with tag-based counters results in **a high area overhead**

# Existing RowHammer Mitigations (III): Area Optimized Mitigations: Hydra



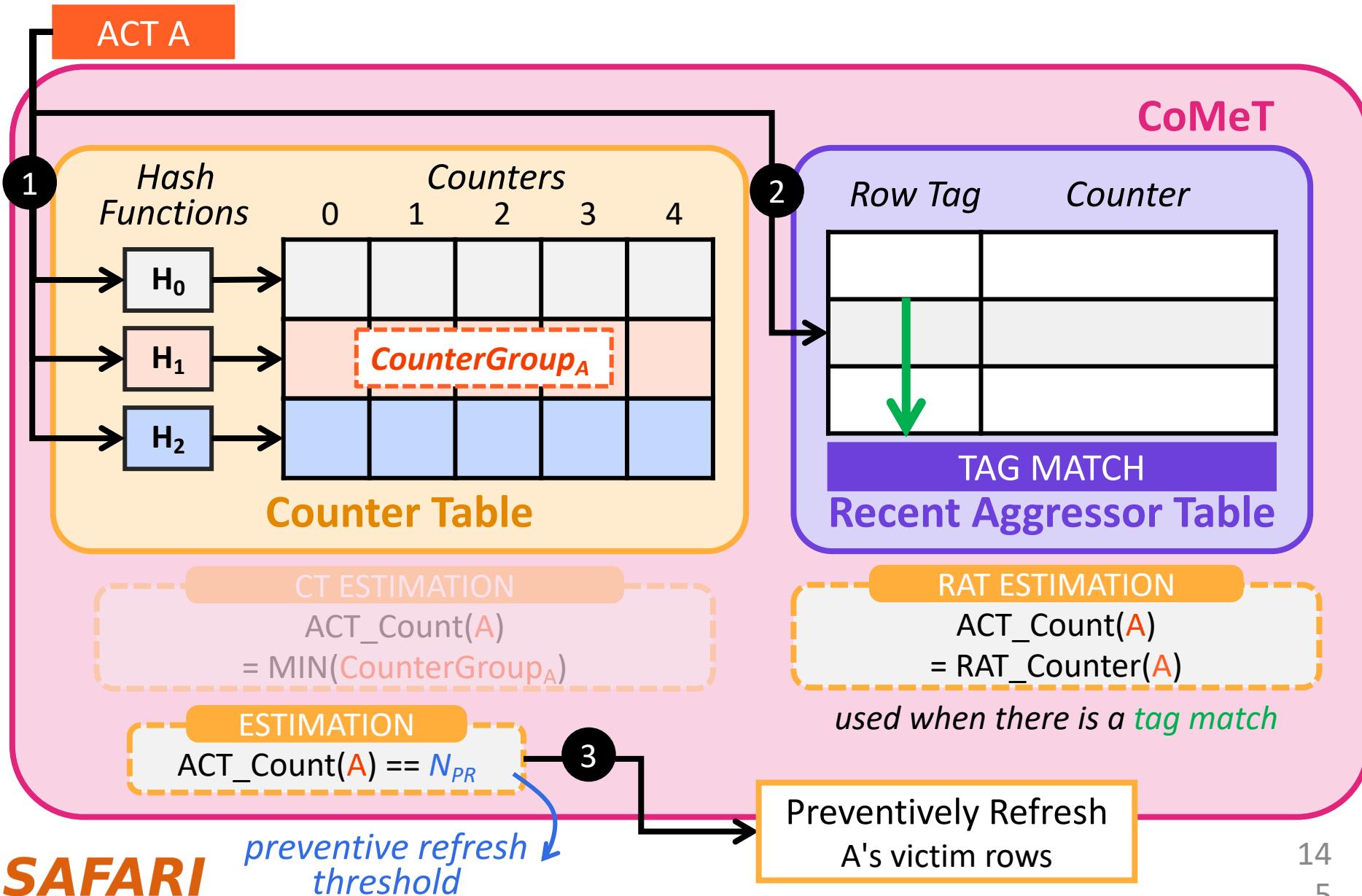
As DRAM becomes more vulnerable to read disturbance, increased off-chip communication results in **high performance and energy overheads**

# Limitations of Existing Mitigations

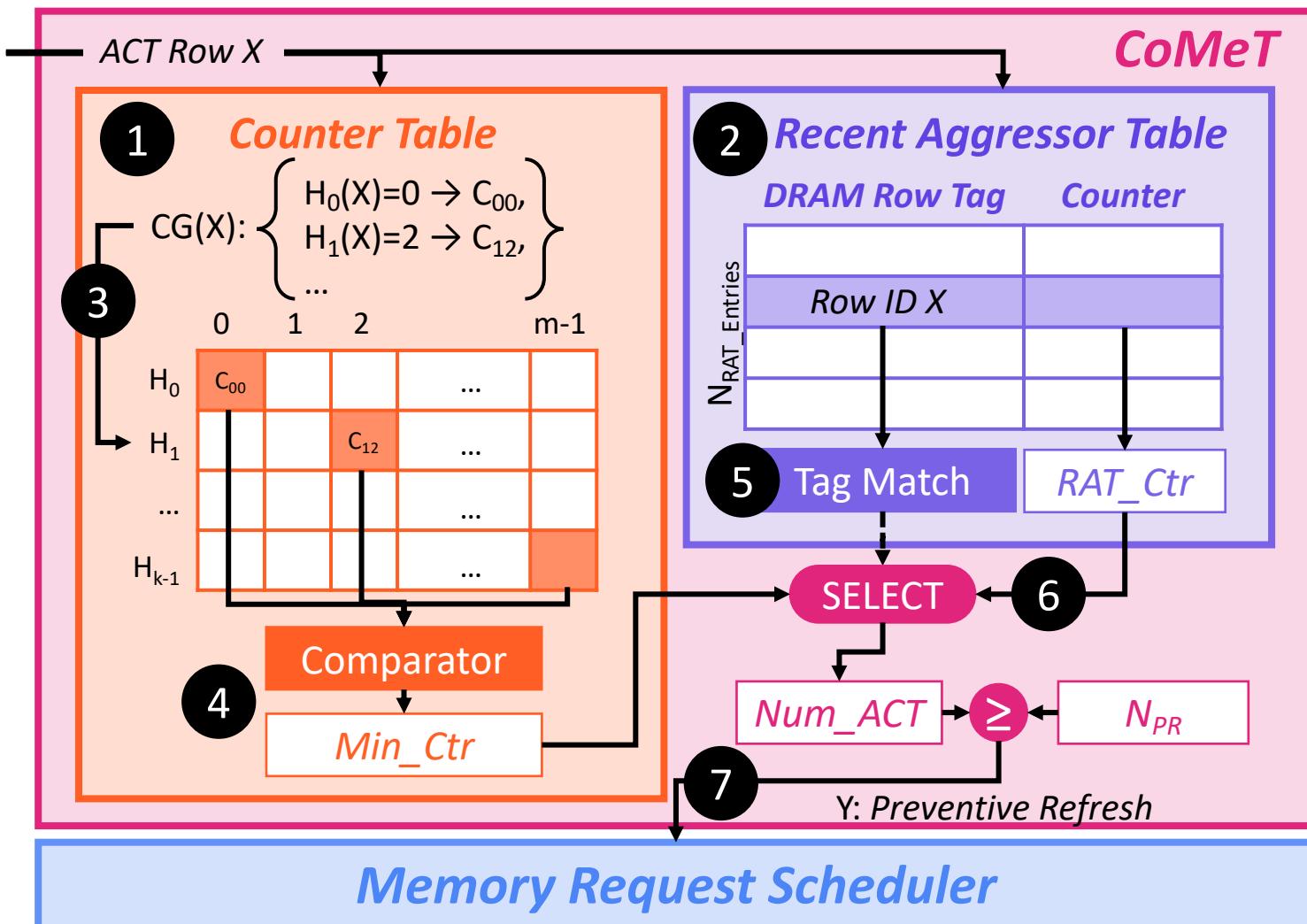


No existing mitigation technique prevents RowHammer bitflips at low area, performance and energy costs

# Operation of CoMeT

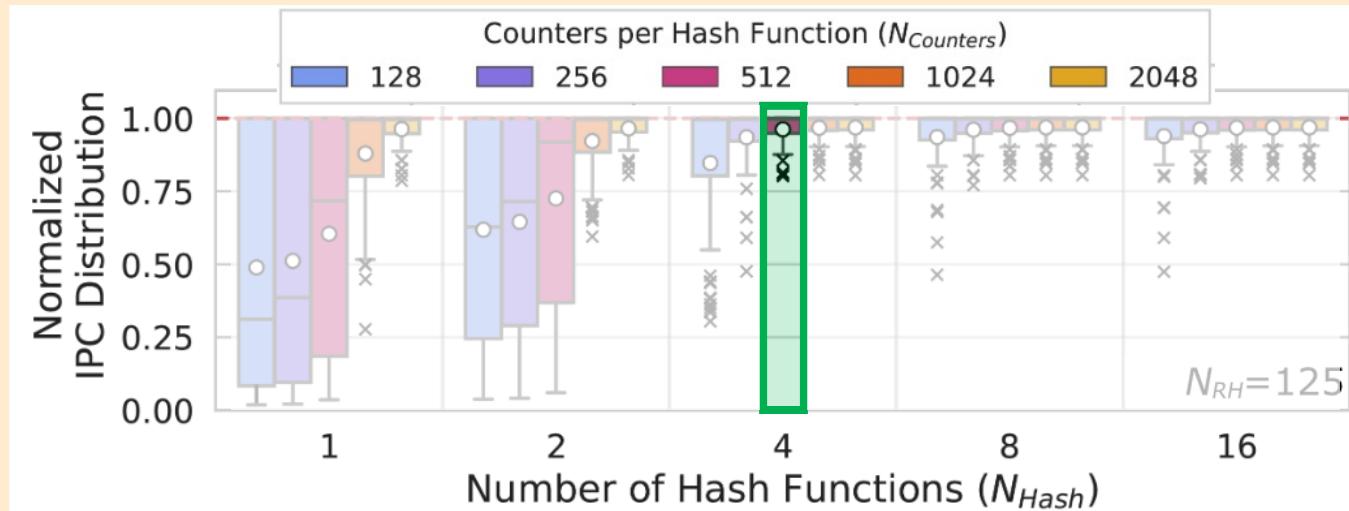


# CoMeT Overview

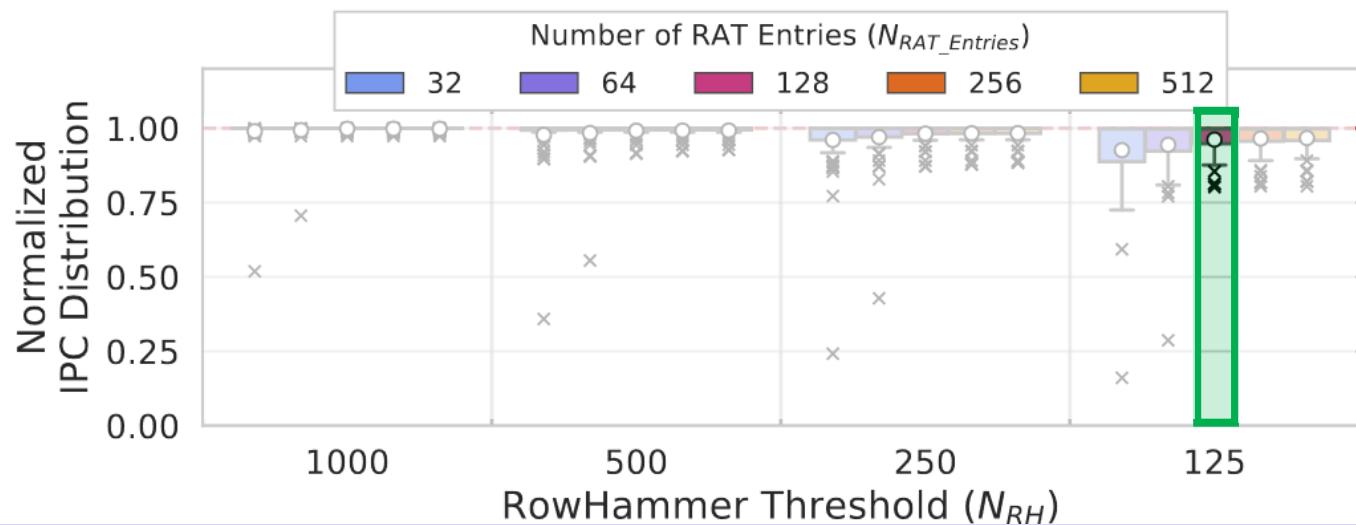


# Configuring CoMeT – Sensitivity Analysis: Counter Table and Recent Aggressor Table

## Configuring Counter Table

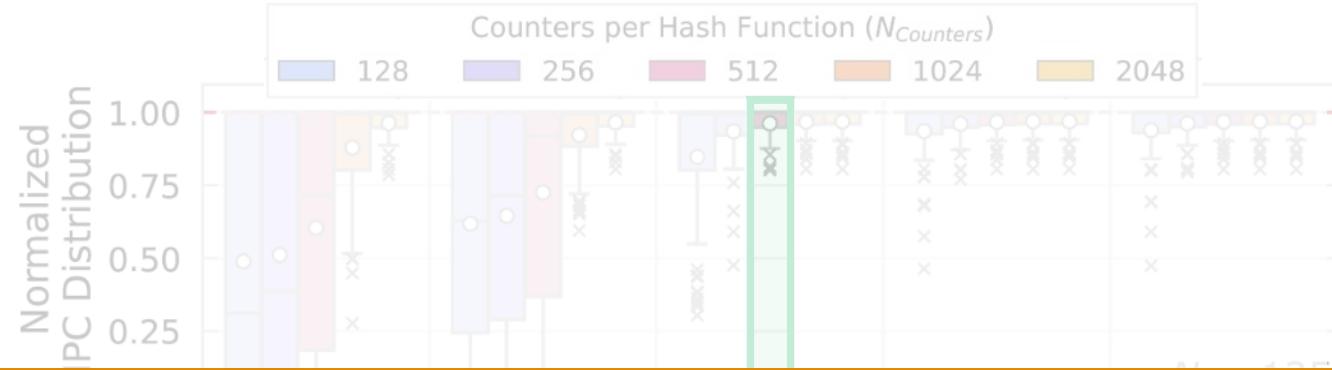


## Configuring RAT



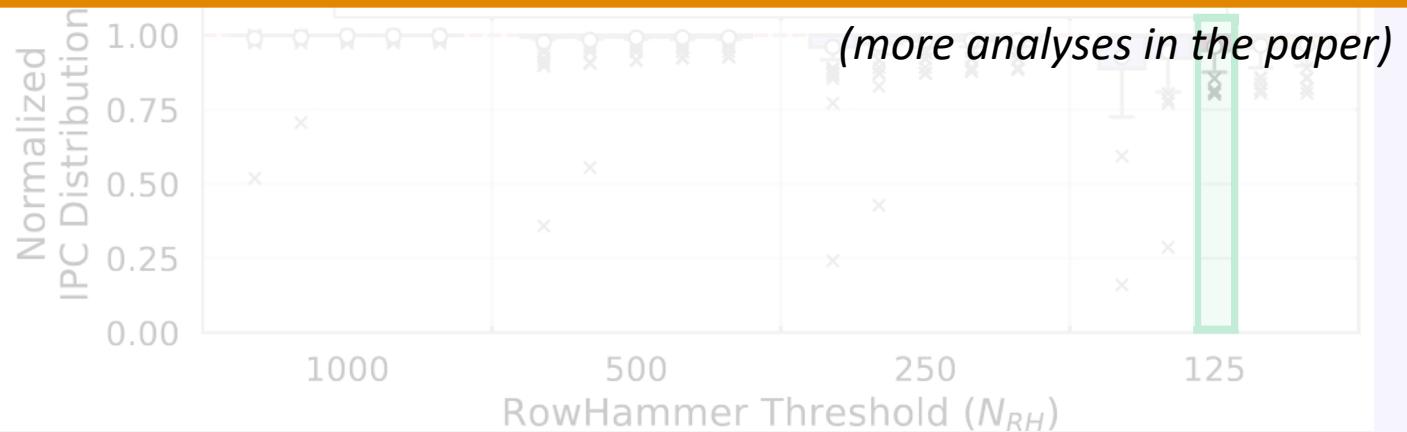
# Configuring CoMeT – Sensitivity Analysis: Counter Table and Recent Aggressor Table

Configuring Counter Table



CoMeT configuration that achieves both performance and area efficiency:  
*Counter Table with 4 hash functions and 512 counters per hash function*  
*Recent Aggressor Table with 128 entry*

Configuring R



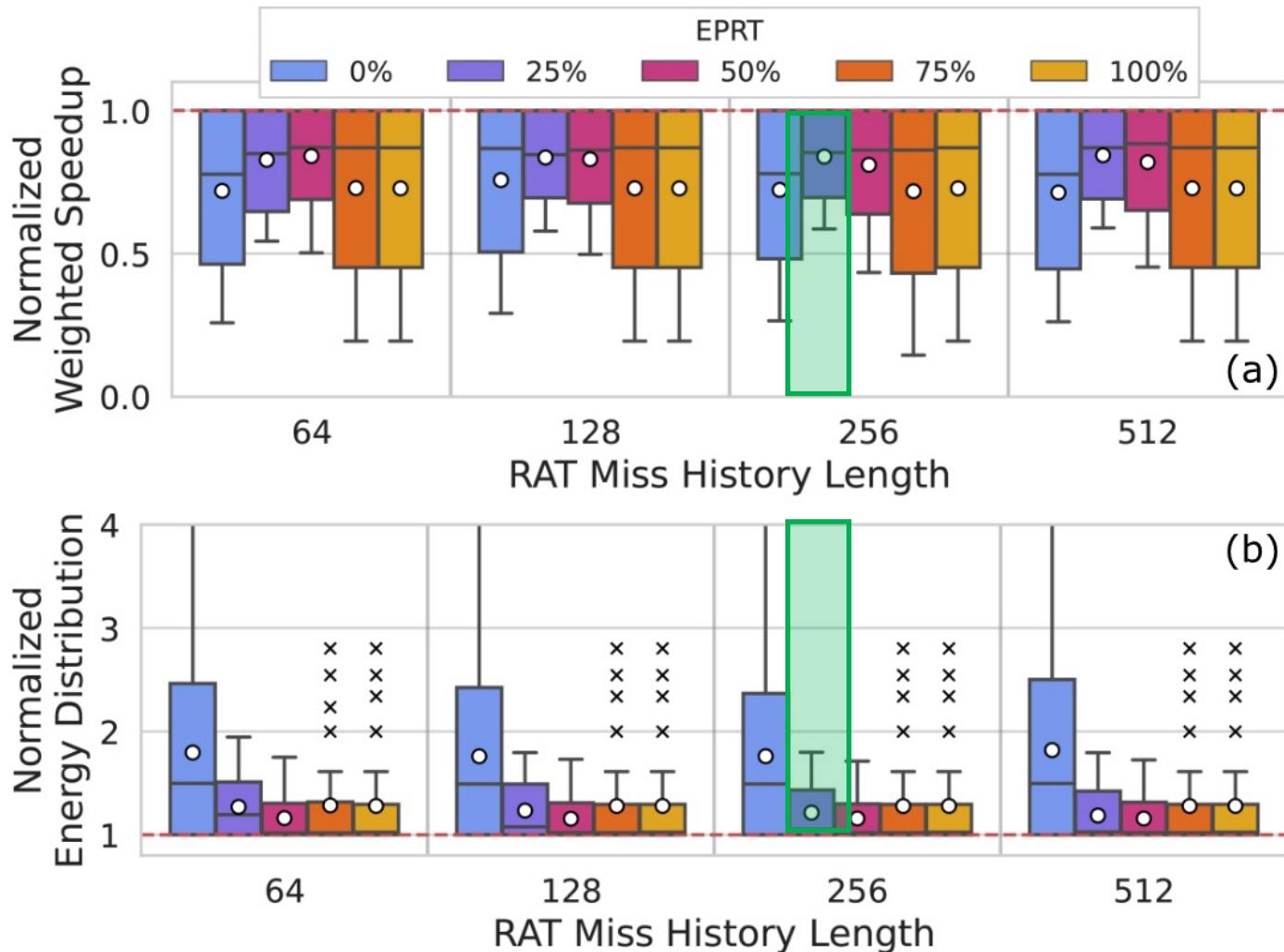
# Configuring CoMeT – Sensitivity Analysis: Counter Reset Period and $N_{PR}$

$$\text{Counter Refresh Period} = \frac{tREFW}{k} \quad (1)$$

$$N_{PR} = \frac{N_{RH}}{k + 1} \quad (2)$$



# The effect of EPRT and RAT Miss History Length on Performance and DRAM energy consumption



# Hardware Implementation

- Storage and area overhead analysis: CACTI
  - Logic circuitry overhead: Verilog HDL implementation and Synopsys DC
- Dual-rank area overhead comparison:

	$N_{RH}=1K$		$N_{RH}=500$		$N_{RH}=250$		$N_{RH}=125$	
	KB	$mm^2$	KB	$mm^2$	KB	$mm^2$	KB	$mm^2$
<b>CoMeT</b>	76.5	0.09	68.0	0.08	59.5	0.07	51.0	0.07
CT (SRAM)	64.0	0.05	56.0	0.05	48.0	0.04	40.0	0.04
RAT (CAM)	12.5	0.03	12.0	0.03	11.5	0.03	11.0	0.02
Logic Circuitry	-	0.005	-	0.005	-	0.005	-	0.005
<b>Graphene [86]</b>	207.2	0.49	398.4	1.13	765.0	3.01	1466.2	4.89
<b>Hydra [90]<sup>8</sup></b>	61.6	0.08	56.5	0.08	51.4	0.07	46.8	0.07

As  $N_{RH}$  decreases, CoMeT's area and storage overheads decrease due to storing fewer bits for its counters

# Hardware Implementation

- Storage and area overhead analysis: CACTI
  - Logic circuitry overhead: Verilog HDL implementation and Synopsys DC
- Dual-rank area overhead comparison:

	$N_{RH}=1K$		$N_{RH}=500$		$N_{RH}=250$		$N_{RH}=125$	
	KB	$mm^2$	KB	$mm^2$	KB	$mm^2$	KB	$mm^2$
<b>CoMeT</b>	76.5	0.09	68.0	0.08	59.5	0.07	51.0	0.07
CT (SRAM)	64.0	0.05	56.0	0.05	48.0	0.04	40.0	0.04
RAT (CAM)	12.5	0.03	12.0	0.03	11.5	0.03	11.0	0.02
Logic Circuitry	-	0.005	-	0.005	-	0.005	-	0.005
<b>Graphene [86]</b>	207.2	0.49	398.4	1.13	765.0	3.01	1466.2	4.89
<b>Hydra [90]<sup>8</sup></b>	61.6	0.08	56.5	0.08	51.4	0.07	46.8	0.07

Compared to the best performing state-of-the-art mitigation,  
CoMeT induces significantly less area overhead

# Hardware Implementation

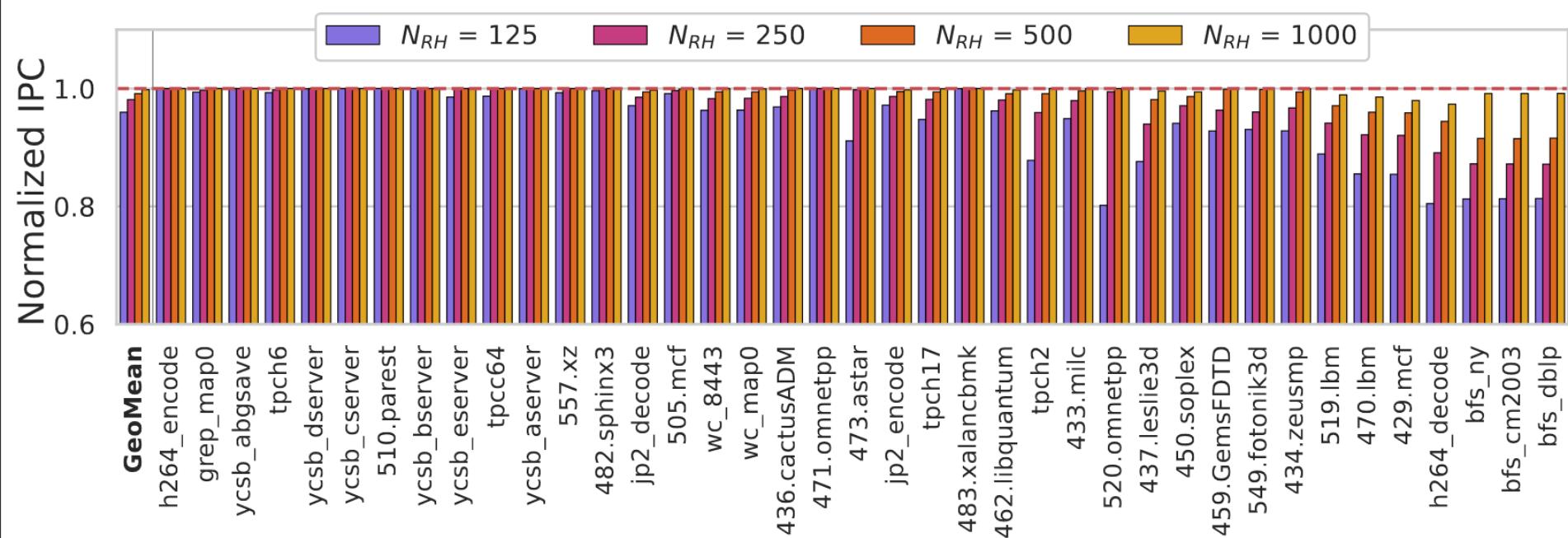
- Storage and area overhead analysis: CACTI
  - Logic circuitry overhead: Verilog HDL implementation and Synopsys DC
- Dual-rank area overhead comparison:

	$N_{RH}=1K$		$N_{RH}=500$		$N_{RH}=250$		$N_{RH}=125$	
	KB	$mm^2$	KB	$mm^2$	KB	$mm^2$	KB	$mm^2$
<b>CoMeT</b>	76.5	0.09	68.0	0.08	59.5	0.07	51.0	0.07
CT (SRAM)	64.0	0.05	56.0	0.05	48.0	0.04	40.0	0.04
RAT (CAM)	12.5	0.03	12.0	0.03	11.5	0.03	11.0	0.02
Logic Circuitry	-	0.005	-	0.005	-	0.005	-	0.005
<b>Graphene [86]</b>	207.2	0.49	398.4	1.13	765.0	3.01	1466.2	4.89
<b>Hydra [90]<sup>8</sup></b>	61.6	0.08	56.5	0.08	51.4	0.07	46.8	0.07

Compared to the best performing low-area-cost mitigation,  
CoMeT induces **similar area overhead**

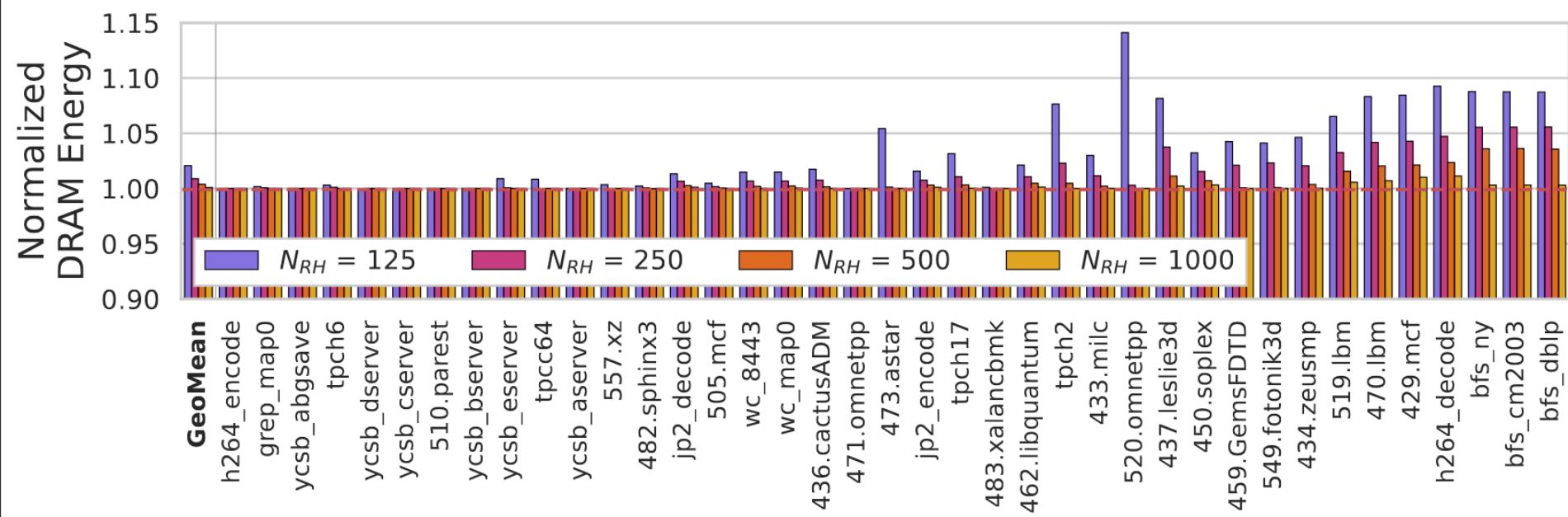
# Single-Core Performance

## Workload Breakdown (Medium and High Intensity)

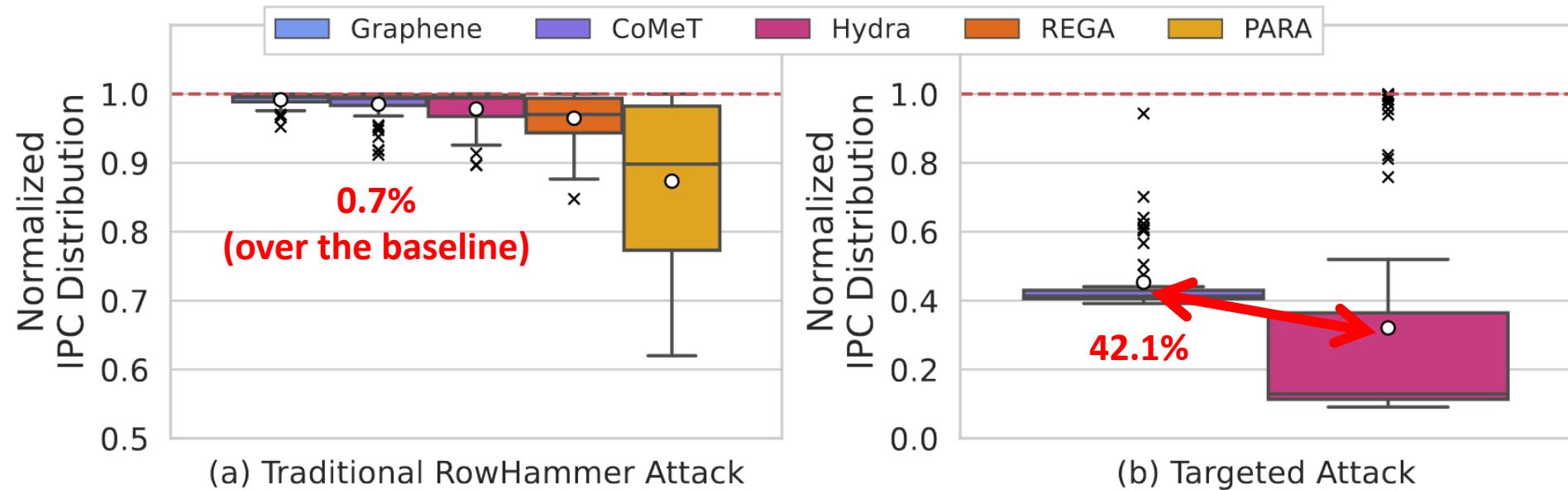


# Single-Core DRAM Energy

## Workload Breakdown (Medium and High Intensity)



# Adversarial Access Patterns



CoMeT incurs negligible additional performance overhead on benign workloads when a traditional RowHammer attack is running at the same time

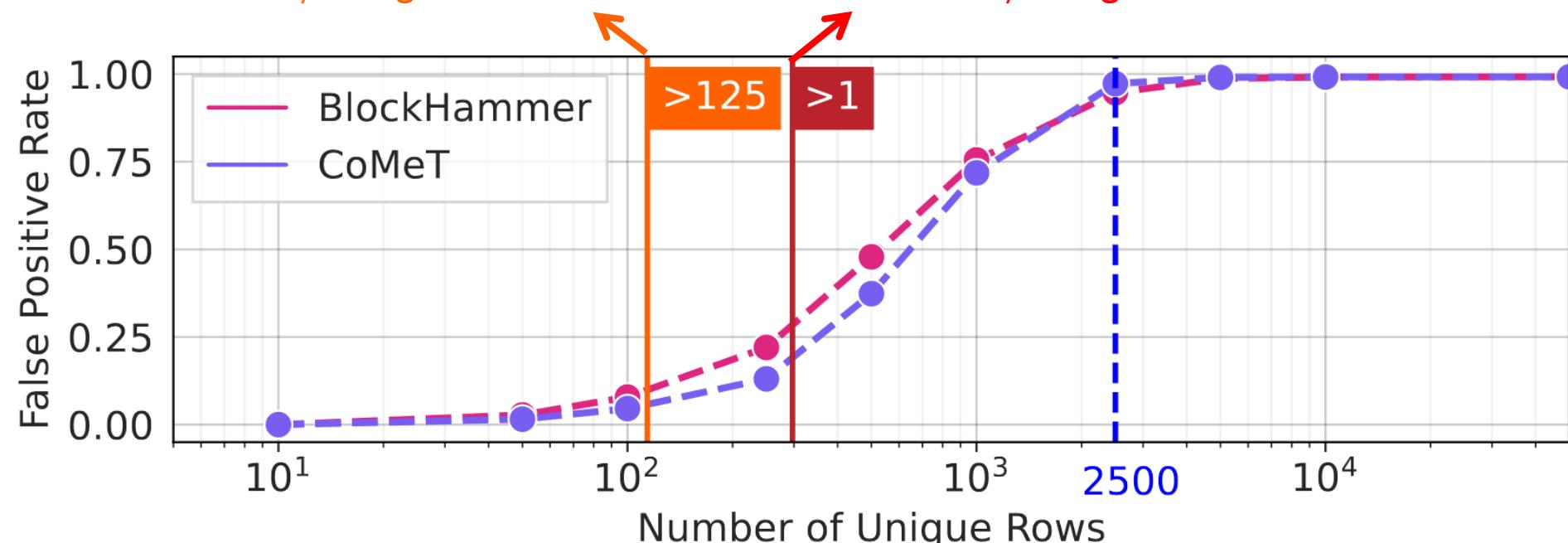
# Comparison Against BlockHammer (I)

## Tracker Comparison

- CoMeT and BlockHammer employs different algorithms and this results in **different DRAM-row-to-counter mappings**

Average # of unique rows that are activated at least 125 times by benign workloads

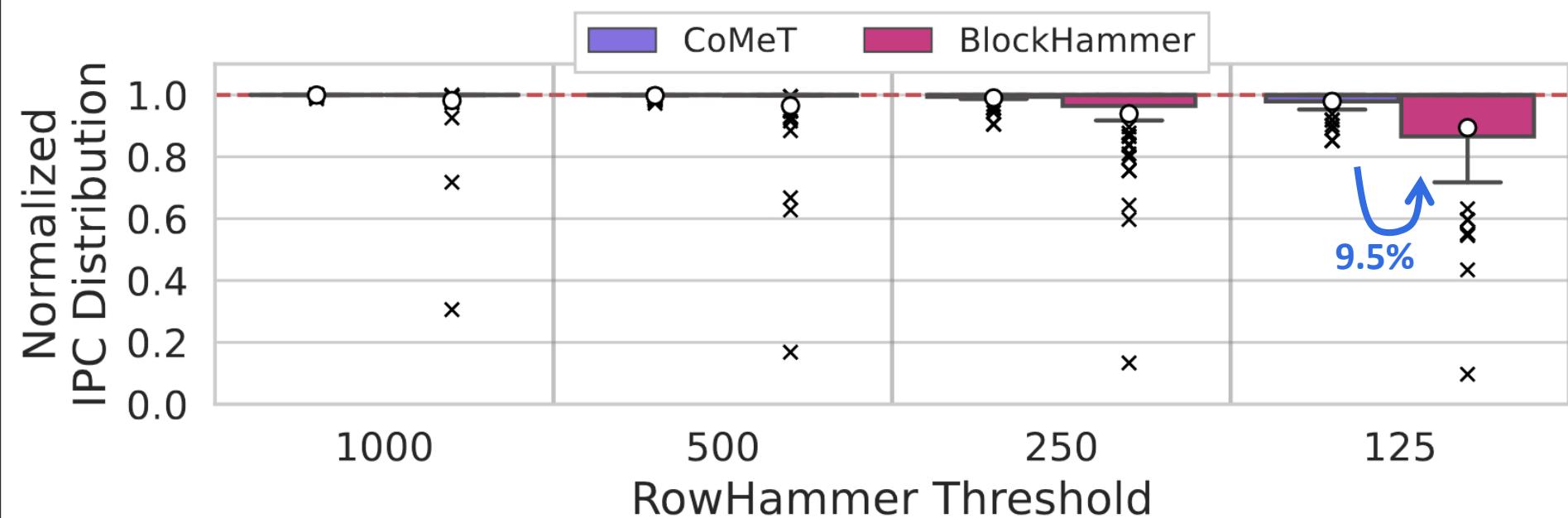
the average number of unique rows touched at least once by benign workloads



When tracking at most 2,500 unique rows,  
CoMeT's tracker outperforms BlockHammer

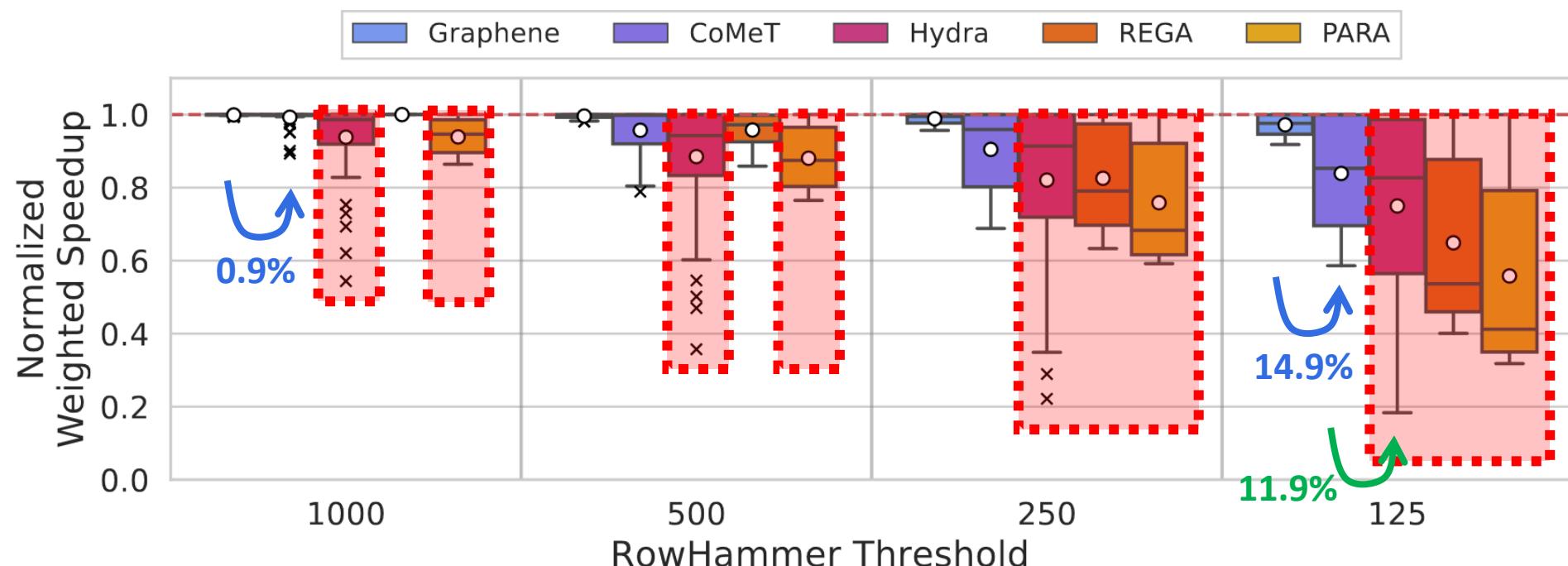
# Comparison Against BlockHammer (II)

## Single-Core Performance Comparison



CoMeT outperforms BlockHammer due to BlockHammer's (i) high false positive rate and  
(ii) increased memory request latencies due to throttling

# Performance Comparison: 8-Core Workloads

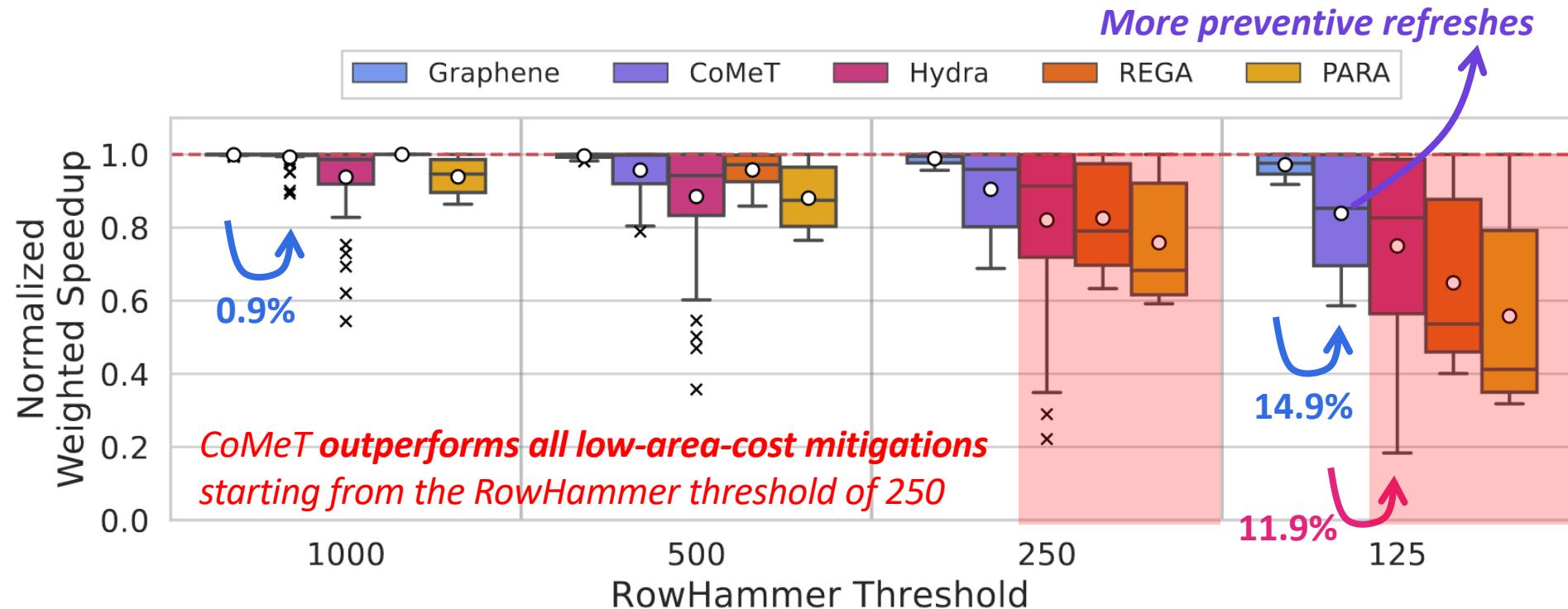


CoMeT's performance overhead over Graphene is **0.9%** and **14.9%**  
at  $N_{RH} = 1K$  and 125, respectively

CoMeT **outperforms** Hydra for all RowHammer thresholds  
(by up to **3.2x** and **11.9%** on average at  $N_{RH} = 125$ )

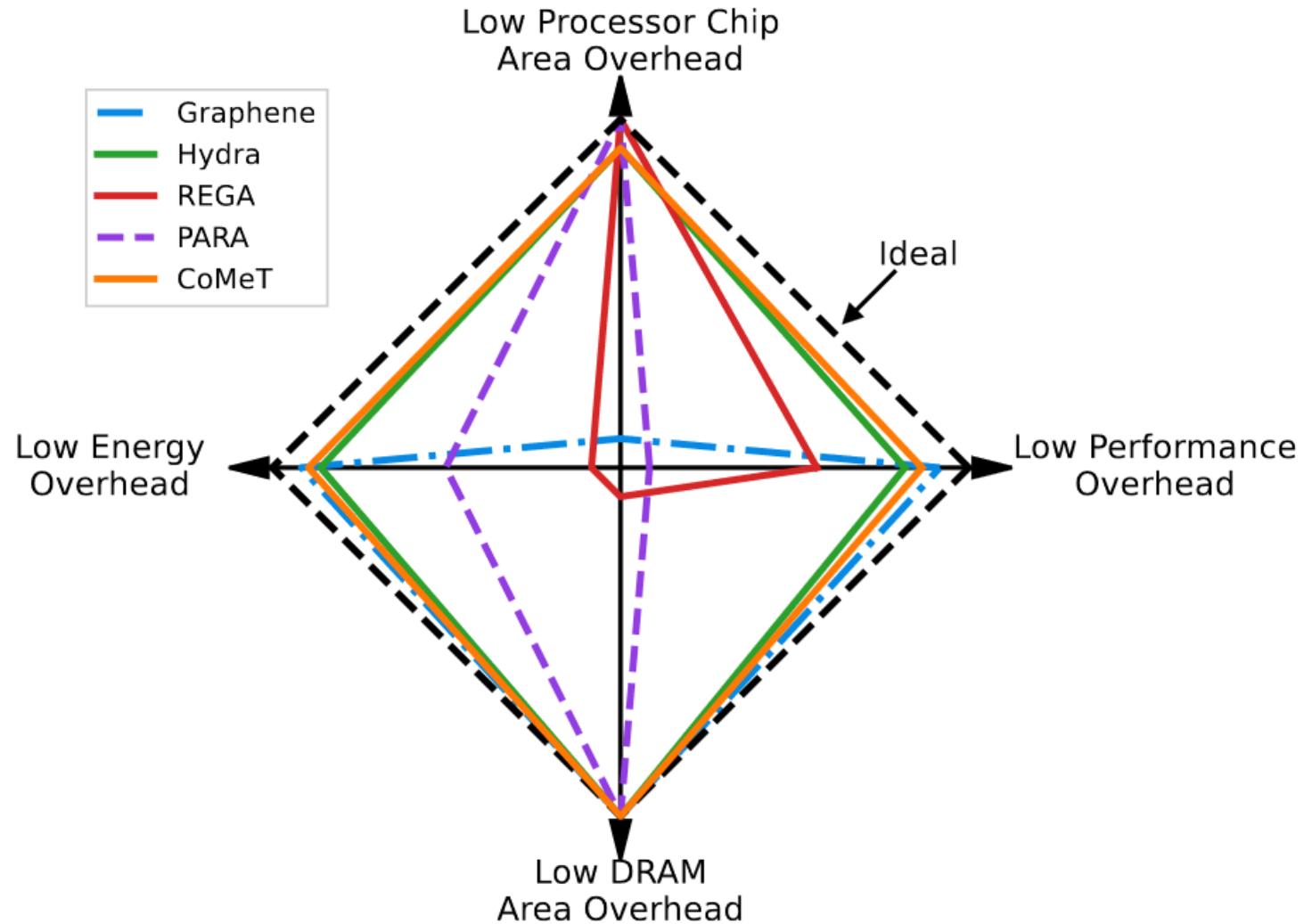
CoMeT **outperforms** *all* low-cost RowHammer mitigations starting from  $N_{RH} = 250$

# Performance Comparison: 8-Core Workloads

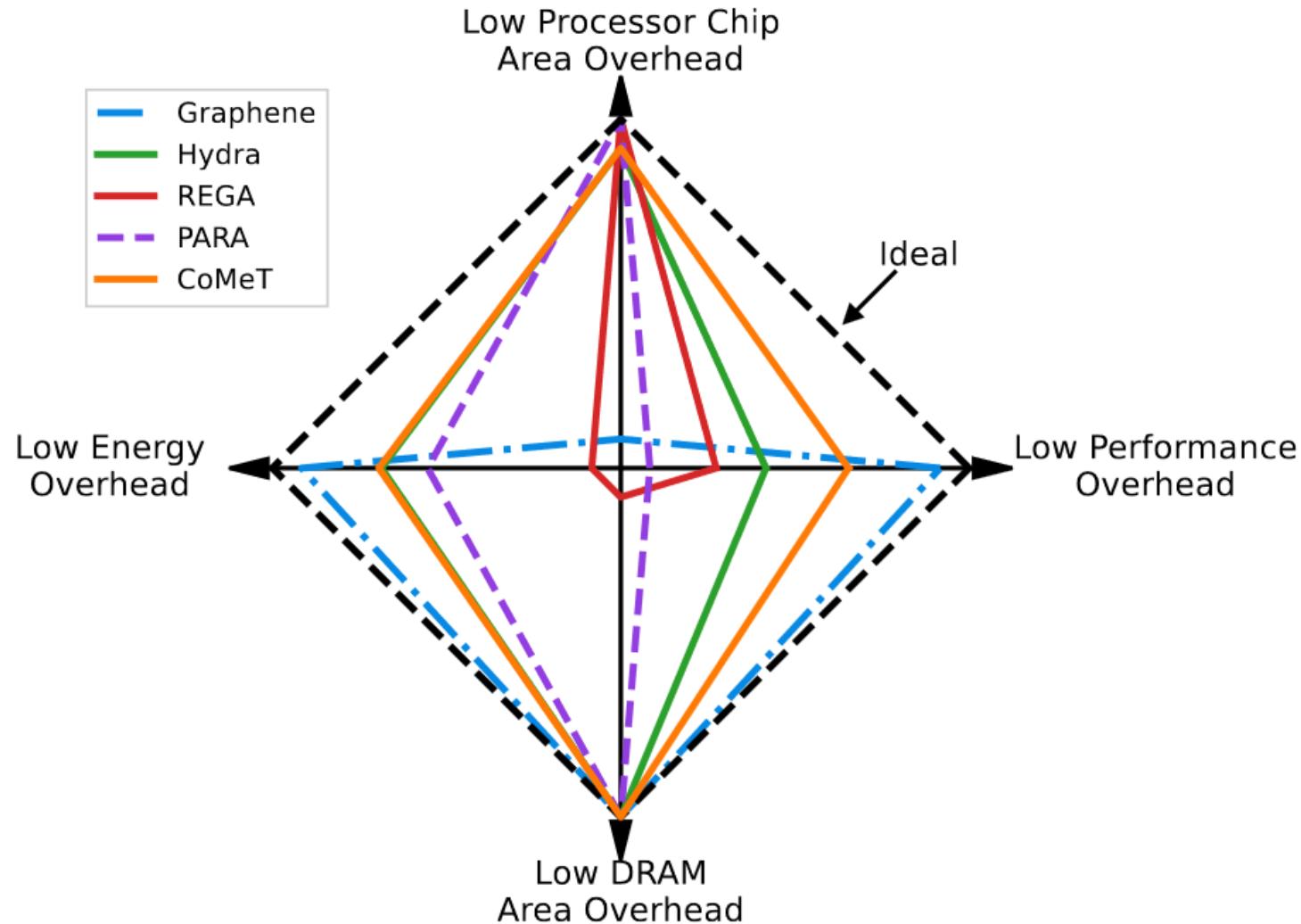


CoMeT maintains a performance overhead between Graphene's and Hydra's performance overheads

# Single-Core Comparison – Radar Chart



# Multi-Core Comparison – Radar Chart



# Evaluated Workloads and Their Characteristics

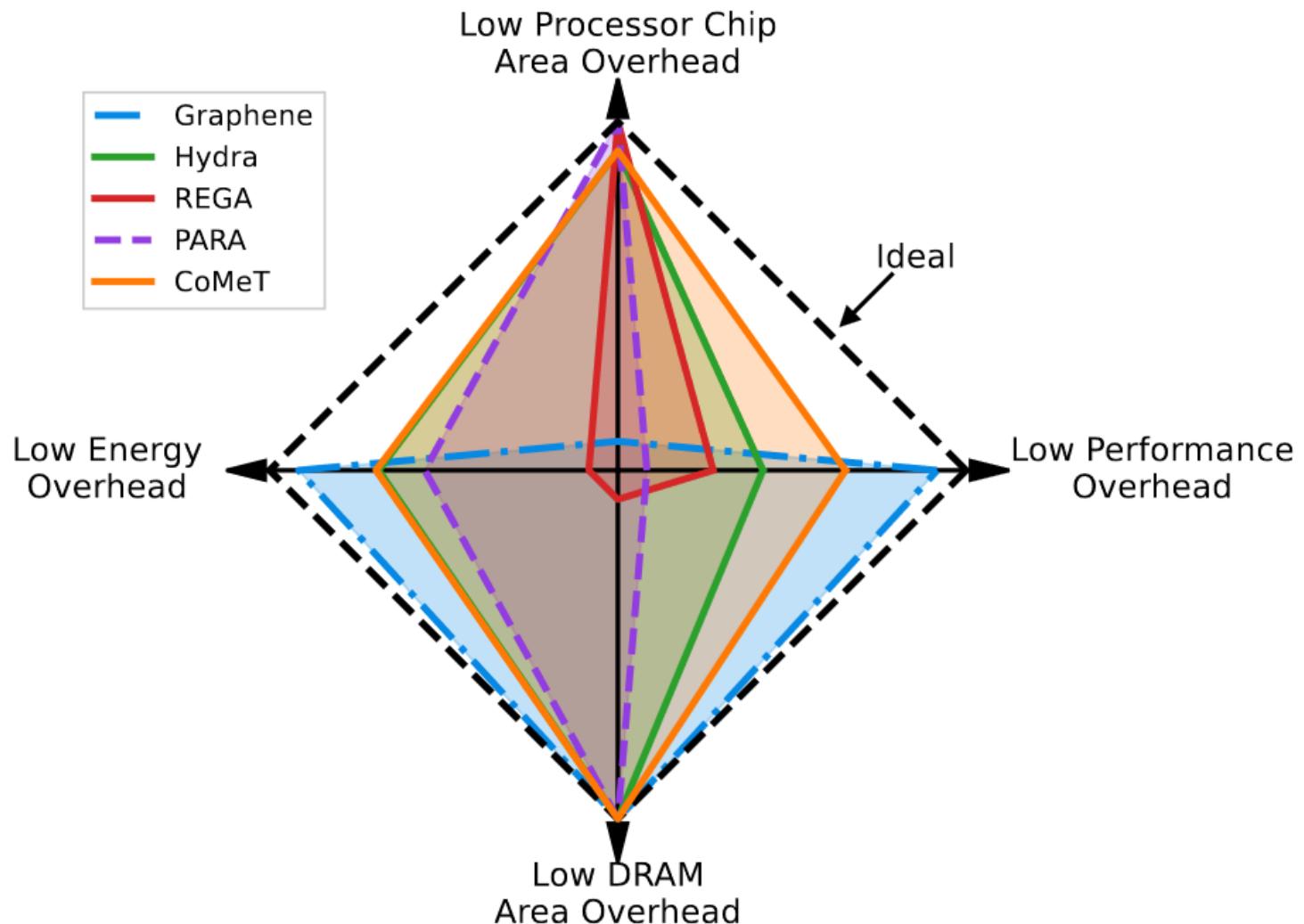
**Table 3: Evaluated Workloads and Their Characteristics**

RBMPKI	Workloads
[10+) (High)	519.lbm (5049 MB/s), 459.GemsFDTD (4788 MB/s), 450.soplex (3212 MB/s), h264_decode (11284 MB/s), 520.omnetpp (2567 MB/s), 433.milc (3595 MB/s), 434.zeusmp (5115 MB/s), bfs dblp (12135 MB/s), 429.mcf (5588 MB/s), 549.fotonik3d (4428 MB/s), 470.lbm (6489 MB/s), bfs_ny (12146 MB/s), bfs_cm2003 (12138 MB/s), 437.leslie3d (3806 MB/s)
[2, 10) (Medium)	510.parest (92 MB/s), 462.libquantum (6089 MB/s), tpch2 (3612 MB/s), wc_8443 (1772 MB/s), ycsb_aserver (1080 MB/s), 473.astar (2473 MB/s), jp2_decode (1390 MB/s), 436.cactusADM (1915 MB/s), 557.xz (1113 MB/s), ycsb_cserver (842 MB/s), ycsb_eserver (721 MB/s), 471.omnetpp (96 MB/s), 483.xalancbmk (187 MB/s), 505.mcf (3760 MB/s), wc_map0 (1768 MB/s), jp2_encode (1706 MB/s), tpch17 (2553 MB/s), ycsb_bserver (854 MB/s), tpcc64 (1472 MB/s), 482.sphinx3 (968 MB/s)
[0, 2) (Low)	502.gcc (180 MB/s), 544.nab (78 MB/s), h264_encode (0.10 MB/s), 507.cactuBSSN (1325 MB/s), 525.x264 (109 MB/s), ycsb_dserver (659 MB/s), 531.deepsjeng (105 MB/s), 526.blender (56 MB/s), 435.gromacs (259 MB/s), 523.xalancbmk (180 MB/s), 447.dealII (24 MB/s), 508.namd (104 MB/s), 538.imagick (8 MB/s), 445.gobmk (97 MB/s), 444.namd (104 MB/s), 464.h264ref (17 MB/s), ycsb_abgsave (362 MB/s), 458.sjeng (131 MB/s), 541.leela (4 MB/s), tpch6 (675 MB/s), 511.povray (1 MB/s), 456.hmmer (28 MB/s), 481.wrf (7 MB/s), grep_map0 (381 MB/s), 500.perlbench (642 MB/s), 403.gcc (79 MB/s), 401.bzip2 (59 MB/s)

# CoMeT's Performance at High RowHammer Thresholds

We evaluate the performance of 61 single-core applications at high RowHammer thresholds of 2000 and 4000. We observe that CoMeT incurs 0.015% and 0.0053% average performance overheads, at  $N_{RH} = 2000$  and  $N_{RH} = 4000$ , respectively. We conclude that CoMeT has negligible performance overhead at high RowHammer thresholds.

# Summary of the Results



# Storage Overhead of Graphene

**Table 1: Storage overhead of a performance-optimized state-of-the-art RowHammer mitigation [86].**

$N_{RH}$	1000	500	250	125
Storage (KB)	207.19	498.44	765.00	1466.25

# Performance Overhead of Hydra

