



# FIFO Queues

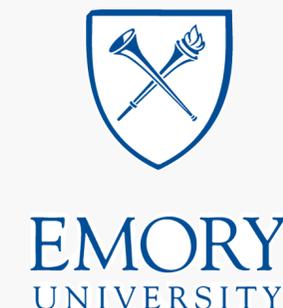
are **all** You Need for Cache Eviction

**Juncheng Yang**

Yazhuo Zhang, Ziyue Qiu, Yao Yue, K. V. Rashmi

**Carnegie  
Mellon  
University**

**Carnegie Mellon  
Parallel Data Laboratory**



# Software cache and eviction

- Ubiquitous deployments of software caches
  - page cache, block cache, database cache
  - key-value cache, object cache...
- Cache metrics
  - efficiency / effectiveness: miss ratio
  - throughput and scalability: requests/sec
  - simplicity
- A core component of cache design: **eviction**



Cachelib



Pelikan



# A long history of research centered around LRU

- Least-recently-used (LRU)
  - maintain objects in a queue with last-access order
  - update metadata (with locking) on each read request
- Problems
  - not scalable
  - not scan-resistant



# A long history of research centered around LRU

- Improve LRU's efficiency
  - **add more techniques/queues/metrics:** LIRS<sup>[SIGMETRICS'02]</sup>, LRU-K<sup>[SIGMOD'93]</sup>, 2Q<sup>[VLDB'94]</sup>, MQ<sup>[ATC'01]</sup>, ARC<sup>[FAST'03]</sup>, TinyLFU<sup>[TOS'17]</sup>, LRB<sup>[NSDI'20]</sup>, CACHEUS<sup>[FAST'21]</sup>...
  - sacrifice throughput and/or scalability
- Improve LRU's throughput and scalability
  - **reduce #operations/locks per-request:** relaxed LRU, CLOCK variants<sup>[NSDI'13]</sup>, FrozenHot<sup>[Eurosys'23]</sup>
  - conventional wisdom: sacrifice efficiency, our finding<sup>[HotOS'23]</sup> shows not true
- **State-of-the-arts: tradeoff between efficiency and throughput**



**[HotOS'23]** FIFO queues  
can be better than LRU

# An alternative: FIFO eviction algorithm

- First-in-first-out (FIFO)
  - simpler
  - fewer metadata
  - less computation
  - more scalable
  - flash-friendly



**The only drawback:**  
**FIFO has a high miss ratio**

**Can we design an efficient  
FIFO-based algorithm?**

# Observation

## More one-hit wonders than you would have expected

- One-hit wonder: objects appeared once in the sequence
- Zipfian workloads: One-hit-wonder ratio **decreases with sequence length (measured in #obj)**
- Why short sequences? A cache starts eviction after seeing a short request sequence



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%
1	4	3	2 (B, C)	66%

# Observation

## More one-hit wonders than you would have expected

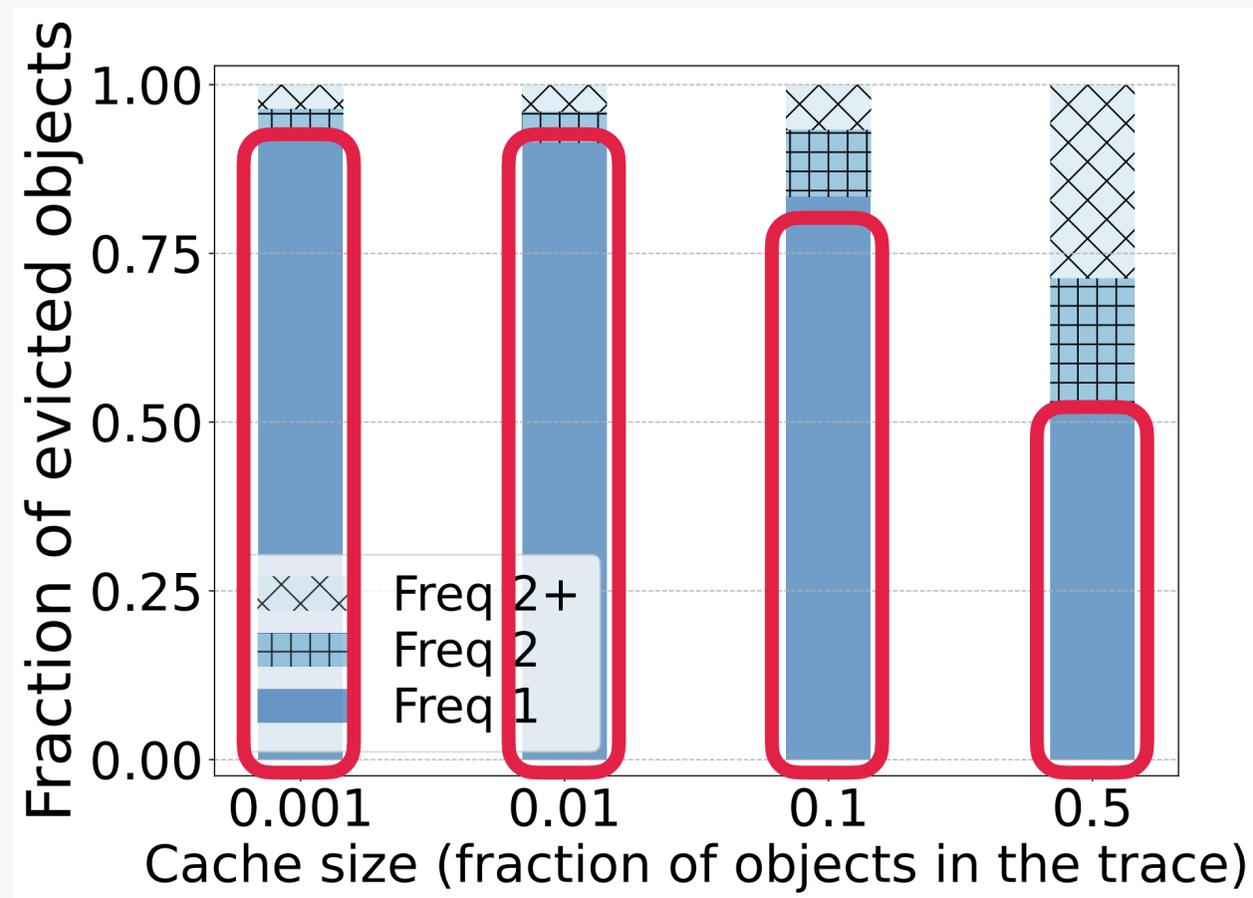
- One-hit wonder: objects appeared once in the sequence
- Zipfian workloads: One-hit-wonder ratio **decreases with sequence length (measured in #obj)**
- Why short sequences? A cache starts eviction after seeing a short request sequence

the one-hit-wonder ratio of 10% of week-long traces:

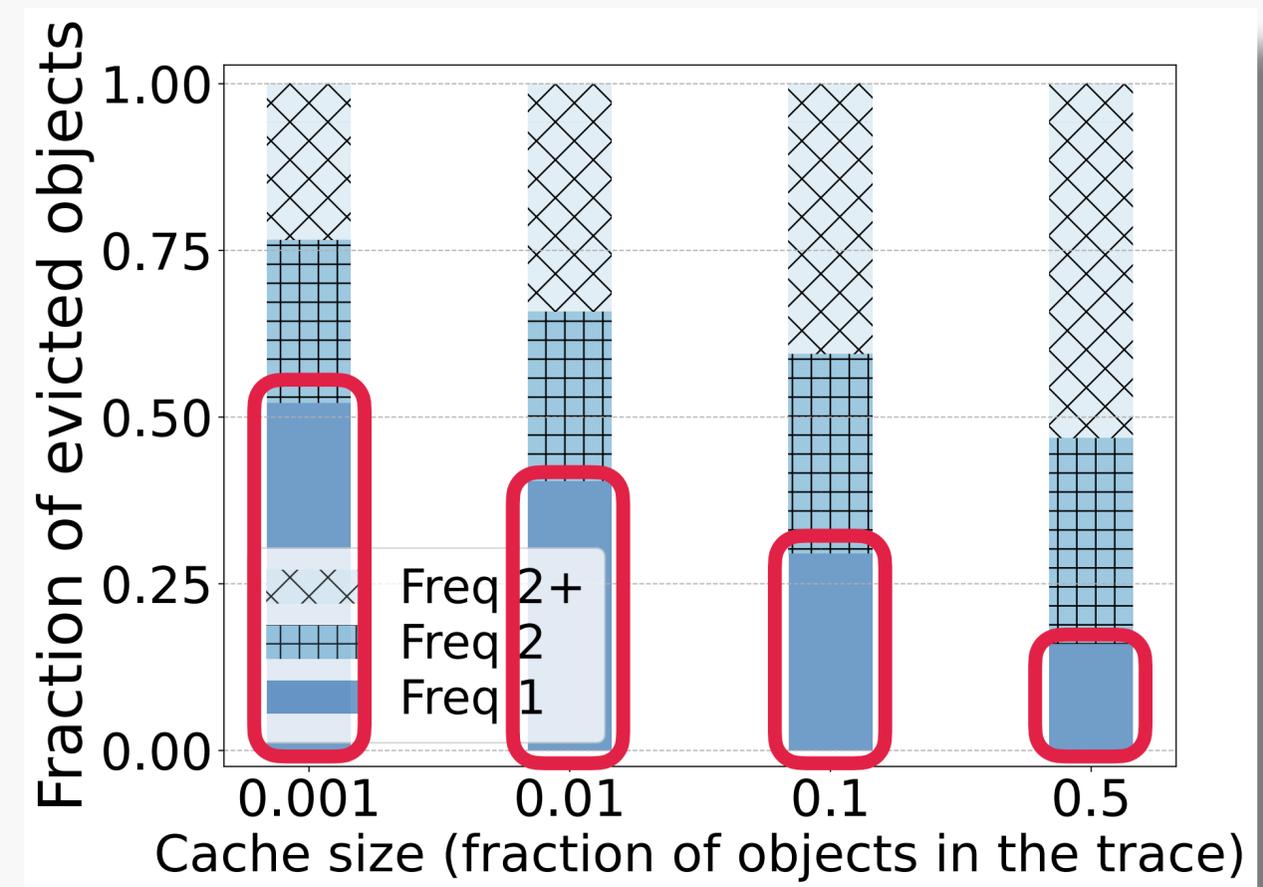
**72%** (mean on **6594** traces)

# Observation

most objects in the cache are one-hit wonders



LRU cache running MSR workload

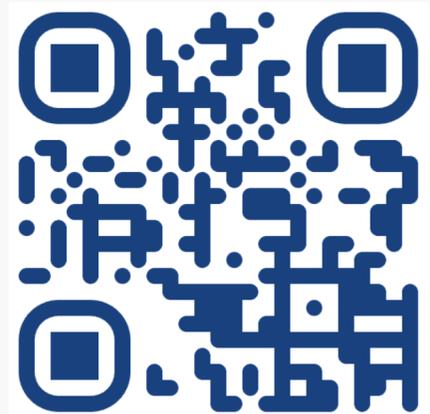


LRU cache running Twitter workload

# S3-FIFO Design

Simple, Scalable caching with three Static FIFO queues

<https://s3fifo.com>

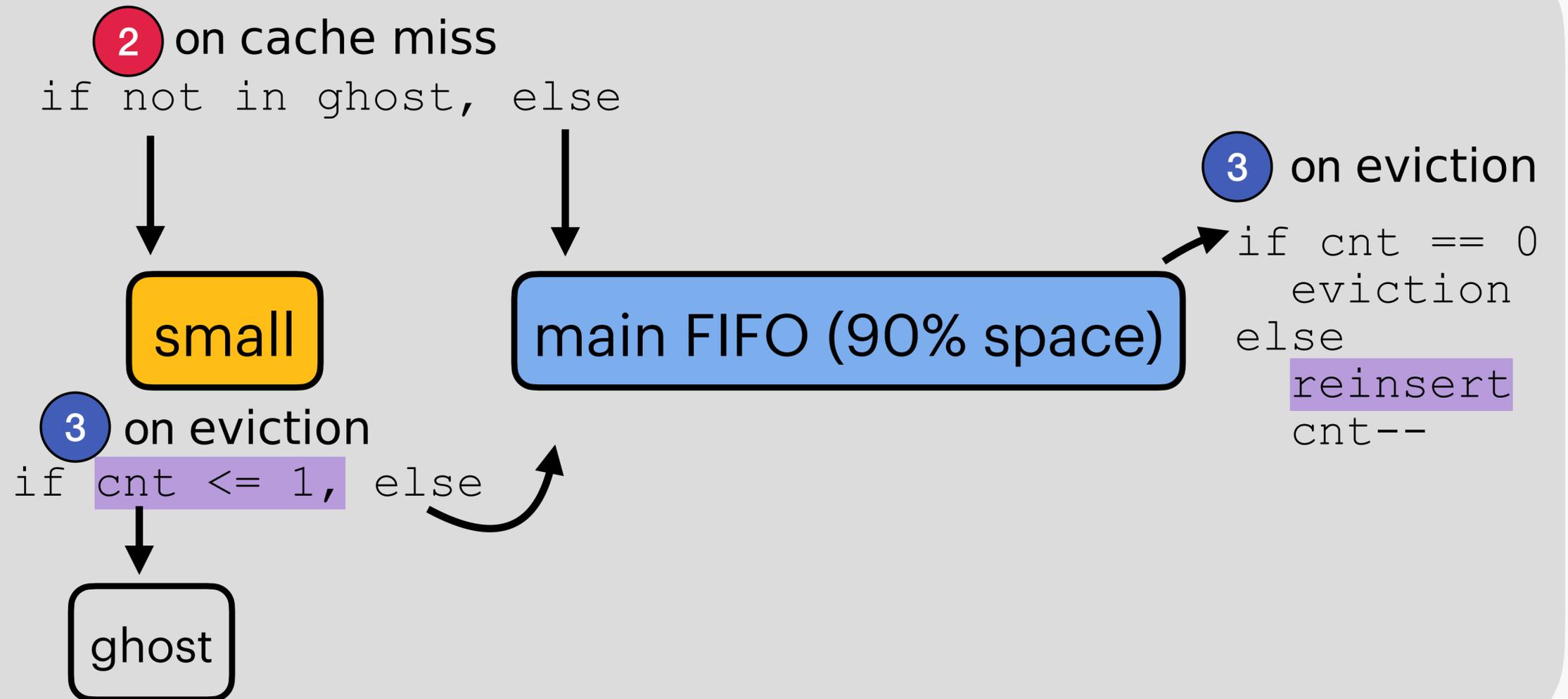


# S3-FIFO design

Simple, Scalable eviction algorithm with three Static FIFO queues

```
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

**Implementation:** one, two or three FIFO-queues

# S3-FIFO evaluation

# Evaluation setup

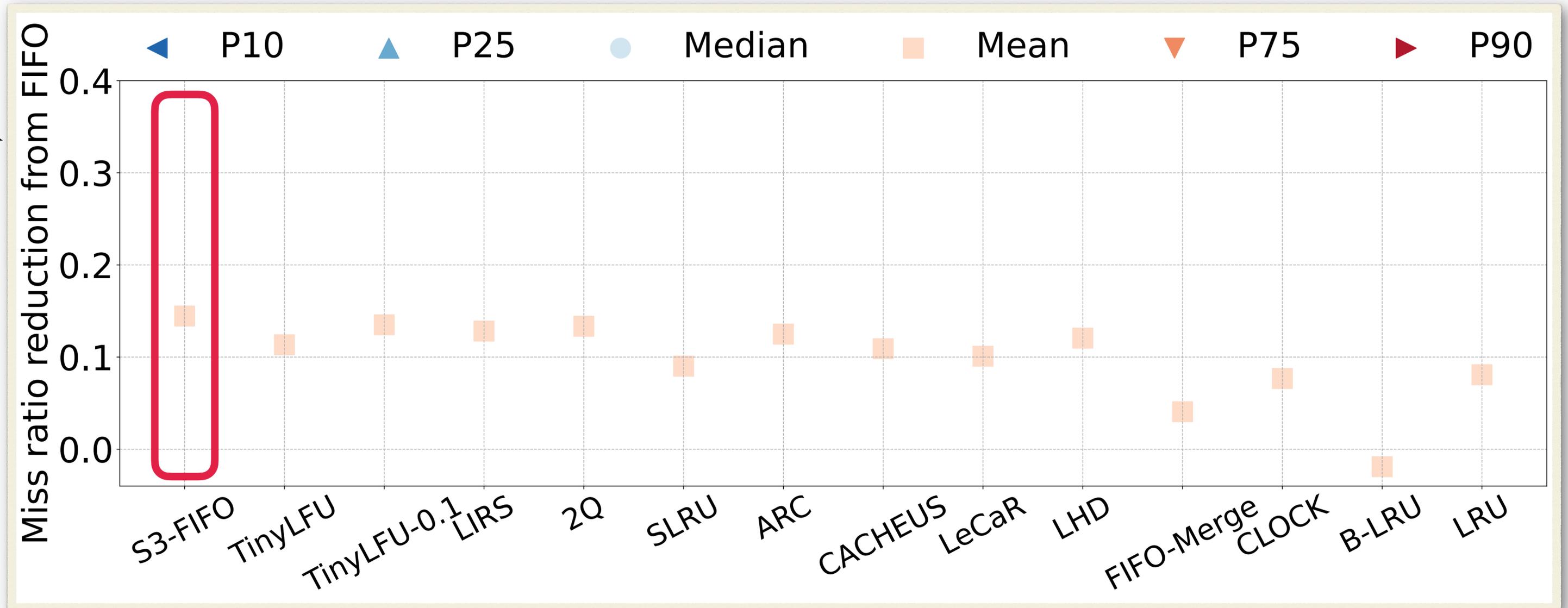
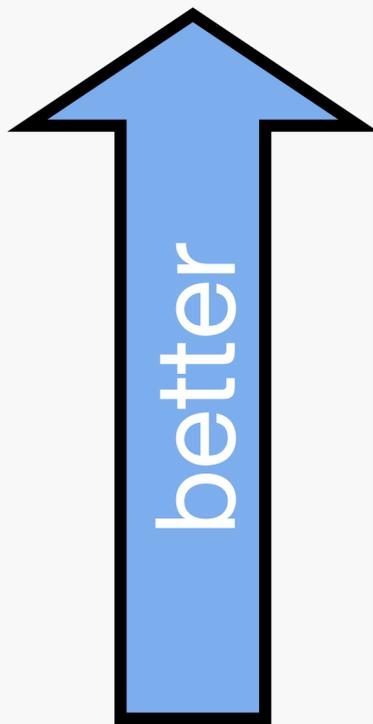
- Data
  - 14 datasets, **6594** traces from Twitter, Meta, Microsoft, Wikimedia, Tencent, Alibaba, major CDNs...
  - **848 billion** requests, **60 billion** objects
  - collected between 2007 and 2023
  - block, key-value, object caches
- Platform
  - libCacheSim, cachelib
  - CloudLab with 1 million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



CloudLab

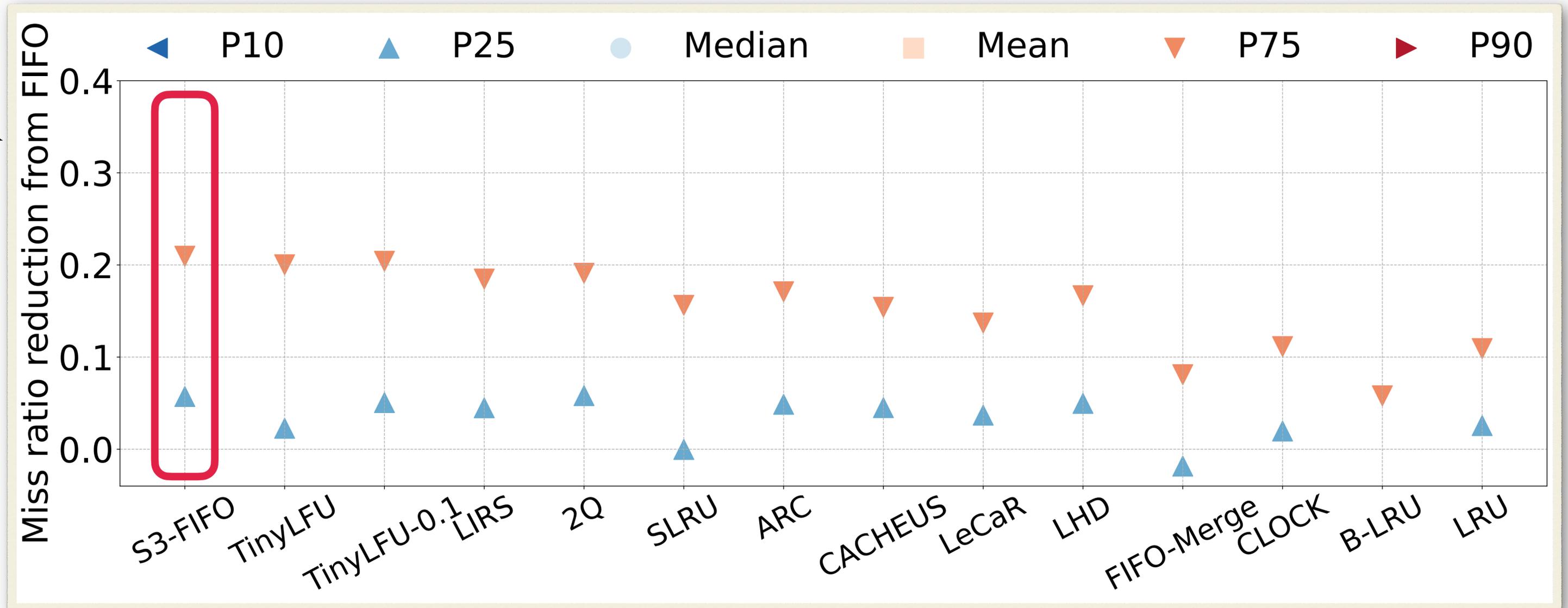
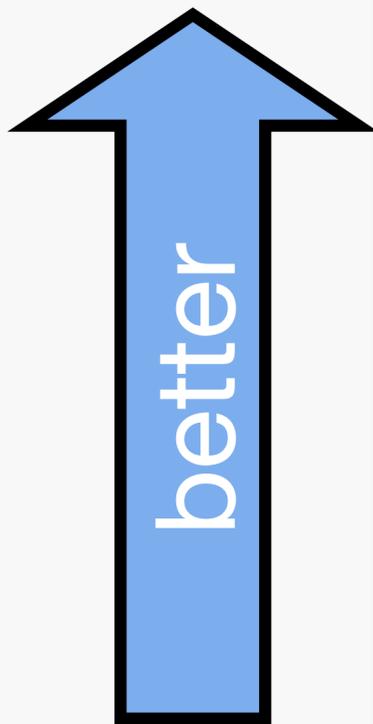
# Efficiency

Miss ratio reduction distribution across all traces



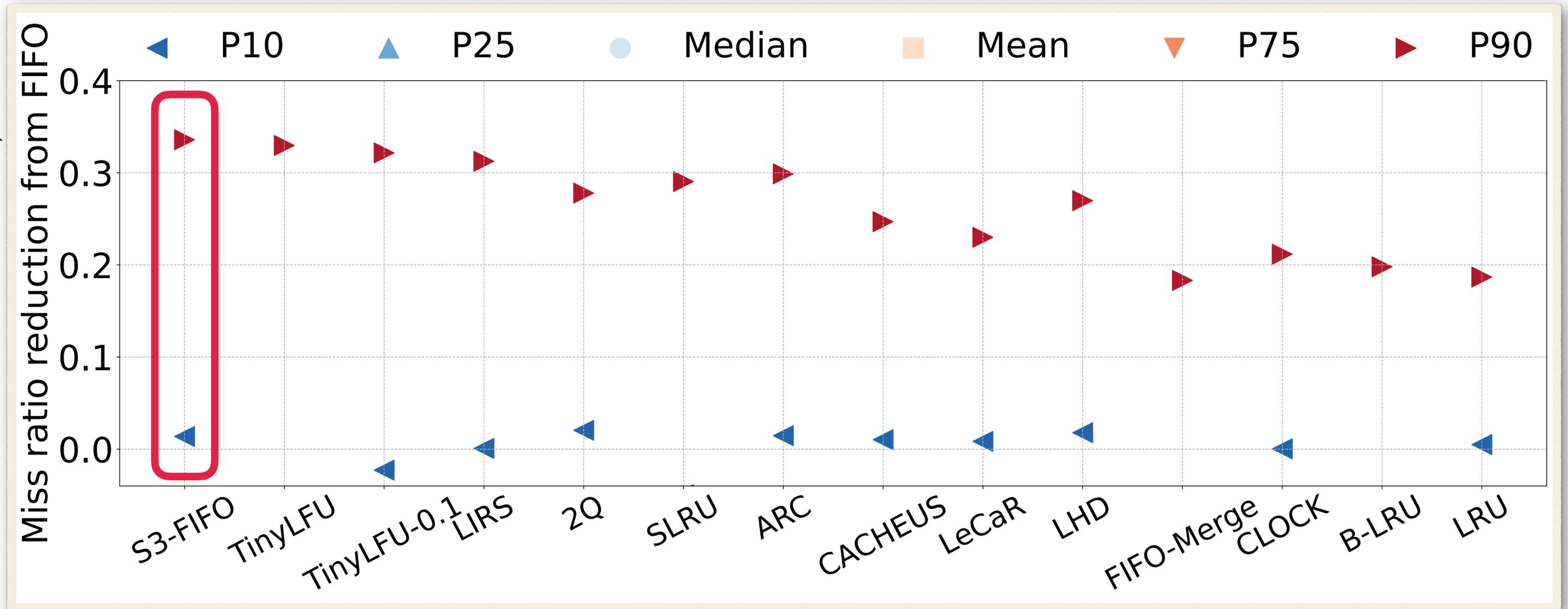
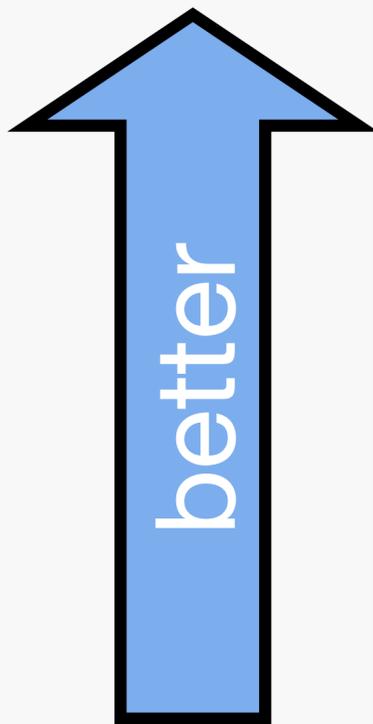
# Efficiency

Miss ratio reduction distribution across all traces



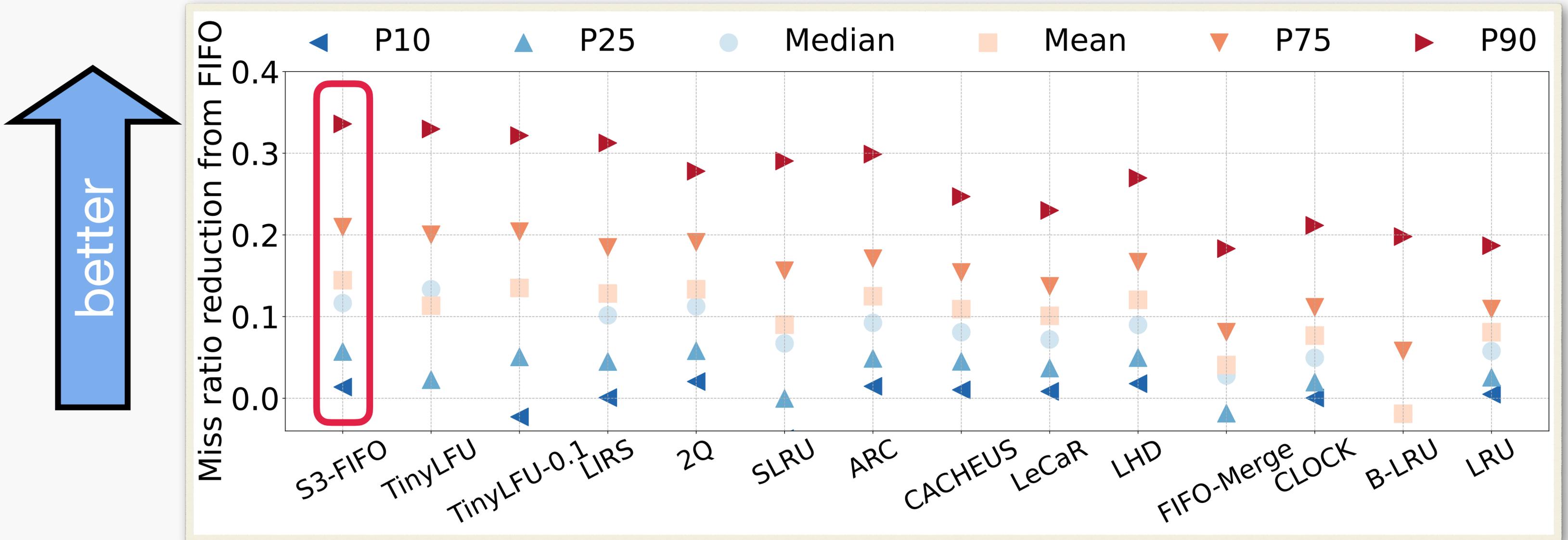
# Efficiency

Miss ratio reduction distribution across all traces



# Efficiency

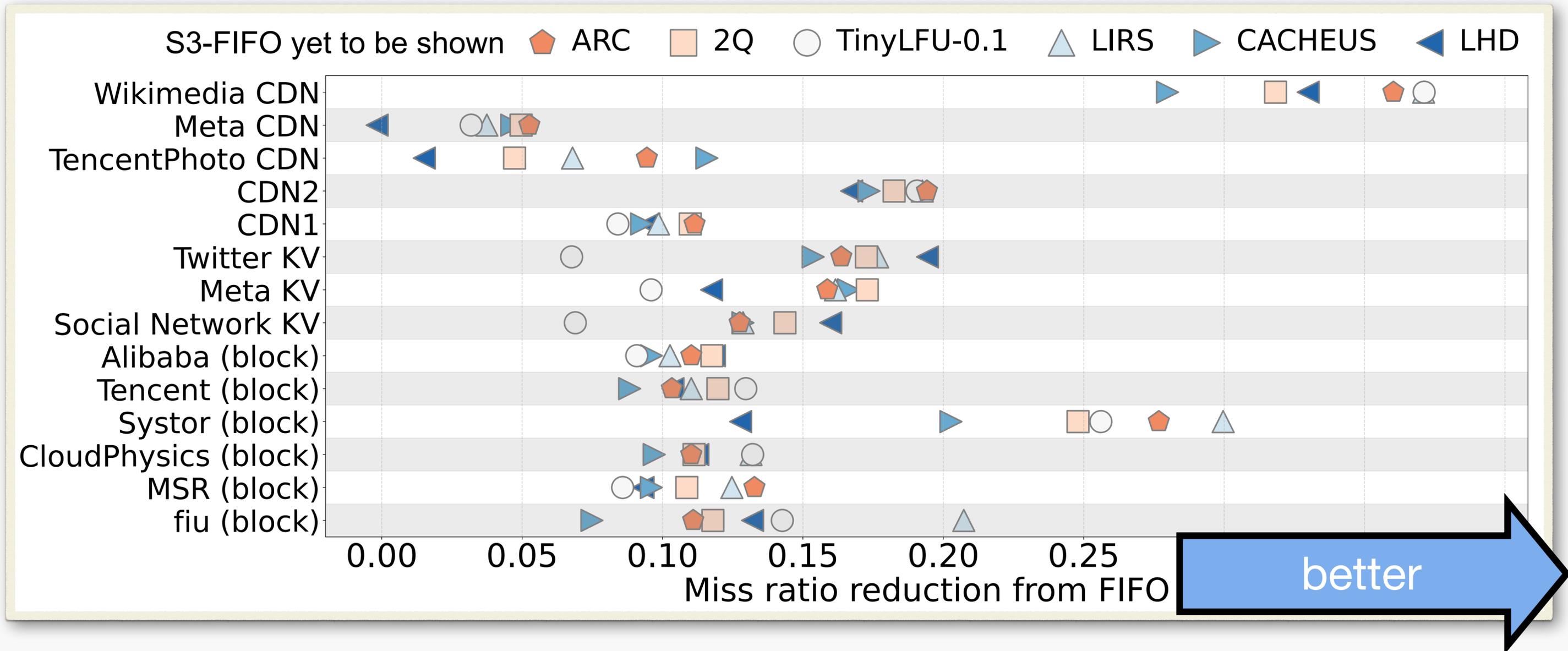
Miss ratio reduction distribution across all traces



More efficient than state-of-the-art algorithms, up to 72% lower miss ratio than LRU

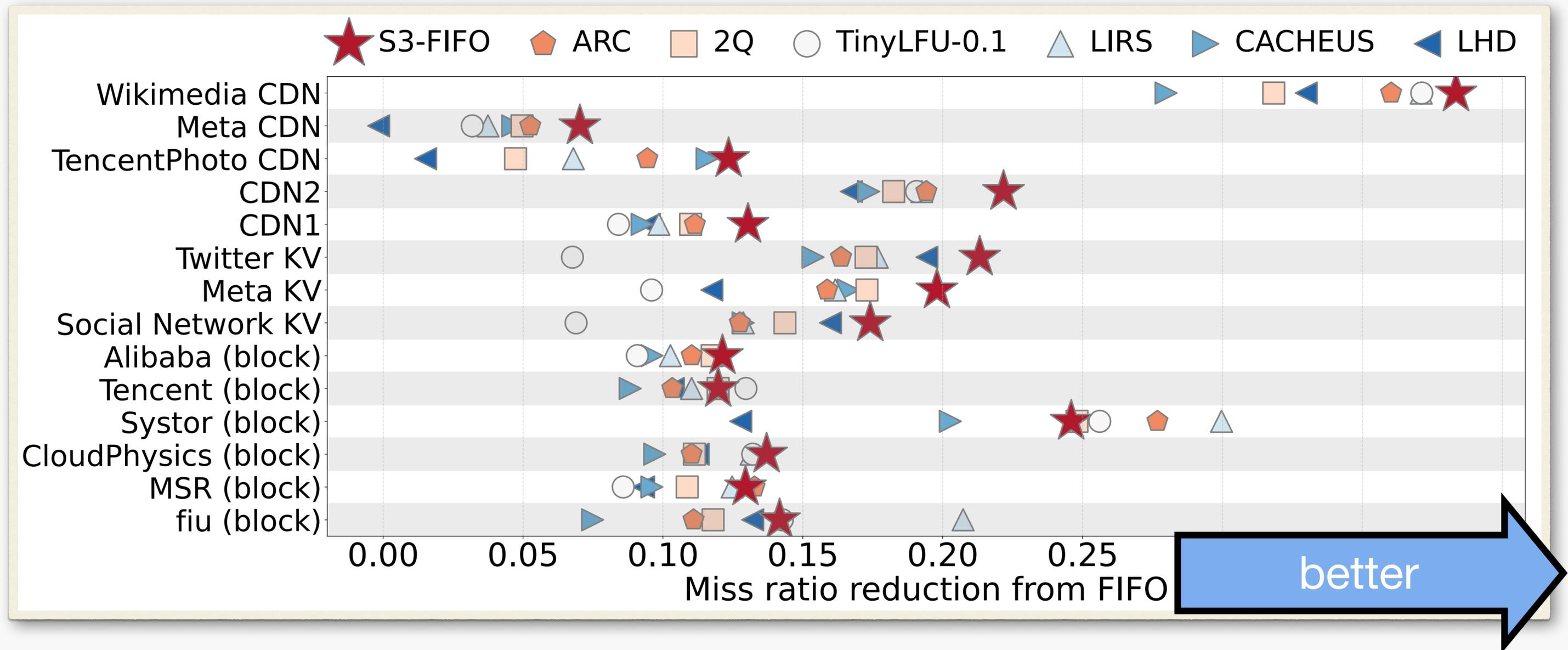
# Efficiency

Mean miss ratio reduction on each dataset



# Efficiency

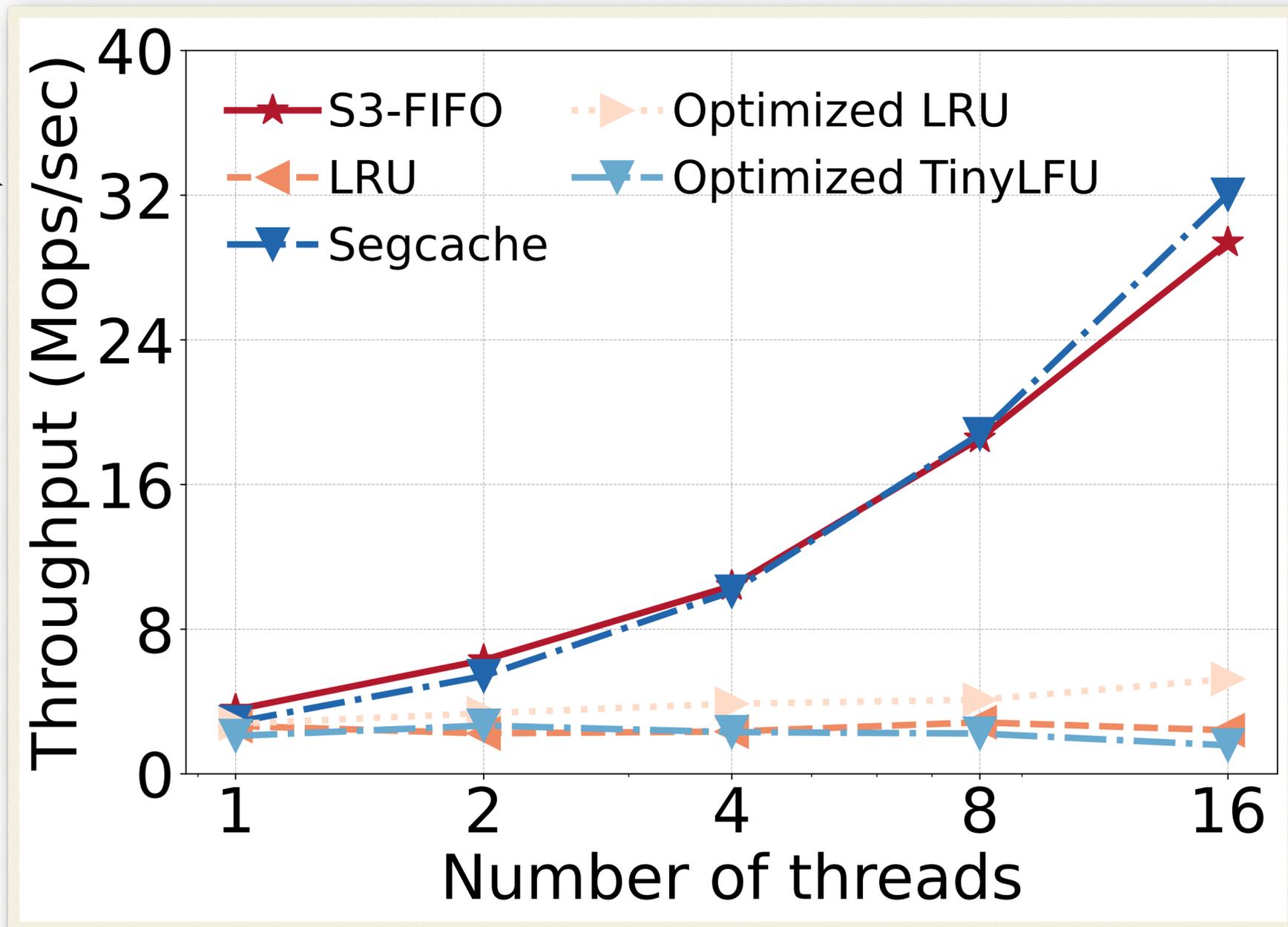
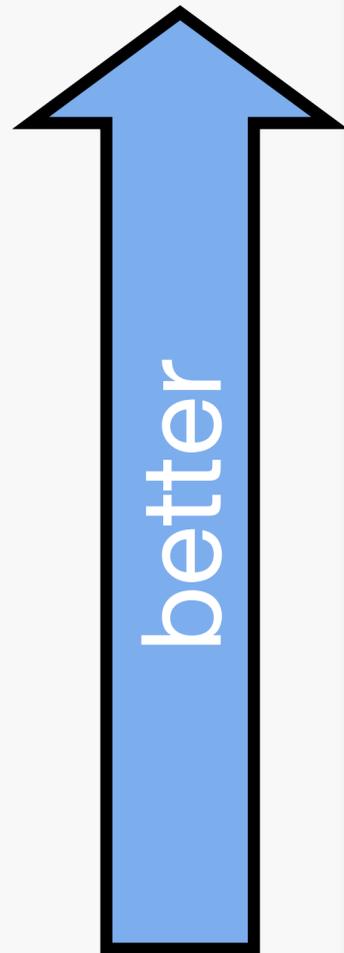
Mean miss ratio reduction on each dataset



- **efficient:** the best algorithm or is close to the best
- **robust:** the best on 10 of the 14 datasets

# Throughput and scalability

Zipf workloads

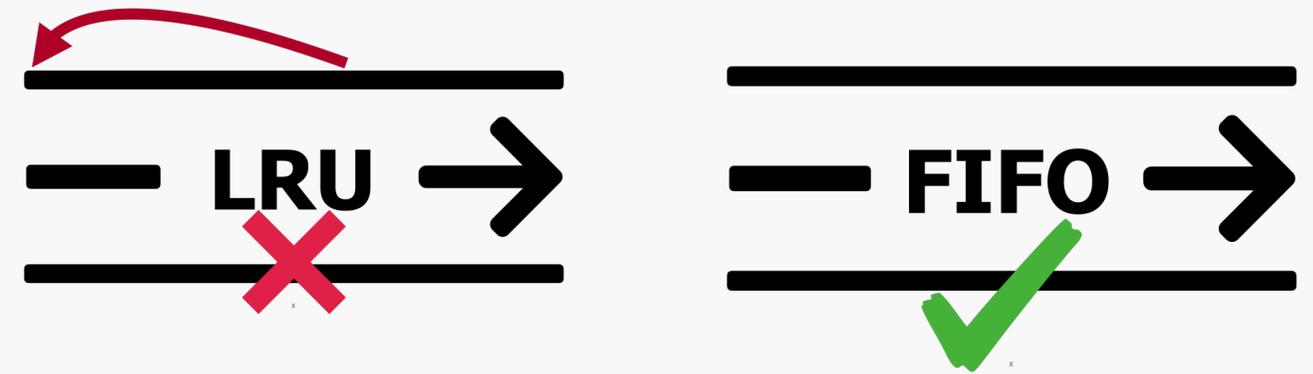


- the fastest on a single thread
- more scalable than optimized LRU, 6x higher throughput
- close to Segcache<sup>[NSDI'21]</sup>

# More in the paper

- Why S3-FIFO is effective
- Implication for flash cache
- Byte miss ratio results
- Impact of FIFO sizes
- What if we replace FIFO with LRU

# Takeaway



- **Cache workloads exhibit high one-hit-wonder ratio**
  - most objects in the cache are not re-accessed before being evicted
  - critical to remove the one-hit wonders early
- **S3-FIFO: simple, scalable caching with three static **FIFO** queues**
- reinsertion to keep popular objects, a small FIFO queue to *quickly* filter out one-hit wonders
- adoption
  - being evaluated at Google, VMWare, Cloudflare, Kuaishou, etc.
  - Python/C++/Rust version of S3-FIFO on GitHub implemented by external parties

[juncheny@cs.cmu.edu](mailto:juncheny@cs.cmu.edu)



<https://s3fifo.com>

<https://github.com/Thesys-lab/sosp23-s3fifo>

