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

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

Executive Summary

Problem:

- DRAM continues to become more vulnerable to RowHammer
- Operations that prevent RowHammer (i.e., 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

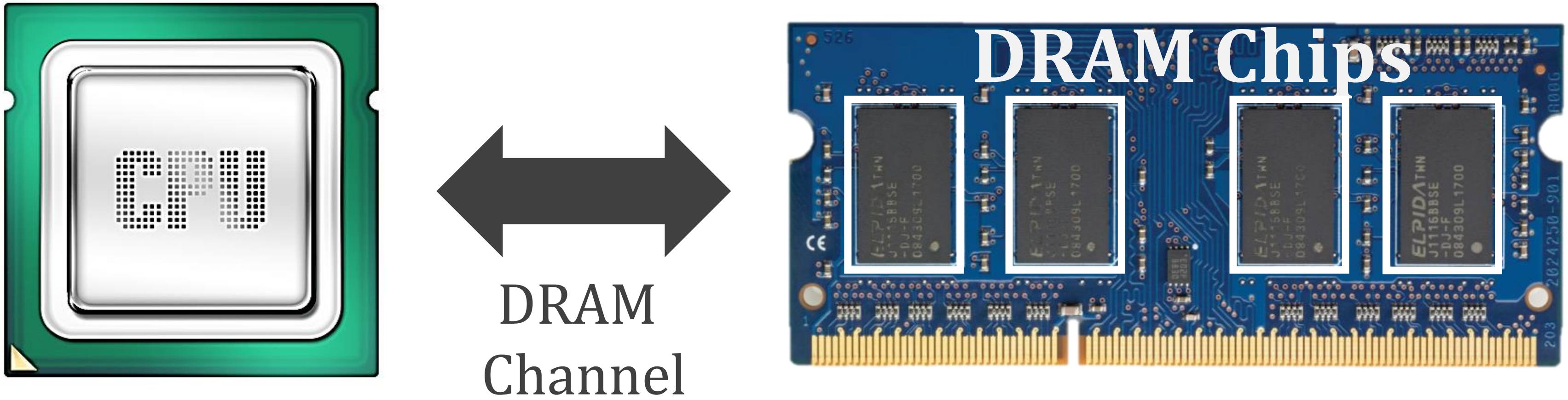
- Observes triggered RowHammer-preventive actions
- Identifies threads that trigger many preventive actions (i.e., suspect threads)
- Reduces the memory bandwidth usage of the suspect threads

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

Outline

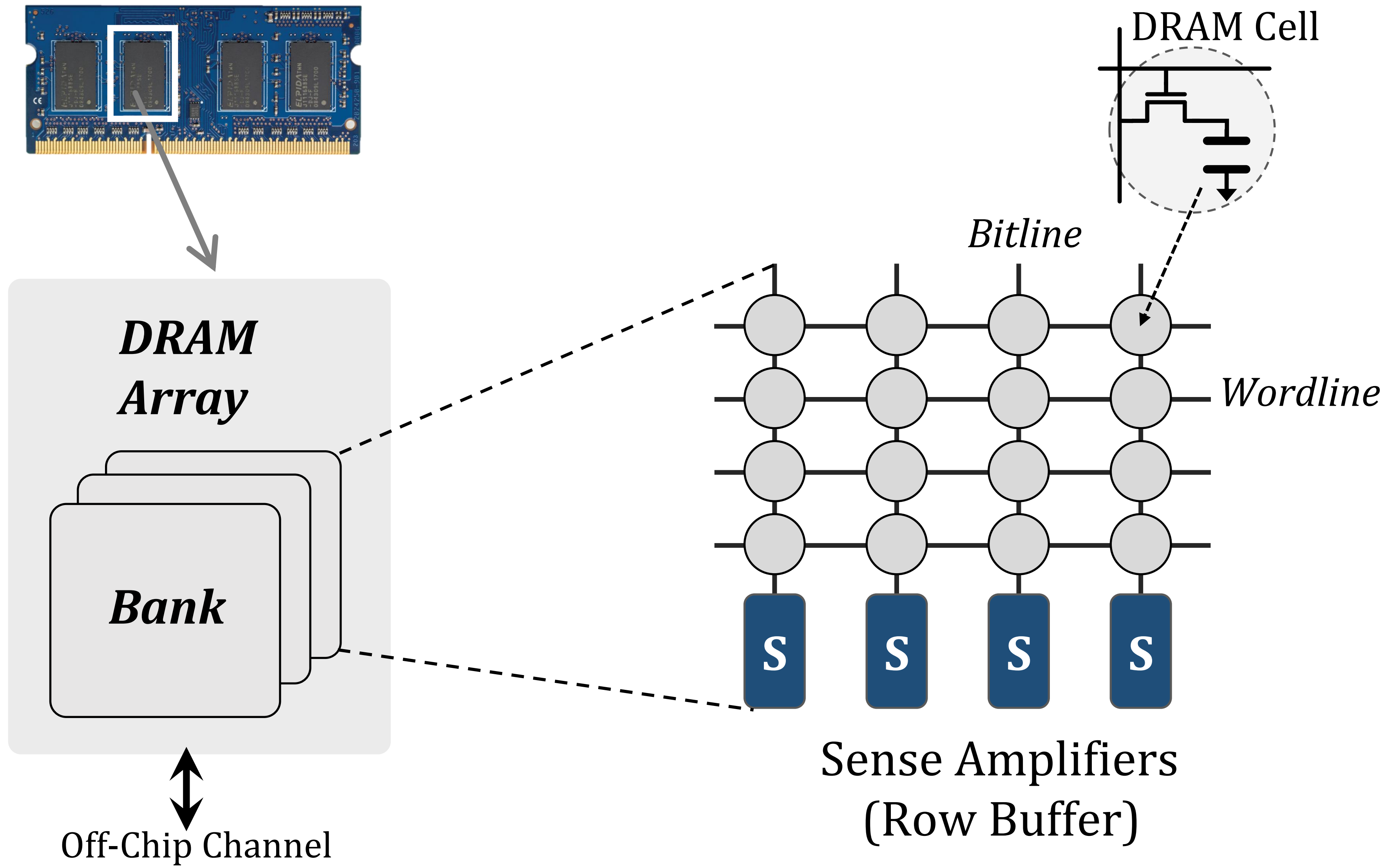
- Background
- Motivation
- BreakHammer
- Evaluation
- Conclusion

DRAM Organization

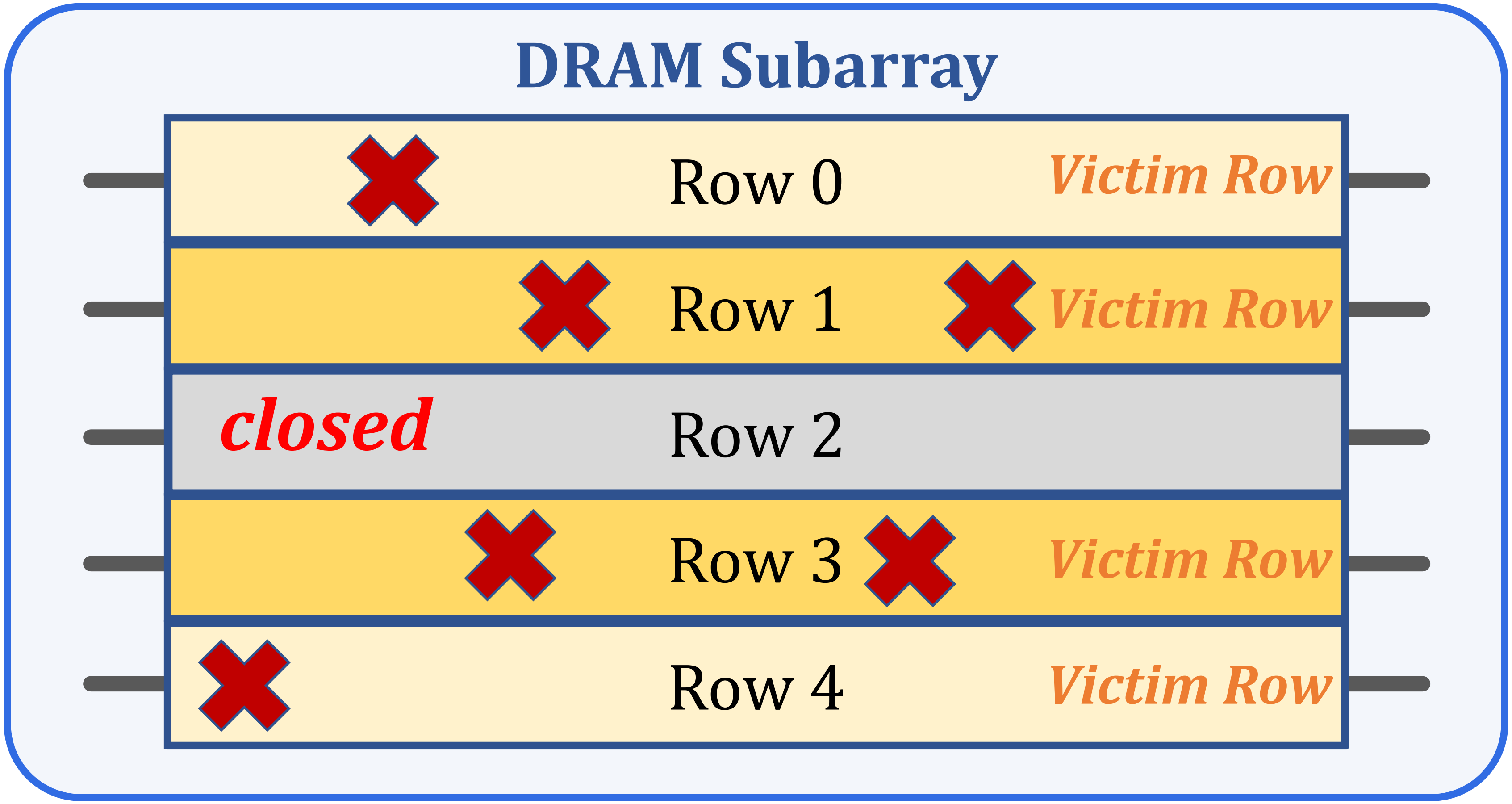


DRAM Module

DRAM Organization



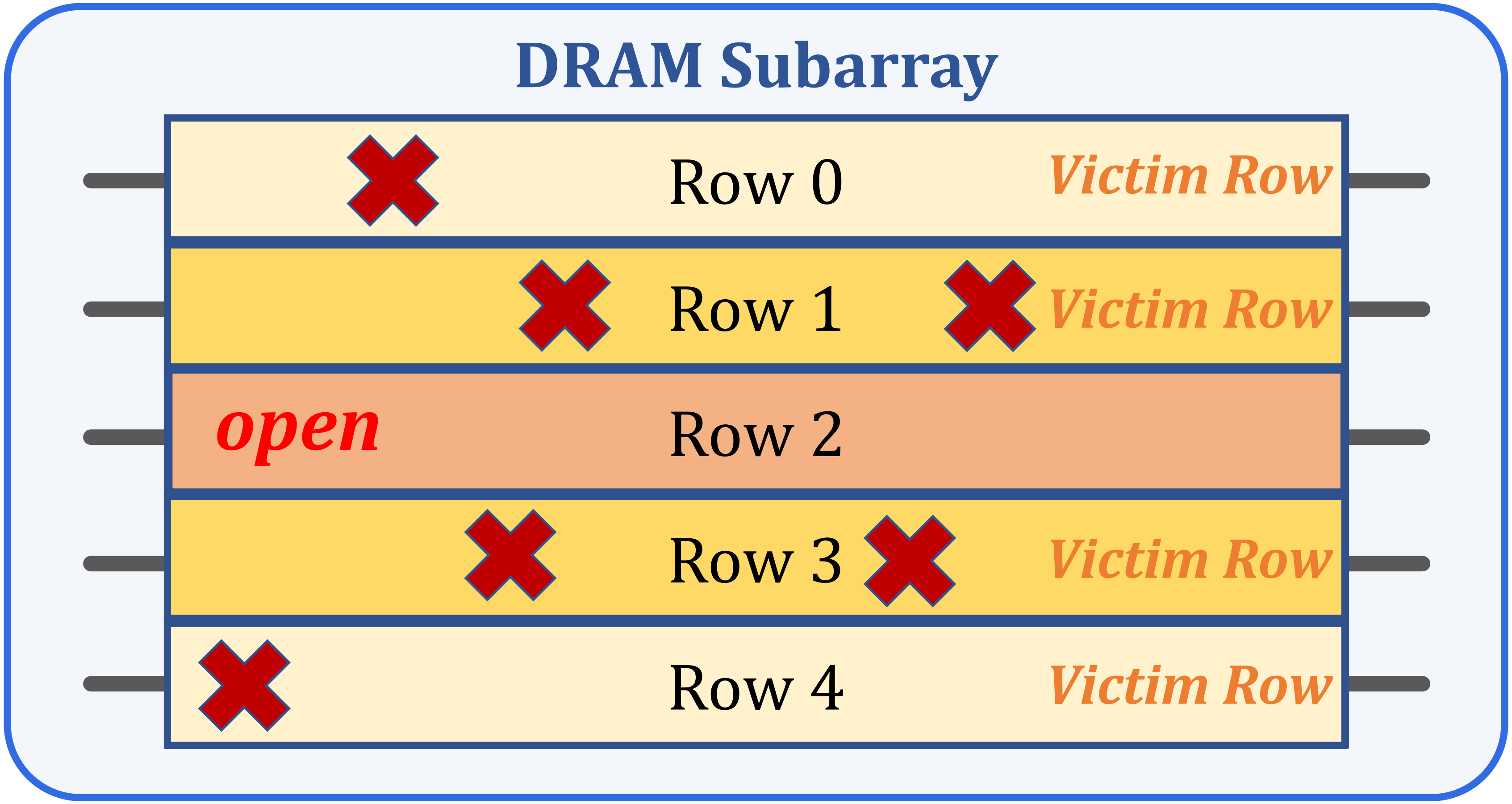
RowHammer: A Prime Example of Read Disturbance



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

[Kim+ ISCA'20]

Read Disturbance Vulnerabilities (I)

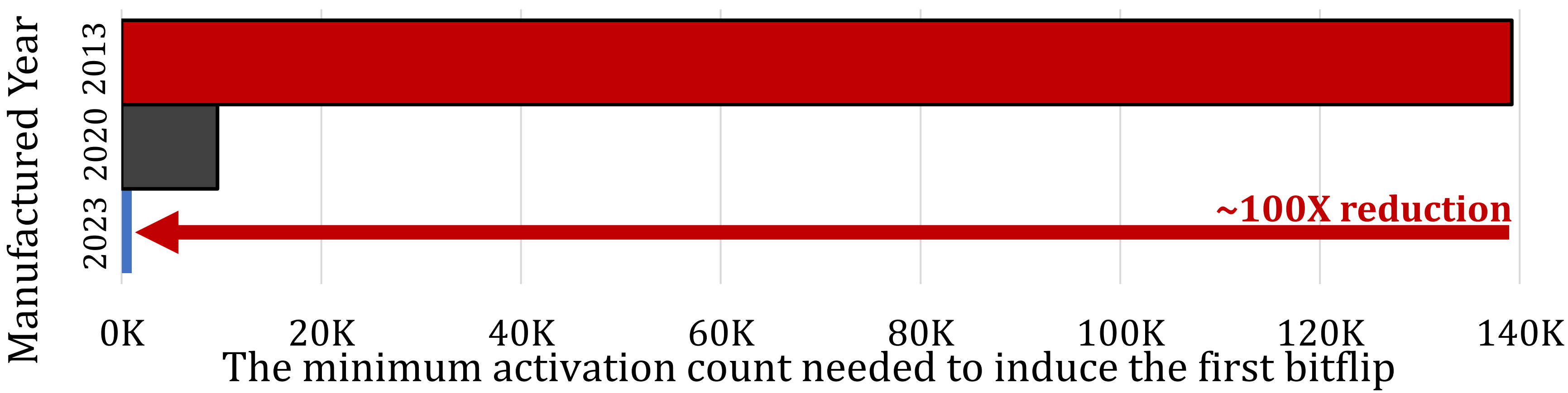


The minimum number of activations that causes a bitflip is called **the RowHammer threshold (N_{RH})**

[Kim+ ISCA'20]

Read Disturbance Vulnerabilities (II)

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



Existing RowHammer Mitigations: RowHammer-Preventive Actions

Many ways to prevent RowHammer via

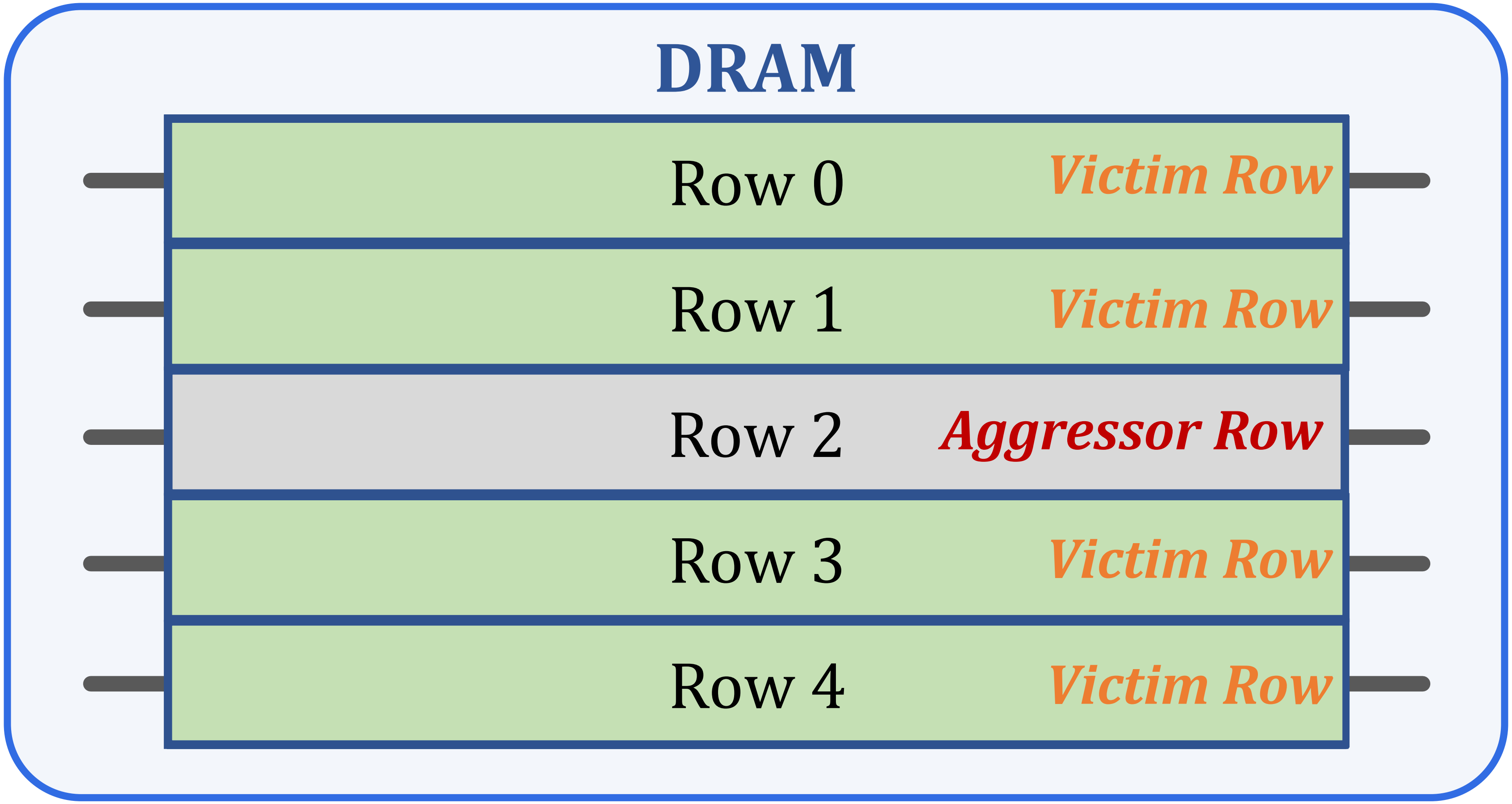
RowHammer-preventive actions:

- **Preventive refresh**
- **Row migration**

State-of-the-art RowHammer mitigation mechanisms adopt these two approaches

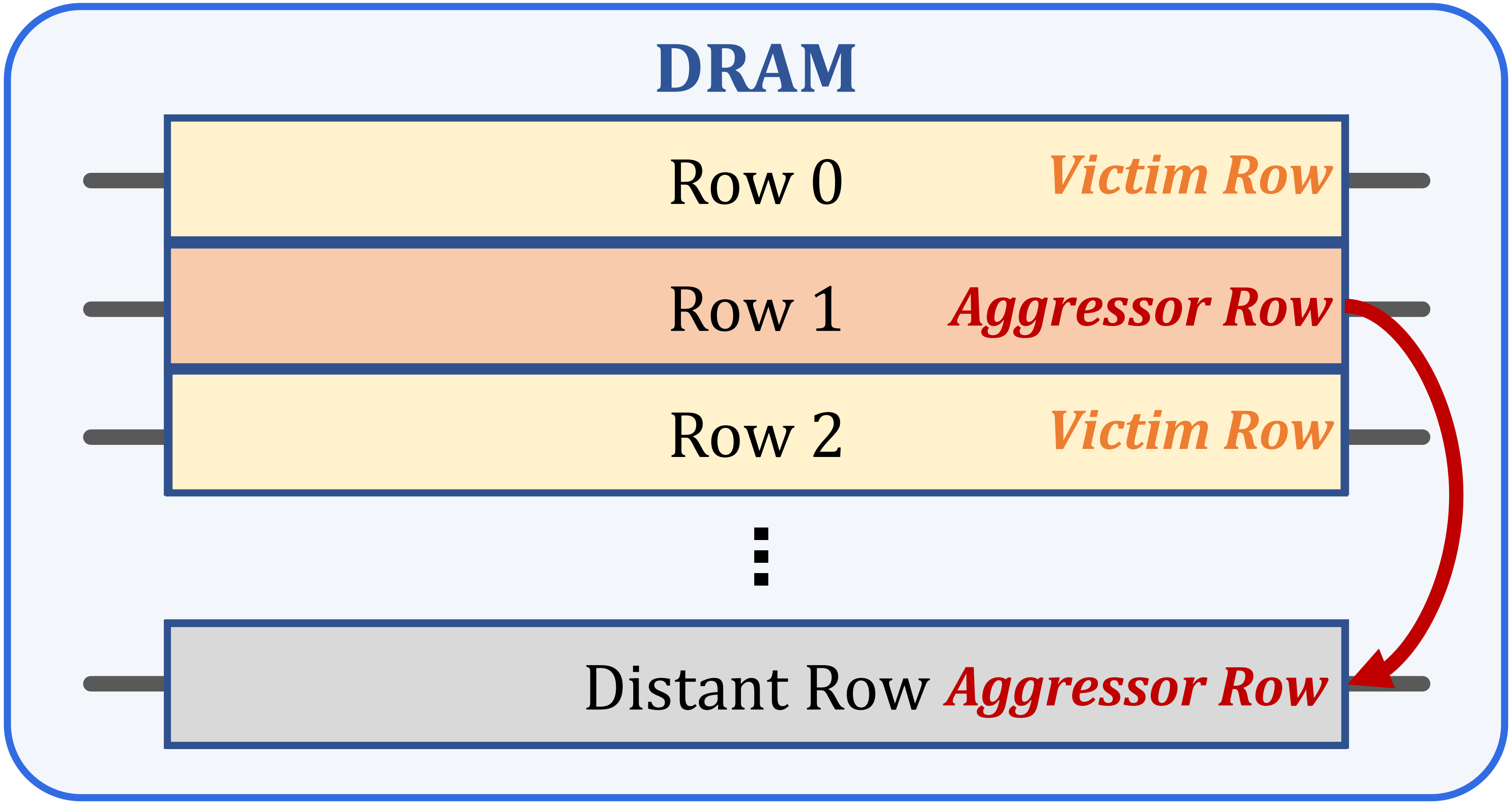
- **Proactive throttling**
- **...**

Preventive Refresh as a RowHammer-Preventive Actions



Refreshing potential victim rows
mitigates RowHammer bitflips

Row Migration as a RowHammer-Preventive Action



Migrating potential aggressor rows to a distant row **mitigates RowHammer bitflips**

Outline

Background

Motivation

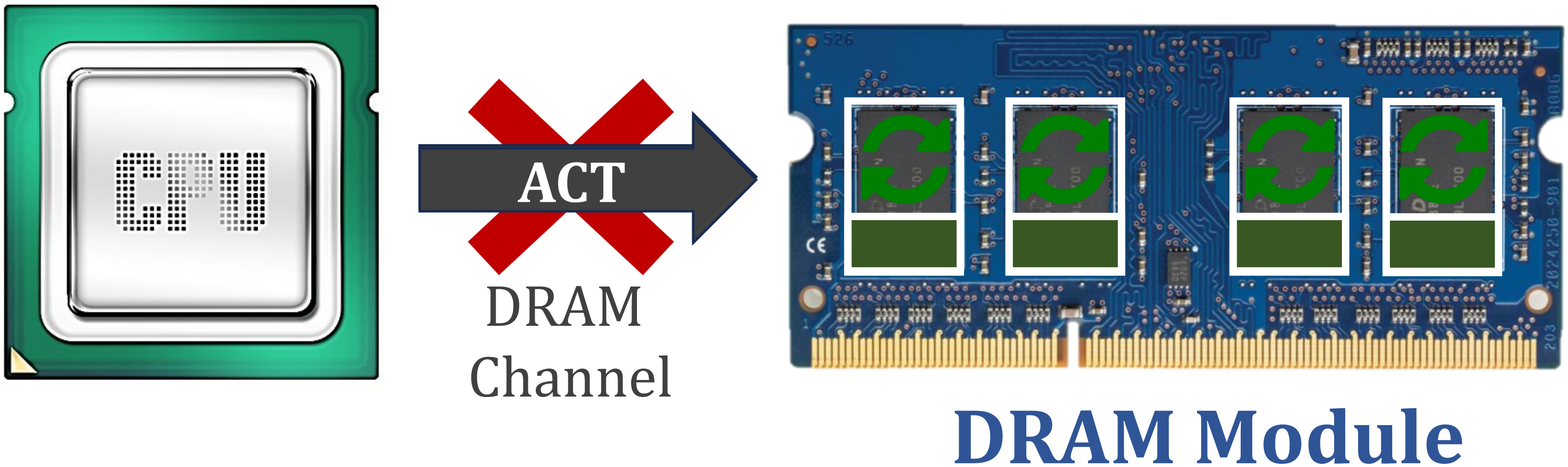
BreakHammer

Evaluation

Conclusion

Root Cause of Performance Overhead

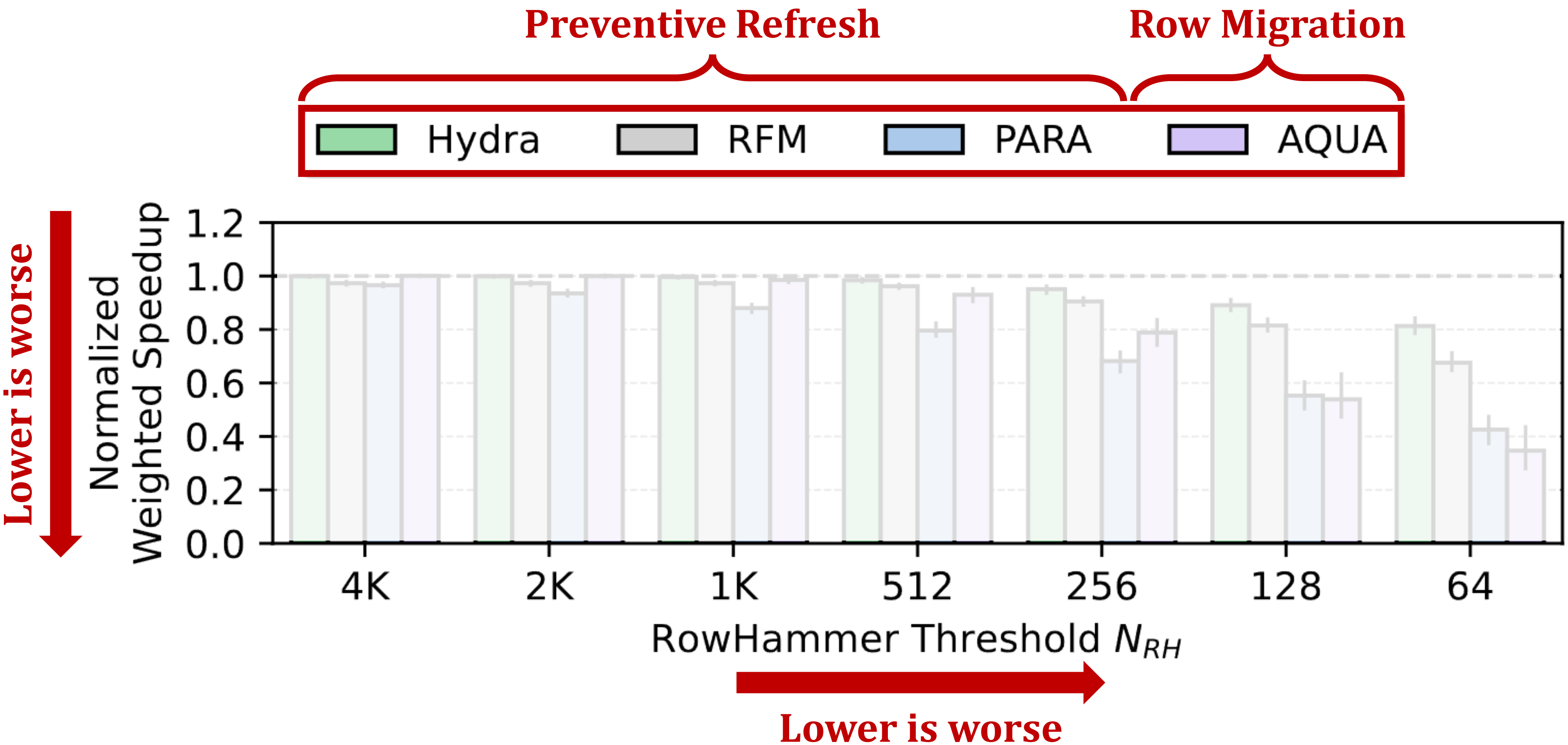
RowHammer-preventive actions are **blocking and time consuming** operations



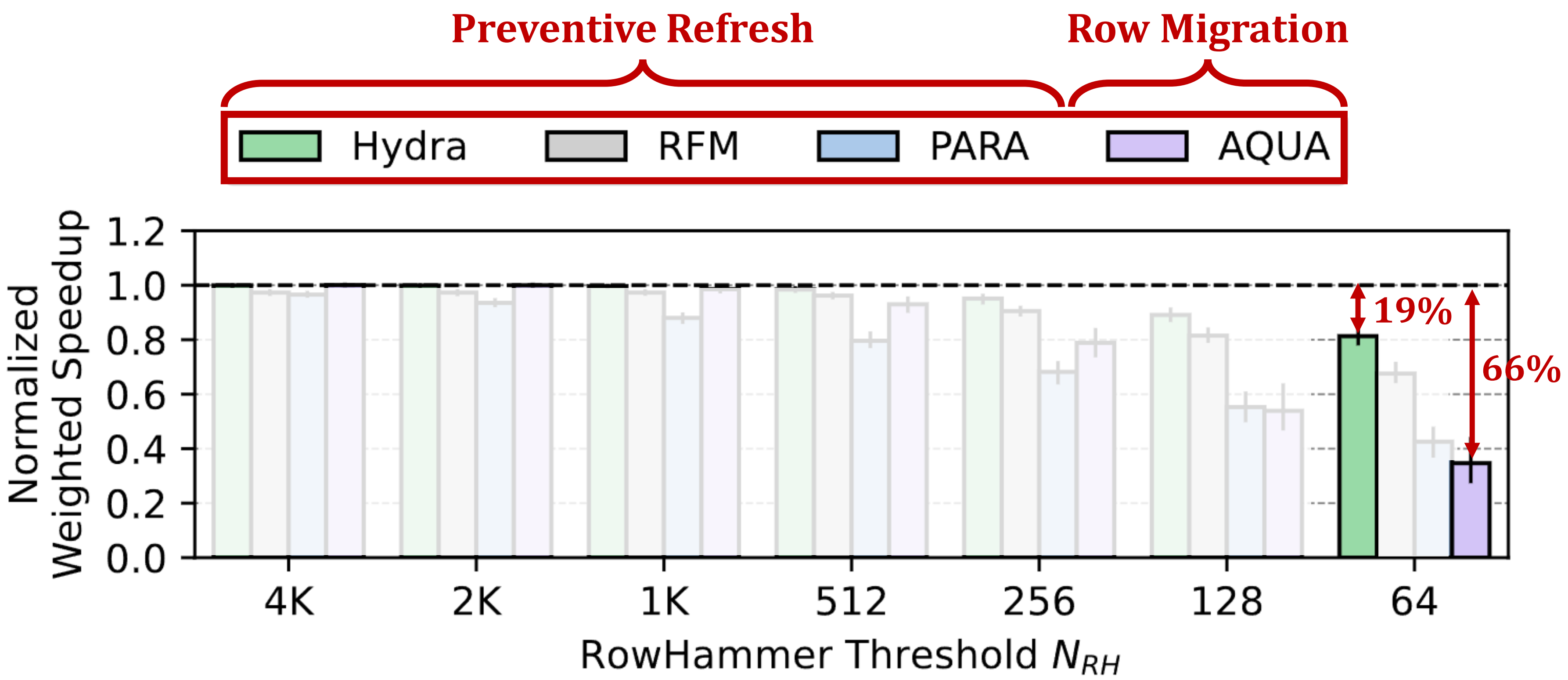
Memory controller **cannot access** a memory bank undergoing a RowHammer-preventive action

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

RowHammer Mitigation Performance Overhead



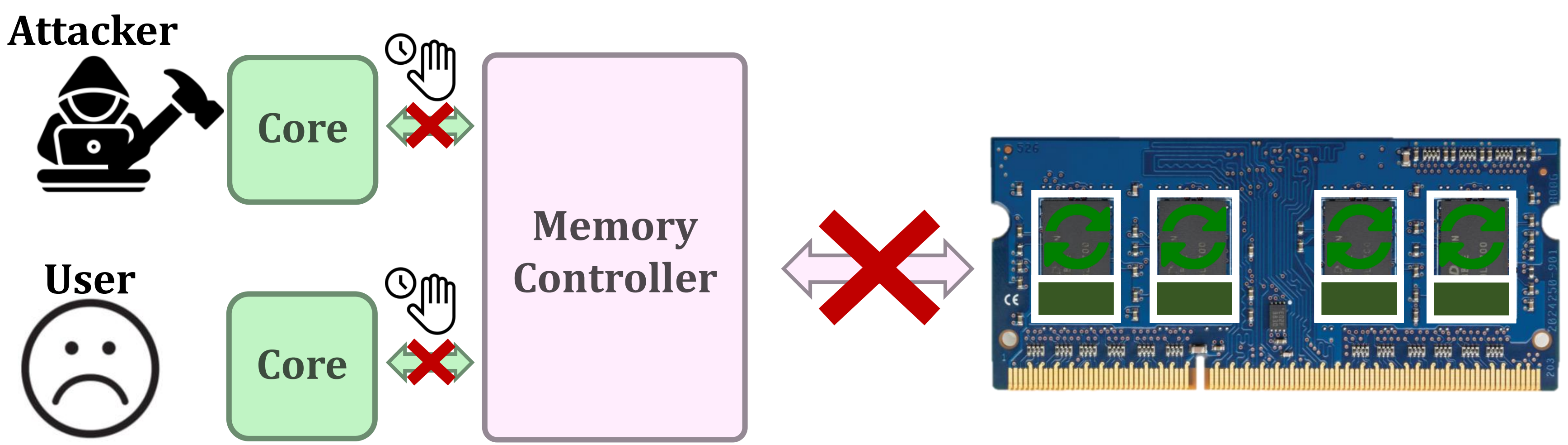
RowHammer Mitigation Performance Overhead



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

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

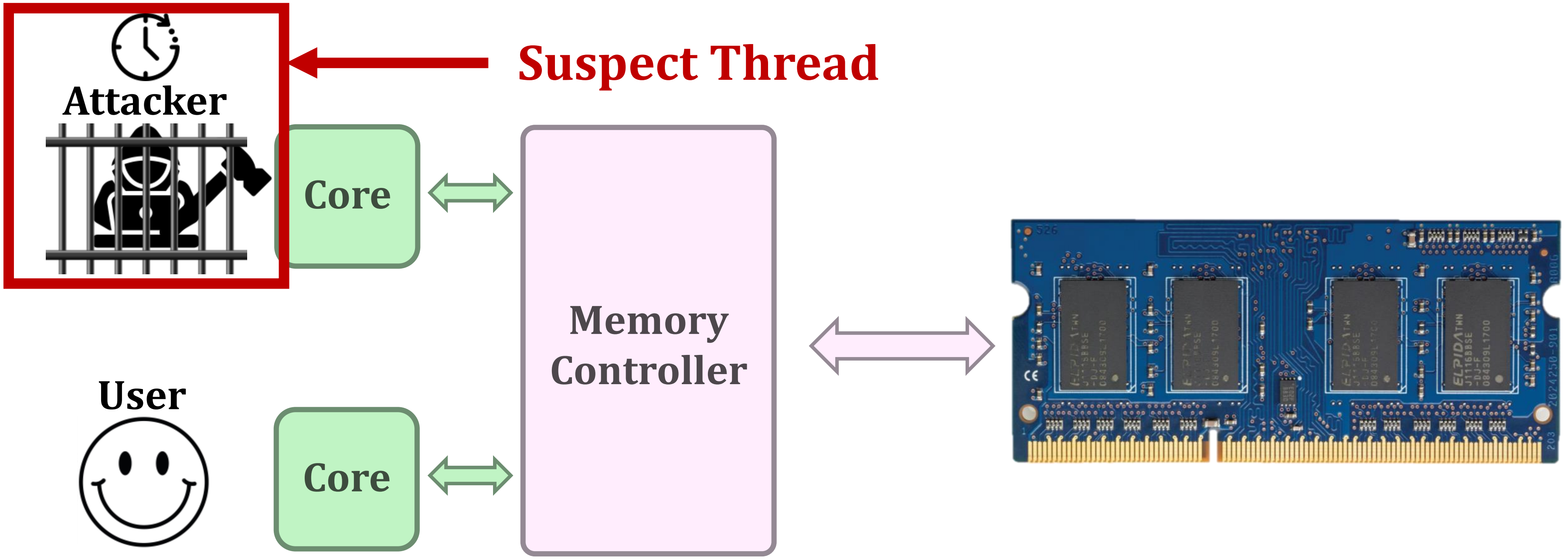
BreakHammer

Evaluation

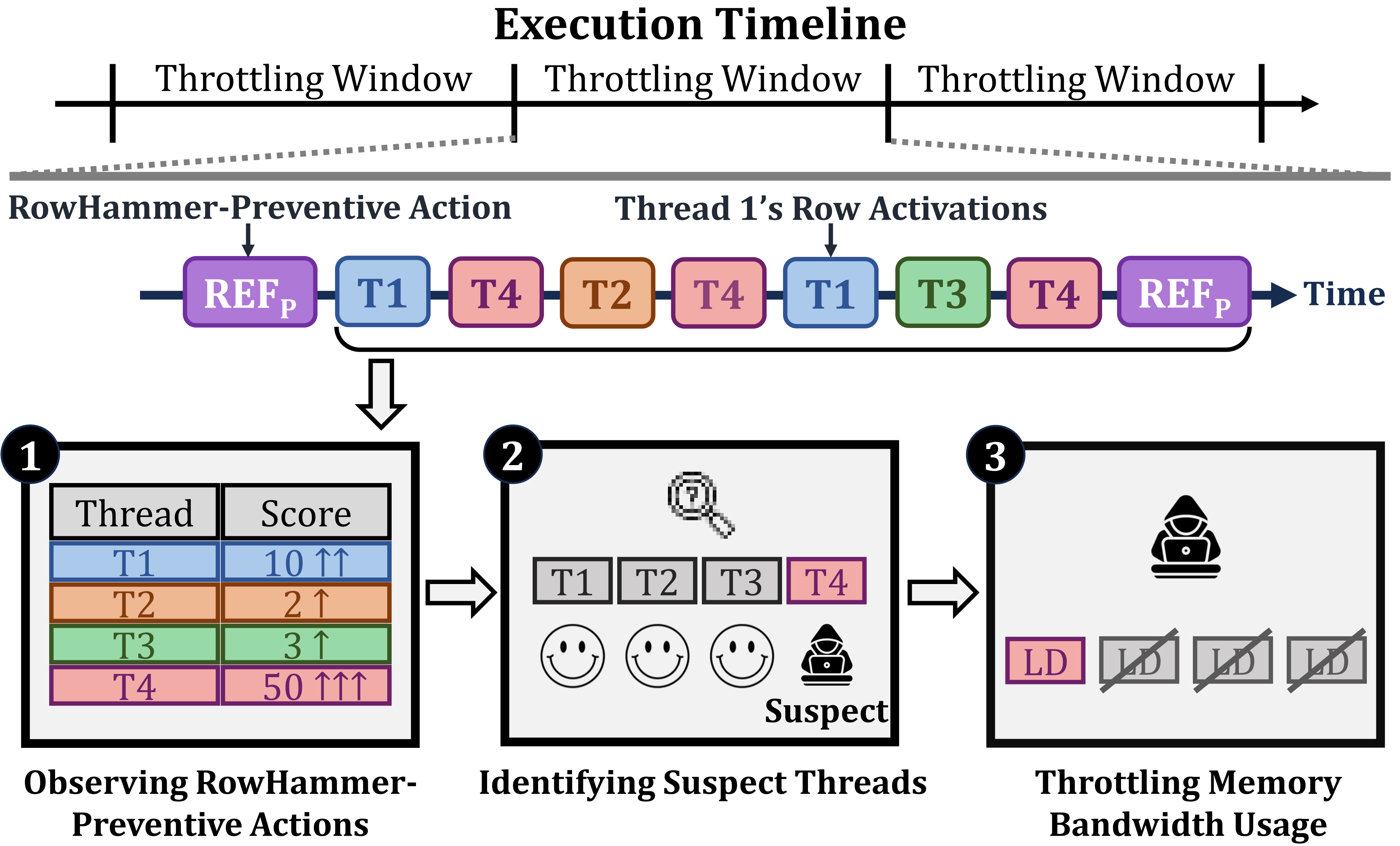
Conclusion

Key Idea

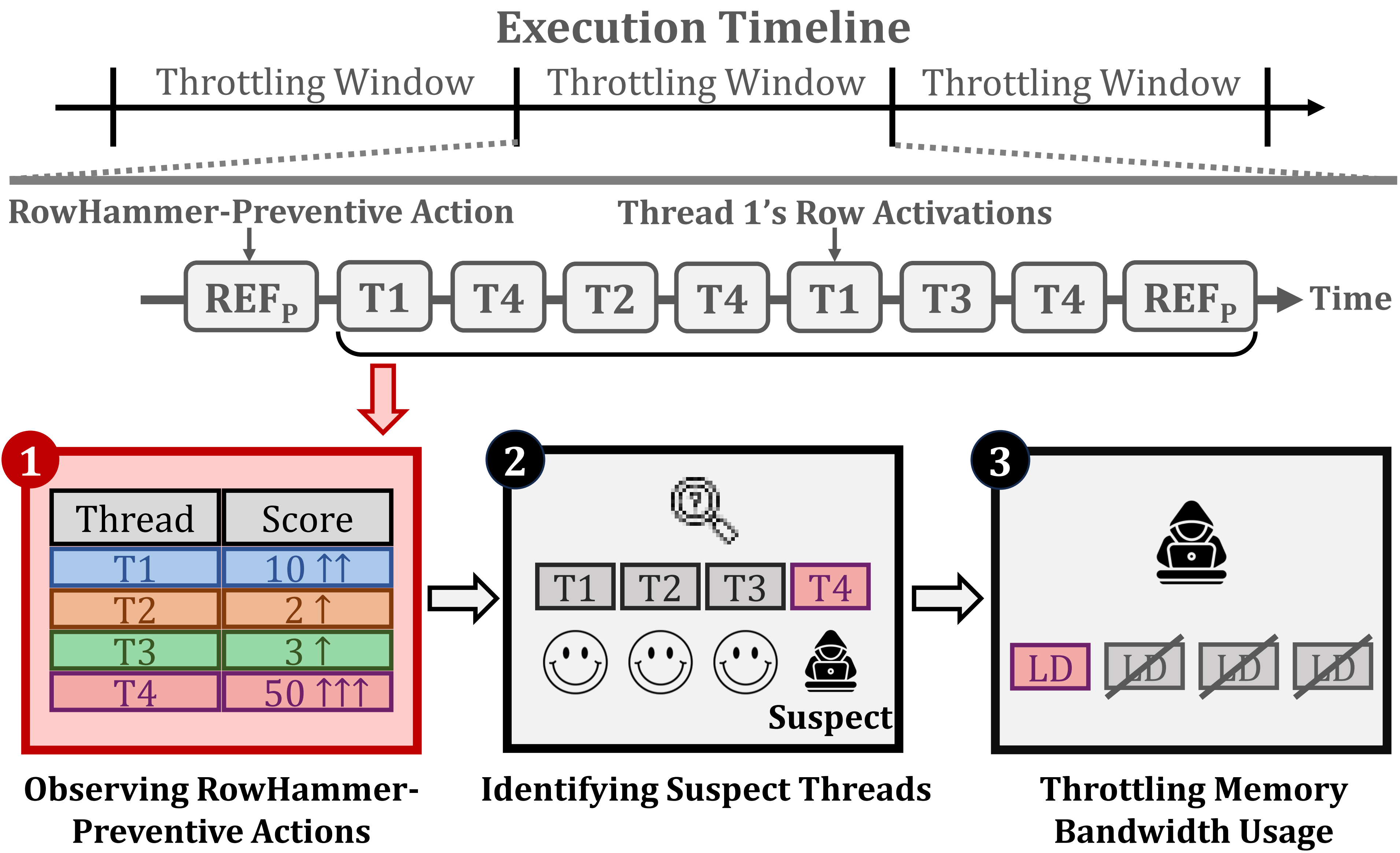
Detect and **slow down** the memory accesses of threads that trigger **many** RowHammer-preventive actions



BreakHammer: Overview

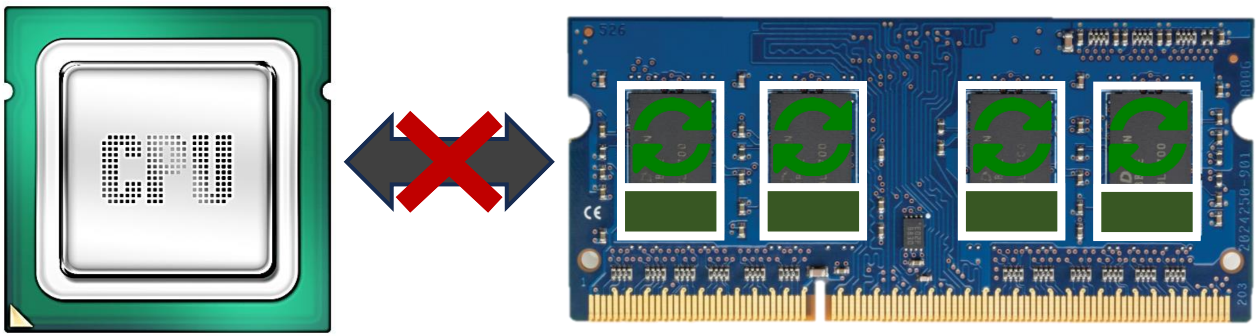


BreakHammer: Overview



Observing RowHammer-Preventive Actions

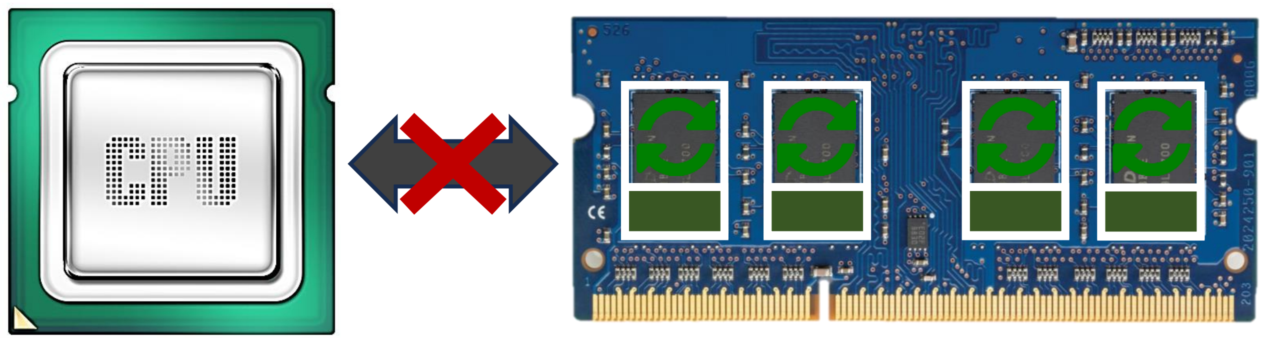
BreakHammer tracks the number
of RowHammer-preventive actions each thread triggers

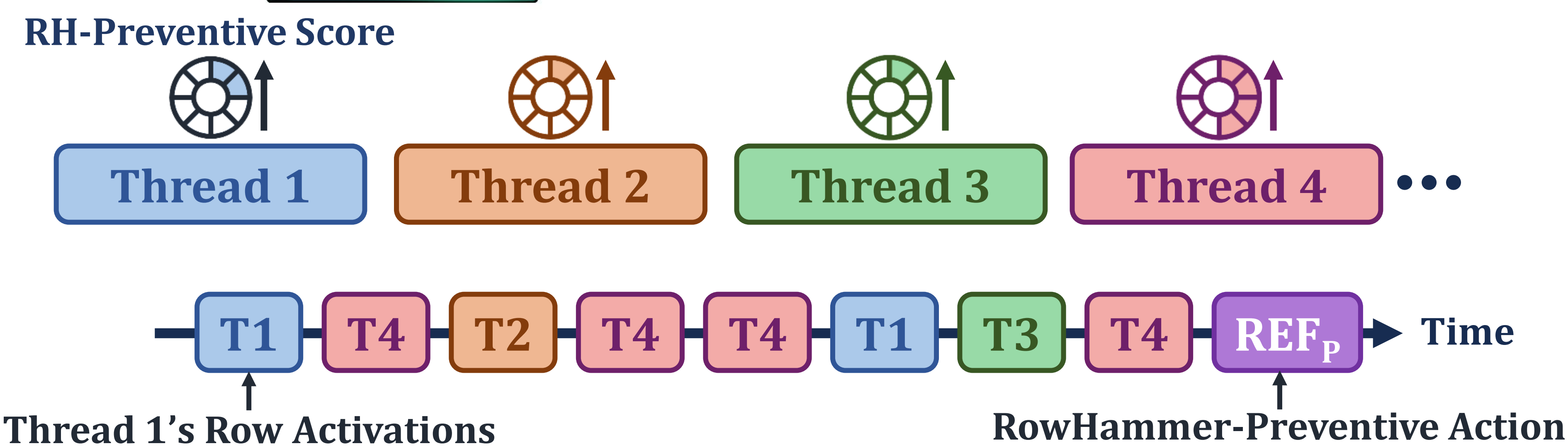


RowHammer-preventive score counter

Observing RowHammer-Preventive Actions: Score Attribution Method

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





Observing RowHammer-Preventive Actions: Integration Showcase

1) Probabilistic Row Activation (PARA)

2) Per Row Activation Counting (PRAC)

Observing RowHammer-Preventive Actions: Integration Showcase

1) Probabilistic Row Activation (PARA)

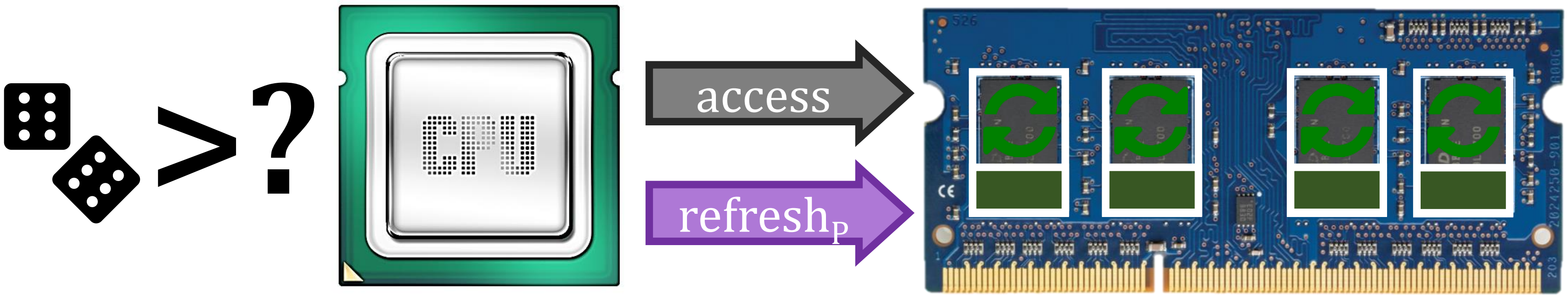
2) Per Row Activation Counting (PRAC)

Observing RowHammer-Preventive Actions: Integration Showcase with PARA (I)

BreakHammer cooperates with existing RowHammer solutions

Probabilistic Row Activation (PARA) [Kim+, ISCA 2024]:

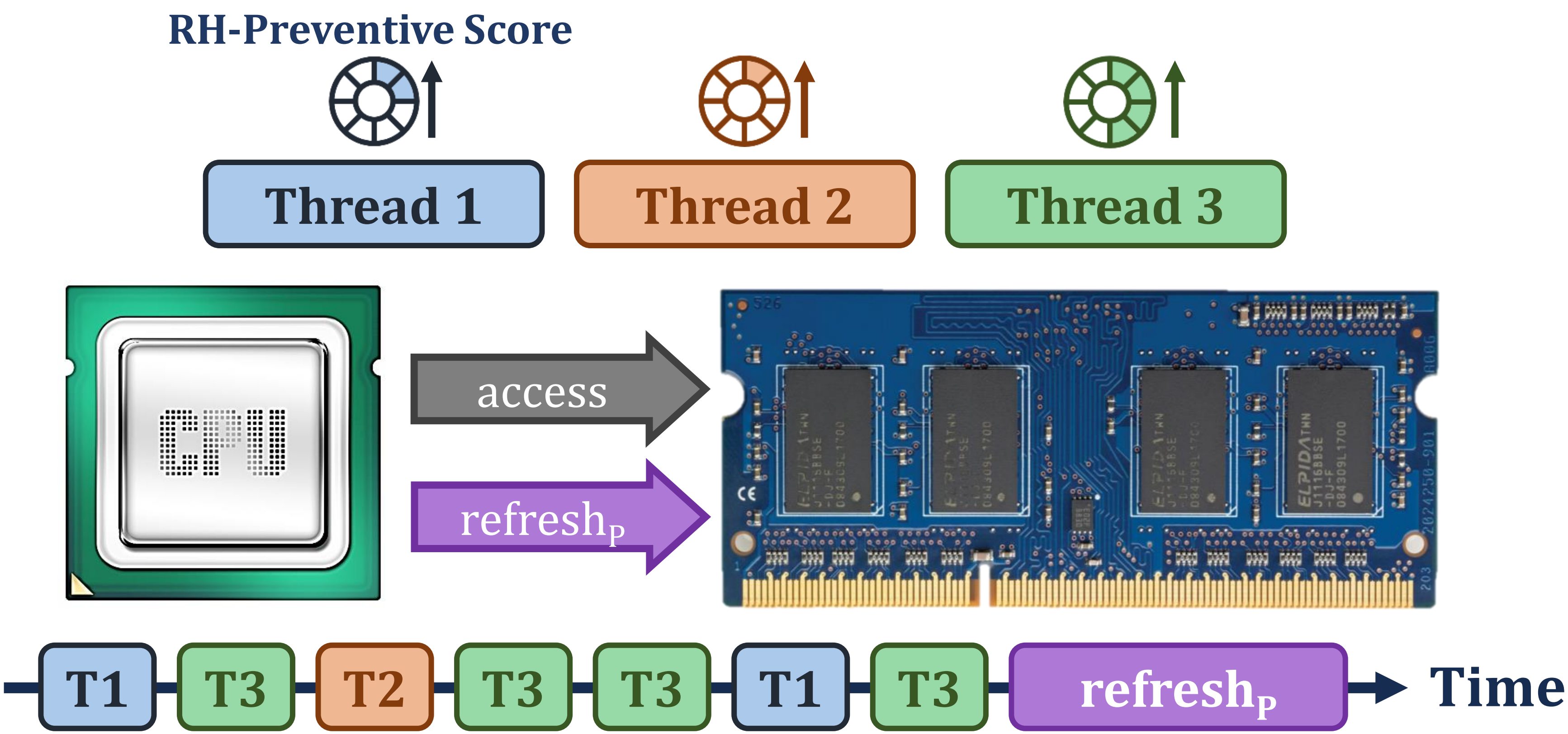
- **Generates** a random number
- Compares the number **with a threshold**
- If the random number exceeds the threshold **performs a preventive refresh**



Observing RowHammer-Preventive Actions: 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



Observing RowHammer-Preventive Actions: Integration Showcase

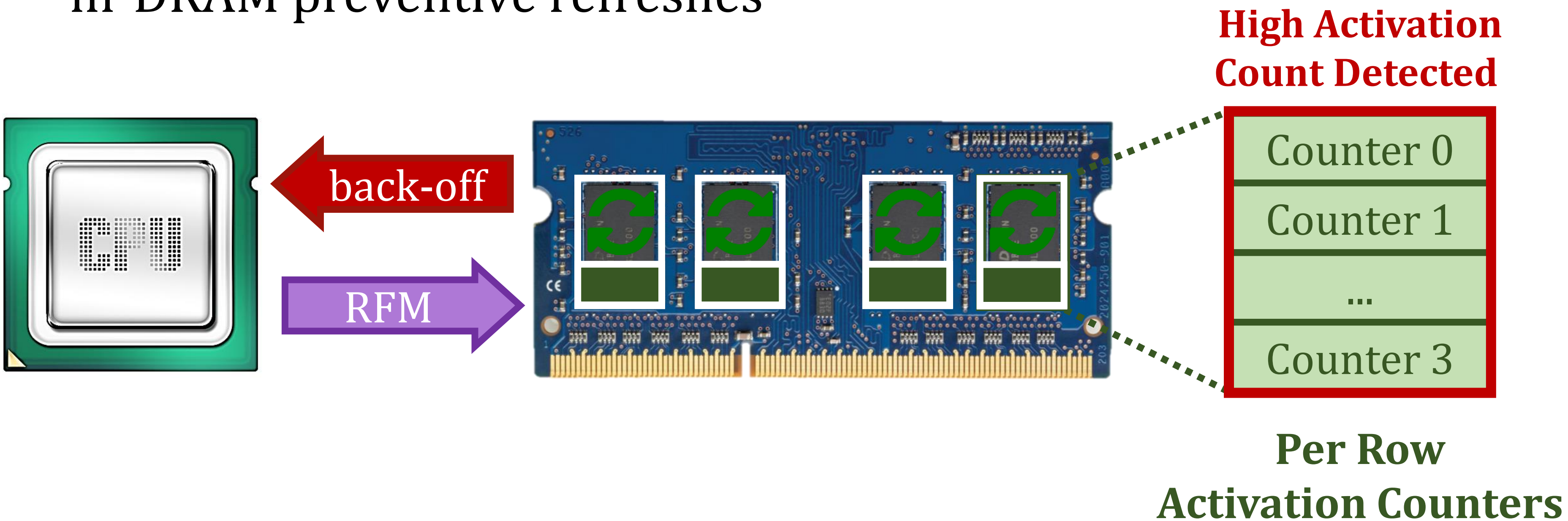
1) Probabilistic Row Activation (PARA)

2) Per Row Activation Counting (PRAC)

Observing RowHammer-Preventive Actions: Integration Showcase with PRAC (I)

BreakHammer cooperates with existing RowHammer solutions

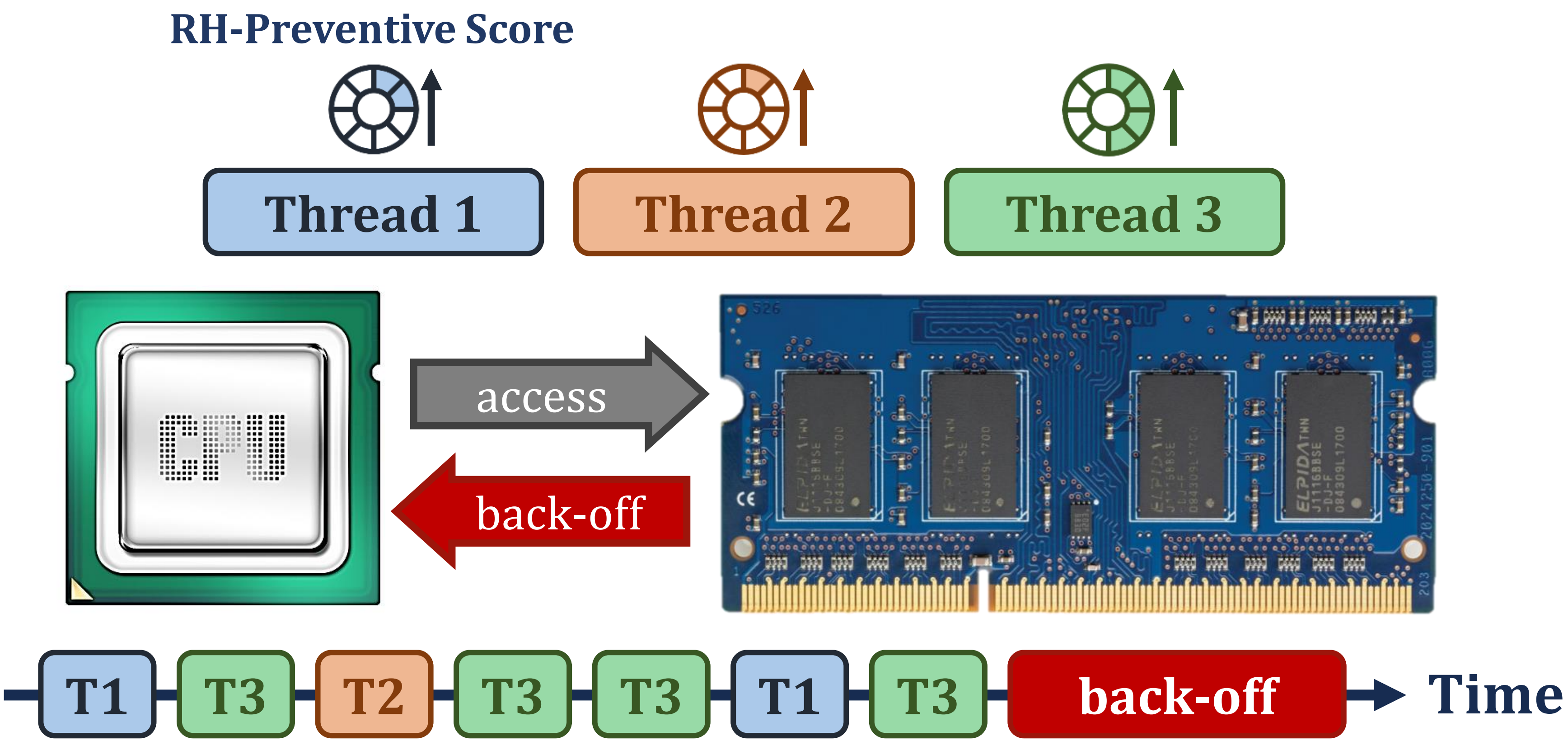
Per Row Activation Counting (PRAC) [JEDEC, 2024]:
- **DRAM** maintains an **activation counter for each DRAM row**
- **DRAM** requests time by triggering a **back-off**
- **Memory controller** **provides time** for in-DRAM preventive refreshes



Observing RowHammer-Preventive Actions: 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



Observing RowHammer-Preventive Actions: 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*

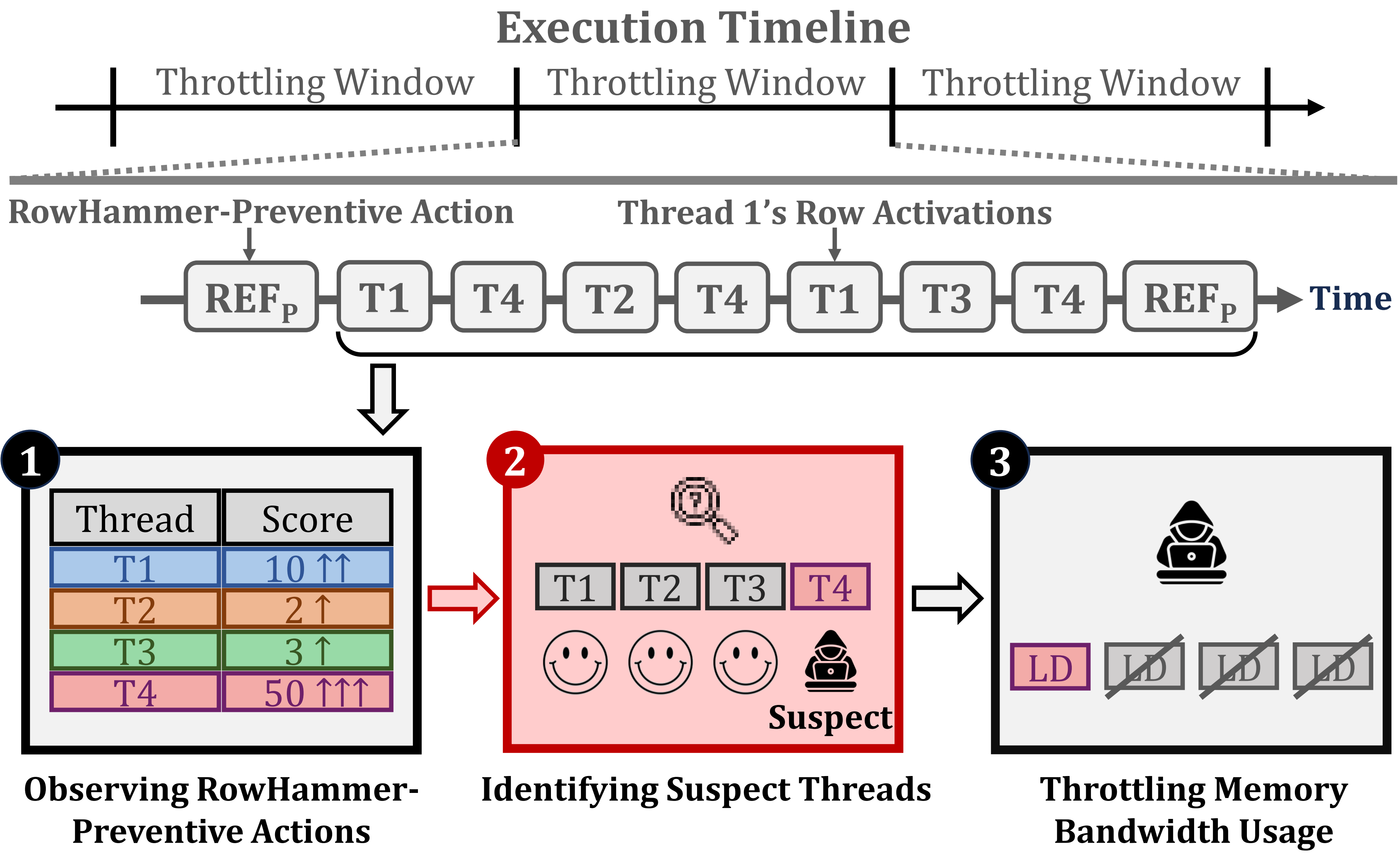
RowHammer is a major read disturbance mechanism in DRAM where repeatedly accessing (hammering) a row of DRAM cells (DRAM row) induces bitflips in other physically nearby DRAM rows. RowHammer solutions perform preventive actions (e.g.,

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

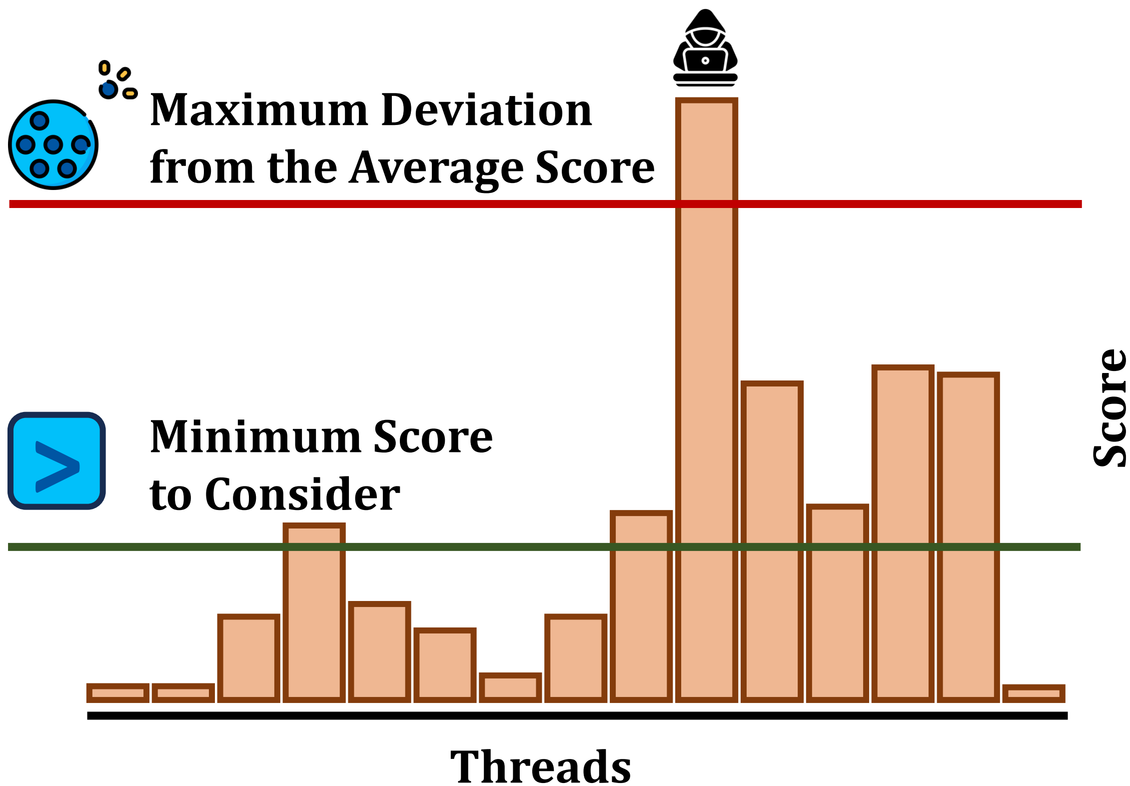
BreakHammer: Overview



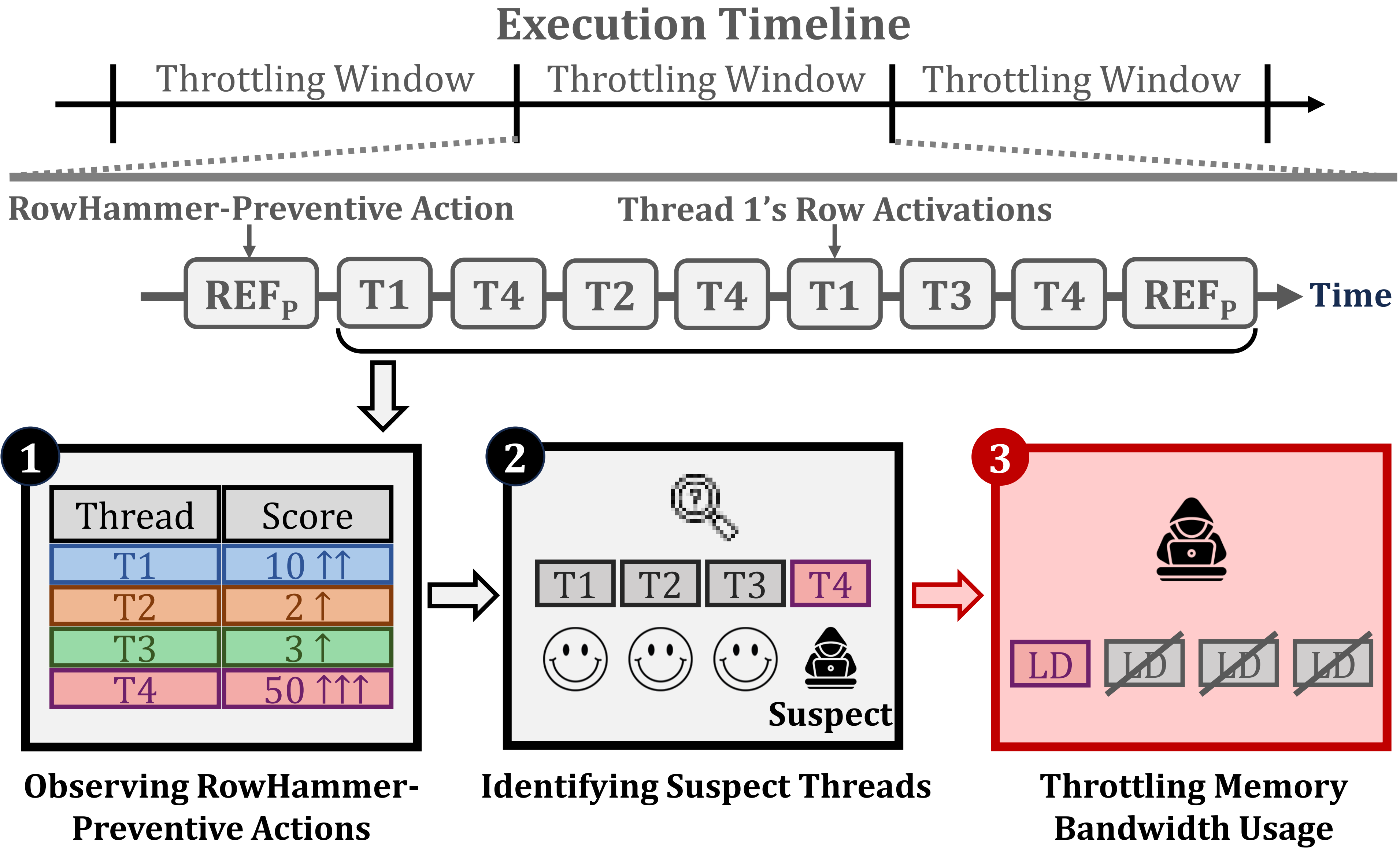
Identifying Suspect Threads: An Example

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

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

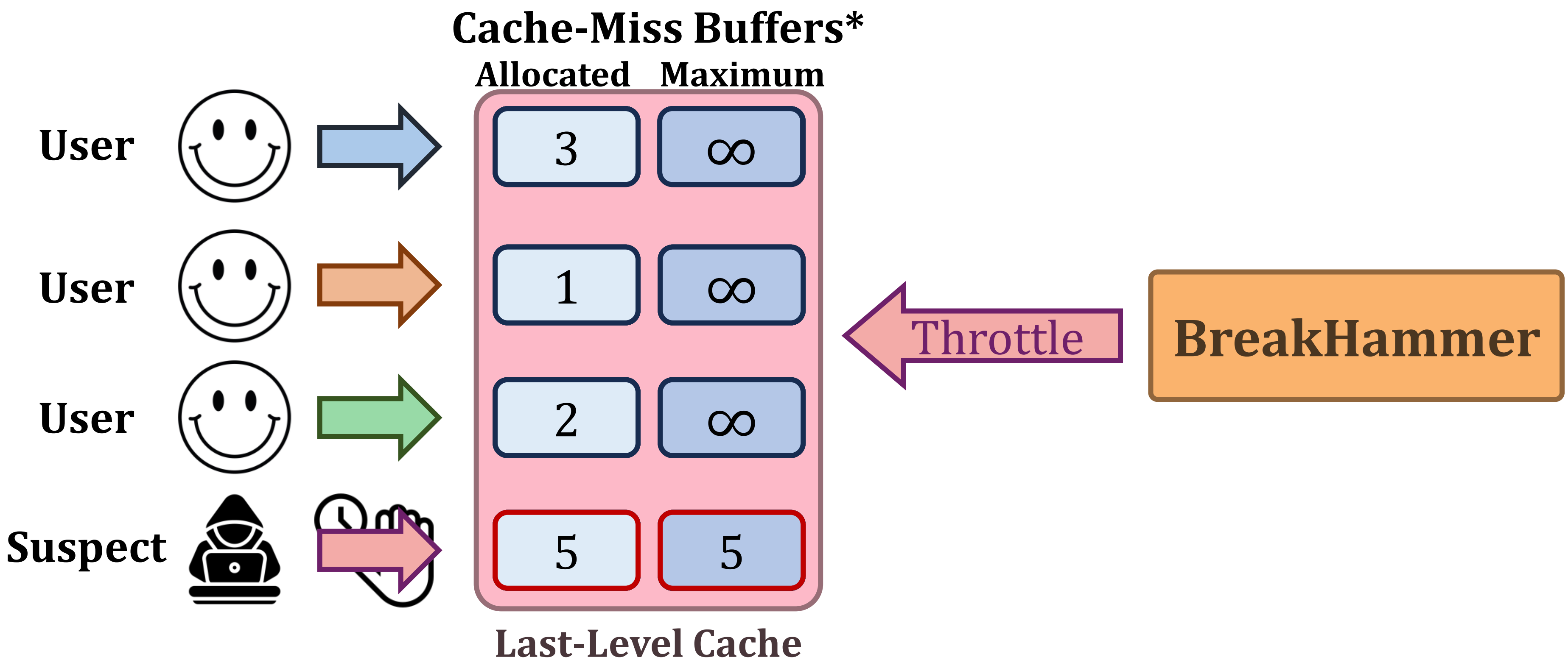


BreakHammer: Overview



Throttling Memory Bandwidth Usage of Suspect Threads

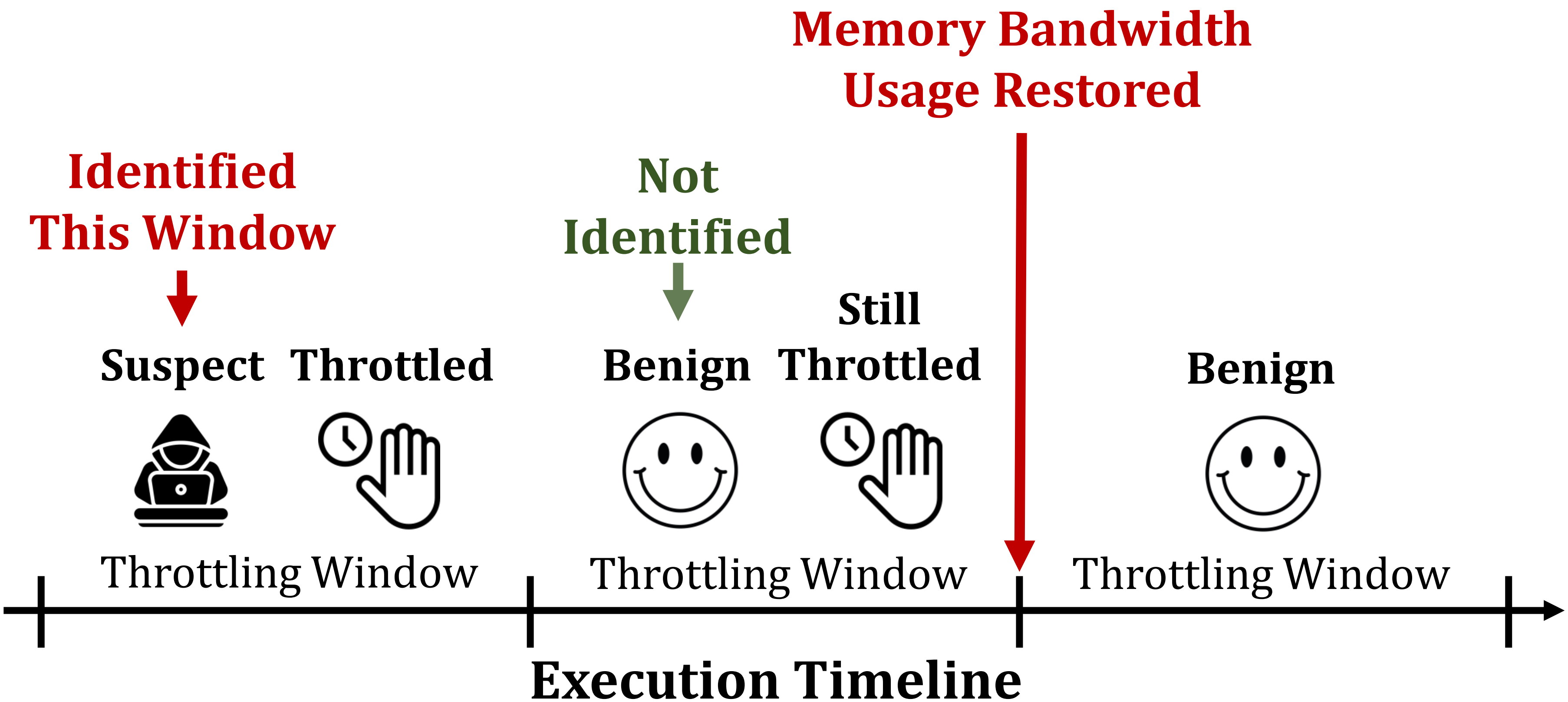
BreakHammer reduces the **memory bandwidth usage** of each **suspect thread**



[*] 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:** cycle-level simulations using Ramulator 2.0 [Luo+, CAL 2023] and DRAMPower [Chandrasekar+, DATE 2013]

- **System Configuration:**

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)

Evaluation Methodology

- **Comparison Points:** Integrated with 8 state-of-the-art RowHammer mitigation mechanisms:
  - **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, SPEC CPU2017, TPC, MediaBench, YCSB
  - 90 mixes with one attacker
  - 90 mixes all benign

Evaluation Results

1) Under Attack

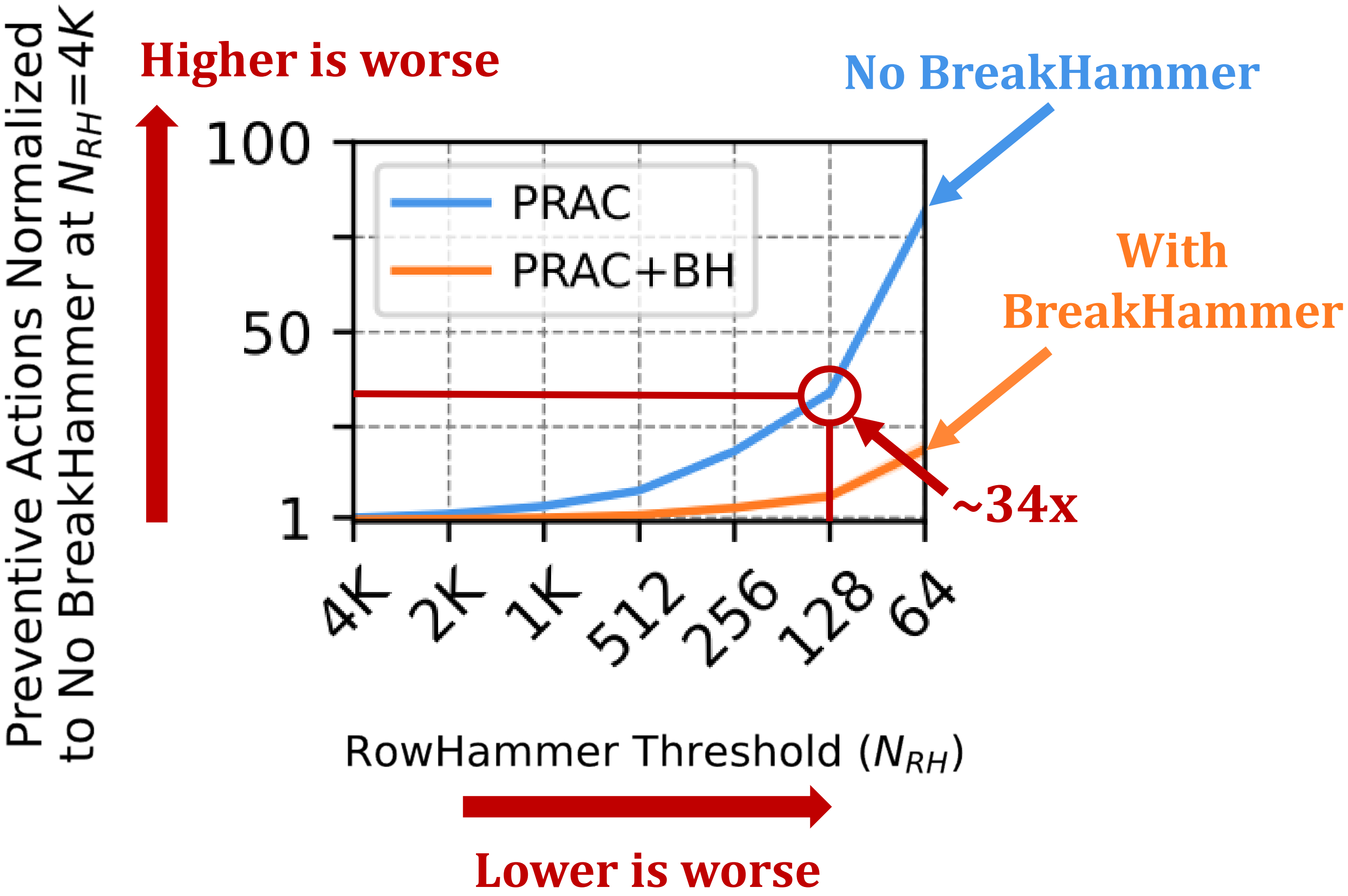
2) No Attack

Evaluation Results

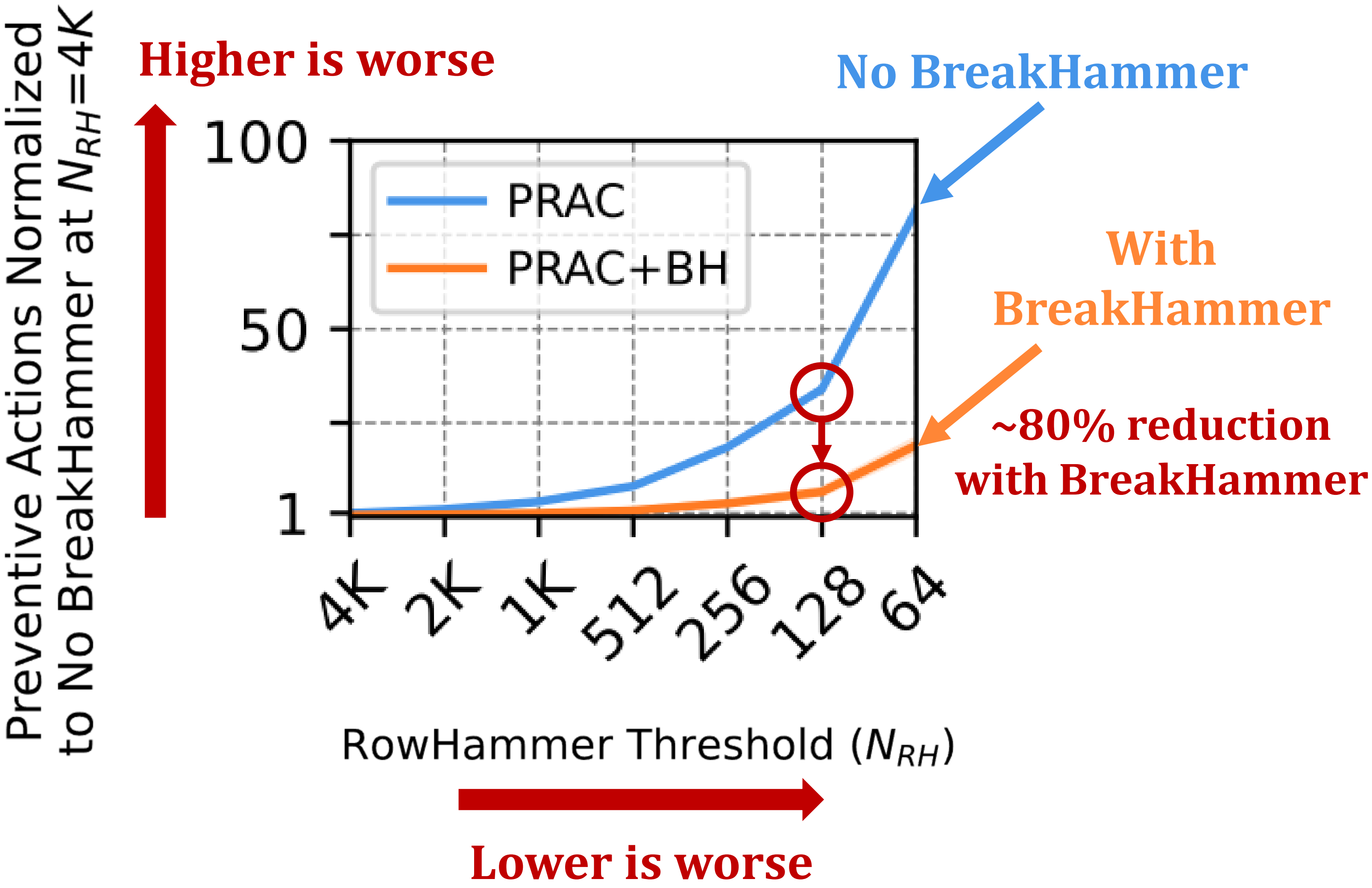
1) Under Attack

2) No Attack

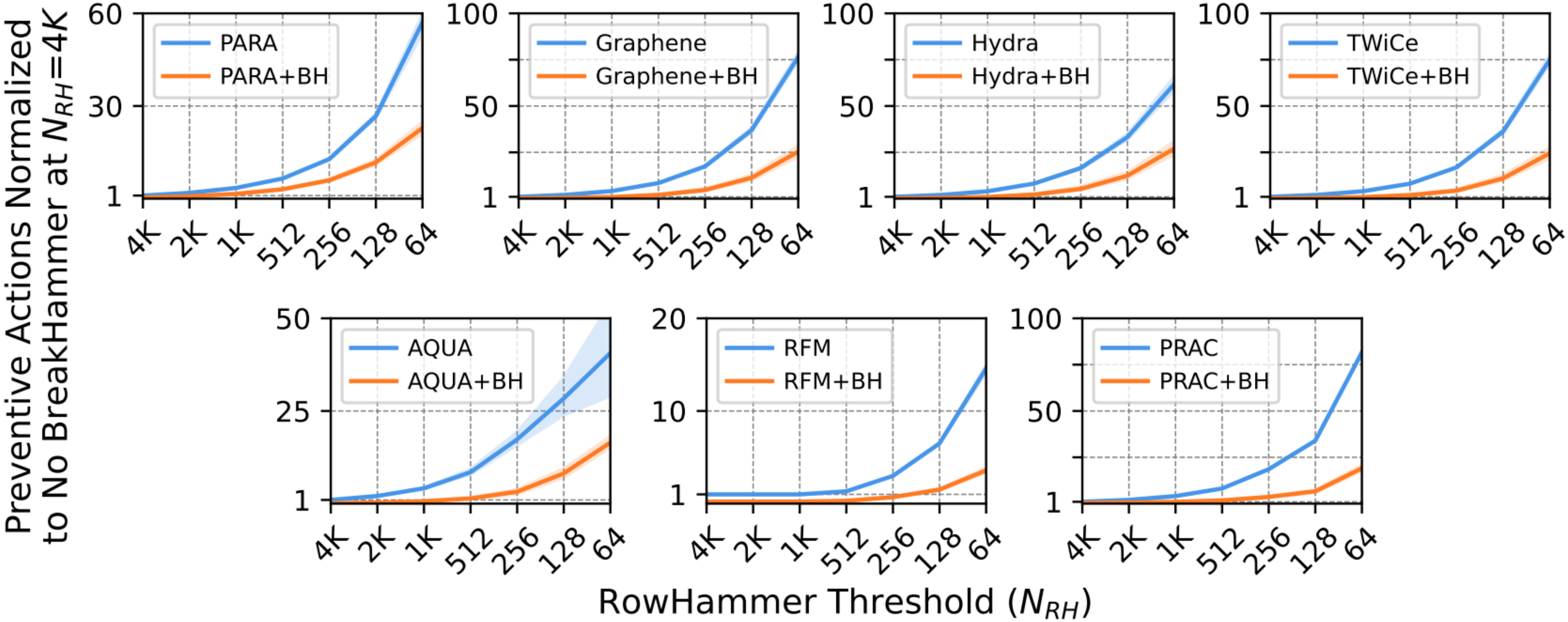
Preventive Action Count and Its Scaling



Preventive Action Count and Its Scaling

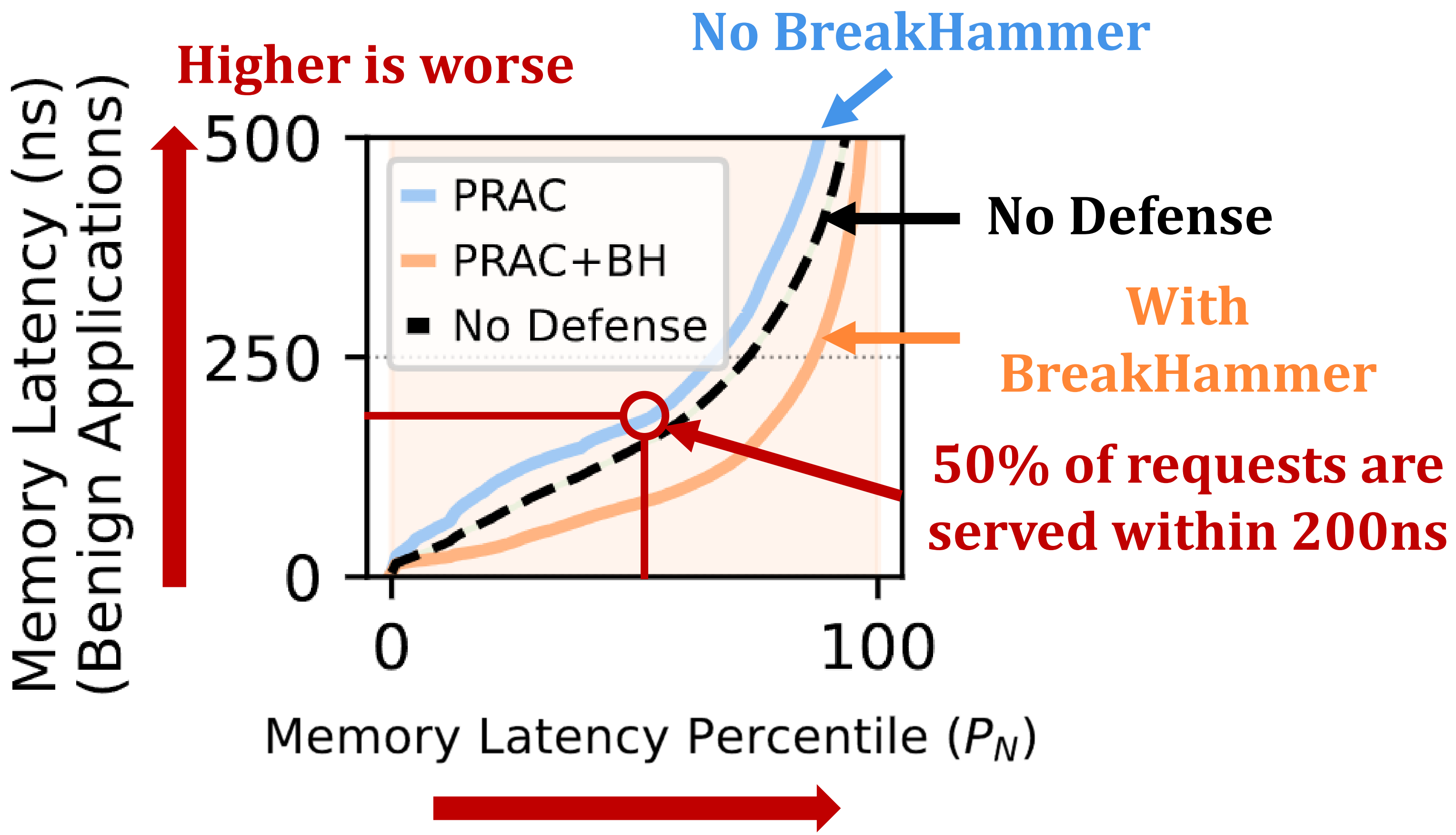


Preventive Action Count and Its Scaling

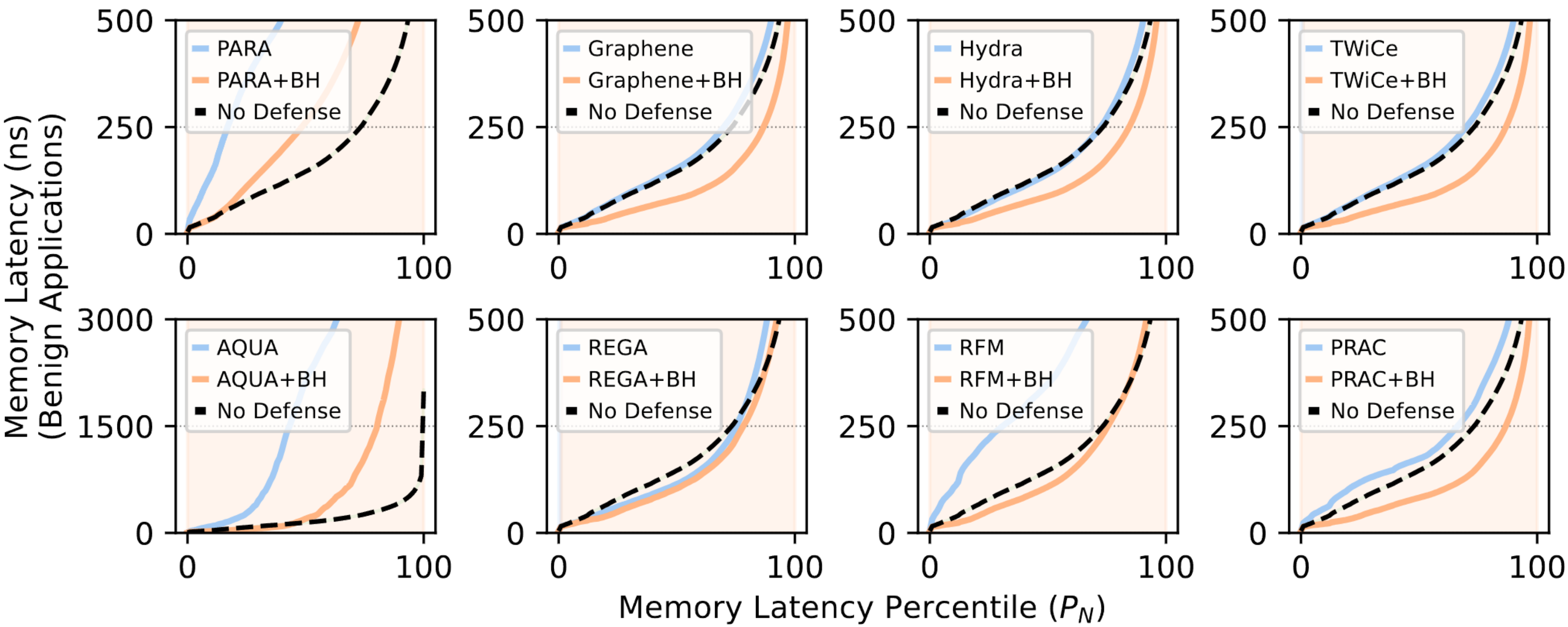


BreakHammer significantly reduces (72% on average) the number of preventive actions performed across all mechanisms

Memory Latency Impact at N_{RH}=64

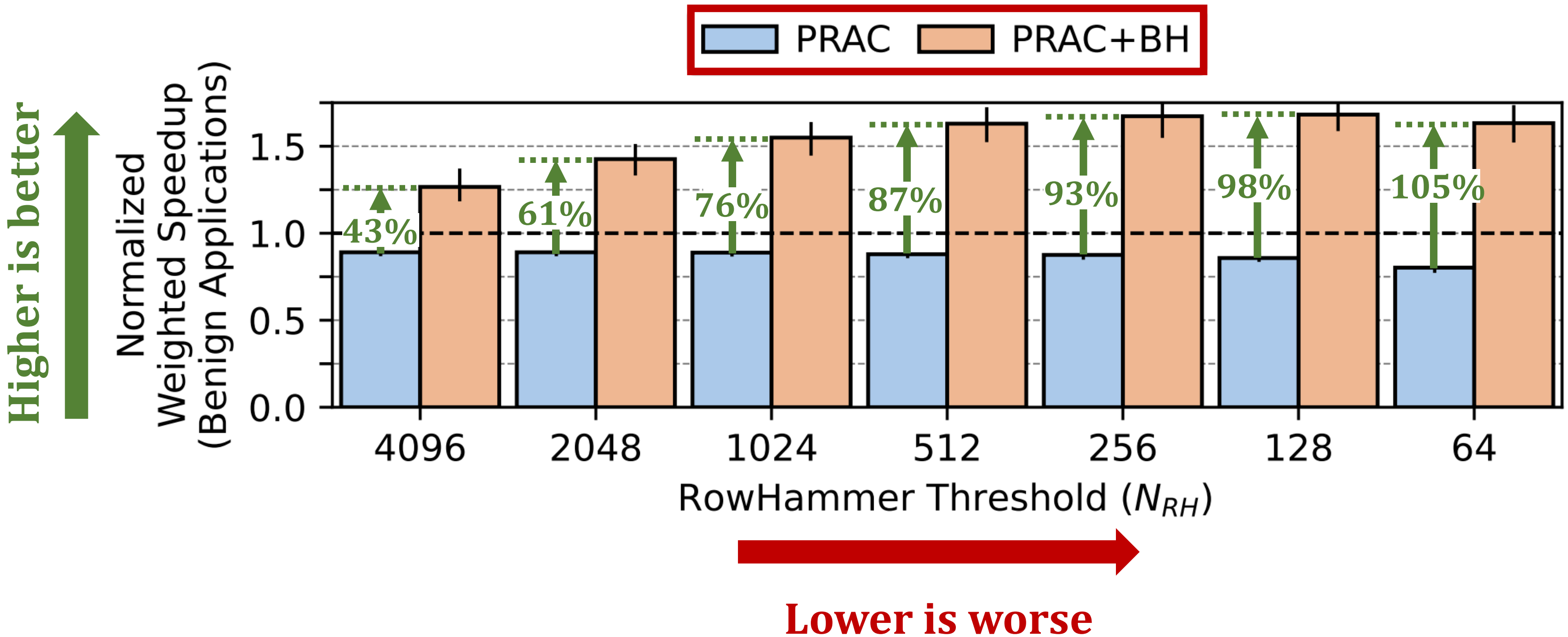


Memory Latency Impact at N_{RH}=64



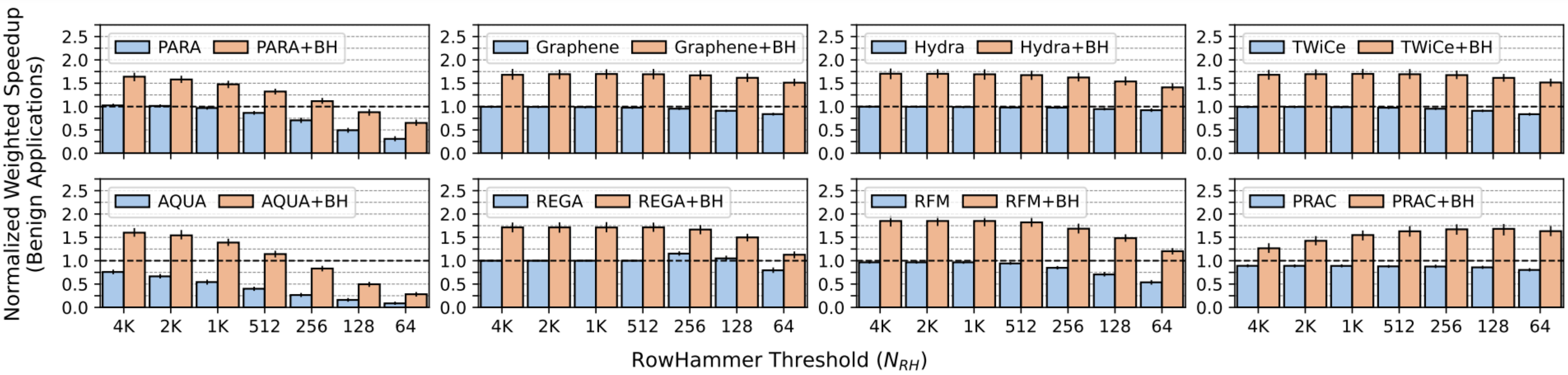
BreakHammer reduces memory latency across all mechanisms

Performance Impact and Its Scaling



BreakHammer significantly increases (81% on average) the performance of PRAC

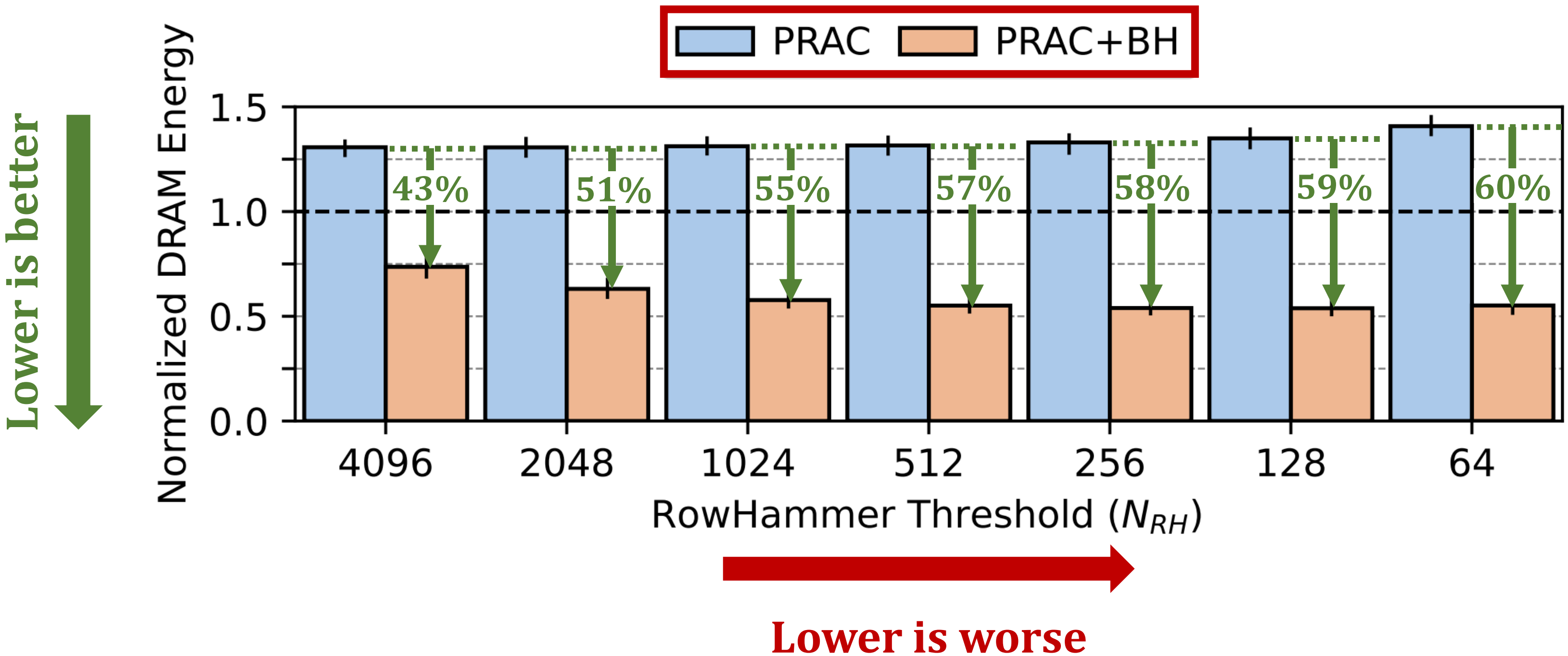
Performance Impact and Its Scaling



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

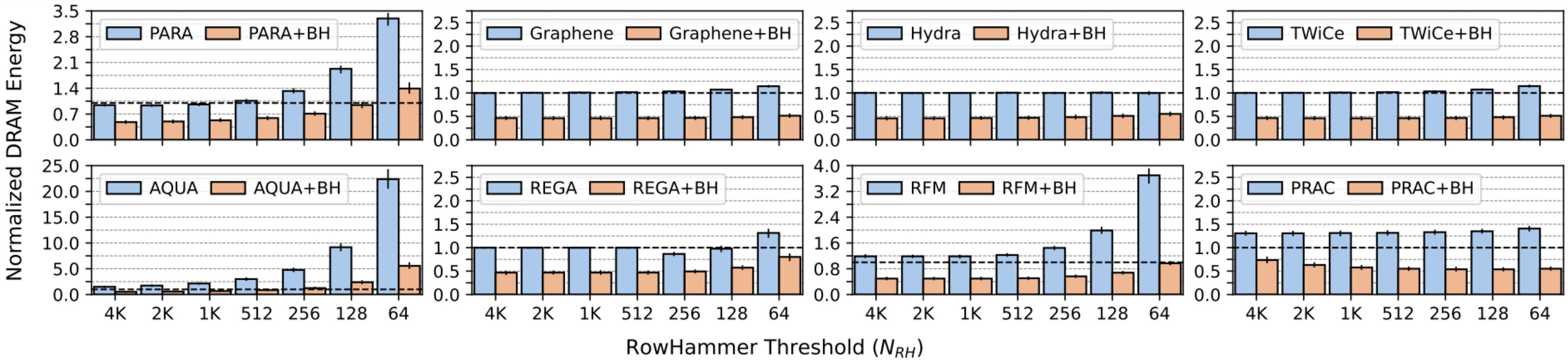
BreakHammer significantly increases system performance **(90% on average)**

DRAM Energy Impact and Its Scaling



BreakHammer significantly reduces (by 55% on average) the energy consumption of PRAC

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 **negative 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

https://arxiv.org/abs/2404.13477

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

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

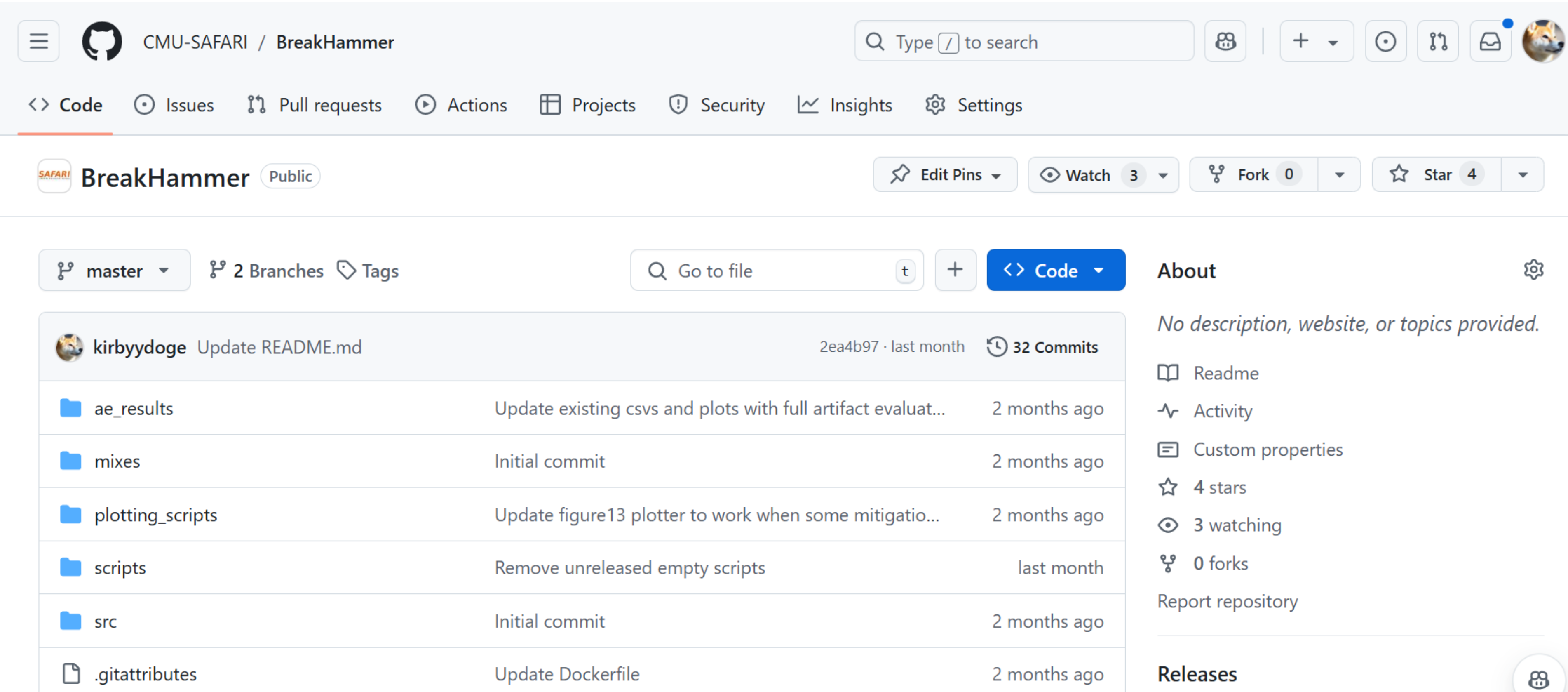
Key Mechanism: BreakHammer

- **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

Open Source and Artifact Evaluated



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

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

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

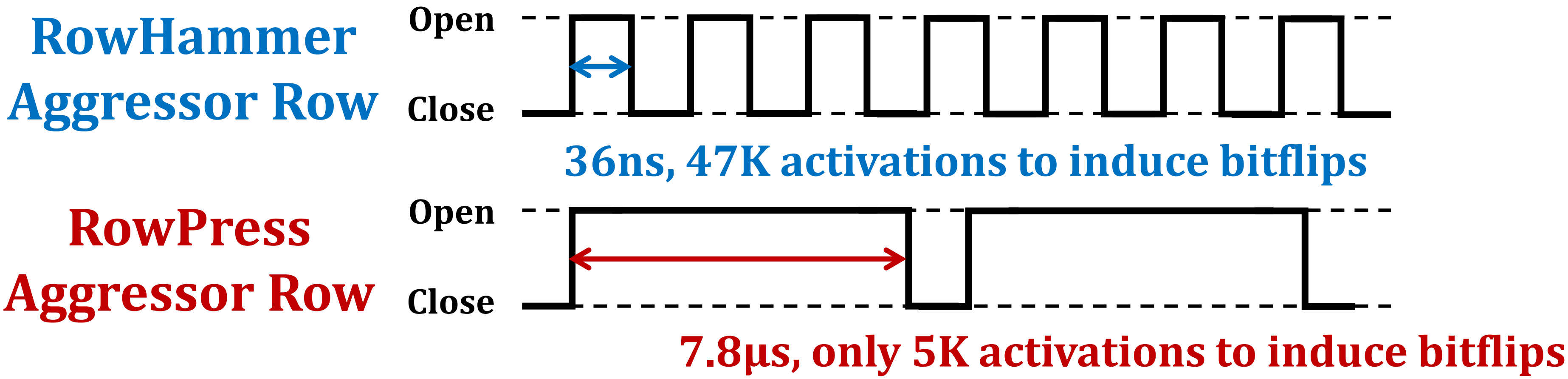
BreakHammer
Enhancing RowHammer Mitigations by Carefully Throttling Suspect Threads

BACKUP SLIDES

Oğuzhan Canpolat A. Giray Yağlıkçı
Ataberk Olgun İsmail E. Yüksel
Yahya C. Tuğrul Konstantinos Kanellopoulos
Oğuz Ergin Onur Mutlu

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

BreakHammer and RowPress

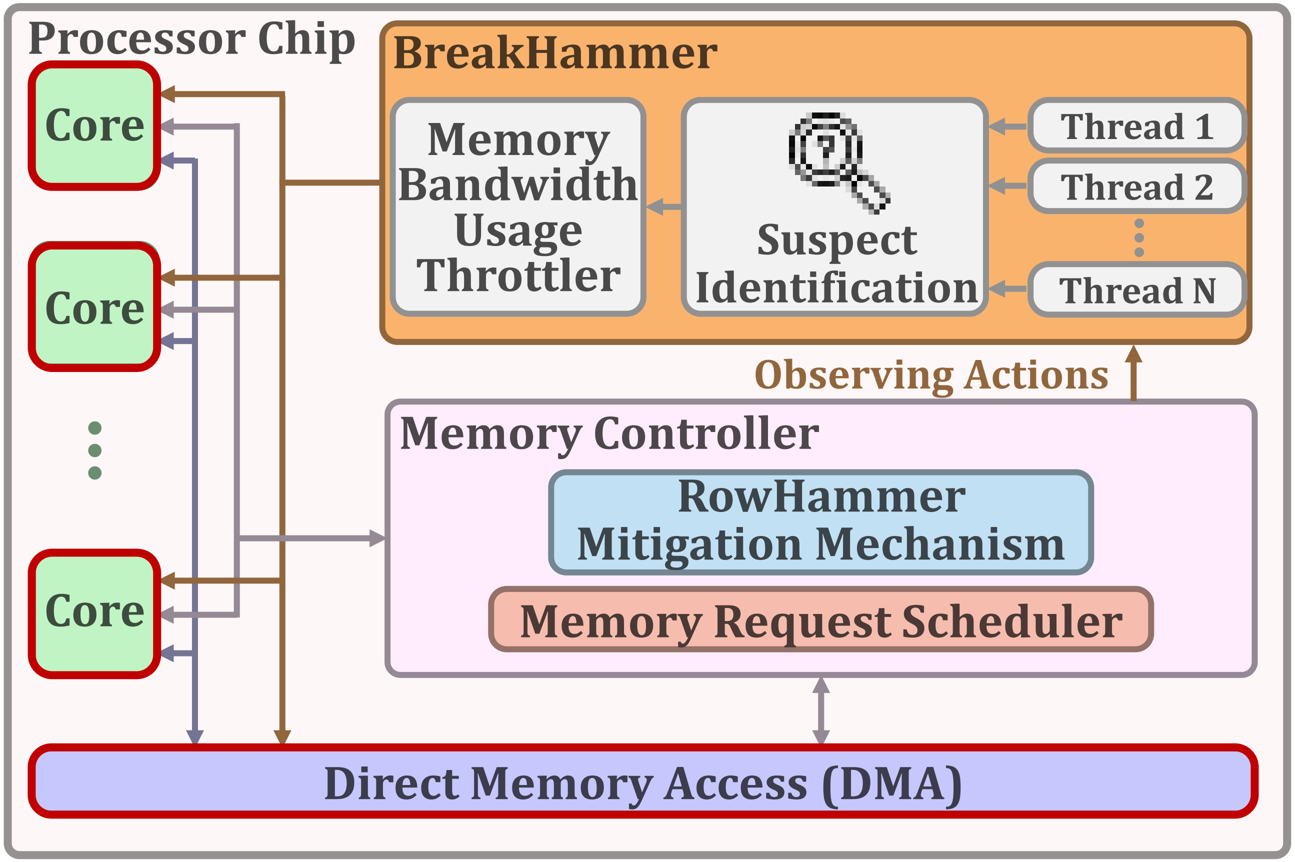


BreakHammer cooperates with a read disturbance solution

BreakHammer can become RowPress aware by:

1) changing the score attribution to consider row active time (e.g., Impress [Qureshi+, MICRO'24])
2) conveying the type of action taken by the read disturbance solution and tracking them differently (i.e., RowHammer or Rowpress-preventive action)

Throttling DMA and Systems without Caches

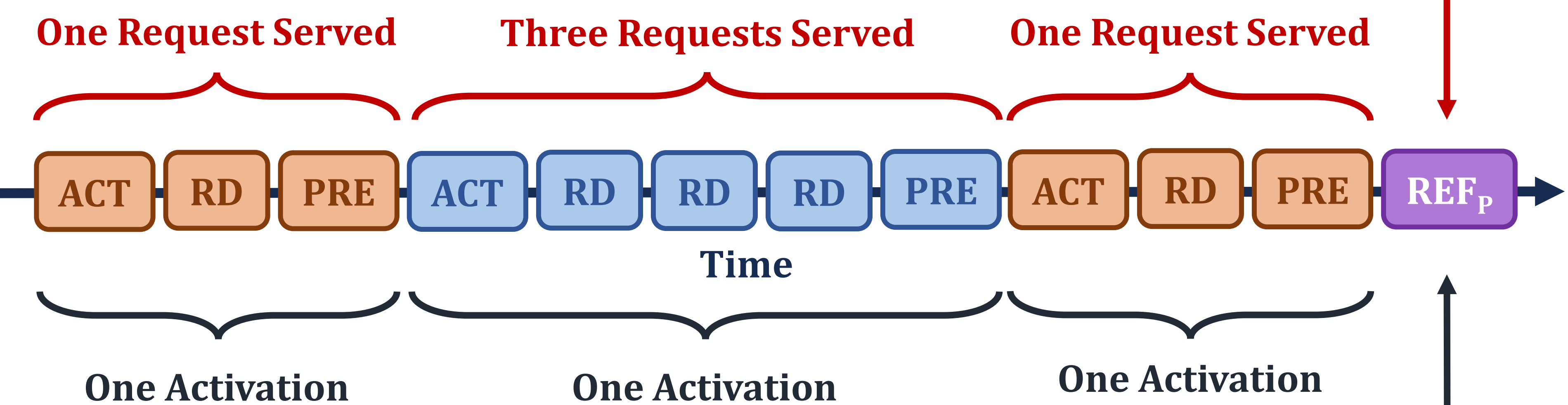


Extend **DMA** and **load-store units of cores**
to **track** and **limit** the number of **unresolved memory requests**

BreakHammer vs BLISS

BLISS only tracks consecutive requests served
Thereby wrongly scores a benign thread with higher score

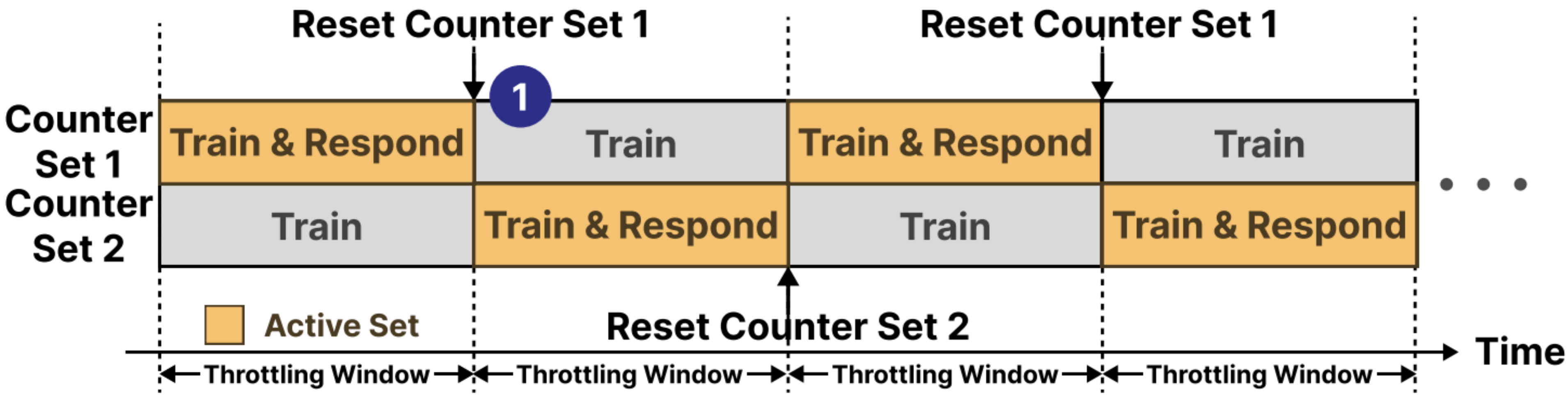
BLISS Is Oblivious To Action



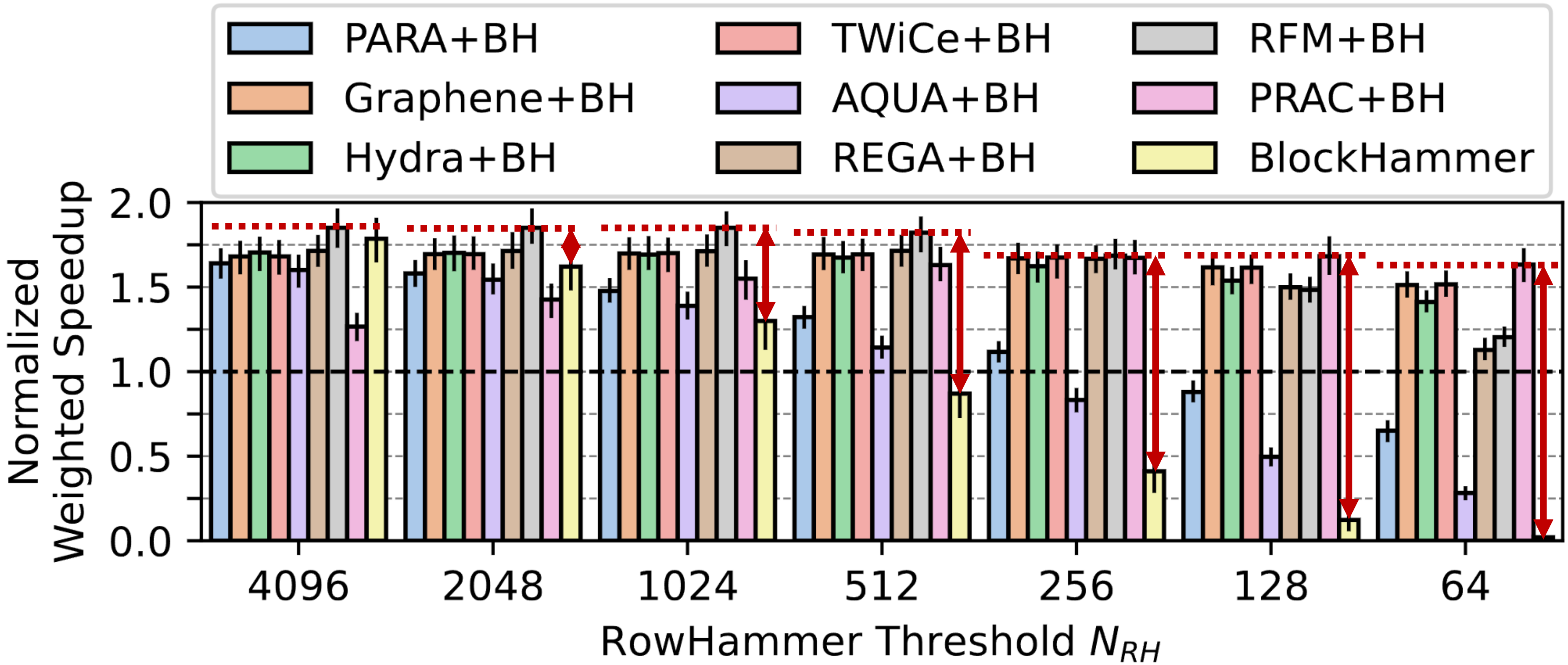
BreakHammer Three Key Operations Take Place

BreakHammer is preventive action contribution aware
Thereby accurately scores the suspect thread with higher score

Resetting Counters

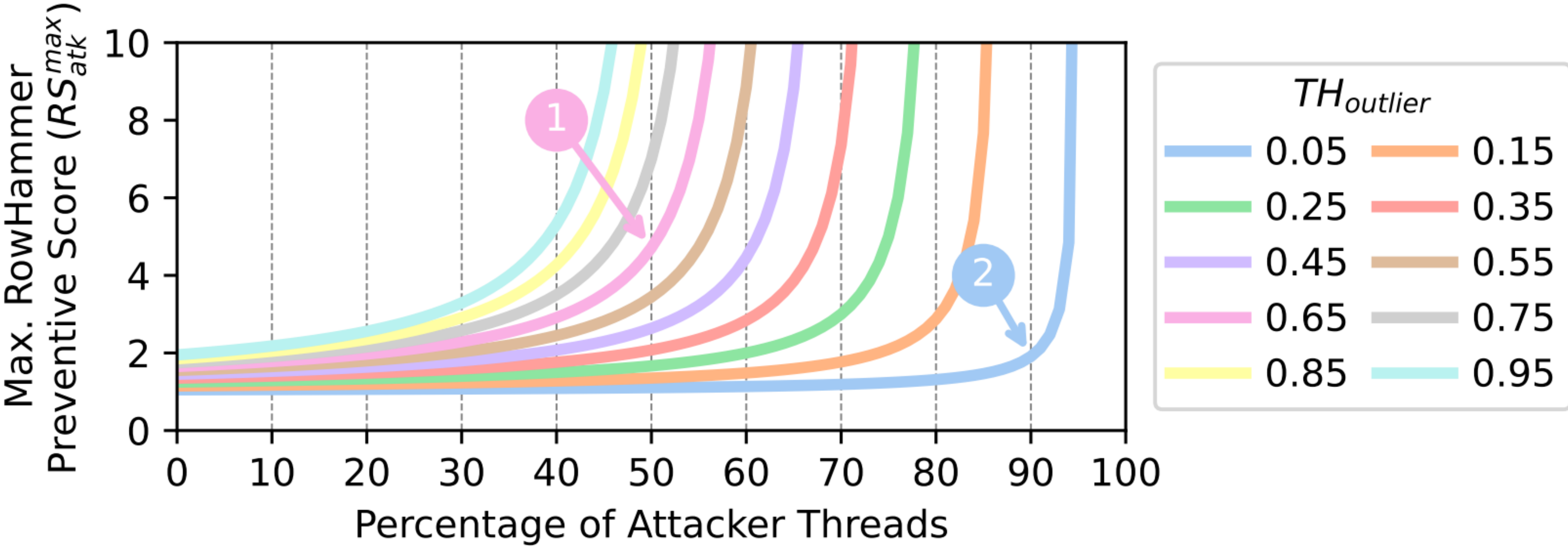


Comparison to BlockHammer



BreakHammer outperforms BlockHammer across all evaluated RowHammer thresholds

Upper Bound on the Overhead an Attacker Can Cause

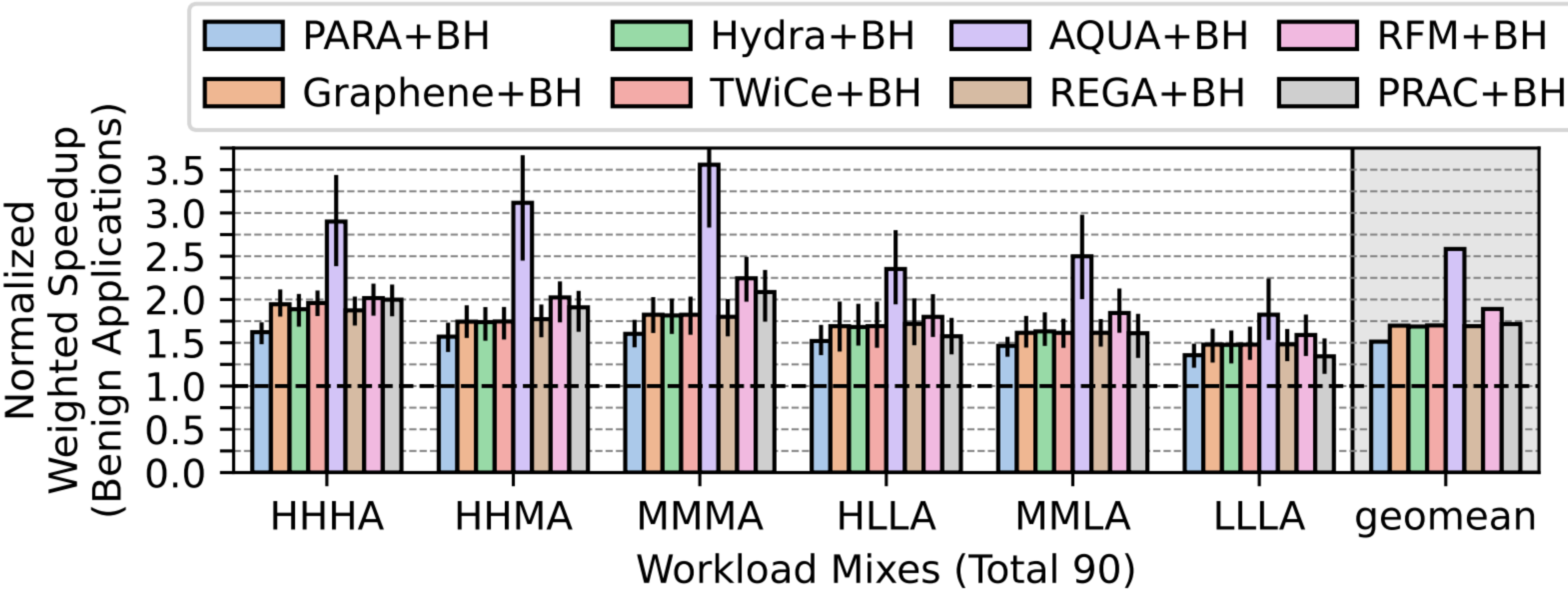


RBMPKI and Repeatedly Activated Row Count

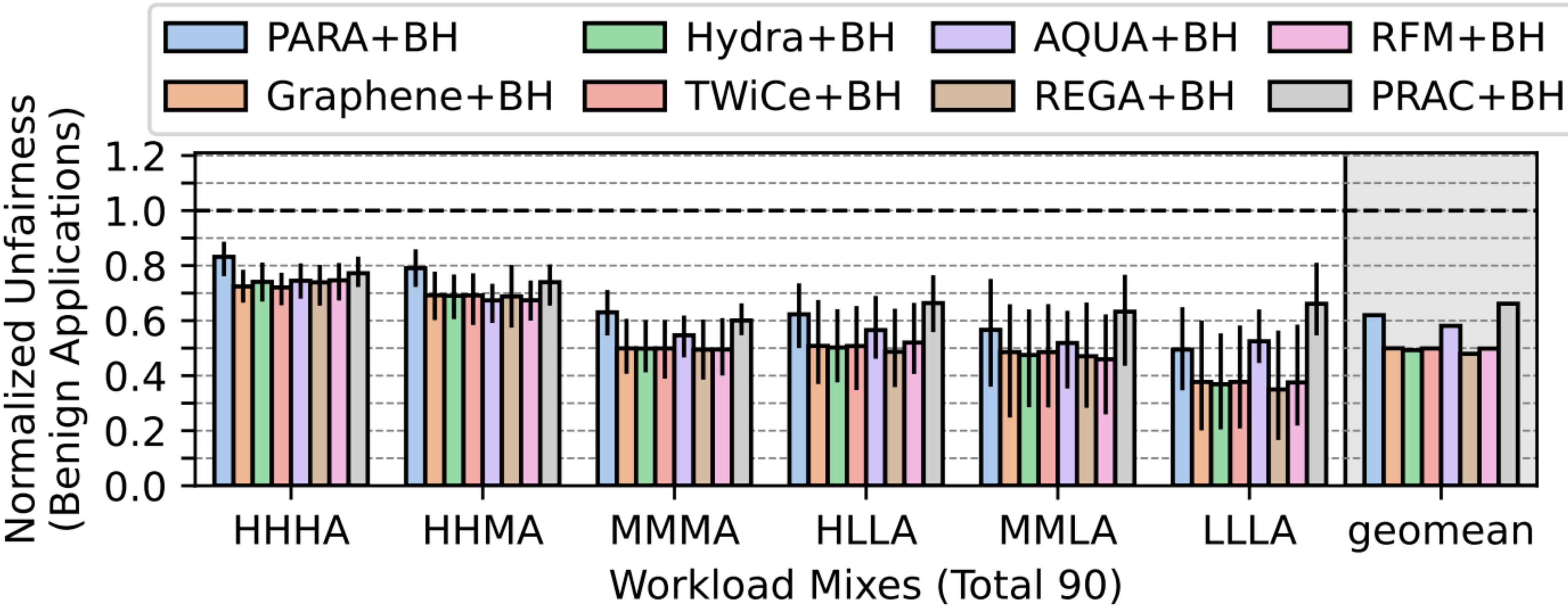
Table 3: Workload Characteristics: RBMPKI and Average Number of Rows with More Than 512+, 128+, and 64+ Activations per 64ms Time Window

Workload	RBMPKI	ACT-512+	ACT-128+	ACT-64+
429.mcf	68.27	2564	2564	2564
470.lbm	28.09	664	6596	7089
462.libquantum	25.95	0	0	1
549.fotonik3d	25.28	0	88	10065
459.GemsFDTD	24.93	0	218	10572
519.lbm	24.37	2482	5455	5824
434.zeusmp	22.24	292	4825	11085
510.parest	17.79	94	185	803
Average	29.615	762	2491	6000

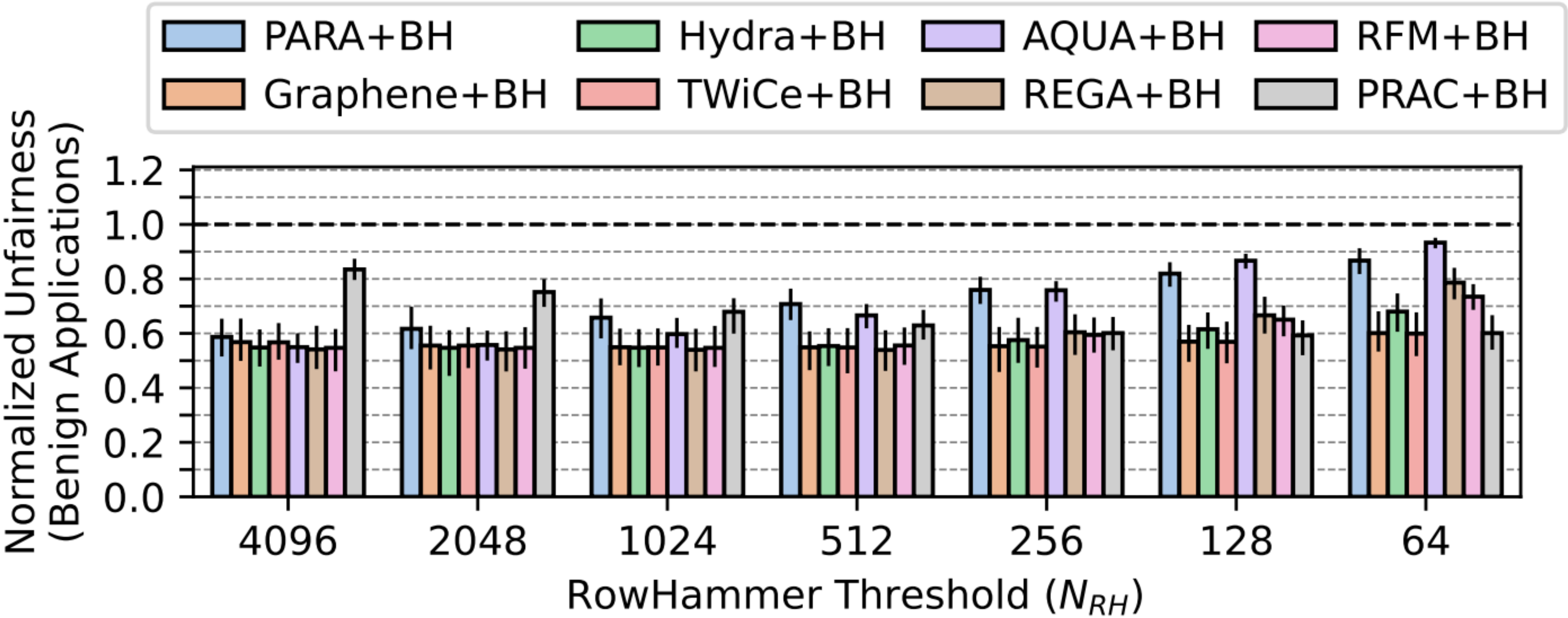
Under Attack Memory Intensity (N_{RH}=1K)



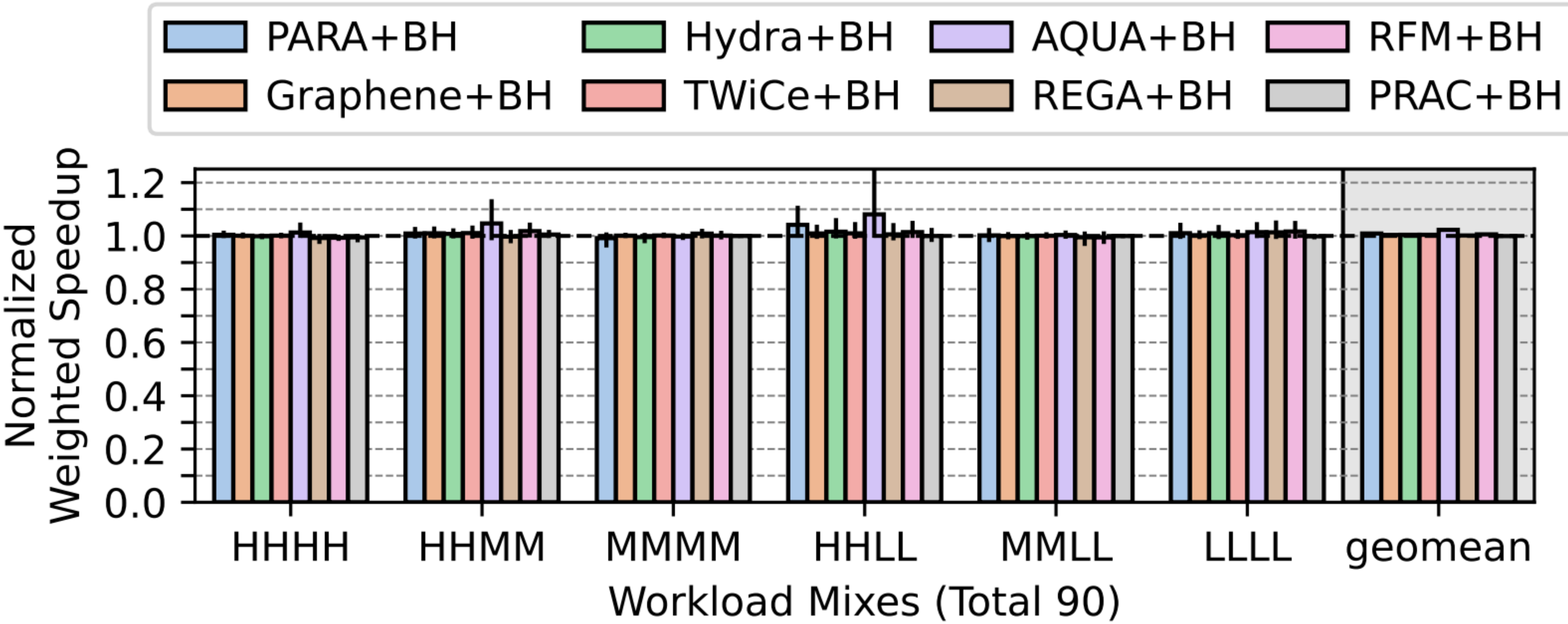
Under Attack Unfairness (N_{RH}=1K)



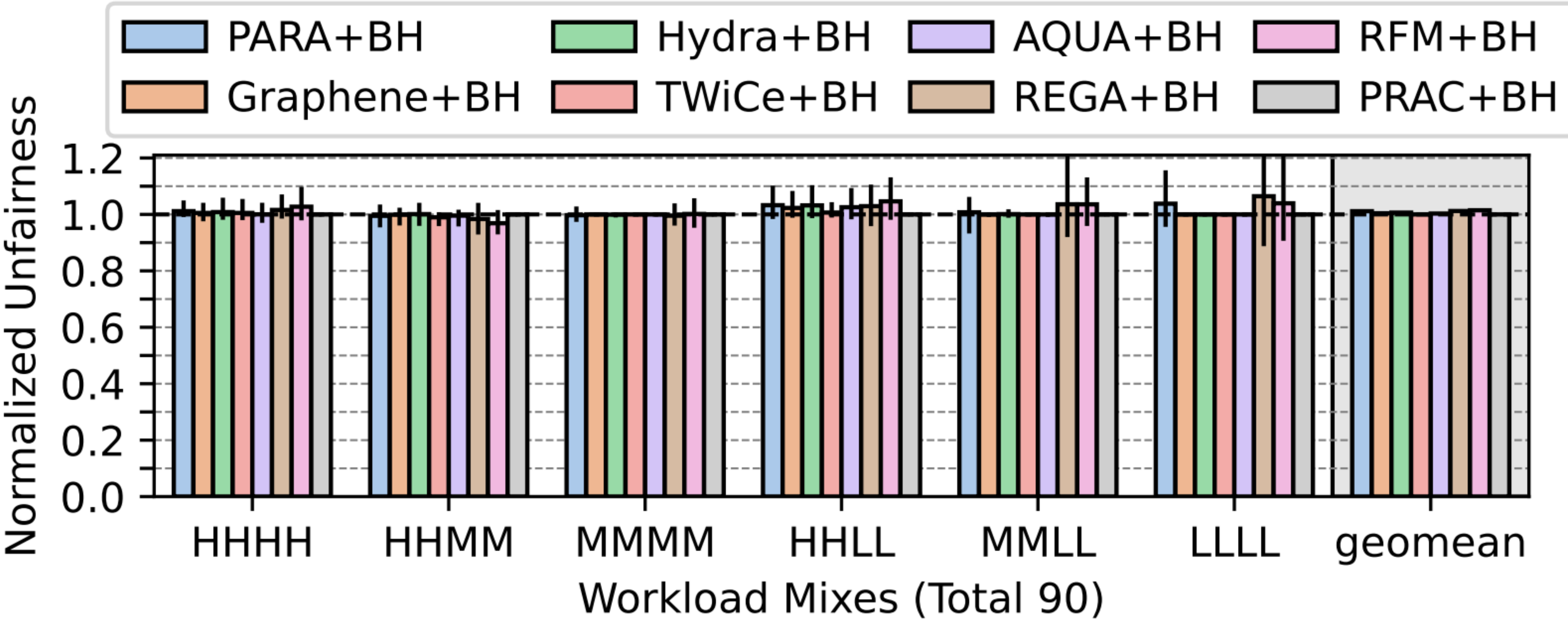
Under Attack Unfairness and Its Scaling



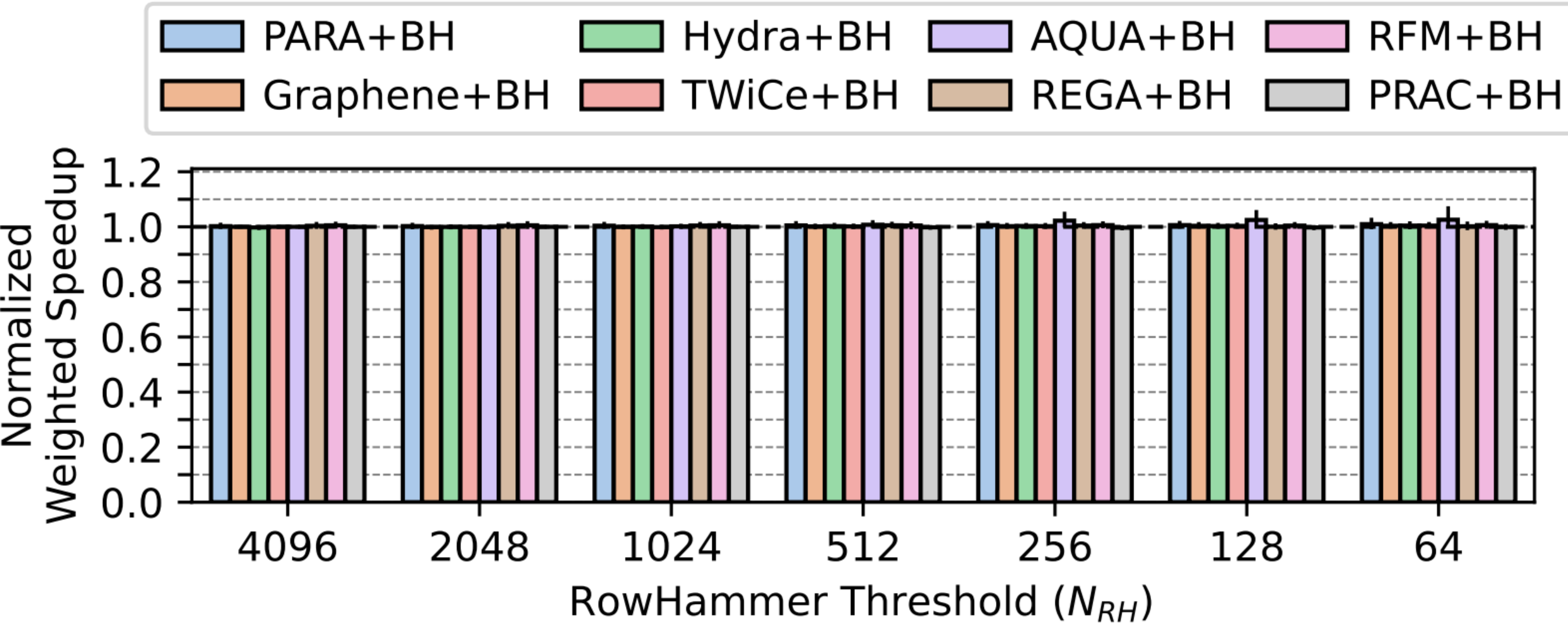
No Attack Memory Intensity (N_{RH}=1K)



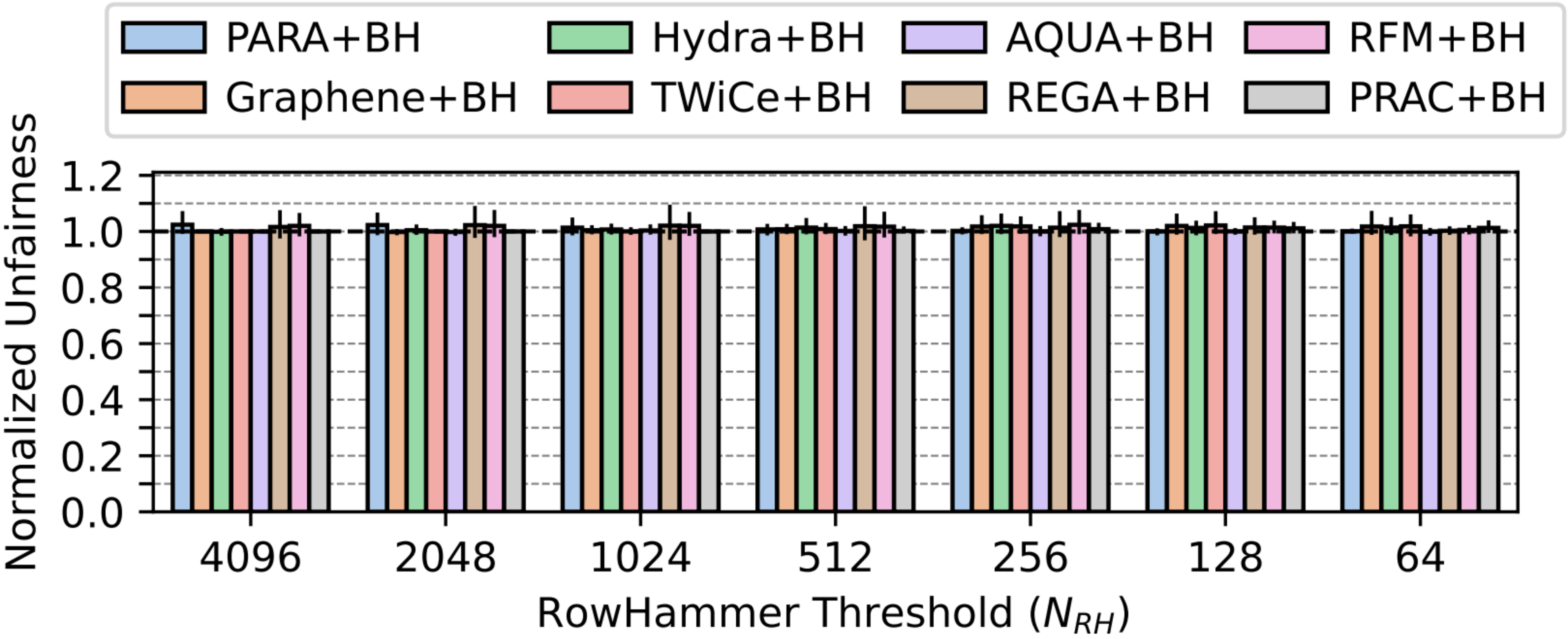
No Attack Unfairness (N_{RH}=1K)



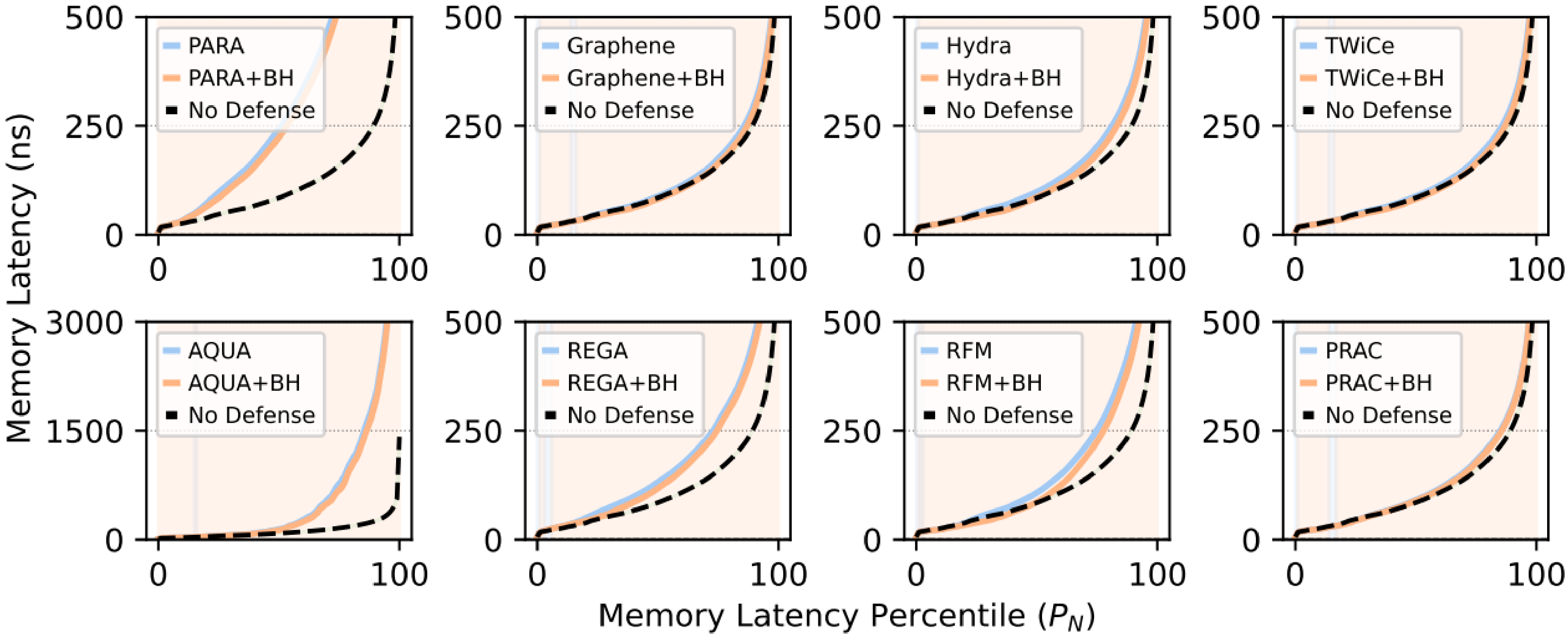
No Attack Performance and Its Scaling



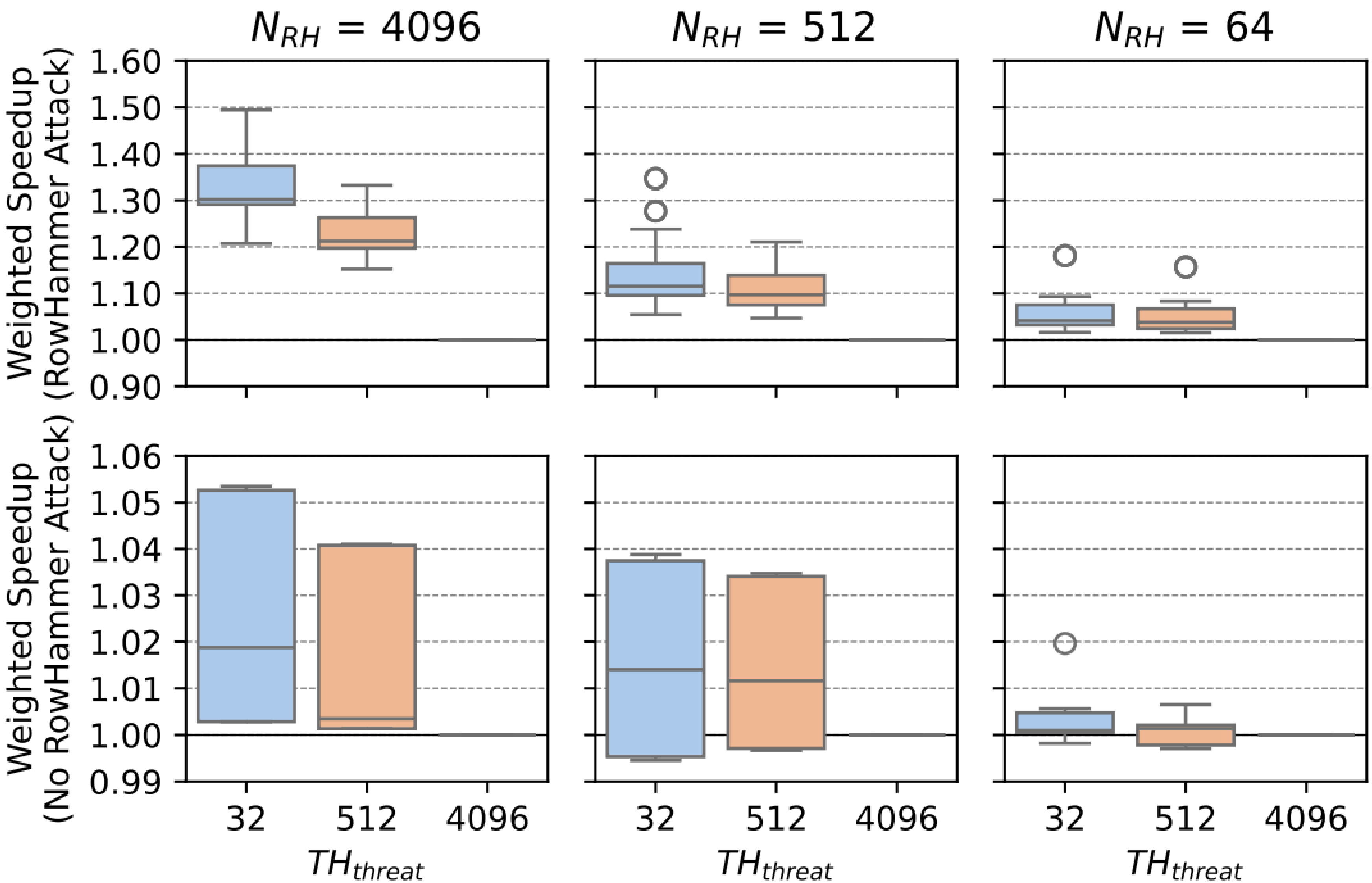
No Attack Unfairness and Its Scaling



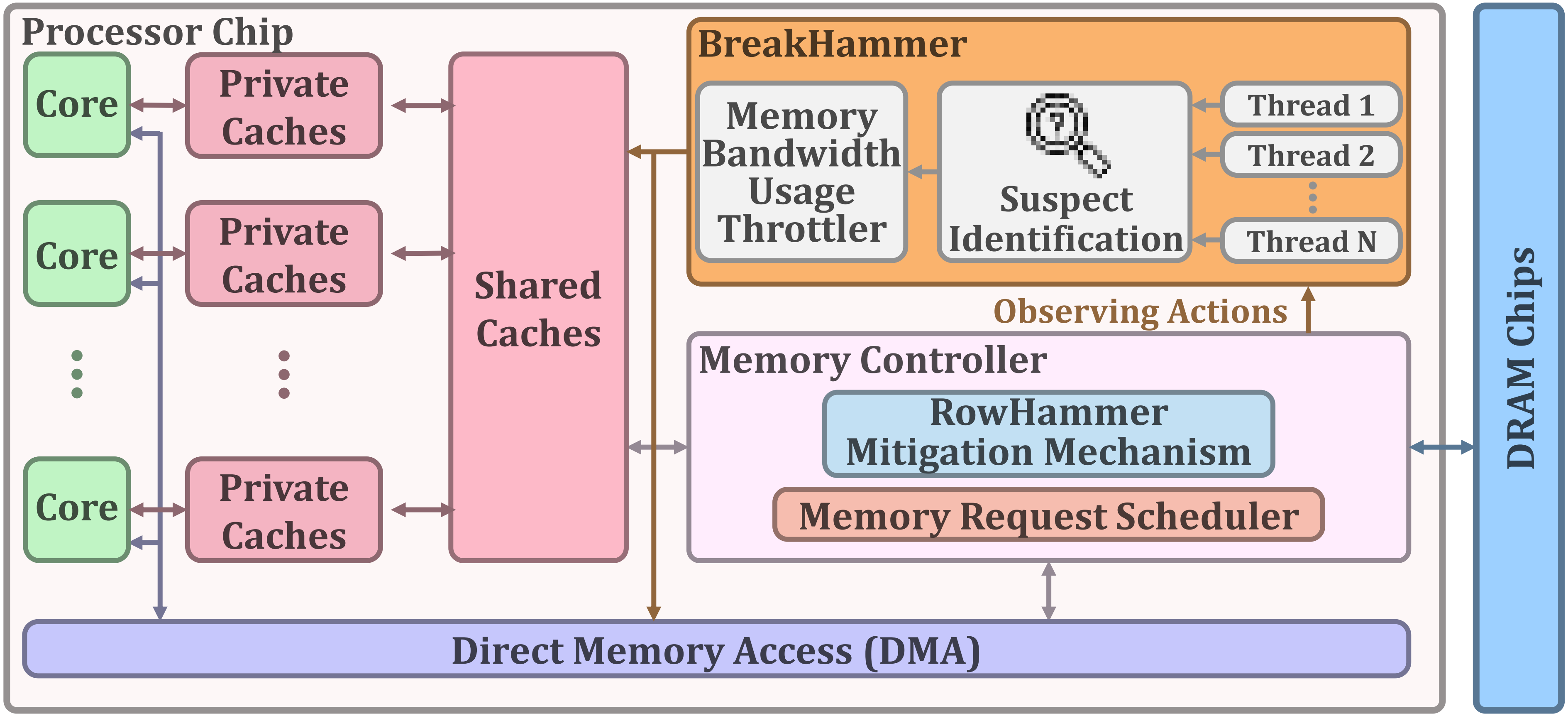
No Attack Memory Latency (N_{RH}=64)



BreakHammer Sensitivity to Minimum Score

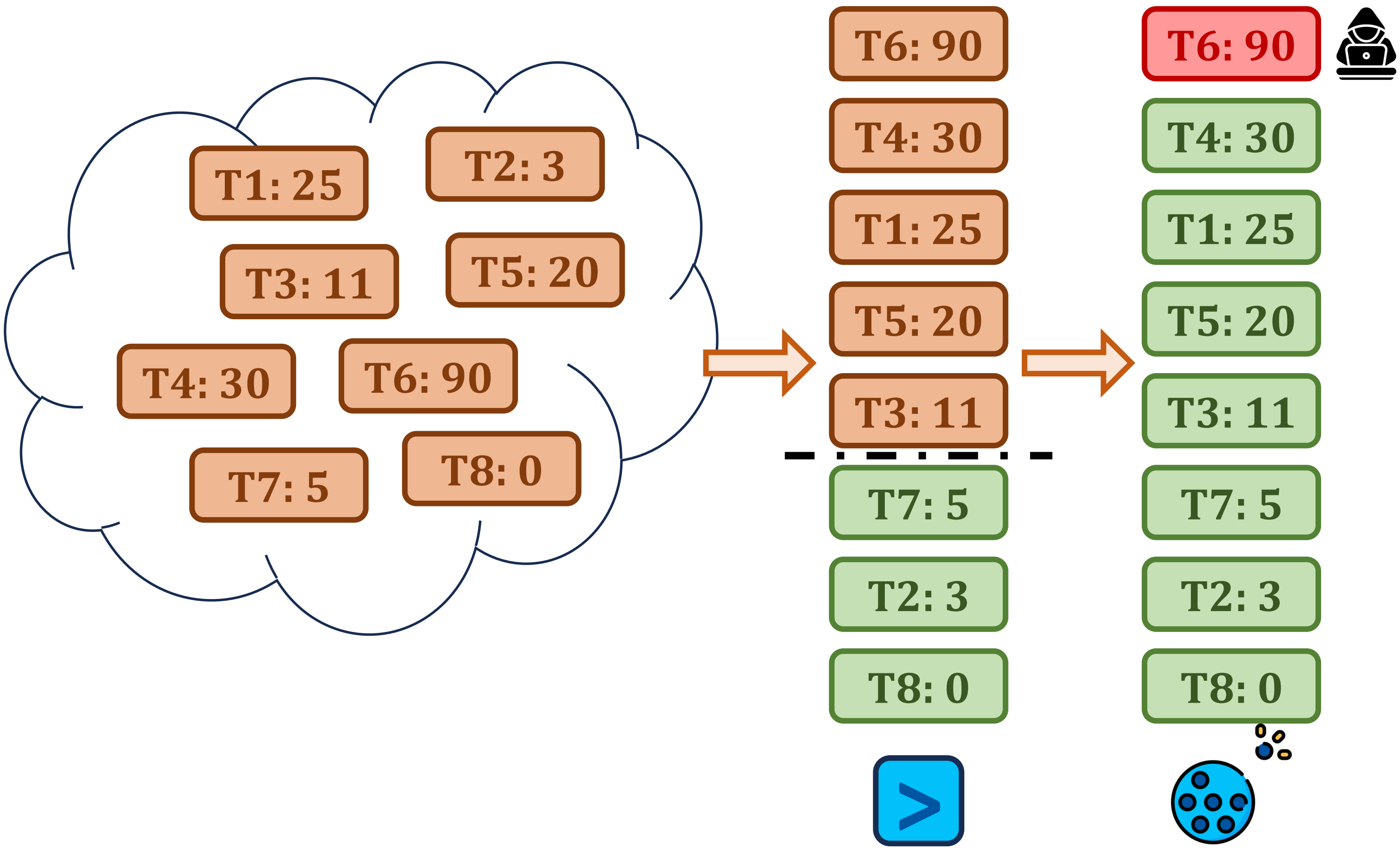


Organization



Identifying Suspect Threads: An Example

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



Identifying Suspect Threads: High Level Algorithm

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