CS/ECE 5381/7381 Computer Architecture Spring 2023

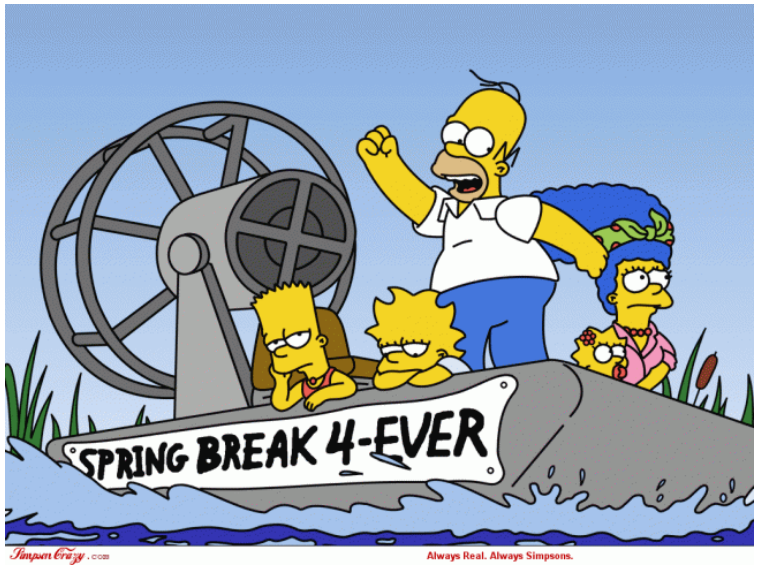
Dr. Manikas

Computer Science

Lecture 13: Mar. 9, 2023

Assignments

- Quiz 6 – due Sat., Mar. 11 (11:59 pm)
  - Covers concepts from Module 7 (this week)
- Next week is Spring Break – no lectures or assignments due



Quiz 6 Details

- The quiz is open book and open notes.
- You are allowed 90 minutes to take this quiz.
- You are allowed 2 attempts to take this quiz - your highest score will be kept.
  - Note that some questions (e.g., fill in the blank) will need to be graded manually
- Quiz answers will be made available 24 hours after the quiz due date.

Review of Memory Hierarchy

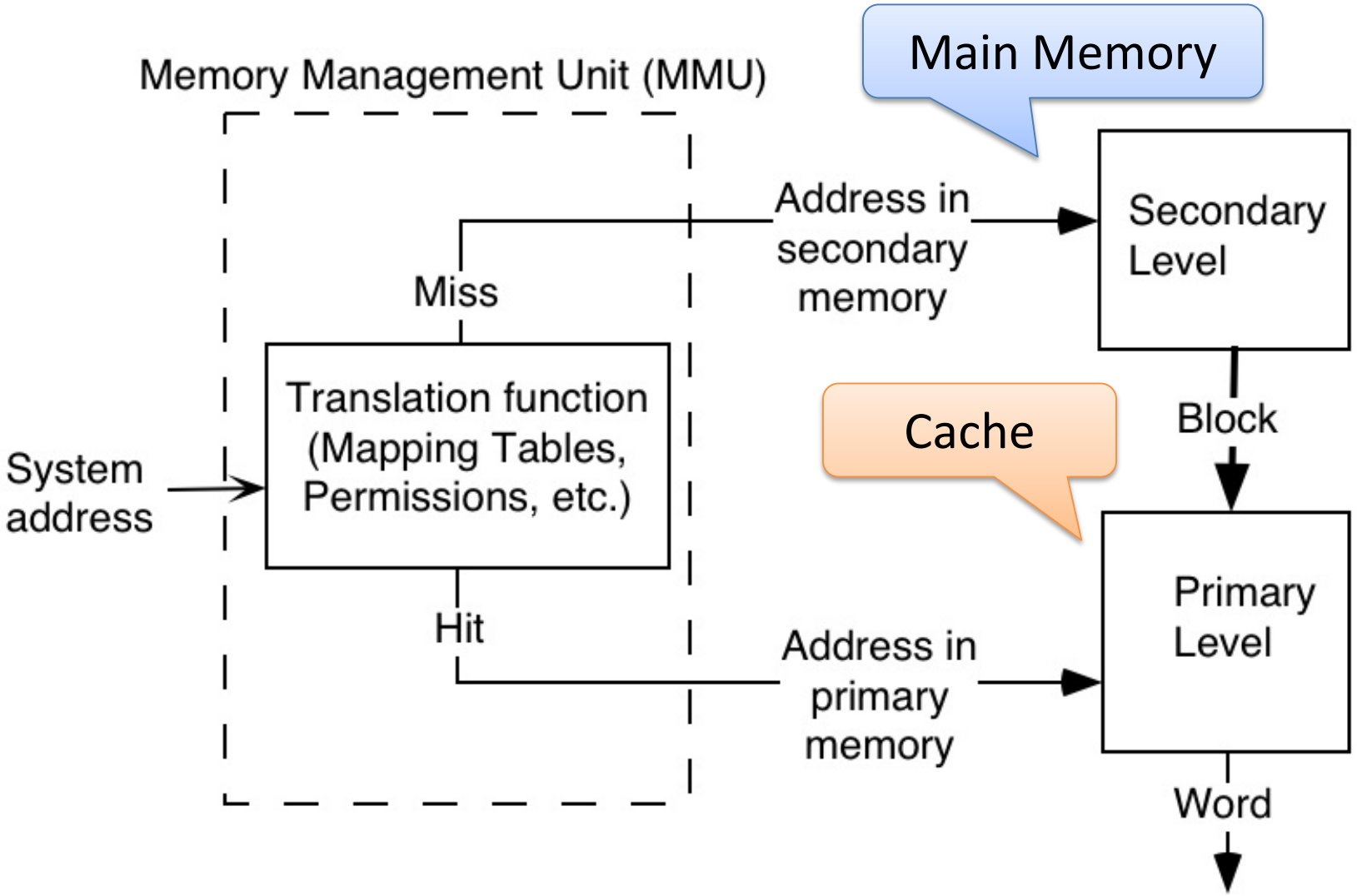
(Appendix B, Hennessy and Patterson)

Note: some course slides adopted from publisher-provided material

Outline

- B.1 Introduction
- B.2 Cache Performance
- B.3 Basic Cache Optimizations
- B.4 Virtual Memory

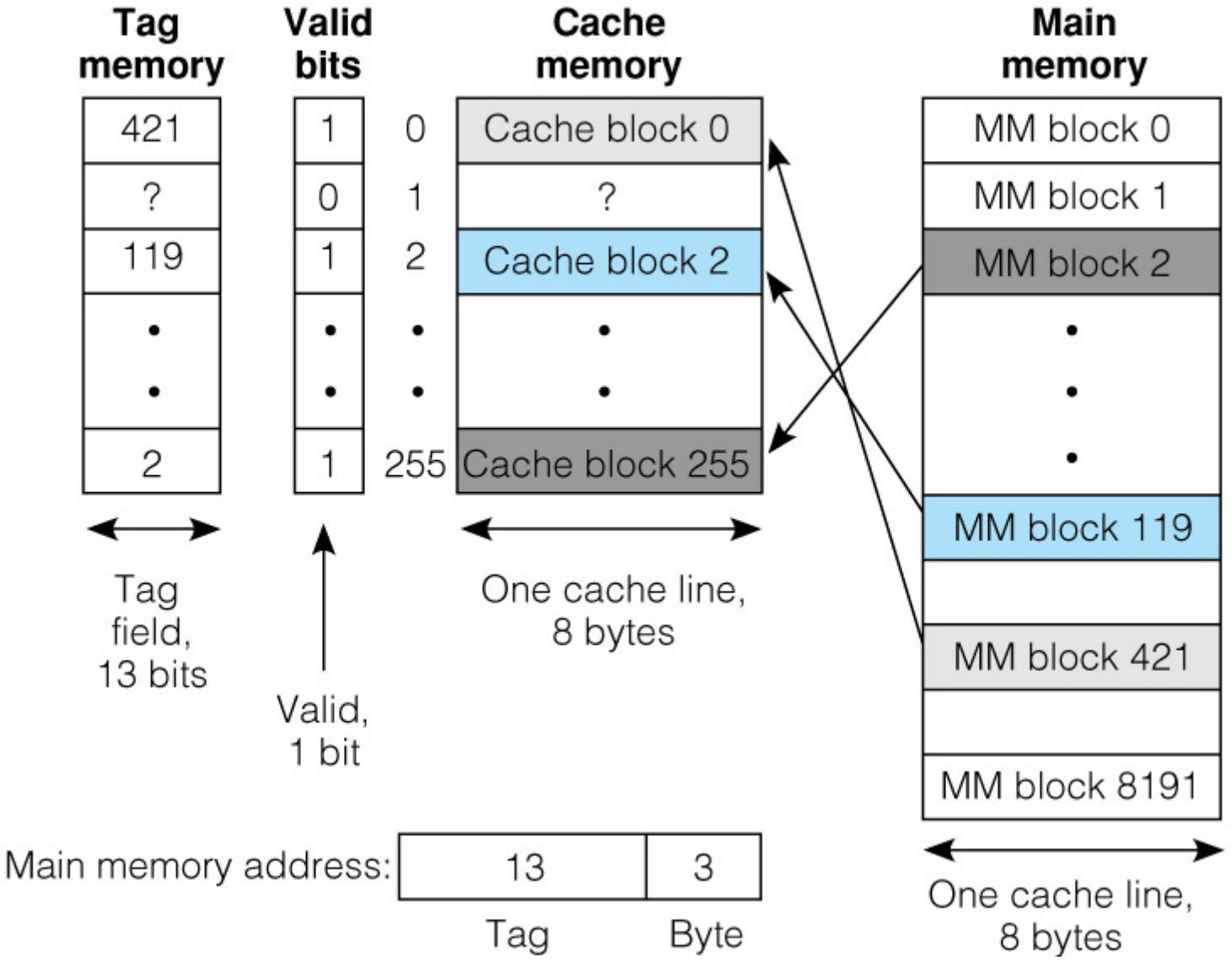
Addressing and Accessing a 2-Level Hierarchy



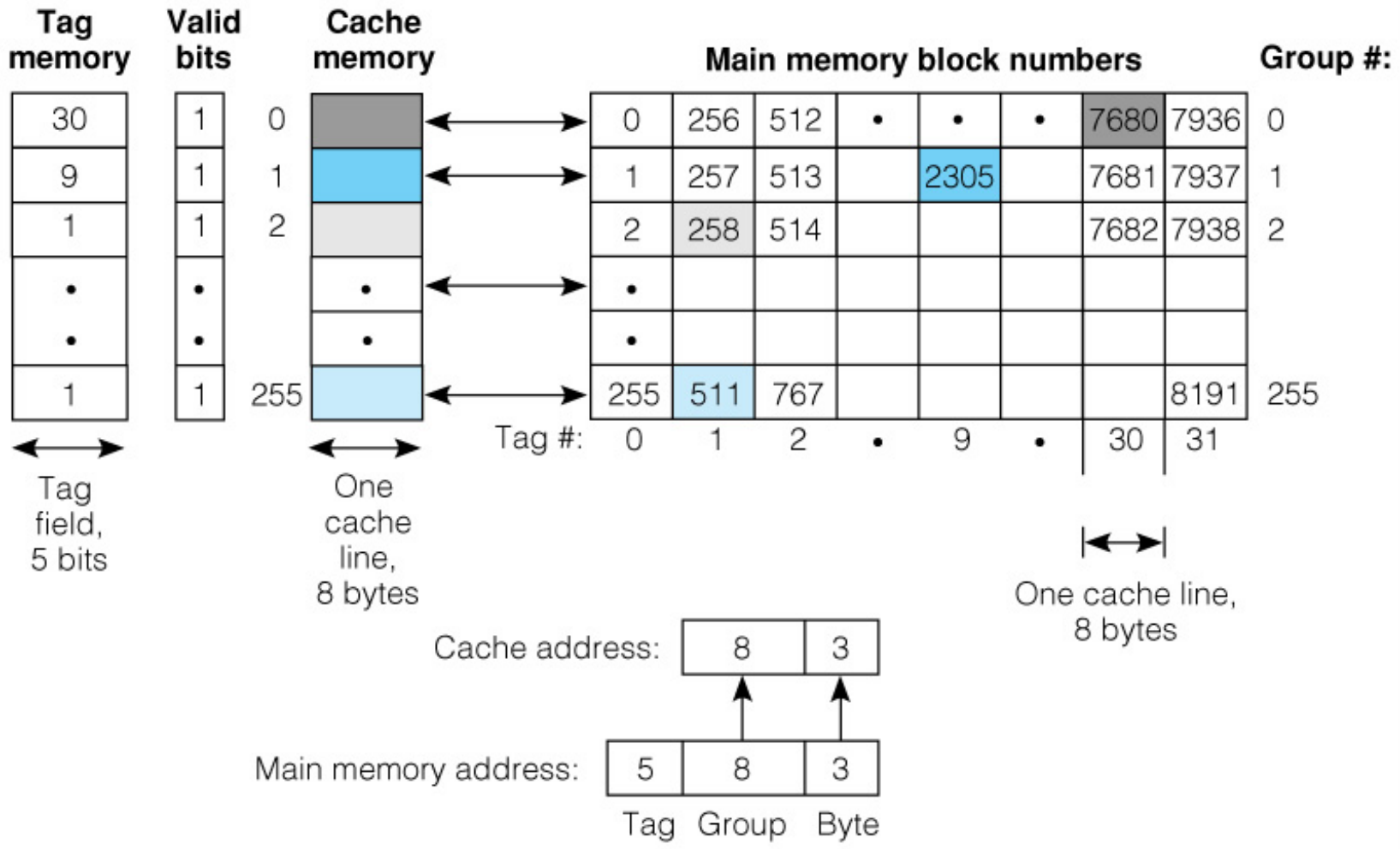
Interpreting Memory Addresses

- Memory size
  - 2^{10} = 1 K (kilo)
  - 2^{20} = 1 M (mega)
  - 2^{30} = 1 G (giga)
  - 2^{40} = 1 T (tera)
  - 2^{50} = 1 P (peta)
- 1 B (byte) = 8 b (bits)

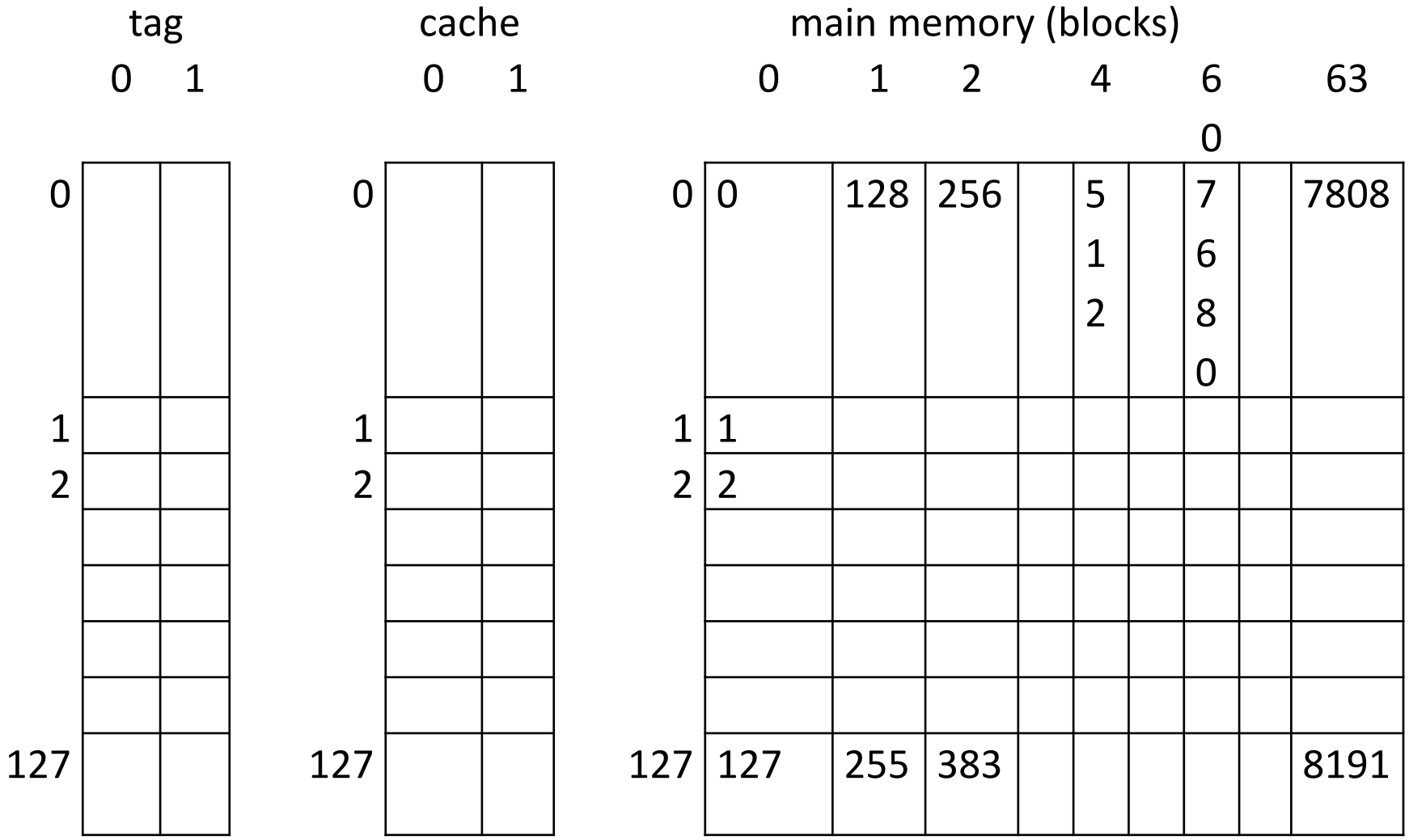
Associative mapped caches



Direct mapped cache

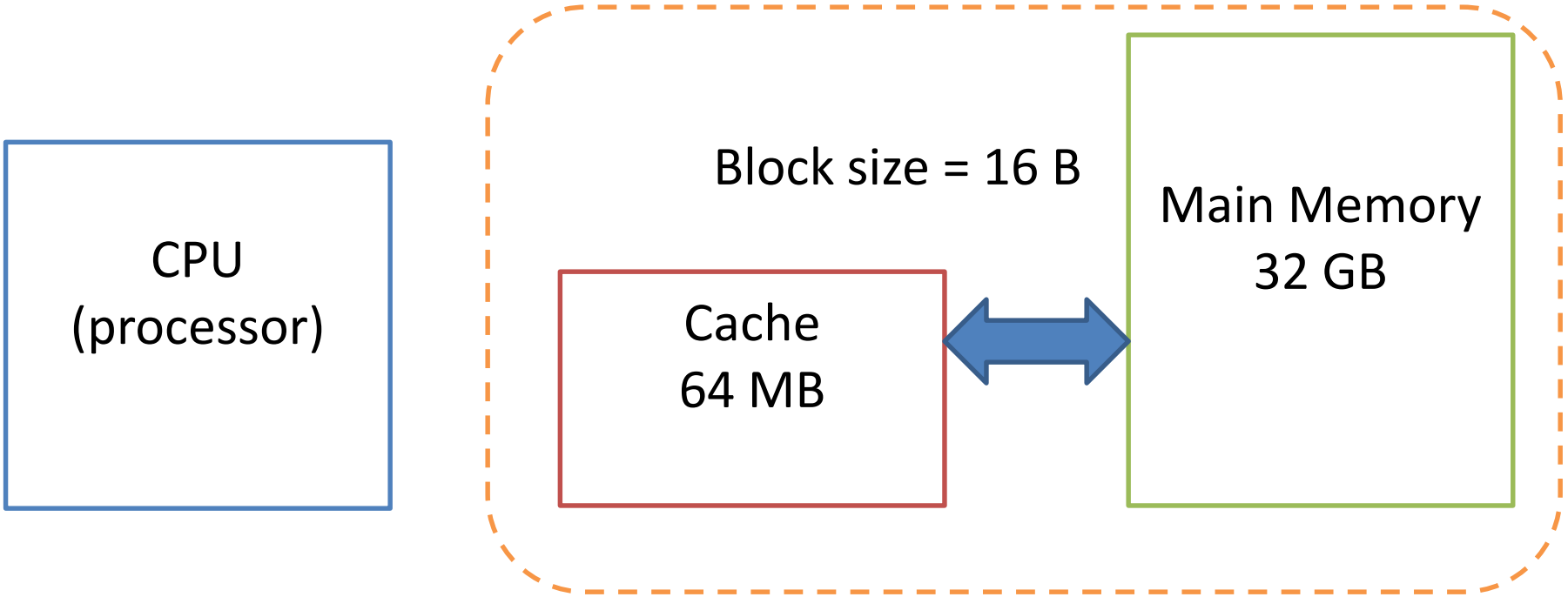


2-Way Set Associative Cache



Example B.1-2

- We have the following configuration for our memory system:



Example B.1-2 (cont.)

a) What is the size of the memory address issued by the CPU (number of bits)?

b) Show the fields in the memory address for an **associative** cache.

c) Show the fields in the memory address for a **direct-mapped** cache.

d) Show the fields in the memory address for a **4-way set-associative** cache.

MAIN MEMORY SIZE = 32 GB = $2^5 \cdot 2^{30}$ B = 2^{35} B

CACHE SIZE = 64 MB = $2^6 \cdot 2^{20}$ = 2^{26} B

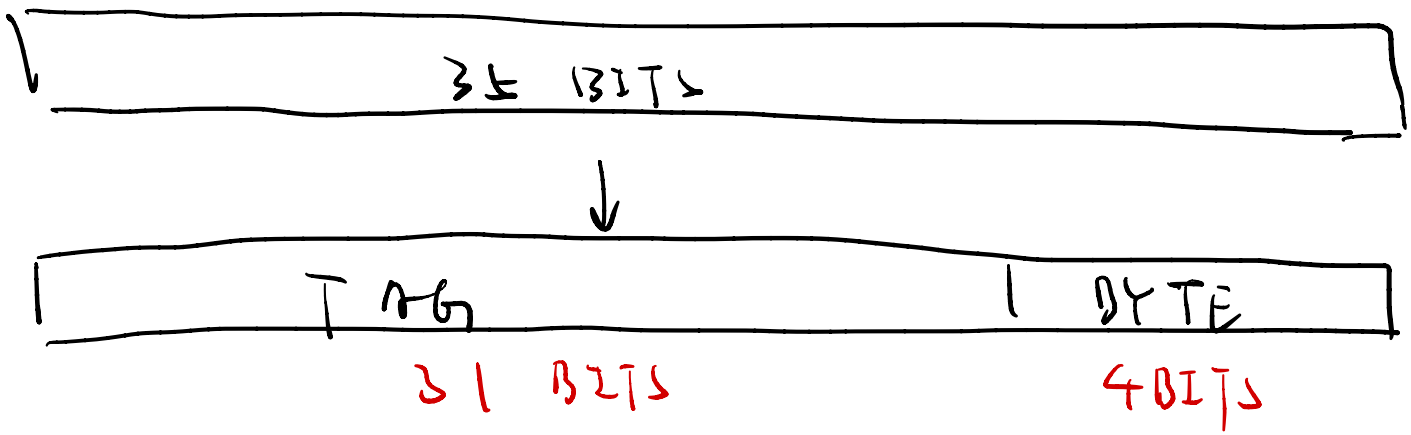
BLOCK SIZE = 16 B = 2^4 B

(a) PROCESSOR NEED TO ACCESS ALL BYTES IN MAIN MEMORY

$\Rightarrow 2^{35}$ B

ADDRESS BUS SIZE = 35 BITS

(b) ASSOCIATIVE CACHE

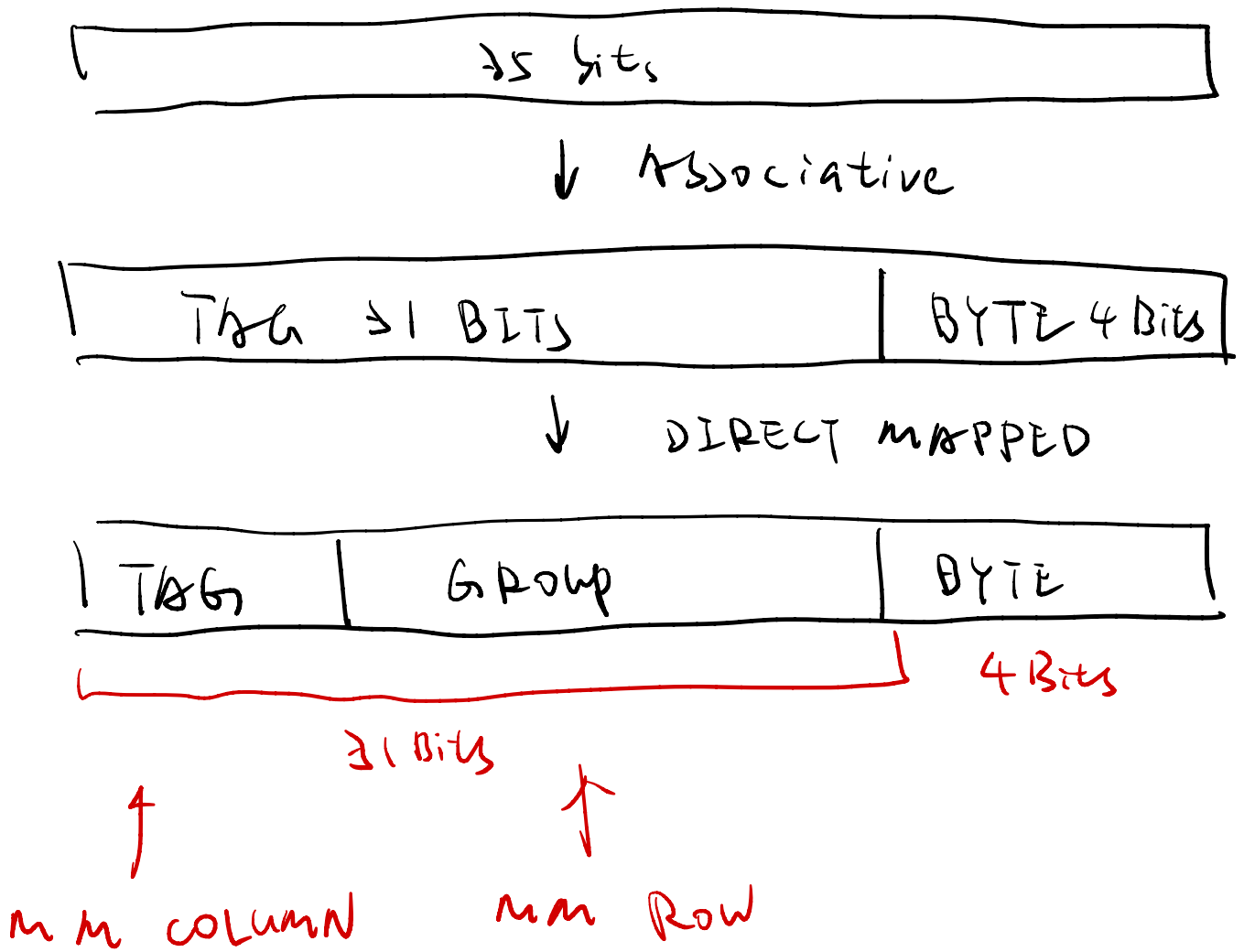


BYTE FIELD? WHAT IS BLOCK SIZE? 2^4 B

TAG FIELD? HOW MANY MAIN MEMORY BLOCKS?

$$\frac{\text{MAIN Memory SIZE} = 2^{32}\ \text{B}}{\text{BLOCK SIZE} = 2^4\ \text{B/Block}} = 2^{(28)}\ \text{Blocks}$$

© DIRECT MAPPED



MM ROWS = # CACHE BLOCKS

How MANY CACHE BLOCKS?

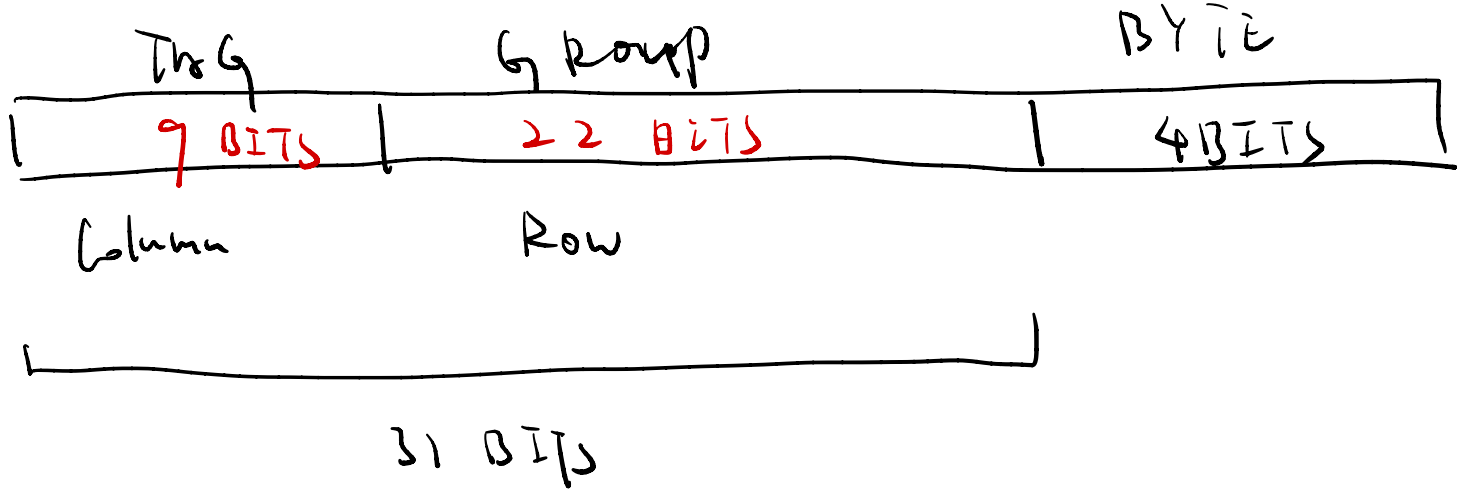
$$\frac{\text{CACHE SIZE} = 2^{26}\ \text{B}}{\text{BLOCK SIZE} = 2^{4}\ \text{B/BLOCK}} = 2^{22}\ \frac{\text{CACHE}}{\text{BLOCK}}$$

$$\Rightarrow 2^{22}\ \text{ROWS}$$

$$\#\ \text{MM COLUMNS} = \frac{\#\ \text{MM BLOCKS}}{\#\ \text{MM ROW}} = \frac{2^{31}}{2^{22}}$$

$$= 2^{9}\ \text{COLUMNS}$$

DIRECT MAPPED FIELDS?



(d) 4-WAY SET ASSOCIATIVE CACHE?

=> 4 column in CACHE

=> DIVIDE # MM ROWS BY 4

=> MULTIPLY # MM COLUMNS BY 4

DIRECT MAPPED FIELDS?

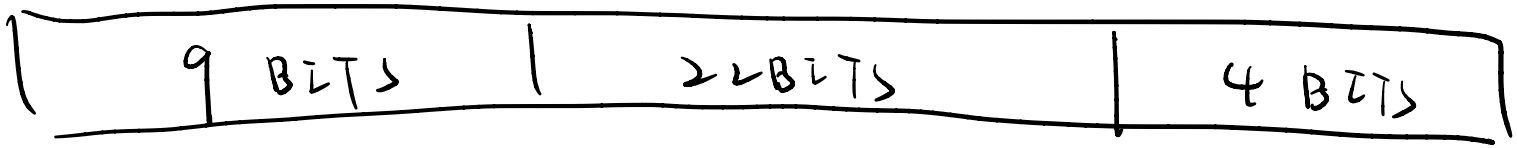
COLUMN ↓ MULT BY 4

ROW ↓ DIVIDE BY 4

MULTIPLY BY 4 => $\times 2^2$ ADD 2 BITS

DIVIDE BY 4 => $\frac{1}{2^2}$ SUBTRACT 2 BITS.

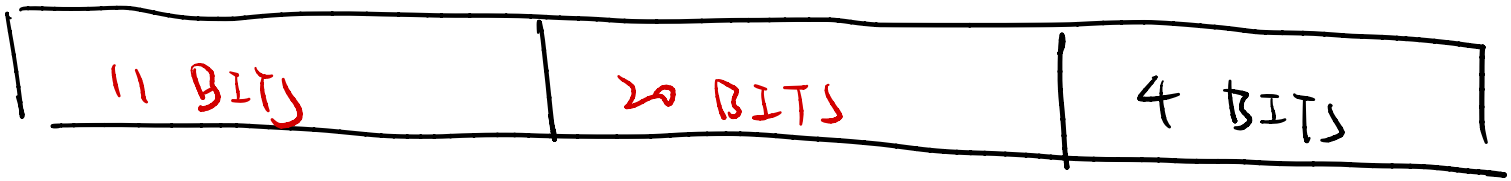
DIRECT MAPPED FIELDS



COL ROW

MULT BY 4 DIVIDE BY 4

ADD 2 BITS SUB 2 BITS



4-WAY SET ASSOCIATIVE

31 BITS

Outline

- B.1 Introduction
- B.2 Cache Performance
- B.3 Basic Cache Optimizations
- B.4 Virtual Memory

Cache Measures

- *Hit rate*: fraction found in that level
  - So high that usually talk about *Miss rate*
- Average memory-access time
  = Hit time + Miss rate x Miss penalty
  (ns or clocks)

Cache Measures (cont)

- *Miss penalty*: time to replace a block from lower level, including time to replace in cache
  - *access time*: time to lower level

    = f(latency to lower level)
  - *transfer time*: time to transfer block

    =f(BW between upper & lower levels)

Memory Access Time

- Bandwidth (BW) (throughput) – transmission rate of bits/bytes (data)
  - Common units: bits/sec (bps)
- Latency (response time) – time (sec) required to access (find) data in specified location
- Access time: latency + (block size)/bandwidth

Cache Performance Metrics

In general:

CPU Execution Time =

(CPU clock cycles) (clock cycle time)

Now, what if we have stalls due to memory access latency (e.g., load, store)? These add cycles, so:

CPU Execution Time =

(CPU clock cycles + memory stall cycles)

(clock cycle time)

Memory Stall Cycles

Memory stall cycles =

(number of cache misses)(miss penalty)

Number of cache misses =

(instruction count)(misses/instruction)

Misses/instruction =

(memory accesses/instruction)(miss rate)

Memory stall cycles = (instruction count) (memory accesses/instruction)(miss rate)(miss penalty)

Instructions Affected by Memory Stalls

- Memory stalls due to cache hits and misses
- Only instructions that access main memory are affected
- MIPS Datapath
  - Instruction memory has I-cache
    - Accessed for EVERY instruction fetch
  - Data memory has D-cache
    - Accessed ONLY during load and store operations

RECALL MIPS PIPELINE STAGES

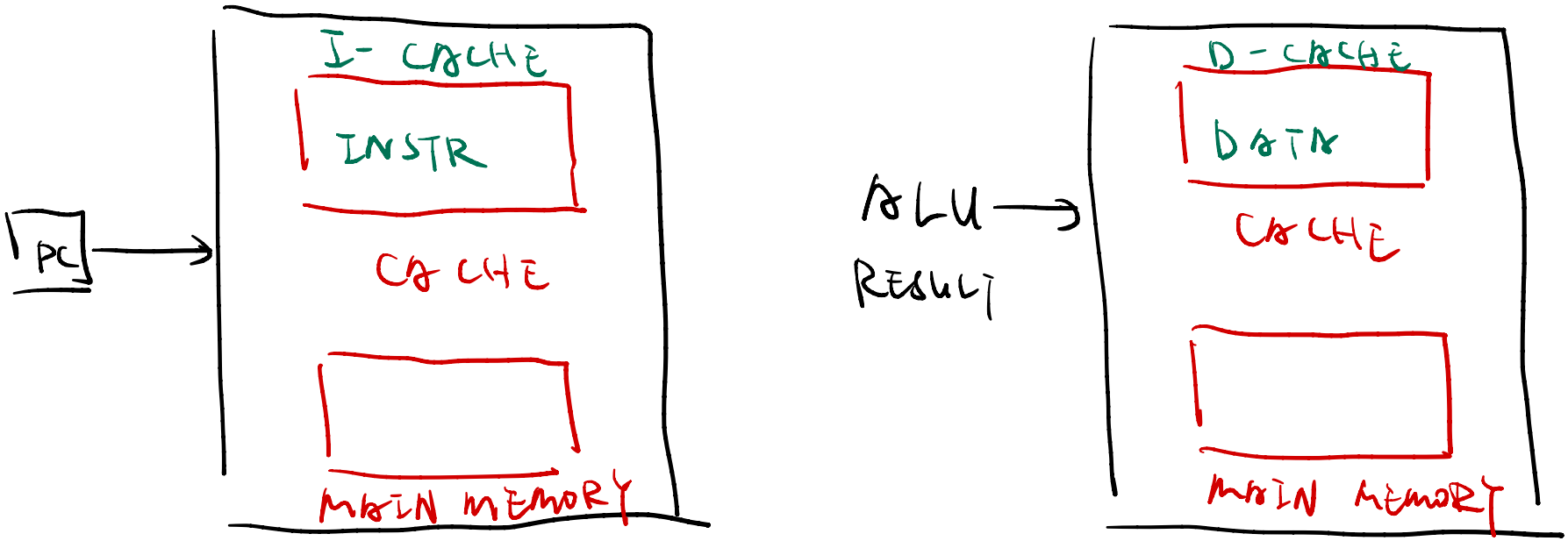
- WHAT STAGES ACCESS MEMORY?

IF

INSTRUCTION MEMORY

MEM

DATA MEMORY



Ideal Cache

Ideal cache: hit rate = 1.00 (100%)

Miss rate = 1 – hit rate = 0

Memory stall cycles = 0

CPU execution time =

(CPU clock cycles)(clock cycle time)

= (IC)(CPI)(clock cycle time)

Real Cache

Real cache: hit rate = h, where $0 < h < 1$

Miss rate = $1 - h$

Memory stall cycles = (instruction count) (memory accesses/instruction)(miss rate)(miss penalty)

CPU execution time = [(IC)(CPI) + memory stall cycles](clock cycle time)

Example B.2-1

We are given a memory system with separate caches for instructions (I-cache) and data (D-cache). The system has the following specifications:

- Base CPI = 2
- Load and stores are 20% of instructions
- I-cache miss rate = 4%
- D-cache miss rate = 2%
- Miss penalty = 100 cycles

What is the **actual CPI** for this system if we consider the effects of cache misses?

BASE CPI = 2

⇒ THIS IS CPI FOR "IDEAL" CACHE

(100% HIT RATE, 0% MISS RATE)

e.g. APP. & EQUATIONS

NON-IDEAL (REAL) CACHE WITH MISSES

ACTUAL CPI = BASE CPI

$+ CPI_{I\text{-}CACHE}$

$+ CPI_{D\text{-}CACHE}$

$CPI_{I\text{-}CACHE}$?

CPI = (% ITEMS ACCESSED)

(MISS RATE)(MISS PENALTY)

$$\% \text{ ITEMS ACCESSED} = \frac{\text{ITEMS}}{\text{INSTRUCTIONS}}$$

$$= 100\% = 1$$

MISS RATE (I-CACHE) = 4% = 0.04

MISS PENALTY = 100 CYCLES

$CPI_{I\text{-}CACHE} = (1)(0.04)(100) = \boxed{4}$

$CPI_{D\text{-}CACHE}$

% ITEMS ACCESSED = 20% = 0.20

MISS RATE = 2%

MISS PENALTY = 100 cycles

$CPI_{D\text{-}CACHE} = (0.2)(0.02)(100) = \boxed{0.4}$

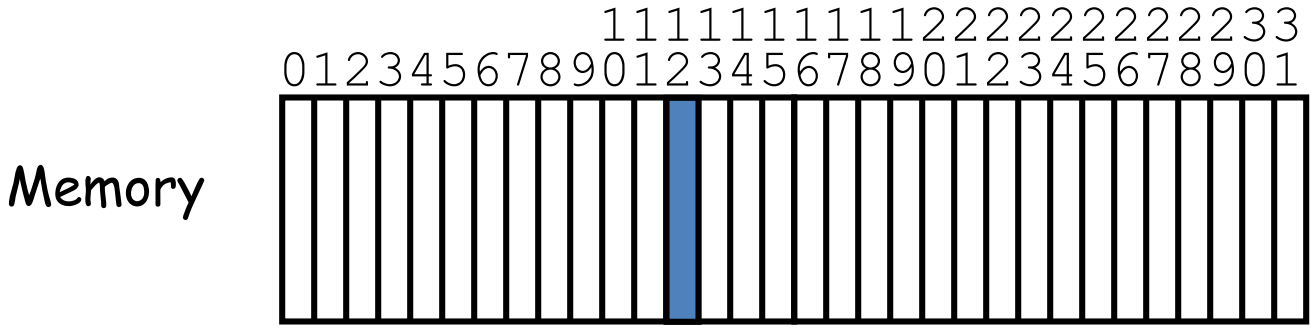
ACTUAL CPI = $2 + 4 + 0.4 = \boxed{6.4}$

4 Questions for Memory Hierarchy

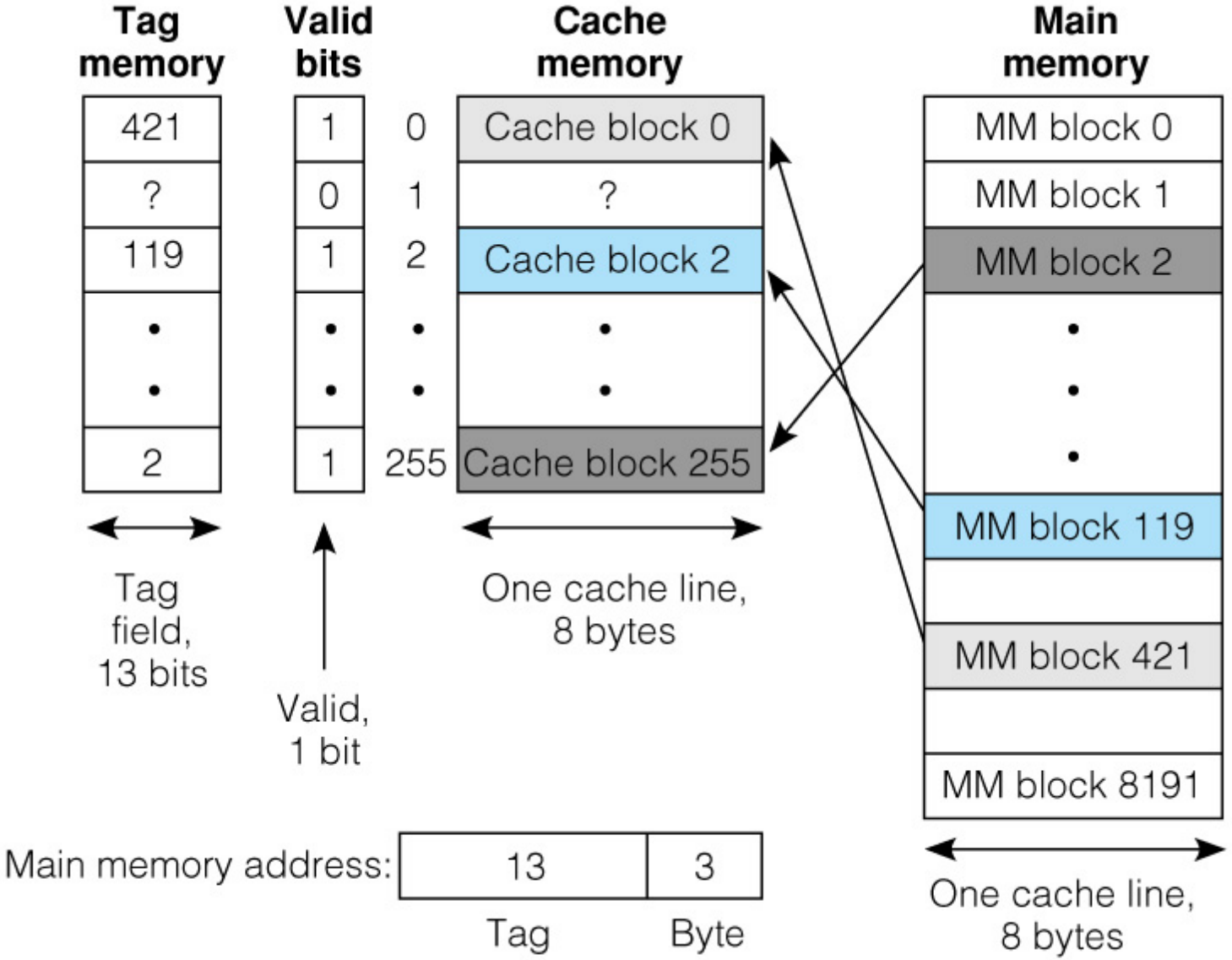
- Q1: Where can a block be placed in the upper level (cache)? *(Block placement)*
- Q2: How is a block found if it is in the upper level? *(Block identification)*
- Q3: Which block should be replaced on a miss? *(Block replacement)*
- Q4: What happens on a write? *(Write strategy)*

Q1: Block Placement

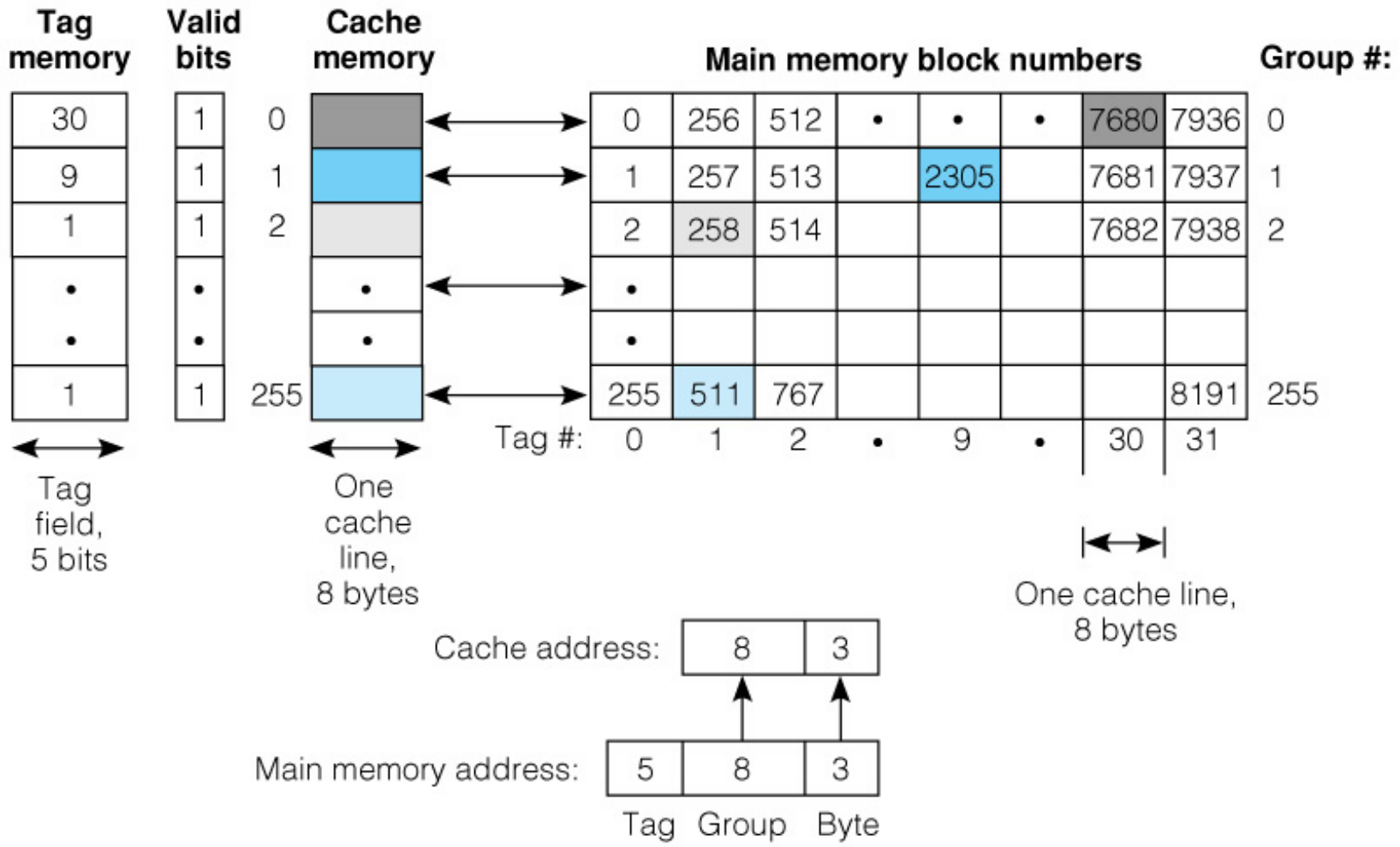
- Block 12 placed in 8 block cache:
  - Fully associative, direct mapped, or 2-way set associative



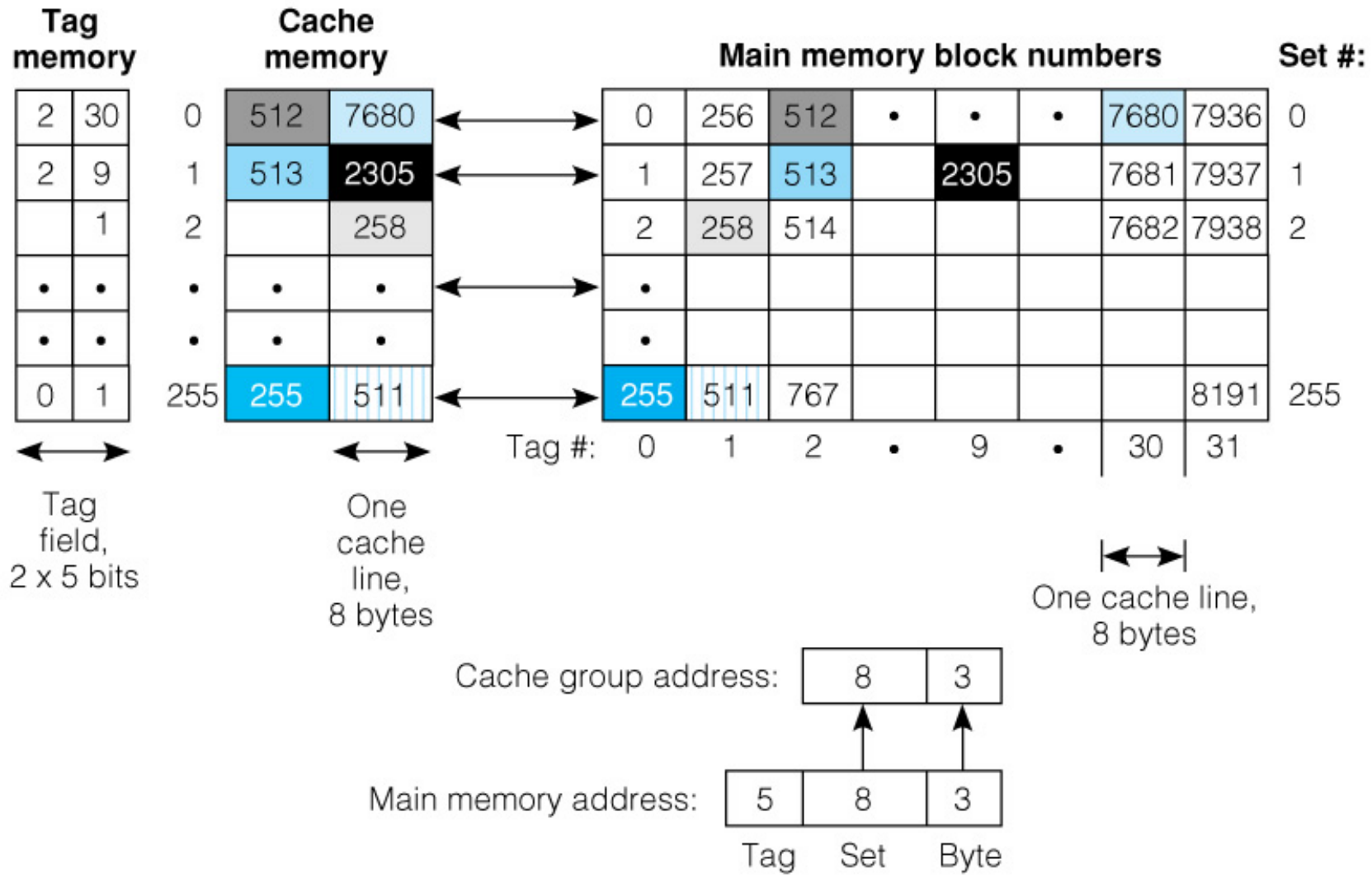
Associative mapped caches



Direct mapped cache

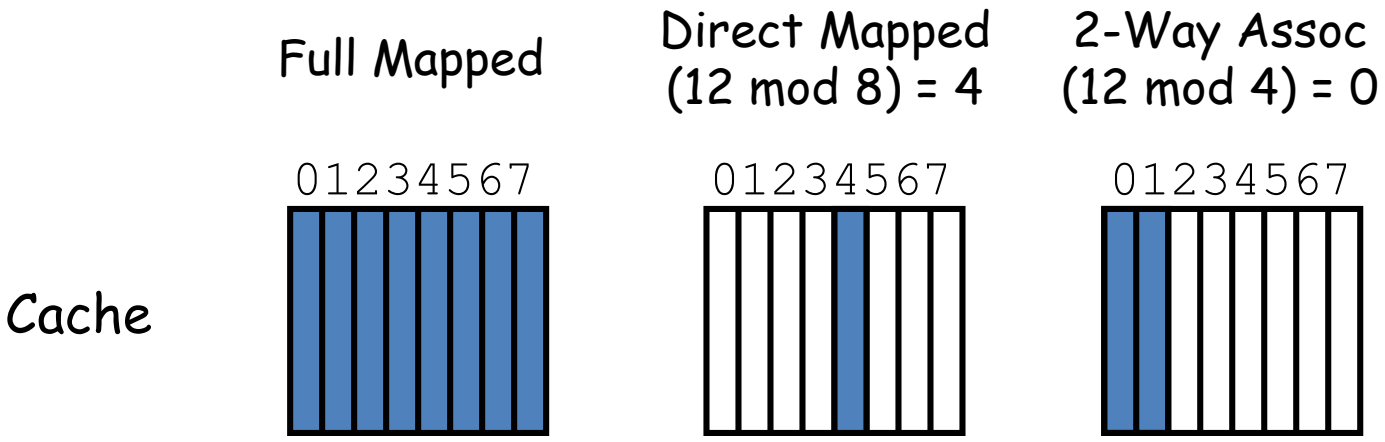


2-Way Set Associative Cache



Q1: Block Placement

- Block 12 placed in 8 block cache:



Q2: Block Identification

- Tag on each block
  - No need to check index or block offset
- Increasing associativity shrinks index, expands tag

Q3: Block Replacement

- Easy for Direct Mapped
- Set Associative or Fully Associative:
  - Random
  - LRU (Least Recently Used)

Q3: After a cache read miss, if there are no empty cache blocks, which block should be removed from the cache?

The Least Recently Used (LRU) block? Appealing, but hard to implement for high associativity

A randomly chosen block? Easy to implement, how well does it work?

Miss Rate for 2-way Set Associative Cache

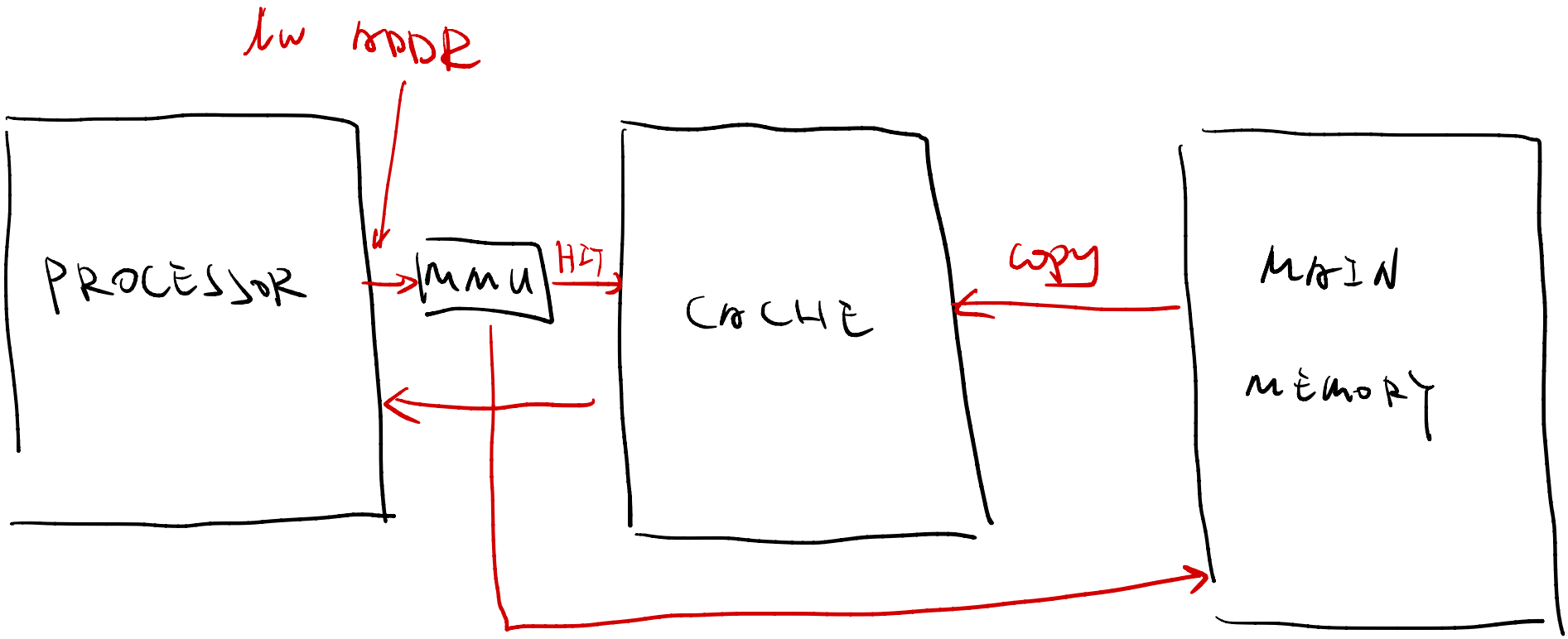
Size	Random	LRU
16 KB	5.7%	5.2%
64 KB	2.0%	1.9%
256 KB	1.17%	1.15%

Also, try other LRU approx.

Q4: What happens on a write?

	Write-Through	Write-Back
Policy	Data written to cache block also written to lower-level memory	Write data only to the cache Update lower level when a block falls out of the cache
Debug	Easy	Hard
Do read misses produce writes?	No	Yes
Do repeated writes make it to lower level?	Yes	No

CACHE OPERATION: READ (e.g. lw)



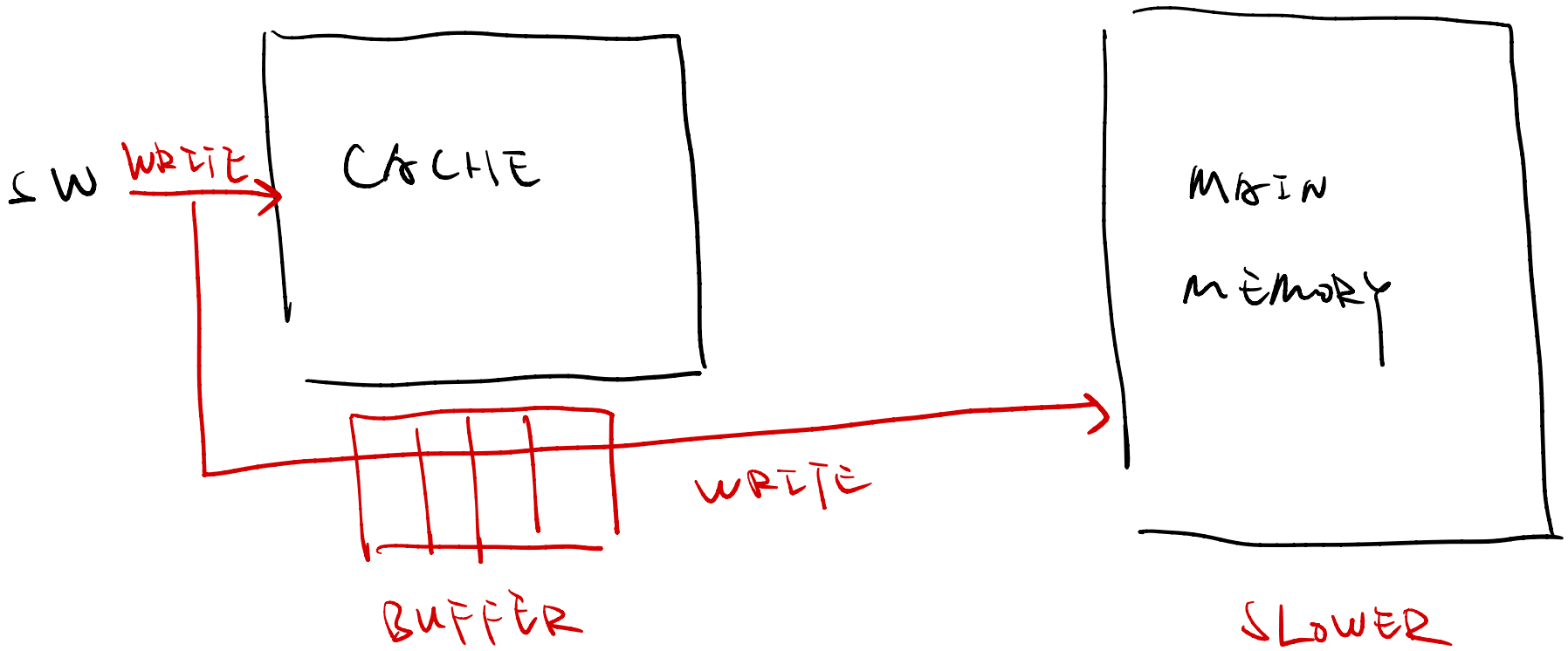
SLOWER

MISS
PENALTY

WHAT ABOUT WRITE (e.g. SW)

UPDATE HERE

UPDATE HERE

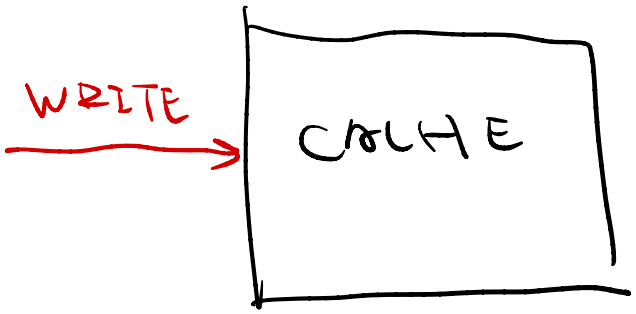


" WRITE-THROUGH "

ADD BUFFER TO HOLD ITEMS

FOR SLOWER MAIN MEMORY

ALTERNATE OPTION



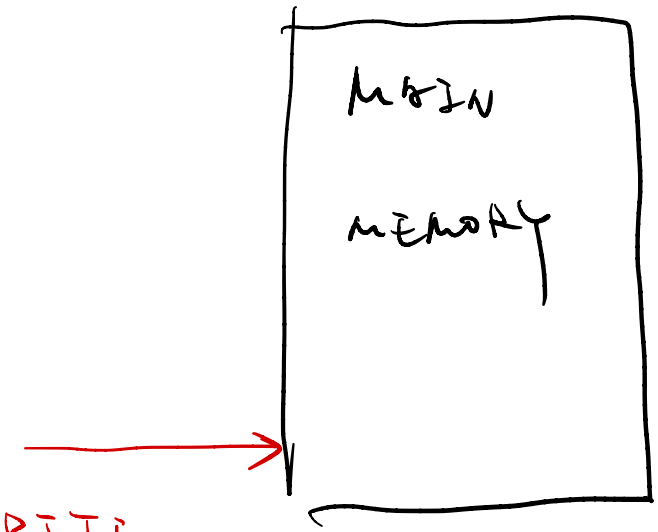
LOCALITY:

CACHE MAY

BE UPDATED

SEVERAL TIMES

```
for (i=0; i<N; i++)
    A[i] = 0;
```



WRITE

ONLY

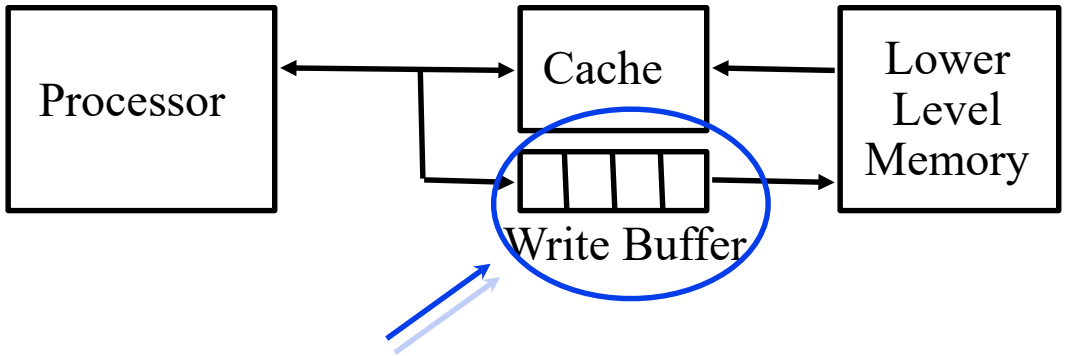
WHEN CACHE

NO LONGER NEEDS

BLOCKS.

e.g. END OF LOOP.

Write Buffers for Write-Through Caches



Holds data awaiting write-through to lower level memory

Q. Why a write buffer ?

A. So CPU doesn't stall

Q. Why a buffer, why not just one register ?

A. Bursts of writes are common.

Q. Are Read After Write (RAW) hazards an issue for write buffer?

A. Yes! Drain buffer before next read, or send read 1st after check write buffers.

Outline

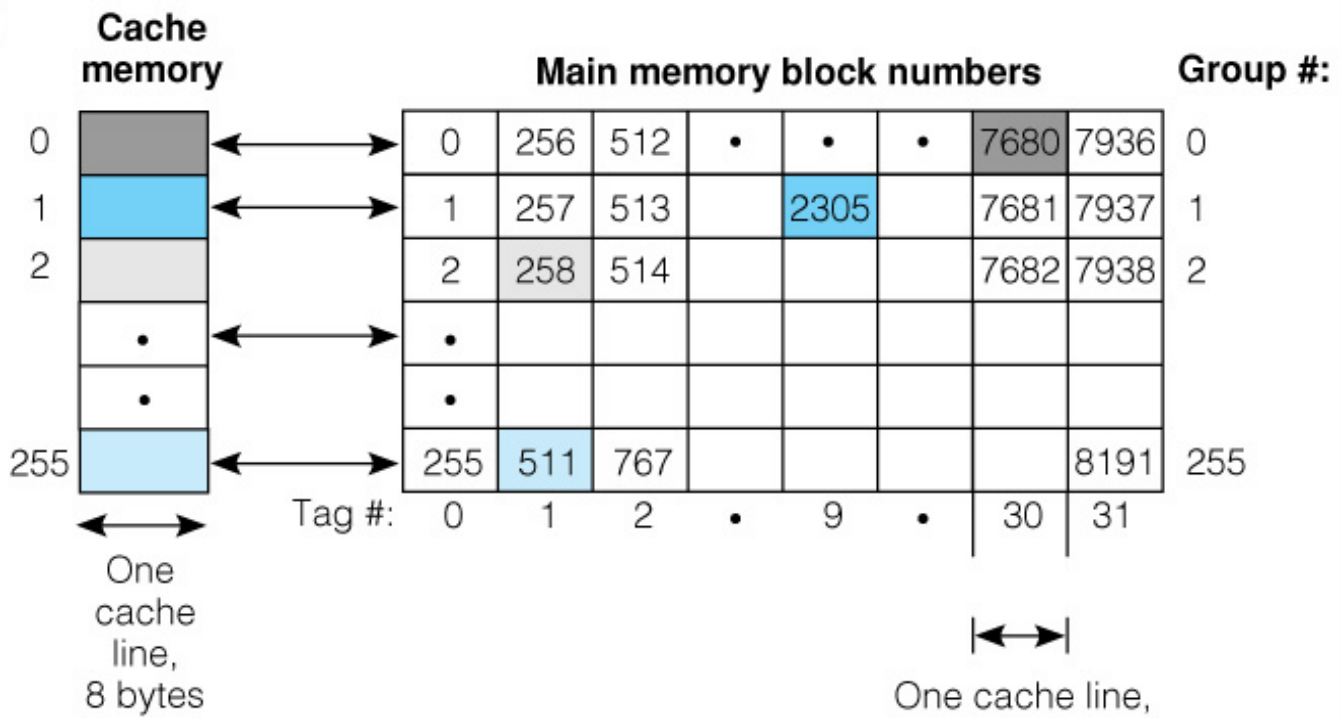
- B.1 Introduction
- B.2 Cache Performance
- B.3 Basic Cache Optimizations
- B.4 Virtual Memory

Types of Misses

- Compulsory
  - first access to a block will not be in cache
- Capacity
  - Cache does not contain all the blocks needed during program execution
- Conflict
  - For direct or set-associative caches
  - Two blocks map to same cache line

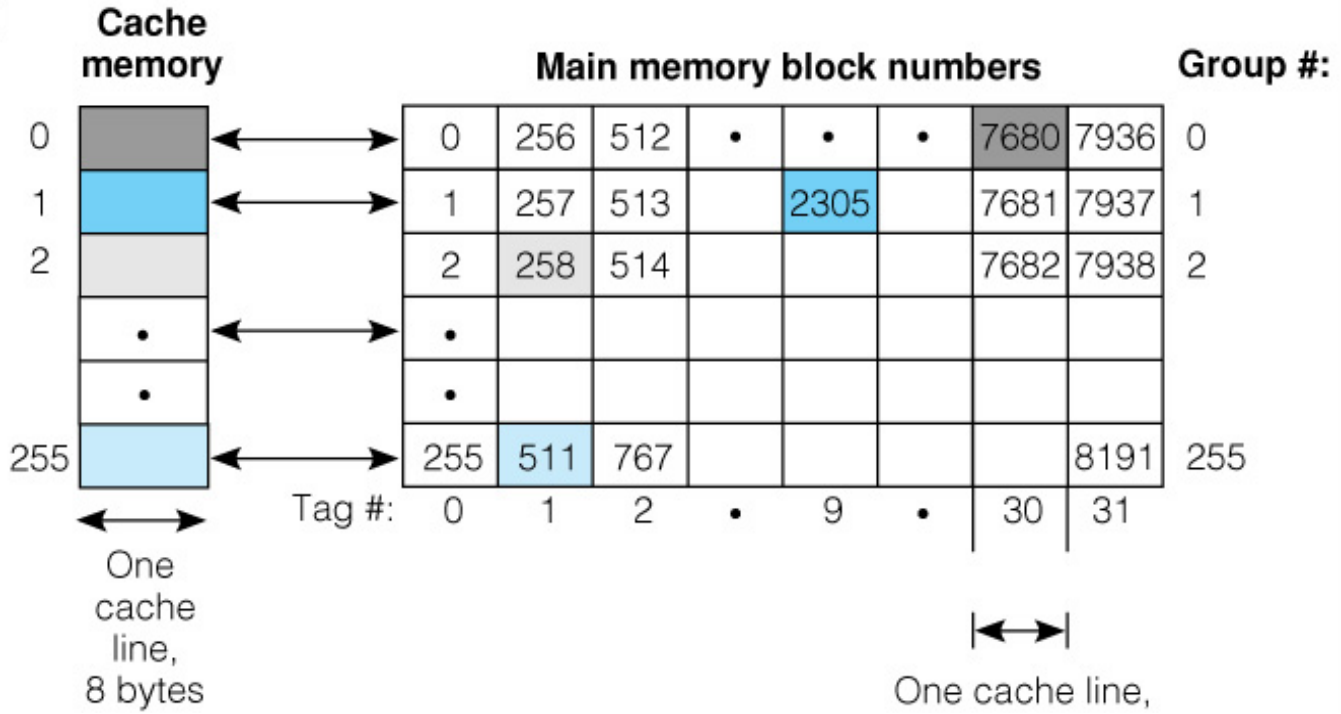
Compulsory Miss

When program starts execution, cache is initially empty



All initial memory references will be misses, as data needs to be loaded from main memory into cache

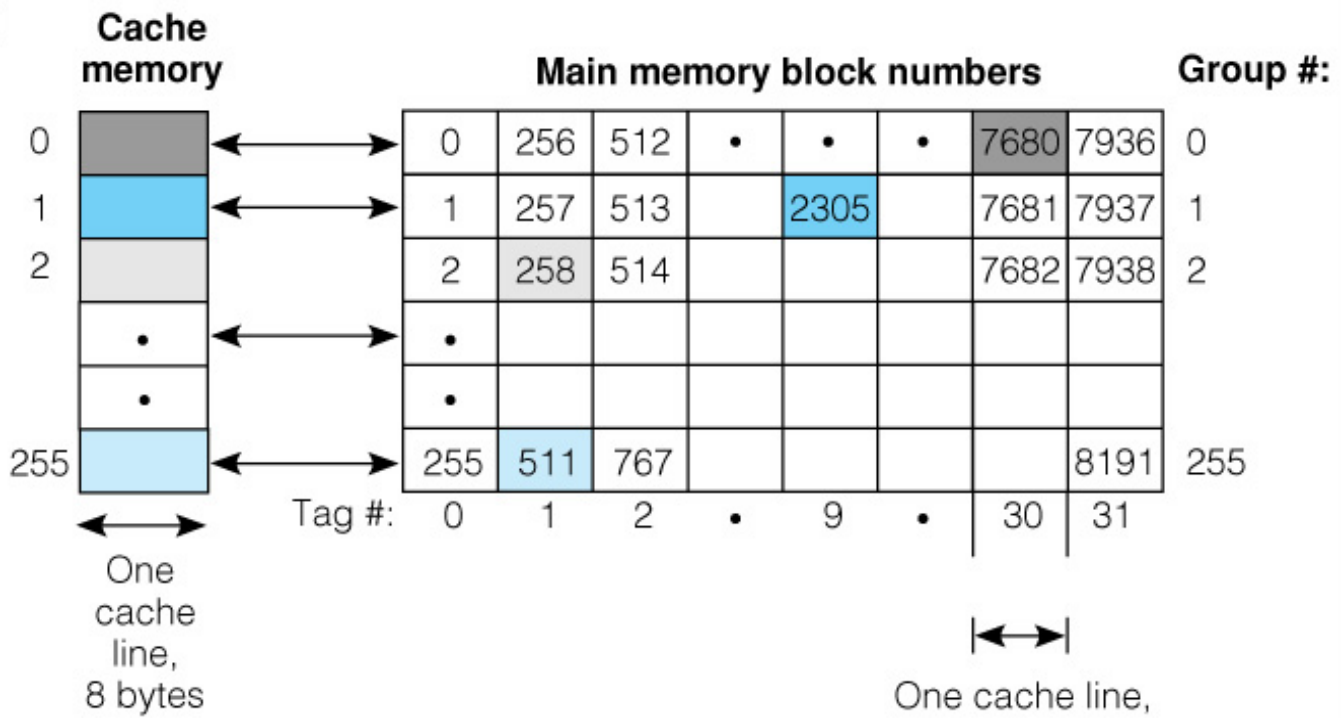
Capacity Miss



Number of blocks required to execute program will likely be larger than number of blocks that can be held by cache, resulting in capacity misses

Conflict Miss

Direct-mapped cache



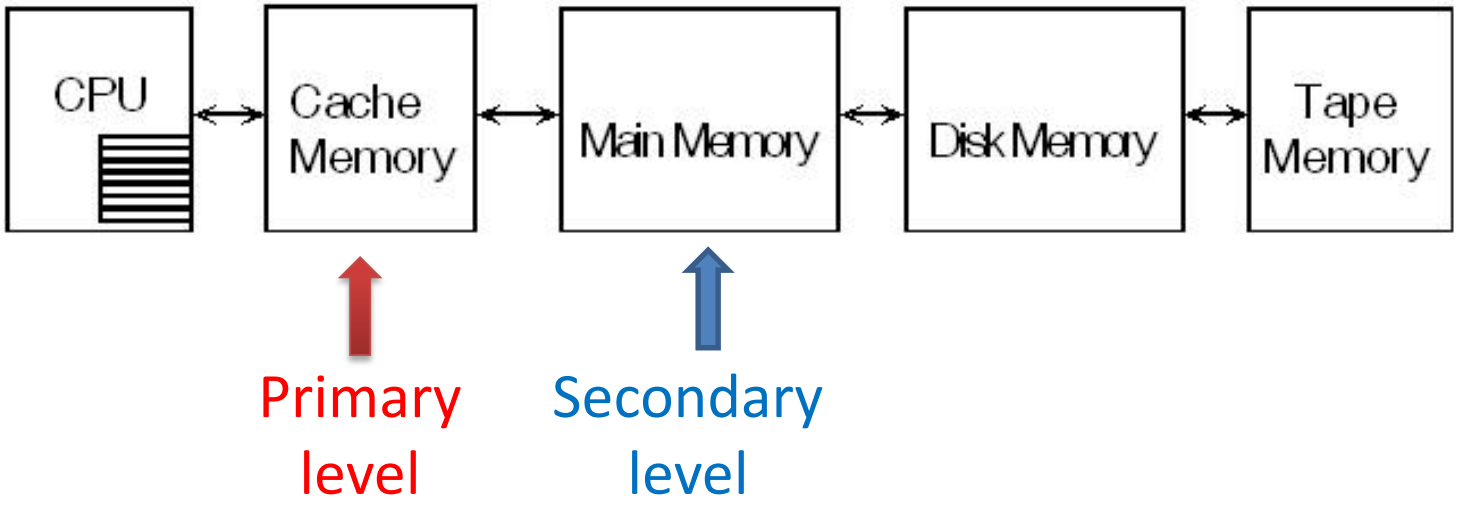
If blocks 255, 511, and 767 are all referenced by CPU, this results in a conflict miss

Outline

- B.1 Introduction
- B.2 Cache Performance
- B.3 Basic Cache Optimizations
- B.4 Virtual Memory

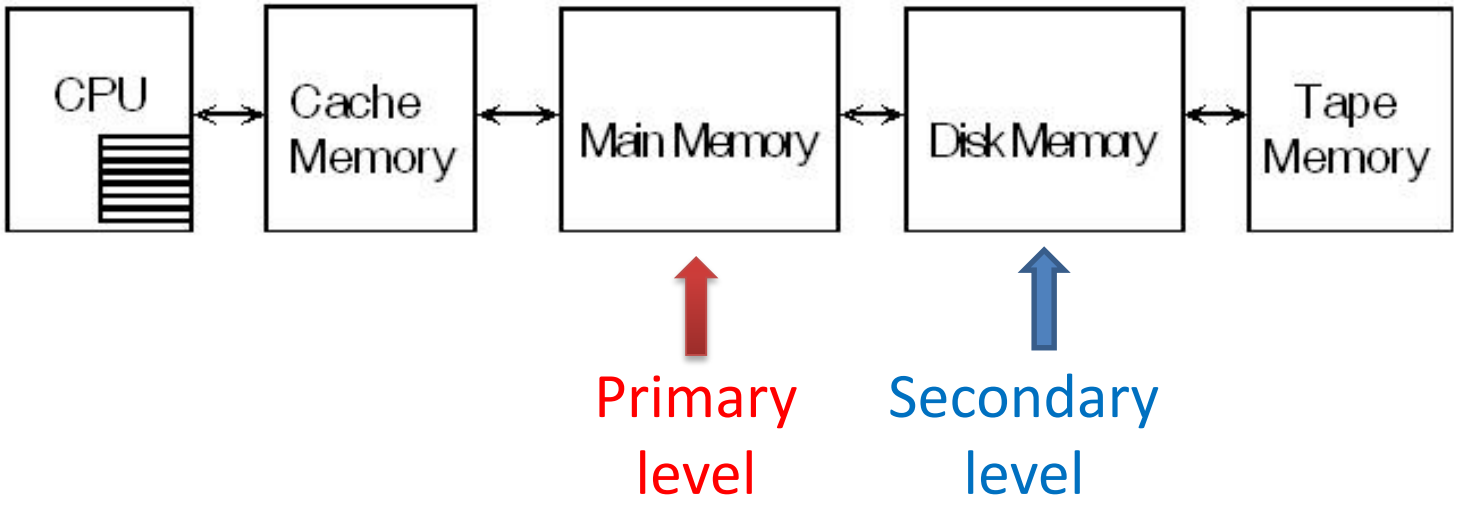
Overview

- Recall memory hierarchy
  - Cache = intermediate memory between CPU and main memory



Overview

- Virtual Memory (VM)
  - Main Memory acts like a “cache” for Disk Memory



Virtual Memory

- Use main memory as a “cache” for secondary (disk) storage
  - Managed jointly by CPU hardware and the operating system (OS)
- E.g., assume that you have Word and Excel open on your laptop
  - These programs and the data files (document, spreadsheet) are *permanently* stored in secondary storage (disk drive)
- However, when the programs are active, **copies** of the program and data are stored and run in main memory
  - since main memory is faster than secondary memory

VM Page Sizes

- Page sizes are usually larger than cache block sizes
  - since it takes longer to access secondary memory (disk drive), we want to grab a large chunk of instructions/data for each access.
- Typical page sizes are 1K to 16K bytes
  - recall cache block size may be 8 or 16 bytes

Virtual Memory Addressing

- Addressing VM is somewhat similar to cache addressing
- Recall for fully associative memory example: we had a 16-bit address word
  - Cache block size was 8 B, so 3 bits to address each byte in the block
  - Remaining 13 bits used for *tag* field to identify specific block address in main memory, and determine if copy located in cache

VM Addressing

- As an example, assume we have a 32-bit address word, and VM with page size of 4KB
- 4KB = $2^2 2^{10}$ so we need 12 bits to address each byte in the page.
  - This is called **page offset** for VM - similar to byte address field in cache.
- We have 20 bits left over – used for page addressing

VM Addressing

- The CPU issues a **virtual address**:
  - contains **virtual page number** and **page offset** fields
- An address translator translates the *virtual page number* to a **physical page number**
- The *physical page number* and *offset fields* combine to form a **physical address**
- The *physical address* can either be main memory (**page hit**) or disk memory (**page miss** or **page fault**)

Address Translation