Operating System

Chapter 7. Memory Management

Lynn Choi

School of Electrical Engineering

Memory Management

❑ Terminology

Frame	A fixed-length block of main memory.
Page	A fixed-length block of data that resides in secondary memory (such as disk). A page of data may temporarily be copied into a frame of main memory.
Segment	A variable-length block of data that resides in secondary memory. An entire segment may temporarily be copied into an available region of main memory (segmentation) or the segment may be divided into pages which can be individually copied into main memory (combined segmentation and paging).

Source: Pearson

❑ Requirements – should provide the following functions

- *Relocation*
- *Protection*
- *Sharing*

Relocation, Protection & Sharing

Relocation

- Programmers typically do not know which other programs will be resident in main memory at the time of execution of their program
- Active processes need to be swapped in and out of main memory to maximize processor utilization
- A process may need to be placed in a different area of memory when it is swapped back

Protection

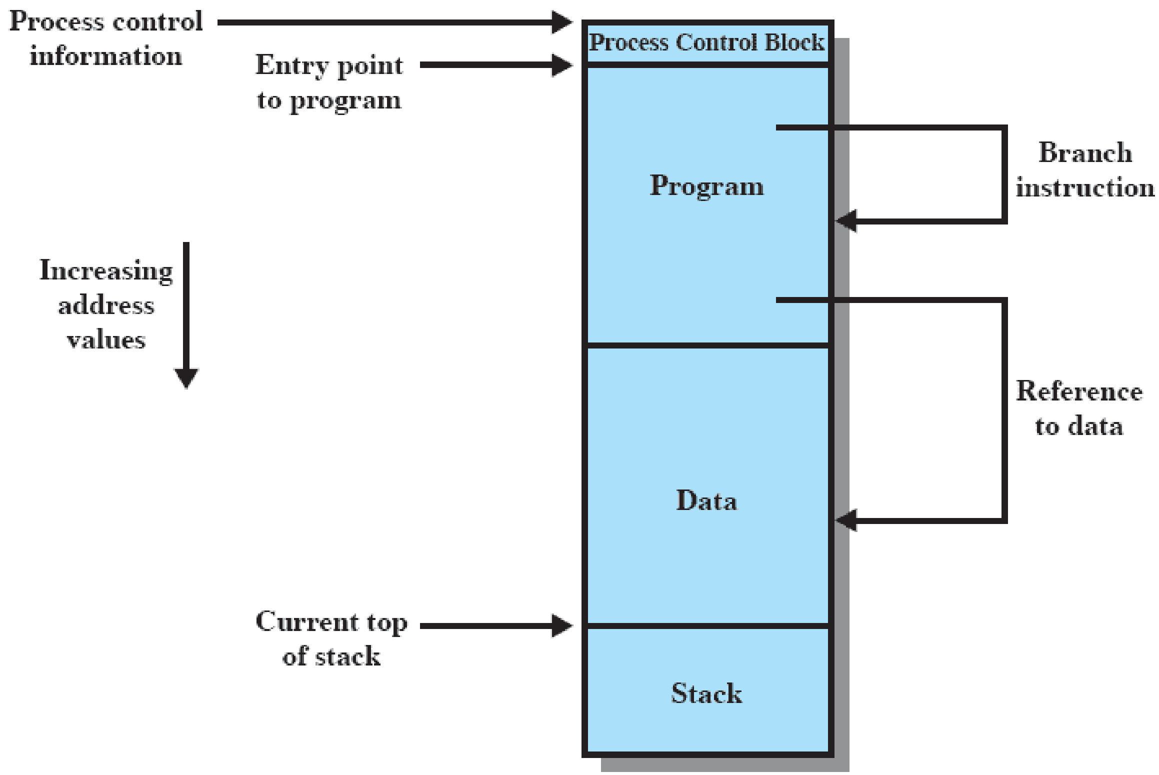
- Processes need to acquire (read, write) permission to reference memory locations
- Memory references generated by a process must be checked at run time to check if they have permissions (*access rights*)
- Mechanisms that support relocation also support protection

Sharing

- Allow each process to access to the same copy of a program rather than having its own separate copy
- Should provide controlled access to shared areas without compromising protection
- Mechanisms that support relocation also support sharing

Addressing Requirement of a Process

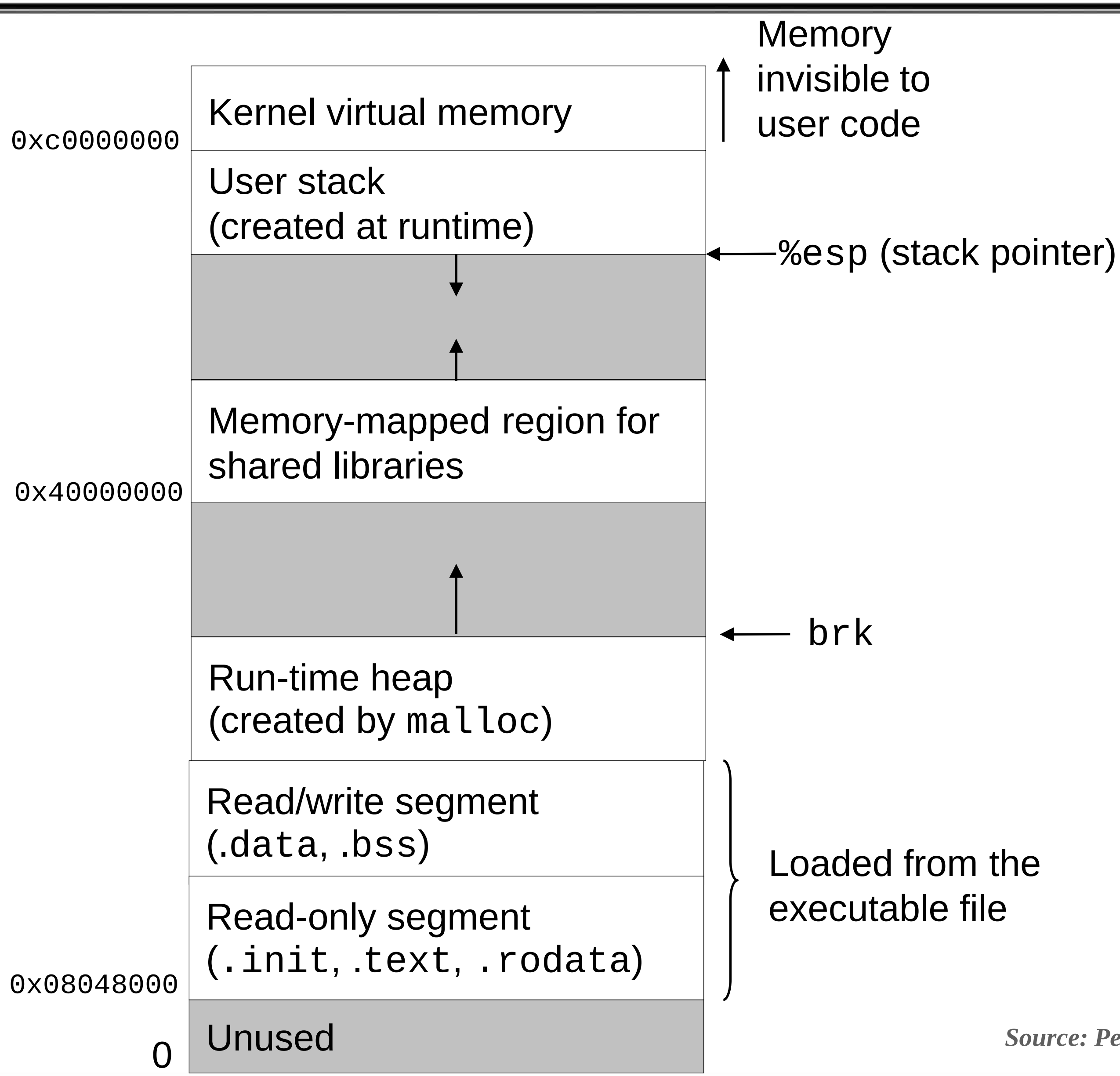
- For relocation, OS must know the location of program (instructions, data) as well as its PCB



Source: Pearson

Figure 7.1 Addressing Requirements for a Process

Process Image



Source: Pearson

Logical/Physical Organization

- Main memory is organized as *a linear array of bytes*
  - While this organization closely mirrors the machine HW, it does not correspond to logical organization (user's view of memory)

Logical organization

- Users view programs as *a collection of modules*
  - Modules can be written and compiled independently
  - Different protection (read-only, execute-only) are given to different modules
  - Module level sharing corresponds to the user's way of viewing the problem
- ***Segmentation*** is the tool that most readily satisfies requirement

Physical organization

- Computer memory is organized as main memory and secondary memory
- Main memory available for a program may not be sufficient
  - Programmer does not know how much space will be available
- Cannot leave the programmer with the responsibility to manage memory
- ***Overlaying*** allows various modules to be assigned the same region of memory

Memory Management

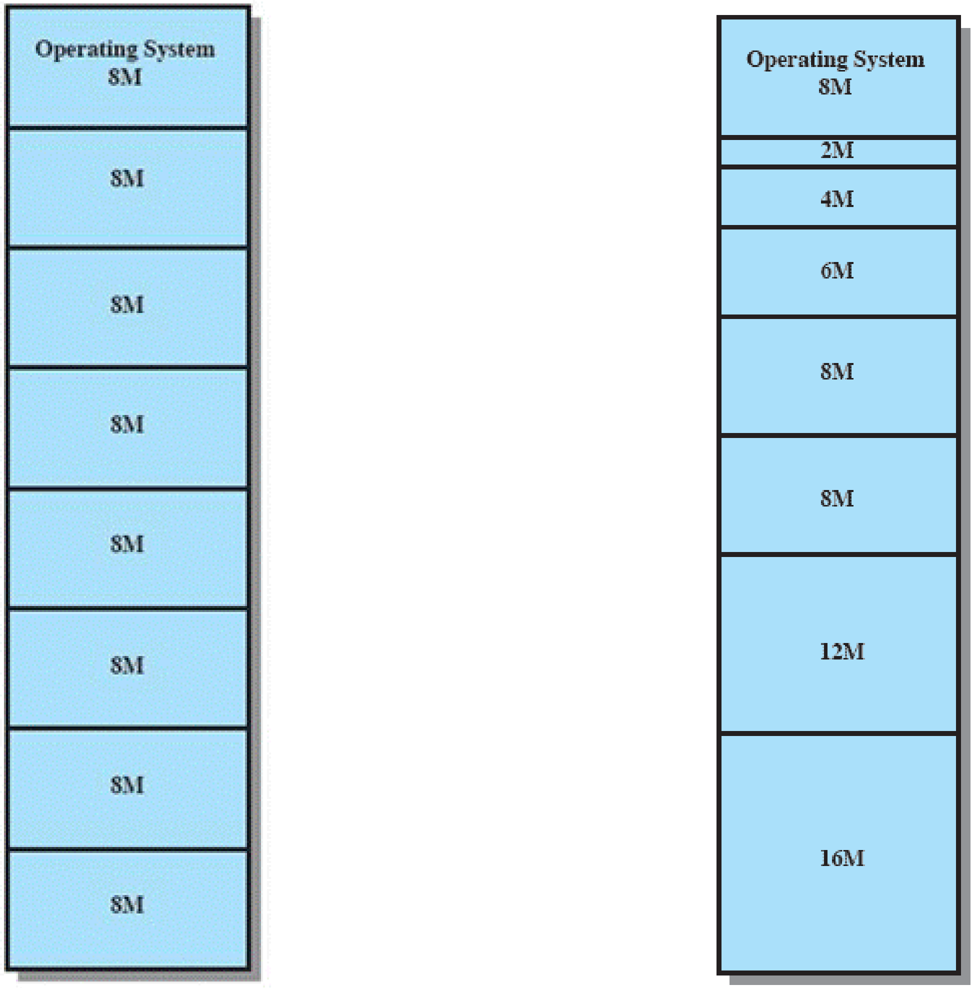
- **Memory management techniques involve**
  - Segmentation, overlaying, and virtual memory, etc.
  - We can classify these techniques as
    - *Memory partitioning* – used in old operating systems
    - *Virtual memory* based on paging and segmentation
- **Fixed partitioning**
  - Equal-size partitions
    - Any process which fit into the partition can be loaded into any available partition
    - Swap out a process if all partitions are full
    - Problems
      - A program may be too big to fit in a partition
        - Program needs to be designed with the use of *overlays*
      - ***Internal fragmentation***
        - Wasted space due inside a partition
  - Unequal-size partitions
    - Can lessen both of the problems
      - Large partitions can accommodate programs without overlays
      - Small partitions can reduce internal fragmentation

Memory Management Techniques

Technique	Description	Strengths	Weaknesses
Fixed Partitioning	Main memory is divided into a number of static partitions at system generation time. A process may be loaded into a partition of equal or greater size.	Simple to implement; little operating system overhead.	Inefficient use of memory due to internal fragmentation; maximum number of active processes is fixed.
Dynamic Partitioning	Partitions are created dynamically, so that each process is loaded into a partition of exactly the same size as that process.	No internal fragmentation; more efficient use of main memory.	Inefficient use of processor due to the need for compaction to counter external fragmentation.
Simple Paging	Main memory is divided into a number of equal-size frames. Each process is divided into a number of equal-size pages of the same length as frames. A process is loaded by loading all of its pages into available, not necessarily contiguous, frames.	No external fragmentation.	A small amount of internal fragmentation.
Simple Segmentation	Each process is divided into a number of segments. A process is loaded by loading all of its segments into dynamic partitions that need not be contiguous.	No internal fragmentation; improved memory utilization and reduced overhead compared to dynamic partitioning.	External fragmentation.
Virtual Memory Paging	As with simple paging, except that it is not necessary to load all of the pages of a process. Nonresident pages that are needed are brought in later automatically.	No external fragmentation; higher degree of multiprogramming; large virtual address space.	Overhead of complex memory management.
Virtual Memory Segmentation	As with simple segmentation, except that it is not necessary to load all of the segments of a process. Nonresident segments that are needed are brought in later automatically.	No internal fragmentation, higher degree of multiprogramming; large virtual address space; protection and sharing support.	Overhead of complex memory management.

Source: Pearson

Equal-size vs Unequal-size Partition



(a) Equal-size partitions

(b) Unequal-size partitions

Source: Pearson

Memory Assignment with Fixed Partitioning

Disadvantages

- The number of partitions specified by the system limits the number of active processes
- Small jobs will not utilize partition space efficiently

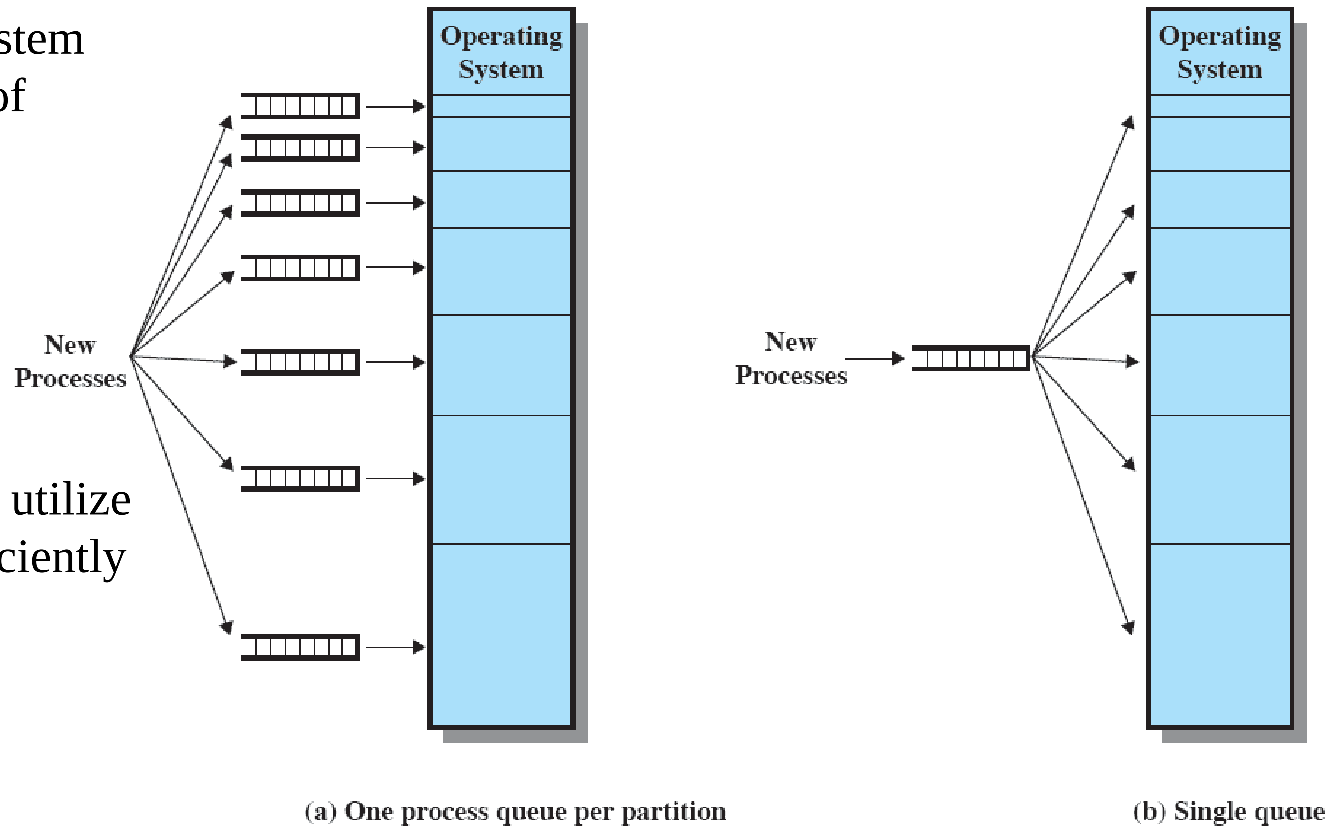


Figure 7.3 Memory Assignment for Fixed Partitioning

Source: Pearson

Dynamic Partitioning

Dynamic partitioning

- Process is allocated as much memory as it requires
- Partitions are of variable length and of variable numbers
- This technique was used by IBM's mainframe operating system, OS/MVT

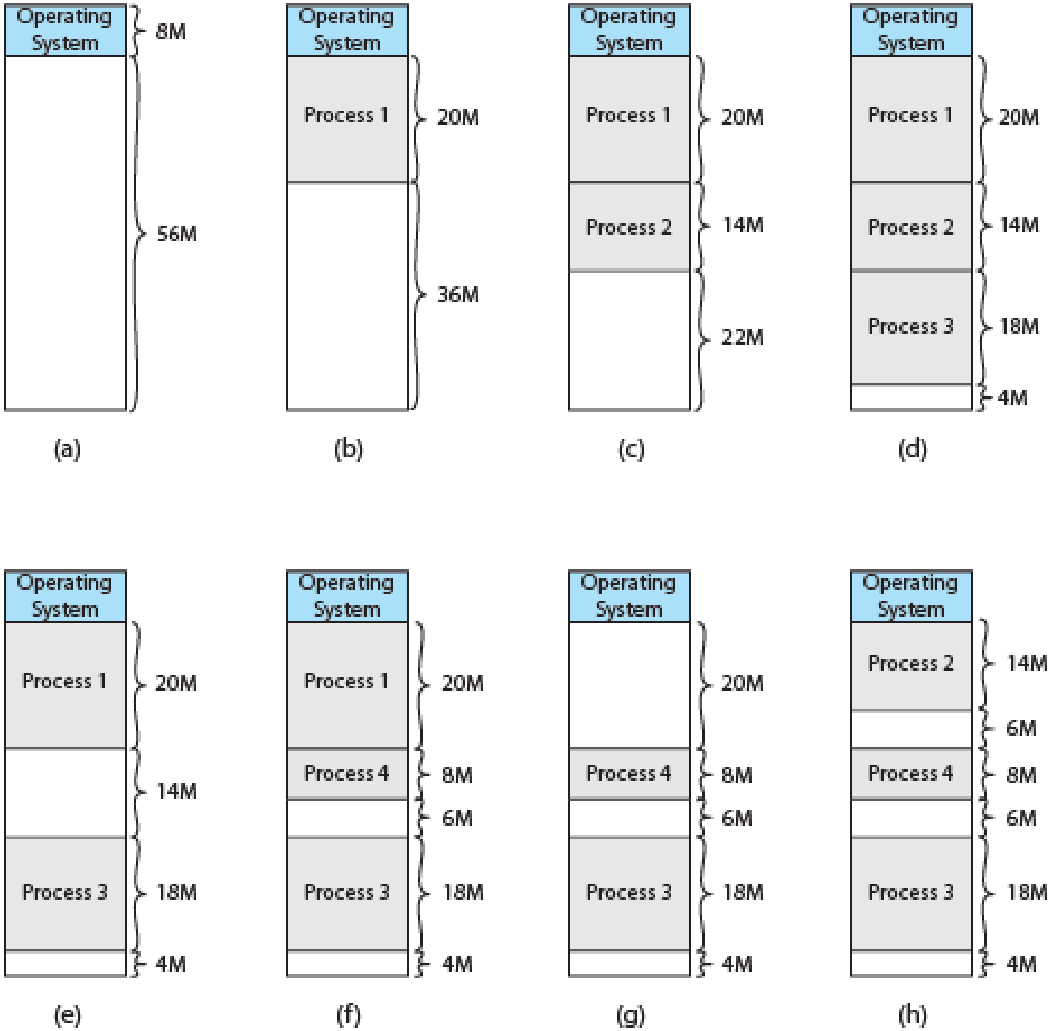
External fragmentation

- Memory becomes more and more fragmented
- Memory utilization declines

Compaction

- Technique to overcome external fragmentation
- OS shifts processes so that they are contiguous
- It is a time consuming process, wasting CPU time

Effect of Dynamic Partitioning



Source: Pearson

Figure 7.4 The Effect of Dynamic Partitioning

Placement Algorithms

First fit

- Search list from the beginning, choose the first free block that fits
- Can take linear time in total number of blocks (allocated and free)
- (+) Tend to retain large free blocks at the end
- (-) Leave small free blocks at beginning

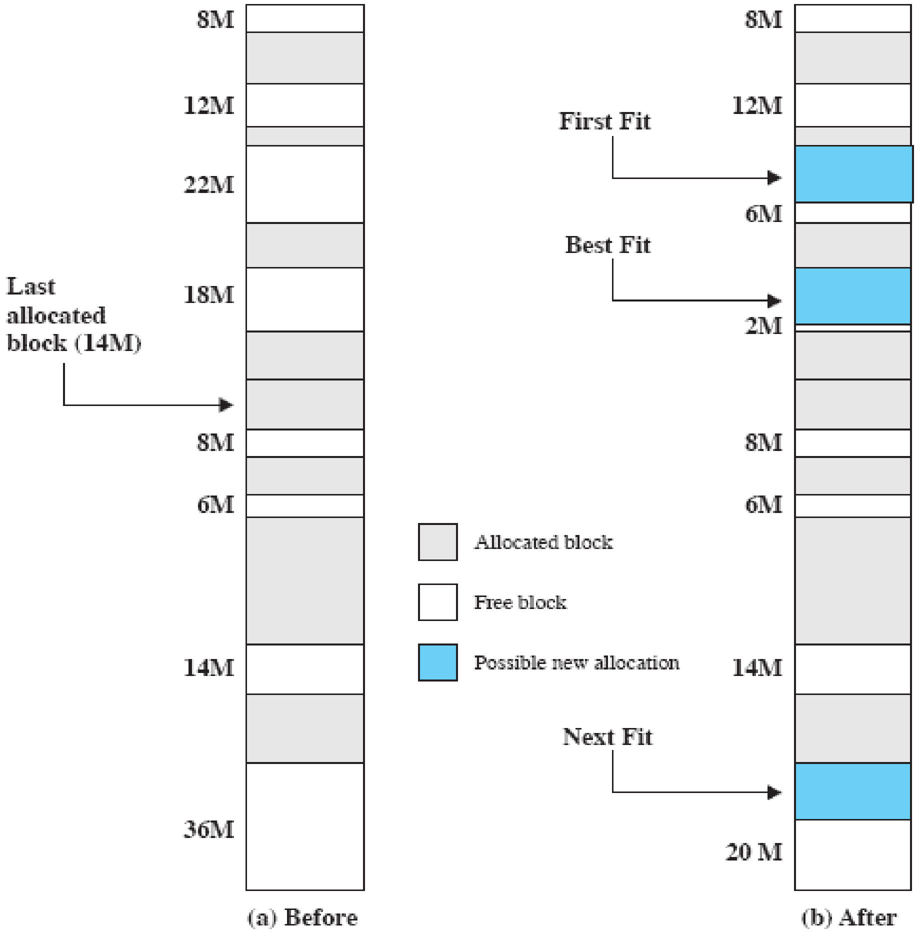
Next fit

- Like first-fit, but search the list starting from the end of previous search
- (+) Run faster than the first fit
- (-) Worse memory utilization than the first fit

Best fit

- Search the list, choose the free block with the closest size that fits
- (+) Keeps fragments small – better memory utilization than the other two
- (-) Will typically run slower – requires an exhaustive search of the heap

Placement Example



Source: Pearson

Figure 7.5 Example Memory Configuration before and after Allocation of 16-Mbyte Block

Buddy System

❑ Buddy system

- Use both fixed and dynamic partitioning
- Memory blocks are available of size 2^K *words, $L \leq K \leq U$,* where
  - 2^L = *smallest size block that is allocated*
  - 2^U = *largest size block that is allocated*
- If a request of size *s* is made, the entire block that fits s is allocated.
- The buddy system maintains a list of *holes* (unallocated blocks)
  - It may *split* a hole in half to create two buddies of half size
  - It may *coalesce* two holes into a single block of double size

Buddy System Example

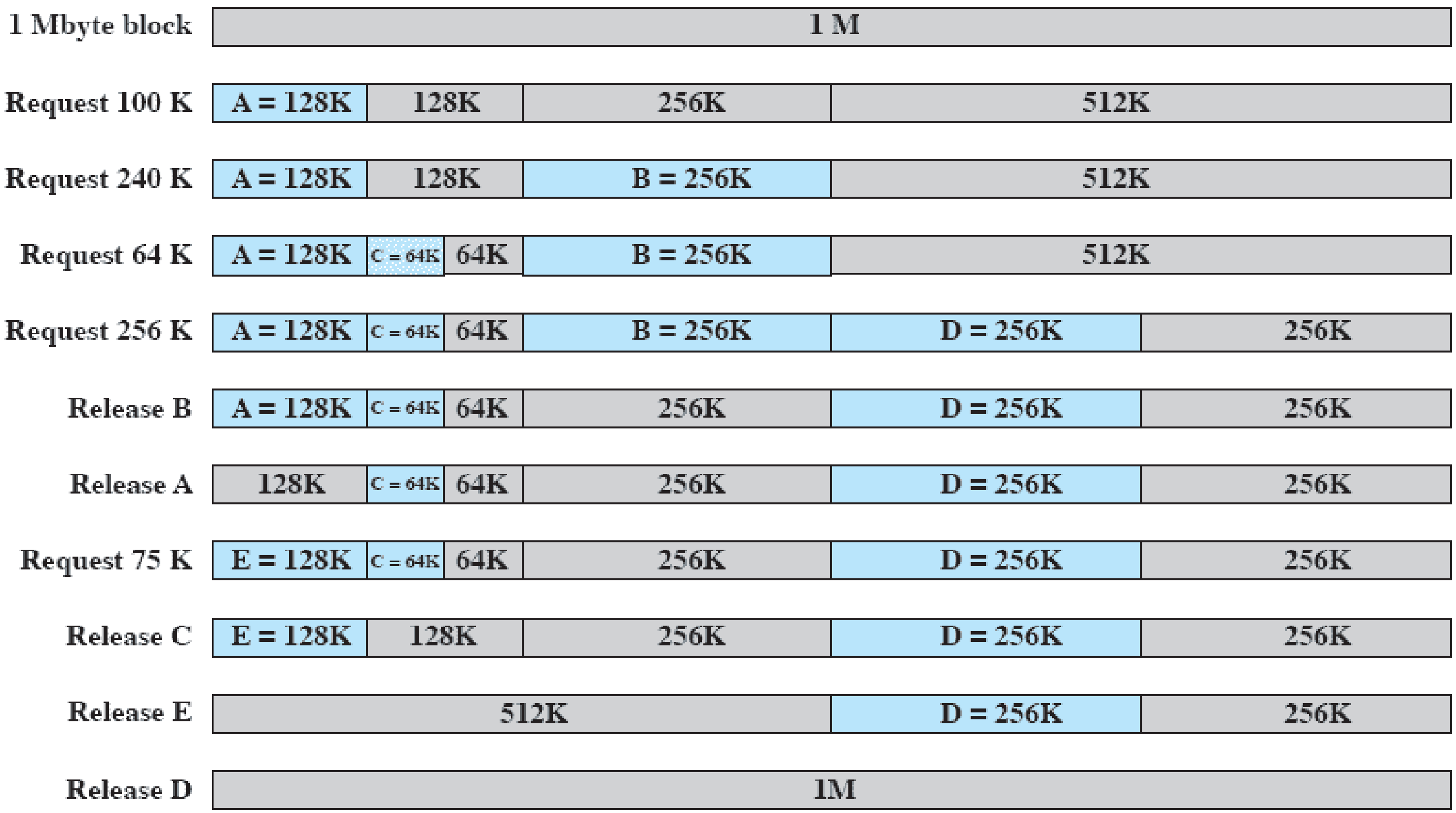
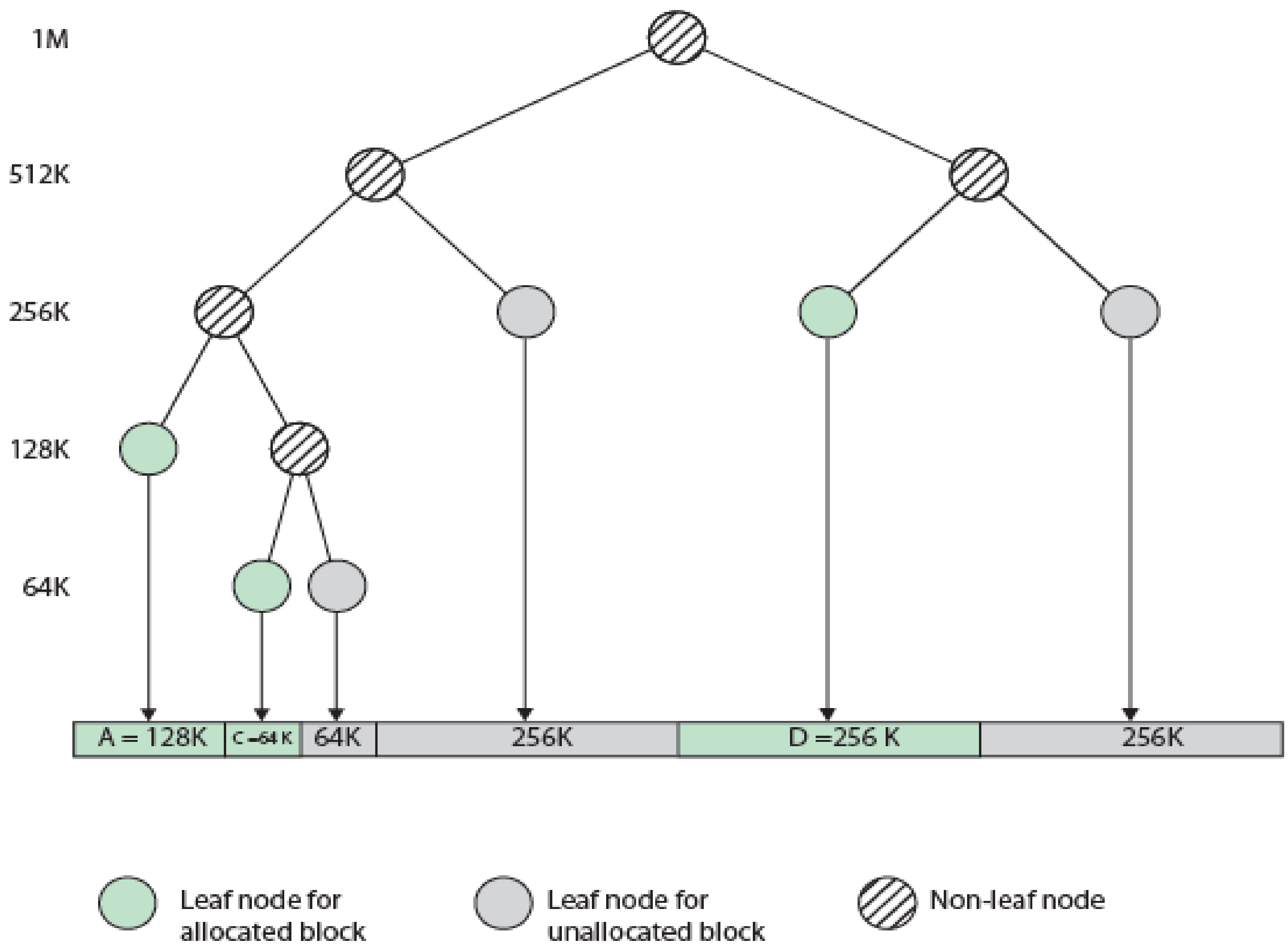


Figure 7.6 Example of Buddy System

Source: Pearson

Tree Representation



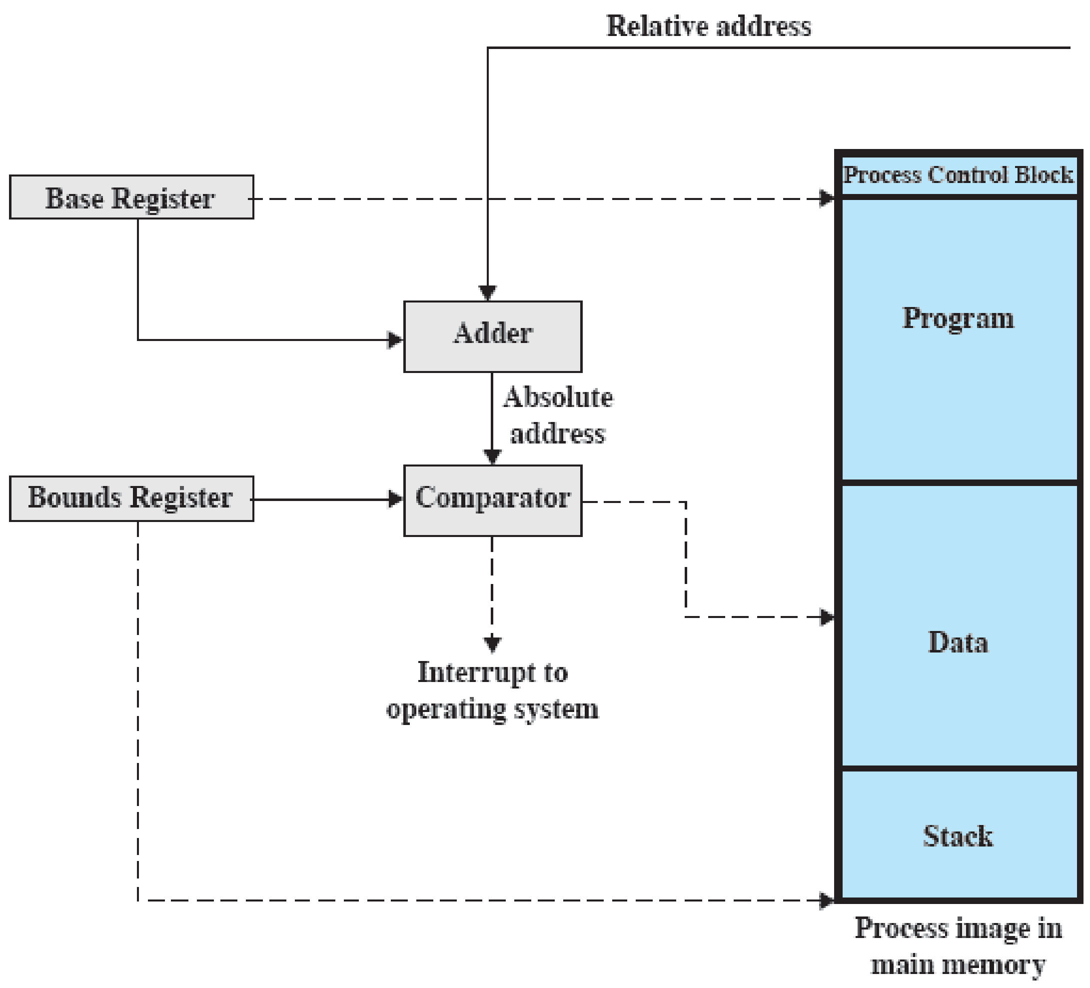
Source: Pearson

Figure 7.7 Tree Representation of Buddy System

Addresses

- **Logical address**
  - Address starts from 0
- **Relative address**
  - An example of logical address
  - Address is expressed as a location relative to some known point
- **Physical address**
  - The actual address in main memory

Hardware Support for Relocation



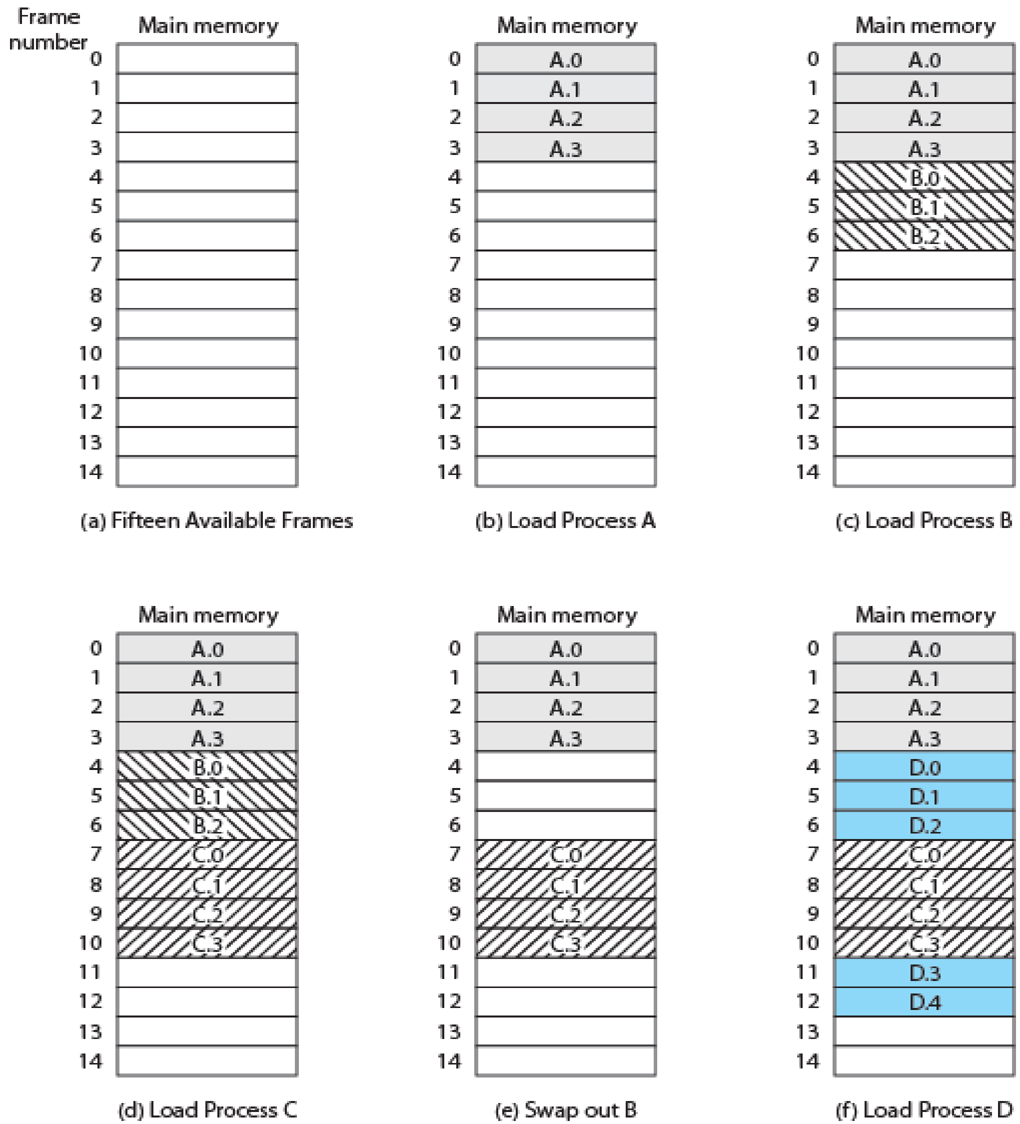
Source: Pearson

Figure 7.8 Hardware Support for Relocation

Paging

- **Paging**
  - Partition memory into equal fixed-size chunks (*page frames*)
  - Process image is divided into the same fixed-size chunks (*pages*)
- **Page table**
  - Contains the mapping between pages and frames
    - For each page in the process, *PTE* (*page table entry*) contains the frame number
  - Maintained by operating system for each process
  - CPU must access the page table to generate a physical address for the current process

Assignment of Processes to Frames



Source: Pearson

Figure 7.9 Assignment of Process Pages to Free Frames

Page Tables

Process A page table

0	0
1	1
2	2
3	3

Process B page table

0	–
1	–
2	–

Process C page table

0	7
1	8
2	9
3	10

Process D page table

0	4
1	5
2	6
3	11
4	12

Free frame list

13
14

Figure 7.10 Data Structures for the Example of Figure 7.9 at Time Epoch (f)

Source: Pearson

Logical Address

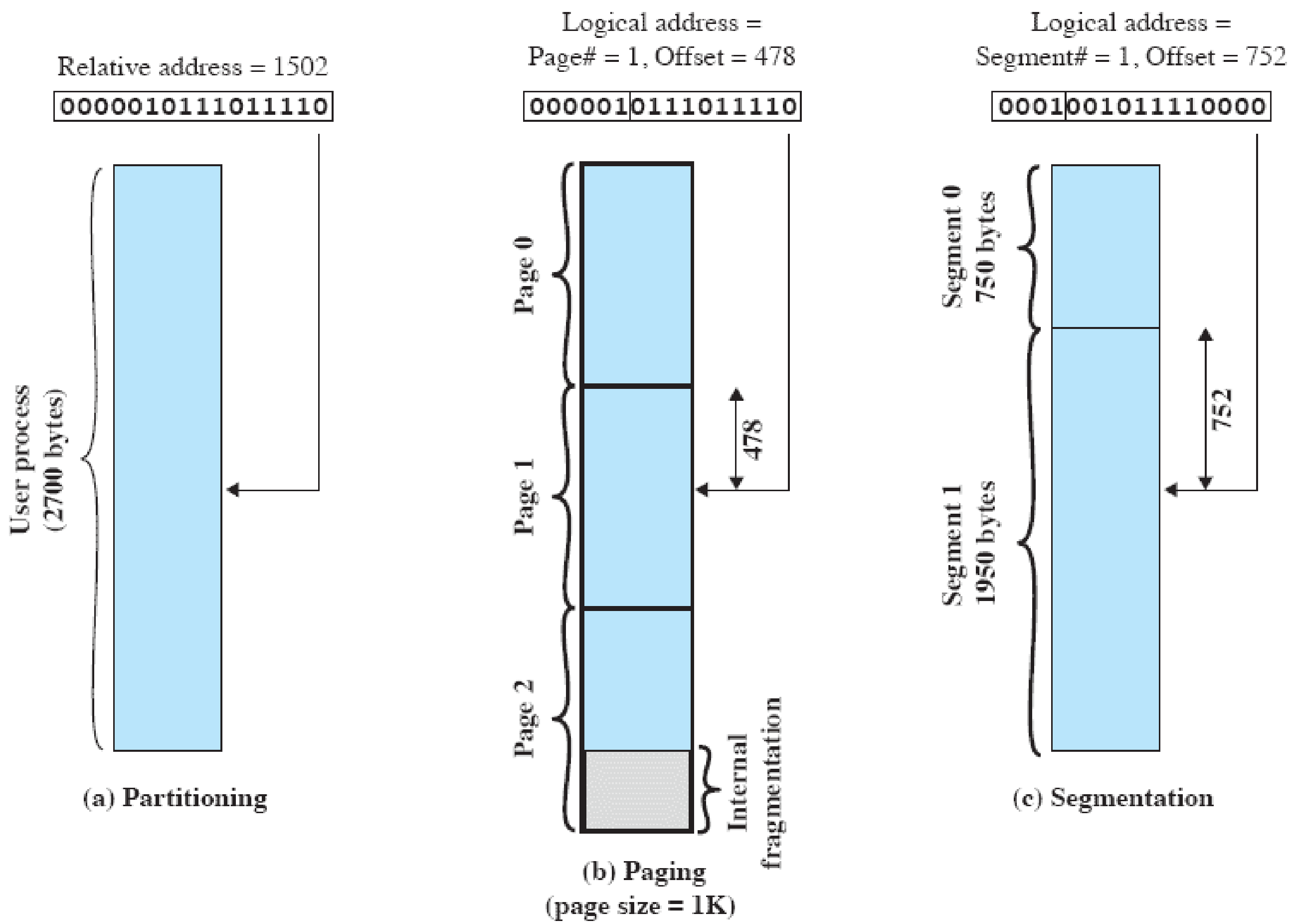
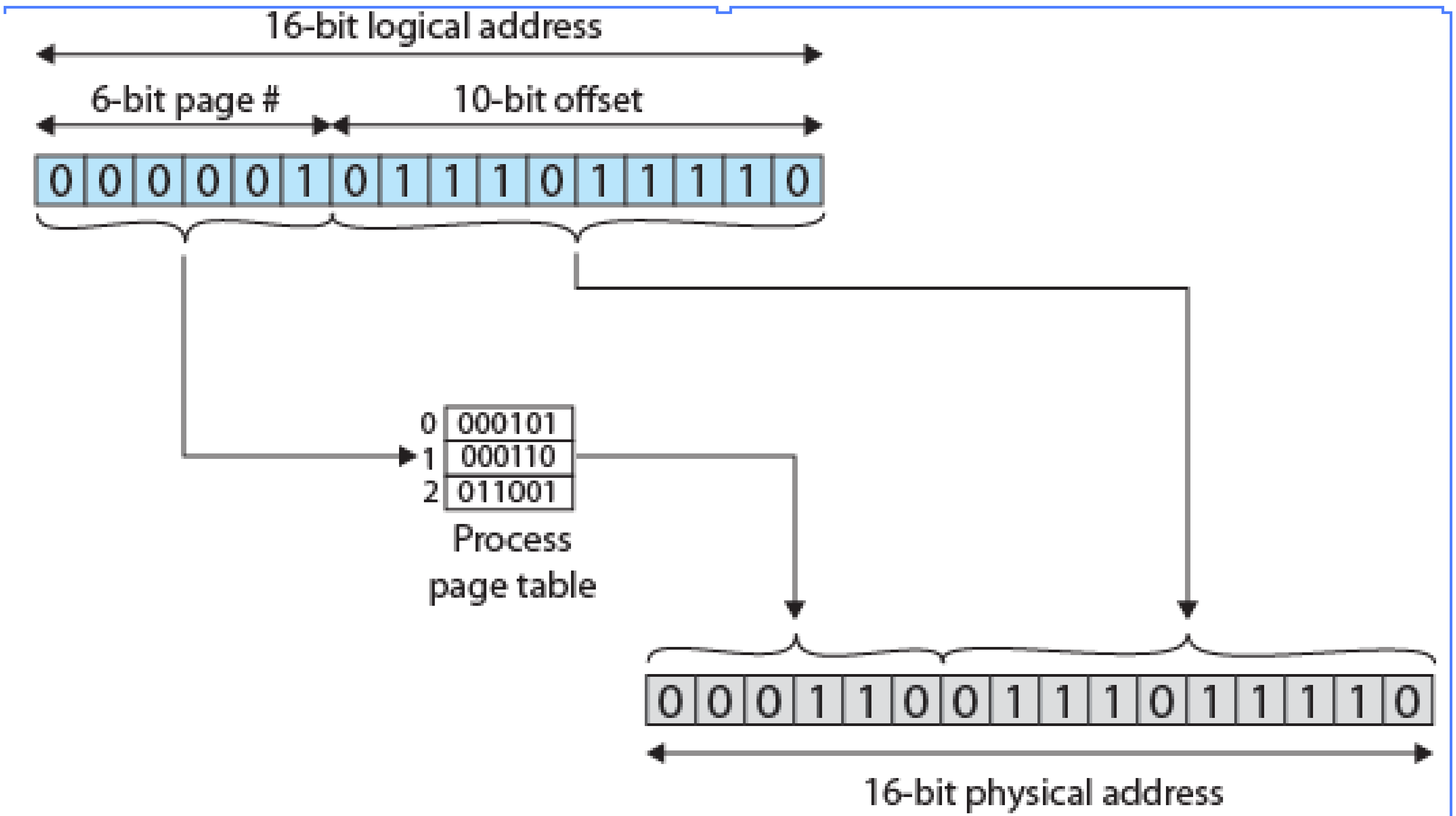


Figure 7.11 Logical Addresses

Source: Pearson

Logical to Physical Address Translation



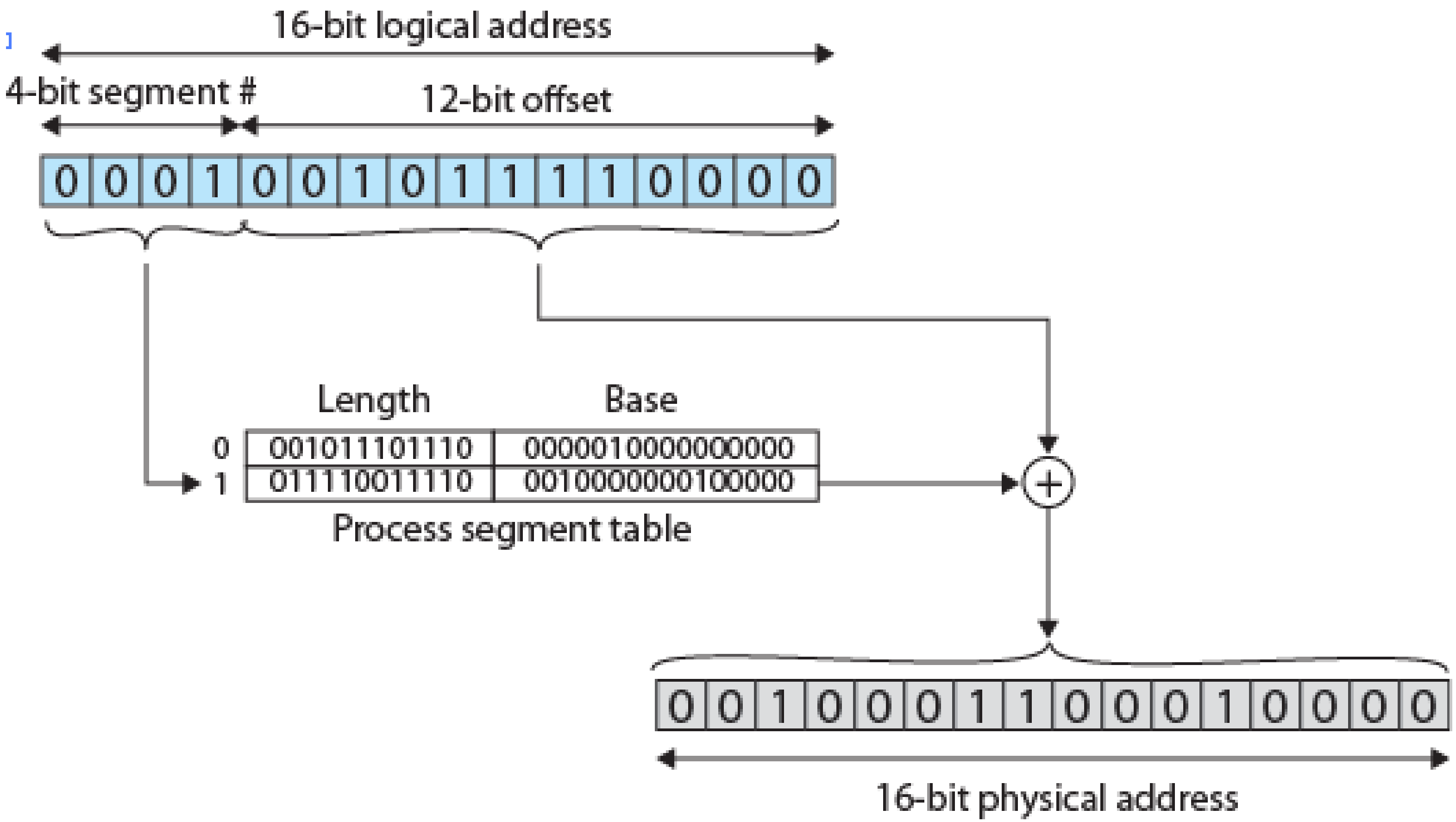
(a) Paging

Source: Pearson

Segmentation

- **Segmentation**
  - A program is divided into variable-length *segments*
  - The address consists of segment number + offset
  - No internal fragmentation
  - But, external fragmentation
    - Similar to dynamic partitioning

Logical to Physical Address Translation



(b) Segmentation

Source: Pearson

Homework 6

- Exercise 7.2
- Exercise 7.6
- Exercise 7.9
- Exercise 7.15