

Logistics

- Homework #3 is due
- Homework #4 handed out – due in one week
 - Some problems from end of chapter 6
 - Some problems from first two sections of chapter 7
 - This material will be on the midterm
 - I will put solutions to HW #4 online 2-3 days before the midterm
- Last Time
 - Details of handling pipeline hazards
 - Bypassing
 - Pipeline bubbles
 - Delay slots

Lecture 13: Caching

- What part of the pipeline have we been glossing over?
- Memory – important to overall machine performance.

Memory System Overview

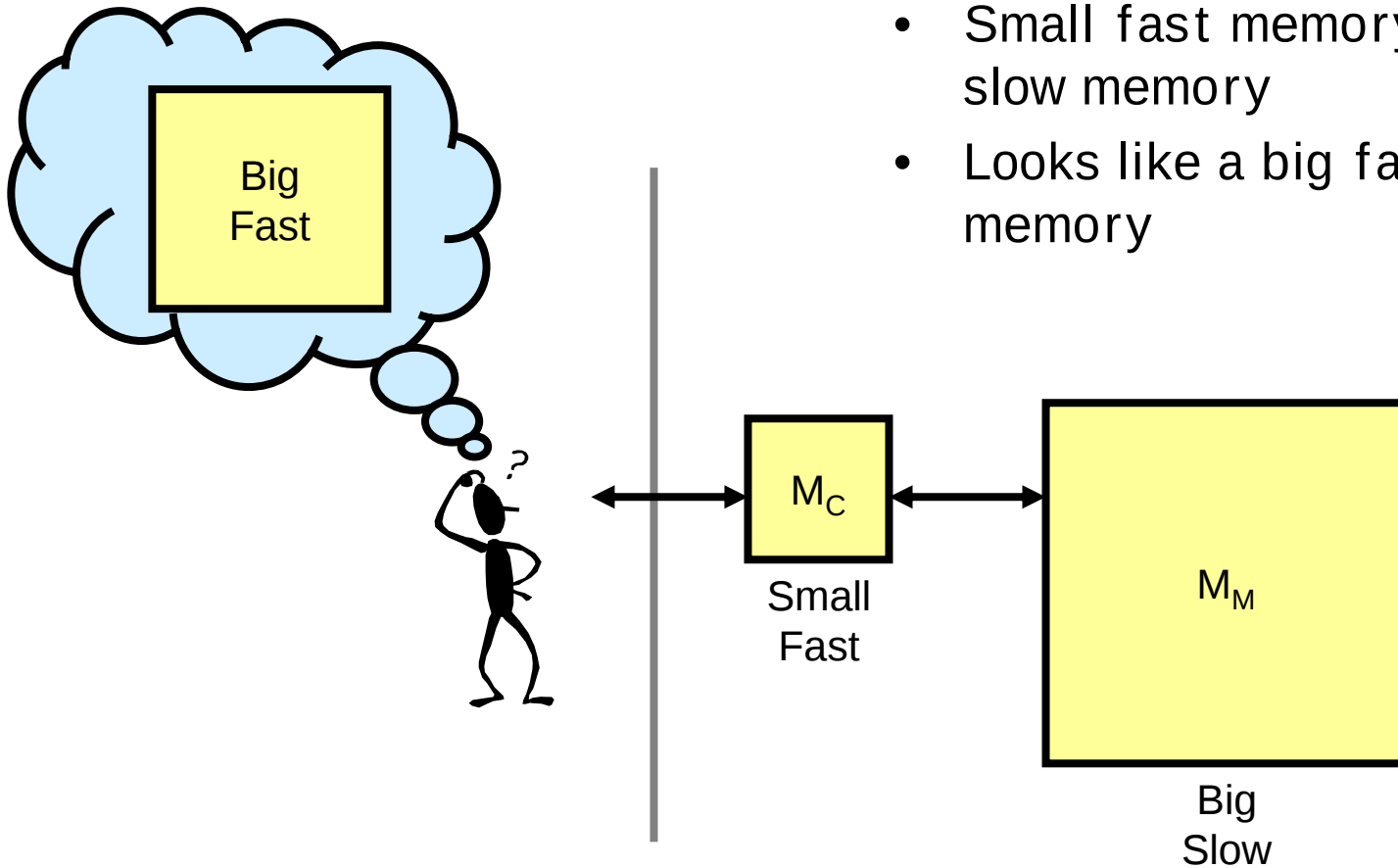
- Memory Hierarchies
 - Latency/Bandwidth/Locality
 - Caches
 - Principles - why does it work
 - Cache organization
 - Cache performance
 - Types of misses (the 3 Cs)
 - Main memory organization
 - DRAM vs. SRAM
 - Bank organization
 - Tracking multiple references
 - Trends in memory system design
- Logical Organization
 - Name spaces
 - Protection and sharing
 - Resource management
 - virtual memory, paging, and swapping
 - Segmentation
 - Capability-based addressing

The Memory Bottleneck

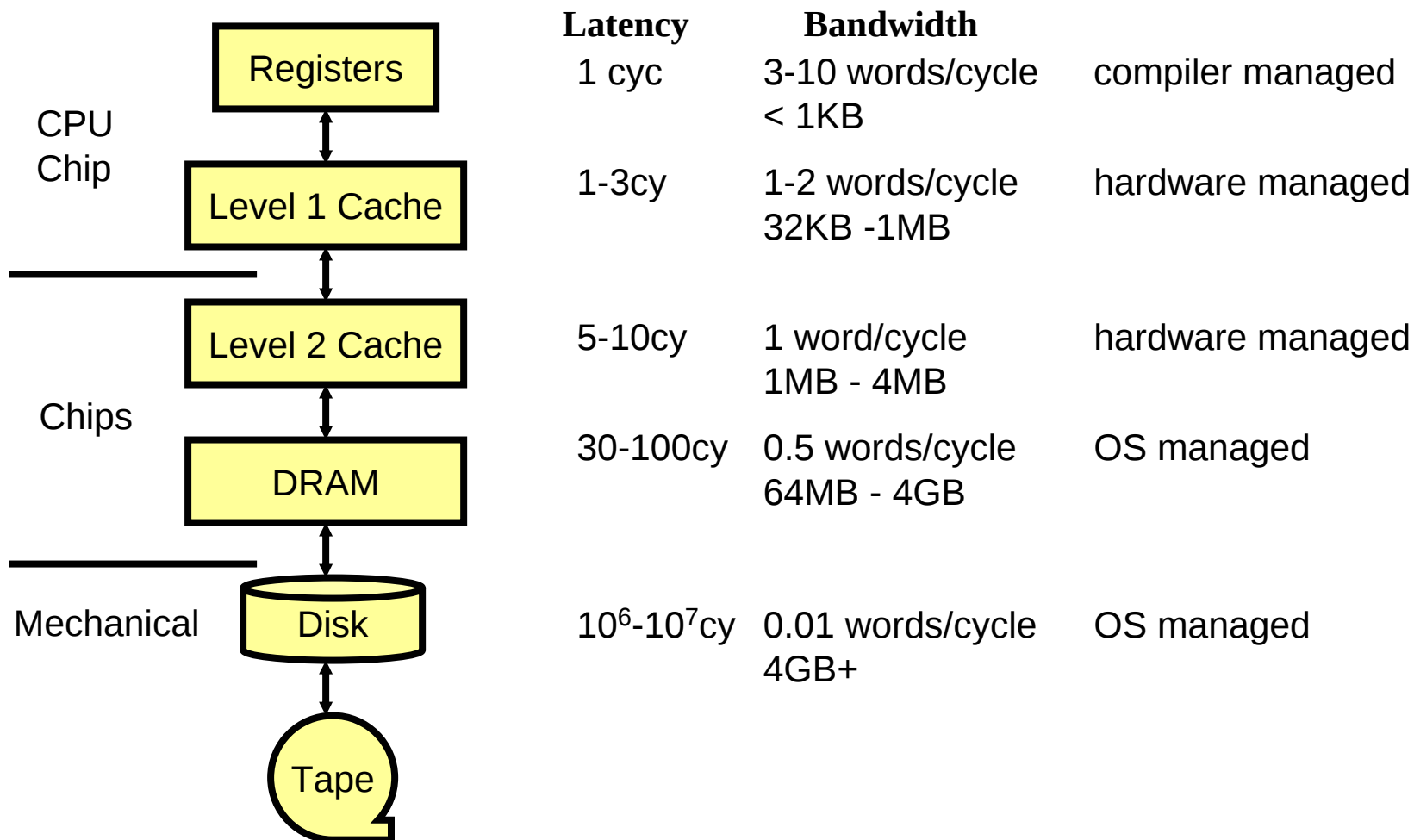
- Typical CPU clock rate
 - 3 GHz (0.33 ns cycle time)
- Typical DRAM access time
 - 30ns (about 100 cycles)
- Typical main memory access
 - 70ns (210 cycles)
 - DRAM (30), precharge (10), chip crossings (15), overhead (15).
- Our pipeline designs assume 1 cycle access
- Average instruction references
 - 1 instruction word
 - 0.3 data words
- This problem gets worse
 - CPUs get *faster*
 - Memories get *bigger*
- Memory delay is mostly communication time
 - reading/writing a bit is *fast*
 - it takes time to
 - select the right bit
 - route the data to/from the bit
- Big memories are *slow*
- Small memories can be made *fast*

Cache Memory

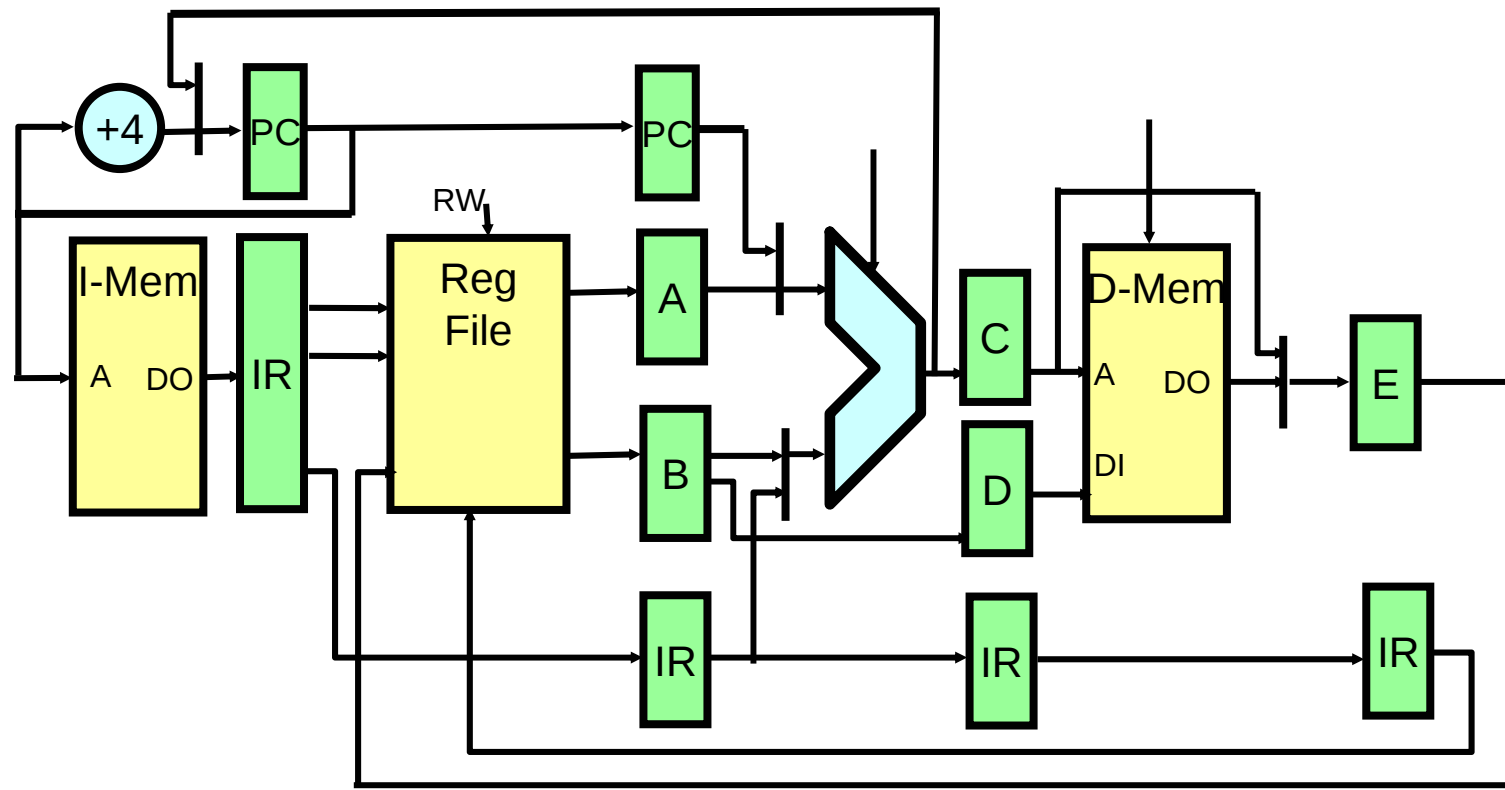
- Small fast memory + big slow memory
- Looks like a big fast memory



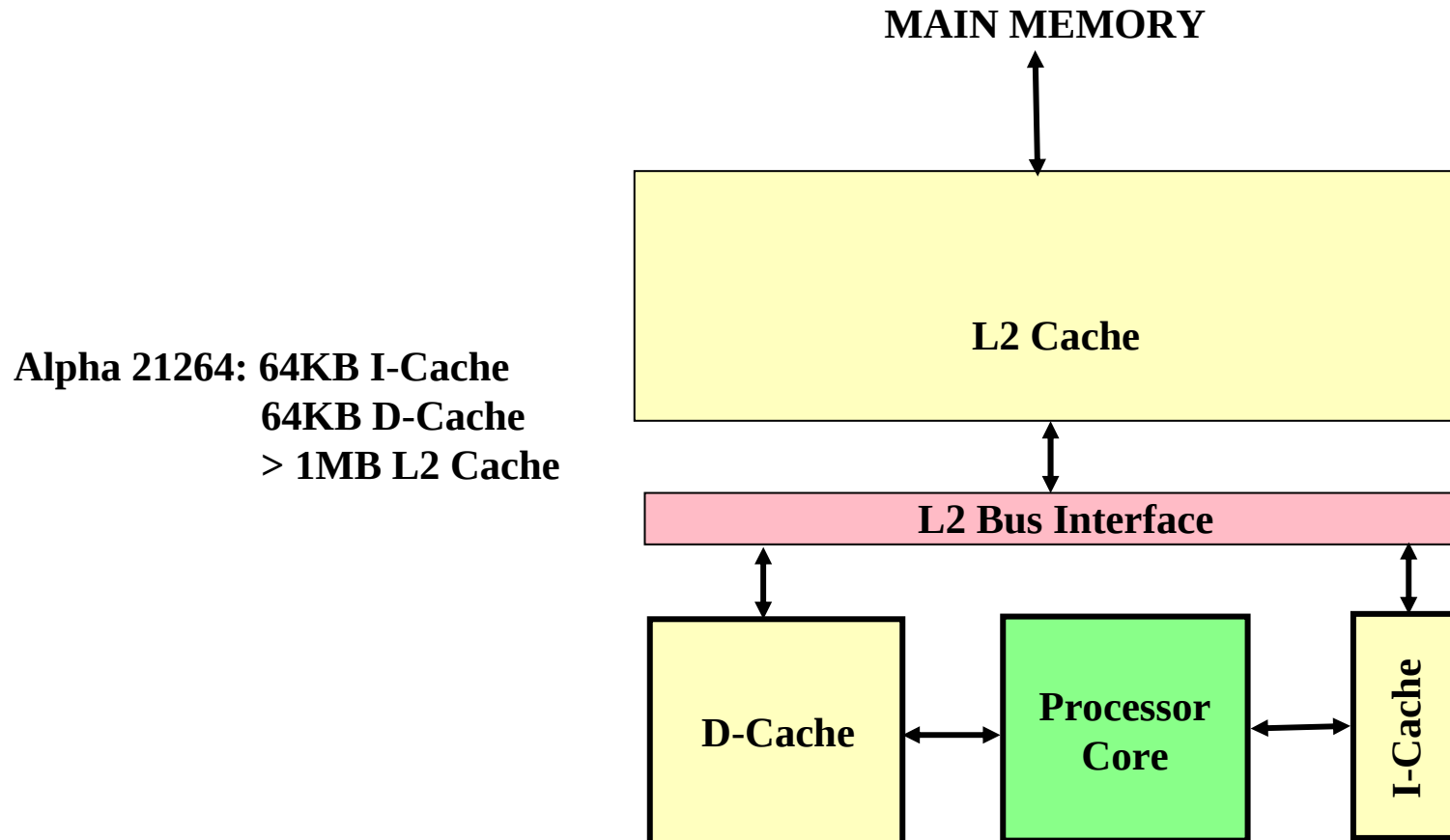
The Memory Hierarchy



Where Does the Memory Hierarchy Fit In?

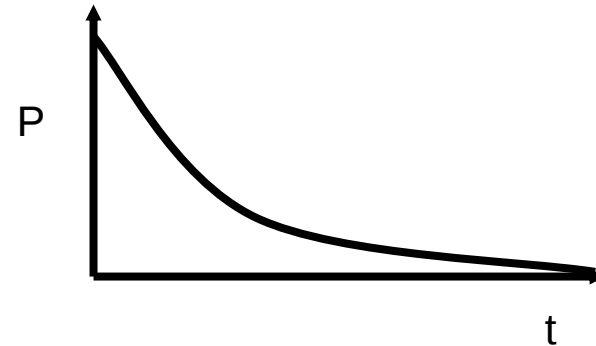
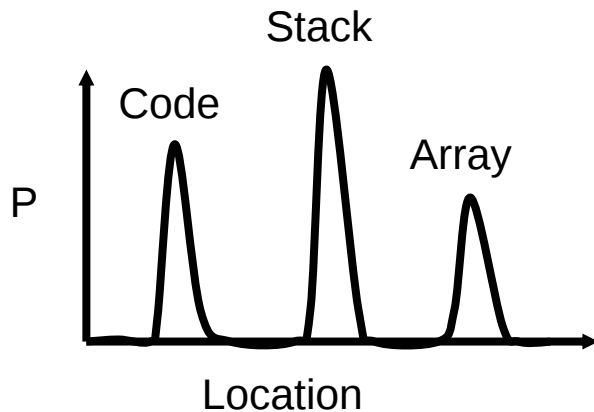


Typical Cache Organization



Locality of Reference

- Spatial Locality
 - likely to reference data *near* recent references
- Temporal Locality
 - likely to reference the same data that was referenced recently

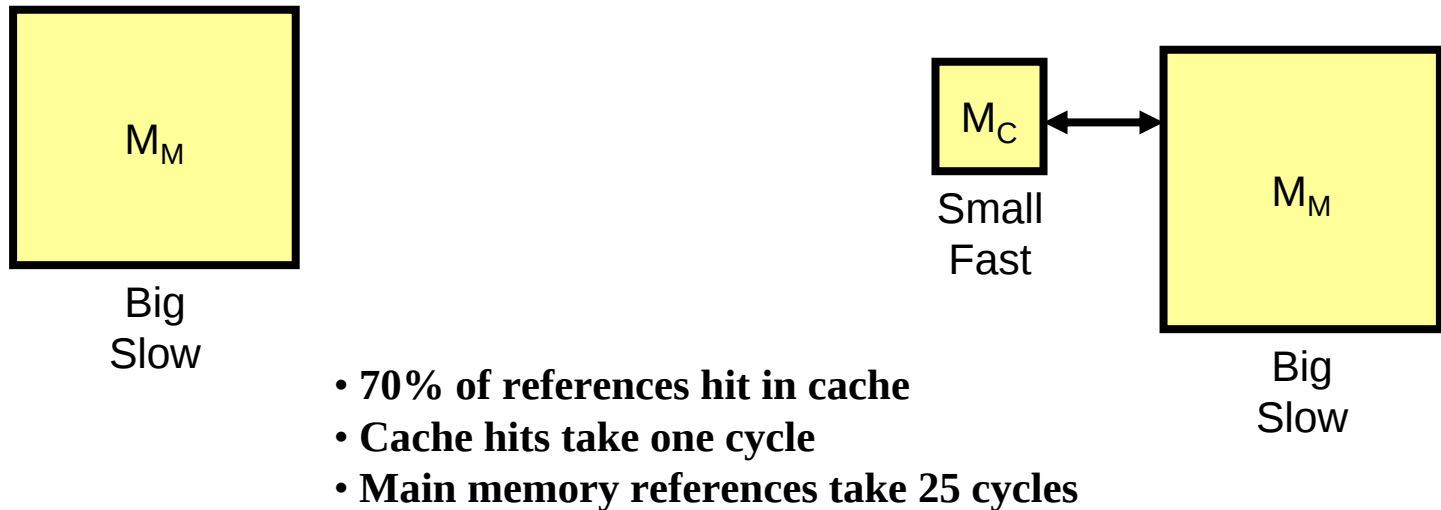


Program Behavior

- Locality depends on type of program
- Some programs 'behave' well
 - small loop operating on data on stack
- Some programs don't
 - frequent calls to nearly random subroutines
 - traversal of large, sparse data set
 - essentially random data references with no reuse
- Most programs exhibit some degree of locality

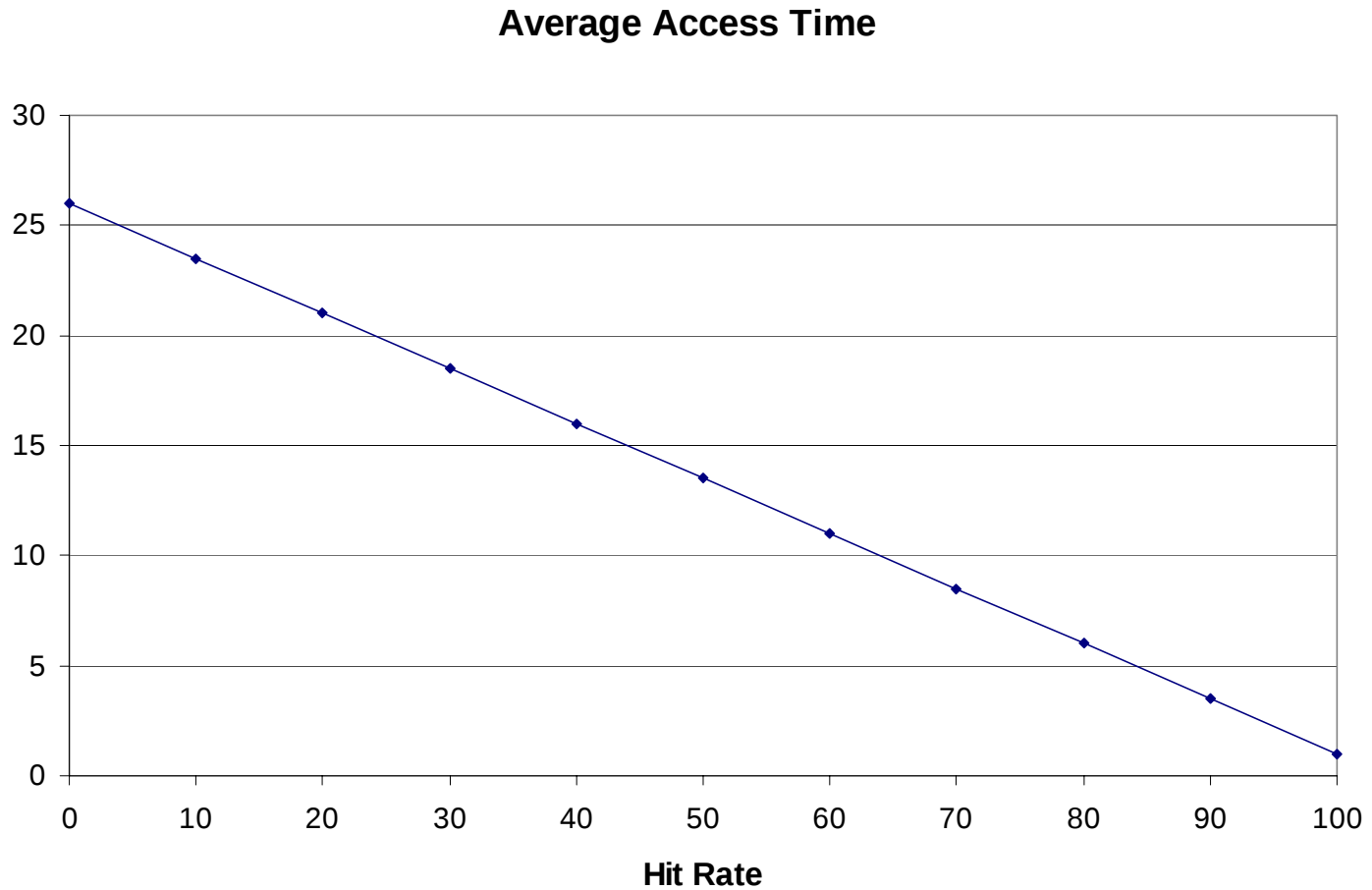
Example

What is the average memory access time?

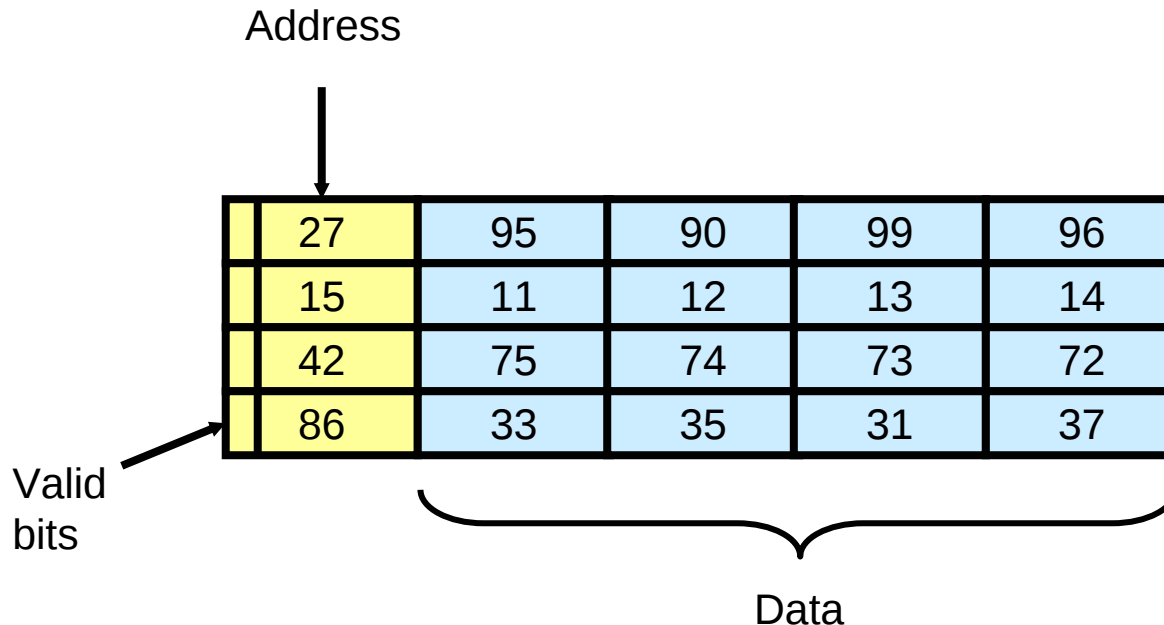


$$AMAT = \text{Latency}_{\text{Hit}} + P(\text{miss}) * \text{Latency}_{\text{Miss}}$$

Impact of Hit Rate



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?

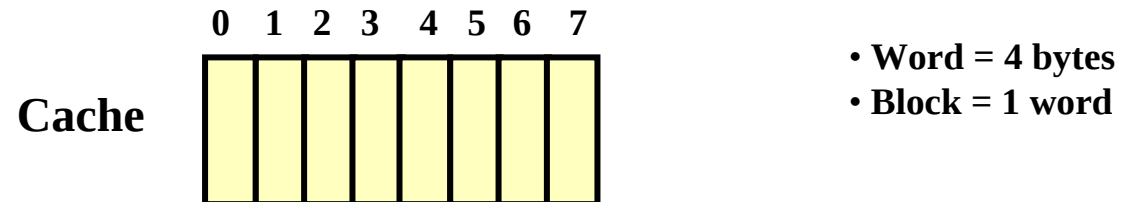
Cache Definitions

- Cache block (= cache line)

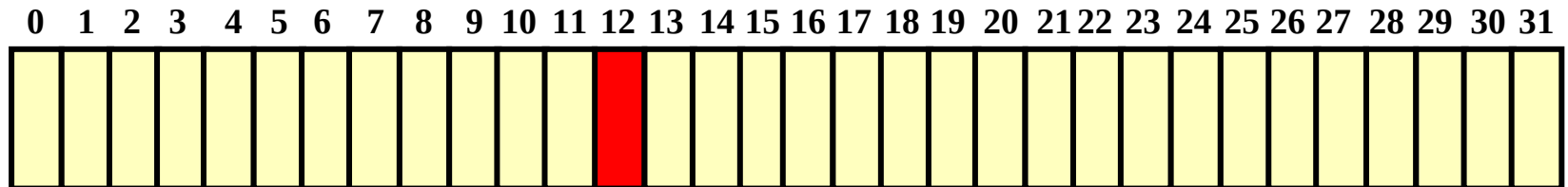
- Miss rate
- Miss penalty
- Index
- Tag
- Offset

	0x0	0x4	0x8	0xc
0x0000				
0x0010				
0x0020				
•				
•				
•				
0x00f0				

Where Does a Block Go in the Cache?



Main Memory

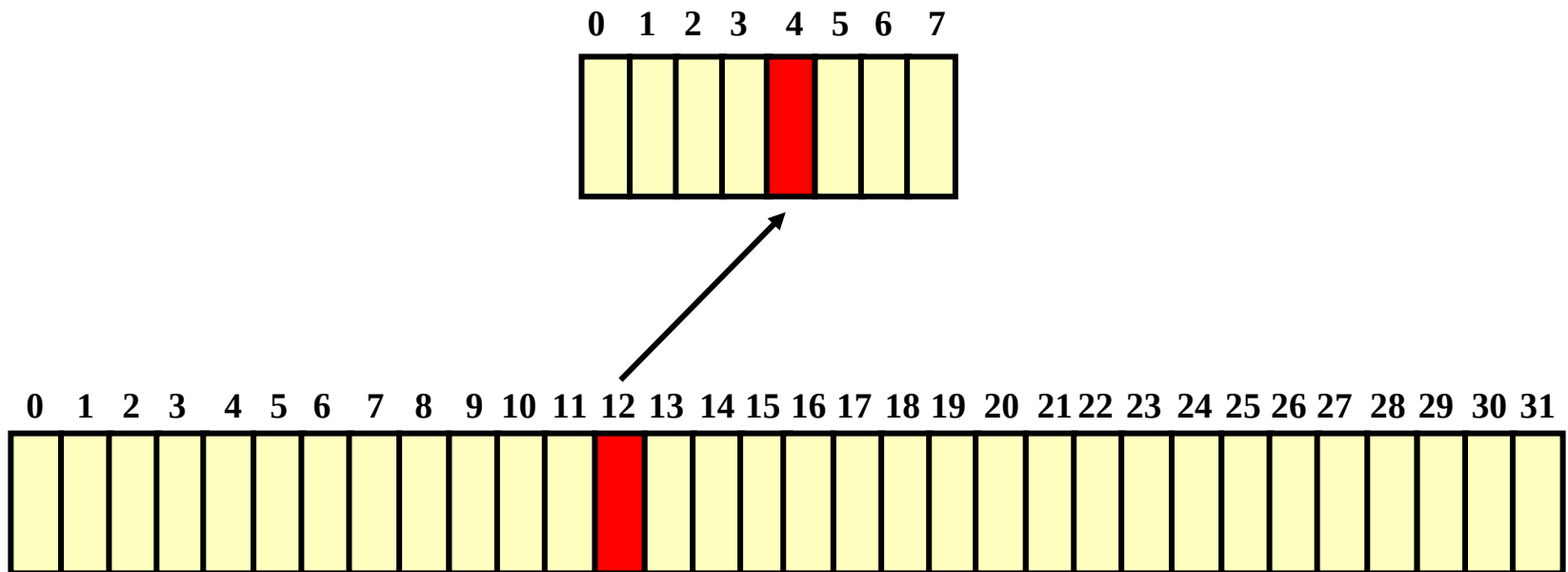


- Where do we put block 12?

Direct Mapped

- Each block mapped to exactly 1 cache location

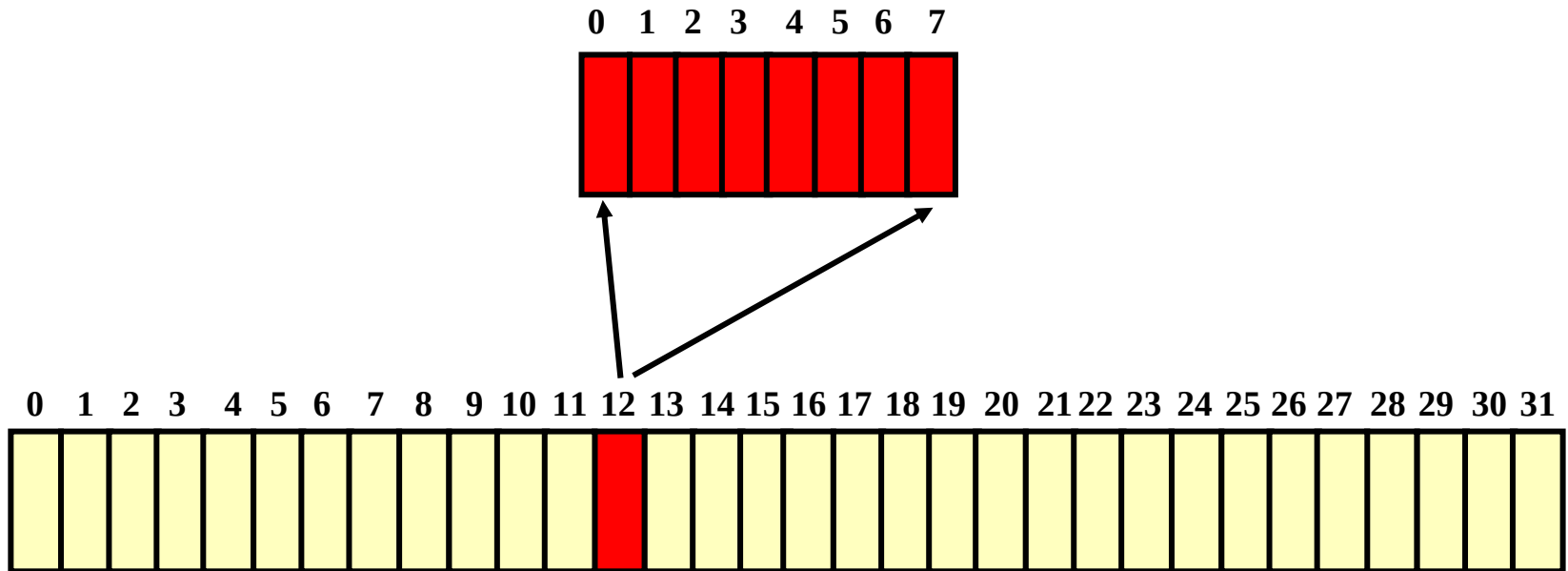
Cache location = (block address) MOD (# blocks in cache)



Fully Associative

- Each block mapped to any cache location

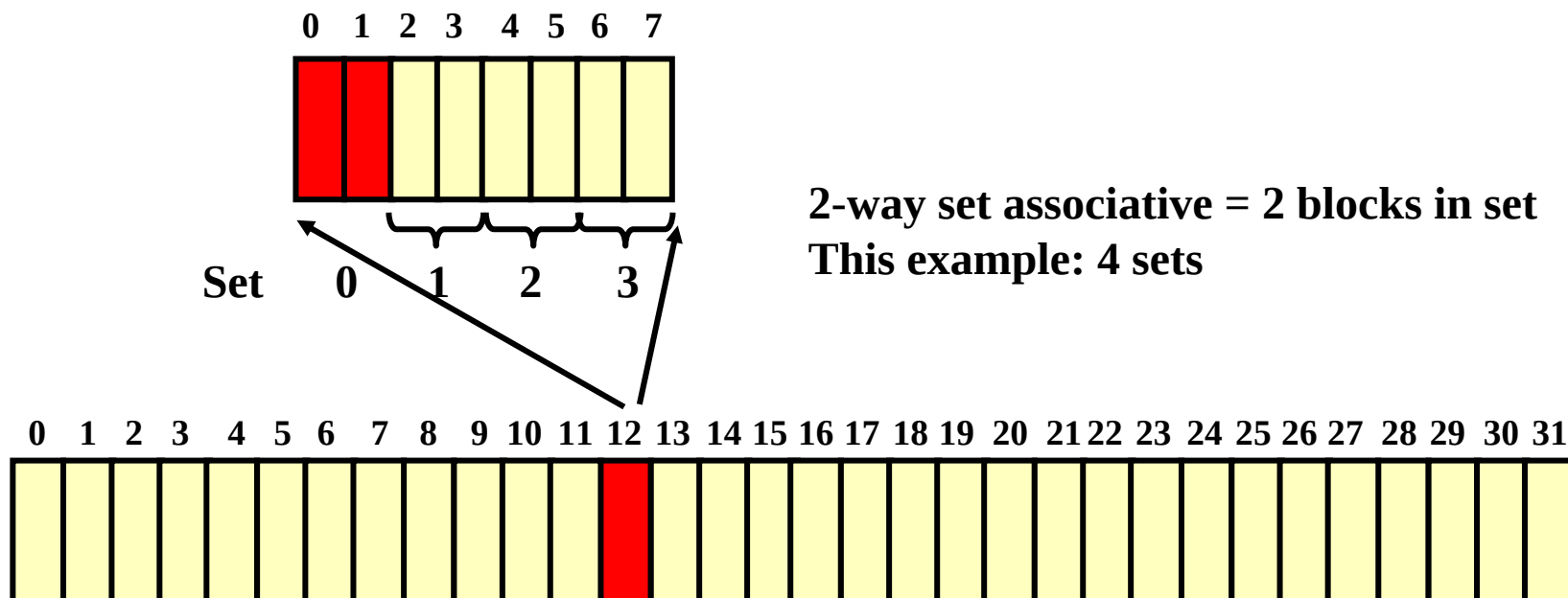
Cache location = any



Set Associative

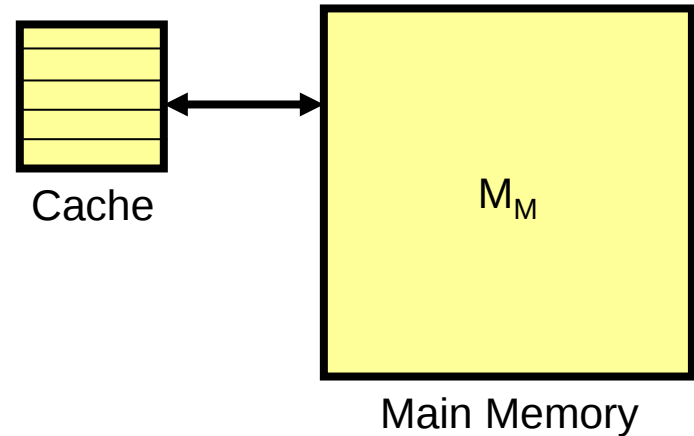
- Each block mapped to subset of cache locations

Set selection = (block address) MOD (# sets in cache)



Block Placement

- Mapping function from Big Memory to Small memory
- On block-by-block basis
 - Direct Mapped: 1 place
 - Fully Associative: Anywhere
 - Set Associative: Subset of cache
- Use address to do mapping and lookup

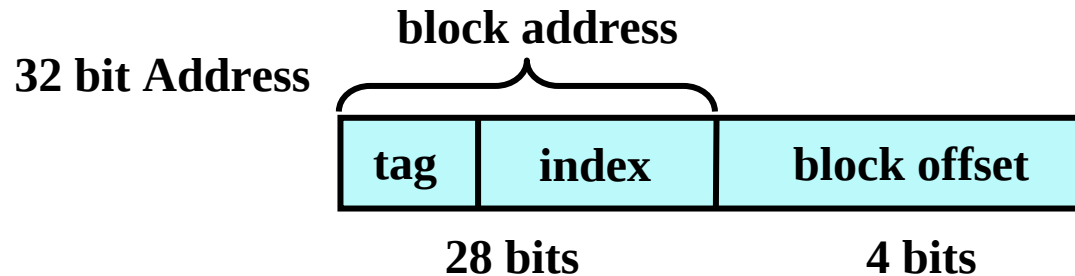


Taking advantage of Spatial Locality

- Instead of each block in cache being just 1 word, what if we made it 4 words?
- When we get our 1 word instruction or 1 word of data from memory to put in the cache, get the next 3 as well, because they are likely to be used soon!
- Need to add a way to choose which of the 4 words in the block we want when we go to cache... called **block offset**.

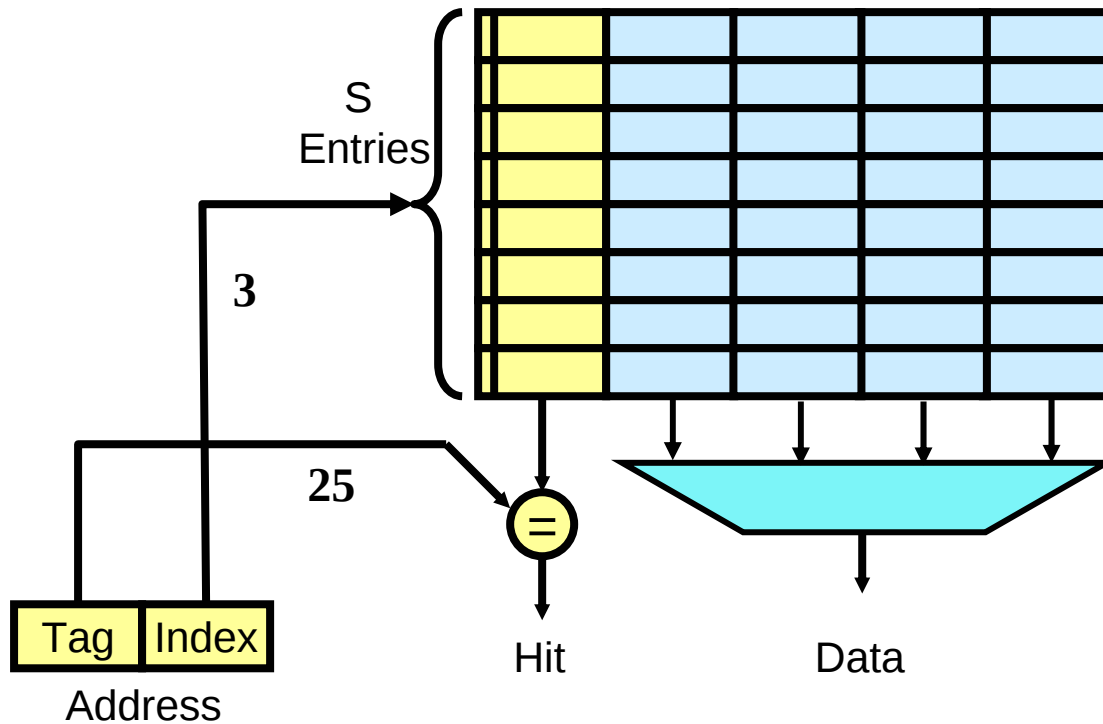
How Do We Find a Block in The Cache?

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



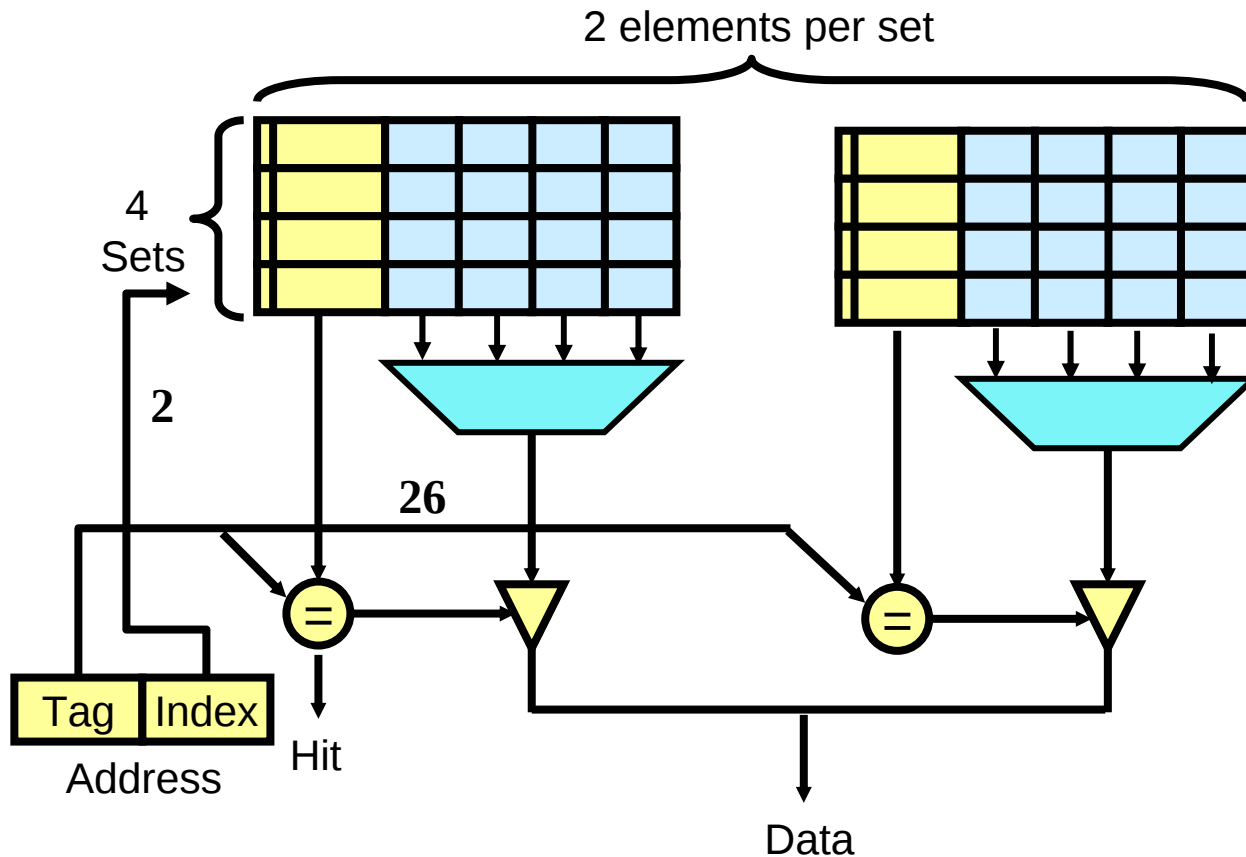
- Valid bit $\Rightarrow$ is cache block good?
- index $\Rightarrow$ which set
- tag $\Rightarrow$ which data/instruction in block
- block offset $\Rightarrow$ which word in block
- # tag/index bits determine the associativity
- tag/index bits can come from anywhere in block address

Finding a Block: Direct-Mapped



With cache capacity = 8 blocks

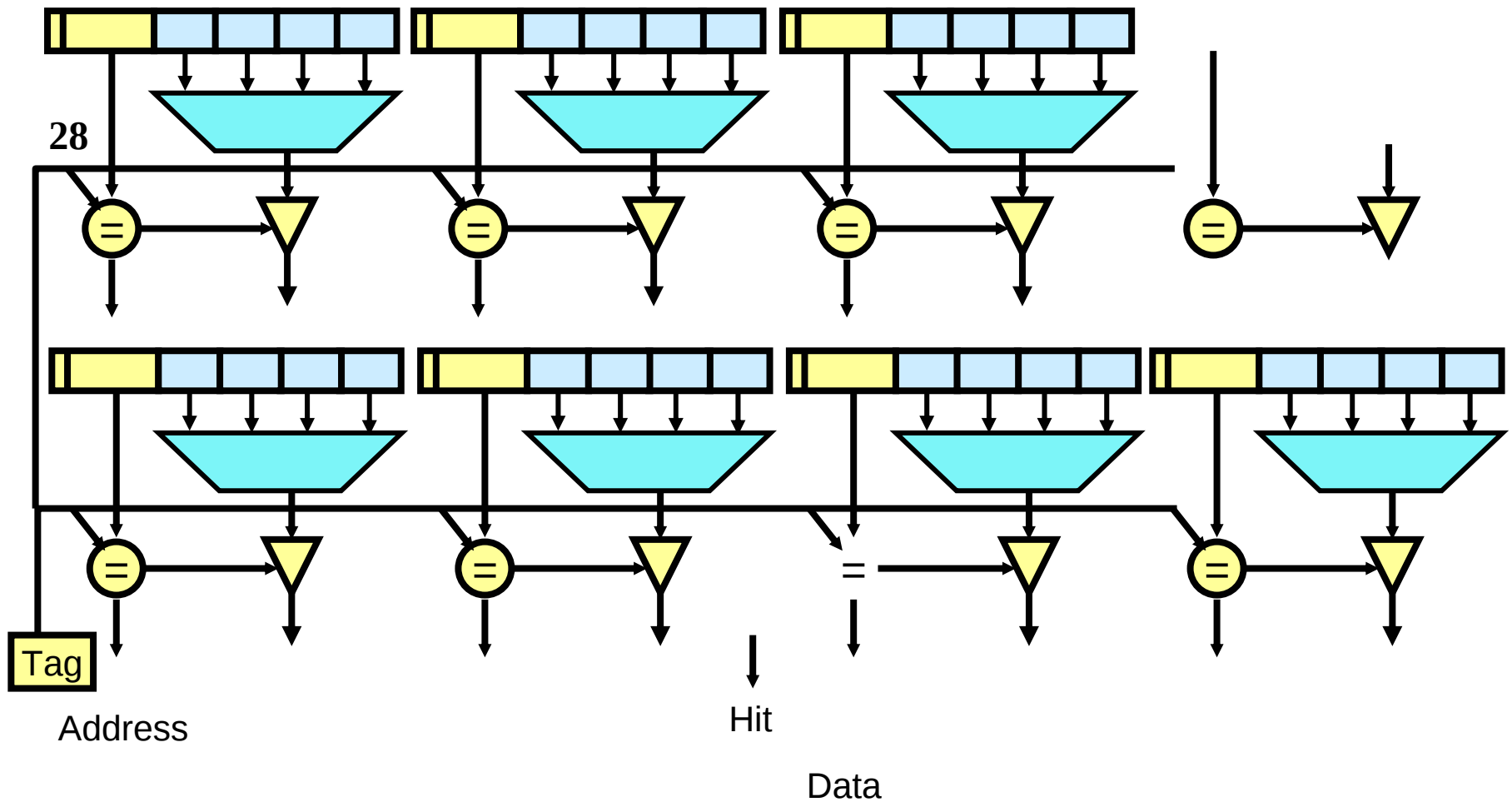
Finding A Block: 2-Way Set-Associative



Set Associative Cache

- S - sets
- A - elements in each set
 - A-way associative
- In the example, S=4, A=2
 - 2-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
 - $\text{Addr} = N \bmod S$
 - 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



Questions to think about

- As the block size goes up, what happens to the miss rate?
- ... what happens to the miss penalty?
- ... what happens to hit time?
- As the associativity goes up, what happens to the miss rate?
- ... what happens to the hit time?