CS 152 Computer Architecture and Engineering
CS252 Graduate Computer Architecture

Lecture 6 – Memory II

Krste Asanovic
Electrical Engineering and Computer Sciences
University of California at Berkeley

http://www.eecs.berkeley.edu/~krste
http://inst.eecs.berkeley.edu/~cs152

Last time in Lecture 6

- Dynamic RAM (DRAM) is main form of main memory storage in use today
  - Holds values on small capacitors, need refreshing (hence dynamic)
  - Slow multi-step access: precharge, read row, read column
- Static RAM (SRAM) is faster but more expensive
  - Used to build on-chip memory for caches
- Cache holds small set of values in fast memory (SRAM) close to processor
  - Need to develop search scheme to find values in cache, and replacement policy to make space for newly accessed locations
- Caches exploit two forms of predictability in memory reference streams
  - Temporal locality, same location likely to be accessed again soon
  - Spatial locality, neighboring location likely to be accessed soon

Recap: Replacement Policy

In an associative cache, which line from a set should be evicted when the set becomes full?

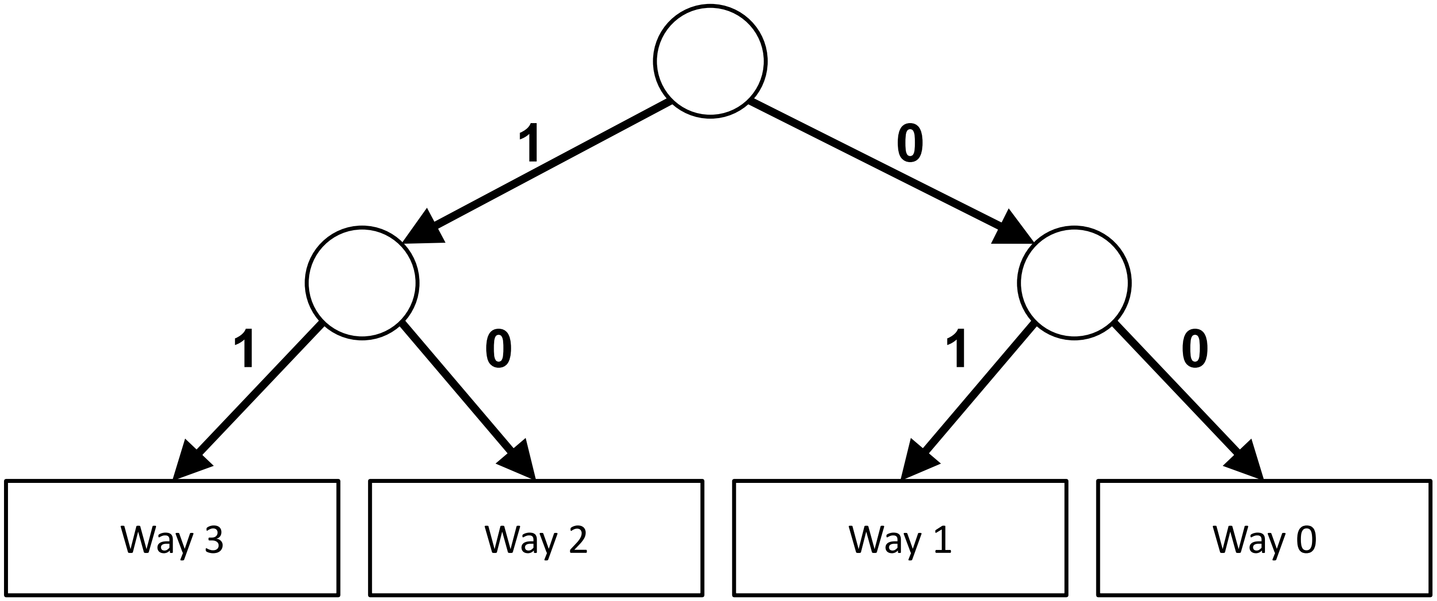
- Random
- Least-Recently Used (LRU)
  - LRU cache state must be updated on every access
  - True implementation only feasible for small sets (2-way)
  - Pseudo-LRU binary tree often used for 4-8 way
- First-In, First-Out (FIFO) a.k.a. Round-Robin
  - Used in highly associative caches
- Not-Most-Recently Used (NMRU)
  - FIFO with exception for most-recently used line or lines

This is a second-order effect. Why?

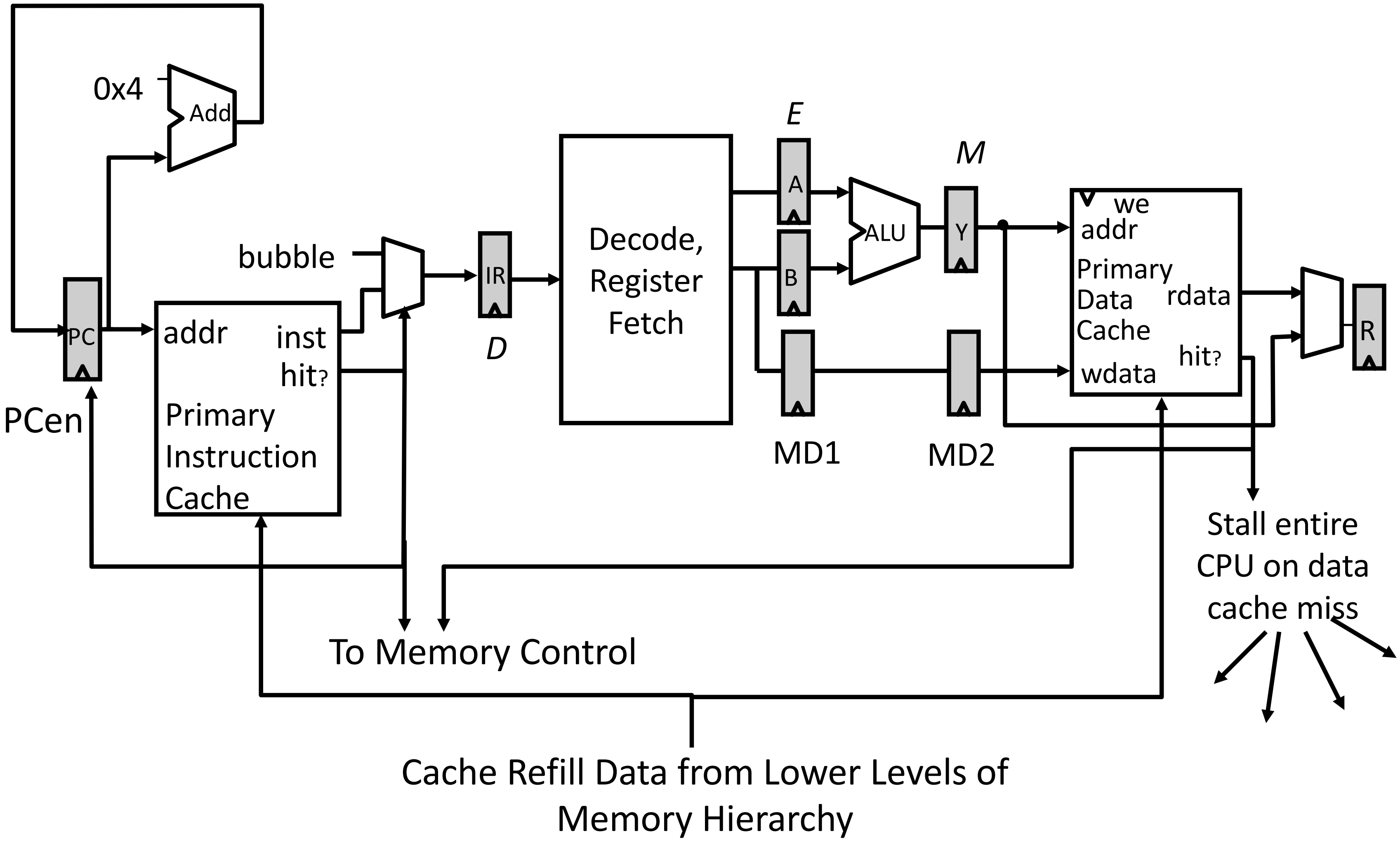
Replacement only happens on misses

Pseudo-LRU Binary Tree

- For 2-way cache, on a hit, single LRU bit is set to point to other way
- For 4-way cache, need 3 bits of state. On cache hit, on path down tree, set all bits to point to other half. On miss, bits say which way to replace



CPU-Cache Interaction
(5-stage pipeline)



Improving Cache Performance

Average memory access time (AMAT) =
Hit time + Miss rate x Miss penalty

To improve performance:

- reduce the hit time
- reduce the miss rate
- reduce the miss penalty

What is best cache design for 5-stage pipeline?

Biggest cache that doesn't increase hit time past 1 cycle (approx 8-32KB in modern technology)

[design issues more complex with deeper pipelines and/or out-of-order superscalar processors]

Causes of Cache Misses: The 3 C's

Compulsory: first reference to a line (a.k.a. cold start misses)

- *misses that would occur even with infinite cache*

Capacity: cache is too small to hold all data needed by the program

- *misses that would occur even under perfect replacement policy*

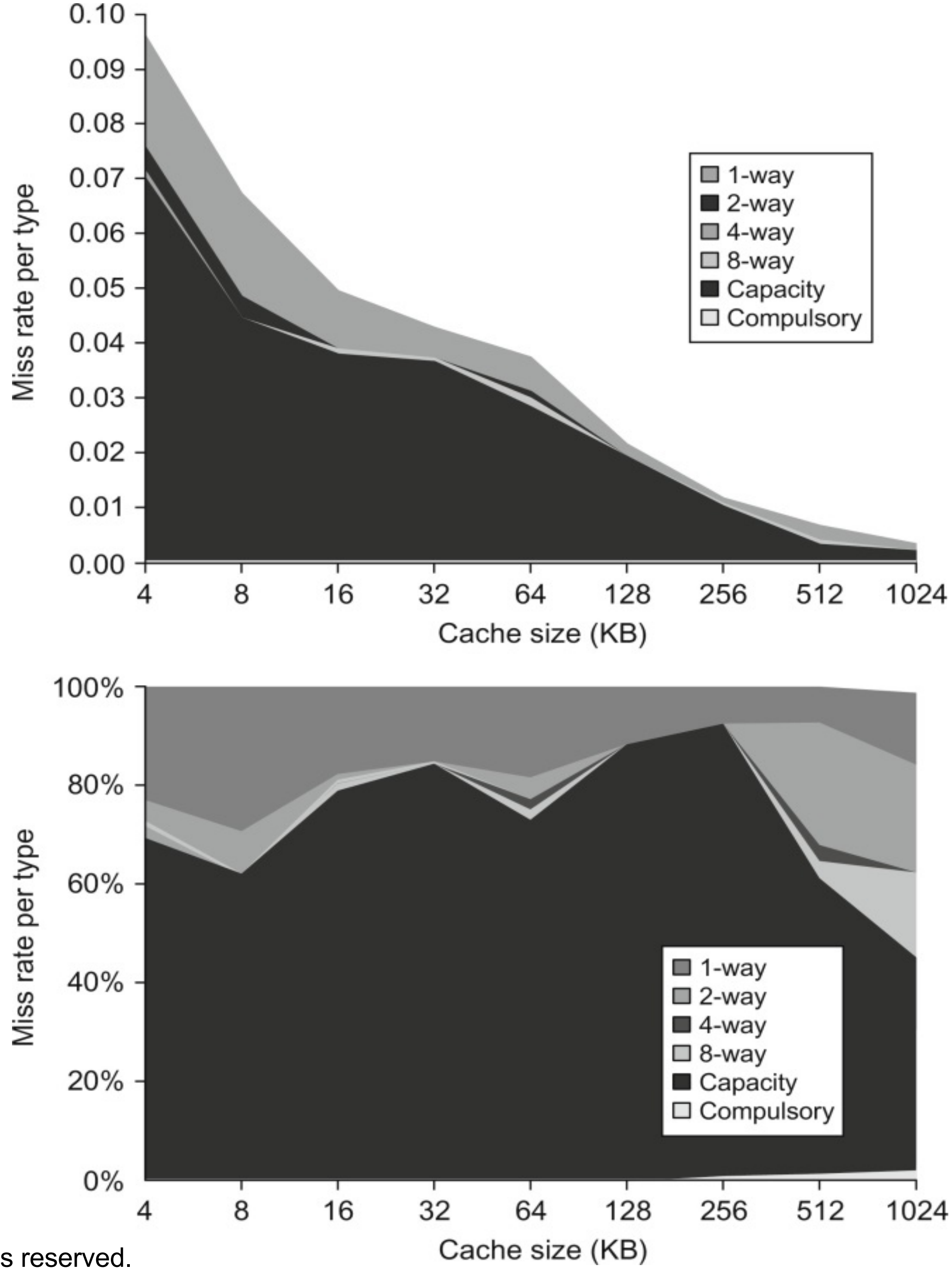
Conflict: misses that occur because of collisions due to line-placement strategy

- *misses that would not occur with ideal full associativity*

Effect of Cache Parameters on Performance

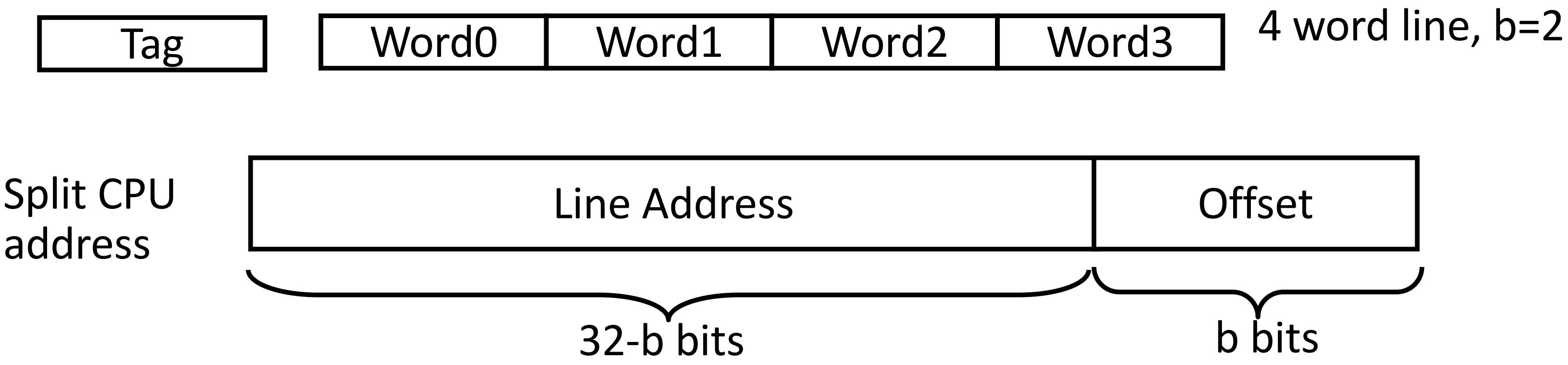
- Larger cache size
  - + reduces capacity and conflict misses
  - - hit time will increase
- Higher associativity
  - + reduces conflict misses
  - - may increase hit time
- Larger line size
  - + reduces compulsory and capacity (reload) misses
  - - increases conflict misses and miss penalty

Figure B.9 Total miss rate (top) and distribution of miss rate (bottom) for each size cache according to the three C's for the data in Figure B.8. The top diagram shows the actual data cache miss rates, while the bottom diagram shows the percentage in each category. (*Space allows* the graphs to show one extra cache size than can fit in Figure B.8.)



Recap: Line Size and Spatial Locality

A line is unit of transfer between the cache and memory



Tag | Word0 | Word1 | Word2 | Word3

4 word line, b=2

Split CPU address

Line Address | Offset

32-b bits

b bits

2^b = line size *a.k.a* line size (in bytes)

Larger line size has distinct hardware advantages

- less tag overhead
- exploit fast burst transfers from DRAM
- exploit fast burst transfers over wide busses

What are the disadvantages of increasing line size?

Fewer lines => more conflicts. Can waste bandwidth.

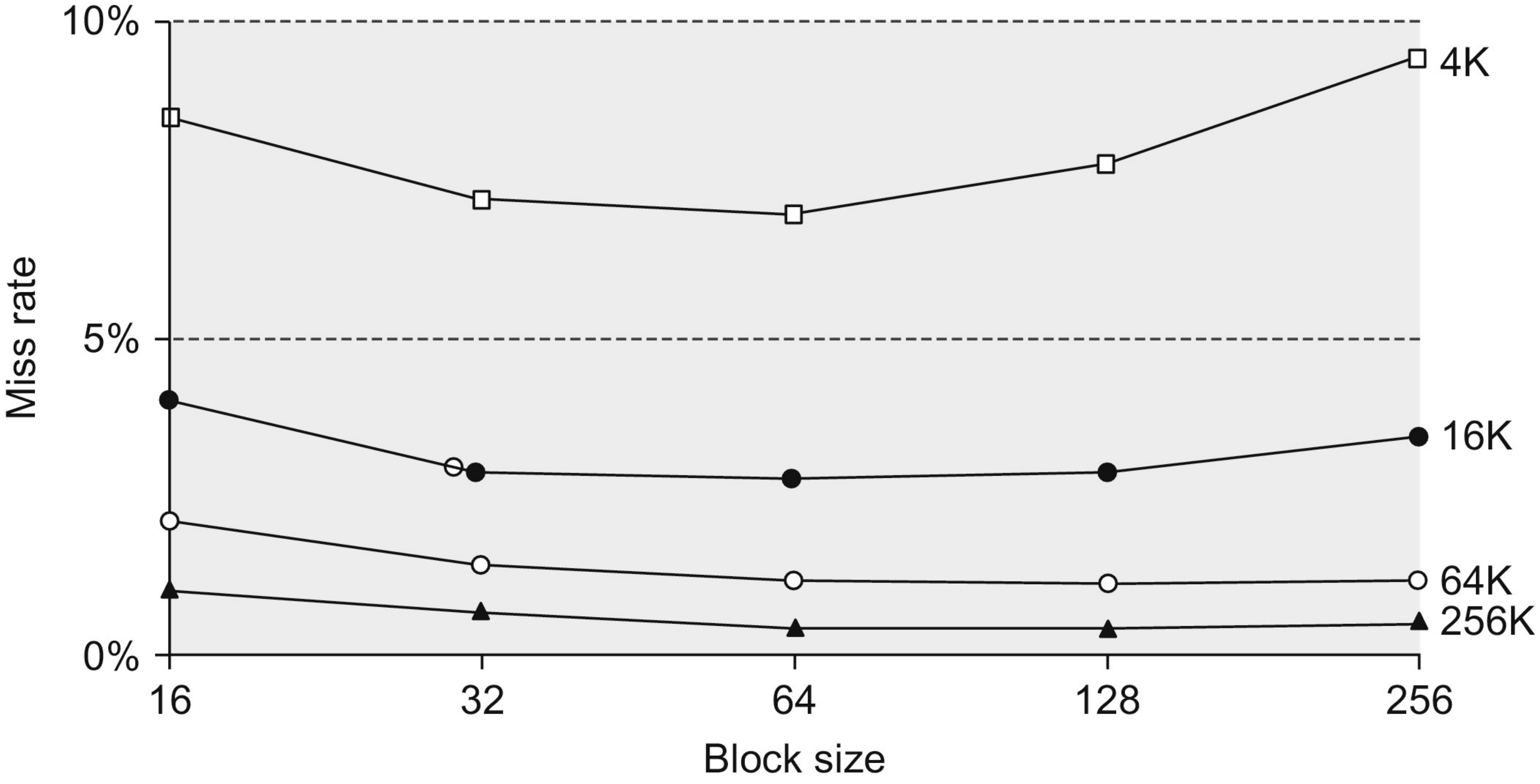
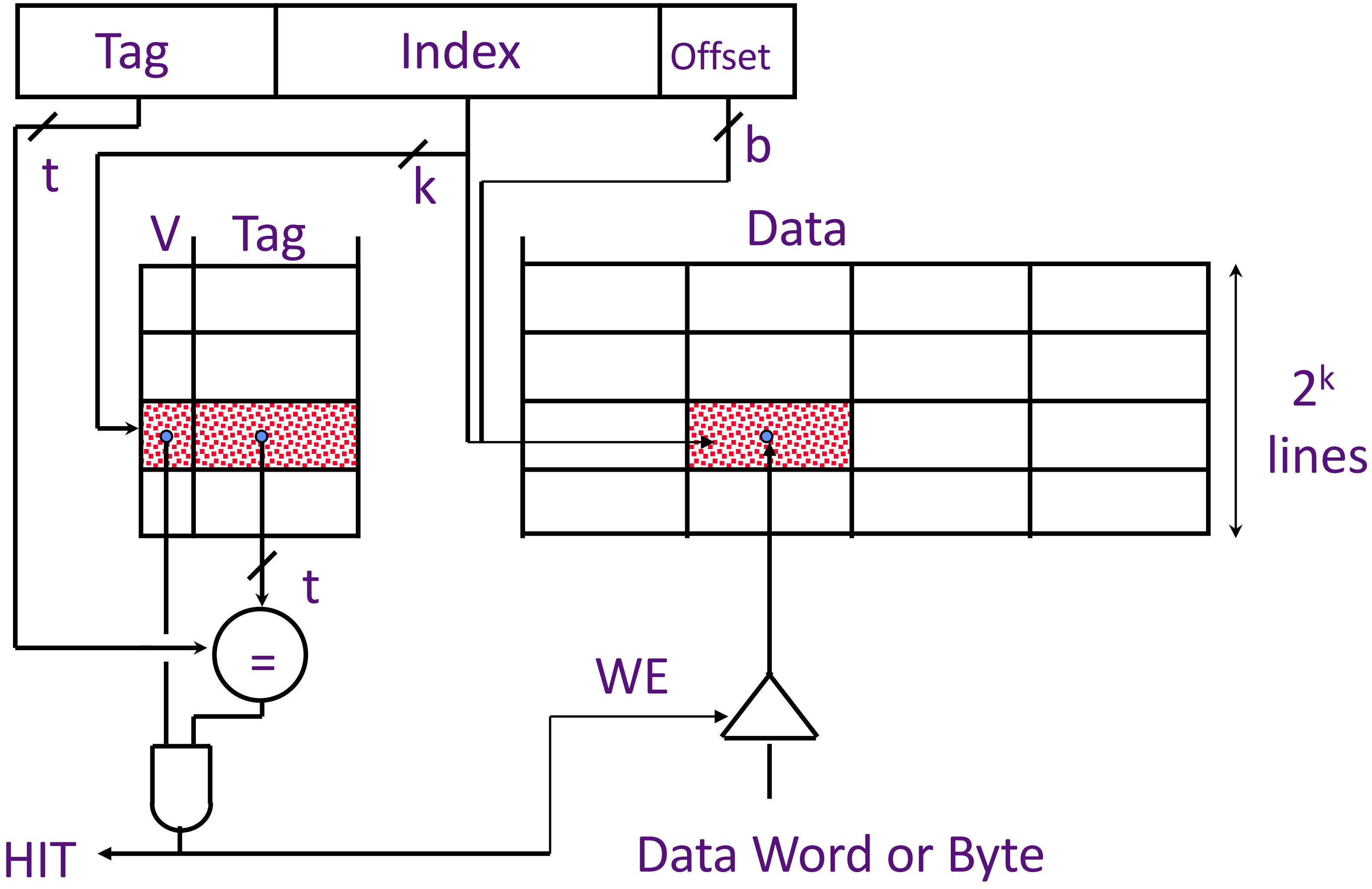


Figure B.10 Miss rate versus block size for five different-sized caches. Note that miss rate actually goes up if the block size is too large relative to the cache size. Each line represents a cache of different size. Figure B.11 shows the data used to plot these lines. Unfortunately, SPEC2000 traces would take too long if block size were included, so these data are based on SPEC92 on a DECstation 5000 (Gee et al. 1993).

Write Policy Choices

- Cache hit:
  - ***write-through***: write both cache & memory
    - Generally higher traffic but simpler pipeline & cache design
  - ***write-back***: write cache only, memory is written only when the entry is evicted
    - A dirty bit per line further reduces write-back traffic
    - Must handle 0, 1, or 2 accesses to memory for each load/store
- Cache miss:
  - ***no-write-allocate***: only write to main memory
  - ***write-allocate*** (aka fetch-on-write): fetch into cache
- Common combinations:
  - write-through and no-write-allocate
  - write-back with write-allocate

Write Performance



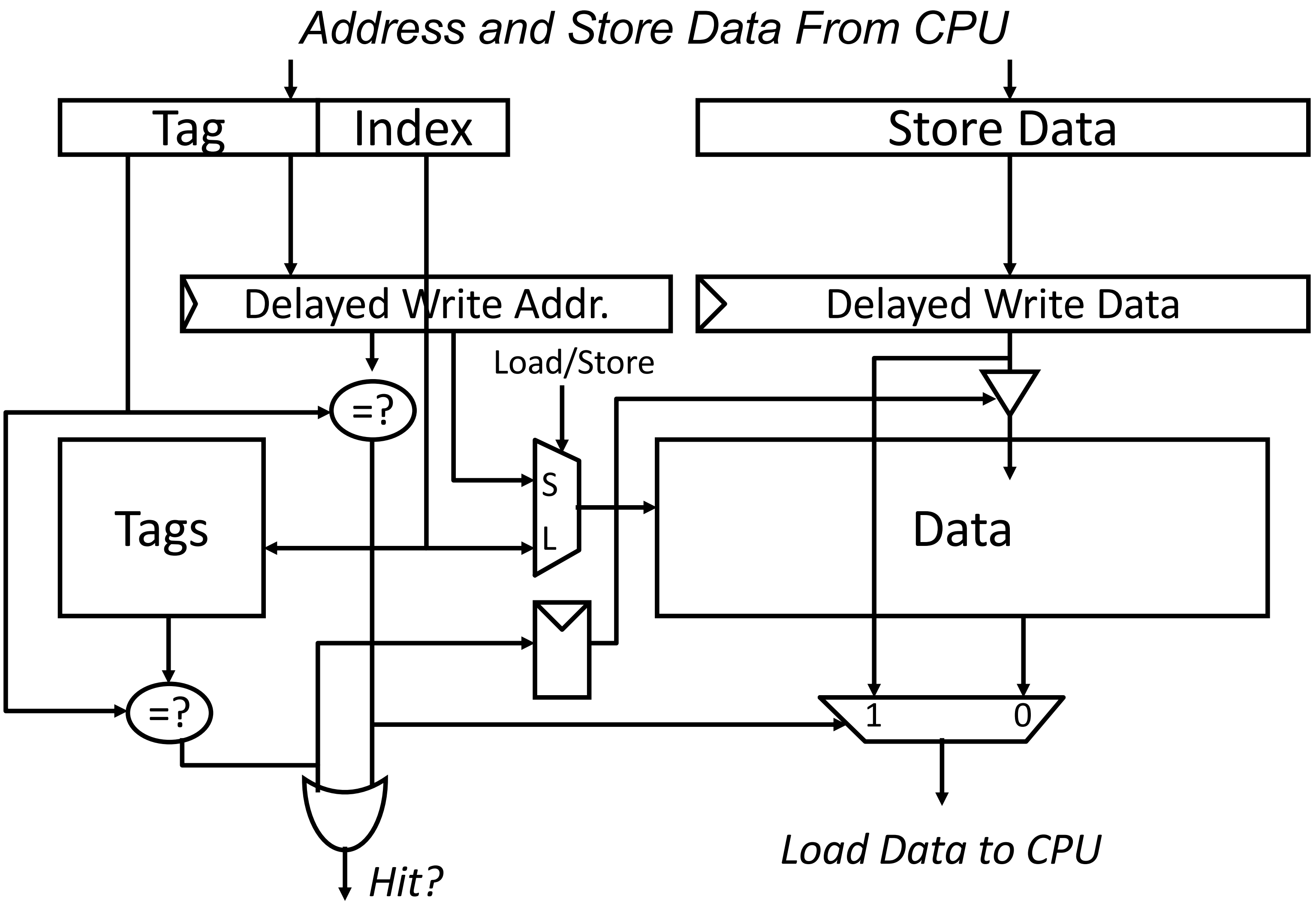
Reducing Write Hit Time

Problem: Writes take two cycles in memory stage, one cycle for tag check plus one cycle for data write if hit

Solutions:

- Design data RAM that can perform read and write in one cycle, restore old value after tag miss
- Pipelined writes: Hold write data for store in single buffer ahead of cache, write cache data during next store's tag check
- Fully-associative (CAM Tag) caches: Word line only enabled if hit

Pipelining Cache Writes



Data from a store hit is written into data portion of cache during tag access of subsequent store

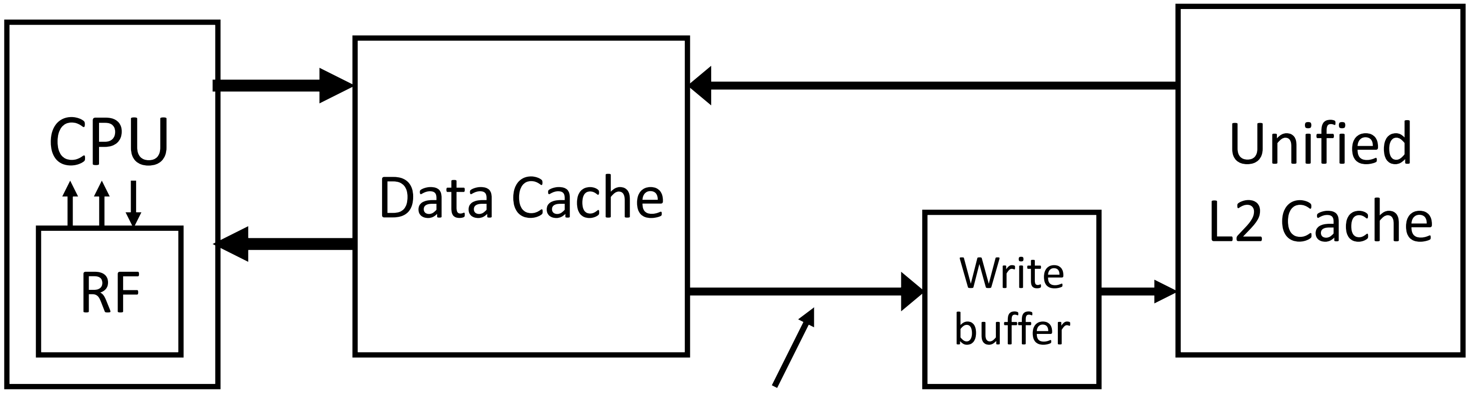
CS152 Administrivia

- PS 1 due 11:59PM today!
  - Solutions discussed in section on Friday
- Lab 1 due 11:59PM Wed Feb 17
- PS 2 out on Wednesday
- Wednesday: Special guest lecture on prefetching from Apple CPU designers
- Monday Feb 15 is President’s Day Holiday, ***no class!***

CS252 Administrivia

- Start thinking of class projects and forming teams of two
- Use Krste’s office hours to discuss ideas
- Proposal due Wednesday February 24th

Write Buffer to Reduce Read Miss Penalty



Evicted dirty lines for write-back cache
OR
All writes in write-through cache

Processor is not stalled on writes, and read misses can go ahead of write to main memory

Problem: Write buffer may hold updated value of location needed by a read miss

Simple solution: on a read miss, wait for the write buffer to go empty

Faster solution: Check write buffer addresses against read miss addresses, if no match, allow read miss to go ahead of writes, else, return value in write buffer

Reducing Tag Overhead with Sub-Blocks

- **Problem**: Tags are too large, i.e., too much overhead
  - Simple solution: Larger lines, but miss penalty could be large.
- **Solution**: Sub-block placement (aka sector cache)
  - A valid bit added to units smaller than full line, called sub-blocks
  - Only read a sub-block on a miss
  - *If a tag matches, is the word in the cache?*

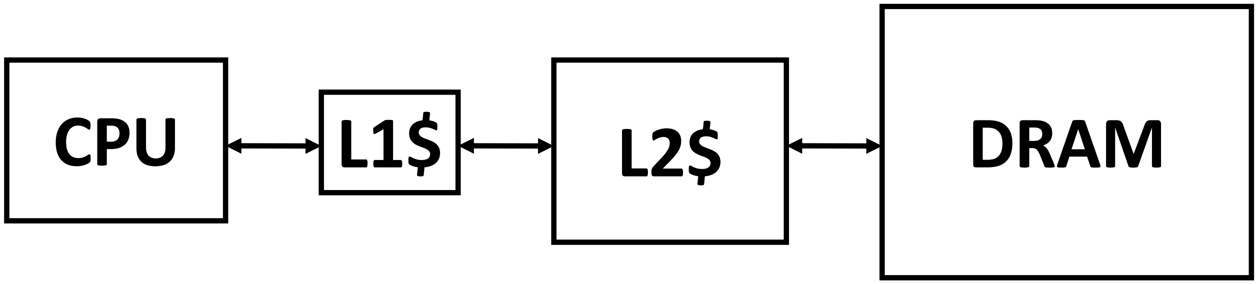
100
300
204

1		*1*		*1*		*1*	
1		*1*		*0*		*0*	
0		*1*		*0*		*1*	

Multilevel Caches

Problem: A memory cannot be large and fast

Solution: Increasing sizes of cache at each level



Local miss rate = misses in cache / accesses to cache

Global miss rate = misses in cache / CPU memory accesses

Misses per instruction = misses in cache / number of instructions

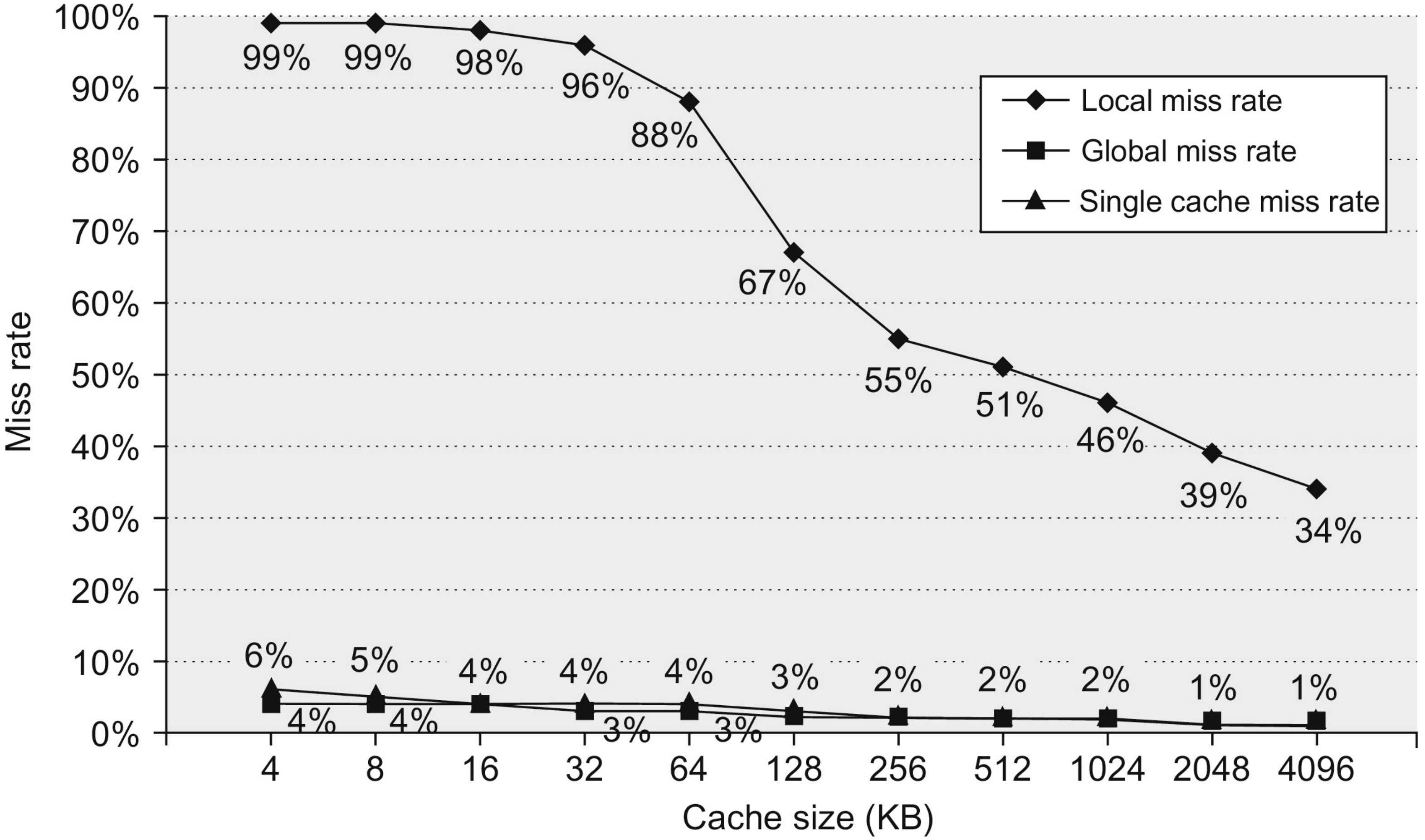


Figure B.14 Miss rates versus cache size for multilevel caches. Second-level caches *smaller* than the sum of the two 64 KiB first-level caches make little sense, as reflected in the high miss rates. After 256 KiB the single cache is within 10% of the global miss rates. The miss rate of a single-level cache versus size is plotted against the local miss rate and global miss rate of a second-level cache using a 32 KiB first-level cache. The L2 caches (unified) were two-way set associative with replacement. Each had split L1 instruction and data caches that were 64 KiB two-way set associative with LRU replacement. The block size for both L1 and L2 caches was 64 bytes. Data were collected as in Figure B.4.

Presence of L2 influences L1 design

- Use smaller L1 if there is also L2
  - Trade increased L1 miss rate for reduced L1 hit time
  - Backup L2 reduces L1 miss penalty
  - Reduces average access energy
- Use simpler write-through L1 with on-chip L2
  - Write-back L2 cache absorbs write traffic, doesn’t go off-chip
  - At most one L1 miss request per L1 access (no dirty victim write back) simplifies pipeline control
  - Simplifies coherence issues
  - Simplifies error recovery in L1 (can use just parity bits in L1 and reload from L2 when parity error detected on L1 read)

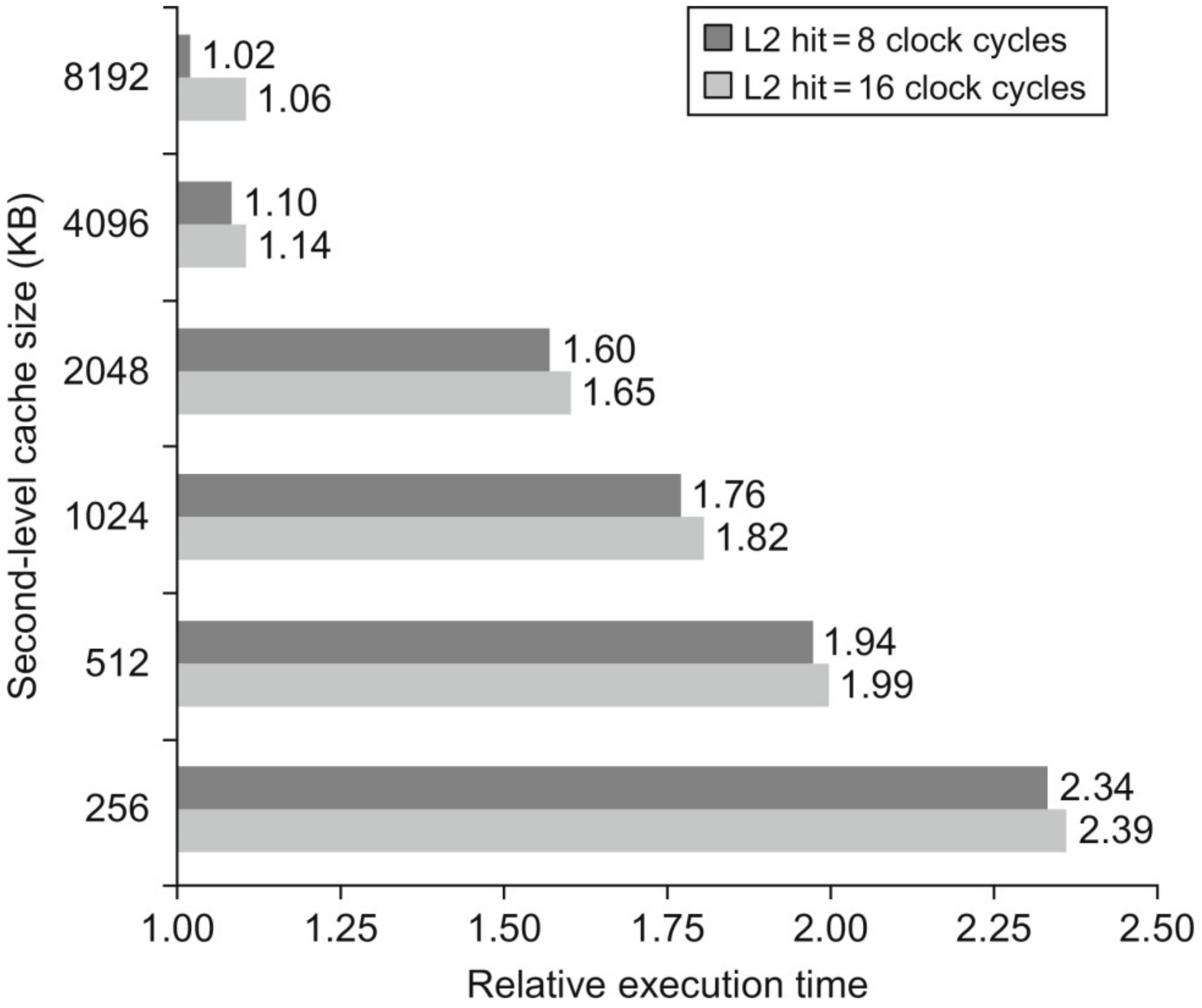


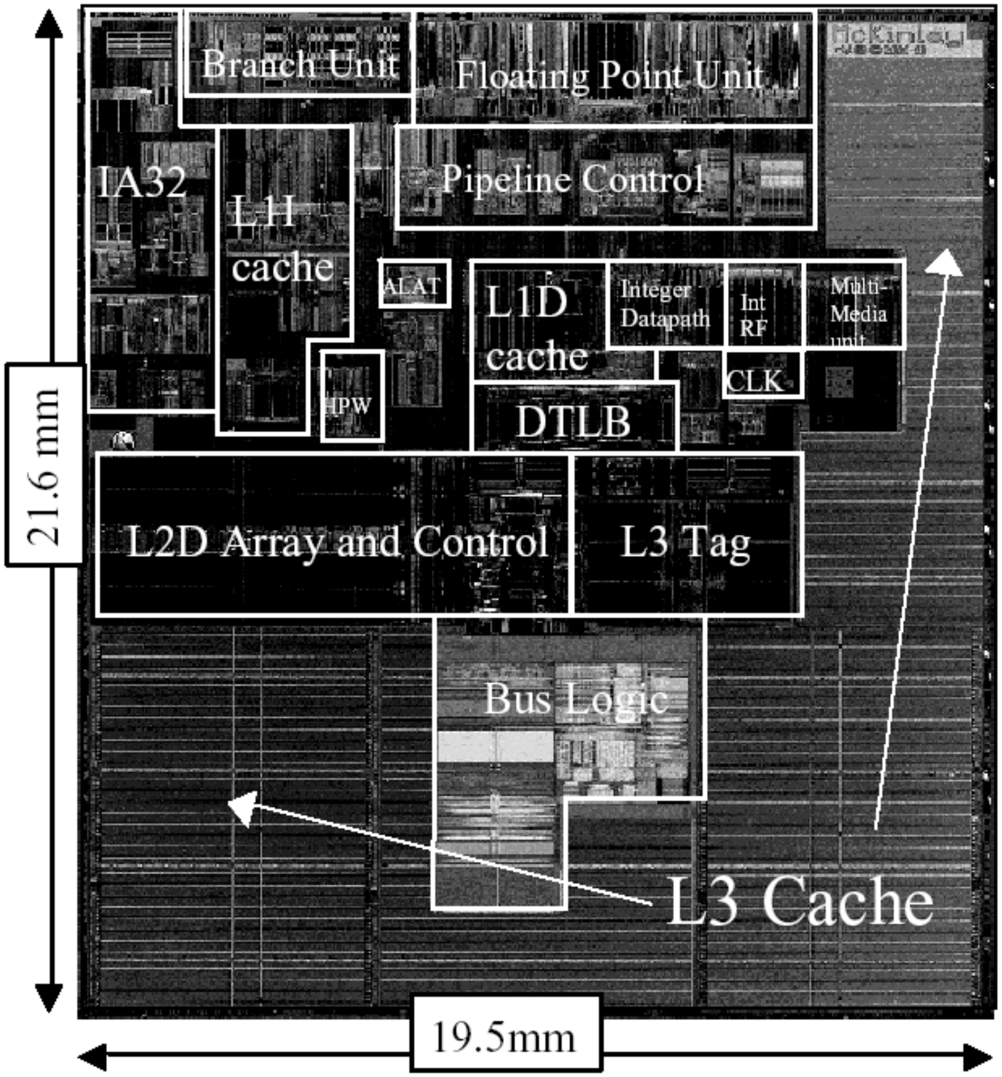
Figure B.15 Relative execution time by second-level cache size. The two bars are for different clock cycles for an L2 cache hit. The reference execution time of 1.00 is for an 8192 KiB second-level cache with a 1-clock-cycle latency on a second-level hit. These data were collected the same way as in Figure B.14, using a simulator to imitate the Alpha 21264.

Inclusion Policy

- Inclusive multilevel cache:
  - Inner cache can only hold lines also present in outer cache
  - External coherence snoop access need only check outer cache
- *Exclusive* multilevel caches:
  - Inner cache may hold lines not in outer cache
  - Swap lines between inner/outer caches on miss
  - Used in AMD Athlon with 64KB primary and 256KB secondary cache

Why choose one type or the other?

Itanium-2 On-Chip Caches
(Intel/HP, 2002)



Level 1: 16KB, 4-way s.a., 64B line, quad-port (2 load+2 store), single cycle latency

Level 2: 256KB, 4-way s.a, 128B line, quad-port (4 load or 4 store), five cycle latency

Level 3: 3MB, 12-way s.a., 128B line, single 32B port, twelve cycle latency

Power 7 On-Chip Caches [IBM 2009]

32KB L1 I$/core

32KB L1 D$/core

3-cycle latency

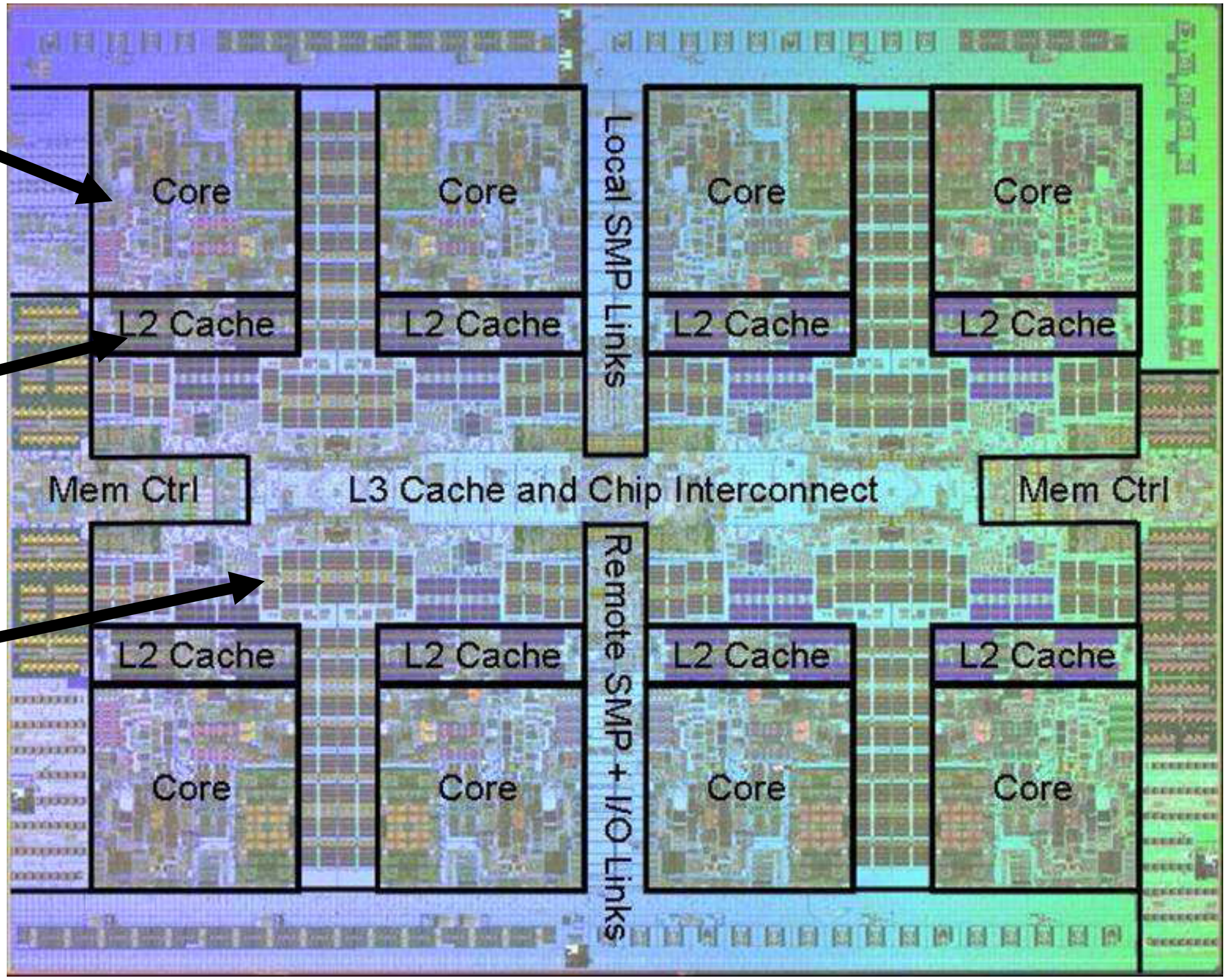
256KB Unified L2$/core

8-cycle latency

32MB Unified Shared L3$

Embedded DRAM (eDRAM)

25-cycle latency to local slice



IBM z196 Mainframe Caches 2010

- 96 cores (4 cores/chip, 24 chips/system)
  - Out-of-order, 3-way superscalar @ 5.2GHz
- L1: 64KB I-$/core + 128KB D-$/core
- L2: 1.5MB private/core (144MB total)
- L3: 24MB shared/chip (eDRAM) (576MB total)
- L4: 768MB shared/system (eDRAM)

Acknowledgements

- This course is partly inspired by previous MIT 6.823 and Berkeley CS252 computer architecture courses created by my collaborators and colleagues:
  - Arvind (MIT)
  - Joel Emer (Intel/MIT)
  - James Hoe (CMU)
  - John Kubiatowicz (UCB)
  - David Patterson (UCB)