VANTAGE: SCALABLE AND EFFICIENT FINE-GRAIN CACHE PARTITIONING

Daniel Sanchez and Christos Kozyrakis
Stanford University

ISCA-38, June 6th 2011

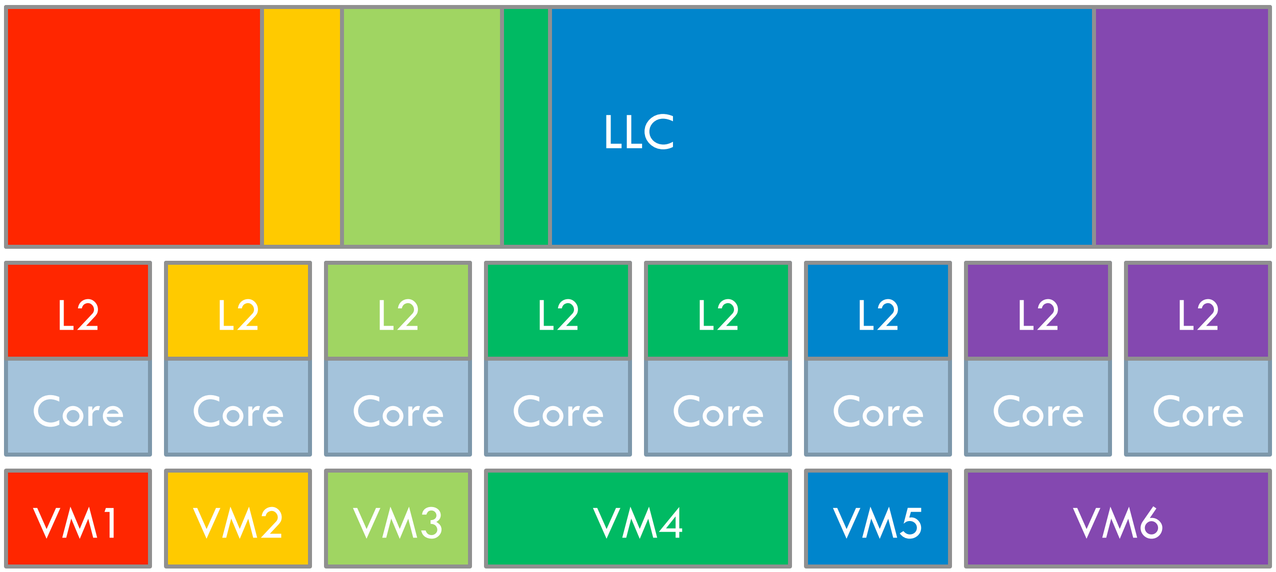
Executive Summary

- Problem: Interference in shared caches
  - Lack of isolation → no QoS
  - Poor cache utilization → degraded performance
- Cache partitioning addresses interference, but current partitioning techniques (e.g. way-partitioning) have serious drawbacks
  - Support few coarse-grain partitions → do not scale to many-cores
  - Hurt associativity → degraded performance
- Vantage solves deficiencies of previous partitioning techniques
  - Supports hundreds of fine-grain partitions
  - Maintains high associativity
  - Strict isolation among partitions
  - Enables cache partitioning in many-cores

Outline

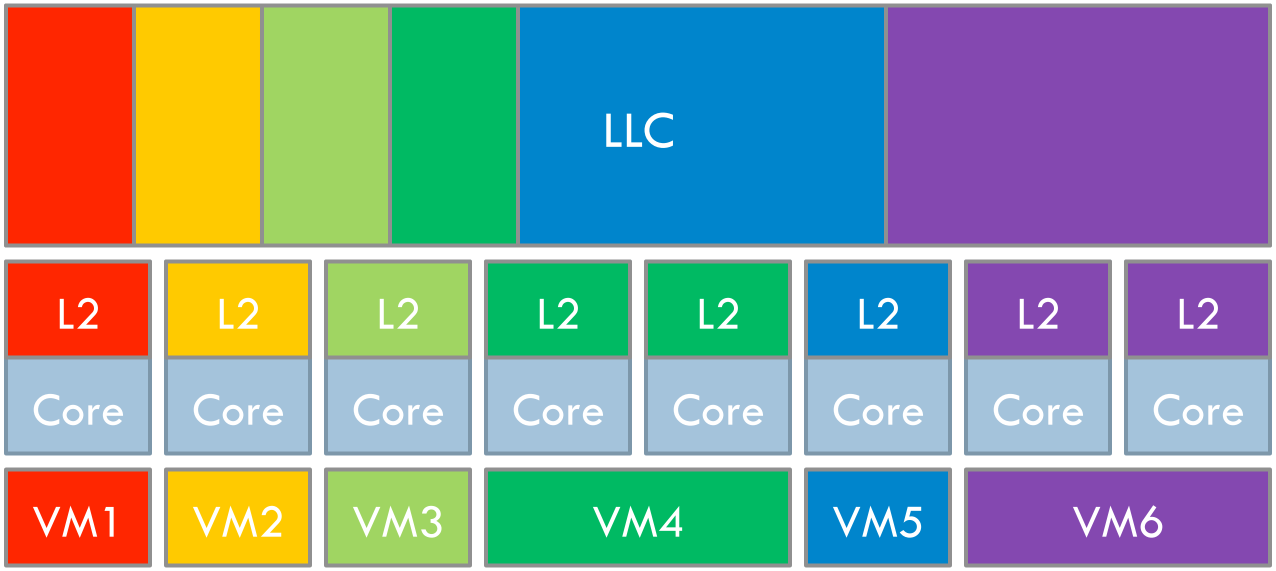
- Introduction
- Vantage Cache Partitioning
- Evaluation

Motivation



- Fully shared last-level caches are the norm in multi-cores
  - ✓ Better cache utilization, faster communication, cheaper coherence
  - ✗ Interference → performance degradation, no QoS
- Increasingly important problem due to more cores/chip and virtualization, consolidation (datacenter/cloud)
  - Major performance and energy losses due to cache contention (~2x)
  - Consolidation opportunities lost to maintain SLAs

Cache Partitioning



- Cache partitioning: Divide cache space among competing workloads (threads, processes, VMs)
  - ✓ Eliminates interference, enabling QoS guarantees
  - ✓ Adjust partition sizes to maximize performance, fairness, satisfy SLA...
  - ✗ Previously proposed partitioning schemes have major drawbacks

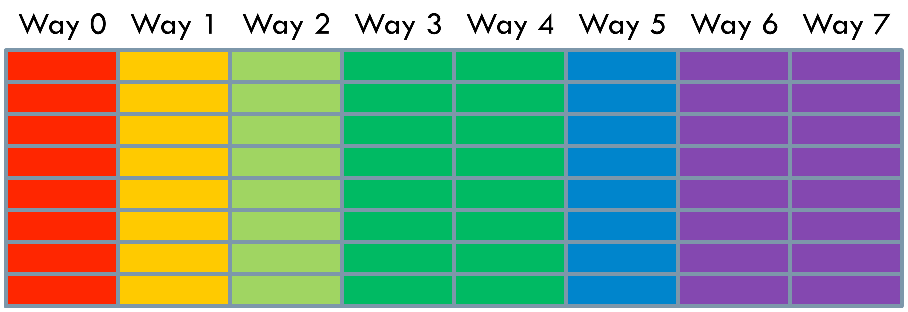
Cache Partitioning = Policy + Scheme

- Cache partitioning consists of a policy (decide partition sizes to achieve a goal, e.g. fairness) and a scheme (enforce sizes)
- Focus on the scheme
- For policy to be effective, scheme should be:
  1. Scalable: can create hundreds of partitions
  2. Fine-grain: partitions sizes specified in cache lines
  3. Strict isolation: partition performance does not depend on other partitions
  4. Dynamic: can create, remove, resize partitions efficiently
  5. Maintains associativity
  6. Independent of replacement policy
  7. Simple to implement

Maintain high cache performance

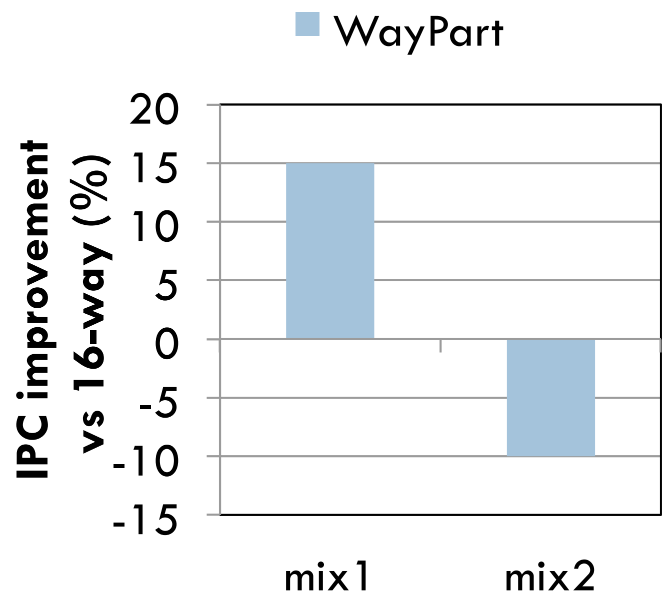
Existing Schemes with Strict Guarantees

- Based on restricting line placement
- Way partitioning: Restrict insertions to specific ways



✓ Strict isolation
✓ Dynamic
✓ Indep of repl policy
✓ Simple

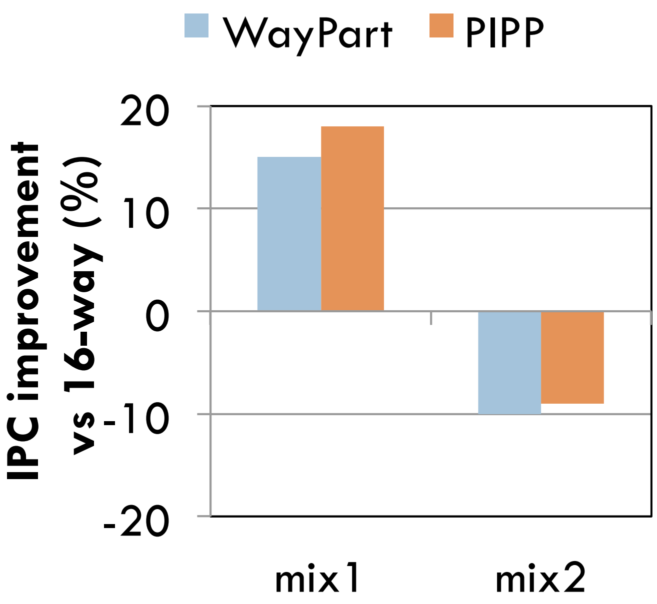
✗ Few coarse-grain partitions
✗ Hurts associativity



Existing Schemes with Soft Guarantees

- Based on tweaking the replacement policy
- PIPP [ISCA 2009]: Lines inserted and promoted in LRU chain depending on the partition they belong to

✓ Dynamic
✓ Maintains associativity
✓ Simple
✗ Few coarse-grain partitions
✗ Weak isolation
✗ Sacrifices replacement policy



Comparison of Schemes

	Way partitioning	PIPP	Reconfig. caches	Page coloring	Vantage
Scalable & fine-grain	✘	✘	✘	✘	✓
Strict isolation	✓	✘	✓	✓	✓
Dynamic	✓	✓	✘	✘	✓
Maintains assoc.	✘	✓	✓	✓	✓
Indep. of repl. policy	✓	✘	✓	✓	✓
Simple	✓	✓	✘	✓	✓
Partitions whole cache	✓	✓	✓	✓	✘ (most)

Outline

- Introduction
- Vantage Cache Partitioning
- Evaluation

Vantage Design Overview

1. Use a highly-associative cache (e.g. a zcache)

2. Logically divide cache in managed and unmanaged regions

3. Logically partition the managed region
   - Leverage unmanaged region to allow many partitions with minimal interference

Analytical Guarantees

- Vantage can be completely characterized using analytical models

$$E_1,\ldots,E_R \sim i.i.d.\ \ U[0,1]$$

$$A = \max\{E_1,\ldots,E_R\}$$

$$F_A(x) = P(A \le x) = x^R, x \in [0,1]$$

$$A_{mgd} = \frac{1}{R \cdot m}$$

$$A_i = \frac{C_i}{\sum_{k=1}^{P} C_k} \frac{\sum_{k=1}^{P} S_k}{S_i} \frac{1}{R \cdot m}$$

$$\sum_{i=0}^{P} \Delta S_i \le \frac{1}{A_{\max}} \frac{1}{R}$$

…

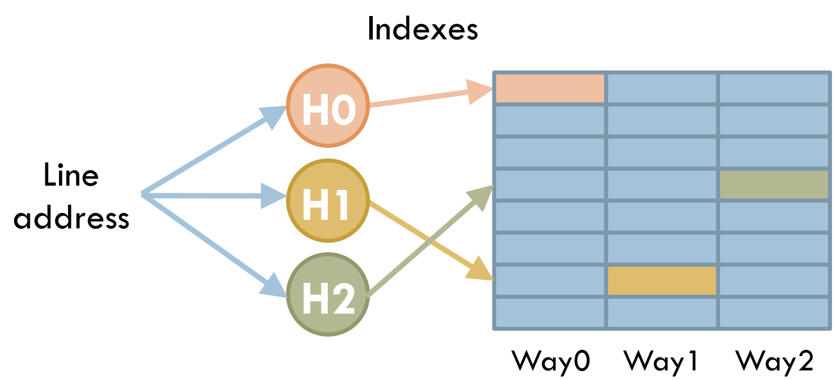
✓ We can *prove* that strict guarantees are kept on partition sizes and interference independently of workload

✗ The paper has too much math to describe it here

- We now focus on the intuition behind the math

ZCache [MICRO 2010]

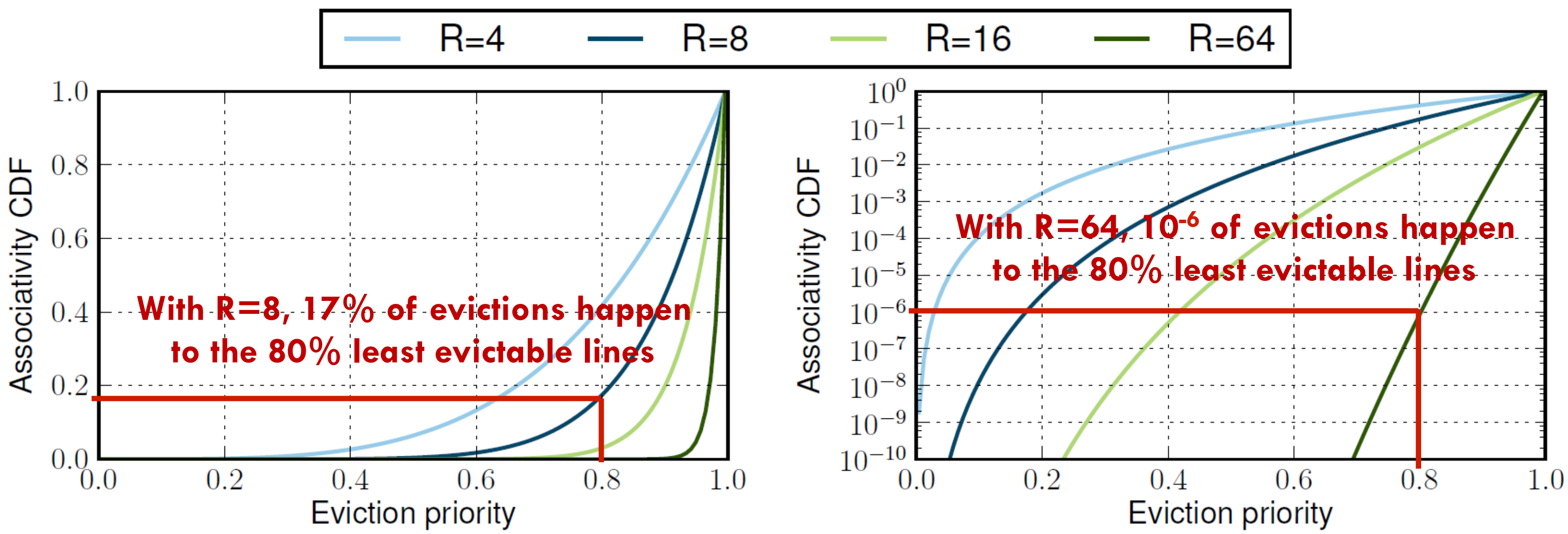
- A highly-associative cache with a low number of ways
  - Hits take a single lookup
  - In a miss, replacement process provides many replacement candidates



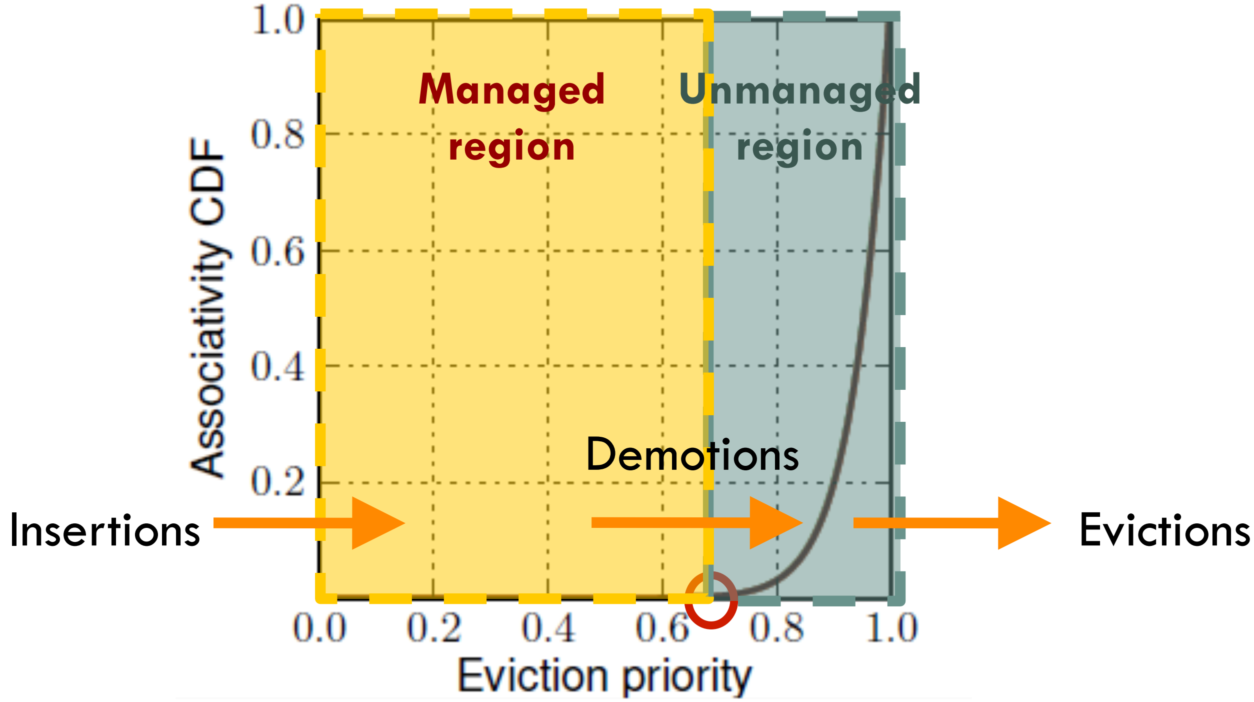
  - Provides cheap high associativity (e.g. associativity equivalent to 64 ways with a 4-way cache)
  - Achieves analytical guarantees on associativity

Analytical Associativity Guarantees

- Eviction priority: Rank of a line given by the replacement policy (e.g. LRU), normalized to [0,1]
  - Higher is better to evict (e.g. LRU line has 1.0 priority, MRU has 0.0)
- Associativity distribution: Probability distribution of the eviction priorities of evicted lines
- In a zcache, associativity distribution depends only on the number of replacement candidates (R)
  - Independent of ways, workload and replacement policy

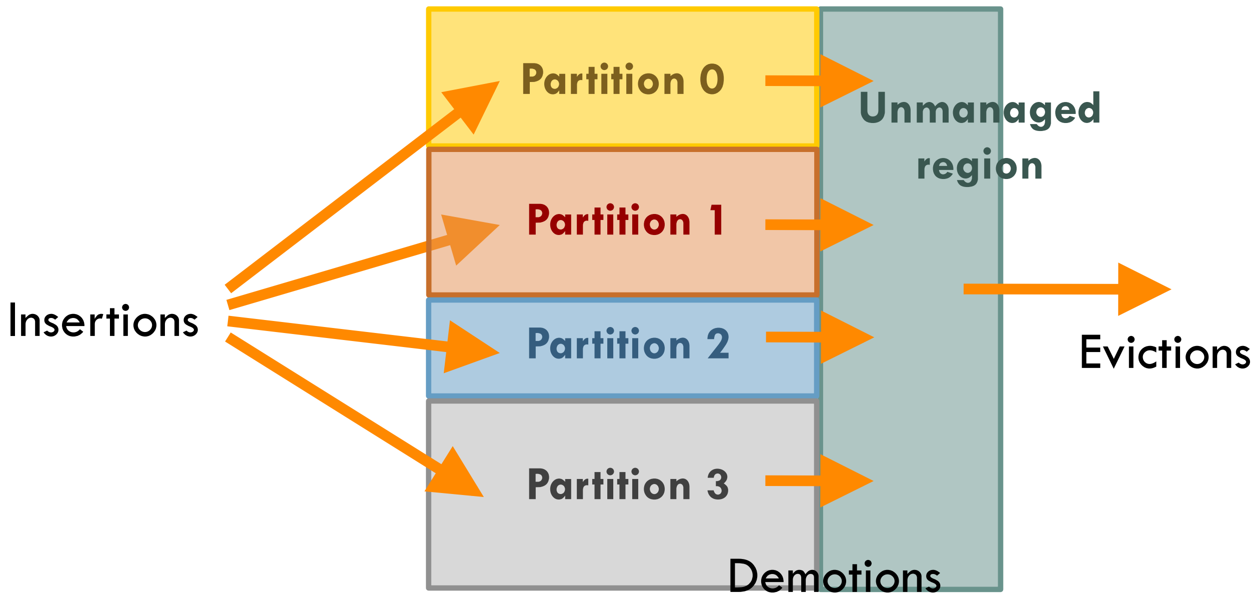


Managed-Unmanaged Region Division



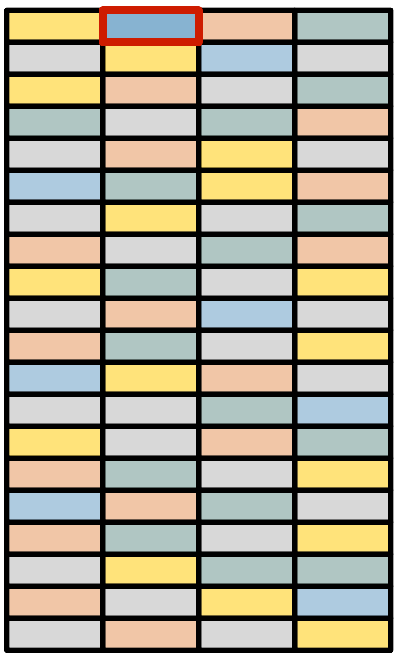
- Logical division (tag each block as managed/unmanaged)
- Unmanaged region large enough to absorb most evictions
- Unmanaged region still used, acts as victim cache (demotion → eviction)
- Single partition with guaranteed size

Multiple Partitions in Managed Region

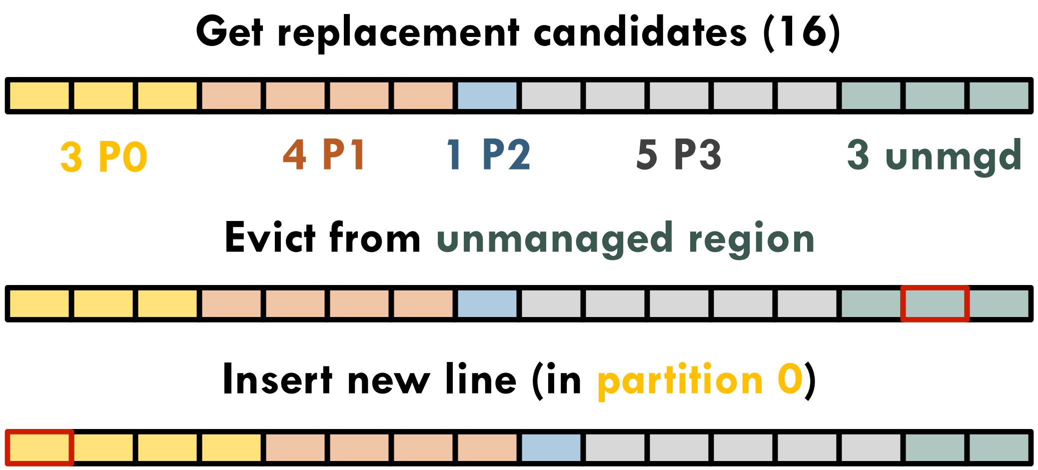


- P partitions + unmanaged region
- Each line is tagged with its partition ID (0 to P-1)
- On each miss:
  - Insert new line into corresponding partition
  - Demote one of the candidates to unmanaged region
  - Evict from the unmanaged region

Churn-Based Management



1. Access A (partition 2) → HIT
2. Access B (partition 0) → MISS

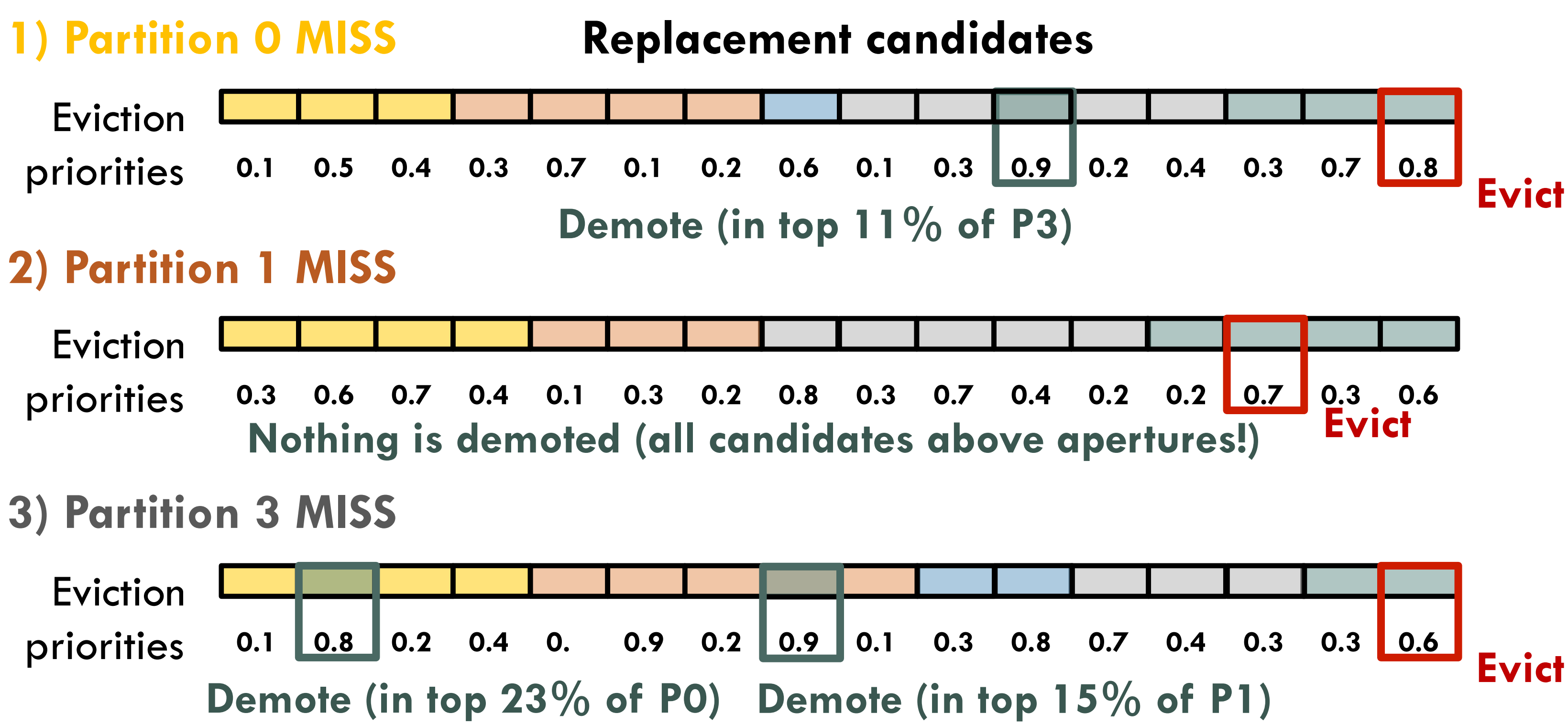


- Problem: always demoting from inserting partition does not scale
  - Could demote from partition 0, but only 3 candidates
  - With many partitions, might not even see a candidate from inserting partition!
- Instead, demote to match insertion rate (*churn*) and demotion rate

Churn-Based Management

- **Aperture**: Portion of candidates to demote from each partition

	Partition 0	Partition 1	Partition 2	Partition 3
Apertures	23%	15%	12%	11%



Managing Apertures

- Set each aperture so that partition churn = demotion rate
  - Instantaneous partition sizes vary a bit, but sizes are maintained
  - Unmanaged region prevents interference
- Each partition requires aperture proportional to its churn/size ratio
  - Higher churn ↔ More frequent insertions (and demotions!)
  - Larger size ↔ We see lines from that partition more often
- Partition aperture determines partition associativity
  - Higher aperture ↔ less selective ↔ lower associativity

Stability

- In partitions with high churn/size, controlling aperture is sometimes not enough to keep size
  - e.g. 1-line partition that misses all the time
  - To keep high associativity, set a maximum aperture *Amax* (e.g. 40%)
  - If a partition needs Ai > Amax, *we just let it grow*

- Key result: Regardless of the number of partitions that need to grow beyond their target, the worst-case total growth over their target sizes is bounded and small!

$$\frac{1}{A_{\max}}\frac{1}{R}$$

  - 5% of the cache with R=52, Amax=0.4
  - Simply size the unmanaged region with that much extra slack
  - Stability and scalability are guaranteed

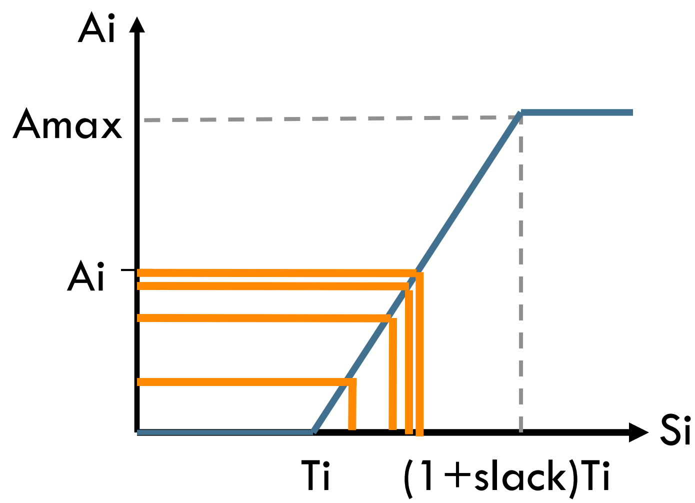
A Simple Vantage Controller

- Directly implementing these techniques is impractical
  - Must constantly compute apertures, estimate churns
  - Need to know eviction priorities of every block

- Solution: Use negative feedback loops to derive apertures and the lines below aperture
  - Practical implementation
  - Maintains analytical guarantees

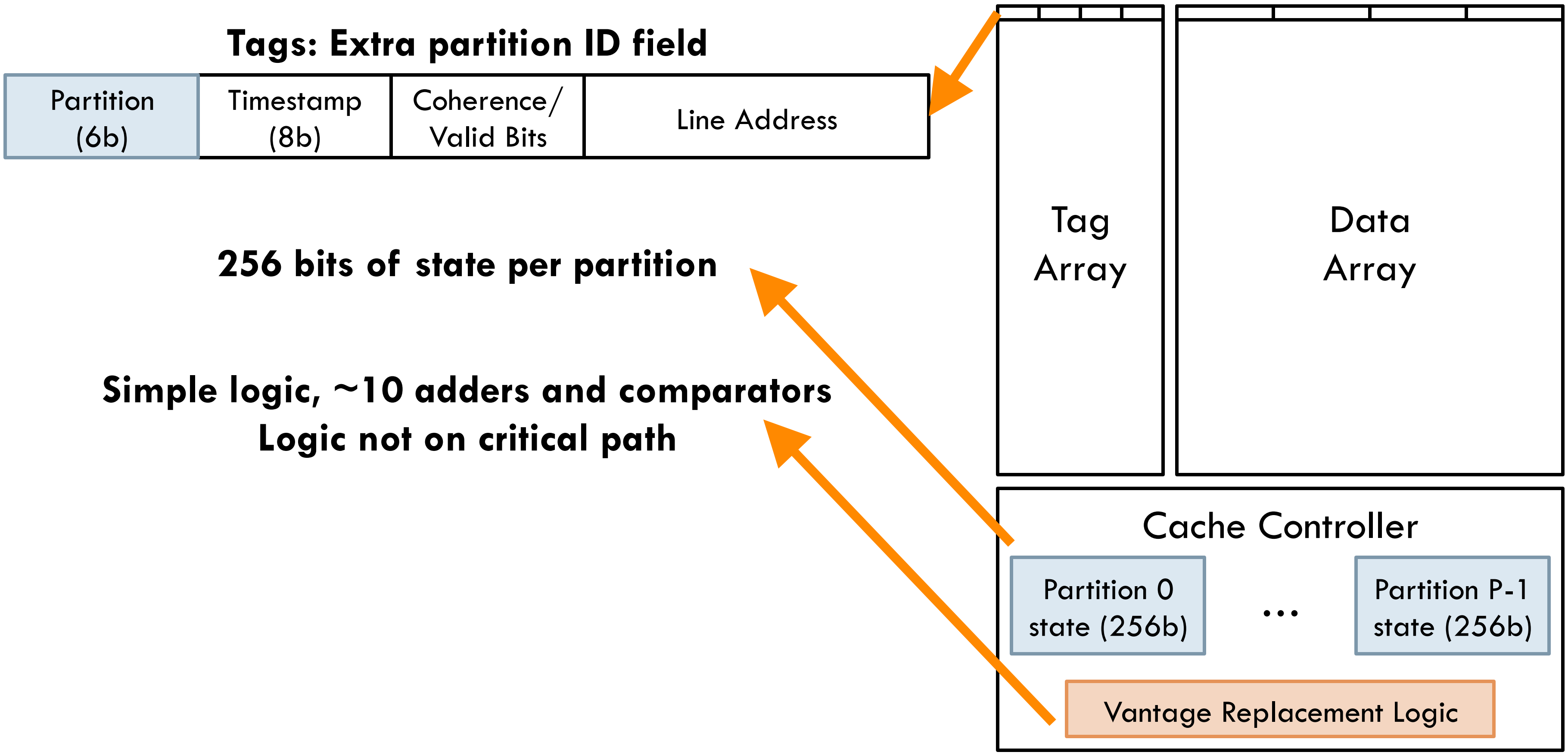
Feedback-Based Aperture Control

- Adjust aperture by letting partition size (Si) grow over its target (Ti):



- Need small extra space in unmanaged region
  - e.g. 0.5% of the cache with R=52, Amax=0.4, slack=10%

Implementation Costs



- See paper for detailed implementation

Vantage Summary

- Use a cache with associativity guarantees
- Maintain an unmanaged region
- Match insertion and demotion rates in each partition
  - Partitions help each other evict lines → maintain associativity
  - Unmanaged region guarantees isolation and stability
- Use negative feedback to simplify implementation

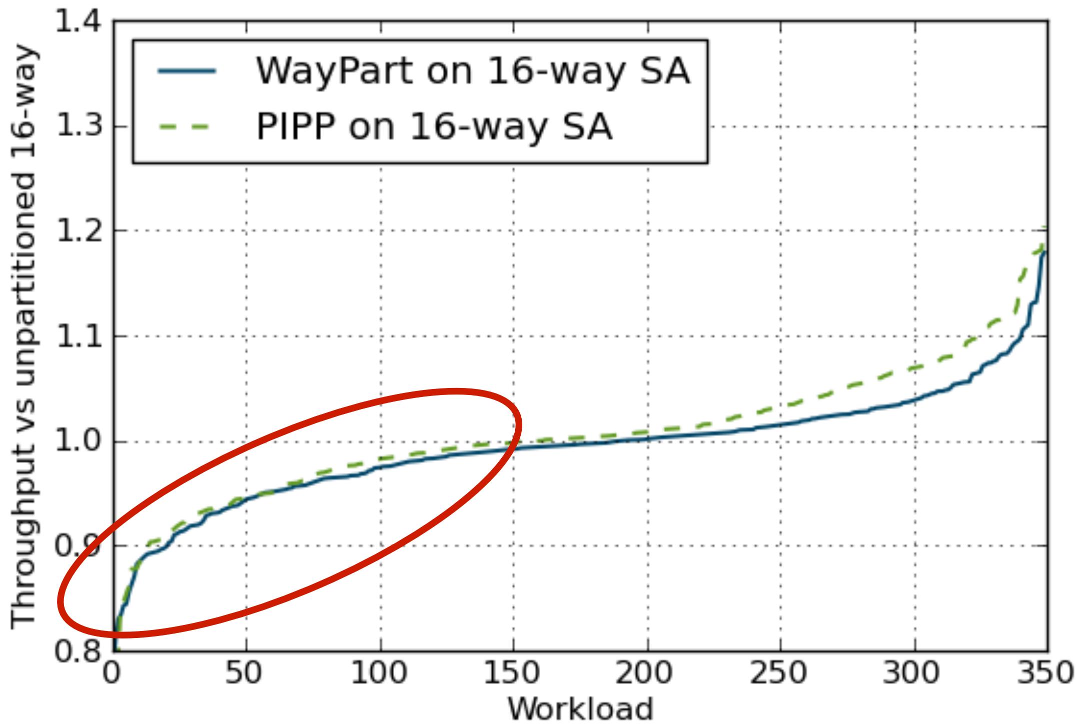
Outline

- Introduction
- Vantage Cache Partitioning
- Evaluation

Methodology

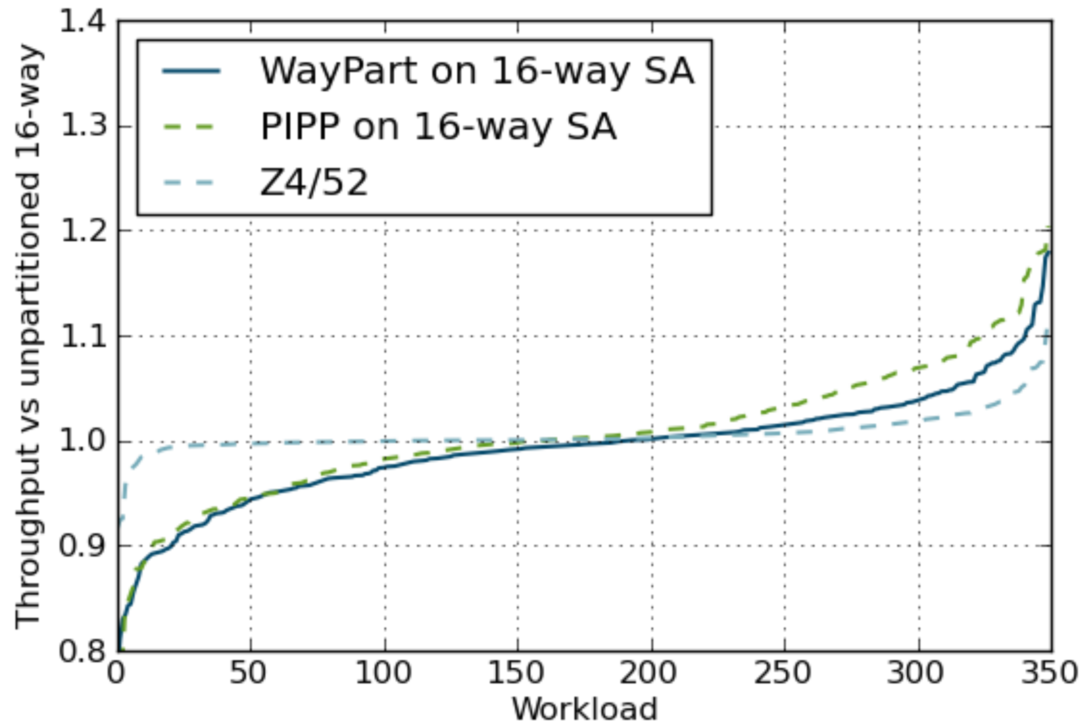
- Simulations of small (4-core) and large (32-core) systems
  - Private L1s, shared non-inclusive L2, 1 partition/core
- Partitioning policy: Utility-based partitioning [ISCA'06]
  - Assign more space to threads that can use it better
- Partitioning schemes: Way-partitioning, PIPP, Vantage
- Workloads: 350 multiprogrammed mixes from SPECCPU2006 (full suite)

Small-Scale: 4 cores, 4 partitions



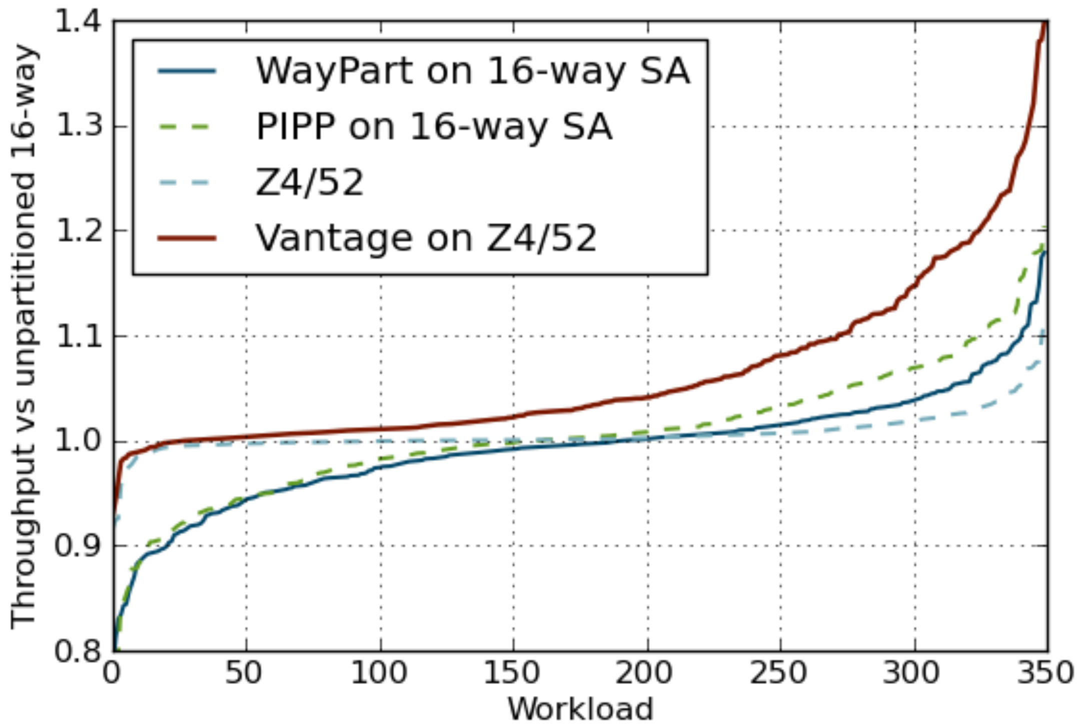
- Each line shows throughput improvement versus an unpartitioned 16-way set-associative cache
- Way-partitioning and PIPP degrade throughput for 45% of workloads

Small-Scale: 4 cores, 4 partitions



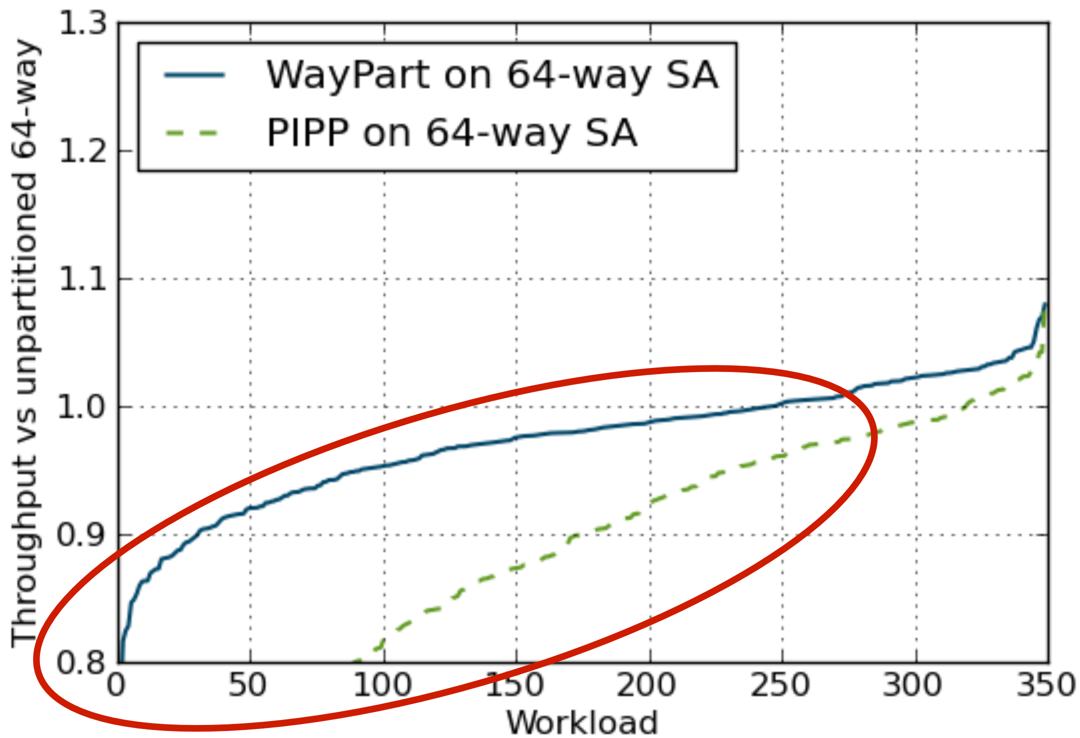
- Vantage works on best on zcaches
- We use Vantage on a 4-way zcache with R=52 replacement candidates

Small-Scale: 4 cores, 4 partitions



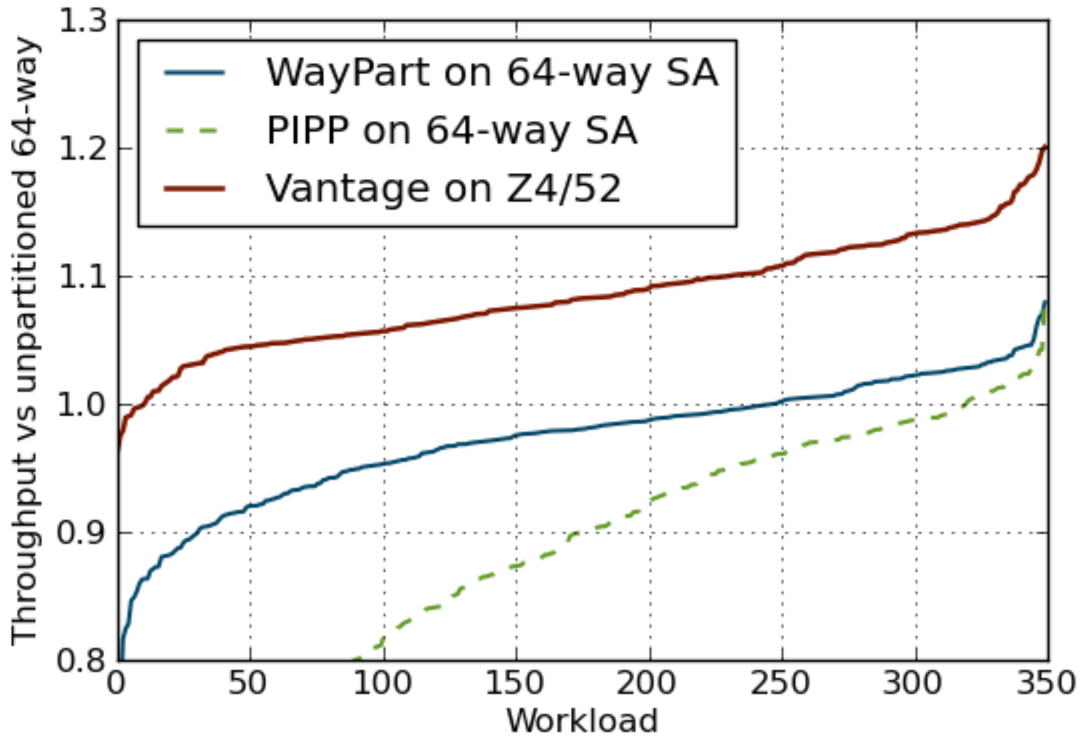
- Vantage improves throughput for most workloads
- 6.2% throughput improvement (gmean), 26% for the 50 most memory-intensive workloads

Large-Scale: 32 cores, 32 partitions



- Way-partitioning and PIPP use a 64-way set-associative cache
- Both degrade throughput for most workloads

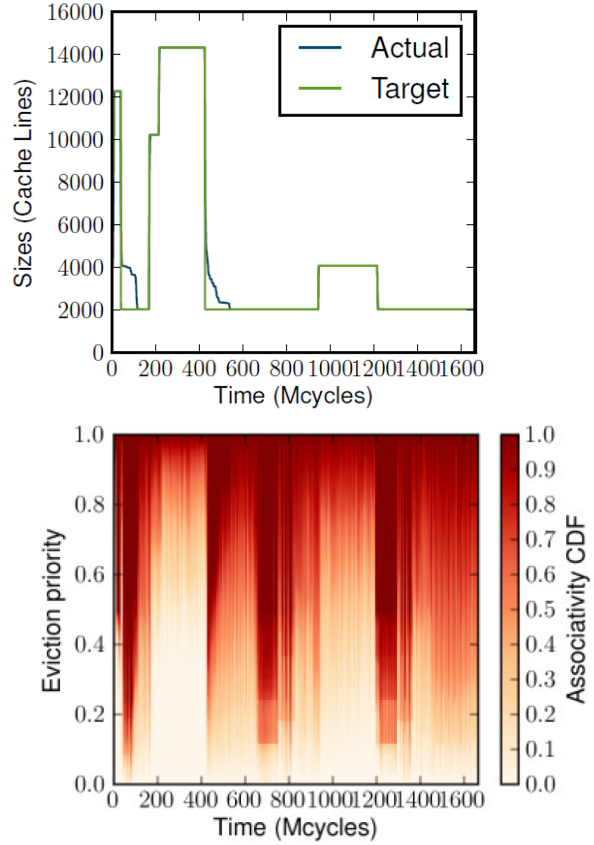
Large-Scale: 32 cores, 32 partitions



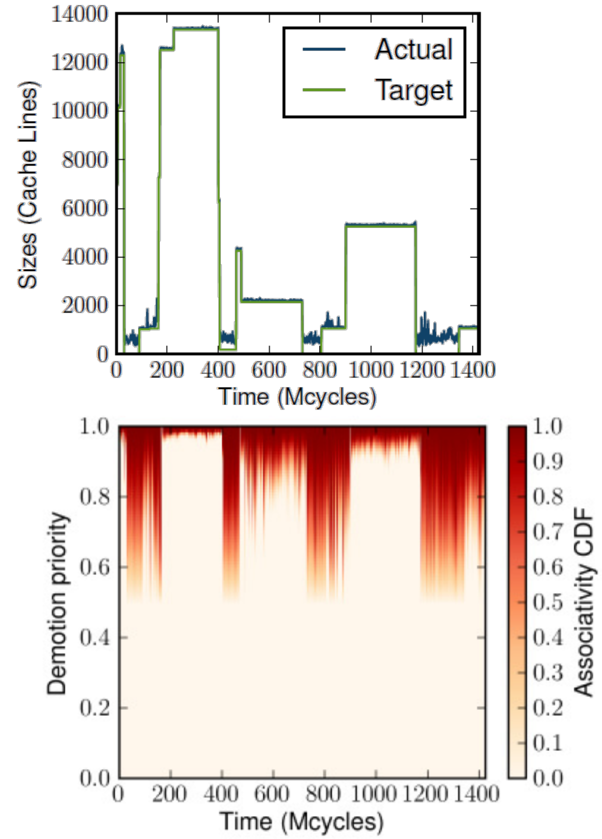
- Vantage uses the same Z4/52 cache as the 4-core system
- Vantage improves throughput for most workloads → scalable

A Closer Look: Sizes & Associativity

Way-partitioning



Vantage



- Vantage maintains strict partition sizes
- Vantage maintains high associativity even in the worst case

Additional Results (see paper)

- Vantage maintains strict control of partition sizes
- Vantage maintains high associativity
- Unmanaged region size vs isolation tradeoff
  - ~5% unmanaged region and moderate isolation
  - ~20% unmanaged region and strict isolation
- Validation of analytical models
- Vantage on set-associative caches
  - Loses analytical guarantees, but outperforms other schemes
- Vantage with other replacement policies (RRIP)

Conclusions

- Vantage enables cache partitioning for many-cores
  - Tens to hundreds of fine-grain partitions
  - High associativity per partition
  - Strict isolation among partitions
  - Derived from analytical models, bounds independent of number of partitions and cache ways
  - Simple to implement

THANK YOU FOR YOUR ATTENTION

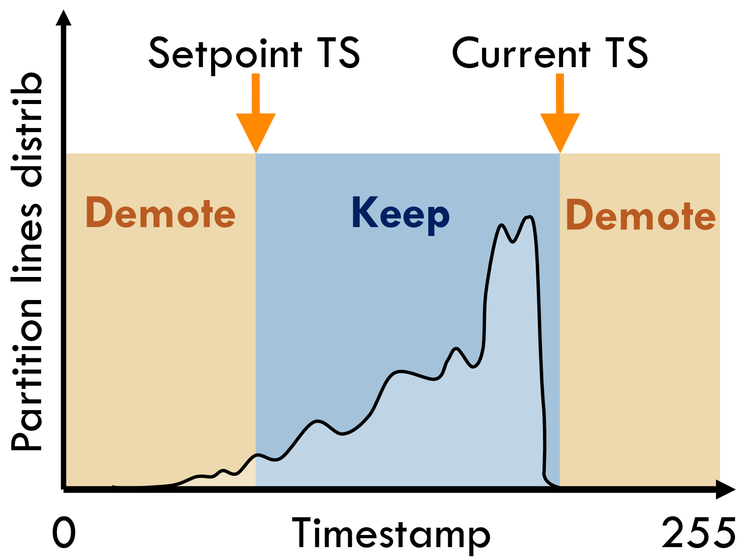
QUESTIONS?

Backup: Associativity Guarantees

- Why does zcache produce uniform random replacement candidates, independently of access pattern?
- ZCache hashing and replacement scheme eliminates spatial locality
- Evictions have negligible temporal locality w.r.t. cache
  - Evictions to the same block are widely separated in time
  - NOTE: Invalidations (e.g. coherence) are not evictions
- No locality → uniform random

Backup: Setpoint-Based Demotions

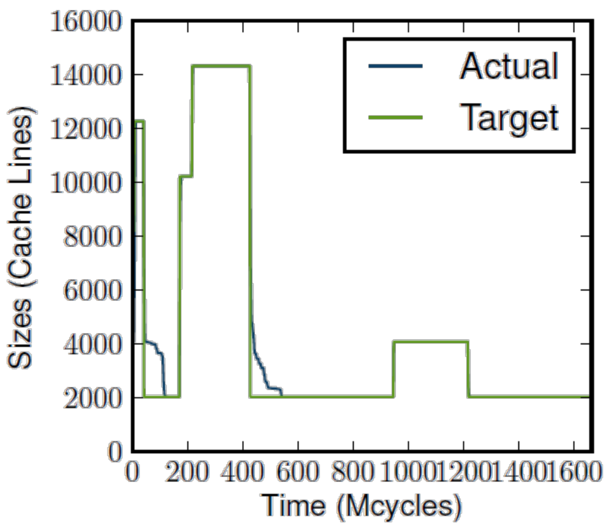
- Derive portion of lines below aperture without tracking eviction priorities
- Coarse-grain timestamp LRU replacement
  - Tag each block with an 8-bit LRU per-partition timestamp
  - Increment timestamp every Si/16 accesses



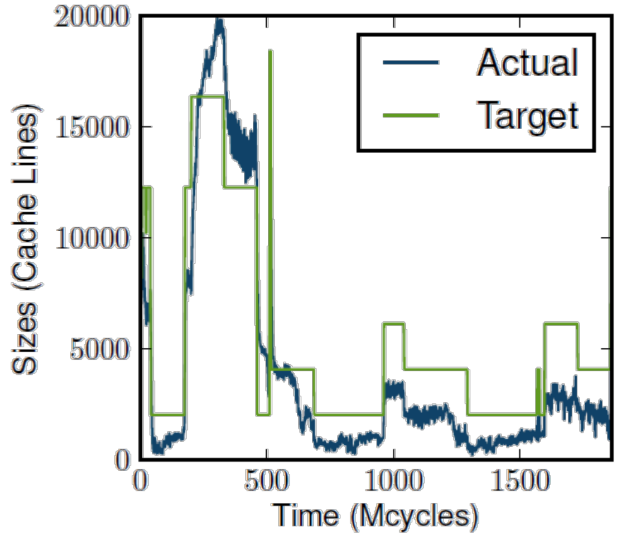
- Demote every candidate below the setpoint timestamp
- Adjust setpoint using negative feedback

A Closer Look: Partition Sizes

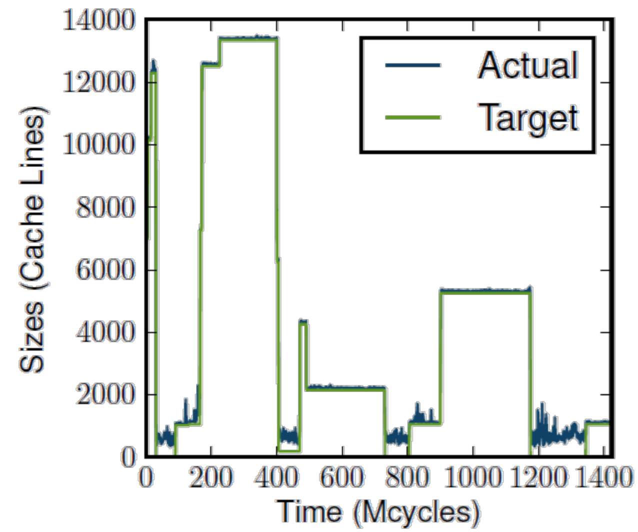
Way-partitioning



PIPP

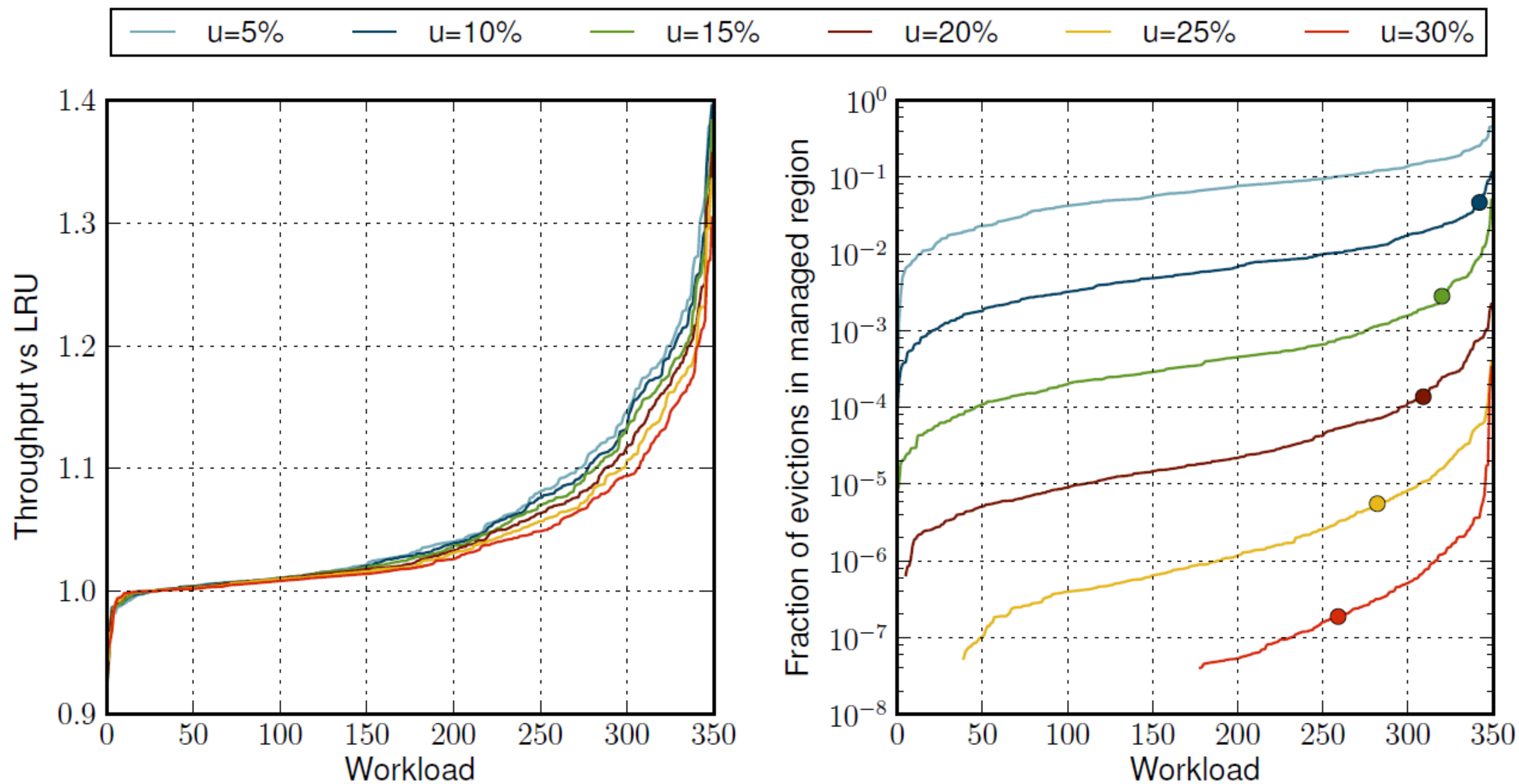


Vantage



Way-partitioning	PIPP	Vantage
✘ Coarse-grain partitions	✘ Coarse-grain partitions	✓ Fine-grain partitions
✓Strict size	✘ Approximate size	✓ Strict size
✘ Slow convergence	✘ No convergence	✓ Fast convergence

Unmanaged Size vs Isolation Trade-off



- A larger unmanaged region reduces UCP perfomance slightly, but gives excellent isolation
- Simulations match analytical models
- See paper for additional results (Vantage on set-associative caches, other replacement policies, etc.)