Chapter 8

Operating System Support

Objectives

- How can a computer be made more convenient to use?
- How are computer system resources used in an efficient manner?
- After studying this chapter, you should be able to:
  - Summarize, at a top level, the key functions of an operating system (OS).
  - Discuss the evolution of operating systems for early simple batch systems to modern complex systems.
  - Explain the differences among long-, medium-, and short-term scheduling.
  - Understand the reason for memory partitioning and explain the various techniques that are used.
  - Assess the relative advantages of paging and segmentation.
  - Define virtual memory.

Contents

- 8.1 Operating System Overview
- 8.2 Scheduling
- 8.3 Memory Management

8.1- Operating System Overview

- The Operating System as a User/Computer Interface
- The Operating System as Resource Manager
- Types of Operating Systems

The Operating System as User/Computer Interface

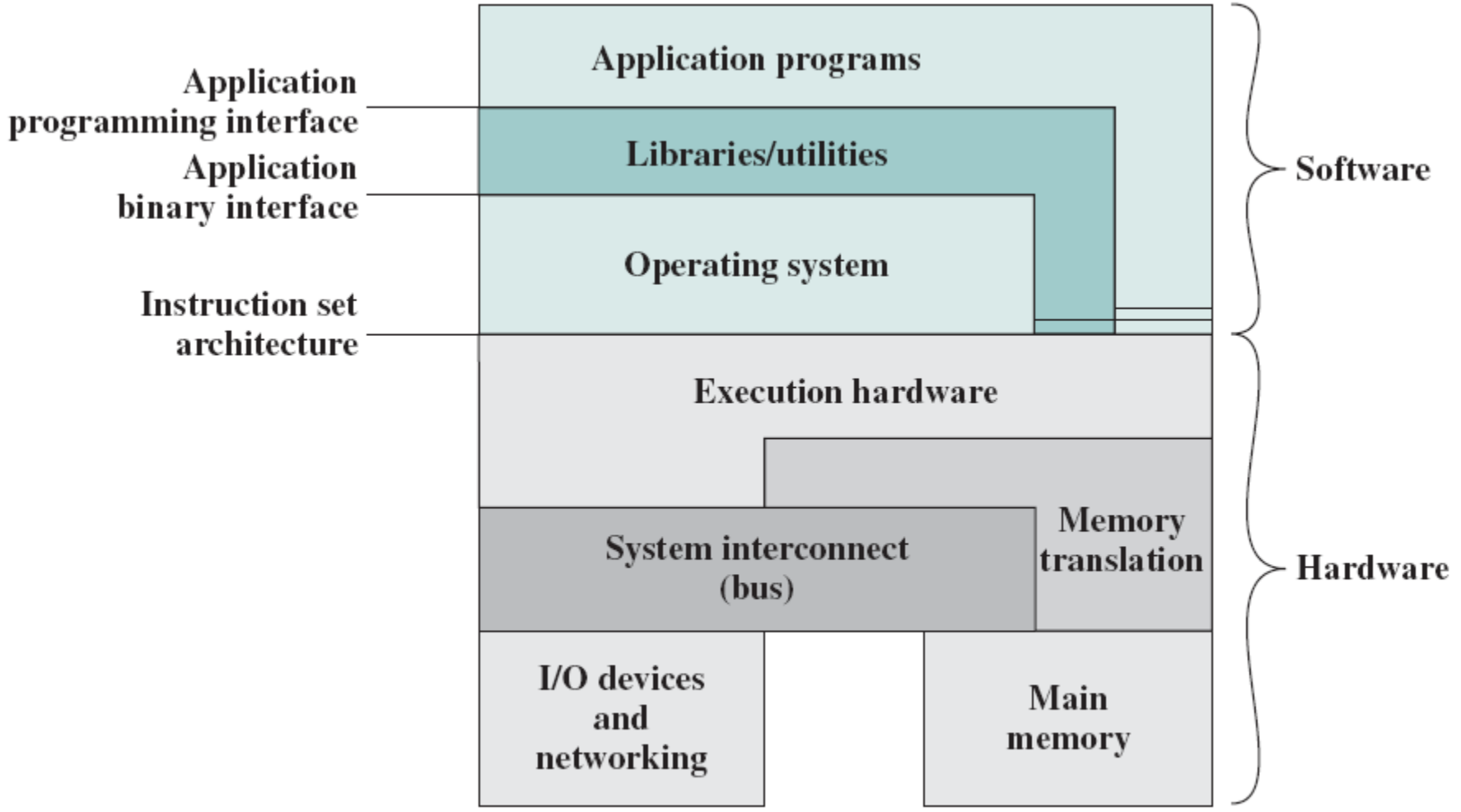


Figure 8.1 Computer Hardware and Software Structure

Operating System (OS) Services

- The most important system program
- Masks the details of the hardware from the programmer and provides the programmer with a convenient interface for using the system
- The OS typically provides services in the following areas:
  - Program creation
  - Program execution
  - Access to I/O devices
  - Controlled access to files
  - System access
  - Error detection and response
  - Accounting

Interfaces

Key interfaces in a typical computer system:

Instruction Set Architecture (ISA)

- Defines the **machine language instructions** that a computer can follow
- **Boundary** between hardware and software

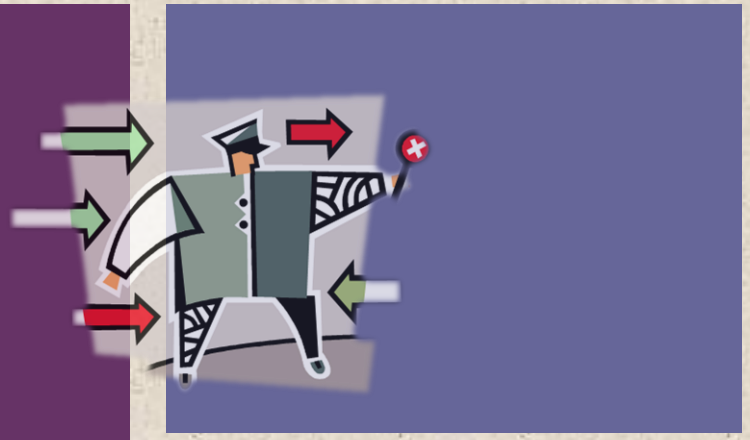
Application Binary Interface (ABI)

- Defines **a standard for binary portability** across programs
- Defines the **system call interface to the operating system and the hardware resources** and services available in a system through the user ISA

Application Programming Interface (API)

- Gives a **program access to the hardware resources and services** available in a system through the user ISA supplemented with **high-level language (HLL) library calls**
- Using **an API enables application software to be ported easily** to other systems that support the same API

Operating System as Resource Manager



A computer is a set of resources for the movement, storage, and processing of data and for the control of these functions

- The OS is responsible for managing these resources

The OS as a control mechanism is unusual in two respects:

- The OS functions in the same way as ordinary computer software – it is a program executed by the processor
- The OS frequently relinquishes (buông thả) control and must depend on the processor to allow it to regain control

The OS as Resource Manager

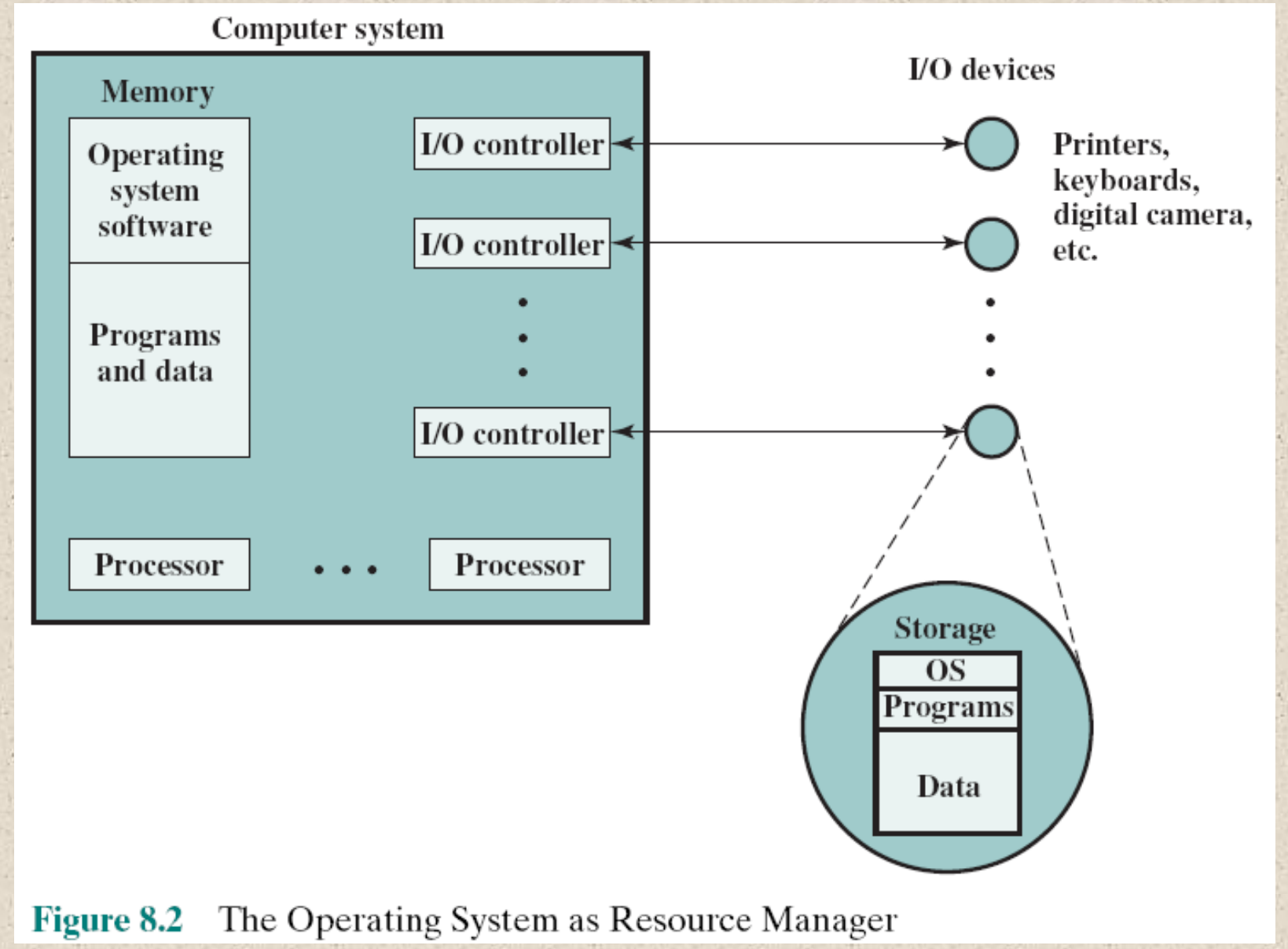
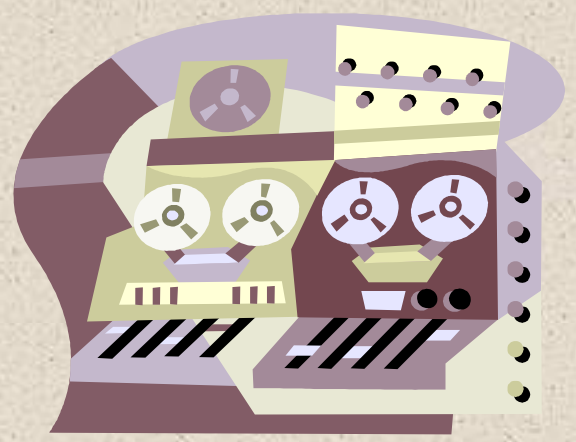


Figure 8.2 The Operating System as Resource Manager

Types of Operating Systems

- **Interactive system**
  - The user/programmer interacts directly with the computer to request the execution of a job or to perform a transaction
  - User may, depending on the nature of the application, communicate with the computer during the execution of the job
- **Batch system**
  - Opposite of interactive
  - The user's program is batched together with programs from other users and submitted by a computer operator
  - After the program is completed results are printed out for the user

Early Systems



- From the late 1940s to the mid-1950s the programmer interacted directly with the computer hardware – there was no OS
  - Processors were run from a console consisting of display lights, toggle switches, some form of input device and a printer
- **Problems:**
  - **Scheduling**
    - Sign-up sheets (bản đăng ký)were used to reserve processor time
      - This could result in wasted computer idle time if the user finished early
      - If problems occurred the user could be forced to stop before resolving the problem
  - **Setup time**
    - A single program could involve
      - Loading the compiler plus the source program into memory
      - Saving the compiled program
      - Loading and linking together the object program and common functions

Simple Batch System:

Memory Layout for a Resident Monitor

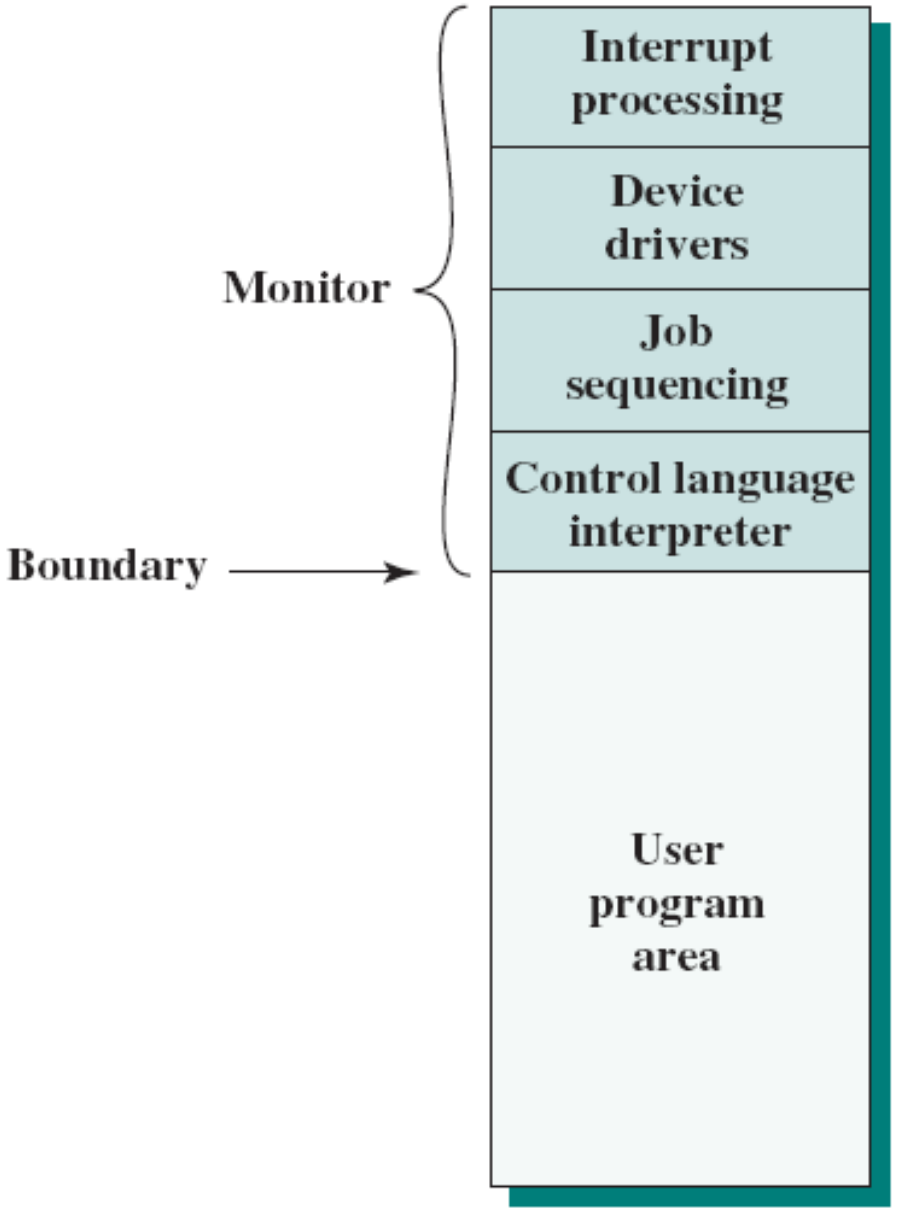


Figure 8.3 Memory Layout for a Resident Monitor

From the View of the Processor . . .

- Processor executes instructions from the portion of main memory containing the monitor
  - These instructions cause the next job to be read in another portion of main memory
  - The processor executes the instruction in the user's program until it encounters an ending or error condition
  - Either event causes the processor to fetch its next instruction from the monitor program
- The monitor handles setup and scheduling
  - A batch of jobs is queued up and executed as rapidly as possible with no idle time
- Job control language (JCL)
  - Special type of programming language used to provide instructions to the monitor

From the View of the Processor . . .

- Example:
  - $JOB
  - $FTN
  - … Some Fortran instructions
  - $LOAD
  - $RUN
  - … Some data
  - $END

**Each FORTRAN instruction and each item of data is on a separate punched card or a separate record on tape. In addition to FORTRAN and data lines, the job includes job control instructions, which are
denoted by the beginning "$".

- Monitor, or batch OS, is simply a computer program
  - It relies on the ability of the processor to fetch instructions from various portions of main memory in order to seize (nắm lấy) and relinquish (từ bỏ) control alternately

Desirable Hardware Features

- **Memory protection**
  - User program must not alter the memory area containing the monitor
  - The processor hardware should detect an error and transfer control to the monitor
  - The monitor aborts the job, prints an error message, and loads the next job
- **Timer**
  - Used to prevent a job from monopolizing (độc chiếm) the system
  - If the timer expires an interrupt occurs and control returns to monitor
- **Privileged instructions**
  - Lệnh đặc biệt, nhạy cảm
  - Can only be executed by the monitor
  - If the processor encounters such an instruction while executing a user program an error interrupt occurs
  - I/O instructions are privileged so the monitor retains control of all I/O devices
- **Interrupts**
  - Gives the OS more flexibility in relinquishing control to and regaining control from user programs

System Utilization Example

Read one record from file	15 μs
Execute 100 instructions	1 μs
Write one record to file	15 μs
TOTAL	31 μs

$$\text{Percent CPU utilization} = \frac{1}{31} = 0.032 = 3.2\%$$

Figure 8.4 System Utilization Example

The processor is often idle ➔ Multiple jobs can be carried out.

Multiprogramming Example

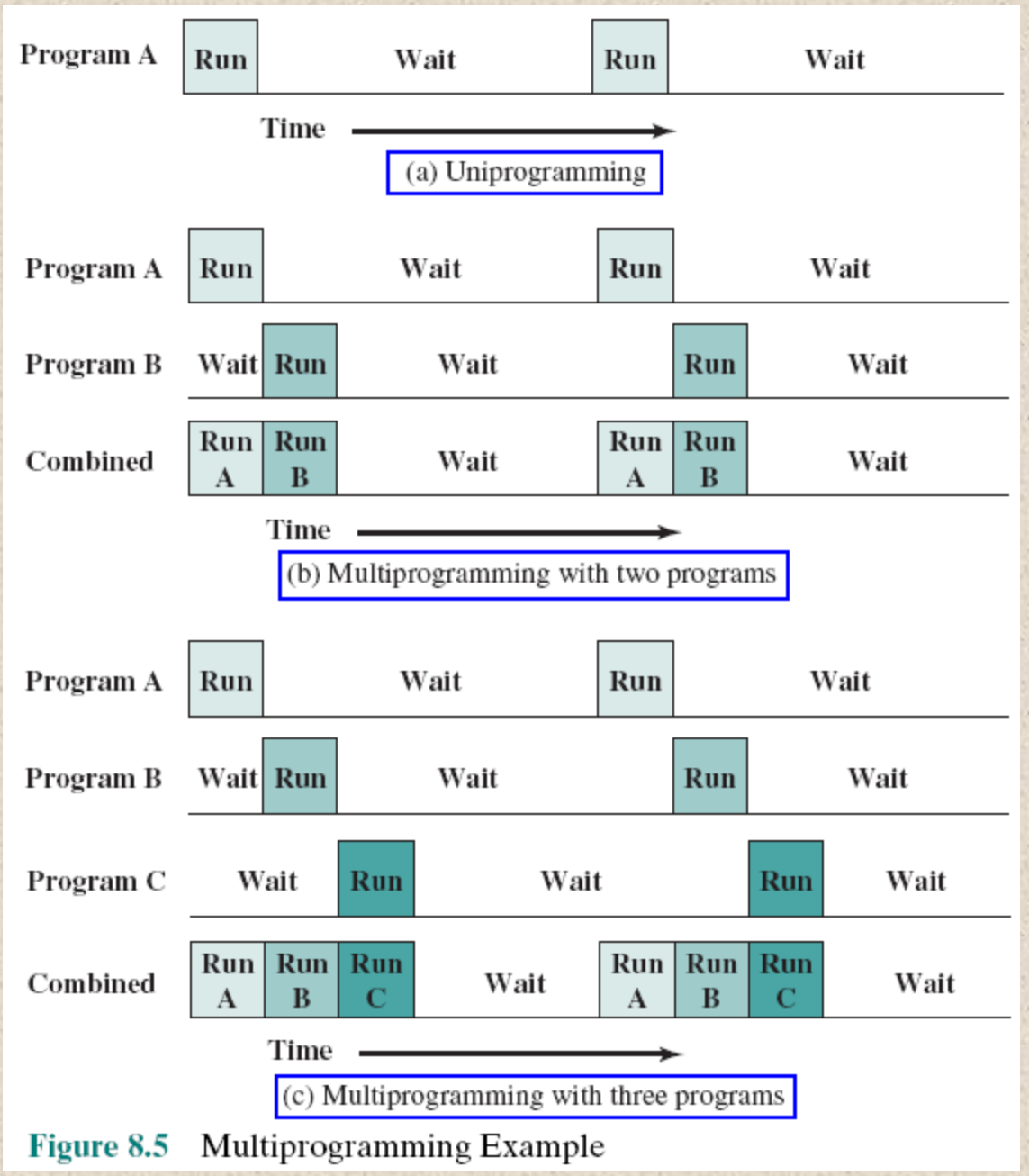


Figure 8.5 Multiprogramming Example

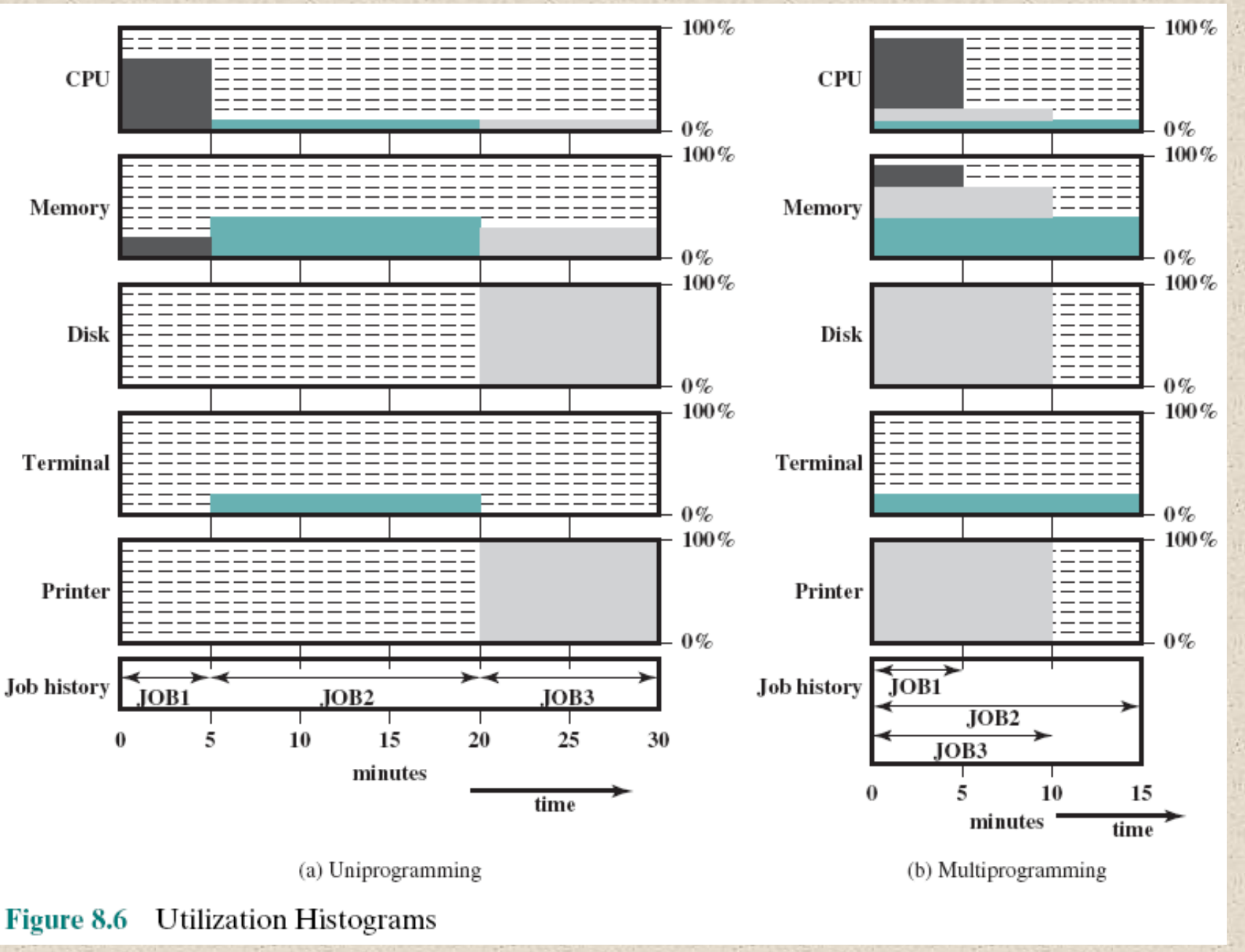


Figure 8.6 Utilization Histograms

Time Sharing Systems

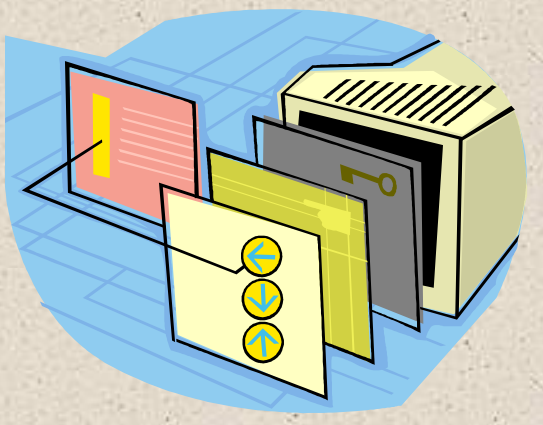
- Used when the user interacts directly with the computer
- Processor's time is shared among multiple users
- Multiple users simultaneously access the system through terminals, with the OS interleaving the execution of each user program in a short burst or quantum (time slice, time slot) of computation
- Example:
  - If there are *n* users actively requesting service at one time, each user will only see on the average 1/*n* of the effective computer speed

Batch Multiprogramming versus Time Sharing

Table 8.3 Batch Multiprogramming versus Time Sharing

	Batch Multiprogramming	Time Sharing
Principal objective	Maximize processor use	Minimize response time
Source of directives to operating system	Job control language commands provided with the job	Commands entered at the terminal

8.2- Scheduling



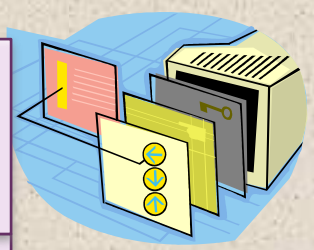
- The key to multiprogramming
- Four types are typically involved:

Table 8.4 Types of Scheduling

Long-term scheduling	The decision to add to the pool of processes to be executed
Medium-term scheduling	The decision to add to the number of processes that are partially or fully in main memory
Short-term scheduling	The decision as to which available process will be executed by the processor
I/O scheduling	The decision as to which process's pending I/O request shall be handled by an available I/O device

Program: executable file stored in external memory
Process: program in execution

Scheduling….



Memory

Which of them will run ?

Short-term Scheduling (current processes)

Process 0
Process 1
Process 2
Process 3

Process Table

P0's mem.
P2's mem.
P4's mem.
P1's mem.

Long-term Scheduling (waiting programs)

Accept ?

New process

Where will be used to load it ?

Medium-term Scheduling (blocked processes)

Each element of the process table is called as **P**rocess **C**ontrol **B**lock

Long Term Scheduling

Determines which programs are submitted for processing

Once submitted, a job becomes a process for the short term scheduler

In some systems a newly created process begins in a swapped-out condition, in which case it is added to a queue for the medium-term scheduler

Swapped-out process: Process stored in RAM must be write to disk. It shields it's memory to swapped-in process.

Batch system

- Newly submitted jobs are routed to disk and held in a batch queue
- The long-term scheduler creates processes from the queue when it can

Time-sharing system

- A process request is generated when a user attempts to connect to the system
- OS will accept all authorized comers until the system is saturated
- At that point a connection request is met with a message indicating that the system is full and to try again later

Medium-Term Scheduling and Short-Term Scheduling

Medium-Term

- Part of the swapping function
- Swapping-in decision is based on the need to manage the degree of multiprogramming
- Swapping-in decision will consider the memory requirements of the swapped-out processes

Short-Term

- Also known as the dispatcher (trình điều phối)
- Executes frequently and makes the fine-grained decision of which job to execute next

Short-Term Scheduling

Five State Process Model

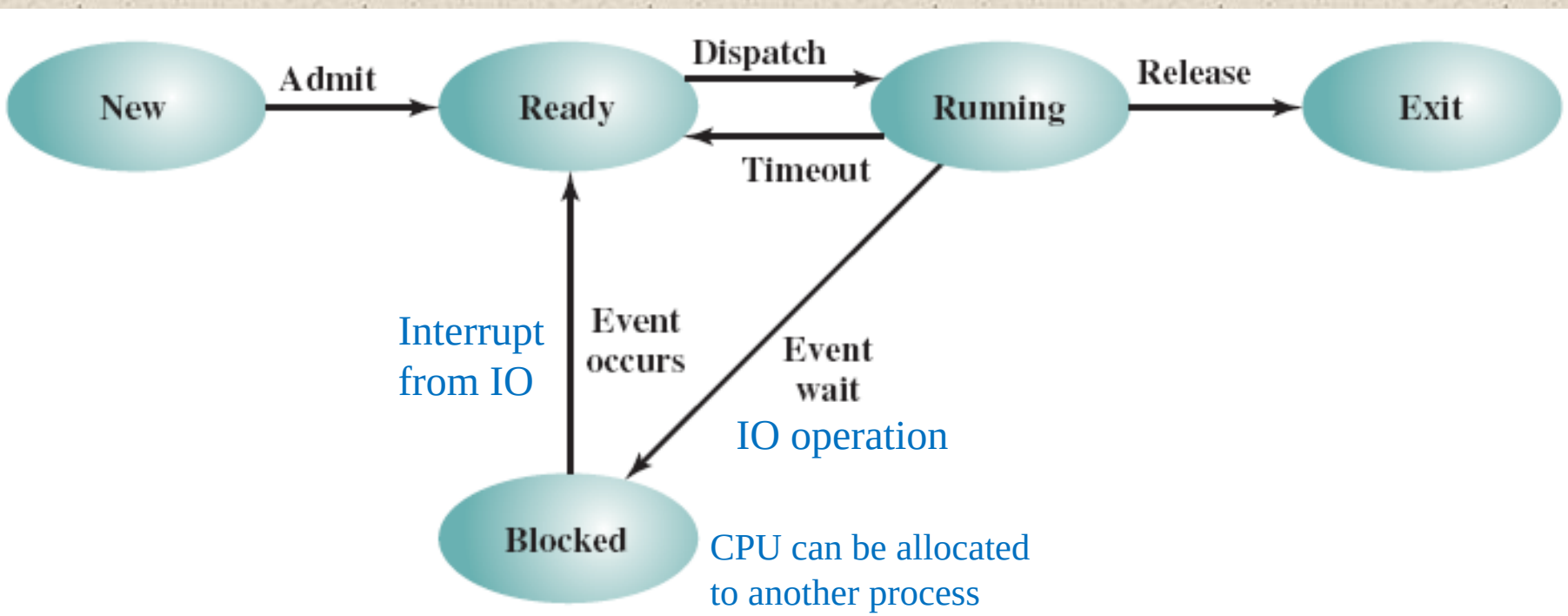


Figure 8.7 Five-State Process Model

Process Control Block (PCB)

What are metadata of a process?

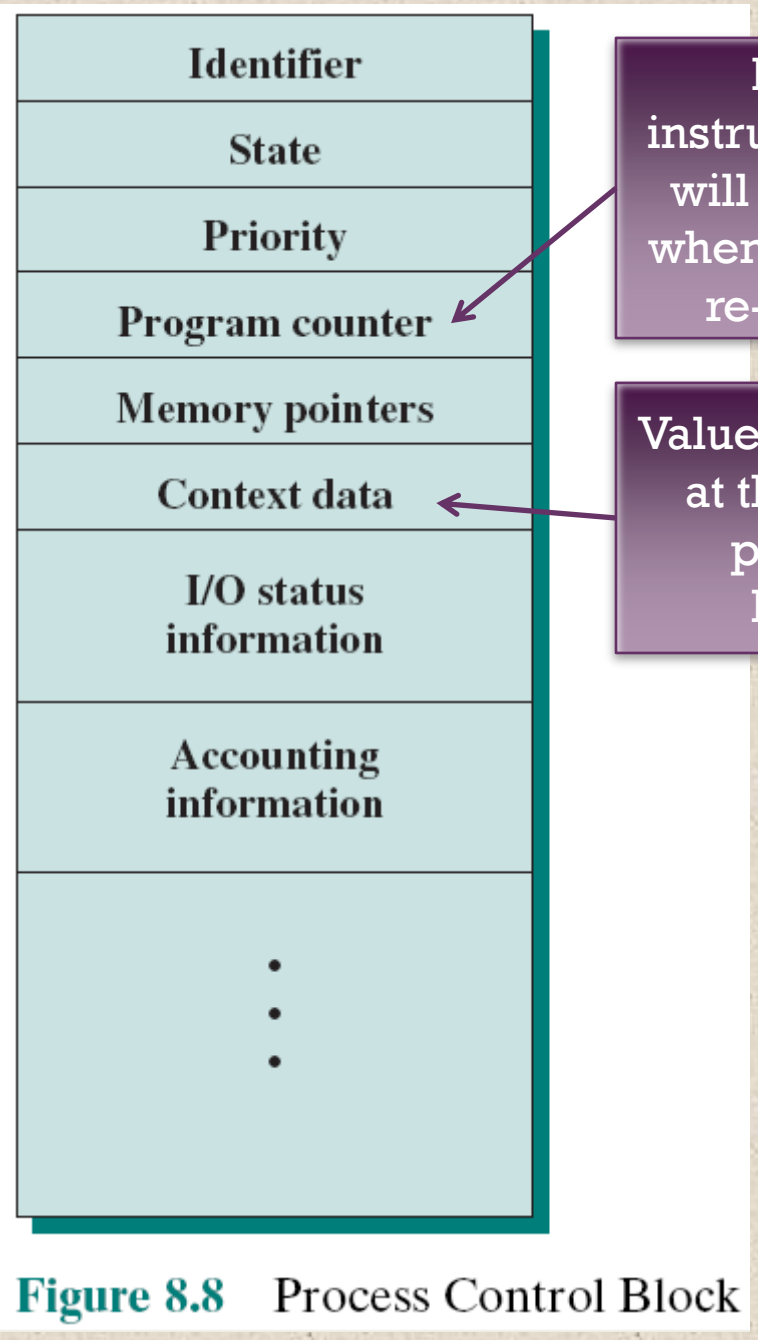


Figure 8.8 Process Control Block

Scheduling Example

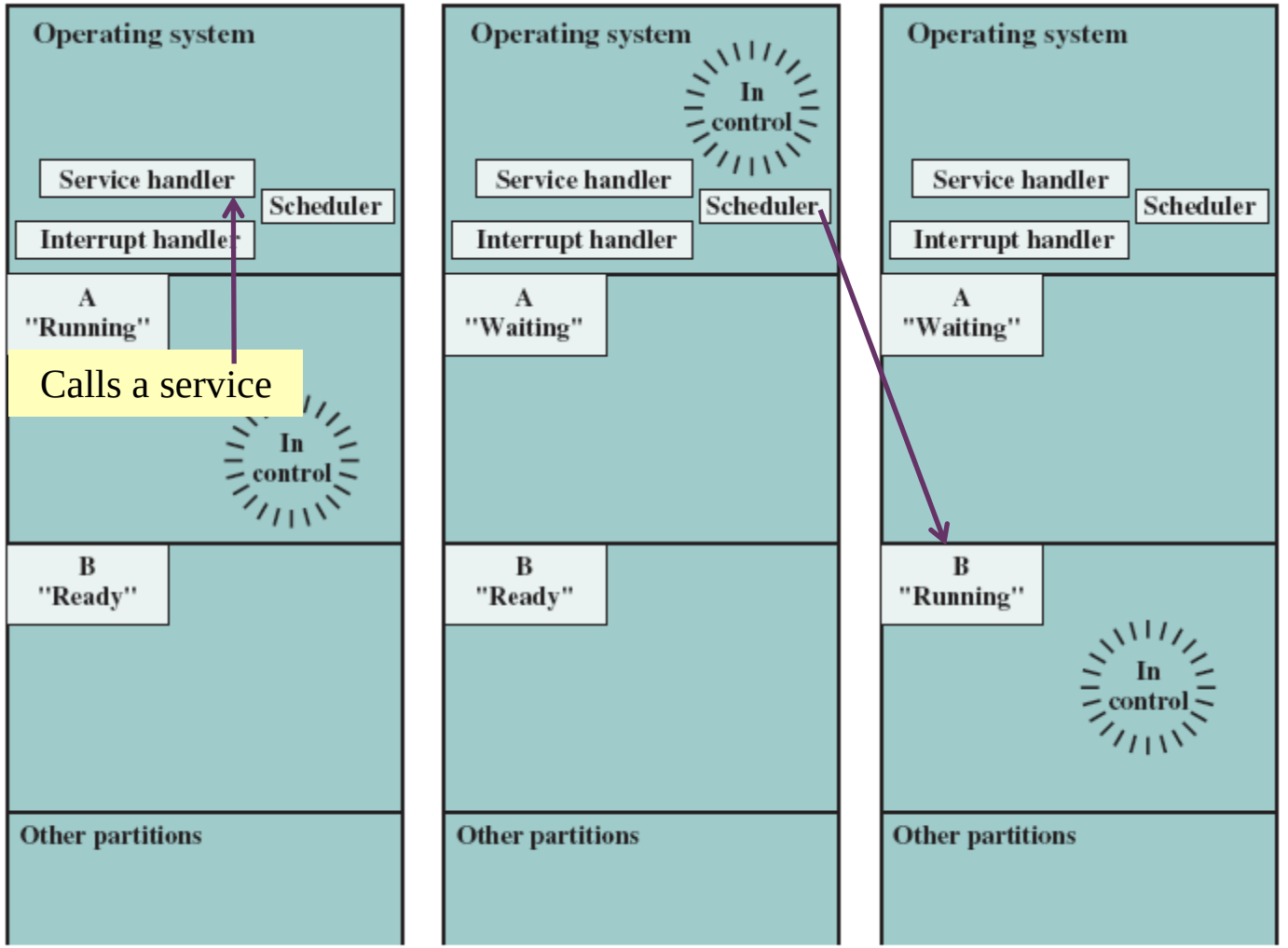


Figure 8.9 Scheduling Example

Key Elements of O/S

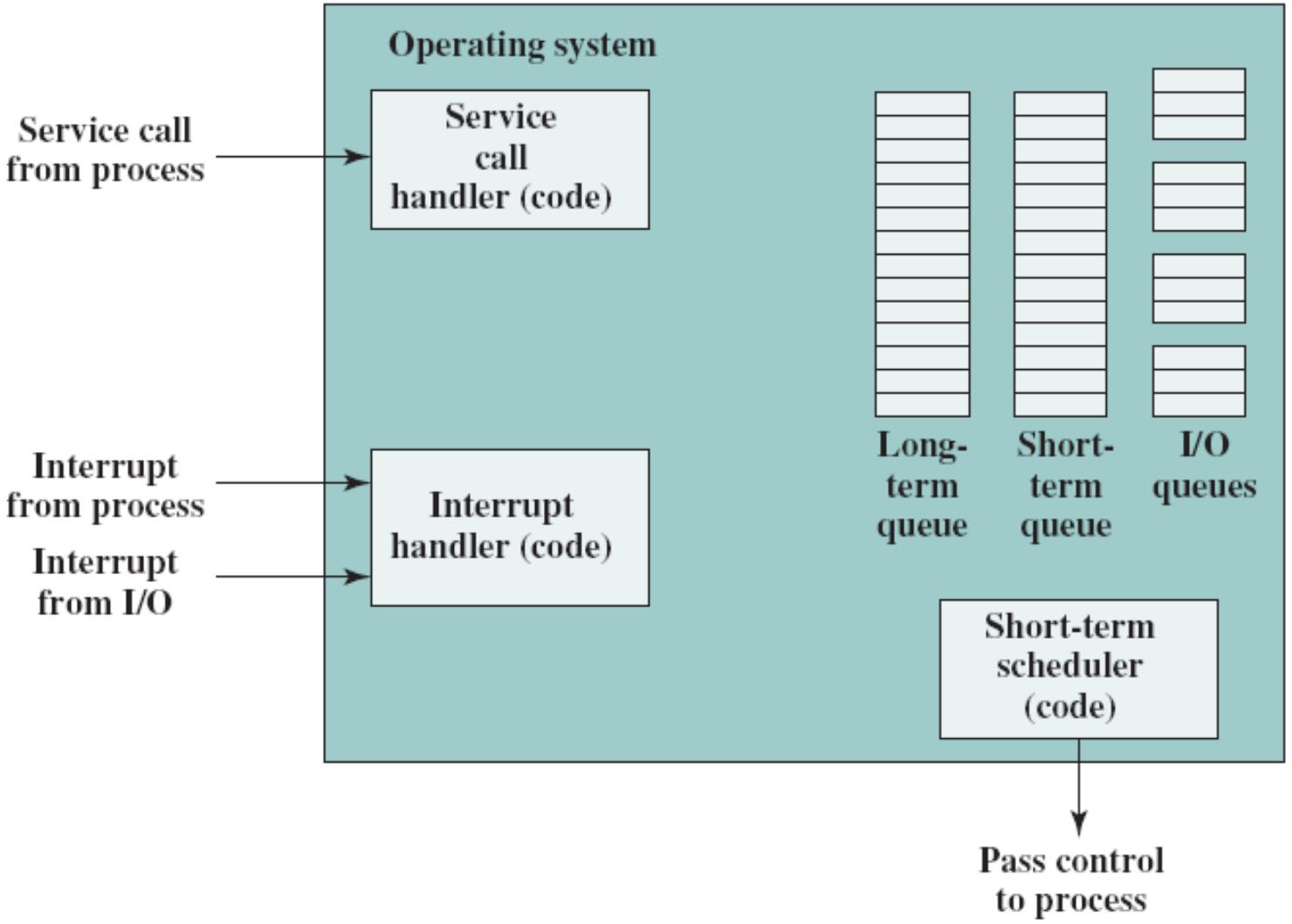


Figure 8.10 Key Elements of an Operating System for Multiprogramming

Process Scheduling

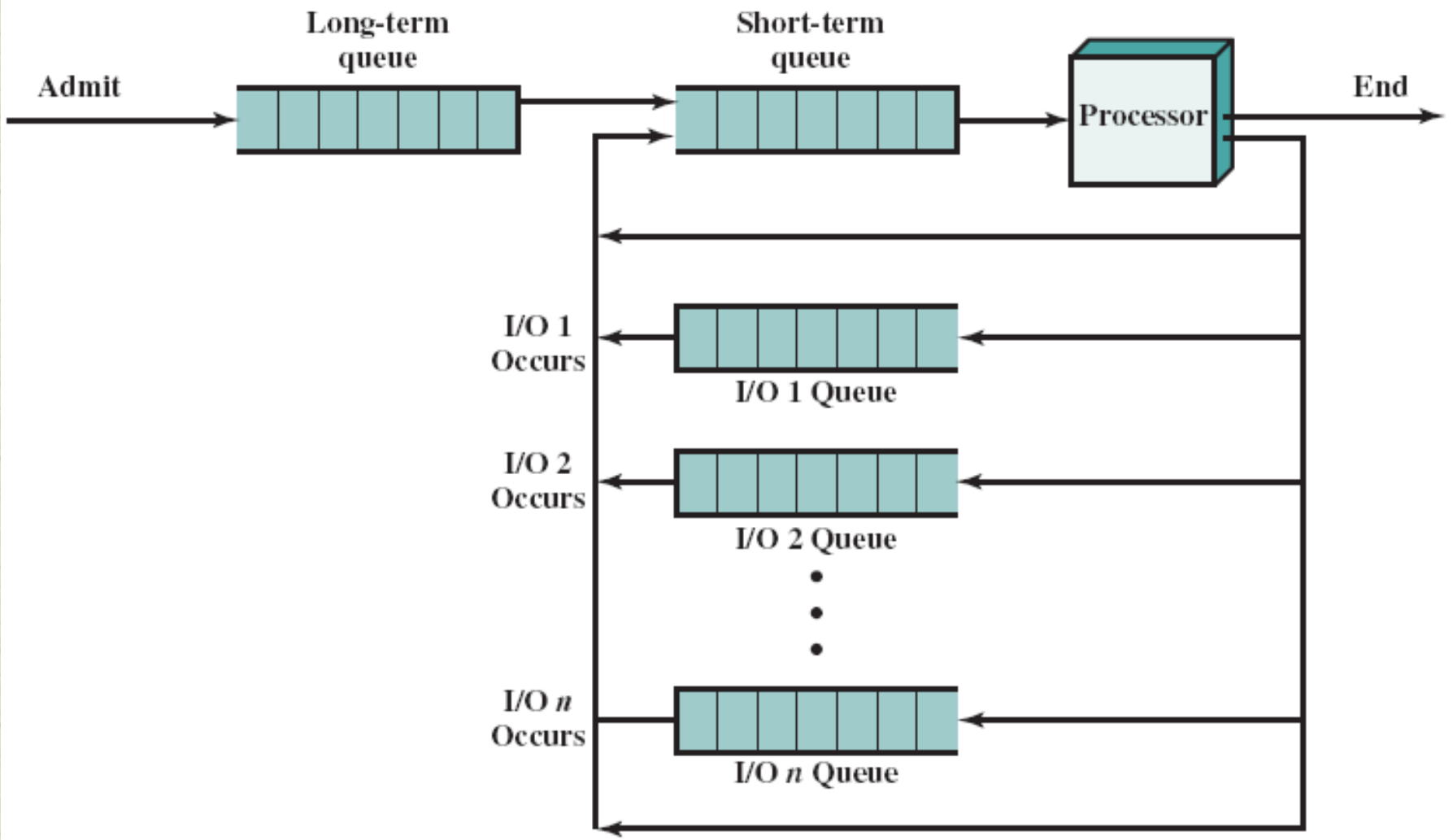


Figure 8.11 Queuing Diagram Representation of Processor Scheduling

8.3- Memory Management

- Memory Management
  - Swapping
  - Partioning
  - Paging
  - Virtual Memory
  - Translation Lookaside Buffer
  - Segmentation

Memory Manager is a part of OS which bears responsibility to manage computer memory at the system level and some above techniques can be applied.

Memory Management: Swapping

Why?

Memory has larger size → Processes request more and more memory, more processes need to run → Memory is not enough to supply → A selected process must be swapped out to disk in order to load new process (SWAP)

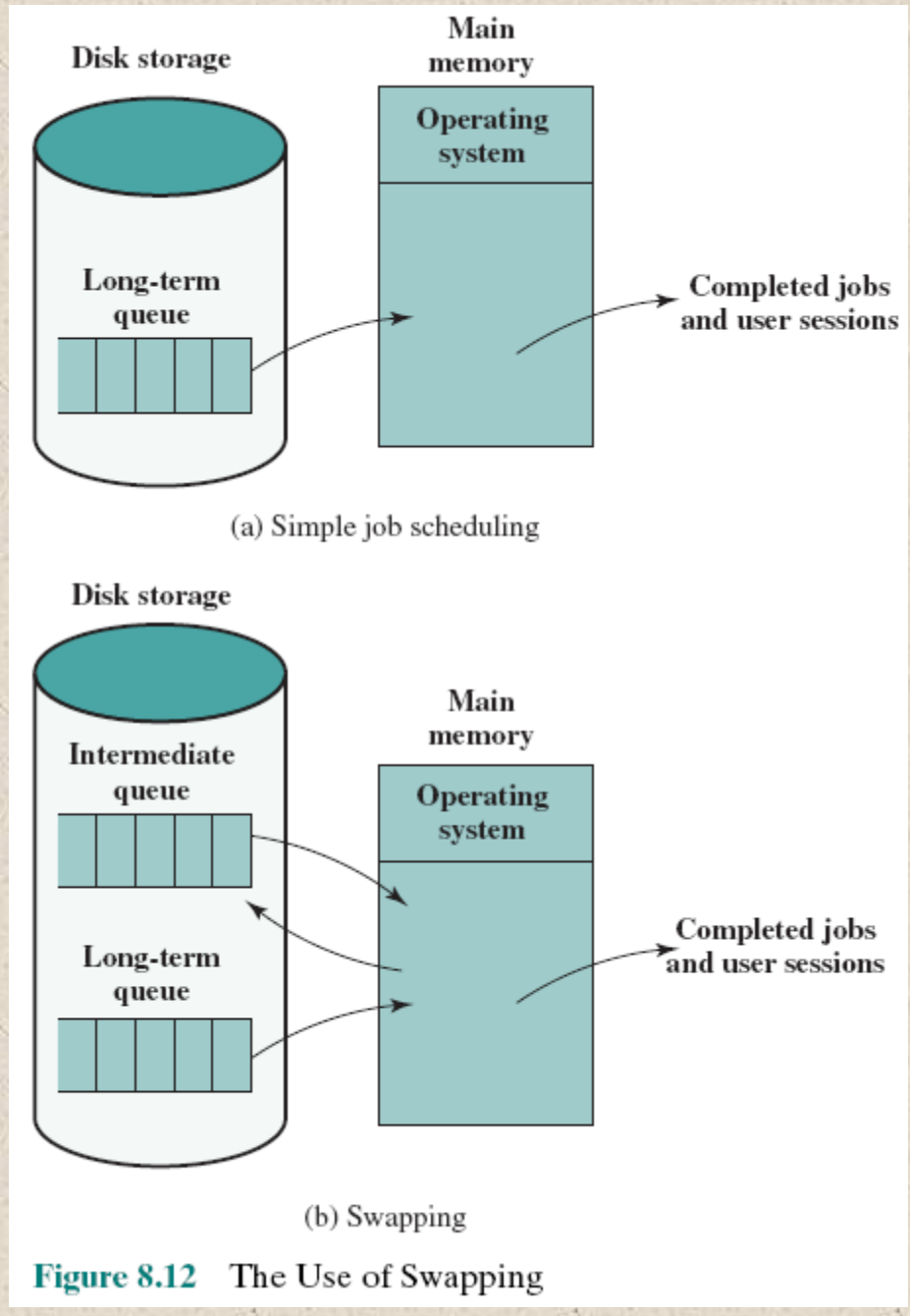
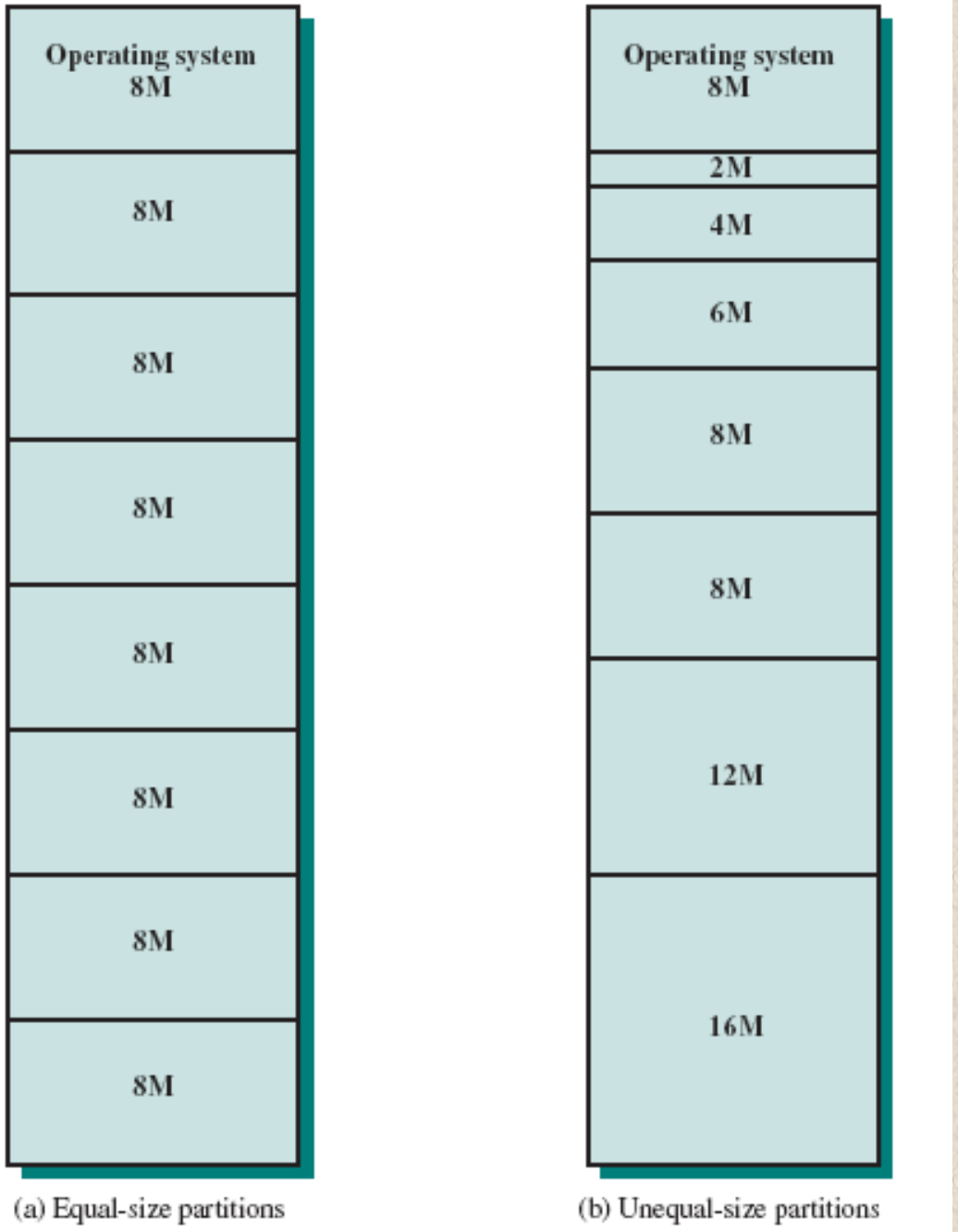


Figure 8.12 The Use of Swapping

Memory Management

Partitioning

Smaller process needs smaller memory → Unequal-size partition is better.



(a) Equal-size partitions (b) Unequal-size partitions

Figure 8.13 Example of Fixed Partitioning of a 64-Mbyte Memory

Effect of Dynamic Partitioning

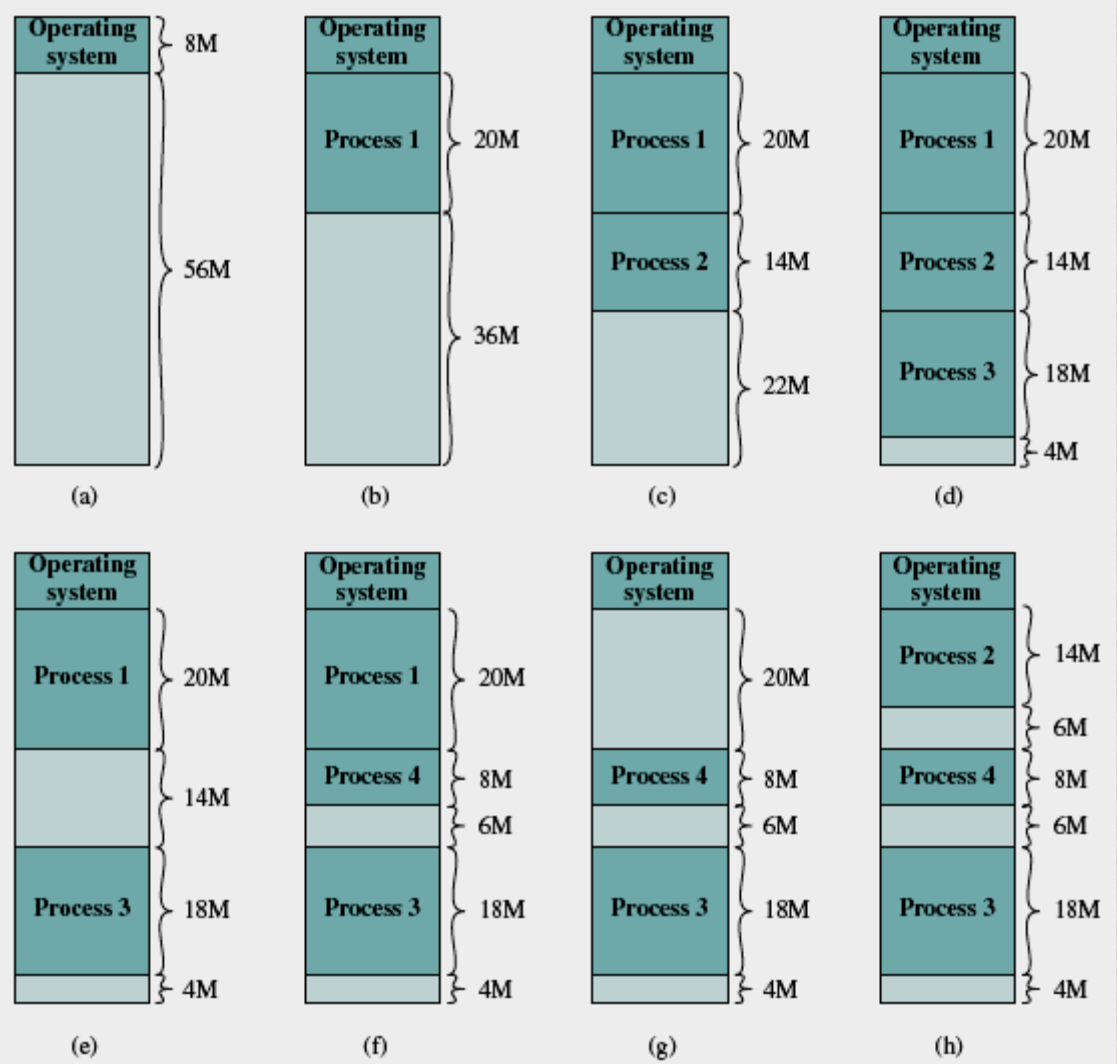


Figure 8.14 The Effect of Dynamic Partitioning

To enable loading a process to any position of memory, program addresses must be expressed as **logical addresses**

Logical address

- expressed as a location relative to the beginning of the program (**offset**)

Physical address

- an actual location in main memory

Base address

- current starting location of the process

Memory Management

Paging

At a time, only one instruction of the current process executes → Only necessary part of each process is loaded → Many processes can be loaded.

Programs are divided into small fixed chunk (ex. 4KB). At a time, only some pages of each process are loaded to memory (frames)

Memory is divided also to frames

Frame size= Page size

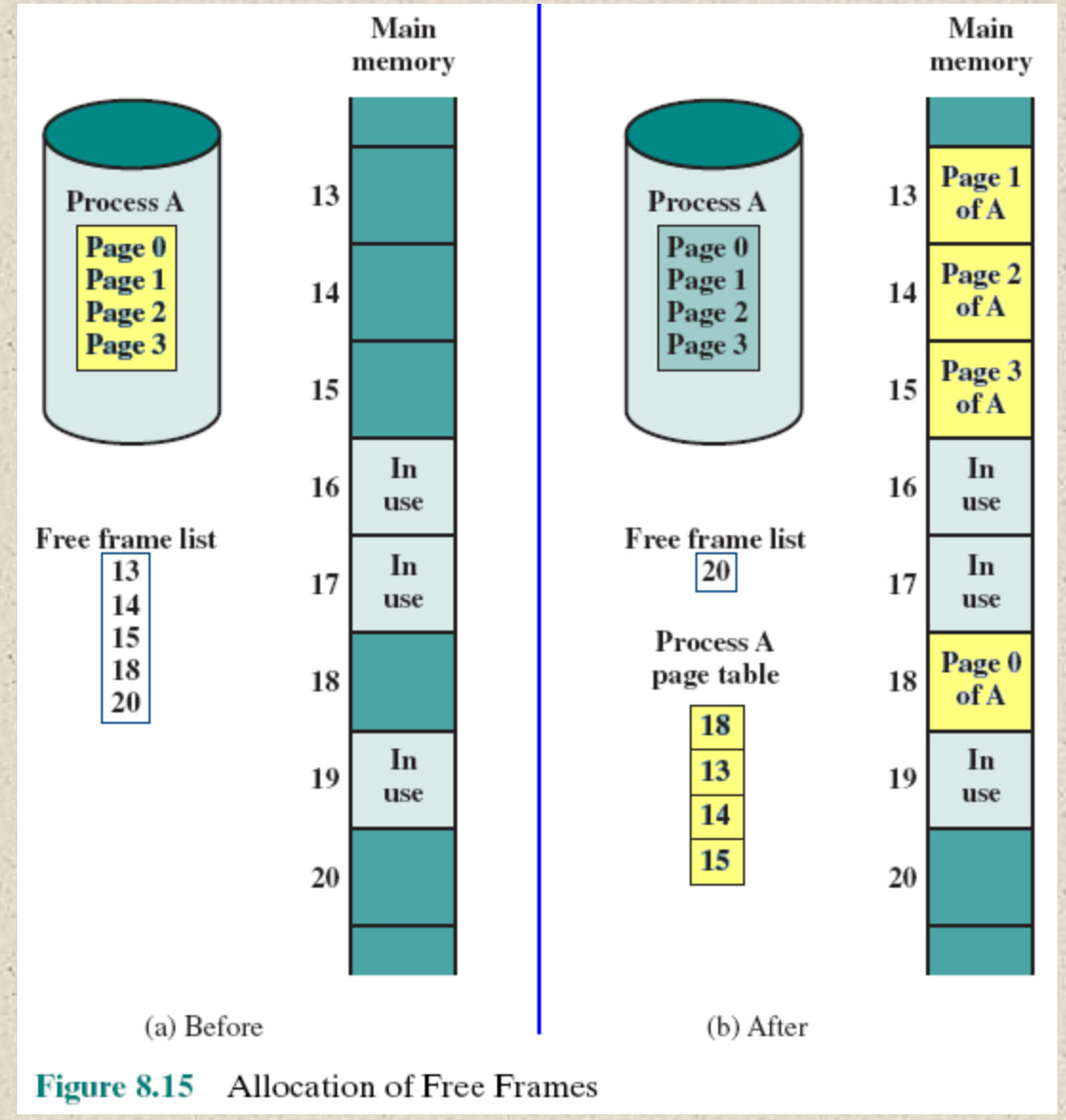


Figure 8.15 Allocation of Free Frames

Loading 4 frames of the process A

Paging

Logical and Physical Addresses

Each program address is expressed as a logical address which is a pair of (page, offset)

How to determine physical address from a logical address?

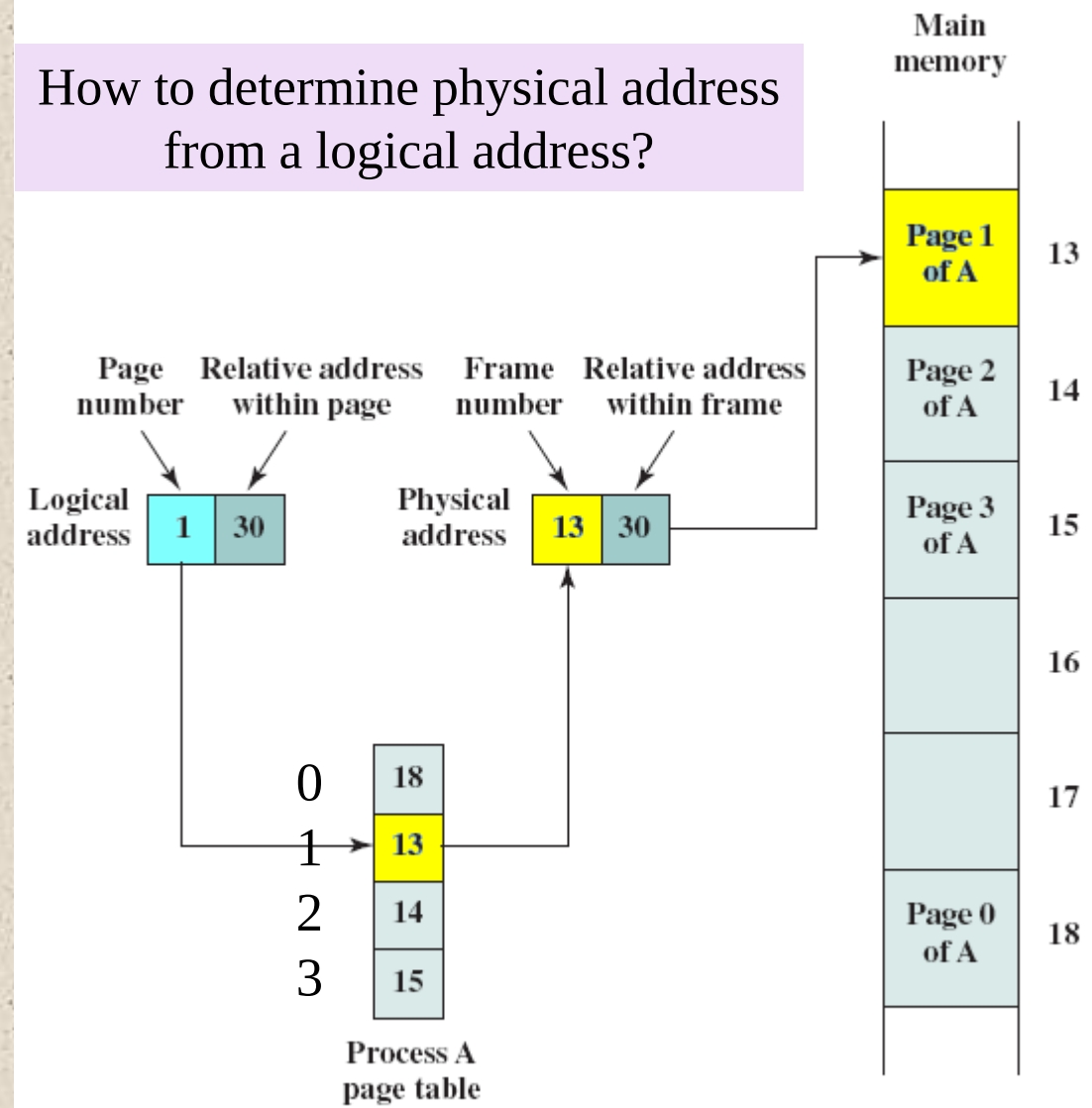


Figure 8.16 Logical and Physical Addresses

Virtual Memory: Demand Paging

- **Each page of a process is brought in only when it is needed**
- **Principle of locality**
  - When working with a large process execution may be **confined (limited) to a small section of a program** (subroutine)
  - It is better use of memory to load in just a few pages
  - If the program **references** data or branches to an instruction on a page not in main memory, **a *page fault*** is triggered which tells the OS to bring in the desired page

Virtual Memory: Demand Paging

- **Advantages:**
  - More processes can be maintained in memory
  - Time is saved because unused pages are not swapped in and out of memory
- **Disadvantages:**
  - When one page is brought in, another page must be thrown out (*page replacement*)
  - If a page is thrown out just before it is about to be used the OS will have to go get the page again
  - *Thrashing (đánh bại- hệ thống trì trệ)*
    - When the processor spends most of its time swapping pages rather than executing instructions

Paging: Inverted Page Table Structure

A large table is used to store all pages of all procsses

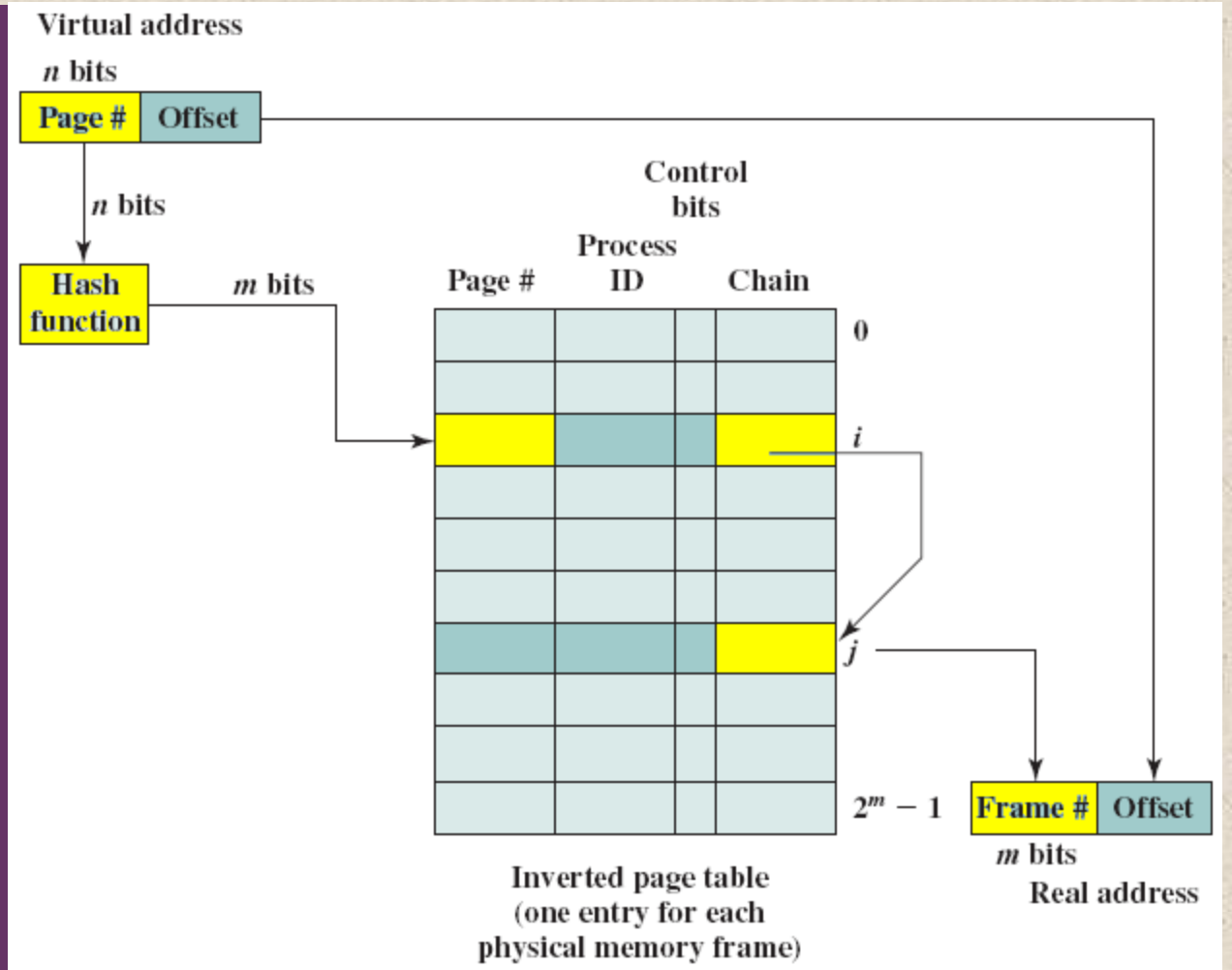


Figure 8.17 Inverted Page Table Structure

Hash function allows determine the position of a table in which data is stored.
Example: **h(n) = n modulo k (n%k)**
If h(n2) = position storing n1, n2 will be stored in the lower position (overflow area) and they are marked in the field **chain.**

Paging: Operation of Paging and Translation Lookaside Buffer (TLB)

TLB is an hardware including some registers. A part of page table is copied to them in order to increase performance of translating virtual addresses to physical addresses.

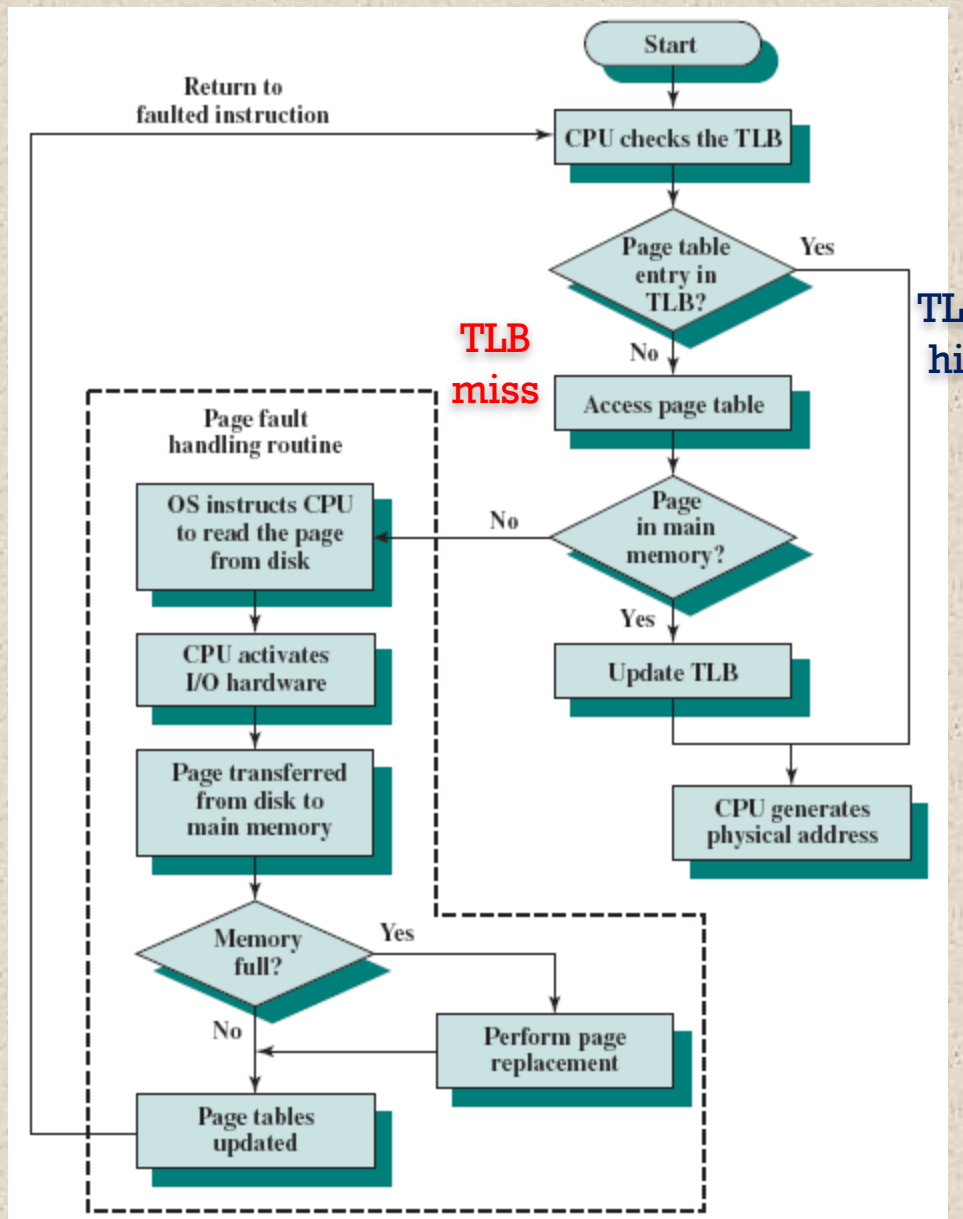


Figure 8.18 Operation of Paging and Translation Lookaside Buffer (TLB)

TLB and Cache Operation

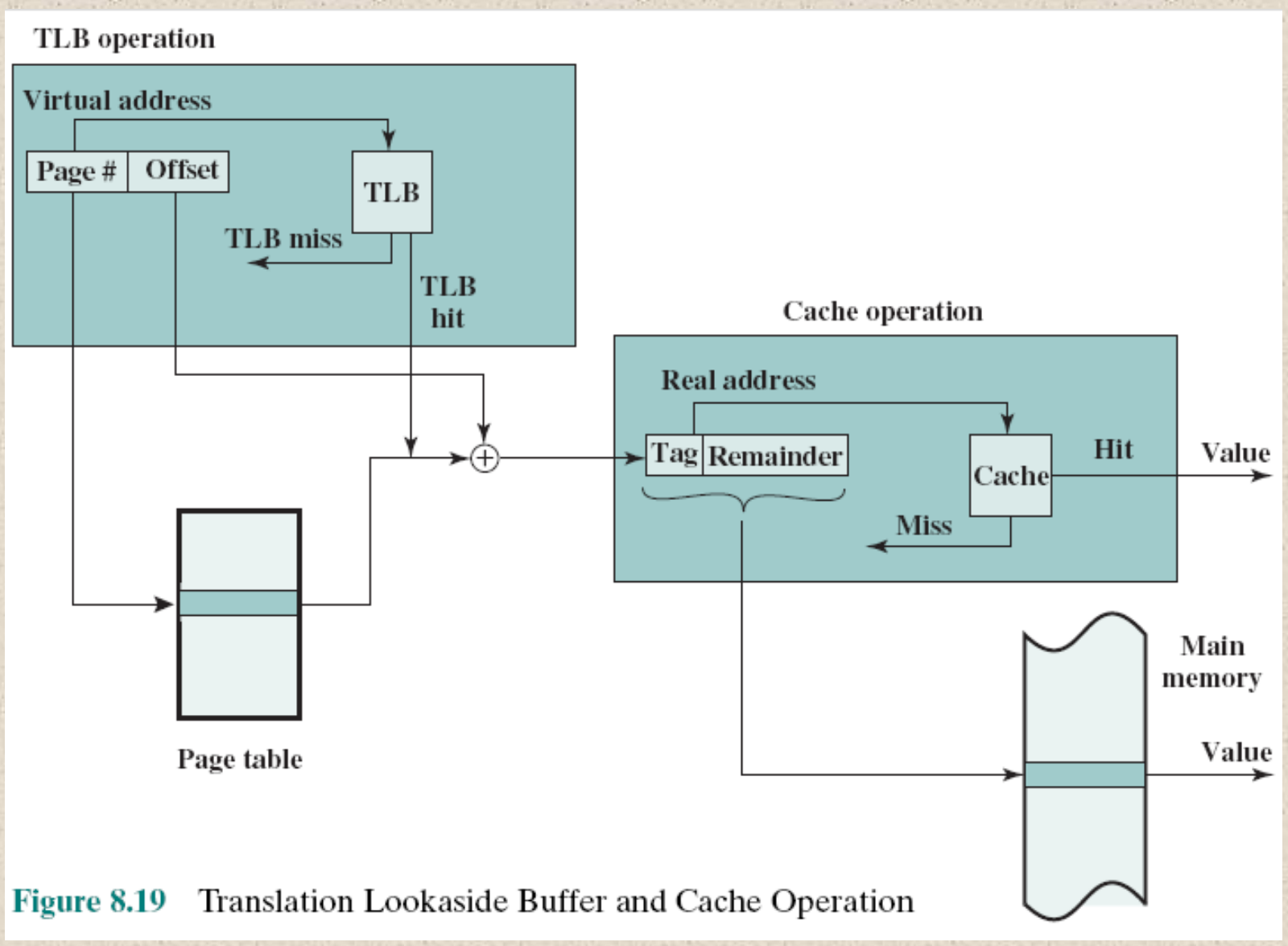


Figure 8.19 Translation Lookaside Buffer and Cache Operation

Segmentation

- **Program is divided in to segments (data, code, stack, heap segments)**

- Usually visible to the programmer
- Provided as a convenience for organizing programs and data and as a means for associating privilege and protection attributes with instructions and data
- Allows the programmer to view memory as consisting of multiple address spaces or segments
- A segment can be divided into some pages.

- **Advantages:**
  - Simplifies the handling of growing data structures
  - Allows programs to be altered and recompiled independently without requiring that an entire set of programs be re-linked and re-loaded
  - A segment can be shared among processes
  - A segment can be added individual protection

Exercises

- 8.1 What is an operating system?
- 8.2 List and briefly define the key services provided by an OS.
- 8.3 List and briefly define the major types of OS scheduling.
- 8.4 What is the difference between a process and a program?
- 8.5 What is the purpose of swapping?
- 8.6 If a process may be dynamically assigned to different locations in main memory, what is the implication for the addressing mechanism?
- 8.7 Is it necessary for all of the pages of a process to be in main memory while the process is executing?
- 8.8 Must the pages of a process in main memory be contiguous?
- 8.9 Is it necessary for the pages of a process in main memory to be in sequential order?
- 8.10 What is the purpose of a translation lookaside buffer?

Summary

Chapter 8

Operating System Support

- Operating system objectives and functions
- Types of operating systems
- Scheduling
  - Long-term scheduling
  - Medium-term scheduling
  - Short-term scheduling
- Memory management
  - Swapping
  - Partitioning
  - Paging
  - Virtual memory
  - Translation lookaside buffer
  - Segmentation