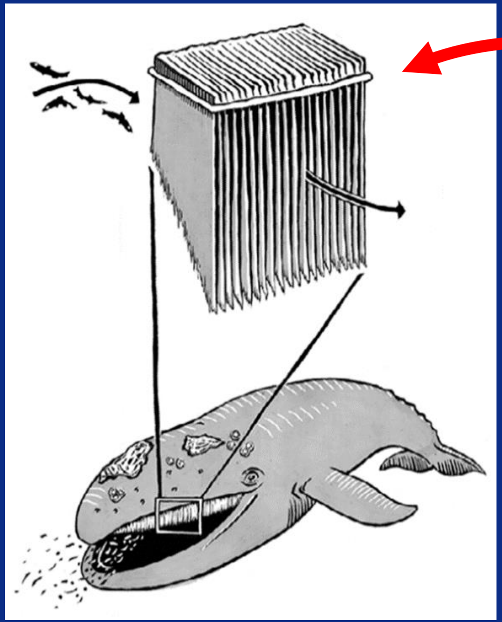


Source: American Museum of Natural History

Baleen: ML Admission & Prefetching for Flash Caches

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

FAST' 24

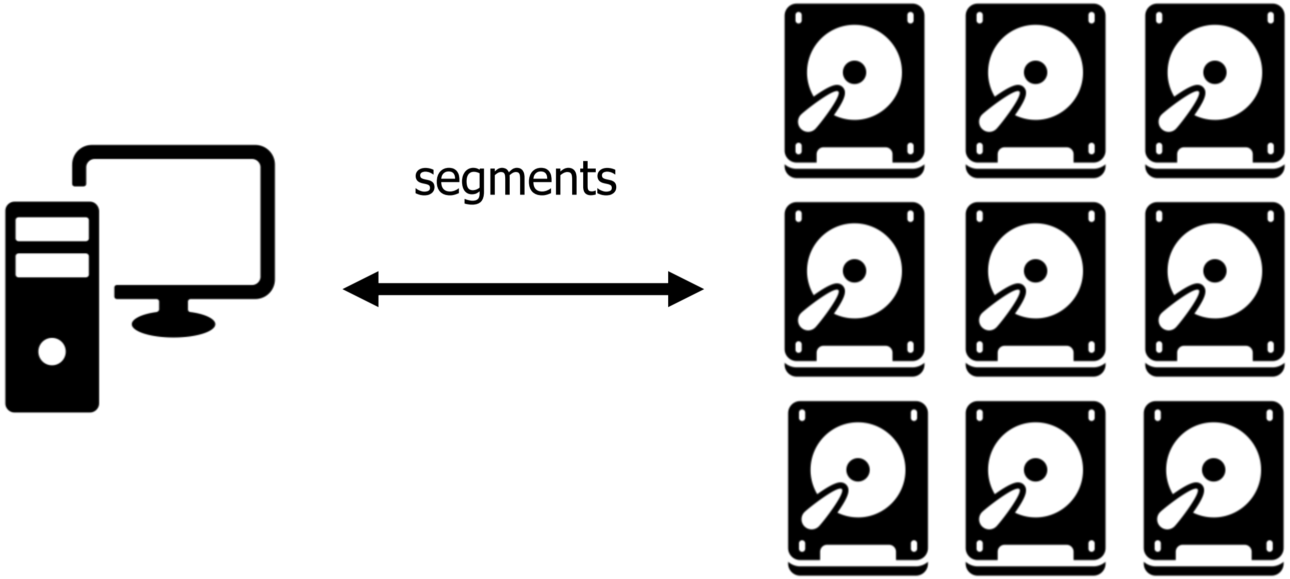
2024.08.28

Presentation by Boseung Kim, Yeojin Oh

bskim1102@dankook.ac.kr, yeojinoh@dankook.ac.kr

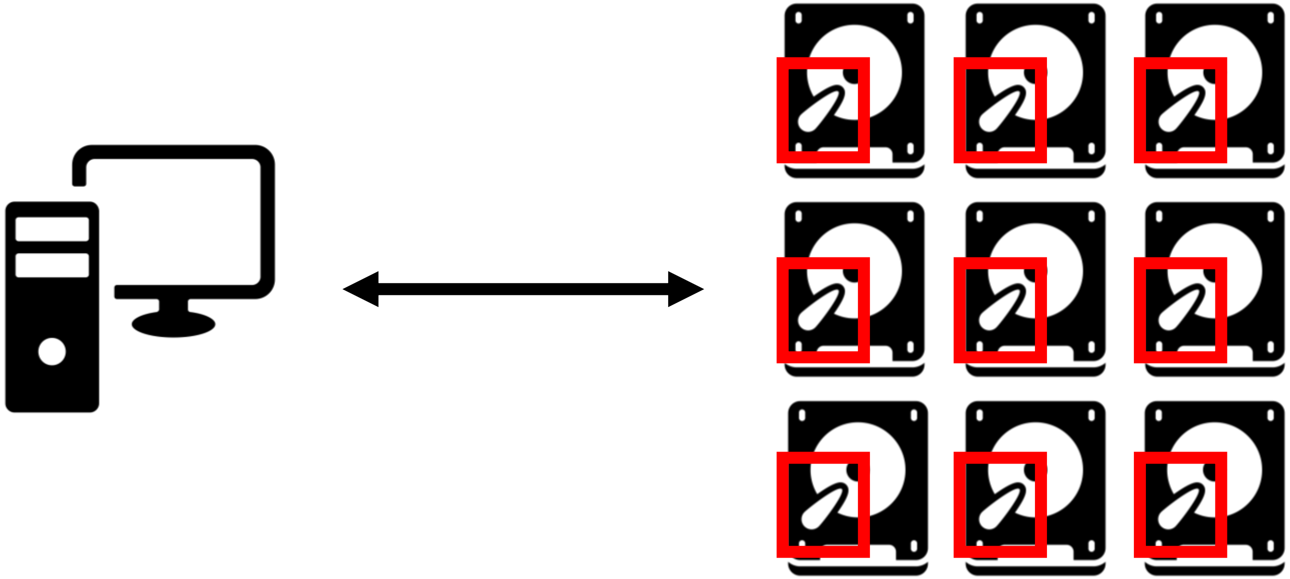
Large-scale storage

- Large-scale storage uses HDD because of its cost-efficiency



Large-scale storage

- Large-scale storage uses HDD because of its cost-efficiency
- However, HDDs have **low bandwidth** and IOPS



Large-scale storage

- Large-scale storage uses HDD because of its cost-efficiency
- However, HDDs have **low bandwidth** and IOPS

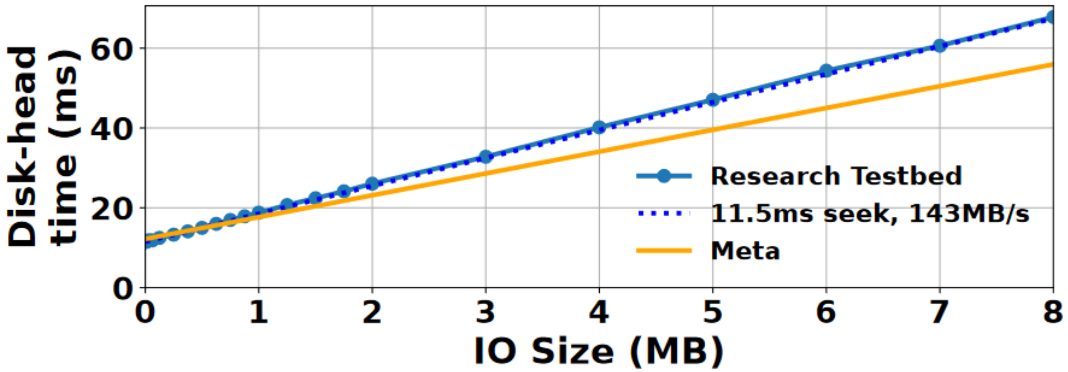
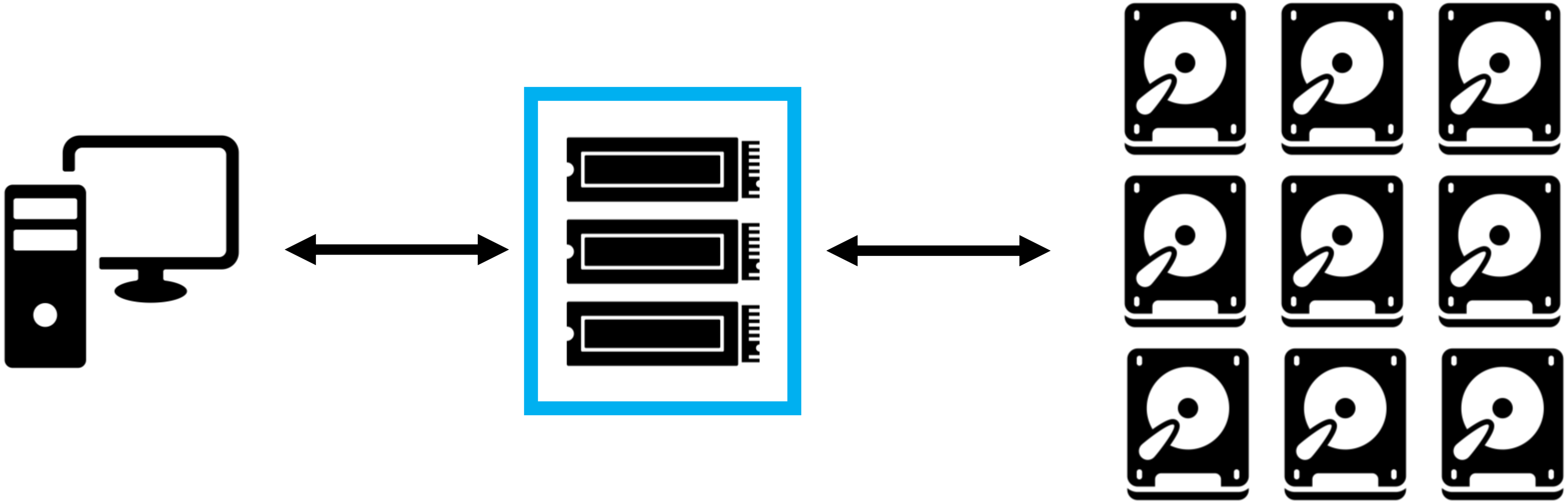


Figure 2: Disk-head Time (DT) for one IO. When a HDD performs an IO, the disk head **seeks** before it **reads** data. For tiny IOs, throughput is limited by *IOPS*; for large IOs, by *bandwidth*. DT encompasses both metrics and generalizes to variable-size IOs.

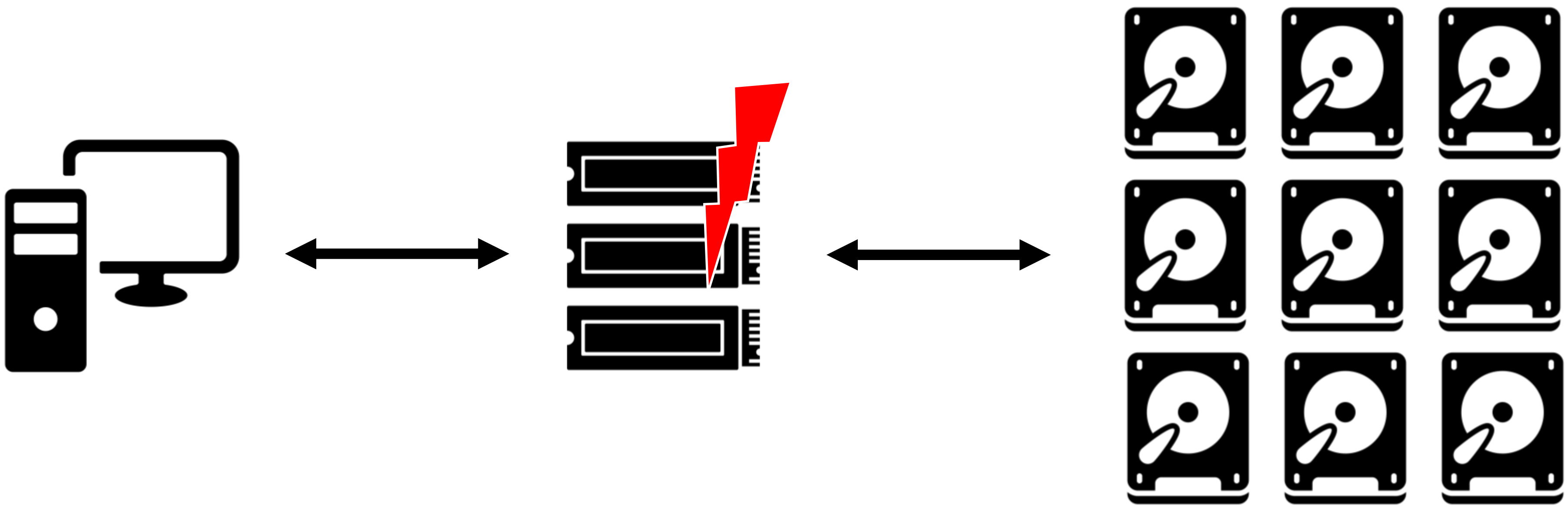
Flash cache

- So, **flash caches** are utilized to absorb a fraction of requests



Flash cache

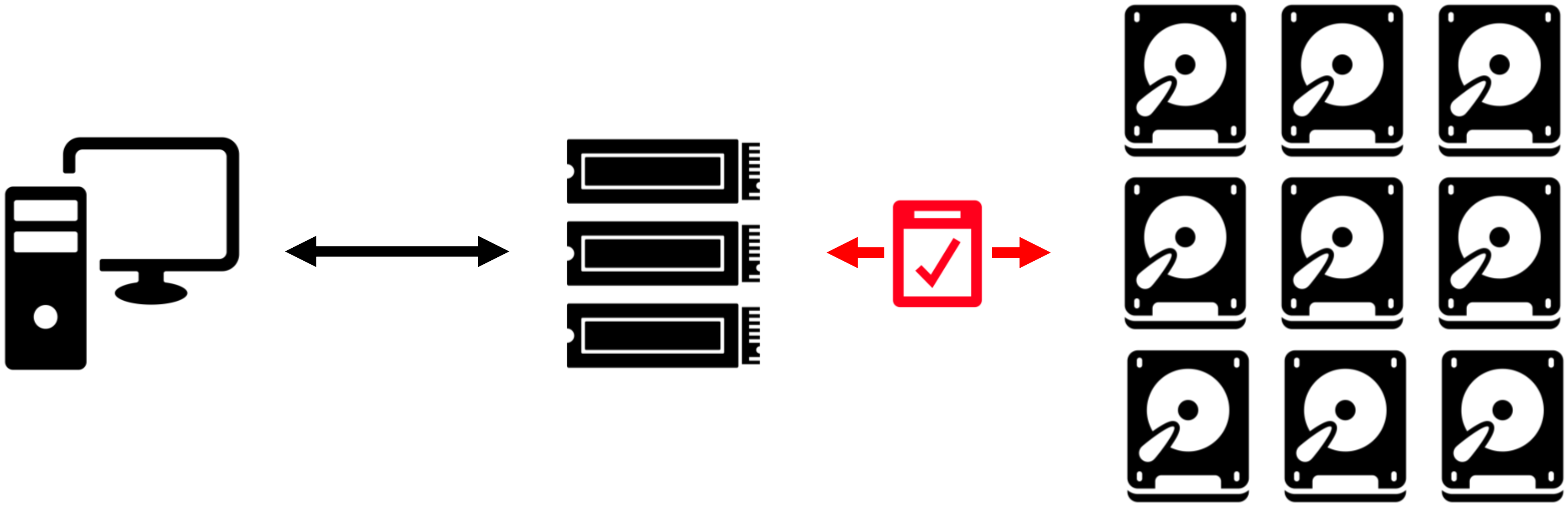
- So, flash caches are utilized to absorb a fraction of requests
- Flash provides higher IOPS, but it **wears out** as it written
  - Specially in caching workload



Flash cache

- These SSD disadvantages require a proper **admission policy**

  ➔ ML policies for flash cache is proposed



Challenges in the flash caching

- Flash admission
  - **Write cost** is determined at caching time
- ML admission
  - **Correct optimization metric** not obvious → Disk-head Time (DT)
  - train on all accesses in a trace, but focus on those that **result in misses**→ Episode
- Prefetching policy
  - **Misprefetching** is costly as it consumes writes and extra DT

Challenges in the flash caching

- Flash admission
  - Write cost is determined at caching time
- ML admission
  - Correct optimization metric not obvious → **Disk-head Time (DT)**
  - Train on all accesses in a trace, but focus on those that result in misses→ Episode
- Prefetching policy
  - Misprefetching is costly as it consumes writes and extra DT

Disk-head time(DT)

- End-to-end throughput metric to evaluate admission
- Cost of serving requests to the backend
  - Seek time + Read time, $DT_i = t_{seek} + n \cdot t_{read}$
  - Backend load: total DT needed to serve misses

$$Util_{DT} = \frac{\sum_i DT_i}{DT_{Provisioned}}, \quad \sum_i DT_i = Fetches_{IOs} \cdot t_{seek} + Fetches_{Bytes} \cdot t_{read} \tag{1}$$

Disk-head time(DT)

- End-to-end throughput metric to evaluate admission
- Cost of serving requests to the backend
  - Seek time + Read time, $DT_i = t_{seek} + n \cdot t_{read}$
  - Backend load: total DT needed to serve misses

$$Util_{DT} = \frac{\sum_i DT_i}{DT_{Provisioned}}, \quad \sum_i DT_i = Fetches_{IOs} \cdot t_{seek} + Fetches_{Bytes} \cdot t_{read} \qquad (1)$$

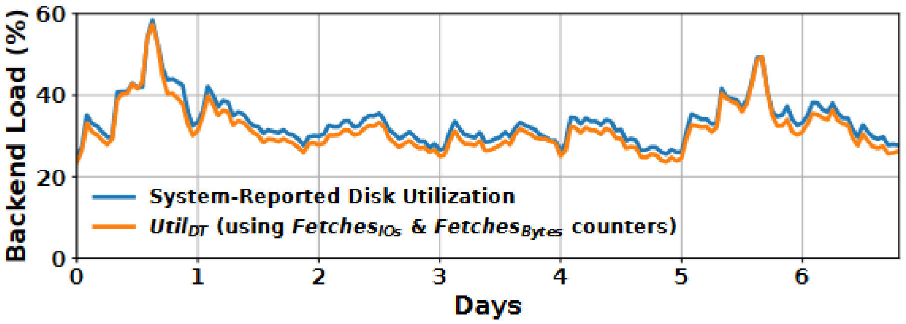


Figure 3: DT validated in production. Our DT formula (plugging counters into Eq 1) matches measured disk utilization (blue) closely. The peak of 58% occurs on Day 0.

Use DT to address sub-problems

- Admission
  - Compare saved disk-head time and write cost when caching items
- Prefetching
  - Trade off DT saved from hits and wasted from incorrect prefetches
- Eviction
  - Using simple LRU
  - Leaving ML-based eviction policies for future work…

Challenges in the flash caching

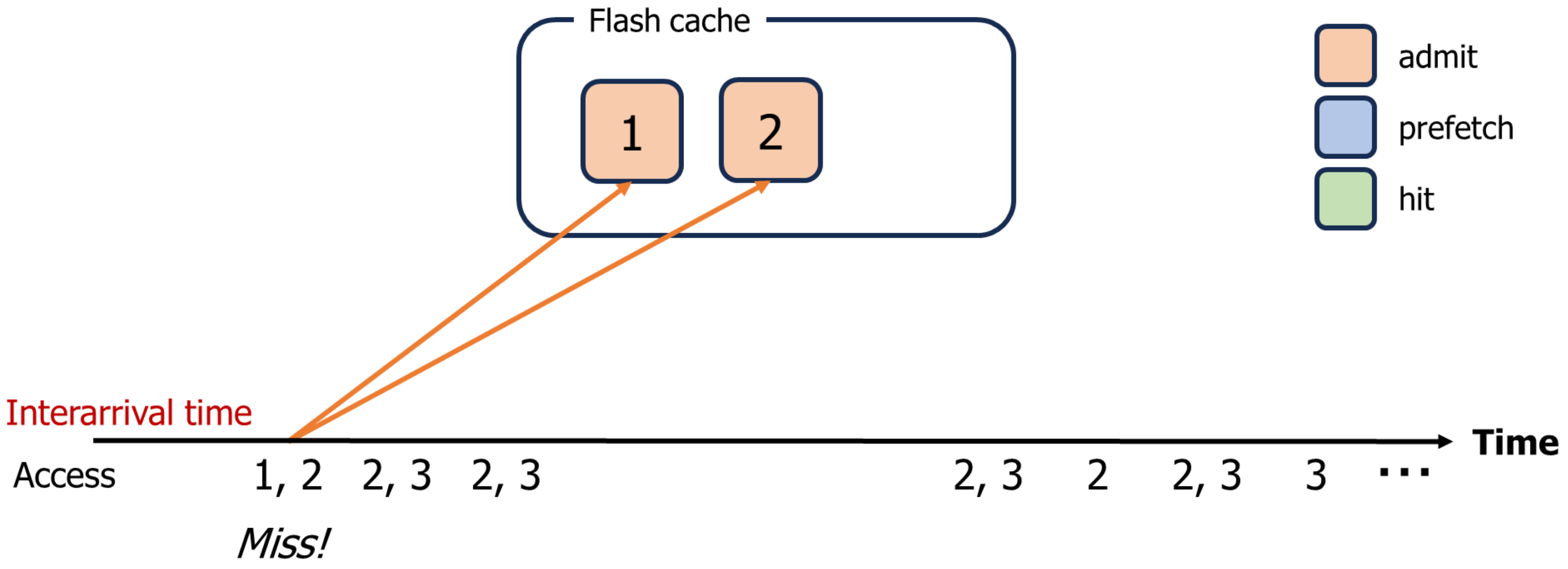
- Flash admission
  - Write cost is determined at caching time
- ML admission
  - Correct optimization metric not obvious → Disk-head Time (DT)
  - Train on all accesses in a trace, but focus on those that result in misses→ **Episode**
- Prefetching policy
  - Misprefetching is costly as it consumes writes and extra DT

Episode

- Definition: a sequence of accesses that would be hits if the corresponding item was admitted
  - The entire process of an item entering and leaving the cache (=life span)
  - It allows us to consider how often and when an item is used in the cache overall

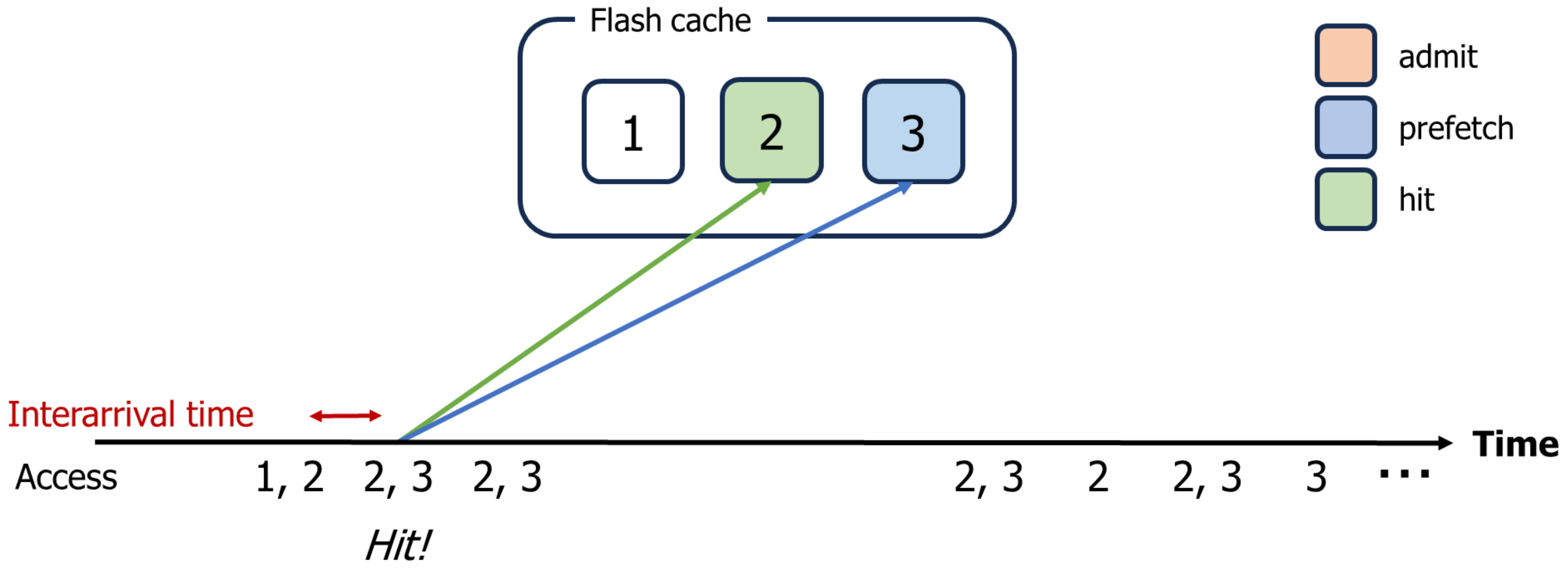
Episode

- Definition: a sequence of accesses that would be hits if the corresponding item was admitted
  - The entire process of an item entering and leaving the cache (=life span)
  - It allows us to consider how often and when an item is used in the cache overall



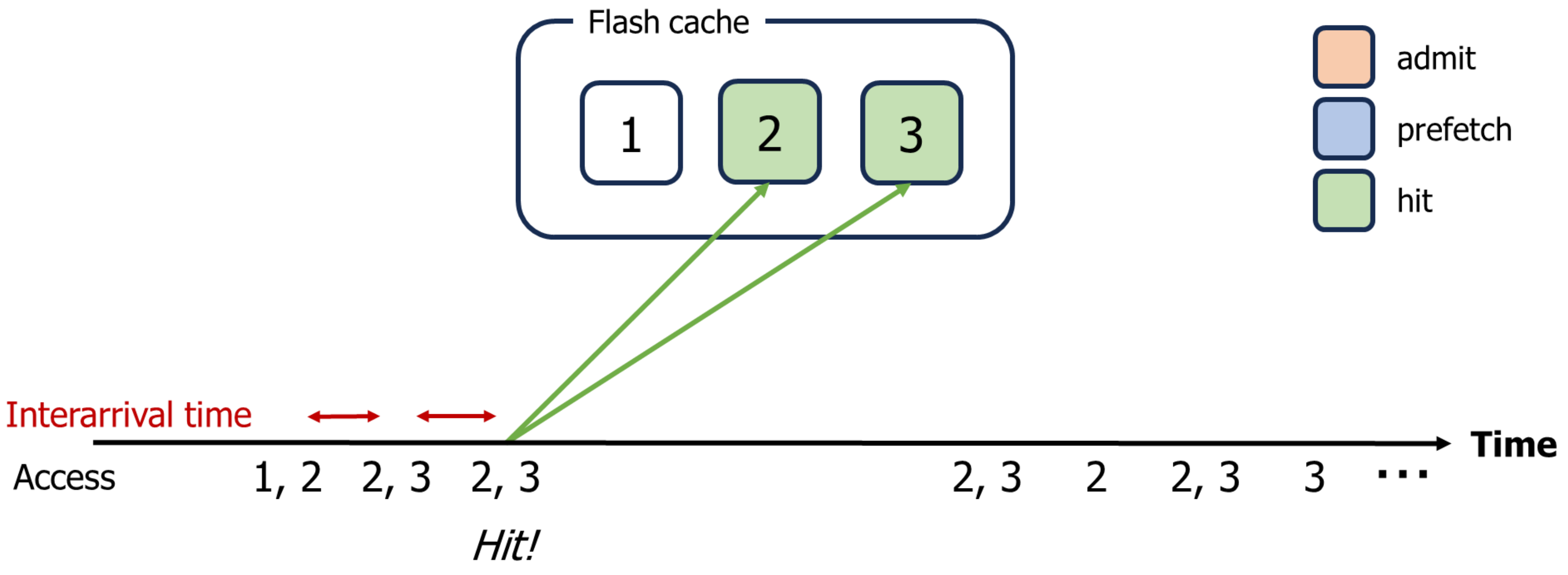
Episode

- Definition: a sequence of accesses that would be hits if the corresponding item was admitted
  - The entire process of an item entering and leaving the cache (=life span)
  - It allows us to consider how often and when an item is used in the cache overall



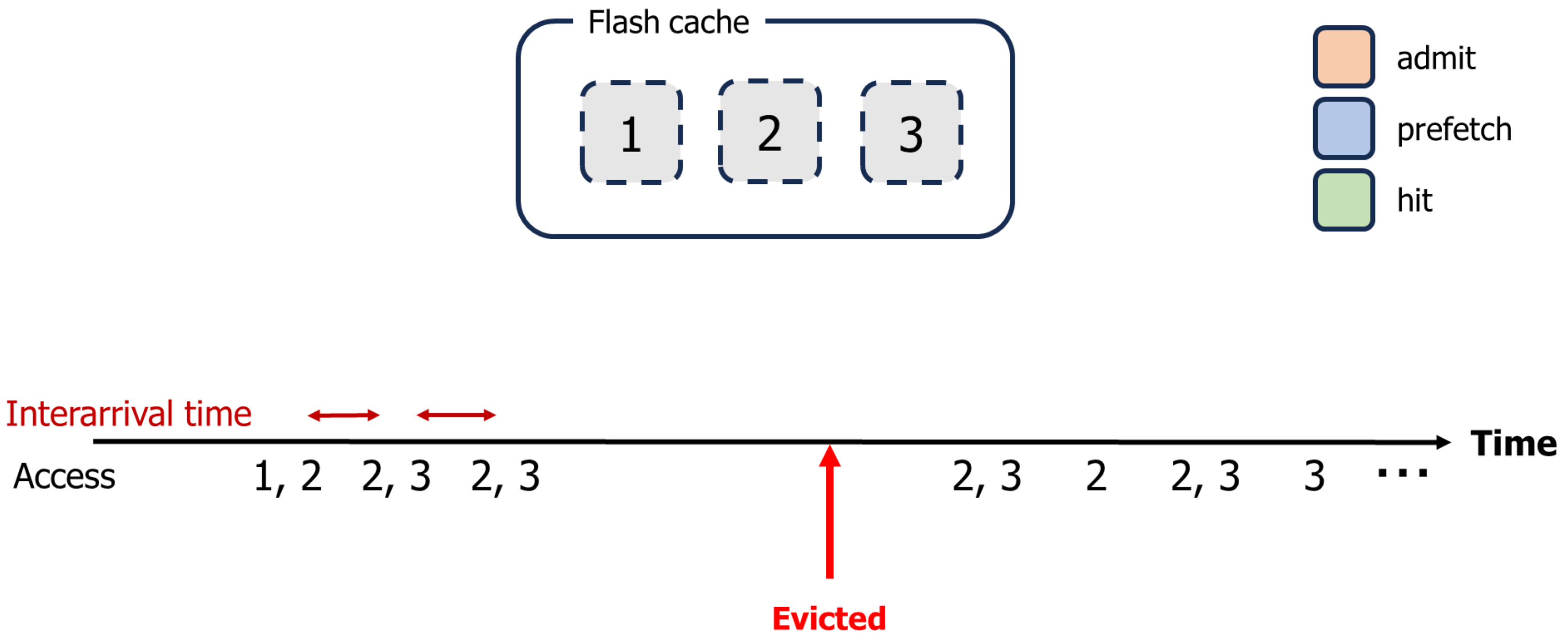
Episode

- Definition: a sequence of accesses that would be hits if the corresponding item was admitted
  - The entire process of an item entering and leaving the cache (=life span)
  - It allows us to consider how often and when an item is used in the cache overall



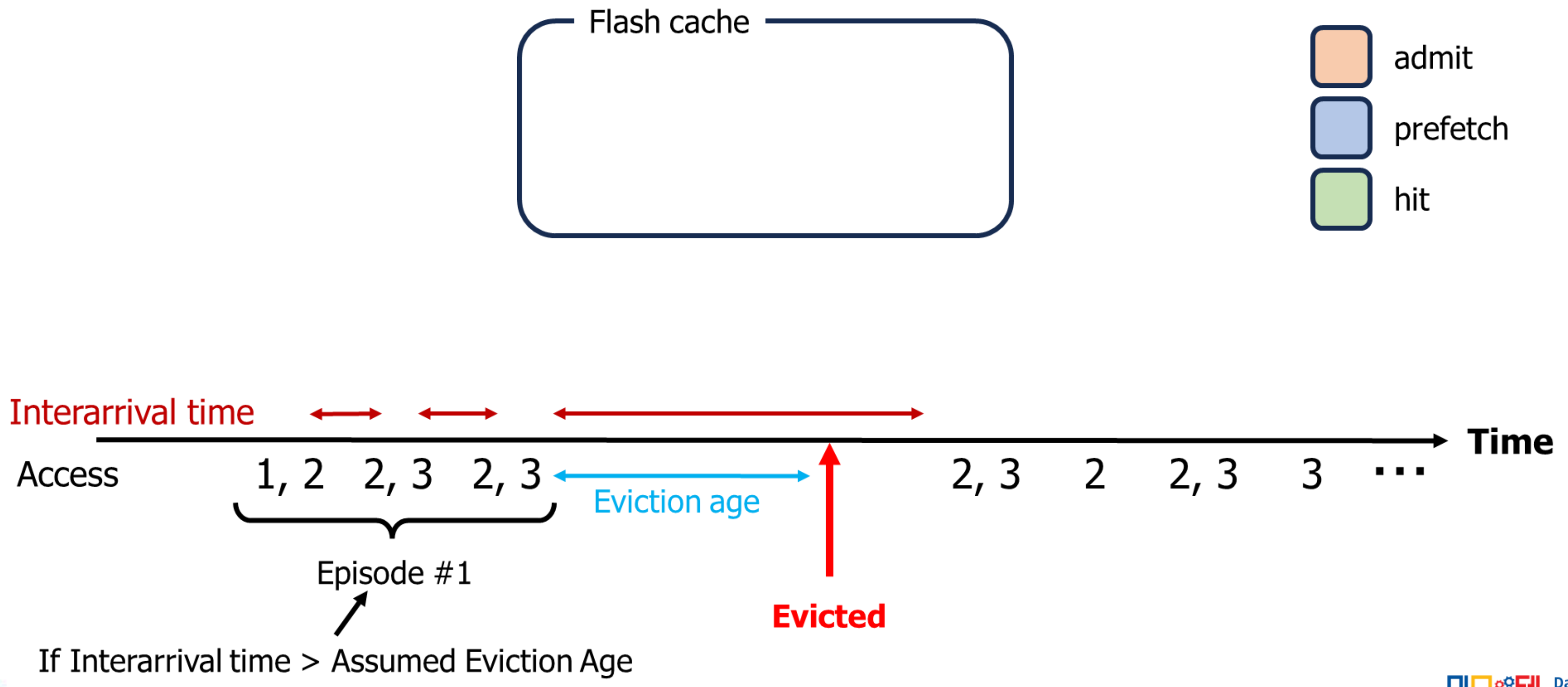
Episode

- Definition: a sequence of accesses that would be hits if the corresponding item was admitted
  - The entire process of an item entering and leaving the cache (=life span)
  - It allows us to consider how often and when an item is used in the cache overall



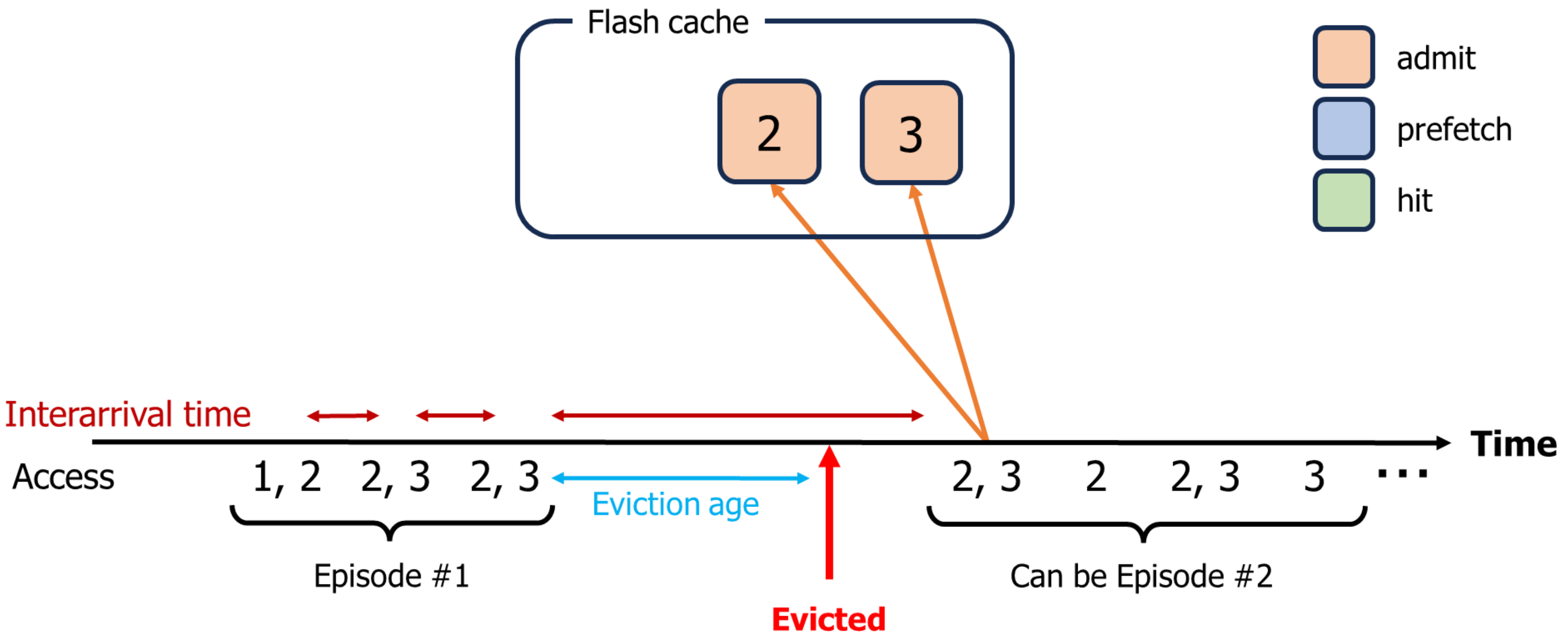
Episode

- Definition: a sequence of accesses that would be hits if the corresponding item was admitted
  - The entire process of an item entering and leaving the cache (=life span)
  - It allows us to consider how often and when an item is used in the cache overall



Episode

- Definition: a sequence of accesses that would be hits if the corresponding item was admitted
  - The entire process of an item entering and leaving the cache (=life span)
  - It allows us to consider how often and when an item is used in the cache overall

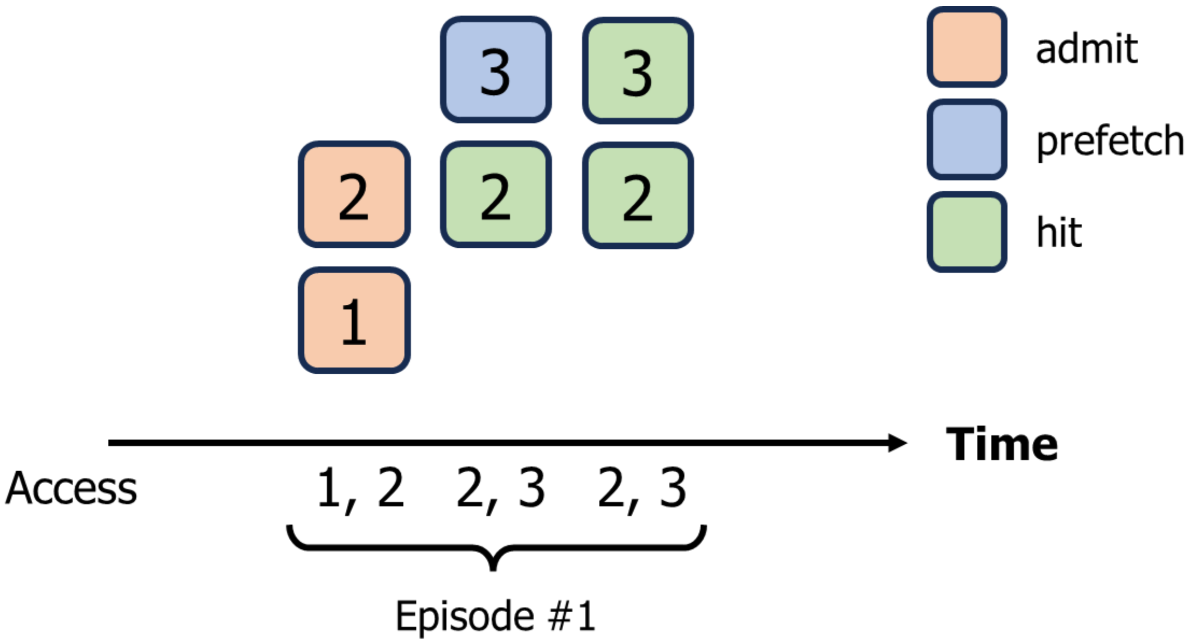


Episode’s cost-benefit

- Episode’s **starting point**
  - The earlier it starts, the more the write cost can be amortized
- Episode’s **length**
  - The longer the episode, the higher the probability of a cache hit each time the item is used
- Episode’s **access pattern**
  - The more accesses there are, the greater the benefit

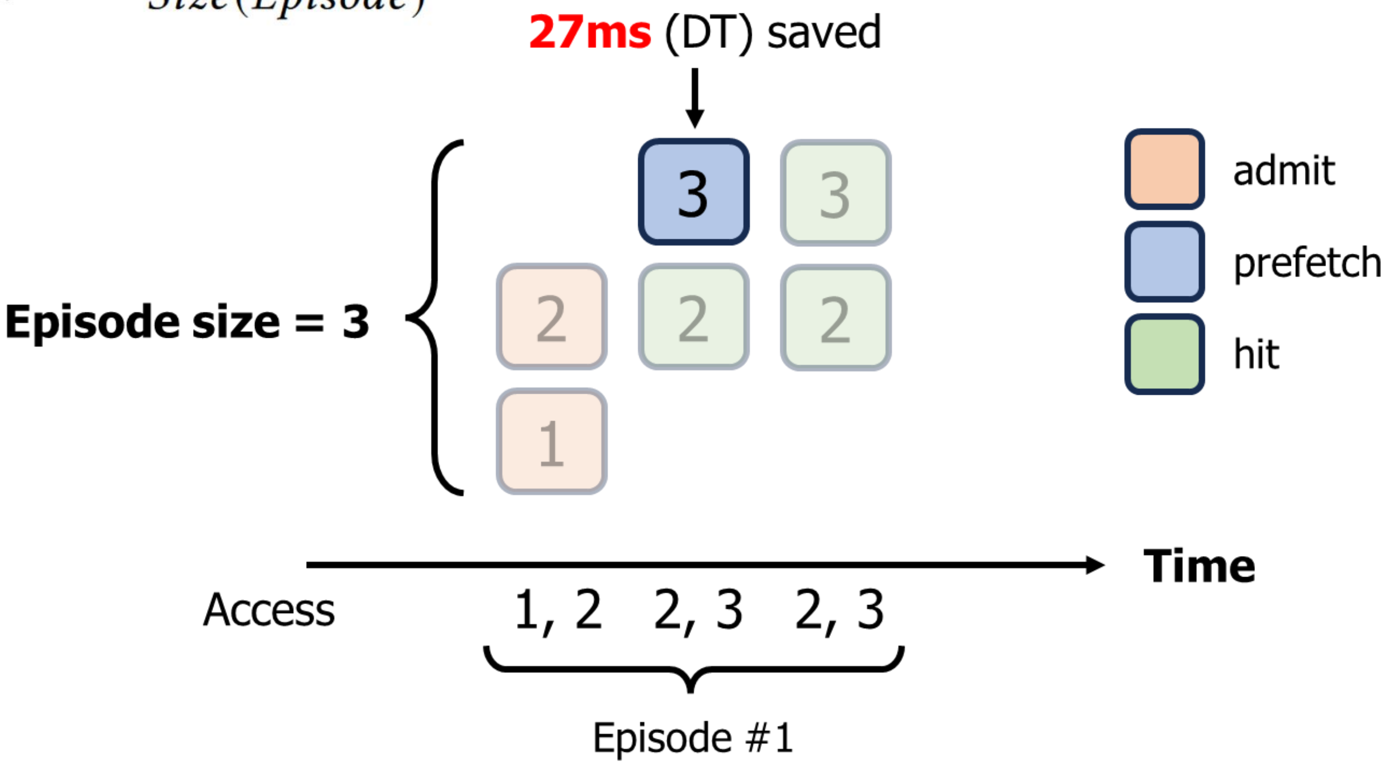
Optimal AP using episode

- DT saved: The amount of data transfer saved by caching the episode
- Size: The amount of flash writes required to cache the episode
- $Score(Episode) = \frac{DTSaved(Episode)}{Size(Episode)}$



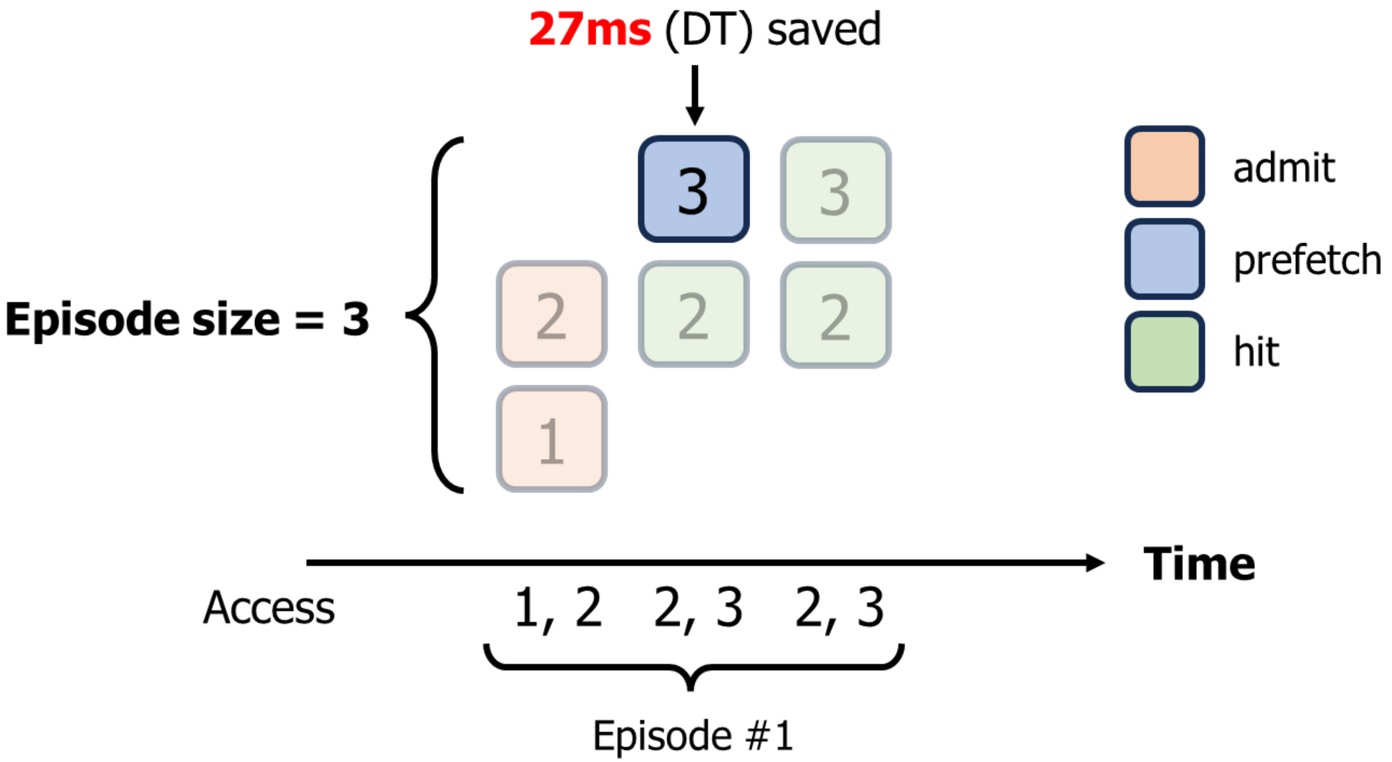
Optimal AP using episode

- DT saved: The amount of data transfer saved by caching the episode
- Size: The amount of flash writes required to cache the episode
- $$Score(Episode) = \frac{DTSaved(Episode)}{Size(Episode)}$$



Optimal AP using episode

- $Score(EP) = \frac{DT\ Saved(Ep)}{FlashWrites(Ep)} = \frac{27ms}{3\ flash\ writes} = 9$
- $Score(Ep) > Cutoff_{Target\ Flash\ Write\ Rate}$



Challenges in the flash caching

- Flash admission
  - Write cost is determined at caching time
- ML admission
  - Correct optimization metric not obvious → Disk-head Time (DT)
  - Train on all accesses in a trace, but focus on those that result in misses→ Episode
- Prefetching policy
  - Misprefetching is costly as it consumes writes and extra DT

Prefetching Policy

- Benefit: decreased DT
- Cost:
  - Overfetch: Minimal DT reduction effect
  - Underfetch: Excessive writes leading to performance degradation
- When?
  - OPT-Ep-Start: Fetch all necessary segments from the start of the episode to maximize DT reduction
- What?
  - Whole-Block: Prefetch the entire 8MB block
  - OPT-Range: Prefetch only the optimal segment range determined by the episode model

Baleen

- Provide episode-based solutions
  - How to train **ML-based admission policy**
  - Using **prefetch** to improve beyond admission policies

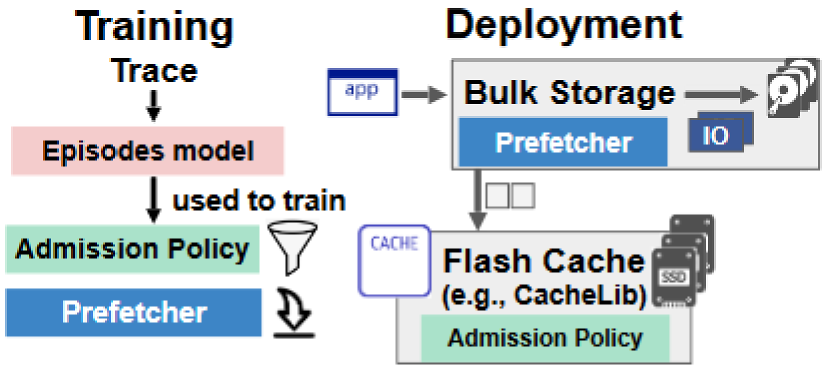


Figure 6: Architecture. An admission policy in CacheLib decides whether to admit items into flash. Prefetching (preloading of data beyond current request) takes place in Tectonic.

Baleen – ML admission policy

- Training Baleen's ML admission policy
  1) How to generate training data and labels from episodes
  2) What features and architecture we use for the ML admission model
  3) how we determine appropriate values for training parameters
  4) how we implement ML admission in CacheLib

Baleen – ML admission policy

- Training Baleen's ML admission policy

  1) how we generate training data and labels from episodes

  - Only the first 6 accesses from each episode are incorporated into training data
  - Baleen learns to imitate OPT

Baleen – ML admission policy

- Training Baleen's ML admission policy

  2) what **features** and architecture we use for the ML admission model

  - Metadata feature
    - **Request Source Identification:** identify the provenance of request (namespace, user)
  - Online dynamic feature
    - **Access Count Tracking:** Records the number of times each block is accessed
    - **IO Statistics:** Tracks the number of input/output operations for each block and the cumulative segment IO operations.

Baleen – ML admission policy

- Training Baleen's ML admission policy

  2) what features and **architecture** we use for the ML admission model

  - Model: **binary classification**
  - the model outputs a probability for admitting misses
  - We admit misses if this probability exceeds the policy threshold

Baleen – ML admission policy

- Training Baleen's ML admission policy

  3) how we determine appropriate values for training parameters

     - Repeated Calculations to Determine Eviction Age and Policy Threshold Parameters
     - Online simulation

       → Iterate until the measured average eviction age (EA) converges with the assumed EA

       → Calculate the policy threshold needed to achieve the desired flash write rate

  4) how we implement ML admission in CacheLib

Baleen – ML Prefetcher

- ML-Range
  - Use of Two Regression Models
    1. Predict the starting point of an episode
    2. Predict the ending point of an episode
  - Train with size-related features
    - Access start index, access end index, access size

Baleen – ML Prefetcher

- ML-When
  - To reduce the effect of prefetching on average episode eviction age
  - Model: binary classification
  - Selected by ϵ

$$\text{ML-When}(eps) = PFBenefit_{eps}^{ML-Range} > \epsilon$$

Evaluation

- Comparative group
  - CoinFlip
    - On a miss, segments for an access are either all admitted, or not at all, with probability p
  - RejectX
    - Rejects a segment the first X times it is seen, use X=1 here
    - Past accesses are tracked using probabilistic data structures like Bloom filters
  - CacheLib with ML policy
    - Improved limitations of Flashield and non-episode-related features

Evaluation

- How to balance #HDD against # SDDs
  - TCO: total cost of HDD reads and written flash byte

$$\mathrm{TCO}_1 \propto \underbrace{\frac{\mathrm{PeakDT}_1}{PeakDT_0} \cdot \#HDDs_0}_{\textbf{Cost of HDD}} + \underbrace{\frac{Cost_{SSD}}{Cost_{HDD}} \cdot \frac{\mathrm{FlashWR}_1}{FlashWR_0} \cdot \#SSDs_0}_{\textbf{Cost of SSD}} \quad (2)$$

- Lower peak load ➔ Fewer hard disks ➔ Lower TCO

Evaluation

TCO: total cost of HDD reads and written flash byte

$$TCO \propto Cost_{HDD} \cdot \#HDDs + Cost_{SSD} \cdot \#SSDs$$

- Baleen reduce TCO and Peak DT

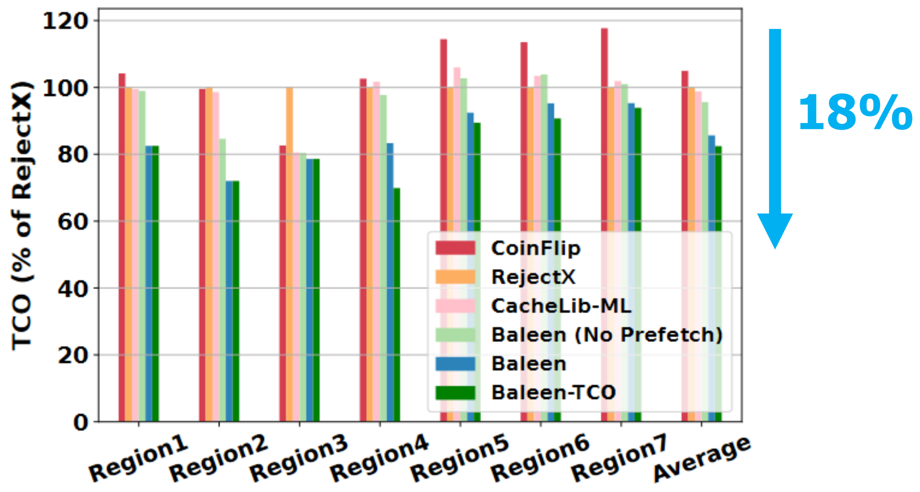


Figure 8: Baleen-TCO reduces TCO.

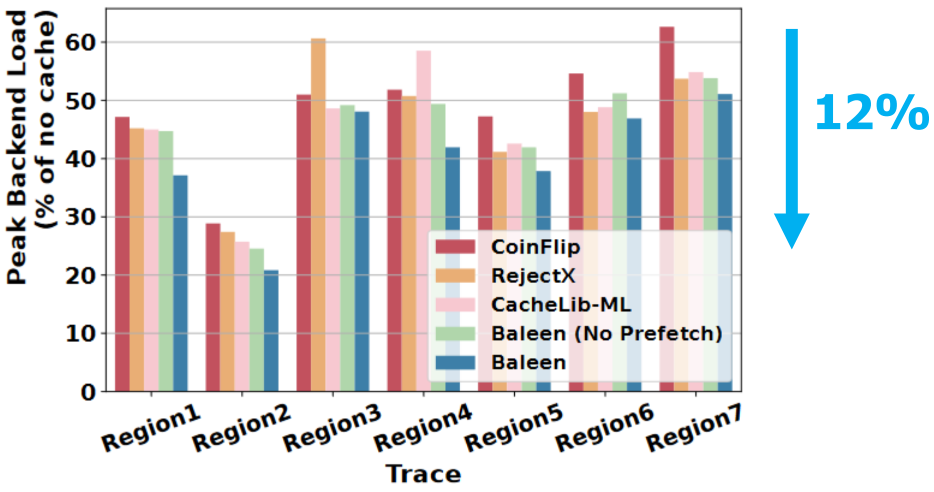


Figure 9: Baleen reduces Peak DT.

Training on episodes is effective for ML prefetching, and allowing Baleen to efficiently reduce TCO and peak DT.

Evaluation

- ML-guided prefetch

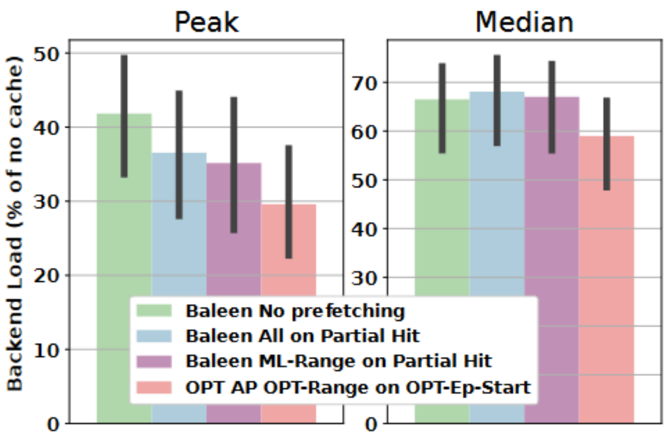


Figure 13: ML-Range saves Peak DT. ML-Range outperforms the baseline (whole block) and No Prefetching at the expense of Median DT.

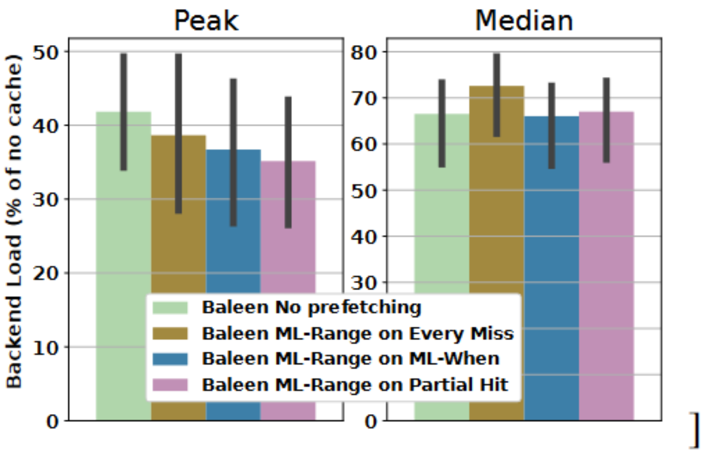


Figure 14: Choose *when* to prefetch. Indiscriminate prefetching (on Every Miss) can hurt. Using ML-When or Partial Hit reduces Peak DT without compromising Median DT.

ML-Range efficiently reduce peak DT by limiting the number of segments prefetched

ML-When helps Baleen discriminate between beneficial and bad prefetching

Evaluation

- Explicitly optimizing Peak DT

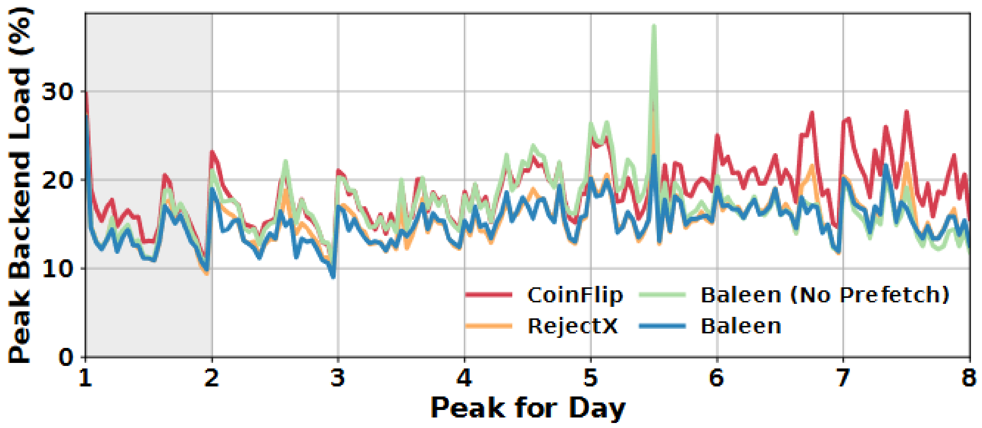


Figure 15: Testbed backend load on Region1. Day 1 (shaded) is used as training data. Peak is on Day 5 and is lowest for Baleen.

Baleen can be aware of the current load level and able to adapt to optimize Peak DT

Conclusion

- Large-scale servers use Flash Cache to mitigate the slow bandwidth of HDDs
  - However, due to the limited lifespan of SSDs, proper admission policies are required
- Instead of using traditional performance evaluation metric, which are inadequate, Baleen used DT and TCO as performance measures
- Baleen used an episode model to improve ML learning efficiency.
  - ML-range: predict a range of segments for prefetching
  - ML-When: Avoid waste caused by misprefetching
- As a result, Baleen lowered DT by 12% and TCO by 17%, effectively reducing costs.

Thank you

Q & A

Optimal AP using episode

- 왜 optimal 한가?

  - 전체 접근 이력을 사용하여 각 episode에 대한 완전한 정보를 가지고 있다

  - 따라서 장기적인 이득이 극대화된다.

  - Budget 고려함

  - 이론적으로 최적 결과가 된다.

Workload

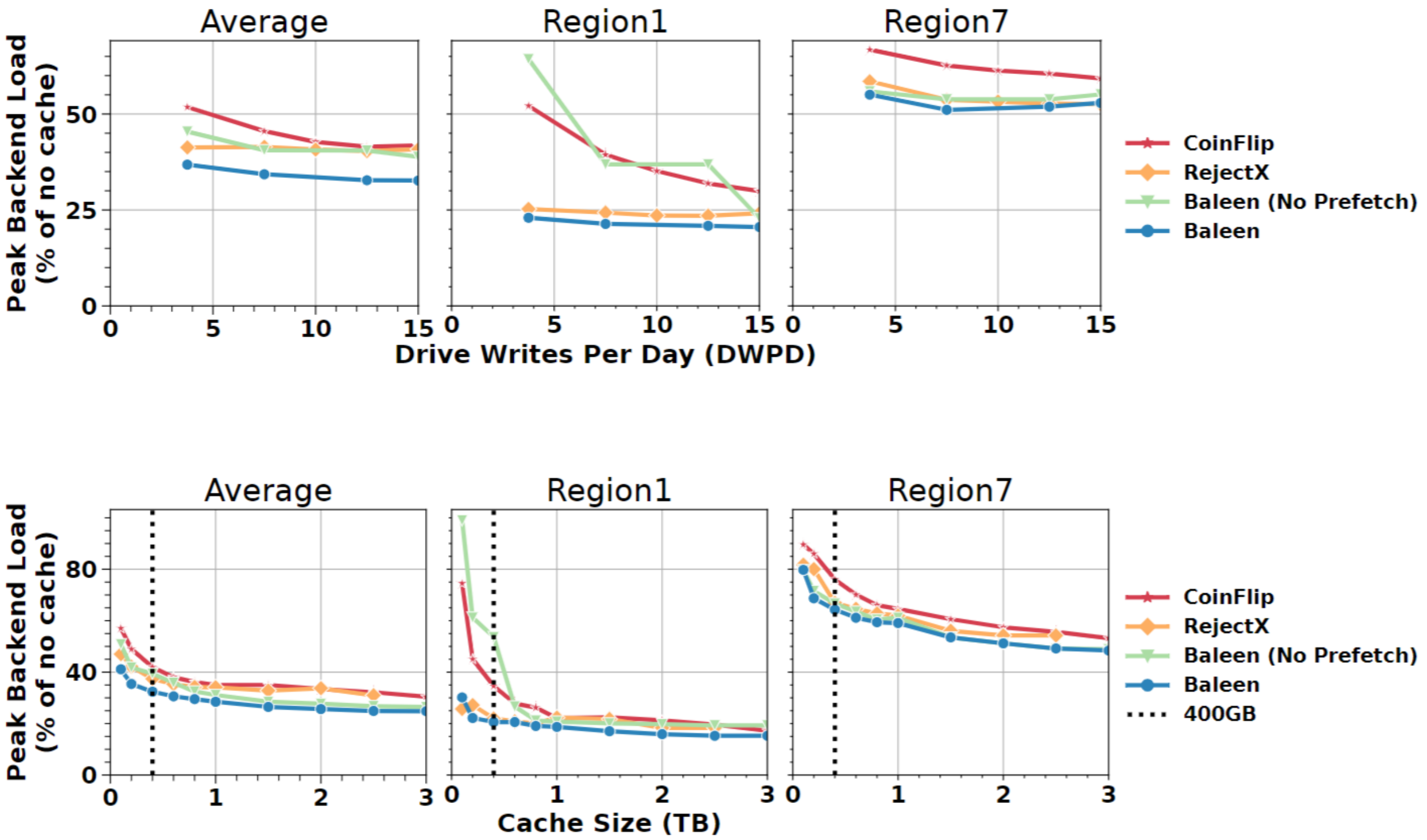


Figure 10: Benefits at higher write rates & cache sizes.