CS 152 Computer Architecture and Engineering

Lecture 7 - Memory Hierarchy-II

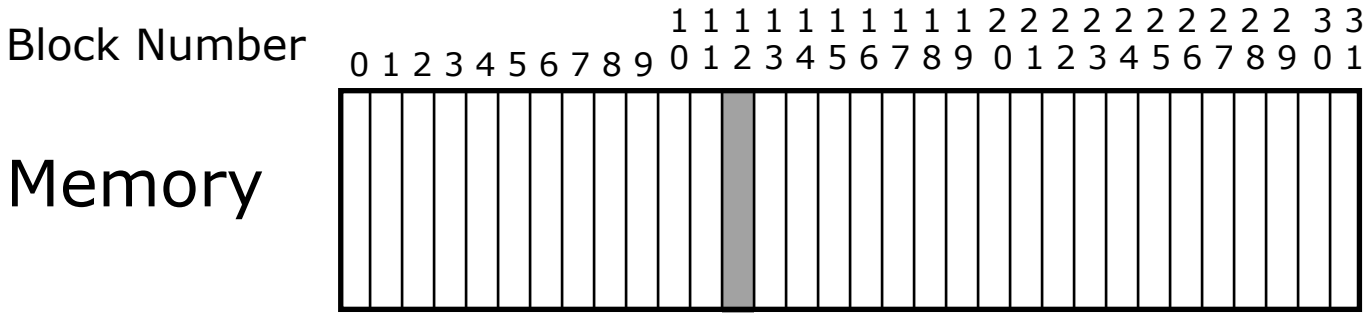
Krste Asanovic
Electrical Engineering and Computer Sciences
University of California at Berkeley

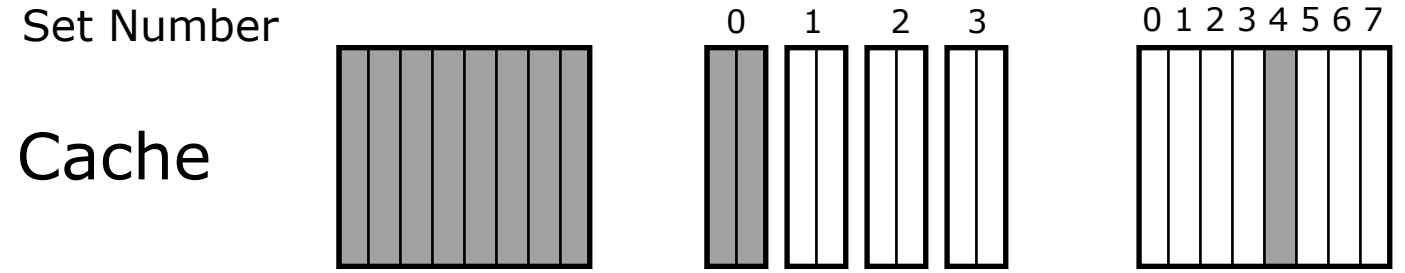
http://www.eecs.berkeley.edu/~krste
http://inst.eecs.berkeley.edu/~cs152

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

Placement Policy





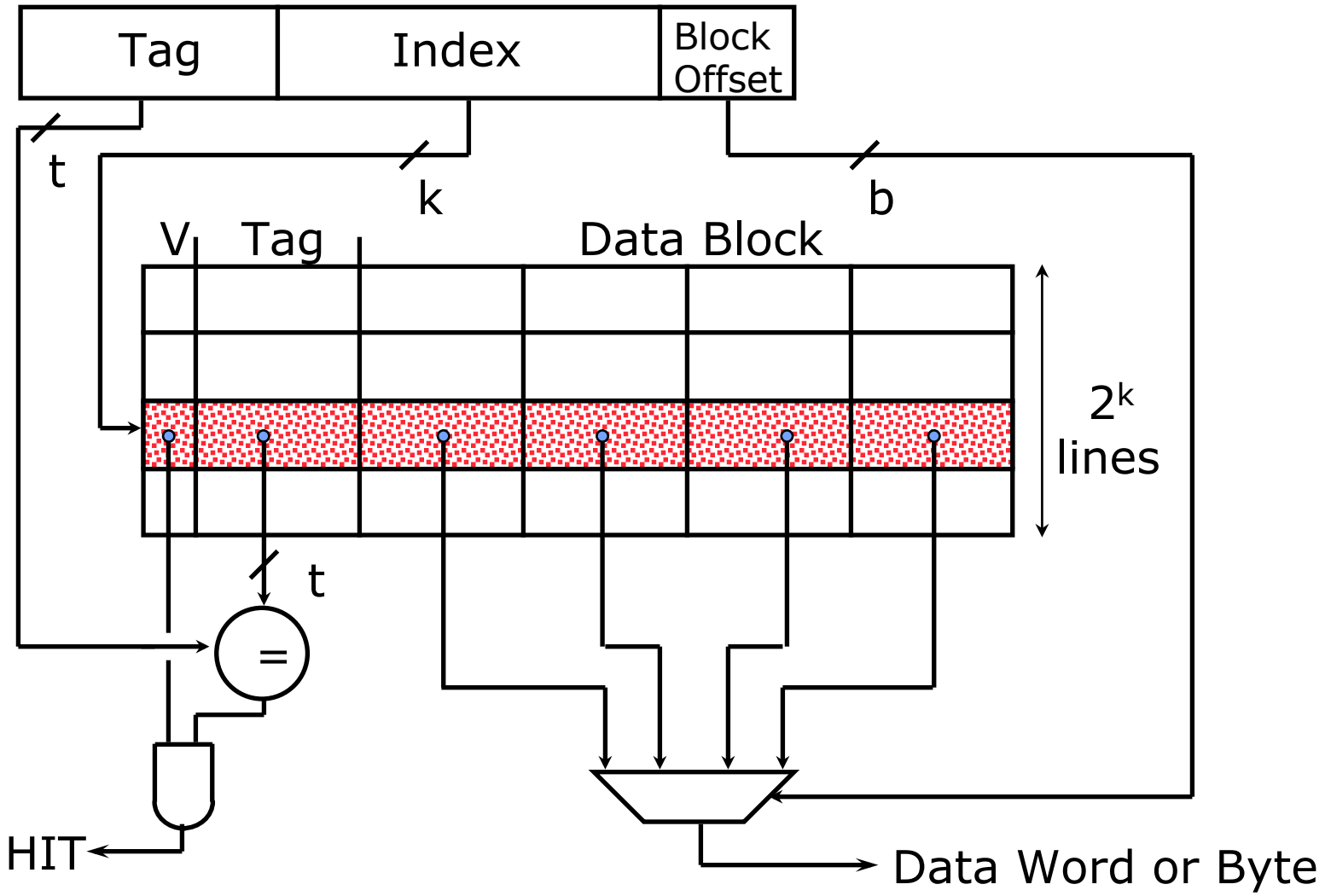
block 12
can be placed

Fully Associative
anywhere

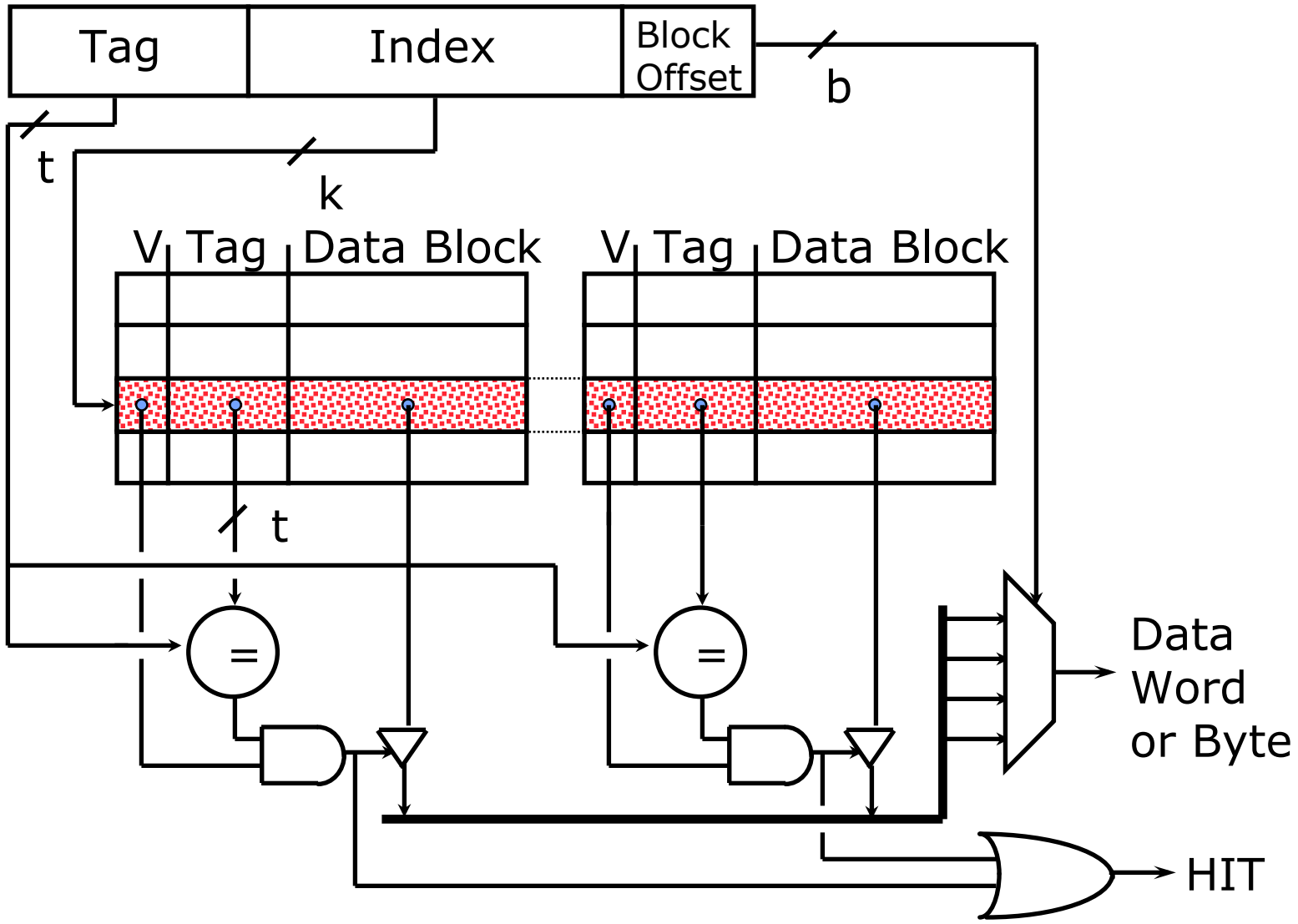
(2-way) Set Associative
anywhere in set 0
(12 mod 4)

Direct Mapped
only into block 4
(12 mod 8)

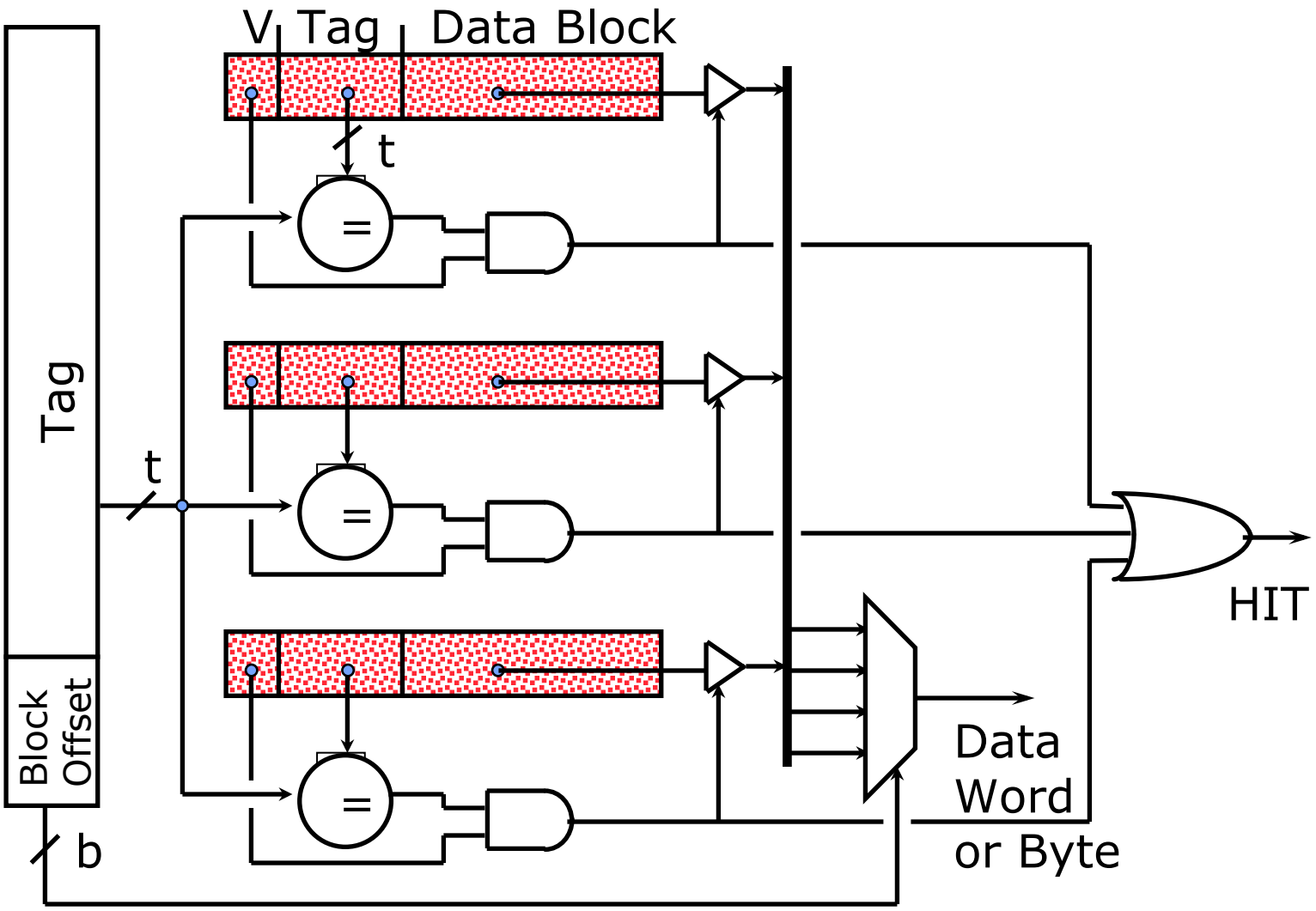
Direct-Mapped Cache



2-Way Set-Associative Cache



Fully Associative Cache



Replacement Policy

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

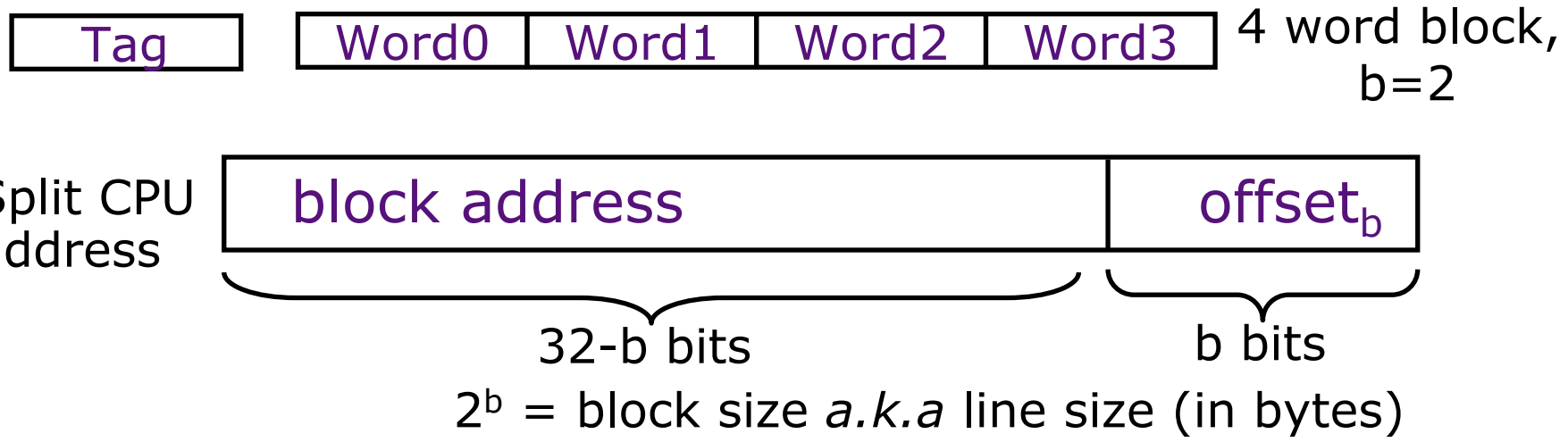
- Random
- Least Recently Used (LRU)
  - LRU cache state must be updated on every access
  - true implementation only feasible for small sets (2-way)
  - pseudo-LRU binary tree often used for 4-8 way
- First In, First Out (FIFO) a.k.a. Round-Robin
  - used in highly associative caches
- Not Most Recently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

This is a second-order effect. Why?

Replacement only happens on misses

Block Size and Spatial Locality

Block is unit of transfer between the cache and memory



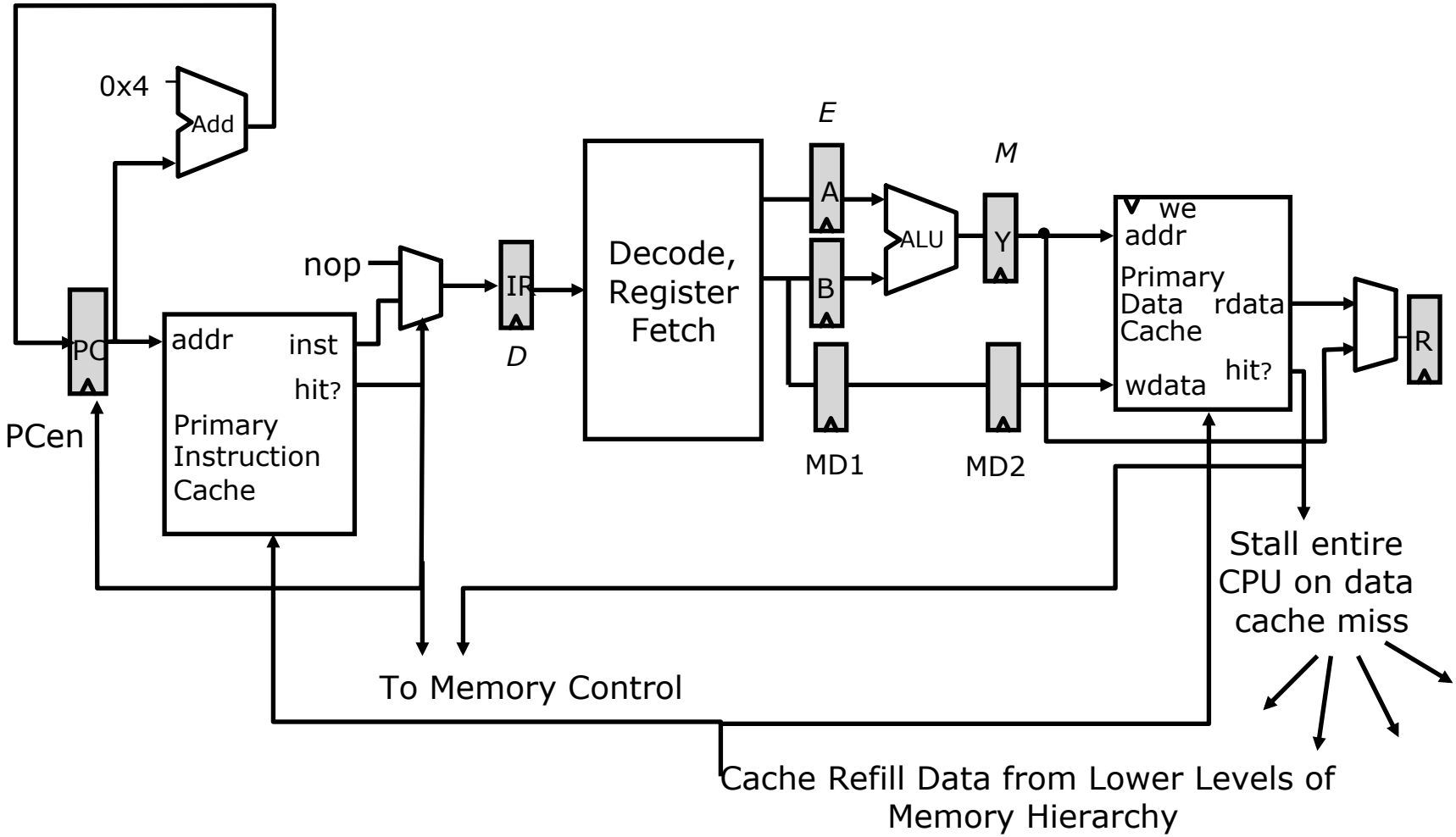
Larger block size has distinct hardware advantages

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

What are the disadvantages of increasing block size?

Fewer blocks => more conflicts. Can waste bandwidth.

CPU-Cache Interaction
(5-stage pipeline)



Improving Cache Performance

Average memory access time =
Hit time + Miss rate x Miss penalty

To improve performance:

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

What is the simplest design strategy?

Biggest cache that doesn't increase hit time past 1-2 cycles (approx 8-32KB in modern technology)

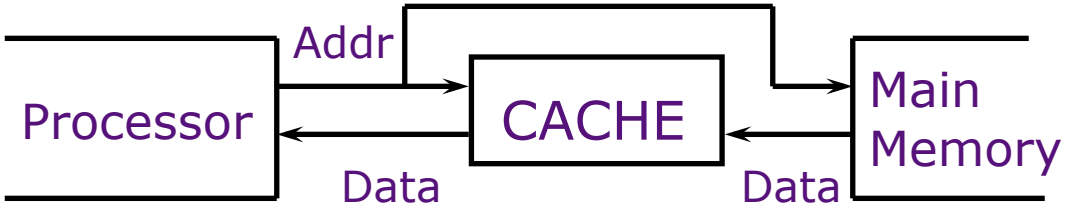
[design issues more complex with out-of-order superscalar processors]

Serial-versus-Parallel Cache and Memory access

α is HIT RATIO: Fraction of references in cache

1 - α is MISS RATIO: Remaining references

Average access time for serial search: $t_{cache} + (1 - \alpha)\, t_{mem}$



Average access time for parallel search: $\alpha\, t_{cache} + (1 - \alpha)\, t_{mem}$

- Savings are usually small, $t_{mem} >> t_{cache}$, hit ratio α high
- High bandwidth required for memory path
- Complexity of handling parallel paths can slow t_{cache}

Causes for Cache Misses

- *Compulsory:* first-reference to a block *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 block-placement strategy
  - misses that would not occur with 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 block size
  + reduces compulsory and capacity (reload) misses
  - increases conflict misses and miss penalty

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 block 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

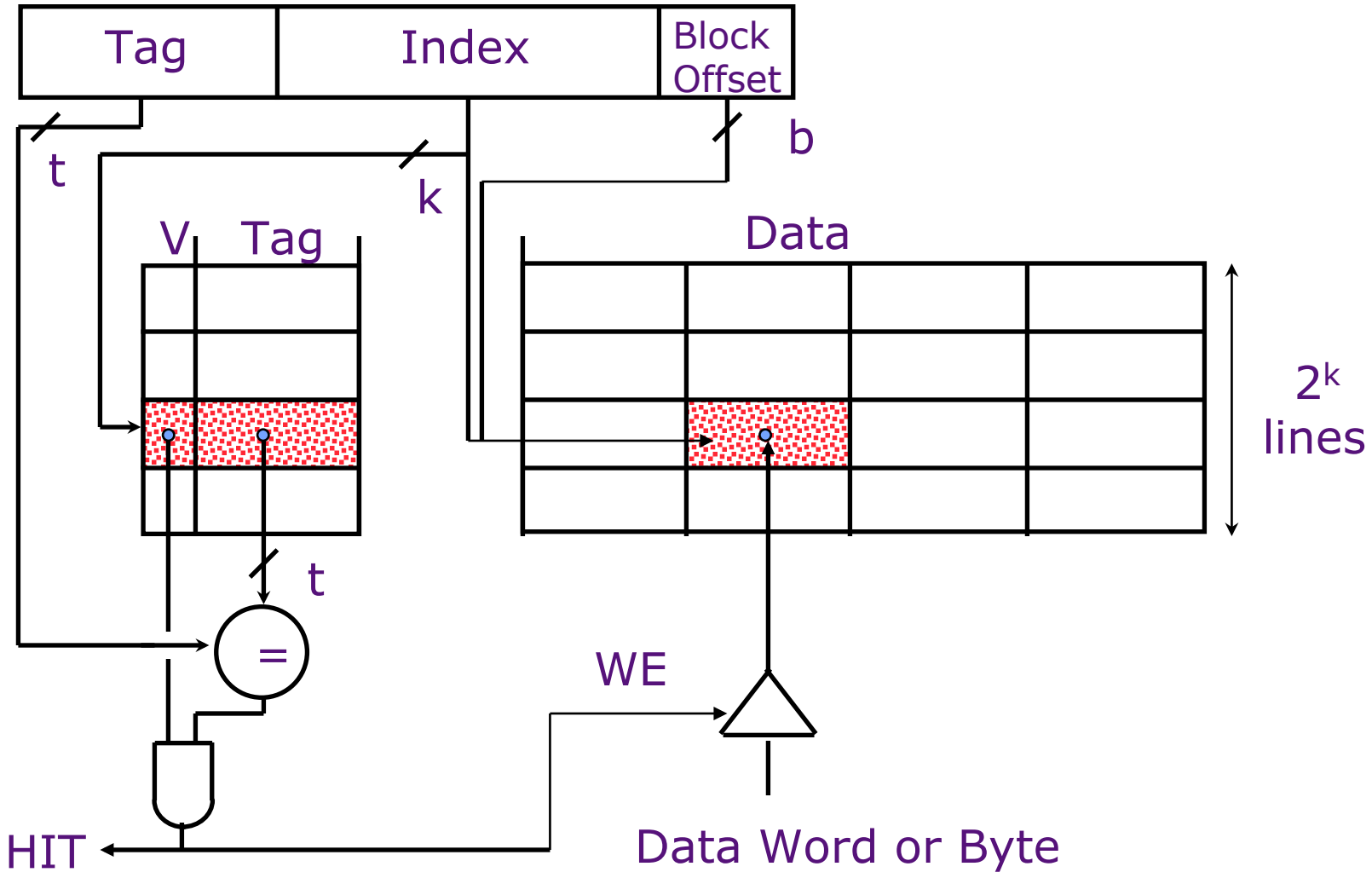
CS152 Administrivia

PS1/Lab1 due at start of class on Thursday

Krste's office hours now 2-3pm, but Monday is a UCB holiday so no office hours.

Quiz 1 on Tuesday.

Write Performance



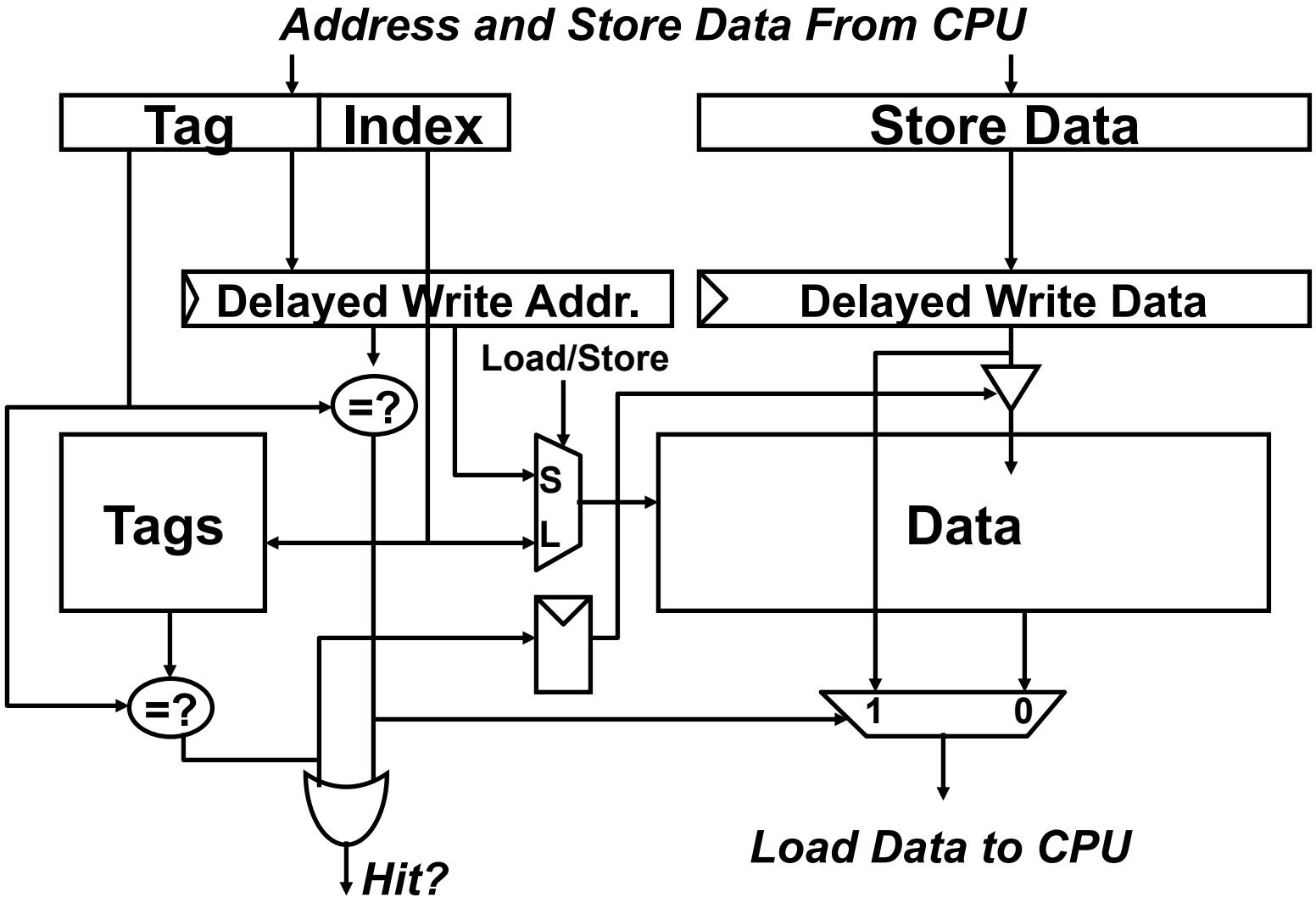
Reducing Write Hit Time

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

Solutions:

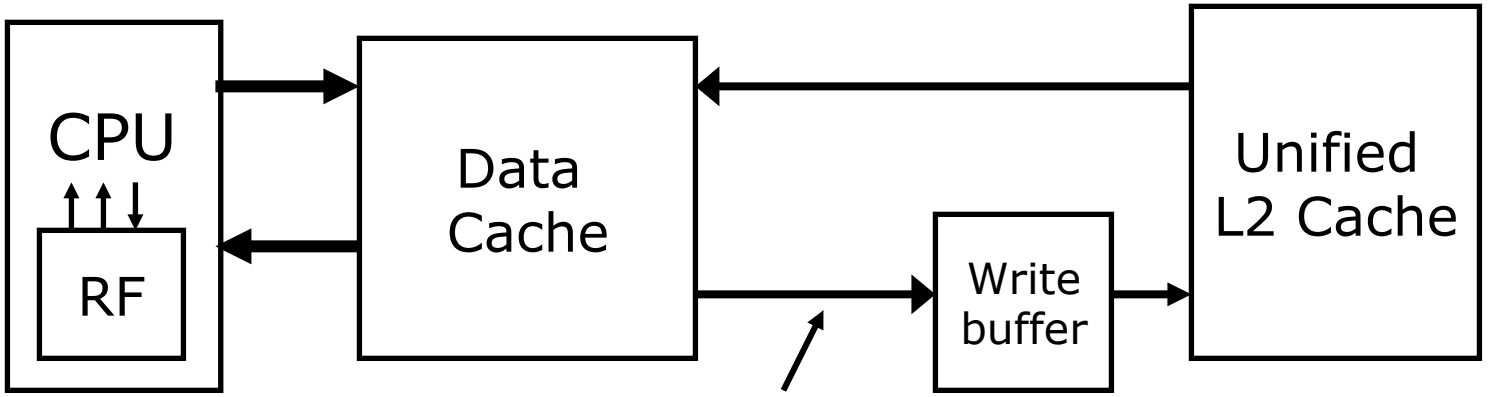
- Design data RAM that can perform read and write in one cycle, restore old value after tag miss
- Fully-associative (CAM Tag) caches: Word line only enabled if hit
- Pipelined writes: Hold write data for store in single buffer ahead of cache, write cache data during next store's tag check

Pipelining Cache Writes



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

Write Buffer to Reduce Read Miss Penalty



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

Faster scheme: 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

Block-level Optimizations

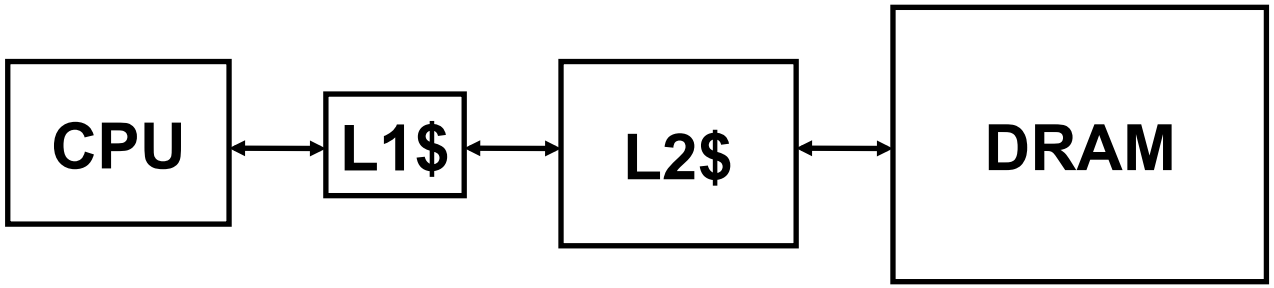
- Tags are too large, i.e., too much overhead
  - Simple solution: Larger blocks, but miss penalty could be large.
- Sub-block placement (aka sector cache)
  - A valid bit added to units smaller than full block, called sub-blocks
  - Only read a sub-block on a miss
  - *If a tag matches, is the word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

100	1		1		1		1	
300	1		1		0		0	
204	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

Presence of L2 influences L1 design

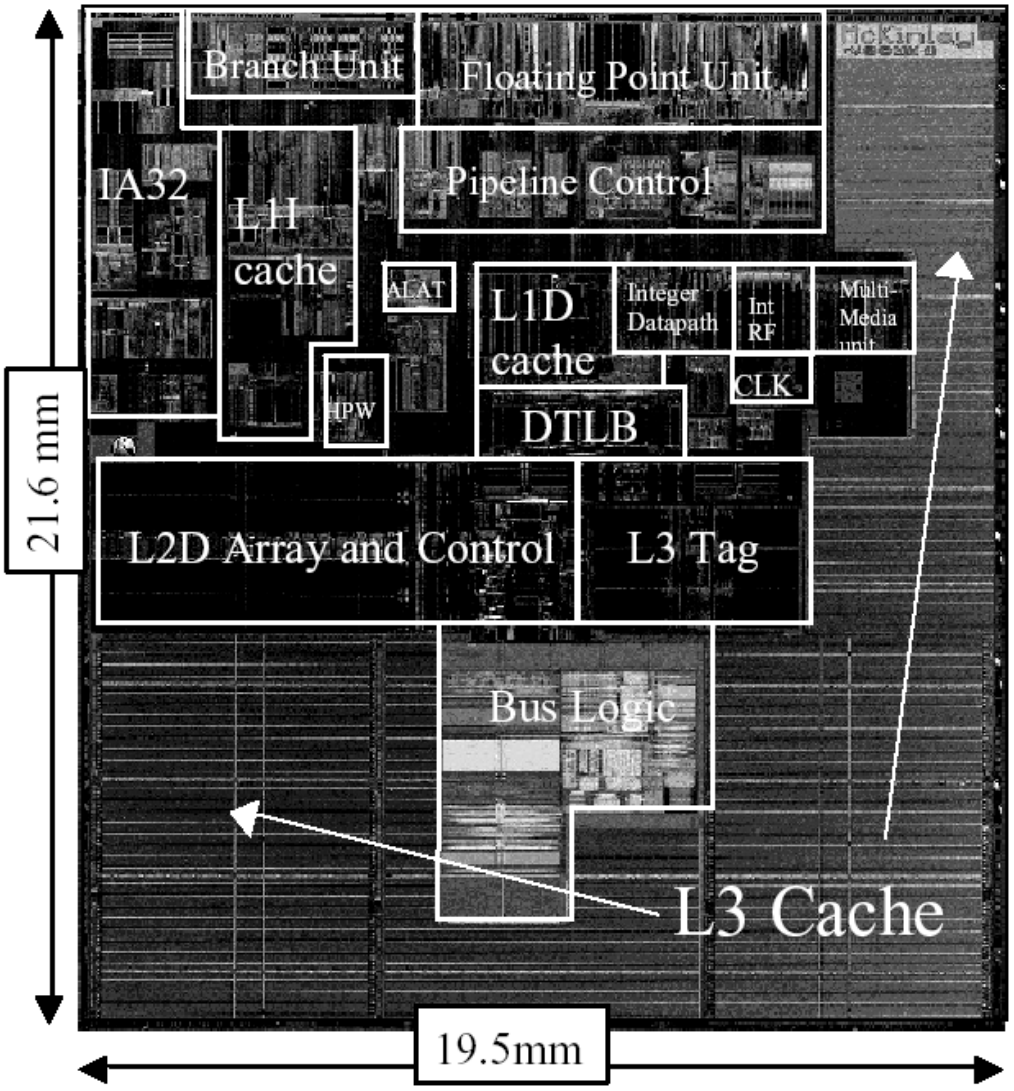
- Use smaller L1 if there is also L2
  - Trade increased L1 miss rate for reduced L1 hit time and reduced L1 miss penalty
  - Reduces average access energy
- Use simpler write-through L1 with on-chip L2
  - Write-back L2 cache absorbs write traffic, doesn't go off-chip
  - At most one L1 miss request 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)

Inclusion Policy

- Inclusive multilevel cache:
  - Inner cache holds copies of data in outer cache
  - External coherence snoop access need only check outer cache
- *Exclusive* multilevel caches:
  - Inner cache may hold data 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

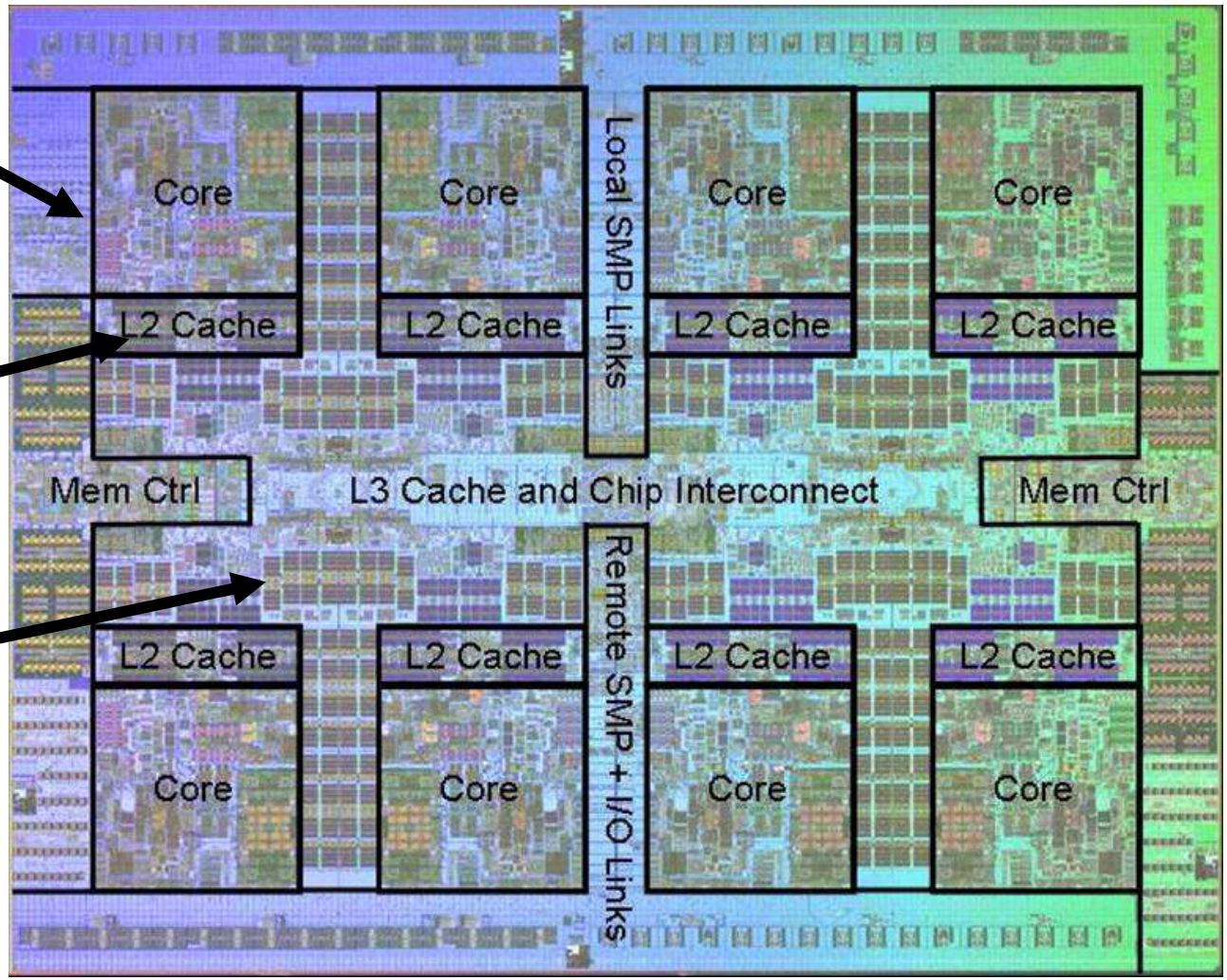
256KB Unified L2$/core

8-cycle latency

32MB Unified Shared L3$

Embedded DRAM

25-cycle latency to local slice



Acknowledgements

- These slides contain material developed and copyright by:
  - Arvind (MIT)
  - Krste Asanovic (MIT/UCB)
  - Joel Emer (Intel/MIT)
  - James Hoe (CMU)
  - John Kubiatowicz (UCB)
  - David Patterson (UCB)

- MIT material derived from course 6.823
- UCB material derived from course CS252