

CPS104
Computer Organization and Programming
Lecture 14: The Cache

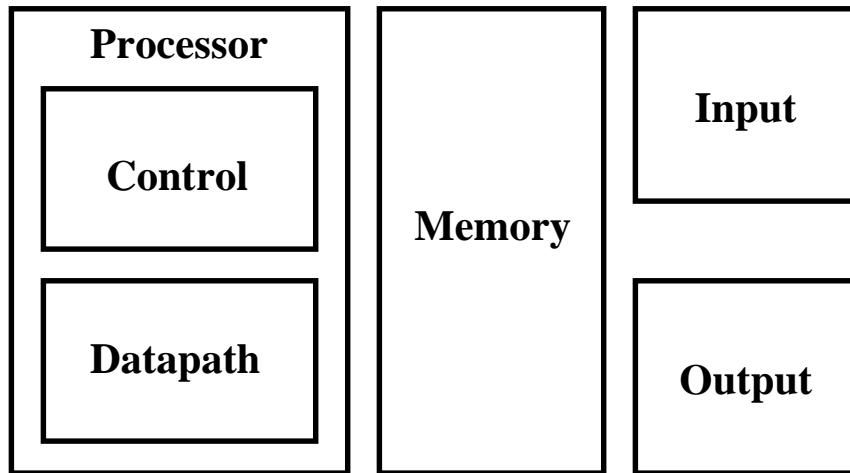
Robert Wagner

Outline of Today's Lecture

- **The Memory Hierarchy**
- **Direct Mapped Cache.**
- **Two-Way Set Associative Cache**
- **Fully Associative cache**
- **Replacement Policies**
- **Write Strategies**

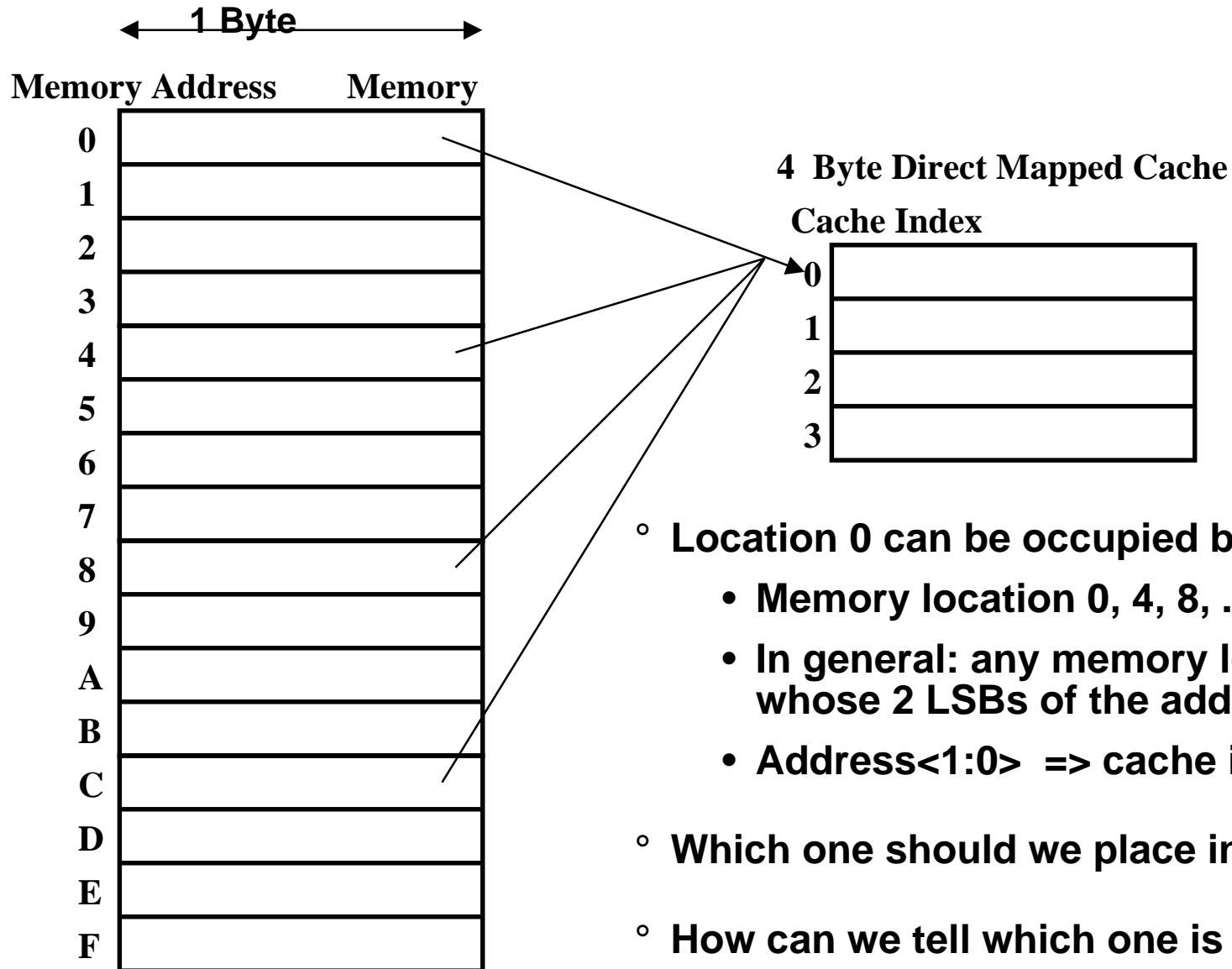
The Big Picture: Where are We Now?

- The Five Classic Components of a Computer



- Today's Topic: Memory System

The Simplest Cache: Direct Mapped Cache



- Location 0 can be occupied by data from:
 - Memory location 0, 4, 8, ... etc.
 - In general: any memory location whose 2 LSBs of the address are 0s
 - Address<1:0> => cache index
- Which one should we place in the cache?
- How can we tell which one is in the cache?

Direct Mapped Cache (Cont.)

For a Cache of 2^M bytes with block size of 2^L bytes

- There are 2^{M-L} cache blocks,
- Lowest L bits of the address are Byte Select bits
- Next $(M - L)$ bits are the Cache-Index.
- The last $(32 - M)$ bits are the Tag bits.



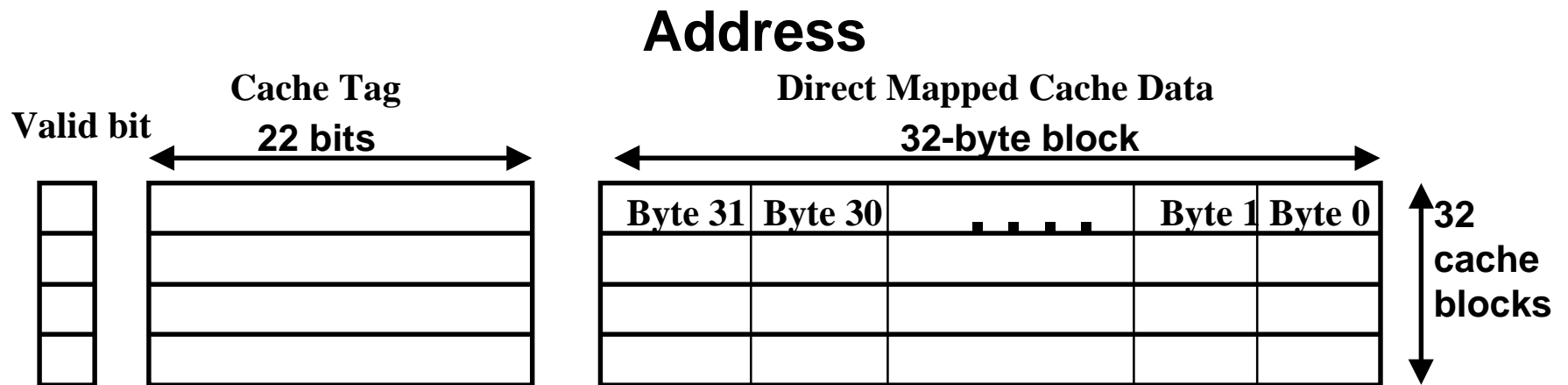
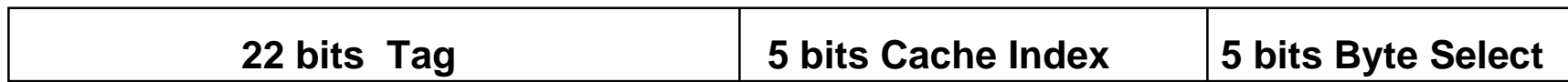
Data Address

Example: 1-KB Cache with 32B blocks:

$$\text{Cache Index} = (\langle \text{Address} \rangle \text{ Mod } (1024)) / 32$$

$$\text{Byte Select} = \langle \text{Address} \rangle \text{ Mod } (32) = \langle \text{Address} \rangle \& 0x1F$$

$$\text{Tag} = \langle \text{Address} \rangle / (1024) = \langle \text{Address} \rangle \gg 10$$

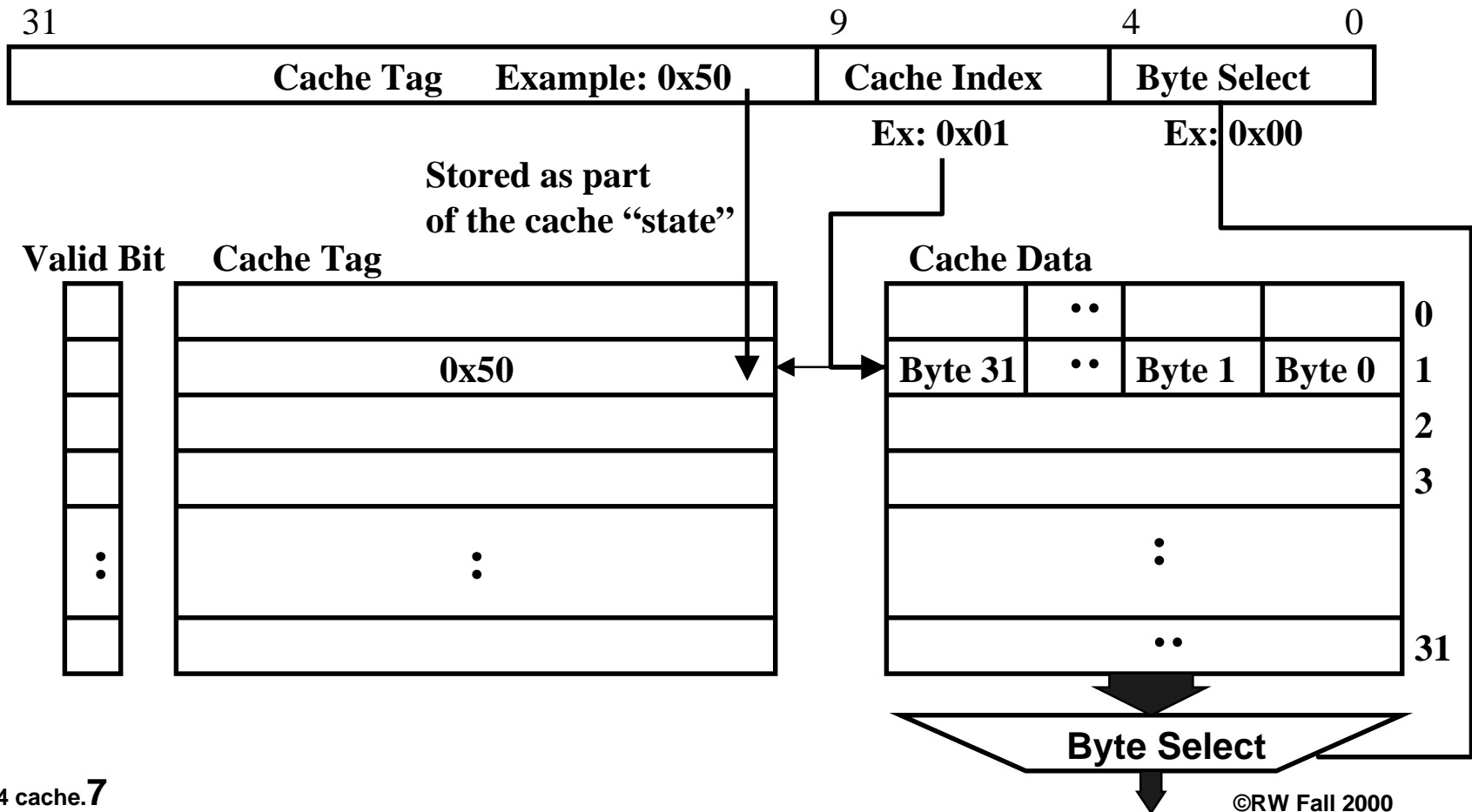


$$1K = 2^{10} = 1024$$

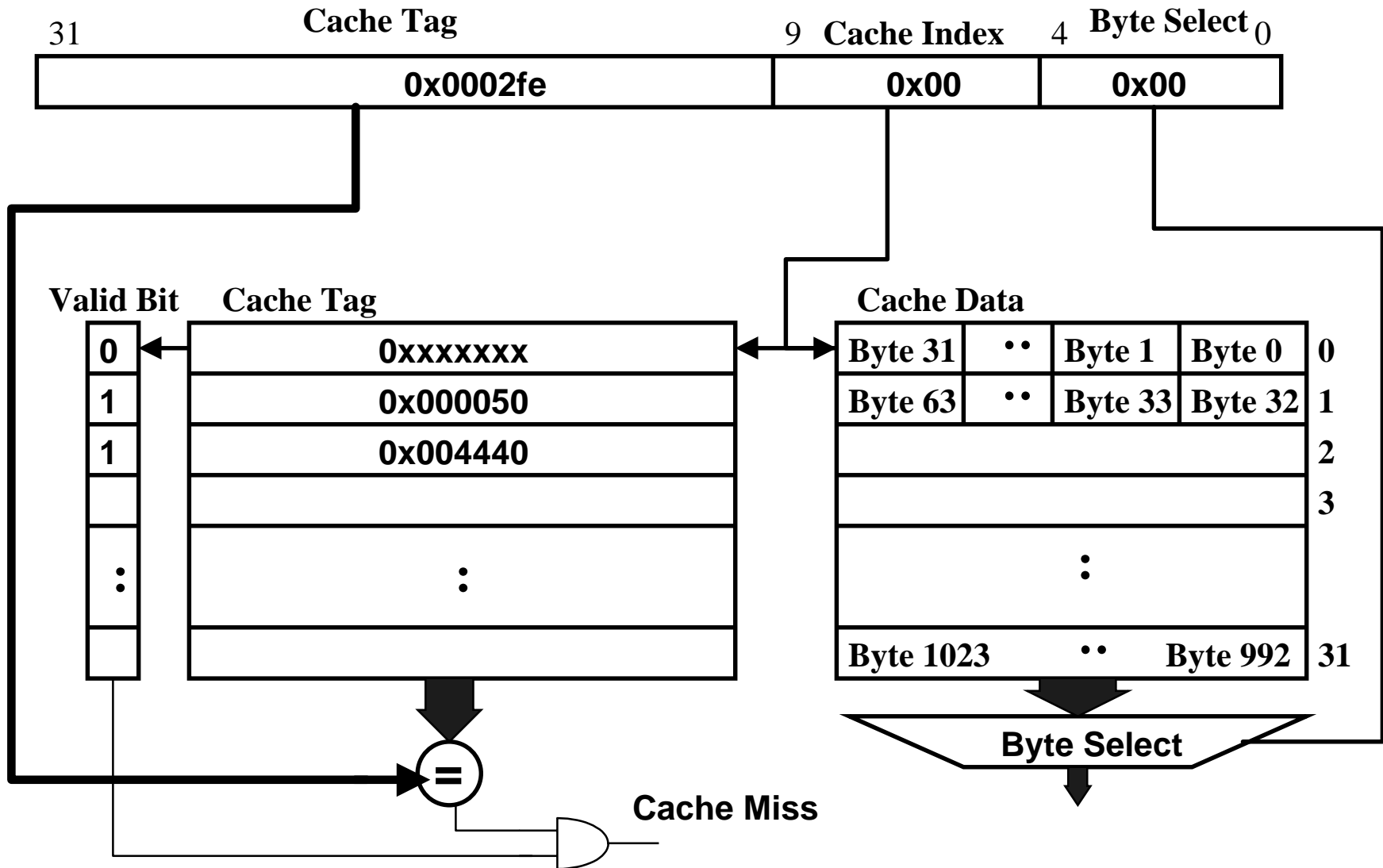
$$2^5 = 32$$

Example: 1KB Direct Mapped Cache with 32B Blocks

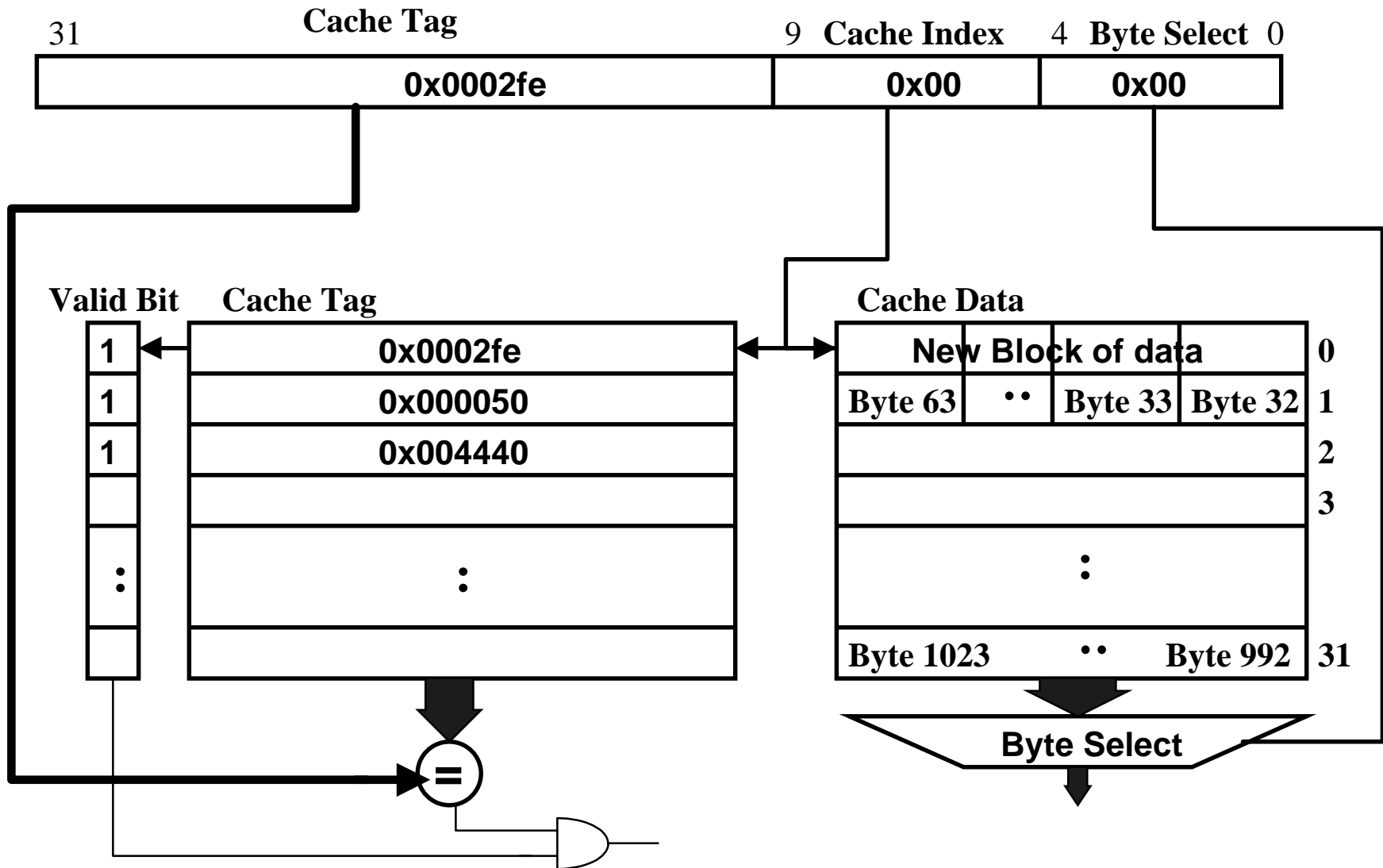
- For a 1024 (2^{10}) byte cache with 32-byte blocks:
 - The uppermost 22 = (32 - 10) address bits are the Cache Tag
 - The lowest 5 address bits are the Byte Select (Block Size = 2^5)
 - The next 5 address bits (bit5 - bit9) are the Cache Index



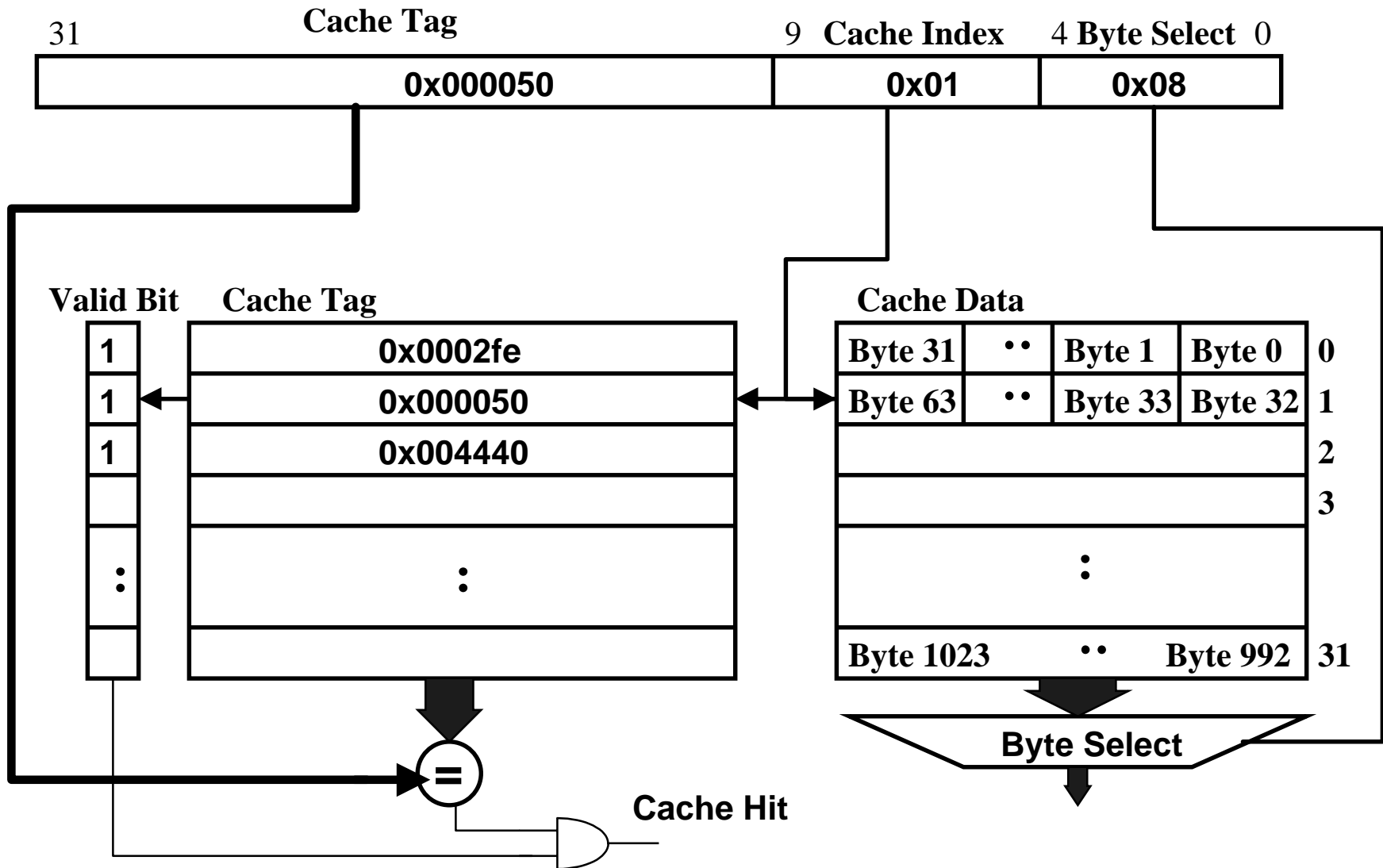
Example: 1K Direct Mapped Cache



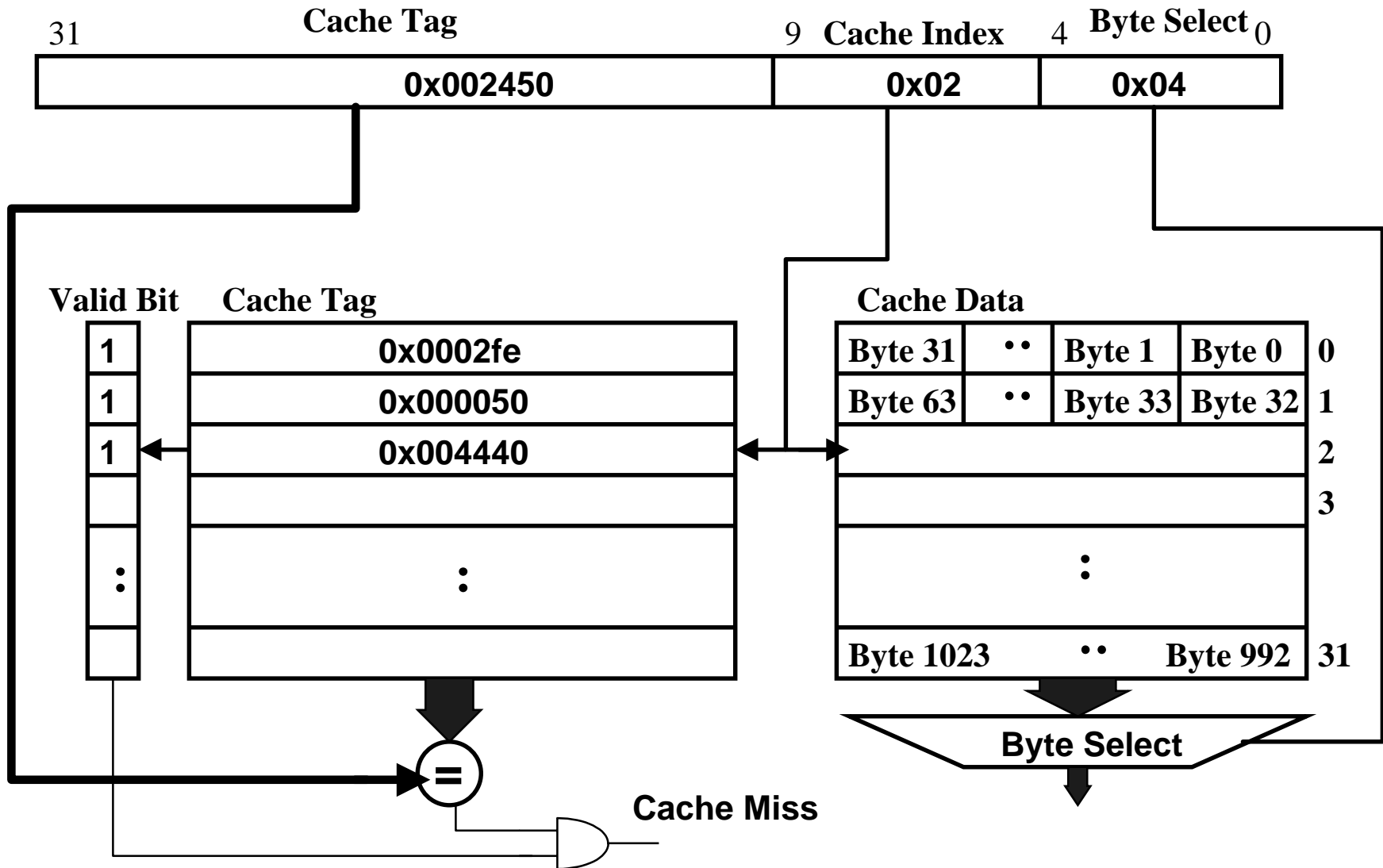
Example: 1K Direct Mapped Cache



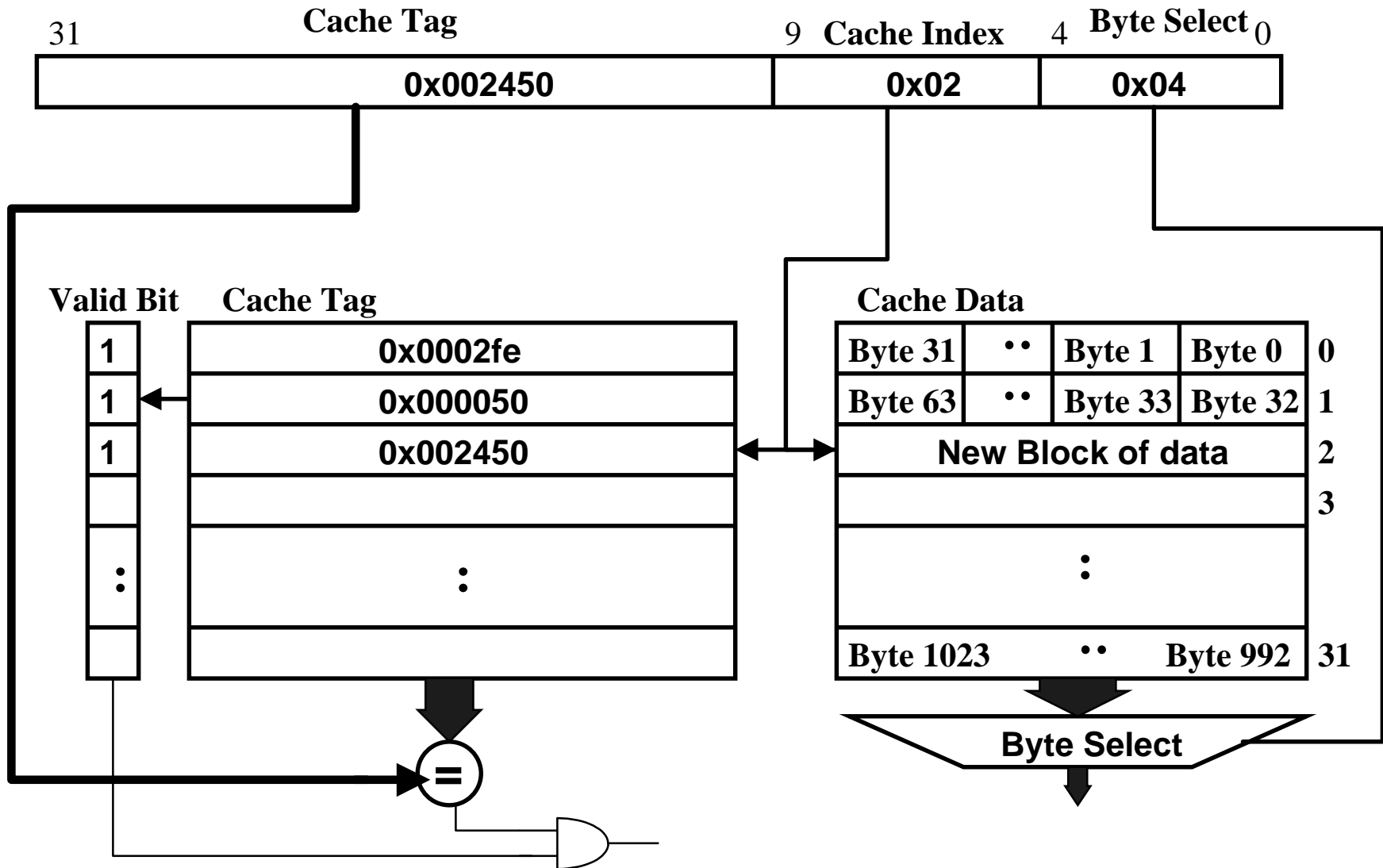
Example: 1K Direct Mapped Cache



Example: 1K Direct Mapped Cache

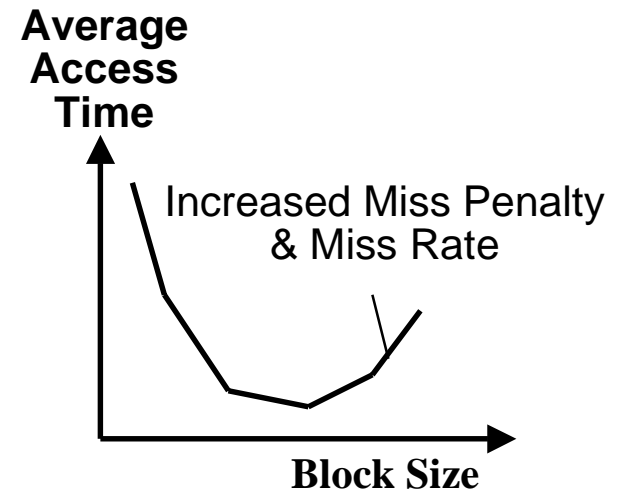
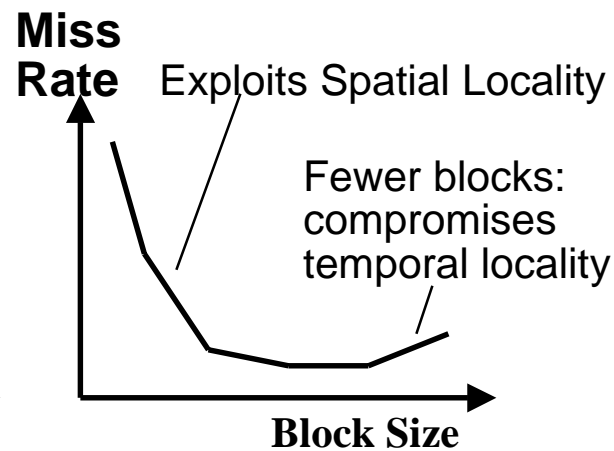
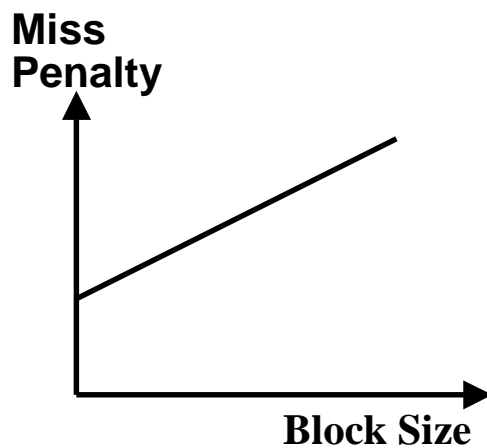


Example: 1K Direct Mapped Cache



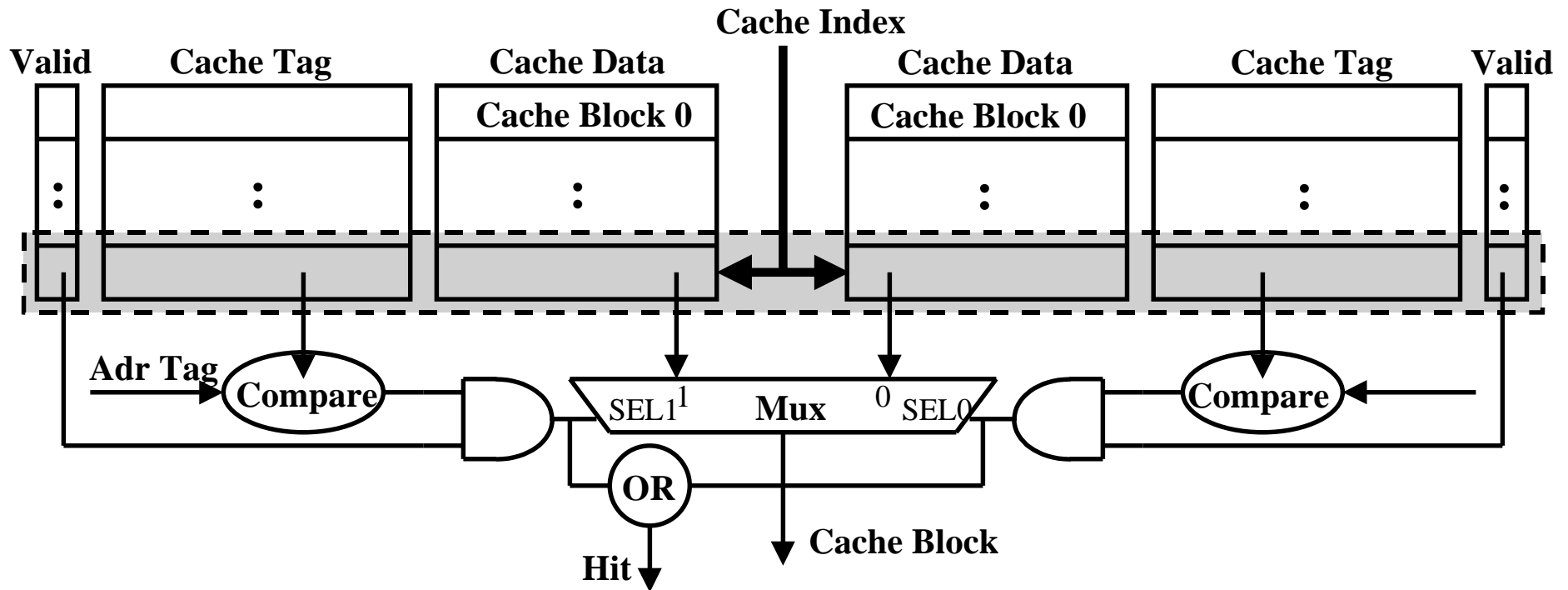
Block Size Tradeoff

- In general, larger block sizes take advantage of spatial locality BUT:
 - Larger block size means larger miss penalty:
 - Takes longer time to fill up the block
 - If block size is too big relative to cache size, miss rate will go up
 - Too few cache blocks
- In general, Average Access Time:
 - $\text{Hit Time} \times (1 - \text{Miss Rate}) + \text{Miss Penalty} \times \text{Miss Rate}$



A N-way Set Associative Cache

- N-way set associative: N entries for each Cache Index
 - N direct mapped caches operating in parallel
- 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

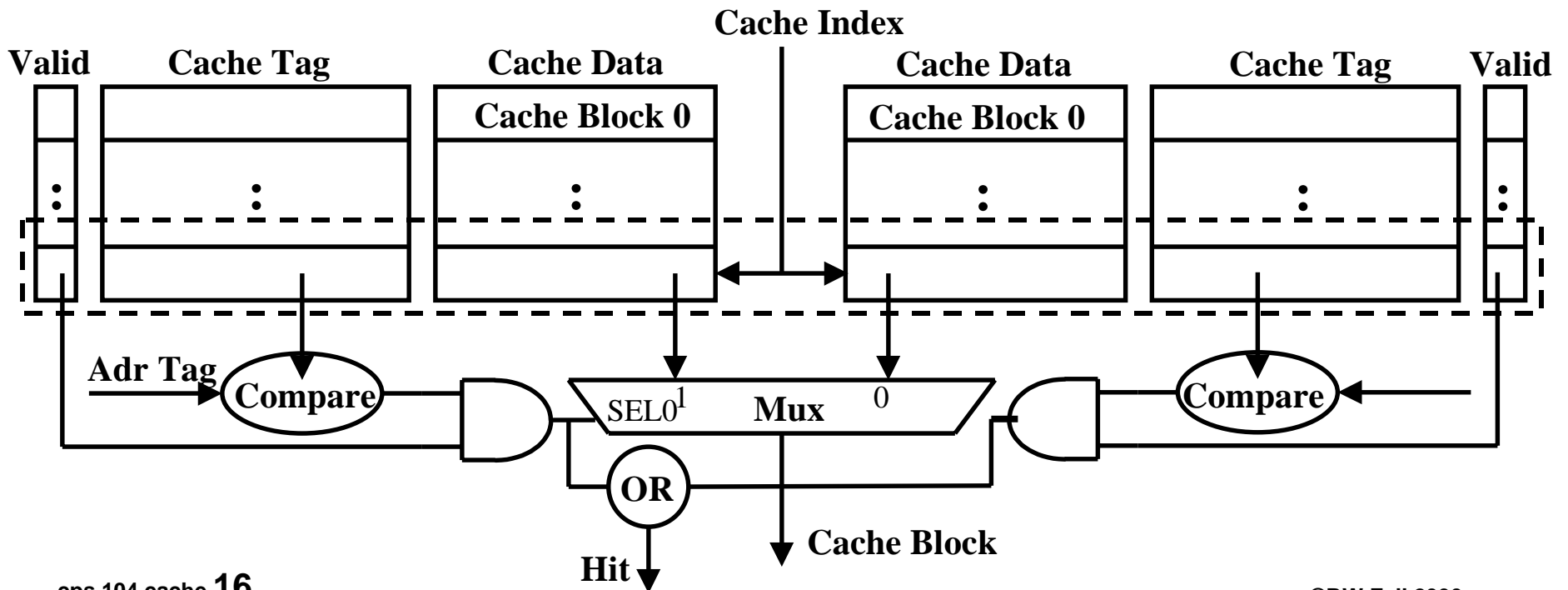


Advantages of Set associative cache

- **Higher Hit rate for the same cache size**
- **Allows 2 independently placed memory blocks to be in the cache at once (E.G.: Array pieces)**
- **Fewer Conflict Misses.**
- **Can have a larger cache but keep the index smaller (So cache index does not overlap virtual page number)**

Disadvantage of Set Associative Cache

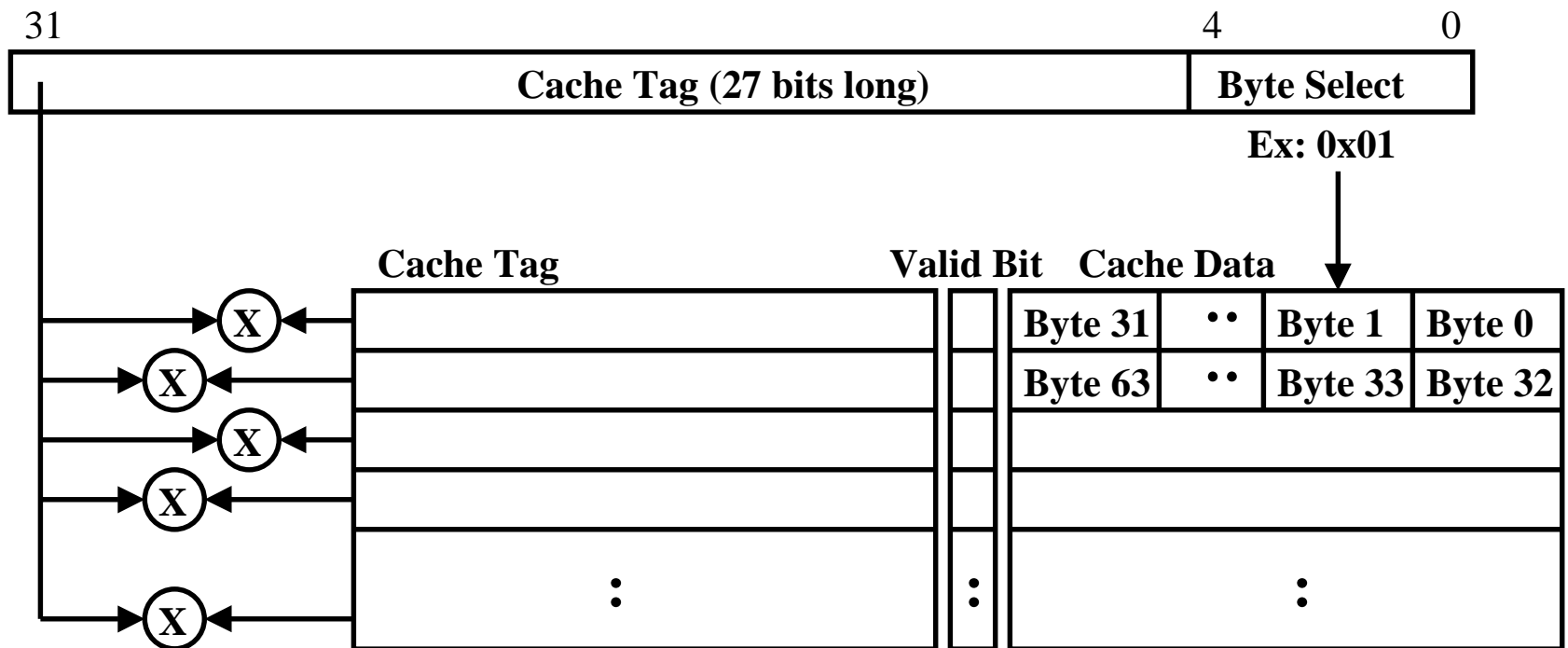
- N-way Set Associative Cache versus Direct Mapped Cache:
 - N comparators vs. 1
 - Extra MUX delay for the data
 - Data comes AFTER Hit/Miss decision and set selection
- In a direct mapped cache, Cache Block is available BEFORE Hit/Miss:
 - Possible to assume a hit and continue. Recover later if miss.



And yet Another Extreme Example: Fully Associative cache

- Fully Associative Cache -- push the set associative idea to its limit!
 - Forget about the Cache Index
 - Compare the Cache Tags of all cache entries in parallel
 - Example: Block Size = 32B blocks, we need N 27-bit comparators

- By definition: Conflict Miss = 0 for a fully associative cache



Sources of Cache Misses

- **Compulsory (cold start or process migration, first reference): first access to a block**
 - **“Cold” fact of life: not a whole lot you can do about it**
- **Conflict (collision):**
 - **Multiple memory locations mapped to the same cache location**
 - **Solution 1: increase cache size**
 - **Solution 2: increase associativity**
- **Capacity:**
 - **Cache cannot contain all blocks accessed by the program between accesses to the same block**
 - **Solution 1: increase cache size**
 - **Solution 2: change the program**
- **Invalidation: other process (e.g., I/O) updates memory**

Sources of Cache Misses

	Direct Mapped	N-way Set Associative	Fully Associative
Cache Size	Big	Medium	Small
Compulsory Miss	Same	Same	Same
Conflict Miss	High	Medium	Zero
Capacity Miss	Low(er)	Medium	High
Invalidation Miss	Same	Same	Same

Note:

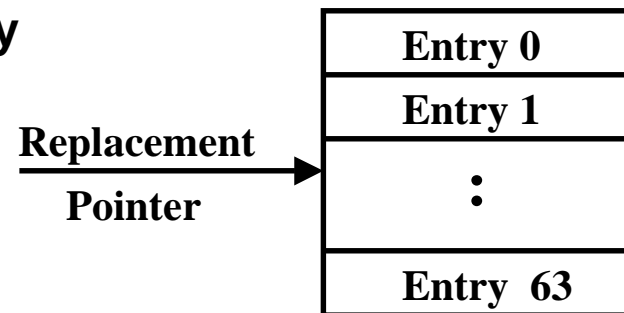
If you are going to run “billions” of instruction, Compulsory Misses are insignificant.

The Need to Make a Decision!

- **Direct Mapped Cache:**
 - Each memory location can only map to 1 cache location
 - No need to make any decision :-)
 - Current item replaced the previous item in that cache location
- **N-way Set Associative Cache:**
 - Each memory location has a choice of N cache locations
- **Fully Associative Cache:**
 - Each memory location can be placed in ANY cache location
- **Cache miss in an N-way Set Associative or Fully Associative Cache:**
 - Bring in new block from memory
 - Throw out a cache block to make room for the new block
 - We need to make a decision on which block to throw out!

Cache Block Replacement Policy Choices

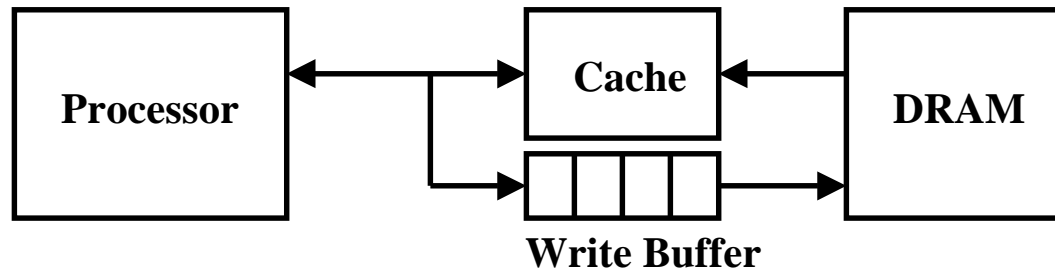
- **Random Replacement:**
 - Hardware randomly selects a cache item and throws it out
- **Least Recently Used:**
 - Hardware keeps track of the access history
 - Replace the entry that has not been used for the longest time.
 - For two way set associative cache one needs one bit for LRU replacement.
- **Example of a Simple “Pseudo” Least Recently Used Implementation:**
 - Assume 64 Fully Associative Entries
 - Hardware replacement pointer points to one cache entry
 - Whenever an access is made to the entry the pointer points to:
 - Move the pointer to the next entry
 - Otherwise: do not move the pointer
 - NOT true LRU -- Why?



Cache Write Policy: Write Through versus Write Back

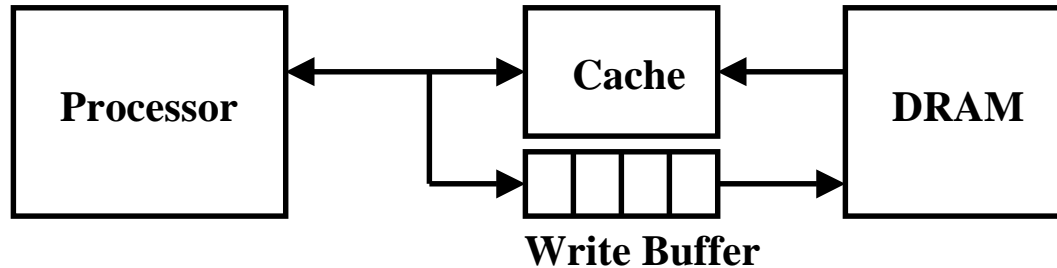
- Cache read is much easier to handle than cache write:
 - Instruction cache is much easier to design than data cache
- Cache write:
 - How do we keep data in the cache and memory consistent?
- Two options (decision time again :-)
 - **Write Back:** write to cache only. Write the cache block to memory when that cache block is being replaced on a cache miss.
 - Need a “dirty bit” for each cache block
 - Greatly reduces the memory bandwidth requirement
 - Control can be complex
 - **Write Through:** write to cache and memory at the same time.
 - What!!! How can this be? Isn't memory too slow for this?

Write Buffer for Write Through

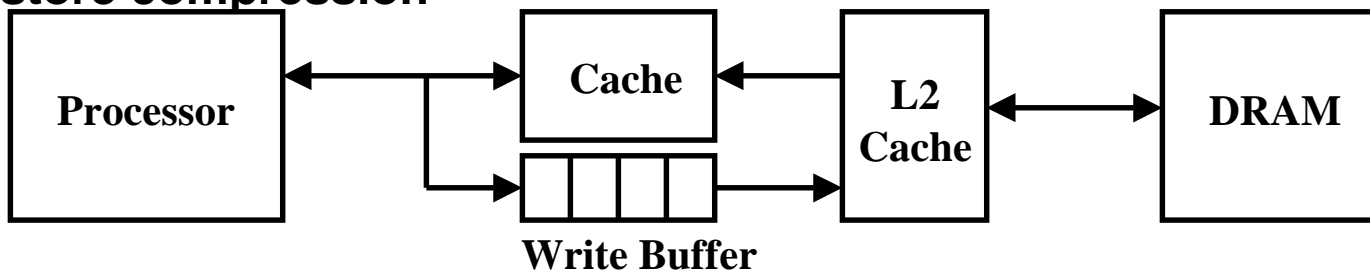


- **A Write Buffer is needed between the Cache and Memory**
 - **Processor:** writes data into the cache and the write buffer
 - **Memory controller:** writes contents of the buffer to memory
- **Write buffer is just a FIFO:**
 - **Typical number of entries: 4**
 - **Works fine if: Store frequency (w.r.t.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

Write Buffer Saturation

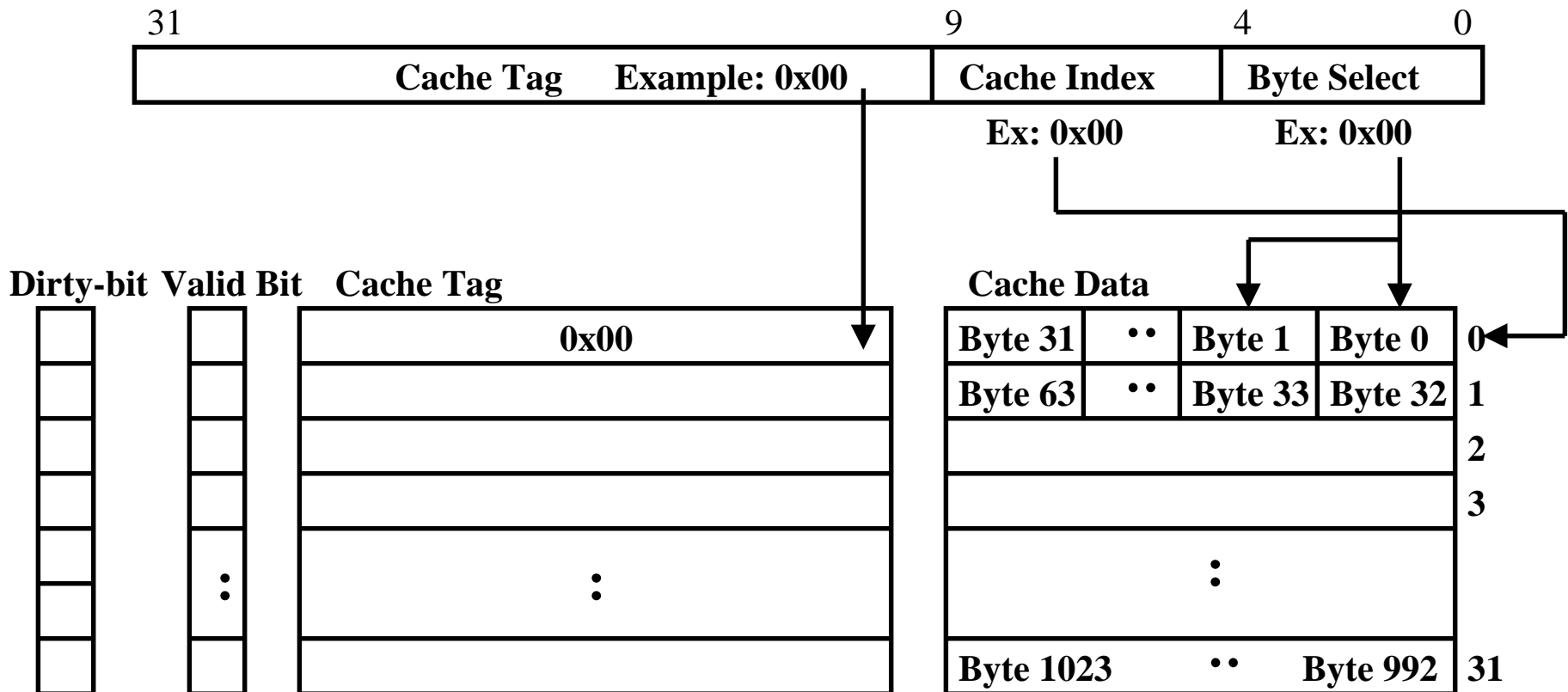


- Store frequency (w.r.t. time) $> 1 / \text{DRAM write cycle}$
 - If this condition exists for a long period of time (CPU cycle time too quick and/or too many store instructions in a row):
 - Store buffer will overflow no matter how big you make it
 - The CPU Cycle Time will slow to DRAM Write Cycle Time
- Solution for write buffer saturation:
 - Use a write back cache
 - Install a second level (L2) cache:
 - store compression



Write Allocate versus Not Allocate

- Assume: a 16-bit write to memory location 0x0 causes a miss
 - Do we read the block into the cache?
 - Yes: Write Allocate
 - No: Write Not Allocate



Four Questions for Memory Hierarchy Designers

- 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)

Summary:

- **The Principle of Locality:**
 - **Program accesses a relatively small portion of the address space at any instant of time.**
 - **Temporal Locality: Locality in Time**
 - **Spatial Locality: Locality in Space**
- **Three Major Categories of Cache Misses:**
 - **Compulsory Misses: sad fact of life. Example: cold start misses.**
 - **Conflict Misses: increase cache size and/or associativity.
Nightmare Scenario: ping pong effect!**
 - **Capacity Misses: increase cache size**
- **Write Policy:**
 - **Write Through: need a write buffer. Nightmare: WB saturation**
 - **Write Back: control can be complex**