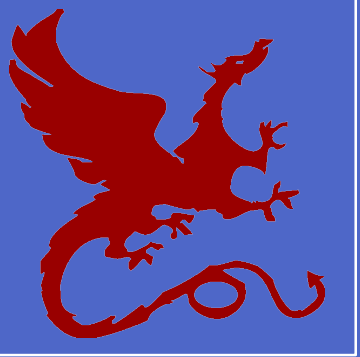
TEACHING AN OLD CACHE NEW TRICKS: LEARNING BETTER CACHING POLICIES ONLINE

Nathan Beckmann, CMU

ML for Systems @ISCA2019



Learning better caching policies online

- **Caching matters:** Caches are everywhere & often determine systems' end-to-end performance.
- **Caching is hard:** Deciding *what to keep in the cache* is hard. Workloads vary too much & the cache has ~no control over its input trace.
- **Caching practice is expensive & fragile:** Most caches use hand-tuned heuristics

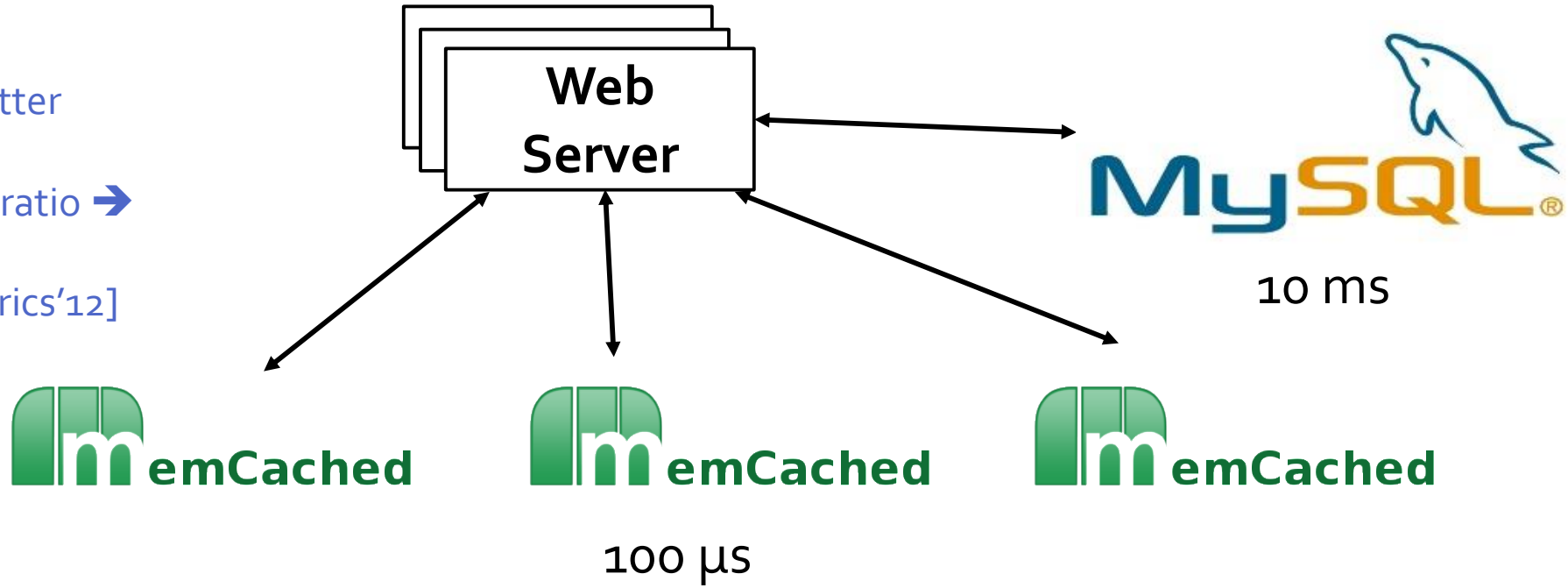
- We'll discuss **caching policies that *learn online*** to tune themselves w/o heuristics
  - Vanilla reinforcement learning does **not** work!
  - Option #1: Cache modeling + Bayesian inference
  - Option #2: Learn to imitate optimal

Caching matters!

Cache is 100× faster than back-end system

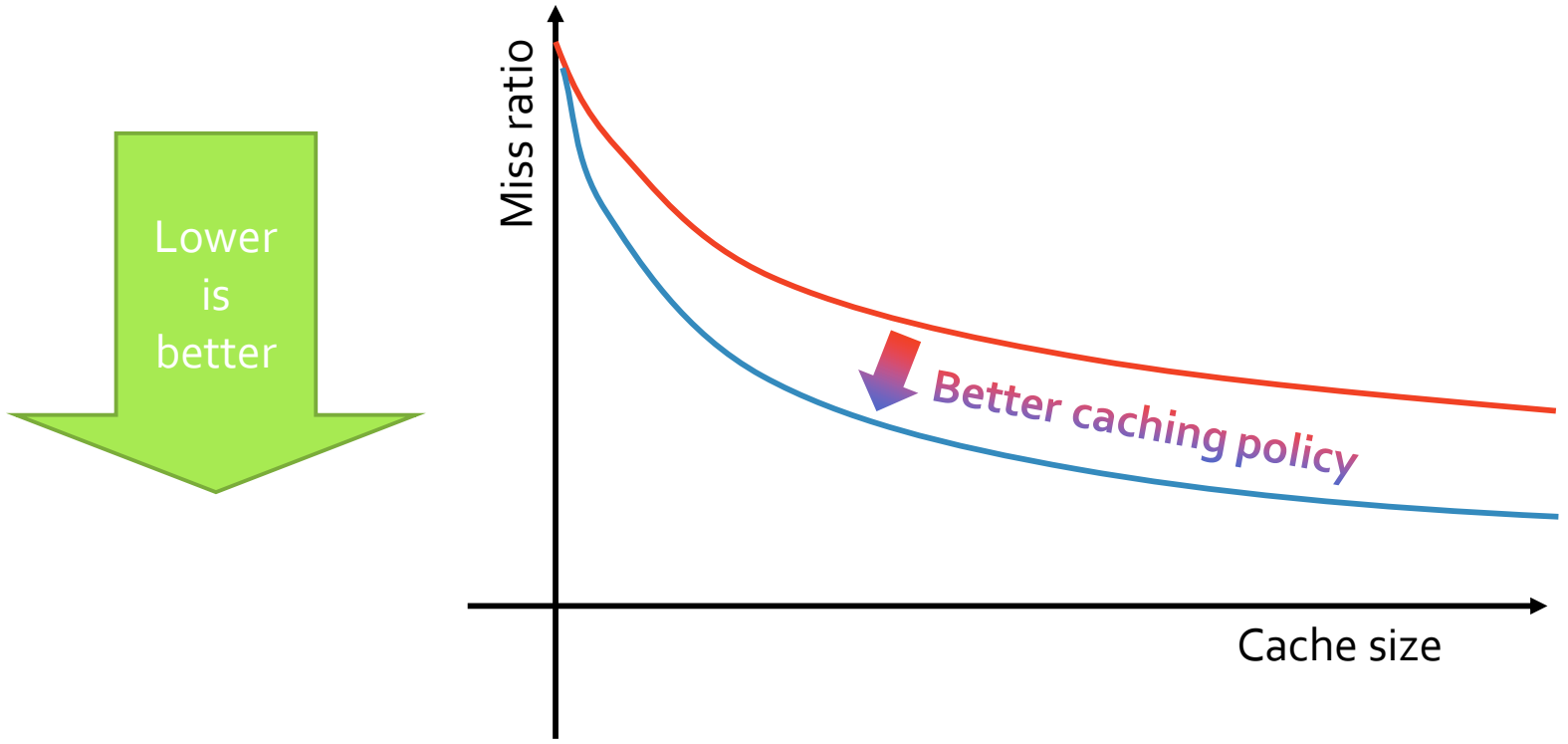
Even small hit rate improvements matter

E.g., 1% better hit ratio ➔ 35% lower latency [Atikogflu, Sigmetrics'12]



Think smarter, not larger

- Empirically, hit ratio follows log-linear law: 2 × larger cache ➔ +Δ hit ratio



Finding a good eviction policy is hard

- **Applications vary a lot**
  - Access objects in different patterns
  - Objects have different sizes

Web caching: Object sizes vary by 9 orders of magnitude [Berger+, NSDI'17]

- Prior policies use heuristics that combine **recency** and **frequency**
  - No theoretical foundation
  - Require hand-tuning ➔ fragile to workload changes

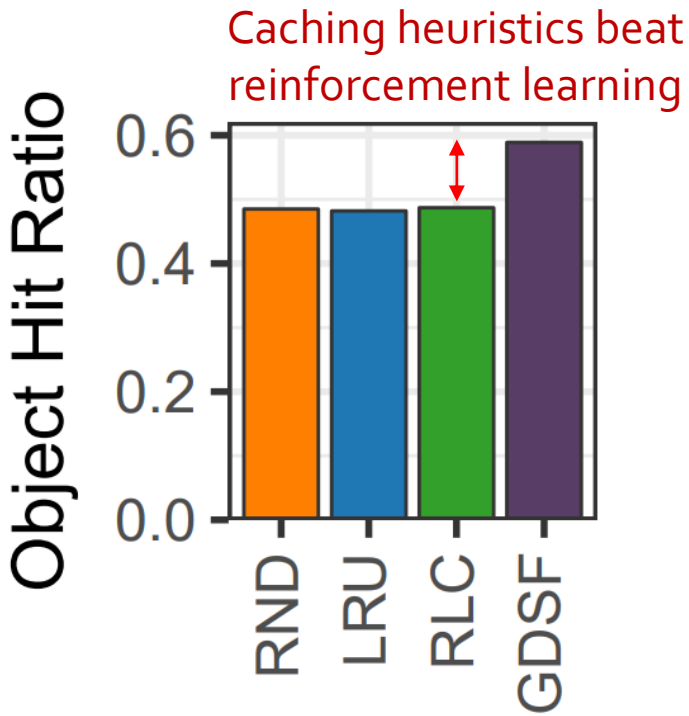
- No single policy works for all workloads

THEME: LEARNING ALONE IS NOT ENOUGH!

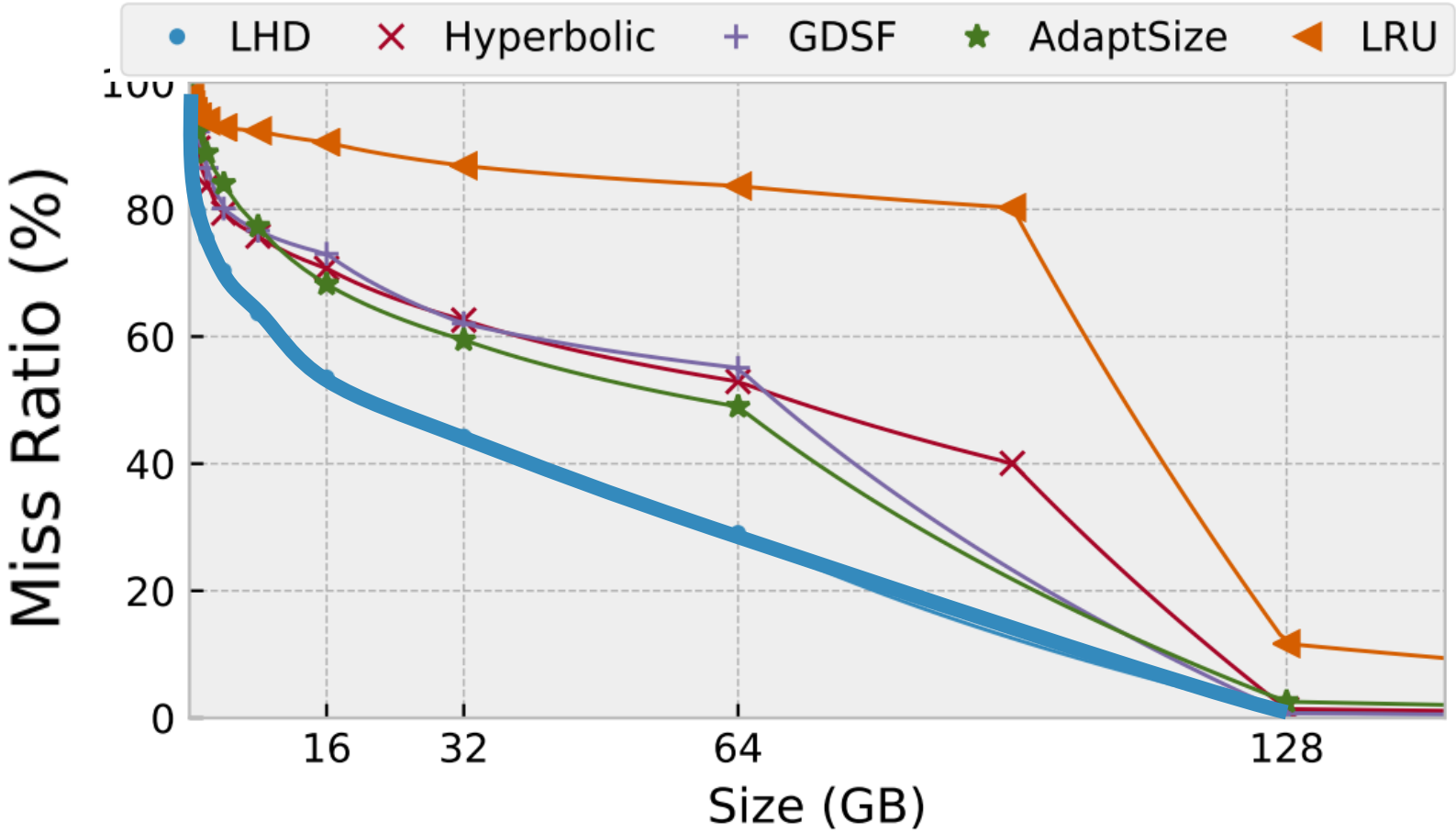
COMBINE LEARNING W/ CACHE THEORY

Reinforcement learning struggles in caching

- "Harvesting randomness to optimize distributed systems" [Lecuyer+, HotNets'17]



Learning online ➔ Much better cache performance!



MSR "src1_0" storage workload

[Beckmann+, NSDI'18]

CACHE THEORY + BAYESIAN INFERENCE

Maximizing Cache Performance Under Uncertainty
Nathan Beckmann, Daniel Sanchez – HPCA'17

LHD: Improving Cache Hit Rate by Maximizing Hit Density
Nathan Beckmann, Haoxian Chen, and Asaf Cidon – NSDI'18

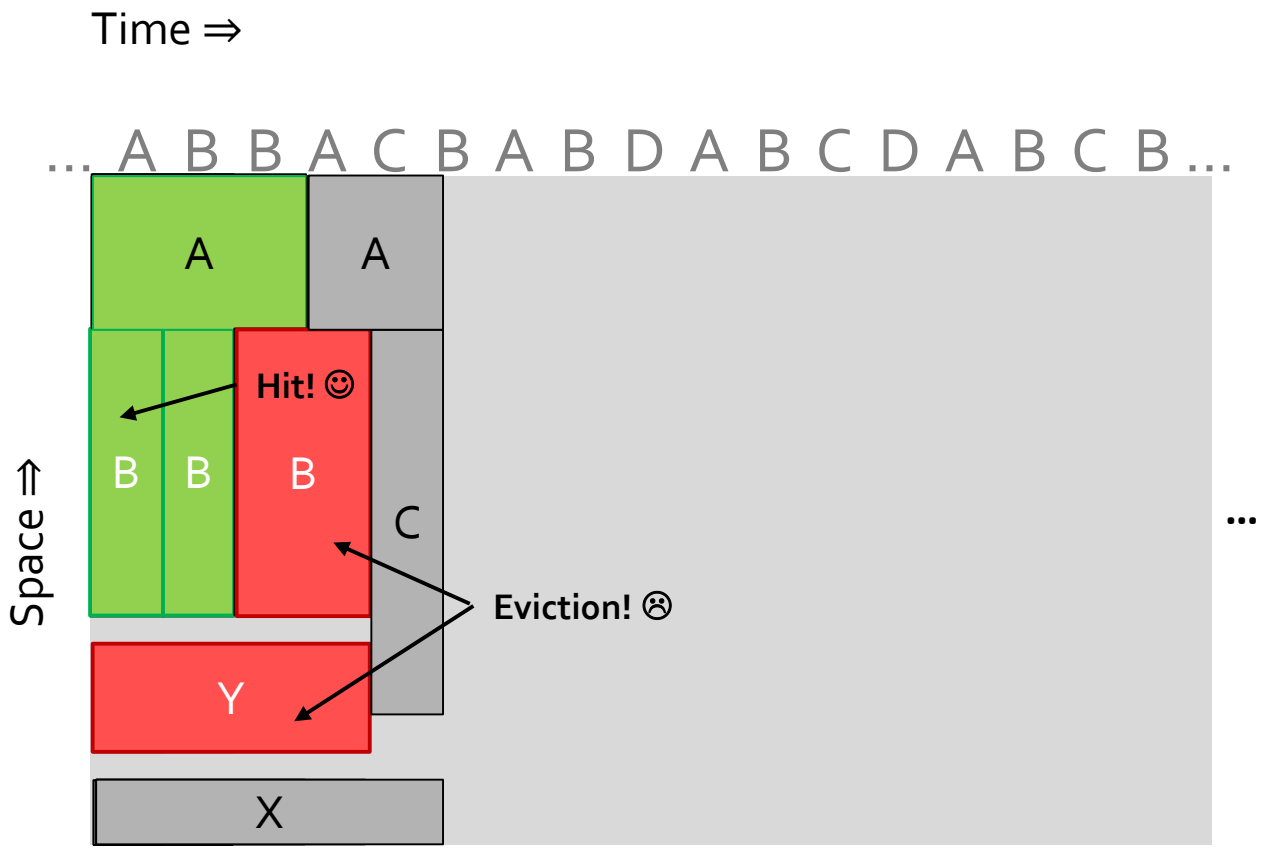
Q:
WHAT'S THE "RIGHT" METRIC FOR CACHING POLICIES?

Caching theory: The "big picture"

- *Goal:* Maximize cache hit rate
- *Constraint:* Limited cache space
- *Uncertainty*: In practice, don't know what is accessed when

Where does cache space go?

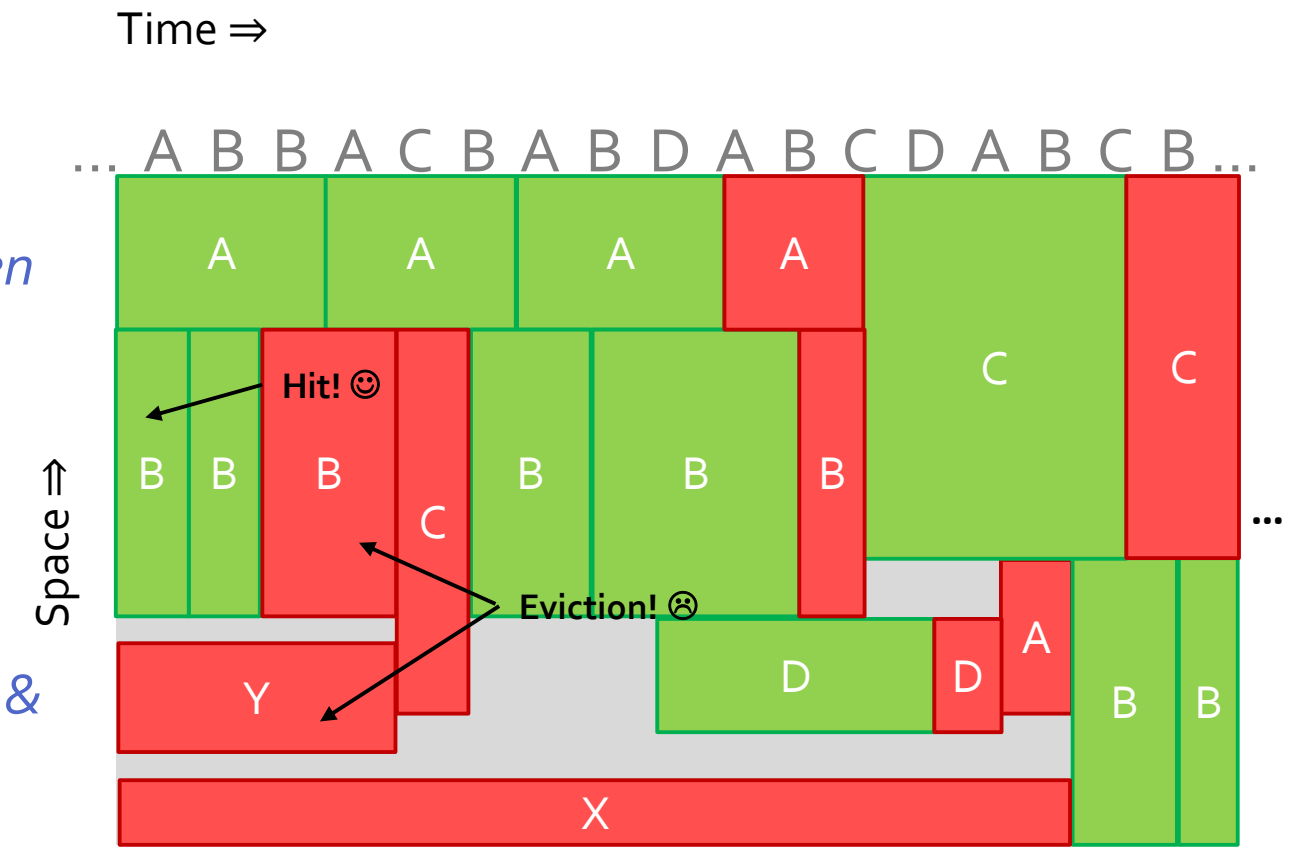
- Let's see what happens on a short trace…



Where does cache space go?

- Green box = 1 hit
- Red box = 0 hits
- ➔ *Want to fit as many green boxes as possible*

- Each box costs resources = area
- ➔ *Cost proportional to size & time spent in cache*



THE KEY IDEA: HIT DENSITY

Our metric: Hit density (HD)

- Hit density combines **hit probability** and **expected cost**

$$\text{Hit density} = \frac{\boldsymbol{Object's\ \text{hit probability}}}{Object's\ \text{size} \times \boldsymbol{Object's\ \text{expected lifetime}}}$$

- Least hit density (LHD) policy: Evict object with smallest hit density
- **But how do we predict these quantities?**

Estimating hit density (HD) via probability

- Age – # accesses since object was last requested
- Random variables
  - H – hit age (e.g., $\mathrm{P}[H = 100]$ is probability an object hits after 100 accesses)
  - L – lifetime (e.g., $\mathrm{P}[\mathrm{L} = 100]$ is probability an object hits *or is evicted* after 100 accesses)

- Easy to estimate HD from these quantities:

$$HD = \frac{\sum_{a=1}^{\infty} \mathrm{P}[H = a]}{Size \times \sum_{a=1}^{\infty} a\, \mathrm{P}[L = a]}$$

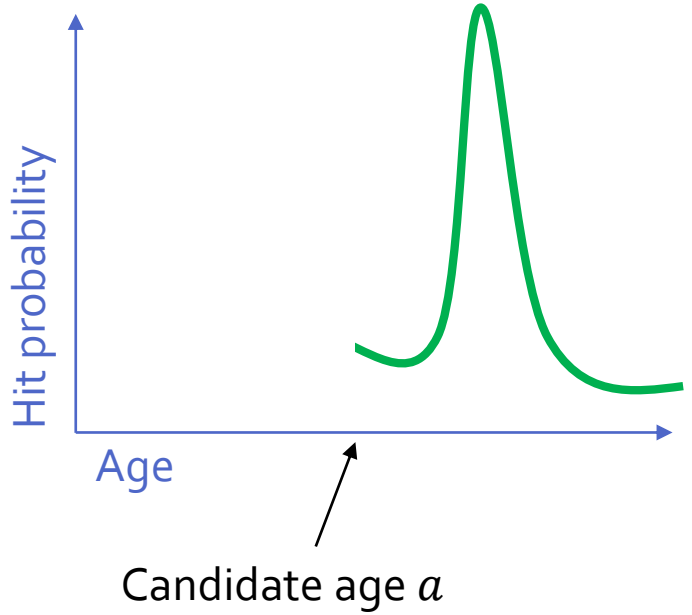
hit probability

expected lifetime

Bayesian inference from an object's age

- Estimate HD using **conditional probability**
- Monitor distribution of H & L online

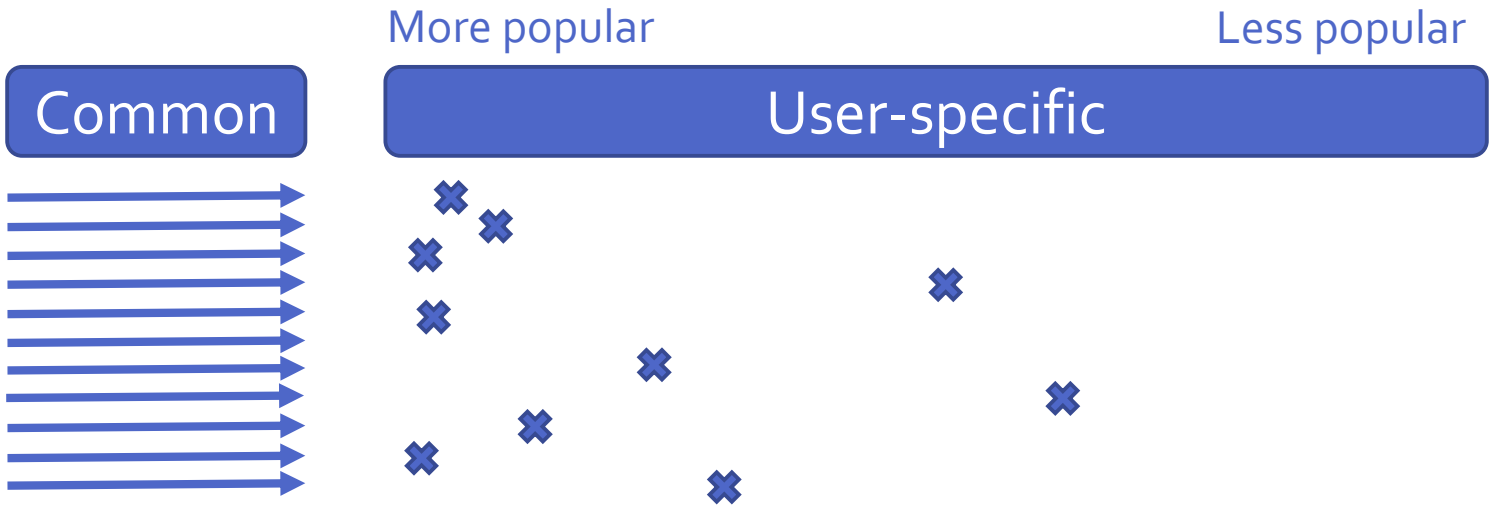
- By definition, object of age a wasn't requested at age $\leq a$
- ➔ Ignore all events before a



- Hit probability $= \mathrm{P}[\text{hit} \mid \text{age } a] = \frac{\sum_{x=a}^{\infty} \mathrm{P}[H=x]}{\sum_{x=a}^{\infty} \mathrm{P}[L=x]}$
- Expected remaining lifetime $= \mathrm{E}[L-a \mid \text{age } a] = \frac{\sum_{x=a}^{\infty} (x-a)\,\mathrm{P}[L=x]}{\sum_{x=a}^{\infty} \mathrm{P}[L=x]}$

LHD by example

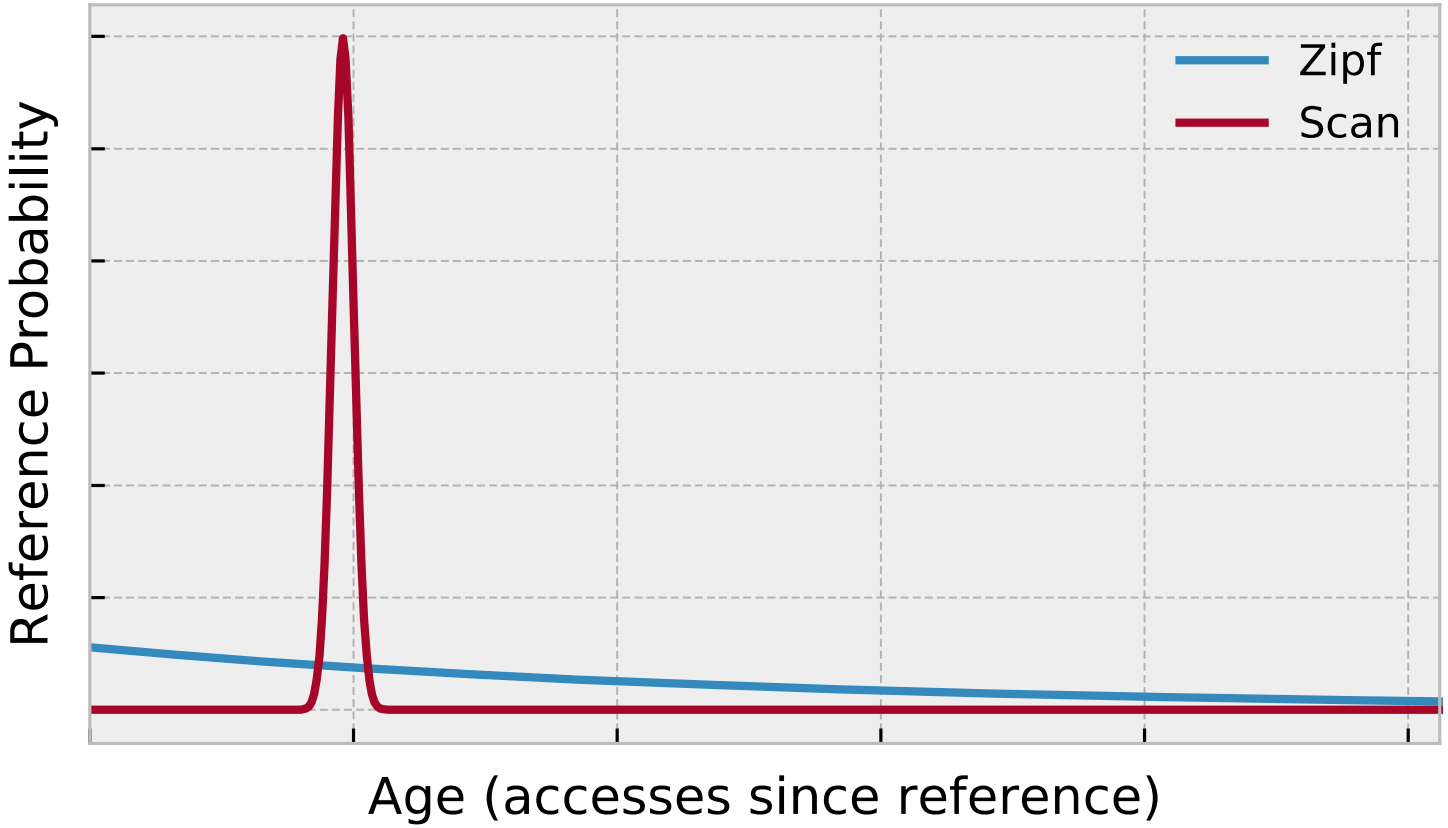
- Users ask repeatedly for common objects and some user-specific objects



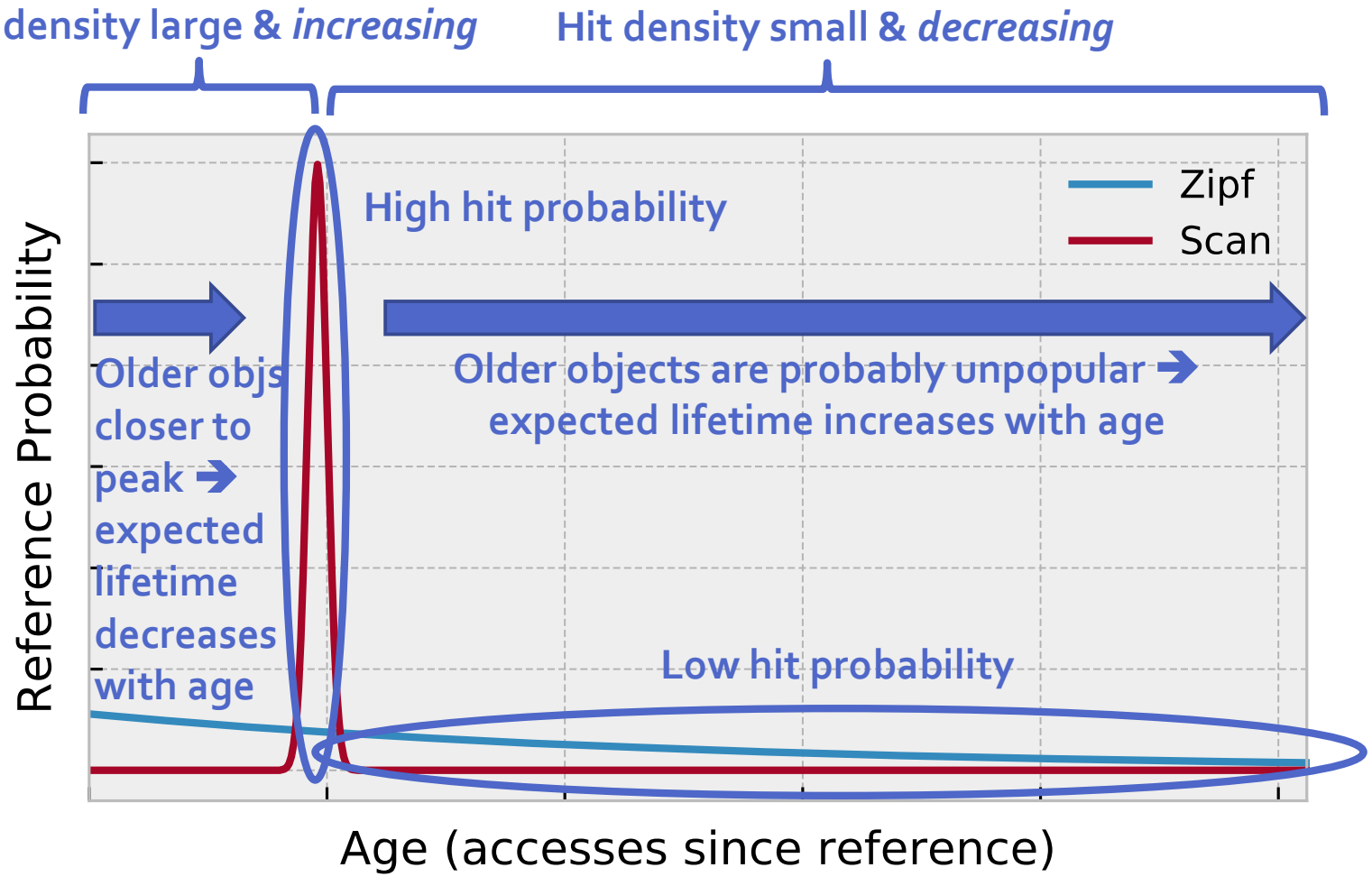
Best hand-tuned policy for this app:
Cache common media + as much user-specific as fits

Probability of referencing object again

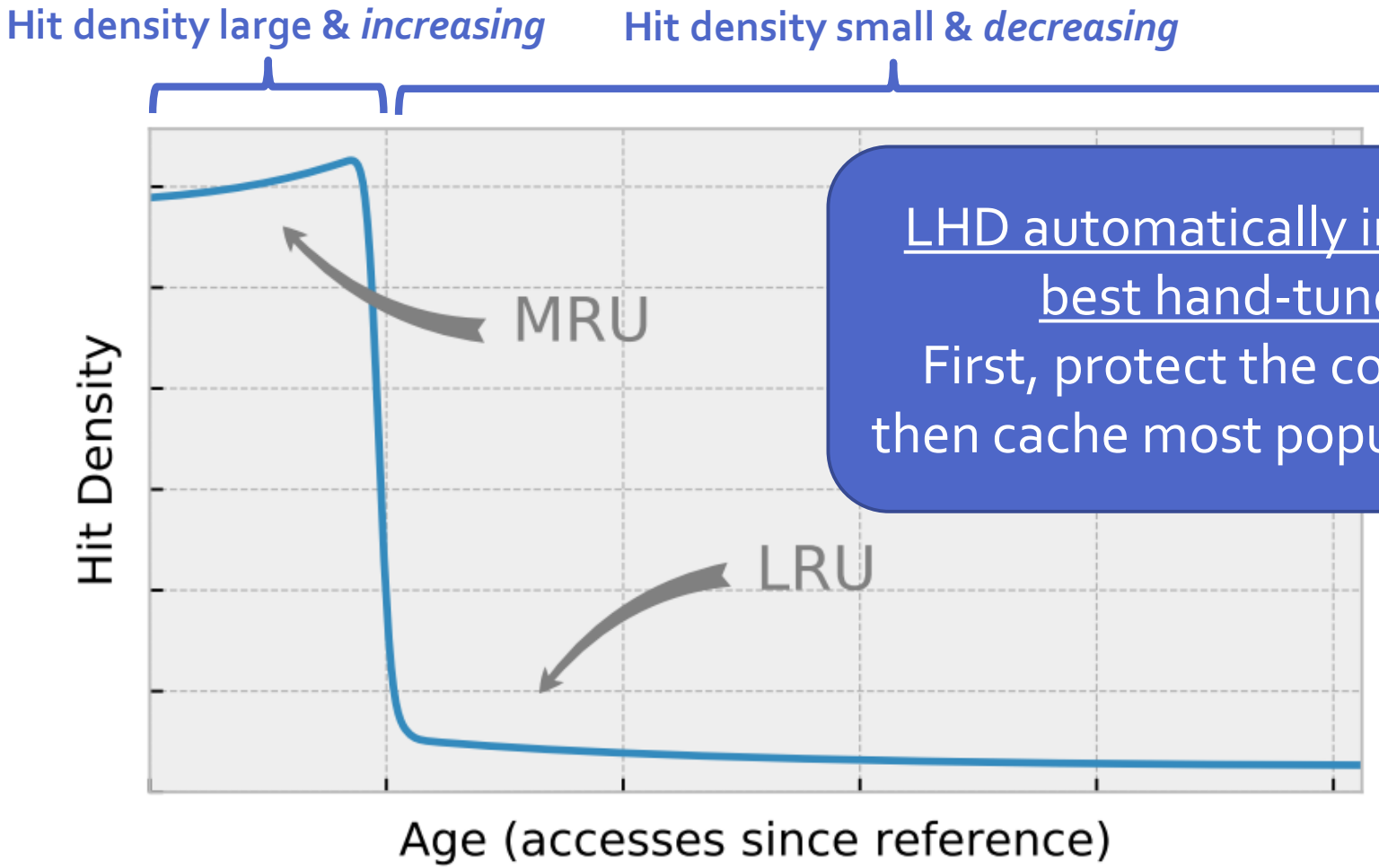
- Common object modeled as scan, user-specific object modeled as Zipf



LHD by example: what's the hit density?



LHD by example: policy summary



LHD automatically implements the best hand-tuned policy:
First, protect the common media, then cache most popular user content

Improving LHD using additional object features

- Conditional probability lets us easily add information!
- Condition H & L upon additional informative object features, e.g.,
  - *Which app requested this object?*
  - *How long has this object taken to hit in the past?*
- Features inform decisions ➔ LHD *learns* the "right" policy w/out heuristics

Inferring HD from add'l object features

- Bayesian inference accepts arbitrary new features
- E.g., suppose for an object feature Y takes value y

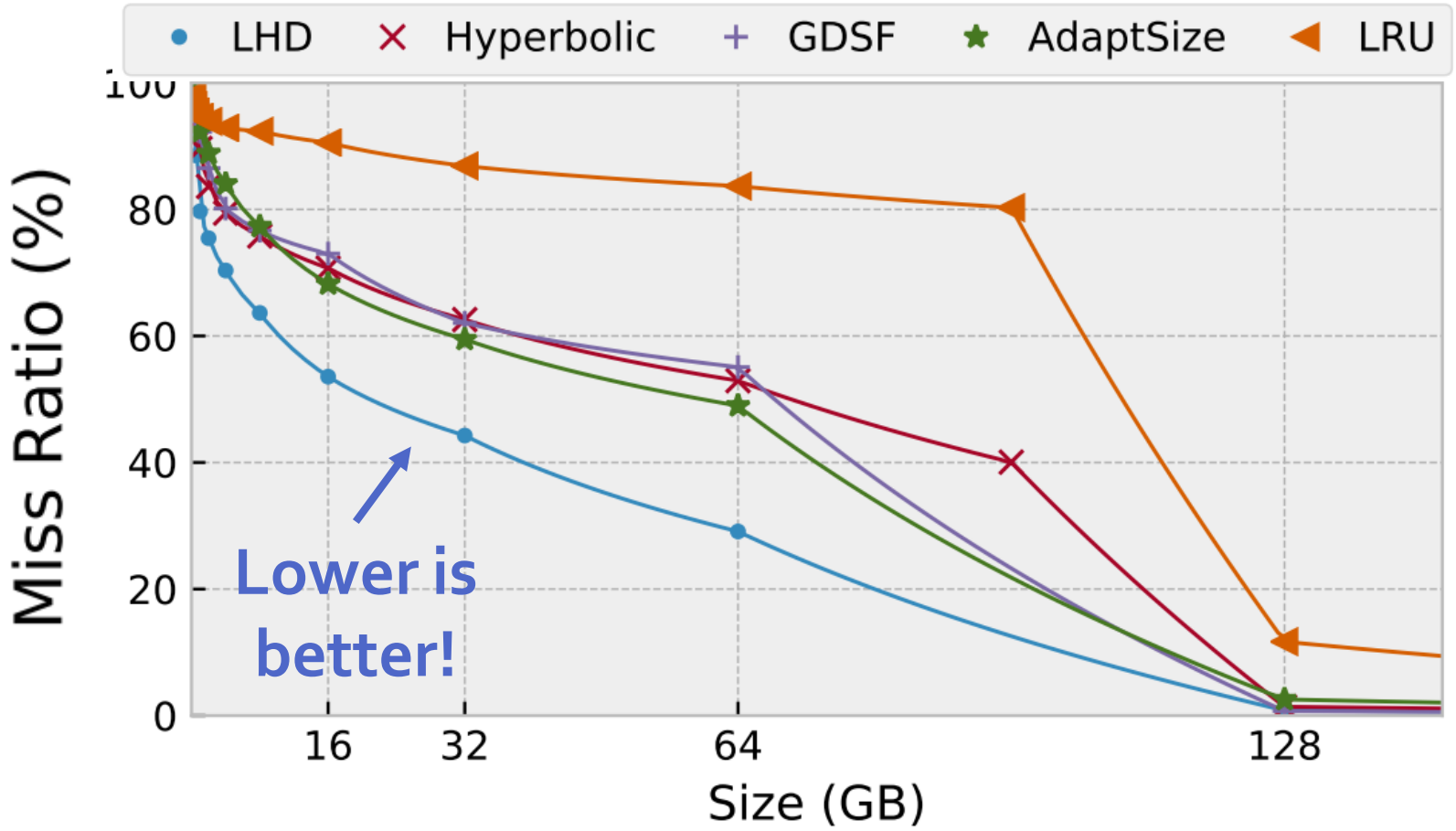
$$(\text{Hit Density} \mid Y = y) = \frac{\text{Hit probability} \mid Y = y}{\text{Expected remaining lifetime} \mid Y = y}$$

$$\text{Hit probability} = \mathrm{P}[\text{hit} \mid \mathrm{Y} = \mathrm{y}, \text{age } a] = \frac{\sum_{x=a}^{\infty} \mathrm{P}[H = x, \boldsymbol{Y = y}]}{\sum_{x=a}^{\infty} \mathrm{P}[L = x, \boldsymbol{Y = y}]}$$

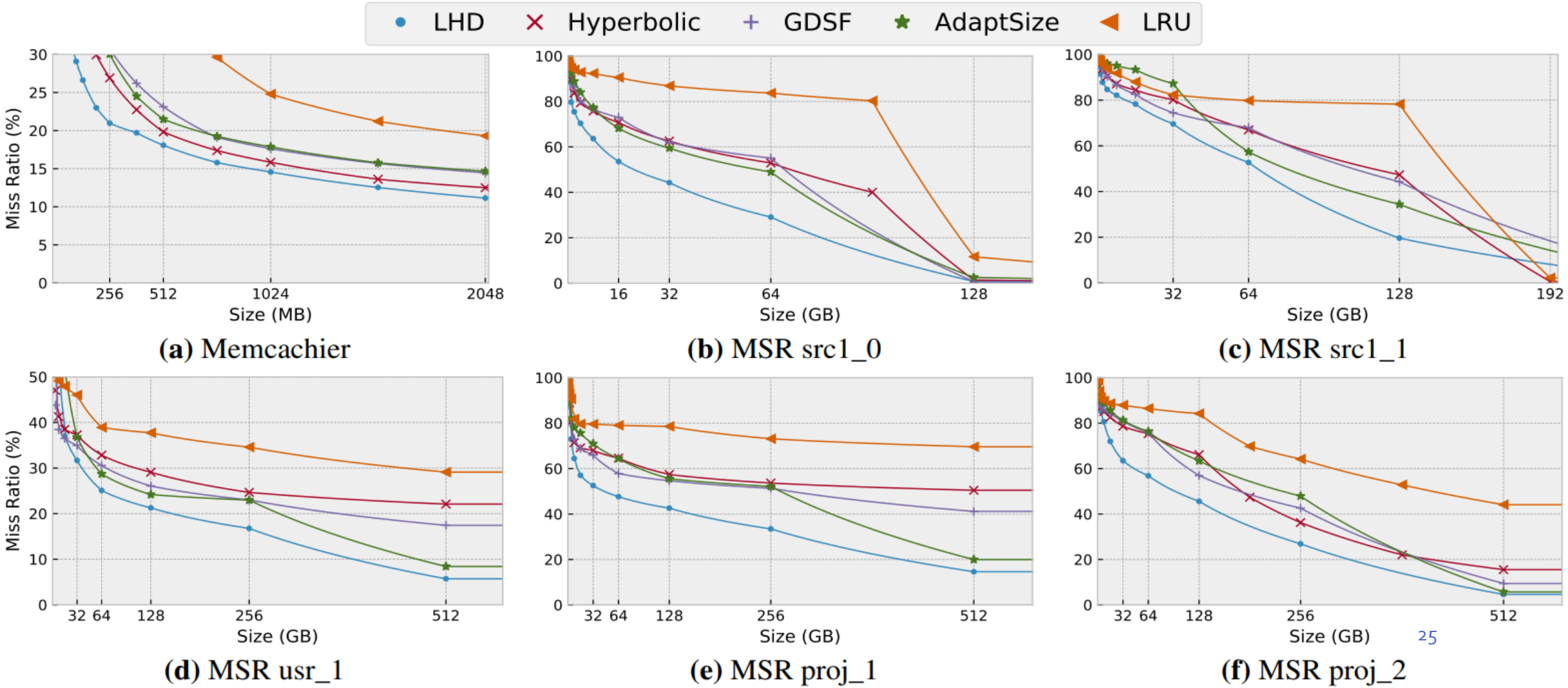
Features added w/ minimal changes to policy & no hand-tuning

$$\text{Expected remaining lifetime} = \mathrm{E}[L - a \mid \mathrm{Y} = \mathrm{y}, \text{age } a] = \frac{\sum_{x=a}^{\infty} (x - a)\, \mathrm{P}[L = x, \boldsymbol{Y = y}]}{\sum_{x=a}^{\infty} \mathrm{P}[L = x, \boldsymbol{Y = y}]}$$

LHD gets more hits than prior policies



LHD gets more hits across many traces



TRANSLATING THEORY TO PRACTICE

Good hit ratio ⇔ Poor throughput?

- Prior complex policies require **complex data structures**
  - Synchronization ➔ poor scalability ➔ unacceptable request throughput
  - E.g., GDSF uses $O(\log N)$ heaps

- Even $O(1)$ LRU is sometimes too slow because of synchronization
- ➔ Many key-value systems approximate LRU with CLOCK / FIFO
  - MemC3 [Fan, NSDI '13], MICA [Lim, NSDI '14]…

- *Can LHD achieve similar request throughput to production systems?* **YES!**

LHD implementation sketch (time view)

Compute estimated hit density for
different object features (age, app id, etc)

Training

Operation

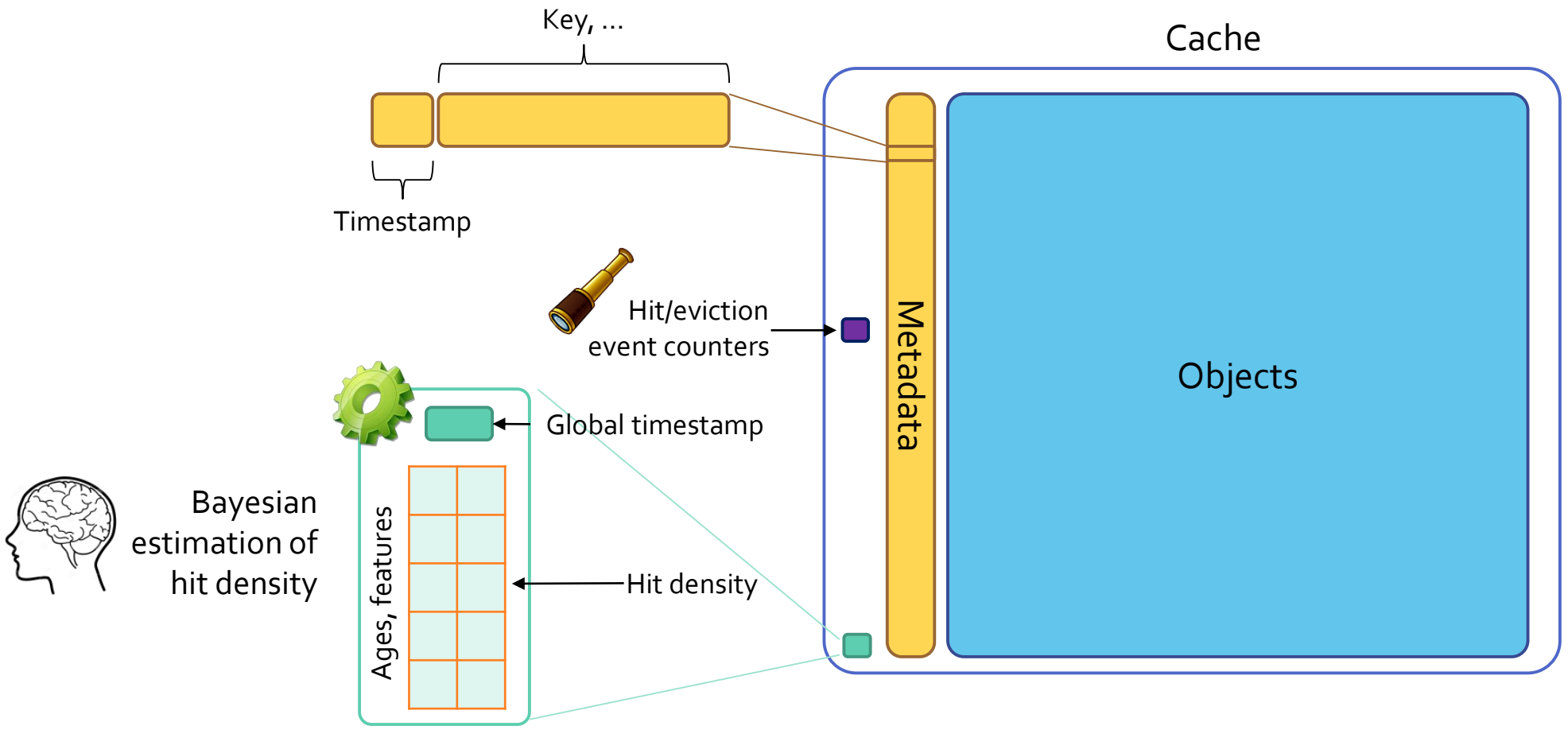
Time

Few thousand requests

Rank objects by pre-computing hit density
(Indexed by object age, …)

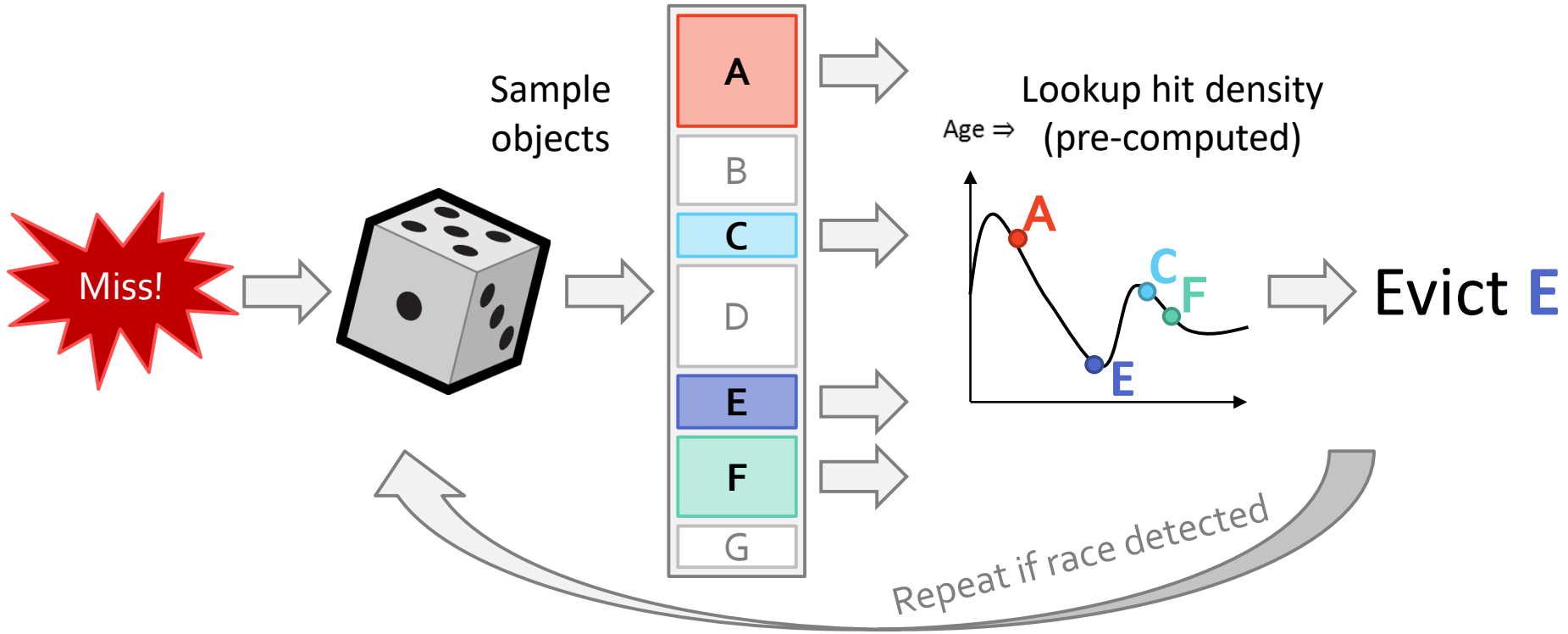
Count when hits & evictions happen (i.e., object age)

LHD implementation sketch (structures)

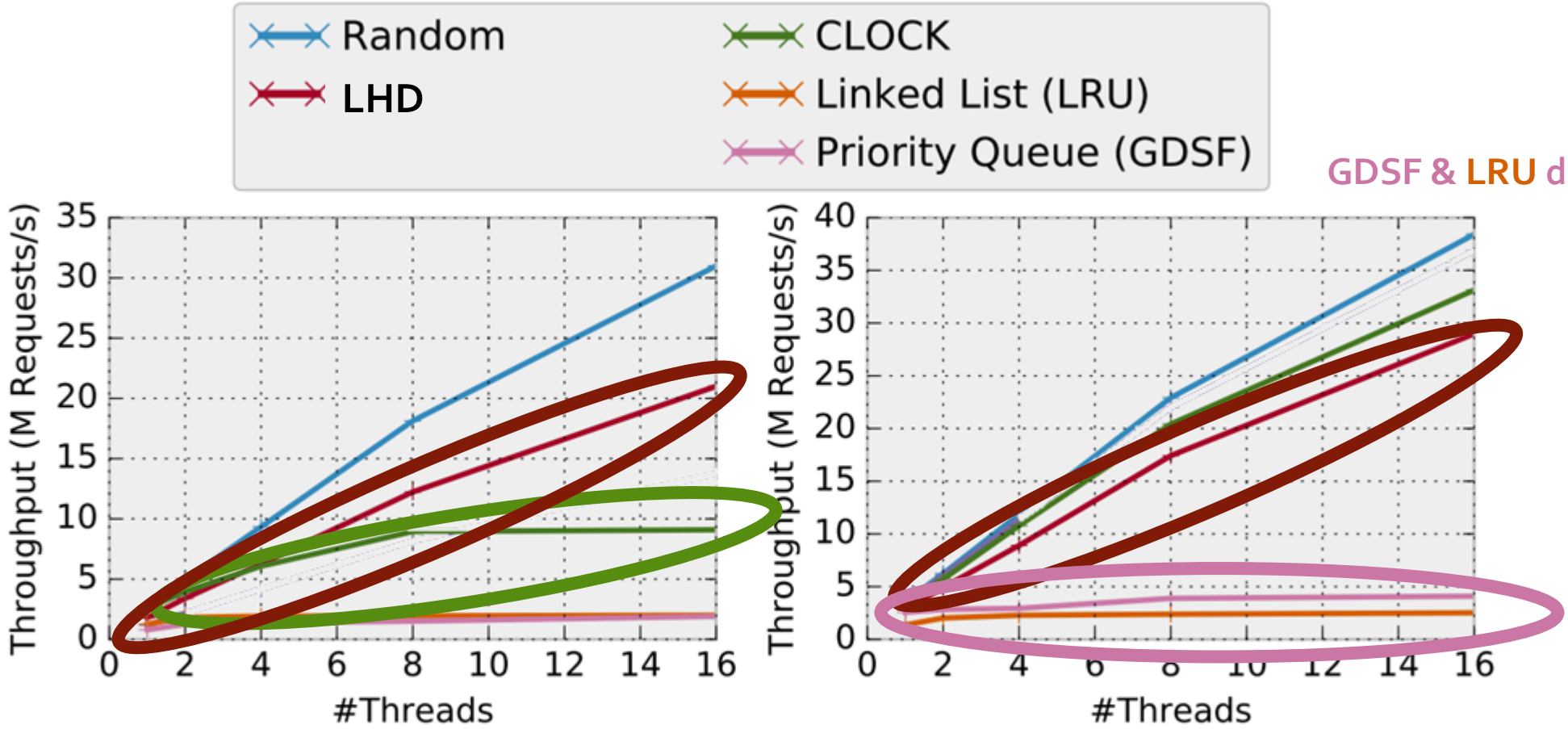


Fast evictions in LHD

- No global synchronization ➔ Great scalability!
  (Even better than CLOCK/FIFO!)



LHD gets best throughput



GDSF & LRU don't scale!

CLOCK doesn't scale when there are even a few misses!

LHD scales well with or without misses!

(a) 90% Hit ratio.

(b) 100% Hit ratio.

Future directions in Bayesian caching

- **Machine learning:** Learn which features matter
  - Many possible features (age, app id, size, frequency, burstiness, etc.)
  - Only a few features can be used (sample quality, runtime & memory overhead)
  - Caching systems should *learn* which features are important for a given workload online

- **Systems:** Real systems care about more than object hit ratio
  - Dynamic latency / bandwidth optimization
  - Optimizing end-to-end response latency (multi-object requests) [Berger+, OSDI'18]
  - Full system modeling (write endurance in FLASH/NVM, replicas in a distributed system)

CAN WE DO BETTER? LEARNING FROM OPTIMAL

Work by many others…

Back to the Future: Leveraging Belady's Algorithm for Improved Cache Replacement [Jain and Lin, ISCA'16]

Ongoing work by [Song, Berger, Li, Lloyd]

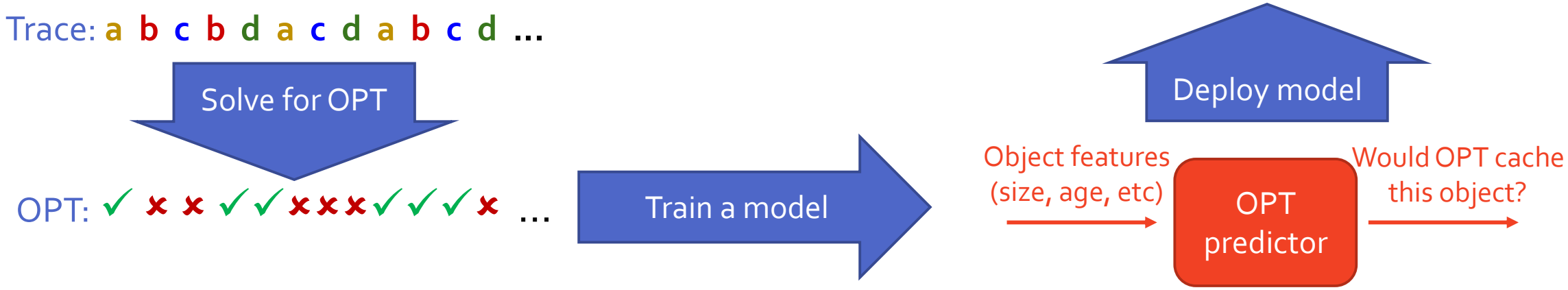
Reinforcement learning struggles in caching

- Caching is a chaotic, input-driven environment
  1. Little to no control over input
  2. Weak signal between decision (eviction) and reward (later hits / evictions)

- RL often struggles in input-driven environments [Mao+, ICLR'19]
- Need to reduce noise & enhance reward response

- Can avoid these problems by **learning to imitate OPT**

Learning the optimal caching policy



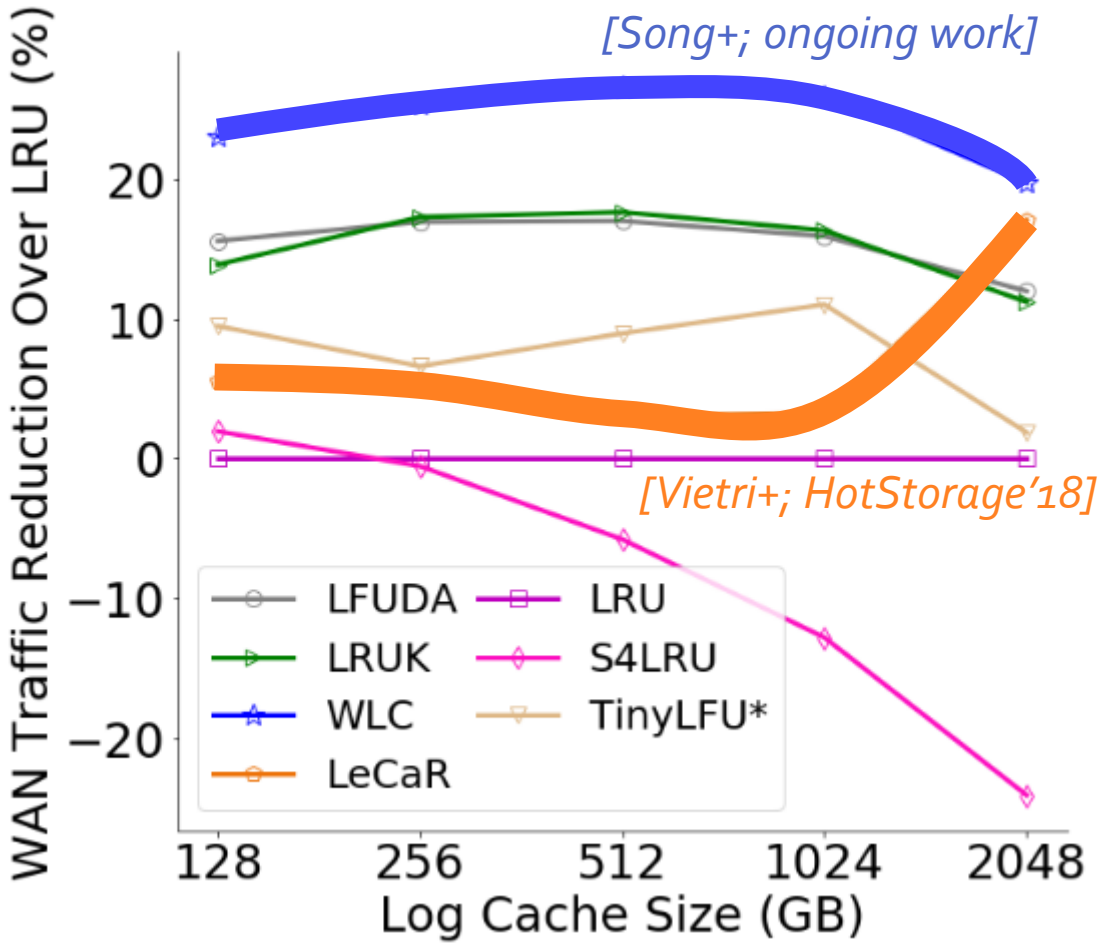
- HawkEye learns to mimic Belady replacement for CPU caches [Jain & Lin, ISCA'16]
  - Load PC is a good predictor for caching policies [Khan+, MICRO'10][Wu+, MICRO'11]
  - HawkEye realizes that load PC also correlates strongly with OPT's decisions
  - ➔ Simple hardware can learn OPT very effectively

Comparing learning approaches

- [Song+; ongoing work] learns to imitate OPT using various object features (age, size, etc.)
- [Vietri+, HotStorage'18] tunes parameters w/ RL

➔ Imitating OPT is more effective!

- (Pure RL is very bad, off bottom of figure)



COMPUTING THE OPTIMAL POLICY

Practical Bounds on Optimal Caching with Variable Object Sizes

Daniel Berger, Nathan Beckmann, and Mor Harchol-Balter

SIGMETRICS'18

How much better can we do?

- Computing optimal can be easy or hard, depending on caching problem
  - **Easy:** Objects are the same size & cost ➔ Belady (i.e., evict max. next-use distance)
  - **Less easy:** Objects have different costs ➔ Network flow methods
  - **Hard:** Objects have different sizes ➔ *Strongly NP-hard!*
  - **Hard:** Writes cost more than reads ➔ *MaxSNP-hard!*
    [McGuffey+, SPAA'19 BA]

Today's focus

Web caching: Object sizes vary by 9 orders of magnitude

Three lessons from this work:

1. **Optimal can be approximated efficiently**
   (Even when computing it exactly is hard)

2. **Obvious extensions to Belady don't always work well**
   (e.g., Next-use distance × object size, Knapsack heuristics)

3. **There is significant room for improving current policies:**
   (Large gap between the best online policies & optimal)

Prior OPT approximations

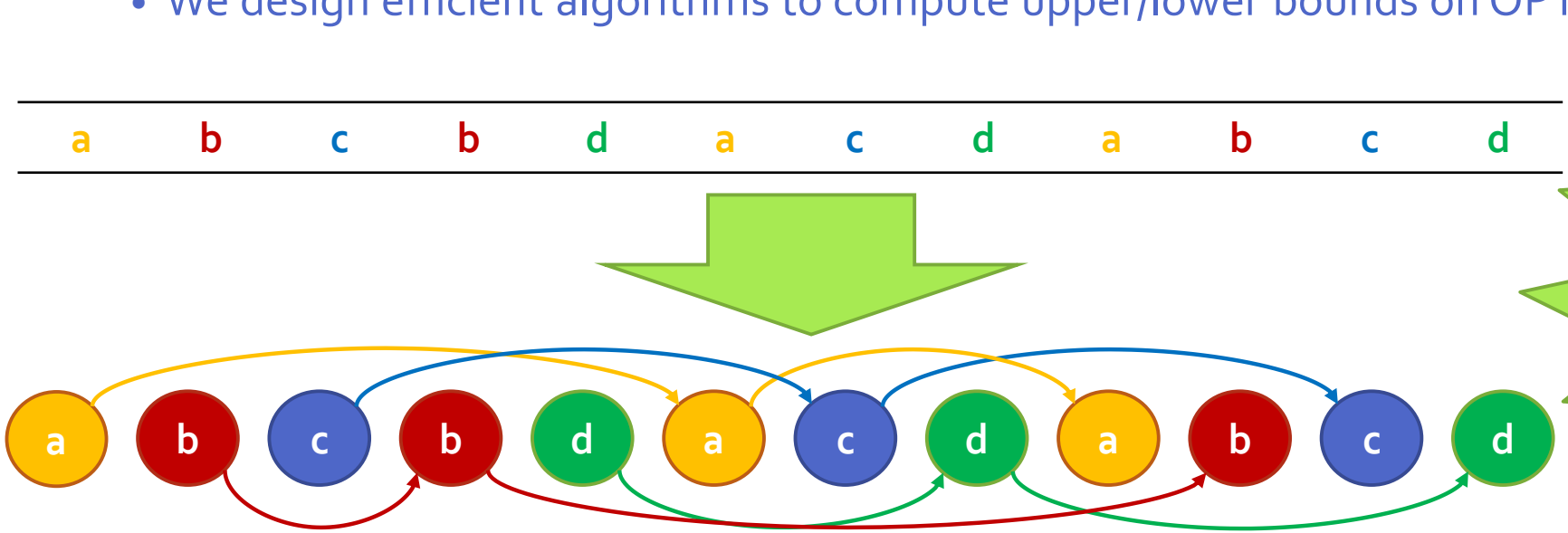
Technique	Time Complexity	Approximation Guarantee
OFMA [STOC'97]	$O(N^2)$	O(log cache size)
LP rounding [SODA'99]	$O(N^5)$	$O(\log (max_{size}/min_{size}))$
LocalRatio [JACM'01]	$O(N^3)$	**4**

State-of-the-art 4-approximation not useful

$O(N^3)$ is too slow anyway

Computing OPT efficiently on real traces

- **Insight:** Traces are not adversarial in practice
- **Idea:** Map OPT to min-cost flow problem ➔ $O(N^2 \log^2 N)$ solution
  - We prove that FOO introduces negligible error on real traces (specifically, under IRM)
  - We design efficient algorithms to compute upper/lower bounds on OPT



FOO: Flow-based Offline Optimal

FOO is accurate & efficient

Technique	Time Complexity	Approximation Guarantee
OFMA [STOC'97]	$O(N^2)$	O(log cache size)
LP rounding [SODA'99]	$O(N^5)$	$O(\log (\max_{size}/\min_{size}))$
LocalRatio [JACM'01]	$O(N^3)$	**4**
FOO*	$O(N^2 \log^2 N)$	**1**

* On non-adversarial traces seen in practice.

FOO is accurate & efficient

Technique	Time Complexity	Approximation Guarantee
OFMA [STOC'97]	$O(N^2)$	O(log cache size)
LP rounding [SODA'99]	$O(N^5)$	$O(\log (\max_{size}/\min_{size}))$
LocalRatio [JACM'01]	$O(N^3)$	**4**
FOO	$O(N^2 \log^2 N)$	**1***
PFOO	$O(N \log N)$	**≈1.06****

* On non-adversarial traces seen in practice.

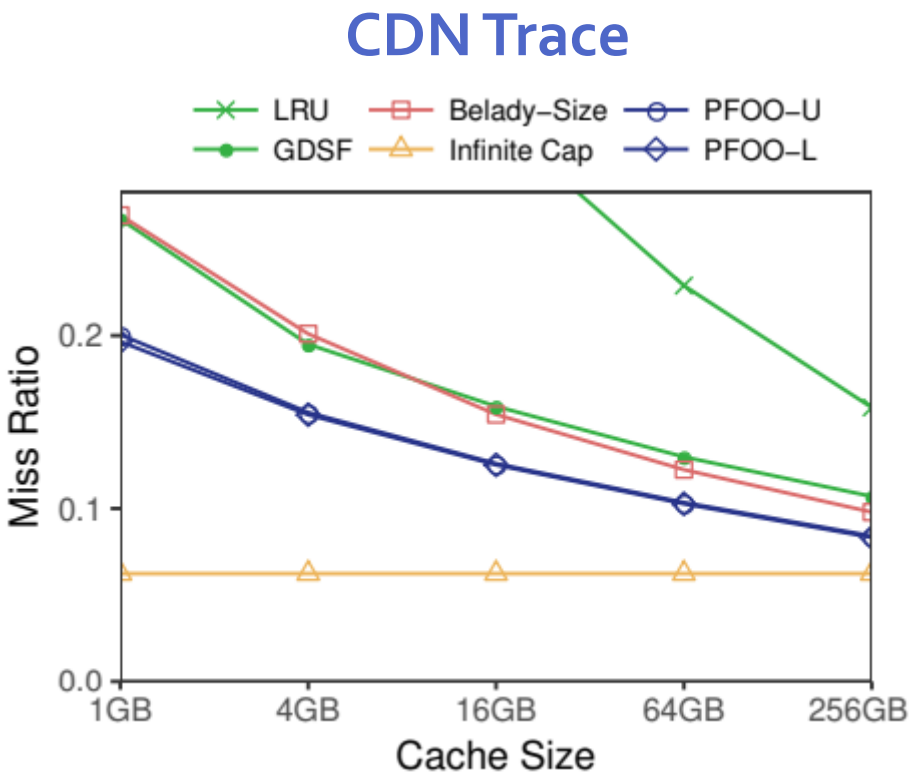
** On storage traces where IRM assumption does not hold.

FOO is accurate across different domains

1. PFOO gives tight bounds on OPT
   (blue lines)

2. There is significant room to improve online policies
   (green vs blue)

3. Prior heuristics are bad OPT bounds ➔ false conclusions
   (green vs red)

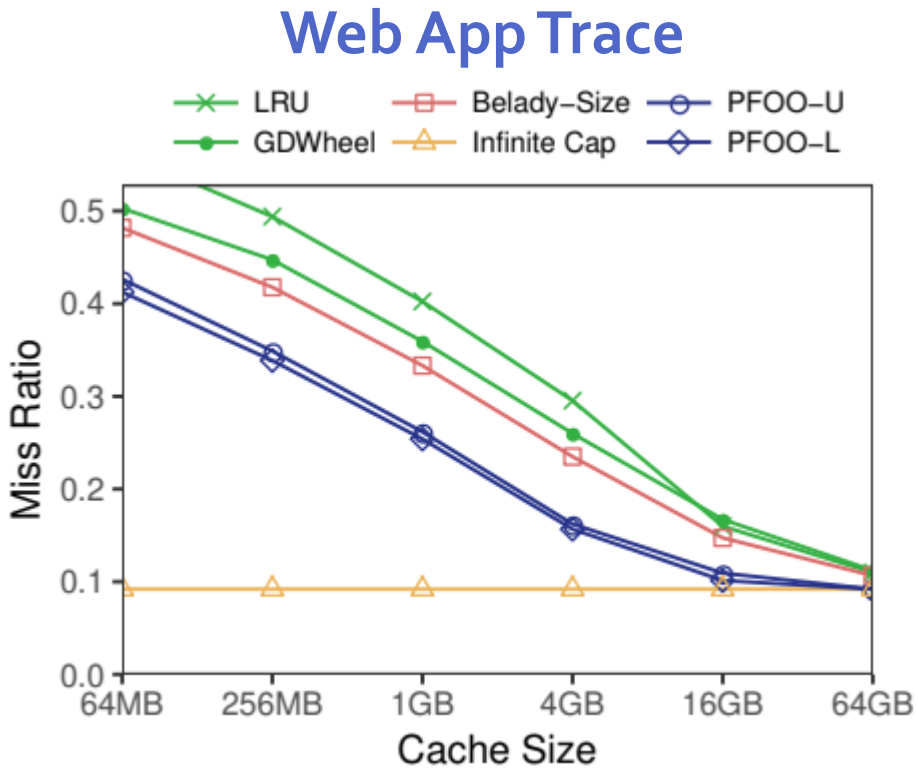


FOO is accurate across different domains

1. PFOO gives tight bounds on OPT
   (blue lines)

2. There is significant room to improve online policies
   (green vs blue)

3. Prior heuristics are bad OPT bounds ➔ false conclusions
   (green vs red)

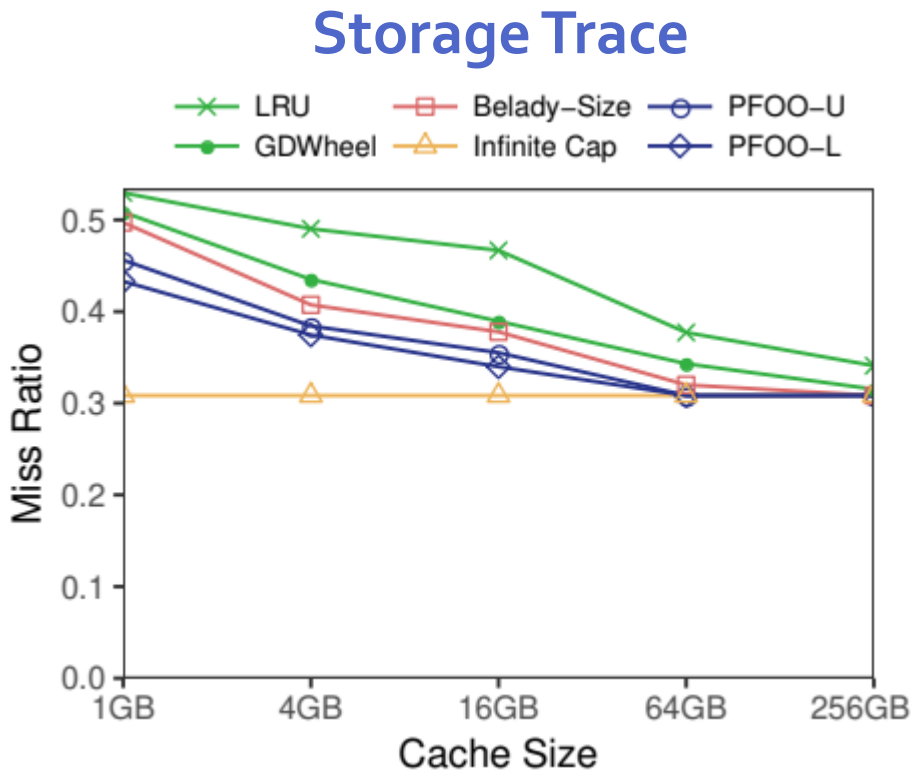


FOO is accurate across different domains

1. PFOO gives tight bounds on OPT
   (blue lines)

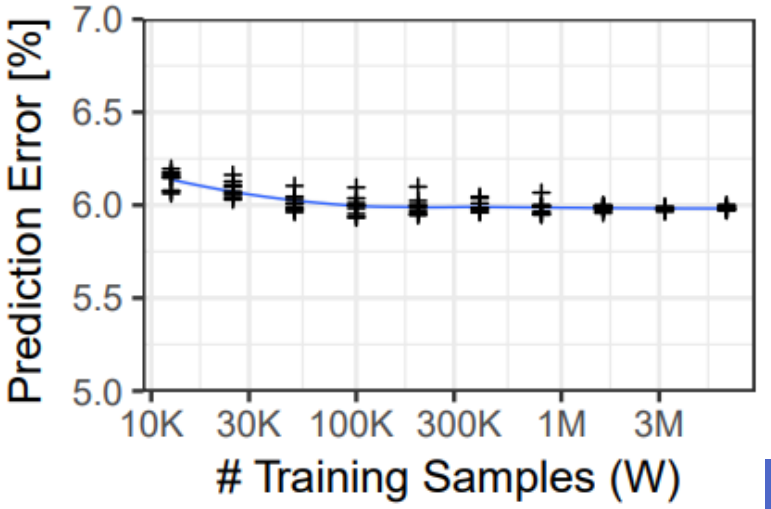
2. There is significant room to improve online policies
   (green vs blue)

3. Prior heuristics are bad OPT bounds ➔ false conclusions
   (green vs red)



OPT is predictable!

- Boosted decision trees achieve 93% accuracy *[Berger, HotNets'19]*
  - Features: Object size, object retrieval cost, inter-request gap
- Classifier must be updated frequently
  - Big drop in accuracy when updating every few M requests
  - ➔ Online training is a must ➔ Training & computing OPT must be fast!
- Fortunately, *we can learn OPT quickly*
  - Converges within a few 10K requests
  - (Comparable to LHD)



Summary

- Caching policies that learn & adapt online are effective & address longstanding problems

- Caching has a ton of structure that should be exploited
  - Object hit density ⇔ Cache hit rate
  - Offline optimal ⇔ Min-cost flow

- There is ample opportunity to improve cache performance further

THANK YOU!