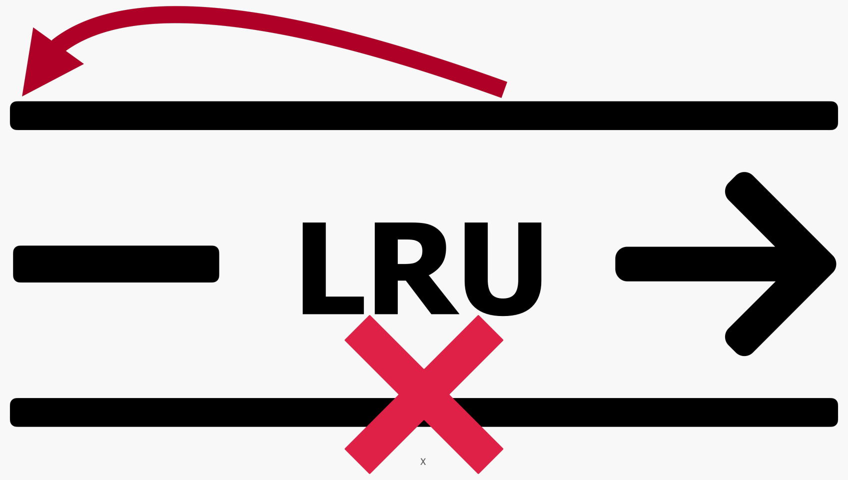


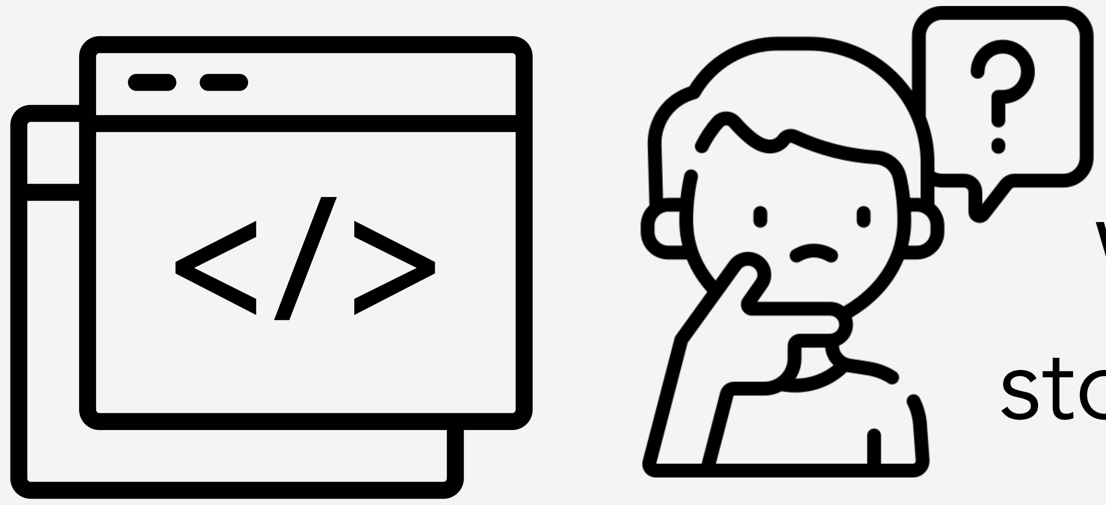
No more LRU, Simple Scalable Caching with only FIFO Queues

Juncheng Yang
Harvard University

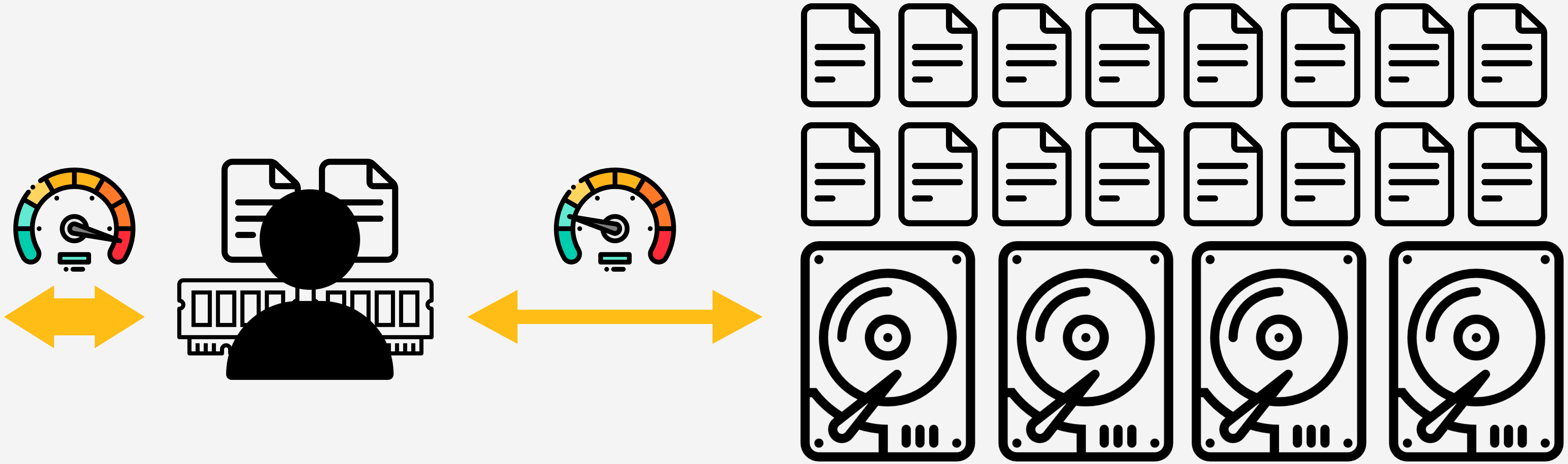
Digital society

Software cache

- Cache: A ***fast but small*** storage storing a portion of the dataset to speed up data access
- Software cache: all decisions made in software



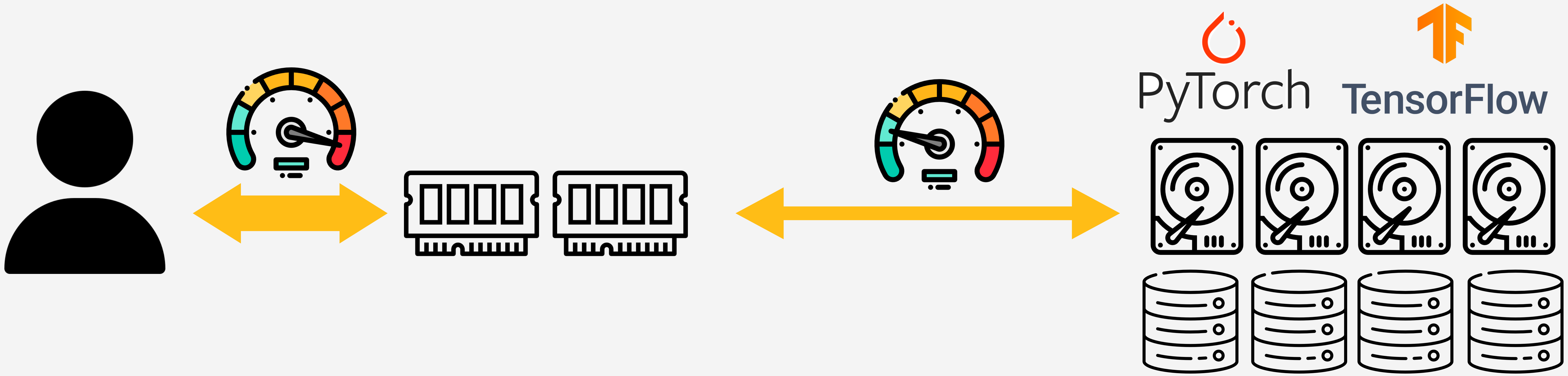
What should be stored in the cache?

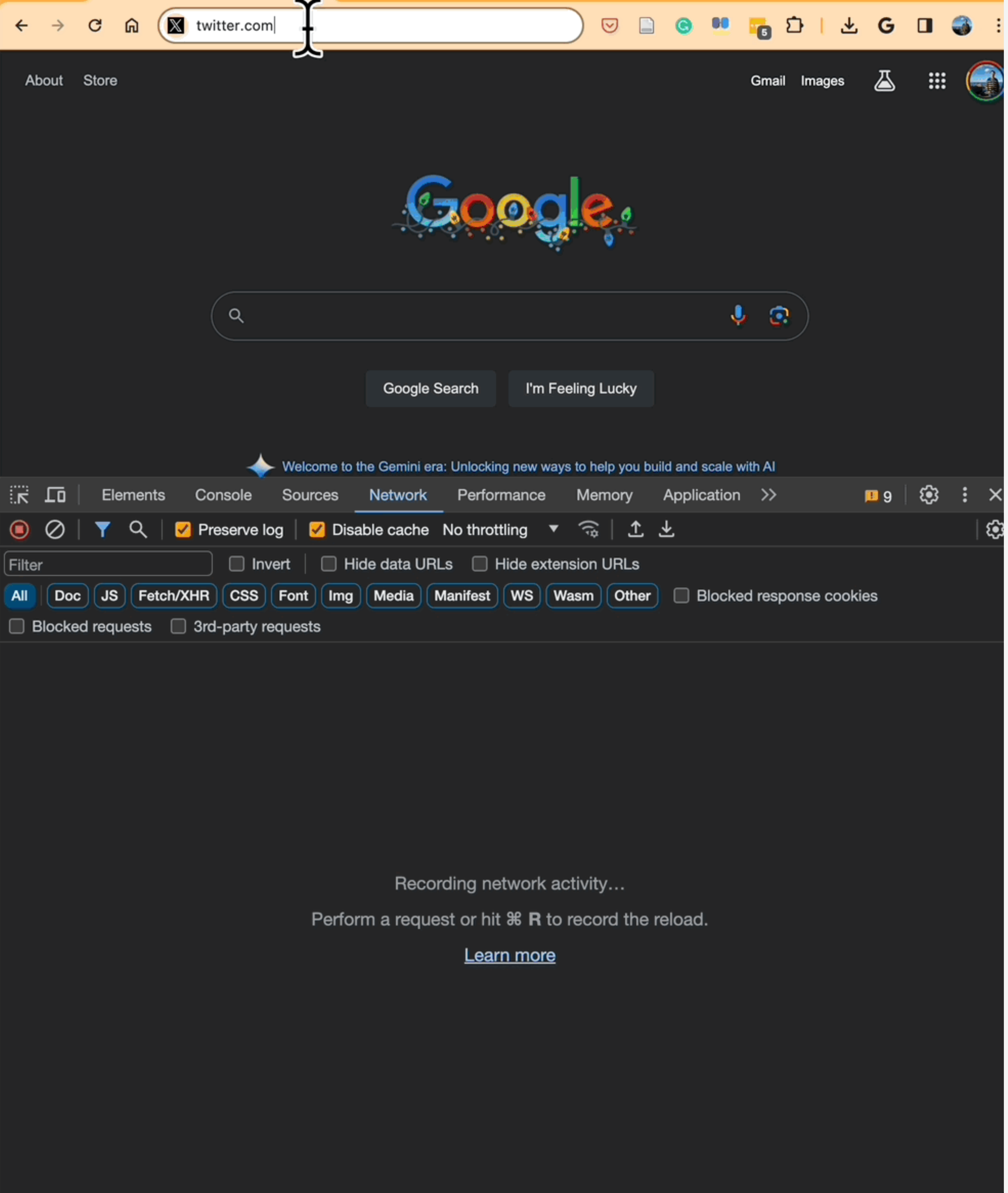


Software cache is deployed everywhere

Ubiquitous deployment

- speed up data access
- reduce data movement
- reduce repeated computation





How many requests are served by caches?

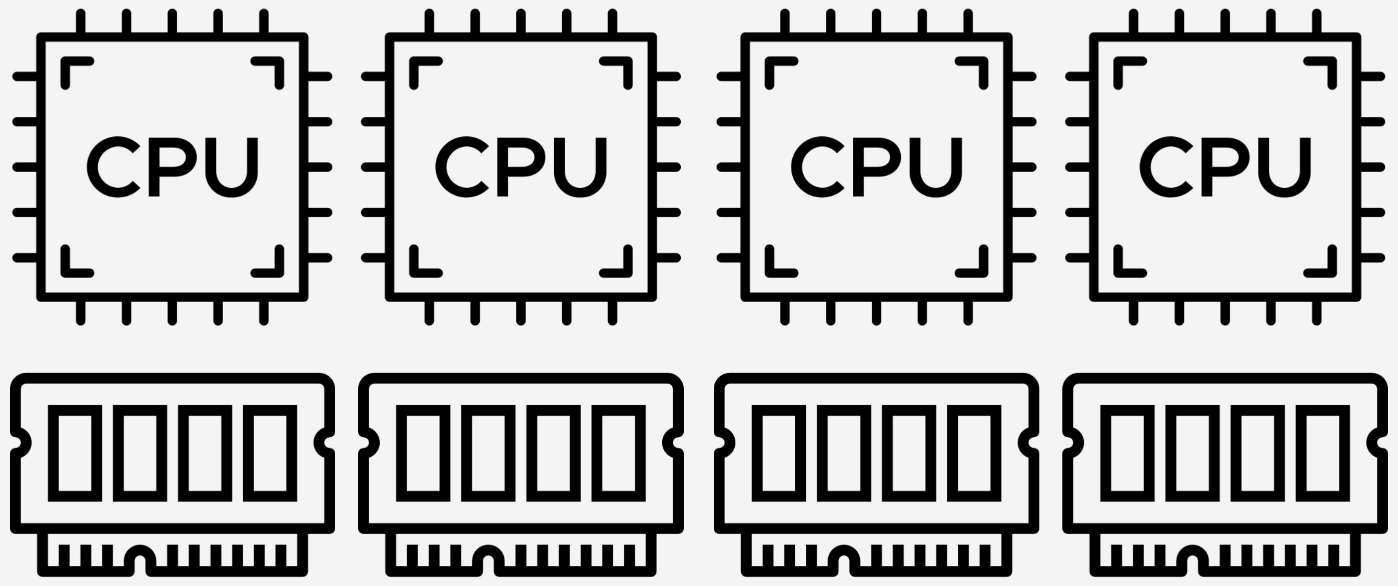
1000s

A typical web application

A typical web application

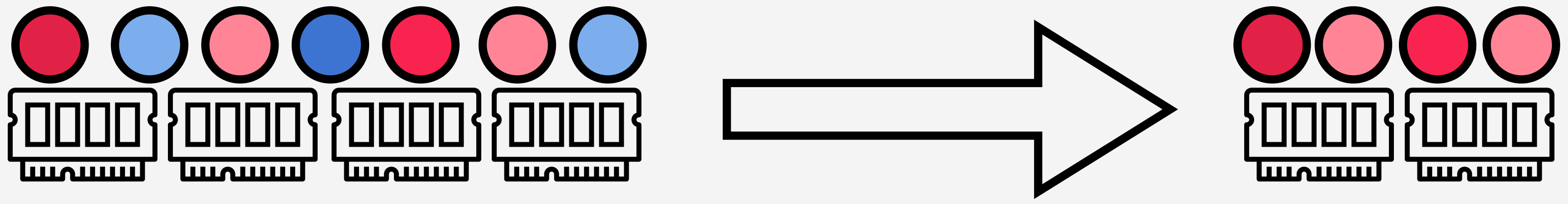
Software cache consumes a huge amount of resources

- Google CliqueMap: PBs of DRAM[1]
- Twitter: 100s clusters, 100s TB of DRAM, 100,000s cores[2]
- Meta, Pinterest…



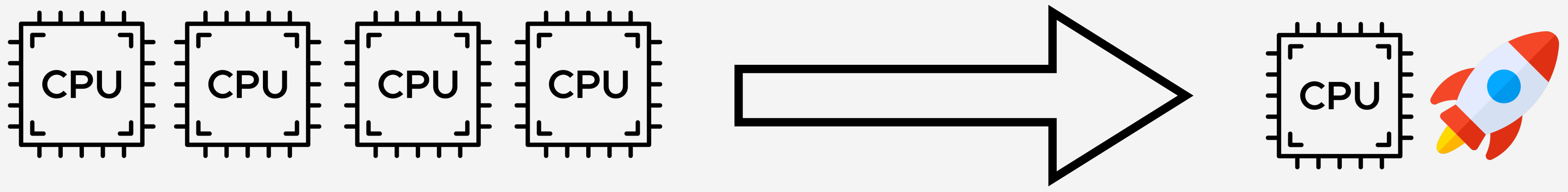
How to make caching more **sustainable**?

- Less DRAM: caching more useful objects



efficient caching

- Fewer CPU cores: faster and concurrent cache hits



fast and scalable caching

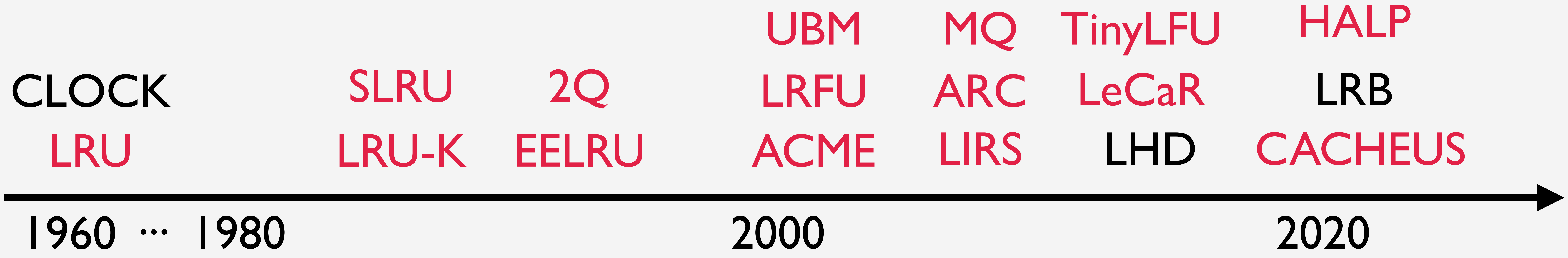
[1] CliqueMap: Productionizing an RMA-Based Distributed Caching System, SIGCOMM'21
[2] A large scale analysis of hundreds of in-memory cache clusters at Twitter, OSDI'20

The core of a cache

eviction algorithm

The need for simple and scalable cache eviction algorithm

- 60+ years of research on designing eviction algorithms: most are LRU-based
    - not scalable
    - increasingly complex



"Predicting which pages will be accessed in the near future is tricky, and the kernel has evolved a number of mechanisms to improve its chances of guessing right. But **the kernel not only often gets it wrong, it also spend a lot of CPU time to make the incorrect decision.**"

— kernel developer

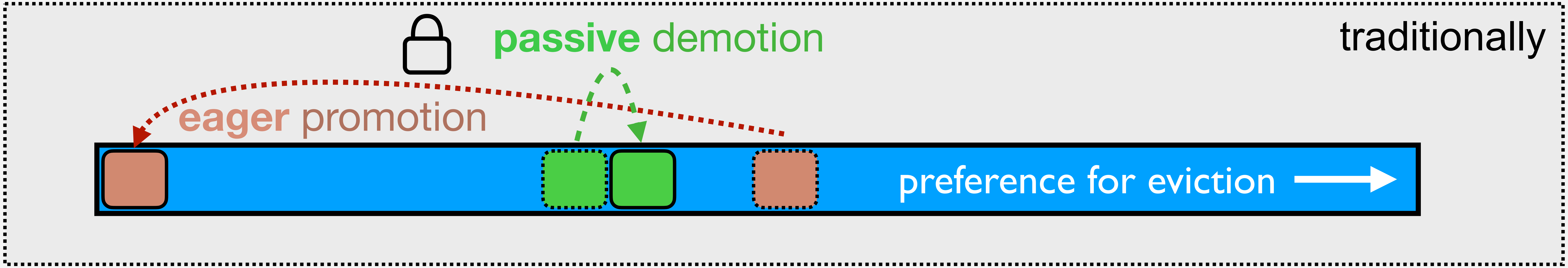
A simple algorithm: FIFO eviction algorithm

- First-in-first-out (FIFO)
  - simpler than LRU
  - fewer metadata
  - no computation
  - more scalable
  - flash-friendly

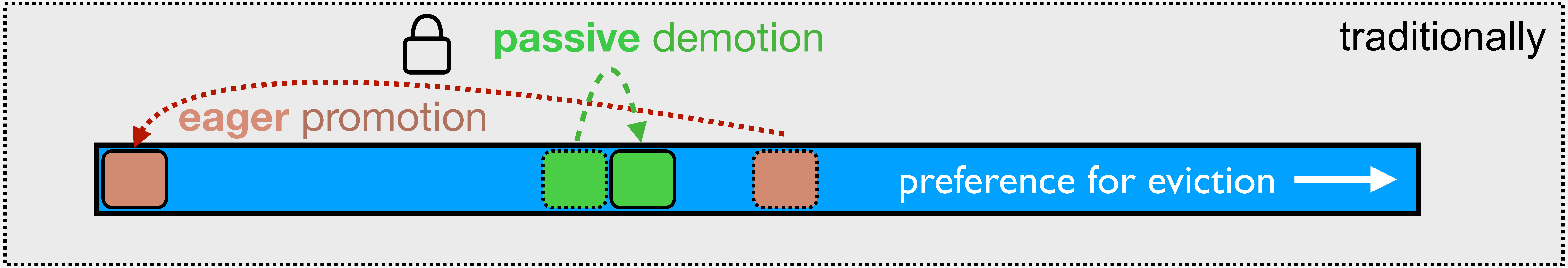
The only drawback:
FIFO has a high miss ratio

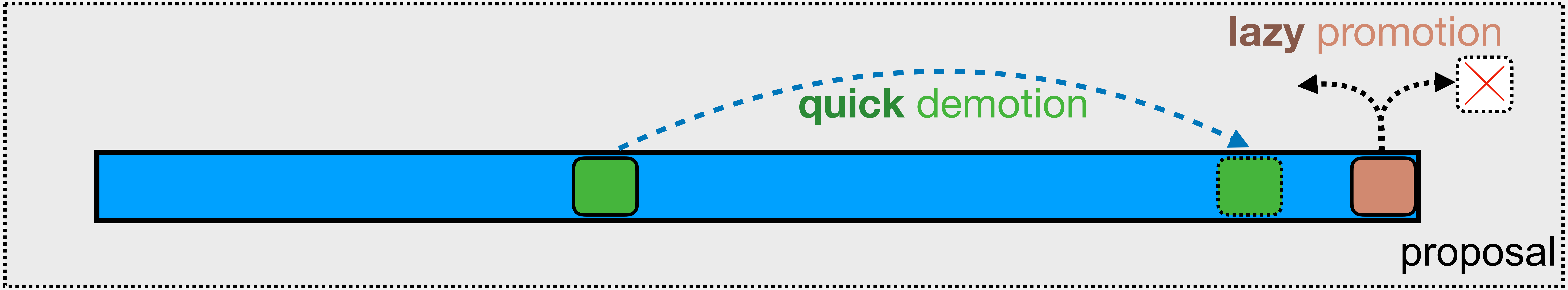
LAZY PROMOTION  QUICK DEMOTION 

Existing eviction algorithms focus on promotion



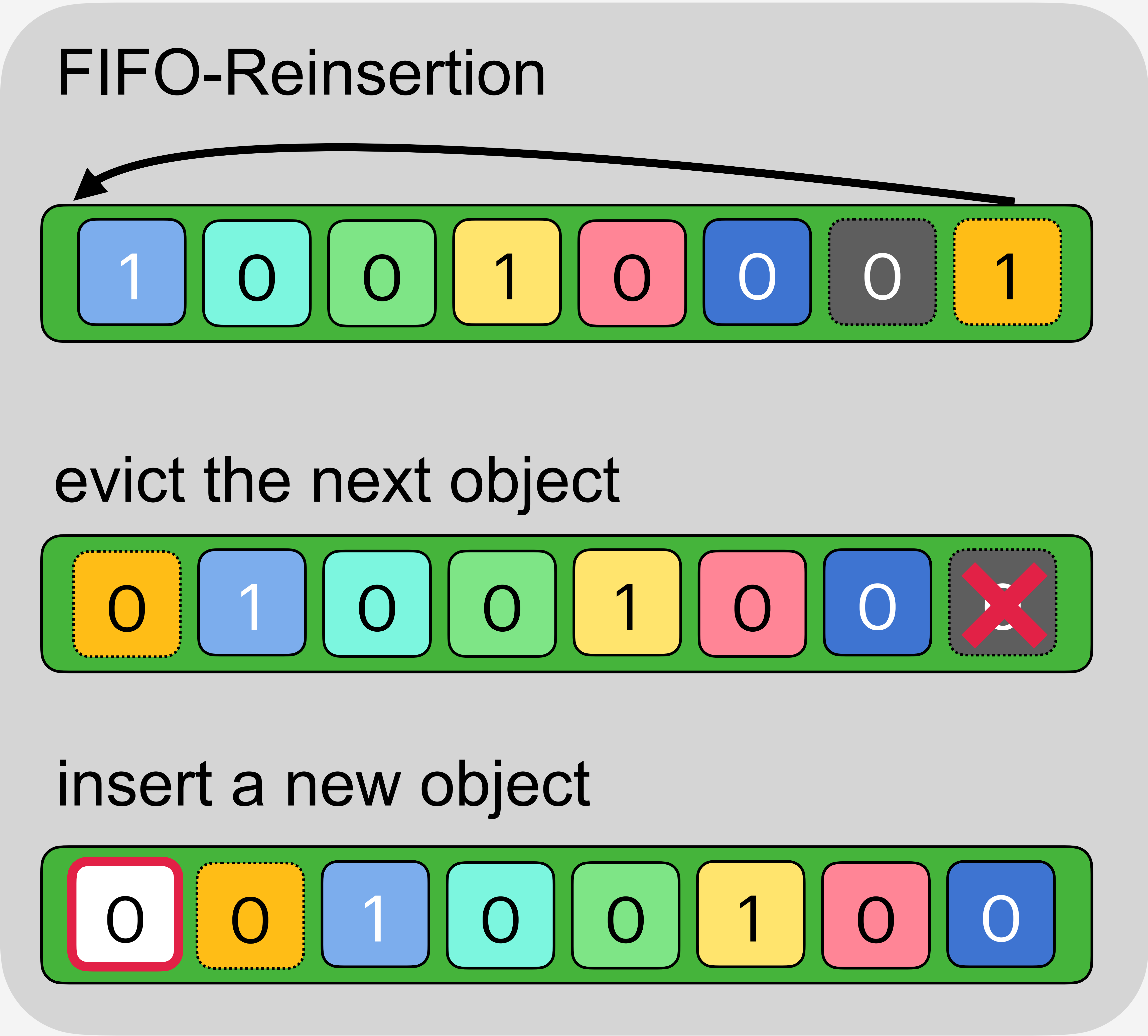
Demotion should be the first-class citizen





Lazy promotion: only promote during eviction

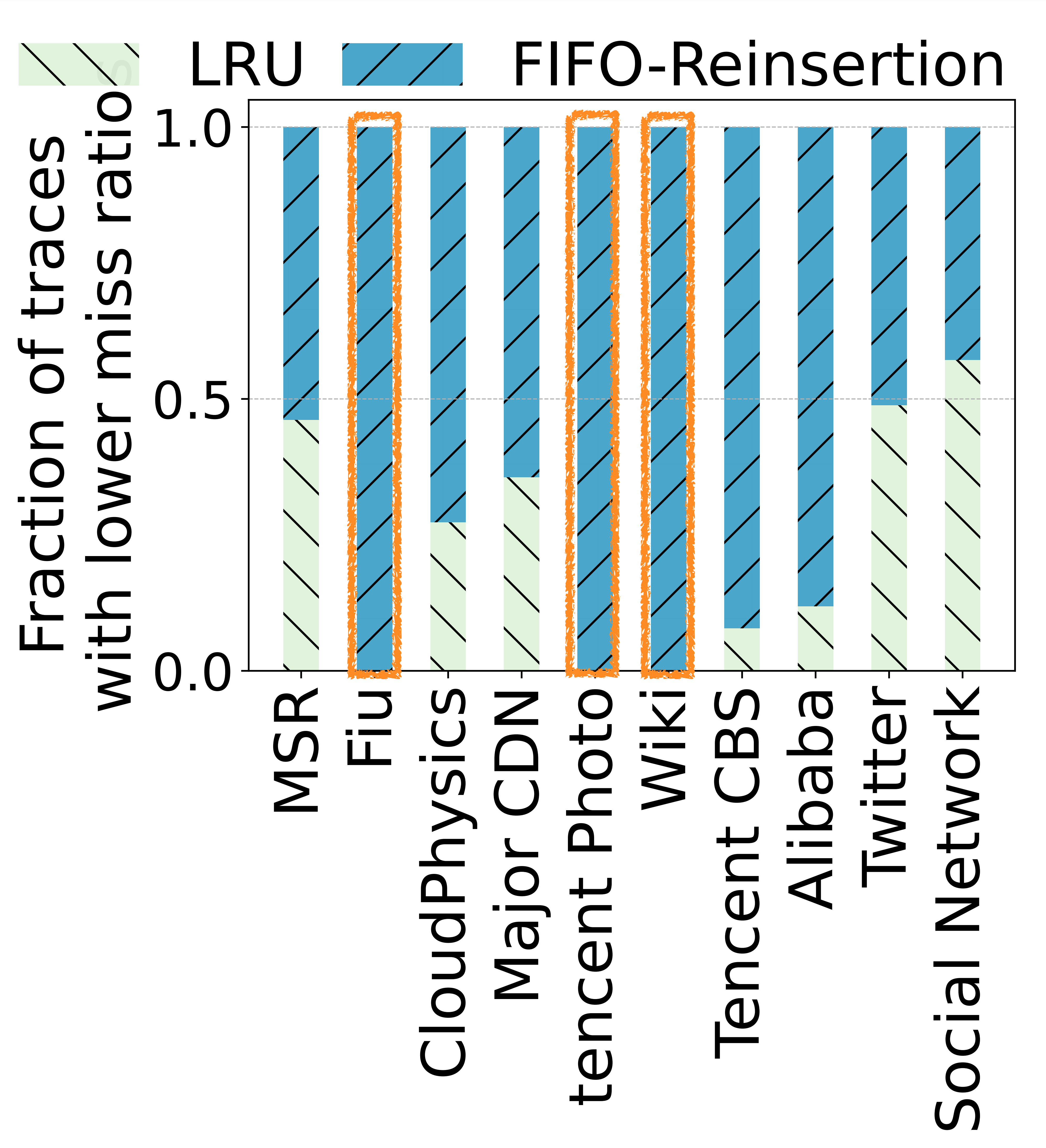
0: not accessed, 1: has accessed



Benefits of lazy promotion

- less computation
- better decision
- more efficient (lower miss ratio)

Lazy promotion: only promote during eviction



Why is FIFO-Reinsertion more efficient?

evict new objects faster!

Objects push N towards eviction	
LRU	A X
FIFO-Reinsertion	A X B

Quick demotion: quickly evict new objects

- One-hit wonders: objects appeared once in the sequence
- Cache workloads: *shorter request sequences have larger one-hit-wonder ratios*

time	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
request	A	B	A	C	B	A	D	A	B	C	B	A	E	C	A	B	D

start time	end time	sequence length (# objects)	# one-hit wonder	one-hit wonder ratio
1	17	5	1 (E)	20%
1	7	4	2 (C, D)	50%

Quick demotion: quickly evict new objects

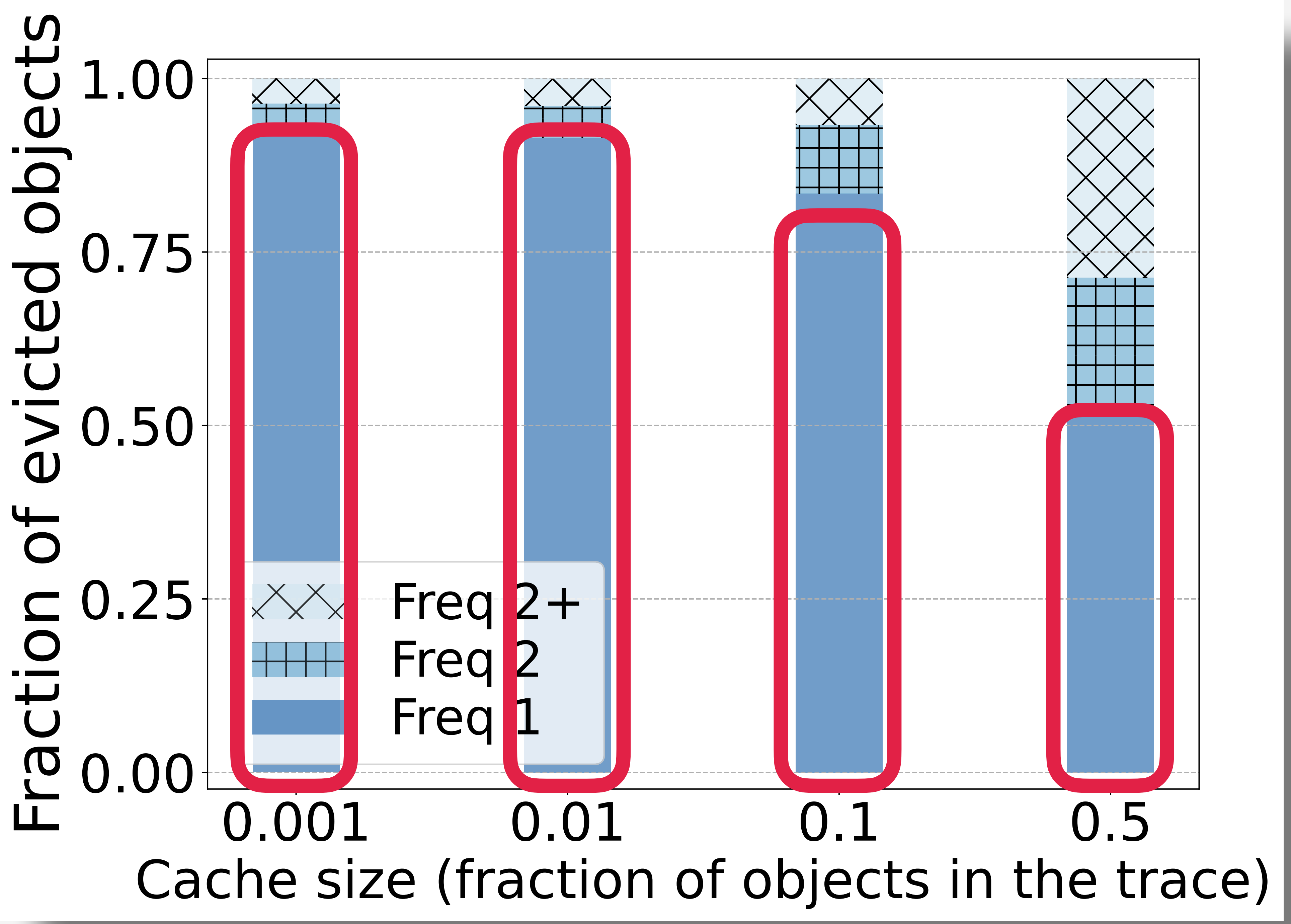
- One-hit wonders: objects appeared once in the sequence
- Cache workloads: *shorter request sequences have larger one-hit-wonder ratios*

One-hit-wonder ratio of week-long traces at 10% length:
72% (mean on **6594** traces)

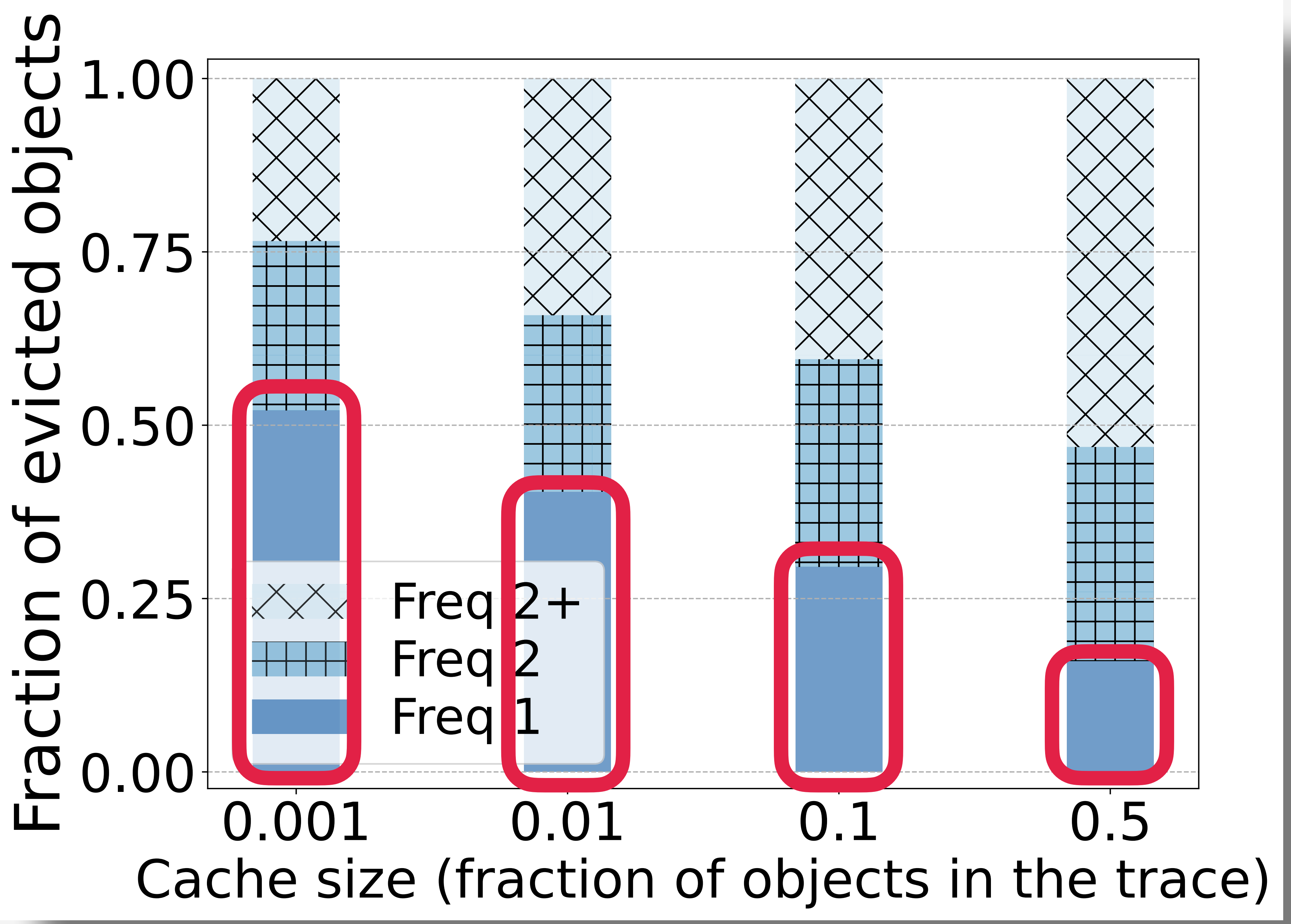
Implication: most objects are not used before evicted

Observation

Most objects are not reused before evicted

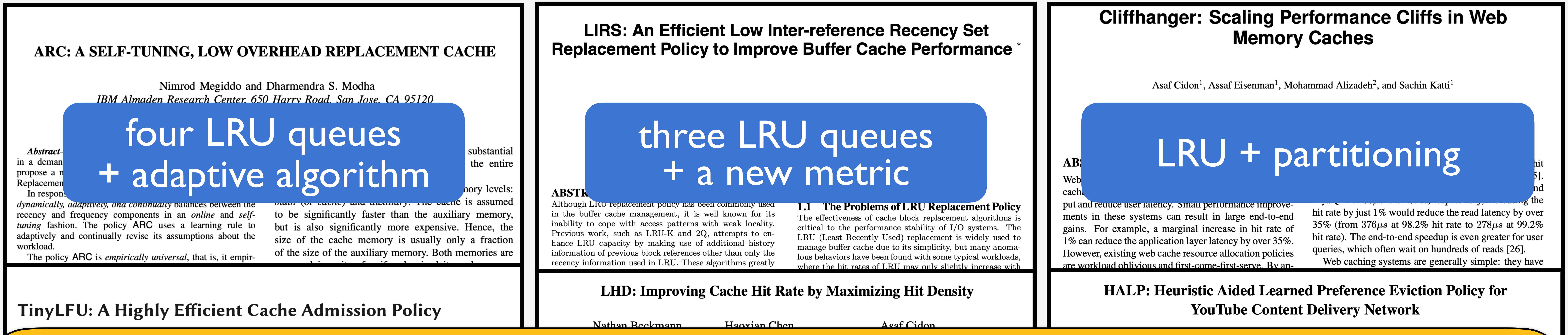


LRU cache running MSR workload



LRU cache running Twitter workload

Quick demotion: quickly evict new objects



The secret sauce of state-of-the-art algorithms:

evicting new objects very aggressively

S3-FIFO Design

Simple, Scalable caching with three Static FIFO queues

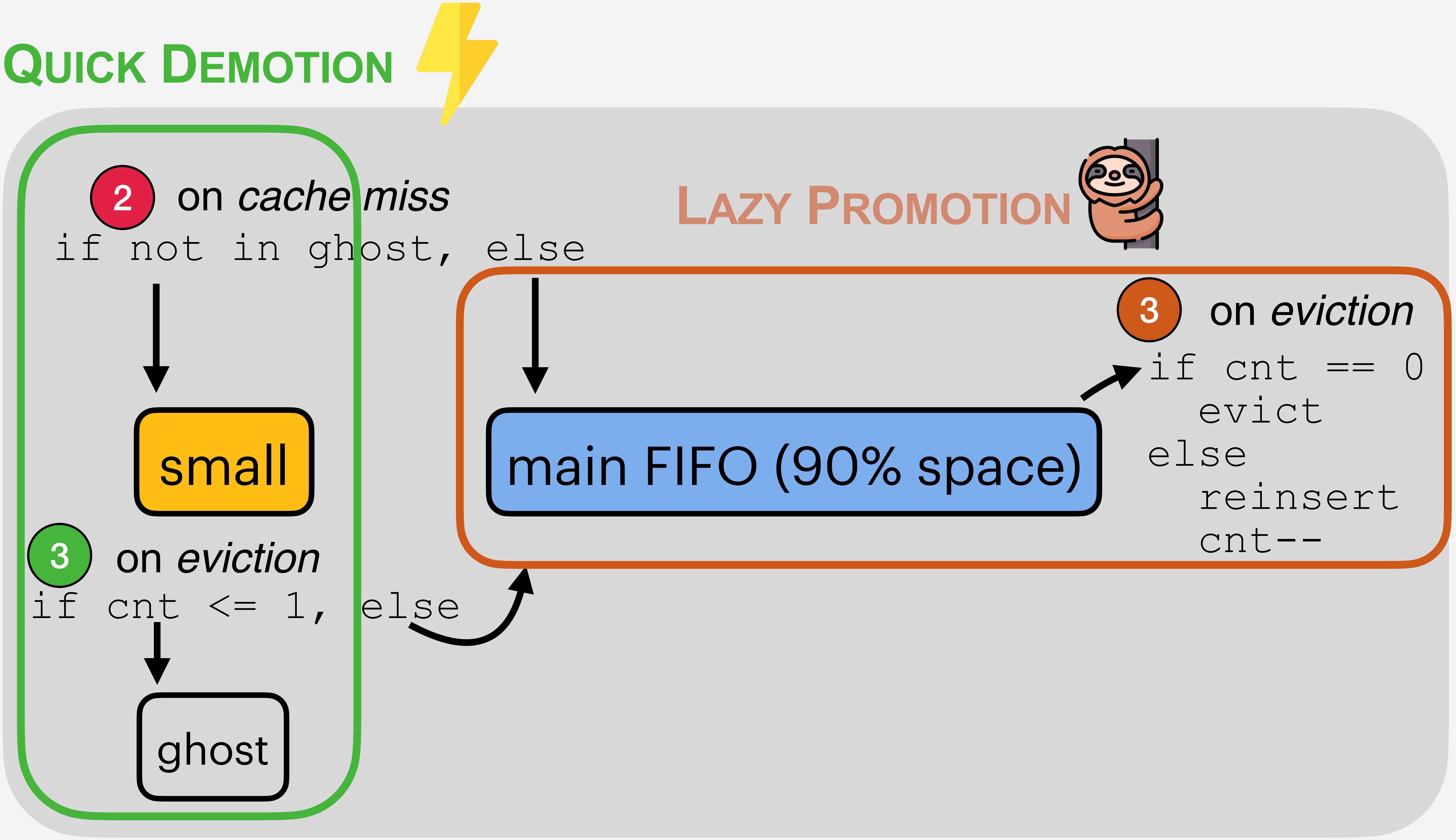
https://s3fifo.com

S3-FIFO design

```
struct object {
  …
  uint8_t cnt:2;
}
```

1 on *cache hit*

```
cnt++
```



S3-FIFO features

- **Simple and robust:** static queues
- **Fast:** no metadata update for most requests
- **Scalable:** no lock
- **Tiny metadata:** 2 bits
- **Flash-friendly:** sequential writes

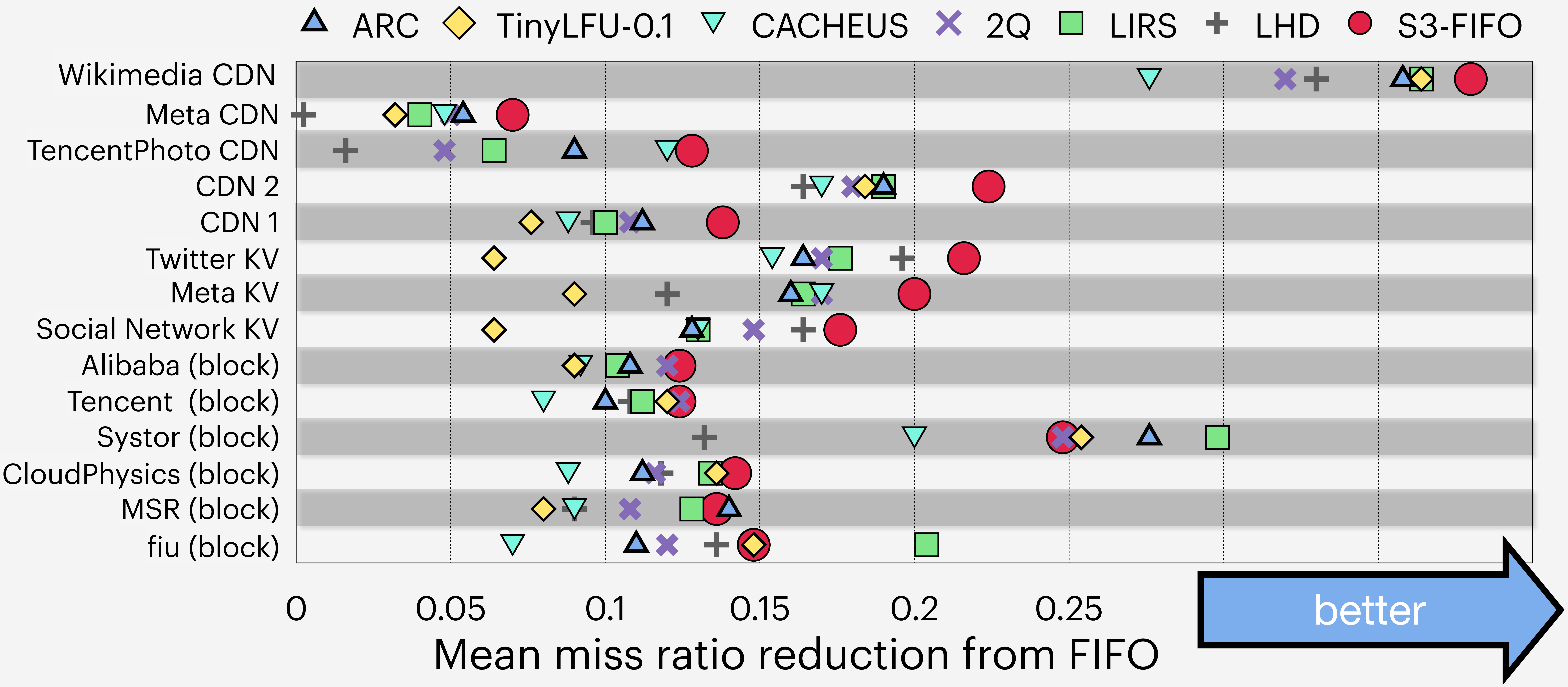
Can be implemented using one, two or three FIFO queues

Evaluation setup

- Data
  - 14 datasets, **6594** traces from Twitter, Meta, Microsoft, Wikimedia, Tencent, Alibaba, major CDNs…
  - **848 billion** requests, **60 billion** objects
  - collected between 2007 and 2023
  - block, key-value, object caches
- Platform
  - libCacheSim, cachelib
  - PDL cluster and CloudLab with 1 million core·hours
- Metric
  - miss ratio reduction from FIFO
  - throughput in Mops/sec

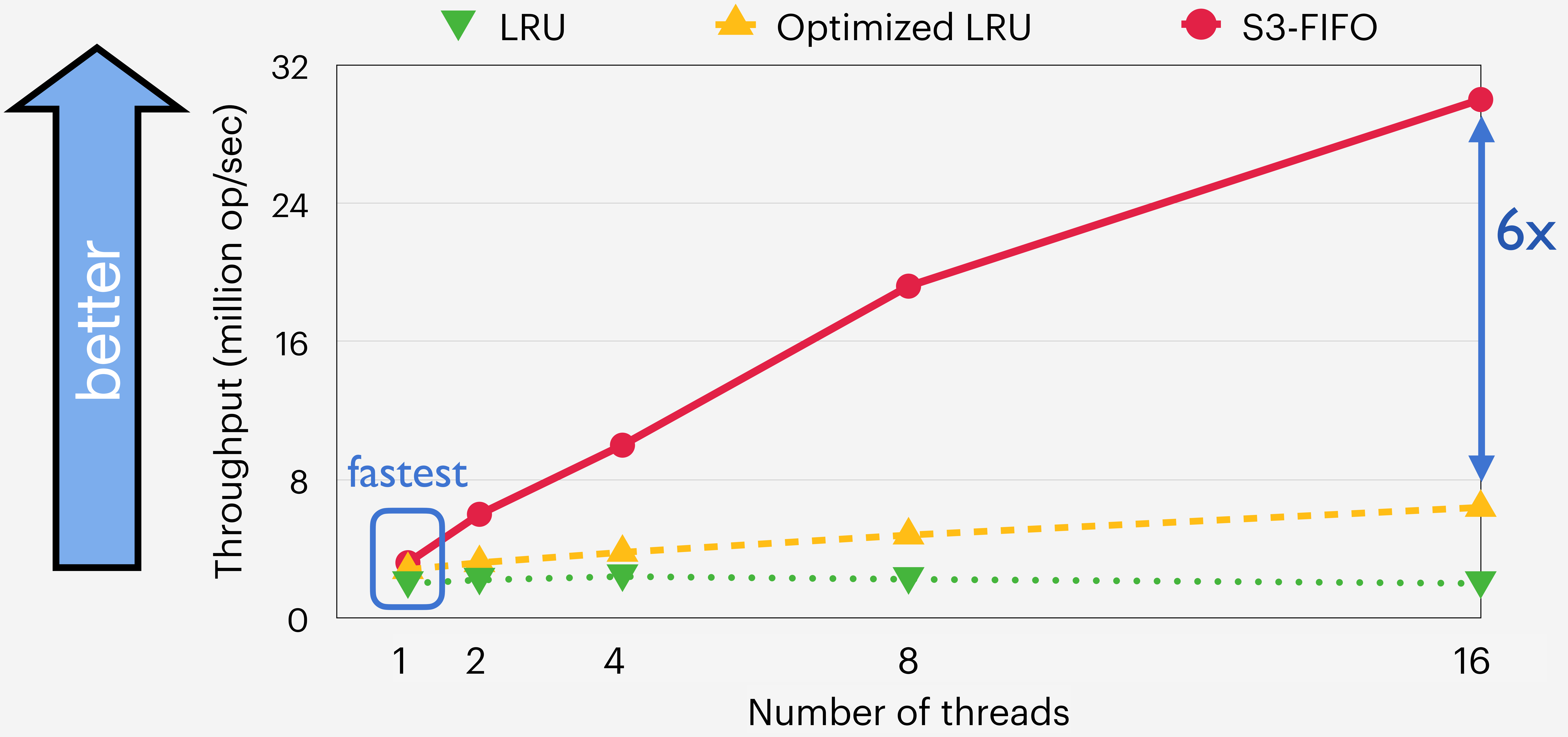
Data and software are open-sourced

S3-FIFO is efficient across datasets



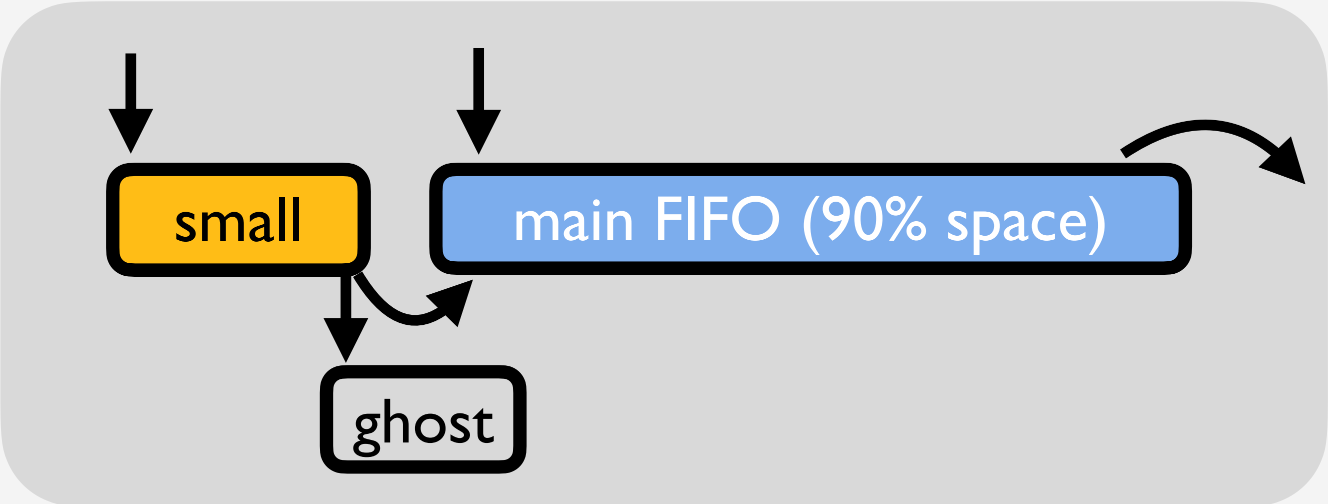
Evaluated on 6594 traces with 848 billion requests from 12 sources, collected between 2007 and 2023
This evaluation is million times larger than previous works

Throughput scales with number of threads



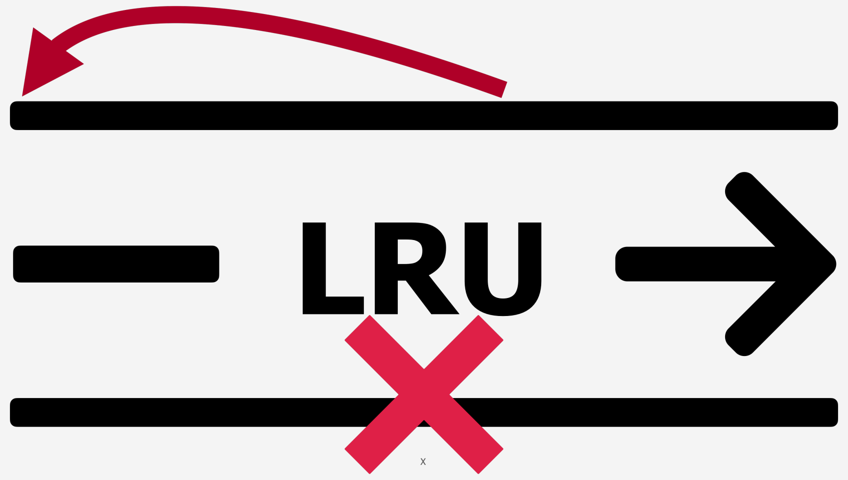
Implemented in Meta CacheLib, synthetic Zipf 1.0 workload, benchmarked on Cloudlab r6420 node

Recap



https://s3fifo.com

- S3-FIFO: simple, scalable caching with three static FIFO queues
  - prevalence of one-hit wonders
  - small FIFO queue: quickly evict one-hit wonders, reinsertion: keep popular objects
- The first work showing that FIFO queue is sufficient to design efficient algorithms
- S3-FIFO recognition and impact
  - Covered by many blogs, newsletters, meetups in English, Chinese, Korean, Japanese…
  - Deployed at Google, VMware, Redpanda…
  - More than ten open-source libraries implemented S3-FIFO in Rust, C, C++, JavaScript, Python…



SIEVE

An eviction algorithm simpler than LRU

https://sieve-cache.com

The secret to designing efficient eviction algorithms

Lazy Promotion

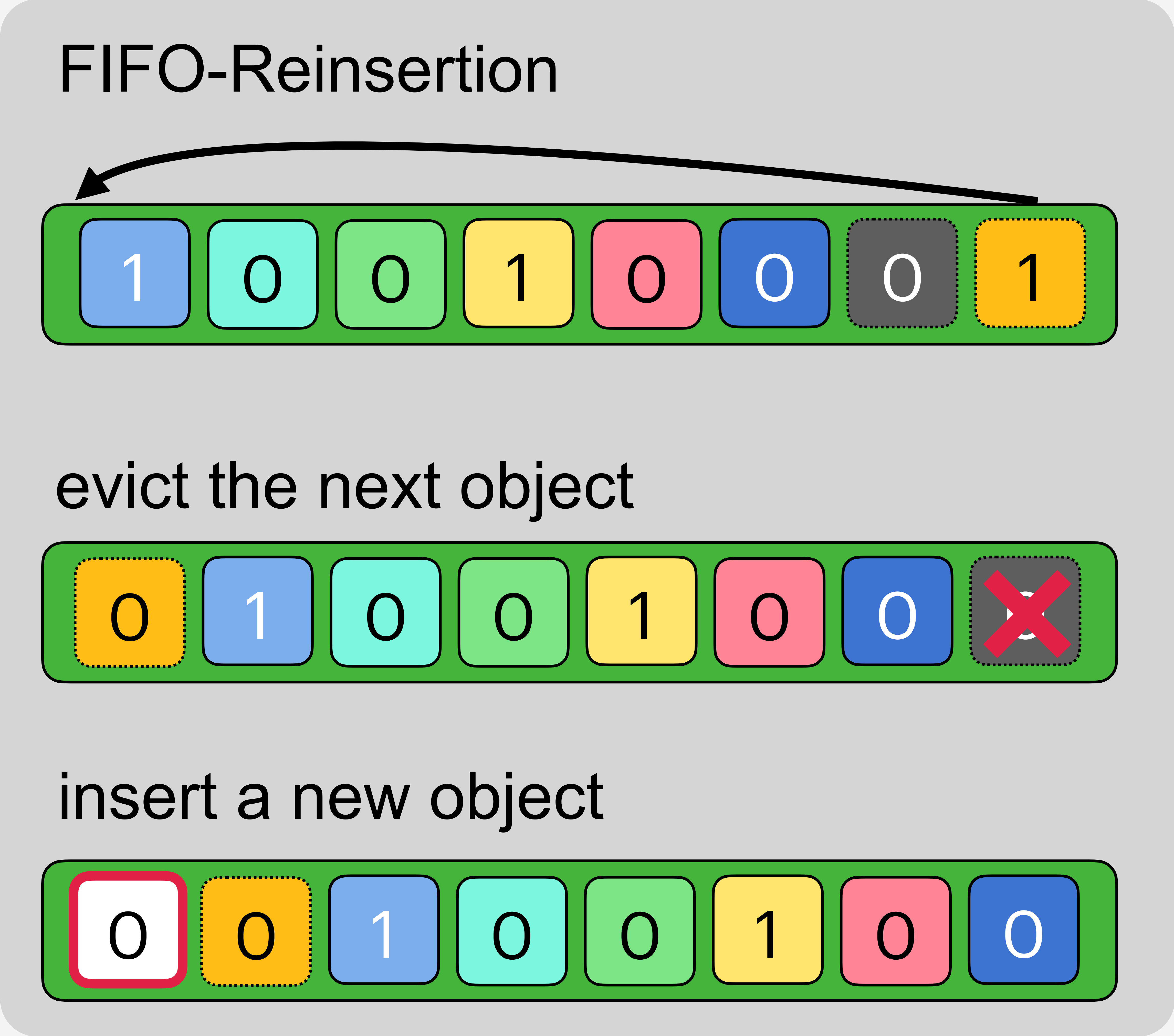
Retain popular objects with minimal effort

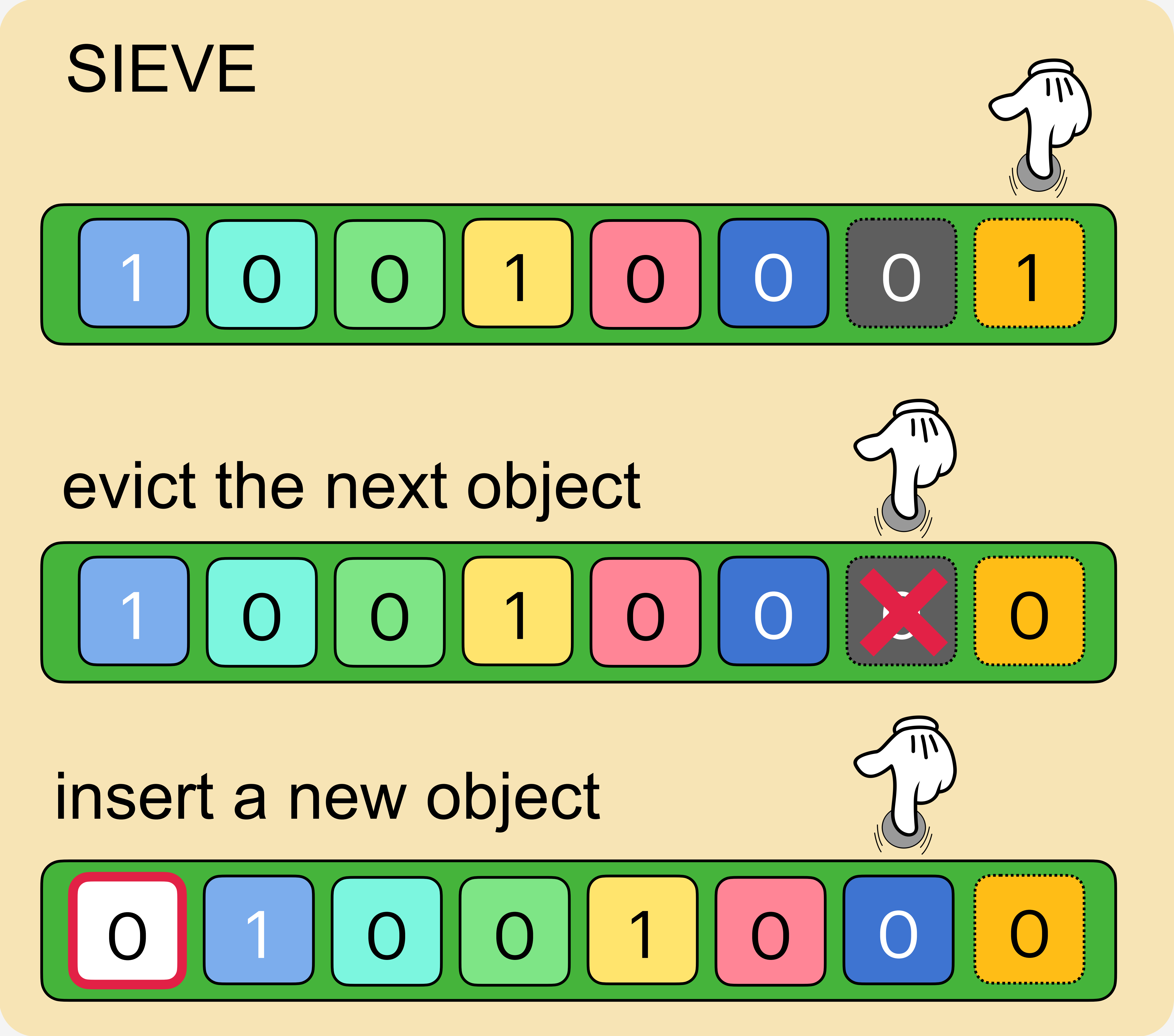
Quick Demotion

Remove unpopular objects fast, such as one-hit-wonders

SIEVE: combining lazy promotion and quick demotion

A small change turn FIFO-Reinsertion to SIEVE





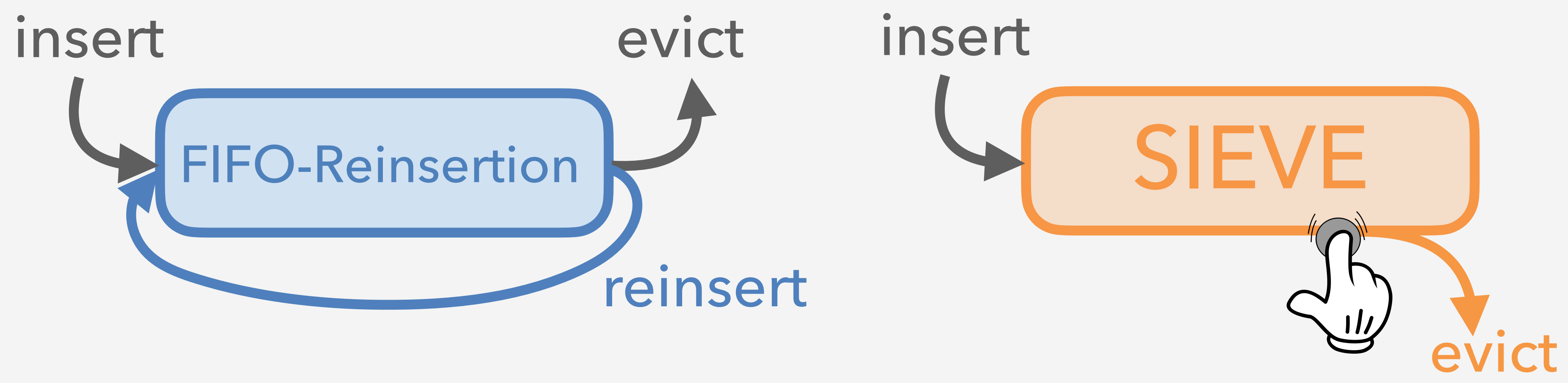
0: not accessed, 1: has accessed

SIEVE features

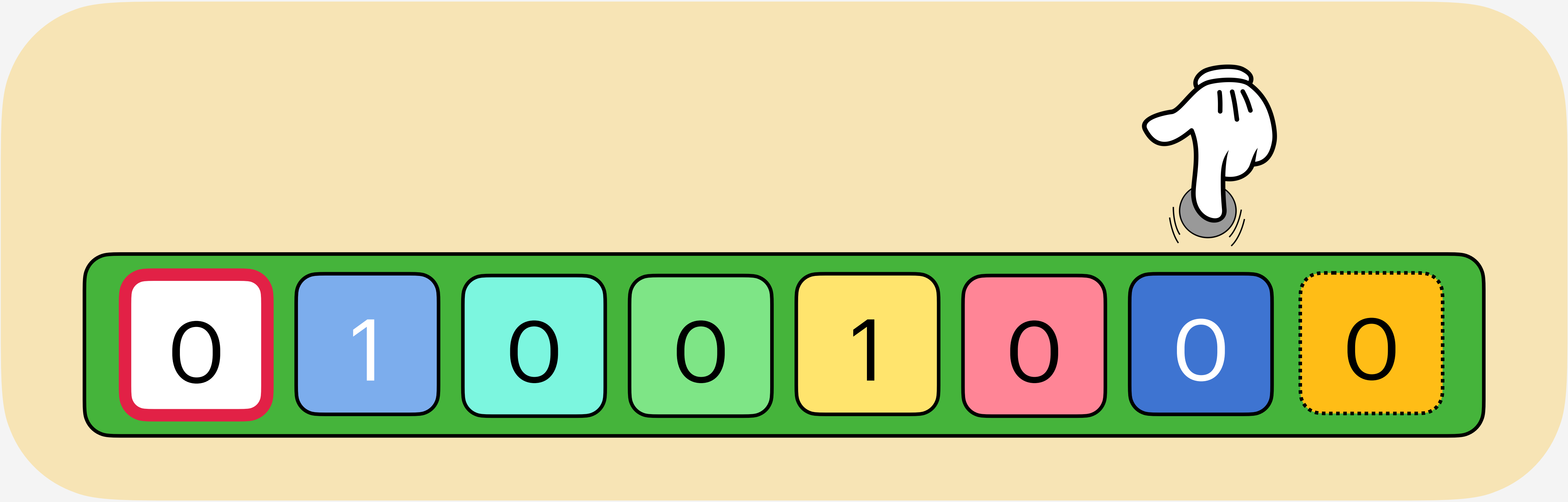
- Extremely simple
- ZERO parameter
- Fast and scalable
- Small per-object metadata
- TTL-friendly

Why does SIEVE work?

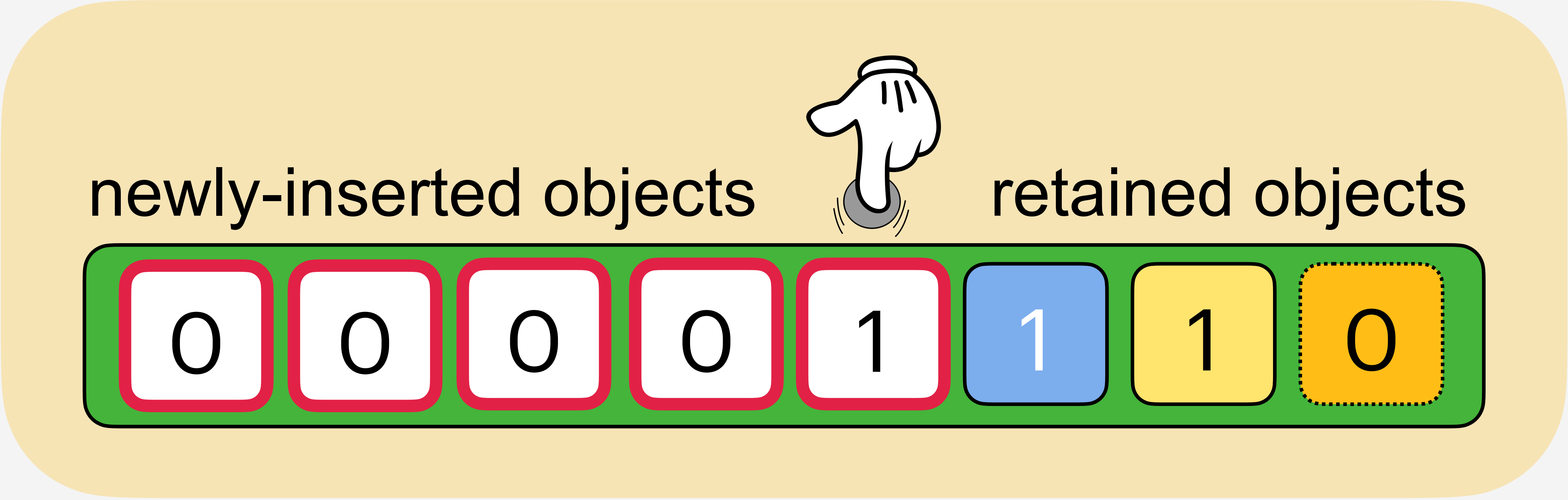
- Retain popular objects effectively



- Evict new objects quickly

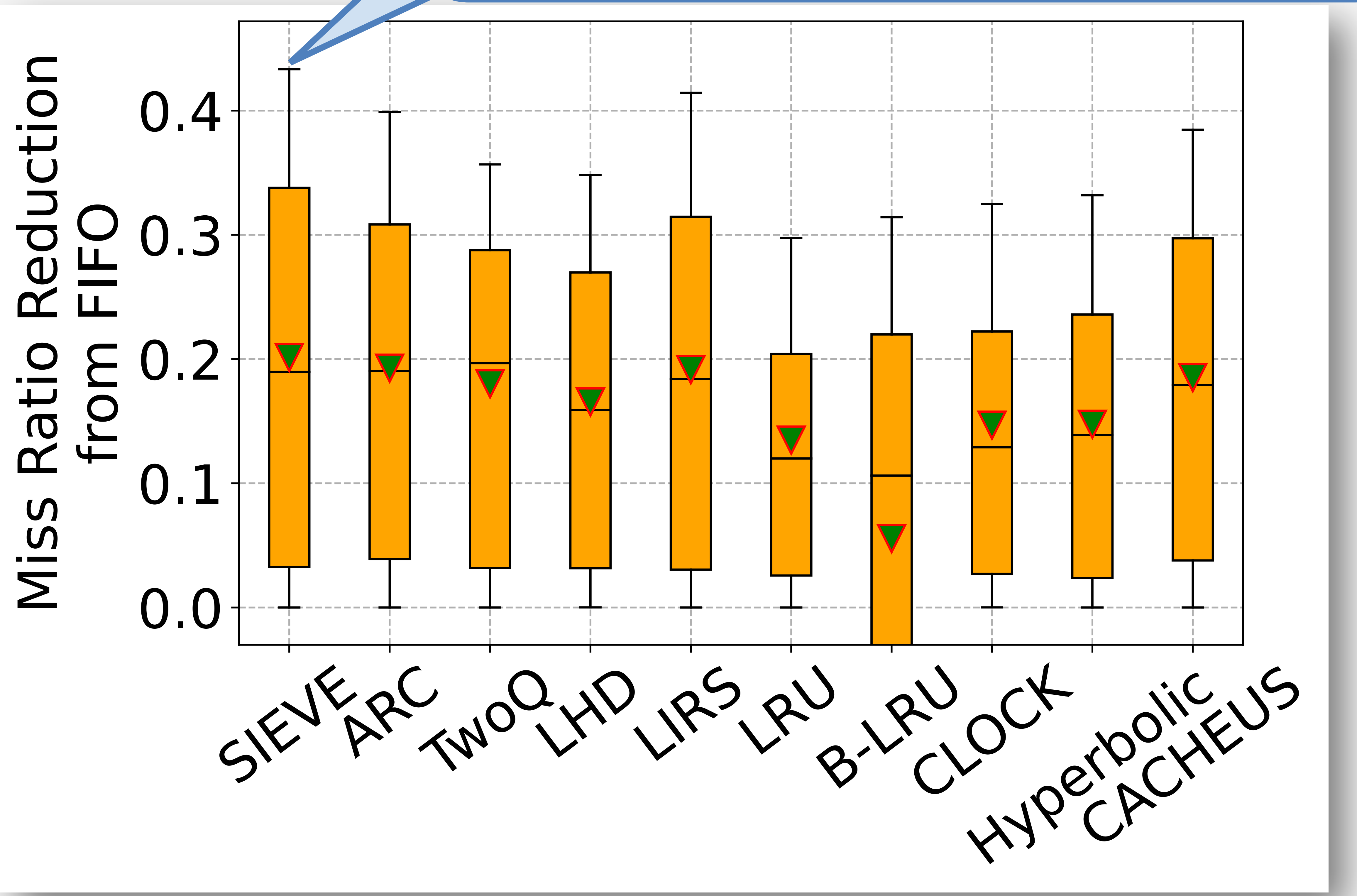


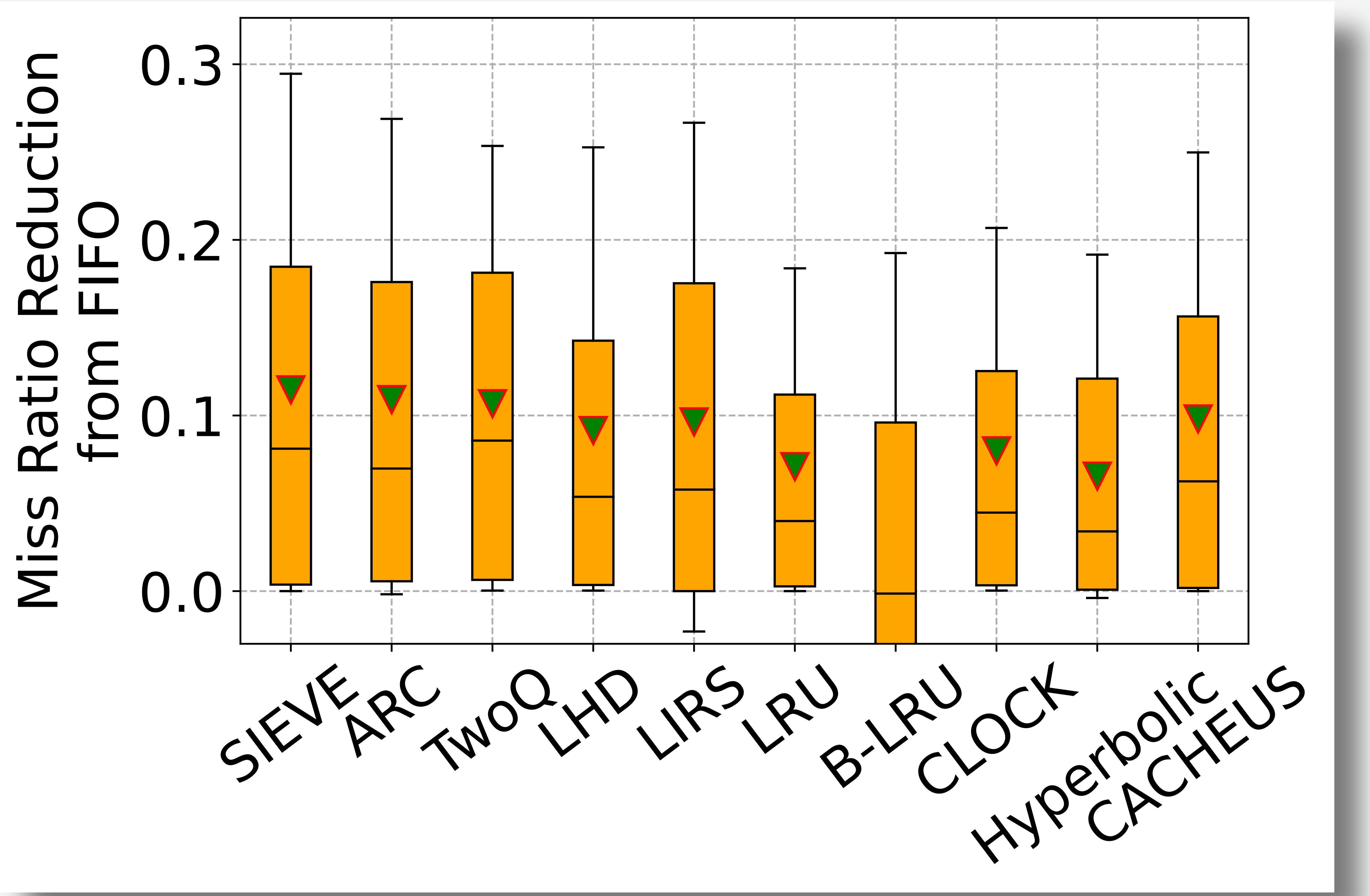
- Separate new and old objects



SIEVE achie… efficiency

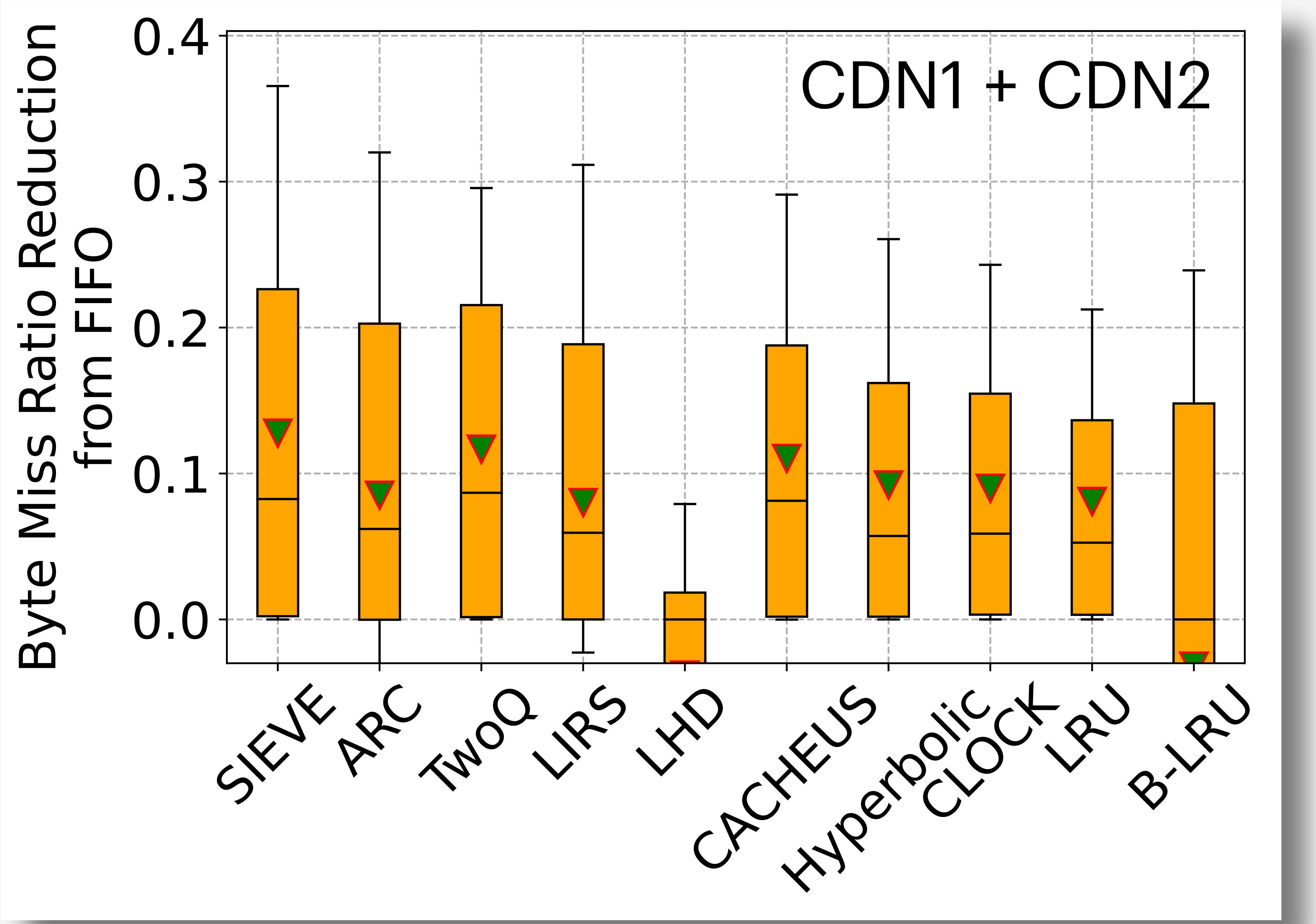
SIEVE reduces FIFO's miss ratio by more than 42% on 10% of the traces (top whisker) with a mean of 21%



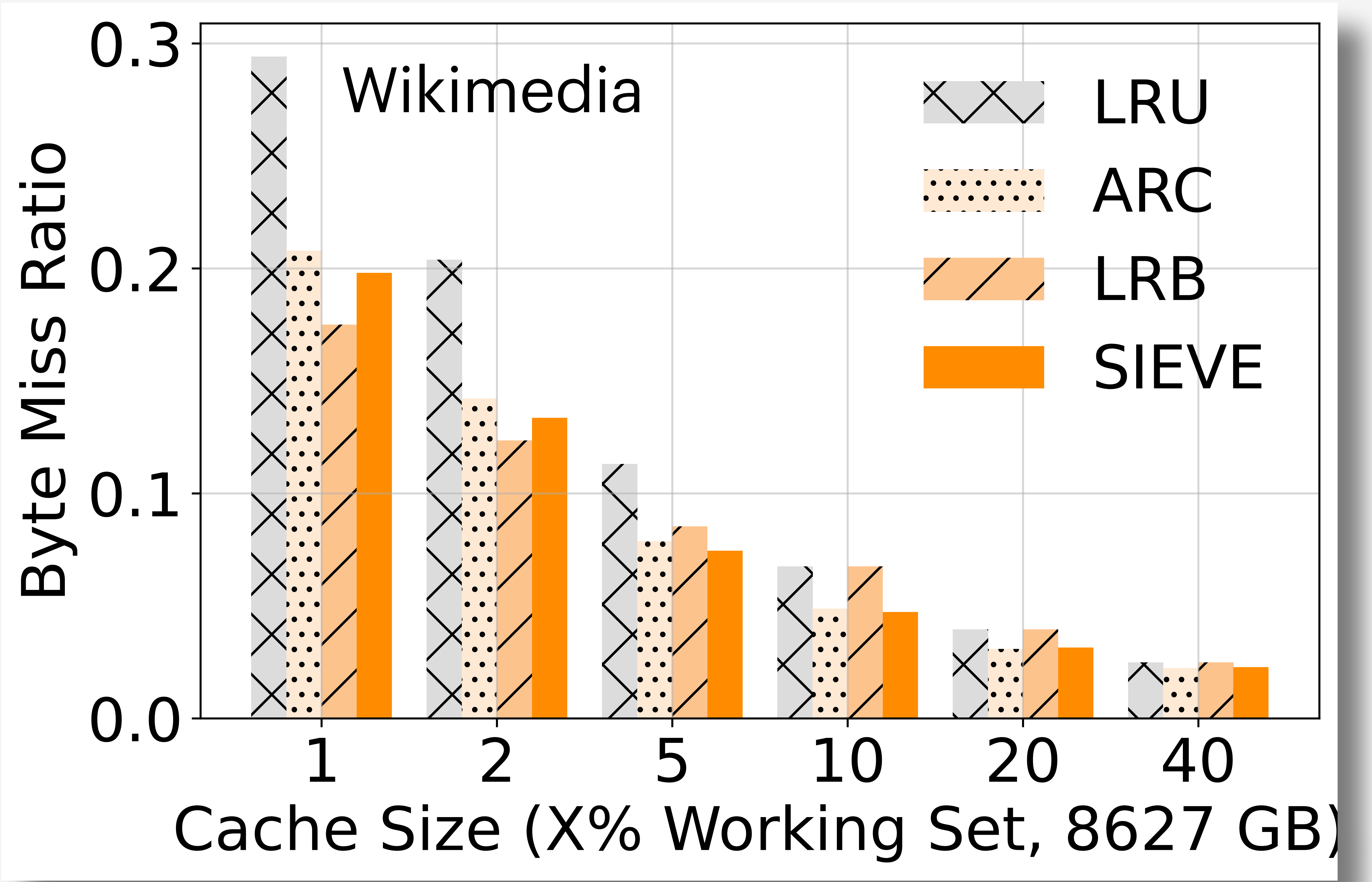


SIEVE also achieves the lowest miss ratio on the well-studied Zipfian workloads

SIEVE achieves the lowest byte miss ratio



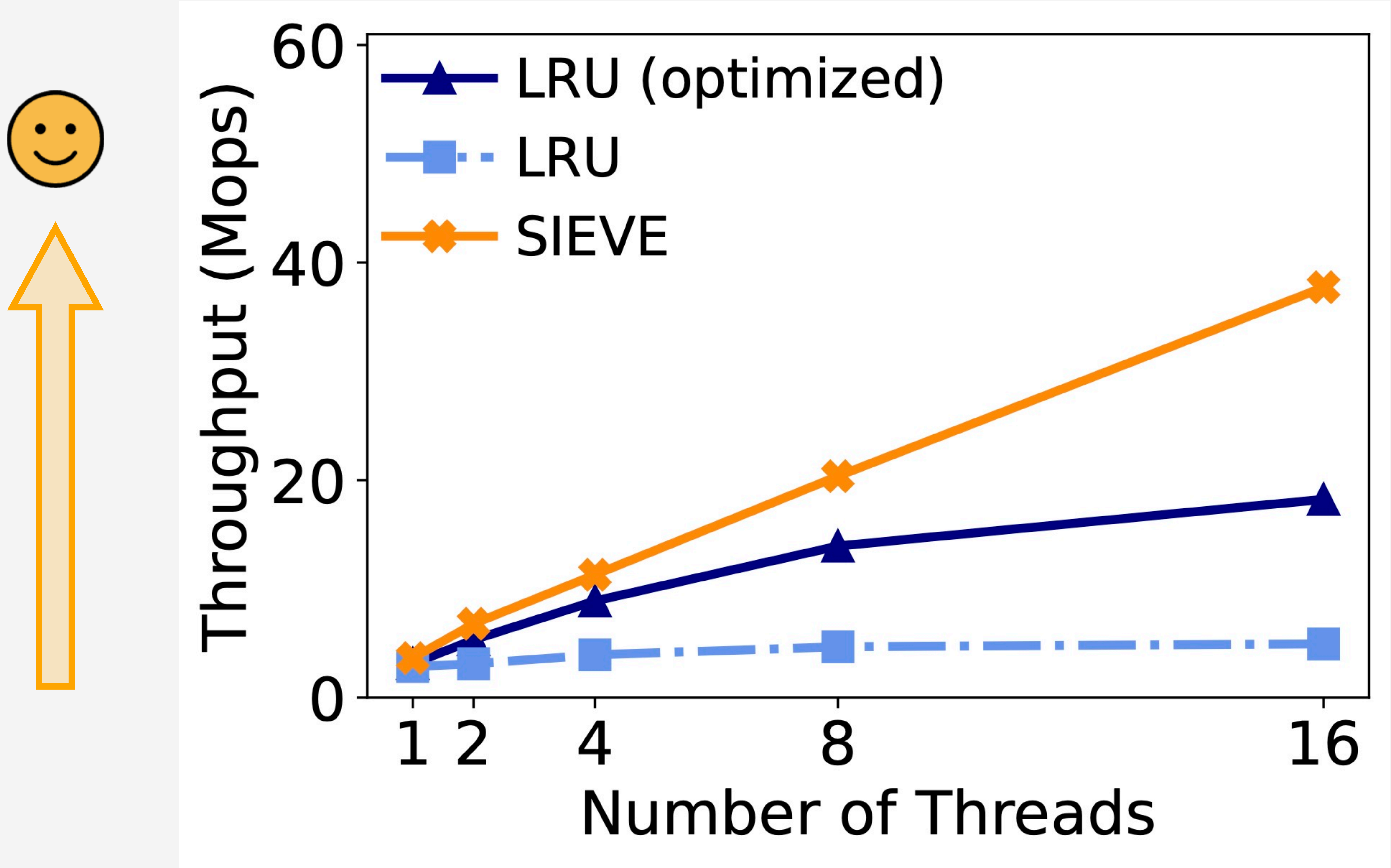
Better than all state-of-the-art algorithms



Small cache: better than ARC, close to LRB

Large cache: better than LRB

SIEVE throughput scales with the number of threads



compared to optimized LRU:
16% faster with a single thread, 2x faster with 16 threads

SIEVE is simple to implement

Cache library	Language	Eviction algorithm	Lines of change
groupcache	Golang	LRU	**21**
mnemonist	Javascript	LRU	**12**
lru-rs	Rust	LRU	**16**
lru-dict	Python + C	LRU	**21**

Adoption

Large systems: Pelikan Nyrkiö SkiftOS DragonFly DNSCrypt-proxy encrypted-dns-resolver

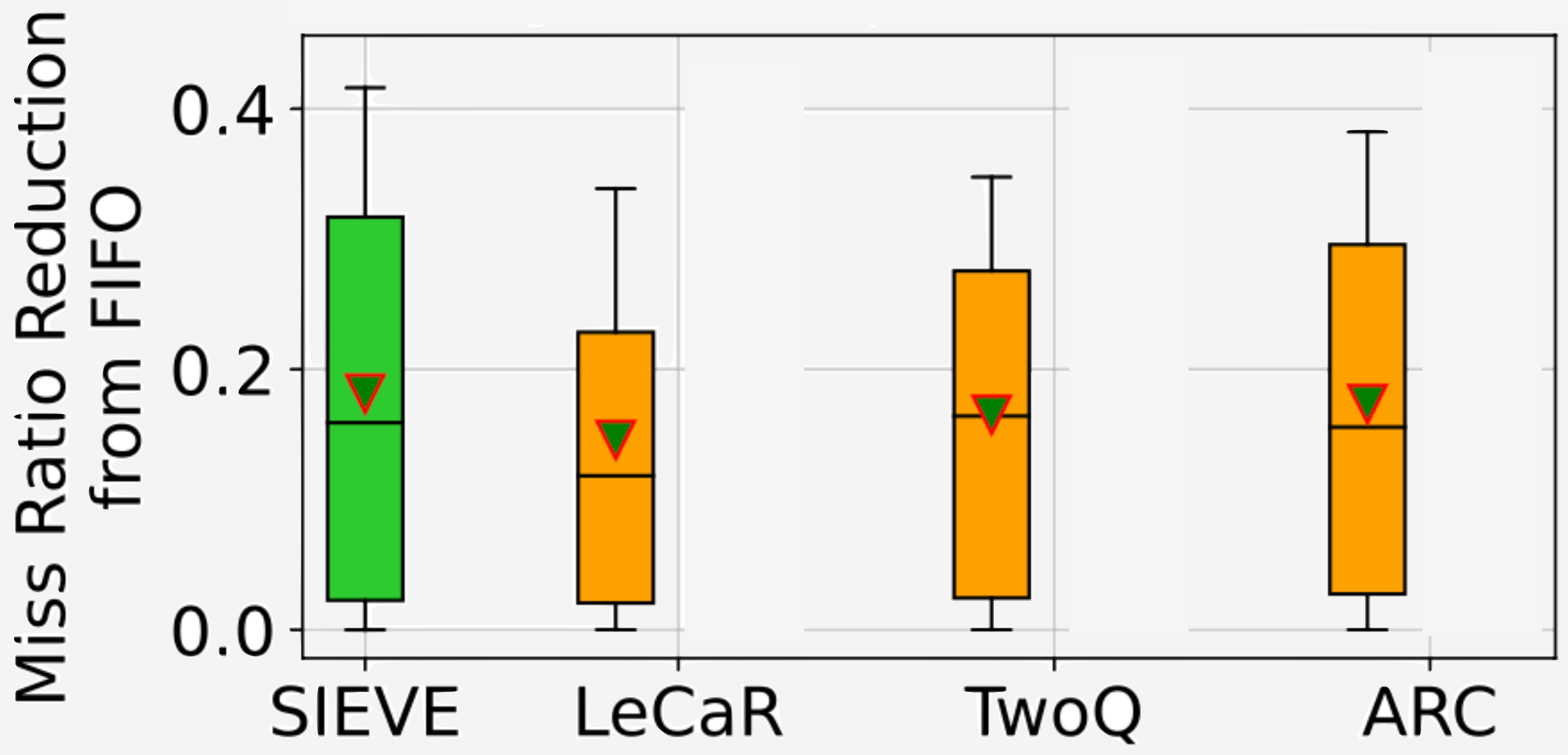
Cache libraries: golang-fifo js-sieve rust-sieve-cache go-sieve sieve_cache (Ruby) zig-sieve (Zig) sieve (Swift) sieve (JavaScript) sieve (Elixir) sieve (Nim) sieve-cache (Java) sieve (Python) sieve-cache-in-rust sieve-cache (JavaScript) gosieve, sieve (typescrpt)

SIEVE can be used as a cache primitive

LeCaR: LRU + LFU + ML

TwoQ: LRU + FIFO

ARC: LRU + LRU + 2 ghost queues



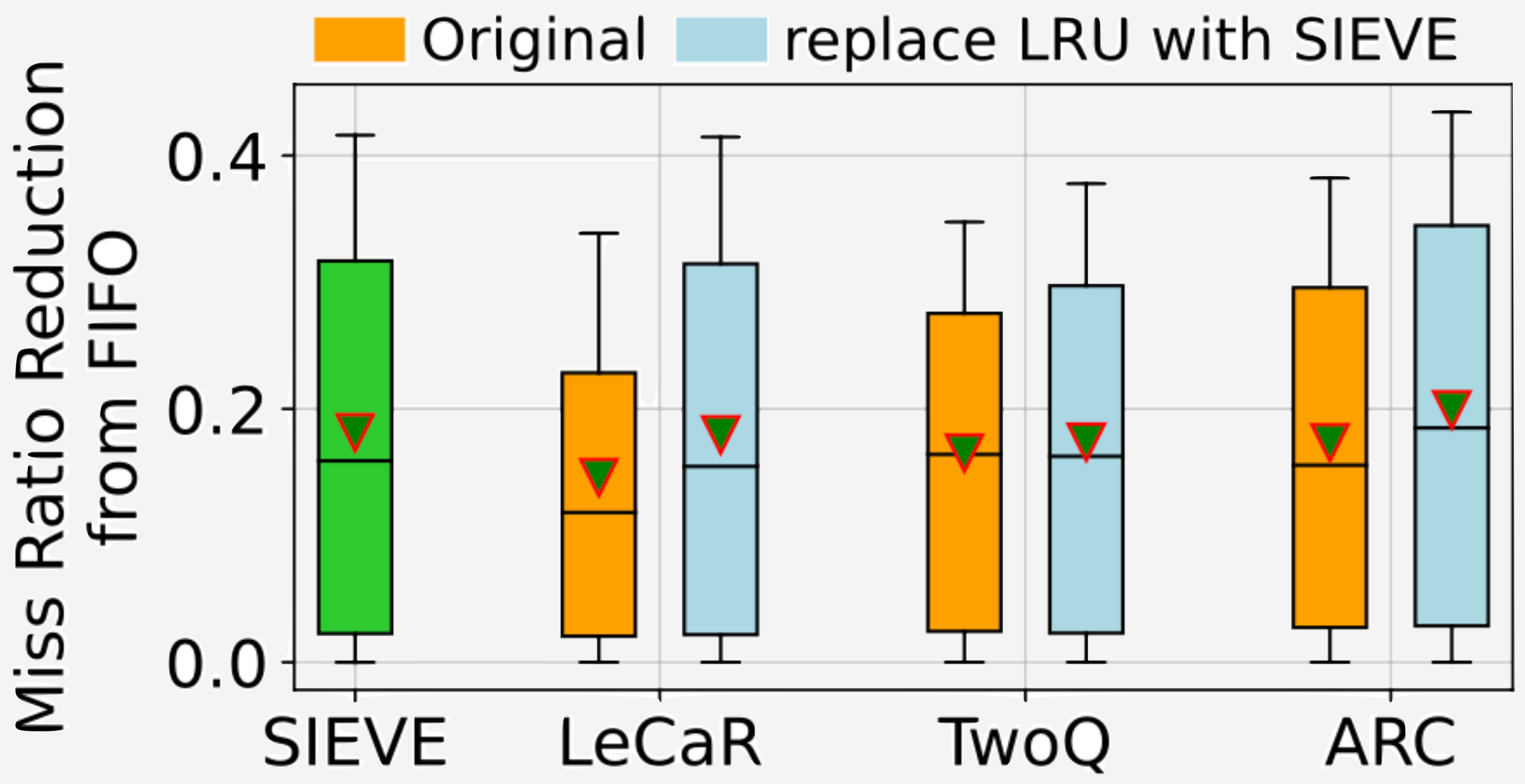
SIEVE can be used as a cache primitive

LeCaR: LRU + LFU + ML

TwoQ: LRU + FIFO

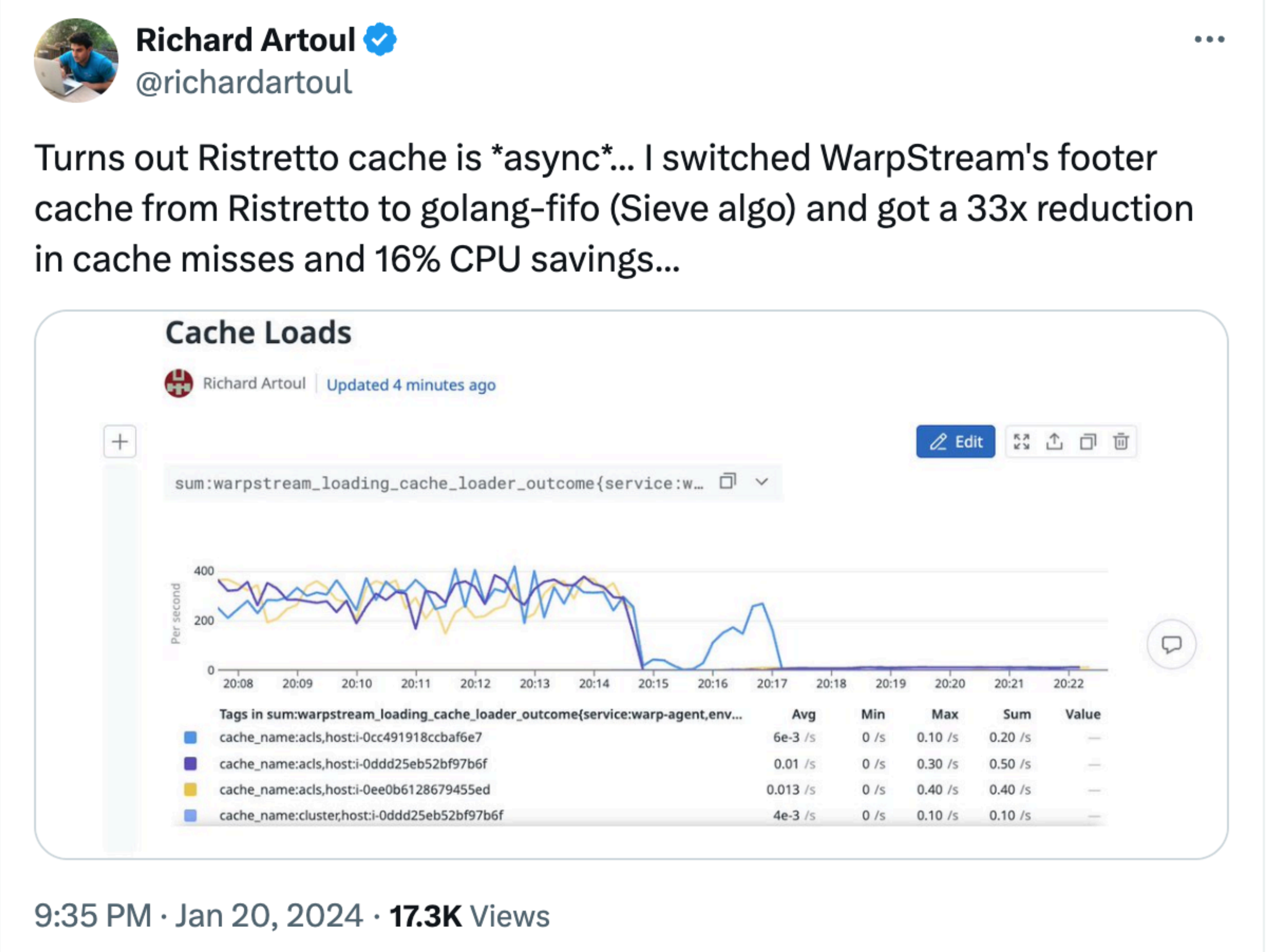
ARC: LRU + LRU + 2 ghost queues

Replace LRU with SIEVE



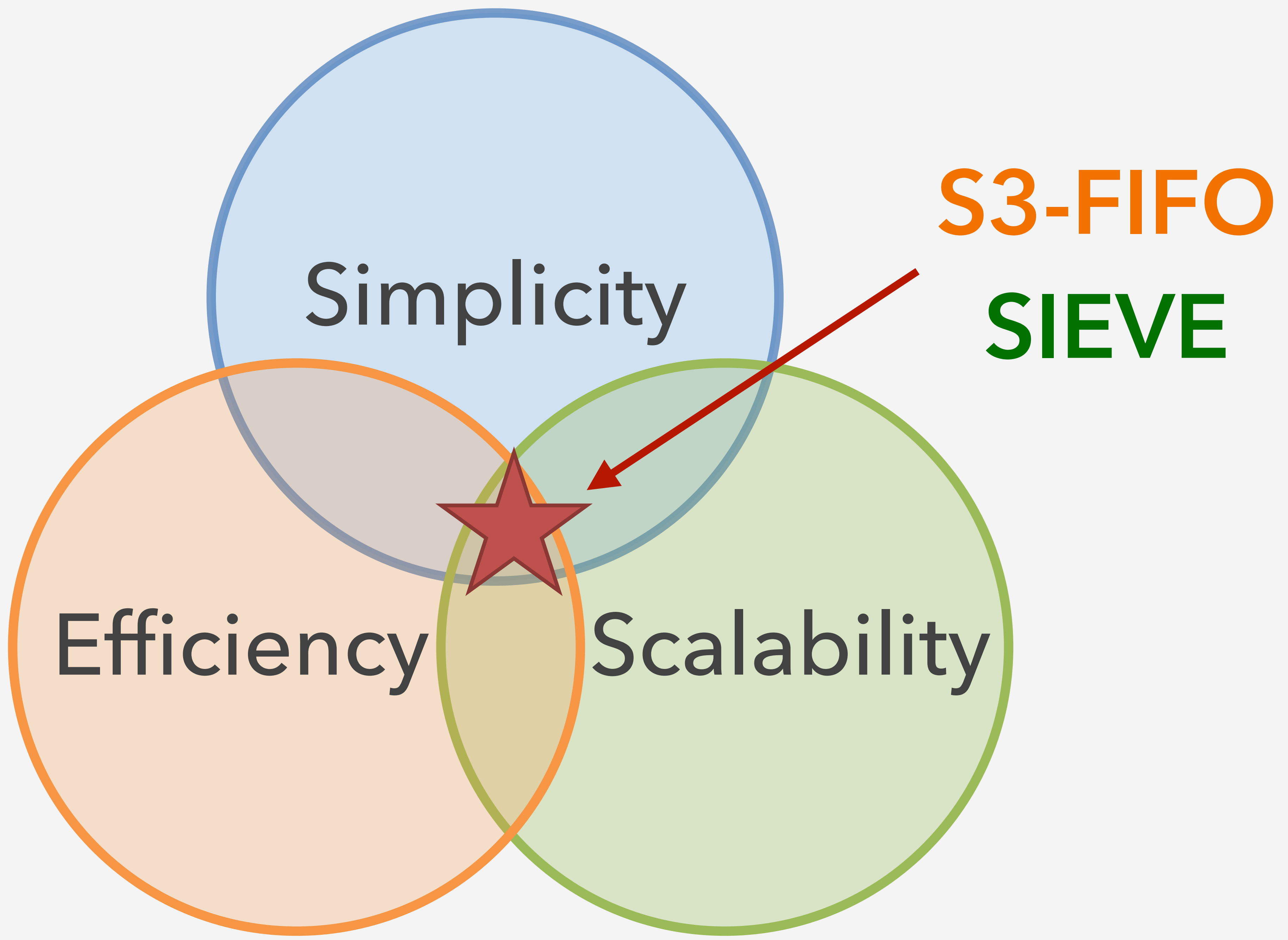
SIEVE has been widely used

- SIEVE is available in **20+** cache libraries with **12** programming languages
- Production systems integrated SIEVE: Pelican, SkiftOS, DragonFly...



Summary

- Two techniques
  - Lazy promotion
  - Quick demotion
- S3-FIFO: simple, scalable caching with three static FIFO queues
  - small queue for filtering
  - main queue with reinsertion
- SIEVE: the simplest algorithm combining lazy promotion and quick demotion



https://s3fifo.com https://sieve-cache.com

How we can build better storage systems together

Juncheng Yang
https://junchengyang.com