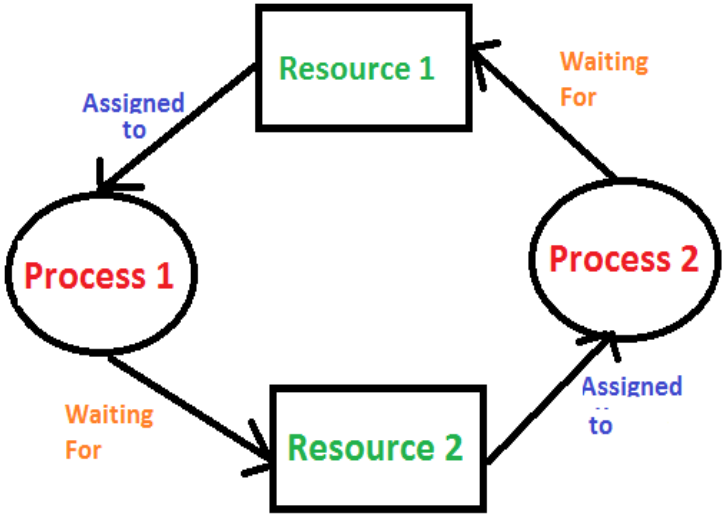
Operating System Concepts

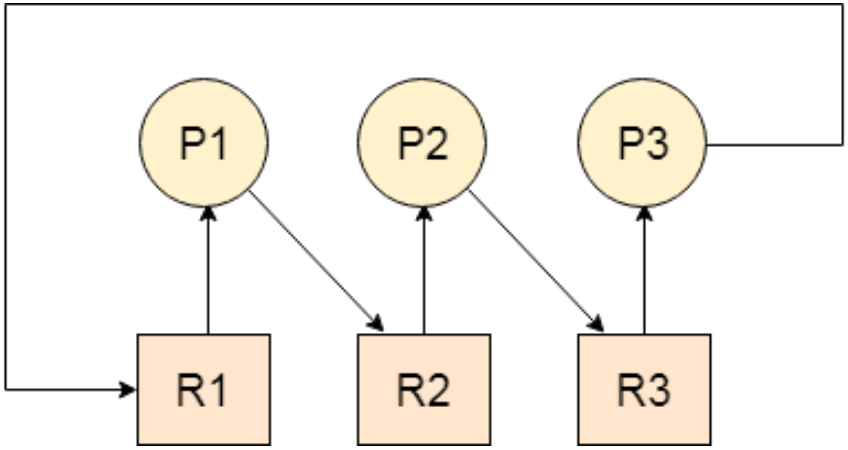
SunBeam Institute of Information & Technology, Hinjwadi, Pune & Karad.

Trainer: Akshita Chanchlani
Email: akshita.chanchlani@sunbeaminfo.com

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.

- control of the resource cannot be taken away forcefully from any process.

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 Prevention
2. Deadlock Detection and Avoidance
3. Deadlock Recovery

Handling Deadlock

1. Deadlock Prevention: deadlock can be prevented by discarding any one condition out of four necessary and sufficient conditions.

2. Deadlock Detection & Avoidance: before allocating resources for processes all input can be given to deadlock detection algorithm in advanced and if there are chances to occur deadlock then it can be avoided by doing necessary changes.

- There are two deadlock detection & avoidance algorithms:

1. Resource Allocation Graph Algorithm

2. Banker's Algorithm

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 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:

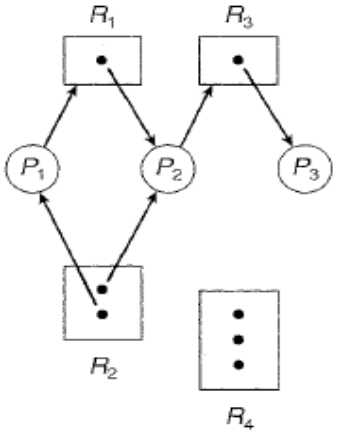
P = { P1, P2, P3 },

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.

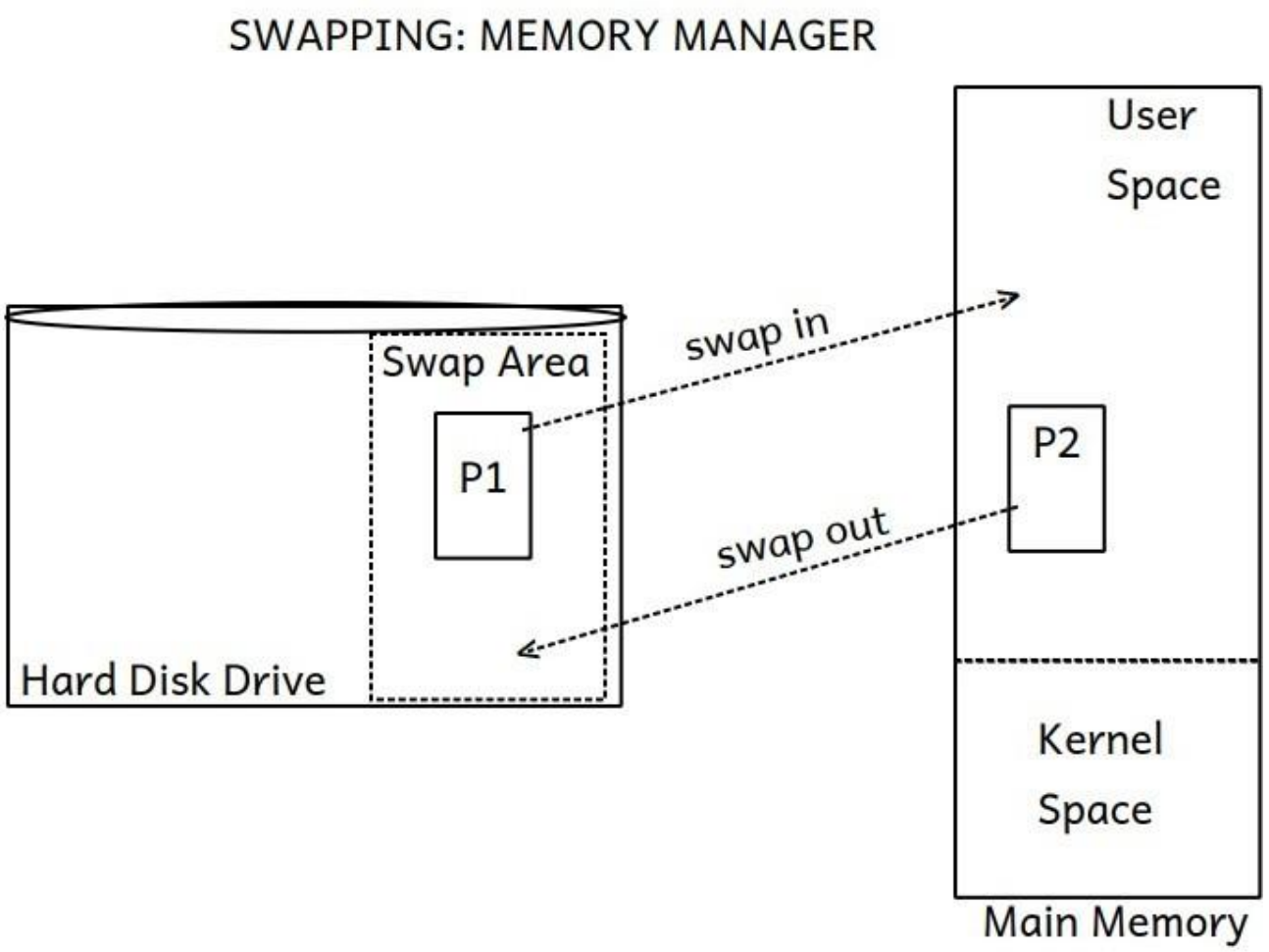
Bankers algorithm

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

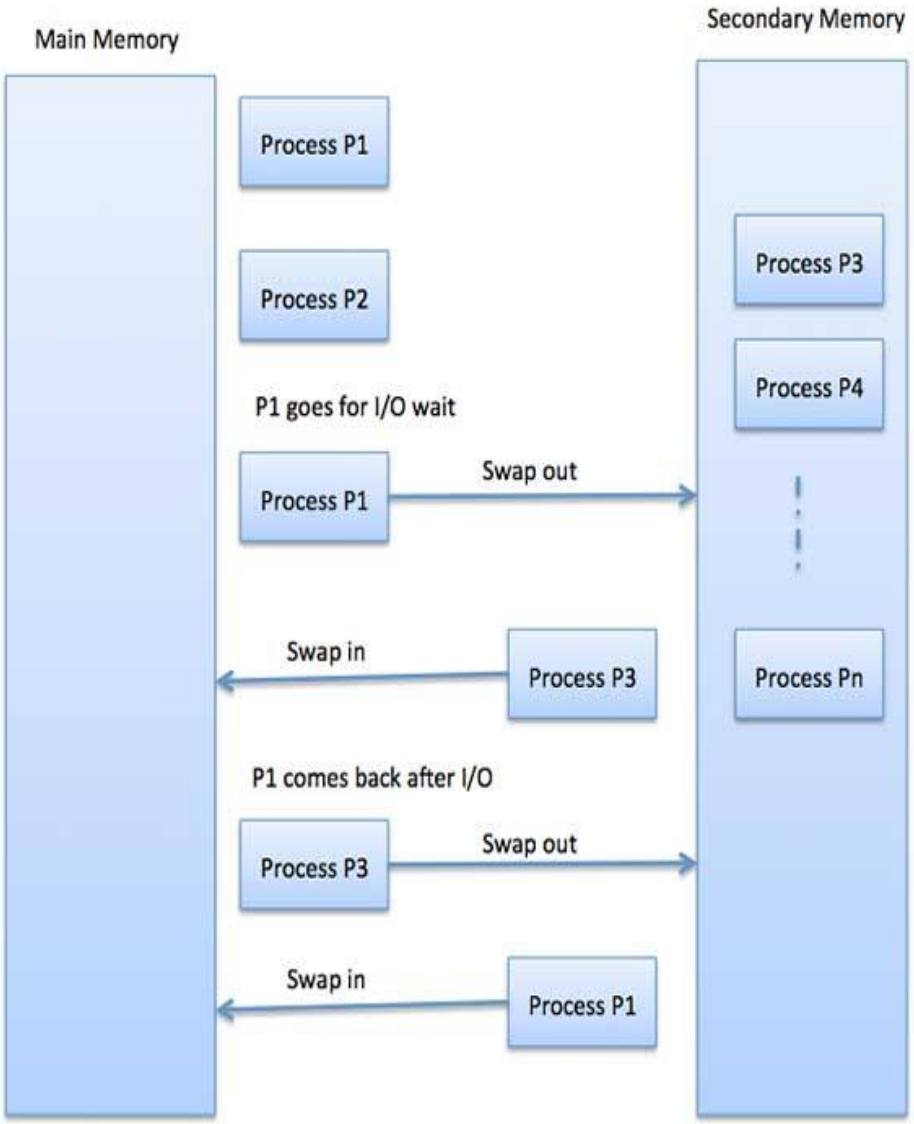
- Is the task carried out by the OS and hardware to accommodate multiple processes in main memory
- If only a few processes can be kept in main memory, then much of the time all processes will be waiting for I/O and the CPU will be idle
- Hence, memory needs to be allocated efficiently in order to pack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

Swapping Memory Manager



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 mai 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**.



Swapping

- **Swap area:** it is a portion of the **hard disk drive** (keep reserved while installation of an OS) can be used by an OS as an extension of the main memory in which inactive running programs can be kept temporarily and as per request processes can be swapped in and swapped out between swap area and the main memory.
- In Linux swap area can be maintained in the form of **swap partition,** whereas in Windows swap area can be maintained in the form of **swap files.**
- **Conventionally size of the swap area should be doubles the size of the main memory,** i.e. if the size of main memory is 2 GB then size of swap area should be 4 GB, if the size of main memory is 4 GB then size of swap area should be 8 GB and so on.
- Swapping done by the system program of an OS named as **Memory Manager**, it swap ins active running programs into the main memory from swap area and swap outs inactive running programs from the main memory and keep them temporarily into the swap area.

there are two variants of **swapping: swap in & swap out.**

Swapping

Addresses generated by compiler (i.e. compiler + linker) are reffered as **logical addresses.**

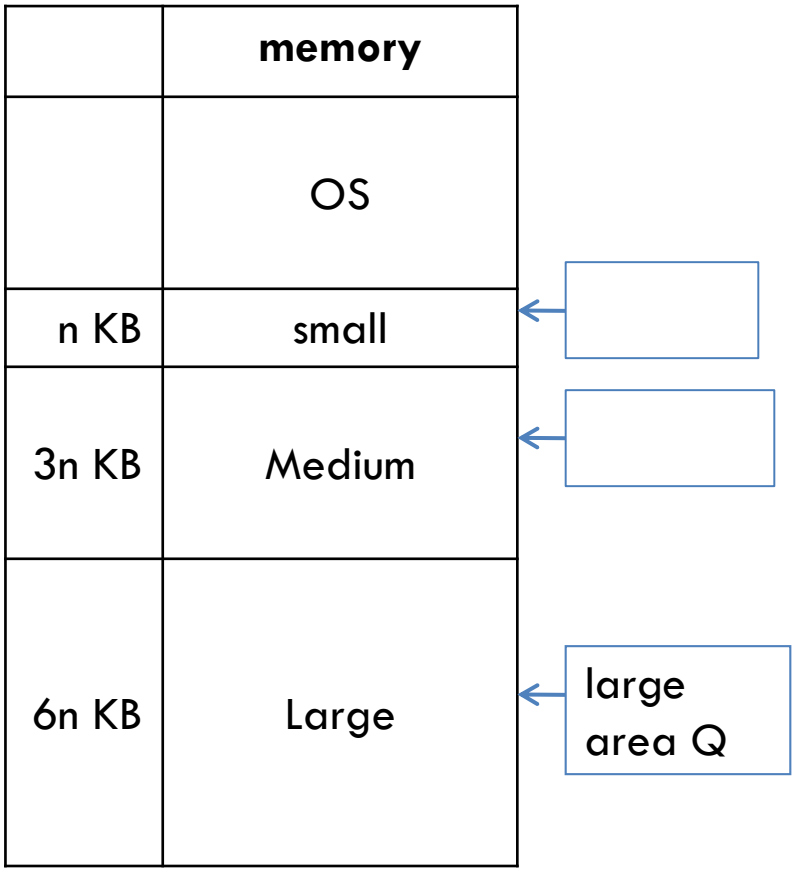
Addresses which can be see by the process when it is in the main memory reffered as **physical addresses.**

-**MMU (Memory Management Unit):** wich is a hardware unit **converts logical address into physical address.**

-MMU is a hardware contains **adder circuit, comparator circuit, base register and limit register.** Values of base register and limit registers gets change during context-switch, and memory space of one process gets protected from another process.

CPU always executes program in its logical memory space.

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.

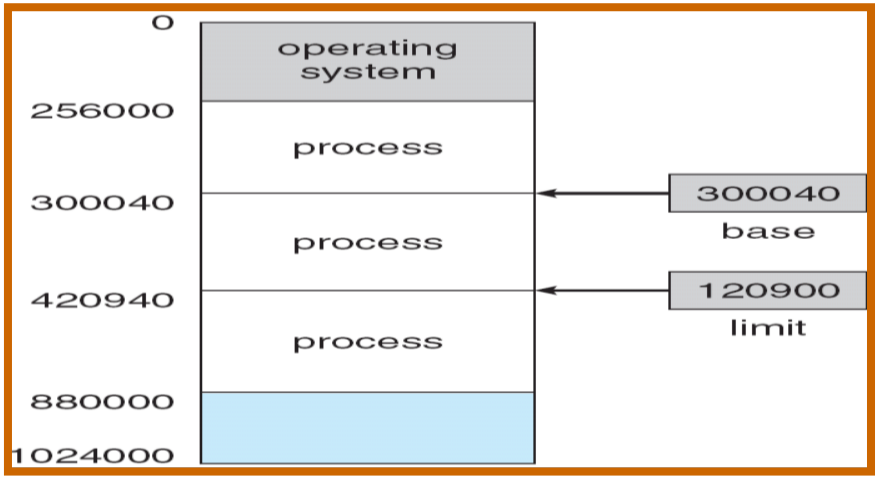
- The main problem with the fixed partitioning method is how to determine the number of partitions, and how to determine their sizes.

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



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

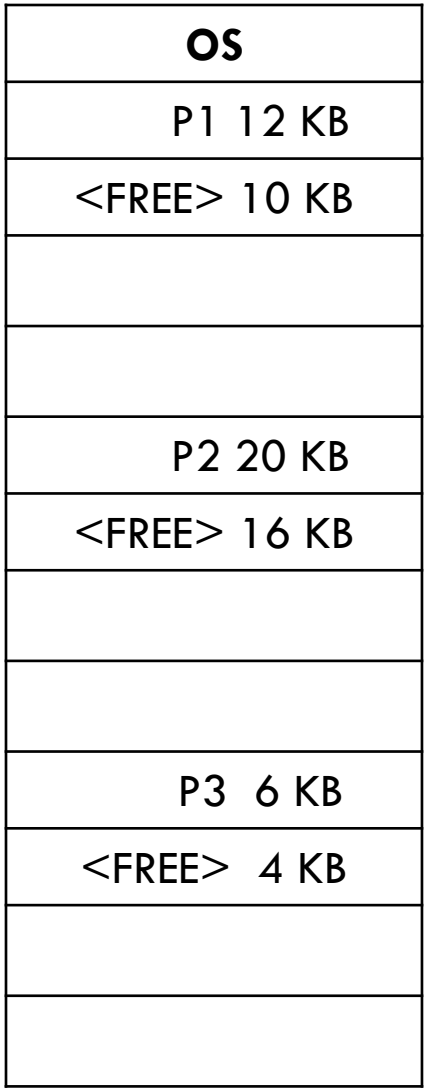
First-fit and best-fit better than worst-fit in terms of speed and storage utilization

first fit

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

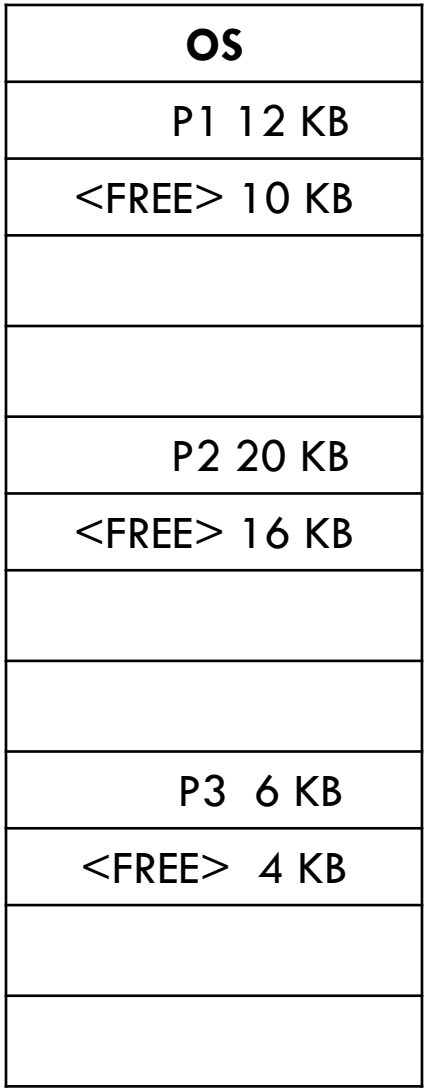
first fit

Initial memory mapping

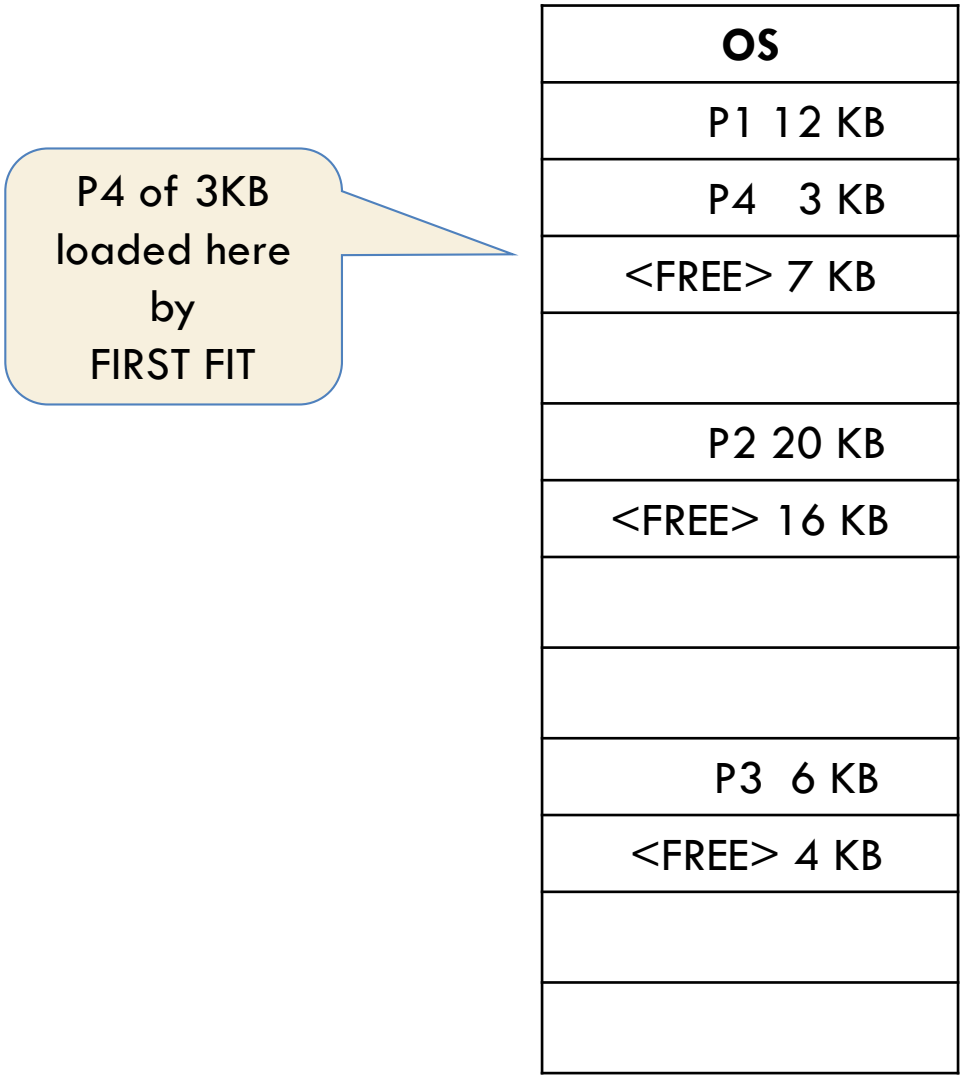


first fit

P4 of 3KB arrives

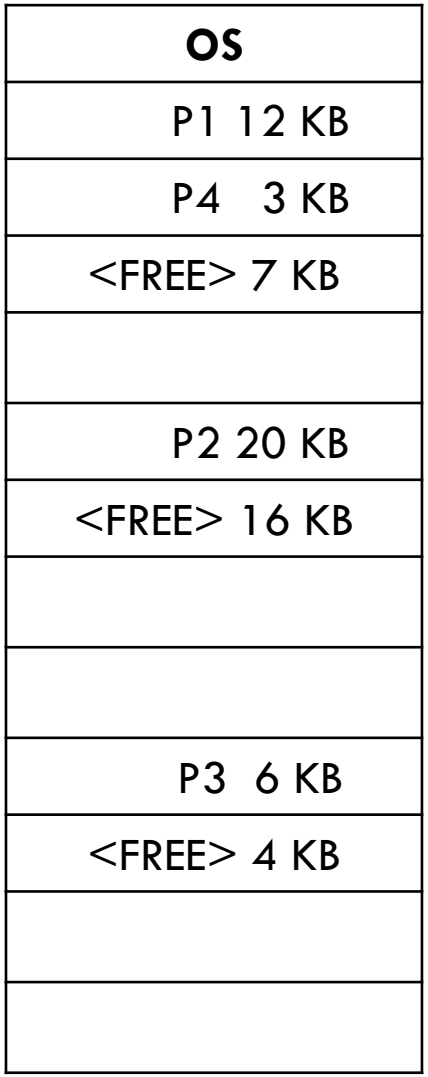


first fit

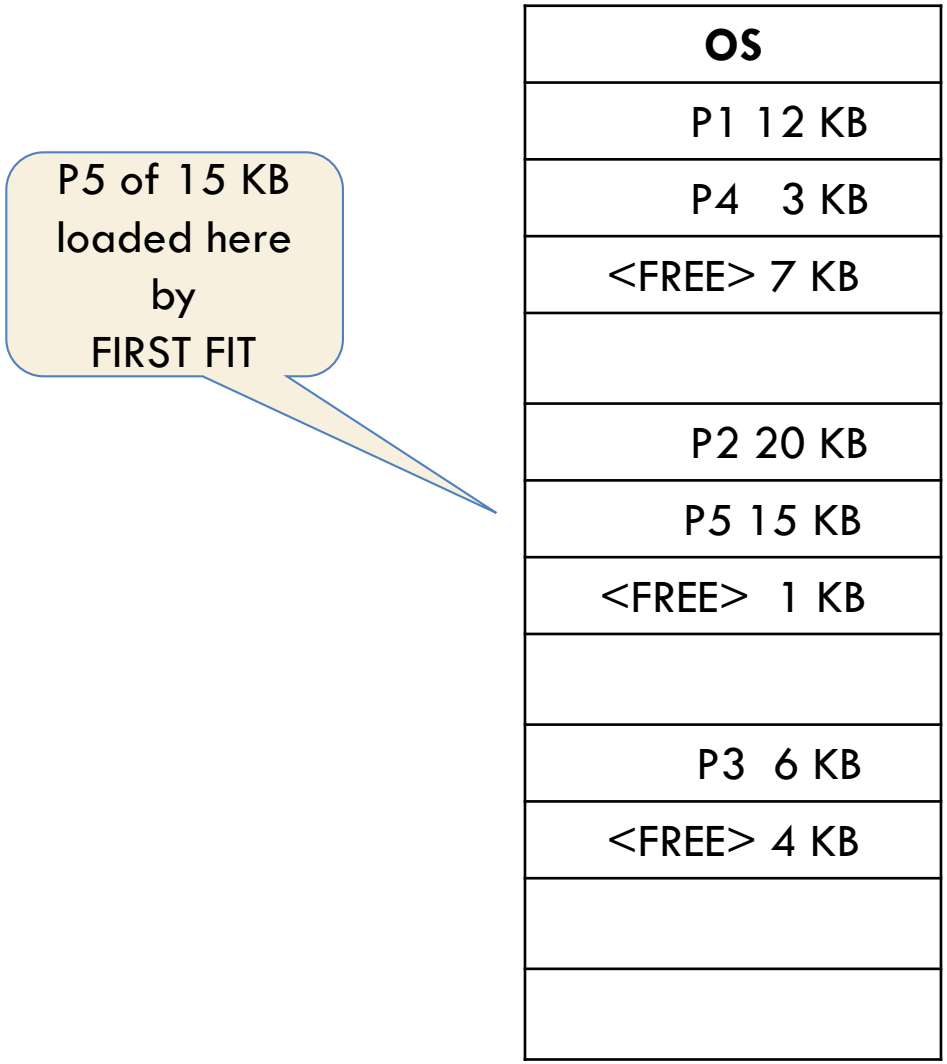

P4 of 3KB loaded here by FIRST FIT
OS
P1 12 KB
P4 3 KB
<FREE> 7 KB
P2 20 KB
<FREE> 16 KB
P3 6 KB
<FREE> 4 KB

first fit

P5 of 15KB arrives



first fit

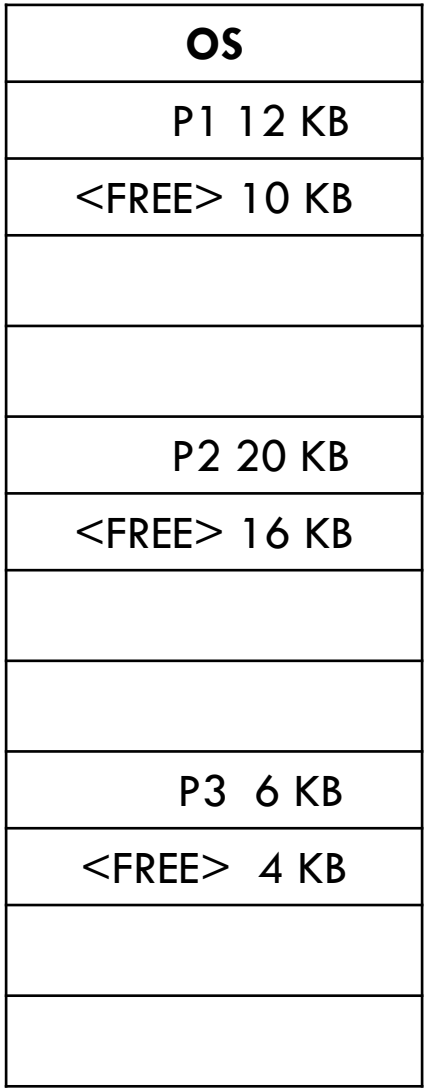

P5 of 15 KB
loaded here
by
FIRST FIT
OS
P1 12 KB
P4 3 KB
<FREE> 7 KB
P2 20 KB
P5 15 KB
<FREE> 1 KB
P3 6 KB
<FREE> 4 KB

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.

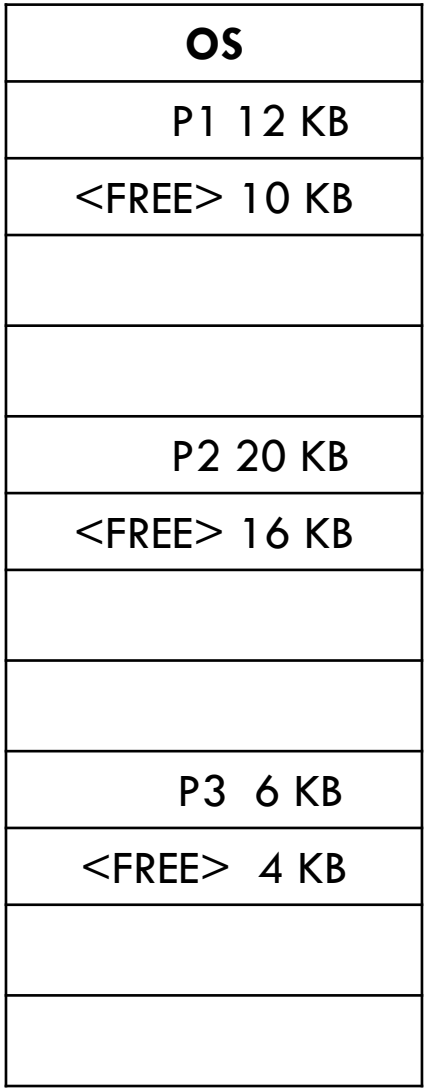
best fit

Initial memory mapping

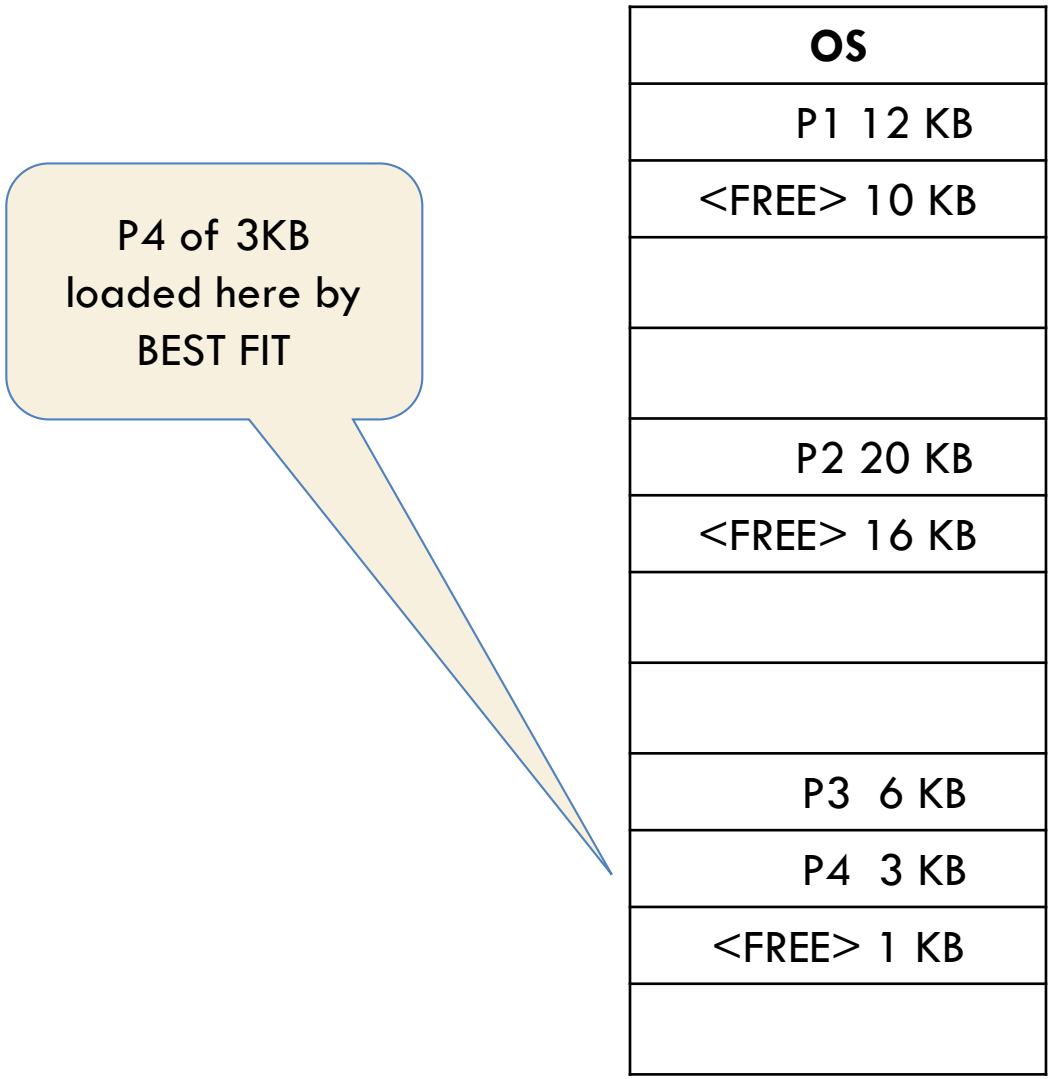


best fit

P4 of 3KB arrives

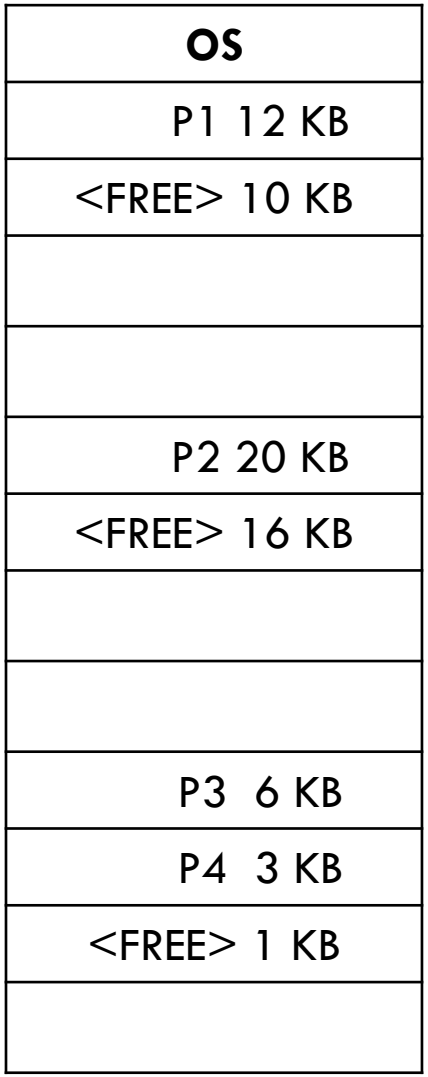


best fit

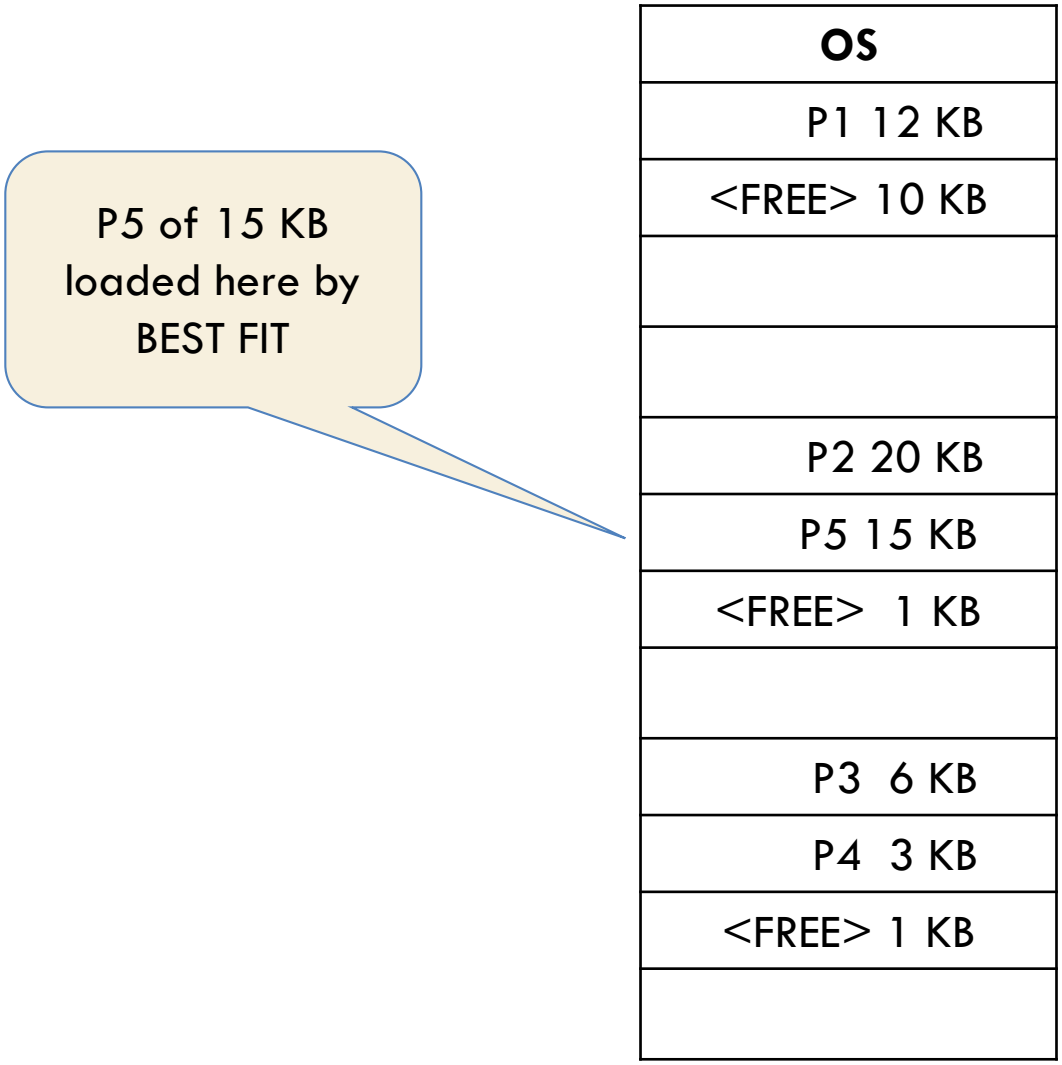

P4 of 3KB
loaded here by
BEST FIT
OS
P1 12 KB
<FREE> 10 KB
P2 20 KB
<FREE> 16 KB
P3 6 KB
P4 3 KB
<FREE> 1 KB

best fit

P5 of 15KB arrives



best fit

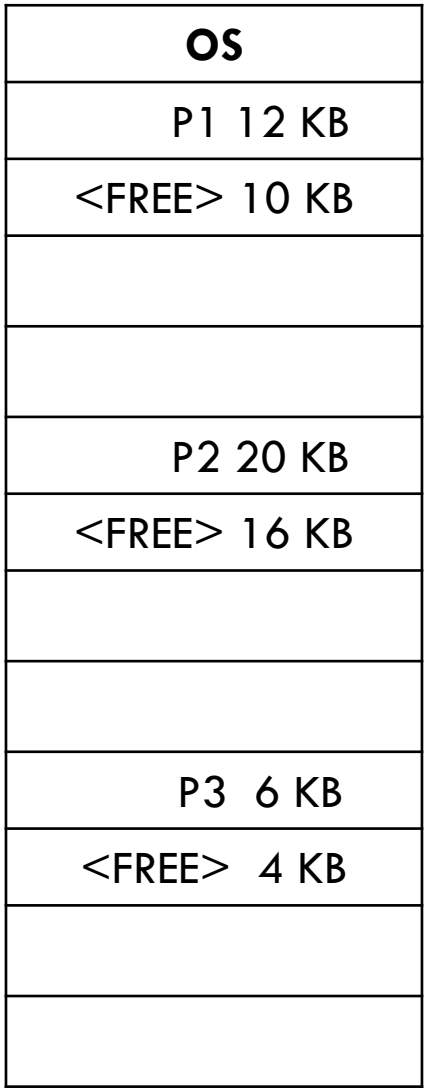

P5 of 15 KB loaded here by BEST FIT
OS
P1 12 KB
<FREE> 10 KB
P2 20 KB
P5 15 KB
<FREE> 1 KB
P3 6 KB
P4 3 KB
<FREE> 1 KB

worst fit

- 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.

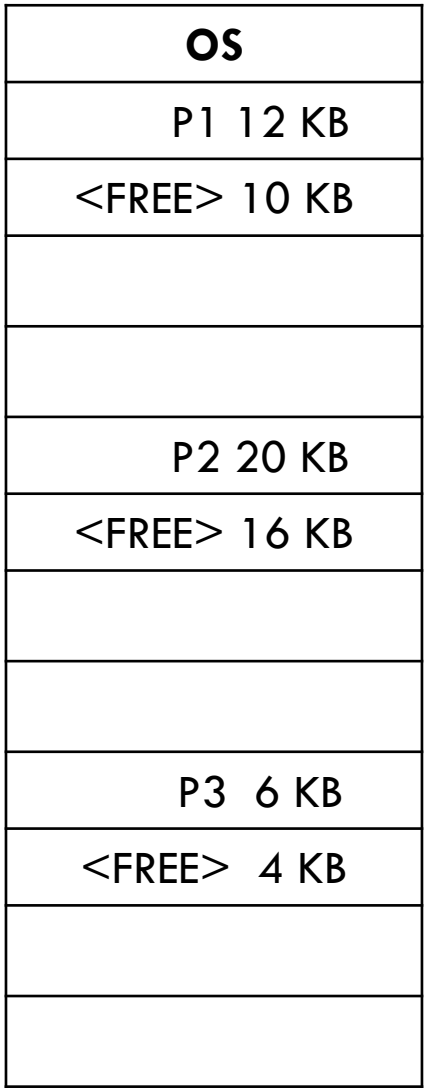
worst fit

Initial memory mapping



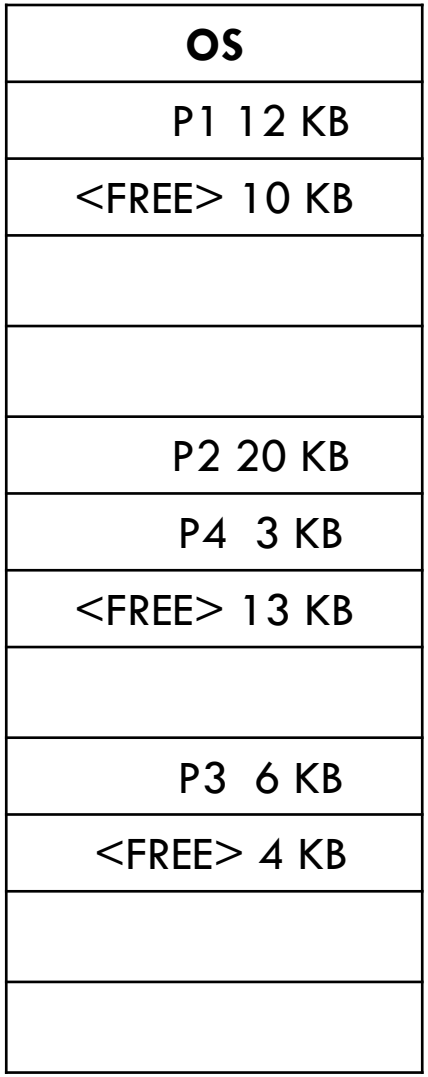
worst fit

P4 of 3KB arrives



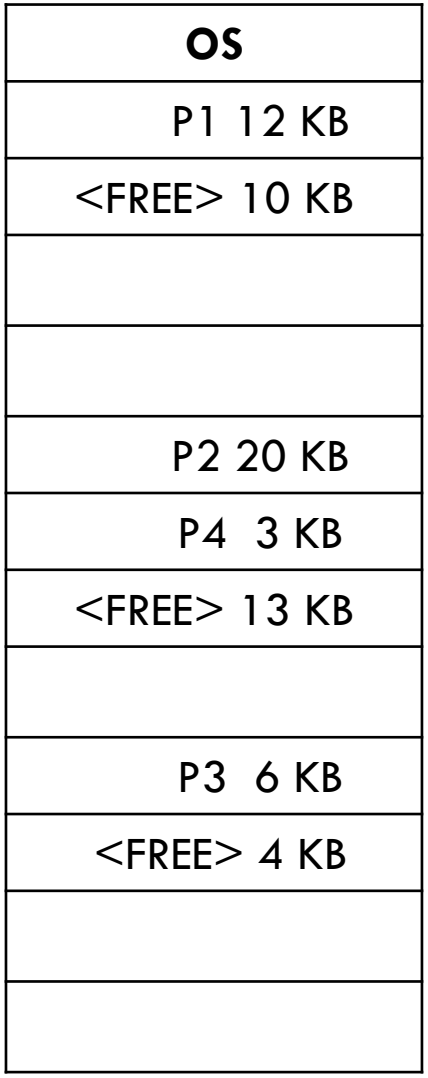
worst fit

P4 of 3KB Loaded here by WORST FIT



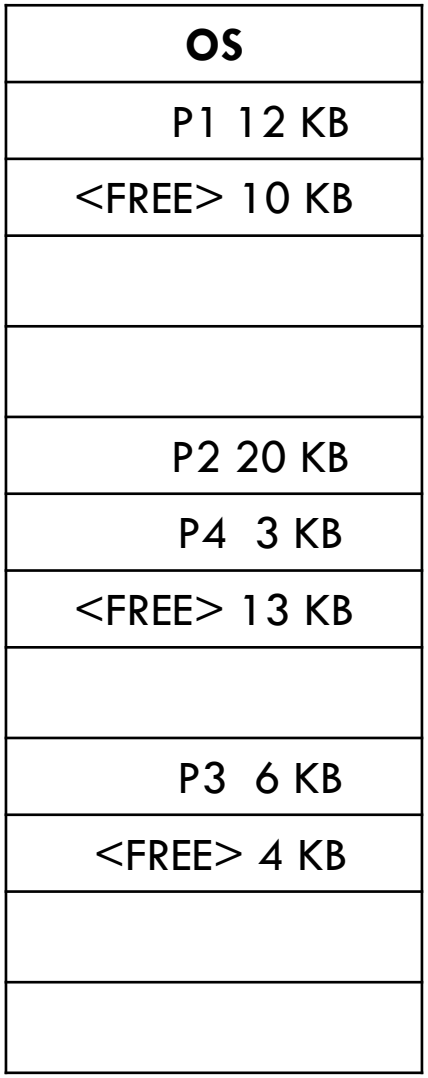
worst fit

No place to load
P5 of 15K



worst fit

No place to load P5 of 15K



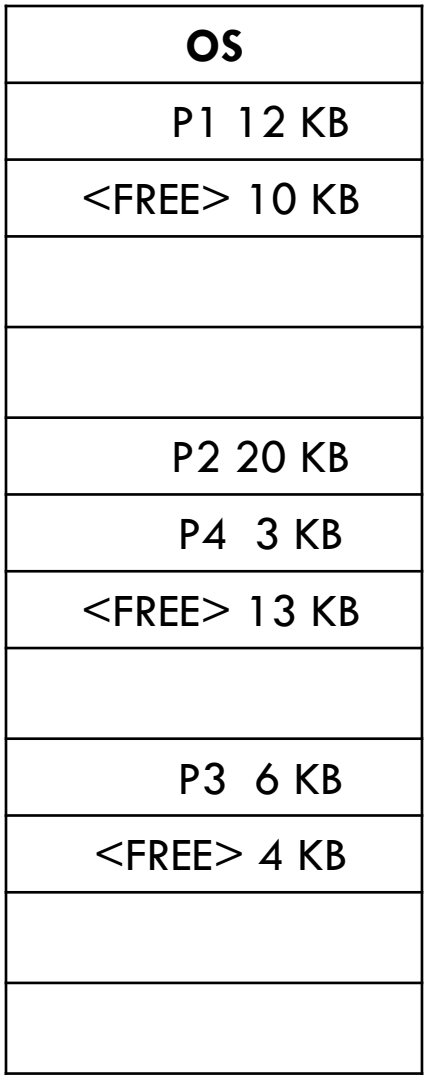
Compaction is needed !!

compaction

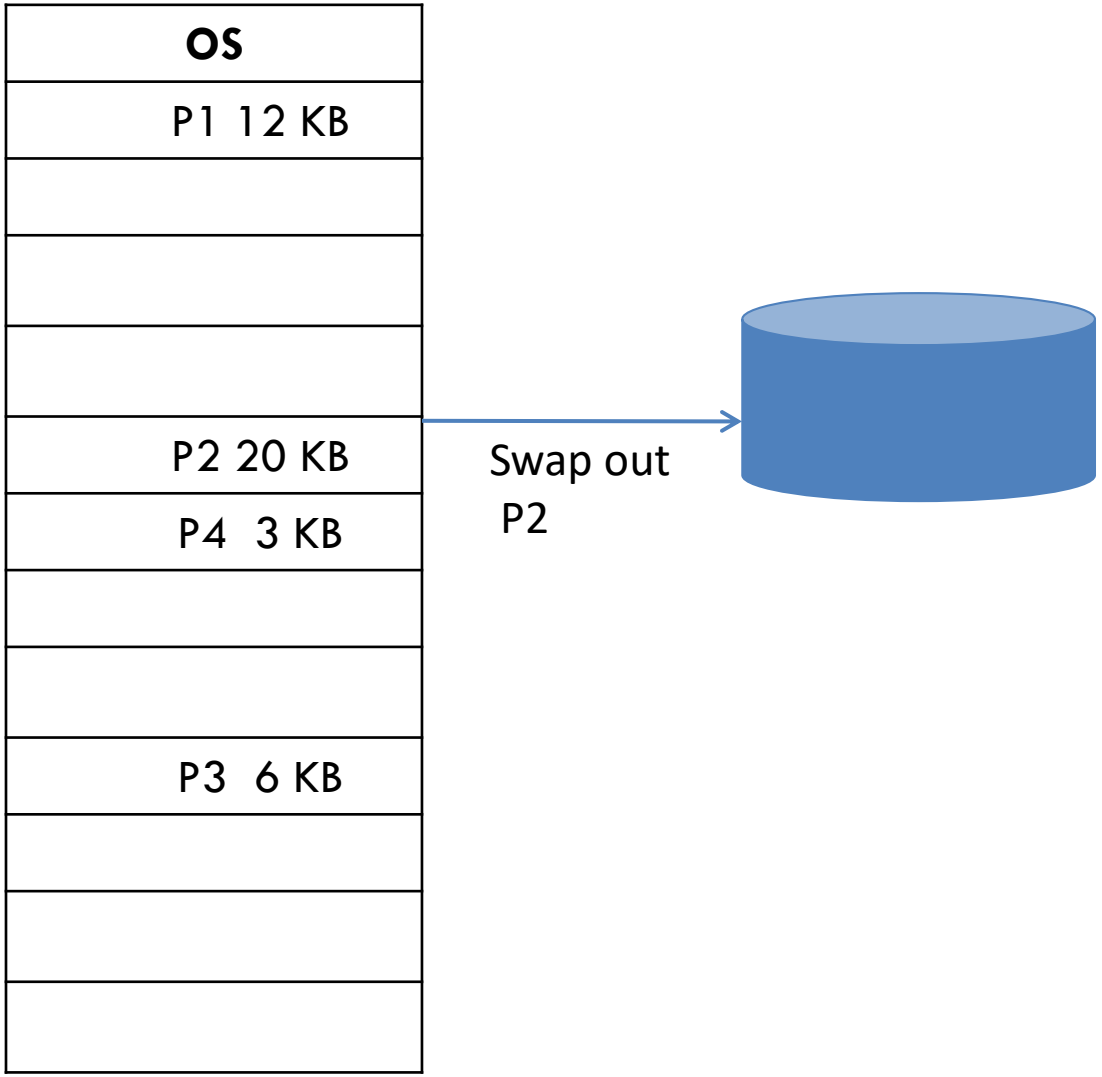
- Compaction is a method to overcome the external fragmentation problem.
- All free blocks are brought together as one large block of free space.
- Compaction requires dynamic relocation.
- One method for compaction is swapping out those processes that are to be moved within the memory, and swapping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

compaction

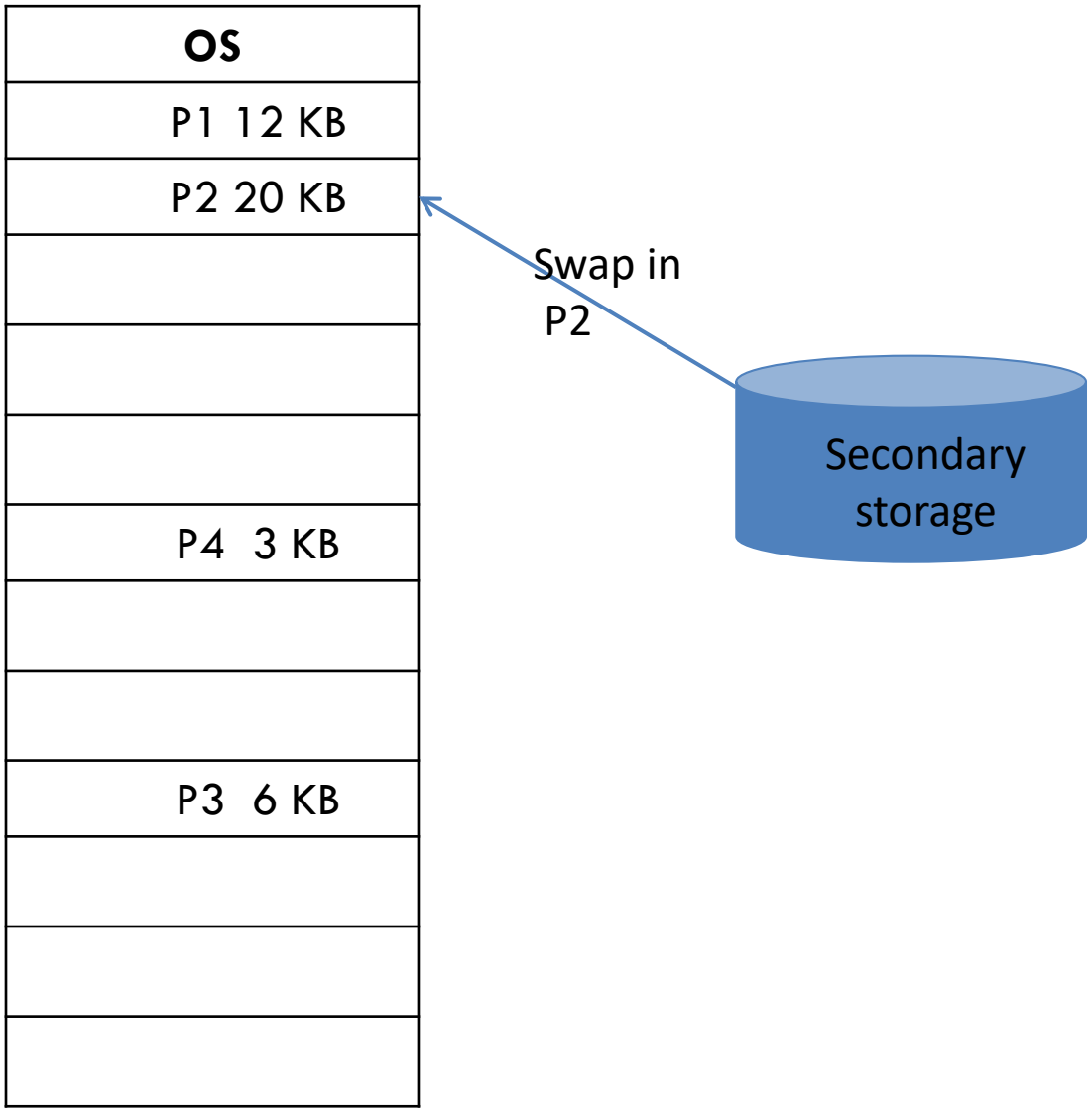
Memory mapping before compaction



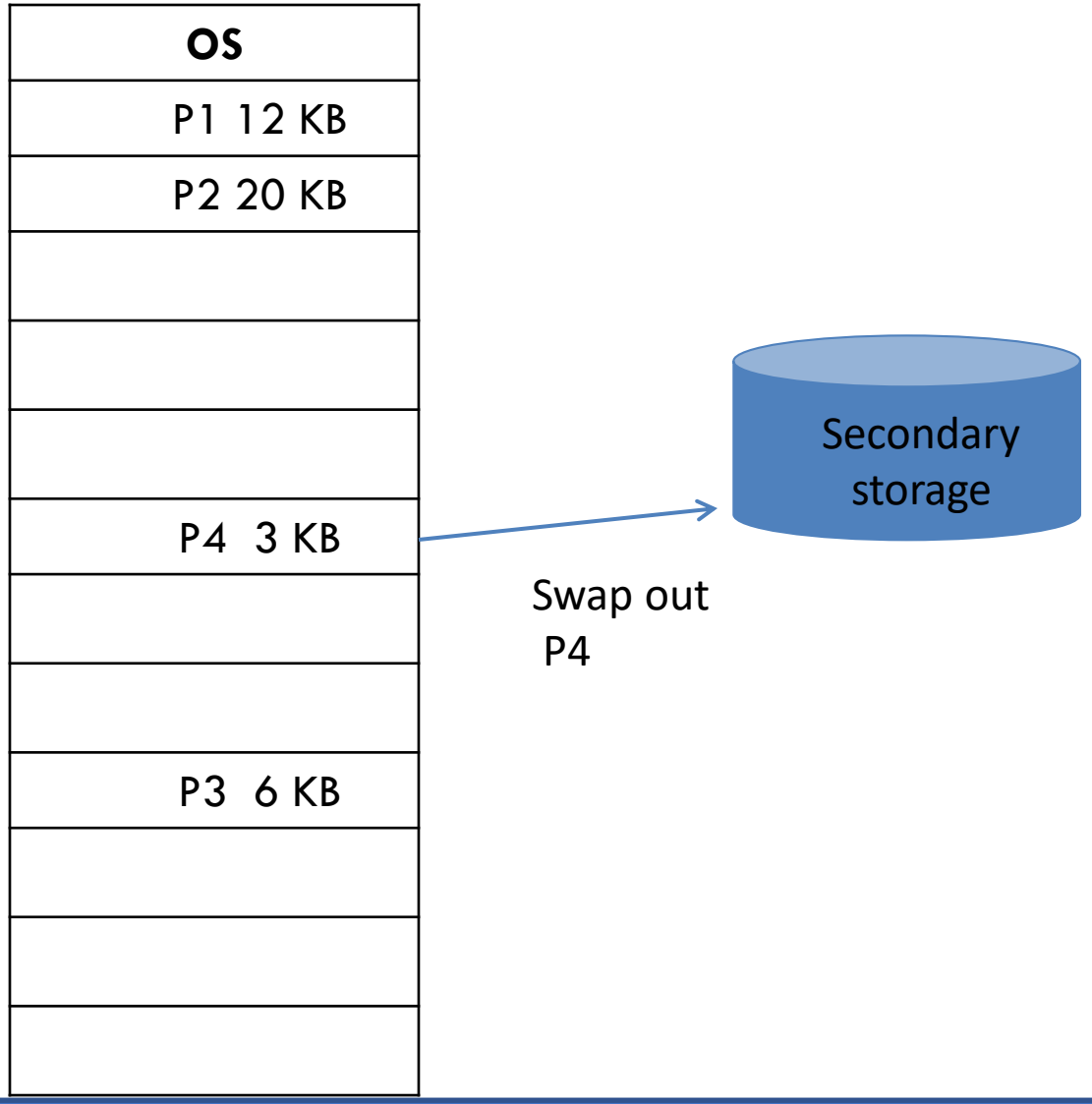
compaction



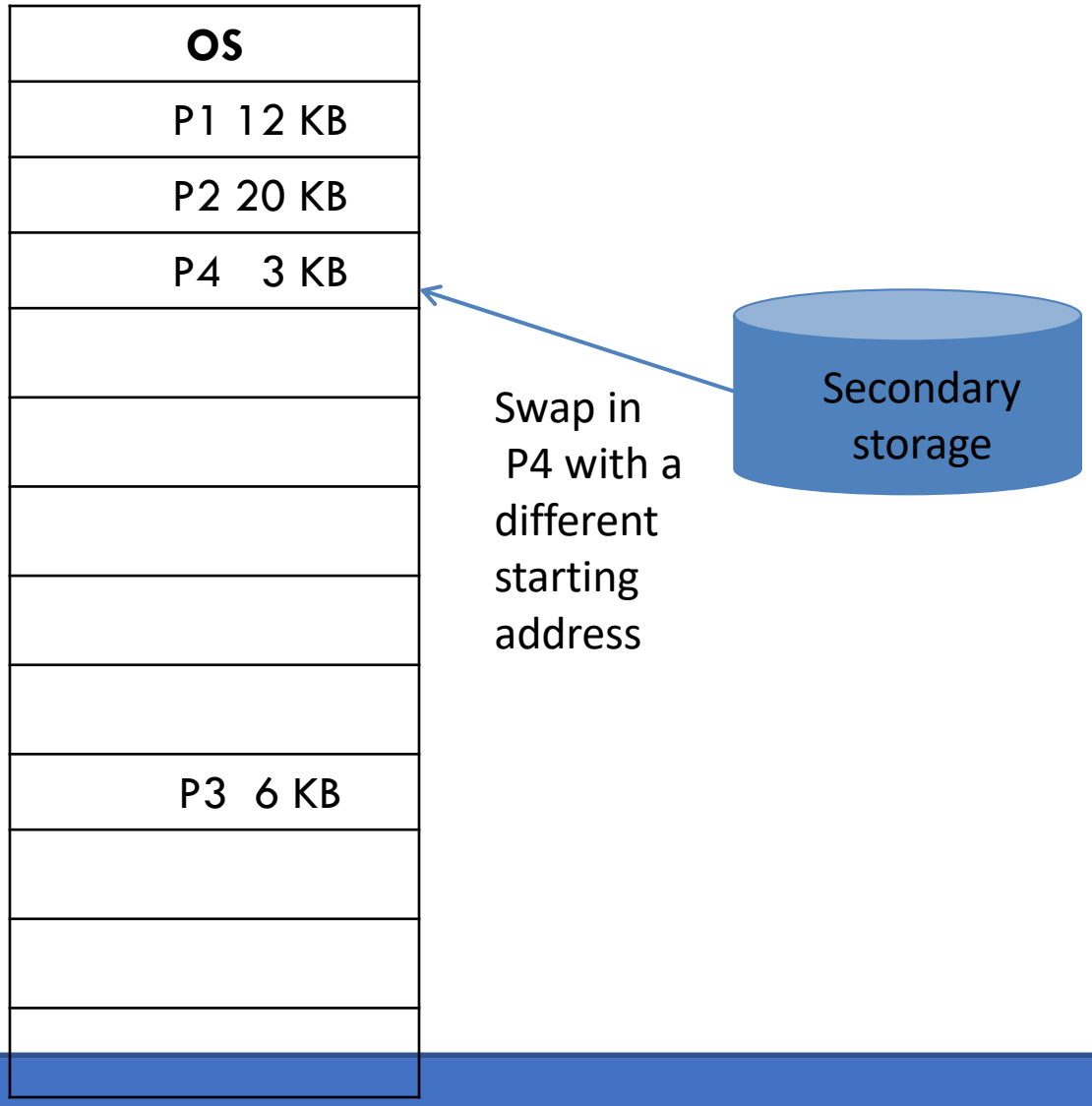
compaction



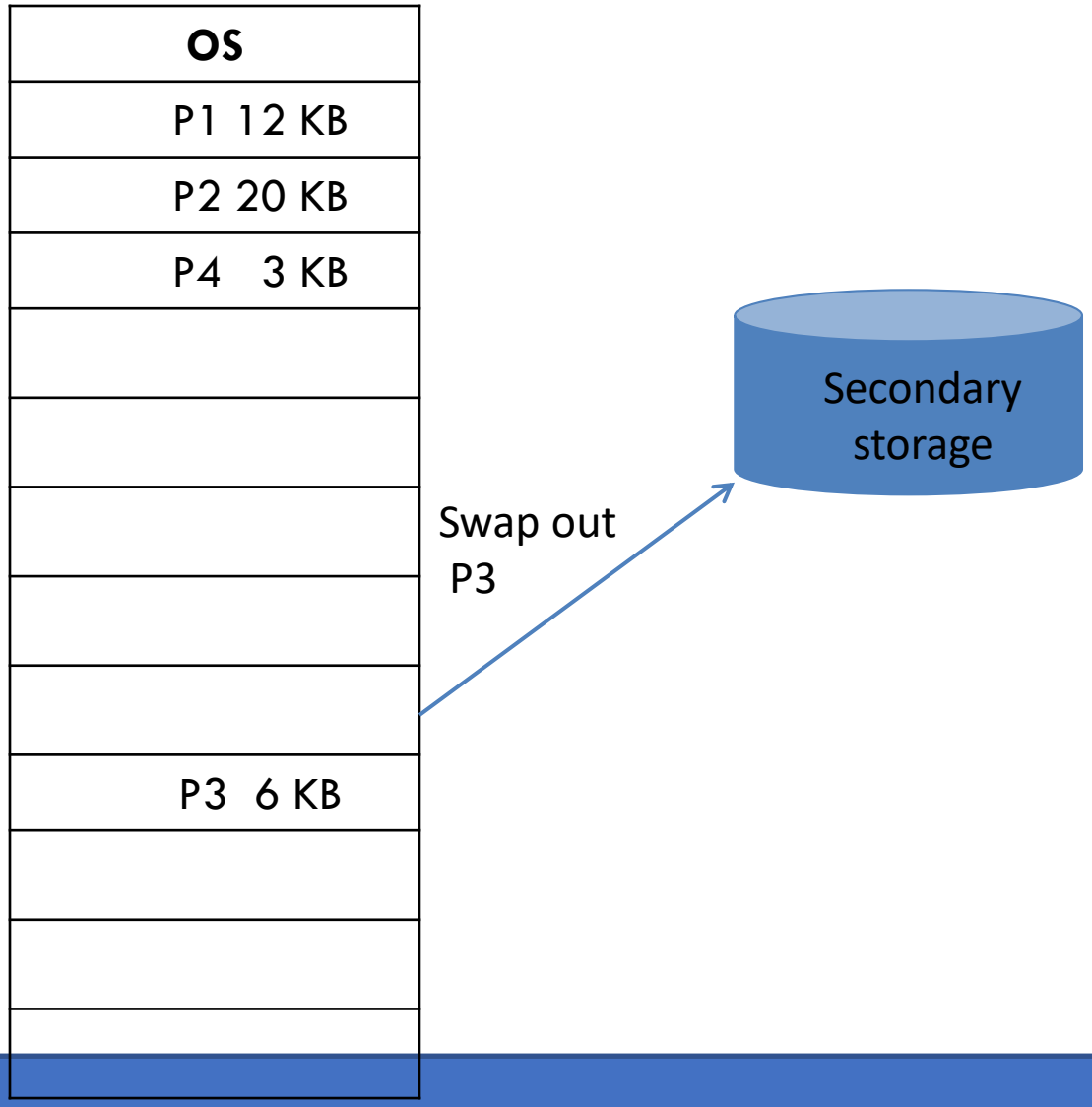
compaction



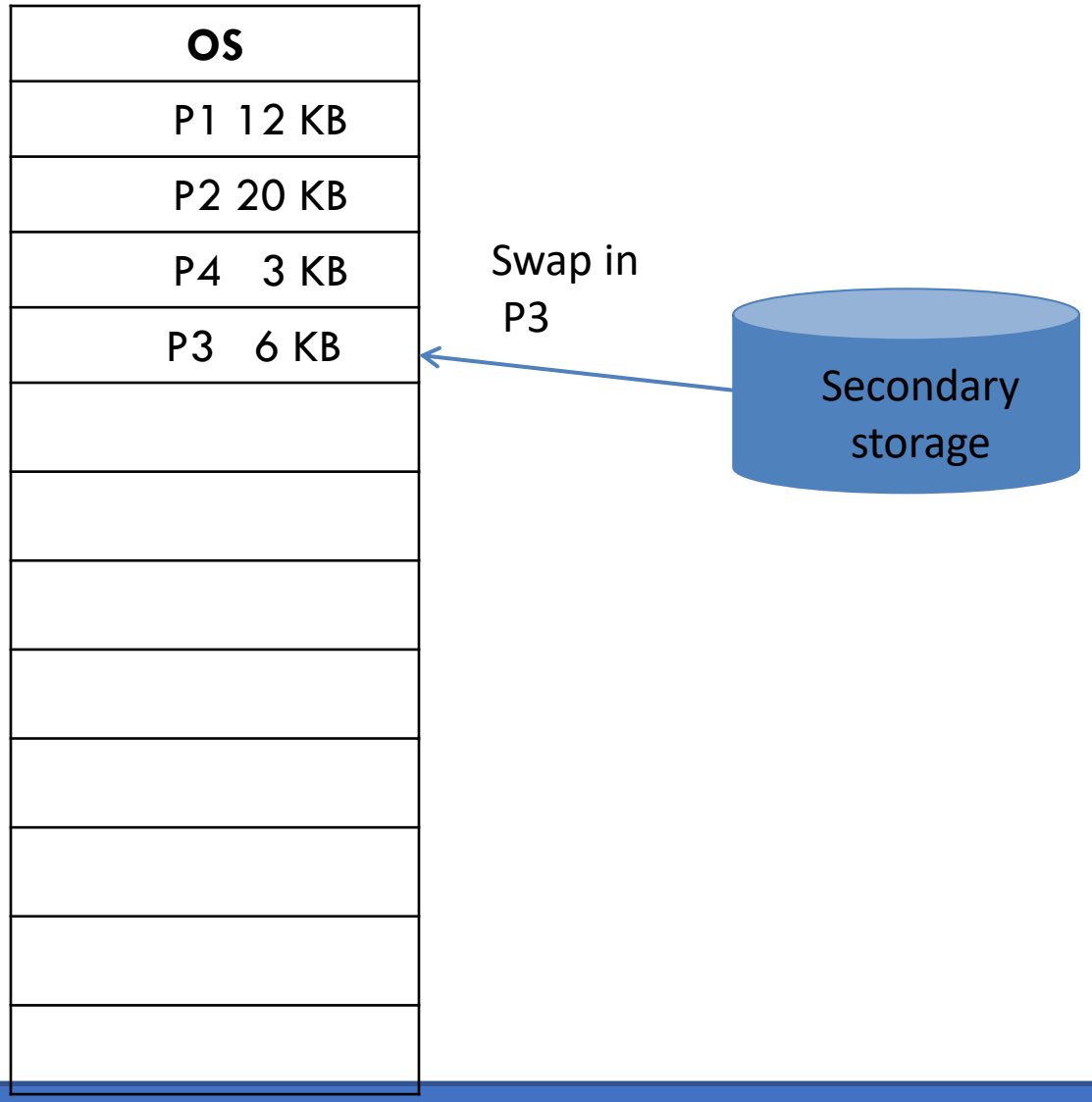
compaction


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

compaction

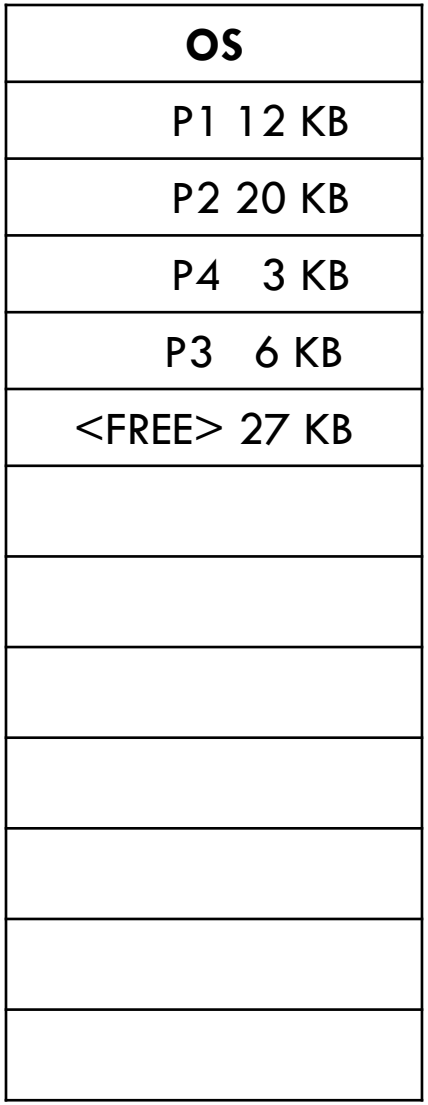


compaction



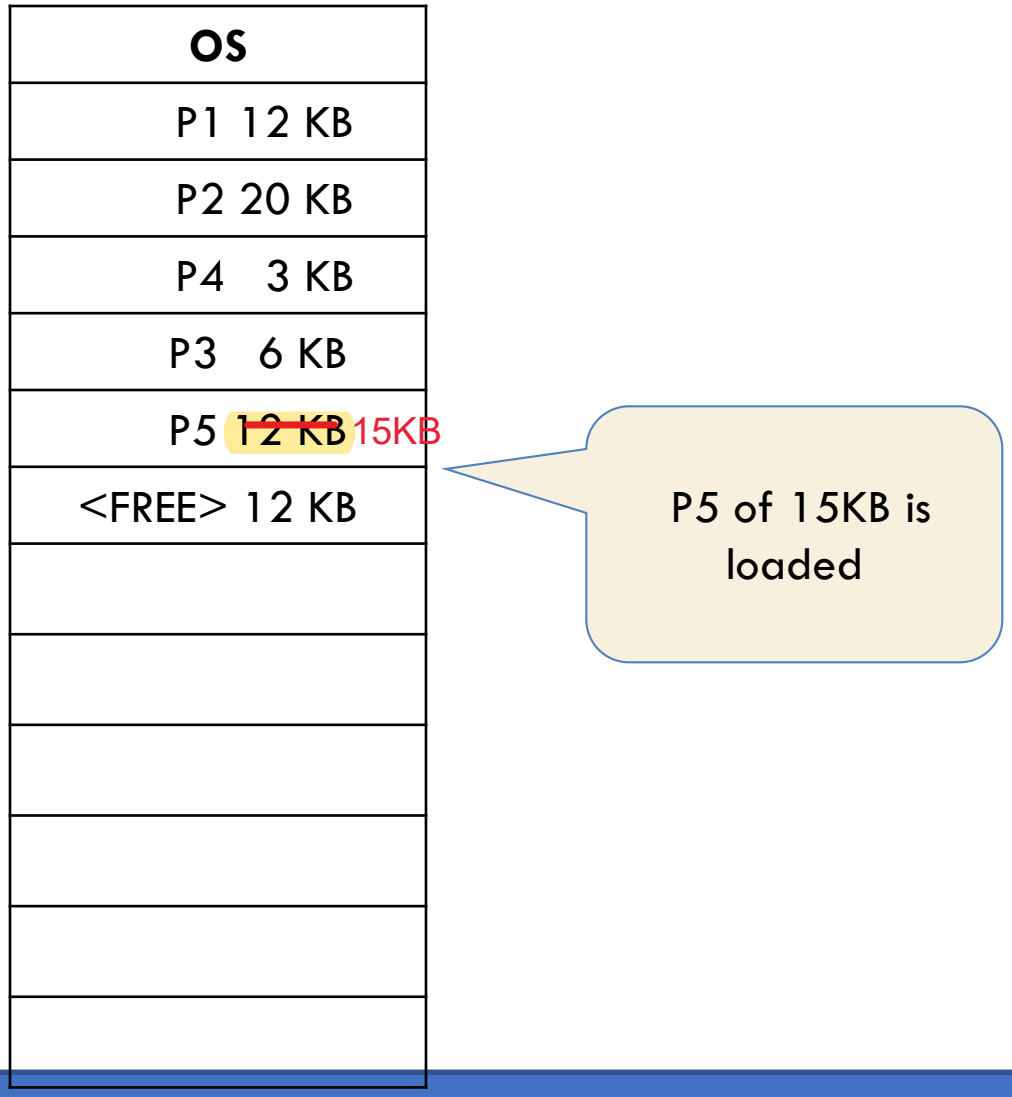
compaction

Memory mapping after compaction



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 15KB
<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.

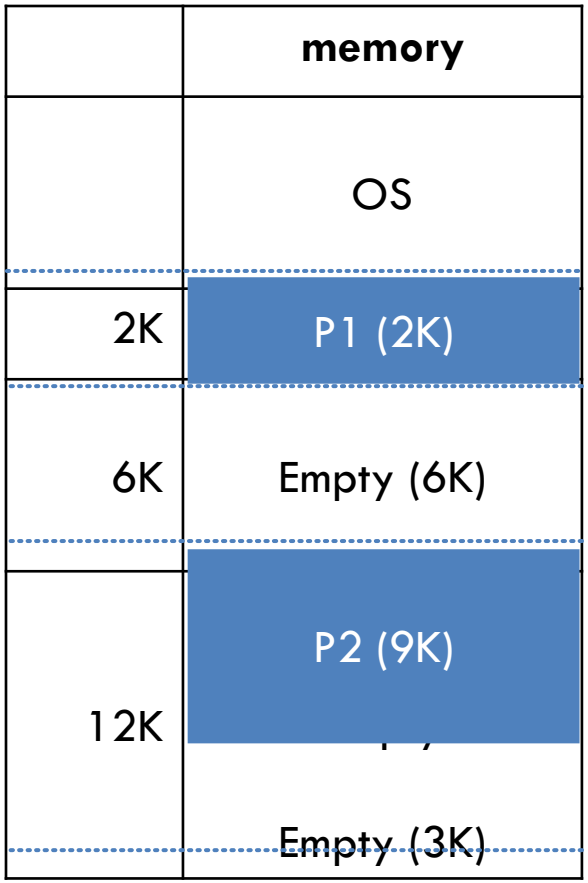
External fragmentation

- 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.***

Memory Allocation Types

Contiguous Memory Allocation **- One process – One piece of memory.**

- **Single** Partition Allocation.
- **Multiple** Partition Allocation
  - **Fixed** size partitioning (equal size, unequal size)
  - **Dynamic** Partitioning (First-Fit, Best-Fit , Worst-fit)
    - Problem (Internal and External Fragmentation)

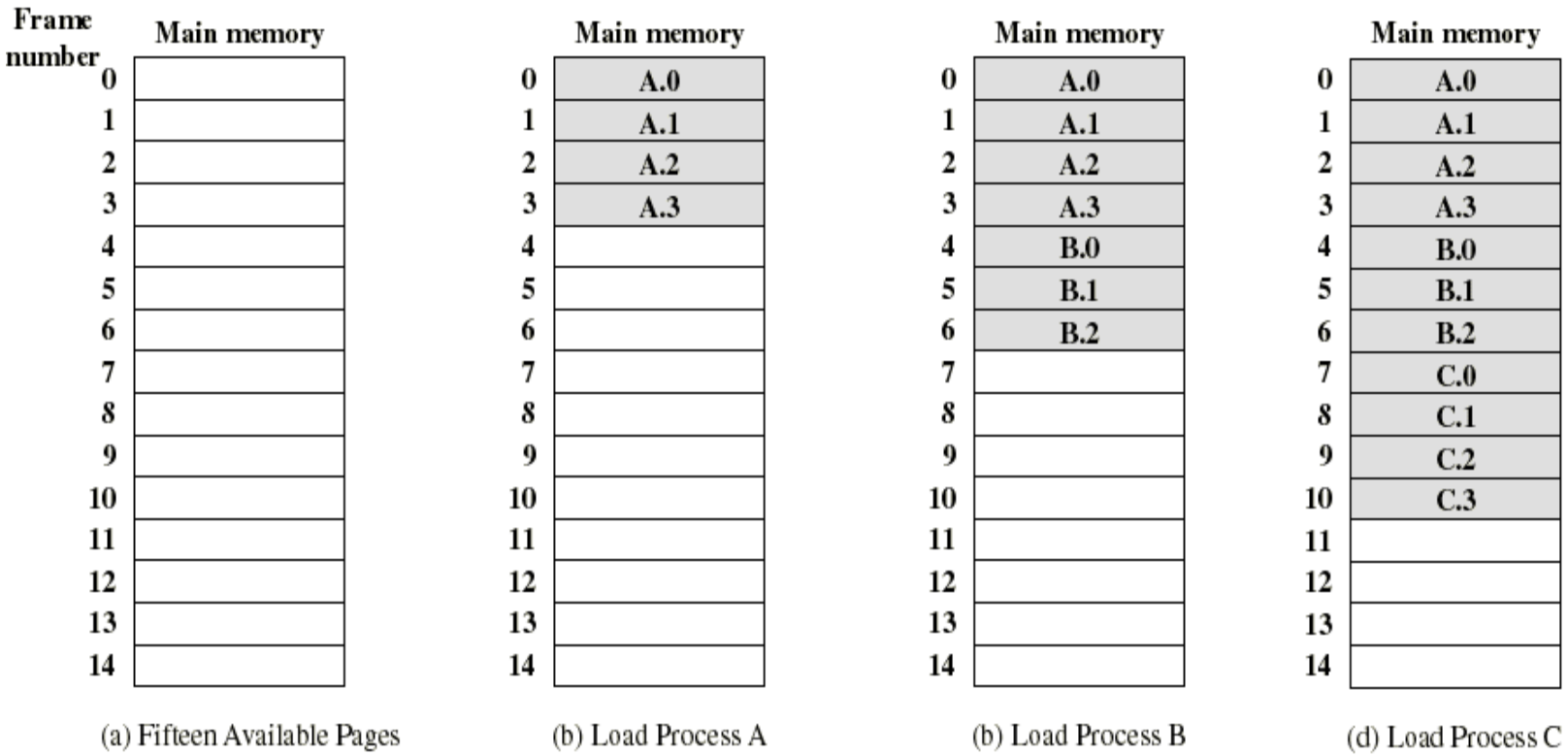
Non Contiguous Memory Allocation - **One process – Many pieces of memory.**

- **Paging**
- **Segmentation**

Paging

- Divide physical memory into fixed-sized blocks called **frames** (size is power of 2, between 512 bytes and 8,192 bytes)
- Divide logical memory into blocks of same size called **pages**
- Keep track of all free frames
- To run a program of size ***n*** pages, need to find *n* free frames and load program
- Set up a page table to translate logical to physical addresses

Example of process loading



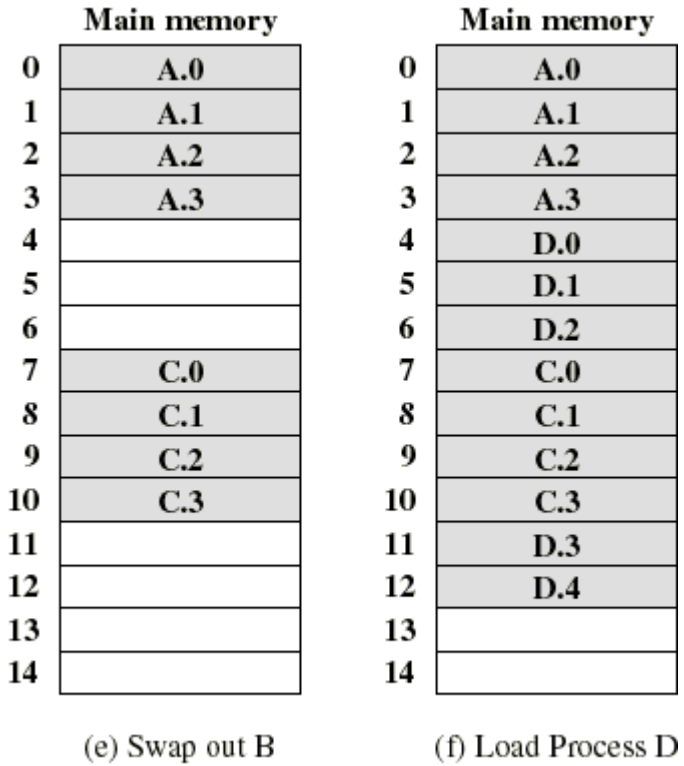
Now suppose that process B is swapped out

Example of process loading (cont.)

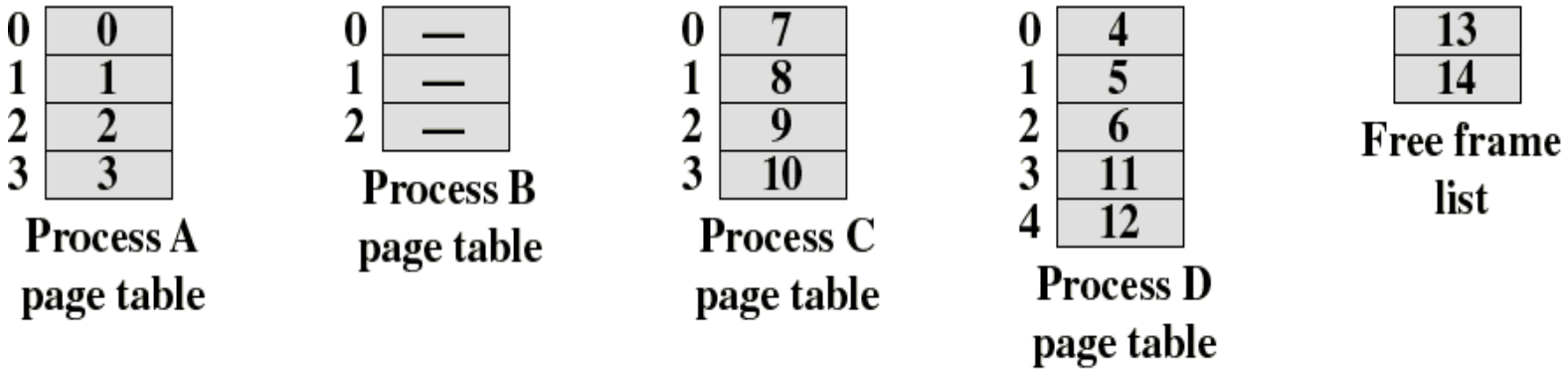
When process A and C are blocked, the pager loads a new process D consisting of 5 pages

Process D does not occupied a contiguous portion of memory

There is no external fragmentation

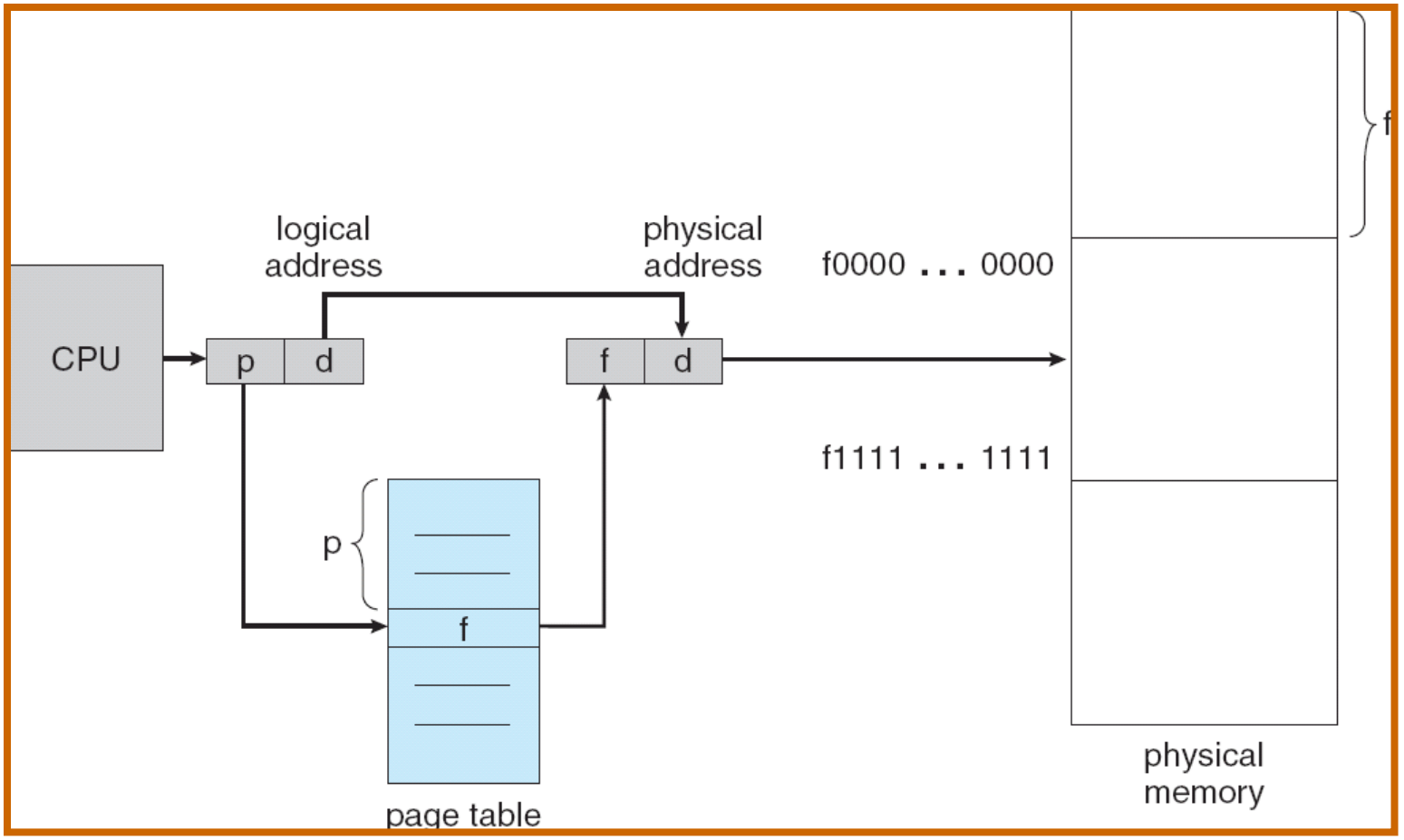


Page Tables



- The OS now needs to maintain (in main memory) a **page table** for each process
- Each entry of a page table consist of the frame number where the corresponding page is physically located
- The page table is indexed by the page number to obtain the frame number
- A free frame list, available for pages, is maintained

Paging Hardware



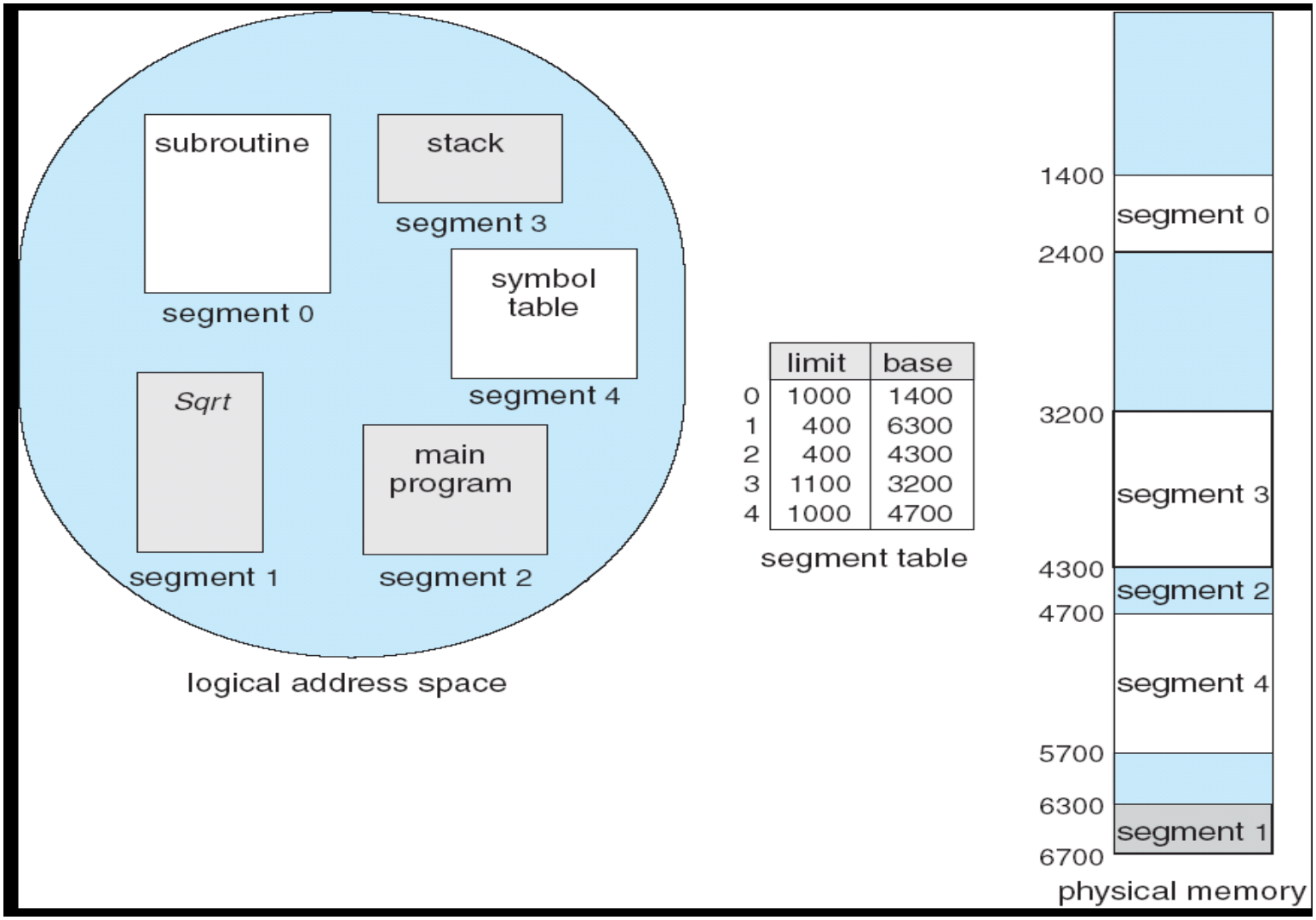
Segmentation

- Each program is subdivided into blocks of non-equal size called segments
- ***Segment***: a region of logically contiguous memory
- ***Segmentation-based transition***: use a table of base-and-bound pairs
- When a process gets loaded into main memory, its different segments can be located anywhere.
- Each segment is fully packed with instructs/data: no internal fragmentation
- There is external fragmentation; it is reduced when using small segments

Segmentation

- In contrast with paging, segmentation is visible to the programmer
  - –provided as a convenience to organize logically programs (ex: data in one segment, code in another segment)
  - –must be aware of segment size limit
- The OS maintains a **segment table** for each process. Each entry contains:
  - – the starting physical addresses of that segment.
  - –the length of that segment (for protection)

Segmentation Example



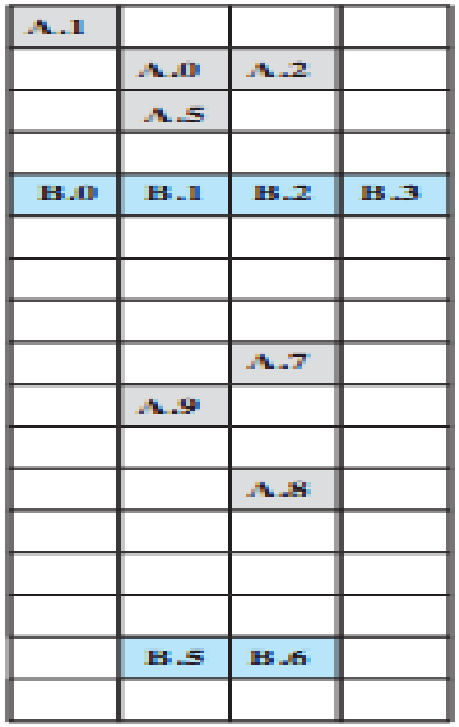
Virtual memory

- **Virtual memory** – separation of user logical memory from physical memory:
  - –Only part of the program needs to be in memory for execution.
  - –Logical address space can therefore be much larger than physical address space.
  - –Allows address spaces to be shared by several processes.
  - –Allows for more efficient process creation.
  - –More programs running concurrently.
  - –Less I/O needed to load or swap processes.
- Virtual memory gives the programmer the impression that he/she is dealing with a huge main memory (relying on available disk space). The OS loads automatically and on-demand pages of the running process.

- A process image may be larger than the entire main memory.
- The required pages need to be loaded into memory whenever required.

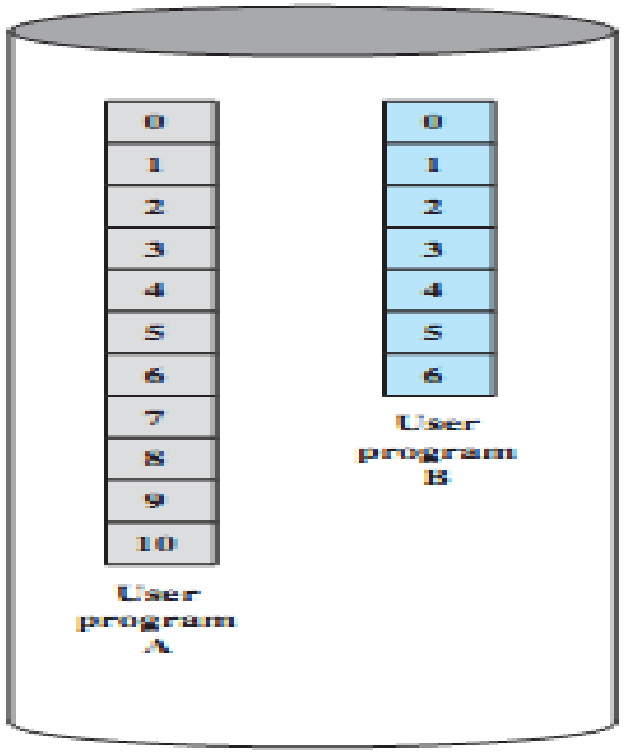
Virtual memory is implemented using Demand Paging or Demand Segmentation.

Virtual memory components



Main Memory

Main memory consists of a number of fixed-length frames, each equal to the size of a page. For a program to execute, some or all of its pages must be in main memory.



Disk

Secondary memory (disk) can hold many fixed-length pages. A user program consists of some number of pages. Pages for all programs plus the operating system are on disk, as are files.

Virtual Memory that is larger than Physical Memory