Computer Organization

Lecture 22 - Memory Hierarchy 2

Reading: 5.3-5.4 (cache), ~~5.7 (virtual memory)~~

Homework :

Outline - Memory Systems

- Overview
  - Motivation
  - General Structure and Terminology
- Memory Technology
  - Static RAM
  - Dynamic RAM
  - Disks
- Cache Memory ◀
- Virtual Memory

Four Key Cache Questions:

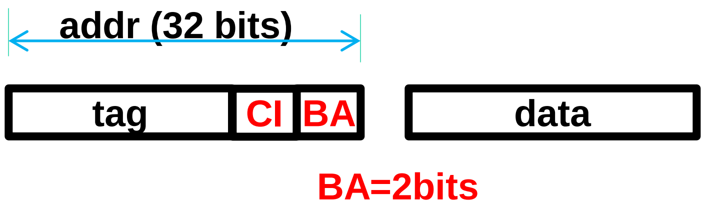
1. Where can block be placed in cache?
   (block placement)
2. How can block be found in cache? …using a tag
   (block identification)
3. Which block should be replaced on a miss?
   (block replacement)
4. What happens on a write?
   (write strategy)

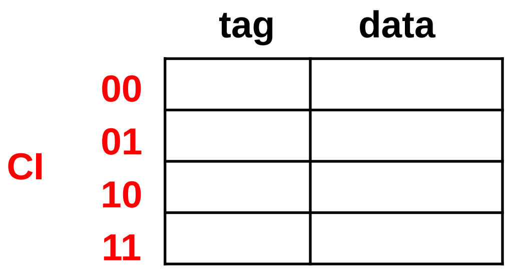
Q1: Block Placement

- Where can block be placed in cache?
  - In one predetermined place - direct-mapped
    - Use fragment of address to calculate block location in cache
    - Compare cache block with tag to test if block is present
  - Anywhere in cache - fully associative
    - Compare tag to every block in cache
  - In a limited set of places - set-associative
    - Use address fragment to calculate set (like direct-mapped)
    - Place in any block in the set
    - Compare tag to every block in set
    - Hybrid of direct mapped and fully associative

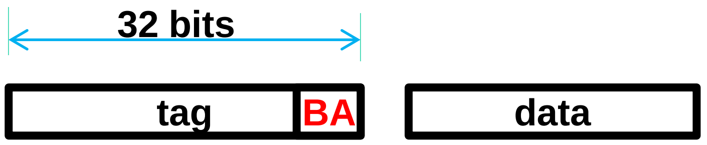
Concept

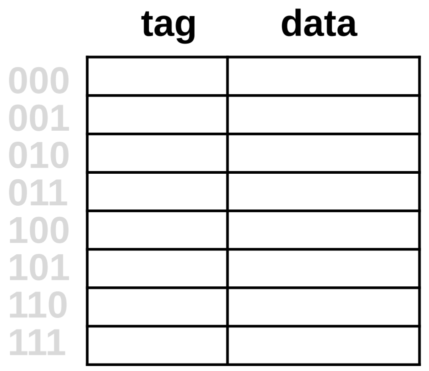
(1) Direct-mapped



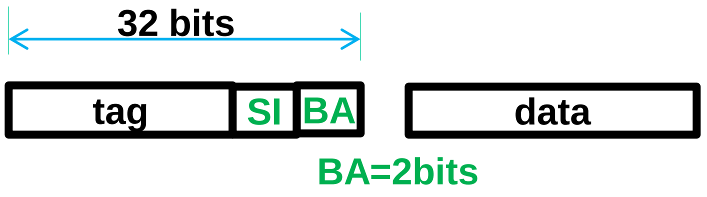


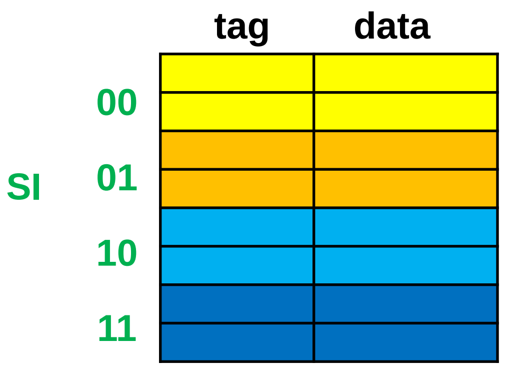
(2) Fully associative



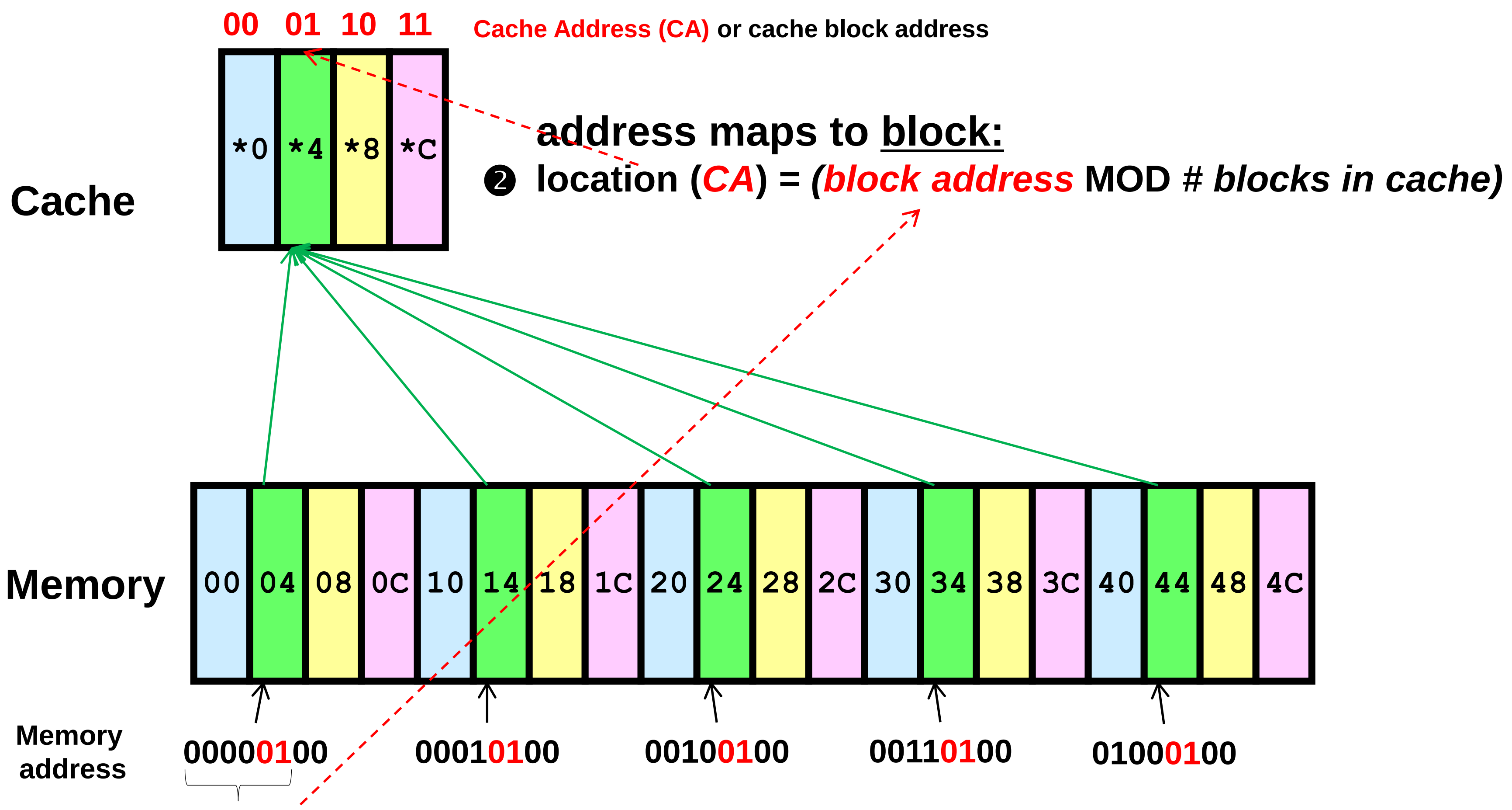


(3) Set-associative





Direct Mapped Block Placement

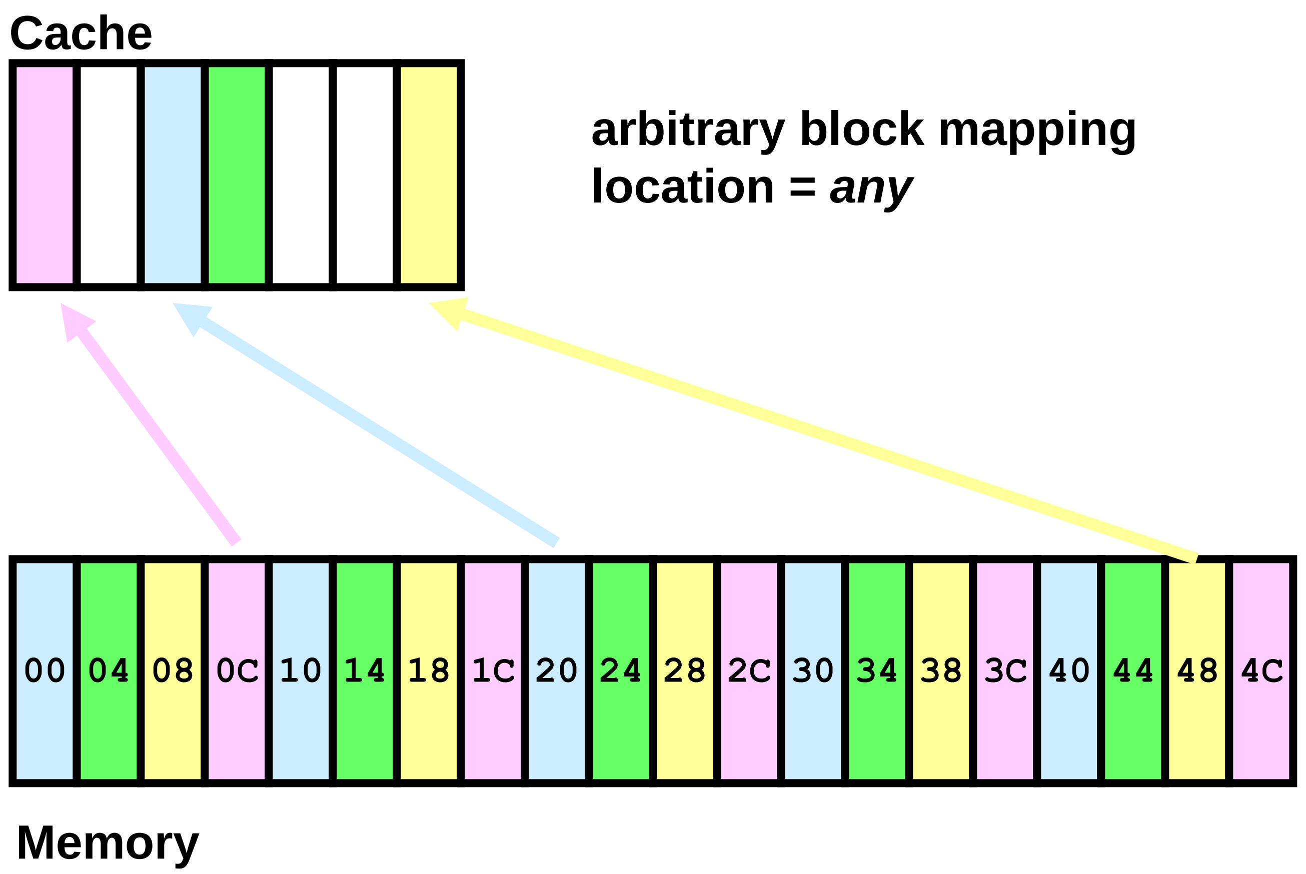


address maps to block:

❷ location (*CA*) = (***block address*** MOD # ***blocks in cache)***

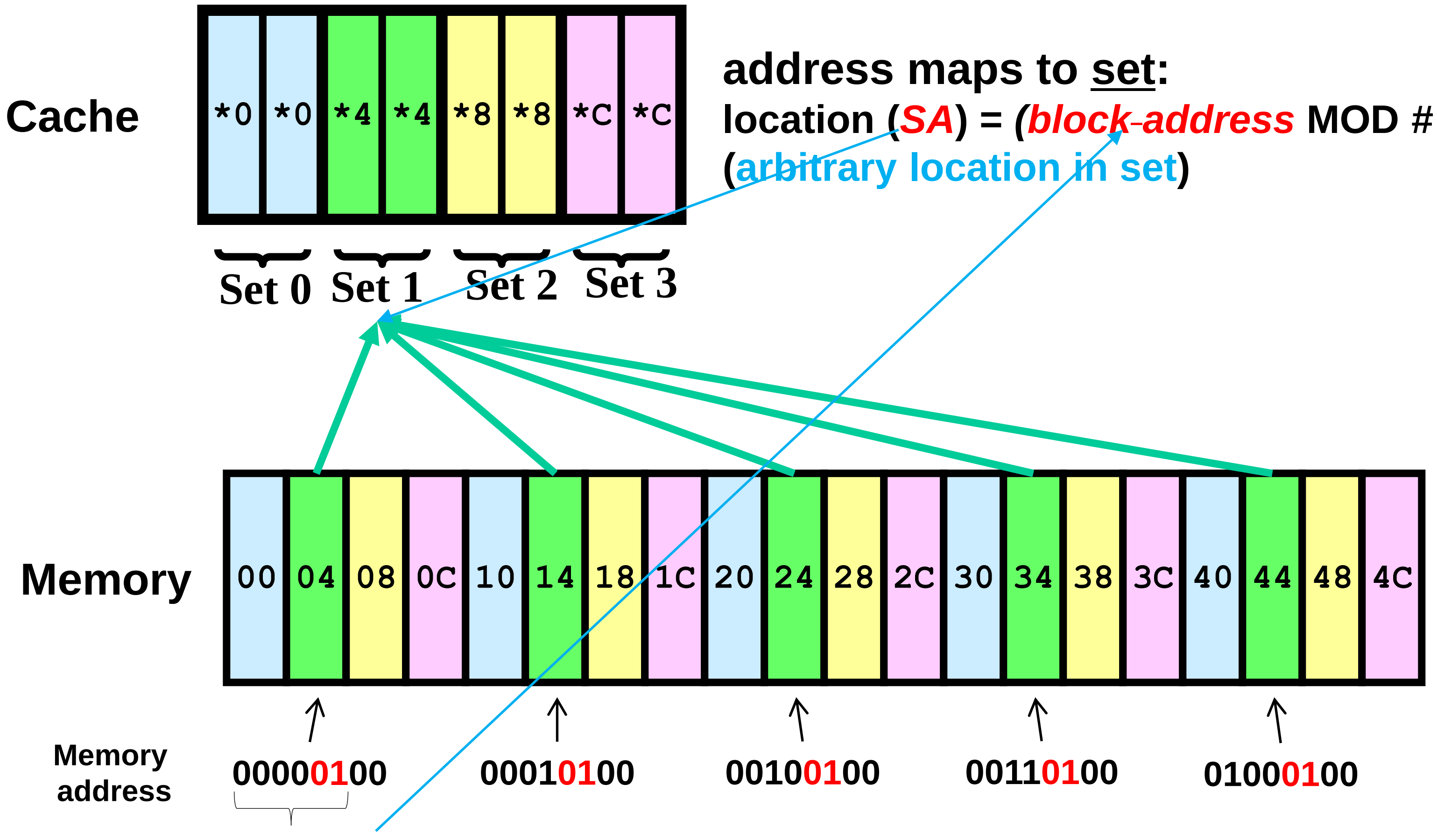
❶ ***block address* = *address / # bytes per block***

Fully Associative Block Placement



arbitrary block mapping
location = *any*

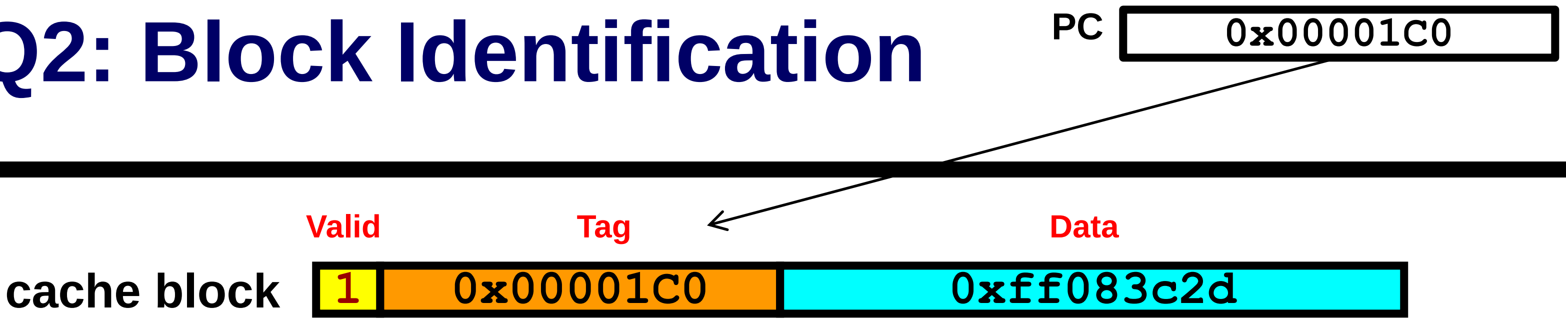
Set-Associative Block Placement



address maps to <u>set</u>:
location (*SA*) = (*block-address* MOD # *<u>sets</u> in cache*)
(arbitrary location in set)

block address* = *address / # bytes per block

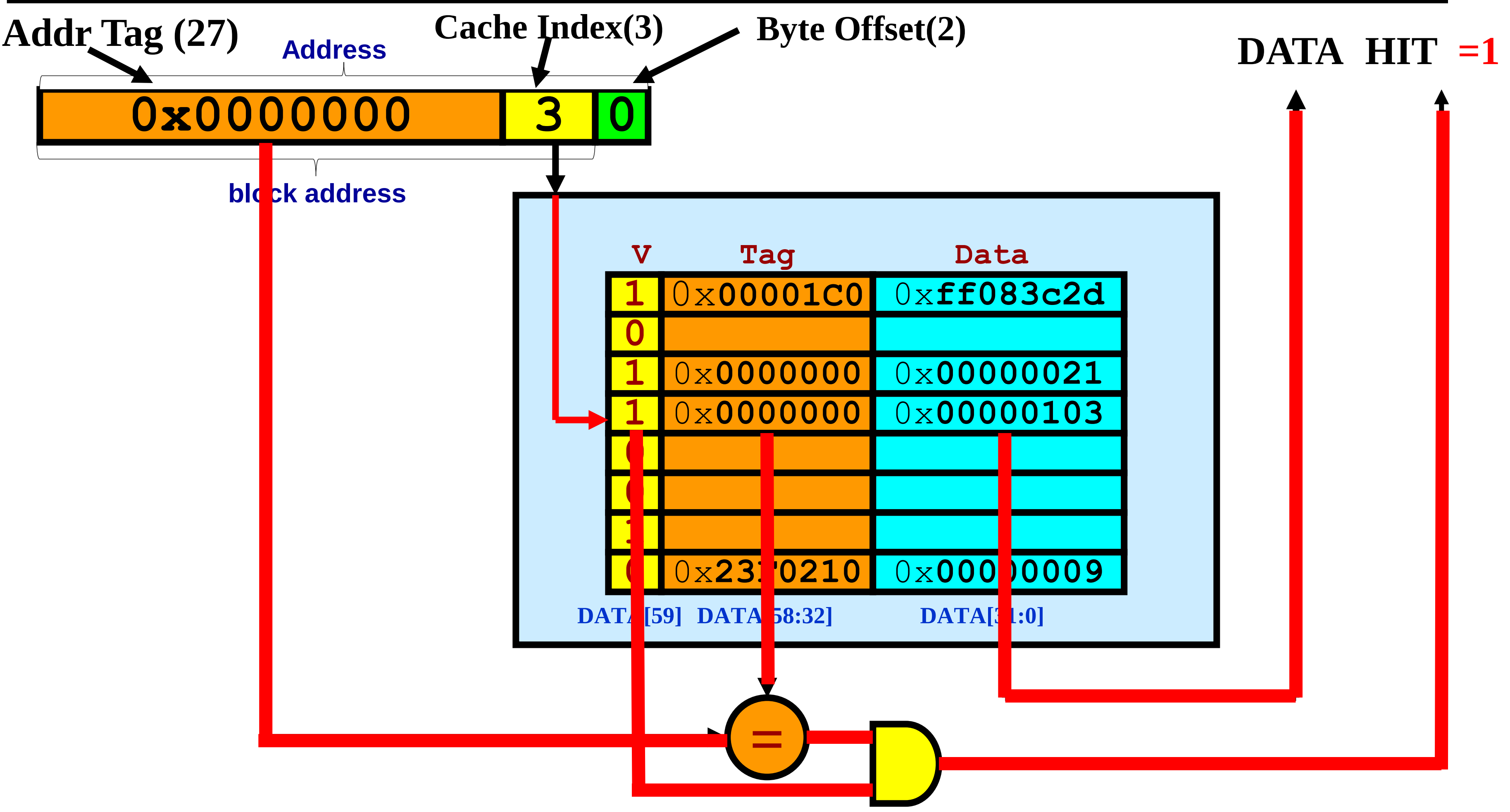
Q2: Block Identification



- Every cache block has an address tag that identifies its location in memory
- Hit when tag and address of desired word match (comparison by hardware)
- Q: What happens when a cache block is empty?
  A: Mark this condition with a valid bit

  (vaild => 1, invalid => 0)

Direct-Mapped Cache Design



Cache Example

- 8-blocks, 1 word/block, direct mapped
- Initial state

	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	N		
111	N		

Cache Example

Block address (b)	Hit/miss
10 110	Miss

Cache block address

	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Cache Example

Block address (b)	Hit/miss
11 010	Miss

	V	Tag	Data
000	N		
001	N		
010	Y	11	Mem[11010]
011	N		
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Cache Example

Block address (b)	Hit/miss
10 110	Hit
11 010	Hit

	V	Tag	Data
000	N		
001	N		
010	Y	11	Mem[11010]
011	N		
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Cache Example

Block address (b)	Hit/miss
10 000	Miss
00 011	Miss
10 000	Hit

	V	Tag	Data
000	Y	10	Mem[10000]
001	N		
010	Y	11	Mem[11010]
011	Y	00	Mem[00011]
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Cache Example

Block address (b)	Hit/miss
10 010	Miss

	V	Tag	Data
000	Y	10	Mem[10000]
001	N		
010	Y	10	Mem[10010]
011	Y	00	Mem[00011]
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Example: Larger Direct-Mapped Cache

Cache Size = 1024 Words = 4KB
Byte Offset = 2 bits
Cache Index - 10 bits
Tag - 20 bits

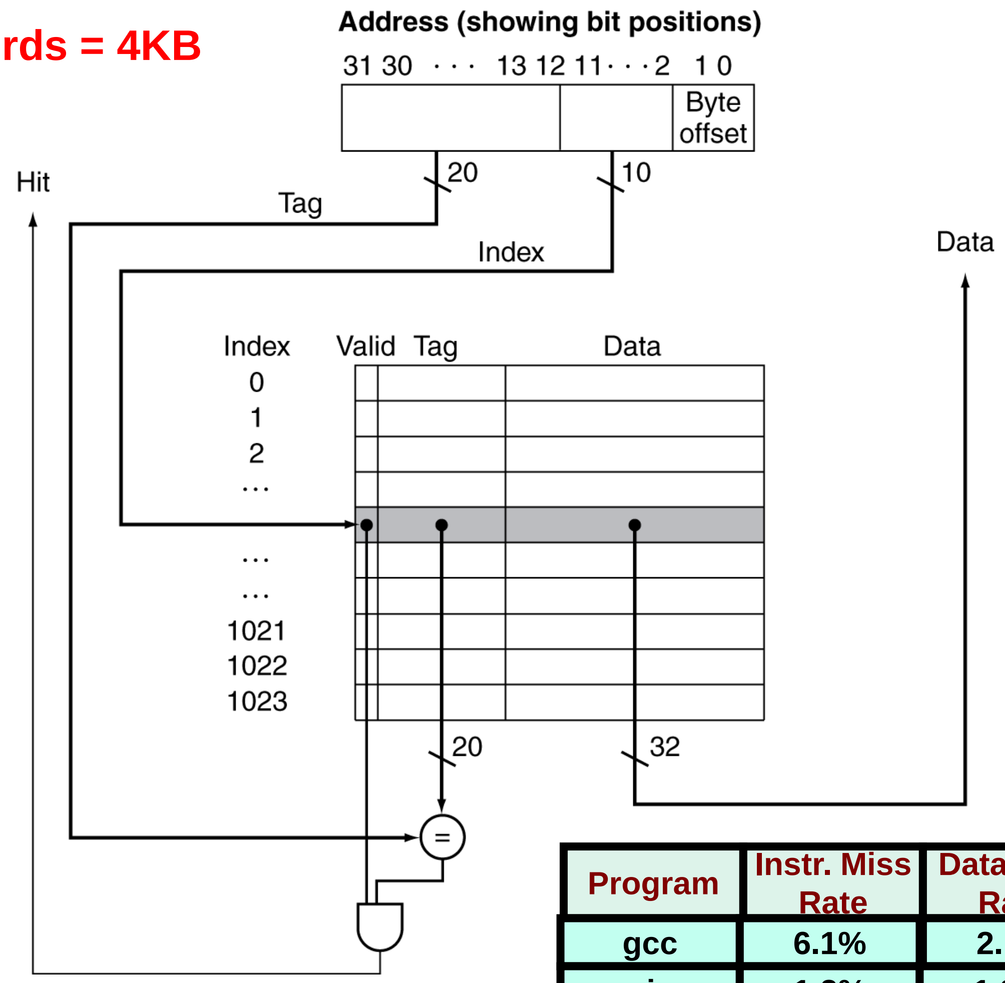


Fig. 5.7

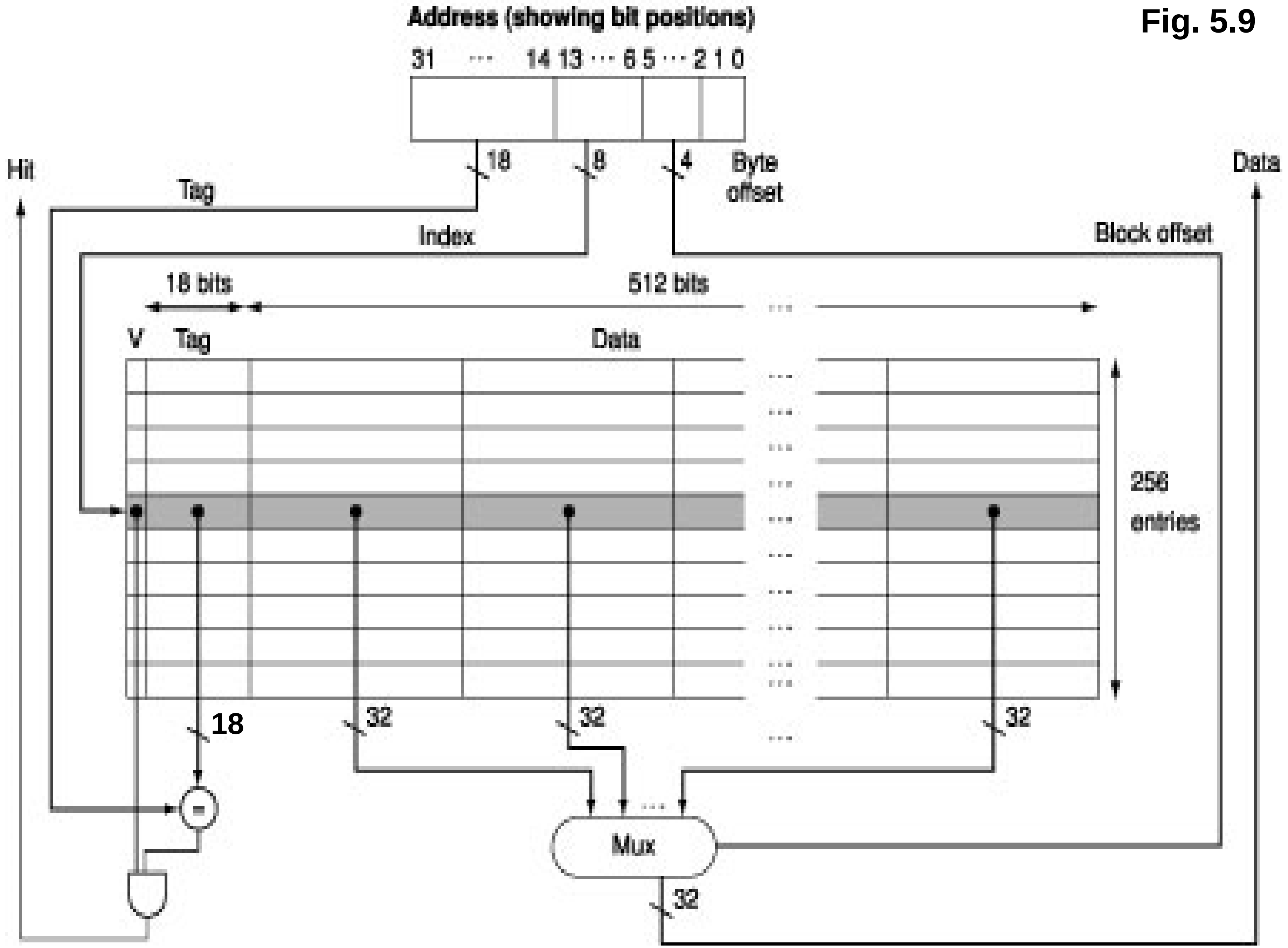
Program	Instr. Miss Rate	Data Miss Rate	Combined Miss Rate
gcc	6.1%	2.1%	5.4%
spice	1.2%	1.3%	1.2%

DecStation 3100 Cache - Old Fig. 7.10

Variation: Larger Block Size

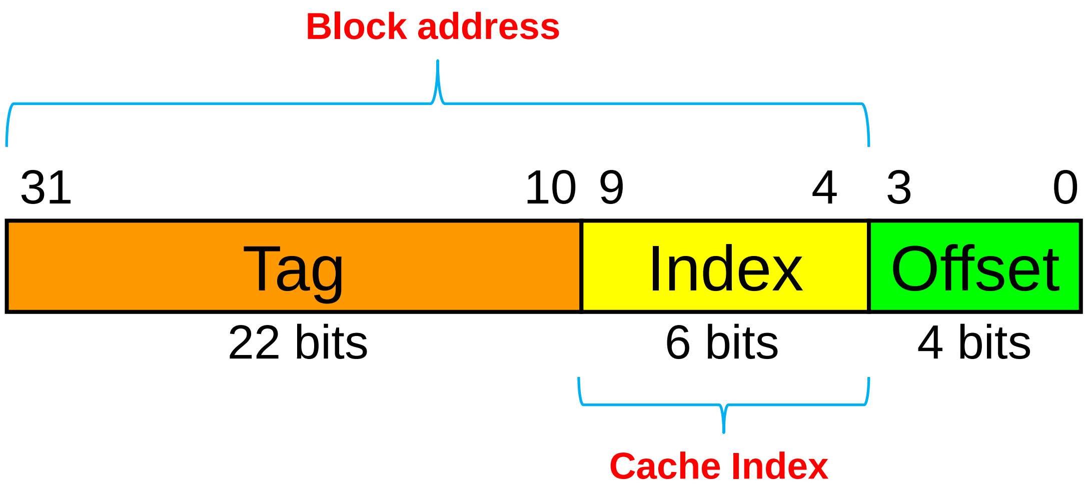
- Key advantage: take advantage of spatial locality
  - Move a number of locations to a cache at a time
- Disadvantages: competition, complicated writes

Fig. 5.9


Address (showing bit positions)
31 ··· 14 13 ··· 6 5 ··· 2 1 0
18
8
4
Byte offset
Hit
Tag
Index
Block offset
Data
18 bits
512 bits
V
Tag
Data
256 entries
18
32
32
32
=
Mux
32

Example: Larger Block Size

- 64 blocks, 16 bytes/block
- Q: To what block number does address 1200 map?
  - Block address = $\lfloor 1200/16 \rfloor$ = 75 (Tag + Index)
  - Cache Index (Address) = 75 modulo 64 = 11 (index)



Block Size Considerations

larger block size vs. smaller block size with a fixed size of cache

- Larger block size can reult in reduced miss rate
  - Due to spatial locality
- But in a fixed-sized cache
  - Larger block size ⇒ fewer number of blocks (블락사이즈가 커지면 블락의 수는 줄어듦)
  - More competition ⇒ increased miss rate
  - Larger block size ⇒ pollution
- Larger miss penalty (블락사이즈가 커지면 Miss 패널티는 커짐)
  - Can override benefit of reduced miss rate
  - Early restart and critical-word-first can help

Miss Rates vs. Block Size

- Note miss rate increases for larger block sizes

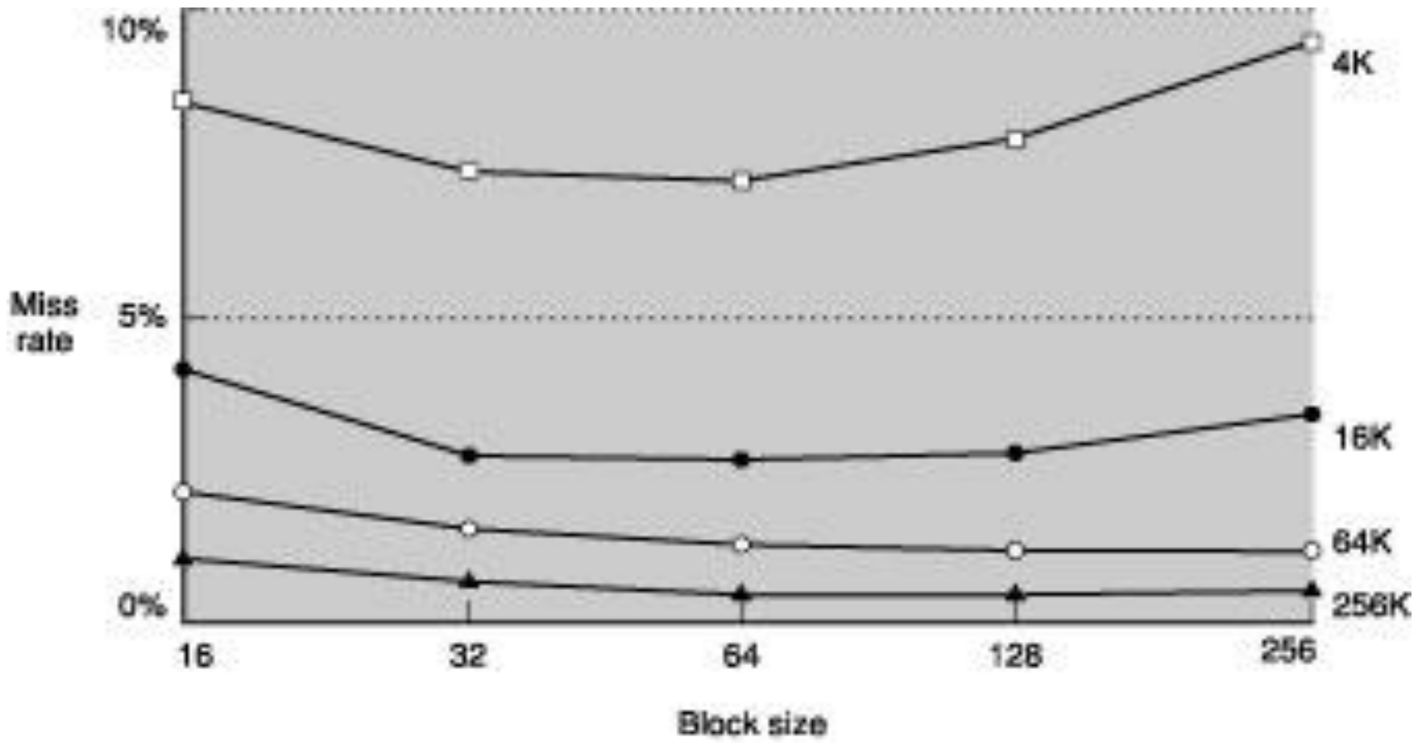


Fig. 5.8

Cache Misses

- On cache hit, CPU proceeds normally

- On cache miss

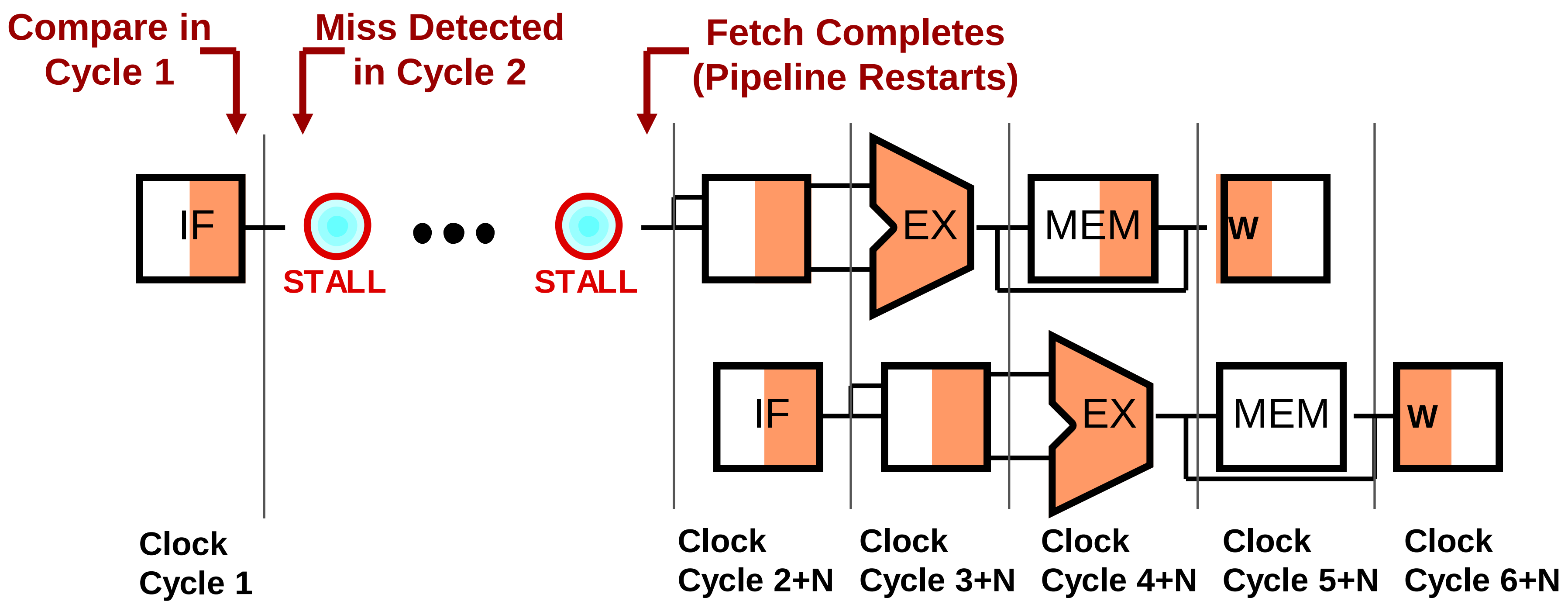
  (1) Stall the CPU pipeline

  (2) Fetch block from next level of memory hierarchy

  (3) Restart instruction fetch OR
  Complete data access

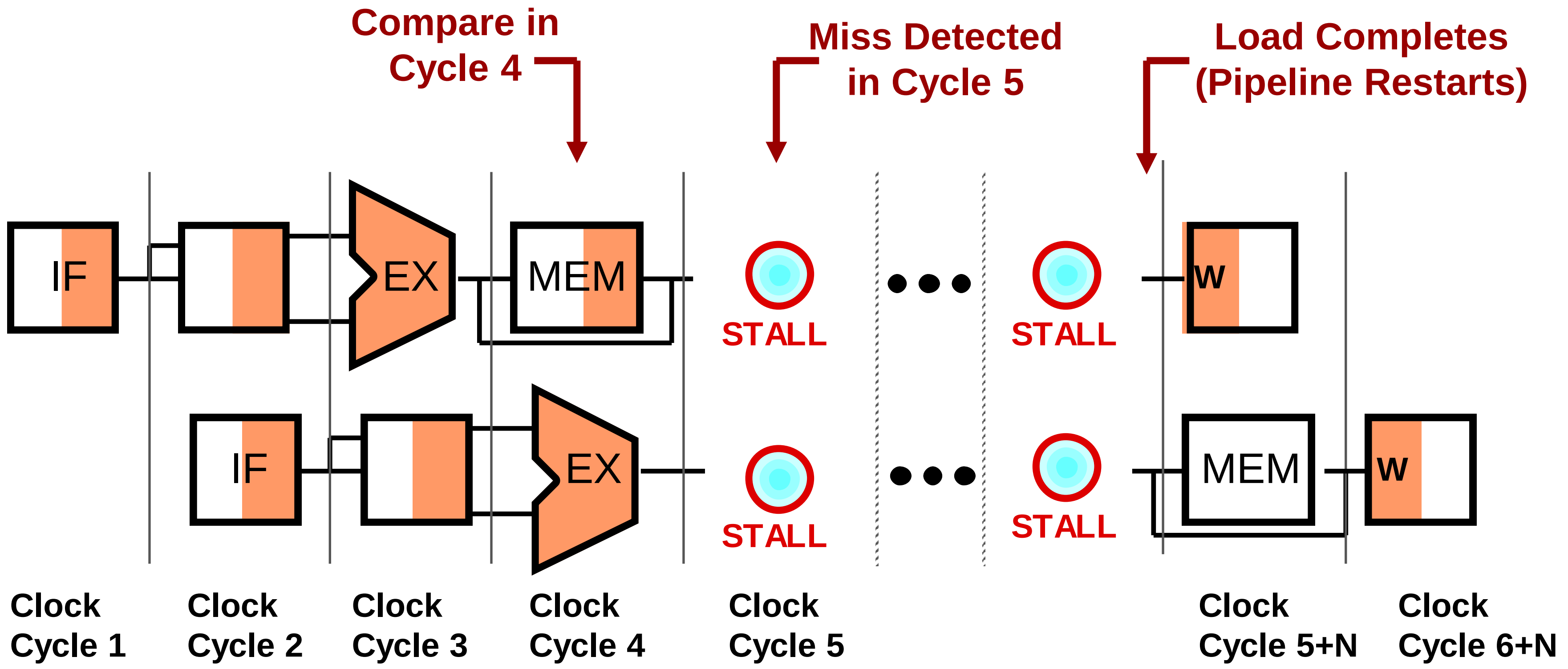
Cache Miss and the MIPS Pipeline

- Instruction Fetch



Cache Miss and the MIPS Pipeline

- Load Instruction



Today’s lecture ends...

Four Key Cache Questions:

1. Where can block be placed in cache?
   (block placement)
2. How can block be found in cache? …using a tag
   (block identification)
3. **Which block should be replaced on a miss?**
   (block replacement)
4. **What happens on a write?**
   (write strategy)

Q4: Write Strategy

- What happens on a write?
  1. Write through - write to memory, stall processor until done
  2. Write buffer - place in buffer (allows pipeline to continue*)
  3. Write back - delay write to memory until block is replaced in cache
- Special considerations required when using DMA and multiprocessors
  - DMA => coherence between different memory units
  - Multiprocessors => coherence between caches

Write-Through

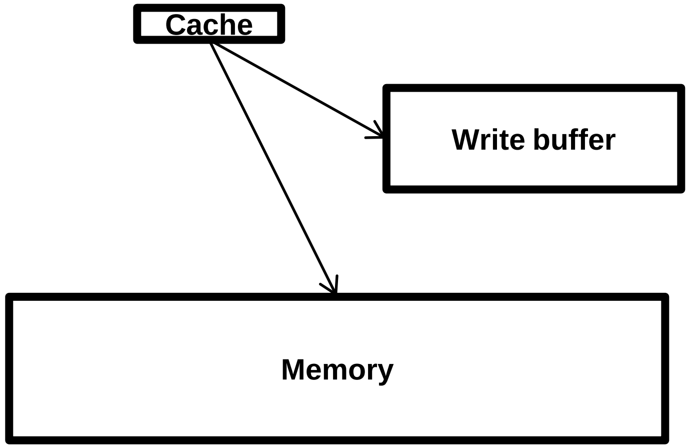
- On data-write hit, just update the block in cache
  - But then cache and memory would be inconsistent
- ① Write through: also update memory at the same time
- But makes writes take longer
  - e.g., if base CPI = 1, 10% of instructions are stores, and write to memory takes 100 cycles,

$$\text{Effective CPI} = 1 + 0.1 \times 100 = 11$$

- Solution: ② Write buffer
  - Holds data waiting to be written to memory
  - CPU continues immediately
    - Only stalls on write if write buffer is already full

③ Write-Back

- Alternative: On data-write hit, just update the block in cache
  - Keep track of whether each block is dirty
- When a dirty block is replaced
  - Write it back to memory
  - Can use a write buffer to allow replacing block to be read first



*Dirty block : a block with modified data

Write Allocation

- What should happen on a write miss?
- Alternatives for write-through
  - Allocate on miss: fetch the block
  - Write around: don’t fetch the block
    - Since programs often write a whole block before reading it (e.g., initialization)
- For write-back
  - Usually fetch the block

Example: Intrinsity FastMATH

- Embedded MIPS processor
  - 12-stage pipeline
  - Instruction and data access on each cycle
- Split cache: separate I-cache and D-cache
  - Each 16KB: 256 blocks × 16 words/block
  - D-cache: write-through or write-back
- SPEC2000 miss rates
  - I-cache: 0.4%
  - D-cache: 11.4%
  - Weighted average: 3.2%

Example: Intrinsity FastMATH

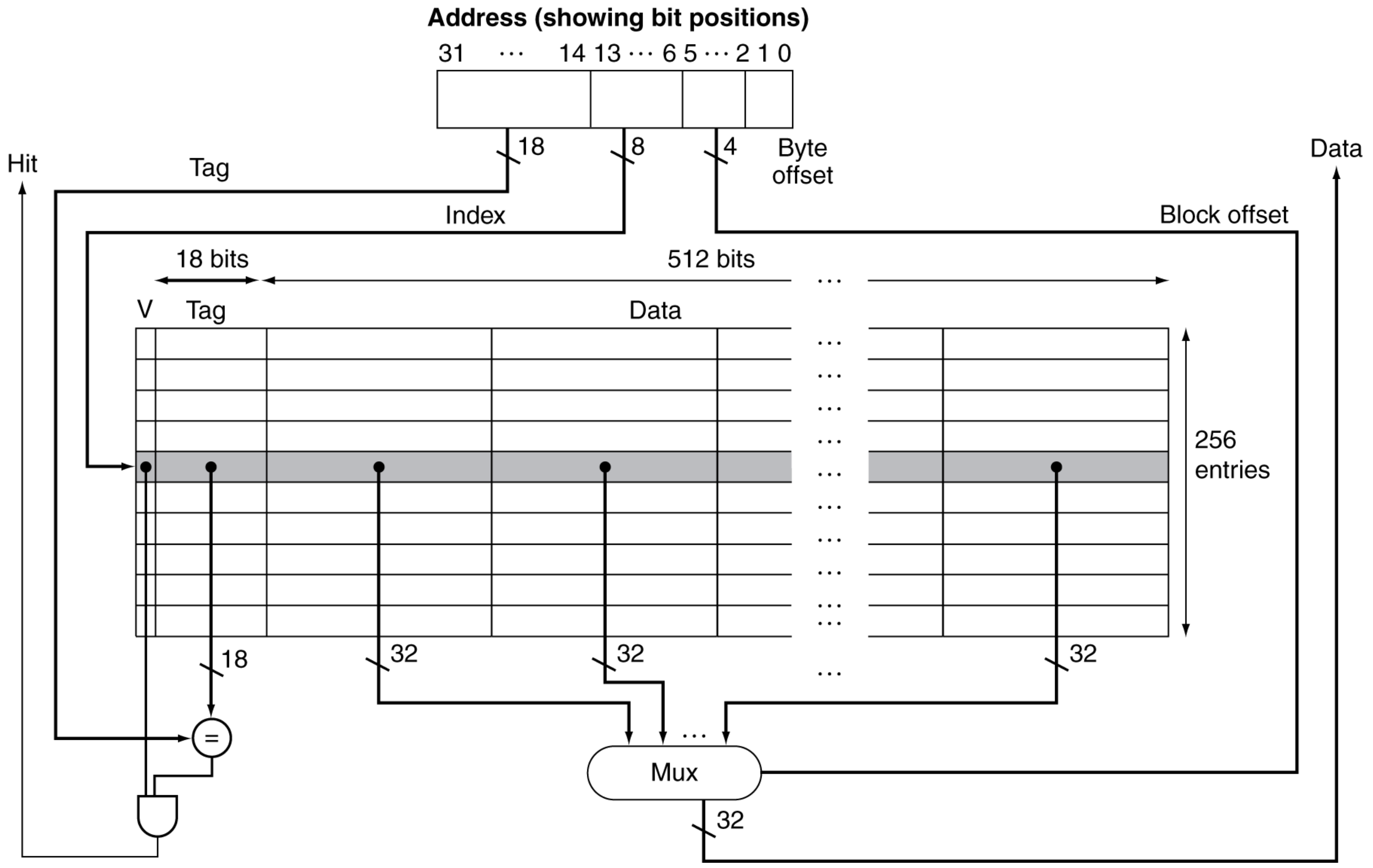


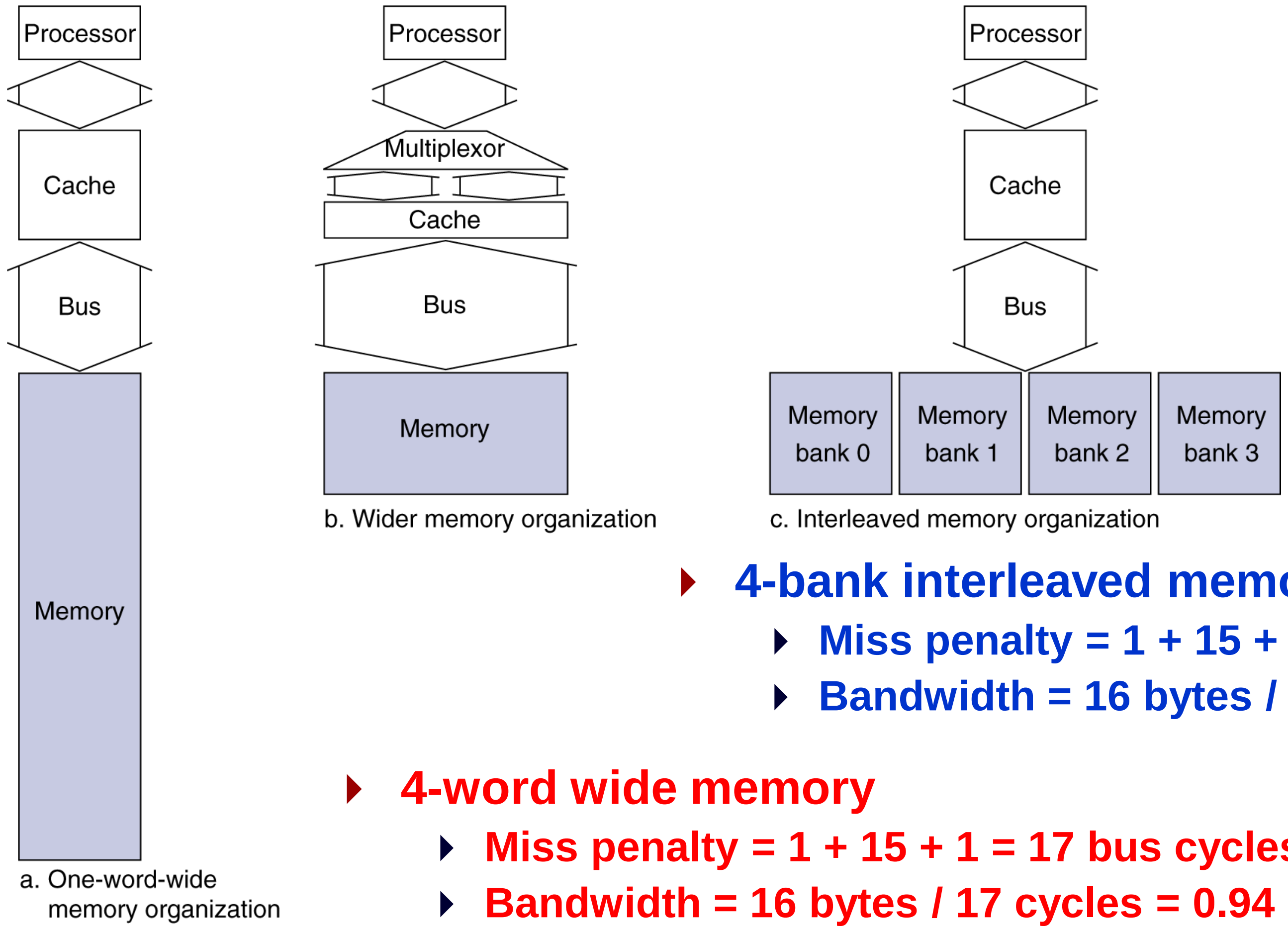
Fig. 5.9

Main Memory Supporting Caches

- Use DRAMs for main memory
  - Fixed width (e.g., 1 word)
  - Connected by fixed-width clocked bus
    - Bus clock is typically slower than CPU clock
- Example cache block read
  - 1 bus cycle for address transfer
  - 15 bus cycles per DRAM access
  - 1 bus cycle per data transfer

Increasing Memory Bandwidth

Fig. 5.11



- **4-bank interleaved memory**
  - **Miss penalty = 1 + 15 + 4 × 1 = 20 bus cycles**
  - **Bandwidth = 16 bytes / 20 cycles = 0.8 B/cycle**

- **4-word wide memory**
  - **Miss penalty = 1 + 15 + 1 = 17 bus cycles**
  - **Bandwidth = 16 bytes / 17 cycles = 0.94 B/cycle**

- **For 4-word cache block, 1-word-wide DRAM**
  - **Miss penalty = 1 + 4 × 15 + 4 × 1 = 65 bus cycles**
  - **Bandwidth = 16 bytes / 65 cycles = 0.25 B/cycle**

Measuring Cache Performance

- Components of CPU time
  - Program execution cycles
    - Includes cache hit time
  - Memory stall cycles
    - Mainly from cache misses
- With simplifying assumptions:

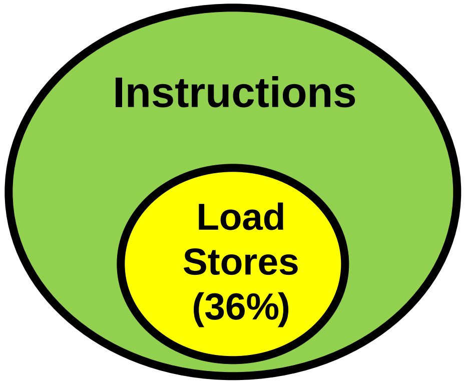
$$\text{Memory stall cycles}$$

$$= \frac{\text{Memory accesses}}{\text{Program}} \times \text{Miss rate} \times \text{Miss penalty}$$

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

Cache Performance Example (p. 477)

- Given
  - I-cache miss rate = 2%
  - D-cache miss rate = 4%
  - Miss penalty = 100 cycles
  - Base CPI (ideal cache) = 2
  - Load & stores are 36% of instructions



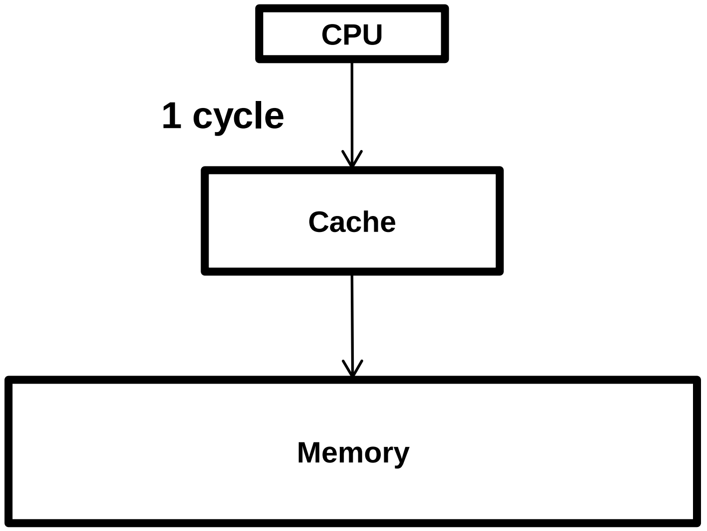
- Miss cycles per instruction
  - I-cache: 0.02 × 100 = 2 (명령어는 cache miss 하면 모두 메모리 엑세스)
  - D-cache: 0.36 × 0.04 × 100 = 1.44 (데이터는 36%중에서 cache miss하면 메모리 엑세스)
- Actual CPI = 2 + 2 + 1.44 = 5.44
  - Ideal CPU (with miss rate = 0) is 5.44/2 = 2.72 times faster than actual system

Cache Performance Example (cont'd)

- What happens if we decrease CPI from 2 to 1?
- Miss cycles per instruction - no change!
  - I-cache: $0.02 \times 100 = 2$
  - D-cache: $0.36 \times 0.04 \times 100 = 1.44$
- Actual CPI = 1 + 2 + 1.44 = 4.44
  - Ideal CPU is 4.44/1 =4.44 times faster than actual system
  - Speedup over original one is 5.44/4.44 = 1.23X
- As CPI drops, fraction of time stalled increases:
  - CPI = 2: 3.44 / 5.44 = 63%
  - CPI = 1: 3.44 / 4.44 = 77%

Average Memory Access Time

- Hit time is also important for performance
- Average memory access time (AMAT)
  - AMAT = Hit time + Miss rate × Miss penalty



- Example: CPU with
  - 1ns clock
  - Hit time = 1 cycle
  - Miss penalty = 20 cycles
  - I-cache miss rate = 5%
  - AMAT = 1 + 0.05 × 20 = 2ns (2 cycles/instruction)

Performance Summary

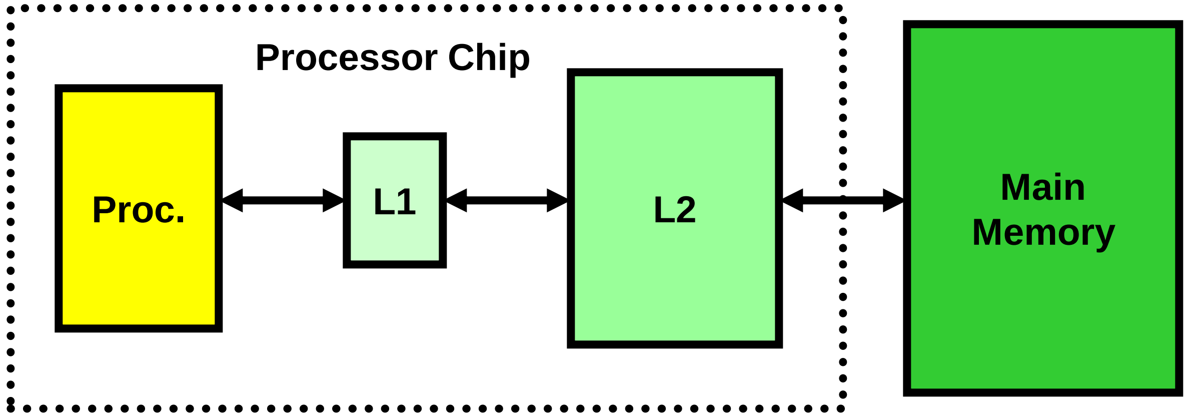
- When CPU performance increased
  - Miss penalty becomes more significant
- Decreasing base CPI (example in PPT 37)
  - Greater proportion of time spent on memory stalls
- Increasing clock rate
  - Memory stalls account for more CPU cycles
- Can’t neglect cache behavior when evaluating system performance

Block Replacement Policy

- Direct mapped: no choice
- Set associative
  - Prefer non-valid entry, if there is one
  - Otherwise, choose among entries in the set
- Least-recently used (LRU)
  - Choose the one unused for the longest time
    - Simple for 2-way, manageable for 4-way, too hard beyond that
- Random
  - Gives approximately the same performance as LRU for high associativity

Multilevel Caches

- L1 cache attached to CPU
  - Small, but fast
- L2 cache services primary (L1) cache misses
  - Larger, slower, but still faster than main memory
- Main memory services L2 cache misses



Multilevel Cache Example

- Given
  - CPU base CPI = 1
  - Clock rate = 4GHz => 0.25ns
  - Miss rate/instruction = 2%
  - Main memory access time = 100ns

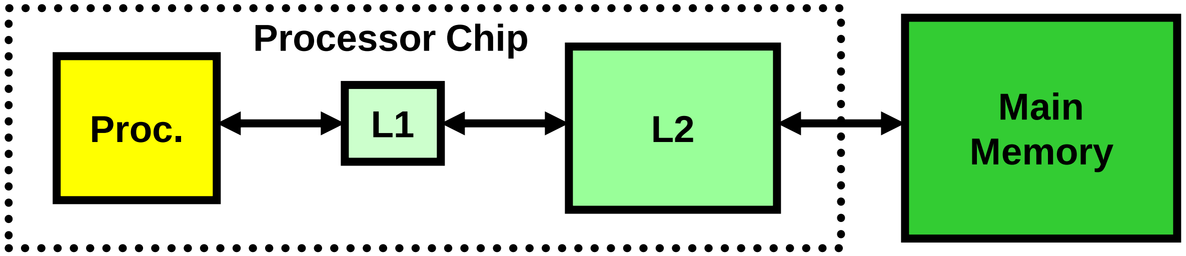
- What is effective CPI with just primary cache?
  - Miss penalty = 100ns/0.25ns = 400 cycles
  - Effective CPI = 1 + $0.02 \times \underset{\substack{\uparrow \\ \text{Memory access penalty}}}{400}$ = 9

Example (cont.)

Miss rate/instruction = 2%

- Now add L2 cache
  - Access time = 5ns
  - Global miss rate to main memory = 0.5%
- Primary miss with L2 hit
  - Penalty = 5ns/0.25ns = 20 cycles (L2 access cycle)
- Primary miss with L2 miss
  - Extra penalty = 400 cycles (memory access cycle)

- CPI = 1 + 0.02 × 20 + 0.005 × 400 = 3.4
- Performance ratio = 9/3.4 = 2.6


Processor Chip
Proc.
L1
L2
Main Memory