Operating Systems

Memory Management

Chapter 7

Outline

- Memory Management Requirements
- Memory Partitioning
- Paging
- Segmentation

Memory Management

- Subdividing memory to accommodate multiple processes
- Memory needs to be allocated efficiently to pack as many processes into memory as possible
- Memory management requirements
  - Relocation
  - Protection
  - Sharing
  - Logical organization
  - Physical organization

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

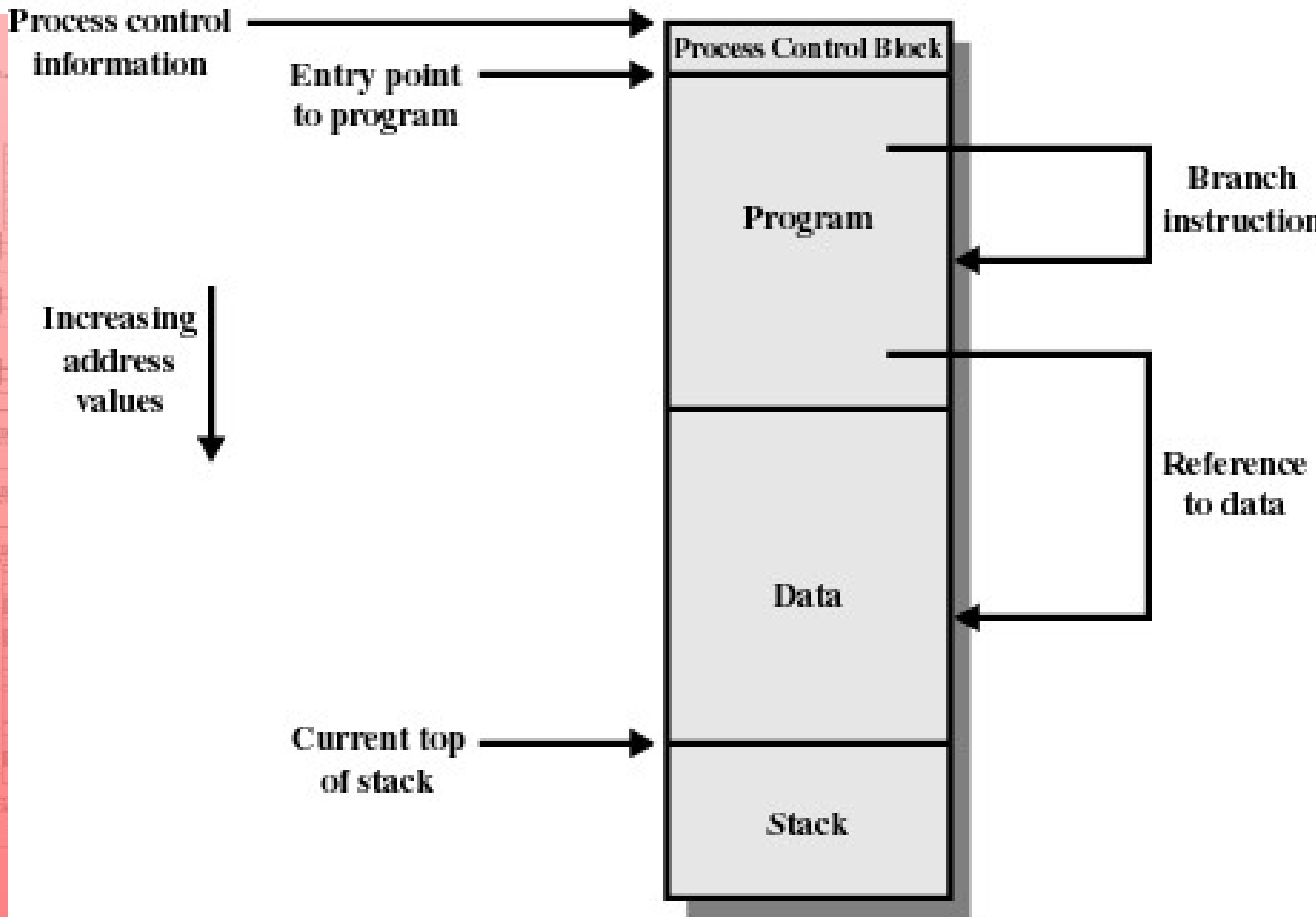


Figure 7.1 Addressing Requirements for a Process

Memory Management Requirements

- Protection
  - Processes should not be able to reference memory locations in another process without permission
  - Impossible to check absolute addresses in programs since the program could be relocated
  - Must be checked during execution
    - Operating system cannot anticipate all of the memory references a program will make

Memory Management Requirements

- Sharing
  – Allow several processes to access the same portion of memory
  – Better to allow each process (person) 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

Memory Management Requirements

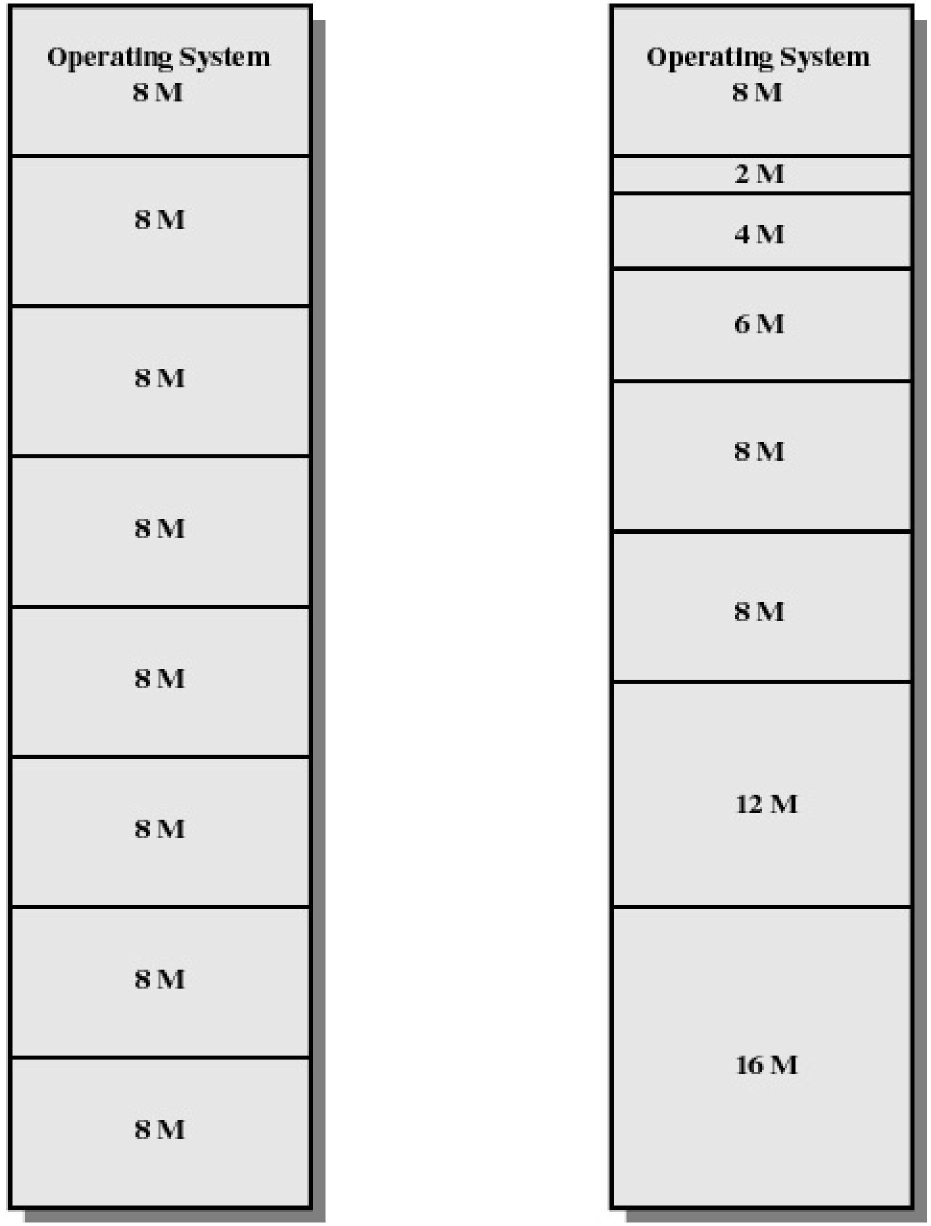
- Physical Organization
  - Memory available for a program plus its data may be insufficient
    - Overlaying allows various modules to be assigned the same region of memory
  - Programmer does not know how much space will be available

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
  - a program may not fit in a partition. The programmer must design the program with overlays

Fixed Partitioning

- Main memory use is inefficient. Any program, no matter how small, occupies an entire partition. This is called *internal fragmentation*.



(a) Equal-size partitions

(b) Unequal-size partitions

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

Placement Algorithm with Partitions

- Equal-size partitions
  - Trivial: because all partitions are of equal size, it does not matter which partition is used
- Unequal-size partitions
  - 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

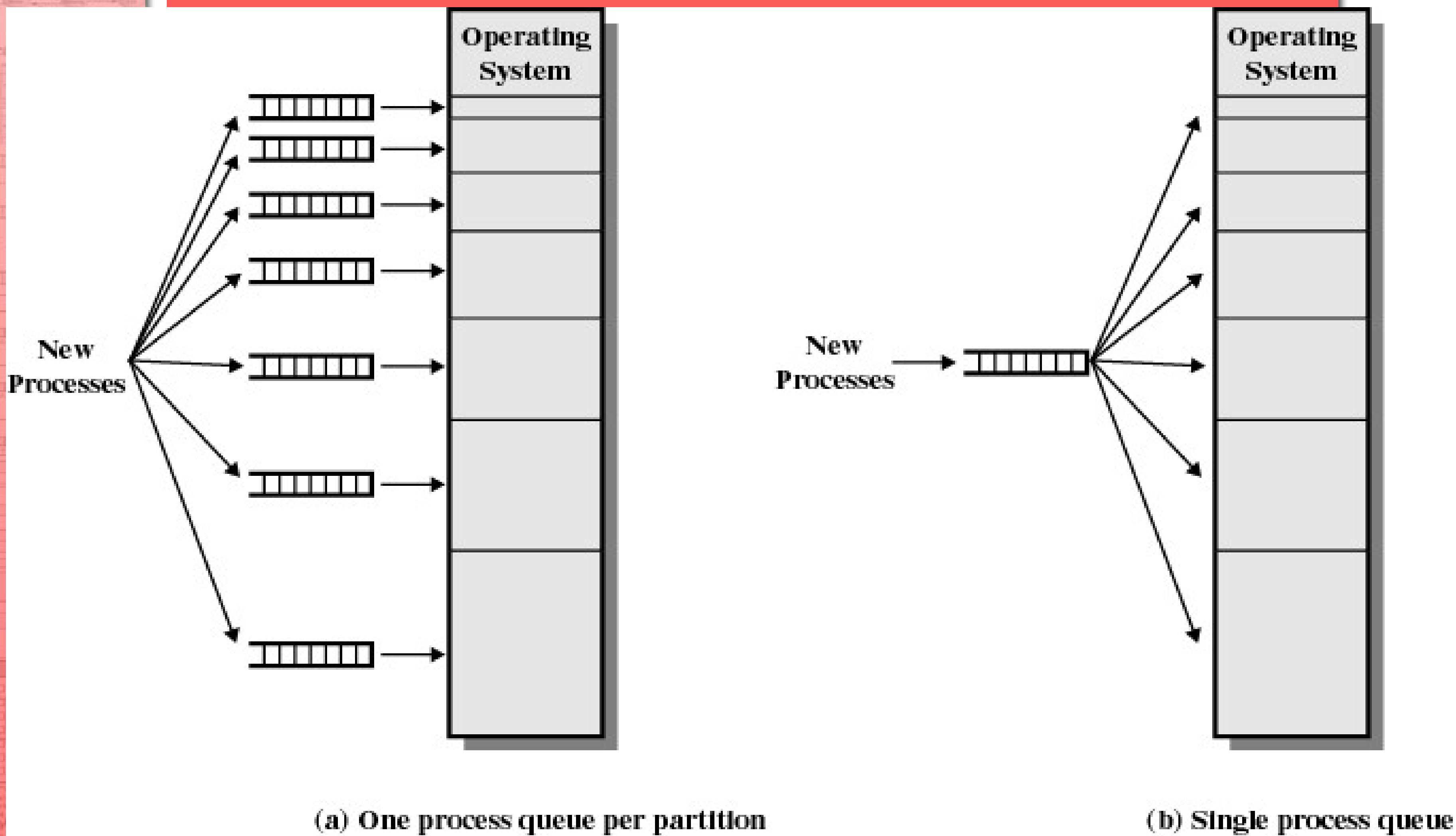


Figure 7.3 Memory Assignment for Fixed Partitioning

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

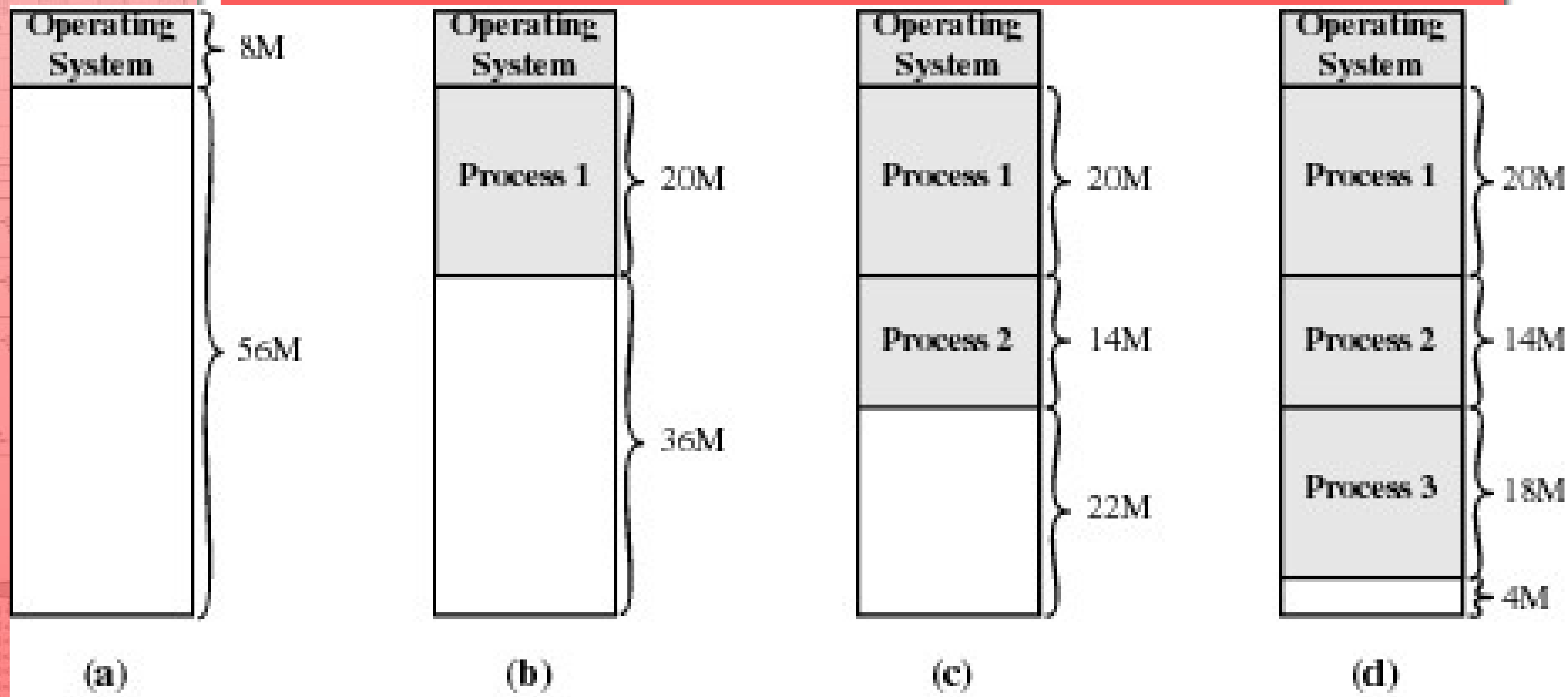


Figure 7.4 The Effect of Dynamic Partitioning

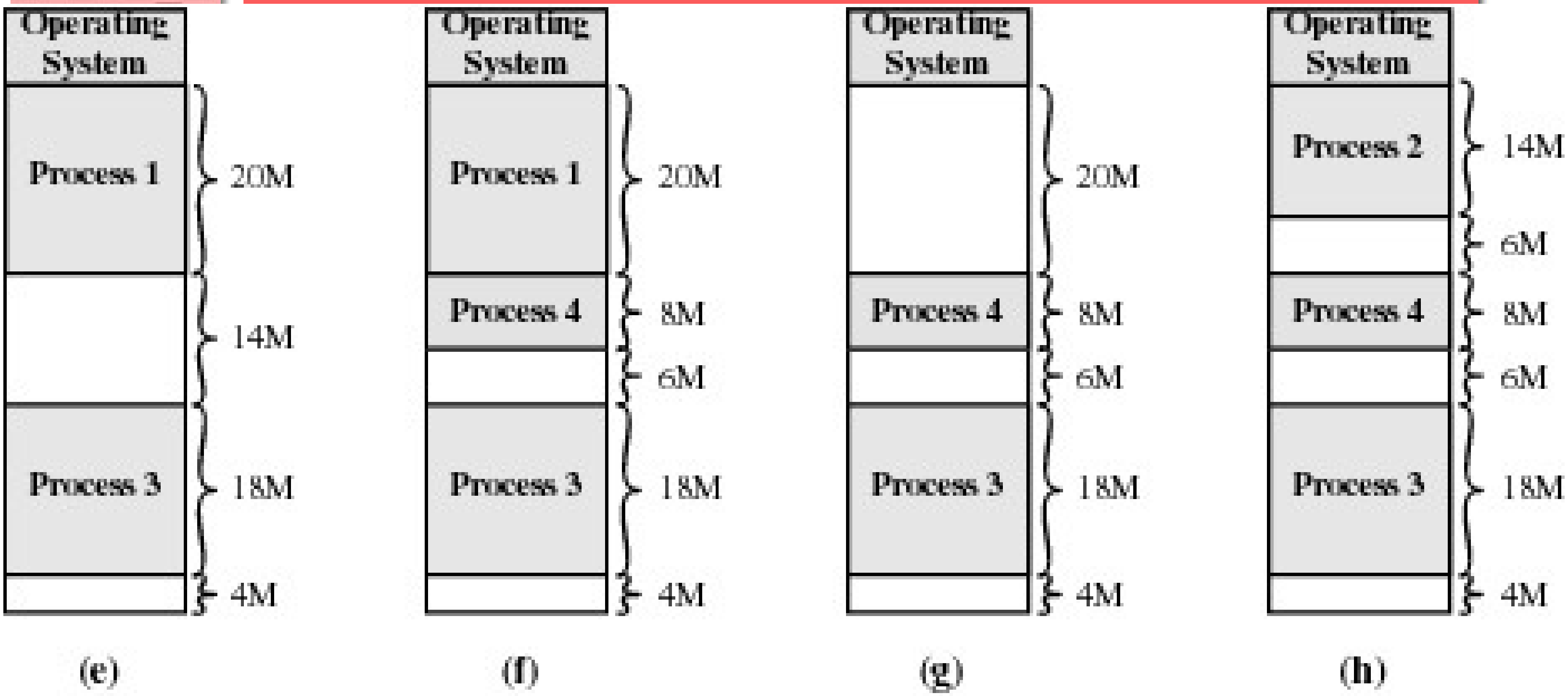


Figure 7.4 The Effect of Dynamic Partitioning

Dynamic Partitioning Placement Algorithm

- 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 Placement Algorithm

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

Dynamic Partitioning Placement Algorithm

- Next-fit
  - Begin to scan memory from the location of the last placement and choose next available block that is large enough
  - More often allocate 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

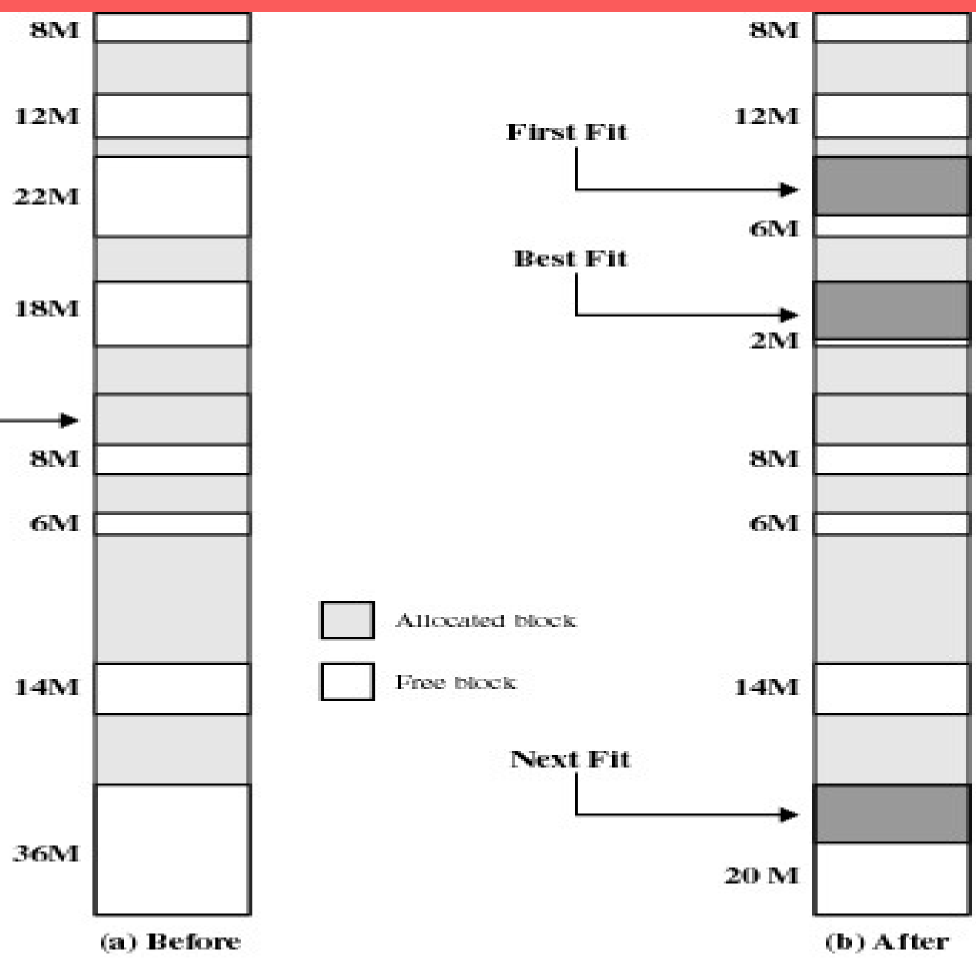


Figure 7.5 Example Memory Configuration Before and After Allocation of 16 Mbyte Block

Buddy System

- Entire space available is treated as a single block of 2^U
- If a request of size *s* such that $2^{U-1} < s <= 2^U$, 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

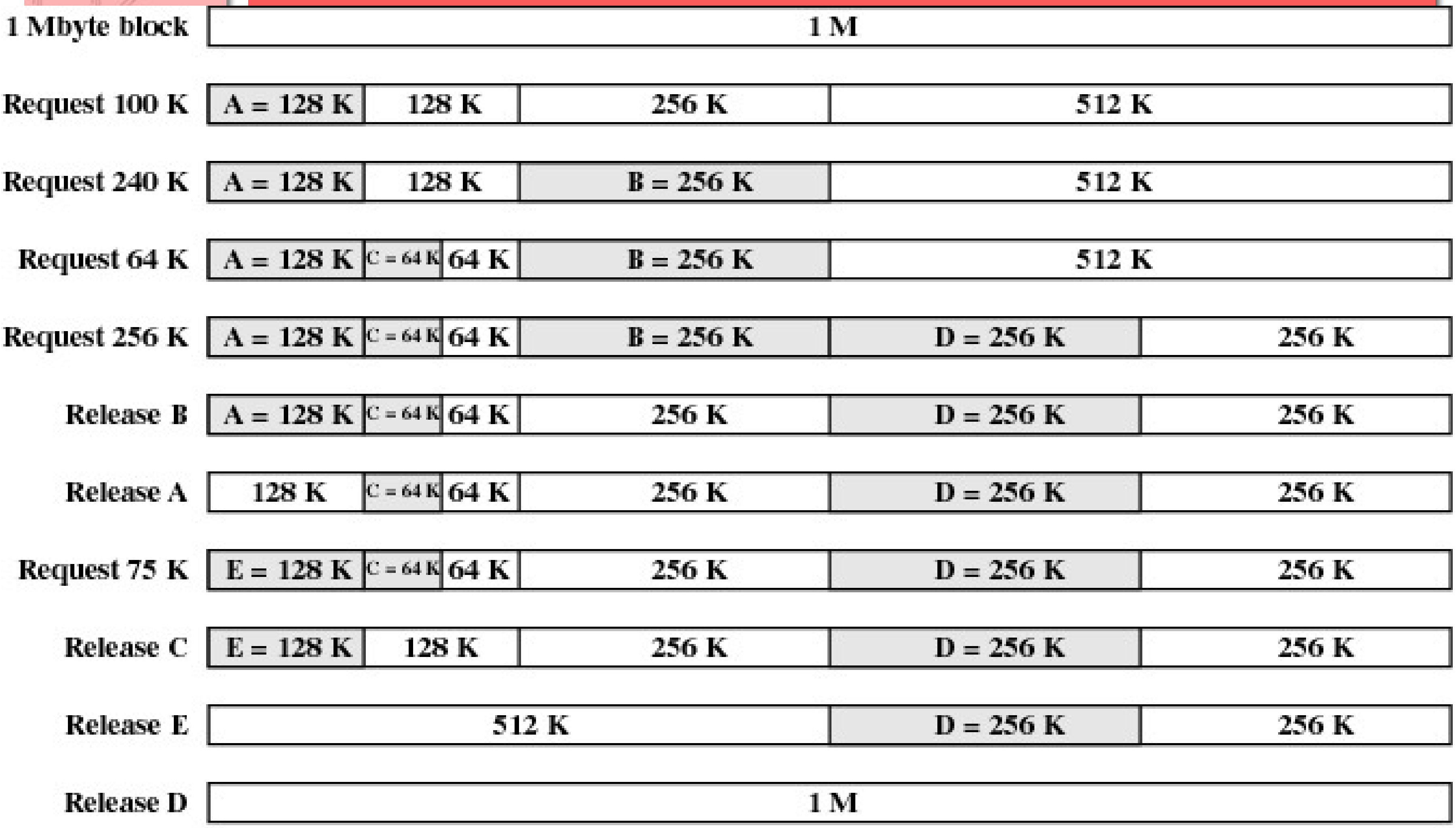


Figure 7.6 Example of Buddy System

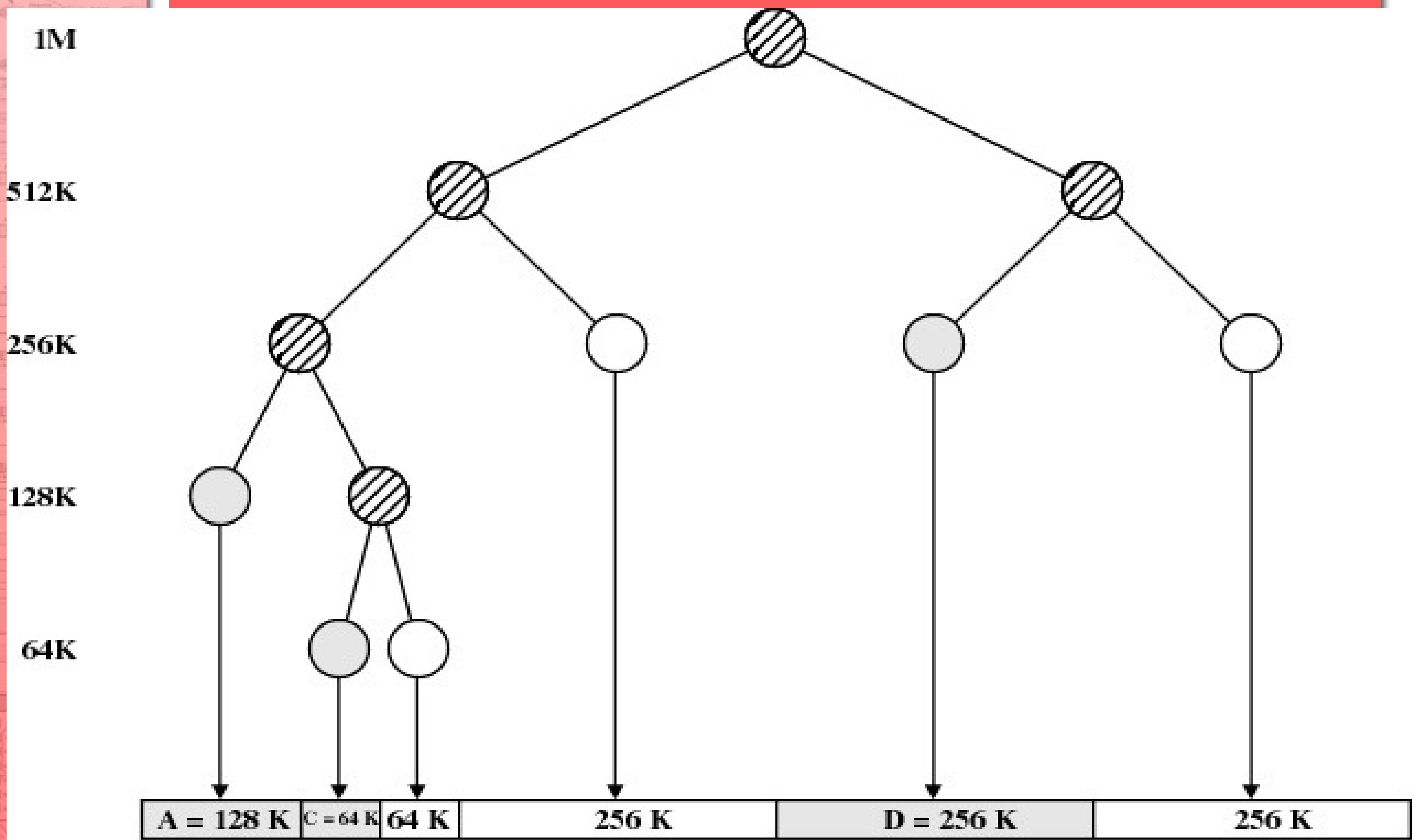


Figure 7.7 Tree Representation of Buddy System

Relocation

- When program 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)
- 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
- Physical
  - the absolute address or actual location in main memory

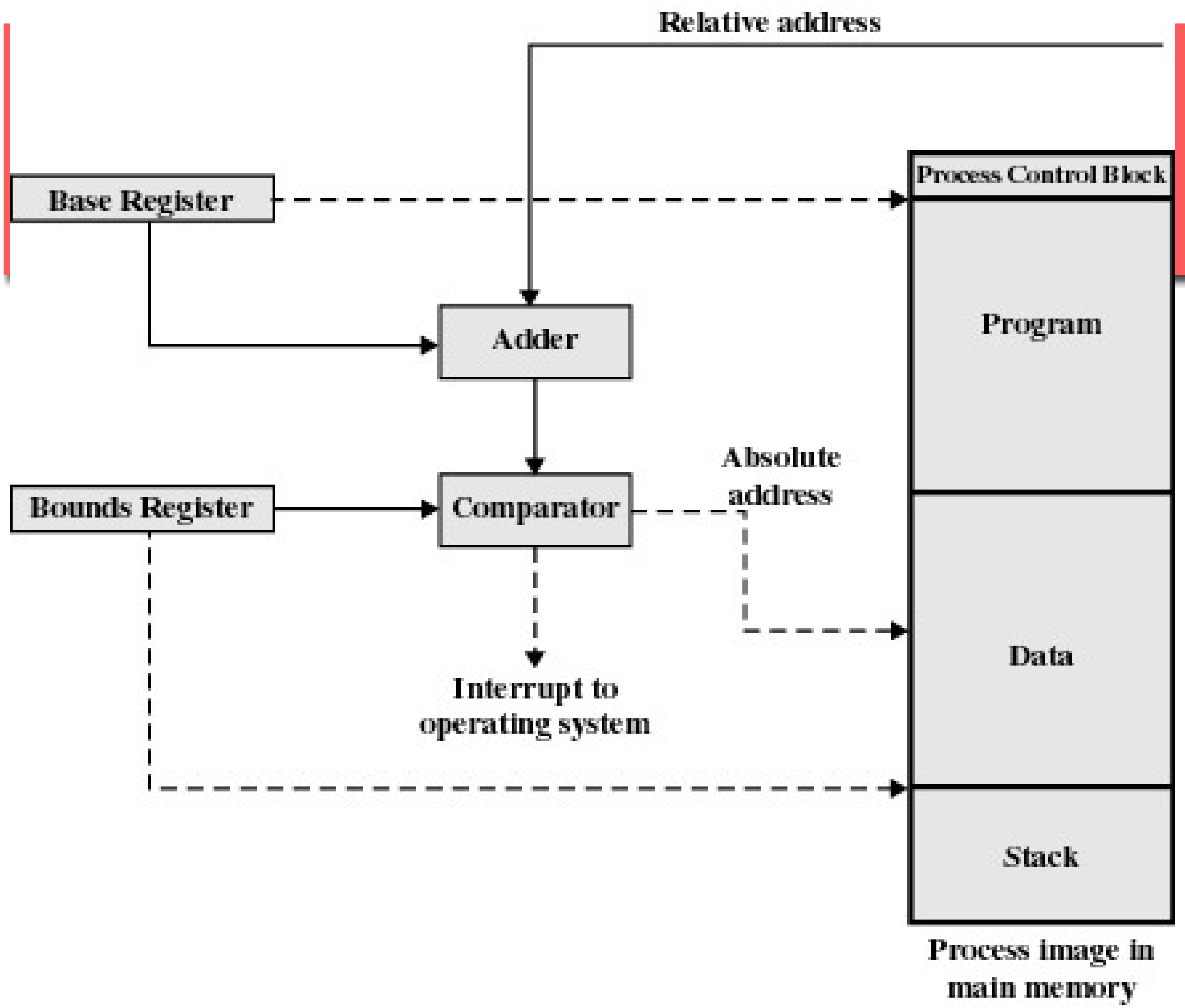


Figure 7.8 Hardware Support for Relocation

Registers Used during Execution

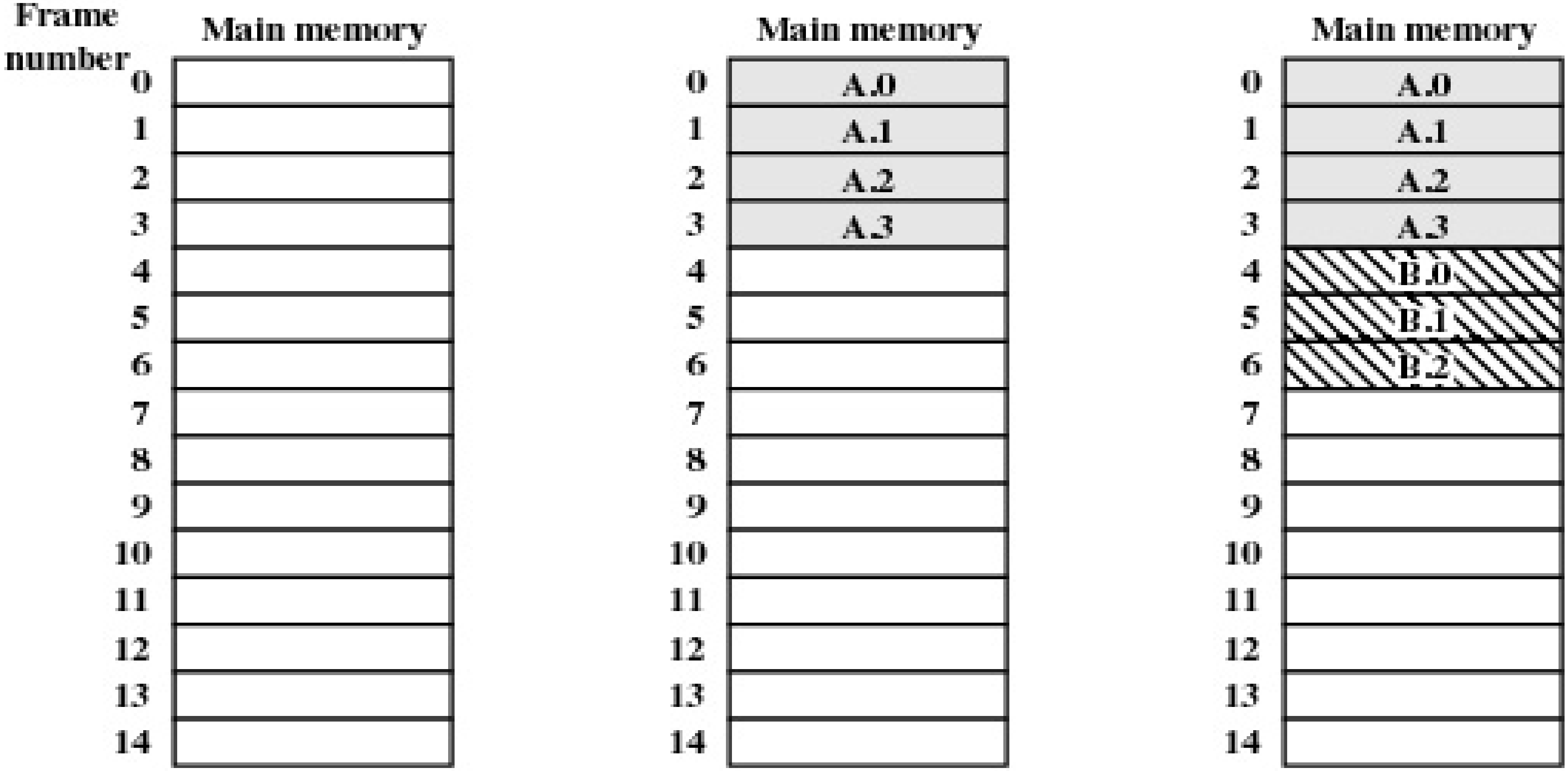
- Base register
  - starting address for the process
- Bounds register
  - ending location of the process
- These values are set when the process is loaded and when the process is swapped in

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-size chunks and divide each process into the same size chunks (more convenient to be a power of 2)
- The chunks of a process are called *pages* and chunks of memory are called *frames*
- 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

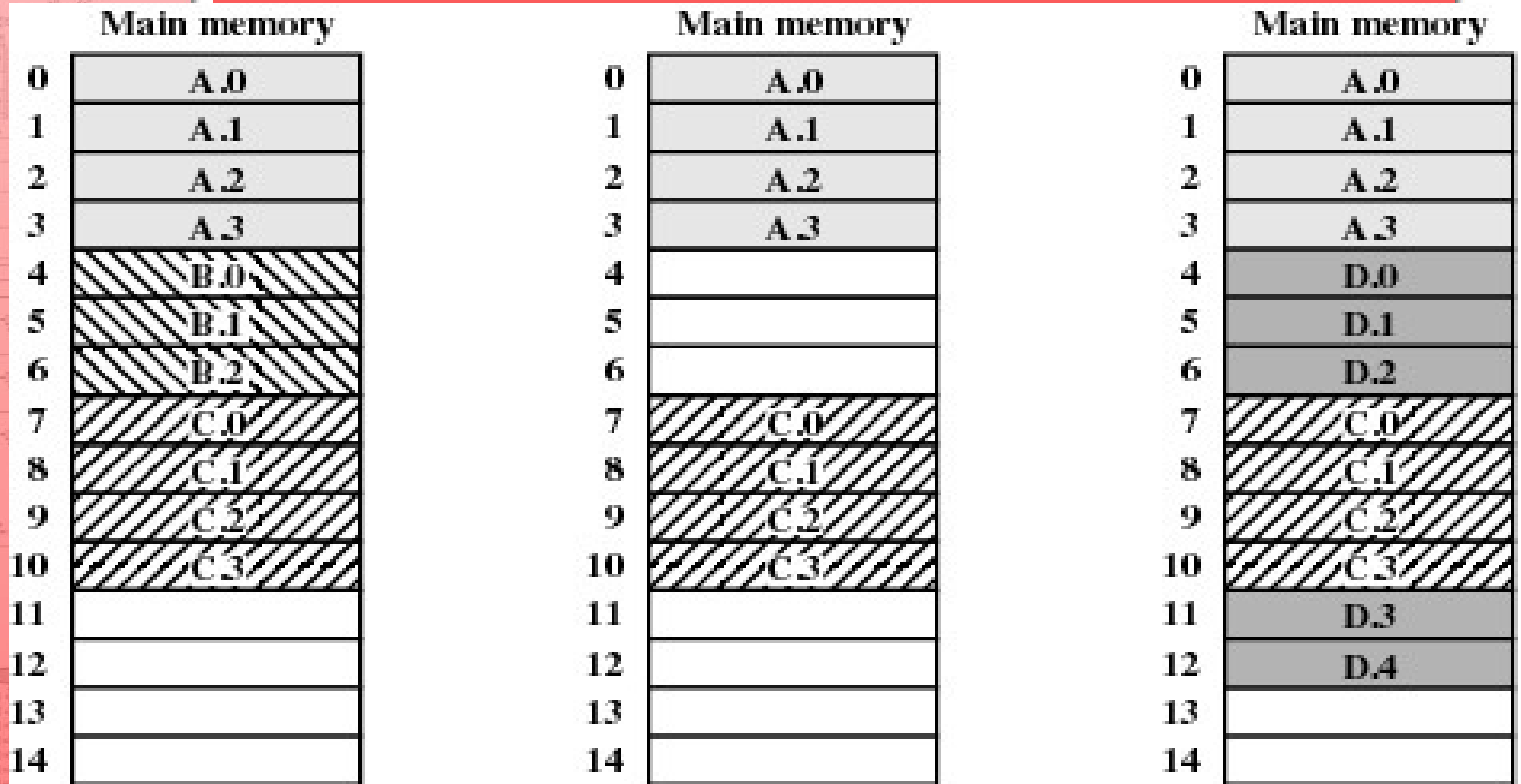


(a) Fifteen Available Frames

(b) Load Process A

(b) Load Process B

Figure 7.9 Assignment of Process Pages to Free Frames



(d) Load Process C (e) Swap out B (f) Load Process D

Figure 7.9 Assignment of Process Pages to Free Frames

Page Tables for Example

0	0
1	1
2	2
3	3

Process A page table

0	—
1	—
2	—

Process B page table

0	7
1	8
2	9
3	10

Process C page table

0	4
1	5
2	6
3	11
4	12

Process D page table

13
14

Free frame list

Need to map logical address (page number, offset) into a physical address (frame number, offset)

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

Why Frame Size power of 2?

- Relative address (with reference to origin of the program) equivalent to logical address (page number, offset)
  - 16 bit addresses used
  - Relative address 1502 with a page size 1024 (10 bits needed for offset) corresponds to page 1, offset 478 (how?)
  - 6 bits for page number (maximum of $2^6 = 64$ pages)
  - 1502 = 0000010111011110
  - Page 1 = 000001 offset 478 = 0111011110
  - Page, offset = 0000010111011110

Why Frame Size power of 2?

- Easy to implement a function in HWR to perform dynamic address translation at run time.
  - (n+m) bit addresses (leftmost n: page #, rightmost m: offset)
  - Extract page # from address
  - Use page # as an index into process page table to find frame number k
  - Starting physical address of frame is $k * 2^m$
  - Physical address is $k * 2^m$ +offset
  - Need not be calculated, *append frame number to offset*

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

Dynamic Address Translation in Segmentation

- (n+m) bit addresses (leftmost n: segment #, rightmost m: offset)
- Extract segment # from address
- Use segment # as an index into process segment table to find starting physical address of segment
- Compare offset (rightmost m bits), to length of segment. If offset is greater → address is invalid
- Physical address is SUM of starting physical address of segment + offset