

CPS104 Computer Organization

Lecture 20

Cache Memory

November 4, 2009

Gershon Kedem

Administratrivia

- Homework 6 & 7 are posted **Due: November 9 & 18.**
- Reading: Chapter 5.

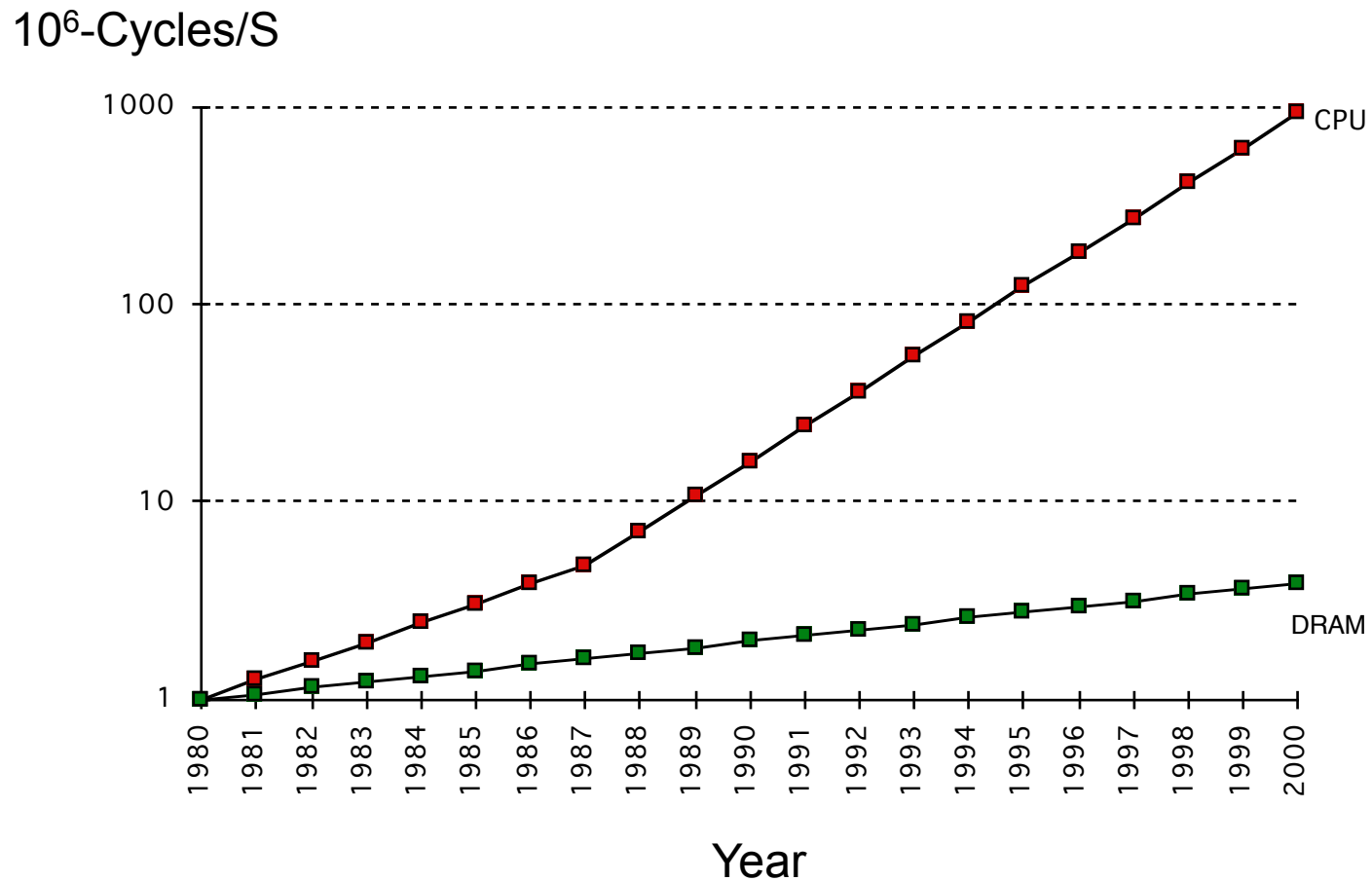
Summary of Memory Technology

- DRAM is **slow** but **cheap** and **dense**:
 - ◆ Good choice for presenting the user with a BIG memory system
 - ◆ Uses one transistor, must be refreshed.
- SRAM is **fast** but **expensive** and **not very dense**:
 - ◆ Good choice for providing the user FAST access time.
 - ◆ Uses six transistors, holds state as long as power is supplied.
- **GOAL:**
 - ◆ Present the user with large amounts of memory using the cheapest technology.
 - ◆ Provide access at the speed offered by the fastest technology.

Next: Caches

Who Cares about Memory Hierarchy?

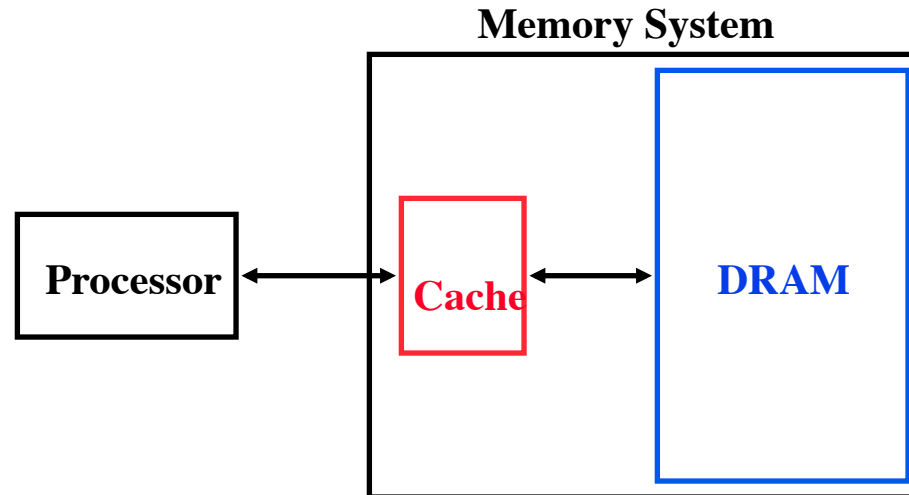
The CPU Memory Gap



The CPU-Memory Speed Gap

- ✱ To illustrate the problem consider “typical” delays, measured in ns.
 - ◆ Clock Period: 0.3ns
 - ◆ Instructions : 1-4 instructions/clock (4-way super-scalar)
 - ◆ On-chip small-fast SRAM (Level-1 cache): 0.3-0.6ns (1-2 clocks).
 - ◆ On-chip large-fast SRAM (Level-2 cache) 4-6ns (12-18 clocks).
 - ◆ Off-chip large-fast SRAM (Level-3 cache) 7-14ns (20-40 clocks)
 - ◆ Off chip large-slow DRAM (Main memory) 90-120ns (270-360 clocks)
- ✱ **Question:** How often does the computer access memory?

The Motivation for Caches



- **Motivation:**
 - ◆ Large memories (DRAM) are **slow**
 - ◆ Small memories (SRAM) are **fast**
- Make the **average access time** shorter by:
 - ◆ Servicing most accesses from a small, fast memory.
- Reduce the **bandwidth** required of the large memory.

Levels of the Memory Hierarchy

Capacity
Access Time
Cost

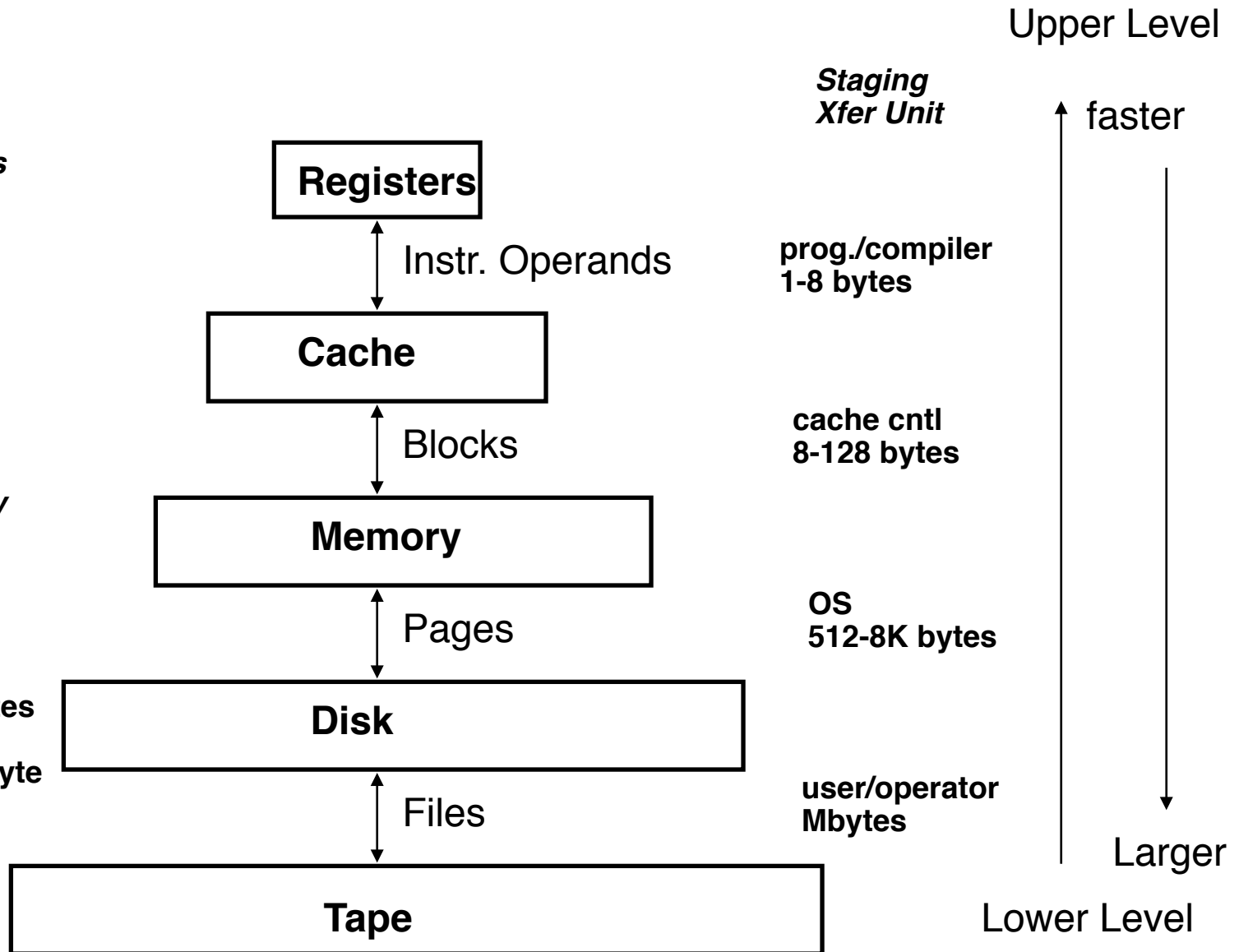
CPU Registers
100s Bytes
<10s ns

Cache
K Bytes
10-100 ns
~\$.0005/bit

Main Memory
M Bytes
100-200ns
~\$10⁻⁷/byte

Disk
10-100 G Bytes
ms
~\$10⁻⁸ - 10⁻¹⁰/byte

Tape
infinite
sec-min
\$10⁻¹¹



The Principle of Locality



- **The Principle of Locality:**

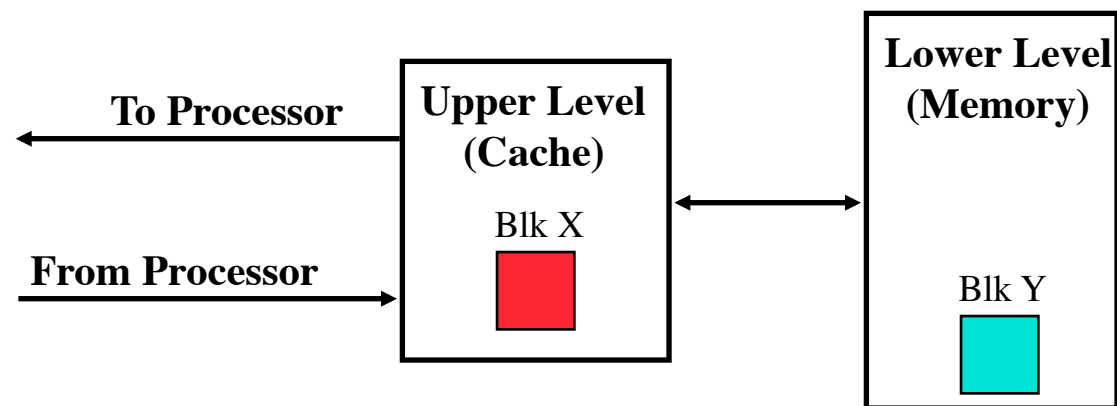
- ◆ Program access a relatively small portion of the address space at any instant of time.
- ◆ Example: **90% of time in 10% of the code**

- **Two Different Types of Locality:**

- ◆ **Temporal Locality** (Locality in Time): If an item is referenced, it will tend to be referenced again soon.
- ◆ **Spatial Locality** (Locality in Space): If an item is referenced, items whose addresses are close by tend to be referenced soon.

Memory Hierarchy: Principles of Operation

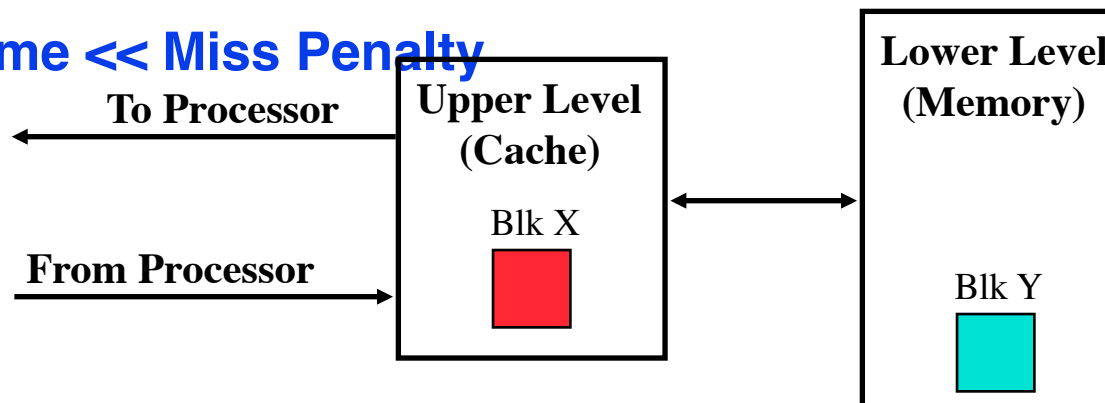
- At any given time, data is copied between only 2 adjacent levels:
 - ◆ Upper Level (Cache) : the one closer to the processor
 - Smaller, faster, and uses more expensive technology
 - ◆ Lower Level (Memory): the one further away from the processor
 - Bigger, slower, and uses less expensive technology
- **Block:**
 - ◆ The minimum unit of information that can either be present or not present in the two level hierarchy



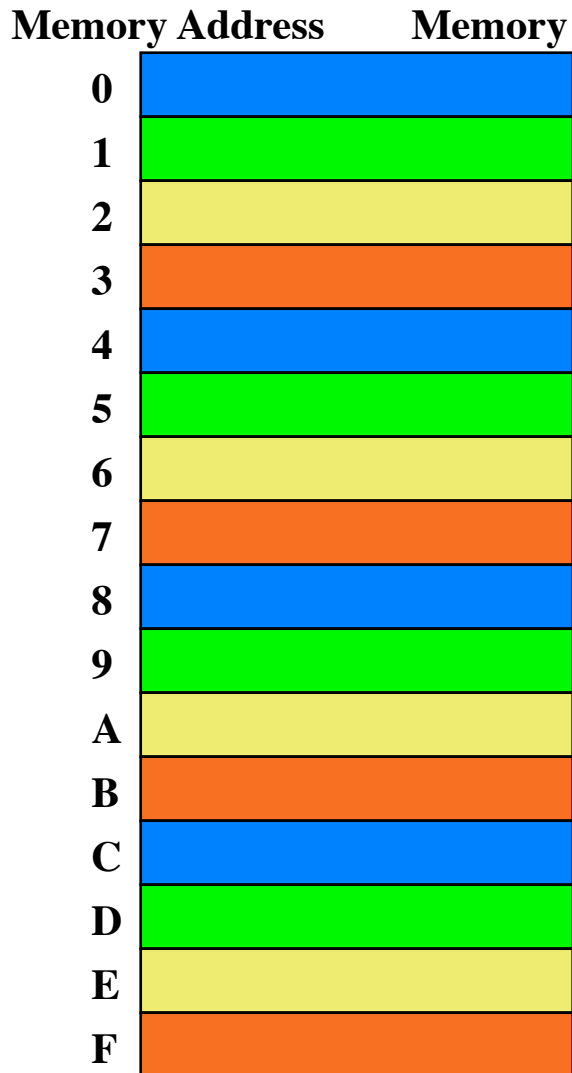
Memory Hierarchy: Terminology

- **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** = $1 - (\text{Hit Rate})$
 - ♦ **Miss Penalty** = Time to replace a block in the upper level +
Time to deliver the block the processor

- **Hit Time** \ll **Miss Penalty**



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 00
 - ♦ $\text{Address}\langle 1:0 \rangle \Rightarrow \text{cache index}$
- 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 **Block-Offset** 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:

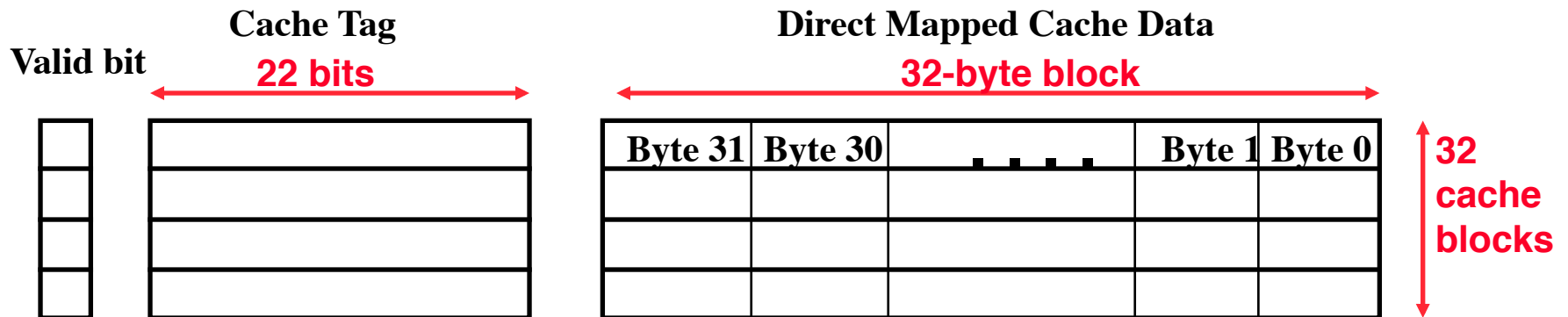
$$\text{Cache Index} = (\langle \text{Address} \rangle \text{ Mod } (1024)) / 32$$

$$\text{Block-Offset} = \langle \text{Address} \rangle \text{ Mod } (32)$$

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



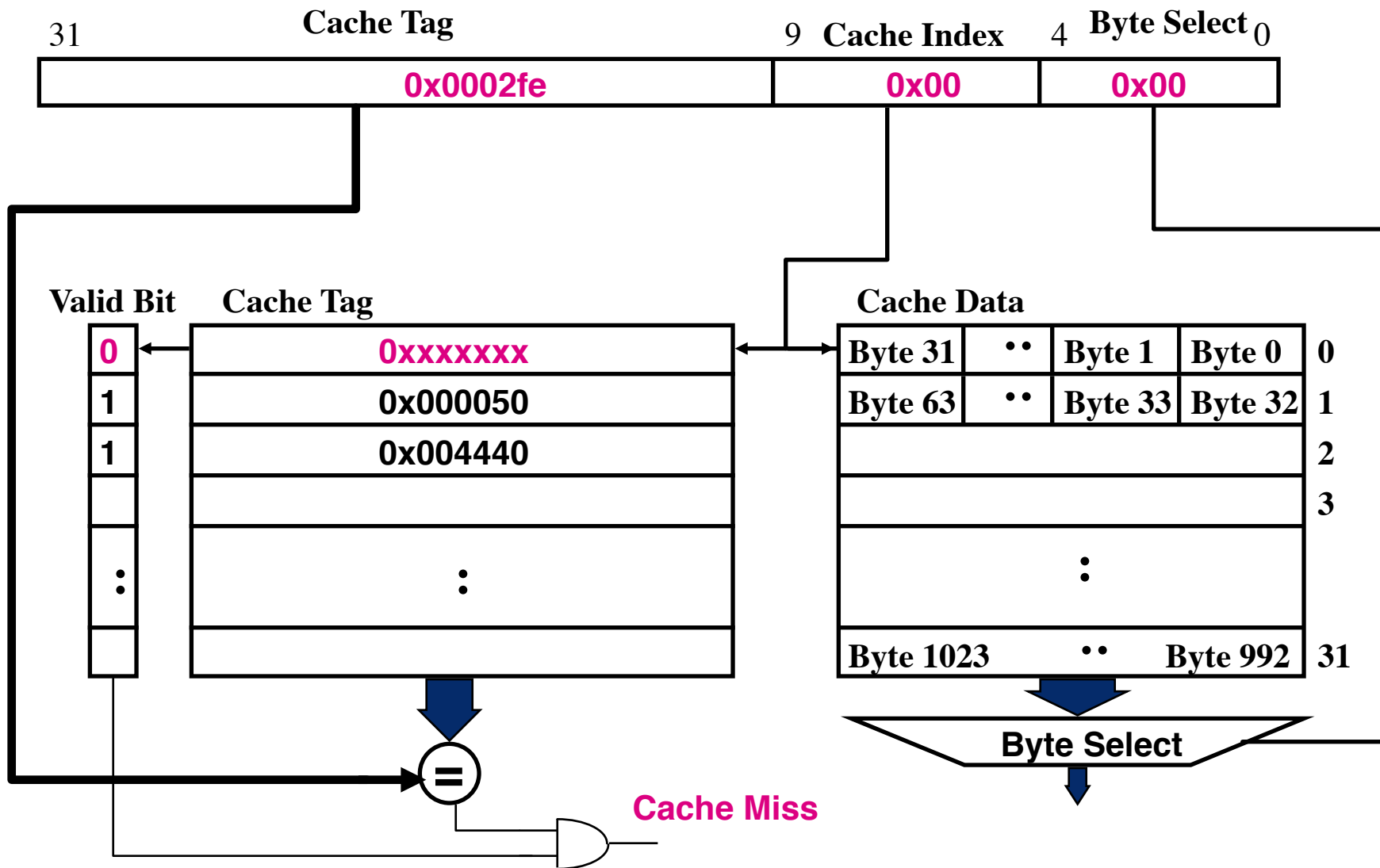
Address



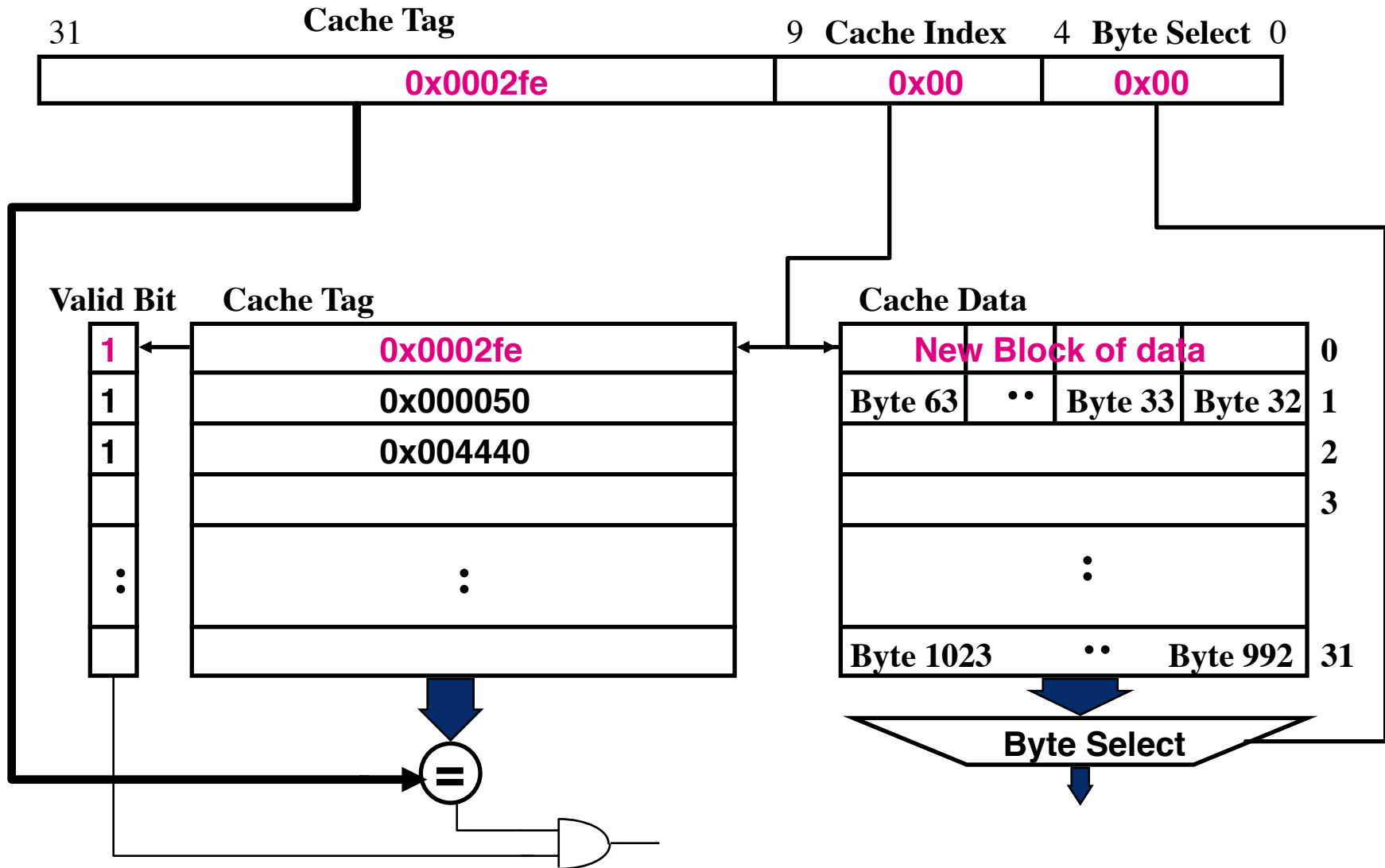
$$1\text{K} = 2^{10} = 1024$$

$$2^5 = 32$$

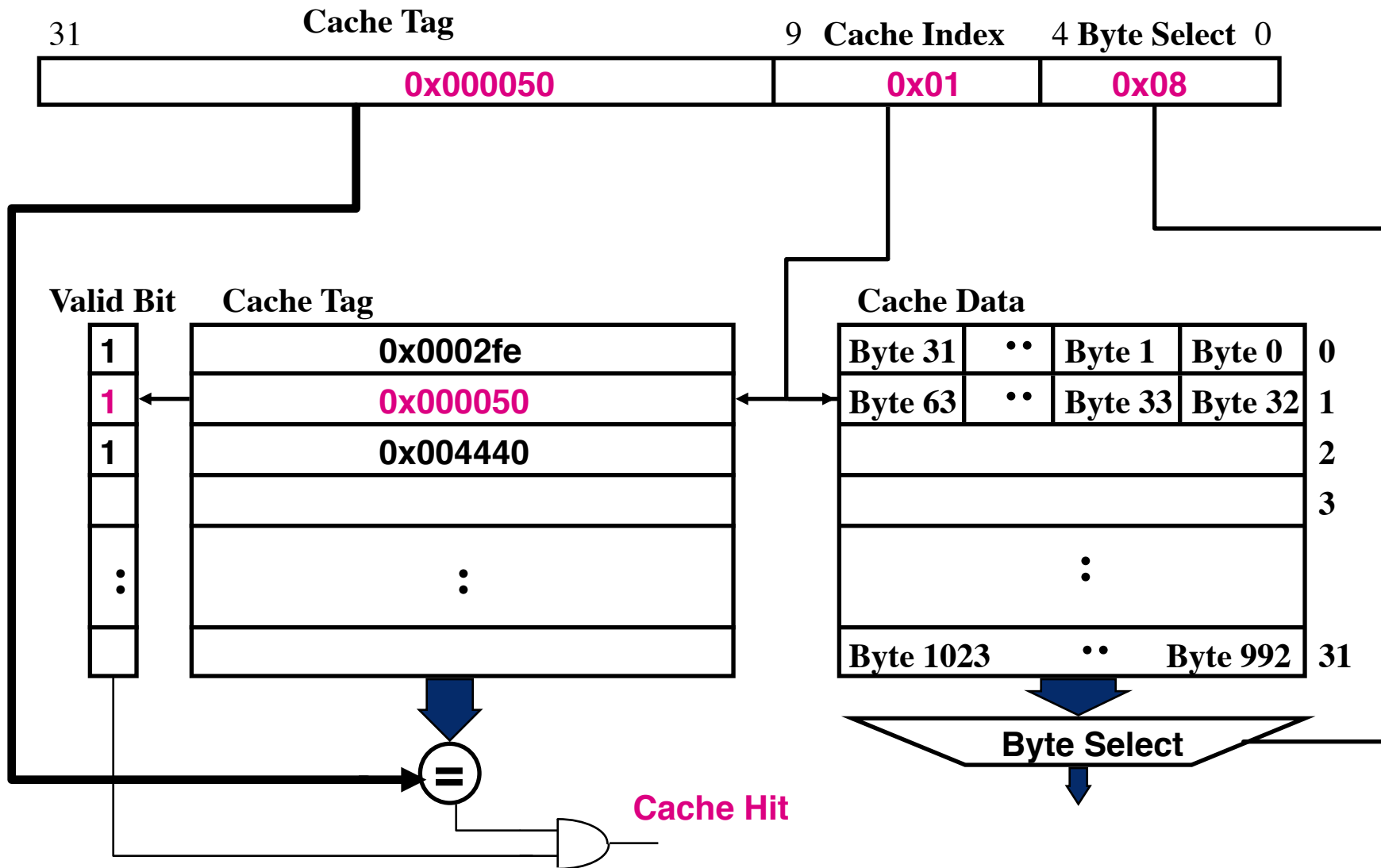
Example: 1K Direct Mapped Cache



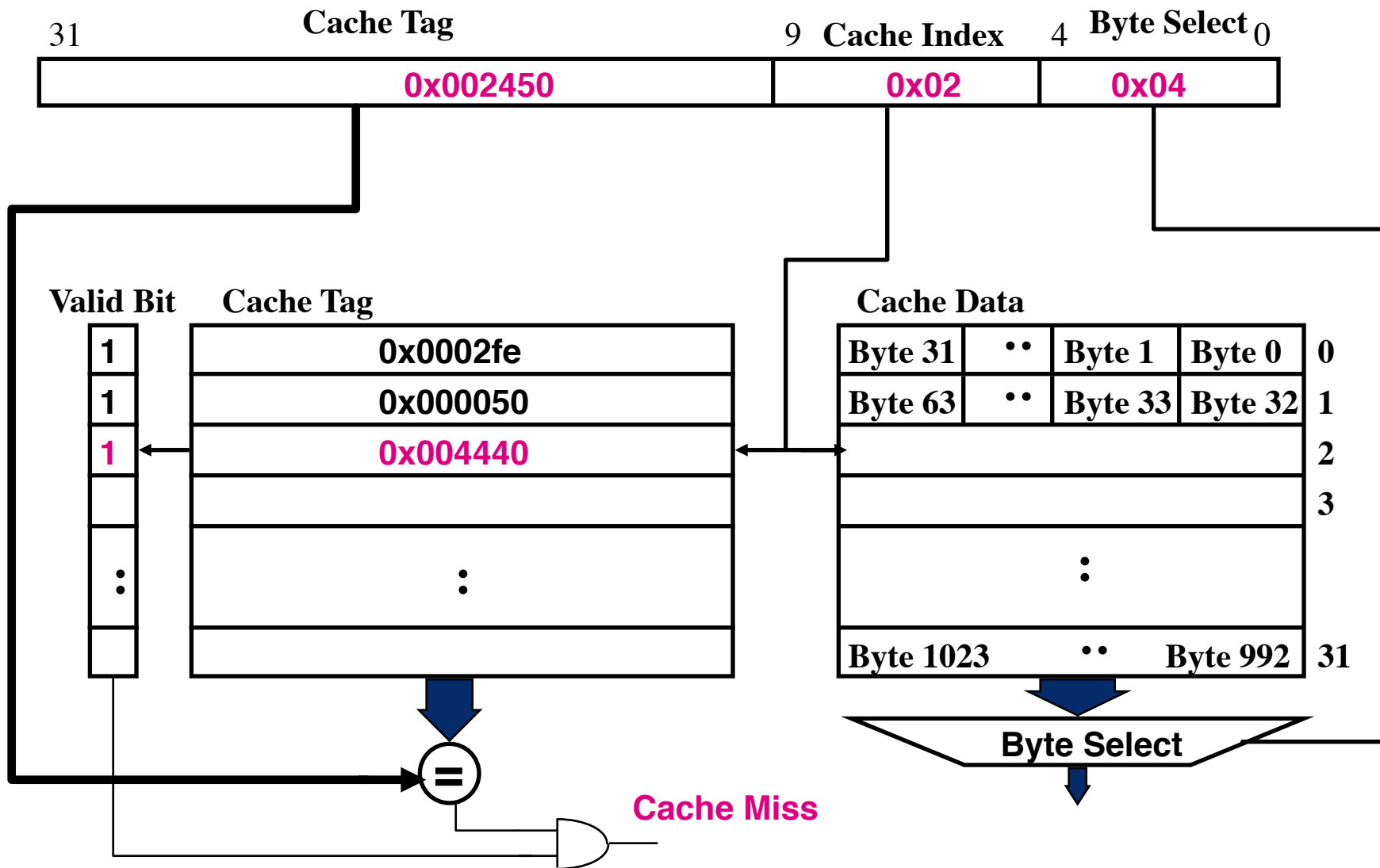
Example: 1K Direct Mapped Cache



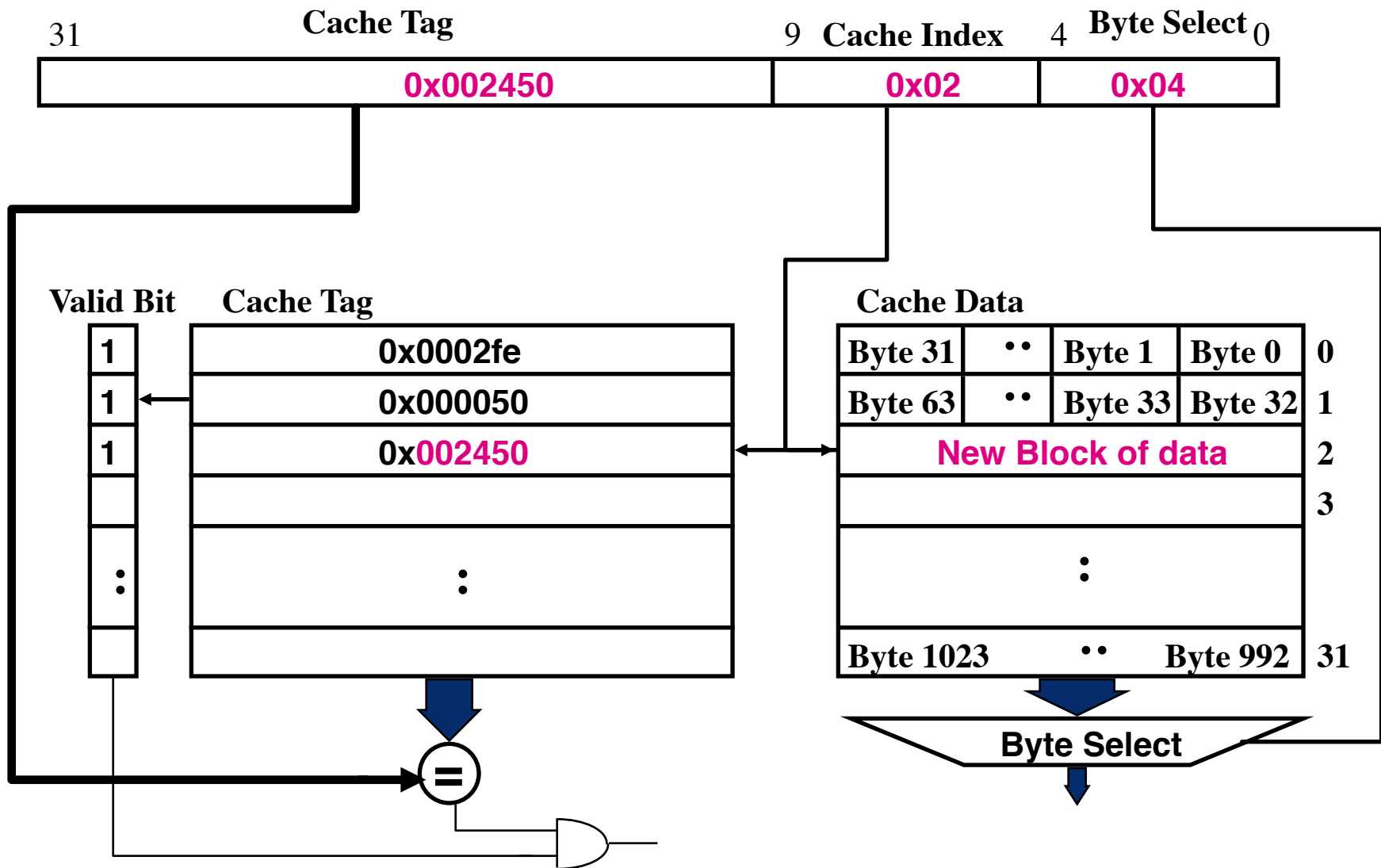
Example: 1K Direct Mapped Cache



Example: 1K Direct Mapped Cache



Example: 1K Direct Mapped Cache

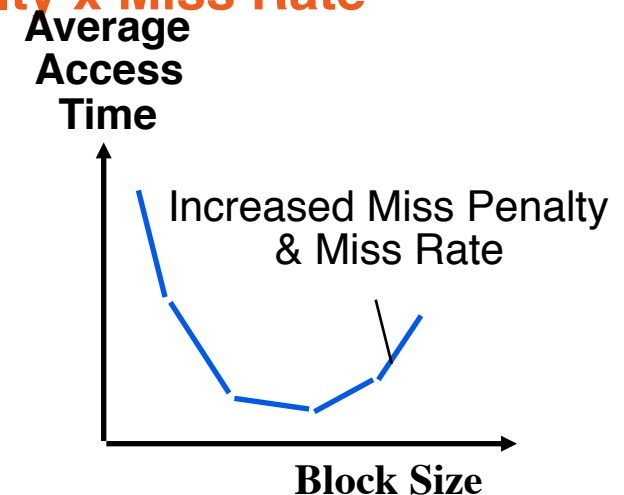
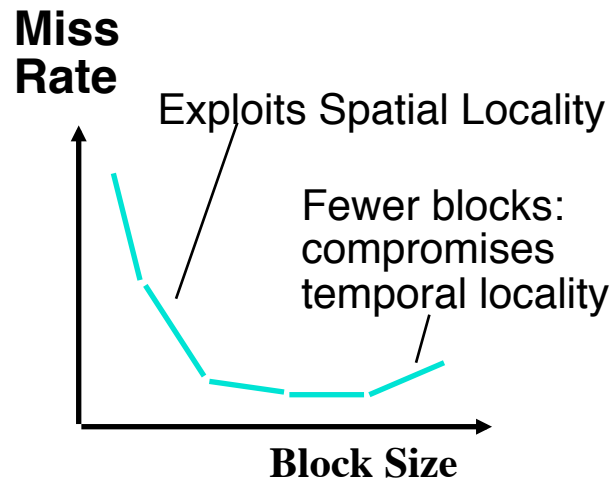
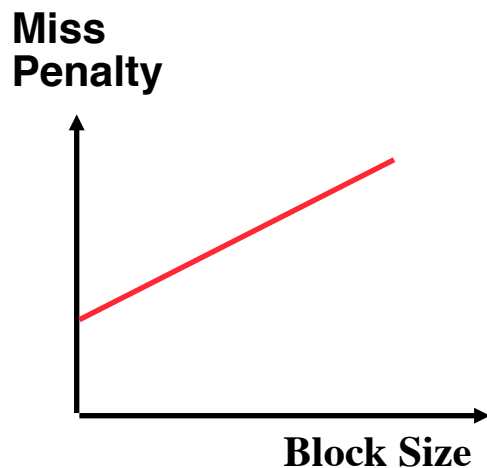


Block Size Tradeoff

- In general, larger block size 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

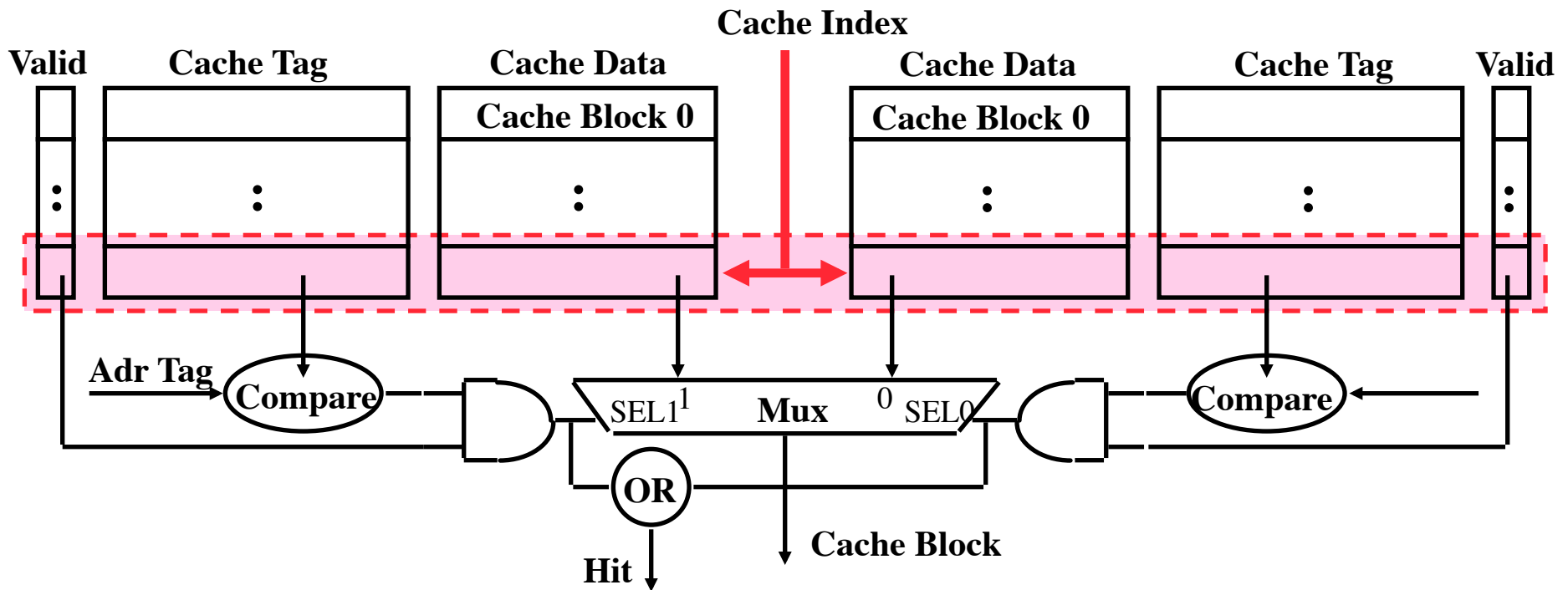
- In general, Average Access Time:

- ♦ $\text{Hit Time} \times (1 - \text{Miss Rate}) + \text{Miss Penalty} \times \text{Miss Rate}$



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.
- ✳ **Fewer Conflict Misses.**
- ✳ **Can have a larger cache but keep the index smaller (same size as virtual page index)**

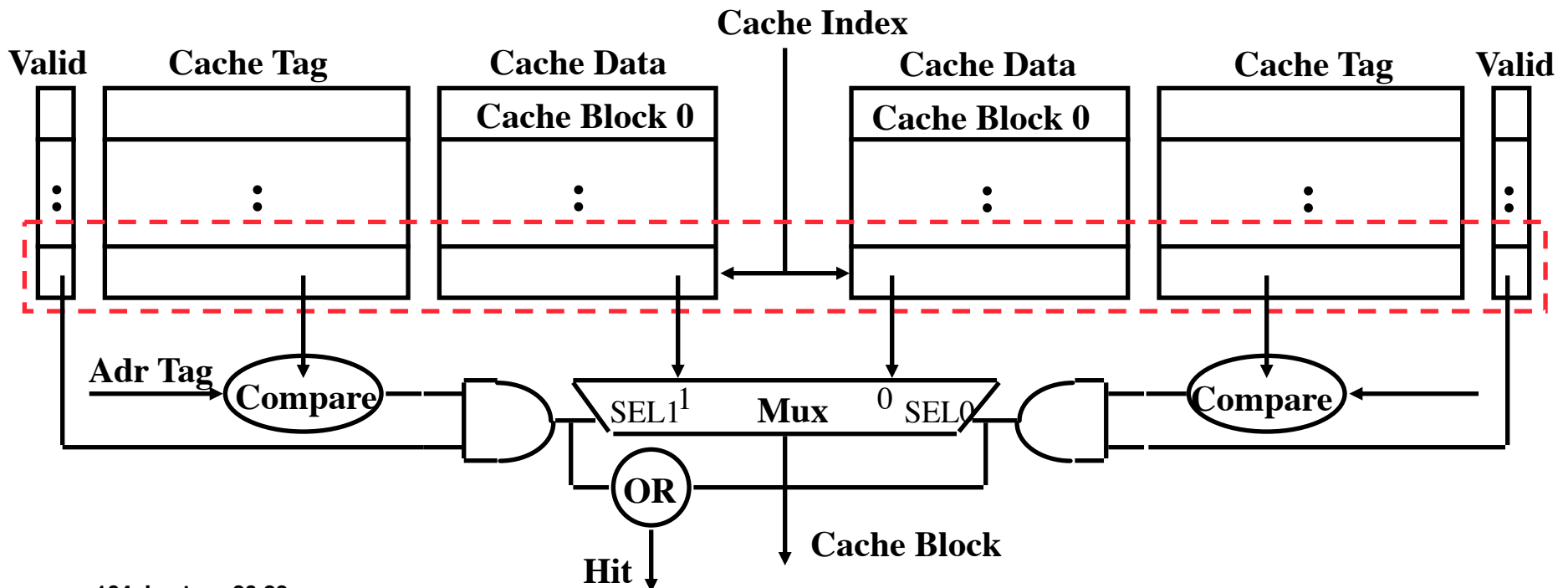
Disadvantage of Set Associative Cache

- **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

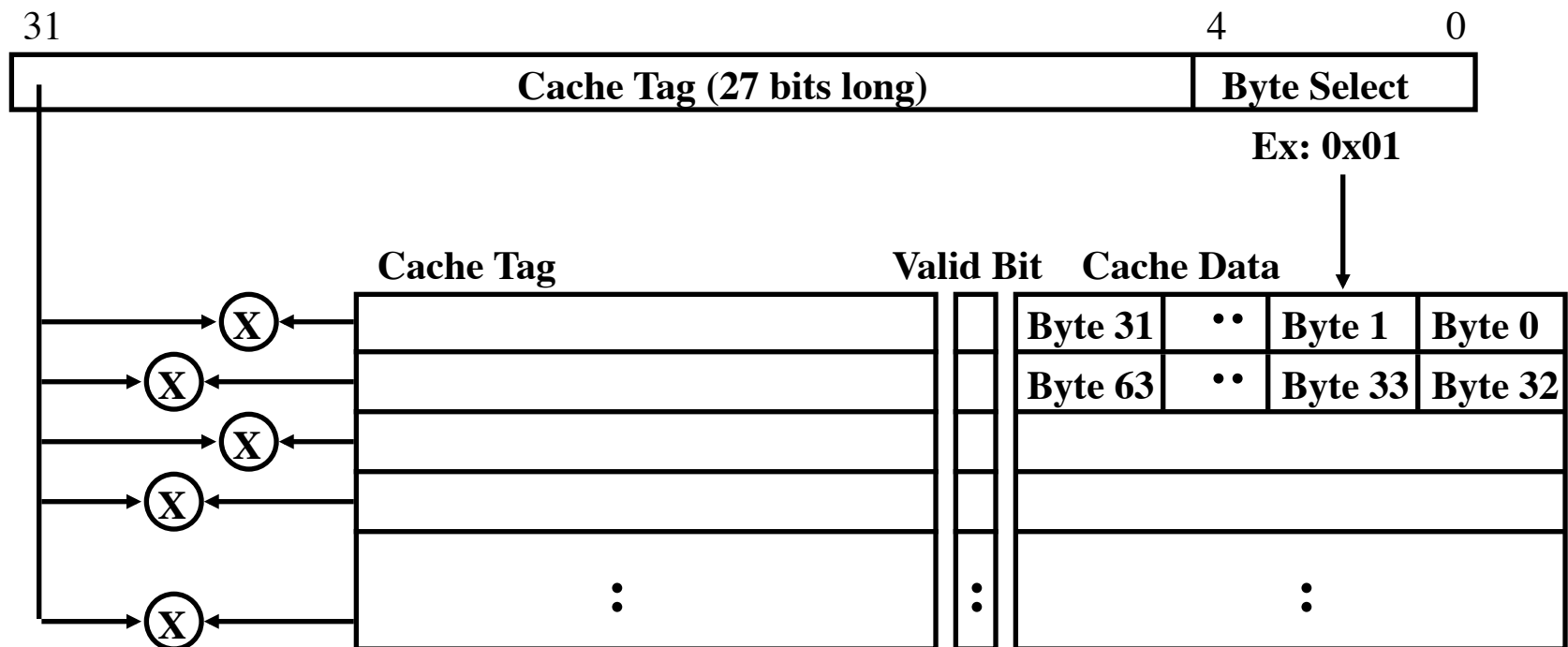
- **In a direct mapped cache, Cache Block is available **BEFORE** Hit/Miss:**

- ◆ Possible to assume a hit and continue. Recover later if miss.



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 access by the program
 - ◆ Solution: increase cache size
- **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.

Replacement; The Need to Make a Decision!

✱ Direct Mapped Cache:

- ◆ Each memory location can only be mapped 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:

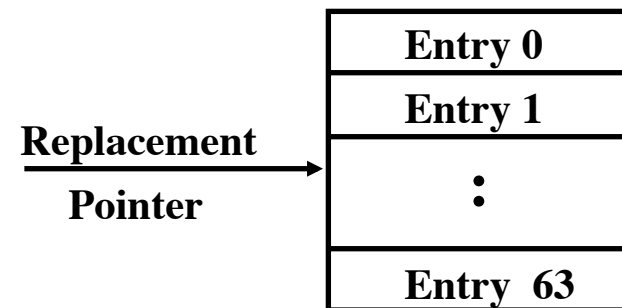
- ◆ Each memory location can be placed in **ANY** cache location

✱ Cache miss in a 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

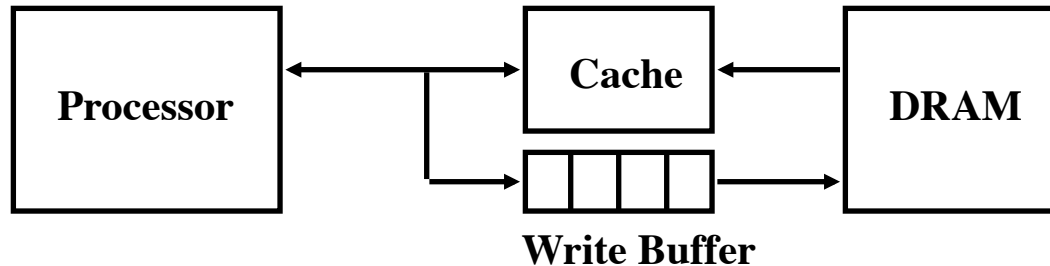
- **Random Replacement:**
 - ◆ Hardware randomly selects a cache block out of the set and replaces it.
- **Least Recently Used:**
 - ◆ Hardware keeps track of the access history
 - ◆ Replace the entry that has not been used for the longest time.
 - ◆ For **two way set associative** cache one needs **one 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



Cache Write Policy: Write Through versus Write Back

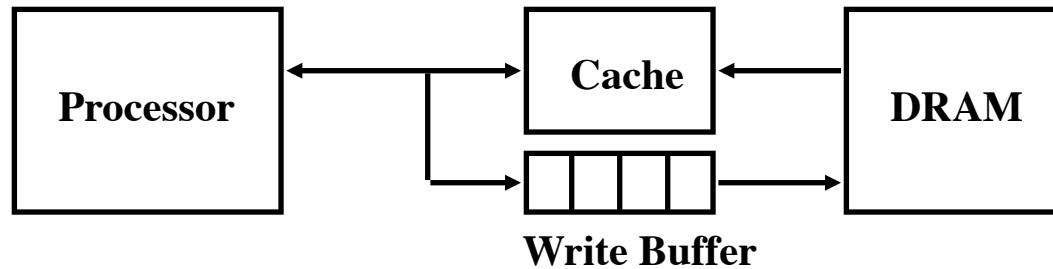
- Cache read is much easier to handle than cache write:
 - ◆ Instruction cache is much easier to design than data cache
- Cache write:
 - ◆ How do we keep data in the cache and memory consistent?
- Two options (decision time again :-)
 - ◆ **Write Back**: write to cache only. Write the cache block to memory when that cache block is being replaced on a cache miss.
 - Need a “**dirty bit**” for each cache block
 - Greatly reduce 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: 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) $\ll 1 / \text{DRAM write cycle}$
- **Memory system designer's nightmare:**
 - ◆ Store frequency (w.r.t. time) $> 1 / \text{DRAM write cycle}$
 - ◆ Write buffer saturation

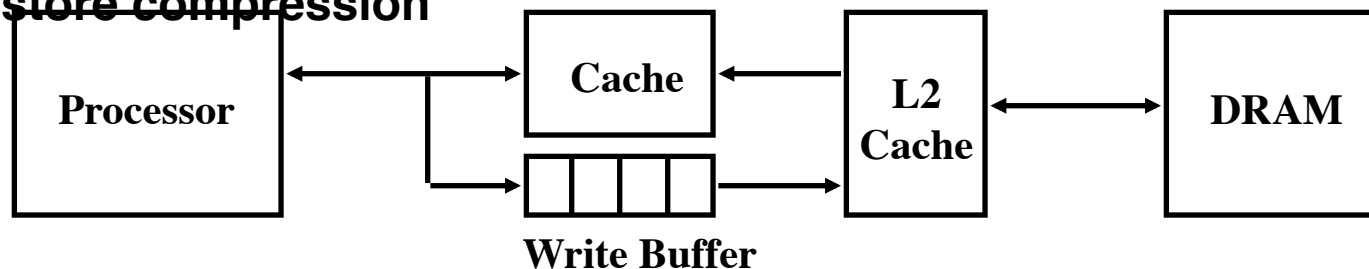
Write Buffer Saturation



- **Store frequency (w.r.t. time) $\rightarrow 1 / \text{DRAM write cycle}$**
 - ♦ If this condition exist for a long period of time (CPU cycle time too quick and/or too many store instructions in a row):
 - \rightarrow Store buffer will overflow no matter how big you make it
 - \rightarrow The CPU Cycle Time \ll 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 and causes a miss

◆ Do we read in the block?

Yes: Write Allocate

No: Write Not Allocate

