

CS/ECE 5381/7381  
Computer Architecture  
Spring 2023

Dr. Manikas  
Computer Science  
Lecture 12: Mar. 7, 2023

# Review of Memory Hierarchy

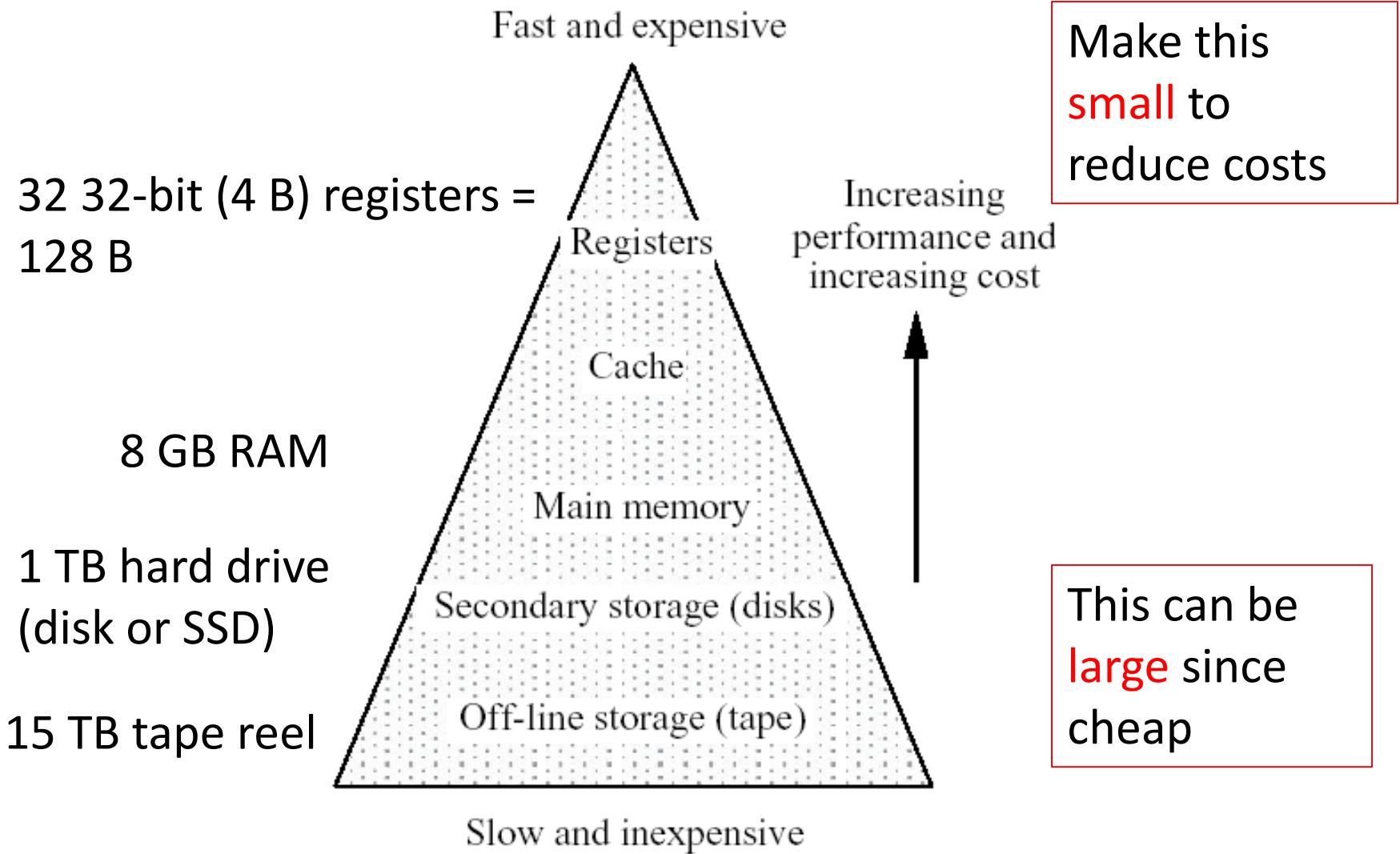
(Appendix B, Hennessy and Patterson)

Note: some course slides adopted from  
publisher-provided material

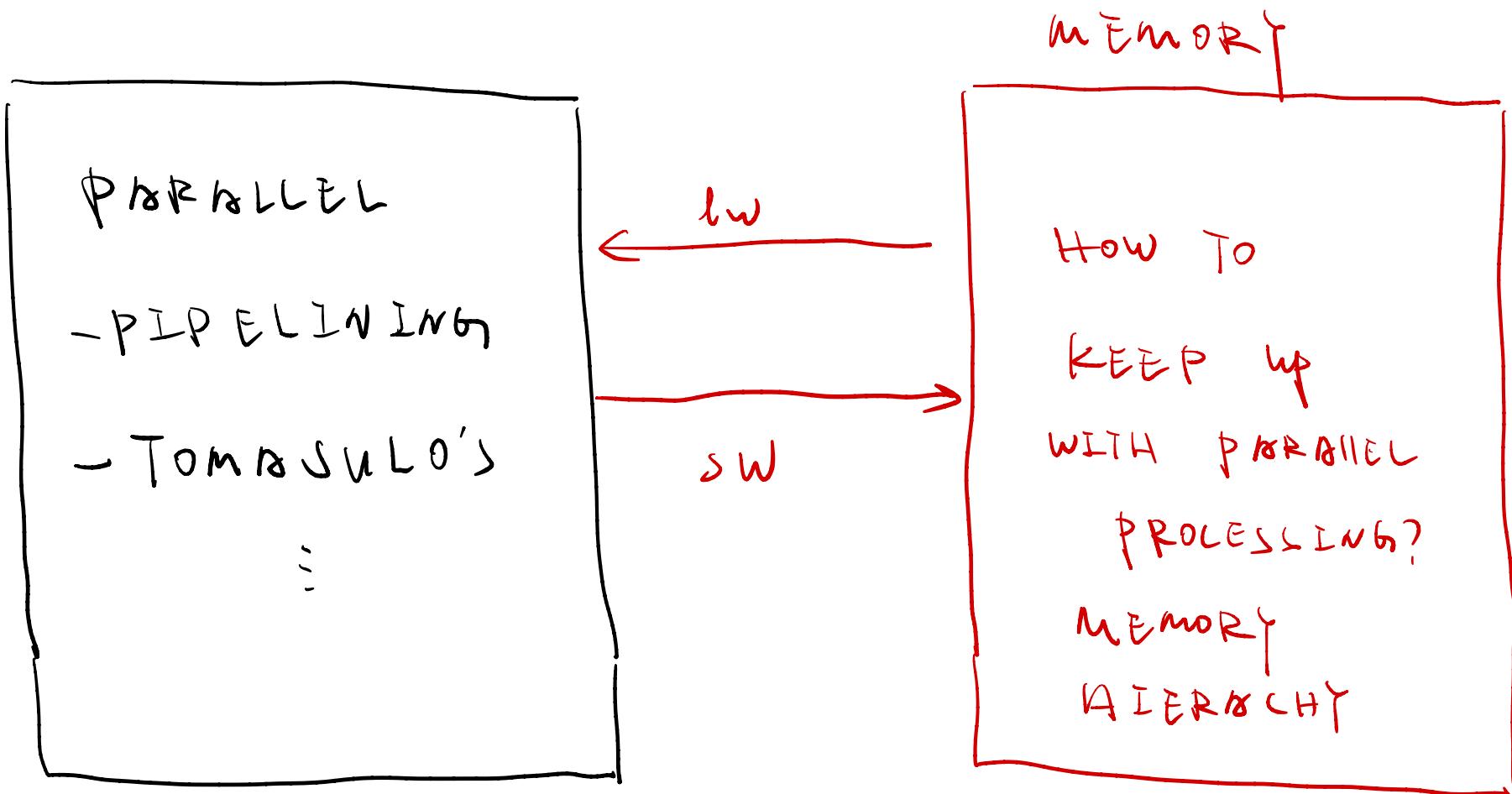
# Outline

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

# Memory Hierarchy



(MIPS)  
PROCESSOR



# Memory Hierarchy

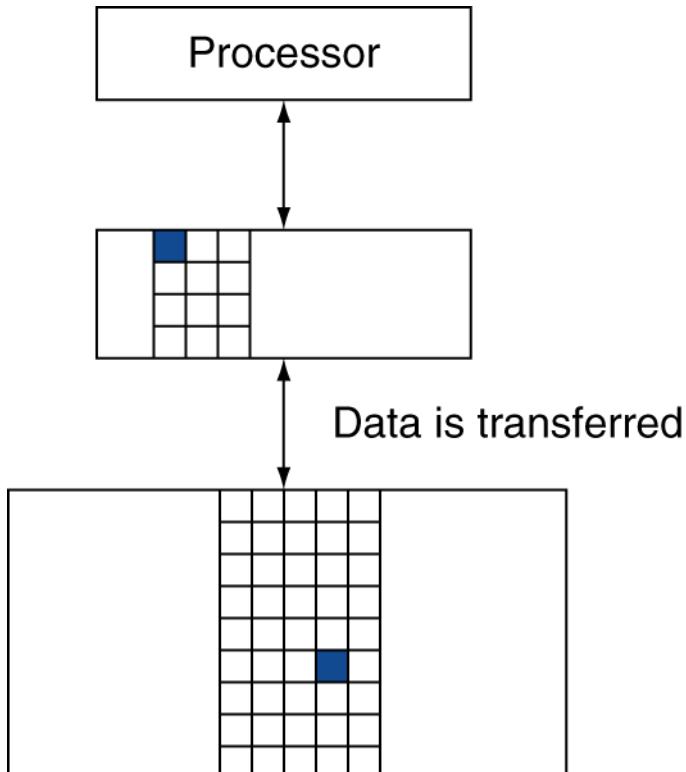
- Ideal memory
  - Fast access time, large capacity, low cost
- We don't have this in any *one* component
- But we can use a *combination* of components to “appear” as “ideal memory” to CPU
  - This combination is called a *memory hierarchy*
  - Use the “principle of locality”

# Principle of Locality

- Programs access a small proportion of their address space at any time
- **Temporal locality**
  - Items accessed recently are likely to be accessed again soon
  - e.g., instructions in a loop, instruction variables
- **Spatial locality**
  - Items near those accessed recently are likely to be accessed soon
  - E.g., sequential instruction access, array data

# Memory Hierarchy Levels

- Block (aka line): unit of copying
  - May be multiple words
- If accessed data is present in upper level
  - Hit: access satisfied by upper level
    - Hit ratio: hits/accesses
- If accessed data is absent
  - Miss: block copied from lower level
    - Time taken: miss penalty
    - Miss ratio: misses/accesses  
 $= 1 - \text{hit ratio}$
  - Then accessed data supplied from upper level



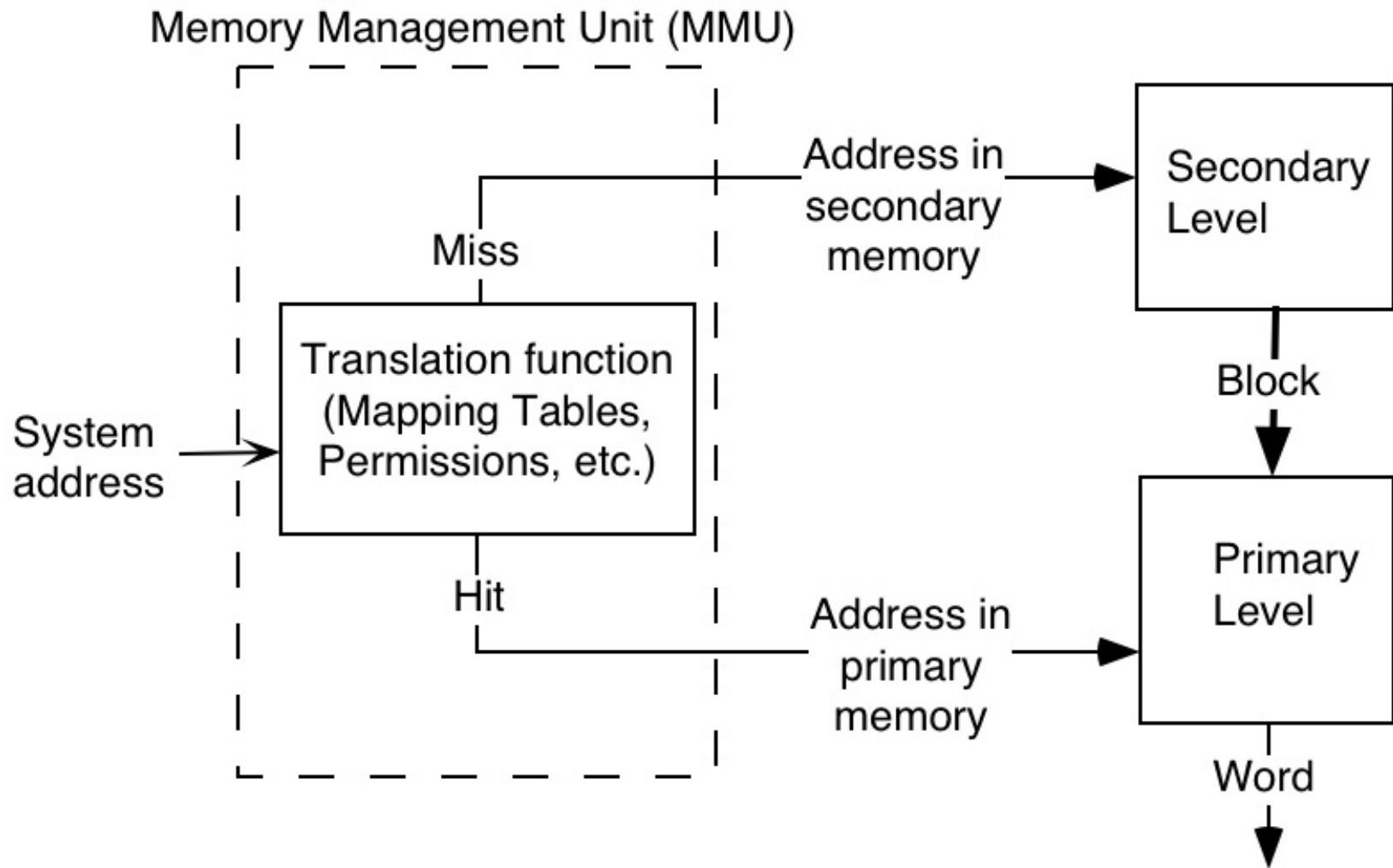
# Memory Hierarchy: Terminology

- **Hit:** data appears in some block in the upper (primary) level
  - **Hit Rate:** the fraction of memory access found in the upper level
  - **Hit Time:** Time to access the upper level which consists of:  
Primary level access time + Time to determine hit/miss

# Memory Hierarchy: Terminology (cont)

- **Miss**: data needs to be retrieved from a block in the lower (secondary) level
  - **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 << Miss Penalty

# Addressing and Accessing a 2-Level Hierarchy



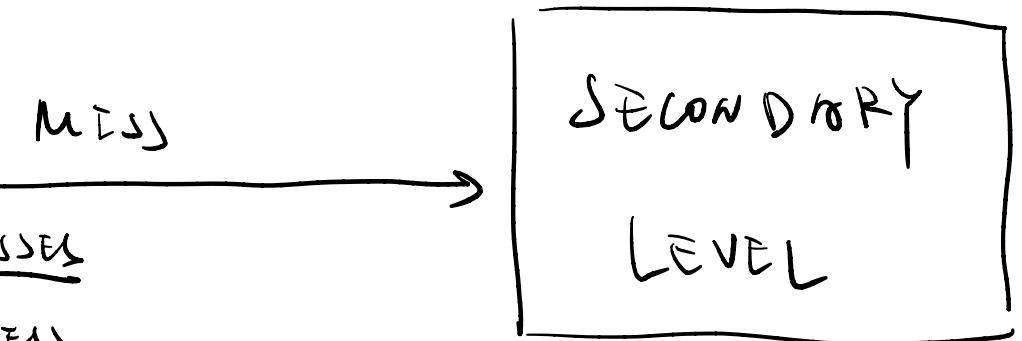
Access Times

$$\text{MISS RATIO} = \frac{\# \text{MISSES}}{\# \text{ACCESS}}$$

(1-h)

$$\frac{2}{10} = 0.2$$

means we have 2 miss

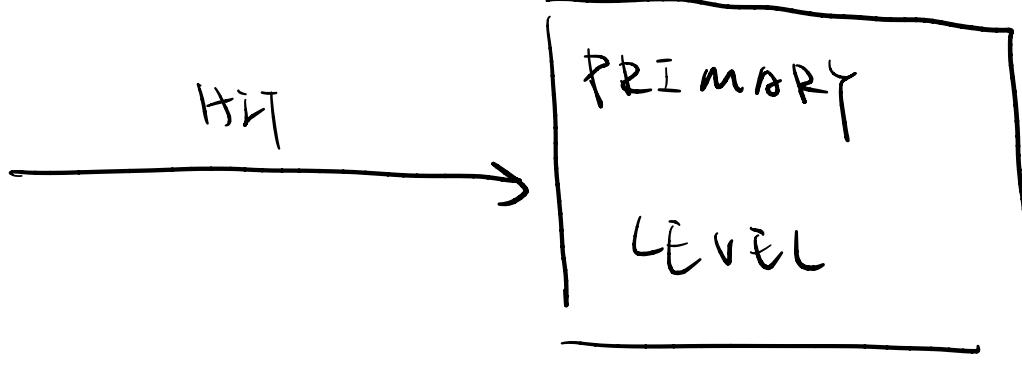


$t_s$

SLOWER, so  $t_s > t_p$

$$\text{HIT RATIO} = h$$

$$= \frac{\# \text{HITS}}{\# \text{ACCESS}}$$



$t_p$

$$\frac{8}{10} = h = 0.8$$

$$\text{AVERAGE ACCESS TIME} = h t_p + (1-h) t_s$$

# Hits and Misses

- **Miss ratio** = #misses / total # of refs in the same program-execution period
- **Hit ratio** = 1 - miss ratio
  - Let  $t_p$  = primary memory access time,
  - $t_s$  = secondary memory access time,
  - $h$  = hit ratio
- Then avg access time for two-level memory hierarchy  $t_a = ht_p + (1-h)t_s$
- The qty  $t_s > t_p$ , so design goal is to minimize miss ratio

# Example B.1-1

- Assume that we have a two-level memory hierarchy with the following memory access times:
  - Primary = 10 ns
  - Secondary = 100 ns (10 times slower)
- What is the **average access time** for this memory hierarchy if the hit rate is **90%** (0.9)?
- What is the **average access time** for this memory hierarchy if the hit rate is **75%** (0.75)?

GIVEN : PRIMARY ACCESS TIME = 10 ns

SECONDARY ACCESS TIME = 100 ns

$$\Rightarrow t_p = 10 \text{ ns} \quad t_s = 100 \text{ ns}$$

$$\begin{aligned} t_d &= h t_p + (1-h) t_s \\ &= h(10 \text{ ns}) + (1-h)(100 \text{ ns}) \end{aligned}$$

EFFECT OF HIT RATE  $h$  ?

① Let HIT RATE = 90%  $\Rightarrow h = 0.9$

$$t_d = (0.9)(10 \text{ ns}) + (0.1)(100 \text{ ns})$$

$$= 9 \text{ ns} + 10 \text{ ns} = 19 \text{ ns}$$

⑤ Let HIT RATE = 75%  $\Rightarrow h = 0.75$

$$t_d = (0.75)(10 \text{ ns}) + (0.25)(100 \text{ ns})$$

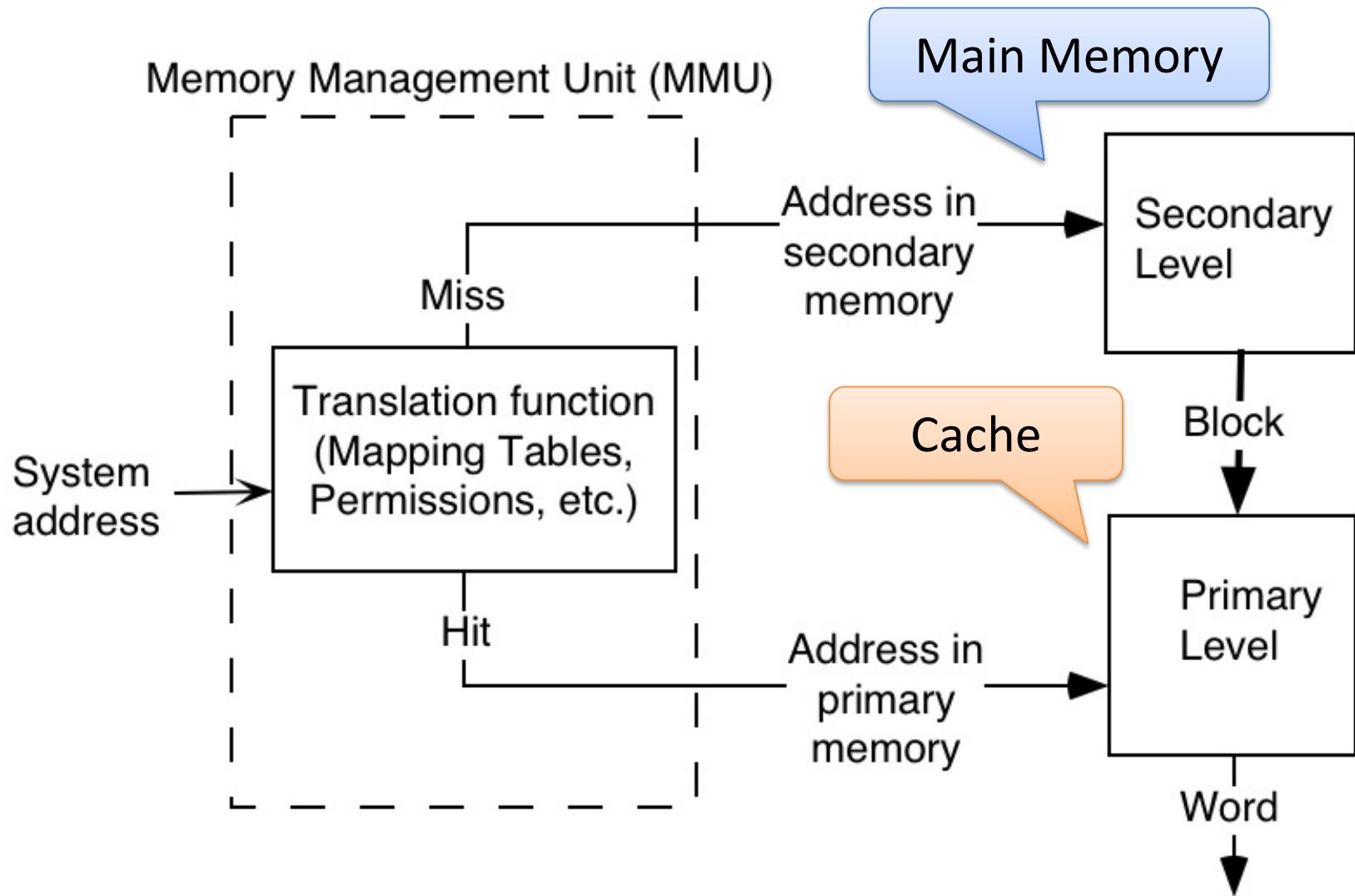
$$= 7.5 \text{ ns} + 25 \text{ ns}$$

$$= 32.5 \text{ ns}$$

# Cache

- A fast memory (RAM) used to hold active part of program and data
- Speeds up operation of active program
- Often incorporated into CPU
- Operates on *Principle of Locality* (recall from earlier)
  - Part of *2-level memory hierarchy*

# Addressing and Accessing a 2-Level Hierarchy



# Interpreting Memory Addresses

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

## MEMORY SIZES

$$8 \text{ GB RAM} \Rightarrow 2^3 2^{30} \text{ B} = 2^{33} \text{ B}$$

$$256 \text{ GB SSD} \Rightarrow 2^8 2^{30} \text{ B} = 2^{38} \text{ B}$$

# CPU Memory Address (no cache)

Assume we have a main memory with 64K bytes. If the processor accesses main memory directly (**no cache**), then it needs an address bus with enough lines to access all 64K bytes.

$$64K = 2^6 2^{10} = 2^{16} \text{ bytes} \quad \Rightarrow \quad 16\text{-bit address}$$

MSB = bit 15

LSB = bit 0

16

## MEMORY SIZES

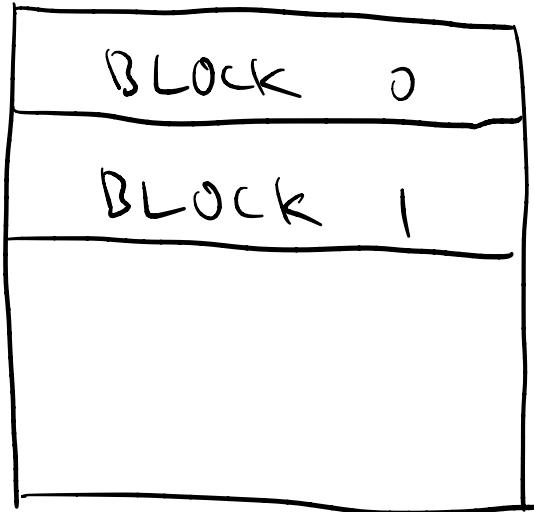
$$8 \text{ fm } RAM \Rightarrow 2^3 \cdot 2^{30} B = 2^{33} B$$

ADDRESS BUS SIZE? 33 BITS

$$256 \text{ GB SSD} \Rightarrow 2^8 \cdot 2^{30} B = 2^{38} B$$

ADDRESS BUS SIZE? 38 BITS

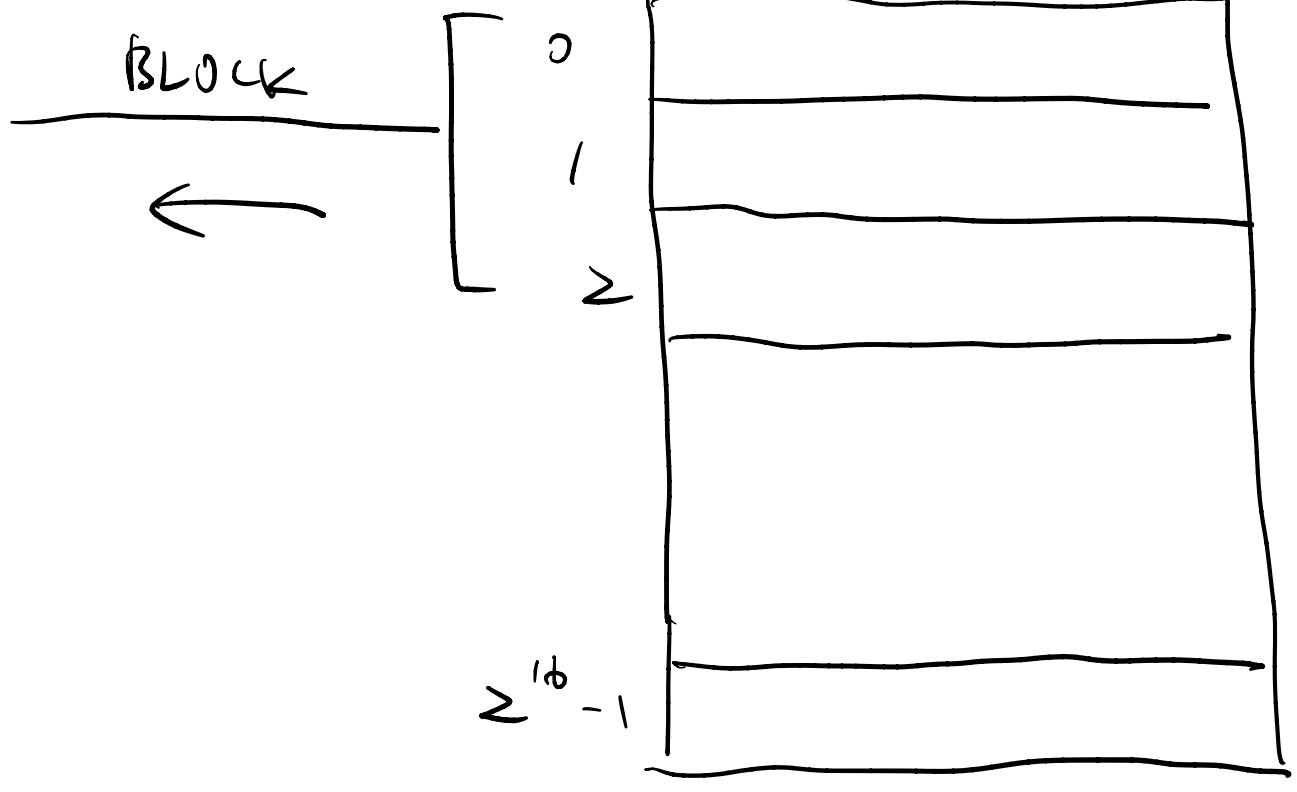
CACHE



GROUP  
OF BYTES

MAIN MEMORY

16 KB ( $2^{16}$  B)



SPATIAL LOCALITY

# CPU Memory Address (with cache)

Now add a **cache** between the processor and main memory.

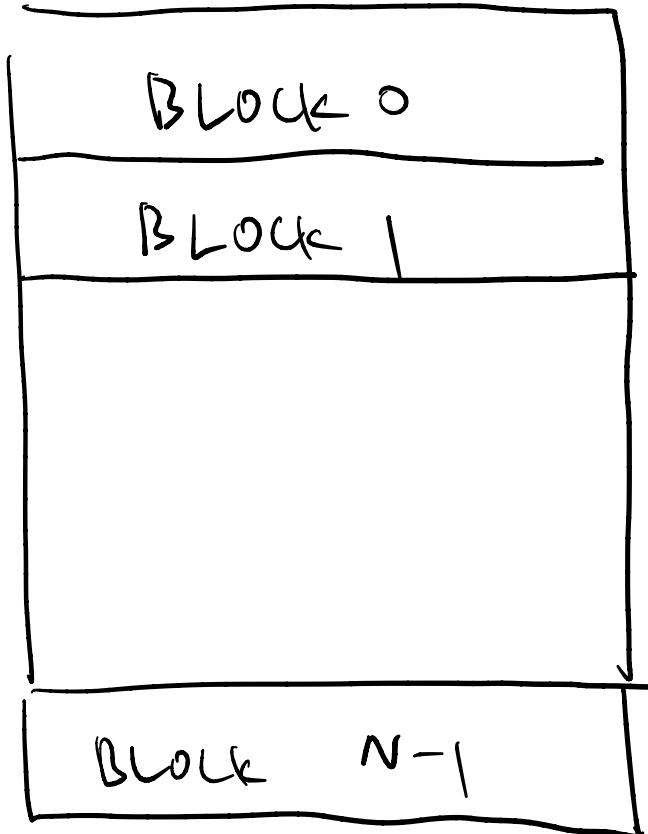
Cache views main memory as **blocks** of bytes. For our example, assume that each block contains 8 bytes.

Then, the cache views our 64K byte main memory as follows:

$$\frac{\frac{64KB}{B}}{8 \frac{block}{block}} = \frac{2^{16}}{2^3} = 2^{13} \text{ blocks}$$

So the main memory is viewed as having  $2^3 2^{10} = 8K$  blocks, or **8192 blocks**.

CACHE



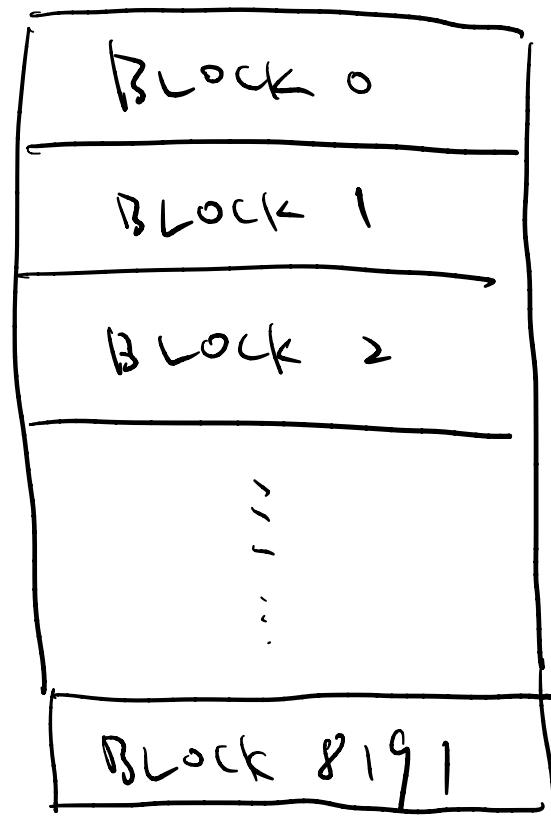
$N = \# \text{ of CACHE}$

BLOCKS < # OF

MAIN MEMORY BLOCKS

MAIN MEMORY

$$2^{16} B = 2^3 \text{ BLOCKS}$$



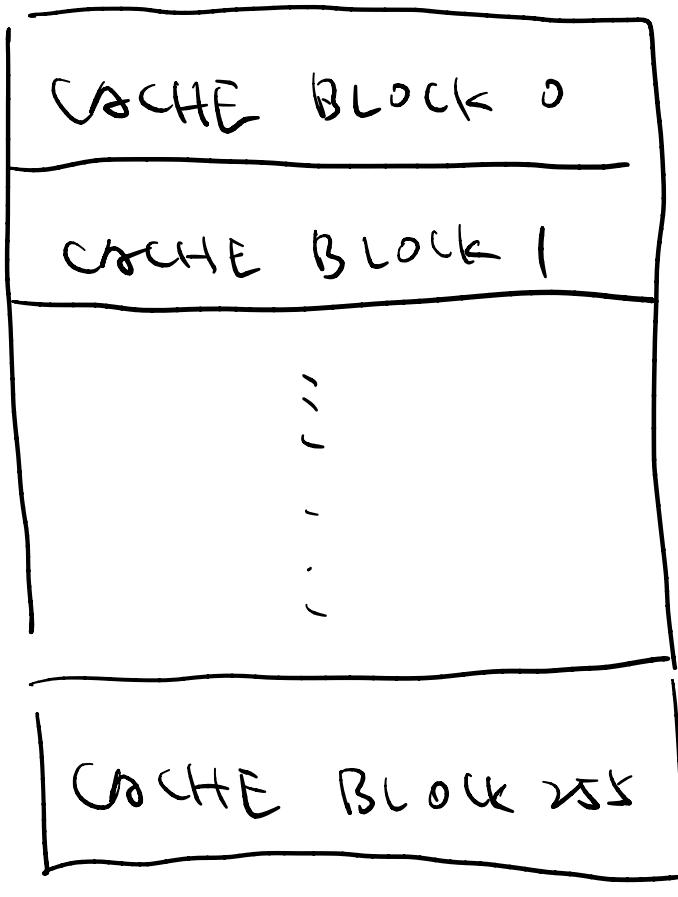
# Cache Size

Cache views main memory as a **group of blocks**. For our example, we have a group of 8K (8192) blocks in main memory.

The cache is much smaller than main memory (cache is faster, so more cost/bit). For our example, let the cache size be **256** ( $2^8$ ) blocks.

CACHE

> 56 BLOCKS

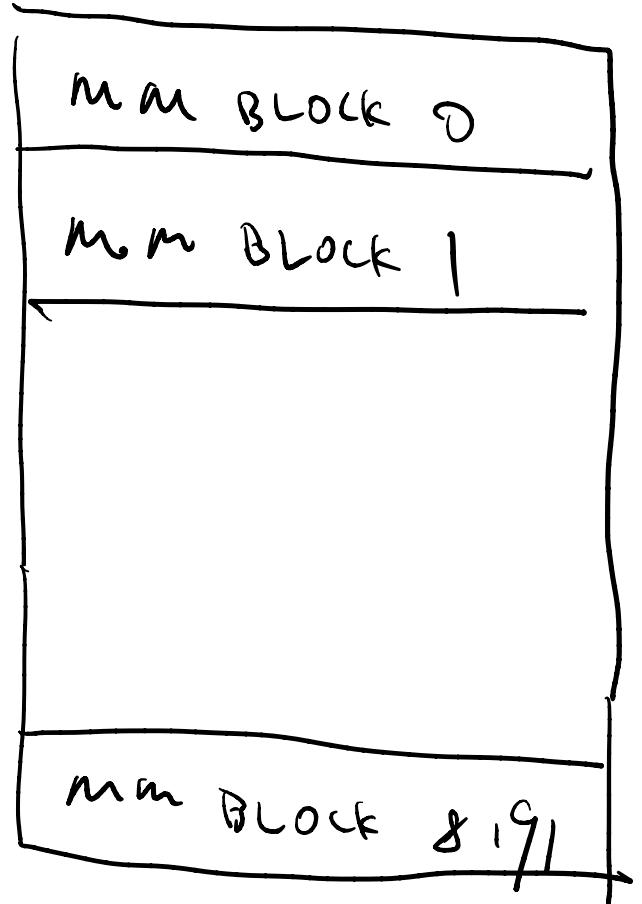


COPY  
BLOCK

"  
CACHE  
MAPPING"

Main memory

8k (8192) BLOCKS



## CACHE MAPPING

- WHERE TO COPY MM BLOCK INTO CACHE?
- HOW TO FIND BLOCK IN CACHE?  
(HIT / MISSES)

"CACHE MAPPING SCHEME"

# CPU Memory Address: Cache Mapping

- The CPU memory address is mapped to the cache
- Mapping scheme depends on the **type** of cache
  - Associative
  - Direct
  - N-way Set-Associative

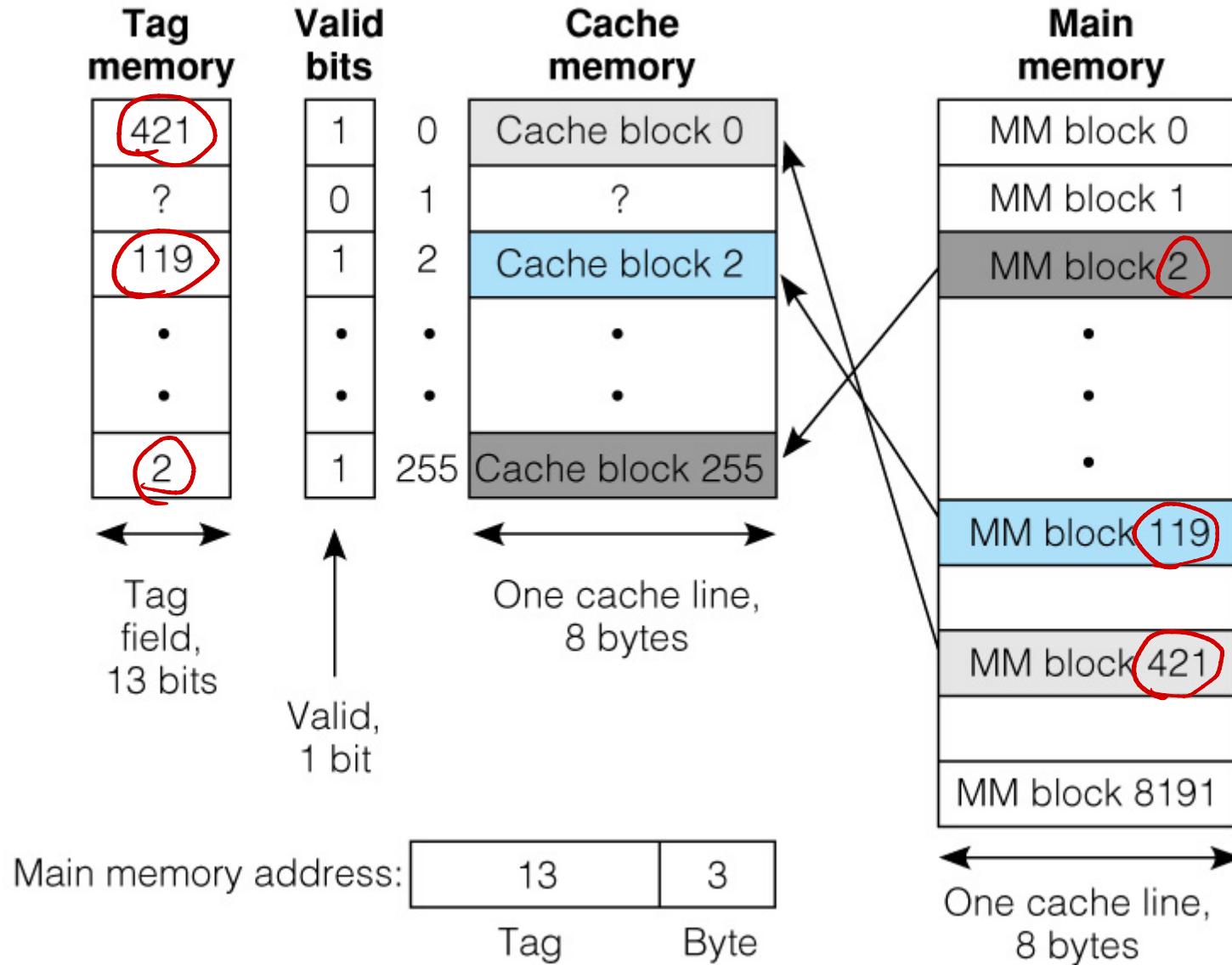
# Associative Cache

- Cache views main memory as a **list** of blocks
- Blocks from main memory may be placed anywhere in cache
- Need **tags** to identify original main memory block location
- We have  $8192 = 2^{13}$  main memory blocks
  - Need 13 bits for tag field
  - Other 3 bits identify byte in block (8 bytes/block)

# Tags and Valid Bits

- How do we know which particular block is stored in a cache location?
  - Store block address as well as the data
  - Actually, only need the high-order bits
  - Called the tag
- What if there is no data in a location?
  - Valid bit: 1 = present, 0 = not present
  - Initially 0

# Associative mapped caches

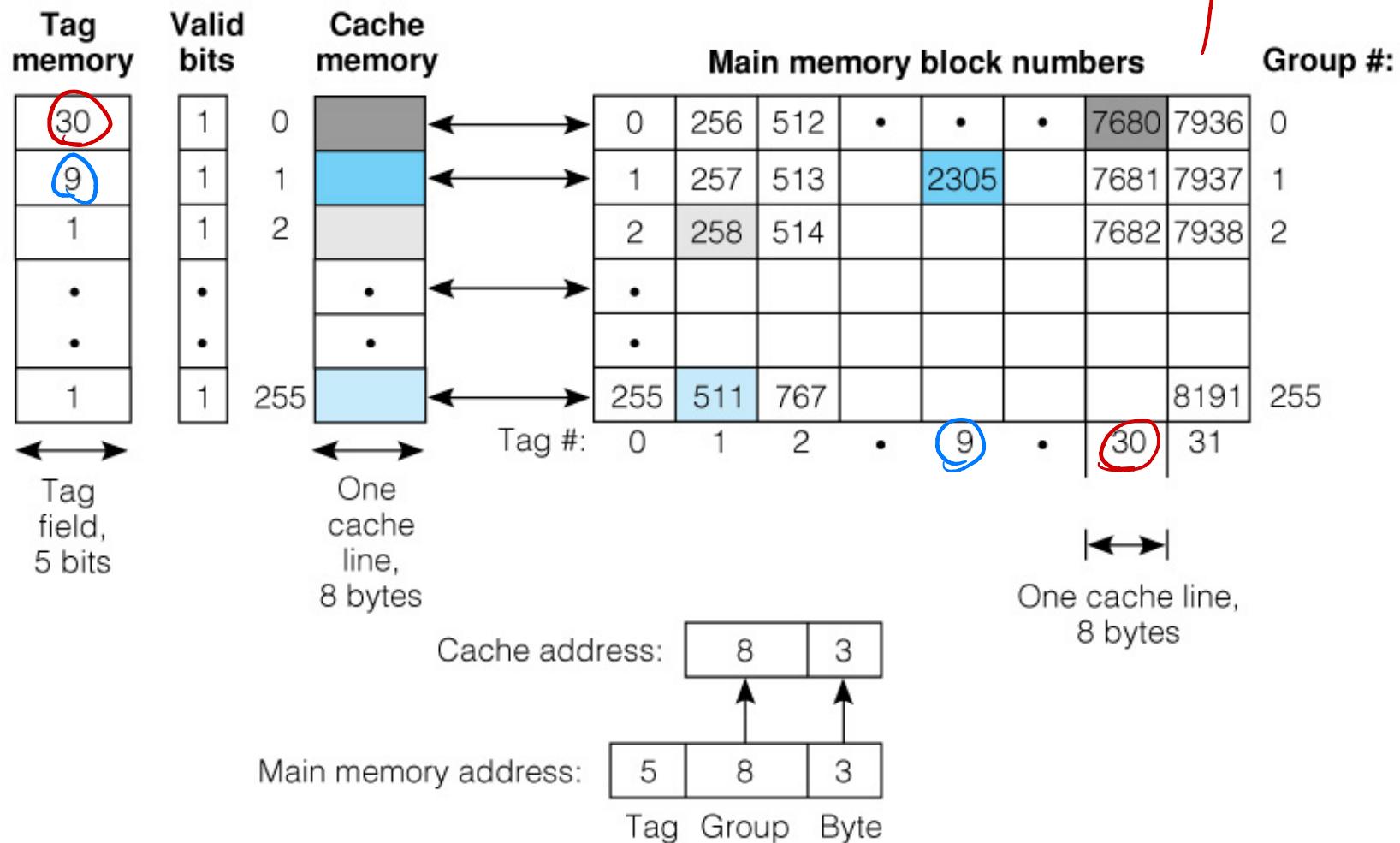


# Direct-mapped Cache

- Cache views main memory as an **array** of blocks
- Blocks from main memory are assigned to specific cache lines, based on array row
  - Number of cache lines = number of rows
- **Group** field identifies array **row**
- **Tag** field identifies array **column**

# Direct mapped cache

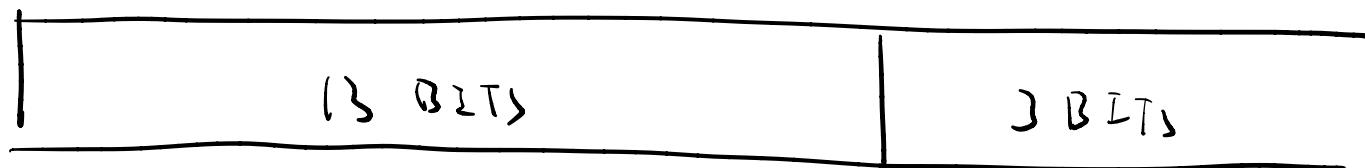
Only put one thing from  
a row at a time.



MEMORY ADDRESS FROM PROCESSOR



MAIN  
CACHE



$2^{13}$  BLOCKS

$2^3$  BYTES/BLOCK

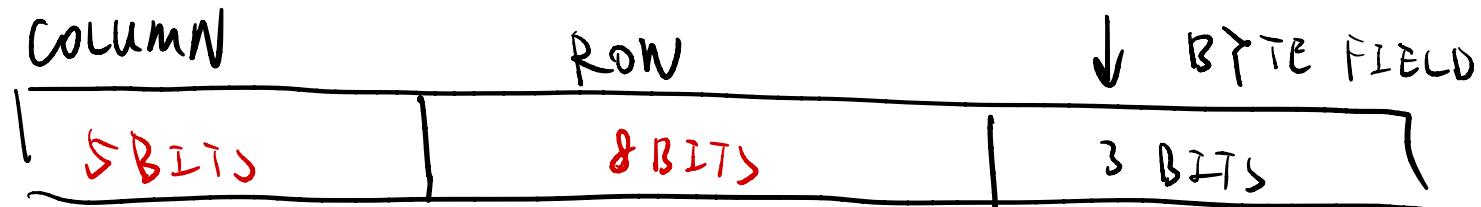
Tag FIELD

BYTE FIELD

in n BLOCK NUMBER

BYTE IN BLOCK

DIRECT  
MAPPED  
CACHE



$$32 = 2^5 \text{ cols} \quad 256 = 2^8 \text{ rows}$$

# Set-Associative Cache

- Cache views main memory as an **array** of blocks, similar to direct-mapped cache
- But, now cache is also arranged as an **array** of blocks
- **Set** field identifies main memory array **row**
- **Tag** field identifies main memory array **column**

# N-way Set-Associative Cache

- $N = 2, 4, 8 \dots$  (power of 2)
- Number of **sets** (main memory rows) = number of cache lines /  $N$ 
  - If  $N = 1$ , we have a direct-mapped cache
- For our example, if  $N=2$ , the 256-line cache becomes a  $128 \times 2$  cache array

We this way can copy two items from this row at the same time.

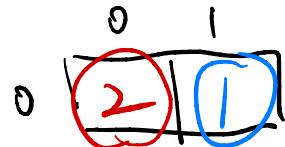
# 2-Way Set Associative Cache

Diagram illustrating the state of tag, cache, and main memory:

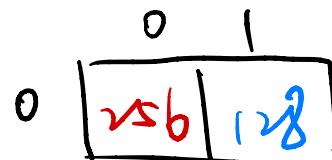
- tag**: Rows 0 to 127.
- cache**: 3 sets of 4 blocks (0-3).
- main memory (blocks)**: 8 blocks (0-7) with values: 0, 128, 256, 512, 7808, 1, 2, 0.

A red line highlights address 127 in both the cache and main memory sections.

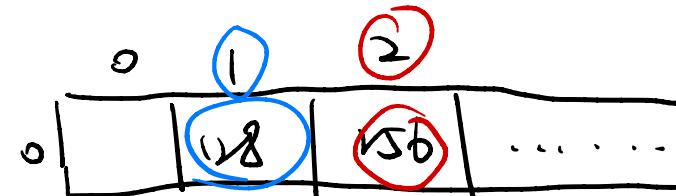
Tag



CACHE



MM

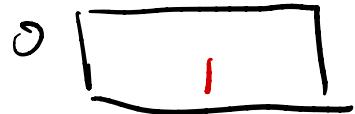


DIRECT

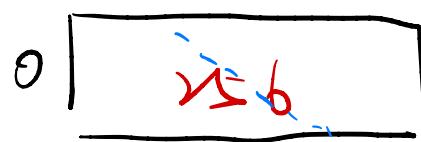
MAPPED

CACHE

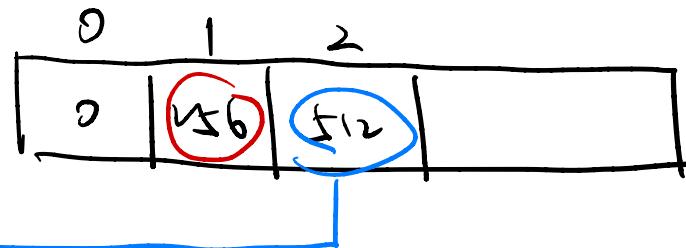
Tag



CACHE



MM



if want to copy 512, have  
to overwrite 256, because  
I can not copy two things

from the row at the same time.

## DIRECT MAPPED

TAG	GROUP	BYTE
5 BITS	8 BITS	3 BITS

$$2^5 = 32$$

$$2^8 = 256$$

COLUMNS

ROWS

2-way SET Assoc.

BYTE

6 BITS	7 BITS	3 BITS
--------	--------	--------

COLUMNS

ROWS

$$64 = 2^6$$

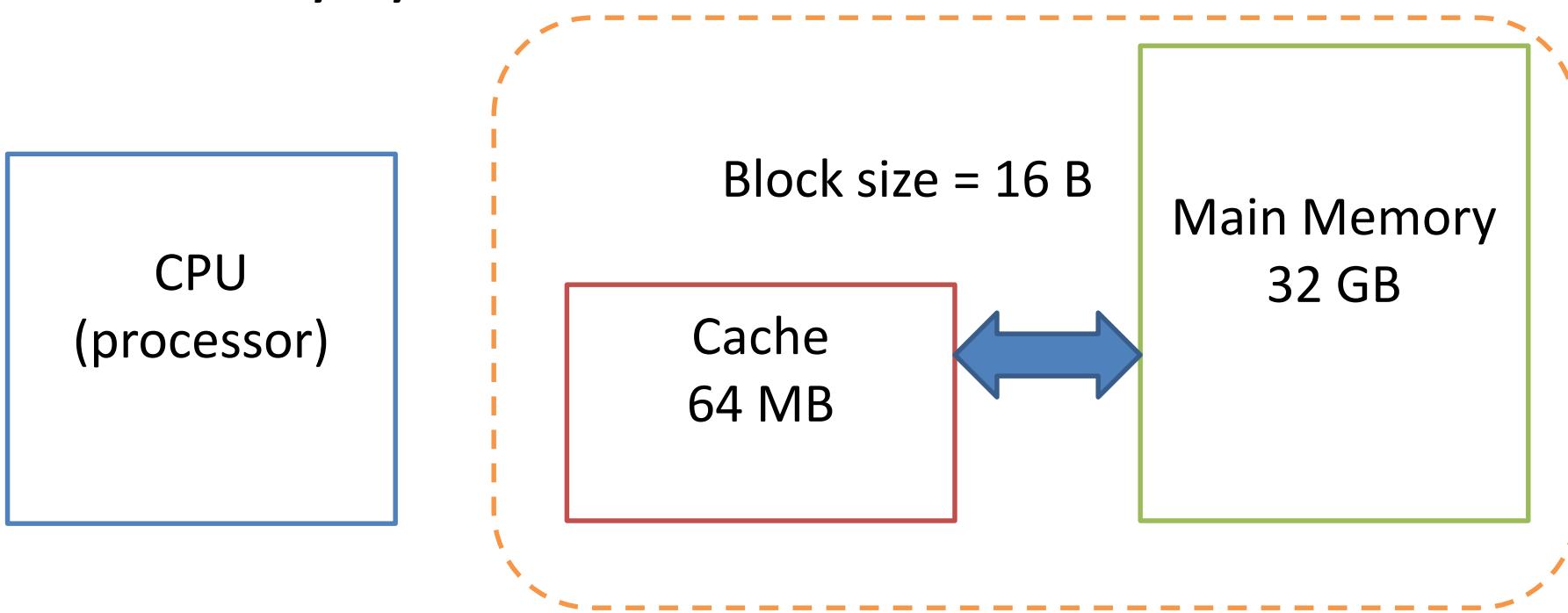
$$128 = 2^7$$

# Cache Fields

- Fields are Tag, Set, Byte
- Byte field is 3 bits as before (8 bytes/block)
- Set field identifies main memory row
  - 128 rows =  $2^7$ , so 7 bits
- Tag field identifies main memory column
  - 64 columns =  $2^6$ , so 6 bits

# Example B.1-2

- We have the following configuration for our memory system:



MAIN MEMORY SIZE = 32 GB

$$= 2^5 \cdot 2^{20} B = \boxed{2^{25} B}$$

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

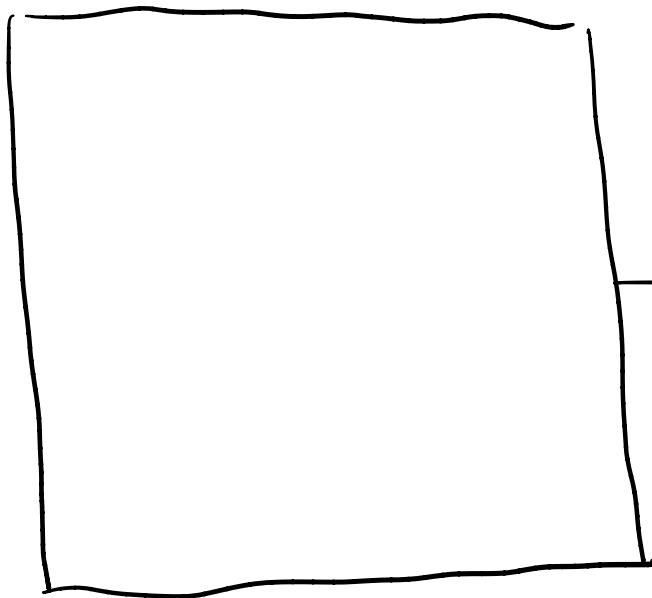
BLOCK SIZE = 16 B =  $\boxed{2^4 B}$

MEMORY ADDRESS SIZE ISSUED BY PROCESSOR?

⇒ NEEDS TO ACCESS ALL BYTES

MAIN MEMORY SIZE =  $2^{35} B \Rightarrow \underline{\underline{35 \text{ BITS}}}$

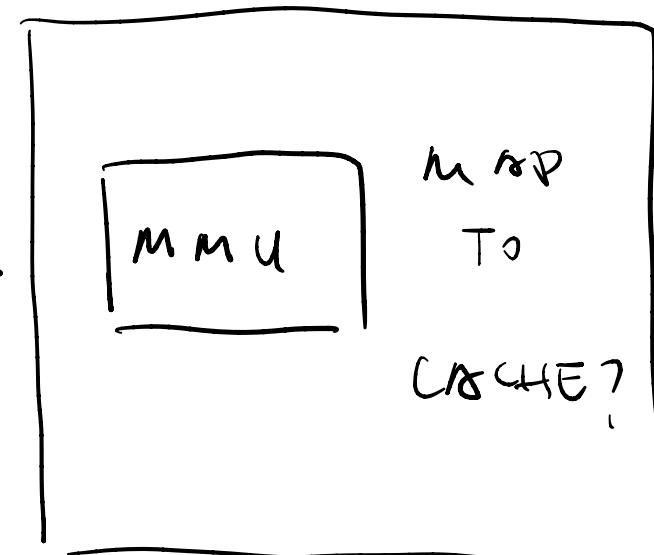
PROCESSOR



Memory  
ADDRESS

35  
(BITS)

MEMORY



## 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.

# 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 =

$$(\text{number of cache misses})(\text{miss penalty})$$

Number of cache misses =

$$(\text{instruction count})(\text{misses/instruction})$$

Misses/instruction =

$$(\text{memory accesses/instruction})(\text{miss rate})$$

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

# 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 =  $[(IC)(CPI) + \text{memory stall cycles}](\text{clock cycle time})$

## 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?

# 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)*