Operating System
Day 3

PPT's Compiled by:
Mrs.Akshita.S.Chanchlani
SunBeam Infotech
akshita.chanchlani@sunbeaminfo.com

Inter process Communication (IPC)

- Passing information between processes
- Used to coordinate process activity
- Processes within a system may be **independent** or **cooperating**

Independent process

- **do not get affected by the execution of another process**

Cooperating process

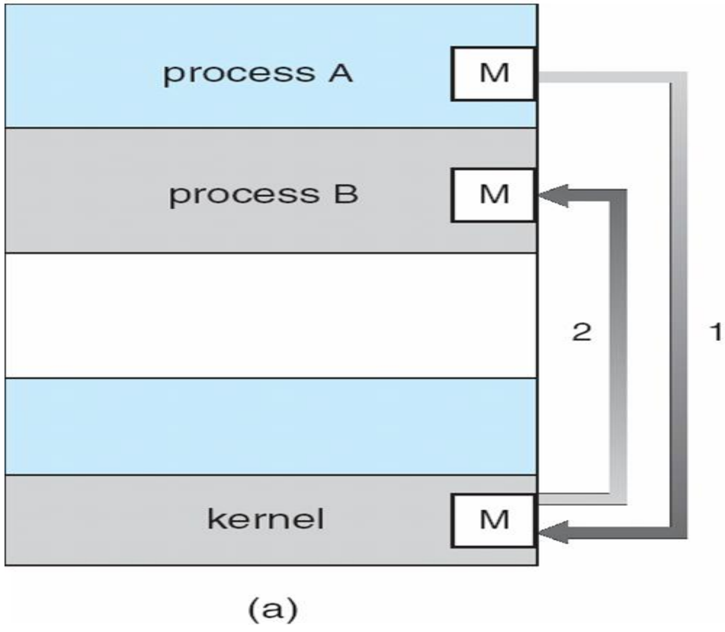
- **get affected by the execution of another process**

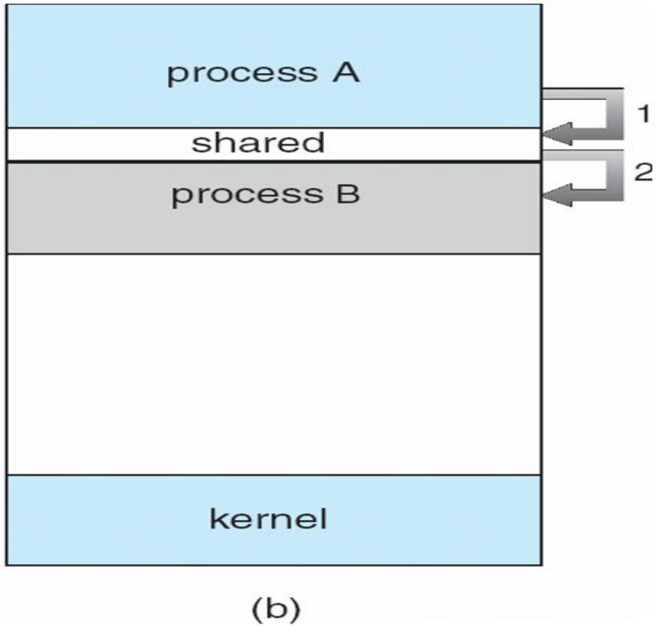
- Reasons for cooperating processes:
  - Information sharing
  - Computation speedup
  - Modularity
  - Convenience
- Cooperating processes need **interprocess communication** (**IPC**)

IPC Requirements

- If *P* and *Q* wish to communicate, they need to:
  - establish communication link between them.
  - exchange messages via send/receive.
- Implementation of communication link:
  - physical (e.g., shared memory, hardware bus)
  - logical (e.g., logical properties)

Two models/Mechanisms of IPC:





(a)Message Passing (b) Shared Memory Model

Message Passing

- communication takes place by means of messages exchanged between the cooperating processes
- Uses two primitives : Send and Receive

Shared Memory

- In the shared-memory model, a region of memory that is shared by cooperating processes is established.
- Processes can then exchange information by reading and writing data to the shared region.

Message Passing Mechanisms

Pipe

- Allowing two processes to communicate in unidirectional way
- by using pipe one process can send message to another process at a time
  - unnamed pipe --- two related processes can communicate with other.
  - **"related process"** -- processes which are of same parent
  - named pipe -- non-related processes can also communicate with each other.

Message Queue

- It is bidirectional communication
- processes communicate with each other through message primitives.
  - **send**(*destination, message*) or **send**(*message*)
  - **receive**(*source, message*) or **receive**(*message*)

Signal

- Processes communicates with each other by means of signals.
- Signals have predefined meaning
- Eg. SIGTERM (Terminate) SIGKILL, SIGSTOP, SIGCONT etc....

Socket

- A **socket** is defined as an *endpoint for communication*
- e.g.Host to Webserver
- socket = ip addr + port number
- e.g. chatting application

Process Synchronization

- Concurrent access to shared data may result in **data inconsistency.**
- Maintaining data consistency requires mechanisms to ensure the orderly execution of cooperating processes.
- **Race condition**: The situation where several processes access – and manipulate shared data concurrently. The final value of the shared data depends upon which process finishes last.

- To prevent race conditions, concurrent processes must be **synchronized**.

The Critical Section / Region Problem

- Occurs in systems where multiple processes all compete for the use of shared data.
- Each process includes a section of code (the **critical section**) where it accesses this shared data.
- The problem is to ensure that **only one process at a time is allowed** to be operating in its critical section.

Critical-Section Problem

- Consider system of n processes $\{p_0, p_1, \ldots p_{n-1}\}$
- Each process has **critical section** segment of code
  - Process may be changing common variables, updating table, writing file, etc
  - When one process in critical section, no other may be in its critical section
- Each process must ask permission to enter critical section in **entry section**, may follow critical section with **exit section**, then **remainder section**
- Problem – ensure that when one process is executing in its critical section, no other process is allowed to execute in its critical section.

```
repeat
  entry_to_section
          critical section
  exit_section
          remainder of program
until false;
```

Solution to Critical-Section Problem

Mutual Exclusion.

- If process P_i is executing in its critical section, then no other processes can be executing in their critical sections.

Progress

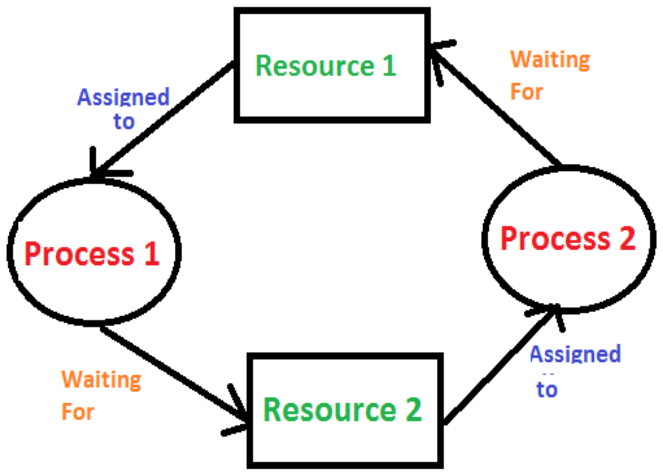
- If no process is executing in its critical section and there exist some processes that wish to enter their critical section, then the selection of the processes that will enter the critical section next cannot be postponed indefinitely.

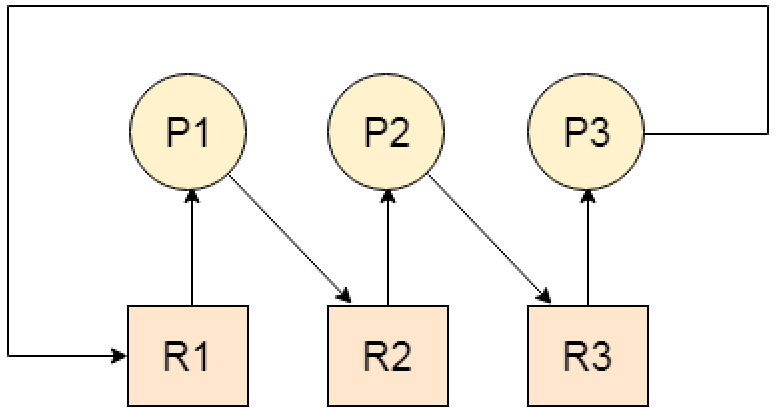
Bounded Waiting

- A bound must exist on the number of times that other processes are allowed to enter their critical sections after a process has made a request to enter its critical section and before that request is granted.

Deadlock

- ***Deadlock*** is a situation where a set of processes are blocked because each process is holding a resource and waiting for another resource acquired by some other process.
- A set of processes is in a ***deadlock state*** when every process in the set is waiting for a resource that can only be released by another process in the set.





Process 1 is holding Resource 1 and waiting for resource 2 which is acquired by process 2, and process 2 is waiting for resource 1.

Conditions for Deadlock

Mutual exclusion

- At least one resource must be held in a non-sharable mode; that is, only one process at a time can use the resource.
- If another process requests that resource, the requesting process must be delayed until the resource has been released.

Hold and wait

- A process must be holding at least one resource and waiting to acquire additional resources that are currently being held by other processes.

No preemption.

- only the process can release its held resource
- A resource can be released only voluntarily by the process holding it, after that process has completed its task.

Circular wait

- A set { P0 , Pl, ... , Pn } of waiting processes must exist such that
- Po is waiting for a resource held by P1,
- P1 is waiting for a resource held by P2, ... ,
- Pn-1 is waiting for a resource held by Pn,
- Pn is waiting for a resource held by Po.

Methods for handling deadlock

1. Deadlock Detection
2. Deadlock Recovery
3. Deadlock Prevention
4. Deadlock Avoidance

Deadlock Detection

- Deadlock can be detected by the resource scheduler as it keeps track of all the resources that are allocated to different processes.
- After a deadlock is detected, it can be handed using the given methods:
  1. All the processes that are involved in the deadlock are terminated. This approach is not that useful as all the progress made by the processes is destroyed.
  2. Resources can be preempted from some processes and given to others until the deadlock situation is resolved.

Deadlock Recovery

- In order to recover the system from deadlocks, either OS considers resources or processes.

For Resource

Preempt the resource

- Snatch one of the resources from the owner of the resource (process) and give it to the other process with the expectation that it will complete the execution and will release this resource sooner.

Rollback to a safe state

- System passes through various states to get into the deadlock state. The operating system can rollback the system to the previous safe state. For this purpose, OS needs to implement check pointing at every state.

For Process

Kill a process

- Killing a process can solve our problem but the bigger concern is to decide which process to kill. Generally, Operating system kills a process which has done least amount of work until now.

Kill all process

- This is not a suggestible approach but can be implemented if the problem becomes very serious. Killing all process will lead to inefficiency in the system because all the processes will execute again from starting.

Deadlock Prevention

Deadlock prevention ensures that deadlock is excluded from the beginning by invalidate at least one of the four necessary conditions, however deadlock prevention is often impossible to implement.

No Mutual Exclusion Condition

- Shared resources such as read-only files do not lead to deadlocks.
- Non-blocking synchronization algorithms can avoid mutual exclusion.

No Hold and Wait

- When a process ready to execute and requires some resources then all resources should be allocated at once that means there will not be wait for required resources. But sometime a process can be suffer from starvation and very low resource utilization.

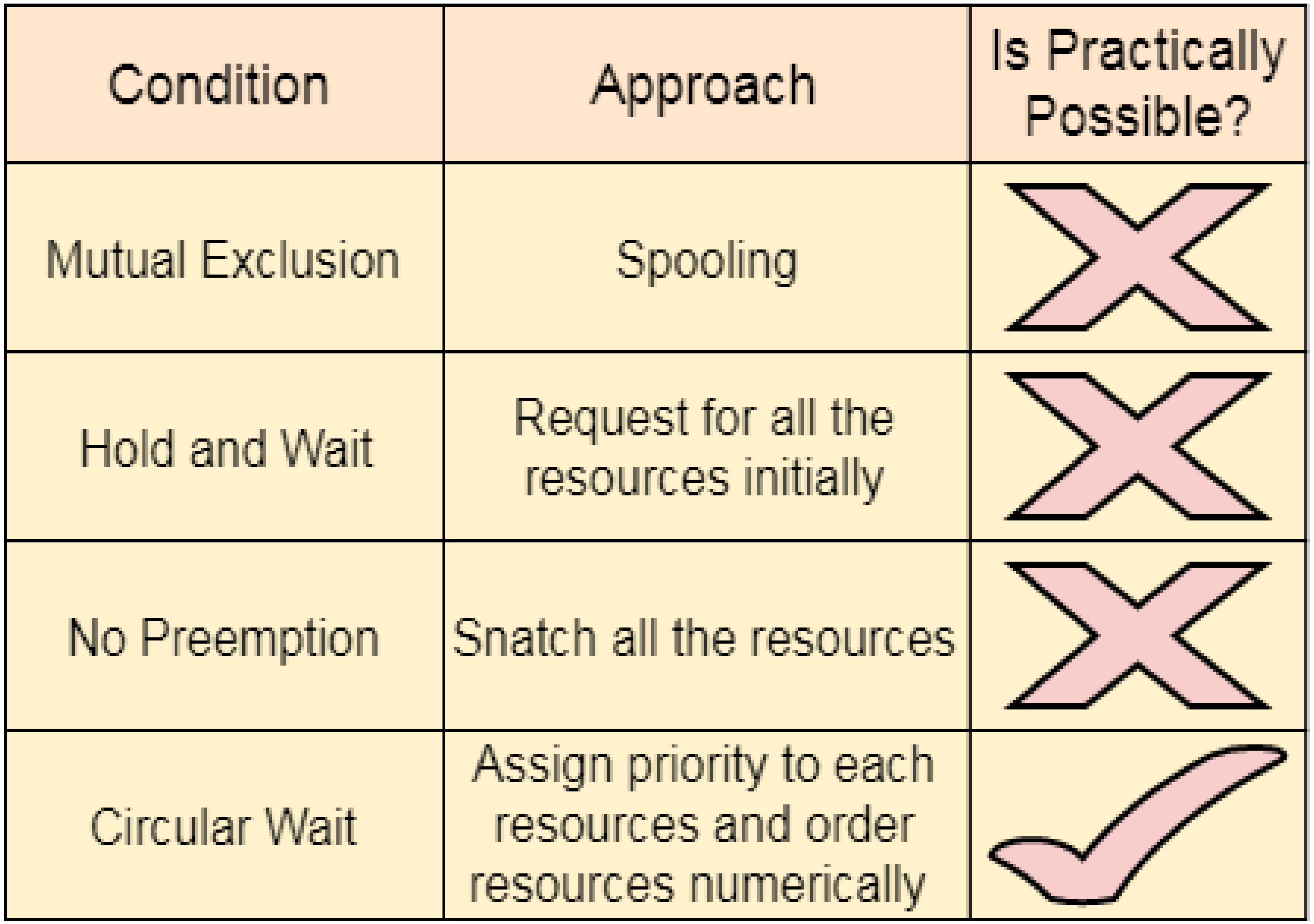
No Preemption

- When a process is holding some resources and waiting for another resources that can not be allocated to it, then this process releases all resources so other process can complete their execution. But some resources like printer, tap drivers can not be preempted.

No Circular Wait

- To avoid circular wait, processes must request resources in increasing order only. But resource numbering may affects efficiency and a process may have to request a resource before it need.

Deadlock Prevention



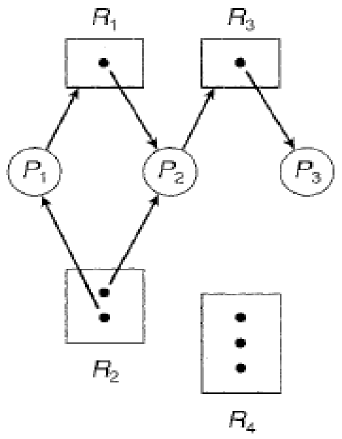
Condition	Approach	Is Practically Possible?
Mutual Exclusion	Spooling	✗
Hold and Wait	Request for all the resources initially	✗
No Preemption	Snatch all the resources	✗
Circular Wait	Assign priority to each resources and order resources numerically	✓

Deadlock Avoidance

- Deadlock avoidance algorithm ensures that a processes will never enter into unsafe or deadlock.
- The system requires additional **prior information** regarding potential use of each resource for each process that means each process declare the maximum number of resources of each type that it may need, number of available resources, allocated resources, maximum demand of the processes.
- Processes inform operating system in advance that how many resources they will need.
- If we allocated resources in a order such that requirement can be satisfied for each process and deadlock can not be occur then this state is called as **safe state.**

Deadlock Avoidance algorithms

- Resource allocation graph(RAG)
  - states resources is held by which process and which process is waiting for a resource of a particular type.
    - G(V,E)
    - V:
    - P = { P1, P2, P3 },
    - R = { R1, R2, R3 }
    - E:
  - request edge: { P1 -> R3, P2 -> R1, P3 -> R2 }
  - assignment edge: { R1 <- P1, R2 <- P2, R3 <- P3 }
  - process can be shown by using a "circle“
  - resources can be shown by using "rectangle", whereas one dot inside rectangle indicates only one instance of a resource is available, and two dots indicate two instances of the resource are available.



- Bankers algorithm
  - tests for safety by simulating the allocation for predetermined maximum possible amounts of all resources, then makes an “s-state” check to test for possible activities, before deciding whether allocation should be allowed to continue.

Memory Management

- Is the task carried out by the OS and hardware to accommodate multiple processes in main memory
- If only a few processes can be kept in main memory, then much of the time all processes will be waiting for I/O and the CPU will be idle
- Hence, memory needs to be allocated efficiently in order to pack as many processes into memory as possible

Memory Management Requirement

Relocation

- programmer cannot know where the program will be placed in memory when it is executed
- a process may be (often) relocated in main memory due to swapping
- swapping enables the OS to have a larger pool of ready-to-execute processes

Protection

- processes should not be able to reference memory locations in another process without permission
- impossible to check addresses at compile time in programs since the program could be relocated
- address references must be checked at run time by hardware

Sharing

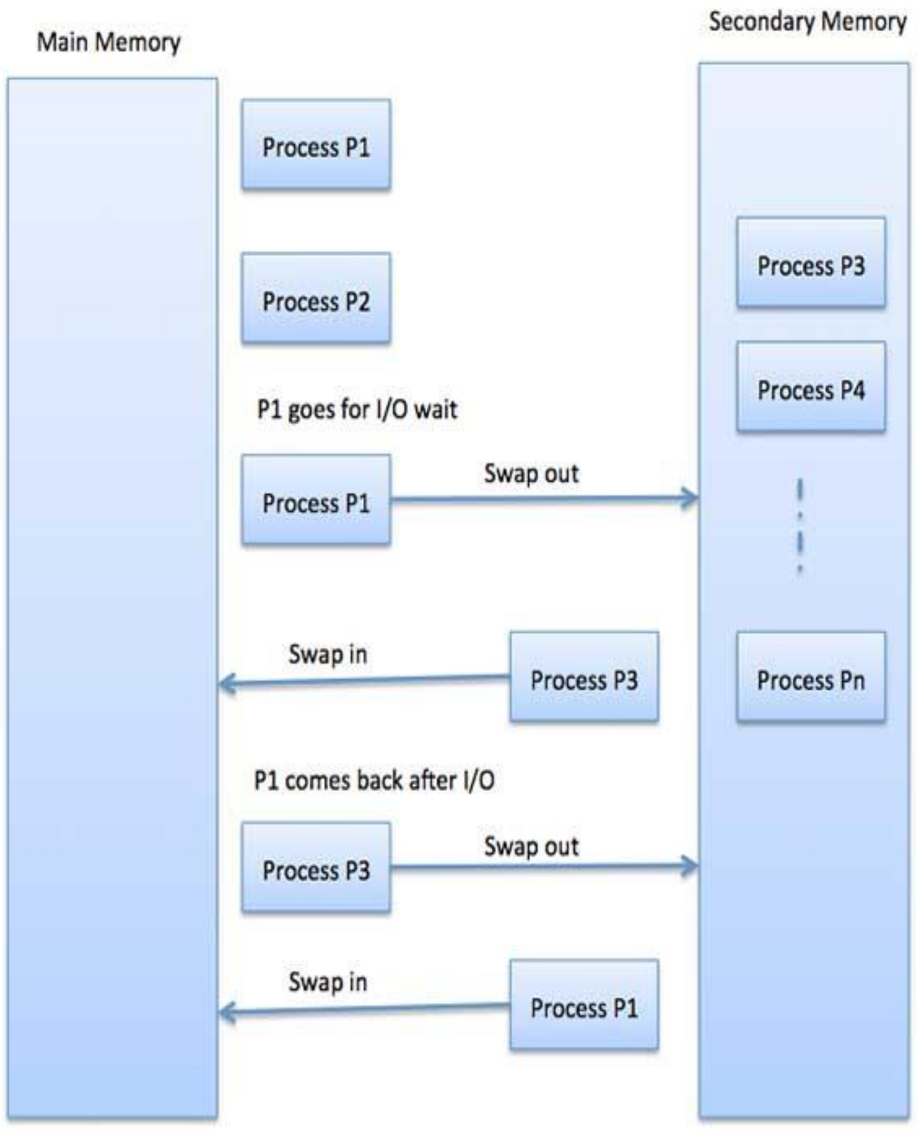
- must allow several processes to access a common portion of main memory without compromising protection
  - cooperating processes may need to share access to the same data structure
  - better to allow each process to access the same copy of the program rather than have their own separate copy

Loading Program

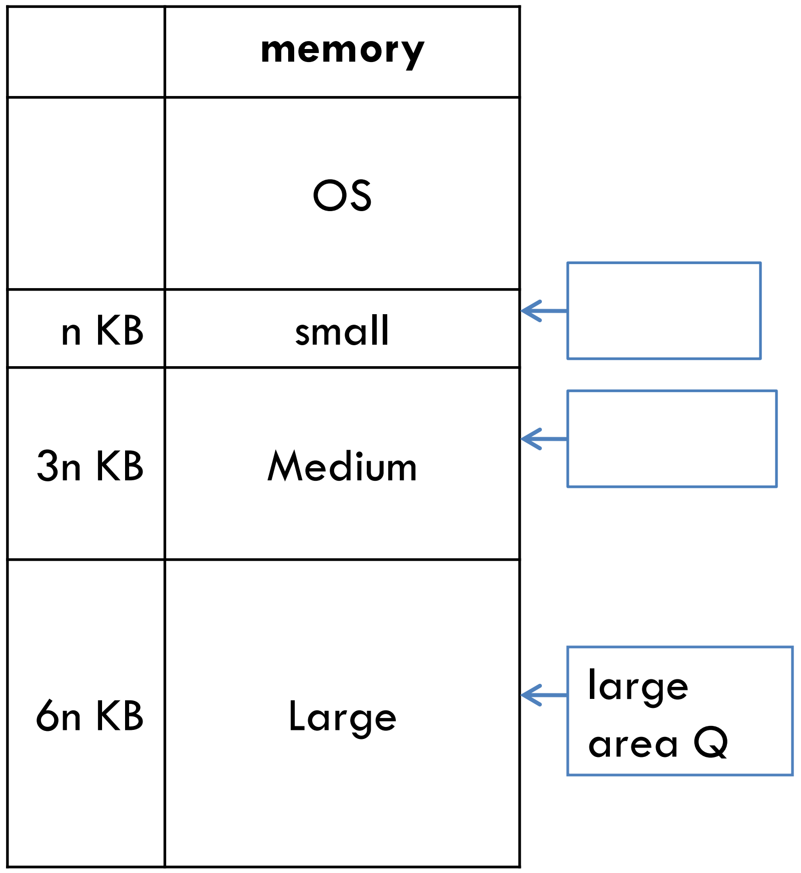
- Sometimes complete program is loaded into the memory, but some times a certain part or routine of the program is loaded into the main memory only when it is called by the program, this mechanism is called **Dynamic Loading**, this enhance the performance.
- Also, at times one program is dependent on some other program. In such a case, rather than loading all the dependent programs, CPU links the dependent programs to the main executing program when its required. This mechanism is known as **Dynamic Linking**.

Swapping

- Swapping is a mechanism in which a process can be swapped temporarily out of main memory (or move) to secondary storage (disk) and make that memory available to other processes.
- the system swaps back the process from the secondary storage to main memory.
- it helps in running multiple and big processes in parallel and that's the reason **Swapping is also known as a technique for memory compaction**.



Fixed Partition



•Processes are classified on entry to the system according to their memory requirements.

•We need one *Process Queue (PQ)* for each class of process.

•If a process is selected to allocate memory, then it goes into memory and competes for the processor.

•The number of fixed partition gives the degree of multiprogramming.

•Since each queue has its own memory region, there is no competition between queues for the memory.

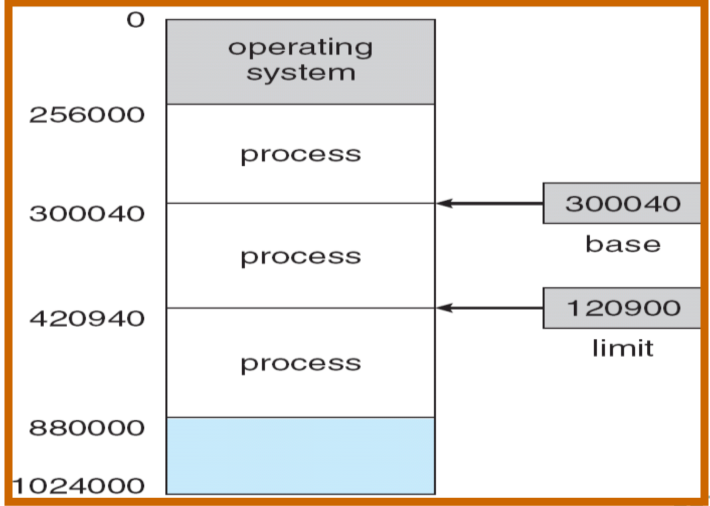
•The main problem with the fixed partitioning method is how to determine the number of partitions, and how to determine their sizes.

Variable Partition

- Initially, the whole memory is free and it is considered as one large block.
- When a new process arrives, the OS searches for a block of free memory large enough for that process.
- We keep the rest available (free) for the future processes.
- If a block becomes free, then the OS tries to merge it with its neighbors if they are also free.

Base and Limit Register

•A pair of **base** and **limit** registers define the logical address space



Logical versus Physical Address Space

- An address generated by the CPU is commonly referred to as a logical address.
- an address seen by the memory unit-that is, the one loaded into the memory address register of the memory is physical address.
- **Logical address** – generated by the CPU; also referred to as **virtual address**
- **Physical address** – address seen by the memory unit
- Logical and physical addresses are the same in compile-time and load-time address-binding schemes; logical (virtual) and physical addresses differ in execution-time address-binding scheme

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

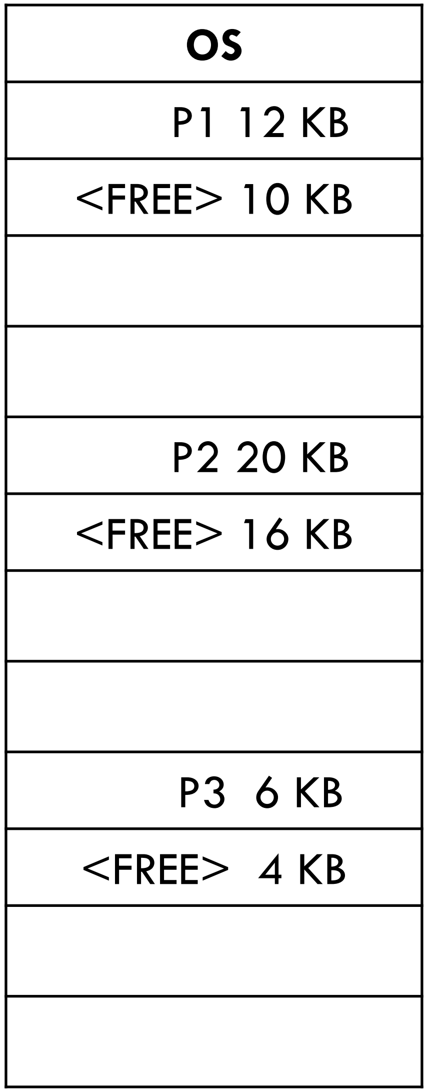
First-fit and best-fit better than worst-fit in terms of speed and storage utilization

first fit

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

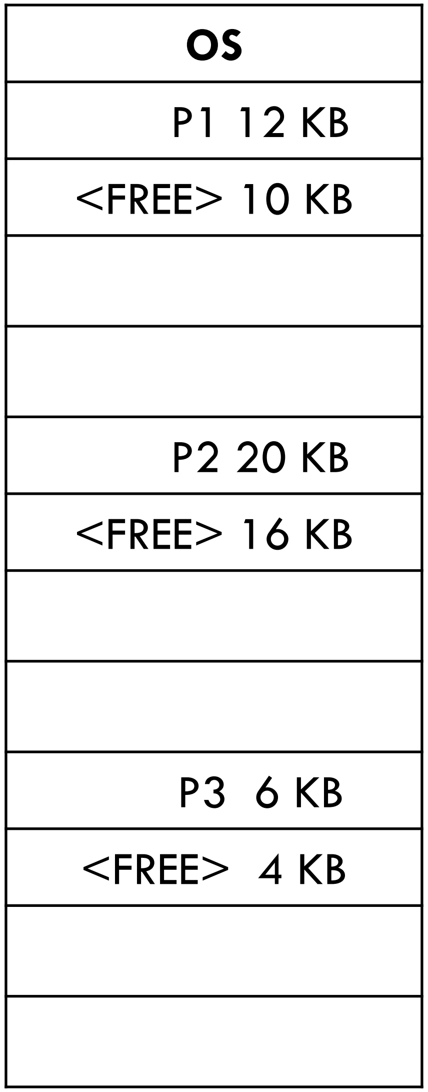
first fit

Initial memory mapping

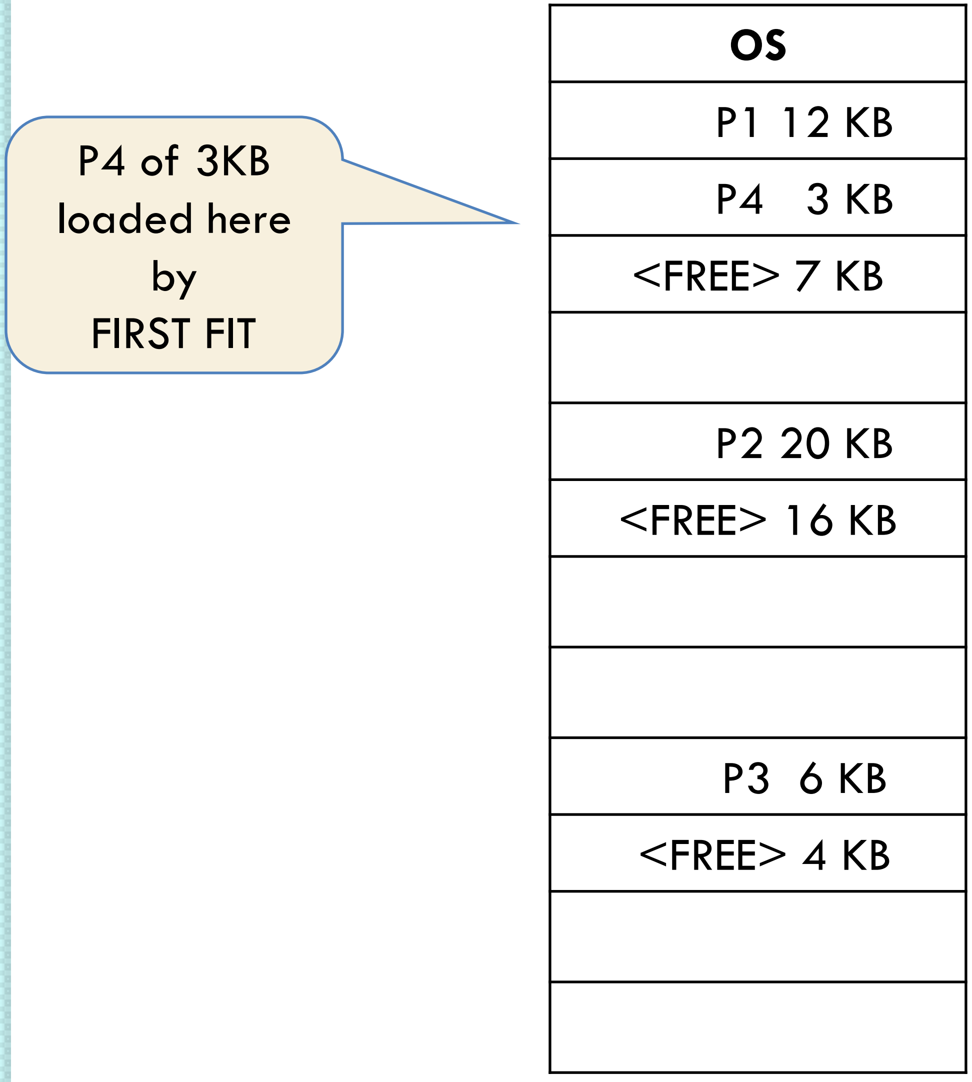


first fit

P4 of 3KB arrives

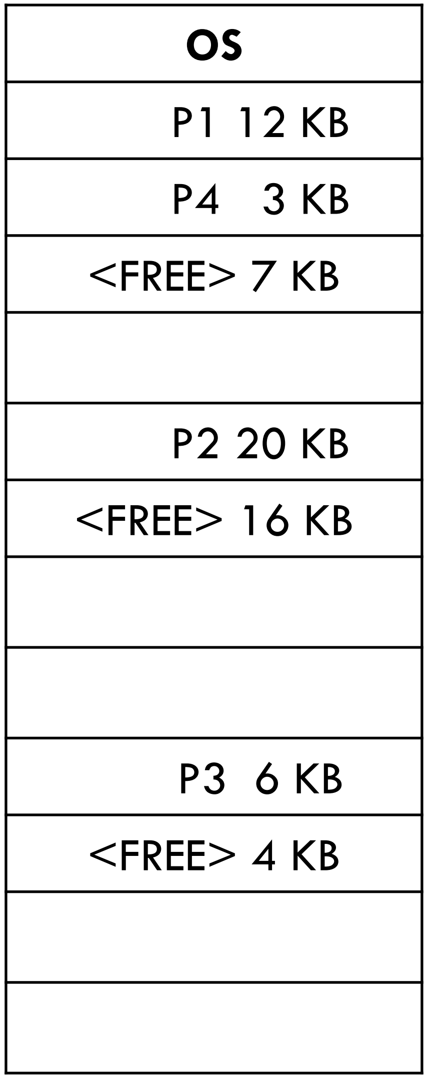


first fit

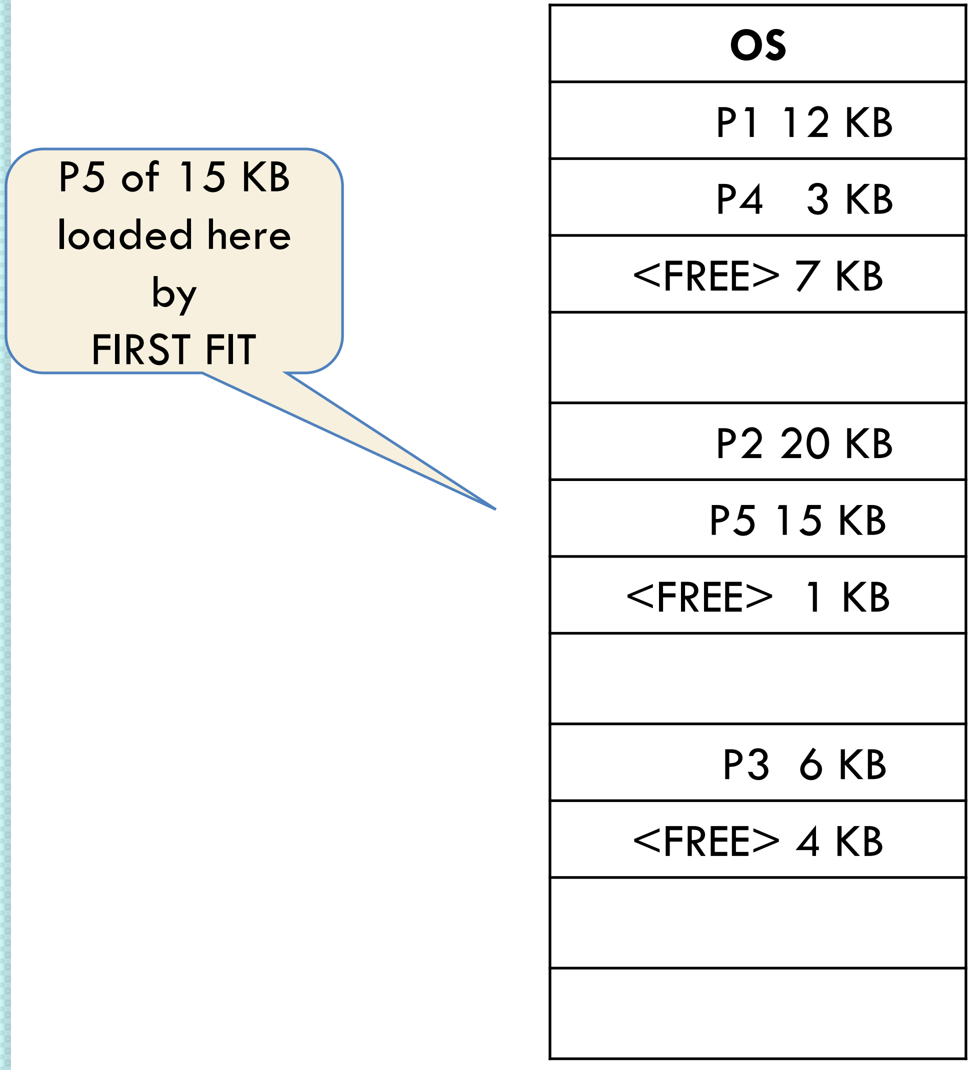

P4 of 3KB loaded here by FIRST FIT
OS
P1 12 KB
P4 3 KB
<FREE> 7 KB
P2 20 KB
<FREE> 16 KB
P3 6 KB
<FREE> 4 KB

first fit

P5 of 15KB arrives



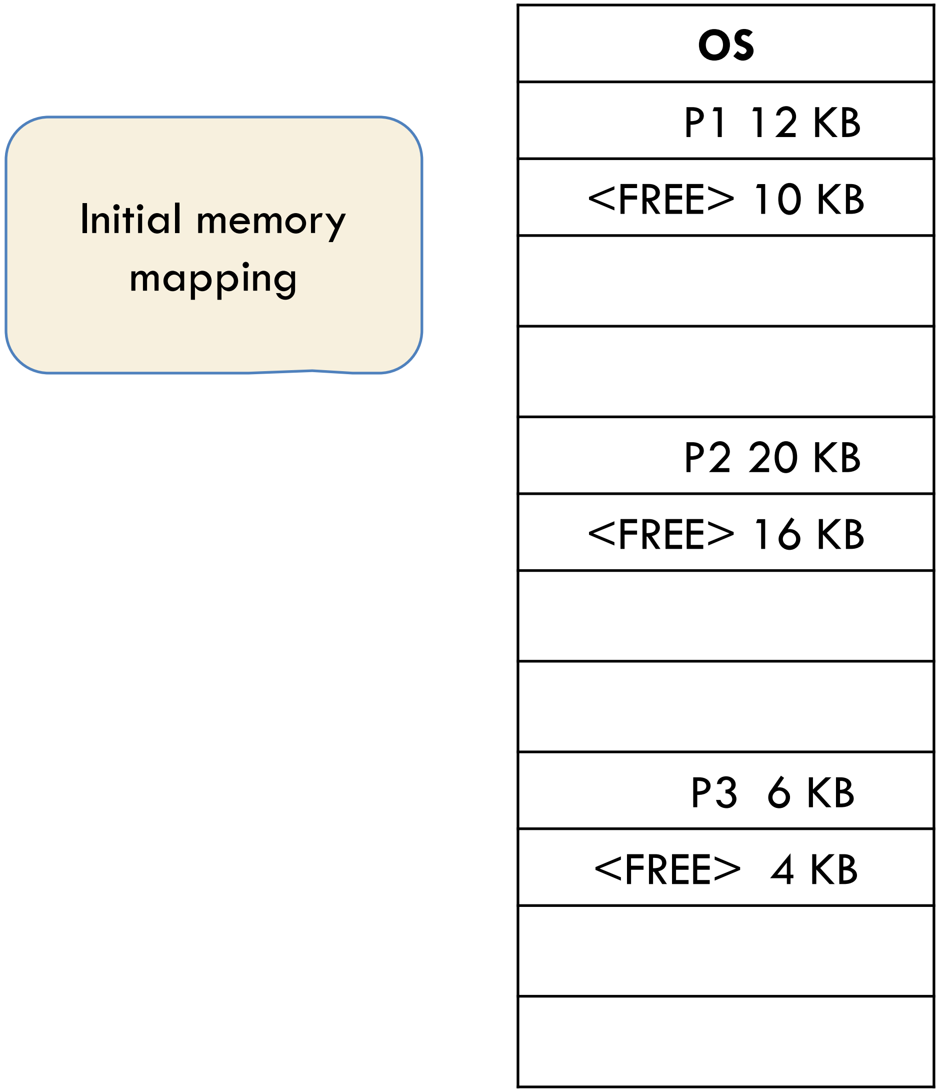
first fit


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

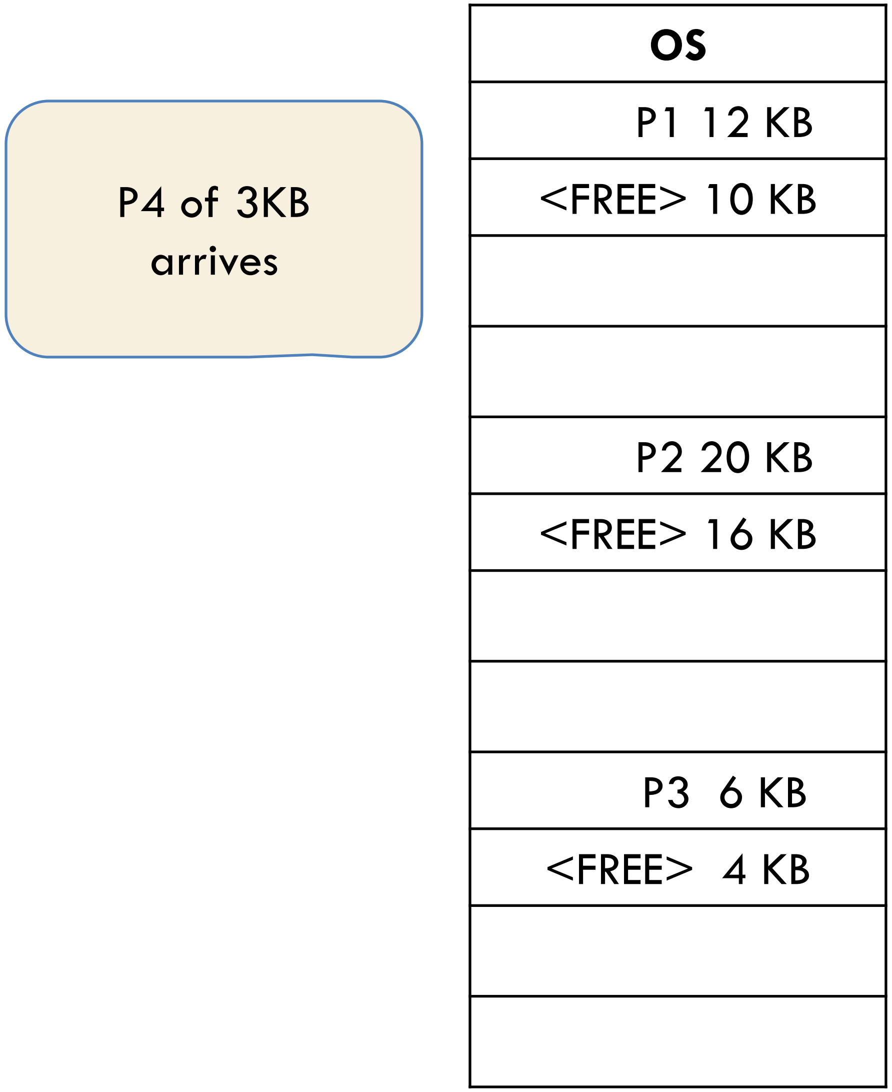
Best fit

- 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.

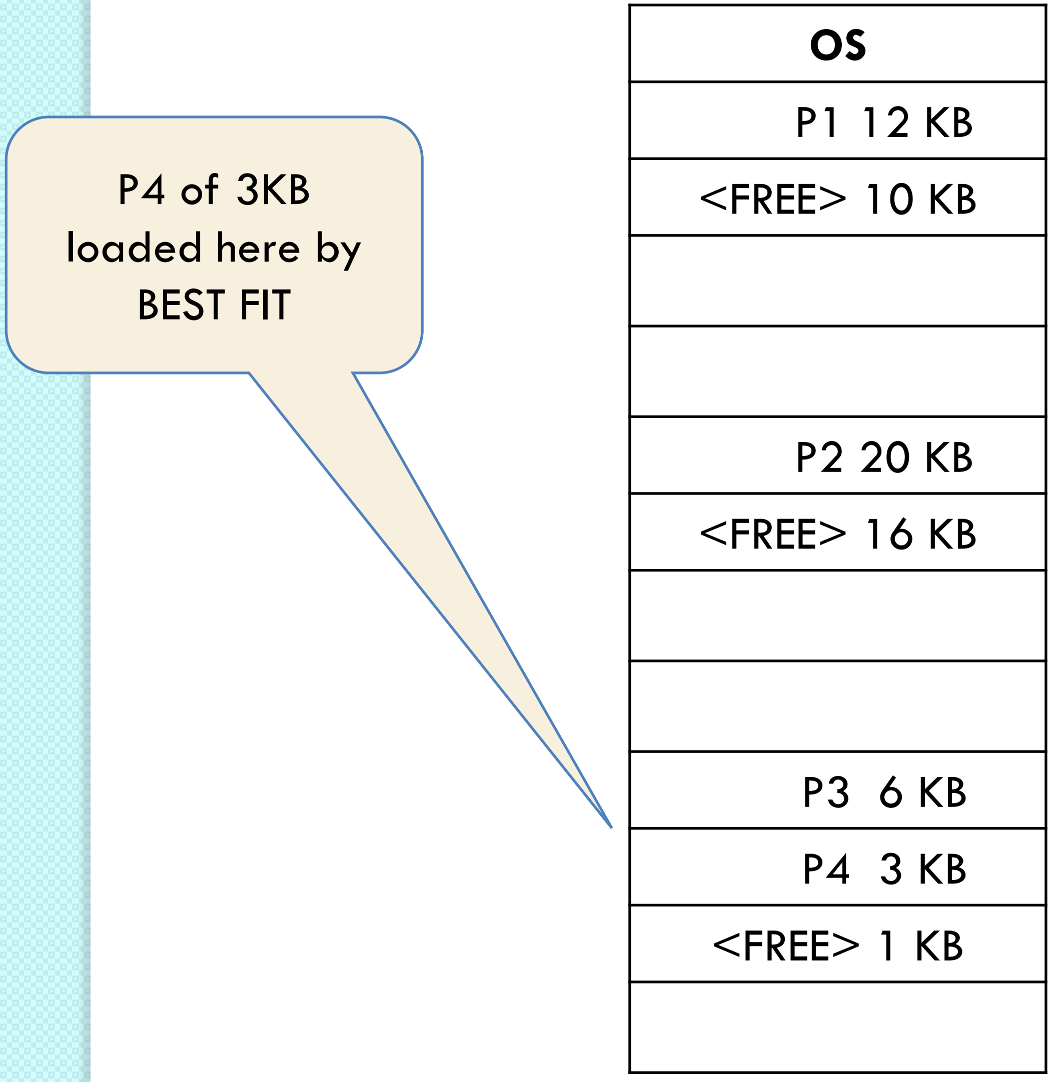
best fit


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

best fit

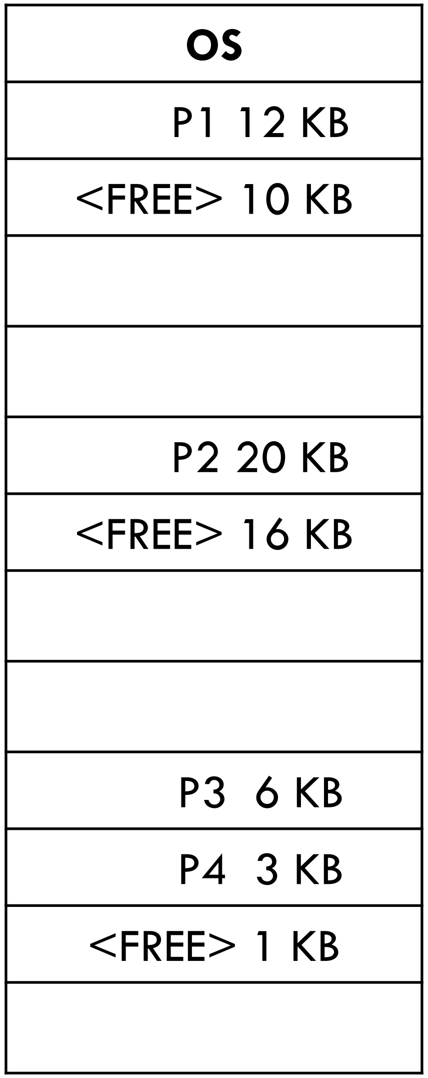


best fit

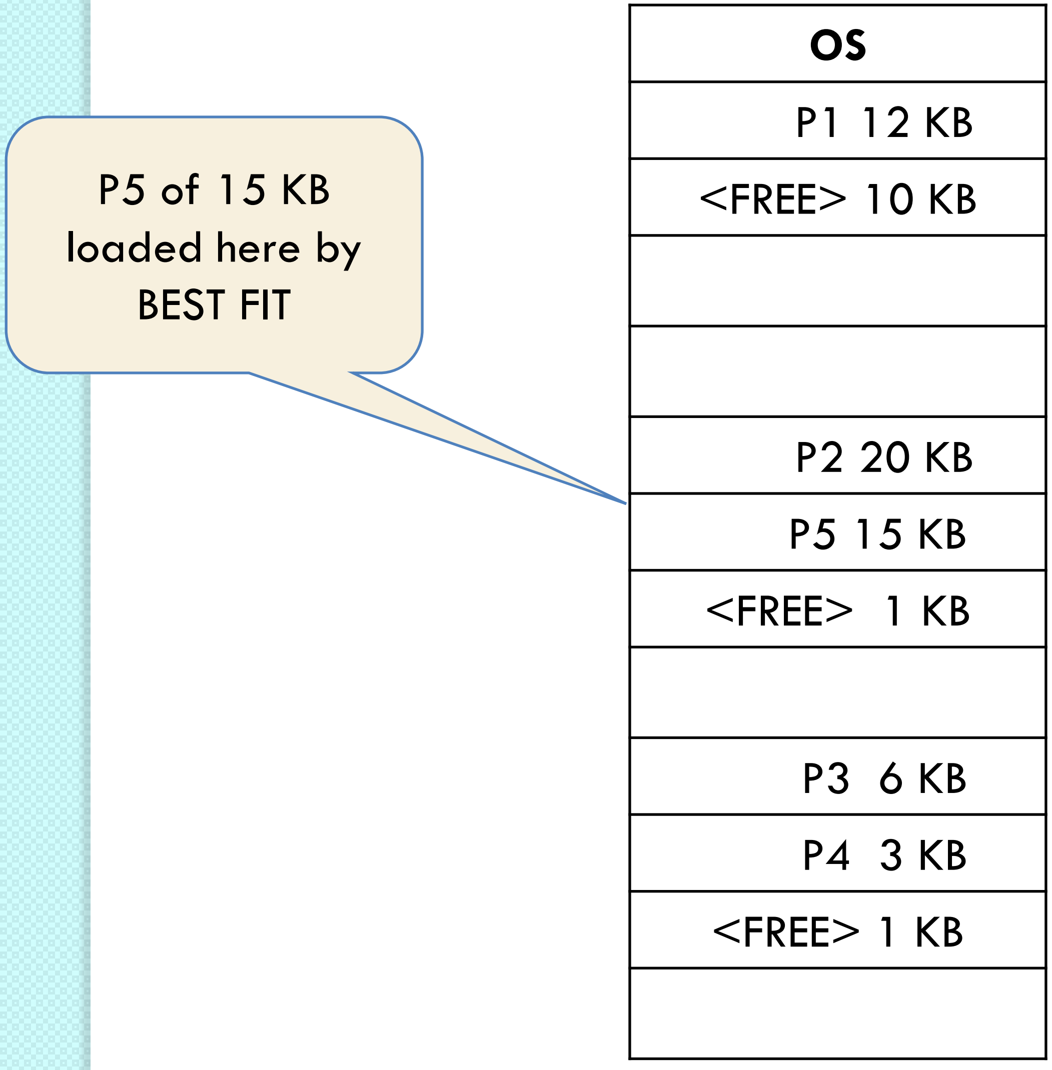

P4 of 3KB loaded here by BEST FIT
OS
P1 12 KB
<FREE> 10 KB
P2 20 KB
<FREE> 16 KB
P3 6 KB
P4 3 KB
<FREE> 1 KB

best fit

P5 of 15KB arrives



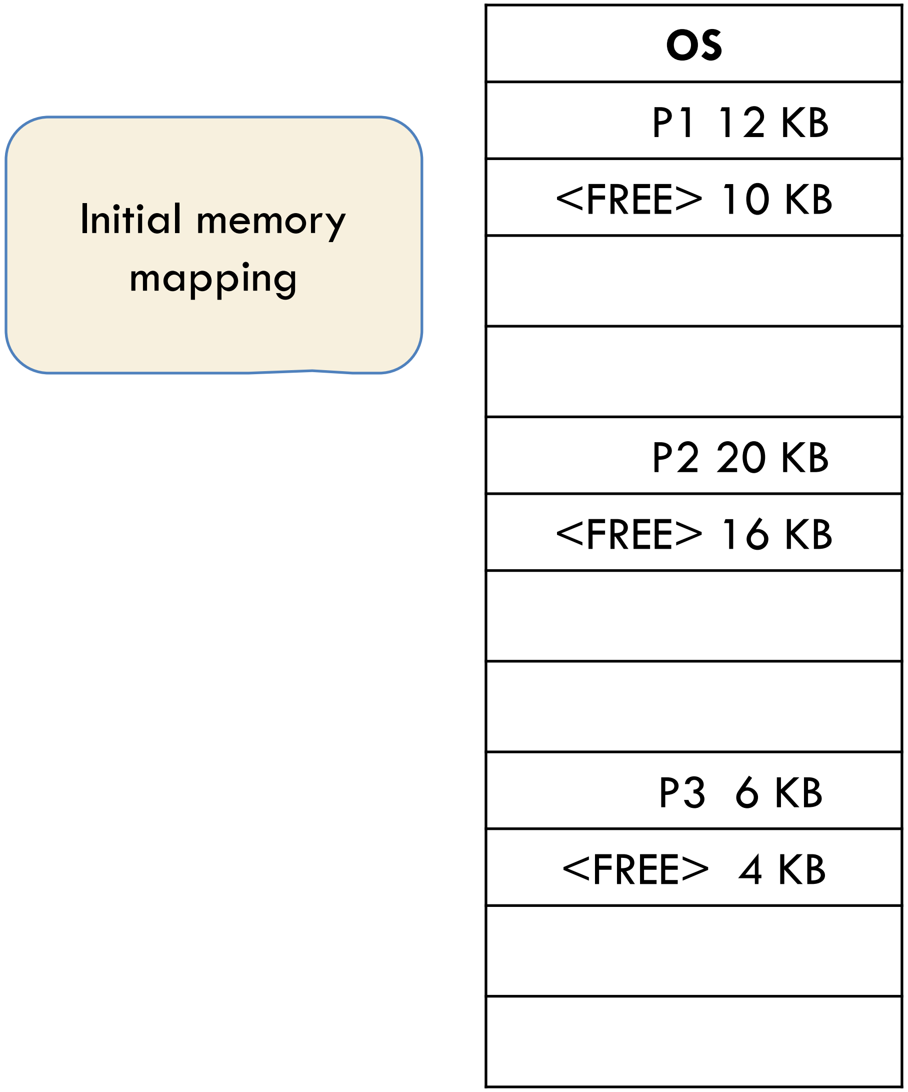
best fit


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

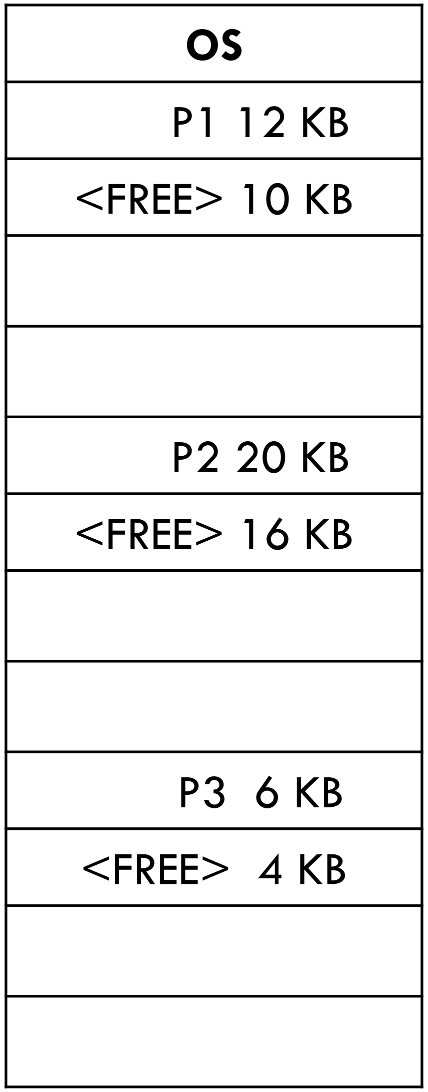
- <u>Worst Fit :</u> Allocate the largest block among those that are large enough for the new process.
- Again a search of the entire list or sorting it is needed.
- This algorithm produces the largest over block.

worst fit

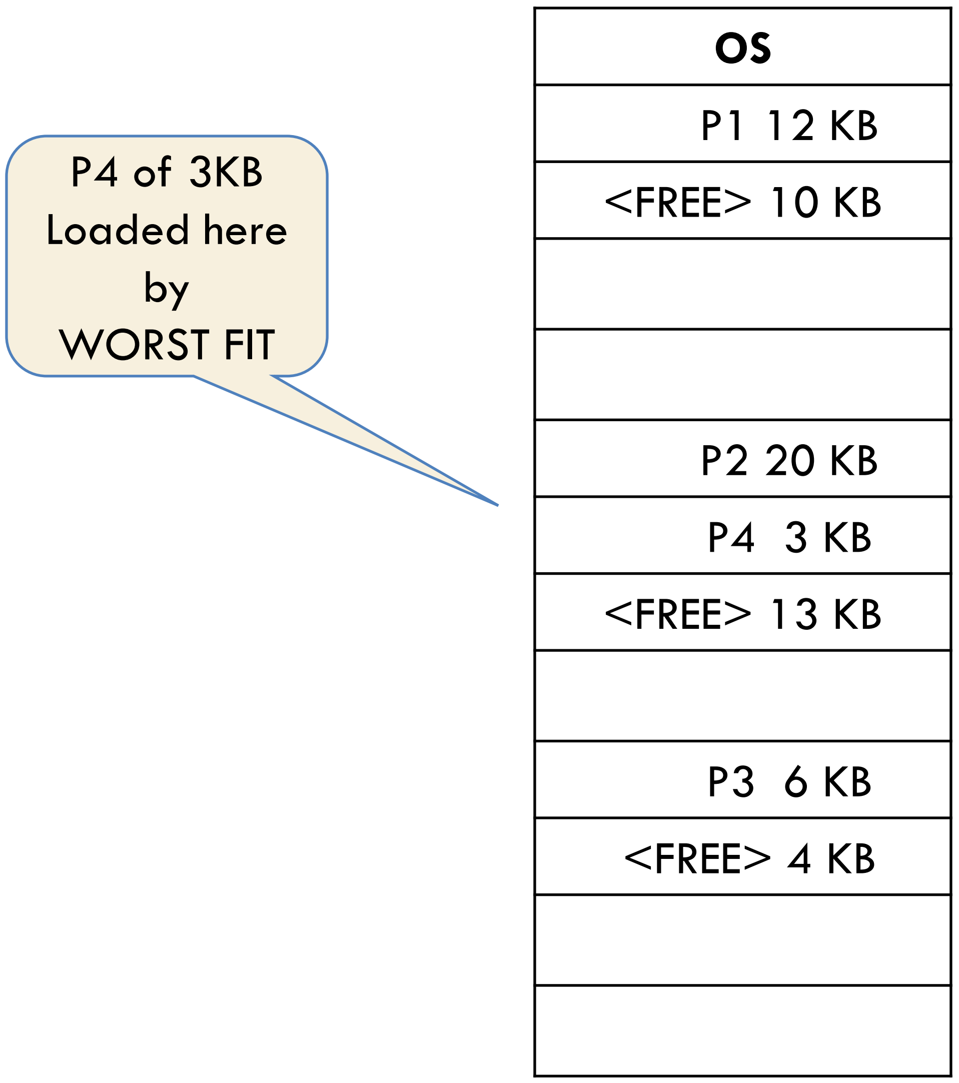

Initial memory mapping
OS
P1 12 KB
<FREE> 10 KB
P2 20 KB
<FREE> 16 KB
P3 6 KB
<FREE> 4 KB

worst fit

P4 of 3KB arrives

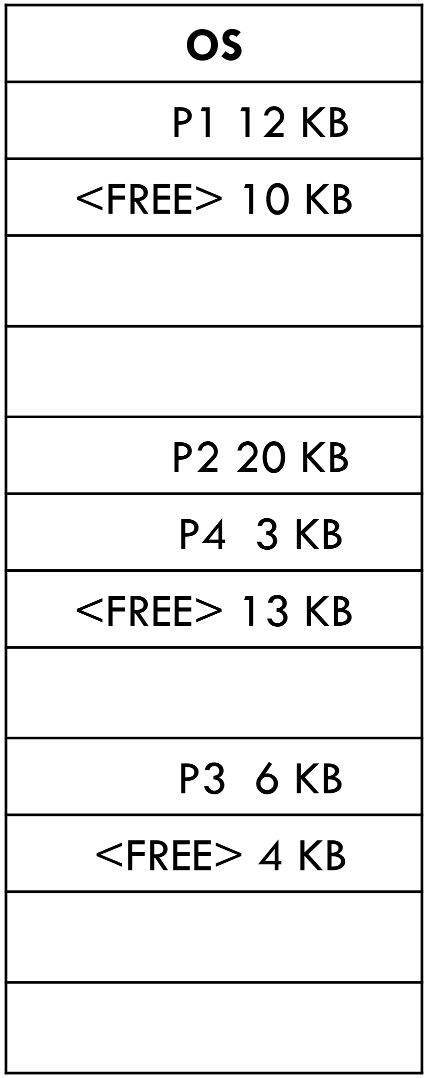


worst fit


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

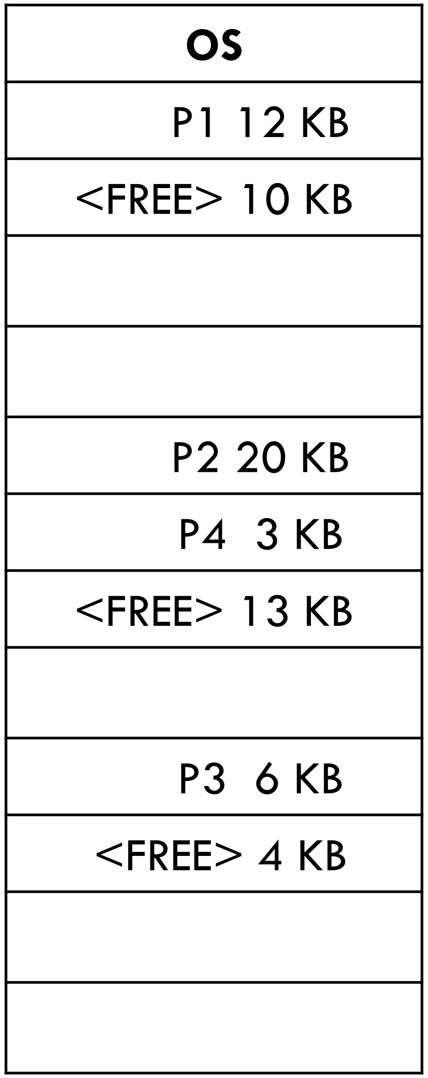
worst fit

No place to load
P5 of 15K



worst fit

No place to load P5 of 15K



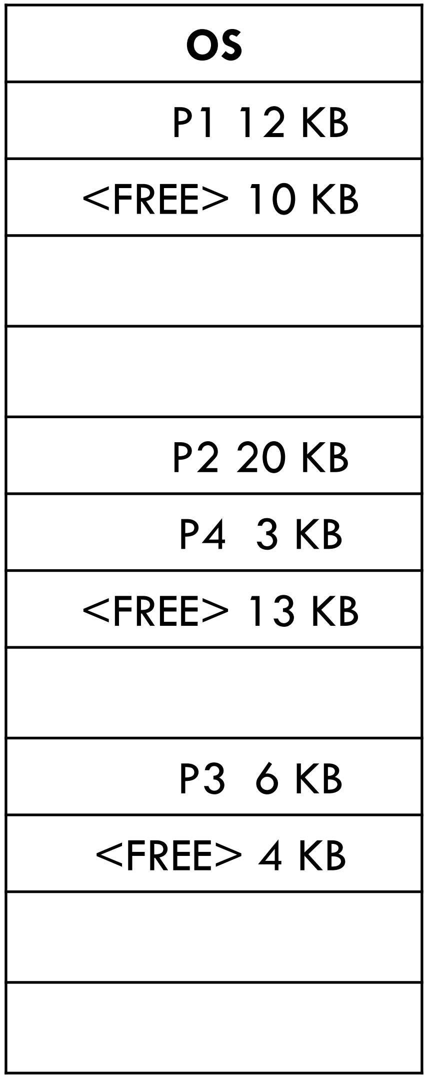
Compaction is needed !!

compaction

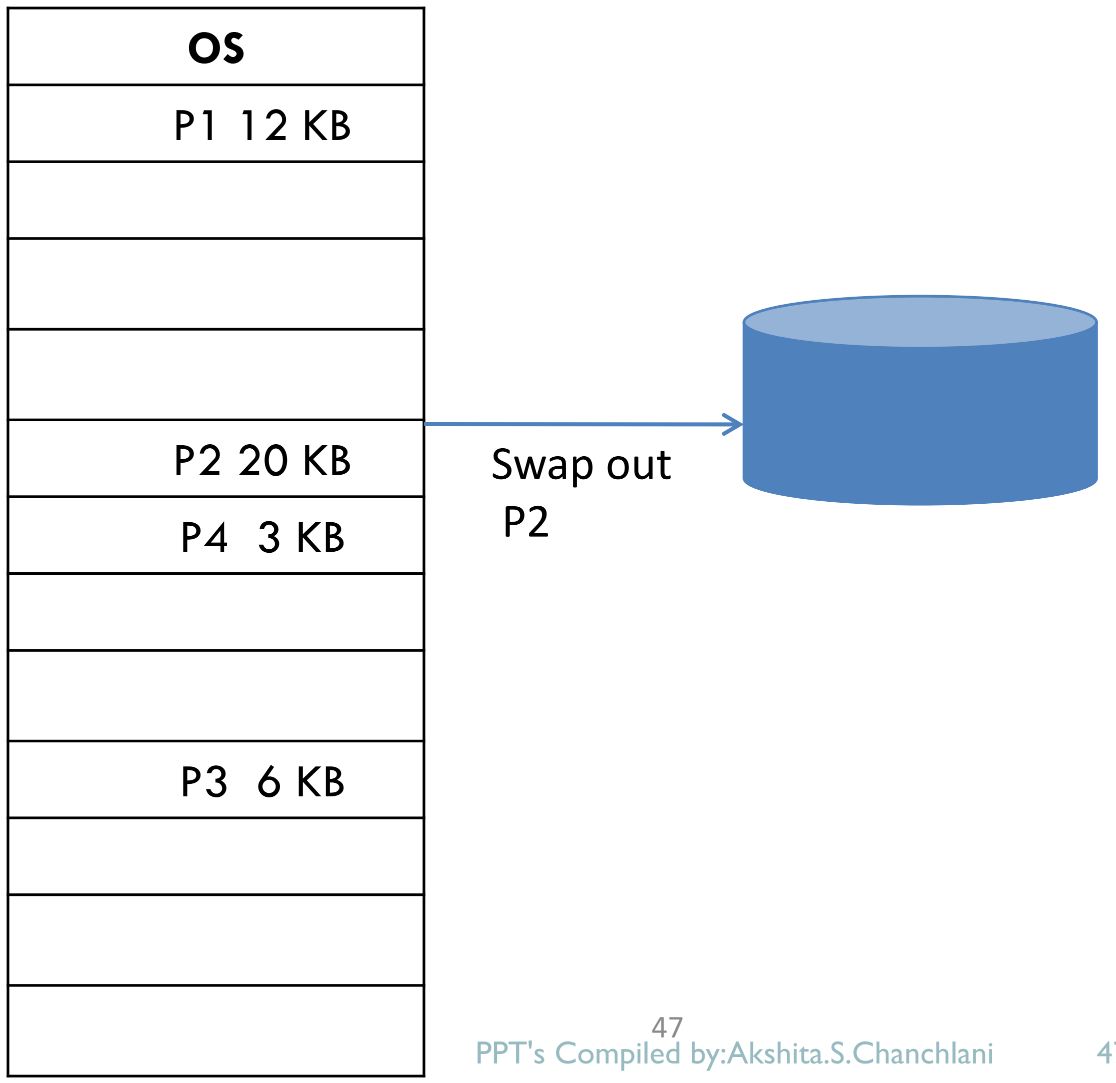
•Compaction is a method to overcome the external fragmentation problem.
•All free blocks are brought together as one large block of free space.
•Compaction requires dynamic relocation.
•One method for compaction is swapping out those processes that are to be moved within the memory, and swapping them into different memory locations

compaction

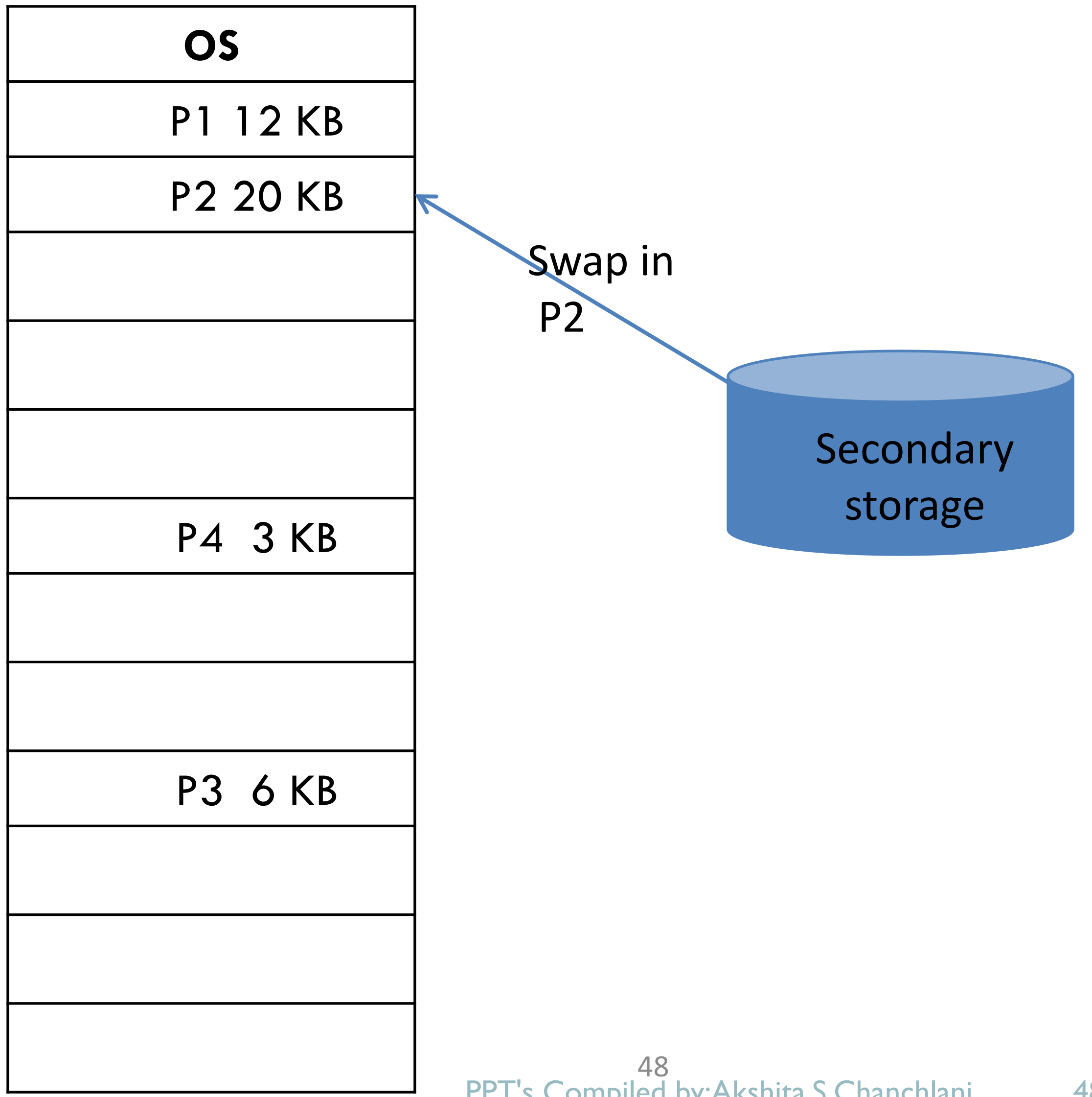
Memory mapping before compaction



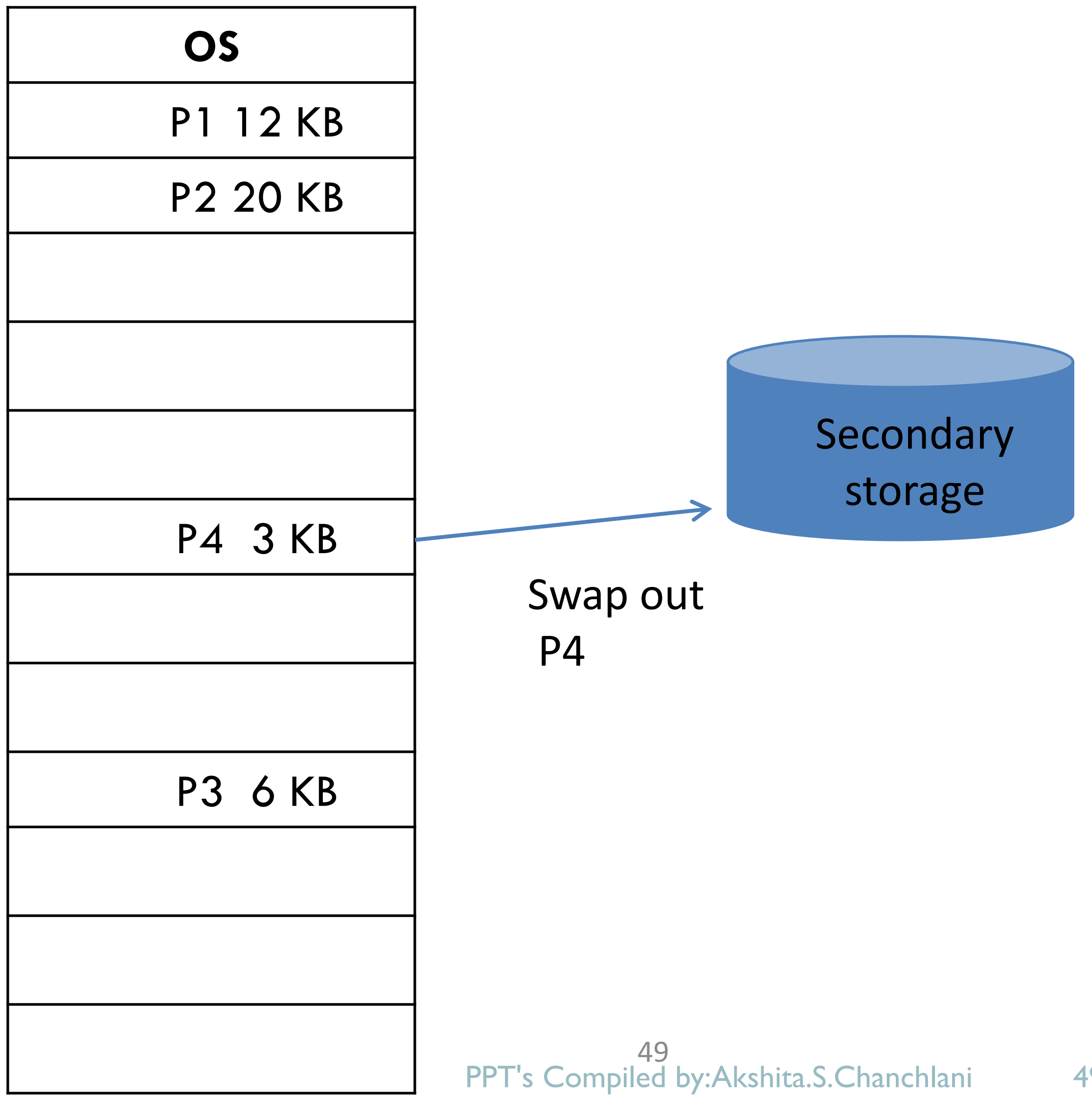
compaction



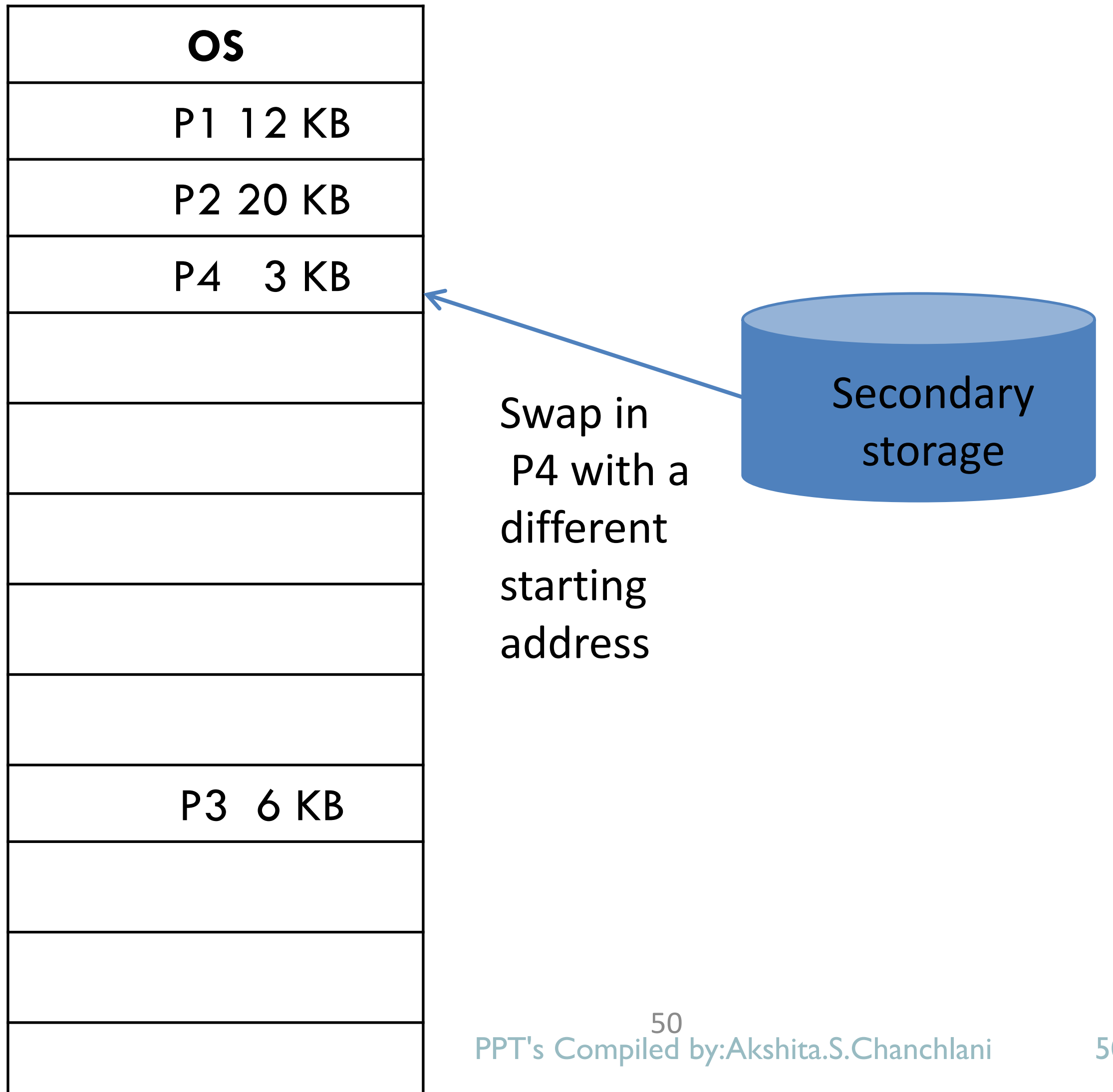
compaction



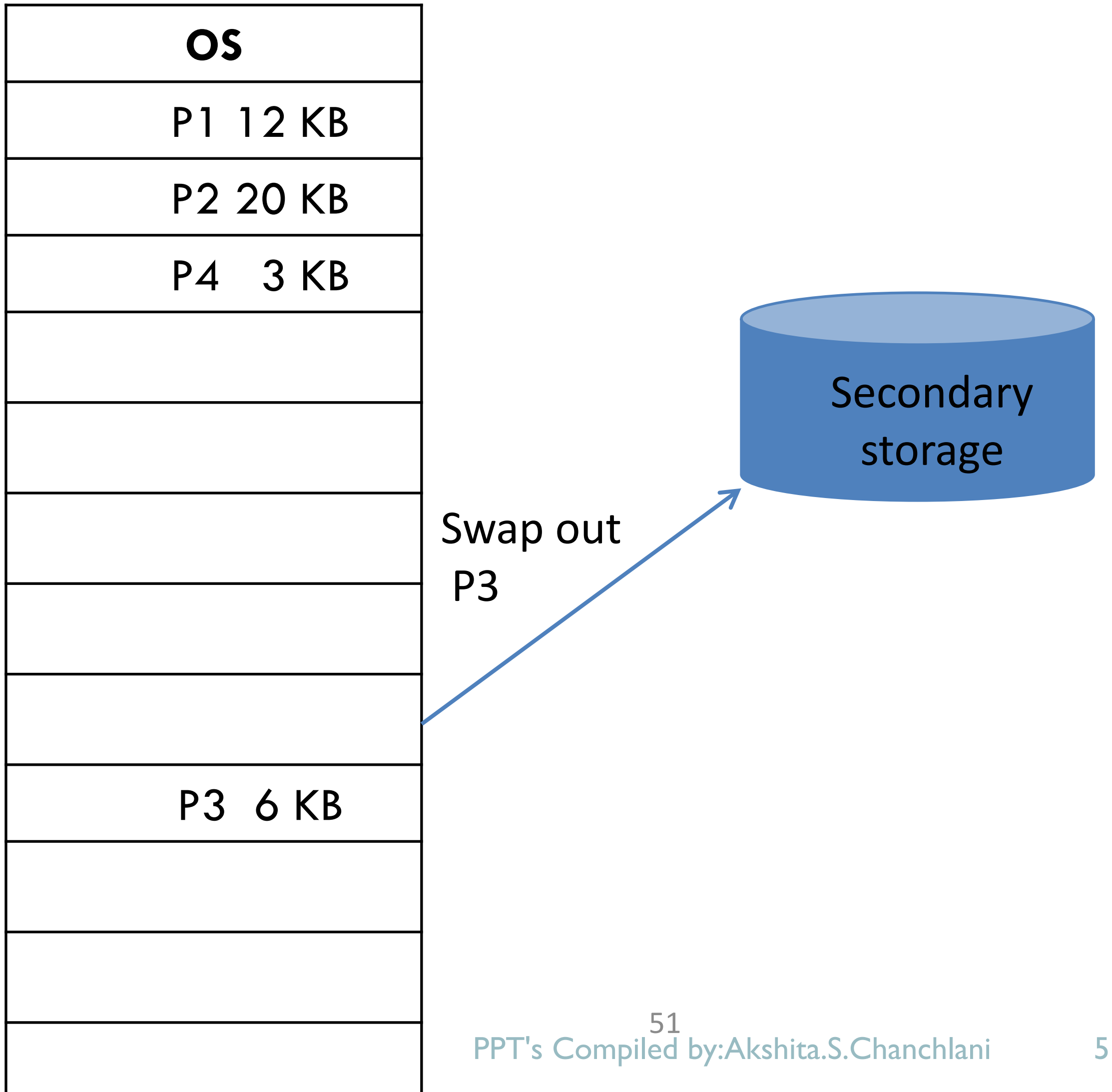
compaction



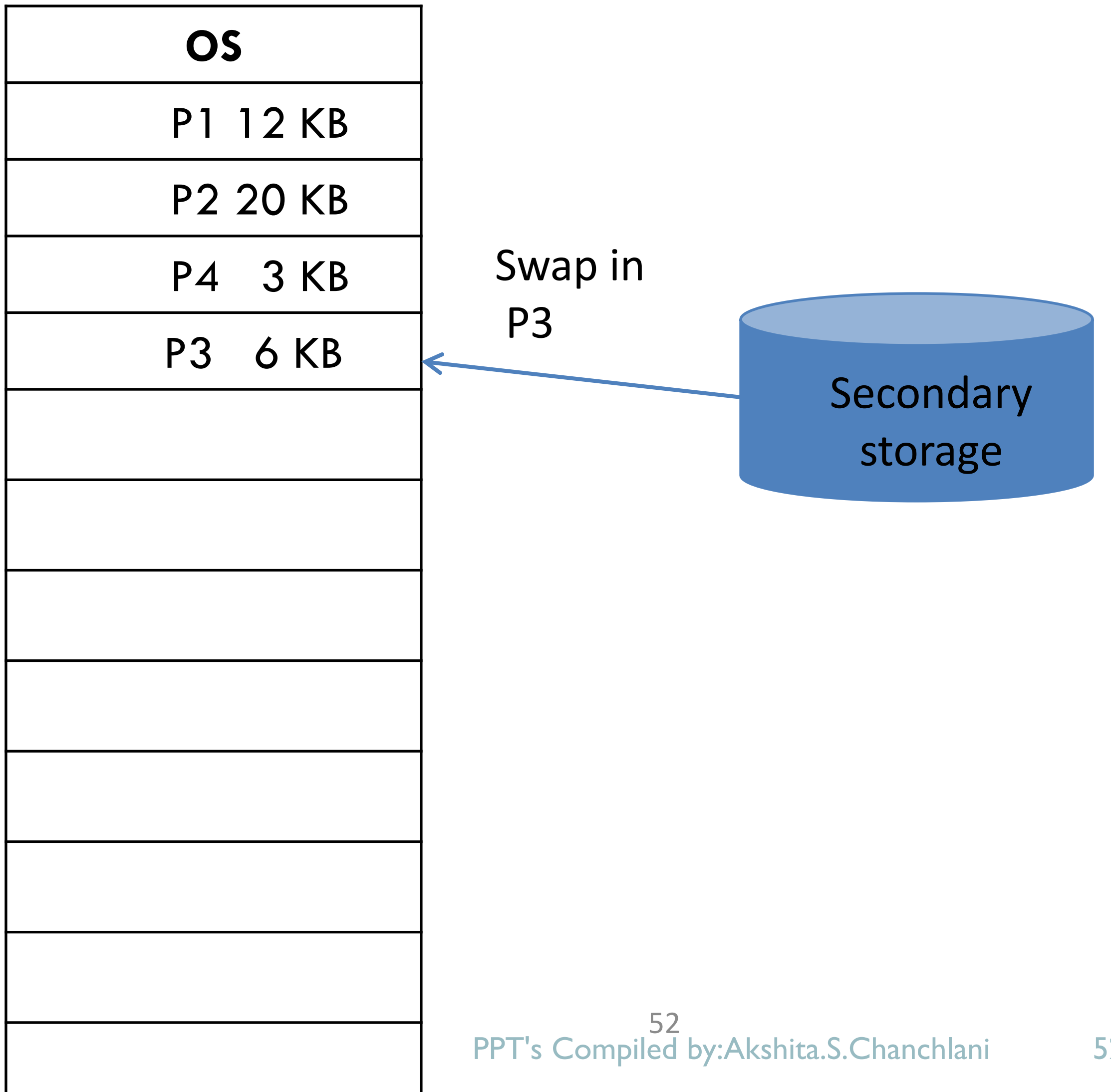
compaction


OS
P1 12 KB
P2 20 KB
P4 3 KB
P3 6 KB
Secondary storage
Swap in P4 with a different starting address

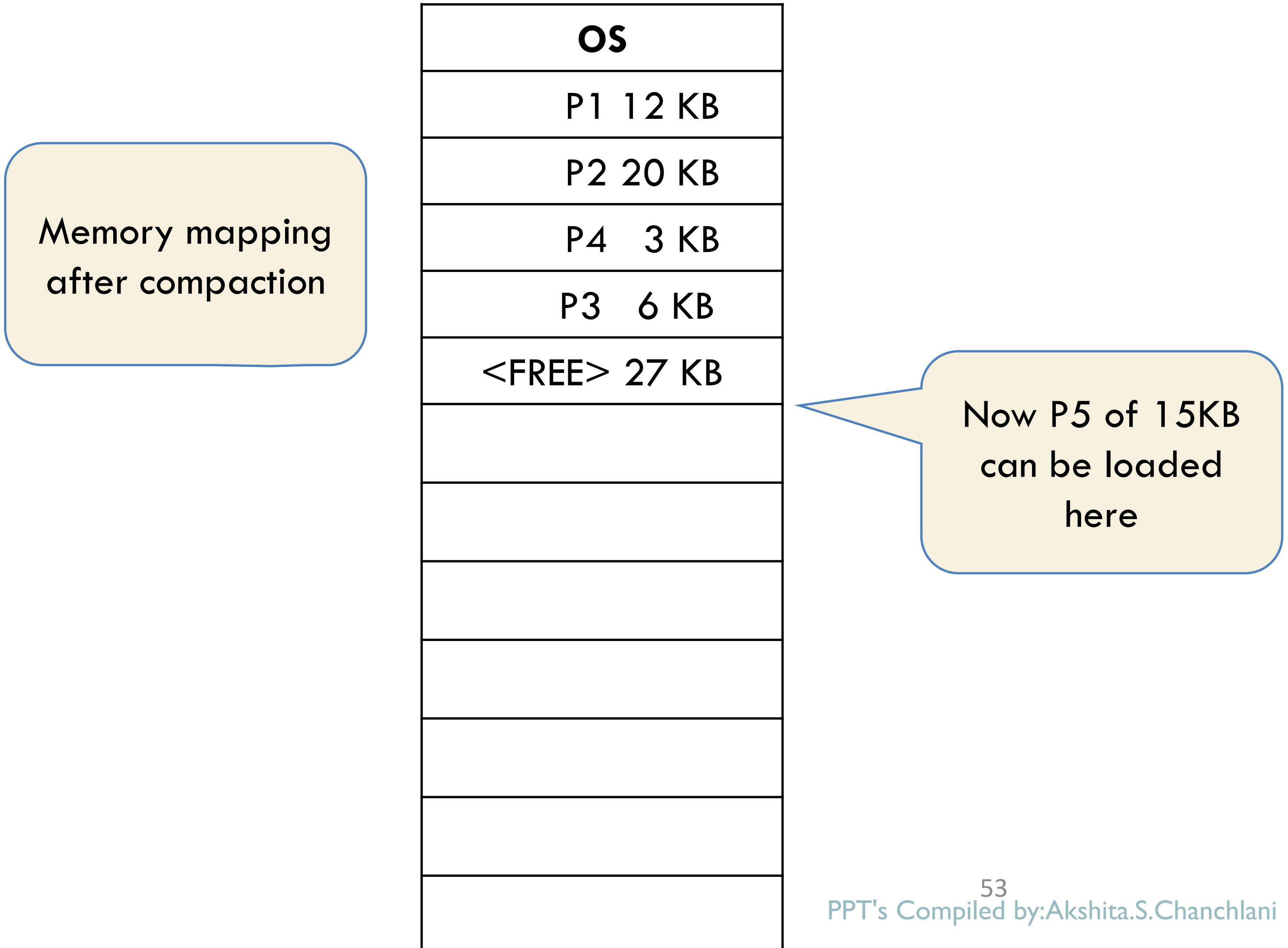
compaction



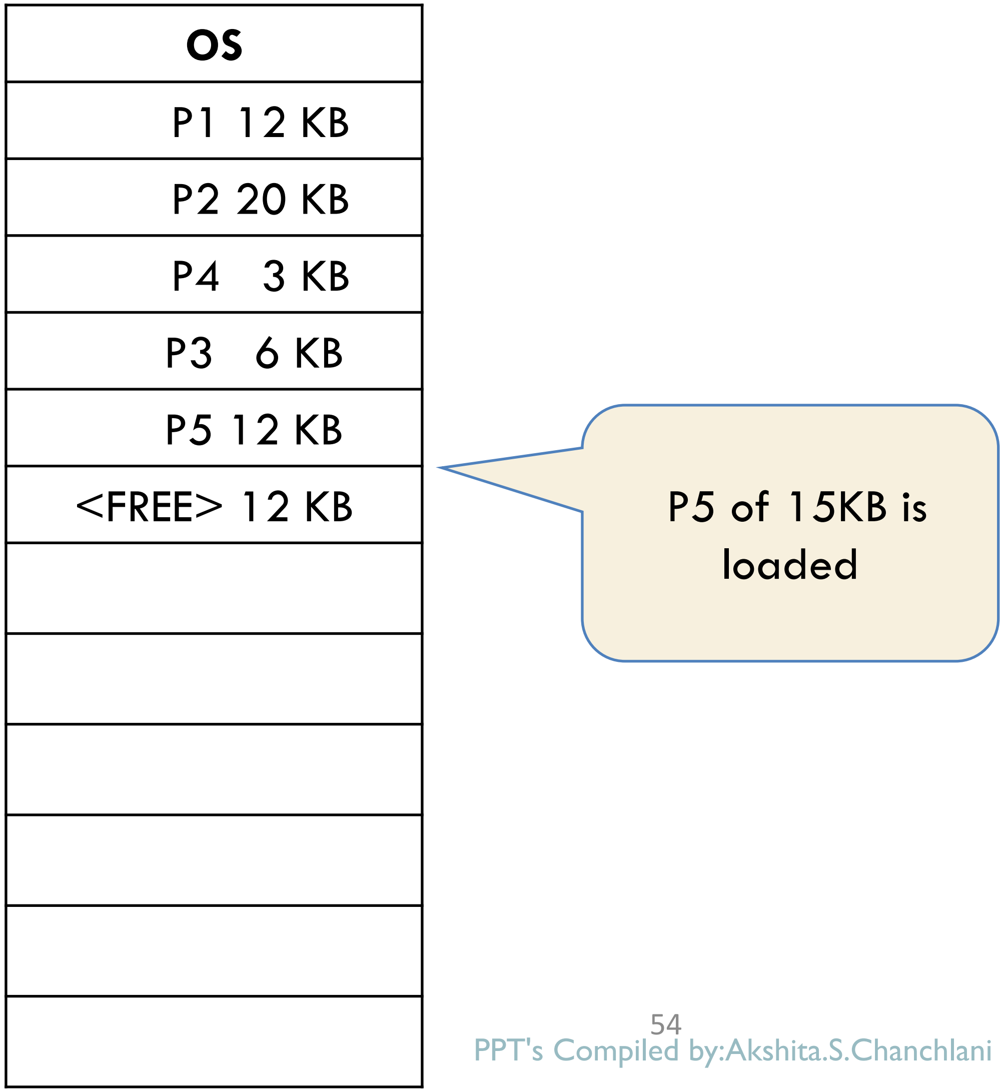
compaction



compaction


Memory mapping after compaction
OS
P1 12 KB
P2 20 KB
P4 3 KB
P3 6 KB
<FREE> 27 KB
Now P5 of 15KB can be loaded here

compaction


OS
P1 12 KB
P2 20 KB
P4 3 KB
P3 6 KB
P5 12 KB
<FREE> 12 KB
P5 of 15KB is loaded

Fragmentation

- As processes are loaded and removed from memory, the free memory space is broken into little pieces.
- It happens after sometimes that processes cannot be allocated to memory blocks considering their small size and memory blocks remains unused.
- This problem is known as Fragmentation.

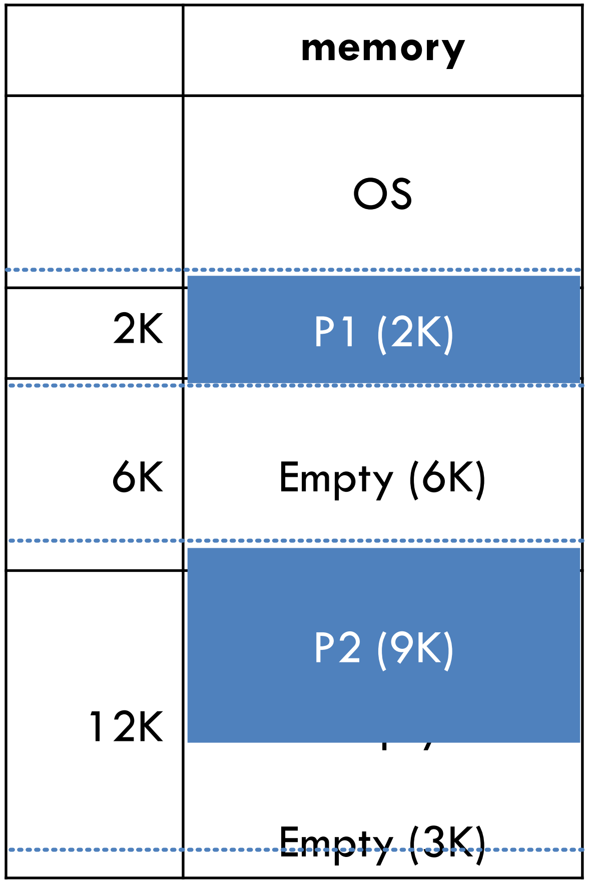
External fragmentation

- Total memory space is enough to satisfy a request or to reside a process in it, but it is not contiguous, so it cannot be used.

Internal fragmentation

- Memory block assigned to process is bigger. Some portion of memory is left unused, as it cannot be used by another process.

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.***

THANNK YOU