Making adaptive filters a general-purpose system tool

Thesis proposal

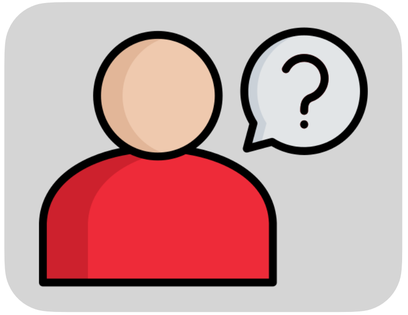
Yuvaraj Chesetti

Advisor: Prashant Pandey

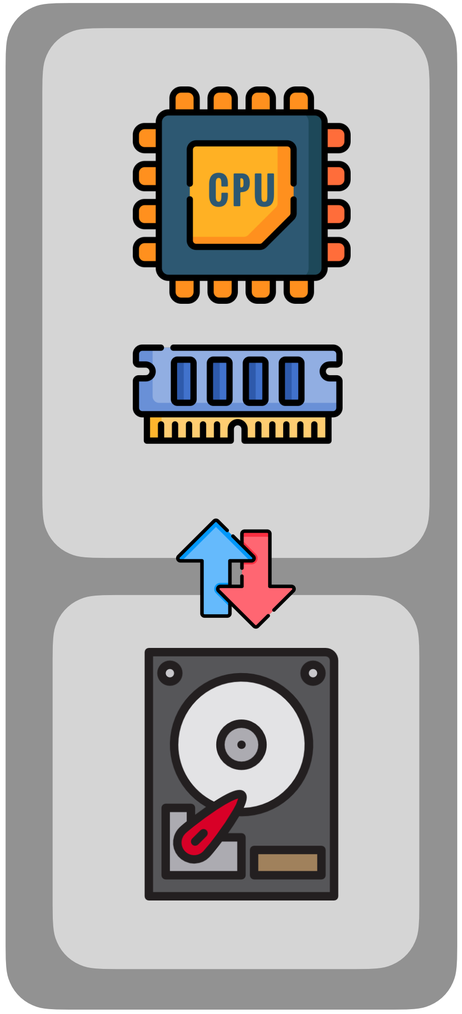
Committee: Mirek Riedewald, Jeff Phillips, Soheil Behnezhad

Northeastern University

Queries



Data System

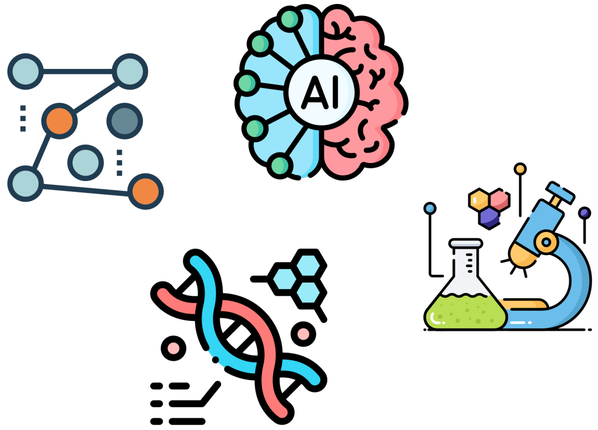


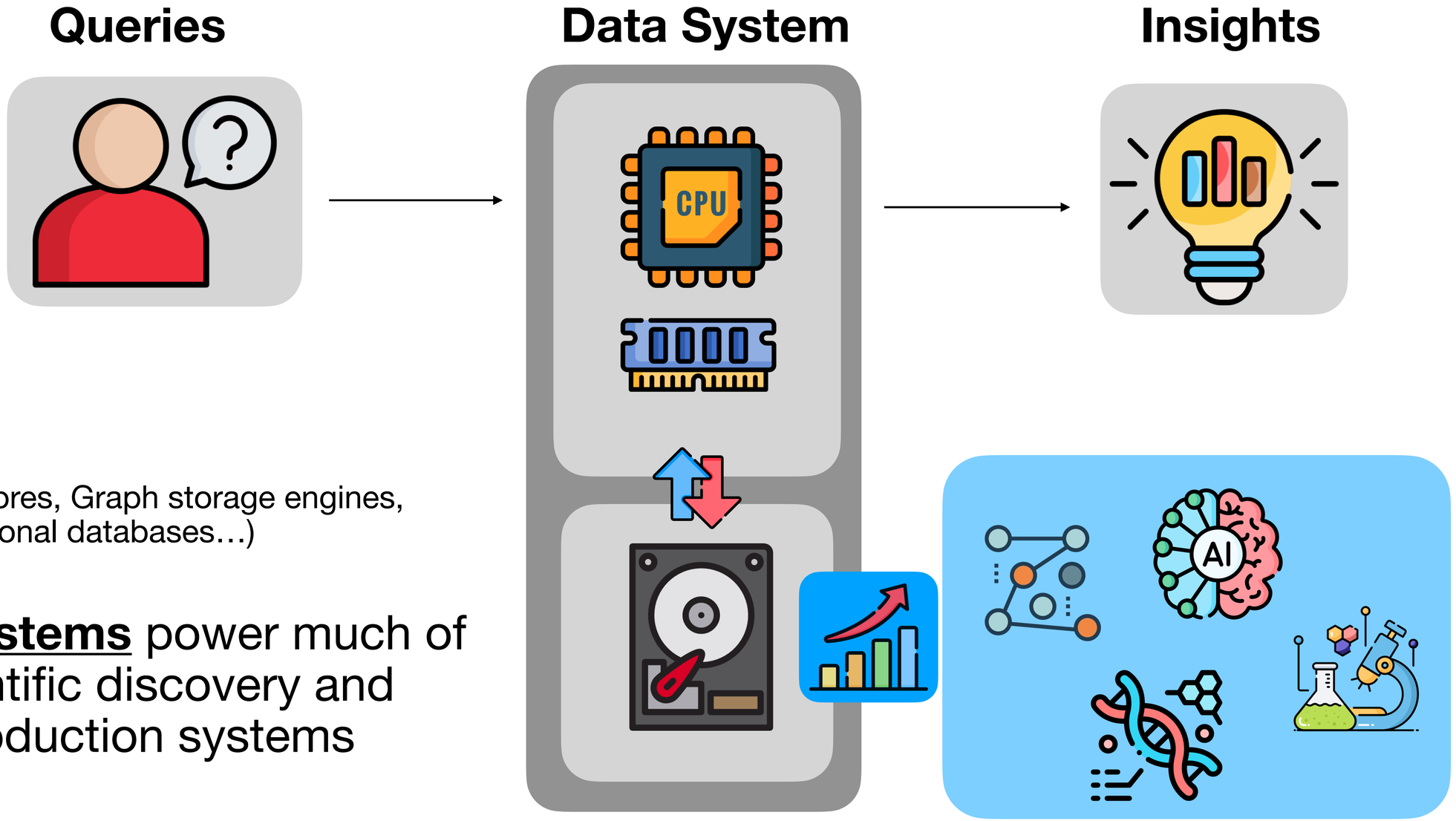
Insights

(Caches, KV-stores, Graph storage engines, relational databases…)

↑

<u>Data systems</u> power much of scientific discovery and production systems

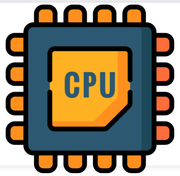
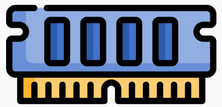




(Caches, KV-stores, Graph storage engines, relational databases…)

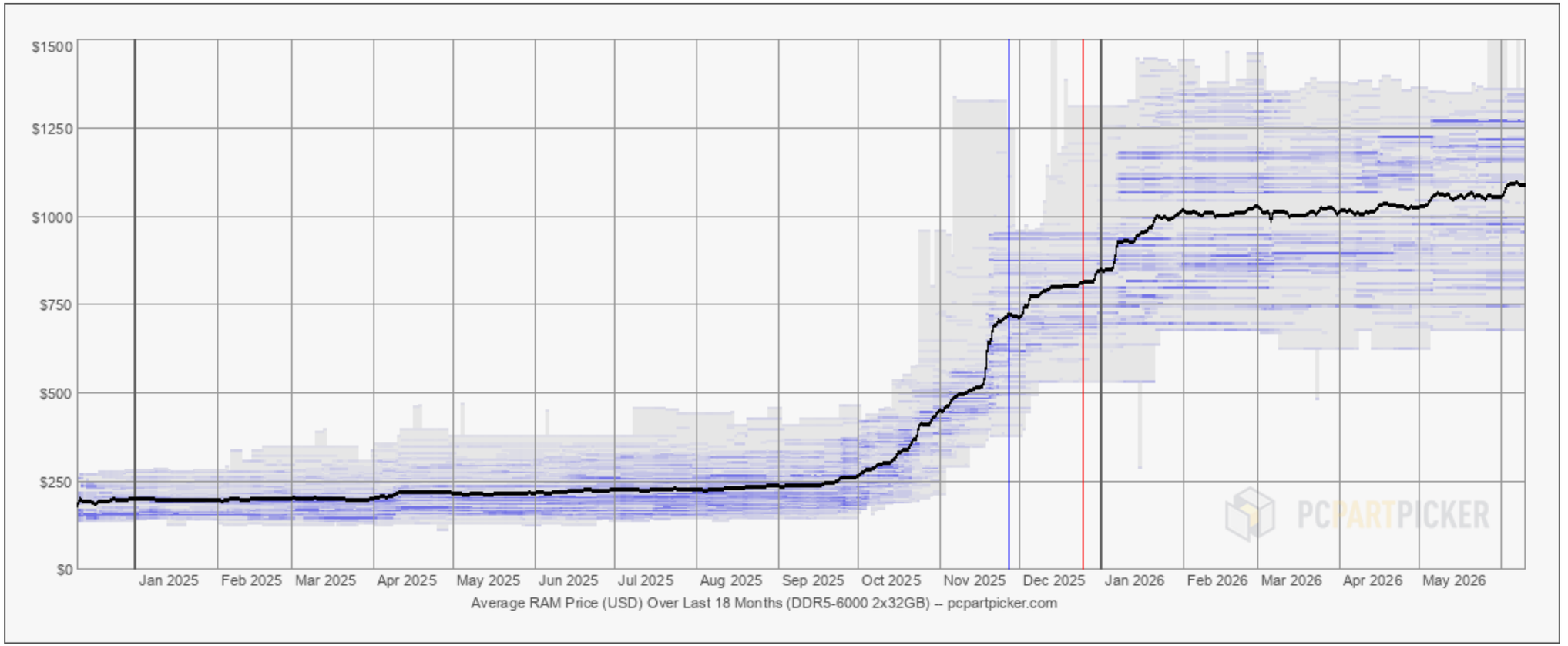
Data systems power much of scientific discovery and production systems

Data movement is a bottleneck

OPERATION		LATENCY	HUMAN SCALE
L1 cache reference Fastest CPU-local memory, ~32–48 KB per core		1 ns	1 sec
Main memory reference (DDR5) RAM access after a cache miss		80 ns	1.3 min
NVMe SSD random read Modern PCIe-attached flash storage		15 µs 15,000 ns	4 hr
Disk seek (Rotational 7200 RPM HDD) Mechanical head movement on a spinning platter		5 ms 5,000,000 ns	2 mo

https://omarish.com/latency-numbers-you-should-know

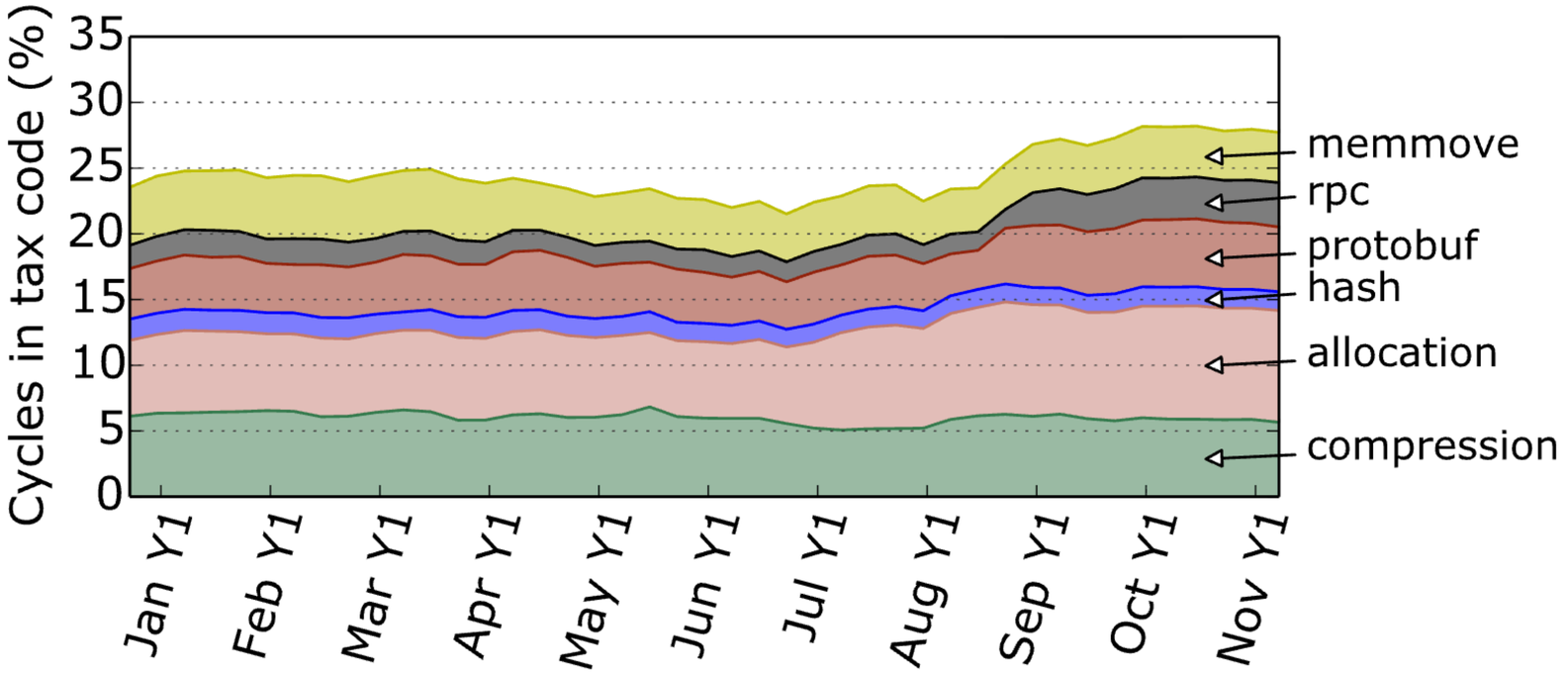
The RAMmageddon is here



DDR5-6000 2x32GB (Average price in USD over last 18 months)

Source: https://pcpartpicker.com/trends/price/memory/

CPU cost of data representation

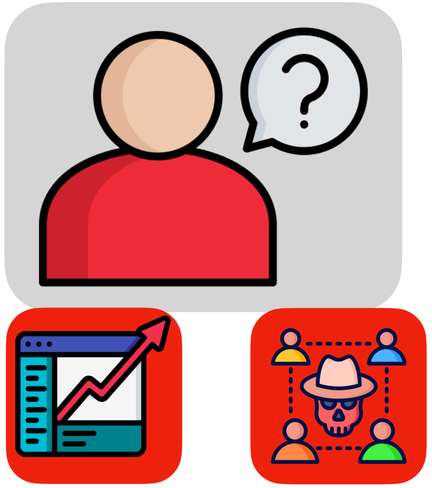


The Datacenter Tax:

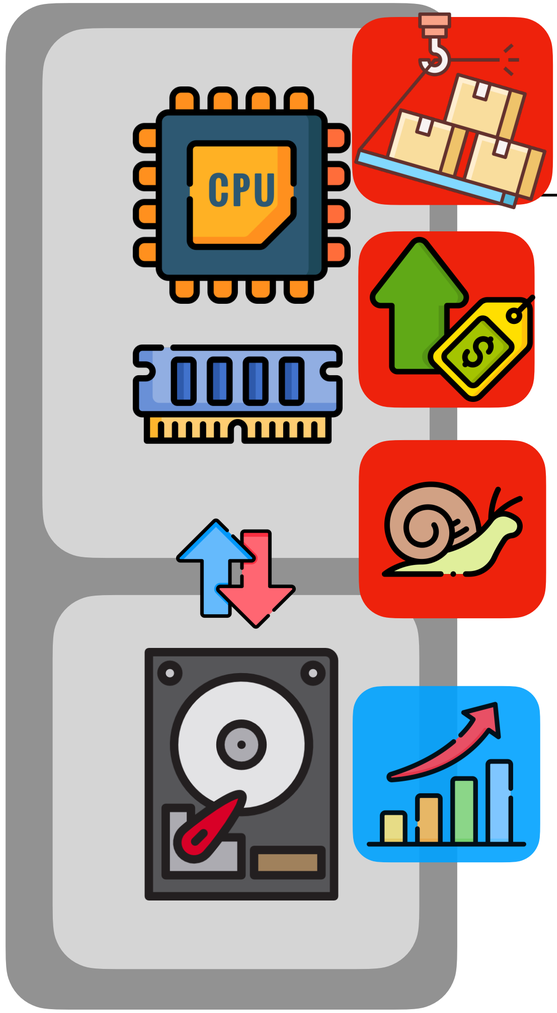
22-27% CPU cycles are spent in representing and moving data

Svilen Kanev, Juan Pablo Darago, Kim Hazelwood, Parthasarathy Ranganathan, Tipp Moseley, Gu-Yeon Wei, and David Brooks. 2015. Profiling a warehouse-scale computer.

Queries



Data System



Insights

Robustness is a critical requirement

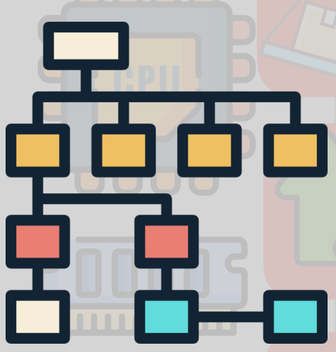
Scalability is a challenge

Data volume increasing exponentially

Queries

Data System

Insights



Scalability is a challenge

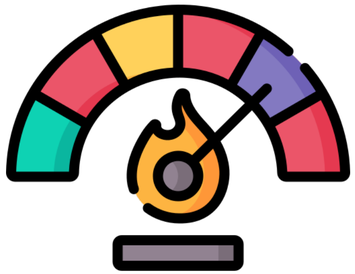
Robustness is a critical requirement

Data volume increasing exponentially

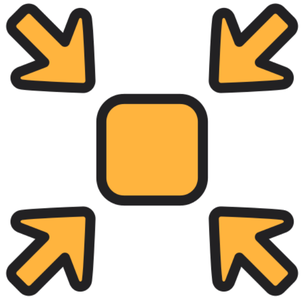
Data structure design is central to improving our data systems

how we compress, organize, move and access data

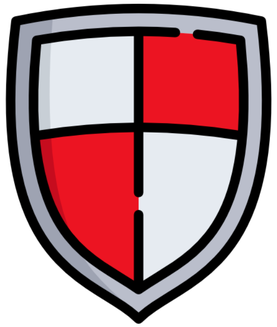
Design objectives for scalability and robustness



High performance



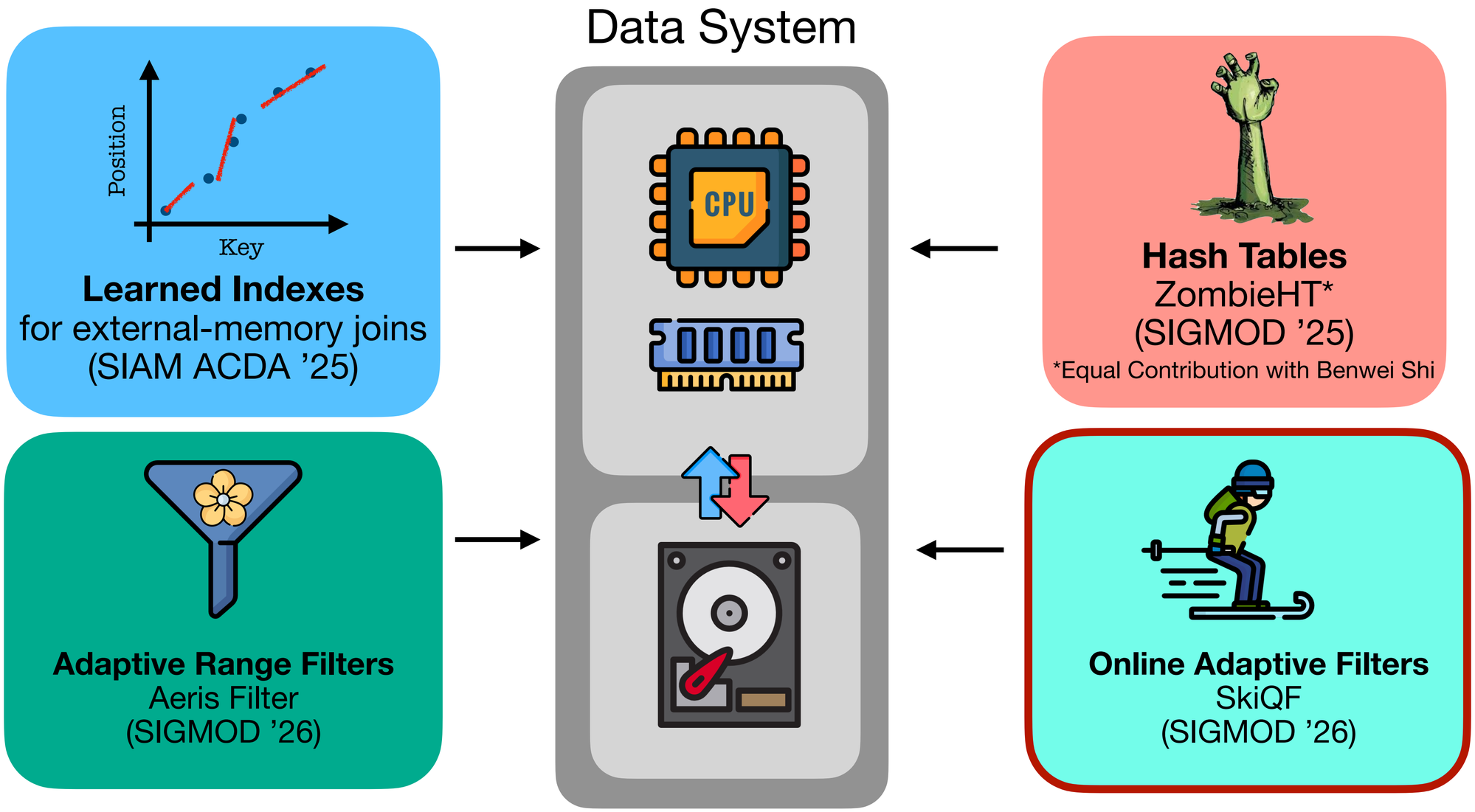
Space efficiency



Rigorous guarantees

Achieving all three at once is a long-standing open challenge!

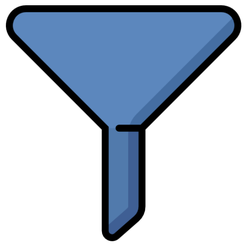
Research overview



Adaptive filters

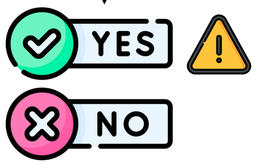
What is a filter?

Filter



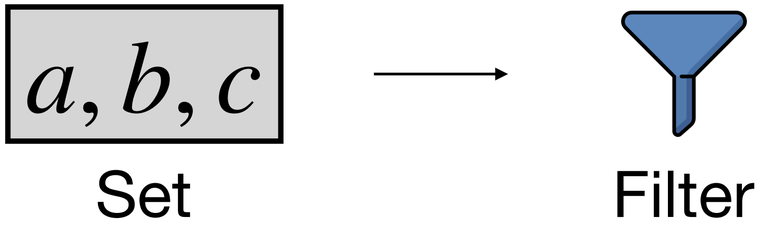
Answer membership queries with false positive probability ϵ

Does the dataset contain this key?



Non-existent key specified as present with probability ϵ

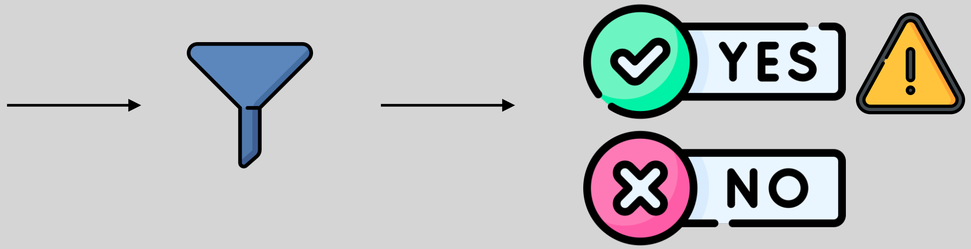
$0 < \epsilon < 1$



Is ‘a’ in the set?

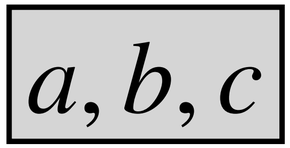
No false negatives

Is ‘q’ in the set?



False positive with probability $\leq \epsilon$

False positives enable filters to be compact



Set

Filter

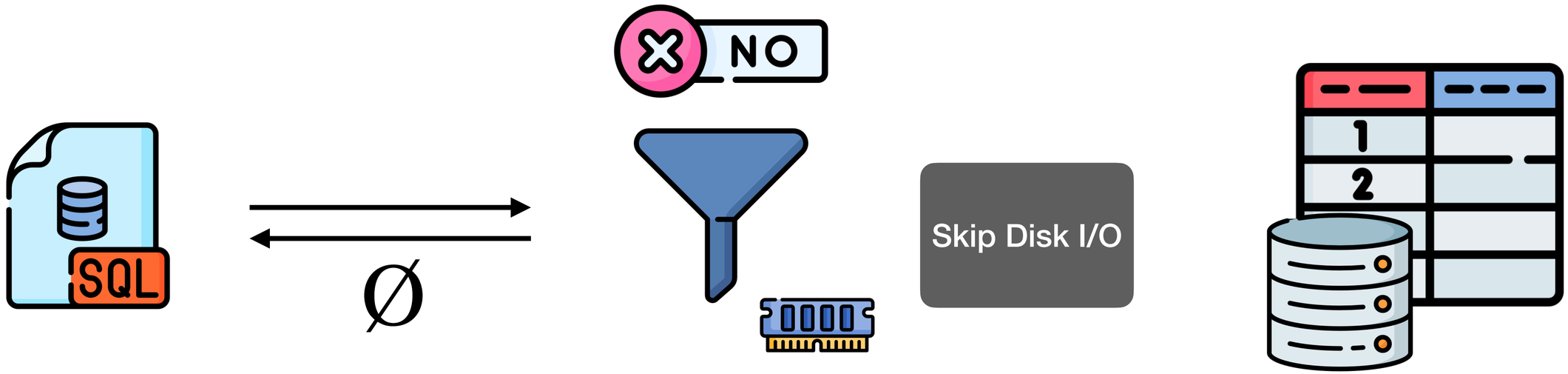
Space (bits)

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

$\Omega(n \log(1/\epsilon))$

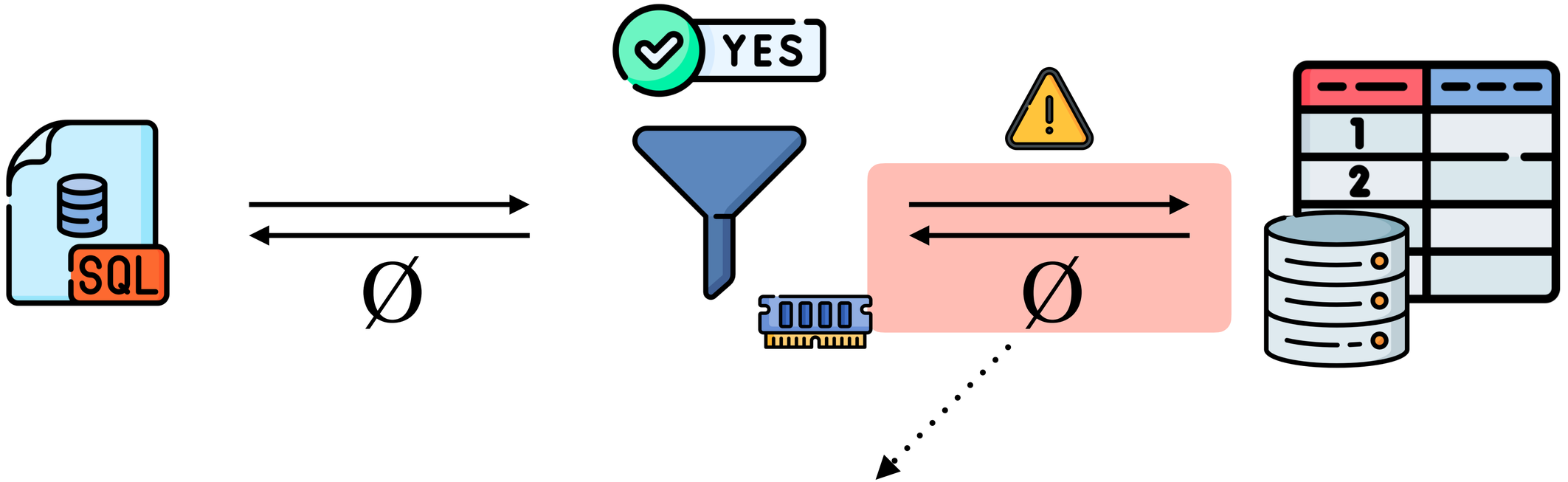
For $\epsilon = 2\,\%$, filters require ~1 Byte/item. Hash table/Tree can take >8-16 Byte/item

How filters are used in databases



Filters block disk I/Os for keys that don’t exist

The cost of false positives



False positives incur disk I/O for non-existent keys

Traditional filters repeat false positive responses

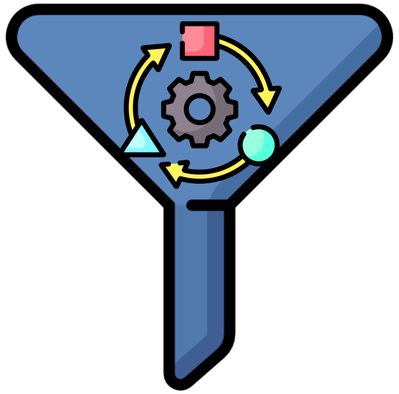
False positive rate guarantees only hold for a single query in isolation

Unbounded false positives, **up to n in worst case**

n : number of queries

Adaptive filters [BFG+'18]

An adaptive filter modifies its state upon feedback



Is **X** in set? YES (**False positive**)

Is **X** in set? NO (**True negative**)

Is **X** in set? NO (**True negative**)

Time

<u>Any distribution</u>: At most $(\epsilon \cdot n)$ false positives w.h.p

ϵ : false positive probability, n : number of queries

[BFG+2018]: Bender, M. A., Farach-Colton, M., Goswami, M., Johnson, R., McCauley, S., & Singh, S. (2018, October). Bloom filters, adaptivity, and the dictionary problem.

From theory to implementation

Theoretical

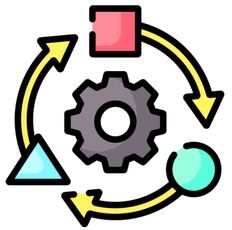
Broom filter

[BFG+’18]

Implementations

Telescoping filter

[LMSS’21]



AdaptiveQF

[WMT+’24]

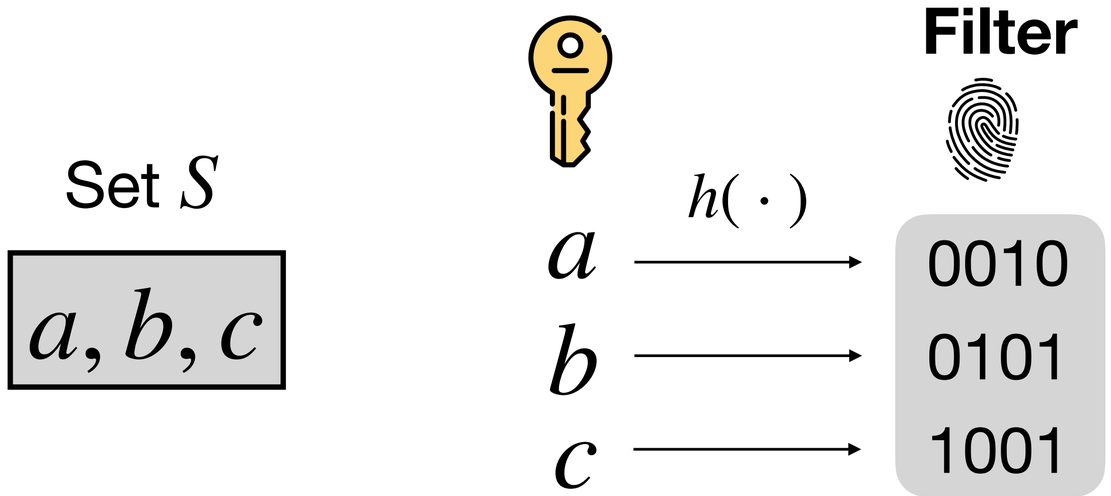
[BFG+2018]: Bender, M. A., Farach-Colton, M., Goswami, M., Johnson, R., McCauley, S., & Singh, S. (2018, October). Bloom filters, adaptivity, and the dictionary problem.

[LMSS+2021]: Lee, David J., Samuel McCauley, Shikha Singh, and Max Stein. Telescoping Filter: A Practical Adaptive Filter.

[WMT+2024]: Wen, Richard, Hunter McCoy, David Tench, Guido Tagliavini, Michael A. Bender, Alex Conway, Martin Farach-Colton, Rob Johnson, and Prashant Pandey. Adaptive quotient filters

How adaptive filters work?

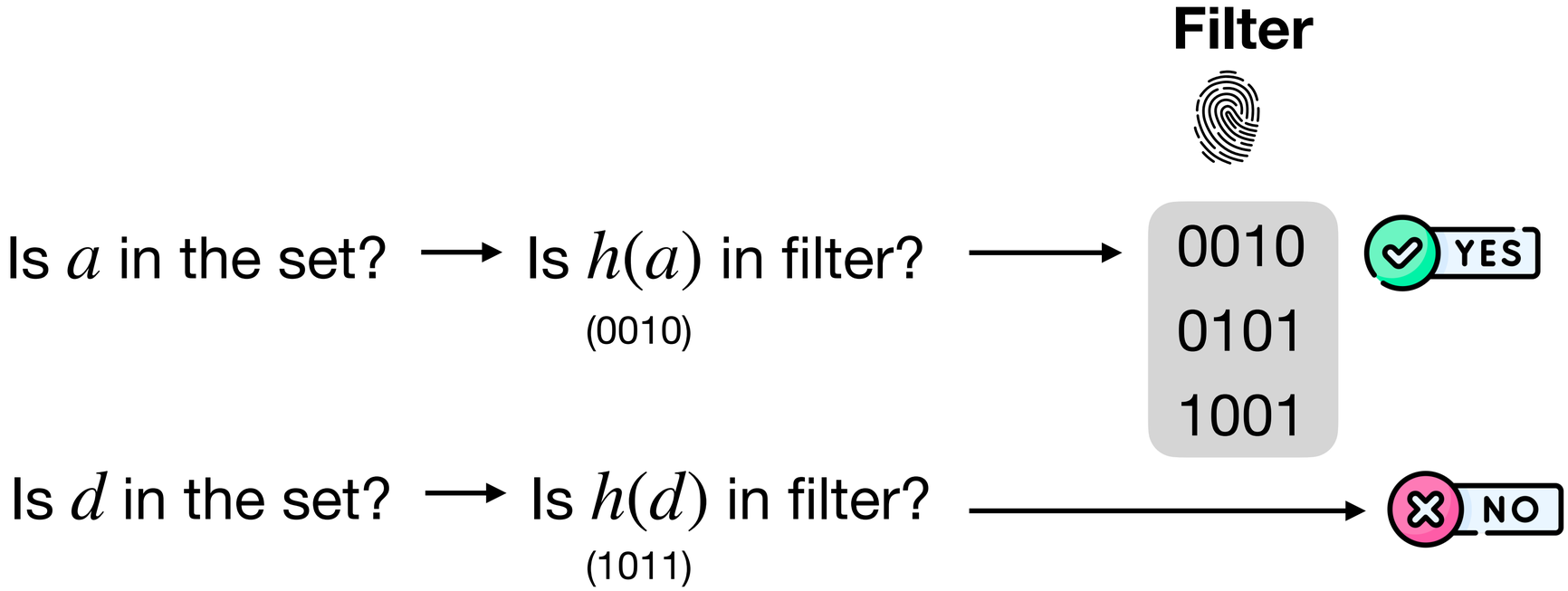
Adaptive filters are fingerprint filters



Each key is mapped to small fingerprint using a hash function

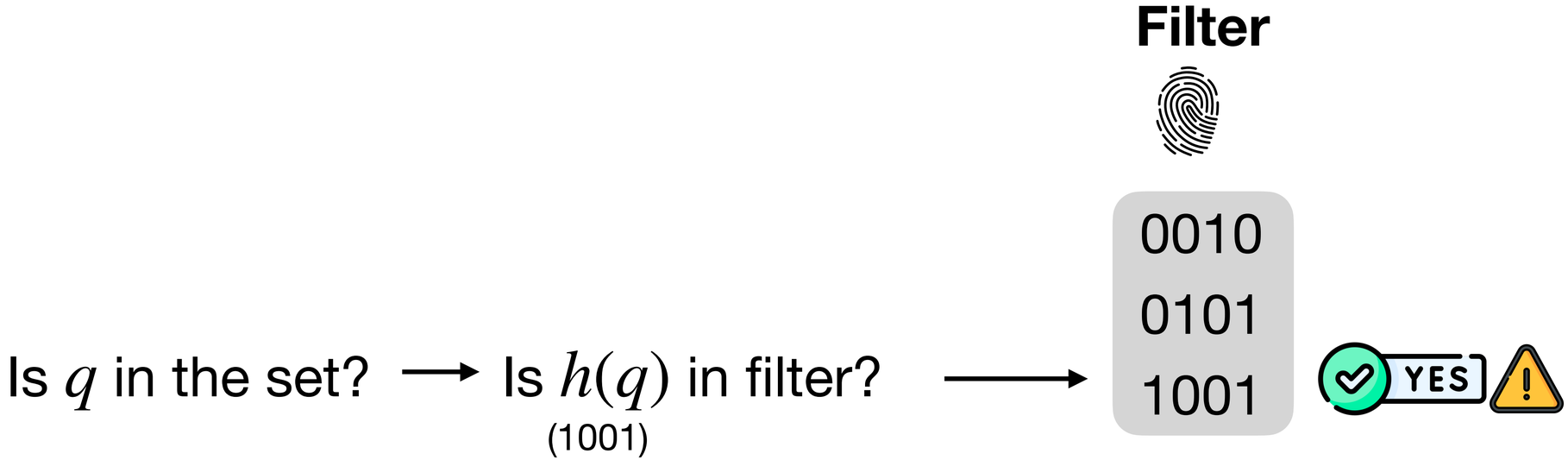
Fingerprints are stored compactly in a hash table

How fingerprint filters test for membership



Queries check if corresponding fingerprint is in filter

Reason for false positives in fingerprint filters

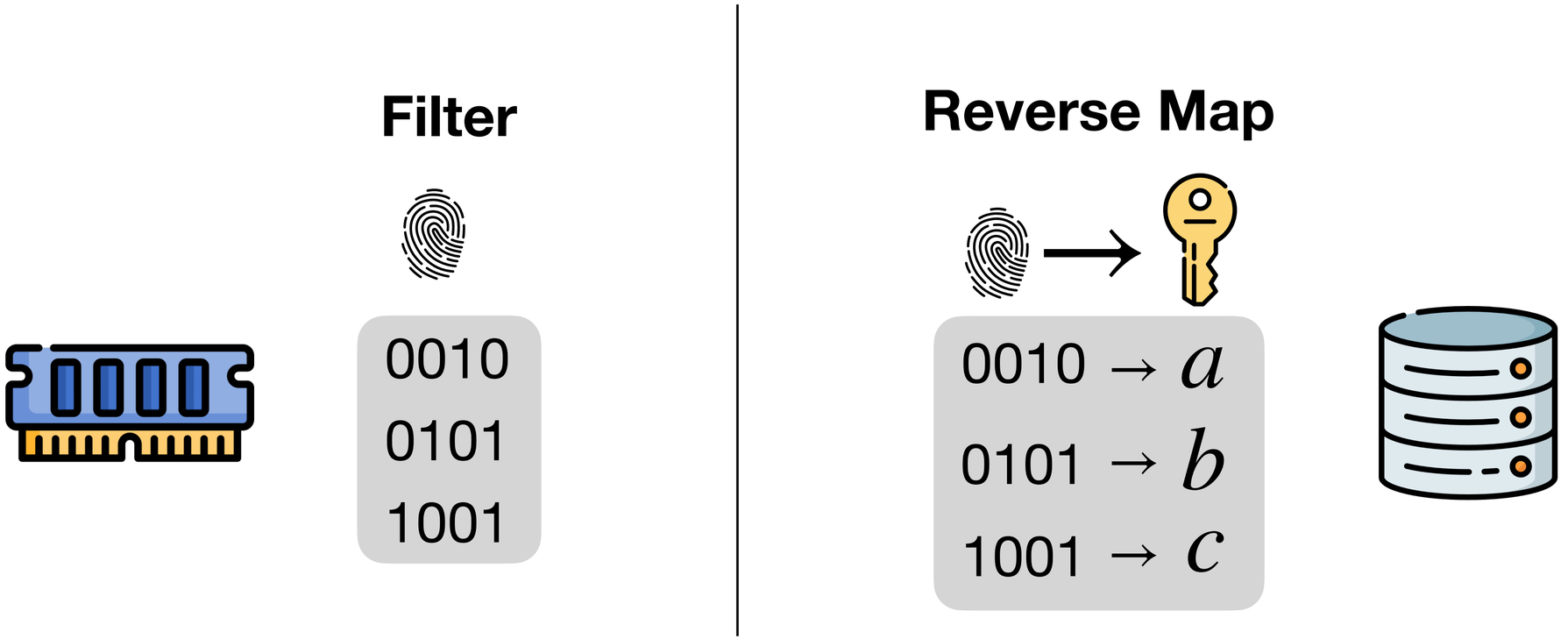


False positive when fingerprint of a non-existent key collides with key in filter

$$h(q) = h(c) = 1001$$

$$q \notin S, c \in S$$

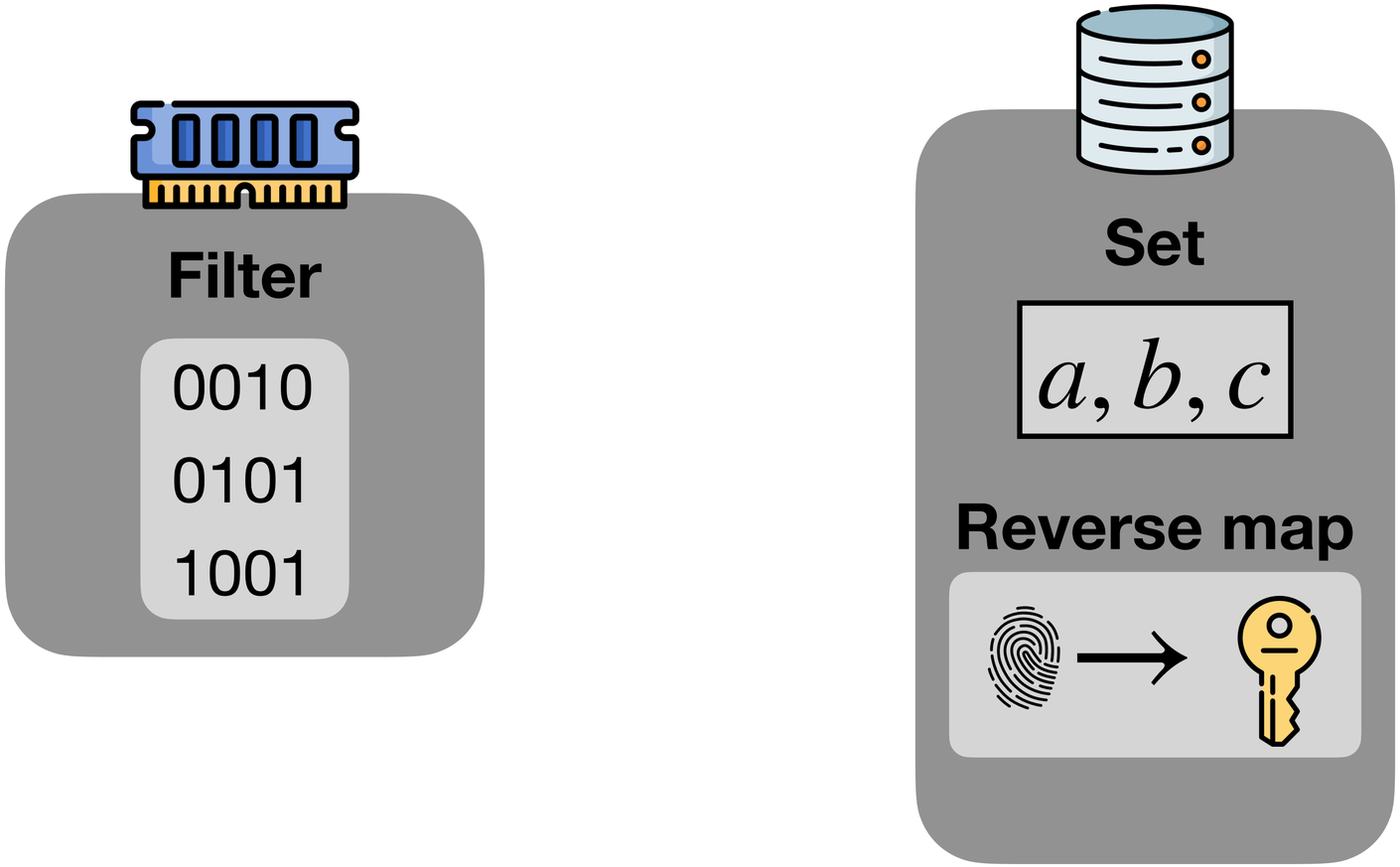
Design of adaptive filters



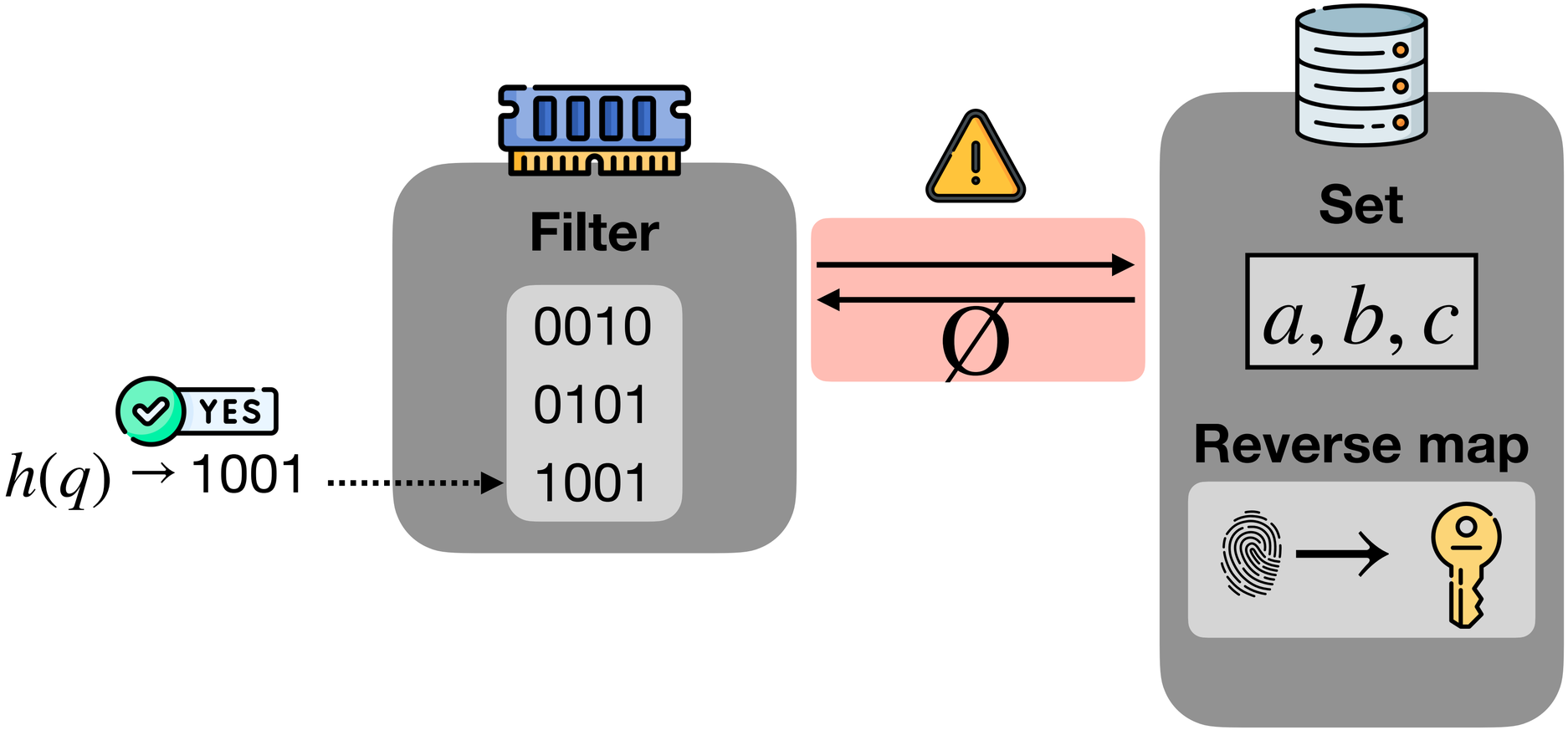
Space lower bound of adaptive filters: $\Omega(\min(n \log\log u, n \log n))$

Split across main memory and disk

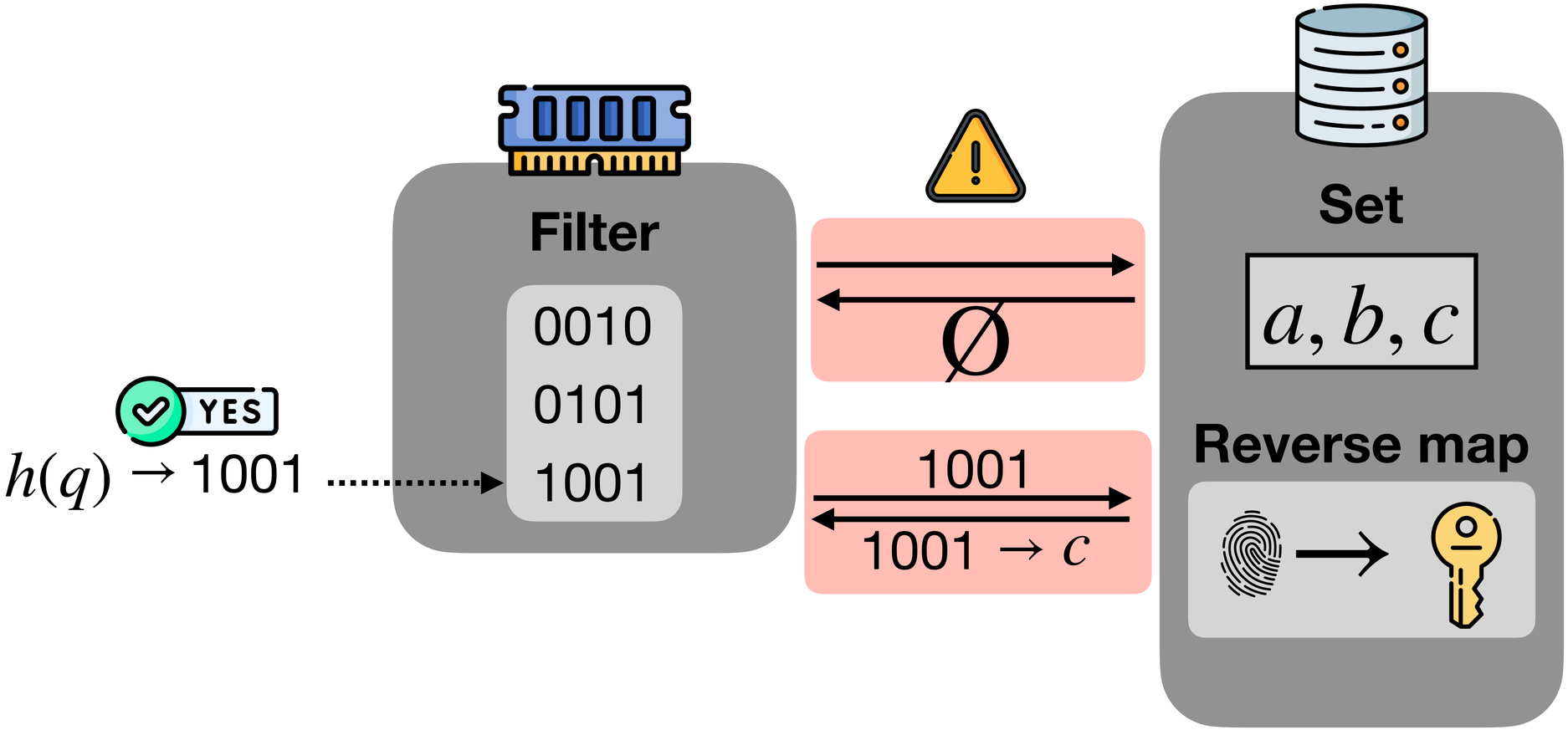
How adaptive filters adapt to false positives



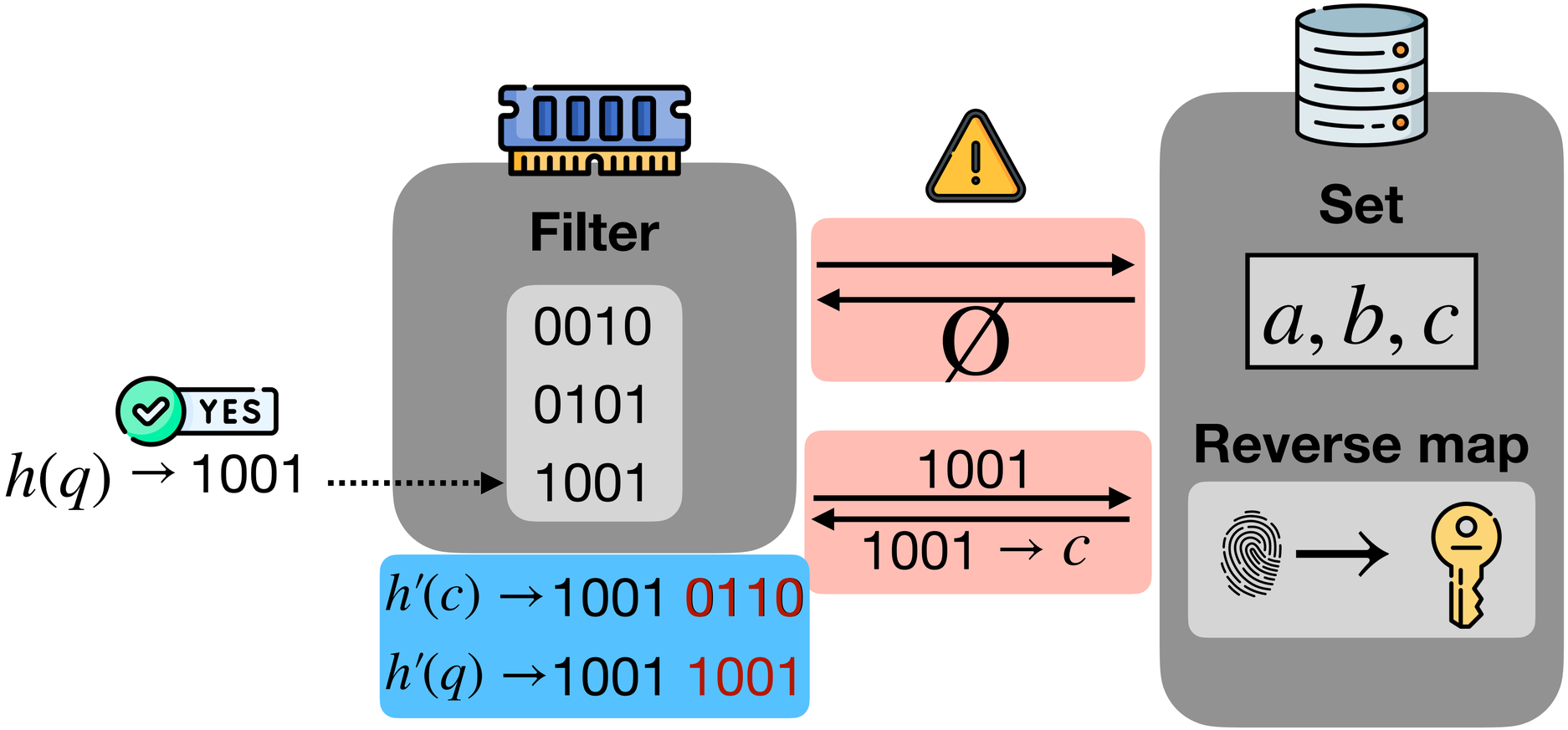
Filter receives feedback on false positive



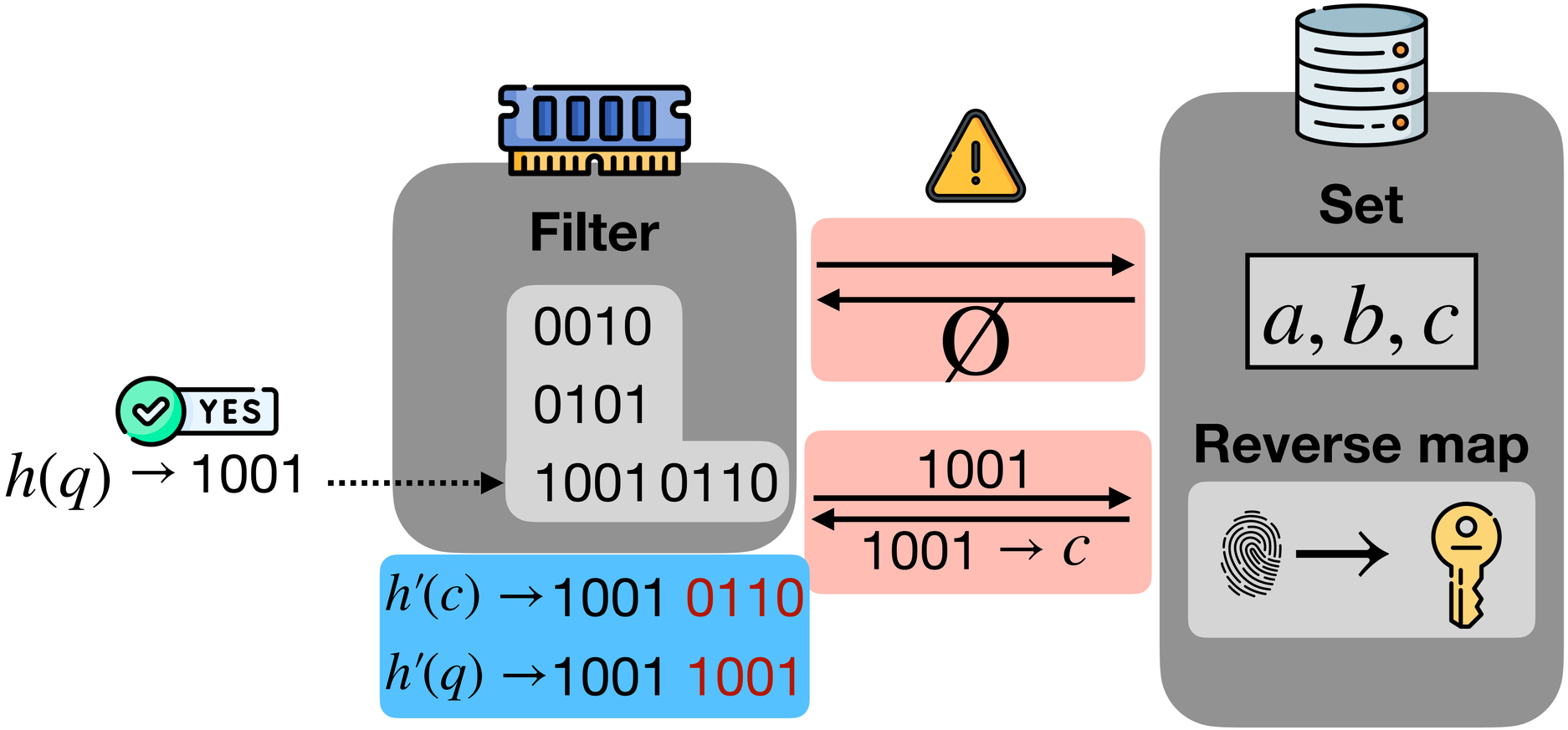
Filter queries reverse map to find colliding key



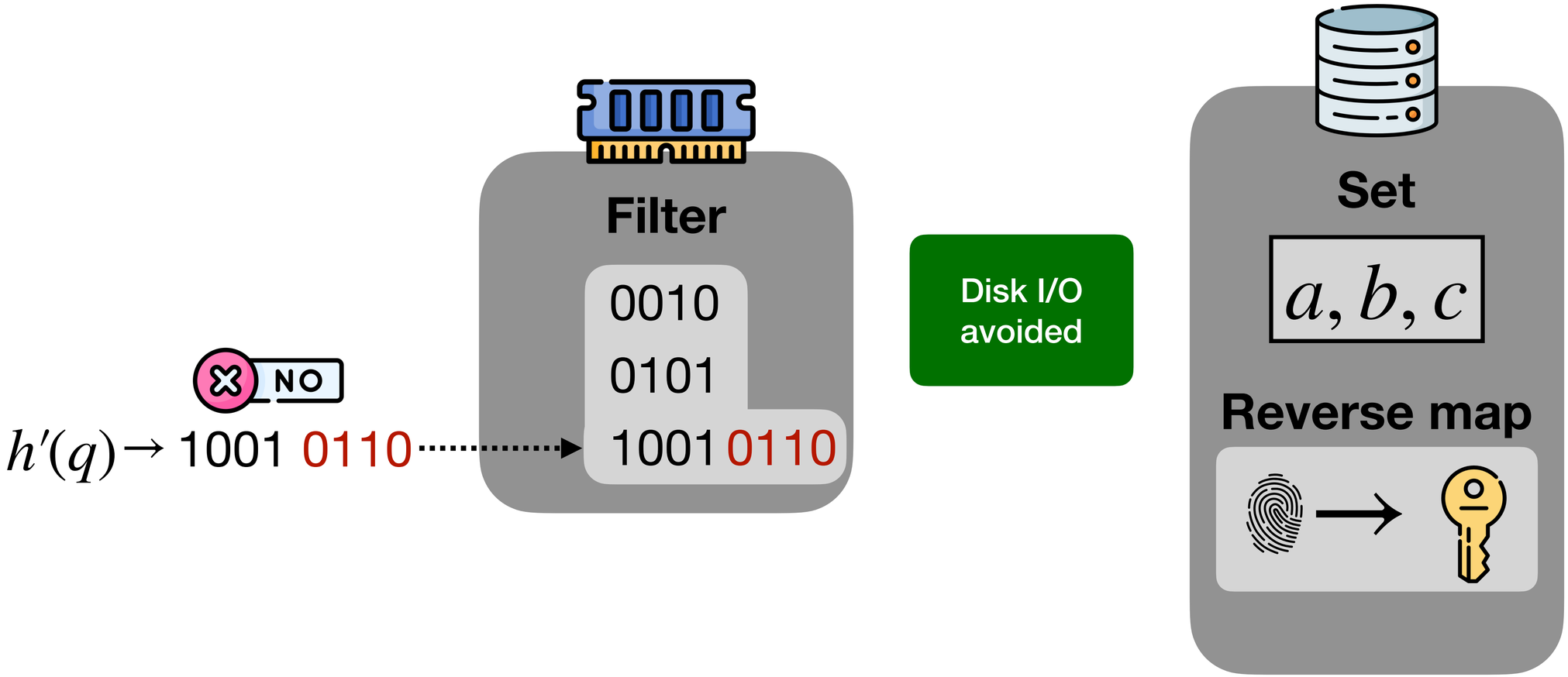
Fingerprint extended until bits differ



Filter updated with extended fingerprint

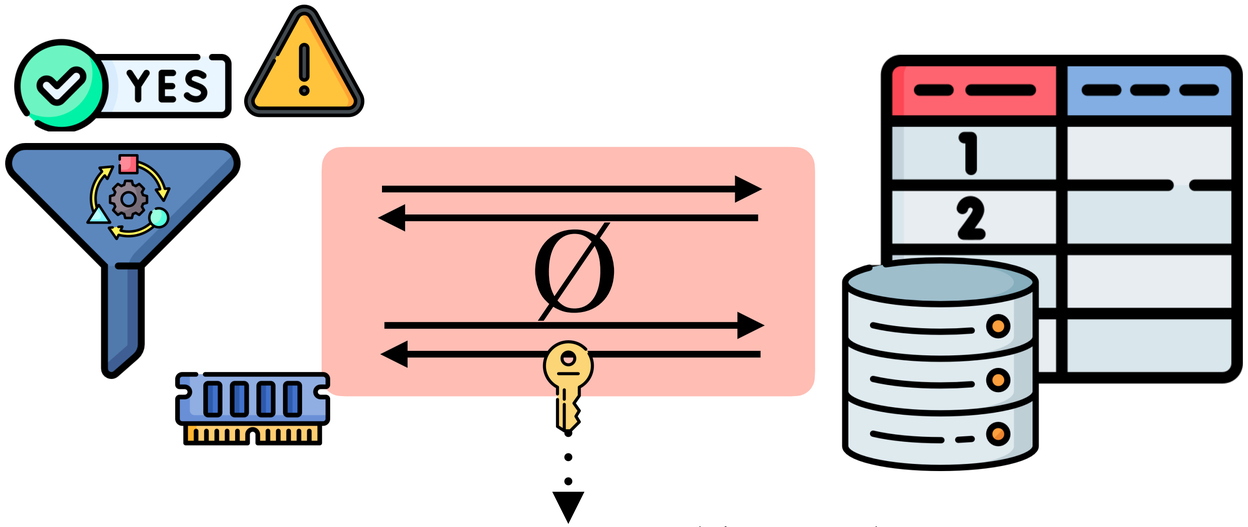


Adapted filter does not repeat false positive



Practical challenges in adaptive filter deployment

C1: Adaptive filters sometimes have I/O overhead

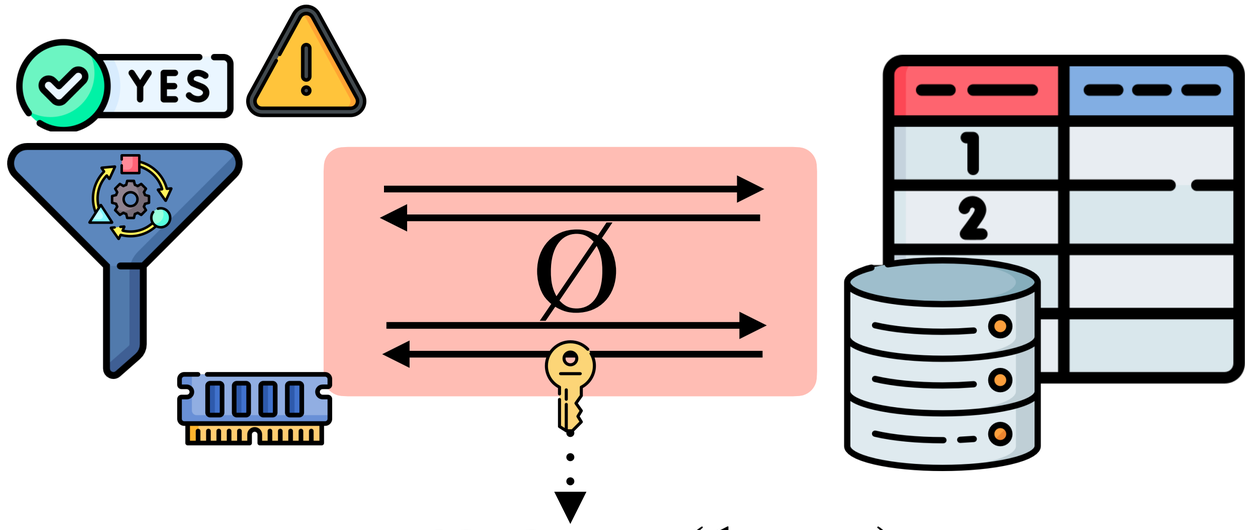


Disk I/O Cost: $(1 + \alpha)$

α: relative extra I/O to fix false positive

If false positives don't repeat, the extra I/O for adapting is an overhead

C1: Adaptive filters sometimes have I/O overhead



Disk I/O Cost: $(1 + \alpha)$

α: relative extra I/O to fix false positive

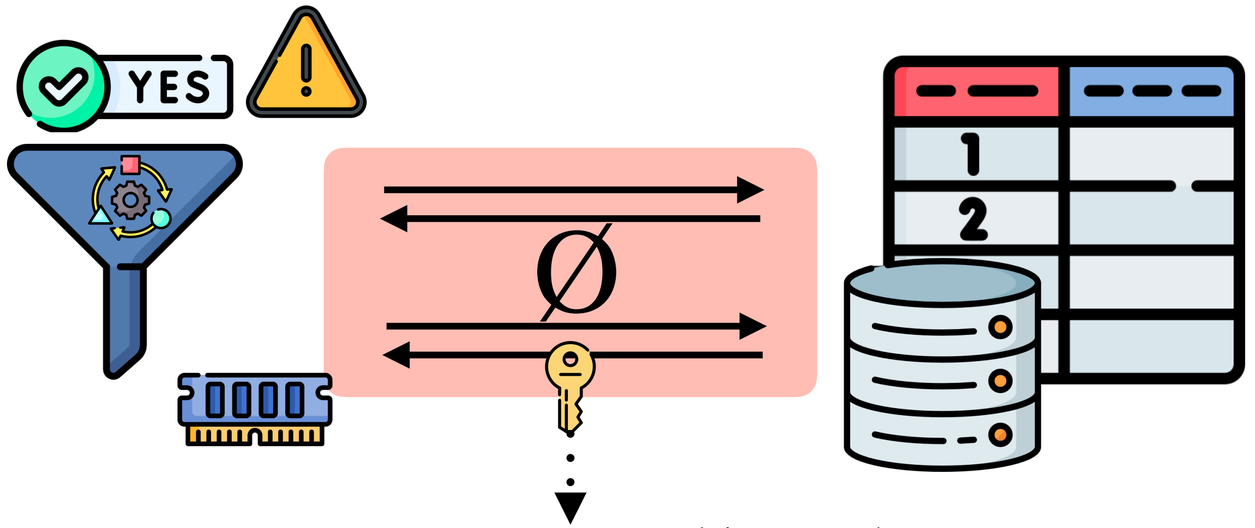
If false positives don't repeat, the extra I/O for adapting is an overhead

Yuvaraj Chesetti and Prashant Pandey. **To Adapt or Not to Adapt, That is the Ski Question. SIGMOD 2026**

Addressing I/O overhead

To adapt or not to adapt? That is the Ski question

Adaptive filters increase false positive cost



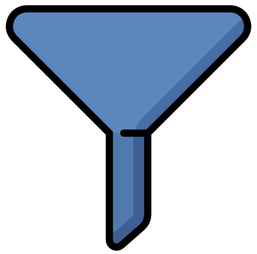
Disk I/O Cost: $(1 + \alpha)$

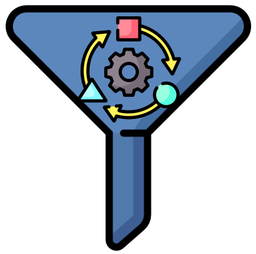
α: relative extra I/O to fix false positive

If false positives don't repeat, the extra I/O for adapting is an overhead

Yuvaraj Chesetti and Prashant Pandey. 2026. **To Adapt or Not to Adapt, That is the Ski Question.** Proc. ACM Manag. Data 4, 3, Article 244 (June 2026), 27 pages. https://doi.org/10.1145/3802121

False positive disk I/Os across distributions





Disk I/O cost	1	$(1+\alpha)$
Skewed	$\leq n$	$(1+\alpha)\cdot(\epsilon n)$
Uniform random	ϵn	$(1+\alpha)\cdot(\epsilon n)$

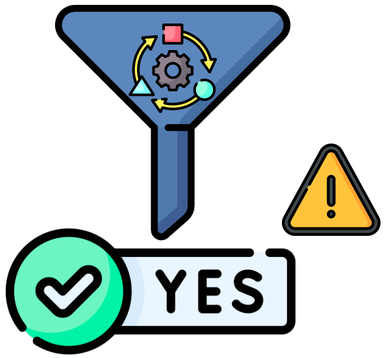
α: relative extra I/O to fix false positive

ϵ: false positive probability of single query

n: number of queries

Online adaptivity

Is **X** in the dataset?

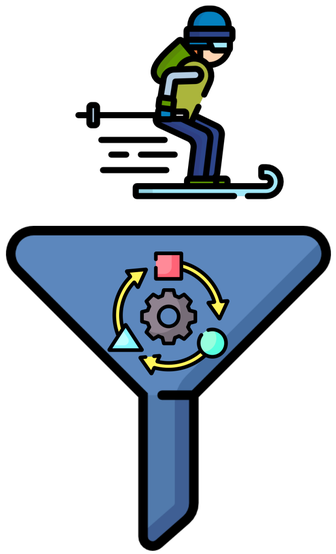


Online Decision

Adapt
Perform the extra I/O to adapt

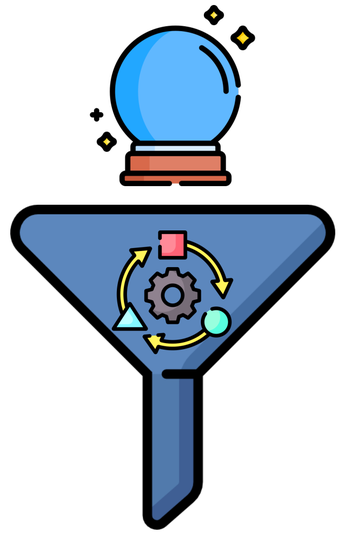
Don’t adapt
Skip the adapt and extra I/O cost

Online adaptivity: SkiQF and HybridSkiQF



SkiQF
Adaptivity as
ski rental problem

1.21x higher throughput over
AdaptiveQF across all workloads



HybridSkiQF
Periodically predicts if
adaptivity is required

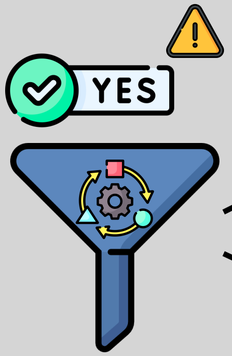
1.28x higher throughput over
AdaptiveQF on stable workloads

The ski-rental problem

Buy

Rent

How much more skiing?



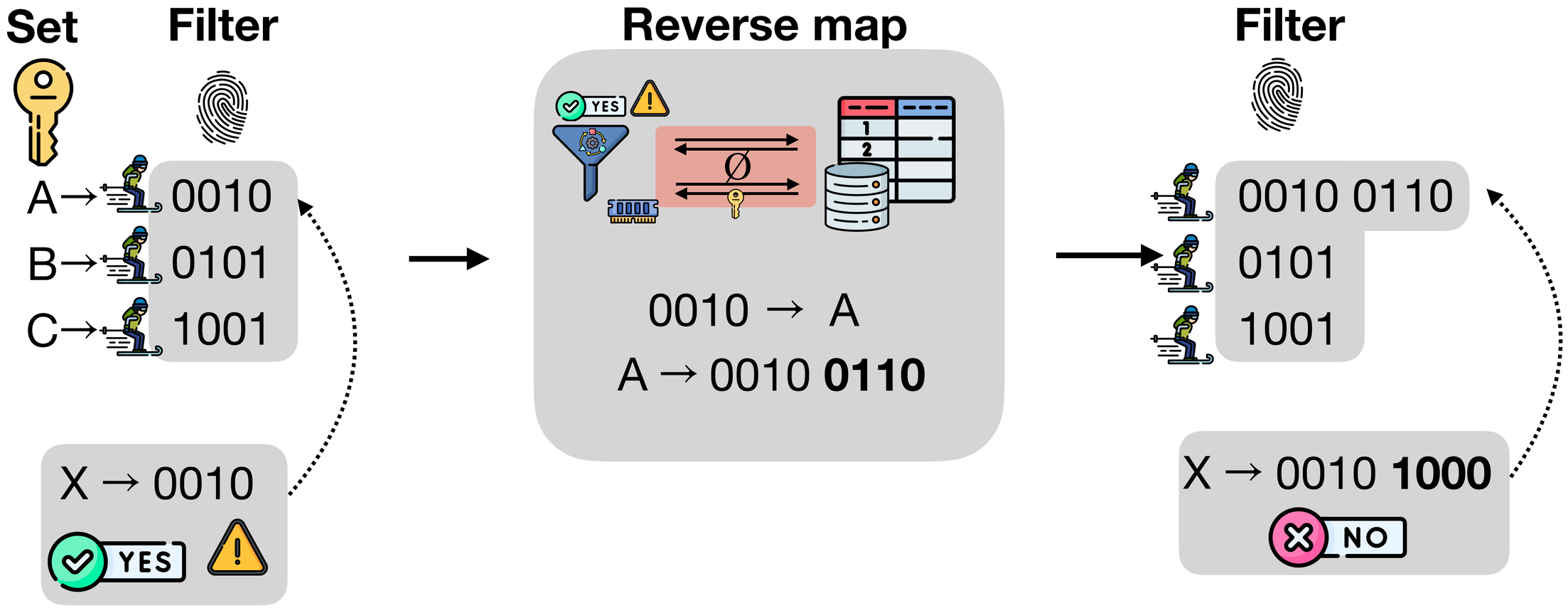
Adapt

Don’t adapt

How many more repetitions?

Models problems with choice between recurring cost or one-time cost under uncertainty

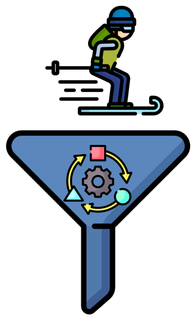
SkiQF: One ski-rental problem per fingerprint



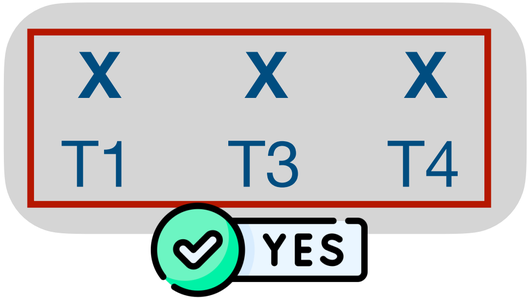
The break-even algorithm

Algorithm : Keep renting until cumulative rental cost exceeds buy cost

Only requires **<u>quantifying buy, rent costs</u>** and **<u>counting prior occurrences</u>**



X → 0010 (3)



→ **FP (Rent):** 1

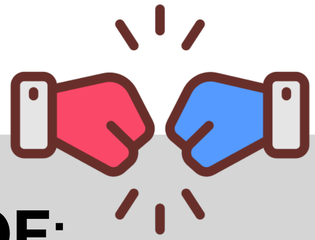
Adapt (Buy): $1+\alpha$

Cumulative Cost: 3

→ **Adapt if** $3 \geq 1+\alpha$

Else increment counter

Competitive analysis

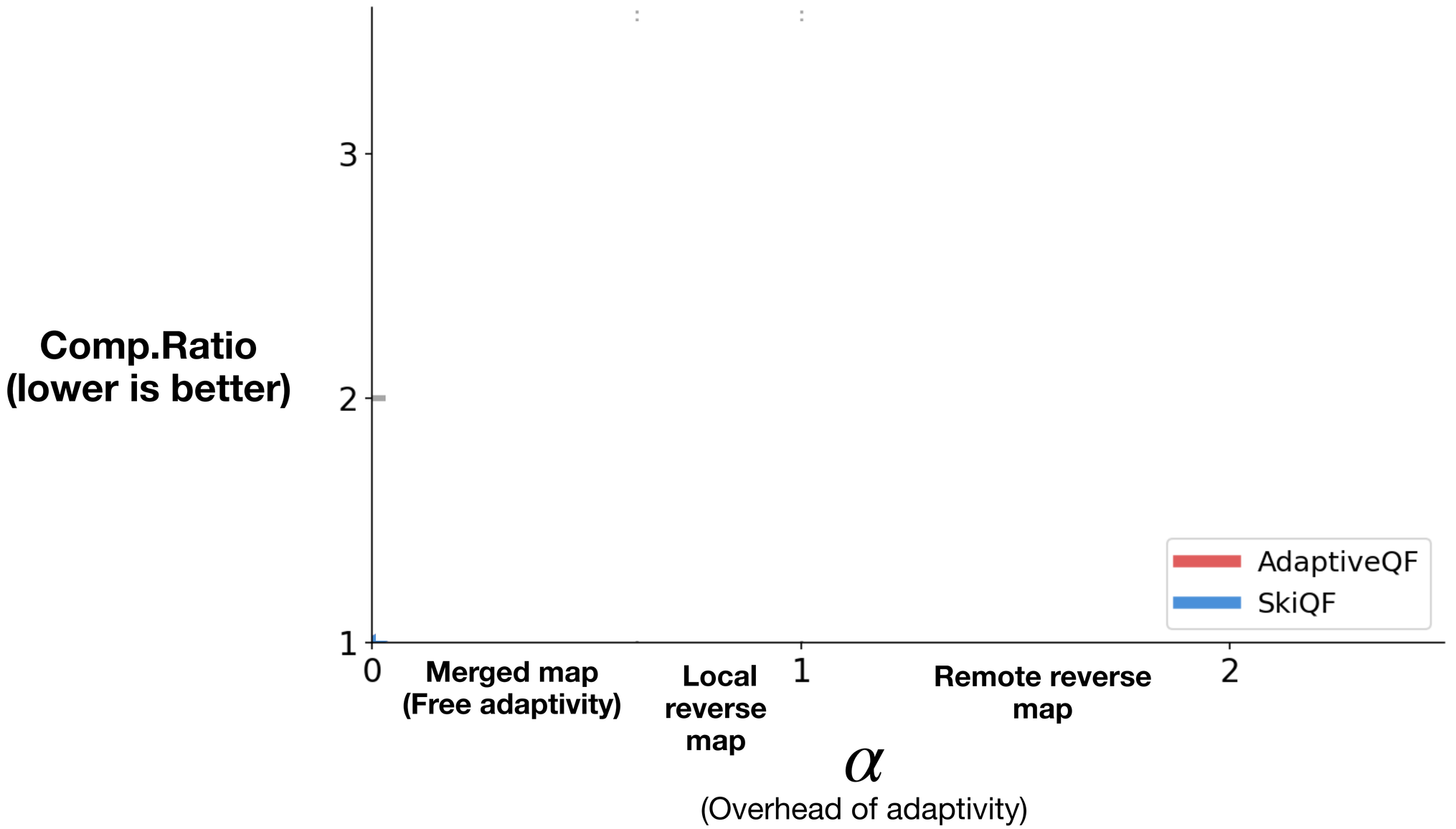


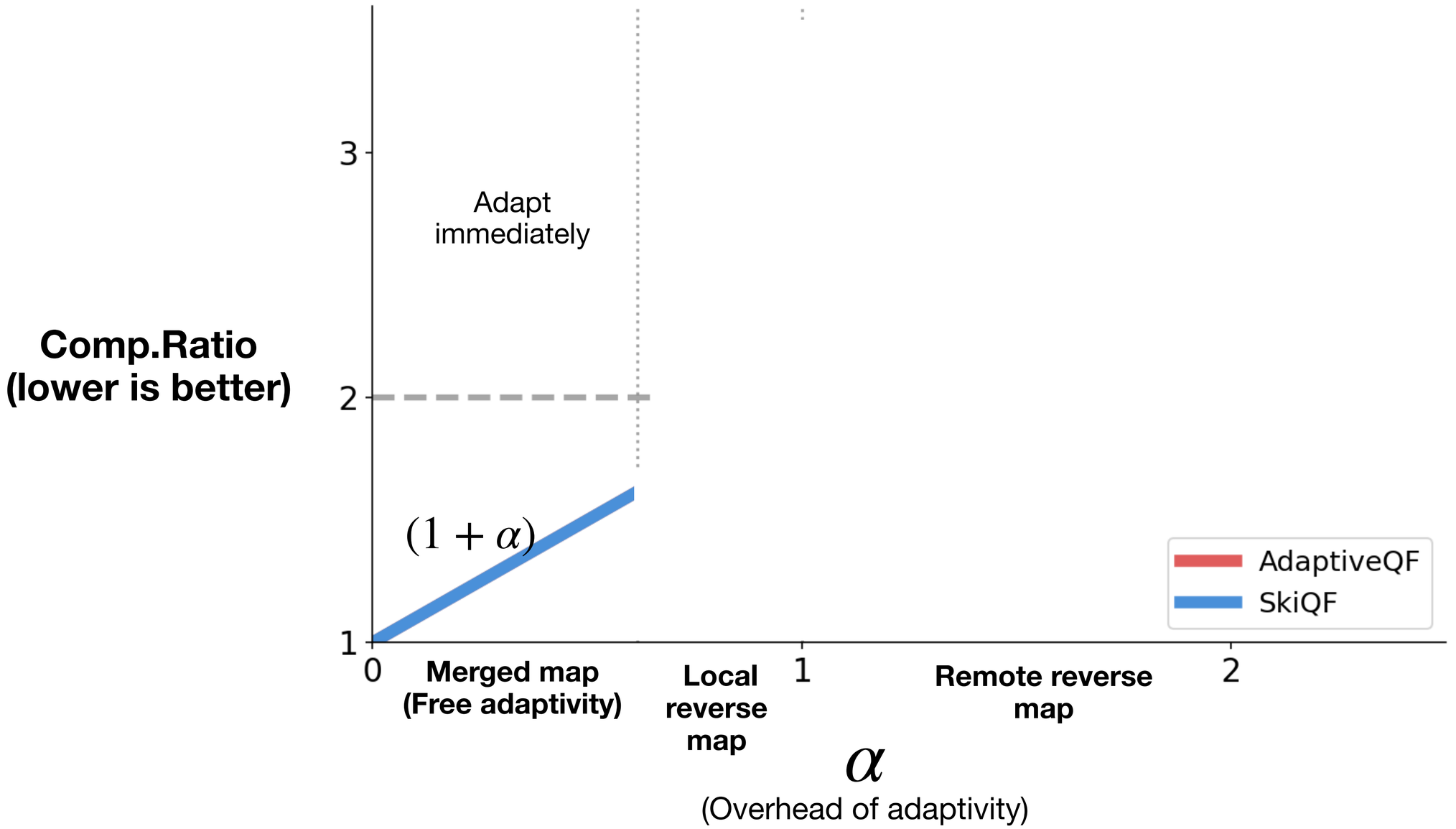
AdaptiveQF: Adapt immediately	**SkiQF**: Adapt using break-even algorithm

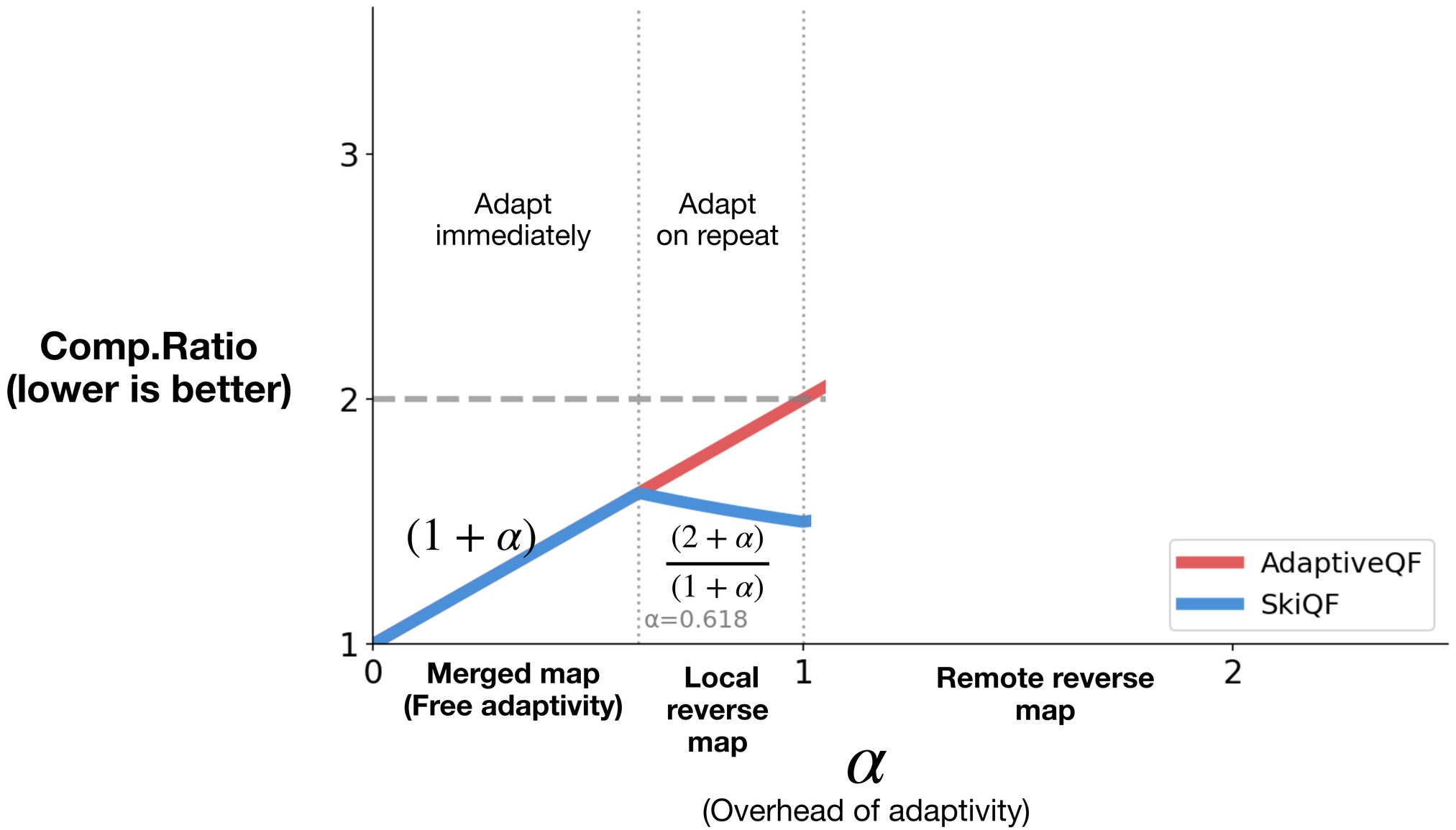
$$\textbf{Competitive ratio} = \frac{\text{Online cost}}{\text{Optimal cost}}$$

k**-competitive:** ratio $\leq k$ over all inputs

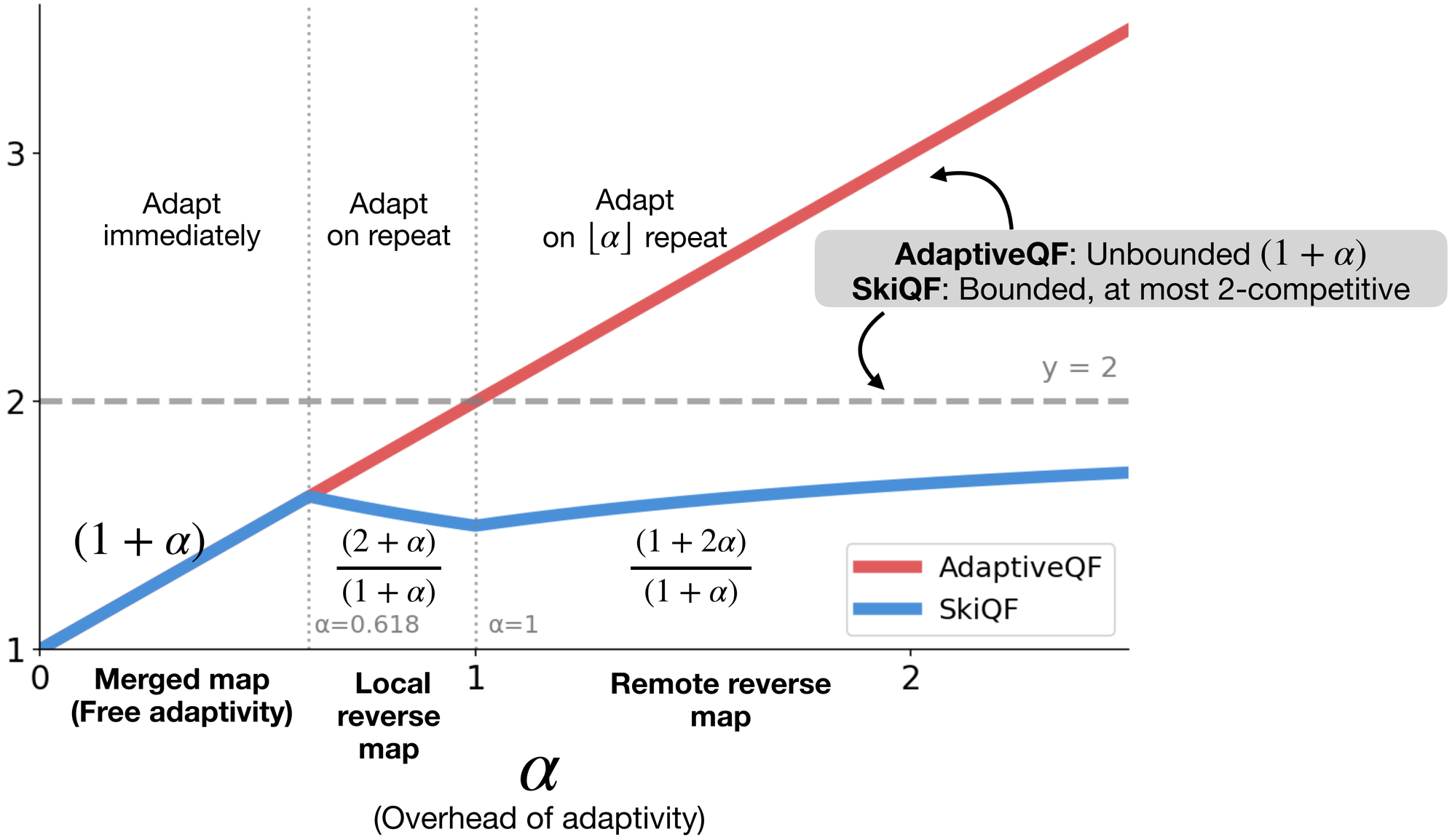
Lower competitive ratio is better, closer to optimal


Comp.Ratio
(lower is better)
3
2
1
0
1
2
AdaptiveQF
SkiQF
Merged map
(Free adaptivity)
Local
reverse
map
Remote reverse
map
α
(Overhead of adaptivity)


Comp.Ratio
(lower is better)
3
2
1
Adapt
immediately
$(1+\alpha)$
AdaptiveQF
SkiQF
0
Merged map
(Free adaptivity)
Local
reverse
map
1
Remote reverse
map
2
α
(Overhead of adaptivity)


Comp.Ratio
(lower is better)
3
2
1
0
Adapt
immediately
Adapt
on repeat
$(1+\alpha)$
$\frac{(2+\alpha)}{(1+\alpha)}$
α=0.618
AdaptiveQF
SkiQF
Merged map
(Free adaptivity)
Local
reverse
map
1
Remote reverse
map
2
α
(Overhead of adaptivity)

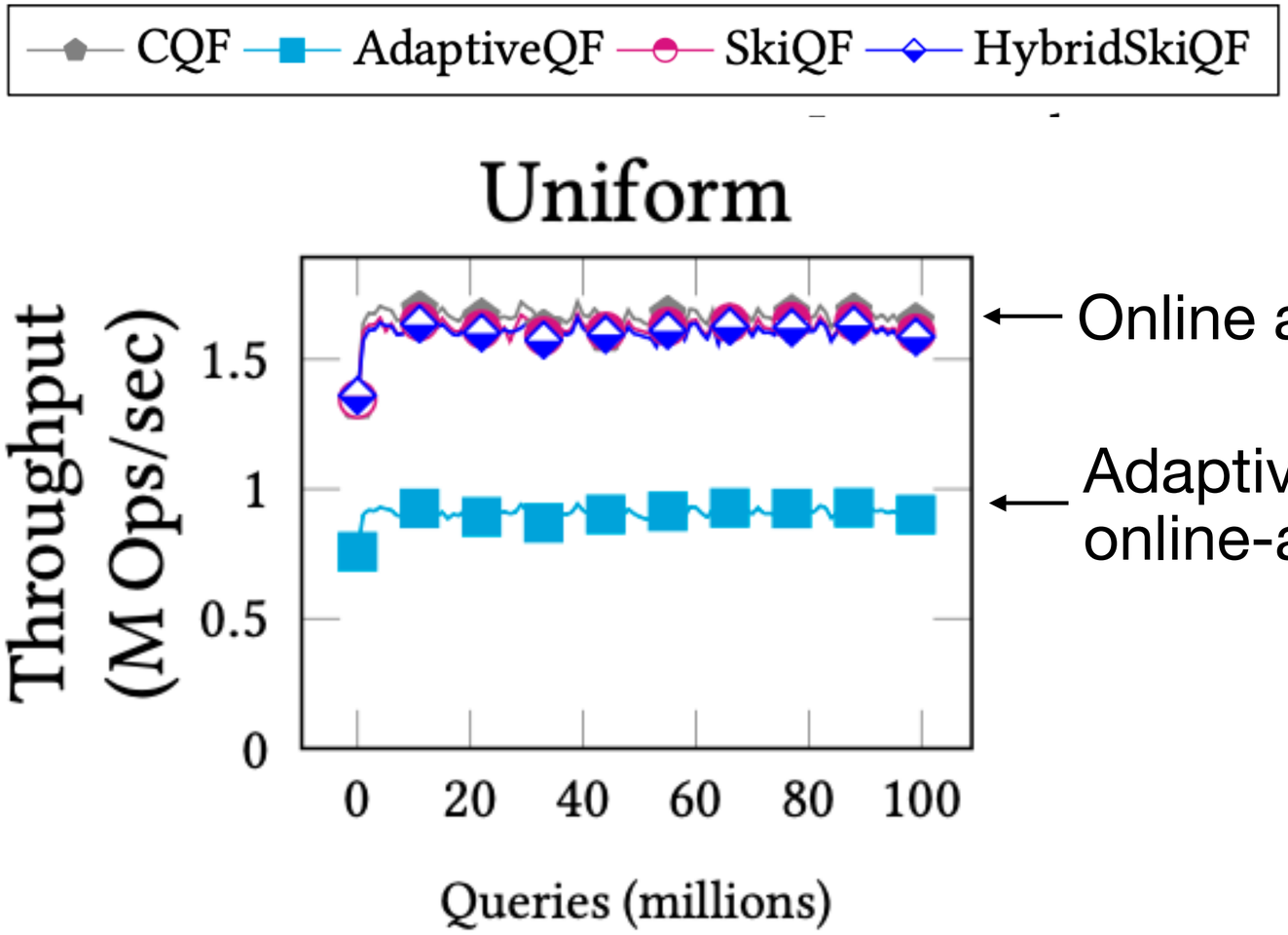
Comp.Ratio (lower is better)



Experimental results

- WiredTiger database (~134 million keys, 100 million queries)
- $\alpha = 1$ (Adapting costs twice, paper also tests varying α)
- Baselines:
  - **Non-Adaptive:** Counting quotient filter (CQF)
  - **Adaptive**: AdaptiveQF
  - **Online Adaptive**: SkiQF, HybridSkiQF

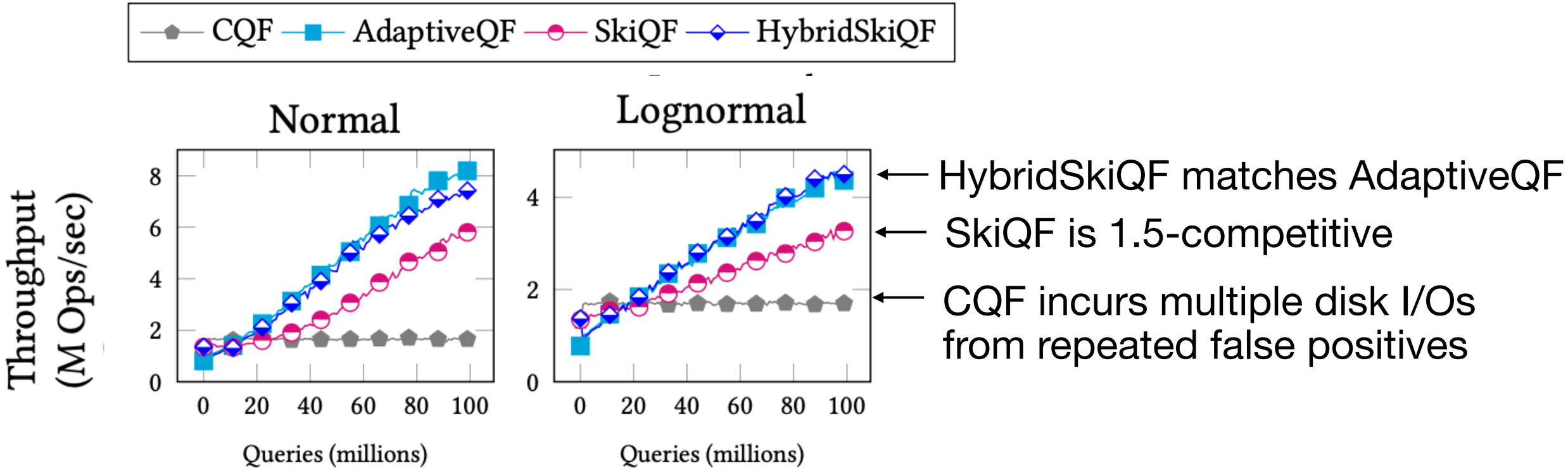
Uniform random workload



← Online adaptive filters match CQF

← AdaptiveQF has half the throughput of online-adaptive and non-adaptive filters

Skewed workloads

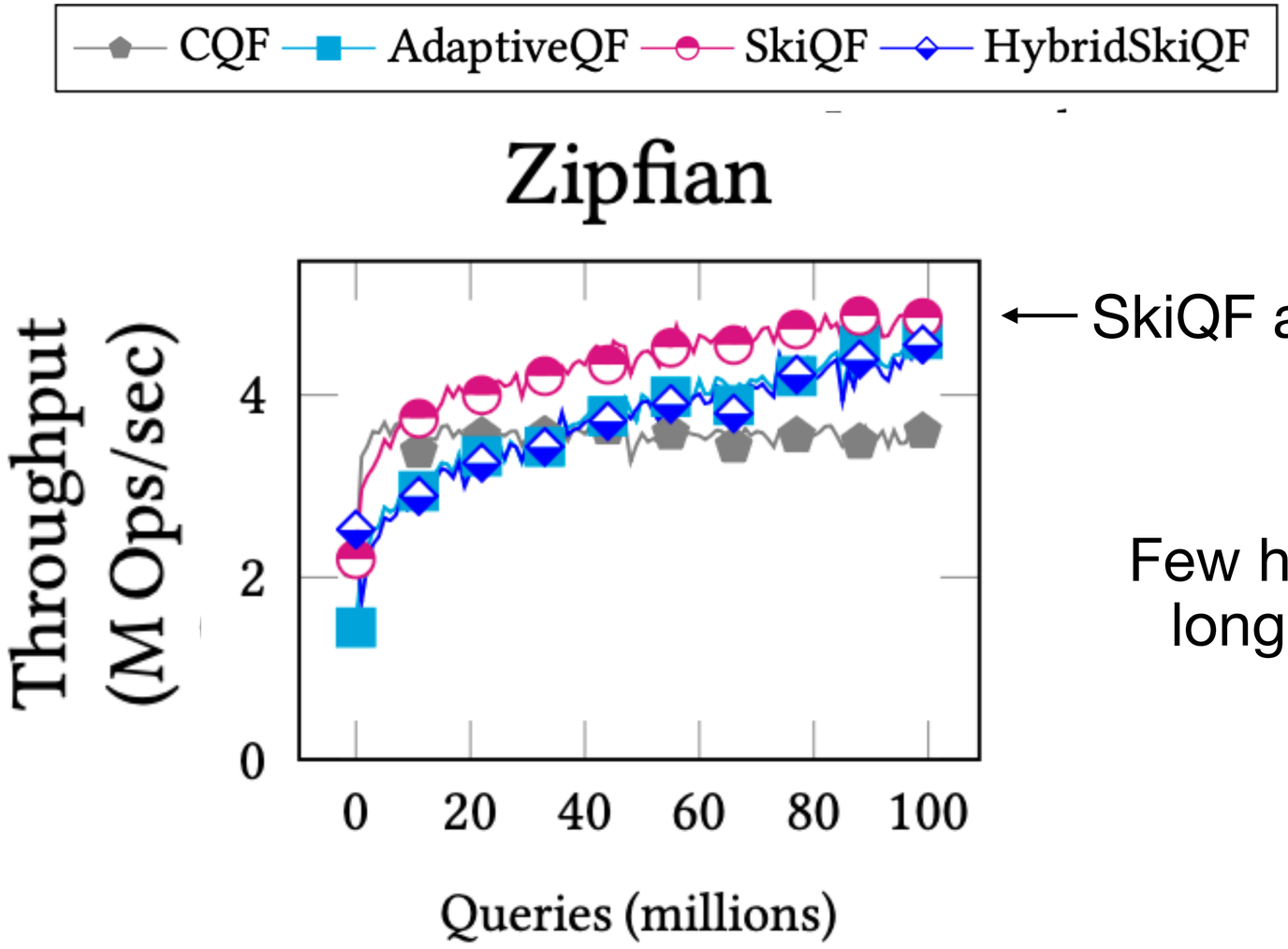


← HybridSkiQF matches AdaptiveQF

← SkiQF is 1.5-competitive

← CQF incurs multiple disk I/Os from repeated false positives

Zipfian workload (extreme skew)

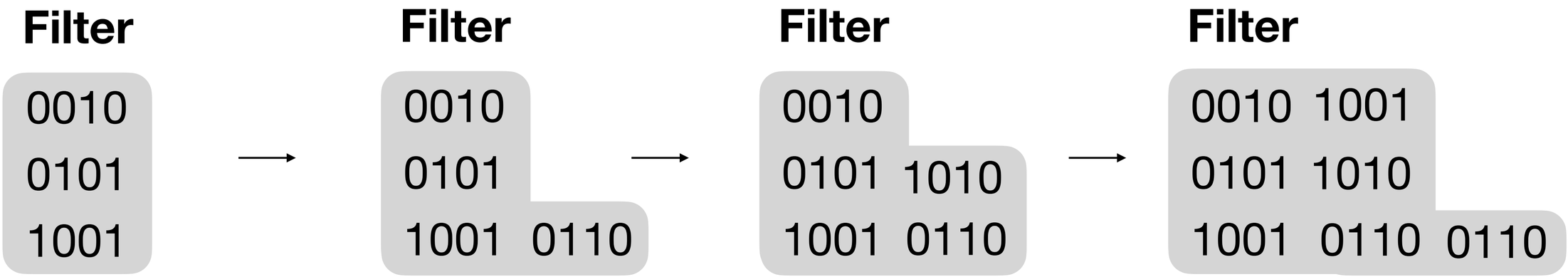


← SkiQF adapts only repeating false positives

Few highly repeating false positives,
long tail of one-off false positives

This proposal

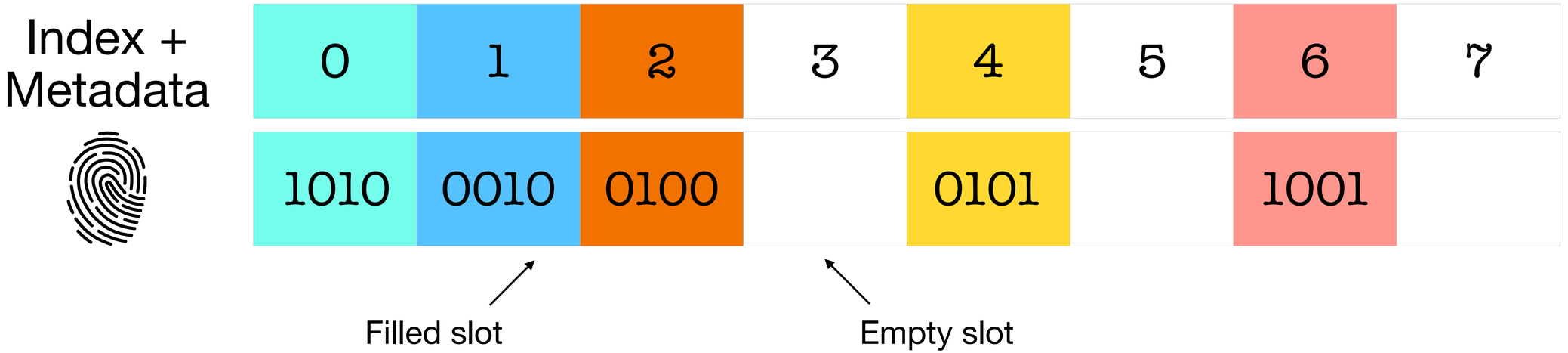
C2: Adaptive filters eventually run out of memory



Filter grows over time as adaptations accumulate

How to handle running out of space with minimal system disruptions?

Adaptive filter data layout

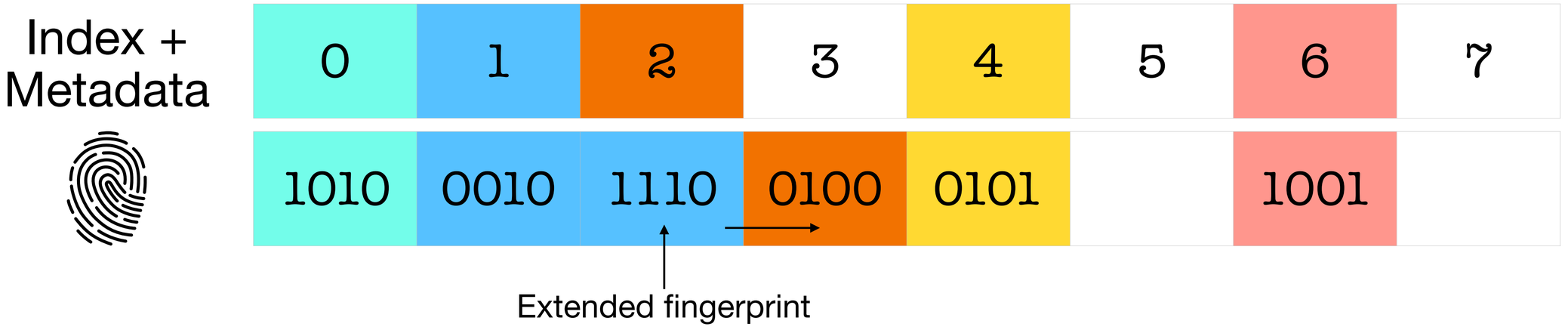


Adaptive filters are implemented as a linear probing hash table*

Allocated with fixed capacity, enables cache efficiency

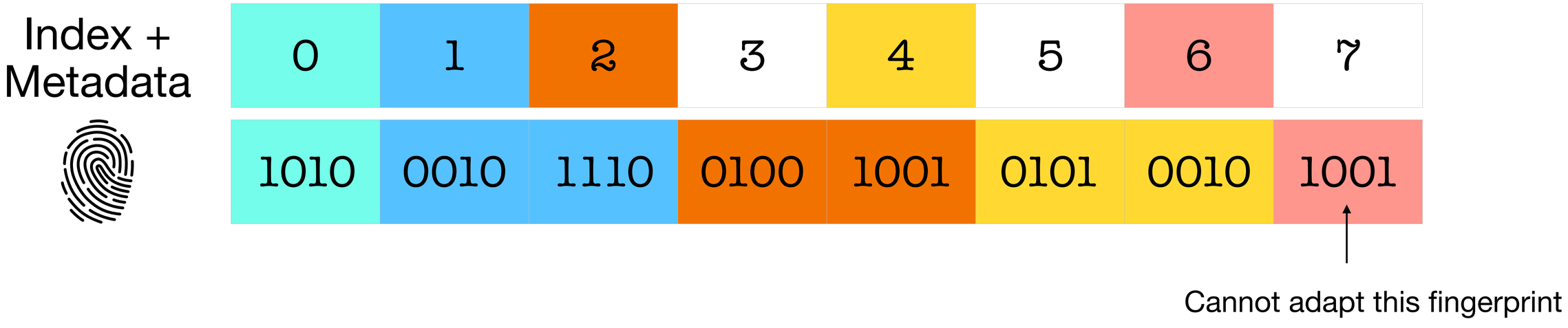
* Rank-and-select quotient filter

Adaptations consume slots in the table



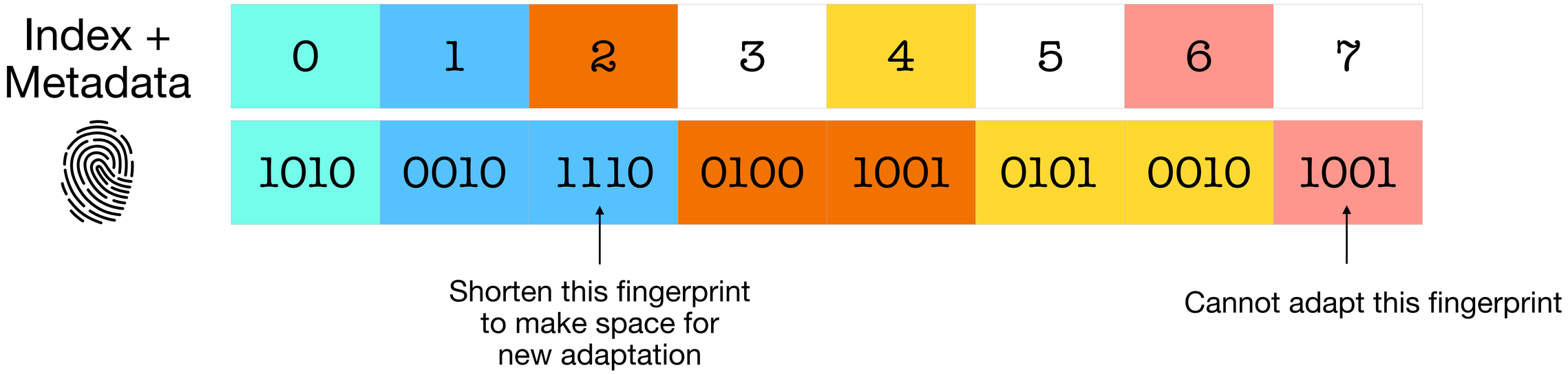
Each adaptations consume free slots in the filter

Eventually the filter will run out of slots

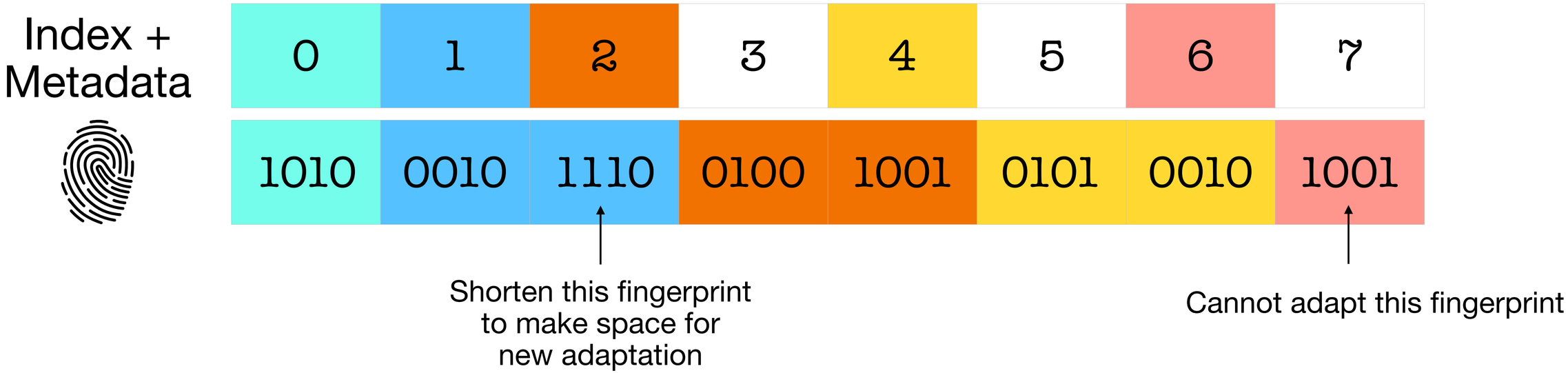


What happens when the filter runs out of slots?

What if we reclaim space from prior adaptations?

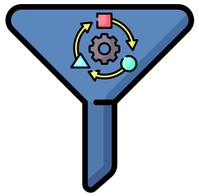


What if we reclaim space from prior adaptations?



No reclamation strategy can bound false positives!

Adversarial game



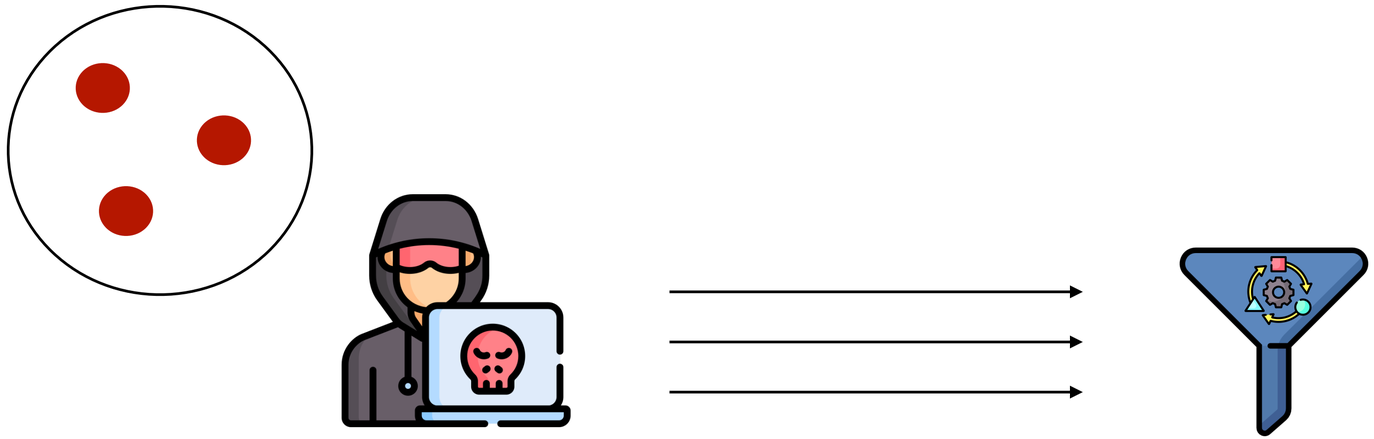
At any time, filter can remember at most "τ" adaptations

Issues n queries and records false positive queries

Can adaptive filter ensure false positive rate $\leq \epsilon$ for all n?

Adversary can play the long game

● : prior false positive

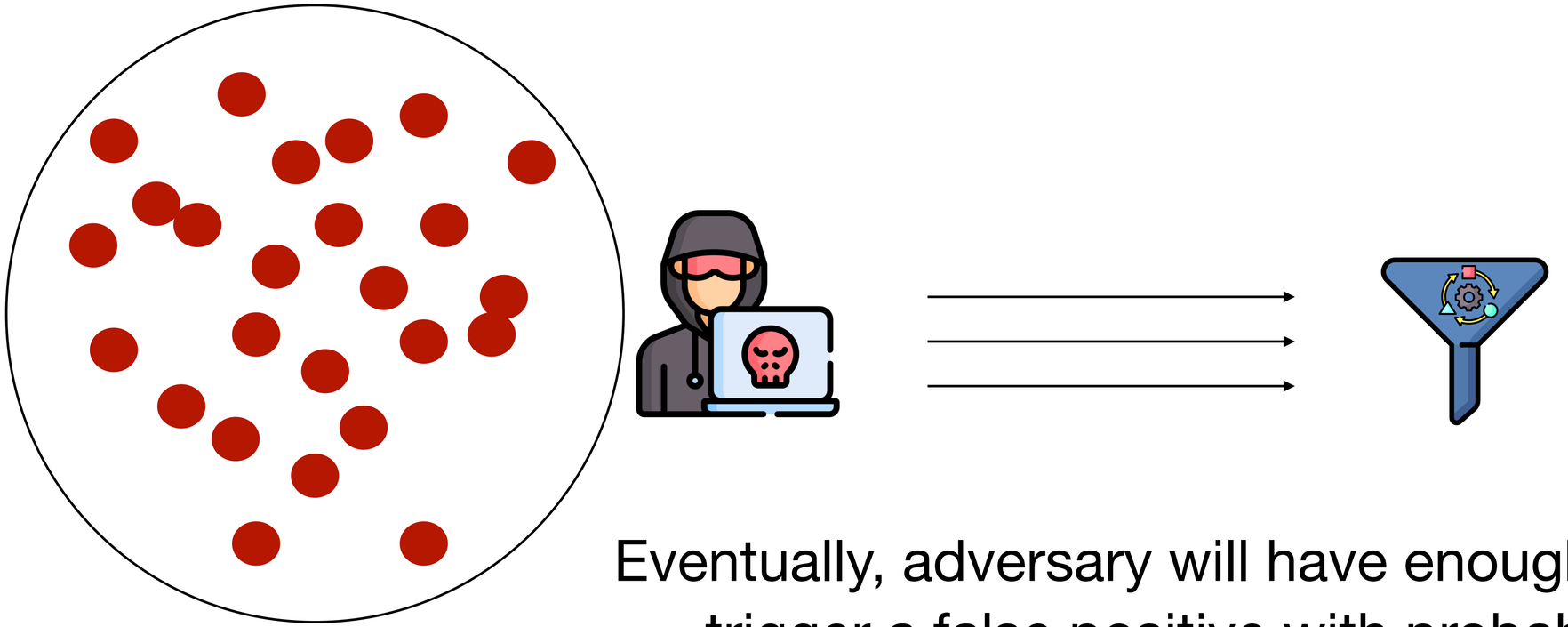


Adversary issues random queries and records which queries were false positives

Initially, adversary cannot reliably trigger false positives as filter has adapted

Adversary can play the long game

● : prior false positive



Eventually, adversary will have enough false positives to trigger a false positive with probability $(1 - \tau/k)$, where k is the number of collected false positives

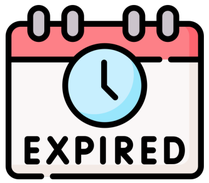
Current options: resize or rehash

Filter

0010 1001

0101 1010

1001 0110 0110



Allocate more space

Rehash all keys

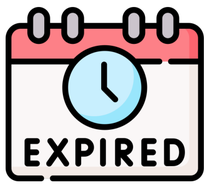
Current options: resize or rehash

Filter

0010 1001

0101 1010

1001 0110 0110

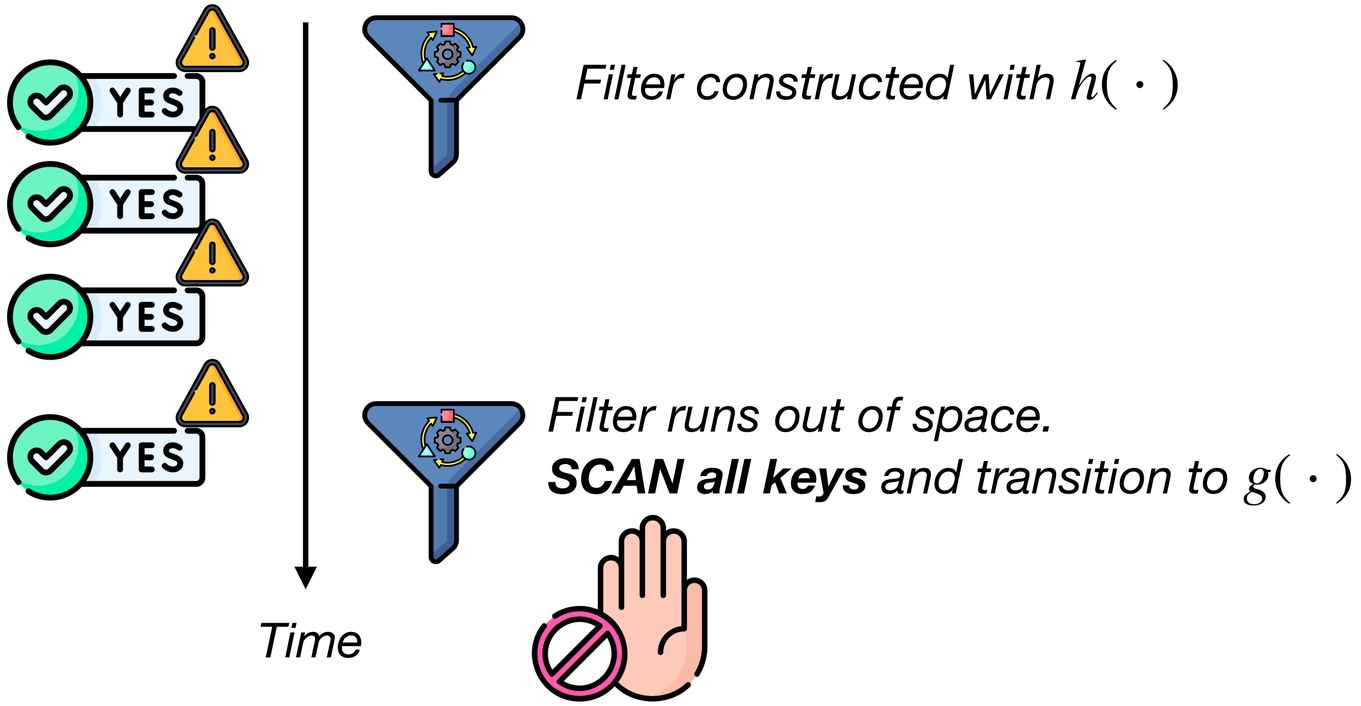


Allocate more space

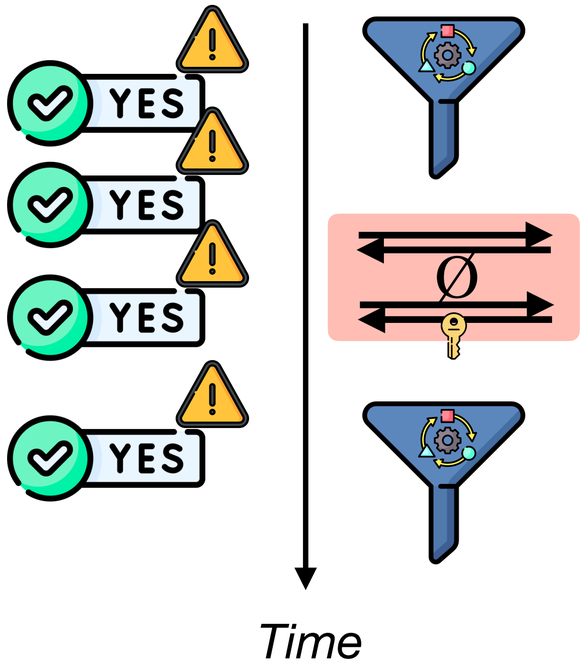
Growth is unbounded

Rehash all keys

Rehashing filter stalls system



Proposed work: Deamortized rehashing



Filter constructed with $h(\cdot)$

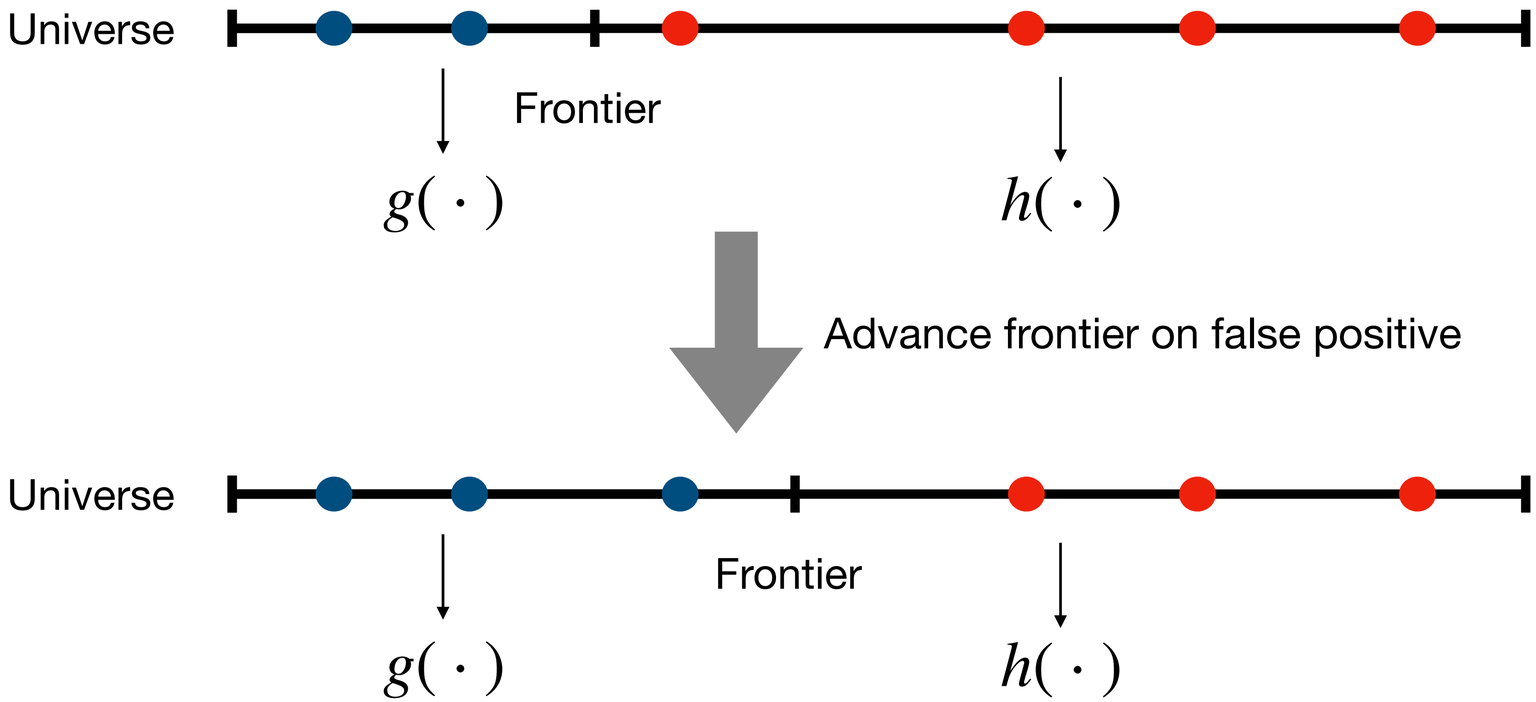
Piggyback on reverse map I/Os to transition to $g(\cdot)$

Filter completes transition to $g(\cdot)$

Bounded work per false positive
+
Preserve false positive rate guarantee

Idea 1: Frontier solution

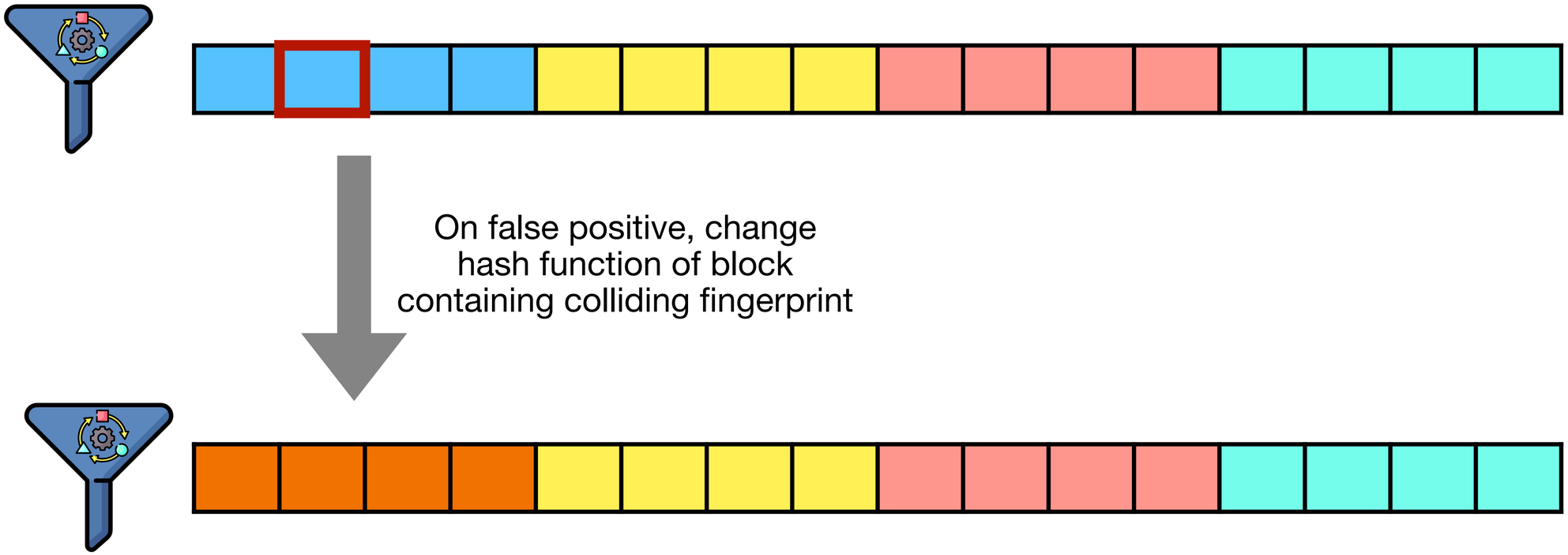
Frontier divides keys into hash function regimes



Proposed in BFG+'18, but not implemented in any adaptive filter
Drawback: Disk access pattern is random

Idea 2: Multiple hash functions divided across blocks

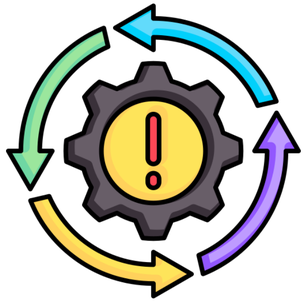
Blocks in filter use same hash function



On false positive, change hash function of block containing colliding fingerprint

I/Os are aligned with reverse map fetches

Plan of action



Design partial rehashing scheme

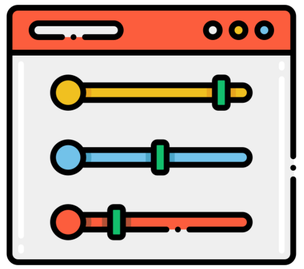
(4 weeks)

Two working ideas, based on AdaptiveQF and Telescoping filter

Formally prove performance and FPR bounds

(4 weeks)

Deamortized rebuilding bounds false positives to ϵn and query complexity



Experiments and analysis

(3 weeks)

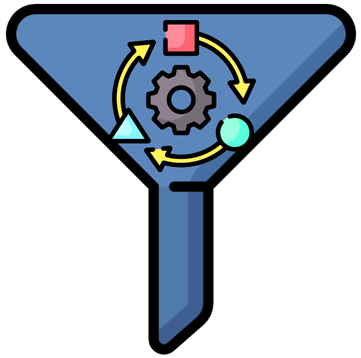
Experiments with real database shows no system disruptions (latency, throughput) on long running workloads

Conclusion

Acknowledgements

- **Funding**: Work funded in part by NSF grants OAC 2517201, 2513656, 2339521 and III 2311954.
- **Collaborators**: Benwei Shi, Jeff Phillips, Navid Eslami, Niv Dayan, Huanchen Zhang
- Data Lab @ Northeastern, UtahDB Lab

Adaptive Filters



Current: Robustness at the cost of system performance

Thesis proposal: Address system performance issues and establish adaptive filters as practical

Impact: Adaptivity is a de-facto standard feature in filters, resulting in robust and resilient systems

Key Takeaways

- **Thesis plan proposes path to making adaptive filters the default filter in data systems**
- Reliable data structures are needed now more than ever
  - **Performance + Space efficiency + Robustness**
- Links to paper, code can be found at ykchesetti.com