Modeling Cache Performance Beyond LRU

Nathan Beckmann and Daniel Sanchez

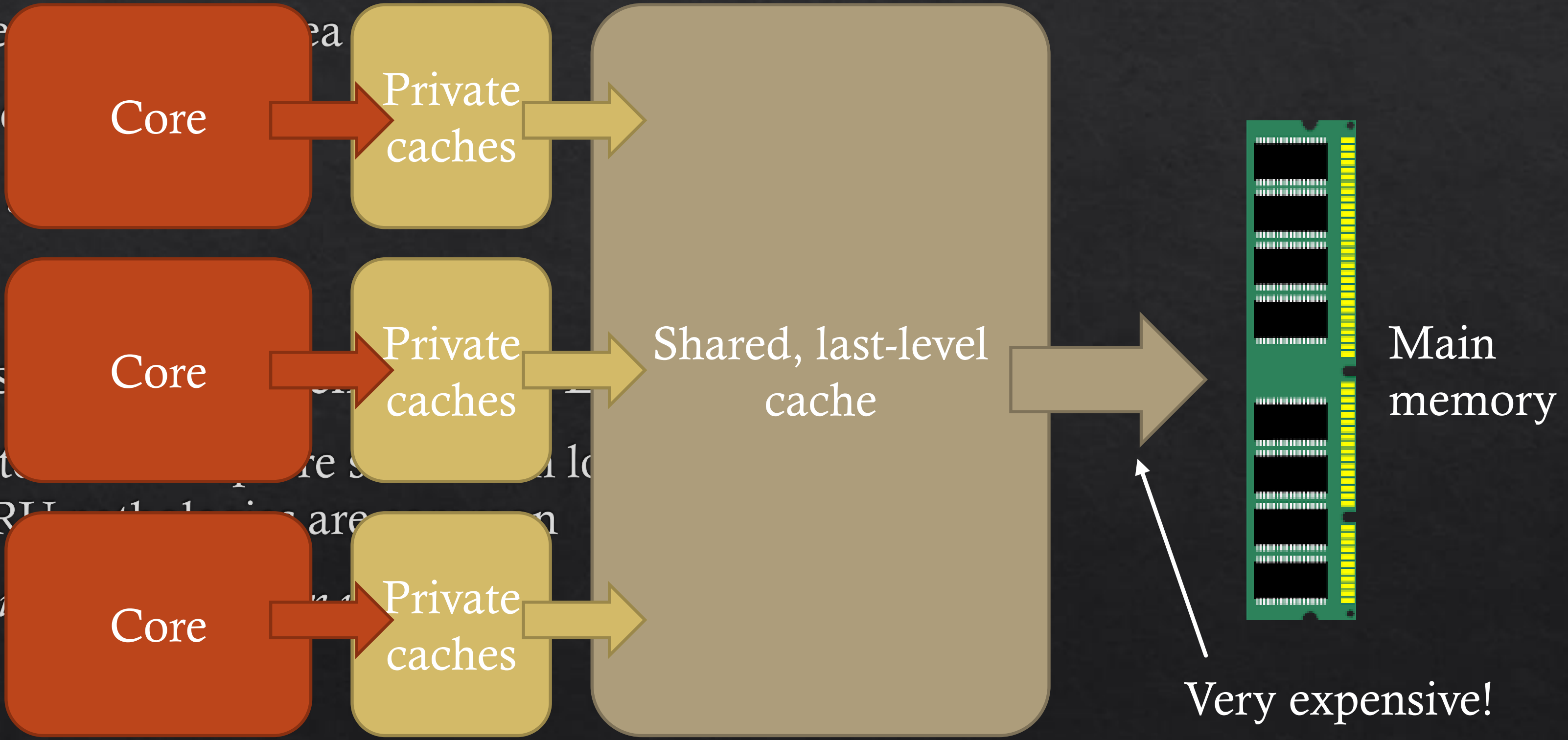
MIT CSAIL – HPCA 2016 – Barcelona, Spain

Motivation

- Predictions of cache performance have many uses:
  - Job scheduling to avoid interference
  - Cache partitioning to improve performance, enhance security, ensure fairness, etc.
- Decades of research on predicting classic replacement policies like LRU or random replacement
- …But not for recent, high-performance replacement policies
  - DRRIP, PDP, IGRD, PRP, etc.
- **We need new modeling techniques that can accurately predict the performance of a broad range of policies**

Background

- Last-level caches (LLCs) are critical to system performance and energy
  - Large ... ea
  - Hash...
  - High ...
- Accesses ...
  - Privat... re s... l lo
    ➔ LRU ... are ... n
  - *LRU ...*



- Abundant recent work on replacement

Background – Replacement policies

- Many different techniques
  - Dynamically protecting cache lines *[DIP, Qureshi ISCA'07][PDP, Duong MICRO'12]*
  - Predicting whether lines will hit *[SBDP, Khan MICRO'10][PRP, Das TACO'15]*
  - Predicting how long until a hit *[DRRIP, Jaleel ISCA'10][IRGD, Takagi ICS'04]*

- Most policies assign value to cache lines which changes over time
  - Value usually increases upon a hit, i.e. promotion
  - Value eventually declines after some time without a hit, i.e. demotion

Background – Cache models

- Prior cache models target LRU, pseudo-LRU, random, etc.

- Many applications require accurate cache predictions
  - Job scheduling *[Mars, MICRO'11][Zhang, EuroSys'13][Delimitrou, ASPLOS'13]*
  - Shared cache partitioning
    - Performance *[Qureshi, MICRO'06][Moreto, OSR'09][Beckmann, PACT'13]*
    - Fairness *[Moreto, OSR'09][Pan, MICRO'13]*
    - Quality-of-service *[Guo, MICRO'07][Kasture, ASPLOS'14][Cook, ISCA'13]*
    - Security, etc. *[Page, Crypto'05][Beckmann, HPCA'15]*

Need cache models for recent, high-performance replacement policies

Our modeling approach

- *Observation 1:* Private caches strip out successive accesses to same cache line
- *Observation 2:* Hashing + high associativity ➔ replacement candidates are well-mixed

Strategy: Model cache replacement as a random process

- *Observation 3:* Many replacement policies rank candidates by *age* (time since last reference)

Strategy: Model replacement policies as arbitrary functions of age

Contributions

- First model for several recent, high-performance replacement policies
  - Based on *absolute reuse distances* – number of accesses between references to address
  - Three related probability equations
  - Easy to model new age-based replacement policies
- Efficient online implementation
- Accurate predictions – mean error of ~3% for LRU, PDP, and IRGD on SPECCPU2006
- *Limitations:* Currently does not model non-age-based policies like DRRIP

Model outline

- Assumptions
- Explain model for LRU
- Generalize model to other policies

To limit math, this talk will use pictures to give intuition and then quickly show corresponding equations – *see paper for detailed derivations*

Model assumptions

- Assume high associativity – i.e., replacement candidates are selected at random
  - Direct model of skew-associative caches, also works for hashed set-associative caches
- Assume reuse distances are independent and identically distributed
  - *Reuse distance* is the number of accesses between references to the same address
  - Intuition: Private caches filter out successive accesses to same address, removing locality at LLC
- **These assumptions are only approximately satisfied in practice, but the model is surprisingly robust to deviations from them**

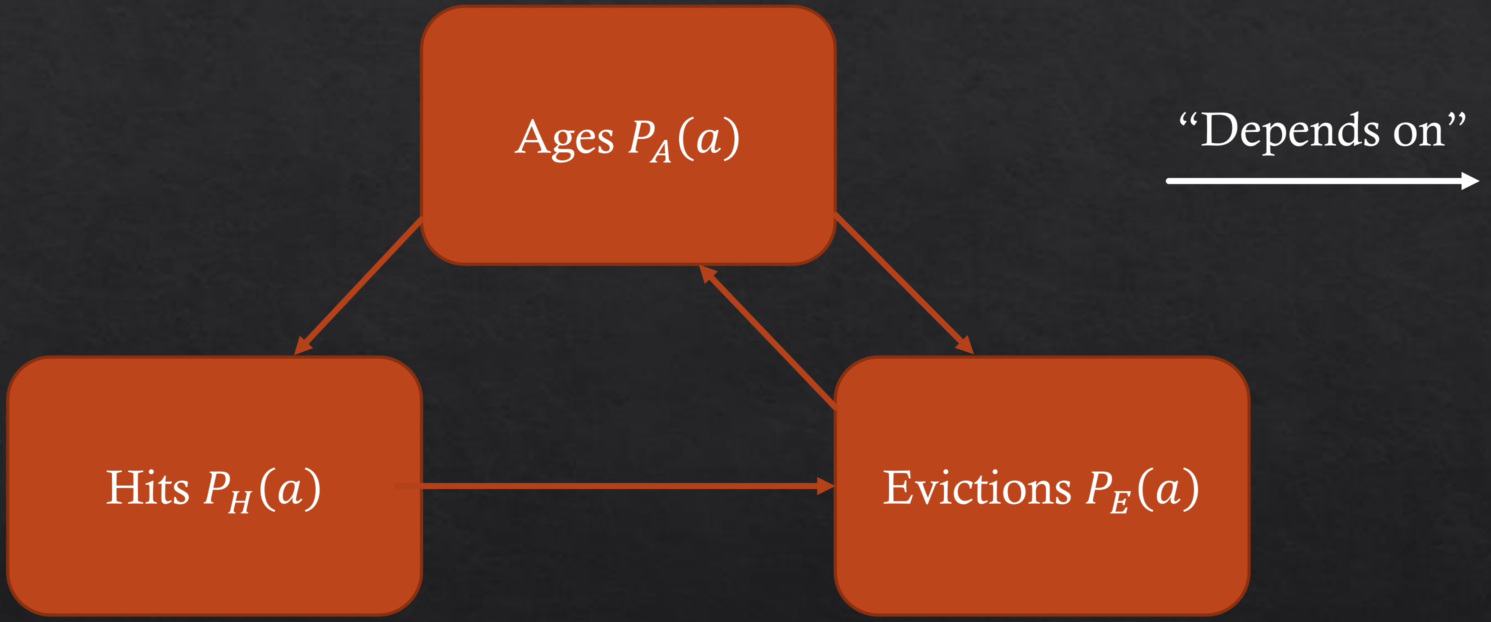
Example and definitions

Requests:	A	A	B	C	B	D	B	C…
	3	*1*	*1*	*2*	*3*	*4*	*1*	*2*
	2	*3*	*4*	*1*	*2*	*1*	*2*	*1*
	1	*2*	*3*	*4*	*1*	*2*	*3*	*4*

- *Age* is the number of accesses since last reference

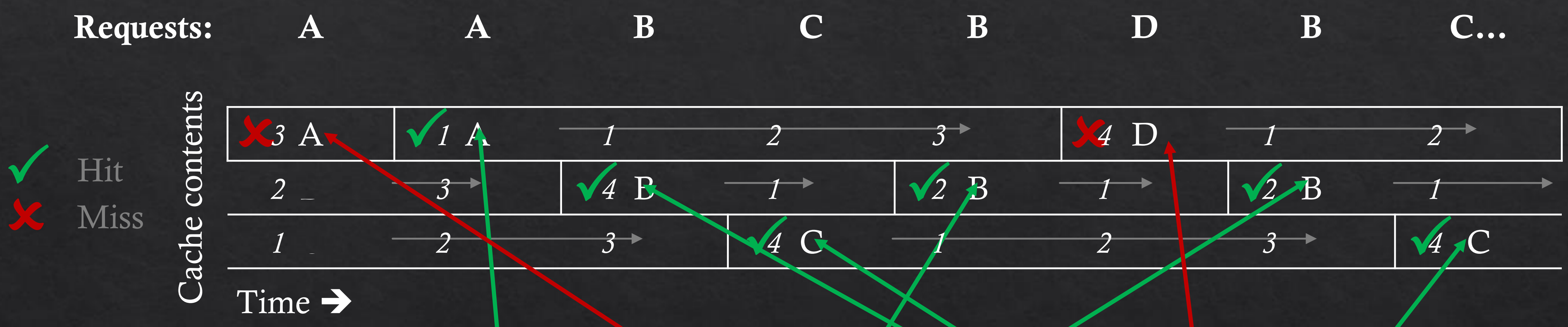
Model overview

- Three interdependent probability distributions



- Cache hit rate is the sum of the hit distribution, i.e. Hit rate $= \sum_{a=1}^{\infty} P_H(a)$

Example and definitions



- *Age* is the number of accesses since last reference

Hit rate = 3/4

	1	2	3	4
Hits	1/8	2/8	–	3/8
Evictions	–	–	1/8	1/8
Ages	8/24	7/24	5/24	4/24

Together sum to 1

Age distribution

- $P_A(a)$ – How many lines have age a?

- *Insight:* Lines at age a must hit or be evicted at age $\geq a$
- ➔ $P_A(a)$ is proportional to number of hits and evictions at higher ages

	1	2	3	4	5
Hits	1	2	–	3	–
Evictions	–	–	1	1	–
Ages	8	7	5	4	0

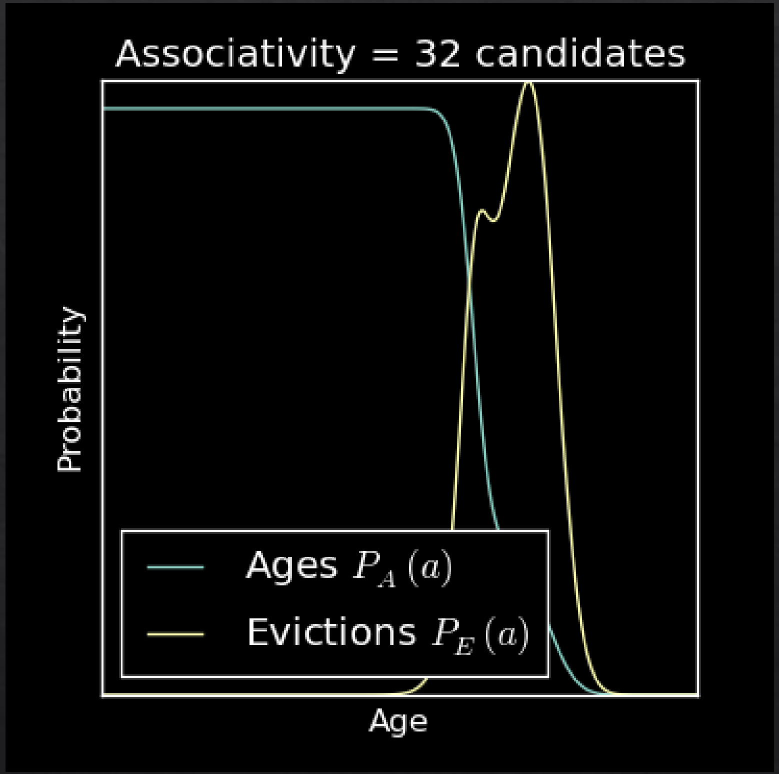
$$P_A(a) = \frac{1}{\text{Cache size}} \times (P[H \geq a] + P[E \geq a])$$

Eviction distribution for LRU

- $P_E(a)$ – How many lines are evicted at age a?

- *Insight:* LRU evicts the oldest (maximum age) candidate
- ➔ Given W randomly-chosen candidates, victim's age is distributed as maximum of W draws from $P_A(a)$

- $P_E(a) = \text{Miss rate} \times \text{Max. age of } W \text{ ages}$
- $\qquad = P[\text{miss}] \times (P[A < a + 1]^W - P[A < a]^W)$

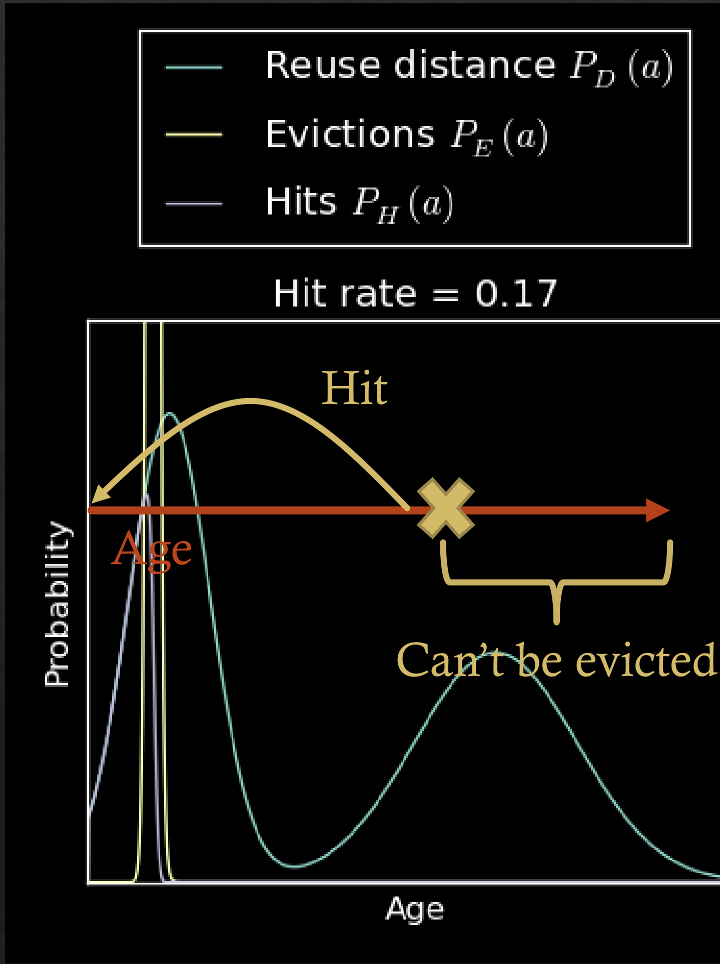


Hit distribution

- $P_H(a)$ – How many hits occur at age a?

- *Insight:* Hits at age a imply (absolute) reuse distance of a
  - Every reuse distance a will hit at age a unless first evicted
- ➔ $P_H(a)$ = Reuse distances at a − Evictions before a
  - Sadly, eviction age and reuse distance aren't independent!

- How do evictions change hit probability?
- *Insight:* Replacement policy doesn't know reuse distance!
- ➔ Evictions at a only imply that reuse distance > a, and lower the probability of all later hits



Model summary for LRU

- Age distribution – cache size
  - $P_A(a) = \frac{1}{\text{Cache size}} \times (P[E \geq a] + P[H \geq a])$
- Eviction distribution – replacement policy & associativity
  - $P_E(a) = P[\text{miss}] \times (P[A < a+1]^W - P[A < a]^W)$
- Hit distribution – access pattern via reuse distance distribution $P_D(a)$
  - $P_H(a) = P_D(a) \times \left(1 - \sum_{x=1}^{a-1} \frac{P_E(x)}{P[D>x]}\right)$

Generalizing to other policies

- How to model different replacement policies?

- We model policies as *ranking functions* of candidates' ages $R(a)$
  - By convention, higher rank ➔ likelier to be evicted

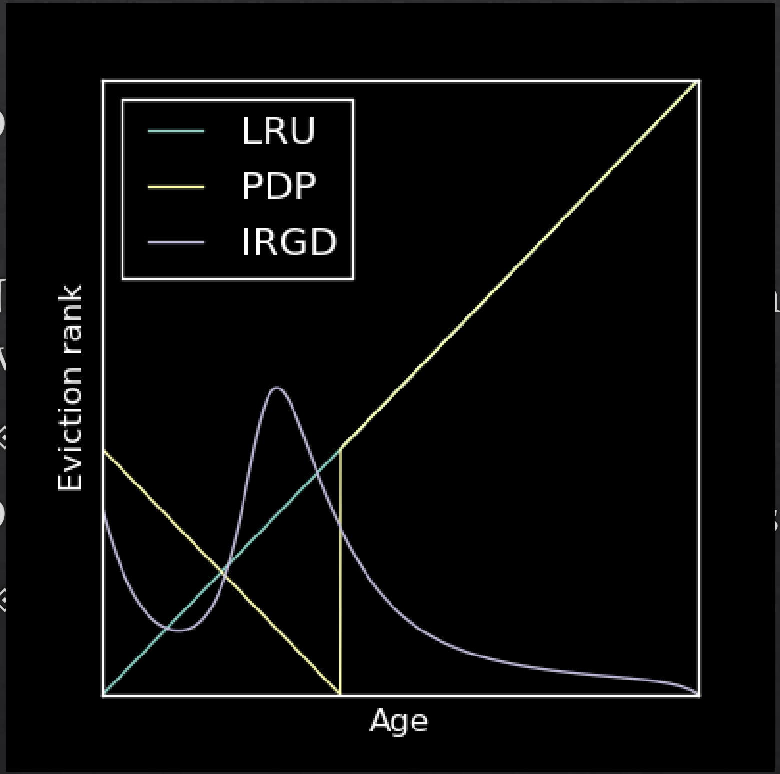
- Replacement model:
  - 1. Given candidates' ages $a_1, a_2 \ \ldots a_W$
  - 2. Rank candidates as $R(a_1), R(a_2) \ldots R(a_W)$
  - 3. Evict candidate with highest $R(a_i)$

Ranking functions

Pros

- Simple + analytically tractable model
- Works for many replacement policies
  - LRU: $R(a) = a$
  - PDP: protect lines until age d_p
  - IRGD: statistical cost function
  - PRP: conditional hit probability

- D
- N ange
  ov
- D



Generalized eviction distribution

- Age and hit distributions do not change!

- LRU evicted the oldest candidate

- *Substitute:* "maximum age" (for LRU) ➔ "maximum rank" (in general)
  - 1. Compute distribution of ranks in cache using $R(a)$ and age distribution
  - 2. Find distribution of maximum rank as W draws from this distribution

- Some corner cases to avoid double counting, etc.

Model summary for arbitrary ranking functions

- Age distribution – cache size
  - $P_A(a) = \frac{1}{\text{Cache size}} \times (P[E \geq a] + P[H \geq a])$

Solve through iteration!
(see paper)

- Eviction distribution – replacement policy & associativity
  - $P_E(a) = P[\text{miss}] \times \cancel{(P[A < a+1]^W - P[A < a]^W)}\ \frac{P_A(a)}{P_{\text{rank}}(R(a))} \times \frac{P[\text{rank} < R(a) + \Delta r]^W - P[\text{rank} < R(a)]^W}{\Delta r}$

- Hit distribution – access pattern via reuse distance distribution $P_D(a)$
  - $P_H(a) = P_D(a) \times \left(1 - \sum_{x=1}^{a-1} \frac{P_E(x)}{P[D>x]}\right)$

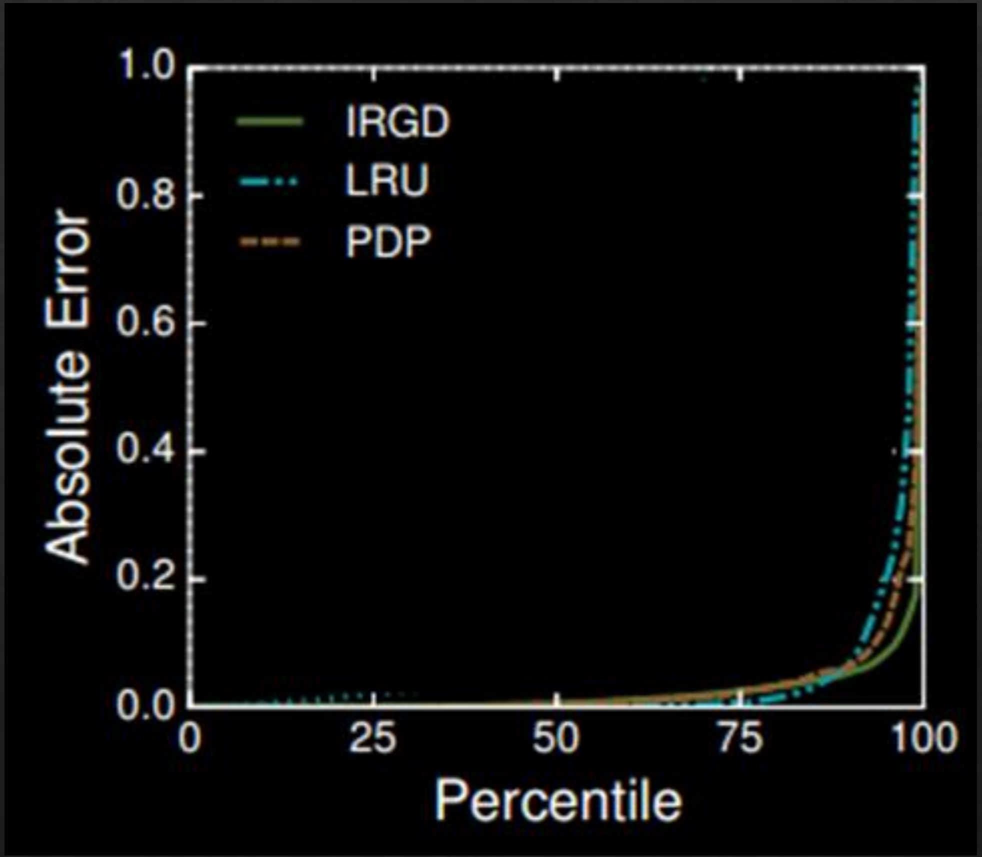
Validation – Simulation methodology

- Run SPECCPU2006 for 20 B instructions using zsim *[Sanchez, ISCA'13]*
- 16-way, set-associative hashed caches from 128 KB – 128 MB
  - LRU, PDP, and IRGD replacement
- Model solved <u>every 100 ms</u> using sample reuse distance distributions
  - Small monitor gathers LLC reuse distance distribution online
  - Compare against simulated cache hit rate
- Demanding workload!
  - Sampling error
  - Reuse distance distributions not in equilibrium

Validation – SPECCPU2006 results

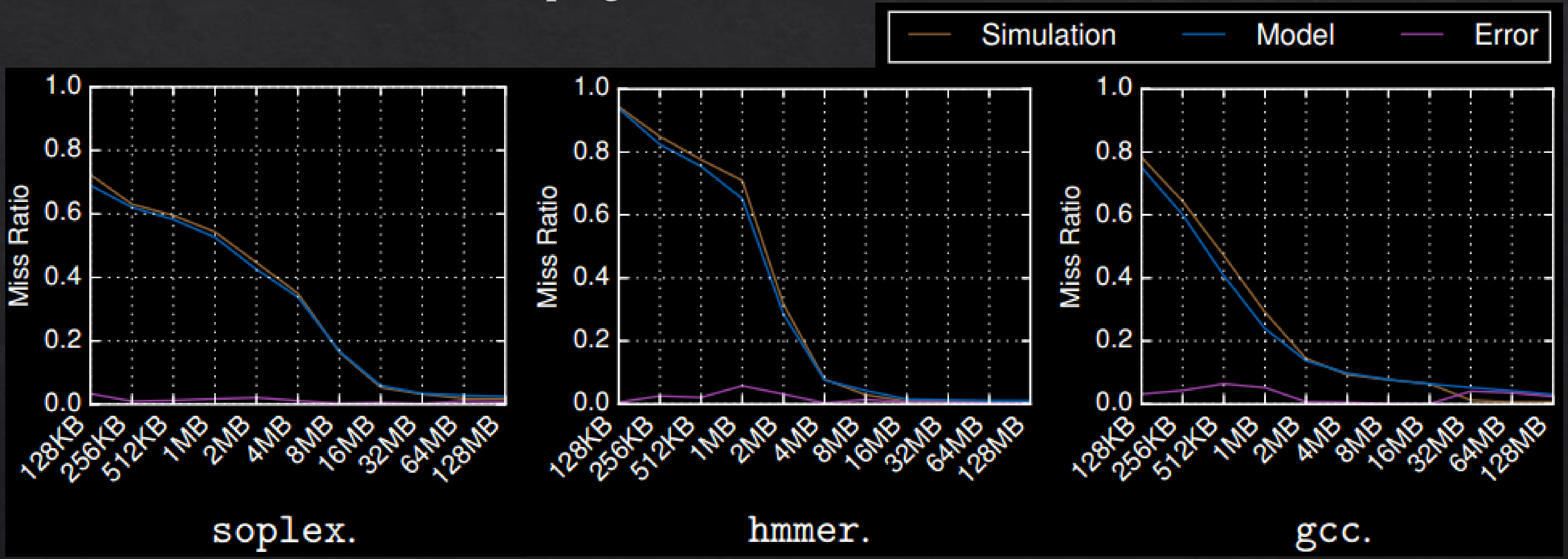
- Low error across 400,000 model solutions
  - 29 applications
  - 11 cache sizes, 128 KB – 128 MB
  - 100 ms interval

- E.g., for IRGD
  - Median error of 0.1%
  - Mean error of 1.9%
  - 90^{th} pctl error of 5.5%



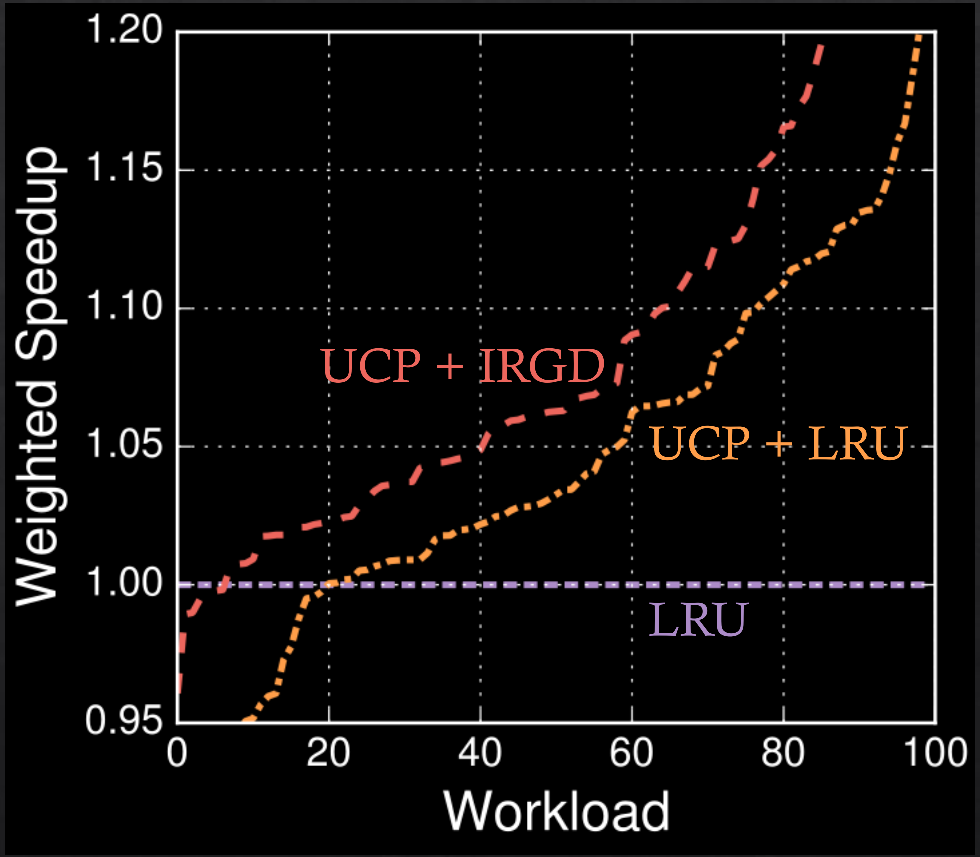
Validation – SPECCPU2006 results

- Even more accurate across full program execution



Case study – Cache partitioning

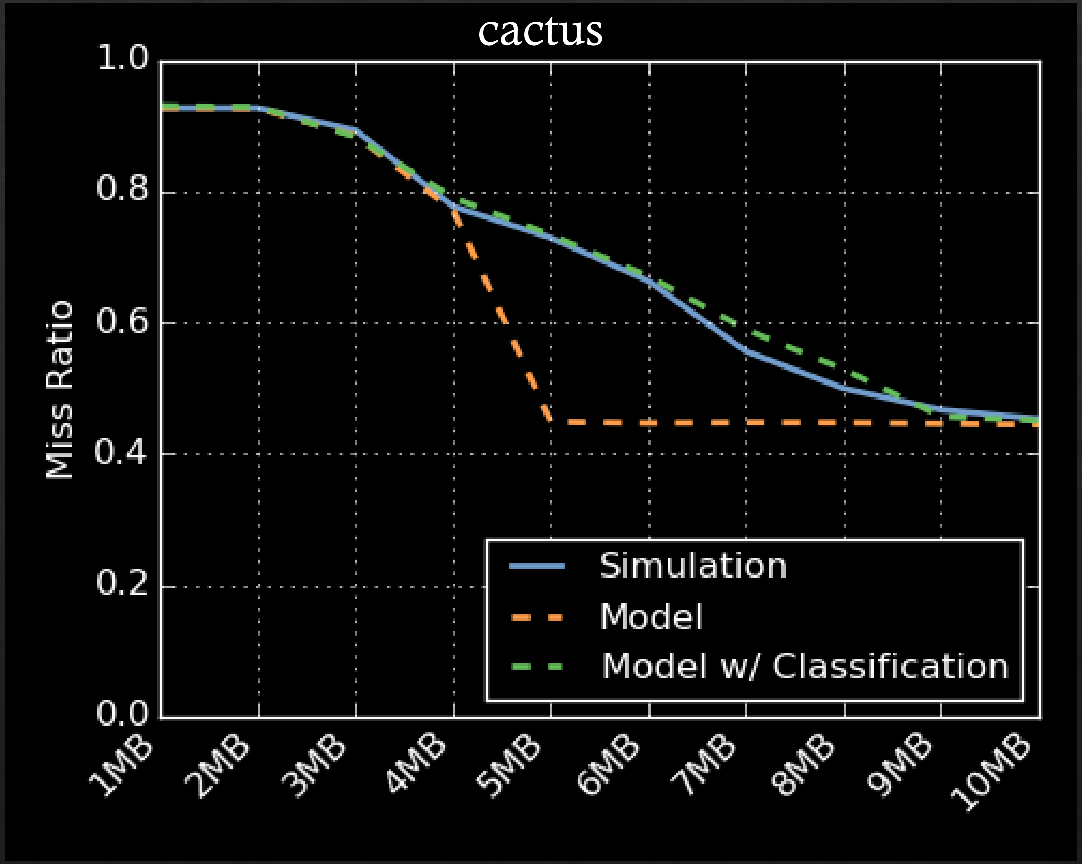
- Cache partitioning with IRGD improves performance significantly
  - *No prior scheme can efficiently predict IRGD!*
  - 4 core system, 4 random apps
  - Utility-based Cache Partitioning (UCP)
    - [Qureshi, MICRO’06]
  - Gmean +10% speedup, up to +44%
    - vs for LRU, gmean +4.5%



Extensions – Classification

[Tech report]

- For some apps, our assumptions are too strong
- *Specifically*: Reuse distances aren't iid
- This is largely addressed by breaking accesses into two classes:
  - Those likely to hit (short reuse)
  - Those unlikely to hit (long reuse)
  - Boundary chosen adaptively



Extensions – Cache calculus *[CAL'16]*

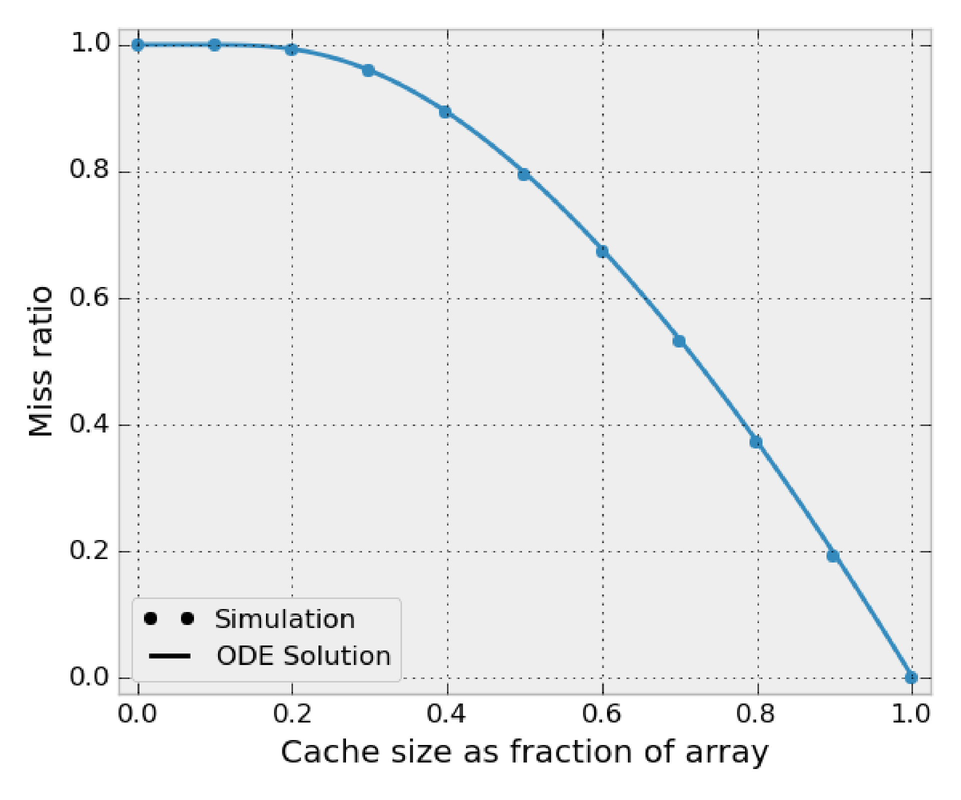
- We can generalize this model into system of ordin

$$H'' = \frac{D''}{D'} H' - \frac{D'}{1-D} E' \qquad \text{and}$$

- Solve ODEs for *closed-form solutions* on particular a

- *Example:* Scanning an array with random replacen

$$\text{miss rate} = 1 - S \times \text{ProductLog}\left(-e^{-1/S}/S\right)$$



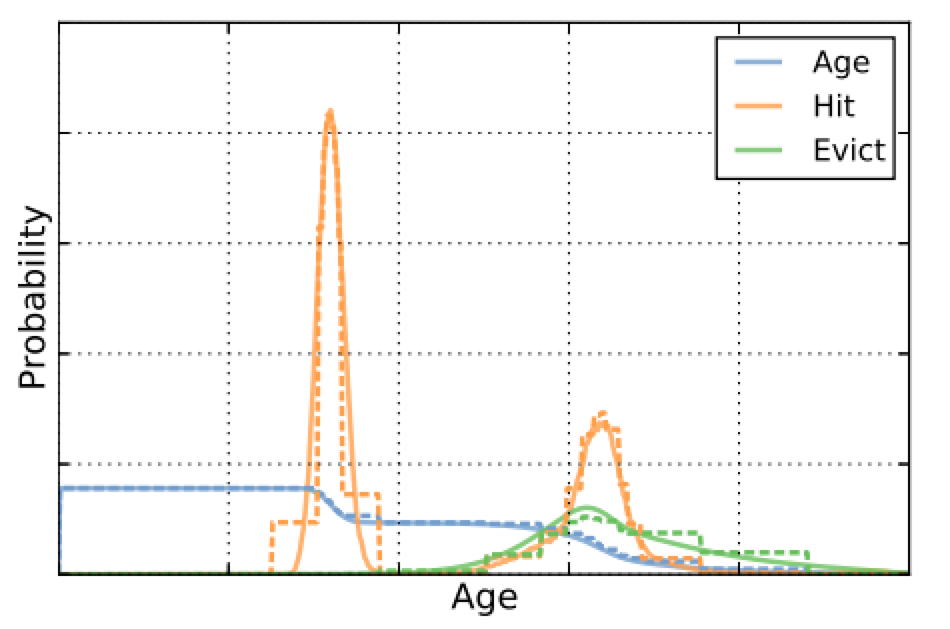
Conclusion

- Accurate predictions of cache behavior are very useful
- Prior models do not support recent high-performance policies
- This work makes a first step towards modeling arbitrary replacement policies
- Efficient implementation and accurate predictions

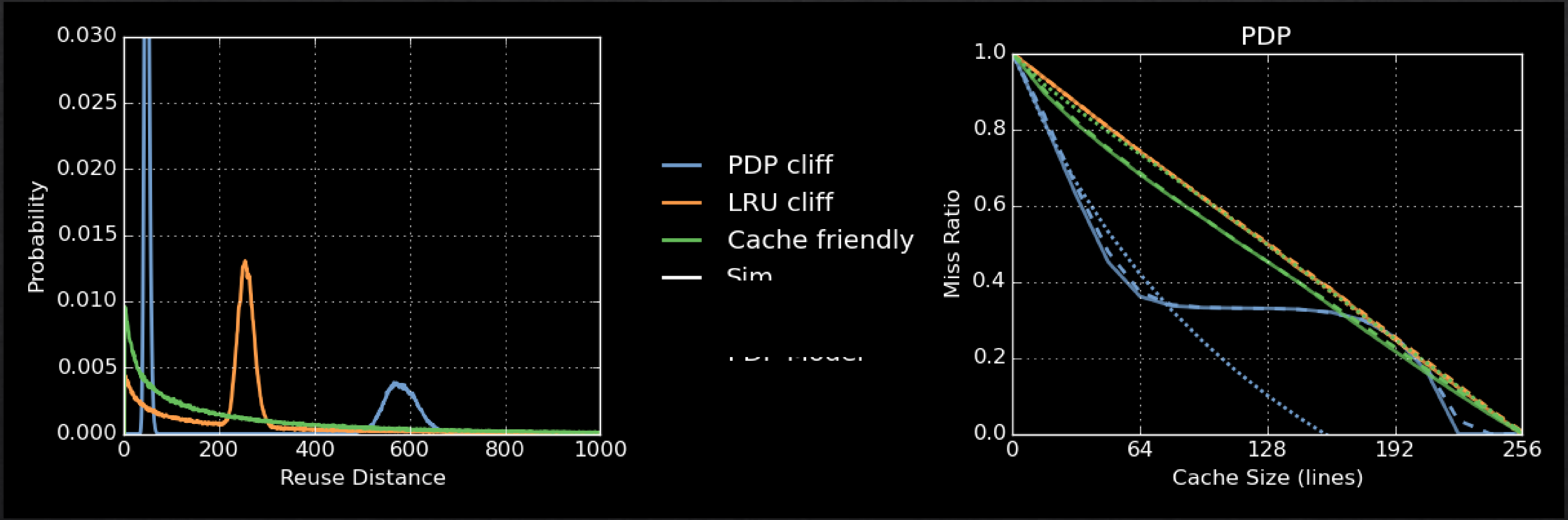
Questions?

Model solution

- Iterate to a fixed point when hit rate stabilizes
  - Typically <50 iterations
  - Each iteration is linear on size of distributions
- Efficient through *coarsened ages*, i.e. age regions
  - Increases performance tremendously – e.g., 64 ×
- Small C++ runtime is publicly available

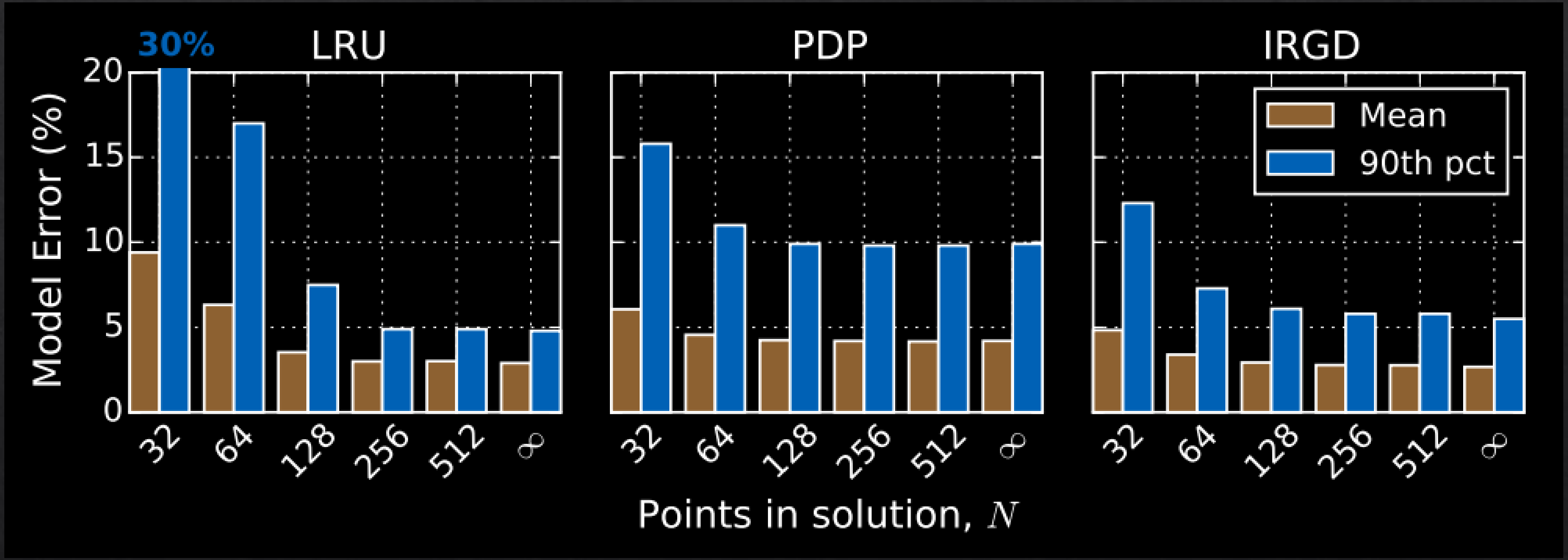


Validation – Synthetic



Validation – SPECCPU2006 results

- Modest error from coarsening ages until ~64 regions *(64× compression)*



Future work

- Model other policies
  - DRRIP – incorporate classification to model promotions + model aging mechanism
  - Generalize from one rank to a distribution of possible ranks?
- New model applications
  - Much more information than just hit rate!
  - How can we use hit, eviction distribution to improve cache performance?