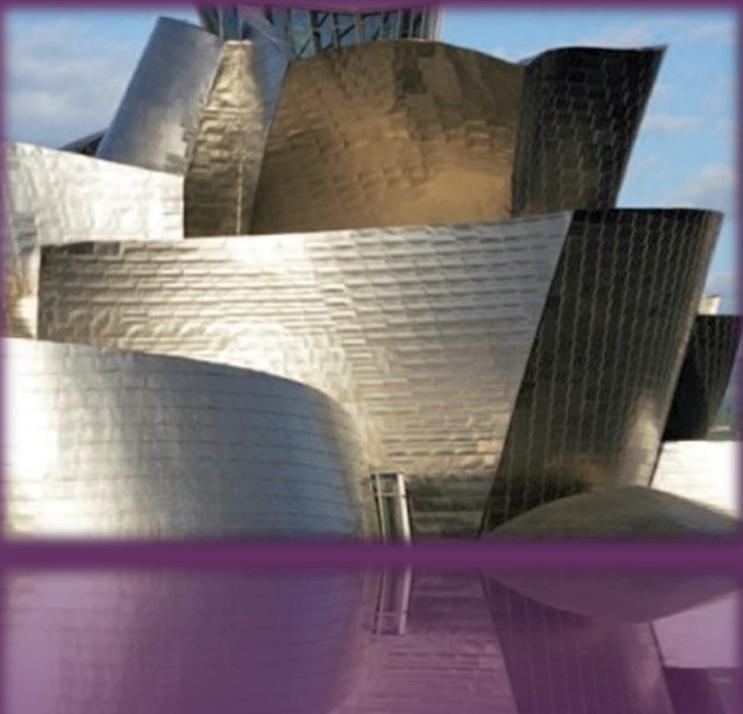


+



Chapter 4 ■ Cache Memory

William Stallings, Computer Organization and Architecture, 9th Edition

Objectives

- How are internal memory elements of a computer structured?
- After studying this chapter, you should be able to:
 - Present an overview of the main characteristics of computer memory systems and the use of a memory hierarchy.
 - Describe the basic concepts and intent of cache memory. Discuss the key elements of cache design.
 - Distinguish among direct mapping, associative mapping, and set-associative mapping.
 - Explain the reasons for using multiple levels of cache.
 - Understand the performance implications of multiple levels of memory.

Contents

4.1- Computer Memory Systems Overview

4.2- Cache Memory Principles

4.3- Elements of Cache Design

4.1- Computer Memory System Overview

- Characteristics of Memory System.
- The Memory Hierarchy

Key Characteristics of Computer Memory Systems

Table 4.1 Key Characteristics of Computer Memory Systems

Location	Performance
Internal (e.g., processor registers, cache, main memory)	Access time
External (e.g., optical disks, magnetic disks, tapes)	Cycle time
Capacity	Transfer rate
Number of words	Physical Type
Number of bytes	Semiconductor
Unit of Transfer	Magnetic
Word	Optical
Block	Magneto-optical
Access Method	Physical Characteristics
Sequential	Volatile/nonvolatile
Direct	Erasable/nonerasable
Random	Organization
Associative	Memory modules

Characteristics of Memory Systems

■ Location

- Refers to whether memory is internal and external to the computer
- Internal memory is often equated (make equal) with main memory
- Processor requires its own local memory, in the form of registers
- Cache is another form of internal memory
- External memory consists of peripheral storage devices that are accessible to the processor via I/O controllers

■ Capacity

- Memory is typically expressed in terms of bytes

■ Unit of transfer

- For internal memory the unit of transfer is equal to the number of electrical lines into and out of the memory module

Method of Accessing Units of Data

Sequential access

Memory is organized into units of data called records

Access must be made in a specific linear sequence

Access time is variable

Direct access

Involves a shared read-write mechanism

Individual blocks or records have a unique address based on physical location

Access time is variable

Random access

Each addressable location in memory has a unique, physically wired-in addressing mechanism

The time to access a given location is independent of the sequence of prior accesses and is constant

Any location can be selected at random and directly addressed and accessed

Main memory and some cache systems are random access

Associative

A word is retrieved based on a portion of its contents rather than its address

Each location has its own addressing mechanism and retrieval time is constant independent of location or prior access patterns

Cache memories may employ associative access

More details: Next slides

Method of Accessing Units of Data: Direct Access

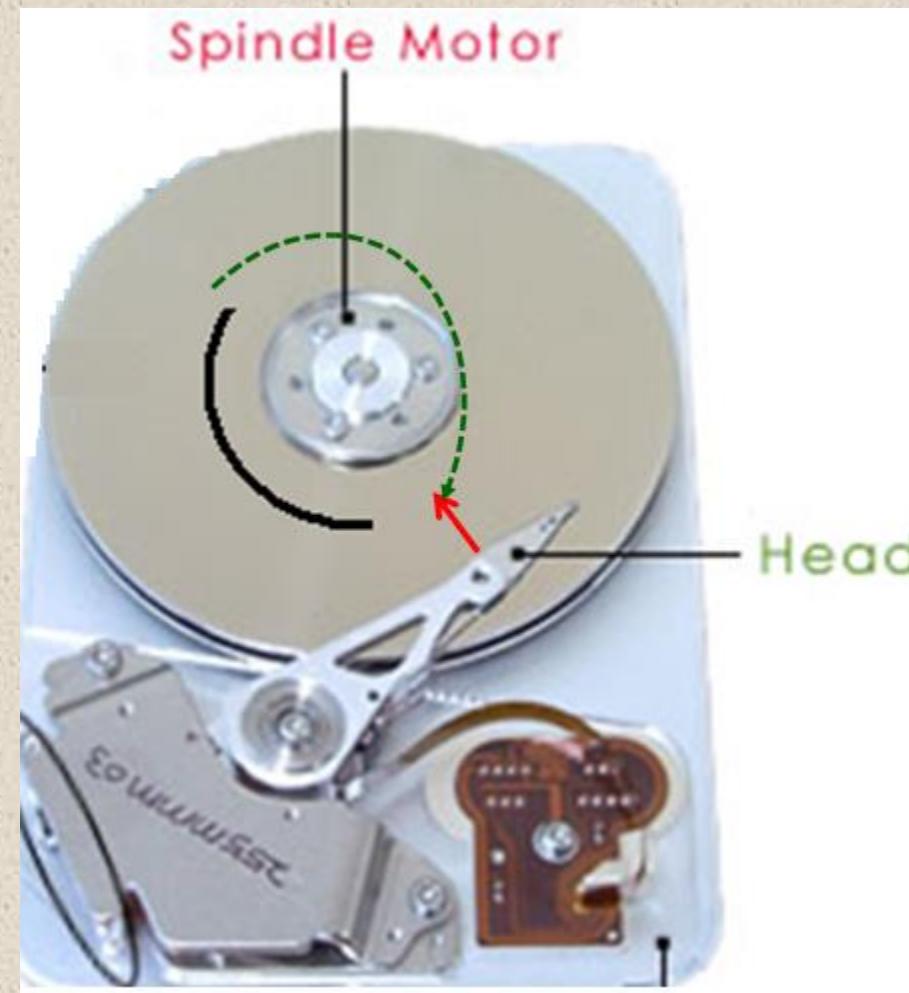
Location of each sector is identified by a unique number

T₁: seek time, time for moving the head to the accessed track

T₂: Rotational delay, time for rotating the disk to position the head to the beginning of the accessed sector

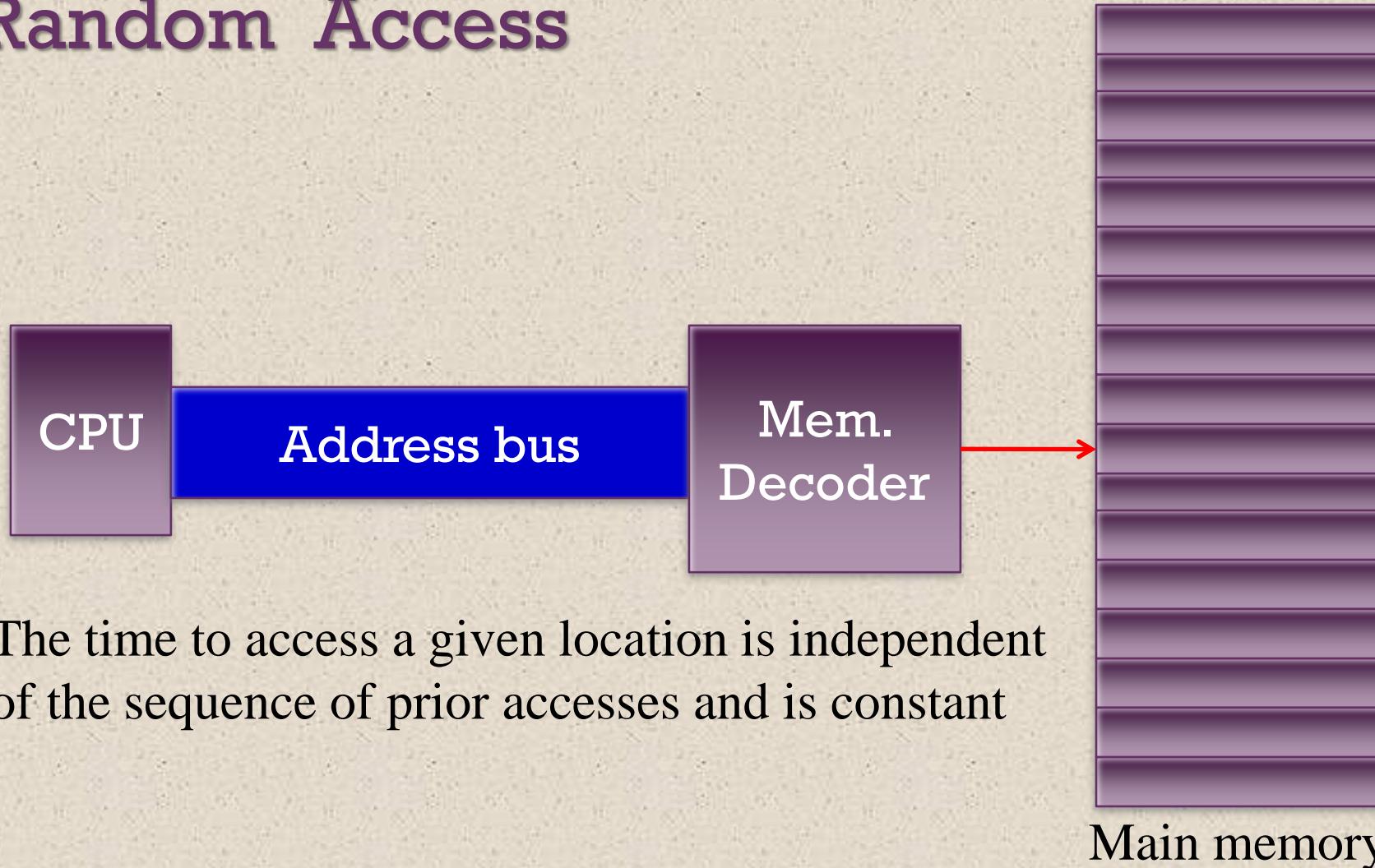
T₃: Transfer time, time for rotating the disk to access all the accessed sector

$$\text{Access time} = T_1 + T_2 + T_3$$



Each sector is accessed using different access time

Method of Accessing Units of Data: Random Access



Capacity and Performance:

The two most important characteristics of memory

Three performance parameters are used:

Access time (latency)

- For random-access memory it is the time it takes to perform a read or write operation
- For non-random-access memory it is the time it takes to position the read-write mechanism at the desired location

Memory cycle time

- Access time plus any additional time required before second access can commence
- Additional time may be required for transients to die out on signal lines or to regenerate data if they are read destructively
- Concerned with the system bus, not the processor

Transfer rate

- The rate at which data can be transferred into or out of a memory unit
- For random-access memory it is equal to $1/(cycle\ time)$

Memory



- The most common forms are:
 - Semiconductor memory
 - Magnetic surface memory
 - Optical
 - Magneto-optical
- Several physical characteristics of data storage are important:
 - Volatile memory
 - Information decays naturally or is lost when electrical power is switched off
 - Nonvolatile memory
 - Once recorded, information remains without deterioration until deliberately changed
 - No electrical power is needed to retain information
 - Magnetic-surface memories : Are nonvolatile
 - Semiconductor memory : May be either volatile or nonvolatile
 - Nonerasable memory
 - Cannot be altered, except by destroying the storage unit
 - Semiconductor memory of this type is known as read-only memory (ROM)
- For random-access memory the organization is a key design issue
 - Organization refers to the physical arrangement of bits to form words

+

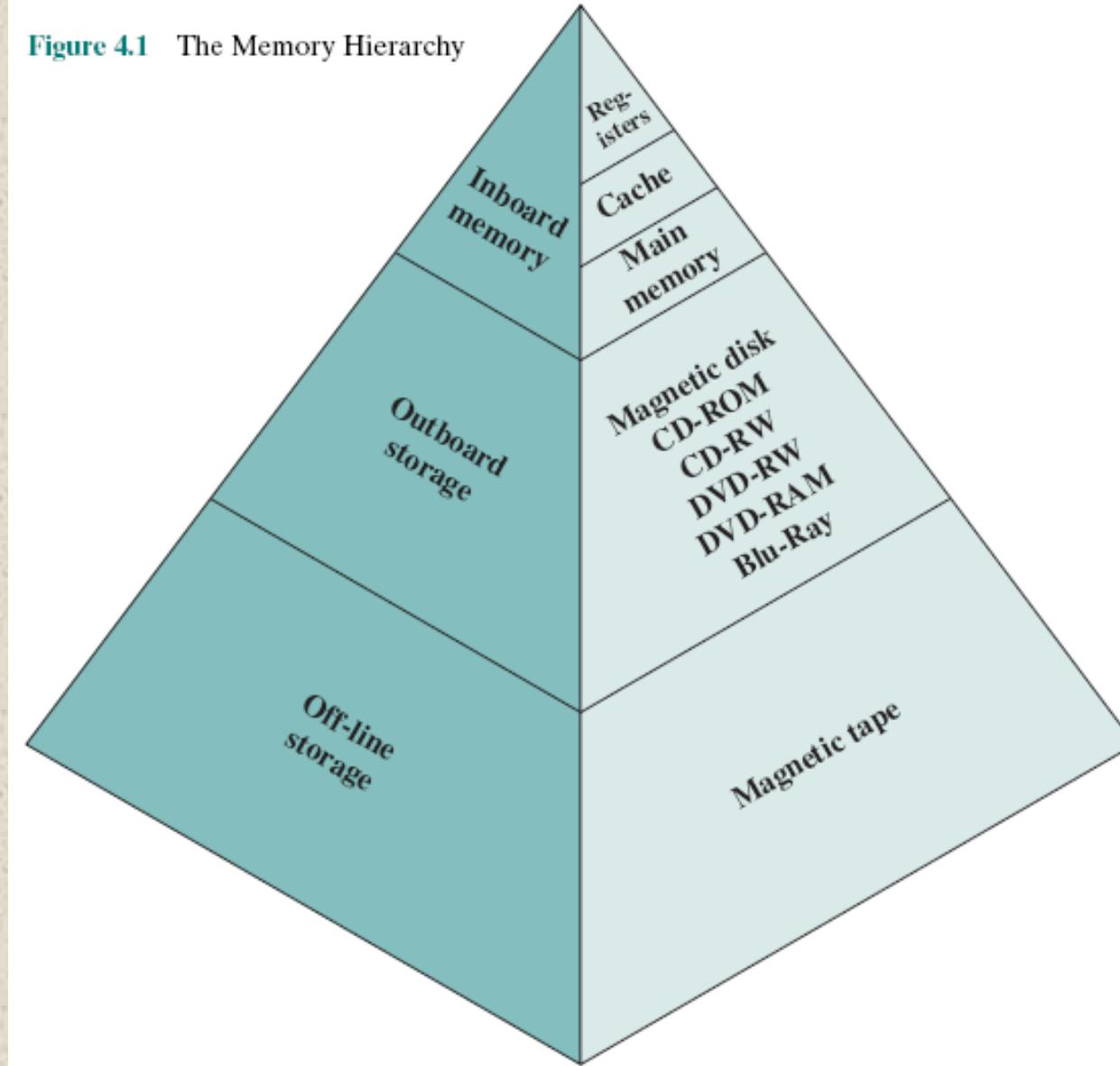
Memory Hierarchy

- Design constraints on a computer's memory can be summed up by three questions:
 - How much, how fast, how **expensive**
- There is a trade-off among capacity, access time, and cost
 - Faster access time, greater cost per bit
 - Greater capacity, smaller cost per bit
 - Greater capacity, slower access time
- The way out of the memory dilemma is not to rely on a single memory component or technology, but to employ a memory hierarchy 

A great capacity memory but cheap and low speed + one or some small capacity memory but fast and more expensive (cache) .

Memory Hierarchy...

Figure 4.1 The Memory Hierarchy



4.2- Cache Memory Principles

- What is cache?
- Cache and Main Memory

What is a Cache?

Cache: A small size, expensive, memory which has high-speed access is located between CPU and RAM (large memory size, cheaper, and lower-speed Memory).

L0: cache in CPU, working with CPU rate. It is not usually implemented.

Memory Management Unit - MMU

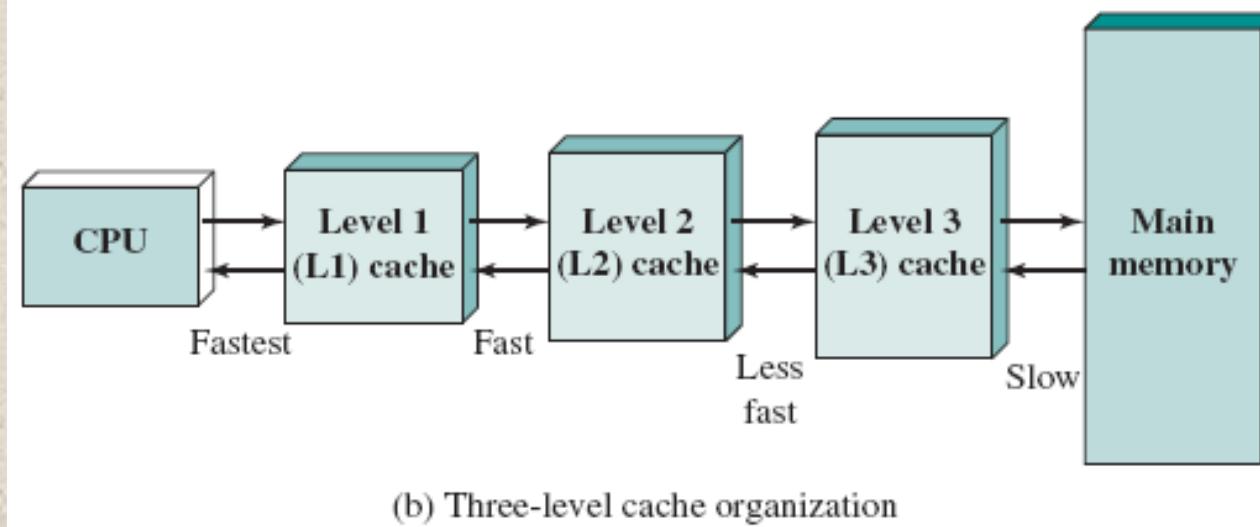
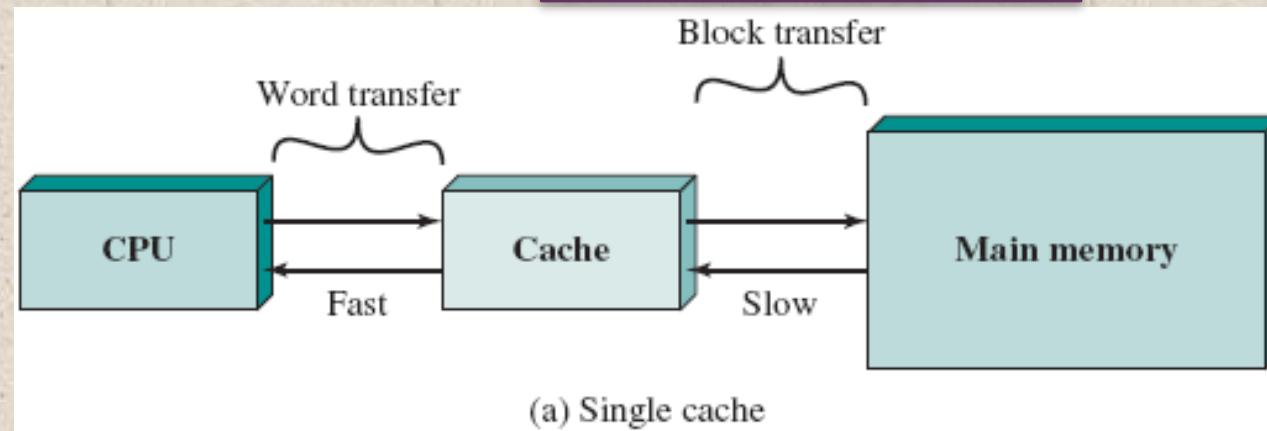
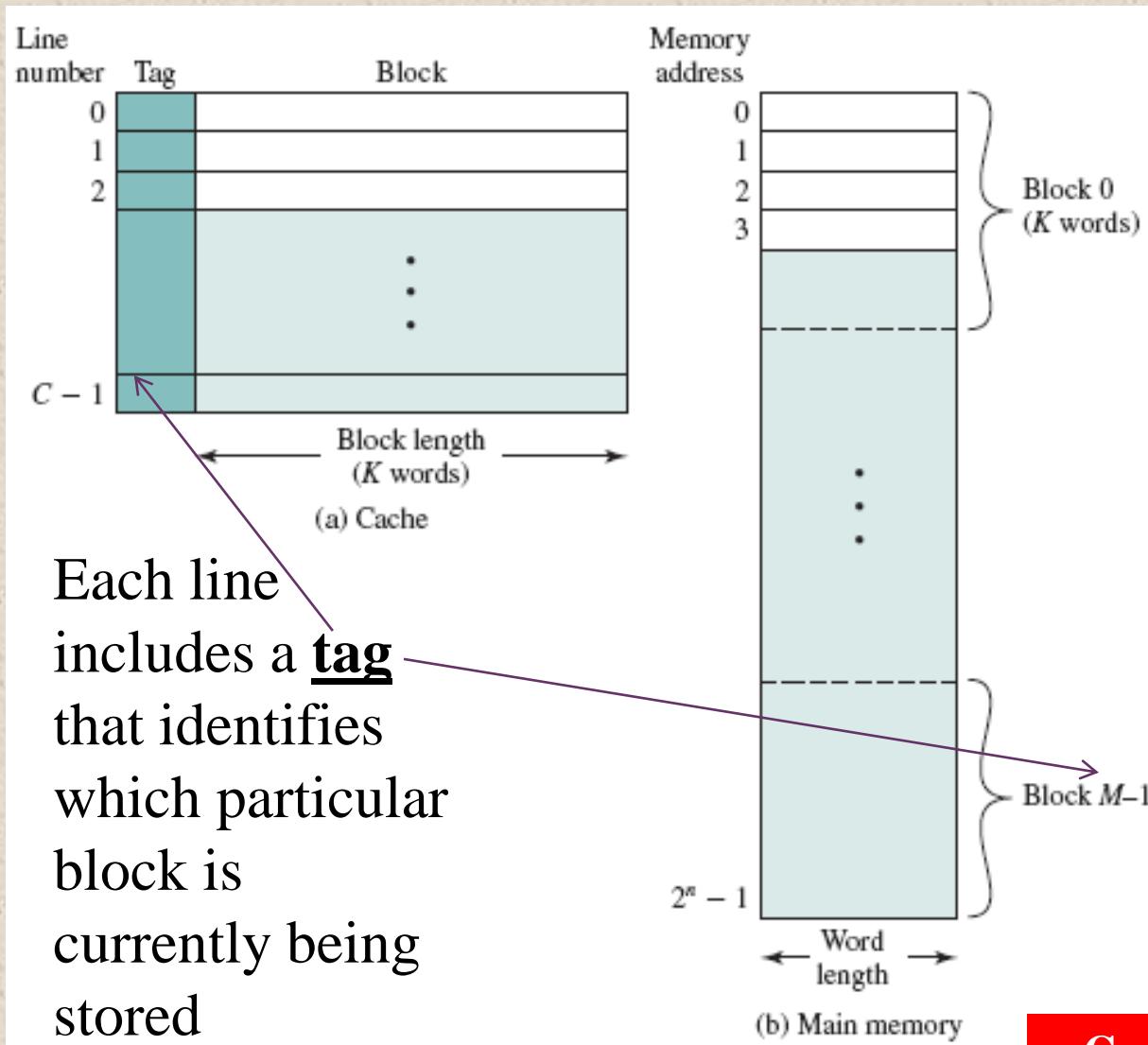


Figure 4.3 Cache and Main Memory

Cache/Main Memory Structure



Each line includes a **tag** that identifies which particular block is currently being stored

Figure 4.4 Cache/Main Memory Structure

Main memory is divided into the same size blocks. Some blocks will be loaded to cache.

Address in cache is different from those in main memory → A mapping is needed.

Cache Addr → Main Mem Addr

4.3- Elements of Cache Design

Overview of cache design parameters

Table 4.2 Elements of Cache Design

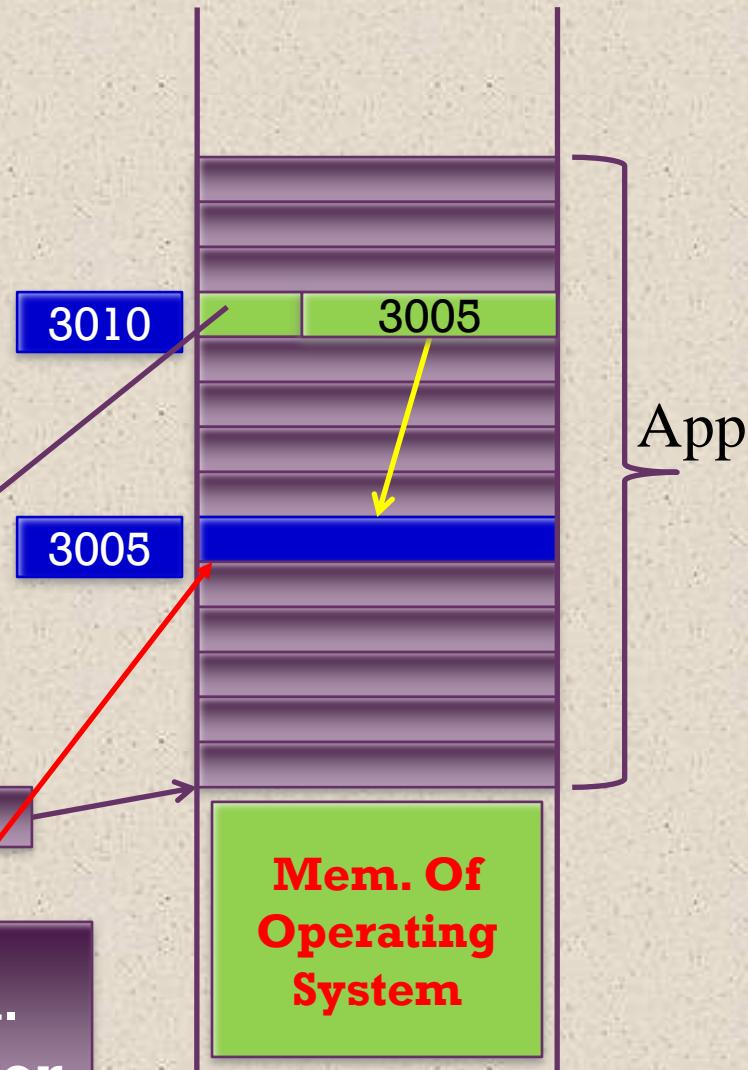
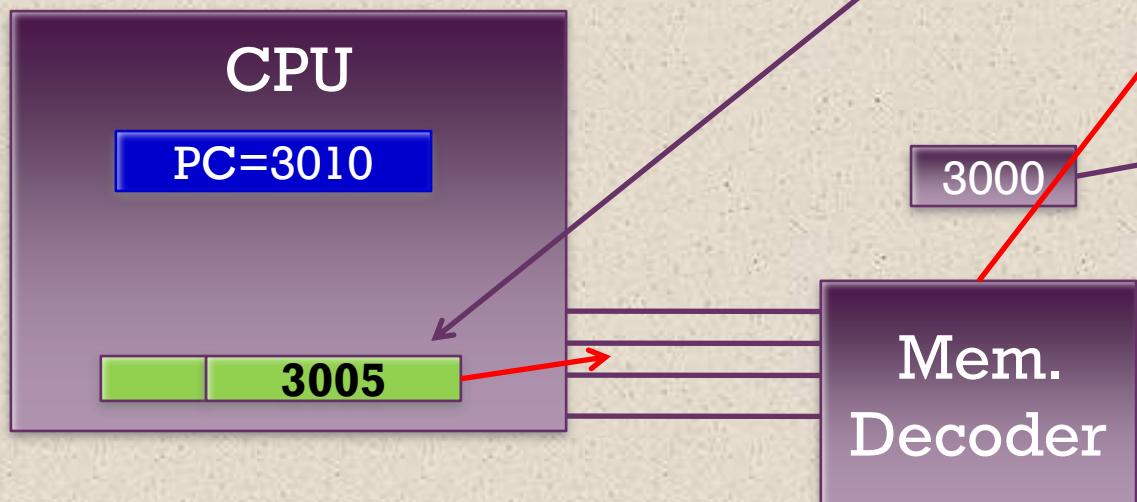
	Cache Addresses	Write Policy
	Logical	Write through
	Physical	Write back
	Cache Size	Line Size
	Mapping Function	Number of Caches
	Direct	Single or two level
	Associative	Unified or split
	Set associative	
	Replacement Algorithm	
	Least recently used (LRU)	
	First in first out (FIFO)	
	Least frequently used (LFU)	
	Random	

Main Memory Address Specifications

Opcode	Memory Add.
--------	-------------

- (1) Physical Address
- (2) Abstract Address
- (3) Virtual Address

Physical Addresses

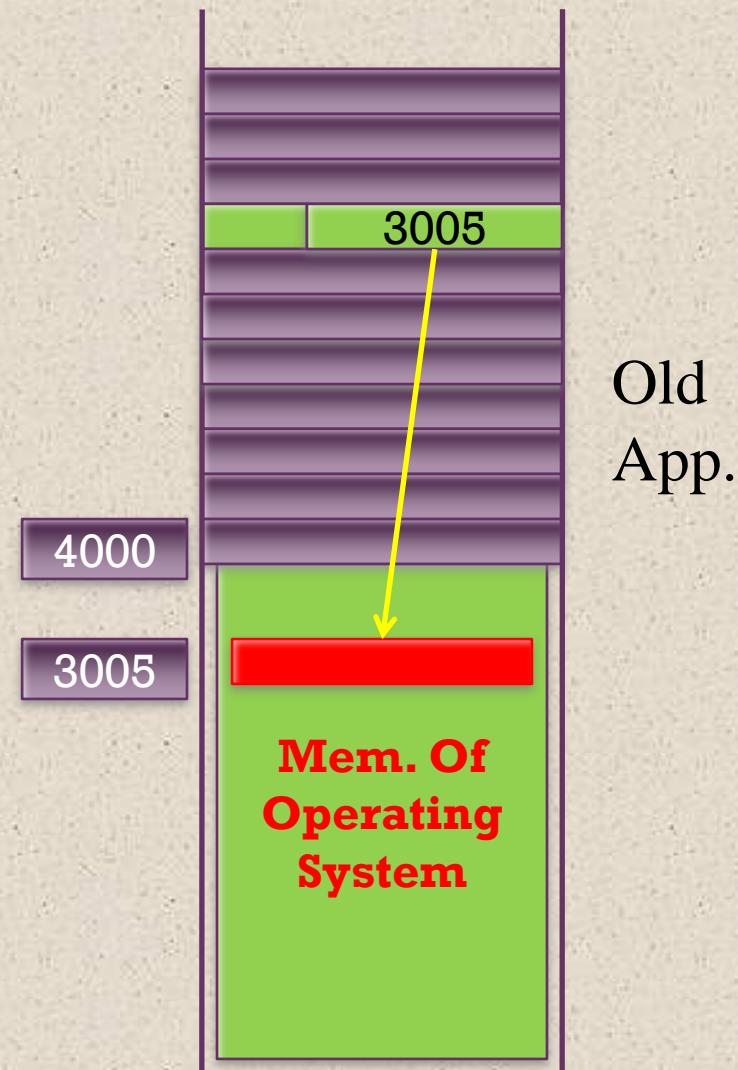


Main Memory Address Specifications

Physical Addresses

When the operating system is upgraded, the OS needs more memory (ex: 4000 bytes), old applications are not compatible.

- Address must be specified by another way to ensure that old programs can be run in new OS.
- Abstract addresses



Old
App.

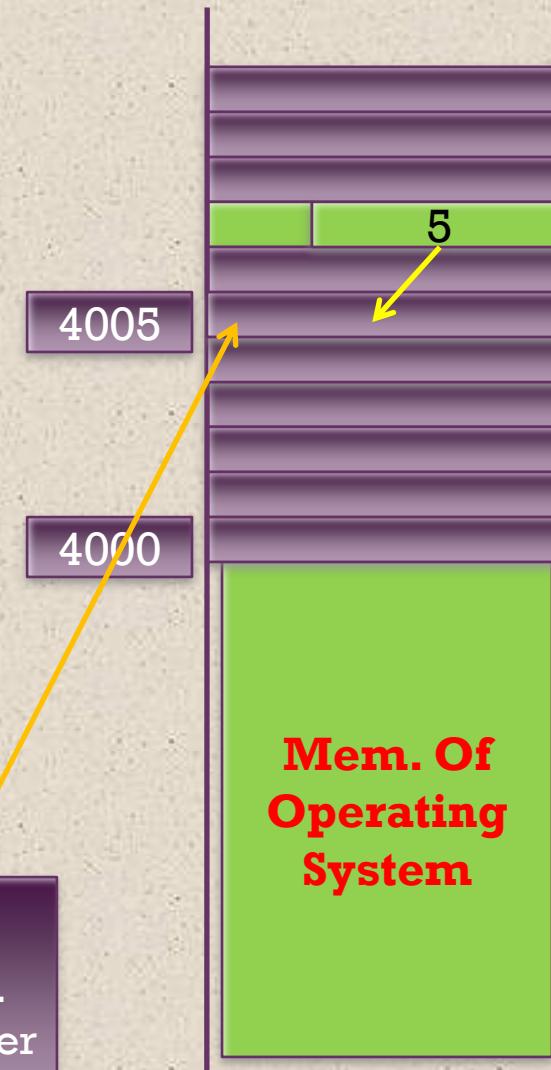
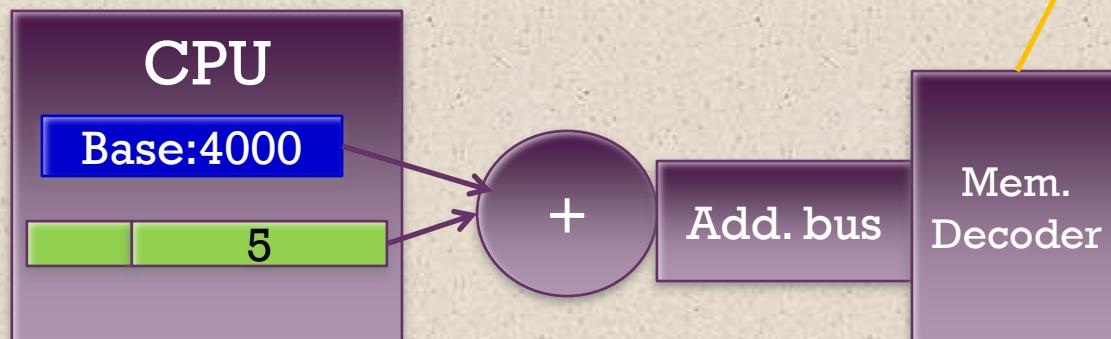
Main Memory Address Specifications

Abstract Addresses

All addresses in an application will be specified by compilers using an offset (difference) from the base address (position at which the app. is loaded)

-A register (base register) must be added to maintain the base address of the process

→ An address is specified by <base, offset>



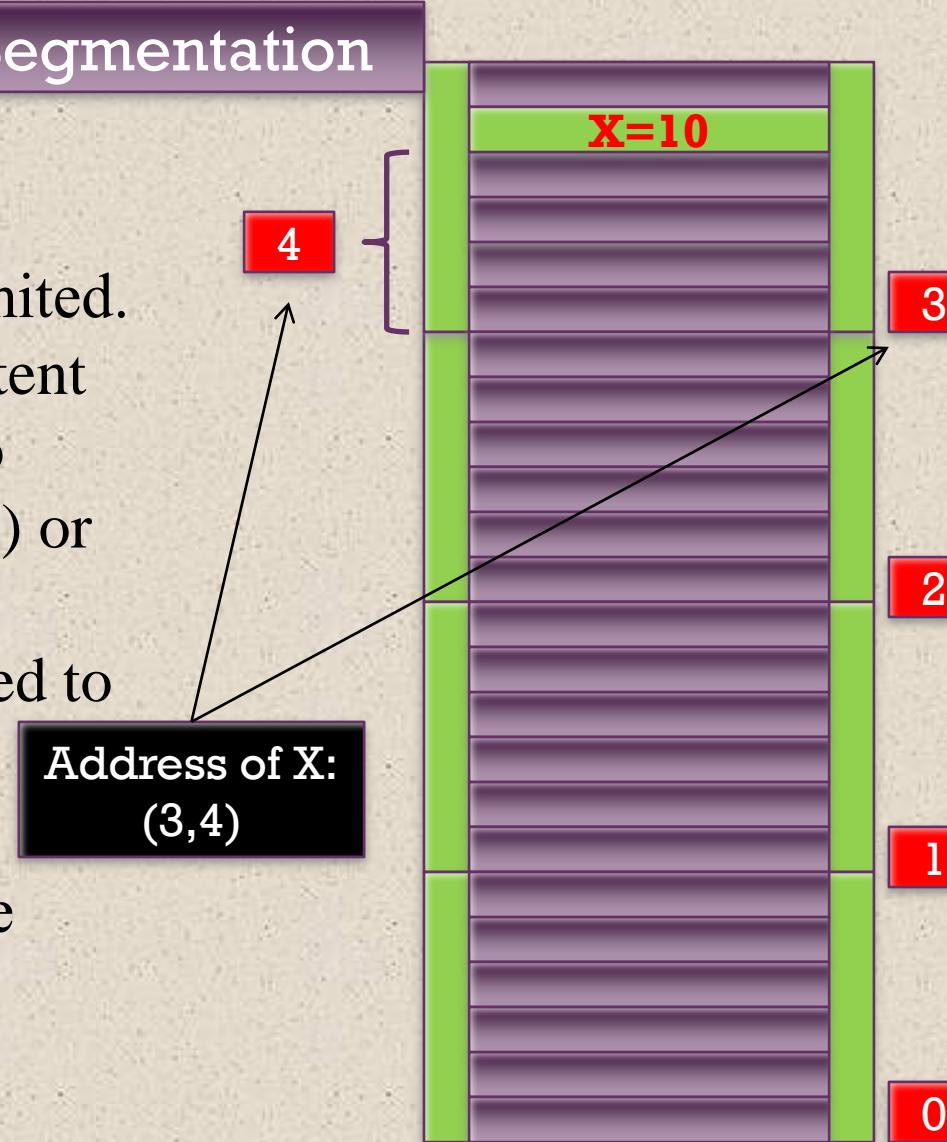
pp.

Main Memory Address Specifications

Virtual Addresses: Paging- Segmentation

Contemporary OSs allows many programs running concurrently although system's memory is limited. Solution is that the program content and memory will be divided into some pages (same-size, ex: 4KB) or segments (different size). Only needed pages/segments are loaded to system memory.

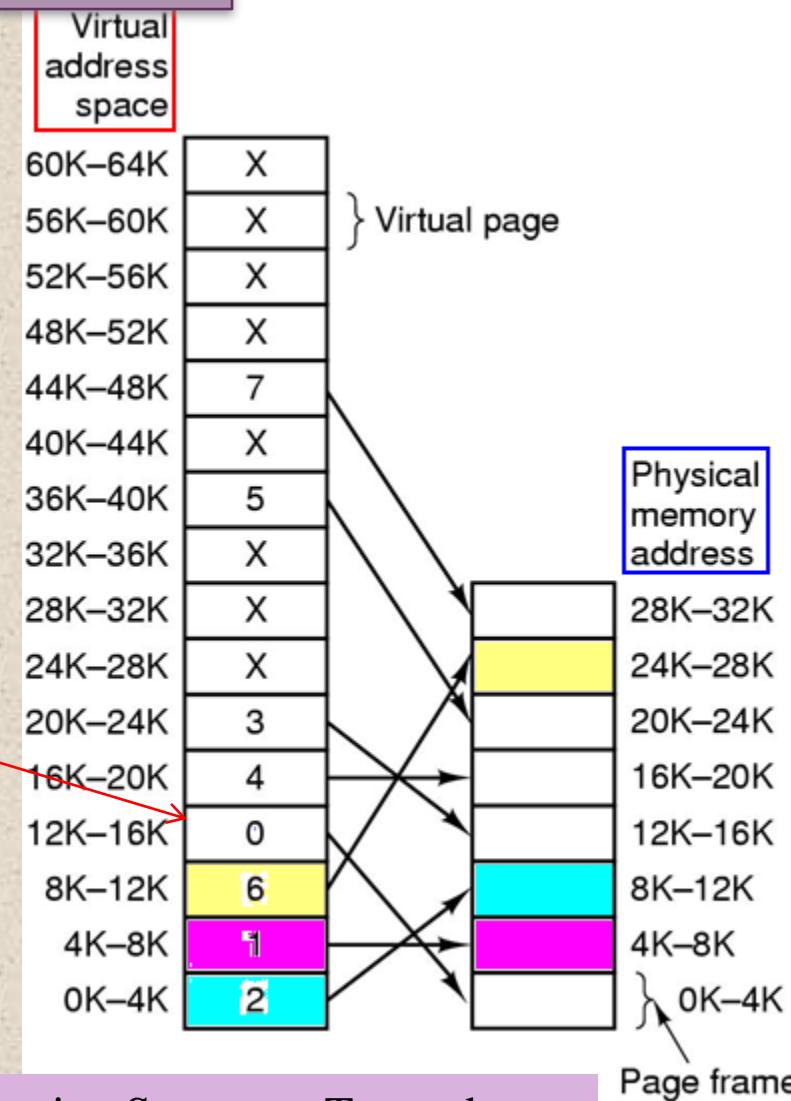
→ Compilers must translate program's addresses to a suitable form (virtual address). Virtual address format:
<page/segment, offset>



Main Memory Address Specifications

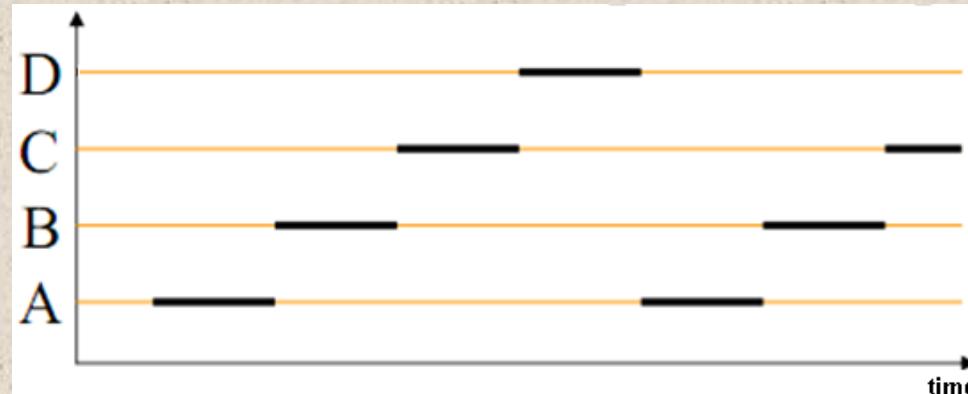
Virtual Addresses: Paging- Segmentation

- When an instruction/data is accessed, physical address must be supplied. A **mapping** is needed as a mean for determining physical addresses from their virtual addresses. This mapping is implemented in OS as a **page table**.
- A hardware is needed to translate virtual address to physical address → **MMU** – Memory Management Unit.



Main Memory Address Specifications

Virtual Addresses: Paging- Segmentation



Time-sharing
mechanism

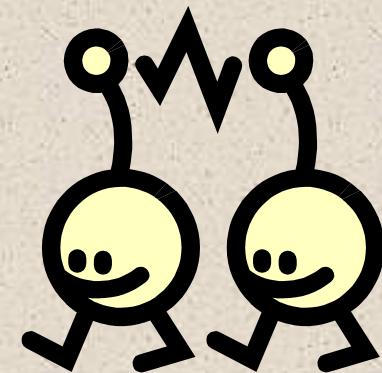
- How do OSs permit many program running concurrently? → Only some pages/segments of a process are loaded and the time-sharing mechanism is applied.
- Advantages of memory paging: Many apps can run concurrently in limited memory. A page of a program can be loaded into arbitrary physical memory location.
- Disadvantages of memory paging: If a page fault occurs, cost must be paid when an in-memory frame must be swapped to disk from memory (swap out) then a page from disk will be loaded to memory (swap in) – Paging replacement.

+

Cache Addresses: Virtual Address

■ **Virtual memory**

- Facility that allows programs to address memory from a logical point of view, without regard to the amount of main memory physically available. Only some needed small parts of a program are loaded to main memory at a time. So, a large program can run although memory size is smaller
- When used, the address fields of machine instructions contain virtual addresses
- For reads from and writes to main memory, a hardware memory management unit (MMU) translates each virtual address into a physical address in main memory



Logical and Physical Caches

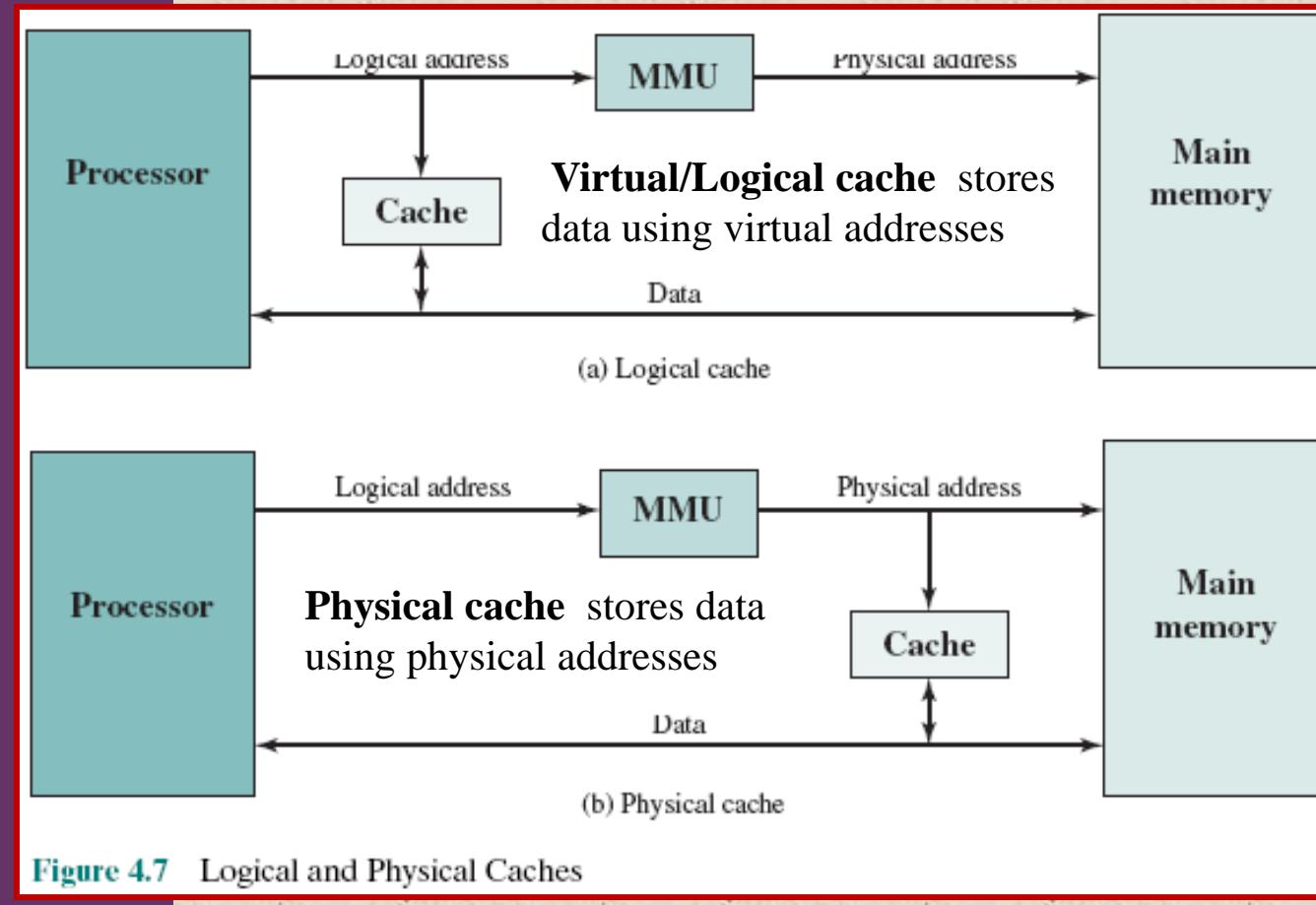


Figure 4.7 Logical and Physical Caches

Mapping Function

- Because there are fewer cache lines than main memory blocks, an algorithm is needed for mapping main memory blocks into cache lines
- Three techniques can be used:

Direct

- The simplest technique
- Maps each block of main memory into only one possible cache line

Associative

- Permits each main memory block to be loaded into any line of the cache
- The cache control logic interprets a memory address simply as a Tag and a Word field
- To determine whether a block is in the cache, the cache control logic must simultaneously examine every line's Tag for a match

Set Associative

- A compromise that exhibits the strengths of both the direct and associative approaches while reducing their disadvantages
- **Read by yourself**



Direct Mapping

The block j in main memory will be loaded to the line i of the cache: $i = j \bmod m$

A block in main memory can be load to any line of the cache

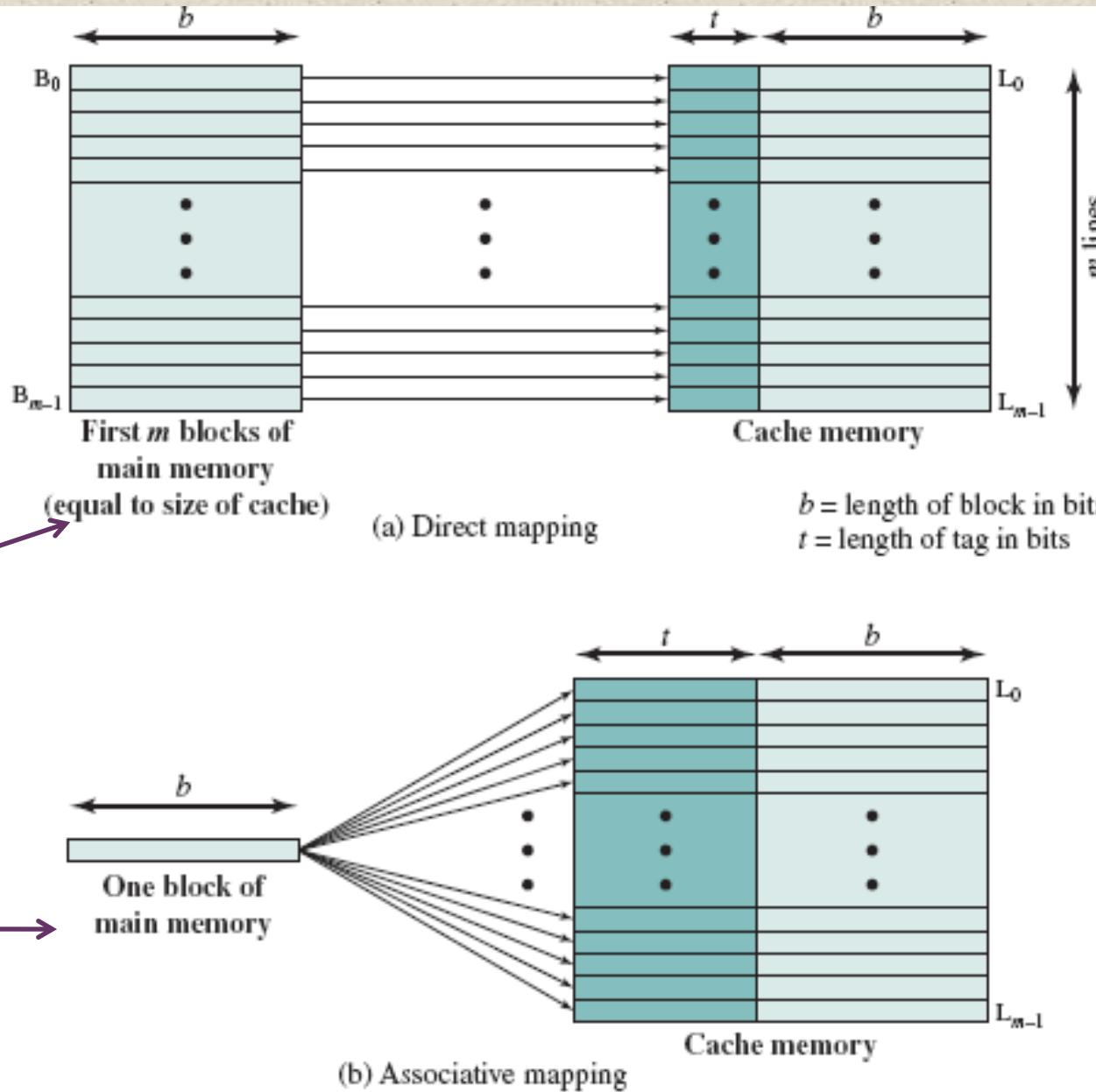


Figure 4.8 Mapping from Main Memory to Cache: Direct and Associative

Direct Mapping Cache Organization

READ BY YOURSELF

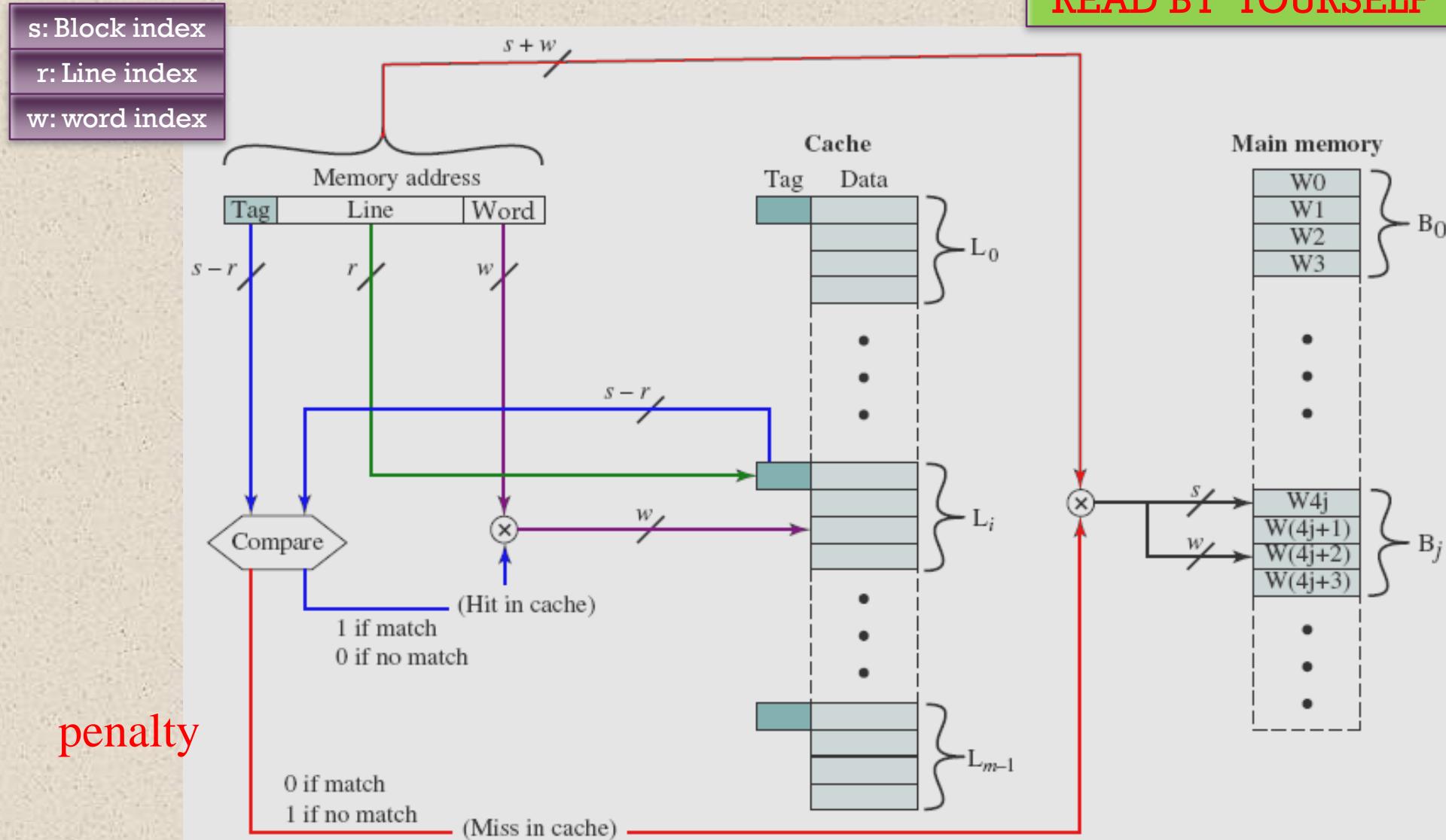


Figure 4.9 Direct-Mapping Cache Organization

+

Direct Mapping Example

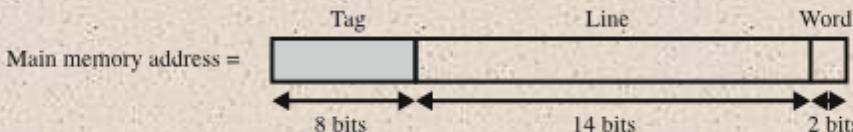
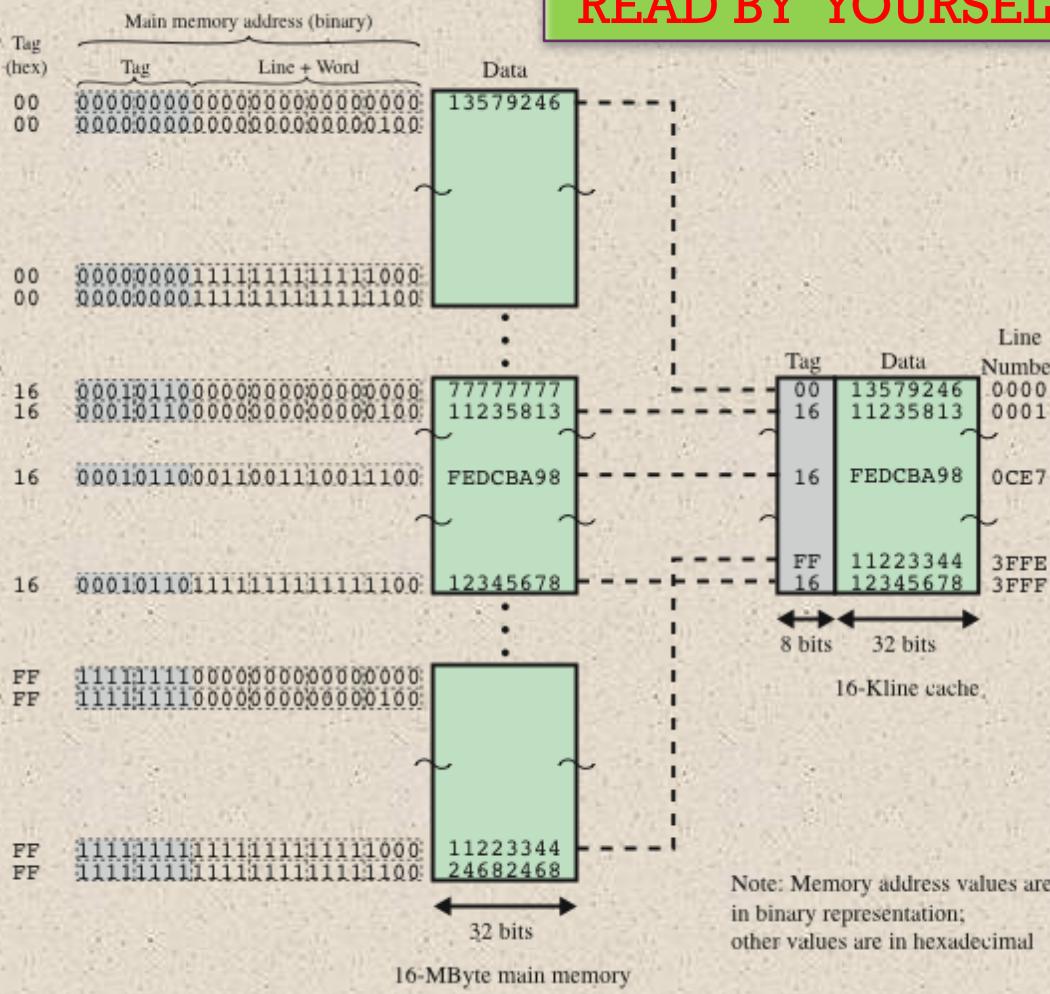
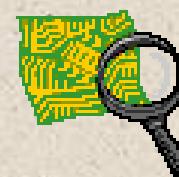


Figure 4.10 Direct Mapping Example

+

Direct Mapping Summary

- Address length = $(s + w)$ bits
- Number of addressable units = 2^{s+w} words or bytes
- Block size = line size = 2^w words or bytes
- Number of blocks in main memory = $2^{s+w}/2^w = 2^s$
- Number of lines in cache = $m = 2^r$
- Size of tag = $(s - r)$ bits





Victim Cache



- Originally proposed as an approach to reduce the conflict misses of direct mapped caches without affecting its fast access time
- Fully associative cache
- Typical size is 4 to 16 cache lines
- Residing between direct mapped L1 cache and the next level of memory

Fully Associative Cache Organization

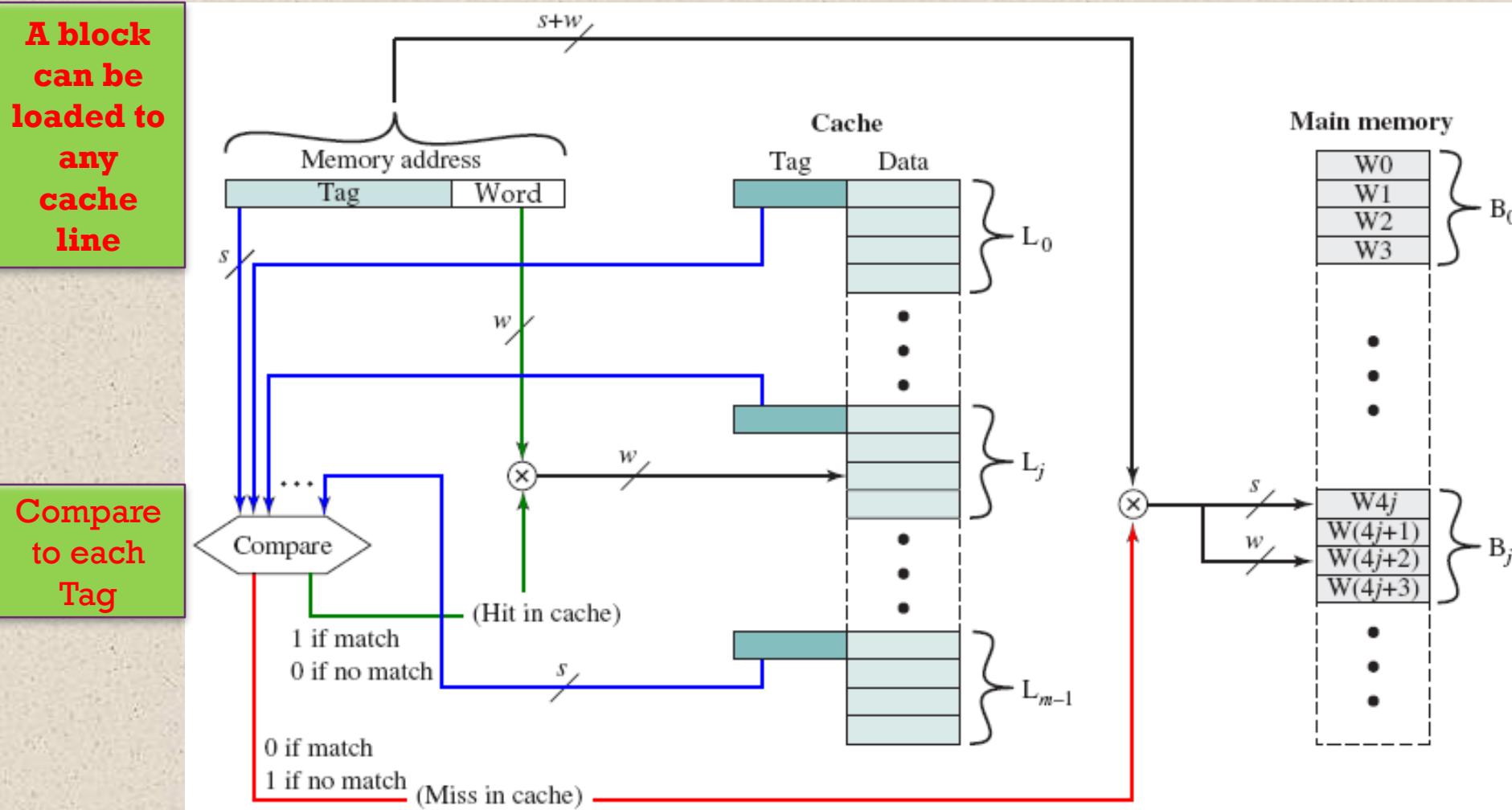


Figure 4.11 Fully Associative Cache Organization

Associative Mapping Example

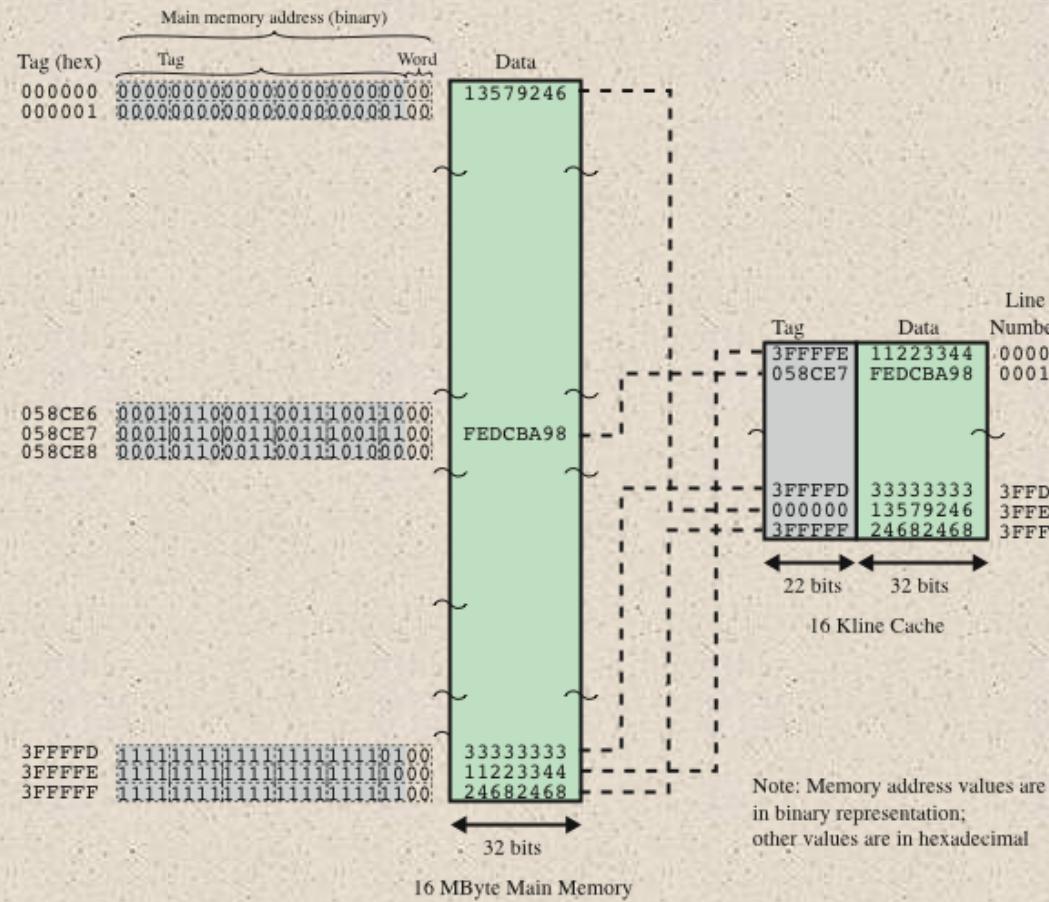
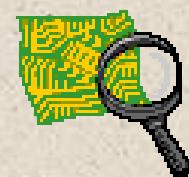


Figure 4.12 Associative Mapping Example

+

Associative Mapping Summary

- Address length = $(s + w)$ bits
- Number of addressable units = 2^{s+w} words or bytes
- Block size = line size = 2^w words or bytes
- Number of blocks in main memory = $2^{s+w}/2^w = 2^s$
- Number of lines in cache = undetermined
- Size of tag = s bits



+

Set Associative Mapping

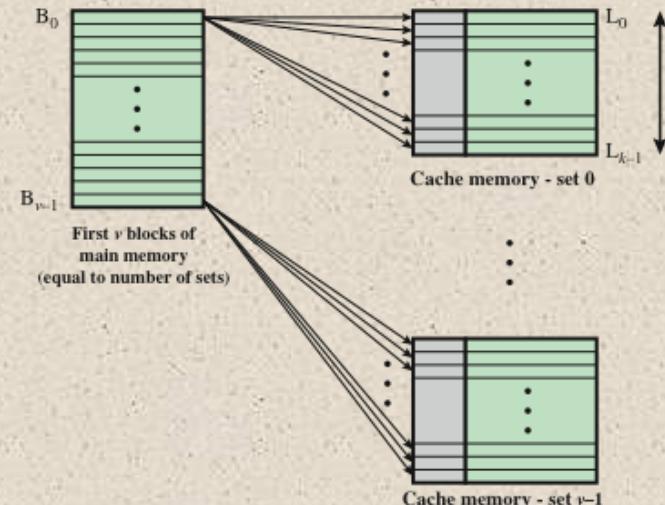
- Compromise (thỏa hiệp) that exhibits the strengths of both the direct and associative approaches while reducing their disadvantages
- Cache consists of a number of sets
- Each set contains a number of lines
- A given block maps to any line in a given set
- e.g. 2 lines per set
 - 2 way associative mapping
 - A given block can be in one of 2 lines in only one set

Go to Replacement
Algorithms



Mapping From Main Memory to Cache: k -Way Set Associative

READ BY YOURSELF



(a) v associative-mapped caches

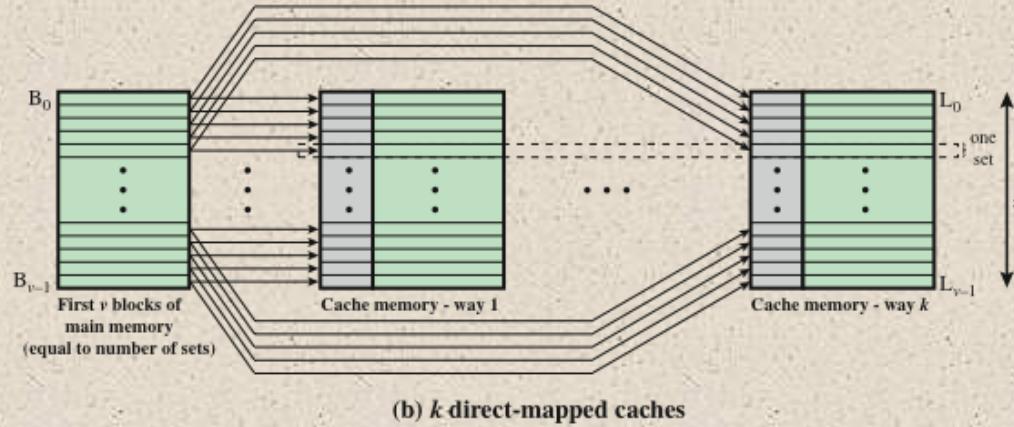


Figure 4.13 Mapping From Main Memory to Cache:
 k -way Set Associative

k-Way Set Associative Cache Organization

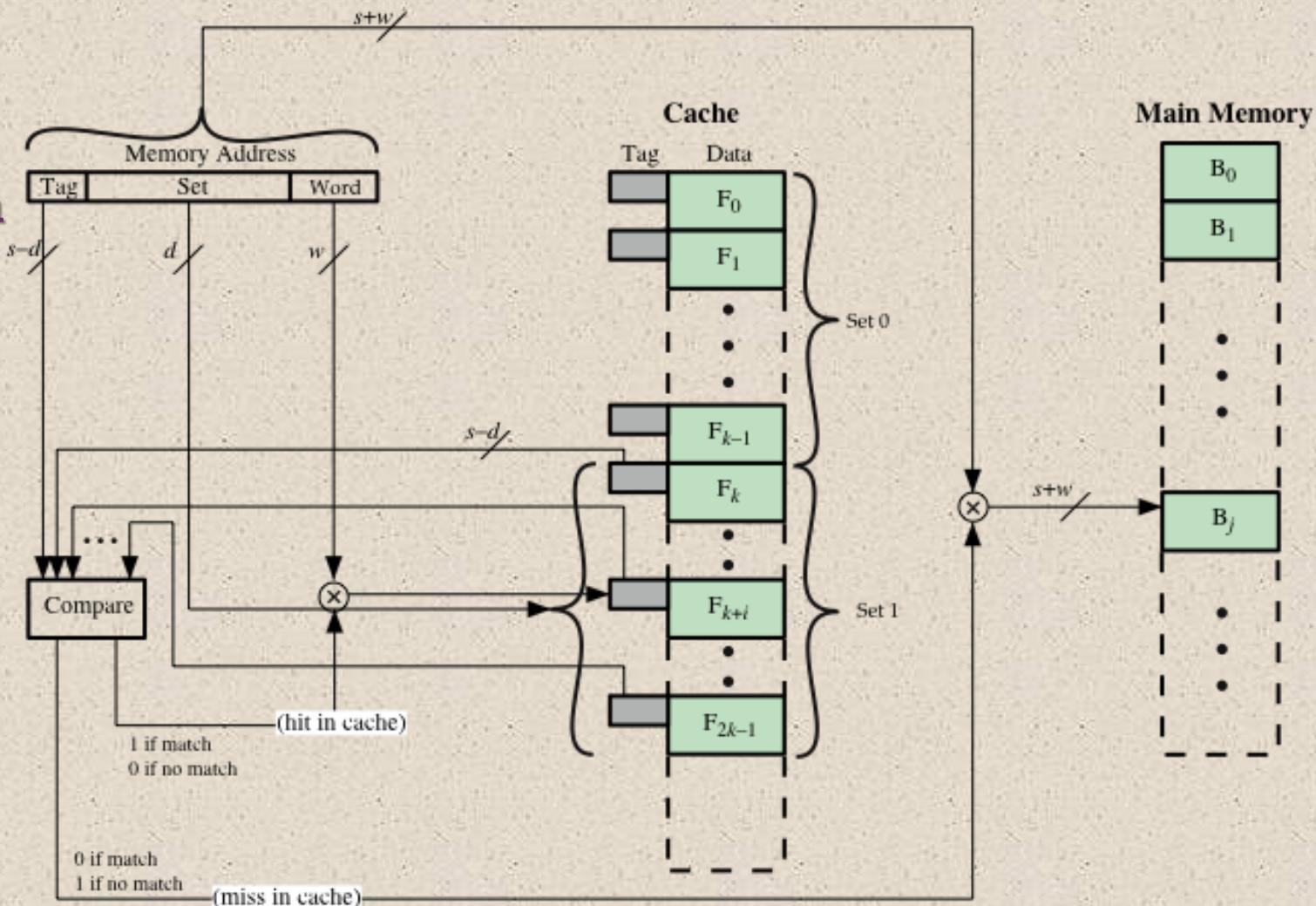


Figure 4.14 *k*-Way Set Associative Cache Organization

+

Set Associative Mapping Summary

- Address length = $(s + w)$ bits
- Number of addressable units = 2^{s+w} words or bytes
- Block size = line size = 2^w words or bytes
- Number of blocks in main memory = $2^{s+w}/2^w=2^s$
- Number of lines in set = k
- Number of sets = v = 2^d
- Number of lines in cache = m=kv = $k * 2^d$
- Size of cache = $k * 2^{d+w}$ words or bytes
- Size of tag = $(s - d)$ bits



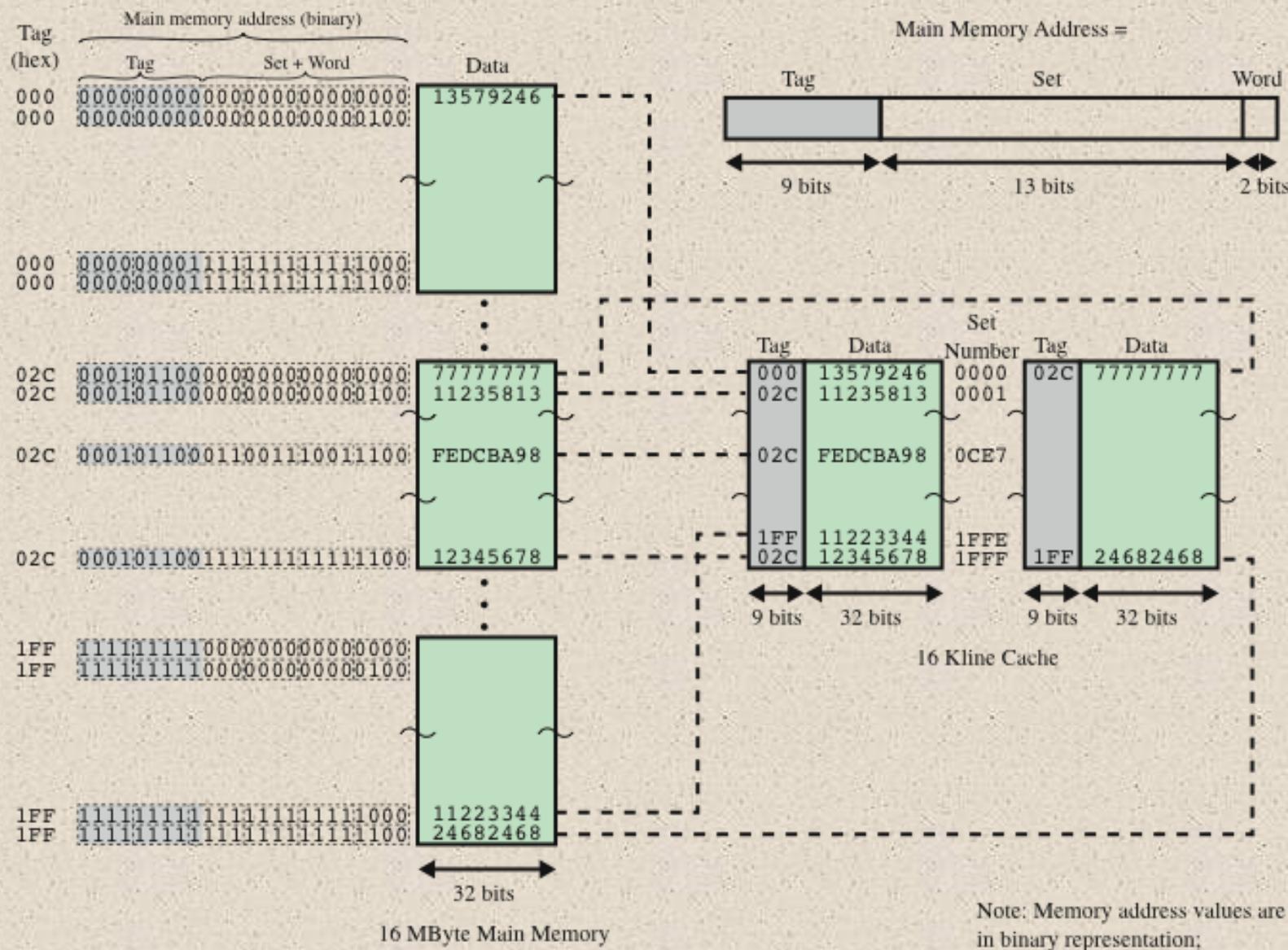


Figure 4.15 Two-Way Set Associative Mapping Example

+

Varying Associativity Over Cache Size

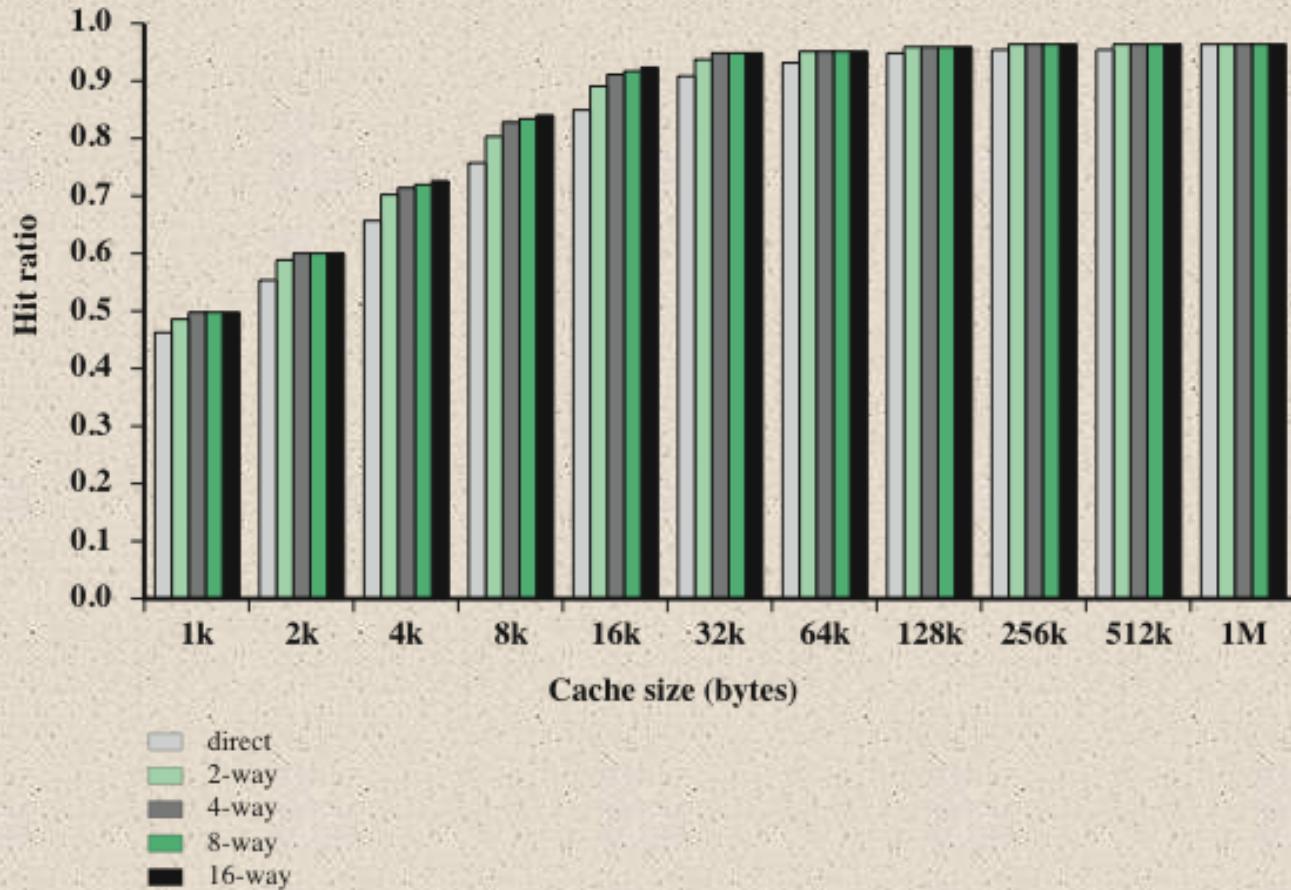
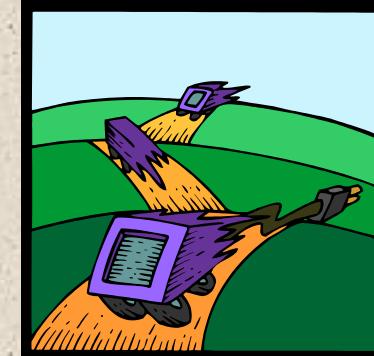


Figure 4.16 Varying Associativity over Cache Size

Replacement Algorithms



- Two situations:
 - Cache hit: Accessed address exists in cache
 - Cache miss: Accessed address does not exist in cache. The memory block containing it must be loaded to the cache
- Once the cache has been filled, when a new block is brought into the cache, one of the existing blocks must be replaced
 - For **direct mapping** there is only one possible line for any particular block and **no choice is possible**
 - For the **associative and set-associative techniques** a **replacement algorithm** is needed
 - To achieve high speed, an algorithm must be implemented in hardware

The four most common replacement algorithms are:

- **Least recently used (LRU)**
 - Most effective
 - Replace that block in the set that has been in the cache longest with no reference to it
 - Because of its simplicity of implementation, LRU is the most popular replacement algorithm

- **First-in-first-out (FIFO)**
 - Replace that block in the set that has been in the cache longest
 - Easily implemented as a round-robin or circular buffer technique

- **Least frequently used (LFU)**
 - Replace that block in the set that has experienced the fewest references
 - Could be implemented by associating a counter with each line

Write Policy

When a block that is resident in the cache is to be replaced there are two cases to consider:

If the old block in the cache has not been altered then it may be overwritten with a new block without first writing out the old block

If at least one write operation has been performed on a word in that line of the cache then main memory must be updated by writing the line of cache out to the block of memory before bringing in the new block

There are two problems to contend (đấu tranh) with:

More than one device may have access to main memory

A more complex problem occurs when multiple processors are attached to the same bus and each processor has its own local cache - if a word is altered in one cache it could conceivably invalidate a word in other caches

+

Write Through and Write Back

■ Write through- Ghi thẳng

- Simplest technique
- All write operations are made to main memory as well as to the cache
- The main disadvantage of this technique is that it generates substantial (heavy) memory traffic and may create a bottleneck

■ Write back-Ghi ngầm

- Minimizes memory writes
- **Updates are made only in the cache**
- **Portions of main memory are invalid and hence accesses by I/O modules can be allowed only through the cache**
- This makes for complex circuitry and a potential bottleneck

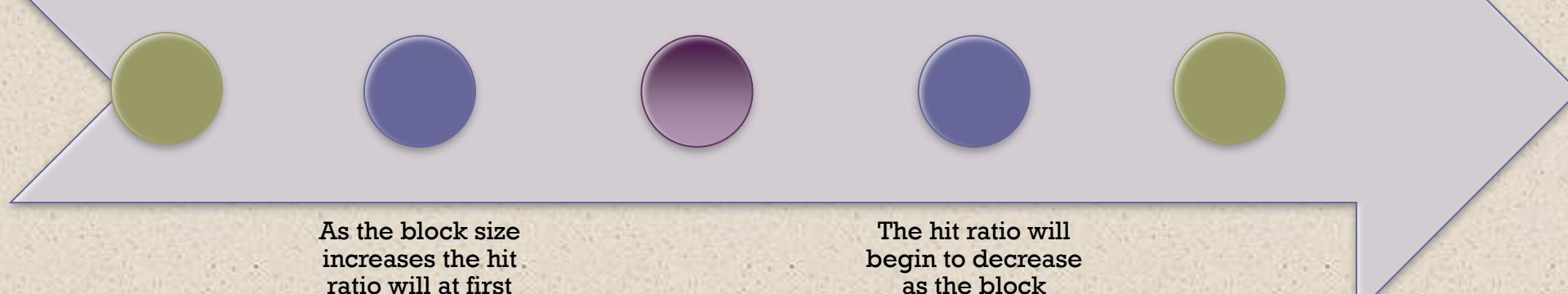
Line Size

When a block of data is retrieved and placed in the cache not only the desired word but also some number of adjacent words are retrieved

As the block size increases more useful data are brought into the cache

Two specific effects come into play:

- Larger blocks reduce the number of blocks that fit into a cache
- As a block becomes larger each additional word is farther from the requested word



As the block size increases the hit ratio will at first increase because of the principle of locality

The hit ratio will begin to decrease as the block becomes bigger and the probability of using the newly fetched information becomes less than the probability of reusing the information that has to be replaced

Larger line size → More data → Cache hit increases, but expensive and more data in cache but not used (Normal: 64-128 bytes)

Multilevel Caches

- As logic density has increased it has become possible to have a cache on the same chip as the processor
- The on-chip cache reduces the processor's external bus activity and speeds up execution time and increases overall system performance
 - When the requested instruction or data is found in the on-chip cache, the bus access is eliminated
 - On-chip cache accesses will complete appreciably faster than would even zero-wait state bus cycles
 - During this period the bus is free to support other transfers
- Two-level cache:
 - Internal cache designated as level 1 (L1)
 - External cache designated as level 2 (L2)
- Potential savings due to the use of an L2 cache depends on the hit rates in both the L1 and L2 caches
- The use of multilevel caches complicates all of the design issues related to caches, including size, replacement algorithm, and write policy

Hit Ratio (L1 & L2) For 8 Kbyte and 16Kbyte L1

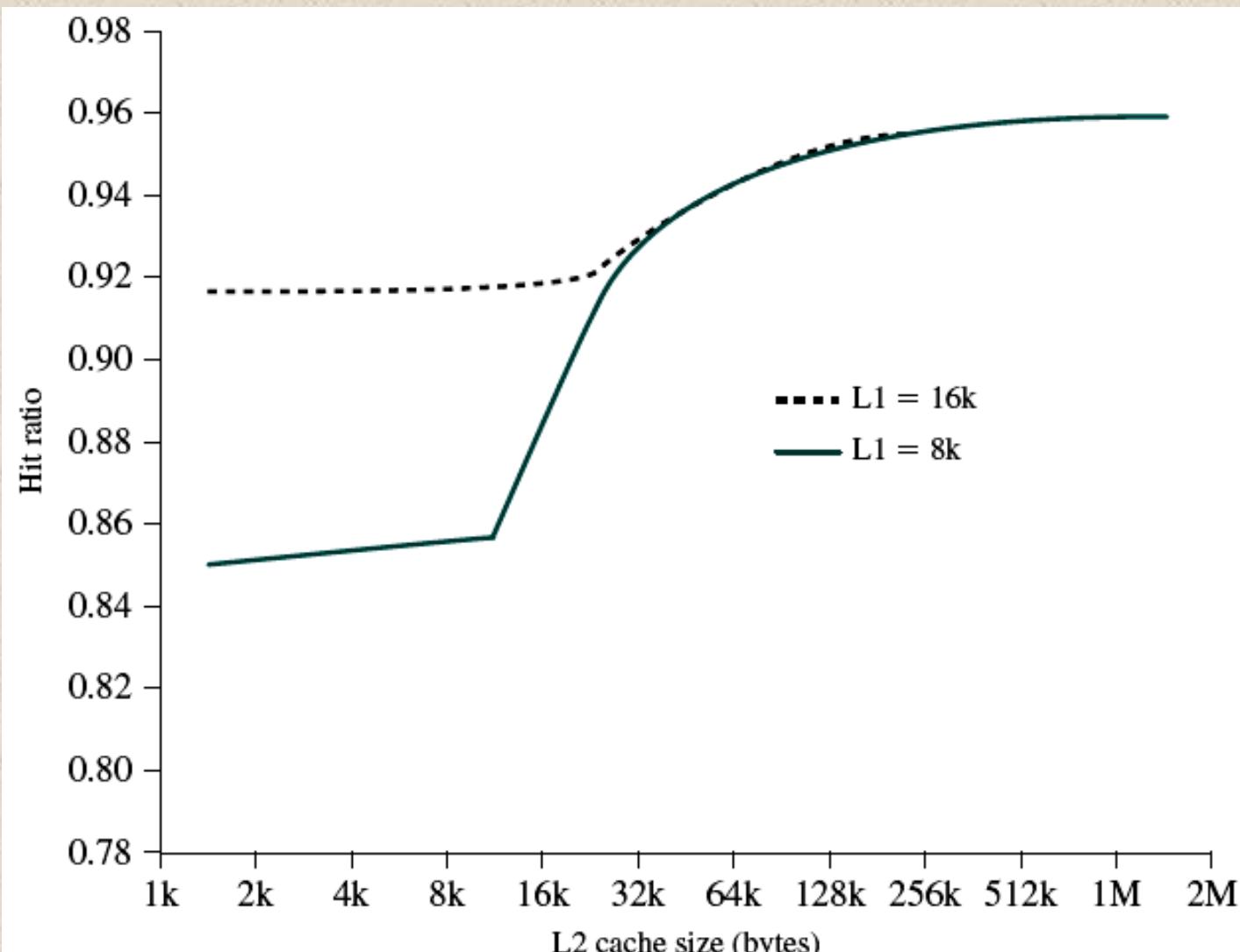


Figure 4.17 Total Hit Ratio (L1 and L2) for 8-Kbyte and 16-Kbyte L1

+

Unified Versus Split Caches

- Has become common to split cache:
 - One dedicated to instructions
 - One dedicated to data
 - Both exist at the same level, typically as two L1 caches
- Advantages of unified cache: Higher hit rate
 - Balances load of instruction and data fetches automatically
 - Only one cache needs to be designed and implemented
- Trend is toward split caches at the L1 and unified caches for higher levels
- Advantages of split cache:
 - Eliminates cache contention (tranh chấp) between instruction fetch/decode unit and execution unit
 - Important in pipelining (cơ chế đường ống, output của xử lý này là input của xử lý kế tiếp)



Exercises

- **4.1- What are the differences among sequential access, direct access, and random access?**
- **4.2-What is the general relationship among access time, memory cost, and capacity?**
- **4.3- How does the principle of locality relate to the use of multiple memory levels?**
- **4.4- What are the differences among direct mapping and associative mapping,?**
- **4.5- For a direct-mapped cache, a main memory address is viewed as consisting of three fields. List and define the three fields.**
- **4.6- For an associative cache, a main memory address is viewed as consisting of two fields. List and define the two fields.**

+

Summary

Chapter 4

Cache Memory

- Characteristics of Memory Systems
 - Location
 - Capacity
 - Unit of transfer
- Memory Hierarchy
 - How much?
 - How fast?
 - How expensive?
- Cache memory principles
- Elements of cache design
 - Cache addresses
 - Cache size
 - Mapping function
 - Replacement algorithms
 - Write policy
 - Line size
 - Number of caches