Memory Management

Chapter 7

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

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
  - Memory protection requirement must be satisfied by the processor (hardware) rather than the operating system (software)
    - 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 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 (cross refs solved at runtime)
  - Different degrees of protection given to modules (read-only, execute-only)
  - Share modules among processes

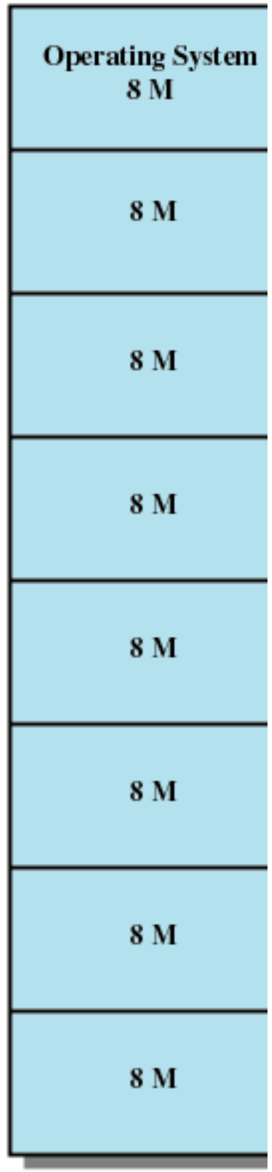
Memory Management Requirements

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

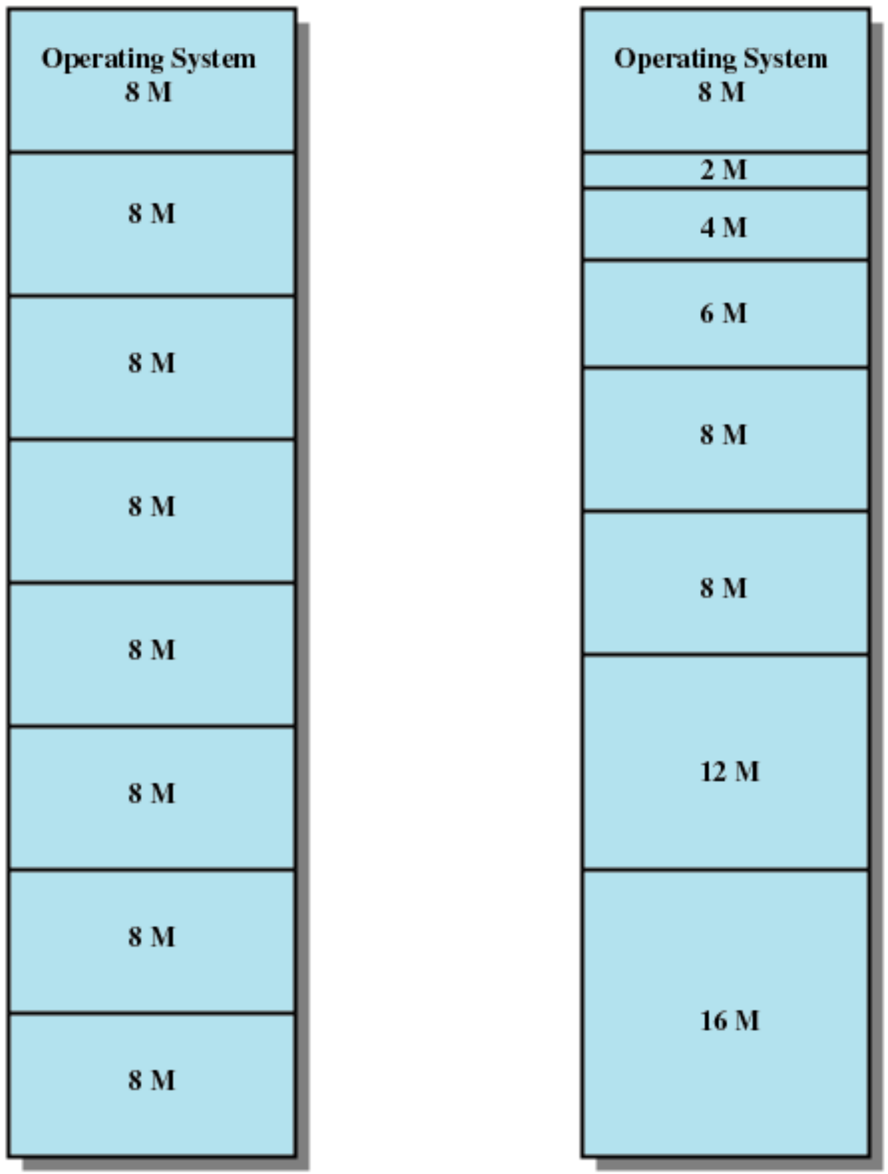
Fixed Partitioning

- 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
  - A process occupies an entire partition!

    => **Internal Fragmentation**



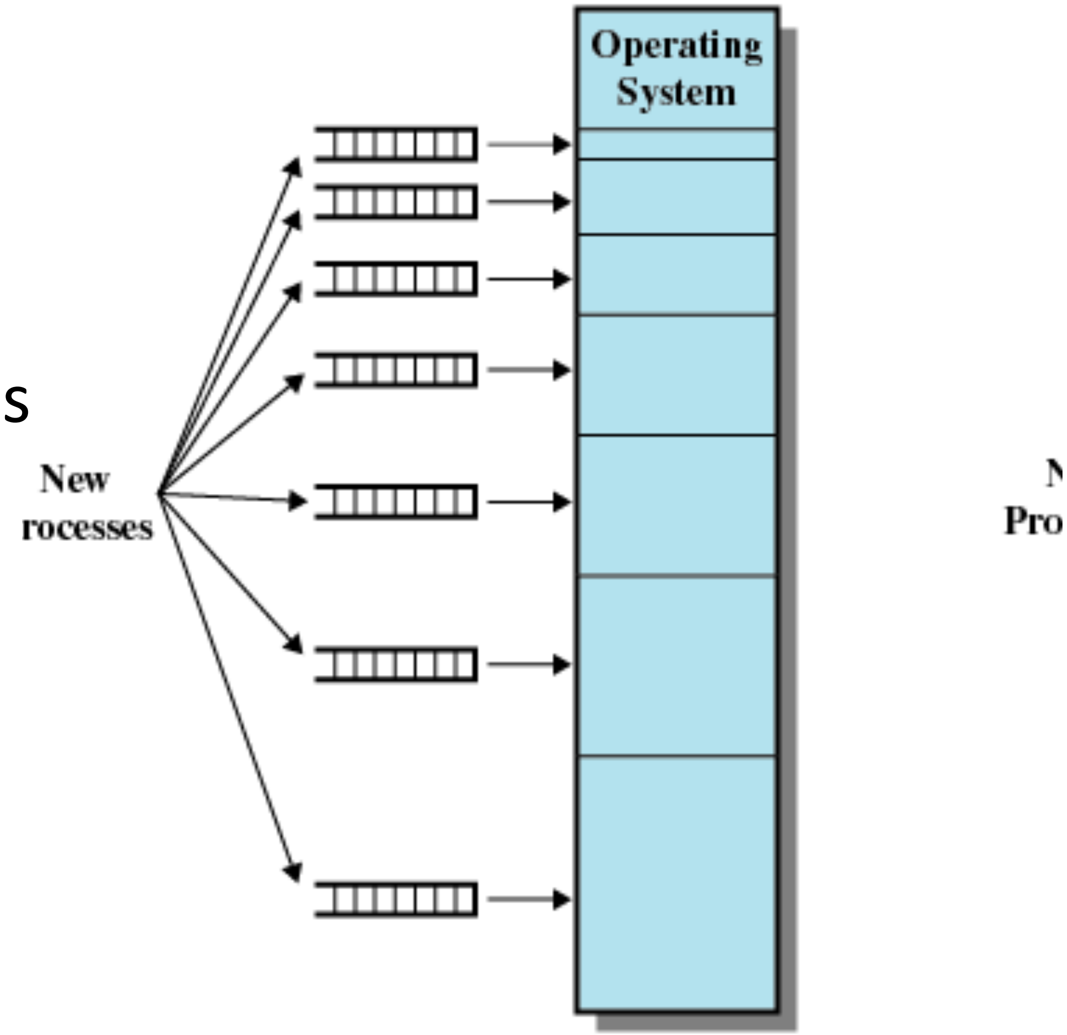
Equal VS Unequal Size Partitioning:



Placement Algorithm with Partitions

- Equal-size partitions
  – Pick one available at random (not important)
- 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

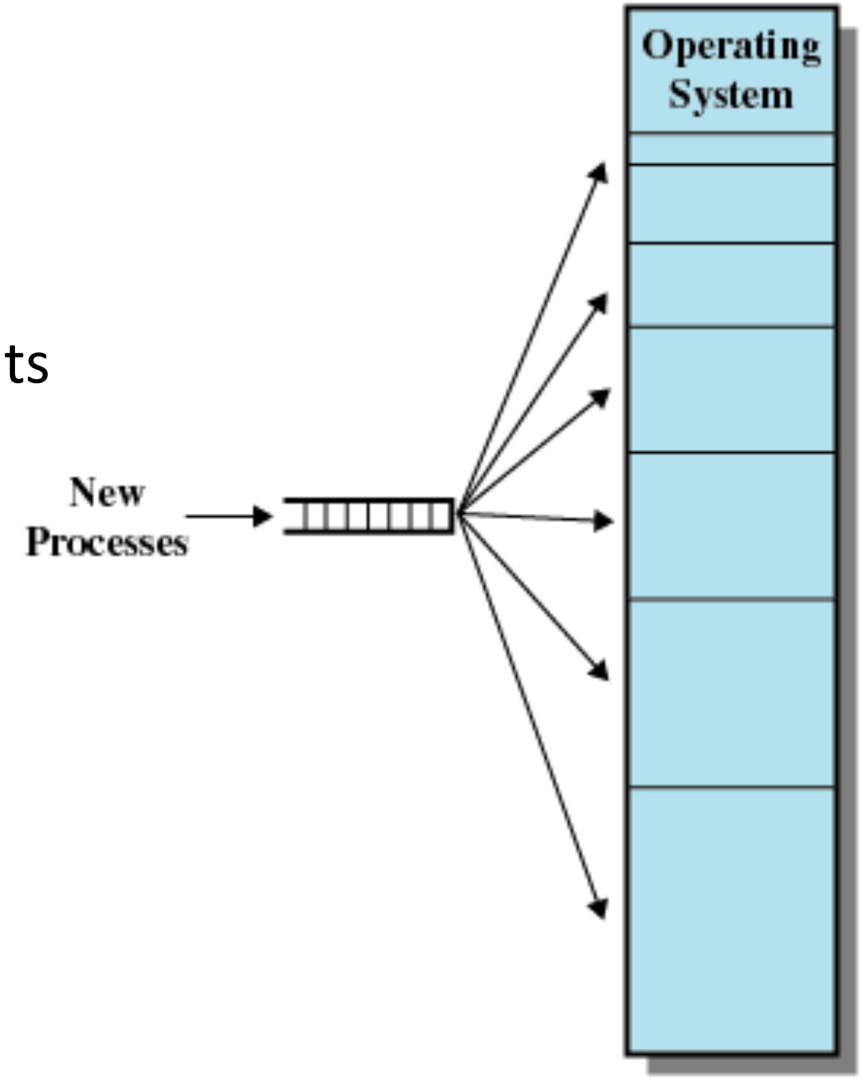
- 1 queue x partition
  - Swapped out processes
- Inefficient for
  - Many small processes
  - Big partitions



(a) One process queue per partition

Figure 7.3 Memory Assignment f

- 1 queue x ALL partitions
  - Swapped out processes
- Select smallest partition that fits process
- Swap out from the smallest partition that fits process



ɔn

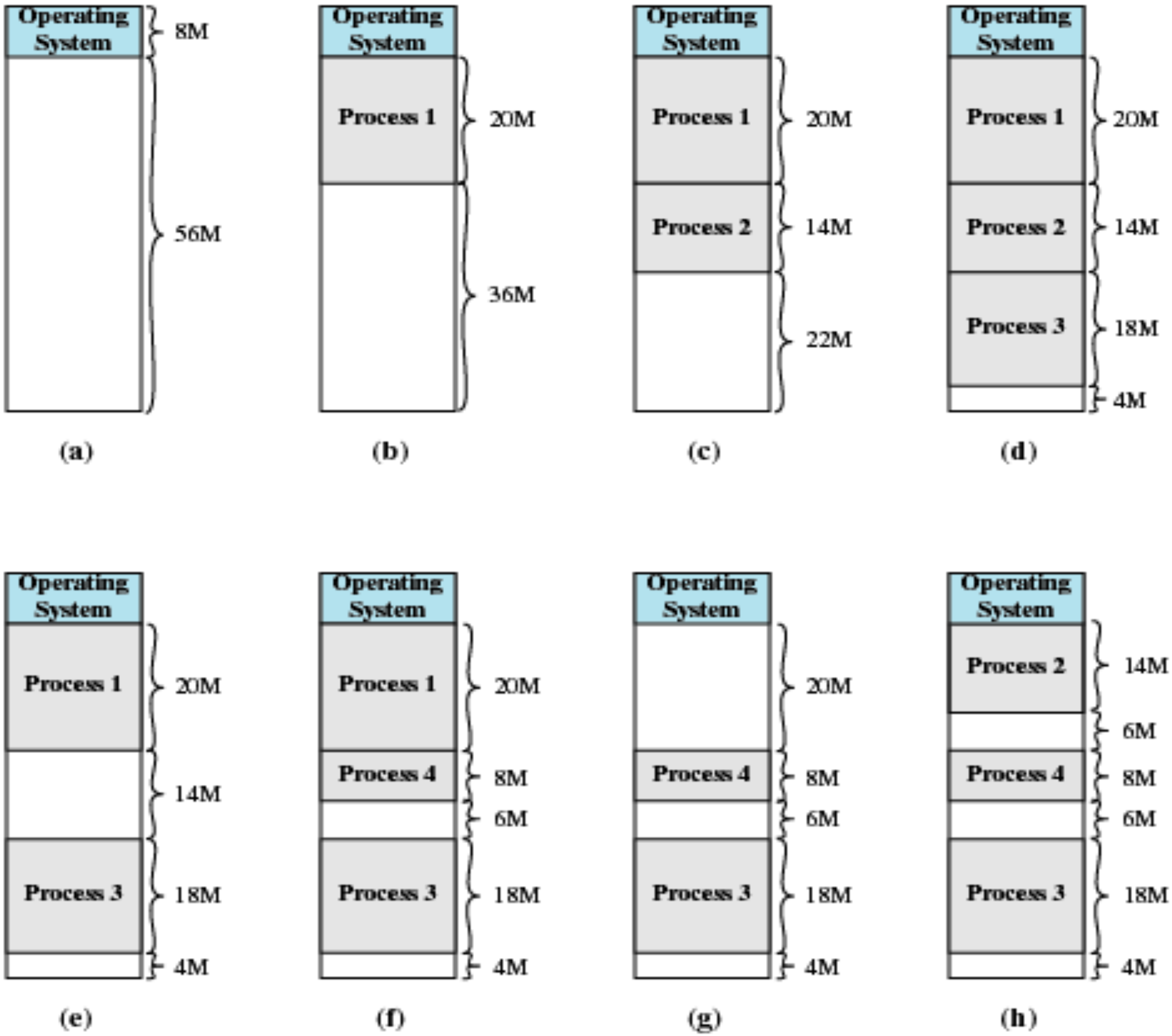
(b) Single queu

Fixed partitioning: problems

- Partition nr limits active process nr
- Need to know min and max job size beforehand
- Not good for small jobs & very big jobs

Dynamic Partitioning

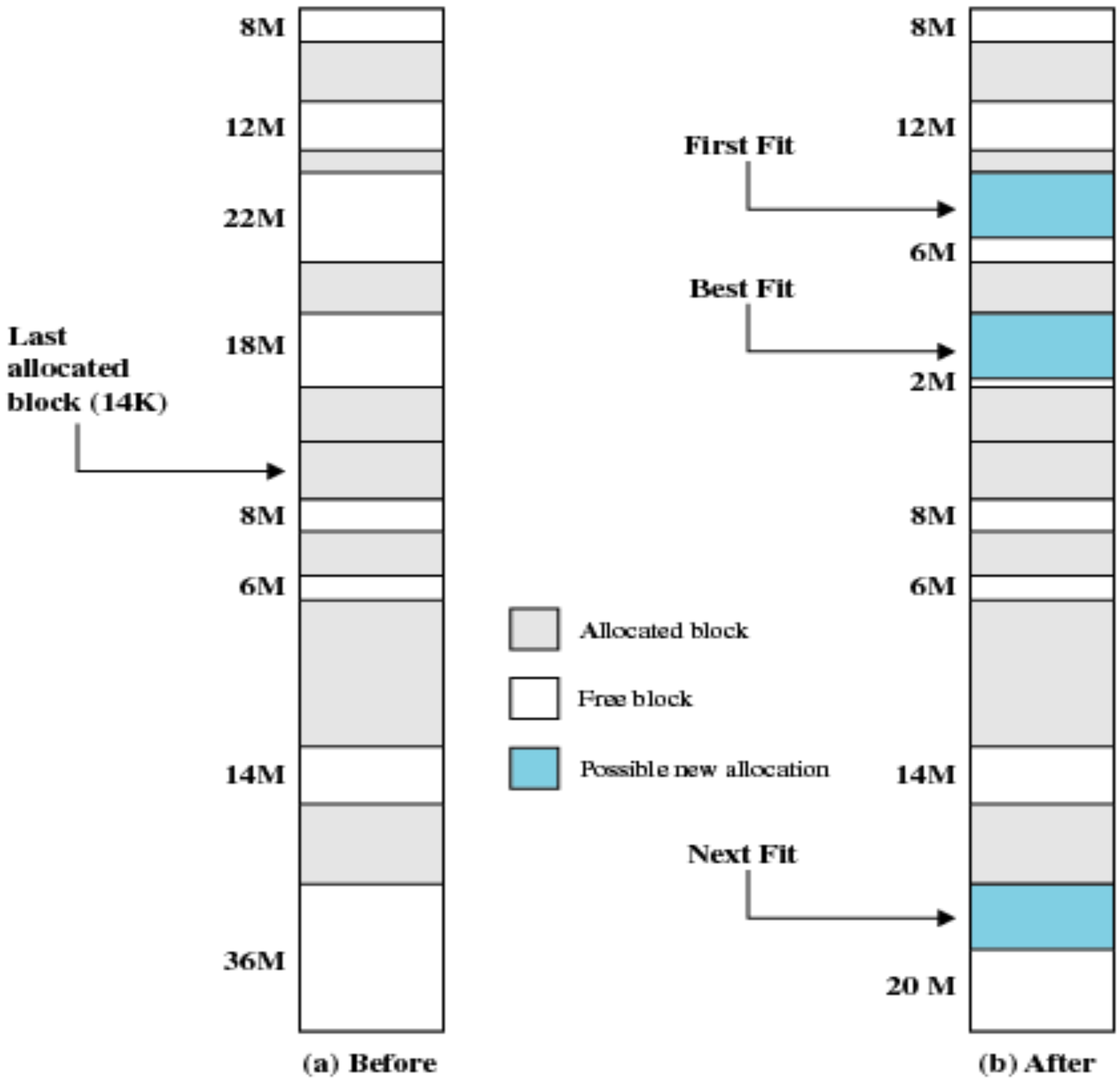
- Partitions are of variable length and number
- Process is allocated exactly as much memory as required
- Eventually get holes in the memory. This is called external fragmentation
- Must use compaction to shift processes so they are contiguous and all free memory is in one block



Operating System
8M
56M
(a)
Operating System
Process 1
20M
36M
(b)
Operating System
Process 1
20M
Process 2
14M
22M
(c)
Operating System
Process 1
20M
Process 2
14M
Process 3
18M
4M
(d)
Operating System
Process 1
20M
14M
Process 3
18M
4M
(e)
Operating System
Process 1
20M
Process 4
8M
6M
Process 3
18M
4M
(f)
Operating System
20M
Process 4
8M
6M
Process 3
18M
4M
(g)
Operating System
Process 2
14M
6M
Process 4
8M
6M
Process 3
18M
4M
(h)

Dynamic Partitioning Placement Algorithm

- Operating system must decide which free block to allocate to a process
- Best-fit
  - Minimizes the remaining empty fragment
- First-Fit
  - The first which fits the process
- Next-Fit
  - The First which fits the process starting from the last allocation


8M
12M
22M
18M
Last allocated block (14K)
8M
6M
14M
36M
(a) Before
Allocated block
Free block
Possible new allocation
8M
12M
First Fit
6M
Best Fit
2M
8M
6M
14M
Next Fit
20 M
(b) After

Dynamic Partitioning Placement Algorithm

- Best-fit
  - Worst performer overall: memory fragmented with many small fragments
- First-fit
  - Fastest
  - Loads many processes in the front-end
- Next-fit
  - Breaks up the largest block into smaller ones
  - Compaction is required to obtain a large block at the end of memory

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

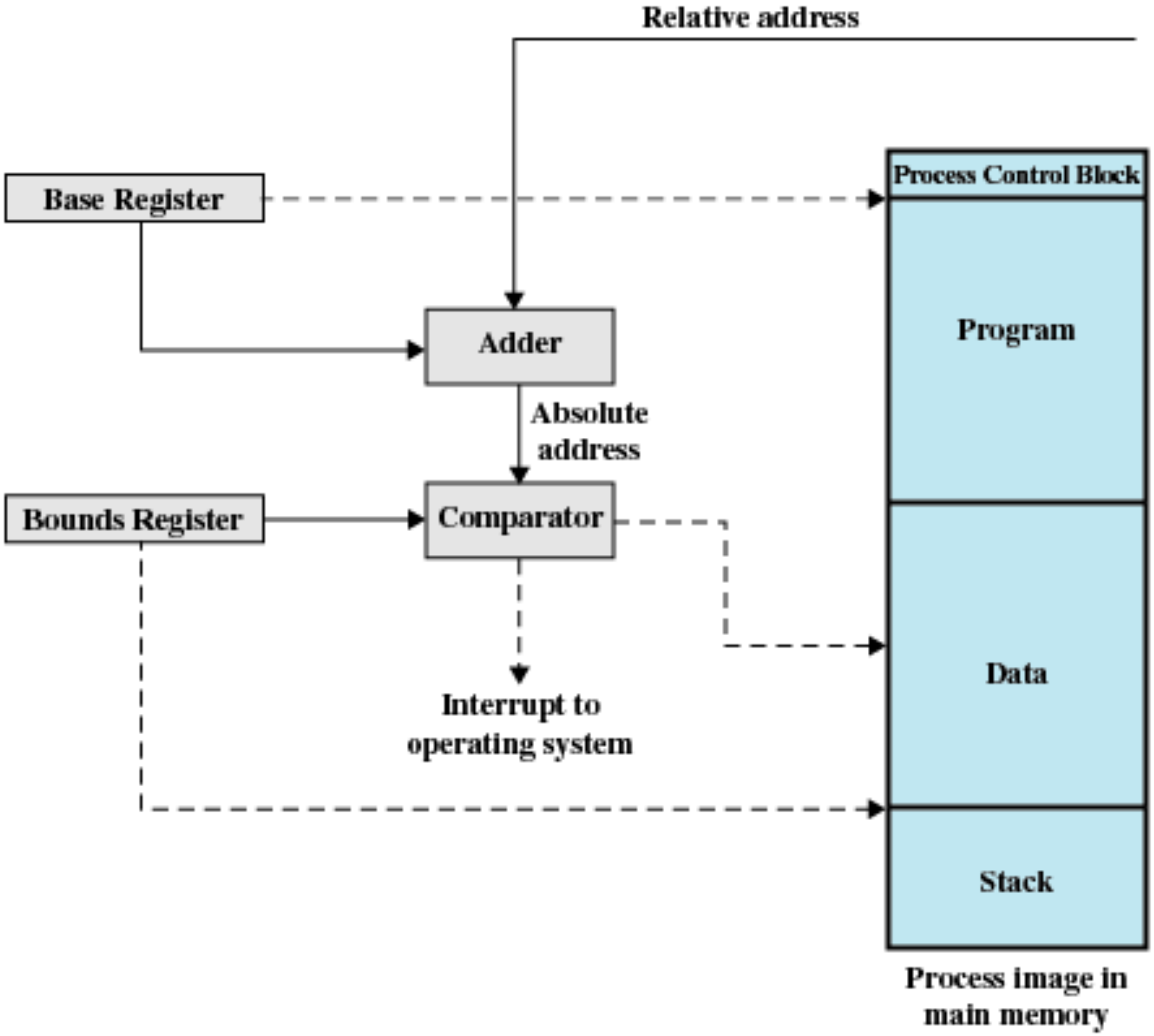
Relocation

- 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 (Particular case of above)
  - Address expressed as a location relative to some known point (e.g. starting point of program)
- Physical
  - The absolute address or actual location in main memory

=> need for hardware support to translate addresses in runtime


Relative address
Base Register
Process Control Block
Adder
Program
Absolute address
Bounds Register
Comparator
Data
Interrupt to operating system
Stack
Process image in main memory

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

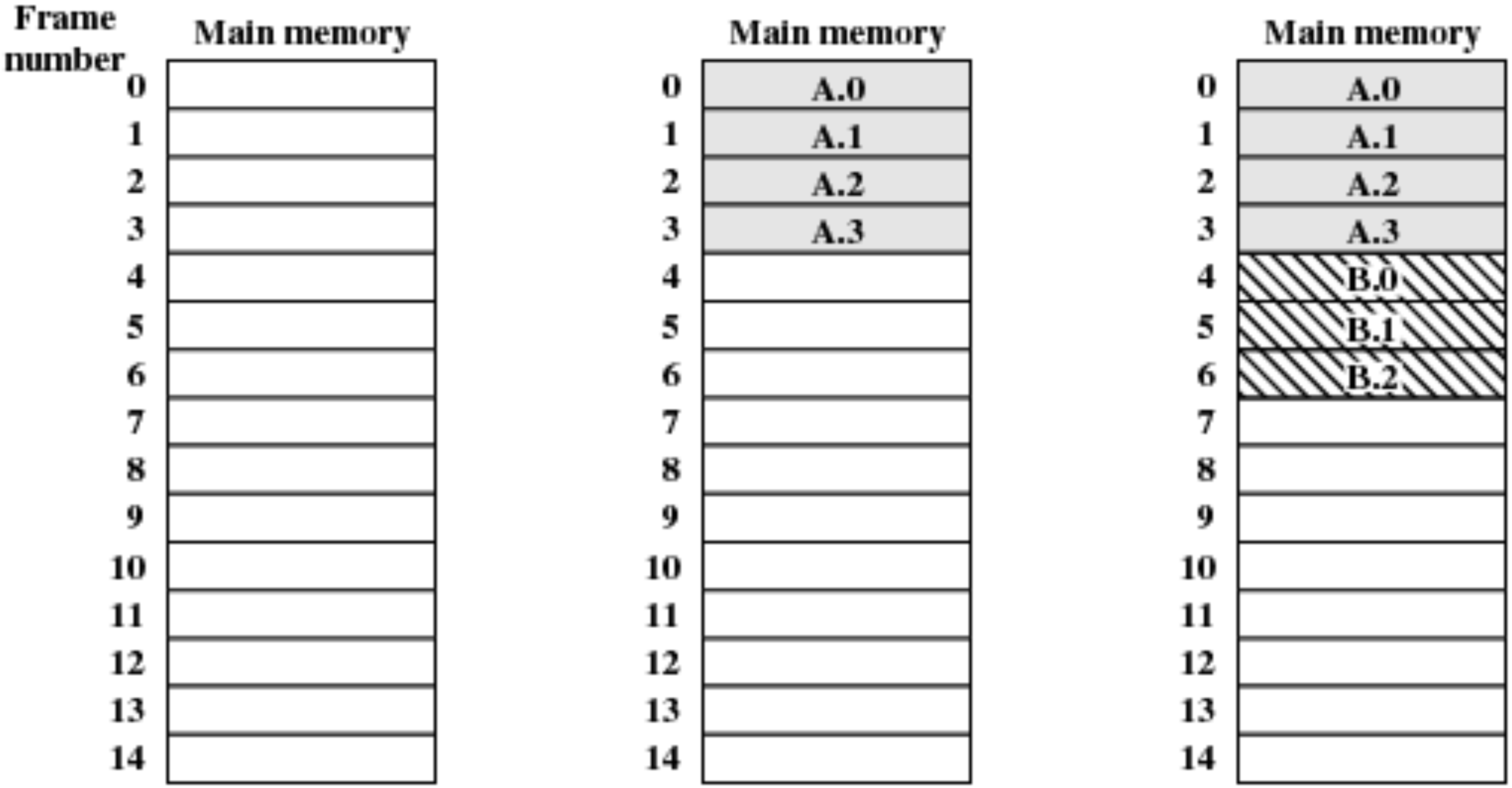
Registers Used during Execution

- 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 register not enough: OS 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

Assignment of Process Pages to Free Frames

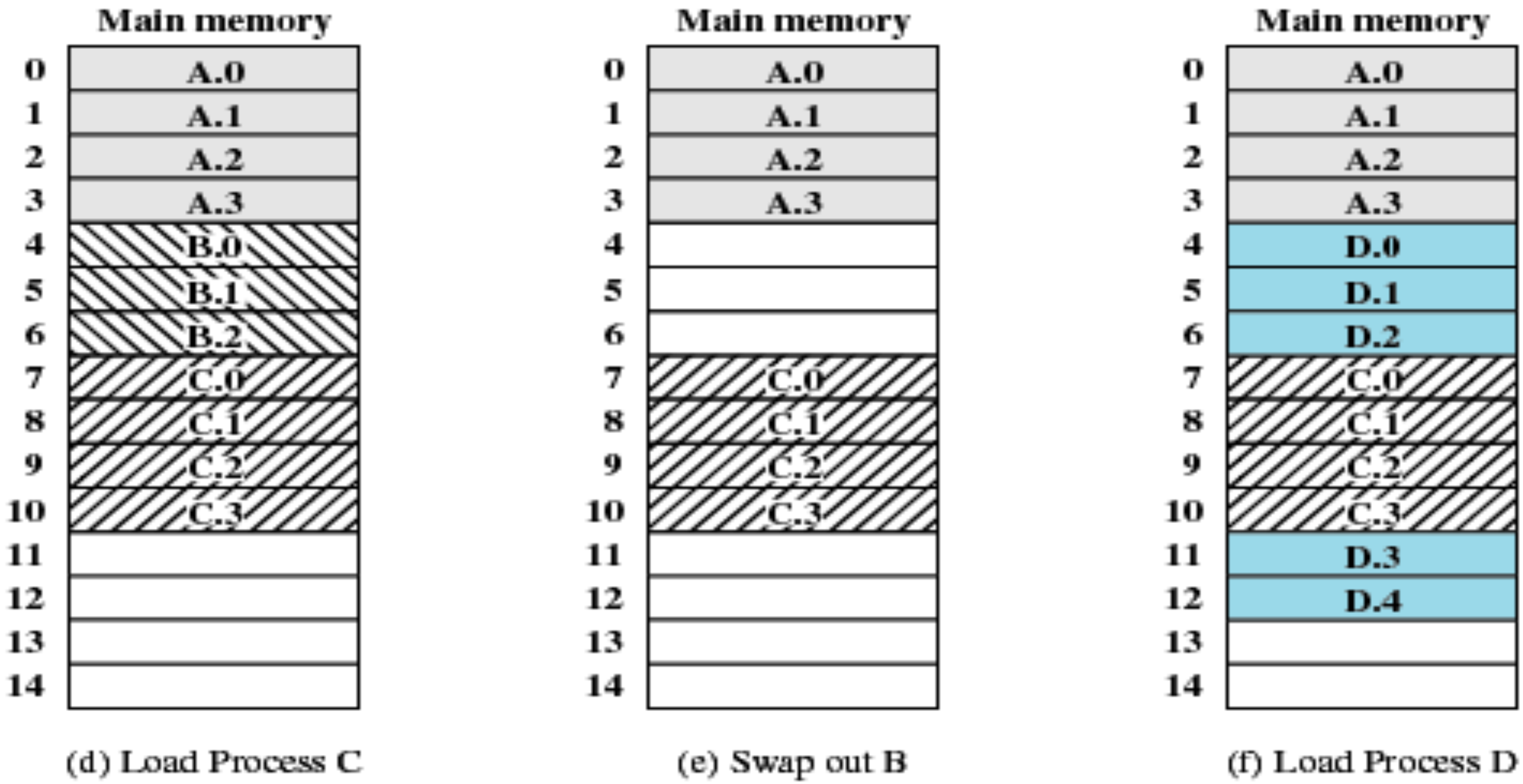


(a) Fifteen Available Frames

(b) Load Process A

(c) Load Process B

Assignment of Process Pages to Free Frames



Page Tables for Example

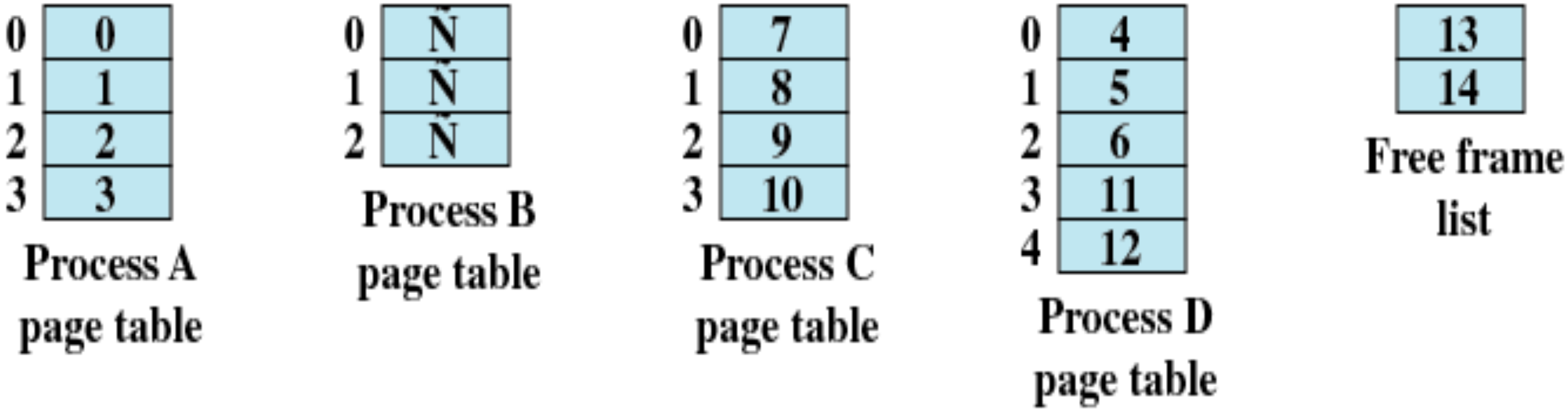
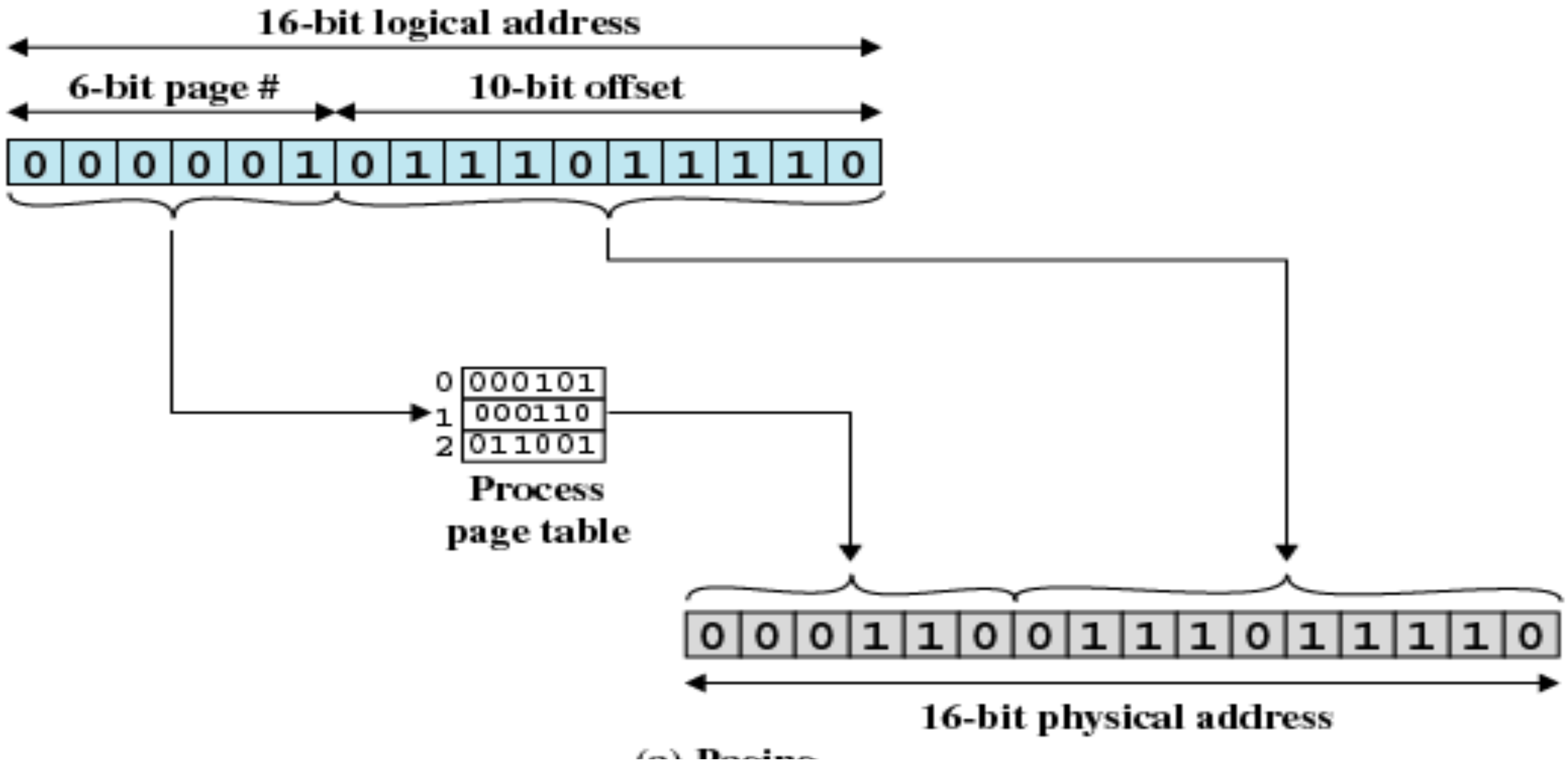


Figure 7.10 Data Structures for the Example of Figure 7.9 at Time Epoch (f)

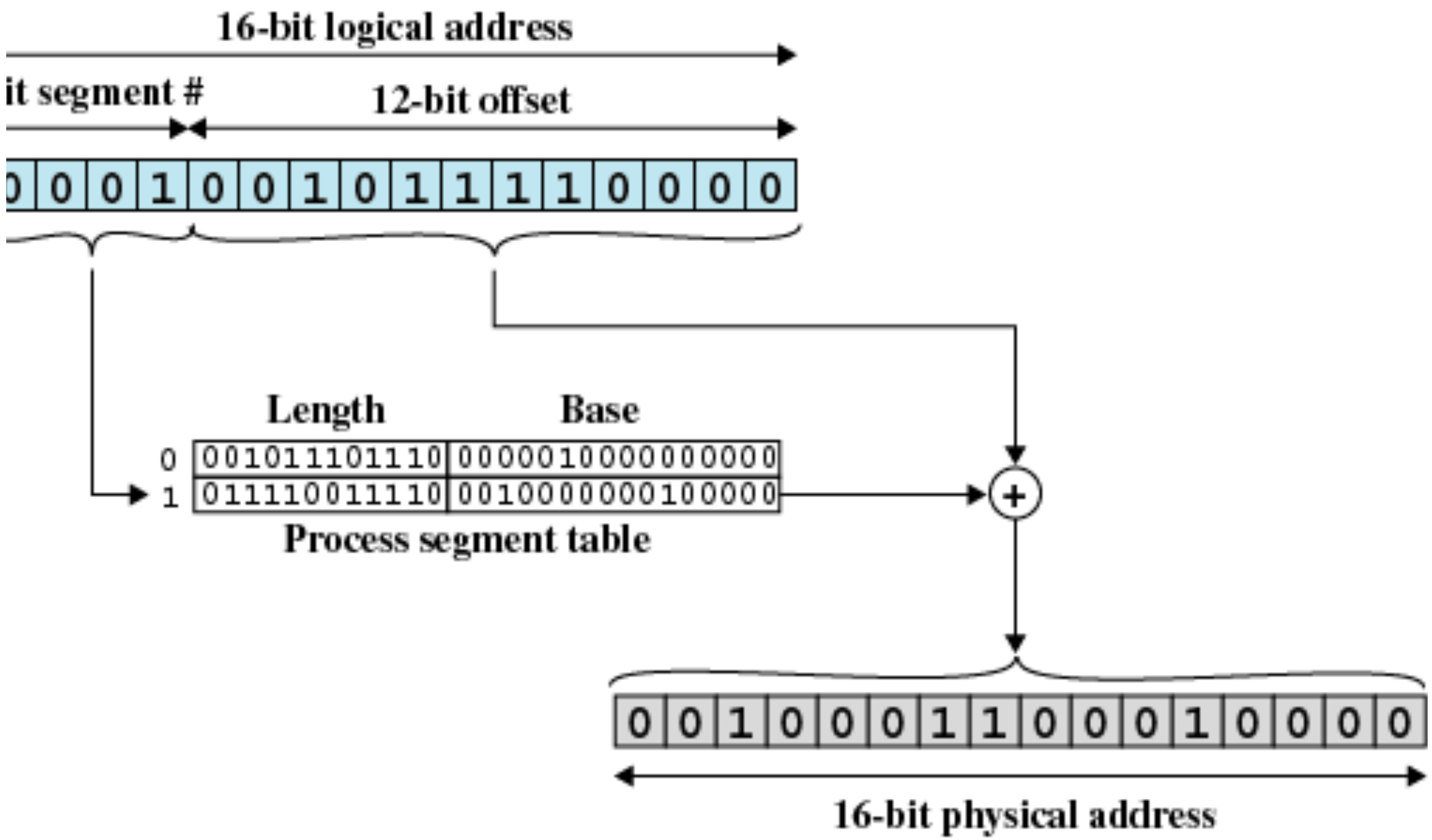
Addressing

- Logical address = PG_nr + offsett
- Physical address = FR_nr + offsett



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
- Segments not equal (dynamic part.)
- Need to keep track of segment length in segment table



(b) Segmentation

Figure 7.12 Examples of Logical-to-Physical Address Translati