CS 61C: Great Ideas in Computer Architecture

Lecture 15: *Caches, Part 2*

Instructor: Sagar Karandikar
sagark@eecs.berkeley.edu

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

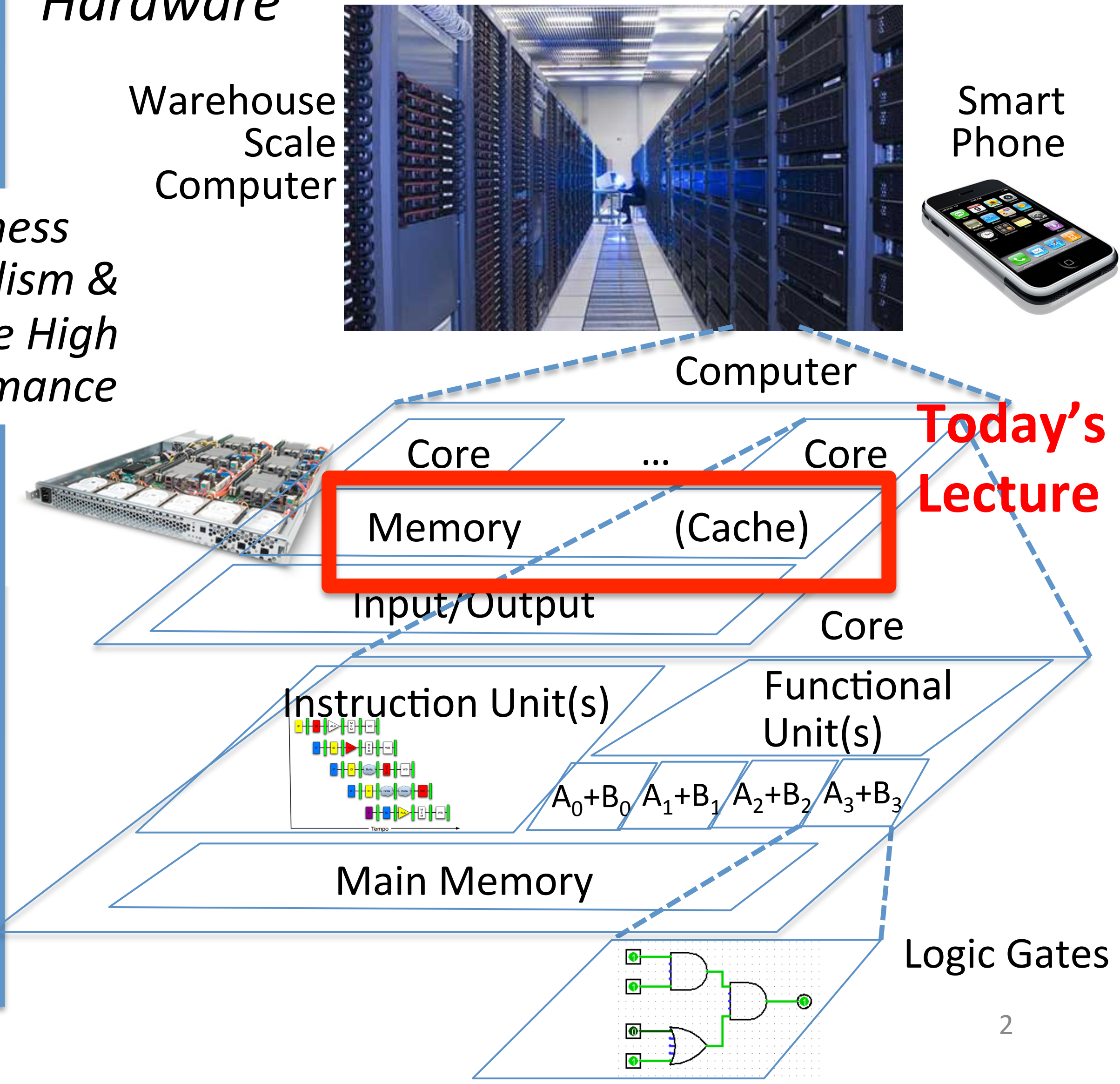
Berkeley EECS
ELECTRICAL ENGINEERING & COMPUTER SCIENCES

You Are Here!

Software | *Hardware*

- Parallel Requests
  Assigned to computer
  e.g., Search "Katz"
- Parallel Threads
  Assigned to core
  e.g., Lookup, Ads
- Parallel Instructions
  >1 instruction @ one time
  e.g., 5 pipelined instructions
- Parallel Data
  >1 data item @ one time
  e.g., Add of 4 pairs of words
- Hardware descriptions
  All gates @ one time
- Programming Languages

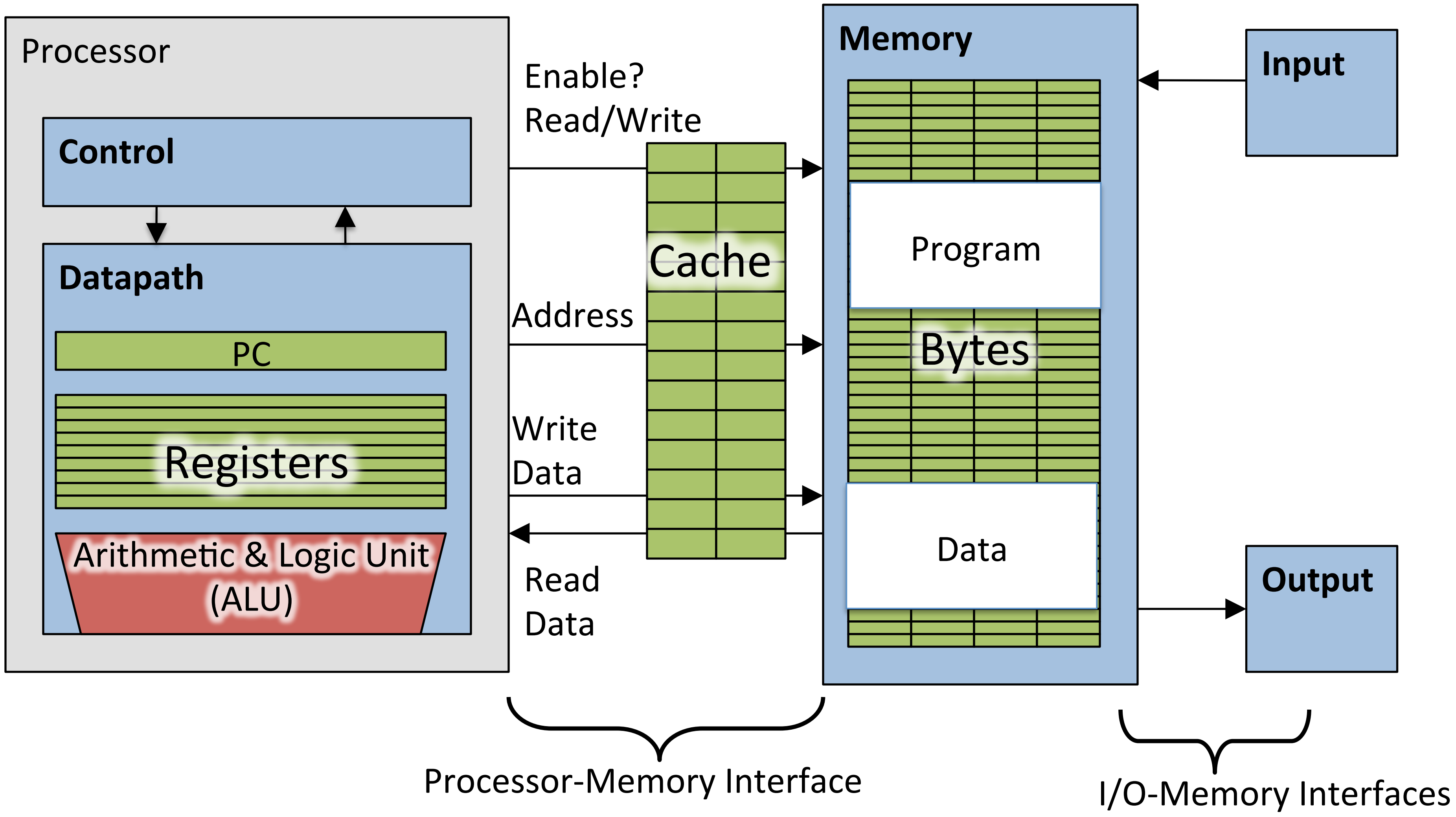
Harness Parallelism & Achieve High Performance



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 Caches
- Cache design choice:
    - Write-Through vs. Write-Back

Review: Adding Cache to Computer

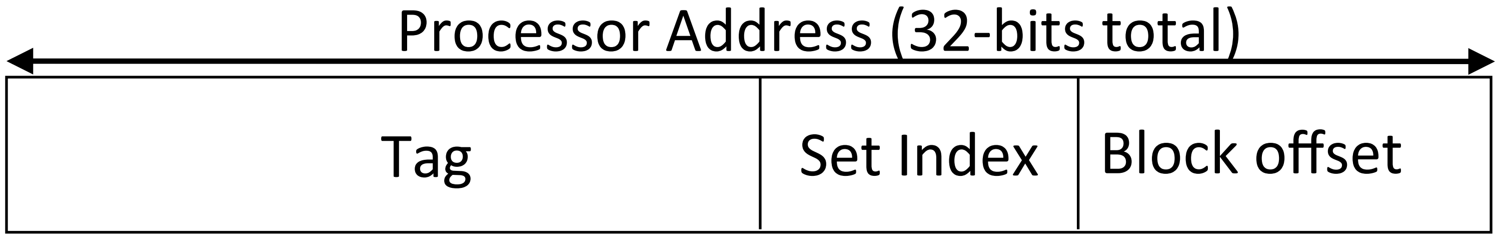


How do we use this thing?

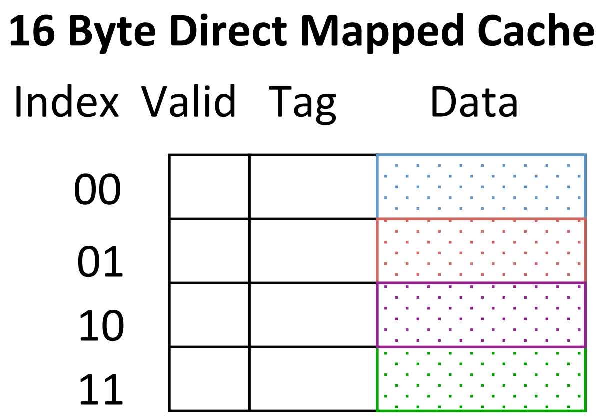
- Nothing changes from the programmer's perspective:
  - Still issuing lw and sw instructions
- Everything we'll talk about is in the hardware:
  - breaking down the address
  - indexing into the cache
  - choosing a block by checking tag
  - extracting bytes using the offset
- Why should a programmer care?
  - Understanding cache parameters = fast programs

Review: Processor Address Fields used by Cache Controller

- Block Offset: Byte address within block
  – #bits = log_2(block size in bytes)
- Index: Selects which index (set) we want
  - # bits = log_2(number of sets)
- Tag: Remaining portion of processor address
  - processor address size - offset bits - index bits



Working with a Direct Mapped Cache

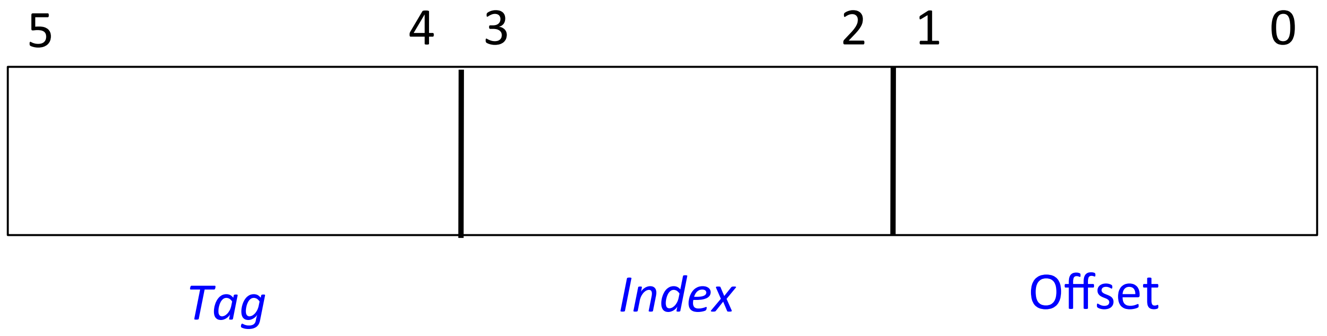


Let's say our cache is what we see on the left.

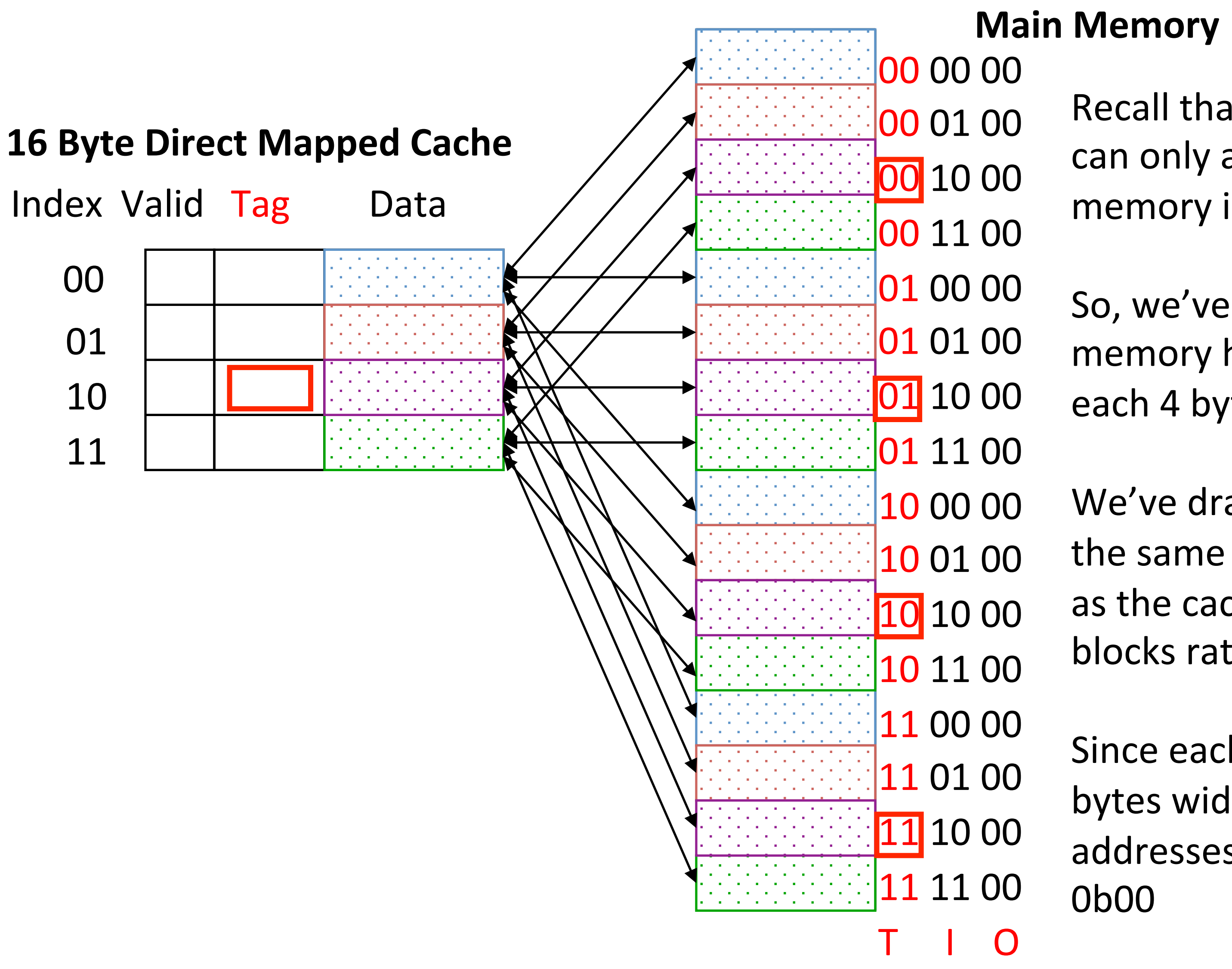
- From the picture, we see that there are 4 indices (sets)
- Given that it is a 16 Byte cache and that there are 4 sets (the same as blocks in a DM cache), each block is 4 bytes
- Our processor can address 2^6 bytes of memory, so it has 6 bit addresses

Mapping an address to our cache

- Using the info from the previous slide…
  - 16 Byte Direct Mapped Cache
  - 4 sets (sets, blocks, indices are all interchangeable in a direct mapped cache)
  - 4 byte blocks
  - Processor issues 6 bit addresses (i.e. has a 64 Byte address space)
- … We can figure out how to break down the address
  - Offset bits = log_2(block size) = log_2(4 byte blocks) = 2
  - Index bits = log_2(number of sets) = log_2(4 sets) = 2
  - Tag = 6 - 2 - 2 = 2



Geometry of our D.M. Cache



Recall that our processor can only address 64 B of memory in this example

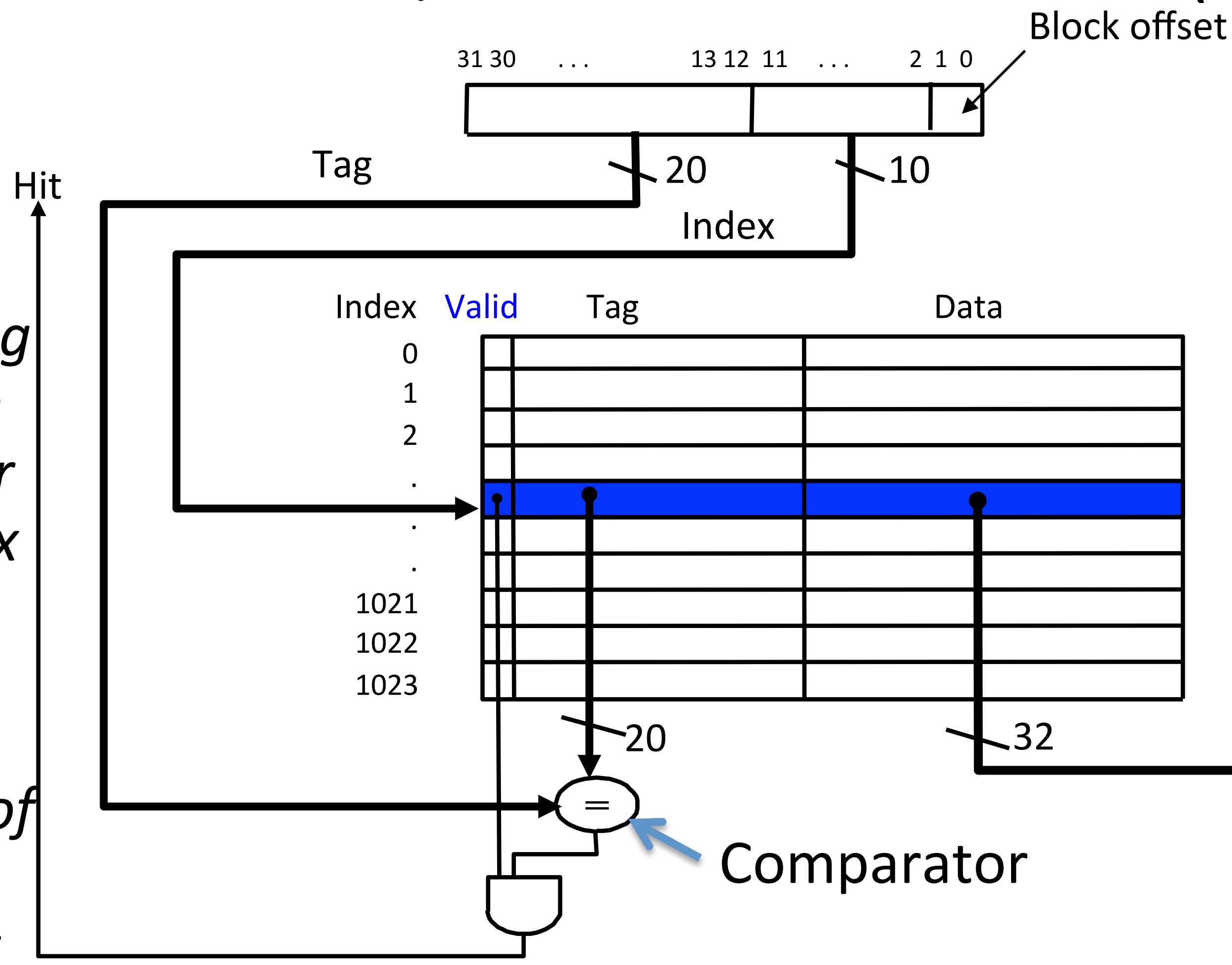
So, we've drawn all of memory here: 16 rows, each 4 bytes wide

We've drawn memory in the same "aspect ratio" as the cache - in units of blocks rather than bytes

Since each block is 4 bytes wide, the addresses you see end in 0b00

Direct Mapped Cache Hardware

- One word blocks, cache size = 1Ki words (or 4KiB)



Valid bit ensures something useful in cache for this index

Compare Tag with upper part of Address to see if a Hit

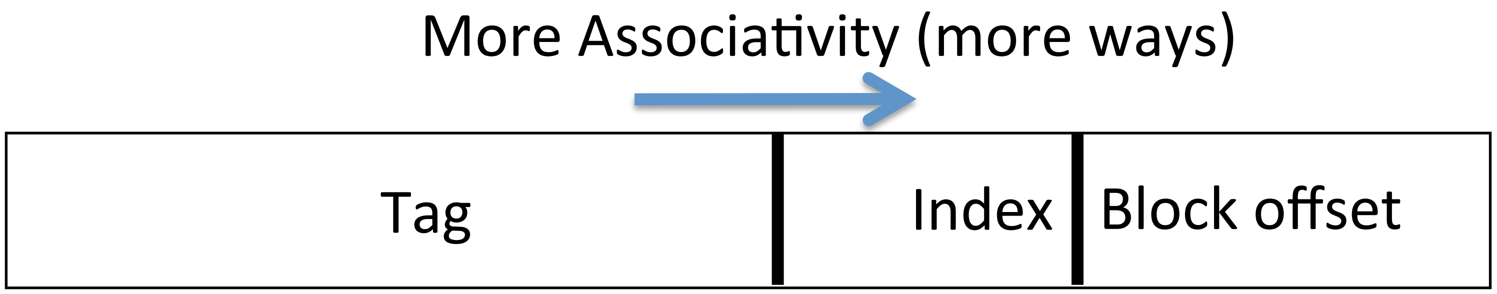
Read data from cache instead of memory if a Hit

Cache Names for Each Organization

- "Fully Associative": Block can go anywhere
  - First design in last 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
  - Fully Associative: N = number of blocks in cache
  - Direct Mapped: N = 1

Range of Set-Associative Caches

- For a fixed-size cache, each increase by a factor of 2 in associativity doubles the number of blocks per set (i.e., the number of "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



Note: IBM persists in calling sets "ways" and ways "sets". They're wrong.

Review: Write-Through vs. Write-Back

So far we handled reads. What about writes?

- Write-Through:
  - Simpler control logic
  - More predictable timing simplifies processor control logic
  - Easier to make reliable, since memory always has copy of data

- 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

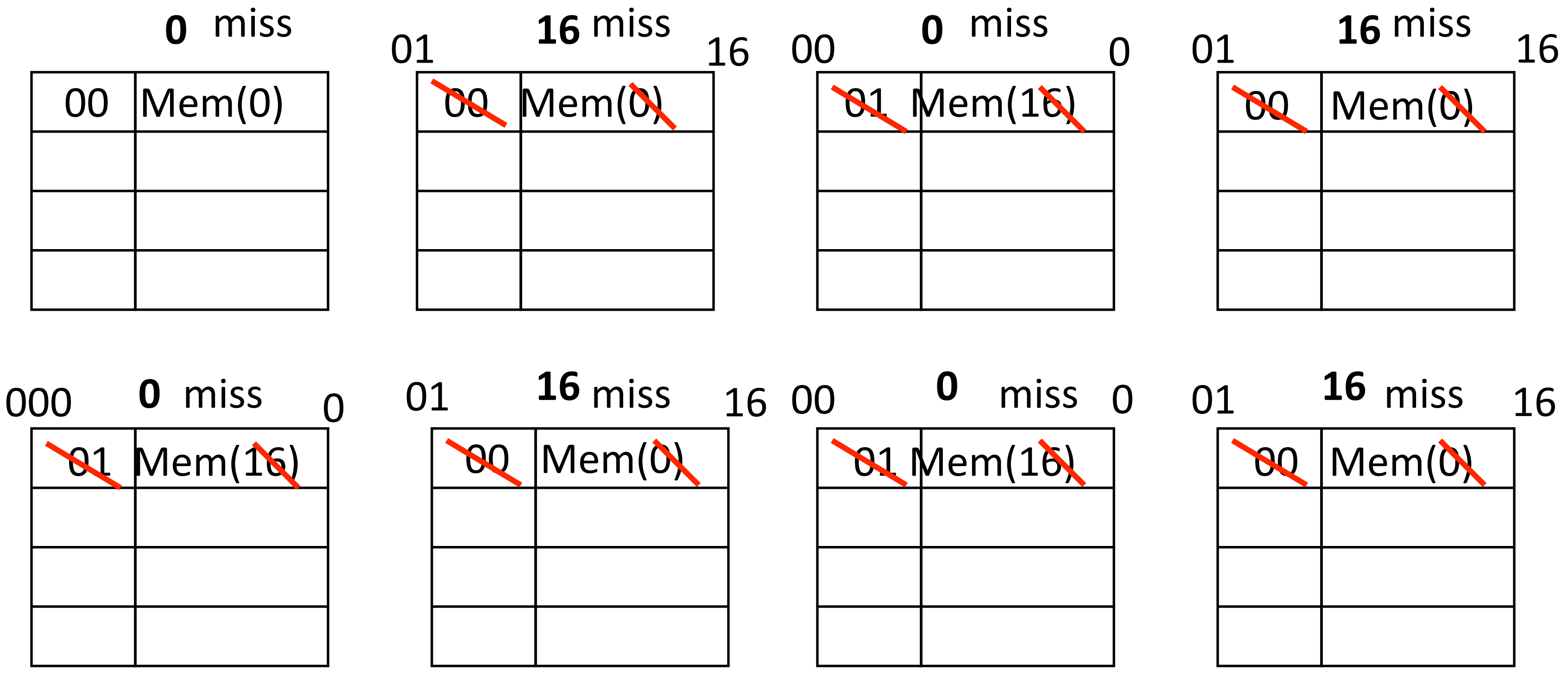
- HW3 Out
- Proj2-2 Out
- Register your Project 3 Teams
- Yes, caches are hard, but don't worry:
  - 1.5 discussions on caches
  - 1 lab on caches (where you use a simulator)
  - 1 more lecture on caches (after today)
  - Homework on caches
  - Guerrilla section on Caches next week
- Guerrilla section today, 6-8pm in Woz, on MIPS CPU
- The good news: You're halfway done!

Example: Direct-Mapped $ with 4 Single-Word Lines, Worst-Case Reference String

- Consider the main memory address reference string

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

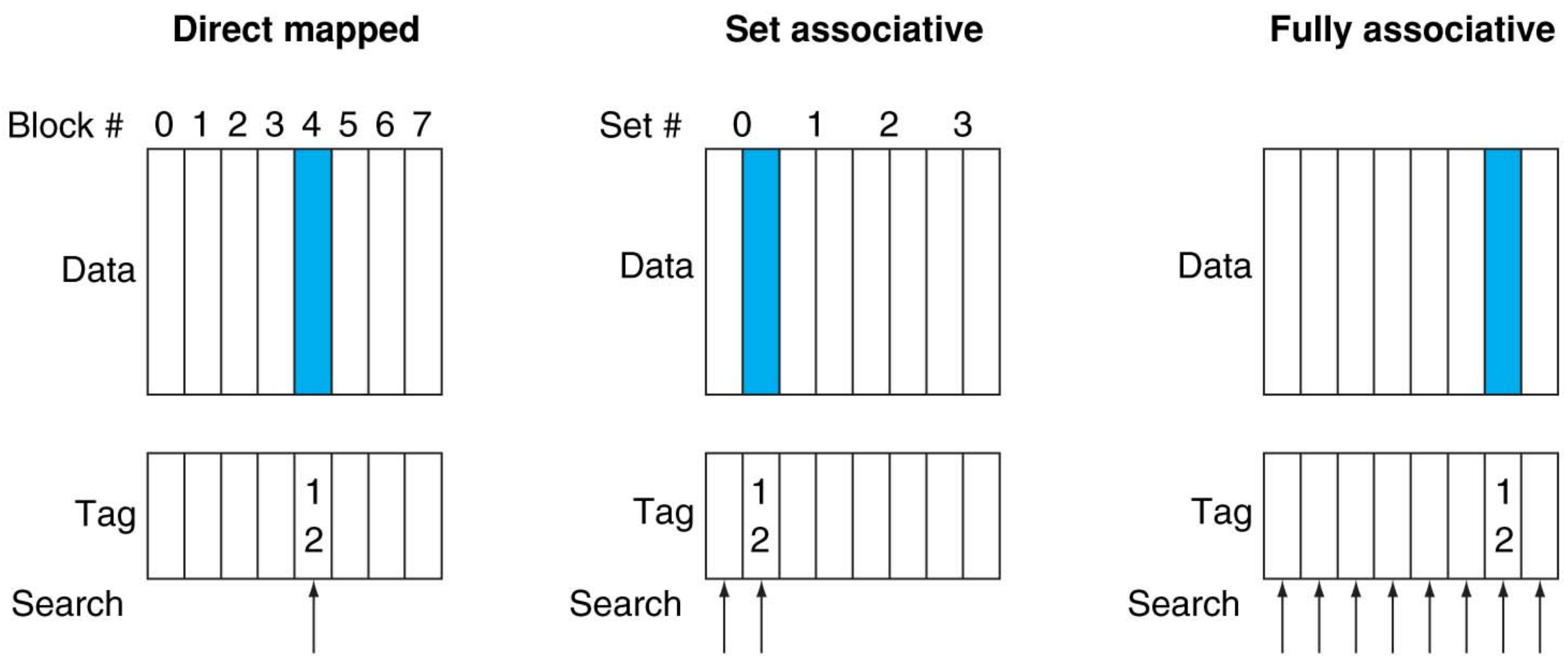
0 16 0 16 0 16 0 16



  - 8 requests, 8 misses

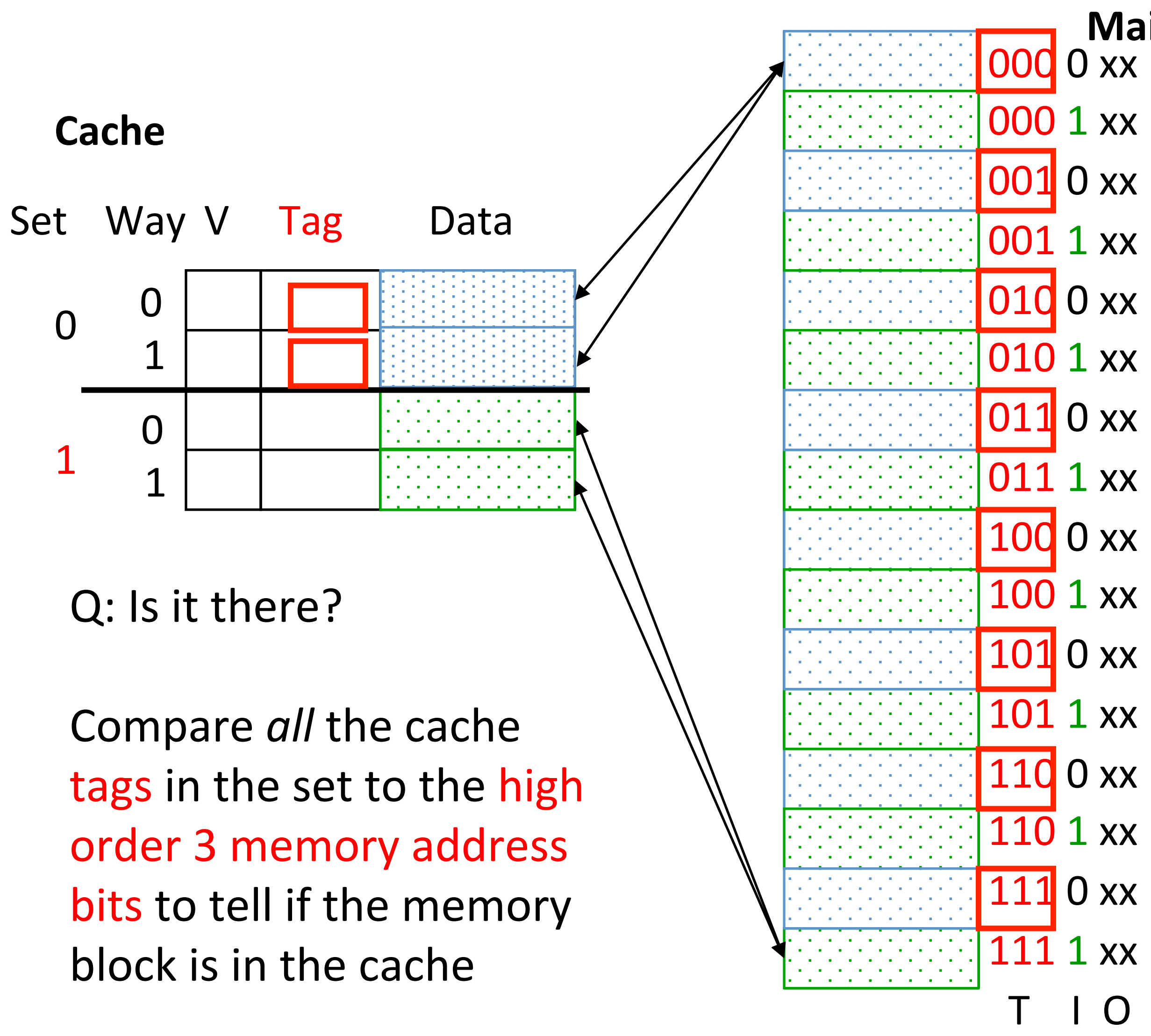
- Effect due to "conflict" misses - two memory locations that map into the same cache block

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: 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

Q: How do we find it?

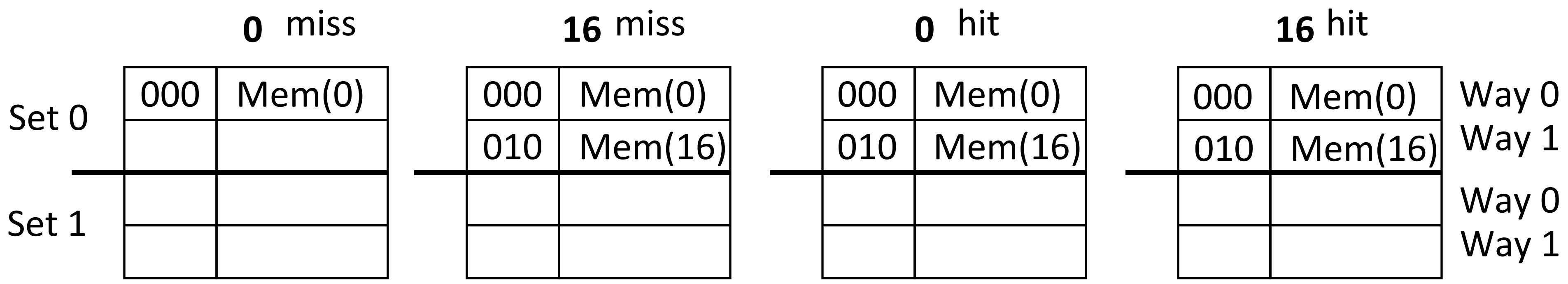
Use next 1 low order memory address bit to determine which cache set (i.e., modulo the number of sets in the cache)

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

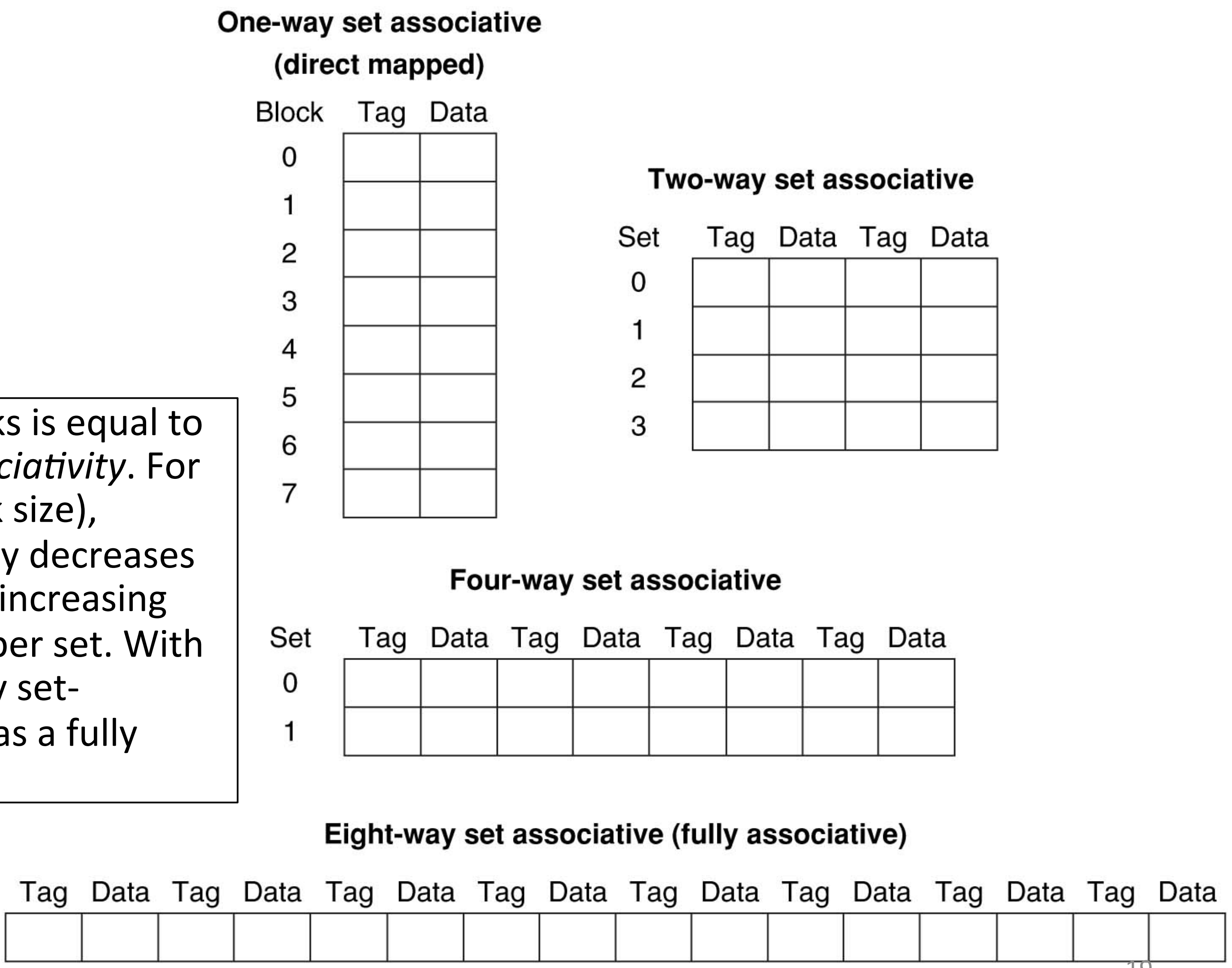
0 16 0 16 0 16 0 16



  - 8 requests, 2 misses

- Solves the "conflict" effect that occurred in a direct-mapped cache (to a degree)
- Because now two memory locations that map into the same cache set can co-exist!
- But what if we did 0, 16, 32, 48, 0, 16, 32, 48, …?

Different Organizations of an Eight-Block Cache

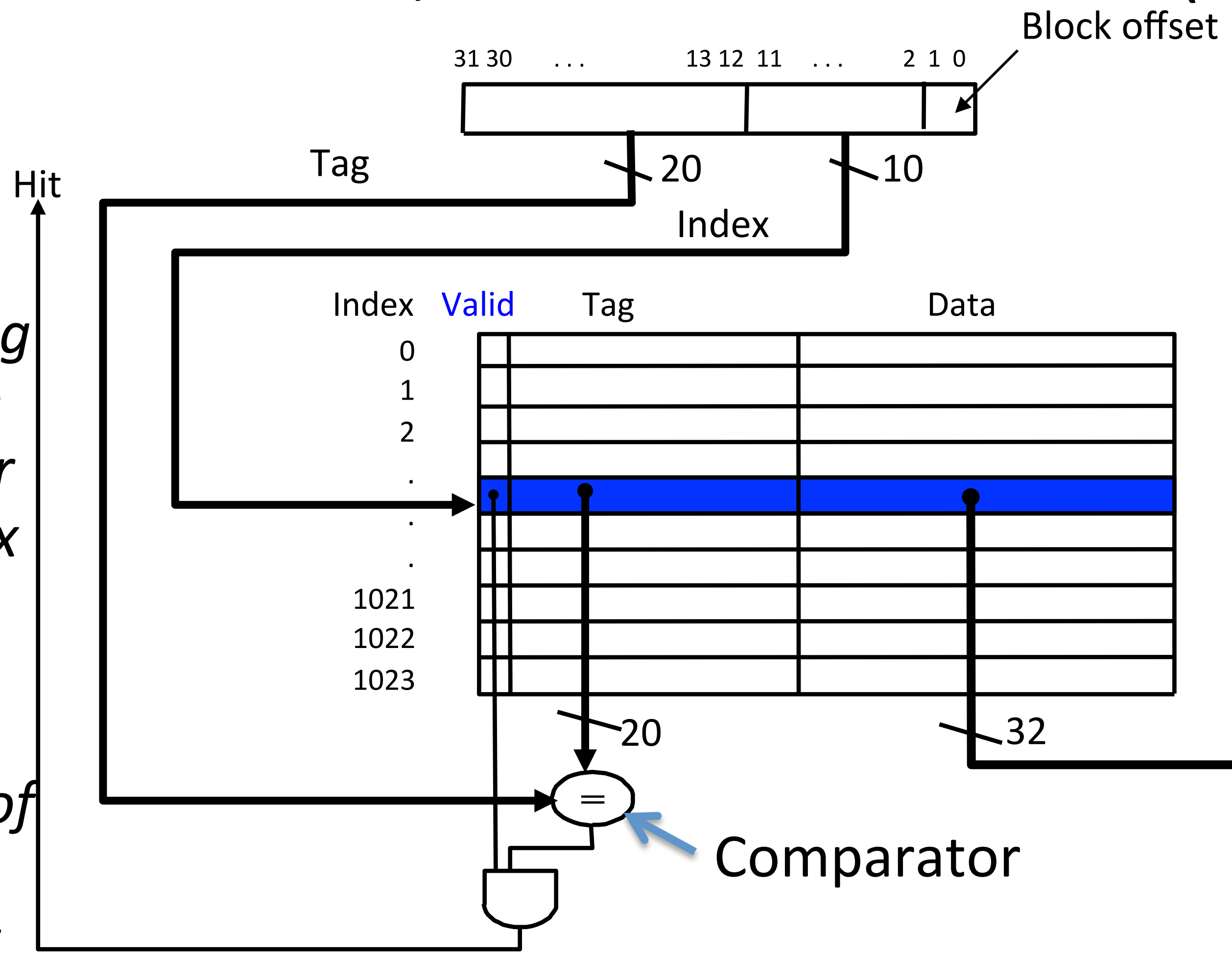


Total size of $ in blocks is equal to *number of sets* x *associativity*. For fixed $ size (and block size), increasing associativity decreases number of sets while increasing number of elements per set. With eight blocks, an 8-way set-associative $ is same as a fully associative $.

Recall: Direct Mapped Cache Hardware

- One word blocks, cache size = 1Ki words (or 4KiB)

Valid bit ensures something useful in cache for this index

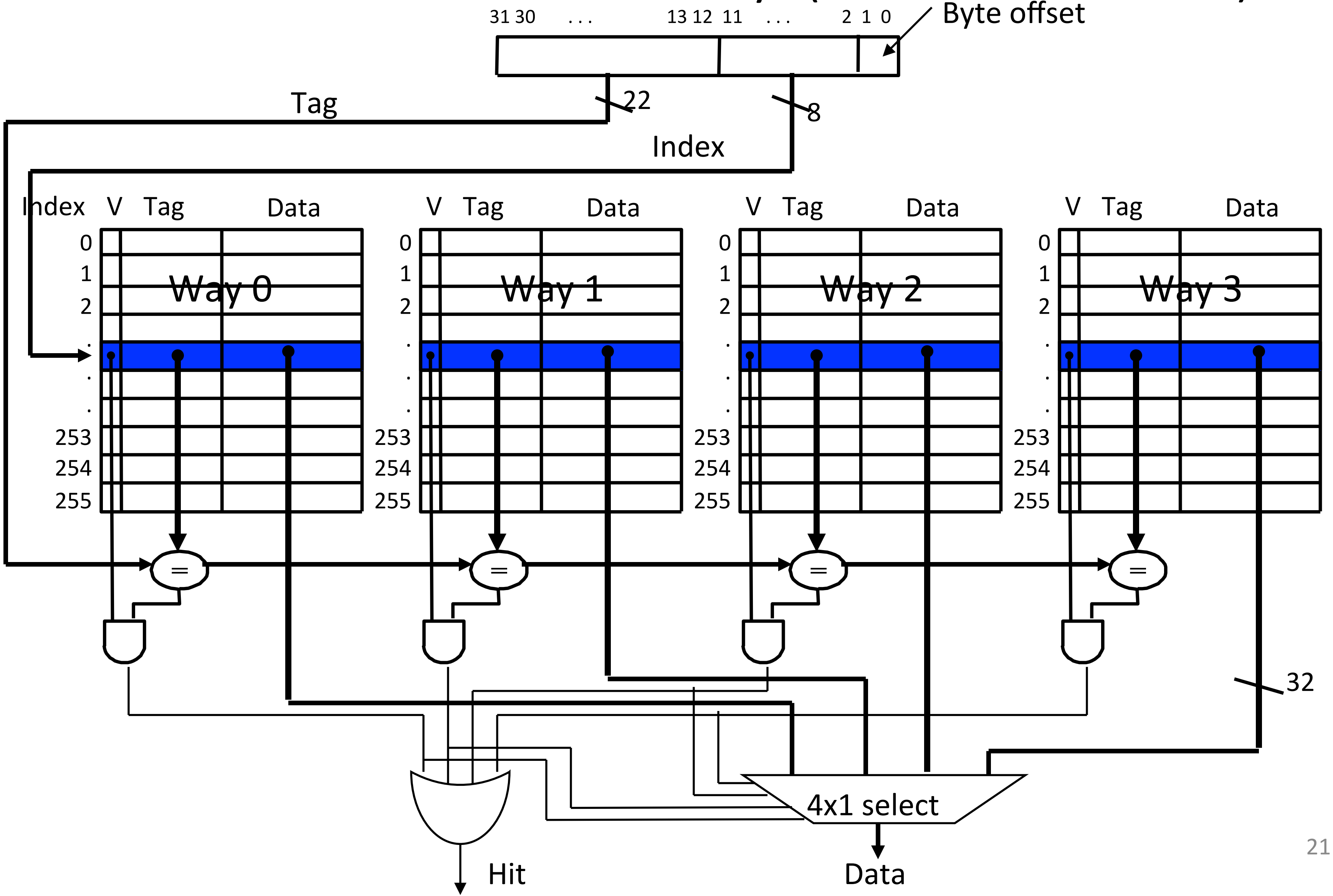


Compare Tag with upper part of Address to see if a Hit

Read data from cache instead of memory if a Hit

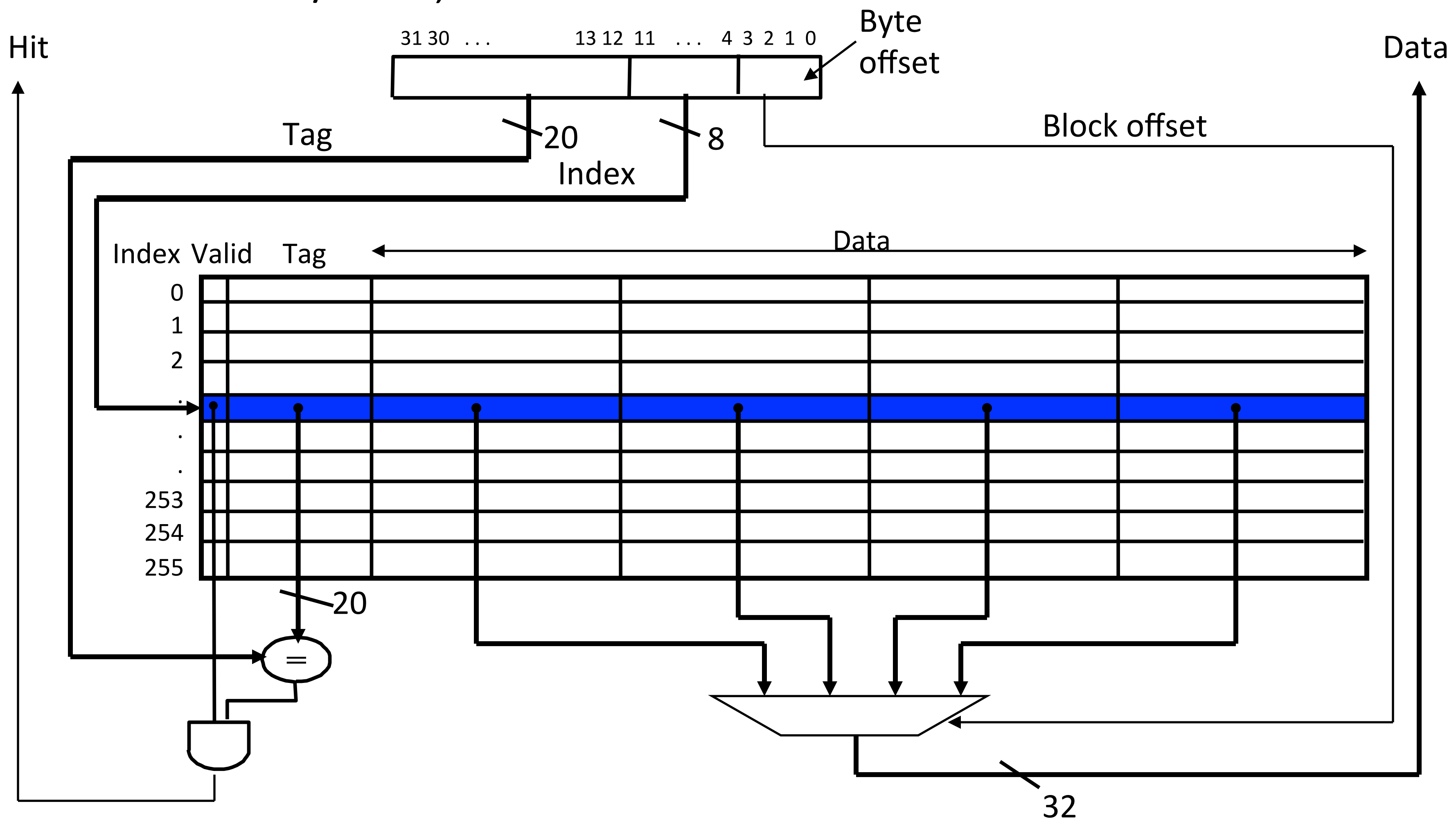
Four-Way Set-Associative Cache

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



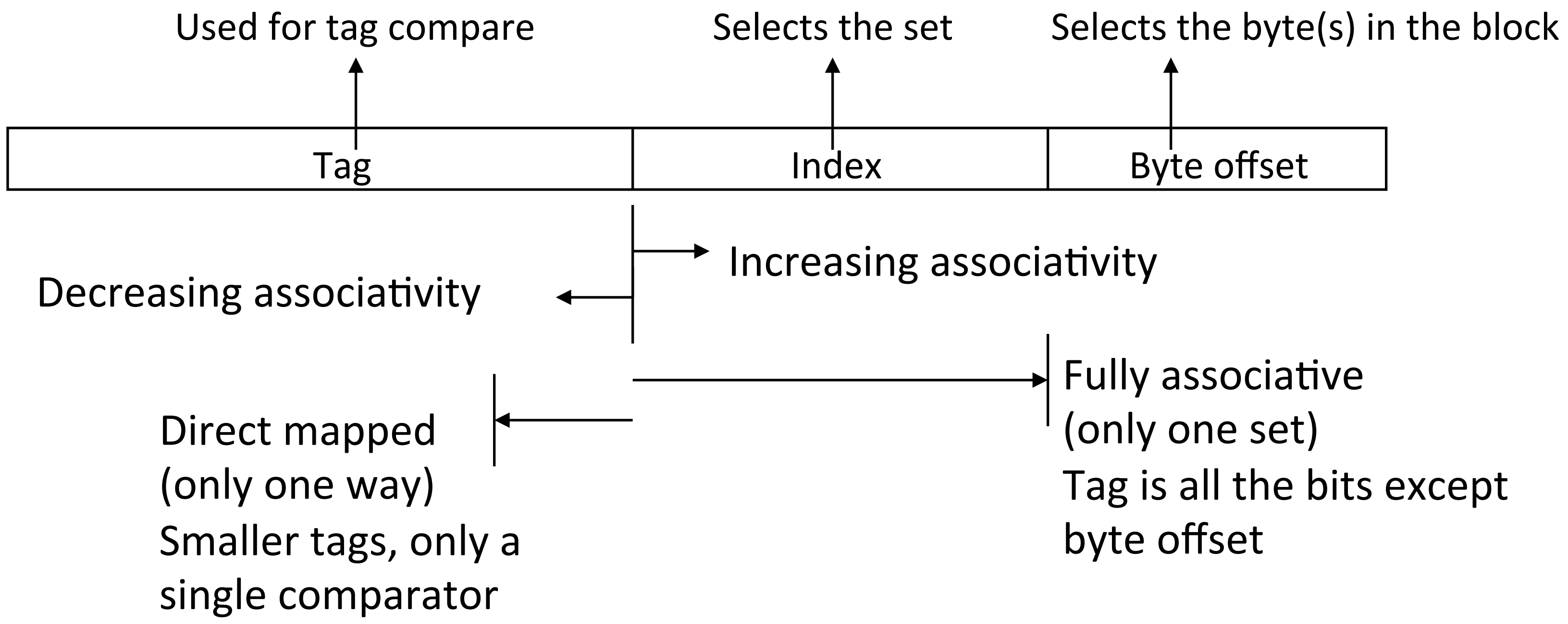
Not the same!: Multiword-Block Direct-Mapped Cache

- Four words/block, cache size = 1Ki words



Range of Set-Associative Caches

- For a *fixed-size* cache, 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

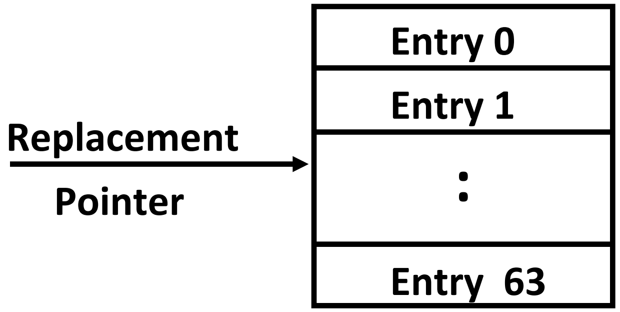


Costs of Set-Associative Caches

- 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")

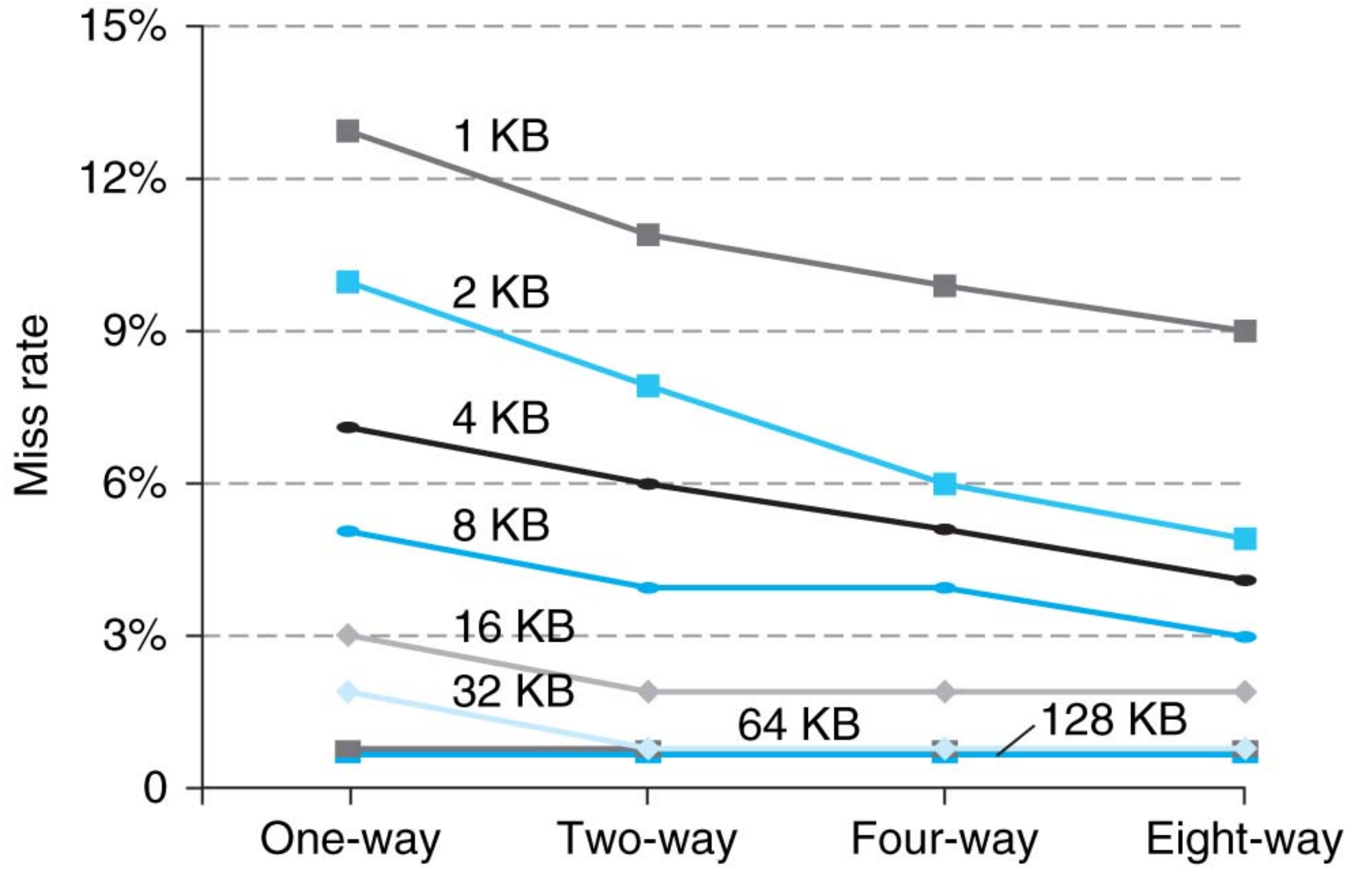
Cache Replacement Policies

- 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 $ or 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)

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 number of tags could stay the same
- D: The block size could be halved
- E: Tag width must increase

Break

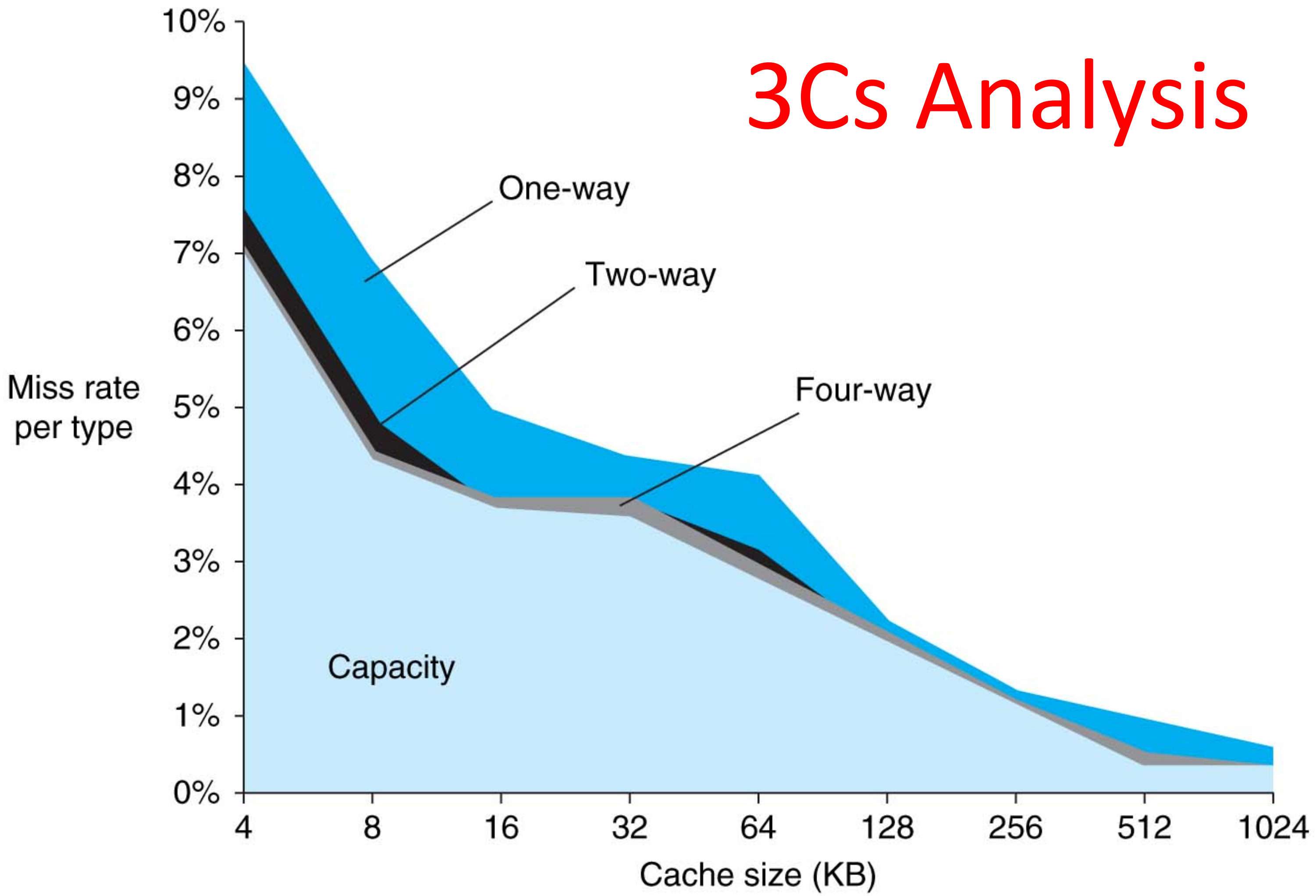
Understanding Cache Misses: The 3Cs

- Compulsory (cold start or process migration, 1st reference):
  - First access to a block in memory 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 a 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

Rules for Determining Miss Type for a Given Access Pattern in 61C

1) Compulsory: A miss is compulsory if and only if it results from accessing the data for the first time. If you have ever brought the data you are accessing into the cache before, it's not a compulsory miss, otherwise it is.

2) Conflict: A conflict miss is a miss that's not compulsory and that would have been avoided if the cache was fully associative. Imagine you had a cache with the same parameters but fully-associative (with LRU). If that would've avoided the miss, it's a conflict miss.

3) Capacity: This is a miss that would not have happened with an infinitely large cache. If your miss is not a compulsory or conflict miss, it's a capacity miss.

Cache Measurement 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

Average Memory Access Time (AMAT)

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

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

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 (next time)

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?

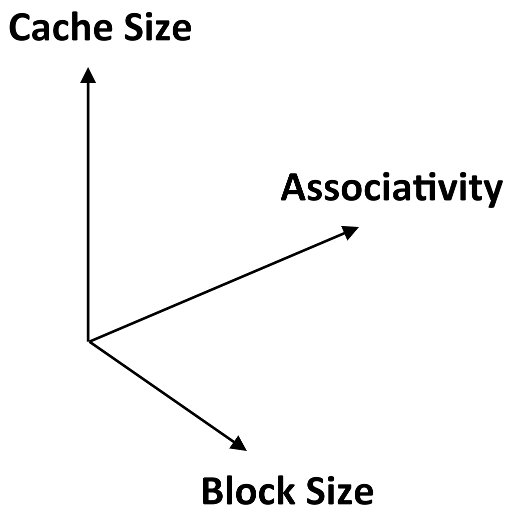
- For fixed total cache capacity and associativity, what is effect of longer blocks on each type of miss:
  - A: Decrease, B: Unchanged, C: Increase
- Compulsory?
- Capacity?
- Conflict?

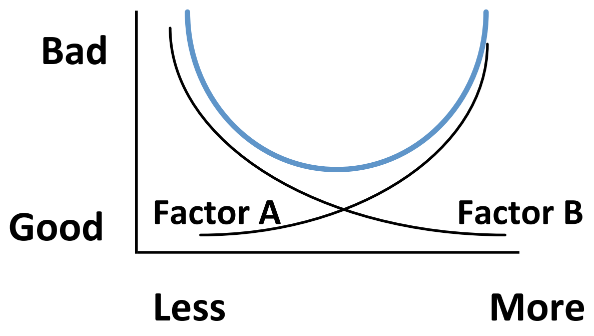
Clickers:
For fixed capacity and fixed block size, how does increasing associativity affect AMAT?

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)