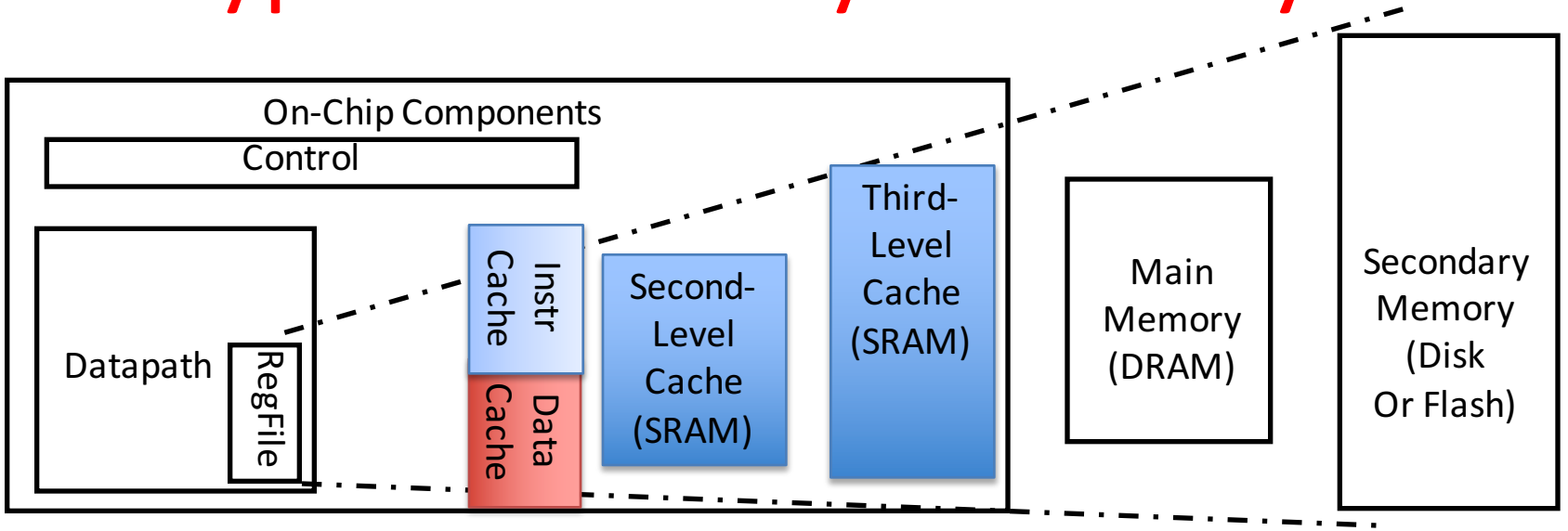
CS 61C: Great Ideas in Computer Architecture (Machine Structures) Caches Part 2

Instructors:

John Wawrzynek & Vladimir Stojanovic

http://inst.eecs.berkeley.edu/~cs61c/

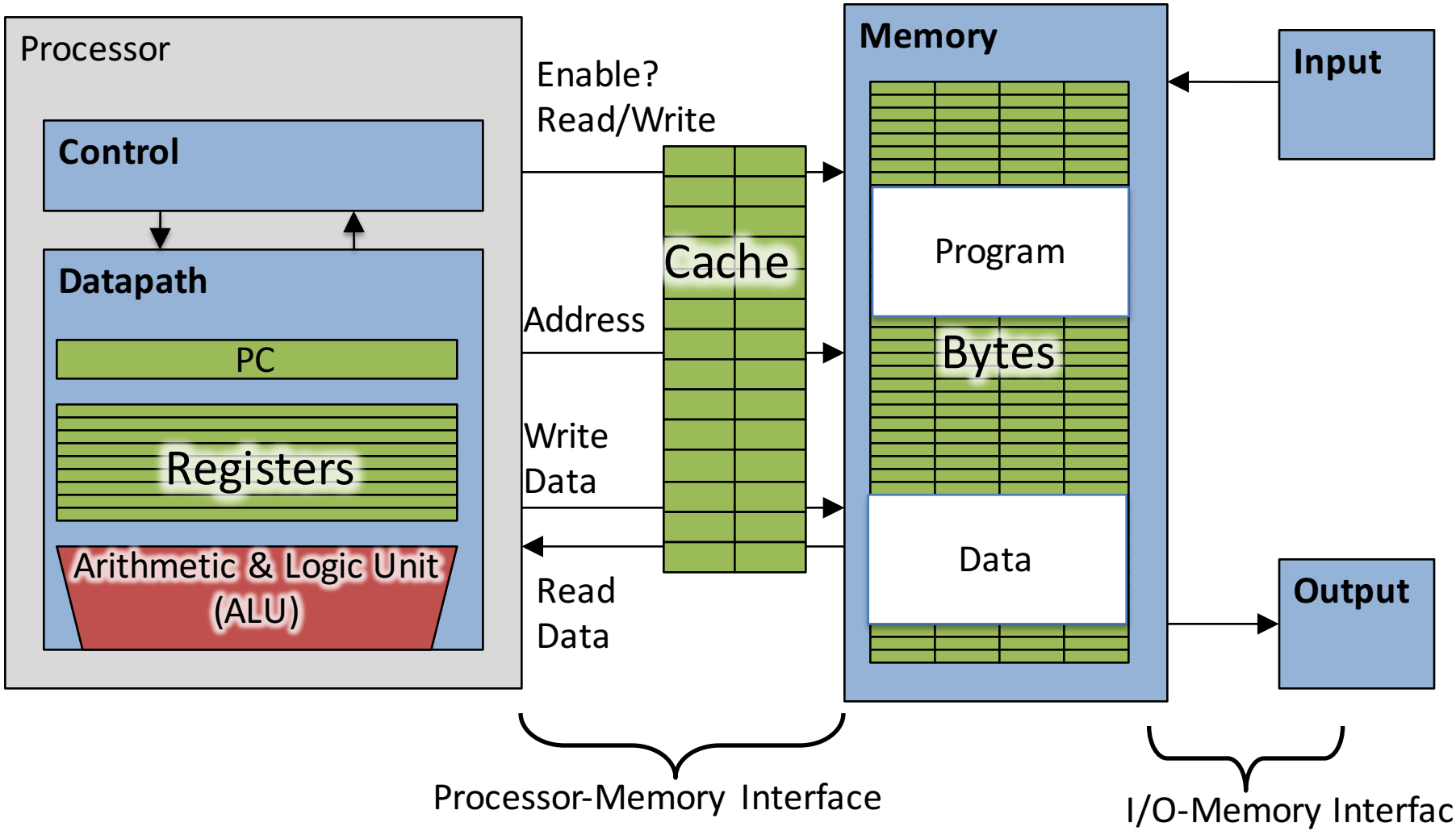
Typical Memory Hierarchy



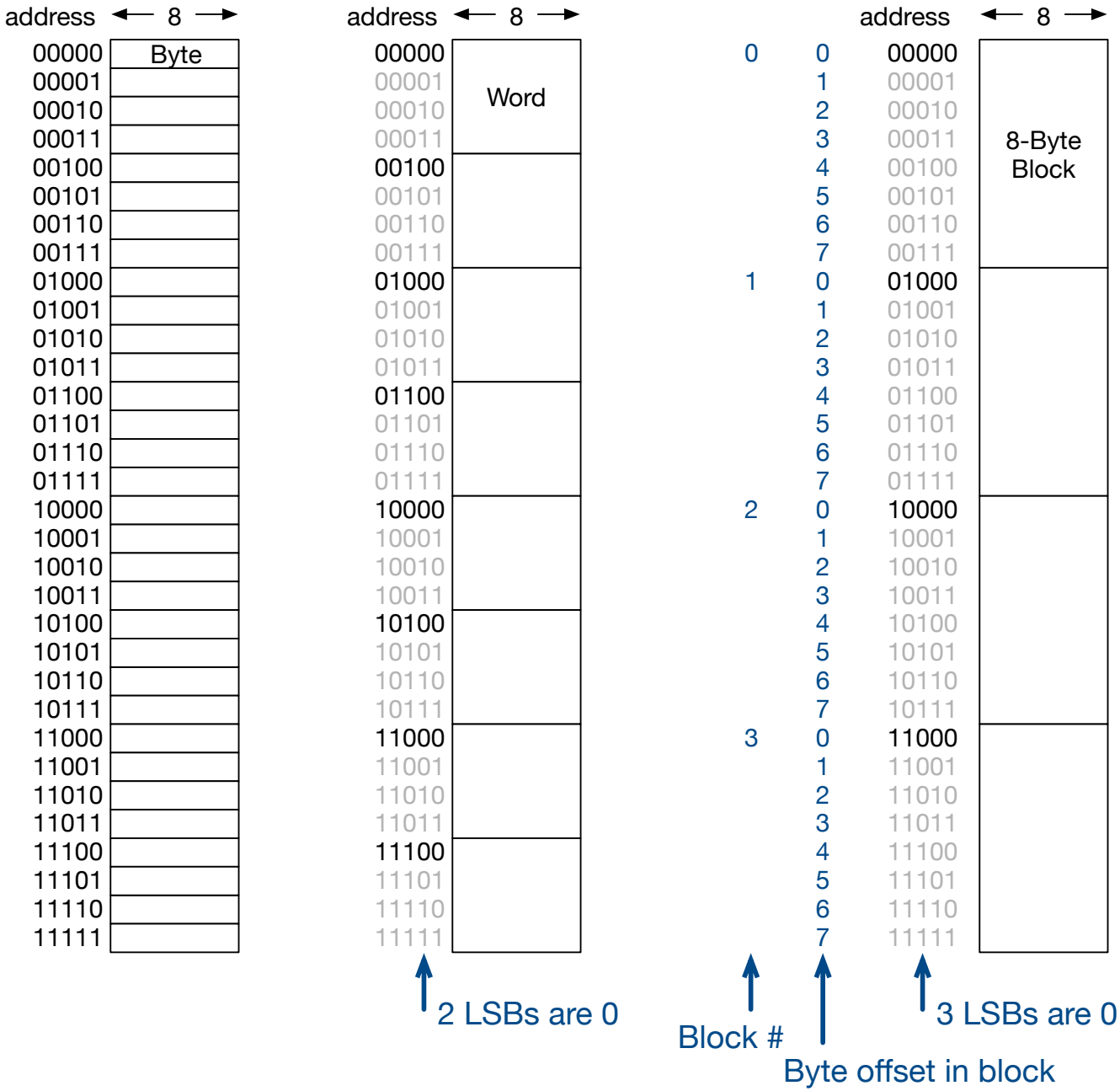
Speed (cycles):	½'s	1's	10's	100's	1,000,000's
Size (bytes):	100's	10K's	M's	G's	T's
Cost/bit:	highest	←		→	lowest

- Principle of locality + memory hierarchy presents programmer with
  ≈ as much memory as is available in the *cheapest* technology at the
  ≈ speed offered by the *fastest* technology

Review: Adding Cache to Computer

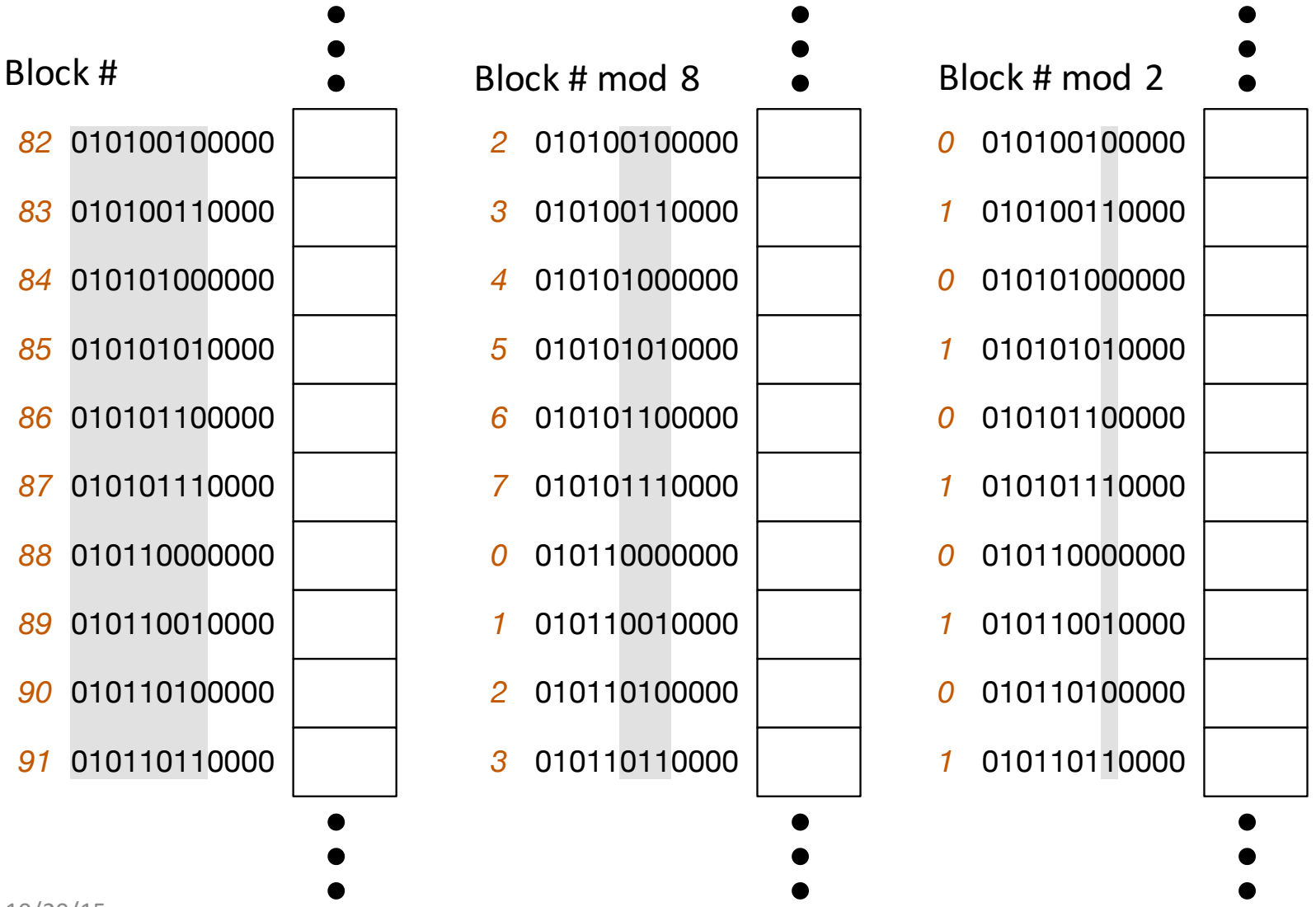


Memory Block-addressing example



Block number aliasing example

12-bit memory addresses, 16 Byte blocks



Caches Review

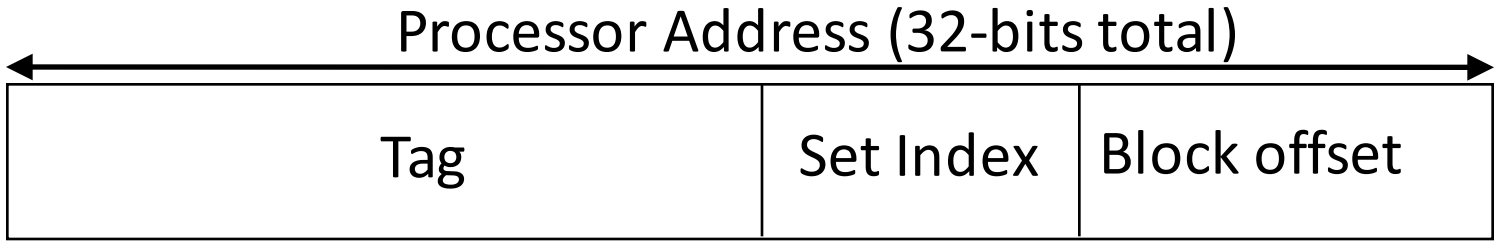
- Principle of Locality
  - Temporal Locality and Spatial Locality
- Hierarchy of Memories (speed/size/cost per bit) to Exploit Locality
- Cache – copy of data in lower level of memory hierarchy
- Direct Mapped to find block in cache using Tag field and Valid bit for Hit
- Cache design organization choices:
  - Fully Associative, Set-Associative, Direct-Mapped

Cache Organizations

- "Fully Associative": Block can go anywhere
  - First design in lecture
  - Note: No Index field, but 1 comparator/block
- "Direct Mapped": Block goes one place
  - Note: Only 1 comparator
  - Number of sets = number blocks
- "N-way Set Associative": N places for a block
  - Number of sets = number of blocks / N
  - N comparators
  - ***Fully Associative: N = number of blocks***
  - ***Direct Mapped: N = 1***

Processor Address Fields used by Cache Controller

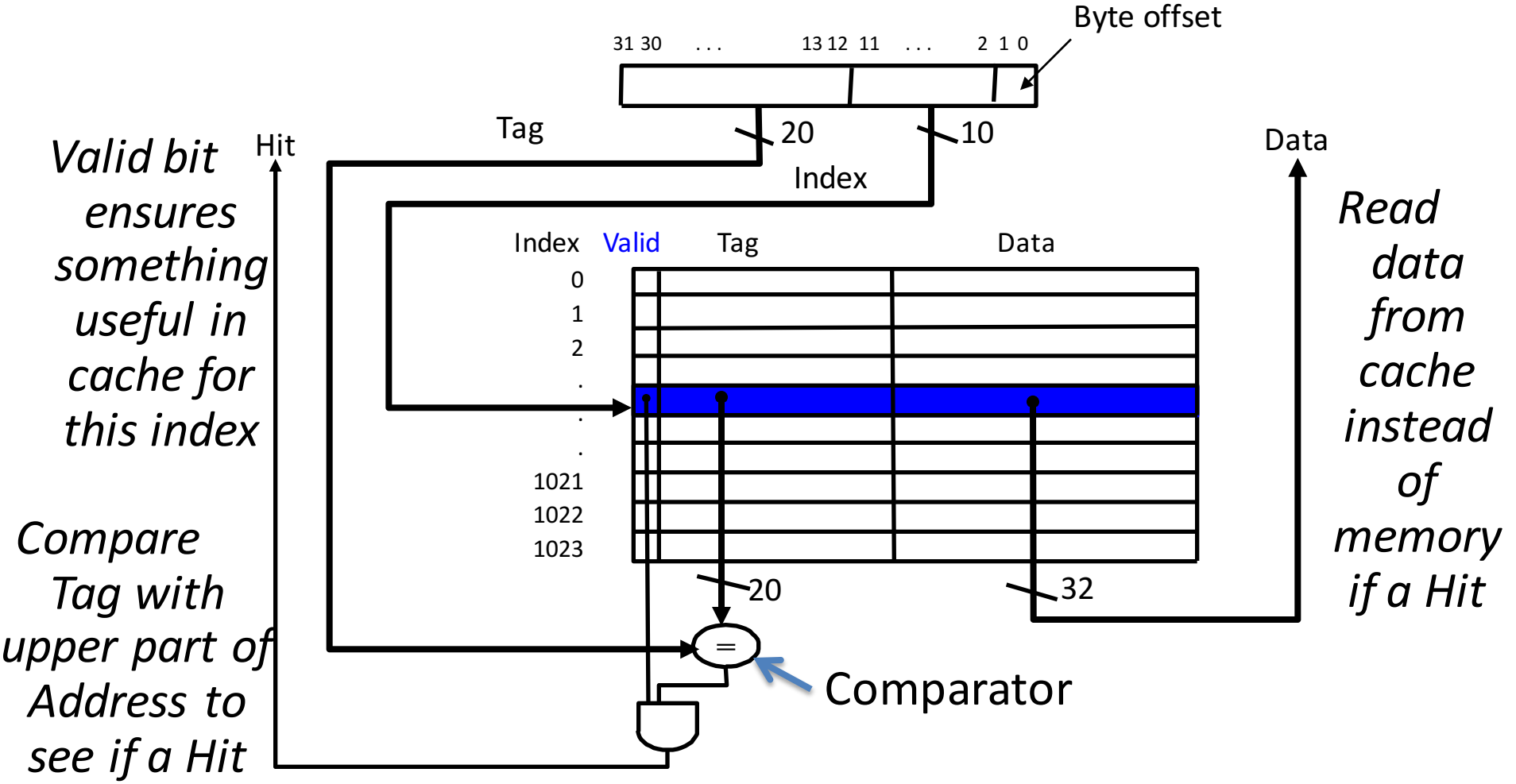
- Block Offset: Byte address within block
- Set Index: Selects which set
- Tag: Remaining portion of processor address



- Size of Index = log2 (number of sets)
- Size of Tag = Address size – Size of Index – log2 (number of bytes/block)

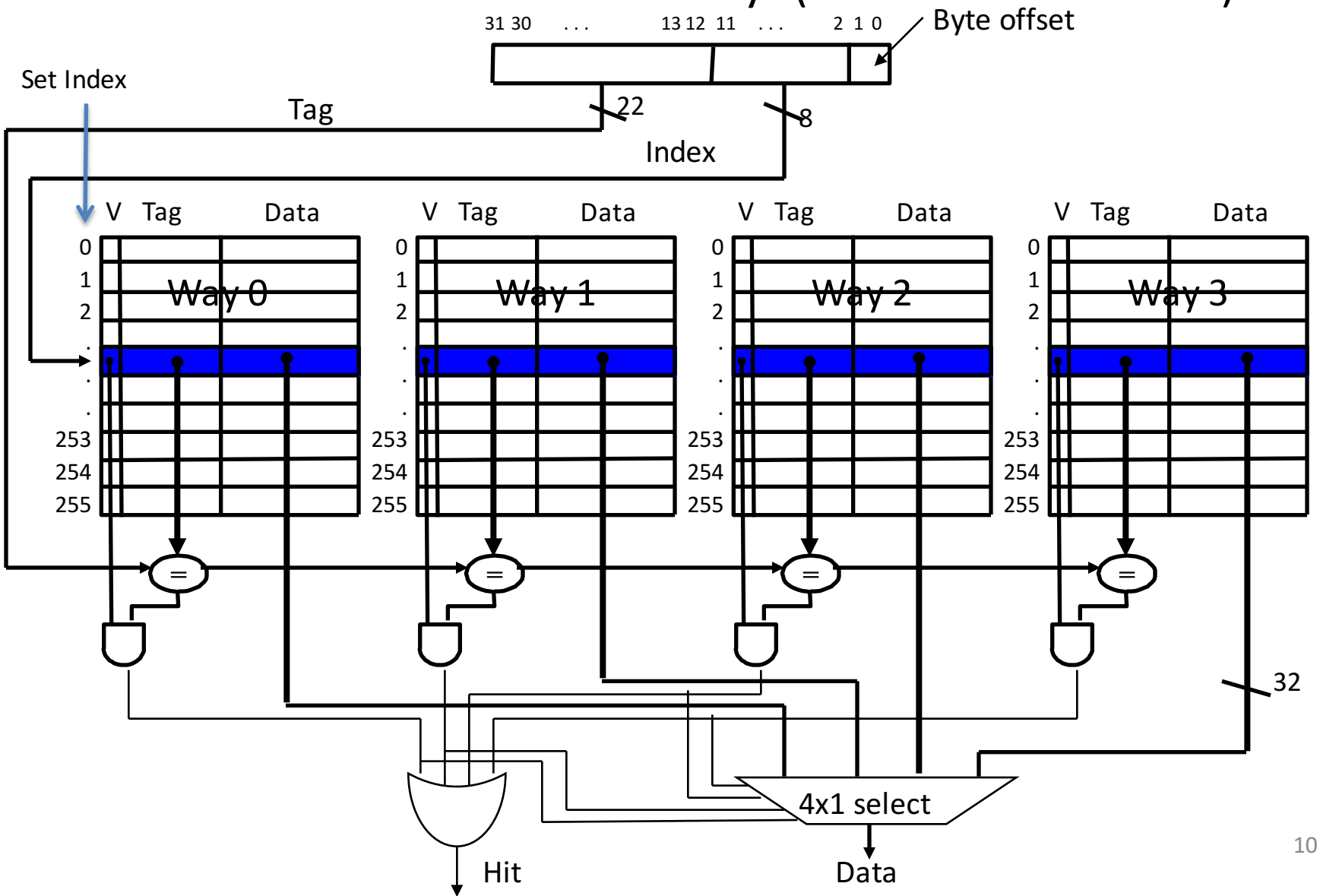
Direct-Mapped Cache Review

- One word blocks, cache size = 1K words (or 4KB)



Four-Way Set-Associative Cache

- 2^8 = 256 sets each with four ways (each with one block)



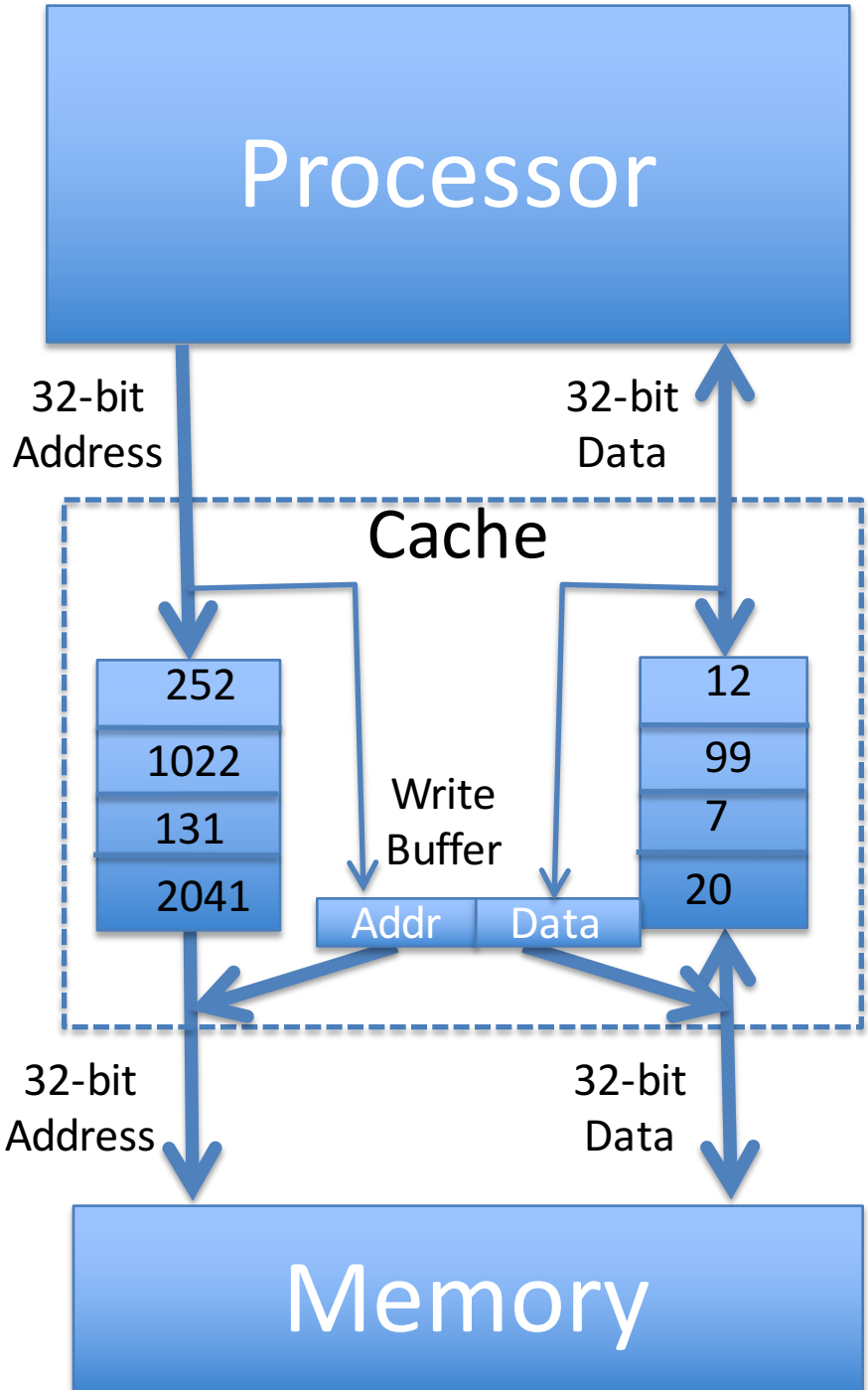
Handling Stores with Write-Through

- Store instructions write to memory, changing values
- Need to make sure cache and memory have same values on writes: 2 policies

1) Write-Through Policy: write cache and write *through* the cache to memory
   - Every write eventually gets to memory
   - Too slow, so include Write Buffer to allow processor to continue once data in Buffer
   - Buffer updates memory in parallel to processor

Write-Through Cache

- Write both values in cache and in memory
- Write buffer stops CPU from stalling if memory cannot keep up
- Write buffer may have multiple entries to absorb bursts of writes
- What if store misses in cache?



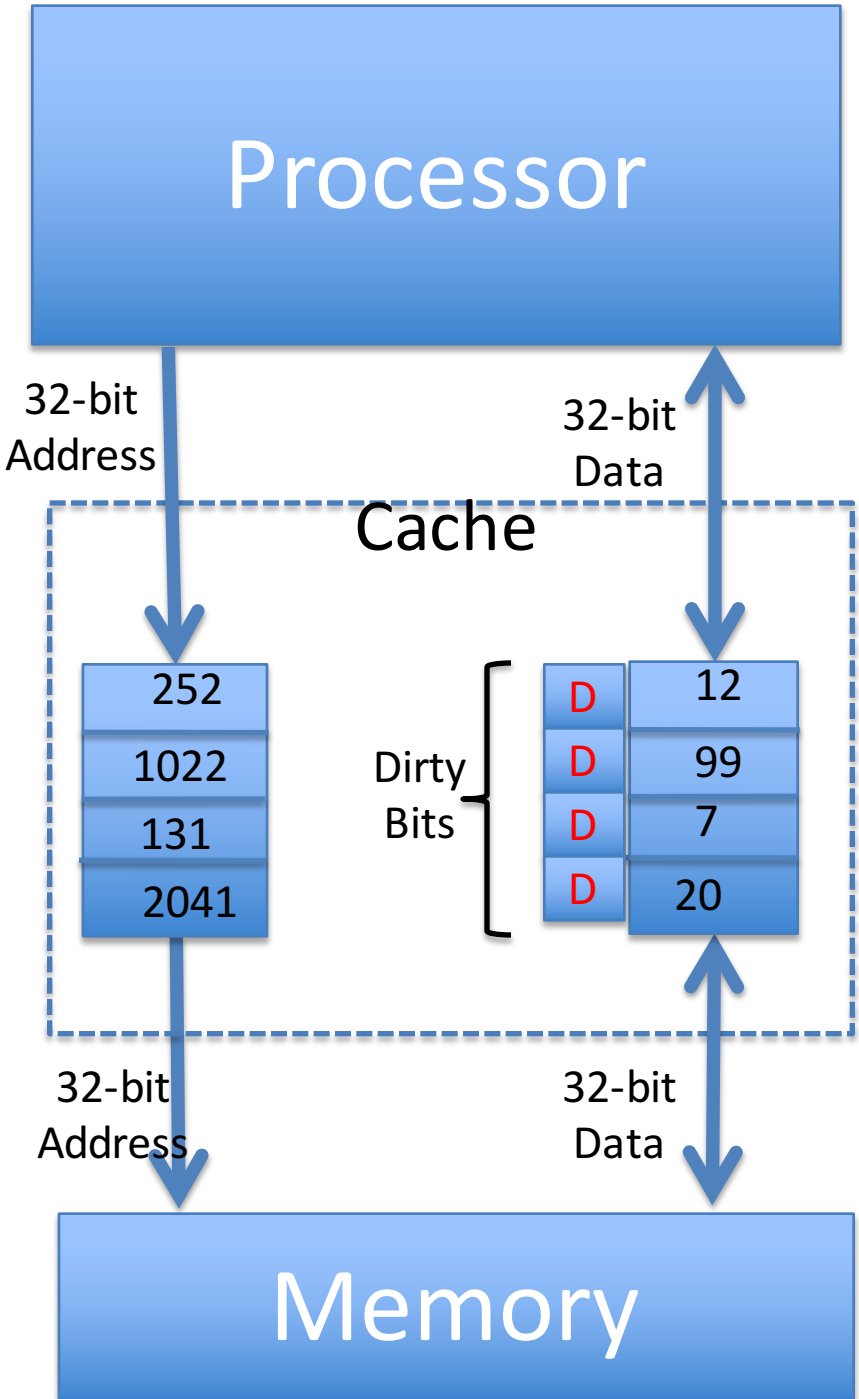
Handling Stores with Write-Back

2) Write-Back Policy: write only to cache and then write cache block *back* to memory when evict block from cache

- Writes collected in cache, only single write to memory per block
- Include bit to see if wrote to block or not, and then only write back if bit is set
  - Called "Dirty" bit (writing makes it "dirty")

Write-Back Cache

- Store/cache hit, write data in cache *only* & set dirty bit
  - Memory has stale value
- Store/cache miss, read data from memory, then update and set dirty bit
  - "Write-allocate" policy
- Load/cache hit, use value from cache
- On any miss, write back evicted block, only if dirty. Update cache with new block and clear dirty bit.



Write-Through vs. Write-Back

- Write-Through:
  - Simpler control logic
  - More predictable timing simplifies processor control logic
  - Easier to make reliable, since memory always has copy of data (big idea: Redundancy!)
- Write-Back
  - More complex control logic
  - More variable timing (0,1,2 memory accesses per cache access)
  - Usually reduces write traffic
  - Harder to make reliable, sometimes cache has only copy of data

Administrivia

- Project 3-1 due date Wednesday 10/21.
- Project 3-2 due date now 10/28 (release 10/21)

- Midterm 1:
  - grades posted

Cache (*Performance)* Terms

- Hit rate: fraction of accesses that hit in the cache
- Miss rate: 1 – Hit rate
- Miss penalty: time to replace a block from lower level in memory hierarchy to cache
- Hit time: time to access cache memory (including tag comparison)

- Abbreviation: “$” = cache (A Berkeley innovation!)

Average Memory Access Time (AMAT)

- Average Memory Access Time (AMAT) is the average time to access memory considering both hits and misses in the cache

AMAT = Time for a hit

+ Miss rate × Miss penalty

Clickers/Peer instruction

AMAT = Time for a hit + Miss rate x Miss penalty

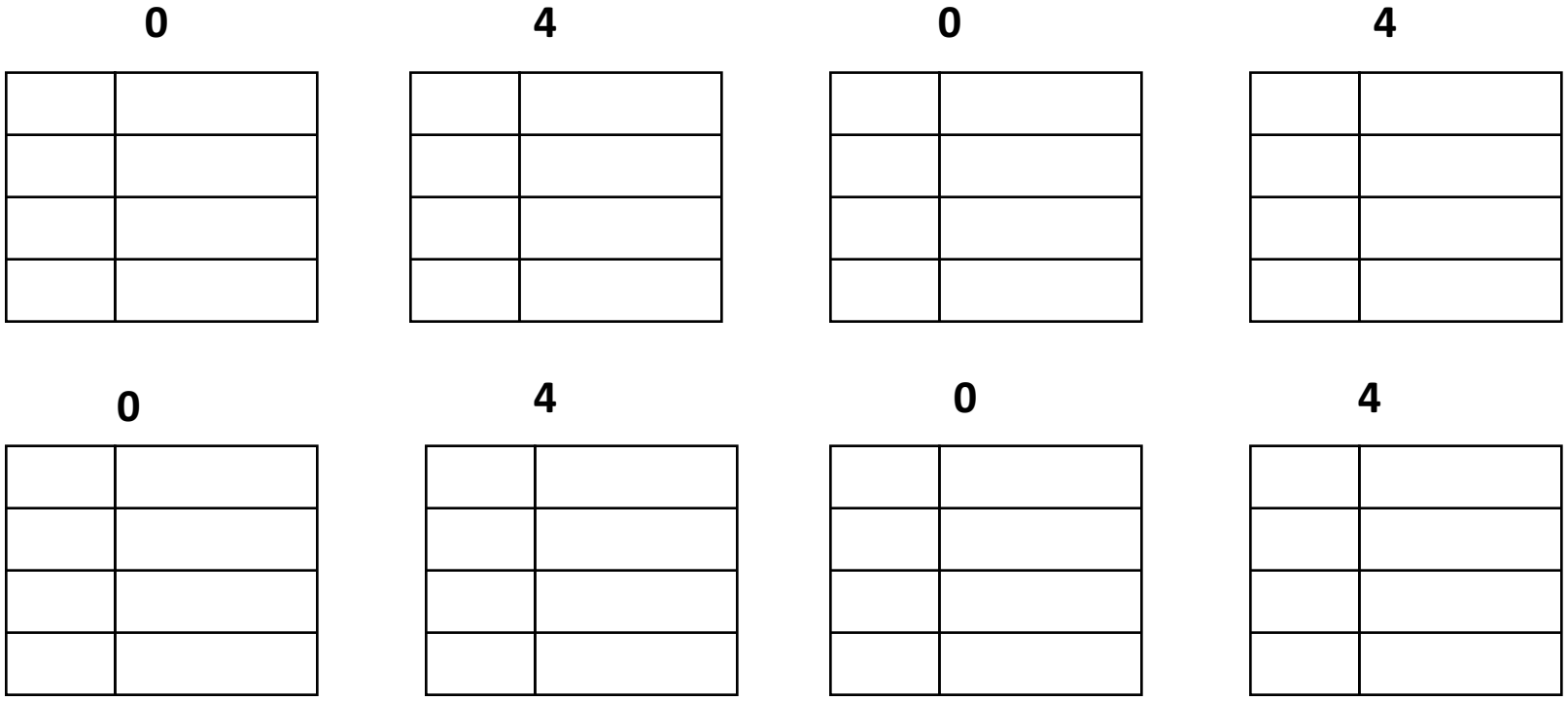
Given a 200 psec clock, a miss penalty of 50 clock cycles, a miss rate of 0.02 misses per instruction and a cache hit time of 1 clock cycle, what is AMAT?

- ☐ A: ≤200 psec
- ☐ B: 400 psec
- ☐ C: 600 psec
- ☐ D: ≥ 800 psec

Example: Direct-Mapped Cache with 4 Single-Word Blocks, Worst-Case Reference String

- Consider the main memory address reference string of word numbers: 0 4 0 4 0 4 0 4

Start with an empty cache - all blocks initially marked as not valid



Example: Direct-Mapped Cache with 4 Single-Word Blocks, Worst-Case Reference String

- Consider the main memory address reference string of word numbers: 0 4 0 4 0 4 0 4

Start with an empty cache - all blocks initially marked as not valid

0 miss

00	Mem(0)

4 miss

~~00~~	Mem(~~0~~)

0 miss

~~01~~	Mem(~~4~~)

4 miss

~~00~~	Mem(~~0~~)

0 miss

~~01~~	Mem(~~4~~)

4 miss

~~00~~	Mem(~~0~~)

0 miss

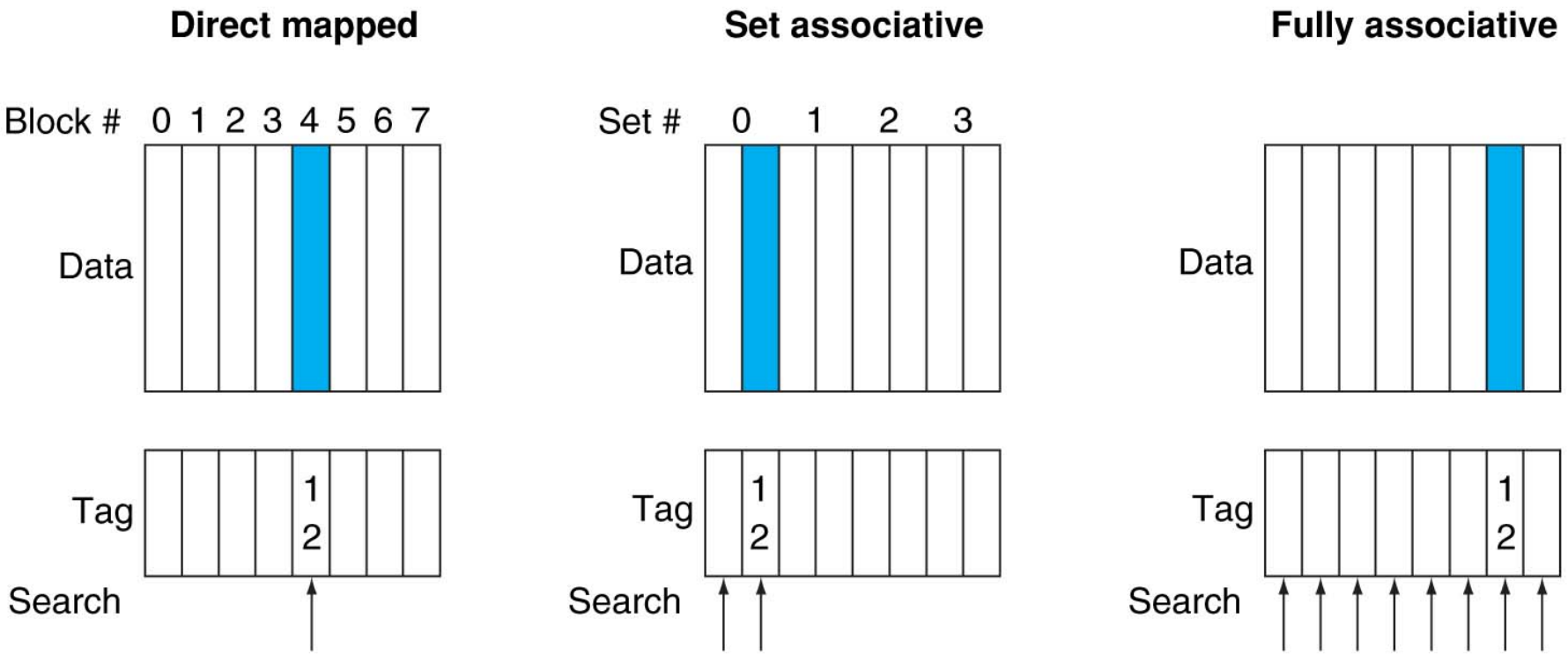
~~01~~	Mem(~~4~~)

4 miss

~~00~~	Mem(~~0~~)

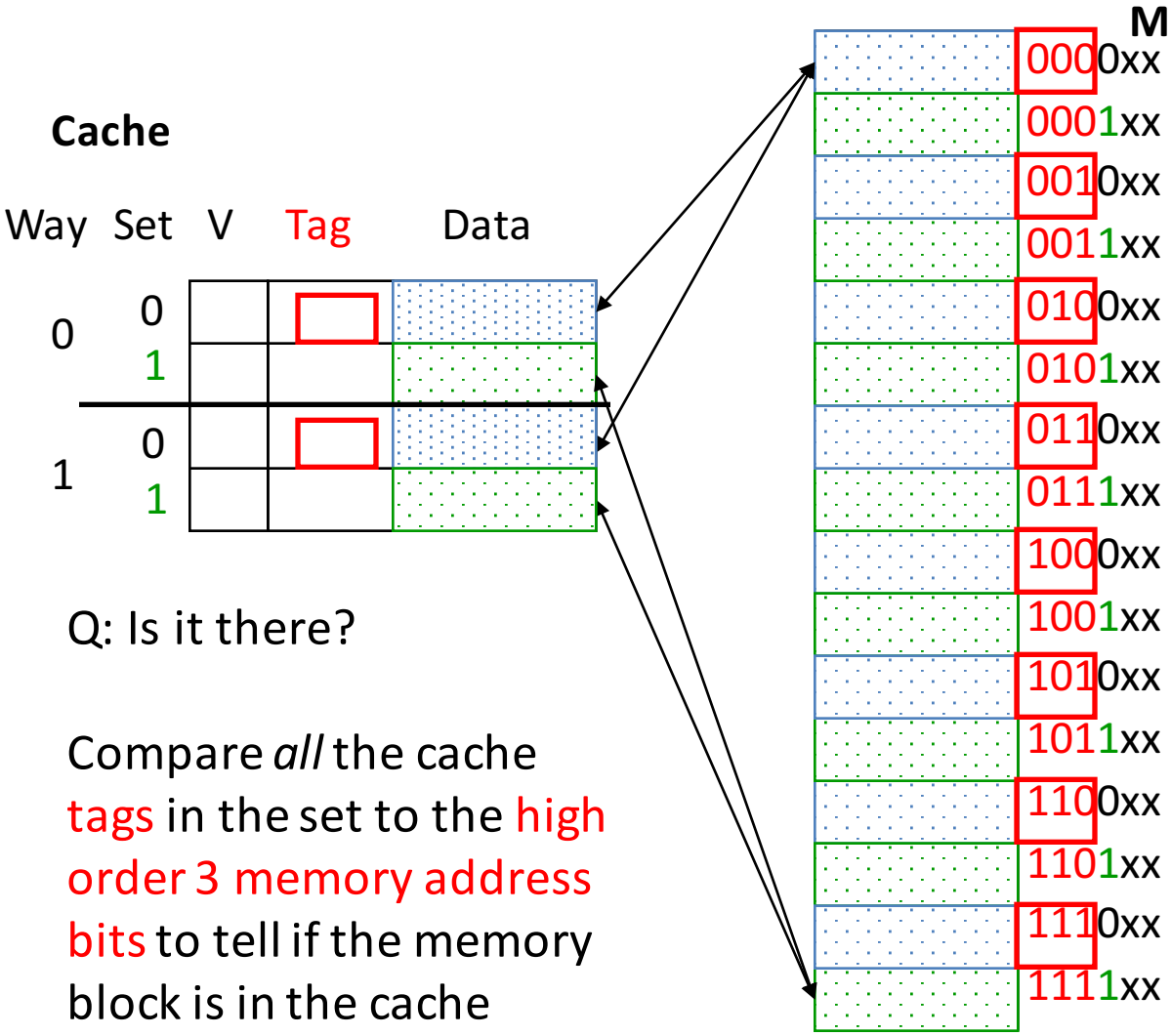
- 8 requests, 8 misses
- Ping-pong effect due to conflict misses - two memory locations that map into the same cache block

Alternative Block Placement Schemes



- DM placement: mem block 12 in 8 block cache: only one cache block where mem block 12 can be found—(12 modulo 8) = 4
- SA placement: four sets x 2-ways (8 cache blocks), memory block 12 in set (12 mod 4) = 0; either element of the set
- FA placement: mem block 12 can appear in any cache blocks

Example: 2-Way Set Associative $
(4 words = 2 sets x 2 ways per set)



One word blocks
Two low order bits define the byte in the word (32b words)

Q: How do we find it?

Use next 1 low order memory address bit to determine which cache set (i.e., modulo the number of sets in the cache)

Q: Is it there?

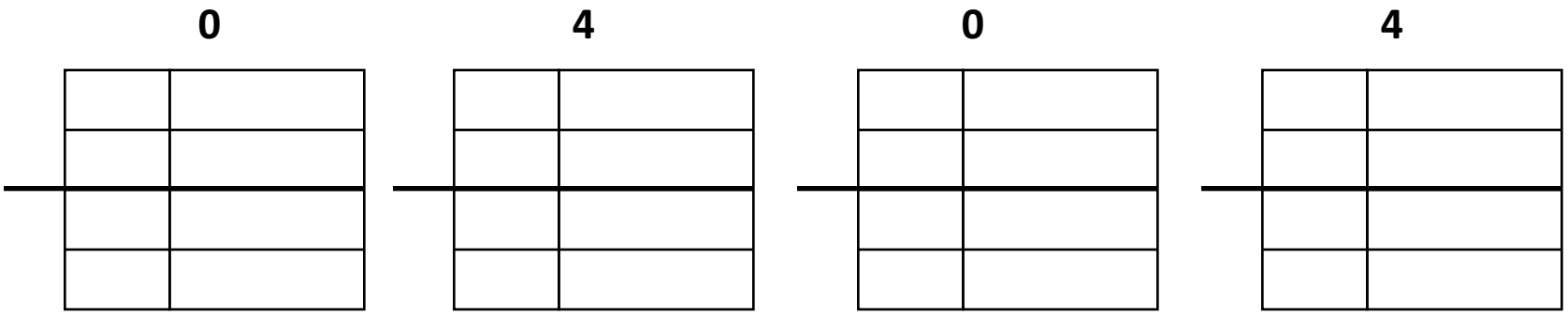
Compare *all* the cache tags in the set to the high order 3 memory address bits to tell if the memory block is in the cache

Example: 4 Word 2-Way SA $ Same Reference String

- Consider the main memory word reference string

Start with an empty cache - all blocks initially marked as not valid

0 4 0 4 0 4 0 4



Example: 4-Word 2-Way SA $ Same Reference String

- Consider the main memory address reference string

Start with an empty cache - all blocks initially marked as not valid

0 4 0 4 0 4 0 4

0 miss

	000	Mem(0)

4 miss

	000	Mem(0)
	010	Mem(4)

0 hit

	000	Mem(0)
	010	Mem(4)

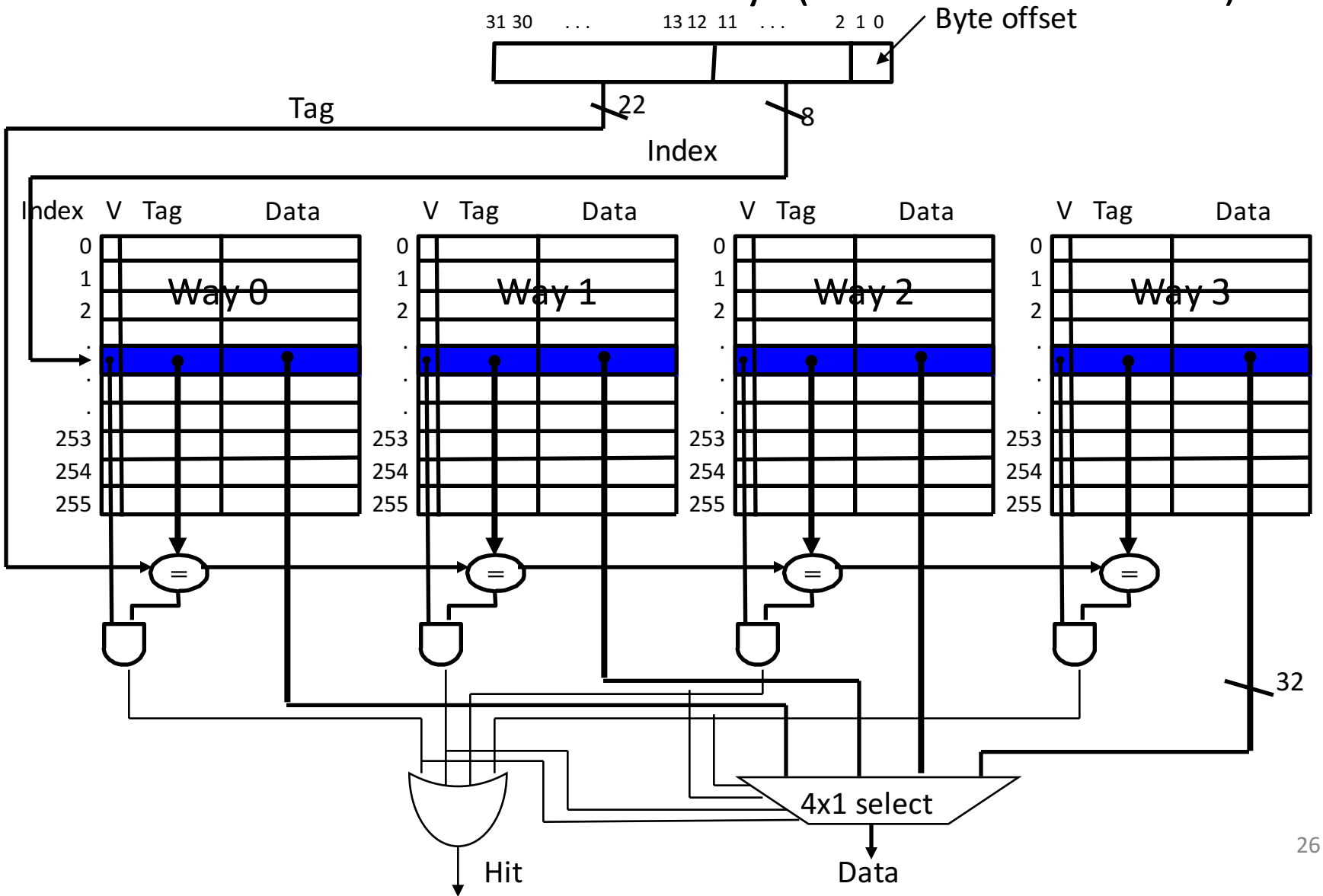
4 hit

	000	Mem(0)
	010	Mem(4)

  - 8 requests, 2 misses
- Solves the ping-pong effect in a direct-mapped cache due to conflict misses since now two memory locations that map into the same cache set can co-exist!

Four-Way Set-Associative Cache

- 2^8 = 256 sets each with four ways (each with one block)



Different Organizations of an Eight-Block Cache

One-way set associative

(direct mapped)

Block	Tag	Data
0		
1		
2		
3		
4		
5		
6		
7		

Two-way set associative

Set	Tag	Data	Tag	Data
0				
1				
2				
3				

Four-way set associative

Set	Tag	Data	Tag	Data	Tag	Data	Tag	Data
0								
1								

Eight-way set associative (fully associative)

Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data

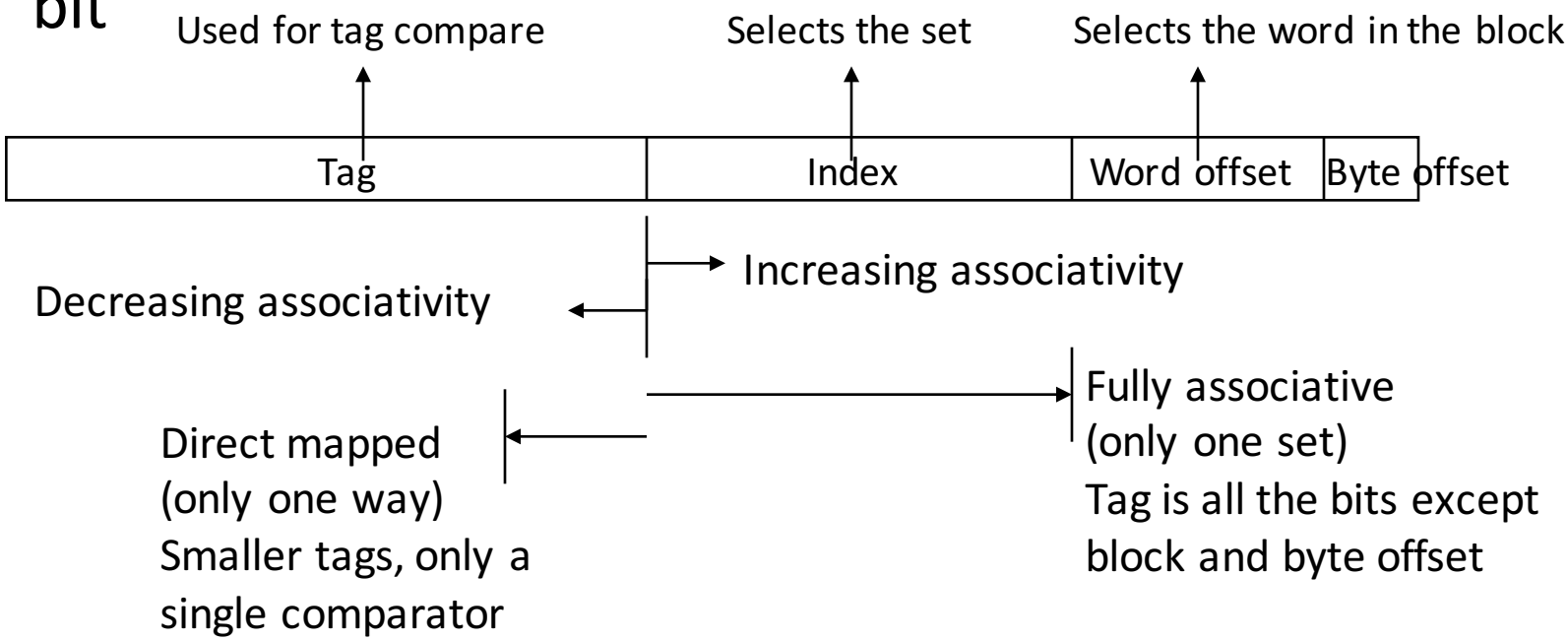
Total size of $ in blocks is equal to *number of sets* × *associativity*. For fixed $ size and fixed block size, increasing associativity decreases number of sets while increasing number of elements per set. With eight blocks, an 8-way set-associative $ is same as a fully associative $.

Range of Set-Associative Caches

- For a fixed-size cache and fixed block size, each increase by a factor of two in associativity doubles the number of blocks per set (i.e., the number or ways) and halves the number of sets – decreases the size of the index by 1 bit and increases the size of the tag by 1 bit

Range of Set-Associative Caches

- For a *fixed-size* cache and fixed block size, each increase by a factor of two in associativity doubles the number of blocks per set (i.e., the number or ways) and halves the number of sets – decreases the size of the index by 1 bit and increases the size of the tag by 1 bit



Total Cache Capacity =

Associativity × # of sets × block_size

Bytes = blocks/set × sets × Bytes/block

$$C = N \times S \times B$$

address_size = tag_size + index_size + offset_size
= tag_size + $\log_2(S)$ + $\log_2(B)$

Clickers/Peer Instruction

- For a cache with constant total capacity, if we increase the number of ways by a factor of 2, which statement is false:
- A: The number of sets could be doubled
- B: The tag width could decrease
- C: The block size could stay the same
- D: The block size could be halved
- E: Tag width must increase

Total Cache Capacity =

Associativity × # of sets × block_size

Bytes = blocks/set × sets × Bytes/block

$C = N \times S \times B$

address_size = tag_size + index_size + offset_size
= tag_size + $\log_2(S)$ + $\log_2(B)$

Clicker Question: C remains constant, S and/or B can change such that
C = 2N * (SB)' => (SB)' = SB/2

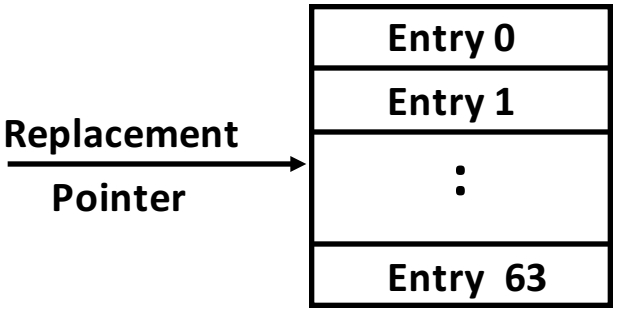
Tag_size = address_size – ($\log_2(S)$ + $\log_2(B)$) = address_size – $\log_2(SB)$
= address_size – $\log_2(SB/2)$
= address_size – ($\log_2(SB)$ – 1)

Costs of Set-Associative Caches

- N-way set-associative cache costs
  - N comparators (delay and area)
  - MUX delay (set selection) before data is available
  - Data available after set selection (and Hit/Miss decision). DM $: block is available before the Hit/Miss decision
    - In Set-Associative, not possible to just assume a hit and continue and recover later if it was a miss
- When miss occurs, which way's block selected for replacement?
  - Least Recently Used (LRU): one that has been unused the longest (principle of temporal locality)
    - Must track when each way's block was used relative to other blocks in the set
    - For 2-way SA $, one bit per set → set to 1 when a block is referenced; reset the other way's bit (i.e., "last used")

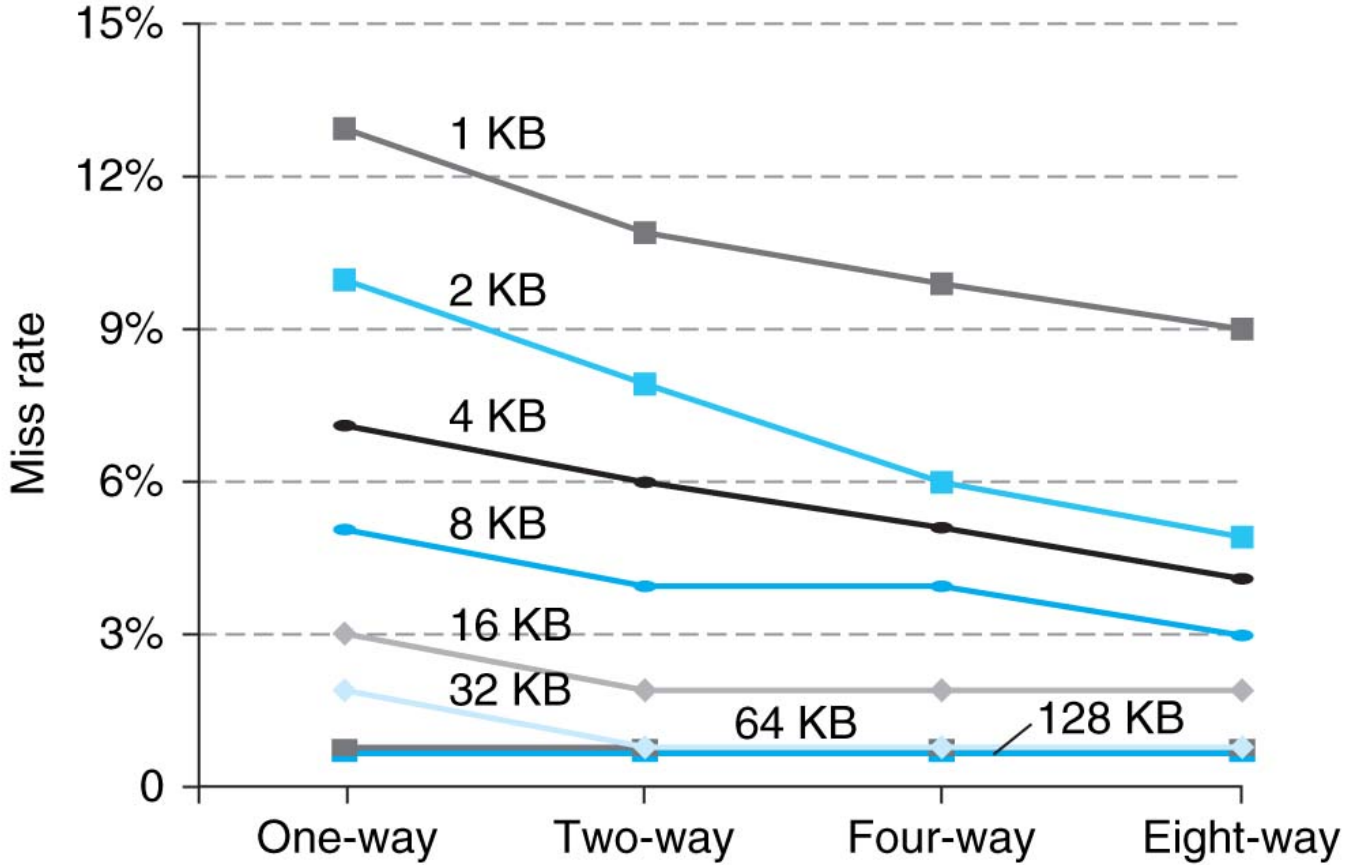
Cache Replacement Policies

- Random Replacement
  - Hardware randomly selects a cache evict
- Least-Recently Used
  - Hardware keeps track of access history
  - Replace the entry that has not been used for the longest time
  - For 2-way set-associative cache, need one bit for LRU replacement
- Example of a Simple "Pseudo" LRU Implementation
  - Assume 64 Fully Associative entries
  - Hardware replacement pointer points to one cache entry
  - Whenever access is made to the entry the pointer points to:
    - Move the pointer to the next entry
  - Otherwise: do not move the pointer
  - (example of "not-most-recently used" replacement policy)



Benefits of Set-Associative Caches

- Choice of DM $ versus SA $ depends on the cost of a miss versus the cost of implementation



- Largest gains are in going from direct mapped to 2-way (20%+ reduction in miss rate)

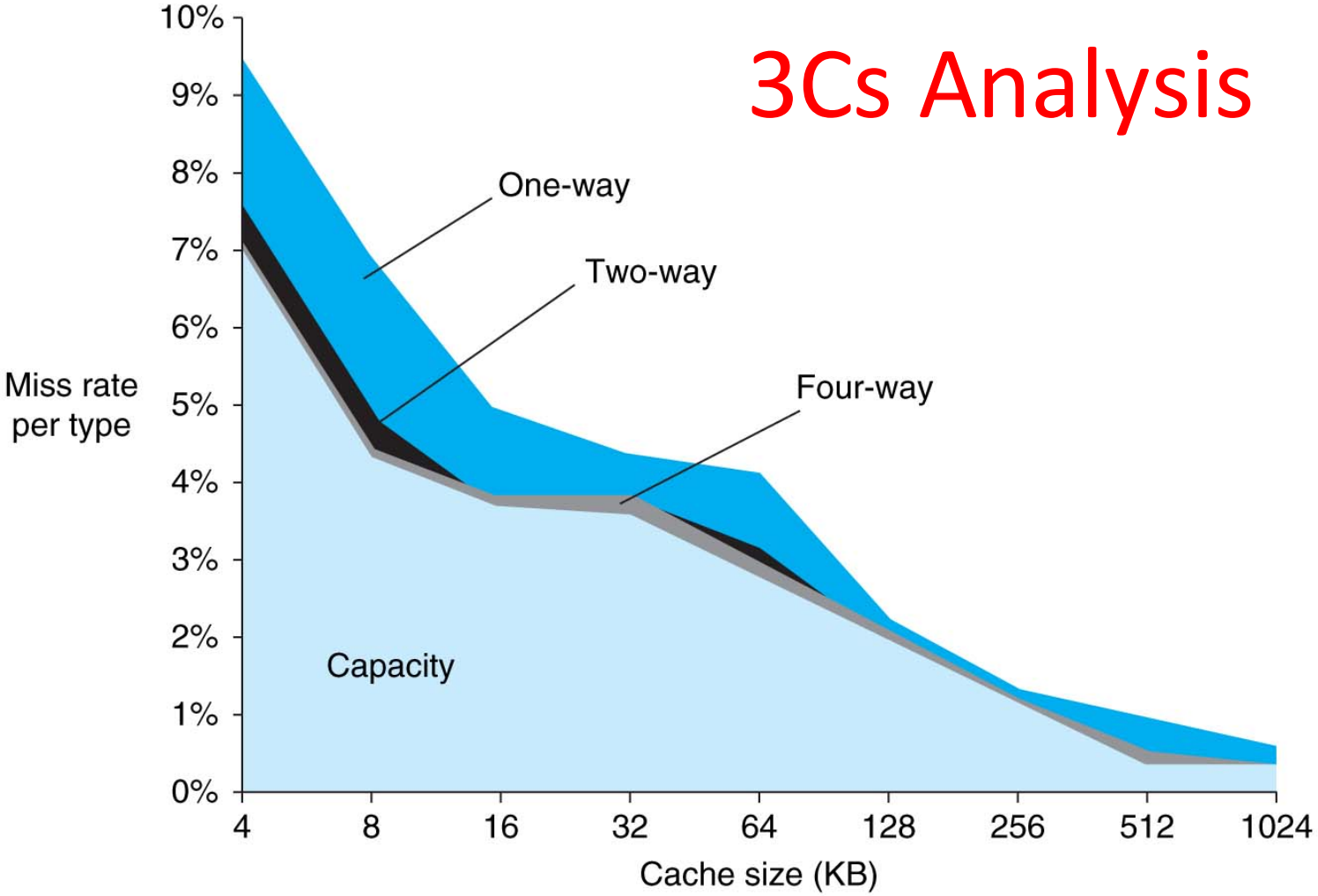
Understanding Cache Misses: The 3Cs

- Compulsory (cold start or process migration, 1st reference):
  - First access to block impossible to avoid; small effect for long running programs
  - Solution: increase block size (increases miss penalty; very large blocks could increase miss rate)
- Capacity:
  - Cache cannot contain all blocks accessed by the program
  - Solution: increase cache size (may increase access time)
- *Conflict (collision):*
  - *Multiple memory locations mapped to the same cache location*
  - *Solution 1: increase cache size*
  - *Solution 2: increase associativity (may increase access time)*

How to Calculate 3C's using Cache Simulator

1. *Compulsory*: set cache size to infinity and fully associative, and count number of misses
2. *Capacity*: Change cache size from infinity, usually in powers of 2, and count misses for each reduction in size
   - 16 MB, 8 MB, 4 MB, … 128 KB, 64 KB, 16 KB
3. *Conflict*: Change from fully associative to n-way set associative while counting misses
   - Fully associative, 16-way, 8-way, 4-way, 2-way, 1-way

3Cs Analysis



- Three sources of misses (SPEC2000 integer and floating-point benchmarks)
  - Compulsory misses 0.006%; not visible
  - Capacity misses, function of cache size
  - Conflict portion depends on associativity and cache size

Improving Cache Performance

AMAT = Time for a hit + Miss rate x Miss penalty

- Reduce the time to hit in the cache
  - E.g., Smaller cache
- Reduce the miss rate
  - E.g., Bigger cache
- Reduce the miss penalty
  - E.g., Use multiple cache levels

Impact of Larger Cache on AMAT?

- 1) Reduces misses (what kind(s)?)
- 2) Longer Access time (Hit time): smaller is faster
  - Increase in hit time will likely add another stage to the pipeline
- At some point, increase in hit time for a larger cache may overcome the improvement in hit rate, yielding a decrease in performance
- Computer architects expend considerable effort optimizing organization of cache hierarchy – big impact on performance and power!

Clickers: Impact of longer cache blocks on misses?

- For fixed total cache capacity and associativity, what is effect of longer blocks on each type of miss:
  - A: Decrease, B: Unchanged, C: Increase
- Compulsory?
- Capacity?
- Conflict?

Clickers: Impact of longer blocks on AMAT

- For fixed total cache capacity and associativity, what is effect of longer blocks on each component of AMAT:
  - A: Decrease, B: Unchanged, C: Increase
- Hit Time?
- Miss Rate?
- Miss Penalty?

Clickers/Peer Instruction:

For fixed capacity and fixed block size, how does increasing associativity effect AMAT?

A: Increases hit time, decreases miss rate

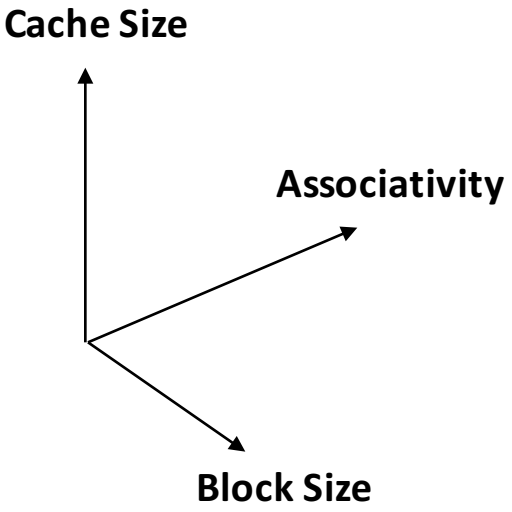
B: Decreases hit time, decreases miss rate

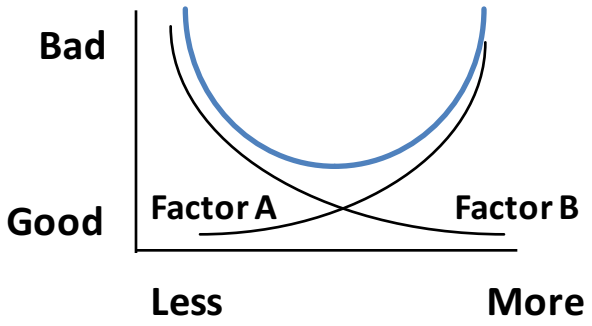
C: Increases hit time, increases miss rate

D: Decreases hit time, increases miss rate

Cache Design Space

- Several interacting dimensions
  - Cache size
  - Block size
  - Associativity
  - Replacement policy
  - Write-through vs. write-back
  - Write allocation
- Optimal choice is a compromise
  - Depends on access characteristics
    - Workload
    - Use (I-cache, D-cache)
  - Depends on technology / cost
- Simplicity often wins





And, In Conclusion …

- Name of the Game: Reduce AMAT
  - Reduce Hit Time
  - Reduce Miss Rate
  - Reduce Miss Penalty
- Balance cache parameters (Capacity, associativity, block size)