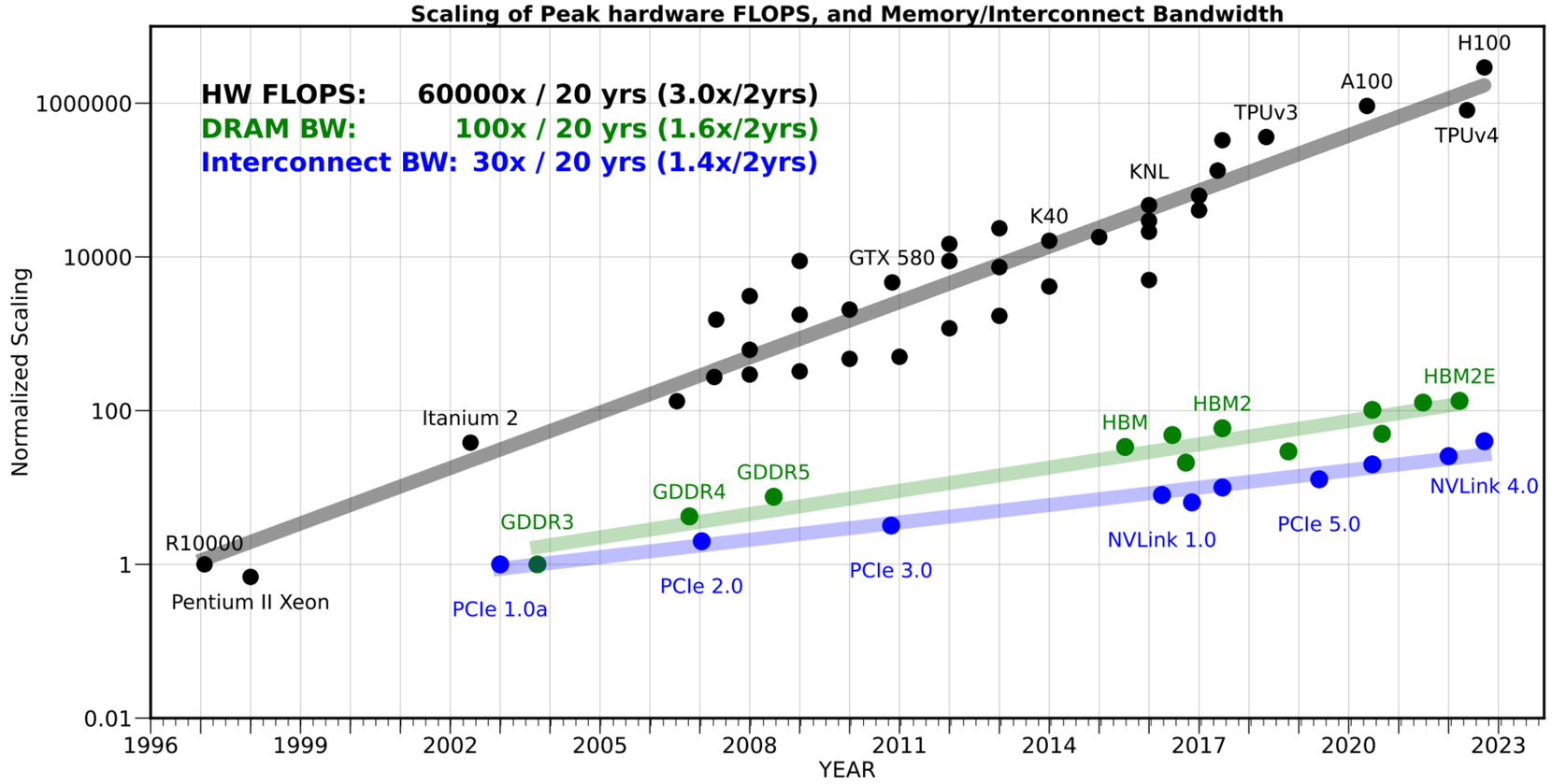
Rethinking Data Structures for Scalable Performance

Filters, Hash tables, Trees, Graphs, Vectors…

Prashant Pandey, Northeastern University, Boston USA
https://prashantpandey.github.io/

Efficient scaling needs efficient data movement



Memory wall

Gholami, Yao, Kim, Hooper, Mahoney, Keutzer IEEE Micro 2024

Why reduce data movement?

- Running time of an algorithm is sum of 3 terms:
  - # flops * time_per_flop
  - # words moved / bandwidth
  - # messages * latency

} **Data movement**

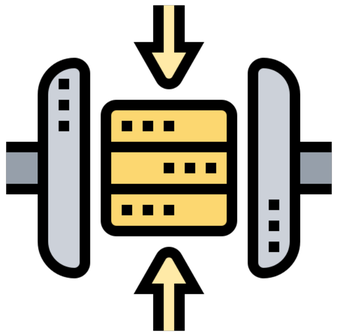
- time_per_flop(γ) << 1/bandwidth(β) << latency(α)

Slide courtesy: Jim Demmel, UC Berkeley

Data movement also dominates energy costs

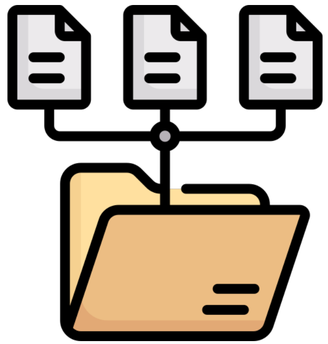
Operation	Energy (pJ)	Relative cost
64-bit integer Op	~1 pJ	1x
L1 cache access	~26 pJ	26x
L2/L3 cache access	50-100 pJ	100x
DRAM access	~1200 pJ	1200x
NVMe SSD	~10K-100K pJ	10^5x
HDD	~100K-1M pJ	10^6x

Three approaches to build scalable data systems



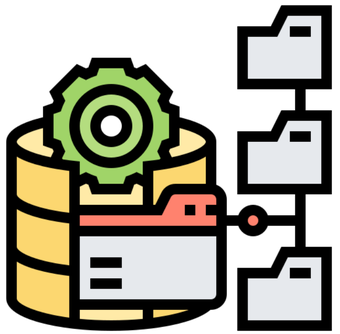
Compress it

Goal: make data smaller to fit inside fast memory



Organize it

Goal: organize data in a I/O friendly way

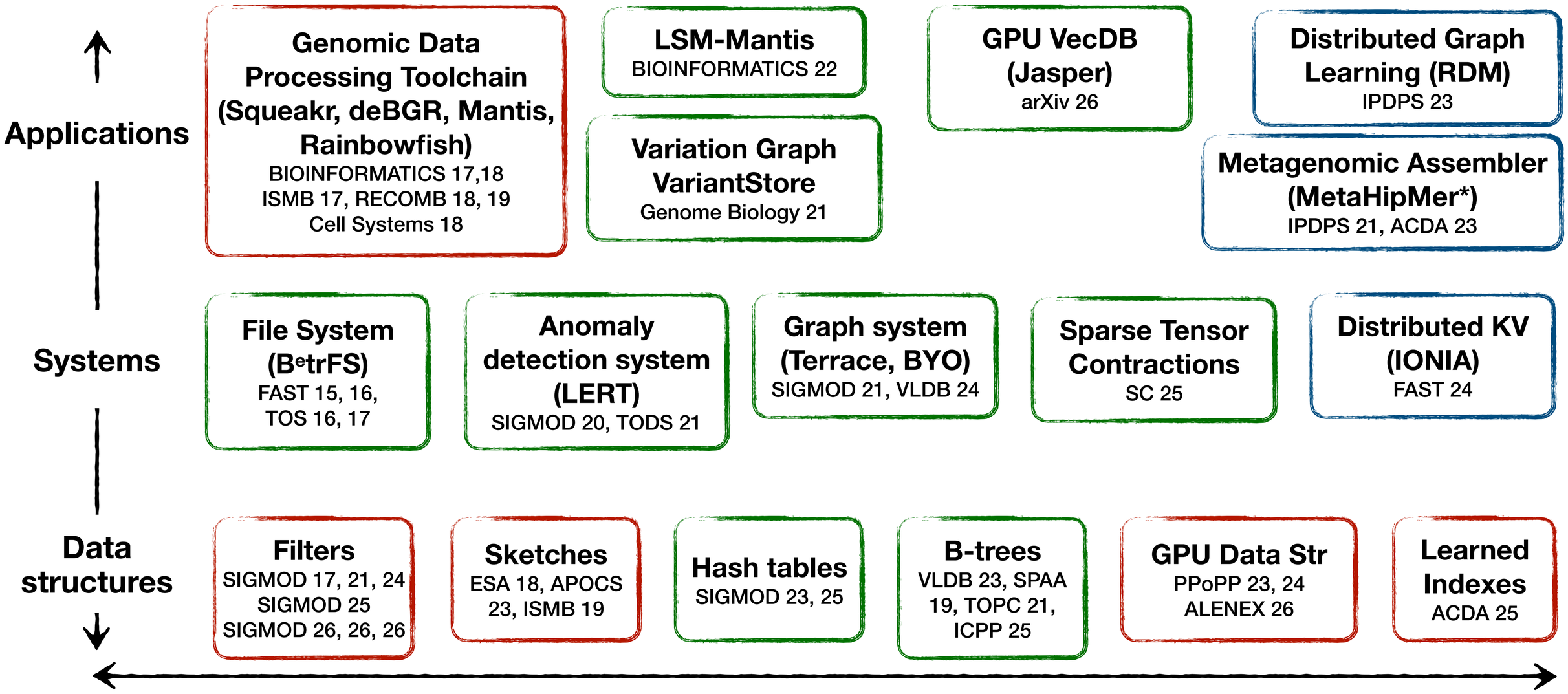


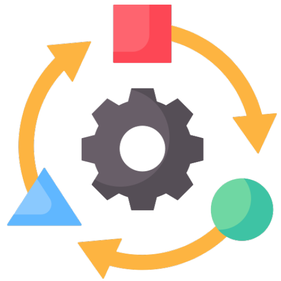
Distribute it

Goal: distribute data & reduce inter-node communication

Vertically integrated research

COMPRESS ORGANIZE DISTRIBUTE



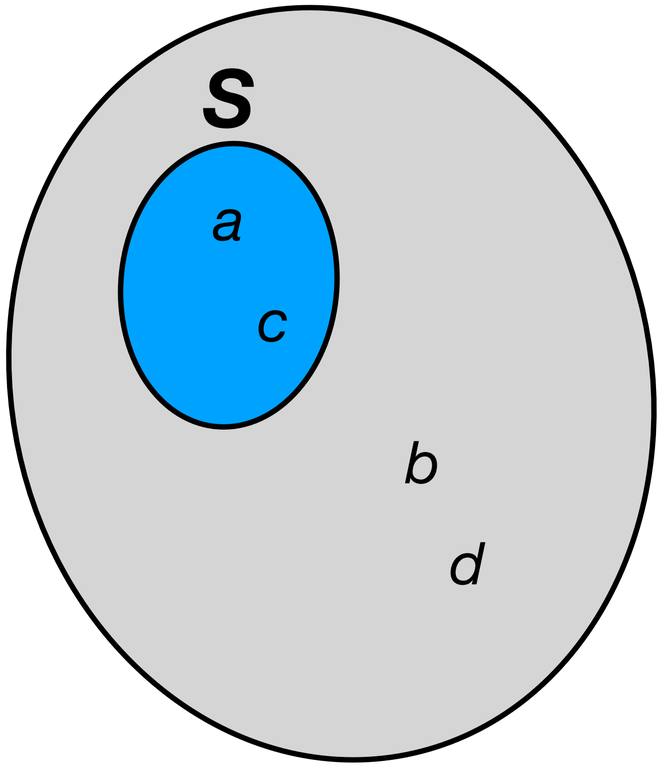


AdaptiveQF [SIGMOD 2025]

Wen, McCoy, Tench, Tagliavini, Bender, Conway, Farach-Colton, Johnson, Pandey

Dictionary data structure

A dictionary maintains a set *S* from universe *U*



membership (*a*): ✓

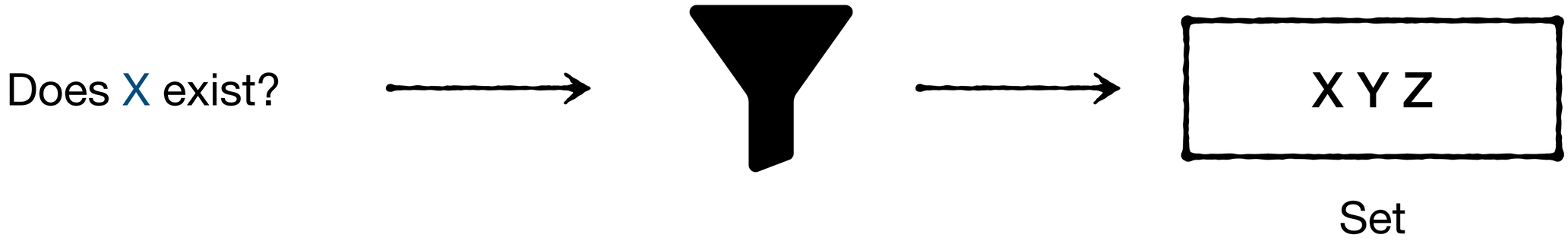
membership (*b*): ✗

membership (*c*): ✓

membership (*d*): ✗

A dictionary supports membership queries on *S*

A filter is an approximate dictionary

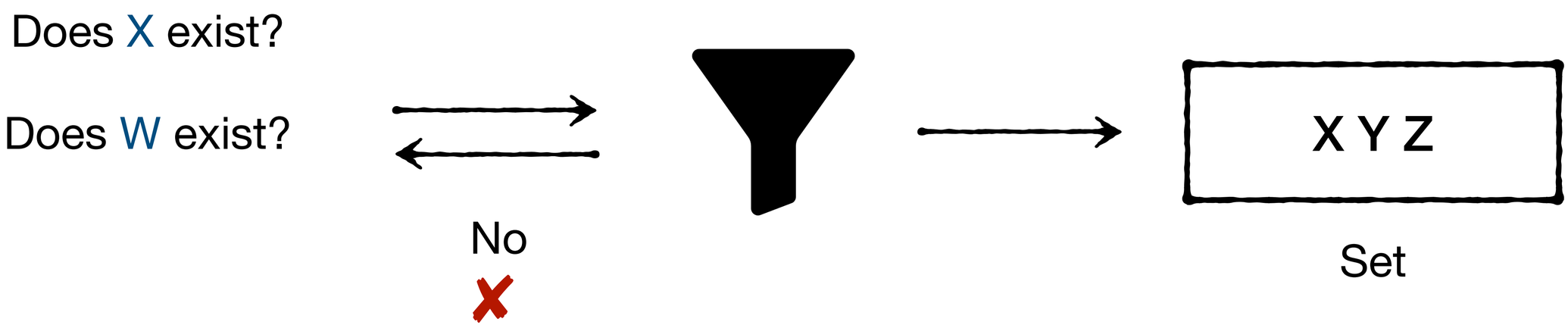


A filter compactly represents a set by trading off accuracy for space efficiency

A filter is an approximate dictionary

A filter compactly represents a set by trading off accuracy for space efficiency

A filter is an approximate dictionary



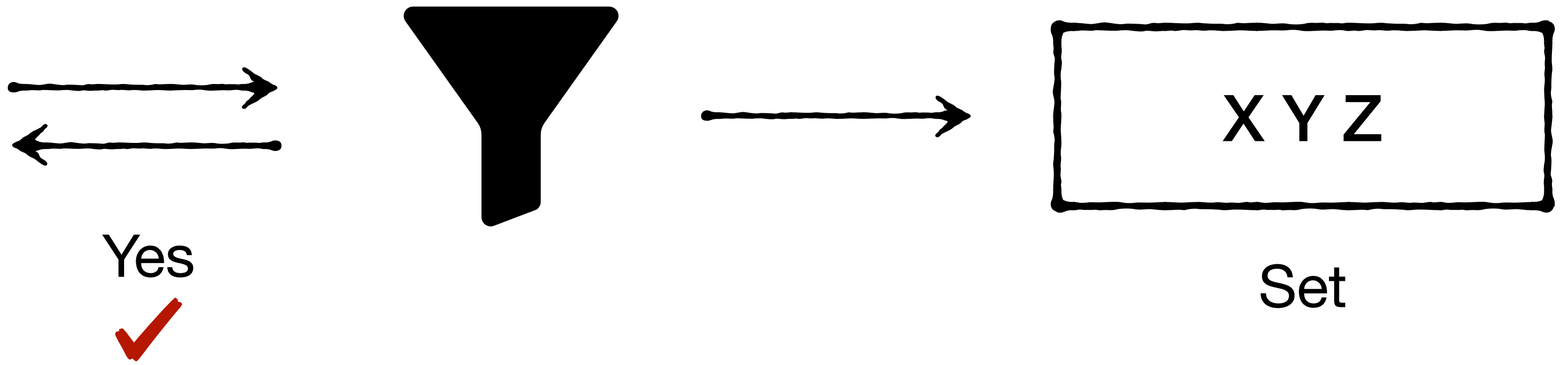
A filter compactly represents a set by trading off accuracy for space efficiency

A filter is an approximate dictionary

Does X exist?

Does W exist?

Does A exist?



A filter compactly represents a set by trading off accuracy for space efficiency

A filter guarantees a false-positive rate ϵ

$q =$ **query item** $S =$ **set of items**

if q ∈ S, return **True** with probability 1 — true positive

if q ∉ S, return {

- **False** with probability $> 1 - \epsilon$ — true negative
- **True** with probability $\leq \epsilon$ — false positive

One-sided errors

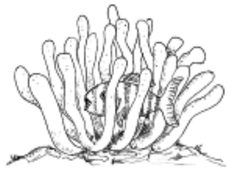
False positives with tunable probability

False-positives enable filters to be compact

n = **number of items** U = **universe of items**

space $\geq n \log(1/\epsilon)$

Small



Filter

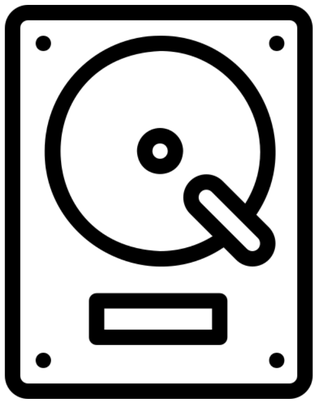
space $= \Omega(n \log(|U|)$

Large

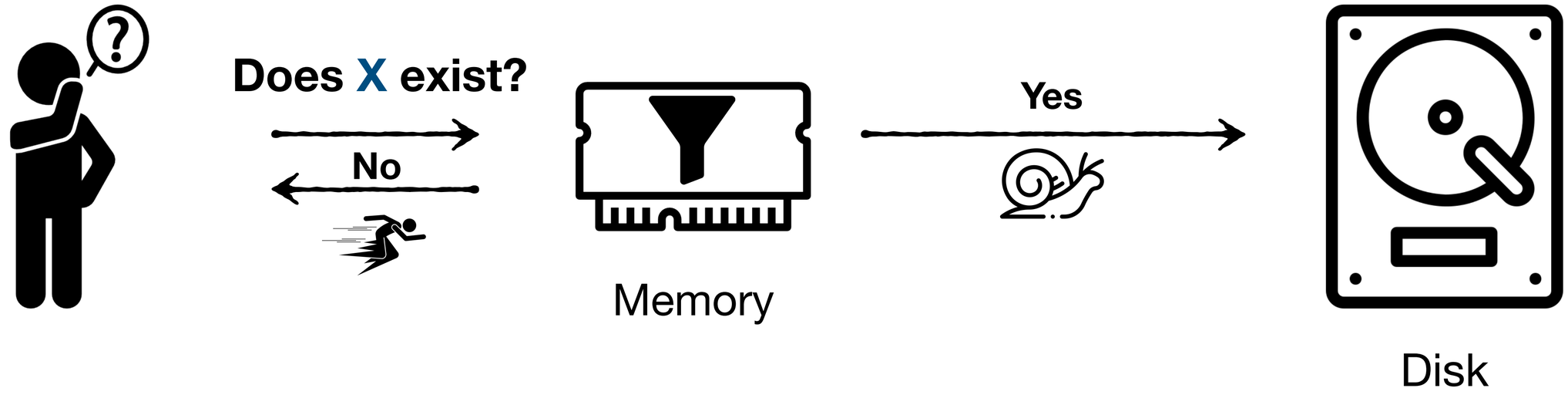
Hash table/Tree

For $\epsilon = 2\,\%$, filters require ~1 Byte/item. Hash table/Tree can take >8-16 Byte/item.

Does X exist?



Disk



Saves unnecessary disk accesses and network hops

Classic filter: The Bloom filter (BF) [Bloom 70]

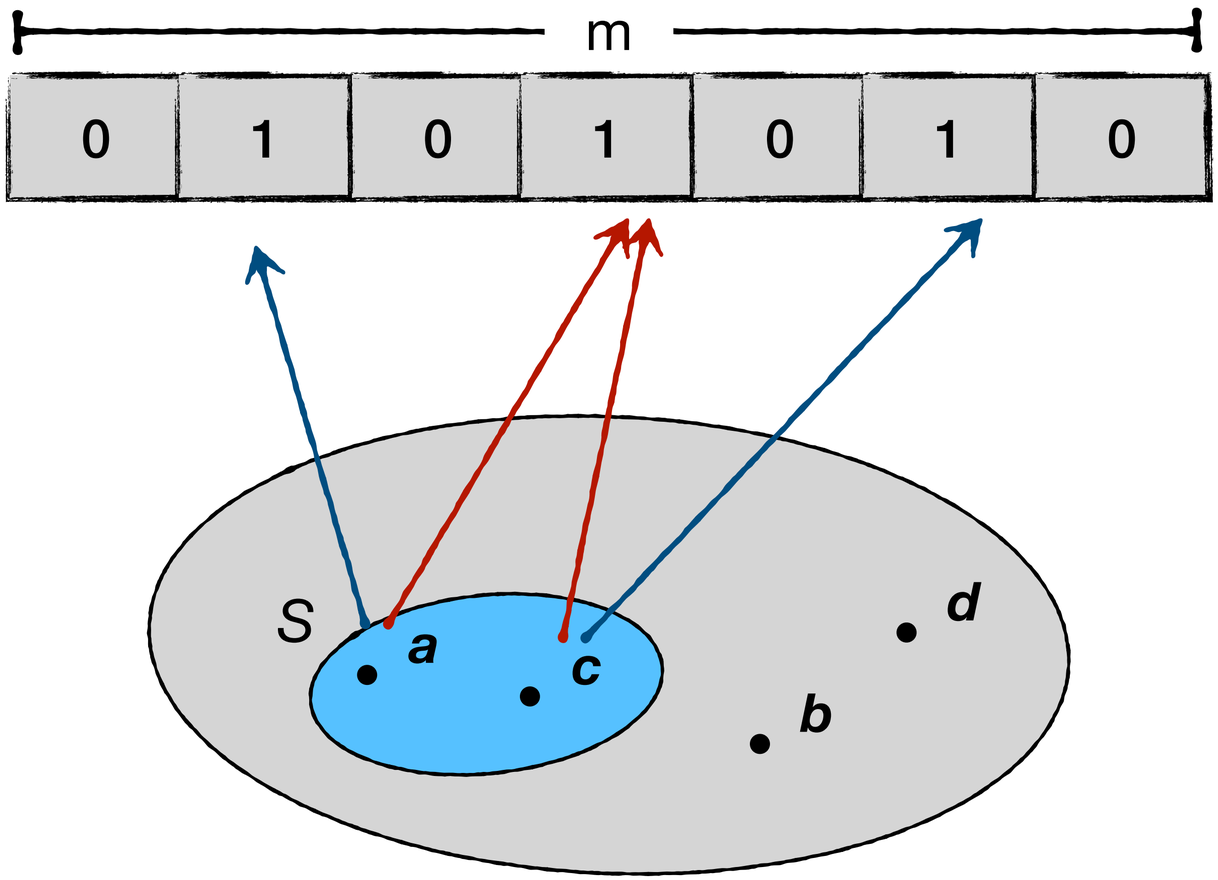
Bloom filter: m bit array + k hash functions (here k=2)

$h_1(a) = 1$

$h_2(a) = 3$

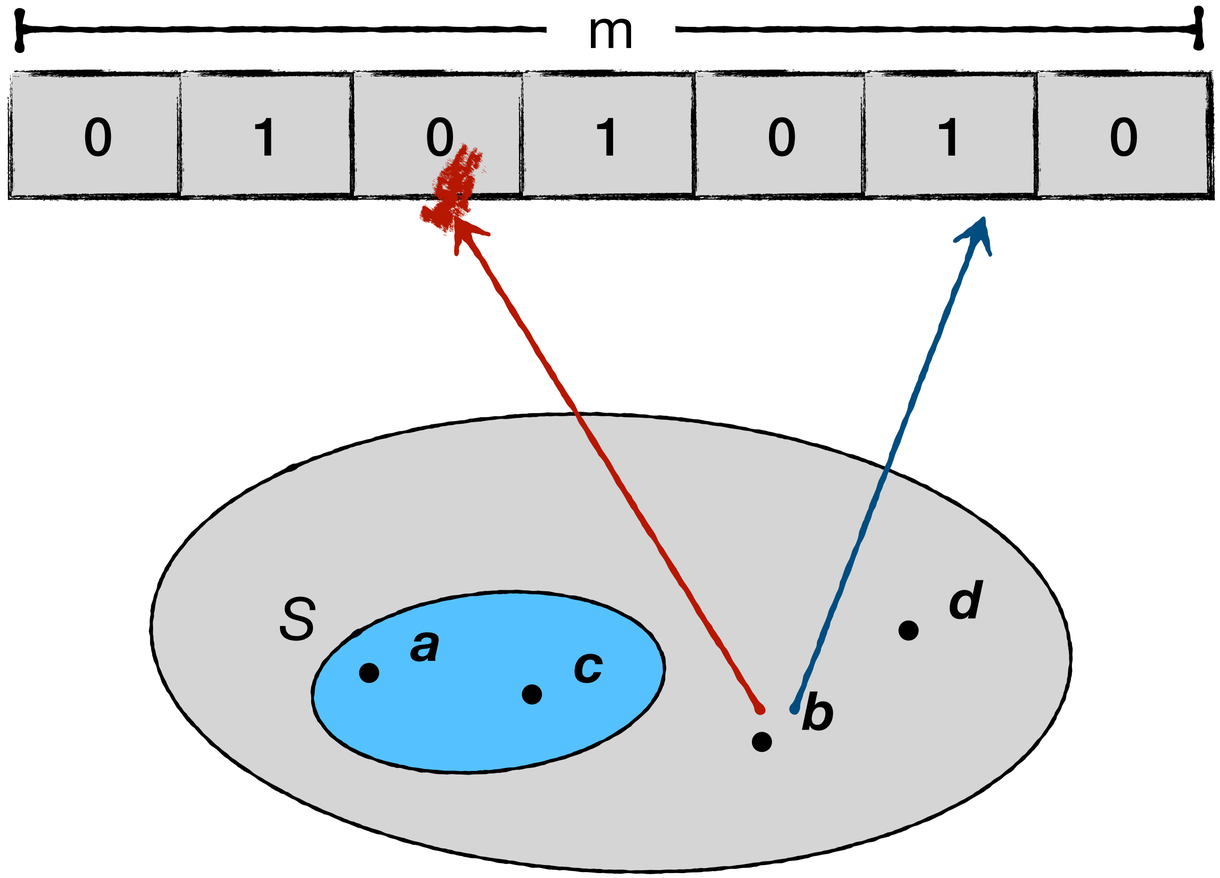
$h_1(c) = 5$

$h_2(c) = 3$



Classic filter: The Bloom filter (BF) [Bloom 70]

Bloom filter: m bit array + k hash functions (here k=2)



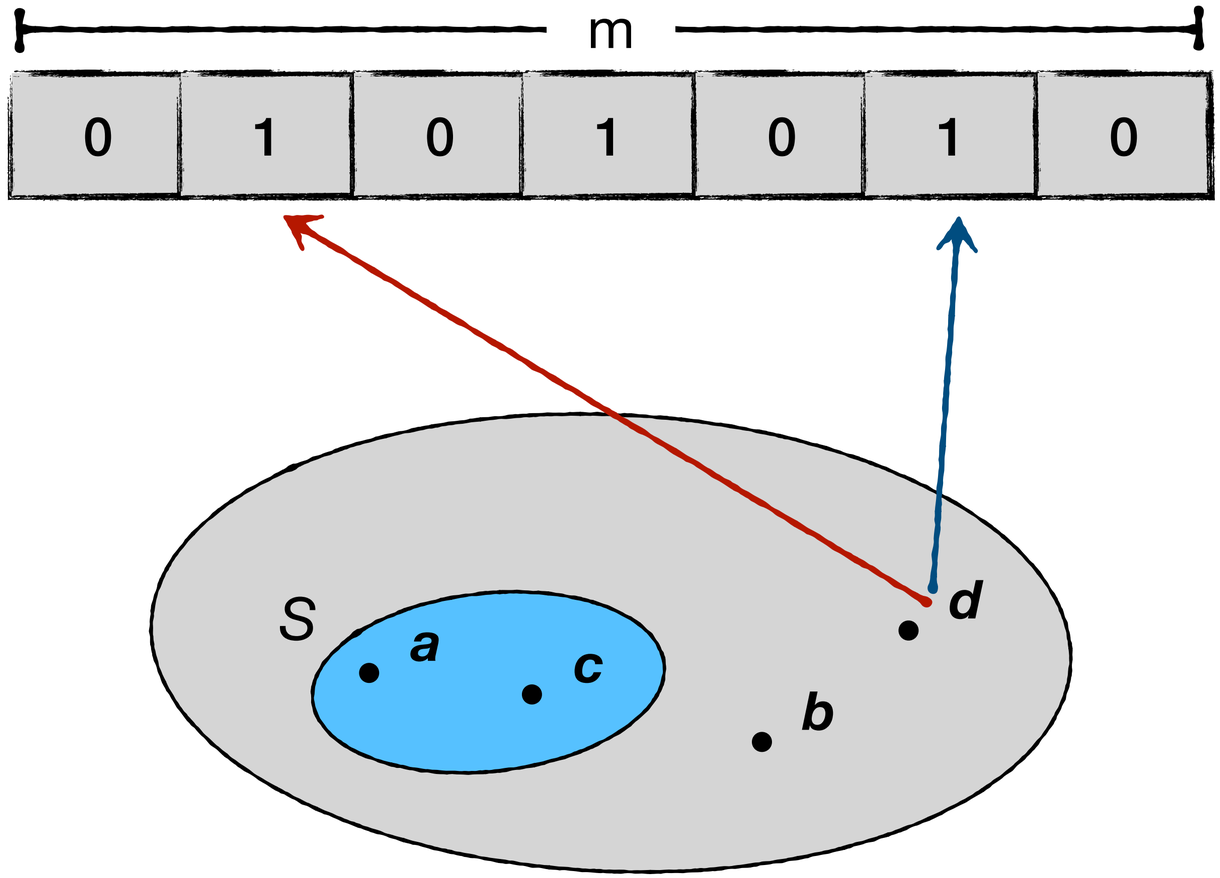
$h_1(b) = 5$

$h_2(b) = 2$

True negative

Classic filter: The Bloom filter (BF) [Bloom 70]

Bloom filter: m bit array + k hash functions (here k=2)

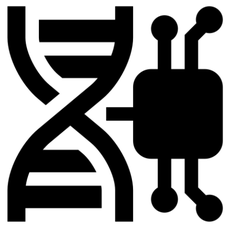


$h_1(d) = 5$

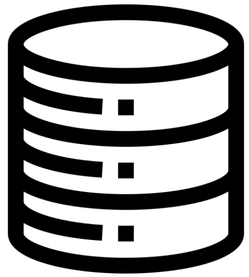
$h_2(d) = 1$

False positive

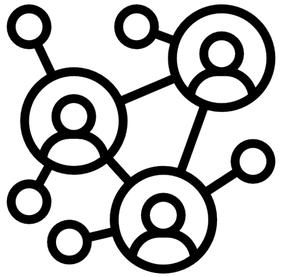
Bloom filters are ubiquitous (> 10K citations)



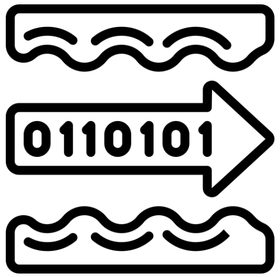
Computational biology



Databases



Networking



Stream processing

Storage systems

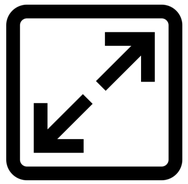
Bloom filters have suboptimal performance

CFGMW 78: Optimal filter bound

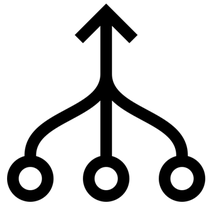
	Bloom filter	**Optimal**
Space (bits)	$\sim 1.44n \log(1/\epsilon)$	$\sim n \log(1/\epsilon) + \Omega(n)$
CPU cost	$\Omega(1/\epsilon)$	$O(1)$
Data locality	$\Omega(1/\epsilon)$ probes	$O(1)$ probes

Bloom filters have several limitations

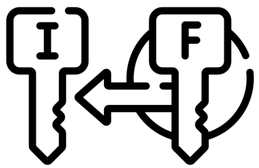
No Deletes



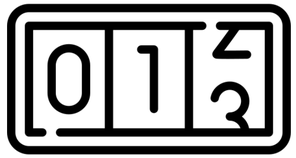
No Resize



No Merging/ Enumeration



No value association



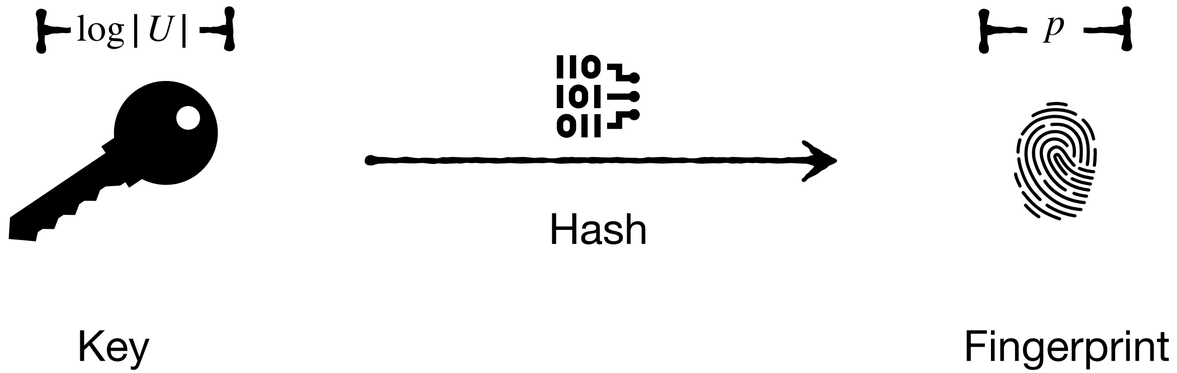
No Counting

Poor cache locality

Bloom filter limitations increase system complexity, waste space, and slow down application performance

Fingerprinting is an alternative to Bloom filters

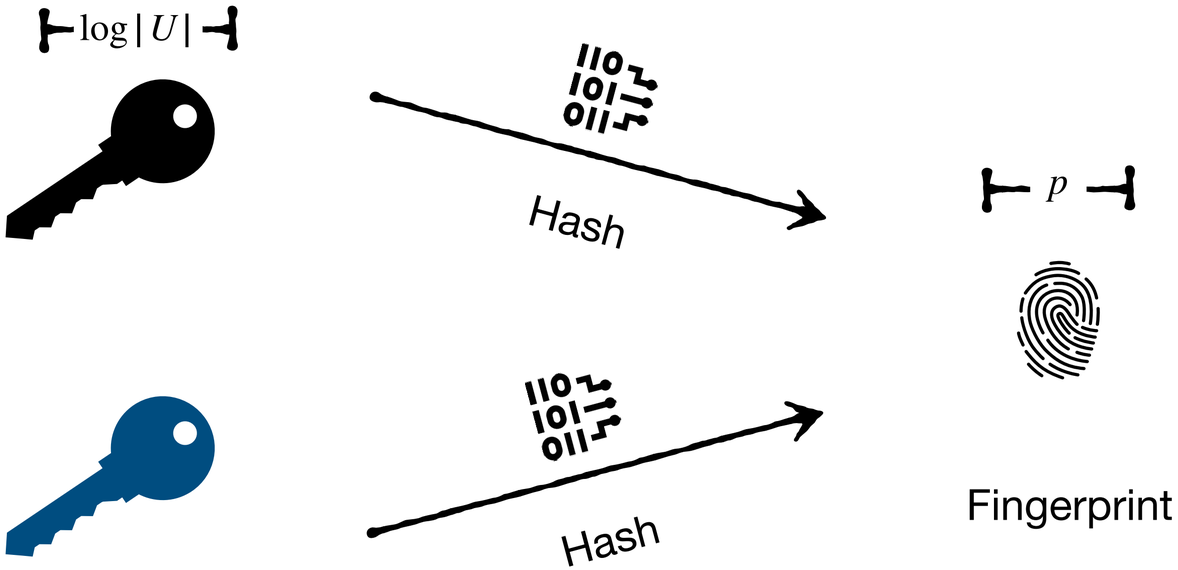
PPR05, DM09, BFJ+12, EF16, PBJ+17



Store fingerprints compactly in a table

Fingerprinting is an alternative to Bloom filters

PPR05, DM09, BFJ+12, EF16, PBJ+17

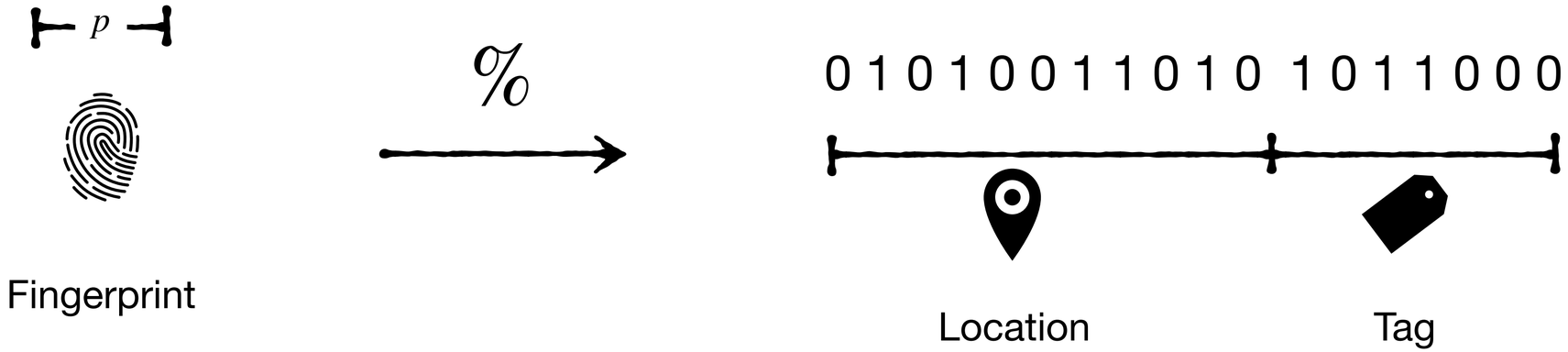


False positives occur only when fingerprints collide

$$\textbf{Pr [collision]} = \frac{1}{2^p}$$

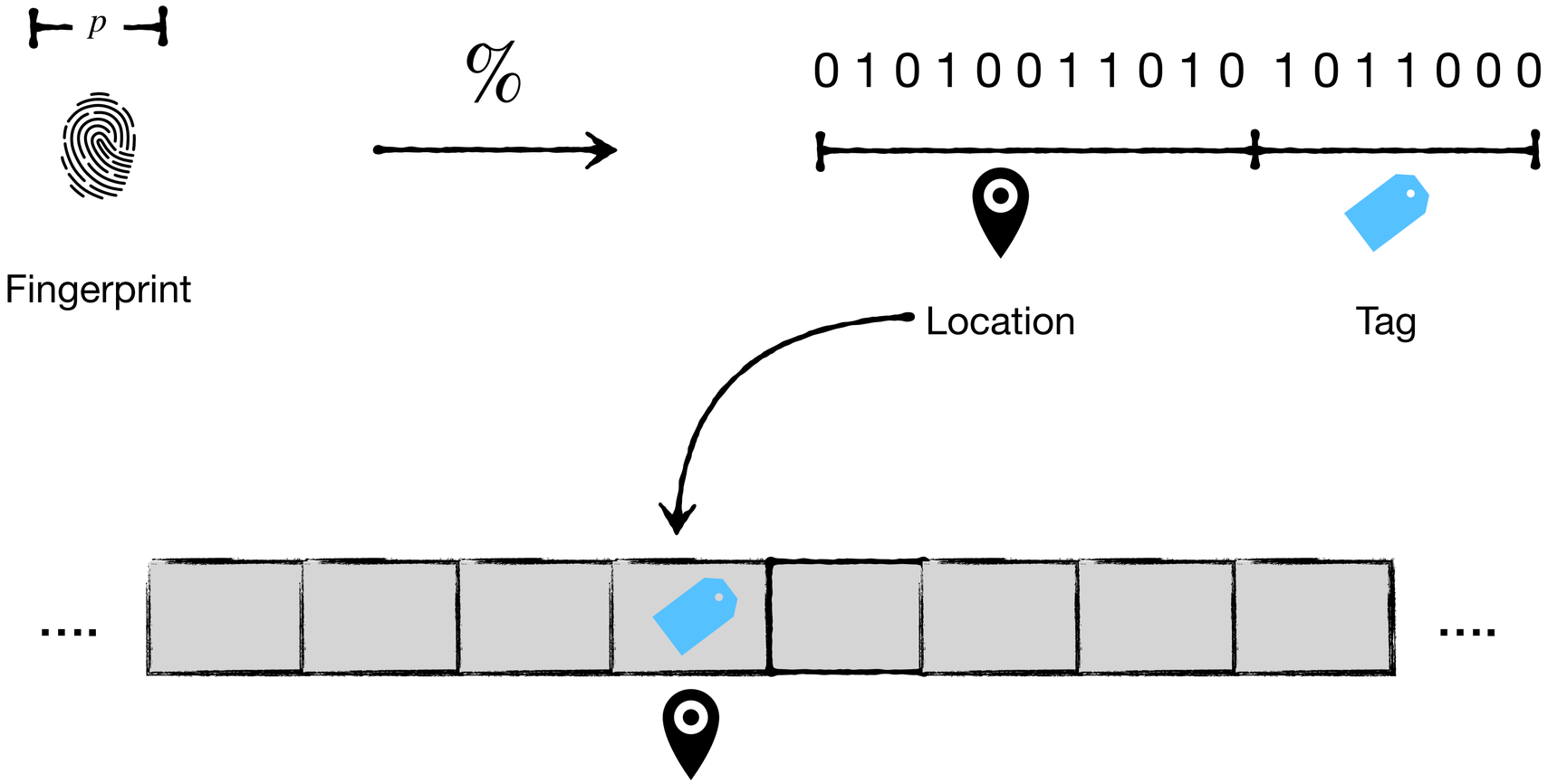
Storing fingerprints compactly using quotienting

Knuth 97



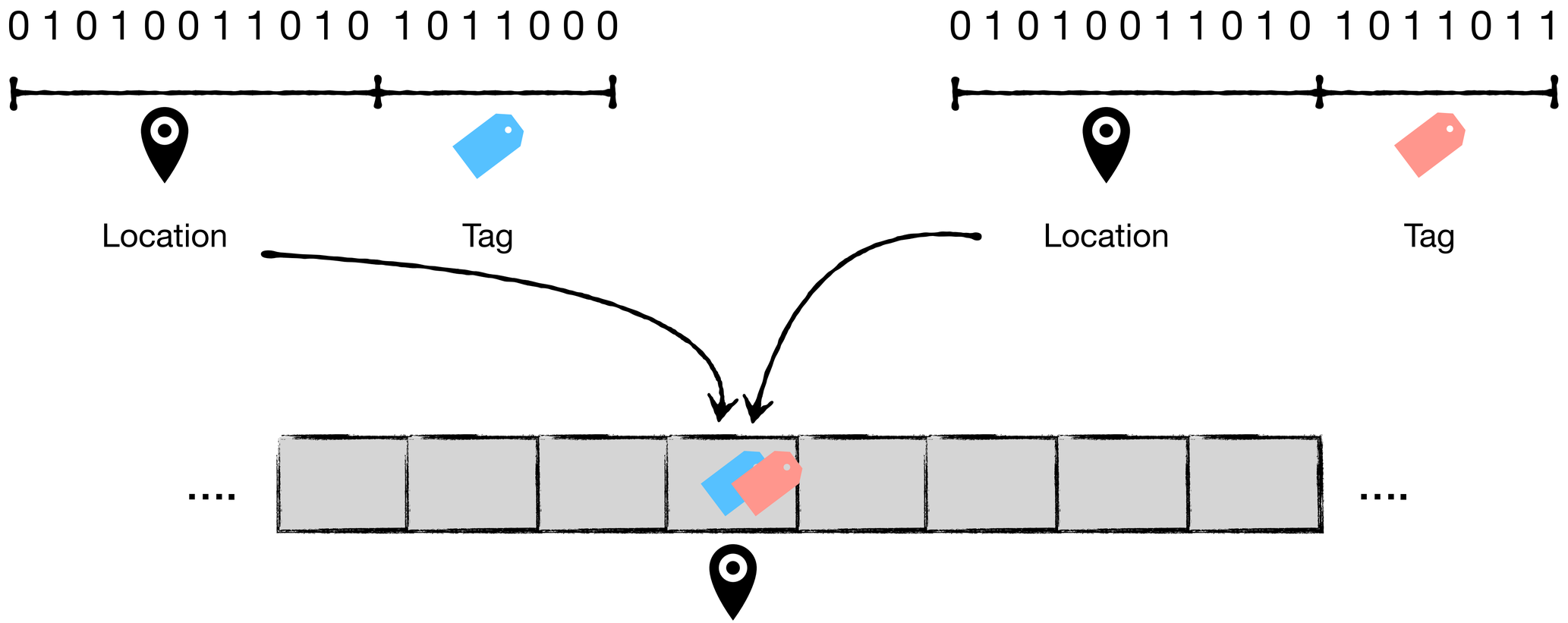
Storing fingerprints compactly using quotienting

Knuth 97



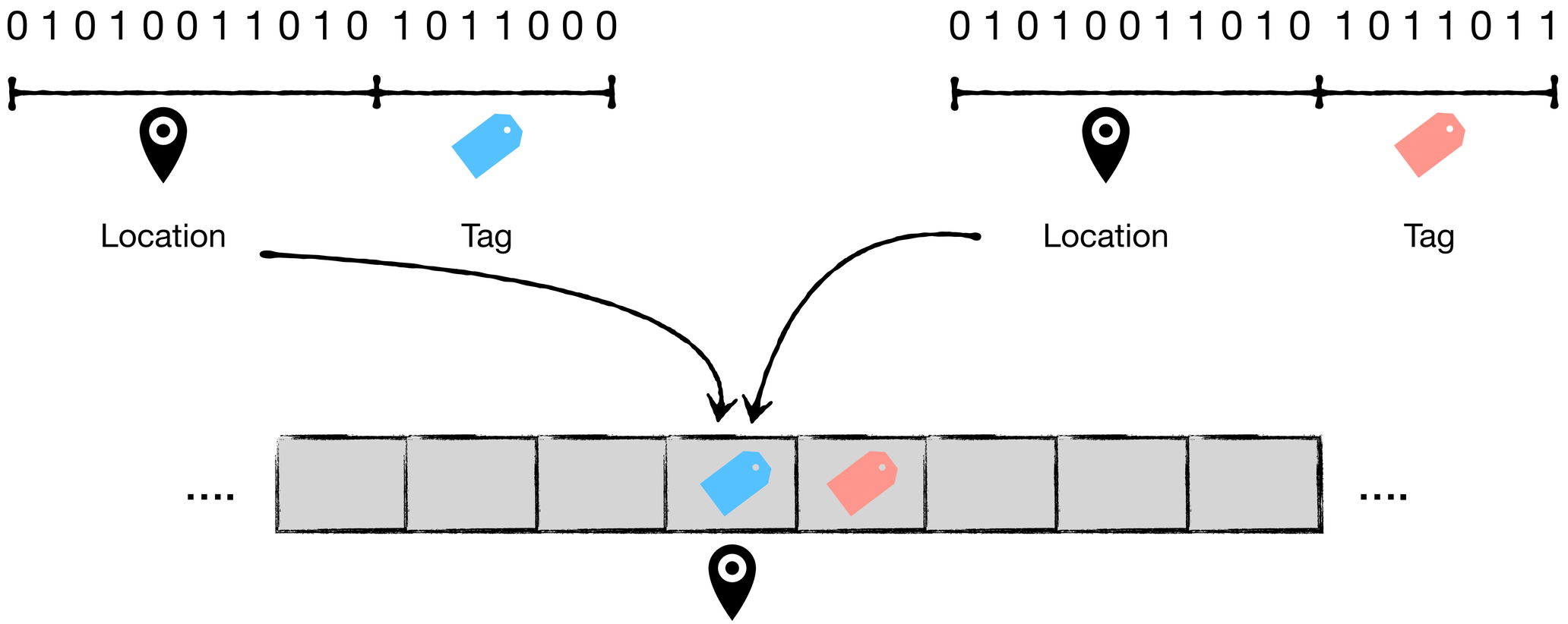
Storing fingerprints compactly using quotienting

Knuth 97



Storing fingerprints compactly using quotienting

Knuth 97



Use linear probing and Robinhood hashing

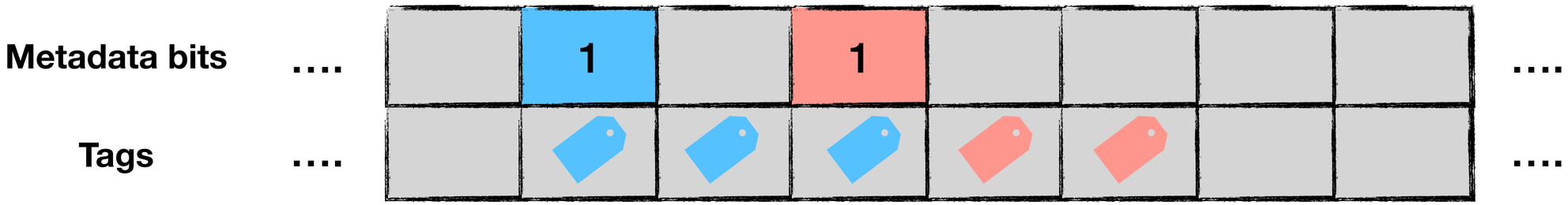
Resolving collisions in quotient filter (QF)

Pandey et al. SIGMOD 17

How to identify the home slot of a given tag?

Resolving collisions in quotient filter (QF)

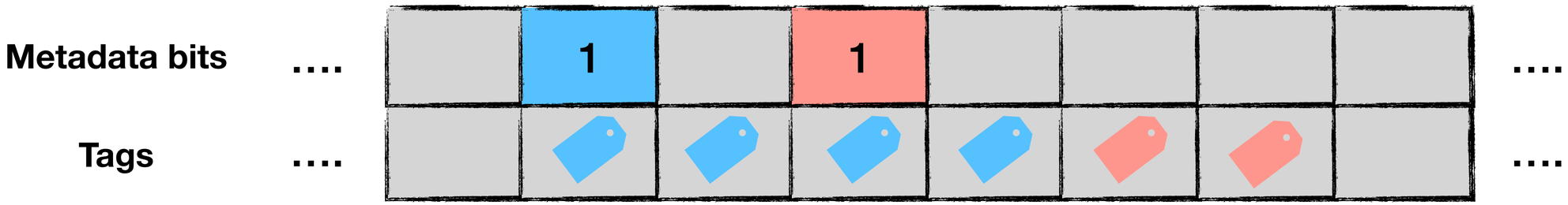
Pandey et al. SIGMOD 17



Use two metadata bits/slot to group tags by their home slot

Resolving collisions in quotient filter (QF)

Pandey et al. SIGMOD 17



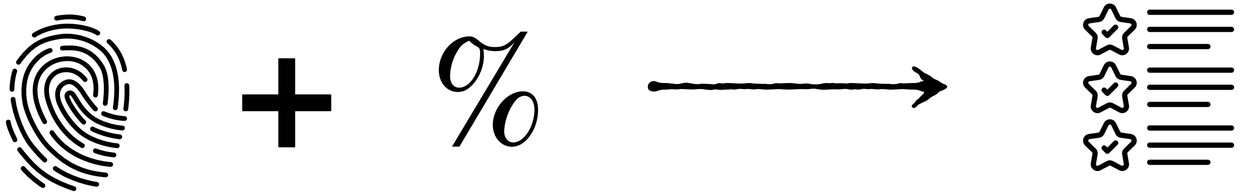
Metadata bits help identify the home slot of each tag

Quotient filters offer better performance than BF

CFGMW 78: Optimal filter bound

	Quotient filter	Bloom filter	Optimal
Space (bits)	$\sim n \log(1/\epsilon) + 2.125n$	$\sim 1.44n \log(1/\epsilon)$	$\sim n \log(1/\epsilon) + \Omega(n)$
CPU cost	$O(1)$ expected	$\Omega(1/\epsilon)$	$O(1)$
Data locality	1 probe + scan	$\Omega(1/\epsilon)$ probes	$O(1)$ probes

Quotient filters have theoretical advantages over Bloom filters



Fingerprinting Quotienting Features

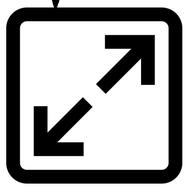
+ %

Fingerprinting Quotienting Features

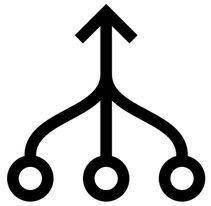
MetaHipMer
PPoPP 23
ACDA 23

Deletes

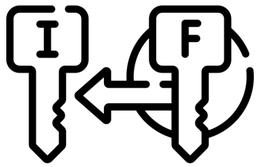
Squeakr
BIOINFORMATICS 17



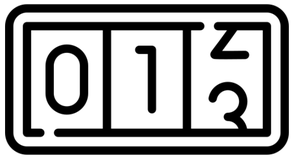
Resize



Merging/ Enumeration



Value association



Counting (Variable length)

High performance & scalability

Cache locality

LERTs
SIGMOD 20, TODS 20

Mantis
Cell systems 18
RECOMB 18

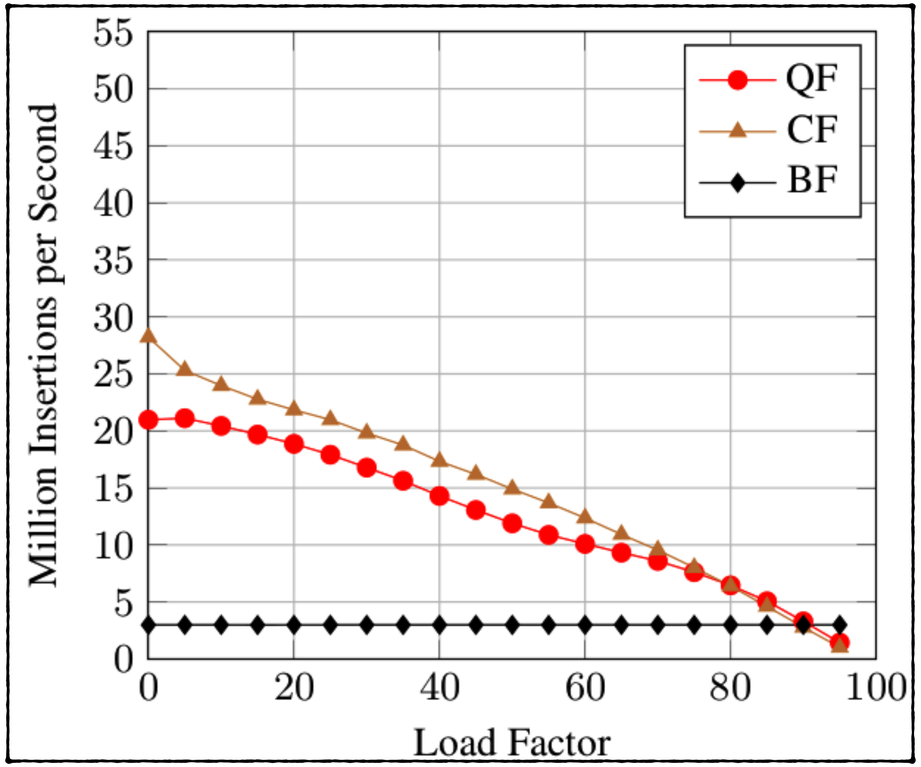
Asymptotically optimal space

Squeakr, deBGR
BIOINFORMATICS 17
ISMB 17

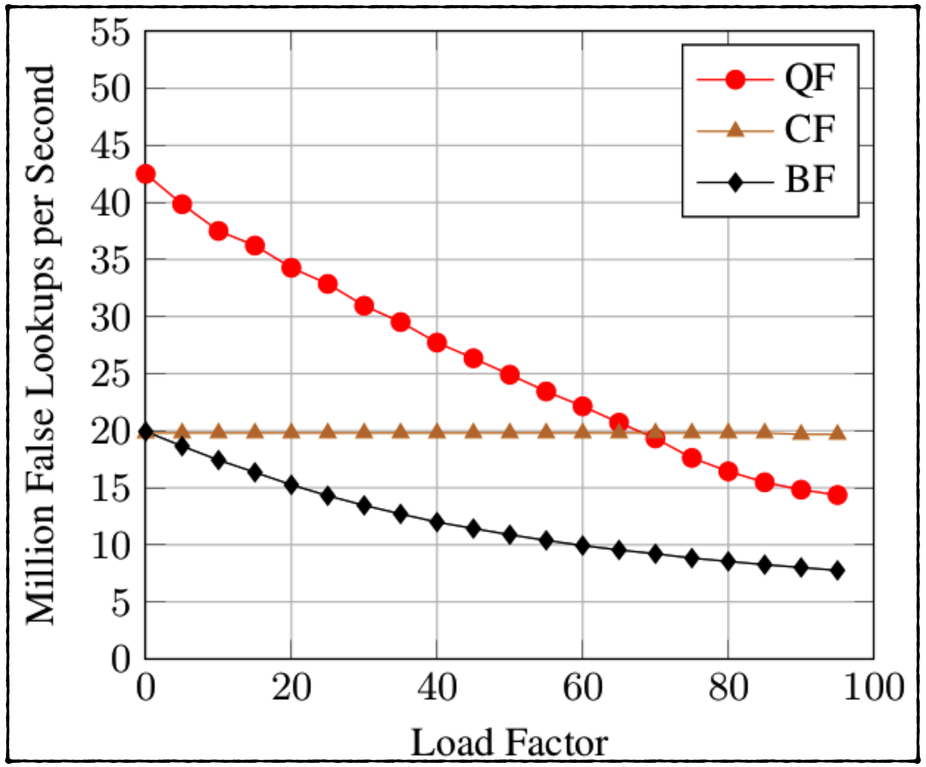
Quotient filters empirical performance

QF: quotient filter
CF*: cuckoo filter [FAK+14]
BF*: Bloom filter

Inserts



Queries

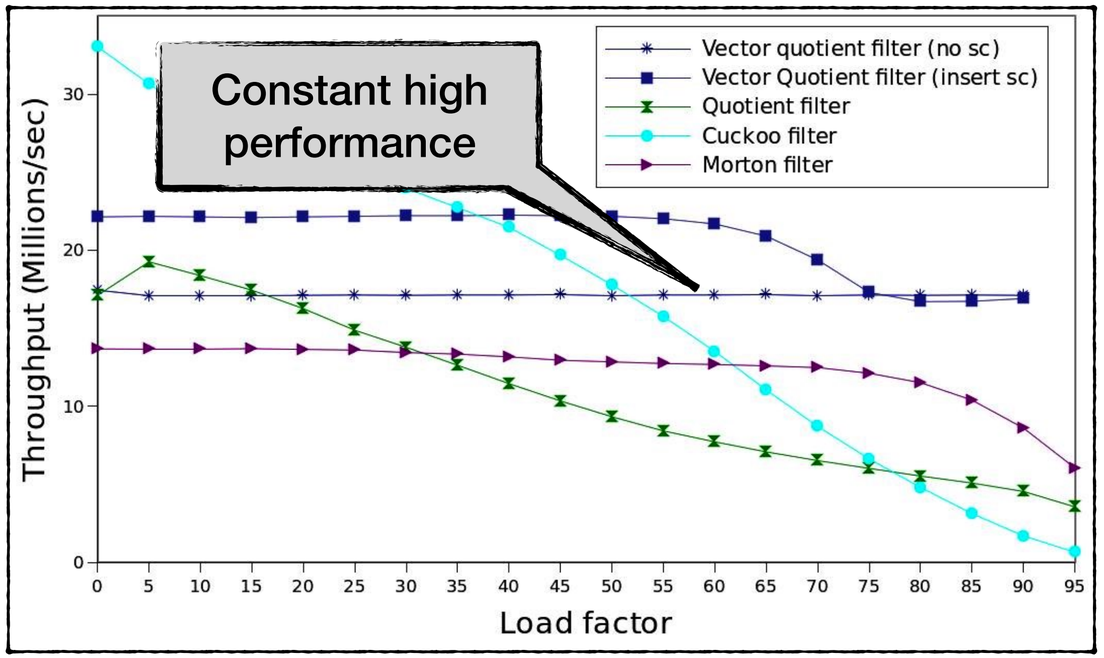


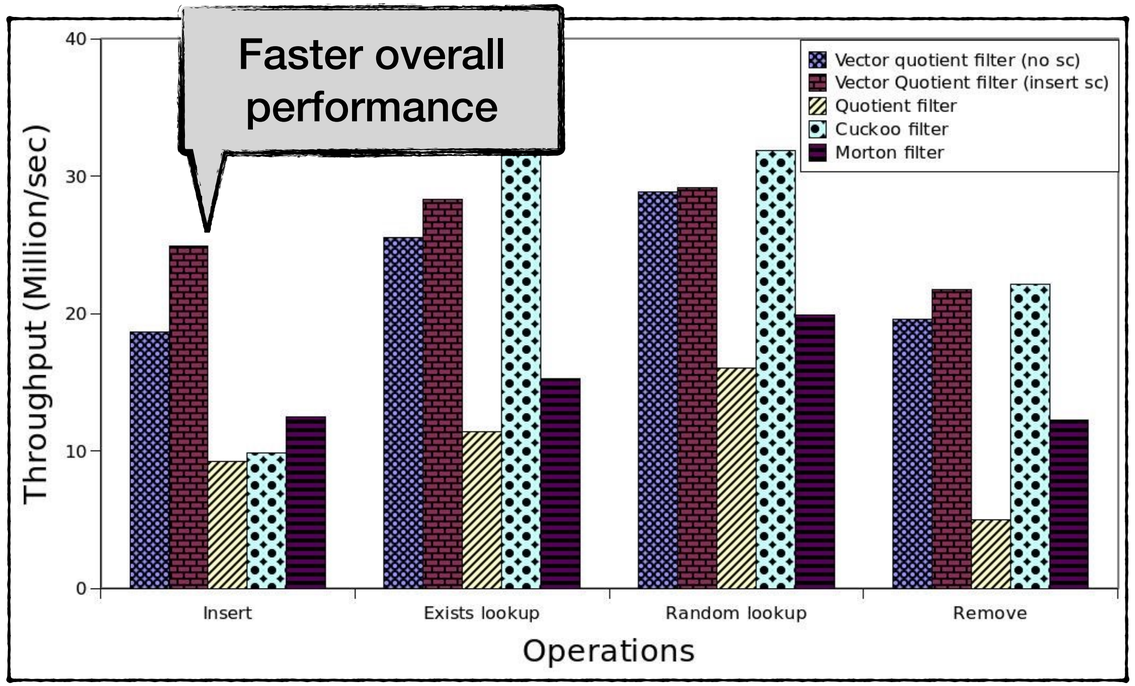
Insert performance is similar to the state-of-the-art non-counting filters

Query performance is significantly fast at low load factors and slightly slower at high load factors

Vector quotient filters [SIGMOD 21]

Pandey et al. SIGMOD 21





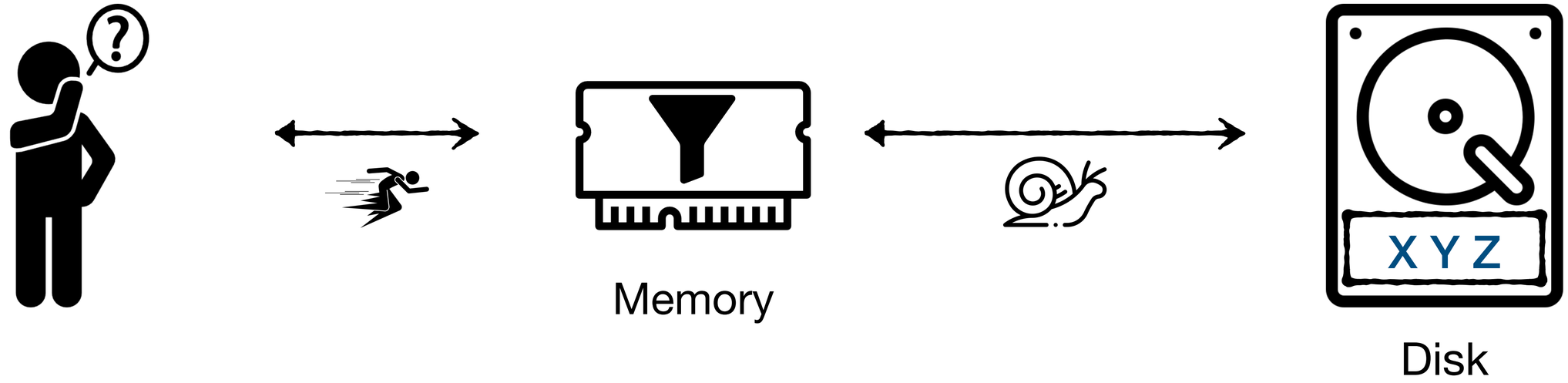
Combining hashing techniques (Robinhood hashing + power of 2-choice hashing)

Using ultra-wide vector instructions (AVX-512)

Filters offer weak guarantees

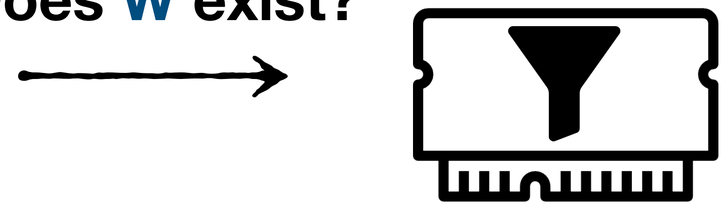
The maximum false positive rate is only guaranteed for a single query and not an arbitrary sequence of queries

Skewed workloads can make filters obsolete

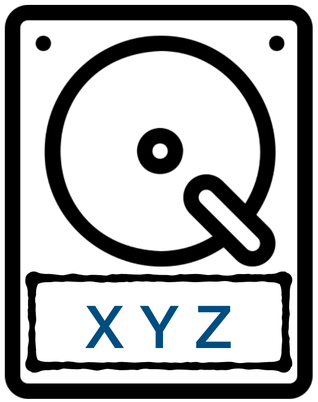


Skewed workloads can make filters obsolete

Does W exist?

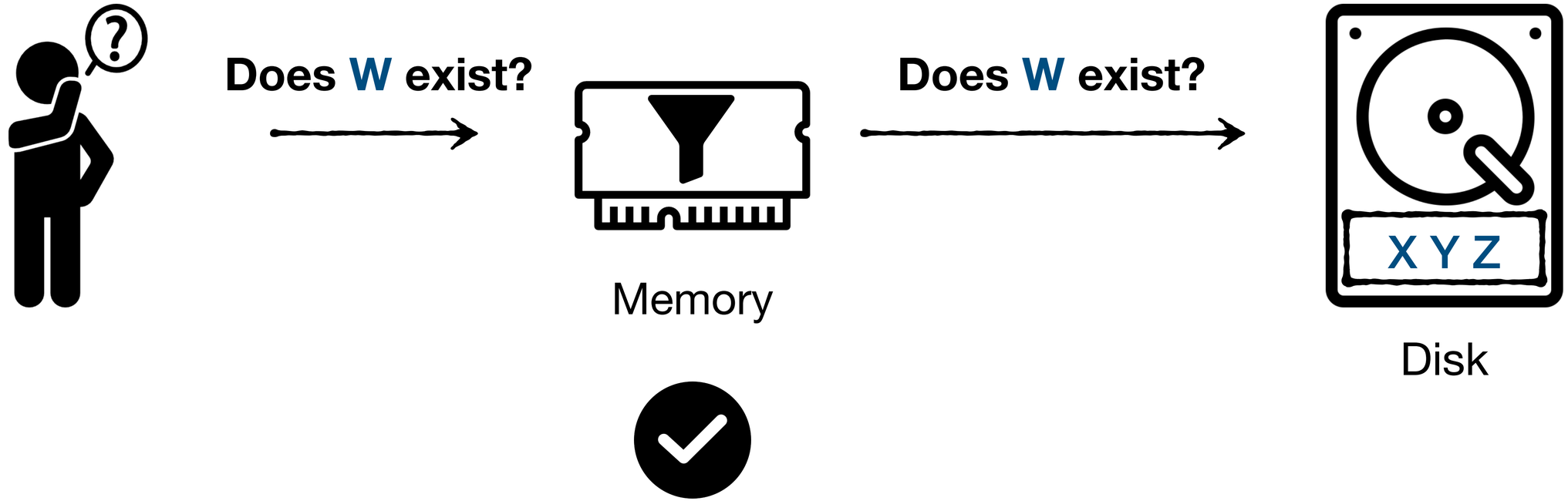


Memory

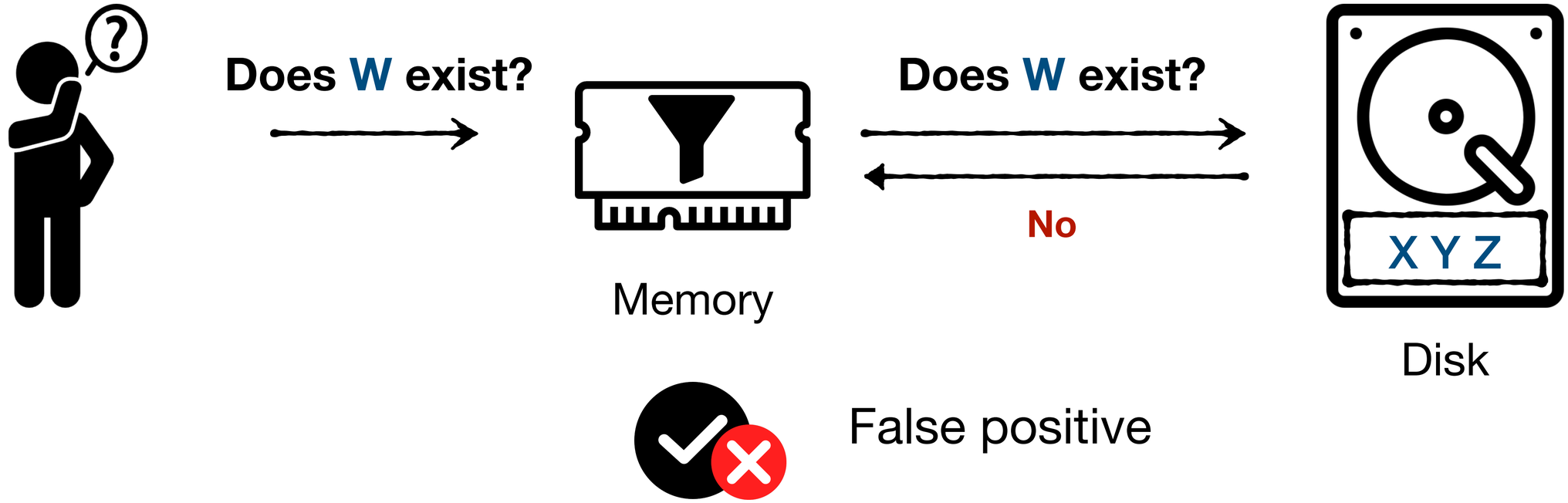


Disk

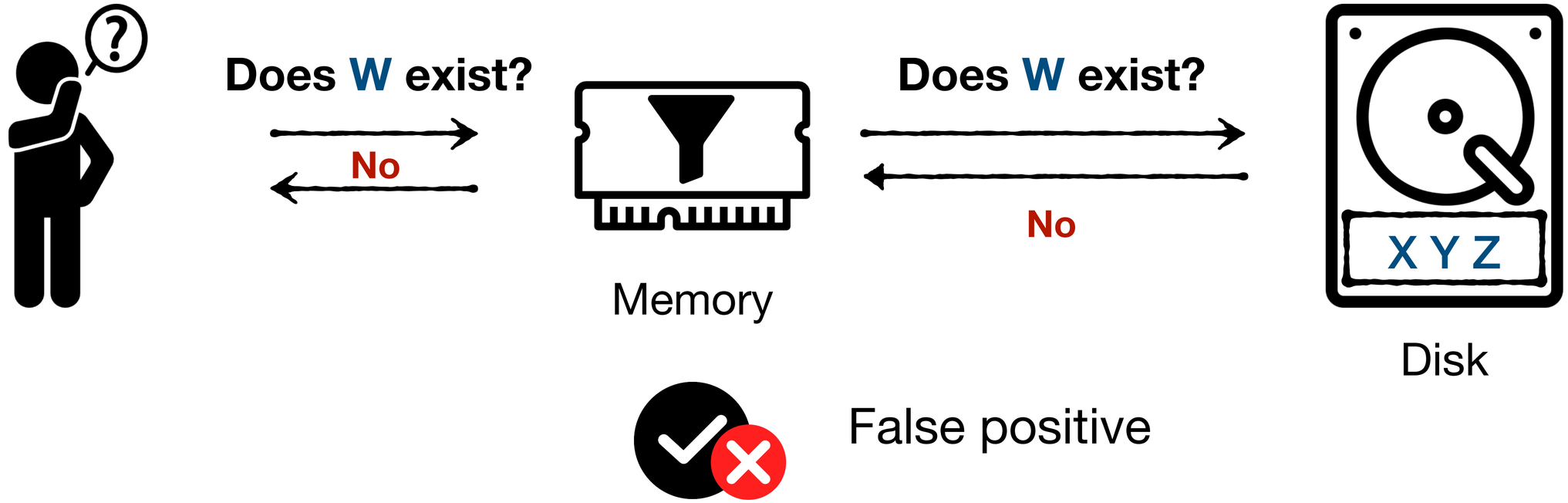
Skewed workloads can make filters obsolete



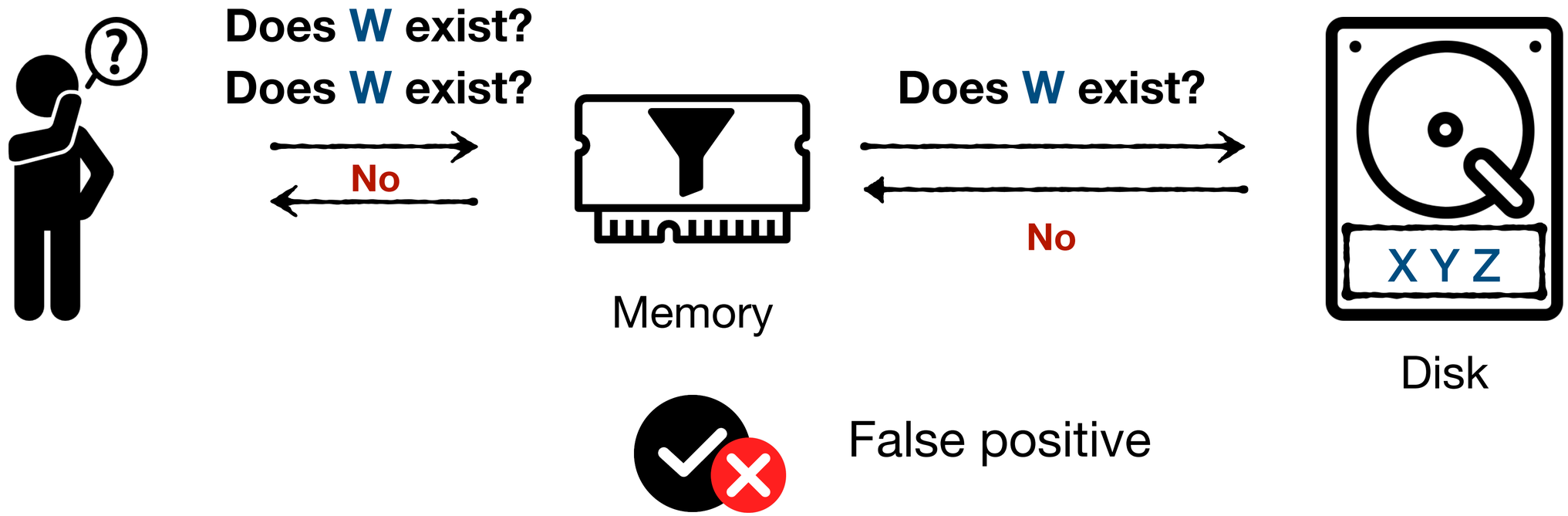
Skewed workloads can make filters obsolete



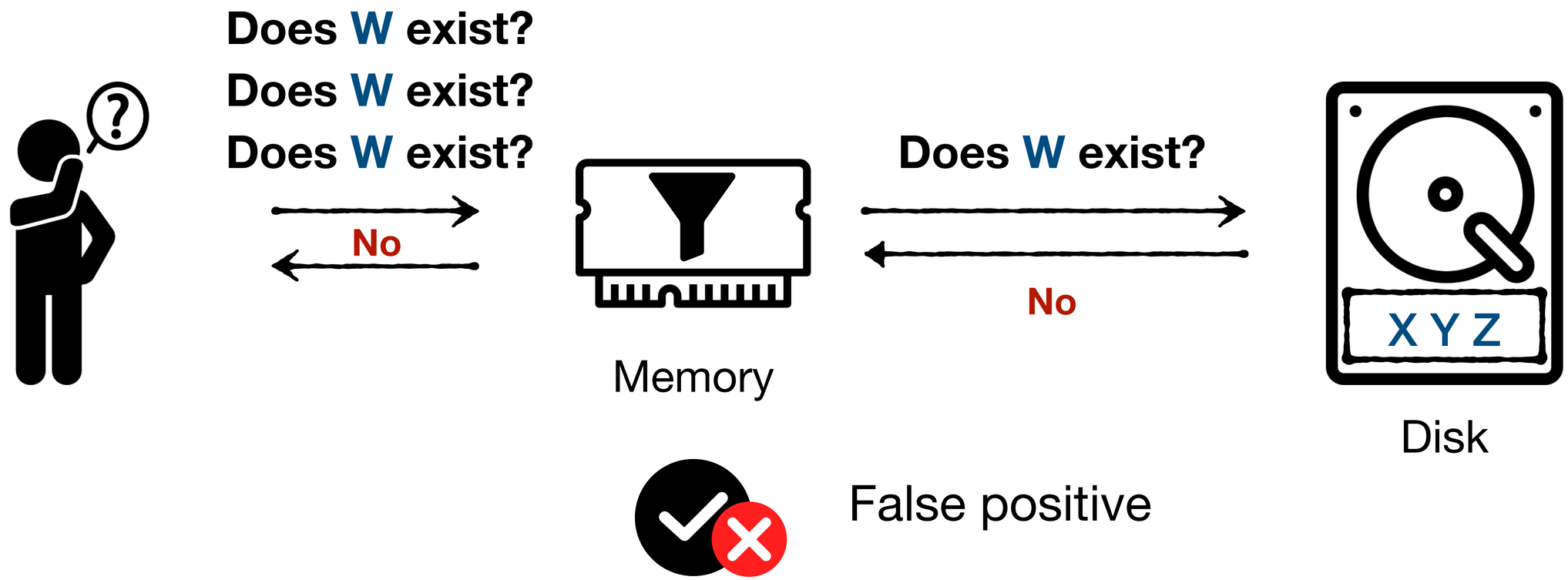
Skewed workloads can make filters obsolete



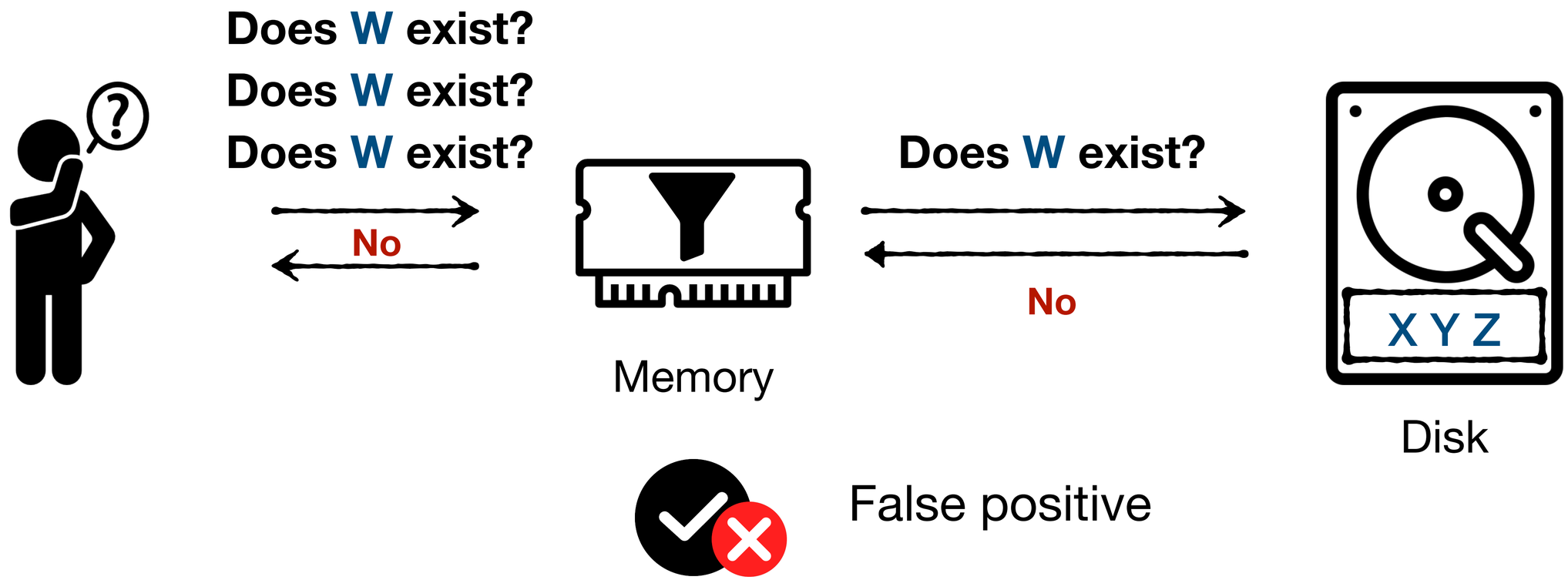
Skewed workloads can make filters obsolete



Skewed workloads can make filters obsolete



Skewed workloads can make filters obsolete



False-positive rate $\leq \epsilon$, only for a single query

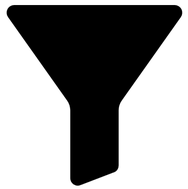
False positives can be really expensive

Malicious URLs

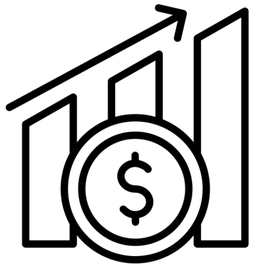
Legitimate URLs

q ∈ Legitimate

YES



Filter containing malicious URLS



Expensive

A false positive can block critical URLs such as a voter registration webpage or emergency weather info

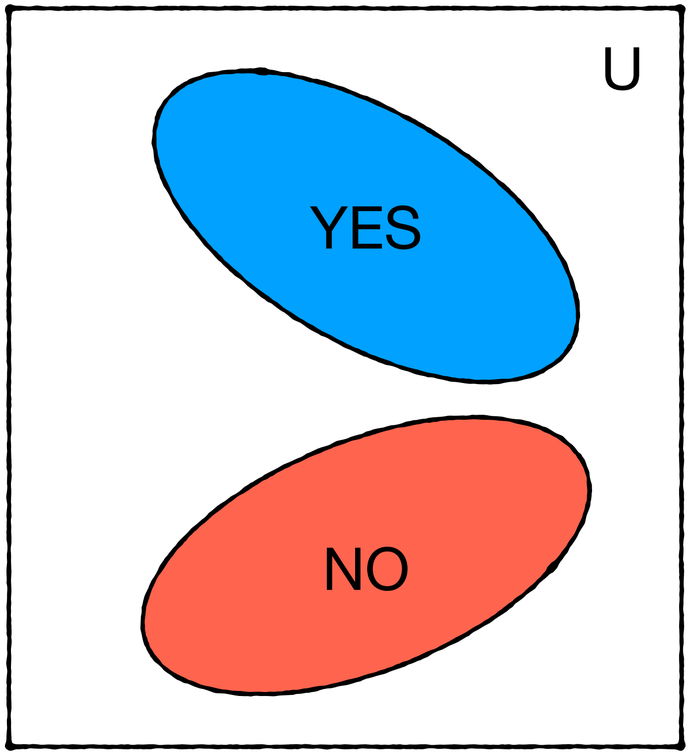
False positive

YES/NO list problem

if q ∈ YES, return **True** with probability 1

if q ∈ NO, return **False** with probability 1

Otherwise **False** with probability $> 1 - \epsilon$



Applications:

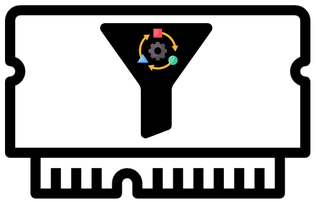
- Detecting malicious URL
- Certificate revocation lists
- De Bruijn graph traversal

Monotonicity is critical to support YES/NO List problem!

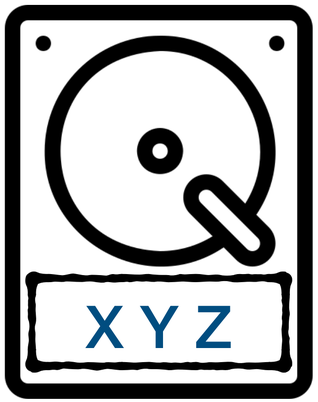
Can we learn from the feedback?

Adaptive filters change their state upon feedback

Does W exist?

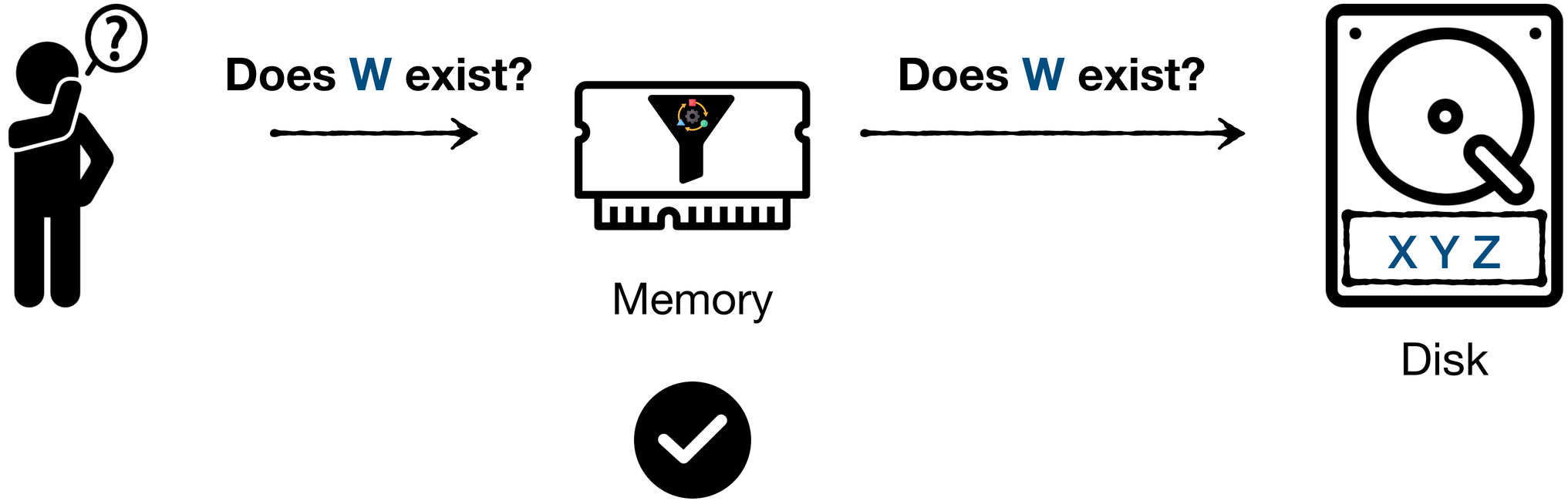


Memory

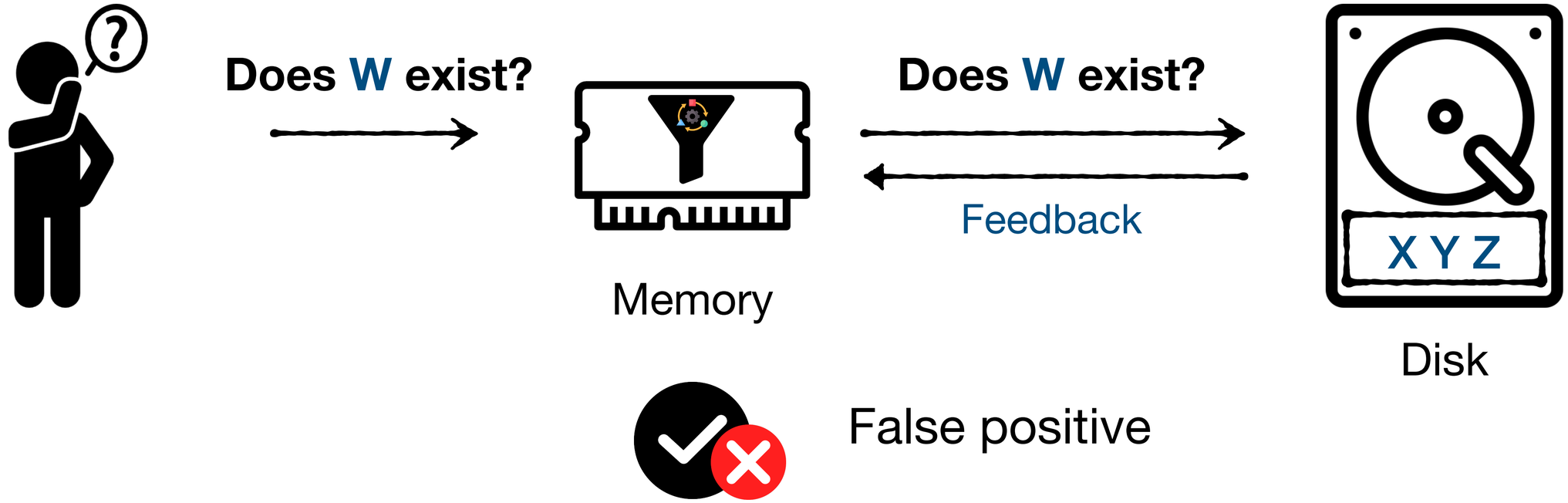


Disk

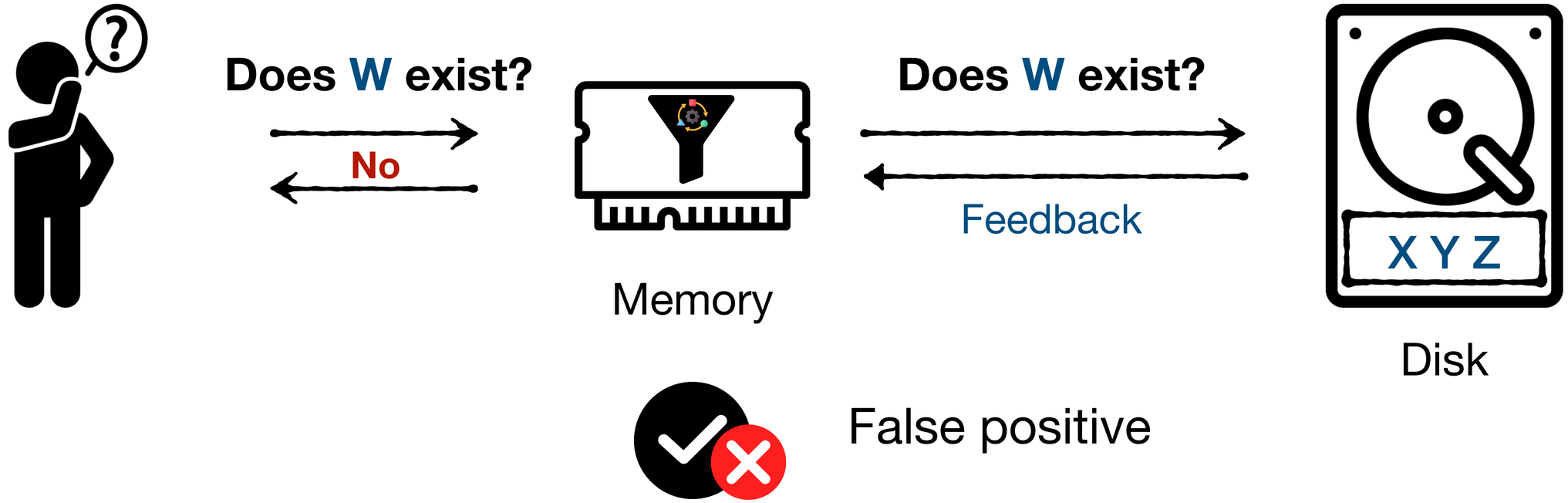
Adaptive filters change their state upon feedback



Adaptive filters change their state upon feedback



Adaptive filters change their state upon feedback

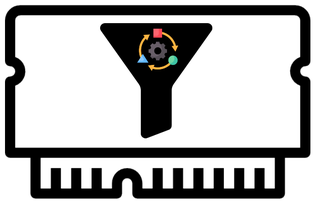


Adaptive filters change their state upon feedback

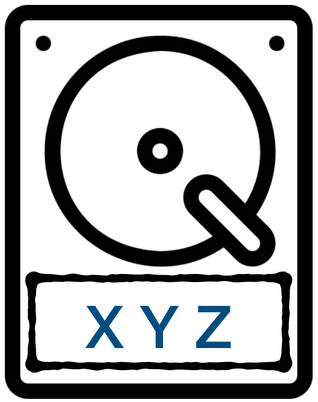
Does W exist?

Does W exist?

No



Memory

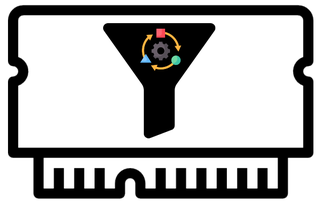


Disk

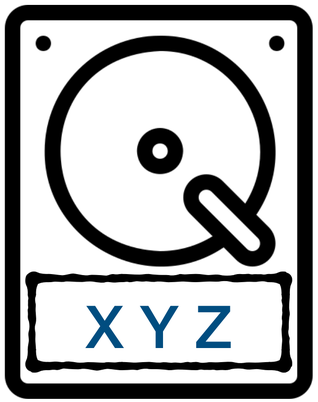
True negative

Adaptive filters change their state upon feedback

Does W exist?
Does W exist?
Does W exist?



Memory



Disk

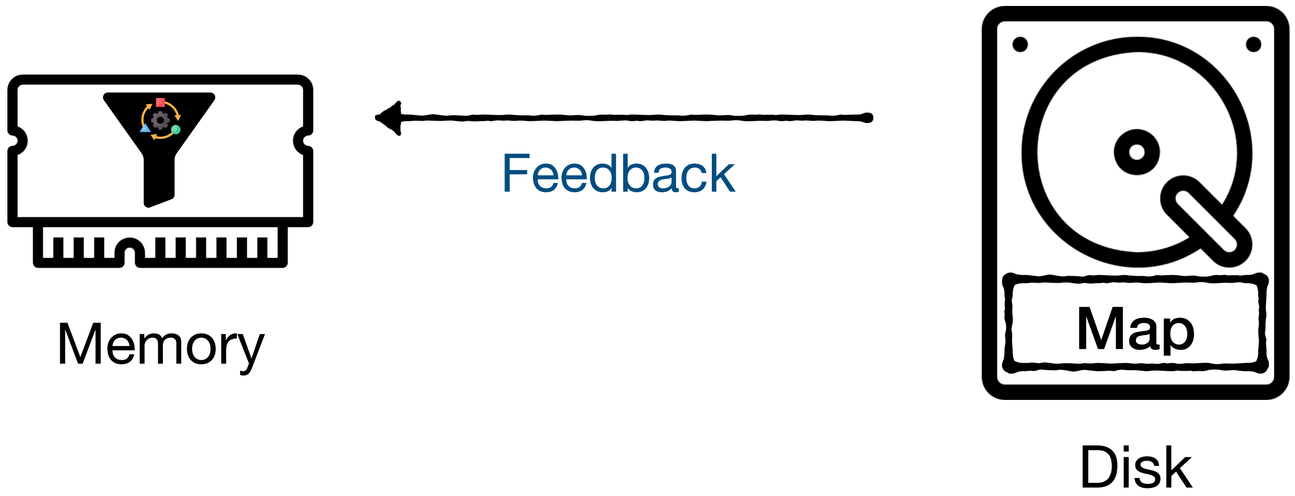
True negative

Adaptive filters [BFG+ 2018]

An adaptive filter modifies its state upon feedback and produces close to $O(\epsilon n)$ false positives for any sequence of n queries

False-positive rate $\leq \epsilon$, independent of the query distribution

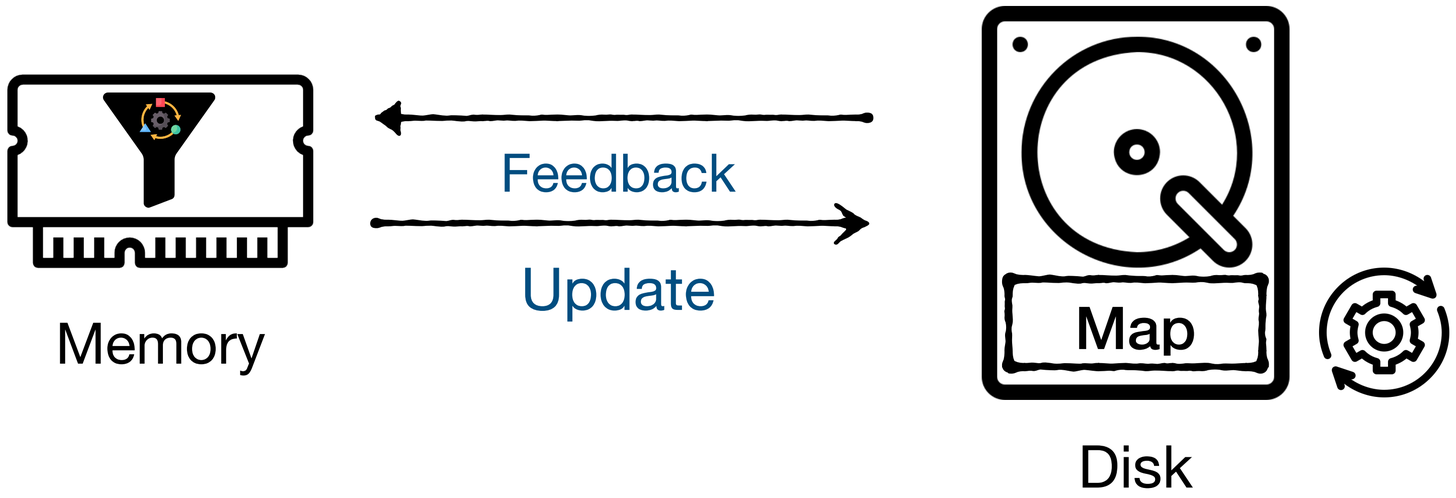
Adaptive filter design has two parts [BFG+ 2018]



Small in-memory filter
accessed on every query

Large disk-resident map
accessed during adaptations

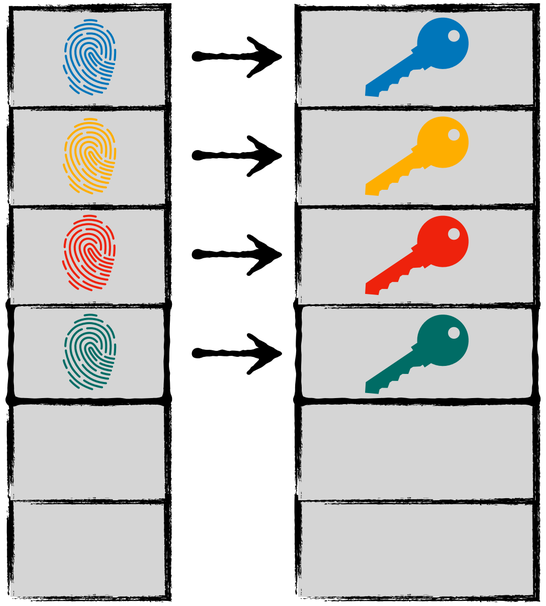
Adaptive filter design has two parts [BFG+ 2018]



On-disk map enables adaptations and is updated to fix fingerprint collisions

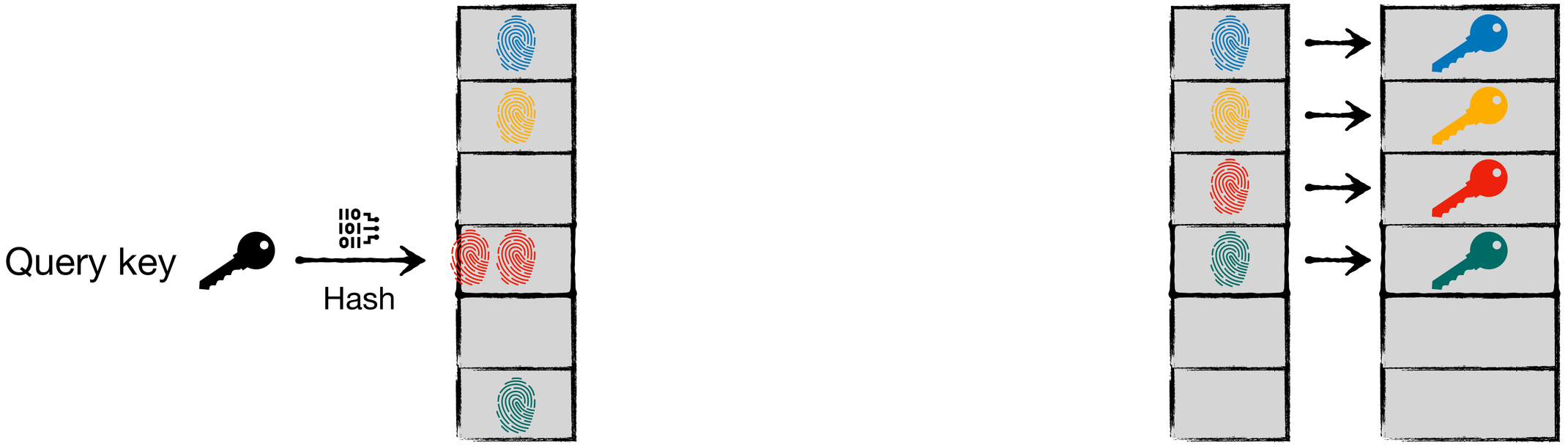
Adaptive filters employ variable-length fingerprints

Adaptive filter
Memory



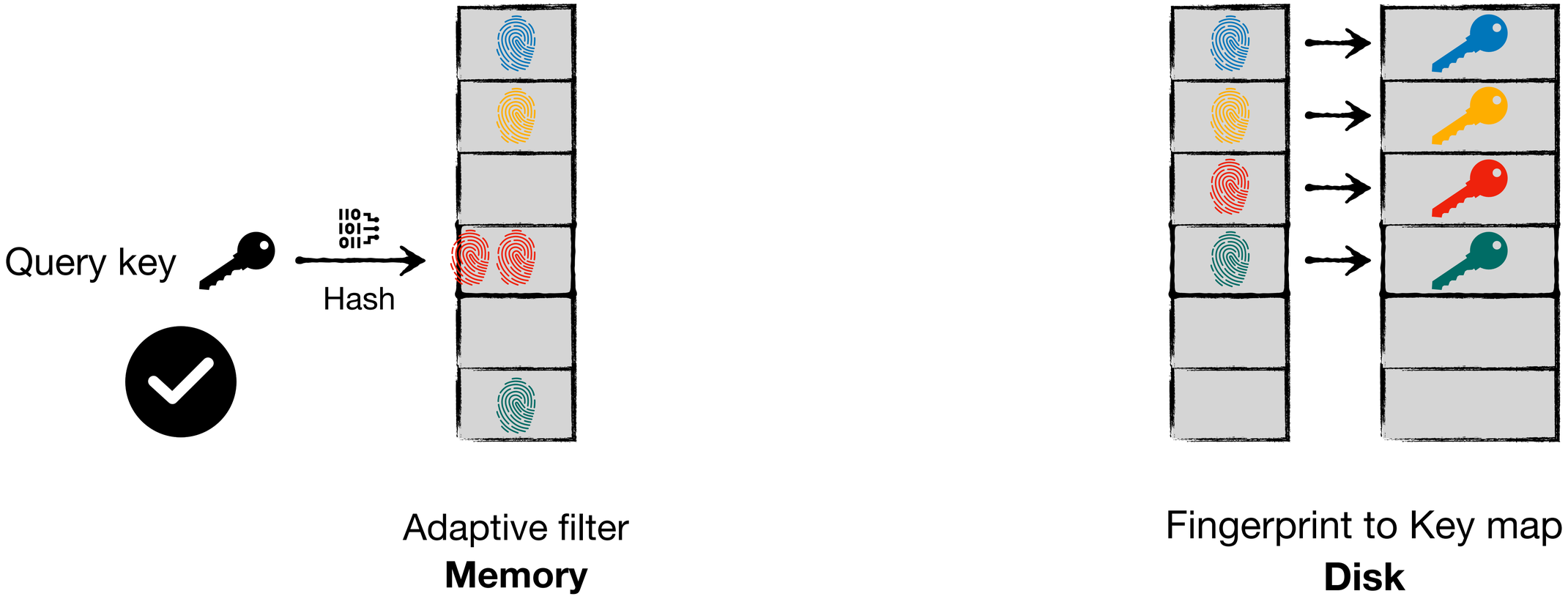
Fingerprint to Key map
Disk

Adaptive filters employ variable-length fingerprints



Fingerprint collisions can cause false positives

Adaptive filters employ variable-length fingerprints

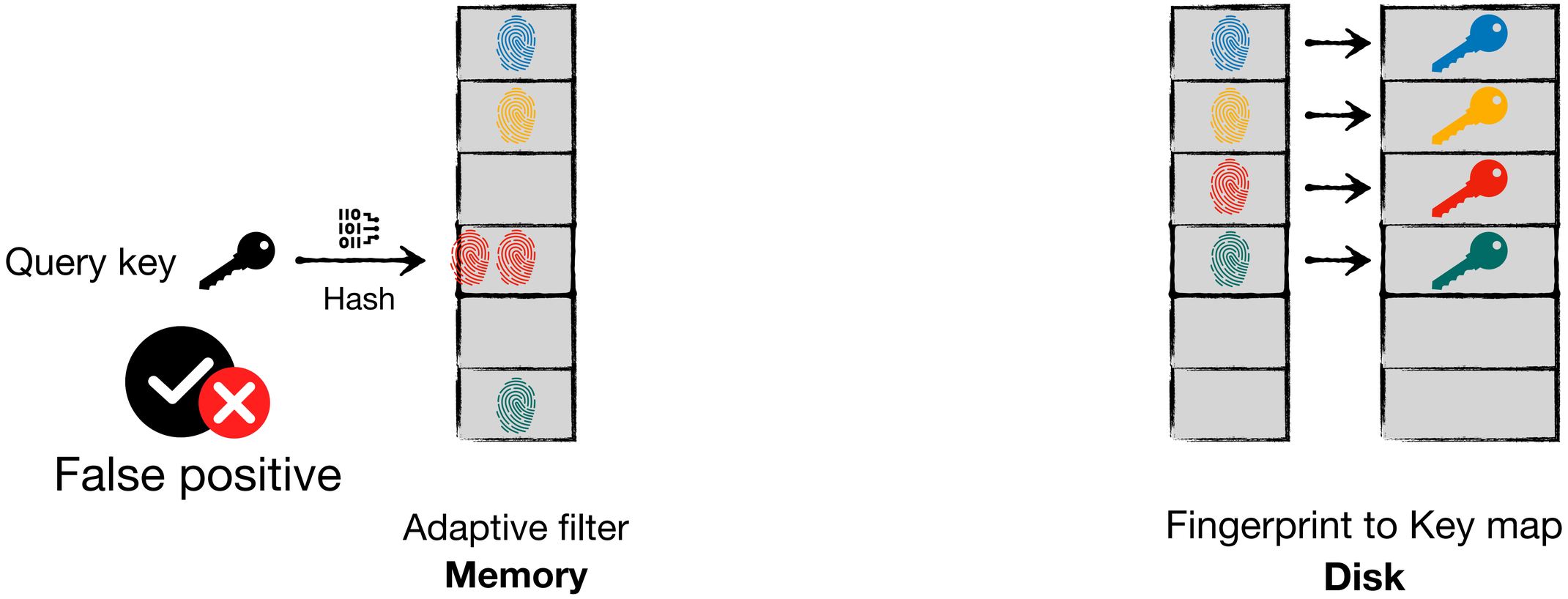


Fingerprint collisions can cause false positives

Adaptive filters employ variable-length fingerprints

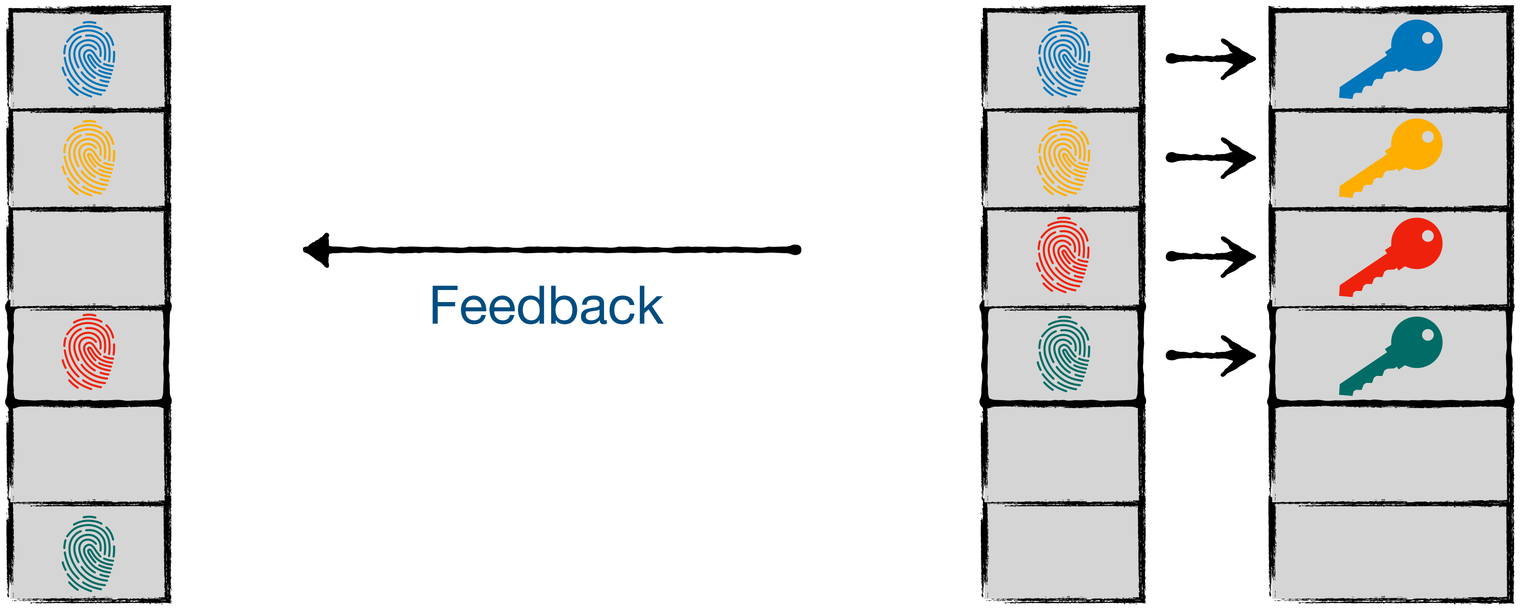
Fingerprint collisions can cause false positives

Adaptive filters employ variable-length fingerprints



Fingerprint collisions can cause false positives

Adaptive filters employ variable-length fingerprints

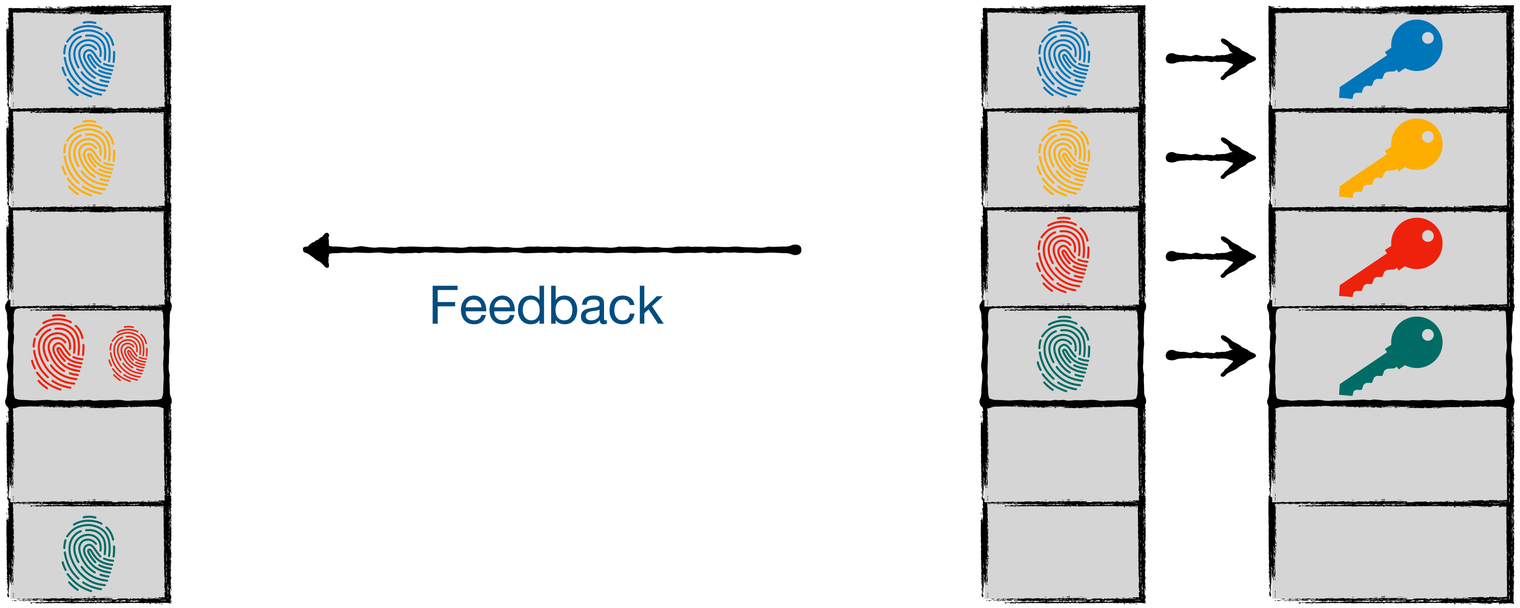


Adaptive filter
Memory

Fingerprint to Key map
Disk

Feedback from the map can help fix the false positive

Adaptive filters employ variable-length fingerprints

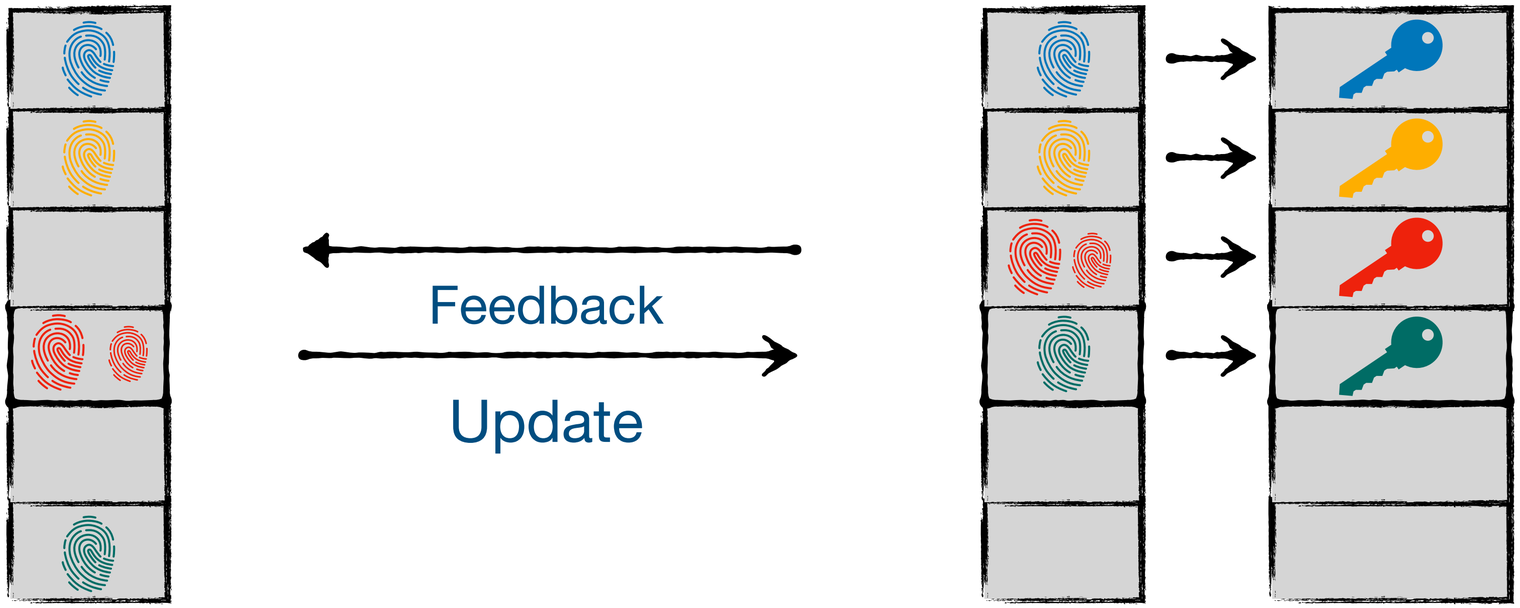


Adaptive filter
Memory

Fingerprint to Key map
Disk

Extending the fingerprint of the existing key can avoid future false positives

Adaptive filters employ variable-length fingerprints

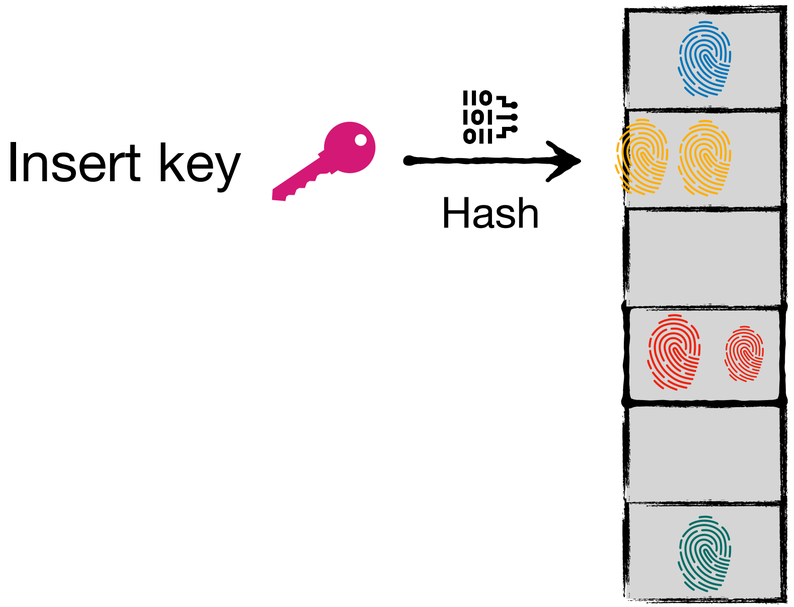


Adaptive filter
Memory

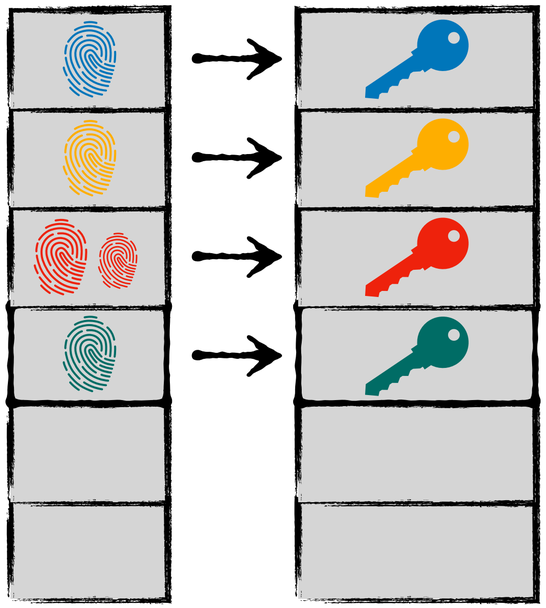
Fingerprint to Key map
Disk

Fingerprint map is updated accordingly

Adaptive filters employ variable-length fingerprints

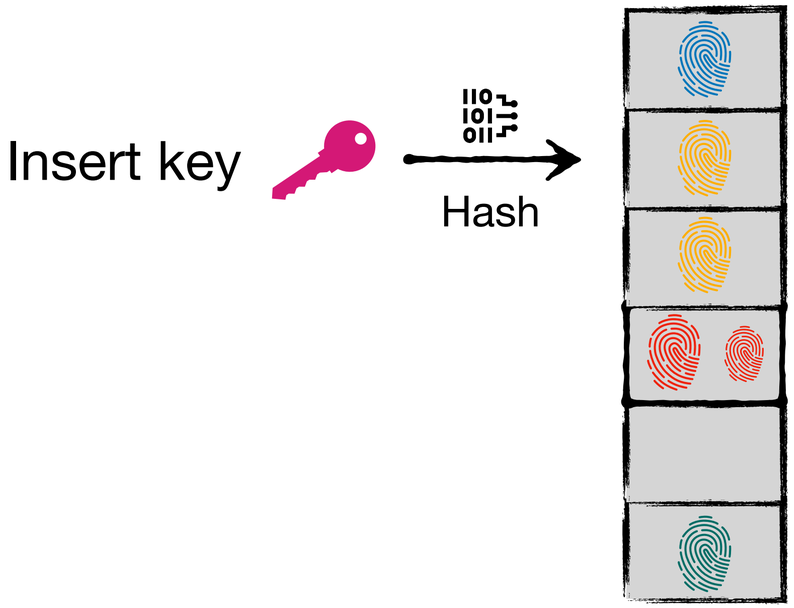


Adaptive filter
Memory

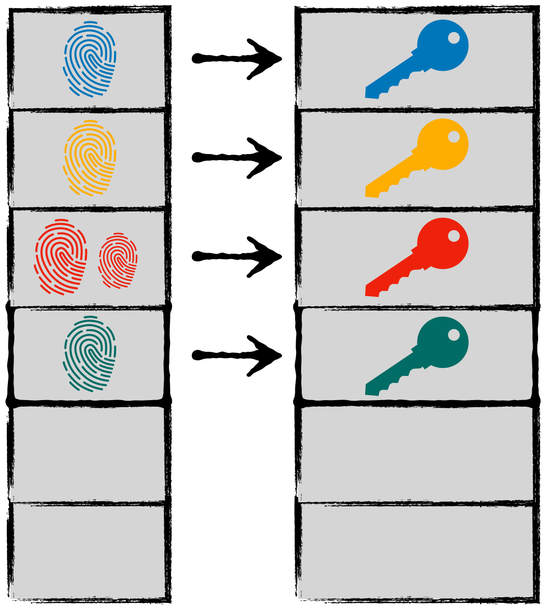


Fingerprint to Key map
Disk

Adaptive filters employ variable-length fingerprints

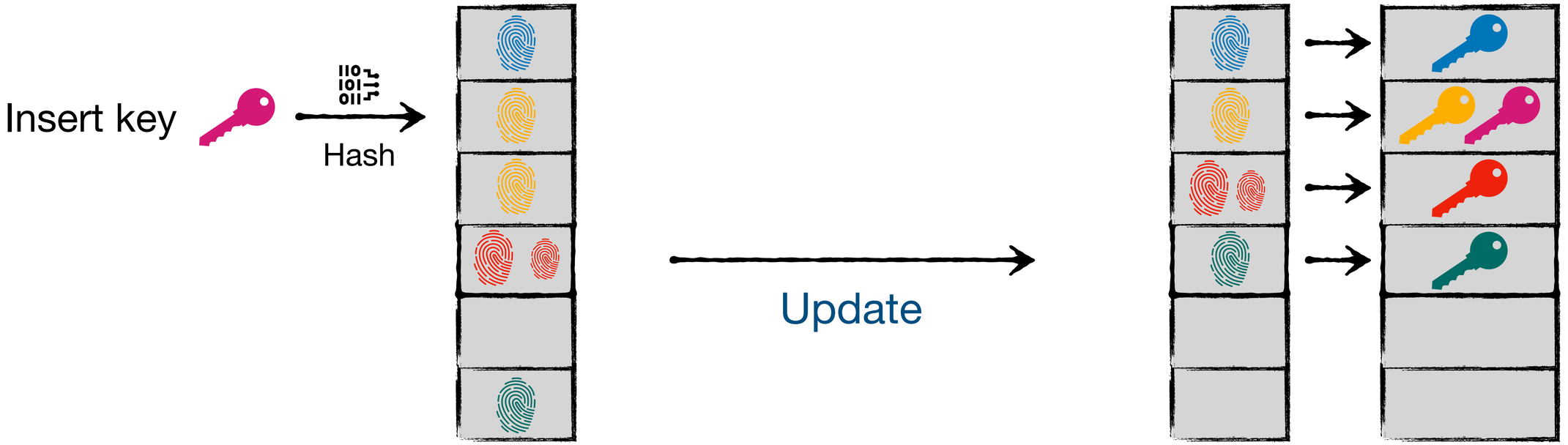


Adaptive filter
Memory



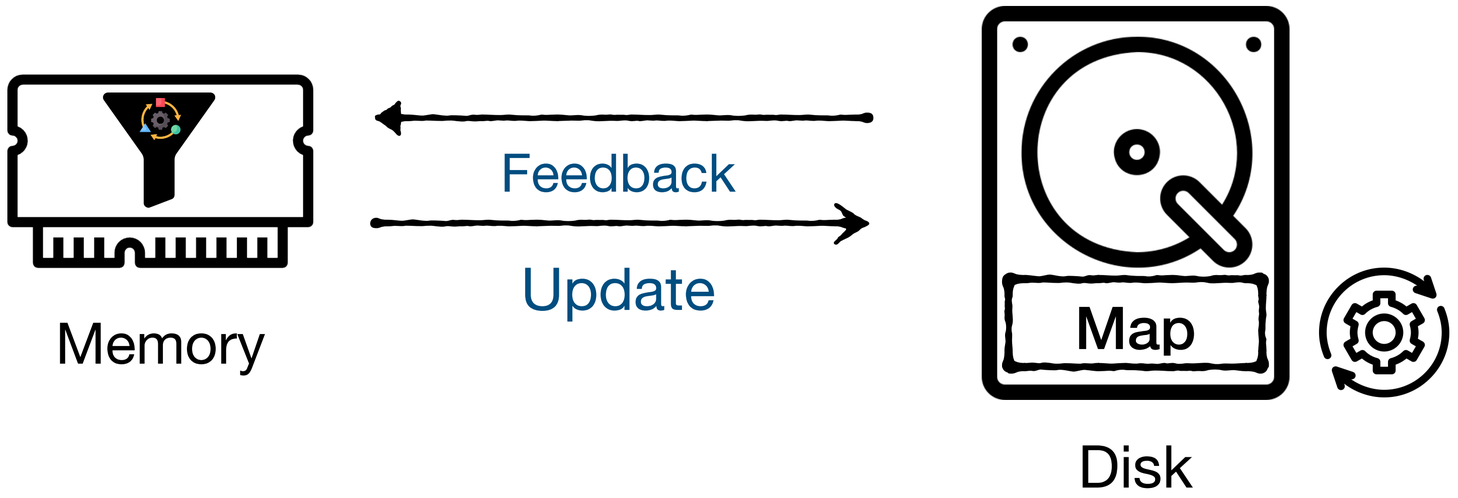
Fingerprint to Key map
Disk

Adaptive filters employ variable-length fingerprints



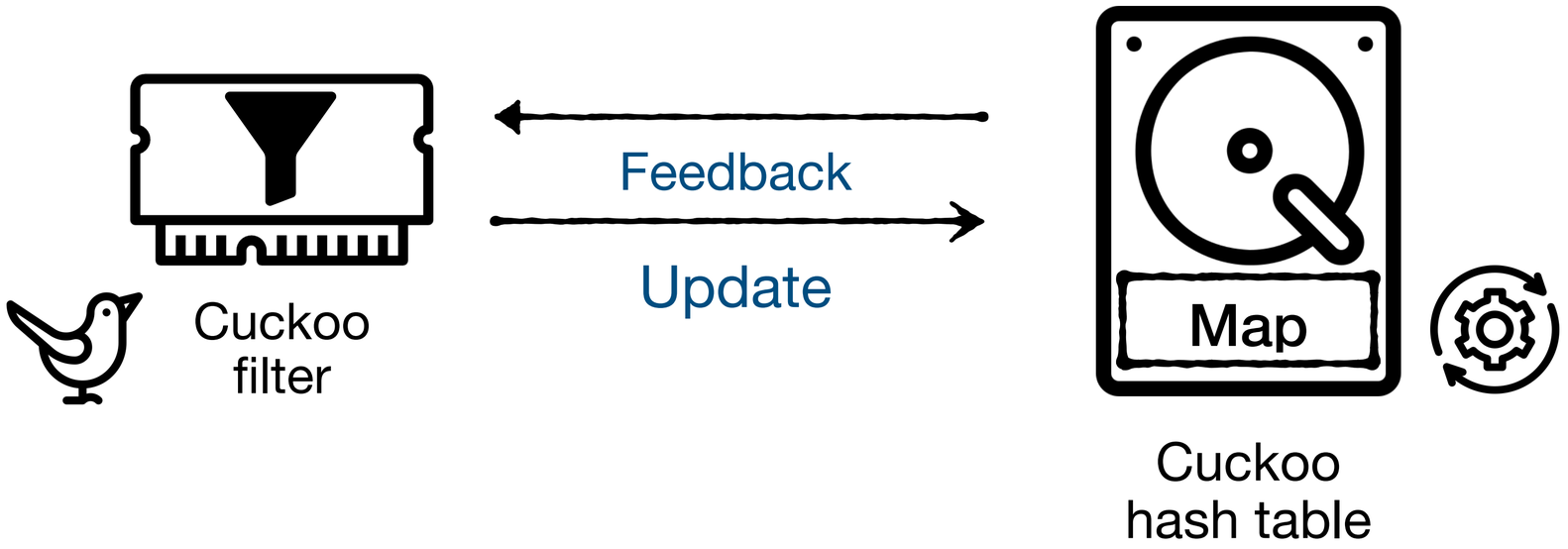
Fingerprint map is updated accordingly

Fingerprint map updates dominate the performance



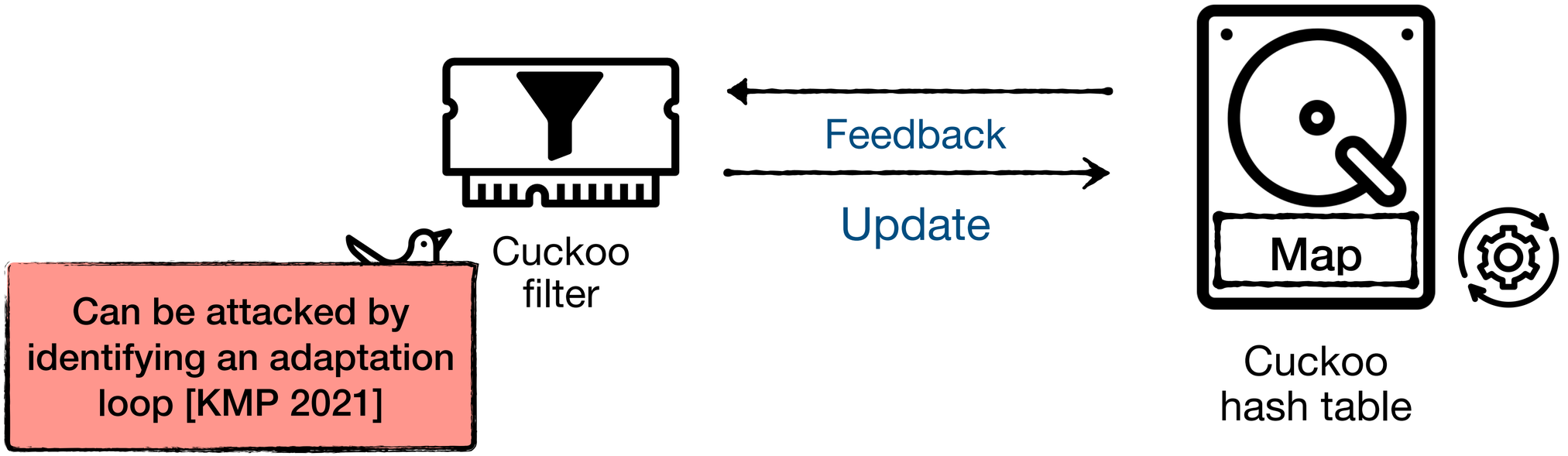
Minimizing the work in the map is crucial for the performance

Adaptive cuckoo filters [MPR+ 2020]



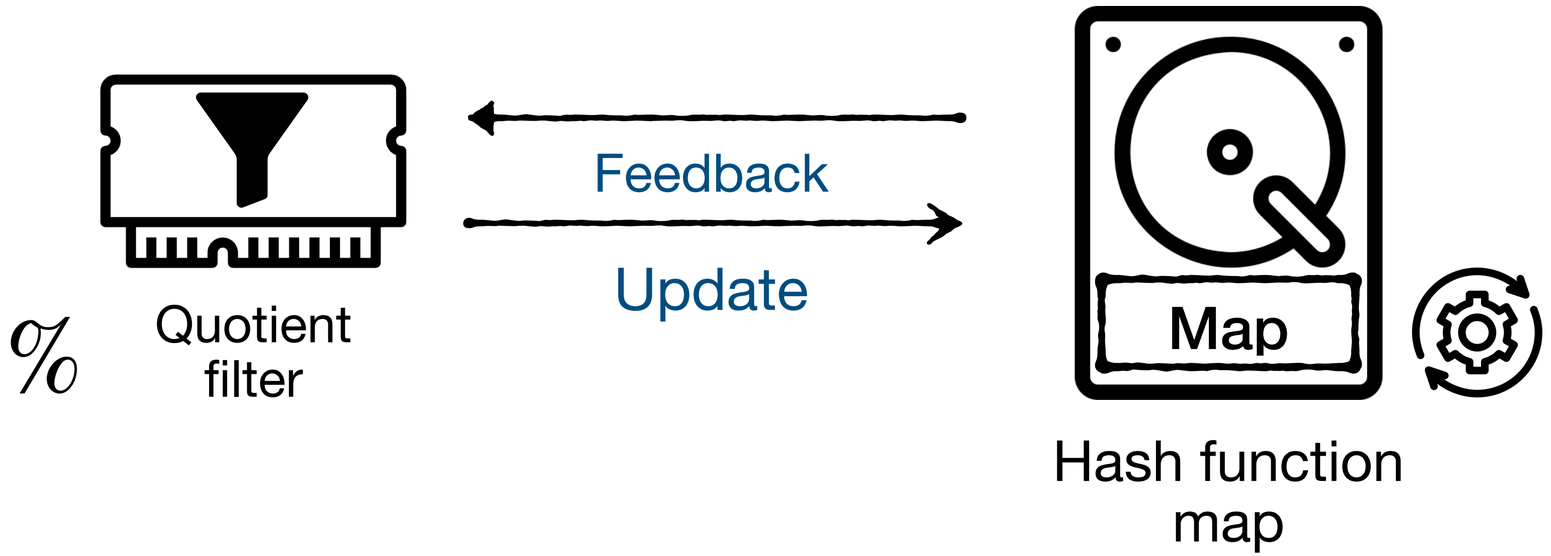
Adaptivity by moving fingerprints around

Adaptive cuckoo filters offer weak adaptivity



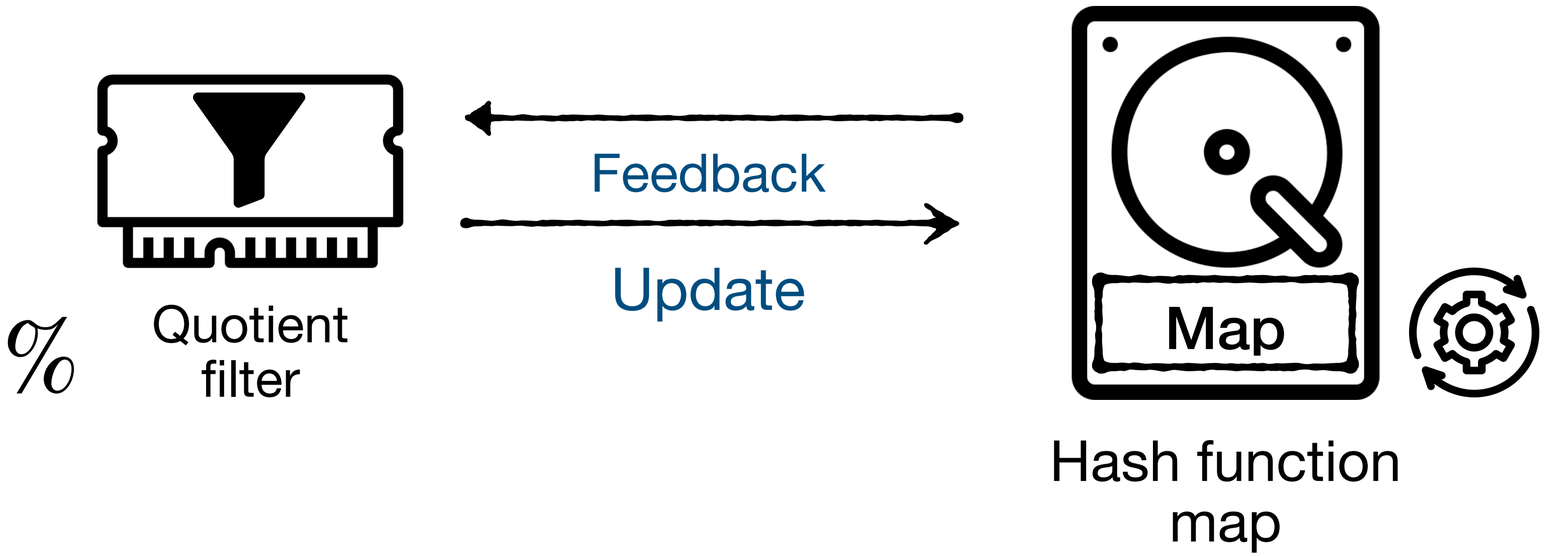
- Adaptivity by moving fingerprints around during insertions/queries
- Can forget previous false positives while adapting for new ones

Telescoping filters [LMS+ 2021]



Adaptivity by changing hash function during insertions/queries

Telescoping filters offer strong adaptivity



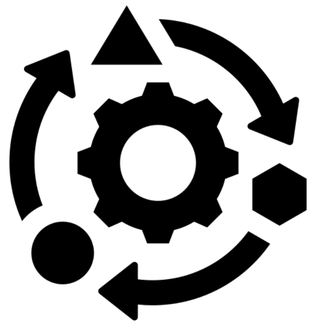
Adaptivity by changing hash function during insertions/queries

Hash map grows during adaptations (variable-length fingerprints)

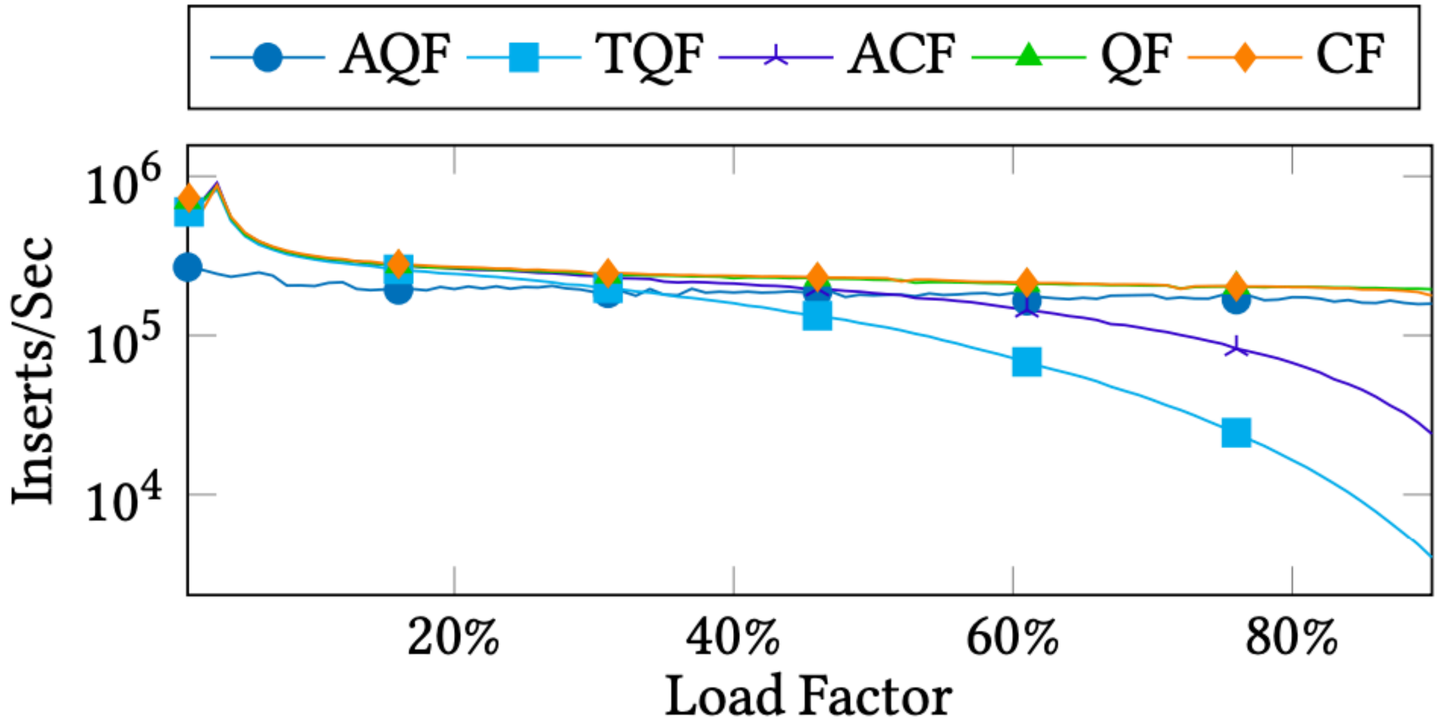
Does not forget previously learned fingerprints

Adaptive quotient filter [WMT+ SIGMOD 2025]

- Adaptivity by using variable-length fingerprints to avoid collisions
- Based on the counting quotient filter (CQF) [PBJ+ 2017]
- Matches the space lower-bound to lower-order terms
- 10X—30X faster than other adaptive filters (ACF, TF) for disk-based database benchmarks
- Up to 6X faster performance than traditional filters (QF, CF) for disk-based database benchmarks



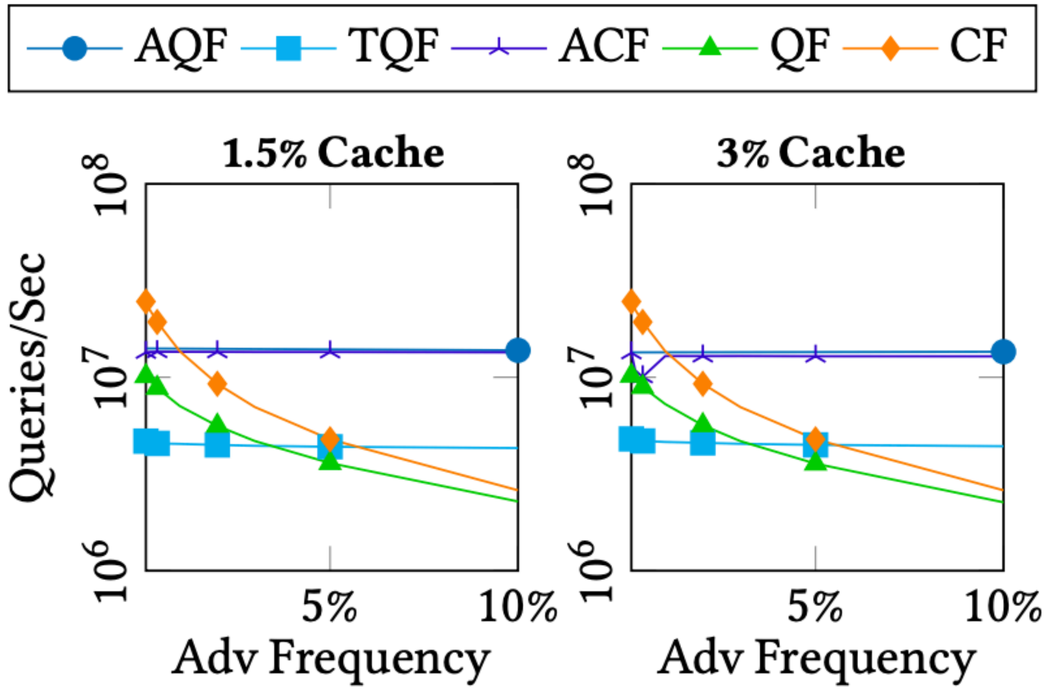
Database insertion performance



AQF performs similarly to QF/CF for database insertions

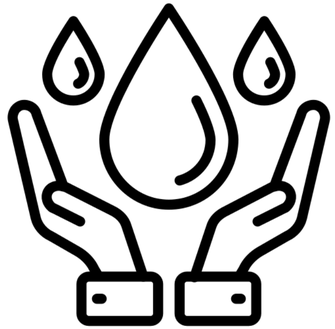
10X—30X faster than other adaptive filters

Database query performance

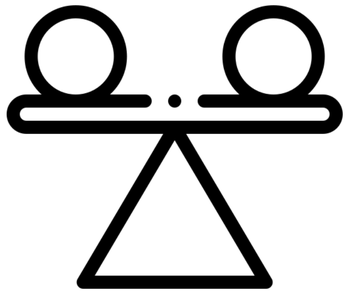


AQF up to 6X faster compared to QF/CF for database queries

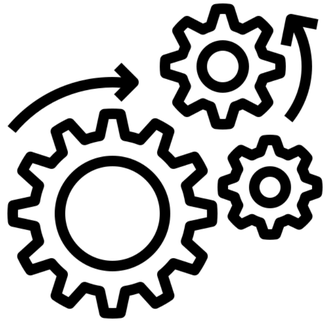
Adaptive quotient filter takeaways



Preserves CQF performance and features



Stable reverse map during insertions



Supports dynamic operations

Source code: https://github.com/splatlab/adaptiveqf

Concluding remarks

https://prashantpandey.github.io/

We need to develop new algorithmic paradigms to better leverage modern hardware

Data systems backed by strong theoretical guarantees are key to tackle future scalability challenges

Yuvaraj Chesetti

Zikun Wang

Hunter McCoy

Jamshed Khan

Acknowledgements:

National Science Foundation

U.S. DEPARTMENT OF ENERGY