
Adaptive Cache Compression for High- Performance Processors

Alaa Alameldeen and David Wood

University of Wisconsin-Madison

Presented by Samira Khan

Outline

- **Overview**
- Cache Compression Framework
- Adaptive Compression
- Evaluation
- Conclusions

Overview

- Transistor performance improves at nearly 21% each year.
- DRAM latency improves latency only 10% per year.

So we need effective cache designs.

Cache compression

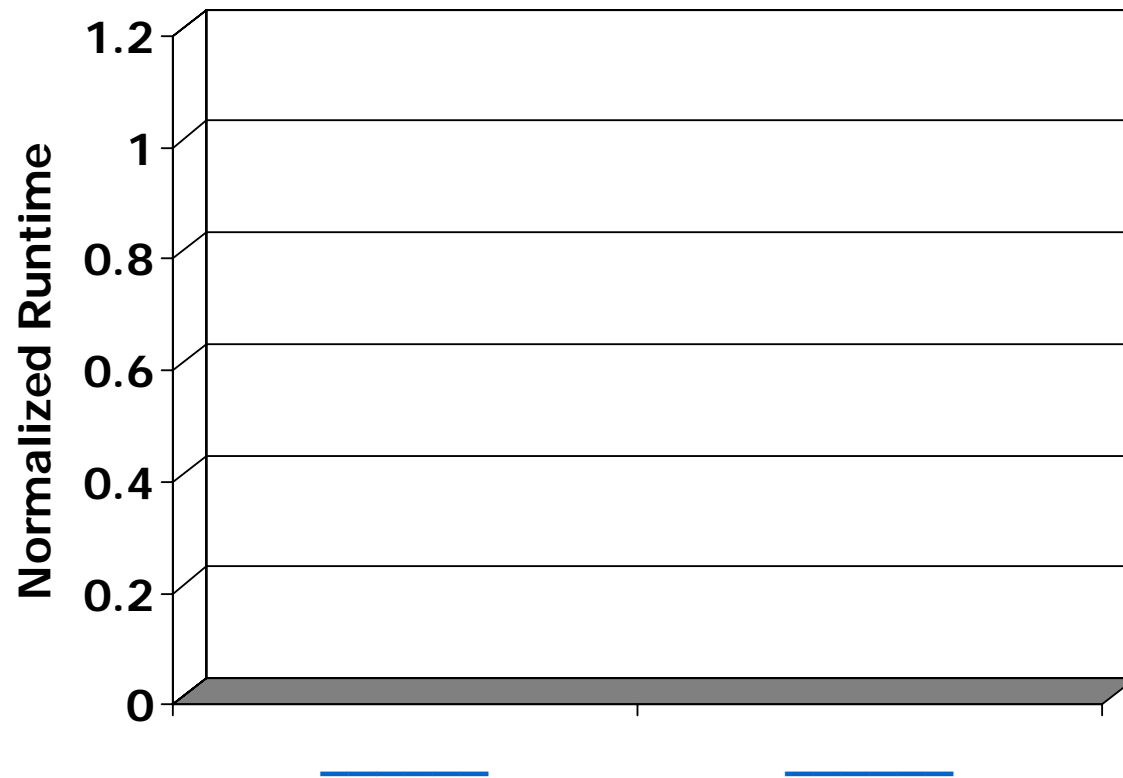
- + Increases effective cache size
- Increases cache hit latency

Cache Compression Helps or Hurts?

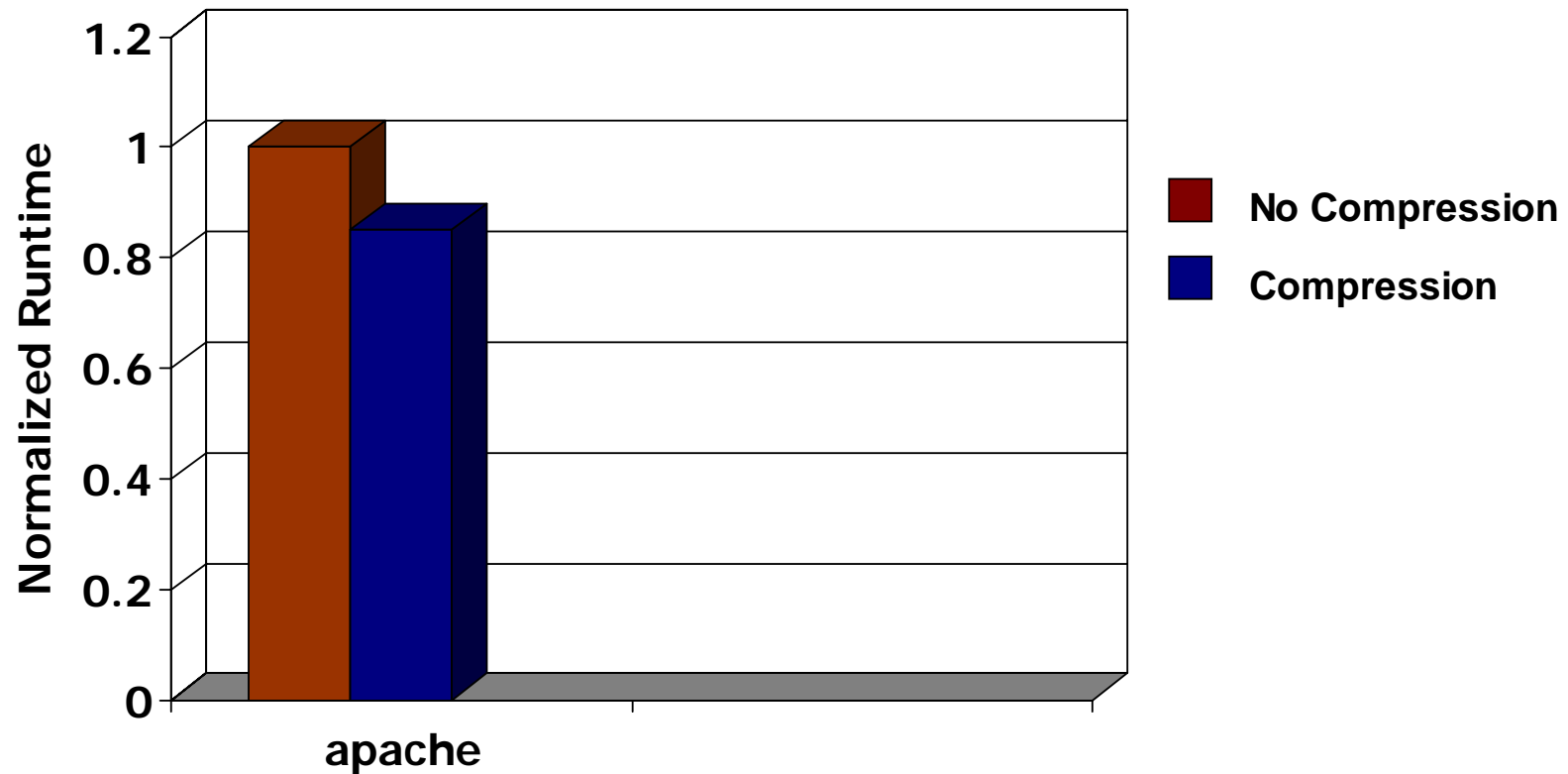
- Commercial benchmarks:
 - Capacity increases by 29-75%
 - Speed up to 17%
- Benchmarks with smaller working set:
 - Do not benefit from greater cache capacity.
 - Degrades performance as much as 18% due to decompression overhead.

So we want an **adaptive cache compression** policy that can dynamically balance the benefit of compression with the cost.

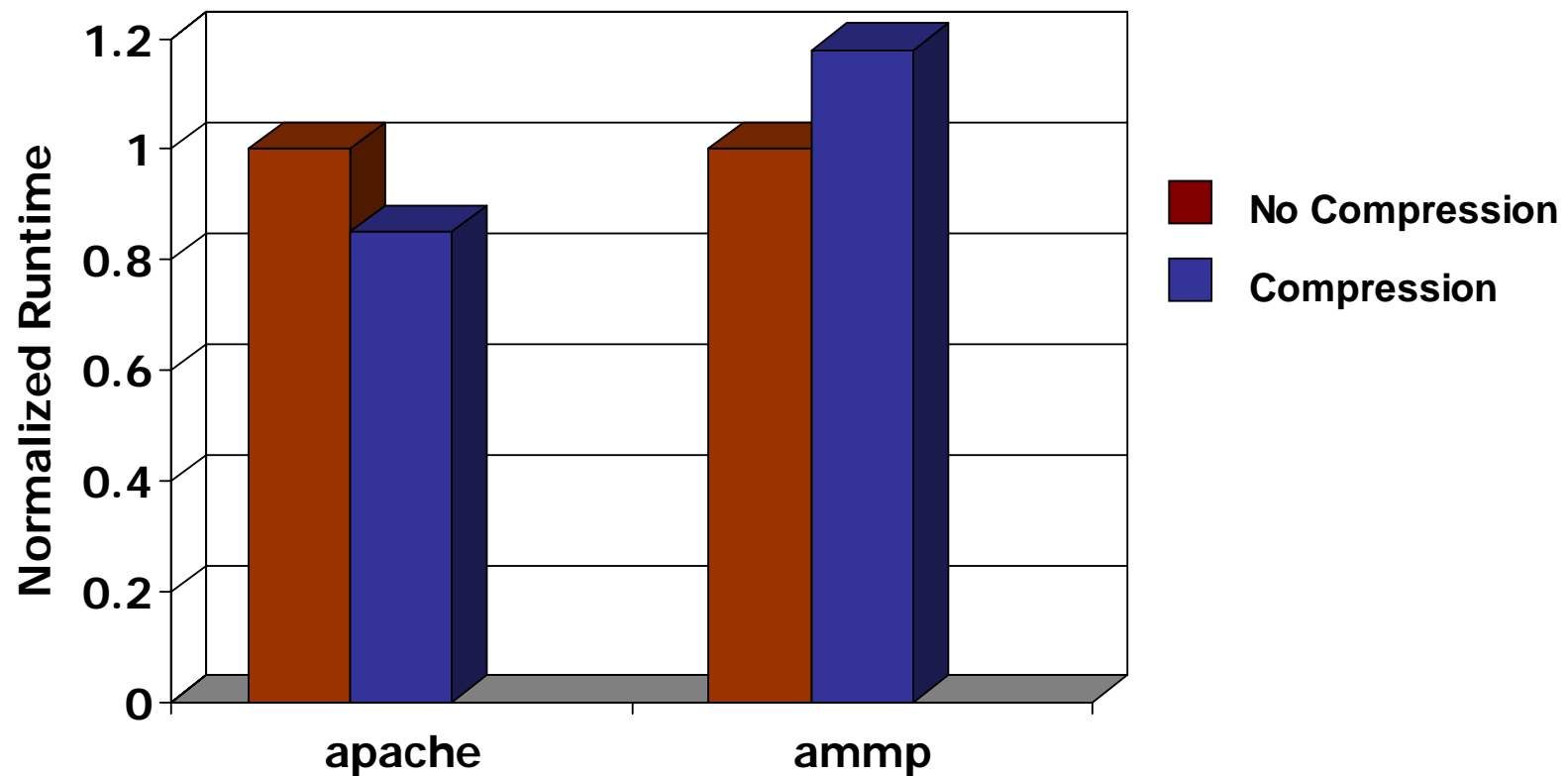
Cache Compression Helps or Hurts?



Cache Compression Helps or Hurts?



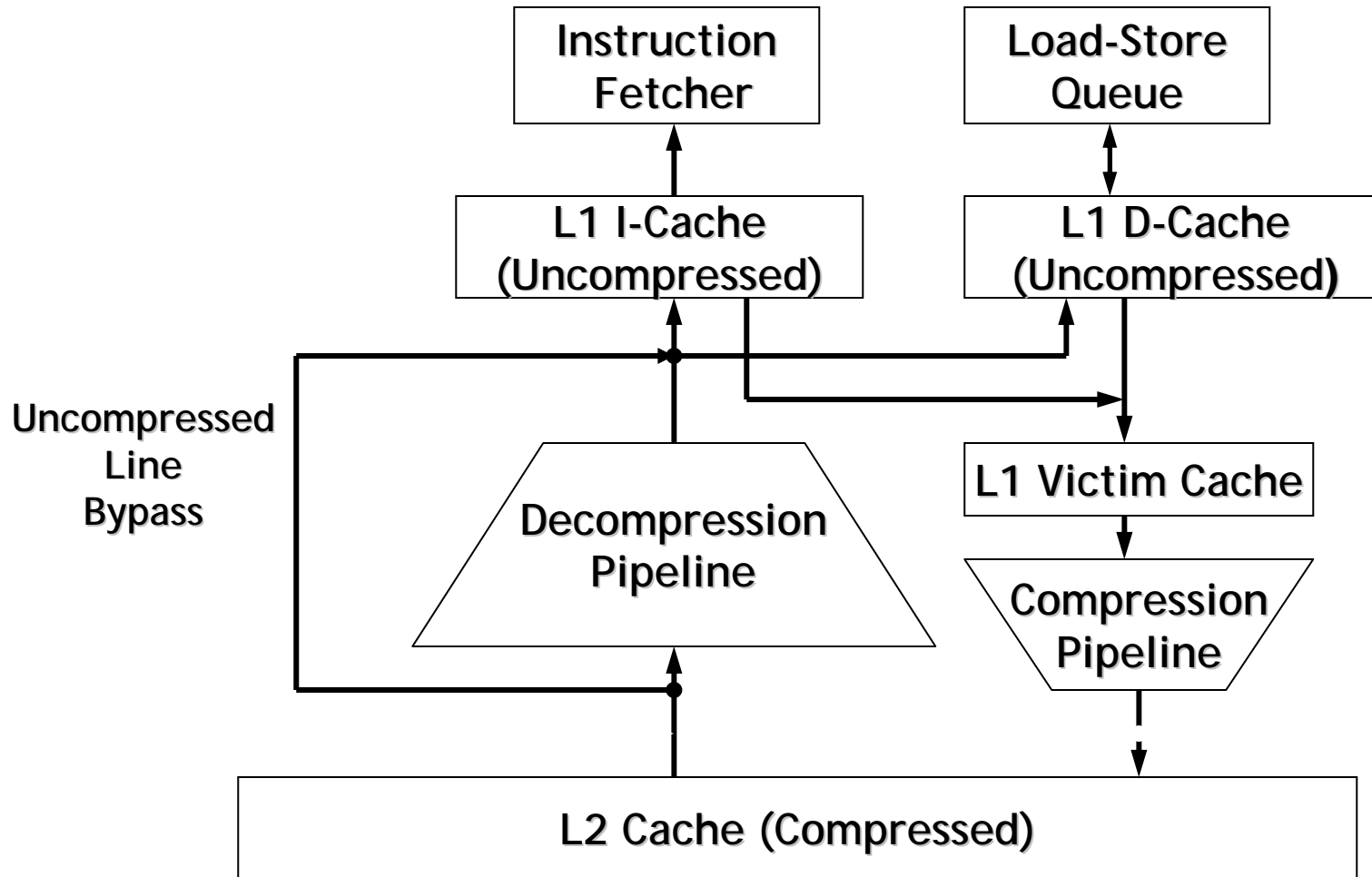
Cache Compression Helps or Hurts?



Outline

- Overview
- **Cache Compression Framework**
 - Compressed Cache Hierarchy
 - Decoupled Variable-Segment Cache
- Adaptive Compression
- Evaluation
- Conclusions

Compressed Cache Hierarchy



Compressed Cache Hierarchy

- L1 instruction and data caches store uncompressed lines.
- On L1 misses controller checks victim cache in parallel with the L2 access.
- On a L2 hit,
 - L2 line is decompressed if stored in compressed form
 - Otherwise it bypasses the decompression pipeline.

Decoupled Variable-Segment Cache

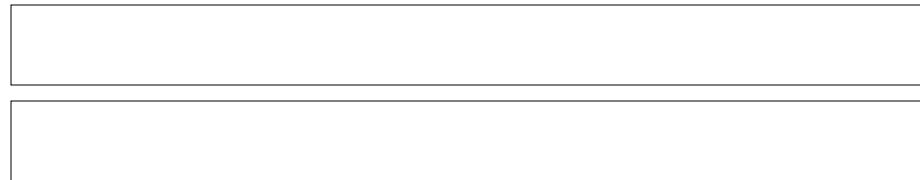
- Objective: pack more lines into the same space
- 2 way set associative with 64 byte line

Tag Area

Data Area

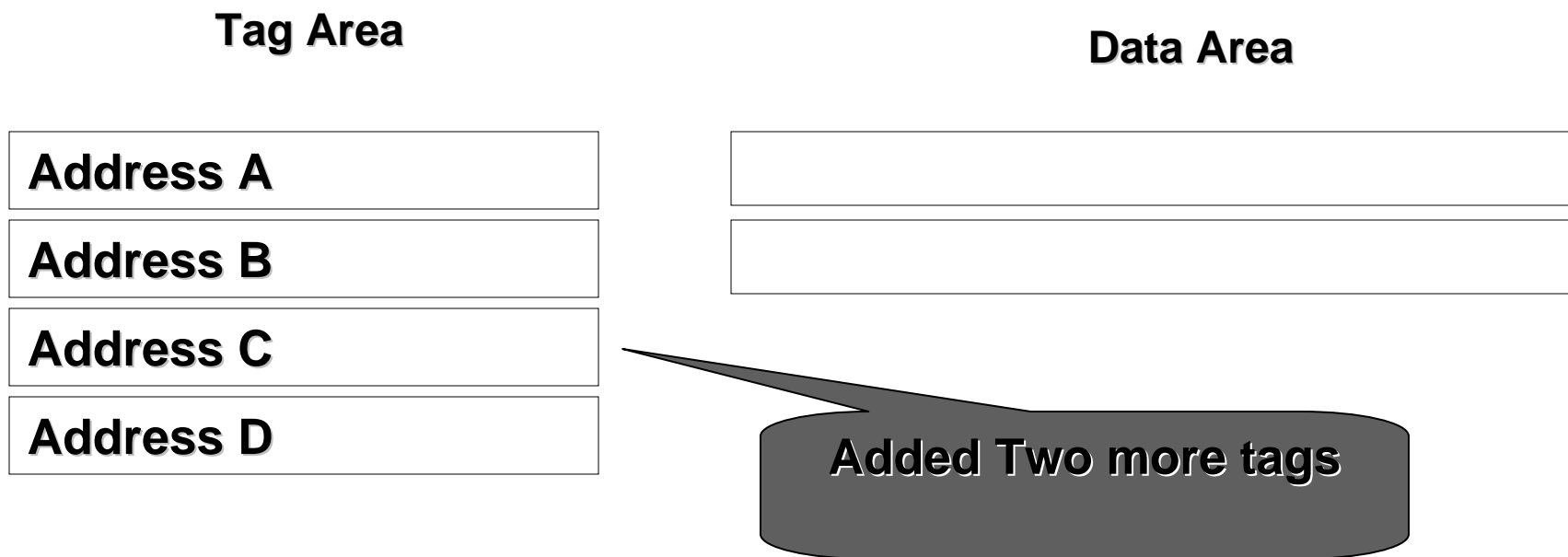
Address A

Address B



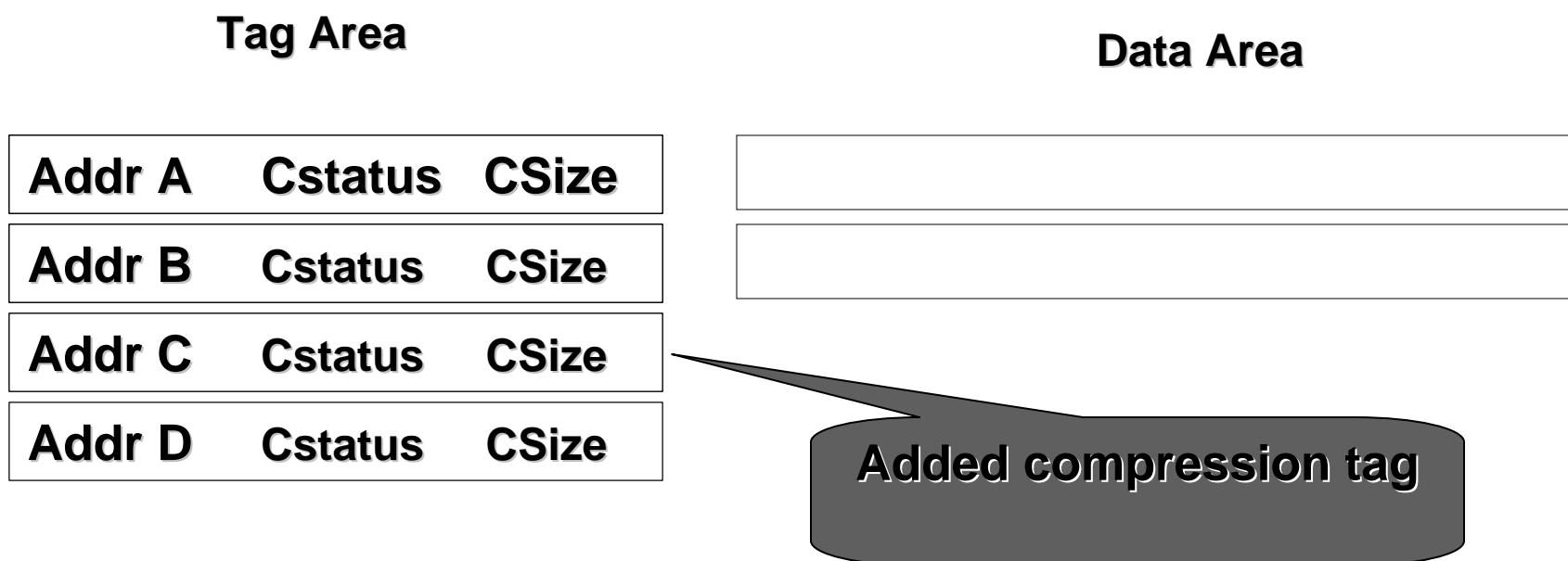
Decoupled Variable-Segment Cache

- Each set contains *four* tags and space for *two* uncompressed lines



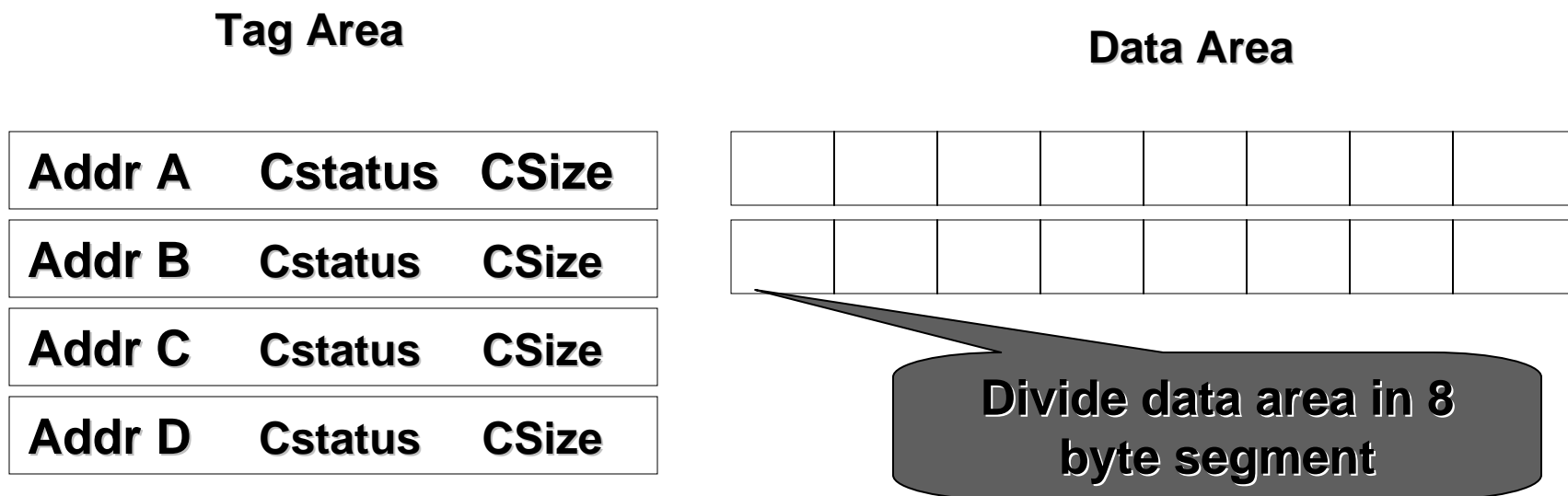
Decoupled Variable-Segment Cache

- Cstatus: compression status
- Csize: compressed size of the line



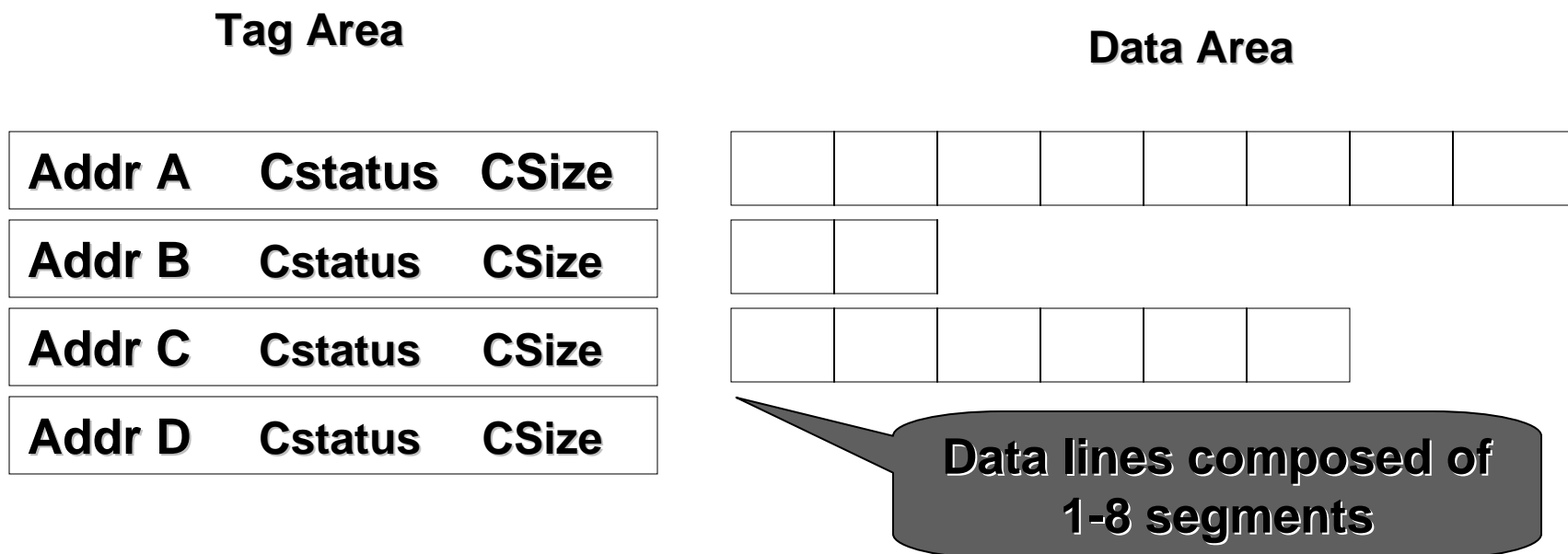
Decoupled Variable-Segment Cache

- Data area is broken into eight byte segments.
- Uncompressed line has eight segments



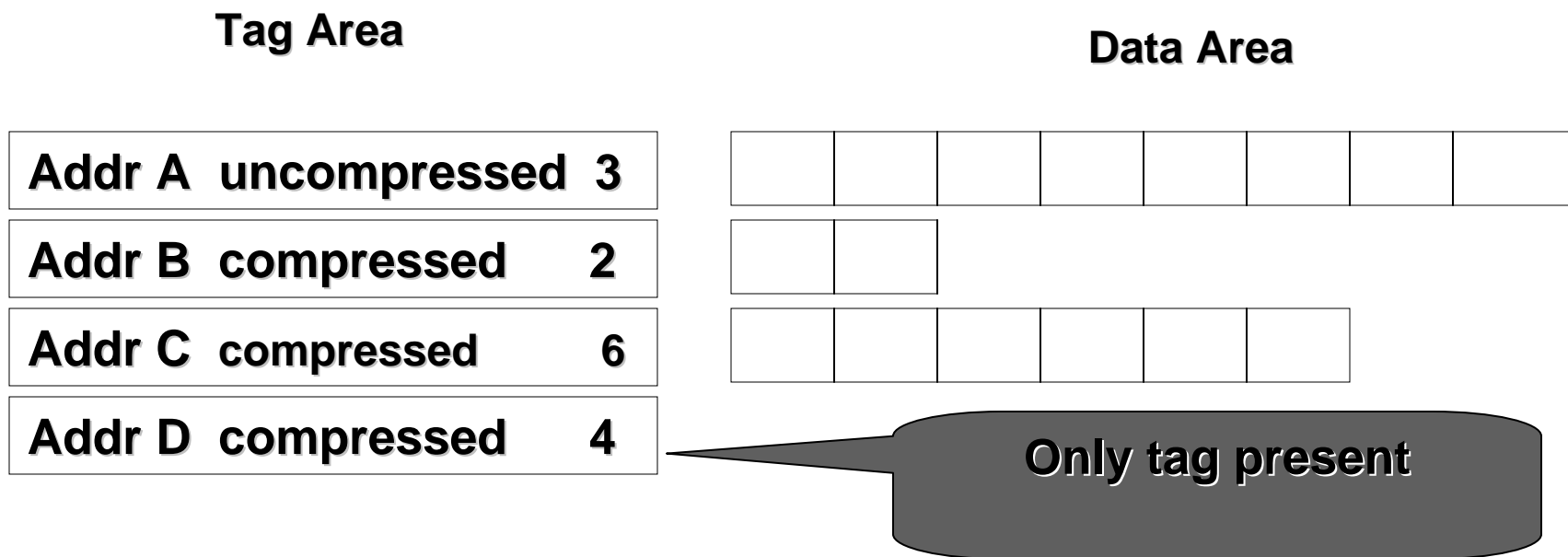
Decoupled Variable-Segment Cache

- Compressed lines can have 1-7 segments.



Decoupled Variable-Segment Cache

- Compression tags are maintained even for the NotPresent lines to use in adaptive compression policy.



Frequent Pattern Compression

- Decompresses 64 byte line in 5 cycles
- Each 64 byte is compressed into 1-8 byte segments
- Compression ratio:
For SPECint benchmark and commercial workload: 1.3-2.4
For SPECfp benchmark 1.0-1.3

Outline

- Overview
- Cache Compression Framework
- **Adaptive Compression**
- Evaluation
- Conclusions

Adaptive Cache Compression

Compress a line only if

benefit (compression) > cost (compression)

= avoided L2 misses * L2 miss penalty >
penalized L2 hits * decompression penalty

Decompression 5 cycle

L2 miss penalty 400 cycle

Compression wins if it eliminates at least one
L2 miss for every $400/5=80$ penalized hits.

Classification of Cache References

Unpenalized hit

LRU Stack

Data Area

Addr A	uncompressed	3							
Addr B	compressed	2							
Addr C	compressed	6							
Addr D	compressed	4							

- Read/Write Address A
 - LRU Stack order = 1 \leq 2 \Rightarrow Hit regardless of compression
 - Uncompressed Line \Rightarrow No decompression penalty
 - Neither cost nor benefit

Classification of Cache References

Penalized hit

LRU Stack

Data Area

Addr A	uncompressed	3								
Addr B	compressed	2								
Addr C	compressed	6								
Addr D	compressed	4								

- Read/Write Address B
 - LRU Stack order = 2 ≤ 2 ⇒ Hit regardless of compression
 - Compressed Line ⇒ Decompression penalty incurred
 - Compression cost

Classification of Cache References

Avoided miss

LRU Stack

Data Area

Addr A	uncompressed	3									
Addr B	compressed	2									
Addr C	compressed	6									
Addr D	compressed	4									

- Read/Write Address C
 - LRU Stack order = 3 > 2 ⇒ Hit only because of compression
 - Compression benefit: Eliminated off-chip miss

Classification of Cache References

Avoidable miss

LRU Stack

Data Area

Addr A	uncompressed	3								
Addr B	compressed	2								
Addr C	compressed	6								
Addr D	compressed	4								

- Read/Write Address D
 - Line is not in the cache but tag exists at LRU stack order = 4
 - Missed only because some lines are not compressed
 - Potential compression benefit

Classification of Cache References

Unavoidable miss

LRU Stack

Data Area

Addr A	uncompressed	3								
Addr B	compressed	2								
Addr C	compressed	6								
Addr D	compressed	4								

- Read/Write Address E
 - LRU stack order $> 4 \Rightarrow$ Compression wouldn't have helped
 - Line is not in the cache and tag does not exist
 - Neither cost nor benefit

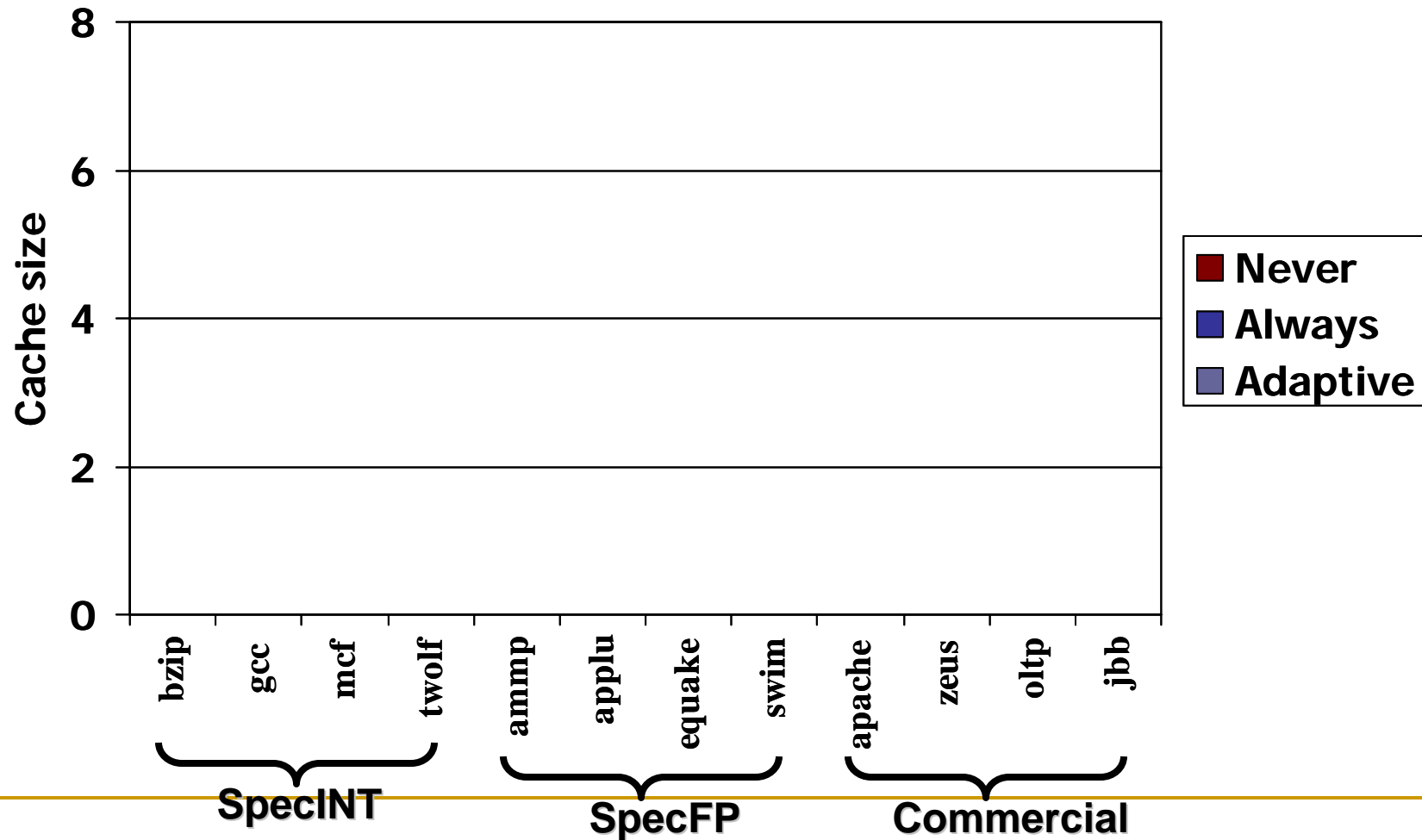
Global Compression Predictor (GCP)

- Uses past behavior to predict future.
- A global saturating counter called GCP estimates recent cost or benefit.
 - On penalized hit : decrement by decompression latency
 - On avoided \ avoidable miss : increment by memory latency.
- Cache Allocation
 - Allocate compressed line if $GCP \geq 0$
 - Allocate uncompressed lines if $GCP < 0$

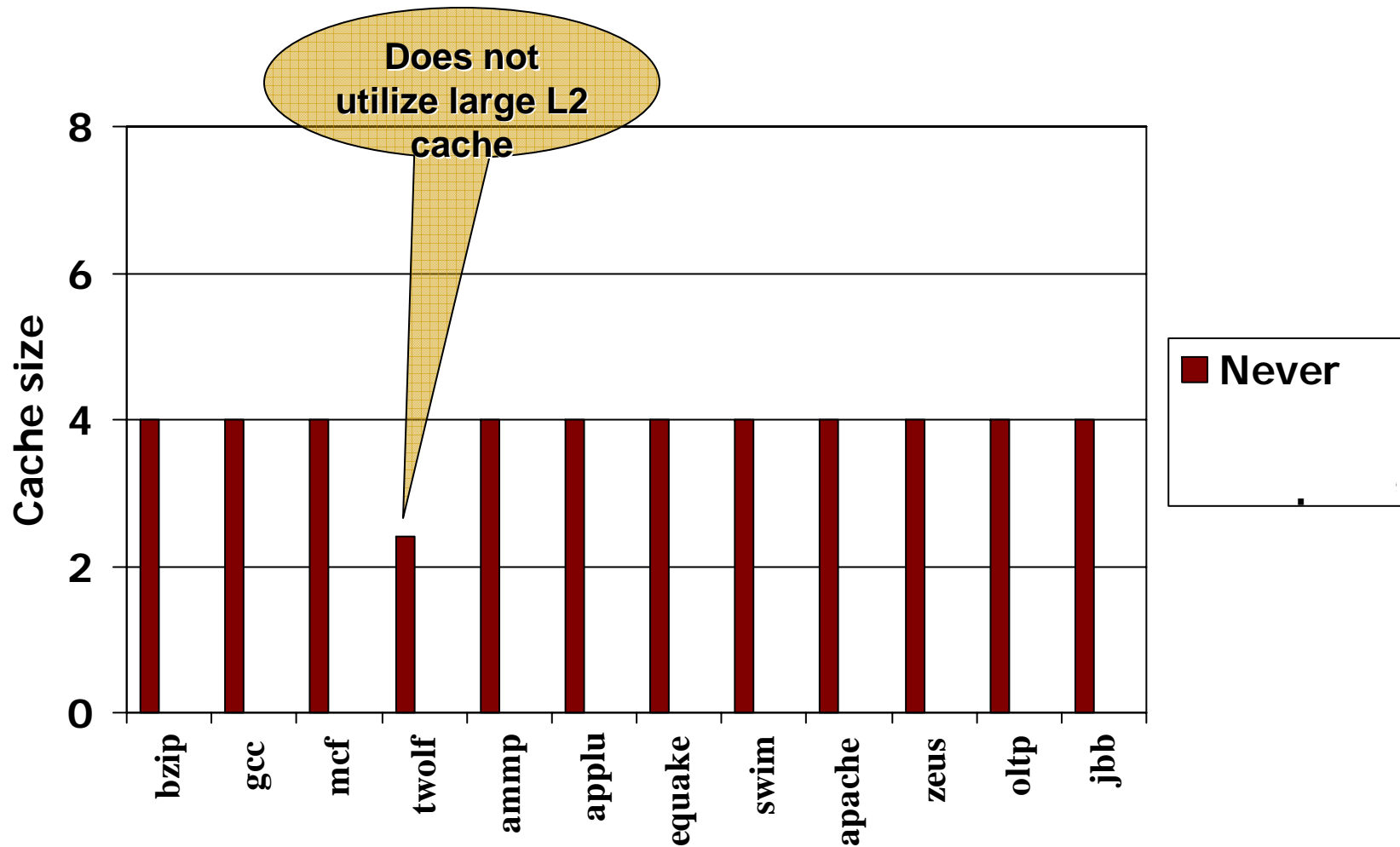
Outline

- Overview
- Cache Compression Framework
- Adaptive Compression
- **Evaluation**
- Conclusions

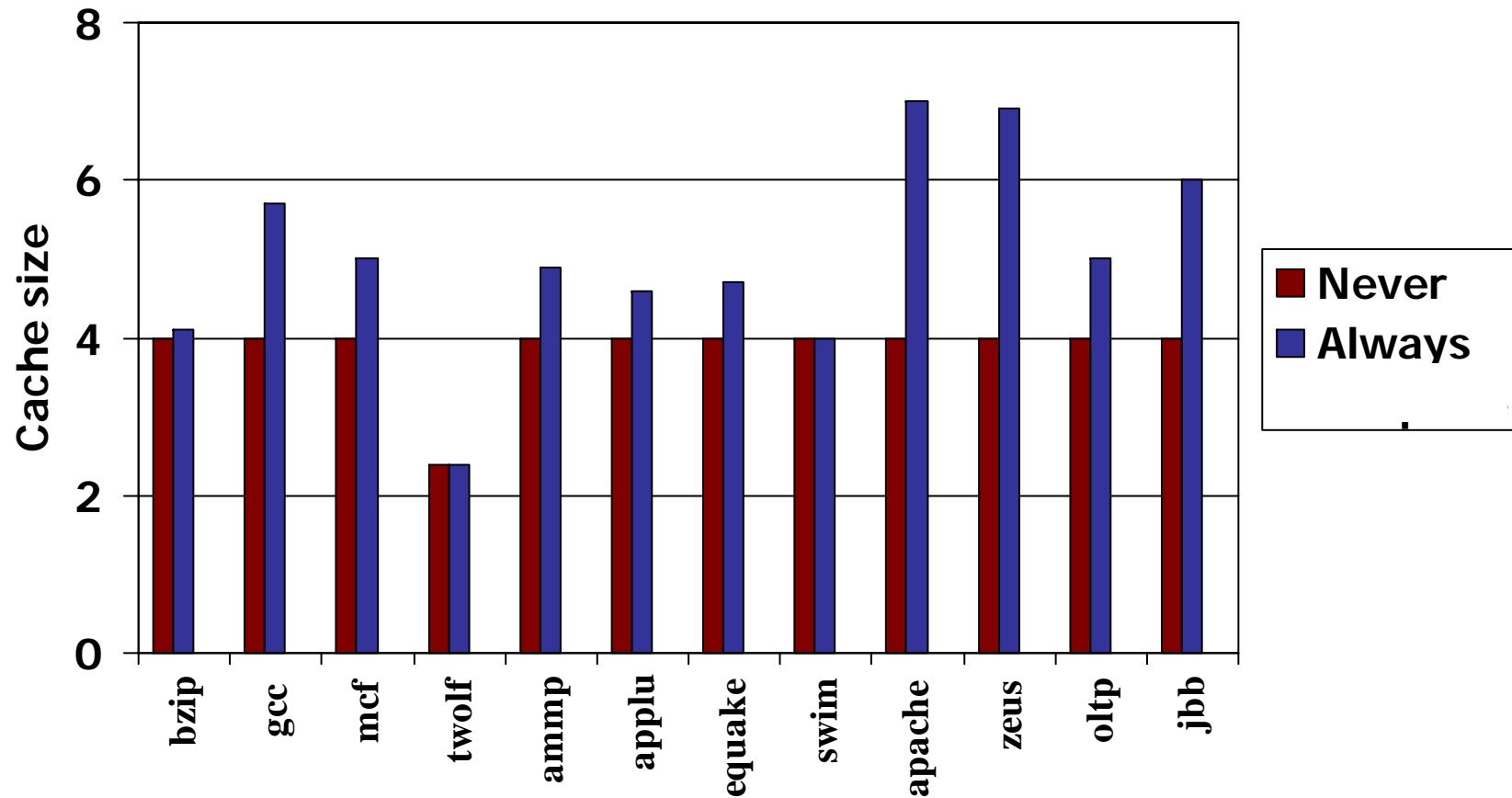
Avg Effective Cache Capacity



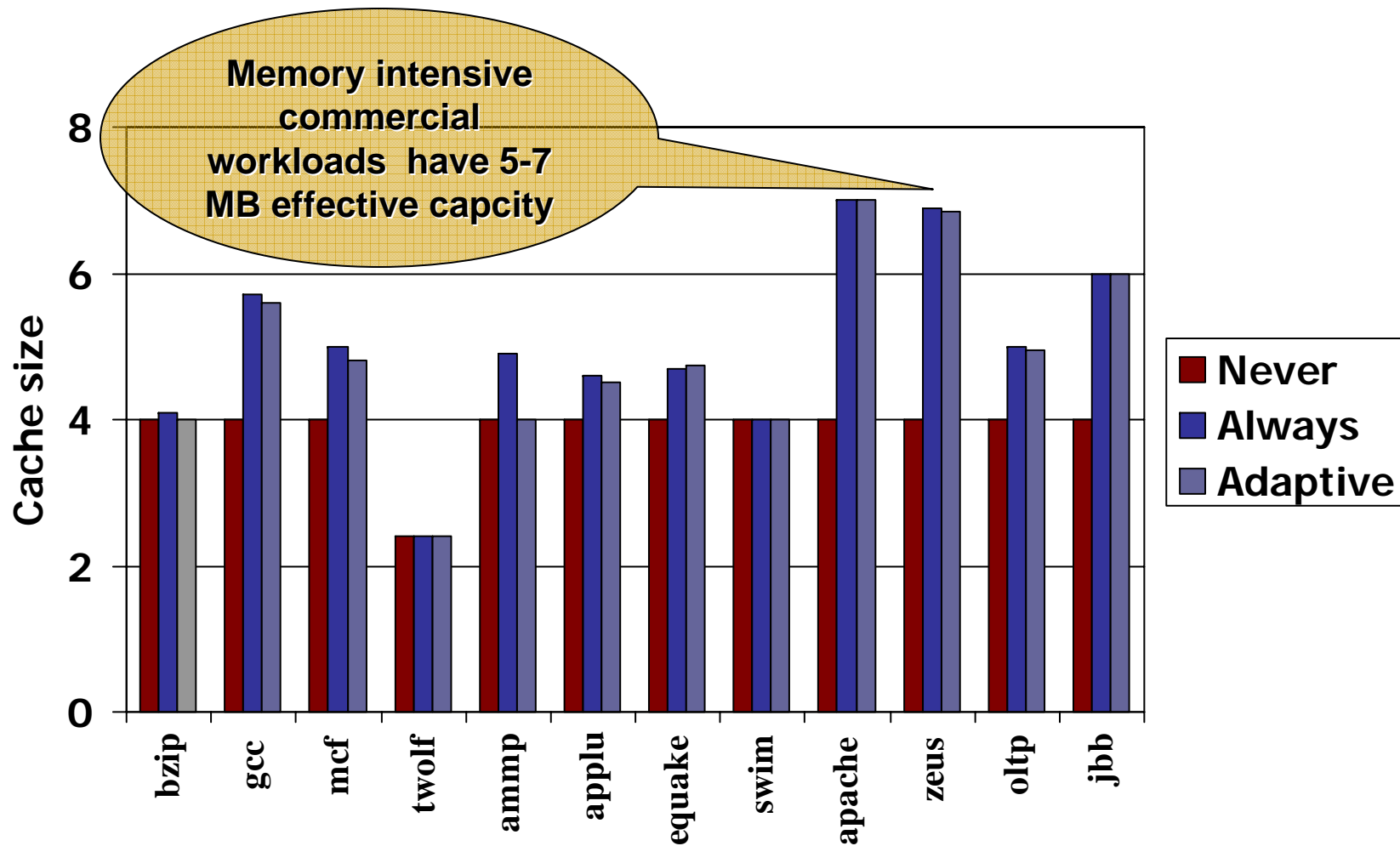
Avg Effective Cache Capacity



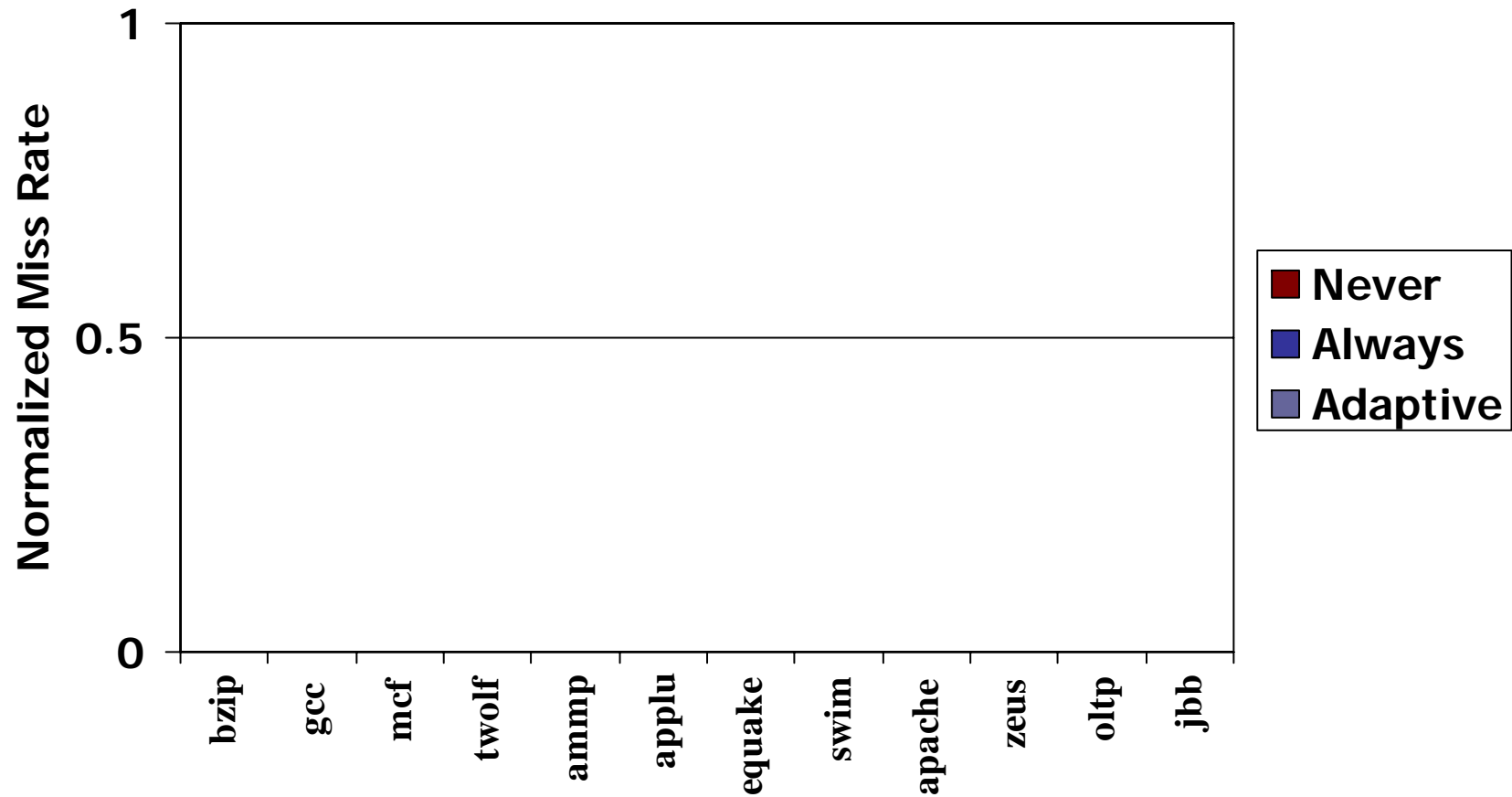
Avg Effective Cache Capacity



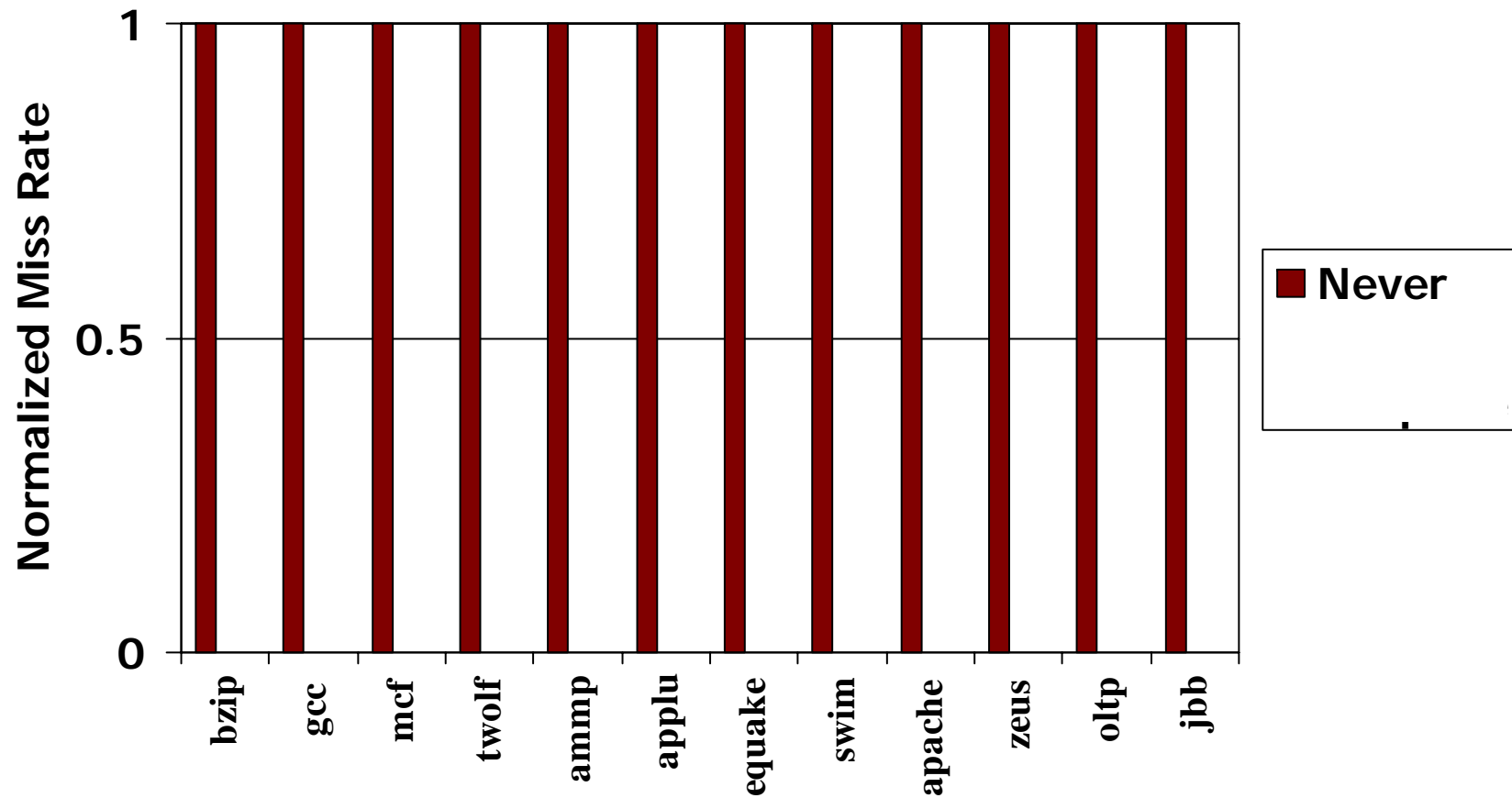
Avg Effective Cache Capacity



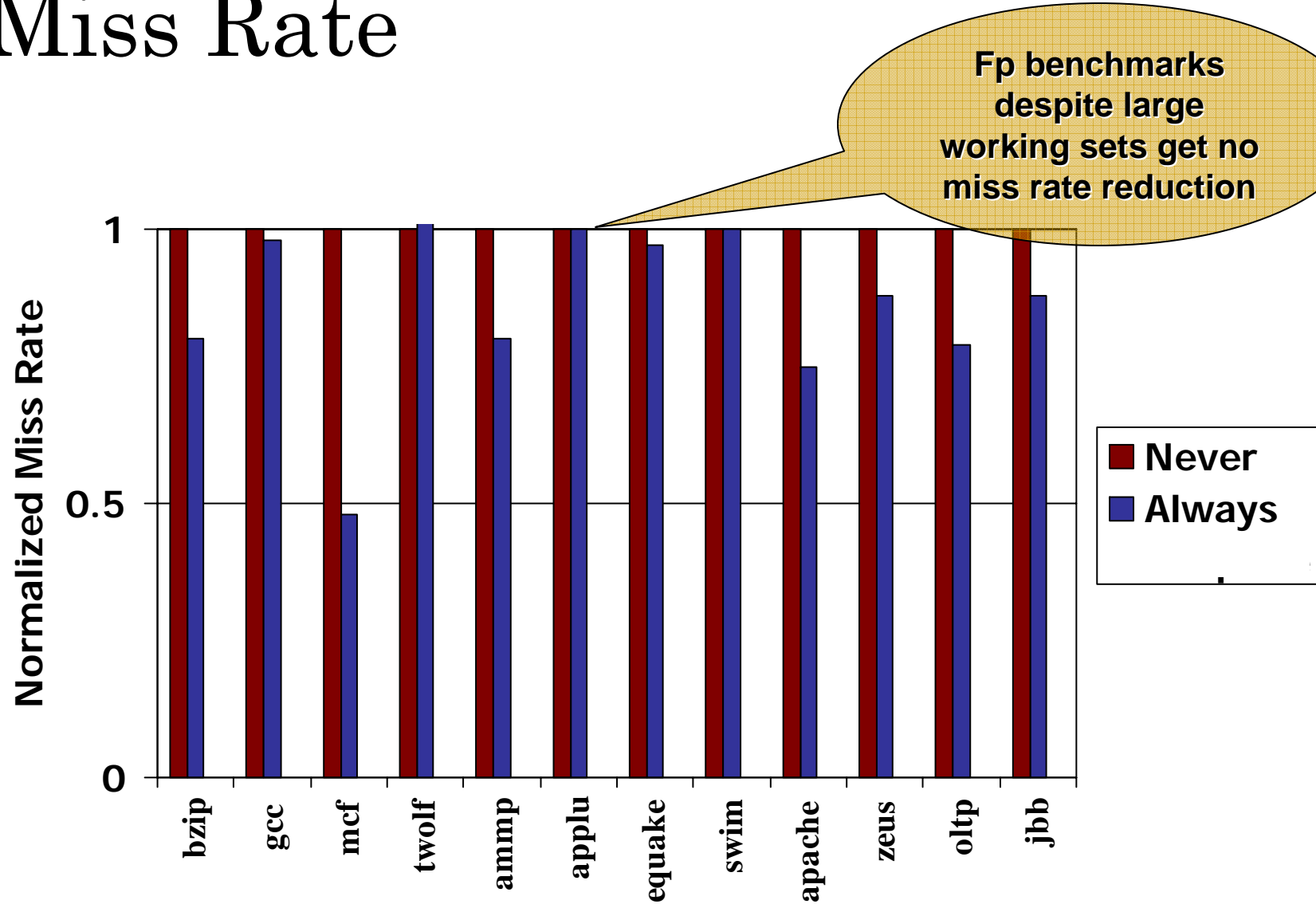
Miss Rate



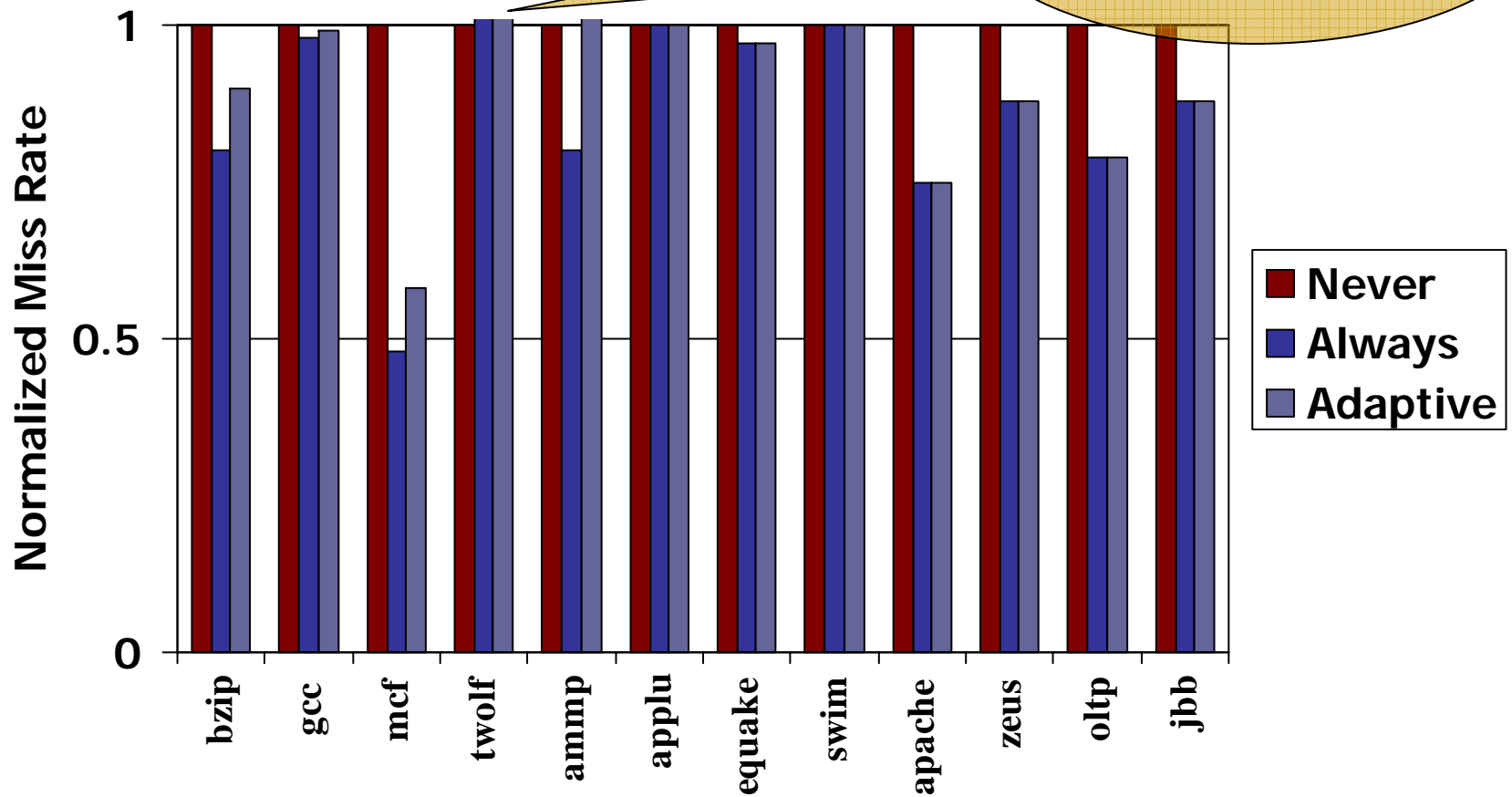
Miss Rate



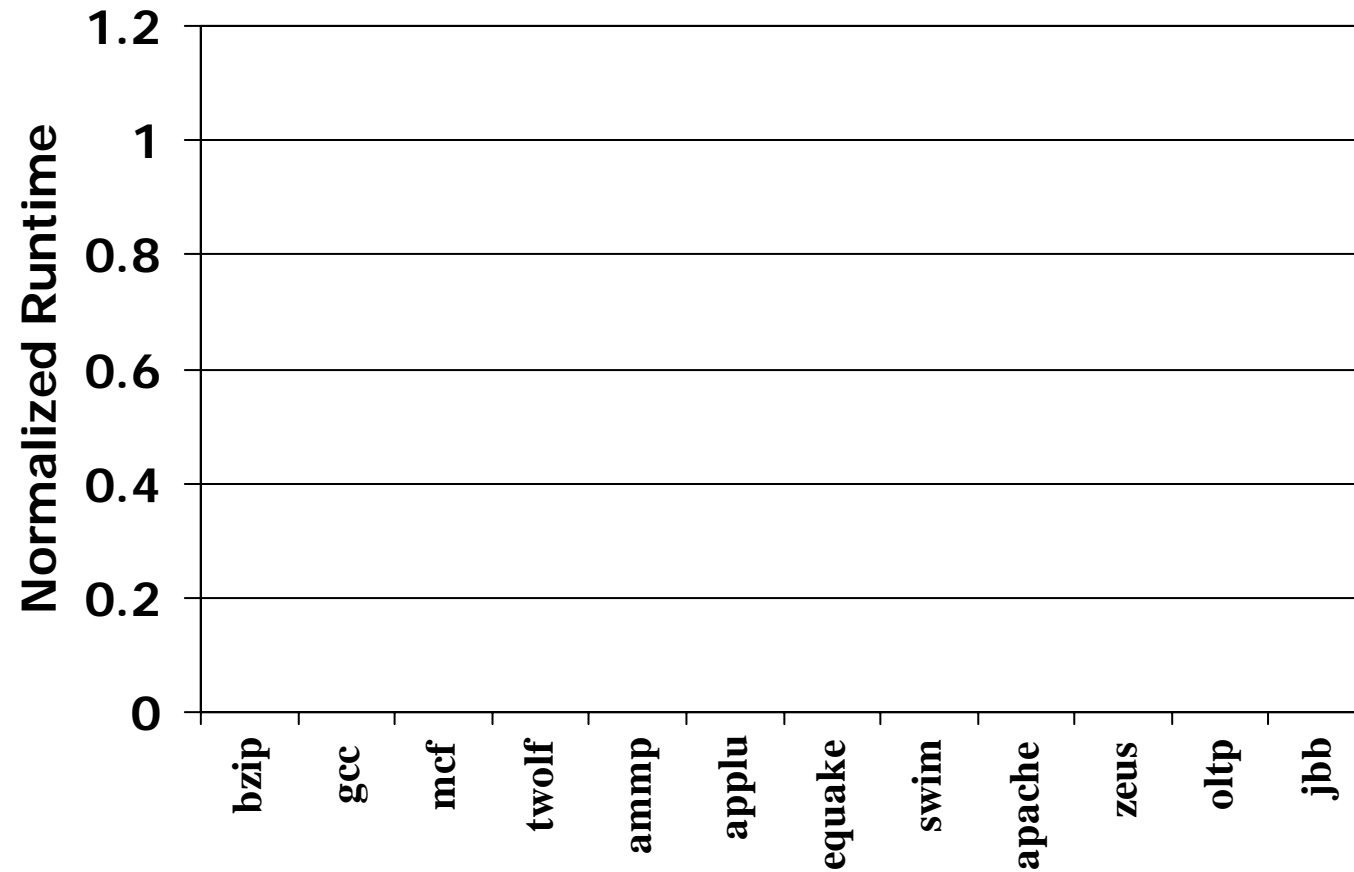
Miss Rate



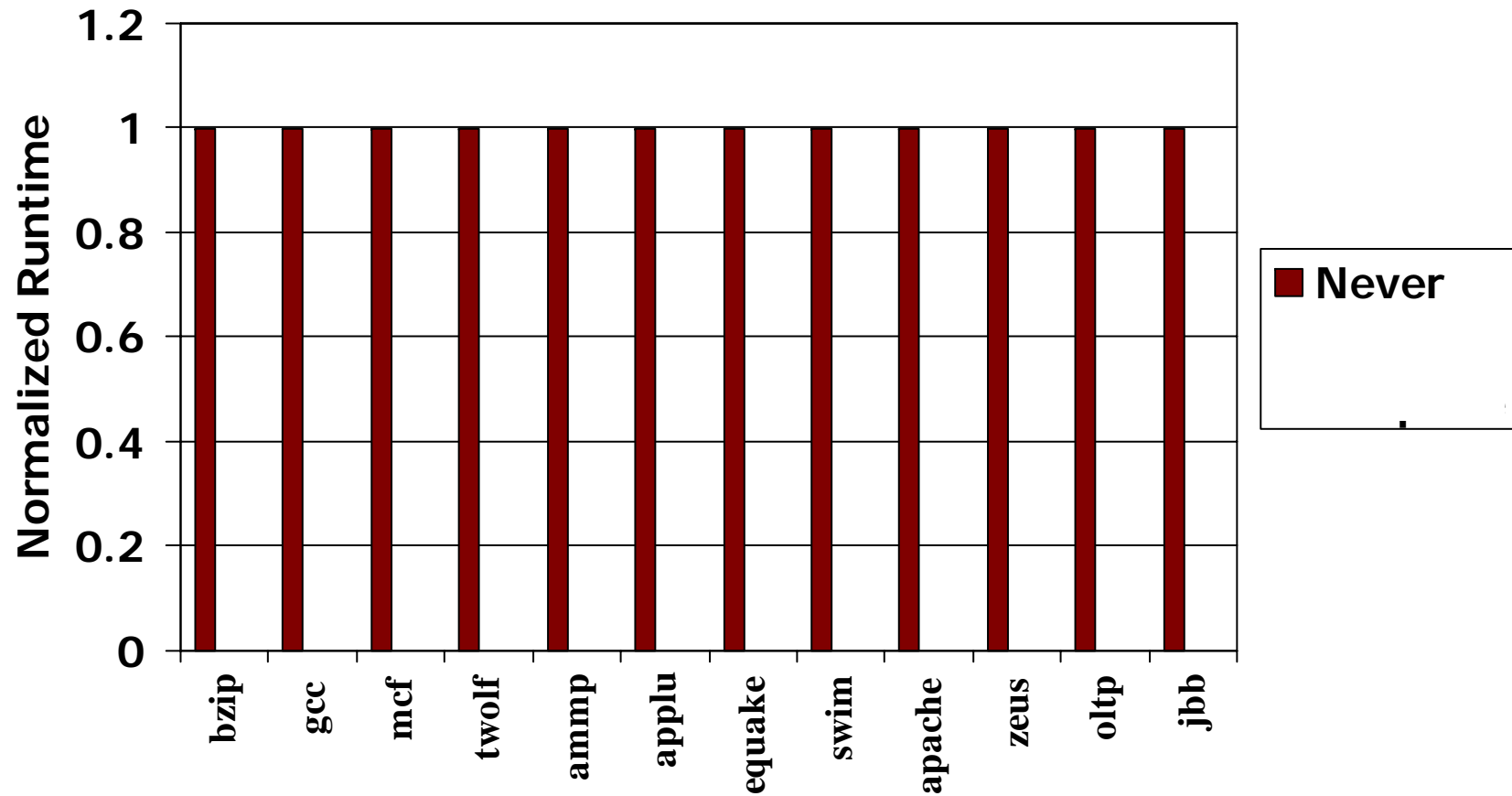
Miss Rate



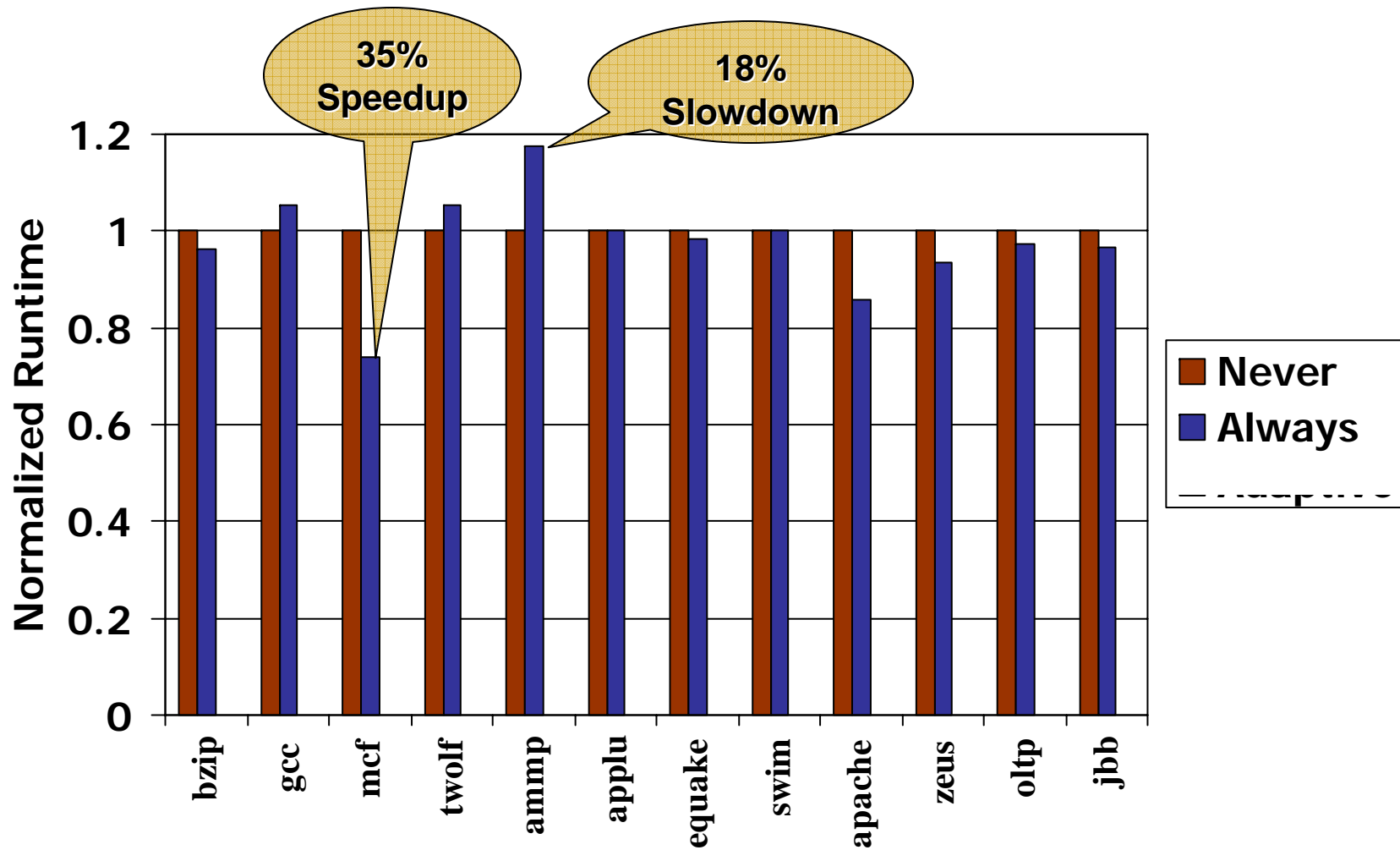
Performance



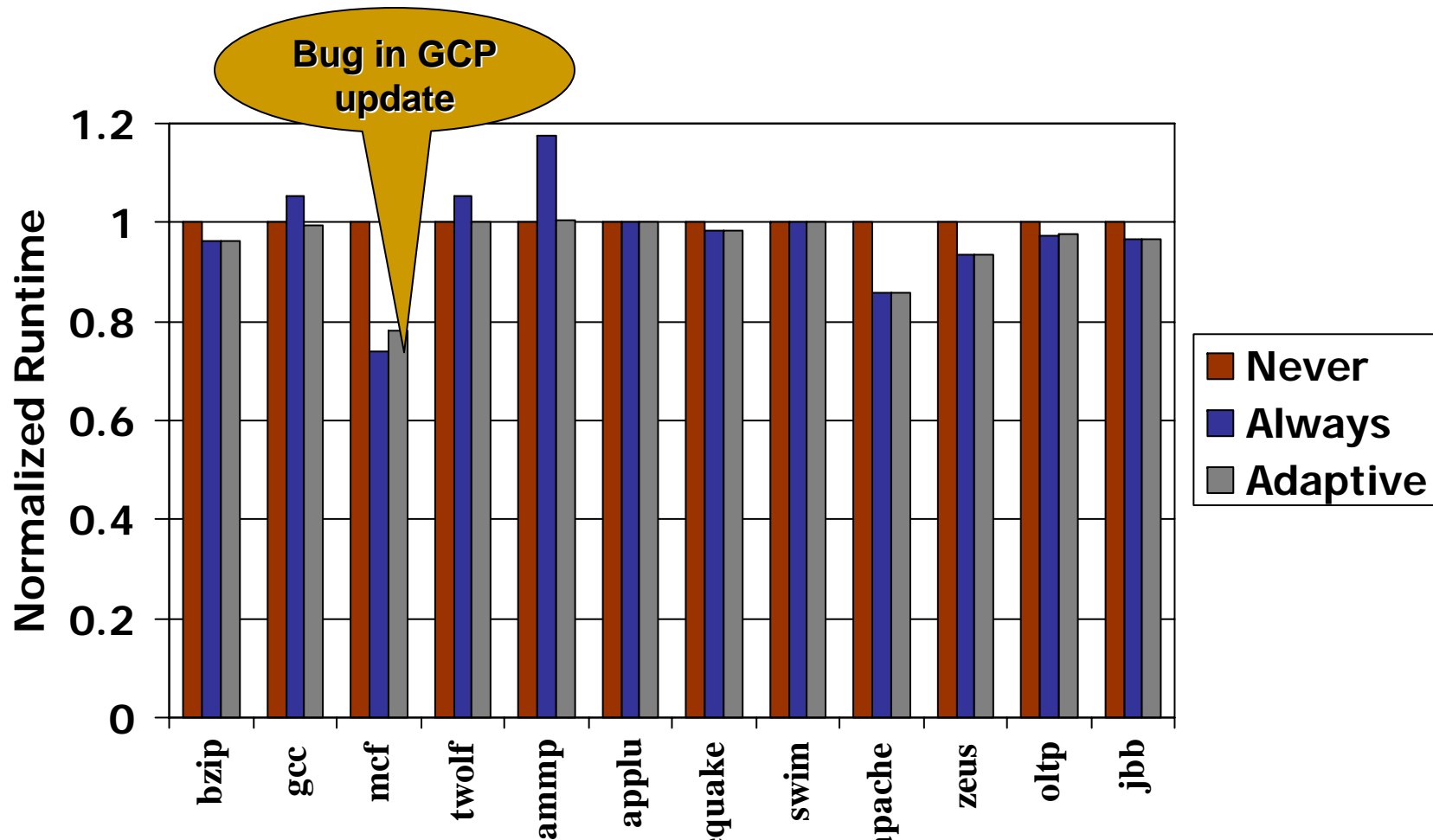
Performance



Performance



Performance



Adaptive performs similar to the best of Always and Never

Behavior of mcf

- Mcf is one benchmark where adaptive performs substantially less well than always.
- This is due to its phase behavior
- Its working set size changes frequently between different phases.
- Our predictor mispredicts alternatively for and against compression.

Outline

- Overview
- Cache Compression Framework
- Adaptive Compression
- Evaluation
- **Conclusions**

Conclusion

- Cache compression increases cache capacity but slows down cache hit time
 - Helps some benchmarks (e.g., apache, mcf)
 - Hurts other benchmarks (e.g., gcc, ammp)
- Adaptive compression
 - Uses (LRU) replacement stack to determine whether compression helps or hurts
 - Updates a single global saturating counter on cache accesses
- Adaptive compression performs similar to the better of *Always Compress* and *Never Compress*

Thank you

Simulation Setup

- Workloads:
 - Commercial workloads:
 - Database servers: OLTP and SPECJBB
 - Static Web serving: Apache and Zeus
 - SPEC2000 benchmarks:
 - SPECint: bzip, gcc, mcf, twolf
 - SPECfp: ammp, applu, equake, swim

System Configuration

L1 Cache	Split I&D, 64KB each, 2-way SA, 64B line, 2-cycles/access
L2 Cache	Unified 4MB, <i>8-way</i> SA, 64B line, 20cycles+decompression latency per access
Memory	4GB DRAM, 400-cycle access time, 128 outstanding requests
Processor pipeline	4-wide superscalar, 11-stage pipeline: fetch (3), decode(3), schedule(1), execute(1+), retire(3)
Reorder buffer	64 entries