BlockHammer: Preventing RowHammer at Low Cost by Blacklisting Rapidly-Accessed DRAM Rows

A. Giray Yağlıkçı, Minesh Patel, Jeremie S. Kim, Roknoddin Azizi, Ataberk Olgun, Lois Orosa, Hasan Hassan, Jisung Park, Konstantinos Kanellopoulos, Taha Shahroodi, Saugata Ghose, Onur Mutlu

ETH Zürich University of Illinois at Urbana–Champaign

HPCA 2021

presented by Philipp Niksch

9.12.2021

Executive Summary

Problem: RowHammer flips bits by accessing adjacent memory rows rapidly

Goal: Efficiently and scalably prevent RowHammer attacks without knowledge or modifications to DRAM internals

Key Idea: Blacklist rows that are being activated too rapidly and throttle further accesses

Mechanism: BlockHammer mitigates attacks in two steps

- *RowBlocker: Blacklists Rows that are being accessed too rapidly*
- *AttackThrottler: Throttles memory bandwidth to potential attacking threads*

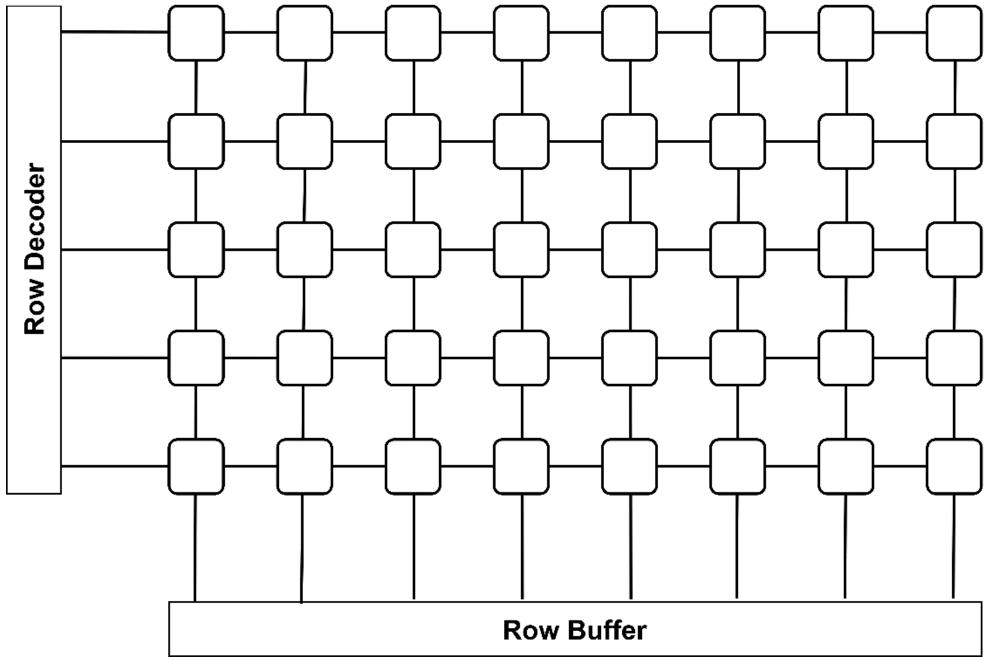
Comparisons: Compared to other techniques BlockHammer's performance is competitive while not under attack and significantly increases performance of benign applications when under attack

Outline

1. **Problem**
2. Previous Solutions
3. BlockHammer
4. Comparisons
5. Strengths and Weaknesses
6. Discussion

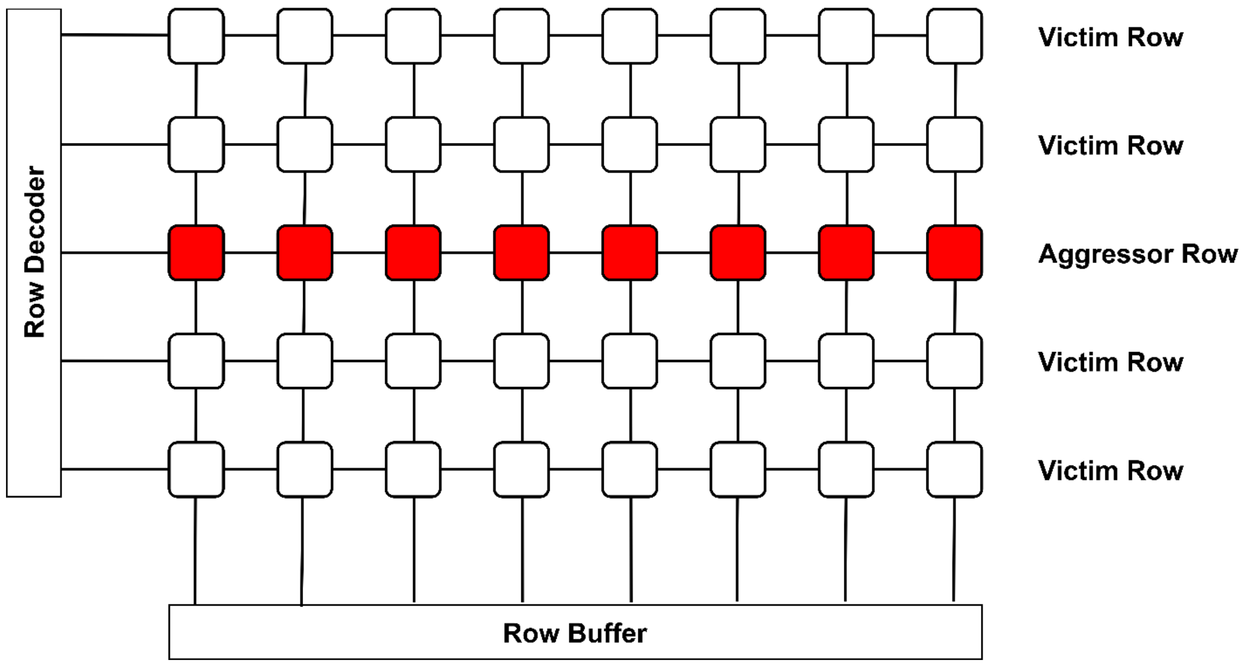
1. Problem

- DRAM is organized as an array of bits. Rows are always accessed entirely
- Activating a row transfers its content into the row buffer
- Cells loose state over time and need to be refreshed periodically



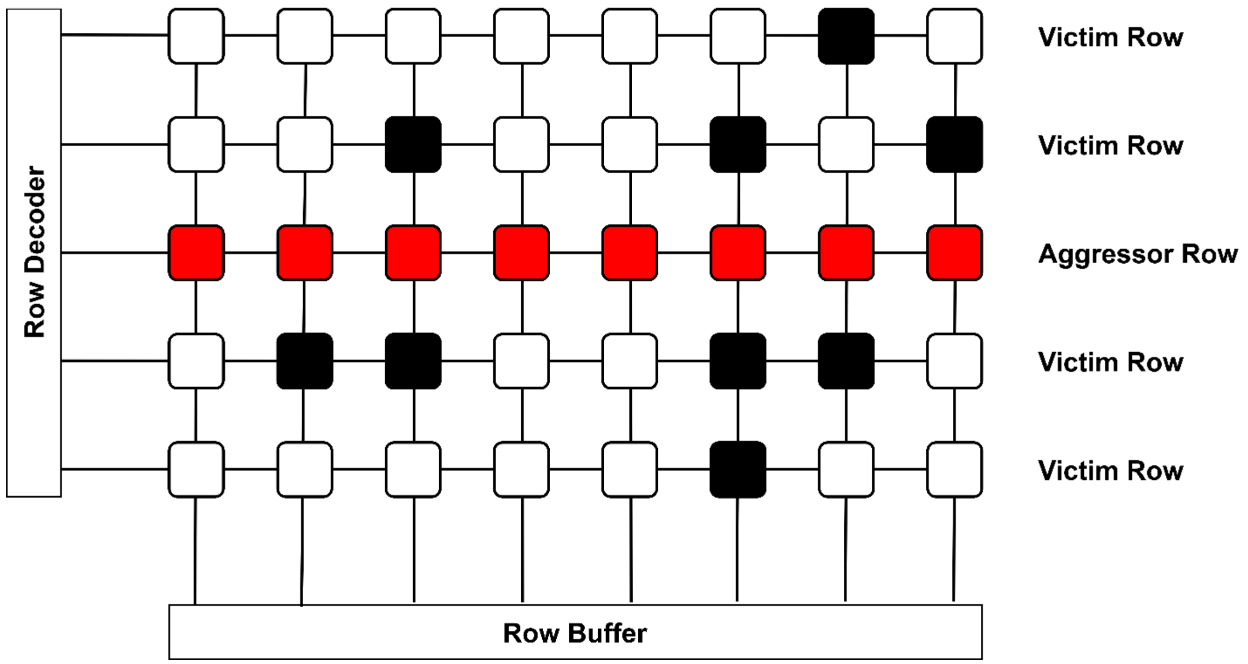
1. Problem

- RowHammer is a DRAM vulnerability caused by rapid activation of the same memory row



1. Problem

- RowHammer is a DRAM vulnerability caused by rapid activation of the same memory row
- Rapidly activating a row can induce bitflips in nearby rows



1. Problem

- It has been shown that RowHammer can be used to gain kernel privileges on certain systems

- Previous work has shown that chips get more vulnerable to RowHammer over the years
  - *Cells are getting smaller and have less charge, so less effort is required to make them flip*
  - *Memory becomes denser, so there is less physical distance between each row*

Outline

1. Problem

2. **Previous Solutions**

3. BlockHammer
4. Comparisons
5. Strengths and Weaknesses
6. Discussion

2. Previous Solutions

There are four high level approaches to mitigate the problem

1. **Increase refresh rate**
   - *Unnecessary refreshes*
2. **Reactive refresh**
   - *Unnecessary refreshes*
3. **Physical isolation**
   - *Waste of memory*
4. **Proactive Throttling**
   - *Throttling of benign threads*

2. Previous Solutions

Challenge 1: Ability to scale with worsening of RowHammer

1. Increase refresh rate
2. Reactive refresh
3. Physical isolation
4. Proactive Throttling

2. Previous Solutions

Challenge 2: Compatibility with commodity DRAM Chips

1. Increase refresh rate
2. Reactive refresh
3. Physical isolation
4. Proactive Throttling

2. Previous Solutions

The goals for RowHammer mitigation mechanism

1. Address a comprehensive threat model
2. Compatibility with commodity DRAM chips
3. Scalability with increasing vulnerability
4. Deterministically prevent all RowHammer attacks

Outline

1. Problem
2. Previous Solutions

3. **BlockHammer**

4. Comparisons
5. Strengths and Weaknesses
6. Discussion

3. BlockHammer

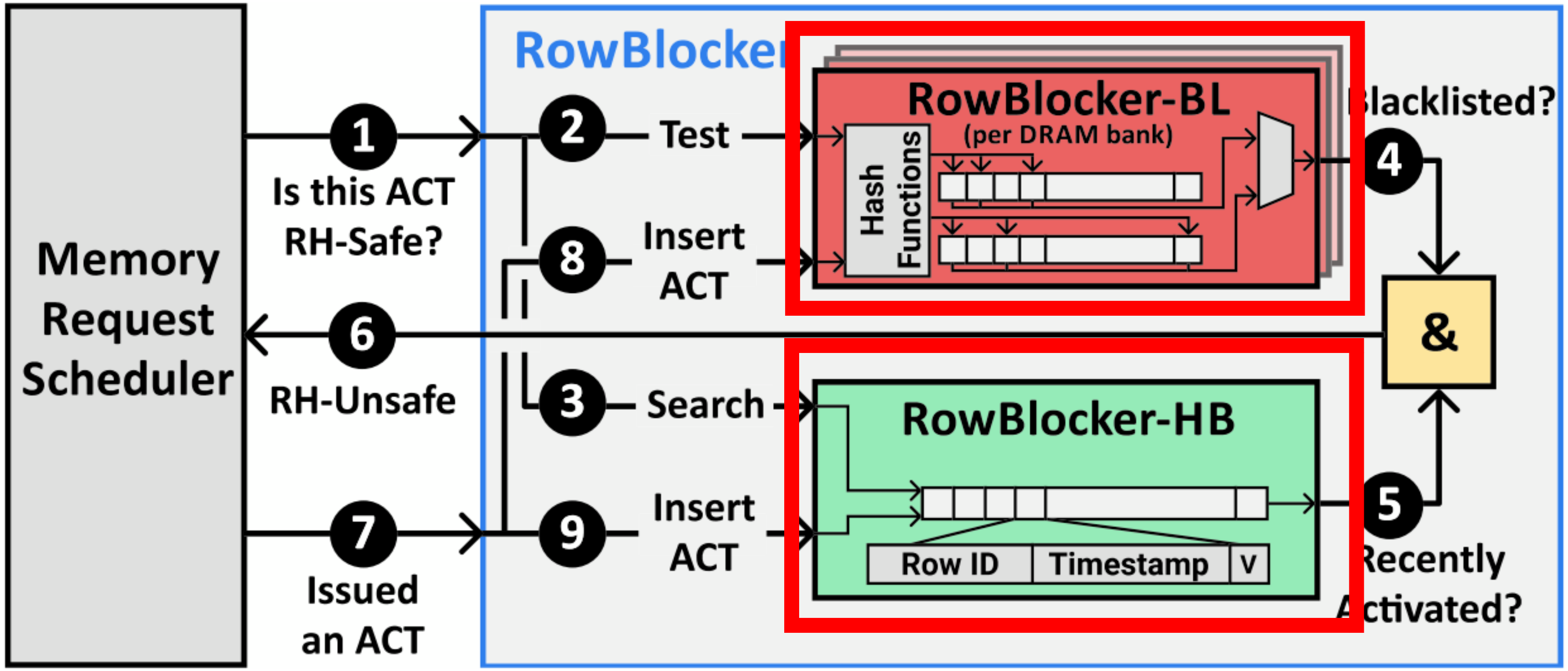
1. RowBlocker
   - *Uses Bloom filters to keep track of row activation rates*
   - *Blacklists potentially dangerous rows*
   - *Delays activations on blacklisted rows*

   → ***Deterministically prevents all RowHammer attacks***

2. AttackThrottler
   - *Identifies threads that are likely to be RowHammer attacks*
   - *Reduces memory bandwidth of identified threads*

   → ***Greatly improves performance of benign threads while under attack***

3. RowBlocker



3. RowBlocker-BL

- Keeps track of the number of row activations for each row
- The naive approach to have a counter for each row is too expensive
- Infrequent false positives are tolerable
- False negatives are bad
- RowBlocker-BL uses Unified Counting Bloom filters for blacklisting
- Bloom filters consists of a bit array and set of hash functions. It implements:
  - *Clear*
  - *Insert(x)*
  - *Test(x)*

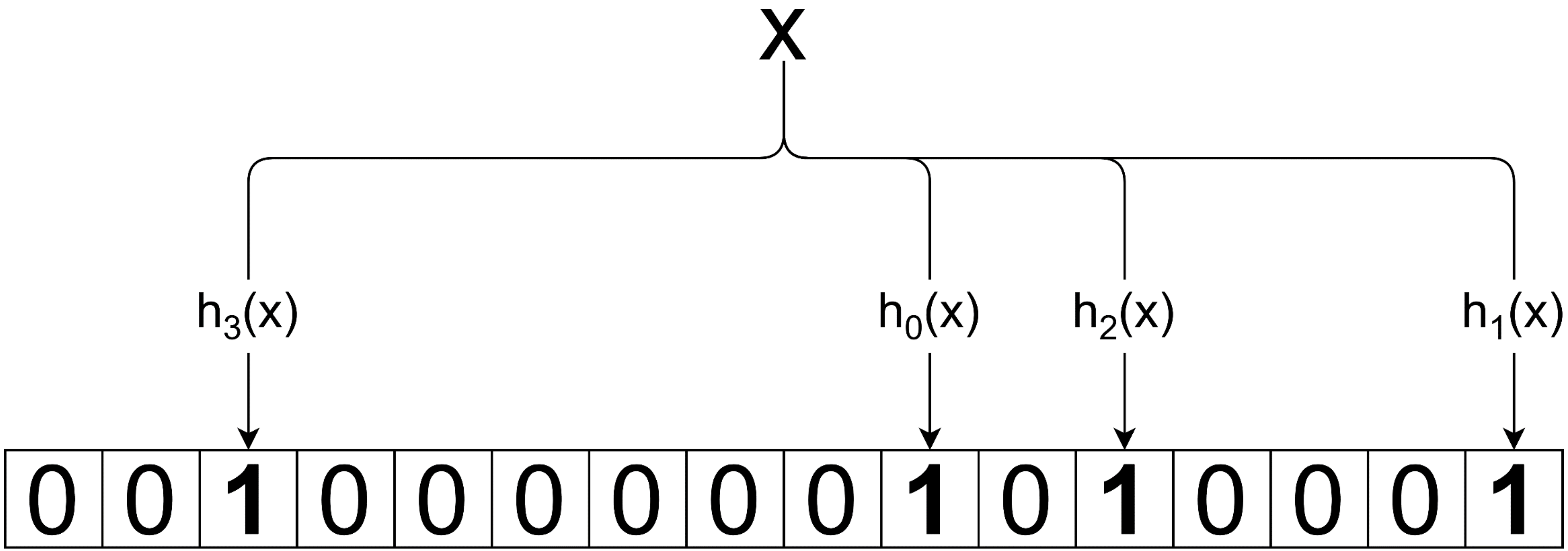
3. RowBlocker-BL

- Bloom filter Clear

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

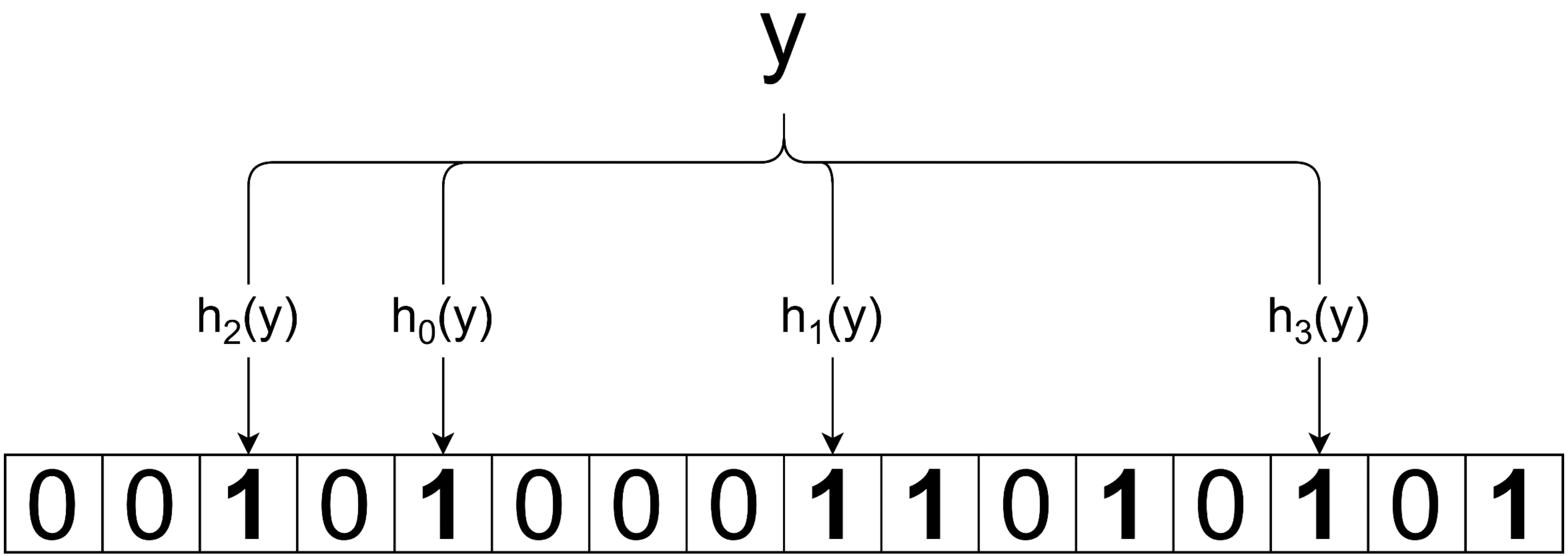
3. RowBlocker-BL

- Bloom filter Insert(x)



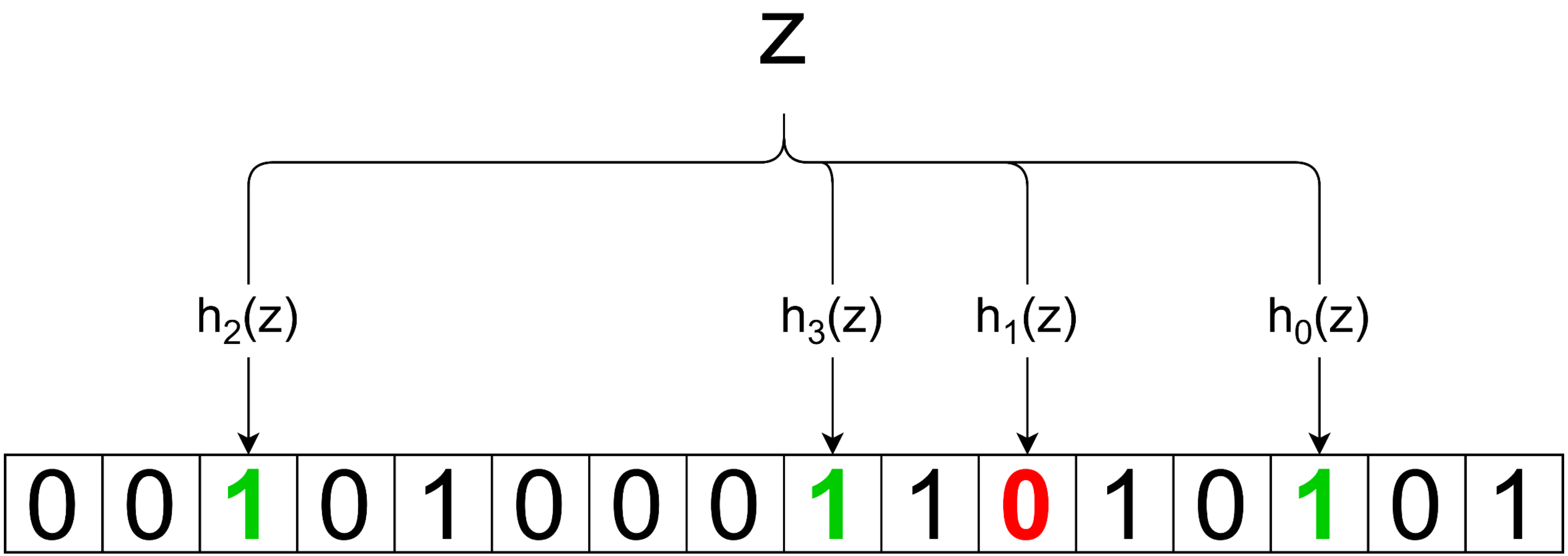
3. RowBlocker-BL

- Bloom filter Insert (y)



3. RowBlocker-BL

- Bloom filter Test(z)



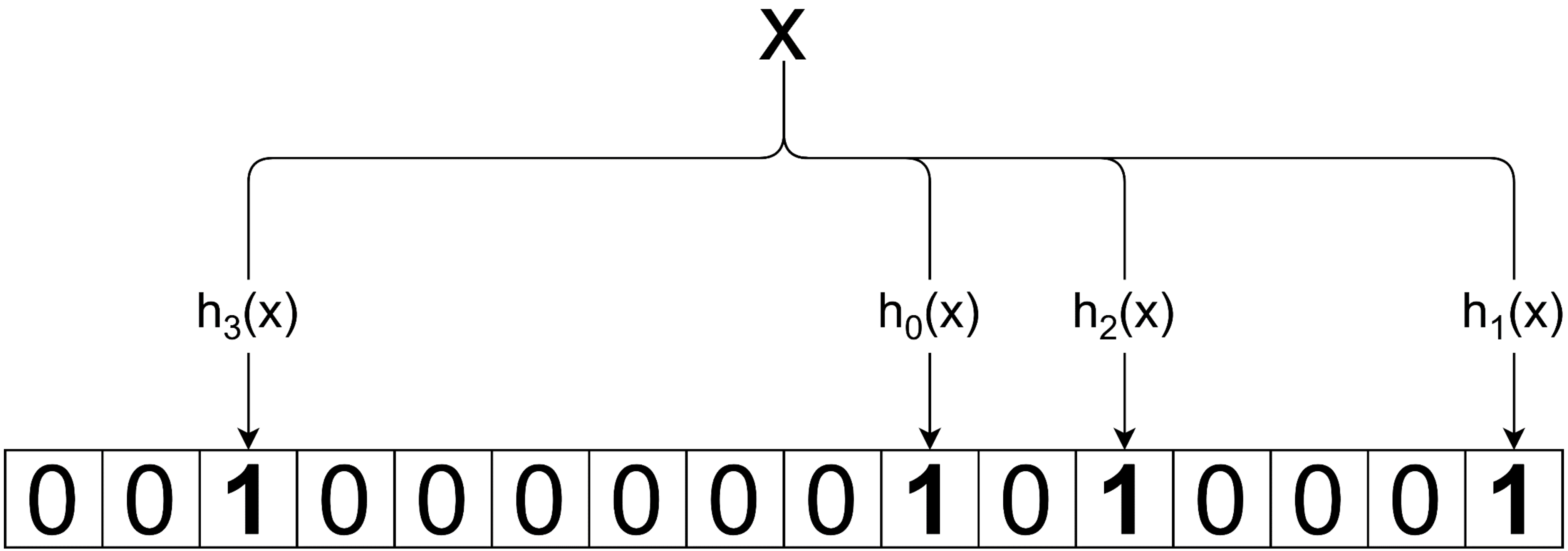
3. RowBlocker-BL

- Counting Bloom filter Clear

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

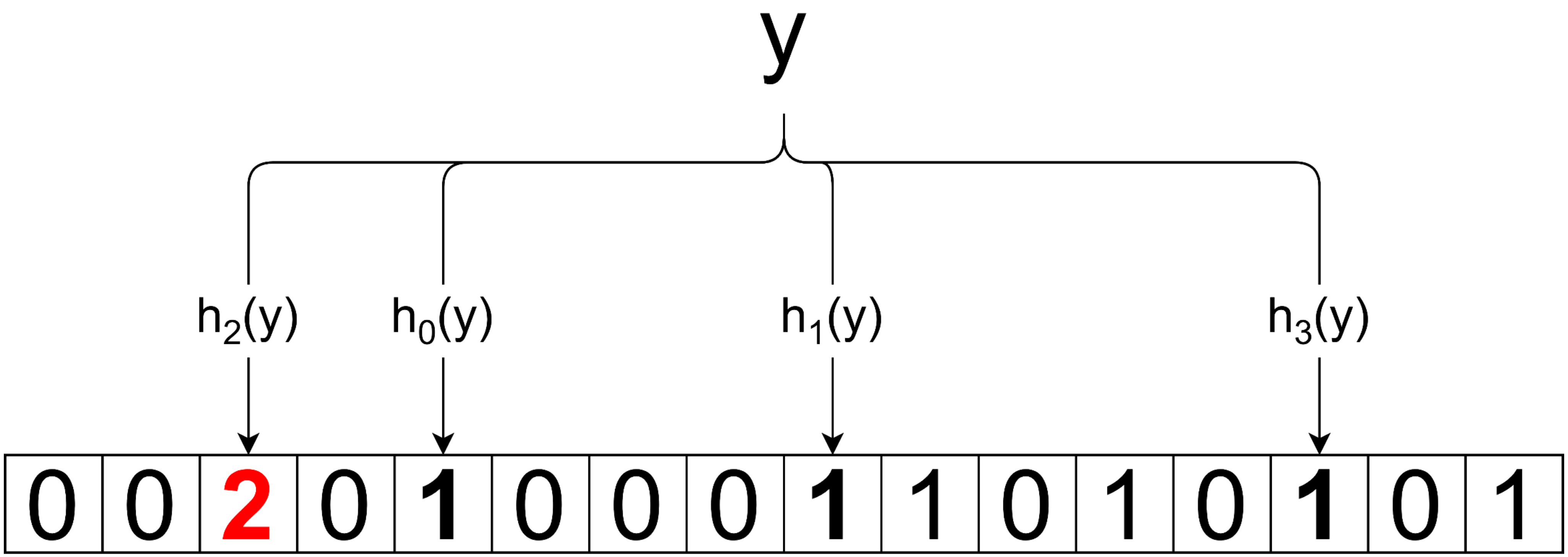
3. RowBlocker-BL

- Counting Bloom filter Insert(x)



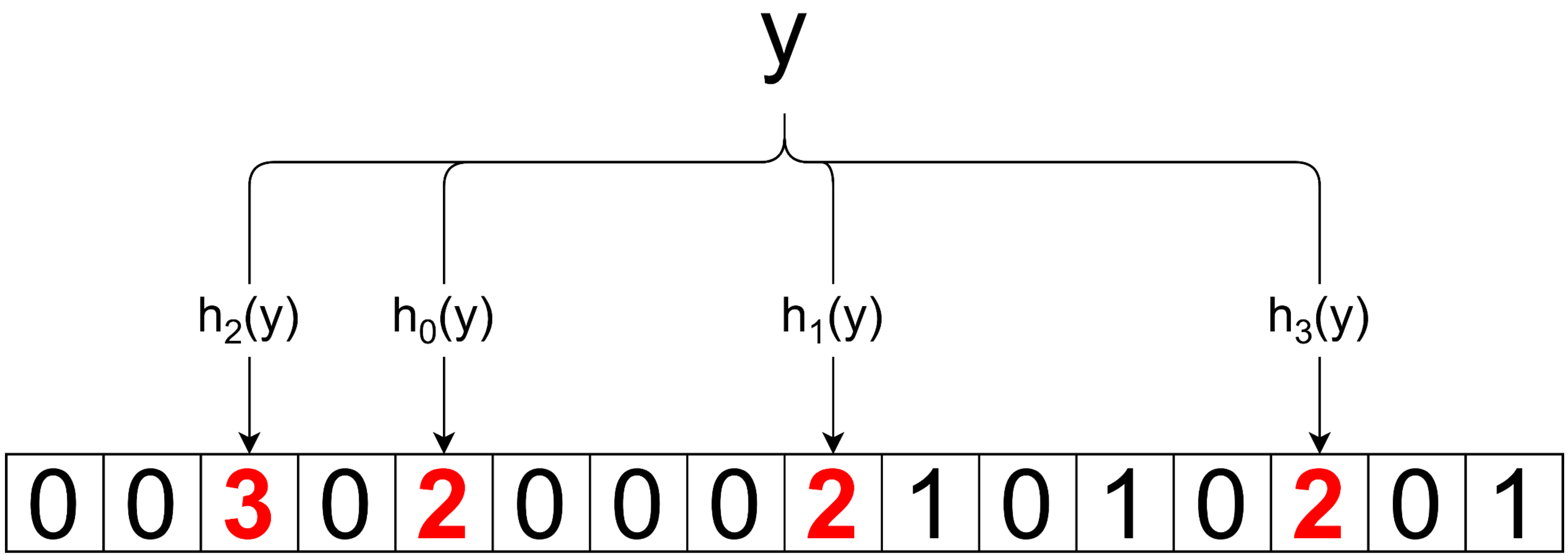
3. RowBlocker-BL

- Counting Bloom filter Insert(y)



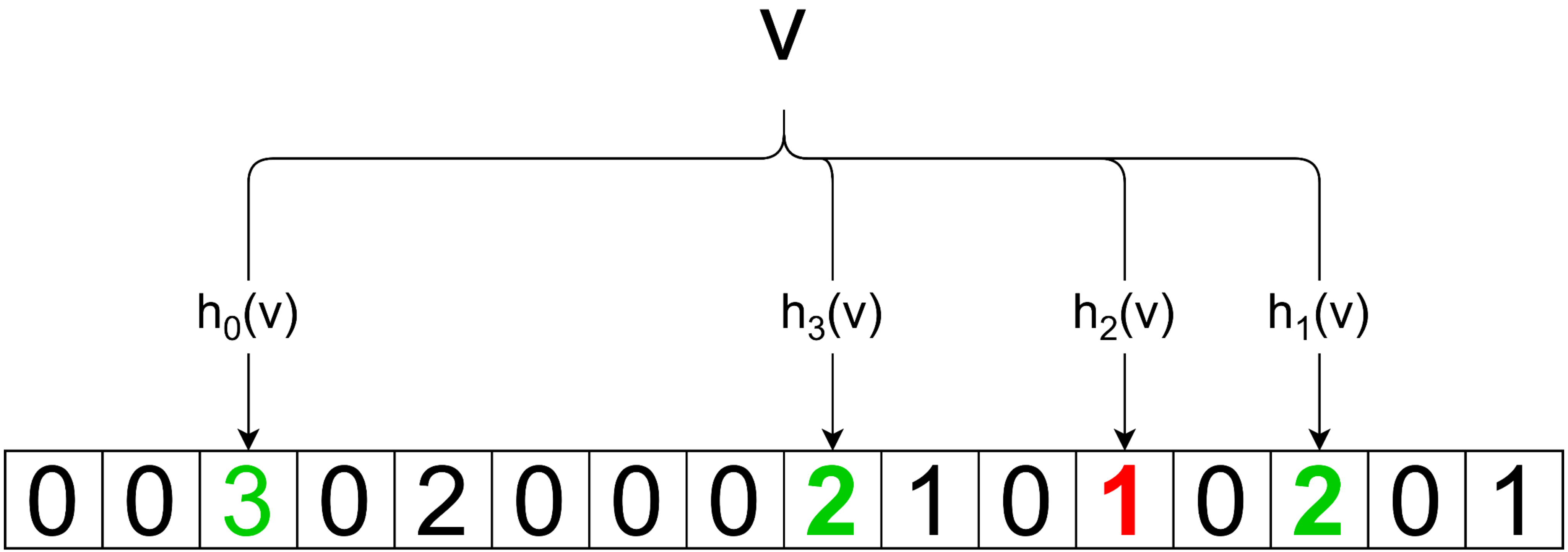
3. RowBlocker-BL

- Counting Bloom filter Insert(y) a second time



3. RowBlocker-BL

- Counting Bloom filter Test(v)



3. RowBlocker-BL

Unified Counting Bloom filter

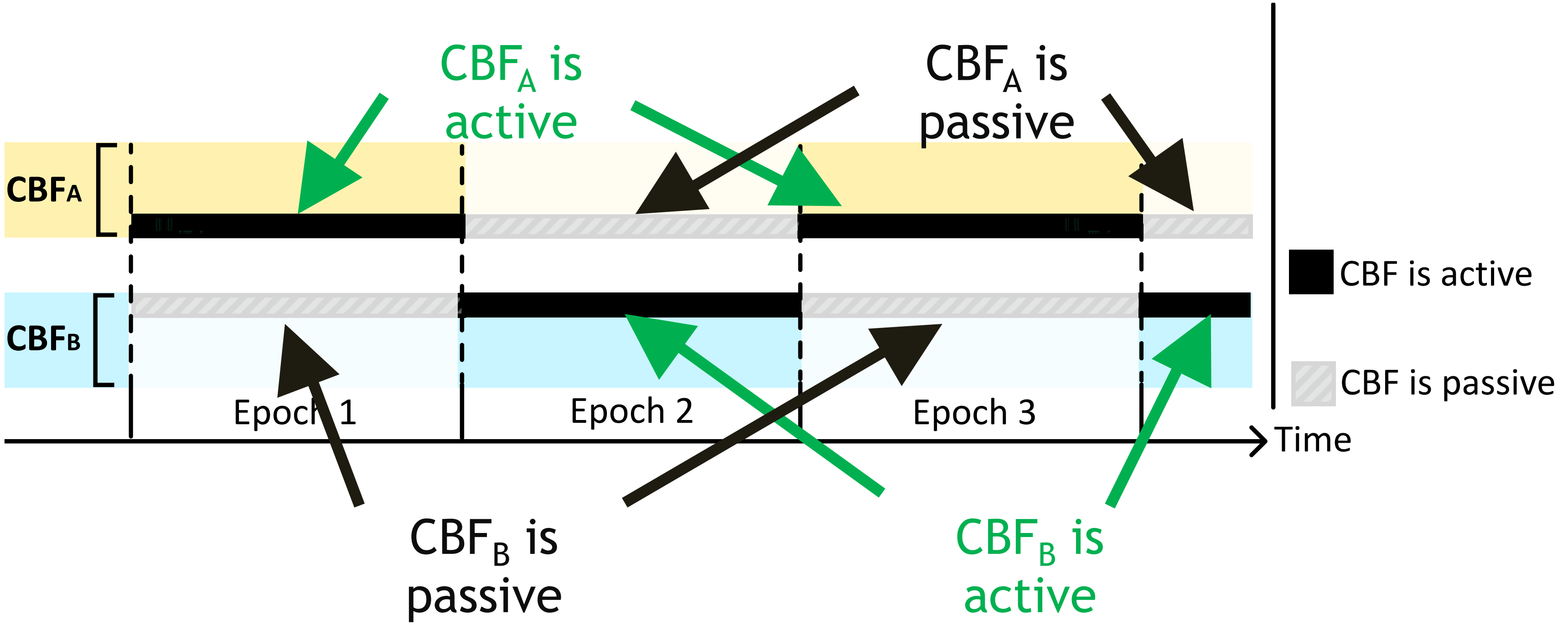
- (Counting) Bloom filters saturate over time increasing the rate of false positives
- Clearing the Bloom filter looses all the information at once leading to potentially dangerous rows not being blacklisted anymore

→ Unified Counting Bloom filters combine two Counting Bloom filters

  - *Elements are always inserted into both filters*
  - *The filters are taking turns clearing*
  - *Test queries are answered by the filter, that has been active for longer*

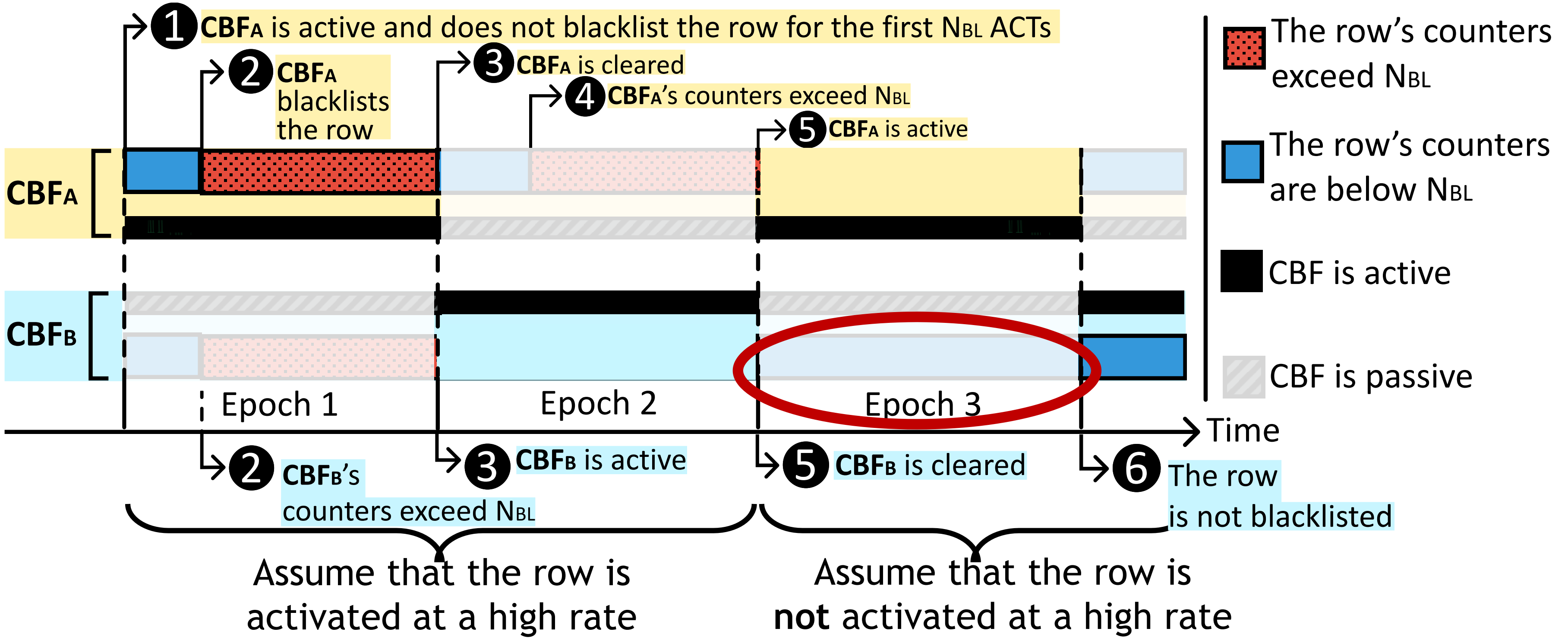
3. RowBlocker-BL

- Unified Counting Bloom filter in action

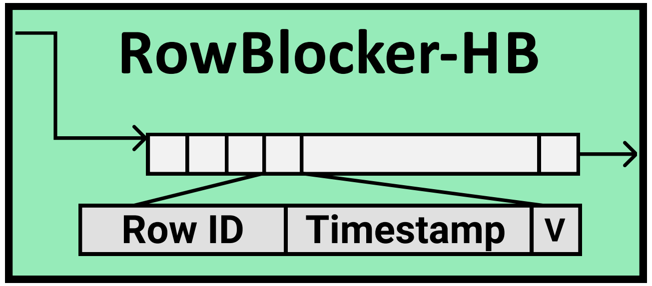


3. RowBlocker-BL

- Unified Counting Bloom filter in action

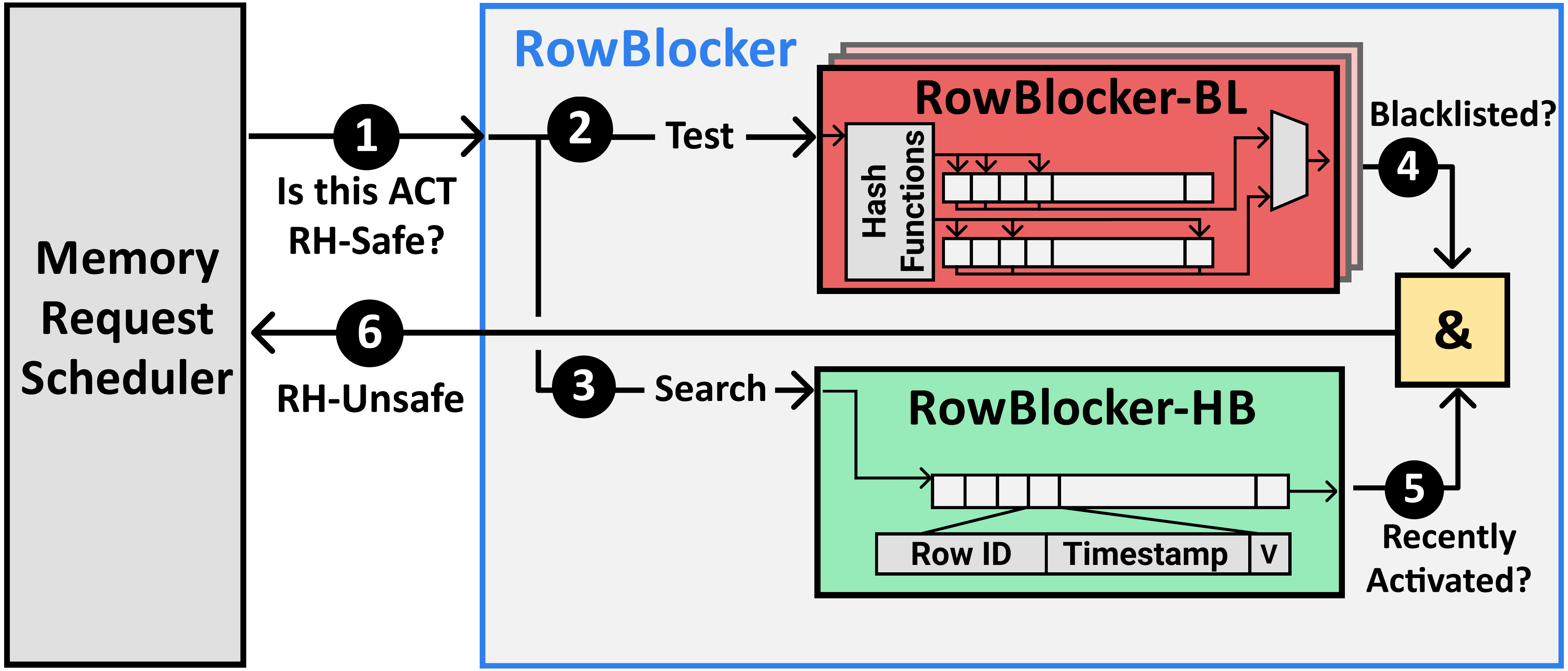


3. RowBlocker-HB

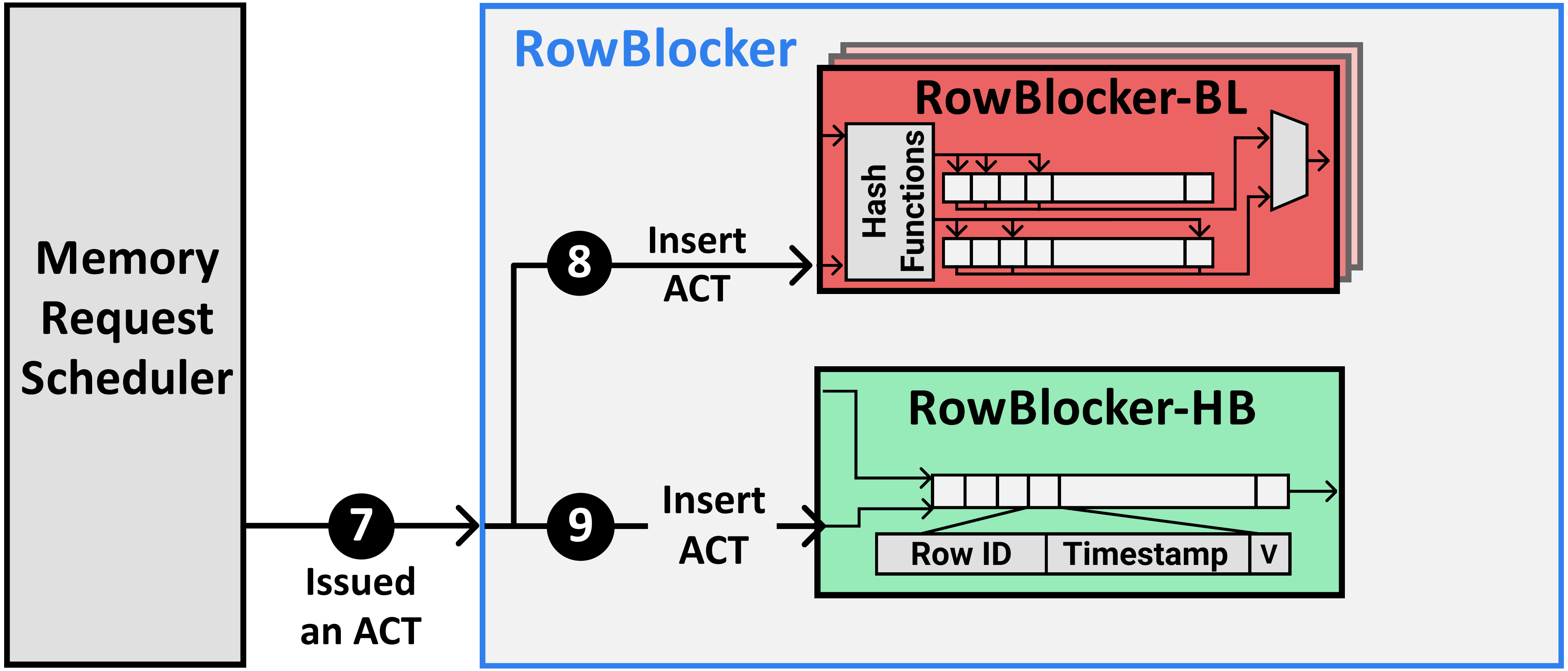


- In order to induce a bitflip, the aggressor row has to be activated with a minimum frequency. If we keep a certain amount of time t_{delay} between each activation, we can guarantee RowHammer safety
- RowBlocker HB maintains a FIFO history buffer containing all row activations in the last time window t_{delay}
- A row access is getting delayed when
  - *The row is blacklisted by RowBlocker-BL*
  - *AND the row was accessed in the last time window t_{delay} and is therefore in the history buffer*

3. RowBlocker



3. RowBlocker



3. AttackThrottler

- Identify and throttle threads that potentially induce bitflips
- AttackThrottler uses a Rowhammer Likelyhood Index (RHLI) between 0 and 1, to identify dangerous threads

$$RHLI \propto Blacklisted\ Row\ Activation\ Count$$

- A benign thread has a RHLI of ≈ 0 because it never accesses blacklisted rows
- A thread performing a RowHammer attack will have a RHLI of ≈ 1
- The maximum memory bandwidth of a thread will be limited more and more strictly as its RHLI goes to 1
- Optionally the operating system has access to the RHLI as well and can take further action

Outline

1. Problem
2. Previous Solutions
3. BlockHammer
4. **Comparisons**
5. Strengths and Weaknesses
6. Discussion

4. Comparisons

- The paper compares BlockHammers performance to 6 other state of the art RowHammer mitigation techniques
  - **PARA** *[Y. Kim et al., “Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors,” in ISCA, 2014.]*
  - **ProHIT** *[M. Son et al., “Making DRAM Stronger Against Row Hammering,” in DAC, 2017]*
  - **MRLoc** *[J. M. You and J.-S. Yang, “MRLoc: Mitigating Row-Hammering Based on Memory Locality,” in DAC, 2019.]*
  - **CBT***[S. M. Seyedzadeh et al., “Mitigating Wordline Crosstalk Using Adaptive Trees of Counters,” in ISCA, 2018.]*
  - **TWiCE** *[E. Lee et al., “TWiCe: Preventing Row-Hammering by Exploiting Time Window Counters,” in ISCA, 2019.]*
  - **Graphene** *[Y. Park et al., “Graphene: Strong yet Lightweight Row Hammer Protection,” in MICRO, 2020.]*

4. Comparisons

- The paper compares the area and energy costs for both a normal and a more vulnerable machine

Mitigation Mechanism	N_{RH}=32K* SRAM KB	CAM KB	Area mm^2	Area % CPU	Access Energy (pJ)	Static Power (mW)	N_{RH}=1K SRAM KB	CAM KB	Area mm^2	Area % CPU	Access Energy (pJ)	Static Power (mW)
BlockHammer	**51.48**	**1.73**	**0.14**	**0.06**	**20.30**	**22.27**	**441.33**	**55.58**	**1.57**	**0.64**	**99.64**	**220.99**
Dual counting Bloom filters	48.00	-	0.11	0.04	18.11	19.81	384.00	-	0.74	0.30	86.29	158.46
H3 hash functions	-	-	< 0.01	< 0.01	-	-	-	-	< 0.01	< 0.01	-	-
Row activation history buffer	1.73	1.73	0.03	0.01	1.83	2.05	55.58	55.58	0.83	0.34	12.99	62.12
AttackThrottler counters	1.75	-	< 0.01	< 0.01	0.36	0.41	1.75	-	< 0.01	< 0.01	0.36	0.41
PARA [73]	**-**	**-**	**< 0.01**	**-**	**-**	**-**	**-**	**-**	**< 0.01**	**-**	**-**	**-**
ProHIT [137]*	**-**	**0.22**	**< 0.01**	**<0.01**	**3.67**	**0.14**	×	×	×	×	×	×
MrLoc [161]*	**-**	**0.47**	**< 0.01**	**<0.01**	**4.44**	**0.21**	×	×	×	×	×	×
CBT [132]	**16.00**	**8.50**	**0.20**	**0.08**	**9.13**	**35.55**	**512.00**	**272.00**	**3.95**	**1.60**	**127.93**	**535.50**
TWiCE [84]	**23.10**	**14.02**	**0.15**	**0.06**	**7.99**	**21.28**	**738.32**	**448.27**	**5.17**	**2.10**	**124.79**	**631.98**
Graphene [113]	**-**	**5.22**	**0.04**	**0.02**	**40.67**	**3.11**	**-**	**166.03**	**1.14**	**0.46**	**917.55**	**93.96**

* PRoHIT [137] and MRLoc [161] do *not* provide a concrete discussion on how to adjust their empirically-determined parameters for different N_{RH} values. Therefore, we (1) report their values for a fixed design point that each paper provides for N_{RH}=2K and (2) mark values we cannot estimate using an ×.

- The area and energy requriements are higher for $N_{RH} = 32k$, but they scale much better than those of other techniques down to $N_{RH} = 1k$

4. Comparisons

- The table shows area and power requirements for a RowHammer threshold of 32k

Mitigation Mechanism	SRAM KB	CAM KB	Area mm^2	%CPU	Access Energy pJ	Static Power mW
BlockHammer	51.48	1.73	0.14	0.06	20.30	22.27
PARA [73]	-	-	<0.01	-	-	-
ProHIT [137]	-	0.22	<0.01	<0.01	3.67	0.14
MRLoc [161]	-	0.47	<0.01	<0.01	4.44	0.21
CBT [132]	16.00	8.50	0.20	0.08	9.13	35.55
TWiCe [84]	23.10	14.02	0.15	0.06	7.99	21.28
Graphene [113]	-	5.22	0.04	0.02	40.67	3.11

- The area and energy requirements are competitive to other mitigation techniques

4. Comparisons

- The table shows area and power requirements for a RowHammer threshold of 1k

Mitigation Mechanism	SRAM KB	CAM KB	Area mm^2	%CPU	Access Energy pJ	Static Power mW
BlockHammer	441.33	55.58	1.57	0.64	99.64	220.99
PARA [73]	-	-	<0.01	-	-	-
ProHIT [137]	x	x	x	x	x	x
MRLoc [161]	x	x	x	x	x	x
CBT [132]	512.00	272.00	3.95	1.60	127.93	535.50
TWiCe [84]	738.32	448.27	5.17	2.10	124.79	631.98
Graphene [113]	-	166.03	1.14	0.46	917.55	93.96

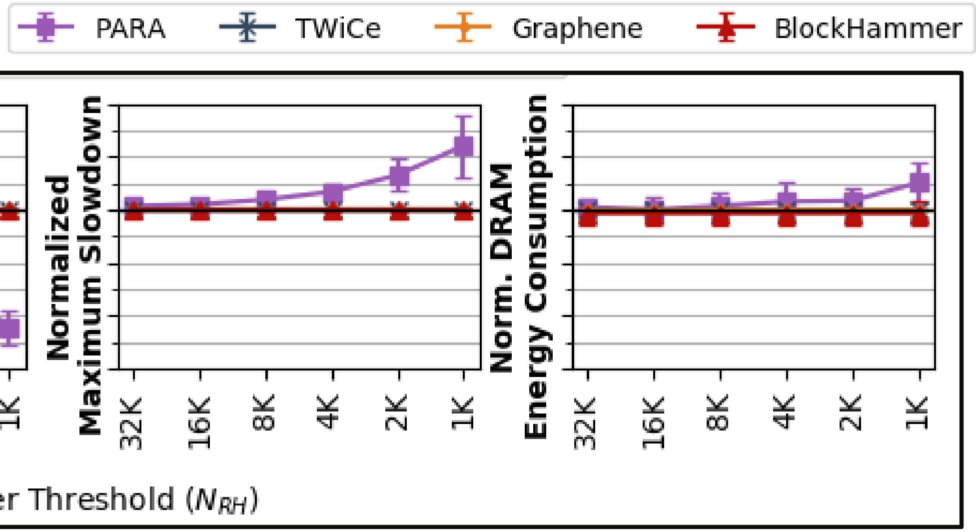
- The area and energy requirements of RowHammer scale more efficiently with increasing vulnerability

4. Comparisons

- Latency Analysis shows a latency of 0.97ns for each blacklist lookup
- DRAM standards enforce a row access latency of 45-50 ns
- While accessing memory, we can check the blacklist for the next request

4. Comparisons

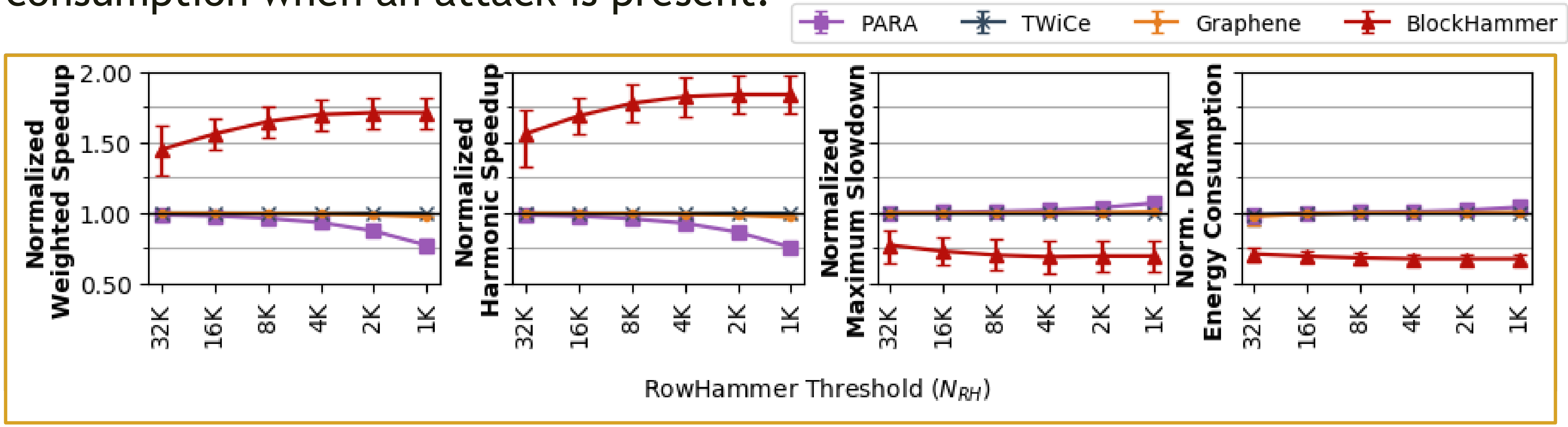
- Evaluating system throughput, job turnaround time, unfairness and DRAM energy consumption when no attack is present.



- BlockHammer introduces (<0.5%) performance and (<0.4%) DRAM energy overheads

4. Comparisons

- Evaluating system throughput, job turnaround time, unfairness and DRAM energy consumption when an attack is present.



- BlockHammer improves performance of benign applications (~45%) and reduces DRAM energy consumption (~29%)

Outline

1. Problem
2. Previous Solutions
3. BlockHammer
4. Comparisons

5. **Strengths and Weaknesses**

6. Discussion

5. Strengths

- The solution described is simple, effective and only involves the memory controller
- Scales better than other mitigation techniques with worsening vulnerability
- AttackThrottler increases the performance of benign applications while a RowHammer Attack is present
- The paper clearly motivates the need for another RowHammer mitigation technique
- The mechanism is deterministic

5. Weaknesses

- Some of the benefits are only apparent once RowHammer has worsened
- An attacker might be able to saturate the Bloom filters to reduce the performance of the system
- The paper mentions other mitigation techniques by name, but only introduces them in chapter 7
- Some of the graphs are too small for the amount of information they contain

Outline

1. Problem
2. Previous Solutions
3. BlockHammer
4. Comparisons
5. Strengths and Weaknesses

6. **Discussion**

6. Discussion

- Could we desaturate the bloom filters by dividing the counters by 2 instead of clearing them?

- Could we retrofit AttackThrottler to other existing RowHammer mitigation methods?

- Instead of using Unified Counting Bloom filters, are there any other data structures worth considering for a blacklisting mechanism?