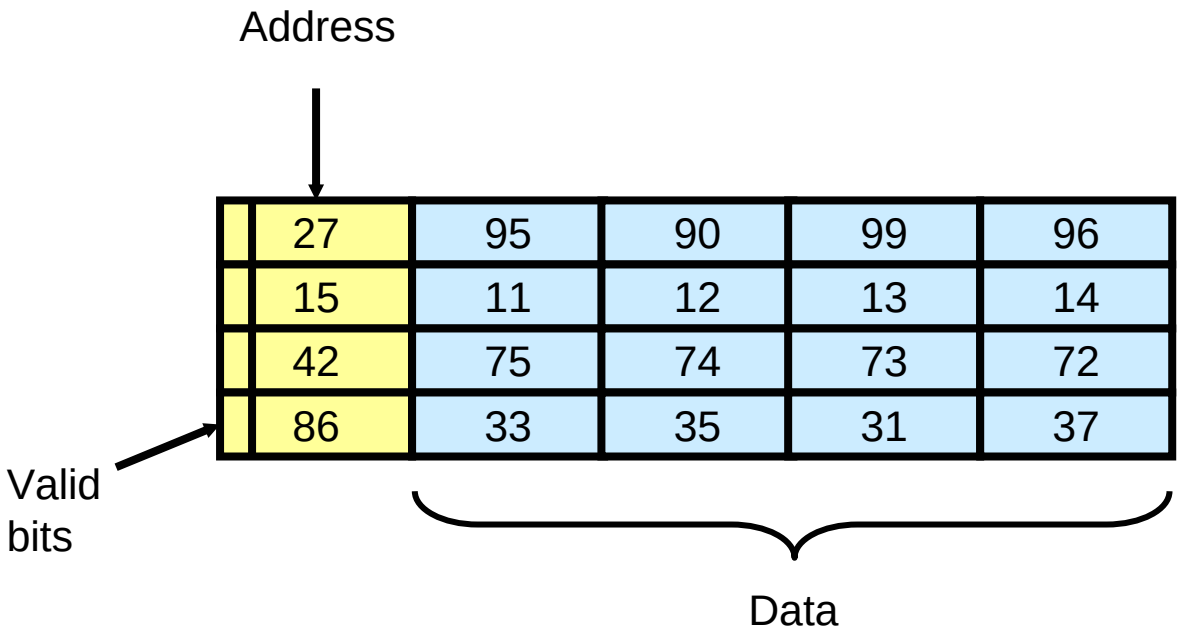
# Lecture 13: Improving Cache Performance

- Logistics:
  - Return HW #2, except for two students.
  - No Wednesday office hour this week
- Last Time:
  - Cache introduction
    - AMAT
    - Set associativity
- Today
  - Replacement and write policies
  - Cache performance optimizations

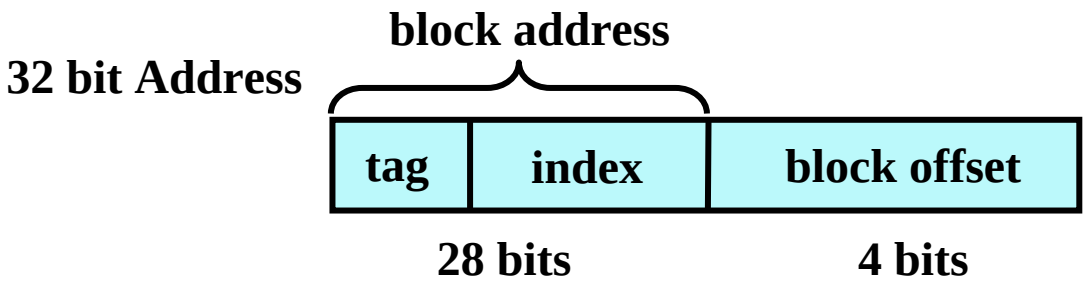
# Cache Organization




- **Where does a block get placed?**
- **How do we find it?**
- **Which one do we replace when a new one is brought in?**
- **What happens on a write?**

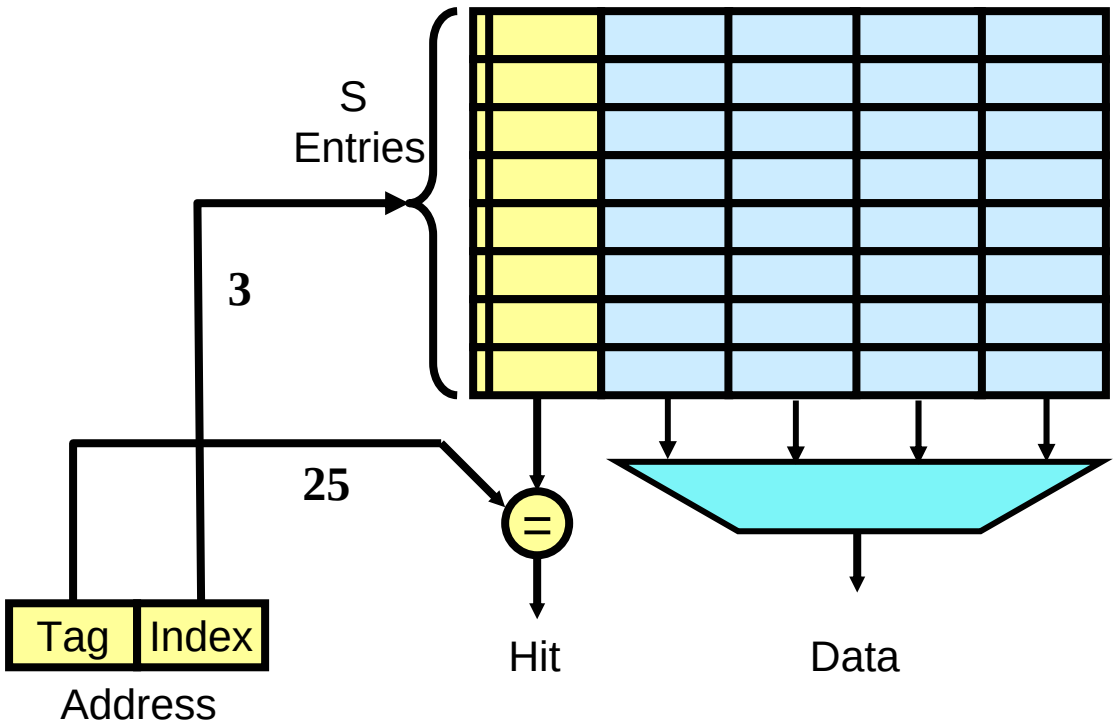
# How Do We Find a Block in The Cache?

- Our Example:
  - Main memory address space = 32 bits (= 4GBytes)
  - Block size = 4 words = 16 bytes
  - Cache capacity = 8 blocks = 128 bytes



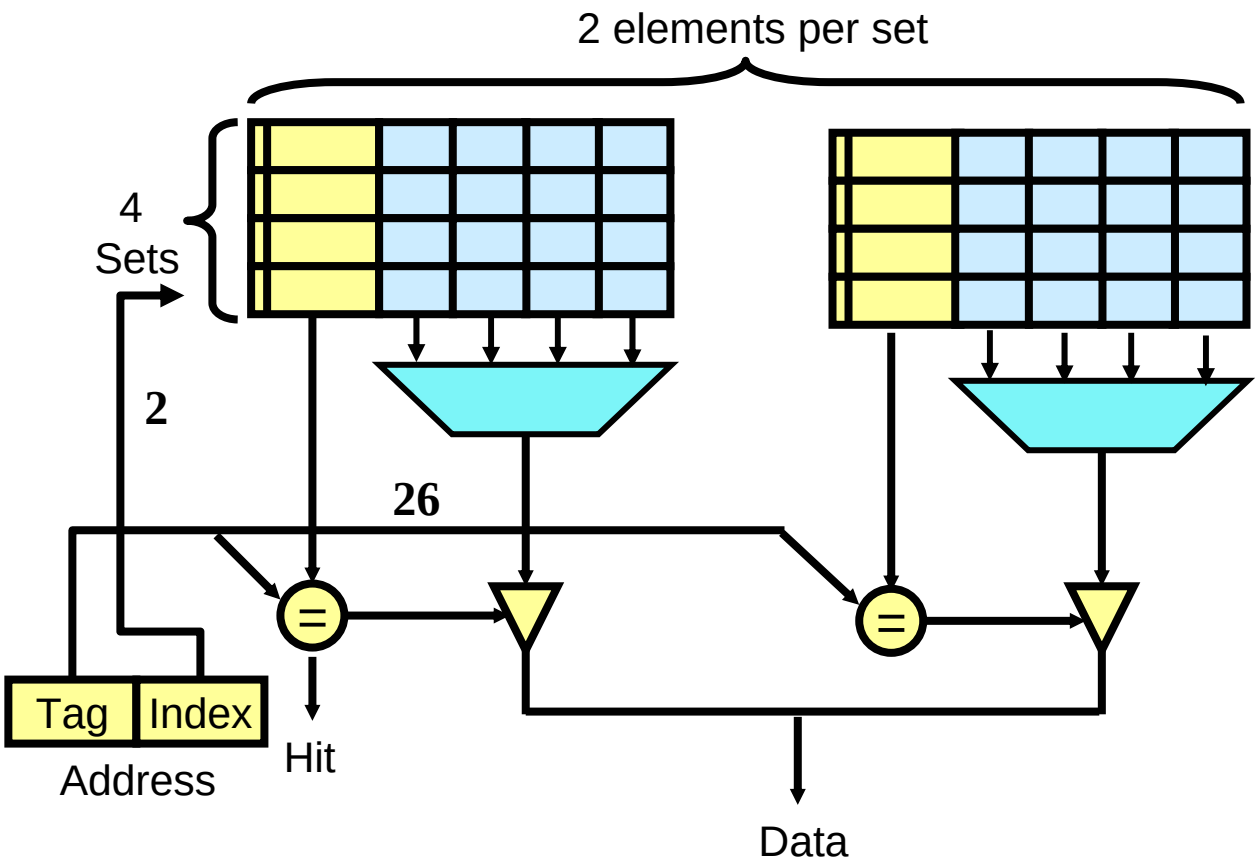

- Valid bit $\Rightarrow$ is cache block good?
- index $\Rightarrow$ which set
- tag $\Rightarrow$ which data/instruction in block
- block offset $\Rightarrow$ which word in block
- # tag/index bits determine the associativity
- tag/index bits can come from anywhere in block address

# Finding a Block: Direct-Mapped




**With cache capacity = 8 blocks**

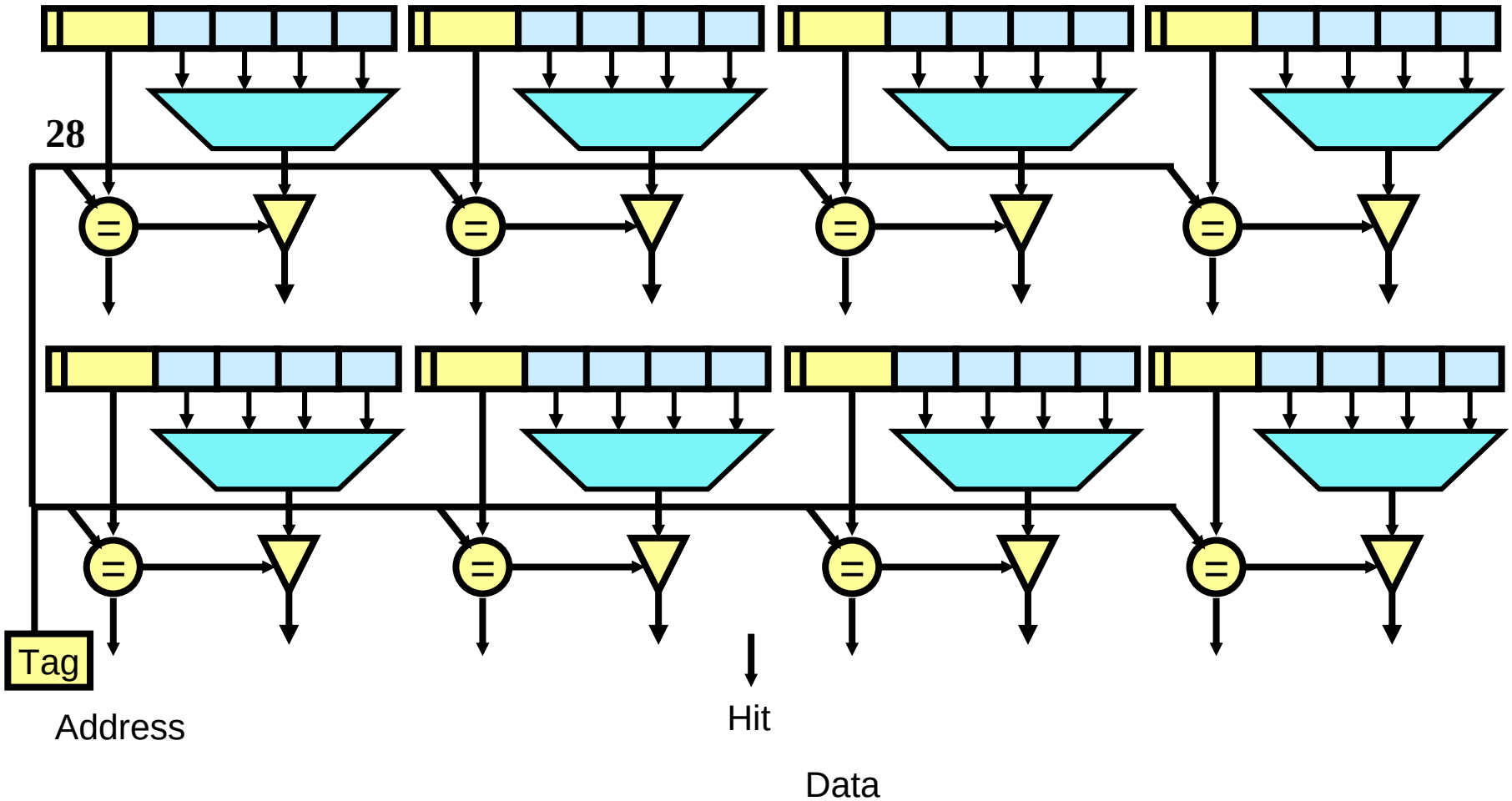
# Finding A Block: 2-Way Set-Associative



# Set Associative Cache

- S - sets
- A - elements in each set
  - A-way associative
- In the example, S=4, A=2
  - 4-way associative 8-entry cache
- All of main memory is divided into S sets
  - All addresses in set N map to same set of the cache
    - Addr = N mod S
    - A locations available
- Shares costly comparators across sets
- Low address bits select set
  - 2 in example
- High address bits are *tag*, used to associatively search the selected set
- Extreme cases
  - A=1: Direct mapped cache
  - S=1: Fully associative
- A need not be a power of 2

# Finding A Block: Fully Associative



# Which Block Should Be Replaced on Miss?

- Direct Mapped
  - Choice is easy - only one option
- Associative
  - Randomly select block in set to replace
  - Least-Recently used (LRU)
- Implementing LRU
  - 2-way set-associative
  - >2 way set-associative

# What Happens on a Store?

- Need to keep cache consistent with main memory
  - Reads are easy - no modifications
  - Writes are harder - when do we update main memory?

- Write-Through
  - On cache write - always update main memory as well
  - Use a write buffer to stockpile writes to main memory for speed

- Write-Back
  - On cache write - remember that block is modified (dirty bit)
  - Update main memory when dirty block is replaced
  - Sometimes need to flush cache (I/O, multiprocessing)

# BUT: What if Store Causes Miss!

- Write-Allocate
  - Bring written block into cache
  - Update word in block
  - Anticipate further use of block

- No-write Allocate
  - Main memory is updated
  - Cache contents unmodified

# Miss Classifications

- Compulsory misses
  - First time data is accessed
- Capacity misses
  - Working set larger than cache size
- Conflict misses
  - One set fills up, but room in other sets

# How Do We Improve Cache Performance?

$$T_{access} = t_{hit} + p_{miss} \bullet penalty_{miss}$$

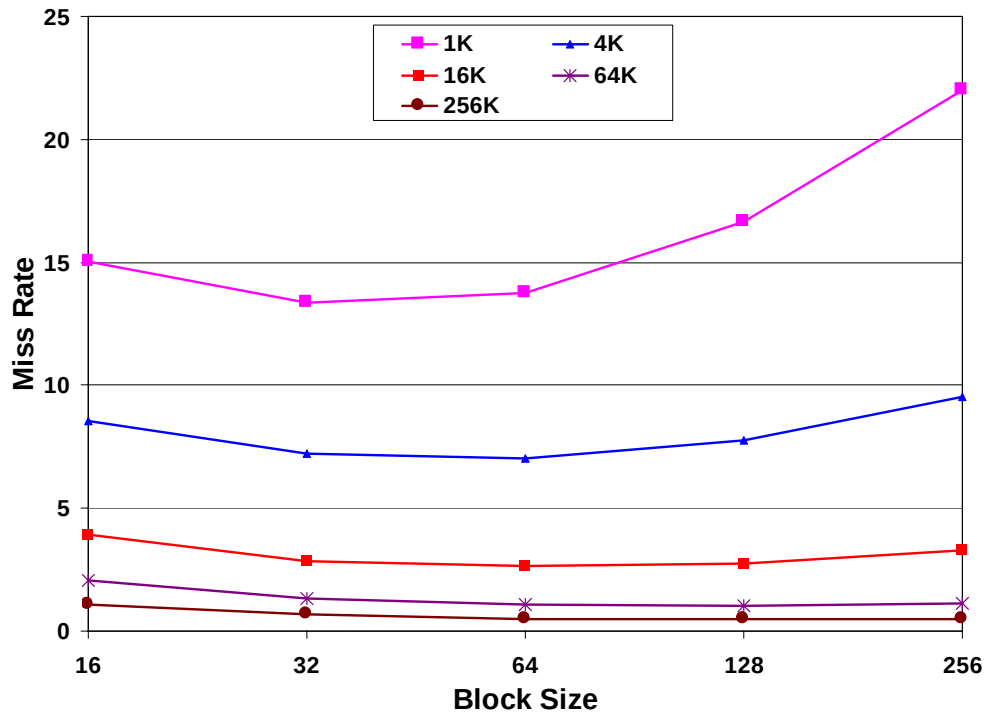
# How Do We Improve Cache Performance?

$$T_{access} = t_{hit} + p_{miss} \bullet penalty_{miss}$$

- Reduce miss rate
- Reduce miss penalty
- Reduce hit time

# Reducing Miss Rate: Increase Block Size

- Fetch more data with each cache miss
  - 16 bytes ⇒ 64, 128, 256 bytes!
  - Works because of Locality (spatial)



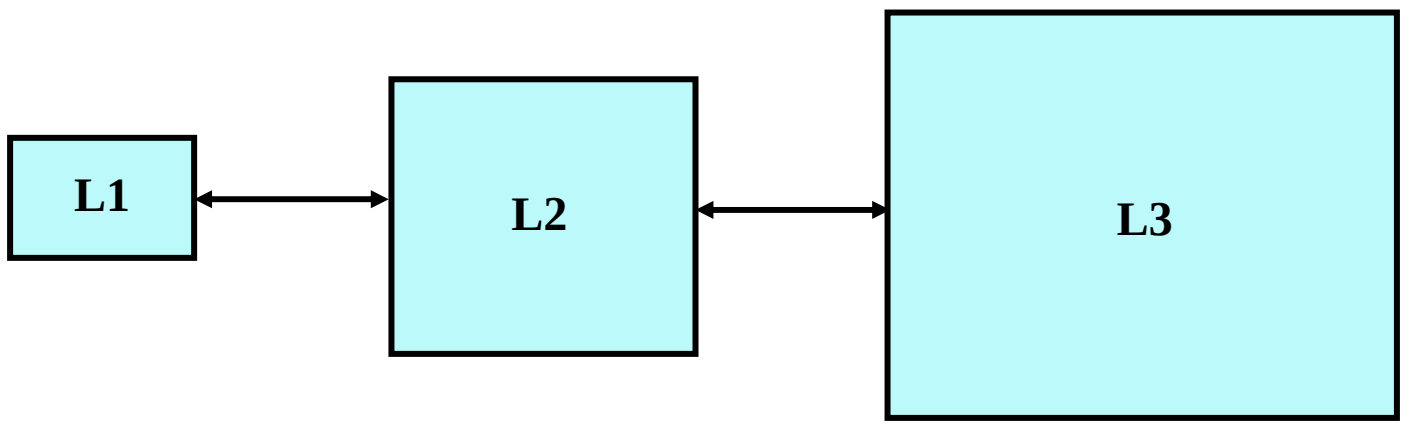
# Reducing Miss Rate: Increase Associativity

- Reduce conflict misses
- Rules of thumb
  - 8-way = fully associative
  - Direct mapped size N = 2-way set associative size N/2

- But!
  - Size N associative is larger than Size N direct mapped
  - Associative typically slower that direct mapped ($t_{hit}$ larger)

# Reduce Miss Penalty: More Cache Levels

- Average access time = $HitTime_{L1}$ + $MissRate_{L1}$ * $MissPenalty_{L1}$
- $MissPenalty_{L1}$ = $HitTime_{L2}$ + $MissRate_{L2}$ * $MissPenalty_{L2}$
- etc.
- Size/Associativity of higher level caches?



# Reduce Miss Penalty: Transfer Time

- We can increase the width of memory and the bus in order to decrease access times and transfer times to move things into cache.
- We can have memory "banks" that allow us to read or write multiple words in 1 access time: called *interleaving* (don't need to change size of bus).

# Reducing Hit Time

- Make Caches small and simple
  - Hit Time = 1 cycle is good (3.3ns!)
  - L1 - low associativity, relatively small
- Even L2 caches can be broken into sub-banks
  - Can exploit this for faster hit time in L2

# Summary

- Today:
  - Replacement and write policies
  - Cache performance optimizations
- Next Time – Steve Keckler guest lecture
  - HW #4 due
  - More cache optimizations: Victim caches and prefetching
  - Introduction to memories and virtual addressing