Lecture 8

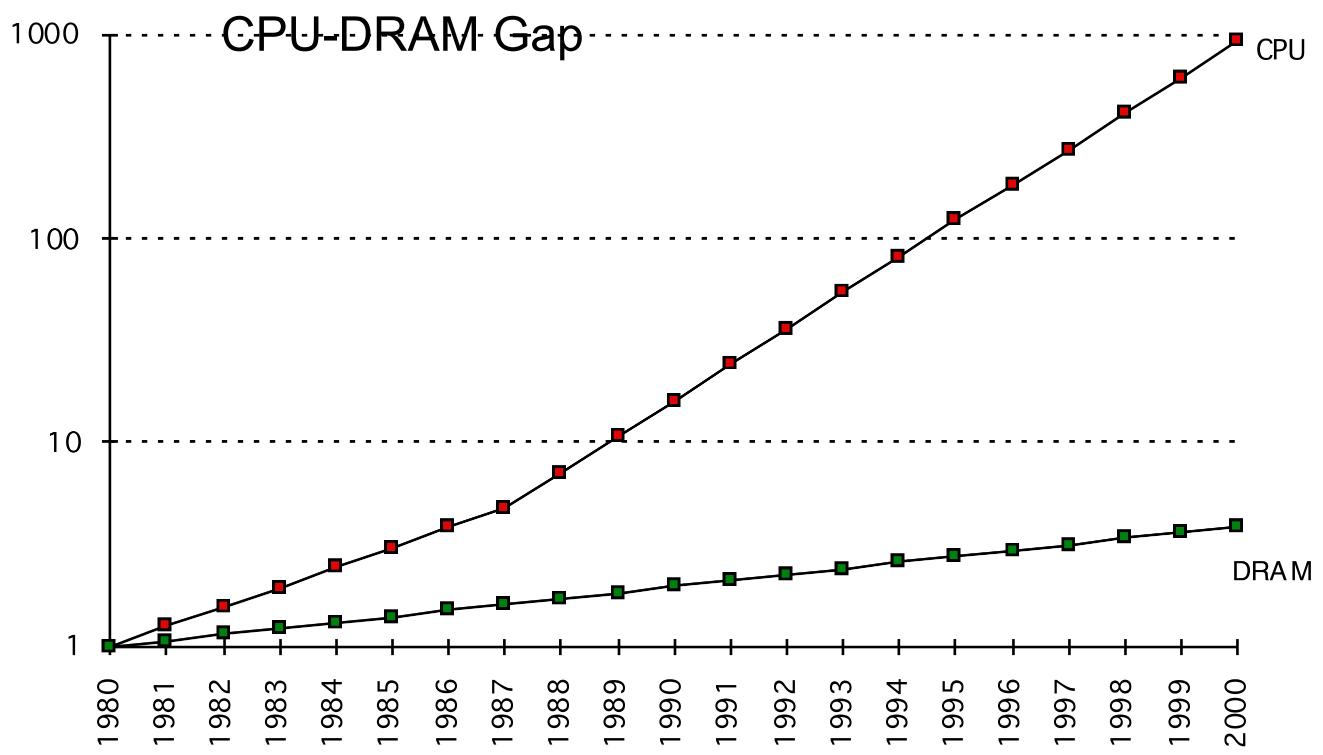
Memory Hierarchy

Memory Hierarchy

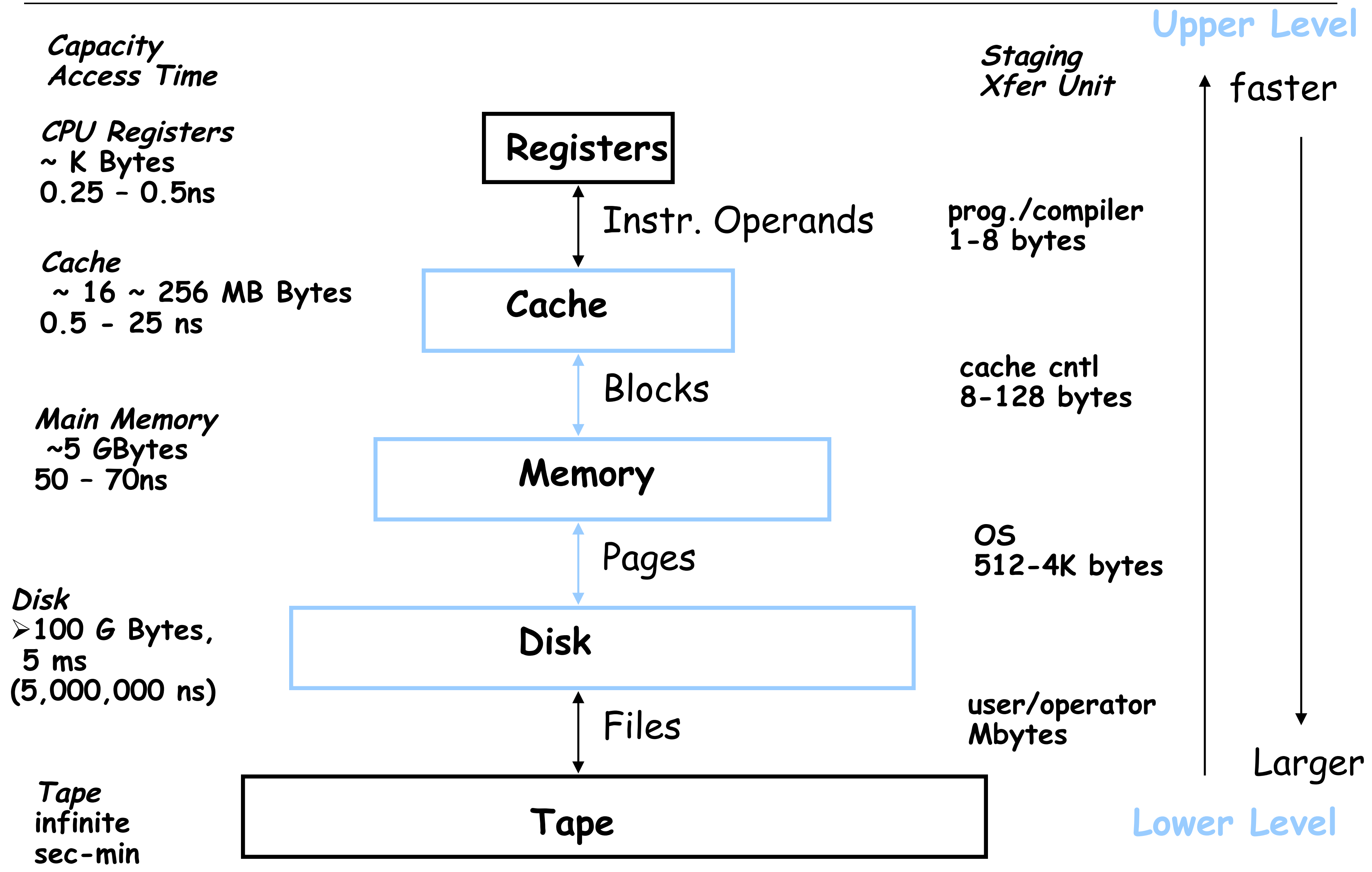
- Keeping useful books close to you
- Keeping useful data close to the CPU
  - Memory Hierarchy

Why Cares about Memory Hierarchy?

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



Levels of the Memory Hierarchy



General Principle

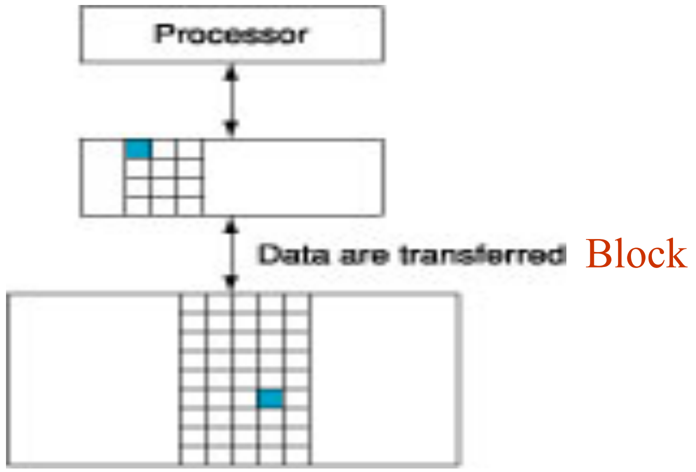
- The Principle of Locality:
  - Programs access a relatively small portion of the address space at any instant of time.
- Two Different Types of Locality:
  - Temporal Locality (Locality in Time):
    - If an item is referenced, it will tend to be referenced again soon (e.g., loops, reuse)
  - Spatial 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)

```
For (i= 0; i++;i:<100)
    {
     A[i]=A[i] + x;
     x++;
     }
```

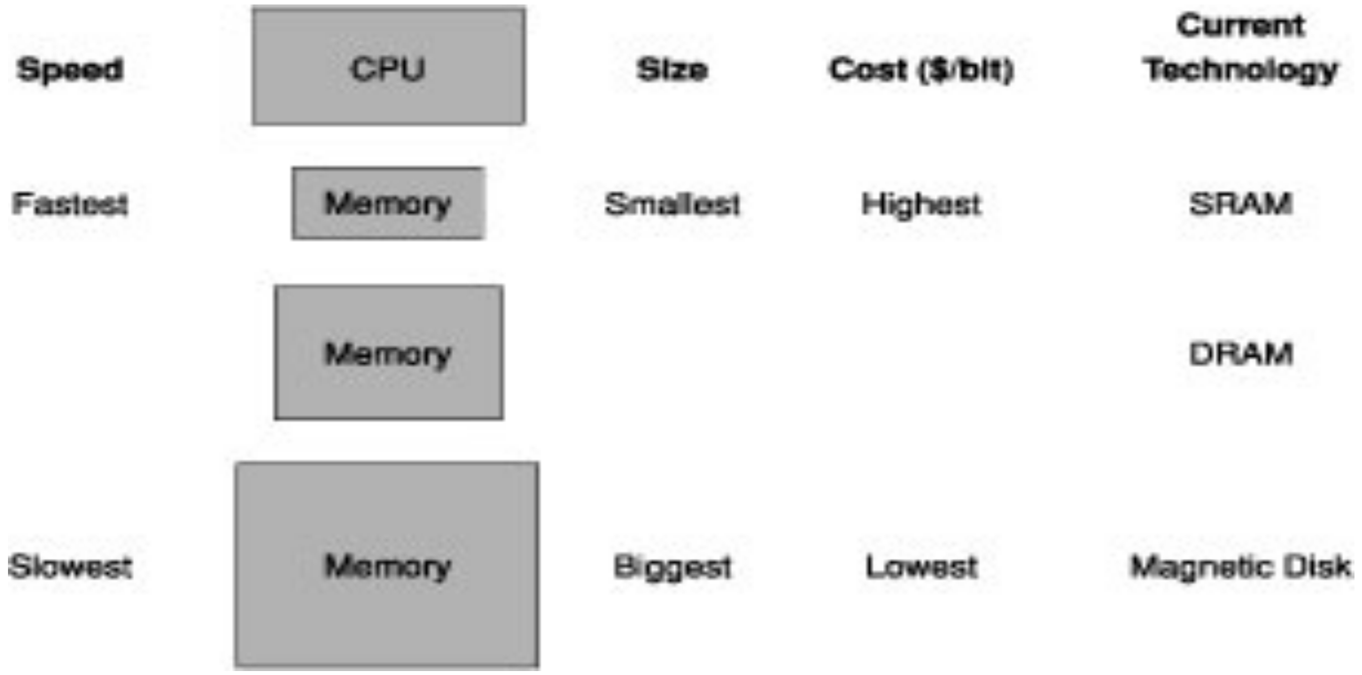
- Locality + smaller HW is faster = memory hierarchy
  - *Levels*: each smaller, faster, more expensive/byte than level below
  - *Inclusive*: data found in top also found in the bottom

Memory Hierarchy: Terminology

- Definitions
  - *Upper* is closer to processor
  - *Block*: minimum unit that present or not in upper level
- 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 retrieved 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 Alpha 21264!)

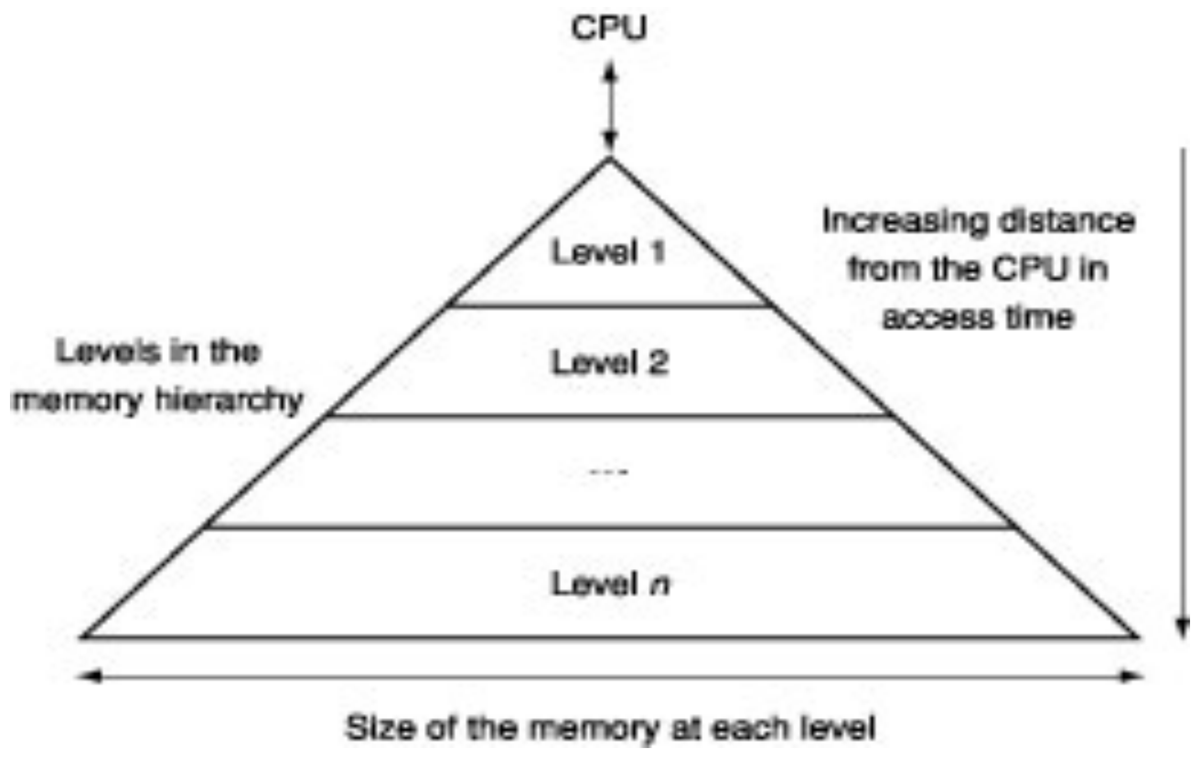


Basic Structure of a Memory Hierarchy



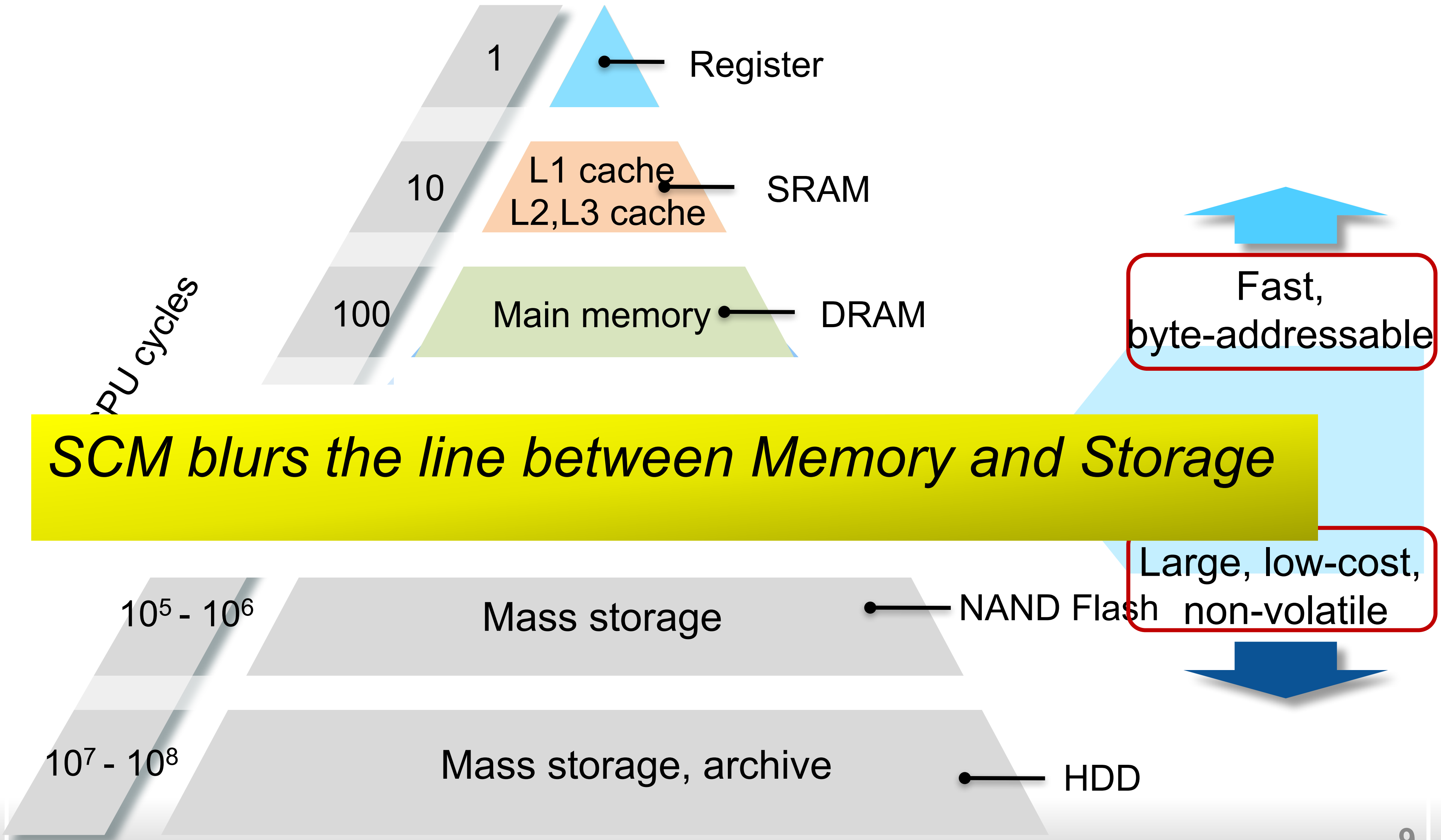
Memory Technology	Typical access time	$ per GB in 2012
SRAM	0.5-5 ns	$500-$1000
DRAM	50 – 70 ns	$10-$20
Flash Memory	5,000 ~ 50,000ns	$0.75 -$1
Magnetic disk	5,000,000-20,000,000 ns	$0.50-$2

Memory Hierarchy


CPU
Level 1
Level 2
...
Level n
Levels in the memory hierarchy
Increasing distance from the CPU in access time
Size of the memory at each level

1. Keeping more recently accessed data items closer to the processor -> temporal locality
2. Moving blocks consisting of multiple contiguous words in memory to upper levels of the hierarchy -> spatial locality
3. Data cannot be present in level i unless it is also present in level i+1

Storage Class Memory



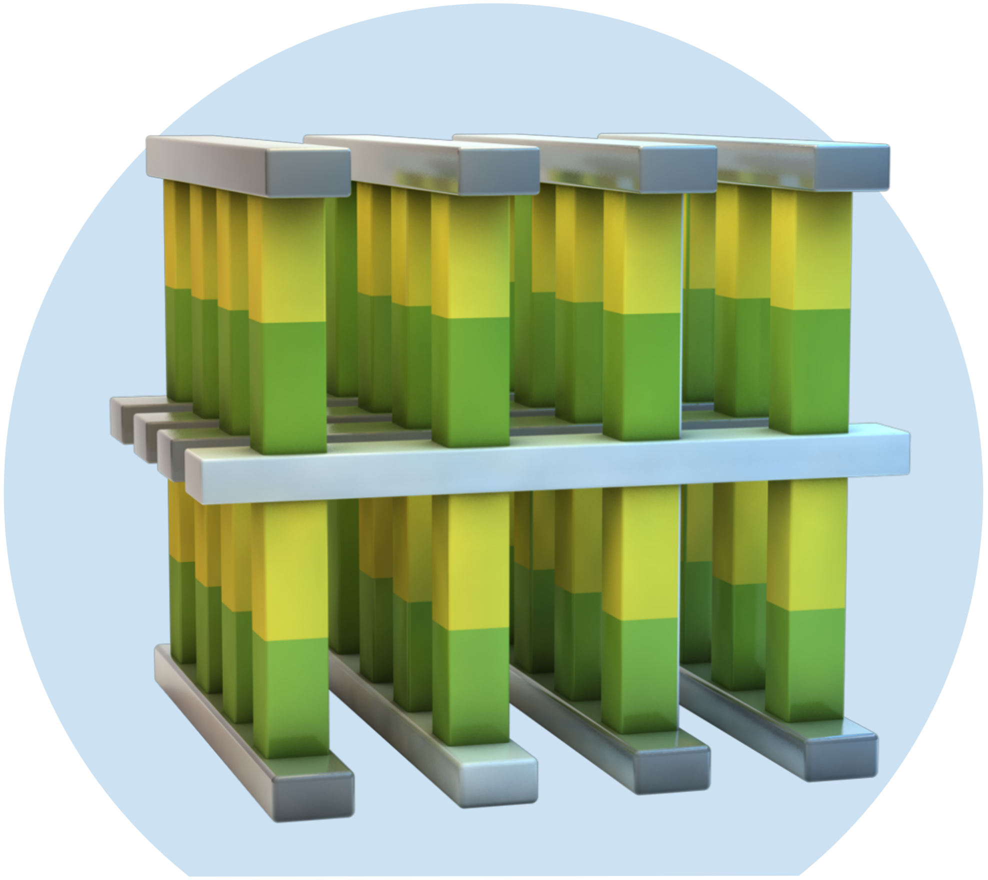
SCM blurs the line between Memory and Storage

Intel/Micron: Introducing 3D XPoint™

1000X
FASTER
THAN NAND

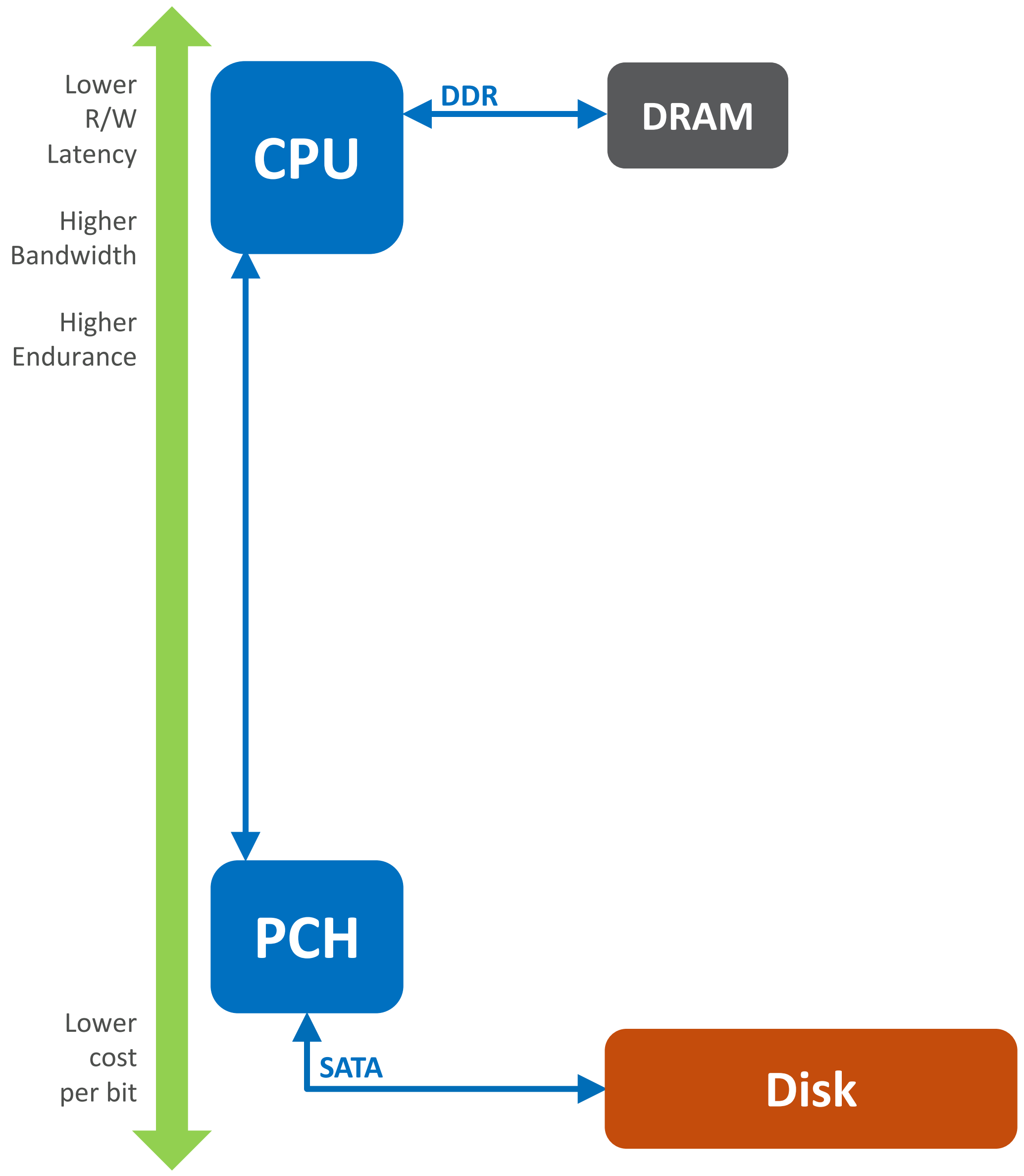
1000X
ENDURANCE
OF NAND

10X
DENSER
THAN CONVENTIONAL MEMORY



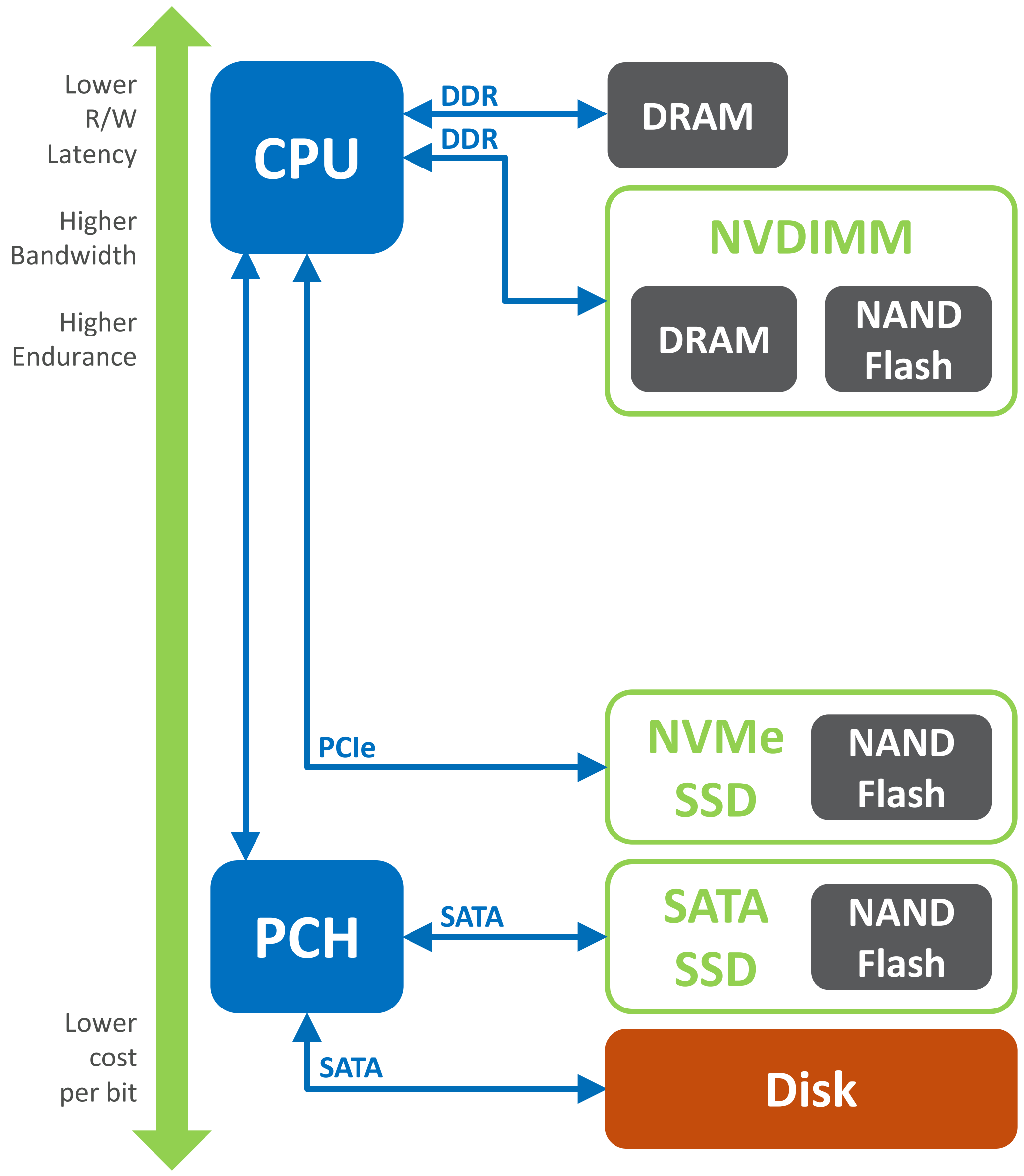
3D XPoint

The Past: Nonvolatile Memories in Server Architectures



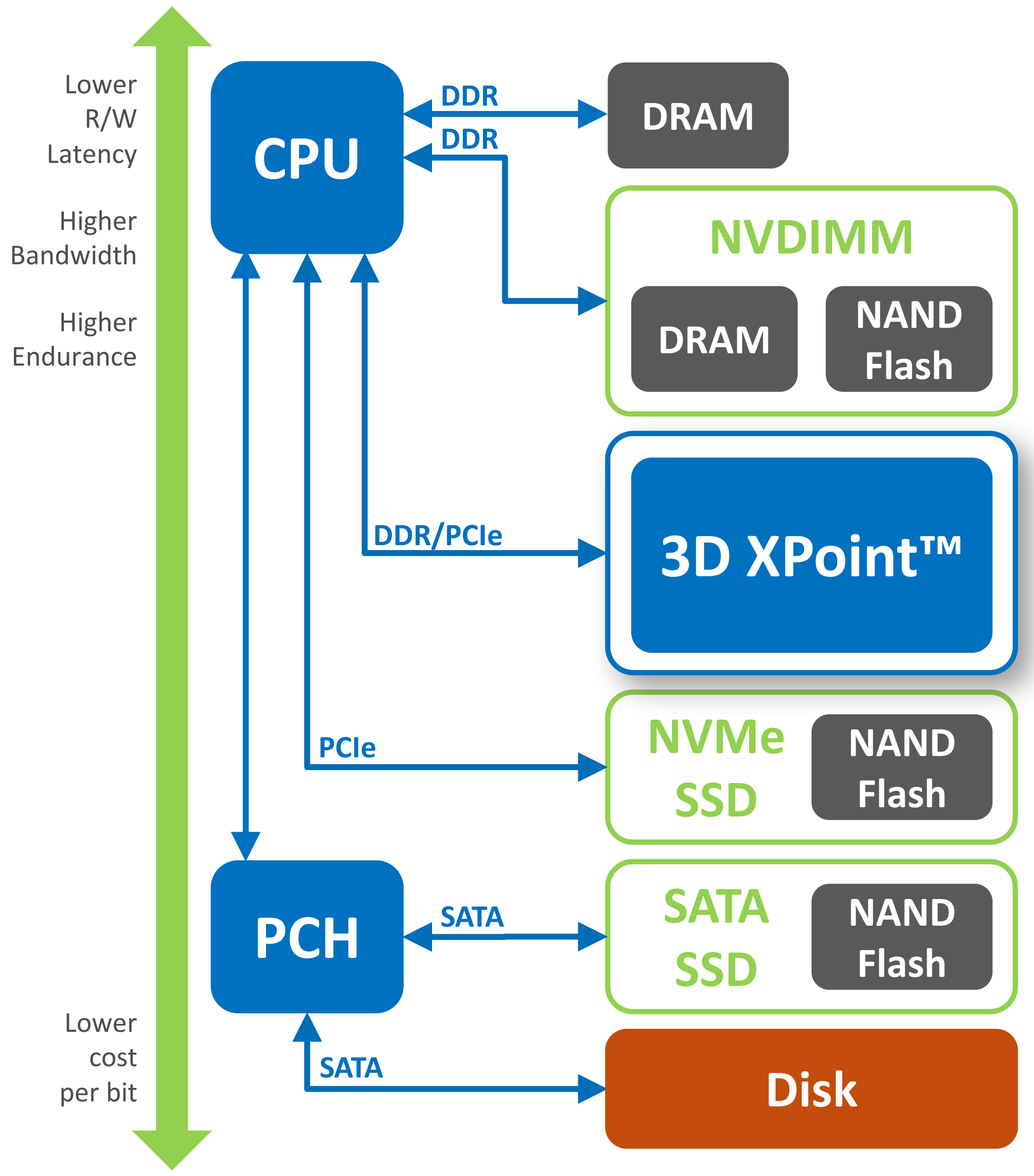
- In the old days of not long ago we had two primary types of memories in computers: DRAM and Hard Disk Drive (HDD)
- DRAM was fast and volatile and HDDs were slower, but nonvolatile
- Data moves from the HDD to DRAM where it is the fed to the processor
- The processor writes the result in DRAM and then it is stored back to disk to remain for future use

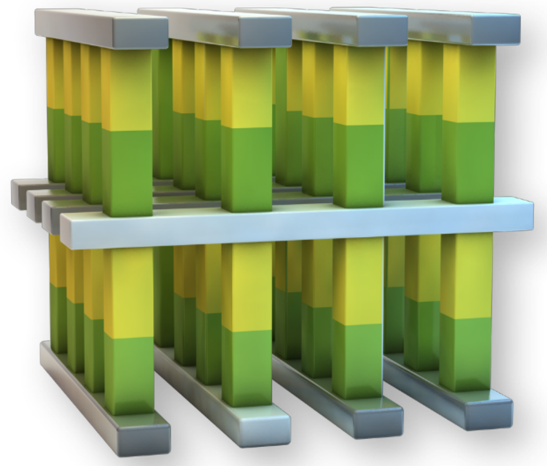
The Present: Nonvolatile Memories in Server Architectures



- System performance increased as the speed of both the interface and the memory accesses improved
- NAND Flash considerably improved the nonvolatile response time
- SATA and PCIe made further optimization to the storage interface
- NVDIMM provides battery- or super capacitor-backed DRAM, operating at DRAM-like speeds and retains data when power is removed

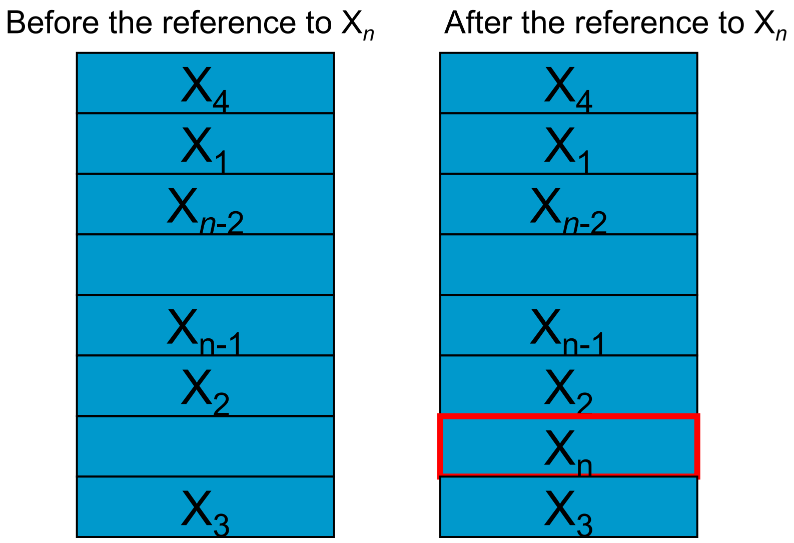
The Future: Nonvolatile Memories in Server Architectures





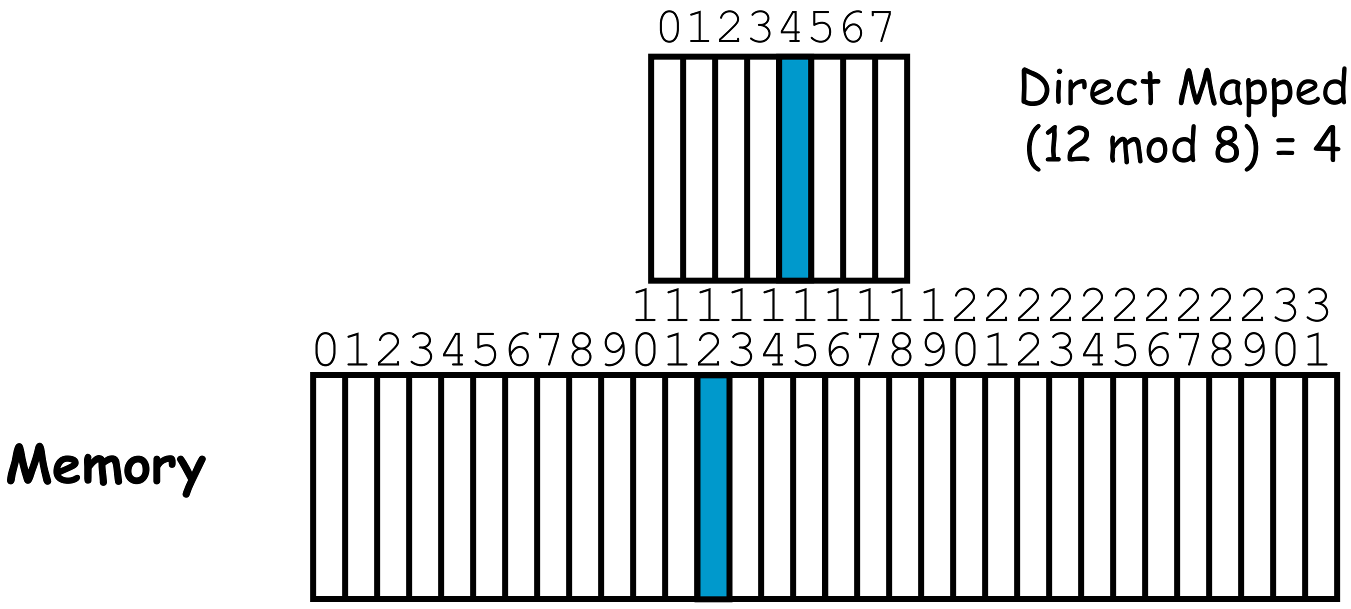
- 3D XPoint technology provides the benefit in the middle
- It is considerably faster than NAND Flash
- Performance can be realized on PCIe or DDR buses
- Lower cost per bit than DRAM while being considerably more dense

Basics of Caches

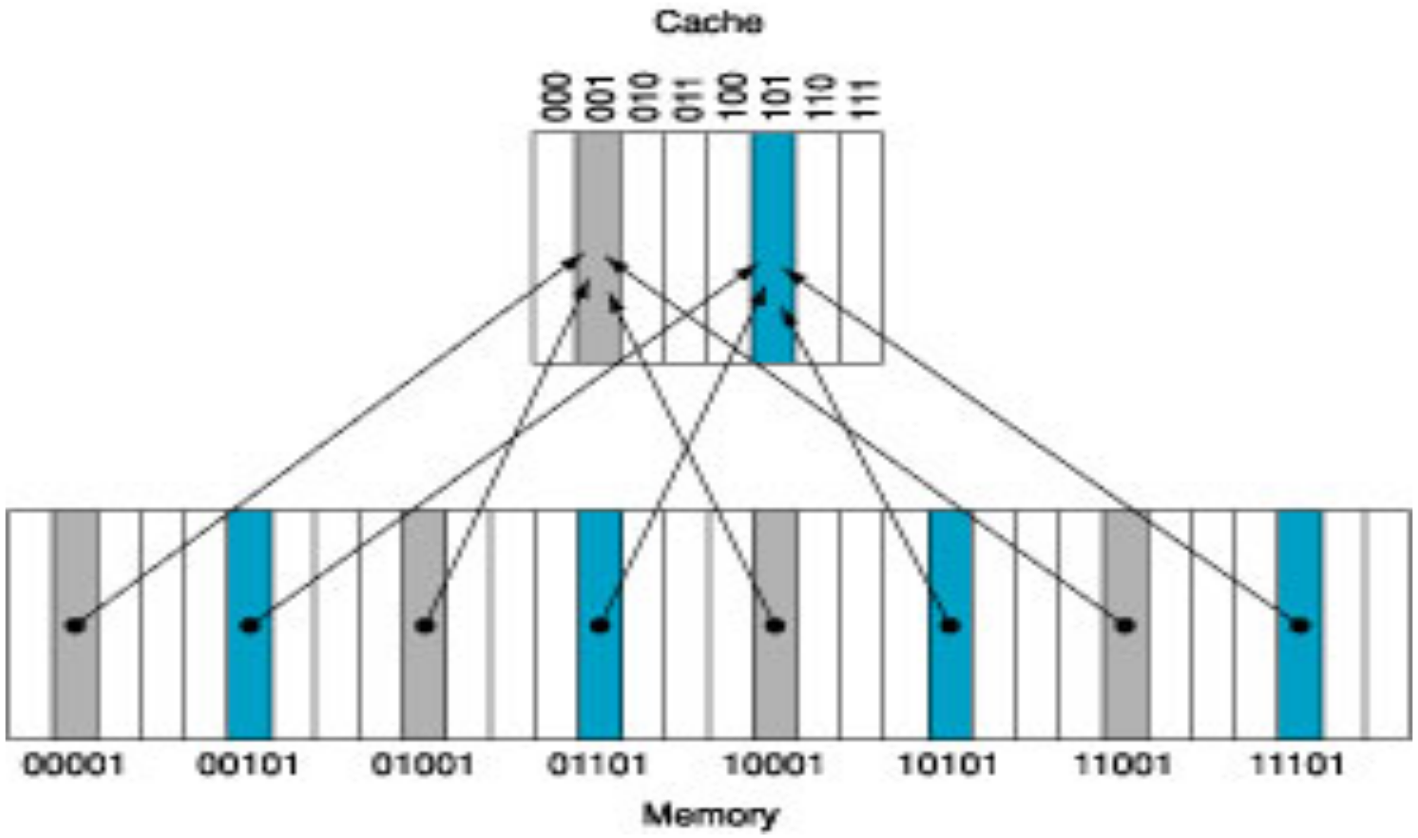


How do we find X*n* in cache?

Direct-mapped cache



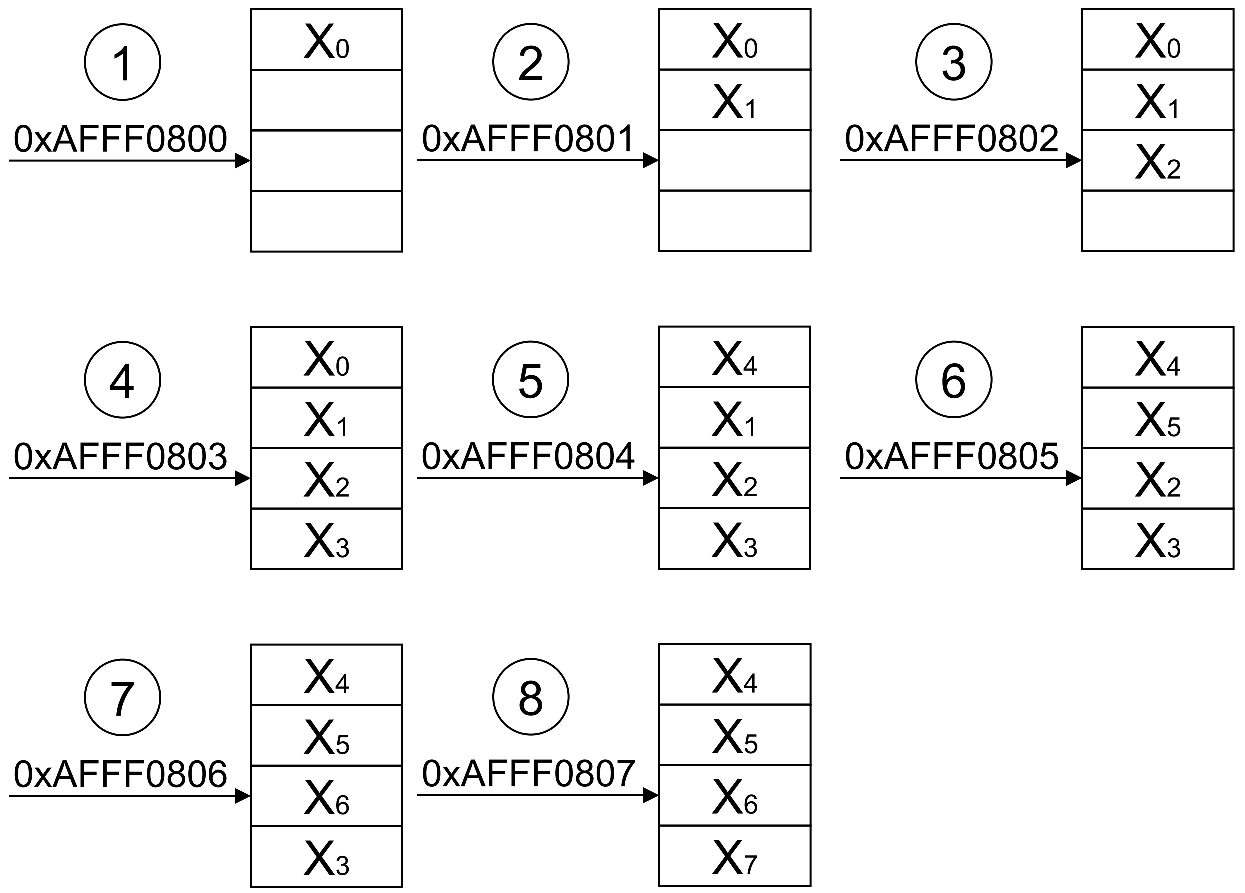
Direct-mapped cache (cont.)



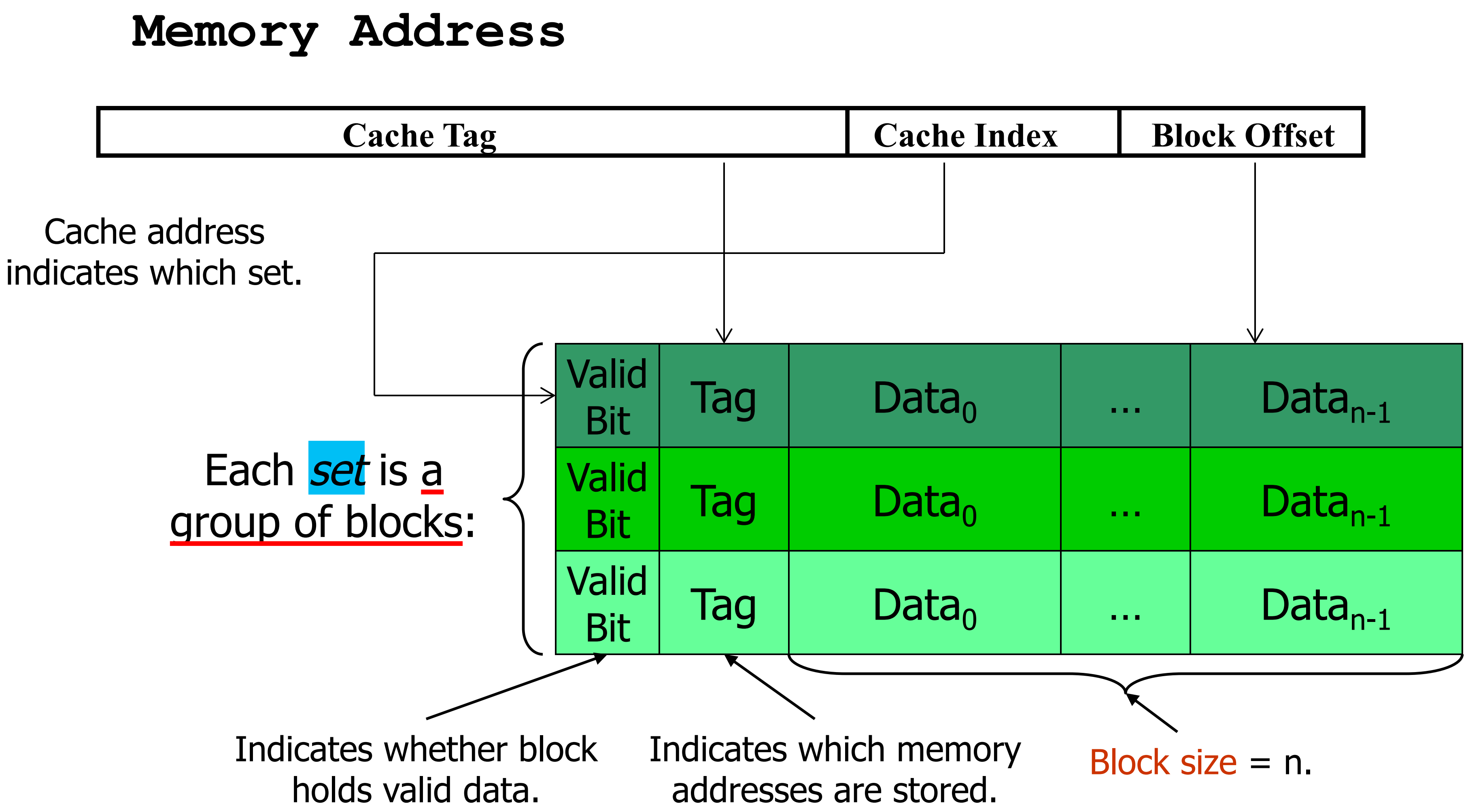
1. Multiple data items map to the same cache location
2. How do we know whether a requested word is in the cache or not?

Example

Address	Data
0xAFFF0800	X_0
0xAFFF0801	X_1
0xAFFF0802	X_2
0xAFFF0803	X_3
0xAFFF0804	X_4
0xAFFF0805	X_5
0xAFFF0806	X_6
0xAFFF0807	X_7



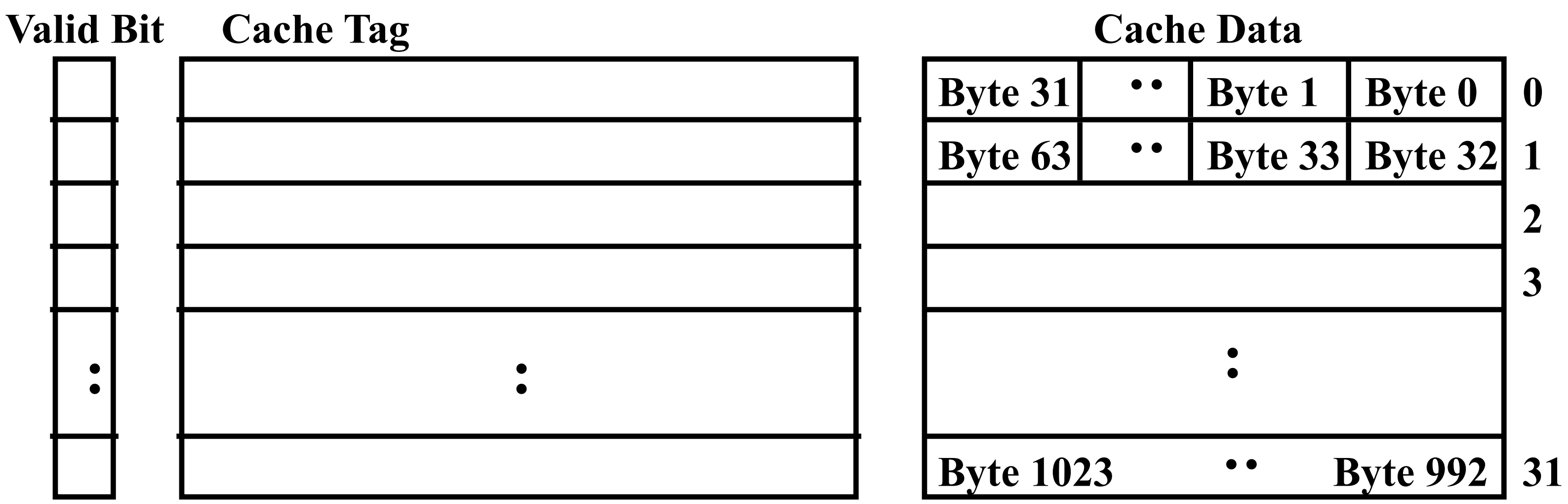
How is a block found if it is in the upper level?


Memory Address
Cache Tag
Cache Index
Block Offset
Cache address indicates which set.
Each *set* is a group of blocks:
Valid Bit
Tag
Data_0
…
Data_{n-1}
Indicates whether block holds valid data.
Indicates which memory addresses are stored.
Block size = n.

1 KB Direct Mapped Cache, 32B blocks

- How many sets?

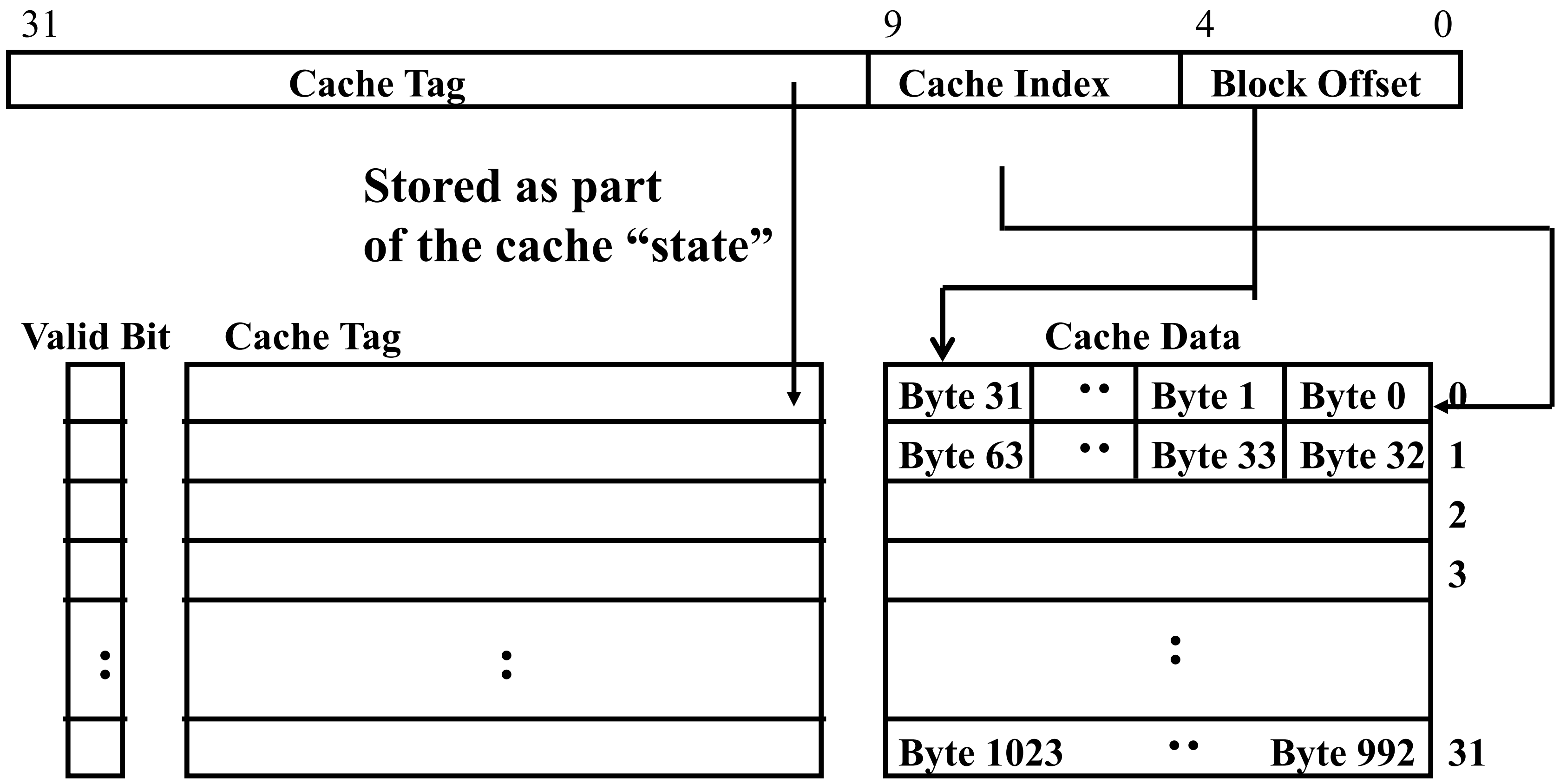
$$\frac{2\text{^}10}{2\text{^}5} = 32$$



1 KB Direct Mapped Cache, 32B blocks

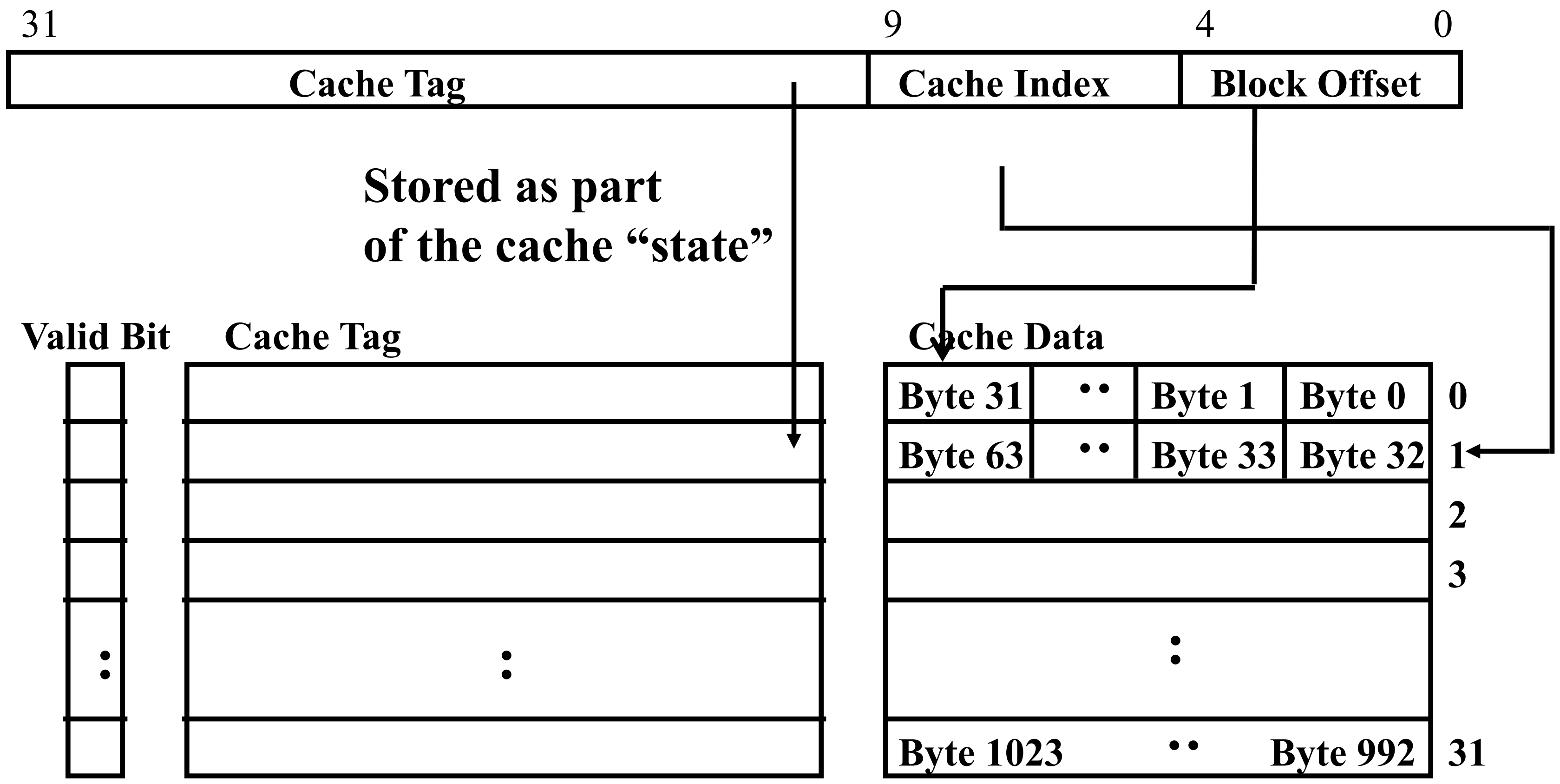
- What is the cache address and tag values for 0x00C0A01F?

0000,0000,1100,0000,1010,0000,0001,1111

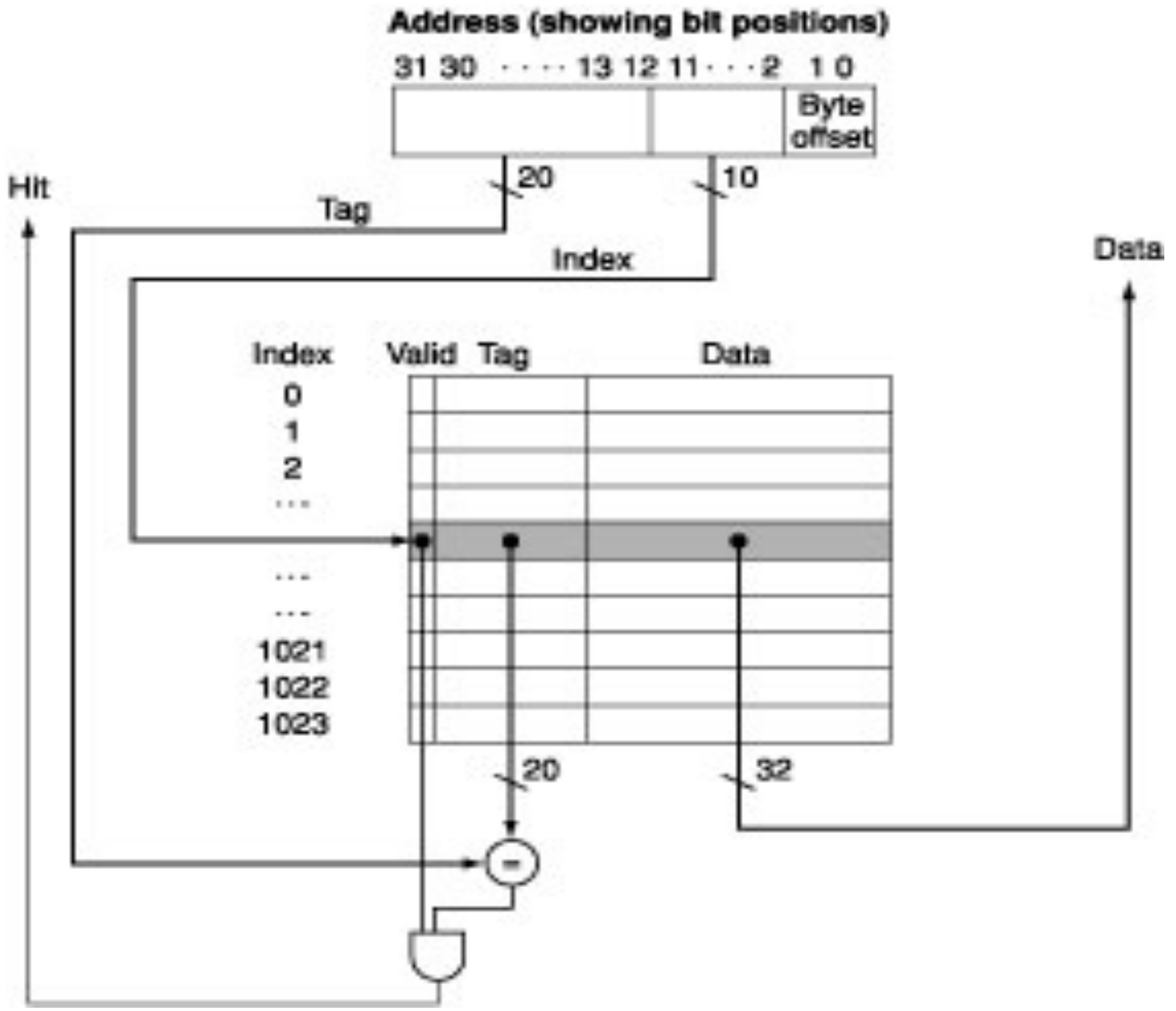


1 KB Direct Mapped Cache, 32B blocks

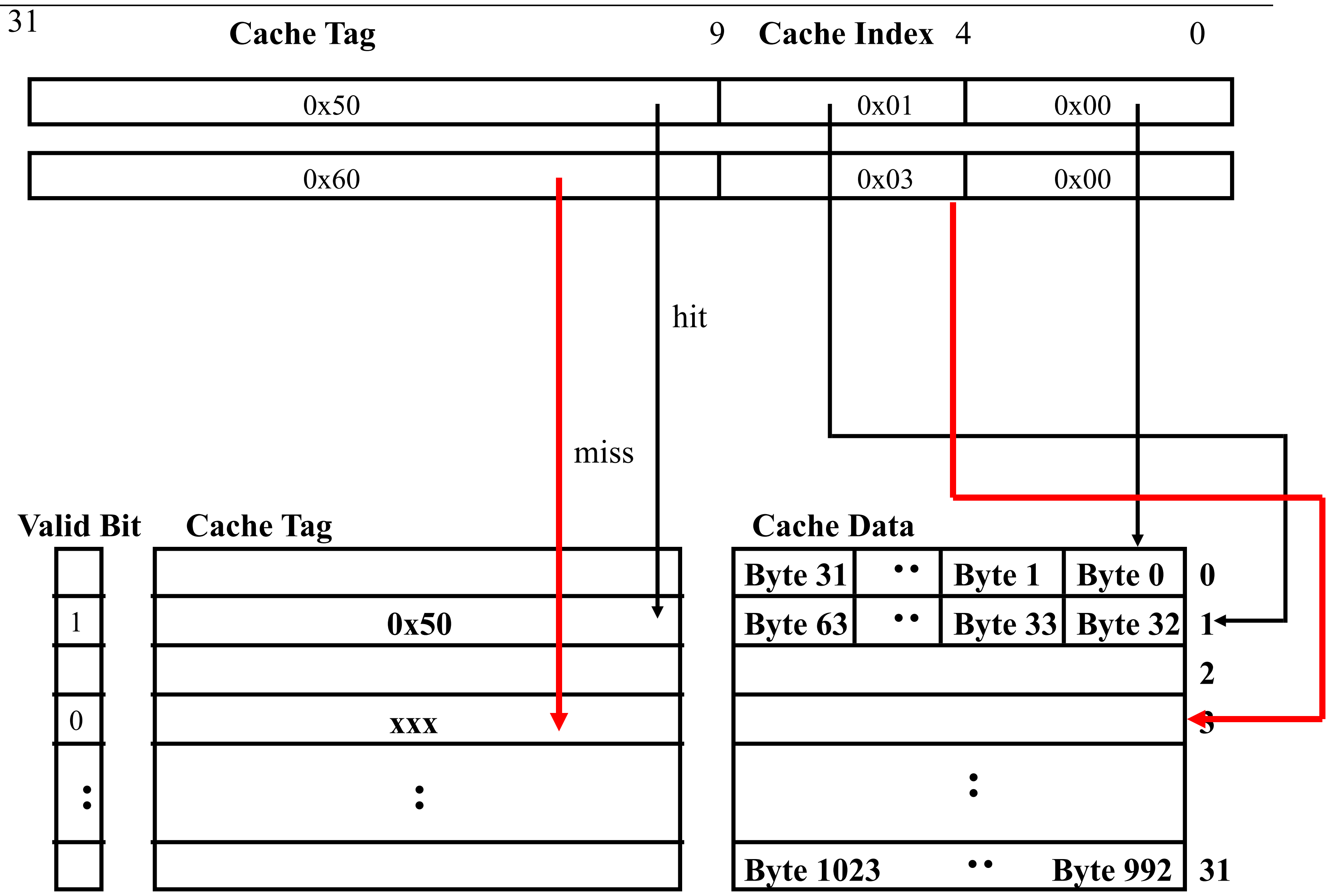
- For a 2 ** N byte direct-mapped cache:
  - The uppermost (32 - N) bits are always the Cache Tag
  - The lowest M bits are the Byte Select (Block Size = 2 ** M)



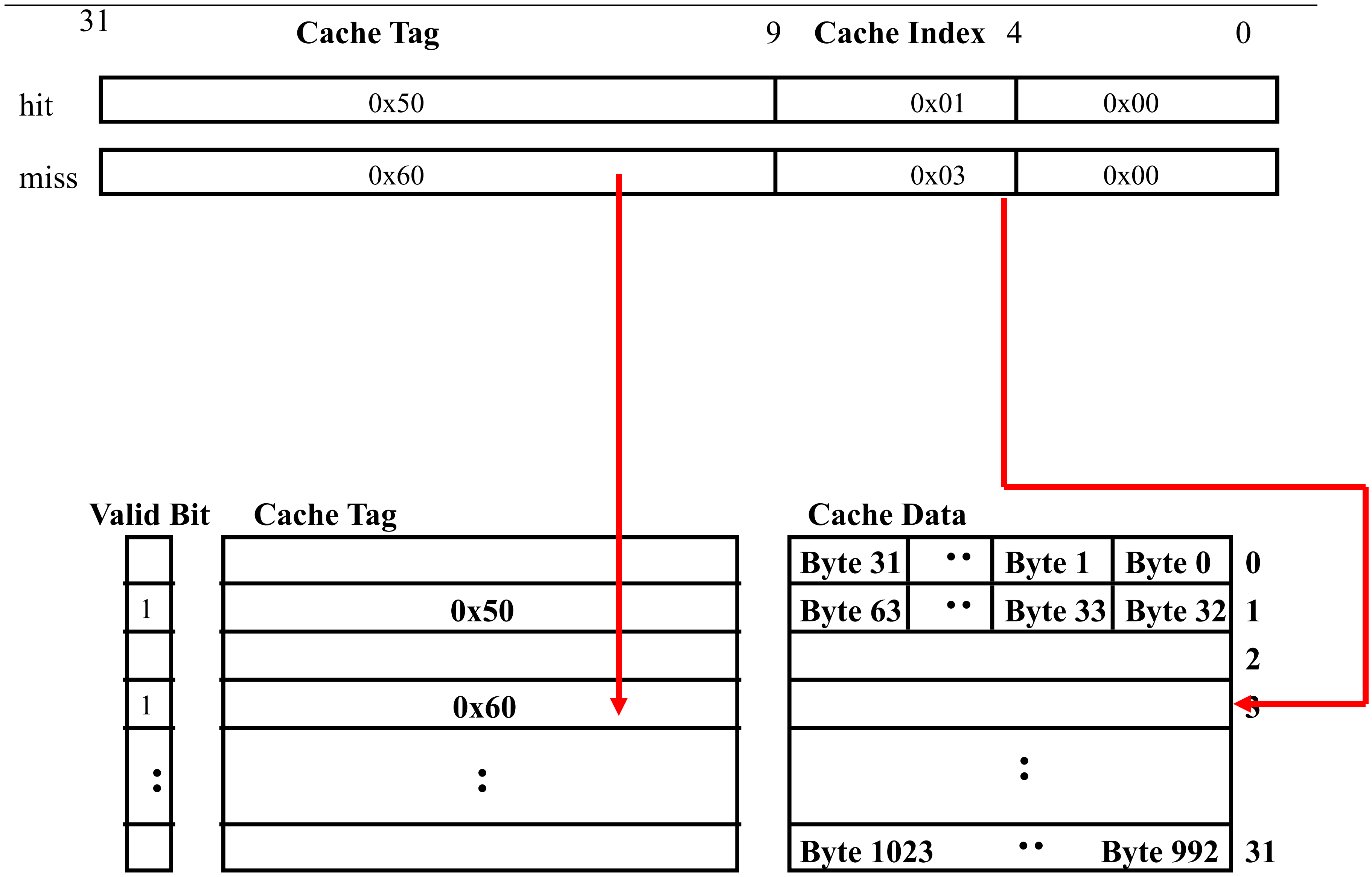
Cache Access



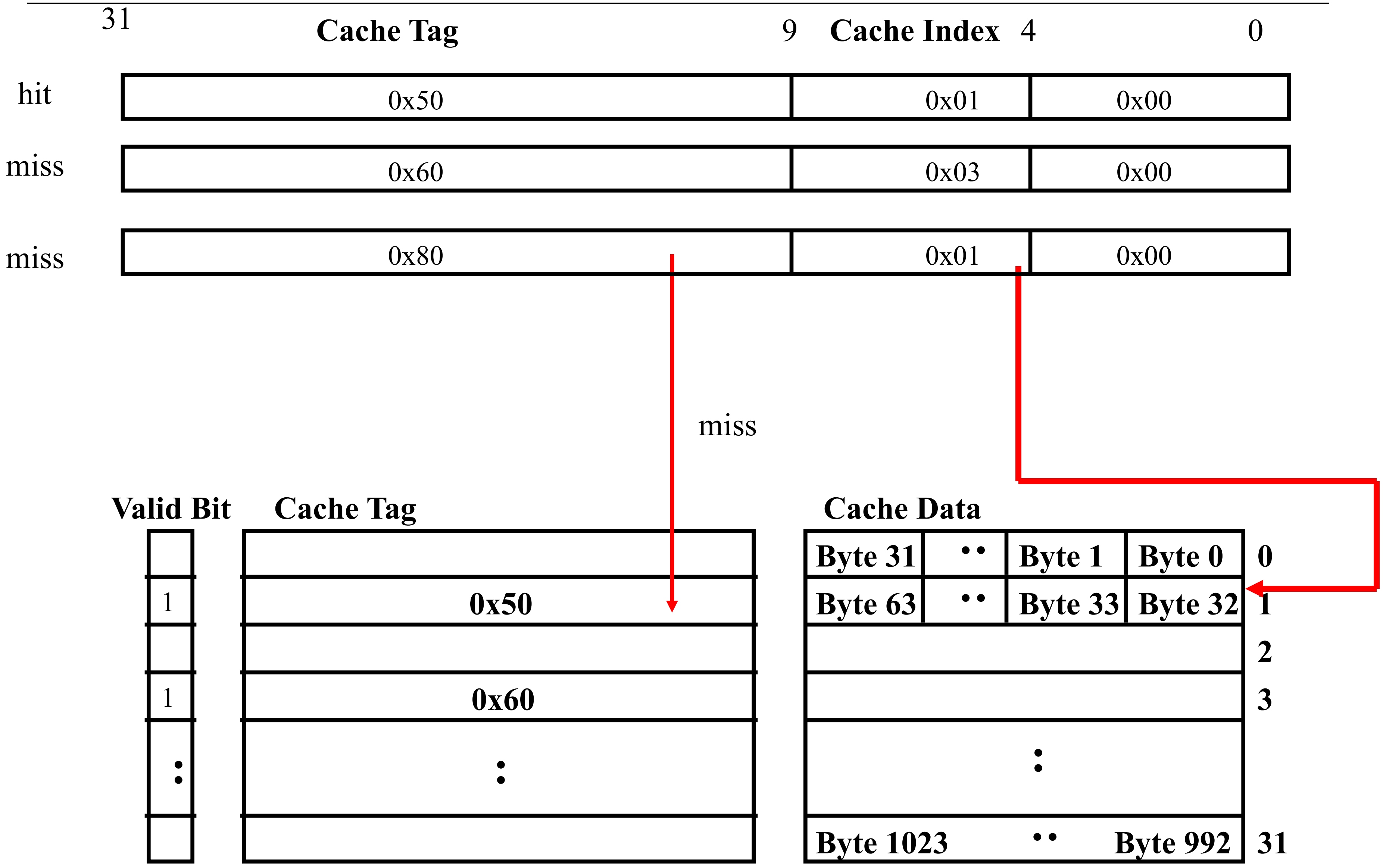
Example: 1 KB Direct Mapped Cache, 32B blocks



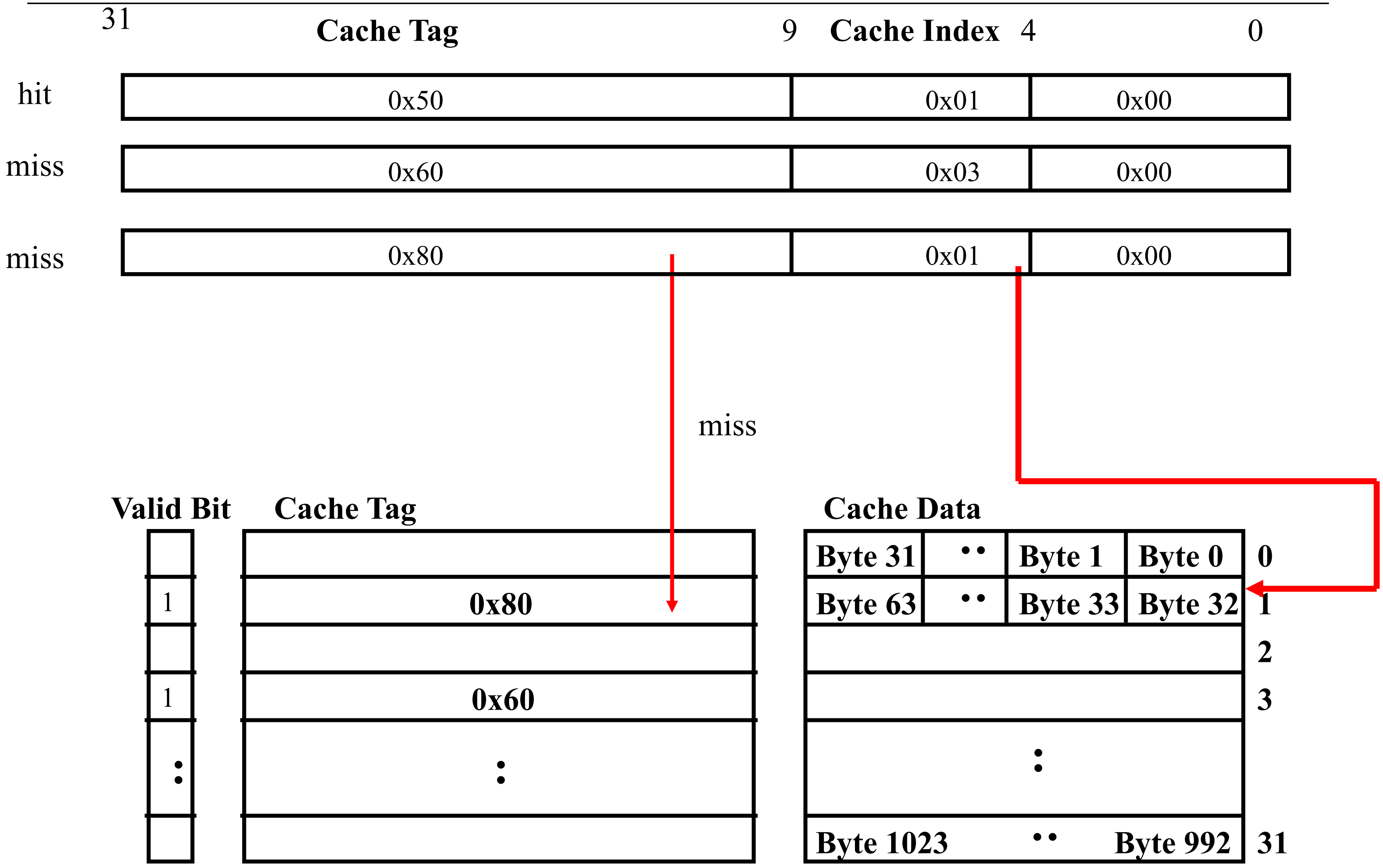
Example: 1 KB Direct Mapped Cache, 32B blocks



Example: 1 KB Direct Mapped Cache, 32B blocks



Example: 1 KB Direct Mapped Cache, 32B blocks



Exercise

- Show the cache contents of an eight-word direct-mapped caches (1-word block size) after each reference for the following address trace (word addressing) :

  10110_{two}, 11010_{two}, 10110_{two}, 110101_{two}, 10000_{two}, 00011_{two}, 10000_{two}, 100010_{two}

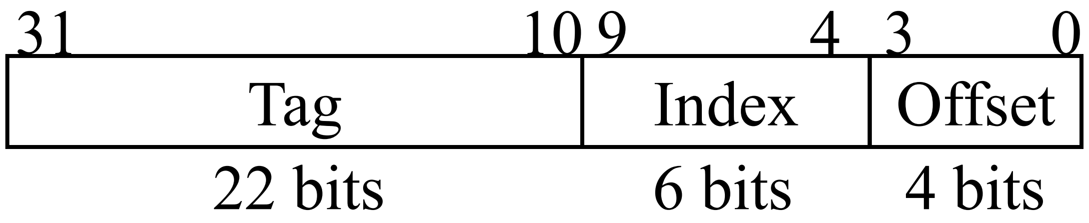
Exercise

- How many total bits are required for a direct-mapped cache with 16 KB of data and 4-word blocks, assuming a 32-bit address?
  - # of sets = ?
  - # of data bits for each set =?
  - # of tag bits for each set = ?
  - Valid bit for each set = 1
  - total cache bits = # of set x (valid bit (1-bit) + tag bits + data bits)

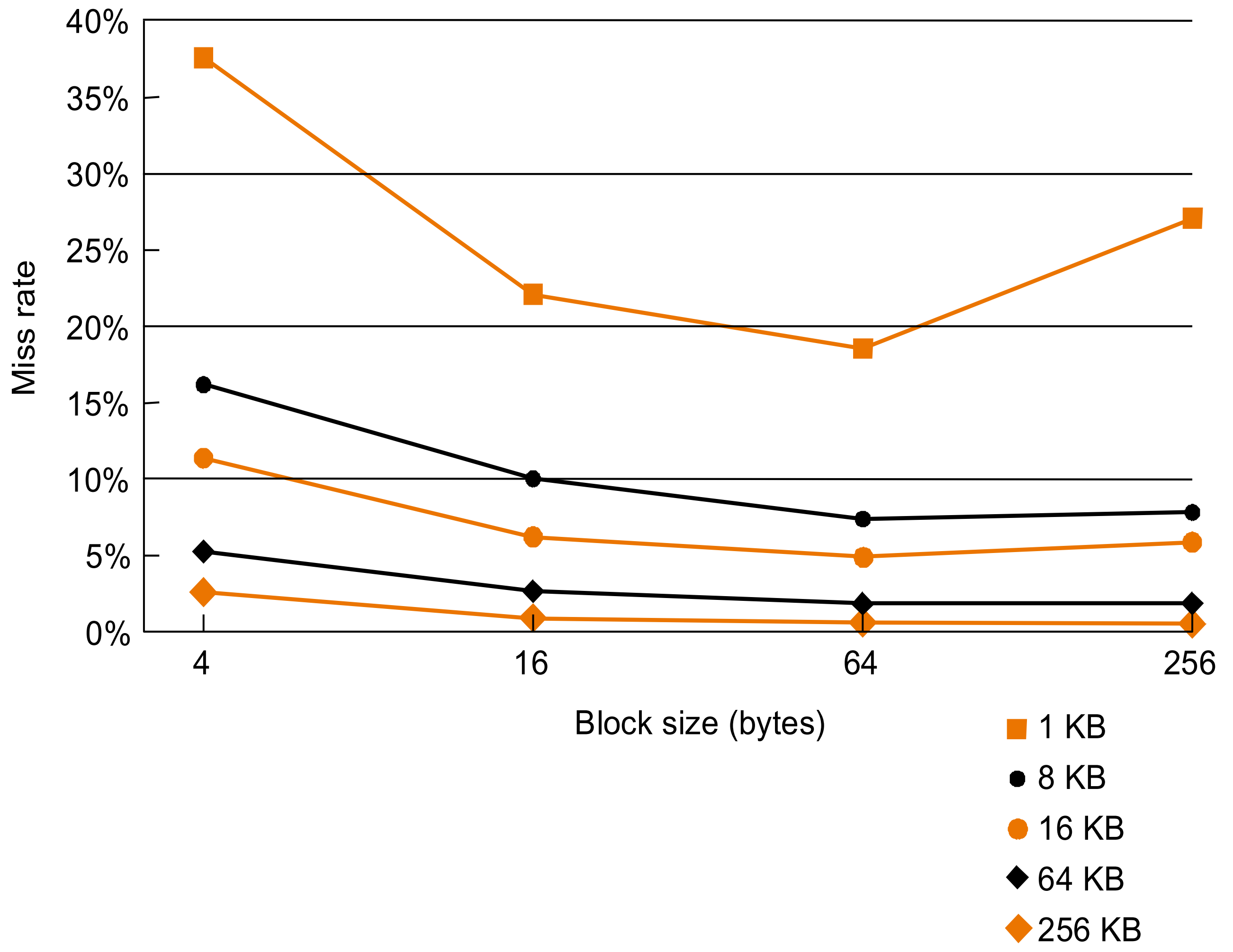
Exercise

- Consider a cache with 64 blocks and a block size of 16 bytes. What block number does byte address 1200 map to?

- Block address = $\lfloor 1200/16 \rfloor$ = 75
- Block number = 75 modulo 64 = 11



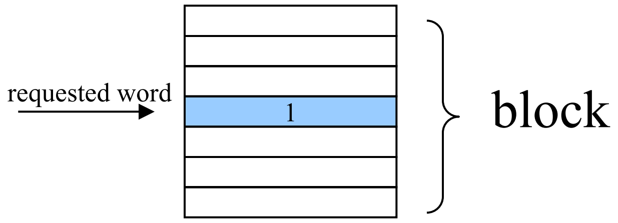
Block Size



40%
35%
30%
25%
20%
15%
10%
5%
0%
Miss rate
4
16
64
256
Block size (bytes)
1 KB
8 KB
16 KB
64 KB
256 KB

Block Size (cont.)

- Advantage of larger block size
  - take advantage of spatial locality
- Disadvantage
  - Too few blocks in cache => high competition
  - Longer cache miss penalty
    - Early restart
      - Resume execution as soon as the requested word of the block is returned
    - Requested word first (critical word first)



Handling Cache Misses

Cache miss =>

Stall the entire pipeline & fetch the requested word

Steps to handle an instruction cache miss:

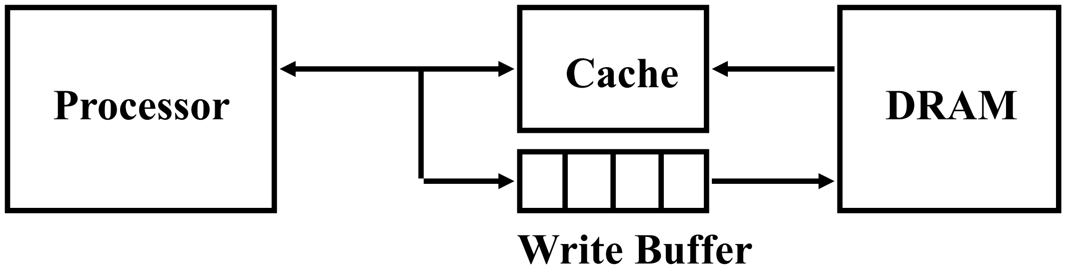
1. Send the original PC value (PC-4) to the memory.
2. Instruct main memory to perform a read and wait for the memory to complete its access.
3. Write the cache entry, putting the data from memory in the data portion of the entry, writing the upper bits of the address (from the ALU) into the tag field, and turning the valid bit on.
4. Restart the instruction execution at the first step, which will refresh the instruction, this time finding it in the cache.

Note that the control of the cache on data access is essentially identical as Instruction access shown above.

Handling Writes

- *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 & data coherency
    - Bad: write stall
  - WB:
    - no repeated writes to same location
    - Write new data to cache & write modified block to the lower level of memory hierarchy

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
- Note: many write-back caches also include write buffers that are used to reduce the miss penalty

Write Miss Policy

- Why there is a “write miss policy”, but no “read miss policy”?

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 cache

	Write through	Write back
Write allocate	hit: write to cache/memory miss: load block into cache; write to cache/memory	hit: write to cache, set dirty bit. miss: load block into cache; write to cache;set dirty bit
Write around	hit: write to cache/memory miss: write to memory	hit: write to cache, set dirty bit. miss: write to memory

Example: Intrinsity FastMath Processor

- Intrinsity FastMATH
  - embedded microprocessor using the MIPS architecture
  - 12-stage pipeline
  - Separate instruction/data caches (split cache), 16 KB, 16-word blocks
  - Offer both write-through and write-back
  - One-entry write buffer.
- Miss rate of Intrinsity FastMATH for SPEC2000:
  - Instruction miss rate : 0.4%
  - Data miss rate: 11.4%
  - Effective combined miss rate : 3.2%

Q: Why instruction miss rate is lower than data miss rate?
A: Because instruction is much more sequential than data and the spatial location is good.

Q1: Why is data miss rate higher than instruction miss rate?

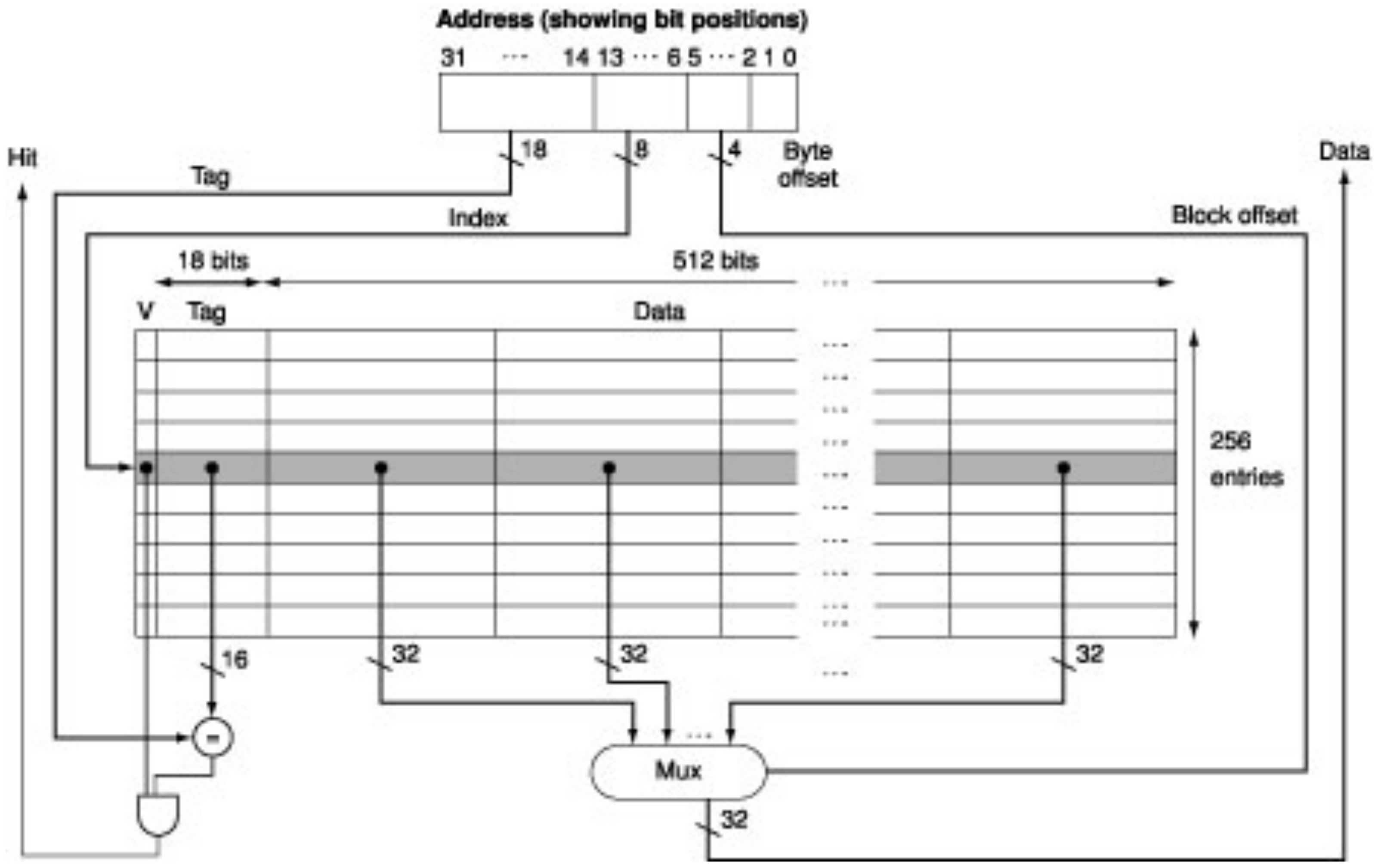
Q2: Why is combined miss rate is lower than data miss rate?

Split cache vs. Combined cache

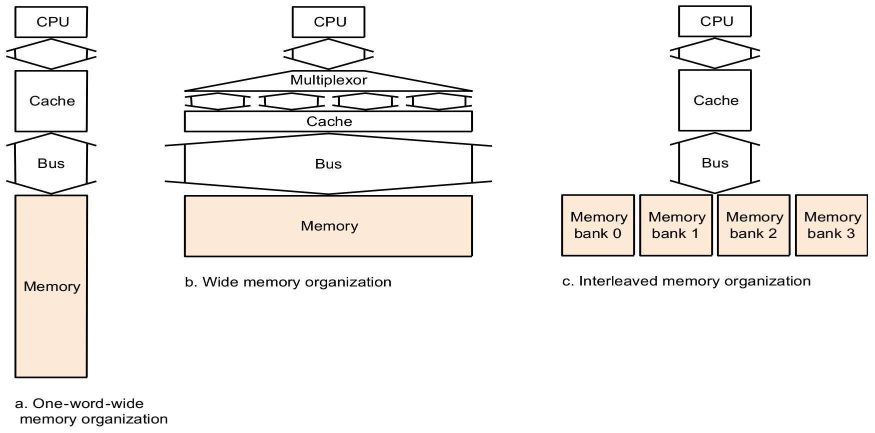
- Combined cache – higher cache hit rate & lower cache bandwidth
- Split cache – lower cache hit rate & higher cache bandwidth

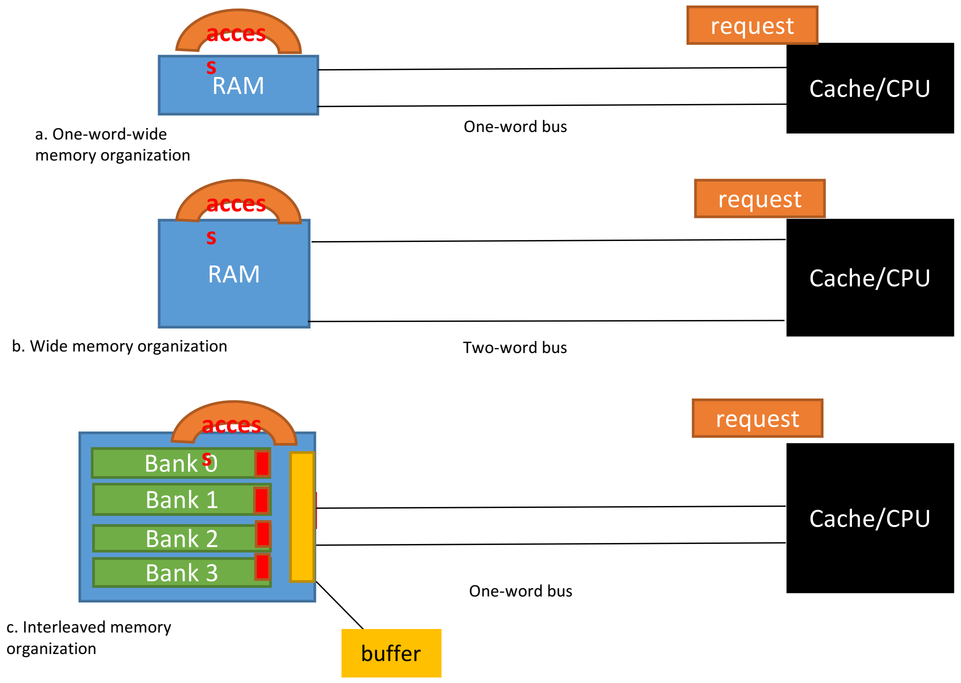
Example: Intrinsity FastMath Processor (cont.)

- The 16 KB caches in the Intrinsity FastMATH each contain 256 blocks with 16 words per block.



Memory Design to Support Cache

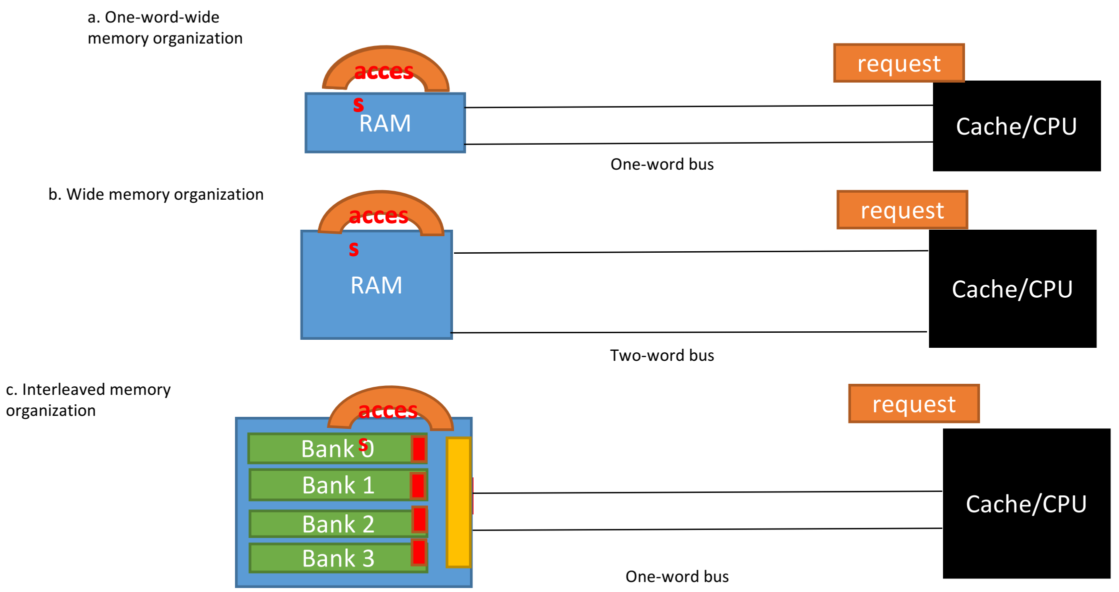




Assume

1 memory bus cycle to send the address
15 memory bus cycles for each DRAM access 15: assumption
1 memory bus cycle to send a word of data
4-word block & on-word-wide memory bank

What is the cache miss penalty?



Send request / Number of access DRAM / Number of send data back

1 + 4x15 + 4x1 = 65

讀4次word，每次花15個cycles

1 + 2x15 + 2x1 = 33

讀4words / 2words bus = 2次，
每次花15個cycles

1 + 1x15 + 4x1 = 20

Cache Performance

- CPU time = (CPU execution clock cycles + Memory stall clock cycles) x clock cycle time
- Memory stall clock cycles = Read-stall cycles + write-stall cycles
- Read-stall cycles = # of Read x Read miss rate X Read miss penalty
- Write-stall cycles = (# of Writes X Write miss rate X Write miss penalty) + Write buffer stalls.
- Memory-stall clock cycles = # Memory accesses X Miss rate X Miss penalty

$$= \frac{\text{Instructions}}{\text{program}} \times \frac{\text{Misses}}{\text{Instruction}} \times \text{Miss penalty}$$

- Average memory access time = Hit time + Miss rate x Miss penalty

Example

- I-Cache miss rate = 2% & D-Cache miss rate = 4%
- Base CPI 2.0
- Miss penalty = 100 cycles
- Frequency of loads and stores is 36%.
- Compare the performance with a perfect cache

- I-cache stall cycles = I x 2% x 100 = 2.00 X I
- D-cache stall cycles = I X 36% X 4% X 100 = 1.44 X I
- CPU time with stalls = I X (2 + 1.44 + 2) X Clock-cycle
  = I X 5.44 X Clock-cycle
- CPU time with perfect caches
  = I X CPI $_{perfect}$ X Clock-cycle
  = I X 2 X Clock-cycle

單位(s)

CPU time = cycles x clock-cycle
= (CPU cycles + memory stall cycles) x clock-cycle

Cache Performance with Increased Clock rate

clock rate 2倍快: new clock rate = 2 old clock rate
new clock cycles = 2 old clock cycles

- How much faster will the computer be with 2x clock rate, assuming the same miss rate as the previous example.

Stall time remains same!

- I-cache stall cycles = I x 2% x 200 = 4.00 X I
- D-cache stall cycles = I X 36% X 4% X 200 = 2.88 X I
- CPU time with stalls = I X (4 + 2.88 + 2) X Clock-cycle

  = I X 8.88 X Clock-cycle

$$\frac{\textit{Performance with fast clock}}{\textit{Performance with slow clock}} = \frac{\textit{Execution time with slow clock}}{\textit{Execution time with fast clock}}$$

$$= \frac{IC \times CPI_{slow\ clock} \times Clock\ Cycle}{IC \times CPI_{fast\ clock} \times \frac{Clock\ Cycle}{2}}$$

$$= \frac{5.44}{8.88 \times \frac{1}{2}} = 1.23$$

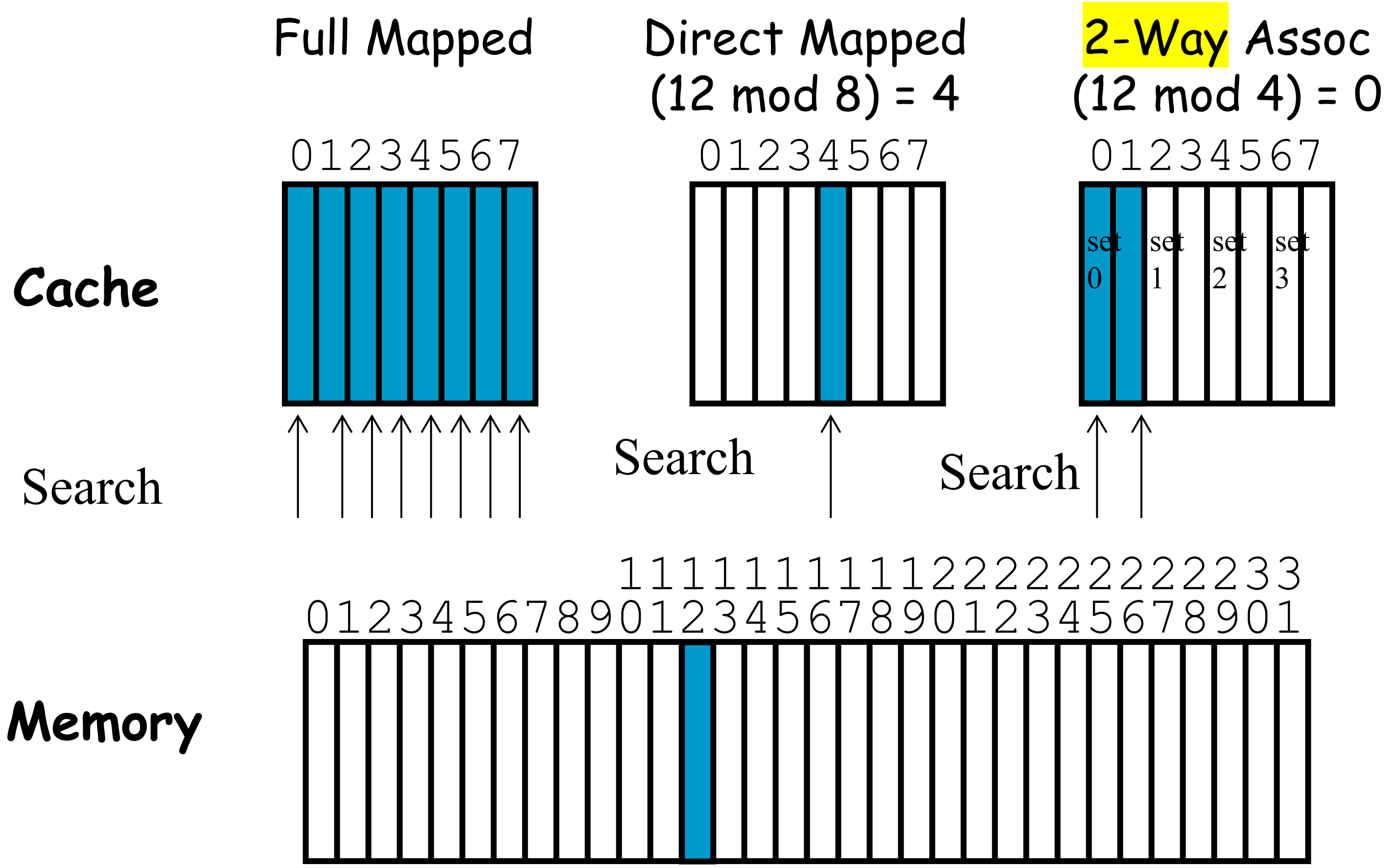
How to Improve Cache Performance?

1. Reduce miss rate -> Increasing associativity
2. Reduce miss penalty -> multi-level cache
3. Reduce hit time -> small cache

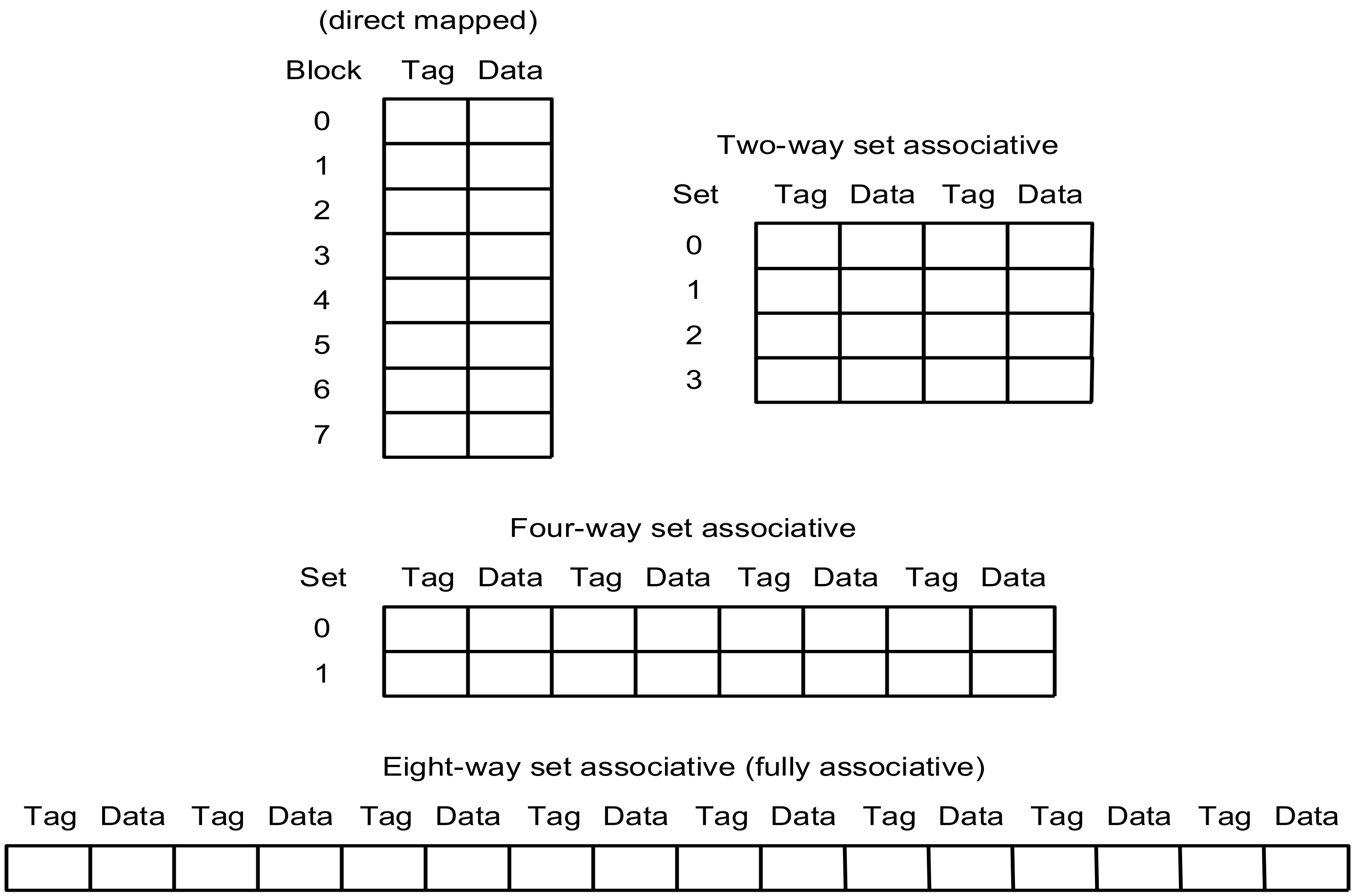
Average memory access time = hit time + miss-rate X miss-penalty

Reducing Cache Misses by More Flexible Placement of Blocks

if 4-Way, there are 2 sets and each with 4 blocks



Possible Associativity Structures


(direct mapped)
Block
Tag
Data
0
1
2
3
4
5
6
7
Two-way set associative
Set
Tag Data Tag Data
0
1
2
3
Four-way set associative
Set
Tag Data Tag Data Tag Data Tag Data
0
1
Eight-way set associative (fully associative)
Tag Data Tag Data Tag Data Tag Data Tag Data Tag Data Tag Data Tag Data

An 8-block cache

Address Conflict

```
float dot_prod(float x[SIZE],
               float y[SIZE])
{
  float sum = 0.0;
  int   i;

  for (i = 0; i < SIZE; i++)
    sum += x[i]*y[i];

  return sum;
}
```

Assume:

Direct-mapped cache.

`x[i]` and **`y[i]`** map to same blocks.

? What is the hit rate? ?

Under these assumptions, every access is a cache miss.
Hit rate = 0%.

What can we do? Increasing the associativity

Exercise

- Three small caches, each consisting of four one-word blocks
  - Direct mapped cache
  - Two-way set associative cache
  - Fully associative cache
- Find the number of misses for each cache for the following sequence
  - 0,8,0,6,8

Exercise (cont.)

- The direct mapped cache

Block Address	Cache Set
0	(0 modulo 4) = 0
6	(6 modulo 4) = 2
8	(8 modulo 4) = 0

Address of memory block accessed	Hit or miss	Contents of cache blocks after reference			
		0	1	2	3
0	Miss	Memory[0]			
8	Miss	Memory[8]			
0	Miss	Memory[0]			
6	Miss	Memory[0]		Memory[6]	
8	miss	Memory[8]		Memory[6]	

Exercise (cont.)

- The two-way set associative cache

Block Address	Cache Set
0	(0 modulo 2) = 0
6	(6 modulo 2) = 0
8	(8 modulo 2) = 0

Address of memory block accessed	Hit or miss	Contents of cache blocks after reference			
		Set 0	Set 0	Set 1	Set 1
0	Miss	Memory[0]			
8	Miss	Memory[0]	Memory[8]		
0	Hit	Memory[0]	Memory[8]		
6	Miss	Memory[0]	Memory[6]		
8	miss	Memory[8]	Memory[6]		

replacement policy: 6進來時，應該要踢掉誰？

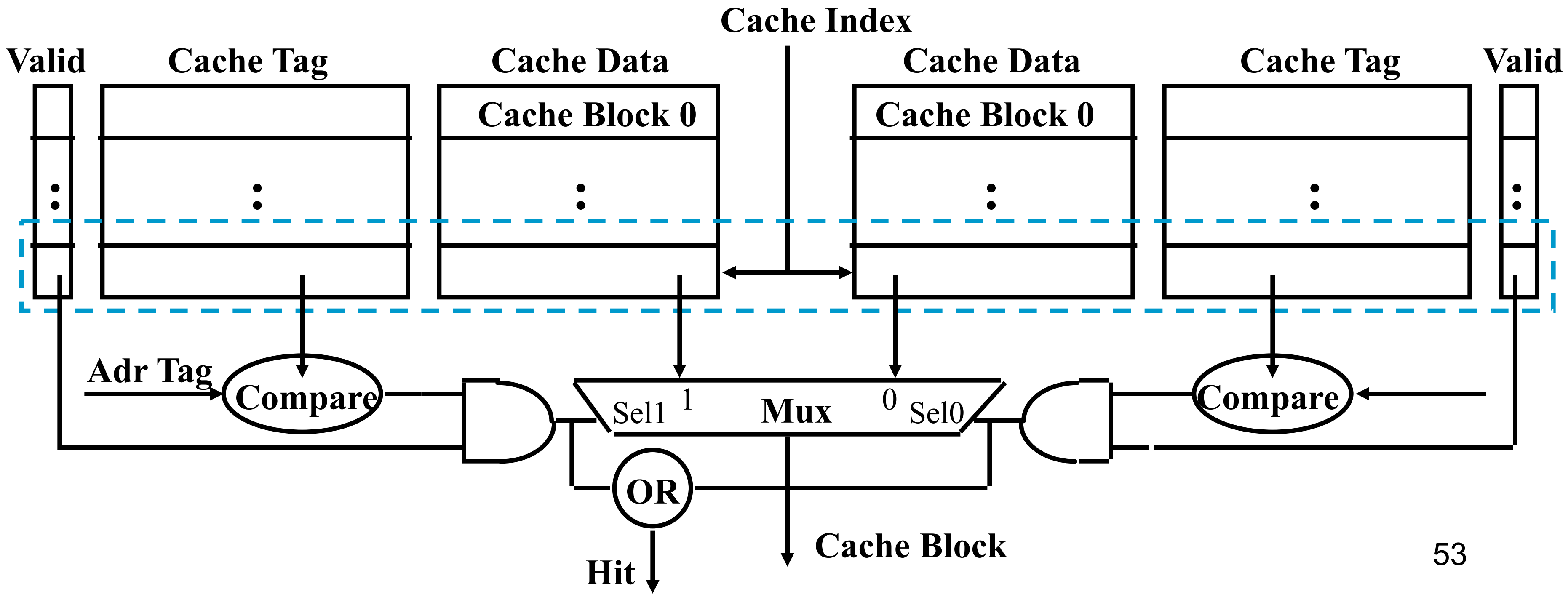
Exercise (cont.)

- The fully associative cache

Address of memory block accessed	Hit or miss	Contents of cache blocks after reference			
		Block 0	Block 1	Block 2	Block 3
0	Miss	Memory[0]			
8	Miss	Memory[0]	Memory[8]		
0	Hit	Memory[0]	Memory[8]		
6	Miss	Memory[0]	Memory[8]	Memory[6]	
8	Hit	Memory[0]	Memory[8]	Memory[6]	

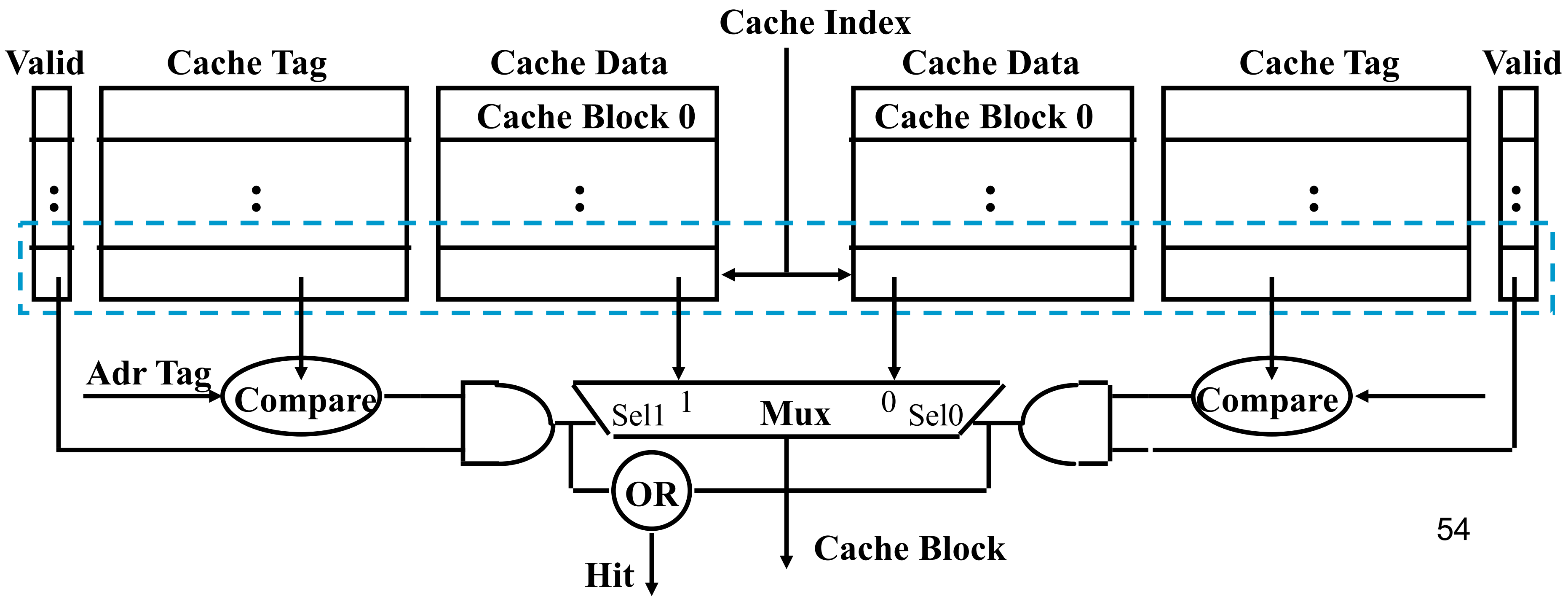
2-way Set Associative Cache

- 1K, 2-way,32B block
  - 16 sets, each set has 2 blocks
  - Index = 4 bits, tag = 23 bits
- Increasing associativity shrinks index, expands tag
- How to find a data in 2-way 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

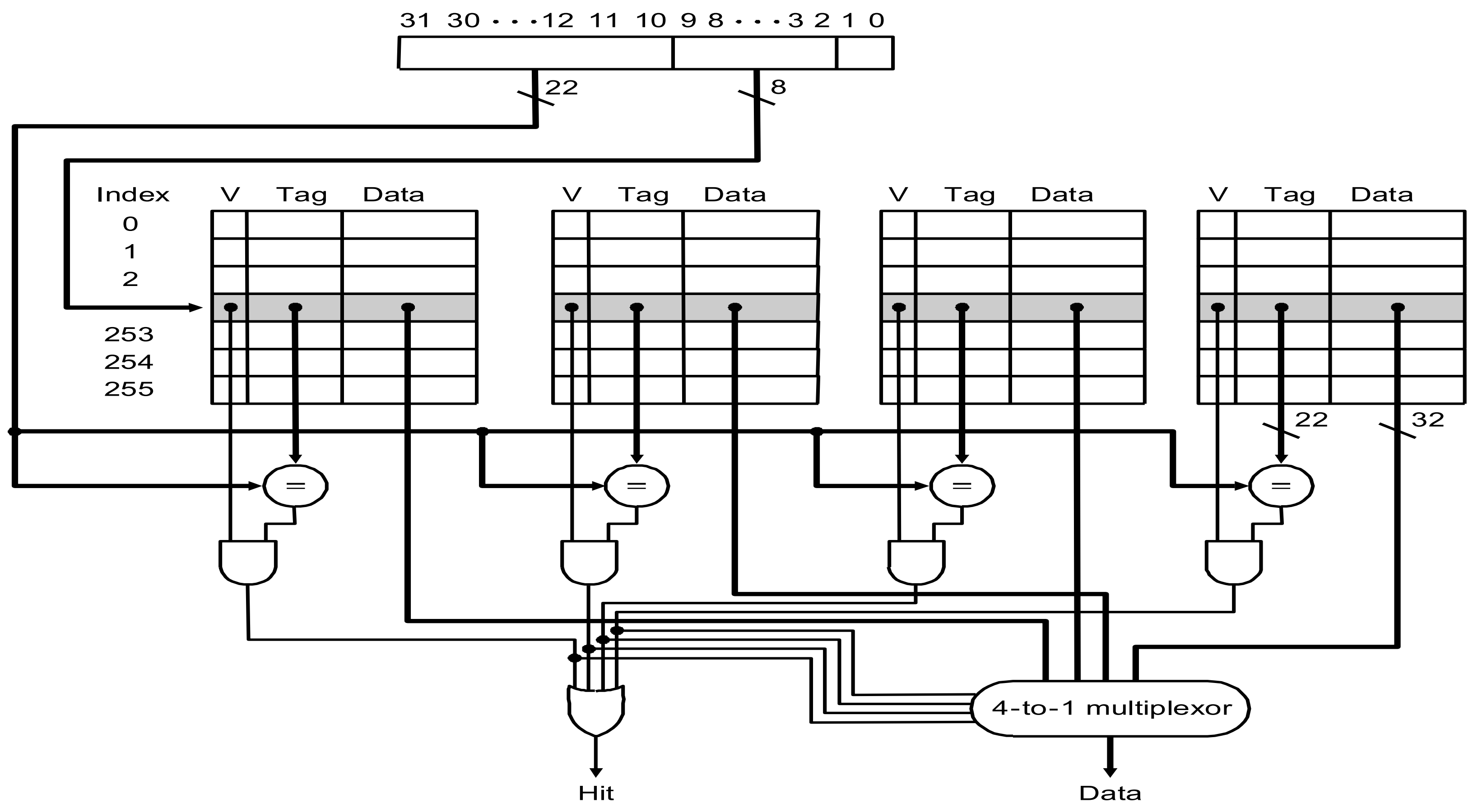


Disadvantage of Set Associative Cache

- N-way Set Associative Cache vs. 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.



A 4-Way Set-Associative Cache



Replacement Policy: Choosing Which Block to Replace

- Easy for direct mapped
- Set associative or fully associative:
  - Random
  - FIFO
  - LRU (Least Recently Used):
    - Hardware keeps track of the access history and replace the block that has not been used for the longest time

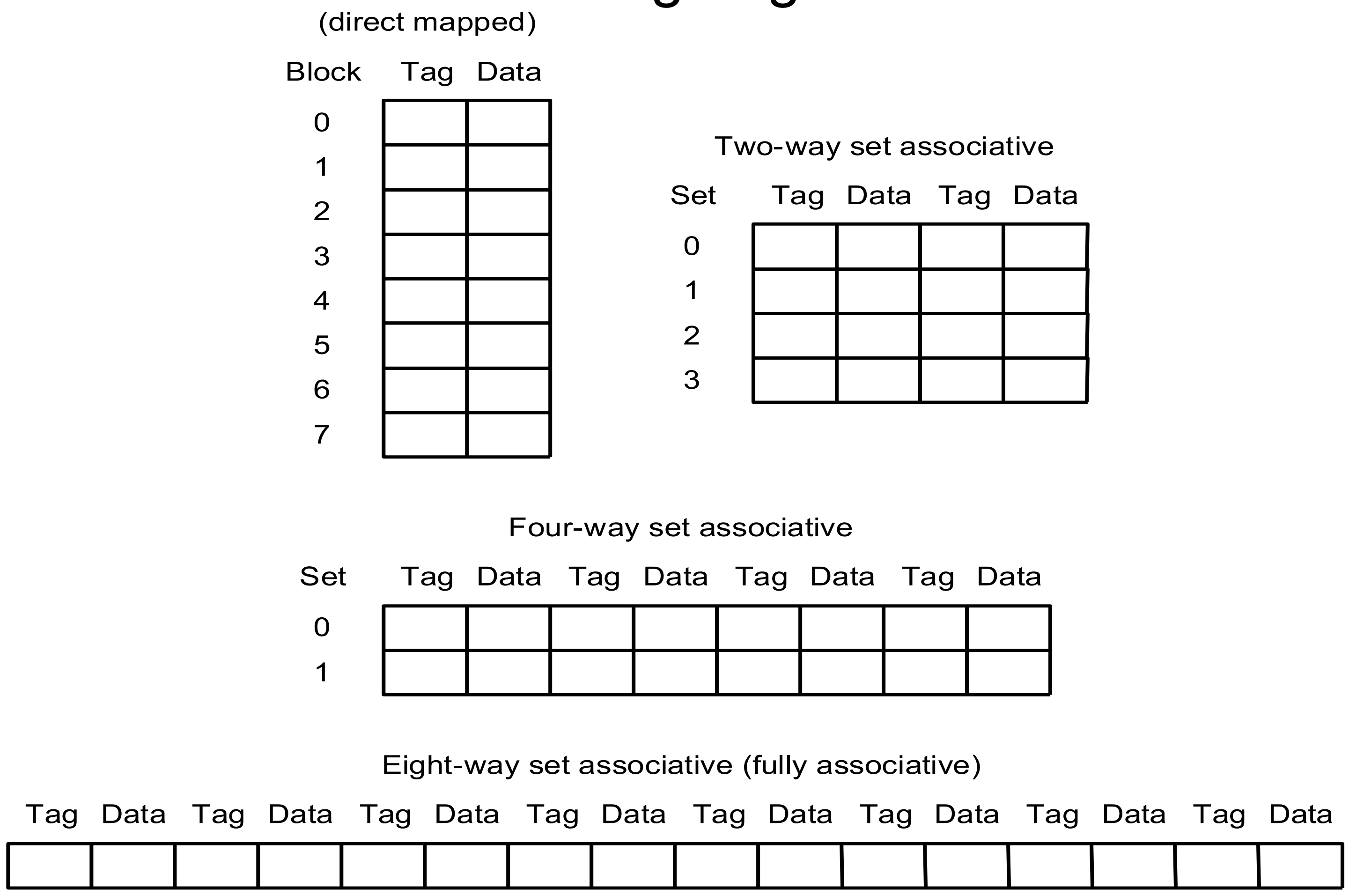
Effects of Associativity

Associativity	Data miss rate
1 direct-map	10.3%
2 way	8.6%
4	8.3%
8	8.1%

Data Source: Spec2000 benchmarks

Tag Size vs. Associtivity

- With the same cache capacity, increasing associtivity increases or decreasing tag bits?

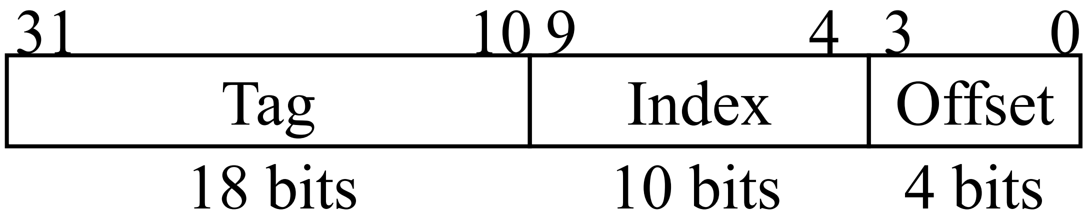

(direct mapped)
Block Tag Data
0 1 2 3 4 5 6 7
Two-way set associative
Set Tag Data Tag Data
0 1 2 3
Four-way set associative
Set Tag Data Tag Data Tag Data Tag Data
0 1
Eight-way set associative (fully associative)
Tag Data Tag Data Tag Data Tag Data Tag Data Tag Data Tag Data Tag Data

Exercise: Size of Tags vs. Set Associativity

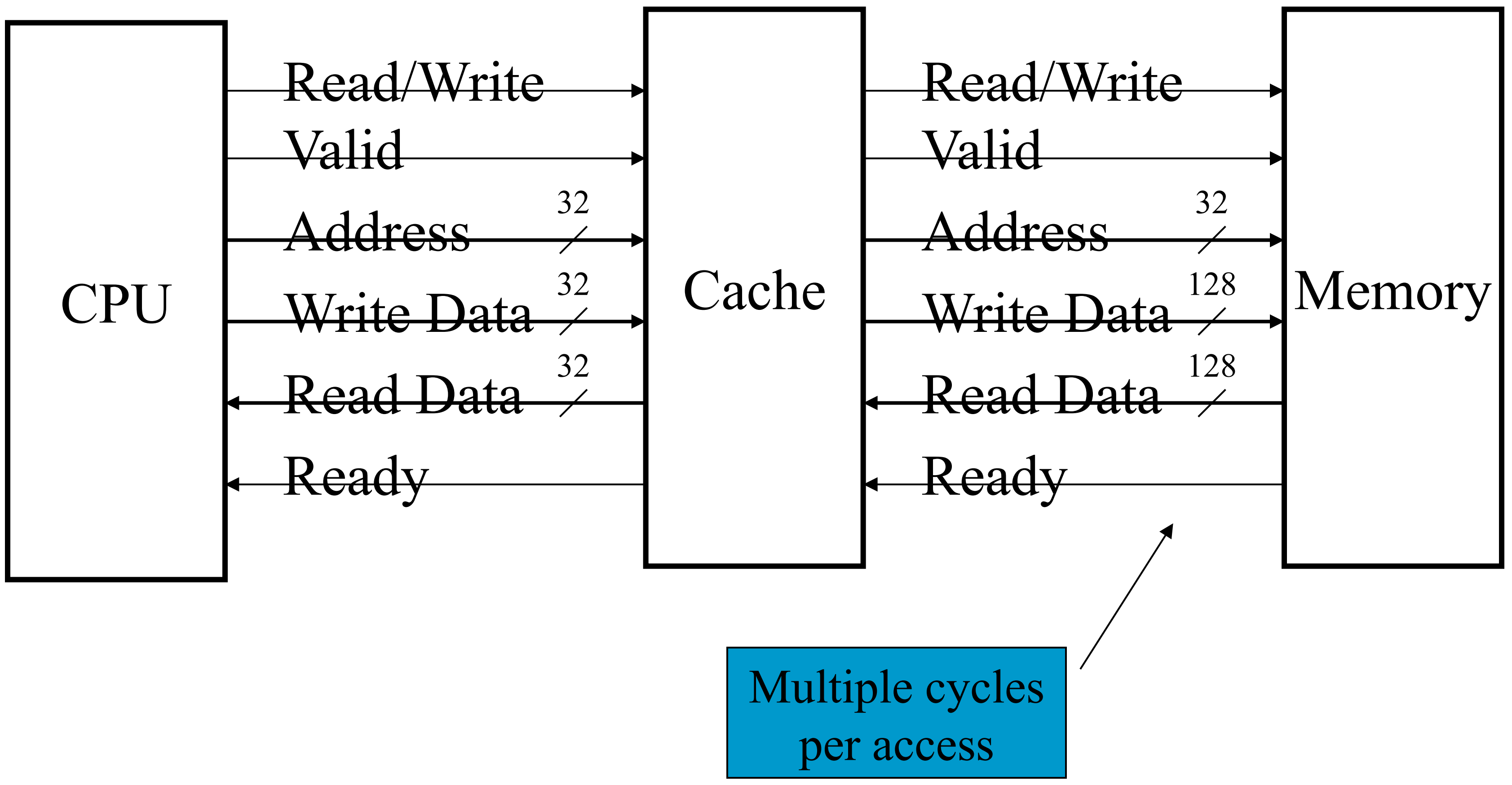
- Find # of sets, total # of tag bits for direct-mapped, two-way and four way. (Capacity- 4K blocks, four-word block, 32 bit address)
  - tag+index = 32- 4 = 28
  - The direct mapped cache
    - 4K sets
    - $\log_2(4K)$ = 12 bits of index
    - Total (28-12) $\times$ 4K = 16 $\times$ 4K = 64 Kbits of tag
  - The two-way set-associative cache
    - 2K sets
    - Total (28-11) $\times$ 2 $\times$ 2K = 34 $\times$ 2K = 68 Kbits of tag
  - The four-way set-associative cache
    - 1K sets
    - Total (28-10) $\times$ 4 $\times$ 1K = 72 $\times$ 1K = 72 Kbits of tag
  - The fully associative cache
    - One set with 4K blocks
    - Tag is 28 bits
    - Total 28 $\times$ 4K $\times$ 1 = 112K tag bits

Cache Control

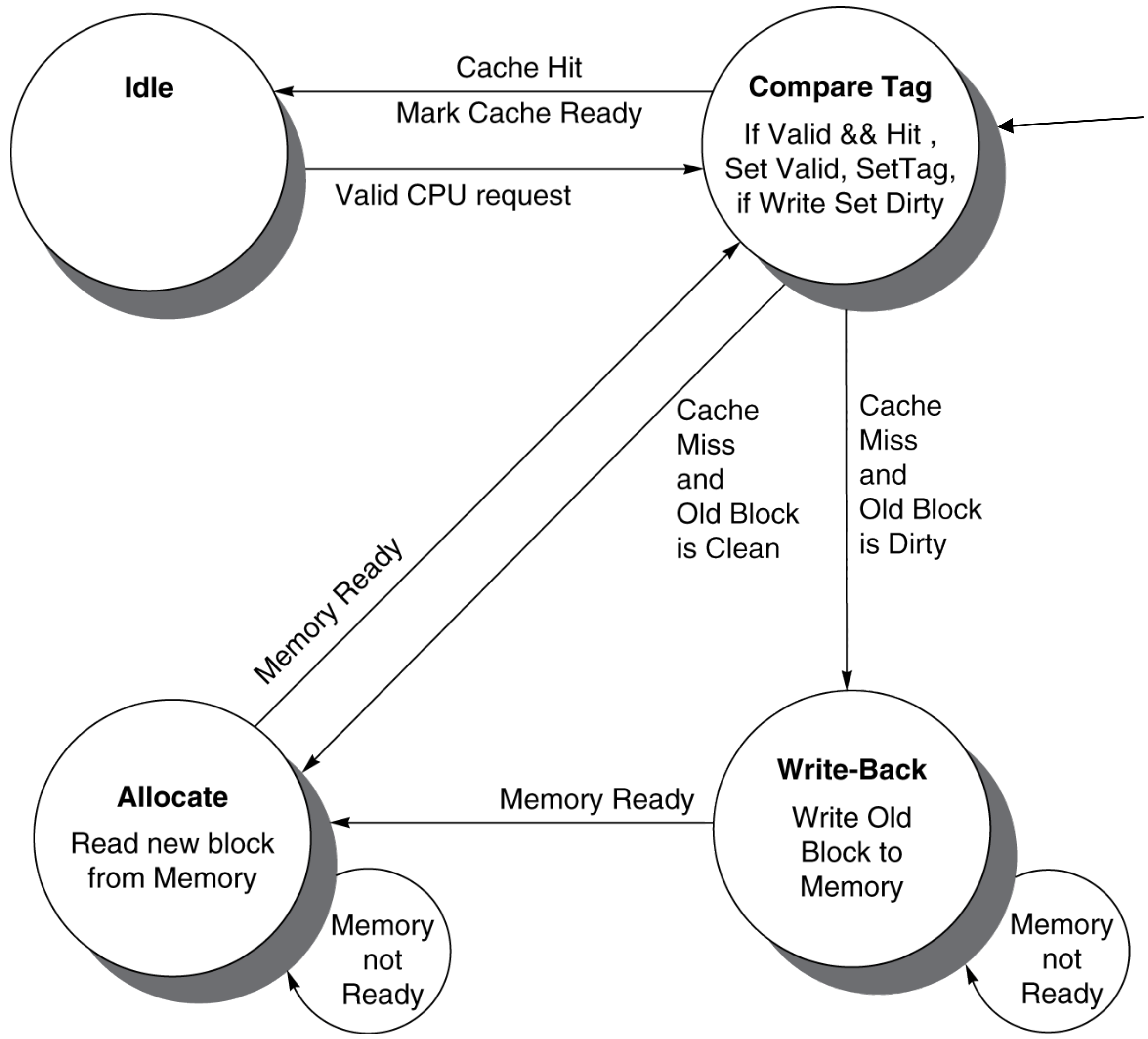
- Example cache characteristics
  - Direct-mapped, write-back, write allocate
  - Block size: 4 words (16 bytes)
  - Cache size: 16 KB (1024 blocks)
  - 32-bit byte addresses
  - Valid bit and dirty bit per block
  - Blocking cache 遇到cache miss就stall
    - CPU waits until access is complete



Interface Signals



Cache Controller FSM

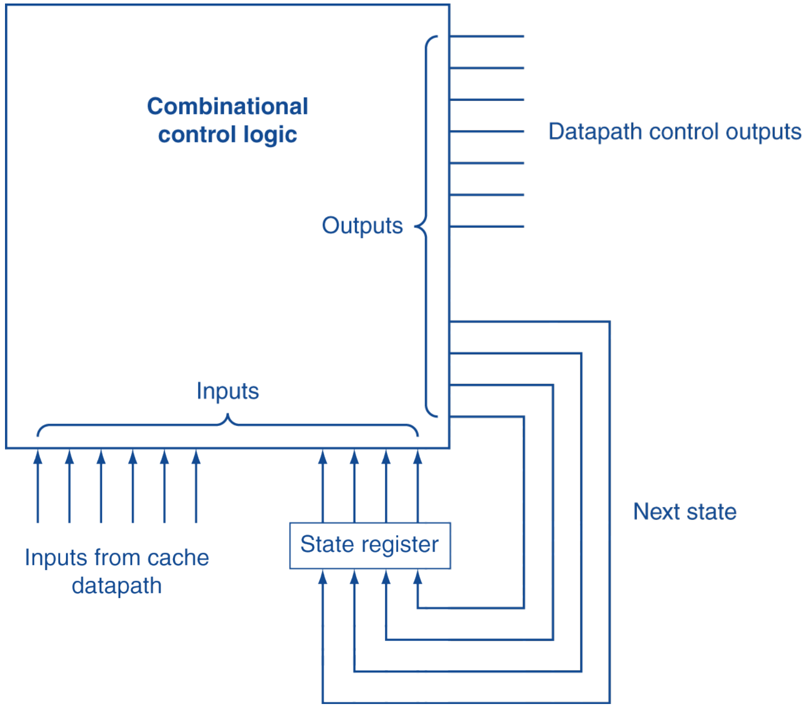


Idle
Cache Hit
Mark Cache Ready
Valid CPU request
Compare Tag
If Valid && Hit , Set Valid, SetTag, if Write Set Dirty
Cache Miss and Old Block is Clean
Cache Miss and Old Block is Dirty
Memory Ready
Allocate
Read new block from Memory
Memory Ready
Write-Back
Write Old Block to Memory
Memory not Ready
Memory not Ready

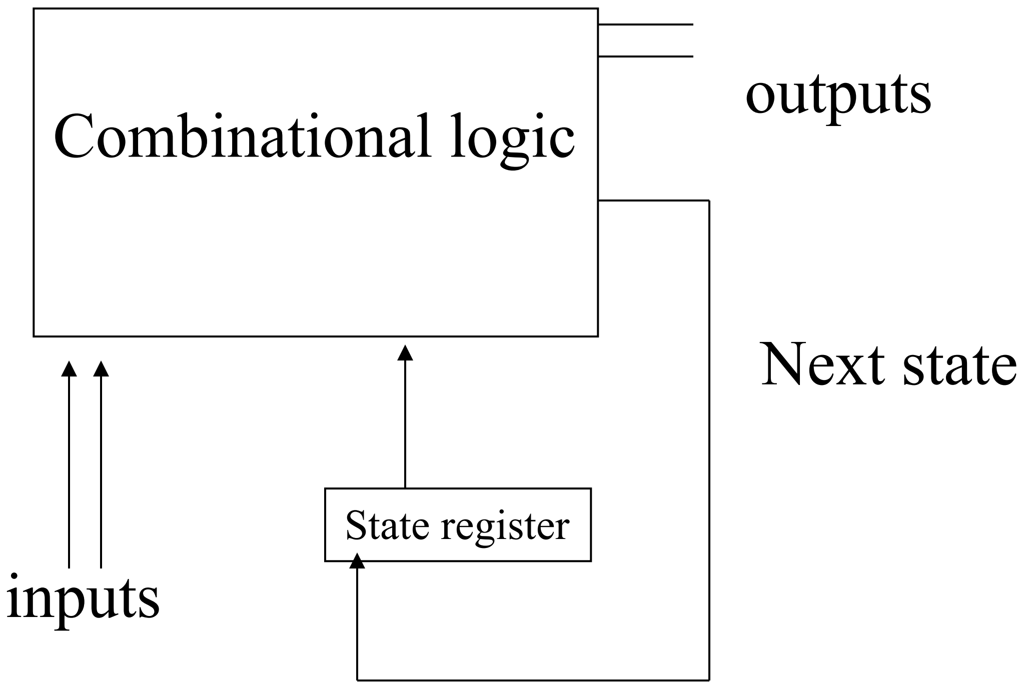
Could partition into separate states to reduce clock cycle time

Finite State Machines

- Use a FSM to sequence control steps
- Set of states, transition on each clock edge
  - State values are binary encoded
  - Current state stored in a register
  - Next state $= f_n$ (current state, current inputs)
- Control output signals $= f_o$ (current state)



Finite State Implementation



How to implement the combinational logic?

- PLA (Programmable Logic Array)
- ROMs

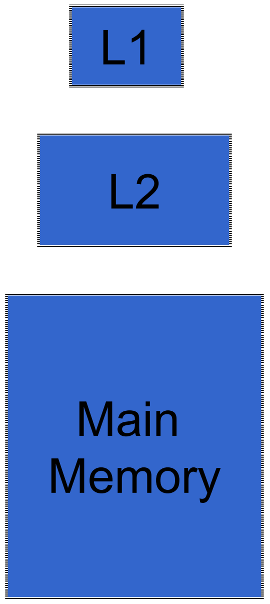
Miss Penalty Reduction: Multi- Level Cache

- Larger cache vs. CPU time
  - Add another level of cache
  - The L2 cache is much larger than L1
- L2 Equations

  $\text{AMAT} = \text{Hit Time}_{L1} + \text{Miss Rate}_{L1} \times \text{Miss Penalty}_{L1}$

  $\text{Miss Penalty}_{L1} = \text{Hit Time}_{L2} + \text{Miss Rate}_{L2} \times \text{Miss Penalty}_{L2}$

  $\text{AMAT} = \text{Hit Time}_{L1} + \text{Miss Rate}_{L1} \times (\text{Hit Time}_{L2} + \text{Miss Rate}_{L2} \times \text{Miss Penalty}_{L2})$
- Definitions:
  - *Local miss rate*— misses in this cache divided by the total number of memory accesses *to this cache* (Miss rate_{L2})
  - *Global miss rate*—misses in this cache divided by the total number of memory accesses *generated by the CPU* ($\text{Miss Rate}_{L1} \times \text{Miss Rate}_{L2}$)



Performance of Multilevel Caches

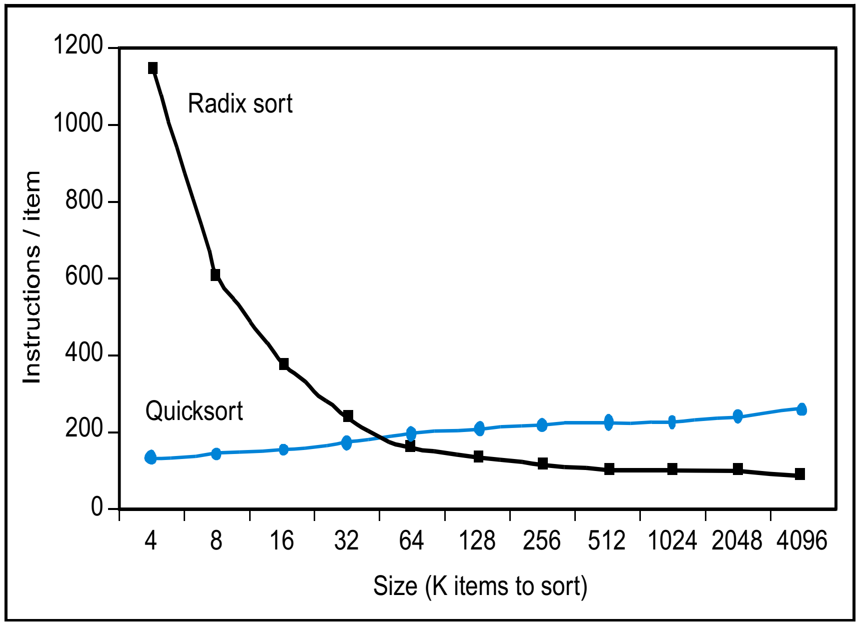
- Suppose we have a processor with the following parameters:
  - Base CPI=1.0, if hit in the L1 cache. L1 cache miss rate is 2%. Clock rate is 5 GHz. Memory access time is 100 ns, including all the miss handling.
  - Miss penalty to main memory is 100ns/0.2 = 500 clock cycle.
  - Total CPI = 1.0 + 2%$\times$500 = 11.0
- If we add a L2 cache that has 5 ns access time. L2 global miss rate = 0.5%
  - Miss penalty to L2 is 5ns/0.2 = 25 clock cycle
  - Total CPI = 1.0+ 2%$\times$25 + 0.5%$\times$500 = 4.0

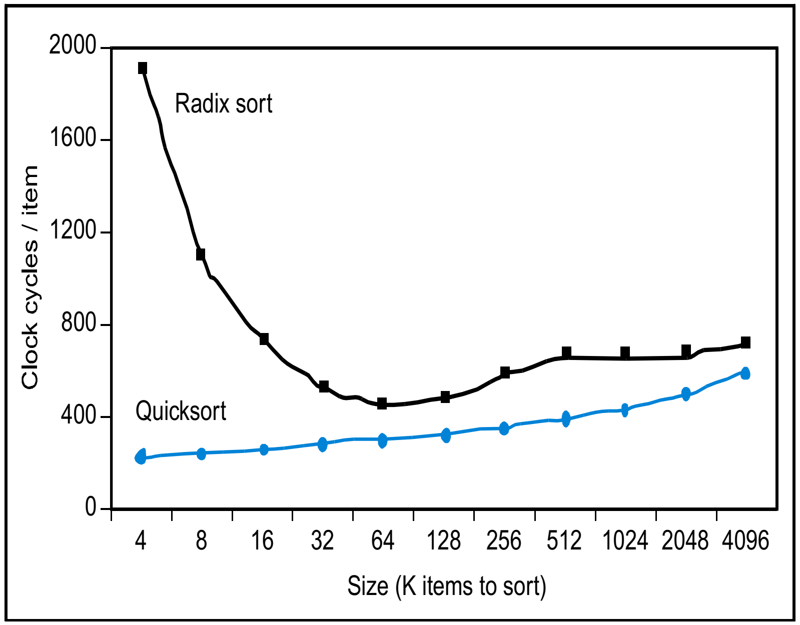
L2 Cache Design Principle

- L2 not tied to CPU clock cycle
  - Different design consideration from L1
  - Hits are less important than misses
  - Larger cache, higher associativity and larger blocks

Caches vs. Performance

included memory performance!

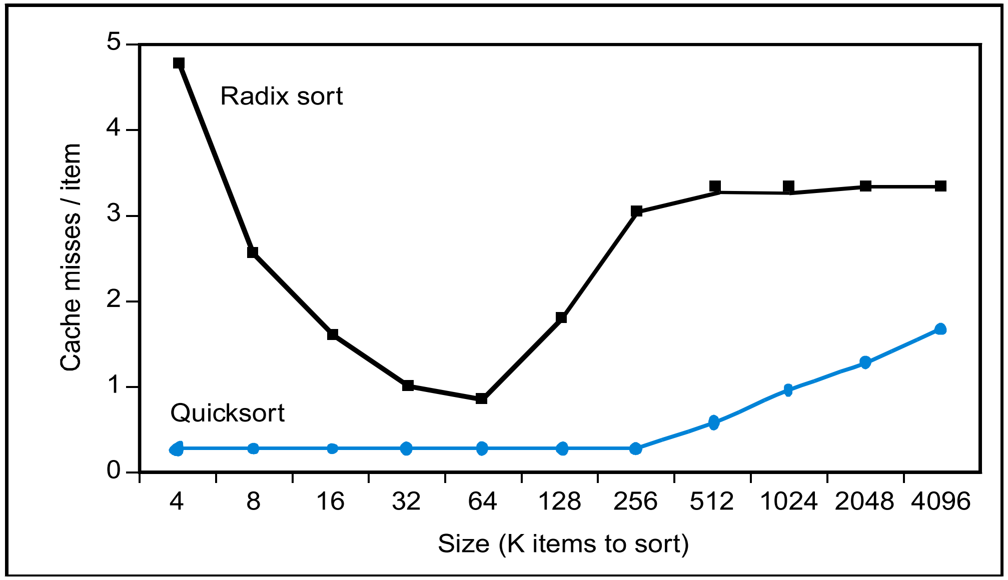




Theoretical behavior of Radix sort vs. Quicksort (instruction/item)

Observed behavior of Radix sort vs. Quicksort (clock cycles/item)

Caches vs. Performance (cont.)



Cache behavior
Radix sort vs. Quicksort
(cache misses /item)

- Memory system performance is often critical factor
  - Multilevel caches, pipelined processors, make it harder to predict outcomes
  - Need experimental data