Chapter 9 – Real Memory Organization and Management

Outline

Objectives

- After reading this chapter, you should understand:
  - the need for real (also called physical) memory management.
  - the memory hierarchy.
  - contiguous and noncontiguous memory allocation.
  - fixed- and variable-partition multiprogramming.
  - memory swapping.
  - memory placement strategies.

9.1 Introduction

- Memory divided into tiers
  - Main memory
    - Relatively expensive
    - Relatively small capacity
    - High-performance
  - Secondary storage
    - Cheap
    - Large capacity
    - Slow
  - Main memory requires careful management

9.2 Memory Organization

- Memory can be organized in different ways
  - One process uses entire memory space
  - Each process gets its own partition in memory
    - Dynamically or statically allocated
- Trend: Application memory requirements tend to increase over time to fill main memory capacities

9.2 Memory Organization

Figure 9.1 Microsoft Windows operating system memory requirements.

Operating System	Release Date	Minimum Memory Requirement	Recommended Memory
Windows 1.0	November 1985	256KB	
Windows 2.03	November 1987	320KB	
Windows 3.0	March 1990	896KB	1MB
Windows 3.1	April 1992	2.6MB	4MB
Windows 95	August 1995	8MB	16MB
Windows NT 4.0	August 1996	32MB	96MB
Windows 98	June 1998	24MB	64MB
Windows ME	September 2000	32MB	128MB
Windows 2000 Professional	February 2000	64MB	128MB
Windows XP Home	October 2001	64MB	128MB
Windows XP Professional	October 2001	128MB	256MB

9.3 Memory Management

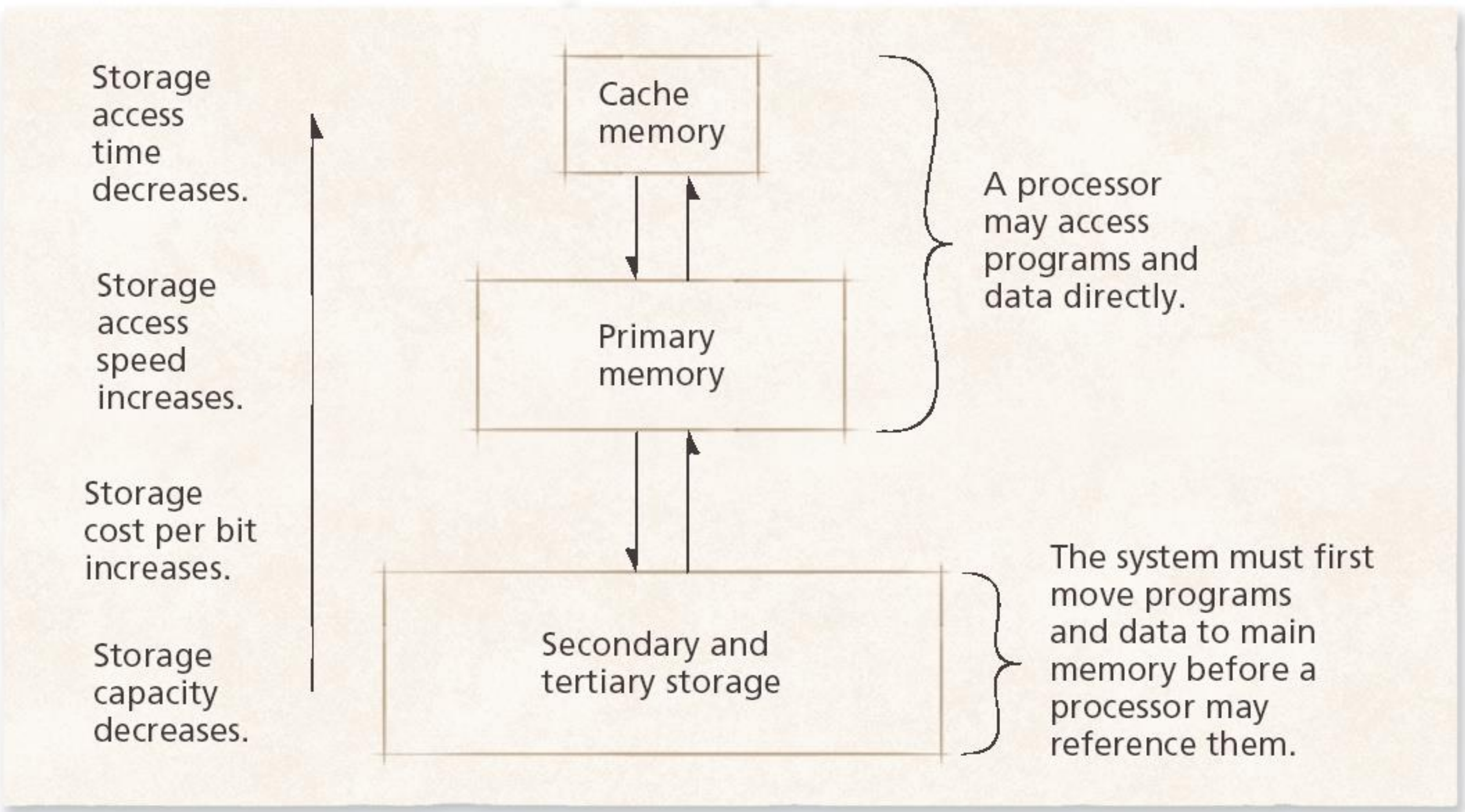
- Strategies for obtaining optimal memory performance
  - Performed by memory manager
    - Which process will stay in memory?
    - How much memory will each process have access to?
    - Where in memory will each process go?

9.4 Memory Hierarchy

- Main memory
  - Should store currently needed program instructions and data only
- Secondary storage
  - Stores data and programs that are not actively needed
- Cache memory
  - Extremely high speed
  - Usually located on processor itself
  - Most-commonly-used data copied to cache for faster access
  - Small amount of cache still effective for boosting performance
    - Due to temporal locality

9.4 Memory Hierarchy

Figure 9.2 Hierarchical memory organization.



9.5 Memory Management Strategies

- Strategies divided into several categories
  - Fetch strategies
    - Demand or anticipatory
    - Decides which piece of data to load next
  - Placement strategies
    - Decides where in main memory to place incoming data
  - Replacement strategies
    - Decides which data to remove from main memory to make more space

9.6 Contiguous vs. Noncontiguous Memory Allocation

- Ways of organizing programs in memory
  - Contiguous allocation
    - Program must exist as a single block of contiguous addresses
    - Sometimes it is impossible to find a large enough block
    - Low overhead
  - Noncontiguous allocation
    - Program divided into chunks called segments
    - Each segment can be placed in different part of memory
    - Easier to find “holes” in which a segment will fit
    - Increased number of processes that can exist simultaneously in memory offsets the overhead incurred by this technique

9.7 Memory Allocation Schemes/Methods

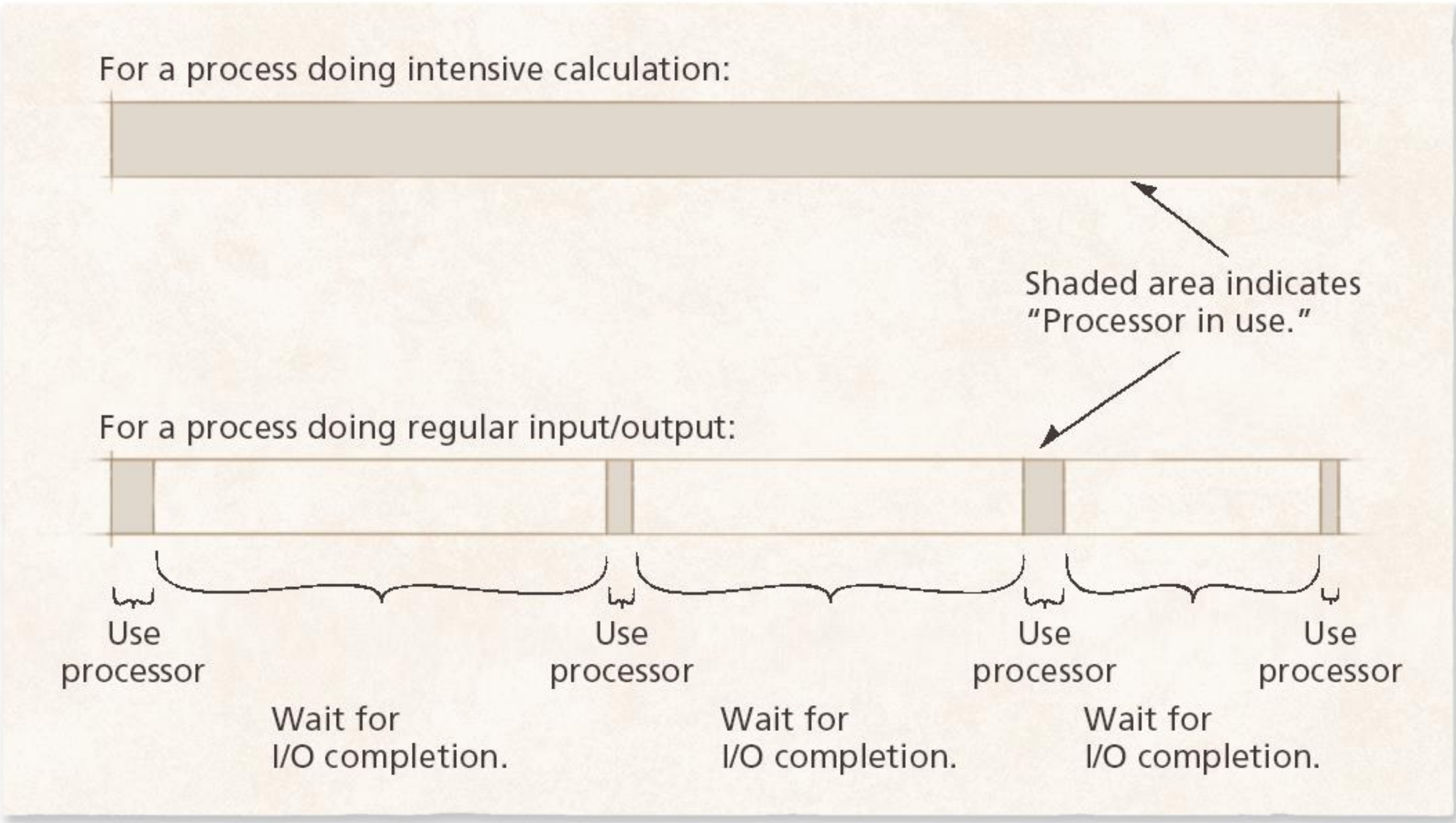
- Swapping
- Memory Allocation Methods
  - Single Fixed Partitioning/Single Absolute & Relocatable Partition
  - Multiple Fixed Partitioning (Internal Fragmentation)
  - Variable Partitioning (External Fragmentation)
    - Coalescing Holes
    - Compaction

9.8 Fixed-Partition Multiprogramming

- I/O requests can tie up a processor for long periods
  - Multiprogramming is one solution
    - Process not actively using a processor should relinquish (handover)it to others
    - Requires several processes to be in memory at once

9.8 Fixed-Partition Multiprogramming

Figure 9.6 Processor utilization on a single-user system. [Note: In many single-user jobs, I/O waits are much longer relative to processor utilization periods indicated in this diagram.]

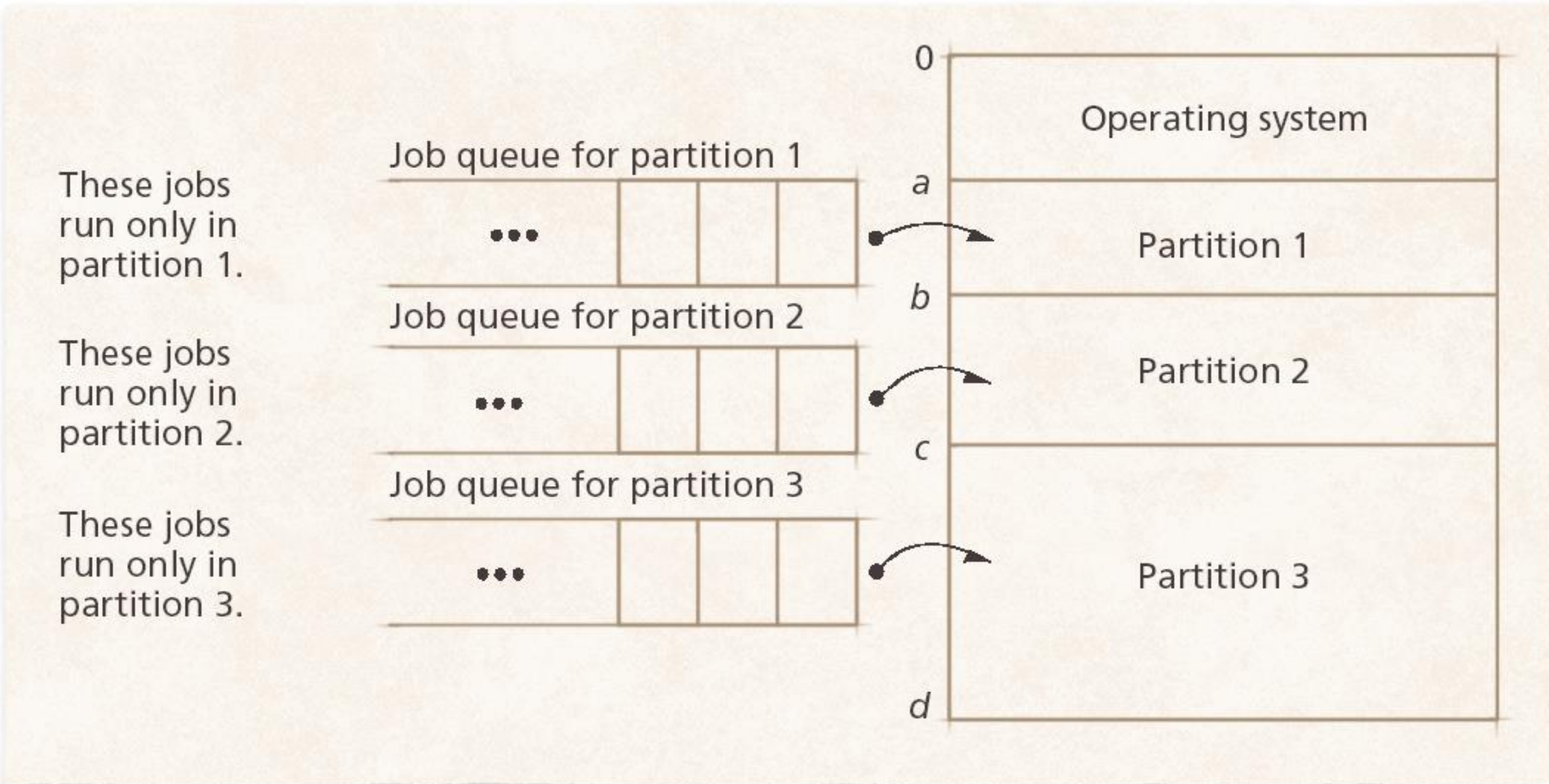


9.8 Fixed-Partition Multiprogramming

- Fixed-partition multiprogramming
  - Each active process receives a fixed-size block of memory
  - Processor rapidly switches between each process
  - Multiple boundary registers protect against damage

9.8 Fixed-Partition Multiprogramming

Figure 9.7 Fixed-partition multiprogramming with absolute translation and loading.

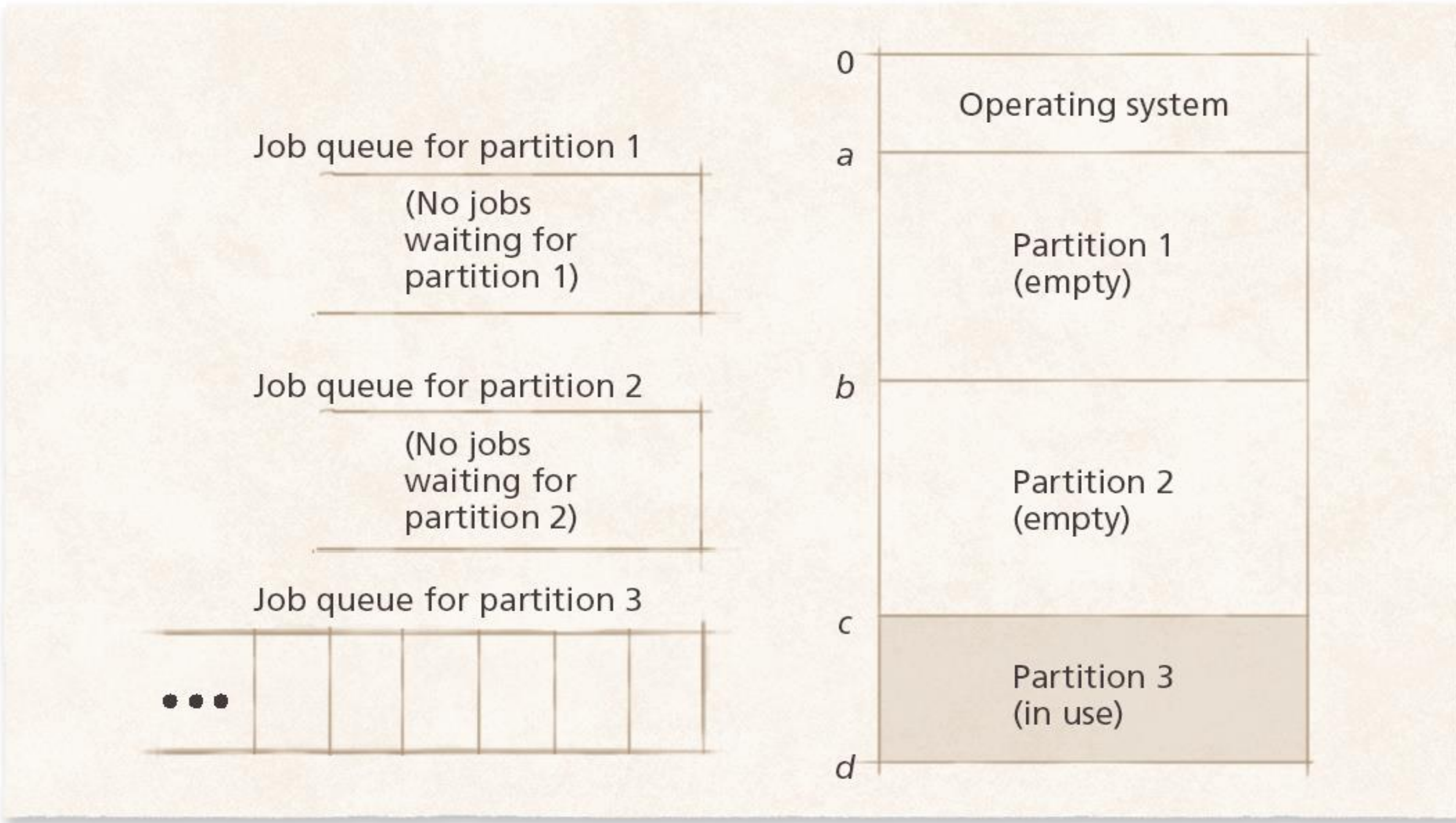


9.8 Fixed-Partition Multiprogramming

- Drawbacks to fixed partitions
  - Early implementations used absolute addresses
    - If the requested partition was full, code could not load
    - Later resolved by relocating compilers

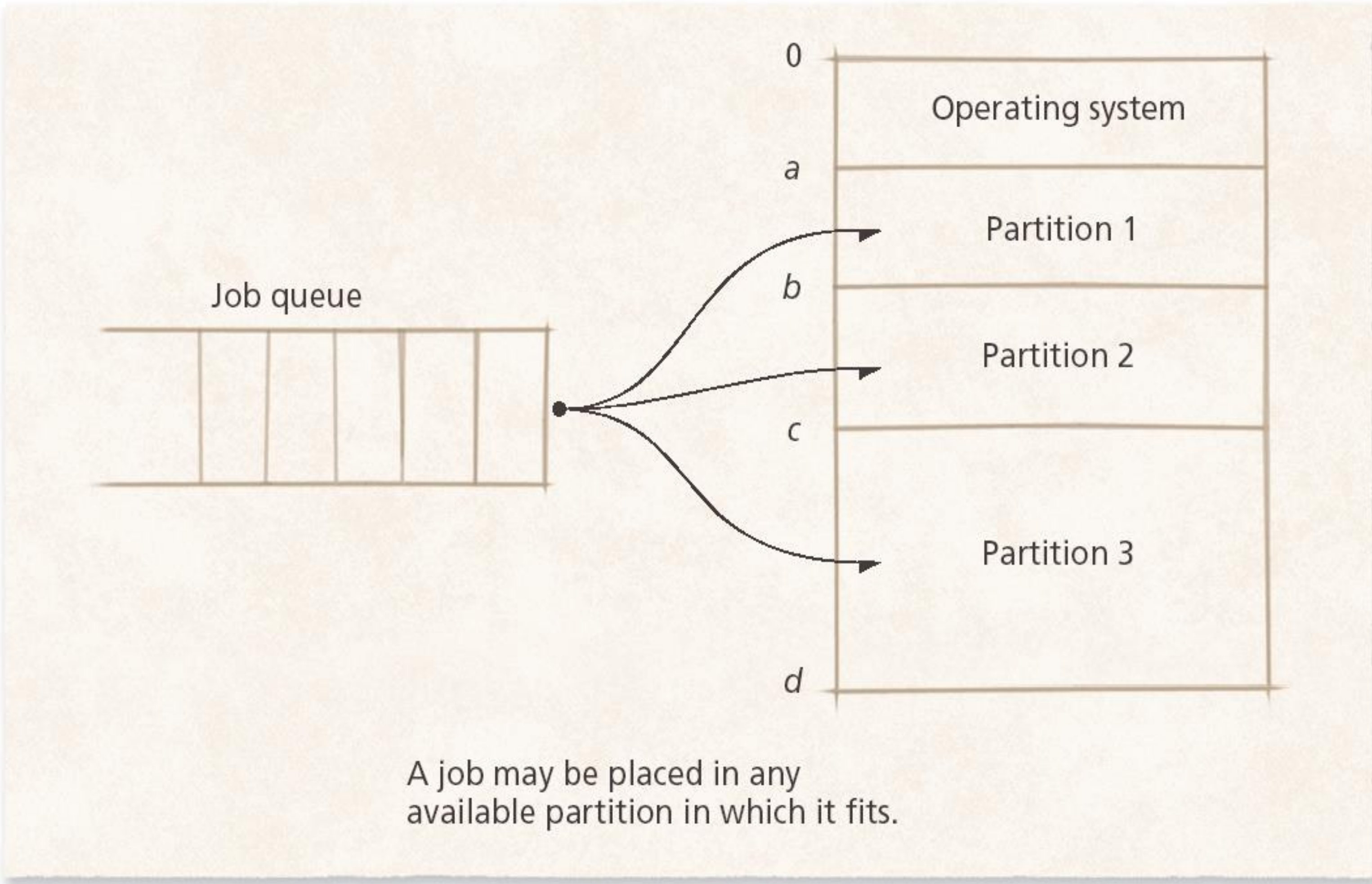
9.8 Fixed-Partition Multiprogramming

Figure 9.8 Memory waste under fixed-partition multiprogramming with absolute (fixed) translation and loading.



9.8 Fixed-Partition Multiprogramming

Figure 9.8 Fixed-partition multiprogramming with relocatable (searching) translation and loading.

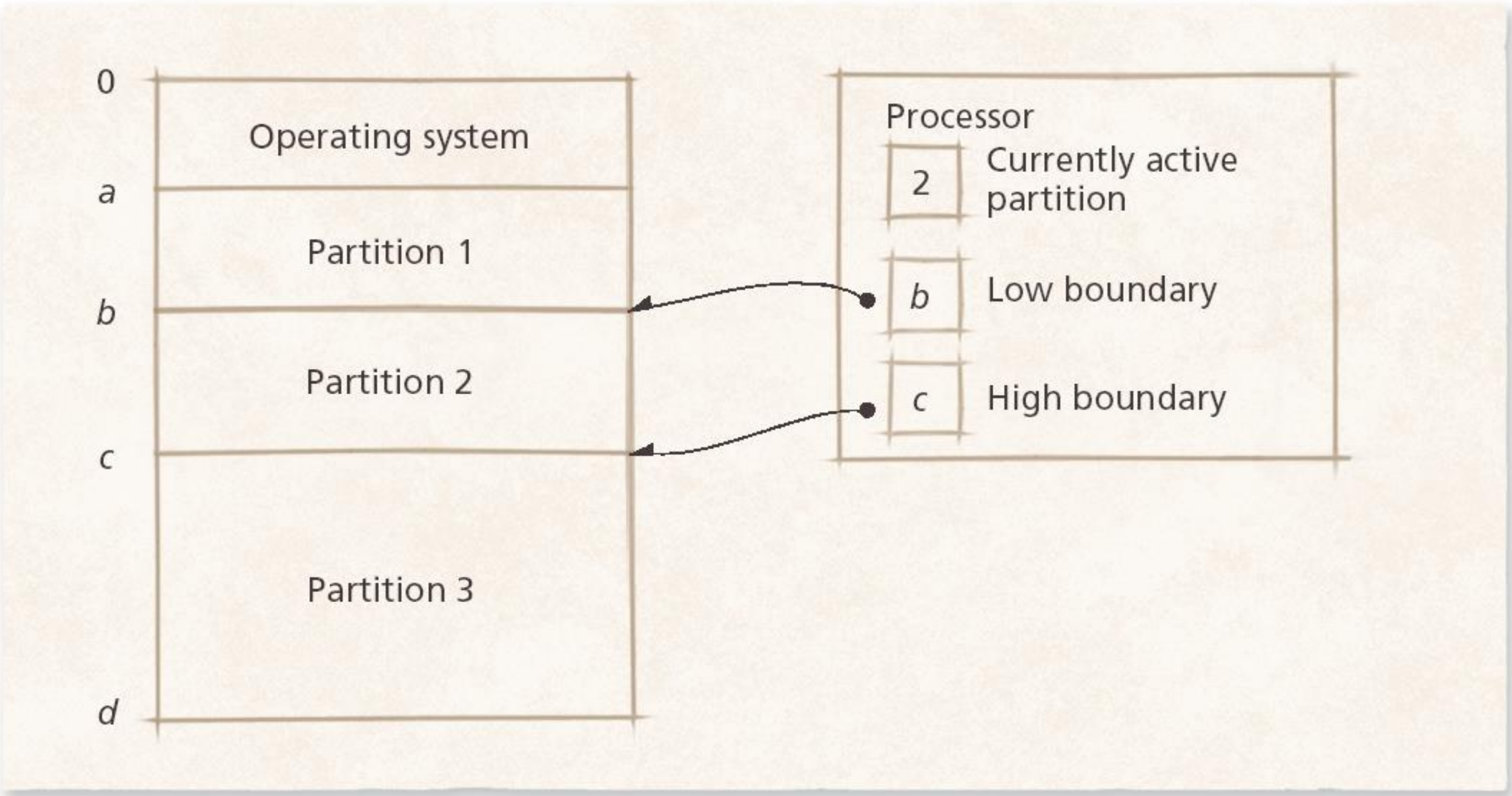


9.8 Fixed-Partition Multiprogramming

- Protection
  - Can be implemented by boundary registers, called base and limit (also called low and high)

9.8 Fixed-Partition Multiprogramming

Figure 9.10 Memory protection in contiguous-allocation multiprogramming systems.

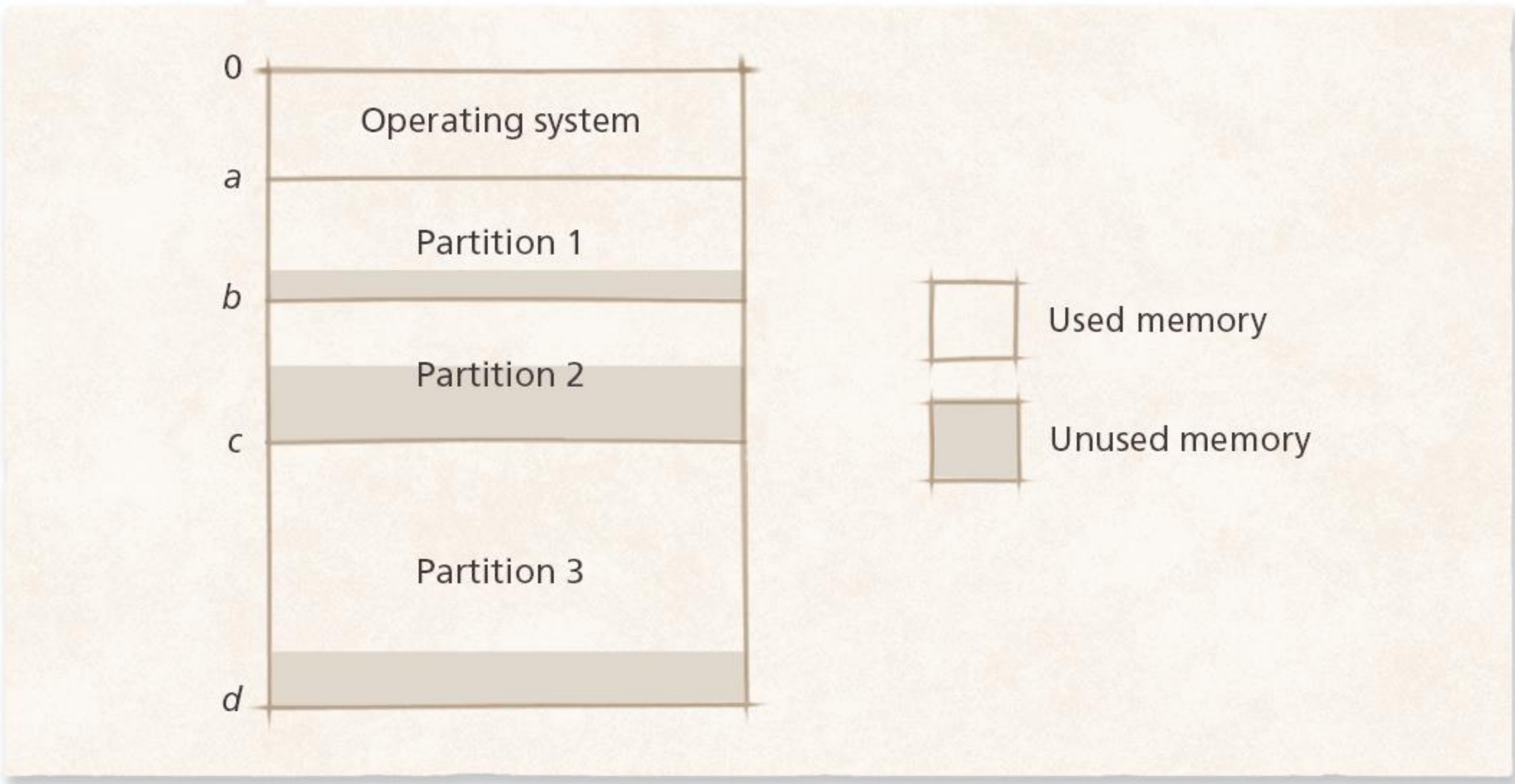


9.8 Fixed-Partition Multiprogramming

- Drawbacks to fixed partitions (Cont.)
  - Internal fragmentation
    - Process does not take up entire partition, wasting memory
  - Incurs (acquire) more overhead
    - Offset (balance) by higher resource utilization

9.8 Fixed-Partition Multiprogramming

Figure 9.11 Internal fragmentation in a fixed-partition multiprogramming system.

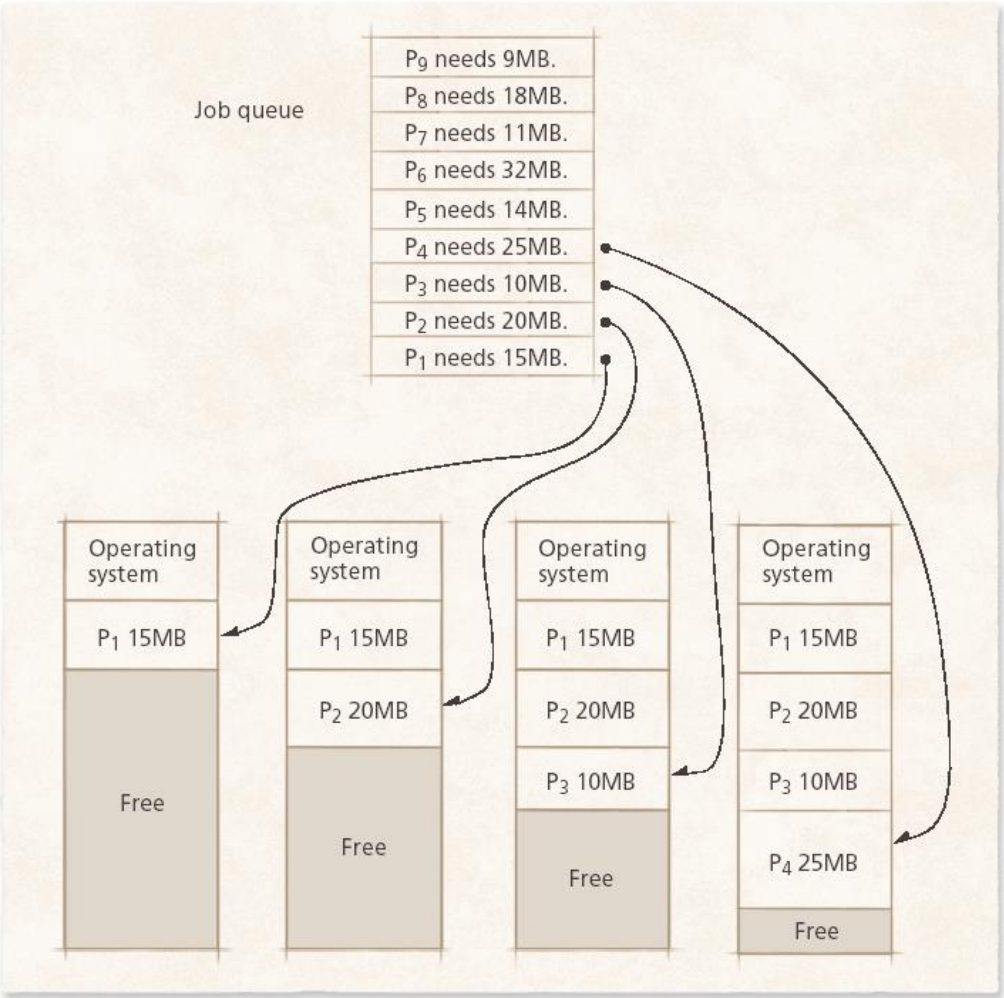


9.9 Variable-Partition Multiprogramming

- System designers found fixed partitions too restrictive
  - Internal fragmentation
  - Potential for processes to be too big to fit anywhere
  - Variable partitions designed as replacement

9.9 Variable-Partition Multiprogramming

Figure 9.12 Initial partition assignments in variable-partition programming.

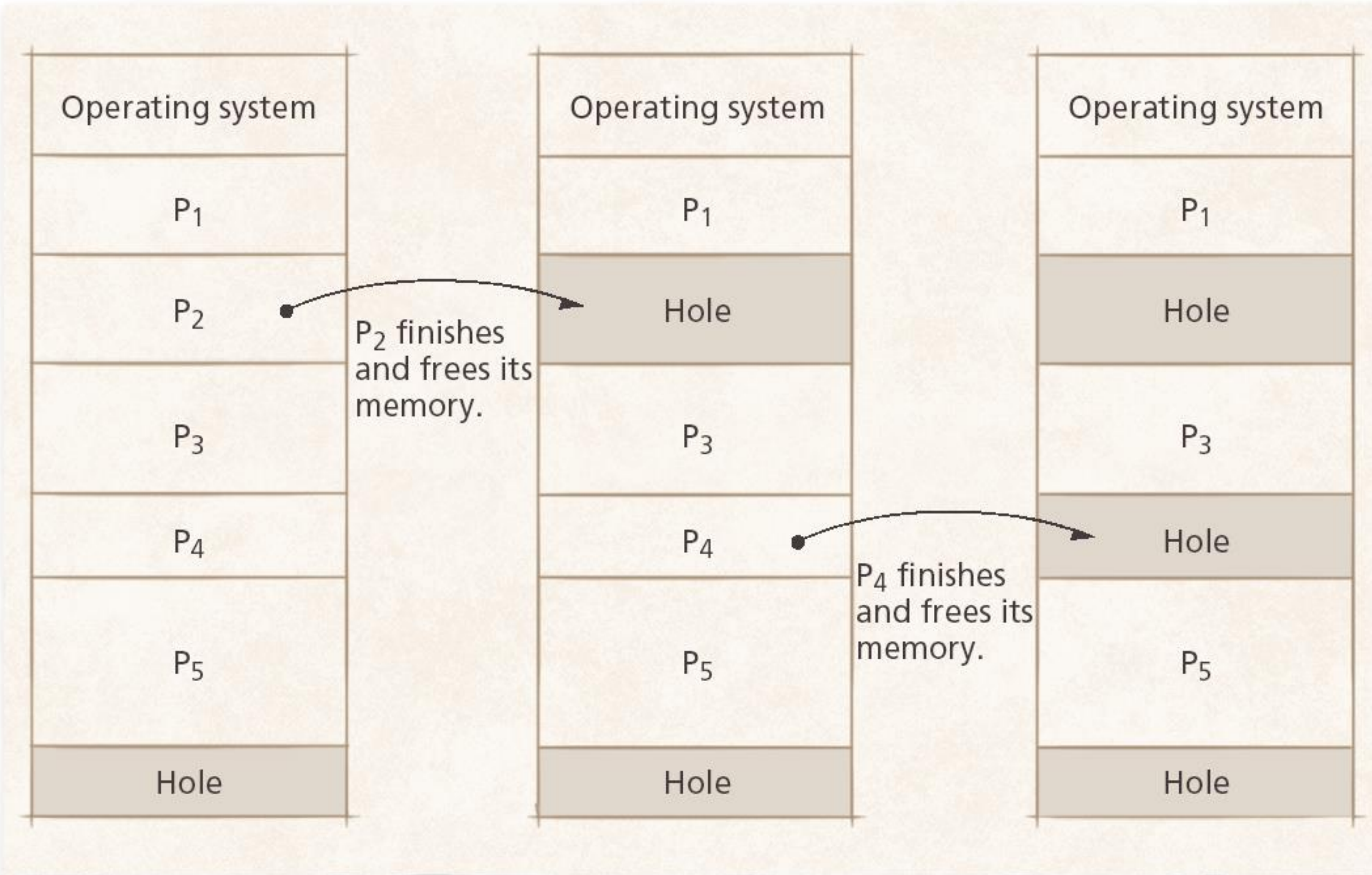


9.9.1 Variable-Partition Characteristics

- Jobs placed where they fit
  - No space wasted initially
  - Internal fragmentation impossible
    - Partitions are exactly the size they need to be
  - External fragmentation can occur when processes removed
    - Leave holes too small for new processes
    - Eventually no holes large enough for new processes

9.9.1 Variable-Partition Characteristics

Figure 9.13 Memory "holes" in variable-partition multiprogramming.

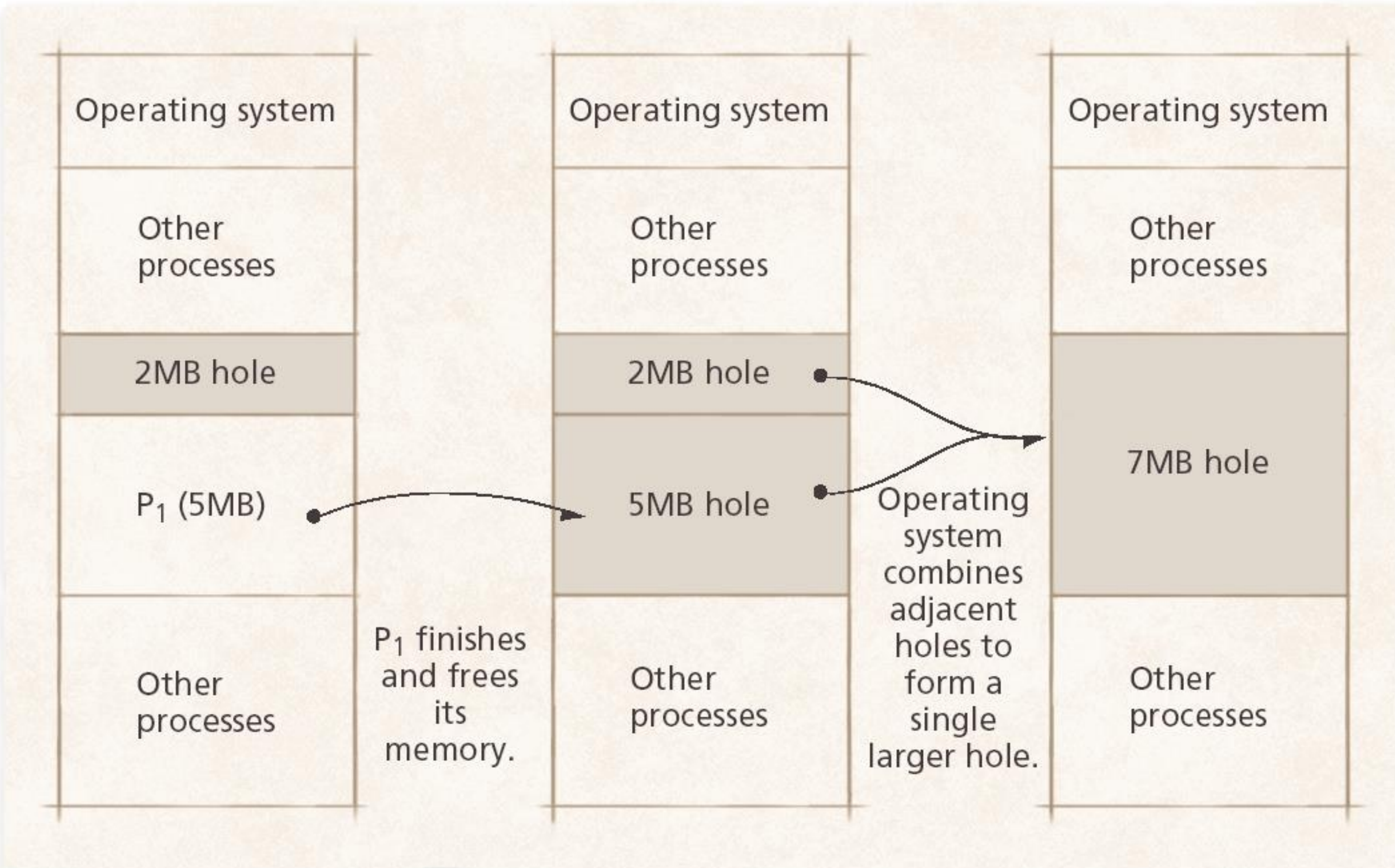


9.9.1 Variable-Partition Characteristics

- Several ways to combat external fragmentation
  - Coalescing
    - Combine adjacent free blocks into one large block
    - Often not enough to reclaim significant amount of memory
  - Compaction
    - Sometimes called garbage collection (not to be confused with GC in object-oriented languages)
    - Rearranges memory into a single contiguous block free space and a single contiguous block of occupied space
    - Makes all free space available
    - Significant overhead

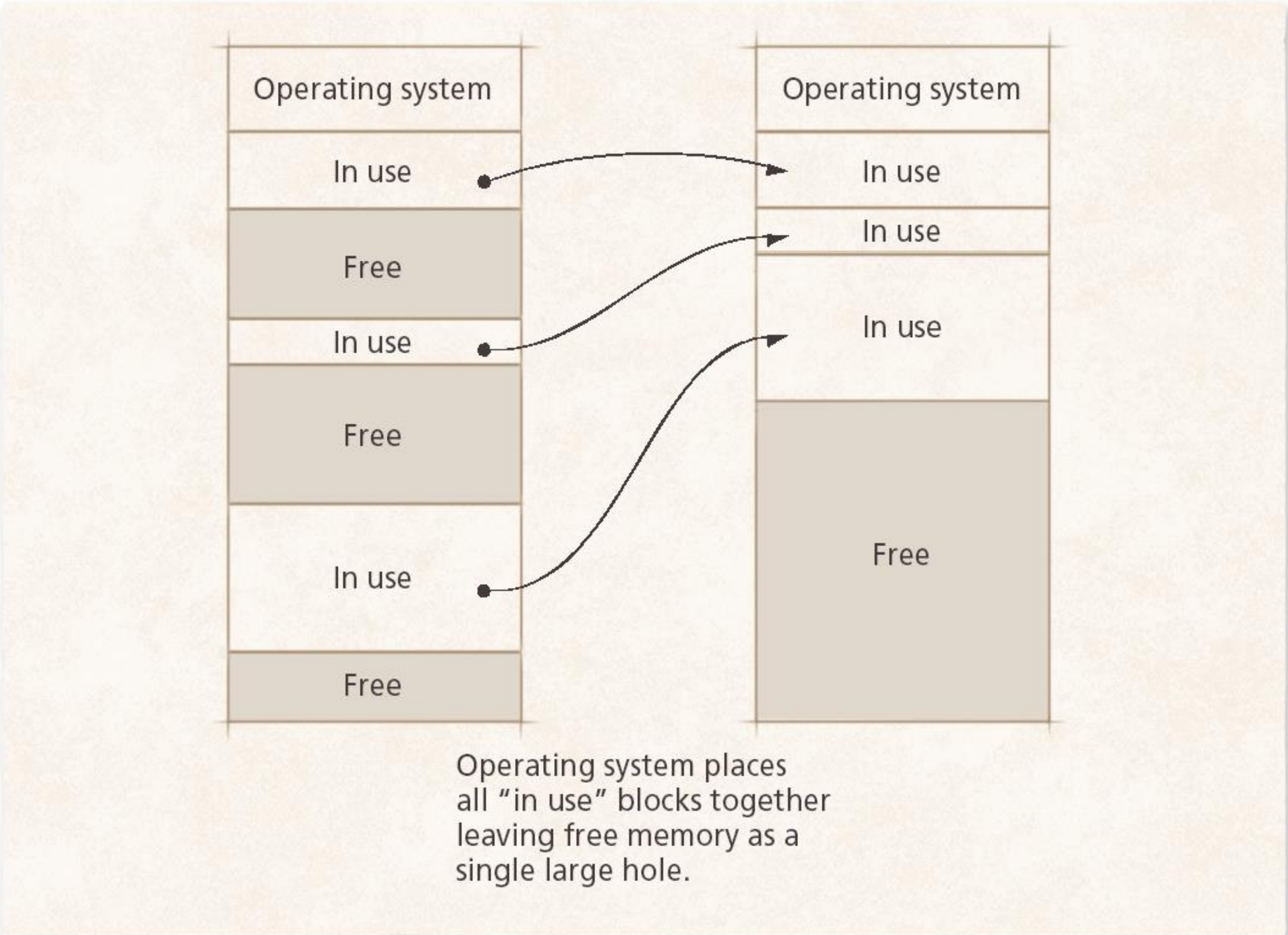
9.9.1 Variable-Partition Characteristics

Figure 9.14 Coalescing memory "holes" in variable-partition multiprogramming.



9.9.1 Variable-Partition Characteristics

Figure 9.15 Memory compaction in variable-partition multiprogramming.

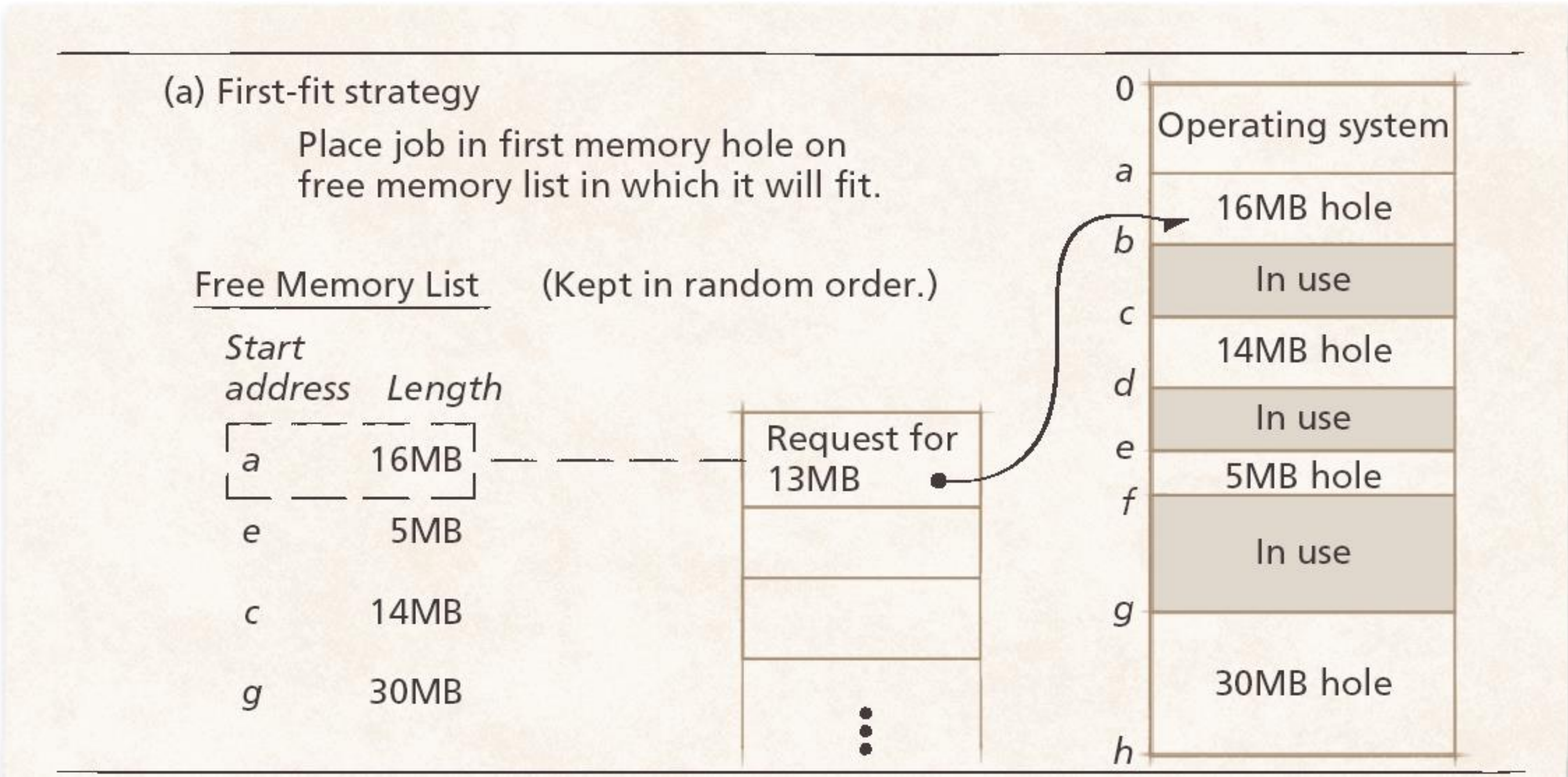


9.9.2 Memory Placement Strategies

- Where to put incoming processes
  - First-fit strategy
    - Process placed in first hole of sufficient size found
    - Simple, low execution-time overhead
  - Best-fit strategy
    - Process placed in hole that leaves least unused space around it
    - More execution-time overhead
  - Worst-fit strategy
    - Process placed in hole that leaves most unused space around it
    - Leaves another large hole, making it more likely that another process can fit in the hole

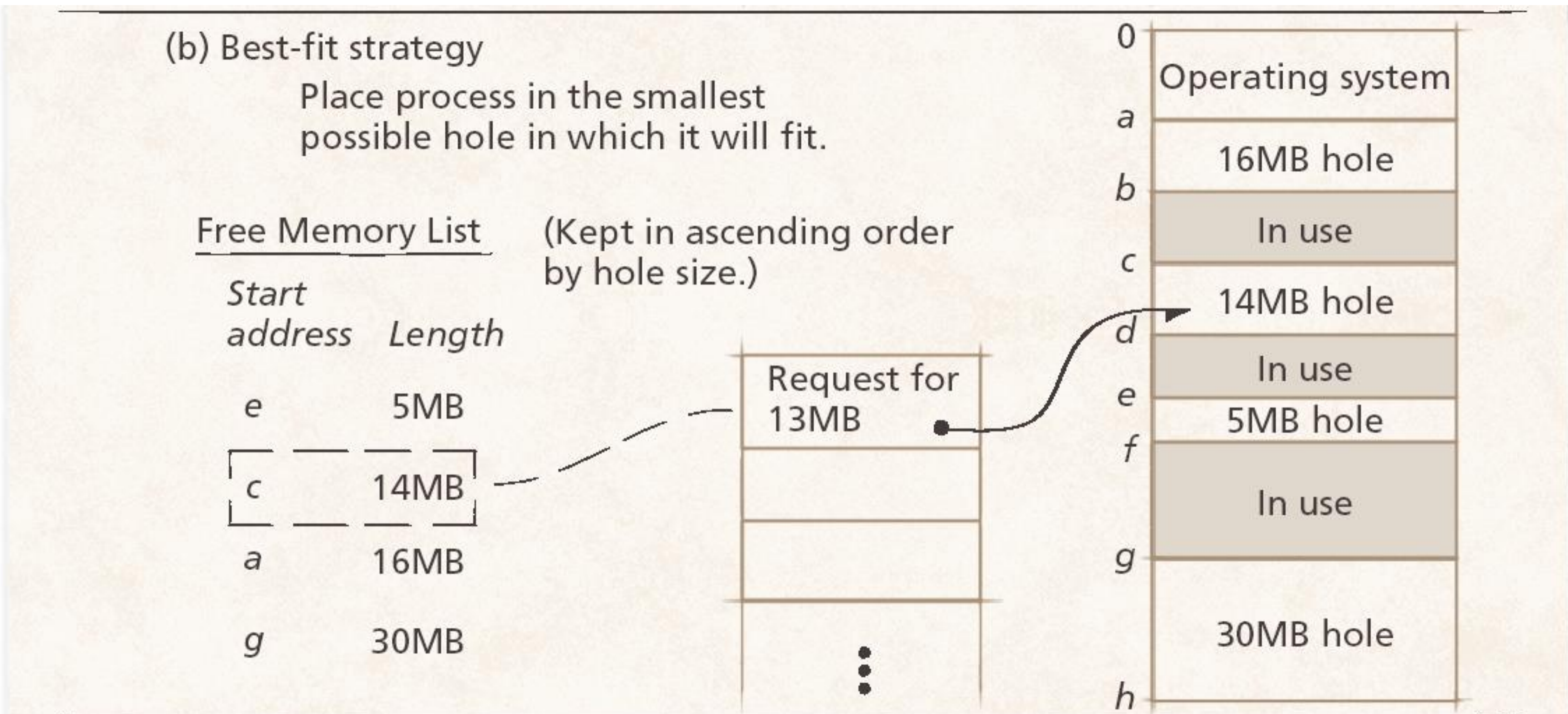
9.9.2 Memory Placement Strategies

Figure 9.16 First-fit, best-fit and worst-fit memory placement strategies (Part 1 of 3).



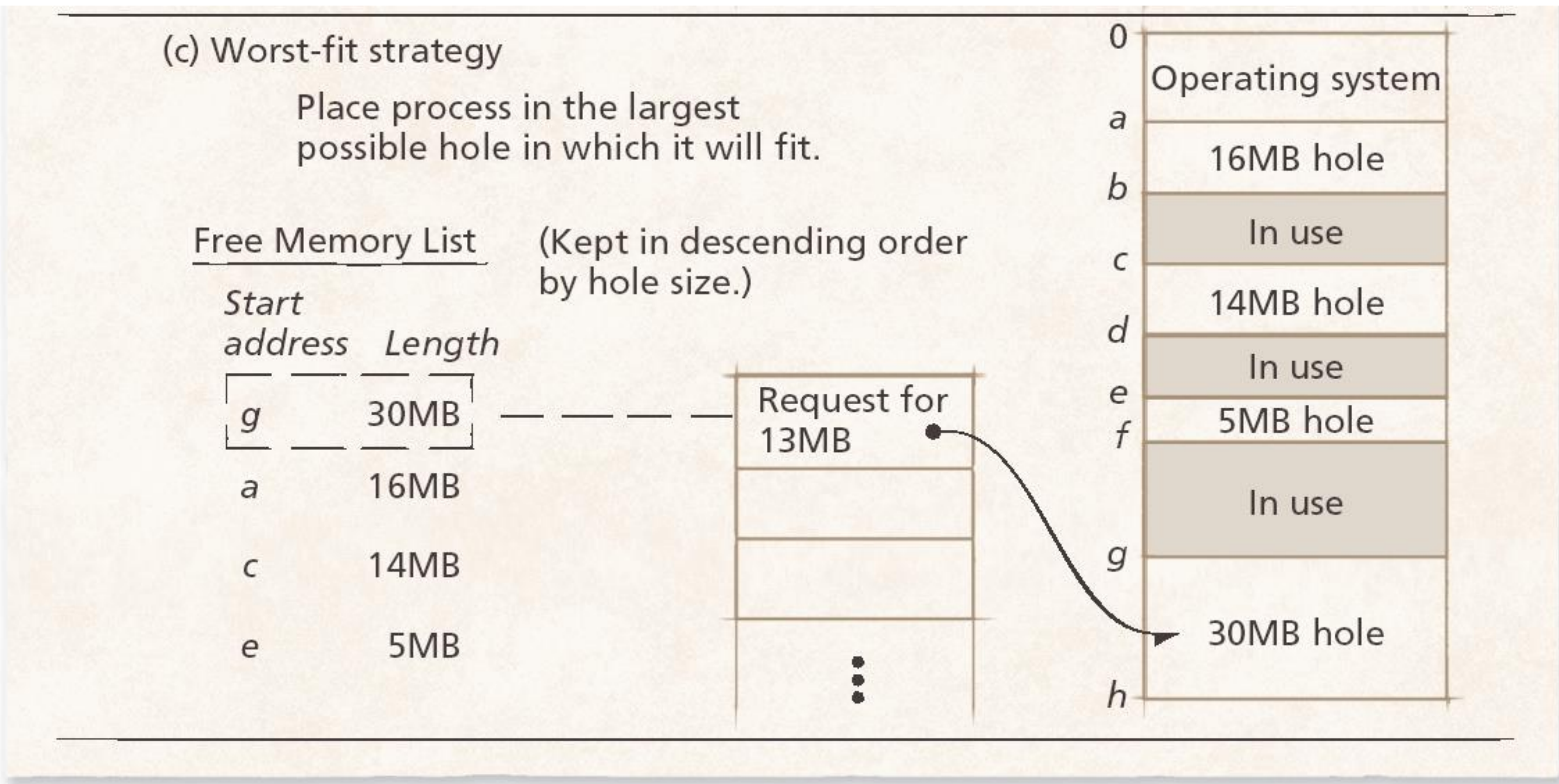
9.9.2 Memory Placement Strategies

Figure 9.16 First-fit, best-fit and worst-fit memory placement strategies (Part 2 of 3).



9.9.2 Memory Placement Strategies

Figure 9.16 First-fit, best-fit and worst-fit memory placement strategies (Part 3 of 3).

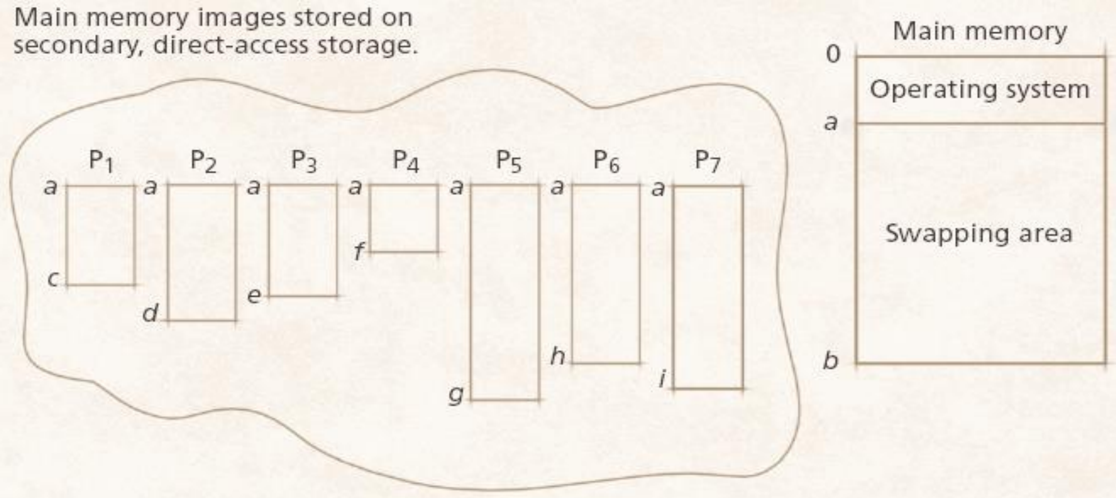


9.10 Multiprogramming with Memory Swapping

- Not necessary to keep inactive processes in memory
  - Swapping
    - Only put currently running process in main memory
      - Others temporarily moved to secondary storage
      - Maximizes available memory
      - Significant overhead when switching processes
    - Better yet: keep several processes in memory at once
      - Less available memory
      - Much faster response times
      - Similar to paging

9.10 Multiprogramming with Memory Swapping

Figure 9.17 Multiprogramming in a swapping system in which only a single process at a time is in main memory.



1. Only one process at a time resides in main memory.

2. That process runs until
   a) I/O is issued,
   b) timer runs out or
   c) voluntary termination.

3. System then swaps out the process by copying the swapping area (main memory) to secondary storage.

4. System swaps in next process by reading that process's main memory image into the swapping area. The new process runs until it is eventually swapped out and the next user is swapped in, and so on.