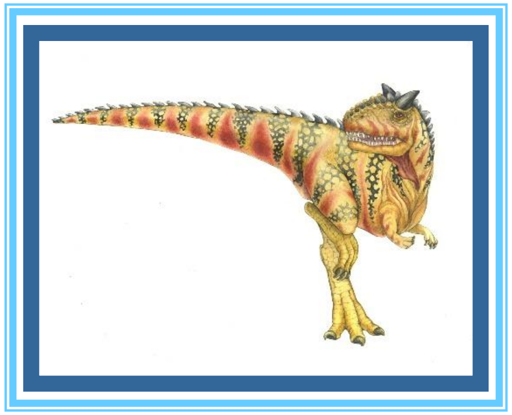
Operating Systems



Lecture 8:
Memory Management

Chapter 7:
Memory Management

Contact Information

Instructor: Engr. Shamila Nasreen

Lecturer

Department of Software Engineering

MUST

Email: shamila_nasreen131@yahoo.com

Office hours: Wednesday: 8:30–10:00 AM

Thursday: 8:30–10:00 AM

Books

- **Text Book:**

  Silberschatz, Galvin, “Operating Sytems Concepts” 8th Edition, John Wiley, 2007

Reference Book:

1. William Stallings, “Operating Systems”
2. Harvey M. Deitel, “Operating Systems Concepts”

Memory Management

- In a uniprogramming system, main memory is divided into two parts:
  - one part for the operating system (resident monitor, kernel)
  - one part for the program currently being executed.
- In a multiprogramming system, the “user” part of memory must be further subdivided to accommodate multiple processes.
- The task of subdivision is carried out dynamically by the operating system and is known as **memory management .**
- Effective memory management is vital in a multiprogramming system.
  - If only a few processes are in memory, then for much of the time all of the processes will be waiting for I/O and the processor will be idle.
  - Thus memory needs to be allocated to ensure a reasonable supply of ready processes to consume available processor time.

Definition

- Memory management is the process of
  - allocating primary memory to user programs
  - reclaiming that memory when it is no longer needed
  - protecting each user’s memory area from other user programs; i.e., ensuring that each program only references memory locations that have been allocated to it.

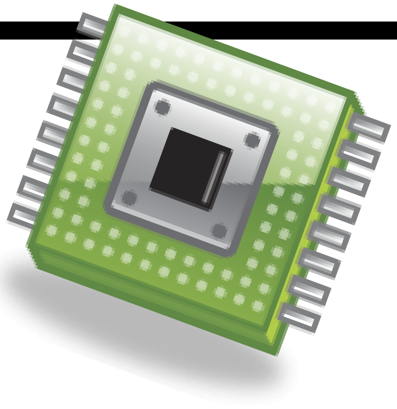
Silberschatz and Galvin ©1999

Requirements

- In order to manage memory effectively the OS must have
  - Memory allocation policies
  - Methods to track the status of memory locations (free or allocated)
  - Policies for preempting memory from one process to allocate to another

Silberschatz and Galvin ©1999

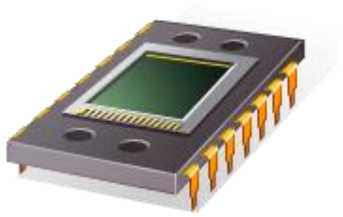
Memory Management Terms



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).**

Memory Partitioning

- Virtual memory management brings processes into main memory for execution by the processor
  - involves virtual memory
  - based on segmentation and paging
- Partitioned memory management
  - used in several variations in some now-obsolete operating systems
  - does not involve virtual memory



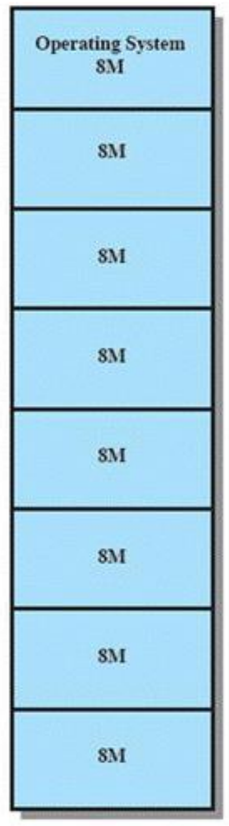
Silberschatz and Galvin ©1999

Table 7.2 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.

Fixed Partitioning

- Equal-size partitions
  - memory is to partition it into regions with fixed boundaries
  - any process whose size is less than or equal to the partition size can be loaded into an available partition
- The operating system can swap out a process if all partitions are full and no process is in the Ready or Running state



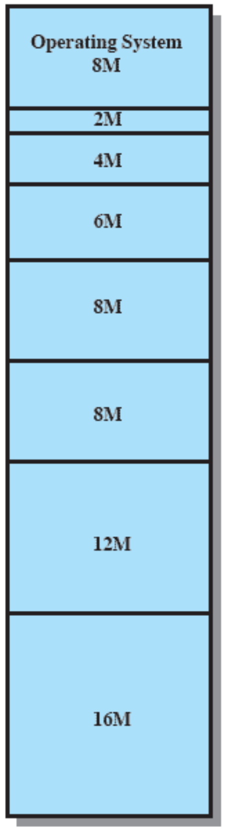
(a) Equal-size partitions

Disadvantages

- A program may be too big to fit in a partition
  - program needs to be designed with the use of overlays
  - only a portion of the program need be in main memory at any one 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

Unequal Size Partitions

- Using unequal size partitions helps lessen the problems
  - programs up to 16M can be accommodated without overlays
  - partitions smaller than 8M allow smaller programs to be accommodated with less internal fragmentation



(b) Unequal-size partitions

PLACEMENT ALGORITHM

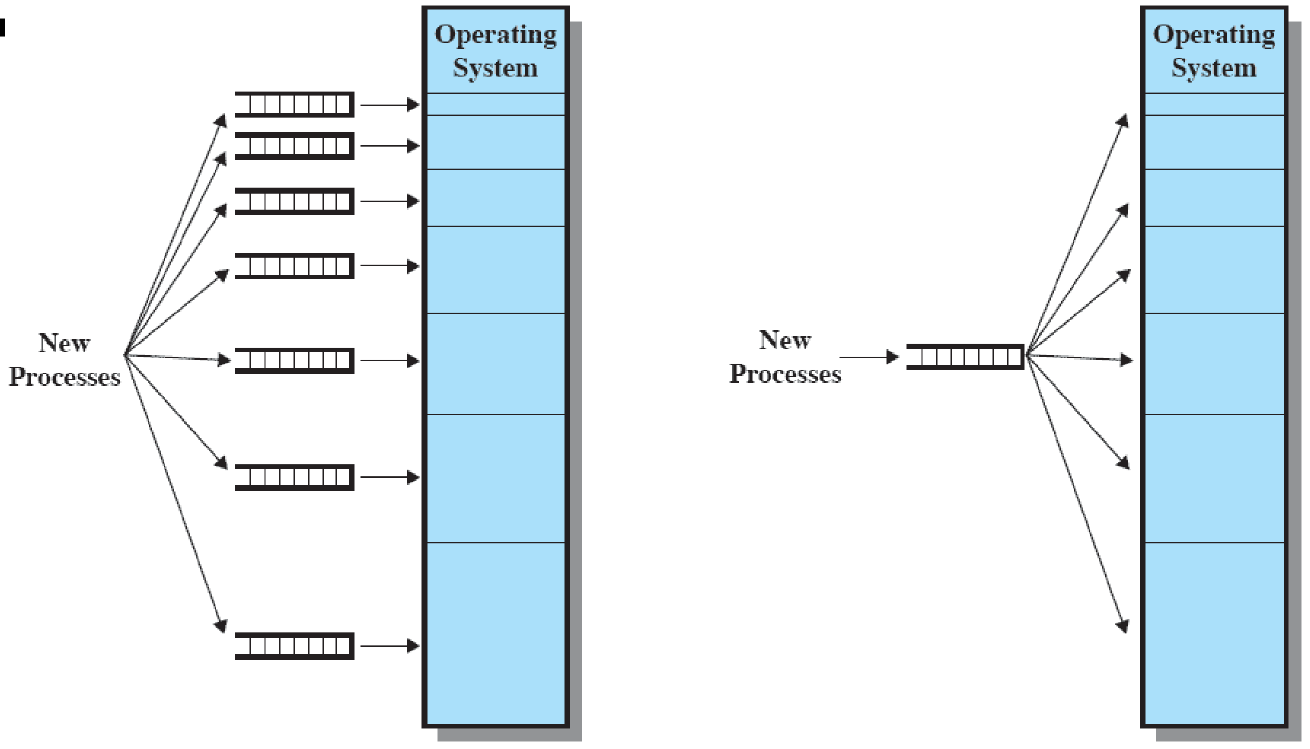
- *With* ***equal-size partitions****, the placement of processes* in memory is trivial.
- As long as there is any available partition, a process can be loaded into that partition.
- All partitions are of equal size, it does not matter which partition is used.
- If all partitions are occupied with processes that are not ready to run, then one of these processes must be swapped out to make room for a new process.
  - Which one to swap out is a scheduling decision

PLACEMENT ALGORITHM

- With **unequal-size** partitions, there are two possible ways:
- 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).
  - **Advantage:** processes are always assigned in such a way as to minimize wasted memory within a partition (internal fragmentation).
  - Disadvantage: It is not optimum from the point of view of the system as a whole.
- A preferable approach would be to employ 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. Preference might be given to swapping out of the smallest partition that will hold the incoming process.

Memory Assignment

Fixed Partitioning



(a) One process queue per partition

(b) Single queue

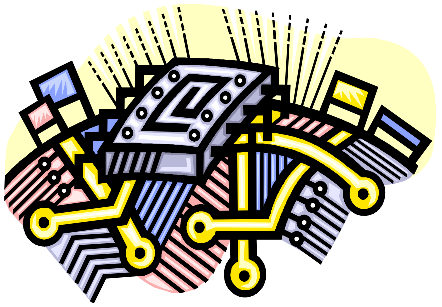
Figure 7.3 Memory Assignment for Fixed Partitioning

Advantage & Disadvantages

- Advantage:
  - unequal-size partitions provides a degree of flexibility to fixed partitioning.
  - fixed-partitioning schemes are relatively simple and require minimal OS software and processing overhead.
- Disadvantage:
  - 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



Effect of Dynamic Partitioning

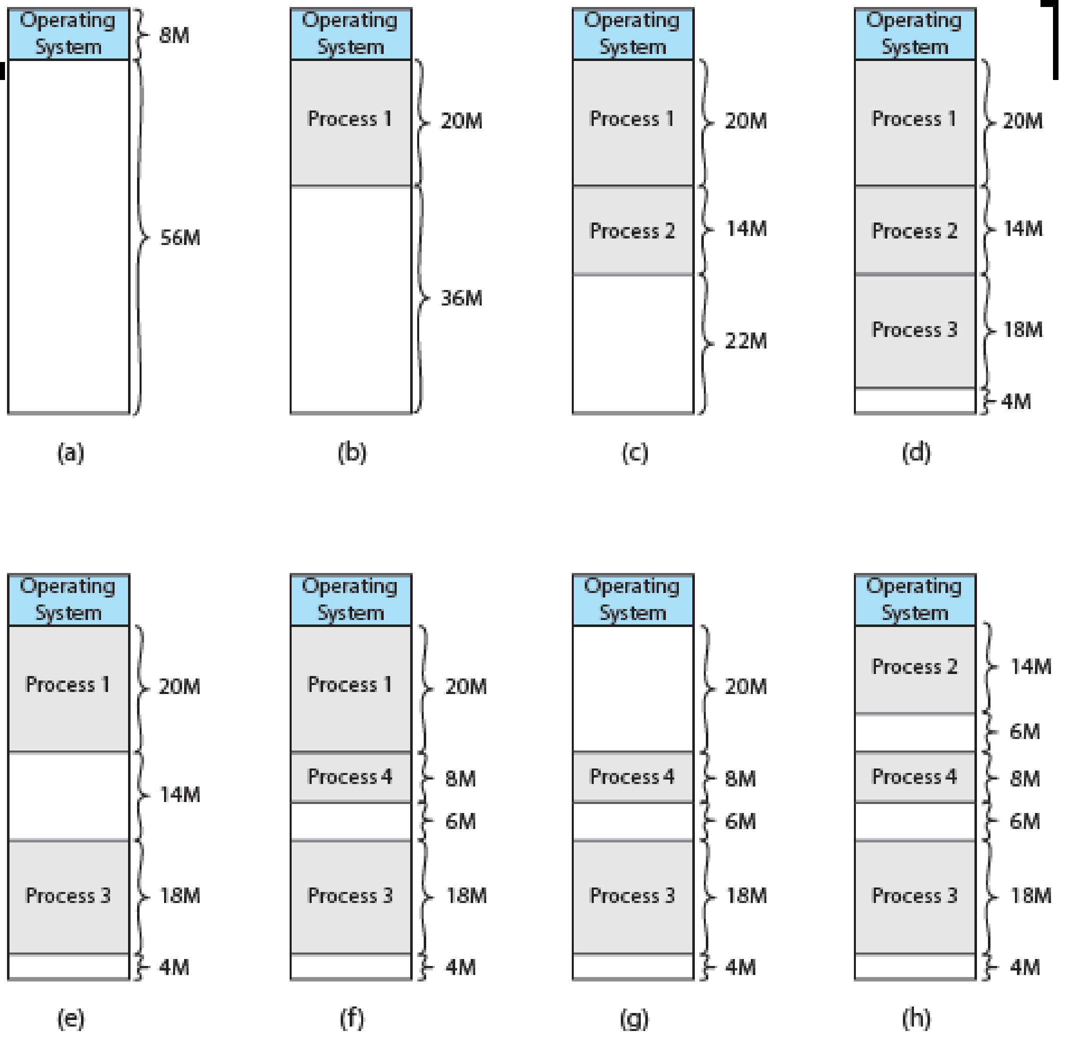


Figure 7.4 The Effect of Dynamic Partitioning

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

Placement Algorithms

- The **first-fit** algorithm is not only the simplest but usually the best and fastest as well.
  - the first-fit algorithm may litter the front end with small free partitions
  - that need to be searched over on each subsequent first-fit pass
- The **next-fit** algorithm tends to produce slightly worse results than the first-fit.
  - The next-fit algorithm will more frequently lead to an allocation from a free block at the end of memory.
  - The result is that 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

- The **best-fit** algorithm, despite its name, is usually the worst performer.
  - This algorithm looks for the smallest block that will satisfy the requirement, it guarantees that the fragment left behind is as small as possible.
  - 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.
  - Thus, memory compaction must be done more frequently than with the other algorithms.

Memory Configuration Example

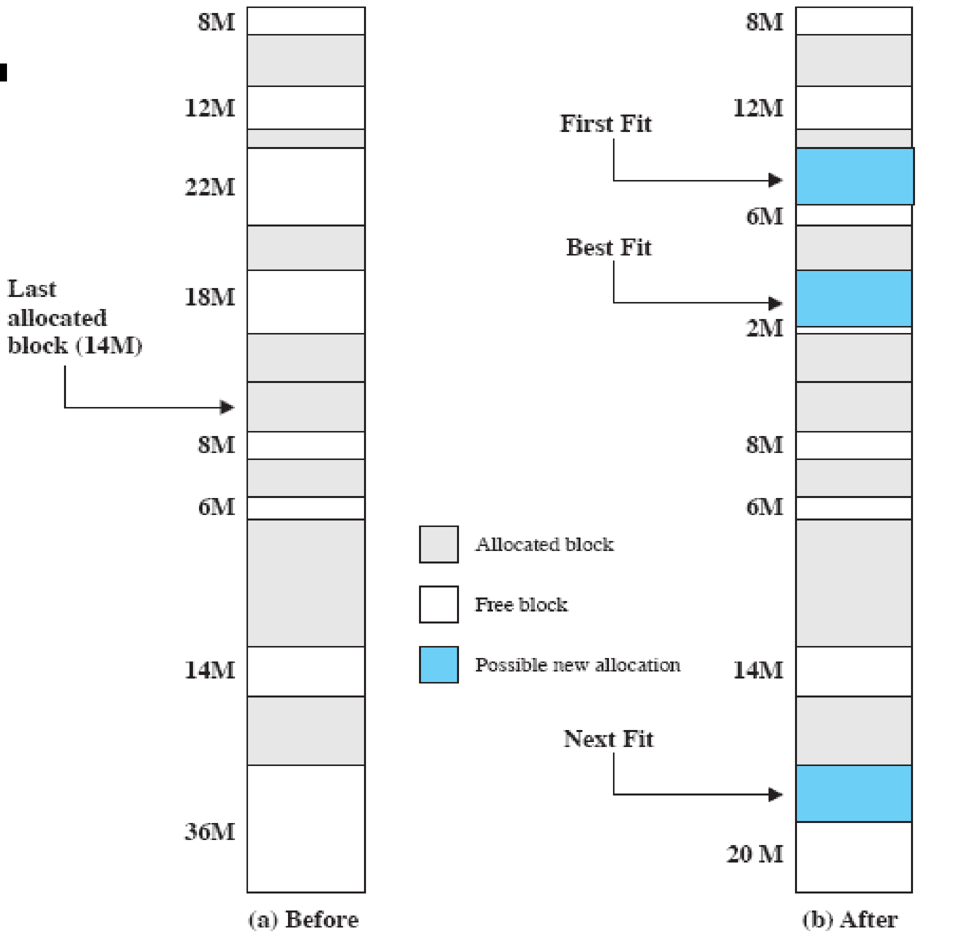


Figure 7.5 Example Memory Configuration before and after Allocation of 16-Mbyte Block

Silberschatz and Galvin ©1999

Addresses

Logical

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

Relative

- address is expressed as a location relative to some known point

Physical or Absolute

- actual location in main memory

Addressing

- Programs that employ relative addresses in memory are loaded using dynamic run-time loading.
- All of the memory references in the loaded process are relative to the origin of the program.
- 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.

Relocation

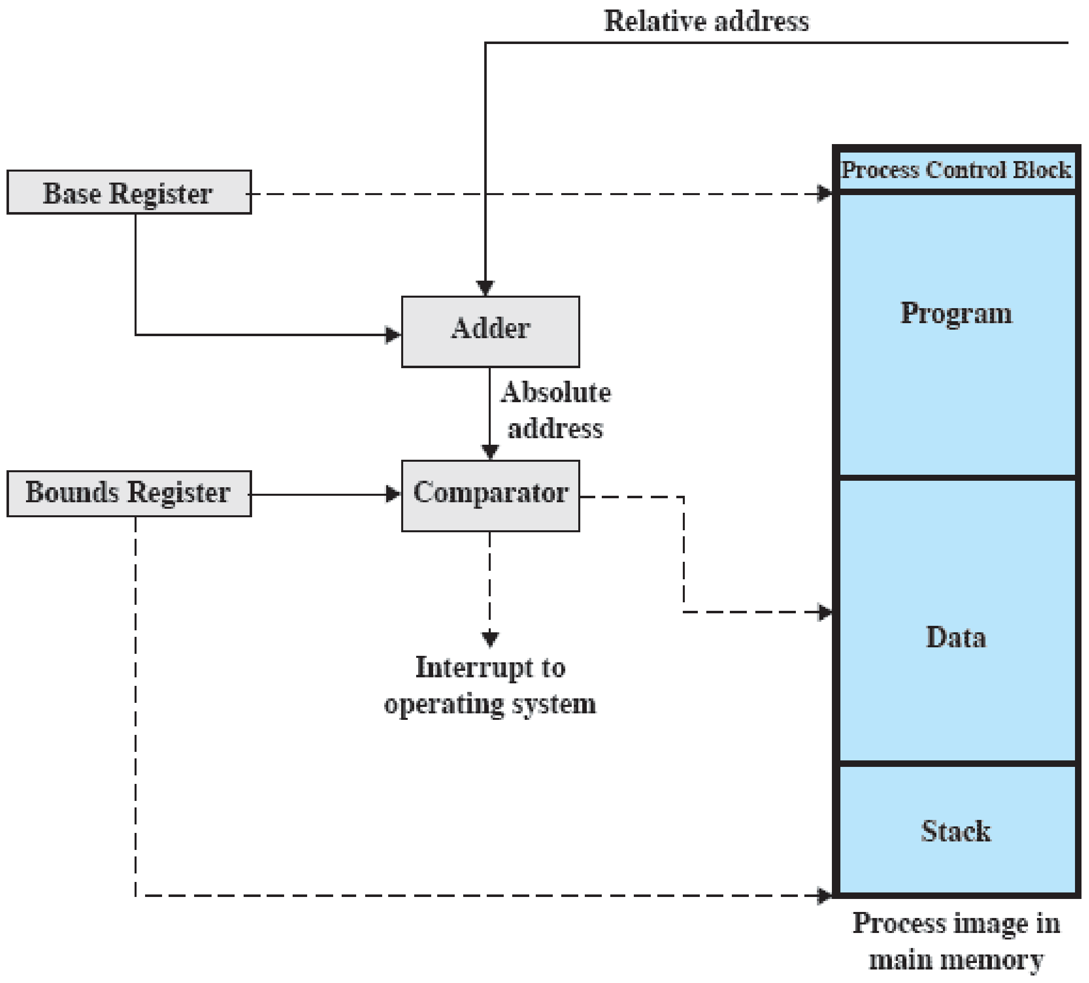


Figure 7.8 Hardware Support for Relocation

Paging

- Both unequal fixed-size and variable-size partitions are inefficient in the use of memory.
  – the former results in internal fragmentation, the latter in external fragmentation.
- In paging, space of a process to be noncontiguous
- - Avoids external fragmentation,- Avoids the need for compaction
- - May still have internal fragmentation

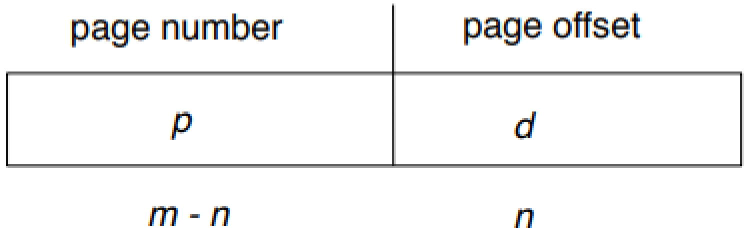
Paging

- Divide physical memory into fixed-sized blocks called frames
  - Size is power of 2, between 512 bytes and 16 Mbytes
  - Must keep track of all free frames
- Divide logical memory into blocks of same size called pages
  - Backing store is also split into pages of the same size
- • To run a program of size N pages, need to find N free frames and load program

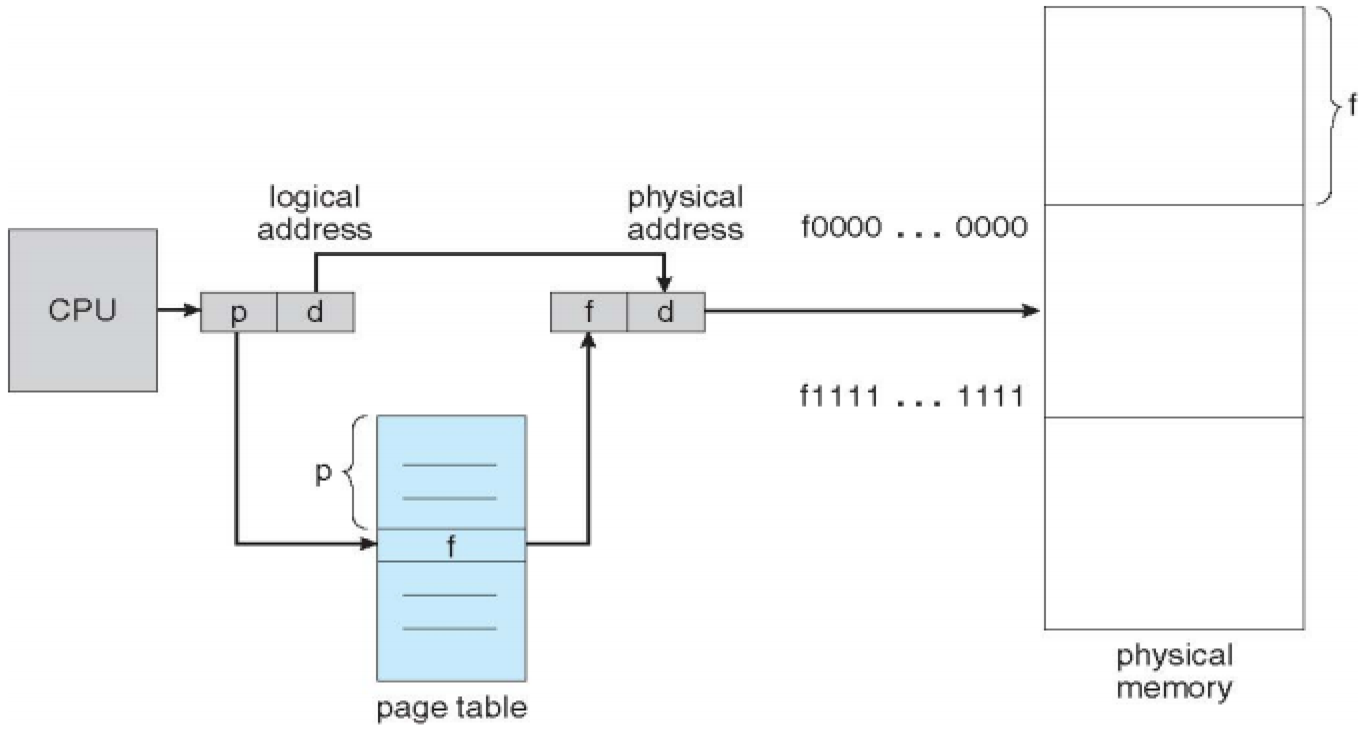
Pages	***Frames***
• chunks of a process	• available chunks of memory

Address Translation Scheme

- Set up a page table to translate logical to physical addresses
  - Address generated by CPU is divided into:
  - **Page number (p)** – used as an index into a page table which contains base
- address of each page in physical memory
  - **Page offset (d)** – combined with base address to define the physical memory
- address that is sent to the memory unit
  - For given logical address space 2m and page size 2n



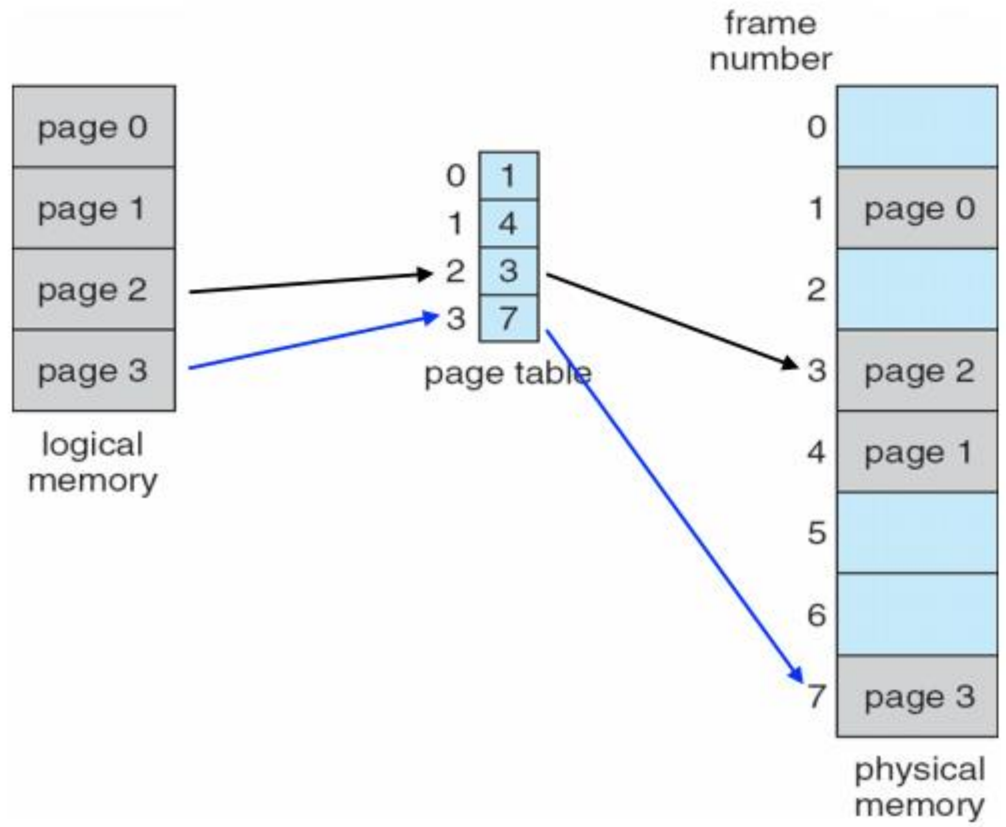
Paging Hardware



Page Table

- A simple base address register will no longer suffice.
- Maintained by operating system for each process
- Contains the frame location for each page in the process
- With paging, the logical-to-physical address translation is still done by processor hardware.
  – Processor must know how to access the page table for the current process
- Presented with a logical address (page number, offset), the processor uses the page table to produce a physical address (frame number, offset).

Paging Model of Logical and Physical Memory



Assignment of Process to Free Frames

At a given point in time, some of the frames in memory are in use and some are free. A list of free frames is maintained by the OS.

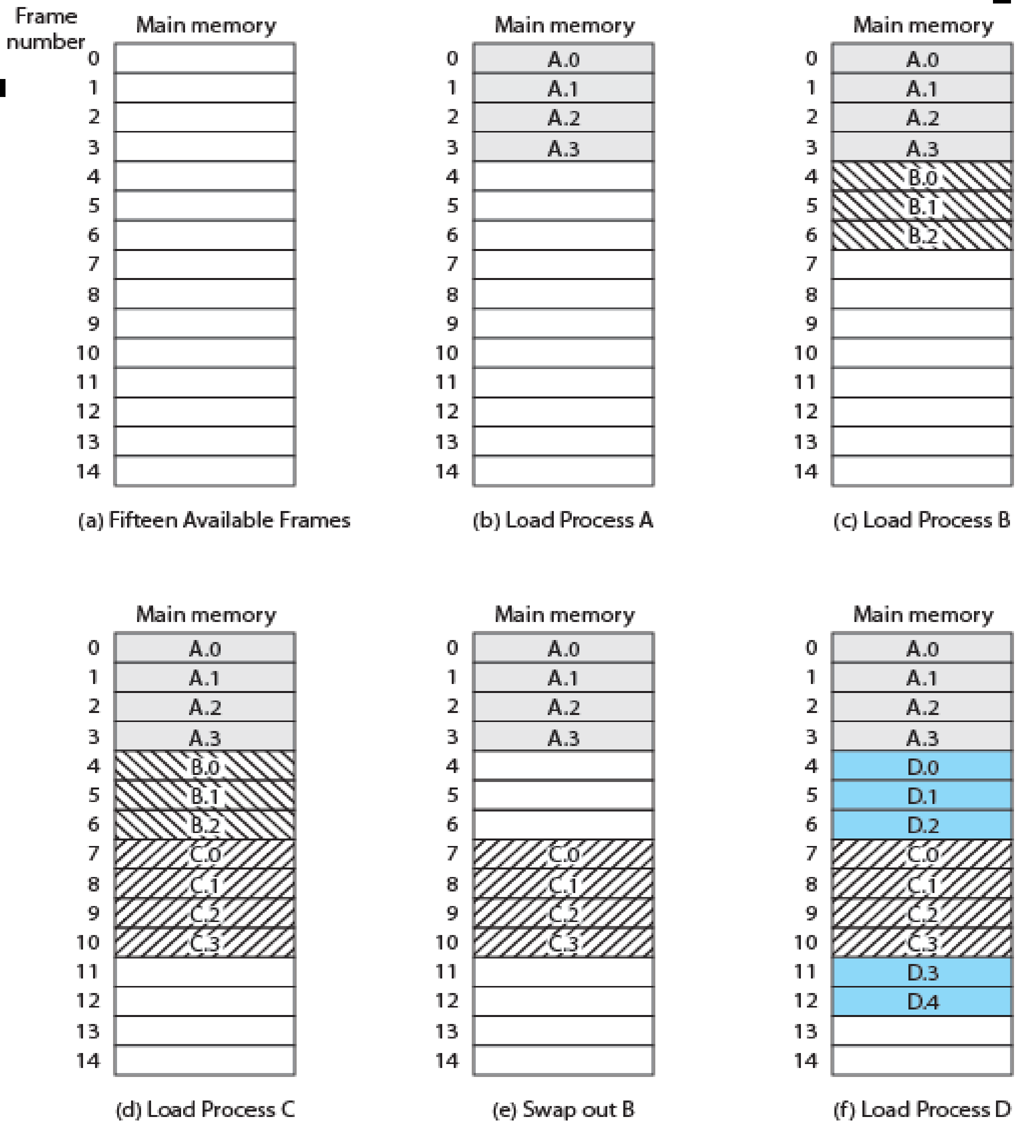
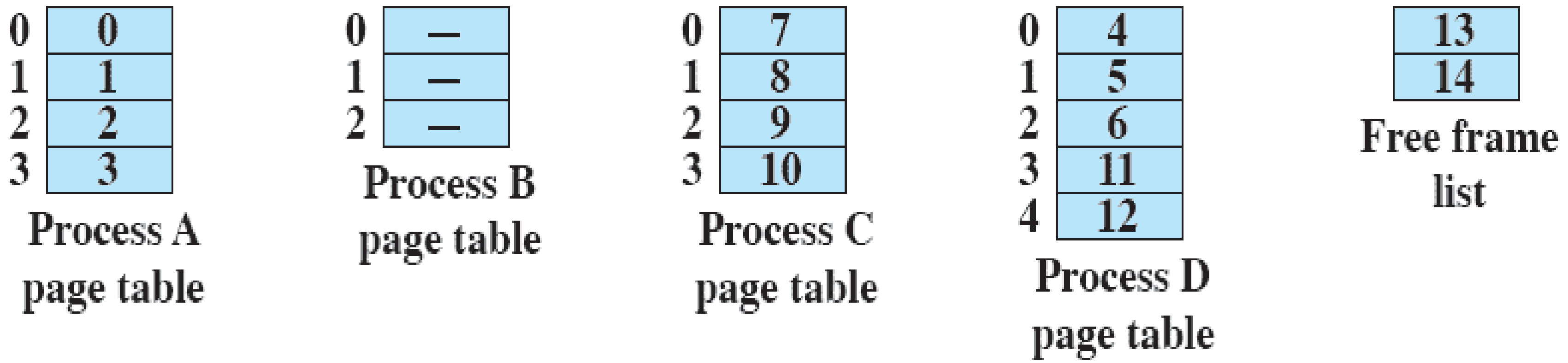


Figure 7.9 Assignment of Process Pages to Free Frames

Paging Model of Logical and Physical Memory



	Process A page table
0	0
1	1
2	2
3	3

	Process B page table
0	–
1	–
2	–

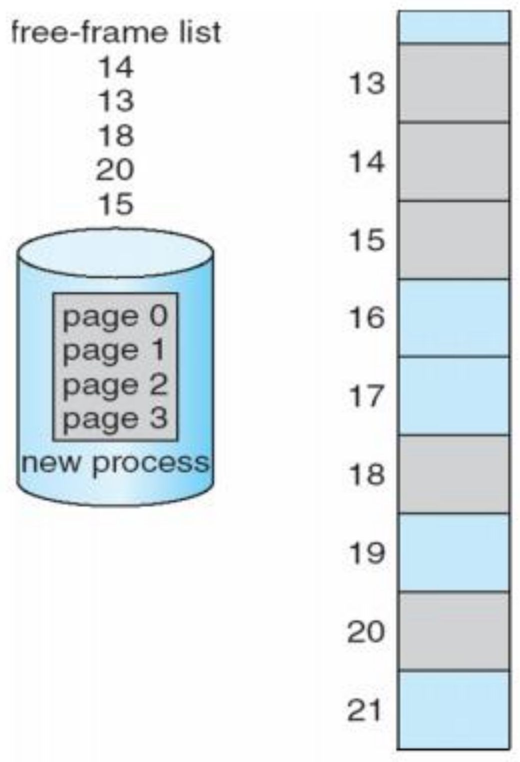
	Process C page table
0	7
1	8
2	9
3	10

	Process D page table
0	4
1	5
2	6
3	11
4	12

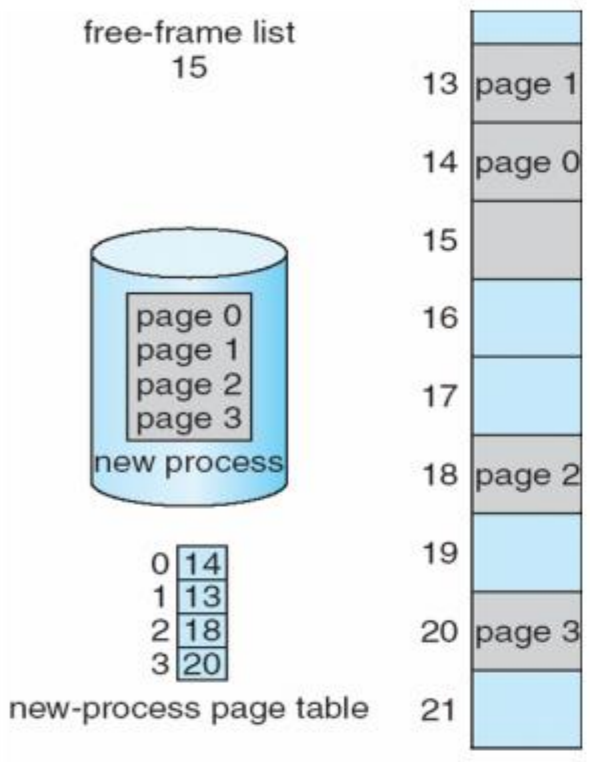
Free frame list
13
14

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

Allocating Frames to a New Process

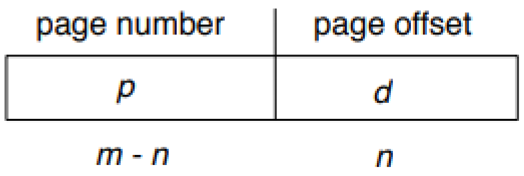


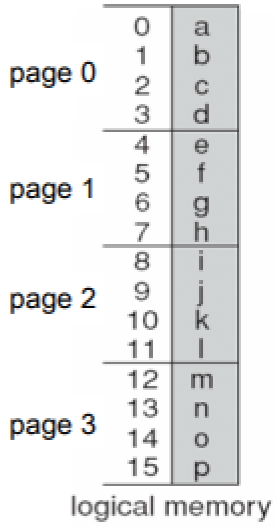
Before allocation

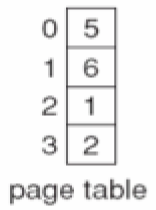


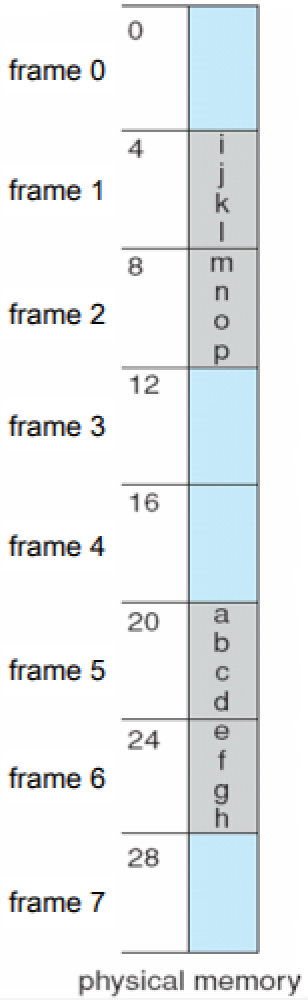
After allocation

Paging Example



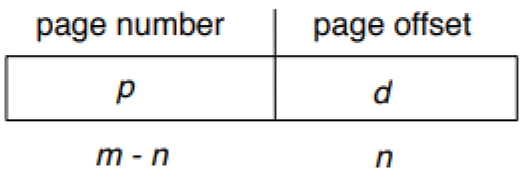


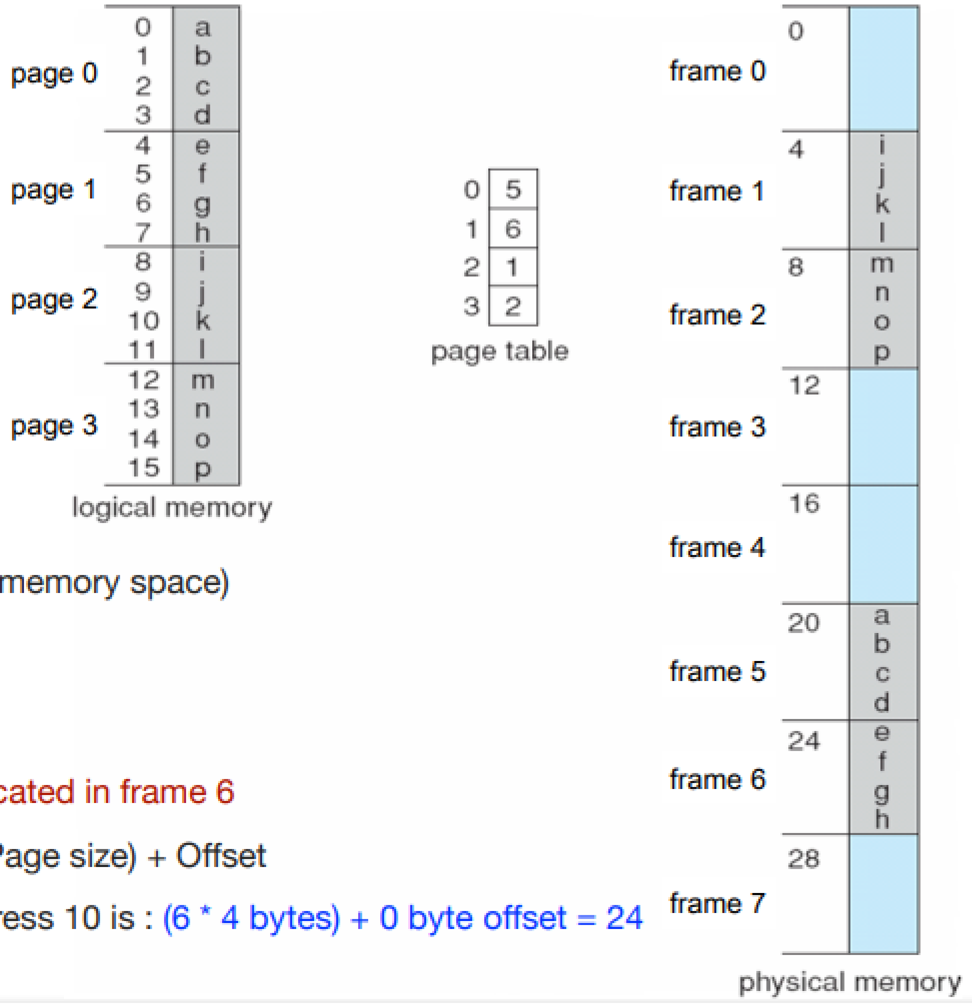




- **Example setup:**
  - n=2 and m=4 (*p* and *d* are each 2-bits)
  - 32-bytes of physical memory and
  - 4-byte pages (8 pages can fit in physical memory space)
- **Example:**
  - Logical address 10 is in page 2 at offset 2
  - According to the page table, page 2 is located in frame 1
  - Physical memory address is: (Frame # * Page size) + Offset
  - Physical memory address for logical address 10 is : (1 * 4 bytes) + 2 byte offset = 6

Paging Example





- **Example setup:**
  - n=2 and m=4 (*p* and *d* are each 2-bits)
  - 32-bytes of physical memory and
  - 4-byte pages (8 pages can fit in physical memory space)
- **Example:**
  - Logical address 4 is in page 1 at offset 0
  - According to the page table, page 1 is located in frame 6
  - Physical memory address is: (Frame # * Page size) + Offset
  - Physical memory address for logical address 10 is : (6 * 4 bytes) + 0 byte offset = 24

Paging

- To make this paging scheme convenient, let us dictate that the page size, hence the frame size, must be a power of 2.
- Relative address, which is defined with reference to the origin of the program and the logical address, expressed as a page number and offset, are the same.
- **Example:.** 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 2^6 (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.

Logical Addresses

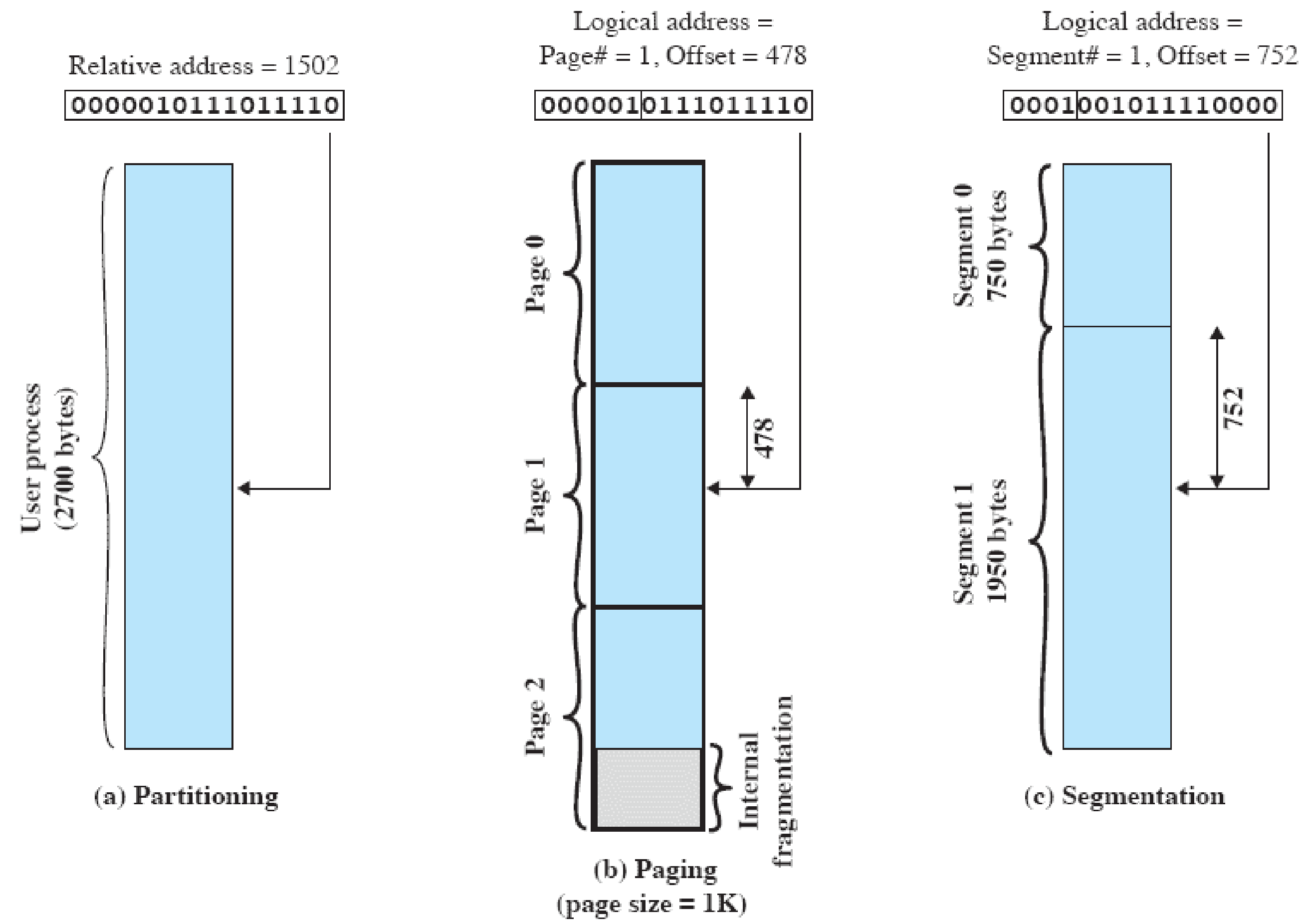


Figure 7.11 Logical Addresses

Simple Partition vs Paging

- simple paging is similar to fixed partitioning.
- With paging, the partitions are rather small;
  - a program may occupy more than one partition;
- With Paging, these partitions need not be contiguous.

Paging

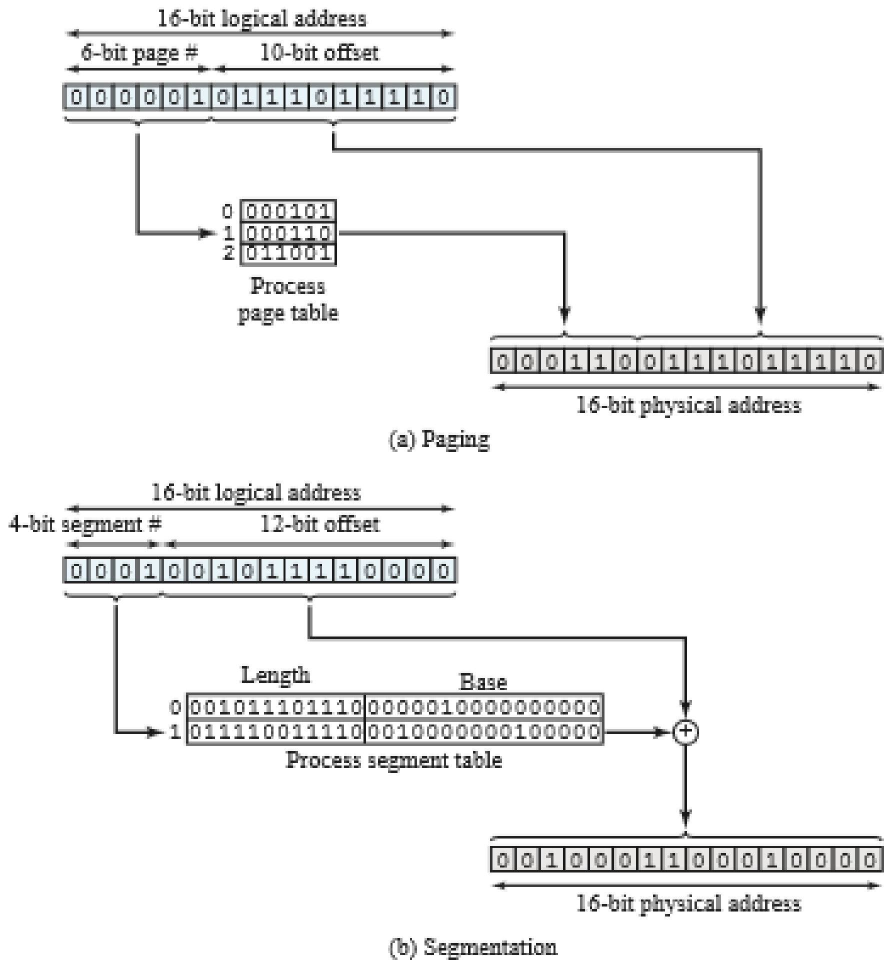
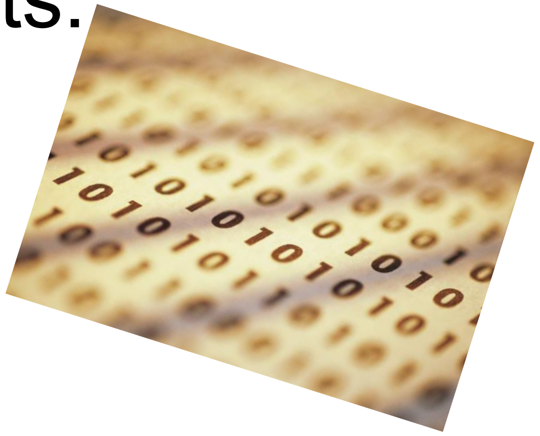


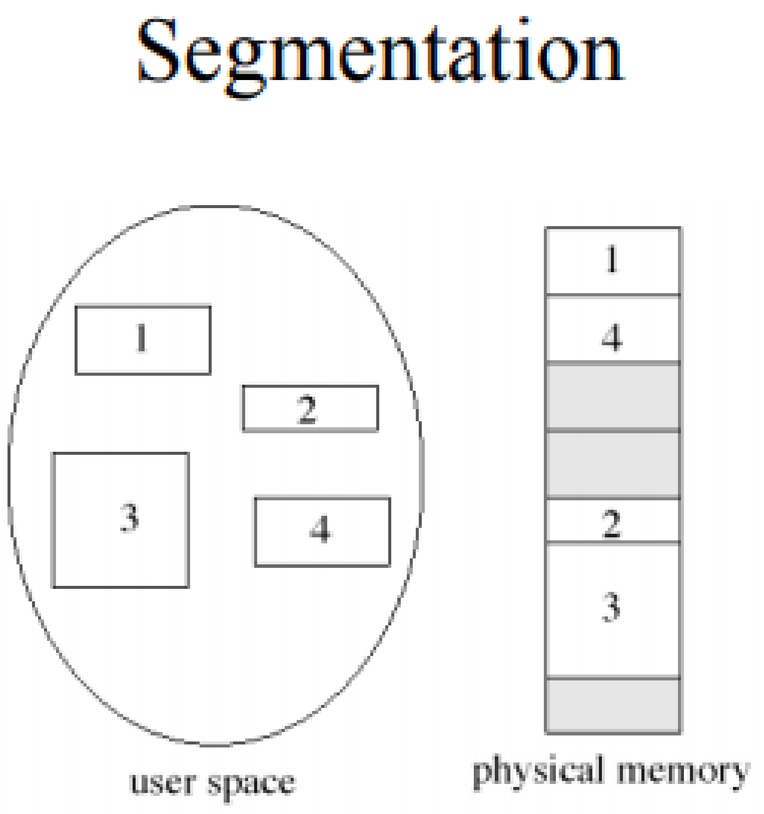
Figure 7.12 Examples of Logical-to Physical Address Translation

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



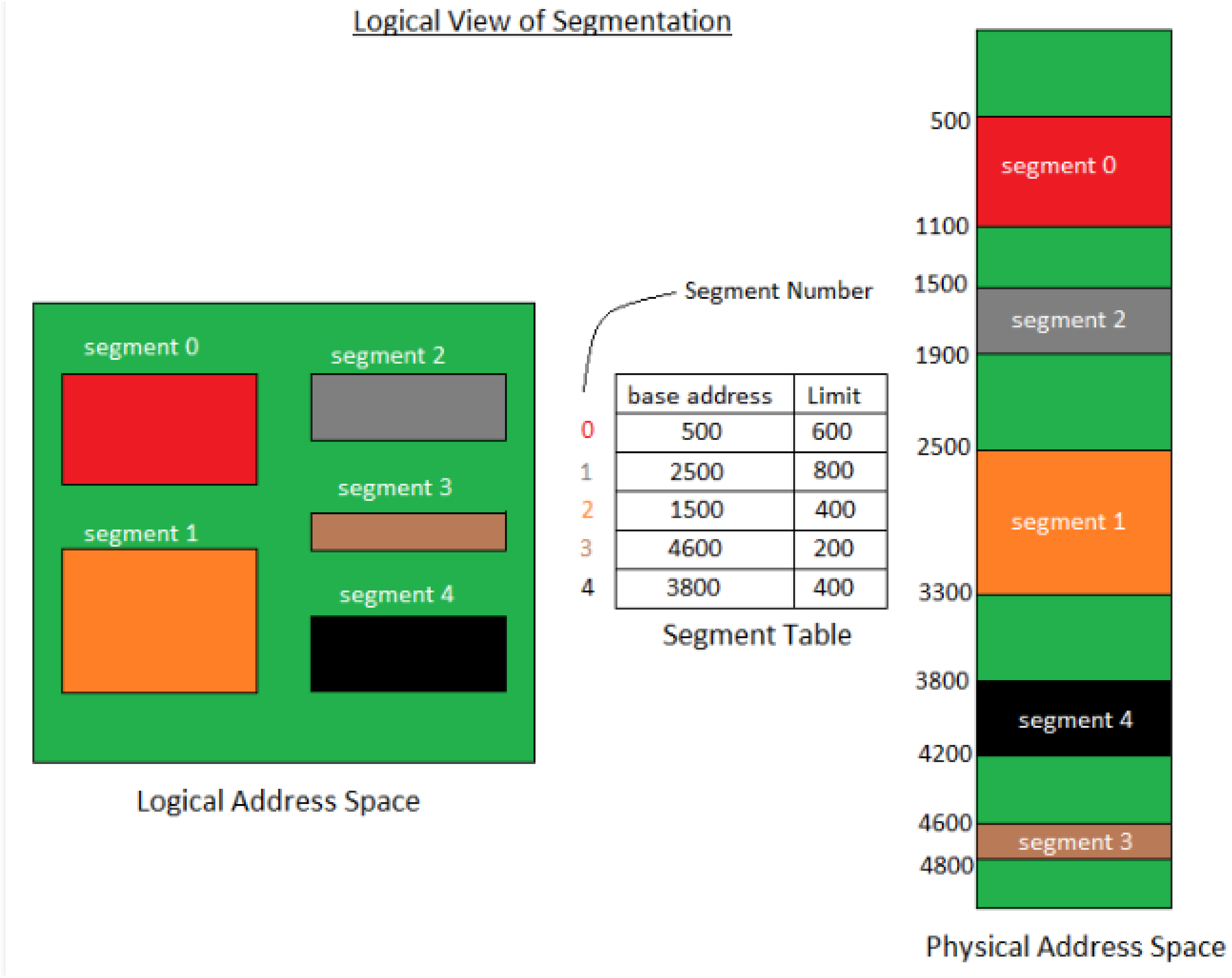
Segmentation

Segment table contains

- **limit**
  - logical addresses must fall within the range 0..limit • error if it doesn’t (out of range) • provides memory access protection
- **–base**
  - added to offset to find physical address

Silberschatz and Galvin ©1999

Segmentation

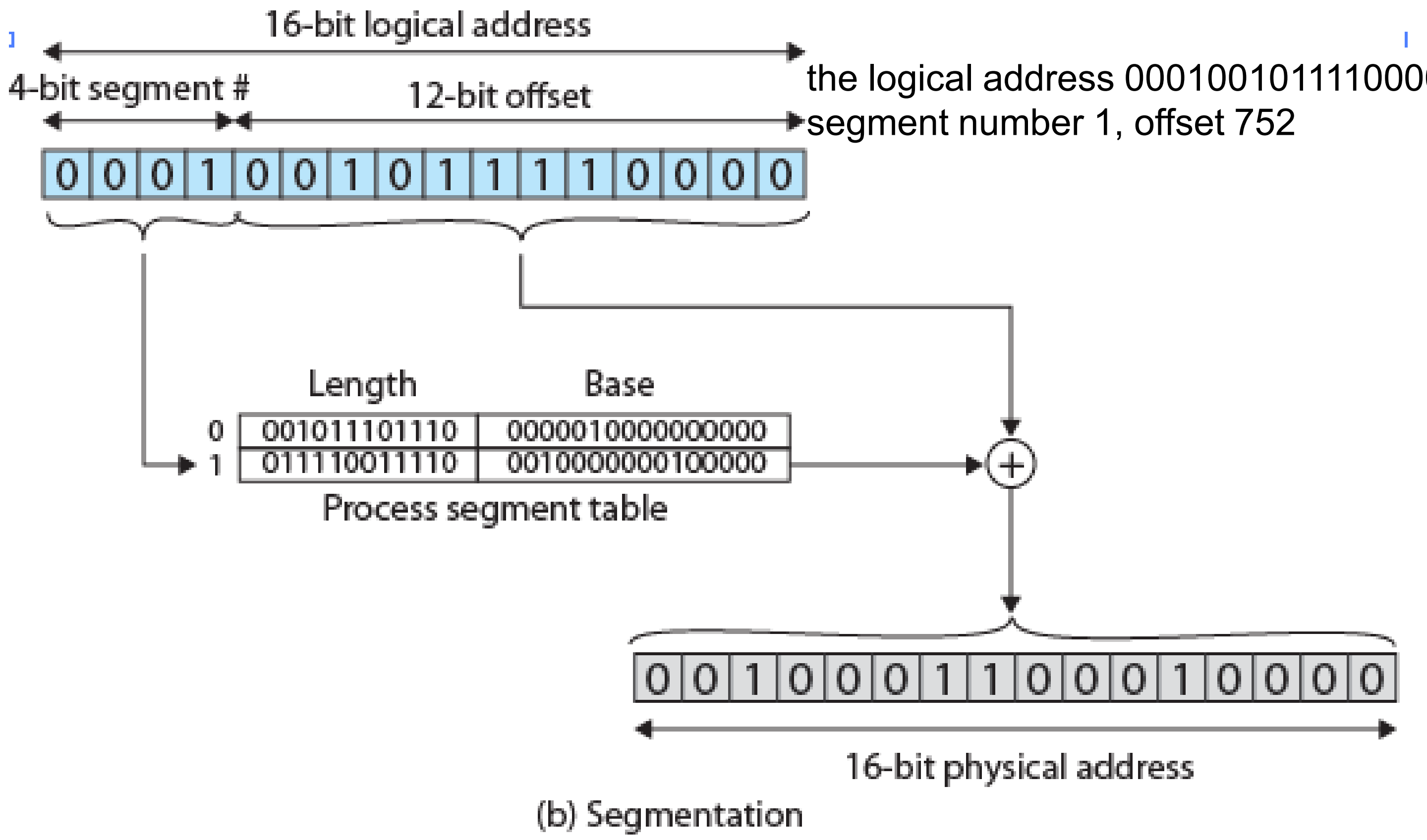


Segment Table

	base address	Limit
0	500	600
1	2500	800
2	1500	400
3	4600	200
4	3800	400

Logical-to-Physical Address

Translation - Segmentation



(b) Segmentation

the logical address 0001001011110000, which is segment number 1, offset 752

The physical address is 0010000000100000 + 001011110000= 0010001100010000

Silberschatz and Galvin ©1999

Steps for Address Translation

The following steps are needed for address translation:

- Extract the segment number as the leftmost n bits of the logical address.
- Use the segment number as an index into the process segment table to find the starting physical address of the segment.
- 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.
- The desired physical address is the sum of the starting physical address of the segment plus the offset.

Problem example

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

	Starting Address	Length (bytes)
0	660	248
1	1752	422
2	222	198
3	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

Segmentation

Advantages:

– No Internal fragmentation.

– Segment Table consumes less space in comparison to Page table in paging.

Disadvantage:

– As processes are loaded and removed from the memory, the free memory space is broken into little pieces, causing External fragmentation.

Summary

- Memory Management
  - one of the most important and complex tasks of an operating system
  - needs to be treated as a resource to be allocated to and shared among a number of active processes
  - desirable to maintain as many processes in main memory as possible
  - desirable to free programmers from size restriction in program development
  - basic tools are paging and segmentation (possible to combine)
    - » paging – small fixed-sized pages
    - » segmentation – pieces of varying size

End of Chapter 7