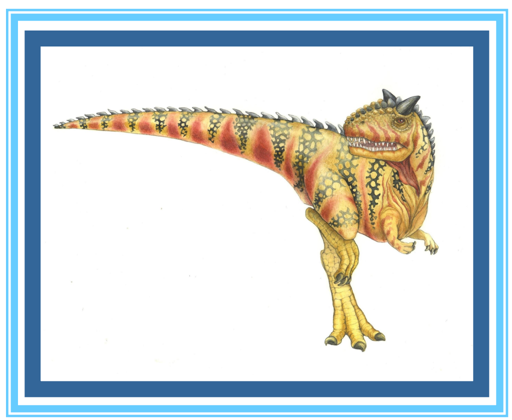
Chapter 8: Main Memory



Recap

- Background
- Address Binding
  - Base and Limit registers
  - Hardware Address Protection
- Logical and Physical addresses
- Dynamic Relocation (MMU)
- Dynamic Linking

Objectives

- Swapping
- Contiguous Memory Allocation
  - Fixed size partitioning
  - Variable size partitioning
- Dynamic Storage-Allocation Problem
  - First Fit
  - Best Fit
  - Worst Fit
- Fragmentation
  - Internal Fragmentation
  - External Fragmentation
- Compaction

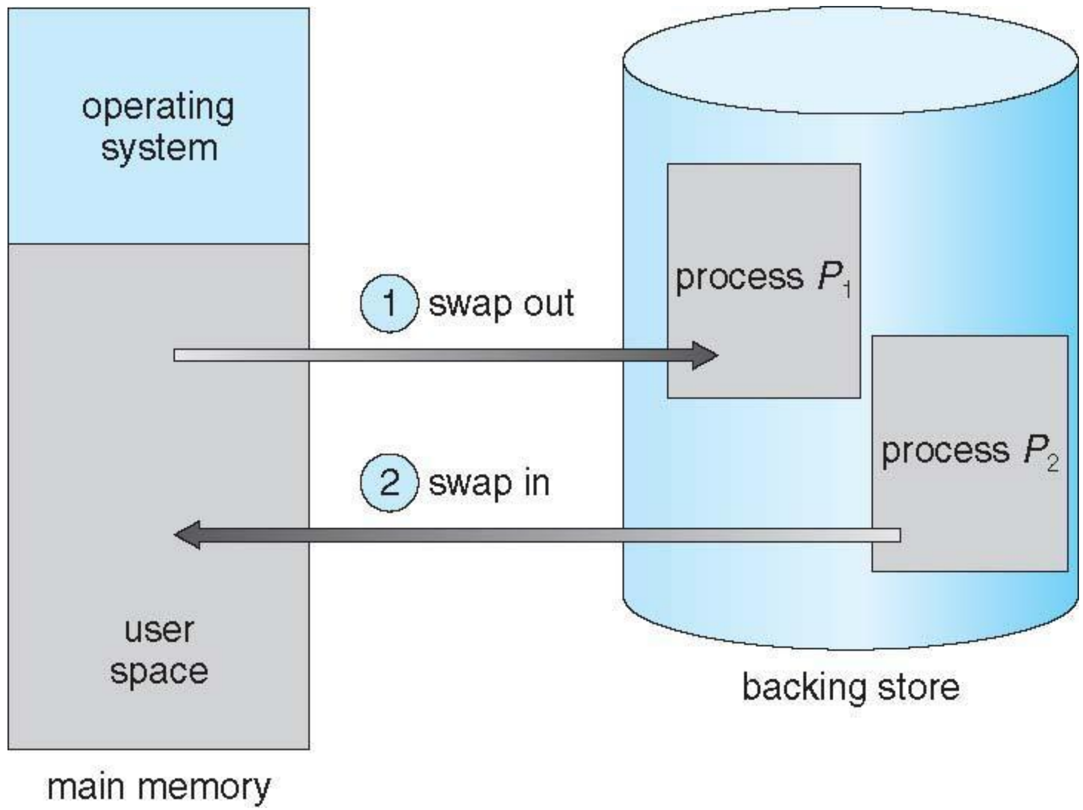
Swapping

- A process can be **swapped** temporarily out of memory to a backing store, and then brought back into memory for continued execution
  - Total physical memory space of processes can exceed physical memory
- **Backing store** – fast disk large enough to accommodate copies of all memory images for all users; must provide direct access to these memory images
- **Roll out, roll in** – swapping variant used for priority-based scheduling algorithms; lower-priority process is swapped out so higher-priority process can be loaded and executed
- Major part of swap time is transfer time; total transfer time is directly proportional to the amount of memory swapped
- System maintains a **ready queue** of ready-to-run processes which have memory images on disk

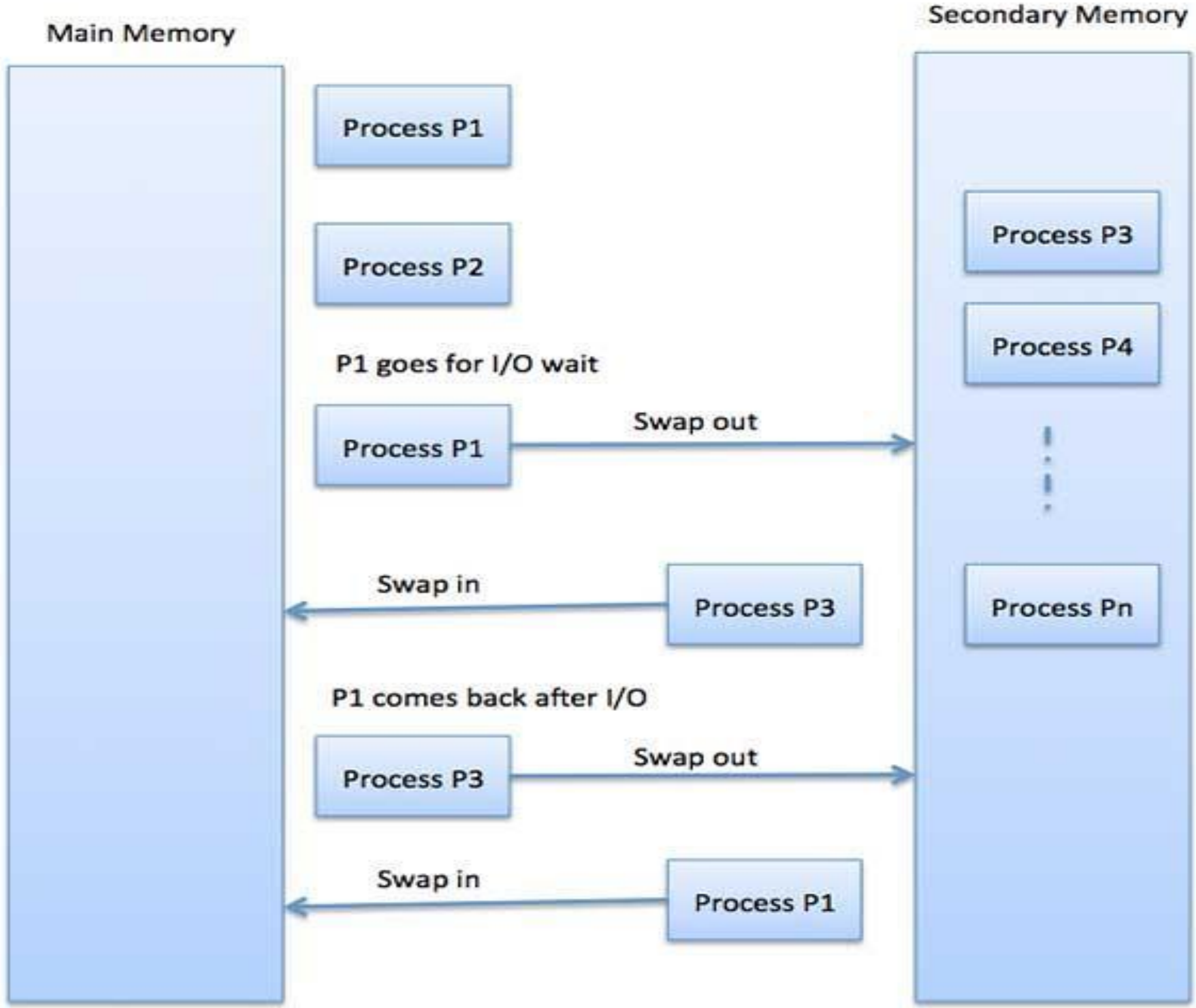
Swapping (Cont.)

- Does the swapped-out process need to swap back into same physical addresses?
- Depends on address binding method
  - Plus consider pending I/O to / from process memory space
- Modified versions of swapping are found on many systems (i.e., UNIX, Linux, and Windows)
  - Swapping normally disabled
  - Started if more than threshold amount of memory allocated
  - Disabled again once memory demand reduced below threshold

Schematic View of Swapping



Schematic View of Swapping



Context Switch Time including Swapping

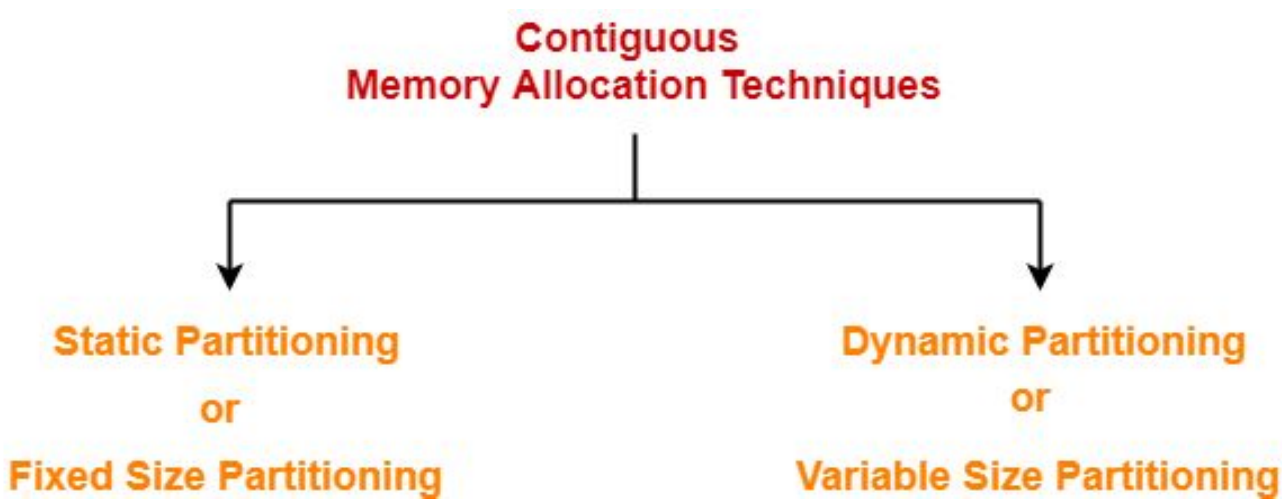
- If next processes to be put on CPU is not in memory, need to swap out a process and swap in target process
- Context switch time can then be very high
- 100MB process swapping to hard disk with transfer rate of 50MB/sec
  - Swap out time of 2000 ms
  - Plus swap in of same sized process
  - Total context switch swapping component time of 4000ms (4 seconds)
- Can reduce if reduce size of memory swapped – by knowing how much memory really being used
  - System calls to inform OS of memory use via `request_memory()` and `release_memory()`

Context Switch Time and Swapping (Cont.)

- Other constraints as well on swapping
  - Pending I/O – can’t swap out as I/O would occur to wrong process
  - Or always transfer I/O to kernel space, then to I/O device
    - Known as **double buffering**, adds overhead
- Standard swapping not used in modern operating systems
  - But modified version common
    - Swap only when free memory extremely low

Contiguous Allocation

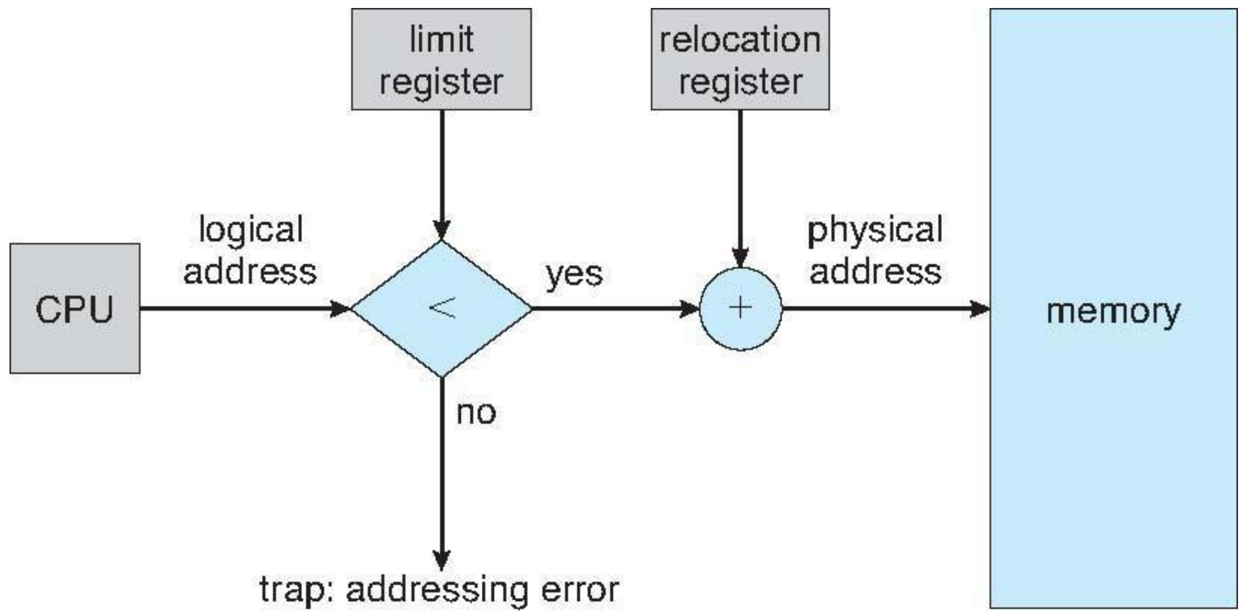
- Main memory must support both OS and user processes
- Limited resource, must allocate efficiently
- Contiguous allocation is one early method
- Main memory usually divided into two **partitions**:
  - Resident operating system, usually held in low memory with interrupt vector
  - User processes then held in high memory
  - Each process contained in single contiguous section of memory



Contiguous Allocation (Cont.)

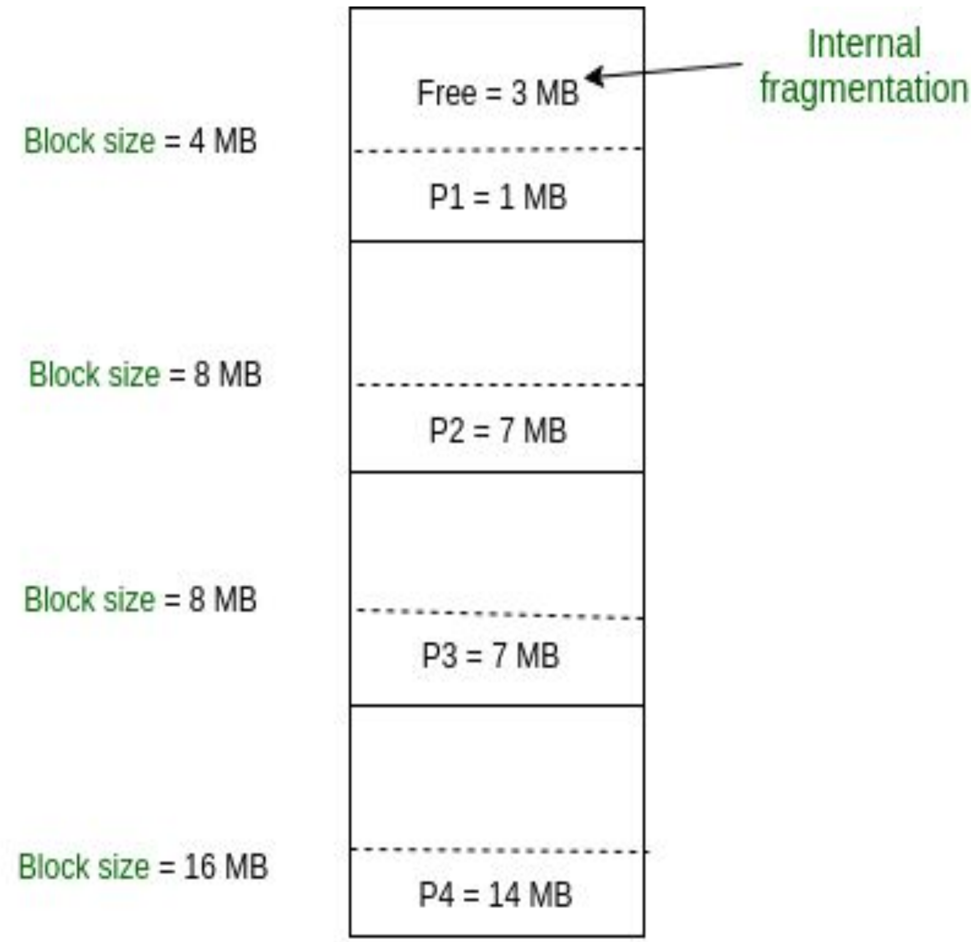
- Relocation registers used to protect user processes from each other, and from changing operating-system code and data
  - Base register contains value of smallest physical address
  - Limit register contains range of logical addresses – each logical address must be less than the limit register
  - MMU maps logical address *dynamically*
  - Can then allow actions such as kernel code being **transient** and kernel changing size

Hardware Support for Relocation and Limit Registers



Fixed Size Partitioning

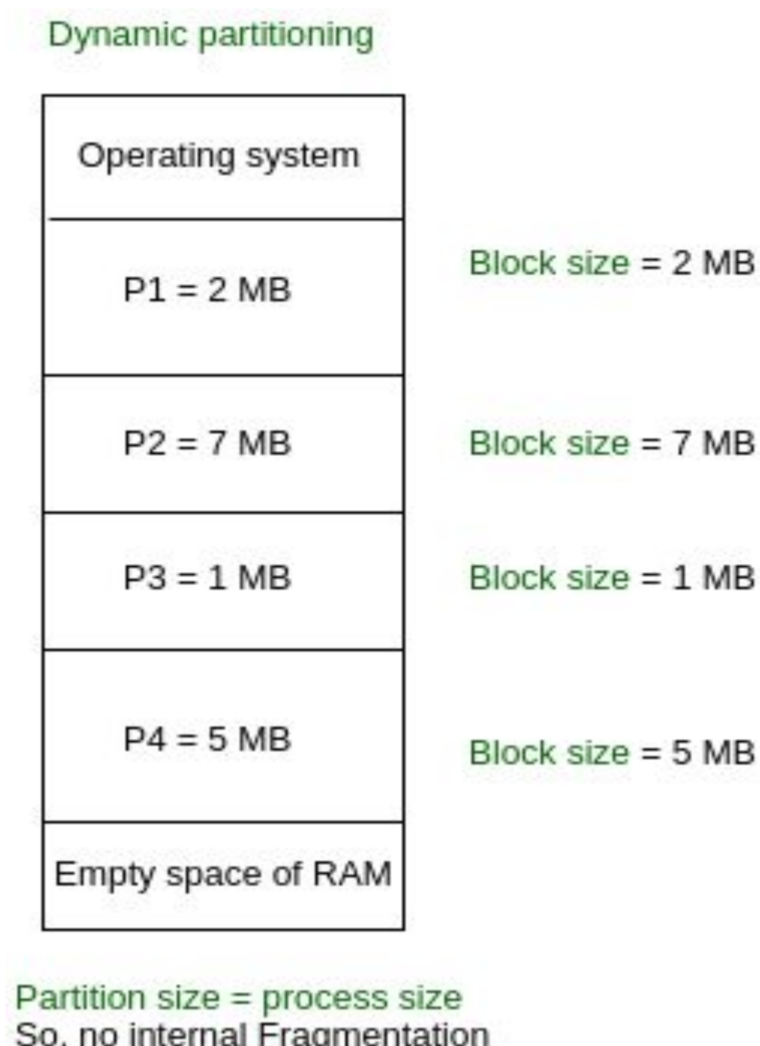
- This is the oldest and simplest technique used to put more than one processes in the main memory. In this partitioning, number of partitions (non-overlapping) in RAM are **fixed but size** of each partition may or **may not be same**.
- As it is **contiguous** allocation, hence no spanning is allowed. Here partition are made before execution or during system configure.+/



Fixed size partition

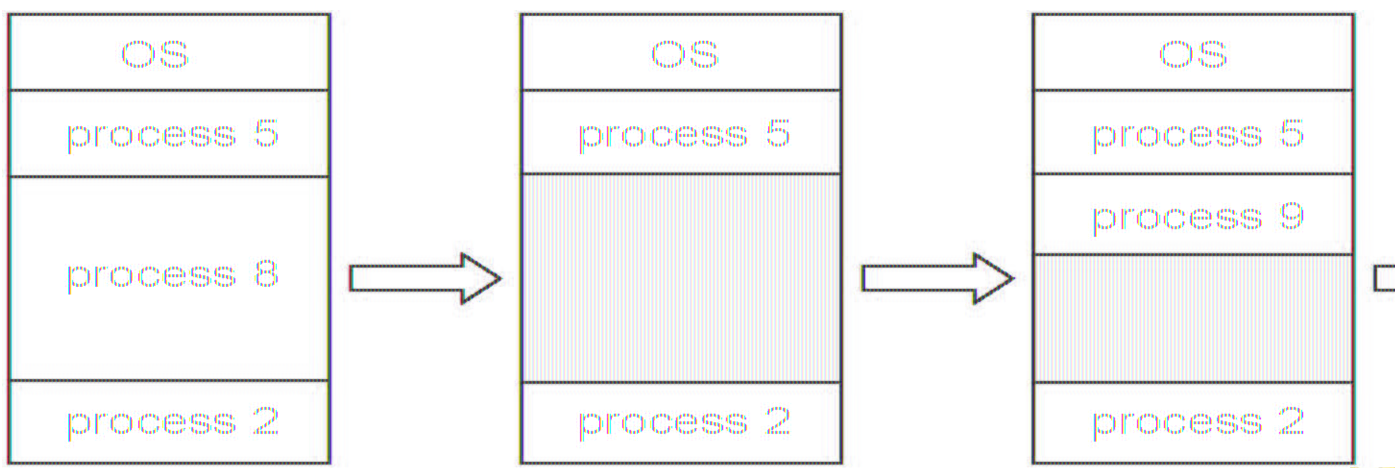
Variable Size Partitioning

- In contrast with fixed partitioning, partitions are not made before the execution or during system configure. Various **features** associated with variable Partitioning-
- Initially RAM is empty, and partitions are made during the run-time according to process's need instead of partitioning during system configure.
- The size of partition will be equal to incoming process.
- The partition size varies according to the need of the process so that the internal fragmentation can be avoided to ensure efficient utilization of RAM.
- Number of partitions in RAM is not fixed and depends on the number of incoming process and Main Memory's size.



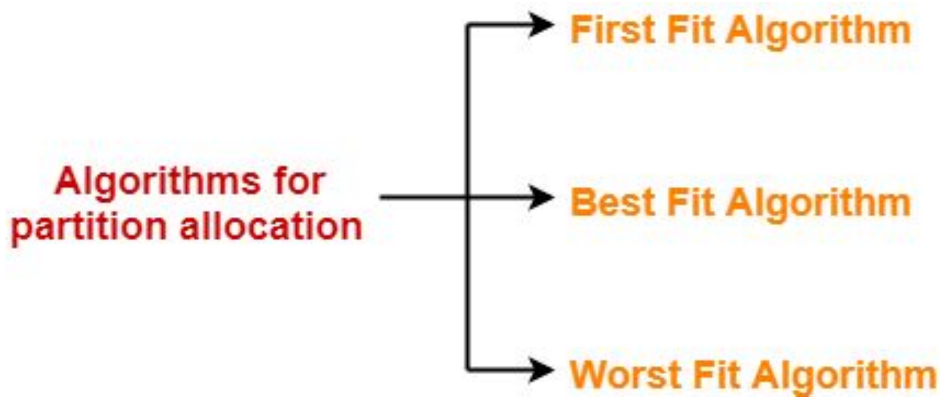
Multiple-partition allocation

- Multiple-partition allocation
  - Degree of multiprogramming limited by number of partitions
  - **Variable-partition** sizes for efficiency (sized to a given process' needs)
  - **Hole** – block of available memory; holes of various size are scattered throughout memory
  - When a process arrives, it is allocated memory from a hole large enough to accommodate it
  - Process exiting frees its partition, adjacent free partitions combined
  - Operating system maintains information about:
    a) allocated partitions   b) free partitions (hole)



Dynamic Storage-Allocation Problem

How to satisfy a request of size ***n*** from a list of free holes?



- **First-fit**: Allocate the ***first*** hole that is big enough
- **Best-fit**: Allocate the ***smallest*** hole that is big enough; must search entire list, unless ordered by size
  - Produces the smallest leftover hole
- **Worst-fit**: Allocate the ***largest*** hole; must also search entire list
  - Produces the largest leftover hole

First-fit and best-fit better than worst-fit in terms of speed and storage utilization

First Fit

- First Fit : Allocate the first free block that is large enough for the new process.
- This is a fast algorithm.

First Fit Example

Initial memory mapping

OS
P1 12 KB
<FREE> 10 KB
P2 20 KB
<FREE> 16 KB
P3 6 KB
<FREE> 4 KB

First Fit Example (Cont.)

P4 of 3KB arrives

OS
P1 12 KB
<FREE> 10 KB
P2 20 KB
<FREE> 16 KB
P3 6 KB
<FREE> 4 KB

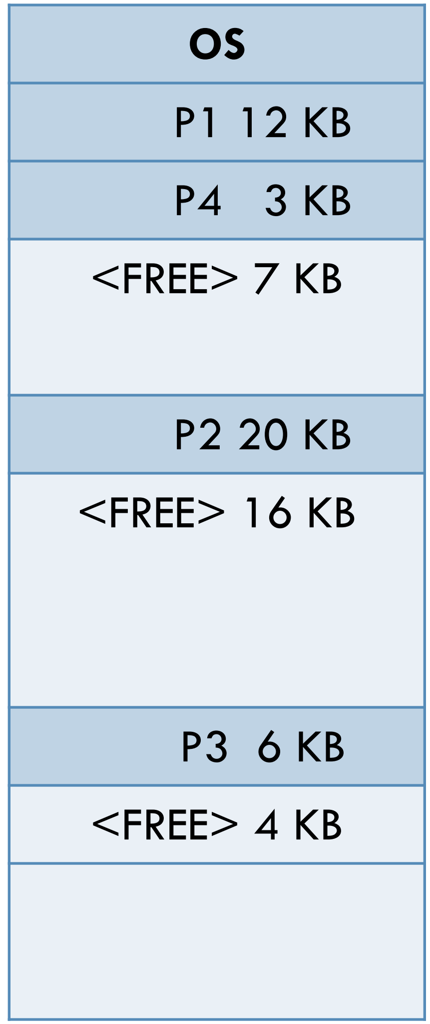
First Fit Example (Cont.)

P4 of 3KB loaded here by FIRST FIT

OS
P1 12 KB
P4 3 KB
<FREE> 7 KB
P2 20 KB
<FREE> 16 KB
P3 6 KB
<FREE> 4 KB

First Fit Example (Cont.)

P5 of 15KB arrives



First Fit Example (Cont.)

P5 of 15 KB
loaded here by
FIRST FIT

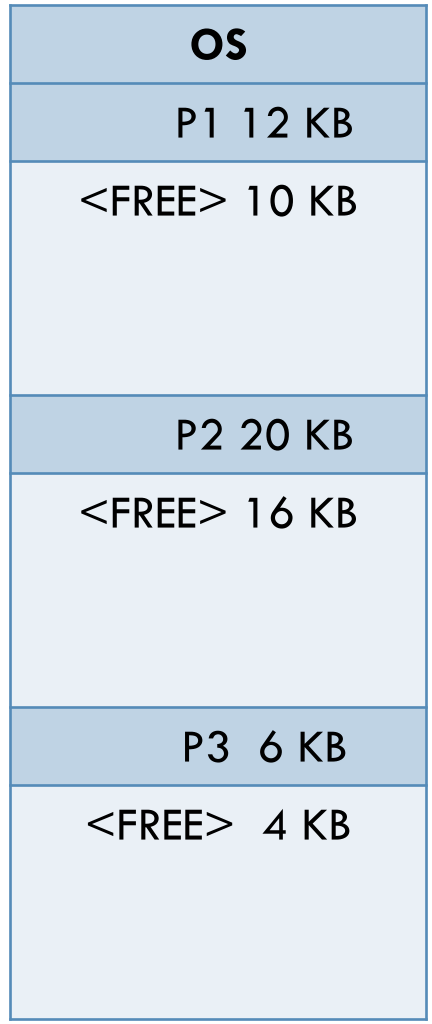
OS
P1 12 KB
P4 3 KB
<FREE> 7 KB
P2 20 KB
P5 15 KB
<FREE> 1 KB
P3 6 KB
<FREE> 4 KB

Best fit

- Best Fit : Allocate the smallest block among those that are large enough for the new process.
- In this method, the OS has to search the entire list, or it can keep it sorted and stop when it hits an entry which has a size larger than the size of new process.
- This algorithm produces the smallest left over block.
- However, it requires more time for searching all the list or sorting it
- If sorting is used, merging the area released when a process terminates to neighboring free blocks, becomes complicated.

Best Fit Example

Initial memory mapping



Best Fit Example (Cont.)

P4 of 3KB arrives

OS
P1 12 KB
<FREE> 10 KB
P2 20 KB
<FREE> 16 KB
P3 6 KB
<FREE> 4 KB

Best Fit Example (Cont.)

P4 of 3KB
loaded here by
BEST FIT

OS
P1 12 KB
<FREE> 10 KB
P2 20 KB
<FREE> 16 KB
P3 6 KB
P4 3 KB
<FREE> 1 KB

Best Fit Example (Cont.)

P5 of 15KB arrives

OS
P1 12 KB
<FREE> 10 KB
P2 20 KB
<FREE> 16 KB
P3 6 KB
P4 3 KB
<FREE> 1 KB

Best Fit Example (Cont.)

P5 of 15 KB loaded here by BEST FIT

OS
P1 12 KB
<FREE> 10 KB
P2 20 KB
P5 15 KB
<FREE> 1 KB
P3 6 KB
P4 3 KB
<FREE> 1 KB

Worst Fit

- Worst Fit : Allocate the largest block among those that are large enough for the new process.
- Again a search of the entire list or sorting it is needed.
- This algorithm produces the largest over block.

Worst Fit Example

Initial memory mapping

OS
P1 12 KB
<FREE> 10 KB
P2 20 KB
<FREE> 16 KB
P3 6 KB
<FREE> 4 KB

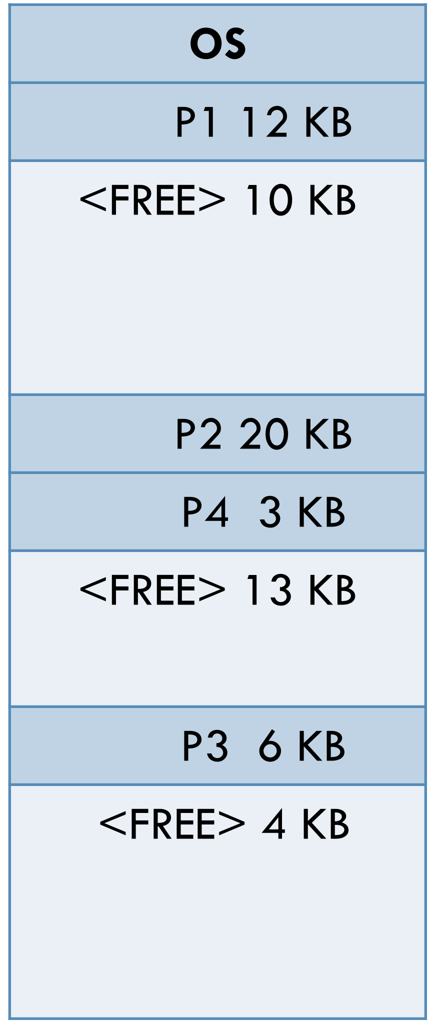
Worst Fit Example (Cont.)

P4 of 3KB
Loaded here by
WORST FIT

OS
P1 12 KB
<FREE> 10 KB
P2 20 KB
P4 3 KB
<FREE> 13 KB
P3 6 KB
<FREE> 4 KB

Worst Fit Example (Cont.)

No place to load
P5 of 15K



Worst Fit Example (Cont.)

No place to load P5 of 15K

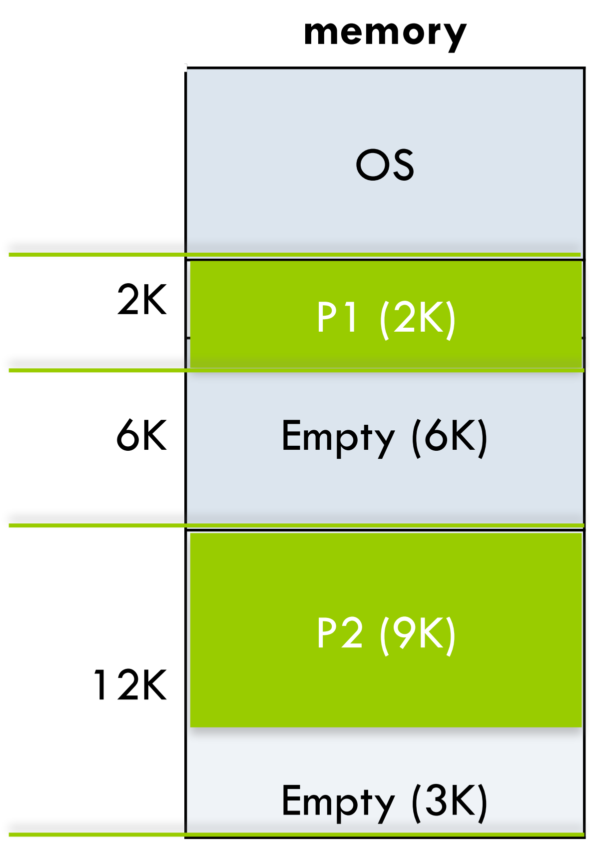
Compaction is needed !!

OS
P1 12 KB
<FREE> 10 KB
P2 20 KB
P4 3 KB
<FREE> 13 KB
P3 6 KB
<FREE> 4 KB

Fragmentation

- **External Fragmentation** – total memory space exists to satisfy a request, but it is not contiguous
- **Internal Fragmentation** – allocated memory may be slightly larger than requested memory; this size difference is memory internal to a partition, but not being used

Fragmentation



If a whole partition is currently not being used, then it is called *an* ***external fragmentation.***

If a partition is being used by a process requiring some memory smaller than the partition size, then it is called an ***internal fragmentation.***

Fragmentation (Cont.)

- Reduce external fragmentation by **compaction**
  - Shuffle memory contents to place all free memory together in one large block
  - Compaction is possible *only* if relocation is dynamic, and is done at execution time
  - I/O problem
    - Latch job in memory while it is involved in I/O
    - Do I/O only into OS buffers
- Now consider that backing store has same fragmentation problems