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

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

Seminar in Computer Architecture HS 2022
24.11.2022
Published in HPCA 2021

Roman Hoffmann

SAFARI
SAFARI Research Group

Executive Summary

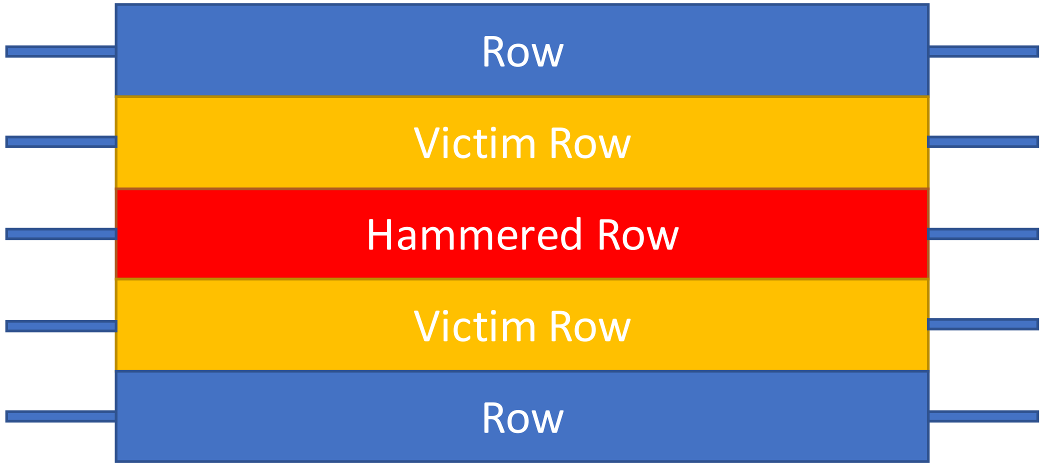
- Background: RowHammer is a serious security issue
- Problem: Mitigation mechanisms have limited support for current/future chips
  - **Compatibility** with commodity DRAM chips
  - **Scalability** with worsening RowHammer effects
- Goal: Efficient and scalable method to prevent RowHammer bit-flips without knowledge of or modifications to DRAM internals
- Key Mechanism:
  - Selectively **limit memory accesses** that may cause RowHammer bit-flips
  - **Identifying** and **throttling** potential attacker
- Key Results:
  - Scalable complexity
  - Highly efficient solution in terms of energy consumption (<0.6%) and performance (71% under an attack)

Outline

- Recab: RowHammer
- Requirements for the solution
- Possible Solutions
- BlockHammer
  - General
  - RowBlocker
  - AttackThrottler
- Evaluation
- Conclusion
- Strengths and Weaknesses
- Discussion

RowHammer

- We are seeing here a DRAM bank

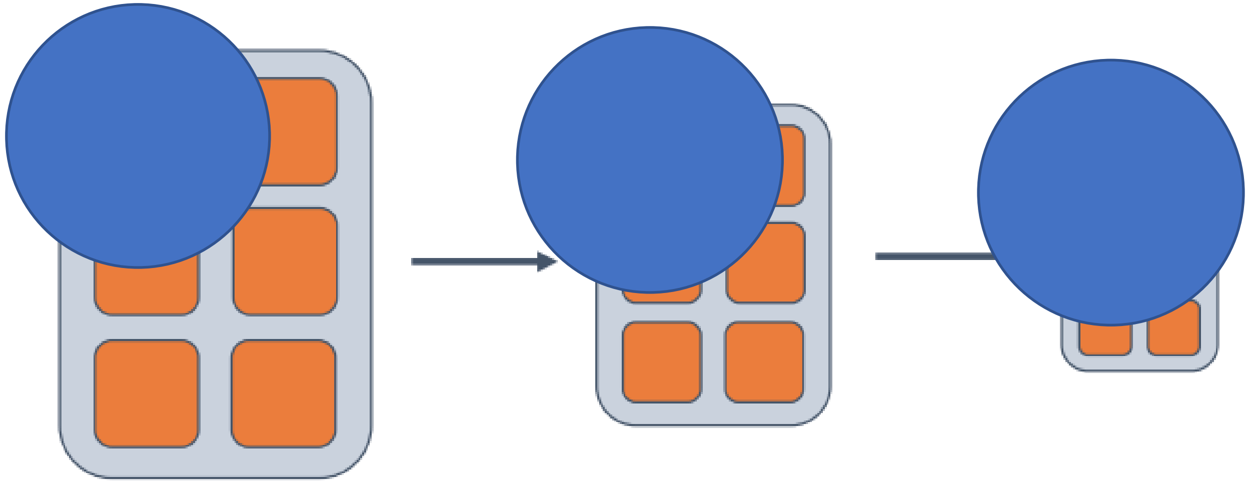


Requirements

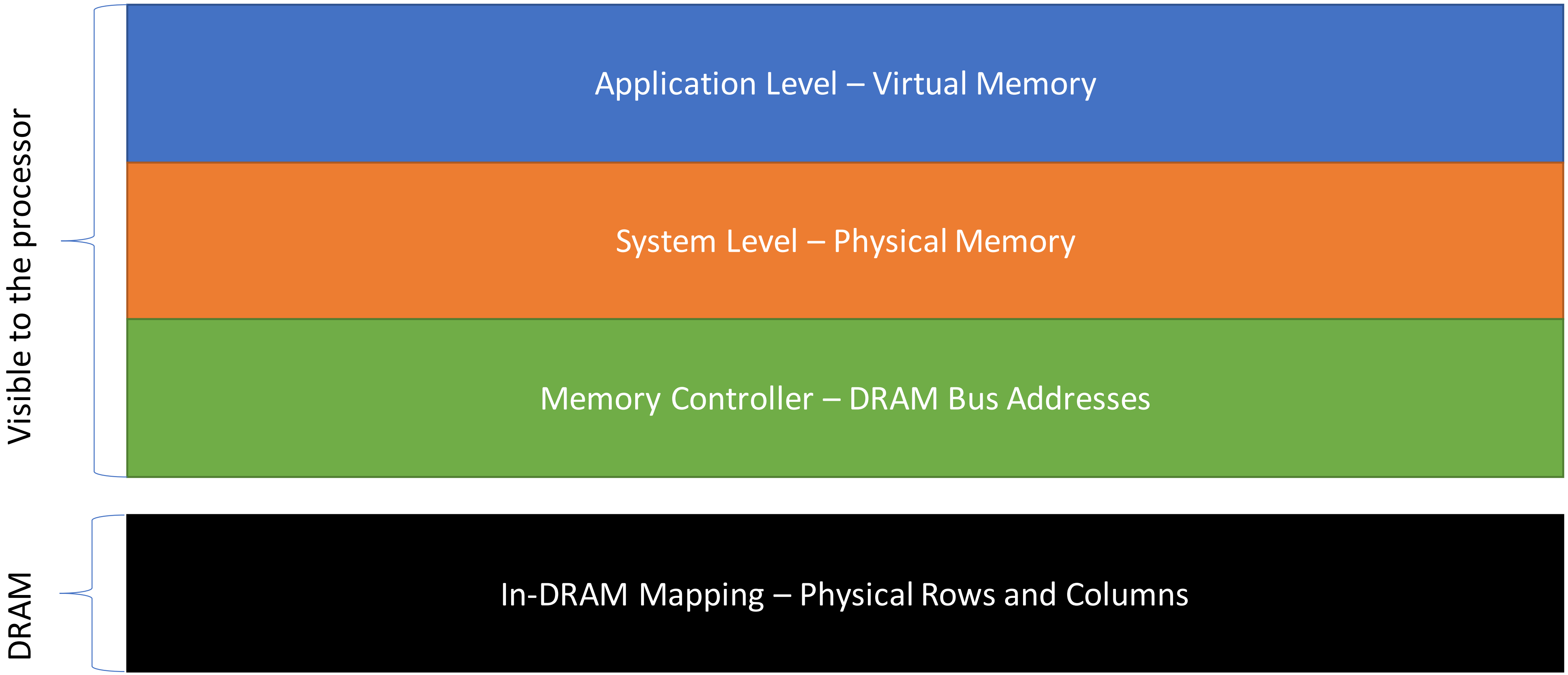
SAFARI
SAFARI Research Group

Scalability

- DRAM chips are more vulnerable to RowHammer today
- The density of DRAM chips increases
- A RowHammer bit-flip occurs with a lower amount of accesses
- Blast radius is increasing



Compatibility



In-DRAM Mapping

- **Design Optimizations**: Provides better density, power and performance by simplifying DRAM circuitry
- **Yield Improvements**: Internal mapping from faulty rows to working rows

- In-DRAM mapping is not published to the outer world

Our solution should not require knowledge of DRAM internals!

Possible Solutions

SAFARI Research Group

Solution 1: Increase Refresh Rate

- Process: Increase the refresh rate of all DRAM rows to prevent RowHammer bit-flips
- Drawbacks:
  - Higher power consumption
  - Performance loss

Available for: OS X Mountain Lion v10.8.5, OS X Mavericks v10.9.5

Impact: A malicious application may induce memory corruption to escalate privileges

Description: A disturbance error, also known as Rowhammer, exists with some DDR3 RAM that could have led to memory corruption. This issue was mitigated by increasing memory refresh rates.

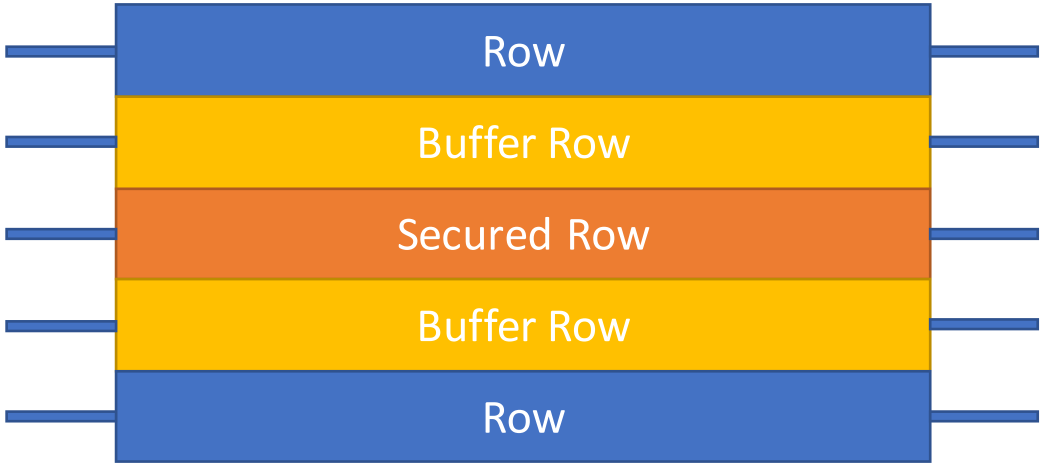
CVE-ID

CVE-2015-3693 : Mark Seaborn and Thomas Dullien of Google, working from original research by Yoongu Kim et al (2014)

Compatible but not scalable

Solution 2: Physical Isolation

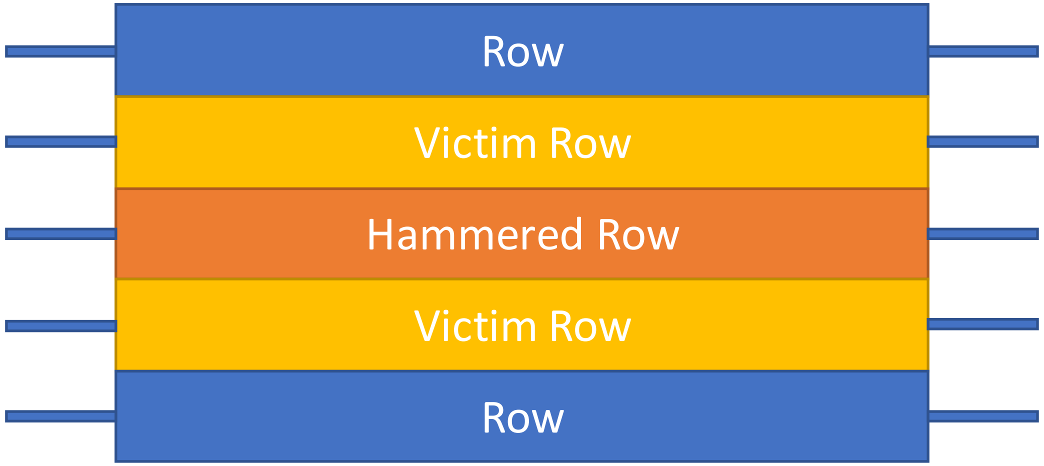
- Process: Separate physically sensitive data
- Drawbacks:
  - Requires a lot memory
  - Because RowHammer is getting worse, the fraction of cells we can protect decreases
  - Requires knowledge on DRAM internals



Not compatible and not scalable

Solution 3: Reactive Refresh

- Process: Observes activations and refreshes potential victim rows
- Used in PARA, Graphene, TwiCe
- Drawback
  - Requires knowledge on DRAM internals



Not compatible but scalable

Solution 4: Proactive throttling

- Process: Limit repeated access to the same row
- Drawback
  - Decreases performance of benign applications

Compatible and scalable but not efficient

Solution

SAFARI Research Group

Goal

Prevent RowHammer bit-flips **efficiently** and **scalably** without any knowledge of or modifications to DRAM

Idea

Selectively **throttle memory accesses** that may cause RowHammer bit-flips

Idea in Detail

- An attacker **hammers** a row
- BlockHammer **detects** a RowHammer attack
- BlockHammer **selectively** throttles accesses from within the memory controller
- Access **limitations** make it impossible for bit-flips to occur
- BlockHammer informs the system software about a potential attack

BlockHammer Overview

- **RowBlocker**
  - Tracks row activations rates
  - Blacklists rows
  - Throttles activations targeting a blacklisted row

  => Limits the row activation rate

- **AttackThrottler**
  - Identifies threads that perform RowHammer attacks
  - Reduces memory bandwidth usage of identified potential threads

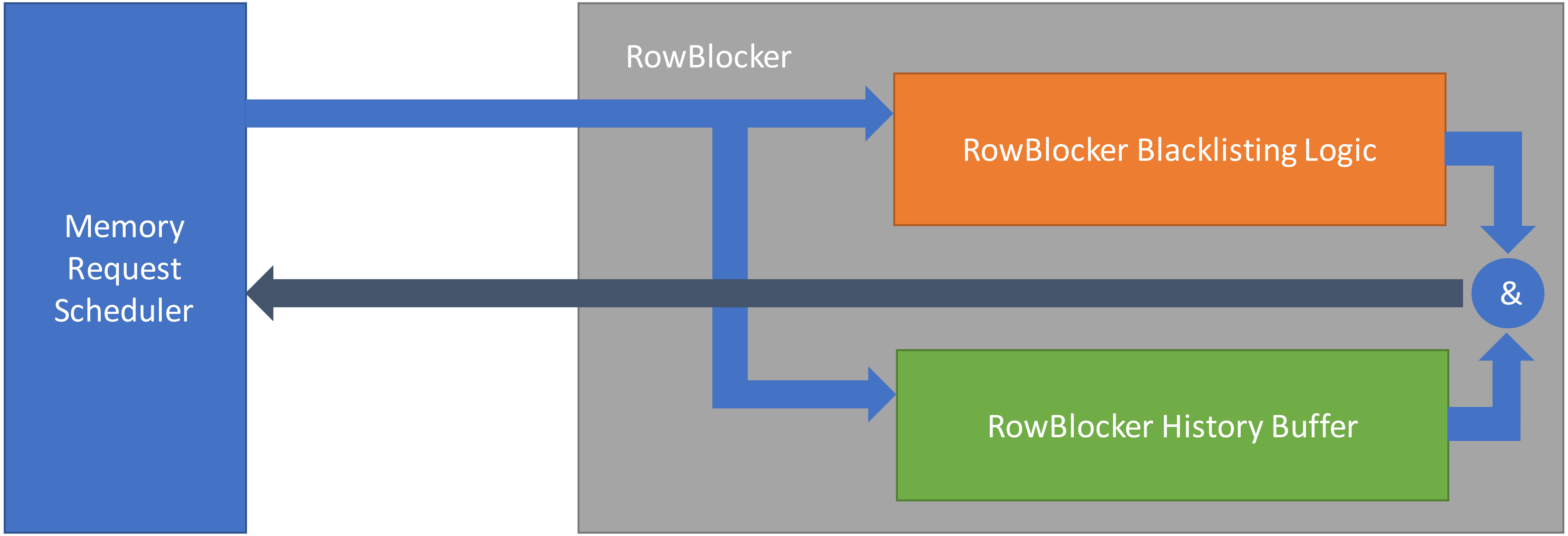
  => Reduces performance degradation during an attack

RowBlocker

SAFARI
SAFARI Research Group

RowBlocker - Overview

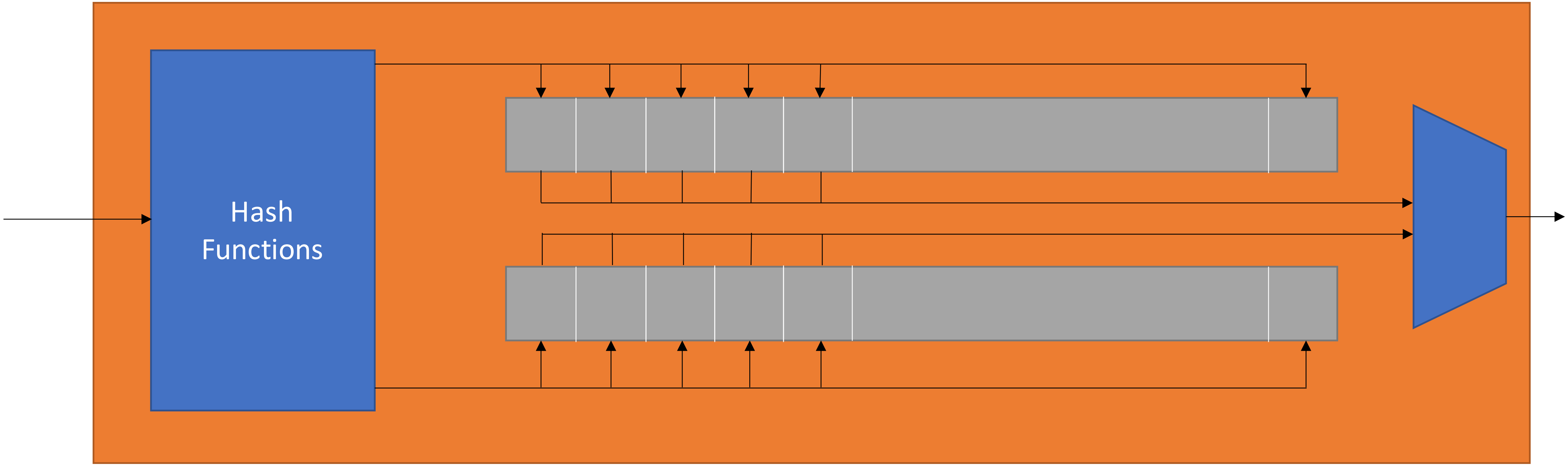
- Throttles row activations
- Blacklists rows and delays activations of blacklisted rows



RowBlocker - Blacklisting Logic

- **Blacklists** a row when the row‘s activation count in a time window exceeds a threshold
- Uses two efficient **Bloom filter** to track the recent accesses
- Accesses are stored in **both filters**
- Only **one** filter is active at a given time

RowBlocker – Blacklisting Logic



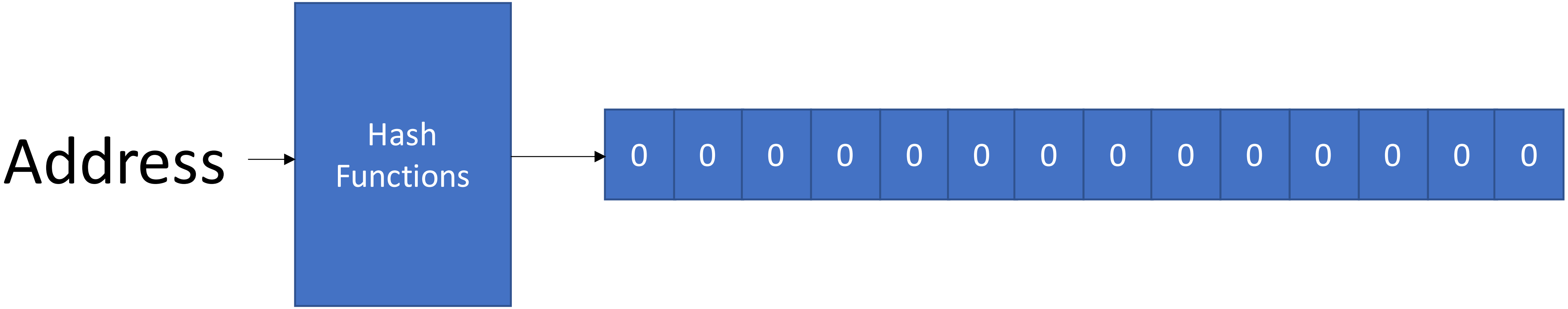
Bloom filter

- We want an efficient method to check whether a row has been recently accessed!
- We do not want false negatives

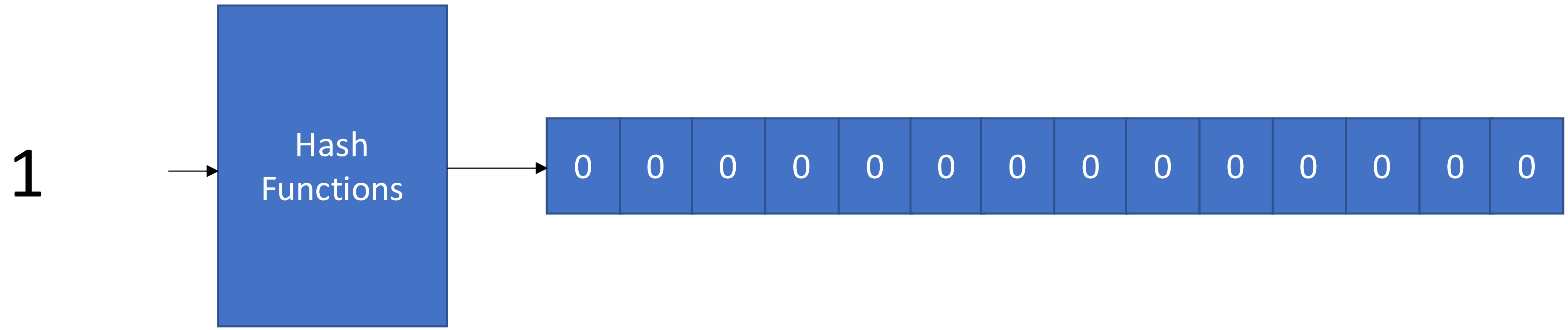
		Actual	
		Positive	Negative
Predicted	Positive	True Positive	False Positive
	Negative	False Negative	True Negative

We can use a Bloom filter

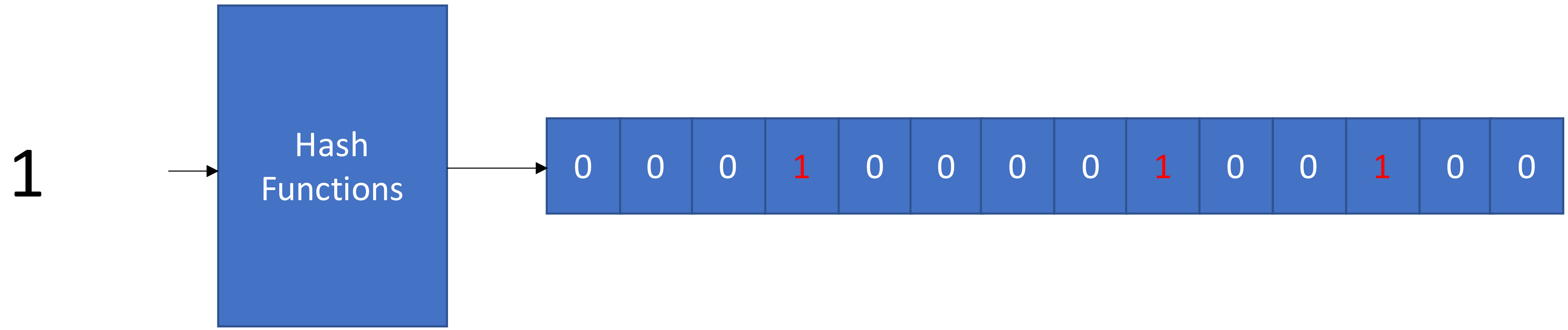
Bloom filter



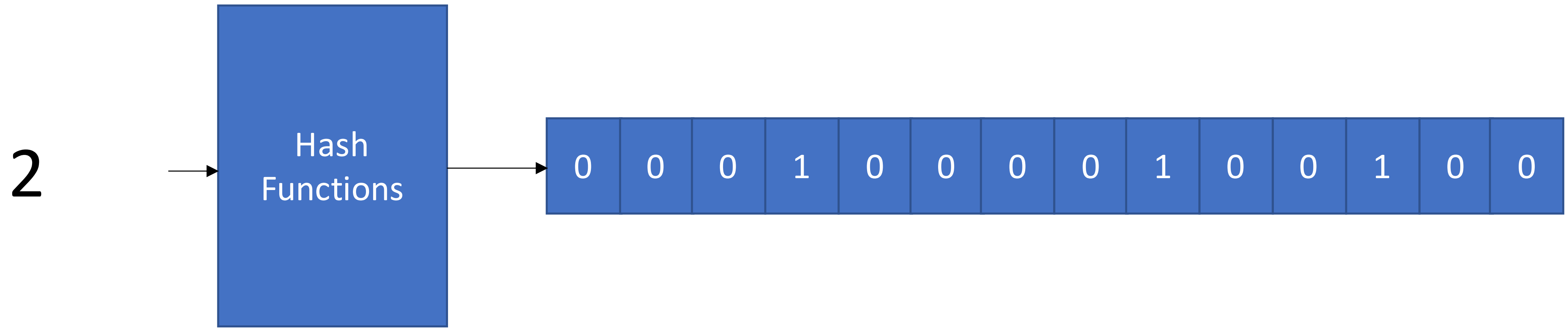
Bloom filter



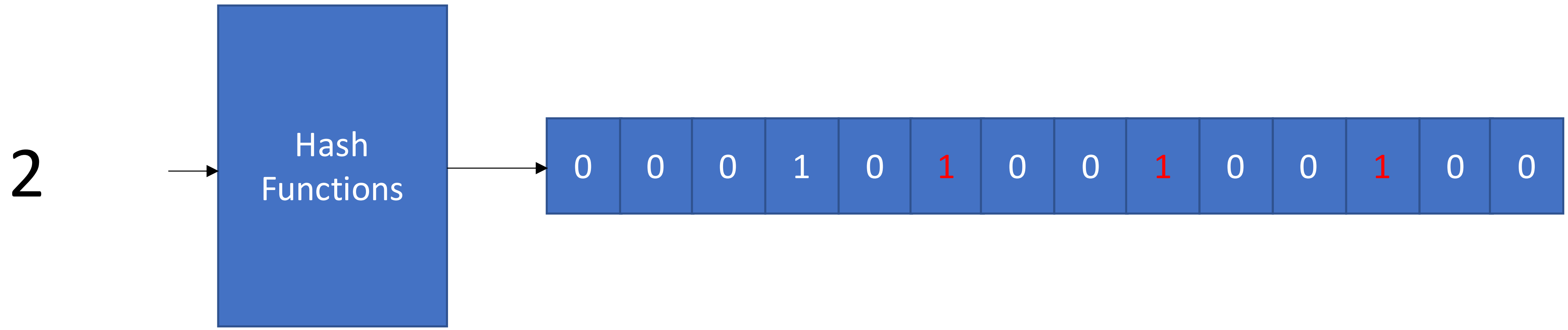
Bloom filter



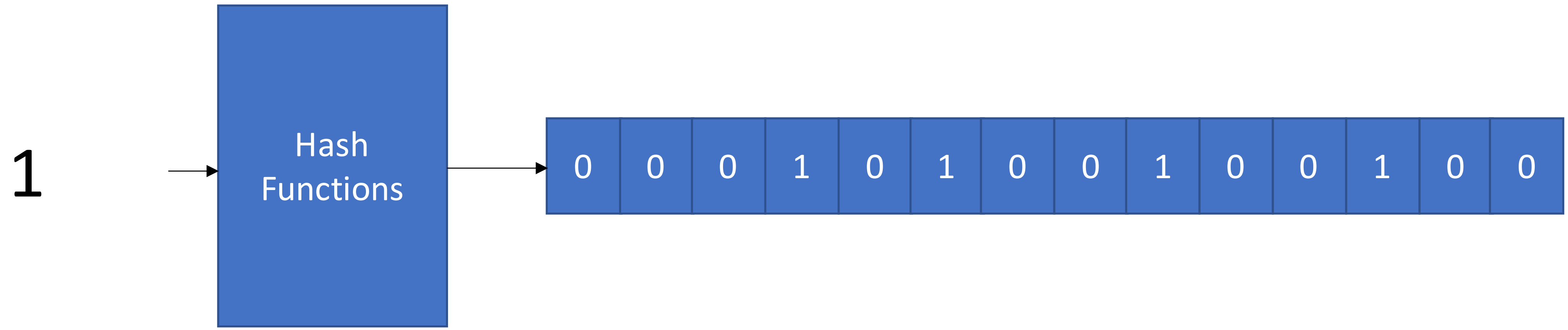
Bloom filter



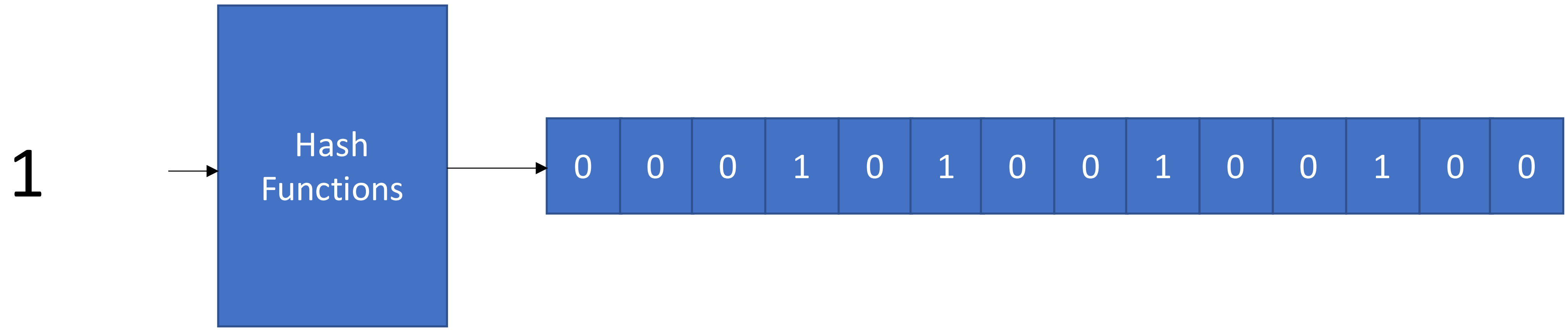
Bloom filter



Bloom filter

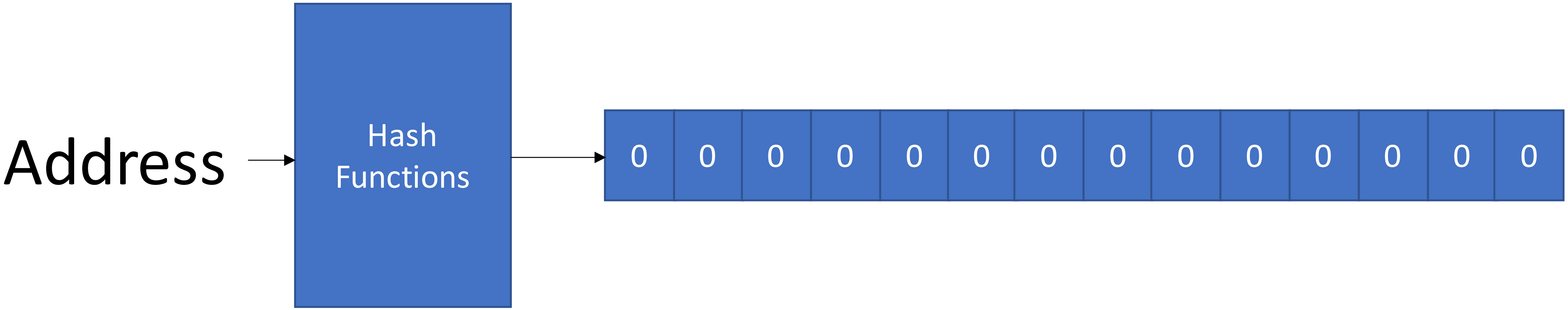


Bloom filter

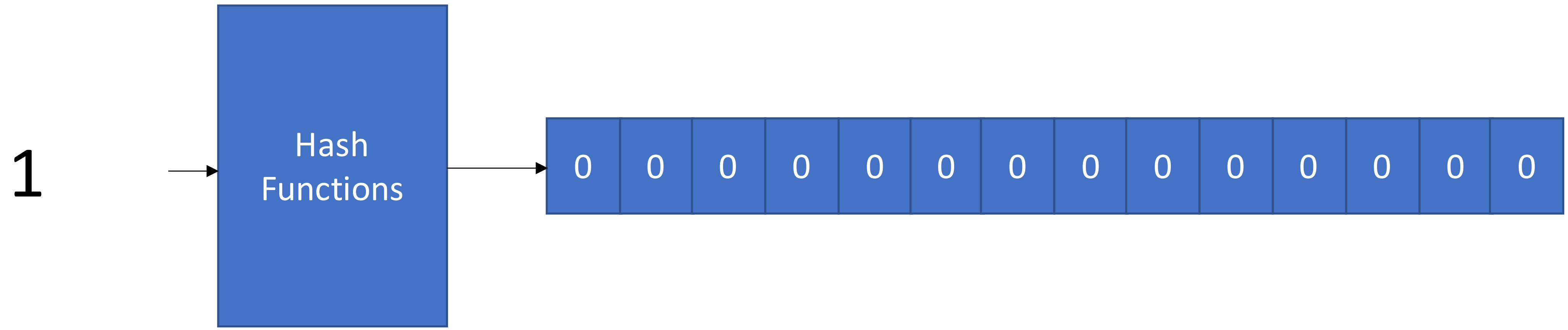


Now, we know if a row has been accessed! But how often have we accessed a row?

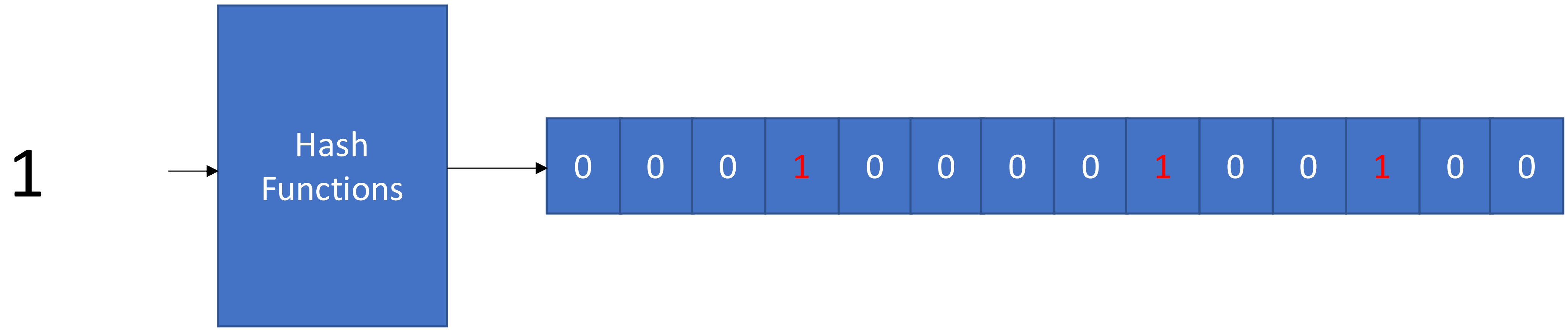
Counting Bloom filter



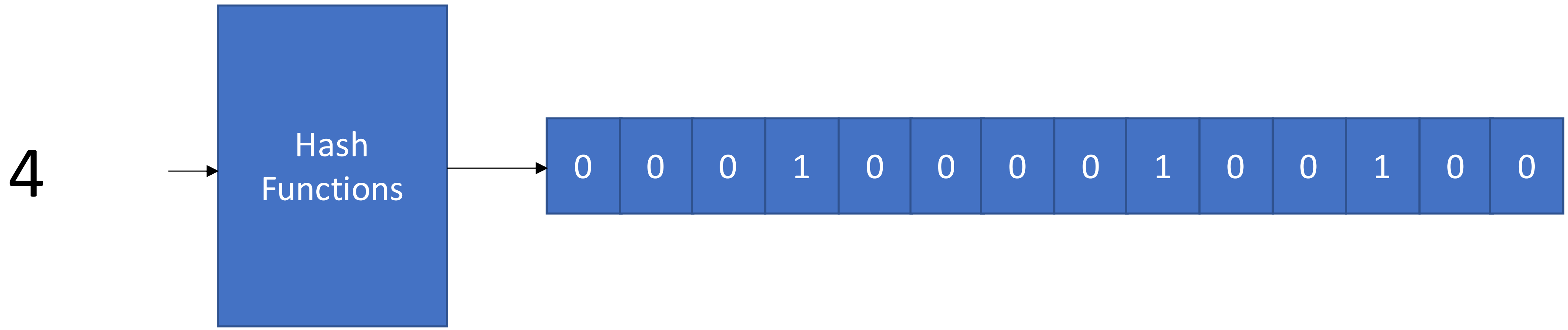
Counting Bloom filter



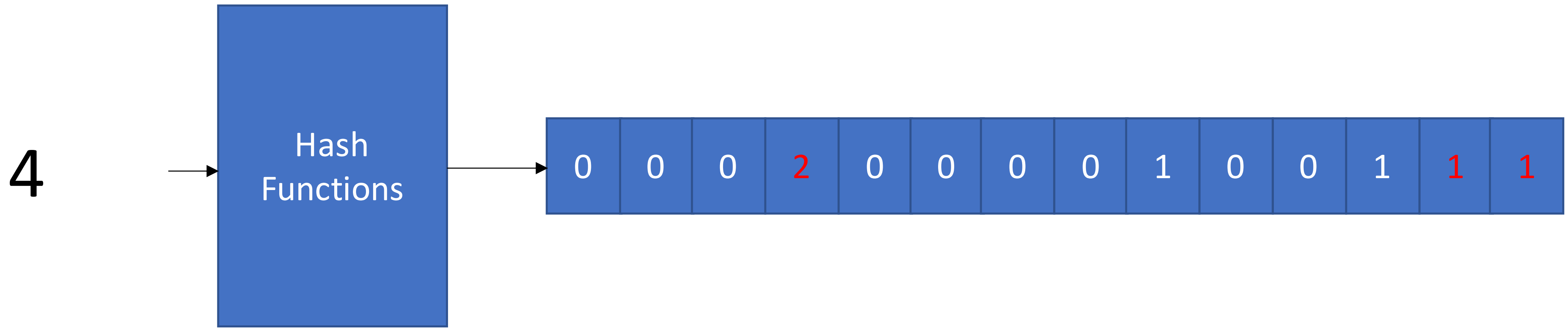
Counting Bloom filter



Counting Bloom filter

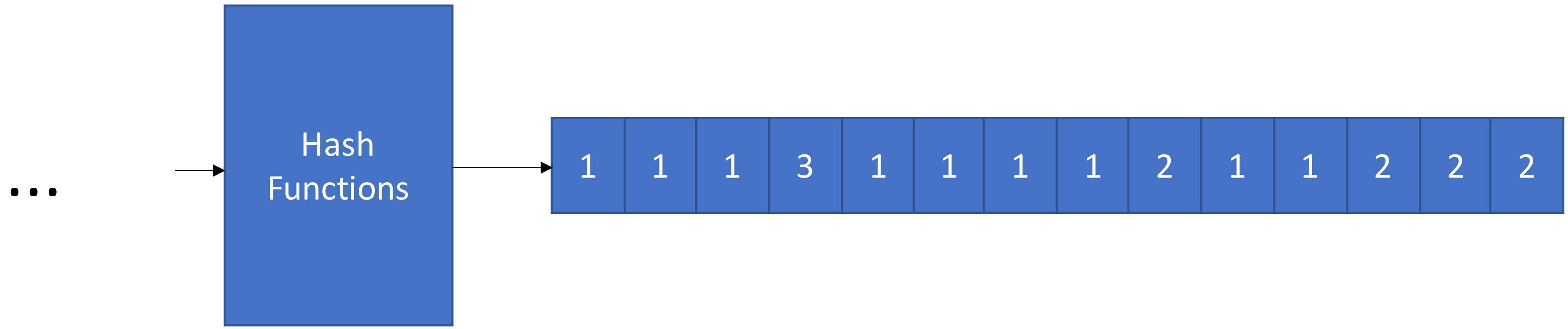


Counting Bloom filter

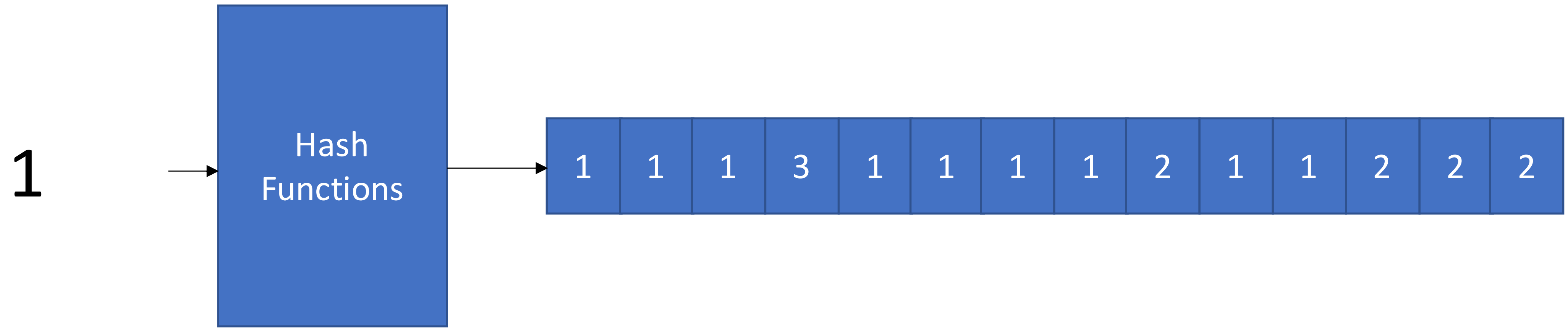


Counting Bloom filter

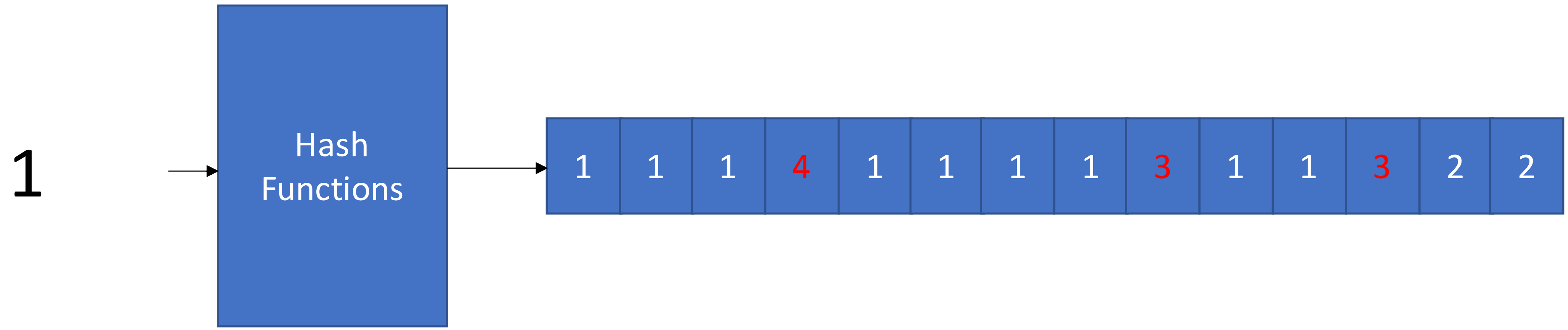
- We continue adding elements...



Counting Bloom filter

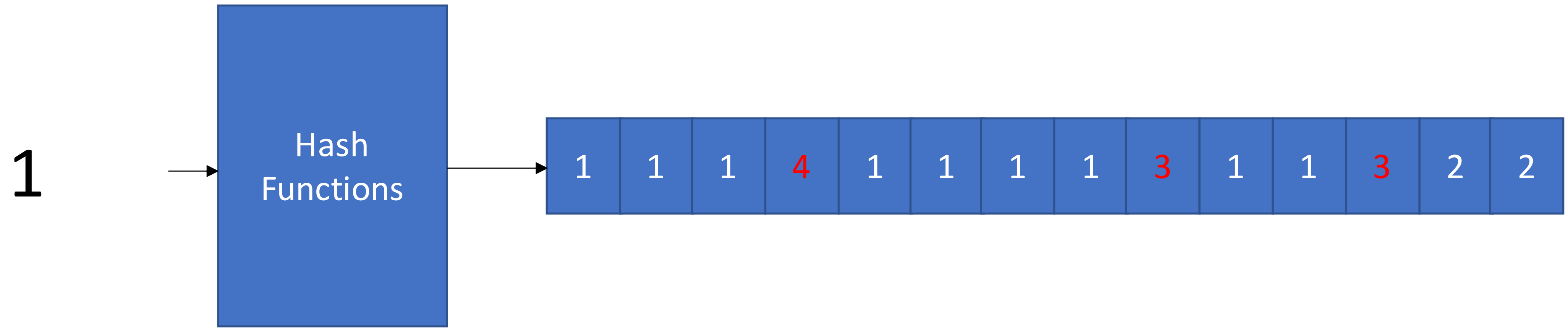


Counting Bloom filter



Counting Bloom filter

- How often have we accessed Row 1 at most?



Now, we have an upper bound!
We want to track the accesses in a specific interval. How can we clear the filter without losing all data?

Why are we using two Bloom filter?

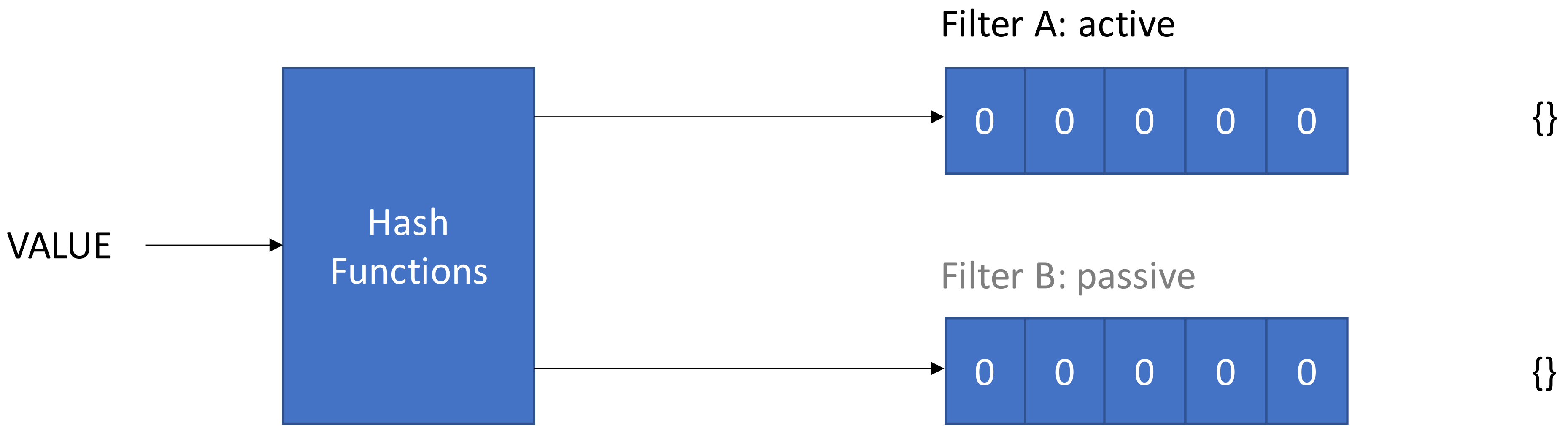
- We want to clear the filter in regular intervals to see the number of accesses in a specific interval.
- How do we achieve this?

Unified Bloom filter

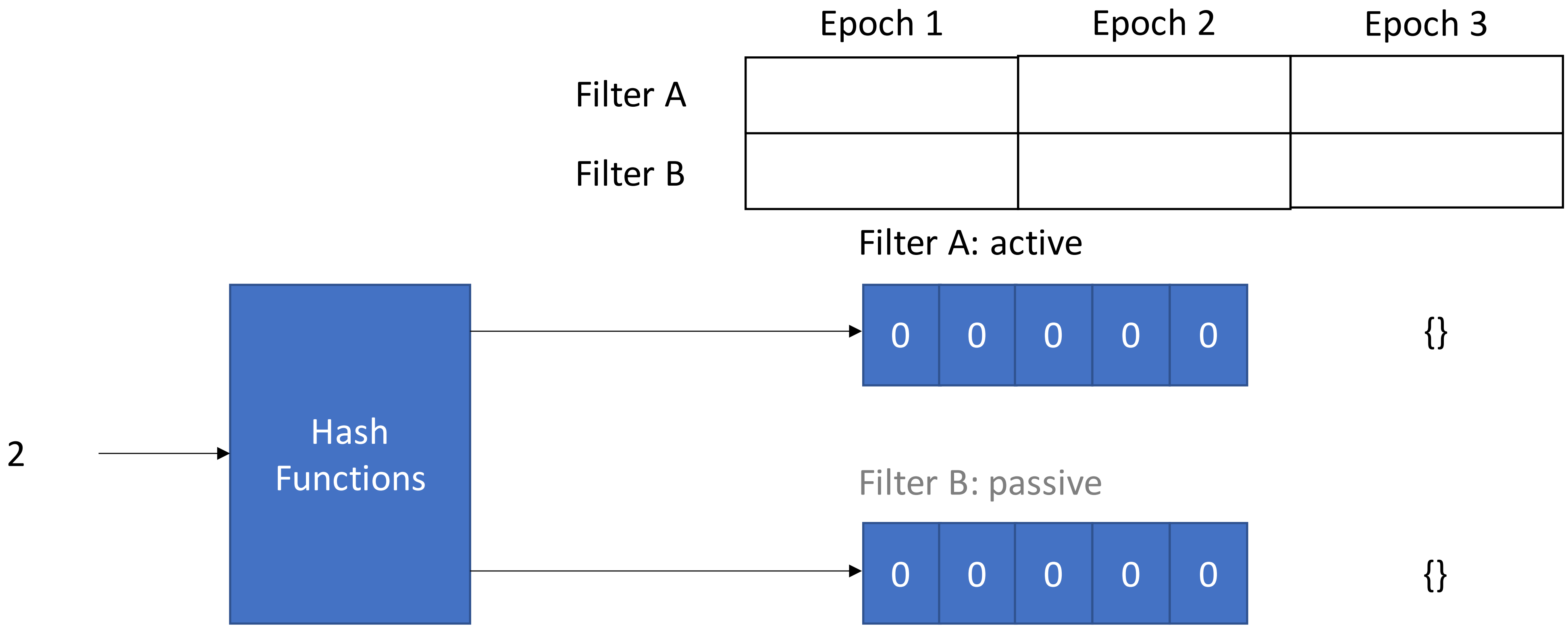
- All elements are inserted **into both** filter
- **Only one** filter is **active** and responses to queries
- Active filter clears array at the end of a specified time interval
- **Switches roles** after an interval

Unified Bloom filter

	Epoch 1	Epoch 2	Epoch 3
Filter A			
Filter B			

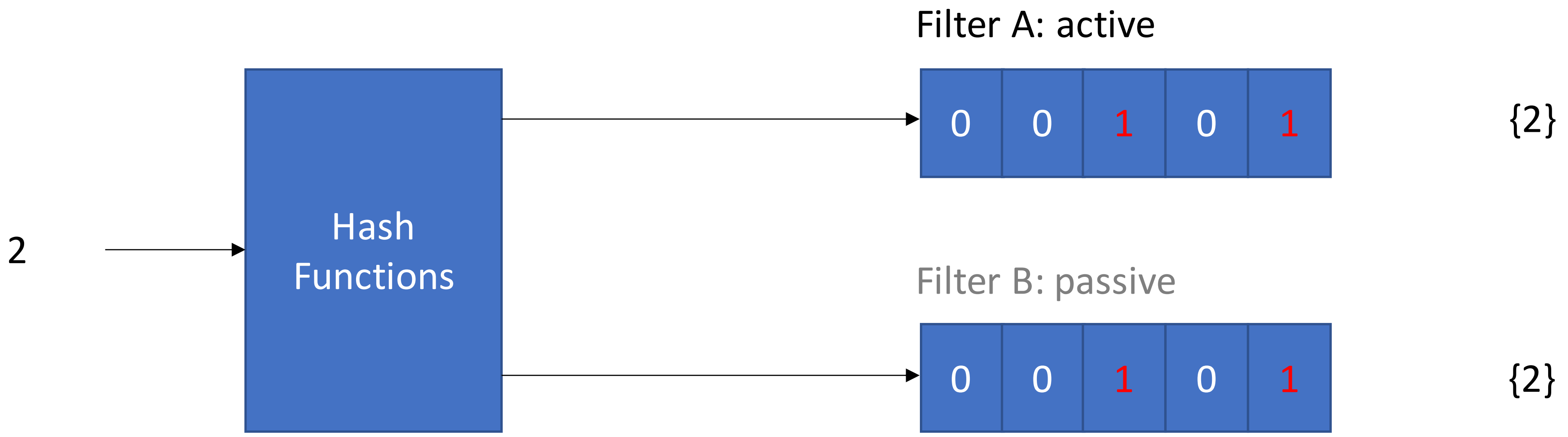


Unified Bloom filter

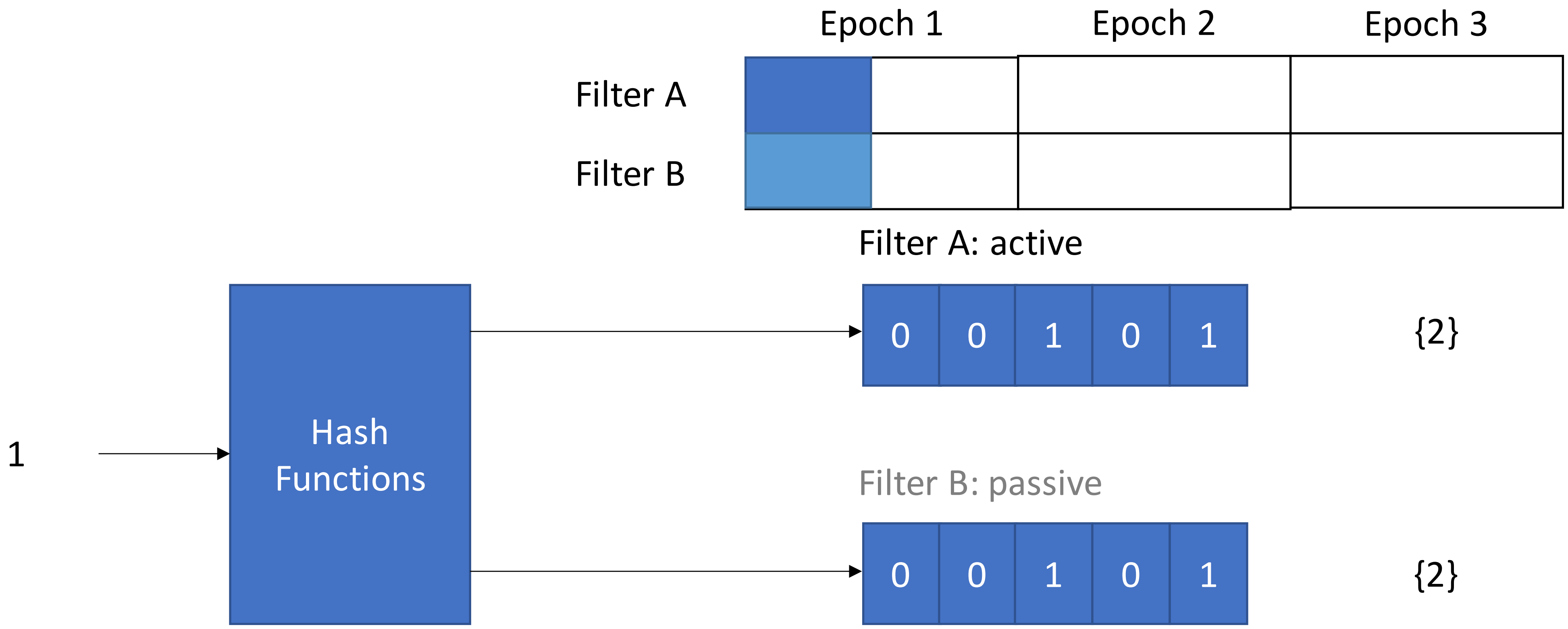


Unified Bloom filter

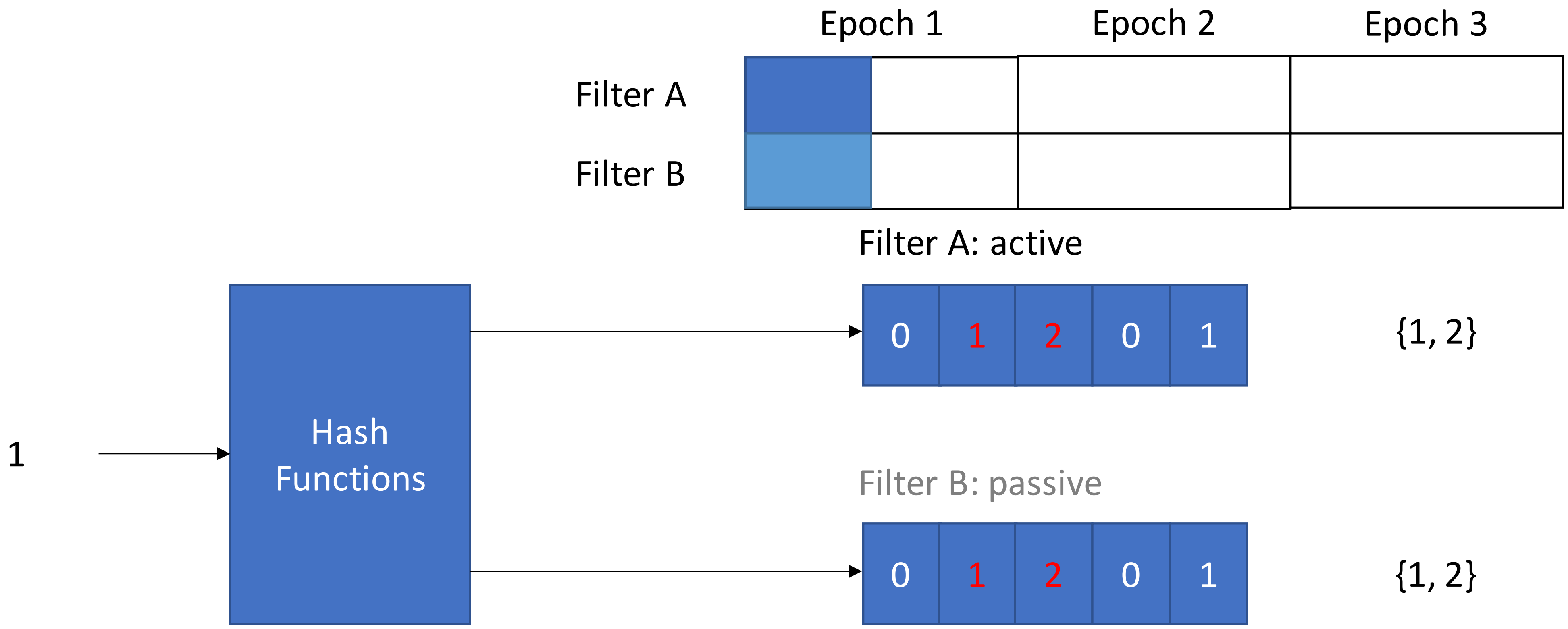
	Epoch 1	Epoch 2	Epoch 3
Filter A			
Filter B			



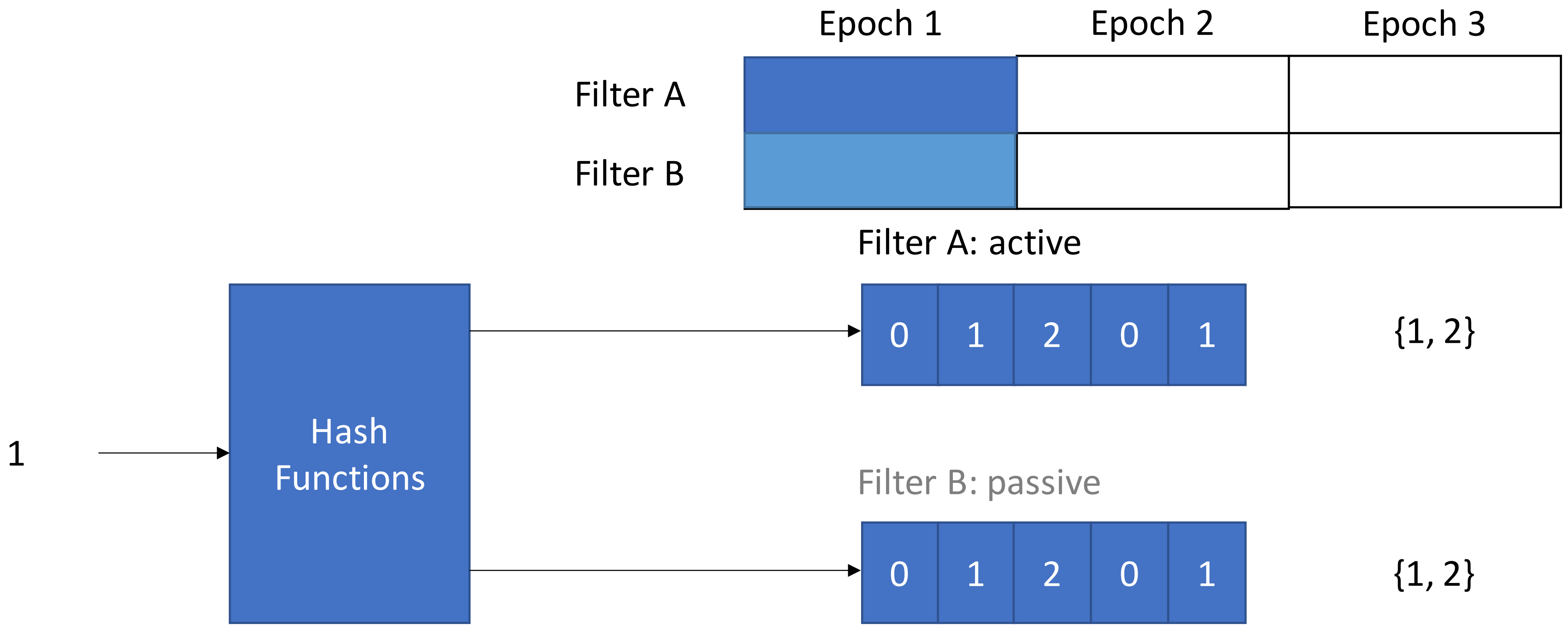
Unified Bloom filter



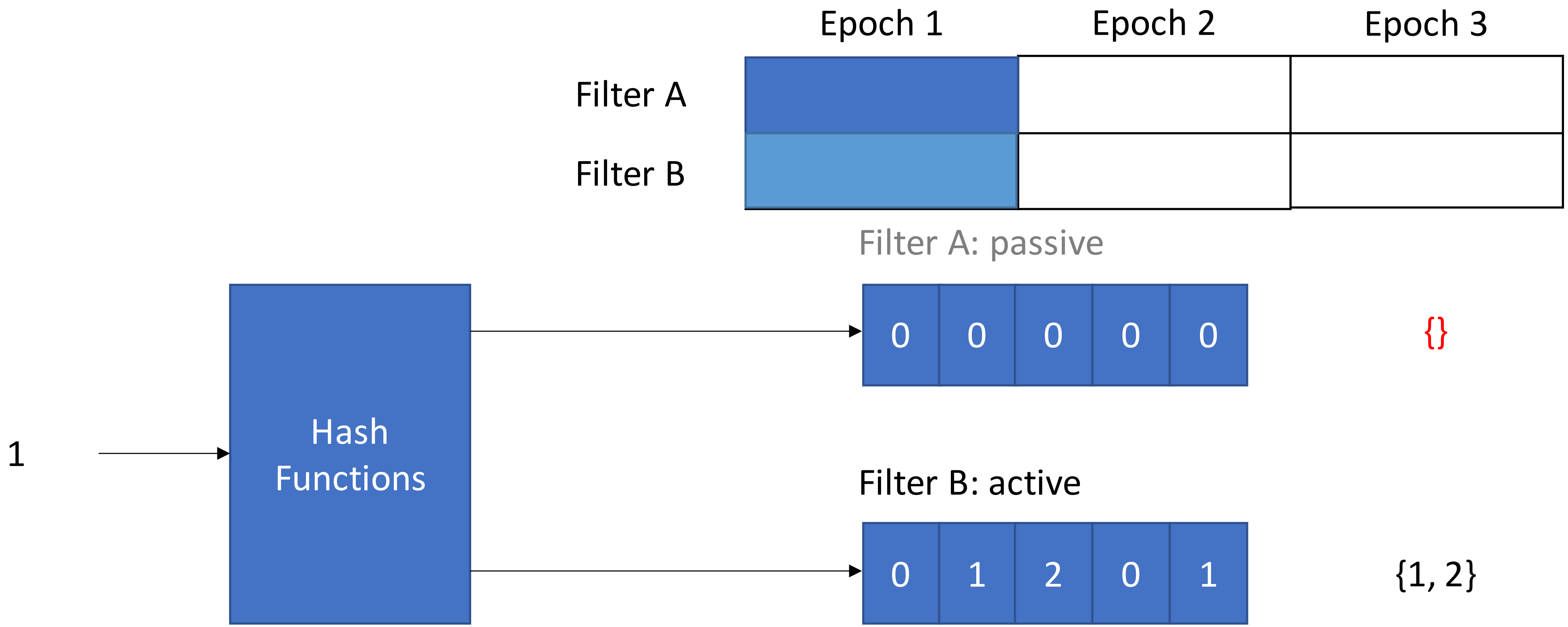
Unified Bloom filter



Unified Bloom filter



Unified Bloom filter



An unified Bloom filter gives us an upper bound of the last n accesses! How can we increase security to make the filter unpredictable?

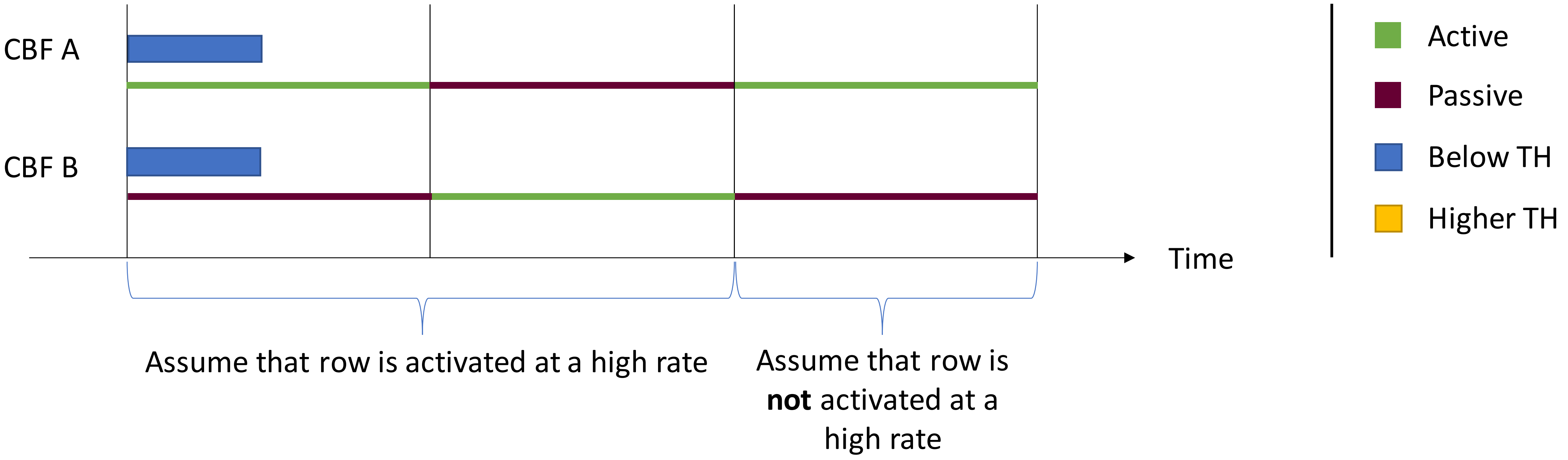
Dual counting Bloom filter

- Both filters use **different hash functions**
- Hash functions of the active filter are **altered** at the **end of each epoch**

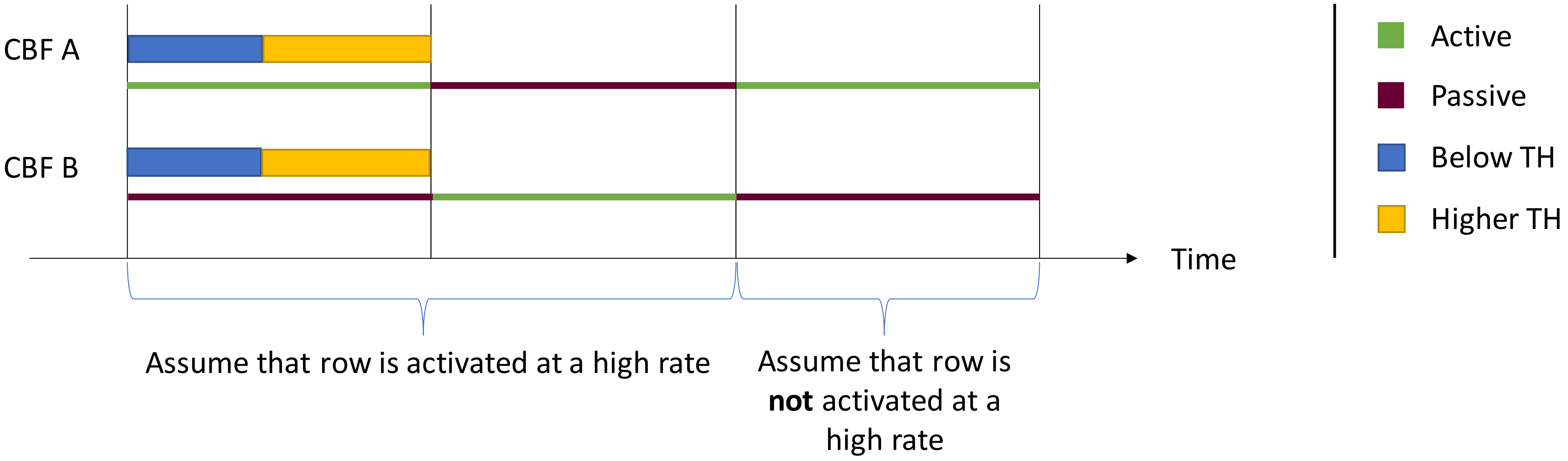
Bloom filter

- With a Bloom filter we can get **false positives**
- With a Bloom filter we cannot get **false negatives**
- A **counting Bloom filter** gives us an **upper bound** of accesses
- Additionally, by using a **unified Bloom filter**, we can track the last n insertions
- A **dual counting Bloom filter** increases security and makes the filter harder to attack

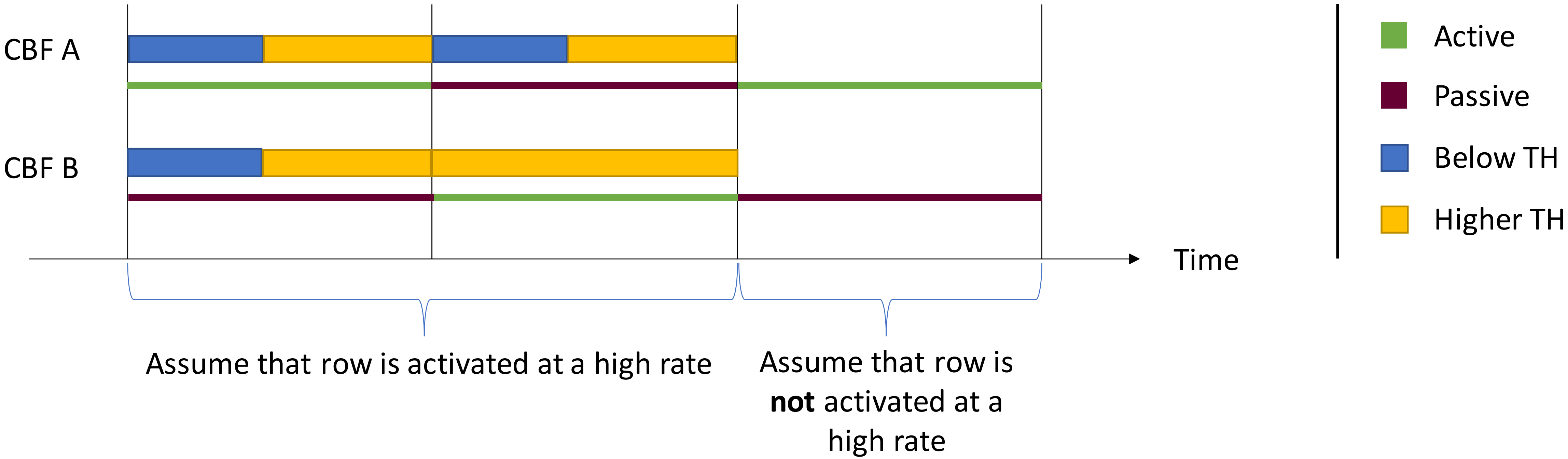
Blacklisting Logic



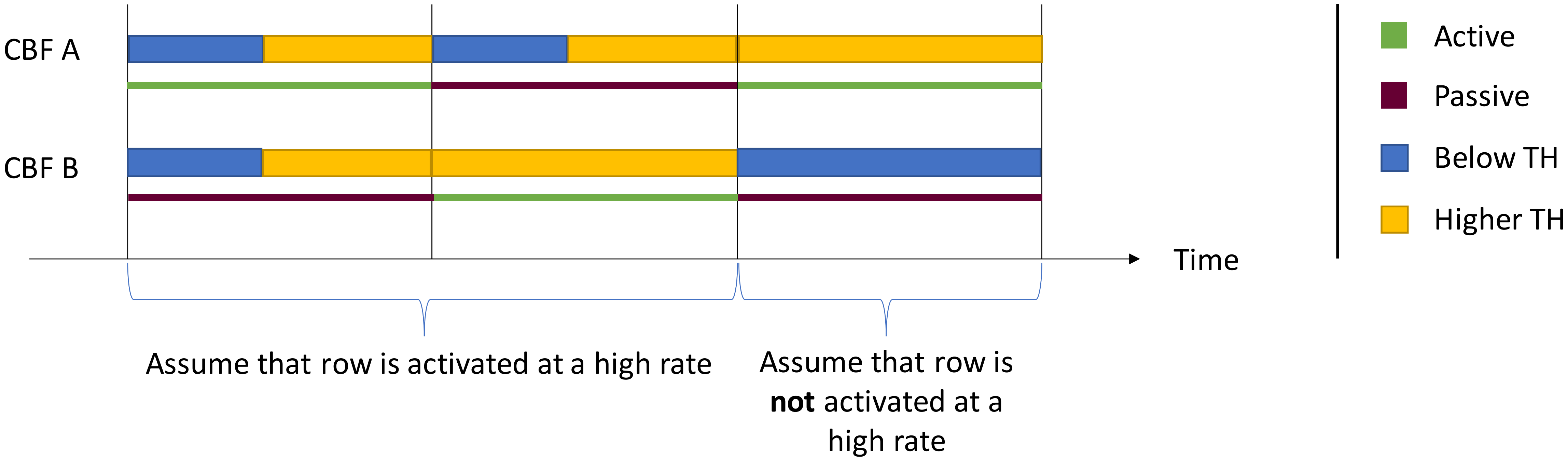
Blacklisting Logic



Blacklisting Logic

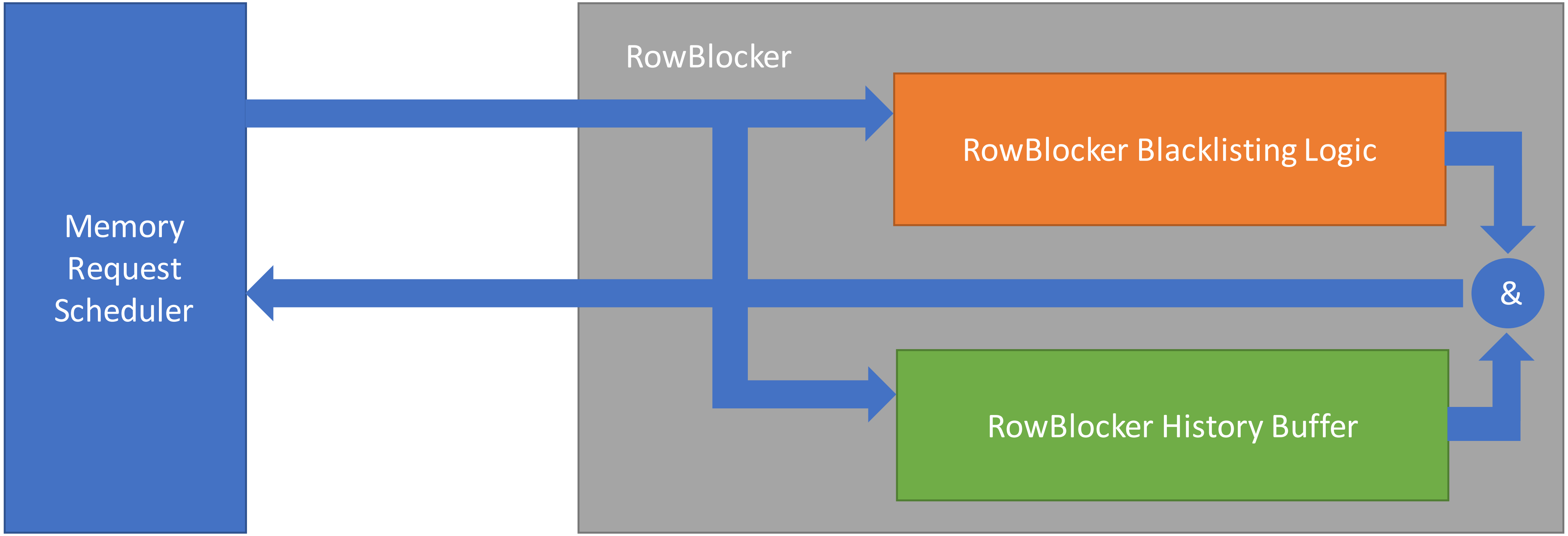


Blacklisting Logic



RowBlocker - Overview

- Throttles row activations
- Blacklists rows and delays activations of blacklisted rows



RowBlocker – History Buffer

- In order to induce a bit-flip, the aggressor row must be activated with a minimum frequency. If we keep a certain amount of time between each activation, we can guarantee RowHammer safety
- History Buffer writes most recently accesses in a (FIFO) queue
- Queue stores
  - Row ID: A rank-unique ID for all rows
  - Timestamp
  - Valid bit

Conclusion for RowBlocker

- Not possible to activate a row often enough to induce a bit-flip
- A row access is delayed when the row is blacklisted and was accessed in the last time window

AttackThrottler

SAFARI
SAFARI Research Group

AttackThrottler

- Reduces the performance degradation and energy wastage during a RowHammer attack
- A RowHammer attack keeps activating blacklisted rows

RowHammer Likelihood Index (RHLI)

- The RHLI defines the possibility of a RowHammer attack
- A benign application has index 0
- A malicious software has index 1

⇒RHLI is larger when a thread‘s access pattern is more similar to a RowHammer attack

$$\Rightarrow \text{RHLI} = \frac{\text{\#blacklisted row activations a thread performed to a DRAM bank}}{\text{maximum \# times a blacklisted row can be activated in a protected system}}$$

AttackThrottler

- If the RHLI is large, then we limit the thread‘s bandwidth

f(x) = 1/x

Identifying attacker threads

- We use an active and a passive counter to track the accesses for every <thread, bank> pair
- Active counter is used for calculating the RHLI
- RHLI could be used by an antivirus to find malicious software

Evaluation with other techniques

- We compare the techniques in the following categories:
  - Hardware complexity analysis (scalable and low cost)
  - Efficiency in terms of performance and energy usage

Other techniques - Overview

- 3 probabilistic mechanisms
  - PARA 2014 (Yoongu Kim)
  - ProHIT 2017 (Mungyu Son)
  - MRLoc 2019 (Jung Min You)
- 3 deterministic mechanisms
  - CBT 2018 (Seyed Mohammed Seydzadeh)
  - TWiCe (Eojin Lee) 2019
  - Graphene (Yeonhong Park) 2020

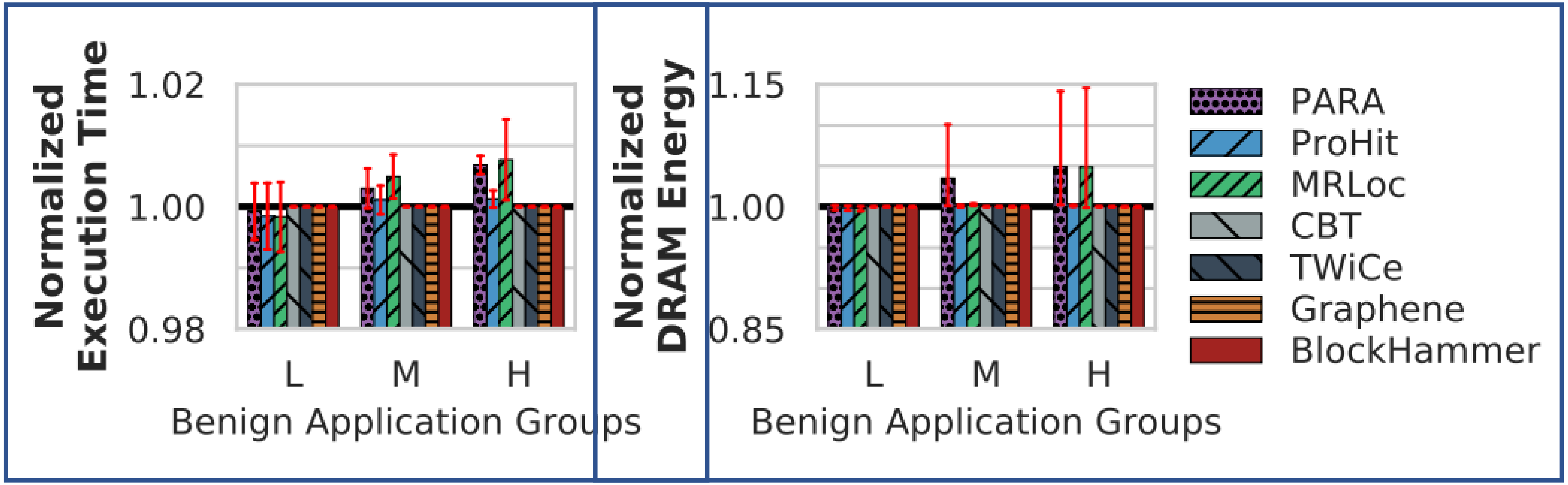
Hardware complexity

Mitigation Mechanism	N_{RH}=32K*						N_{RH}=1K					
	SRAM KB	*CAM* KB	*Area* mm^2	*Area* % CPU	Access Energy (pJ)	Static Power (mW)	*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 ×.

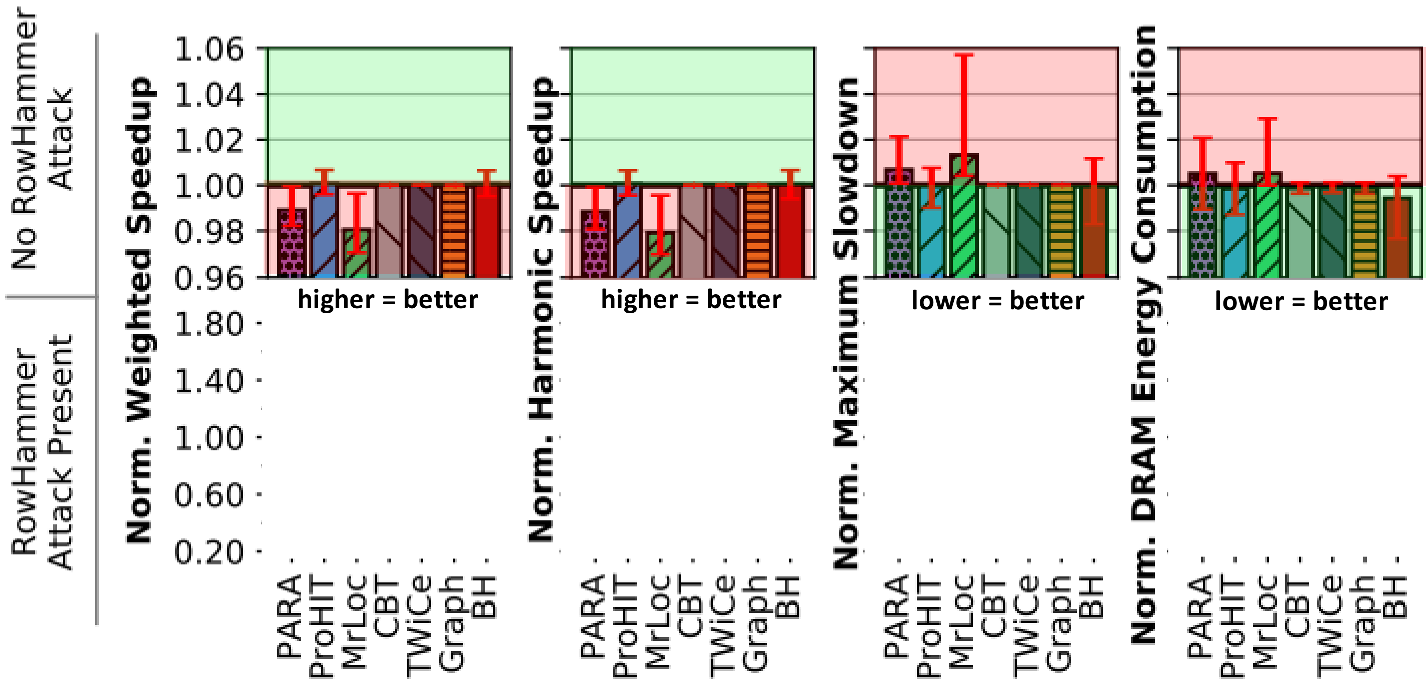
- PARA, PRoHIT, MRLoc -> probabilistic methods and are therefore very area-efficient
- If we reduce the threshold, then BlockHammer scales better than the other techniques

Evaluation – Single-Core



Deterministic approaches do not have a high overhead because benign applications do not reach the threshold!

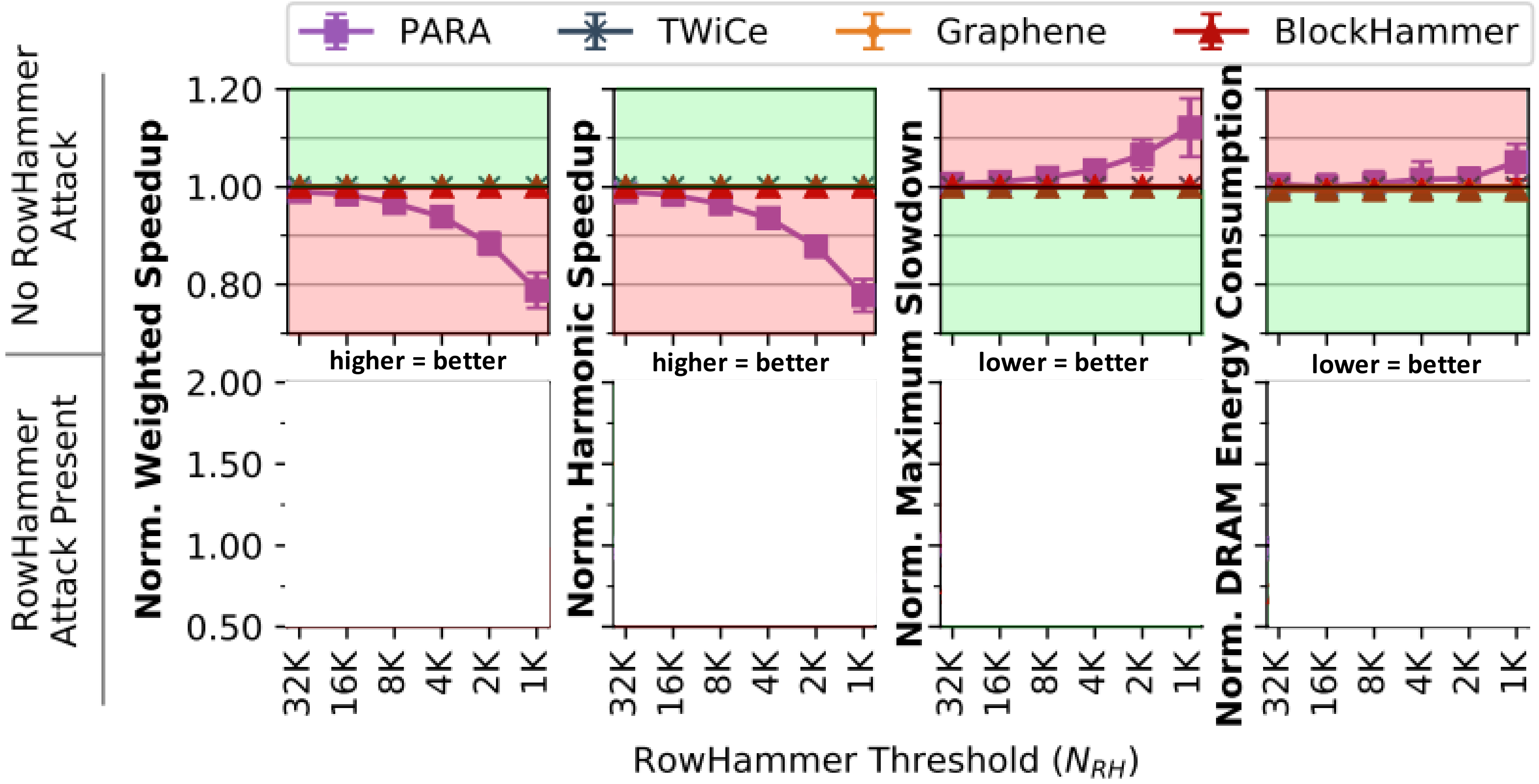
Evaluation – Multi-Core



BlockHammer has competitive performance and energy usage when no attack is present

Under an attack, BlockHammer has much higher performance and much lower DRAM energy consumption!

Evaluation – Energy consumption



BlockHammer has negligible performance and energy overheads, even if chips become more vulnerable to RowHammer!

BlockHammer has significantly better performance and lower energy consumption as RH worsens

Conclusion

- Most mechanisms to prevent RowHammer bit-flips **do not work perfectly** and do **not scale** accordingly
- Many solutions often require **knowledge** of or modification to DRAM **internals**
- Finds a scalable and efficient mechanism that works without any knowledge of or modification to DRAM internals
- BlockHammer consists of two parts
  - **RowBlocker**: Tracks all row activations and limits potentially unsafe accesses
  - **AttackThrottler**: Calculates RHLI and limits potential attacker's bandwidth
- When there is no attack, then **BlockHammer** is competitive with other mechanisms
- If there it an attack, then BlockHammer **outperforms** the other mechanisms

Strengths

- BlockHammer has high **potential** in the future, as it **scales well** with upcoming DRAM chips
- Keeps high **efficiency** when running benign and attacking threads
- BlockHammer is **compatible** with all DRAM chips
- Creates an interface for other applications
  - I.e., gives an antivirus access to the RHLI

Weaknesses

- Is implemented in memory controller -> Cannot be implemented in **already manufactured** chips
- Potentially opens a door for other attacks. An attacker could use the **false positive rate** to decrease performance
- What is the impact on virtual machines? AttackThrottler would start limiting bandwidth
  - => Open door for potential denial of service attacks

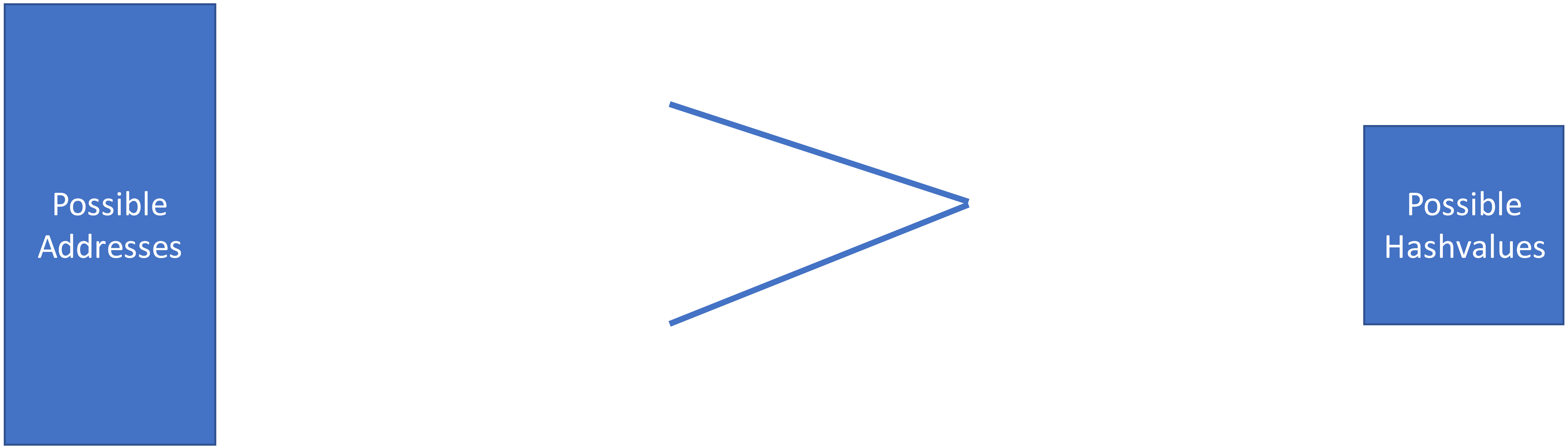
Discussion

Discussion

- Do you see any potential attacks that can be made possible by using BlockHammer? Do you see any solution to prevent the presented attacks?

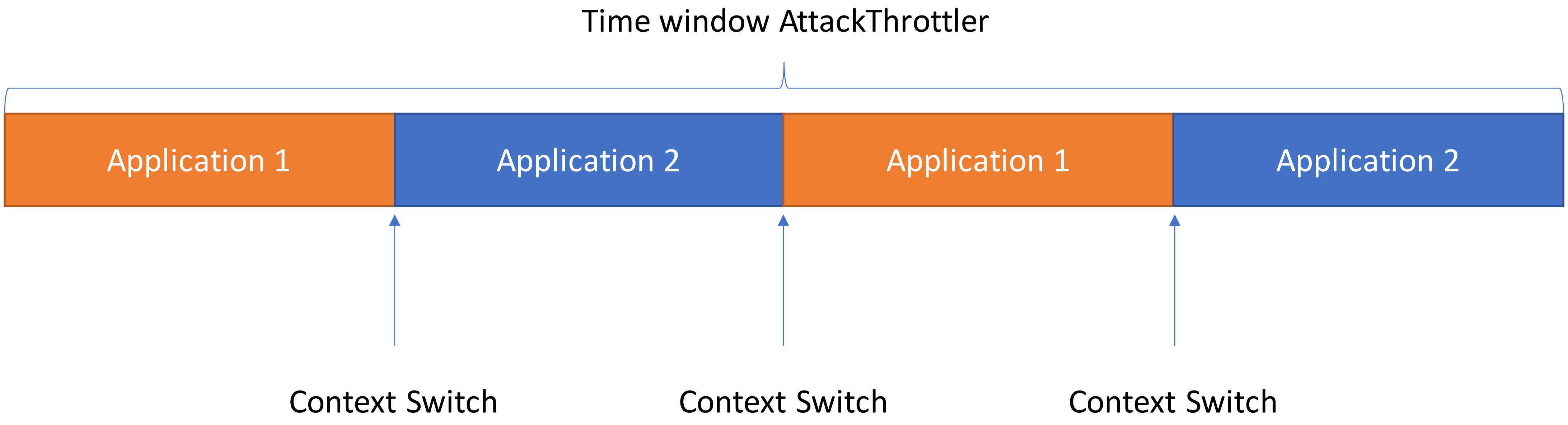
Discussion – Possible Attack

- Blacklist specific rows



Discussion – Possible Attack

- We attack a system using AttackThrottler



Application 1 is hammering a row
Application 2 is a benign application

Discussion

- Do you see any potential attacks that can be made possible by using BlockHammer? Do you see any solution to prevent these attacks?

Discussion

- We have seen many potential solutions against RowHammer. Do you have an idea how we can improve BlockHammer even further?
- Should we find a solution in software? What are the main differences between a solution in software versus hardware?
- Should we implement BlockHammer in DRAM?

Discussion

- We could give the operating system access to the RHLI
- What can we do with this value?
  - For example: Improve database for antivirus
  - Optimize caching

Big thanks to the mentors!

- Abdullah Giray Yaglikci
- Ataberk Olgun
- Konstantinos Kanellopoulos