

A Comparison of Capacity Management Schemes for Shared CMP Caches

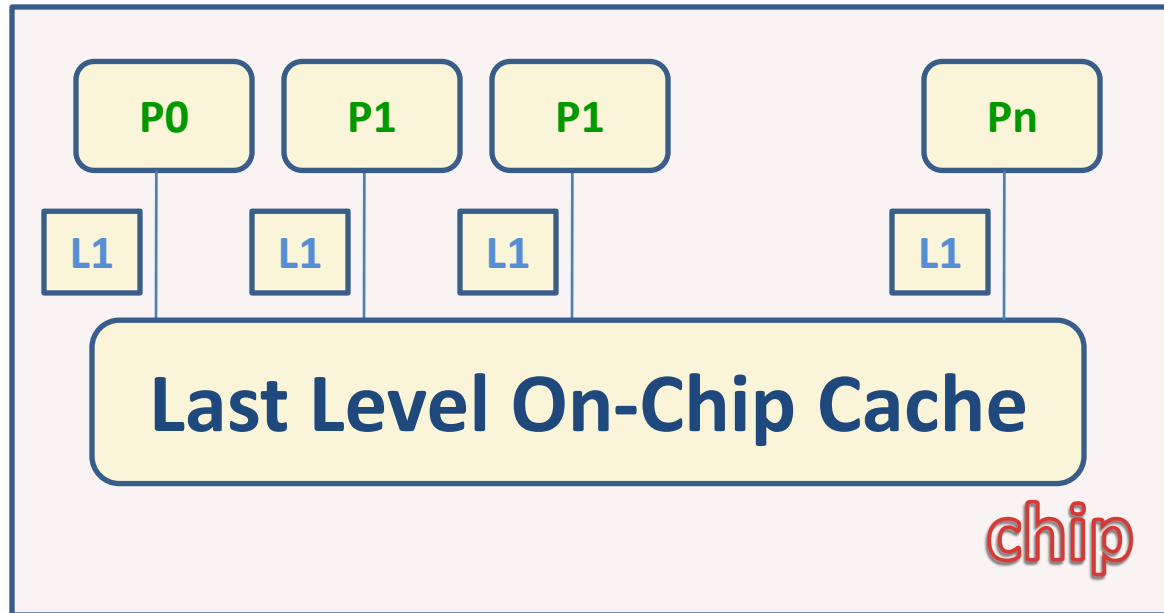
Carole-Jean Wu and Margaret Martonosi

Princeton University

7th Annual WDDD

6/22/2008

Motivation

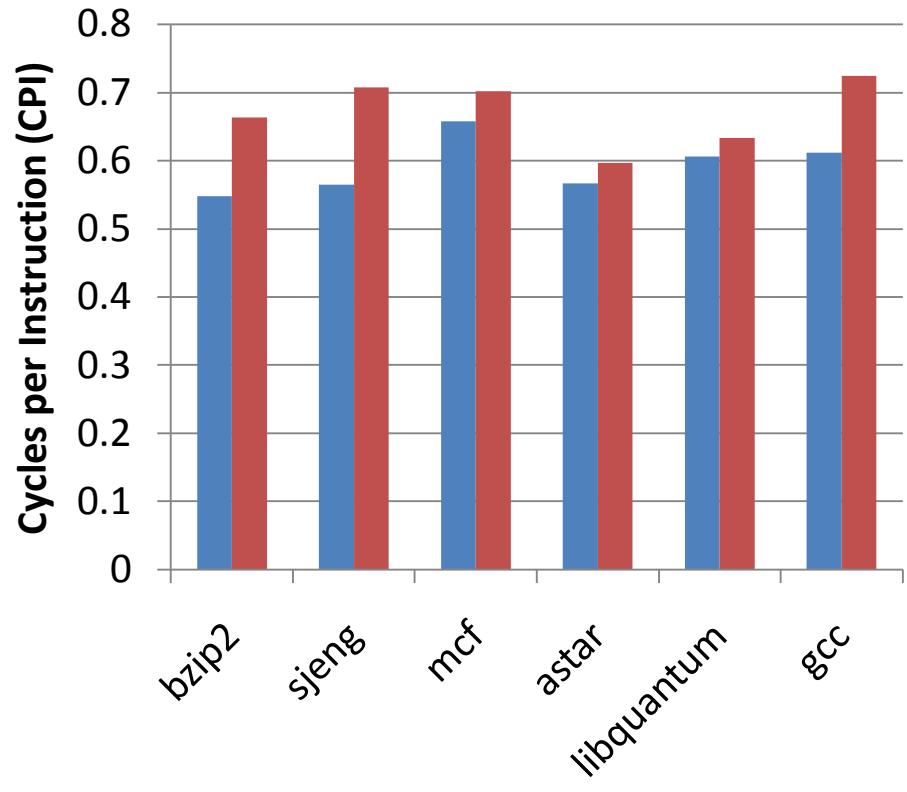
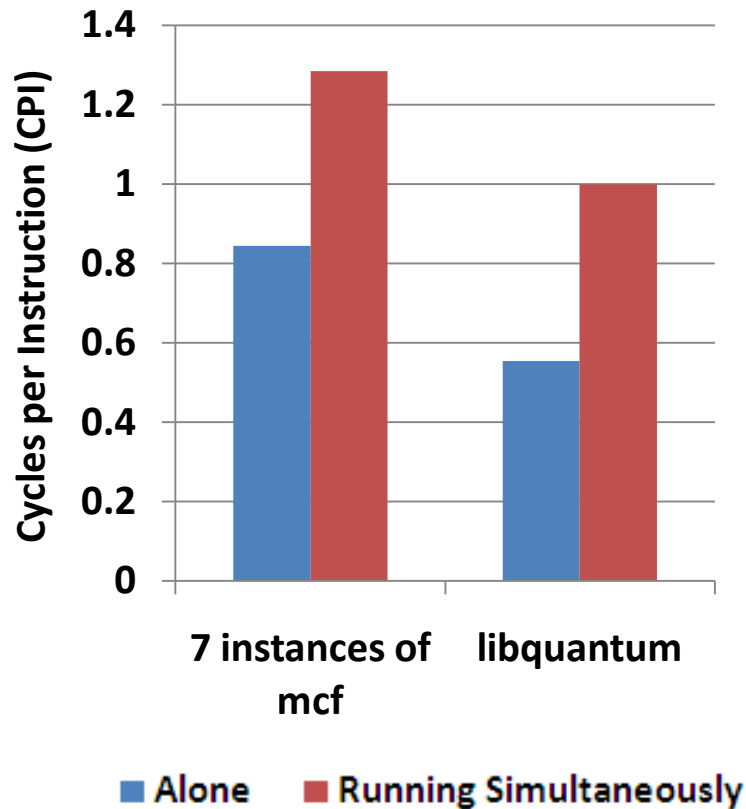


Heterogeneous Workloads

- Web servers
- Video streaming
- Graphic intensive
- Scientific
- Data mining
- Security scan
- File/Data Base

- Core counts scaling up
 - Shared cache becomes highly contested
- LRU replacement is not enough
 - No distinction between **process priority** and applications' **memory needs**

When there is no capacity management...



Performance is severely degraded among all concurrently running applications

This paper

- Offer an extensive and detailed study of shared resource management schemes
 - Way-partitioned management [D. Chiou, MIT PhD Thesis, '99]
 - Decay-based management [Petoumenos et al., IEEE Workload Characterization, '06]
- Demonstrate potential benefits of each management scheme
 - Cache space utilization
 - Performance
 - Flexibility and scalability

Outline

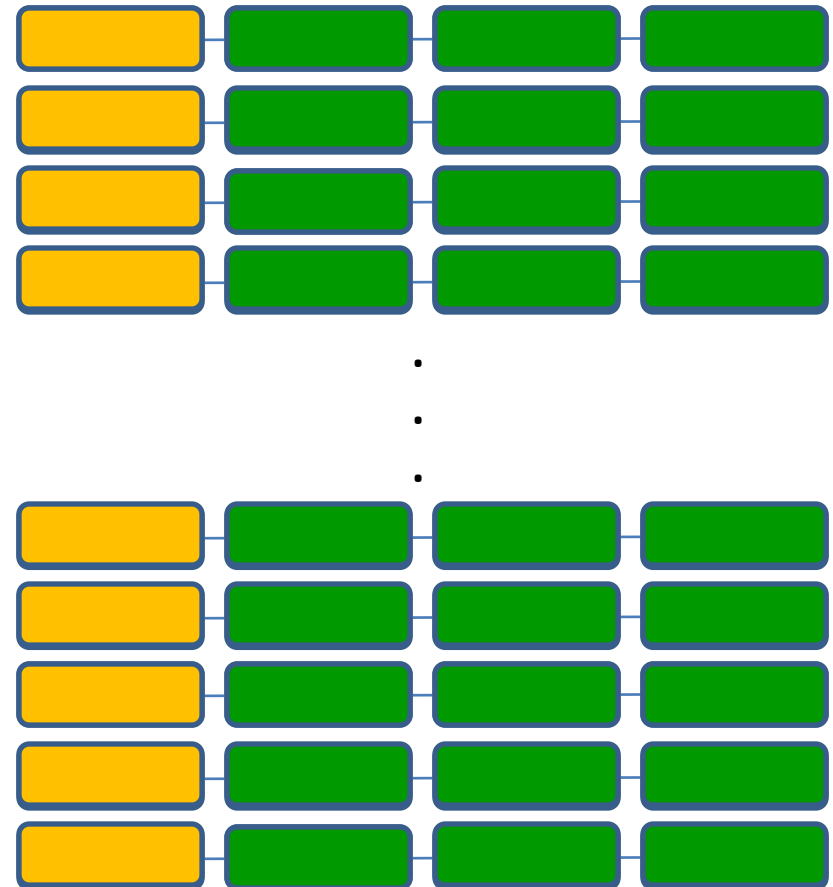
- ~~Motivation~~
- Shared Cache Capacity Management
- Experimental Setup and Evaluation
- Related Work
- Conclusion

Shared Cache Capacity Management

- Apportioning shared cache resources among multiple processor cores
 - Way-Partitioned Management [D. Chiou, MIT PhD Thesis, '99]
 - Decay-Based Management [Petoumenos et al., IEEE Workload Characterization, '06]

Way-Partitioned Management

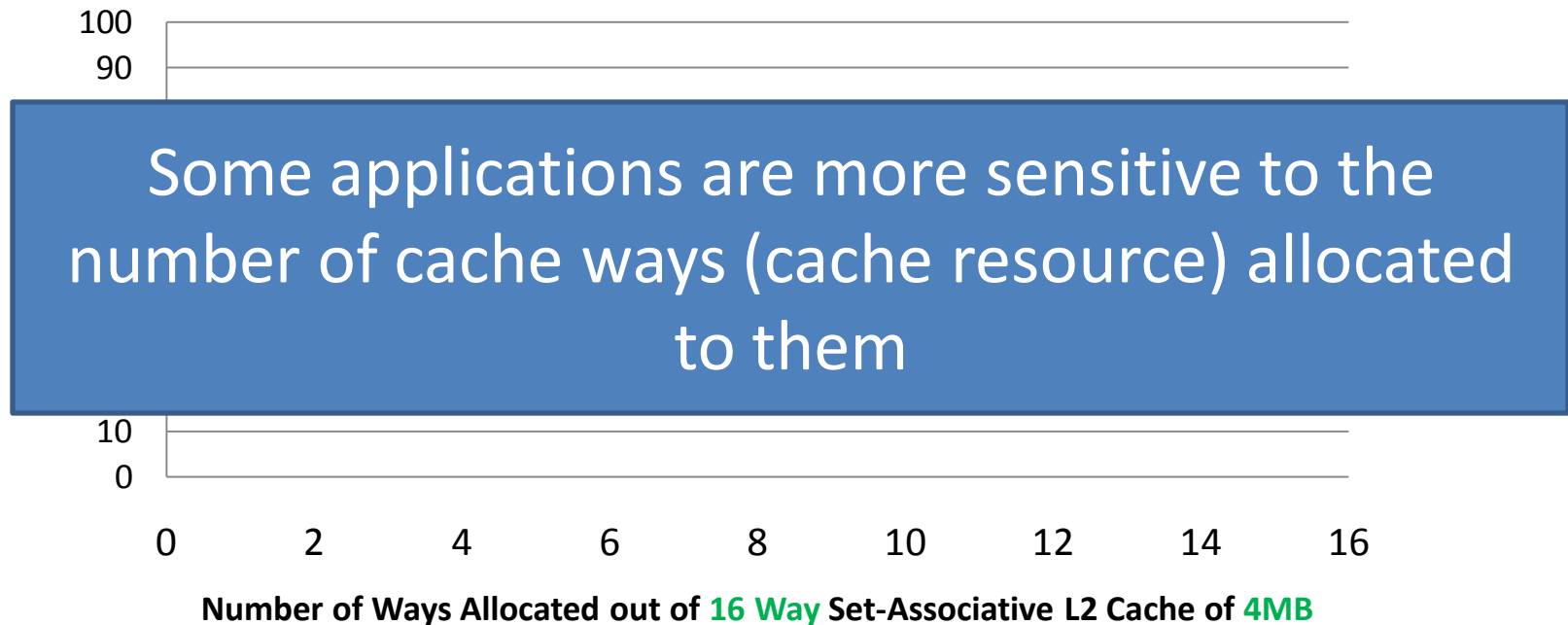
- Statically allocate number of L2 cache ways to processes



4-way set-associative cache

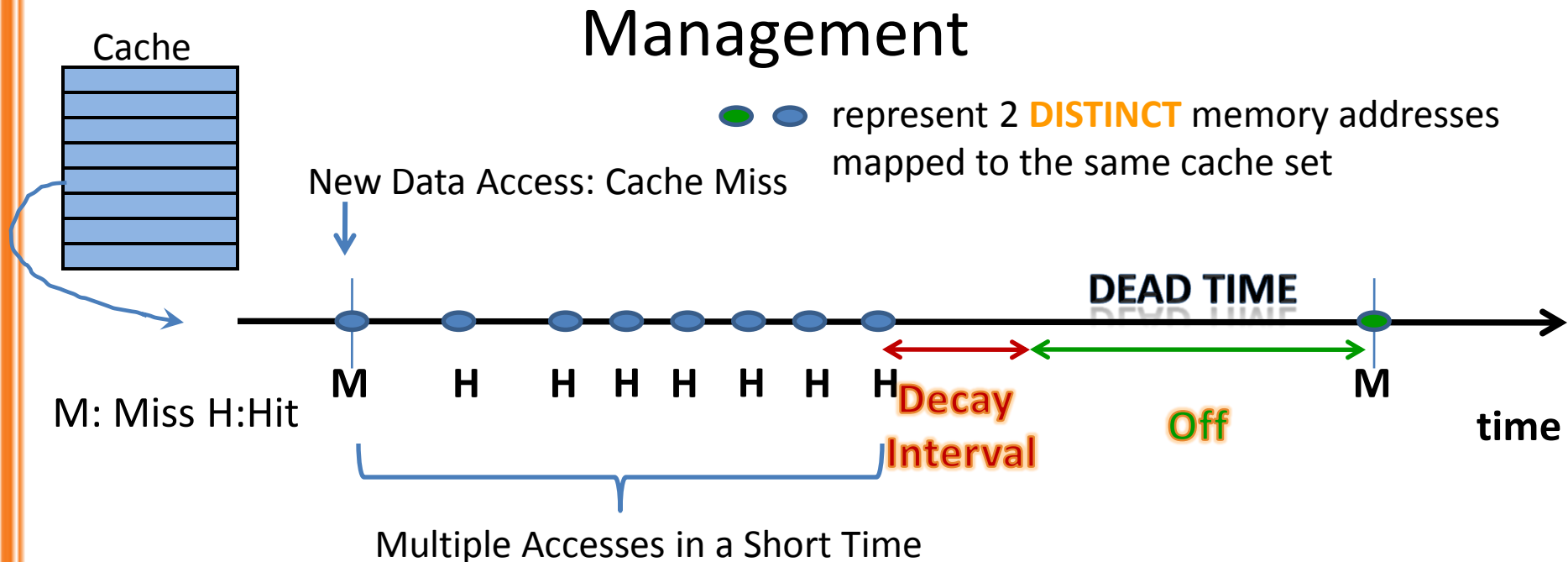
How do applications benefit from cache sizes and set-associativity?

Number of Ways Allocated vs. L2 Miss Rate



- Miss rates are improved as the number of cache ways allocated to them increases.
- Used to achieve performance predictability.

Prior Work: Cache Decay for Leakage Power Management



timer per cache line

- If cache line accessed frequently, maintain power: reset timer w/ every access
- If not accessed for long time switch off V_{dd} : timer = **decay interval** → switch off V_{dd}
- Re-Power a decayed line on an access

Decay for Capacity Management

- Decay counter reaches 0
 - Cache line becomes **an immediate candidate for replacement**, even if NOT LRU
- Set decay counters on per-process basis
 - Long decay interval → high priority process
 - Short decay interval → low priority process, so cache lines are evicted more frequently

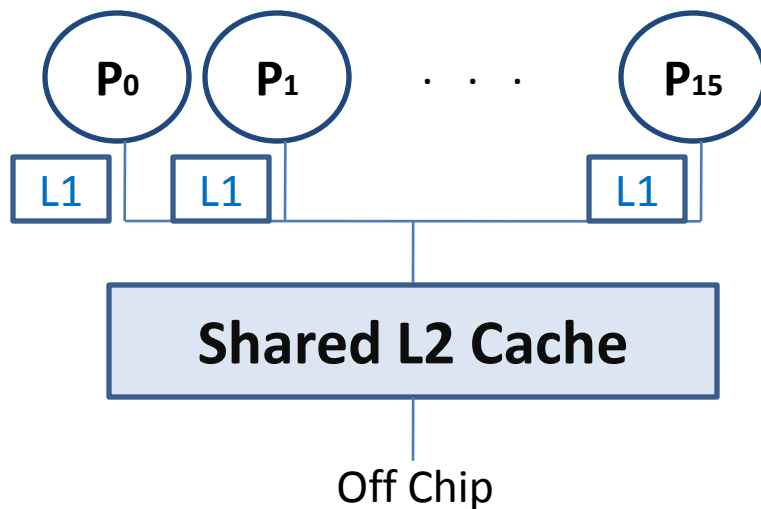
Employ some aspects of priority-based replacement while responding to data temporal locality

Outline

- ~~Motivation~~
- ~~Shared Cache Capacity Management~~
- Experimental Setup and Evaluation
- Related Work
- Conclusion

Experimental Setup

- Simulation Framework
 - GEMS: Full system simulator [Simics+Ruby]
 - 16-core multiprocessor on the Sparc architecture running unmodified Solaris 10 operating system
- Workload
 - SPEC2006 CINT Benchmark Suite [program initialization is included]



Private L1: 32KB each; 4-way; 64B cache line
Shared L2: 4MB; 16-way; 64B cache line

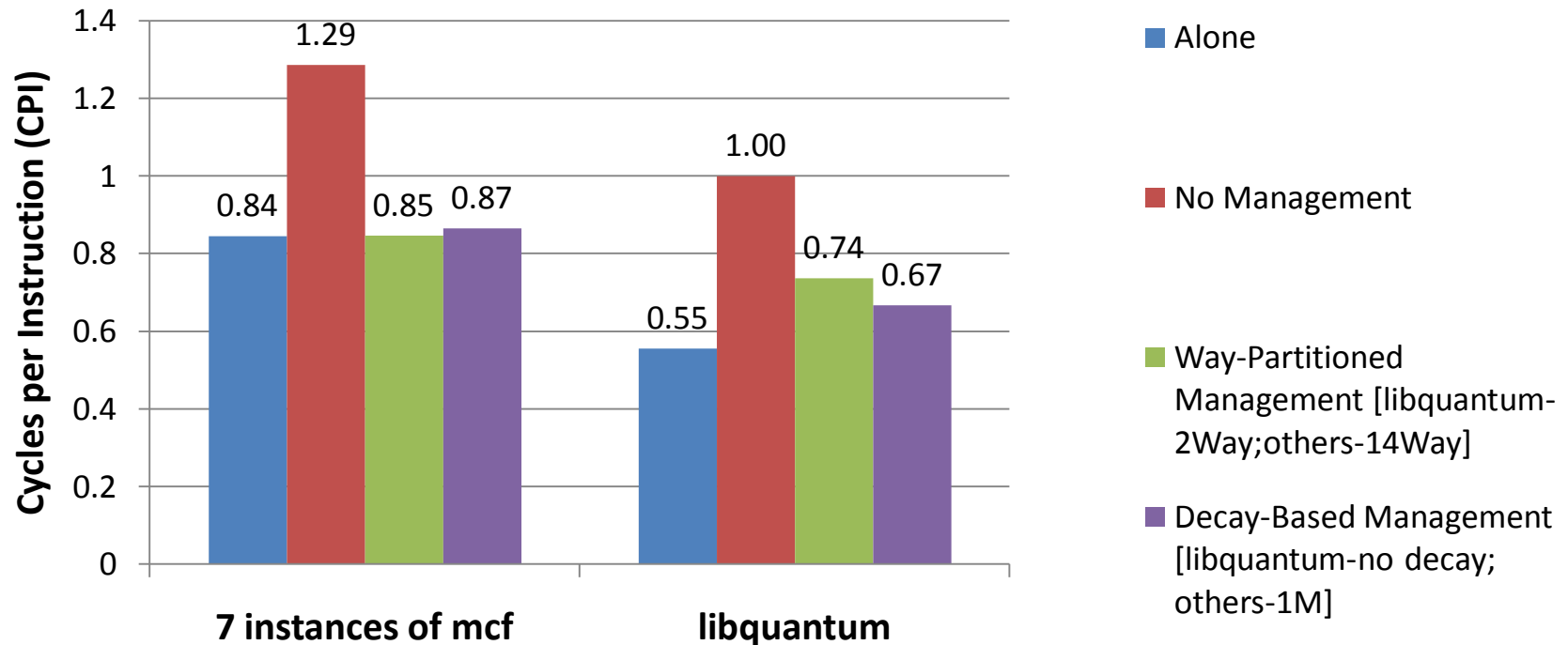
L1 miss latency: 20 cycles
L2 miss latency: 400 cycles

MESI Directory protocol between L1 and L2

Evaluation

- Mechanisms
 - Baseline: No Cache Capacity Management
 - Way-Partitioned Management
 - Decay-Based Management
- Scenarios
 - High contention
 - General Workload 1: Constraining one memory-intensive application
 - General Workload 2: Protecting a high-priority application (refer to the paper)

High Contention Scenario



No management: taking turns evicting each other's cache lines out repetitively

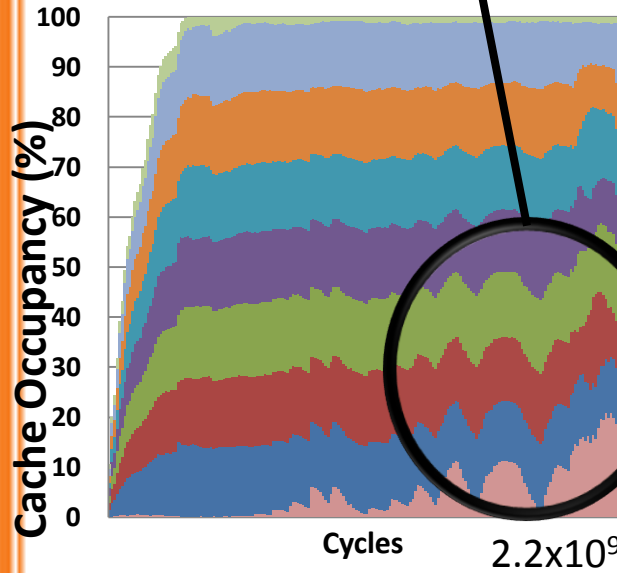
Way-partitioning: performance is improved by 52% and 47%

Decay-based: performance is improved by 50% and 60%

Cache Space Distribution

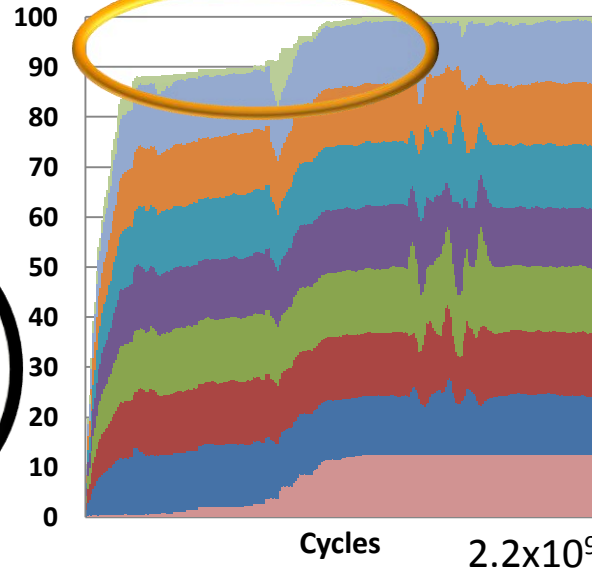
Memory interference

No Management

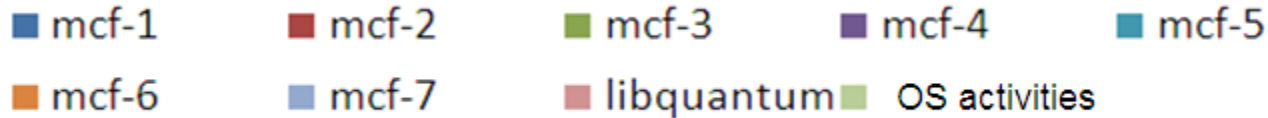
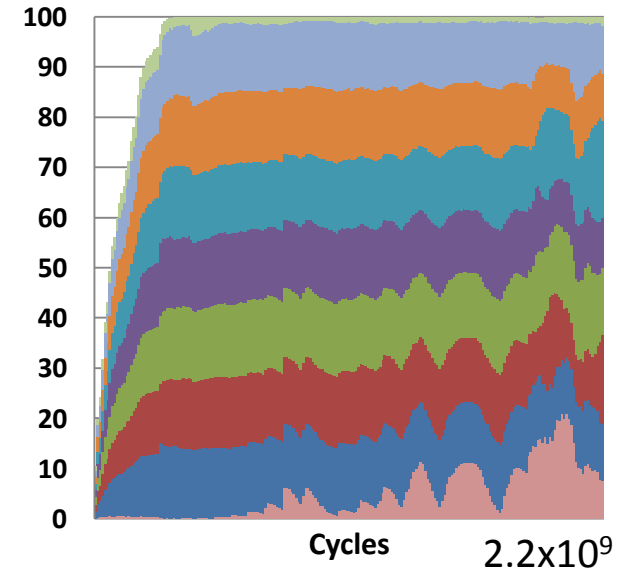


Inefficient space allocation

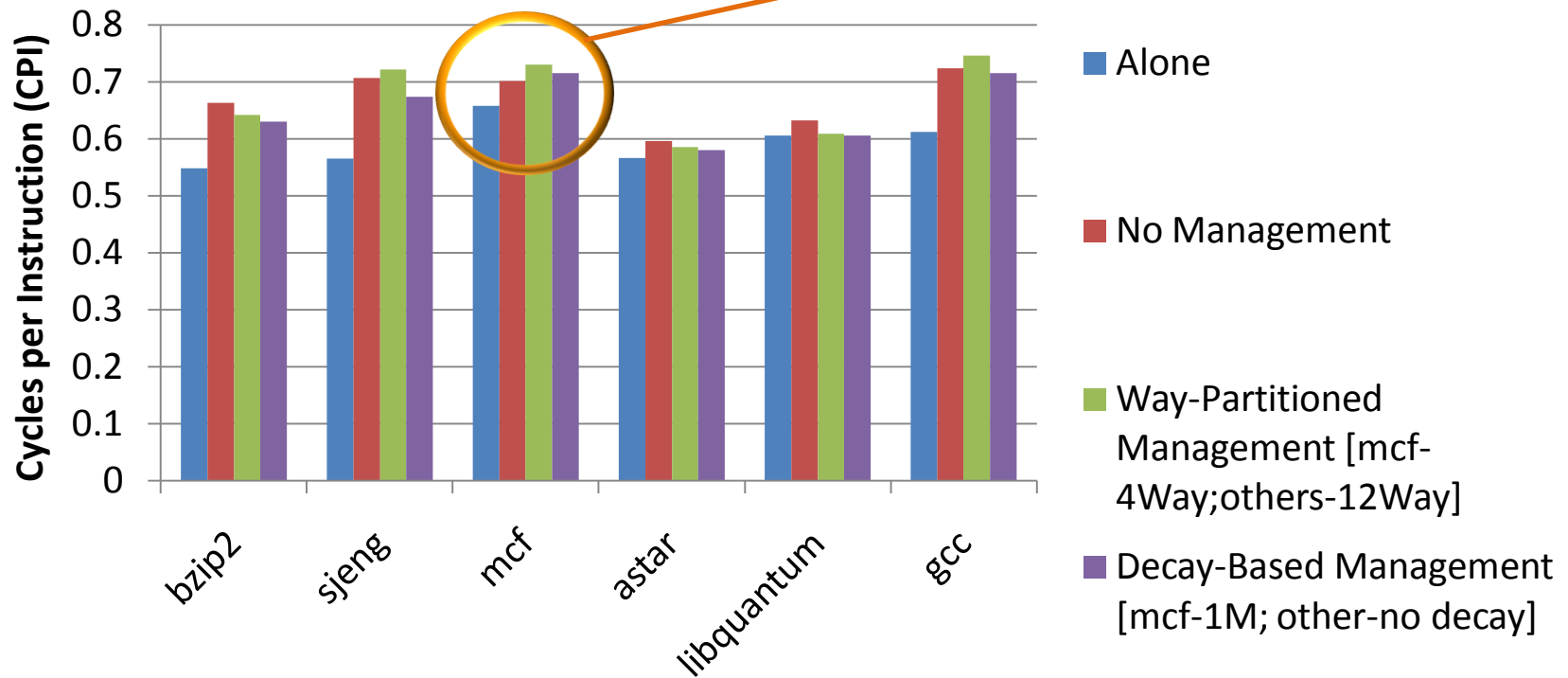
Way-Partitioned Management
(libquantum-2Way; mcf+OS-14Way)



Decay-Based Management
(libquantum-no decay; mcf+OS-10K)



Constraining a Memory-Intensive Application



--Way-partitioning's coarse granularity control **trades off 5% performance** ↓ for mcf with an **average 1% performance improvement** for the rest

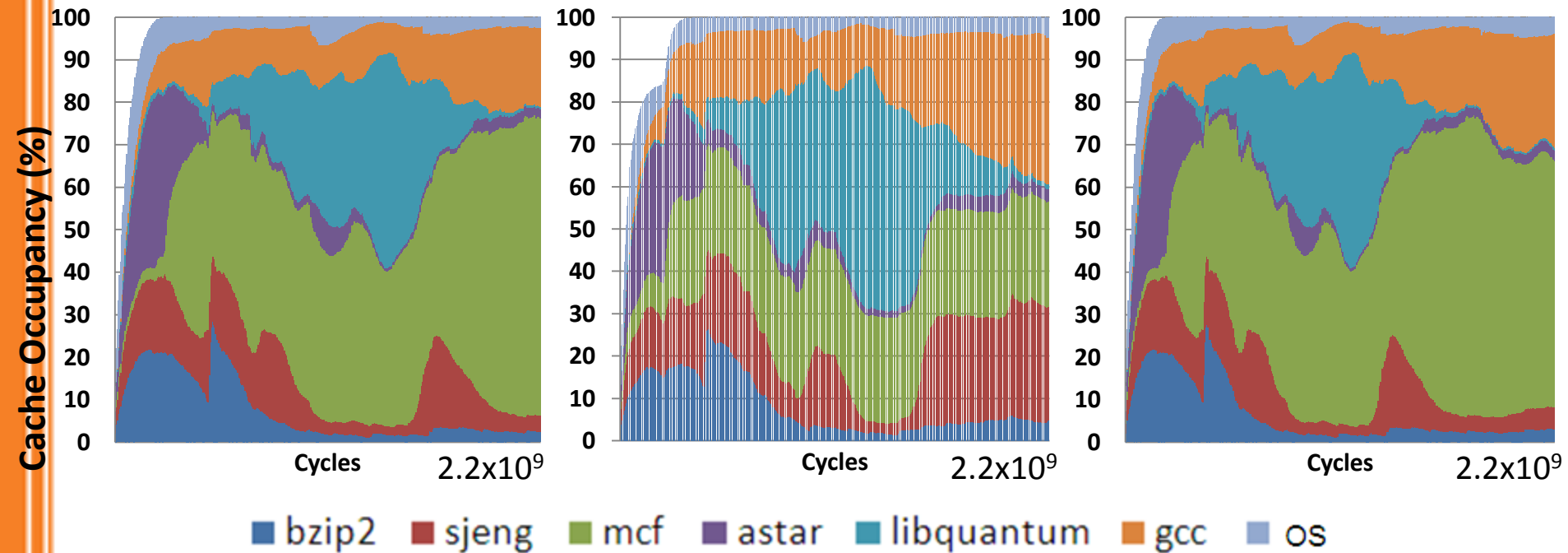
--Decay-based management: **only 2%** ↓ for mcf and **others ↑ 3%** because of its **fine-grained control** and improved ability to exploit **data temporal locality**

Cache Space Distribution

No Management

Way-Partitioned Management
(mcf-4Way; others-12Way)

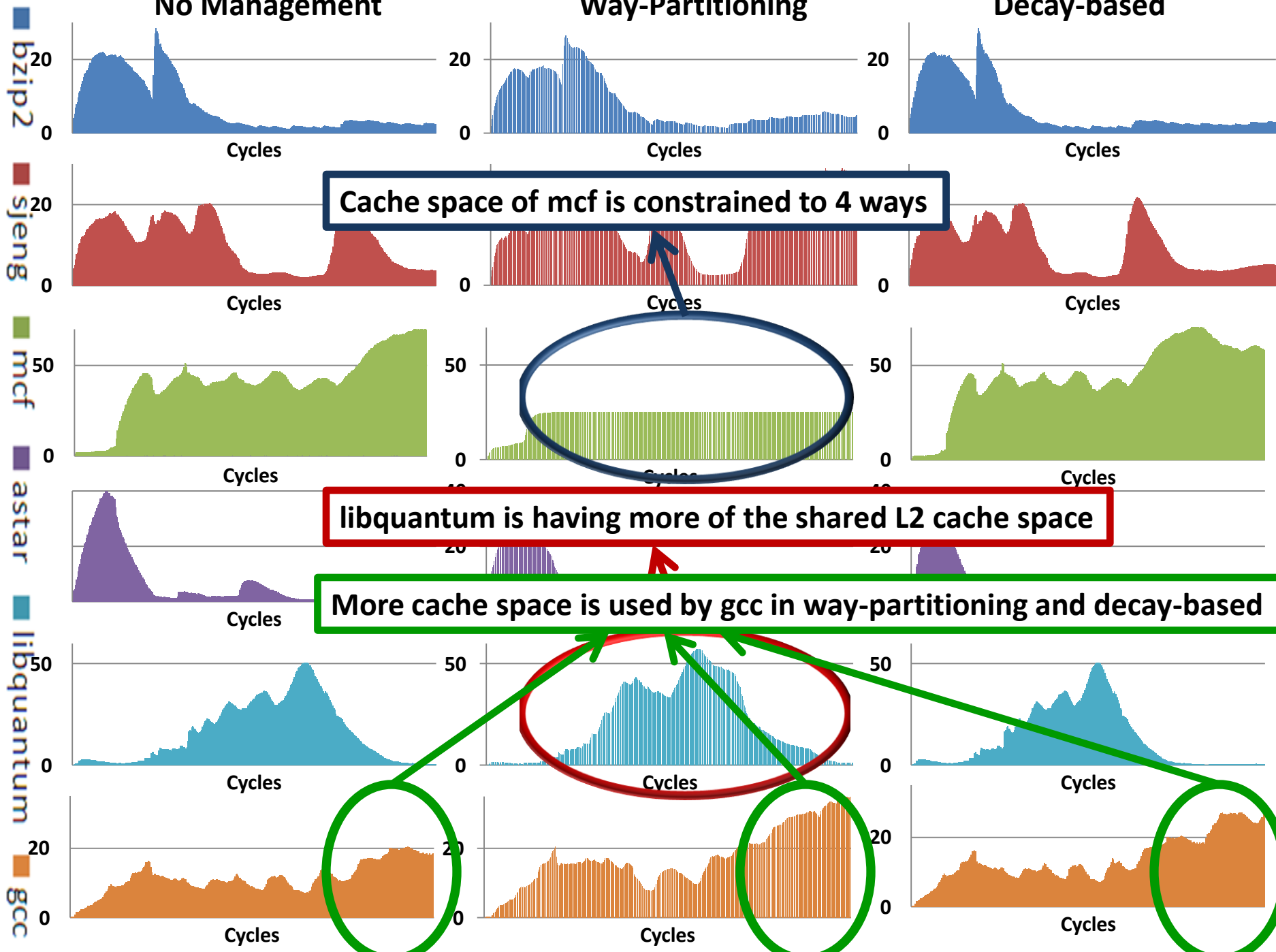
Decay-Based Management
(mcf-1M; others-no decay)



No Management

Way-Partitioning

Decay-based



Outline

- ~~Motivation~~
- ~~Shared Cache Capacity Management~~
- ~~Experimental Setup and Evaluation~~
- Related Work
- Conclusion

- Priority classification and enforcement to achieve differentiable QoS [Iyer, ICS '04]
- Architectural support for optimizing performance of high priority application with minimal performance degradation based on QoS policies [Iyer et al., SIGMETRICS '07]
- Performance metric, such as miss rates, bandwidth usage, IPC, and fairness, to assist resource allocation

Further **cache fairness policies** can be incorporated into both capacity management mechanisms discussed in this work.

performance [Iyer et al., SIGMETRICS '07]

Related Work: Dynamic Cache Capacity Management

- OS distributes equal amount of cache space to all running processes, keeps statistics on the fly, and dynamically adjust cache space distribution [Suh et al., HPCA '02; Kim et al., PACT '04; Qureshi and Patt, MICRO '06]
- Adaptive set pinning to eliminate inter-process misses [Srikantaiah et al., ISCA '08]

To the best of our knowledge, there has not been any prior work based on decay management taking **full system effects into account.**

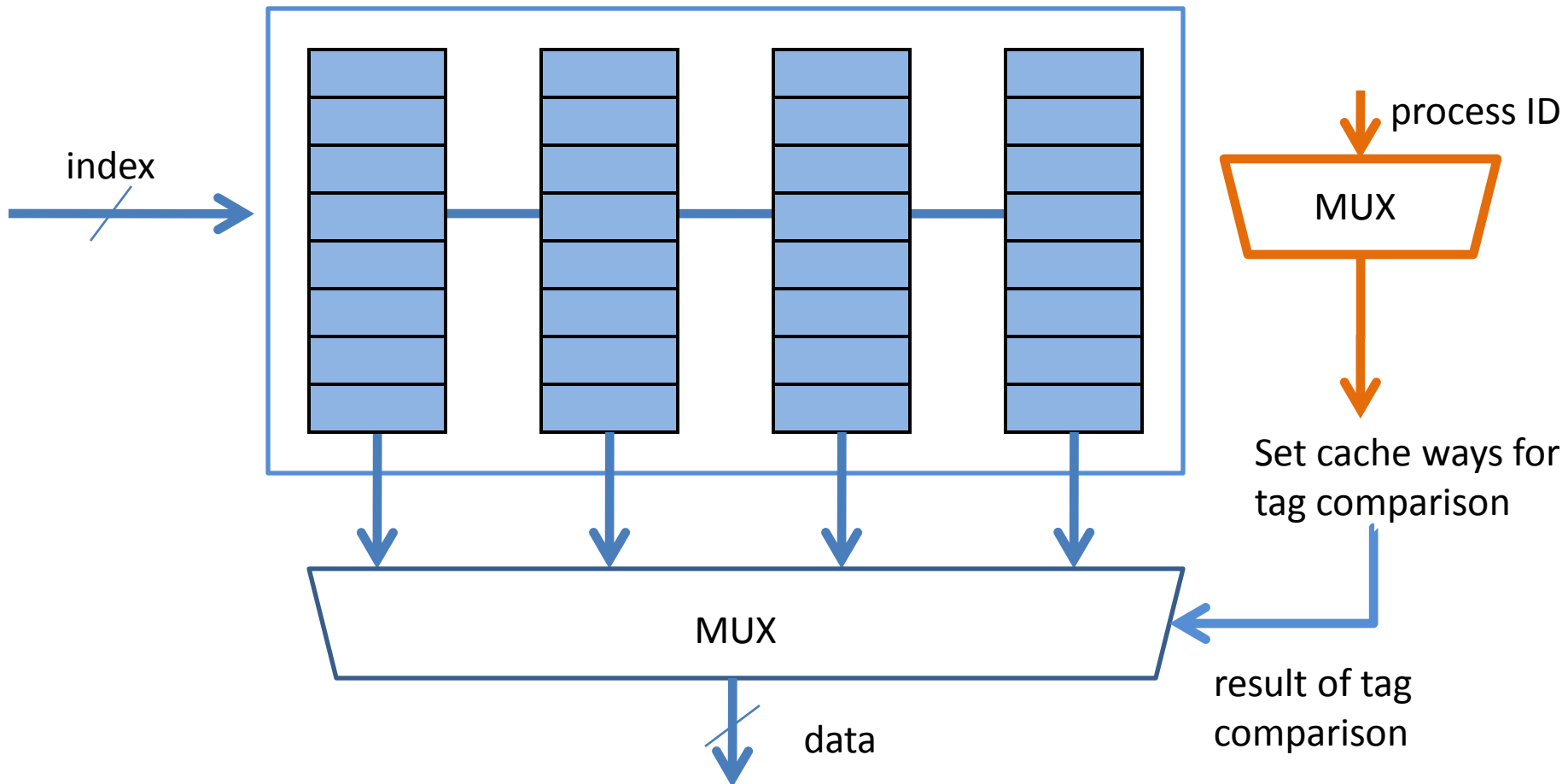
Conclusion

	Way-Partitioned Management	Decay-Based Management
Advantages	<ul style="list-style-type: none"> ▪ Simple hardware complexity ▪ Straightforward technique ▪ Great performance isolation <div style="border: 1px solid blue; padding: 5px; display: inline-block; margin-top: 10px;">50%</div>	<ul style="list-style-type: none"> ▪ Fine granularity control → more effective space utilization ▪ Data remaining in the cache: high priority and good temporal locality <div style="border: 1px solid blue; padding: 5px; display: inline-block; margin-top: 10px;">55%</div>
Drawbacks	<ul style="list-style-type: none"> ▪ Preferably, the number of cache ways \geq the number of concurrent processes ▪ Coarse granularity in space allocation → inefficient space utilization 	<ul style="list-style-type: none"> ▪ More complex hardware

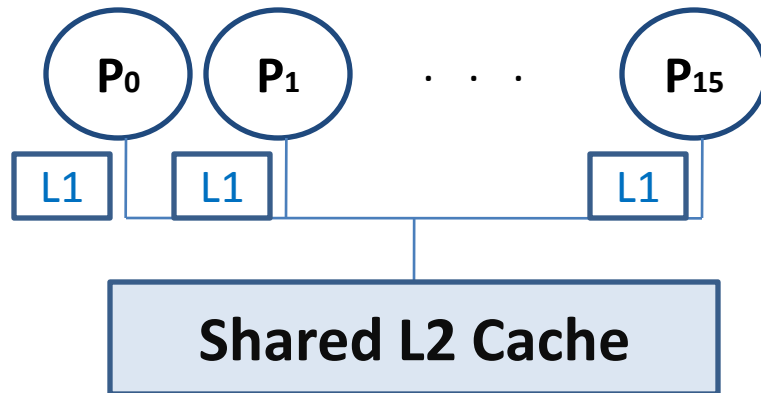
Thank you 😊

And

Hardware Overhead: Way-partitioning



What happens to the replaced lines?

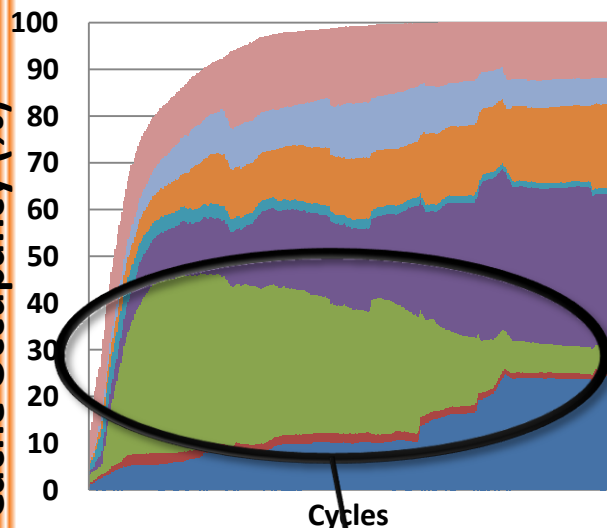


L2's replacement lines are replaced without evicting L1's copy.

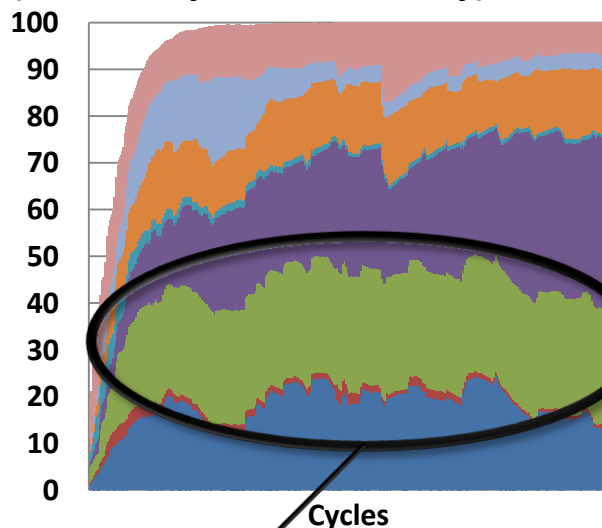
Works because L1 and L2 cache blocks are the same size!! 64Bytes.

Cache Space Distribution

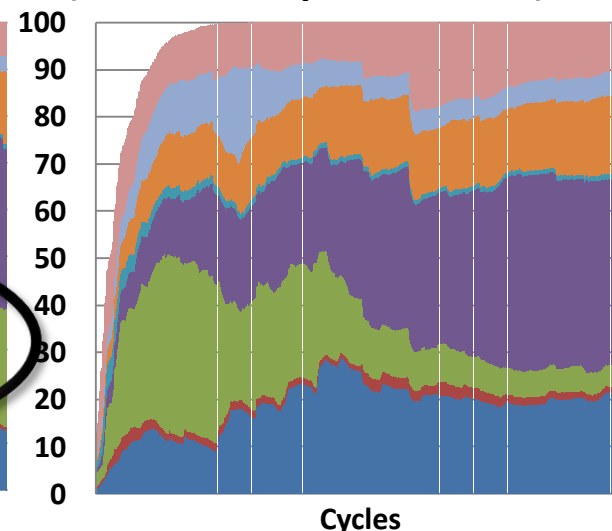
No Management



Way-Partitioned Management
(lbm-4Way; others-12Way)



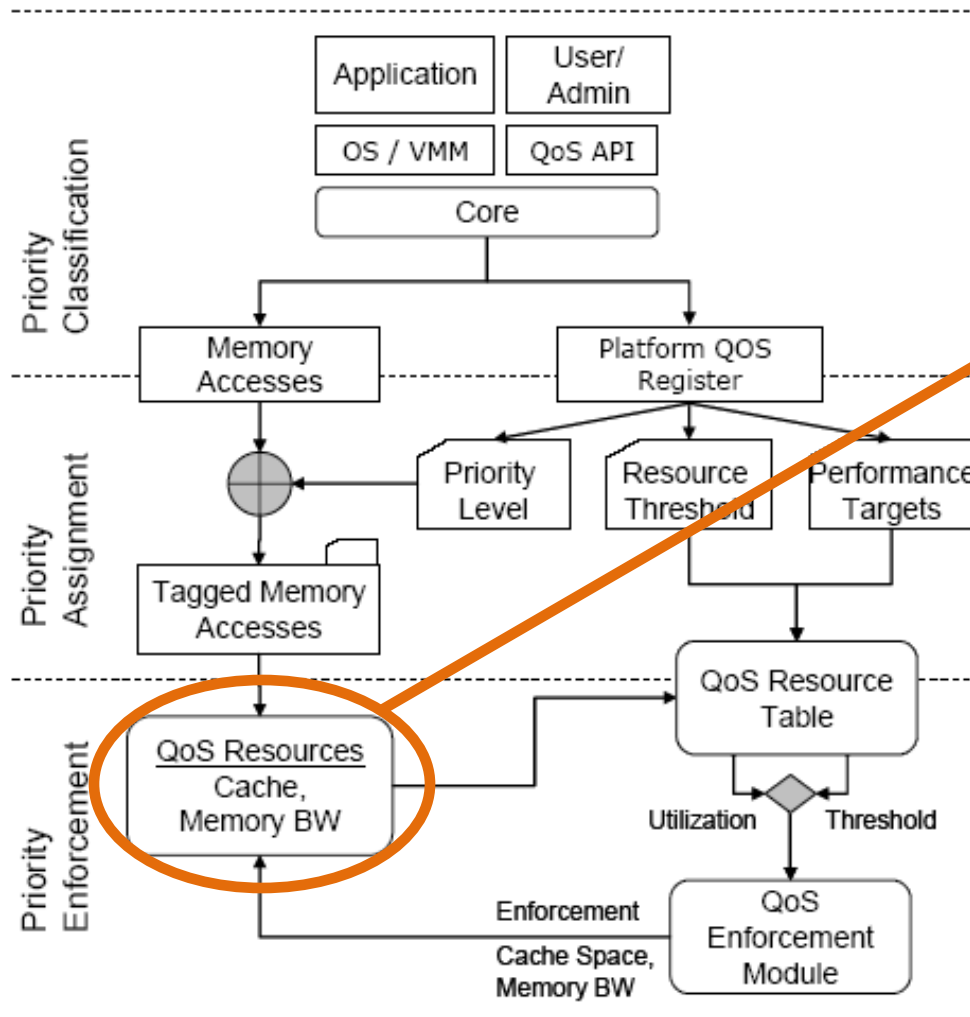
Decay-Based Management
(lbm-no decay; others-10K)



■ xalanc ■ hmmer ■ lbm ■ soplex ■ povray ■ omnetpp ■ namd ■ others

4 ways out of 16 way set-associative cache
allocated to lbm for its exclusive access

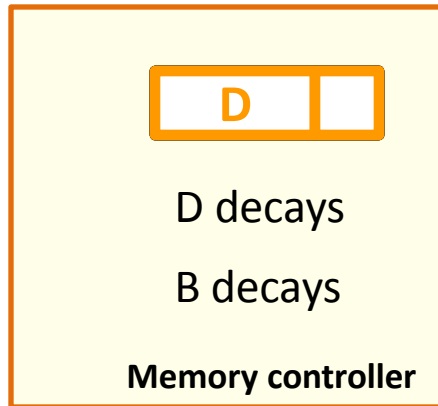
Related Work: Iyer's QoS



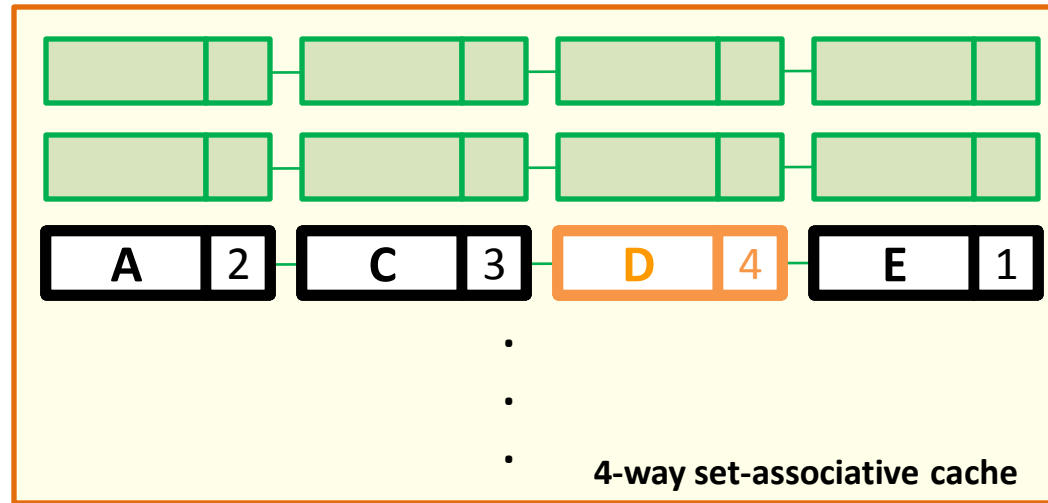
Shared Cache Capacity Management

Decay-based Management

reference stream EA...BC...D...EA...BC...



A, C, E from the **HIGH PRIORITY** process -> NO DECAY
 B, D from the **LOW PRIORITY** process -> **DECAY**



- 5 out of 9 are hits
 - All 5 hits belong to the high priority process
- LRU: NO HITS at all

LRU: Temporal Behaviors
Decay-based: Process Priority and Temporal Behaviors