
Operating System Concepts

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

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



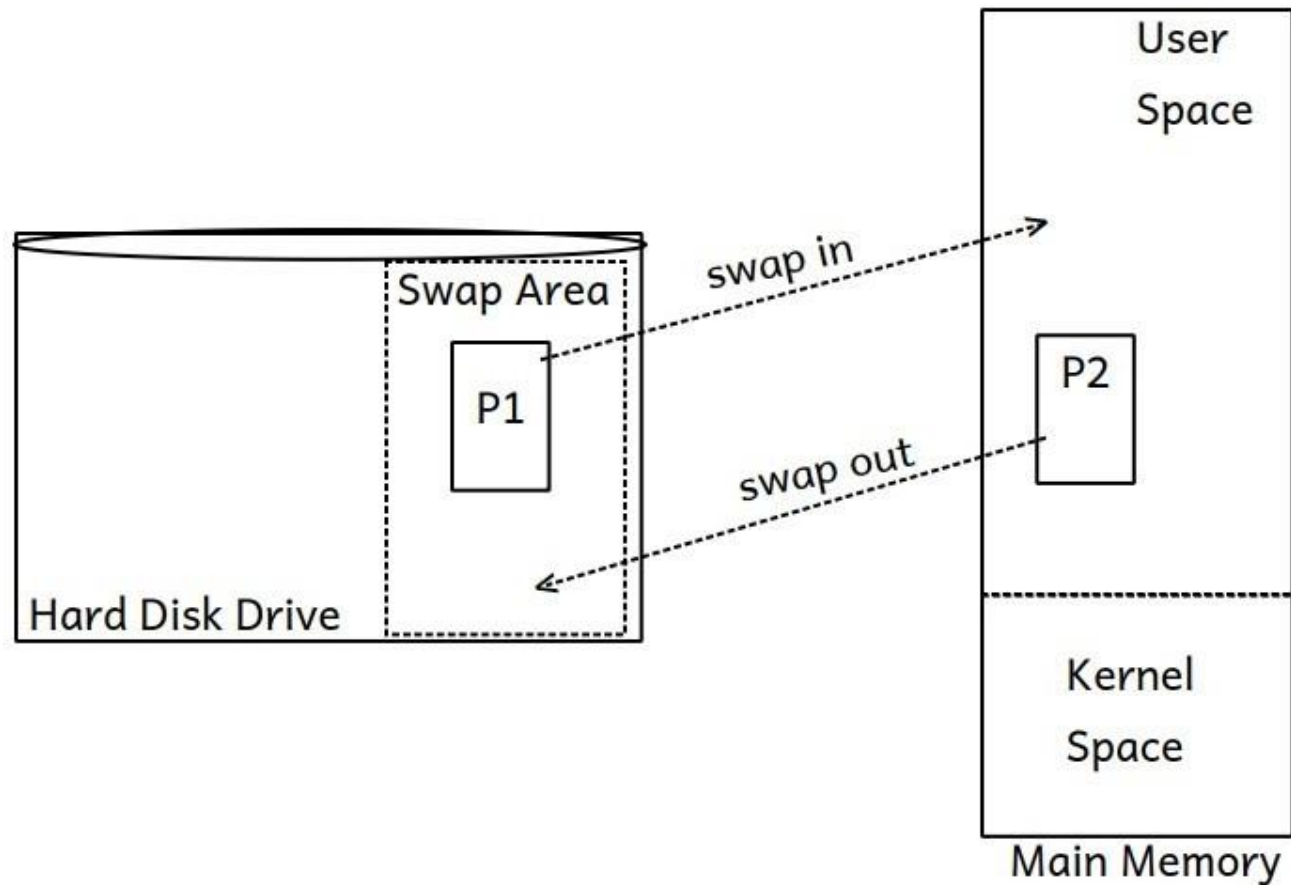
Memory Management

- 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



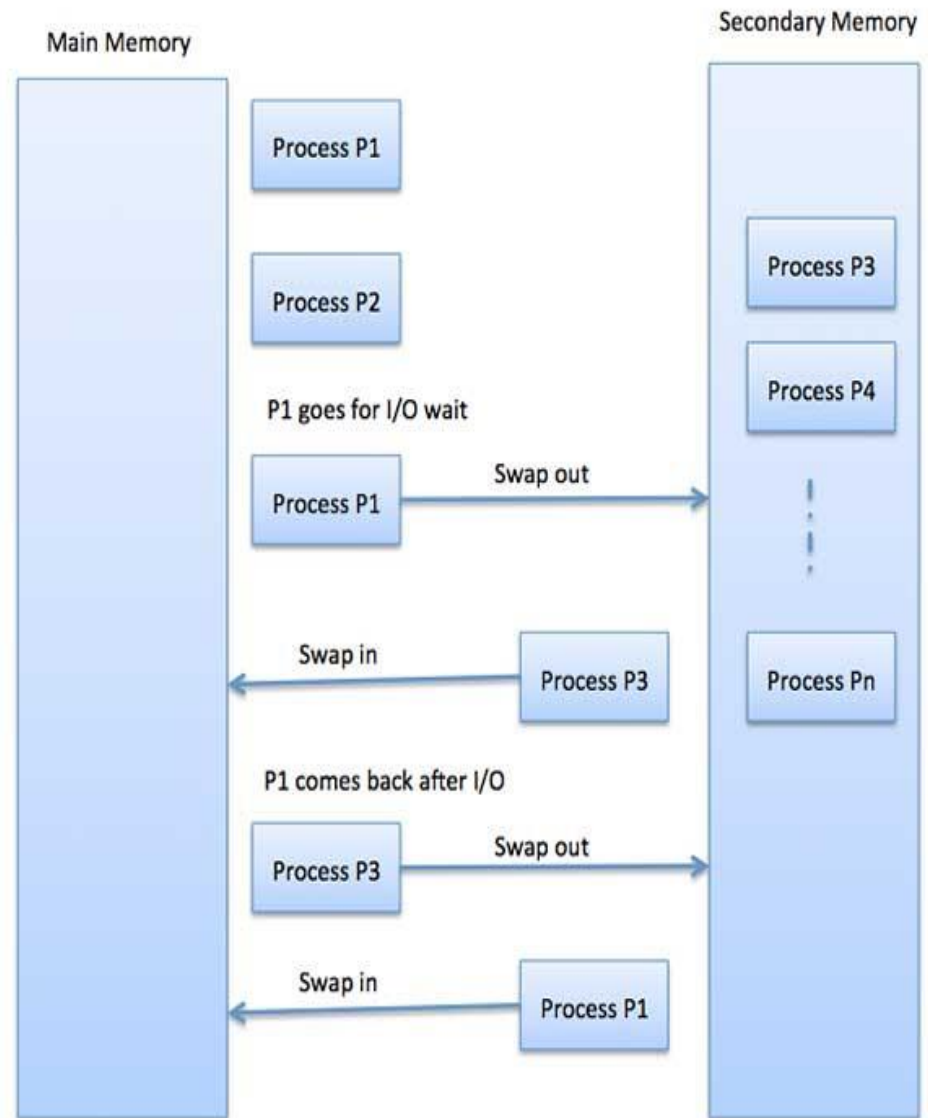
Swapping Memory Manager

SWAPPING: 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 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.**



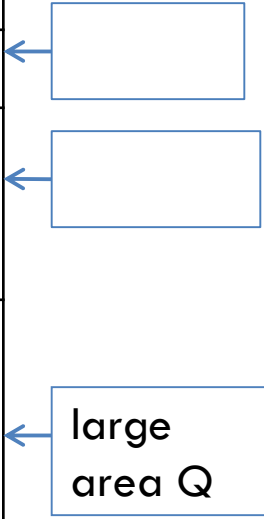
Swapping

- **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**.



Fixed Partition

	memory
	OS
n KB	small
3n KB	Medium
6n KB	Large



- 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.

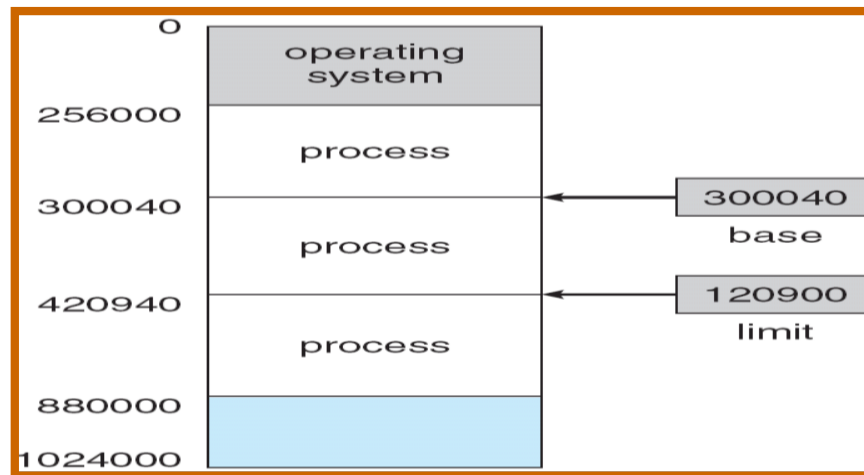


Variable Partition

- 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

- **First –fit**
- **Best Fit**
- **Worst Fit**



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

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



first fit

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

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

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 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



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

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

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

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 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

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

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



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

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

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

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



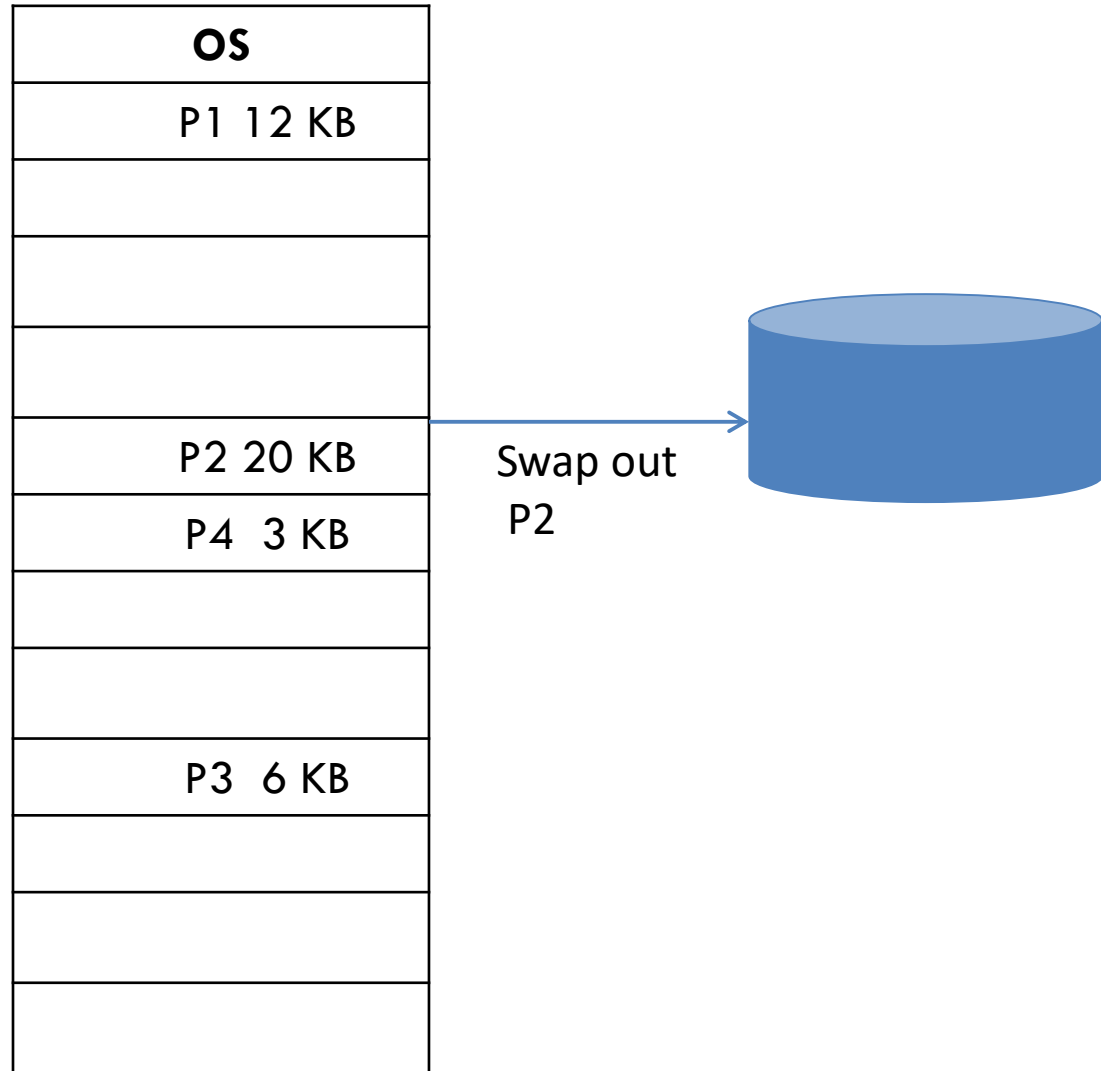
compaction

Memory mapping
before
compaction

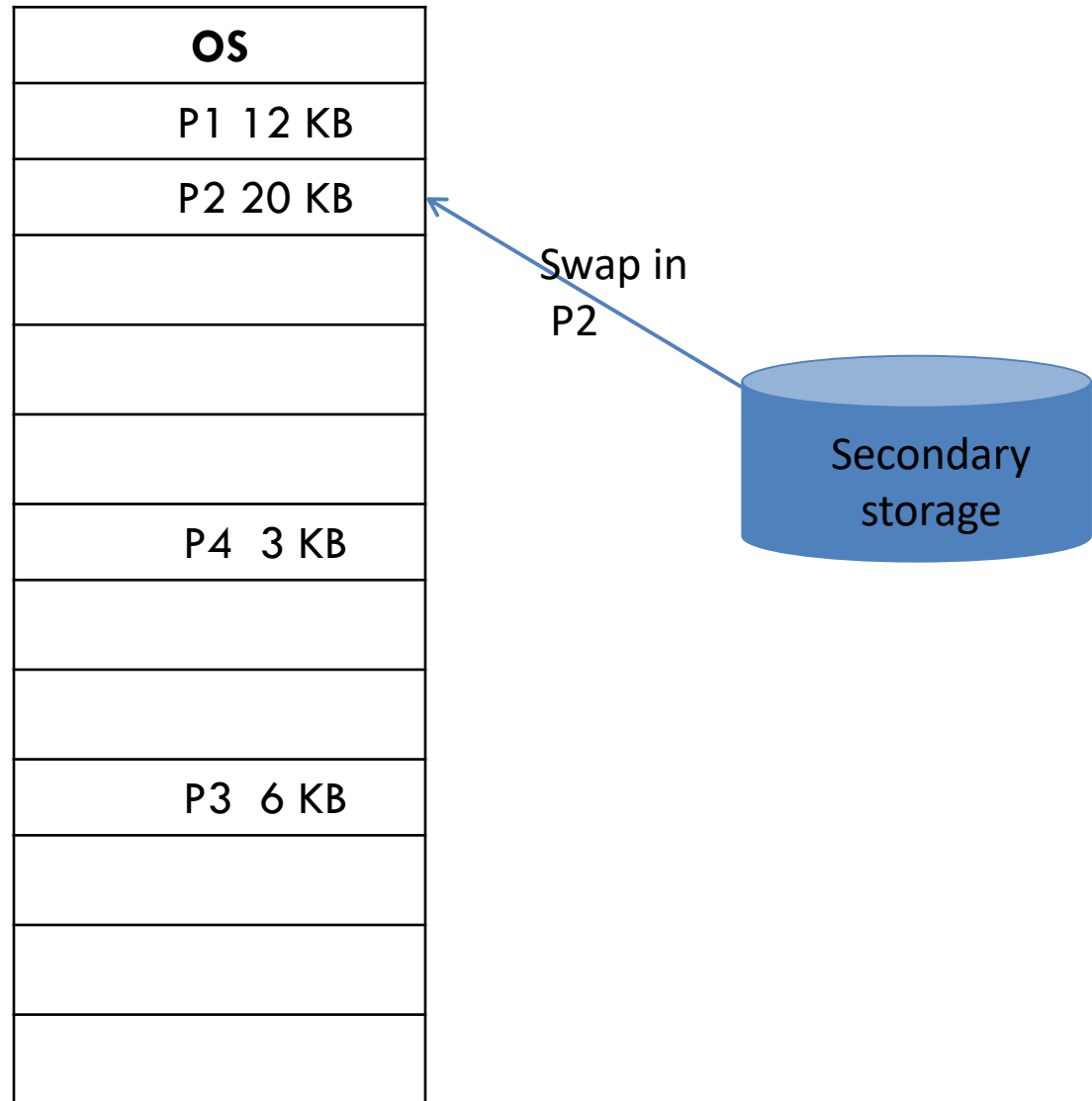
OS
P1 12 KB
<FREE> 10 KB
P2 20 KB
P4 3 KB
<FREE> 13 KB
P3 6 KB
<FREE> 4 KB



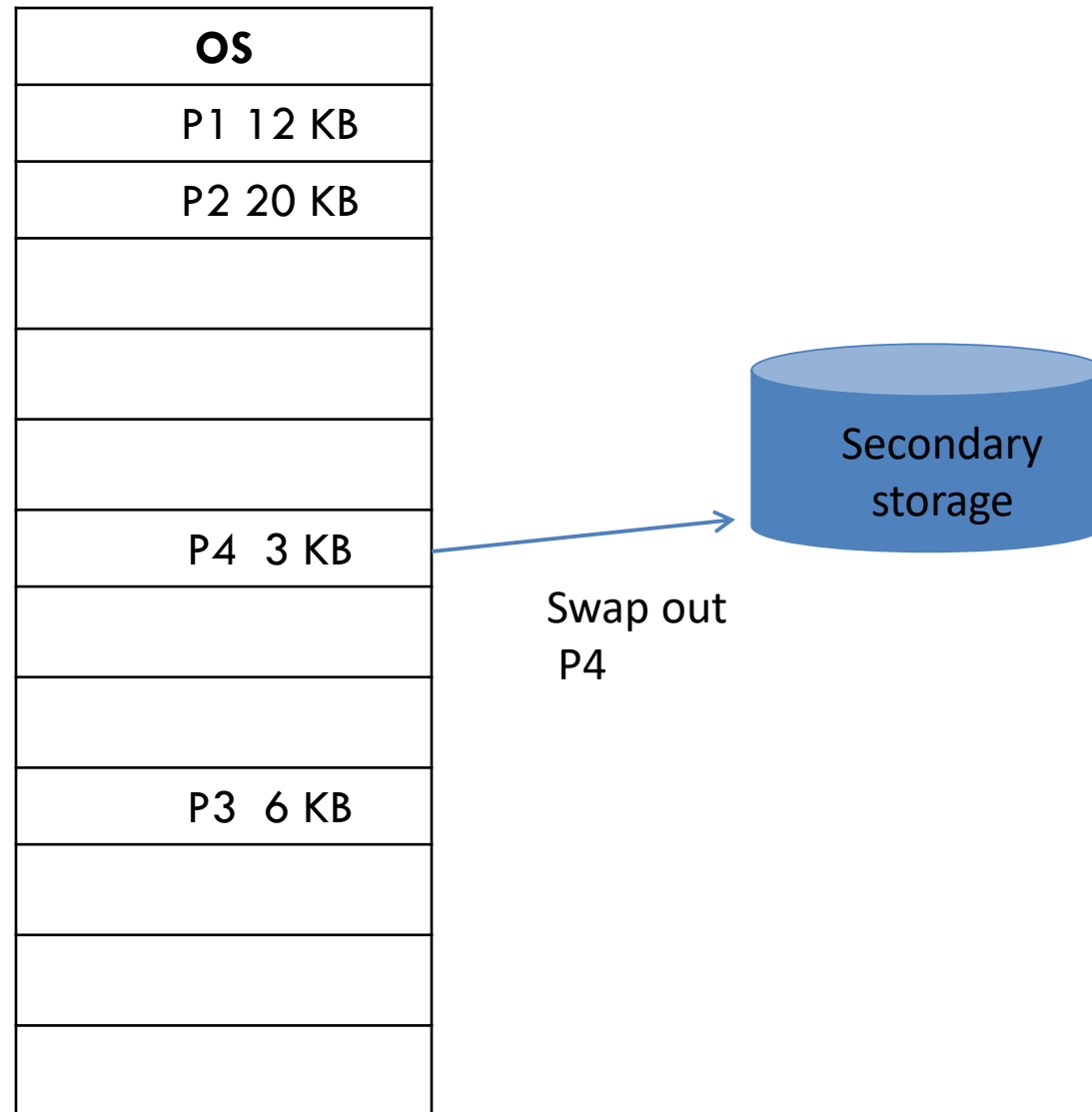
compaction



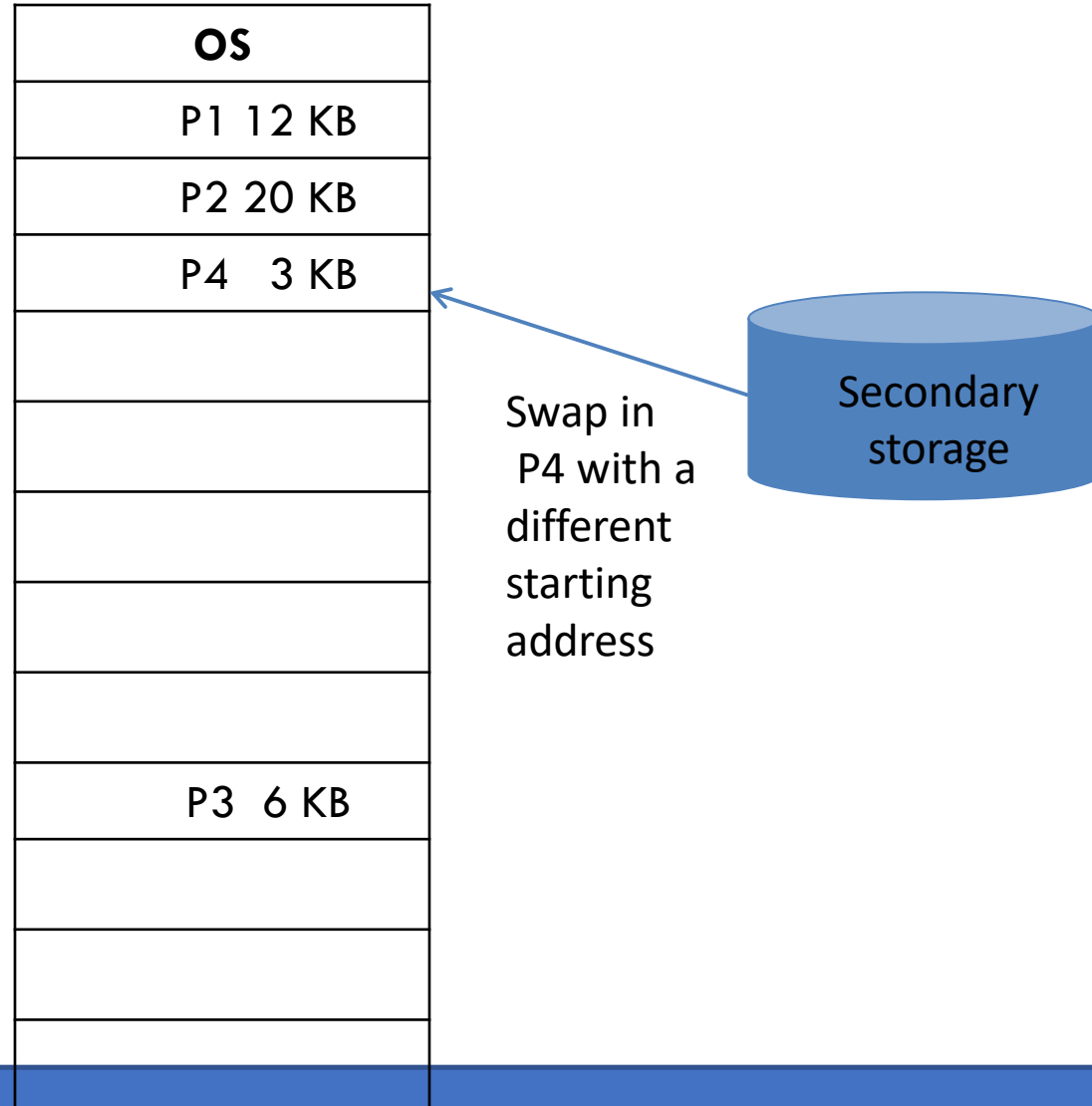
compaction



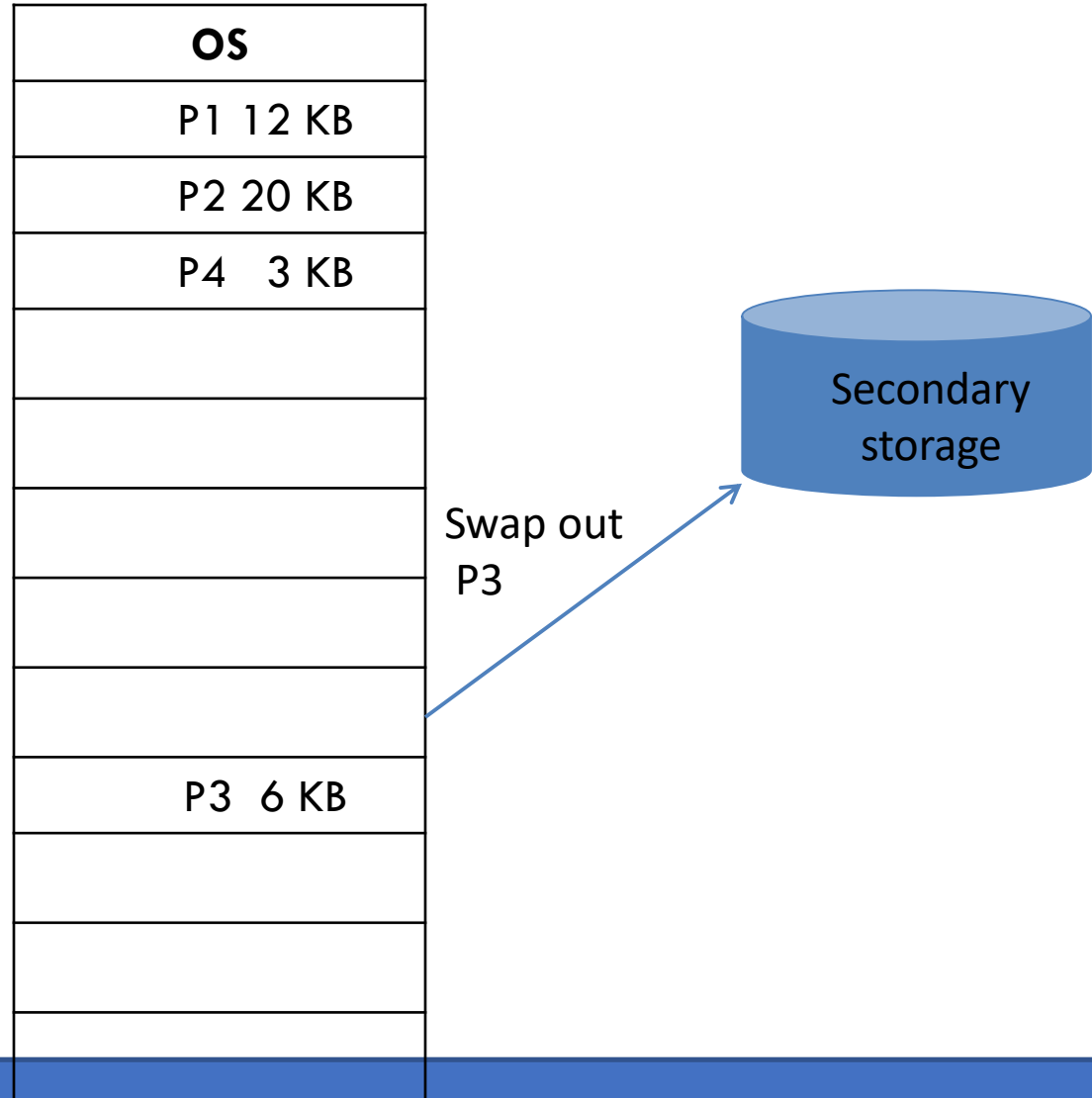
compaction



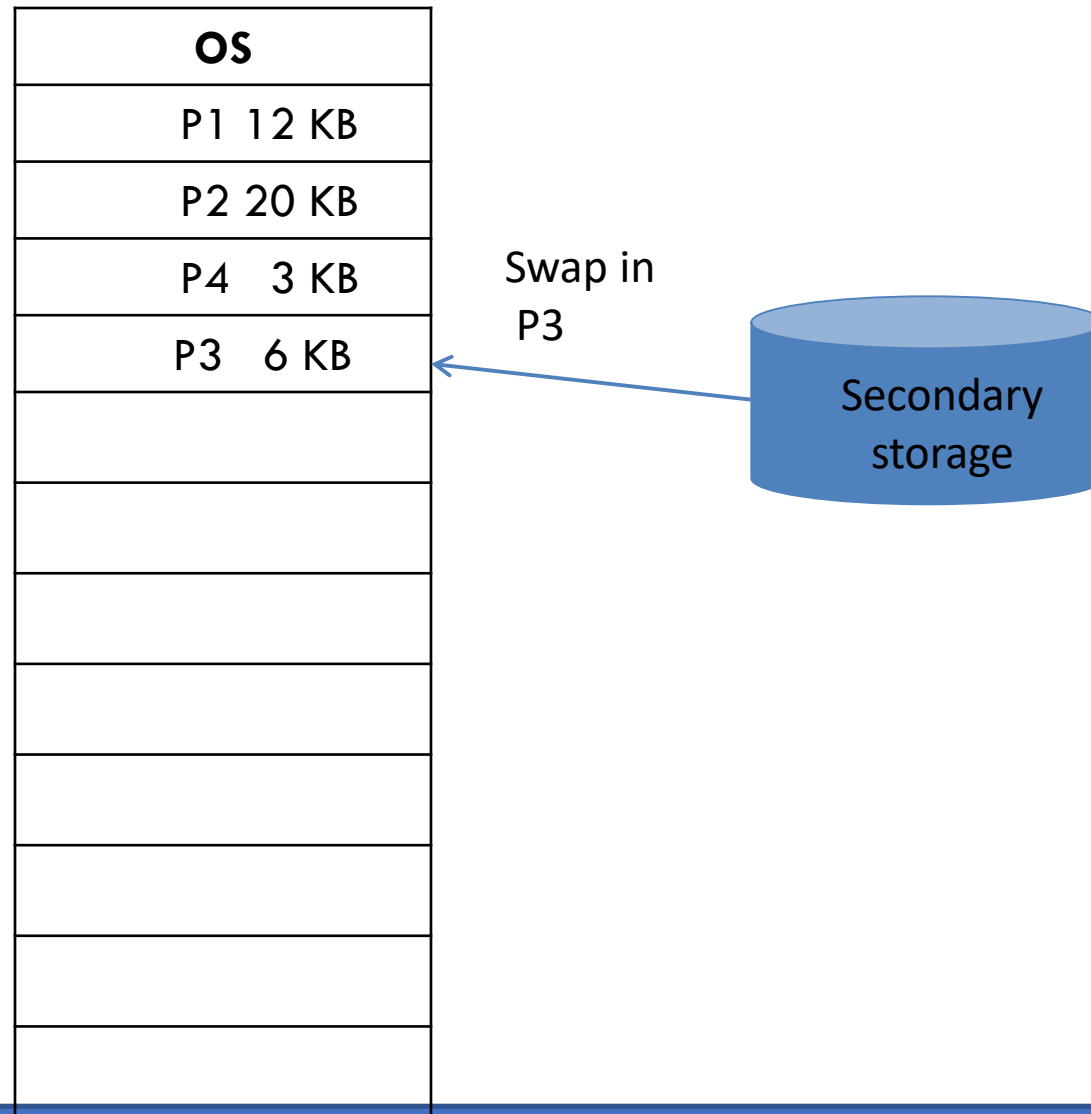
compaction



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

15KB

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

	memory
	OS
2K	P1 (2K)
6K	Empty (6K)
12K	P2 (9K)
	Empty (3K)

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**



Simple Paging

- Main memory is partitioned into equal fixed-sized chunks (of relatively small size)
- Trick: each process is also divided into chunks of the same size called **pages**
- The process pages can thus be assigned to the available chunks in main memory called **frames** (or page frames)
- Consequence: a process does not need to occupy a contiguous portion of memory



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

Frame number	Main memory
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	

(a) Fifteen Available Pages

Frame number	Main memory
0	A.0
1	A.1
2	A.2
3	A.3
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	

(b) Load Process A

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

(b) Load Process B

Frame number	Main memory
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	

(d) Load Process C

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

Main memory	
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	

(e) Swap out B

Main memory	
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	

(f) Load Process D

Page Tables

0	0
1	1
2	2
3	3

Process A
page table

0	—
1	—
2	—

Process B
page table

0	7
1	8
2	9
3	10

Process C
page table

0	4
1	5
2	6
3	11
4	12

Process D
page table

13
14

Free frame
list

- 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



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 instructions/data: no internal fragmentation
- There is external fragmentation; it is reduced when using small segments

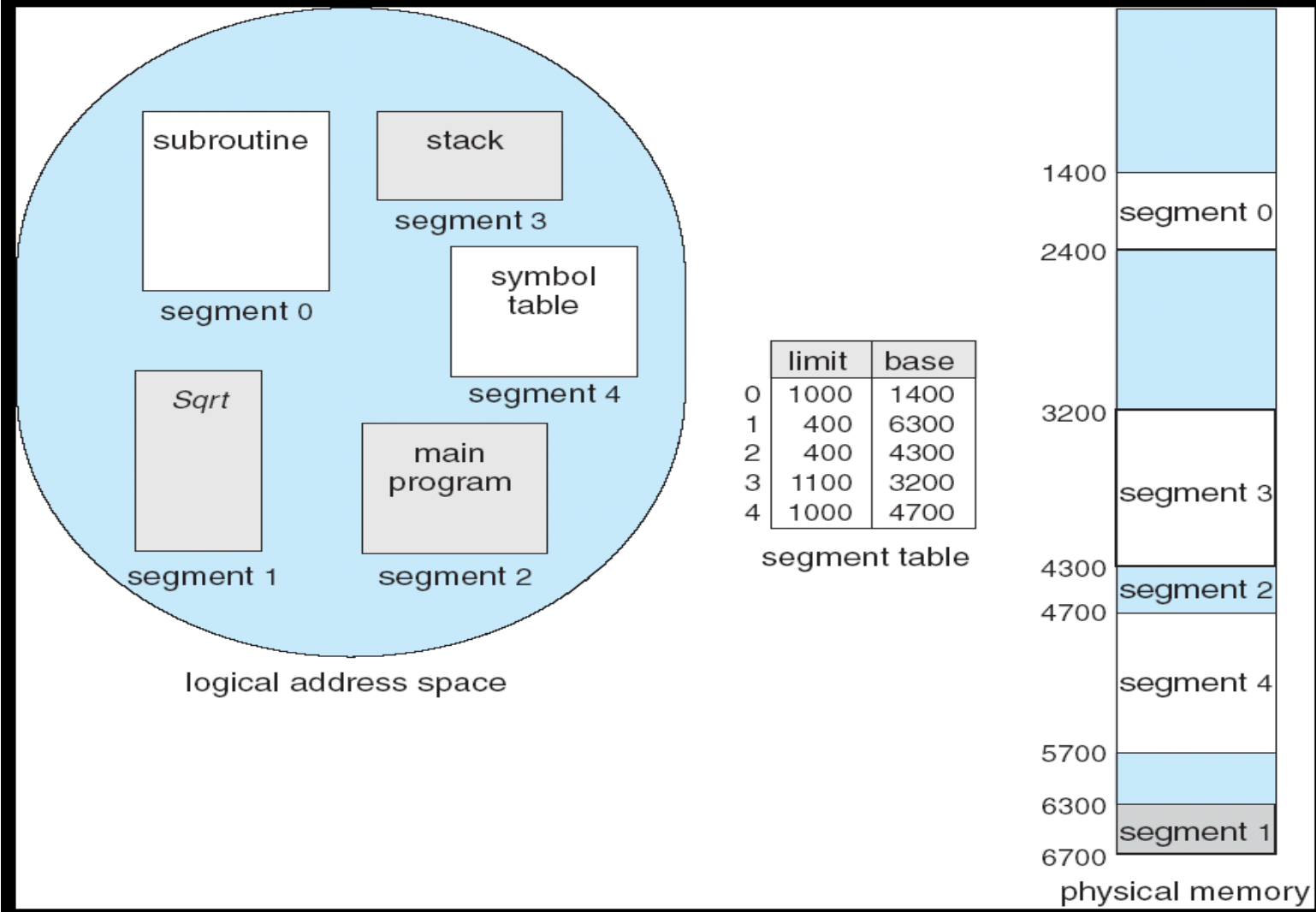


Segmentation

- 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.**



Demand Paging

- The process of loading the page into memory on demand (whenever page fault occurs) is known as demand paging.
- Bring a page into memory only when it is needed:
 - Less I/O needed, no unnecessary I/O
 - Less memory needed
 - Faster response
 - More users
- Page is needed \Rightarrow reference to it:
 - invalid reference \Rightarrow abort
 - not-in-memory \Rightarrow bring to memory
- Similar to paging system with swapping.
- Lazy swapper – never swaps a page into memory unless page will be needed; Swapper that deals with pages is a pager.**



Page Table when some pages are not in Main Memory

0	A
1	B
2	C
3	D
4	E
5	F
6	G
7	H

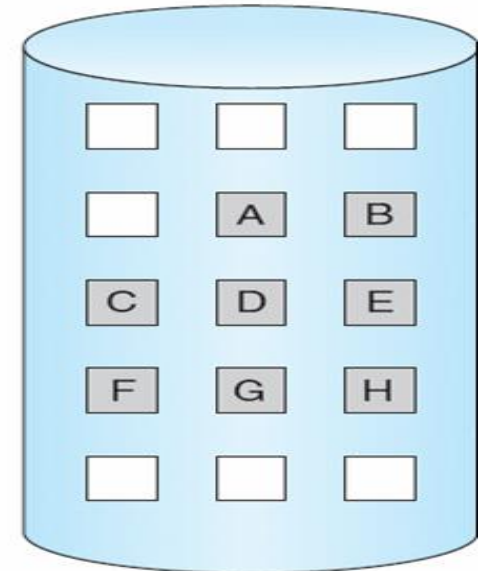
logical
memory

valid-invalid	
frame	bit
0	4 v
1	i
2	6 v
3	i
4	i
5	9 v
6	i
7	i

page table

0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

physical memory

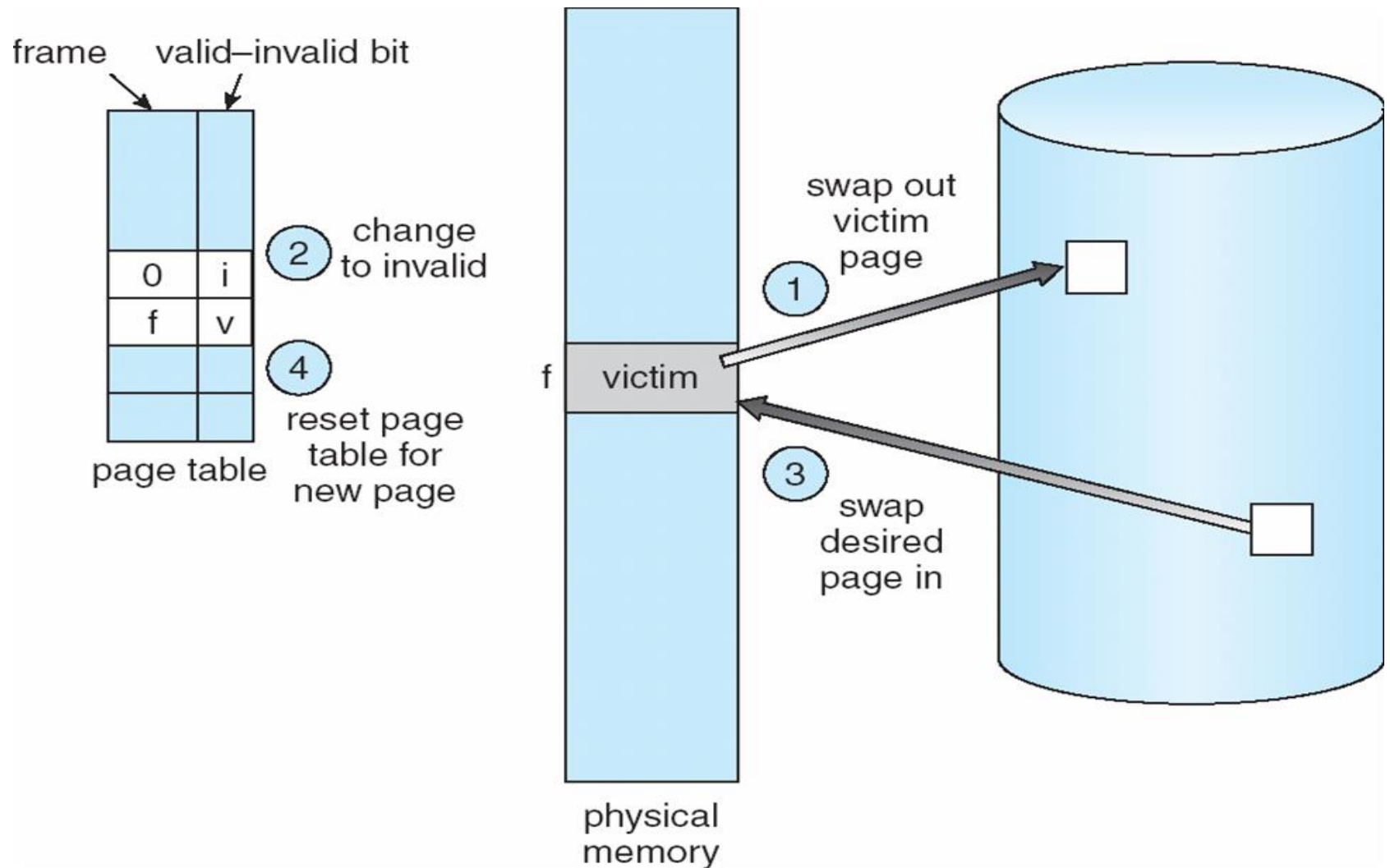


What happens if there is no free frame?

- Page replacement – find some page in memory, but not really in use, swap it out.
- Need page replacement algorithm.
- Performance – want an algorithm which will result in minimum number of page faults.
- Same page may be brought into memory several times.



Steps in handling a Page Replacement



Page Replacement Algorithms

- FIFO,
- LRU,
- Optimal,



FIFO Page Replacement

reference string

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

7	7	7	2																
	0	0	0																
		1	1																

2	2	4	4	4	0														
3	3	3	2	2	2														
1	0	0	0	3	3														

0	0																		
1	1																		
3	2																		

7	7	7																	
1	0	0																	
2	2	1																	

page frames



LRU Page Replacement

reference string

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

7	7	7	2		2		4	4	4	0			1		1		1		
	0	0	0		0		0	0	3	3			3		0		0		
		1	1		3		3	2	2	2			2		2		7		

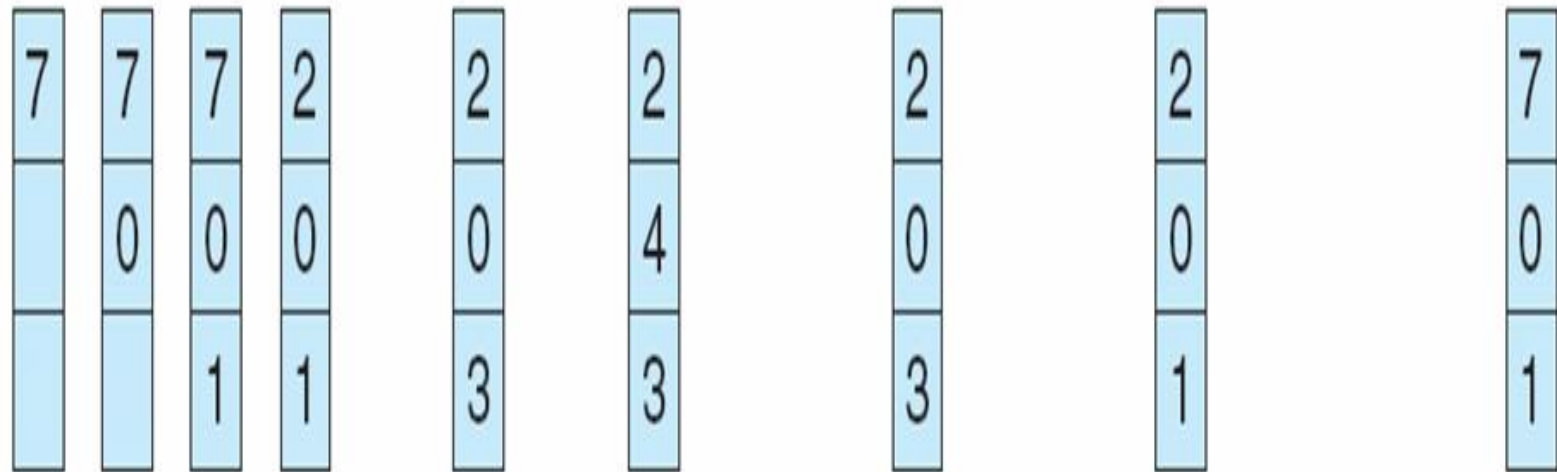
page frames



Optimal Page Replacement

reference string

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1



page frames



File

- file is a collection of logically related data or information
- file is a stream of bytes/bits
- file is a basic storage unit
 - File = data+metadata
 - Data : Actual File Contents
 - Metadata : Information about file.
- File Attributes:
 - Name,type,location,size etc..
- File Operations
 - Create,delete,write,read
- **"file system" is a way to store data onto the disk in an organized manner so that it can accessed efficiently**
- e.g. Each OS has its own filesystem like, UNIX: UFS(UNIX Filesystem), Linux: Extended filesystem ext2, ext3, ext4, Windows: FAT, NTFS etc..., MAC OSX: HFS(Hierarchical Filesystem) etc...



What is an inode / FCB

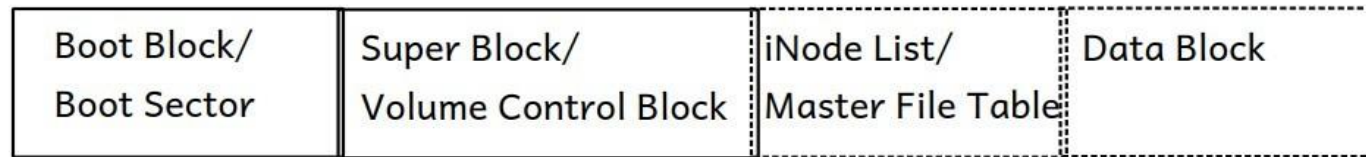
- An inode (index node) is a control structure that contains key information needed by the OS to access a particular file. Several file names may be associated with a single inode, but each file is controlled by exactly ONE inode.
- On the disk, there is an inode table that contains the inodes of all the files in the filesystem. When a file is opened, its inode is brought into main memory and stored in a memory-resident inode table.
- Information about the file can be kept in one structure referred as "FCB" i.e. File Control Block/iNode
 - inode/FCB contains info about the file like:
 - name of the file
 - type of the file
 - size of the file
 - parent folder location
 - access perms for user/owner, grp member and others etc



File System Structure

File system divides disk/partition logically into sectors/blocks, like **boot sector/boot block**, **volume control block/super block**, **master file table/iNode list block** and **data**

FILESYSTEM STRUCTURE



1. Boot Block: It contains information about booting the system like bootstrap program, bootloader etc...
2. Super Block: It contains information about remaining sections, like total no. of data blocks, no. of free data blocks, no. of allocated data blocks etc....
3. iNode List: It contains linked list of iNode's of all files exists on a disk.
4. Data Block: It contains actual data.

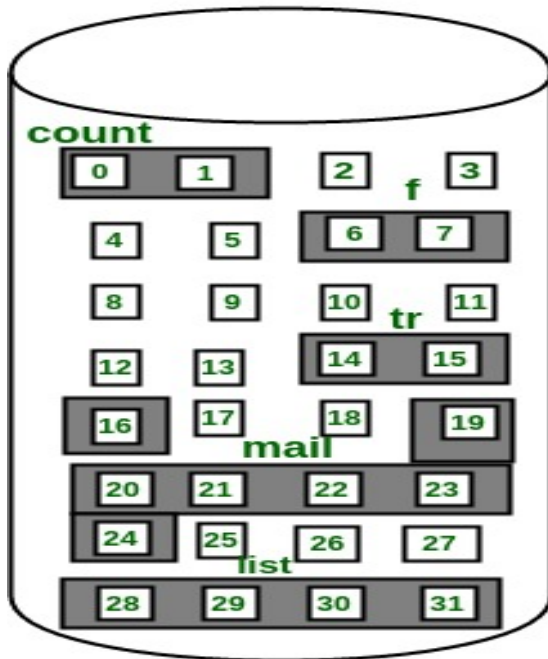


File Allocation on Disk

- Low level access methods for a file depend upon the disk allocation scheme used to store file data
 - Contiguous
 - Linked
 - Block or indexed



Contiguous Allocation



Directory

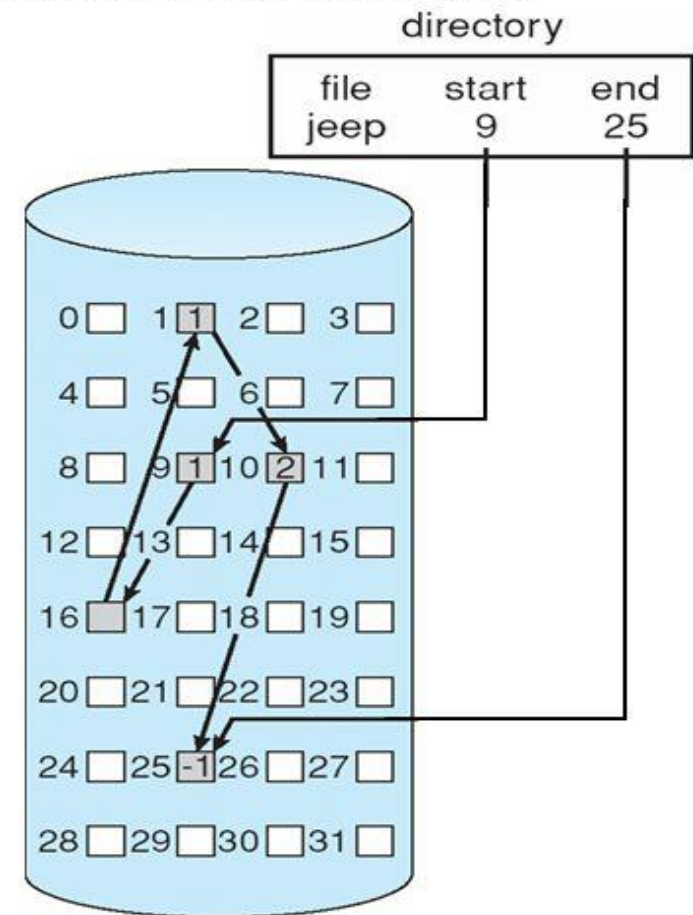
file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

- File is allocated large contiguous chunks
- Expanding the file requires copying
- Dynamic storage allocation - first fit, best fit
- **External fragmentation occurs on disk**



Linked Allocation

- Each file is a linked list of disk blocks, which may be scattered on the disk
- Directory contains a pointer to the first and last blocks, and each block contains a pointer to the next block
- **Advantages:**
 - No external fragmentation
 - Easy to expand the size of a file
- **Disadvantages:**
 - Not suitable for random access within a file
 - Pointers take up some disk space
 - Difficult to recover a file if a pointer is lost or damaged
- Blocks may be collected into **clusters** of several blocks
 - Fewer pointers are necessary
 - Fewer disk seeks to read an entire file
 - Greater internal fragmentation

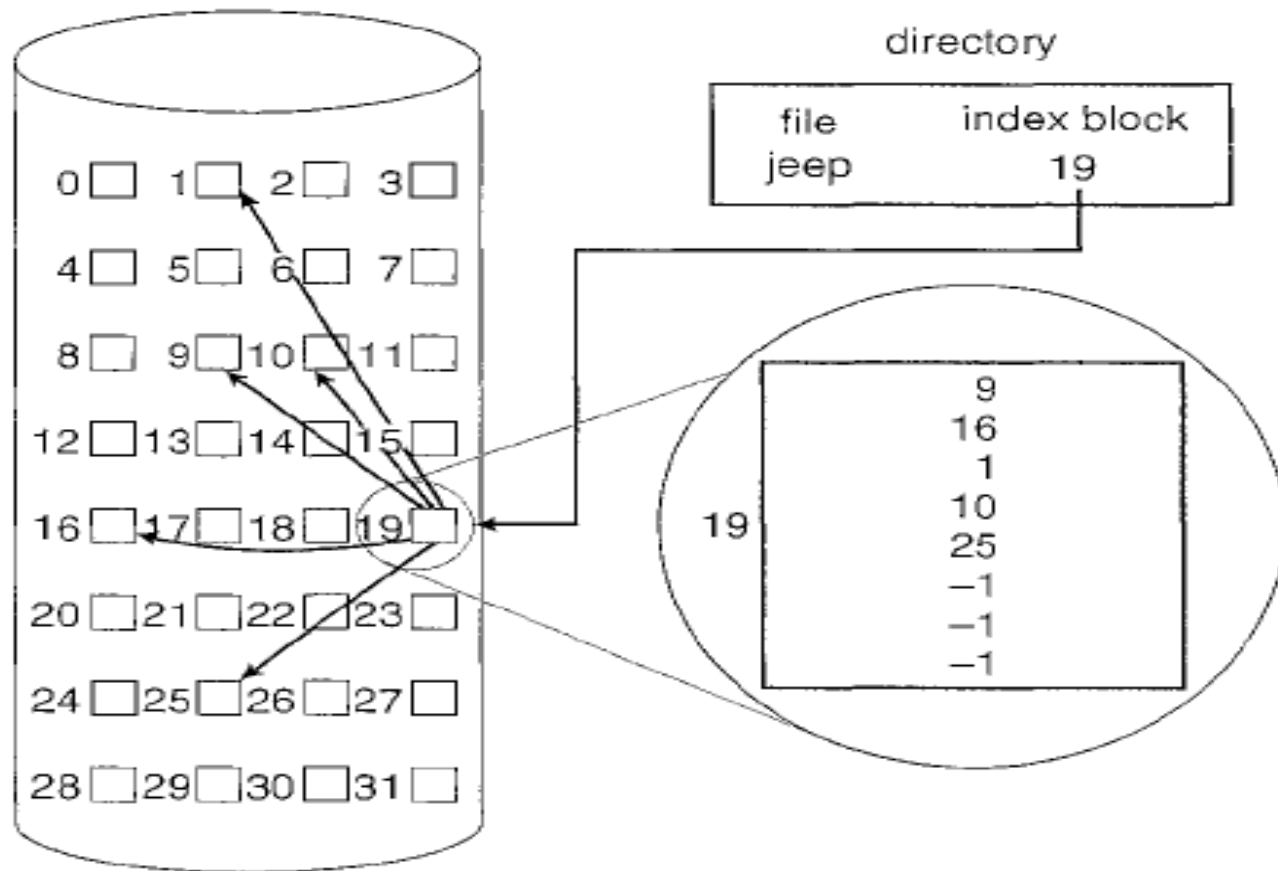


Block / Indexed

- A special block known as the **Index block** contains the pointers to all the blocks occupied by a file.
- The i th entry in the index block contains the disk address of the i th file block.
- The directory entry contains the address of the index block.
- When the file is created, all pointers in the index block are set to *nil*.
- *When* the i th block is first written, a block is obtained from the free-space manager and its address is put in the i th index-block entry.



Indexed Allocation



Disk Scheduling

- **Disk scheduling** is done by operating systems to schedule I/O requests arriving for the disk.
- Disk scheduling is important because:
 - Multiple I/O requests may arrive by different processes and only one I/O request can be served at a time by the disk controller. Thus other I/O requests need to wait in the waiting queue and need to be scheduled.
 - Two or more request may be far from each other so can result in greater disk arm movement.
 - Hard drives are one of the slowest parts of the computer system and thus need to be accessed in an efficient manner.



Disk Scheduling Algorithms

First Come First Serve

- FCFS, the requests are addressed in the order they arrive in the disk queue.

Shortest Seek Time First

- SSTF (Shortest Seek Time First), requests having shortest seek time are executed first.
- it decreases the average response time and increases the throughput of system.

Scan/Elevator

- SCAN algorithm the disk arm moves into a particular direction and services the requests coming in its path and after reaching the end of disk, it reverses its direction and again services the request arriving in its path.

CSCAN

- disk arm moves in a circular fashion and this algorithm is also similar to SCAN algorithm and hence it is known as C-SCAN (Circular SCAN).

Look

- similar to the SCAN disk scheduling algorithm except for the difference that the disk arm in spite of going to the end of the disk goes only to the last request to be serviced in front of the head and then reverses its direction from there only.

Clook

- CLOOK, the disk arm in spite of going to the end goes only to the last request to be serviced in front of the head and then from there goes to the other end's last request.



FCFS(First Come First Serve)

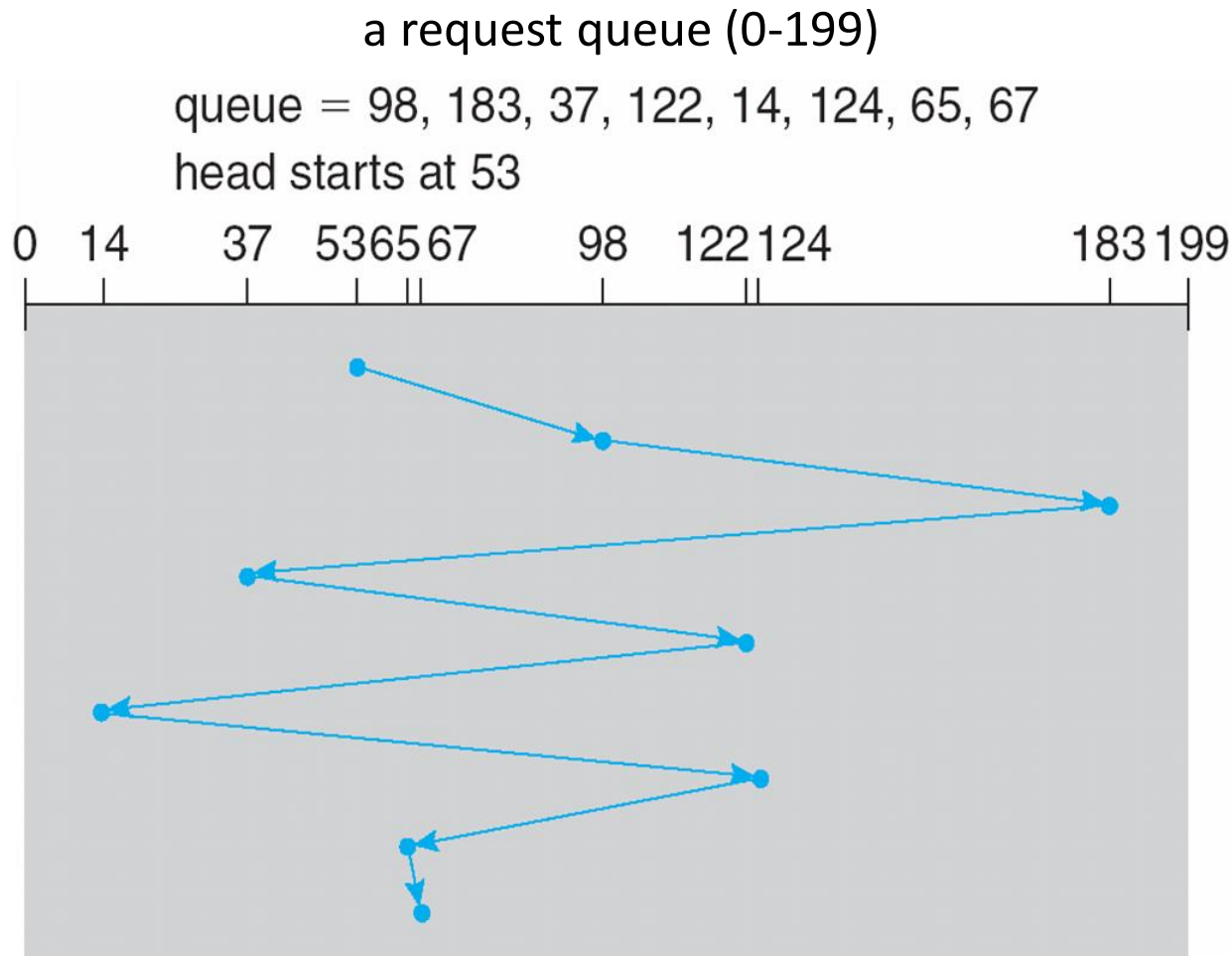


Illustration shows total head movement of 640 cylinders.



SSTF(Shortest Seek Time First)

- Selects the request with the minimum seek time from the current head position
- SSTF scheduling is a form of SJF scheduling; may cause starvation of some requests

queue = 98, 183, 37, 122, 14, 124, 65, 67

head starts at 53

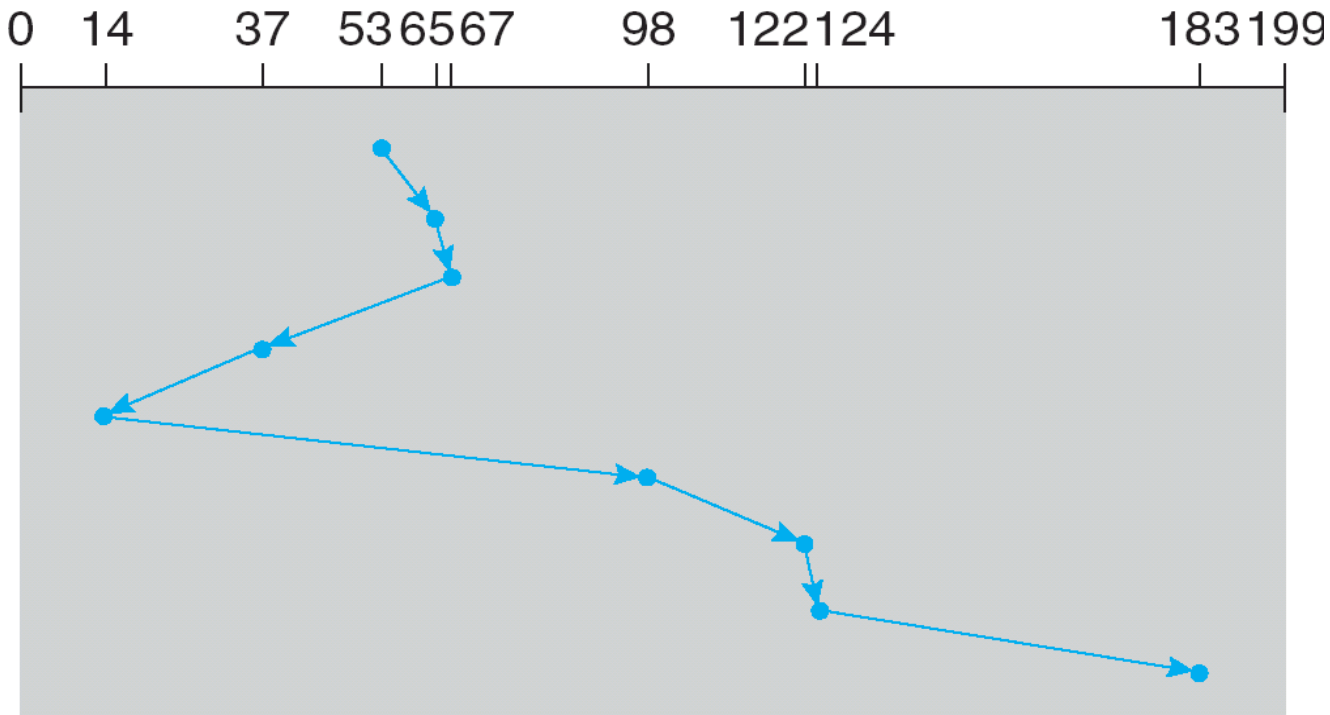


Illustration shows total head movement of 236 cylinders.



SCAN

- The disk arm starts at one end of the disk, and moves toward the other end, servicing requests until it gets to the other end of the disk, where the head movement is reversed and servicing continues.

- SCAN algorithm Sometimes called the [elevator algorithm](#)

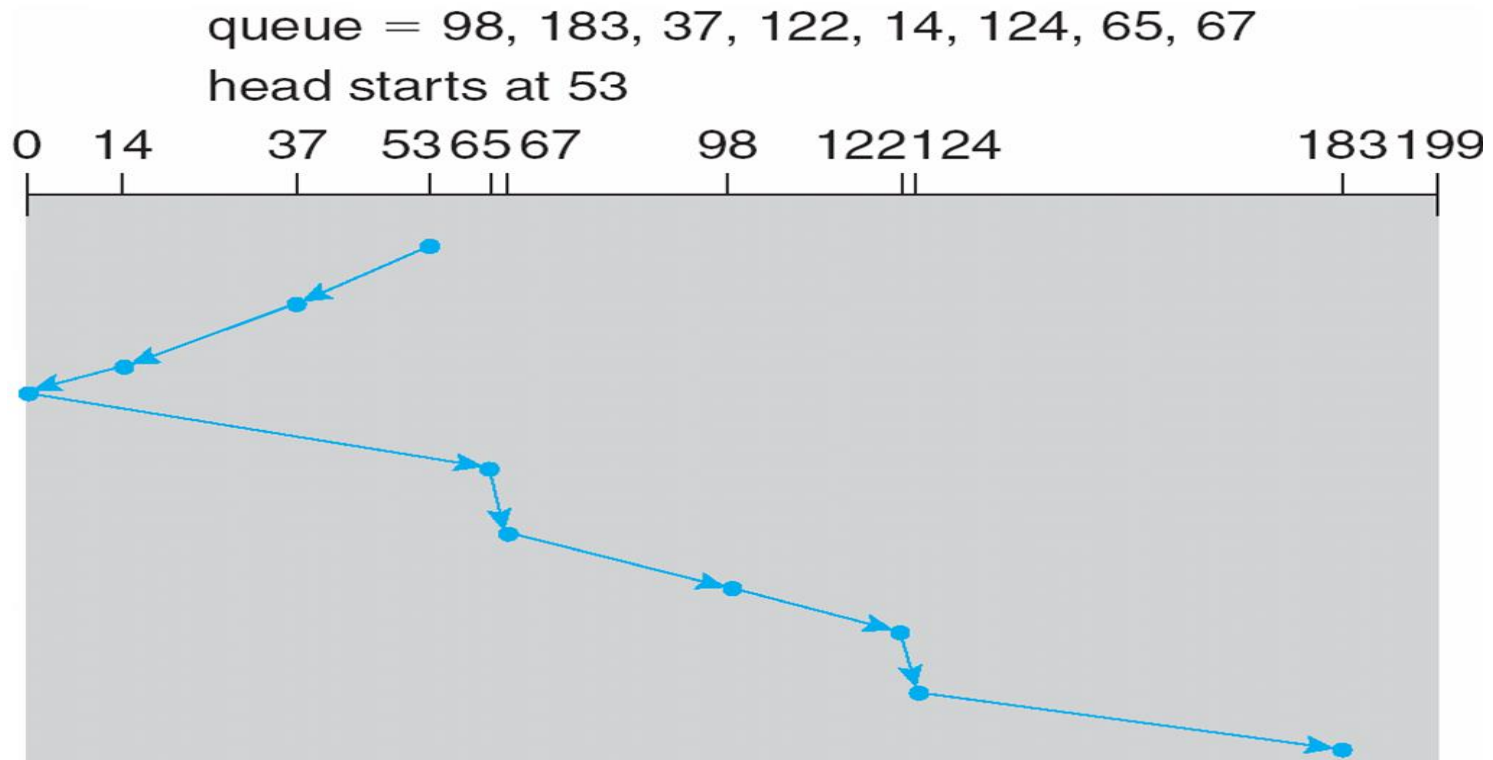
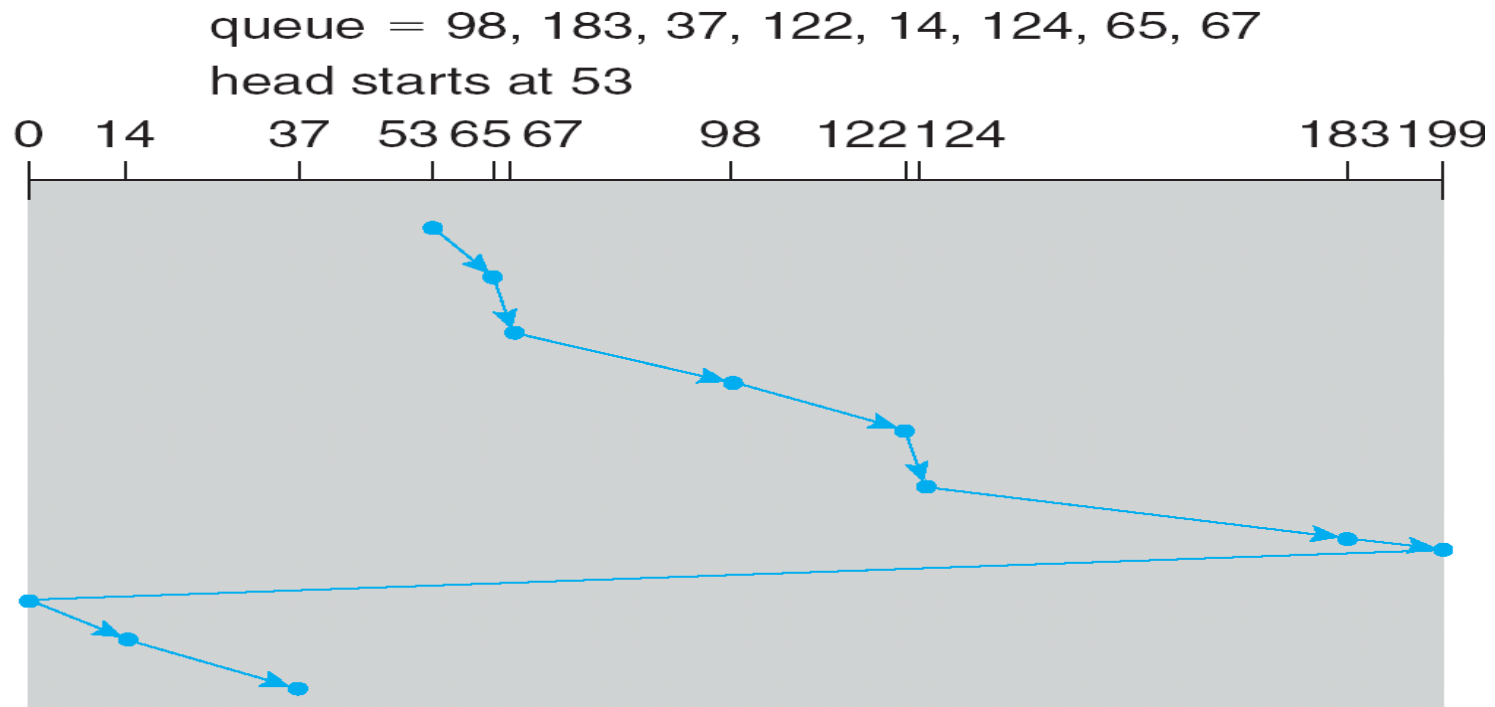


Illustration shows total head movement of 208 cylinders.



C-SCAN (Cont)

- Provides a more uniform wait time than SCAN
- The head moves from one end of the disk to the other, servicing requests as it goes
 - When it reaches the other end, however, it immediately returns to the beginning of the disk, without servicing any requests on the return trip
- Treats the cylinders as a circular list that wraps around from the last cylinder to the first one



C-LOOK

- Version of C-SCAN
- Arm only goes as far as the last request in each direction, then reverses direction immediately, without first going all the way to the end of the disk

