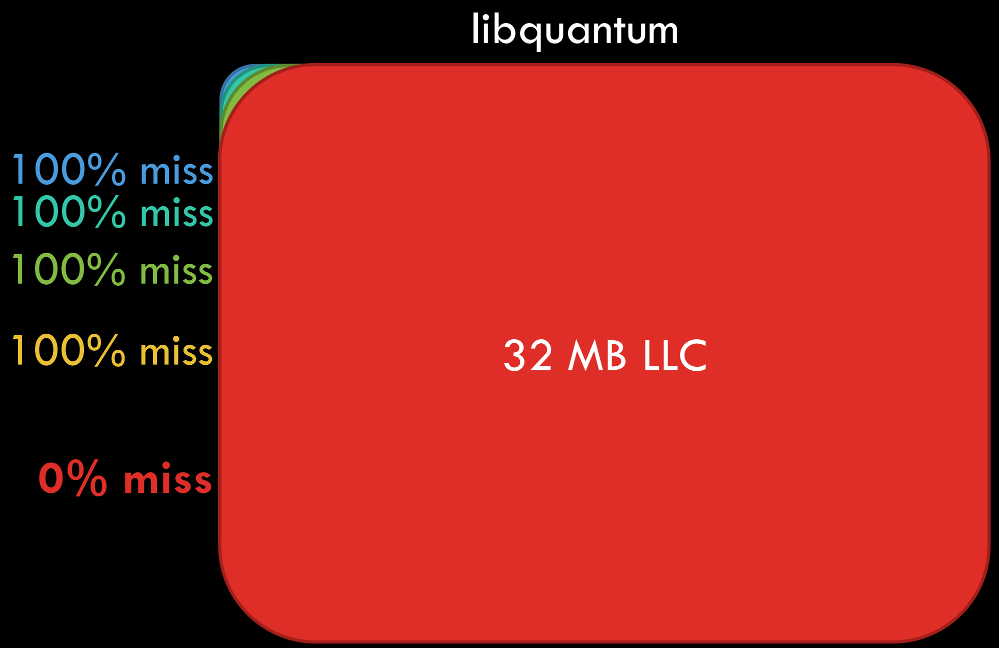
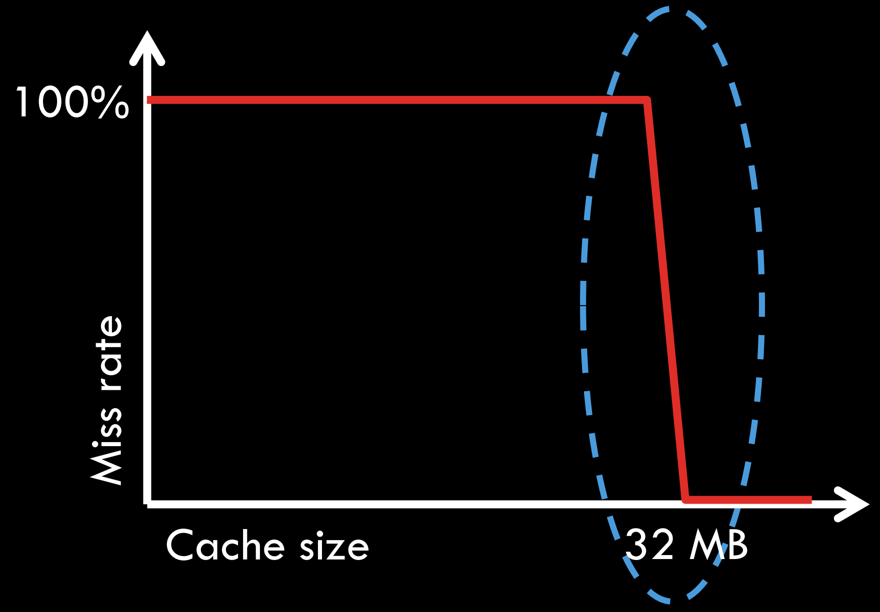
TALUS: A SIMPLE WAY TO REMOVE PERFORMANCE CLIFFS IN CACHES

Nathan Beckmann
Daniel Sanchez
MIT
CSAIL

CACHES HAVE PERFORMANCE CLIFFS





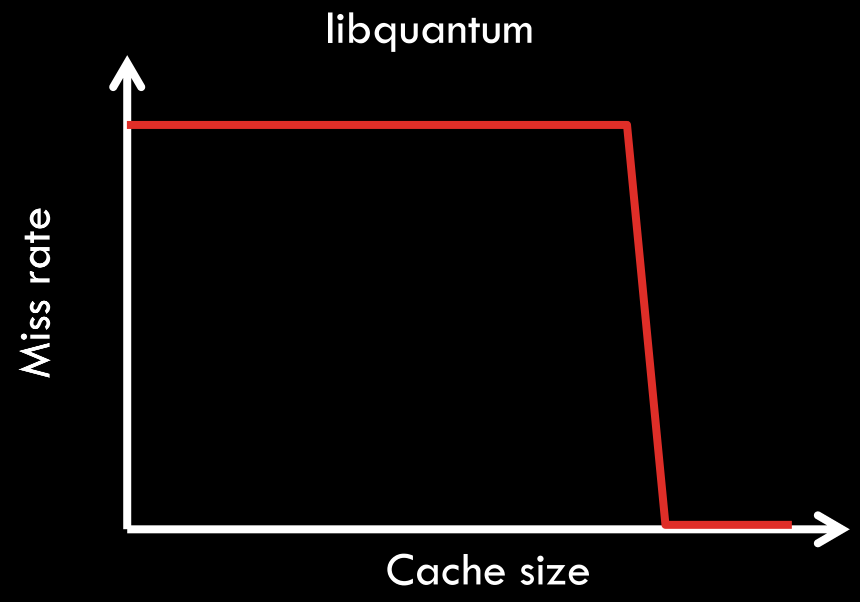
CLIFFS ARE A PROBLEM

Cliffs are wasteful

Cliffs cause annoying performance bugs

Cliffs complicate cache partitioning

- *NP-hard problem*



PRIOR WORK: HIGH-PERFORMANCE REPLACEMENT VS. CACHE PARTITIONING

Individual apps: High-performance replacement

- E.g., RRIP [ISCA'10]

Shared caches: Cache partitioning

- E.g., UCP [MICRO'06]

For shared ... nt

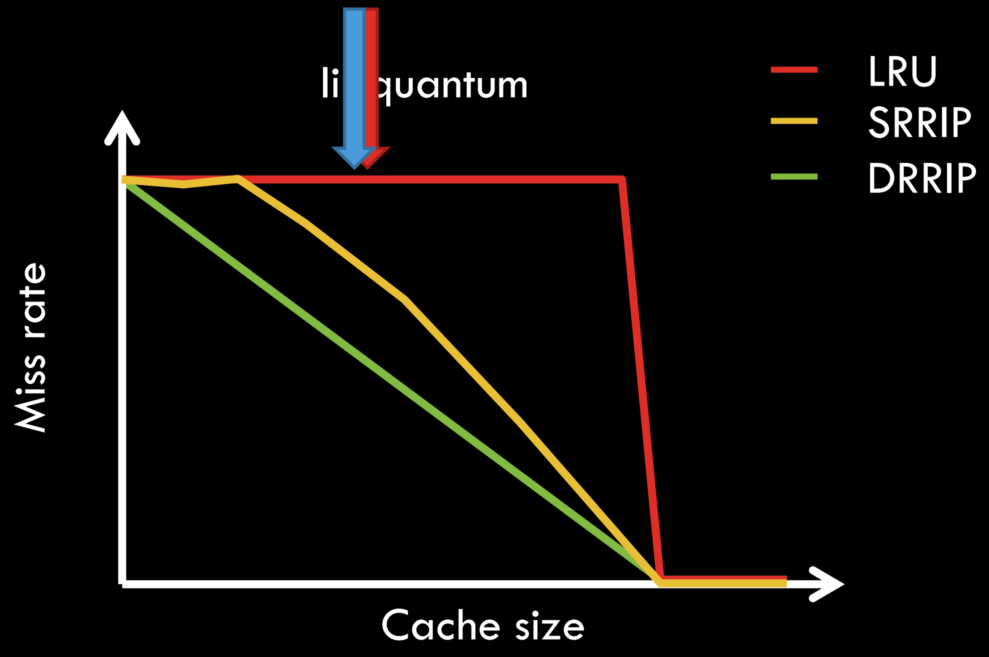
- Both in perf...

Partitioning ... ance replacemen...

Can partitic...

32 MB
libquantum #1

0% miss



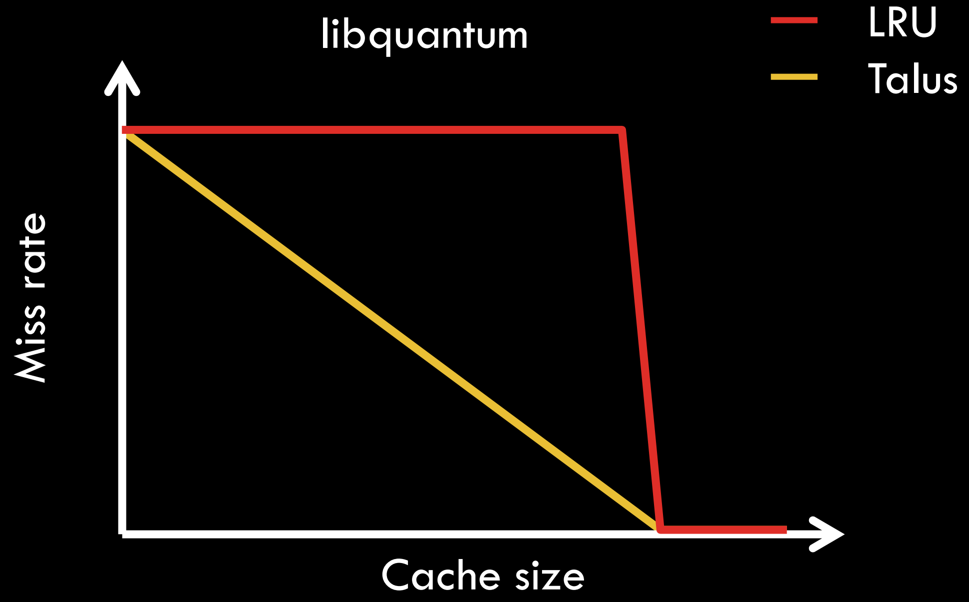
IN THIS TALK WE WILL…

Give a simple technique to eliminate cliffs (Talus)
- Talus partitions *within* a single access stream

Prove it works under simple assumptions
- *Agnostic to app or replacement policy*

No cliffs ➔ Simpler cache partitioning

Talus combines the benefits of high-performance replacement and partitioning



ROAD MAP

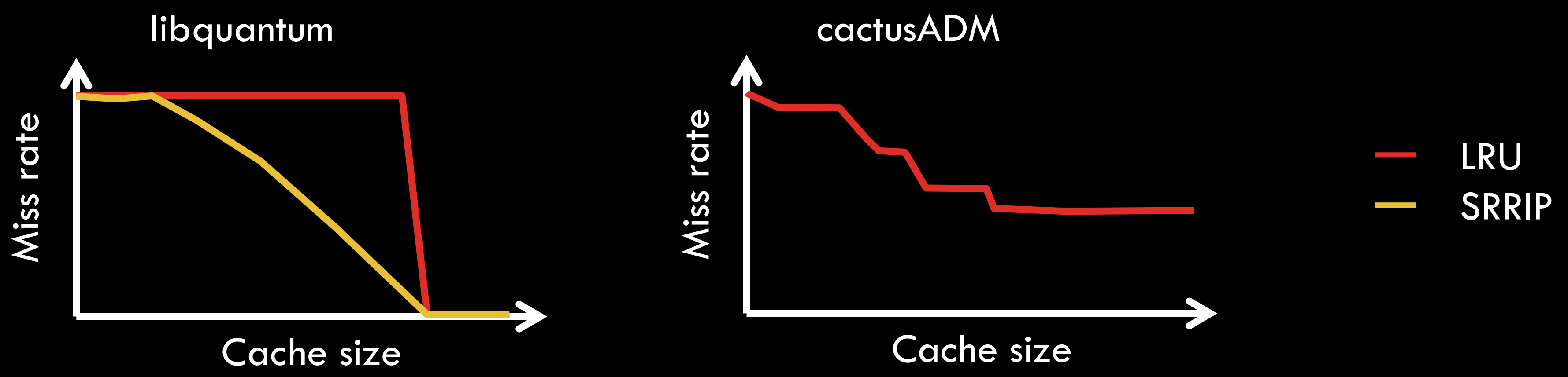
Talus example

Theory

Implementation

Evaluation

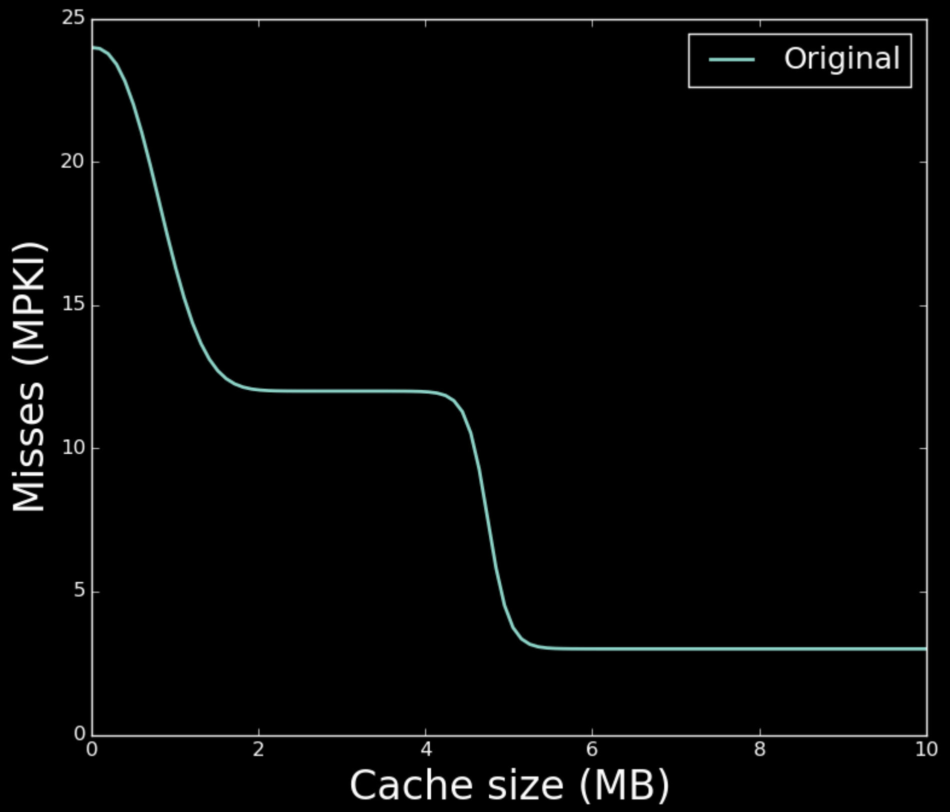
TALUS USES MISS CURVES



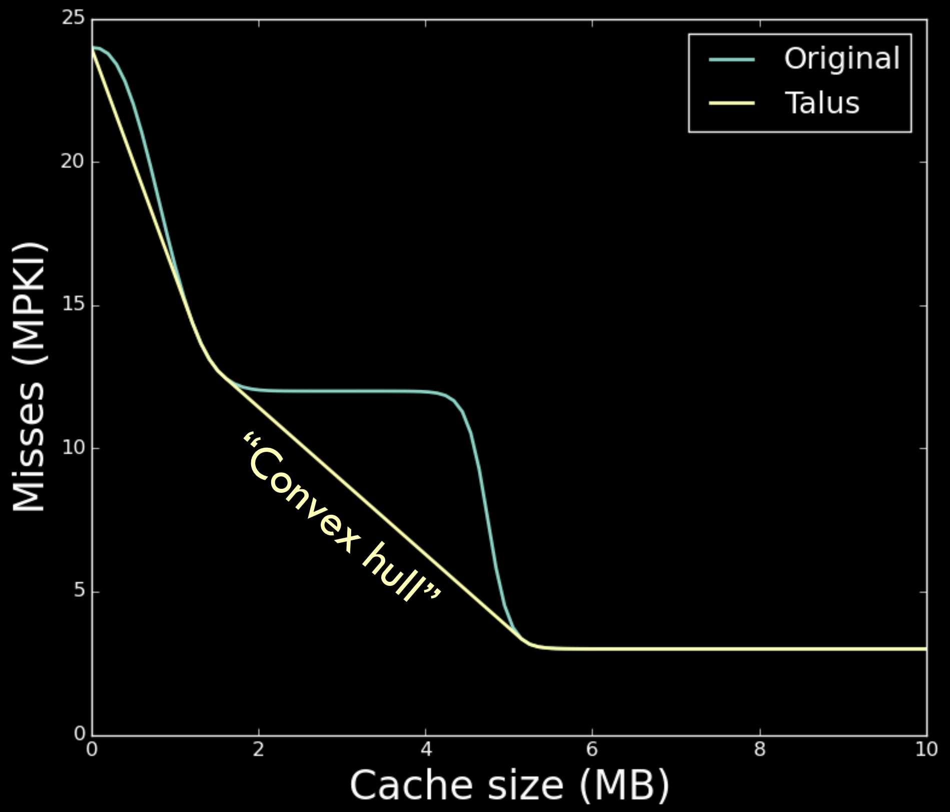
Cliffs occur under a variety of access pattern and replacement policies

Talus works on miss curves only;
Talus is agnostic to app and replacement policy

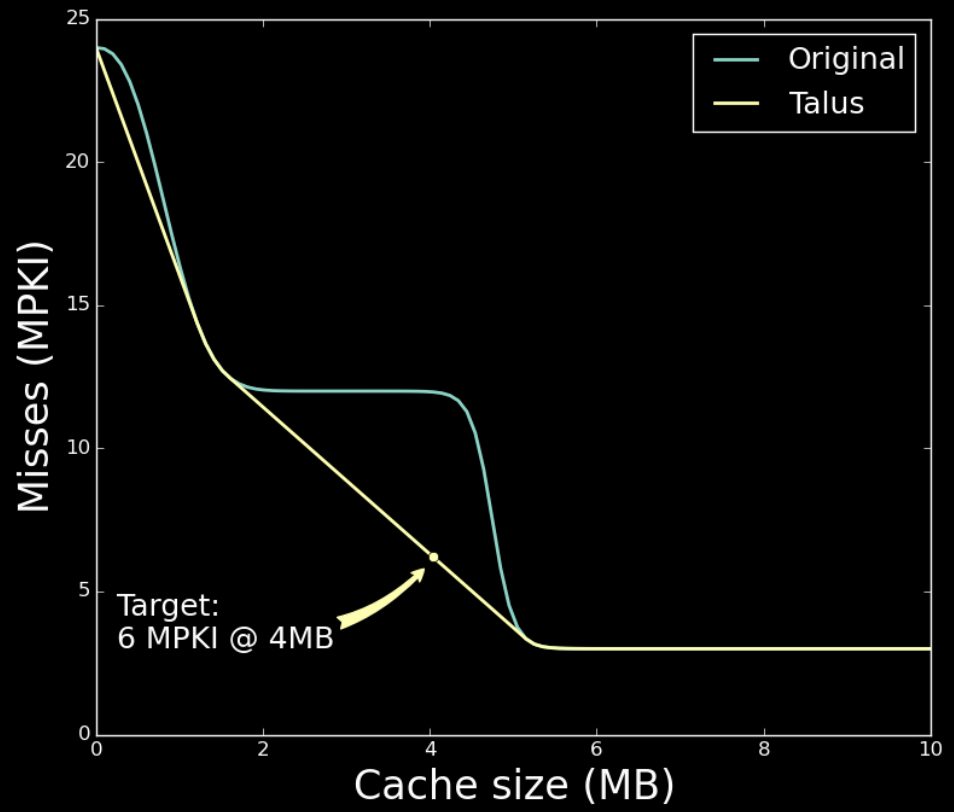
TALUS EXAMPLE



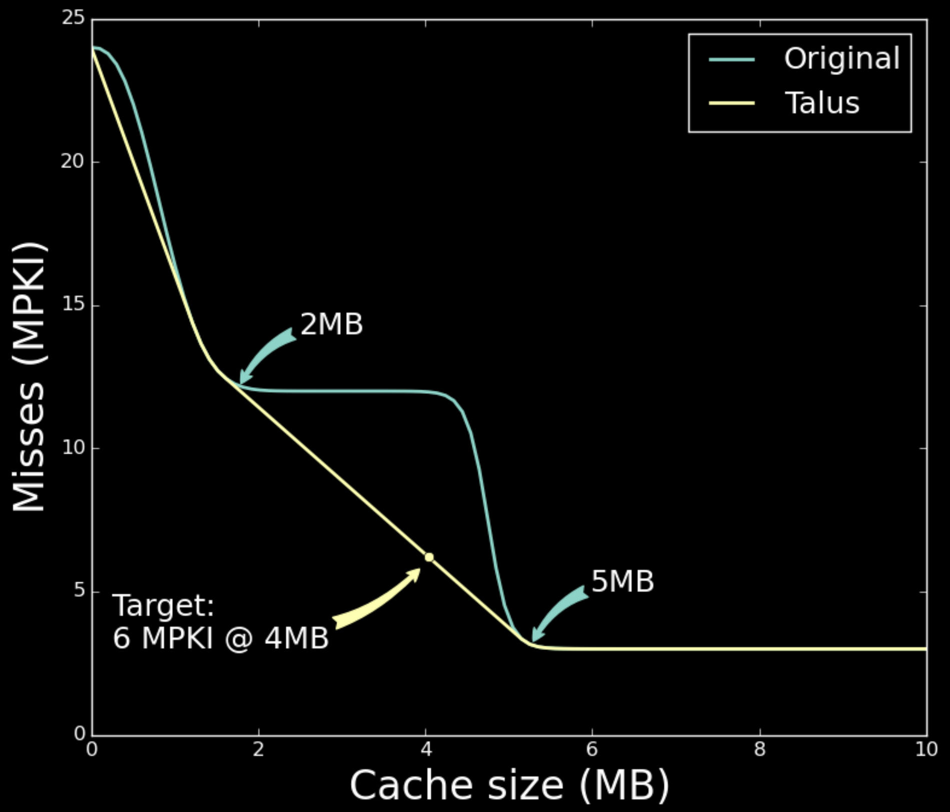
TALUS EXAMPLE



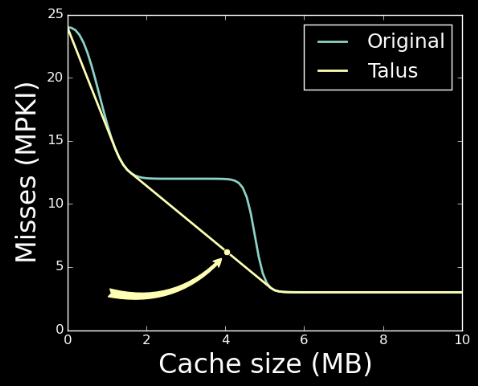
TALUS EXAMPLE



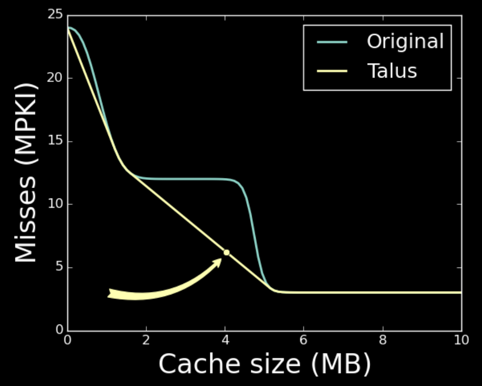
TALUS EXAMPLE



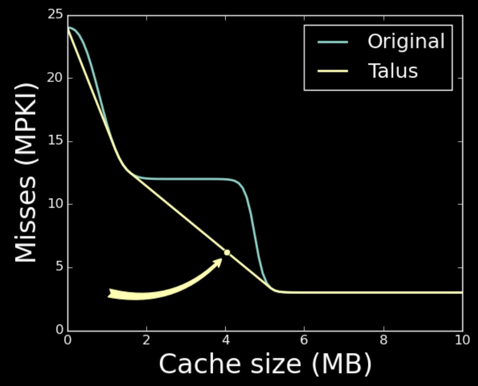
(HYPOTHETICAL) BASELINE CACHE AT 2 MB

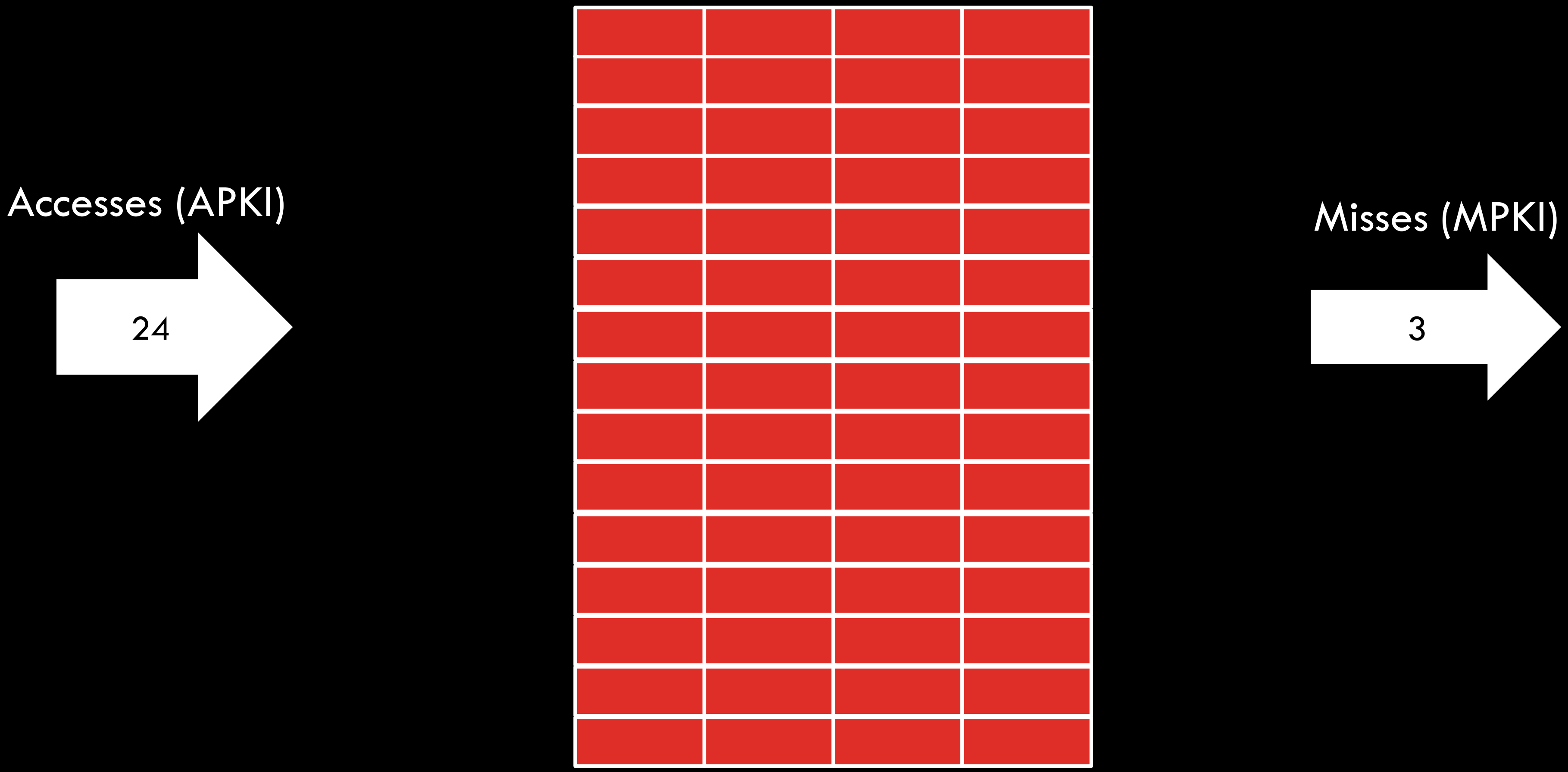


(HYPOTHETICAL) BASELINE CACHE AT 2 MB

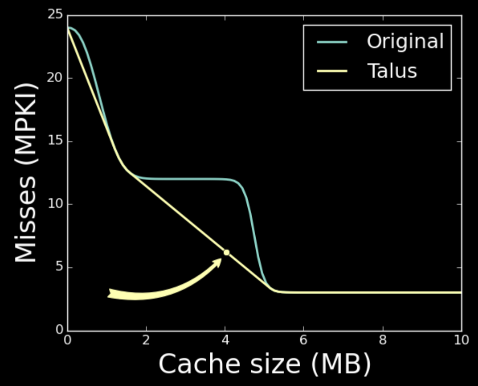


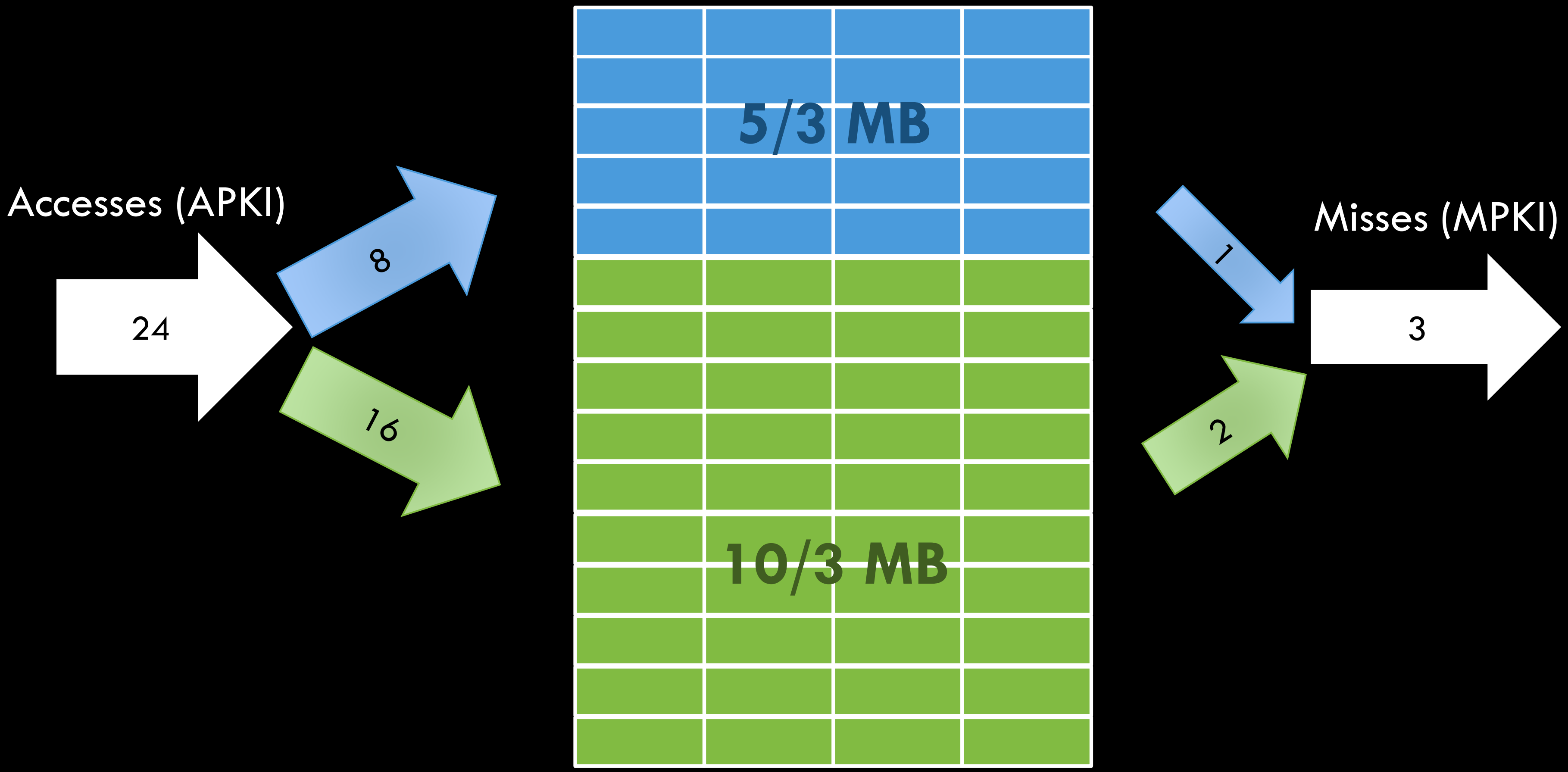
(HYPOTHETICAL) BASELINE CACHE AT 5 MB



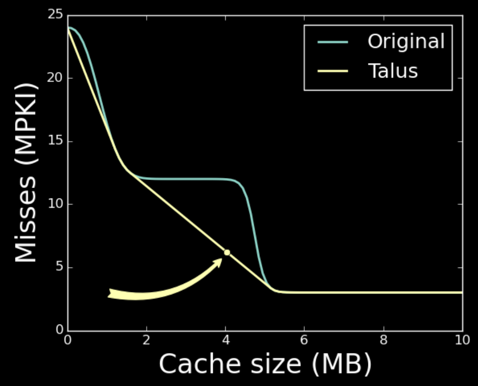


(HYPOTHETICAL) BASELINE CACHE AT 5 MB

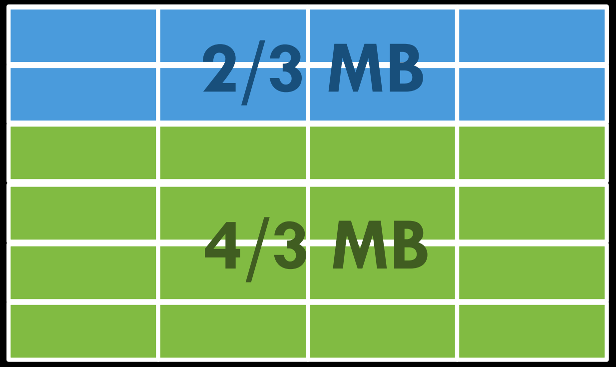


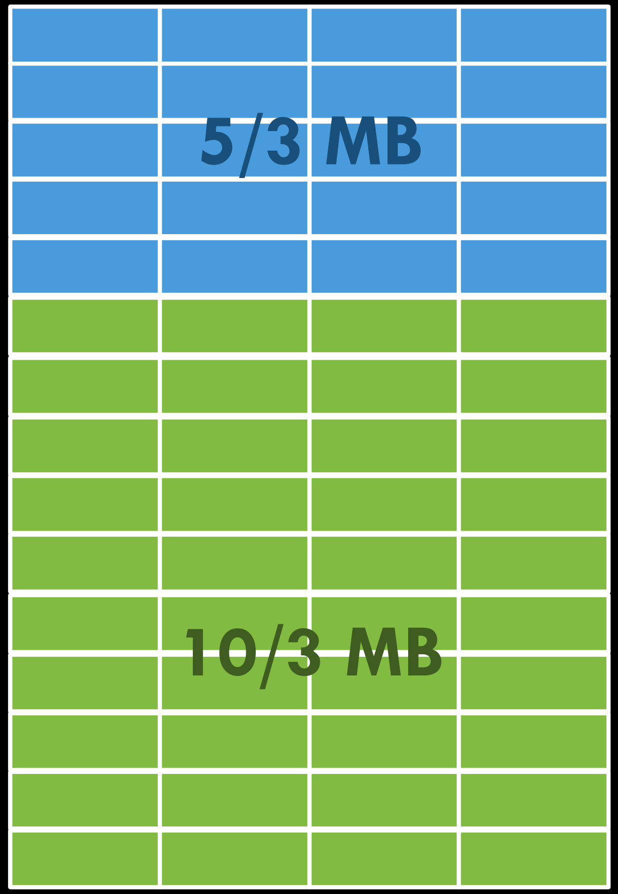


TALUS AT 4 MB

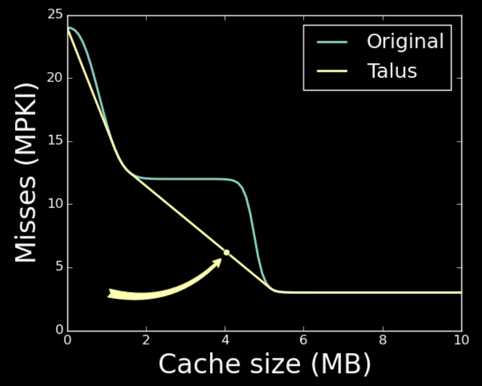


Combine hypothetical baseline 2 MB & 5 MB

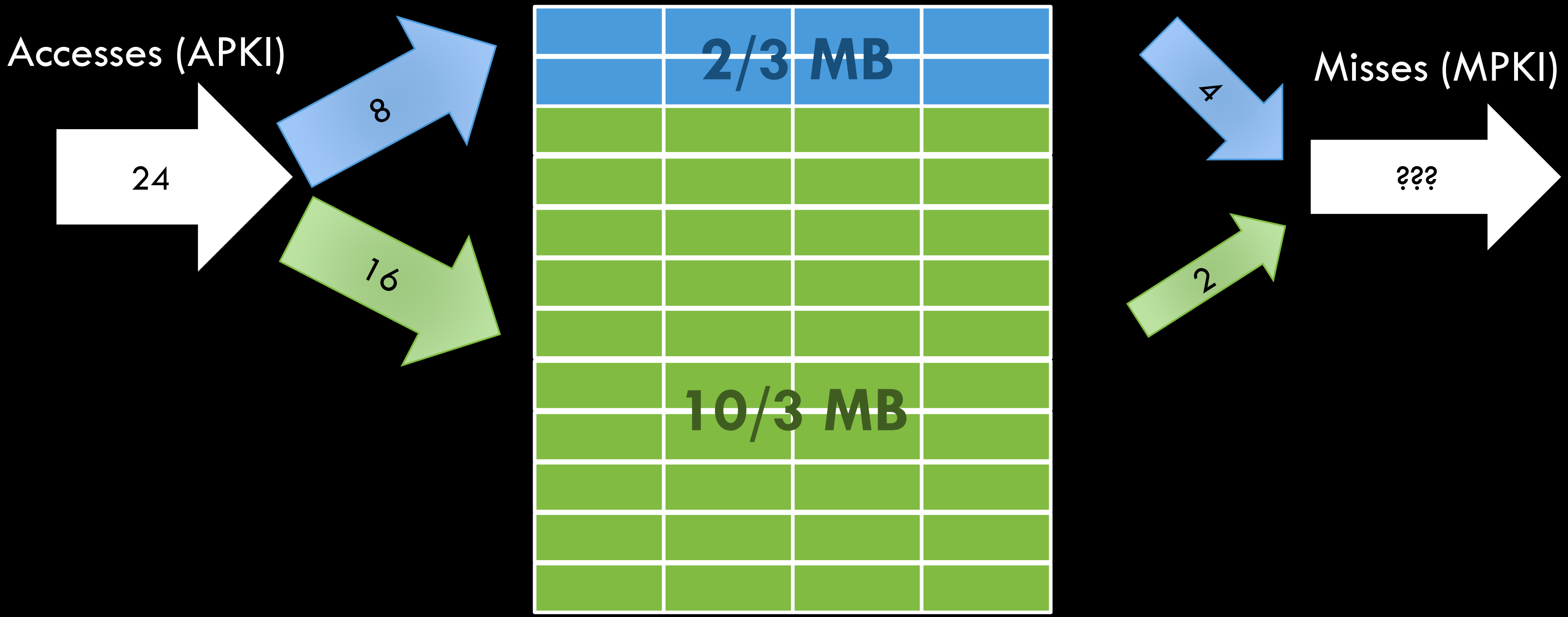




TALUS AT 4 MB



Spread accesses *disproportionally* across partitions to match baselines



EXAMPLE SUMMARY

Talus avoids cliffs by combining *efficient cache sizes* of baseline

Does not know or care about app or replacement details

- Just needs miss curve!

Nothing special about set partitioning; Talus works on other partitioning techniques

But how to choose partition configuration?

ROAD MAP

Talus example

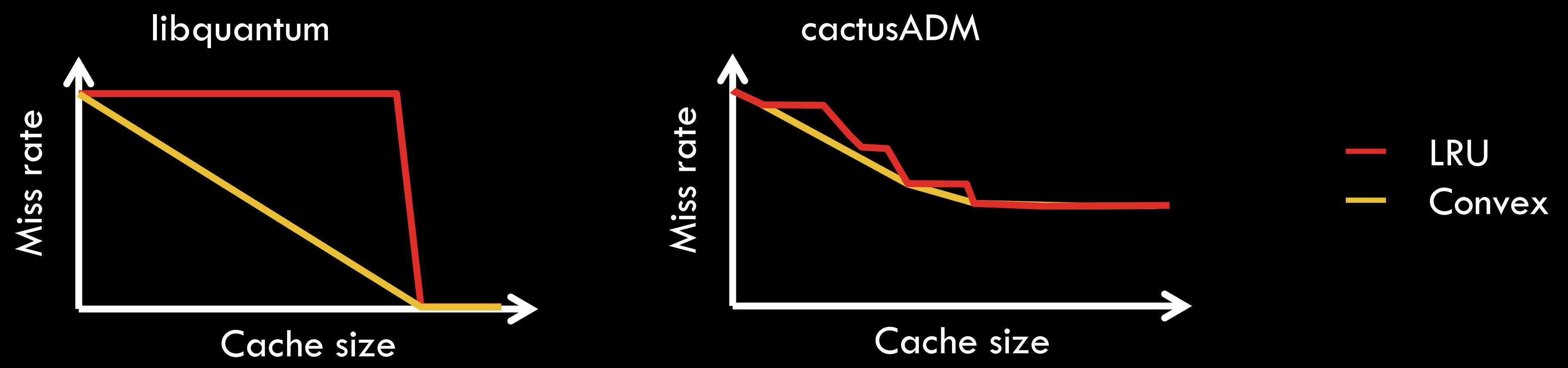
Theory

- Proof sketch
- Talus vs prior policies

Implementation

Evaluation

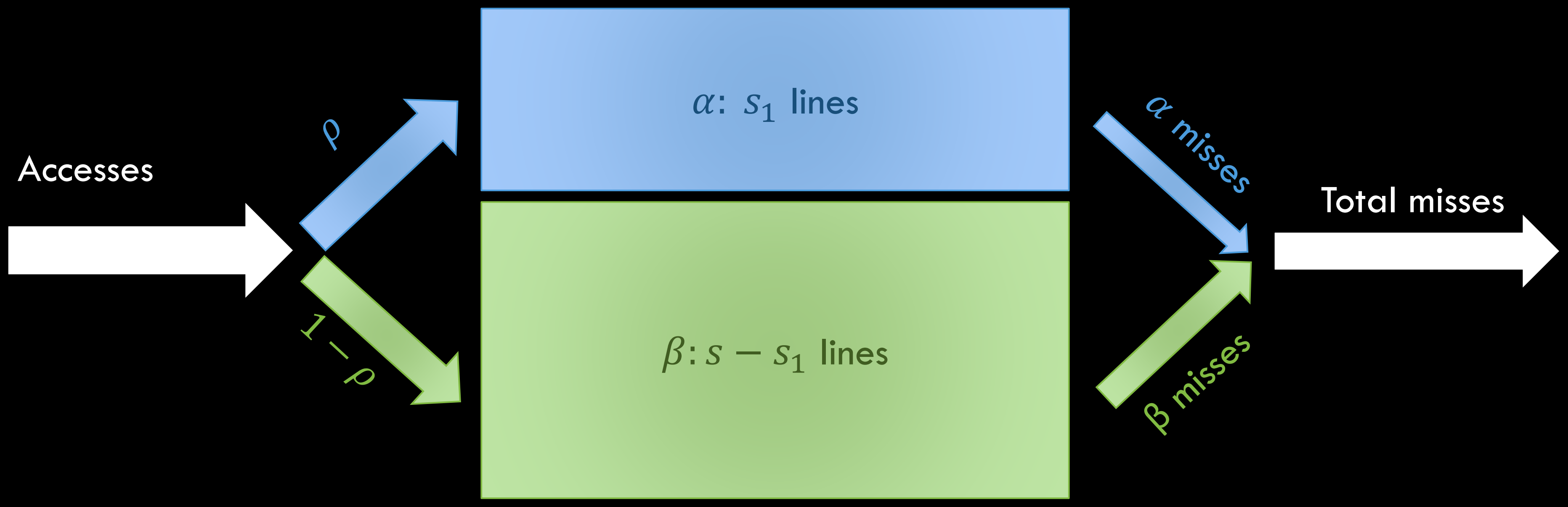
GOAL: *CONVEXITY* AVOIDS CLIFFS



Convex miss curves do not have cliffs

SHADOW PARTITIONING

Talus divides the cache (of size s) into *shadow partitions*, invisible to software



***Talus ensures convexity** under simple assumptions*

ASSUMPTIONS

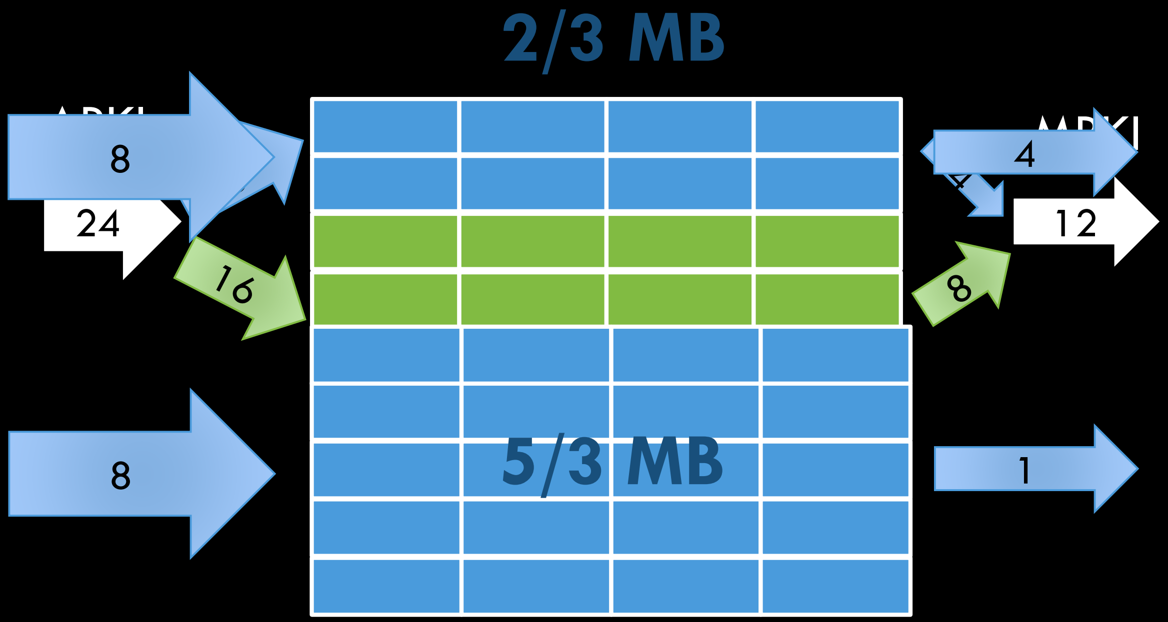
Miss curves are stable (eg, across tens of milliseconds)

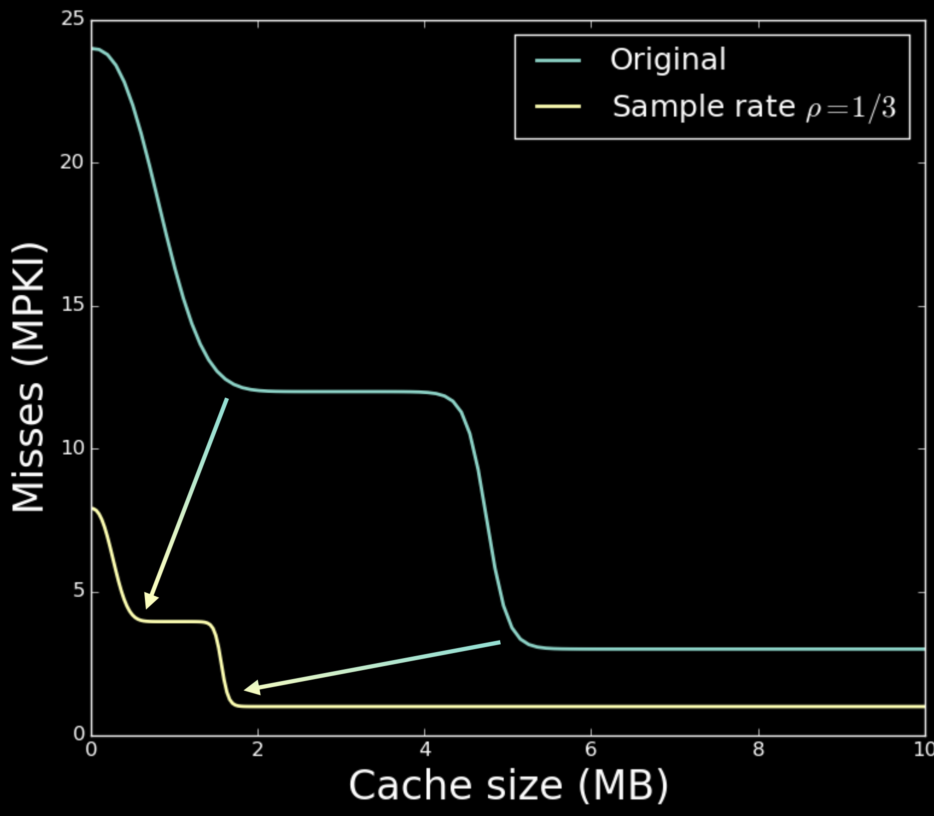
Cache size is the dominant factor in miss rate (ie, not associativity)

Pseudo-random sampling of an access stream yields a *statistically self-similar* stream

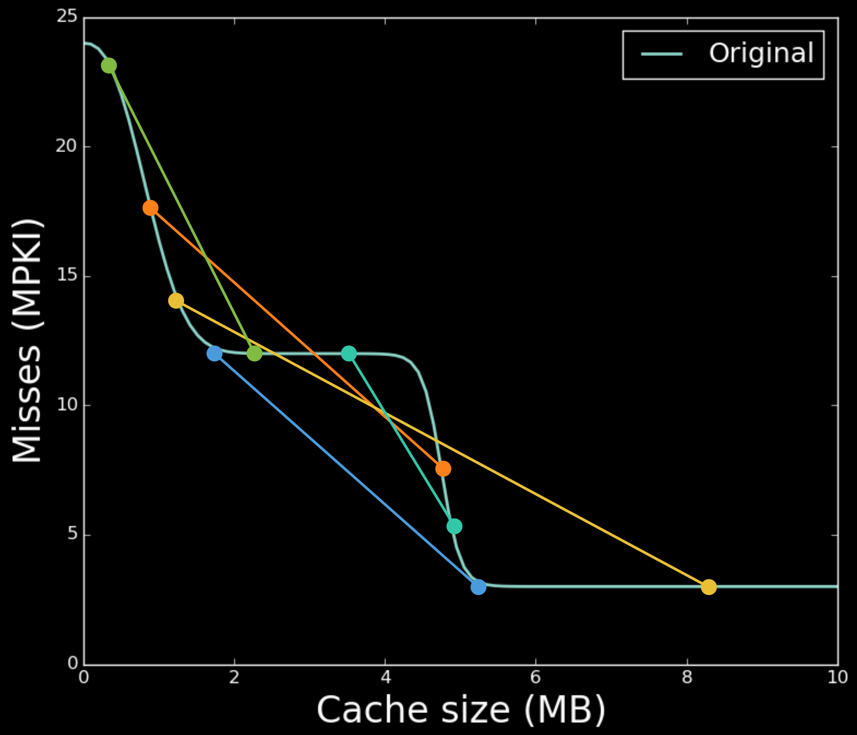
These assumptions are implicit in prior work (see paper)

SAMPLING SCALES THE MISS CURVE



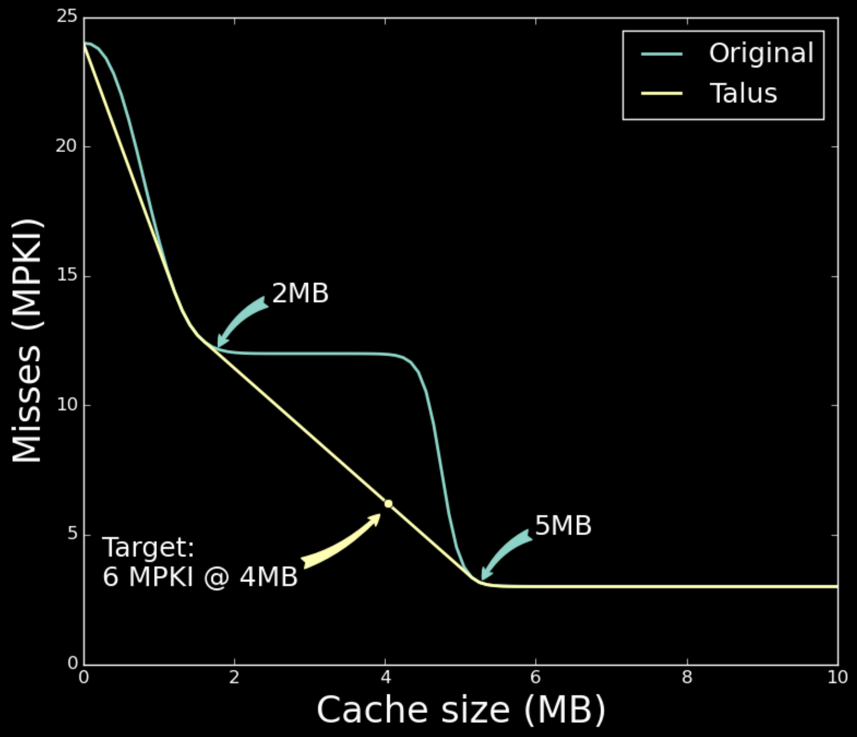


SHADOW PARTITIONING INTERPOLATES MPKI OF THE ORIGINAL MISS CURVE



TALUS GUARANTEES CONVEXITY

Just interpolate the *convex hull* of the original miss curve!



THERE'S MATH!

Miss curve scaling:

$$m'(s') = \rho\, m\left(\frac{s'}{\rho}\right)$$

Shadow partitioned miss rate:

$$m_{\text{shadow}(s)} = \rho\, m\left(\frac{s_1}{\rho}\right) + (1-\rho)\, m\left(\frac{s-s_1}{1-\rho}\right)$$

How to interpolate between α and β:

$$\rho = \frac{\beta - s}{\beta - \alpha},\ s_1 = \rho\alpha$$

ROAD MAP

Motivation

Talus example

Theory

- Proof sketch
- **Talus vs prior policies**

Implementation

Evaluation

PRIOR TECHNIQUE: BYPASSING

Bypassing is a common replacement technique to avoid thrashing

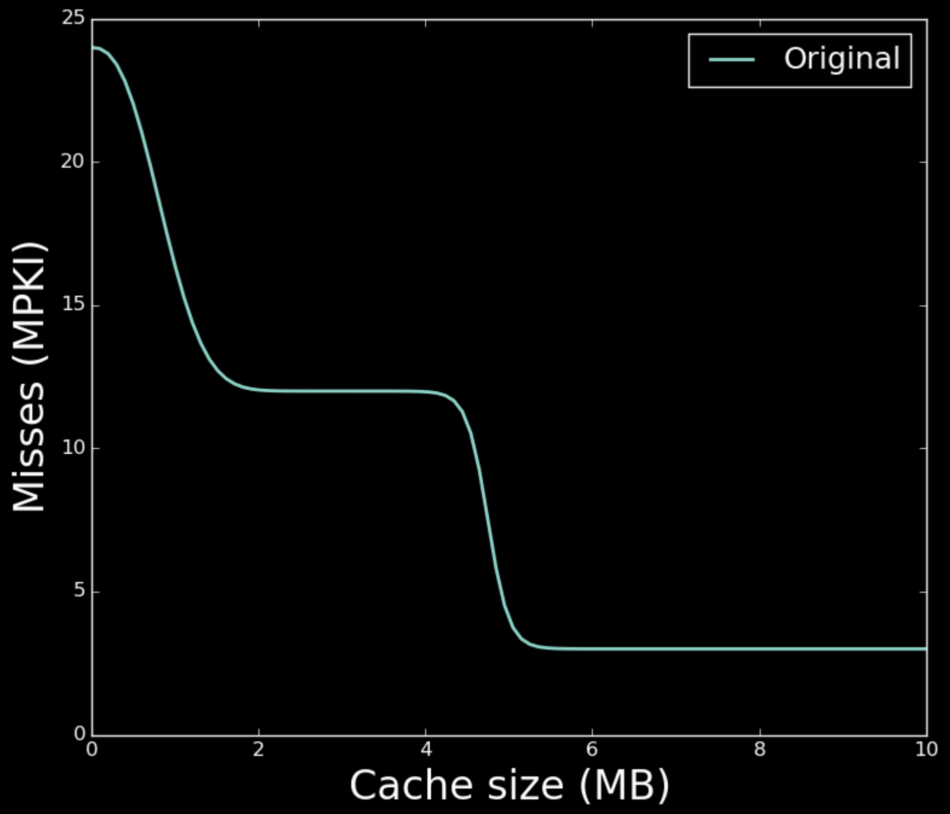
- E.g., BIP [ISCA'07] bypasses 31/32 accesses

We compute optimal bypassing rate from miss curve

Bypassing handles some kinds of cliffs, but not all

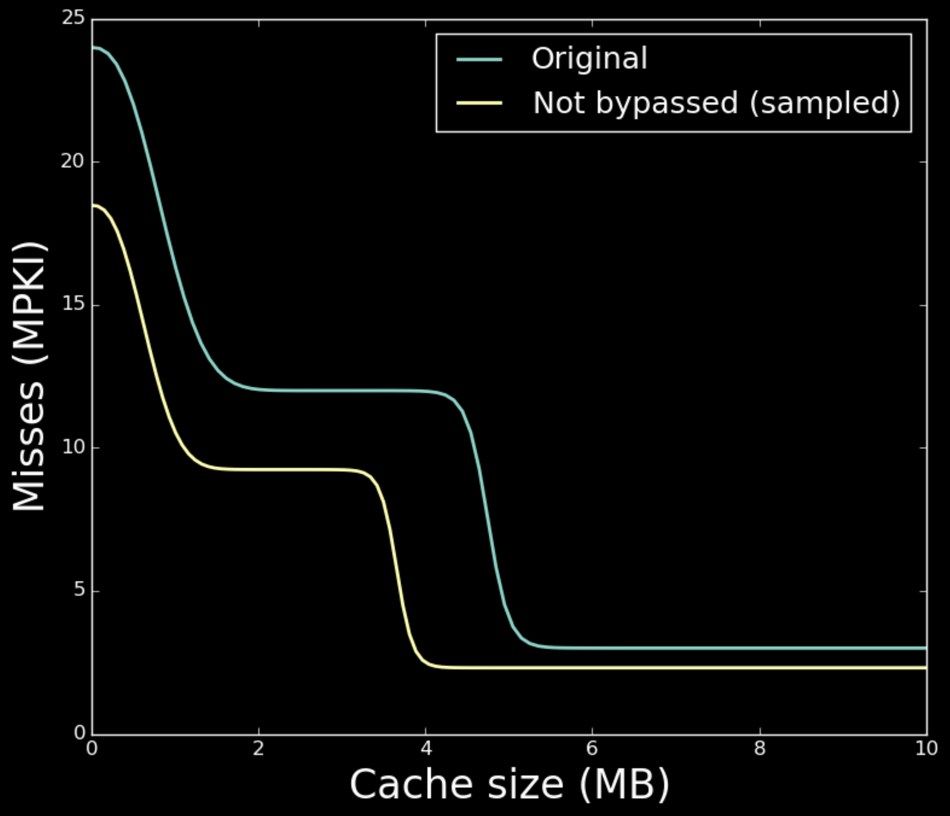
→ ***Talus outperforms bypassing on some access patterns***

BYPASSING PRODUCES COMPETING EFFECTS



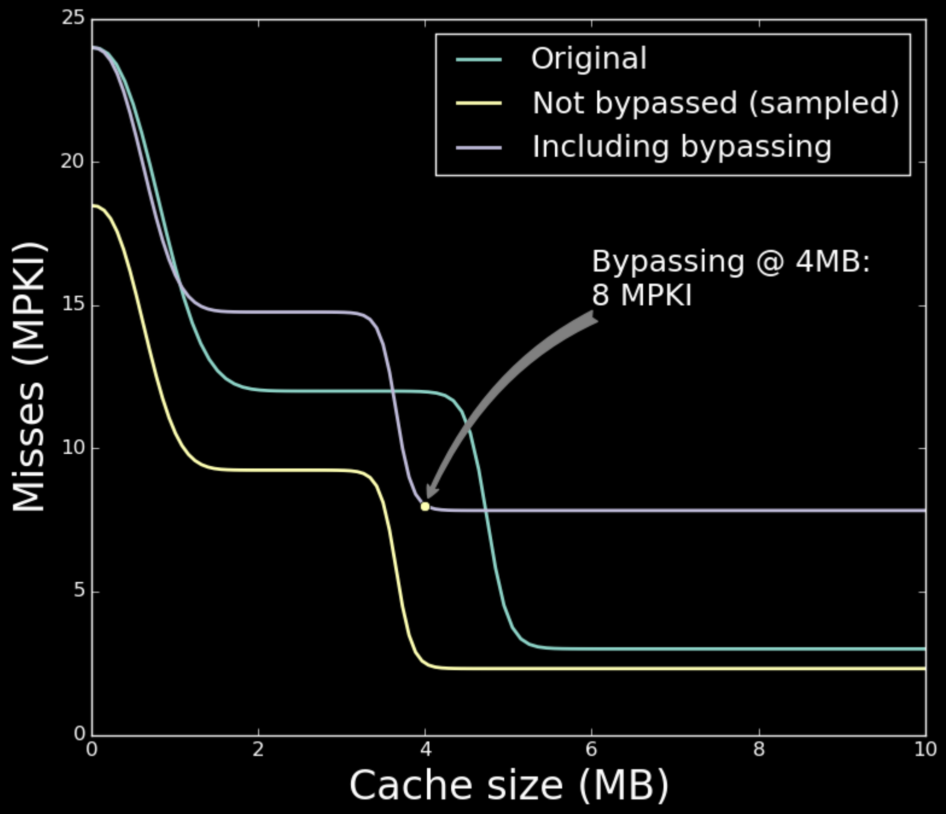
BYPASSING PRODUCES COMPETING EFFECTS

- Bypassing reduces misses for sampled accesses



BYPASSING PRODUCES COMPETING EFFECTS

- Bypassing reduces misses for sampled accesses
- …But adds misses for bypassed accesses



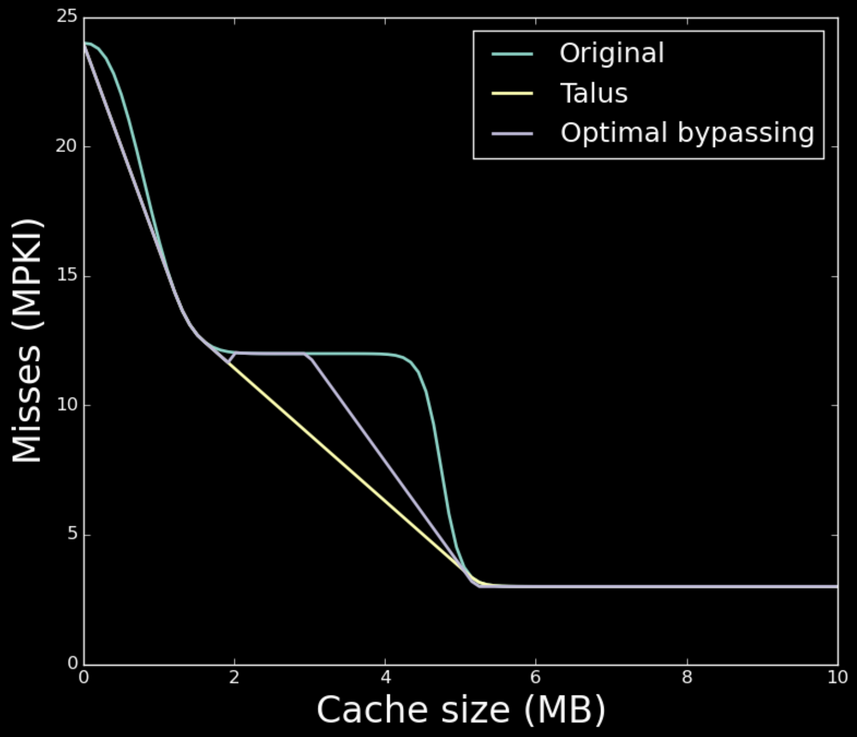
See paper for details!

TALUS VS BYPASSING

Talus reduces miss rate

Talus is convex

- I.e., avoids cliffs!



ROAD MAP

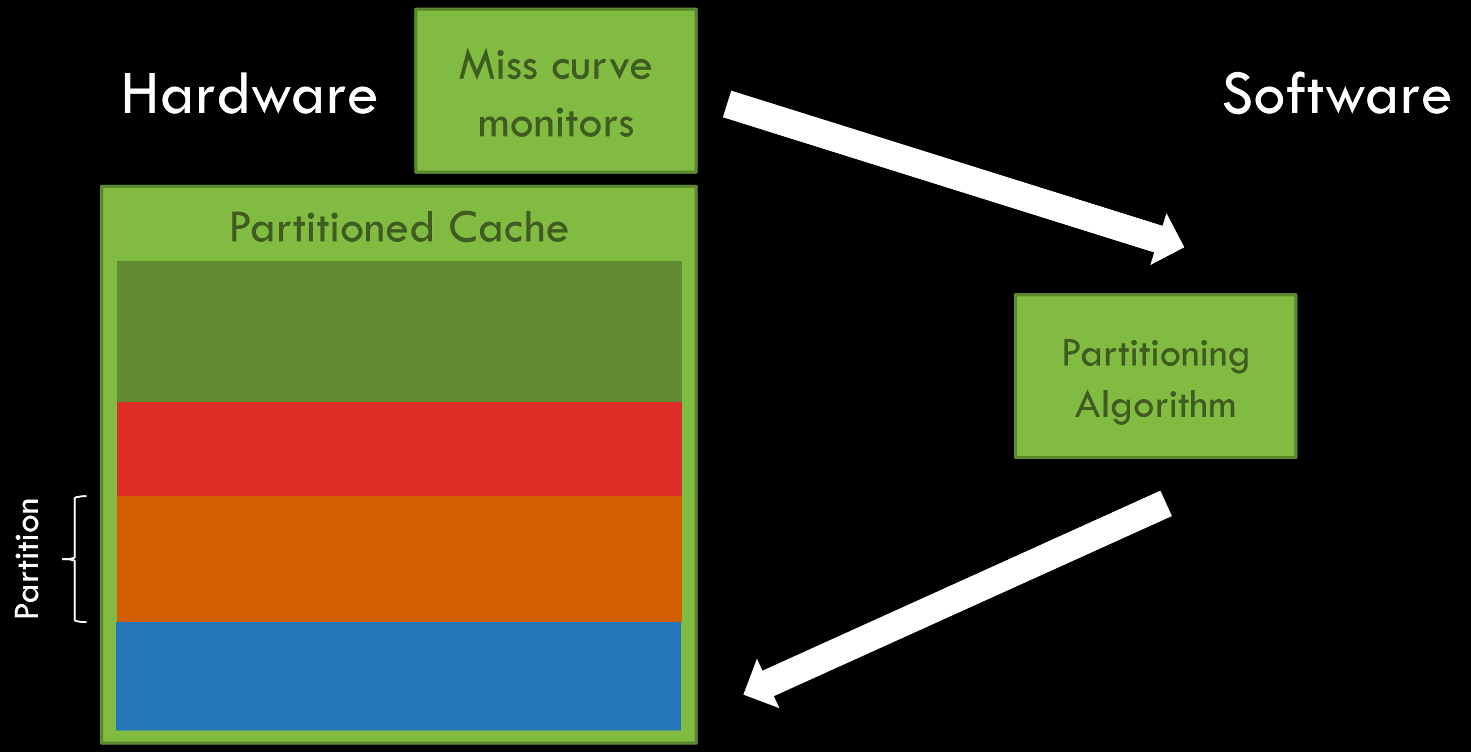
Talus example

Theory

Implementation

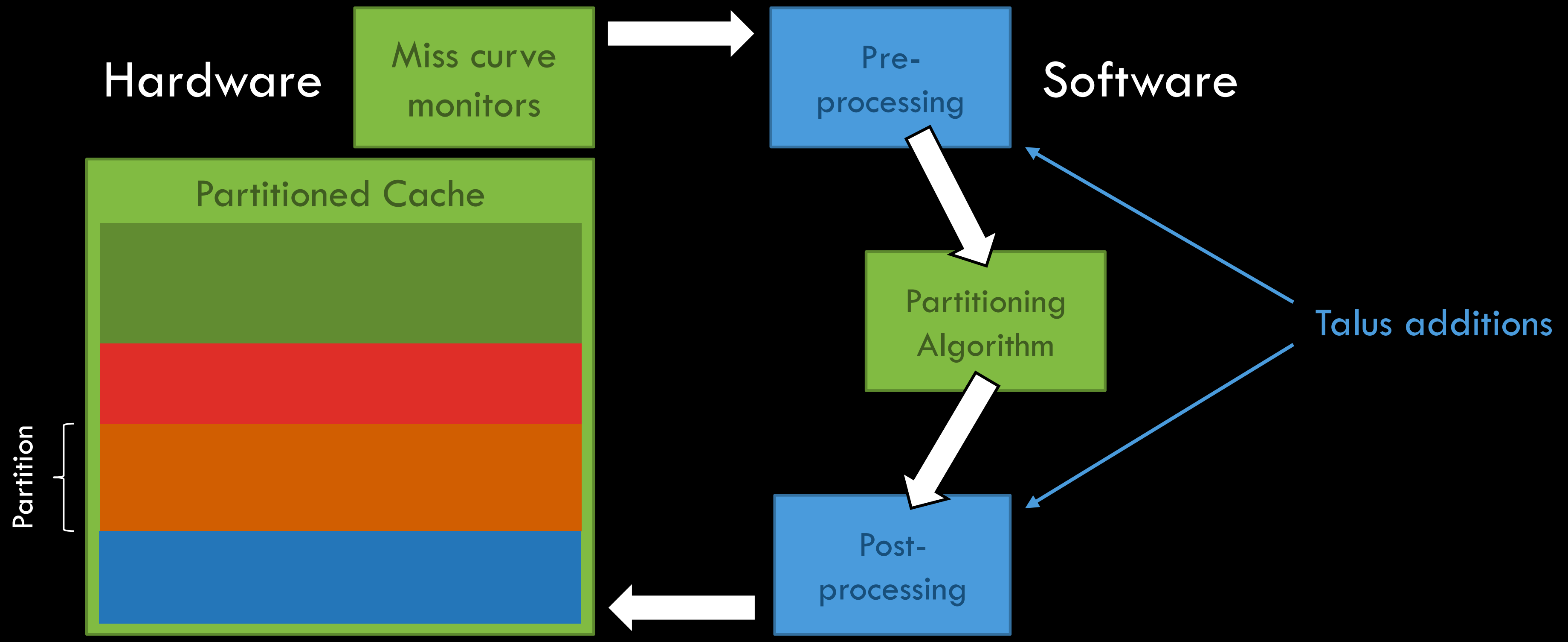
Evaluation

CONVENTIONAL PARTITIONED CACHE

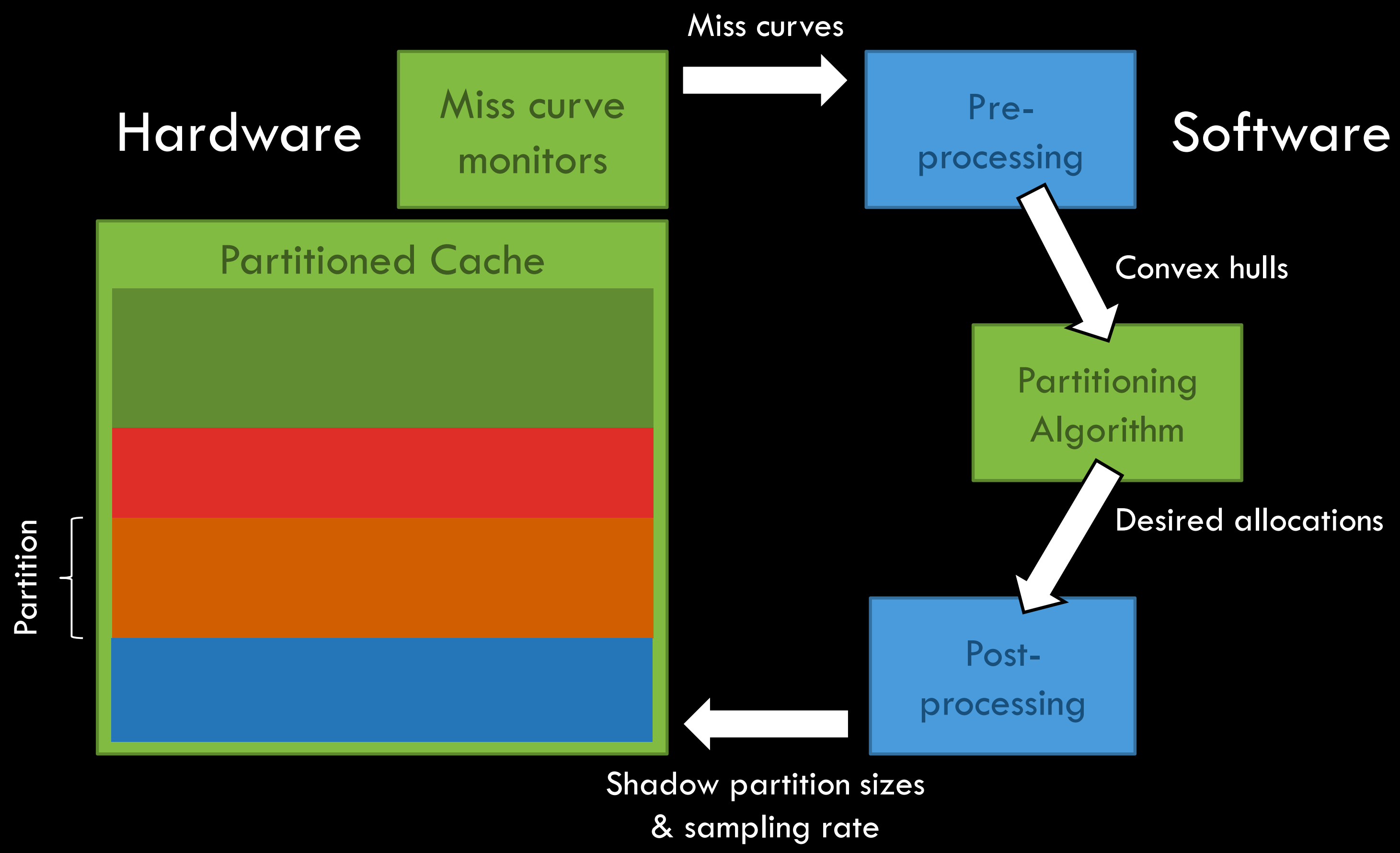


Eg, UCP [MICRO'06]

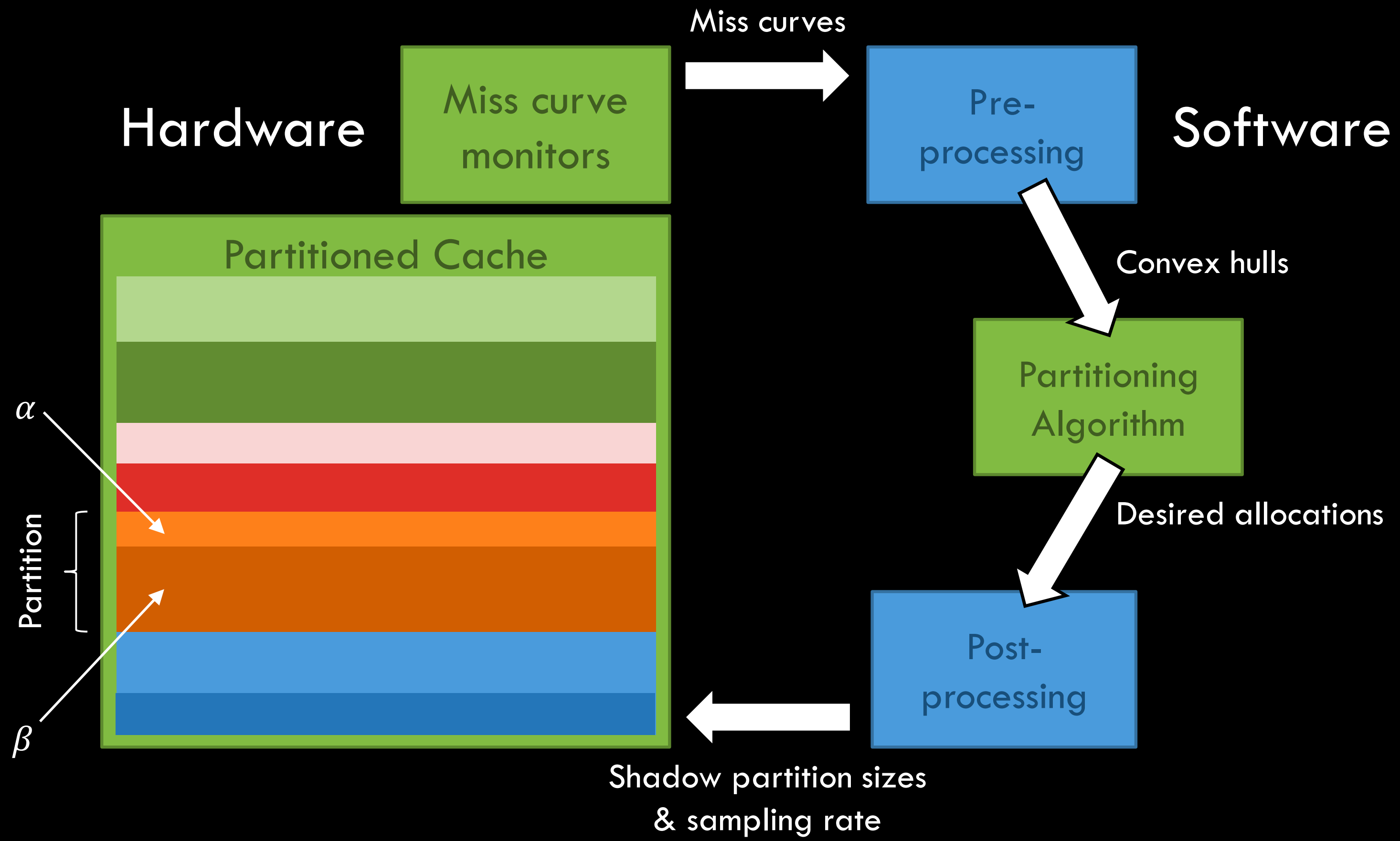
EFFICIENT TALUS IMPLEMENTATION



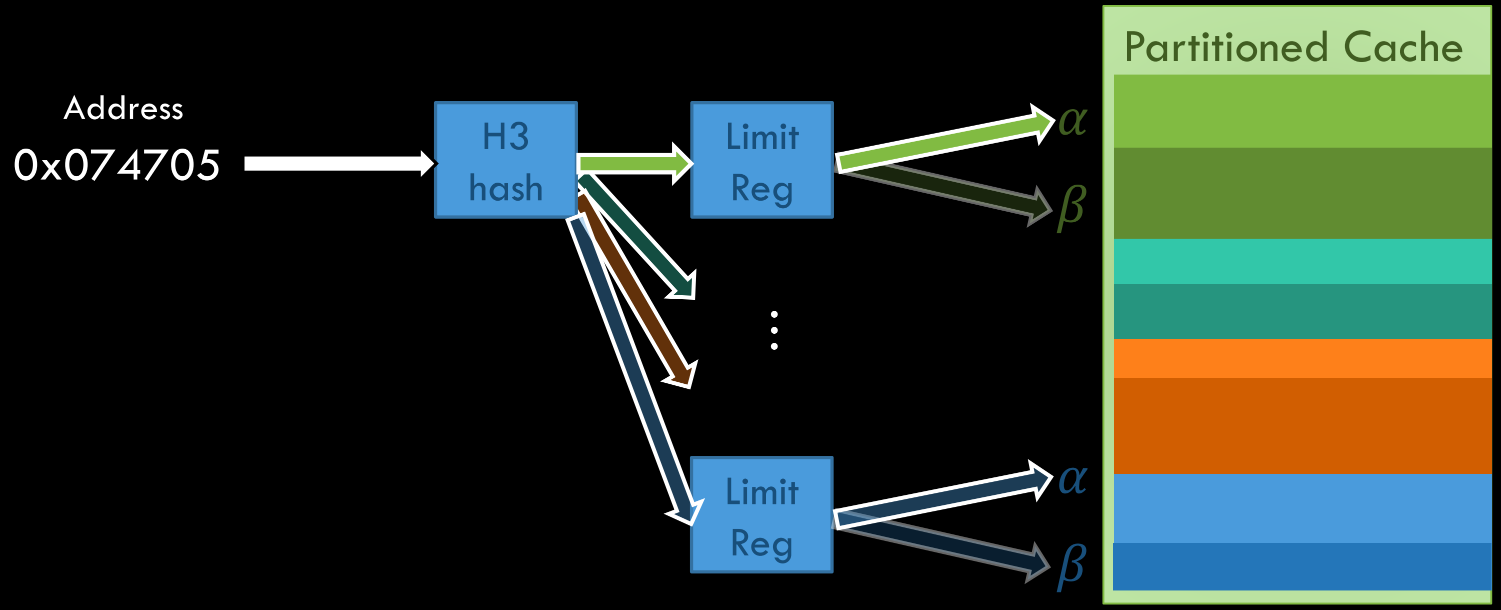
EFFICIENT TALUS IMPLEMENTATION



EFFICIENT TALUS IMPLEMENTATION



EFFICIENT TALUS IMPLEMENTATION



TALUS IMPOSES LOW OVERHEADS

Computing convex hulls is cheap: $O(N)$

Computing shadow partition sizes is cheap: $O(1)$

Talus reduces software overheads by making simple algorithms perform well

Software

Shadow partitioning is cheap: similar monitors to prior work (see paper), 1 bit per tag, 8 bits per partition, simple hash function

Talus improves cache performance and adds <1% state

Hardware

EVALUATION CLAIMS

We compare Talus to high-performance replacement policies and partitioning schemes

Talus is convex in practice

Single-program: Talus gets similar performance to prior replacement policies

Multi-program: Talus greatly simplifies cache partitioning and slightly outperforms prior, complex partitioning algorithms

Talus combines the benefits of high-performance replacement and partitioning

METHODOLOGY

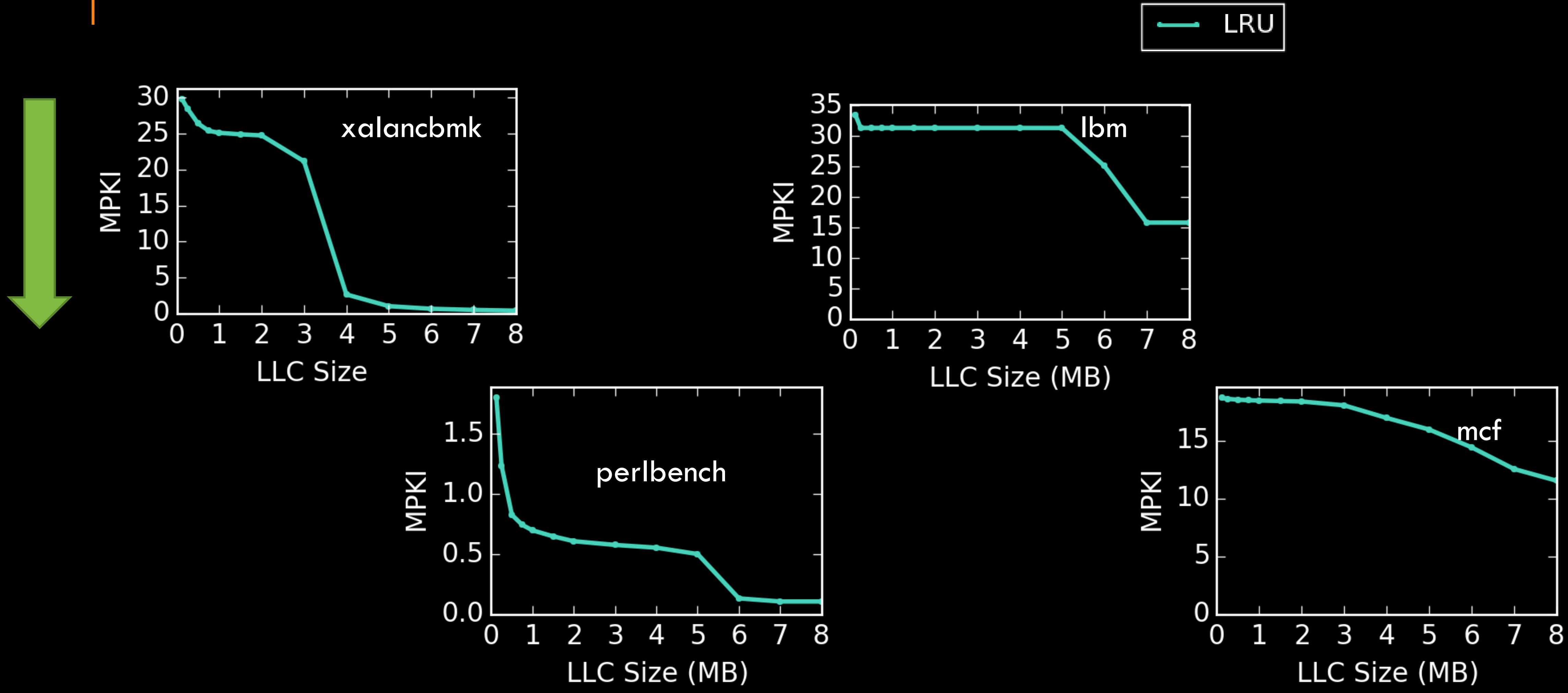
Evaluate 1- and 8-core system similar to Silvermont on zsim

- See paper for details

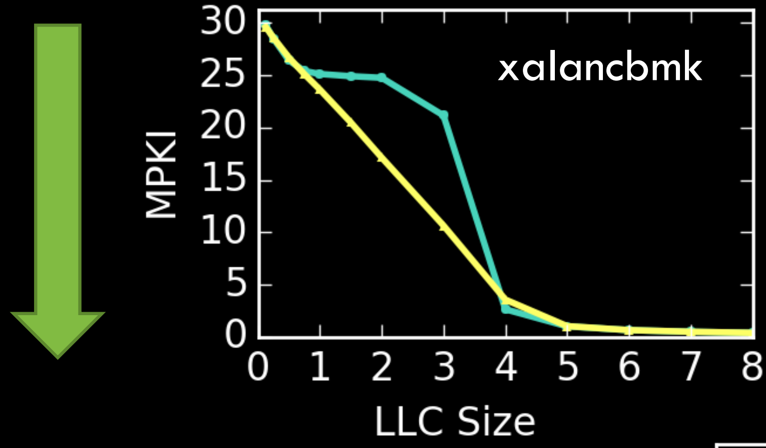
Individual SPEC CPU2006 benchmarks + random mixes

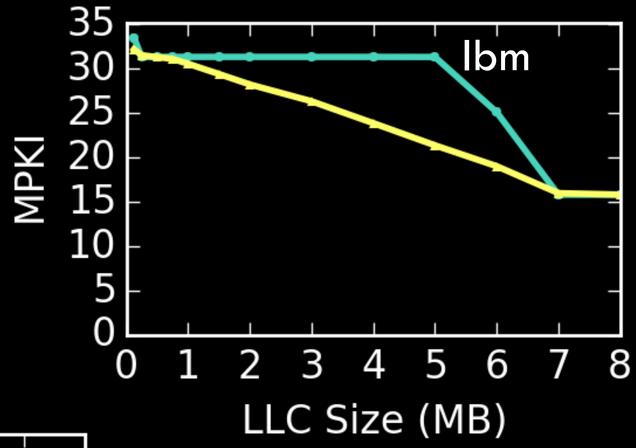
Talk only shows Talus on LRU with Vantage partitioning (*Talus +V/LRU*)

EVALUATION: SINGLE-THREADED

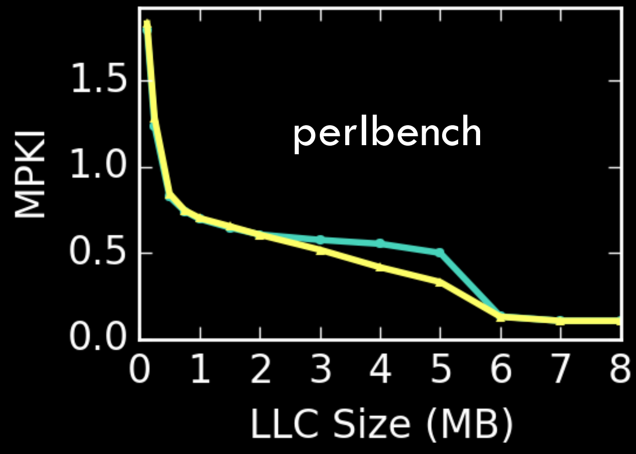


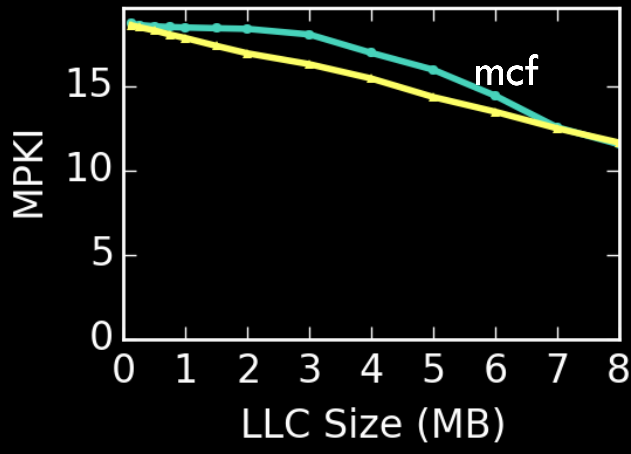
EVALUATION: SINGLE-THREADED



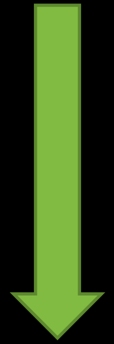


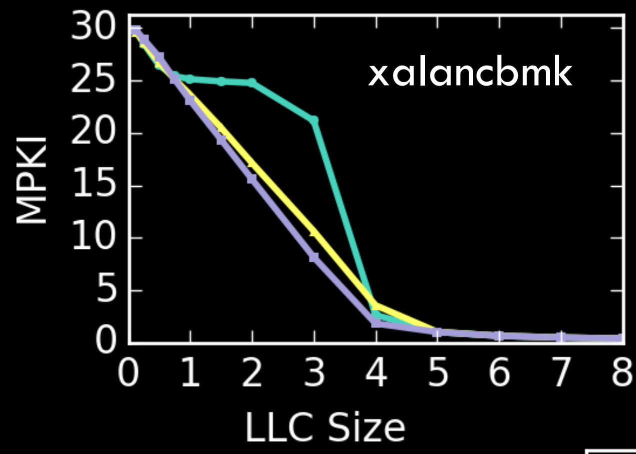
Talus is convex in practice!

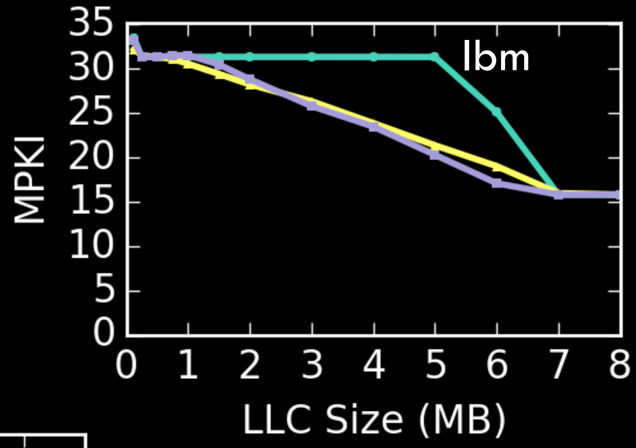




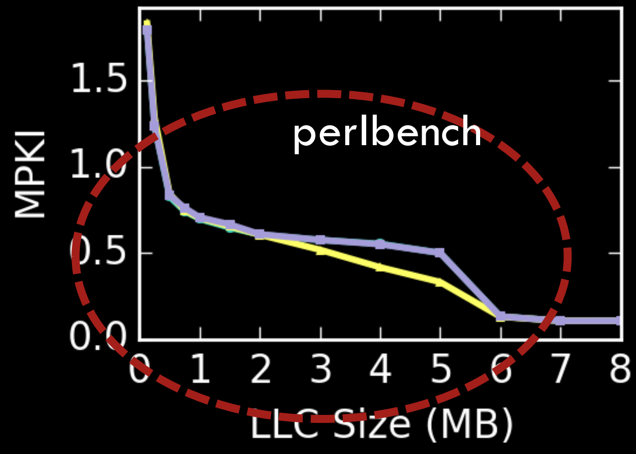
EVALUATION: SINGLE-THREADED

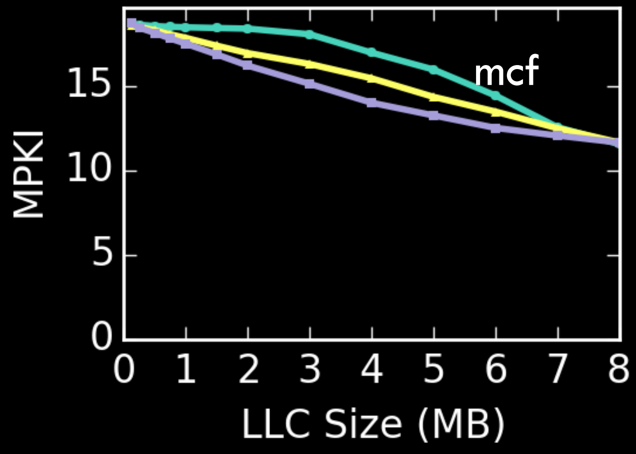




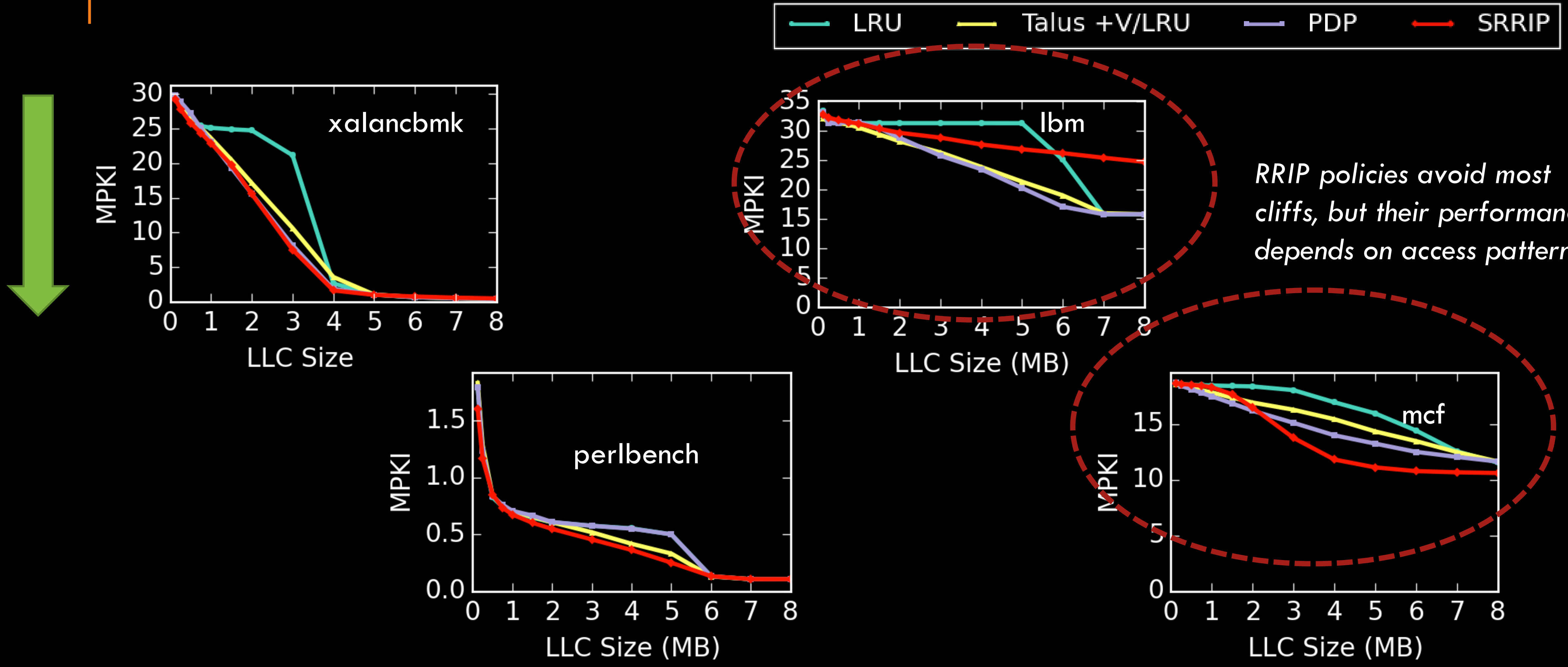


PDP performs similarly but is not always convex



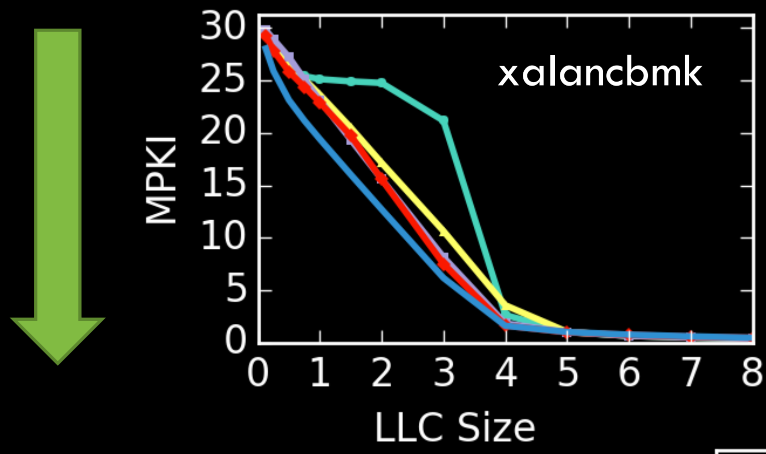


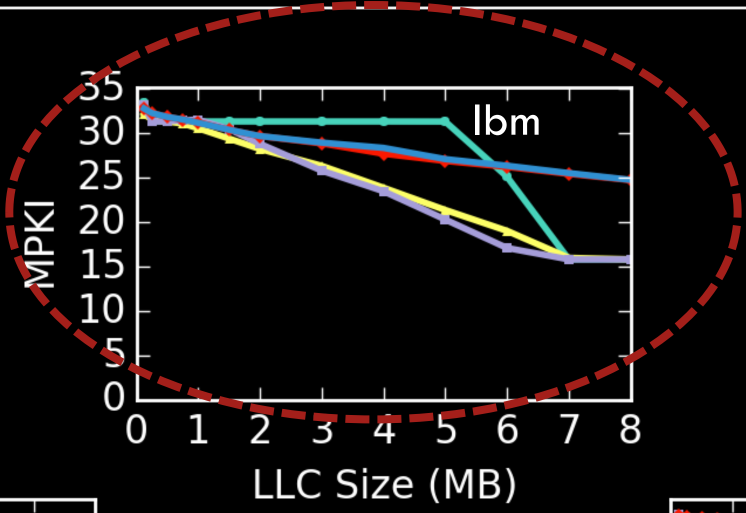
EVALUATION: SINGLE-THREADED



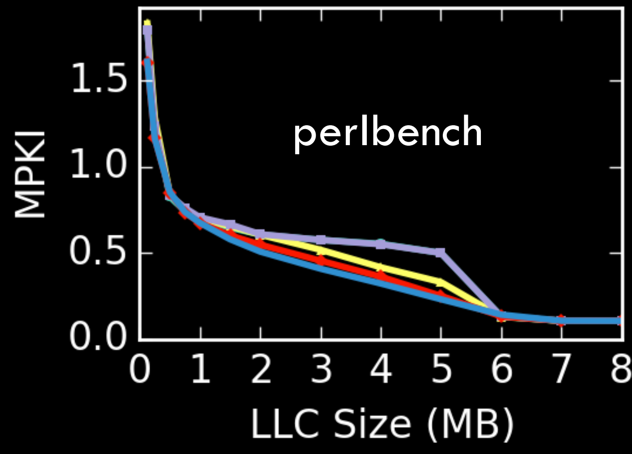
RRIP policies avoid most cliffs, but their performance depends on access pattern

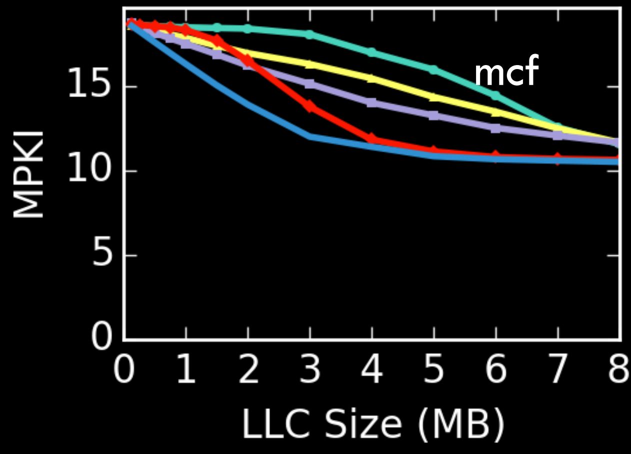
EVALUATION: SINGLE-THREADED



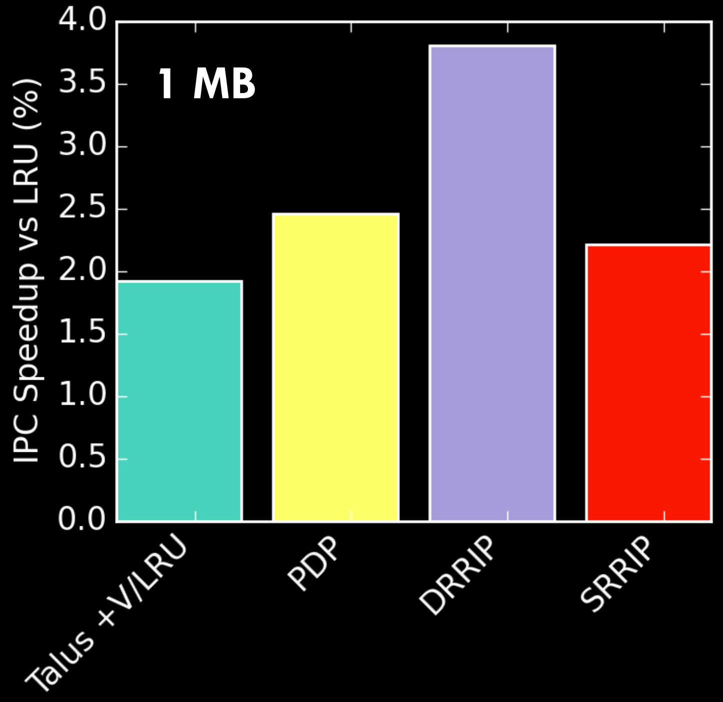


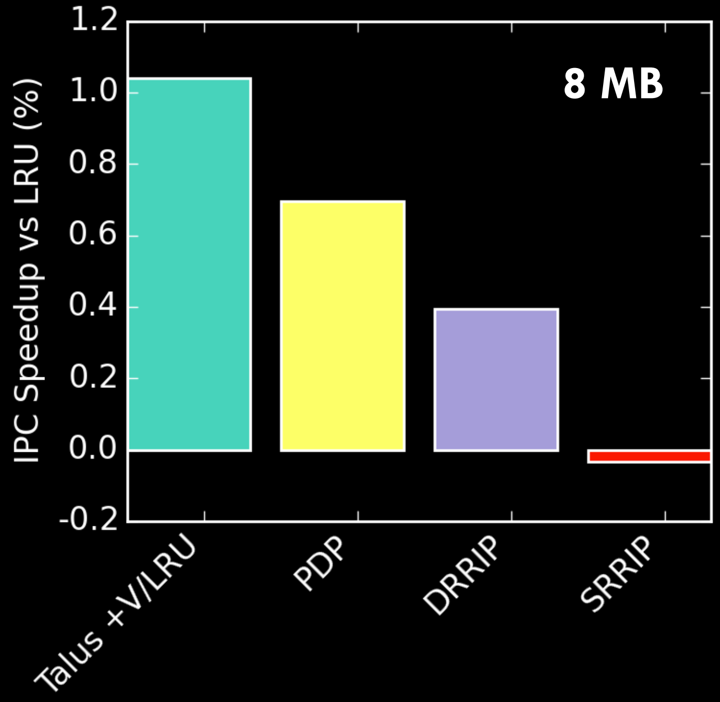
RRIP policies avoid most cliffs, but their performance depends on access pattern





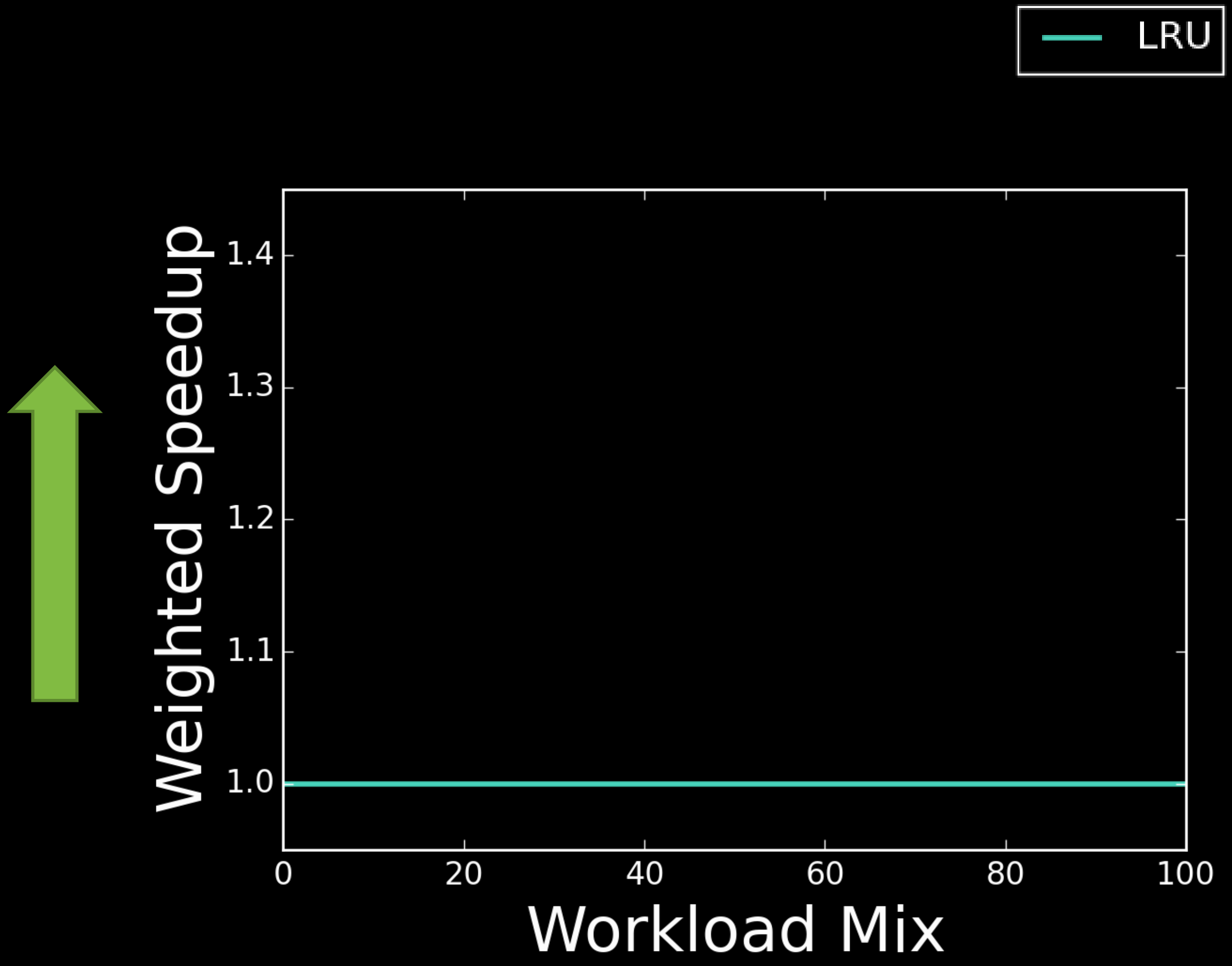
GMEAN IPC IMPROVEMENT VS LRU



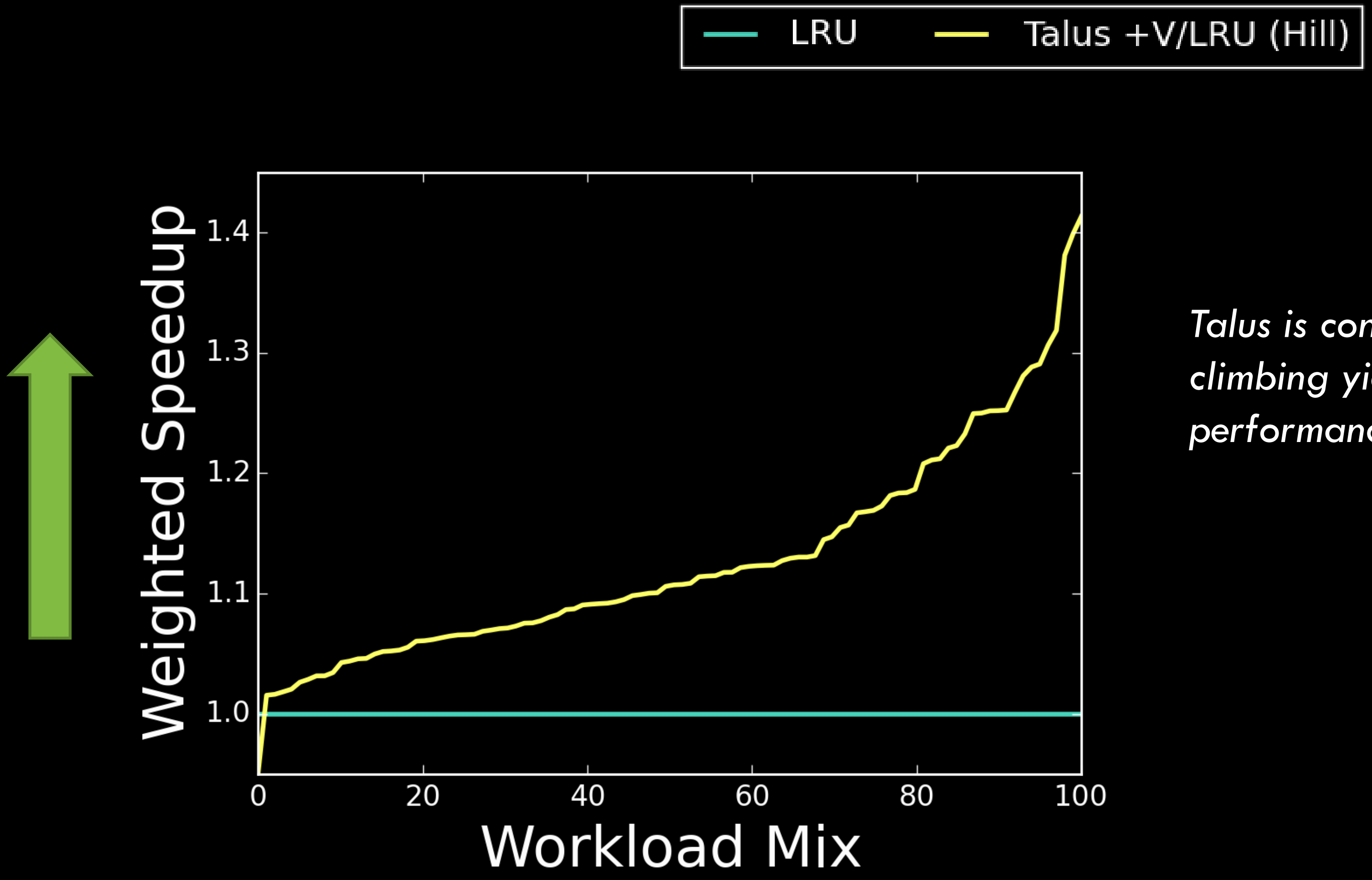


Talus on LRU gets similar speedups to prior policies.

MULTI-PROGRAMMED PERFORMANCE

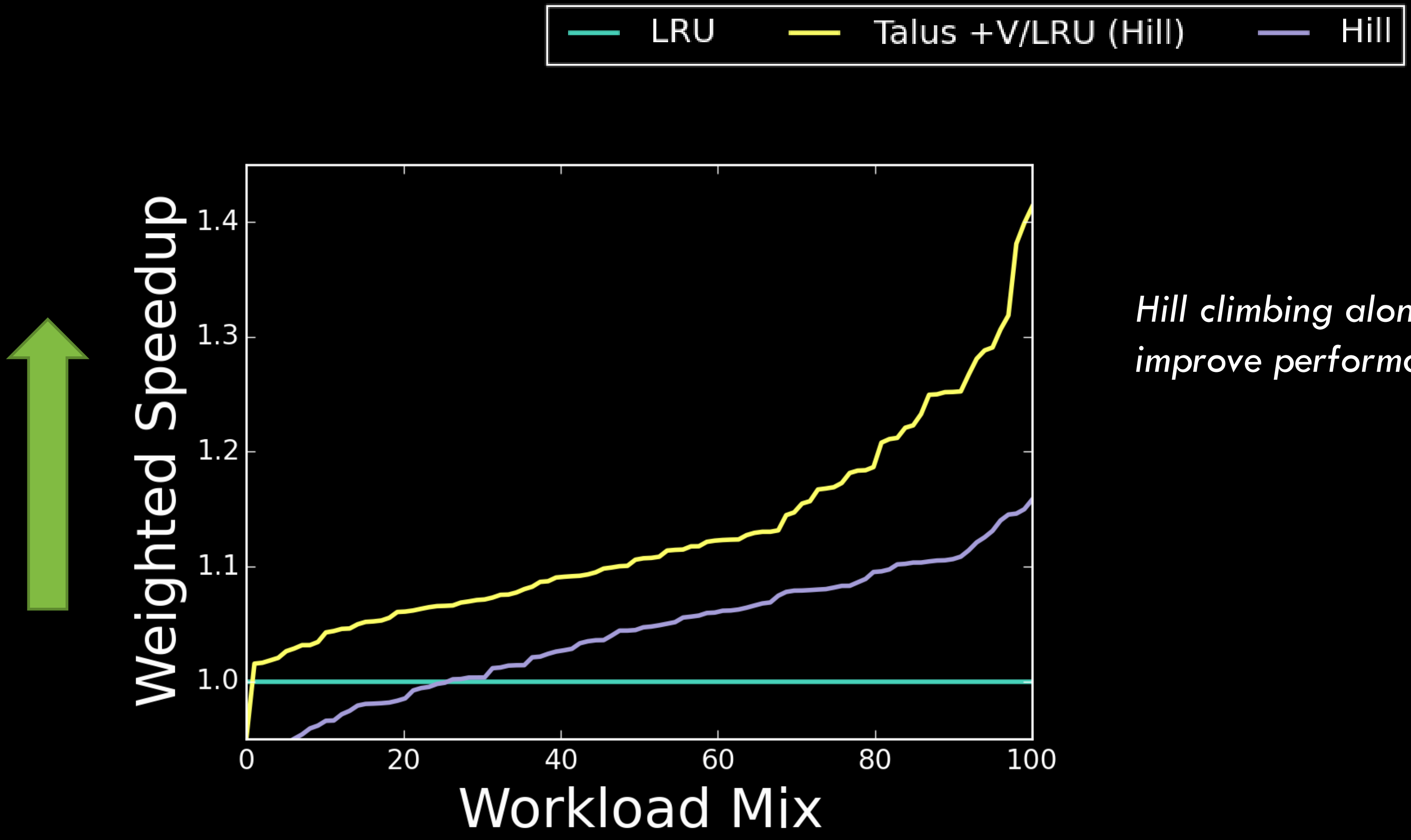


MULTI-PROGRAMMED PERFORMANCE



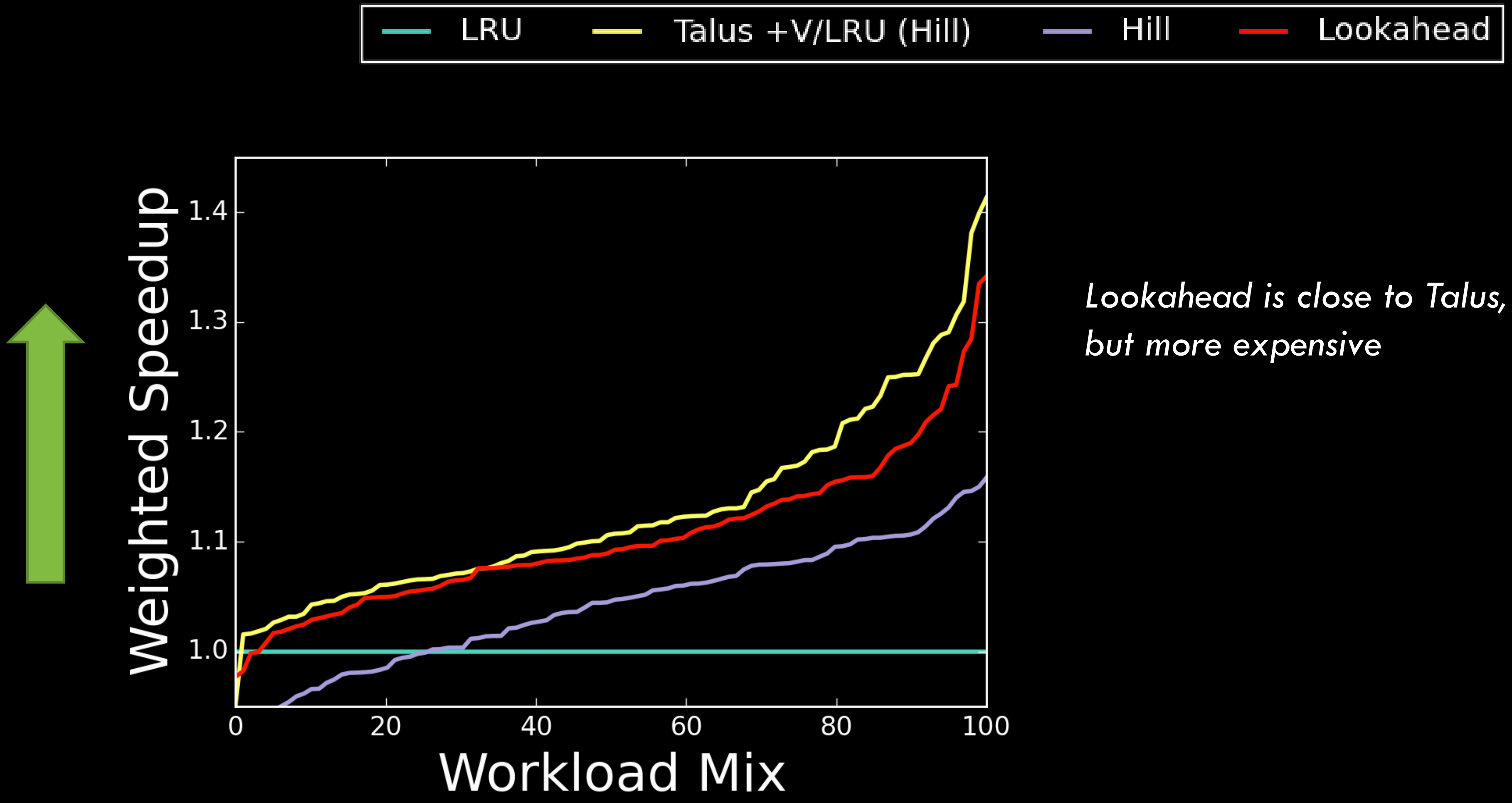
Talus is convex ➔ naïve hill climbing yields large performance gains

MULTI-PROGRAMMED PERFORMANCE



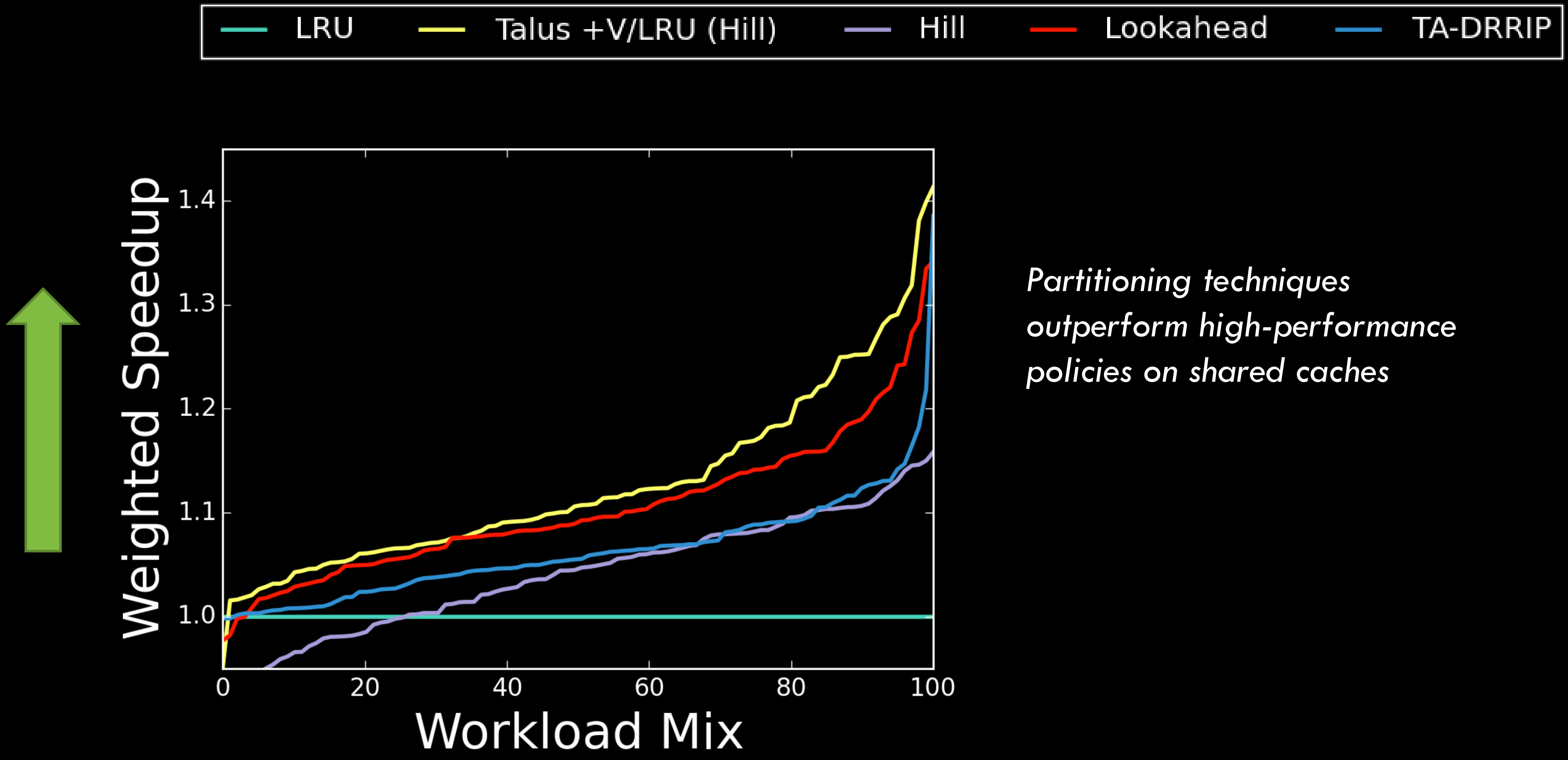
Hill climbing alone does not improve performance much

MULTI-PROGRAMMED PERFORMANCE



Lookahead is close to Talus, but more expensive

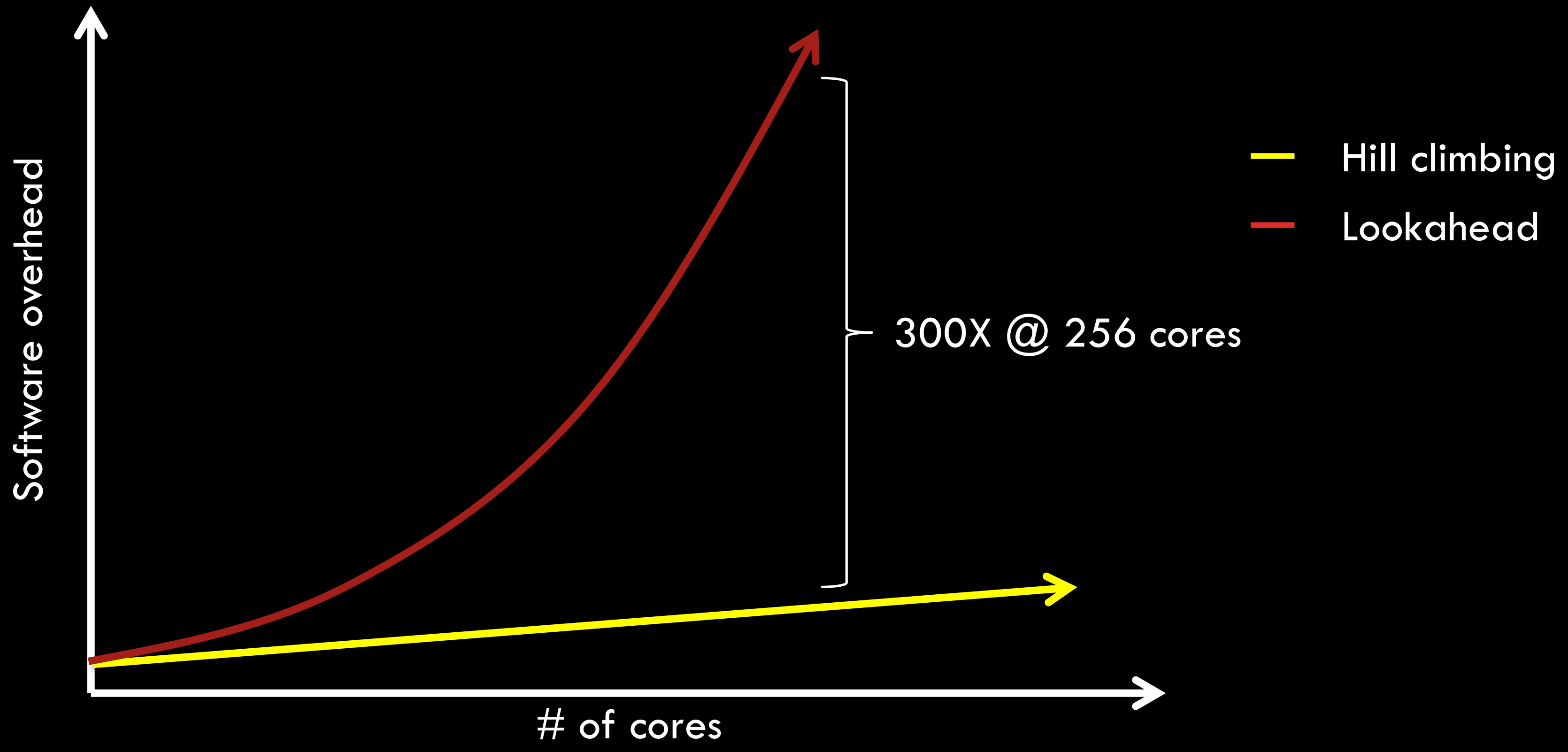
MULTI-PROGRAMMED PERFORMANCE



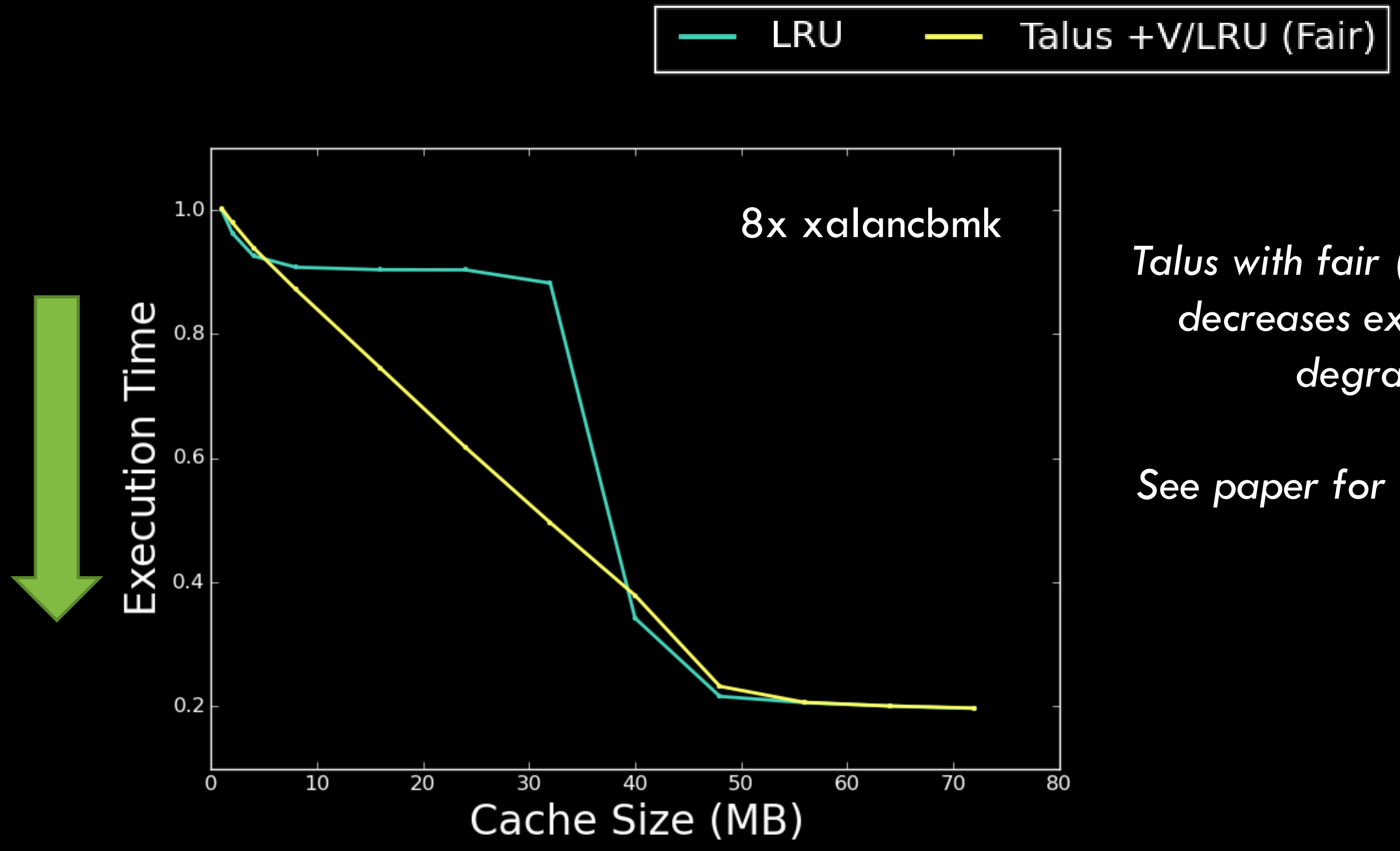
Partitioning techniques outperform high-performance policies on shared caches

TALUS SIMPLIFIES PARTITIONING ALGORITHMS AND REDUCES OVERHEADS

Efficient alternatives to Lookahead add significant complexity!



MULTI-PROGRAMMED FAIRNESS



Talus with fair (equal-sized) partitions decreases execution time without degrading fairness.

See paper for other apps & schemes!

MORE CONTENT IN PAPER!

Detailed proofs

Prove optimal replacement is convex

Evaluation:

- Talus works on way partitioning
- Talus works with SRRIP
- More benchmarks
- Talus works with pre-fetching and multi-threading

THANK YOU!

- Talus avoids cliffs and ensures convexity
  - Proven under simple assumptions
  - Verified by experiment
- Analysis of *shadow partitioning* shows advantages vs bypassing
- Talus improves performance and simplifies cache partitioning
- *Talus combines the benefits of high-performance replacement and partitioning*