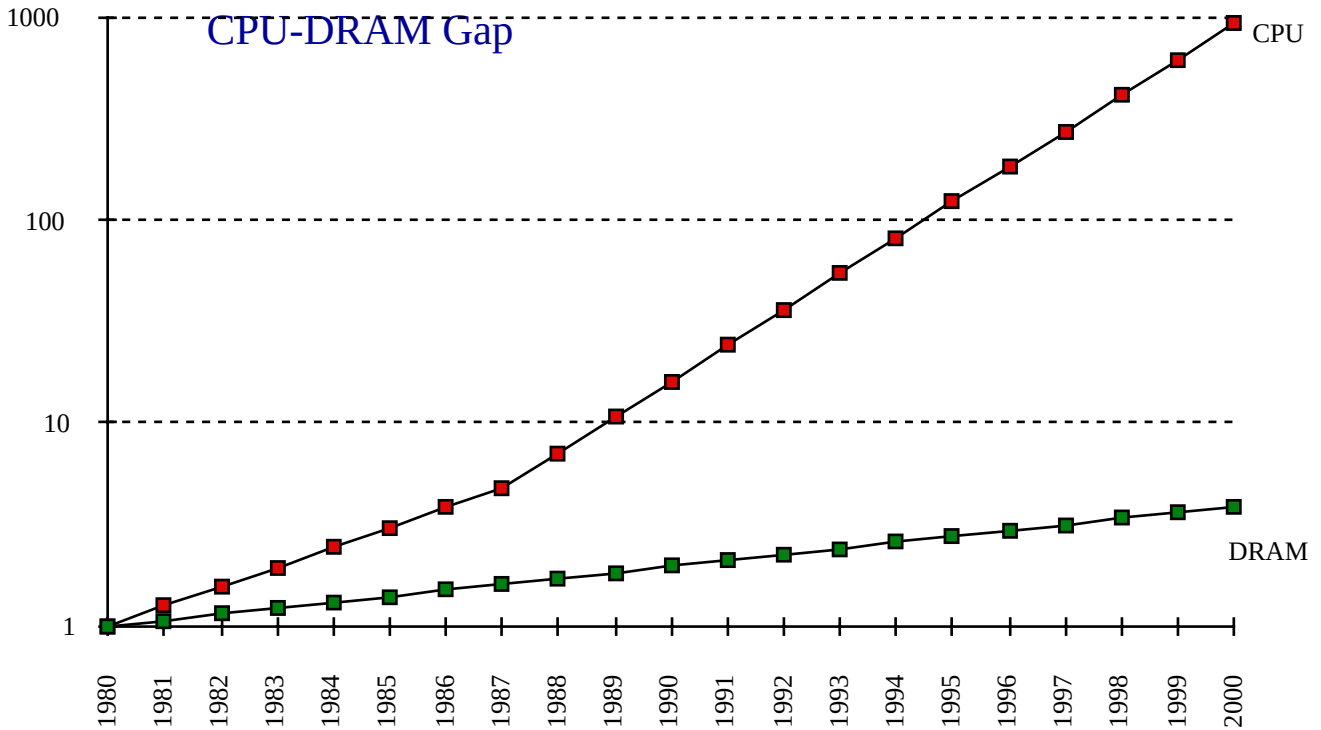
Lecture 8

Memory Hierarchy Design

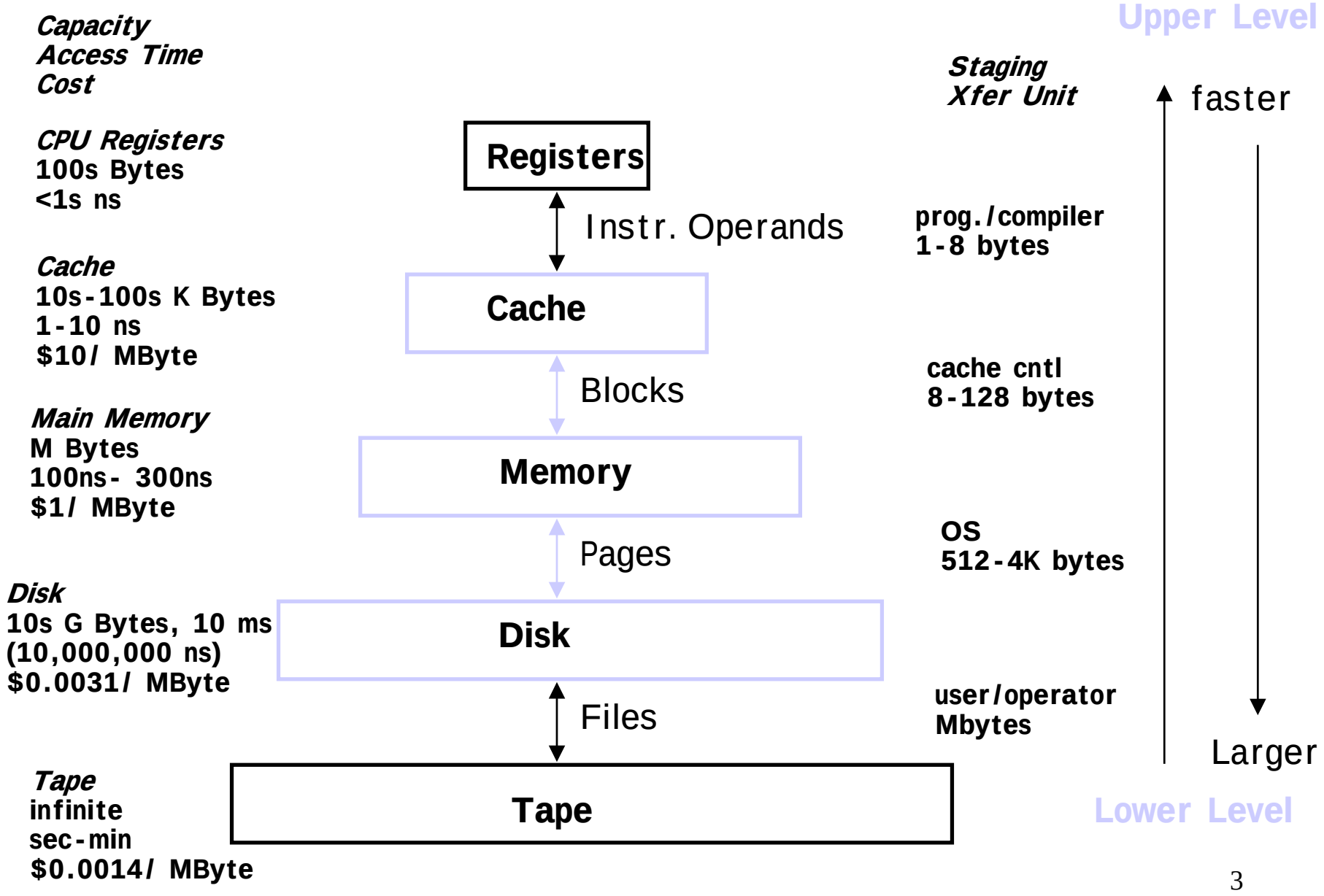
Who Cares about Memory Hierarchy?

- Processor Only Thus Far in Course
  - CPU cost/performance, ISA, Pipelined Execution



- 1980: no cache in µproc;
  1995 2-level cache, 60% trans. on Alpha 21164 µproc

Levels of the Memory Hierarchy

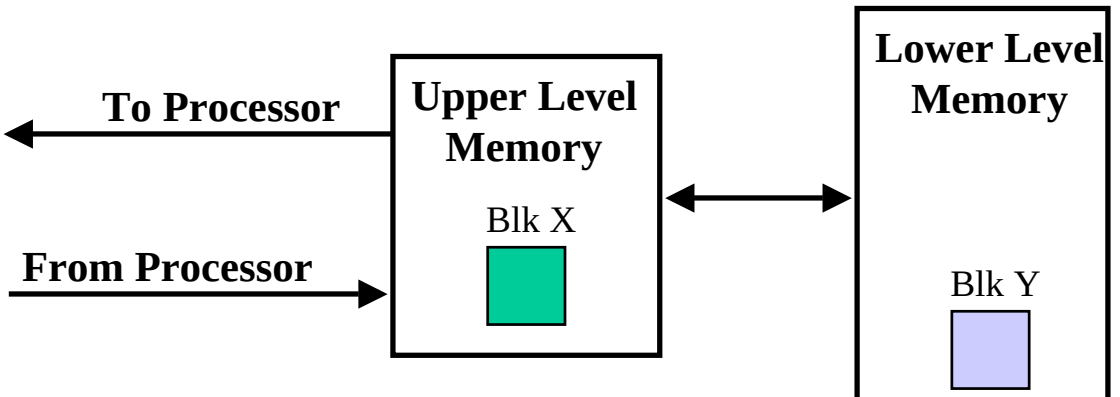


General Principle

- The Principle of Locality:
  - Program access a relatively small portion of the address space at any instant of time.
- Two Different Types of Locality:
  - Temporal Locality (Locality in Time):
    - If an item is referenced, it will tend to be referenced again soon (e.g., loops, reuse)
  - Spatial Locality (Locality in Space):
    - If an item is referenced, items whose addresses are close by tend to be referenced soon (e.g., straightline code, array access)
- Locality + smaller HW is faster = memory hierarchy
  - *Levels*: each smaller, faster, more expensive/byte than level below
  - *Inclusive*: data found in top also found in the bottom
- Definitions
  - *Upper* is closer to processor
  - *Block*: minimum unit that present or not in upper level

Memory Hierarchy: Terminology

- Hit: data appears in some block in the upper level (example: Block X)
  - Hit Rate: the fraction of memory access found in the upper level
  - Hit Time: Time to access the upper level which consists of
    RAM access time + Time to determine hit/miss
- Miss: data needs to be retrieve from a block in the lower level (Block Y)
  - Miss Rate = 1 - (Hit Rate)
  - Miss Penalty: Time to replace a block in the upper level +
    Time to deliver the block the processor
- Hit Time << Miss Penalty (500 instructions on 21264!)

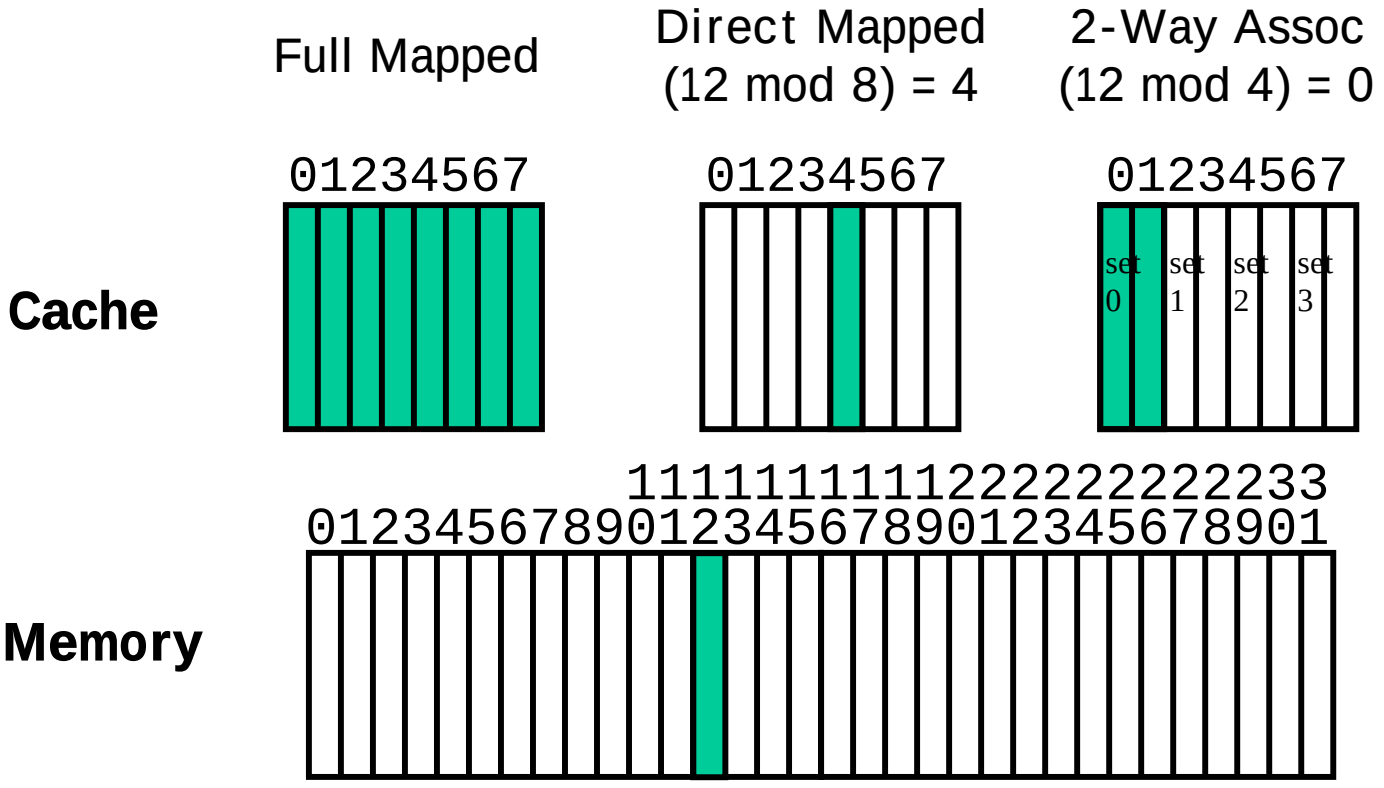


4 Questions for Memory Hierarchy

- Q1: Where can a block be placed in the upper level? *(Block placement)*
- Q2: How is a block found if it is in the upper level?
  (Block identification)
- Q3: Which block should be replaced on a miss?
  (Block replacement)
- Q4: What happens on a write?
  (Write strategy)

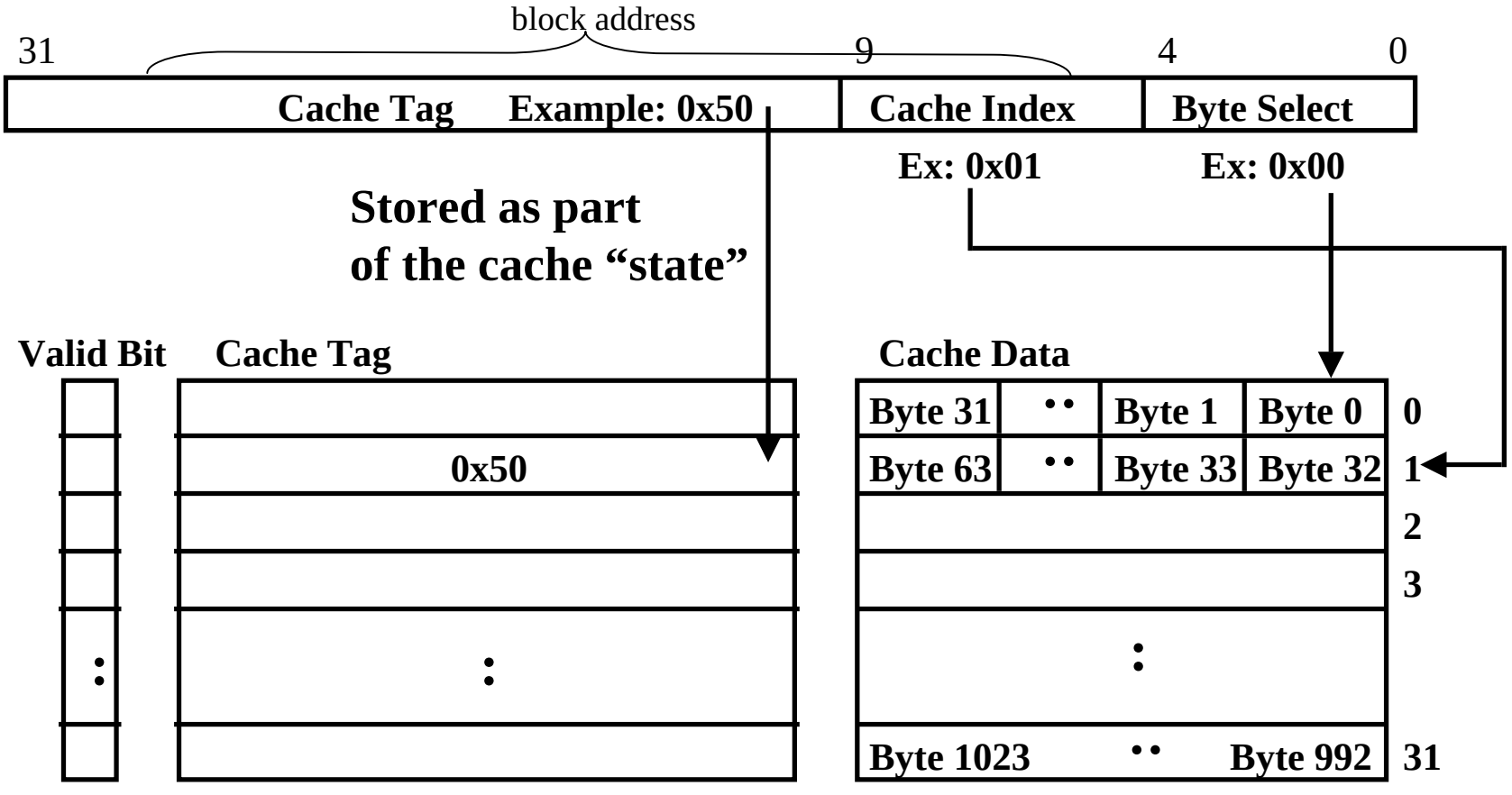
Q1: Where can a block be placed in the upper level?

- Block 12 placed in 8 block cache:
  - Fully associative, direct mapped, 2-way set associative
  - S.A. Mapping = Block Number Modulo Number Sets



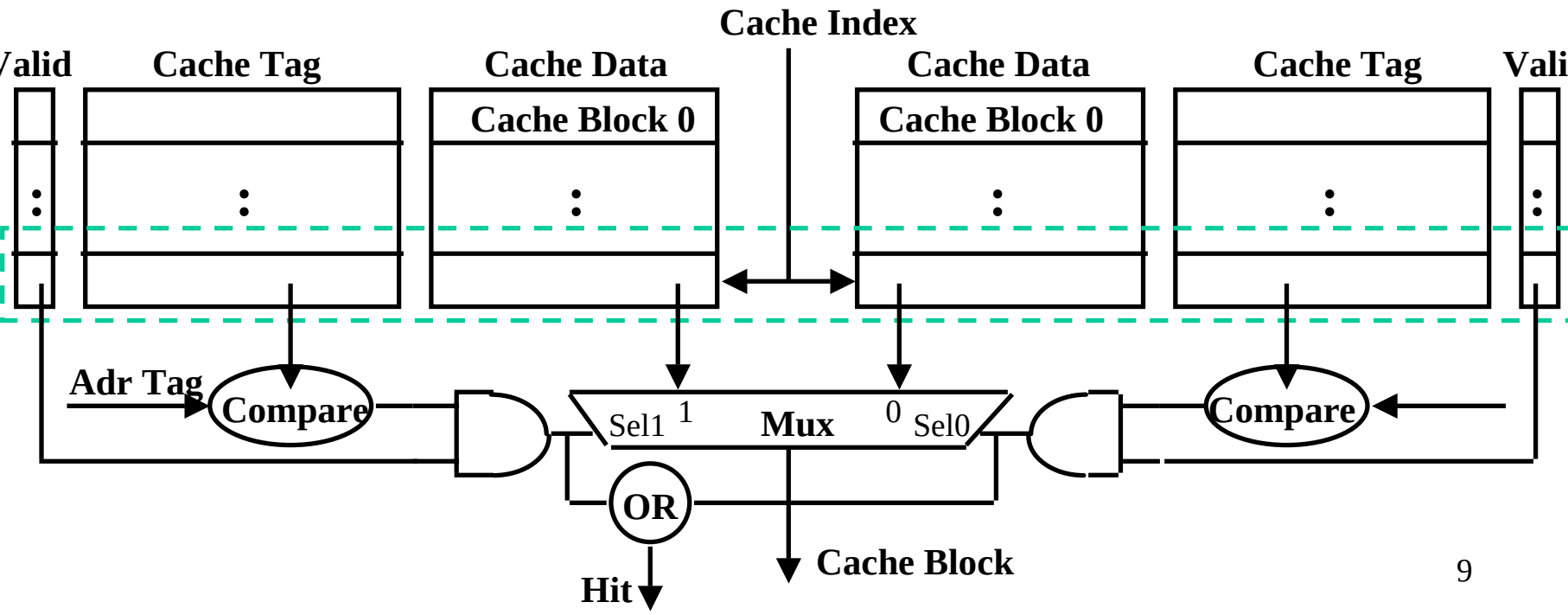
1 KB Direct Mapped Cache, 32B blocks

- For a 2 ** N byte cache:
  - The uppermost (32 - N) bits are always the Cache Tag
  - The lowest M bits are the Byte Select (Block Size = 2 ** M)



Two-way Set Associative Cache

- N-way set associative: N entries for each Cache Index
  - N direct mapped caches operates in parallel (N typically 2 to 4)
- Example: Two-way set associative cache
  - Cache Index selects a “set” from the cache
  - The two tags in the set are compared in parallel
  - Data is selected based on the tag result
- Increasing associativity shrinks index, expands tag

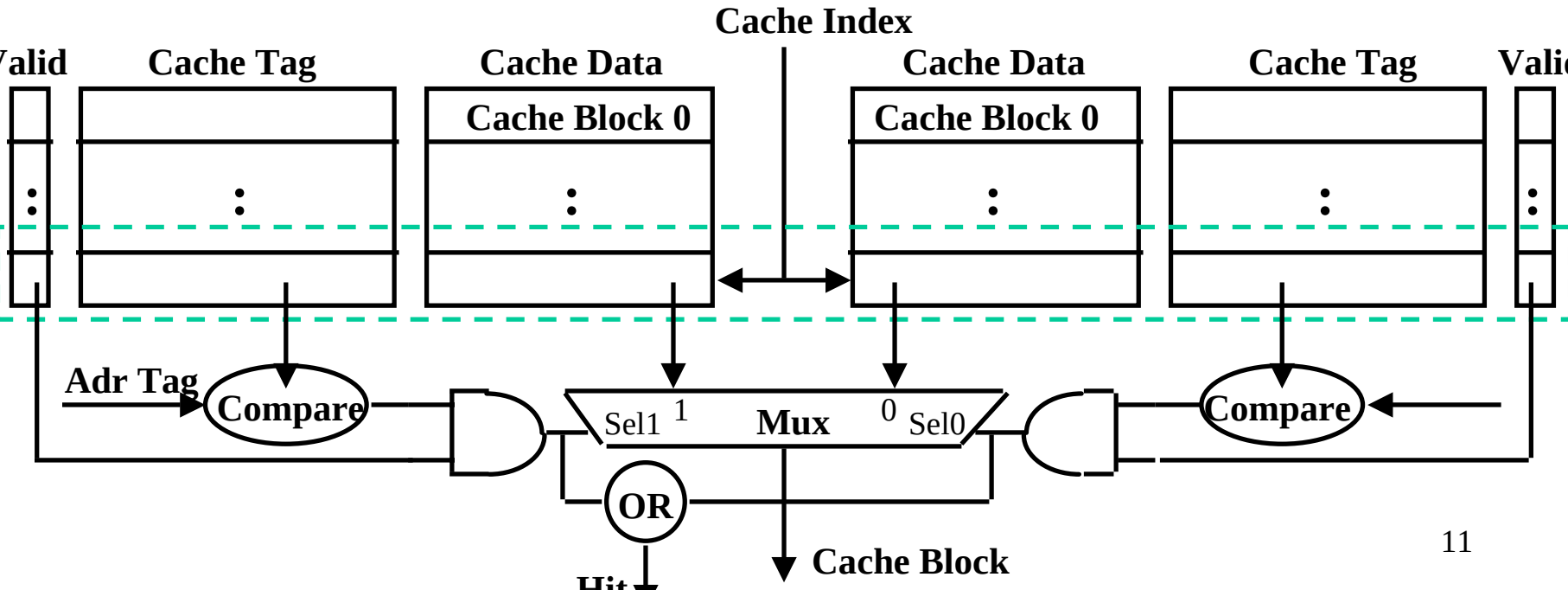


Exercise

- 1KB, Direct-Mapped, 64 Bytes
- 2KB, Direct-Mapped, 32 Bytes
- 2KB. 4-way associative, 32 Bytes

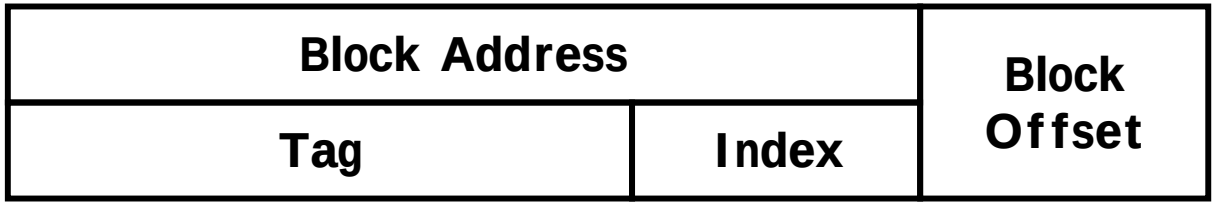
Disadvantage of Set Associative Cache

- N-way Set Associative Cache v. Direct Mapped Cache:
  - N comparators vs. 1
  - Extra MUX delay for the data
  - Data comes AFTER Hit/Miss
- In a direct mapped cache, Cache Block is available BEFORE Hit/Miss:
  - Possible to assume a hit and continue. Recover later if miss.

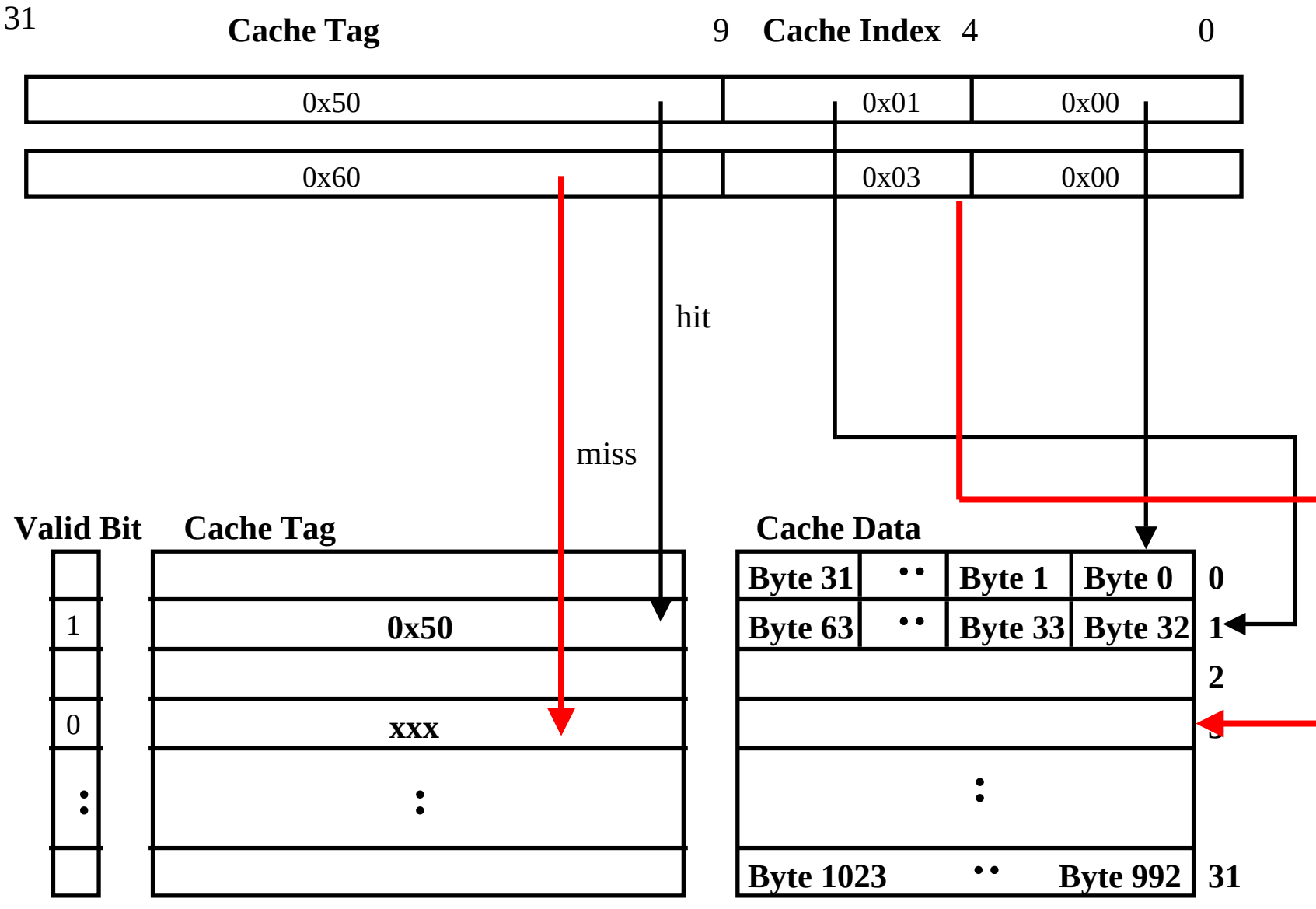


Q2: How is a block found if it is in the upper level?

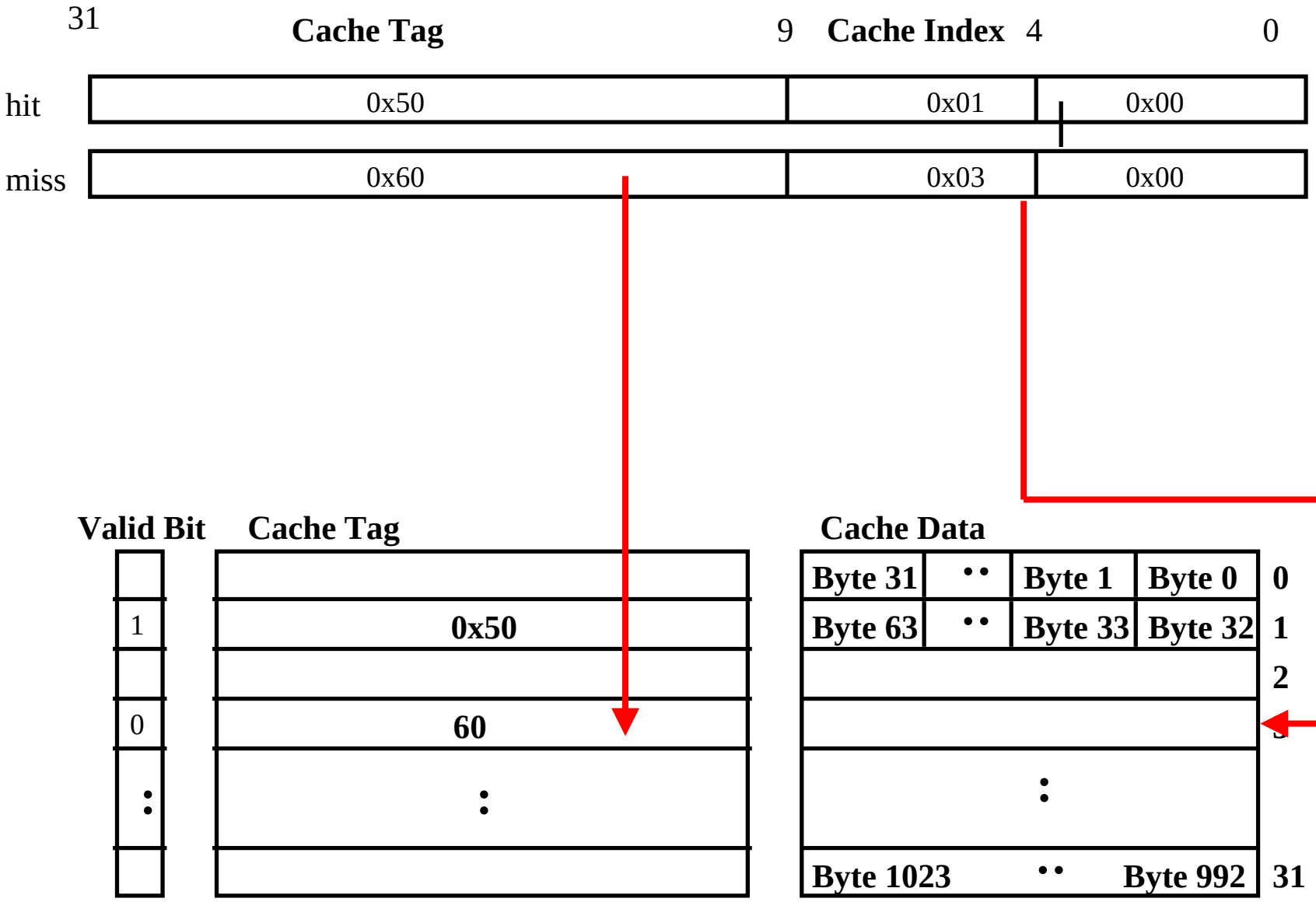
- Tag on each block
  - No need to check index or block offset



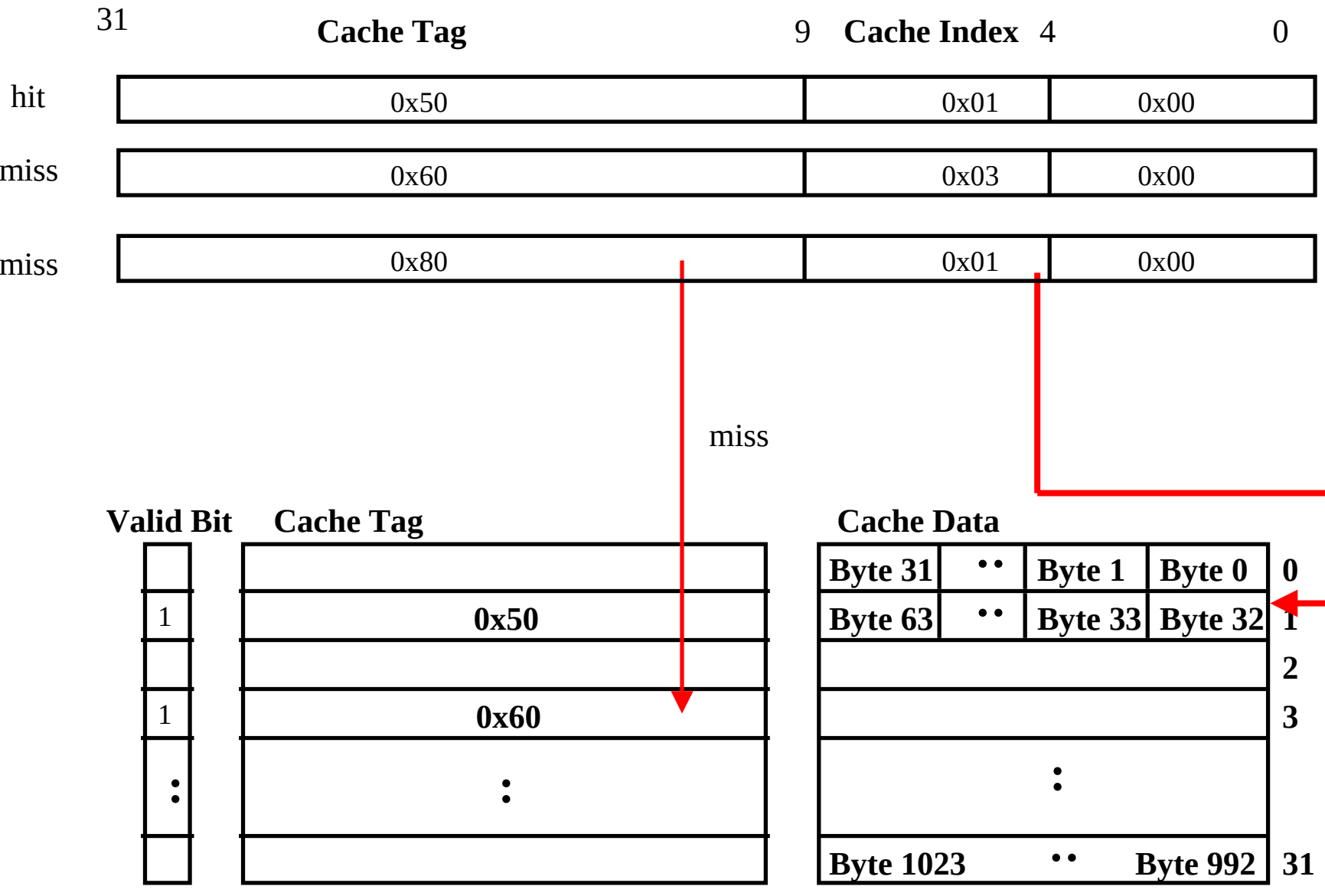
Example: 1 KB Direct Mapped Cache, 32B blocks



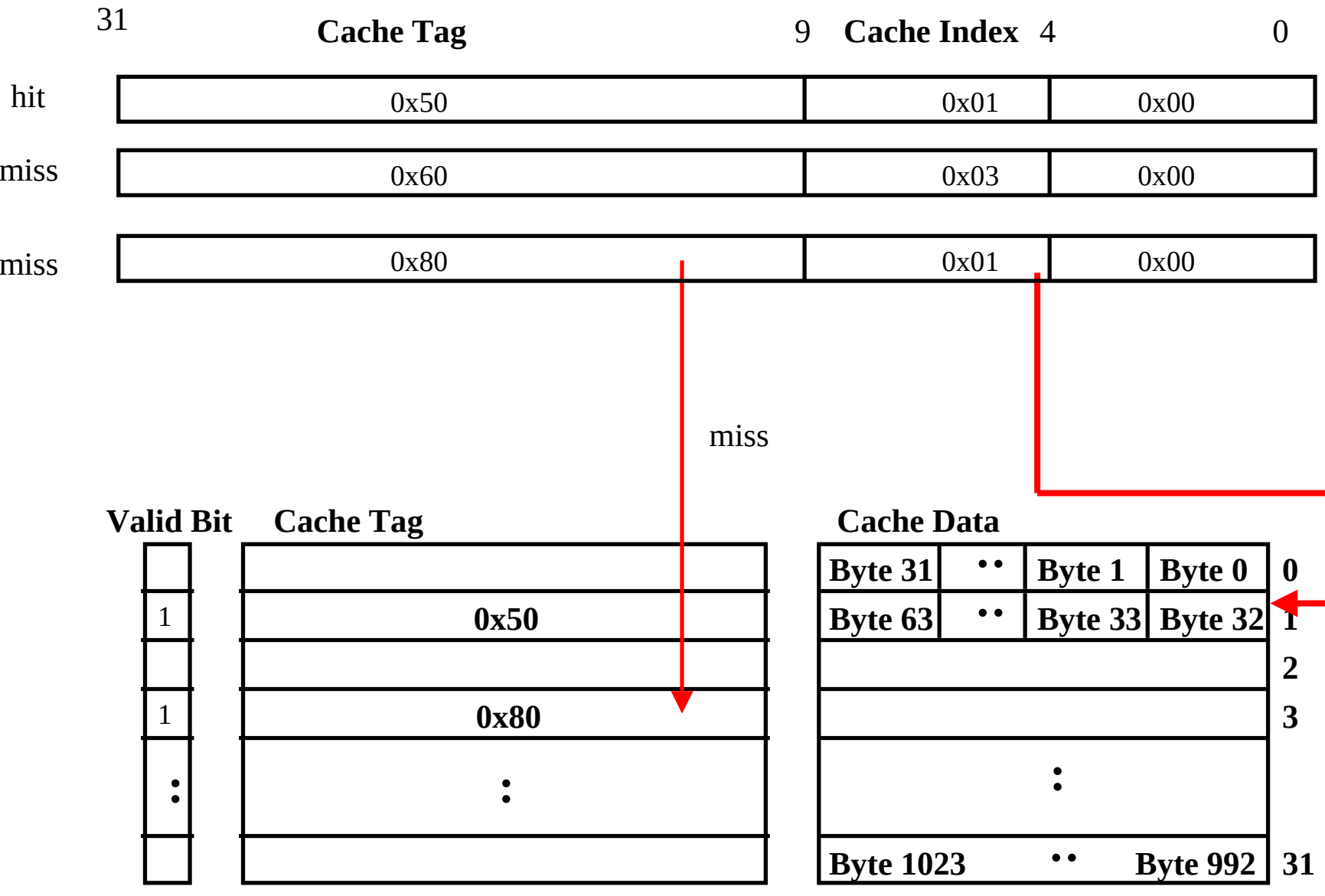
Example: 1 KB Direct Mapped Cache, 32B blocks



Example: 1 KB Direct Mapped Cache, 32B blocks



Example: 1 KB Direct Mapped Cache, 32B blocks



Q3: Which block should be replaced on a miss?

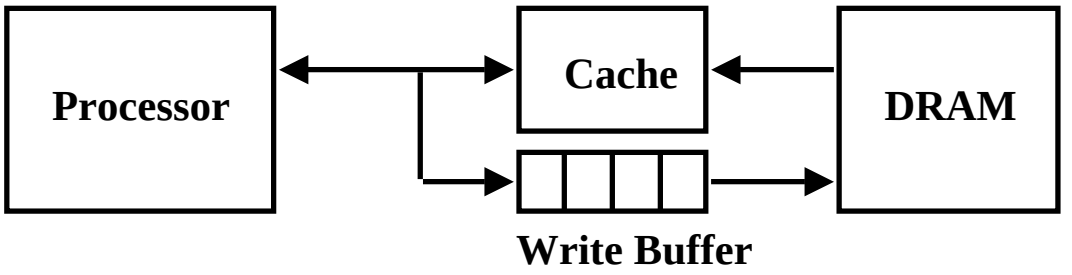
- Easy for Direct Mapped
- Set Associative or Fully Associative:
  - Random
  - LRU (Least Recently Used)
    - Hardware keeps track of the access history
    - Replace the entry that has not been used for the longest time

Assoc:	2-way		4-way		8-way	
Size	LRU	Ran	LRU	Ran	LRU	Ran
16 KB	5.2%	5.7%	4.7%	5.3%	4.4%	5.0%
64 KB	1.9%	2.0%	1.5%	1.7%	1.4%	1.5%
256 KB	1.15%	1.17%	1.13%	1.13%	1.12%	1.12%

Q4: What happens on a write?

- *Write through*—The information is written to both the block in the cache and to the block in the lower-level memory.
- *Write back*—The information is written only to the block in the cache. The modified cache block is written to main memory only when it is replaced.
  - is block clean or dirty?
- Pros and Cons of each?
  - WT:
    - Good: read misses cannot result in writes
    - Bad: write stall
  - WB:
    - no repeated writes to same location
    - Read misses could result in writes
- WT always combined with write buffers so that don't wait for lower level memory

Write Buffer for Write Through



- A Write Buffer is needed between the Cache and Memory
  - Processor: writes data into the cache and the write buffer
  - Memory controller: write contents of the buffer to memory
- Write buffer is just a FIFO:
  - Typical number of entries: 4
  - Works fine if: Store frequency (w.r.t. time) << 1 / DRAM write cycle
- Memory system designer's nightmare:
  - Store frequency (w.r.t. time) > 1 / DRAM write cycle
  - Write buffer saturation

Write Miss Policy

- Write allocate (fetch on write)
  - The block is loaded on a write miss
- No-write allocate (write-around)
  - The block is modified in the lower level and not loaded into the cahce

Cache Performance

CPU time = (CPU execution clock cycles + Memory stall clock cycles) x clock cycle time

Memory stall clock cycles = (Reads x Read miss rate x Read miss penalty + Writes x Write miss rate x Write miss penalty)

Memory stall clock cycles = Memory accesses x Miss rate x Miss penalty

Cache Performance

CPUtime = IC x ($CPI_{execution}$ + Mem accesses per instruction x Miss rate x Miss penalty) x Clock cycle time

Misses per instruction = Memory accesses per instruction x Miss rate

CPUtime = IC x ($CPI_{execution}$ + Misses per instruction x Miss penalty) x Clock cycle time

Example

- Miss penalty 50 clocks
- Miss rate 2%
- Base CPI 2.0
- 1.33 references per instruction
- Compute the CPU time

- CPU time = IC x (2.0 + (1.33 x 0.02 x 50)) x clock
- CPU time = IC x 3.33 x Clock
- So CPI increased from 2.0 to 3.33 with a 2% miss rate

Example 2

- Two caches: both 64KB, 32 byte block, miss penalty 70ns, 1.3 references per instruction, CPI 2.0 with perfect cache
- Direct mapped
  - Cycle time 2ns
  - Miss rate 1.4%
- 2-way associative
  - Cycle time increases by 10%
  - Miss rate 1.0%
- Which is better?
  - Average memory access time?
  - CPU time?

Example 2 continued

- Average memory access time : hit time + (miss rate x miss penalty)
  - 1-way: 2.0 + (0.014 x 70) = 2.98ns
  - 2-way: 2.2 + (0.010 x 70) = 2.90ns
- CPU time = IC x CPI x Cycle
  - 1-way: IC x ((2.0 x 2.0) + (1.3 x 0.014 x 70)) = 5.27 x IC
  - 2-way : IC x (2.0 x 2.2) + (1.3 x 0.010 x 70)) = 5.31 x IC

Which one is better?