Chapter 7
Memory Management

(based on original slides by Pearson)

Memory Management

- Subdividing memory to accommodate multiple processes
- Memory needs to be allocated to ensure a reasonable supply of ready processes to consume available processor time

Memory Management Requirements

- Relocation
  - Programmer does not know where the program will be placed in memory when it is executed
  - While the program is executing, it may be swapped to disk and returned to main memory at a different location (relocated)
  - Memory references must be translated in the code to actual physical memory address

Addressing Requirement

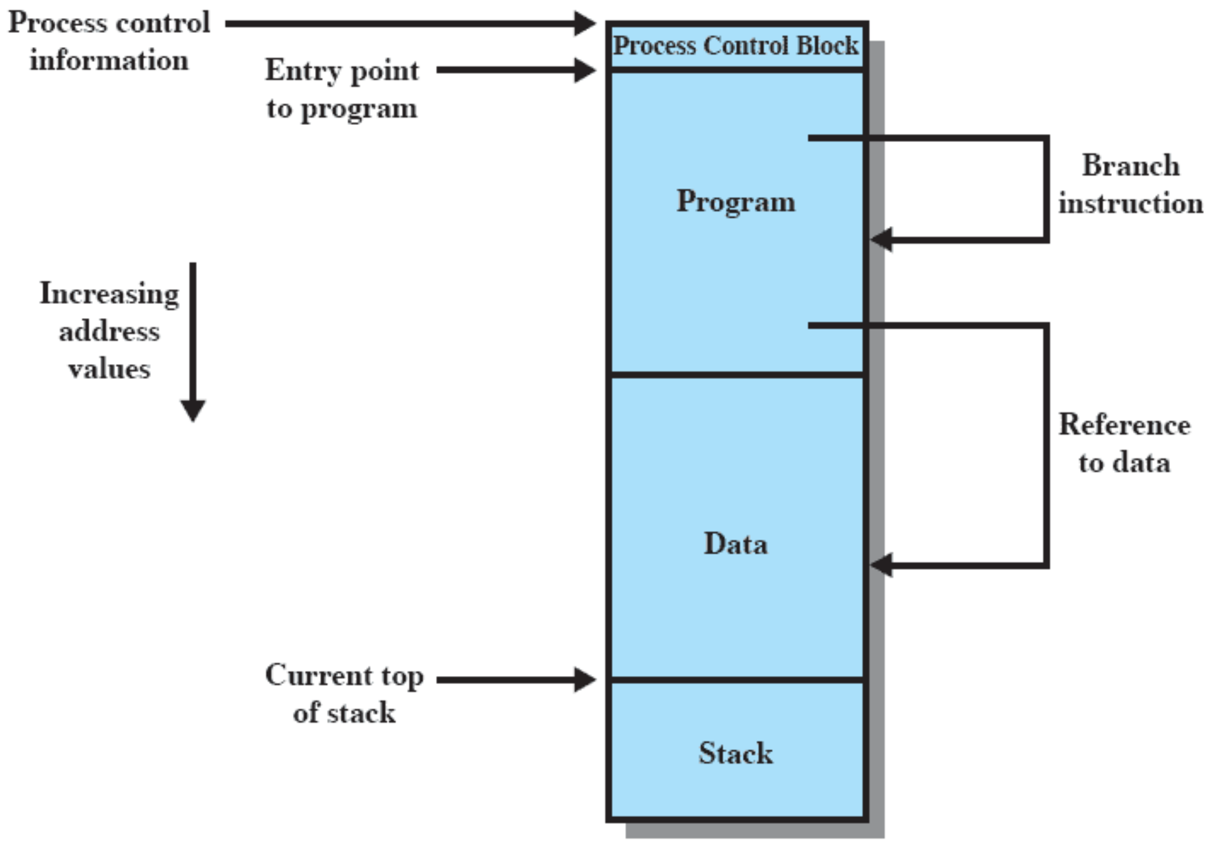


Figure 7.1 Addressing Requirements for a Process

Memory Management Requirements

- Protection
  - Processes should not be able to reference memory locations in another process without permission
  - Impossible to check absolute addresses at compile time
    => Must be checked at run time
    => see your ‘segmentation fault’ bugs

Memory Management Requirements

- Sharing
  - Allow several processes to access the same portion of memory
  - Better to allow each process access to the same copy of the program rather than have their own separate copy

Memory Management Requirements

- Logical Organization
  - Programs are written in modules
  - Modules can be written and compiled independently
  - Different degrees of protection given to modules (read-only, execute-only)
  - Share modules among processes

Memory Management Requirements

- Physical Organization
  - Memory available for a program plus its data may be insufficient
    - Overlaying allows various modules to be assigned the same region of memory
  - Programmer does not know how much space will be available

Fixed Partitioning

- Equal-size partitions
  - Any process whose size is less than or equal to the partition size can be loaded into an available partition
  - If all partitions are full, the operating system can swap a process out of a partition

Fixed Partitioning

- Equal-size partitions
  - A program may not fit in a partition. The programmer must design the program with overlays
  - Main memory use is inefficient. Any program, no matter how small, occupies an entire partition.
    - This is called internal fragmentation.

Fixed Partitioning

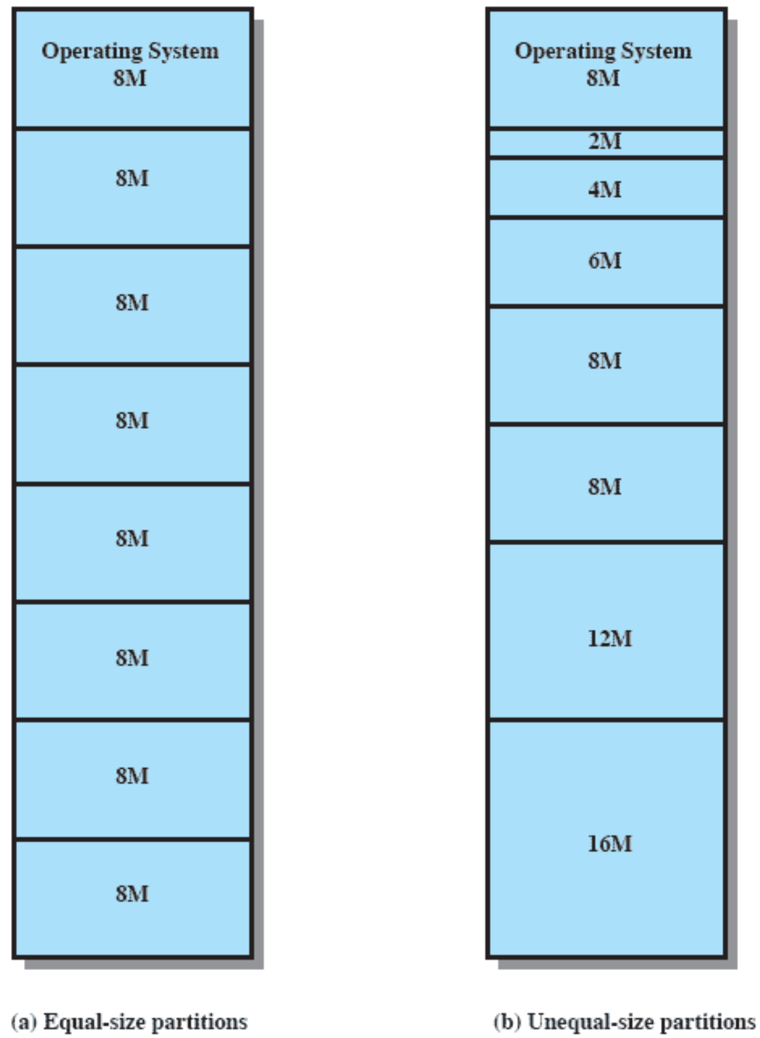


Figure 7.2 Example of Fixed Partitioning of a 64-Mbyte Memory

Placement Algorithm

- Equal-size
  – Placement it trivial
- Unequal-size
  – Can assign each process to the smallest partition within which it will fit
  – Queue for each partition
  – Processes are assigned in such a way as to minimize wasted memory within a partition

Fixed Partitioning

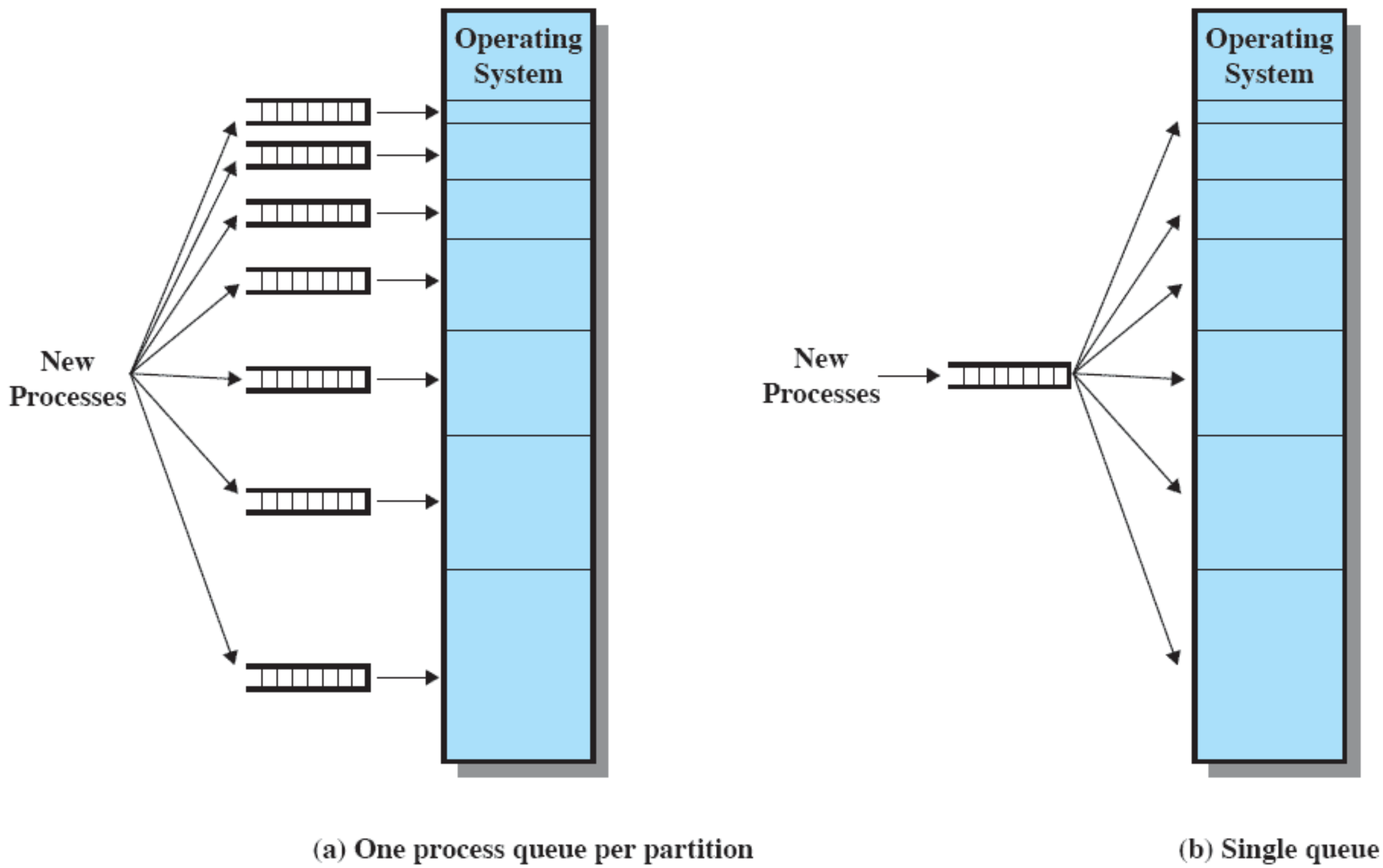
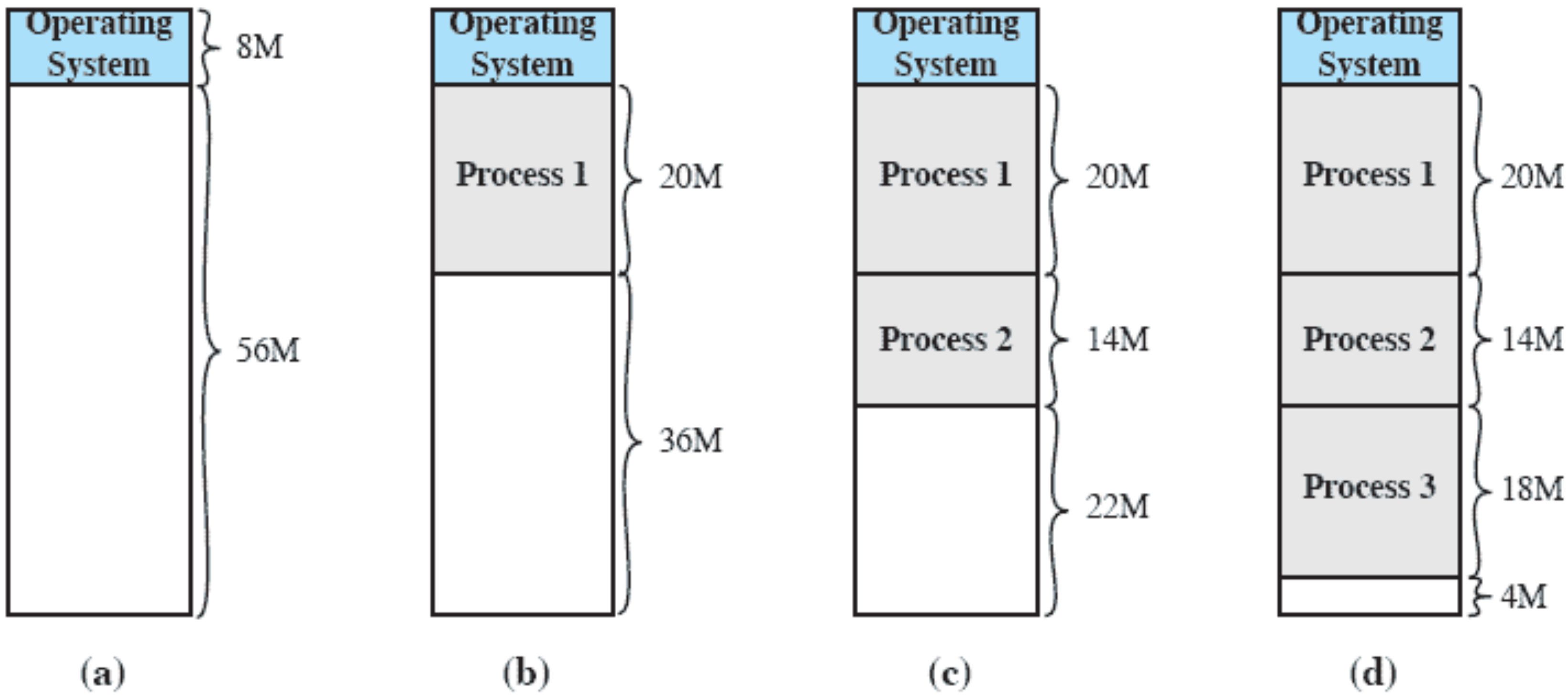


Figure 7.3 Memory Assignment for Fixed Partitioning

Dynamic Partitioning

- Partitions are of variable length and number
- Process is allocated exactly as much memory as required
- Eventually get holes in the memory. This is called external fragmentation
- Must use compaction to shift processes so they are contiguous and all free memory is in one block

Dynamic Partitioning



Dynamic Partitioning

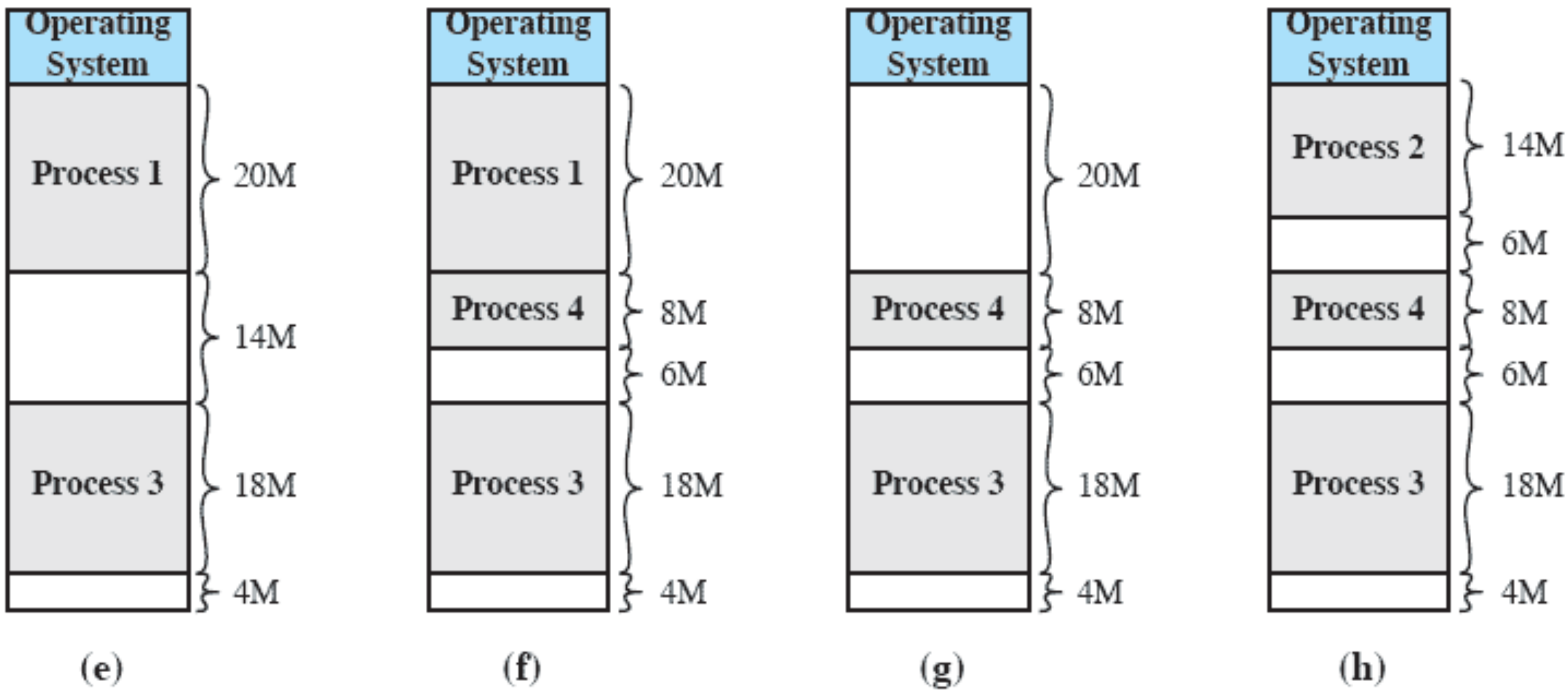


Figure 7.4 The Effect of Dynamic Partitioning

Dynamic Partitioning

- Operating system must decide which free block to allocate to a process
- Best-fit algorithm
  - Chooses the block that is closest in size to the request
  - Worst performer overall
  - Since smallest block is found for process, the smallest amount of fragmentation is left
  - Memory compaction must be done more often

Dynamic Partitioning

- First-fit algorithm
  - Scans memory from the beginning and chooses the first available block that is large enough
  - Fastest
  - May have many process loaded in the front end of memory that must be searched over when trying to find a free block

Dynamic Partitioning

- Next-fit
  - Scans memory from the location of the last placement
  - More often allocates a block of memory at the end of memory where the largest block is found
  - The largest block of memory is broken up into smaller blocks
  - Compaction is required to obtain a large block at the end of memory

Allocation

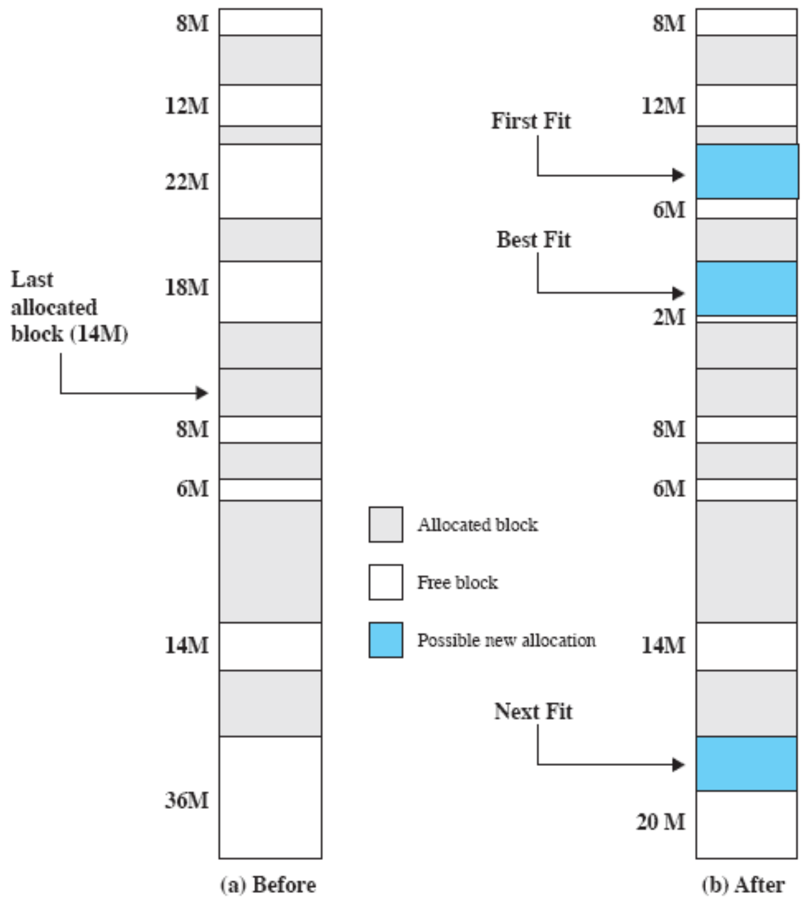


Figure 7.5 Example Memory Configuration before and after Allocation of 16-Mbyte Block

Buddy System

- Entire space available is treated as a single block of 2U
- If a request of size s such that 2U-1 < s <= 2U, entire block is allocated
  - Otherwise block is split into two equal buddies
  - Process continues until smallest block greater than or equal to s is generated

Example of Buddy System

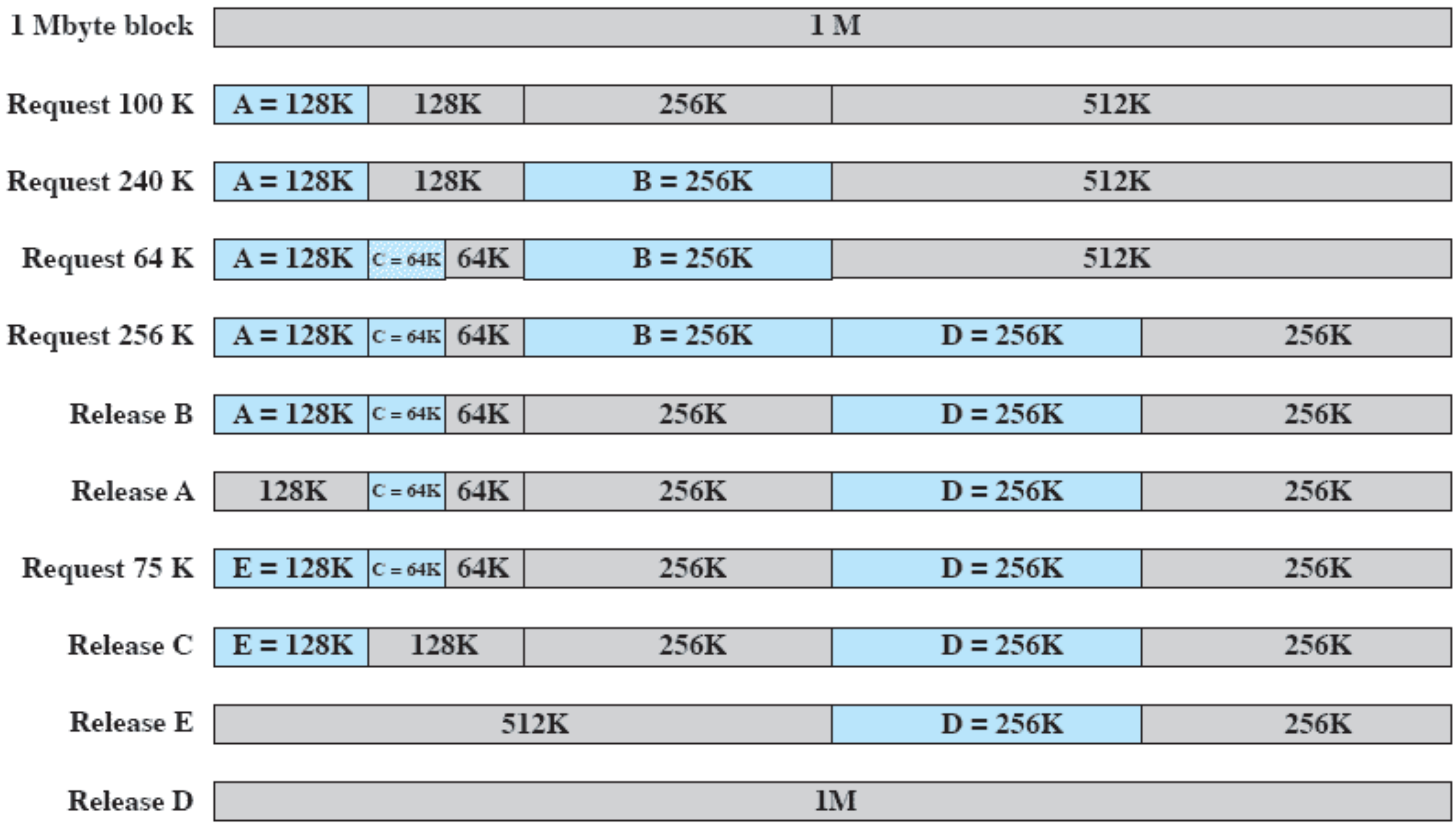


Figure 7.6 Example of Buddy System

Free Representation of Buddy System

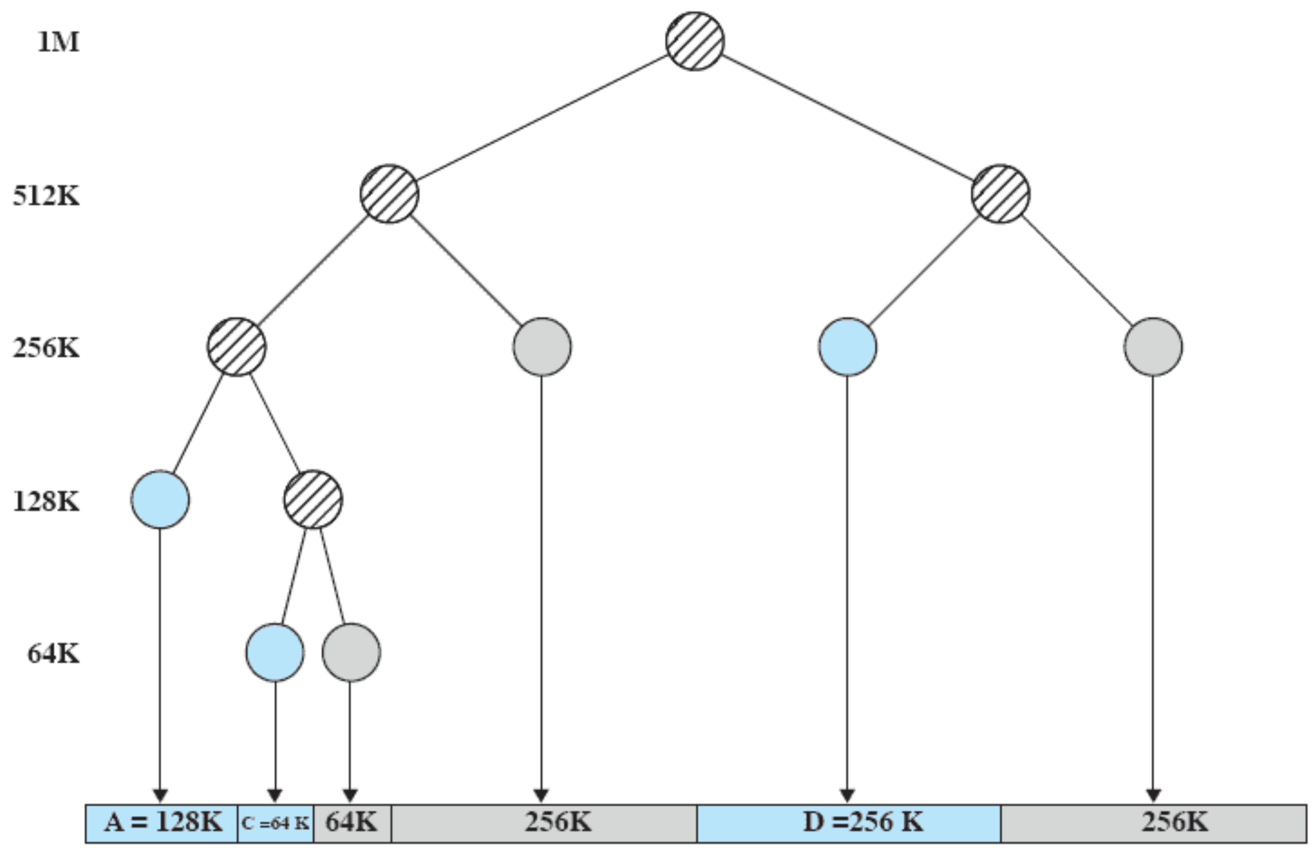


Figure 7.7 Tree Representation of Buddy System

Relocation

- When program is loaded into memory, the actual (absolute) memory locations are determined
- A process may occupy different partitions, which means different absolute memory locations during execution (from swapping)

Relocation

- Compaction will also cause a program to occupy a different partition, which means different absolute memory locations

Addresses

- Logical
  - Reference to a memory location independent of the current assignment of data to memory
  - Translation must be made to the physical address
- Relative
  - Address expressed as a location relative to some known point

Addresses

- Physical
  - The absolute address or actual location in main memory

Relocation

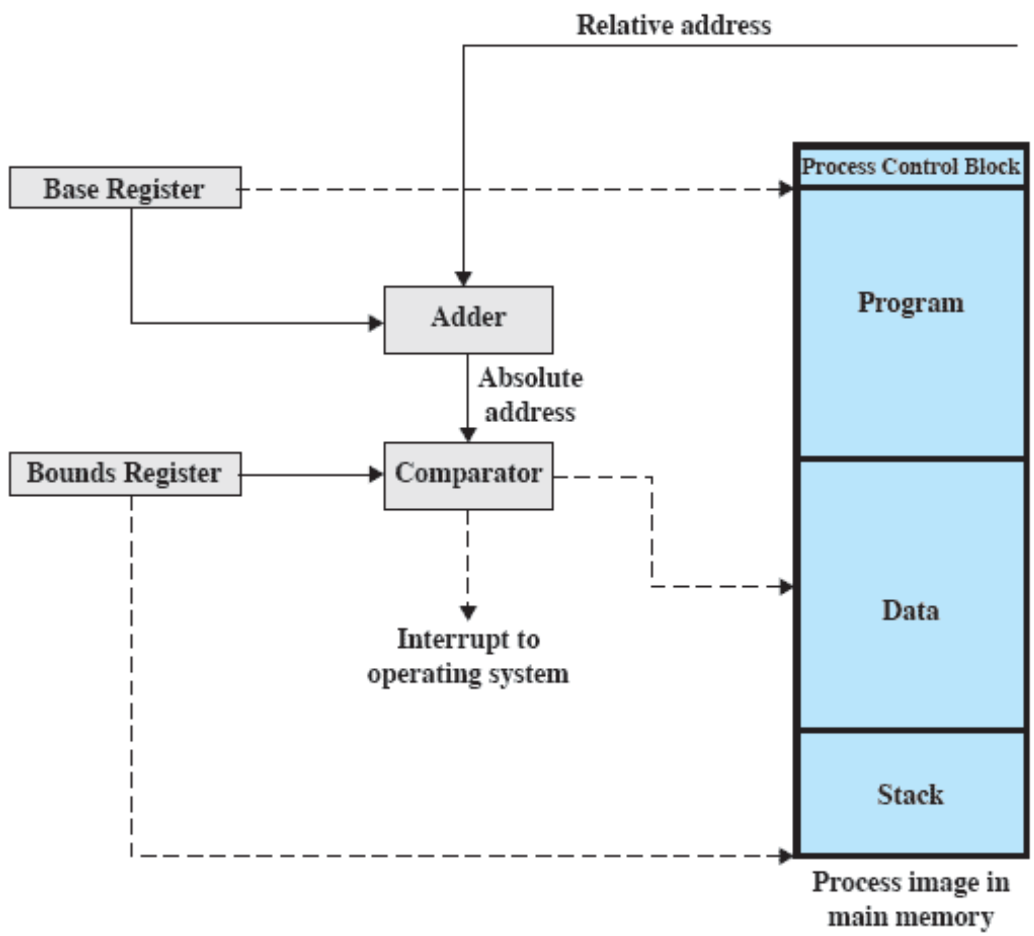


Figure 7.8 Hardware Support for Relocation

Question

Registers Used during Execution

- Base register
  - Starting address for the process
- Bounds register
  - Ending location of the process
- These values are set when the process is loaded or when the process is swapped in

Registers Used during Execution

- The value of the base register is added to a relative address to produce an absolute address
- The resulting address is compared with the value in the bounds register
- If the address is not within bounds, an interrupt is generated to the operating system

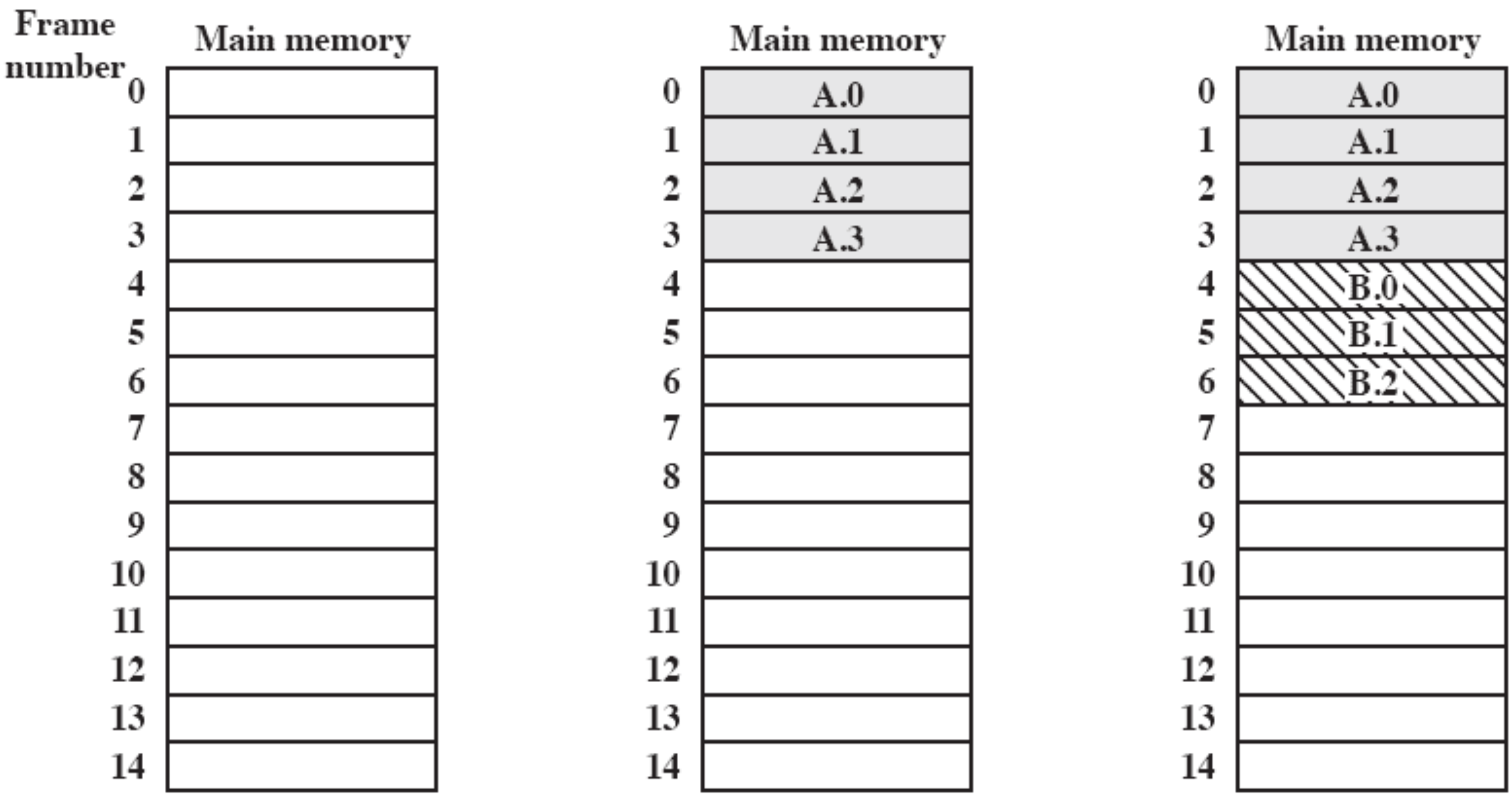
Paging

- Partition memory into small equal fixed-size chunks and divide each process into the same size chunks
- The chunks of a process are called pages and chunks of memory are called frames
- Base address pointer no longer sufficient

Paging

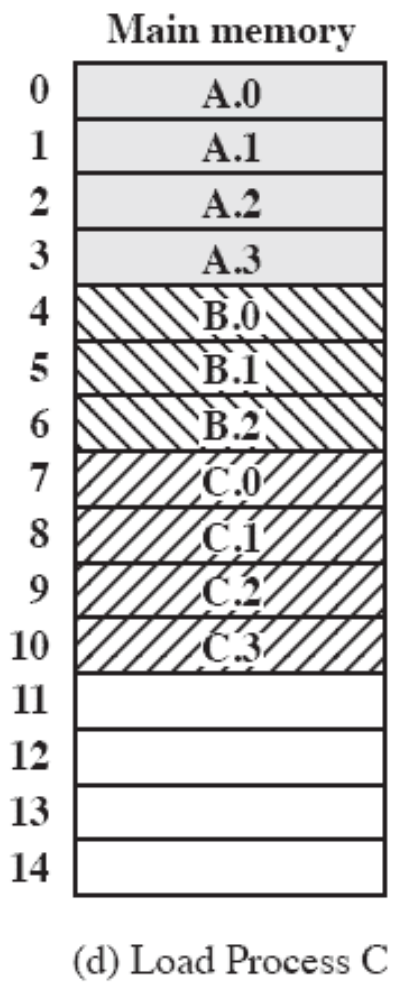
- Operating system maintains a page table for each process
  - Contains the frame location for each page in the process
  - Memory address consist of a page number and offset within the page
- Logical-to-physical is still done by hardware
  - Logical address: (page, offset)
  - Physical address: (frame, offset)

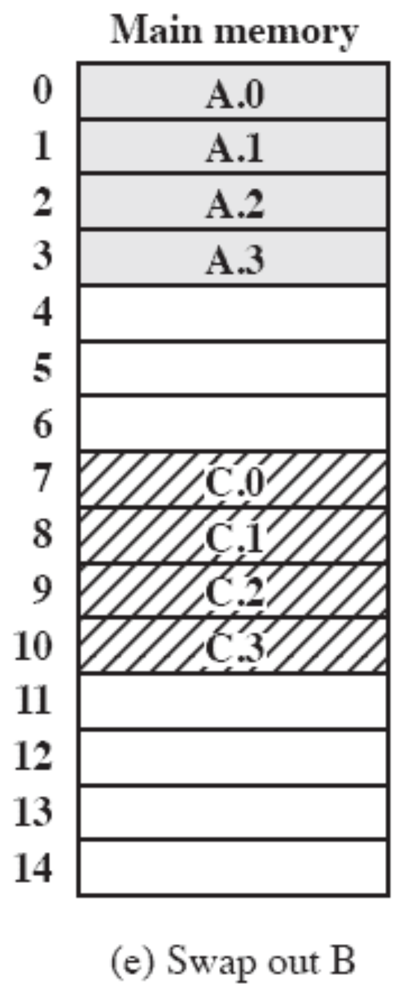
Process and Frames



(a) Fifteen Available Frames (b) Load Process A (c) Load Process B

Process and Frames





Can we now allocate 5 frames for Process D?

Figure 7.9 Assignment of Process Pages to Free Frames

Page Table

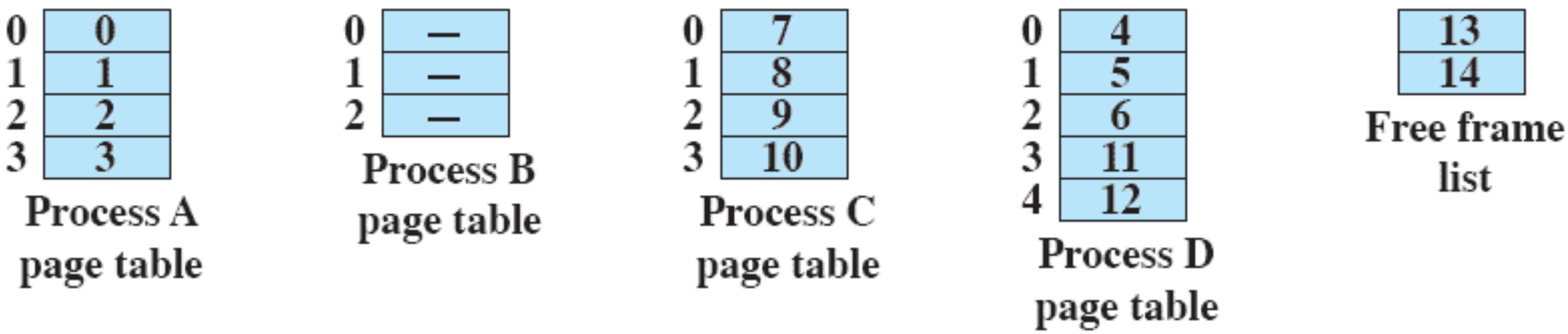


Figure 7.10 Data Structures for the Example of Figure 7.9 at Time Epoch (f)

Segmentation

- All segments of all programs do not have to be of the same length
- There is a maximum segment length
- Addressing consist of two parts - a segment number and an offset
- Since segments are not equal, segmentation is similar to dynamic partitioning
  - But program can have more segments

Logical Addresses

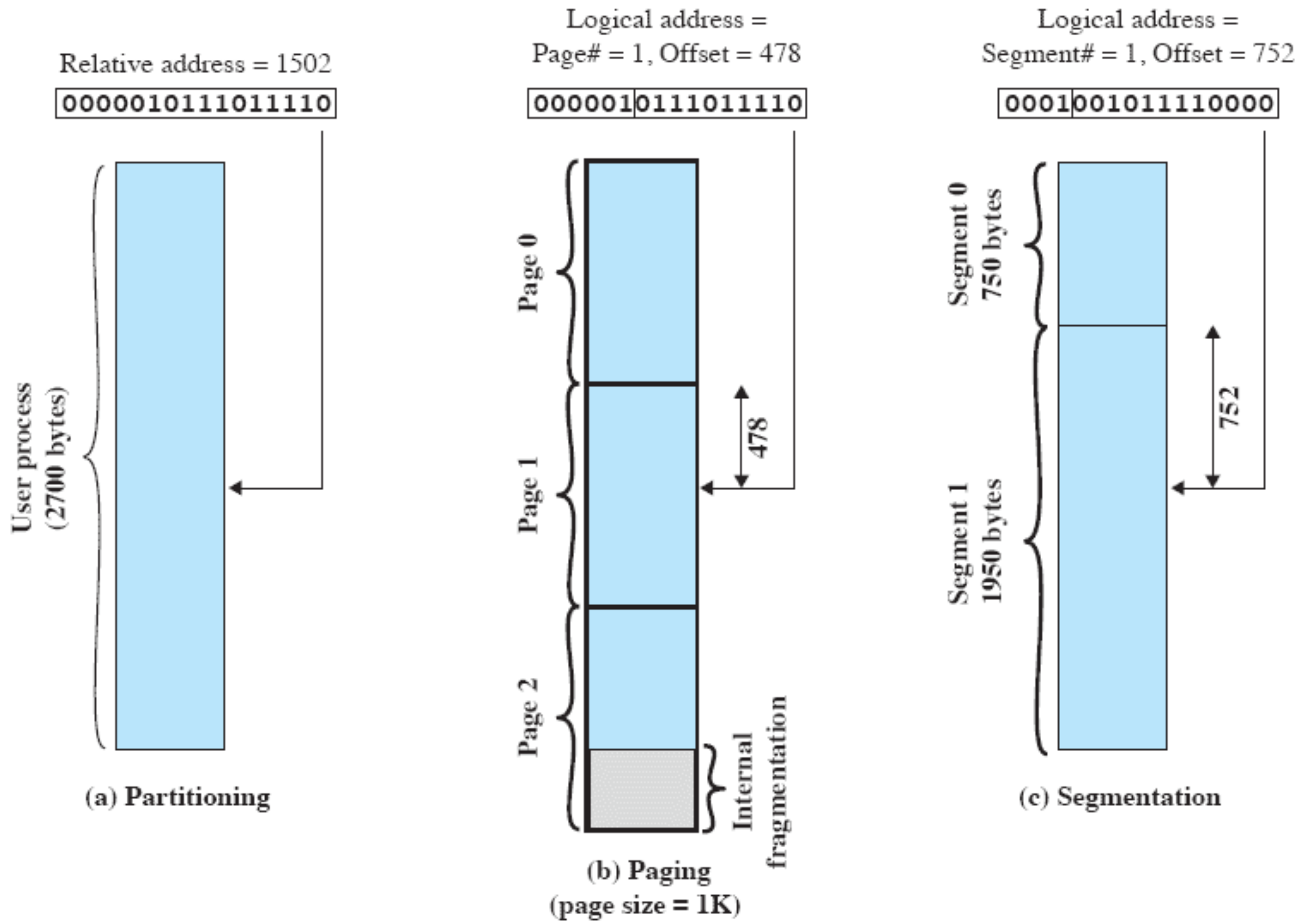
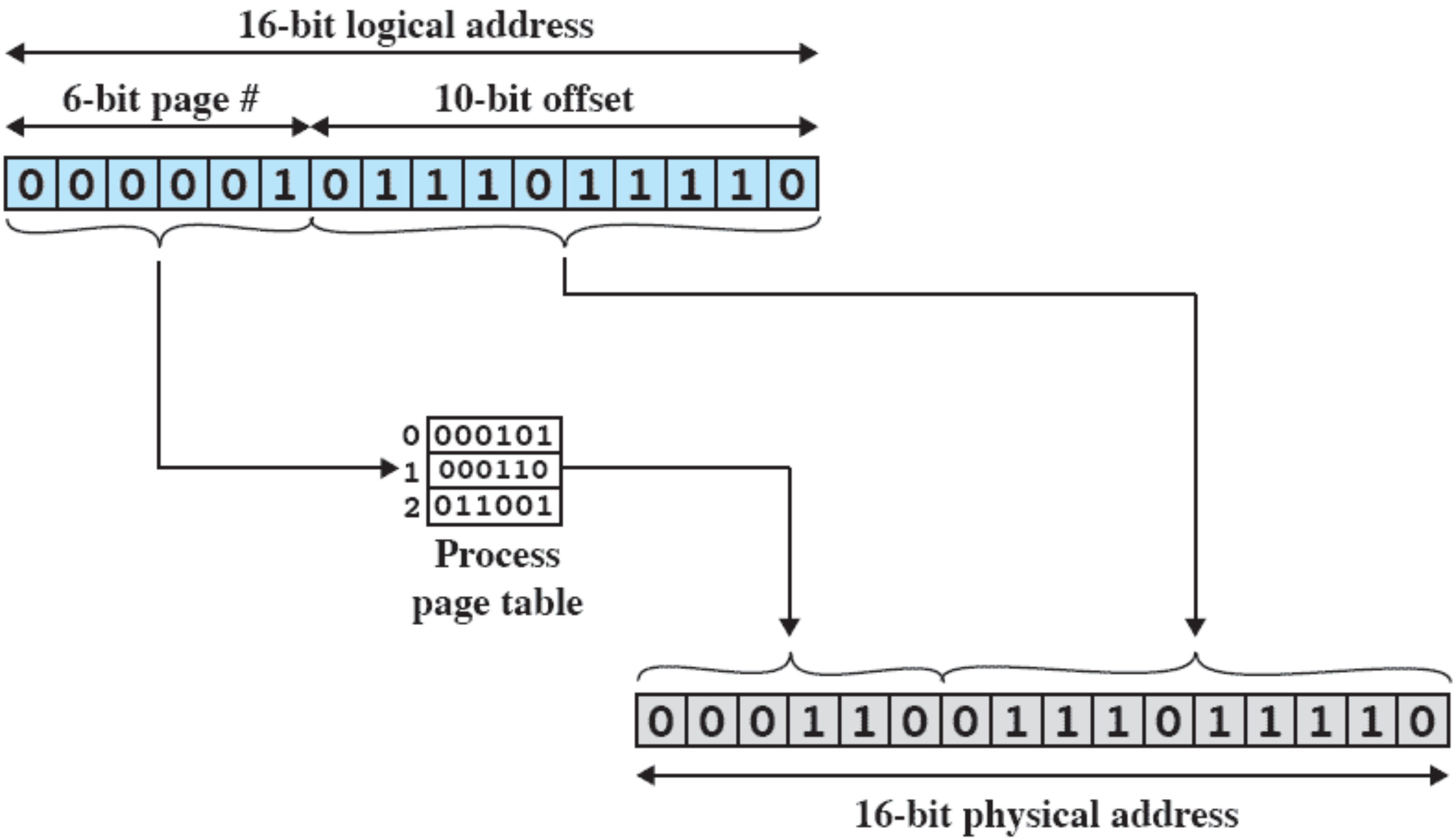


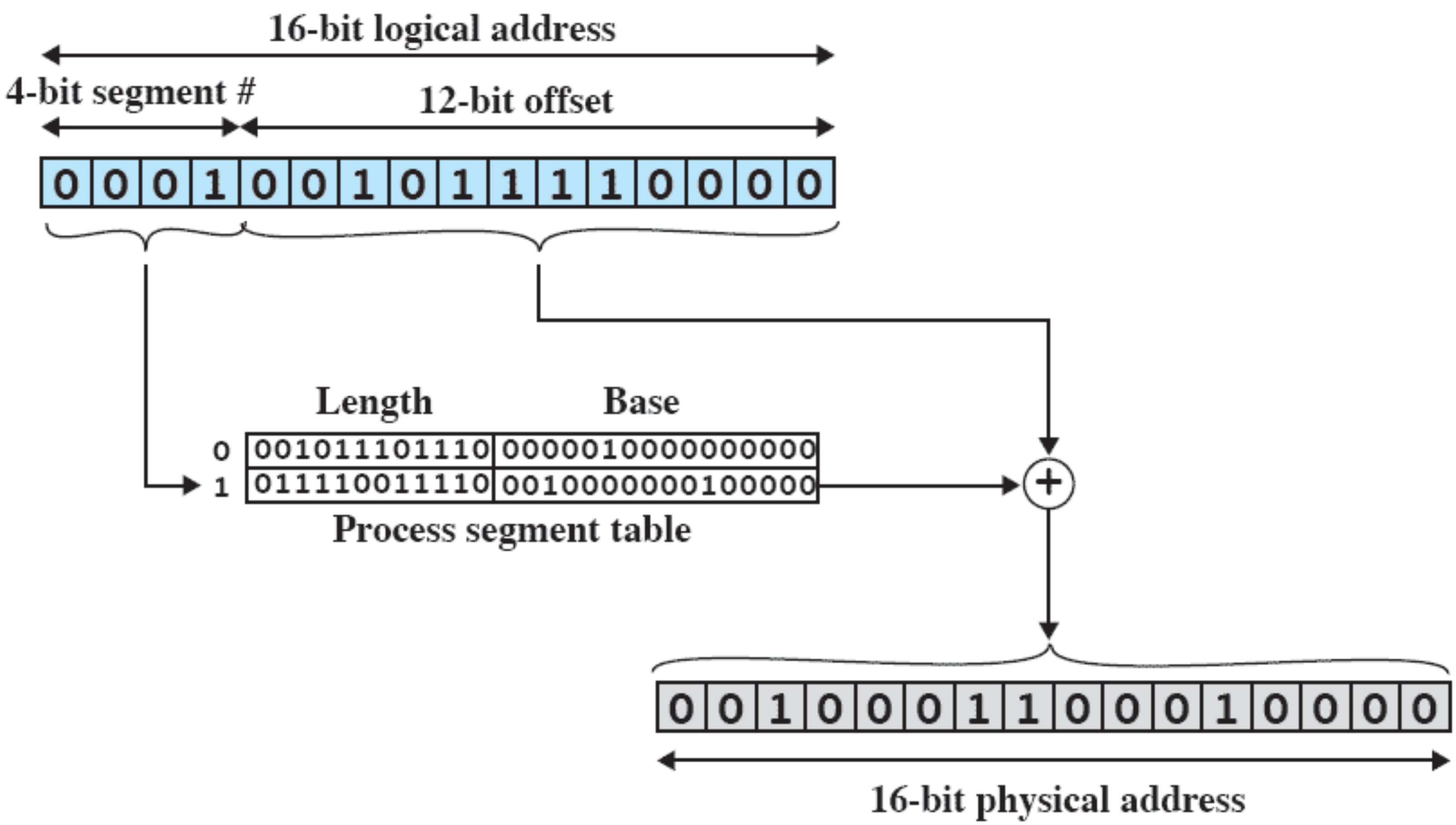
Figure 7.11 Logical Addresses

Paging



(a) Paging

Segmentation



(b) Segmentation

Figure 7.12 Examples of Logical-to-Physical Address Translation

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.