Traces & code
pdl.cmu.edu/CILES

ARTIFACT EVALUATED usenix ASSOCIATION AVAILABLE

ARTIFACT EVALUATED usenix ASSOCIATION FUNCTIONAL

ARTIFACT EVALUATED usenix ASSOCIATION REPRODUCED

Baleen: ML Admission & Prefetching for Flash Caches

Daniel L.-K. Wong
wonglkd@cmu.edu

Hao Wu†, Carson Molder§, Sathya Gunasekar†, Jimmy Lu†, Snehal Khandkar†
Abhinav Sharma†, Daniel S. Berger‡, Nathan Beckmann, Gregory R. Ganger
†Meta, ‡Microsoft/UW, §UT Austin

PARALLEL DATA LABORATORY
Carnegie Mellon University

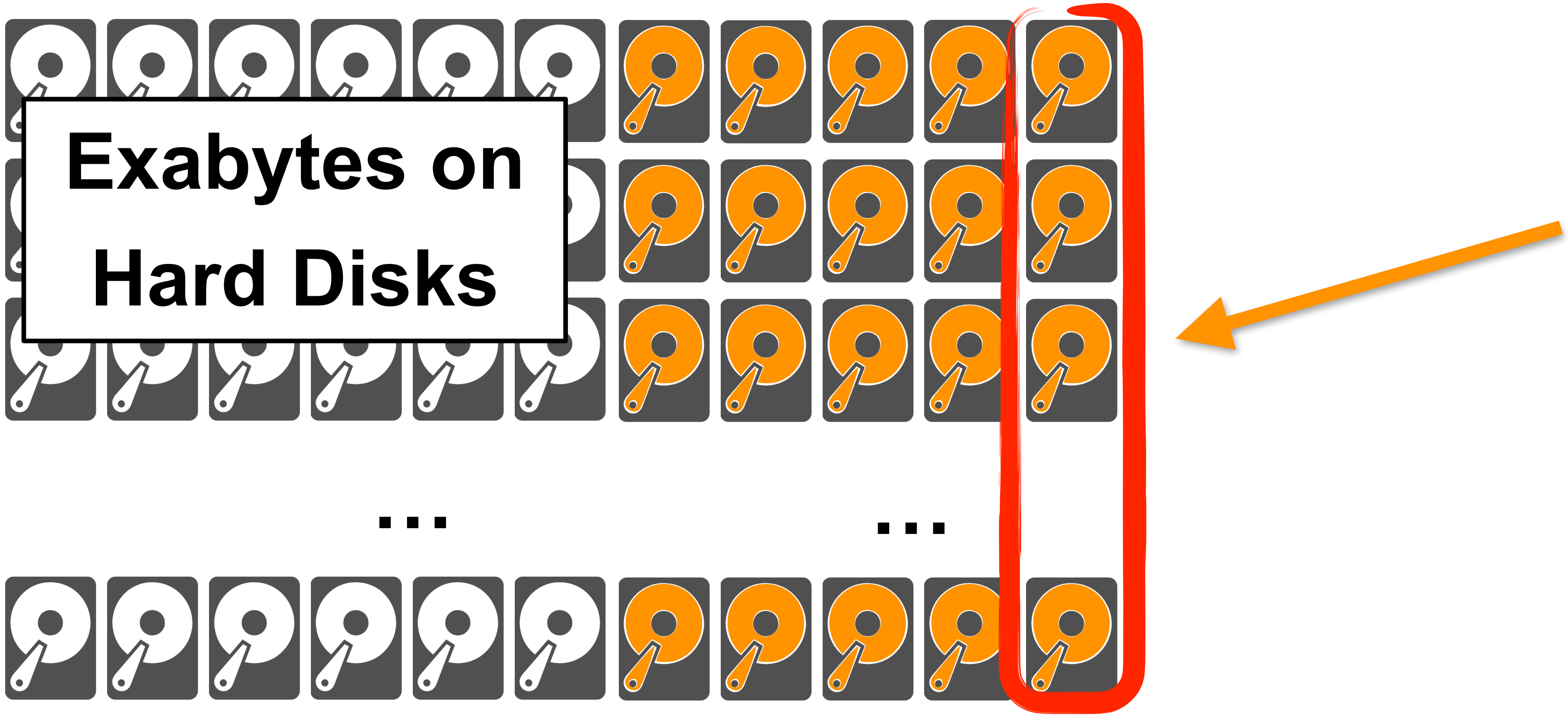
Carnegie Mellon
Parallel Data Laboratory

Meta

Bulk storage systems depend on flash caches

Bulk Storage

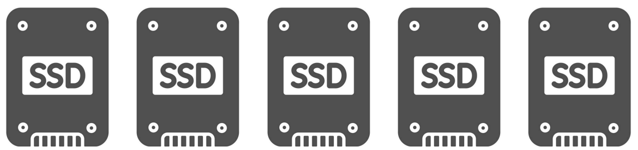
(Tectonic, Colossus, …)



Extra HDDs needed
for IOPS & bandwidth

Flash caches absorb HDD load

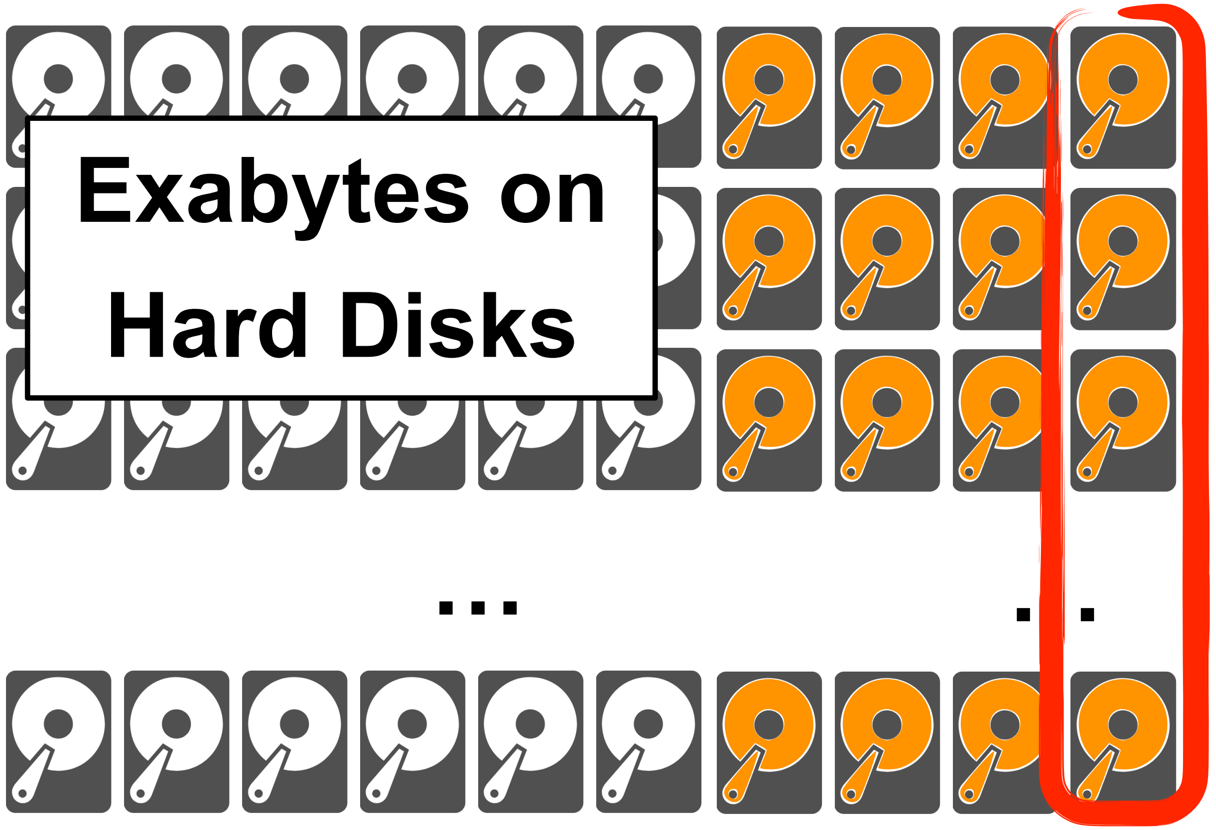
(CacheLib, …)



Better flash caches save more HDDs

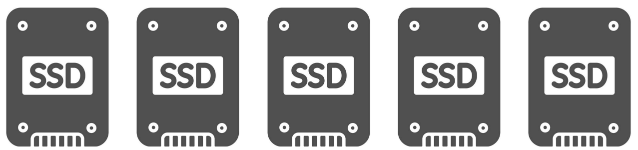
Bulk Storage

(Tectonic, Colossus, …)



Flash caches absorb HDD load

(CacheLib, …)



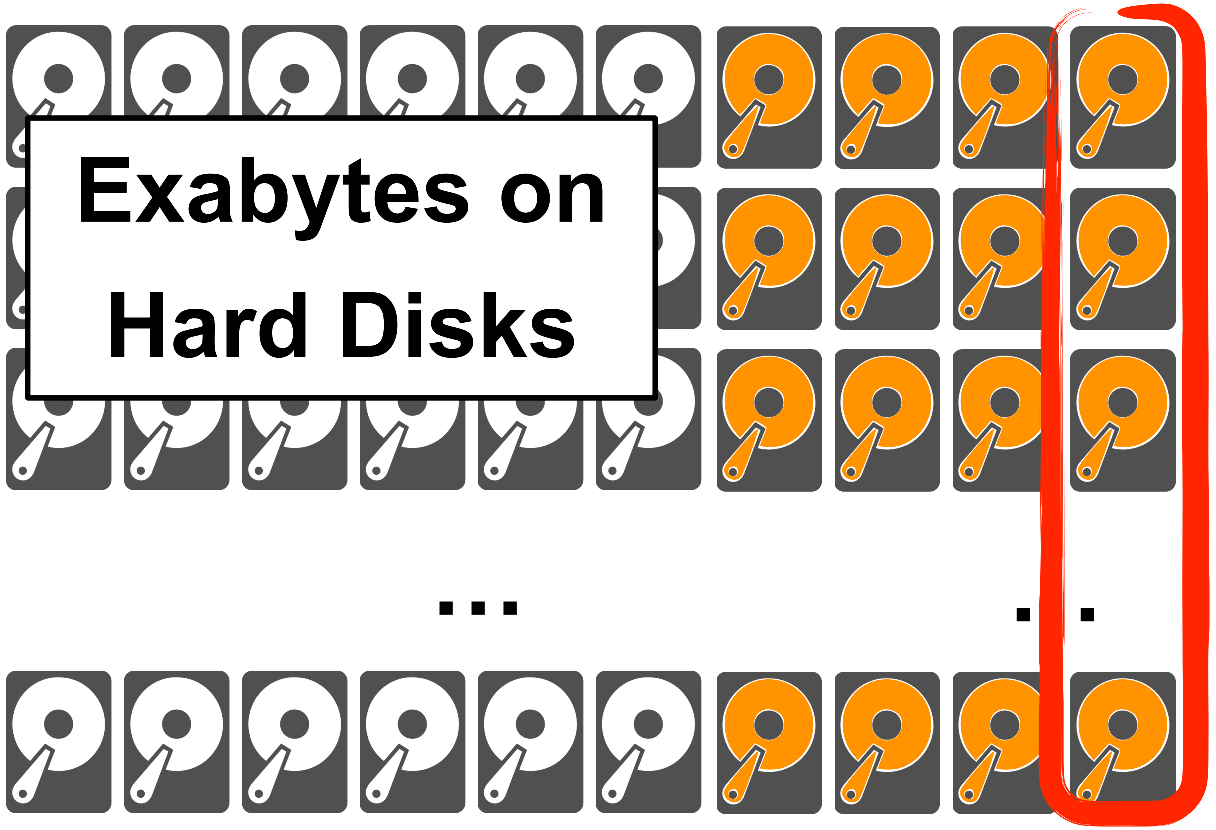
Better cache?

Extra HDDs needed for IOPS & bandwidth

Flash caches are write-heavy

Bulk Storage

(Tectonic, Colossus, …)

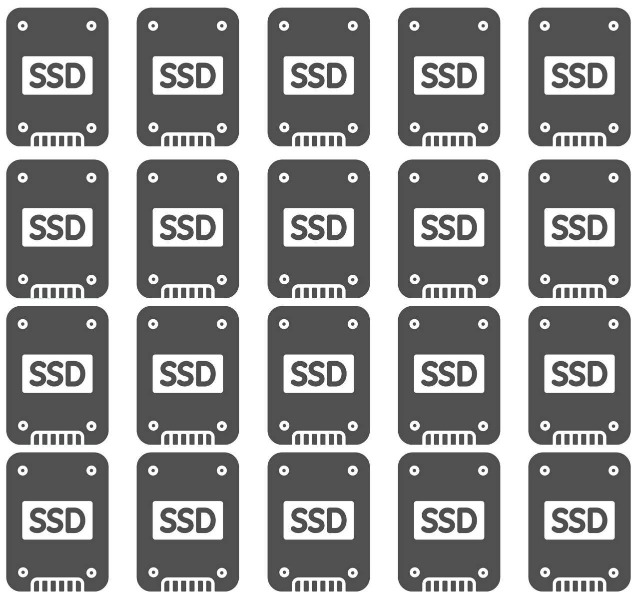


Extra HDDs needed for IOPS & bandwidth

Better cache?

Flash caches absorb HDD load

(CacheLib, …)



Problem: **Limited write endurance**

Carnegie Mellon
Parallel Data Laboratory

Costs dominated by #HDDs & #SSDs

Baleen reduces costs by 17% on 7 traces

Trend: Lower flash endurance

Even more important with denser storage!

How does Baleen reduce costs by 17%?

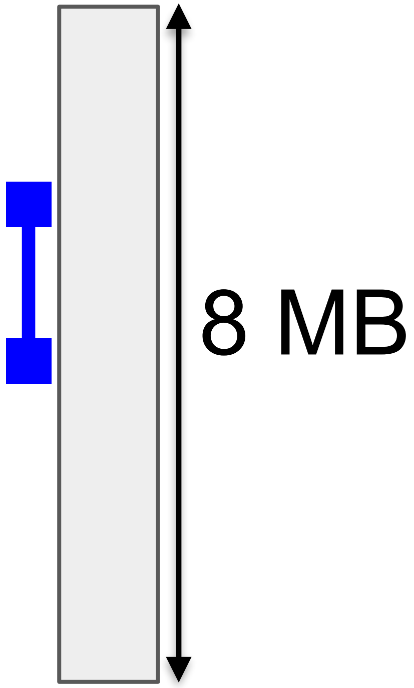
- 3 key ideas
  - Exploit a new cache residency model (**episodes**)
  - Train ML admission & ML prefetching policies
  - Optimize an end-to-end metric (disk-head time)
- *Why ML over heuristics?*
  - *More savings, more adaptive*

Bulk storage clients access byte ranges within blocks

Access

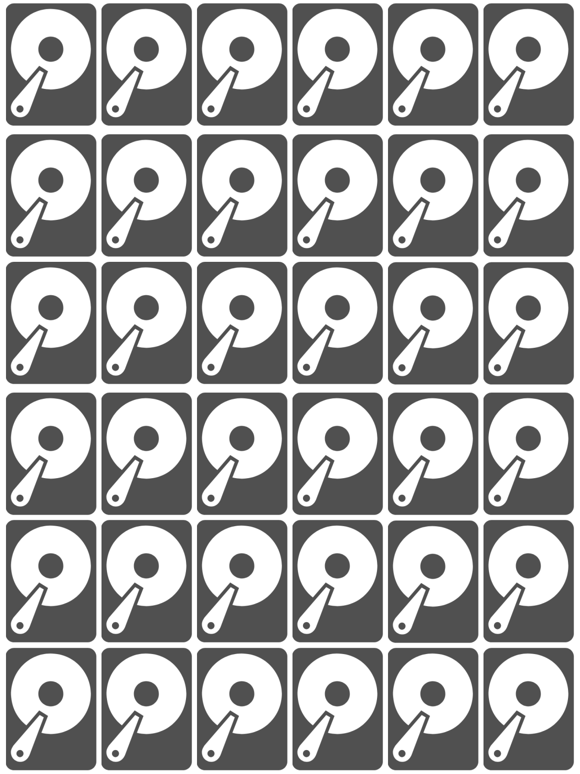
1. Block ID
2. Byte Range

Block



Host

36 hard disks



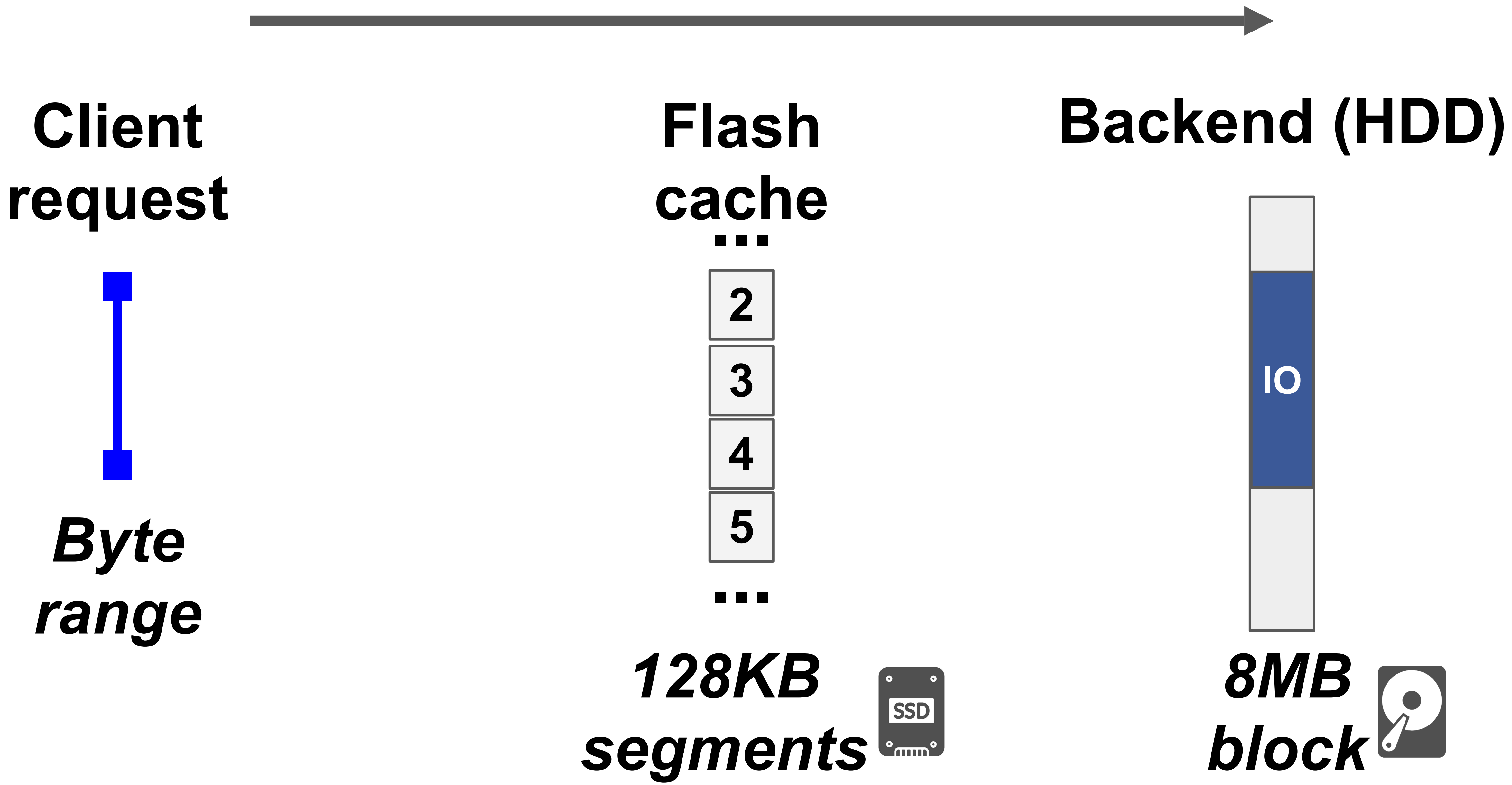
Cluster

Data center

1000s of hosts

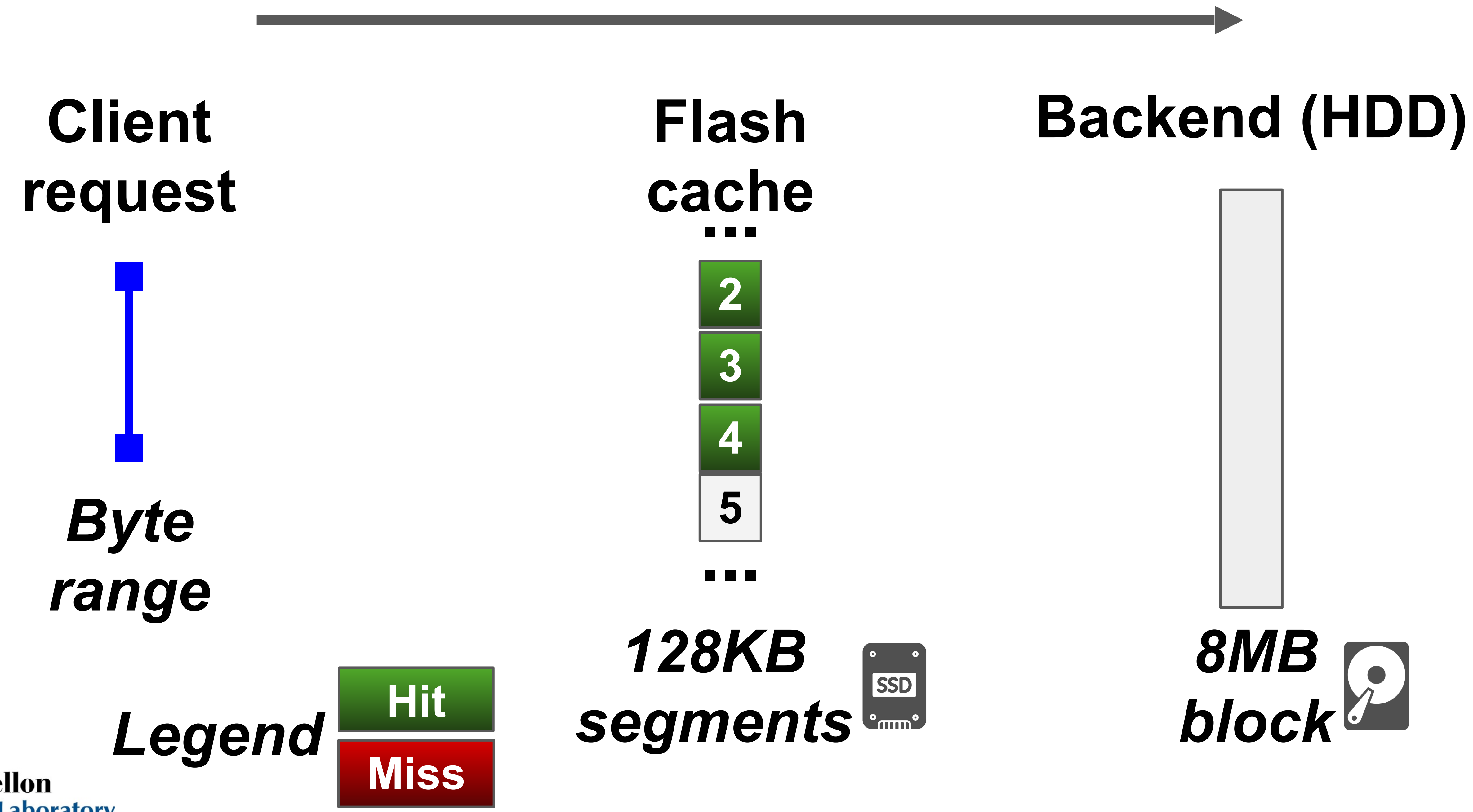
Fetching bytes from backend causes disk IO

Cache stores segments (subset of block)

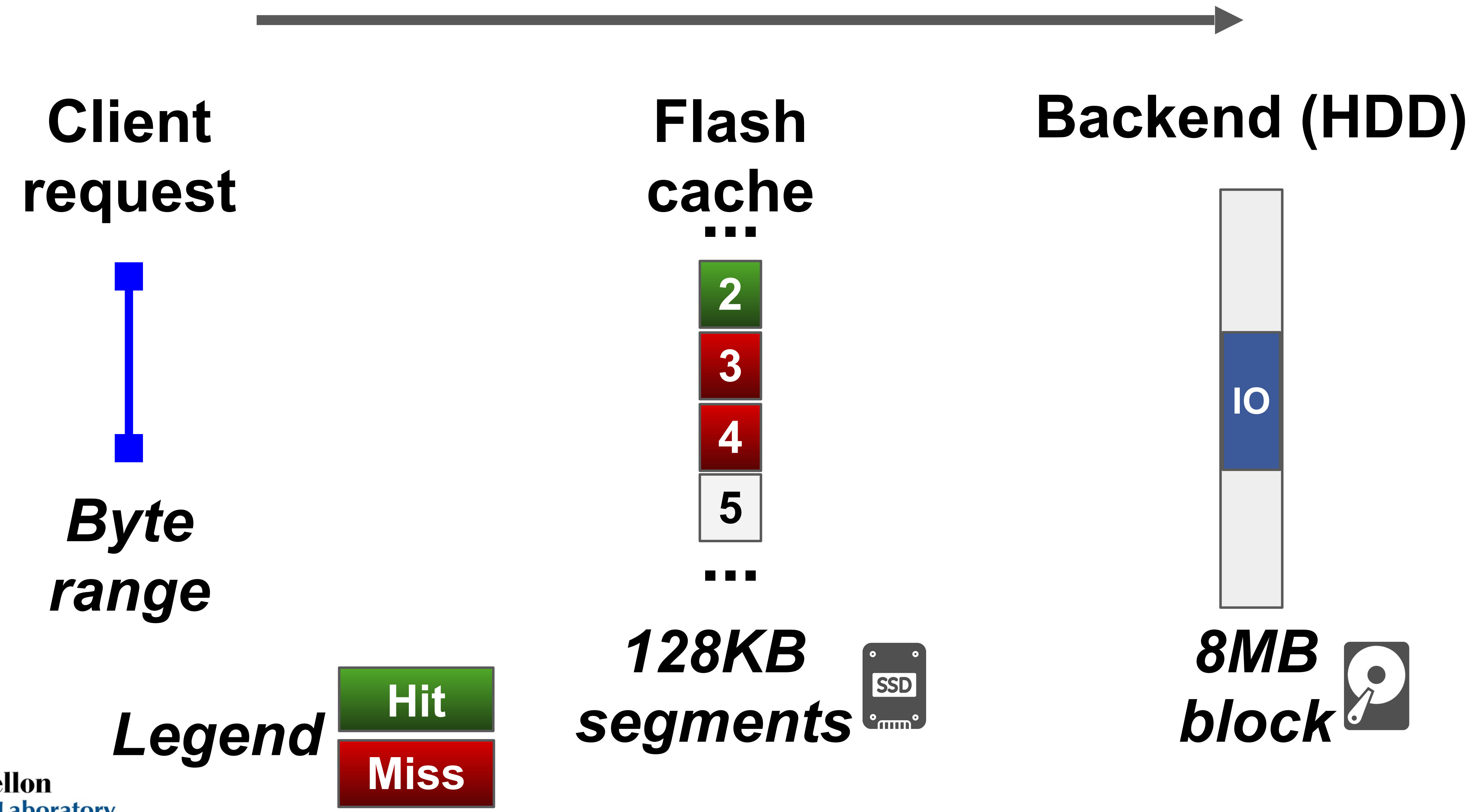


Carnegie Mellon
Parallel Data Laboratory

Cache hits save disk IO



Cache miss causes disk IO



Carnegie Mellon
Parallel Data Laboratory

Decompose flash caching into 3 decisions

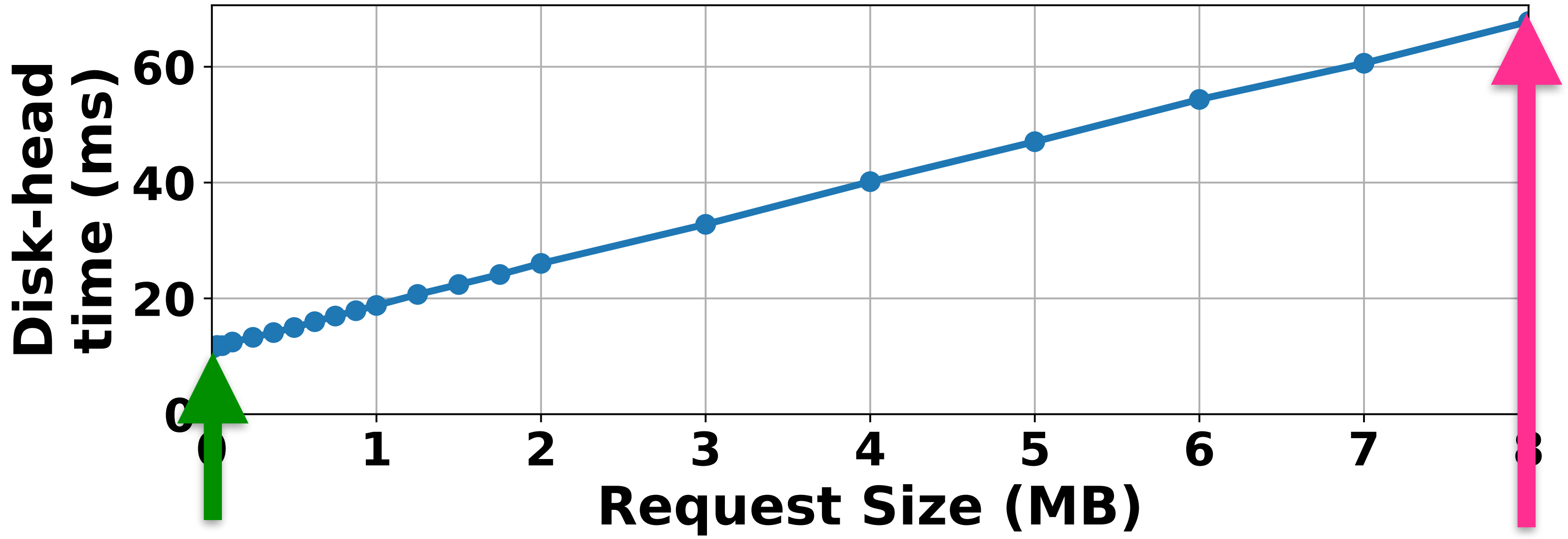
Goal: Reduce HDD load without excessive flash writes

Policy Decisions: | **Flash cache**

(a) Admit misses?	3 4	**Baleen**	… 2 3 4 5 …
(b) Prefetch?	5	**Baleen**	
(c) When to evict?		*LRU*	

Our metric: Disk-head time (DT)

- *Q:* Why DT instead of miss rates?
  - *A:* **Variable size IOs** (reducing **#IOs** & **Size of IOs** both important)
  - Using only **IO hit rate** or **byte miss rate** is an easy misstep (we did!)
- ***DT = Positioning time + Read time***



constrained by **IOPS**

constrained by **bandwidth**

- *Intuition:* DT is weighted sum of **#IOs** & **#Bytes**

Design

Episodes model

ML for caching not straightforward

Typical supervised learning

- e.g., “Is this picture a cat?”

ML for Caching

- Data: trace of accesses
- *Multiple related decisions: Admit now? Later? Never?*
  - *Depend on AND affect cache contents, future decisions*
- **Tend to overfit on easy decisions**
- **Underfit on examples at margin that distinguish policies**

Training on accesses non-trivial

What is an episode?

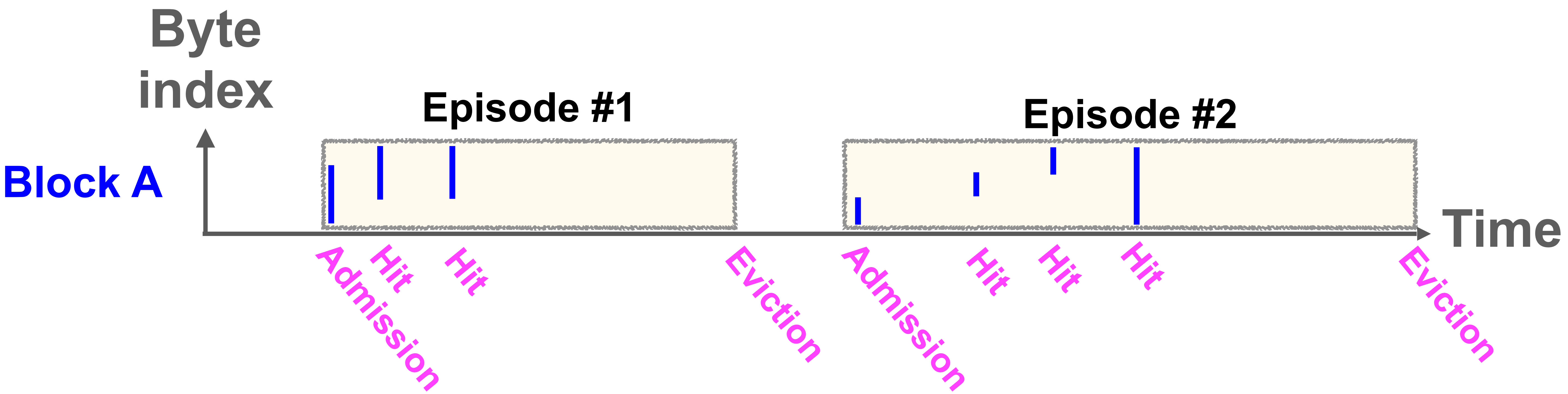
Episode:

Group of accesses corresponding to
the block's residency in flash
if you admitted it on the 1st access

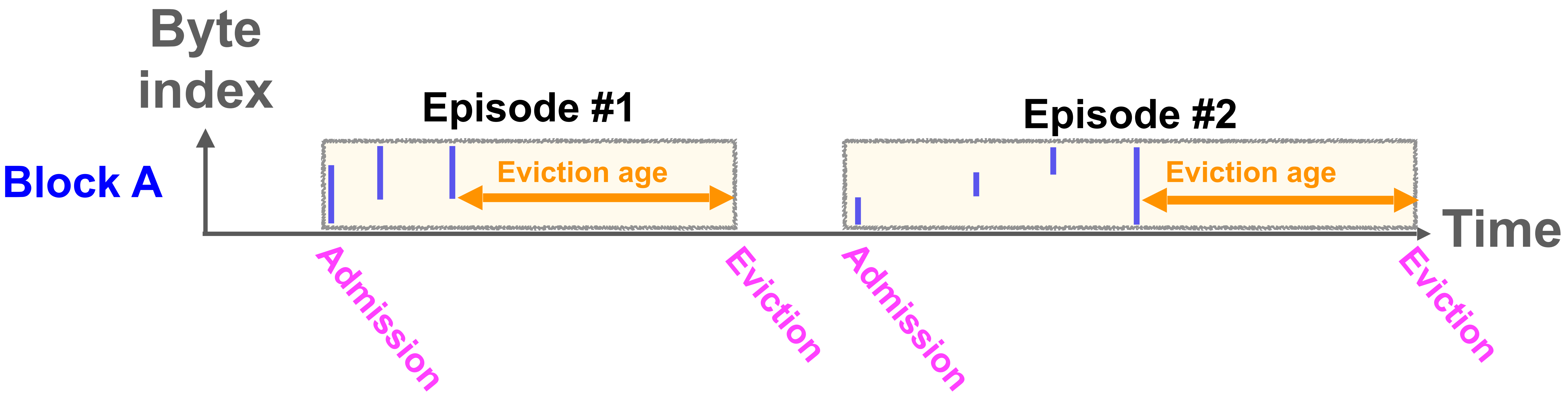
Why use **episodes** to train ML?

- **Right granularity**
  - Focus on first access instead of all accesses
  - Policies see misses, not accesses
- **Right examples**
  - Avoid overfitting on popular blocks with many accesses but only 1 miss
- **Right labels**
  - Costs & benefits defined on admission to eviction

Episodes: from admission to eviction

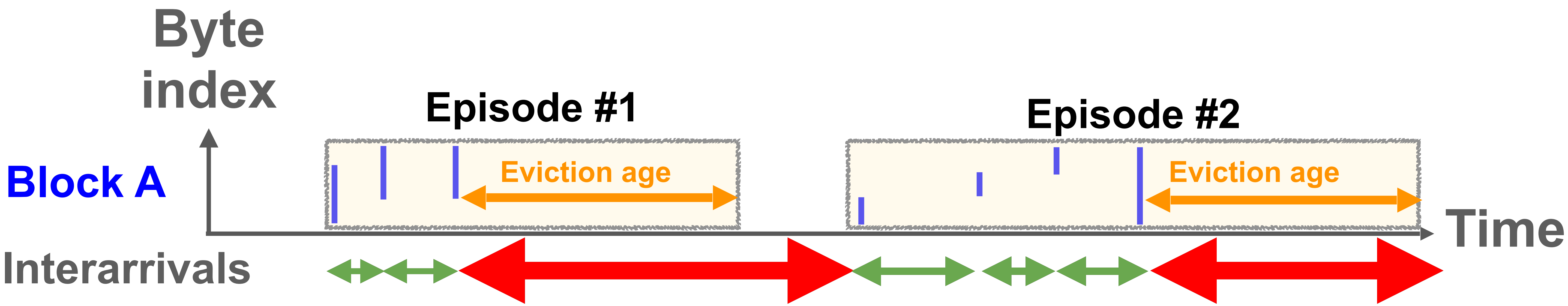


How to know when eviction happens?



How: model LRU cache state with assumed eviction age

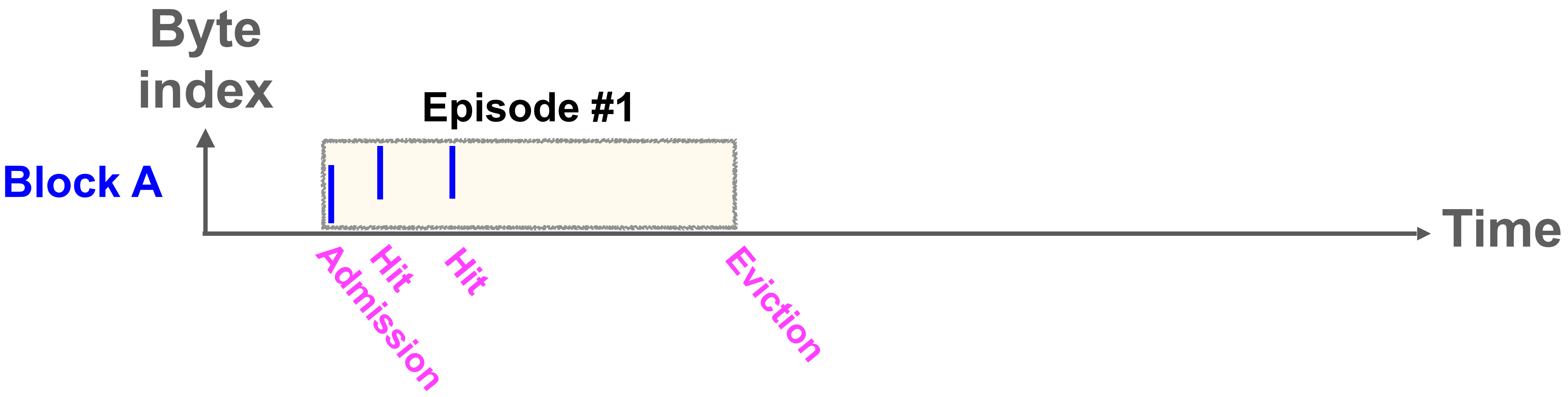
How episodes are generated



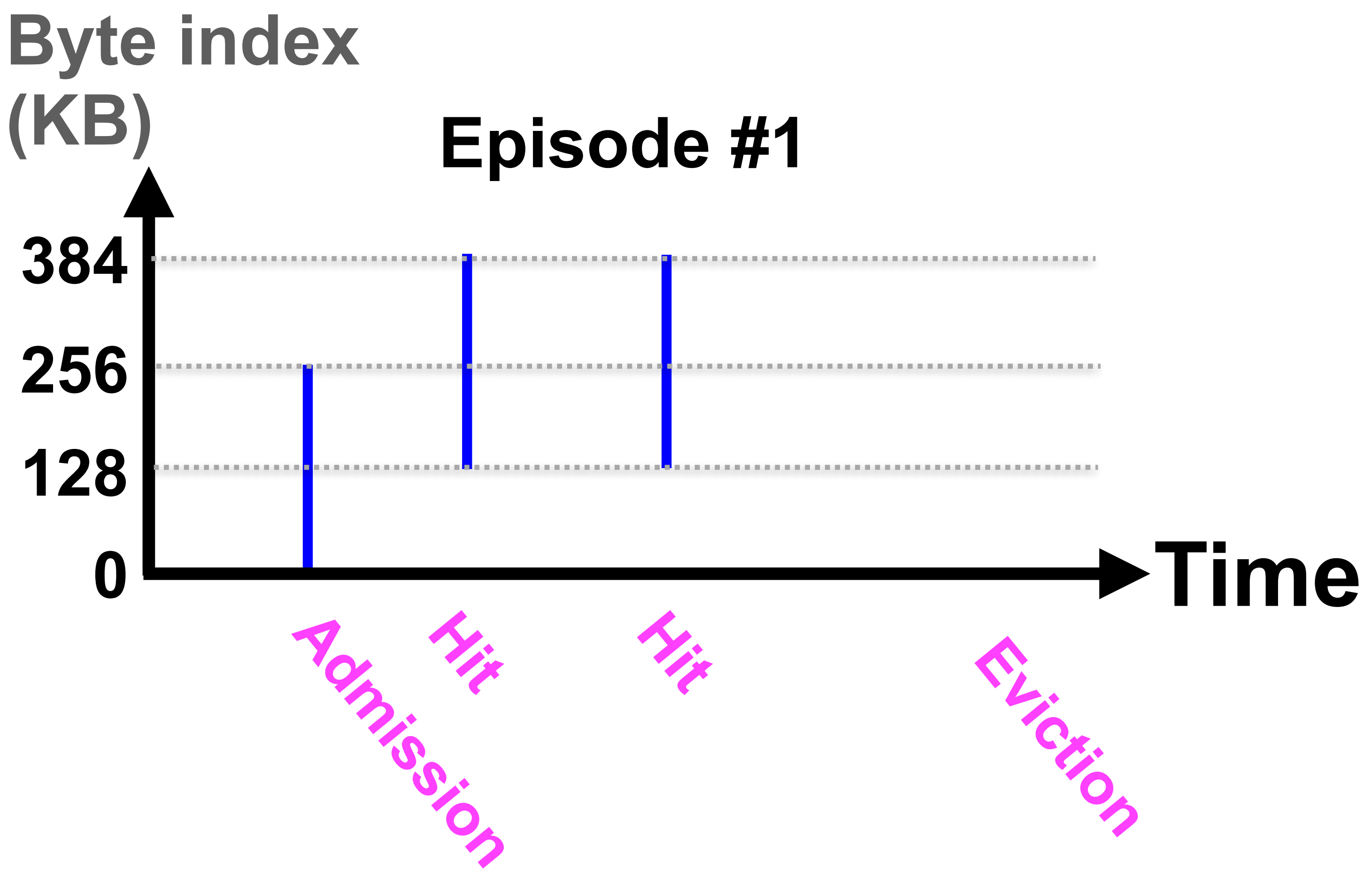
Consider interarrival times of accesses

Split into episodes when interarrival > eviction age

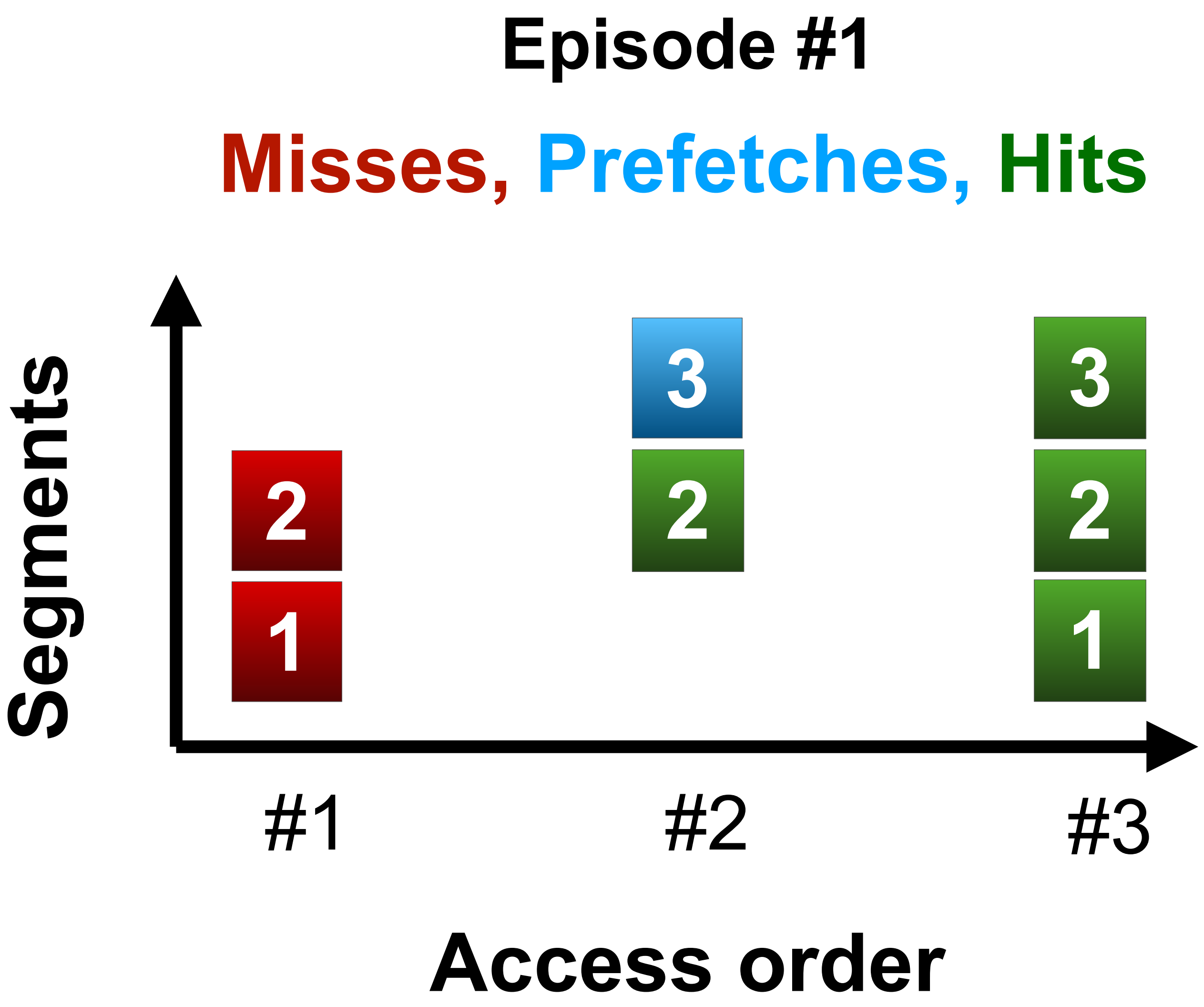
Focusing on Episode 1…



Reason about episodes instead of accesses



Benefits & costs defined on episodes



- **Benefit:** 27ms of DT saved
- **Cost:** 3 flash writes needed

Design

Using episode-based policies to answer "What does good look like?"

Admission: Baleen learns from episode-based OPT

OPT (approx. optimal) admits highest scoring episodes

$$\text{Score}(Ep) = \frac{\text{DTSaved}(Ep)}{\text{FlashWrites}(Ep)} = \frac{27\text{ ms}}{3\text{ flash writes}}$$ ***Episode #1***

OPT emits binary labels based on flash write budget

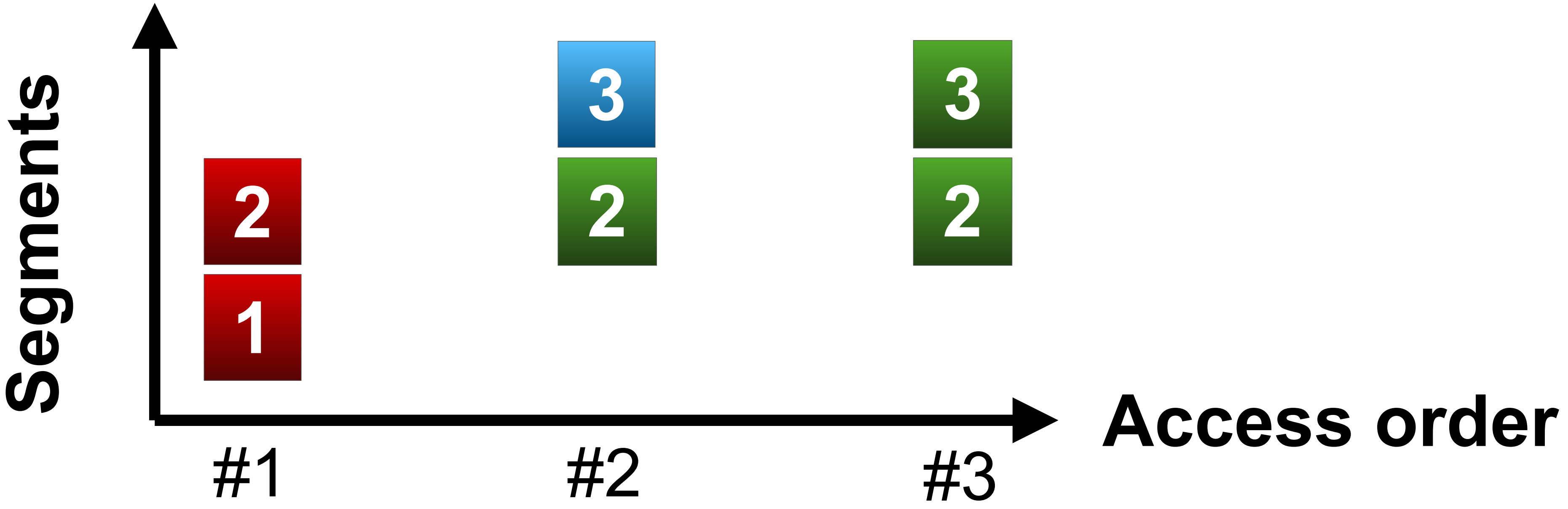
$$\text{Yes if Score}(Ep) > \text{Cutoff}_{\text{TargetFlashWriteRate}}$$

Baleen imitates OPT admission

Baleen's ML-Range learns **what** to prefetch

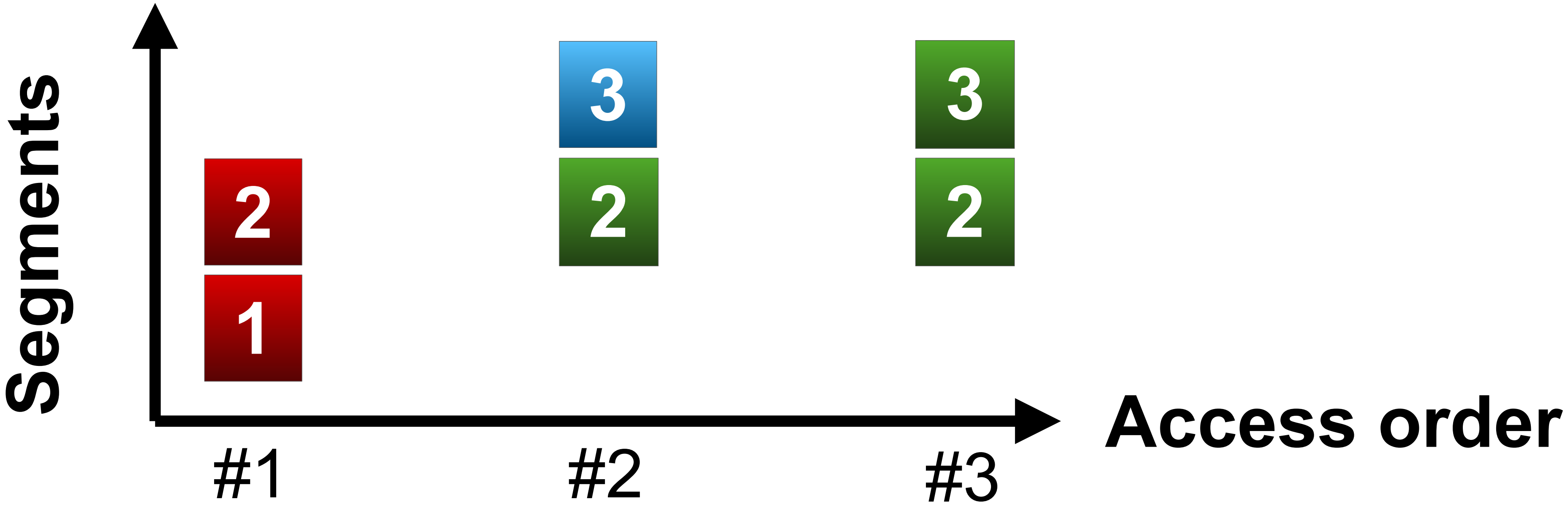
- **What** range to prefetch
  - OPT-Range Start: lowest segment
  - OPT-Range End: highest segment
- **ML-Range** is trained on OPT-Range

$$\text{OPT-Range}(Ep) = [1, 3]$$



Baleen's ML-When learns **when** to prefetch

- **When** to prefetch
  - Bad prefetching hurts: wasted DT & cache space
  - Prefetch only when confident of benefits
  - **ML-When**: Yes if $\mathrm{PrefetchBenefit}(\mathrm{Ep}) > \epsilon$



Baleen-TCO balances HDD savings against SSD cost

- Q: How to balance #HDD against #SSDs?

HDD cost **+** **SSD cost**

Measure *Vary*

$$\mathbf{TCO_1} \propto \frac{\boxed{\mathrm{PeakDT}_{\mathbf{1}}}}{\mathrm{PeakDT}_0} \cdot \#\mathrm{HDDs}_0 + \frac{\mathrm{Cost}_{\mathrm{SSD}}}{\mathrm{Cost}_{\mathrm{HDD}}} \cdot \frac{\boxed{\mathrm{FlashWR}_{\mathbf{1}}}}{\mathrm{FlashWR}_0} \cdot \#\mathrm{SSDs}_0$$

- Baleen-TCO picks optimal flash write rate
  - for each workload

*TCO function based on Google's CacheSack [Yang23]

Evaluation

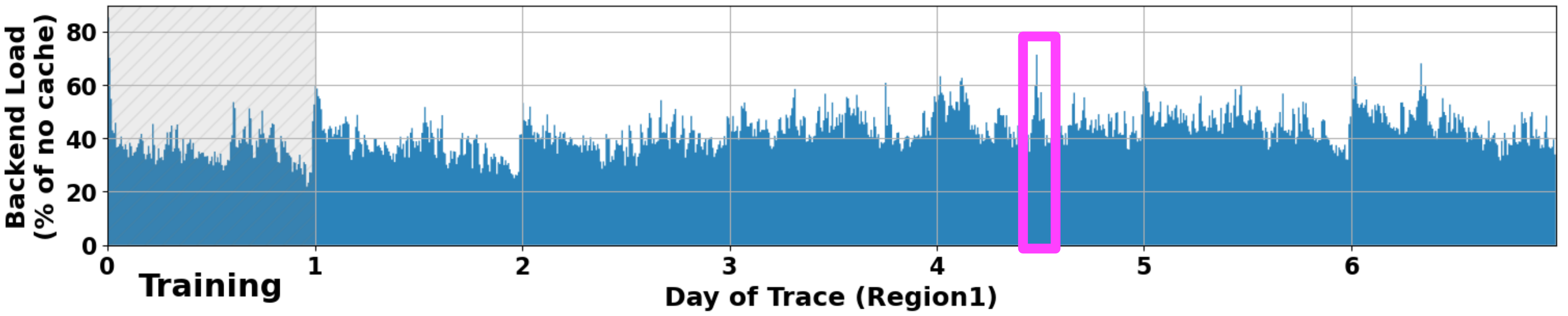
- Production workloads from Meta's Tectonic
  - 7 clusters from 3 years (2019, 2021, 2023)
  - Each serves 1-10 tenants, e.g., data warehouse
  - Each tenant serves 100s of applications
- *More details on Tectonic in Pan et al (FAST 2021)*
- *Traces & simulator code released*

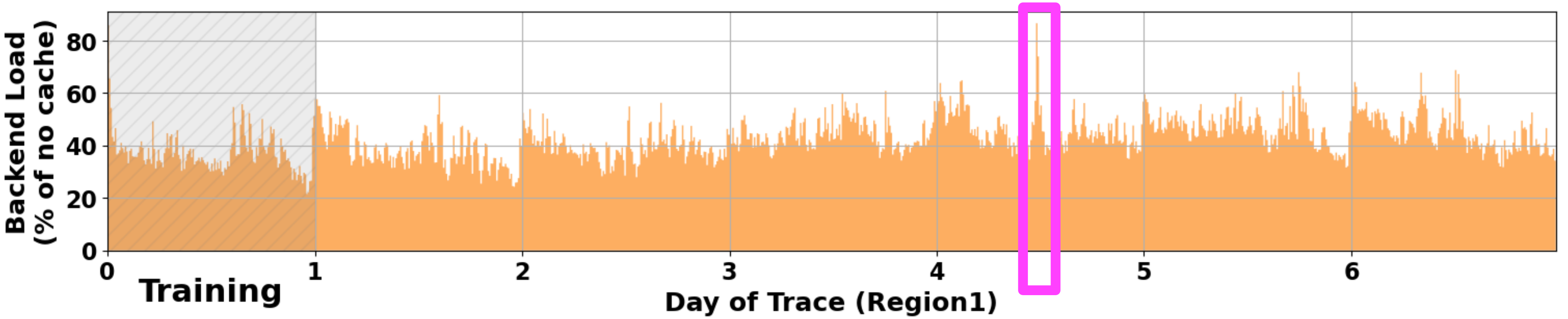
Baseline admission policies

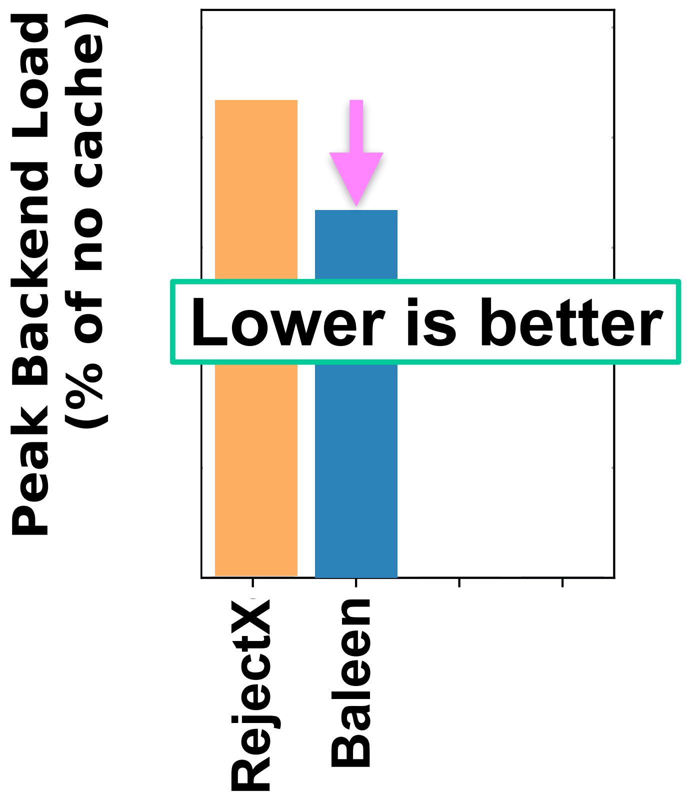
- CoinFlip: flip a coin for each IO
  - Simplest, requires no state
- RejectX (e.g., X=1: accept segment after 1 reject)
  - Used by Meta, Google as baseline
  - 2nd access is always a miss
- CacheLib-ML
  - Used by Meta in production for 3 years
  - Trained on accesses, not episodes

Minimize peak backend load to minimize cost

- We train (offline) on Day 1 and evaluate on Day 2-7
- We compare policies' **Peak DT** (as a % of no caching)



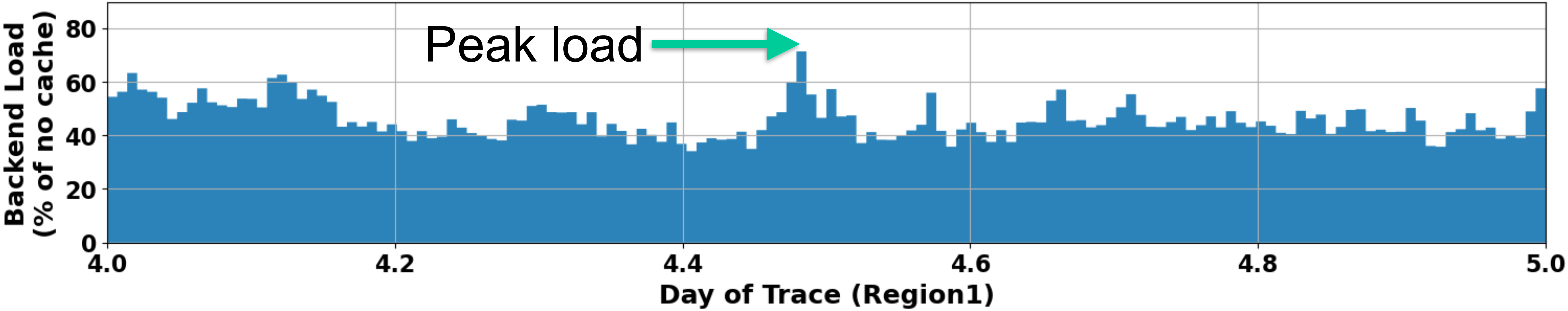




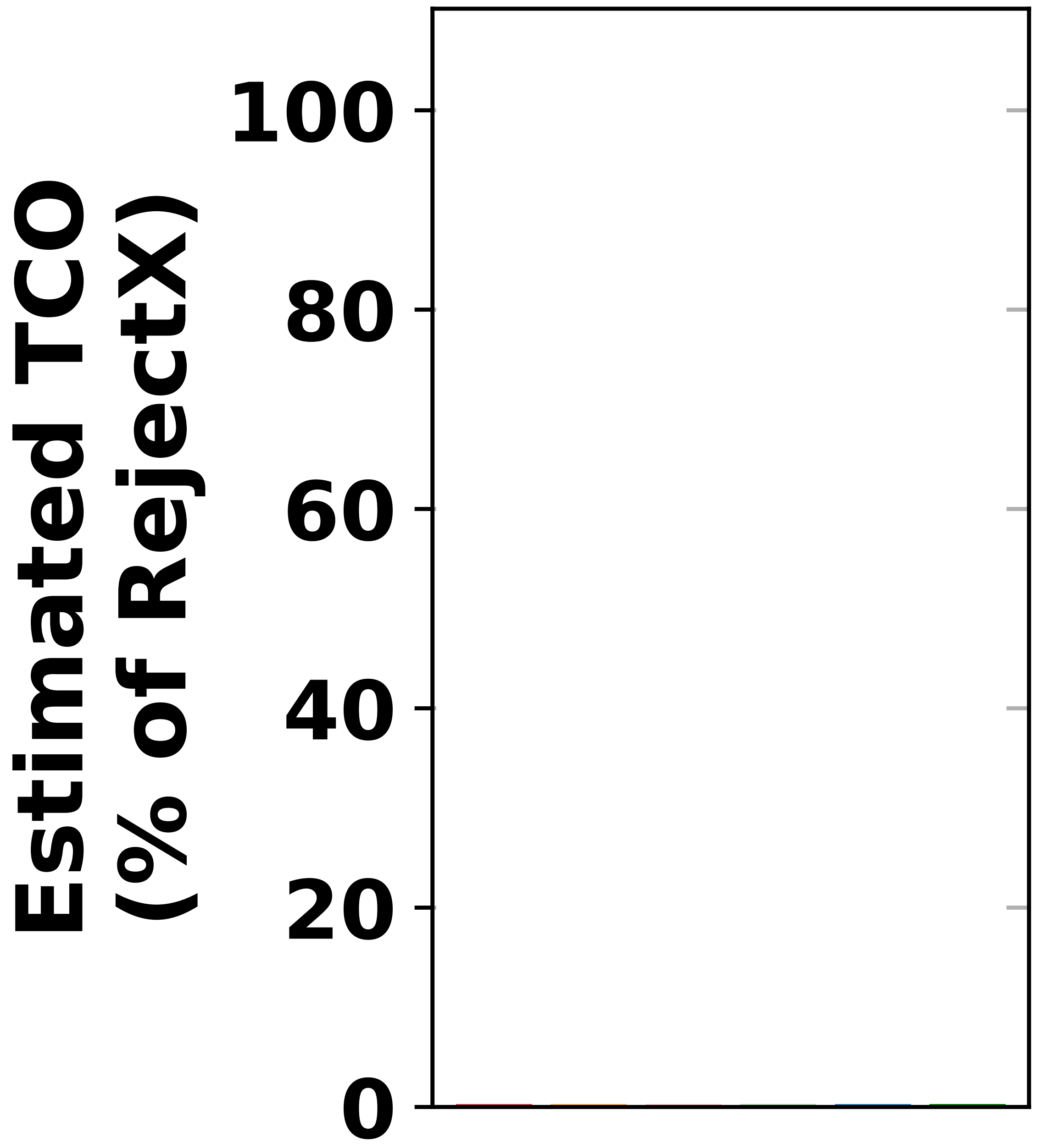
Carnegie Mellon
Parallel Data Laboratory

Reduce peak load to lower total cost

Lower peak load → **Fewer hard disks** & backend servers → **Lower TCO** *Total Cost of Ownership* dominated by media costs

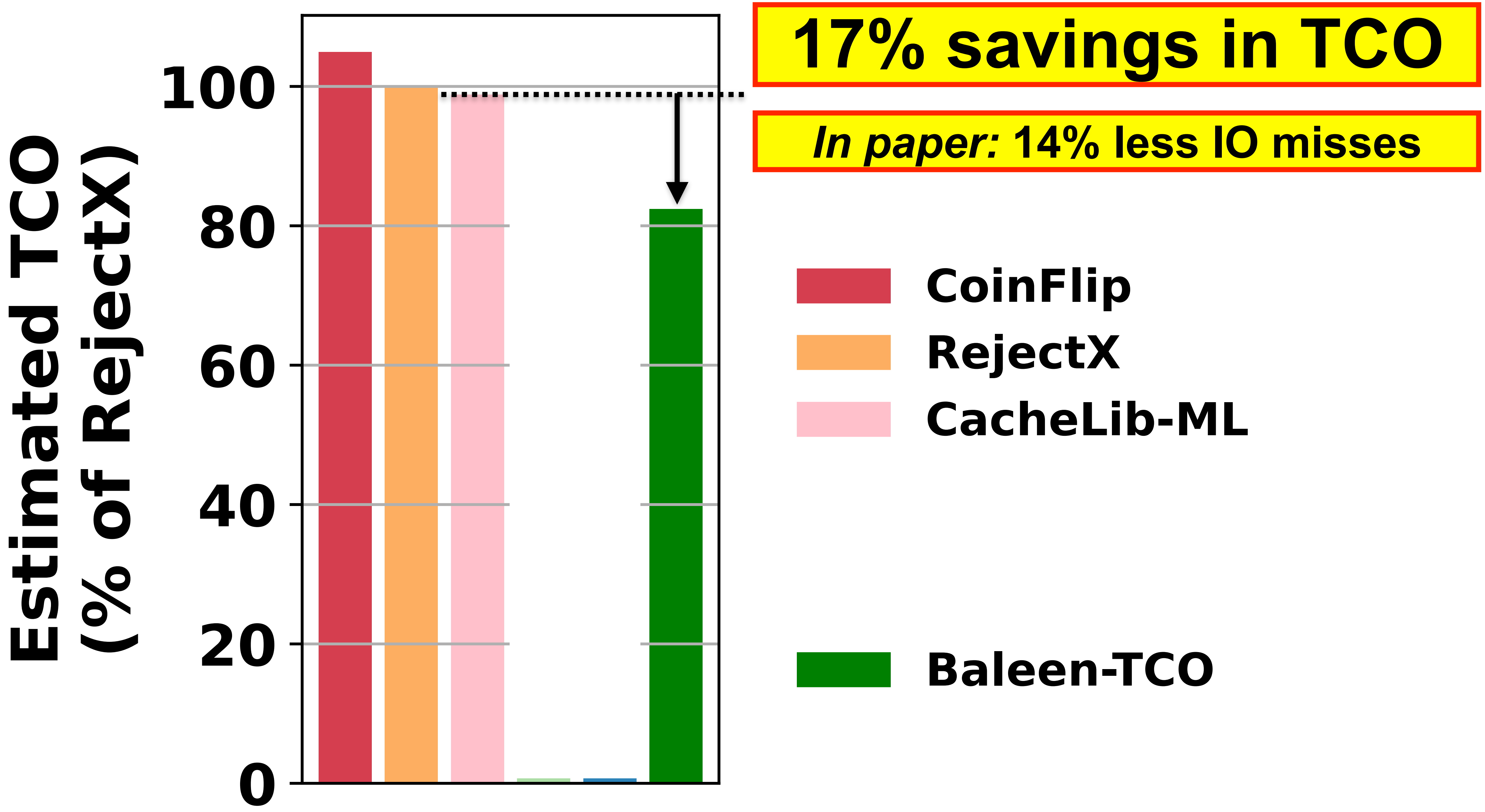


Baleen saves most cost



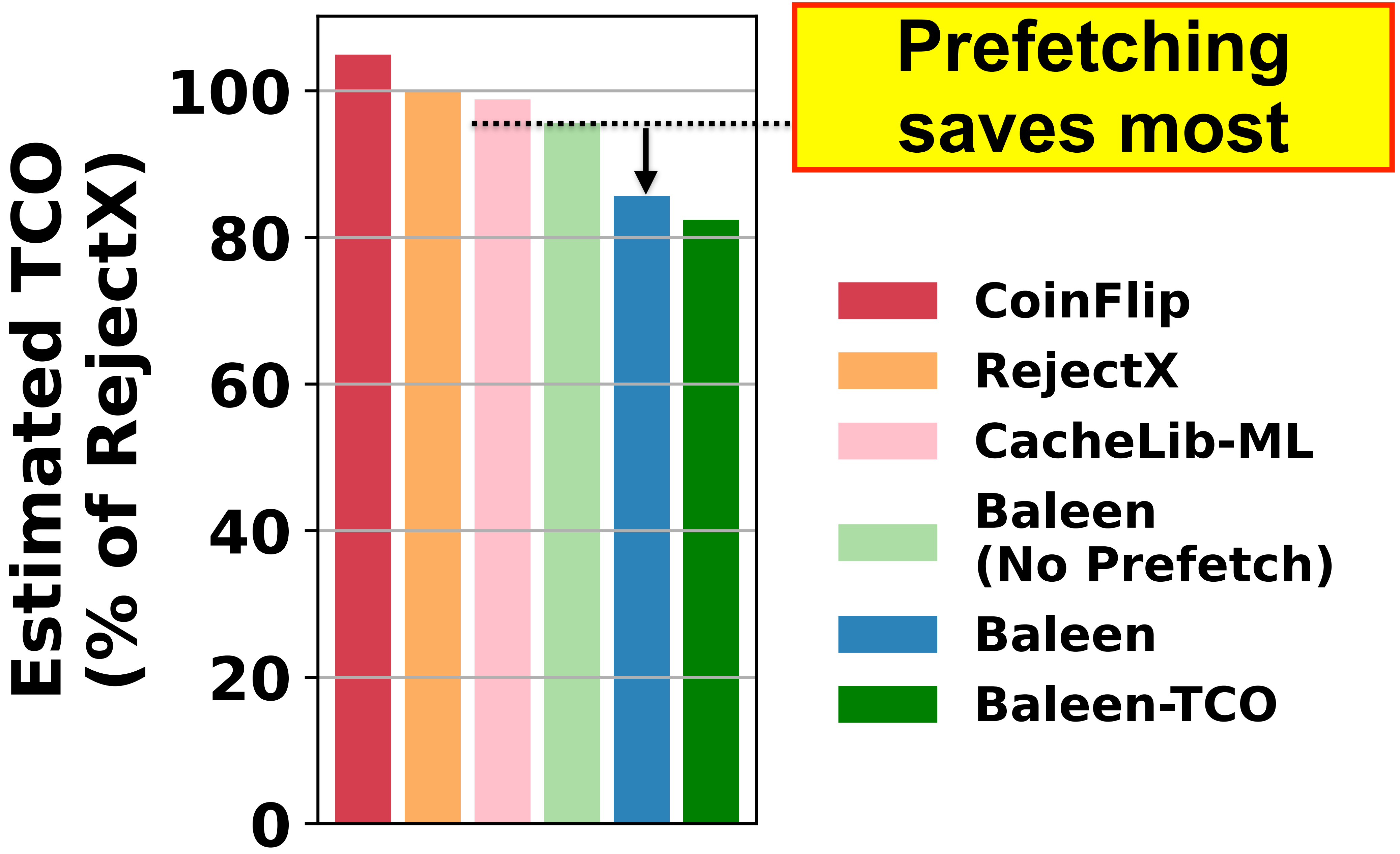
Carnegie Mellon
Parallel Data Laboratory

Baleen saves most cost



Carnegie Mellon
Parallel Data Laboratory

Prefetching accounts for most benefit



Prefetching depends on good admission decisions

- Choice of admission policy matters
  - ML prefetching makes admission baselines worse
- Even with ML admission, 2 models required
  - ML-Range to know what to prefetch
  - ML-When to select when to prefetch

Conclusion

Traces & code
pdl.cmu.edu/CILES

The Boy and the Big Blue Whale
Dr Rose Wadenya, Maria Andrieieva

- Baleen reduces cost by 17%
- Episodes guide ML training
- Optimize for Disk-head Time metric
- Smart admission & prefetching
  - ML-Range predicts what to prefetch
  - ML-When estimates confidence in ML-Range
- Ongoing work: workload drift mitigation
  - **Seeking longer traces with features! (>1 week)**

wonglkd@cmu.edu / www.cs.cmu.edu/~dlwong

Carnegie Mellon
Parallel Data Laboratory

Backup slides

- Benefits of episodes
- What features are used?
- What if we use more complex models?
- What if we vary cache size?
- Architecture
- What workloads?

Model Features

Features	Models	Output
• Namespace • User • Temp. / Perm. • IO start, end • Hourly #accesses (last 6)	Admission	Y/N
	Prefetch-Range	start end
	Prefetch-When	Y/N

Overhead

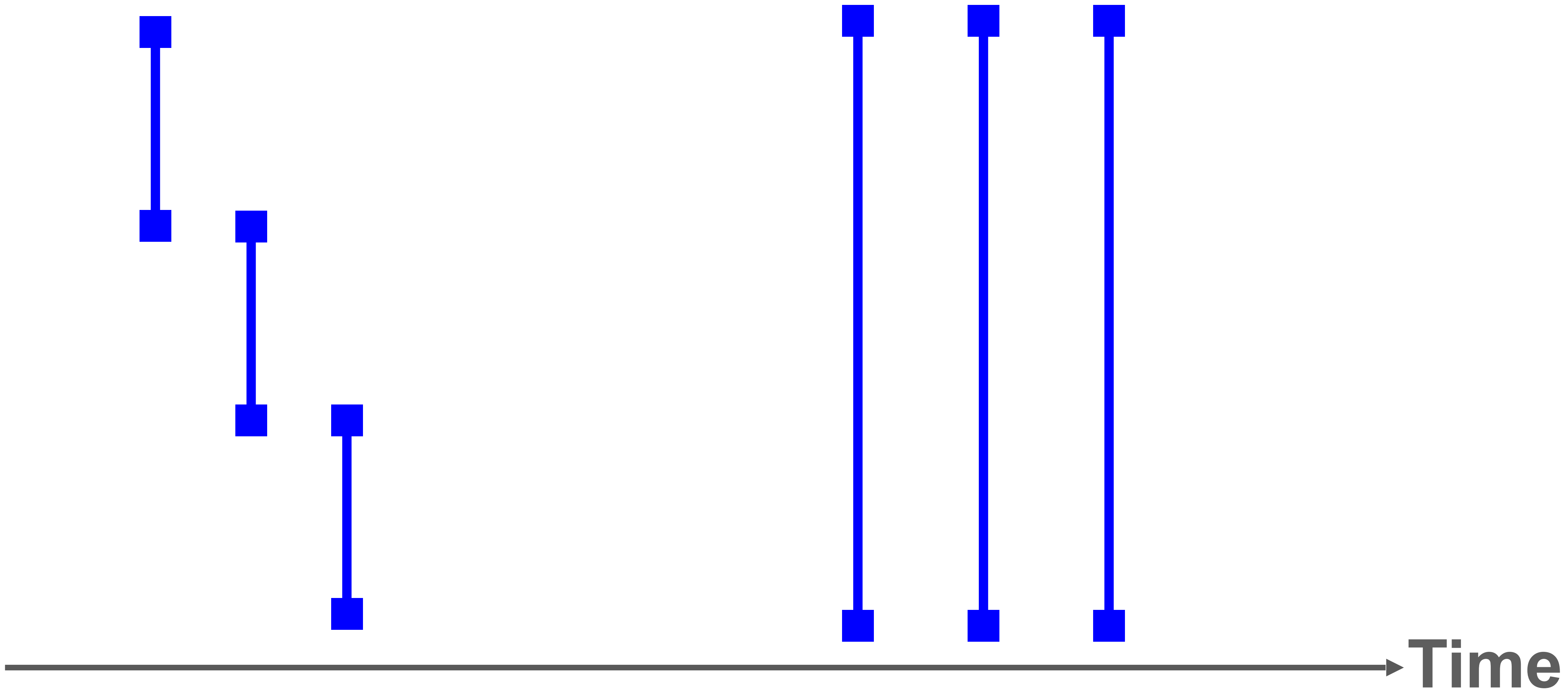
- Limiting factor: latency of a miss going to disk
  - IO: 13 to 56 ms
- Training: 1-5 mins
- Inference latency: ~30μs per inference
  - 4 inferences per access
- Metadata: <1kB per 128kB segment (<1%)

What about write amplification?

- Baleen focuses on larger items (~1MB)
  - Focus on reducing the long-term flash write rate
  - Minimum flash write: 128KB (a segment)
- Kangaroo focuses on small objects
- How you would use this in production
  - CacheLib with a small object cache (Kangaroo) and large object cache (Baleen)

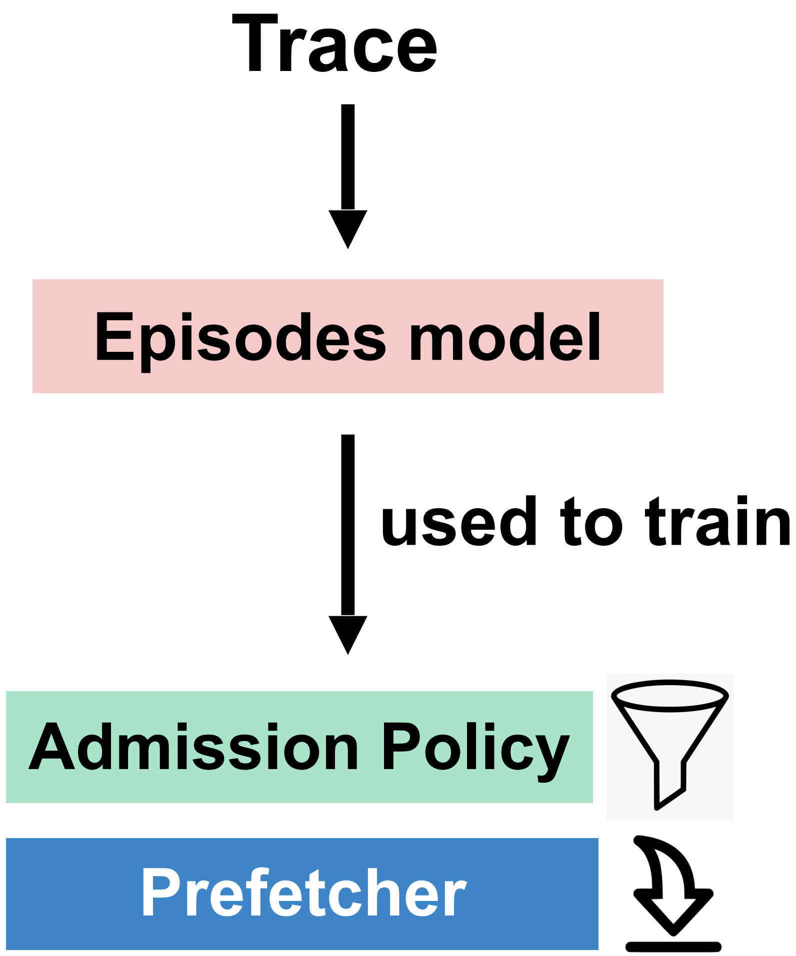
Why DT matters: example

- Same flash writes, same number of IOs saved
- Right saves more DT (and thus disk load)

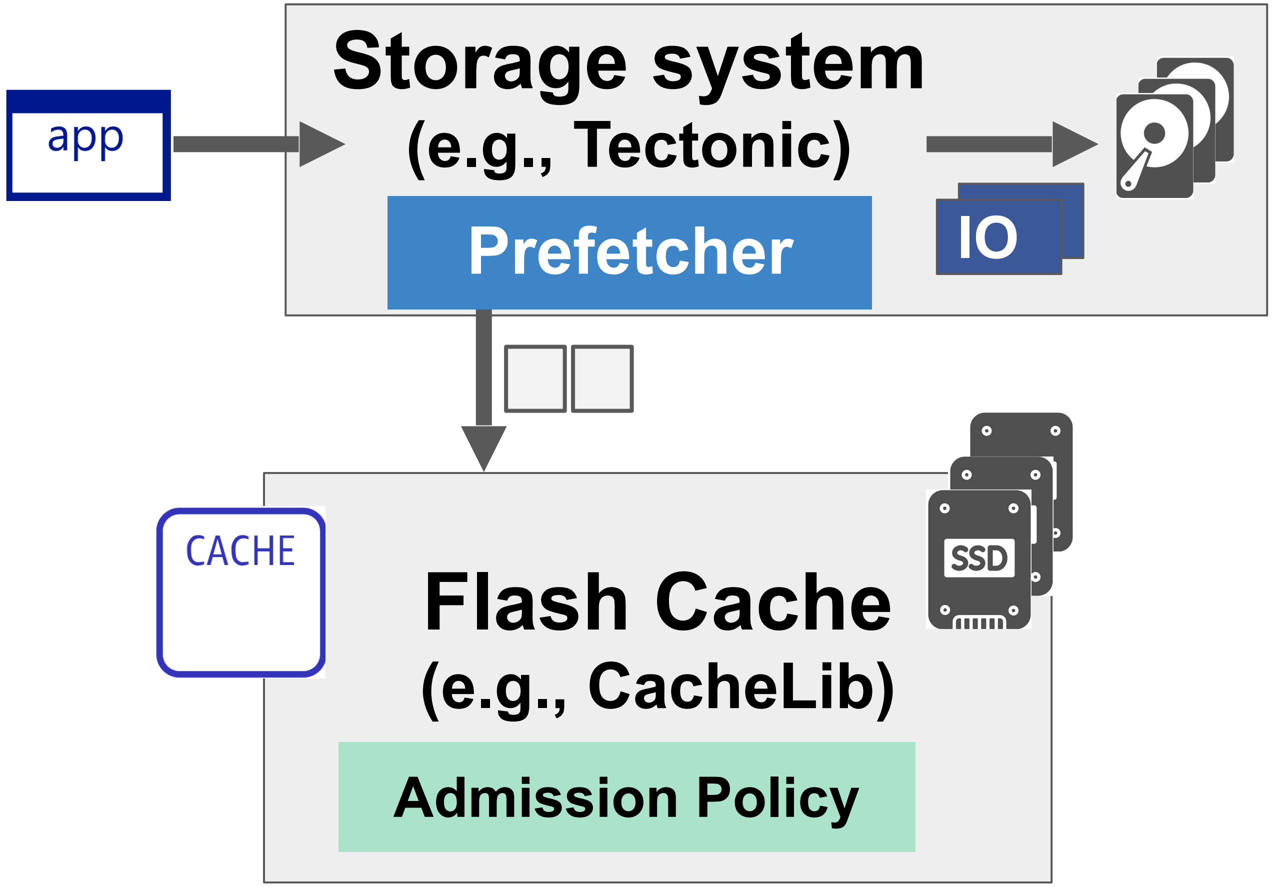


Overall Architecture

Training Time



Deployment Time



Online Baleen

- Keeping track of information needed to score episode
  - Admissions & evictions (to know boundaries)

$$\text{Score}(Ep) = \frac{\text{DTSaved}(Ep)}{\text{FlashWrites}(Ep)}$$

- Determine score cut-off dynamically

$$\text{Score}(Ep) > \text{DynamicCutoff}_{\text{TargetFlashWriteRate}}$$

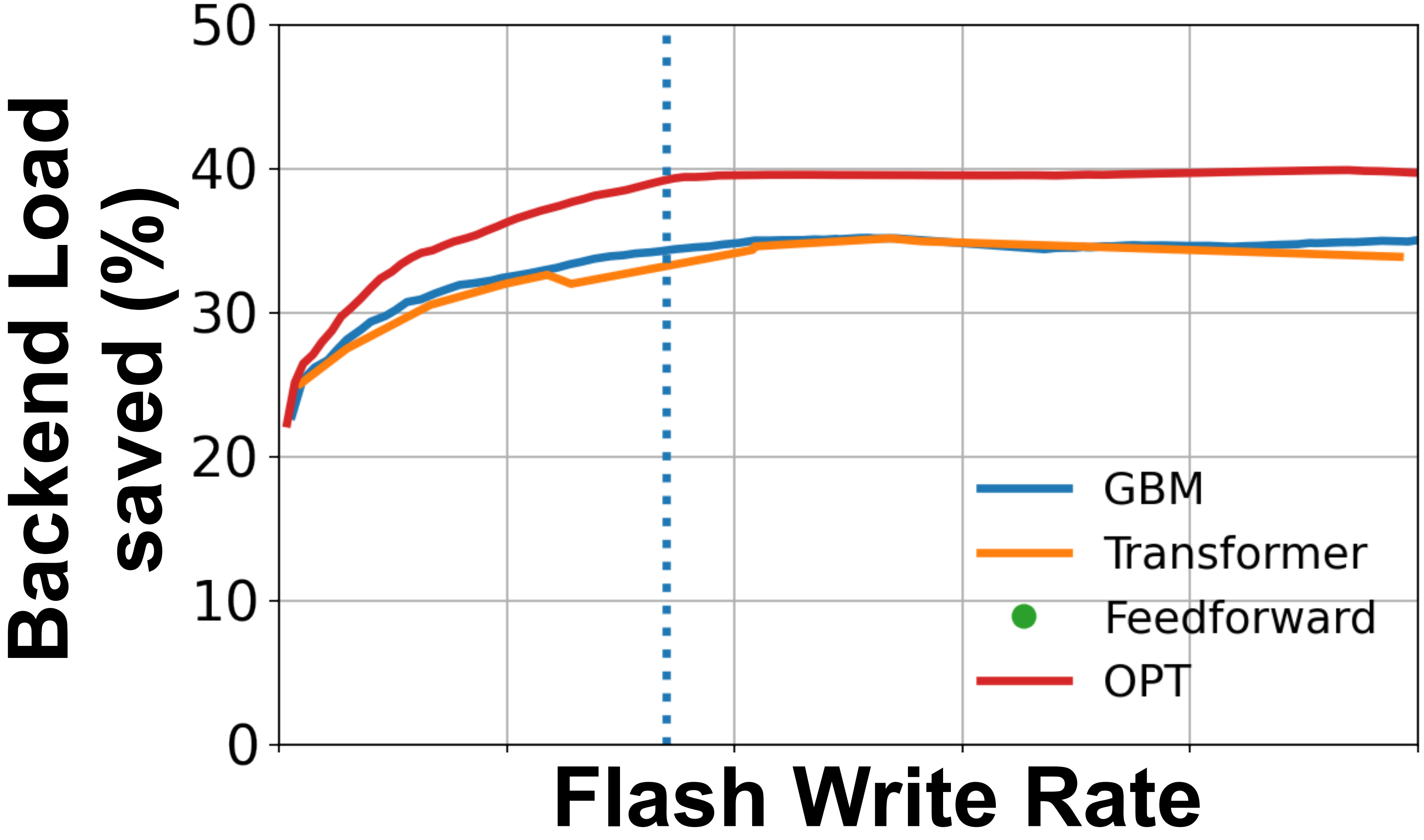
Why we use GBMs

LightGBM

Gradient-Boosting Machines (decision trees)

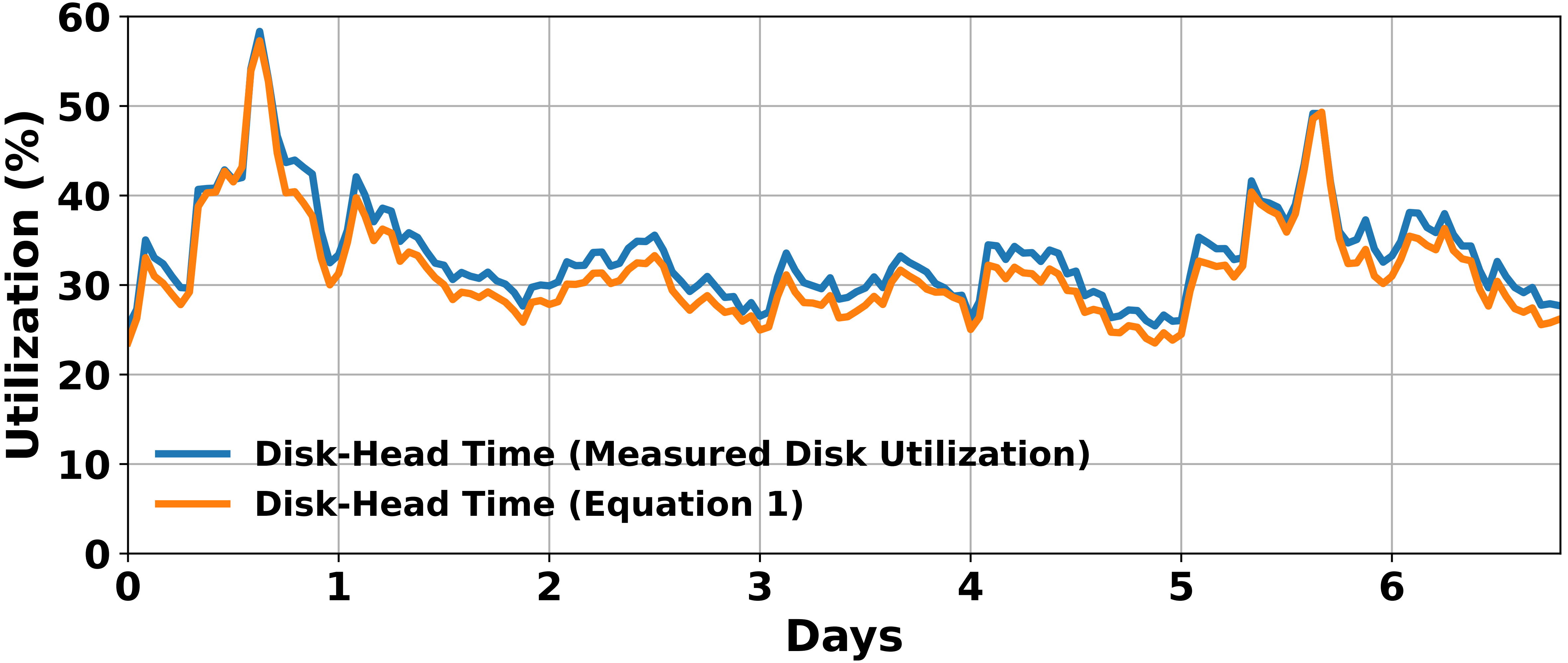
- Accuracy
  - On par with our attempt at a Cache Transformer
- Robustness
- Low inference overhead
  - <1% increase in overall CPU usage

GBM performs as well as Transformer



Calculated DT matches measured DT

- ***DT* = *Seek time x #IOs* + *Read time x #Bytes* (Eq 1)**

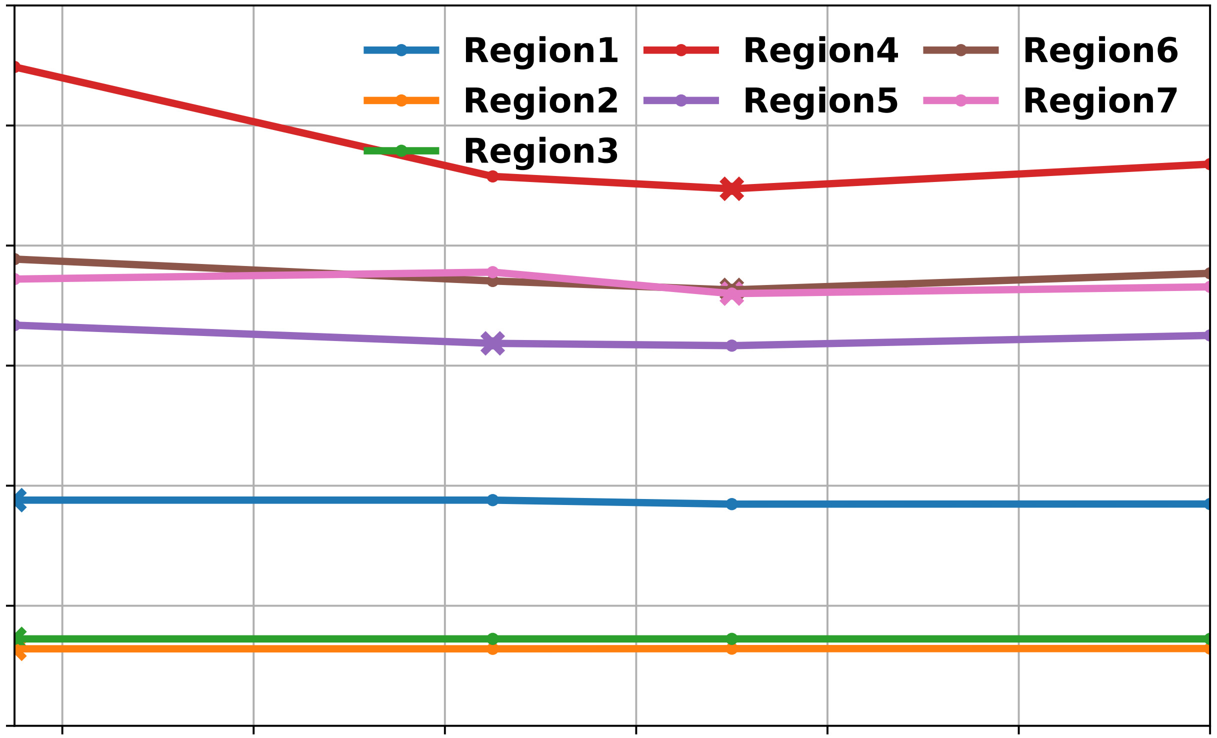


Baleen-TCO

- Picks the optimal flash write rate to minimize 'TCO'

$$\mathbf{TCO_1} \propto \frac{\text{PeakDT}_\mathbf{1}}{\text{PeakDT}_0} \cdot \#\text{HDDs}_0 + \frac{\text{Cost}_\text{SSD}}{\text{Cost}_\text{HDD}} \cdot \frac{\text{FlashWR}_\mathbf{1}}{\text{FlashWR}_0} \cdot \#\text{SSDs}_0$$

Peak Backend Load (PeakDT_1)



Flash Write Rate (FlashWR_1)

Carnegie Mellon
Parallel Data Laboratory