Computer Architecture
Seminar/Proseminar
Lecture 1b: Example Presentation

A. Giray Yaglikci

27 April 2026

S4L ROOTSEC CISPA HELMHOLTZ CENTER FOR INFORMATION SECURITY UNIVERSITÄT DES SAARLANDES

BreakHammer: Enhancing RowHammer Mitigations by Carefully Throttling Suspect Threads

Oğuzhan Canpolat A. Giray Yağlıkçı Ataberk Olgun İsmail E. Yüksel

Yahya C. Tuğrul Konstantinos Kanellopoulos Oğuz Ergin Onur Mutlu

SAFARI ETH zürich kasırga

Venue : *MICRO 2024*
Presenter : *A. Giray Yaglikci*
Date : *27 April 2026*

S4L ROOTSEC CISPA HELMHOLTZ CENTER FOR INFORMATION SECURITY UNIVERSITÄT DES SAARLANDES

Executive Summary

Problem:
- DRAM continues to become more vulnerable to RowHammer
- RowHammer-preventive actions are time consuming and block access to memory

Key Exploit: Mount a memory performance attack by triggering RowHammer-preventive actions to block memory access for long time periods

Goal: Reduce the performance overhead of RowHammer mitigation mechanisms by reducing the number of performed RowHammer-preventive actions without compromising system robustness

Key Idea: Throttle threads that frequently trigger RowHammer solutions

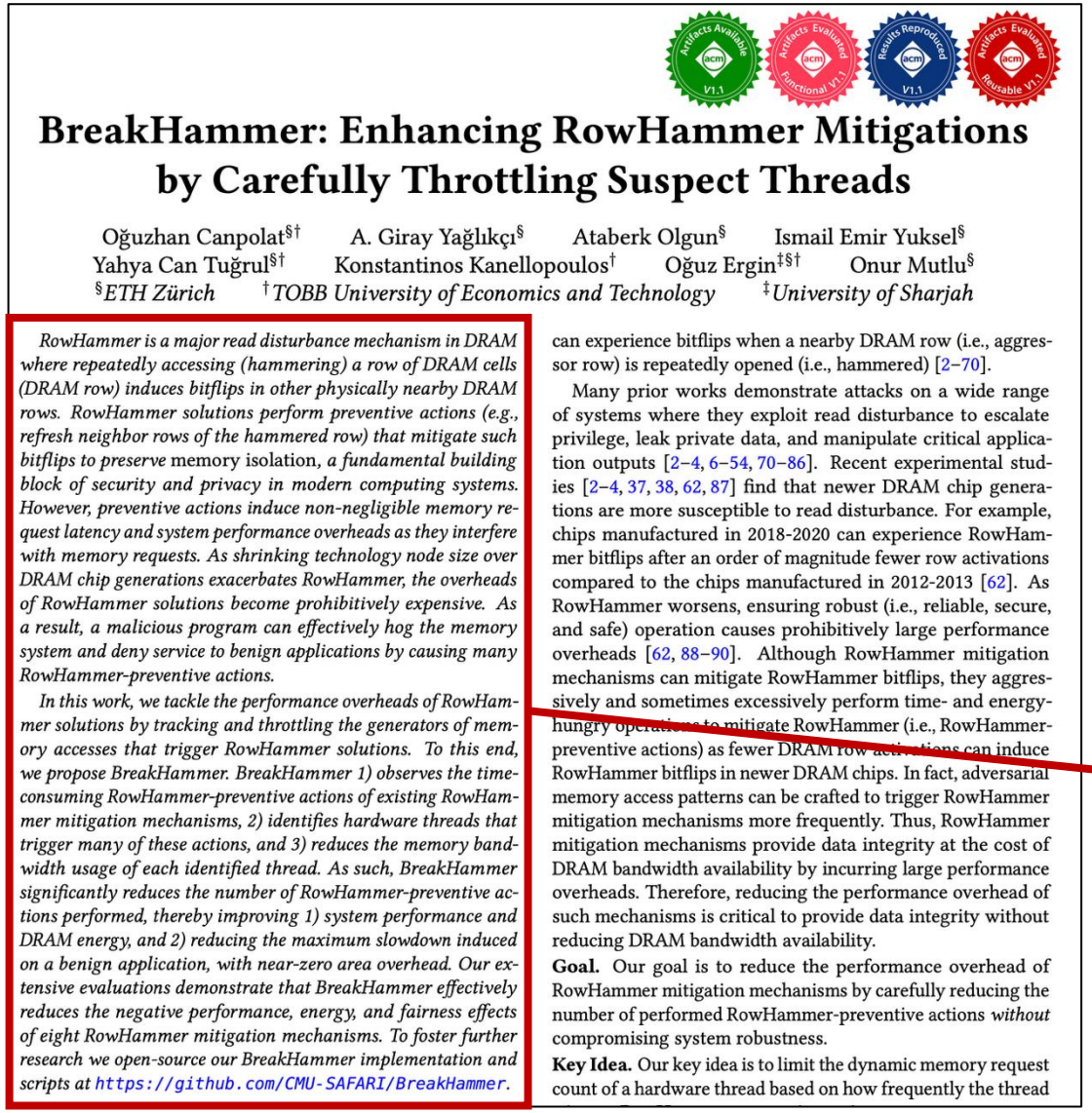
Key Mechanism: BreakHammer
- Observes triggered RowHammer-preventive actions
- Identifies threads that trigger many preventive actions (i.e., suspect threads)
- Reduces the memory bandwidth usage of the suspect threads

Key Results: BreakHammer significantly reduces the negative effects of RowHammer mitigation mechanisms on performance, energy, and fairness

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

S4L ROOTSEC CISPA HELMHOLTZ CENTER FOR INFORMATION SECURITY UNIVERSITÄT DES SAARLANDES

Taking a Step Back: What is an Executive Summary?



BreakHammer: Enhancing RowHammer Mitigations by Carefully Throttling Suspect Threads

Oğuzhan Canpolat[§†] A. Giray Yağlıkçı[§] Ataberk Olgun[§] Ismail Emir Yuksel[§]
Yahya Can Tuğrul[§†] Konstantinos Kanellopoulos[†] Oğuz Ergin[‡§†] Onur Mutlu[§]
[§]*ETH Zürich* [†]*TOBB University of Economics and Technology* [‡]*University of Sharjah*

RowHammer is a major read disturbance mechanism in DRAM where repeatedly accessing (hammering) a row of DRAM cells (DRAM row) induces bitflips in other physically nearby DRAM rows. RowHammer solutions perform preventive actions (e.g., refresh neighbor rows of the hammered row) that mitigate such bitflips to preserve memory isolation, *a fundamental building block of security and privacy in modern computing systems. However, preventive actions induce non-negligible memory request latency and system performance overheads as they interfere with memory requests. As shrinking technology node size over DRAM chip generations exacerbates RowHammer, the overheads of RowHammer solutions become prohibitively expensive. As a result, a malicious program can effectively hog the memory system and deny service to benign applications by causing many RowHammer-preventive actions.*

In this work, we tackle the performance overheads of RowHammer solutions by tracking and throttling the generators of memory accesses that trigger RowHammer solutions. To this end, we propose BreakHammer. BreakHammer 1) observes the time-consuming RowHammer-preventive actions of existing RowHammer mitigation mechanisms, 2) identifies hardware threads that trigger many of these actions, and 3) reduces the memory bandwidth usage of each identified thread. As such, BreakHammer significantly reduces the number of RowHammer-preventive actions performed, thereby improving 1) system performance and DRAM energy, and 2) reducing the maximum slowdown induced on a benign application, with near-zero area overhead. Our extensive evaluations demonstrate that BreakHammer effectively reduces the negative performance, energy, and fairness effects of eight RowHammer mitigation mechanisms. To foster further research we open-source our BreakHammer implementation and scripts at https://github.com/CMU-SAFARI/BreakHammer.

can experience bitflips when a nearby DRAM row (i.e., aggressor row) is repeatedly opened (i.e., hammered) [2–70].

Many prior works demonstrate attacks on a wide range of systems where they exploit read disturbance to escalate privilege, leak private data, and manipulate critical application outputs [2–4, 6–54, 70–86]. Recent experimental studies [2–4, 37, 38, 62, 87] find that newer DRAM chip generations are more susceptible to read disturbance. For example, chips manufactured in 2018-2020 can experience RowHammer bitflips after an order of magnitude fewer row activations compared to the chips manufactured in 2012-2013 [62]. As RowHammer worsens, ensuring robust (i.e., reliable, secure, and safe) operation causes prohibitively large performance overheads [62, 88–90]. Although RowHammer mitigation mechanisms can mitigate RowHammer bitflips, they aggressively and sometimes excessively perform time- and energy-hungry operations to mitigate RowHammer (i.e., RowHammer-preventive actions) as fewer DRAM row activations can induce RowHammer bitflips in newer DRAM chips. In fact, adversarial memory access patterns can be crafted to trigger RowHammer mitigation mechanisms more frequently. Thus, RowHammer mitigation mechanisms provide data integrity at the cost of DRAM bandwidth availability by incurring large performance overheads. Therefore, reducing the performance overhead of such mechanisms is critical to provide data integrity without reducing DRAM bandwidth availability.

Goal. Our goal is to reduce the performance overhead of RowHammer mitigation mechanisms by carefully reducing the number of performed RowHammer-preventive actions *without* compromising system robustness.

Key Idea. Our key idea is to limit the dynamic memory request count of a hardware thread based on how frequently the thread

- Summary of the paper in a nutshell
- Tells what this paper/talk about and sets expectations
- Serves the same purpose as the paper's abstract
- Helps the reader/audience to decide if they want to
  - read the rest of the paper
  - pay attention to the rest of the talk

Executive Summary

Problem:
- DRAM continues to become more vulnerable to RowHammer
- RowHammer-preventive actions are time consuming and block access to memory

Key Exploit: Mount a memory performance attack by triggering RowHammer-preventive actions to block memory access for long time periods

Goal: Reduce the performance overhead of RowHammer mitigation mechanisms by reducing the number of performed RowHammer-preventive actions without compromising system robustness

Key Idea: Throttle threads that frequently trigger RowHammer solutions

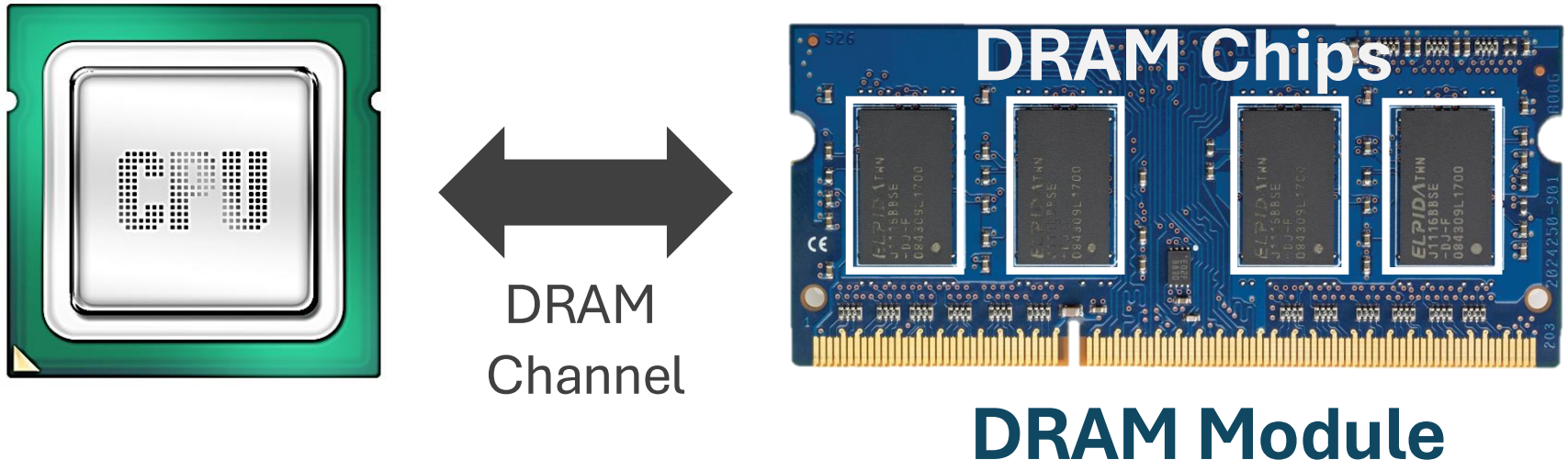
Key Mechanism: BreakHammer
- Observes triggered RowHammer-preventive actions
- Identifies threads that trigger many preventive actions (i.e., suspect threads)
- Reduces the memory bandwidth usage of the suspect threads

Key Results: BreakHammer significantly reduces the negative effects of RowHammer mitigation mechanisms on performance, energy, and fairness

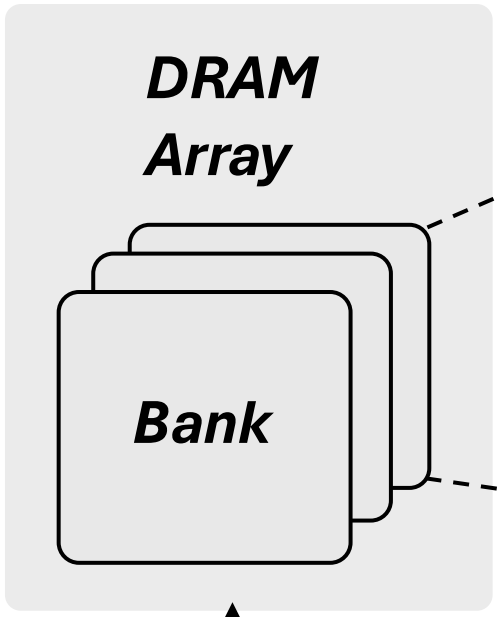
https://github.com/CMU-SAFARI/BreakHammer

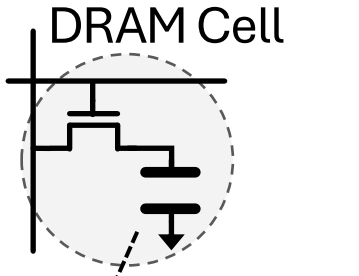
Outline

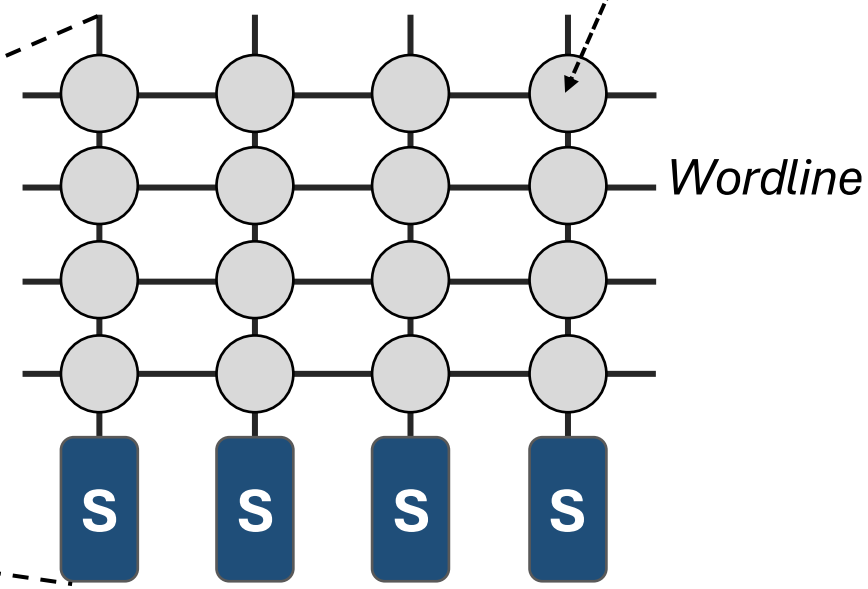
DRAM Organization



DRAM Organization



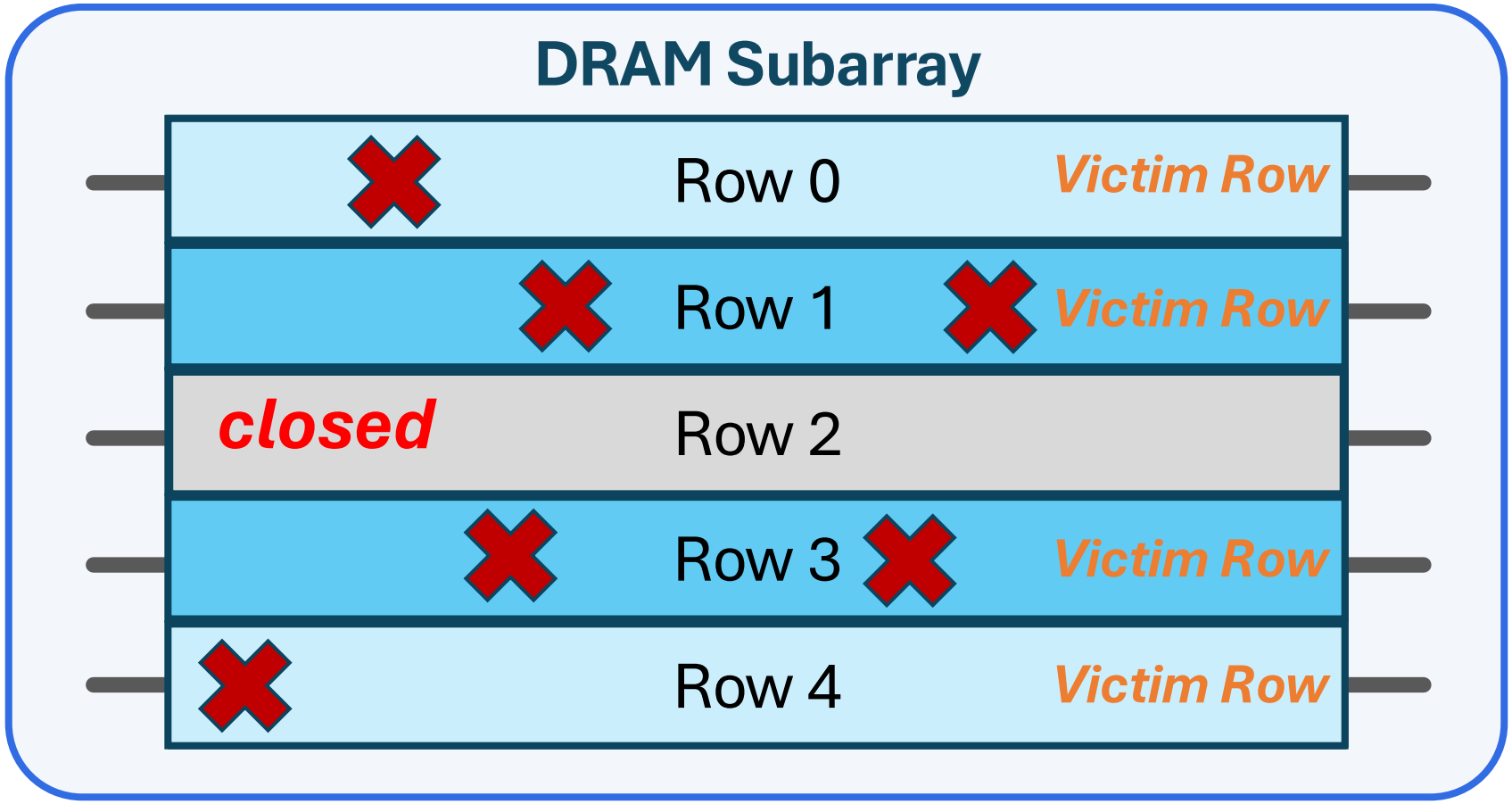




Sense Amplifiers
(Row Buffer)

Off-Chip Channel

RowHammer: A Prime Example of Read Disturbance

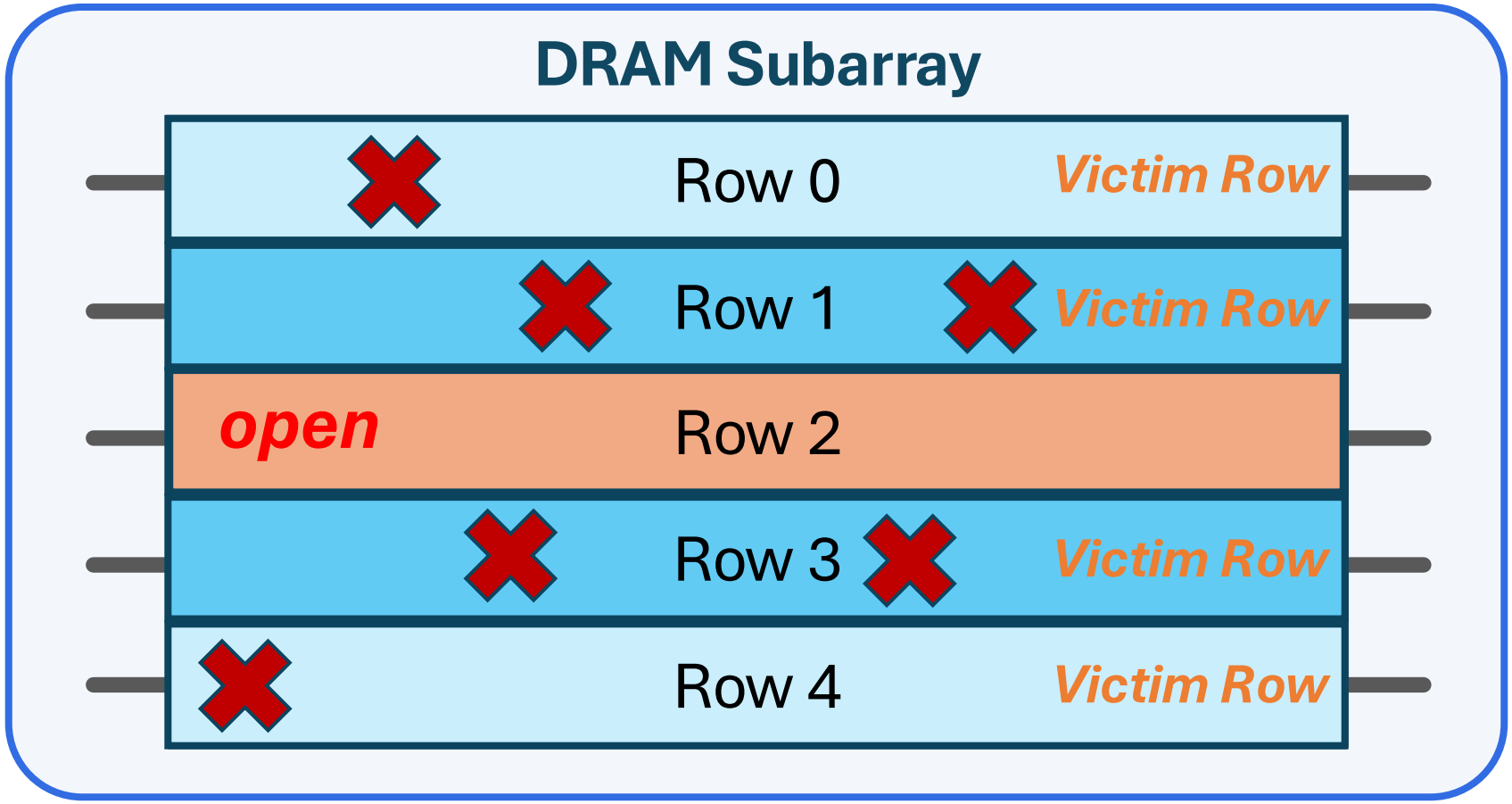


Repeatedly **opening** (activating) and **closing** (precharging)

a DRAM row causes **read disturbance bitflips** in nearby cells

[Kim+ ISCA'20]

Read Disturbance Vulnerabilities (I)

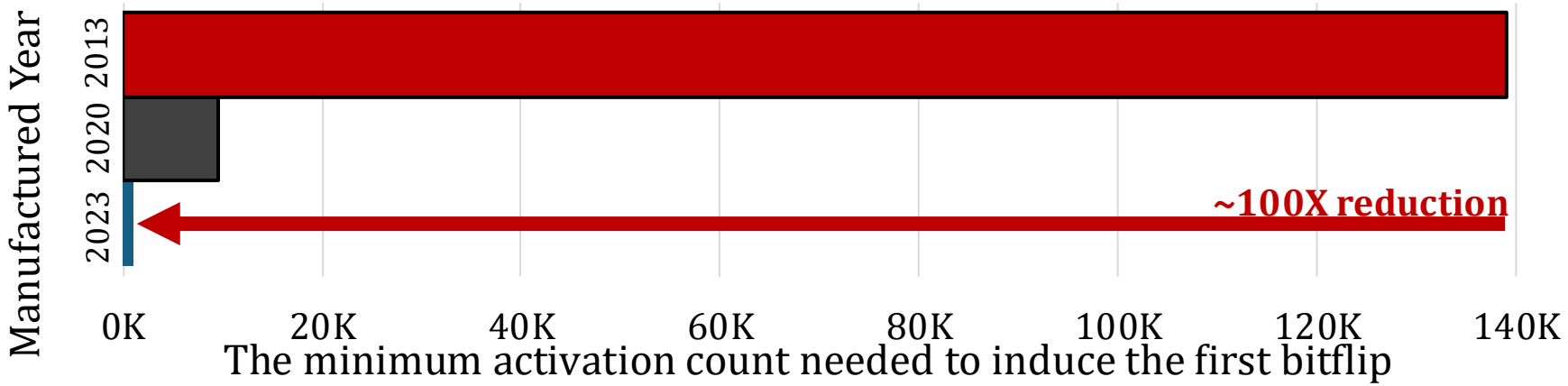


The minimum number of activations that causes a bitflip is called **the RowHammer threshold (N_{RH})**

[Kim+ ISCA'20]

Read Disturbance Vulnerabilities (II)

It is **critical** to prevent read disturbance bitflips **effectively** and **efficiently** for highly vulnerable systems



Read Disturbance Vulnerabilities (II)

This is a takeaway box, use it

Explicitly tells the audience
your take from the slide
and what they should remember

It is **critical** to prevent read disturbance bitflips
effectively and **efficiently** for highly vulnerable systems

RowHammer-Preventive Actions

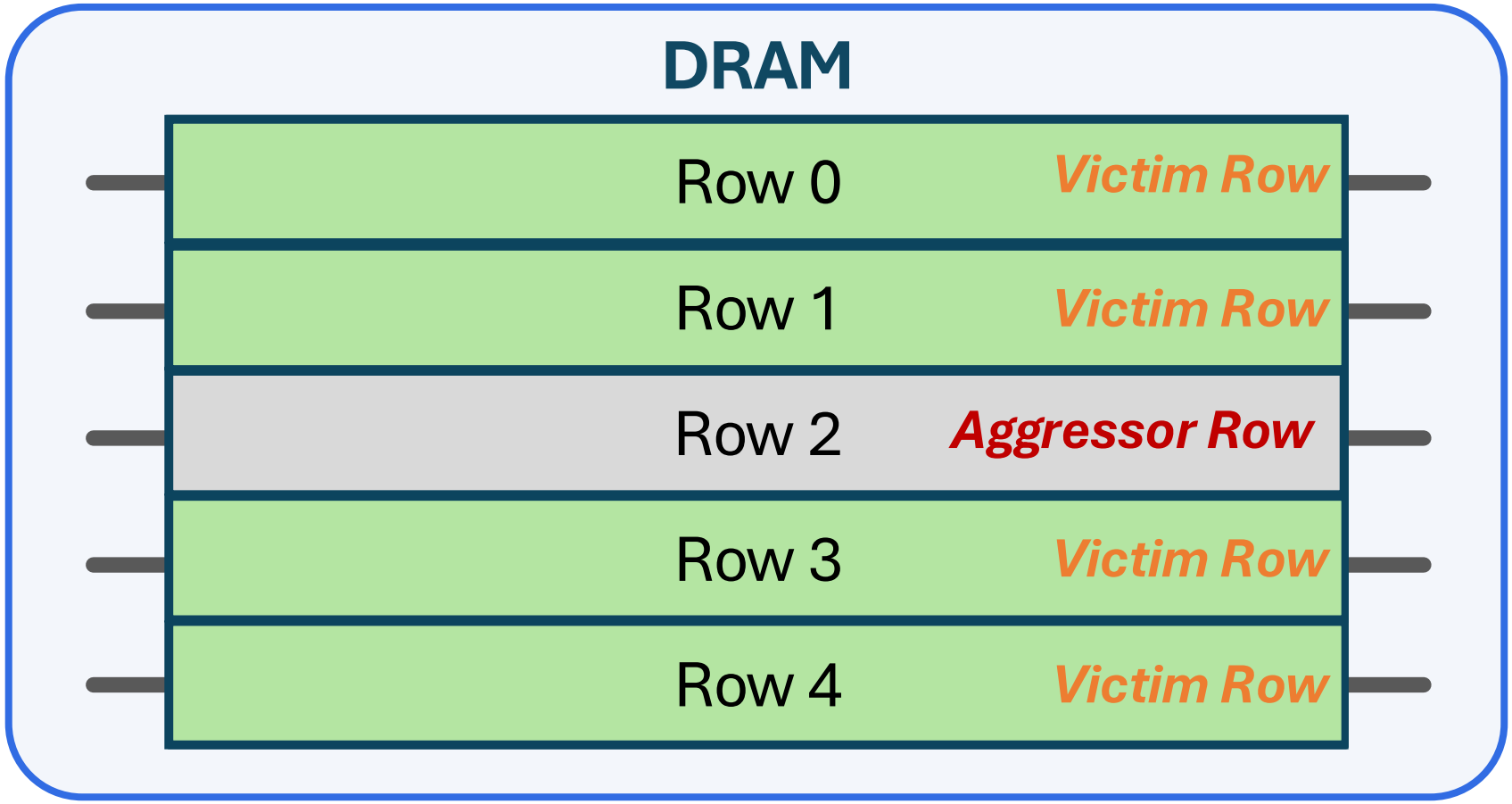
Many ways to prevent RowHammer via

RowHammer-preventive actions:

- **Preventive refresh**
- **Row migration**
- **Proactive throttling**
- **…**

State-of-the-art RowHammer mitigations adopt these two approaches

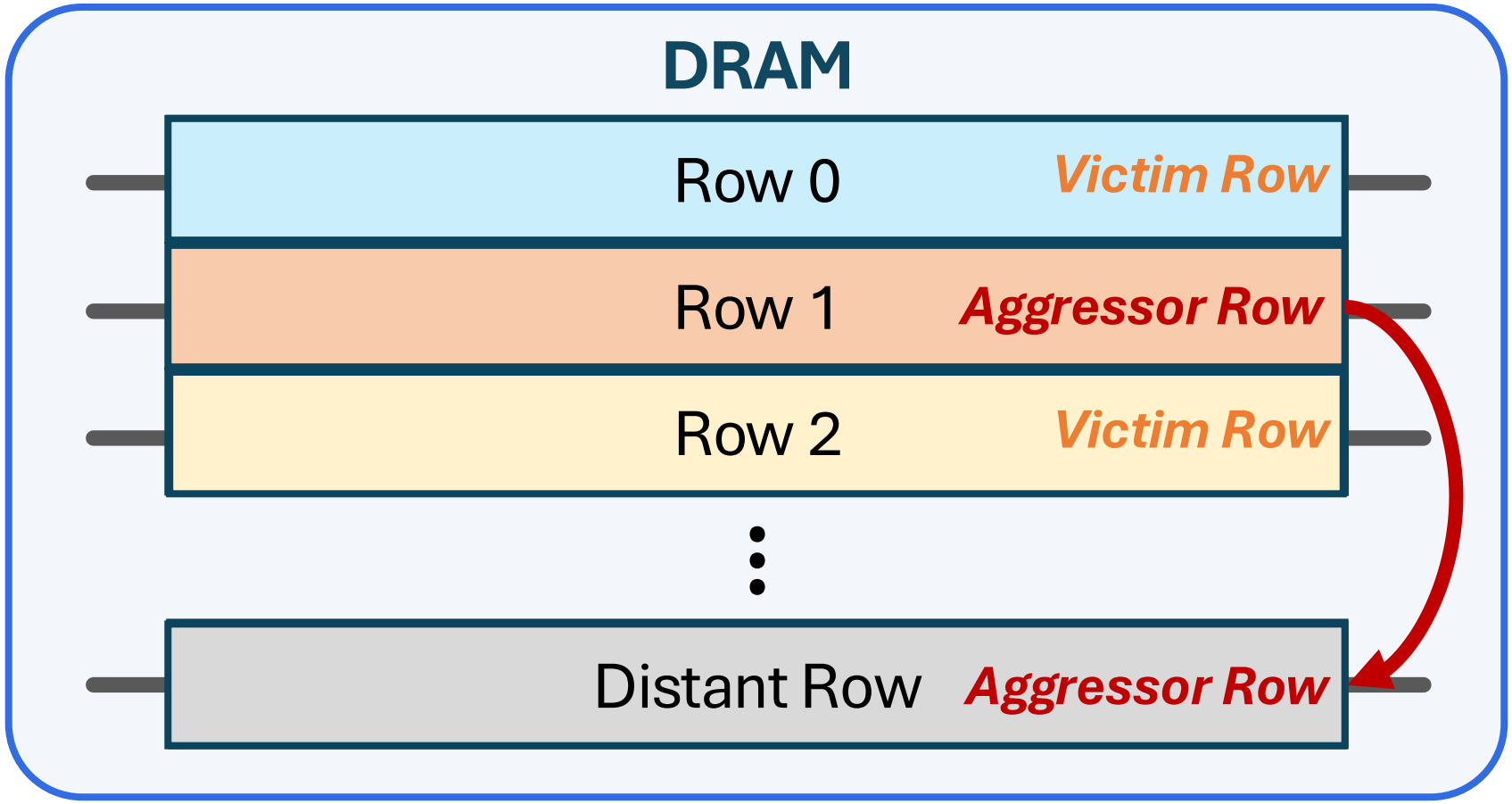
Preventive Refresh as a RowHammer-Preventive Action



Refreshing potential victim rows
mitigates RowHammer bitflips

[Kim+ ISCA'20]

Row Migration as a RowHammer-Preventive Action



Migrating potential aggressor rows

to a distant row **mitigates RowHammer bitflips**

Outline

Background

Motivation

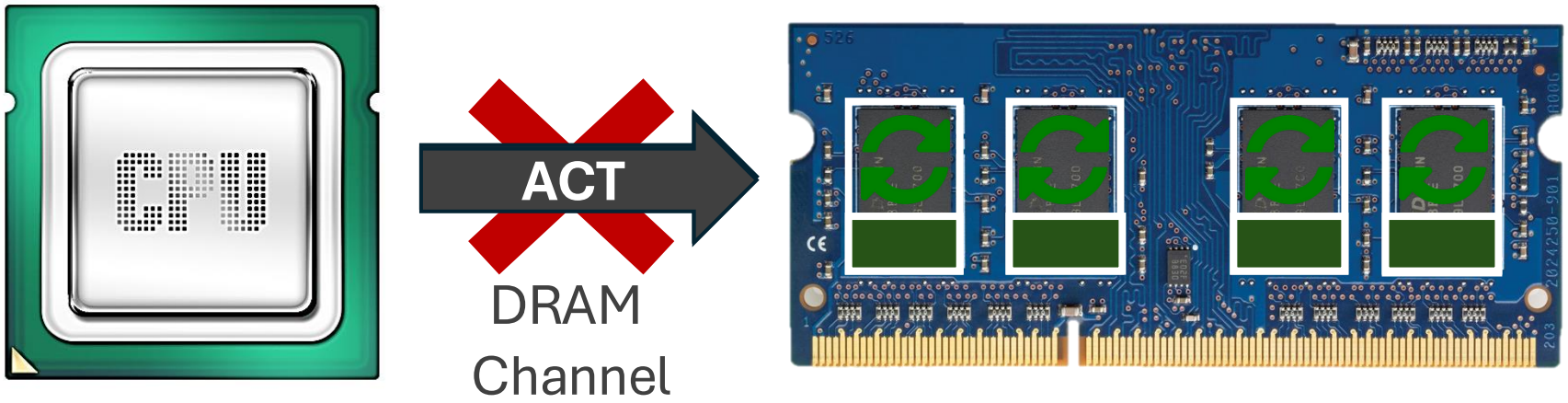
BreakHammer

Evaluation

Conclusion

Root Cause of Performance Overhead

RowHammer-preventive actions are
blocking and time consuming operations

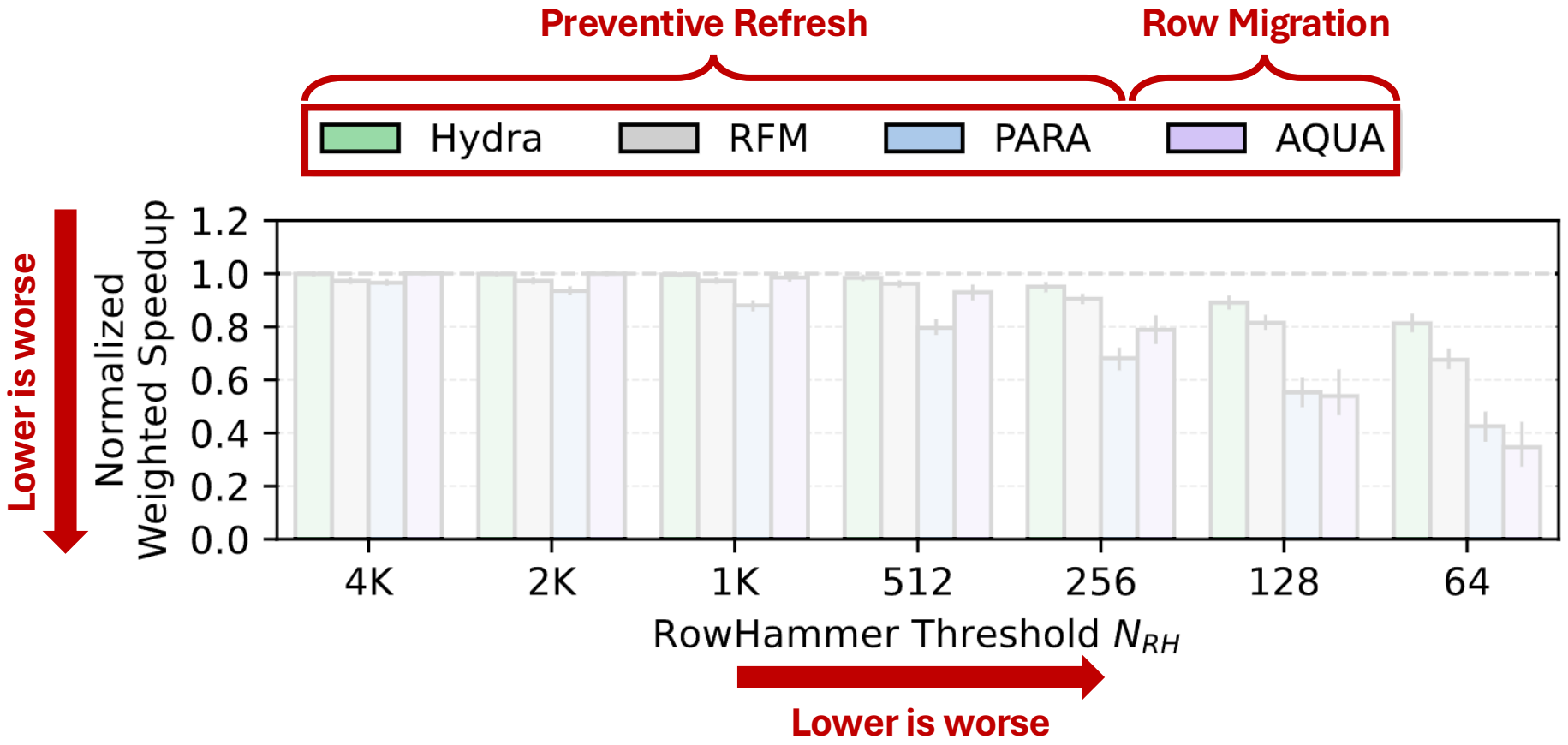


DRAM Module

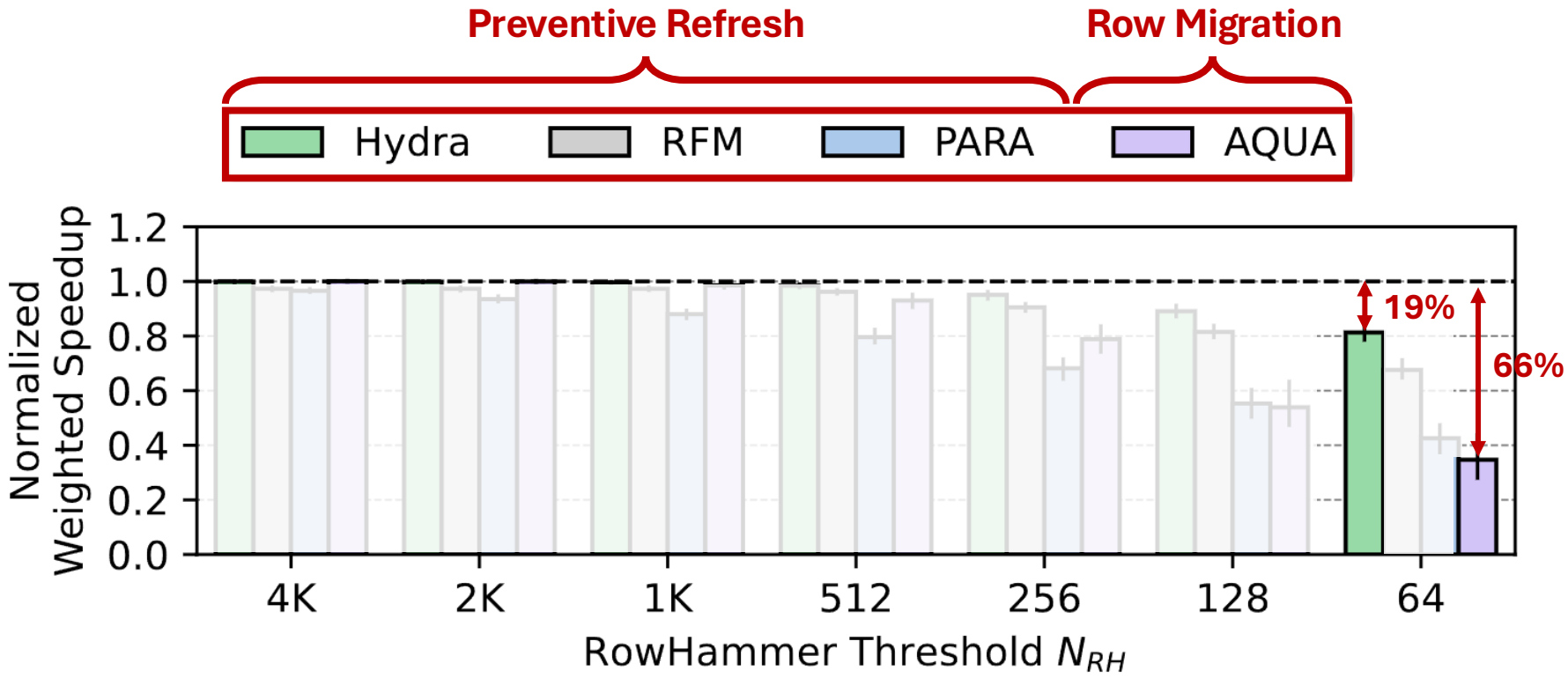
Memory controller **cannot access** a memory bank
undergoing a RowHammer-preventive action

Refreshing **KBs** of data can block access to **GBs** of data

RowHammer Mitigation Performance Overhead



RowHammer Mitigation Performance Overhead



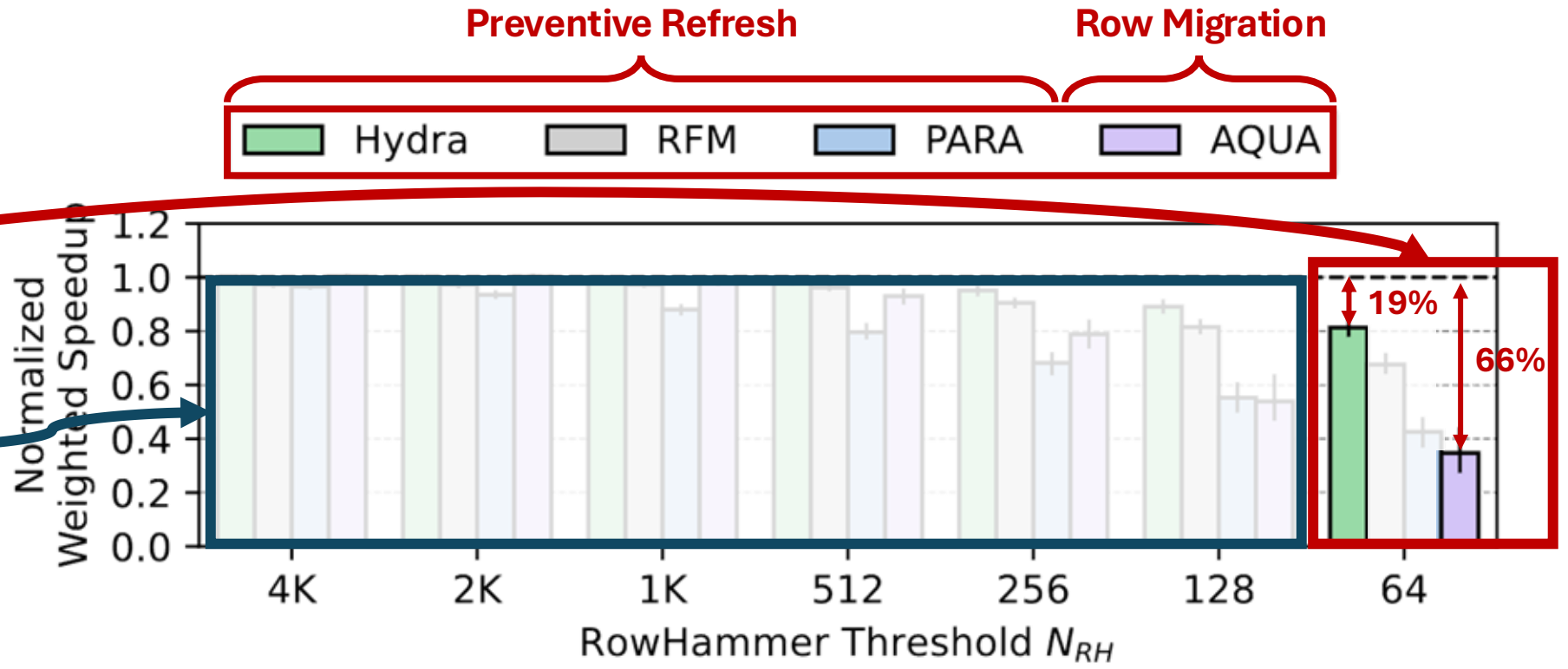
RowHammer mitigation mechanisms incur **increasingly large performance overhead** as the RowHammer threshold **decreases**

Presenting a Plot

- Explain the plot before making observations
  - What do the axes show?
  - Which way is better?
  - What are the colors?
- Highlight observations

Transparent boxes are your friends

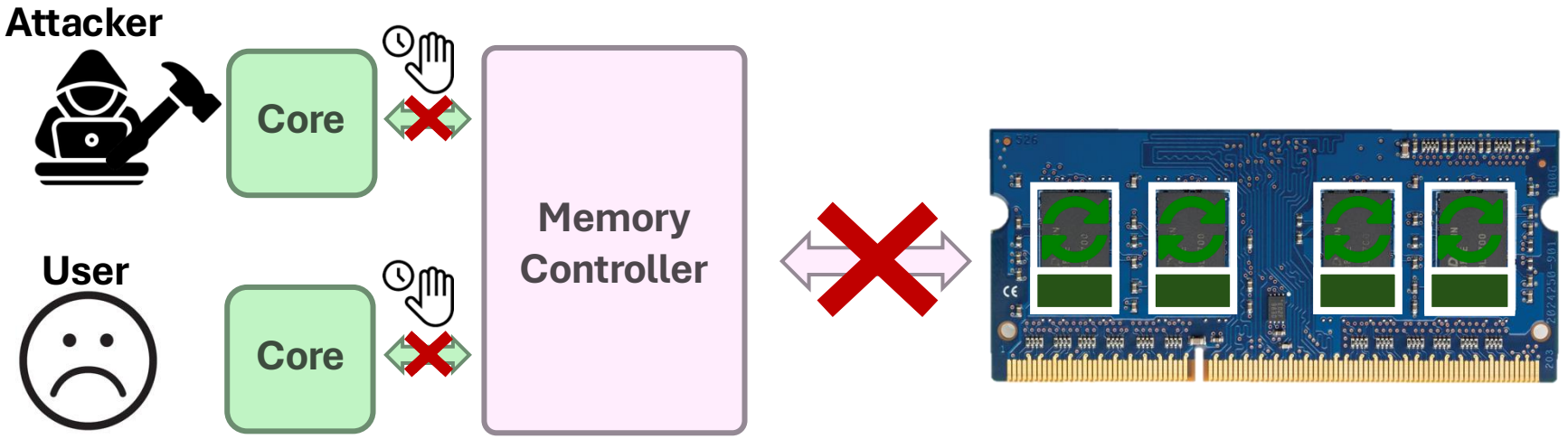
- Give your key takeaway



RowHammer mitigation mechanisms incur **increasingly large performance overhead** as the RowHammer threshold **decreases**

Memory Performance Attack

Attacker can trigger **many** preventive actions
to **block access** to main memory



Preventive actions can be exploited
to **reduce DRAM bandwidth availability**

Problem & Goal

Problem

Operations that prevent RowHammer lead to
DRAM bandwidth availability issues
as they can frequently **block access to memory**

Goal

Reduce the performance overhead
of RowHammer mitigation mechanisms
by reducing the number of RowHammer-preventive actions
without compromising system robustness

Outline

Background

Motivation

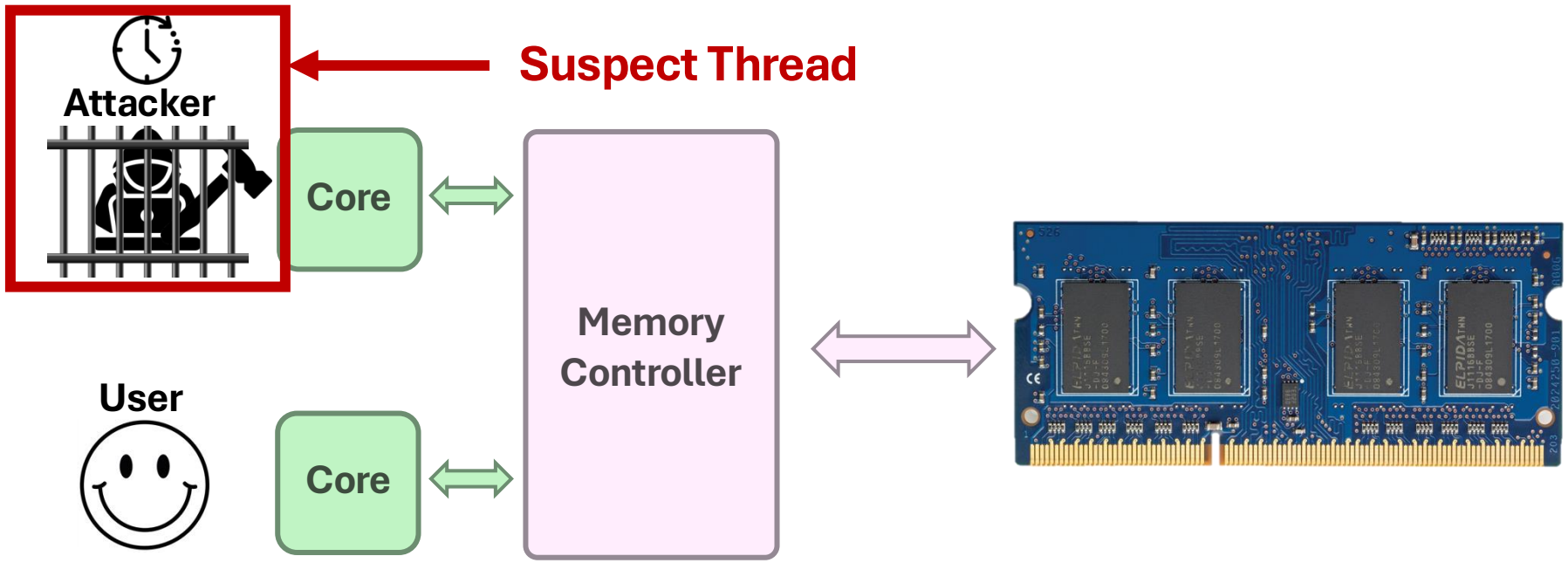
BreakHammer

Evaluation

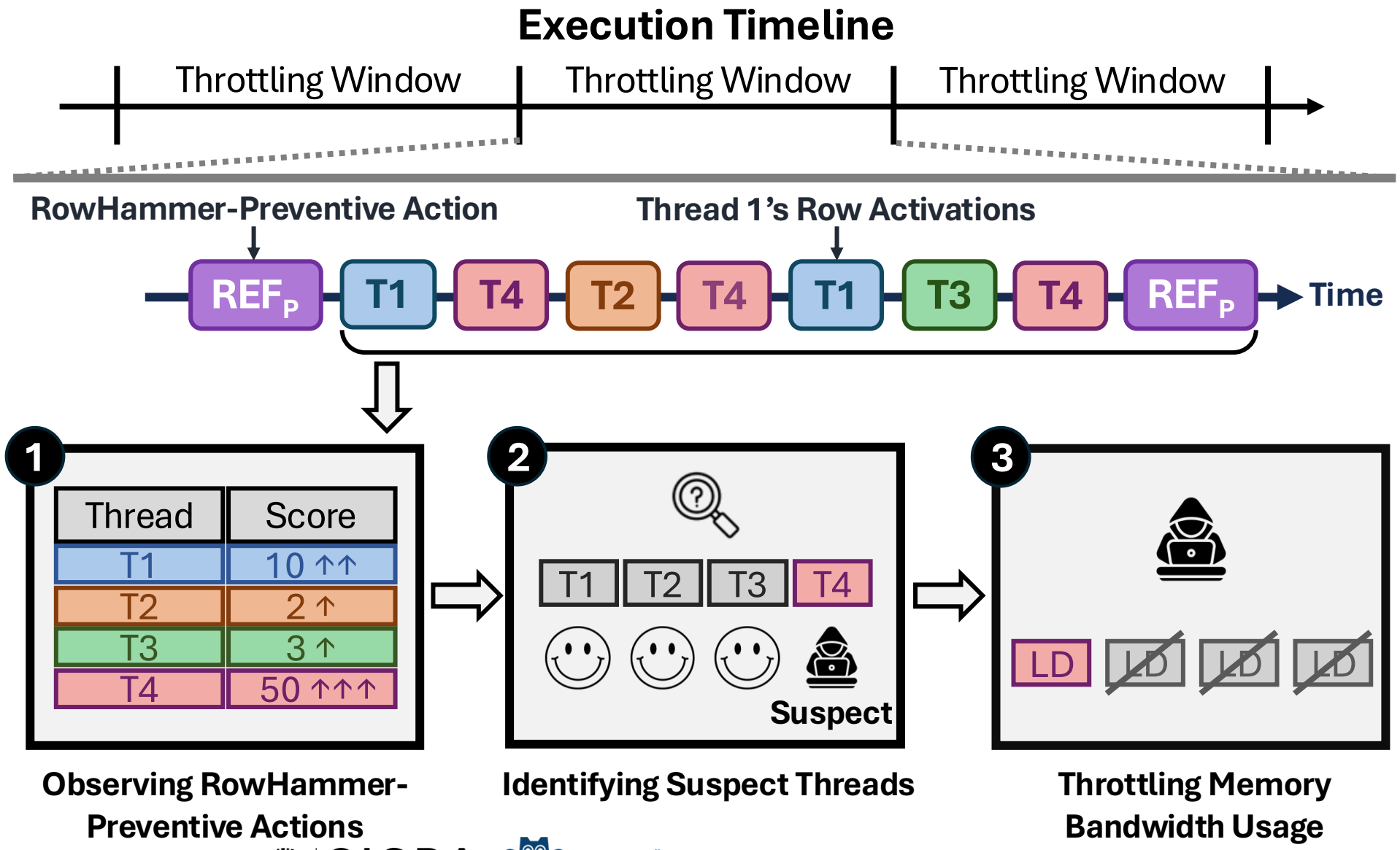
Conclusion

Key Idea

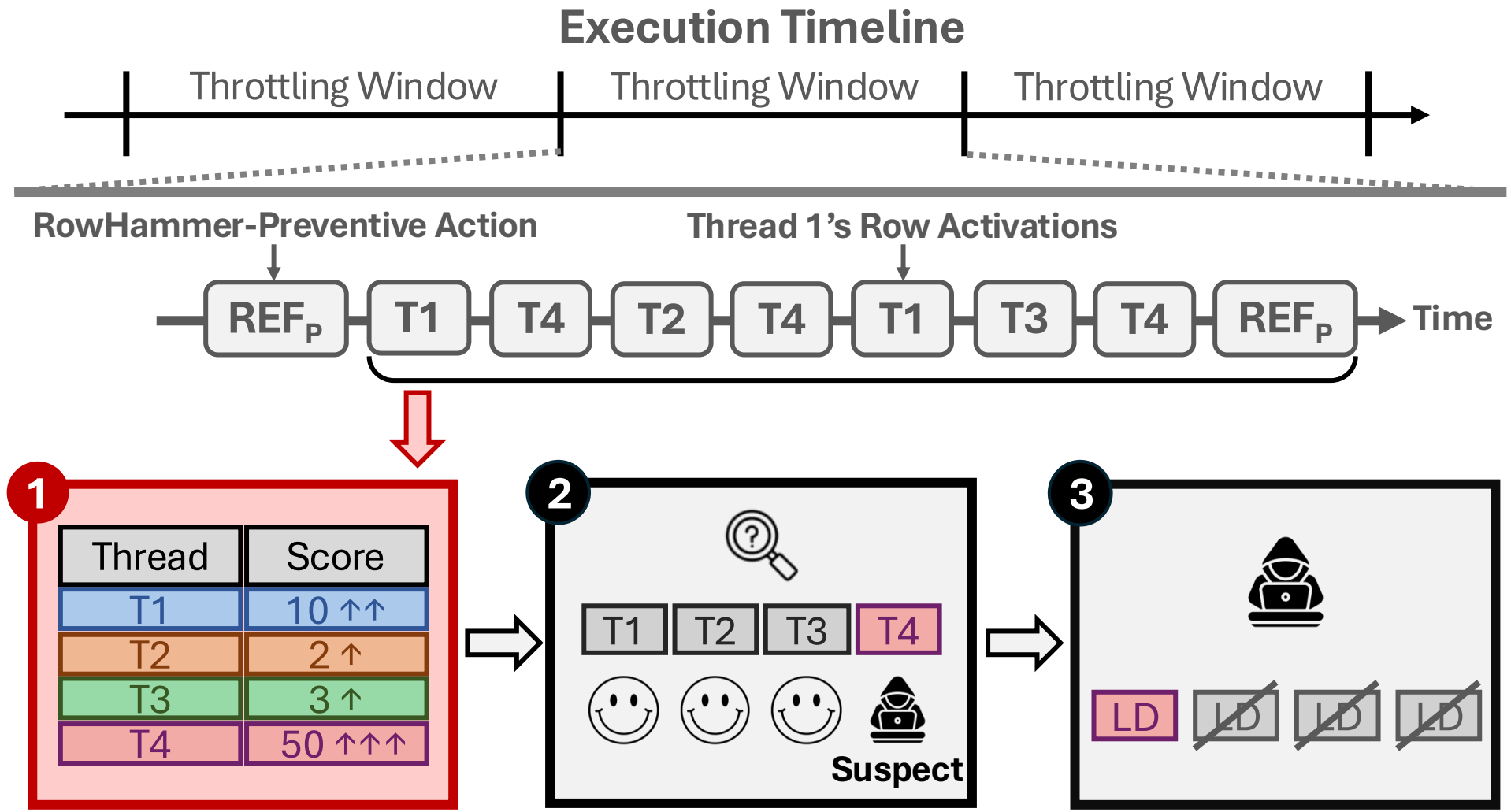
Detect and **slow down** the memory accesses of threads that trigger **many** RowHammer-preventive actions



BreakHammer: Overview

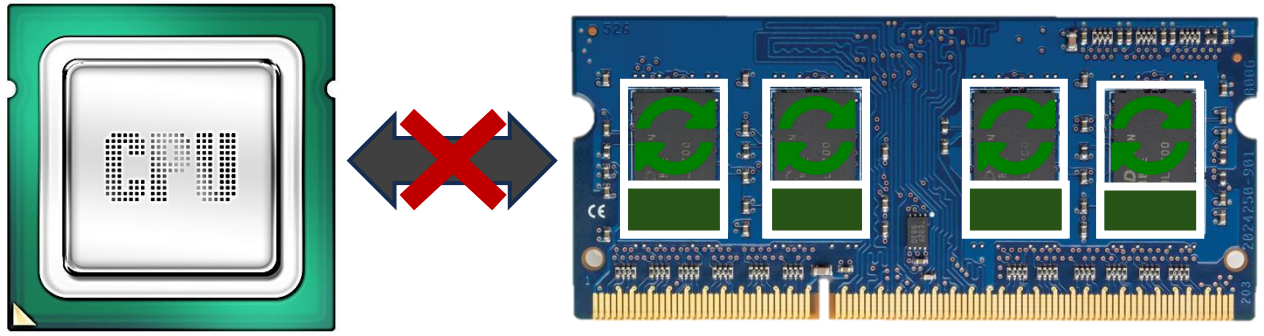


BreakHammer: Overview



Observing RowHammer-Preventive Actions

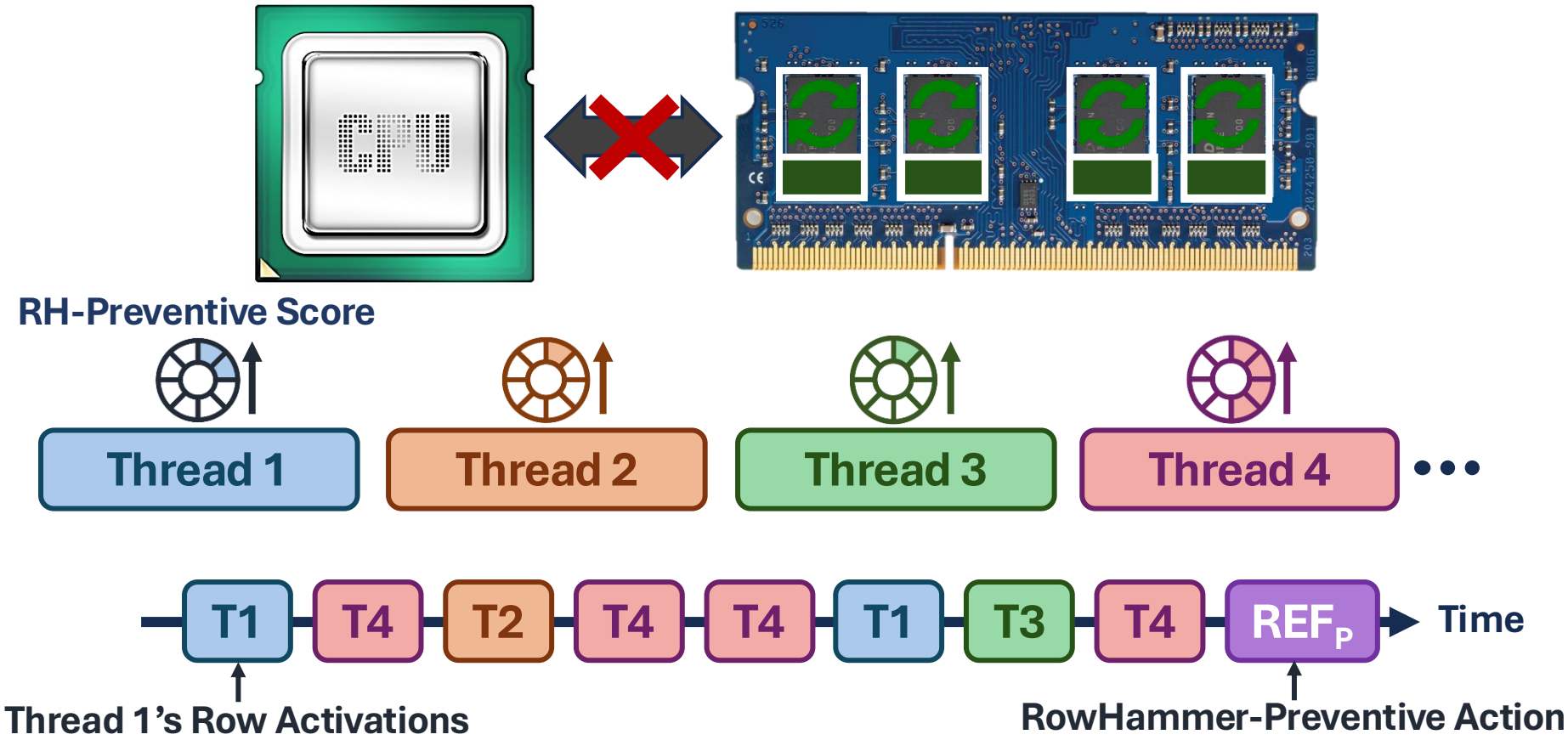
BreakHammer tracks the number
of RowHammer-preventive actions each thread triggers



RowHammer-preventive score counter

Score Attribution Method

A **RowHammer-preventive action** is generally caused by
a **stream of memory requests** from **many hardware threads**



Integration Showcase

1) Probabilistic Row Activation (PARA)

2) Per Row Activation Counting (PRAC)

Integration Showcase

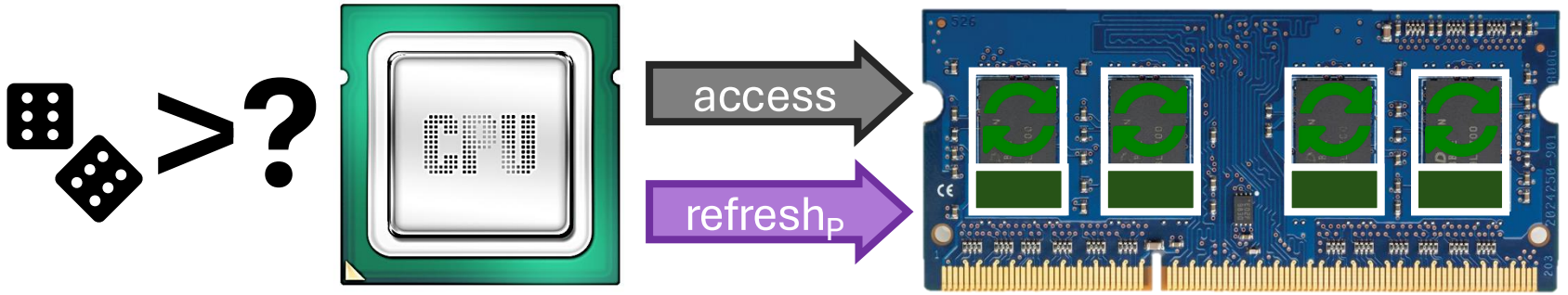
1) Probabilistic Row Activation (PARA)

2) Per Row Activation Counting (PRAC)

Integration Showcase with PARA (I)

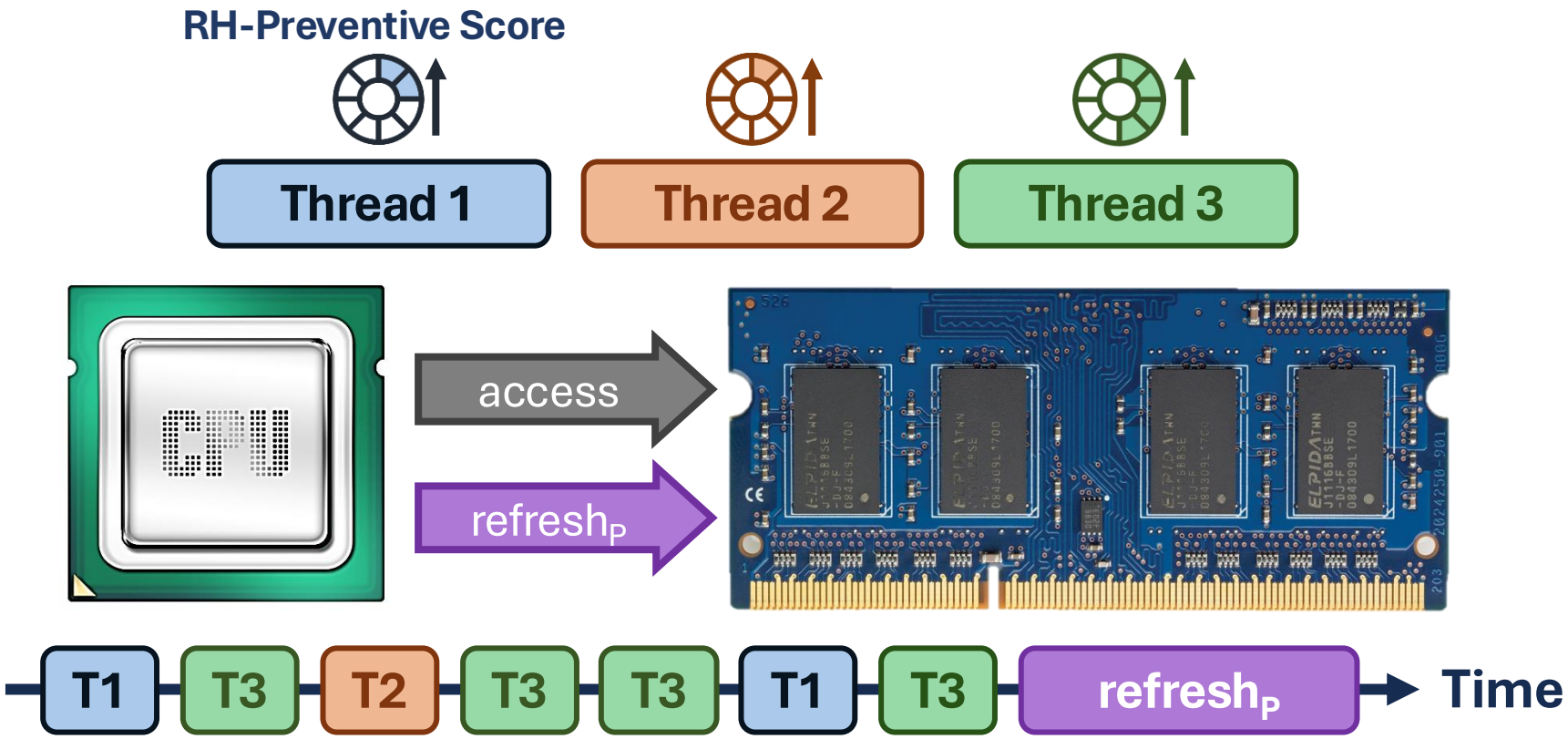
- **BreakHammer** cooperates with existing RowHammer solutions

- Probabilistic Row Activation (PARA) [Kim+, ISCA 2024]:
  - **Generates** a random number
  - Compares the number **with a threshold**
  - If the random number exceeds the threshold **performs a preventive refresh**



Integration Showcase with PARA (II)

- Probabilistic Row Activation + BreakHammer (PARA+BH):
  - **Track row activation** count of each thread **between preventive refreshes**
  - Increment each thread's score proportionally to its activations



Integration Showcase

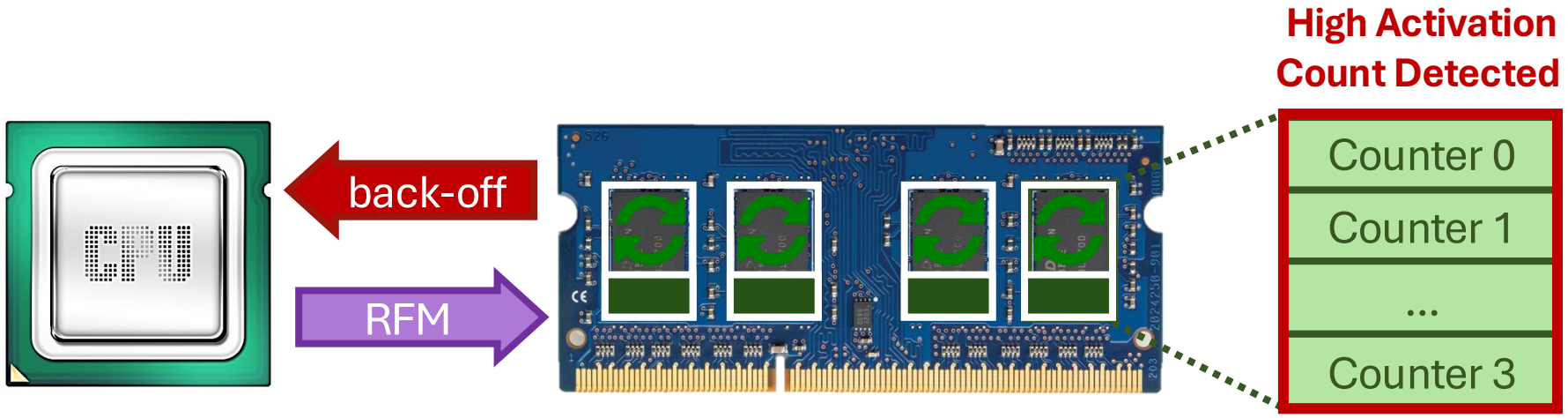
1) Probabilistic Row Activation (PARA)

2) Per Row Activation Counting (PRAC)

Integration Showcase with PRAC (I)

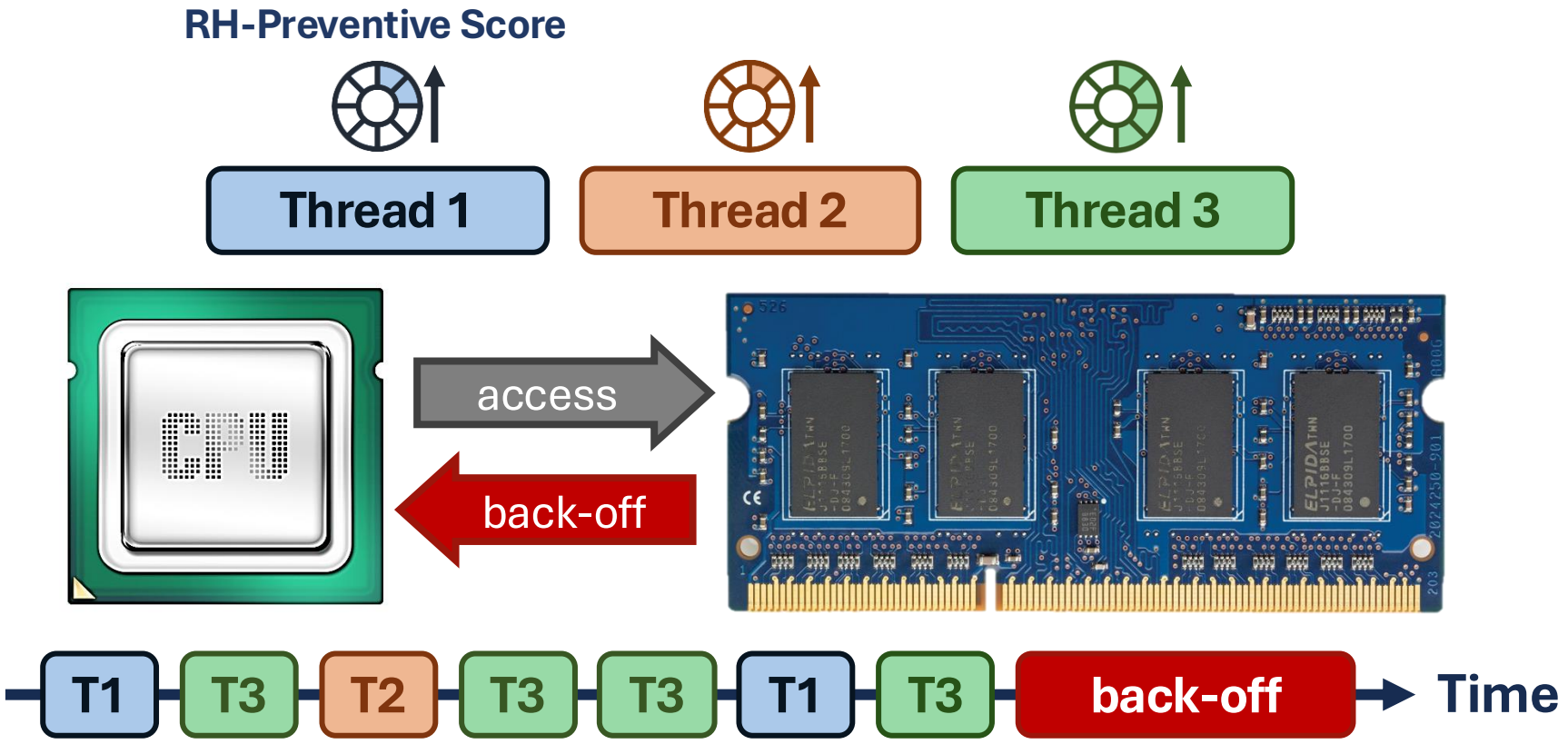
- **BreakHammer** cooperates with existing RowHammer solutions

- Per Row Activation Counting (PRAC) [JEDEC, 2024]:
  - **DRAM** maintains an **activation counter for each DRAM row**
  - **DRAM** requests time by triggering a **back-off**
  - **Memory controller provides time** for in-DRAM preventive refreshes



Integration Showcase with PRAC (II)

- Per Row Activation Counting + BreakHammer (PRAC+BH):
  - **Track row activation** count of each thread **between back-offs**
  - Increment each thread's score proportionally to its activations



Integration with Other Mechanisms

- We integrate **BreakHammer** with **eight** RowHammer solutions:
  - **PARA** [Kim+, ISCA 2014]
  - **Graphene** [Park+, MICRO 2020]
  - **Hydra** [Qureshi+, ISCA 2022]
  - **TWiCe** [Lee+, ISCA 2019]
  - **AQUA** [Saxena+, MICRO 2022]
  - **REGA** [Marazzi+, S&P 2023]
  - **RFM** [JEDEC 2020]
  - **PRAC** [JEDEC 2024]

BreakHammer: Enhancing RowHammer Mitigations by Carefully Throttling Suspect Threads

Oğuzhan Canpolat[§†] A. Giray Yağlıkçı[§] Ataberk Olgun[§] Ismail Emir Yuksel[§]
Yahya Can Tuğrul[§†] Konstantinos Kanellopoulos[†] Oğuz Ergin[‡§†] Onur Mutlu[§]
[§]*ETH Zürich* [†]*TOBB University of Economics and Technology* [‡]*University of Sharjah*

RowHammer is a major read disturbance mechanism in DRAM where repeatedly accessing (hammering) a row of DRAM cells (DRAM row) induces bitflips in other physically nearby DRAM rows. RowHammer solutions perform preventive actions (e.g.,

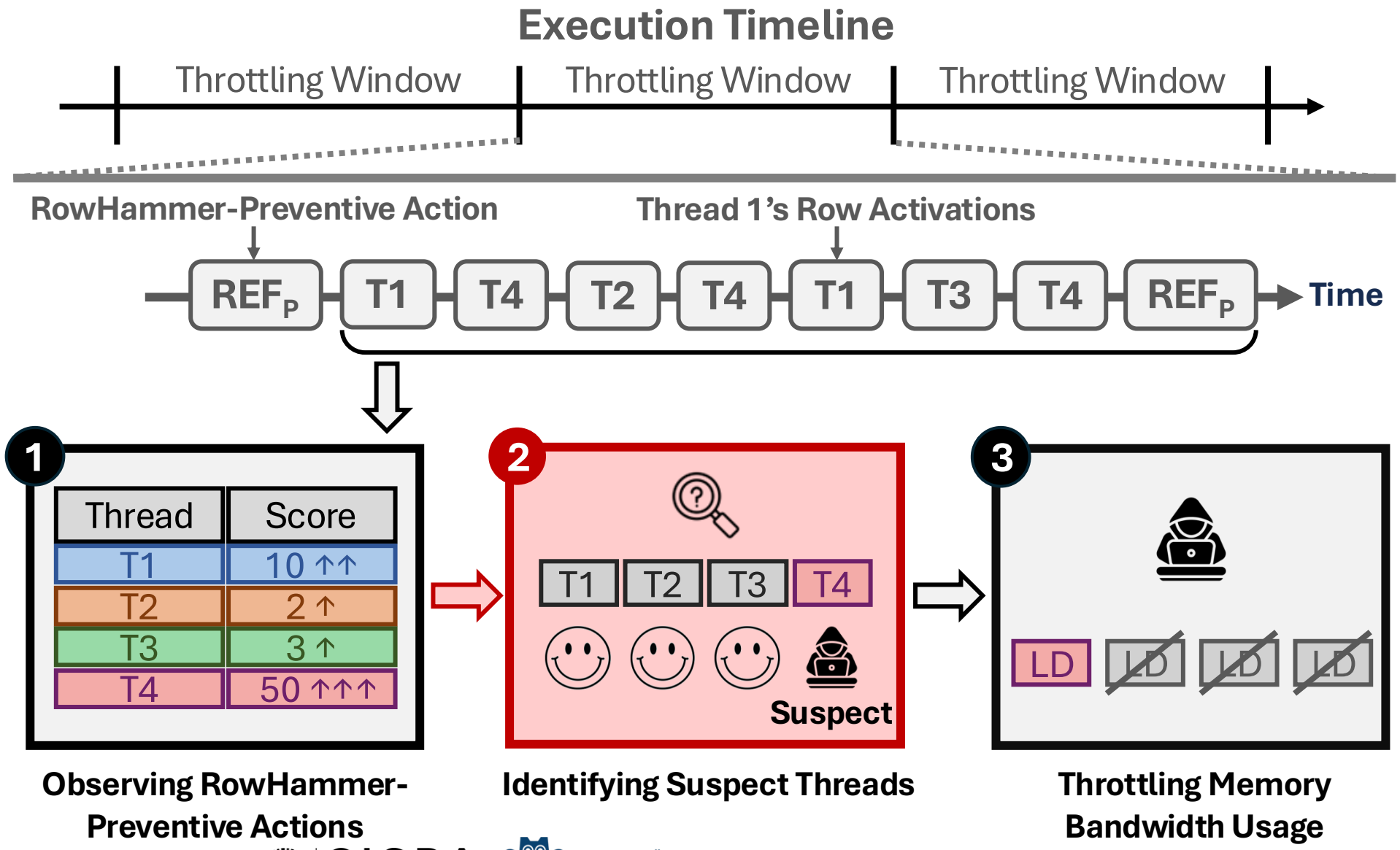
can experience bitflips when a nearby DRAM row (i.e., aggressor row) is repeatedly opened (i.e., hammered) [2–70].

Many prior works demonstrate attacks on a wide range of systems where they exploit read disturbance to escalate

https://arxiv.org/abs/2404.13477 https://github.com/CMU-SAFARI/BreakHammer

S4L ROOTSEC CISPA HELMHOLTZ CENTER FOR INFORMATION SECURITY UNIVERSITÄT DES SAARLANDES

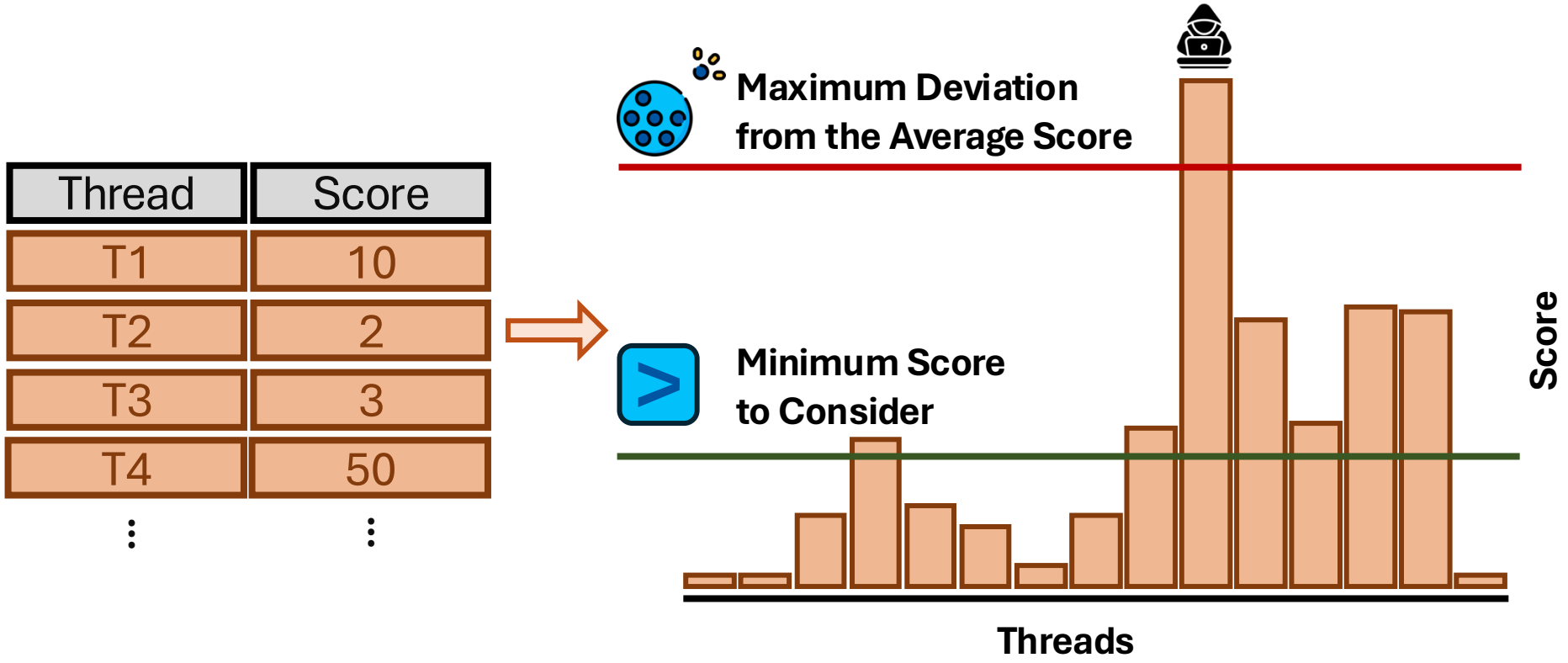
BreakHammer: Overview



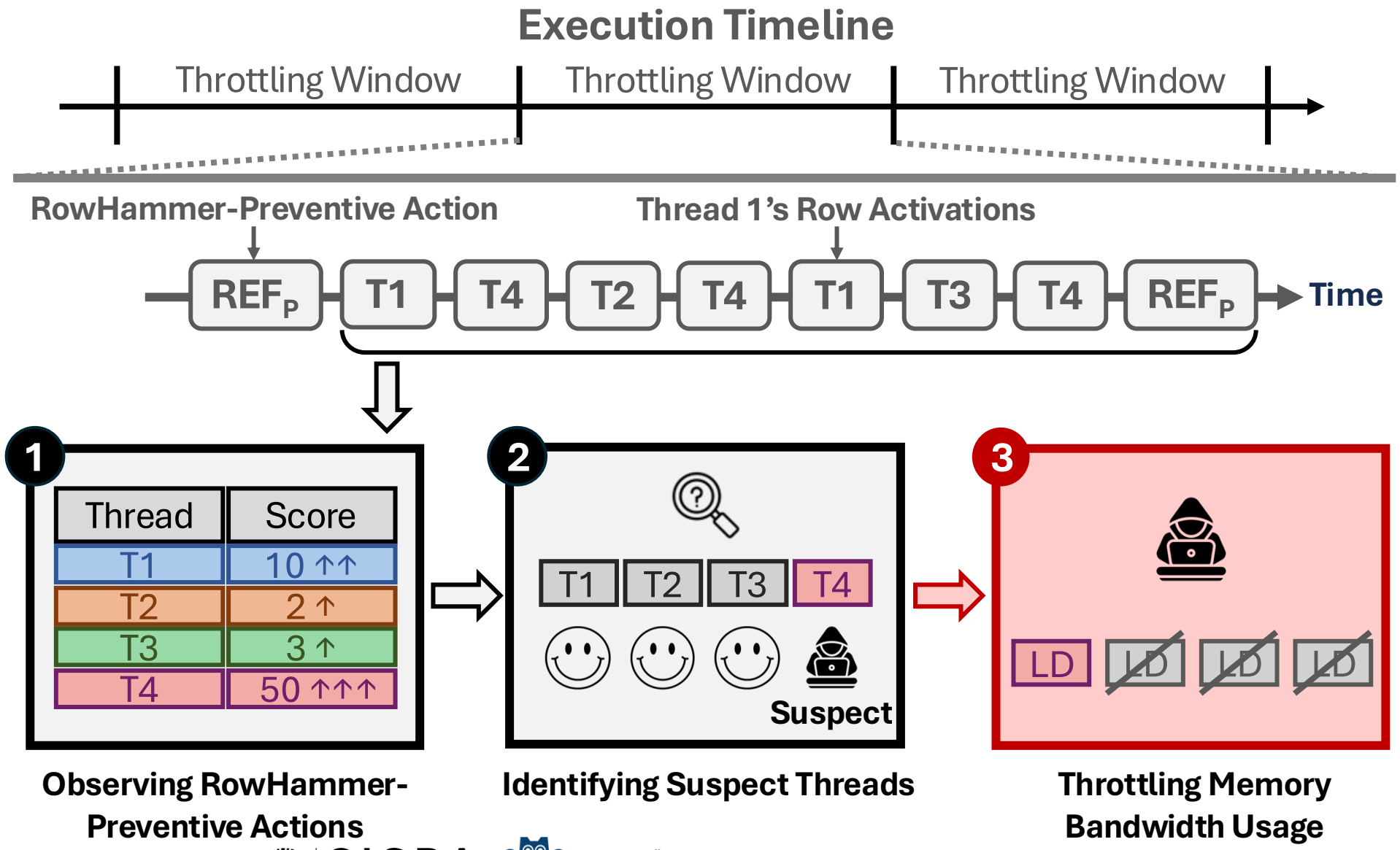
Identifying Suspect Threads: An Example

BreakHammer detects threads that trigger **too many** RowHammer-preventive actions

Thread	Score
T1	10
T2	2
T3	3
T4	50
⋮	⋮

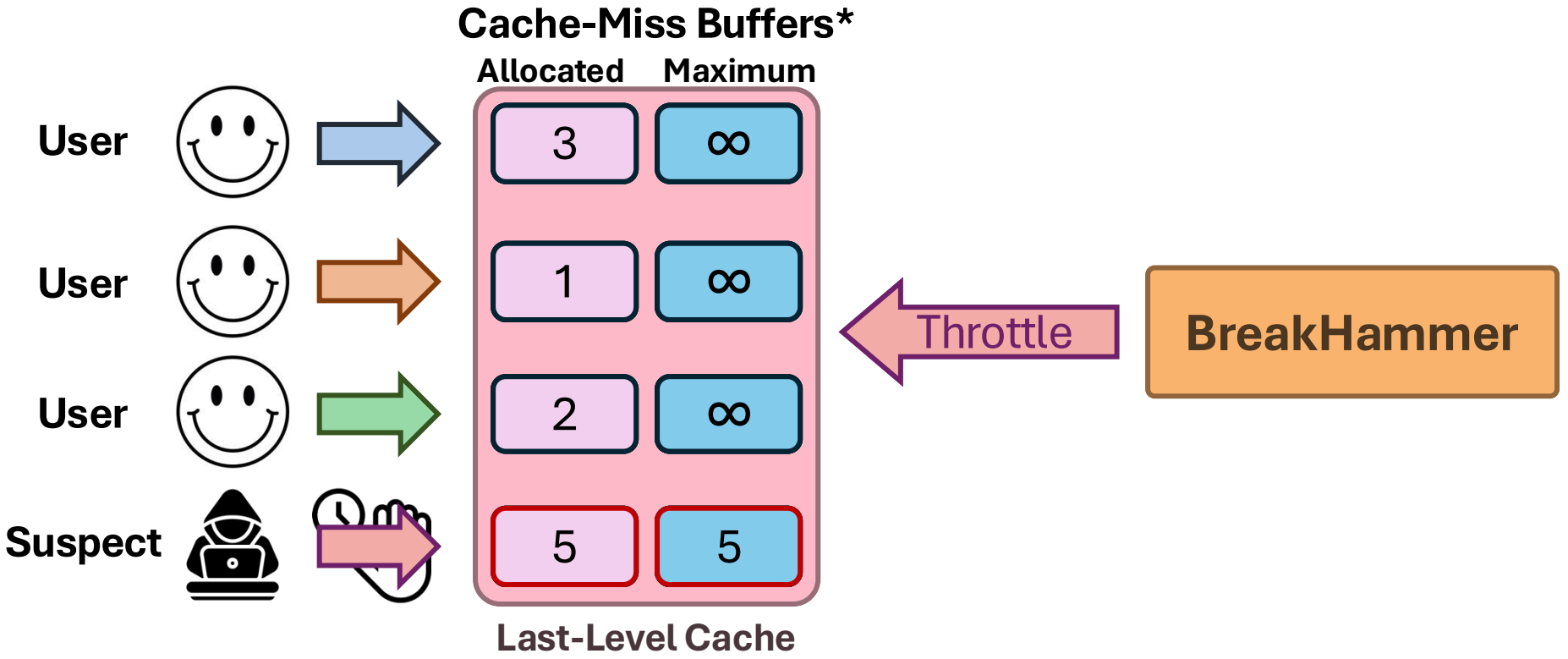


BreakHammer: Overview



Throttling Memory Bandwidth Usage of Suspect Threads

BreakHammer reduces the **memory bandwidth usage** of each **suspect thread**

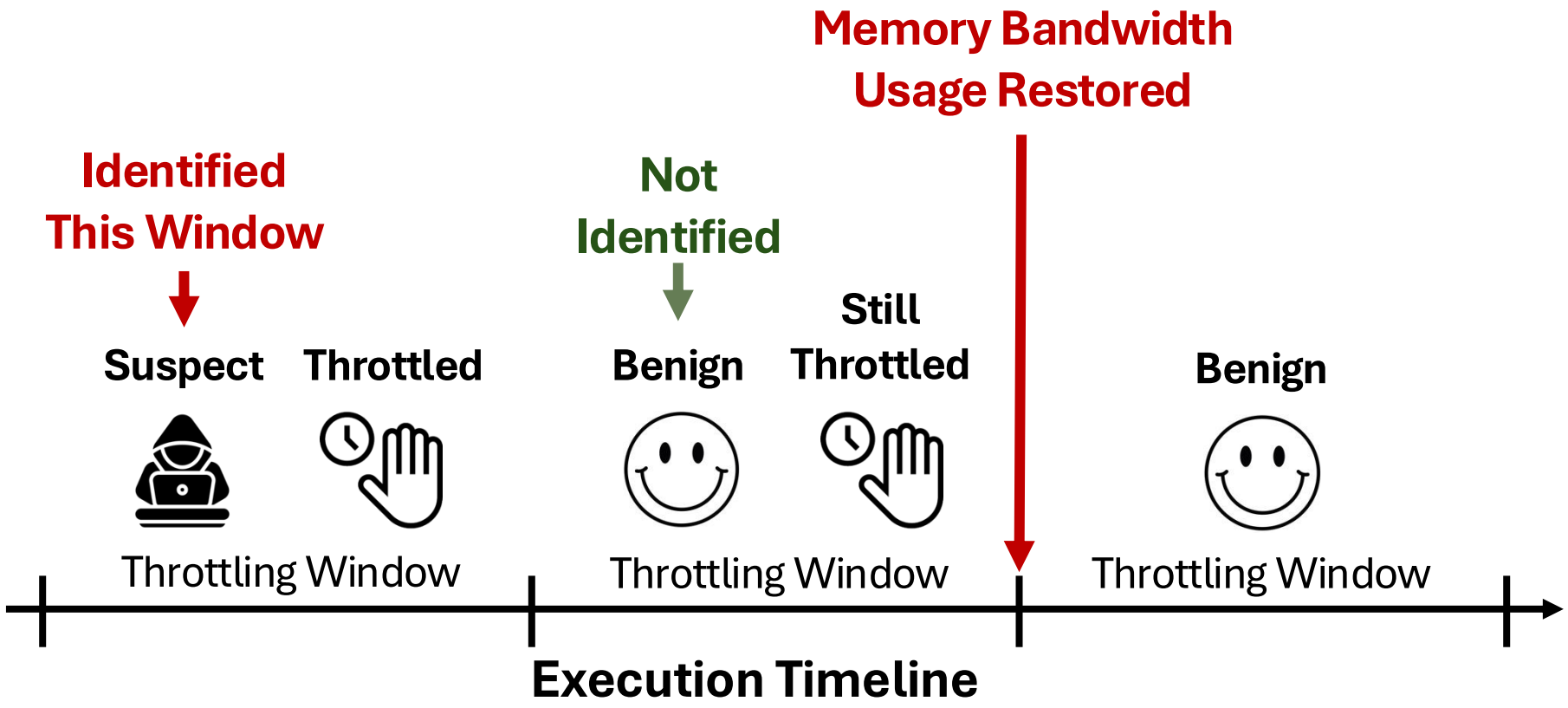


[*] also known as miss status holding registers (MSHRs)

Restoring Memory Bandwidth of Suspect Threads

BreakHammer

restores the **memory bandwidth usage** of a **suspect thread**
if the thread stays benign for the full duration of a throttling window



Outline

Background

Motivation

BreakHammer

Evaluation

Conclusion

Evaluation Methodology

- **Performance and energy consumption evaluation**
  using Ramulator 2.0 [Luo+, CAL 2023] and DRAMPower [Chandrasekar+, DATE 2013]

Processor	4 cores, 4.2GHz clock frequency, 4-wide issue, 128-entry instruction window
DRAM	DDR5, 1 channel, 2 rank/channel, 8 bank groups, 4 banks/bank group, 64K rows/bank
Memory Ctrl.	64-entry read and write requests queues, Scheduling policy: FR-FCFS with a column cap of 4
Last-Level Cache	8 MiB (4-core)

- **Comparison Points:** Integrated with 8 state-of-the-art RowHammer mitigations
  - **PARA** [Kim+, ISCA 2014]
  - **Graphene** [Park+, MICRO 2020]
  - **Hydra** [Qureshi+, ISCA 2022]
  - **TWiCe** [Lee+, ISCA 2019]
  - **AQUA** [Saxena+, MICRO 2022]
  - **REGA** [Marazzi+, S&P 2023]
  - **RFM** [JEDEC 2020]
  - **PRAC** [JEDEC 2024]

- **Workloads:** 4-core workload mixes from SPEC CPU2006/2017, TPC, MediaBench, YCSB
  → 90 mixes with one attacker and → 90 mixes with all benign

Evaluation Results

1) Under Attack

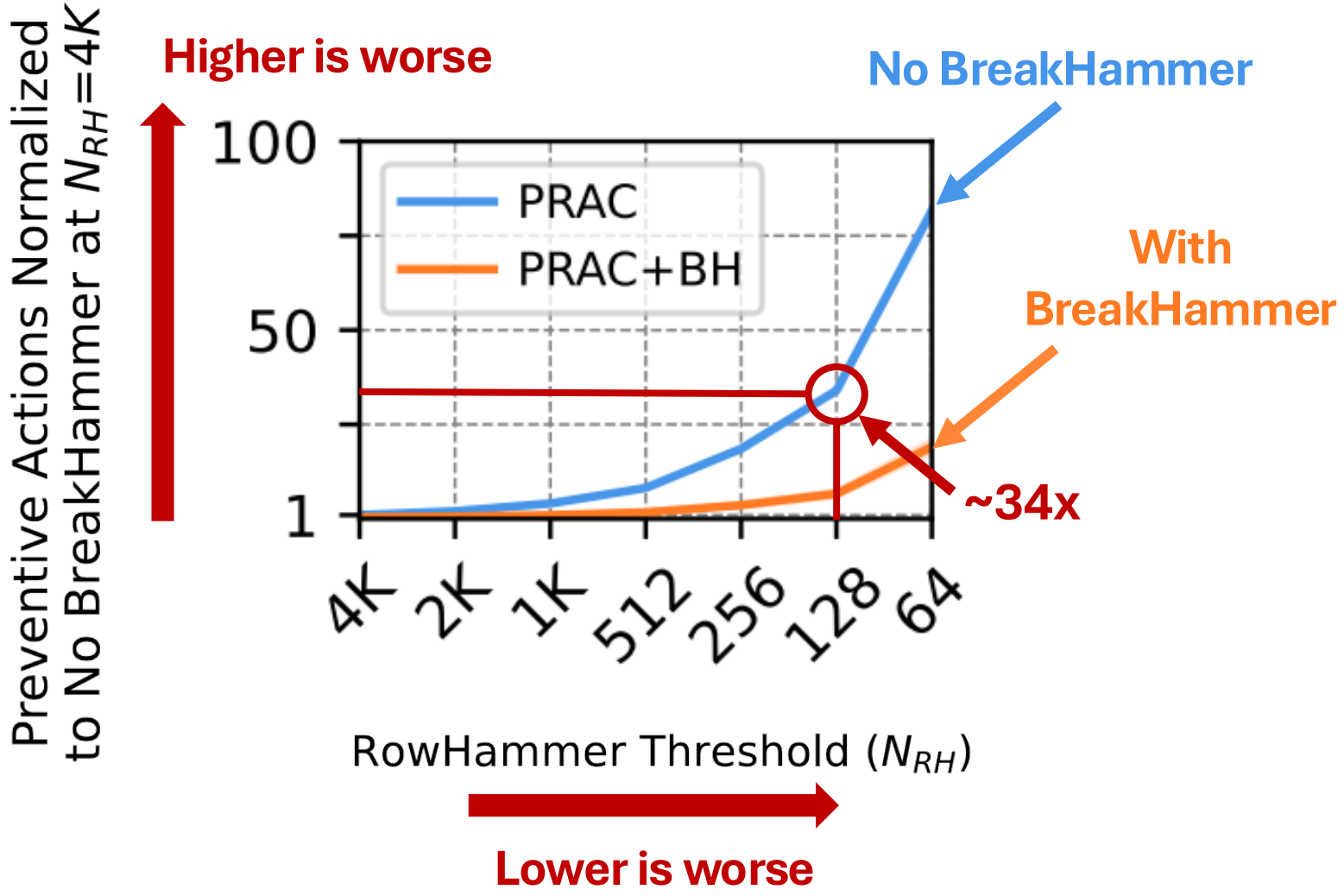
2) No Attack

Evaluation Results

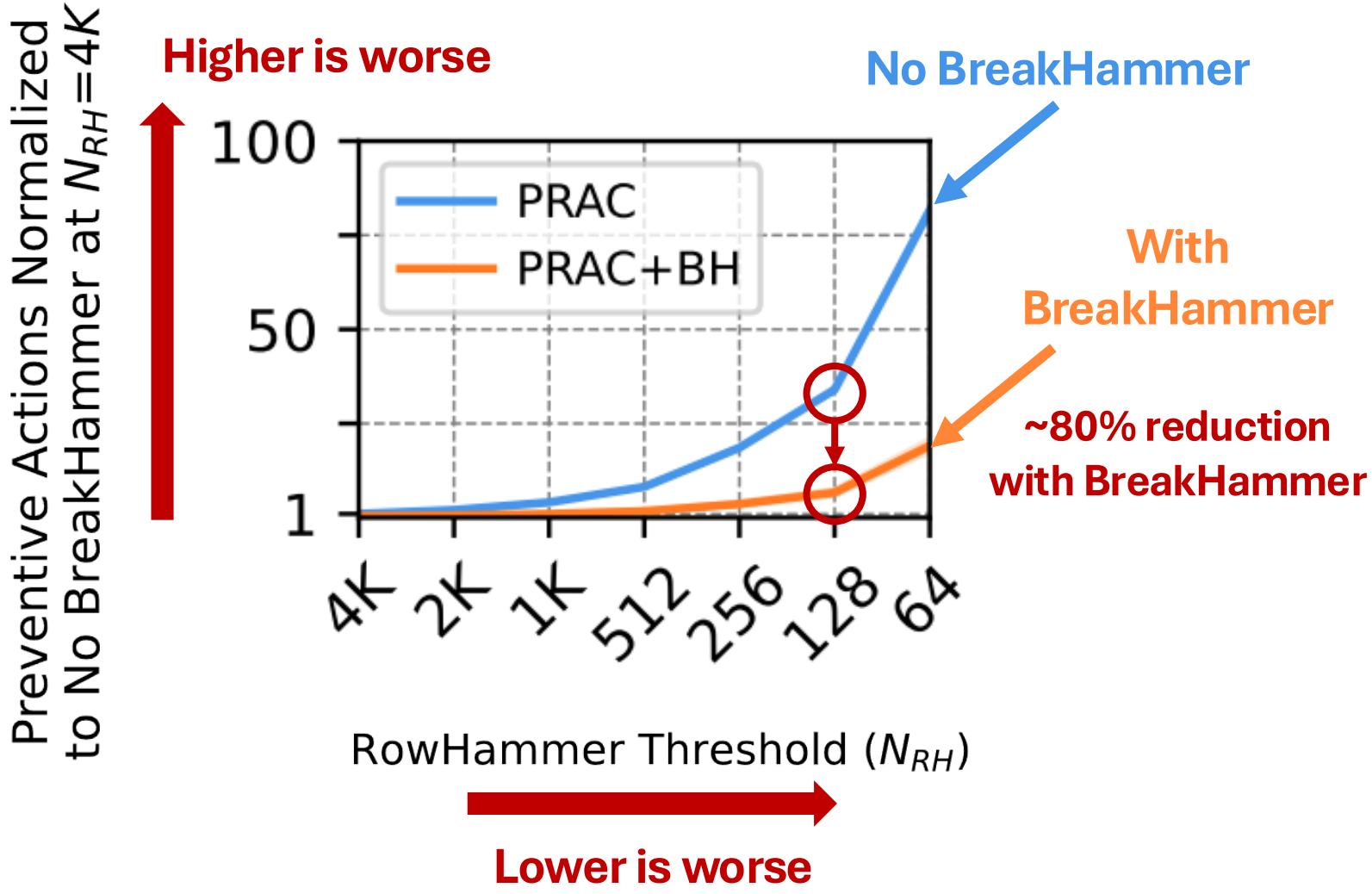
1) Under Attack

2) No Attack

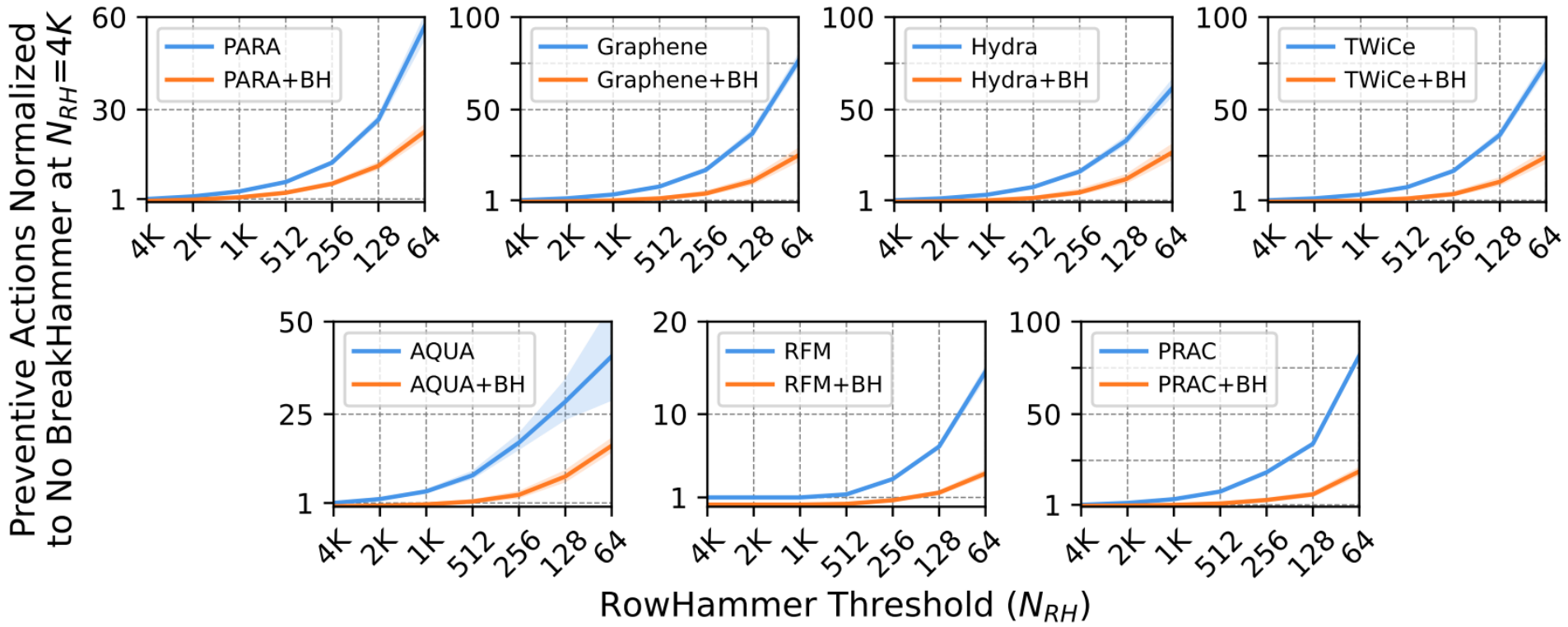
Preventive Action Count and Its Scaling



Preventive Action Count and Its Scaling

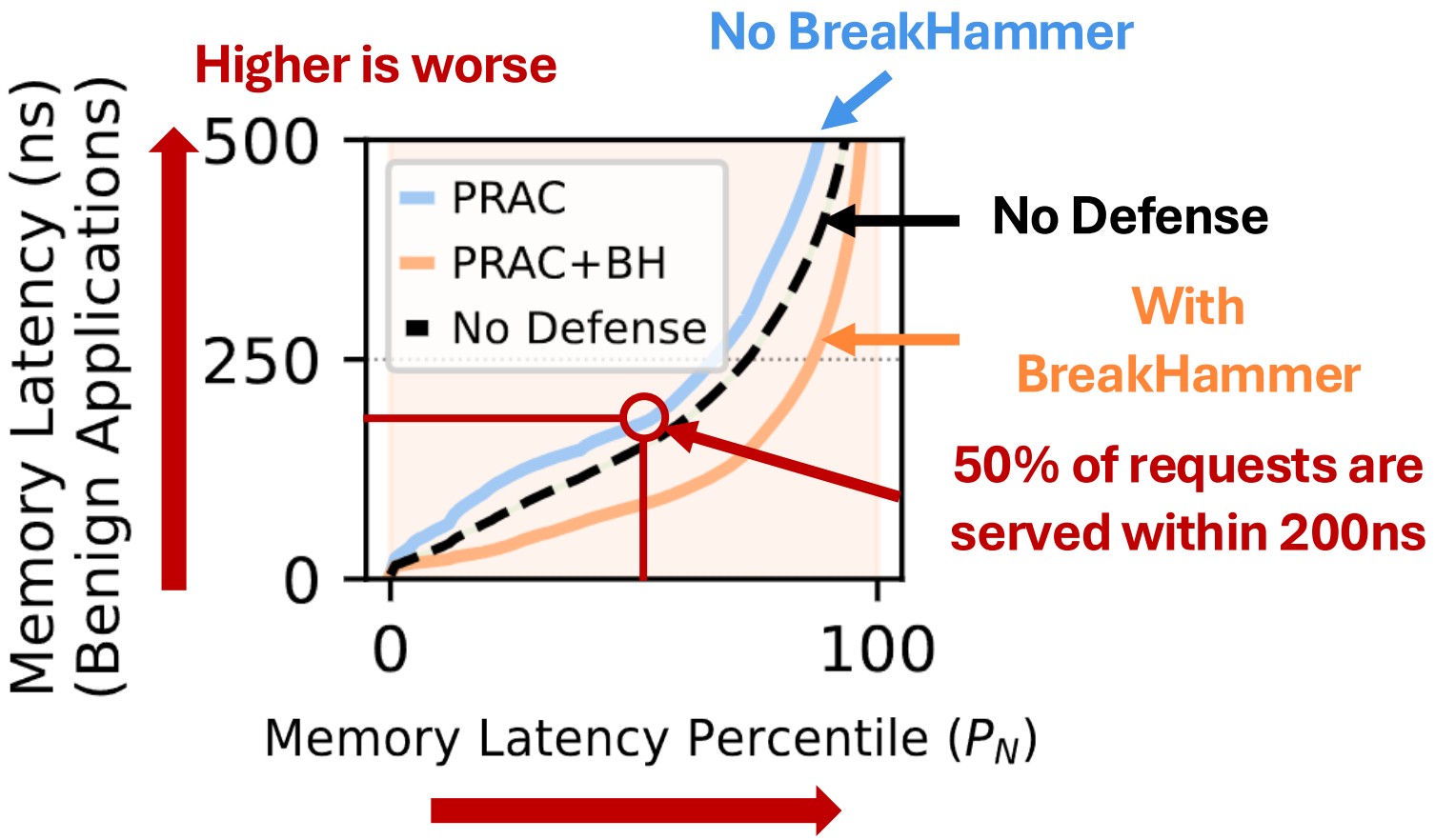


Preventive Action Count and Its Scaling

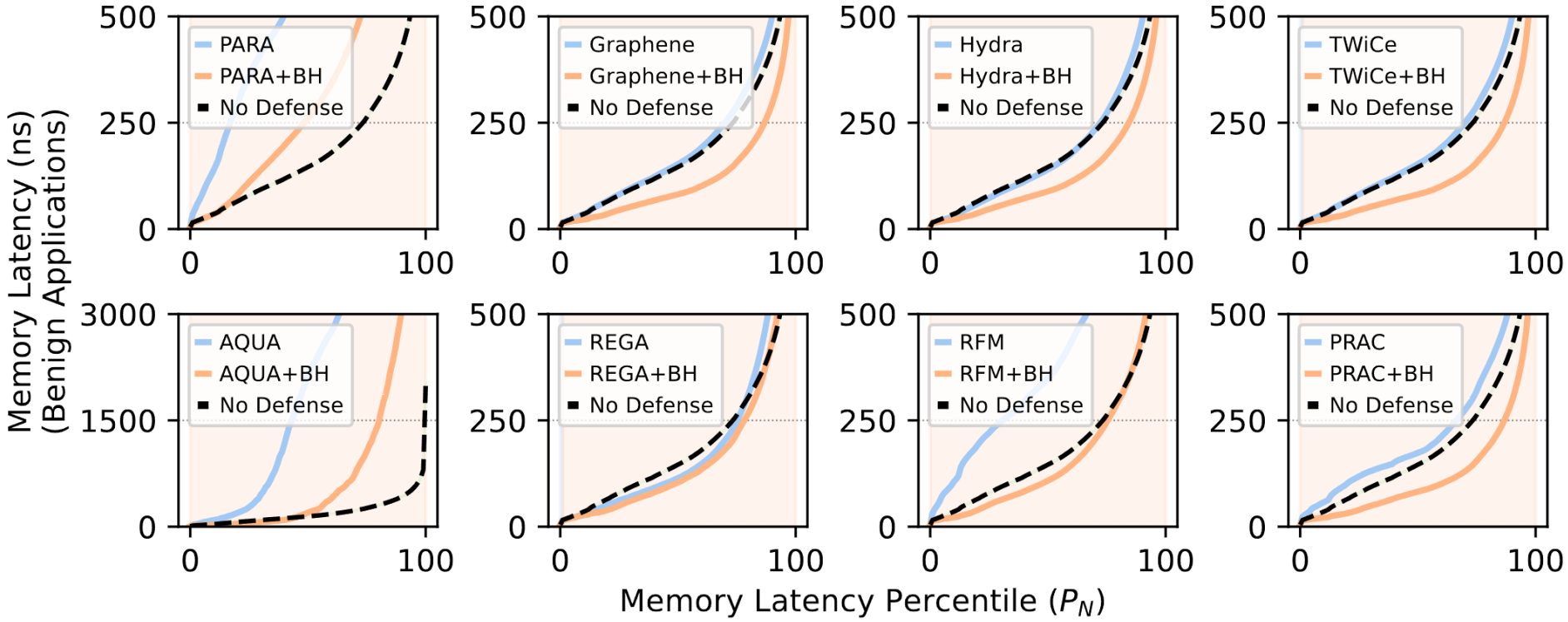


BreakHammer significantly reduces (72% on average)
the number of preventive actions performed across all mechanisms

Memory Latency Impact at N_{RH}=64

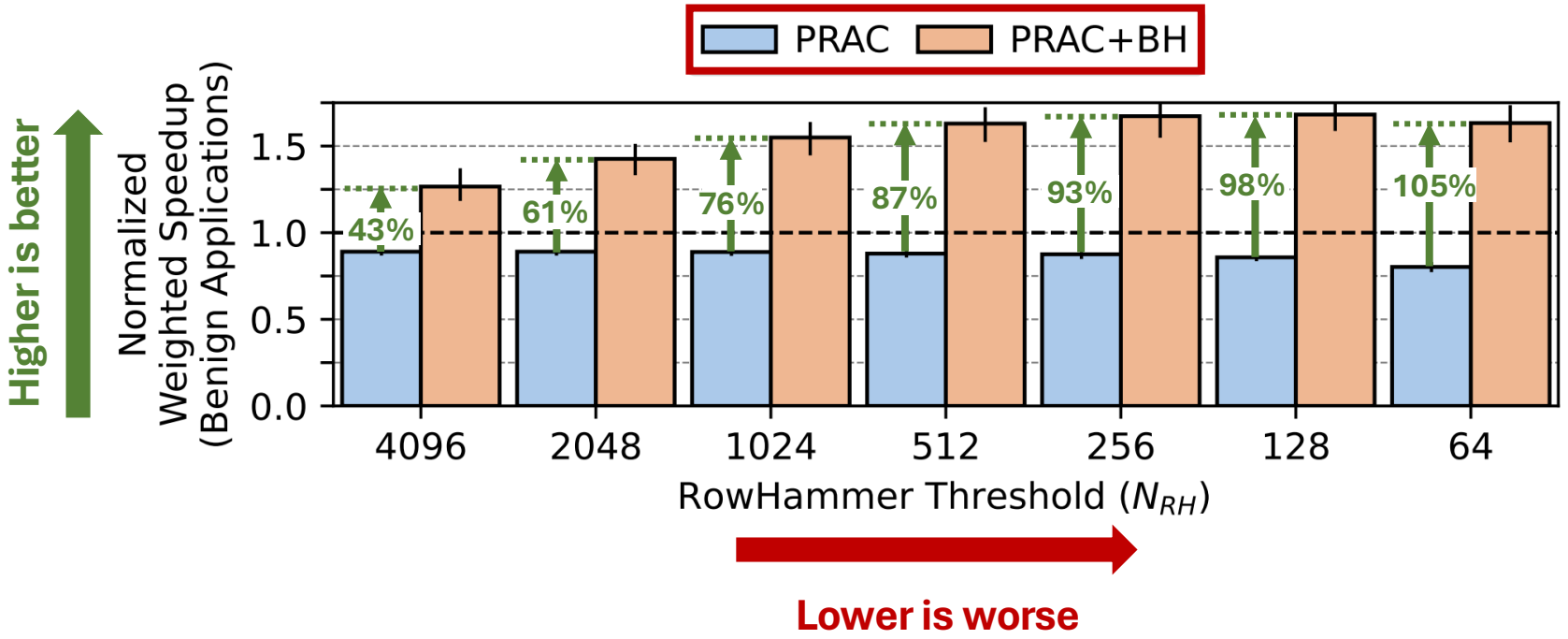


Memory Latency Impact at N_{RH}=64



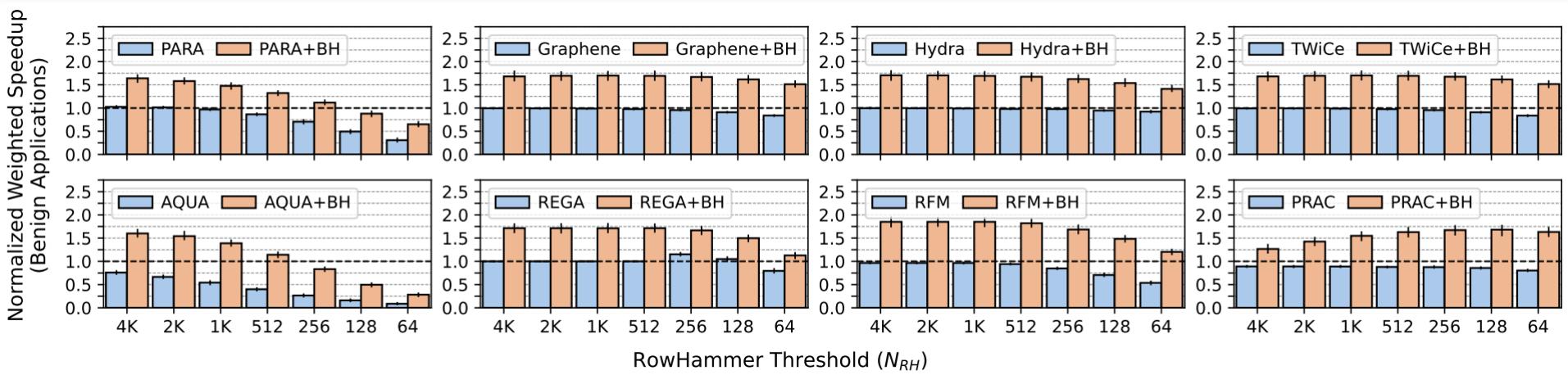
BreakHammer reduces memory latency across all mechanisms

Performance Impact and Its Scaling



BreakHammer significantly increases (81% on average) PRAC's performance

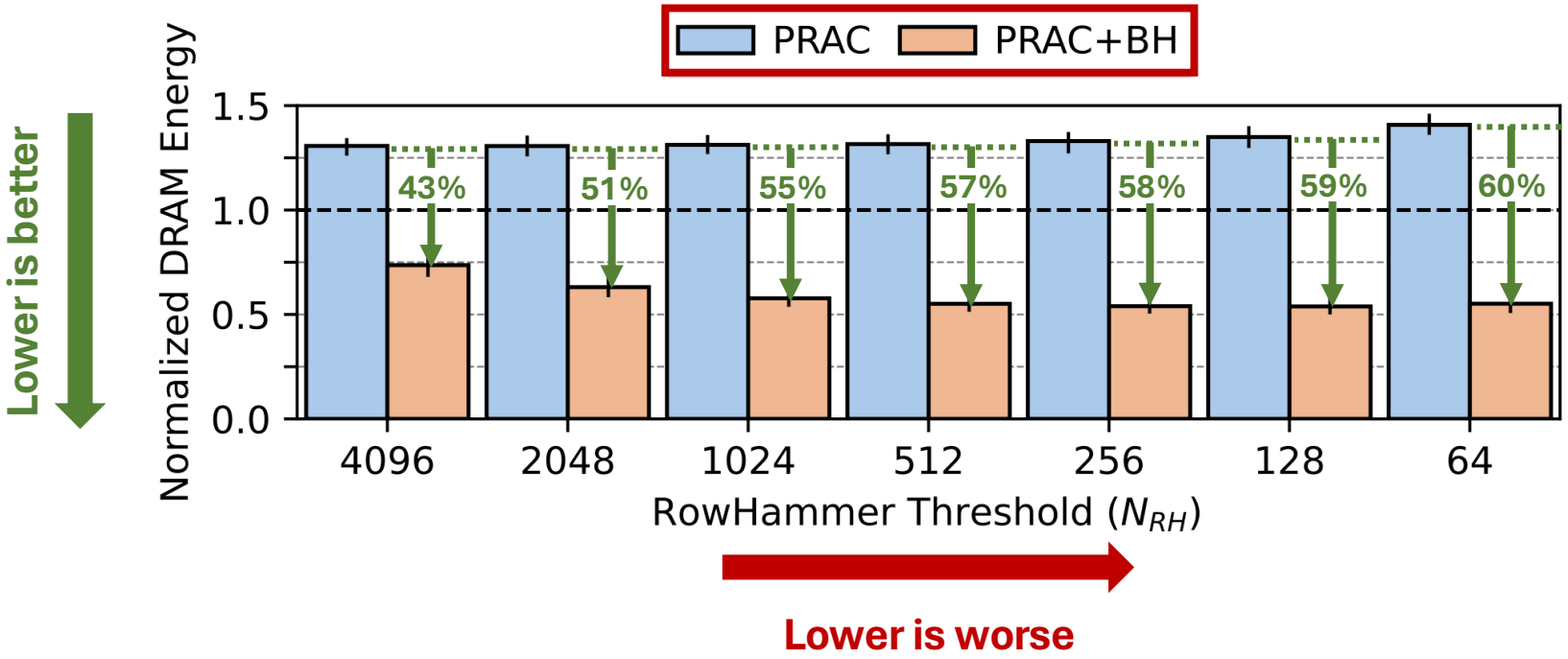
Performance Impact and Its Scaling



As RowHammer threshold **decreases**,
RowHammer mitigation mechanisms incur **increasing performance** overhead

BreakHammer significantly increases system performance **(90% on average)**

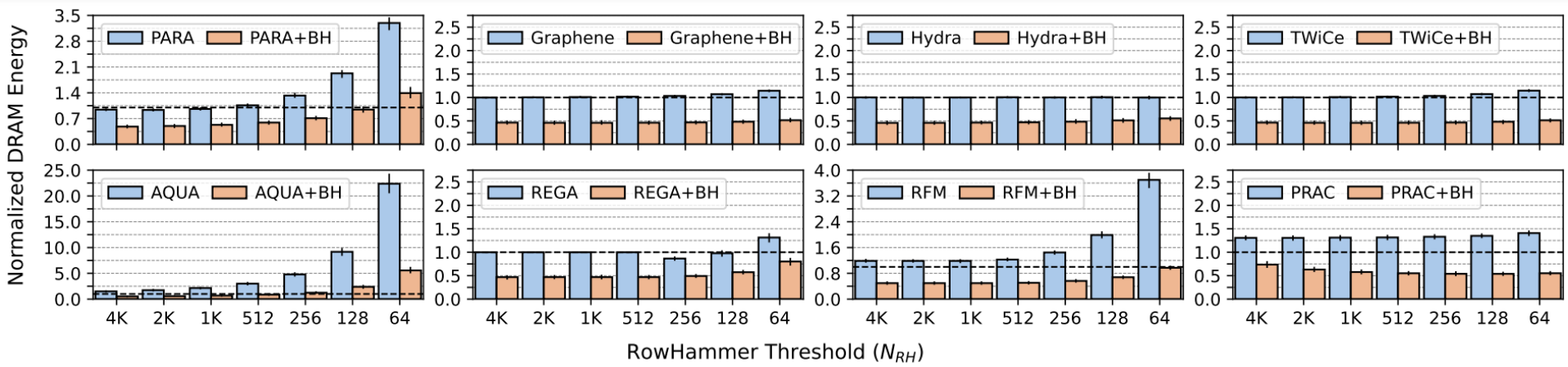
DRAM Energy Impact and Its Scaling



BreakHammer significantly reduces (by 55% on average) PRAC's energy consumption

S4L ROOTSEC CISPA HELMHOLTZ CENTER FOR INFORMATION SECURITY UNIVERSITÄT DES SAARLANDES

DRAM Energy Impact and Its Scaling



As RowHammer threshold **decreases**, RowHammer mitigation mechanisms consume **significantly increasing DRAM energy**

BreakHammer significantly decreases energy consumption **(by 55% on average)**

Under Attack Summary

BreakHammer significantly **reduces** the **performance and energy overheads** of existing RowHammer mitigation mechanisms when a **memory performance attack** is present

1) **BreakHammer accurately detects** suspect threads

2) **BreakHammer effectively reduces** the memory interference caused by suspect threads

Evaluation Results

1) Under Attack

2) No Attack

No Attack Summary

- Across 90 four-core benign workload mixes:

- BreakHammer slightly (<1%) improves
  - **memory access latency**
  - **system performance**
  - **DRAM energy efficiency**

More in the Paper

- More **implementation details**
  - Resetting **BreakHammer counters**
  - Tracking **software threads**
  - Throttling **DMA** and **systems *without* caches**
  - Configuration parameters

- **Security analysis**
  - Upper bound on the **overhead an attacker can cause**
  - Security against **multi-threaded attackers**

- **Performance evaluation**
  - Unfairness results
  - Sensitivity to memory intensity of workloads
  - Comparison to BlockHammer
  - Sensitivity analysis of BreakHammer parameters

The Paper

BreakHammer: Enhancing RowHammer Mitigations by Carefully Throttling Suspect Threads

Oğuzhan Canpolat[§†] A. Giray Yağlıkçı[§] Ataberk Olgun[§] Ismail Emir Yuksel[§]
Yahya Can Tuğrul[§†] Konstantinos Kanellopoulos[†] Oğuz Ergin[‡§†] Onur Mutlu[§]
[§]*ETH Zürich* [†]*TOBB University of Economics and Technology* [‡]*University of Sharjah*

RowHammer is a major read disturbance mechanism in DRAM where repeatedly accessing (hammering) a row of DRAM cells (DRAM row) induces bitflips in other physically nearby DRAM rows. RowHammer solutions perform preventive actions (e.g., can experience bitflips when a nearby DRAM row (i.e., aggressor row) is repeatedly opened (i.e., hammered) [2–70].

Many prior works demonstrate attacks on a wide range of systems where they exploit read disturbance to escalate

https://arxiv.org/abs/2404.13477

S4L ROOTSEC CISPA HELMHOLTZ CENTER FOR INFORMATION SECURITY UNIVERSITÄT DES SAARLANDES

Outline

Background

Motivation

BreakHammer

Evaluation

Conclusion

Conclusion

Key Exploit: Mount a **memory performance attack** by triggering RowHammer-preventive actions to **block memory accesses** for long periods of time

Key Mechanism: BreakHammer

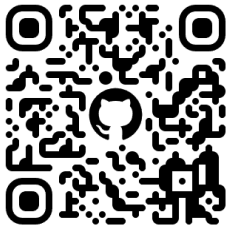
- **Observes** triggered RowHammer-preventive actions
- Identifies threads that trigger **many preventive actions (i.e., suspect threads)**
- **Reduces the memory bandwidth usage** of the suspect threads

Key Results:

- **Under attack**:
  - Significantly **improves** system performance (by 90% on average)
  - Significantly **reduces** energy consumption (by 55% on average)
- **No attack**:
  - Slightly (<1%) **improves** performance and energy consumption

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

Open Source and Artifact Evaluated



CMU-SAFARI / BreakHammer

Type / to search

Code Issues Pull requests Actions Projects Security Insights Settings

BreakHammer Public

Edit Pins Watch 3 Fork 0 Star 4

master 2 Branches Tags Go to file t Code

kirbyydoge Update README.md 2ea4b97 · last month 32 Commits

ae_results	Update existing csvs and plots with full artifact evaluat...	2 months ago
mixes	Initial commit	2 months ago
plotting_scripts	Update figure13 plotter to work when some mitigatio...	2 months ago
scripts	Remove unreleased empty scripts	last month
src	Initial commit	2 months ago
.gitattributes	Update Dockerfile	2 months ago

About

No description, website, or topics provided.

Readme
Activity
Custom properties
4 stars
3 watching
0 forks
Report repository

Releases

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

S4L ROOTSEC CISPA HELMHOLTZ CENTER FOR INFORMATION SECURITY UNIVERSITÄT DES SAARLANDES

BreakHammer: Enhancing RowHammer Mitigations by Carefully Throttling Suspect Threads

Oğuzhan Canpolat A. Giray Yağlıkçı Ataberk Olgun İsmail E. Yüksel

Yahya C. Tuğrul Konstantinos Kanellopoulos Oğuz Ergin Onur Mutlu

SAFARI ETH zürich kasırga

Venue : *MICRO 2024*
Presenter : *A. Giray Yaglikci*
Date : *27 April 2026*

S4L ROOTSEC CISPA HELMHOLTZ CENTER FOR INFORMATION SECURITY UNIVERSITÄT DES SAARLANDES

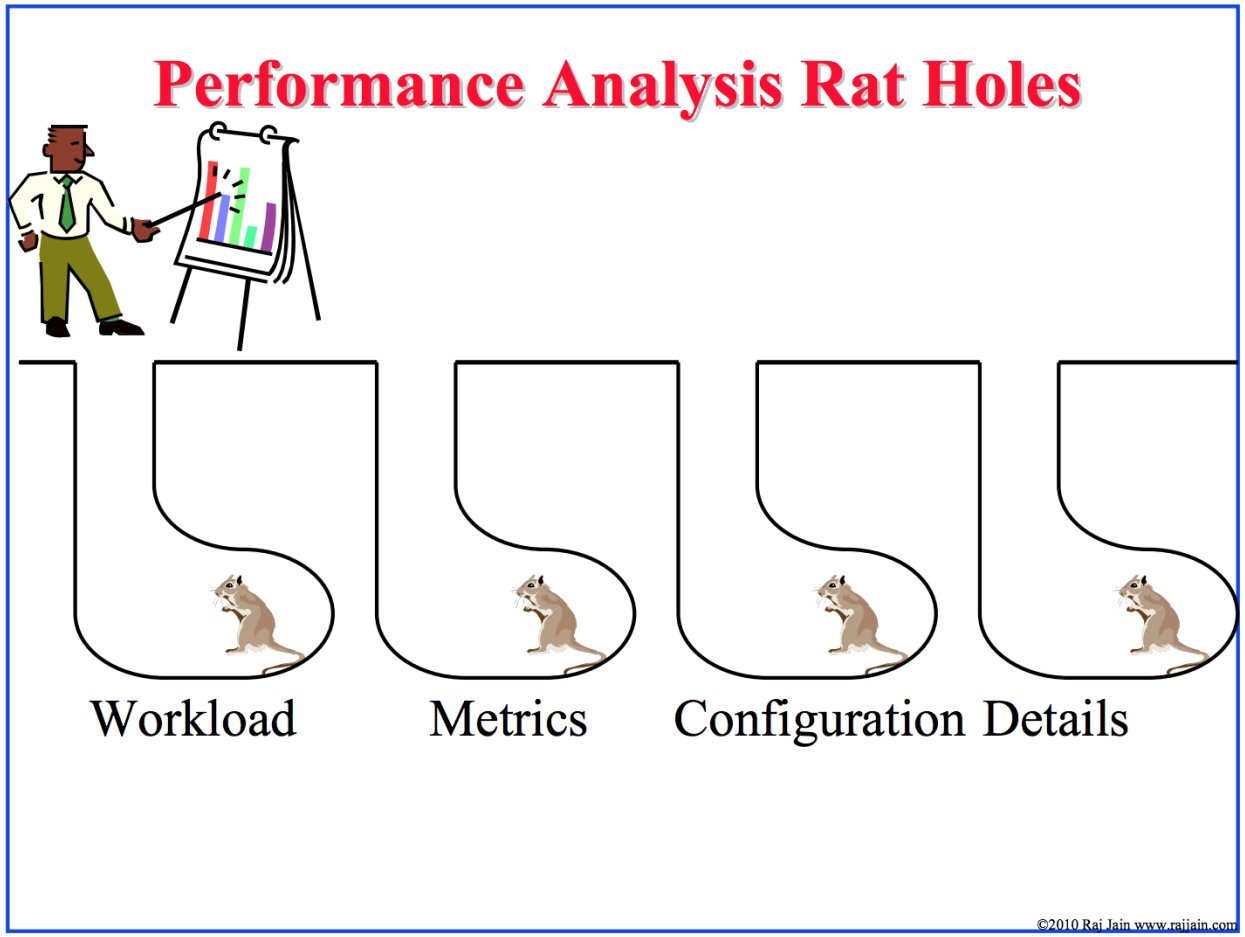
Strengths

- The paper tackles an important problem:
  Performance and energy overheads of RowHammer mitigation mechanisms
- BreakHammer is a lightweight and effective mechanism
- BreakHammer can be integrated with many existing RowHammer mitigations
- The authors already explain and open source integration with 8 mitigations
- Evaluation shows that BreakHammer
  - alleviates the negative impact of a RowHammer attack on system performance and energy consumption
  - does not slowdown benign workloads
- BreakHammer's implementation and evaluation scripts are opensourced
- …

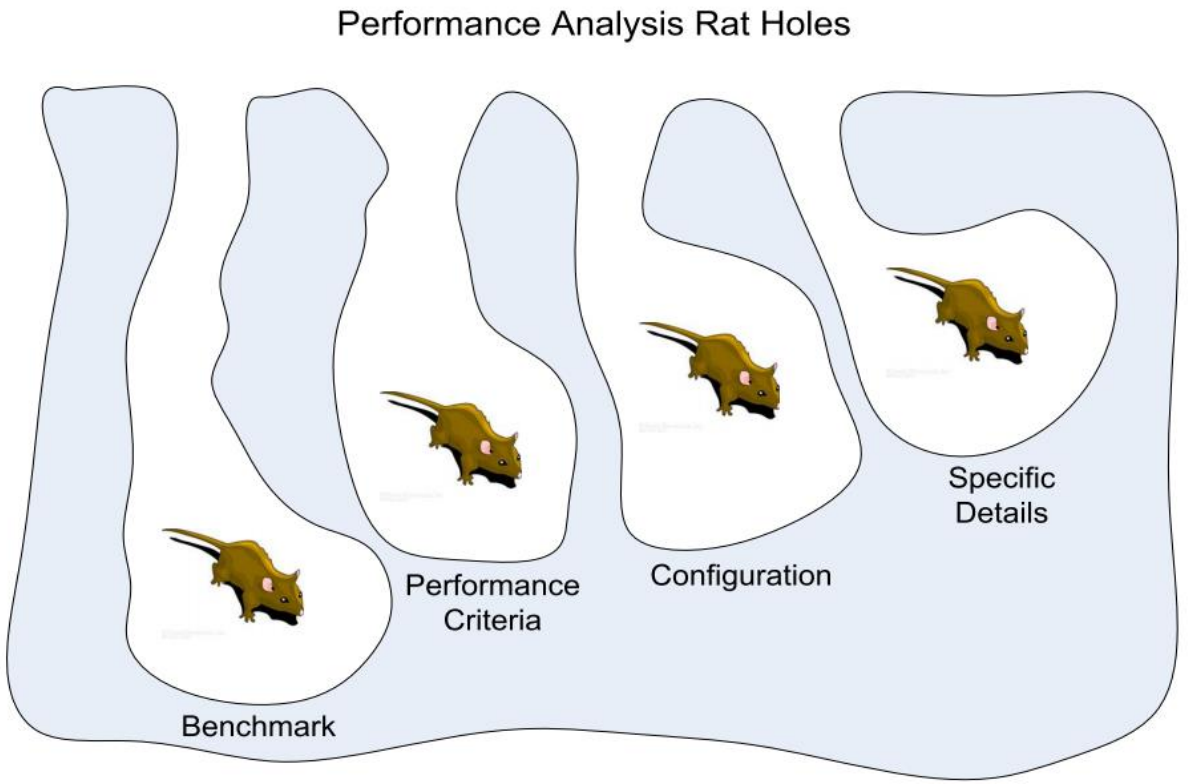
Weaknesses

- The memory performance attack is not demonstrated on a real system
- BreakHammer is a hardware-level mechanism
  - Does not have the context of software threads, processes, users, and address spaces
  - Cannot track an attacker across threads and processes
- As a memory controller-based mechanism, BreakHammer cannot observe the internals of in-DRAM solutions
  - A thread's score can increase if it performs a few activations to each of many rows
  - Processing in Memory (PiM) enables computation within DRAM chips
    - PiM cores generate memory requests within DRAM chips (after BreakHammer)
    - BreakHammer cannot identify RowHammer attacks performed from within PiM cores
- Majority of the plots are bar plots, and they are not very easy to read
- …

Avoid Rat Hole Discussions



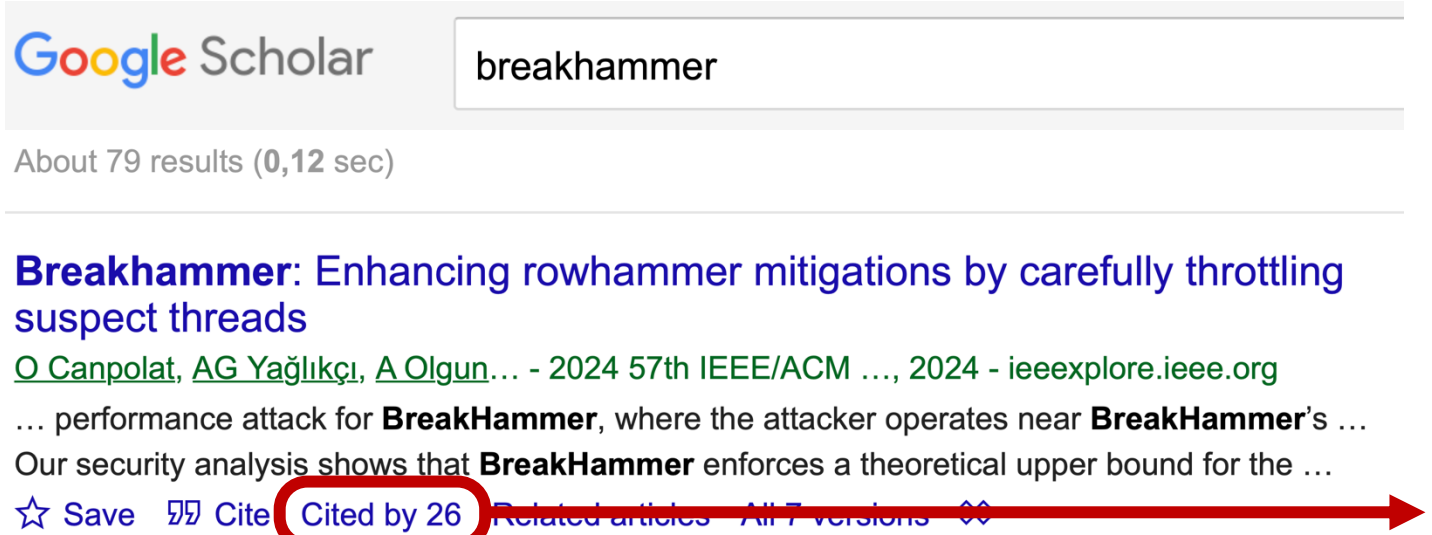
Source: https://www.cse.wustl.edu/~jain/iucee/ftp/k_10adp.pdf



Source: P. Jarupunphol, “Using Buddhist Insights to Analyse the Cause of System Project Failures,” Ph.D. Thesis, 2013

Every work has strengths and weaknesses

- Papers are filtered into our class as they have merit
- They should already emphasize their strengths: start with their major contributions
- There is always room for improvement, so there are weaknesses
- Some papers acknowledge some of their weaknesses
- Many weaknesses are not obvious, but you have a brain and many others' brains
- Check the arguments in papers that improve the paper you present
- Important! You should be skeptical about those claims too. Practice caution!



fenses. BreakHammer identifies potentially malicious threads by tracking the number of triggered RH mitigations per hardware thread and throttles their memory requests. However, it depends on prior RH solutions for mitigation and requires microarchitectural modifications, such as changes to caches, to propagate thread information to the throttling logic in the memory controller. In contrast, DAPPER provides standalone *Perf-Attack* resilience with minimal overhead without requiring additional microarchitectural changes. Furthermore, DAPPER can be combined with BreakHammer to enhance protection against *Perf-Attacks*. Another concurrent

Jeonghyun Woo and Prashant J. Nair, "Dapper: A Performance-Attack-Resilient Tracker for RowHammer Defense," in HPCA, 2025.

Discussion Points

- Some bad discussion point examples for this paper:
  - RowHammer is a problem for DRAM. What about other memory technologies?
    → This is a discussion way out of this paper's scope
  - What are the mitigation mechanisms integrated with BreakHammer?
    → Discussion is not a quiz for the audience, the goal is to brainstorm
  - RowHammer mitigation mechanisms incur higher performance overheads with increasing DRAM density. Should we stop scaling DRAM density?
    → Too philosophical and arguably backwards looking
- Weaknesses are your friends to find good discussion points
- Find fundamental weaknesses
- Ask: Can we do better?
- Discuss how to do better

Discussion Points

- Example weakness:
  “Memory performance attacks are not demonstrated on a real system.”
- Discussion point: Should we do that?
  - A good starting point, but needs more work
  - What is the benefit of demonstrating them? What research problem does it solve?
  - Any papers solving that problem? What is missing?
  - What is your solution?
    - any hypothesis?
    - key insights, observations, and ideas?
    - expected results?
    - What are some practical challenges?
      - What do we need to reverse engineer?
      - Do we have real systems that implement studied or other RowHammer mitigations?
- Leading a discussion is more than throwing open-ended questions
- Be constructive: how can we advance the state-of-the-art?

Discussion Points

- Example Weakness: "*BreakHammer cannot track an attacker across threads and processes.*"
- Is this a problem? Yes. You can bypass BreakHammer via multithreading. Still no bitflips, but you can block memory
  - Each thread activates a row a few times
  - Cumulative activation count across threads triggers RowHammer mitigations
  - RowHammer mitigations cause performance degradation and increase energy consumption
  - Practically: BreakHammer becomes useless
- Goal: Enable tracking RowHammer-based memory performance attacks across threads
- Key idea: Hardware-OS cooperation
  - BreakHammer collects scores at the hardware thread level
  - OS sums up the scores of a process or user an identifies process- and user-level behavioral patterns
  - OS performs throttling:
    - OS asks BreakHammer to throttle memory accesses of a hardware thread
    - OS deprioritizes suspicious users and processes in CPU scheduling
    - OS uses memory bandwidth allocation features of modern processors
- Any similar works?
  - Fiedler et al., "*Memory Band-Aid: A Principled RowHammer Defense-in-Depth,*" in NDSS, 2026.
  - This is a software-only throttling-based RowHammer mitigation mechanism, aiming to prevent bitflips
  - As a software-level mechanism, it can incur very high performance overheads, opposite of what we want
- Expected results:
  - Providing an end-to-end solution to RowHammer, where
  - bitflips are prevented within the DRAM chip
  - attackers are stopped at the OS level

Computer Architecture
Seminar/Proseminar
Lecture 1b: Example Presentation

A. Giray Yaglikci

27 April 2026

S4L ROOTSEC CISPA HELMHOLTZ CENTER FOR INFORMATION SECURITY UNIVERSITÄT DES SAARLANDES