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

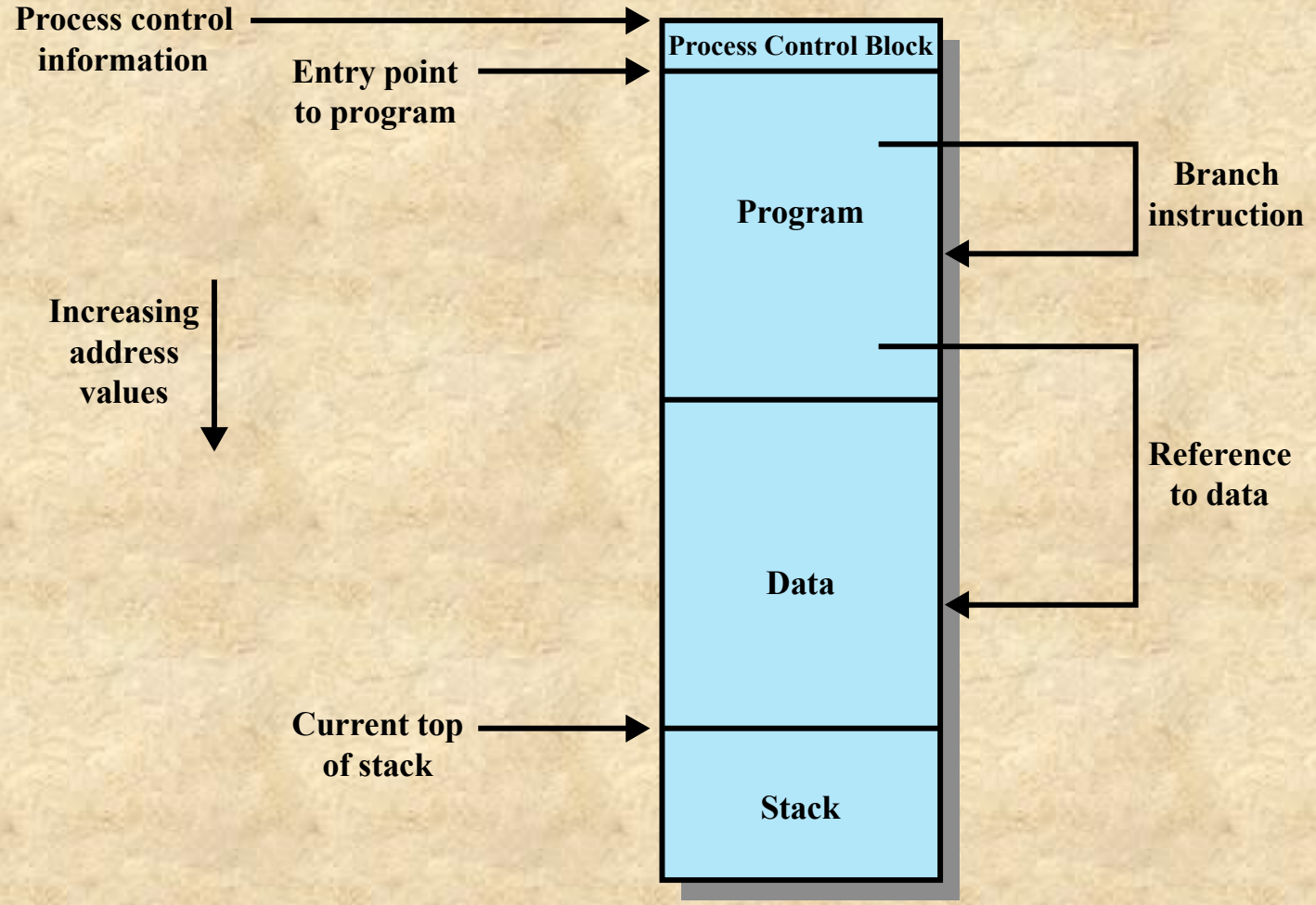
Memory Management Requirements

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
  - Instead, may need to *relocate* the process to a different area of memory



Assume the process image occupies a contiguous region of main memory. 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.

Figure 7.1 Addressing Requirements for a Process

Addressing Requirements for A Process

- The operating system will need to know
  - The location of process control information and of the execution stack
  - The entry point to begin execution of the program for this process.
- The processor must deal with memory references within the program
  - Branch instructions contain an address to reference the instruction to be executed next, https://en.wikipedia.org/wiki/Branch_(computer_science)
  - Data reference instructions contain the address of the byte or word of data referenced
- 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. But programs are organized into modules

Programs are written in modules

- Modules can be written and compiled independently
- Different degrees of protection given to modules (read-only, execute-only)
- Modules can be shared among processes and sharing on a module level corresponds to the user's way of viewing the problem. Easier for the user to specify the sharing

- 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

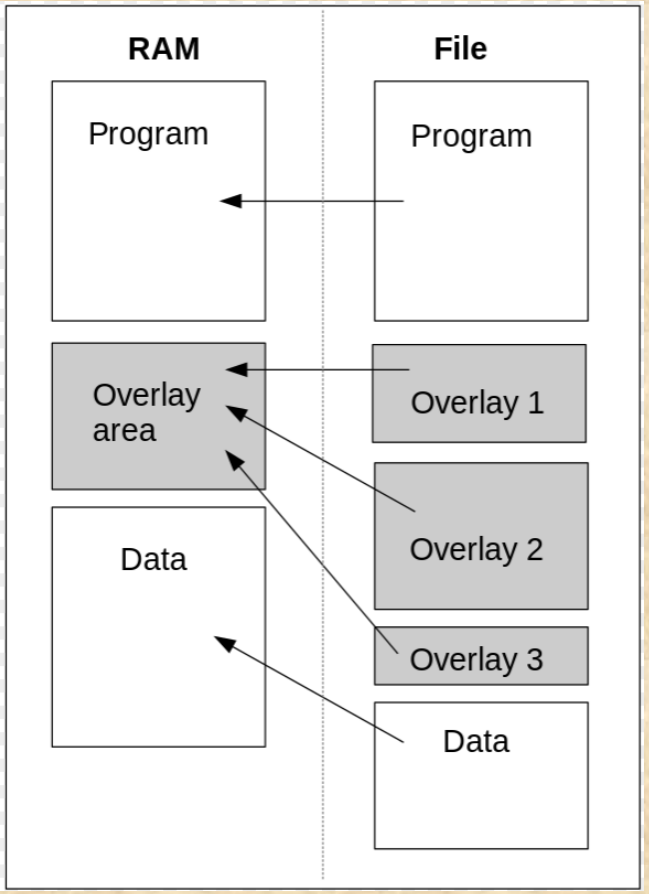
Programmer does not know how much space will be available

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

In overlay programming, there is a main program responsible for switching the modules in and out as needed

The task of moving information between the two levels of memory should be a system responsibility. This task is the essence of memory management

Overlay Programming



Programmers are responsible for splitting the program into segments and for loading them. There is a main program responsible for switching the modules in and out as needed. Even with the aid of compiler tools, overlay programming wastes programmer time. Overlays remain useful in embedded systems.

Figure: https://commons.wikimedia.org/wiki/File:Overlay_Programming.svg

Memory Partitioning

- The **principal operation** of memory management is to bring processes into main memory for execution by the processor
  - Involves virtual memory in modern multiprogramming systems. Based on segmentation and paging
- Partitioning
  - Used in several variations in some now-obsolete operating systems
  - **Does not involve virtual memory**

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)

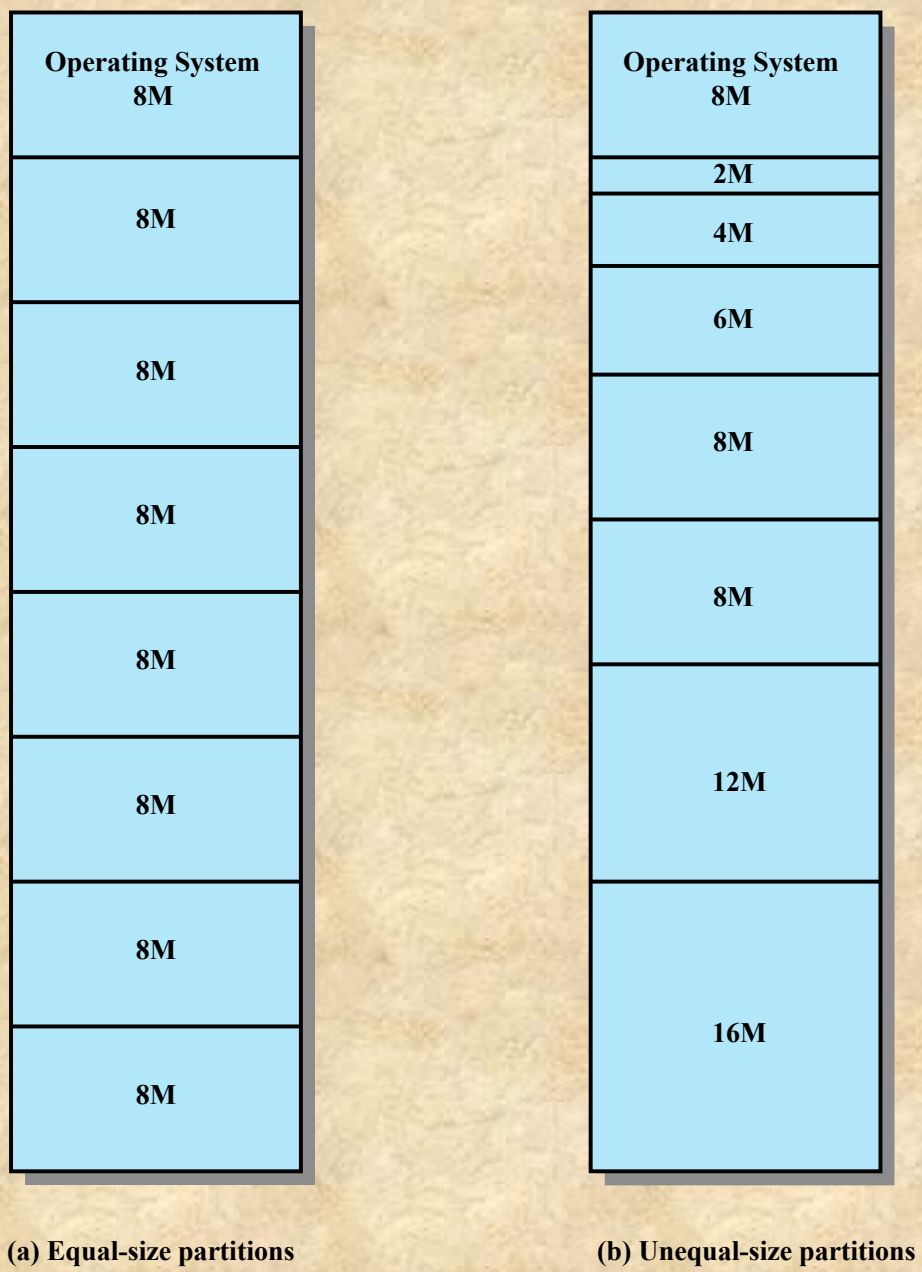
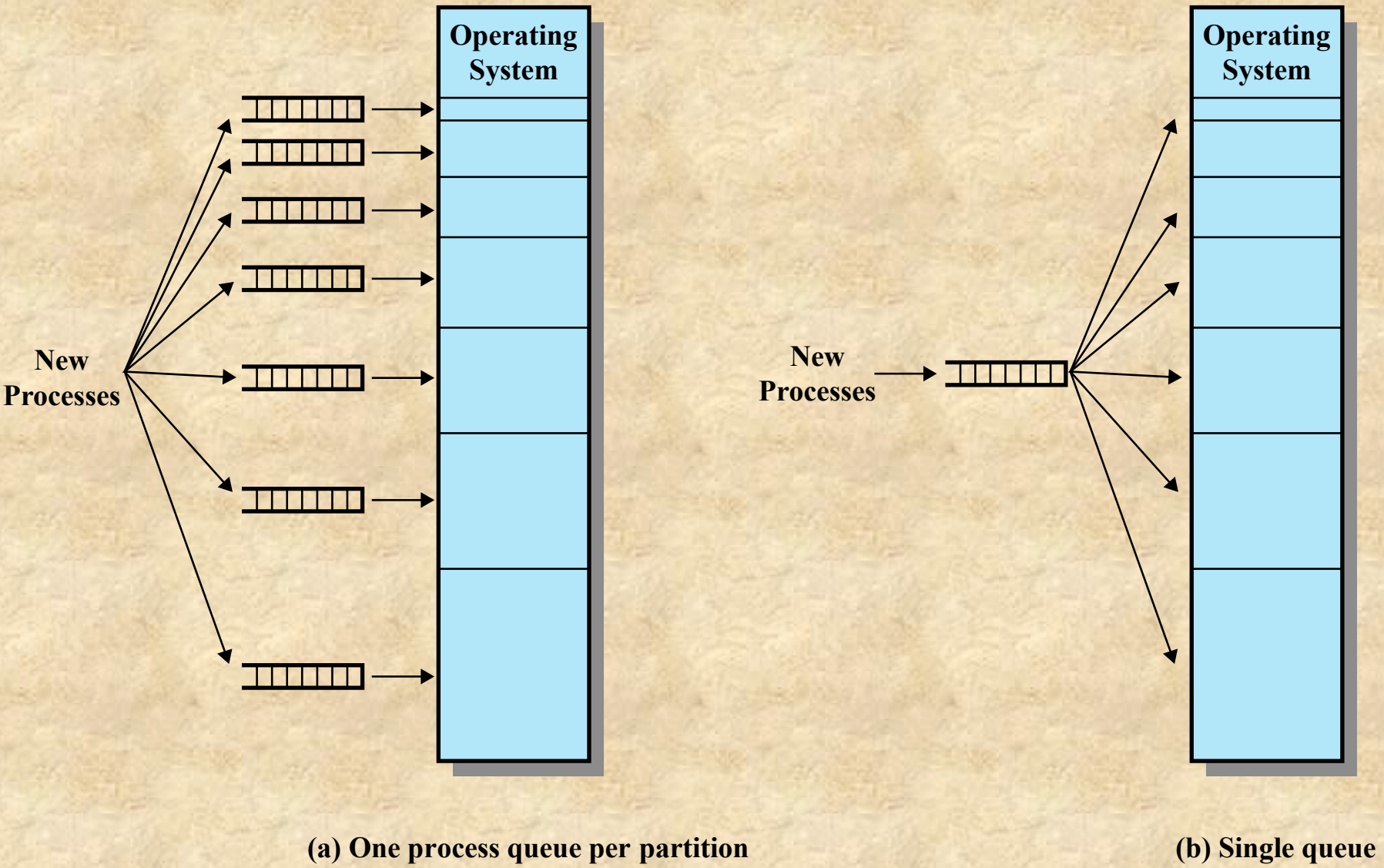


Figure 7.2 Example of Fixed Partitioning of a 64-Mbyte Memory

Disadvantages of Equal-Size Fixed Partitions

- A program may be too big to fit in a partition
  - Program needs to be designed with the use of overlays. **Overlay programming wastes programmer time**
- 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



(a) One process queue per partition

(b) Single queue

Figure 7.3 Memory Assignment for Fixed Partitioning

Process Assignment

With unequal-size partitions, there are two possible ways to assign processes to partitions

- 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)
- Employs a single queue for all processes (Figure 7.3b). When it is time to load a process into main memory, the smallest available partition that will hold the process is selected. If all partitions are occupied, then a swapping decision must be made

Process Assignment

- The first technique seems optimum from the point of view of an individual partition, it is not optimum from the point of view of the system as a whole. In Figure 7.2b , for example, consider a case in which there are no processes with a size between 12 and 16M at a certain point in time. In that case, the 16M partition will remain unused, even though some smaller process could have been assigned to it.

Disadvantages of Unequal-Size Fixed Partitions

- The **use of unequal-size partitions** provides a degree of flexibility to fixed partitioning; relatively simple and require minimal OS software and processing overhead
- Disadvantages (also applied to equal-size fixed partitions)
  - 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 since partition sizes are *preset*

Dynamic Partitioning

- Partitions are of variable length and number
- Process is allocated exactly as much memory as it requires
- No internal fragmentation; more efficient use of main memory
- This technique was used by IBM's mainframe operating system, OS/MVT

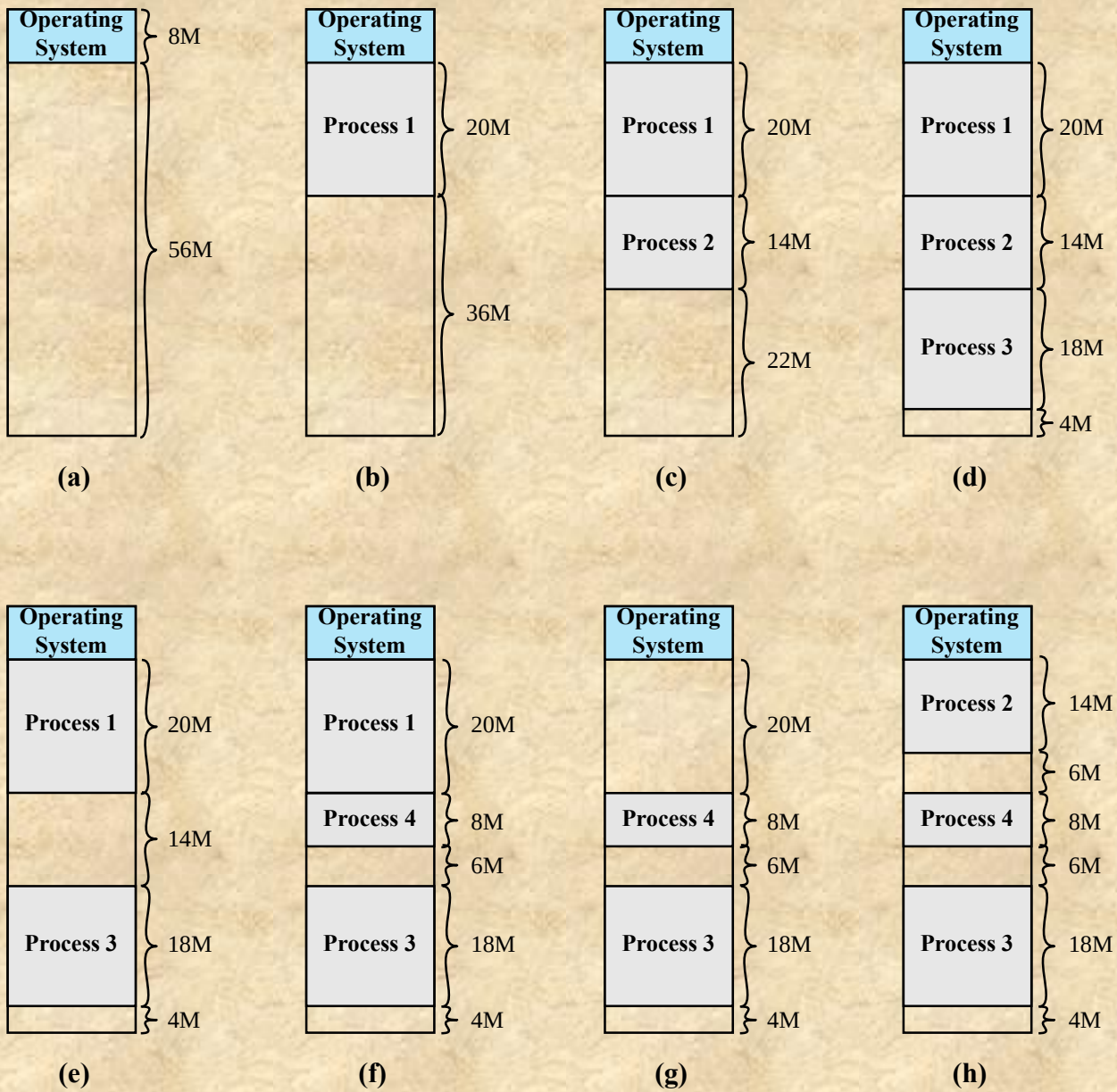


Figure 7.4 The Effect of Dynamic Partitioning

Dynamic Partitioning

External Fragmentation

- The memory that is external to all partitions 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 for Dynamic Partitioning

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

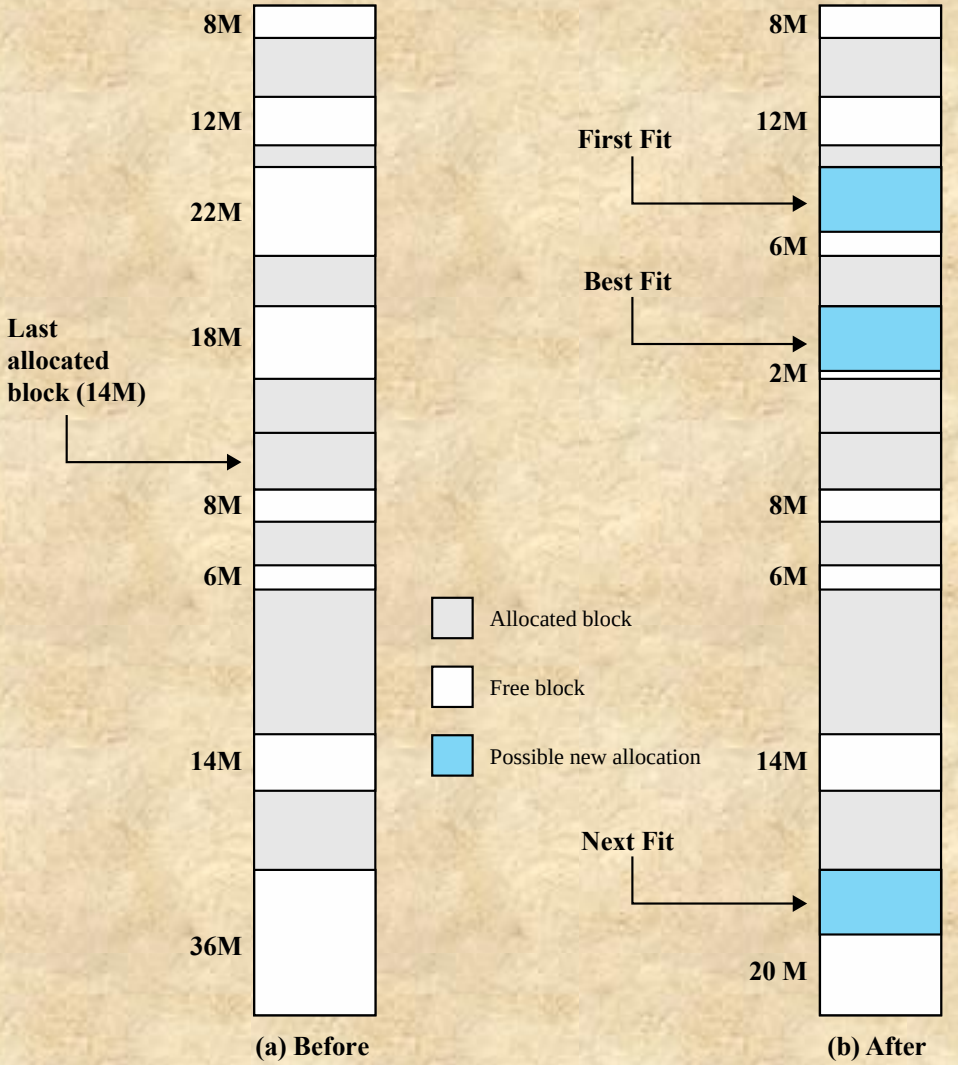


Figure 7.5 Example Memory Configuration before and after Allocation of 16-Mbyte Block

Placement Algorithms for Dynamic Partitioning

Which approach is the best depends on the exact sequence of process swapping that occurs and the size of those processes

- First-fit: the simplest, and usually the best and fastest as well
- Next-fit: tends to be slightly worse than the first-fit
  - More frequently lead to an allocation from a free block at the end of memory, so the largest block of free memory, which usually appears at the end of the memory space, is quickly broken up into small fragments
  - Compaction may be required more frequently with next-fit

Placement Algorithms for Dynamic Partitioning

- Best-fit: usually the worst performer
  - Each memory request always wastes the smallest amount of memory, the result is that main memory is quickly littered by blocks too small to satisfy memory allocation requests
  - Memory compaction must be done more frequently than with the other algorithms

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

Relocation

- When the fixed partition scheme in Figure 7.3a is used (one process queue per partition), 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

Relocation

- In the case of equal-size partitions (Figure 7.2a) and the case of a single process queue for unequal-size partitions (Figure 7.3b), a process may occupy different partitions during 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

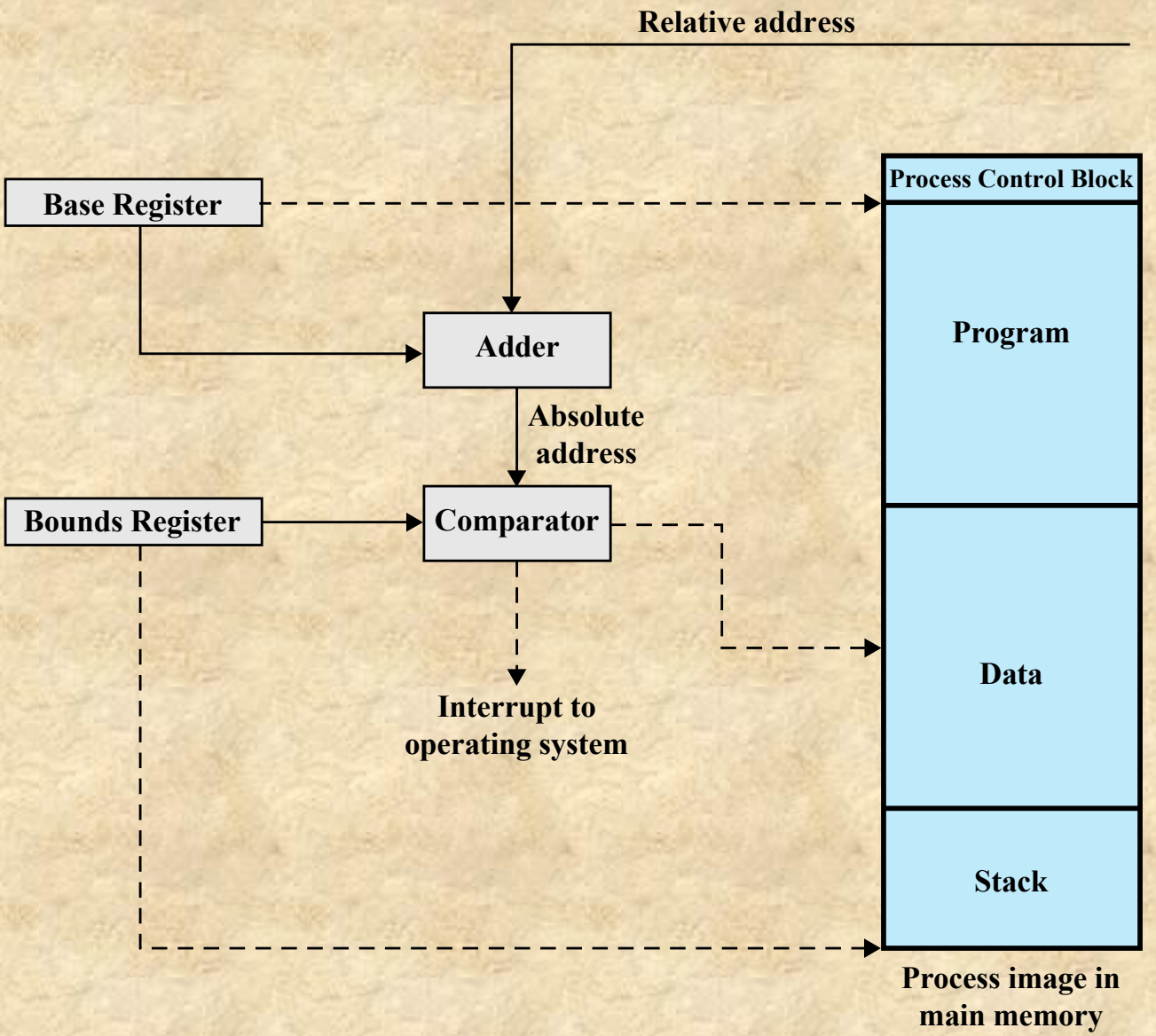


Figure 7.8 Hardware Support for Relocation

Hardware Support for Relocation

- 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. When a process is assigned to the Running state
  - A special processor register, sometimes called the **base register**, is loaded with the **starting address** in main memory of the program
  - A "bounds" register indicates the **ending location** of the program
- These values must be set when the program is loaded into memory or when the process image is swapped in

Hardware Support for Relocation

- During the course of execution of the process, relative addresses go through two steps of manipulation by the processor
  - First, the value in the base register is added to the relative address to produce an absolute address
  - Second, the resulting address is compared to the value in the bounds register. If the address is within bounds, then the instruction execution may proceed. Otherwise, an interrupt is generated to the operating system, which must respond to the error in some fashion

Paging

Both fixed partitioning and dynamic partitioning are inefficient in the use of memory

- The former results in internal fragmentation, the latter in external fragmentation

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
- Paging eliminates external fragmentation, but still suffers from internal fragmentation

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

Simple paging is similar to fixed partitioning. The differences are that

- With paging, the partitions are rather small
- A program may occupy more than one partition; and these partitions need not be contiguous

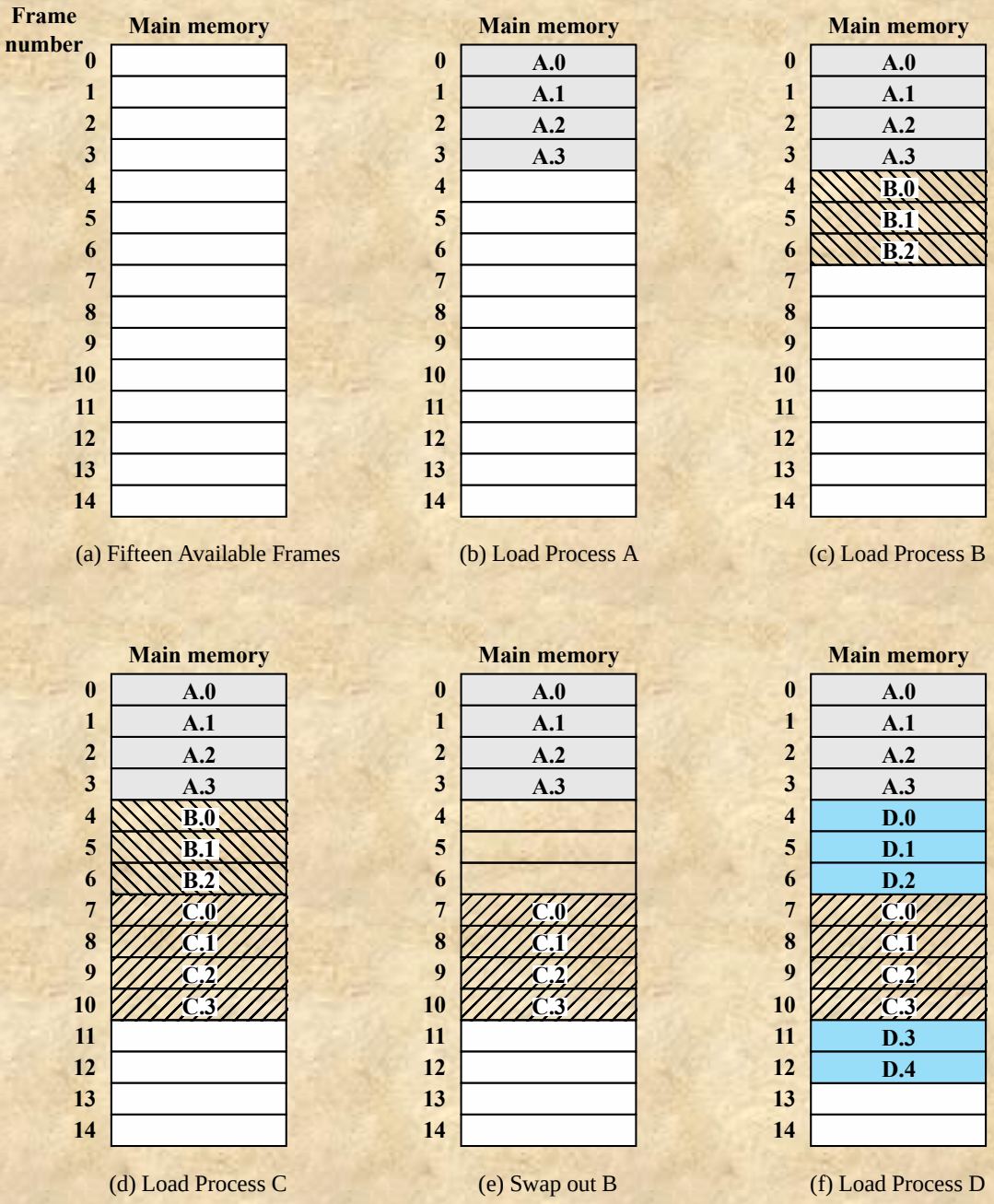


Figure 7.9 Assignment of Process Pages to Free Frames

Page Table

- A simple base address register will no longer suffice (why?). Rather, the operating system maintains a **page table** for each process
  - Contains the frame location for each page in the process
    - Within the program, each logical address consists of a page number and an offset within the page
  - Processor must know how to access the page table of the current process
    - Presented with a logical address, the processor uses the logical address to produce a physical address

Each page table entry contains the number of the frame in main memory, if any, that holds the corresponding page. In addition, the OS maintains a single free-frame list of all the frames in main memory that are currently unoccupied and available for pages

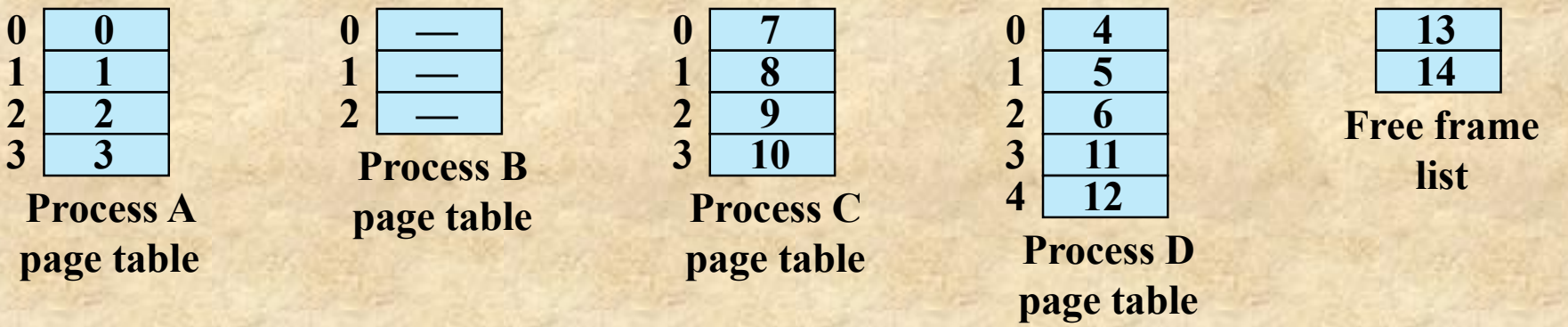
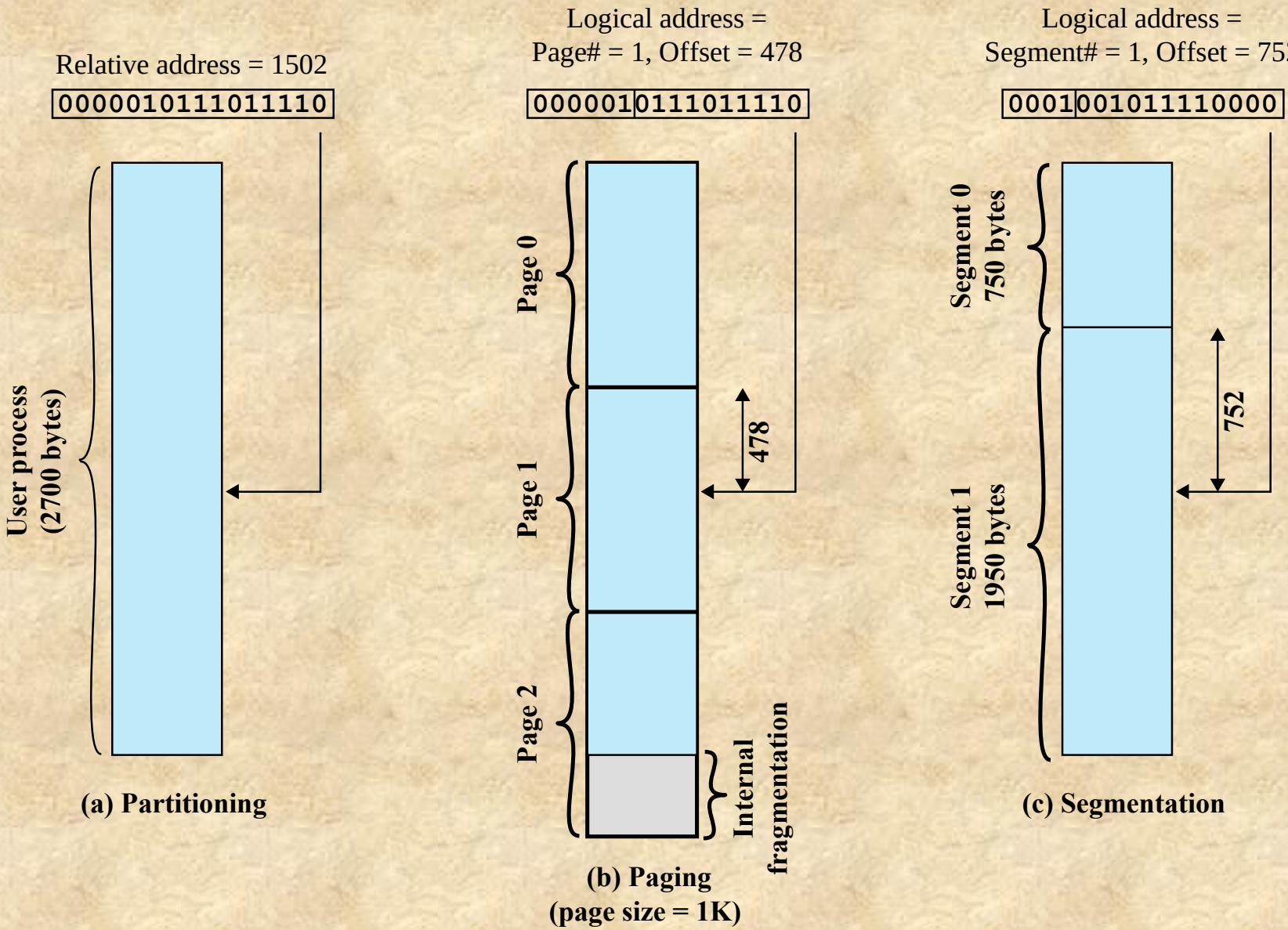


Figure 7.10 Data Structures for the Example of Figure 7.9 at Time Epoch (f)



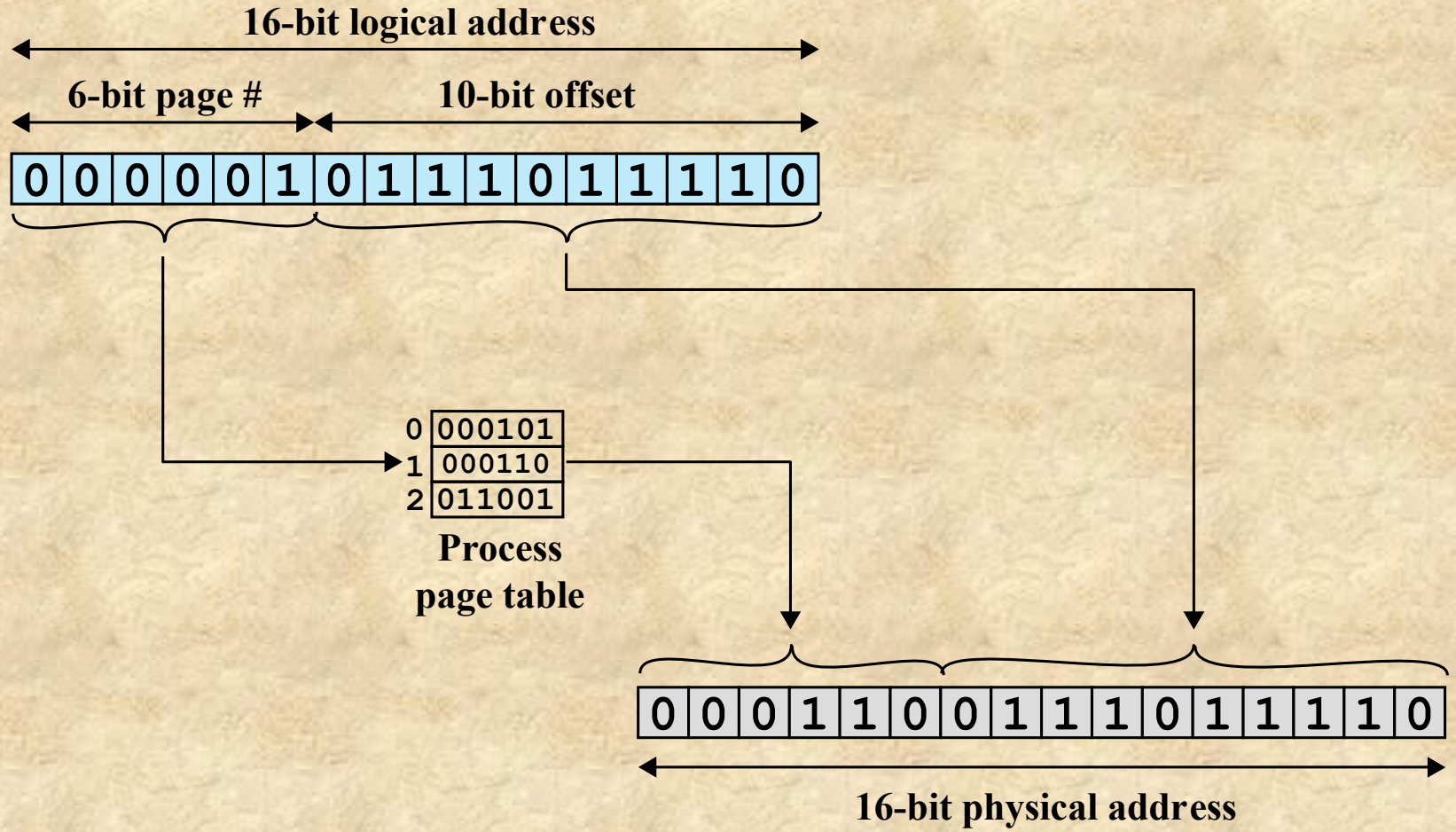
In this example, 16-bit addresses are used, and the page size is 1K = 1,024 bytes

Figure 7.11 Logical Addresses

Address Translation for Paging

- The following steps are needed for address translation
  - Extract the page number as the leftmost n bits of the logical address
  - Use the page number as an index into the process page table to find the frame number, k.
  - The starting physical address of the frame is $k \times 2^m$, and the physical address of the referenced byte is that number plus the offset

https://www.rapidtables.com/convert/number/binary-to-decimal.html



(a) Paging

Figure 7.12 Examples of Logical-to-Physical Address Translation

Simple Paging Summary

- With simple paging, main memory is divided into many small equal-size frames. Each process is divided into frame-size pages. Smaller processes require fewer pages; larger processes require more. When a process is brought in, all of its pages are loaded into available frames, and a page table is set up

- The internal fragmentation consists of only a fraction of the last page of a process

Segmentation

- A program can be subdivided into segments
  - May vary in length
  - There is a maximum length
- A logical address using segmentation consists of two parts:
  - Segment number
  - An offset

Segmentation

- Similar to dynamic partitioning due to the use of unequal-size segments
  - In the absence of an overlay scheme or the use of virtual memory, it would be required that all of a program's segments be loaded into memory for execution
  - But may occupy more than one partition, and these partitions need not be contiguous
- Eliminates internal fragmentation
  - May have external fragmentation like dynamic partitioning

Segmentation

- Paging is invisible to the programmer; but segmentation is 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 for Segmentation

- Another consequence of unequal size segments is that there is no simple relationship between logical addresses and physical addresses
- Use a segment table for each process and a list of free blocks of main memory
- Each segment table entry would have to give the starting address in main memory of the corresponding segment. The entry should also provide the length of the segment to assure that invalid addresses are not used. When a process enters the Running state, the address of its segment table is loaded into a special register used by the memory management hardware

Address Translation for Segmentation

The following steps are needed for address translation:

1. Extract the segment number as the leftmost *n* bits of the logical address
2. Use the segment number as an index into the process segment table to find the starting physical address of the segment
3. Compare the offset, expressed in the rightmost *m* bits, to the length of the segment. If the offset is greater than or equal to the length, the address is invalid
4. The desired physical address is the sum of the starting physical address of the segment plus the offset

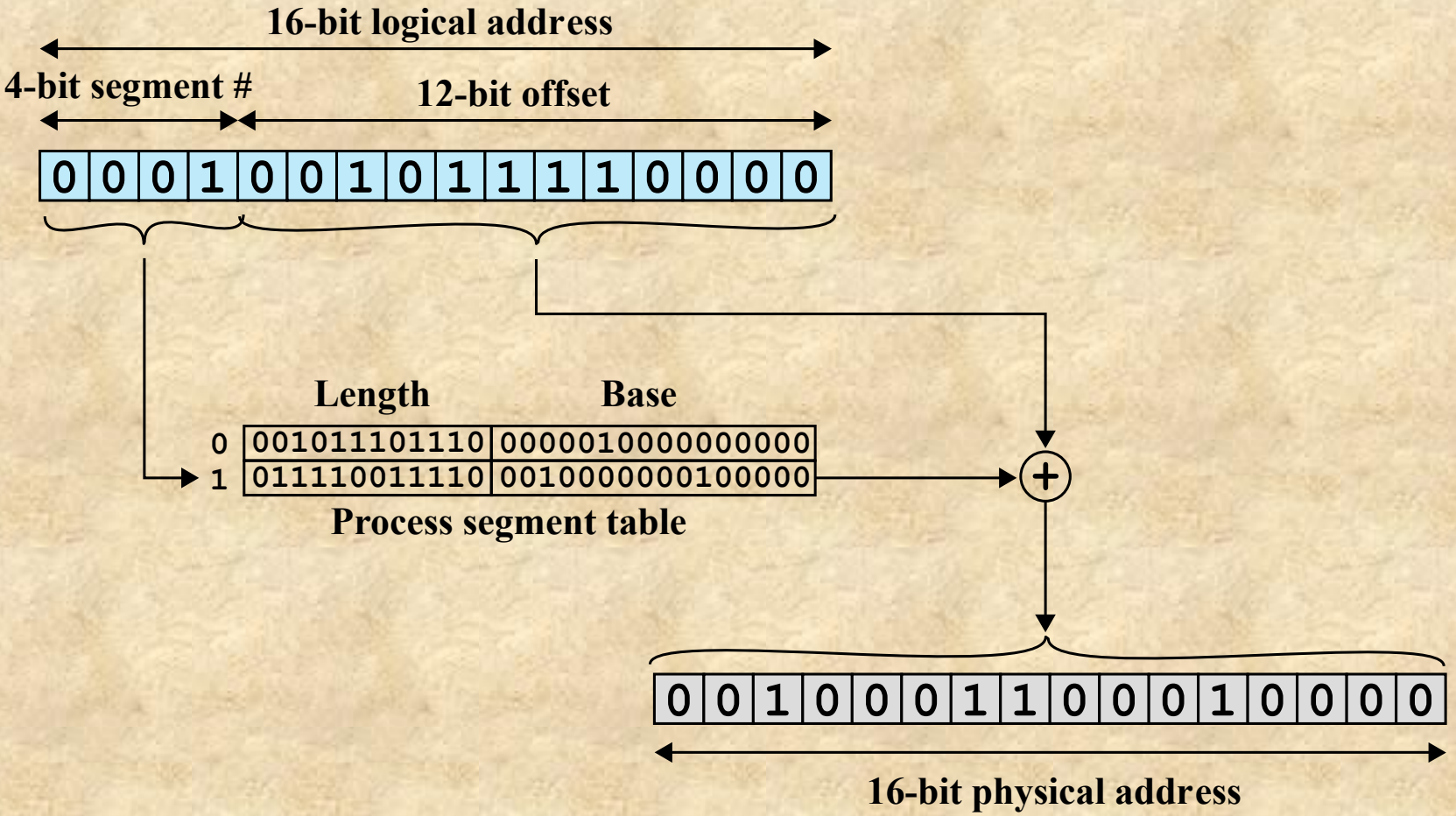


Figure 7.12 Examples of Logical-to-Physical Address Translation

Simple Segmentation Summary

- With simple segmentation, a process is divided into a number of segments that need not be of equal size
- When a process is brought in, all of its segments are loaded into available regions of memory, and a segment table is set up

7.14 Consider a simple segmentation system that has the following segment table:

Starting Address	Length (bytes)
660	248
1,752	422
222	198
996	604

For each of the following logical addresses, determine the physical address or indicate if a segment fault occurs:

a. 0, 198
b. 2, 156
c. 1, 530
d. 3, 444
e. 0, 222

Summary

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