Demystifying and Improving Lazy Promotion in Cache Eviction

Qinghan Chen[1]

Muhammad Haekal Muhyidin Al-Araby[2]

Ziyue Qiu[1]

Presenter

Zhuofan Chen[1]

Rashmi Vinayak[1]

Juncheng Yang[3]

[1] Carnegie Mellon University

Parallel Data Laboratory

[2] Sepuluh Nopember Institute of Technology

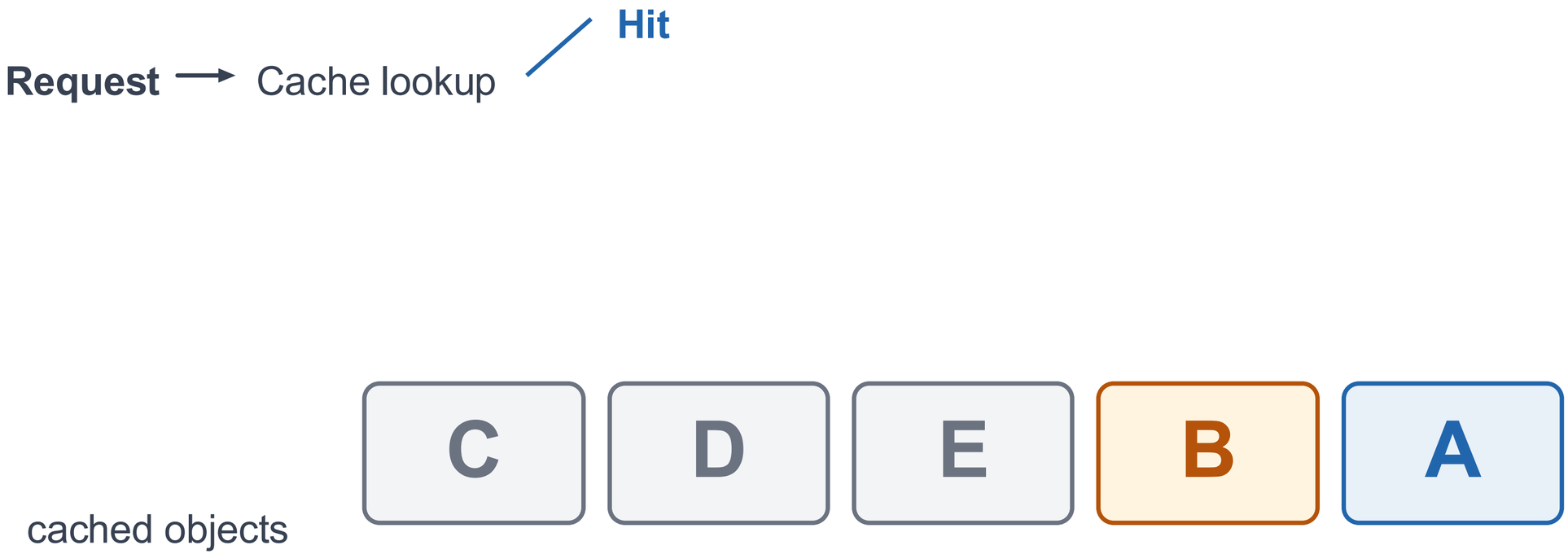
[3] Harvard University

LRU promotions limit scalability

Request → Cache lookup

cached objects

LRU promotions limit scalability

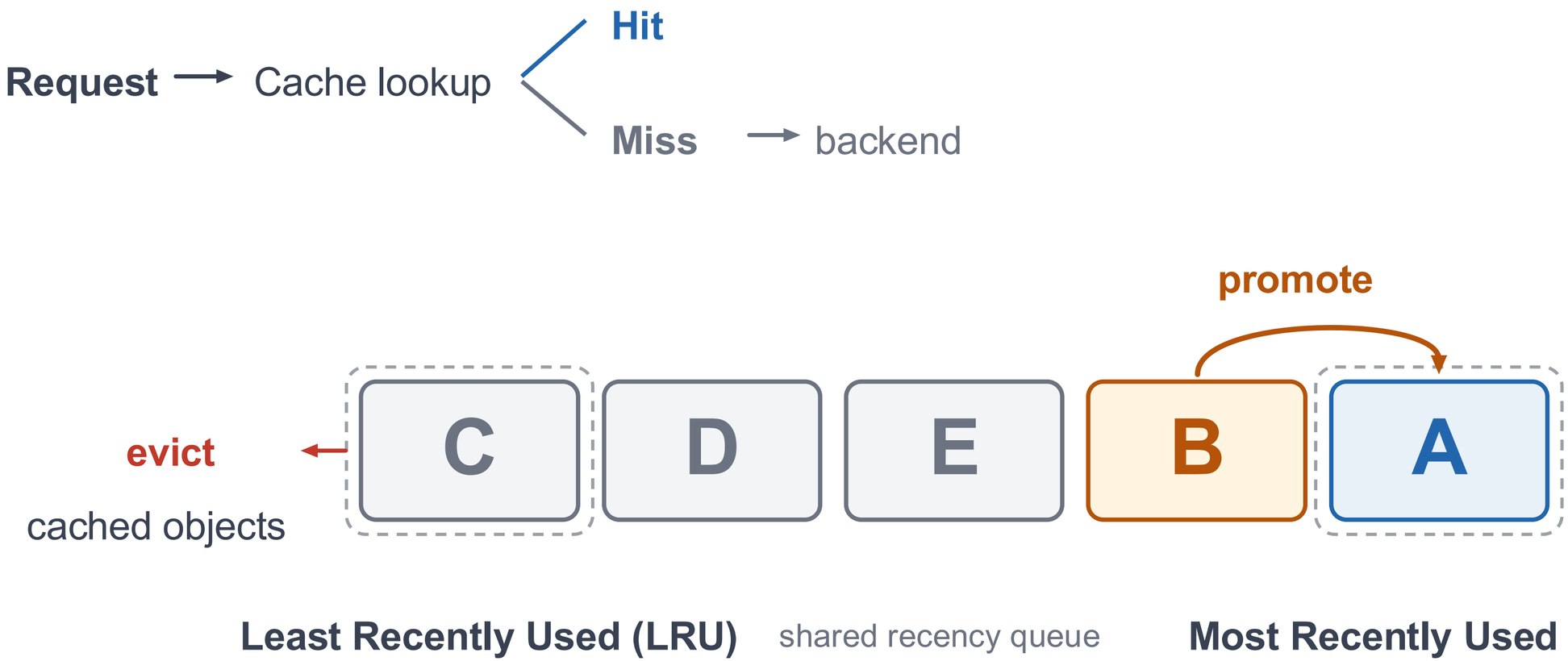


LRU promotions limit scalability

LRU promotions limit scalability

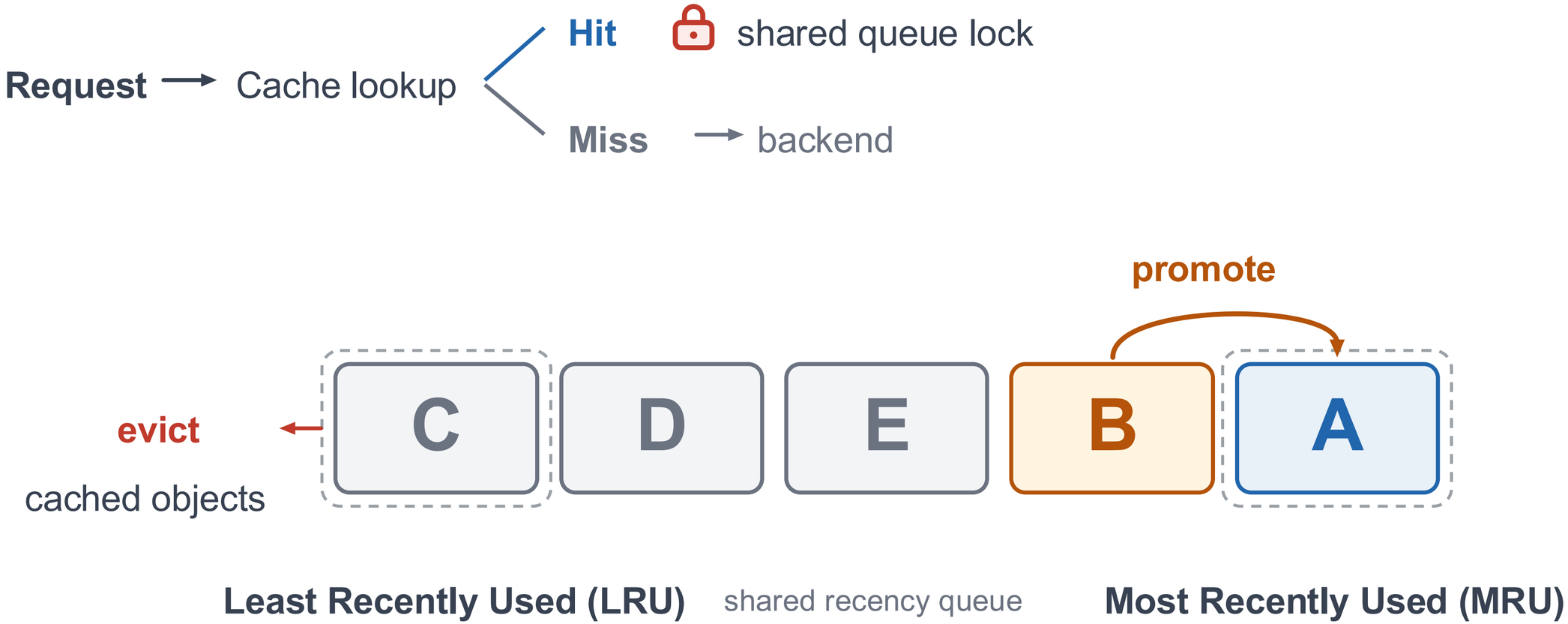
LRU promotions limit scalability

- On a hit, LRU moves the object to the Most Recently Used (MRU) slot.



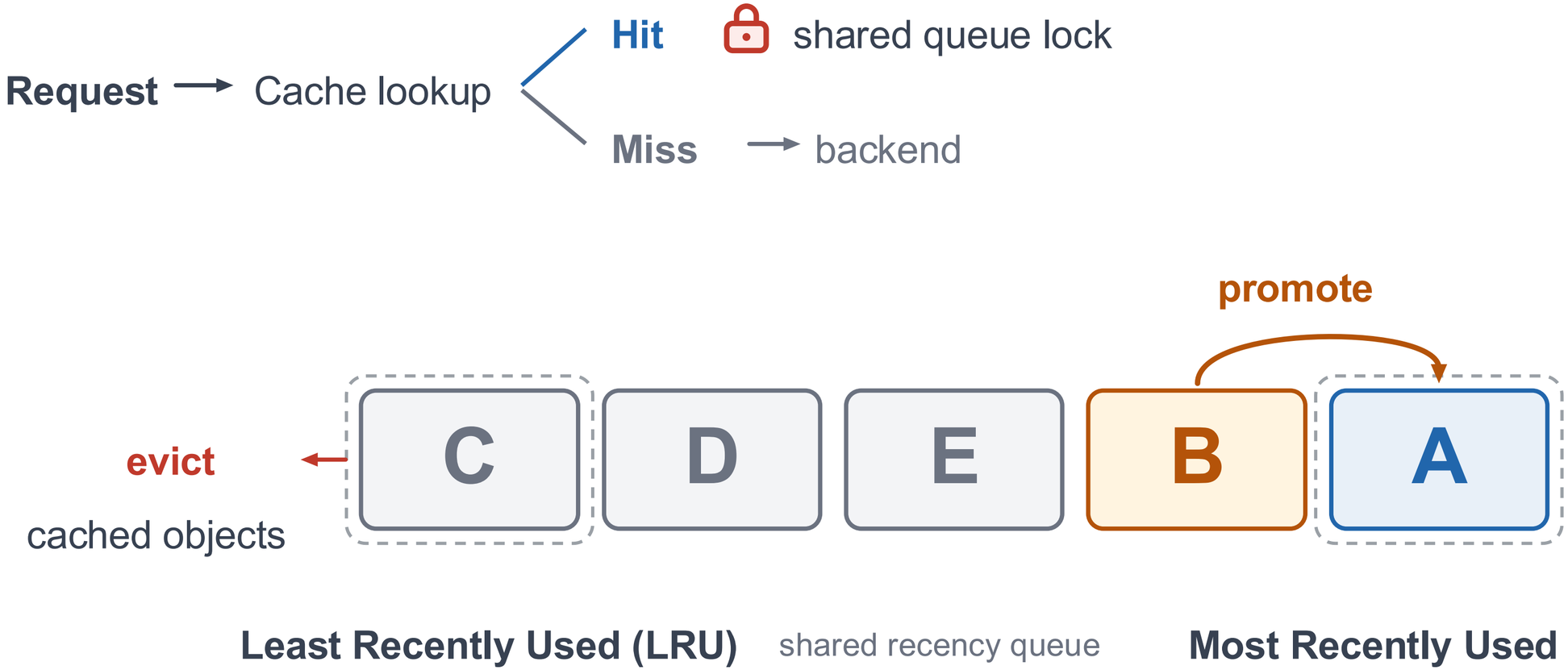
LRU promotions limit scalability

- On a hit, LRU moves the object to the Most Recently Used (MRU) slot.



LRU promotions limit scalability

- On a hit, LRU moves the object to the Most Recently Used (MRU) slot.
- Lazy promotion addresses this bottleneck by intentionally performing fewer promotions.

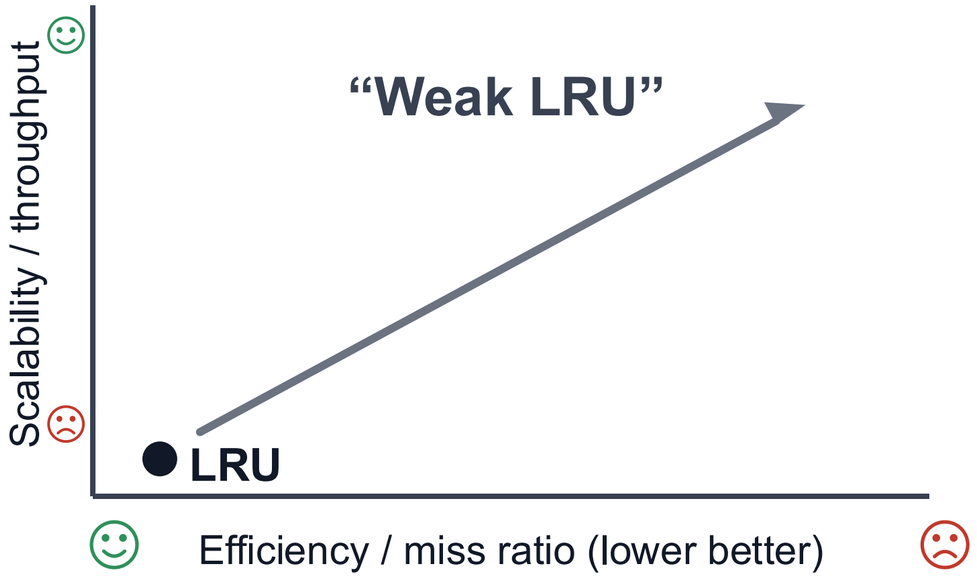


Overview: Lazy promotion = “Weak LRU”?

Conventional wisdom: fewer promotions trade miss ratio for throughput.

Overview: Lazy promotion = “Weak LRU”?

Conventional wisdom: fewer promotions trade miss ratio for throughput.

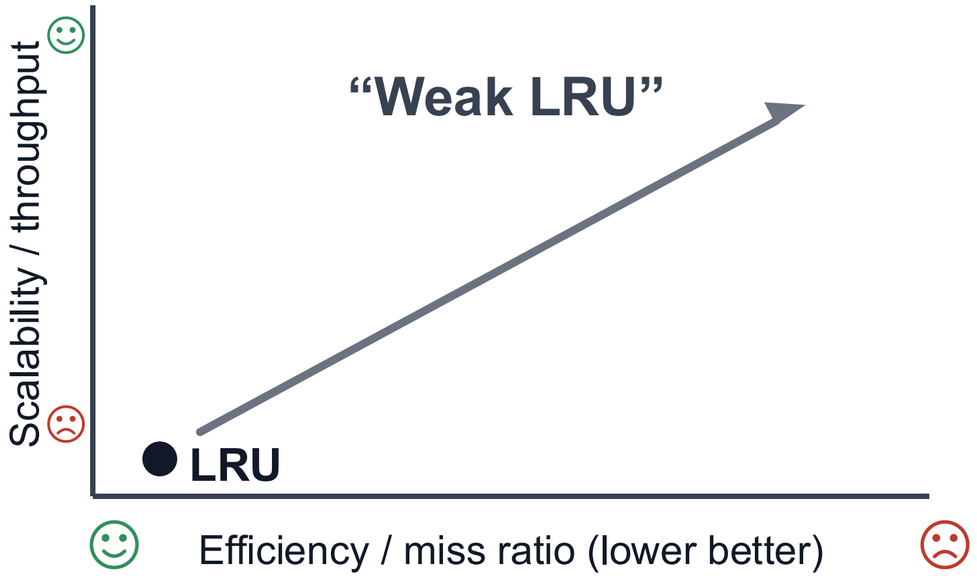


Overview: Lazy promotion = “Weak LRU”?

Conventional wisdom: fewer promotions trade miss ratio for throughput.

Production systems use five lazy promotion techniques, but no study compares them comprehensively.

HHVM, CacheLib | CliqueMap | Ristretto | Redis | RocksDB | PostgreSQL

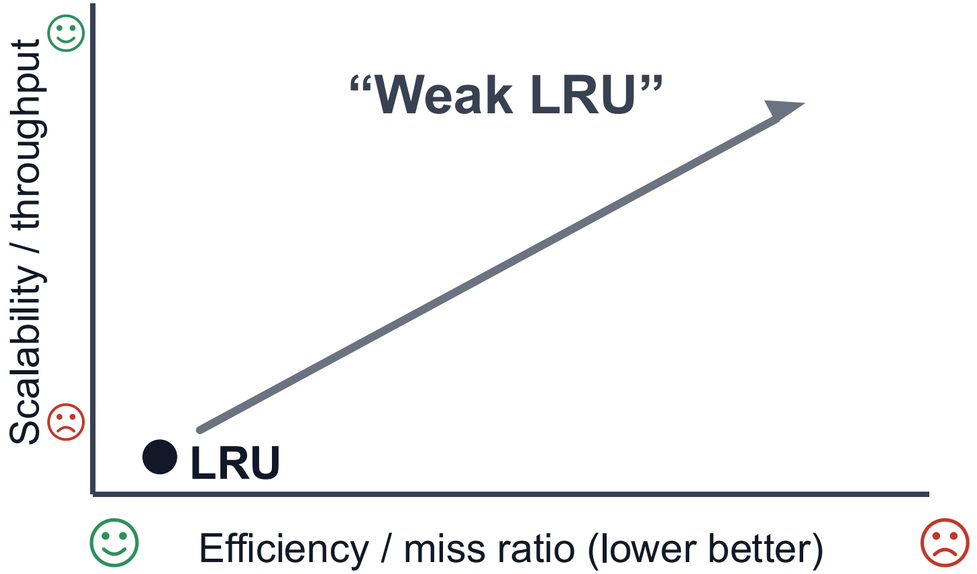


Overview: Lazy promotion = “Weak LRU”?

Conventional wisdom: fewer promotions trade miss ratio for throughput.

Production systems use five lazy promotion techniques, but no study compares them comprehensively.

HHVM, CacheLib | CliqueMap | Ristretto | Redis | RocksDB | PostgreSQL



Our study asks:

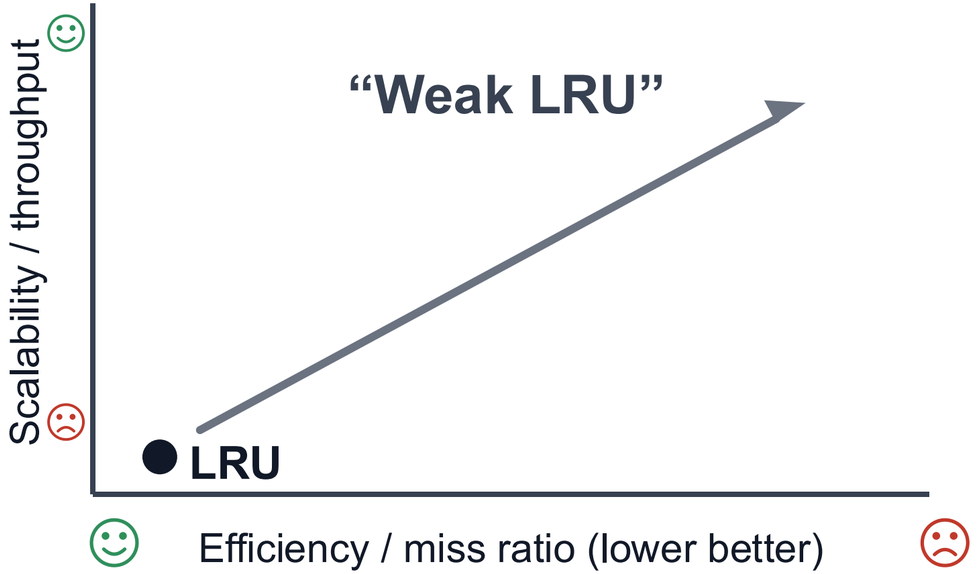
1. Are they really just weaker versions of LRU?

Overview: Lazy promotion = “Weak LRU”?

Conventional wisdom: fewer promotions trade miss ratio for throughput.

Production systems use five lazy promotion techniques, but no study compares them comprehensively.

HHVM, CacheLib | CliqueMap | Ristretto | Redis | RocksDB | PostgreSQL



Our study asks:

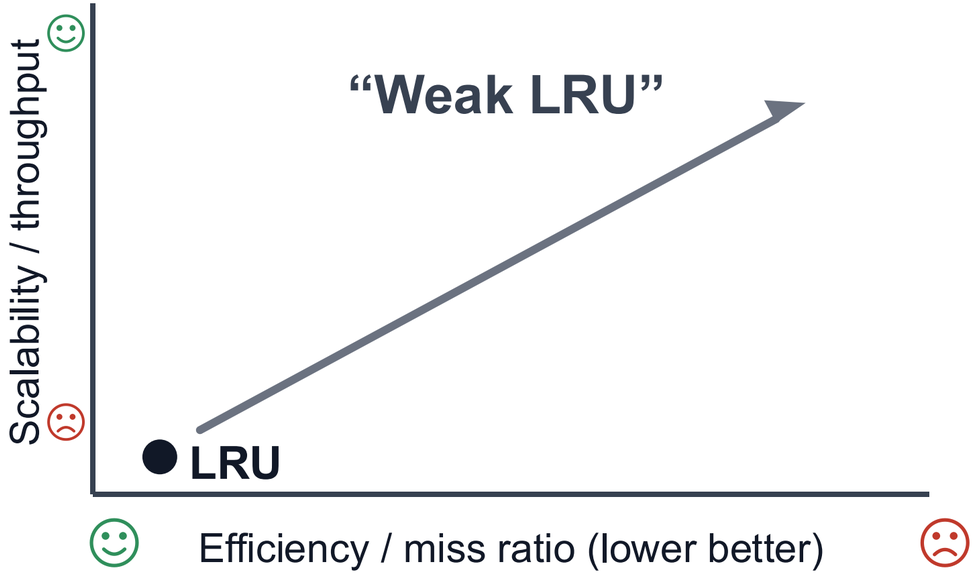
1. Are they really just weaker versions of LRU?
   No. Some techniques improve both axes.

Overview: Lazy promotion = “Weak LRU”?

Conventional wisdom: fewer promotions trade miss ratio for throughput.

Production systems use five lazy promotion techniques, but no study compares them comprehensively.

HHVM, CacheLib | CliqueMap | Ristretto | Redis | RocksDB | PostgreSQL



Our study asks:

1. Are they really just weaker versions of LRU?
   No. Some techniques improve both axes.

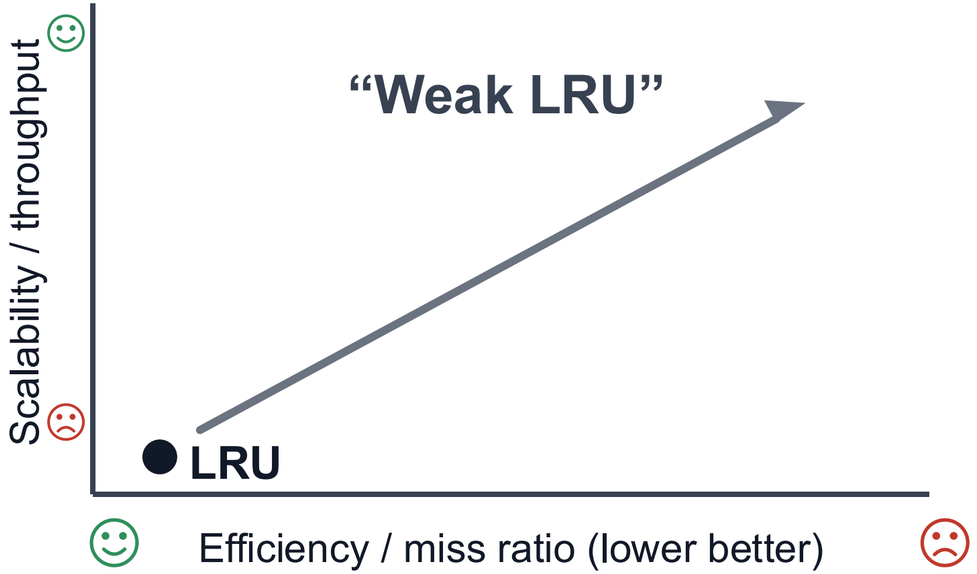
2. How effective is each technique?

Overview: Lazy promotion = “Weak LRU”?

Conventional wisdom: fewer promotions trade miss ratio for throughput.

Production systems use five lazy promotion techniques, but no study compares them comprehensively.

HHVM, CacheLib | CliqueMap | Ristretto | Redis | RocksDB | PostgreSQL



Our study asks:

1. Are they really just weaker versions of LRU?
   No. Some techniques improve both axes.

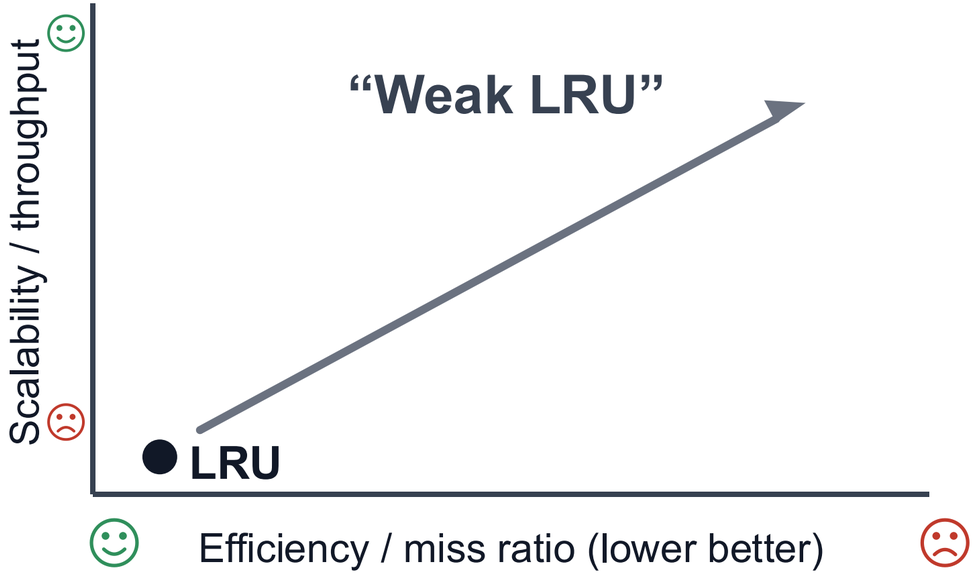
2. How effective is each technique?
   Very differently. Each has its own trade-off.

Overview: Lazy promotion = “Weak LRU”?

Conventional wisdom: fewer promotions trade miss ratio for throughput.

Production systems use five lazy promotion techniques, but no study compares them comprehensively.

HHVM, CacheLib | CliqueMap | Ristretto | Redis | RocksDB | PostgreSQL



Our study asks:

1. Are they really just weaker versions of LRU?
   No. Some techniques improve both axes.

2. How effective is each technique?
   Very differently. Each has its own trade-off.

3. Why do they differ, and can we use that to do better?

First comprehensive evaluation: 5 deployed techniques, 6,357 traces, 346B requests.

Probabilistic-LRU skips promotions, never waits

HHVM

Probabilistic-LRU skips promotions, never waits

HHVM

Core idea: On a hit, *promote* now or *skip*.

Probabilistic-LRU skips promotions, never waits

HHVM

Core idea: On a hit, *promote* now or *skip*.

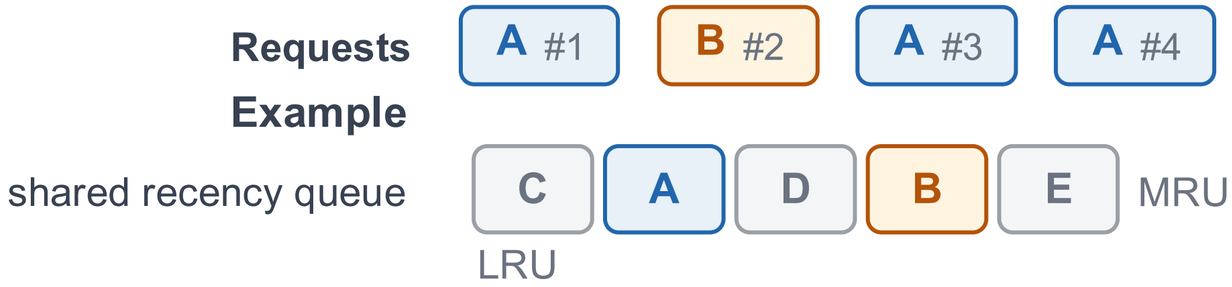
Rule: Promote only if the *try-lock* succeeds.

Probabilistic-LRU skips promotions, never waits

HHVM

Core idea: On a hit, *promote* now or *skip*.

Rule: Promote only if the *try-lock* succeeds.

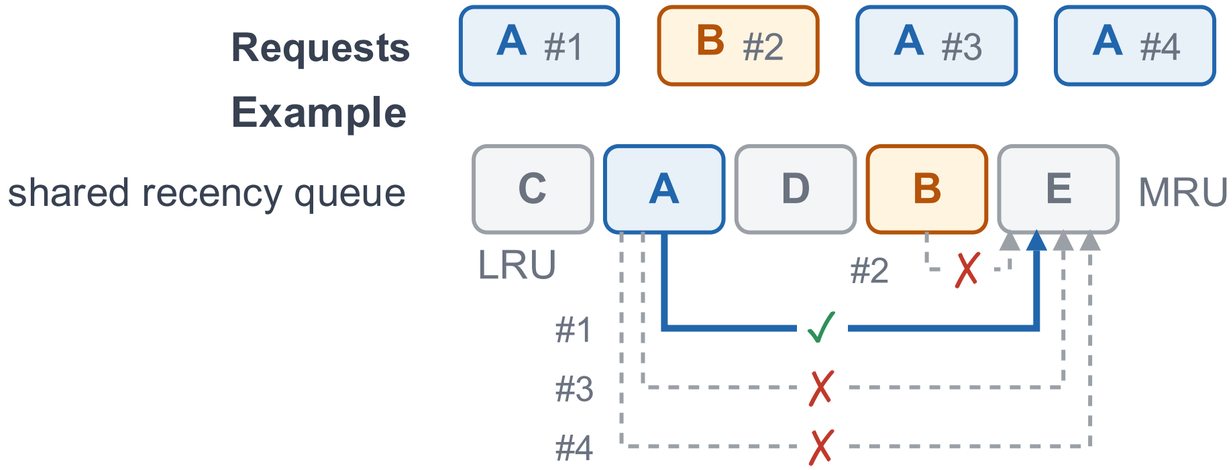


Probabilistic-LRU skips promotions, never waits

HHVM

Core idea: On a hit, *promote* now or *skip*.

Rule: Promote only if the *try-lock* succeeds.

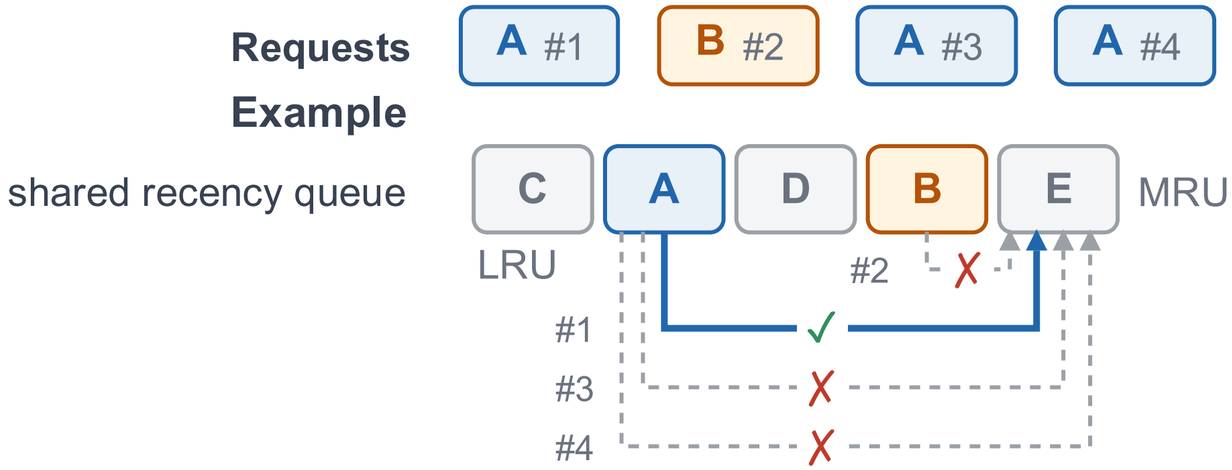


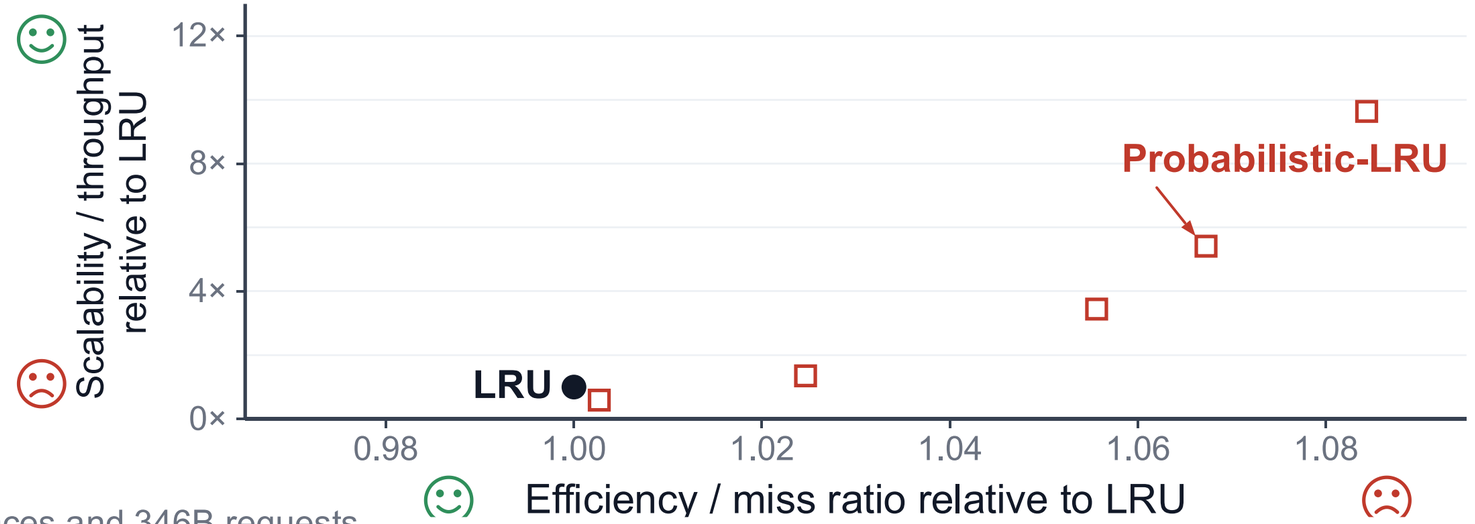
Probabilistic-LRU skips promotions, never waits

HHVM

Core idea: On a hit, *promote* now or *skip*.

Rule: Promote only if the *try-lock* succeeds.





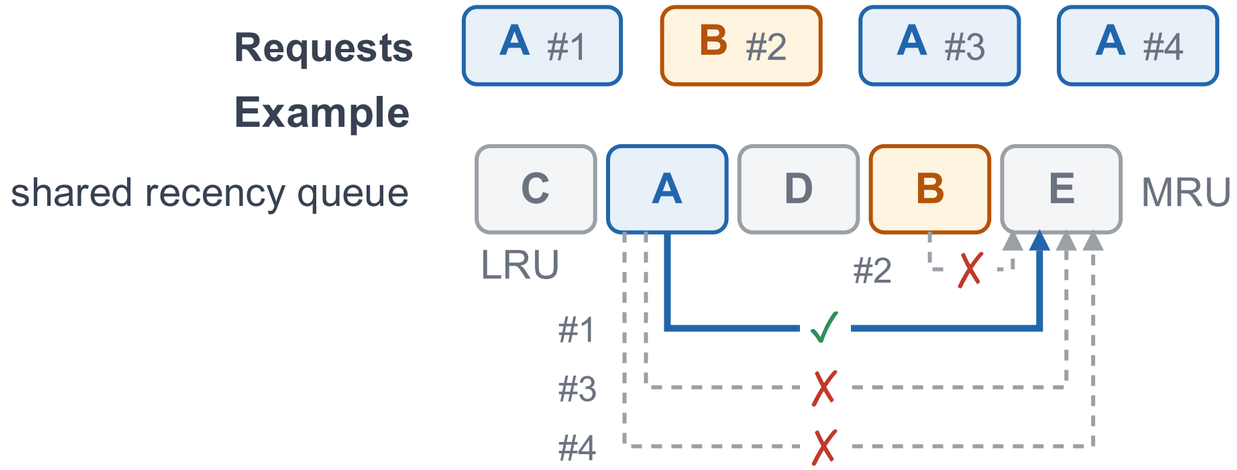
Evaluated on 6,357 traces and 346B requests

Probabilistic-LRU skips promotions, never waits

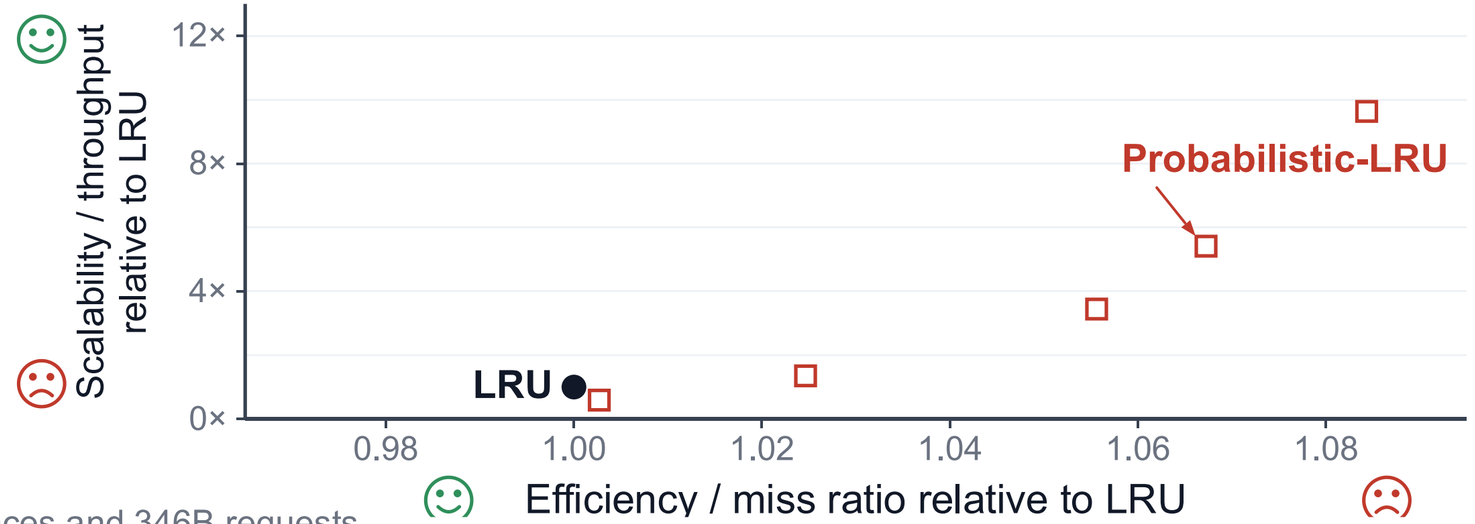
HHVM

Core idea: On a hit, *promote* now or *skip*.

Rule: Promote only if the *try-lock* succeeds.



Takeaway: Try-lock can skip useful promotions, so miss ratio rises quickly.



Evaluated on 6,357 traces and 346B requests

Batch-LRU amortizes one lock across many hits

CliqueMap Ristretto

Requests
Example

Evaluated on 6,357 traces and 346B requests

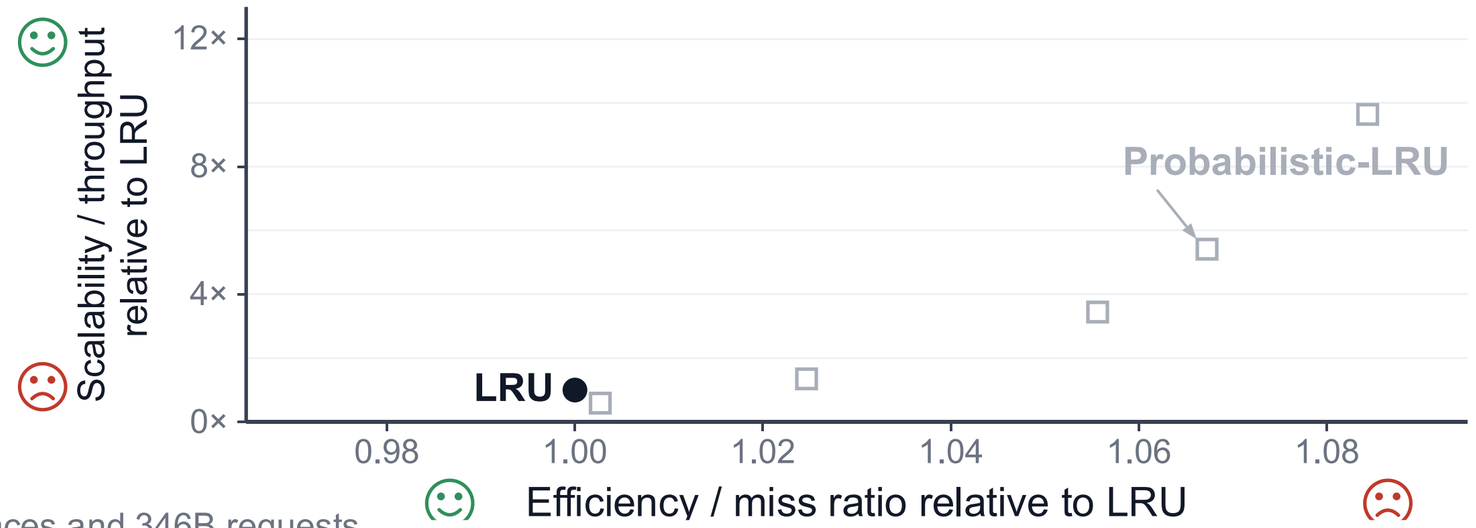
Batch-LRU amortizes one lock across many hits

CliqueMap Ristretto

Requests A #1 | B #2 | A #3 | A #4

Example

Core idea: Process many promotions under *one lock*.



Evaluated on 6,357 traces and 346B requests

Batch-LRU amortizes one lock across many hits

CliqueMap Ristretto

Requests A #1 B #2 A #3 A #4

Example

Core idea: Process many promotions under *one lock*.

Rule: Promote each object *at most once per batch*.

Evaluated on 6,357 traces and 346B requests

Batch-LRU amortizes one lock across many hits

CliqueMap Ristretto

Core idea: Process many promotions under *one lock*.

Rule: Promote each object *at most once per batch*.

Evaluated on 6,357 traces and 346B requests

Batch-LRU amortizes one lock across many hits

CliqueMap Ristretto

Core idea: Process many promotions under *one lock*.

Rule: Promote each object *at most once per batch*.

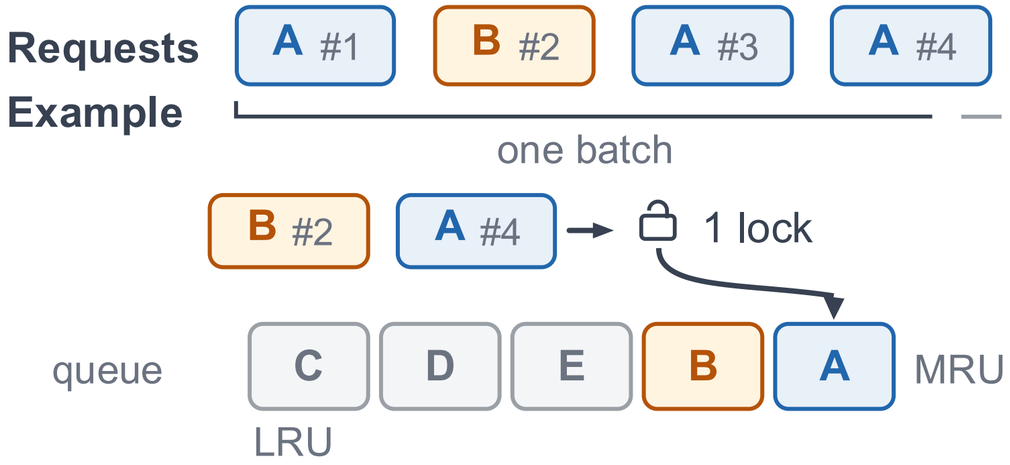
Evaluated on 6,357 traces and 346B requests

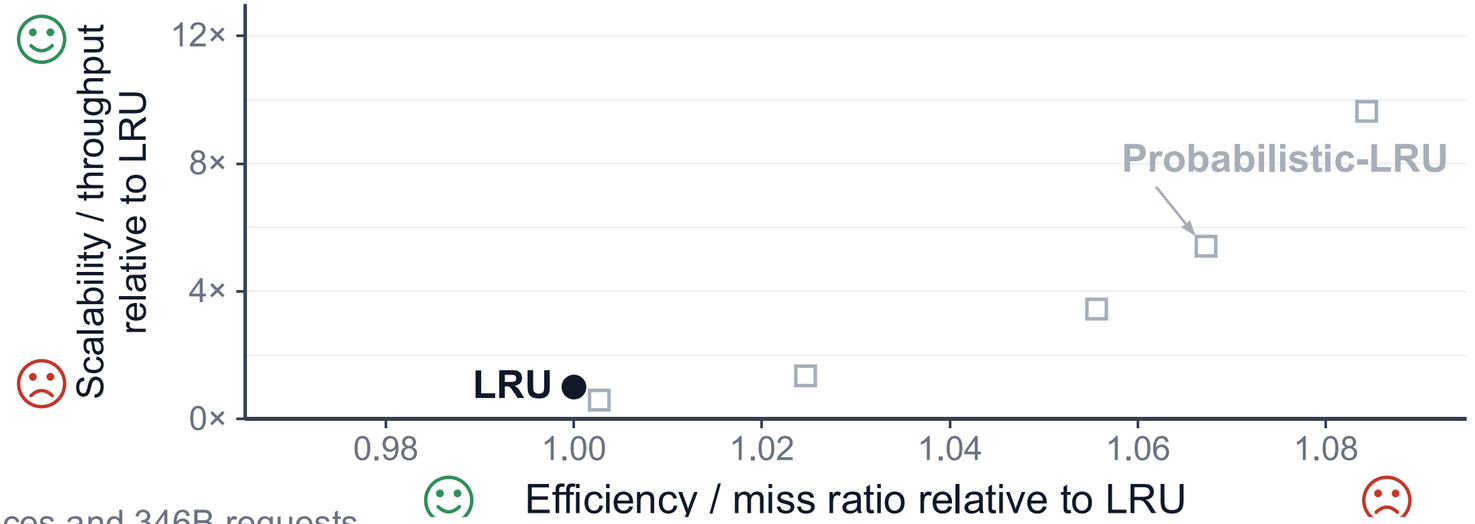
Batch-LRU amortizes one lock across many hits

CliqueMap Ristretto

Core idea: Process many promotions under *one lock*.

Rule: Promote each object *at most once per batch*.





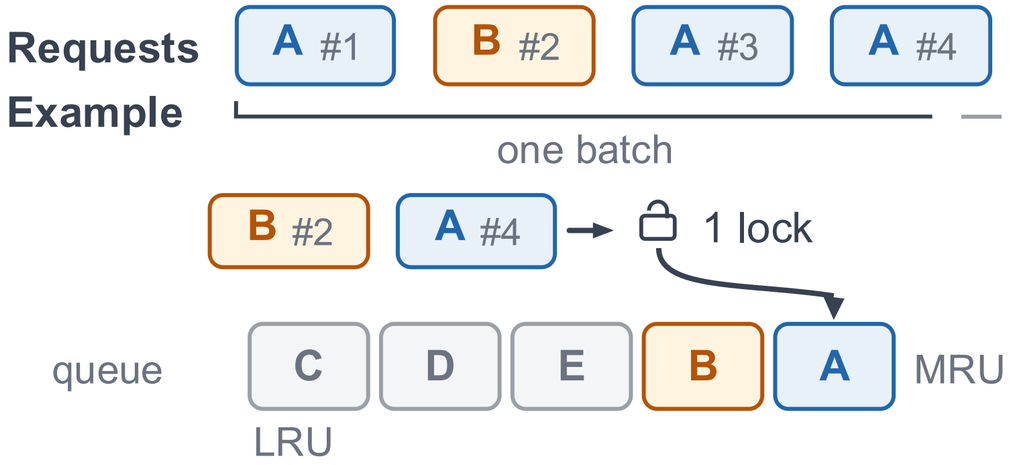
Evaluated on 6,357 traces and 346B requests

Batch-LRU amortizes one lock across many hits

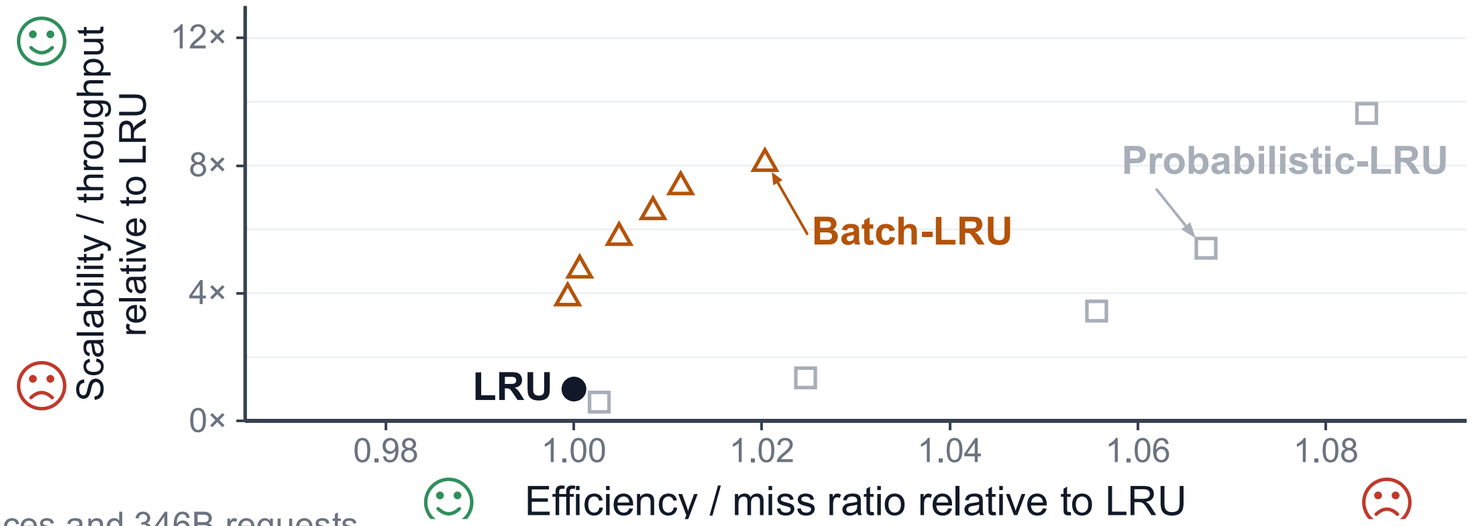
CliqueMap Ristretto

Core idea: Process many promotions under *one lock*.

Rule: Promote each object *at most once per batch*.



Takeaway: Batching creates opportunities to eliminate duplicate promotions.



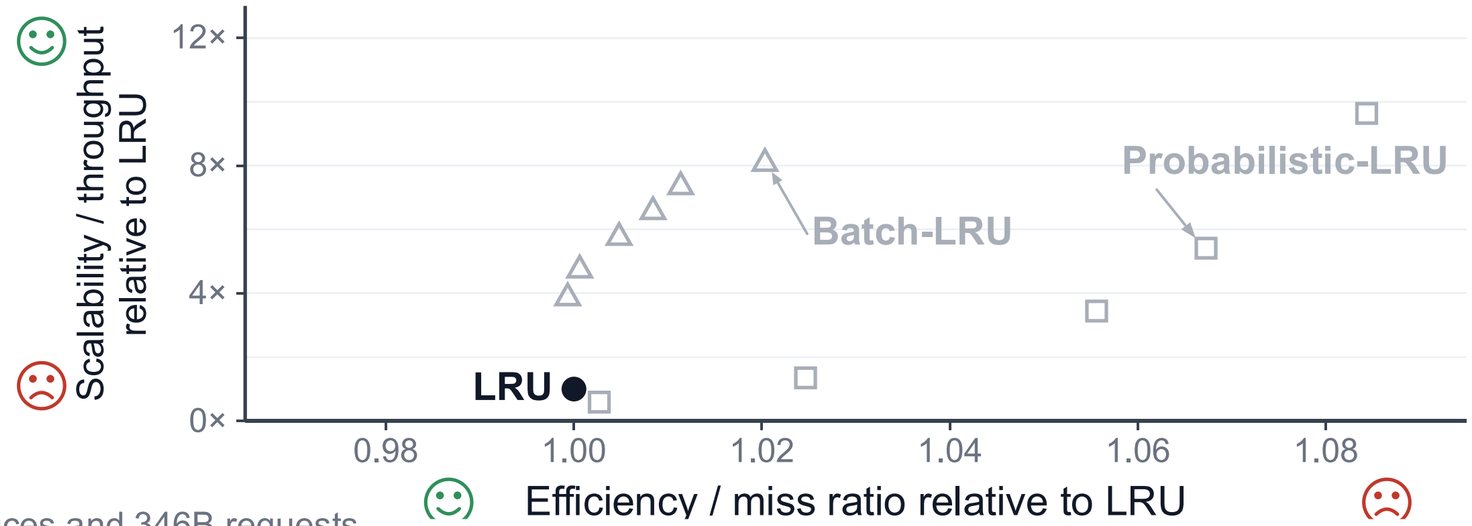
Evaluated on 6,357 traces and 346B requests

Random-LRU approximates LRU without a queue

Redis

Requests A #1 | B #2 | A #3 | A #4

Example



Evaluated on 6,357 traces and 346B requests

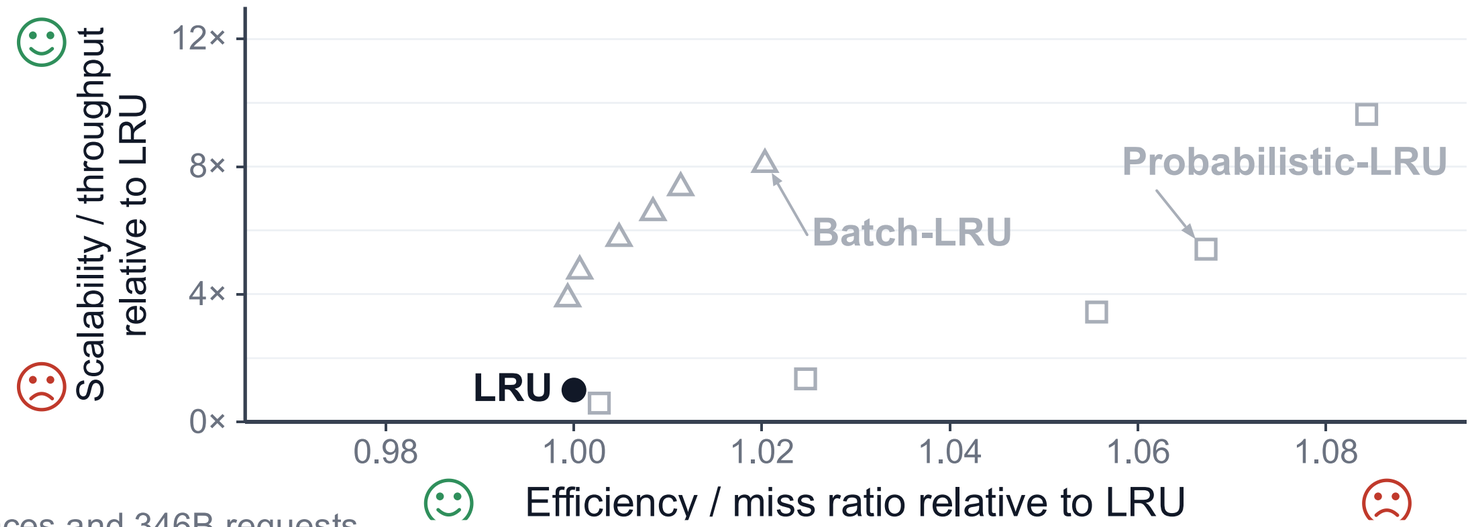
Random-LRU approximates LRU without a queue

Redis

Requests A #1 B #2 A #3 A #4

Example

Core idea: Approximate LRU *without a shared recency queue*.



Evaluated on 6,357 traces and 346B requests

Random-LRU approximates LRU without a queue

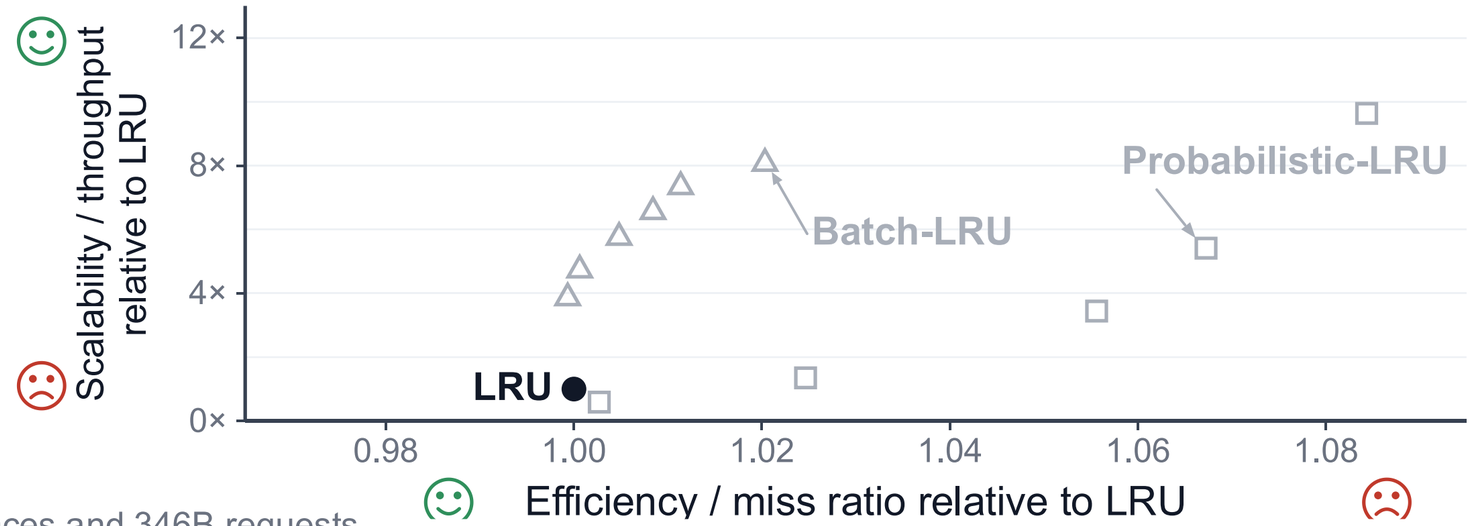
Redis

Requests: A #1 | B #2 | A #3 | A #4

Example

Core idea: Approximate LRU *without a shared recency queue*.

Rule: Store last-access *timestamps*; evict the *oldest sampled object*.



Evaluated on 6,357 traces and 346B requests

Random-LRU approximates LRU without a queue

Redis

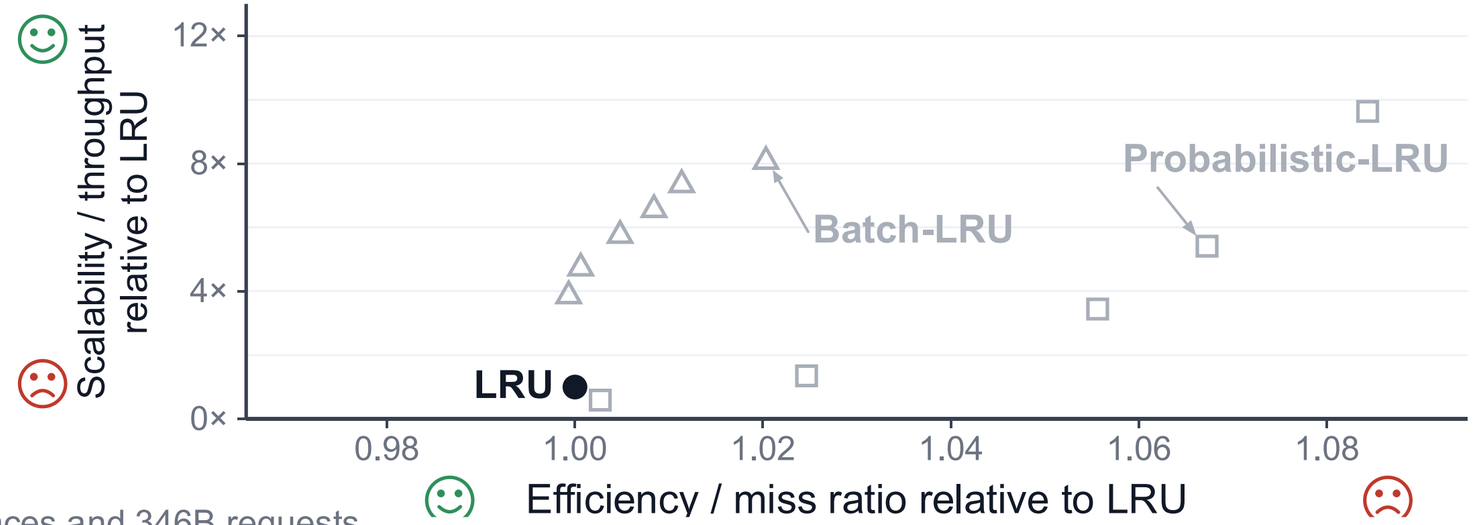
Requests A #1 B #2 A #3 A #4

Example

Core idea: Approximate LRU *without a shared recency queue*.

Rule: Store last-access *timestamps*; evict the *oldest sampled object*.

no shared queue



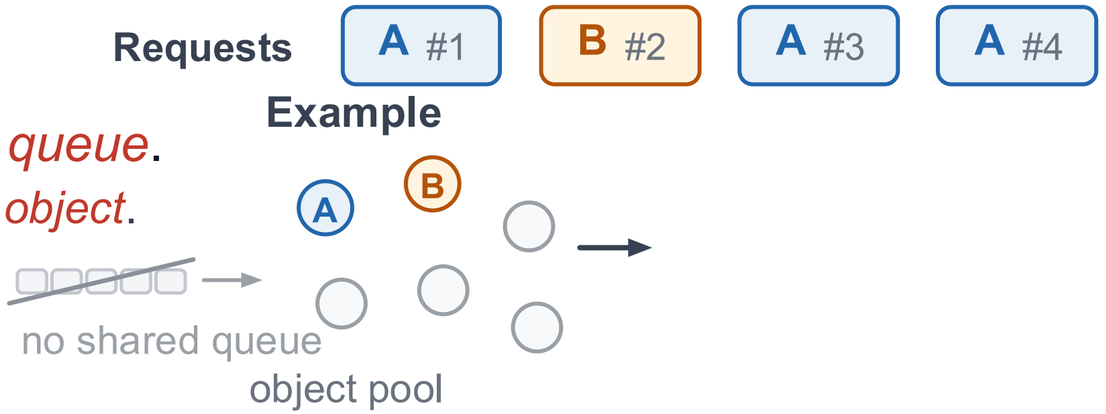
Evaluated on 6,357 traces and 346B requests

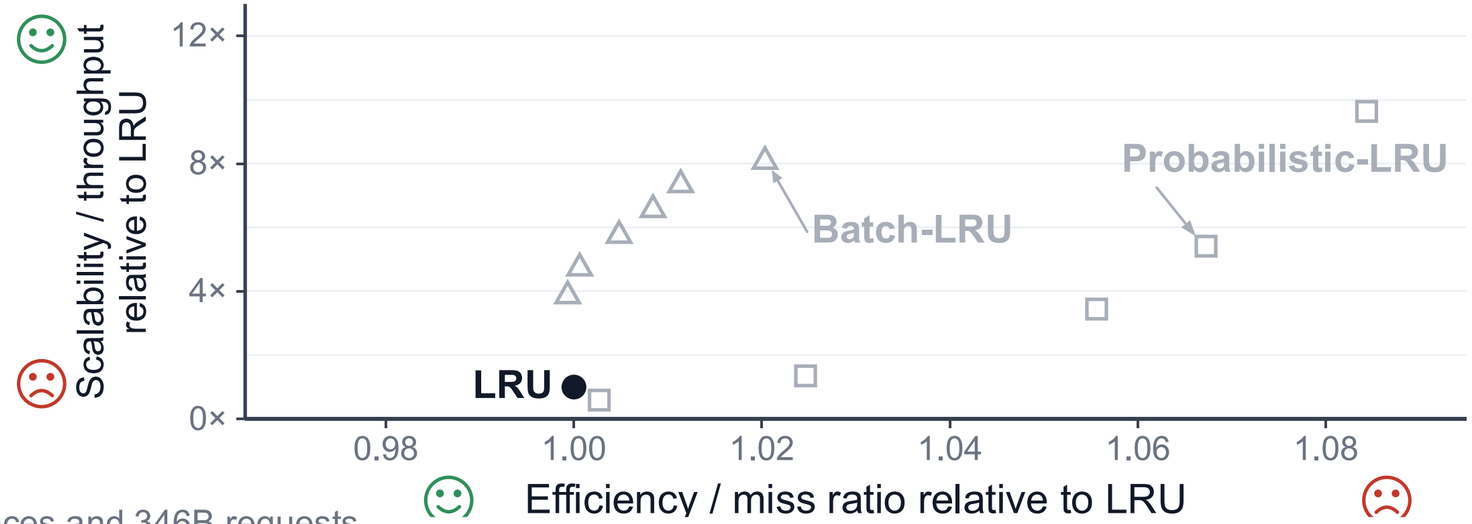
Random-LRU approximates LRU without a queue

Redis

Core idea: Approximate LRU *without a shared recency queue*.

Rule: Store last-access *timestamps*; evict the *oldest sampled object*.





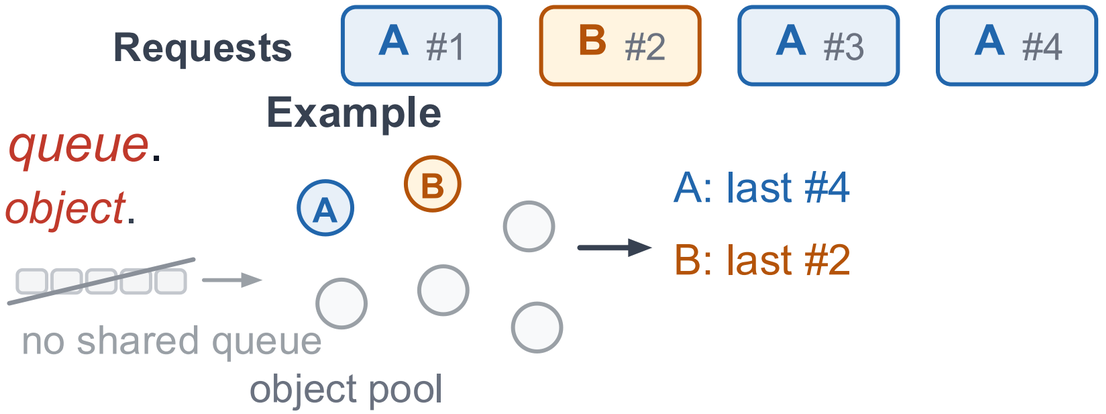
Evaluated on 6,357 traces and 346B requests

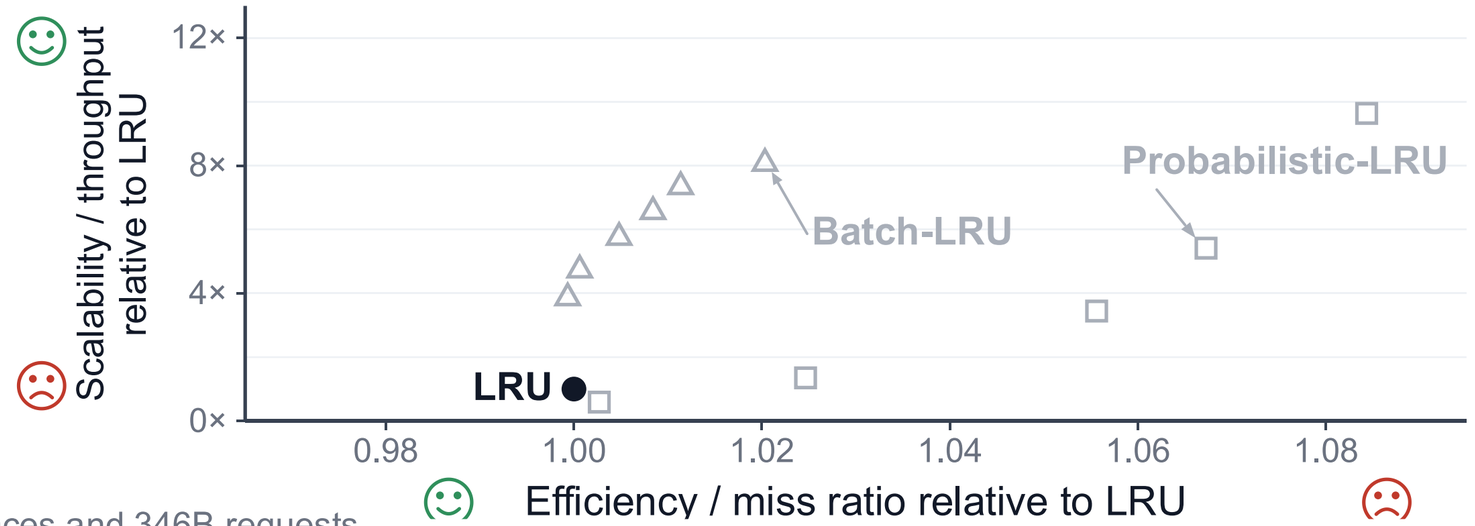
Random-LRU approximates LRU without a queue

Redis

Core idea: Approximate LRU *without a shared recency queue*.

Rule: Store last-access *timestamps*; evict the *oldest sampled object*.





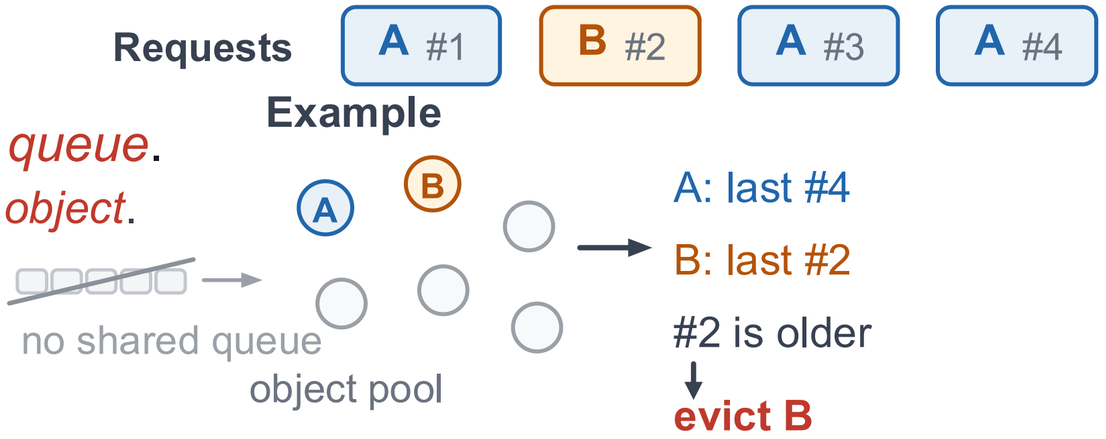
Evaluated on 6,357 traces and 346B requests

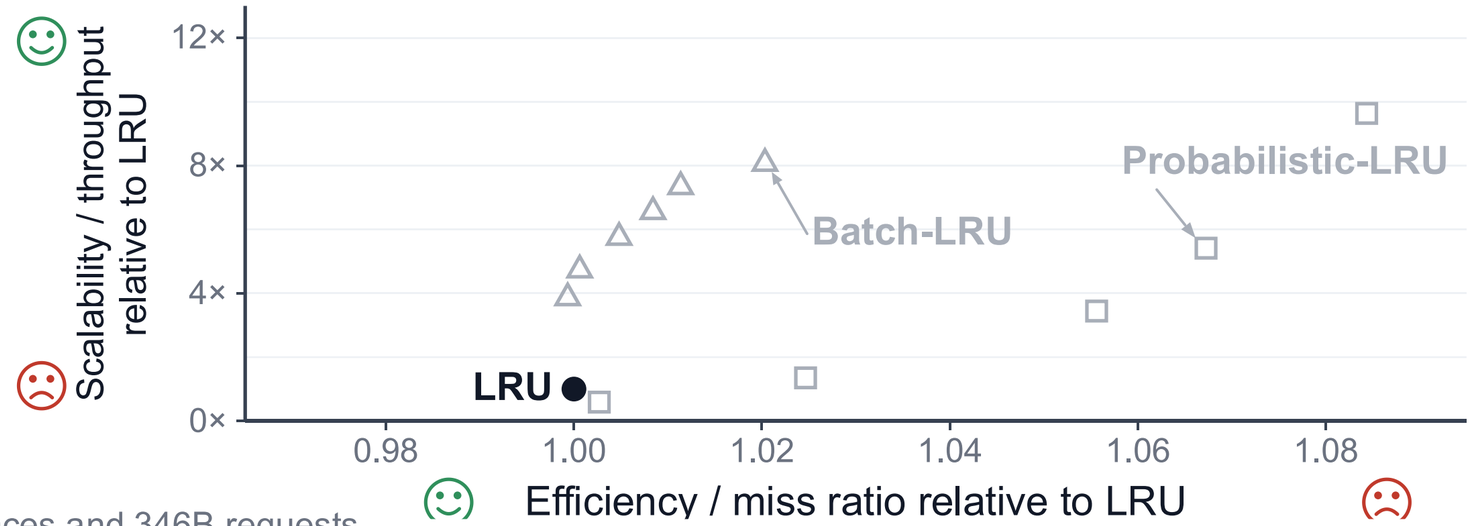
Random-LRU approximates LRU without a queue

Redis

Core idea: Approximate LRU *without a shared recency queue*.

Rule: Store last-access *timestamps*; evict the *oldest sampled object*.





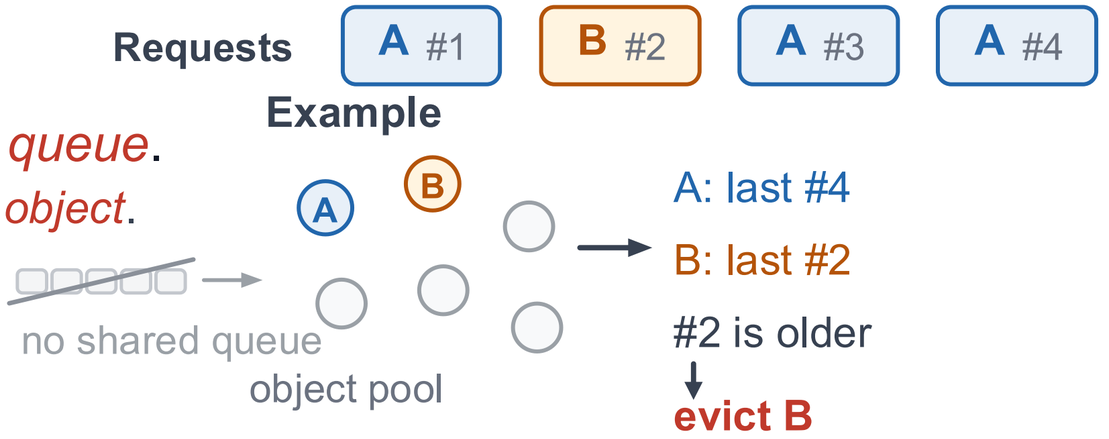
Evaluated on 6,357 traces and 346B requests

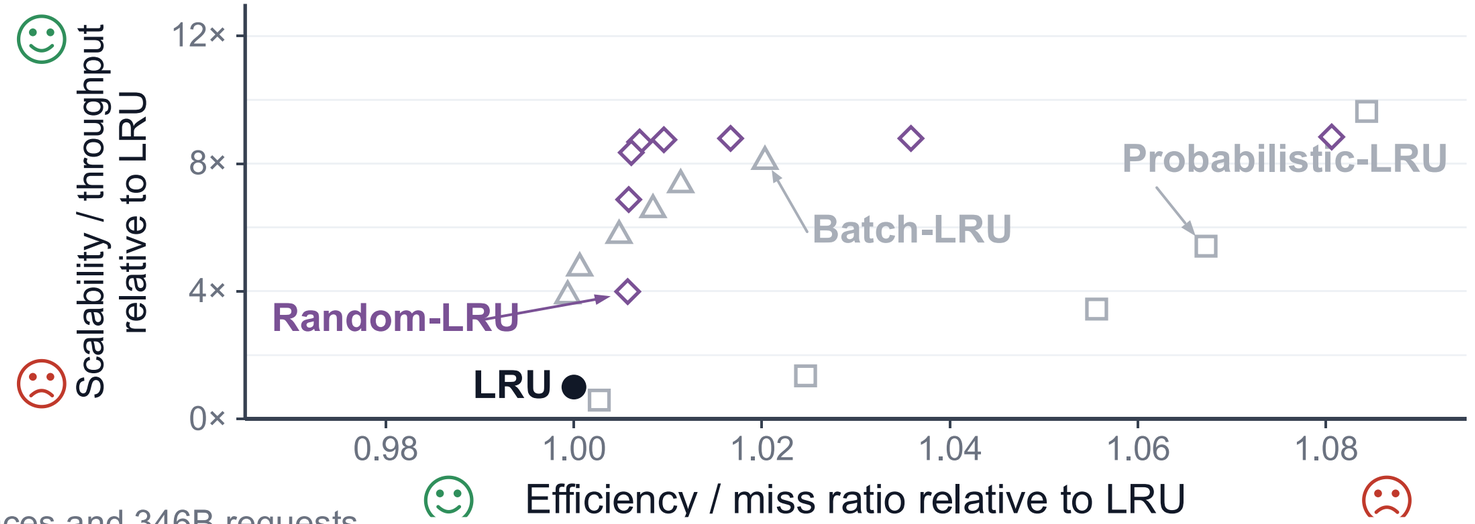
Random-LRU approximates LRU without a queue

Redis

Core idea: Approximate LRU *without a shared recency queue*.

Rule: Store last-access *timestamps*; evict the *oldest sampled object*.





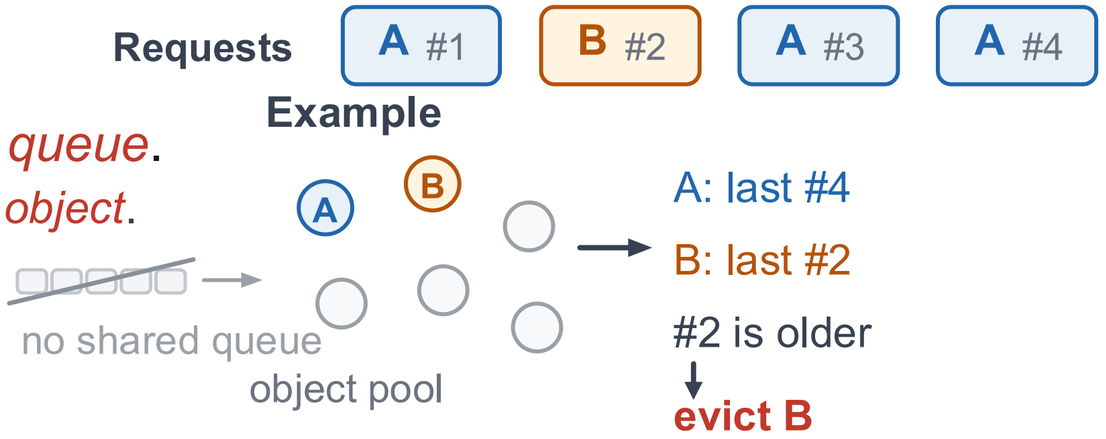
Evaluated on 6,357 traces and 346B requests

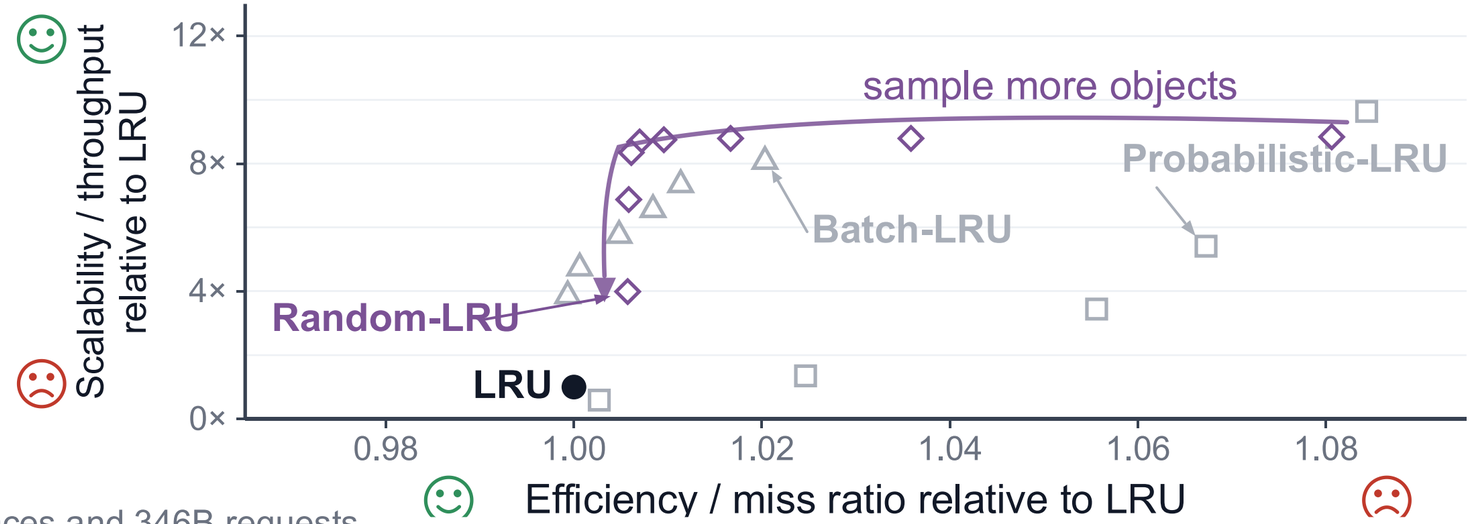
Random-LRU approximates LRU without a queue

Redis

Core idea: Approximate LRU *without a shared recency queue*.

Rule: Store last-access *timestamps*; evict the *oldest sampled object*.





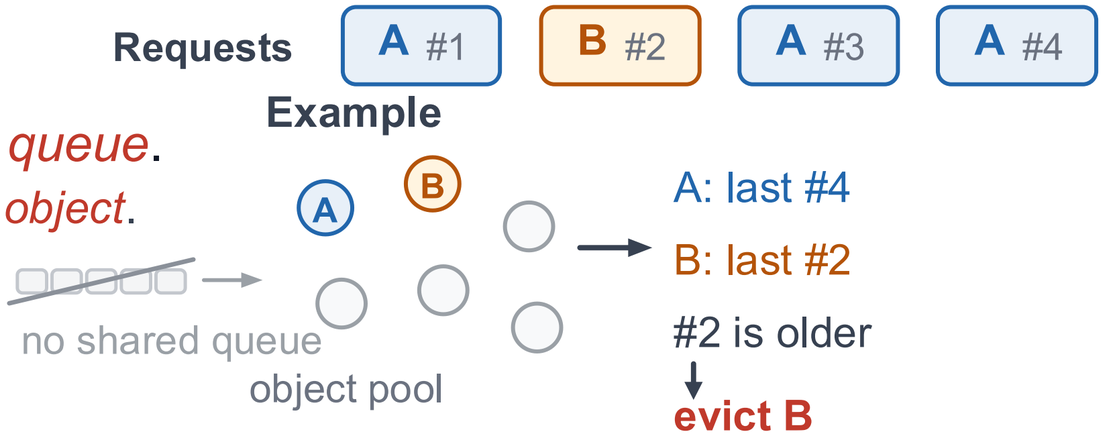
Evaluated on 6,357 traces and 346B requests

Random-LRU approximates LRU without a queue

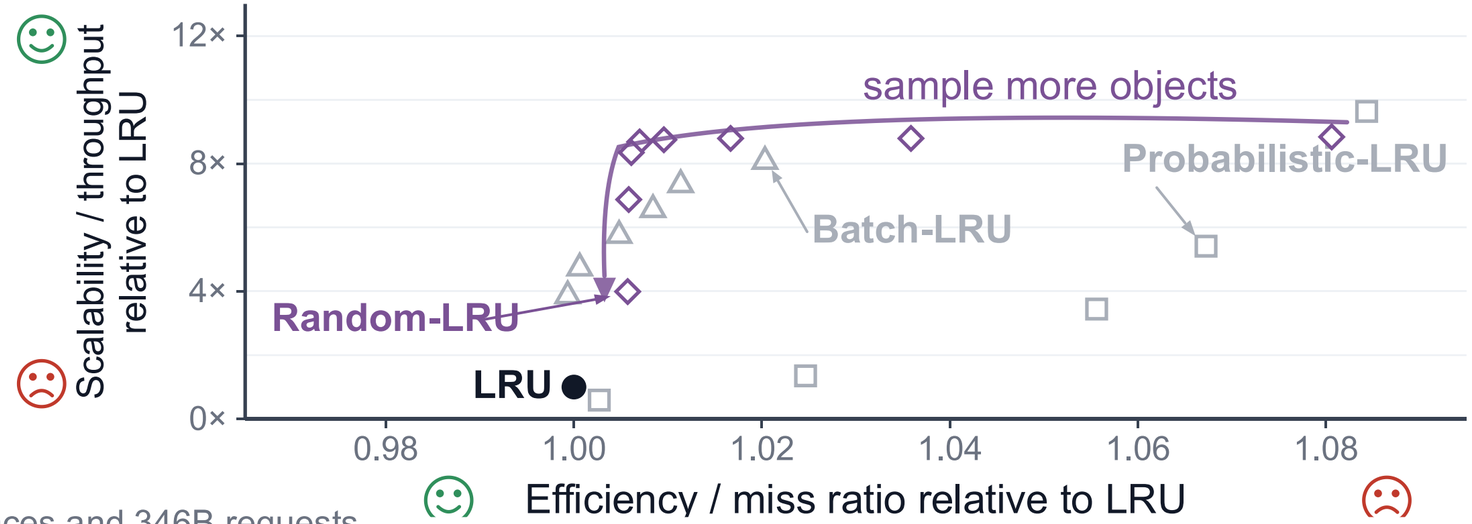
Redis

Core idea: Approximate LRU *without a shared recency queue*.

Rule: Store last-access *timestamps*; evict the *oldest sampled object*.



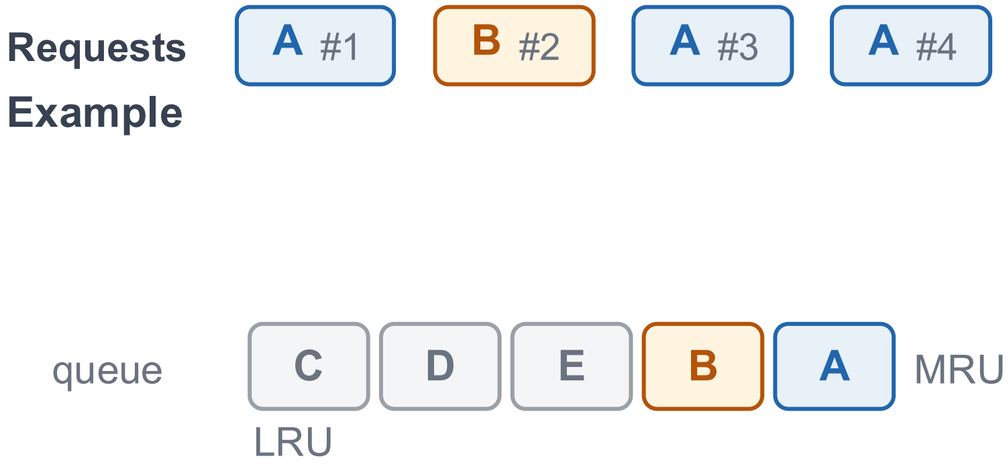
Takeaway: Larger samples approach LRU's miss ratio but lose throughput.

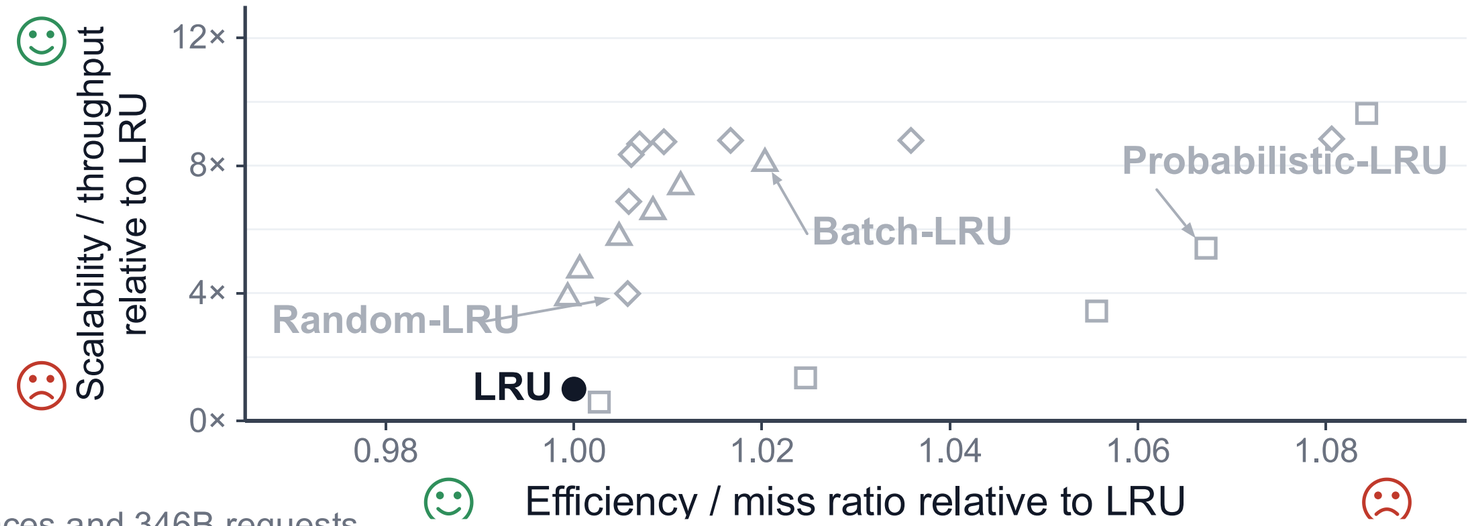


Evaluated on 6,357 traces and 346B requests

Delay-LRU filters repeated promotions in a window

CacheLib



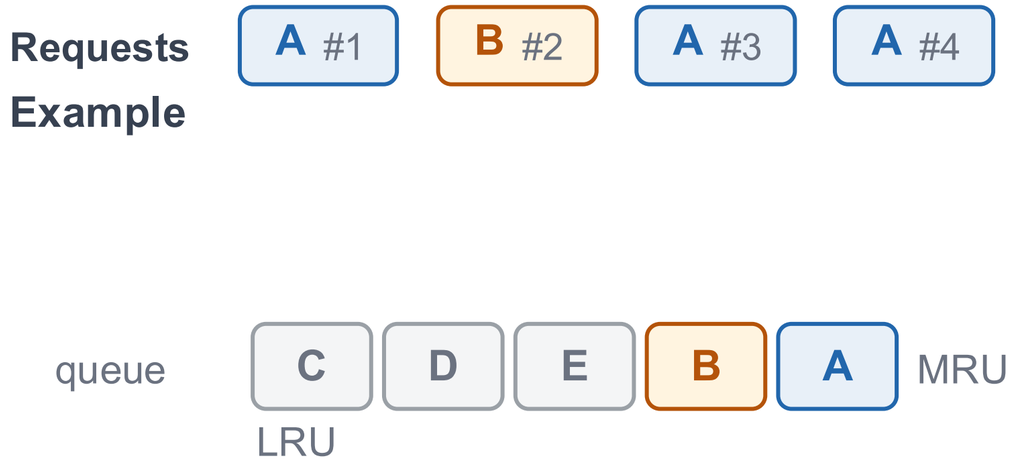


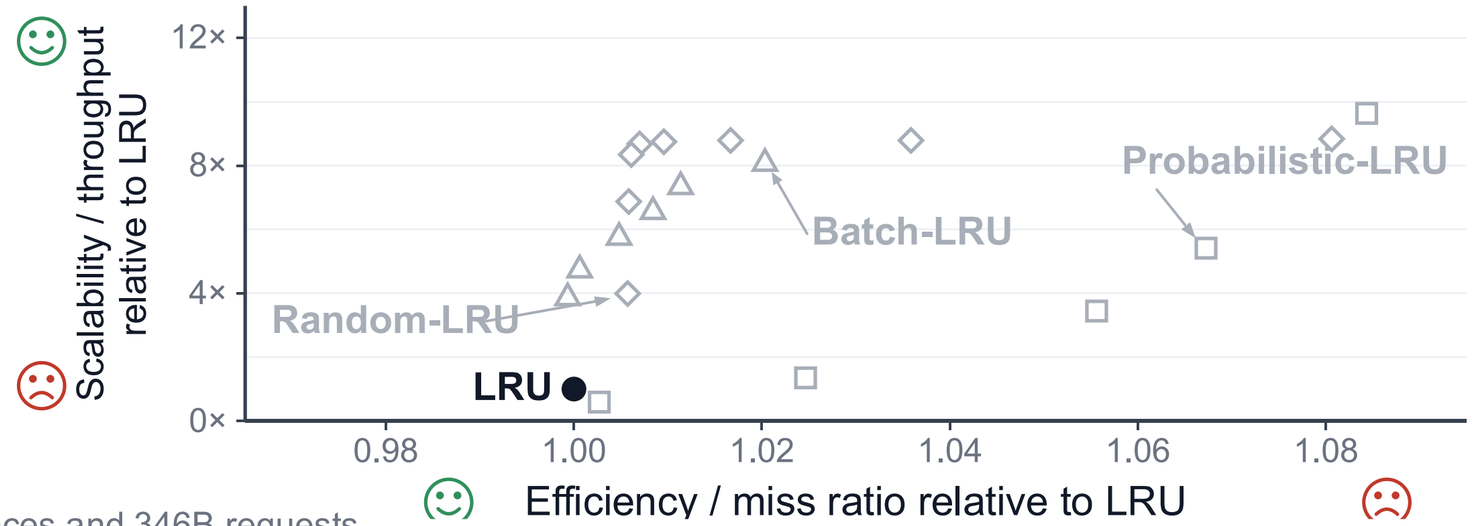
Evaluated on 6,357 traces and 346B requests

Delay-LRU filters repeated promotions in a window

CacheLib

Core idea: Start a *delay window* after each promotion.





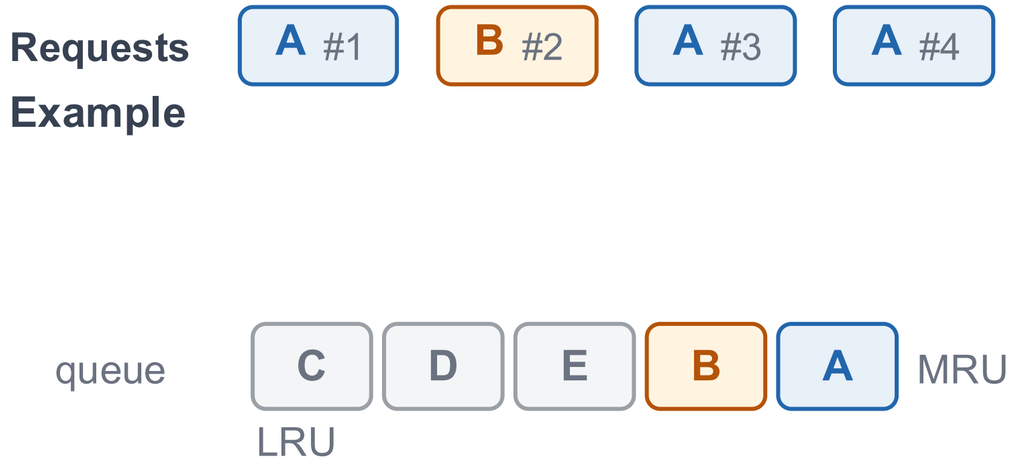
Evaluated on 6,357 traces and 346B requests

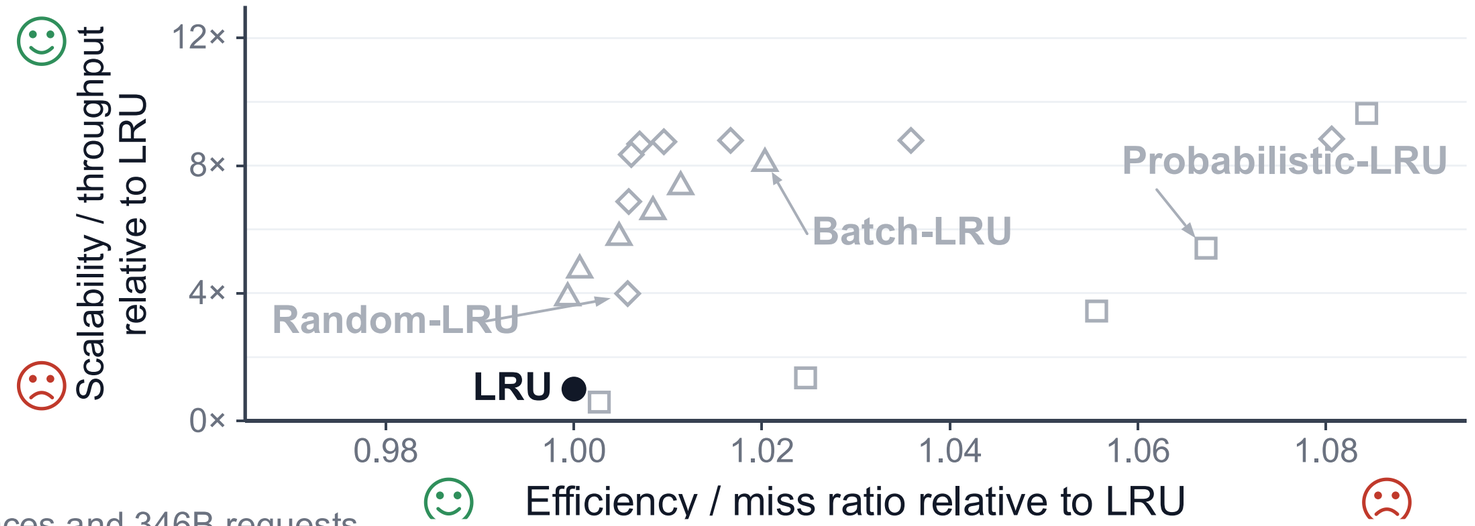
Delay-LRU filters repeated promotions in a window

CacheLib

Core idea: Start a *delay window* after each promotion.

Rule: *Skip* later promotions to the same object.





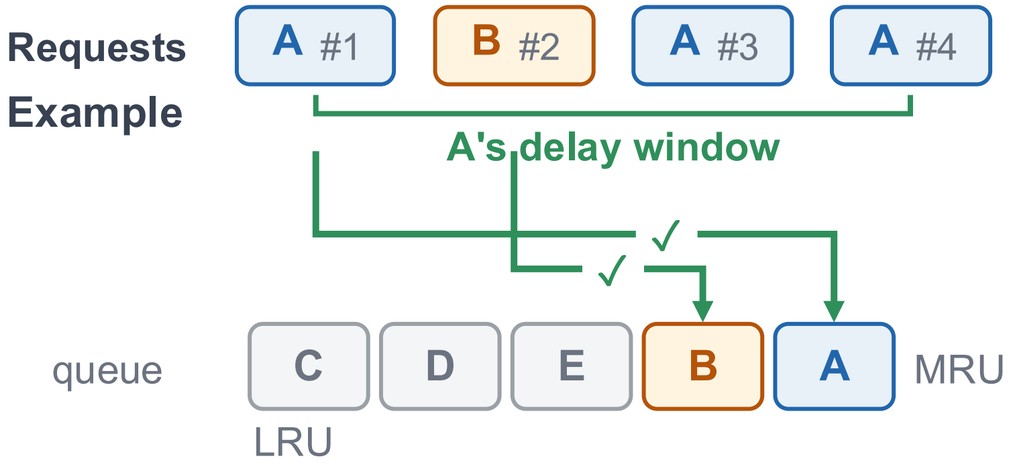
Evaluated on 6,357 traces and 346B requests

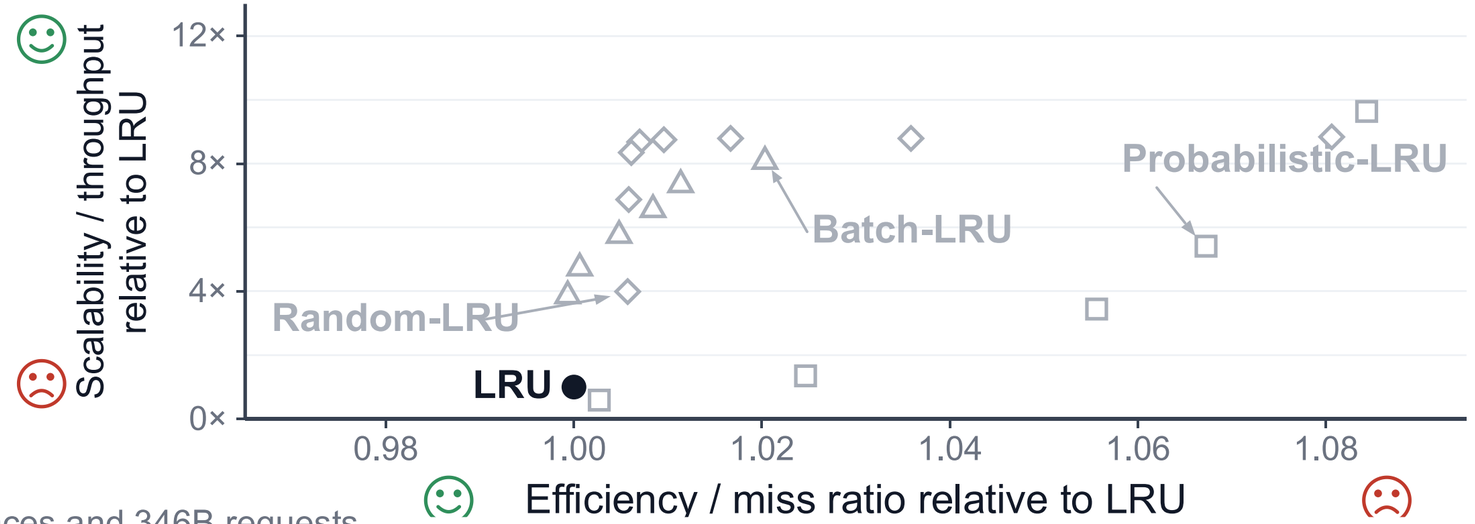
Delay-LRU filters repeated promotions in a window

CacheLib

Core idea: Start a *delay window* after each promotion.

Rule: *Skip* later promotions to the same object.





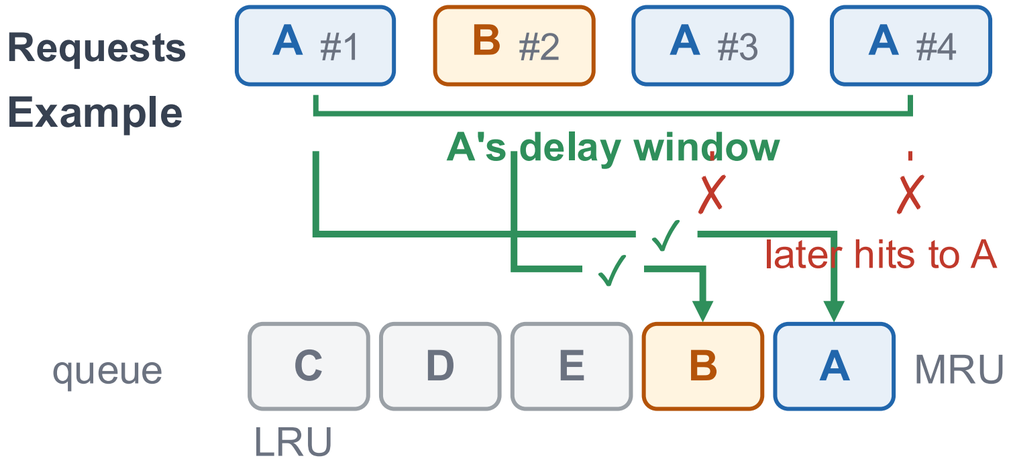
Evaluated on 6,357 traces and 346B requests

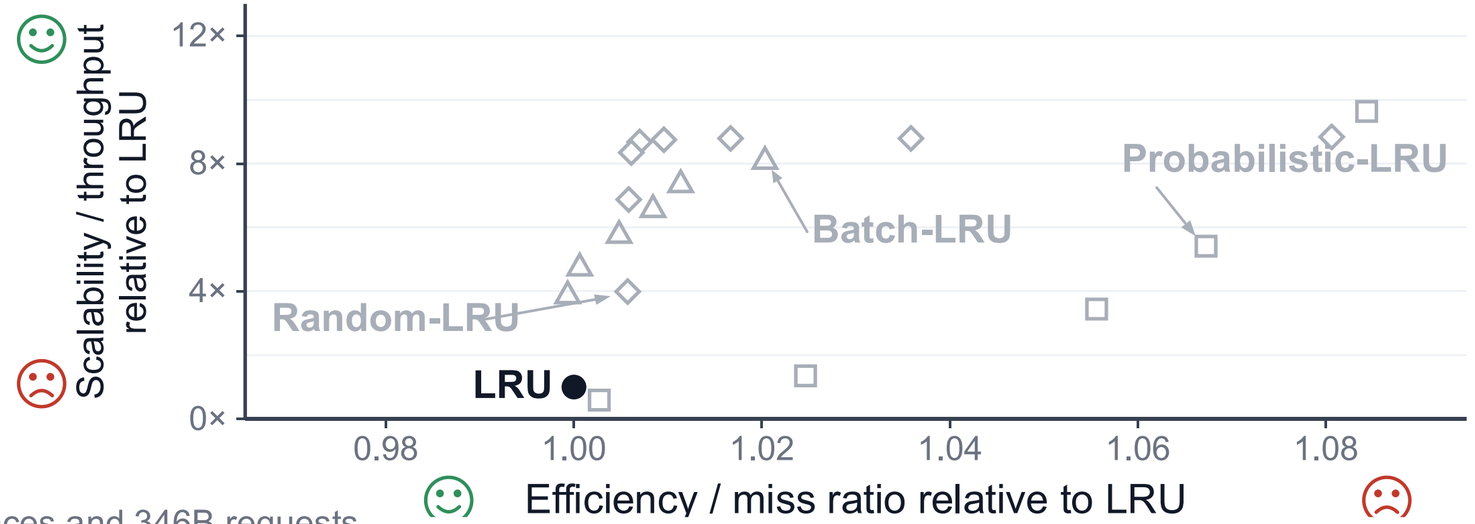
Delay-LRU filters repeated promotions in a window

CacheLib

Core idea: Start a *delay window* after each promotion.

Rule: *Skip* later promotions to the same object.





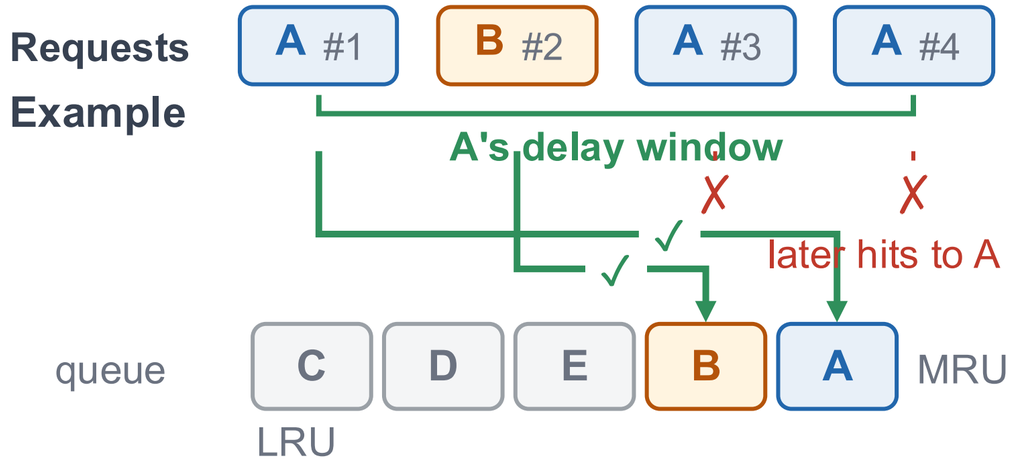
Evaluated on 6,357 traces and 346B requests

Delay-LRU filters repeated promotions in a window

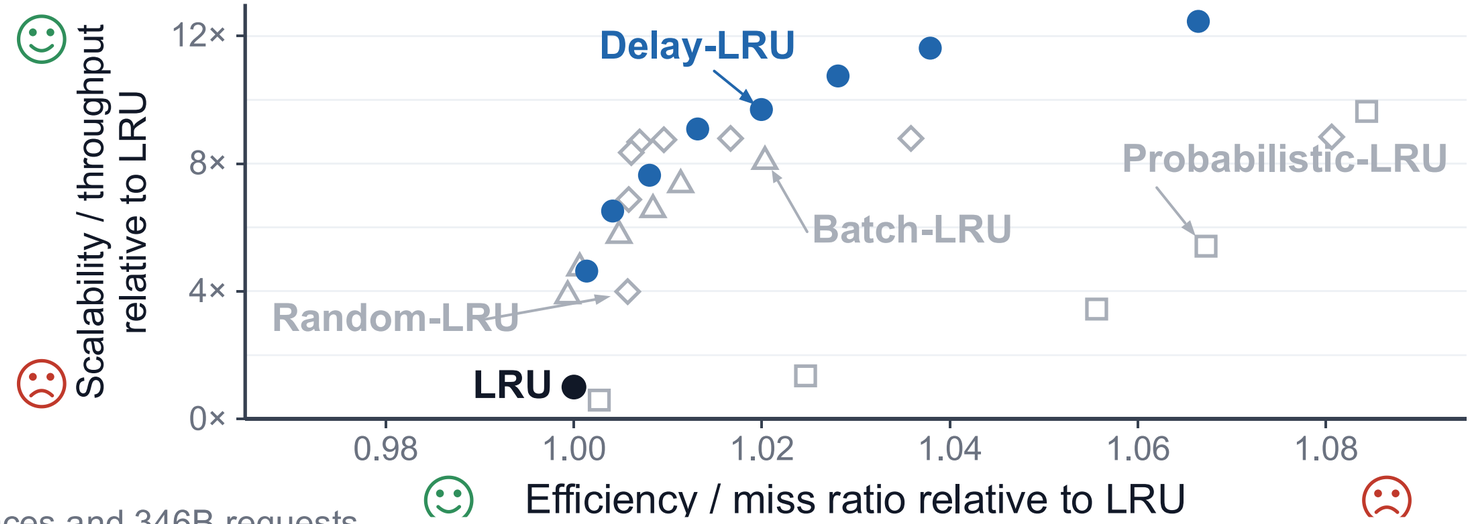
CacheLib

Core idea: Start a *delay window* after each promotion.

Rule: *Skip* later promotions to the same object.



Takeaway: Among the first four, Delay-LRU offers the best performance trade-off.



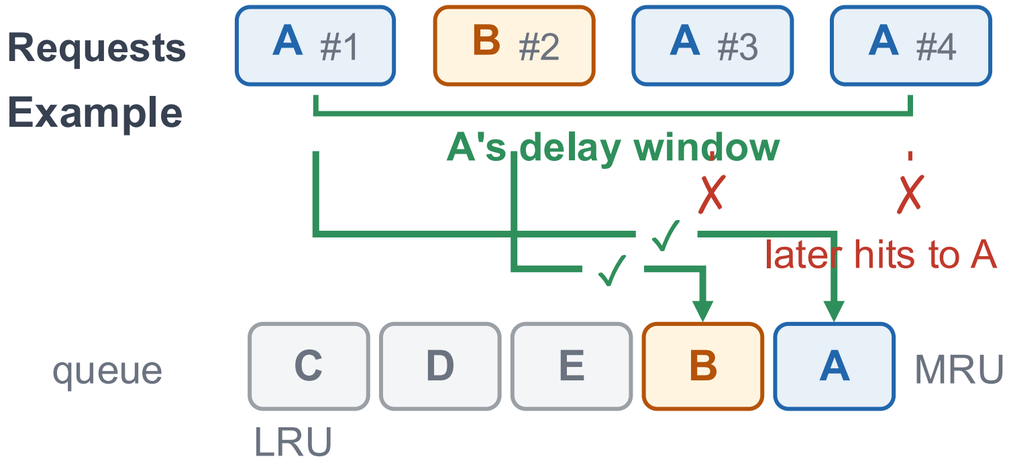
Evaluated on 6,357 traces and 346B requests

Delay-LRU filters repeated promotions in a window

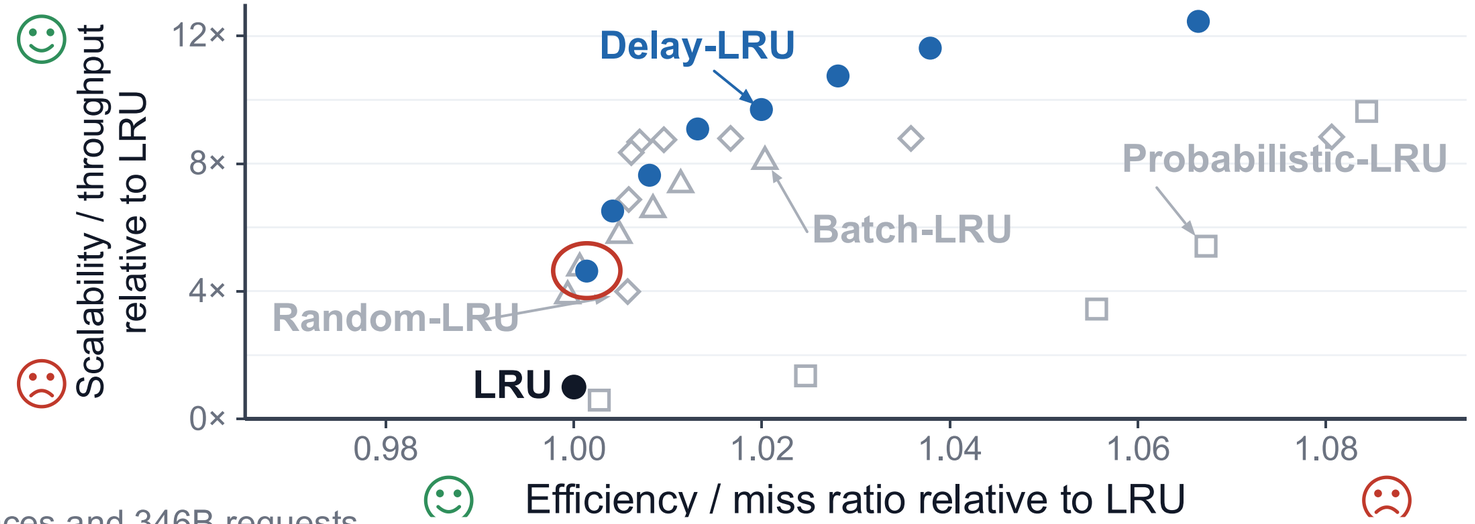
CacheLib

Core idea: Start a *delay window* after each promotion.

Rule: *Skip* later promotions to the same object.



Takeaway: Among the first four, Delay-LRU offers the best performance trade-off.



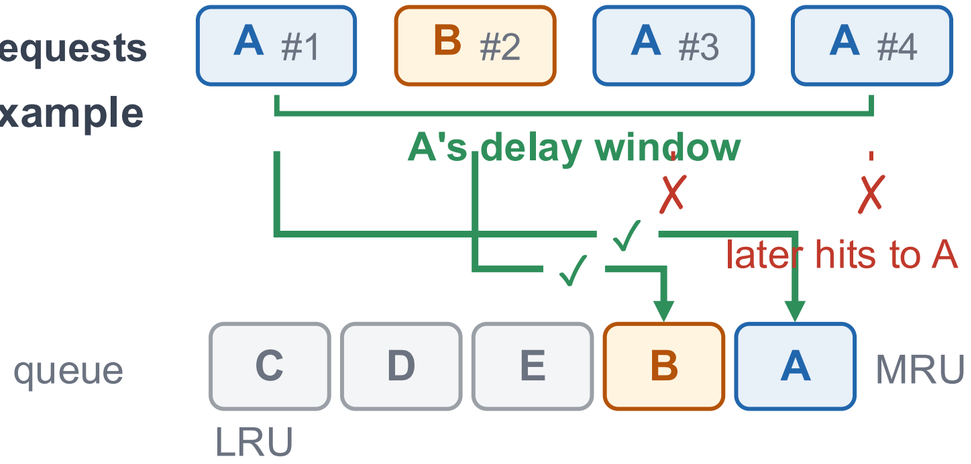
Evaluated on 6,357 traces and 346B requests

Delay-LRU filters repeated promotions in a window

CacheLib

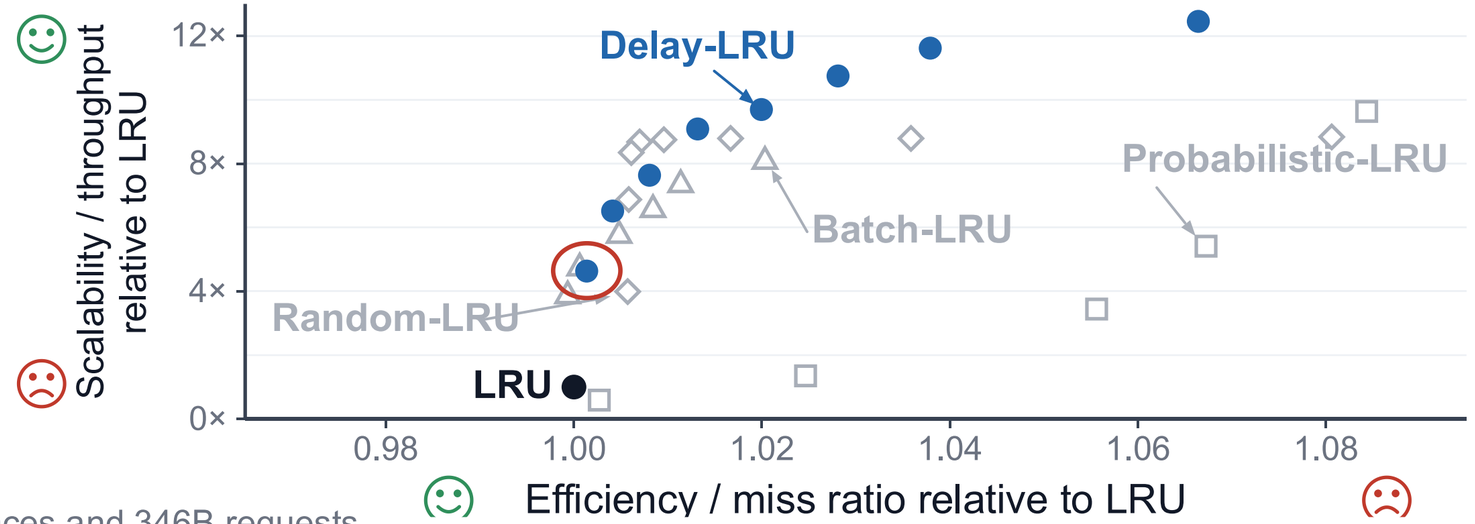
Core idea: Start a *delay window* after each promotion.

Rule: *Skip* later promotions to the same object.



Our conjecture: Filtering repeated hits to one object removes redundant promotions.

Takeaway: Among the first four, Delay-LRU offers the best performance trade-off.



Evaluated on 6,357 traces and 346B requests

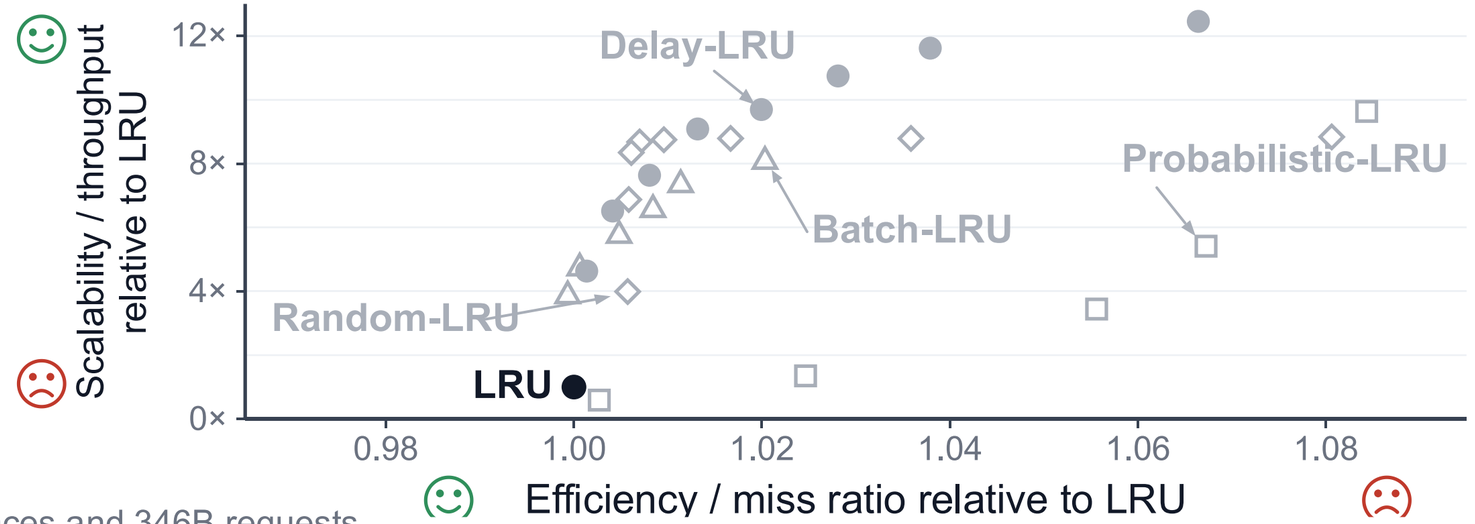
FIFO-reinsertion decides at eviction

RocksDB PostgreSQL

Simplified pseudocode

```
on hit(x):
  marked[x] = true
```

Related: QDLP · S3-FIFO · SIEVE



Evaluated on 6,357 traces and 346B requests

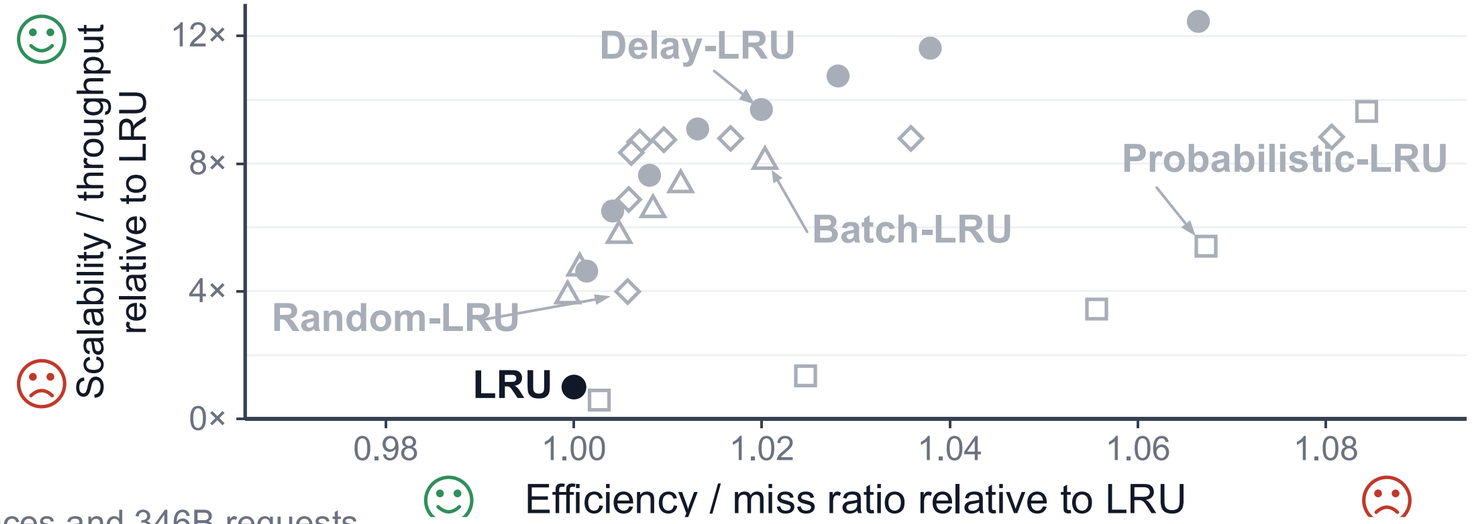
FIFO-reinsertion decides at eviction

RocksDB PostgreSQL

Simplified pseudocode

```
on hit(x):
  marked[x] = true

on eviction:
  while front is marked: clear; move to back
  evict the clear front
```

Related: QDLP · S3-FIFO · SIEVE



Evaluated on 6,357 traces and 346B requests

FIFO-reinsertion decides at eviction

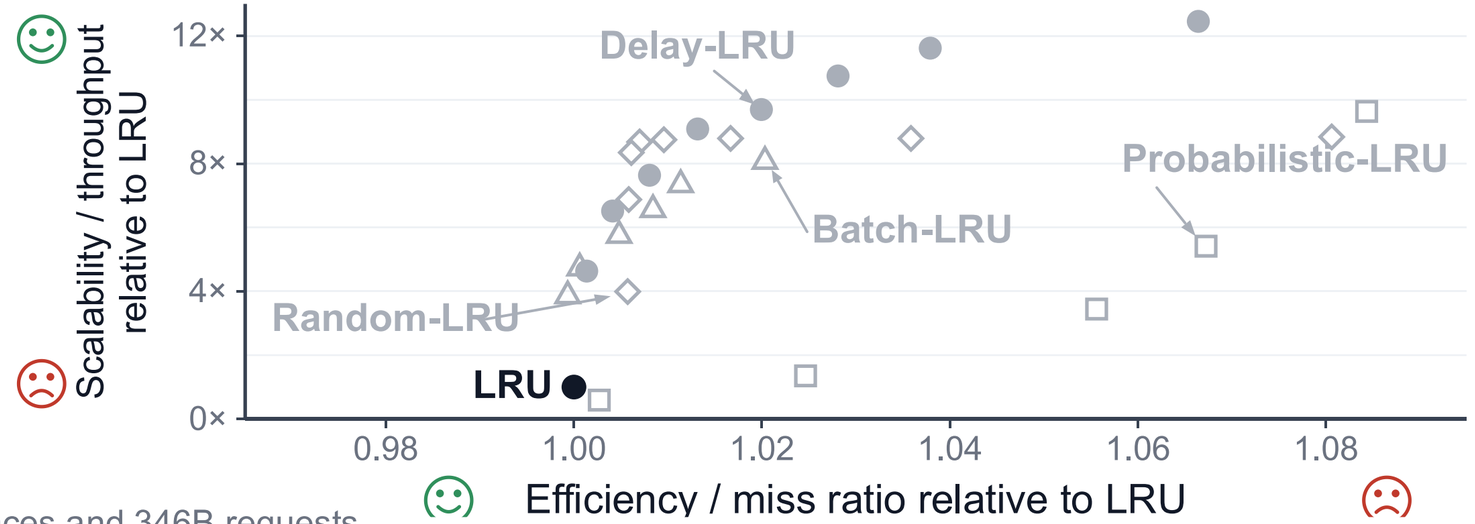
RocksDB PostgreSQL

Simplified pseudocode

```
on hit(x):
  marked[x] = true

on eviction:
  while front is marked: clear; move to back
  evict the clear front
```

Example

Related: QDLP · S3-FIFO · SIEVE



Evaluated on 6,357 traces and 346B requests

FIFO-reinsertion decides at eviction

RocksDB PostgreSQL

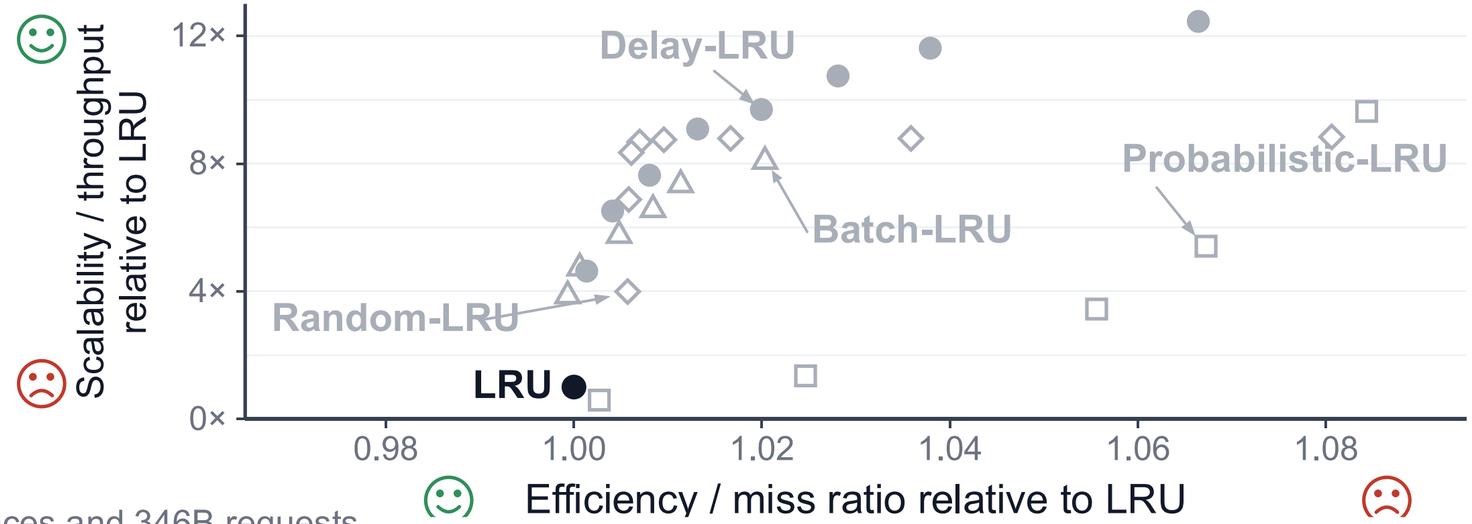
Simplified pseudocode

```
on hit(x):
  marked[x] = true

on eviction:
  while front is marked: clear; move to back
  evict the clear front
```

Example

cache miss on X

Related: QDLP · S3-FIFO · SIEVE



Evaluated on 6,357 traces and 346B requests

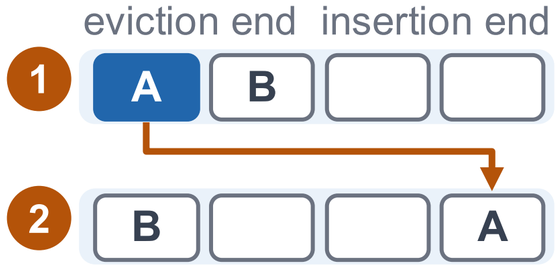
FIFO-reinsertion decides at eviction

RocksDB PostgreSQL

Simplified pseudocode

```
on hit(x):
  marked[x] = true

on eviction:
  while front is marked: clear; move to back
  evict the clear front
```

Example



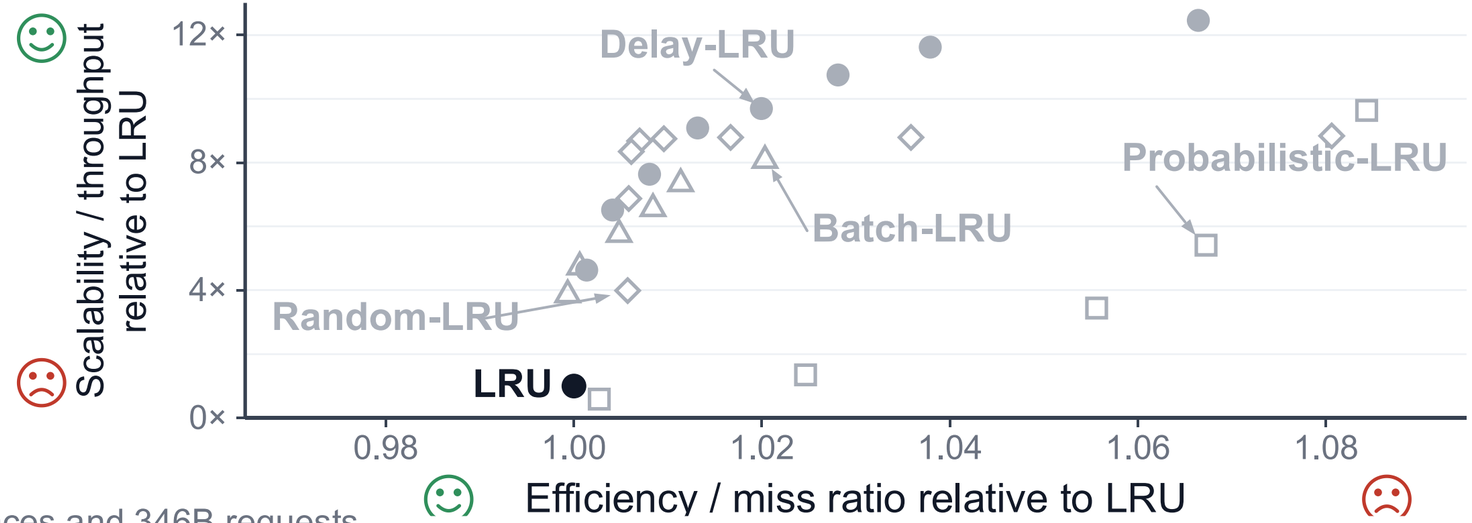
earlier hit marks A

reinsert A, clear marker

promotion from the earlier hit

cache miss on X

Related: QDLP · S3-FIFO · SIEVE



Evaluated on 6,357 traces and 346B requests

FIFO-reinsertion decides at eviction

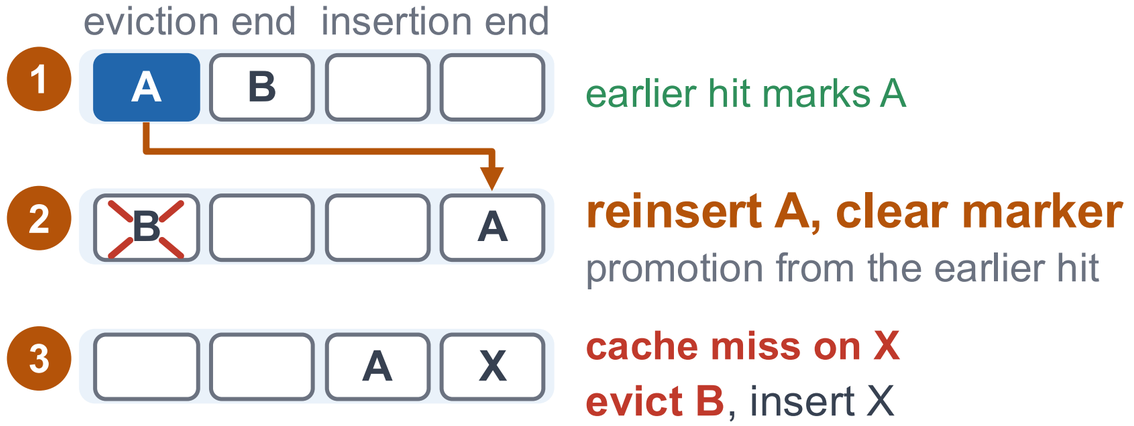
RocksDB PostgreSQL

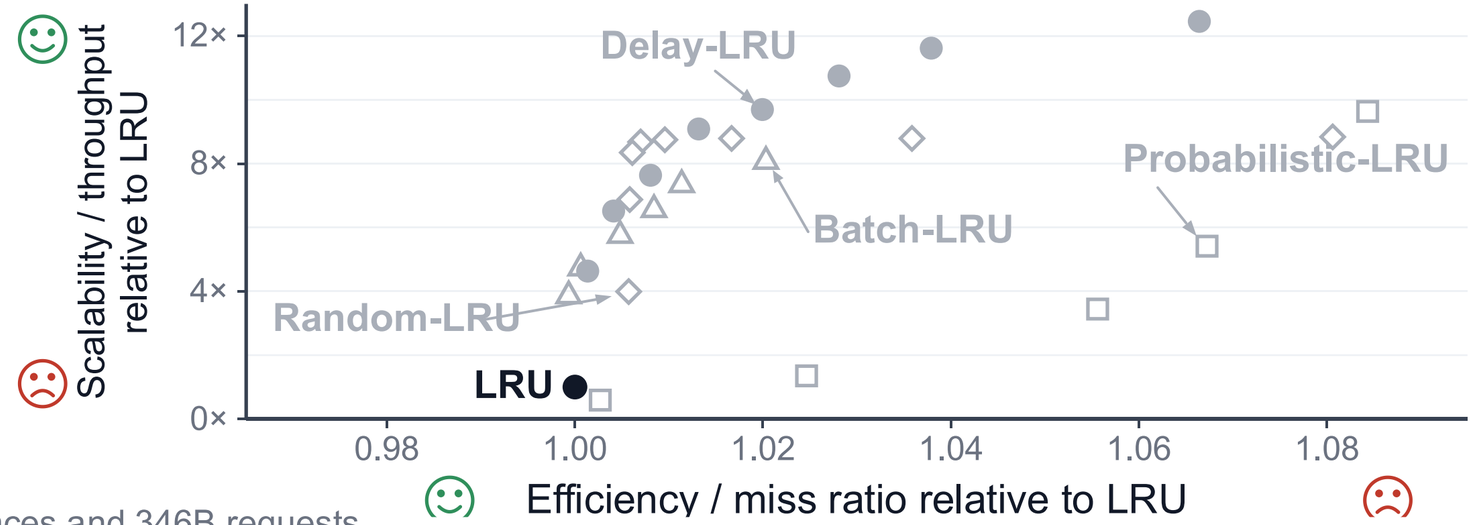
Simplified pseudocode

```
on hit(x):
  marked[x] = true

on eviction:
  while front is marked: clear; move to back
  evict the clear front
```

Related: QDLP · S3-FIFO · SIEVE

Example





Evaluated on 6,357 traces and 346B requests

FIFO-reinsertion decides at eviction

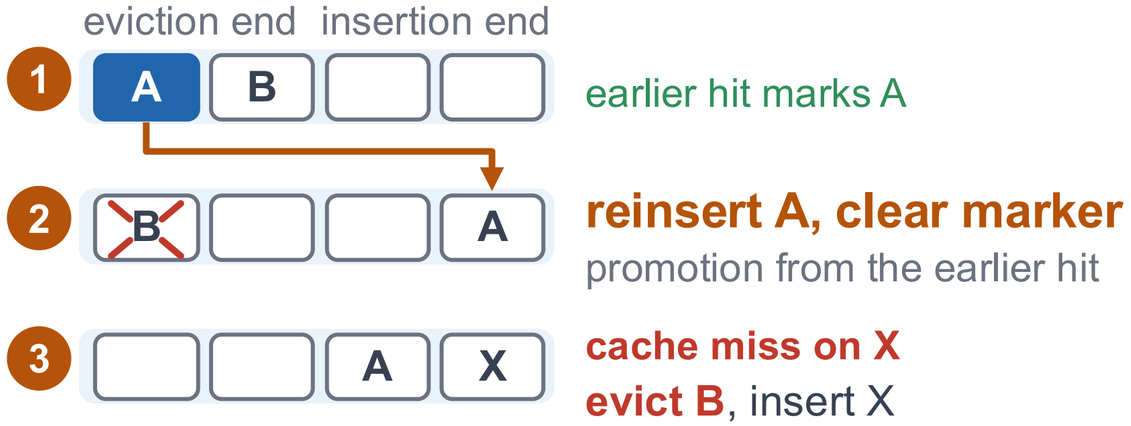
RocksDB PostgreSQL

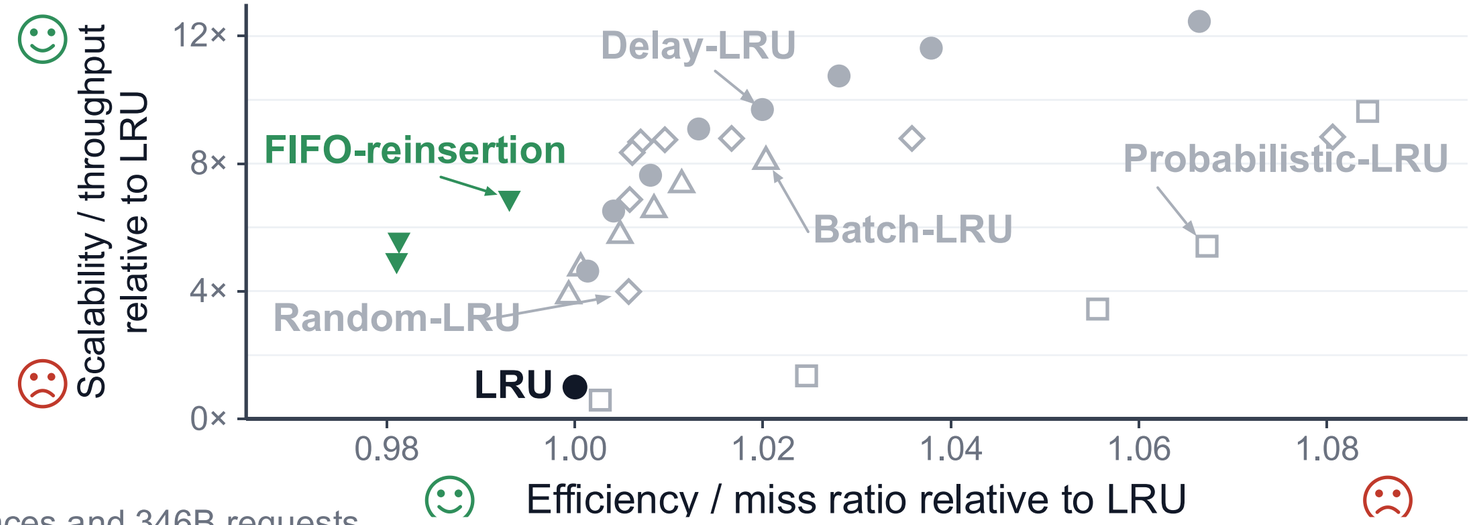
Simplified pseudocode

```
on hit(x):
  marked[x] = true

on eviction:
  while front is marked: clear; move to back
  evict the clear front
```

Related: QDLP · S3-FIFO · SIEVE

Example





Evaluated on 6,357 traces and 346B requests

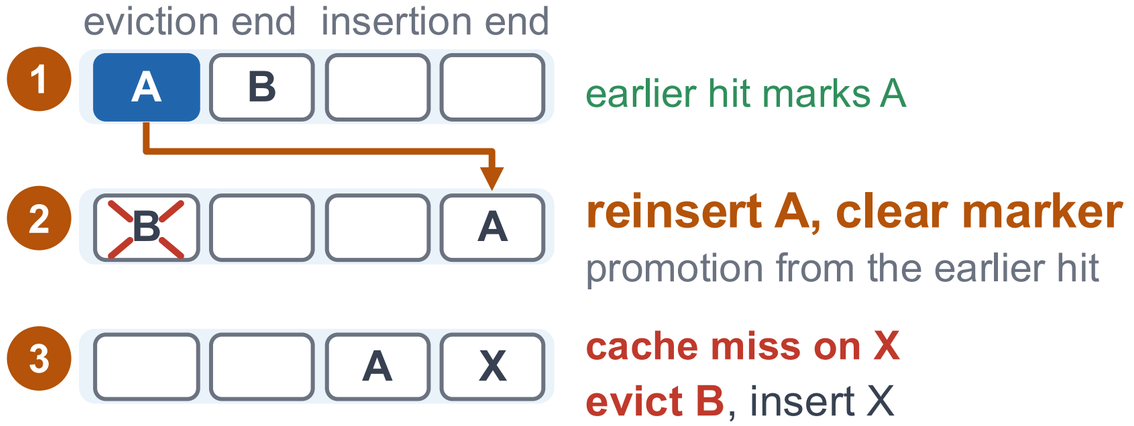
FIFO-reinsertion decides at eviction

RocksDB PostgreSQL

Simplified pseudocode

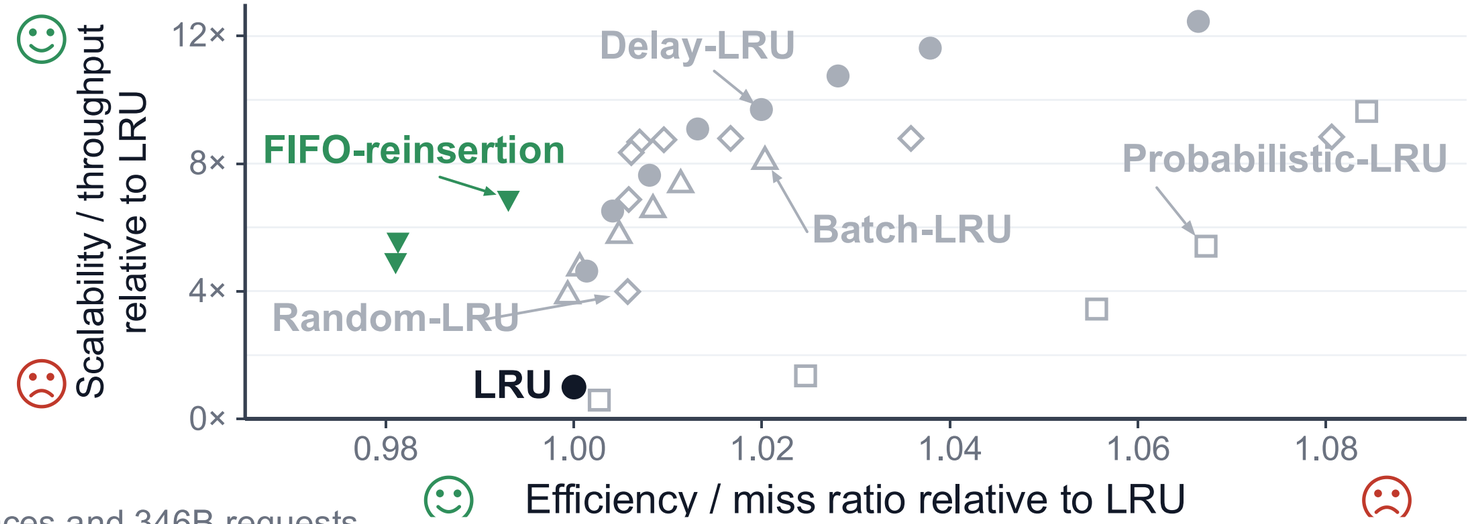
```
on hit(x):
  marked[x] = true

on eviction:
  while front is marked: clear; move to back
  evict the clear front
```

Example



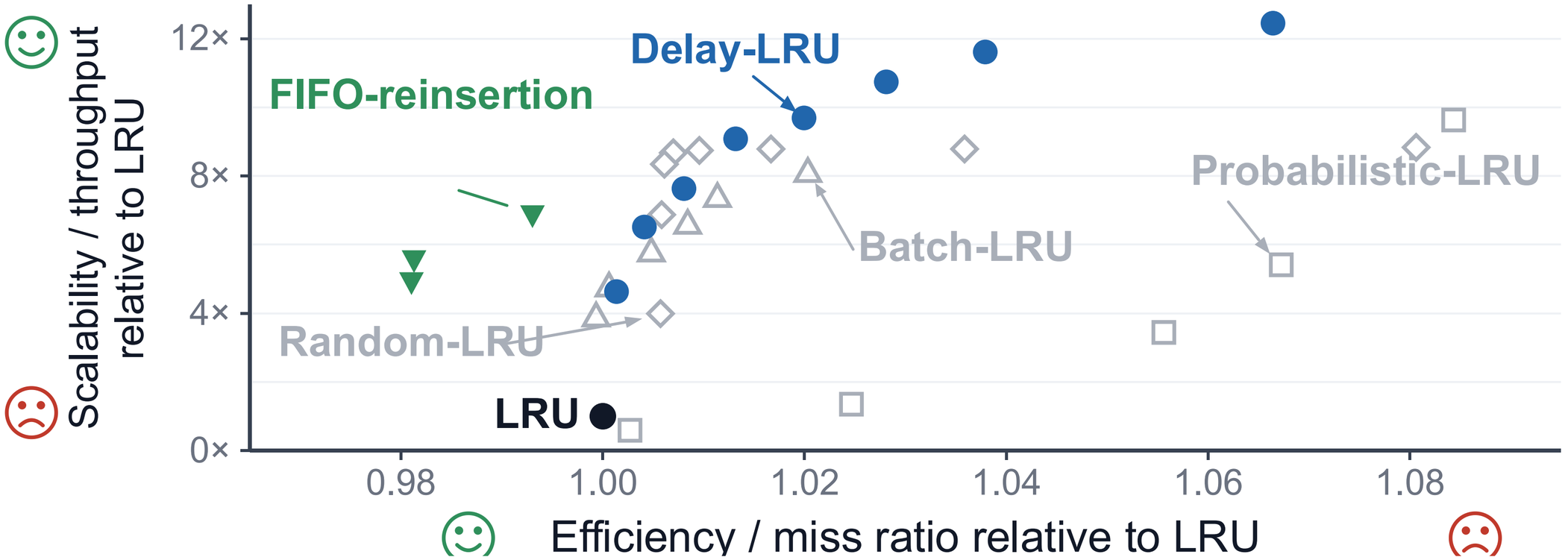
Related: QDLP · S3-FIFO · SIEVE

Takeaway: FIFO-reinsertion improves both throughput and miss ratio vs. LRU.



Evaluated on 6,357 traces and 346B requests

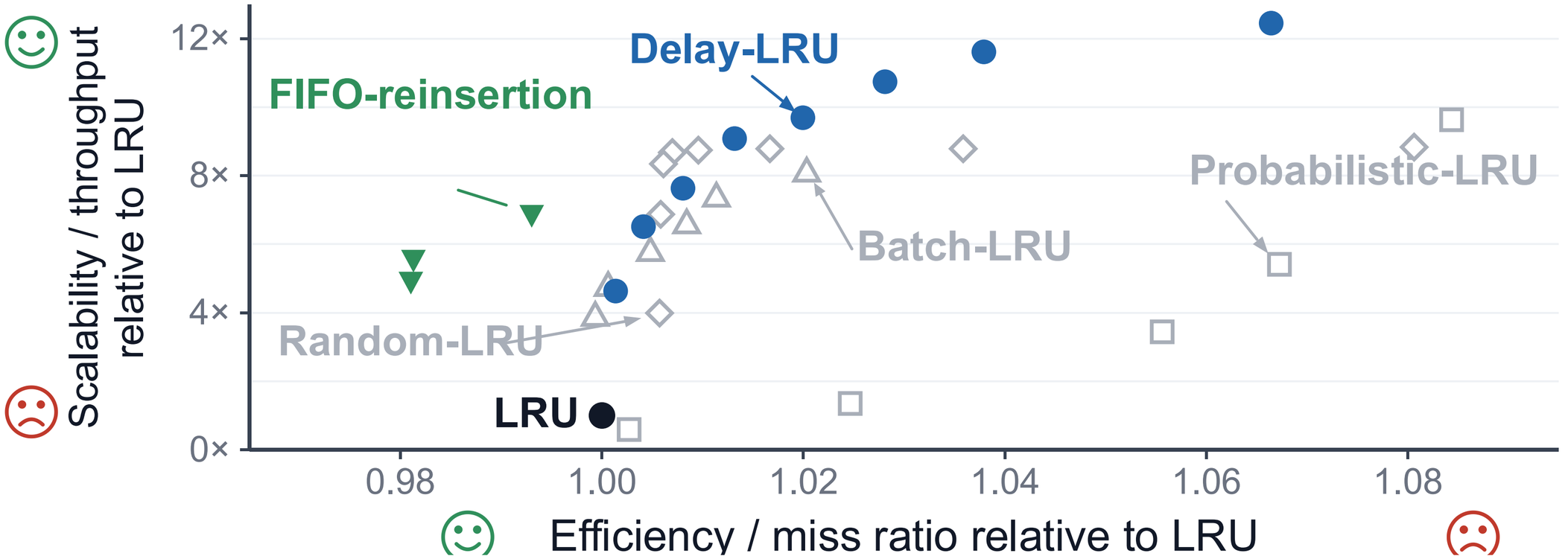
Each technique has its own performance trade-off



Evaluated on 6,357 traces and 346B requests

Each technique has its own performance trade-off

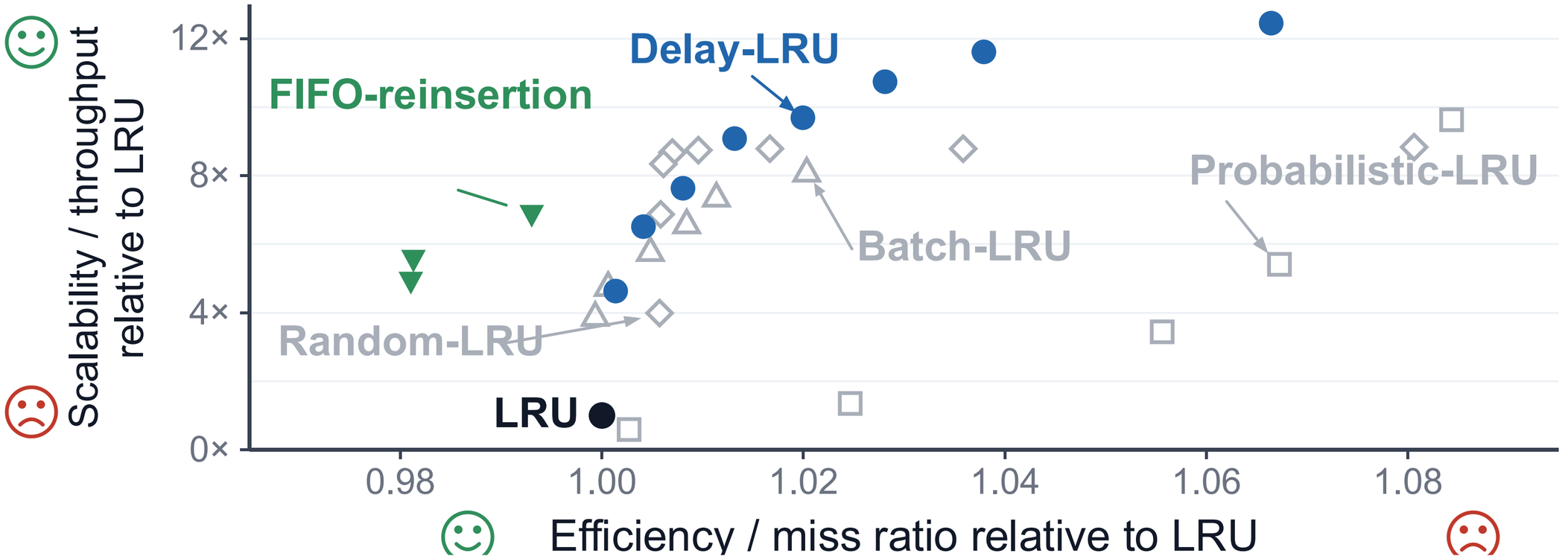
- **Delay-LRU** filters repeated promotions selectively, reaching about 5× LRU's throughput.



Evaluated on 6,357 traces and 346B requests

Each technique has its own performance trade-off

- **Delay-LRU** filters repeated promotions selectively, reaching about 5× LRU's throughput.
- **FIFO-reinsertion** decides at eviction, lowering mean miss ratio by almost 2%.

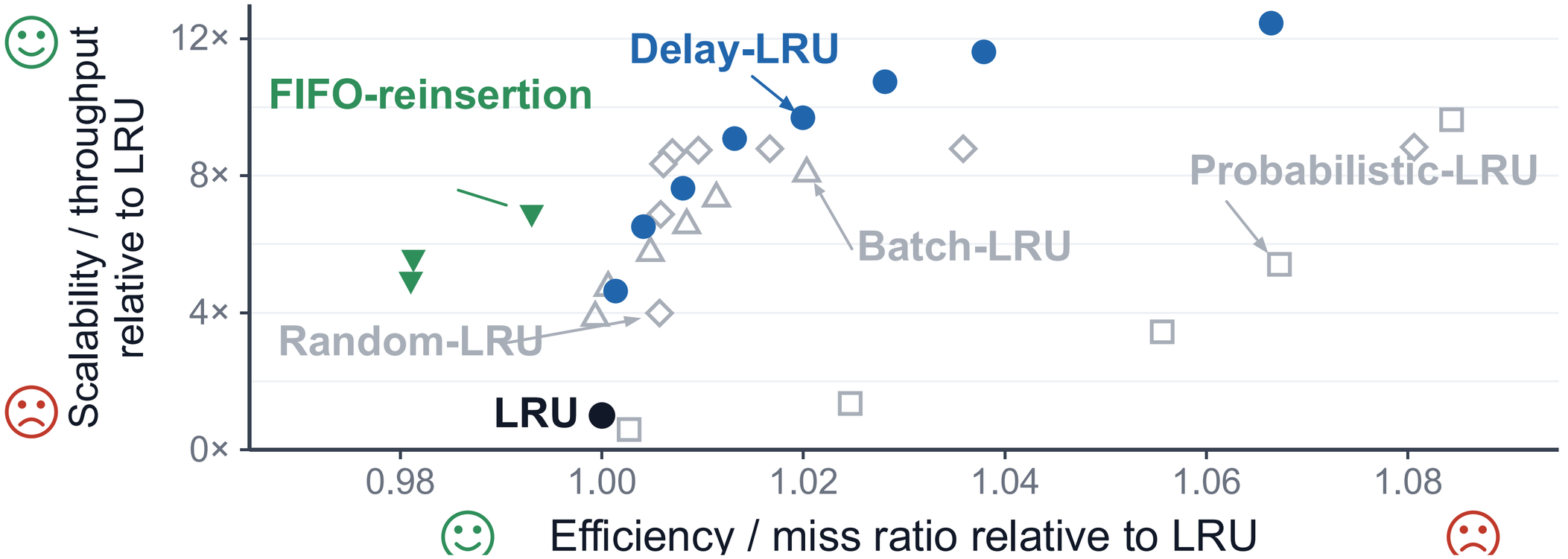


Evaluated on 6,357 traces and 346B requests

Each technique has its own performance trade-off

- **Delay-LRU** filters repeated promotions selectively, reaching about 5× LRU's throughput.
- **FIFO-reinsertion** decides at eviction, lowering mean miss ratio by almost 2%.

Can we reduce FIFO-reinsertion's promotions without losing its low miss ratio?

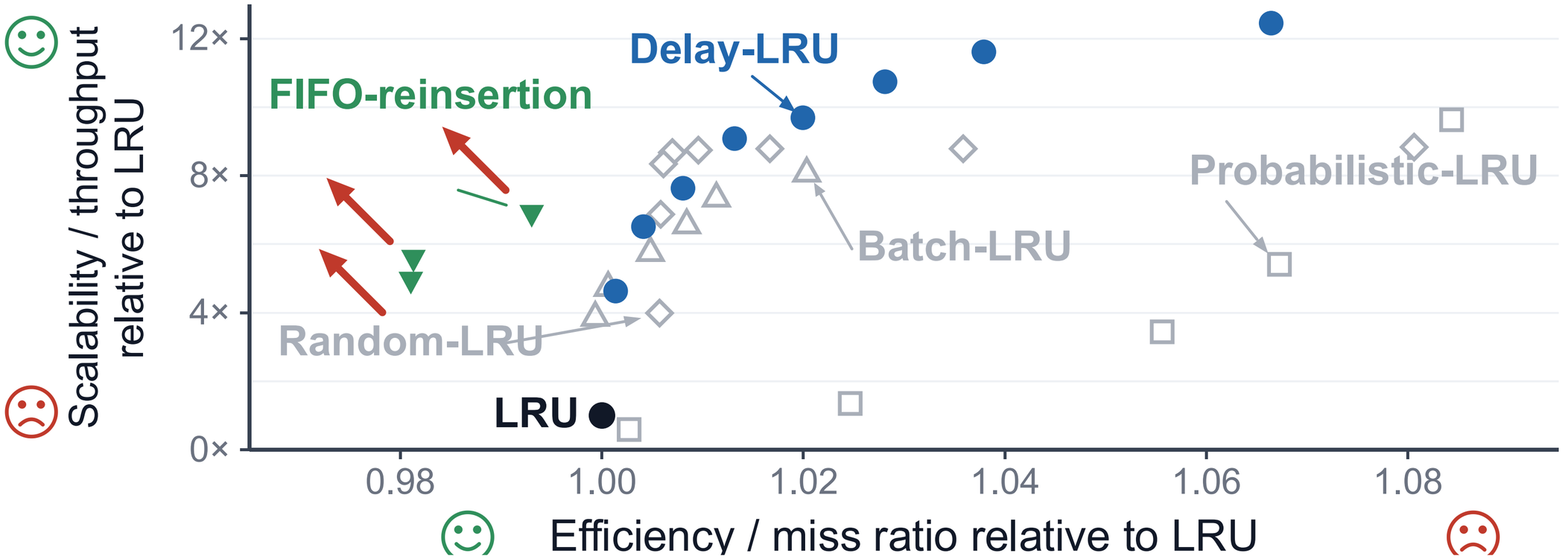


Evaluated on 6,357 traces and 346B requests

Each technique has its own performance trade-off

- **Delay-LRU** filters repeated promotions selectively, reaching about 5× LRU's throughput.
- **FIFO-reinsertion** decides at eviction, lowering mean miss ratio by almost 2%.

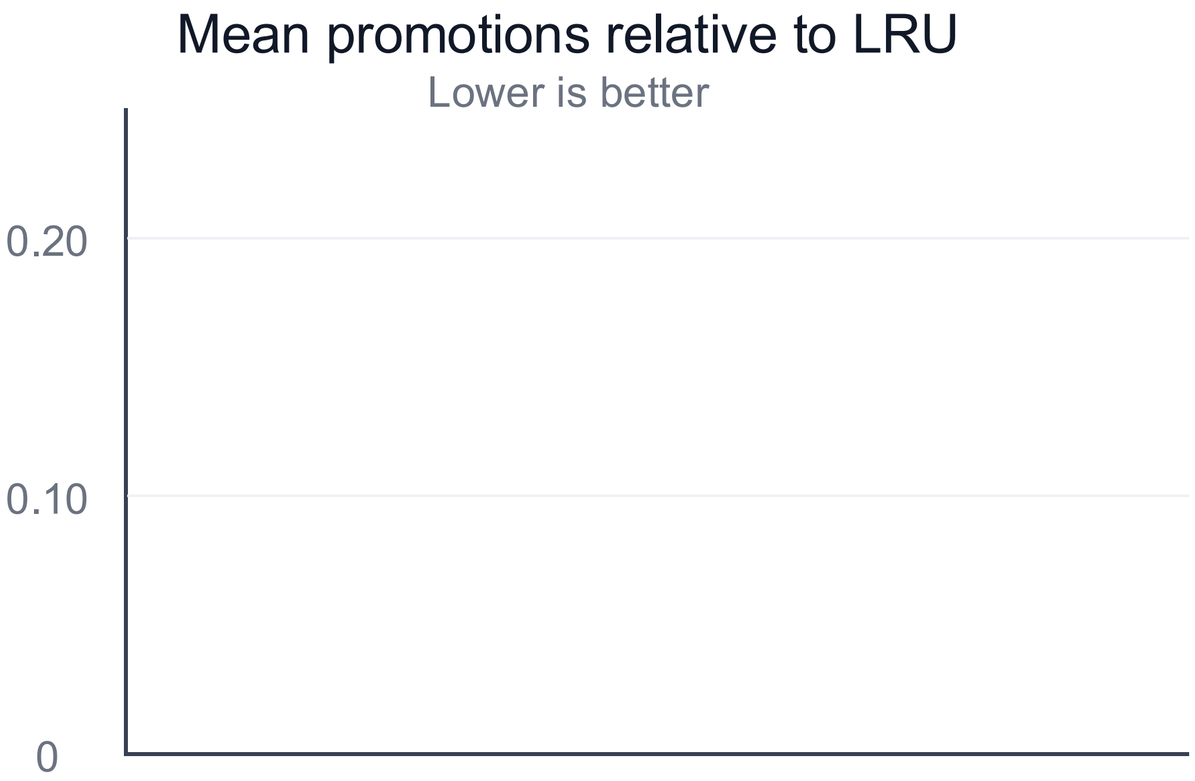
Can we reduce FIFO-reinsertion's promotions without losing its low miss ratio?

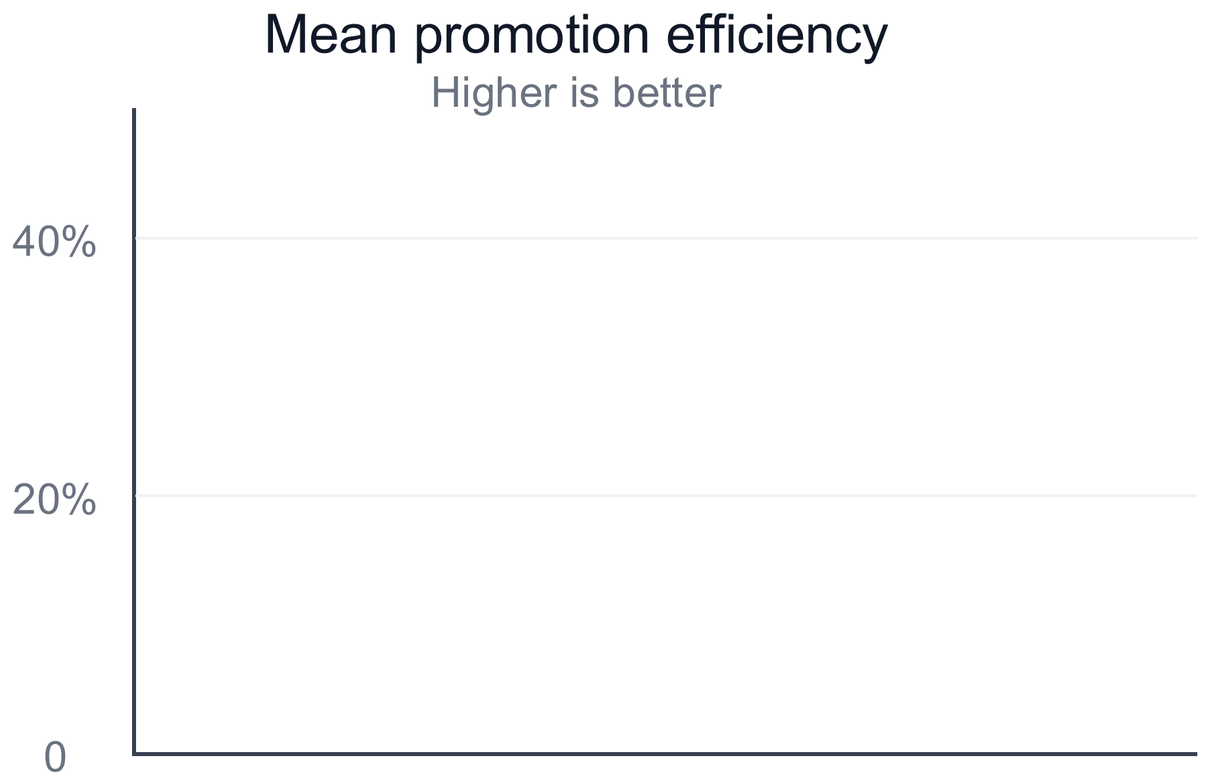


Evaluated on 6,357 traces and 346B requests

D-FR doubles promotion efficiency

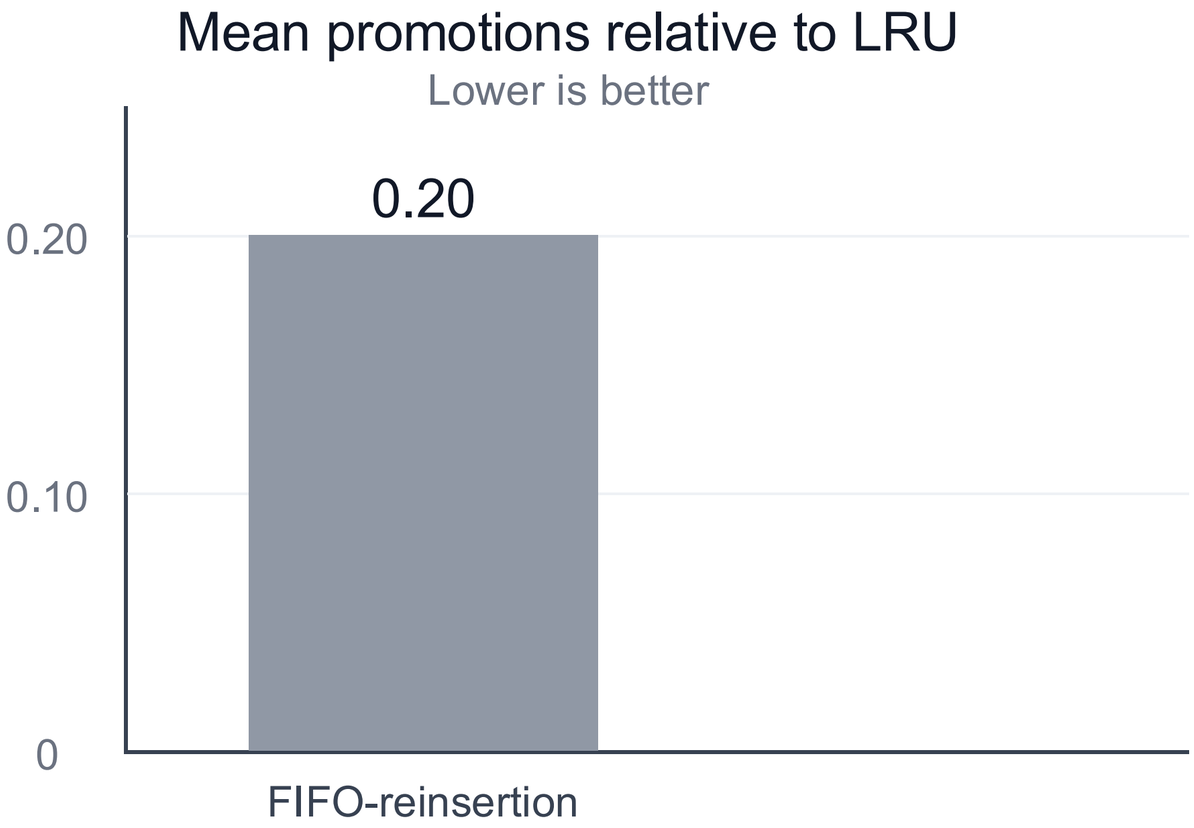
- Delayed FIFO-reinsertion (D-FR) adds a delay window to FIFO-reinsertion.
  - A hit inside the window does not set the marker.

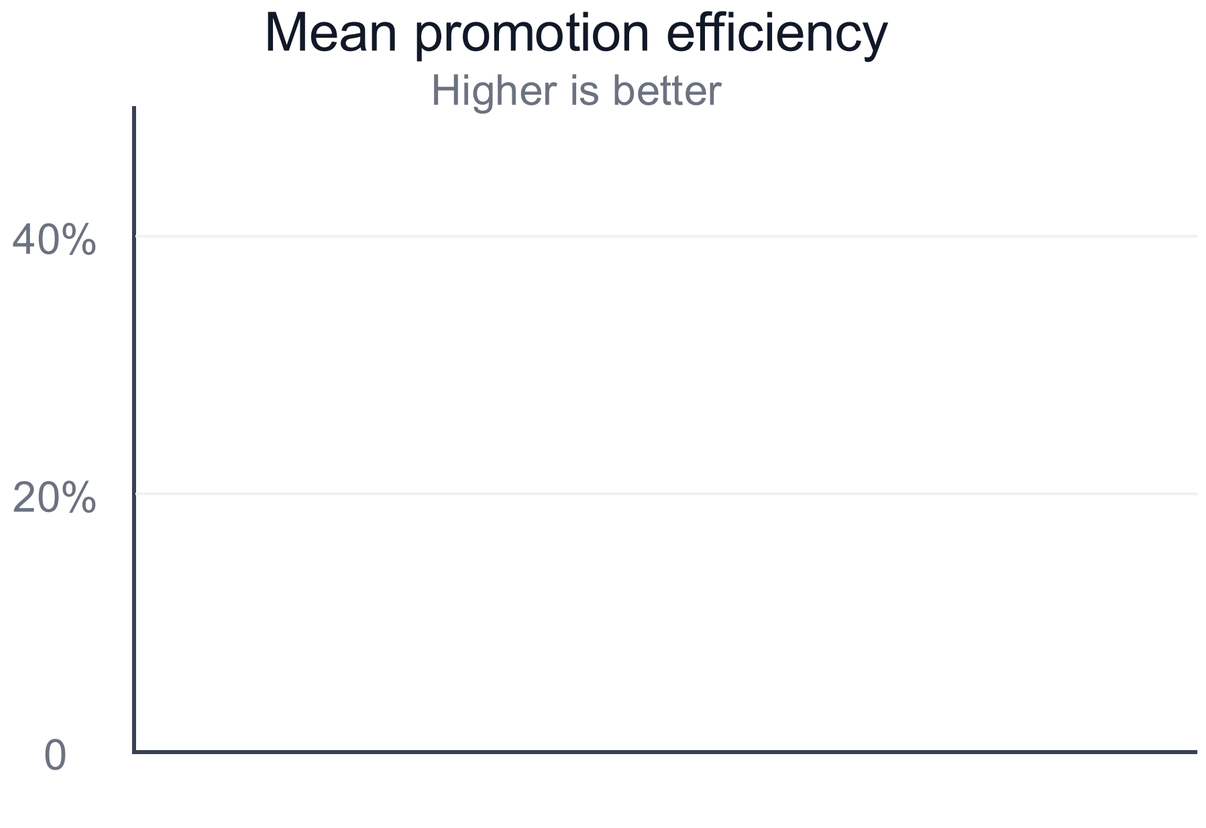




D-FR doubles promotion efficiency

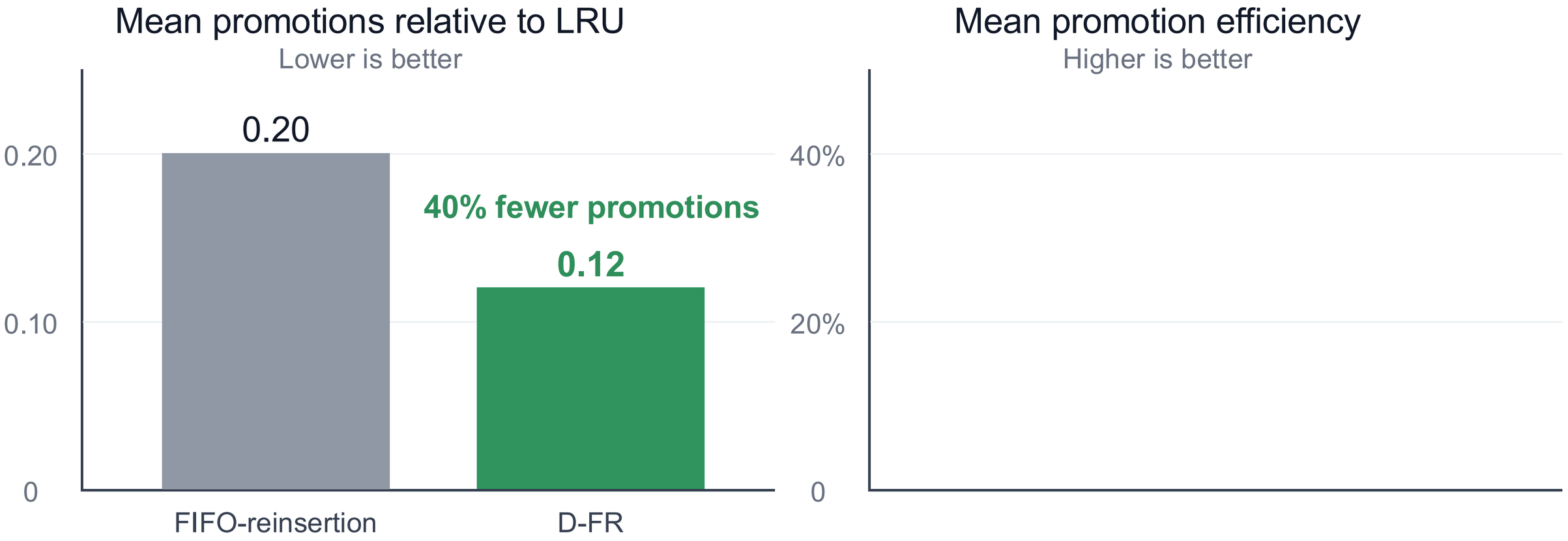
- Delayed FIFO-reinsertion (D-FR) adds a delay window to FIFO-reinsertion.
  - A hit inside the window does not set the marker.





D-FR doubles promotion efficiency

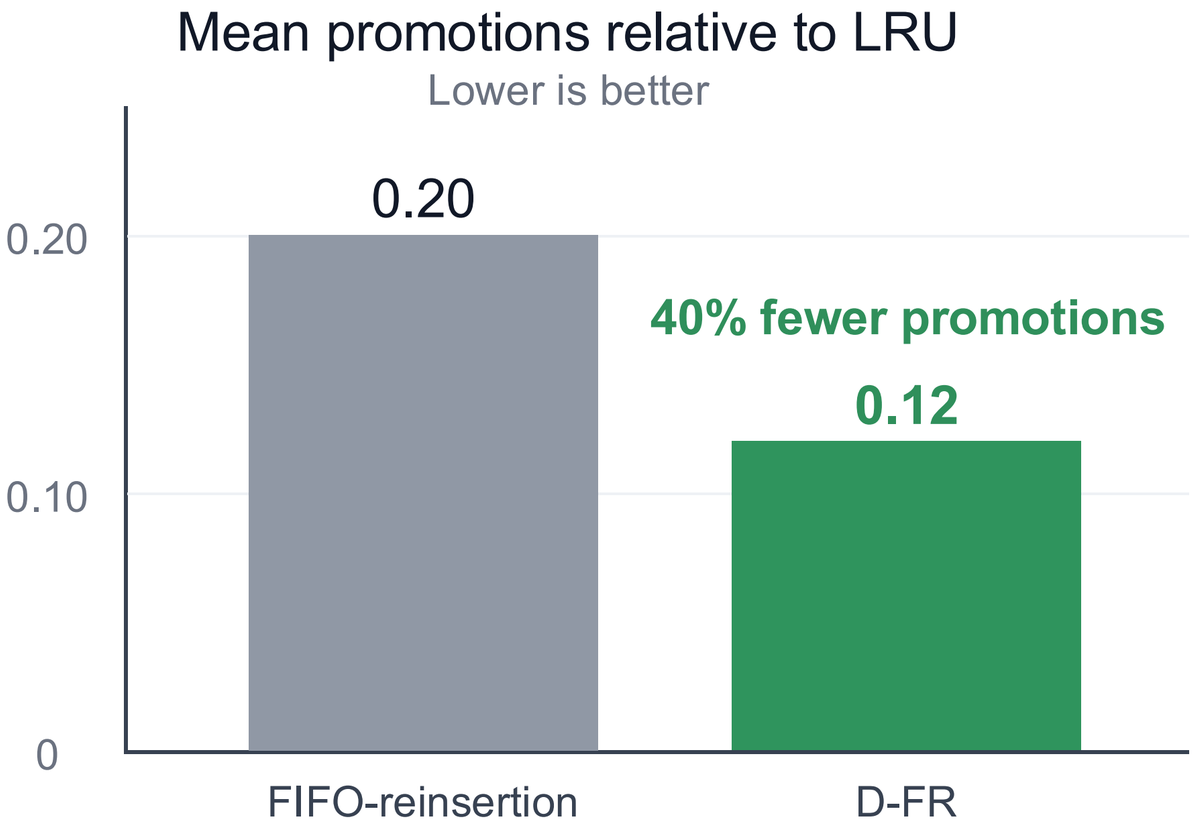
- Delayed FIFO-reinsertion (D-FR) adds a delay window to FIFO-reinsertion.
  - A hit inside the window does not set the marker.

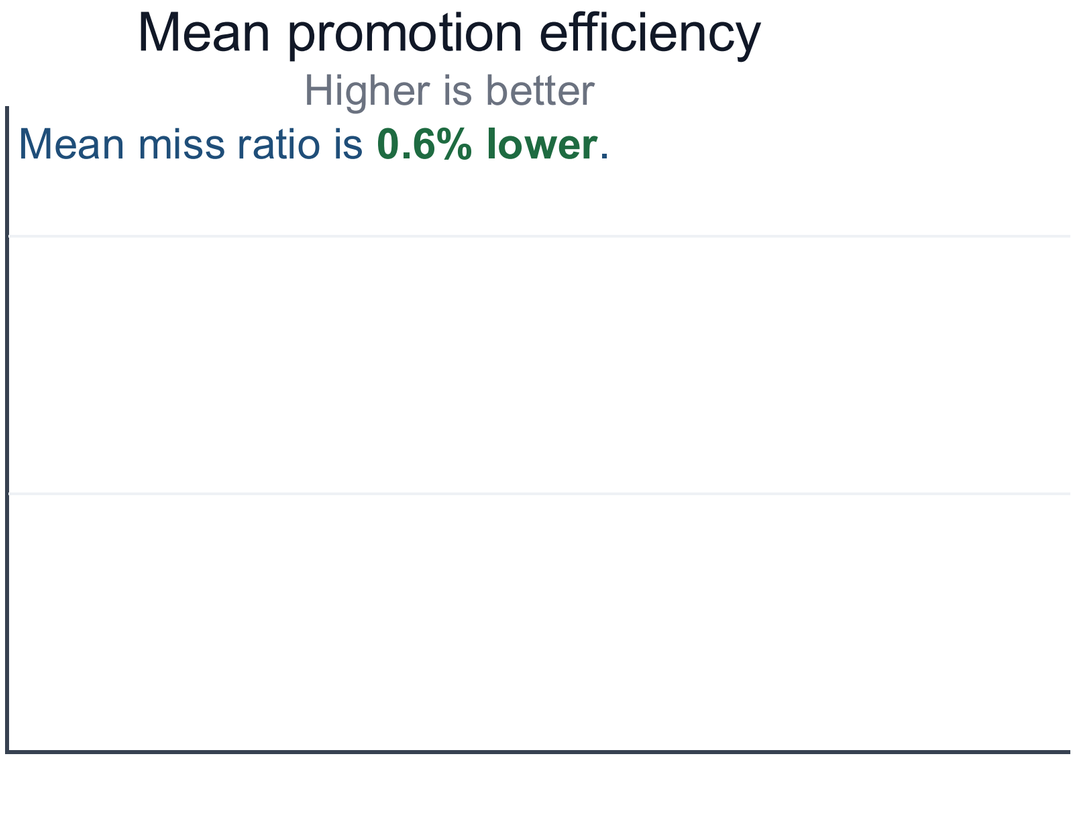


D-FR doubles promotion efficiency

- Delayed FIFO-reinsertion (D-FR) adds a delay window to FIFO-reinsertion.
  - A hit inside the window does not set the marker.

$$\text{Promotion efficiency} = \frac{\text{misses avoided}}{\text{promotions}}$$

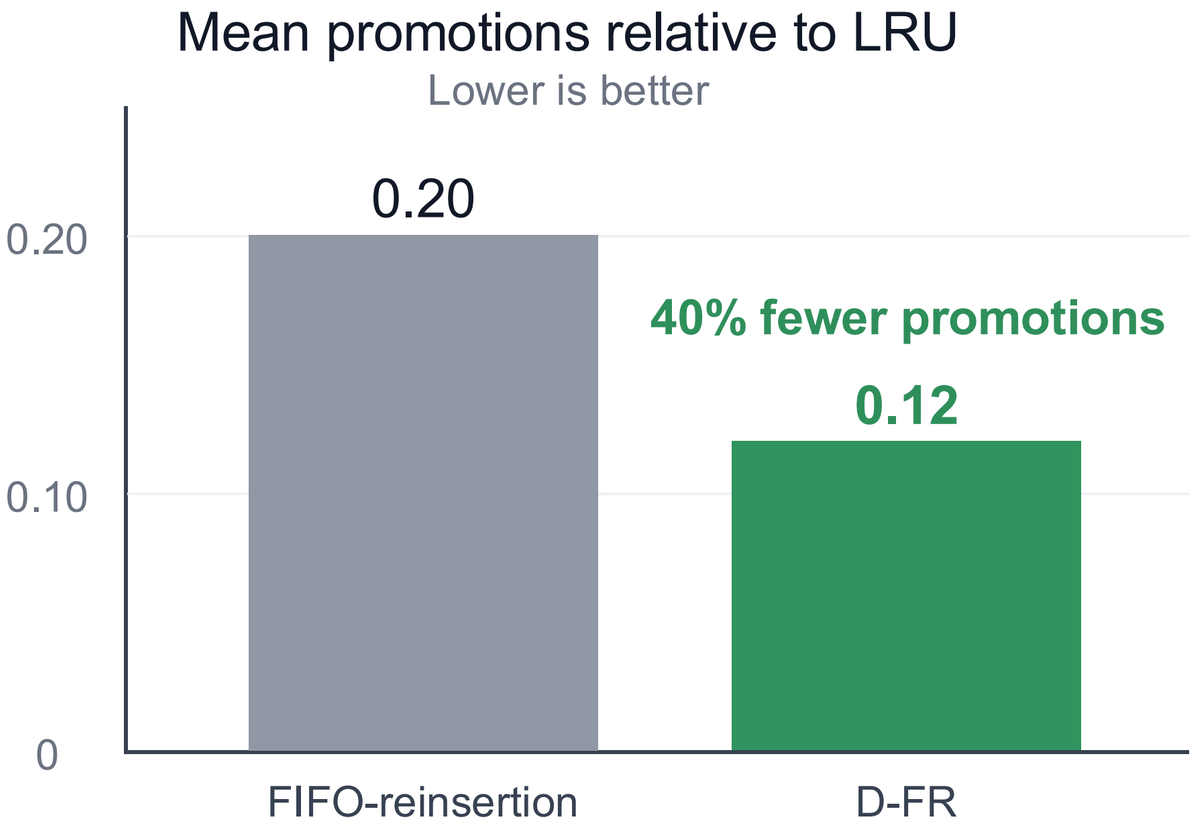


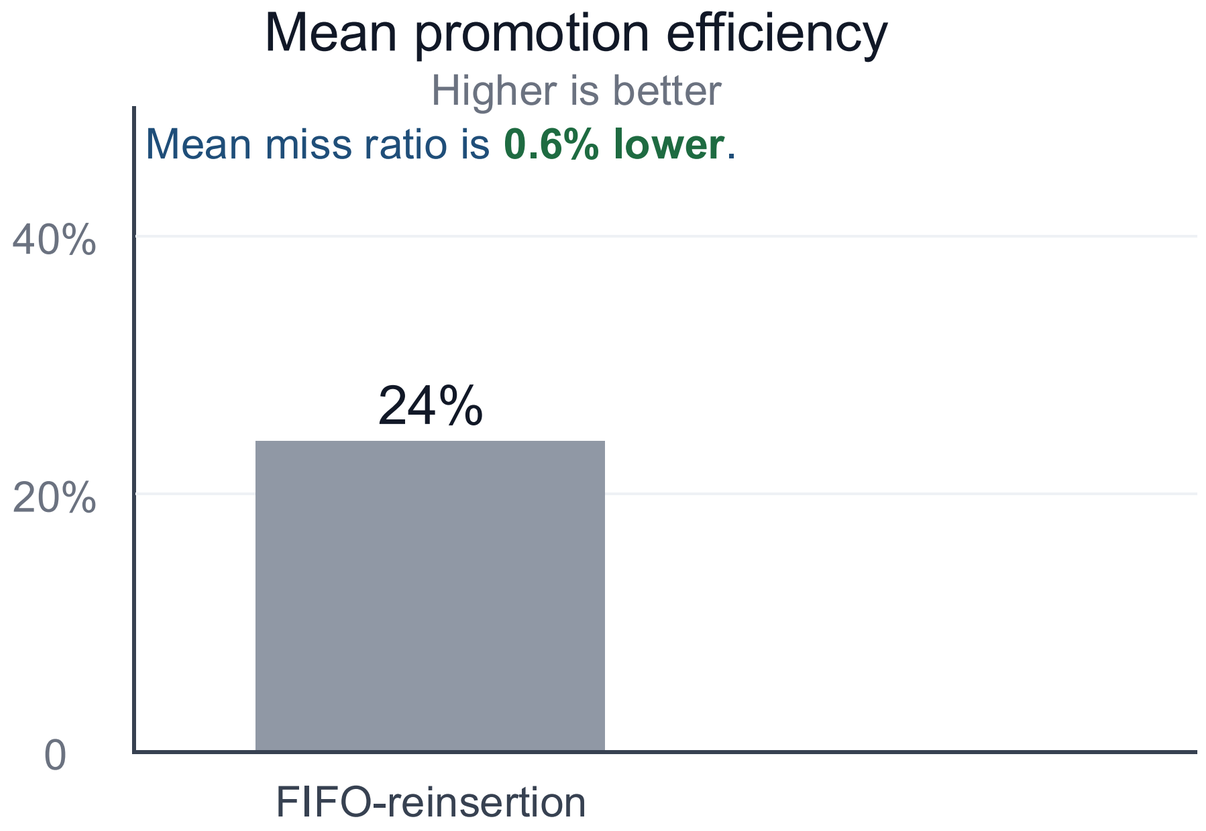


D-FR doubles promotion efficiency

- Delayed FIFO-reinsertion (D-FR) adds a delay window to FIFO-reinsertion.
  - A hit inside the window does not set the marker.

$$\text{Promotion efficiency} = \frac{\text{misses avoided}}{\text{promotions}}$$

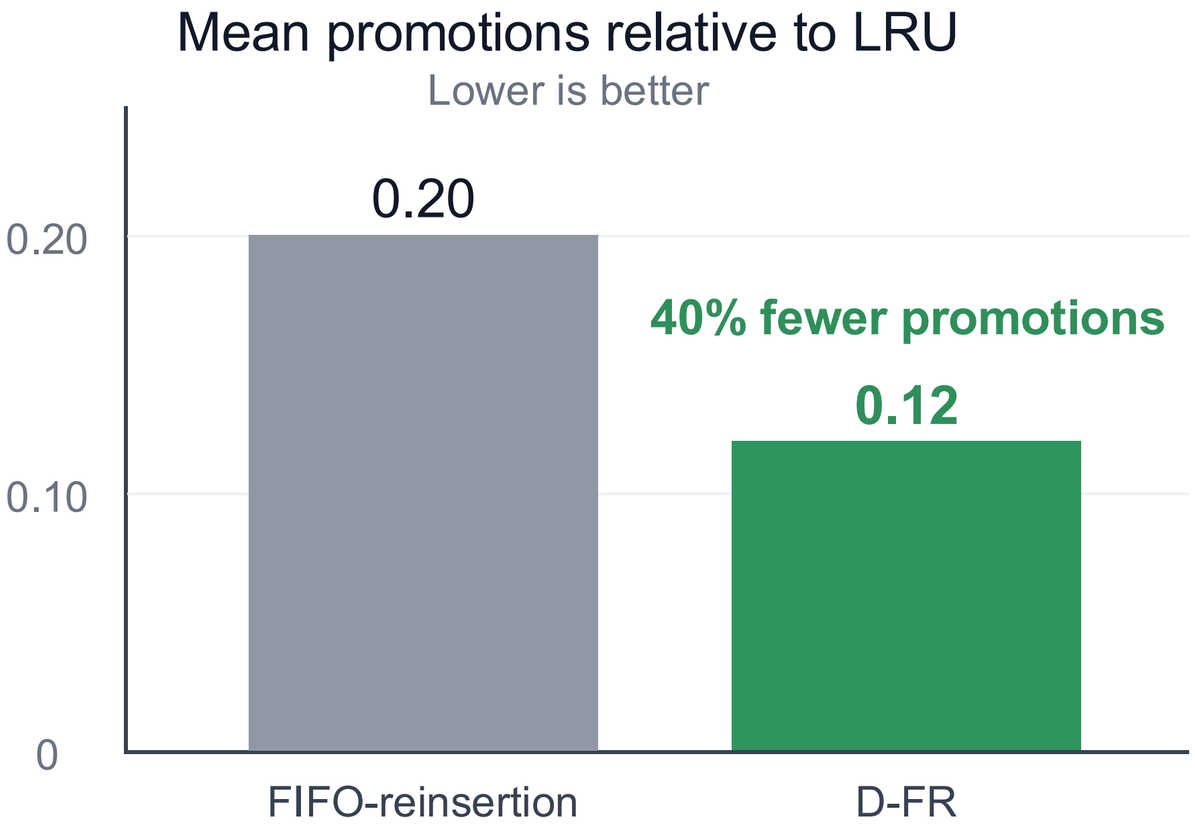


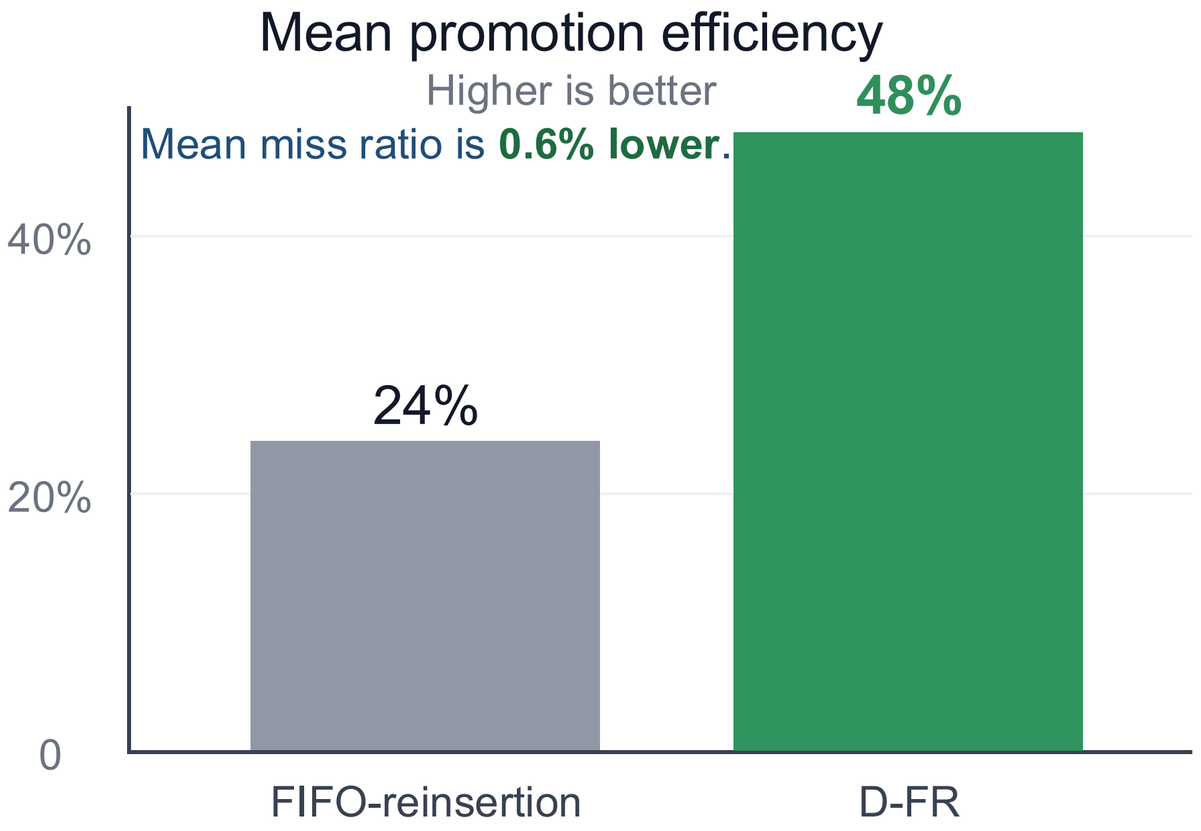


D-FR doubles promotion efficiency

- Delayed FIFO-reinsertion (D-FR) adds a delay window to FIFO-reinsertion.
  - A hit inside the window does not set the marker.

$$\text{Promotion efficiency} = \frac{\text{misses avoided}}{\text{promotions}}$$

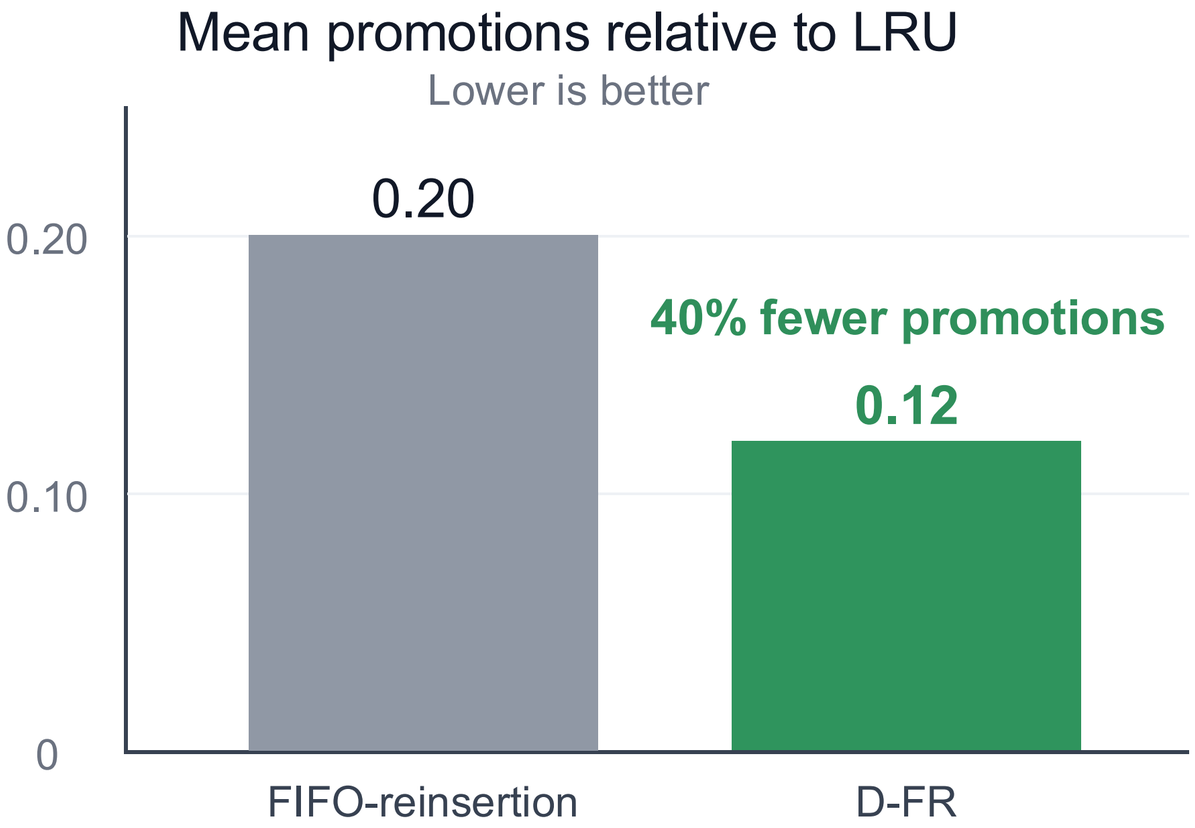


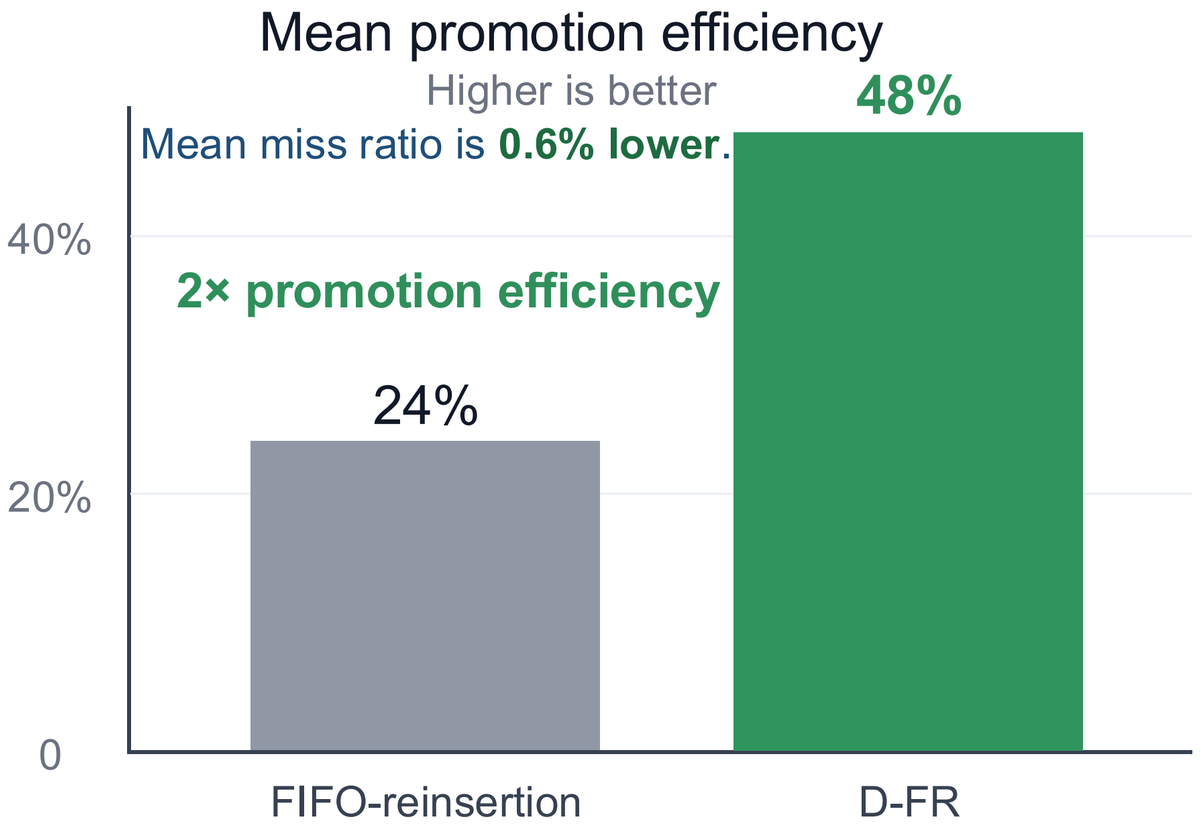


D-FR doubles promotion efficiency

- Delayed FIFO-reinsertion (D-FR) adds a delay window to FIFO-reinsertion.
  - A hit inside the window does not set the marker.

$$\text{Promotion efficiency} = \frac{\text{misses avoided}}{\text{promotions}}$$





More in the paper

The paper also studies another design, other cache policies, and the remaining headroom.

- **Age-Guided Eviction (AGE)**
  - Skip reinsertion when an object has not been accessed recently.
- **Beyond LRU**
  - Evaluate lazy promotion with ARC and 2Q.
- **Oracle analysis**
  - Estimate how many more promotions could be skipped.

The five deployed techniques behave very differently

- We present the first comprehensive evaluation of five deployed techniques.

Ziyue Qiu: ziyueqiu@cs.cmu.edu · ziyueqiu.github.io
Juncheng Yang: juncheng@seas.harvard.edu · junchengyang.com
CacheMon: **cachemon.org**

cachemon.org

The five deployed techniques behave very differently

- We present the first comprehensive evaluation of five deployed techniques.

- Their different performance trade-offs reveal which promotions are safe to skip.

Ziyue Qiu: ziyueqiu@cs.cmu.edu · ziyueqiu.github.io
Juncheng Yang: juncheng@seas.harvard.edu · junchengyang.com
CacheMon: **cachemon.org**

cachemon.org

The five deployed techniques behave very differently

- We present the first comprehensive evaluation of five deployed techniques.
- Their different performance trade-offs reveal which promotions are safe to skip.
- D-FR combines Delay-LRU's window with FIFO-reinsertion.

Ziyue Qiu: ziyueqiu@cs.cmu.edu · ziyueqiu.github.io
Juncheng Yang: juncheng@seas.harvard.edu · junchengyang.com
CacheMon: **cachemon.org**

cachemon.org

The five deployed techniques behave very differently

- We present the first comprehensive evaluation of five deployed techniques.
- Their different performance trade-offs reveal which promotions are safe to skip.
- D-FR combines Delay-LRU's window with FIFO-reinsertion.
  - vs. FIFO-reinsertion: **40%** fewer promotions, **2×** promotion efficiency, **0.6%** lower mean miss ratio.

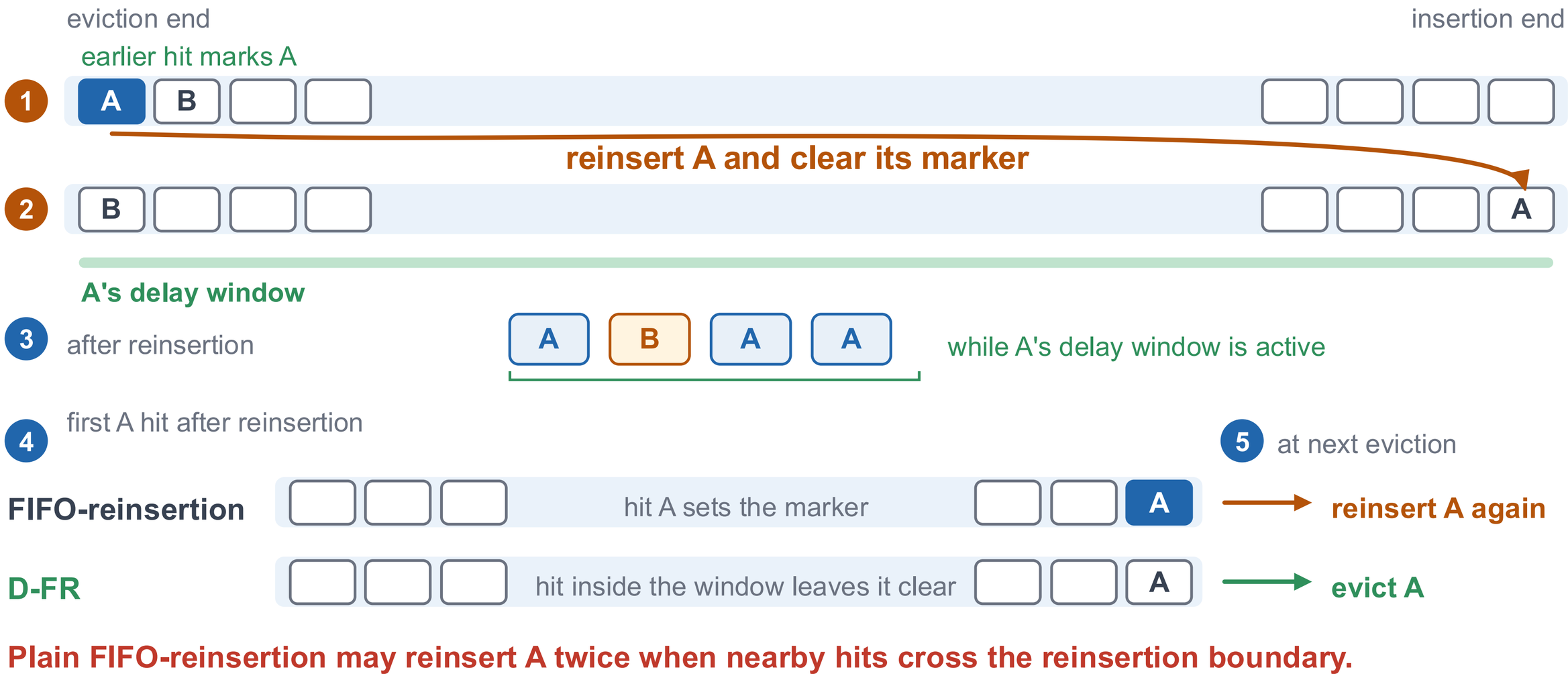
Ziyue Qiu: ziyueqiu@cs.cmu.edu · ziyueqiu.github.io
Juncheng Yang: juncheng@seas.harvard.edu · junchengyang.com
CacheMon: **cachemon.org**

cachemon.org

D-FR keeps the marker clear in the delay window

Backup

- FIFO-reinsertion sets the marker on every hit.
  - **D-FR** leaves it clear when the object is still inside its delay window.



Plain FIFO-reinsertion may reinsert A twice when nearby hits cross the reinsertion boundary.