Memory Management

Chapter 7

Contents

- Memory management requirements
- Memory partitioning
  - Fixed partitioning
  - Dynamic partitioning
- Simple Paging
- Simple 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(relocation)
  - memory references in the code must be translated to actual physical memory address

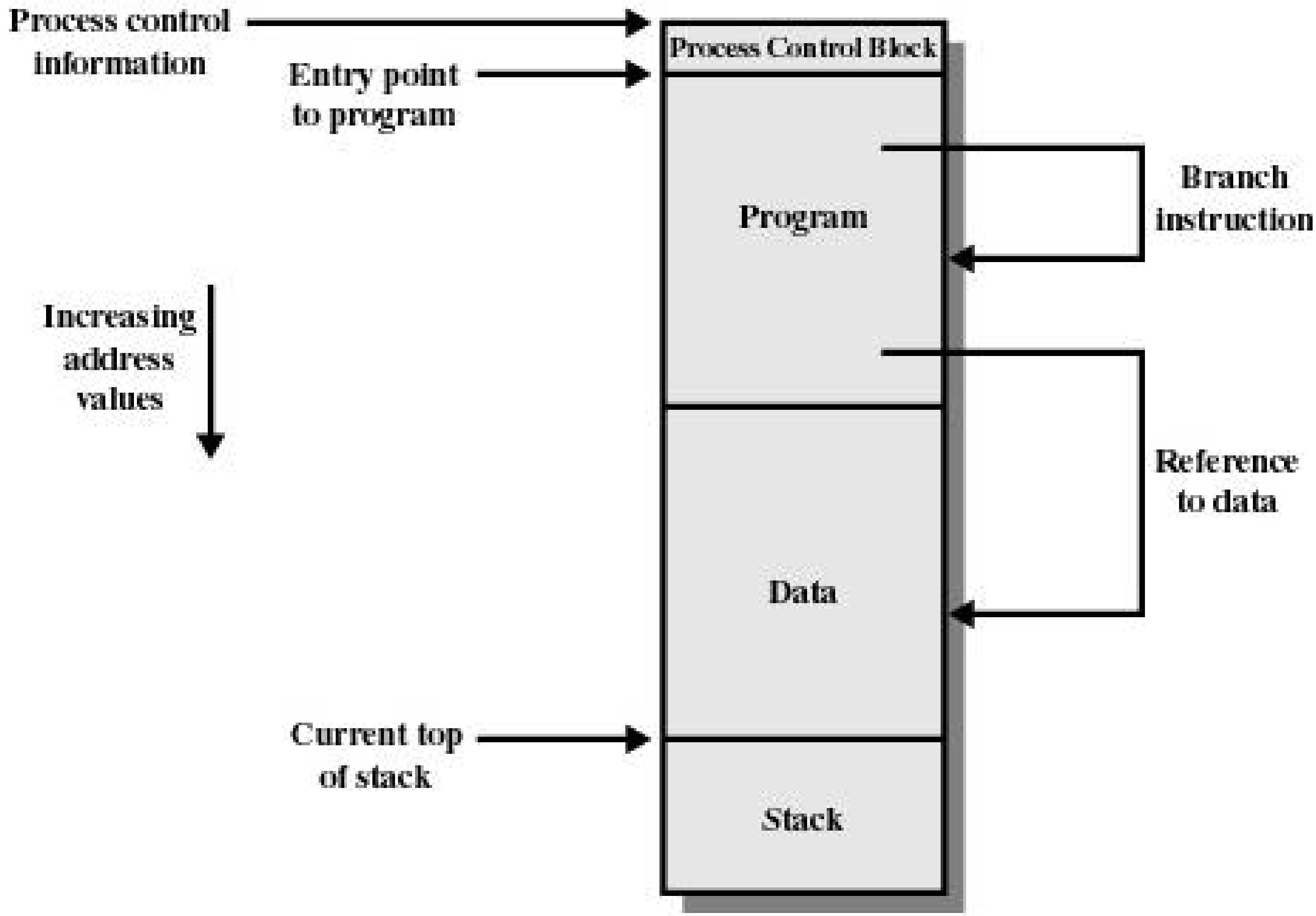


Figure 7.1 Addressing Requirements for a Process

Memory Management Requirements

- Protection
  - processes should not be able to reference memory locations in another process without permission
  - impossible to check absolute addresses in compile time since the program could be relocated
  - so it must be checked during execution by hardware(processor)

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 its own separate copy

Memory Management Requirements

- Physical main memory is a sequence of bytes
- Logical Organization
  - programs are written in modules
  - modules can be written and compiled independently
    - shared modules
      - shared memory
  - different degrees of protection given to modules (read-only, execute-only)

Memory Management Requirements

- Physical Organization
  - computer memory is organized into at least two levels
    - main memory, secondary memory
  - task of moving information between the two levels is a major system concern
    - virtual memory

Memory Management Techniques

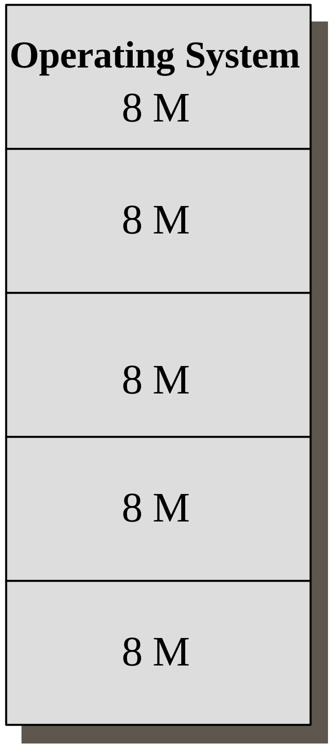
- Memory partitioning
  - fixed partitioning
  - dynamic partitioning
- Simple paging
- Simple segmentation
- Virtual Memory paging
- Virtual Memory segmentation

Fixed Partitioning

- Partition available memory into regions with fixed boundaries
- 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

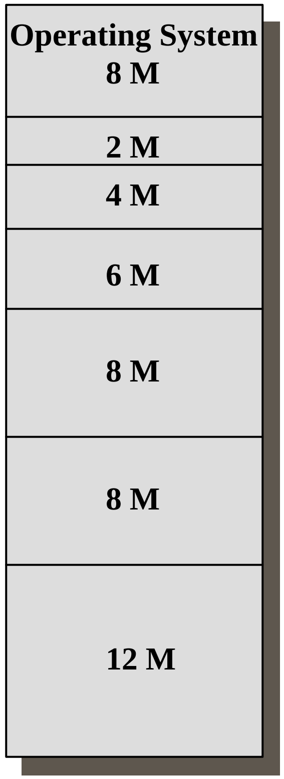
Fixed Partitioning

- Inefficient use of main memory
  - Any program, no matter how small, occupies an entire partition. This is called internal fragmentation.



Fixed Partitioning

- Unequal-size partitions
  - lessens the problem with equal-size partitions

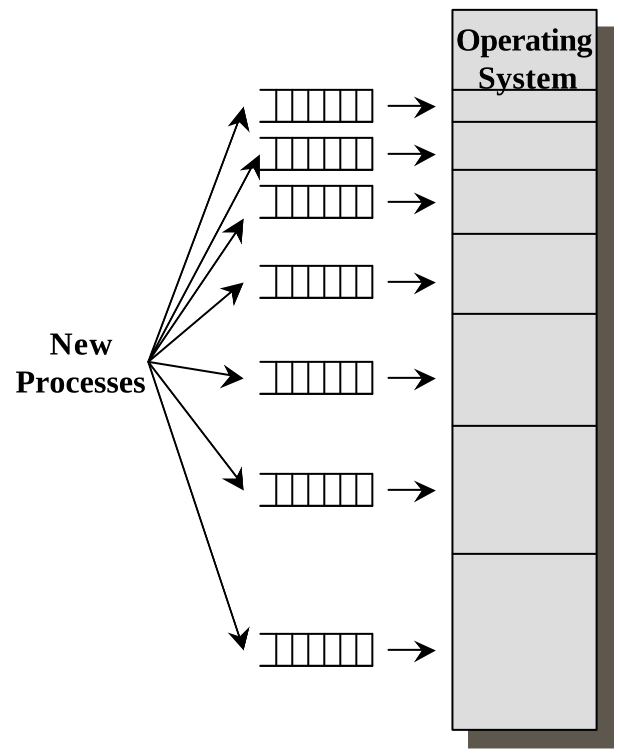


Placement Algorithm with Partitions

- Equal-size partitions
  - 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
    - processes are assigned in such a way as to minimize wasted memory within a partition

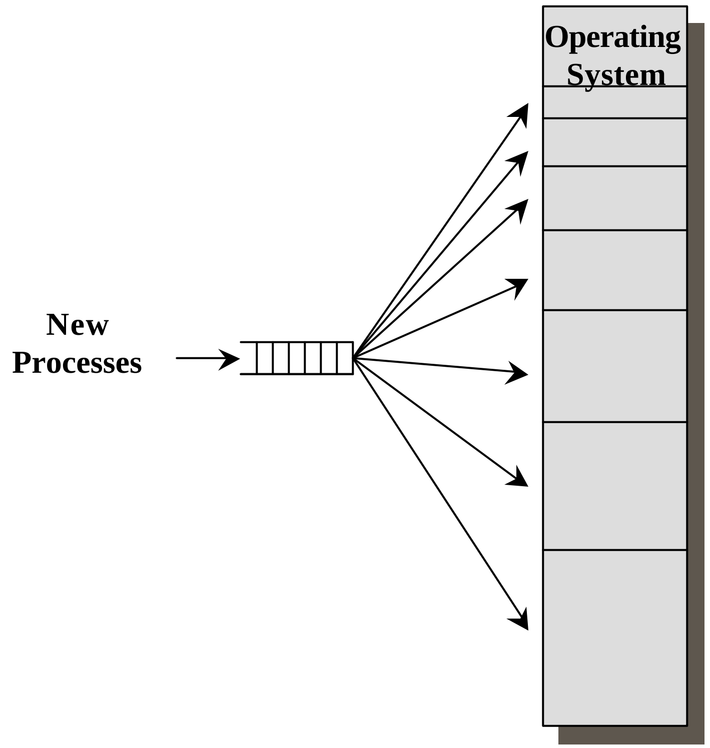
One Process Queue per Partition

- Assign each process to the smallest partition within which it will fit
- Assume that one knows the maximum amount of memory that a process will require



Single Process Queue

- When its time to load a process into main memory the smallest available partition that will hold the process is selected



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

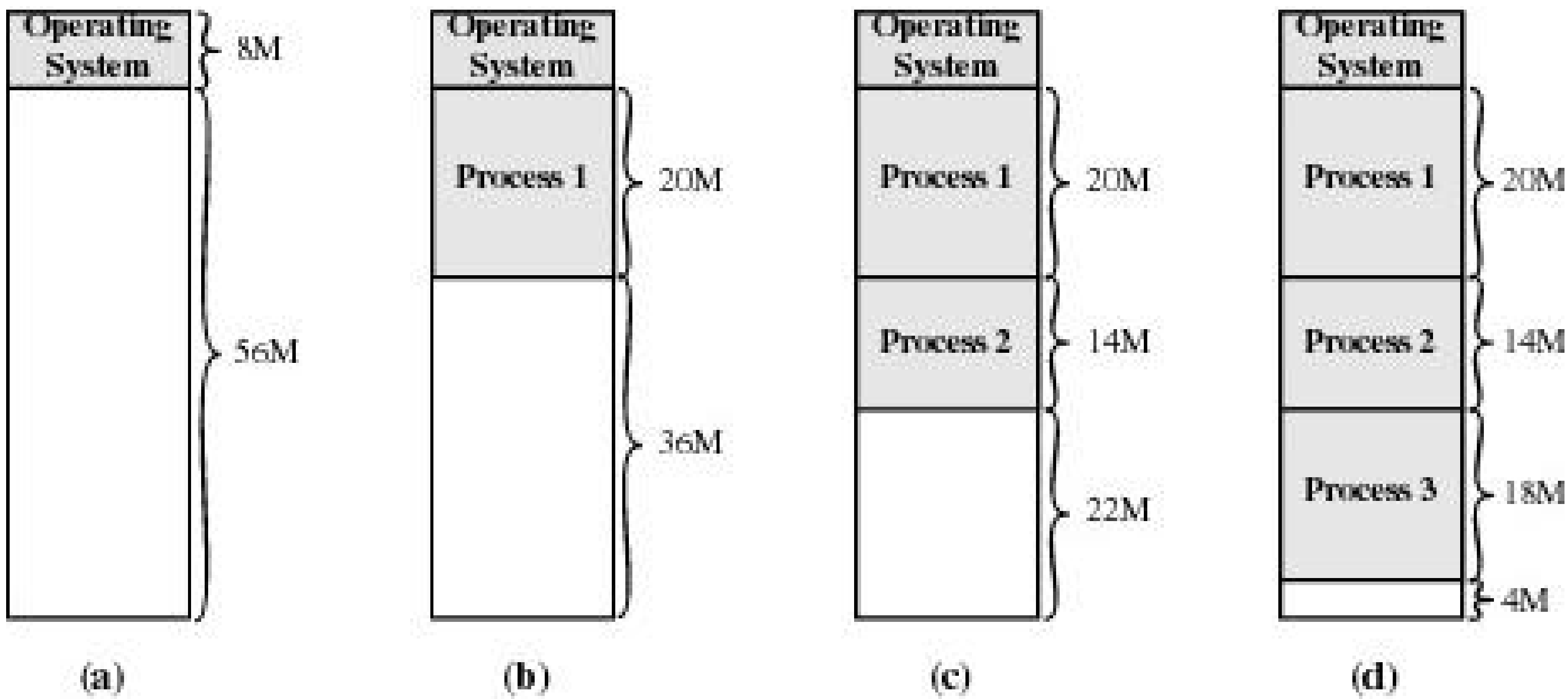


Figure 7.4 The Effect of Dynamic Partitioning

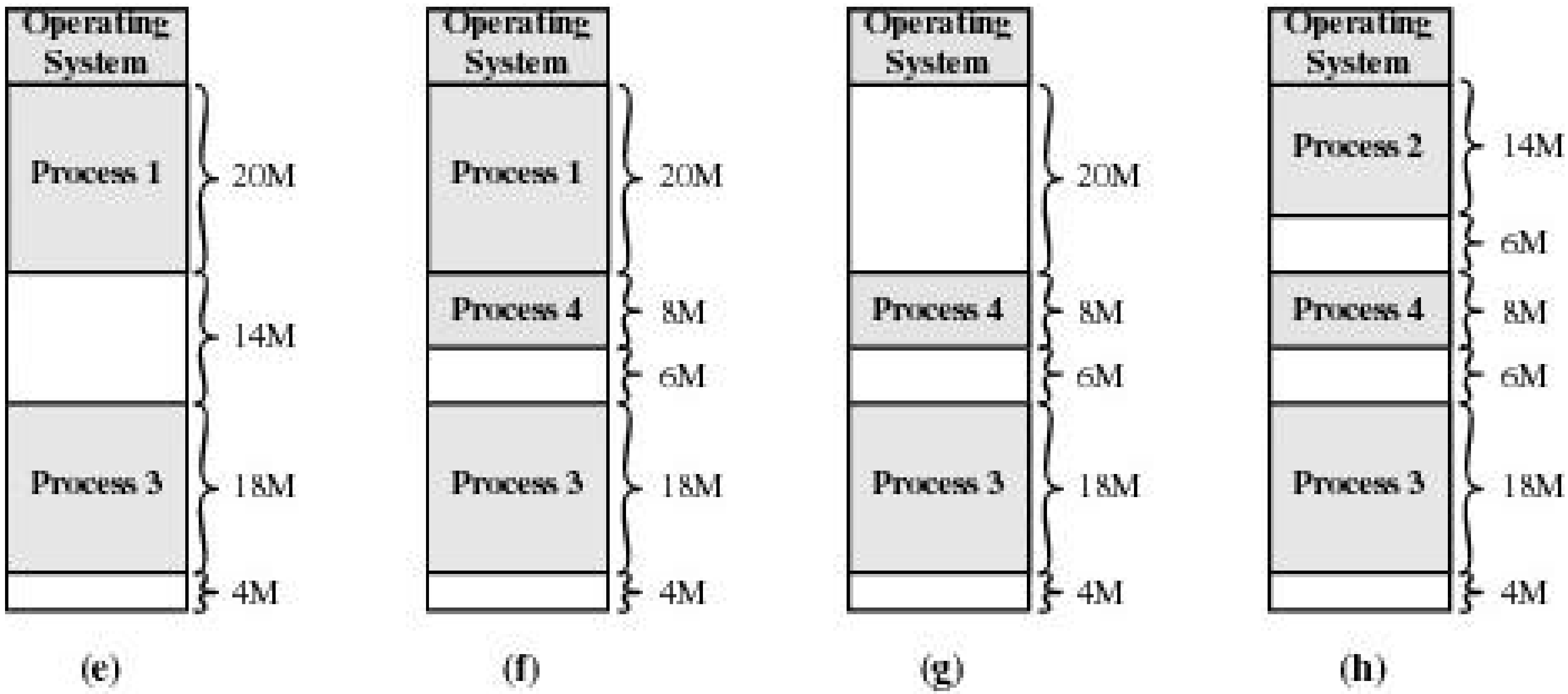


Figure 7.4 The Effect of Dynamic Partitioning

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
  - starts scanning 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 Placement Algorithm

- Next-fit algorithm
  - starts scanning memory from the location of the last placement and chooses the 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

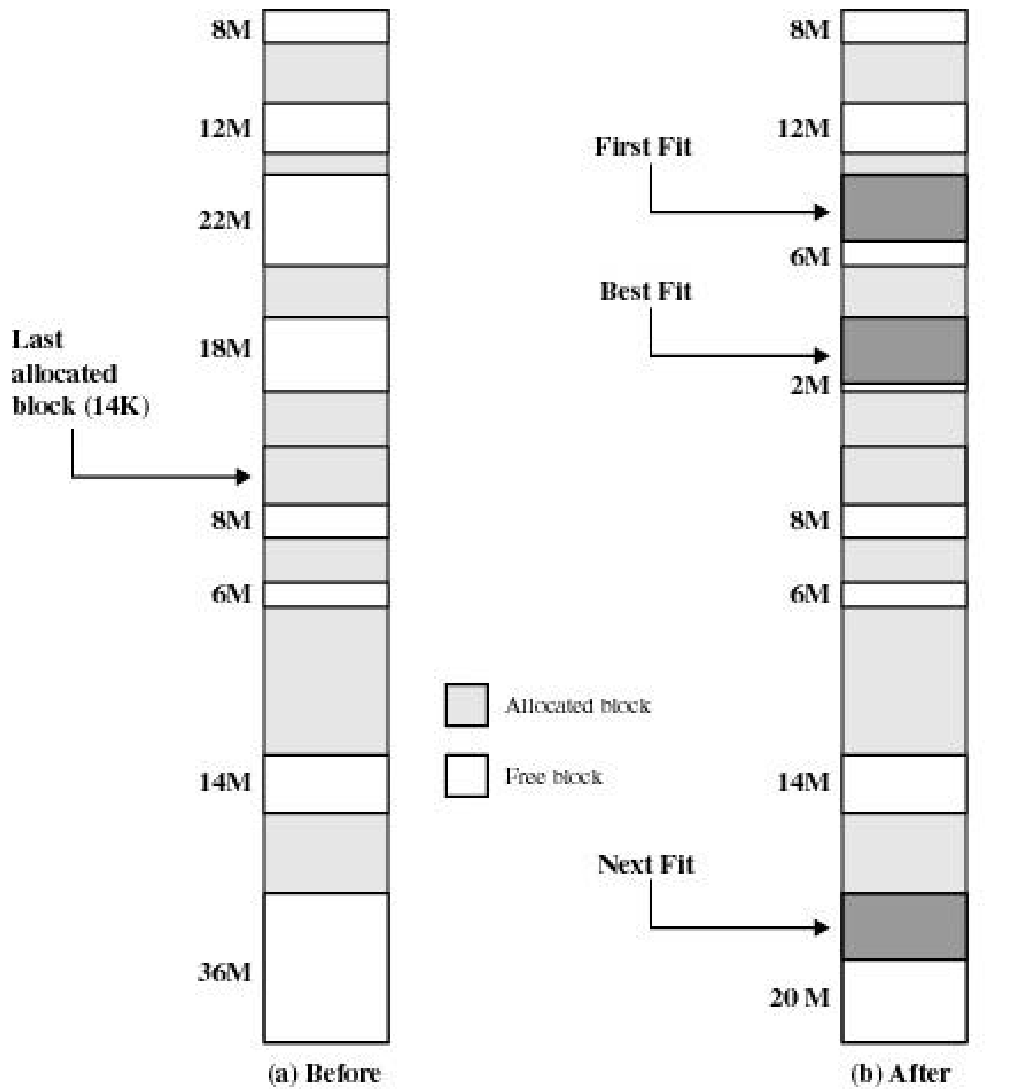


Figure 7.5 Example Memory Configuration Before and After Allocation of 16 Mbyte Block

Buddy System

- Problems of fixed and dynamic partitioning
  - limit the number of active processes and use space inefficiently
  - more complex to maintain and the overhead of compaction
- Buddy system as a compromise

Buddy System

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

Buddy System

```
procedure get_hole(i);
begin
    if (i == U + 1) then failure;
    if (i_list  empty) then begin
        get hole(i + 1);
        split hole into buddies;
        put buddies on i_list;
    end;
    take first hole on i_list;
end;
```

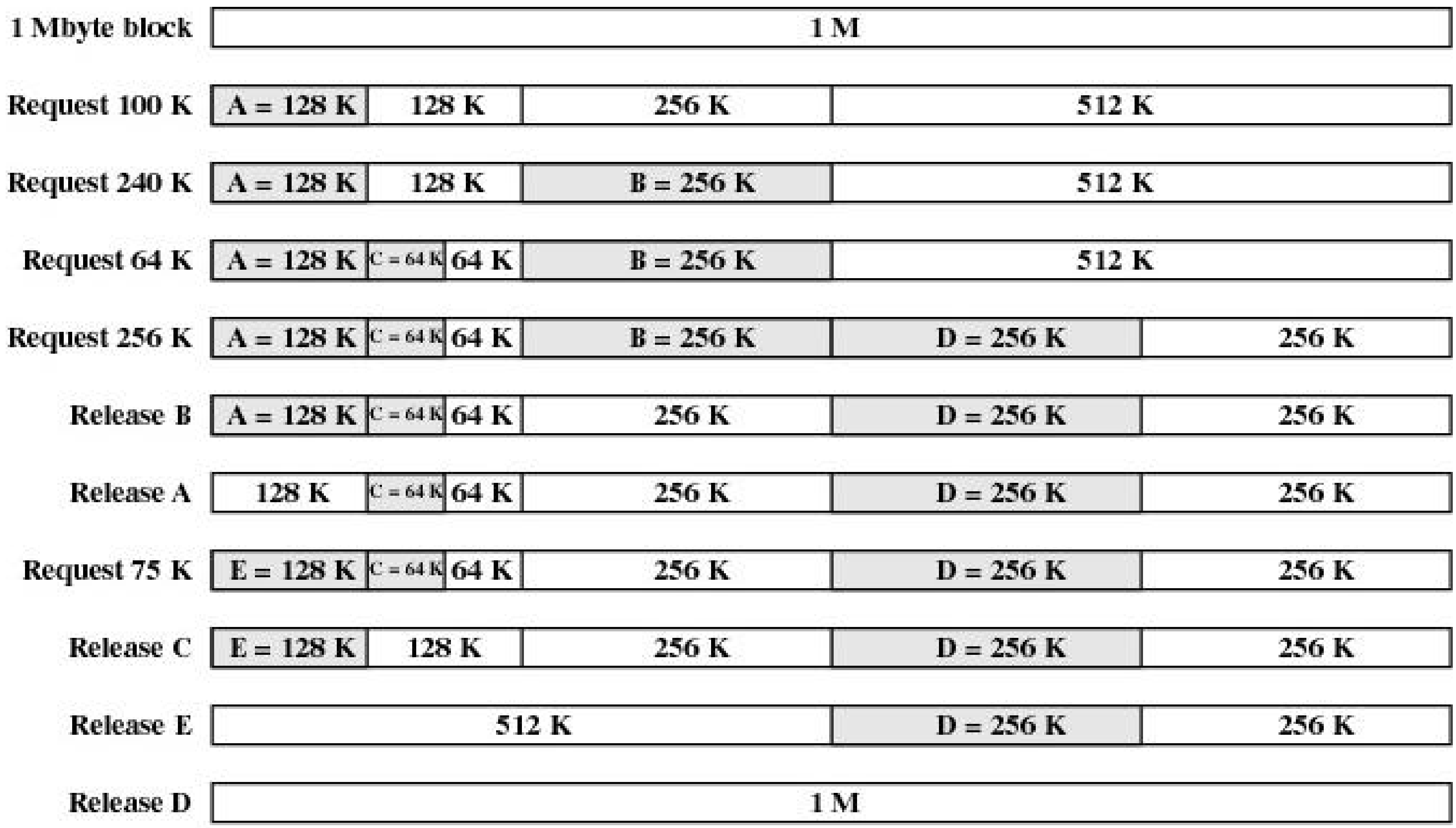


Figure 7.6 Example of Buddy System

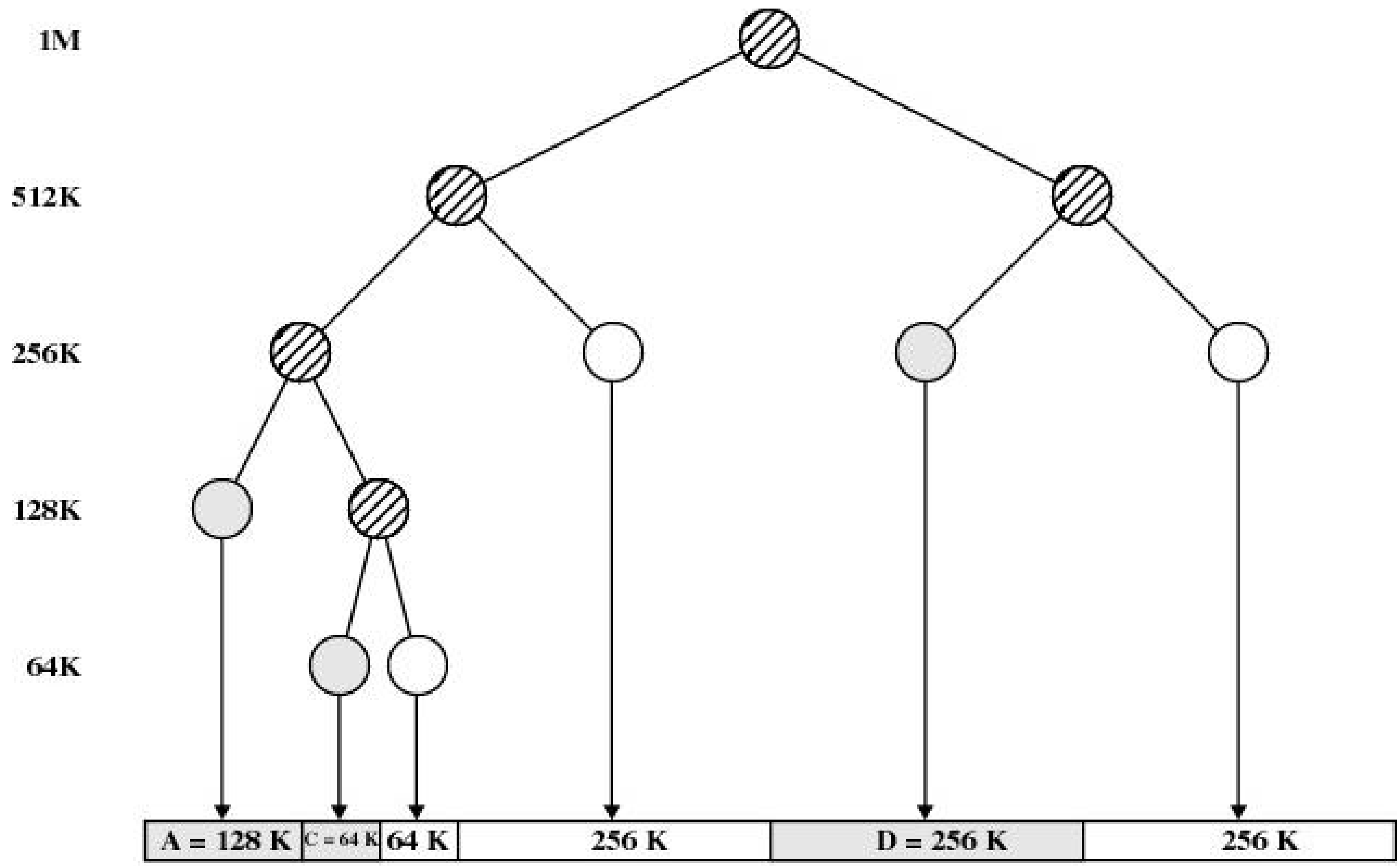


Figure 7.7 Tree Representation of Buddy System

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

Addresses

- Logical address
  - reference to a memory location independent of the current assignment of data to memory
  - translation must be made to the physical address
- Relative address
  - an example of logical address
  - address expressed as a location relative to some known point
- Physical
  - the absolute address or actual location

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

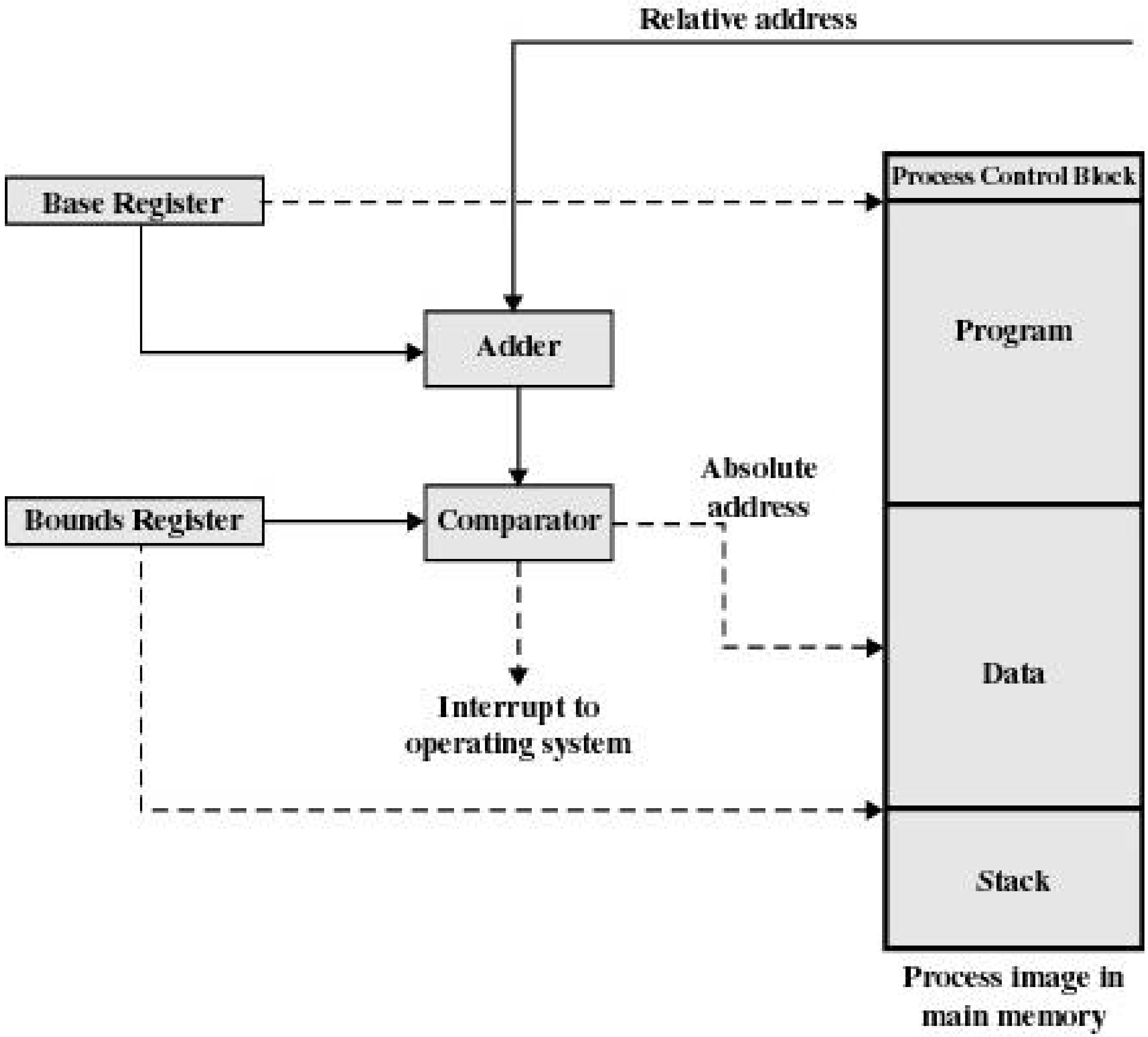


Figure 7.8 Hardware Support for Relocation

Paging

- Partition memory into small equal-size chunks and divide each process into the same size chunks
- The chunks of a process are called <u>pages</u> and chunks of memory are called <u>page frames</u>
- 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

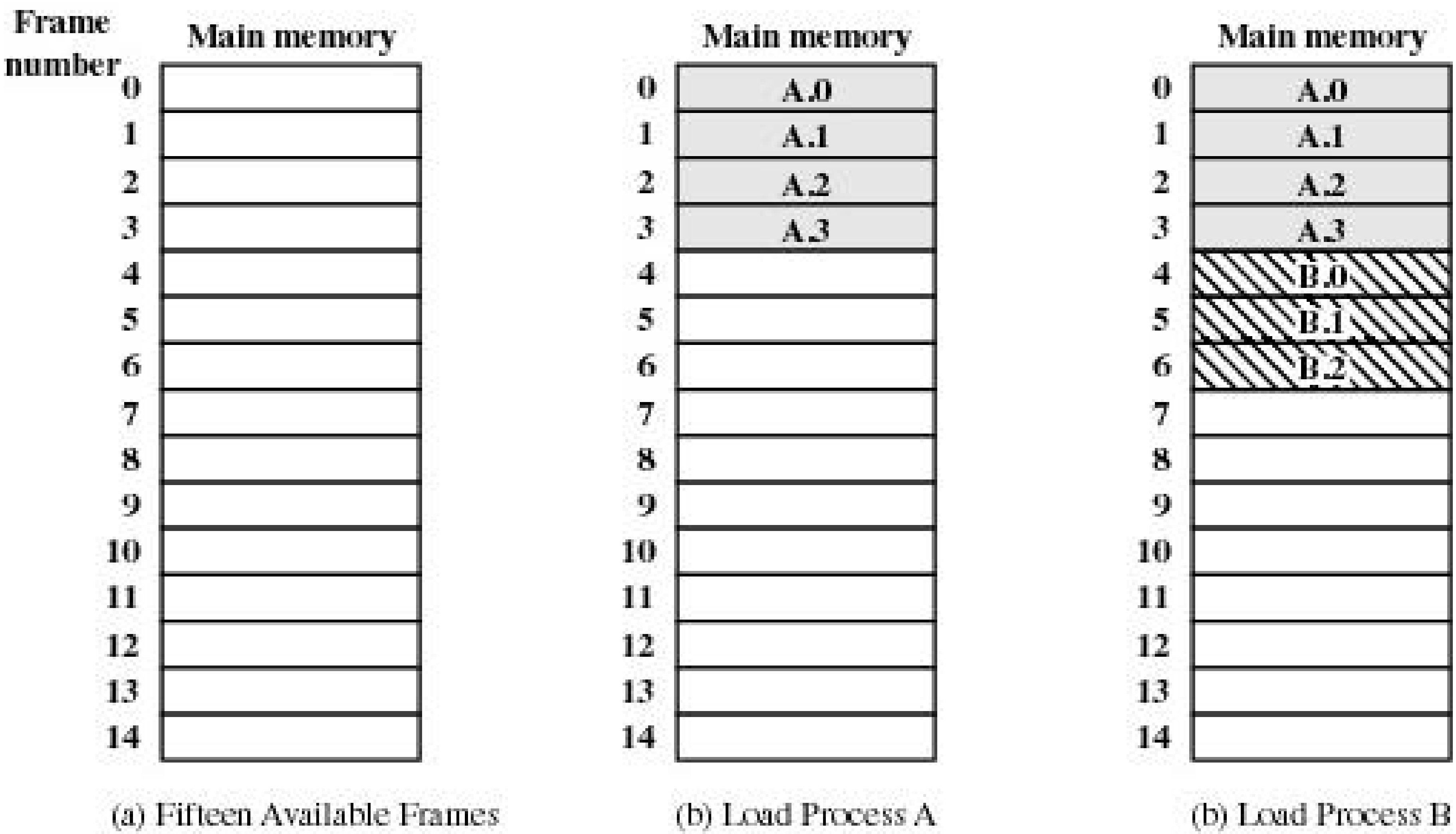


Figure 7.9 Assignment of Process Pages to Free Frames

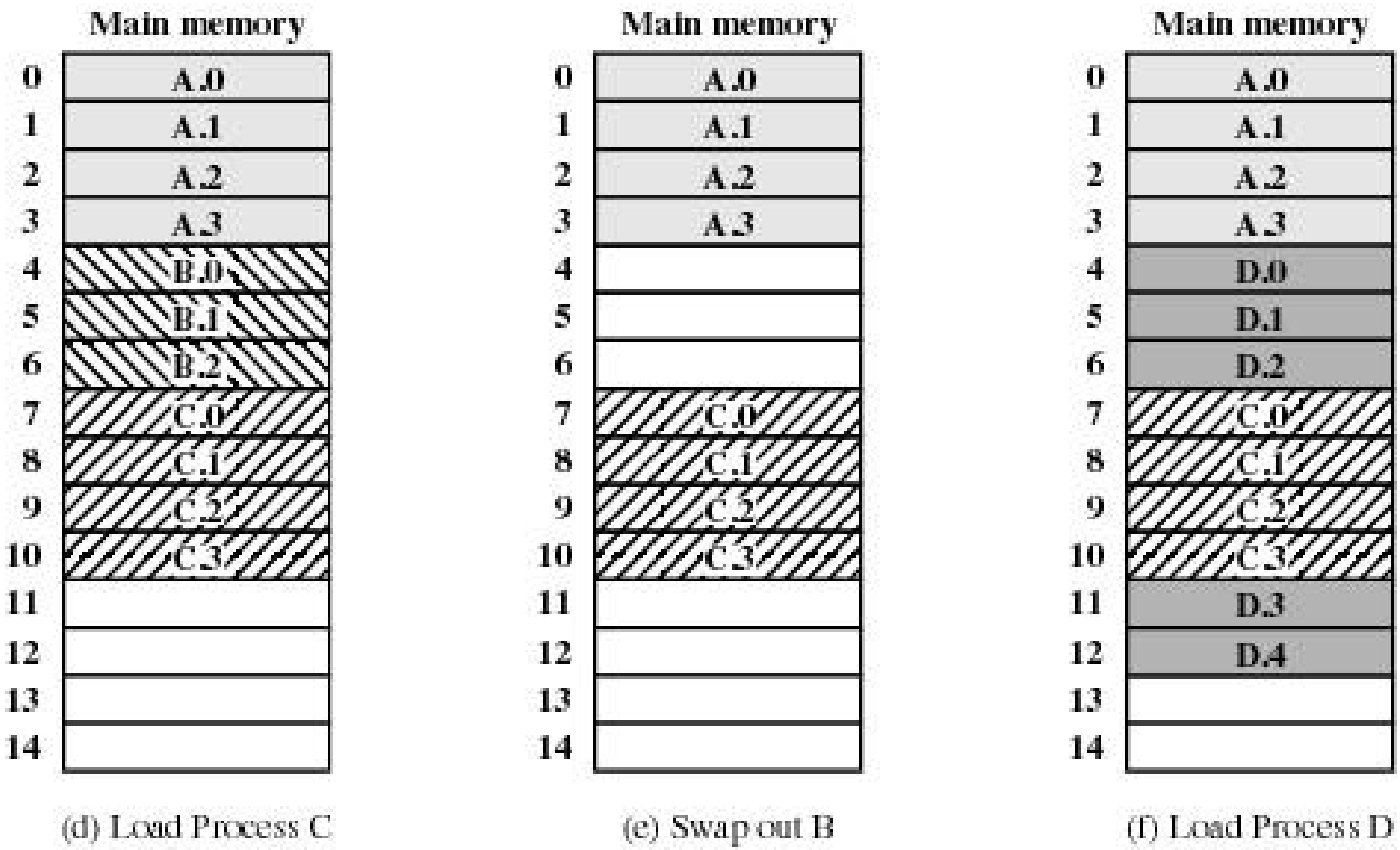


Figure 7.9 Assignment of Process Pages to Free Frames

Page Tables for Example

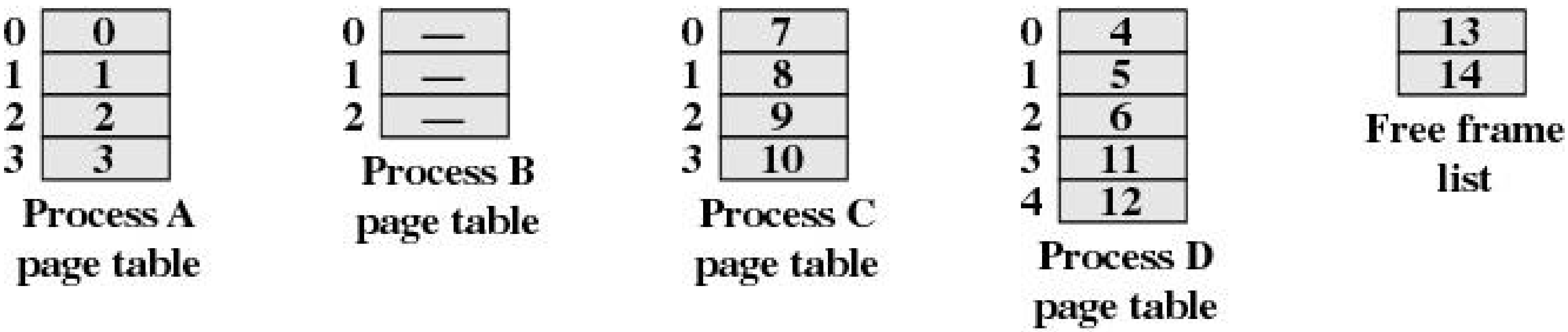


Figure 7.10 Data Structures for the Example of Figure 7.9 at Time Epoch (f)

An Example(paging)

- Consider an address of n+m bits
  - leftmost n bits are the page number
  - rightmost m bits are the offset
- 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
  - starting physical address of the frame is k x 2^m
    - the physical address can easily be constructed by appending the frame number to the offset

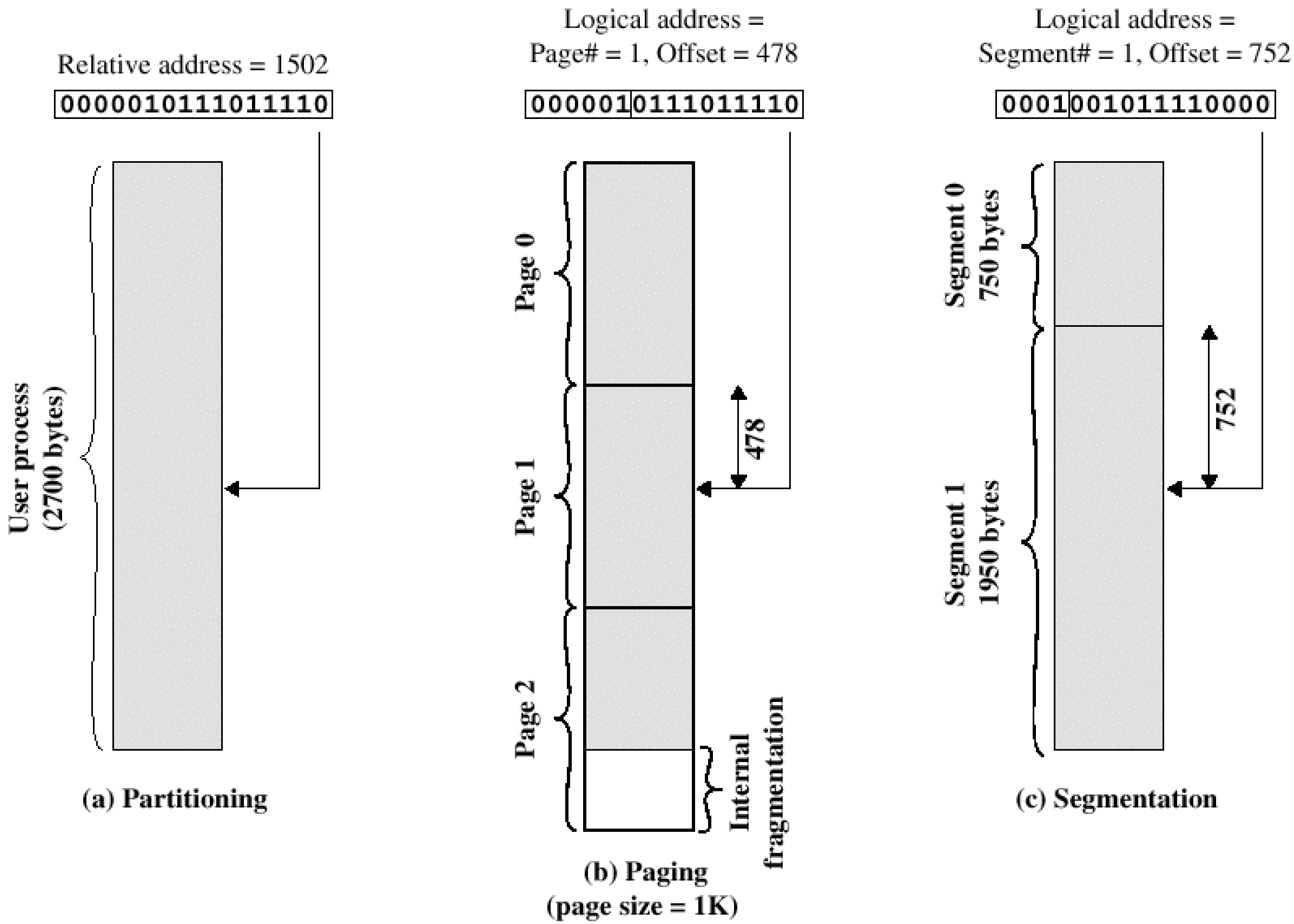


Figure 7.11 Logical addresses

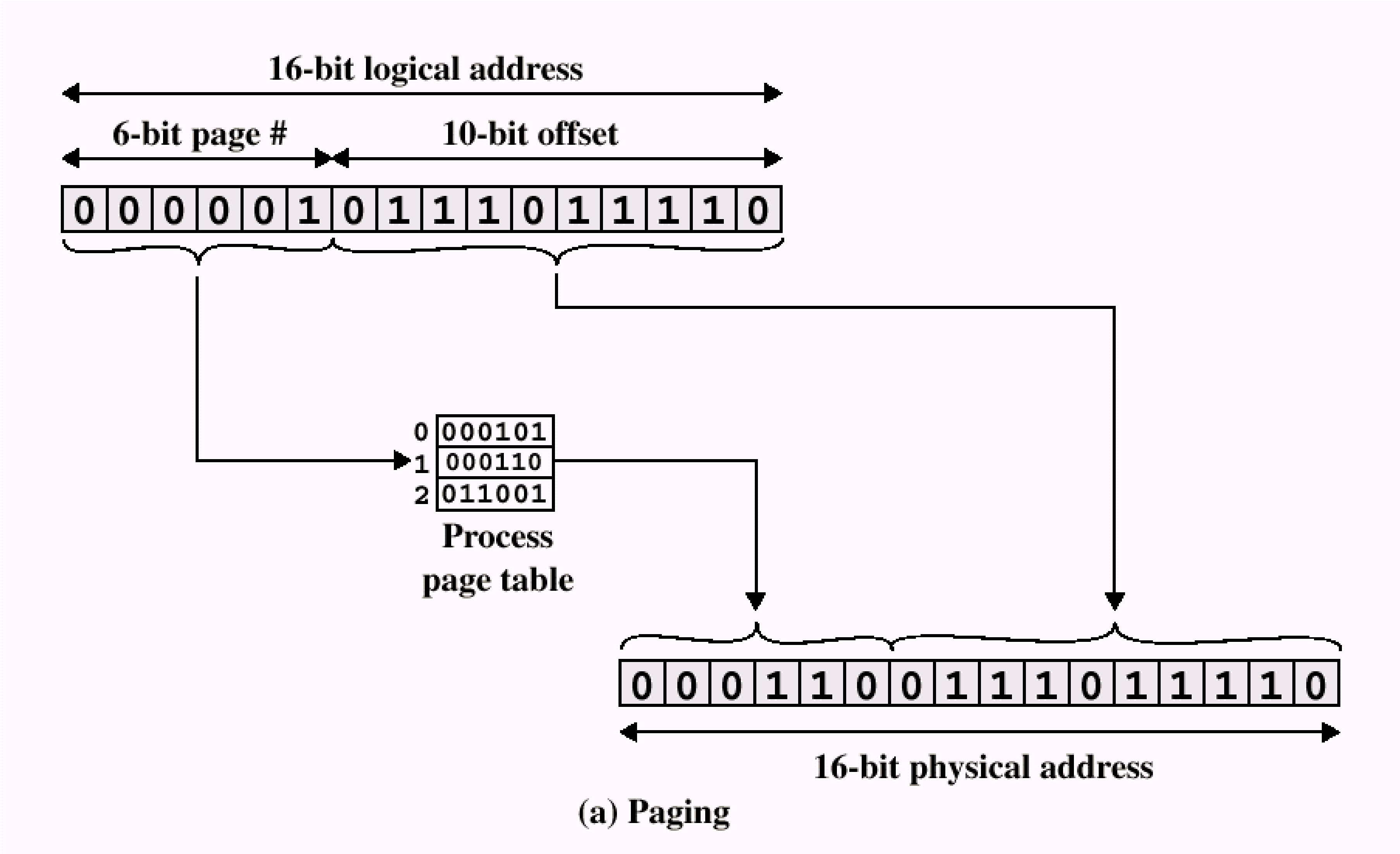


Figure 7.12 Examples of Logical-to-Physical address translation

Segmentation

- All segments of a process do not have to be of the same length
  - text, data, stack, PCB, shared memory…
- Each process has a segment table
- Addressing consist of two parts
  - a segment number and an offset

An Example(segmentation)

- Consider an address of n+m bits
  - leftmost n bits are the page number
  - rightmost m bits are the offset
- 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
  - compare the offset to the length of the segment
  - the physical address is the sum of the starting physical address plus the offset

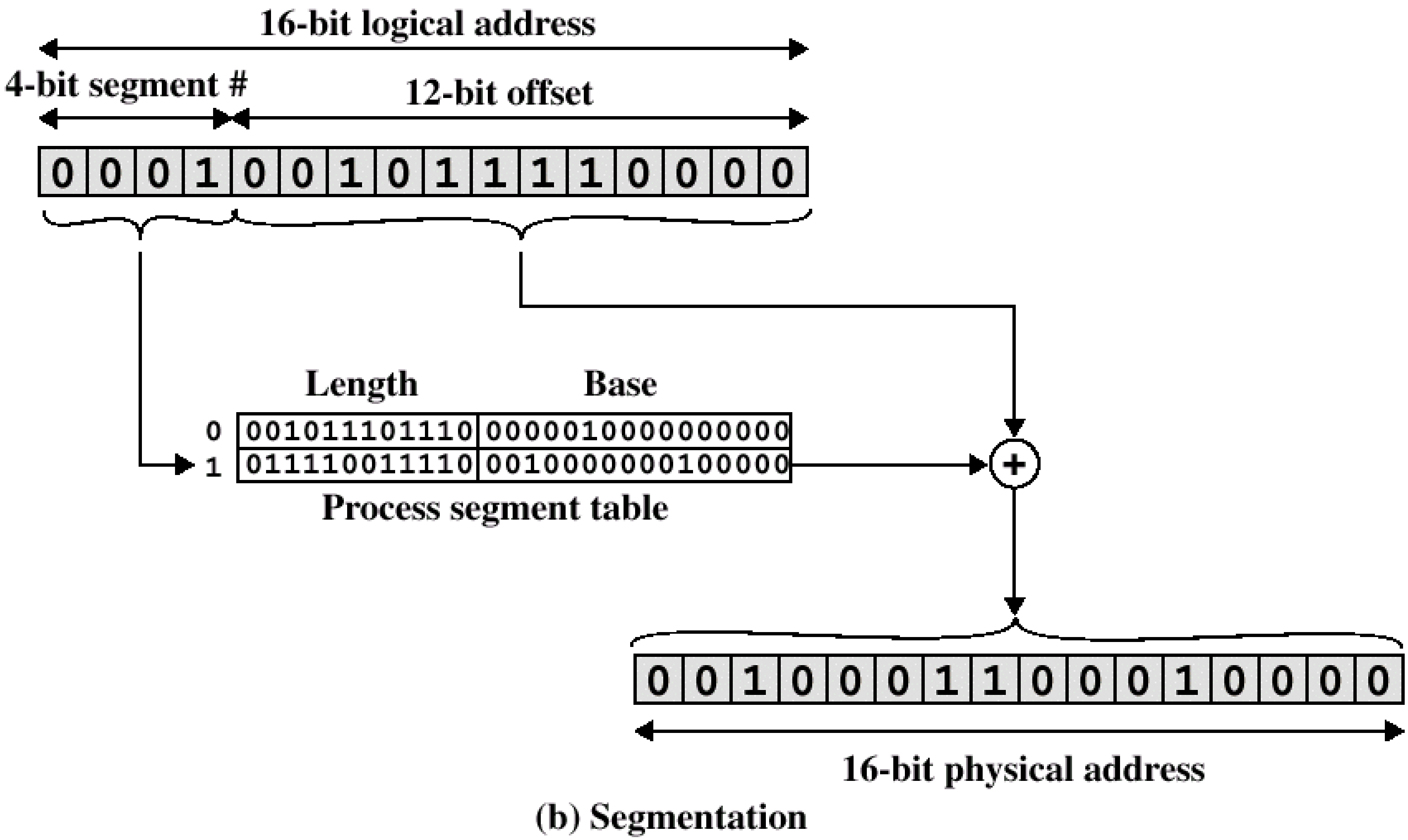


Figure 7.12 Examples of Logical-to-Physical address translation