Latency-Aware Caching With Delayed Hits: From Bursty Traffic to Pipeline Architectures

Nadav Keren

Gil Einziger

Gabriel Scalosub

nadavker-bgu@pm.me

BGU

Ben-Gurion University of the Negev

ARTIFACT EVALUATED USENIX AVAILABLE

ARTIFACT EVALUATED USENIX FUNCTIONAL

ARTIFACT EVALUATED USENIX REPRODUCED

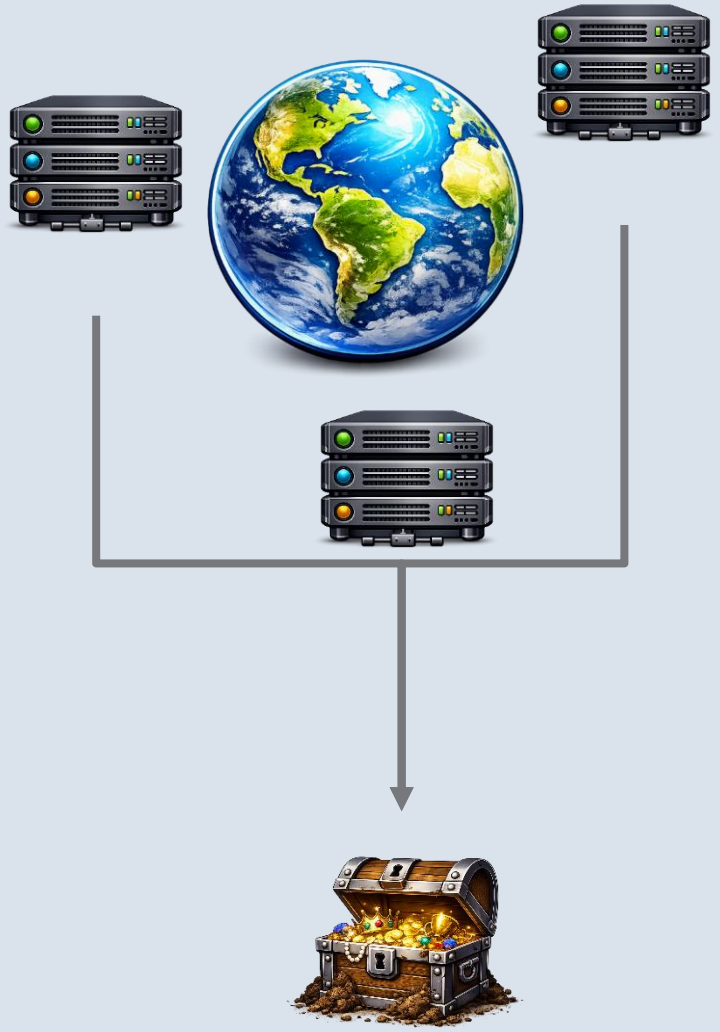
Agenda

- Motivation
- Our Contribution
  - **New Caching Architecture:** Adaptive Pipeline Cache
  - **A New Policy:** LBU
- Experimental Results

Latency Aware Caching (LAC)

- Consider the following setting:
  - Object storage
  - Different locations
  - Varied latencies (100ms – 1000ms)
  - Single cache

- Not all misses are the same!
  - Bursts may be an issue!



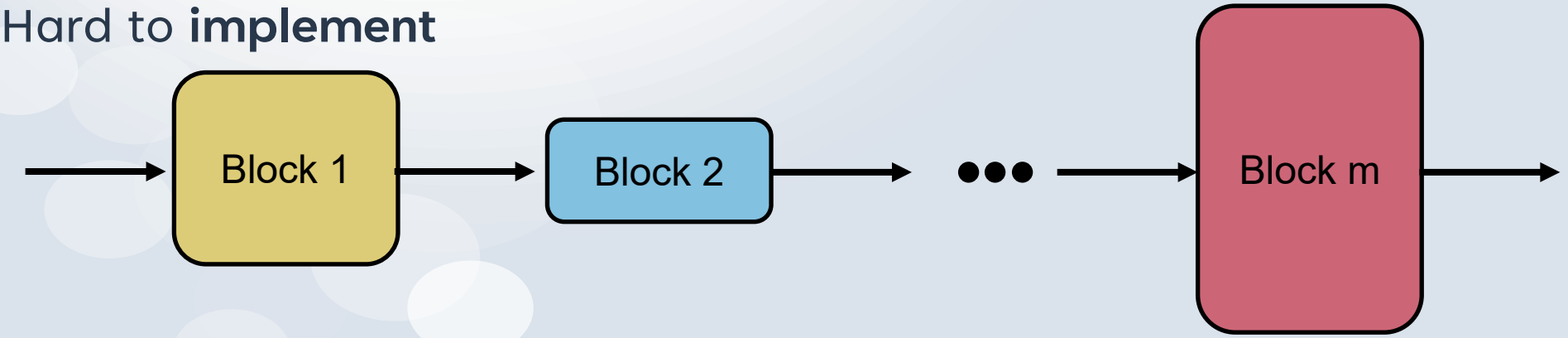
LAC: Pain Points and Our Contribution

Current Pain Points

- Ad-hoc design
- Workload-Specific
- Not always adaptive
- Monolith design
- Hard to **extend**
- Hard to **maintain**
- Hard to **implement**

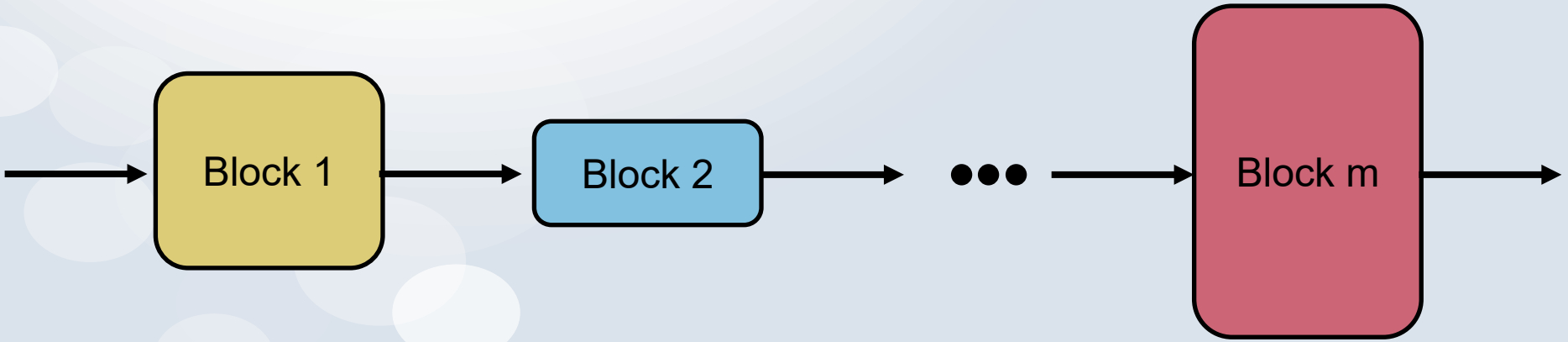
Our Solution : Adaptive Pipeline Cache

- Modular design
  - Simple building blocks
  - “Plug and Play”
- Adaptive
  - Workload aware
- Consistent Performance

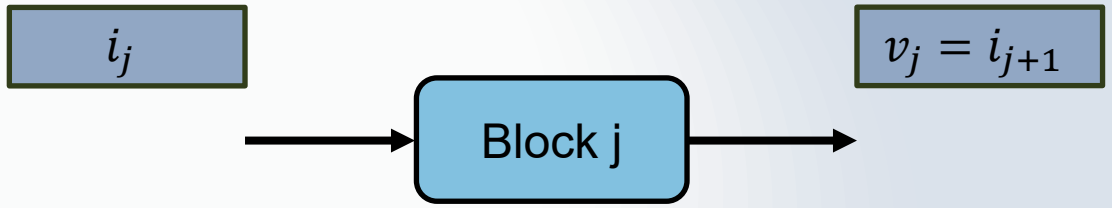


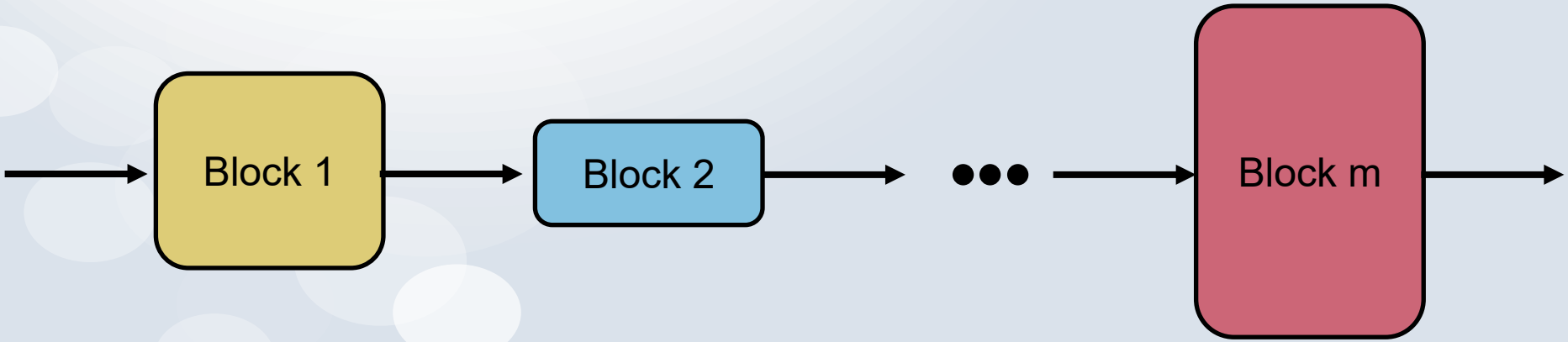
Pipeline Cache

- Block j allocation:
  - q_j quanta
- Allocation granularity (quantum):
  - k items (unit-sized)
- The total cache size is $M = k \cdot \sum_j q_j$

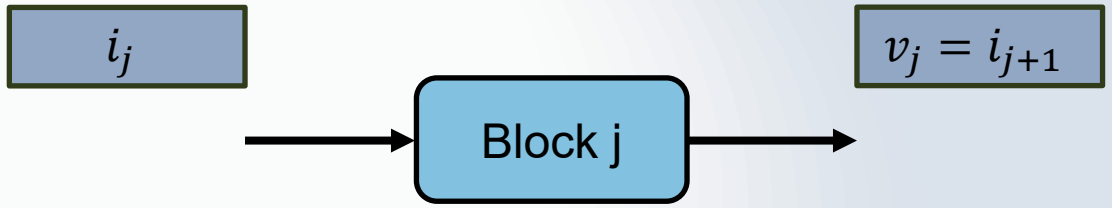


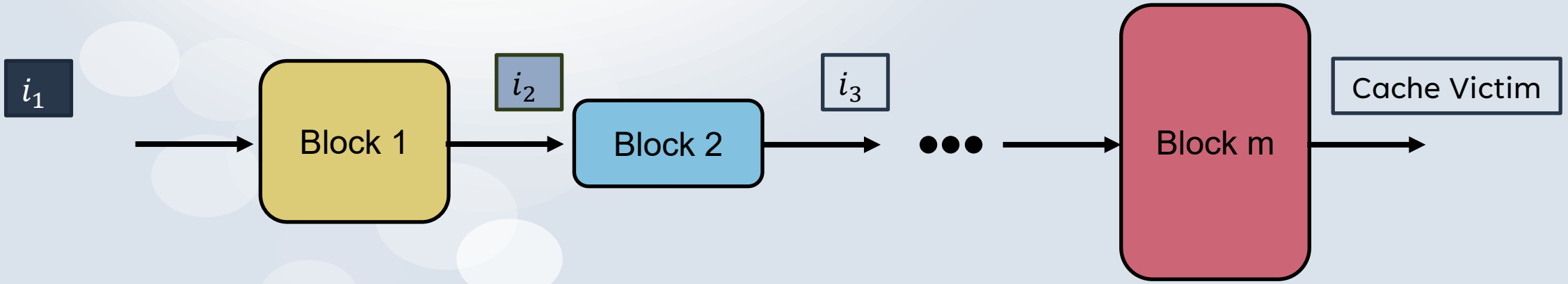
An Arrival of a (New) Item





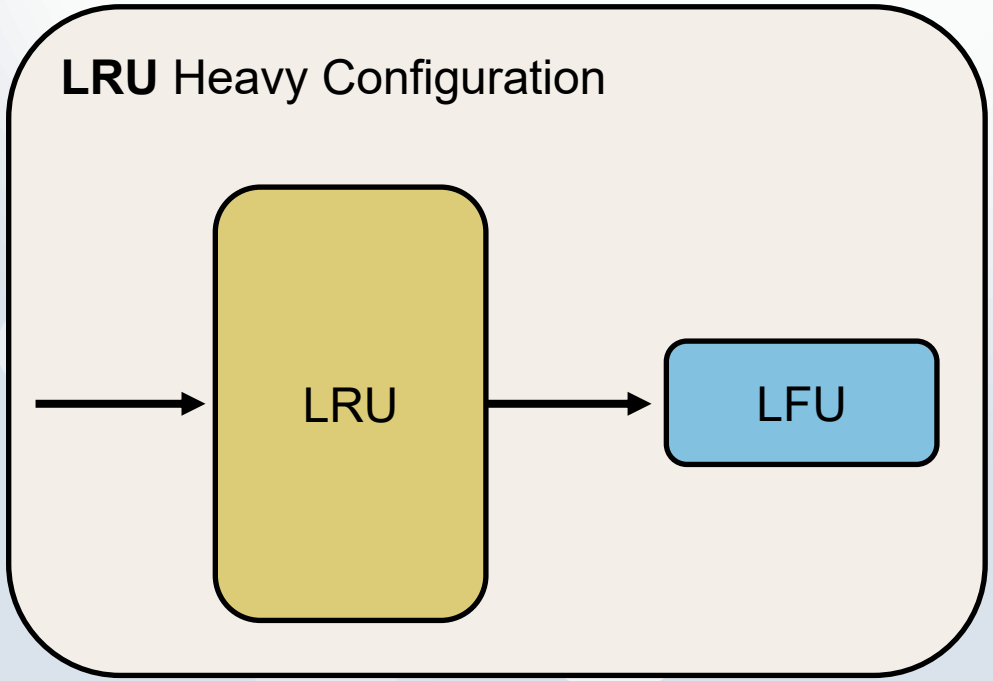
An Arrival of a (New) Item

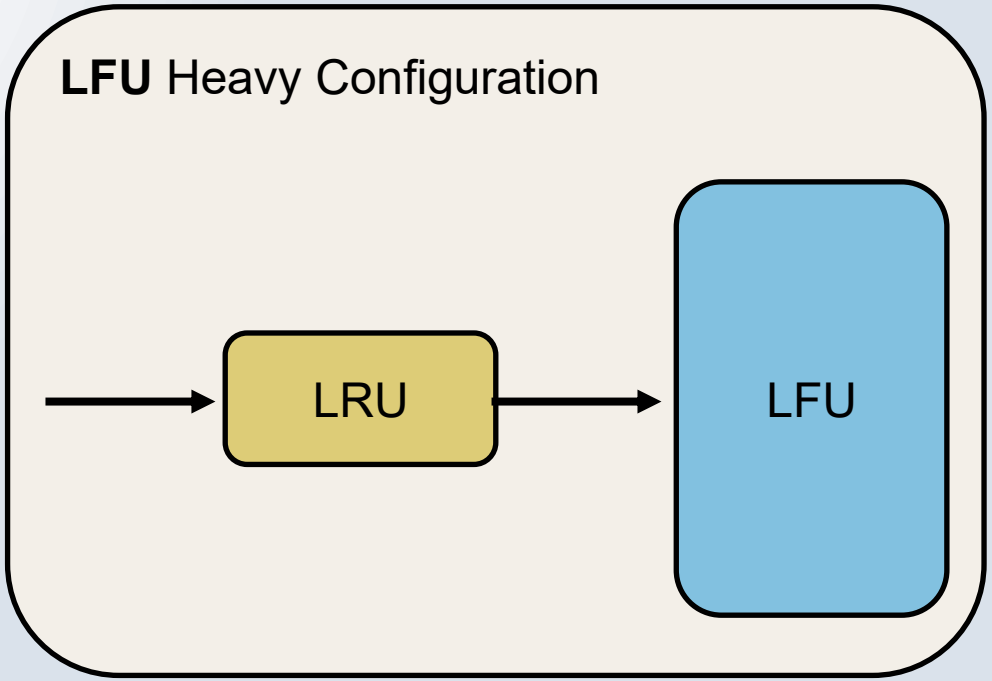




Adaptive Resizing

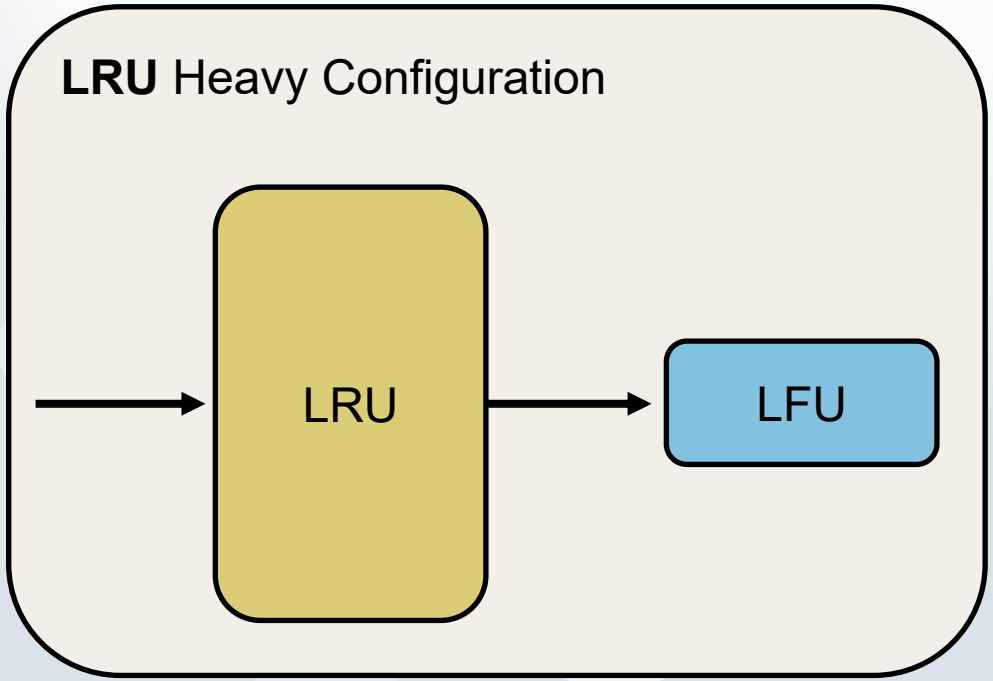
Which Configuration is better?

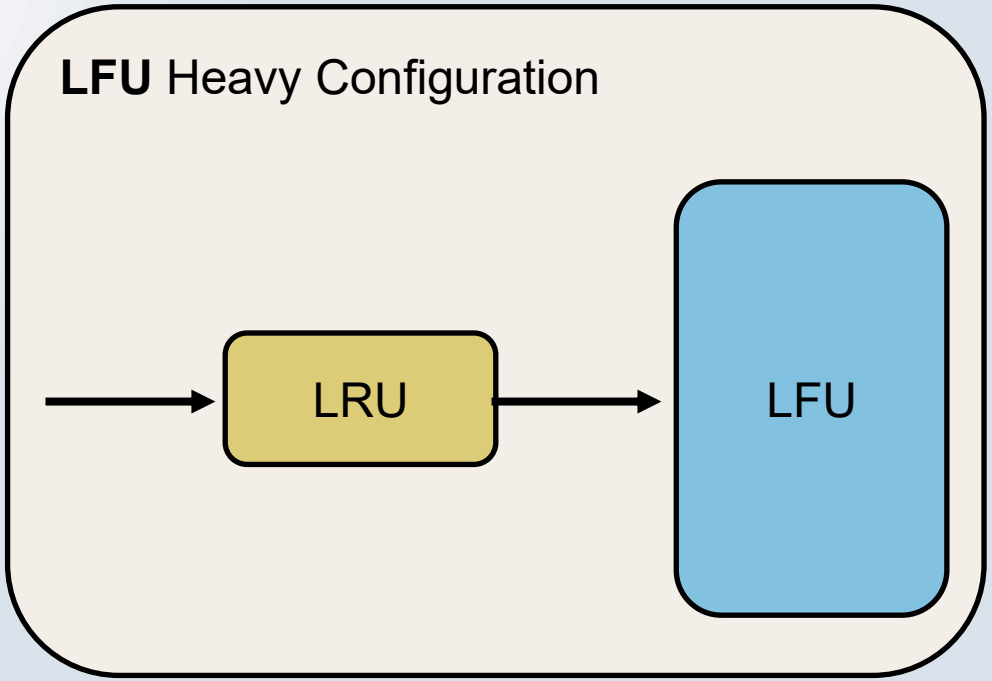




Adaptive Resizing

At runtime: Adapt to best performing configuration



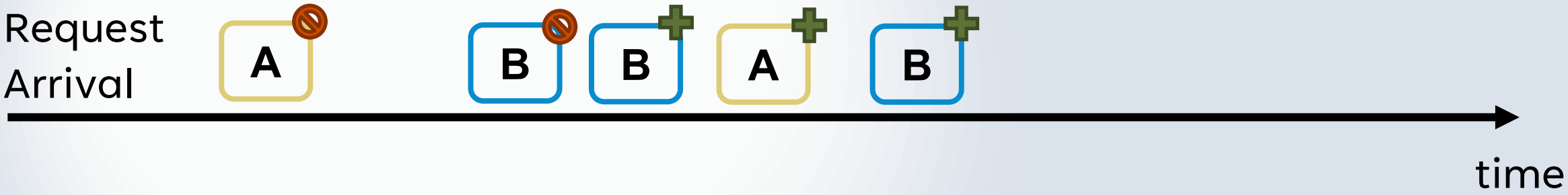


Design Challenge: Factoring in Delayed Hits

Recent work: ASWB@SIGCOMM’20, YL@ATC’22, MW@CoRR’20

Design Challenge: Factoring in Delayed Hits

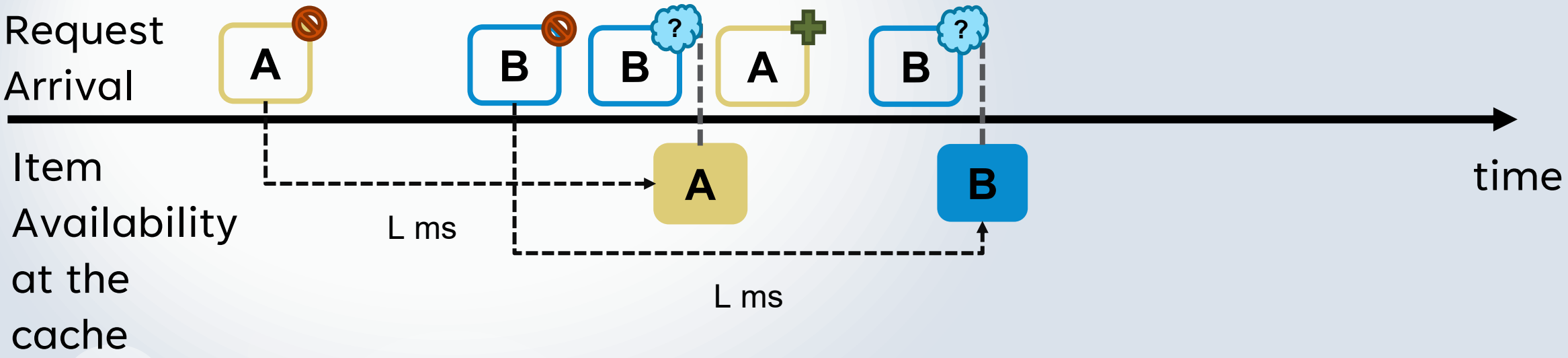
Classical caching:



Ignoring the item availability in the cache

Design Challenge: Factoring in Delayed Hits

Latency-Aware caching:

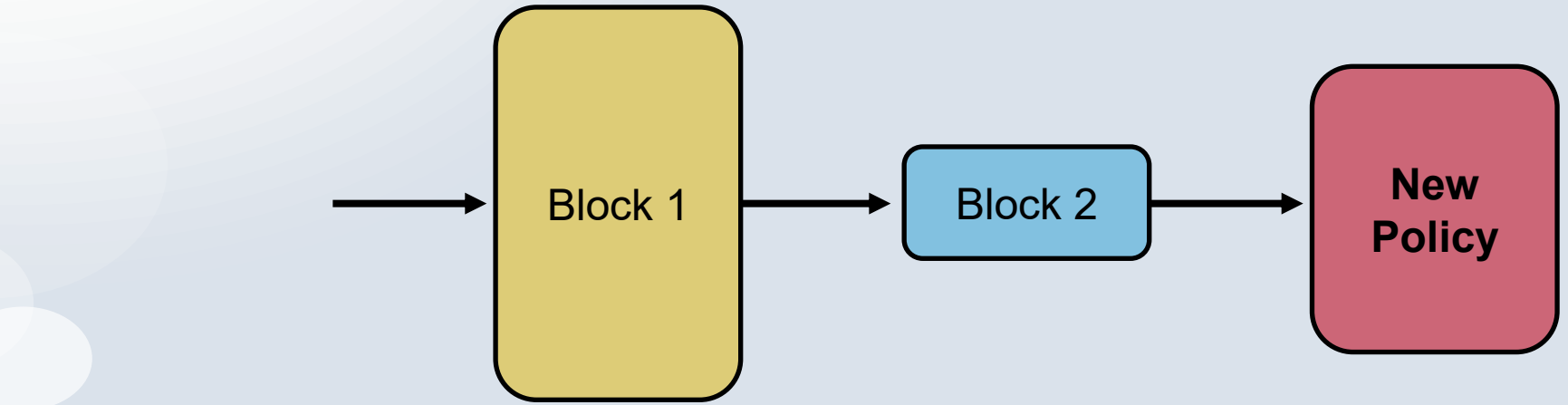


We pay more for B

If B always arrives in bursts: keep it in the cache!

Case Study: Incorporating a New Policy

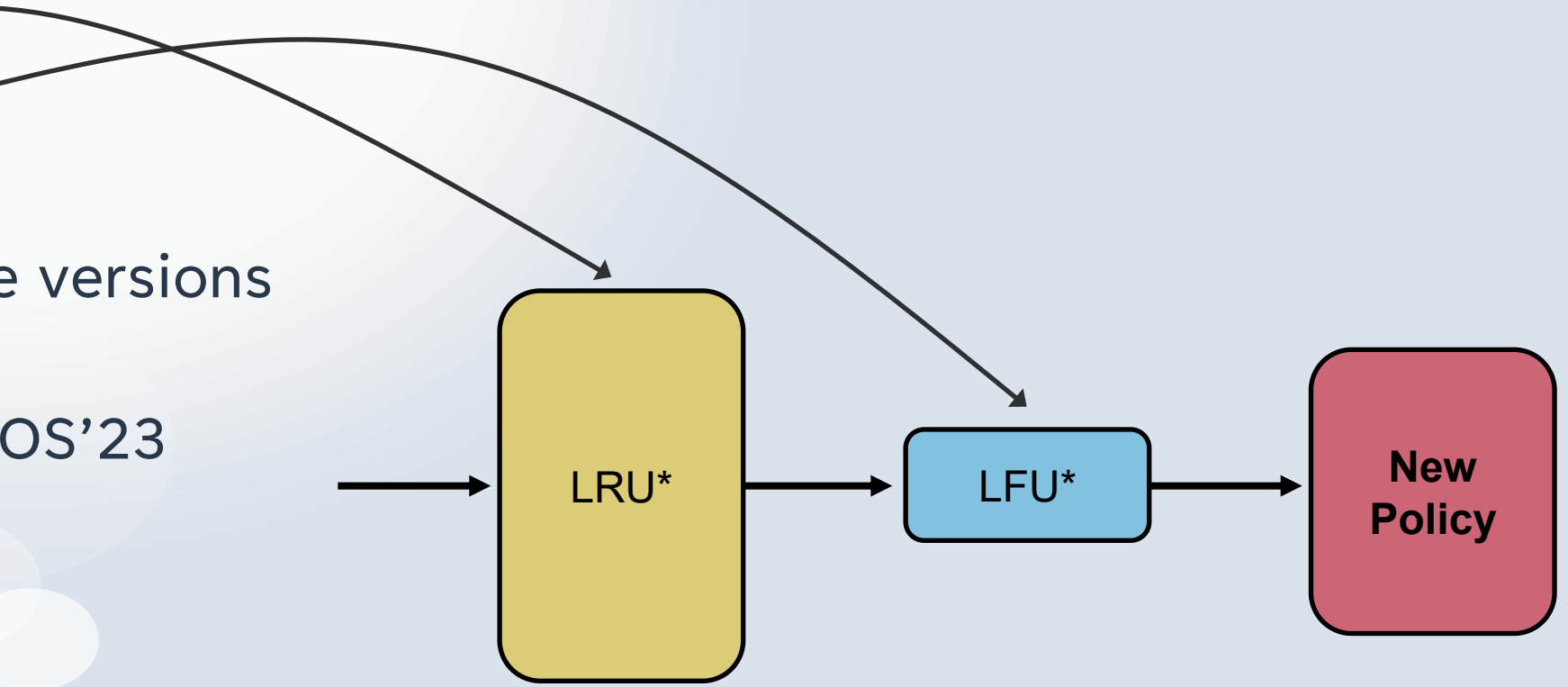
- Integrating a new specialized policy



Case Study: Incorporating a New Policy

- Integrating a new specialized policy

- LRU*/LFU*
  - Latency aware versions of LRU/LFU EHW@ACM-TOS'23

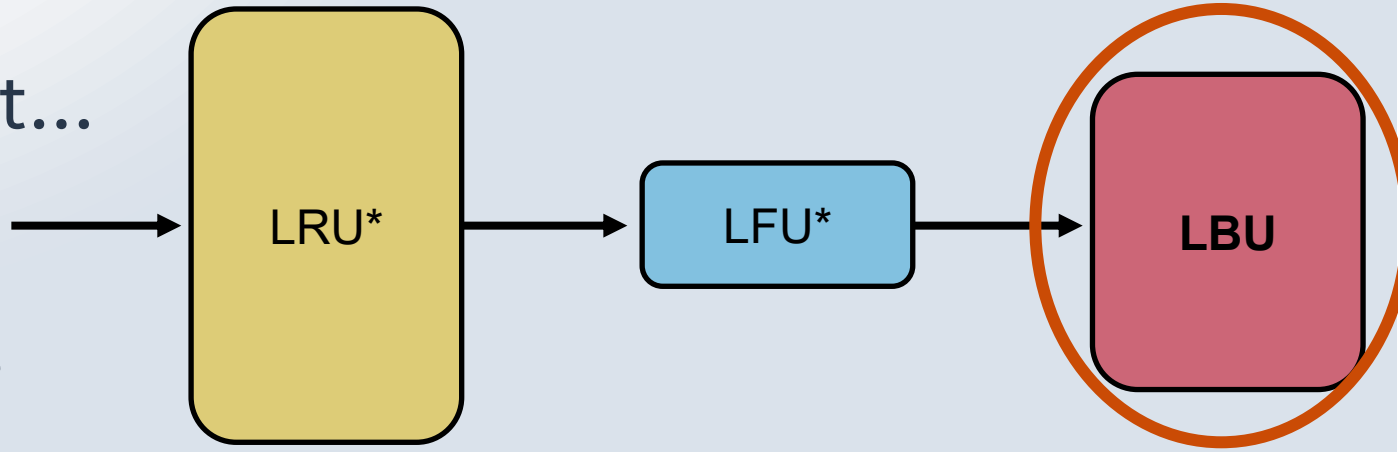


The Least Bursty Used (LBU) Policy

- Specialized
- Capturing only items that are “bursty”
- A new scoring metric
- Very situational

- LBU is **NOT** a silver bullet…
  - Captures something *new*

Next: Experimental results



The Experimental Settings

- 13 traces
  - Diverse characteristics
- 3 sources
  - IBM, Meta, Twitter

Trace
IBM10
IBM12
IBM24
IBM29
IBM31
IBM34
IBM45
Meta2
Meta4
Twitter1
Twitter3
Twitter9
Twitter28

The Experimental Settings

- 13 traces
  - Diverse characteristics
- 3 sources
  - IBM, Meta, Twitter

- Best Static RFB
  - Serves as the baseline

Trace	Best Static RFB Config
IBM10	(14,2,0)
IBM12	(1,1,14)
IBM24	(1,14,1)
IBM29	(16,0,0)
IBM31	(2,14,0)
IBM34	(13,2,1)
IBM45	(16,0,0)
Meta2	(0,15,1)
Meta4	(1,14,1)
Twitter1	(0,15,1)
Twitter3	(0,16,0)
Twitter9	(0,16,0)
Twitter28	(2,9,5)

The Results

- 12 Competitive Policies
- Average request latency across the traces
  - Compared to baseline

- LBU:
  - performs poorly on its own

Policy	Average Across Traces
GD-Wheel	123.7% ± 129.6%
LA Cache	106.2% ± 52.4%
LRU*	59.6% ± 84.3%
LBU	40.6% ± 30.9%
LFU*	33.8% ± 46.4%
LHD	31.9% ± 30.6%
ACA TinyLFU	29.0% ± 43.1%
ARC	25.1% ± 29.9%
FRD	24.1% ± 16.3%
SIEVE	22.8% ± 18.7%
Hyperbolic LA	19.3% ± 14.3%
S3FIFO	15.6% ± 12.6%
LRB	14.6% ± 16.5%

The Results

- 12 Competitive Policies
- Average performance across the traces
  - Compared to baseline

- Adaptive RF
  - **Without LBU**
    Outperforms all policies (on average)

Policy	Average Across Traces
GD-Wheel	123.7% ± 129.6%
LA Cache	106.2% ± 52.4%
LRU*	59.6% ± 84.3%
LBU	40.6% ± 30.9%
LFU*	33.8% ± 46.4%
LHD	31.9% ± 30.6%
ACA TinyLFU	29.0% ± 43.1%
ARC	25.1% ± 29.9%
FRD	24.1% ± 16.3%
SIEVE	22.8% ± 18.7%
Hyperbolic LA	19.3% ± 14.3%
S3FIFO	15.6% ± 12.6%
LRB	14.6% ± 16.5%
Adaptive RF	**7.4% ± 10.9%**

The Results

- 12 Competitive Policies
- Average performance across the traces
  - Compared to baseline

- Adaptive RF
  - Without LBU
    Outperforms all policies (on average)

- Adaptive RFB: SoTA!!!
  - 10% better than LRB

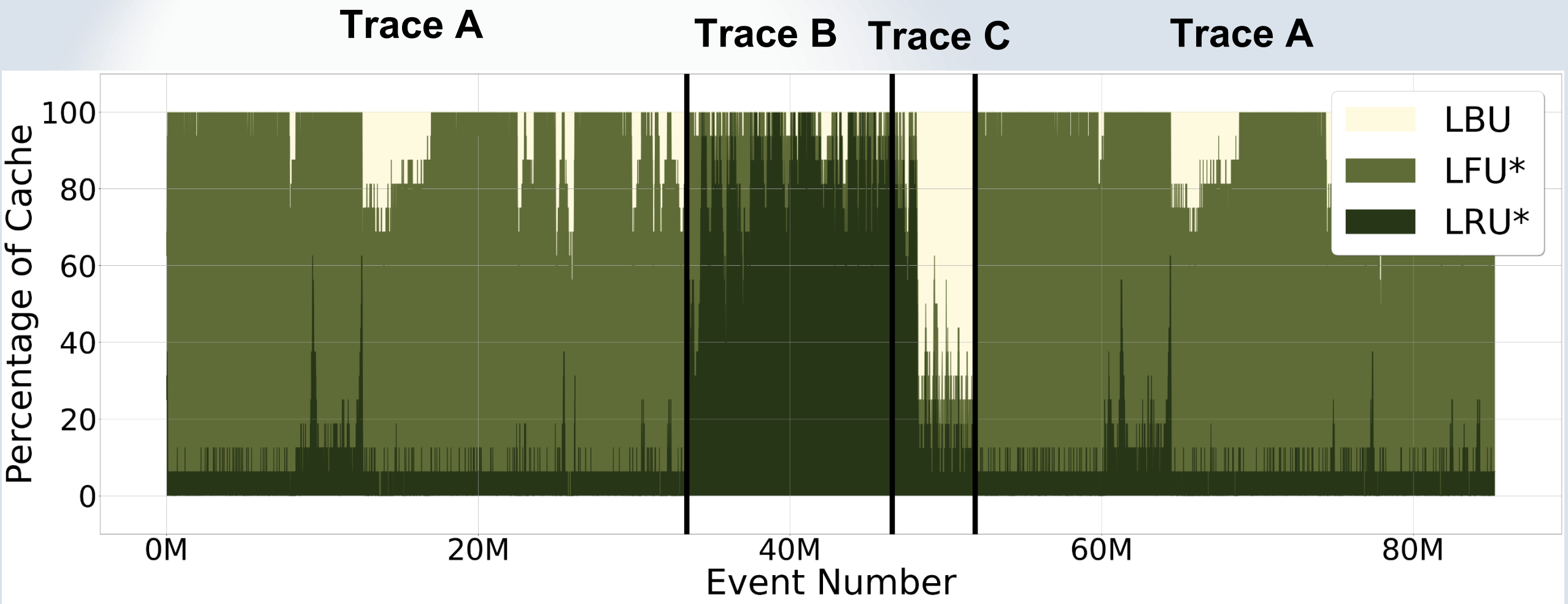
Policy	Average Across Traces
GD-Wheel	123.7% ± 129.6%
LA Cache	106.2% ± 52.4%
LRU*	59.6% ± 84.3%
LBU	40.6% ± 30.9%
LFU*	33.8% ± 46.4%
LHD	31.9% ± 30.6%
ACA TinyLFU	29.0% ± 43.1%
ARC	25.1% ± 29.9%
FRD	24.1% ± 16.3%
SIEVE	22.8% ± 18.7%
Hyperbolic LA	19.3% ± 14.3%
S3FIFO	15.6% ± 12.6%
LRB	14.6% ± 16.5%
Adaptive RF	**7.4% ± 10.9%**
Adaptive RFB	**4.8% ± 8.5%**

The Results

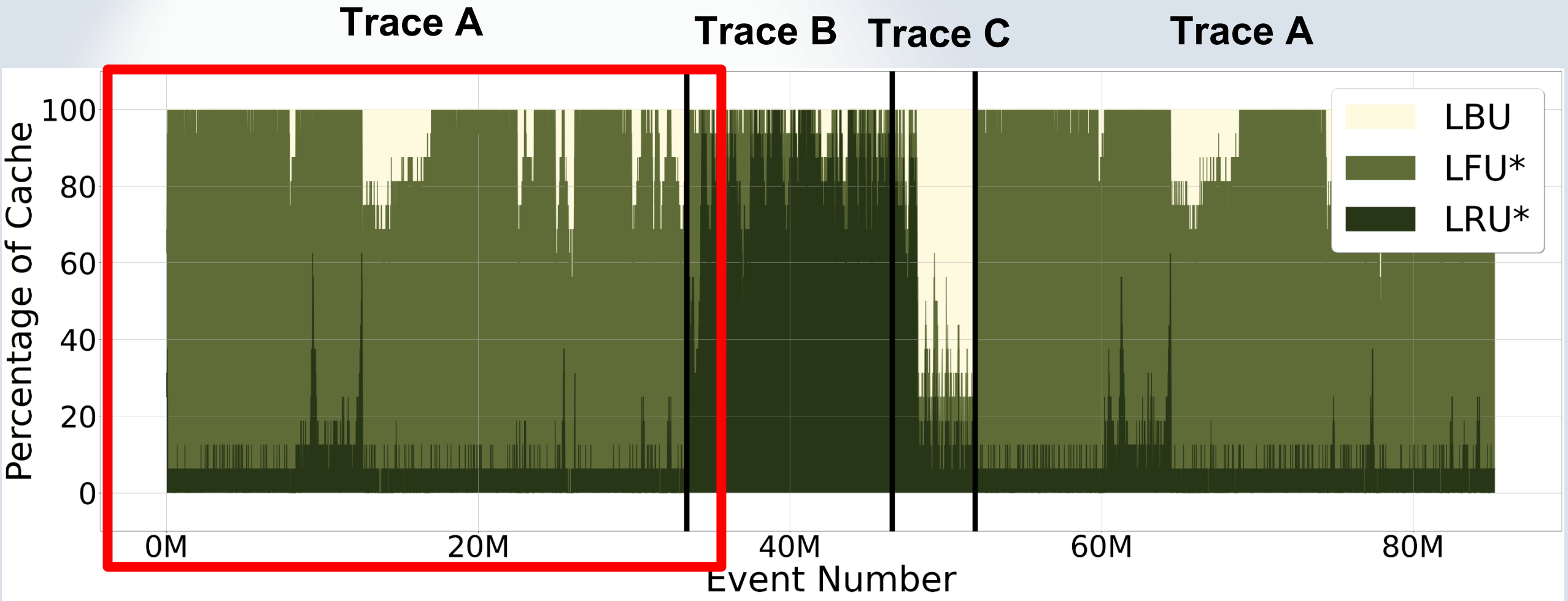
- Concatenated Trace
  - Combination of 3 traces
  - Trace A, B, C, A
- Adaptive RFB outperforms the **best static**

Policy	Average Across Traces	Concatenated Trace
GD-Wheel	123.7% ± 129.6%	37.3%
LA Cache	106.2% ± 52.4%	36.5%
LRU*	59.6% ± 84.3%	32.0%
LBU	40.6% ± 30.9%	32.7%
LFU*	33.8% ± 46.4%	8.0%
LHD	31.9% ± 30.6%	20.8%
ACA TinyLFU	29.0% ± 43.1%	5.9%
ARC	25.1% ± 29.9%	26.8%
FRD	24.1% ± 16.3%	16.2%
SIEVE	22.8% ± 18.7%	7.1%
Hyperbolic LA	19.3% ± 14.3%	18.6%
S3FIFO	15.6% ± 12.6%	9.6%
LRB	14.6% ± 16.5%	4.8%
Adaptive RF	**7.4% ± 10.9%**	**-4.0%**
Adaptive RFB	**4.8% ± 8.5%**	**-5.5%**

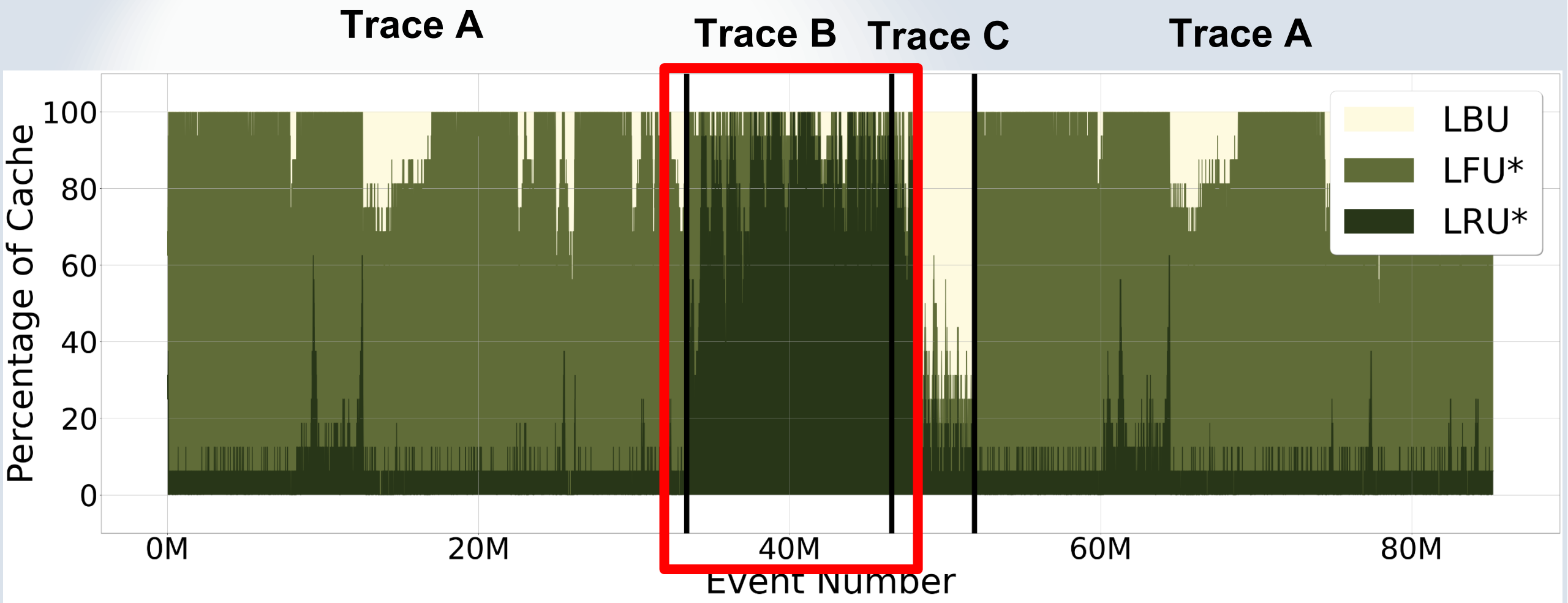
Under the Hood: Concatenated Traces



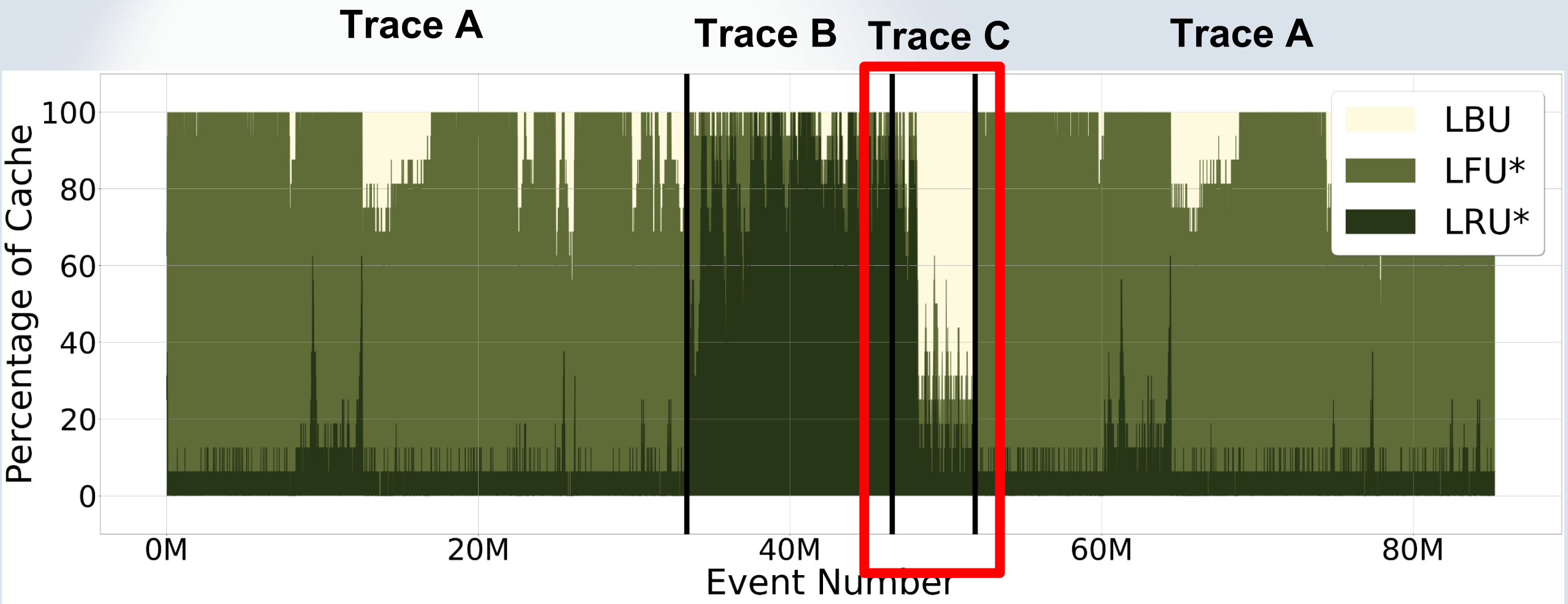
Under the Hood: Concatenated Traces



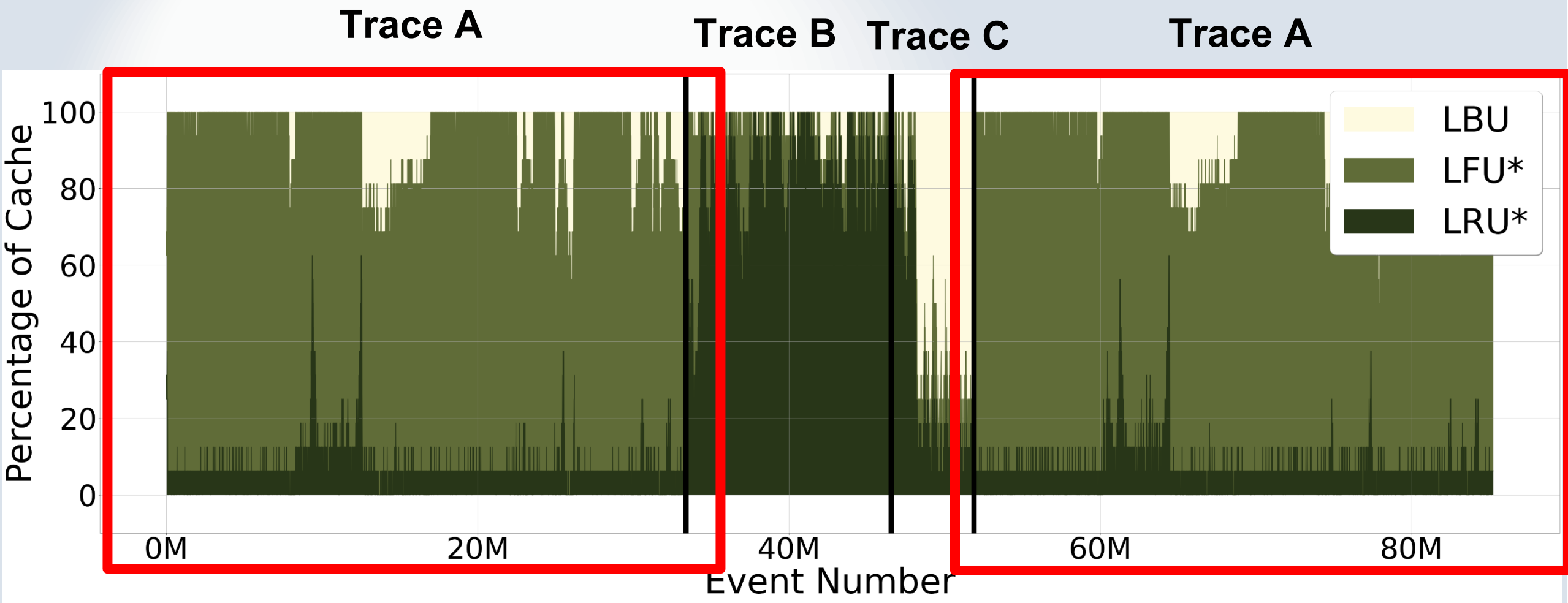
Under the Hood: Concatenated Traces



Under the Hood: Concatenated Traces



Under the Hood: Concatenated Traces



Not Covered in This Talk

- Design and Implementation details of LBU
  - Item scoring
  - LBU structure
- Hill climbing process
- Implementation optimization
  - Sampling

Summary

- New caching architecture:
  - Modular
  - Adaptive
  - Latency-Aware
- New caching policy for bursts

Artifact available at:

github.com/NadavKeren/Adaptive-Pipeline-Cache

Contact Us

nadavker-bgu@pm.me

Thank you for your time!

Questions?