Operating Systems: Internals and Design Principles

Chapter 7 Memory Management

Ninth Edition

William Stallings

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).

Table 7.1

Memory Management Terms

Memory Management Requirements

- Memory management is intended to satisfy the following requirements:
  - Relocation
  - Protection
  - Sharing
  - Logical organization
  - Physical organization

Relocation

- Programmers typically do not know in advance which other programs will be resident in main memory at the time of execution of their program
- Active processes need to be able to be swapped in and out of main memory in order to maximize processor utilization
- Specifying that a process must be placed in the same memory region when it is swapped back in would be limiting
  - May need to *relocate* the process to a different area of memory

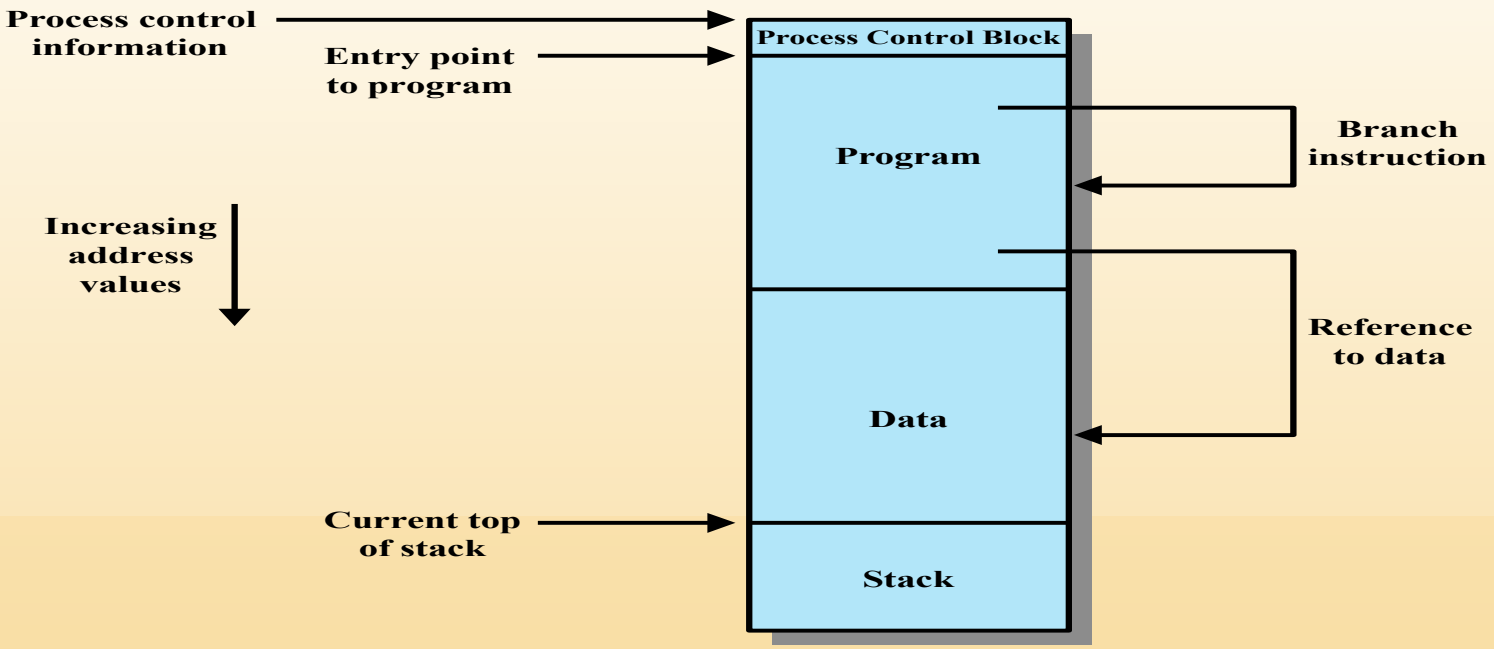


Figure 7.1 Addressing Requirements for a Process

- Thus, we cannot know ahead of time where a program will be placed, and we must allow for the possibility that the program may be moved about in main memory due to swapping.
- The processor hardware and operating system software must be able to translate the memory references found in the code of the program into actual physical memory addresses, reflecting the current location of the program in main memory.

Protection

- Processes need to acquire permission to reference memory locations for reading or writing purposes
- Location of a program in main memory is unpredictable
- Memory references generated by a process must be checked at run time
- Mechanisms that support relocation also support protection

Sharing

- Advantageous to allow each process access to the same copy of the program rather than have their own separate copy
- Memory management must allow controlled access to shared areas of memory without compromising protection
- Mechanisms used to support relocation support sharing capabilities

Logical Organization

- Memory is organized as linear

Programs are written in modules

- Modules can be written and compiled independently
- Different degrees of protection given to modules (read-only, execute-only)
- Sharing on a module level corresponds to the user's way of viewing the problem

- Segmentation is the tool that most readily satisfies requirements

Physical Organization

Cannot leave the programmer with the responsibility to manage memory

Memory available for a program plus its data may be insufficient

Overlaying allows various modules to be assigned the same region of memory but is time consuming to program

Programmer does not know how much space will be available

- As we discussed in Section 1.5 , computer memory is organized into at least two levels, referred to as main memory and secondary memory.
- In this two-level scheme, the organization of the flow of information between main and secondary memory is a major system concern. The responsibility for this flow could be assigned to the individual programmer, but this is impractical and undesirable

Memory Partitioning

- Memory management brings processes into main memory for execution by the processor
  - Involves virtual memory (VM)
  - VM is based on segmentation and paging
- Partitioning (simple)
  - Used in several variations in some now-obsolete operating systems
  - Does not involve virtual memory
- We will first look at and understand simple techniques before studying the sophisticated VM scheme

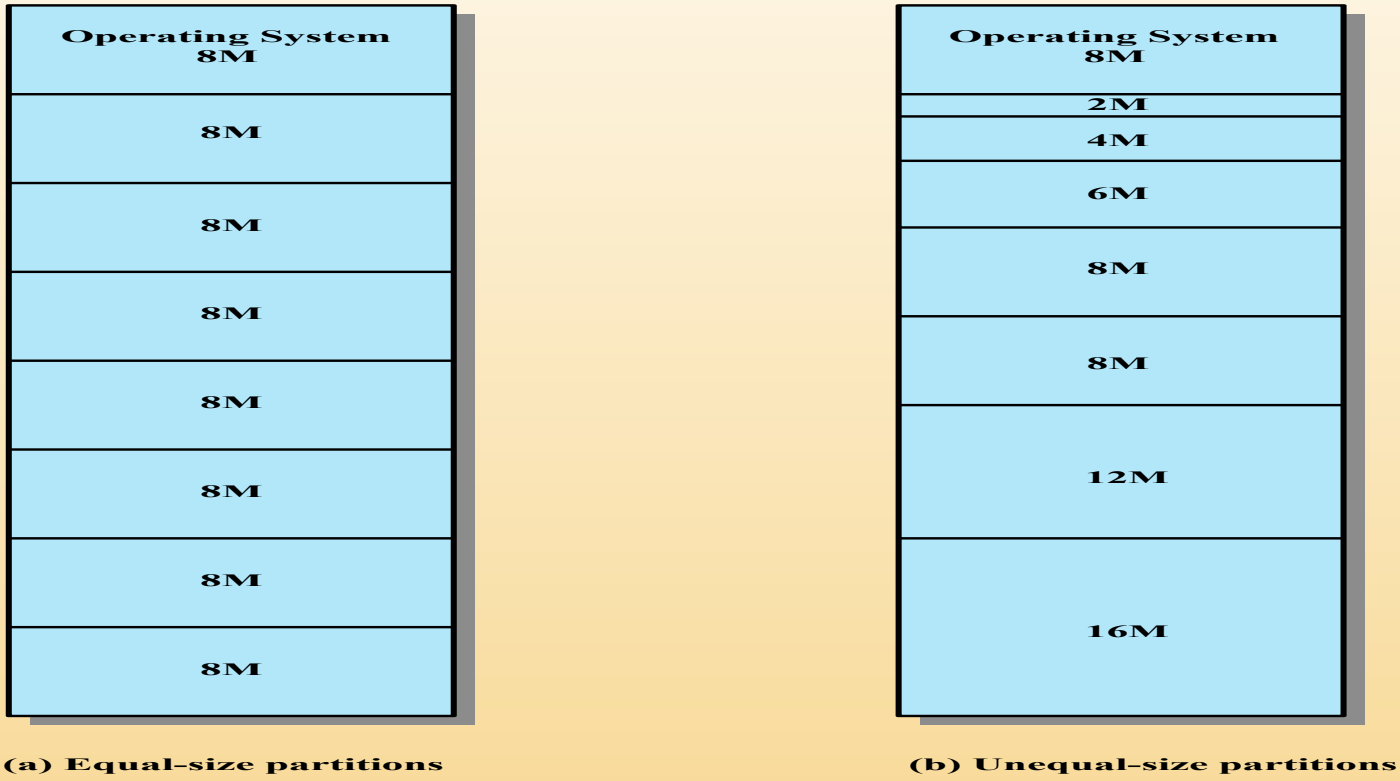
Technique	Description	Strengths	Weaknesses
Fixed Partitioning	Main memory is divided into a number of static partitions at system generation time. A process may be loaded into a partition of equal or greater size.	Simple to implement; little operating system overhead.	Inefficient use of memory due to 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.

Table 7.2

Memory Management Techniques

(Table is on page 317 in textbook)

Fixed partitioning



(a) Equal-size partitions

(b) Unequal-size partitions

Figure 7.2 Example of Fixed Partitioning of a 64-Mbyte Memory

we can assume that the OS occupies some fixed portion of main memory and that the rest of main memory is available for use by multiple processes. The simplest scheme for managing this available memory is to partition it into regions with fixed boundaries.

Disadvantages

- A program may be too big to fit in a partition
  - Program needs to be designed with the use of overlays
- Main memory utilization is inefficient
  - Any program, regardless of size, occupies an entire partition
  - ***Internal fragmentation***
    - Wasted space due to the block of data loaded being smaller than the partition

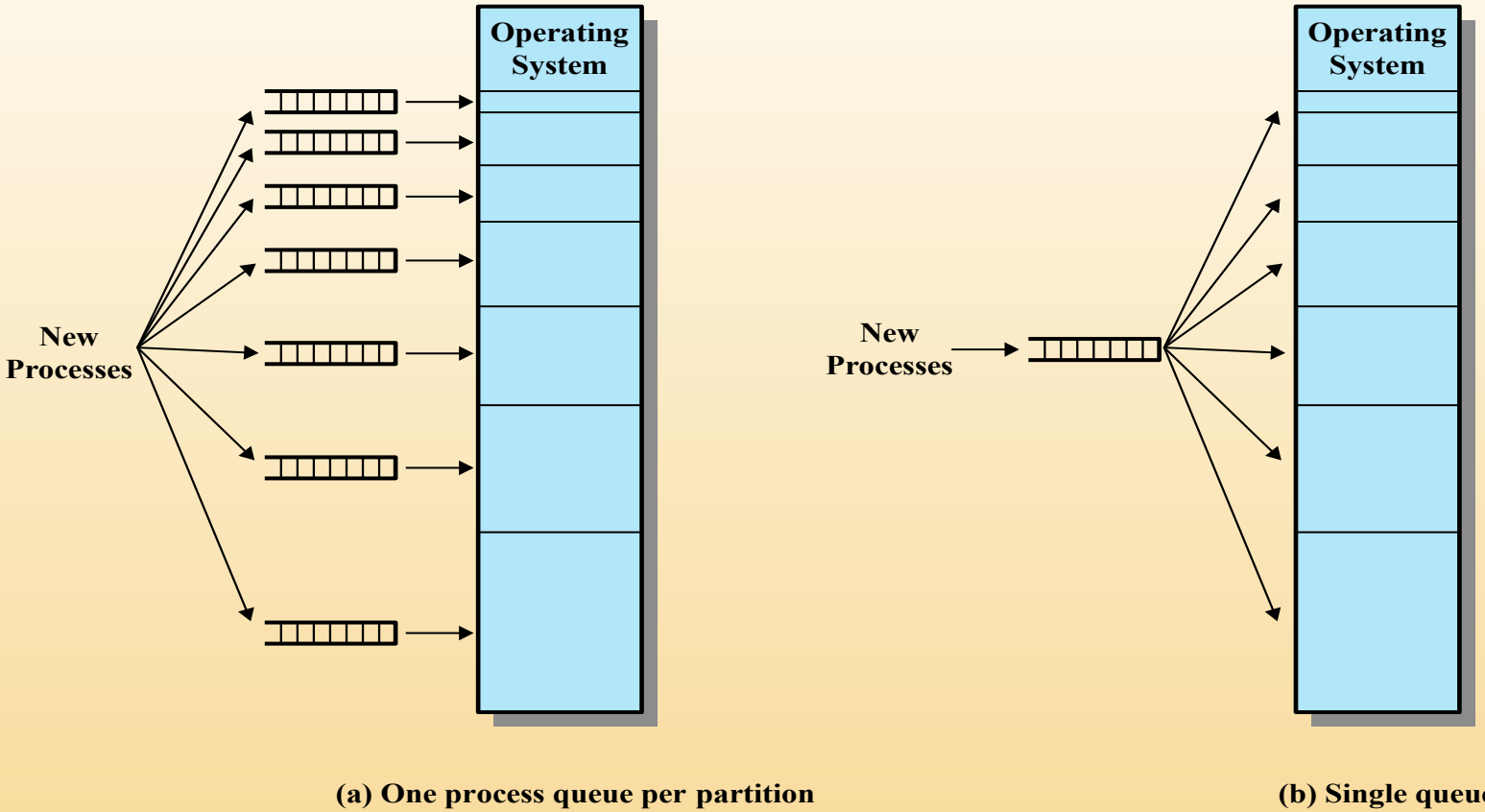


Figure 7.3 Memory Assignment for Fixed Partitioning

- With unequal-size partitions, there are two possible ways to assign processes to partitions. The simplest way is to assign each process to the smallest partition within which it will fit. In this case, a scheduling queue is needed for each partition, to hold swapped-out processes destined for that partition (Figure 7.3a).
- The advantage of this approach is that processes are always assigned in such a way as to minimize wasted memory within a partition (internal fragmentation).

Disadvantages

- The number of partitions specified at system generation time limits the number of active processes in the system
- Small jobs will not utilize partition space efficiently

Dynamic Partitioning

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

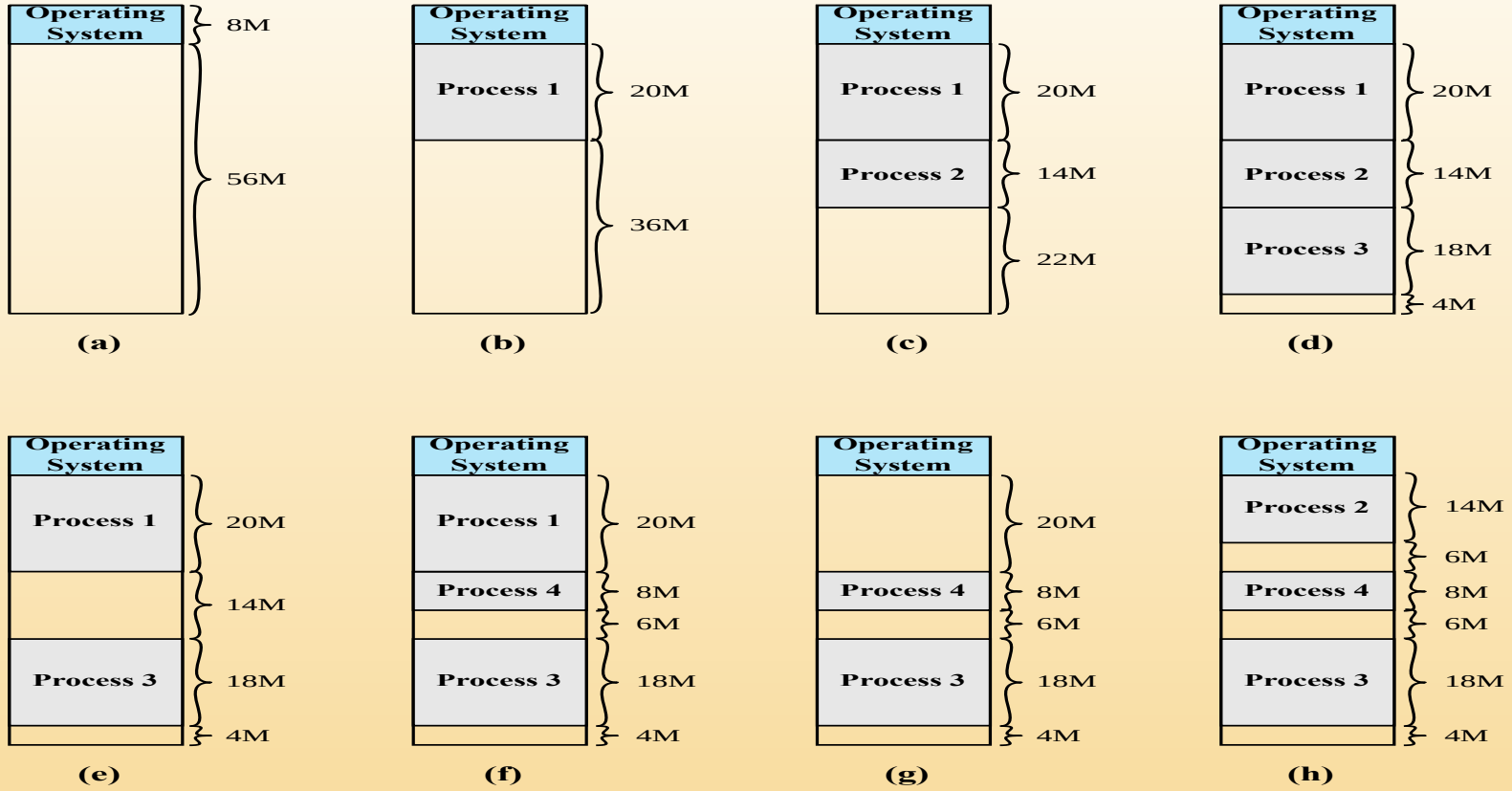


Figure 7.4 The Effect of Dynamic Partitioning

As this example shows, this method starts out well, but eventually it leads to a situation in which there are a lot of small holes in memory. As time goes on, memory becomes more and more fragmented, and memory utilization declines. This phenomenon is referred to as **external fragmentation** , indicating that the memory that is external to all partitions becomes increasingly fragmented. This is in contrast to internal fragmentation, referred to earlier.

Dynamic Partitioning

External Fragmentation

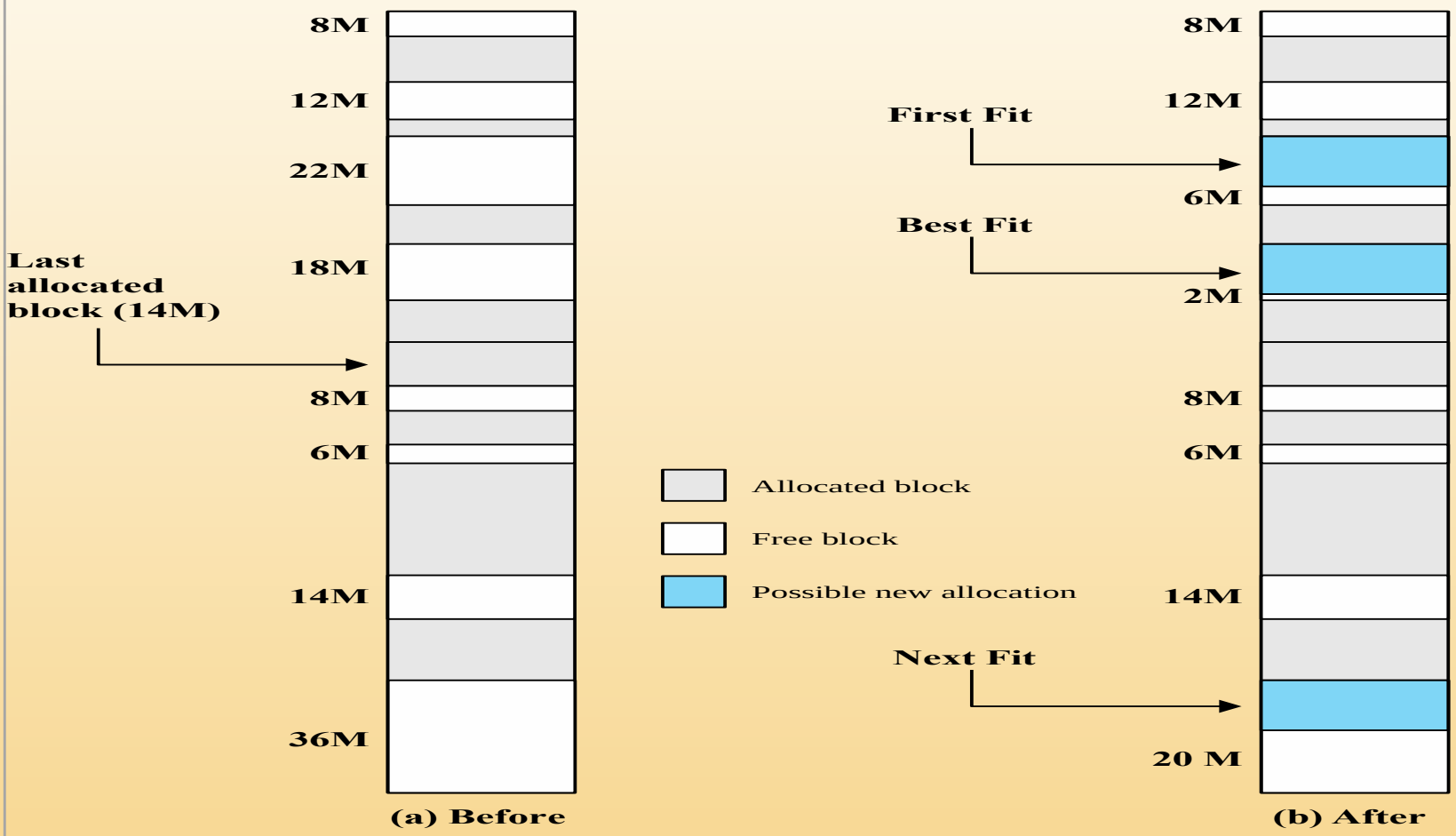
- Memory becomes more and more fragmented
- Memory utilization declines

Compaction

- Technique for overcoming external fragmentation
- OS shifts processes so that they are contiguous
- Free memory is together in one block
- Time consuming and wastes CPU time

Placement Algorithms

Best-fit	First-fit	Next-fit
• Chooses the block that is closest in size to the request	• Begins to scan memory from the beginning and chooses the first available block that is large enough	• Begins to scan memory from the location of the last placement and chooses the next available block that is large enough



Allocate a 16-Mbyte for a process

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

- Which of these approaches is best will depend on the exact sequence of process swapping that occurs and the size of those processes. However, some general comments can be made (see also [BREN89], [SHOR75], and [BAYS77]).

- The first-fit algorithm is not only the simplest but usually the best and fastest as well.

Buddy System

- Comprised of fixed and dynamic partitioning schemes
- Space available for allocation is treated as a single block
- 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; generally 2^U is the size of the entire memory available for allocation

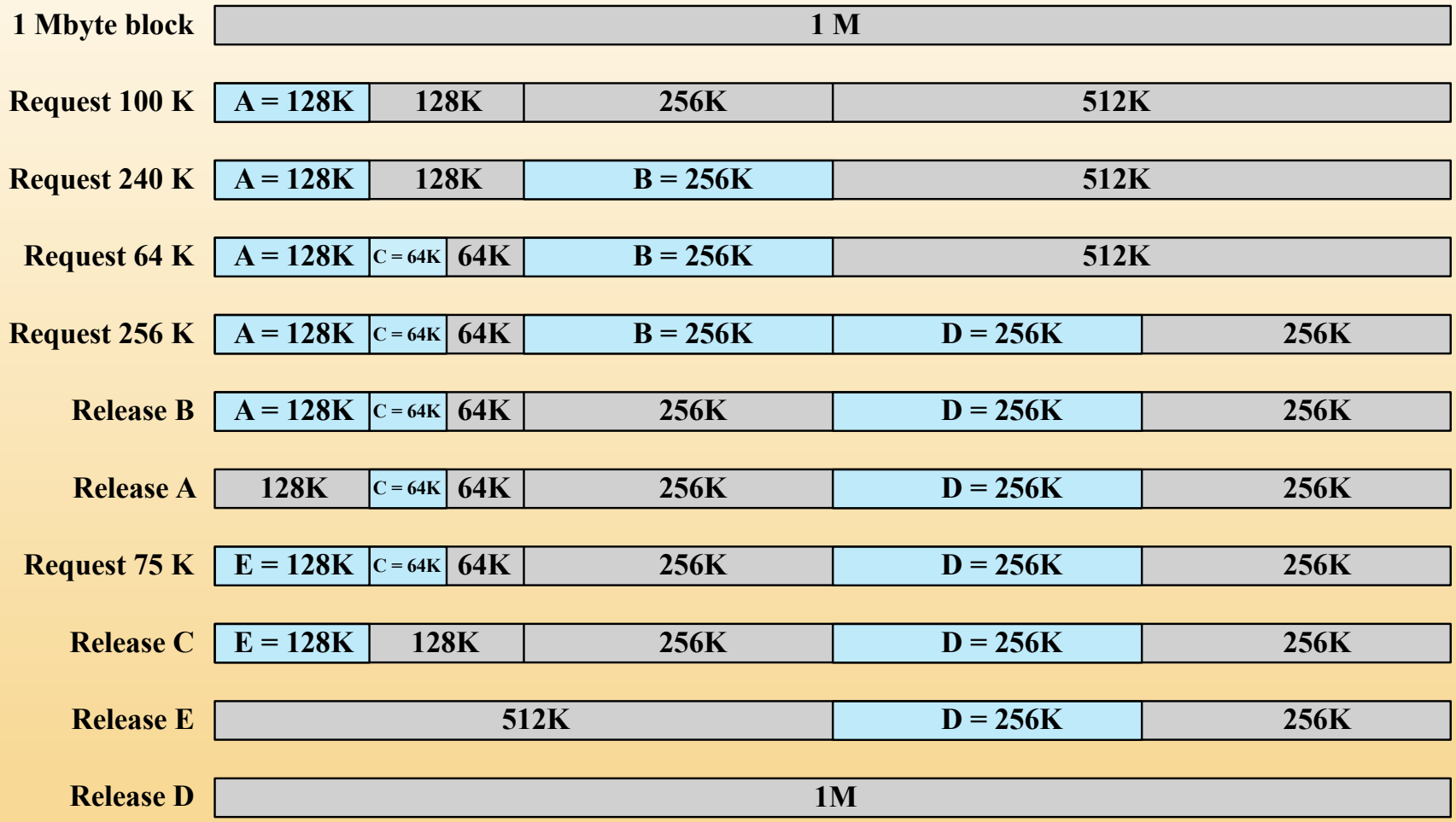


Figure 7.6 Example of Buddy System

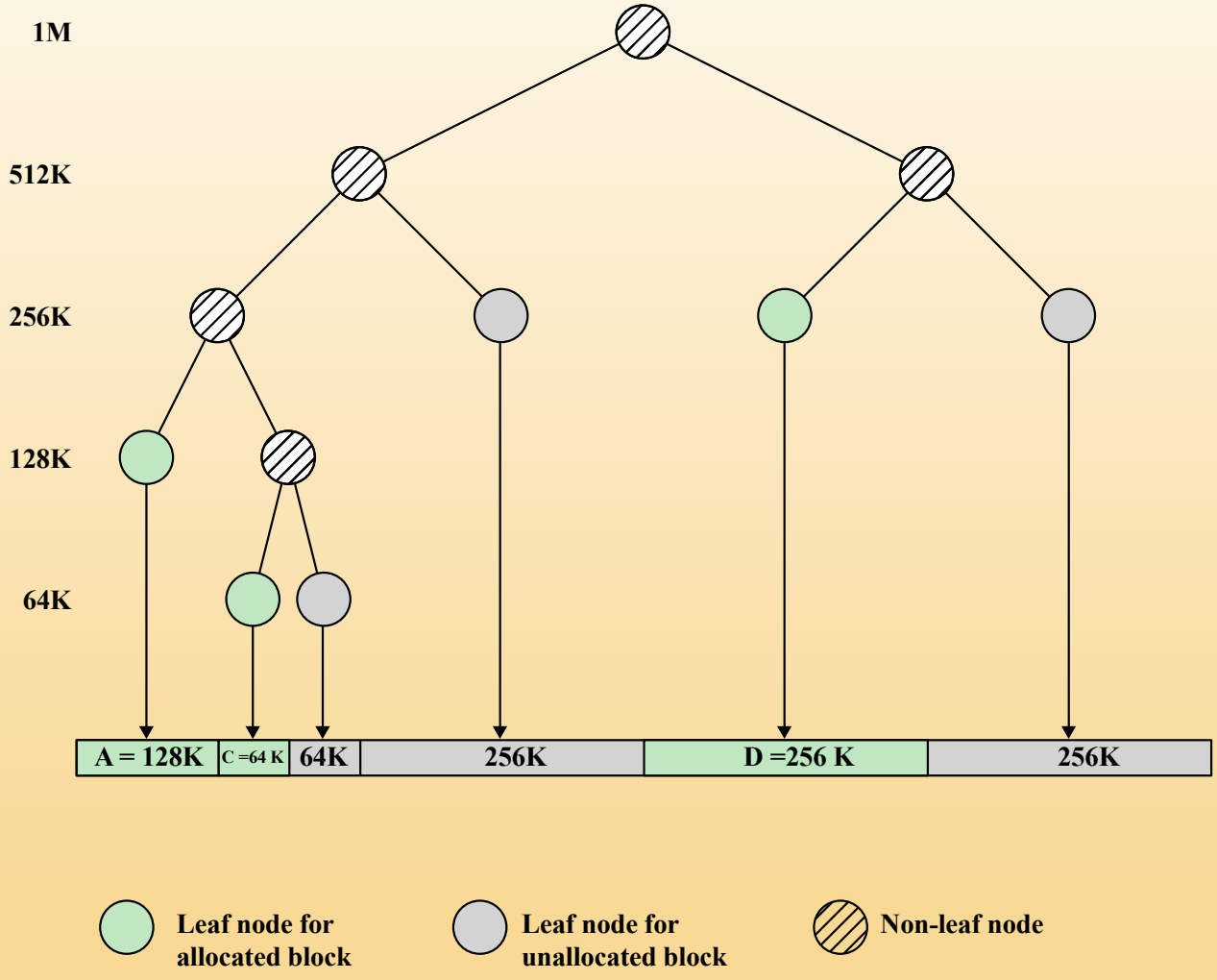


Figure 7.7 Tree Representation of Buddy System

Relocation

- When the fixed partition scheme is used, we can expect a process will always be assigned to the same partition
  - Whichever partition is selected when a new process is loaded will always be used to swap that process back into memory after it has been swapped out
  - In that case, a simple relocating loader can be used
  - When the process is first loaded, all relative memory references in the code are replaced by absolute main memory addresses, determined by the base address of the loaded process
- In the case of equal-size partitions and in the case of a single process queue for unequal-size partitions, a process may occupy different partitions during the course of its life
  - When a process image is first created, it is loaded into some partition in main memory; Later, the process may be swapped out
  - When it is subsequently swapped back in, it may be assigned to a different partition than the last time
  - The same is true for dynamic partitioning
- When compaction is used, processes are shifted while they are in main memory
- **Thus, the locations referenced by a process are not fixed, they will change each time a process is swapped in or shifted**

Addresses

Logical

- Reference to a memory location independent of the current assignment of data to memory

Relative

- A particular example of logical address, in which the address is expressed as a location relative to some known point

Physical or Absolute

- Actual location in main memory

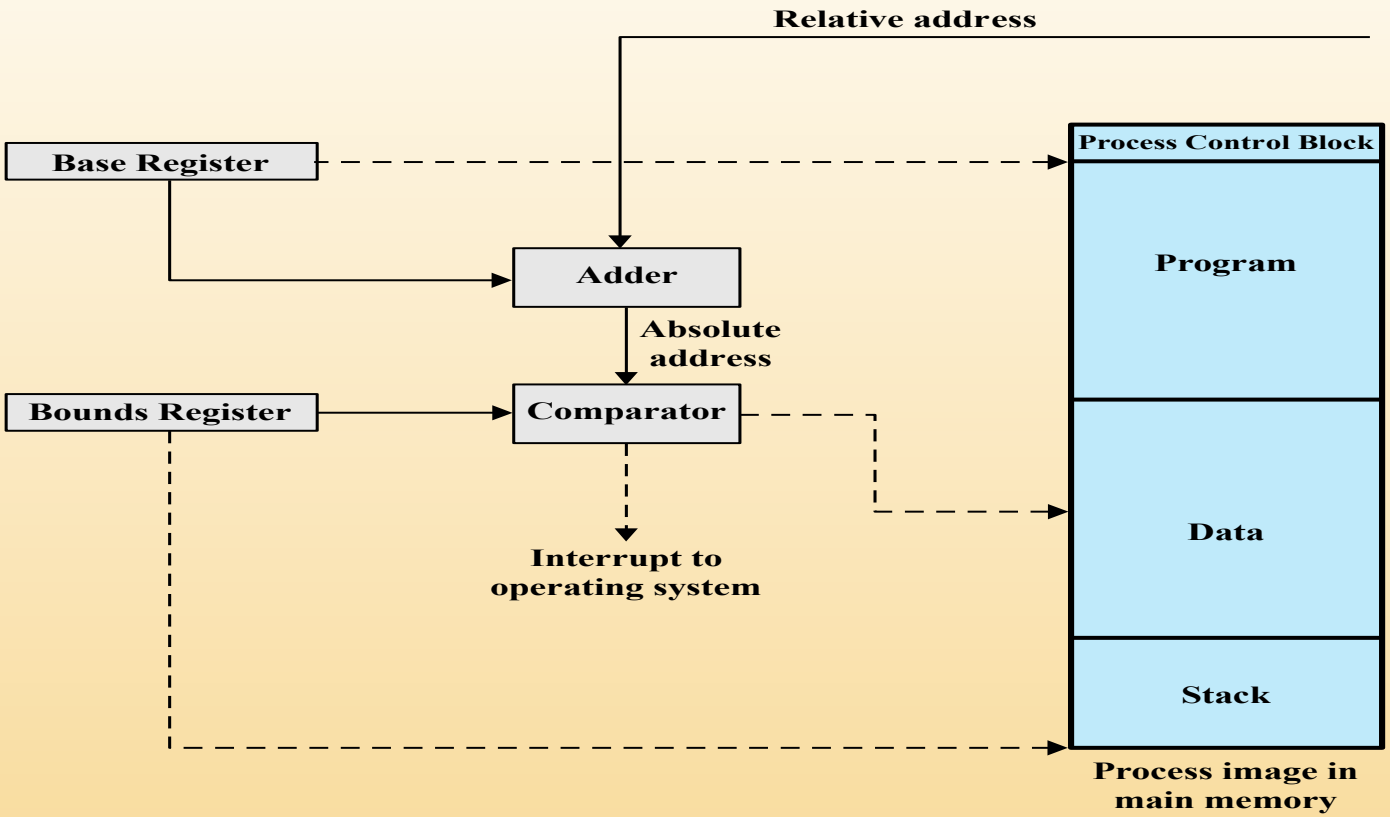


Figure 7.8 Hardware Support for Relocation

Programs that employ relative addresses in memory are loaded using dynamic run-time loading (see Appendix 7A for a discussion). Typically, all of the memory references in the loaded process are relative to the origin of the program. Thus a hardware mechanism is needed for translating relative addresses to physical main memory addresses at the time of execution of the instruction that contains the reference.

Paging

- Partition memory into equal fixed-size chunks that are relatively small
- Process is also divided into small fixed-size chunks of the same size

Pages	Frames
• Chunks of a process	• Available chunks of memory

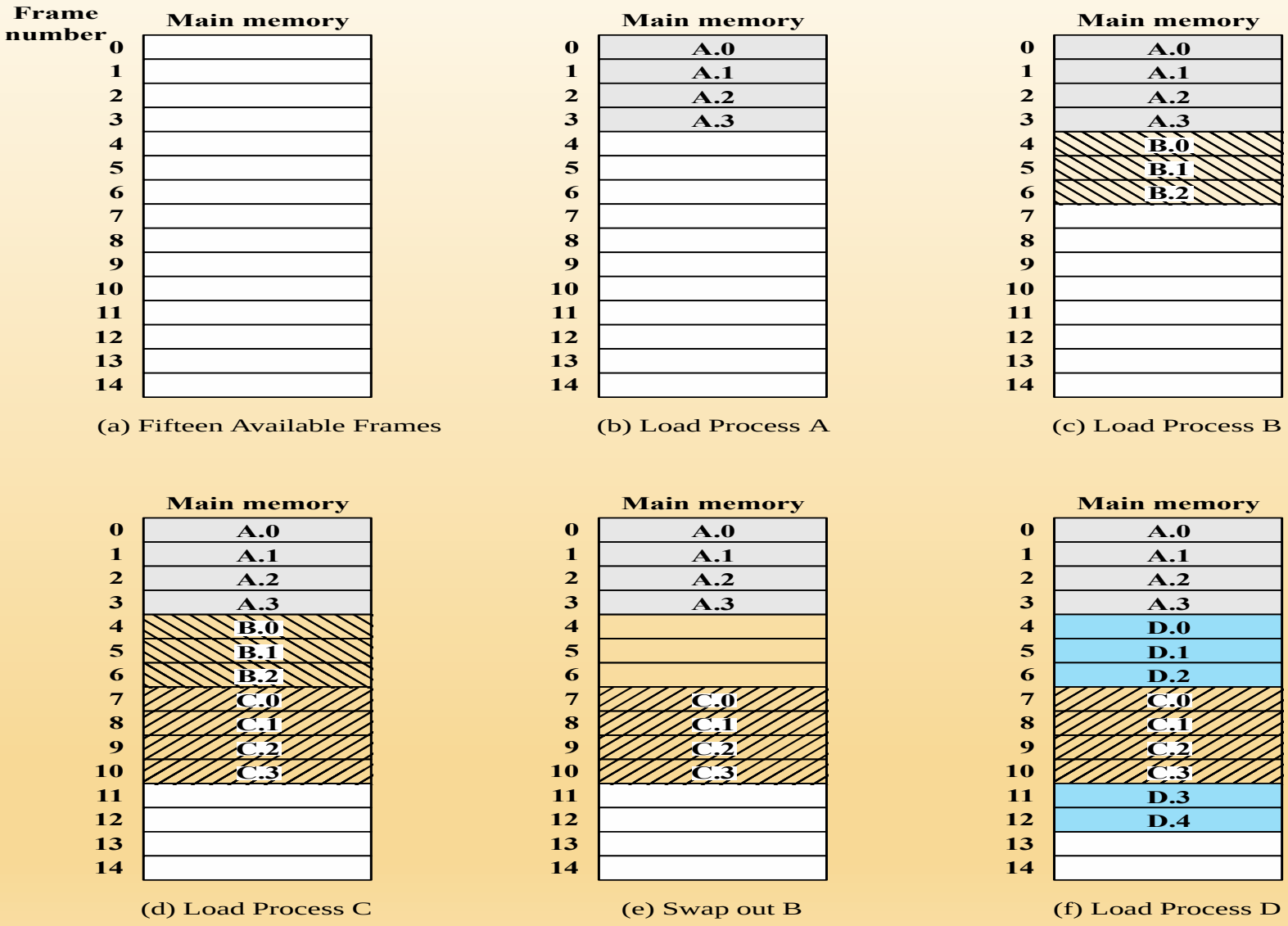


Figure 7.9 Assignment of Process Pages to Free Frames

Page Table

- Maintained by operating system for each process
- Contains the frame location for each page in the process
- Processor must know how to access for the current process
- Used by processor to produce a physical address

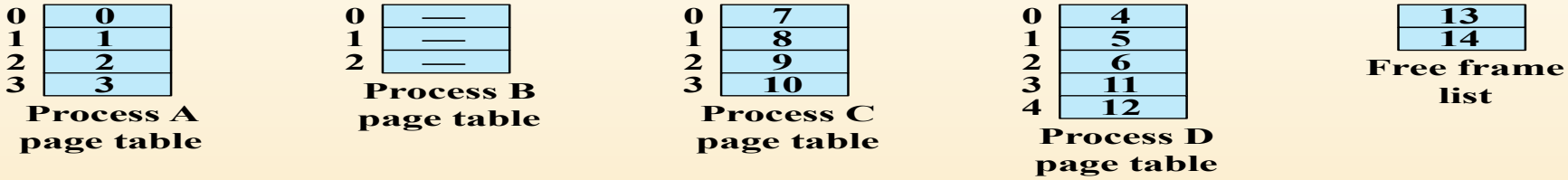


Figure 7.10 Data Structures for the Example of Figure 7.9 at Time Epoch (f)

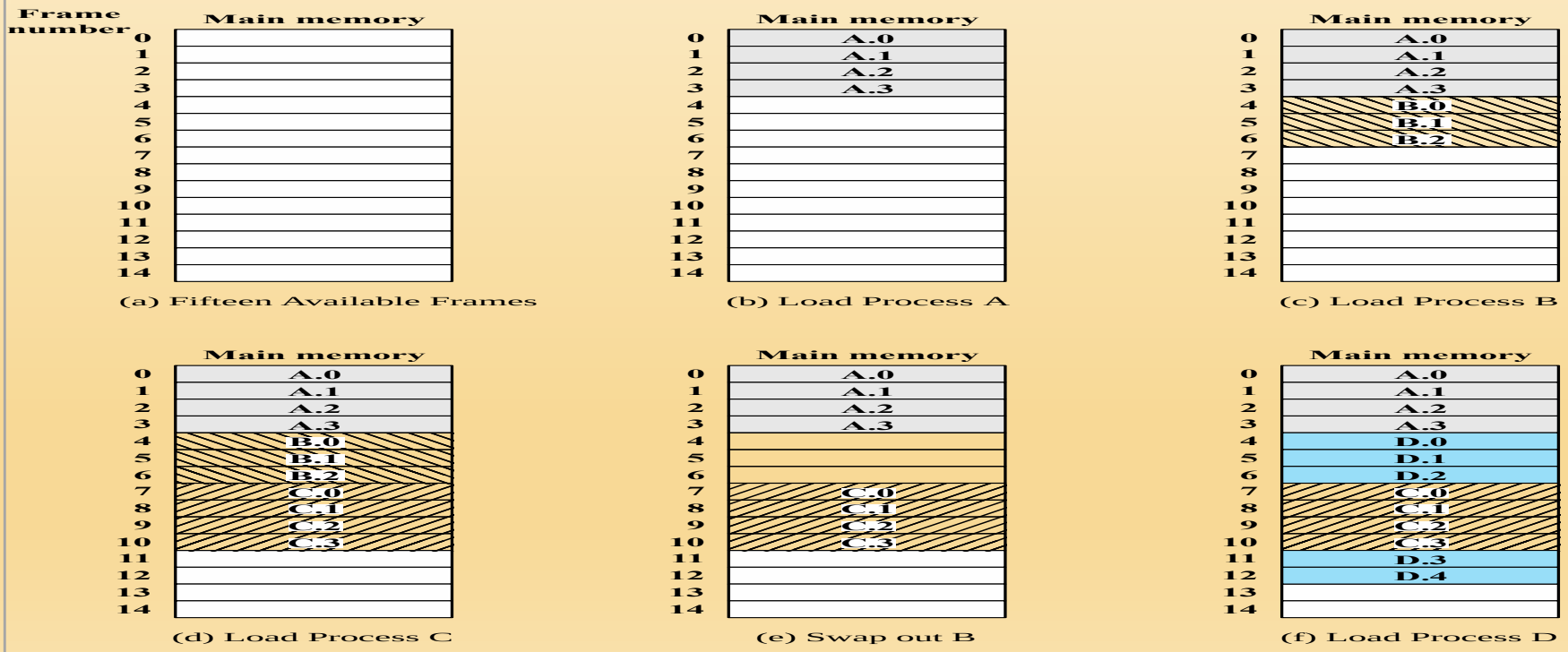


Figure 7.9 Assignment of Process Pages to Free Frames

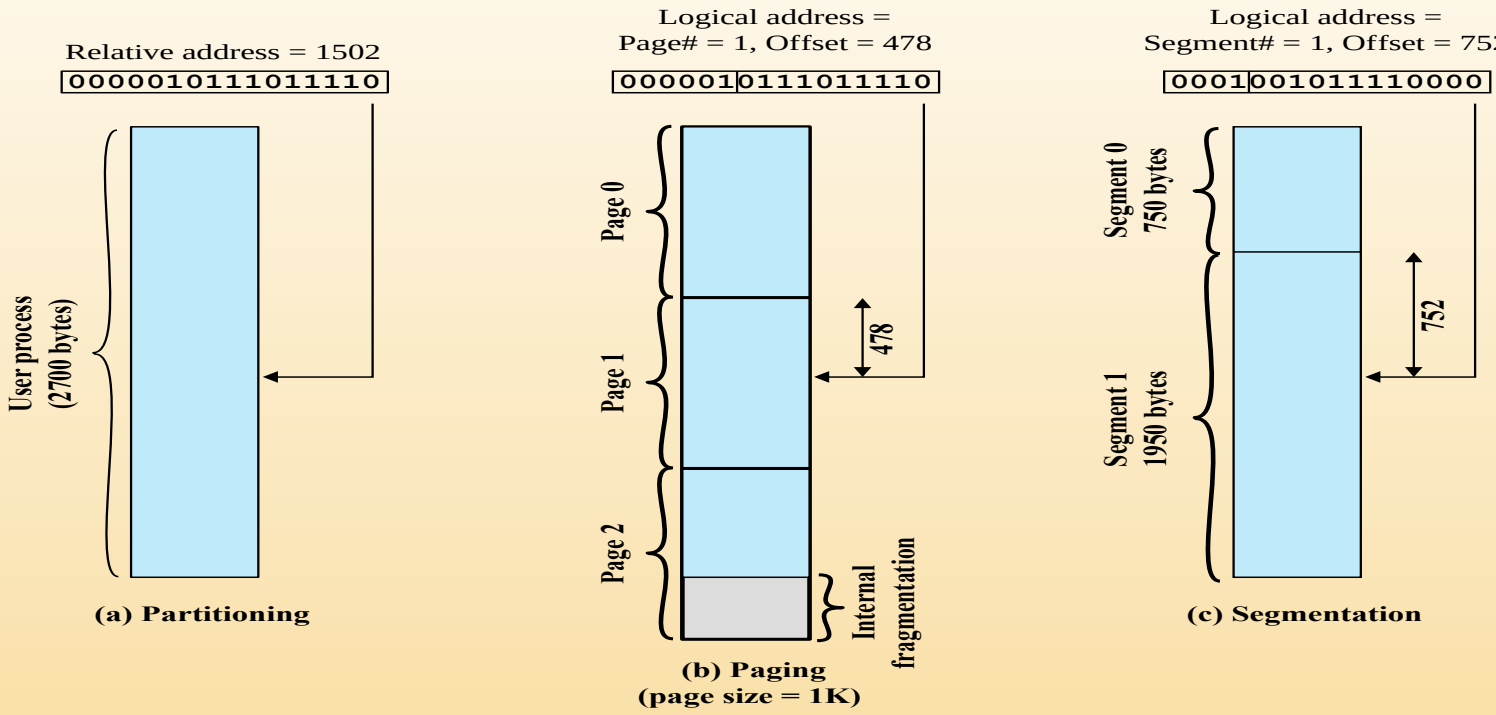


Figure 7.11 Logical Addresses

- The logical address, expressed as a page number and offset.
- An example is shown in Figure 7.11 . In this example, 16-bit addresses are used, and the page size is 1K 1,024 bytes.
- The relative address 1502, in binary form, is 0000010111011110.
- With a page size of 1K, an offset field of 10 bits is needed, leaving 6 bits for the page number.
- Thus a program can consist of a maximum of 26 = 64 pages of 1K bytes each.
- As Figure 7.11b shows, relative address 1502 corresponds to an offset of 478 (0111011110) on page 1 (000001), which yields the same 16-bit number, 0000010111011110.

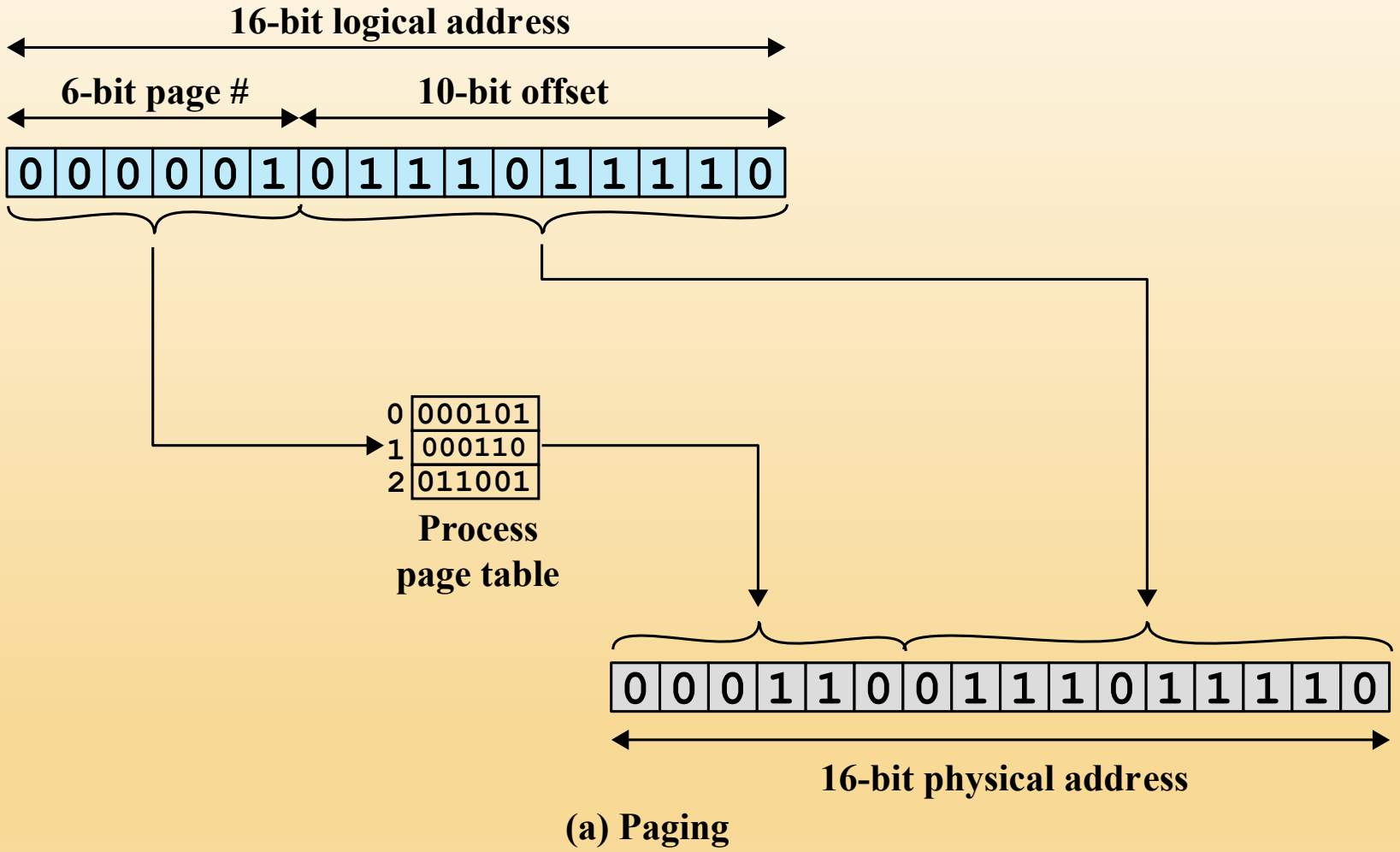


Figure 7.12 Examples of Logical-to-Physical Address Translation

Segmentation

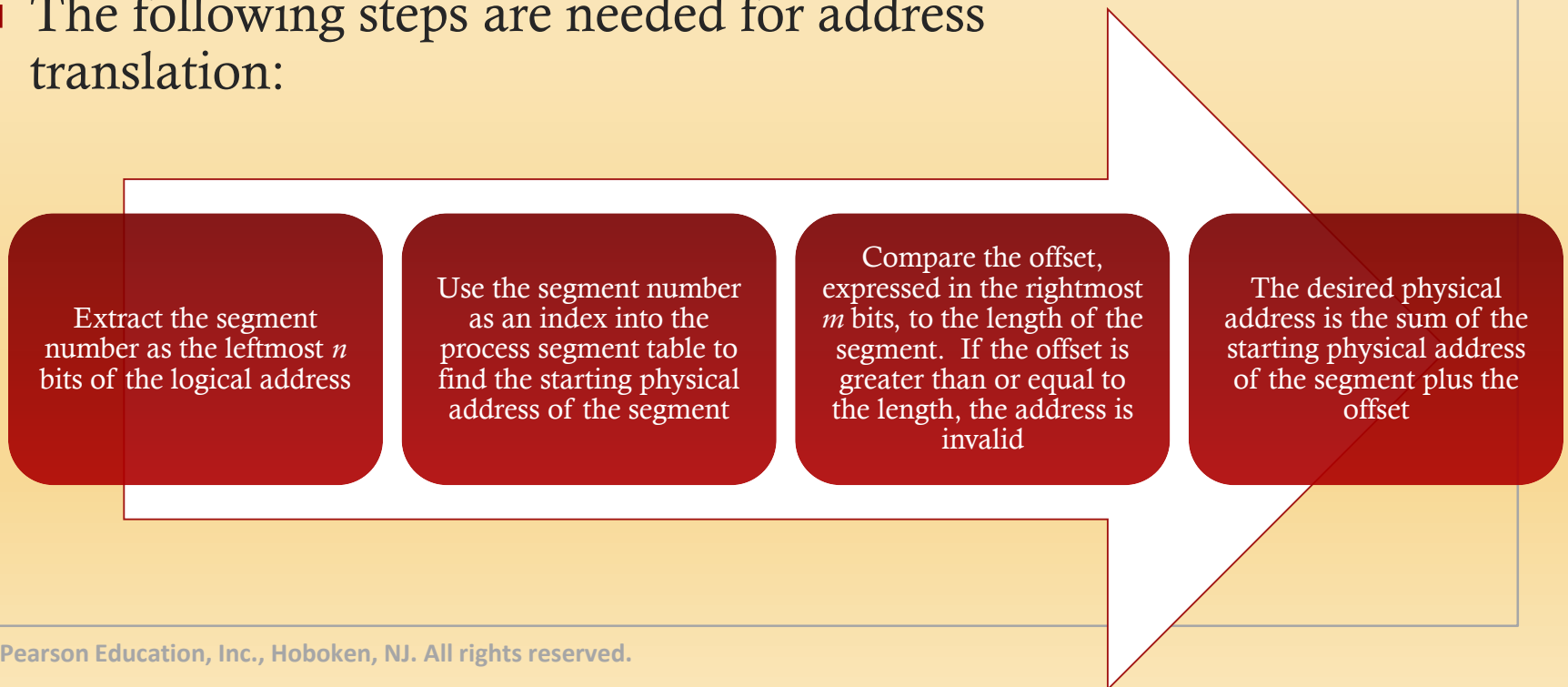
- A program can be subdivided into segments
  - May vary in length
  - There is a maximum length
- Addressing consists of two parts:
  - Segment number
  - An offset
- Similar to dynamic partitioning
- Eliminates internal fragmentation

Segmentation

- Usually visible
- Provided as a convenience for organizing programs and data
- Typically the programmer will assign programs and data to different segments
- For purposes of modular programming the program or data may be further broken down into multiple segments
  - The principal inconvenience of this service is that the programmer must be aware of the maximum segment size limitation

Address Translation

- Another consequence of unequal size segments is that there is no simple relationship between logical addresses and physical addresses
- The following steps are needed for address translation:



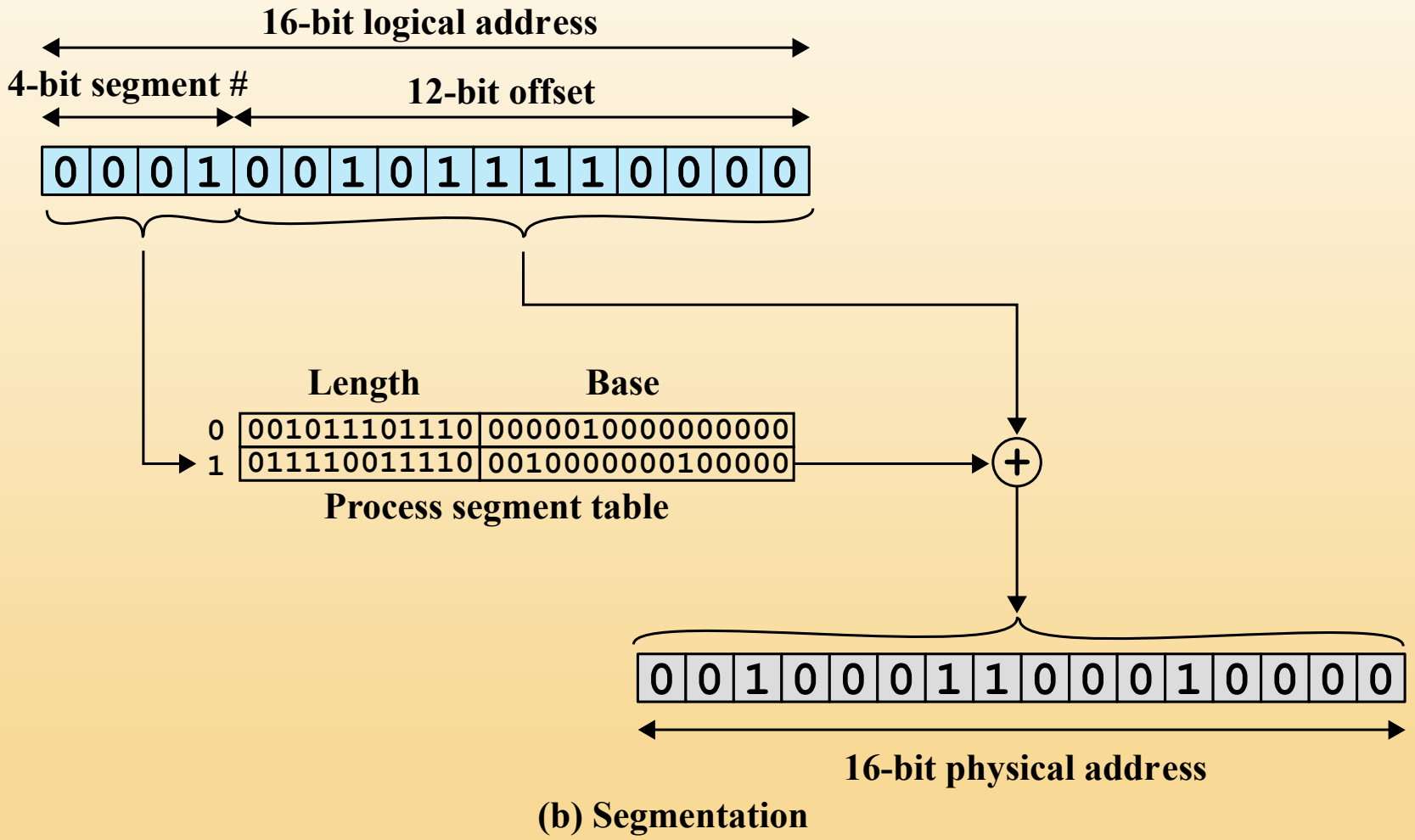


Figure 7.12 Examples of Logical-to-Physical Address Translation

Summary

- Memory management requirements
  - Relocation
  - Protection
  - Sharing
  - Logical organization
  - Physical organization
- Paging
- Memory partitioning
  - Fixed partitioning
  - Dynamic partitioning
  - Buddy system
  - Relocation
- Segmentation