Memory Management

Chapter 7

Memory Management

- ✓ Subdividing memory to accommodate multiple processes
- ✓ Memory needs to be allocated efficiently to pack as many processes into memory as needed/possible (and still ensure adequate performance)

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
  - 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 addresses in programs since addresses can be generated during execution
  - hence: addresses must be checked during execution, by hardware

Memory Management Requirements

- Sharing
  - allow several processes to access the same portion of memory
  - allow each process to access the same copy of the programs (e.g., Unix shell in a multi-user system) rather than creating a separate copy each time

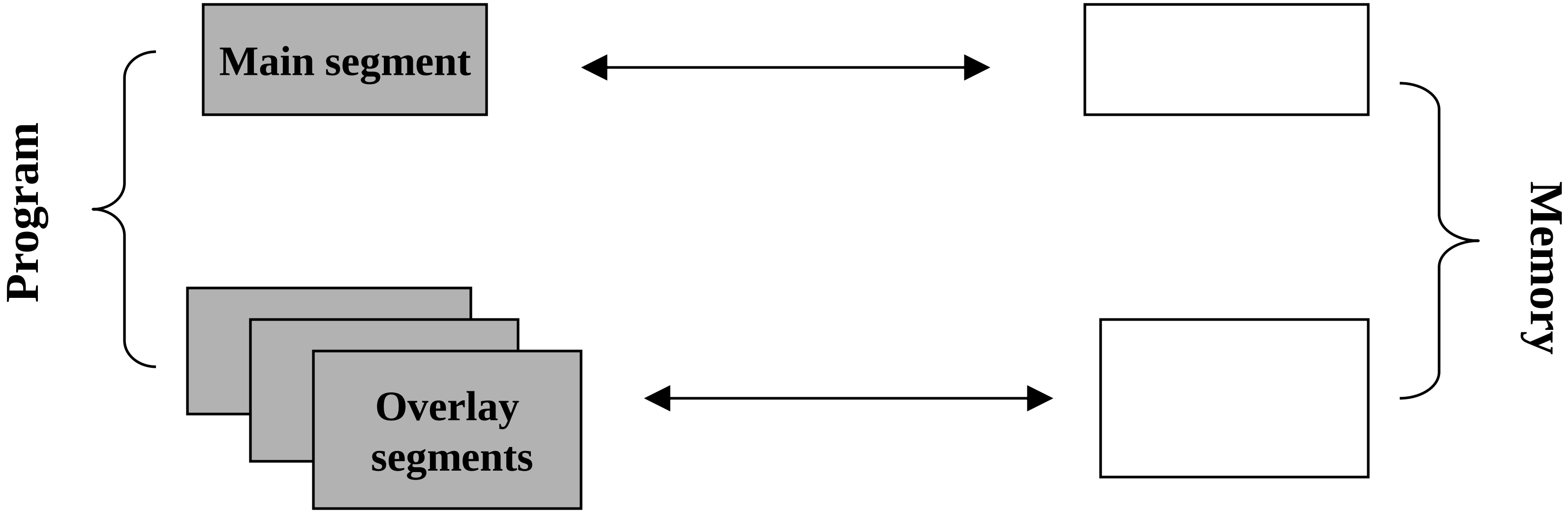
Memory Management Requirements

- Logical Organization
  - programs are written in modules
  - different degrees of protection given to modules (read-only, execute-only)
  - modules can be shared

Memory Management Requirements

- Physical Organization
  - memory available for a program plus its data might be insufficient
    - *overlaying* allows various modules to be assigned the same region of memory
  - secondary memory cheaper, larger capacity, and permanent, hence temporarily keep parts of the program data in secondary memory

Overlaying



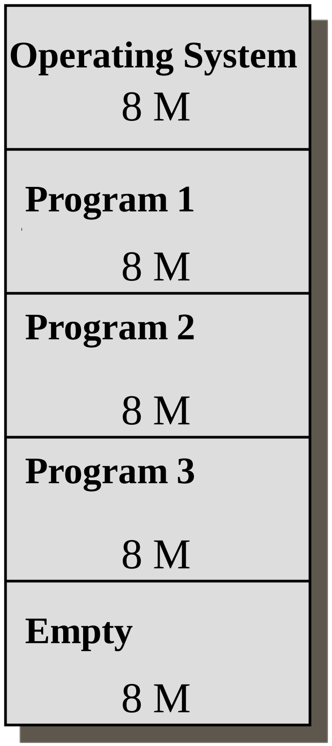
- Overlay segments are loaded over each other
- Programmer is responsible for splitting application into segments and for loading them.

Fixed Partitioning

- Partition available memory into regions with **fixed** boundaries
- Method 1: *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

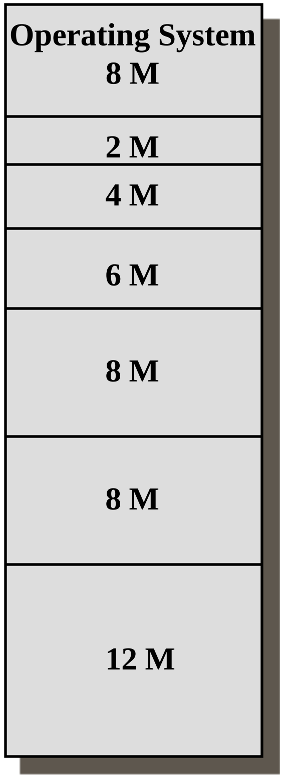
- Main memory use is inefficient. Any program, no matter how small, occupies an entire partition. This is called *internal fragmentation*.



Fixed Partitioning

✓ Method 2: ***Unequal-size partitions***

- lessens the problem with equal-size partitions
- ***External fragmentation:*** some partitions might be too small for some jobs, even though the **sum** of the partition sizes might be large enough



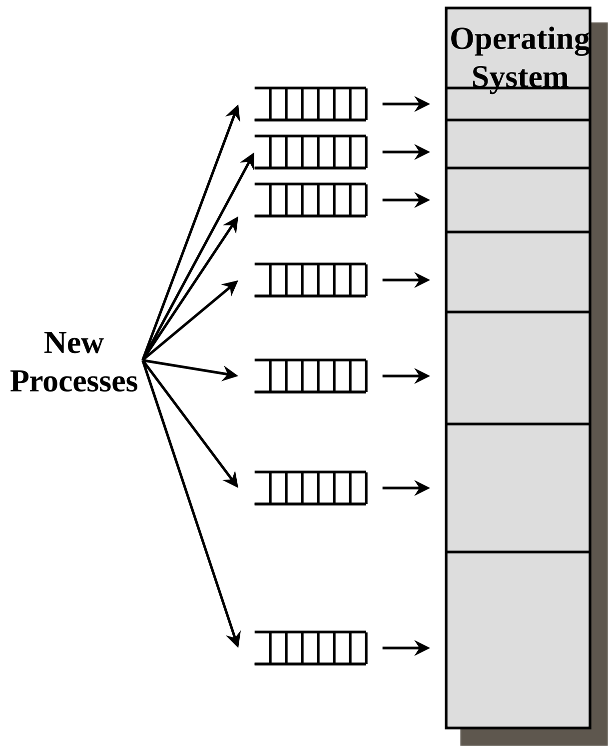
Fixed Partitions Problems

- ✓ External fragmentation
- ✓ Internal fragmentation
- ✓ Processes may grow/shrink

Placement Algorithm with Partitions

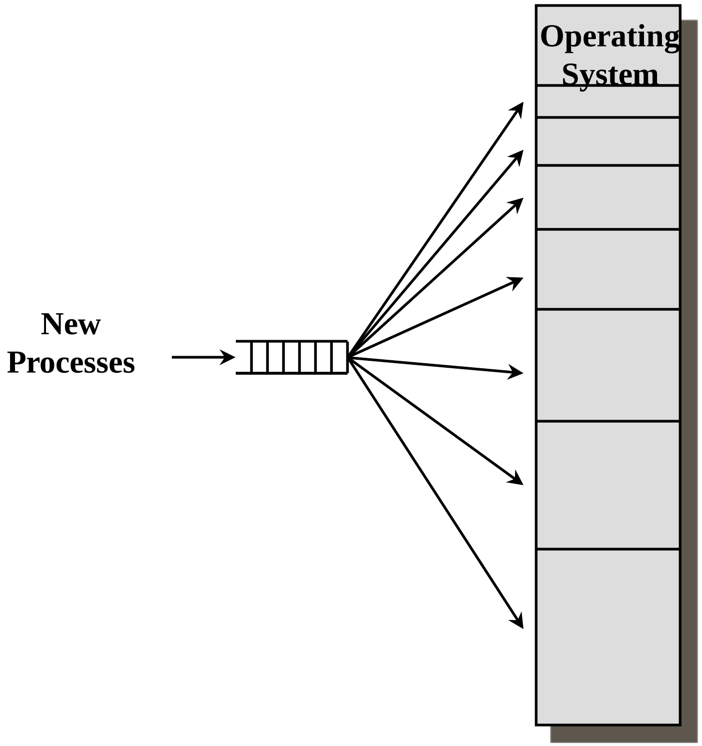
- Equal-size partitions
  - because all partitions are of equal size, it does not matter which partition is used
- Unequal-size partitions
  - can assign each process to the smallest partition within which it will fit
  - queue for each partition
  - processes are assigned in such a way as to minimize wasted memory within a partition

One Queue of Processes per Partition



One Global Process Queue

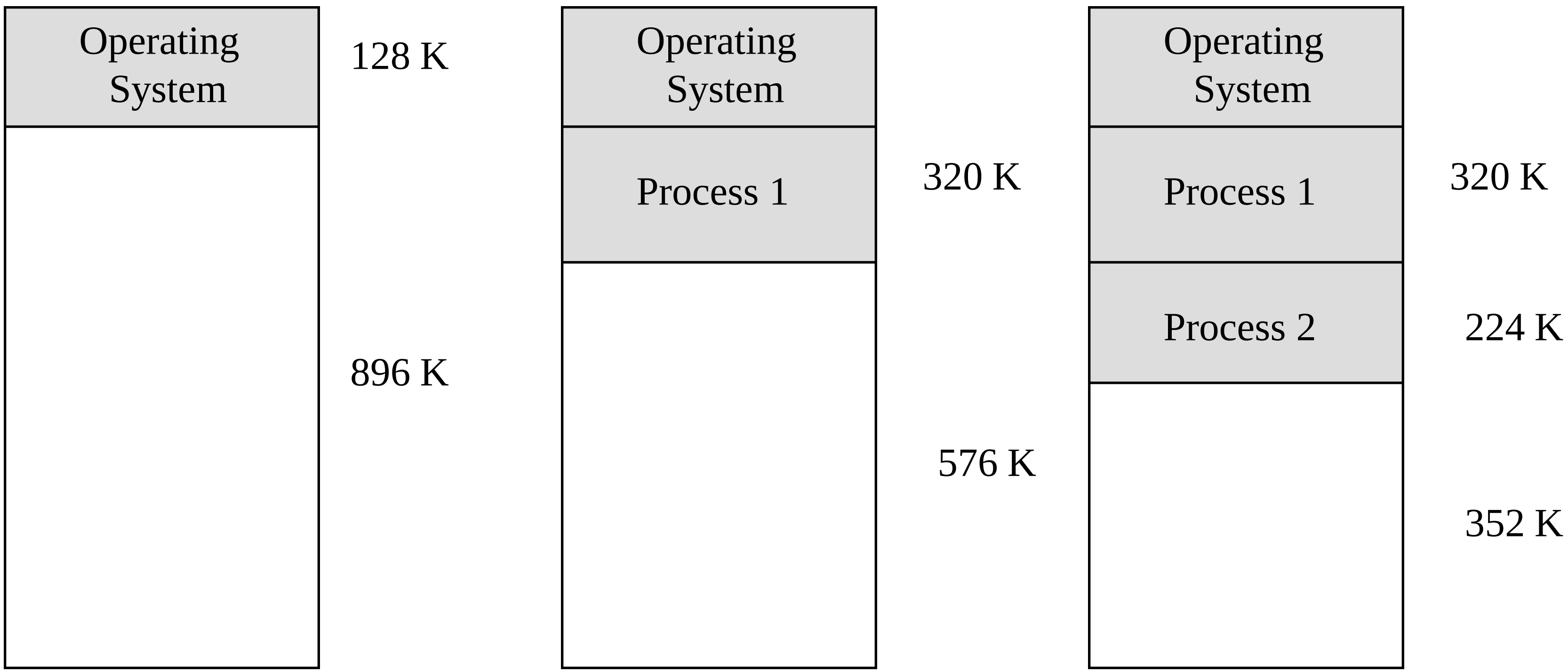
- ✓ When its time to load a process into main memory, the smallest available partition that will hold the process is selected



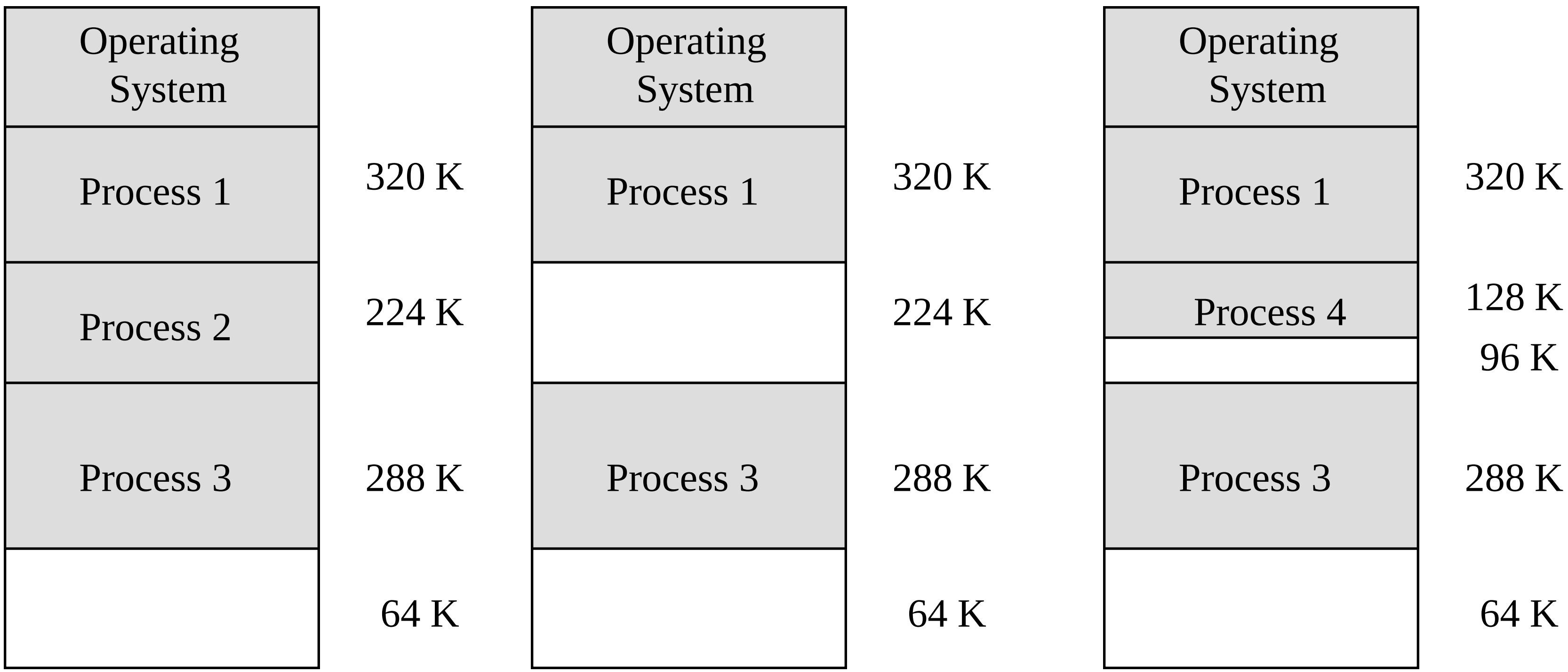
Dynamic Partitioning

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

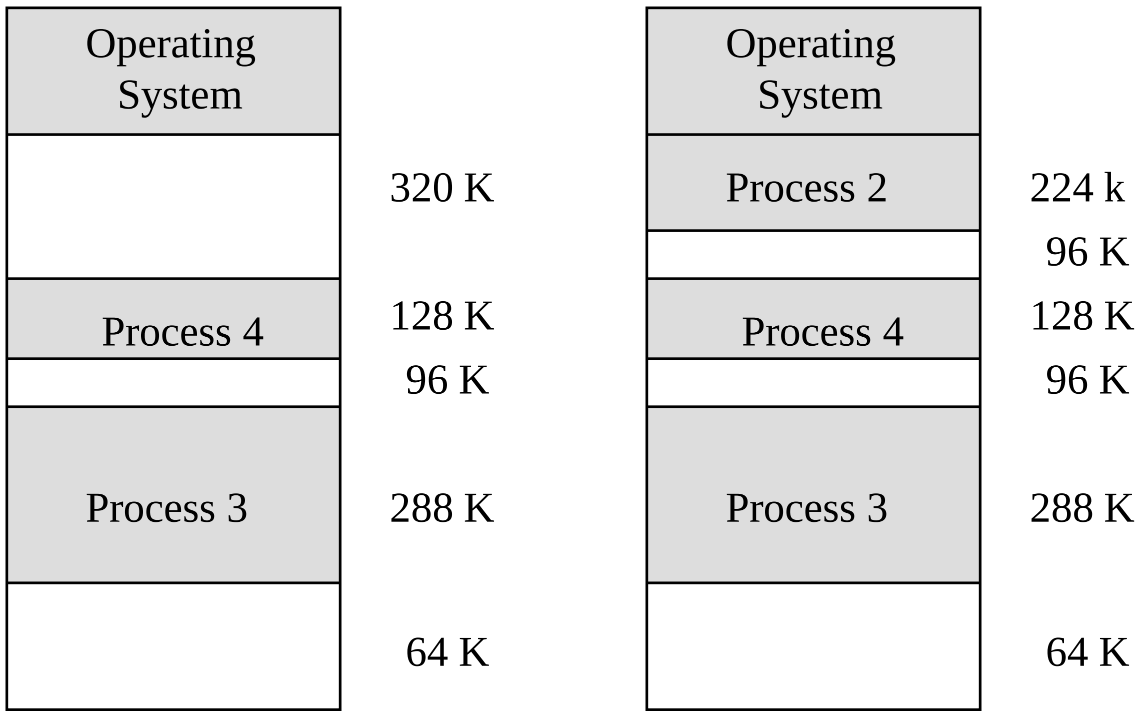
Example of Dynamic Partitioning



Example of Dynamic Partitioning



Example 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 (must scan the entire list of free blocks)
  - tends to leave small chunks of free space around; hence memory compaction must be done more often

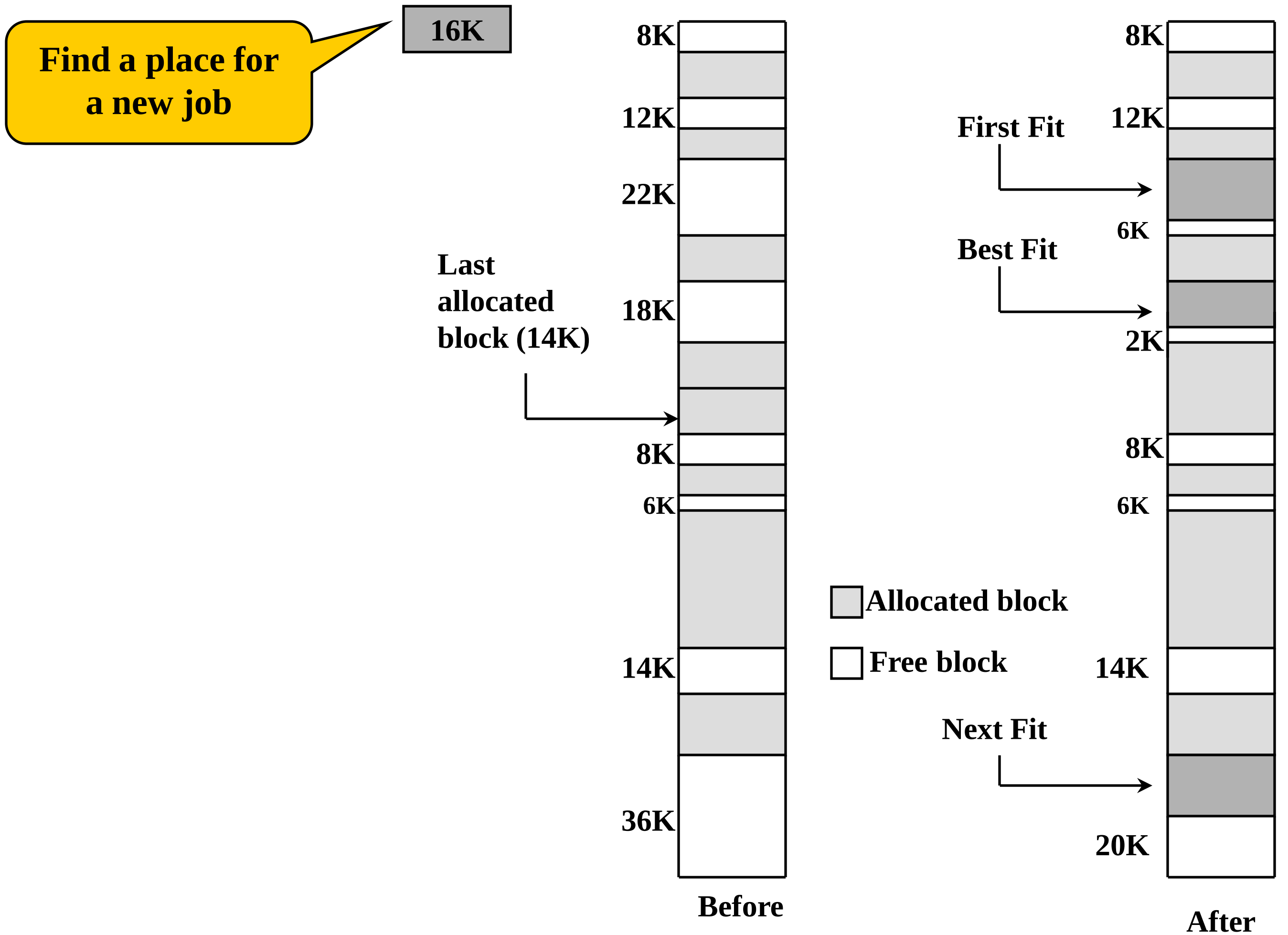
Dynamic Partitioning Placement Algorithm

- ***First-fit*** algorithm
  - starts scanning memory from the beginning and chooses the first available block that is large enough.
  - fastest
  - may have many 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***
  - starts scanning memory from the location of the last placement and chooses the next available block that is large enough
  - 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

Dynamic Partitioning Placement Algorithm



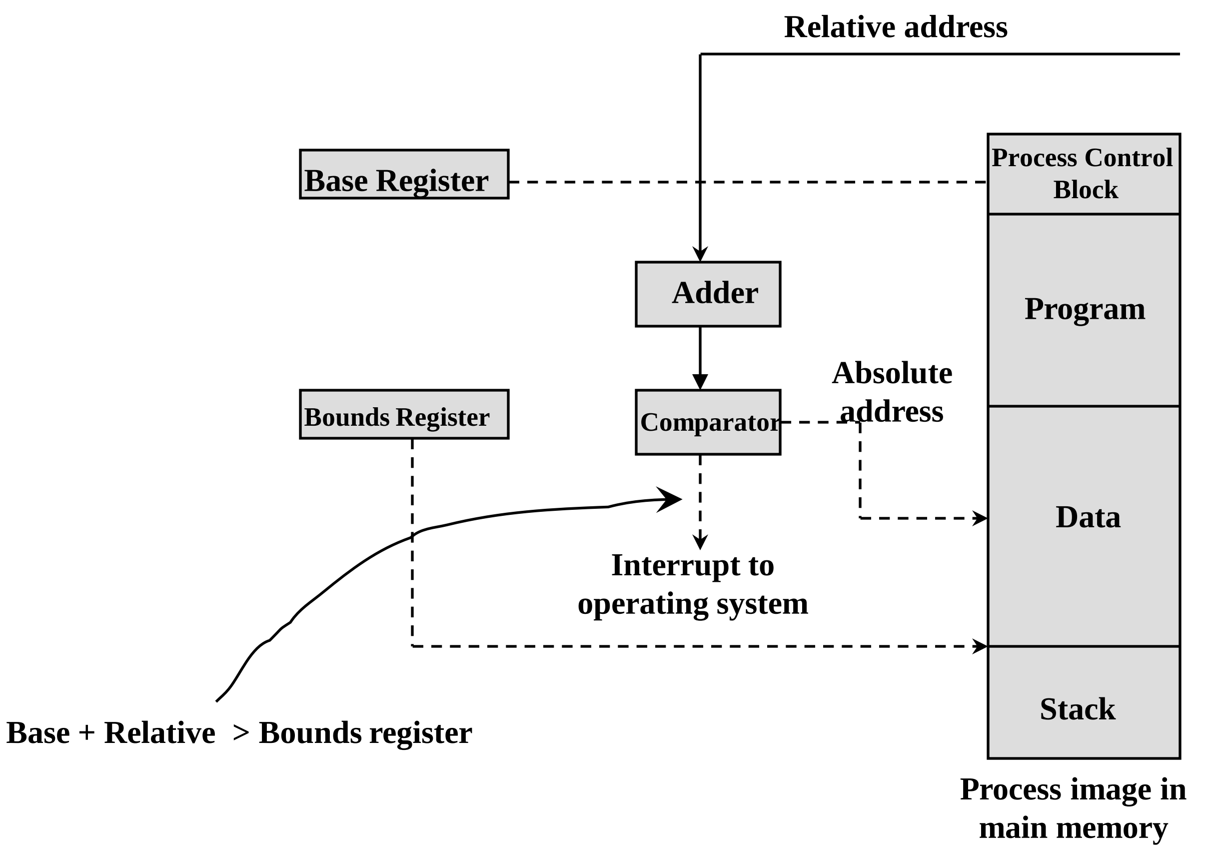
Relocation

- 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 (due to swapping)
- Compaction might also cause a program to occupy a different absolute memory location

Addresses

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

Hardware Support for Program 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

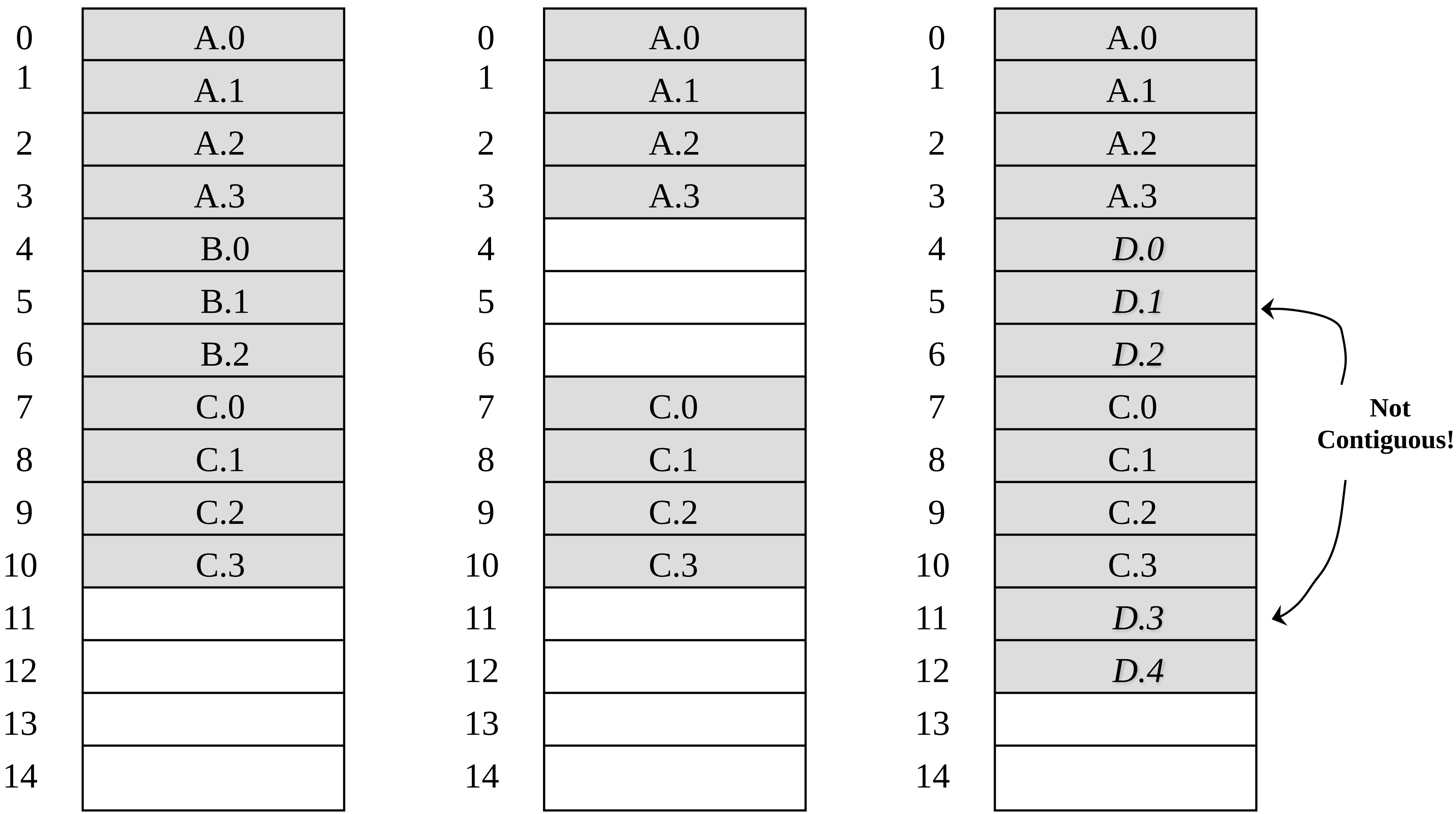
Paging

- Partition memory into small equal-size chunks and divide each process into the same size chunks
- The chunks of a process are called *pages* and chunks of memory are called *frames*
- Operating system maintains a *page table* for each process
  - page table contains the frame location for each page in the process
  - memory address within the program consist of a page number and offset within the page

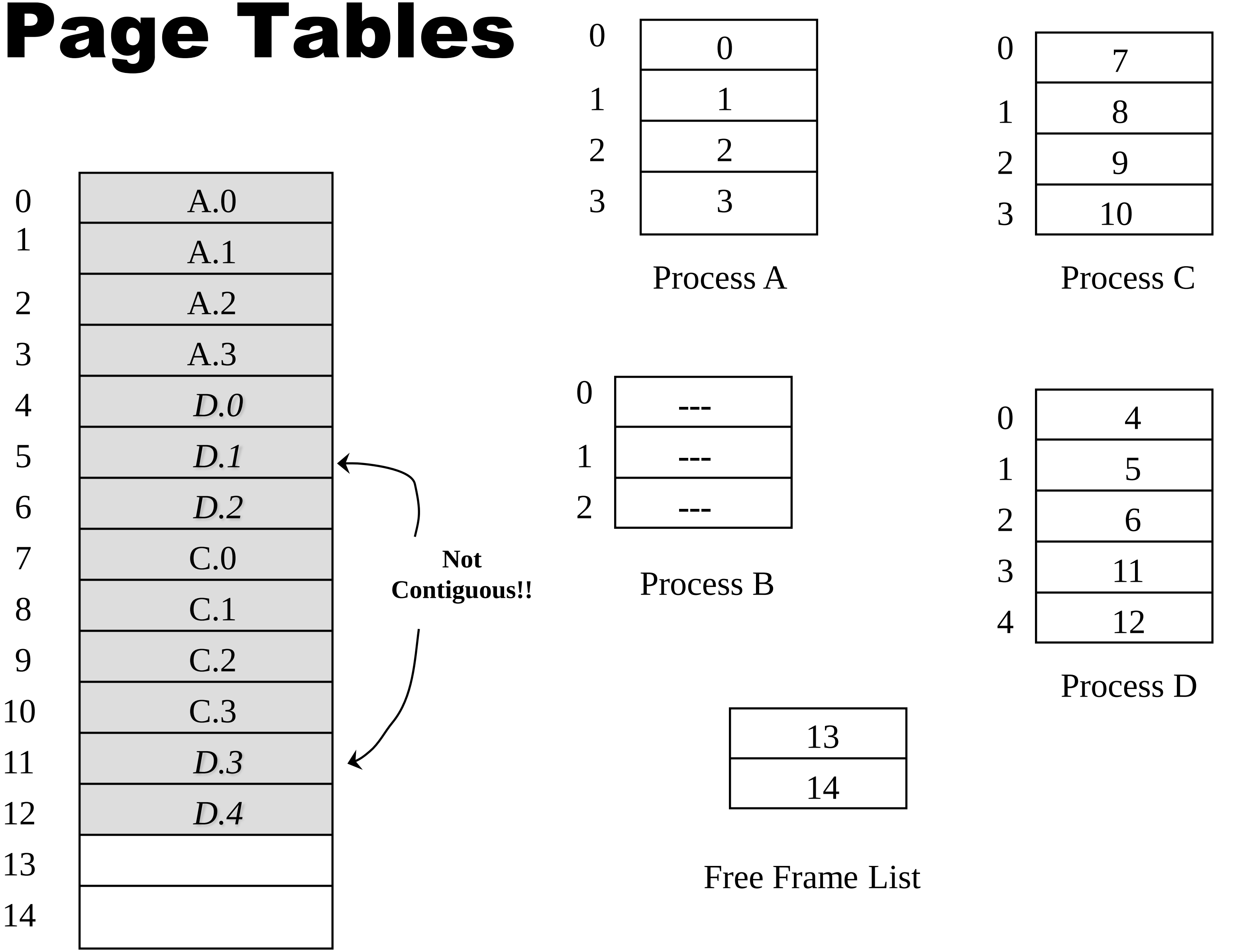
Paging

Frame Number		Frame Number		Frame Number	
0		0	A.0	0	A.0
1		1	A.1	1	A.1
2		2	A.2	2	A.2
3		3	A.3	3	A.3
4		4		4	B.0
5		5		5	B.1
6		6		6	B.2
7		7		7	
8		8		8	
9		9		9	
10		10		10	
11		11		11	
12		12		12	
13		13		13	
14		14		14	

Paging


0 A.0
1 A.1
2 A.2
3 A.3
4 B.0
5 B.1
6 B.2
7 C.0
8 C.1
9 C.2
10 C.3
11
12
13
14
0 A.0
1 A.1
2 A.2
3 A.3
4
5
6
7 C.0
8 C.1
9 C.2
10 C.3
11
12
13
14
0 A.0
1 A.1
2 A.2
3 A.3
4 D.0
5 D.1
6 D.2
7 C.0
8 C.1
9 C.2
10 C.3
11 D.3
12 D.4
13
14
Not Contiguous!!

Page Tables



0 A.0
1 A.1
2 A.2
3 A.3
4 D.0
5 D.1
6 D.2
7 C.0
8 C.1
9 C.2
10 C.3
11 D.3
12 D.4
13
14
Not Contiguous!!
0 0
1 1
2 2
3 3
Process A
0 ---
1 ---
2 ---
Process B
0 7
1 8
2 9
3 10
Process C
0 4
1 5
2 6
3 11
4 12
Process D
13
14
Free Frame List

Segmentation

- ✓ Segments of the 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

Segmentation

- Segments may or may not be contiguous
  - A non-contiguous segment can be organized using paging (each segment will then have a page table)
- Segment table: gives starting address and length to each segment

Segment Table