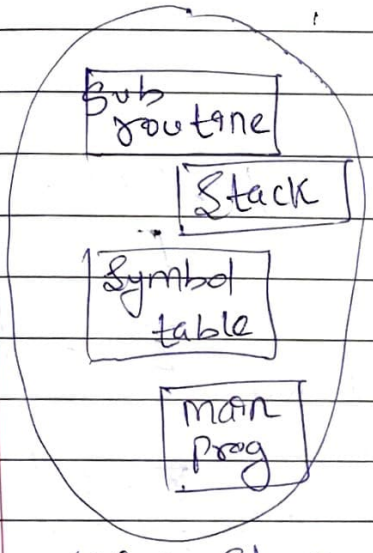
classmate
Date 6/DEC
Page

Segmentation —

- A program is a collection of segments
- A segment is a logical unit such as
  main program
  procedure
  function
  local var., global var



User space

⇒ Specifies each address by 2 quantities
(a) segment name
(b) segment offset

logical address ⇒ <Segment #, offset>

Segmentation Architecture —

→ logical add. contain 2 tuples <seg no., offset>

→ Segment table — maps 2 dimensional physical address —
- base :- contains the starting physical add.
- limit :- Specifies the length of seg.

Segment table

Base	limit
add. of seg.	length of seg.

* Segment table base register (STBR) —

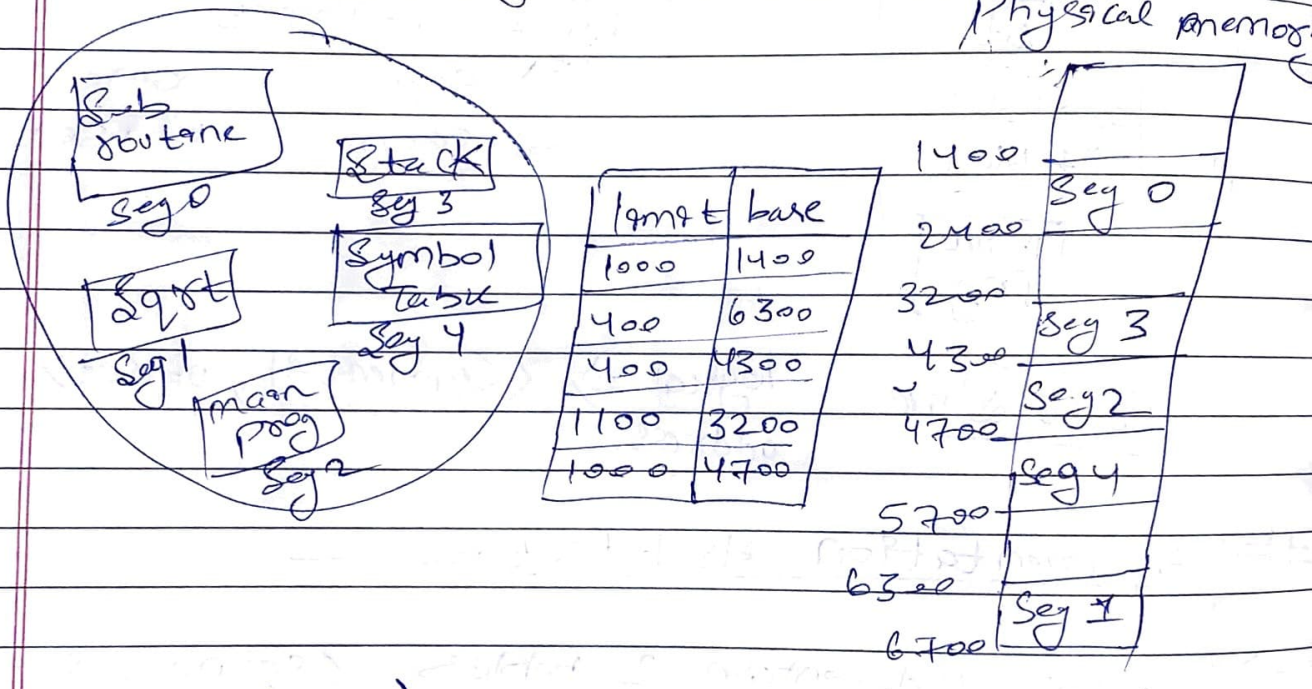
Points to the seg. tables location in memory

* Segment table length register (STLR) —

Indicates no. of segments used by prog.

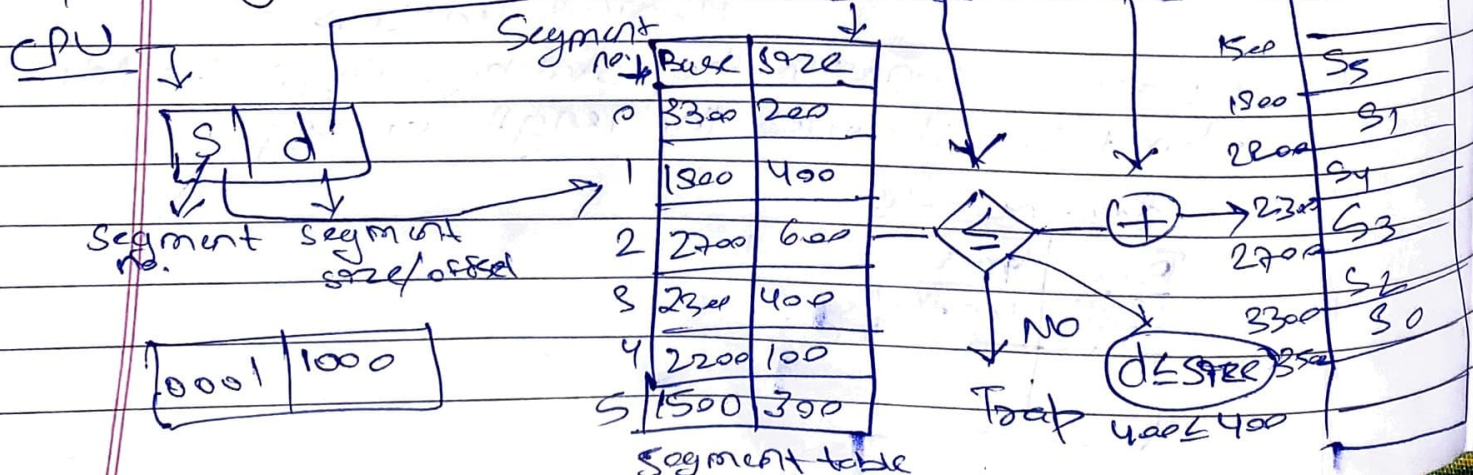
. Seg no. s is legal if s < STLR.

Example of Segmentation —

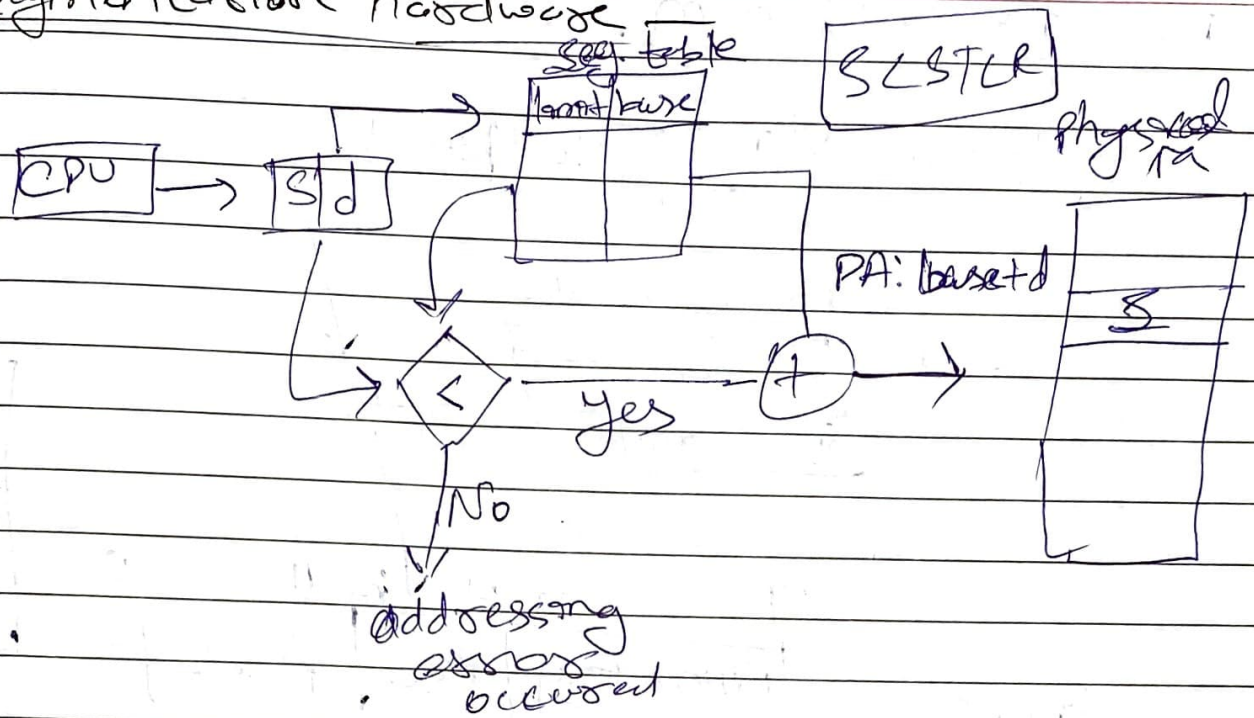


(Write Smeshur)

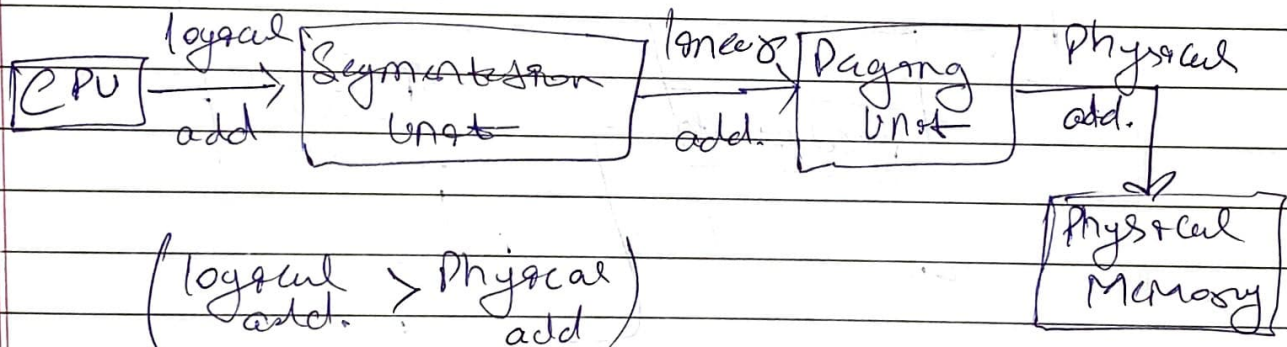
Segmentation with Paging —



* Segmentation hardware

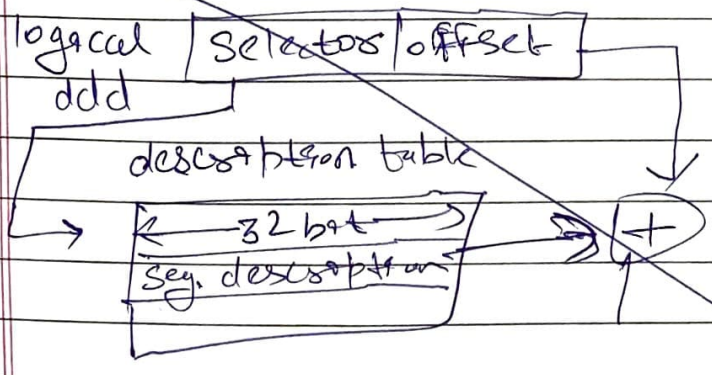


* Segmentation with Paging —

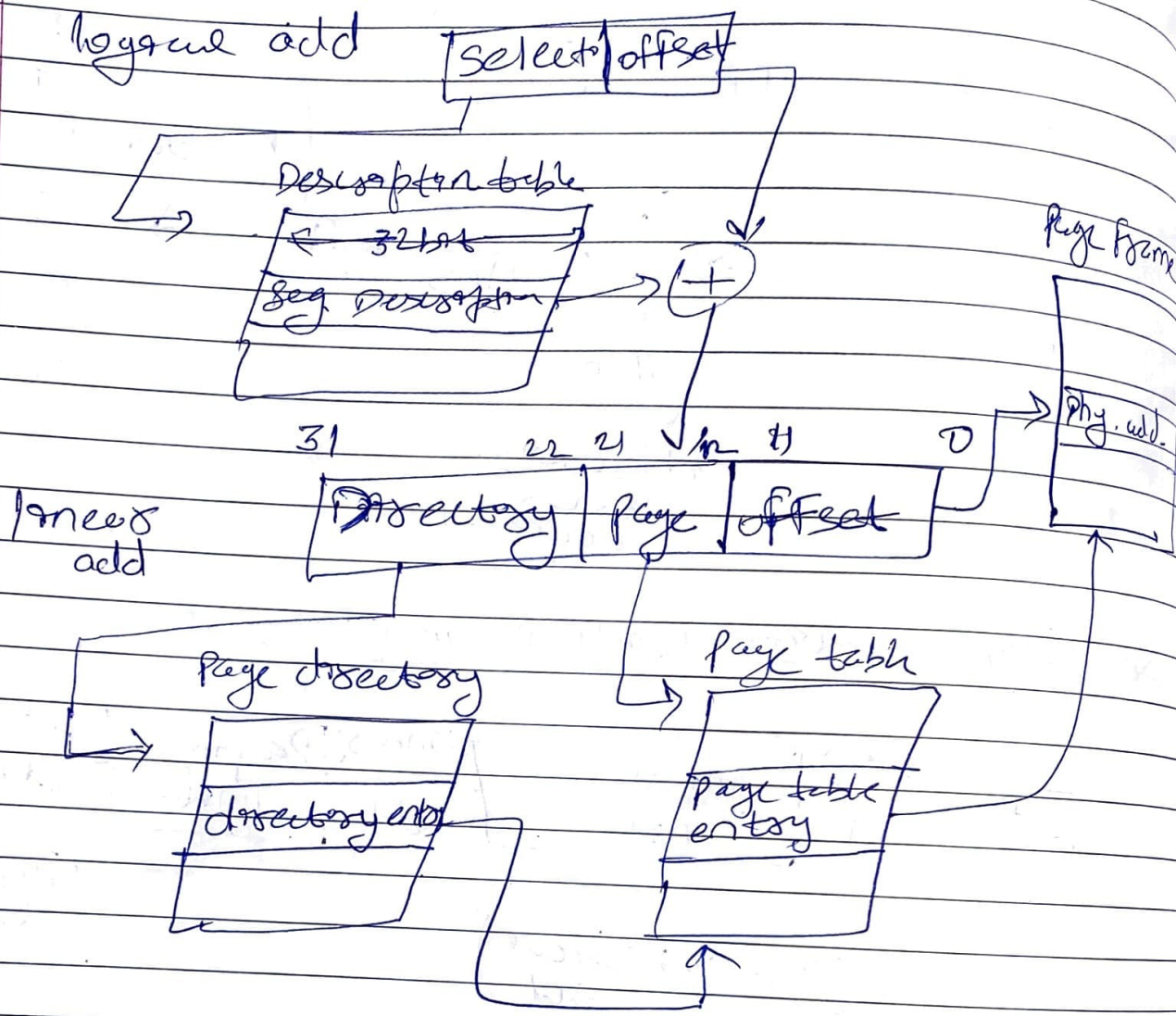


(logical add. > Physical add)

Ex → Intel pentium,



Ex intel pentium →



Virtual Memory

16 GB ⟺ 4 GB

User use V.M. — Real memory of computer

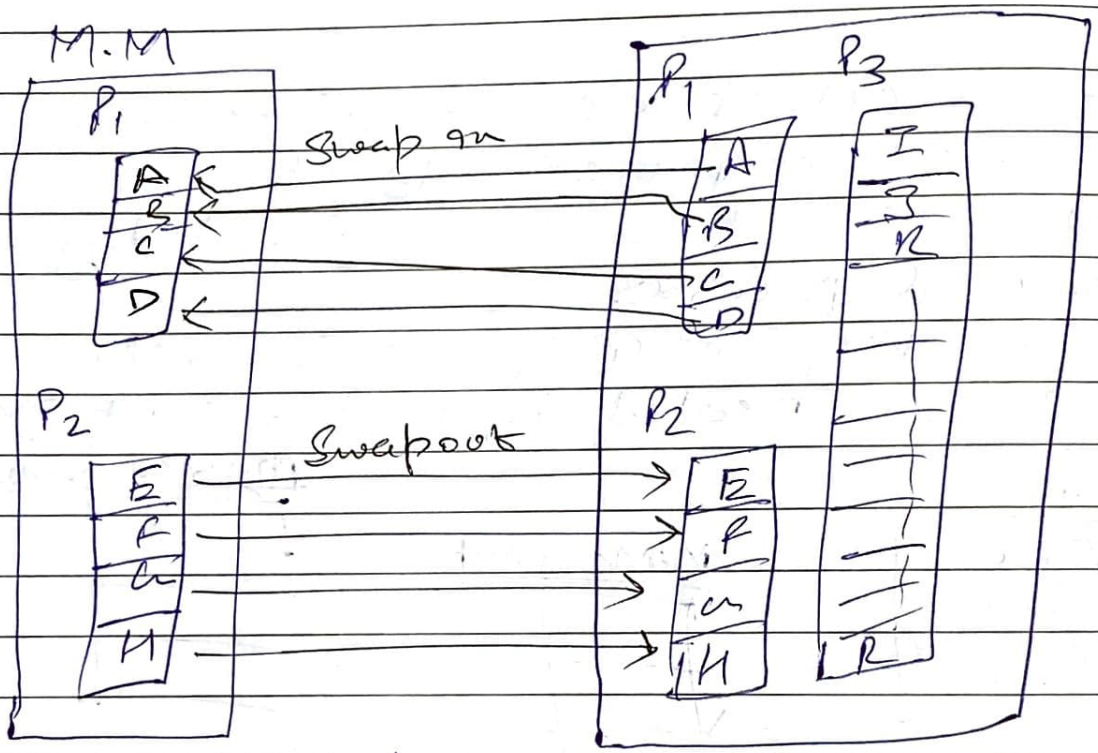
- A programmer can write a prog. which requires more memory than the capacity of M.M. Such a prog. is executed by V.M. Technique.
- logical add. space can therefore be much larger than physical add. space
- Allows add. spaces to be shared by several processes.
- More prog. running concurrently
- less I/O needed to load or swap processes.

V.M cont

- V.M. can be implemented via —
  - Demand Paging
  - Demand Segmentation

* Demand Paging —

The processes reside on Secondary Memory & Pages are loaded only on demand not in advance

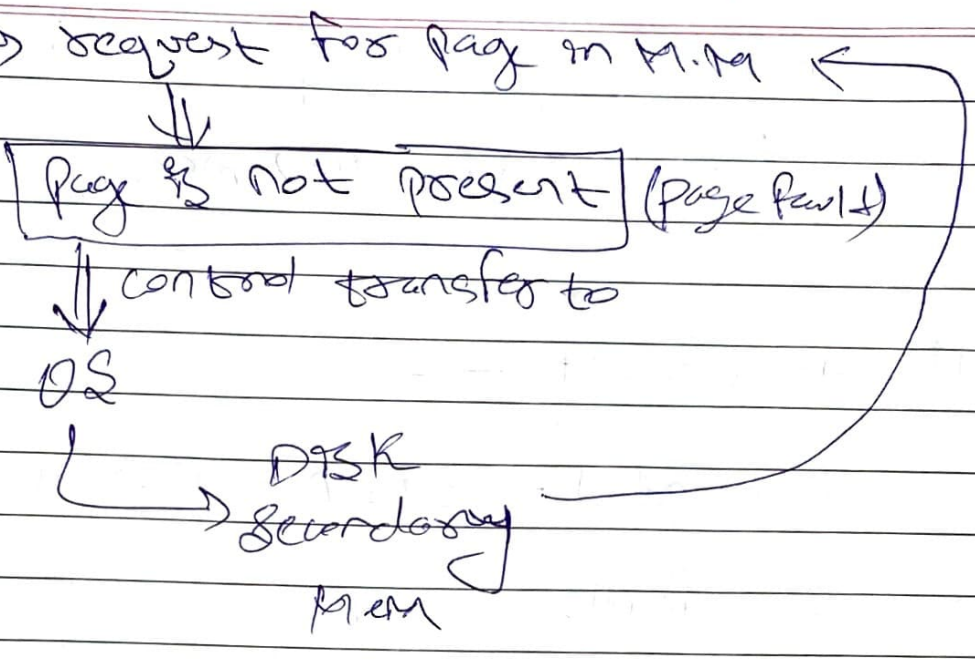


Demand
CPU → E (Page)

If the page is not present on M.M the we call it as **Page fault.**

How to handle it?

CPU → transfer control from prog to OS to demand the page back into memory.



Performance of Demand Paging —

⇒ Page fault rate $0 \le P \le 1.0$
if $P = 0$, no page fault
if $P = 1$, every reference is a fault

⇒ Effective Access time (EAT)

EAT = $(1-P) \times$ Memory Access
$+ P$ (Page fault overhead
+
Swap page out
+
Swap page in
+
restart overhead)

Ex Memory access time = 200 nano sec
avg. page fault service time = 8 mil sec

EAT = $(1-P) \times 200 + P(8 \text{ m sec})$
$= 200 - 200P + 8000000P$
$= 200 + 7999800P$

if one access out of 1000 causes a page fault,

EAT = 8.2 msec

Pagefault —

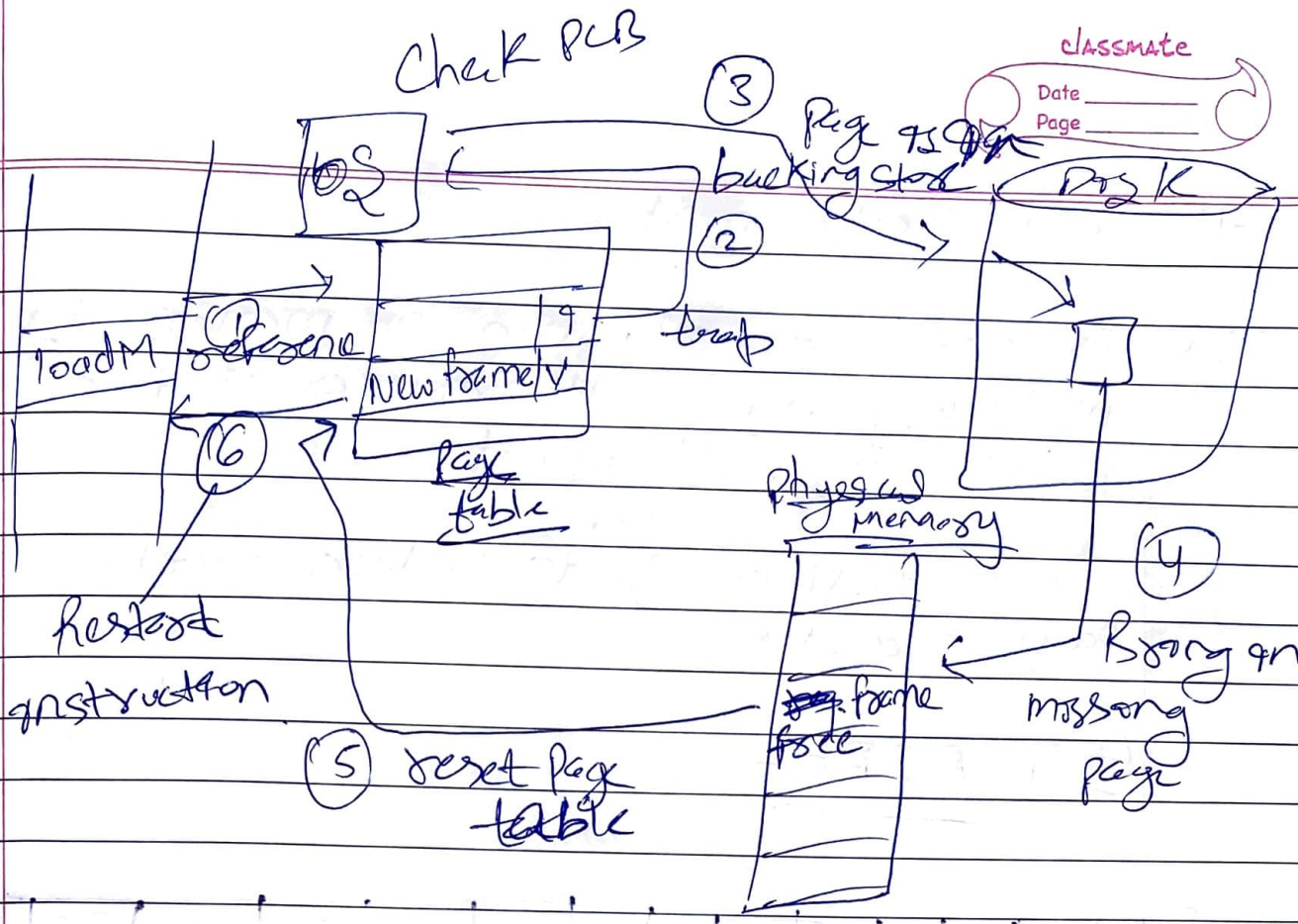
Demanded page is not present on M.M

Page fault can be handled by OS:

1. OS looks at PCB to decide:
   - invalid reference ⇒ abort
   - Just not on memory ⇒ (load the page)
   (how it loads the page)?
2. Get empty frame
3. Swap page into frame via disk operation.
4. Reset page table to indicate page now in memory,

   Set validation bit = V

5. Restart the instruction that caused the page fault.



What happens if there is no free frame?

(1) Page replacement :
find some page in memory, but not really in use, swap it out.

(2) Same page may be brought into memory several times.

Page replacement —

→ Prevent over-allocation of memory by modifying page-fault services routine to include page replacement.

→ use modify (dirty) bit to reduce overhead of page transfer.

(I) FIFO —

f_3			1	1	1	1	0	0	0	3	3	3	3	2	2
f_2		0	0	0	0	3	3	3	2	2	2	2	1	1	1
f_1	7	7	7	2	2	2	2	4	4	4	0	0	0	0	0
	*	*	*	*	HIT	*	*	*	*	*	*	HIT	*	*	HIT

Reference String: 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 1, 2, 0

Page HIT = 3
Page fault = 12
Page miss

HIT ratio = $\frac{\text{No. of hits}}{\text{No. of Reference}}$

$= \frac{3}{15} \times 100 \Rightarrow$ 20%

* Belady's Anomaly on FIFO

f_3		3	3	3	2	2	2	2	2	4	4	
f_2	2	2	2	1	1	1	1	1	3	3	3	
f_1	1	1	1	4	4	4	5	5	5	5	5	
	*	*	*	*	*	*	*	HIT	HIT	*	*	HIT

HIT ⇒ (3)
Page fault ⇒ (9)

Strong → 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

f_4
f_3
f_2
f_1

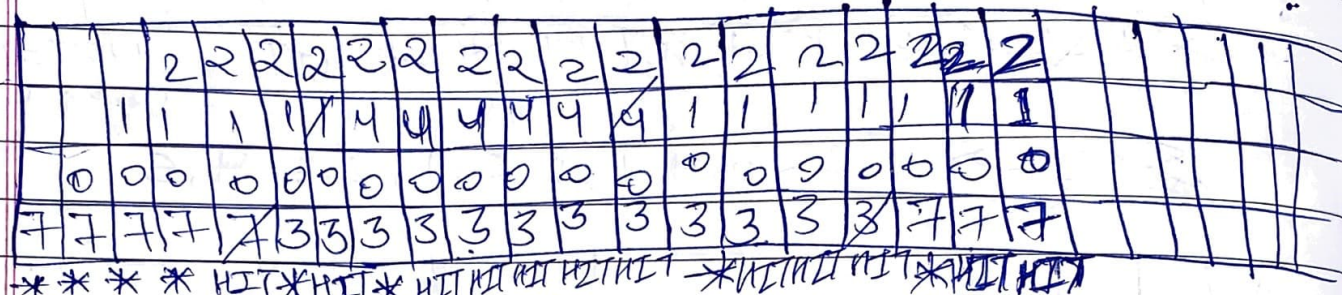
↳ HIT ⇒ 2
Page faults ⇒ 10

This is Belady's anomaly

No. of frames increases so page fault also increases and HIT decreases

Optimal Page replacement —

Replace the page which is not used in longest Dimension of time in future.



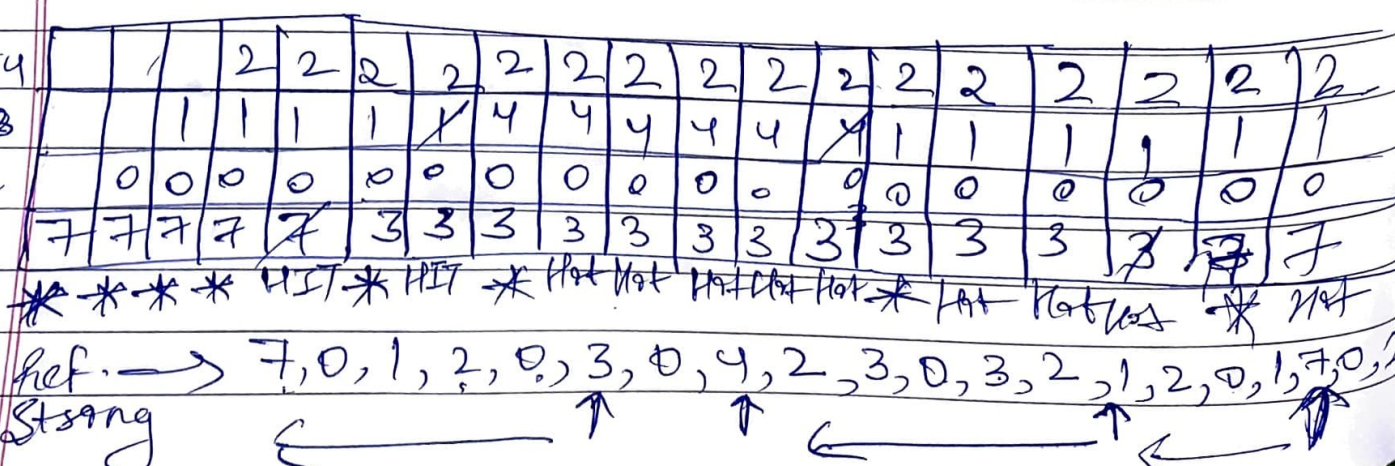
Ref. → 7,0,1,2,0,3,0,4,2,3,0,3,2,1,2,0,1,7,0,1
Strong

Page hit = 12
Page fault = 8

Hit ratio ⇒ $\frac{12}{20} \times 100$

Least Recently used — (LRU)

Replace the least recently used page in Past.

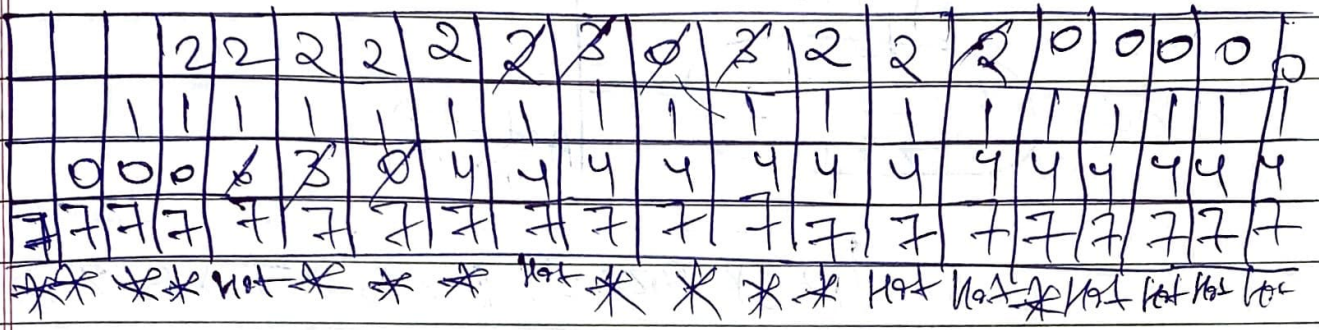


Ref. → 7,0,1,2,0,3,0,4,2,3,0,3,2,1,2,0,1,7,0,1
Strong

Page fault = 8
Page hits = 12

Most recently used —

Replace the most recently used page in past.

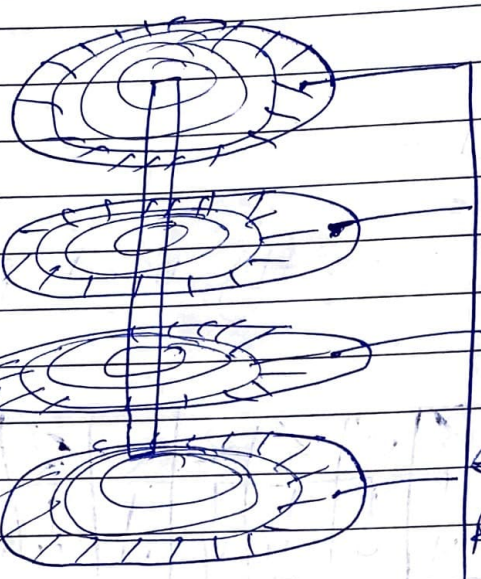


Ref String → 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1,

7, 0, 1

Page fault = 12
Page Hits = 8

Disk Architecture —



Platters → Surface → Track → Sectors → Data

8 R x 256 x 512 x 512 B

Disk size ⇒ P x S x T x S x D

1 K = 2^{10}
1 M = 2^{20}
1 G = 2^{30}
1 T = 2^{40}

Disk Access time —

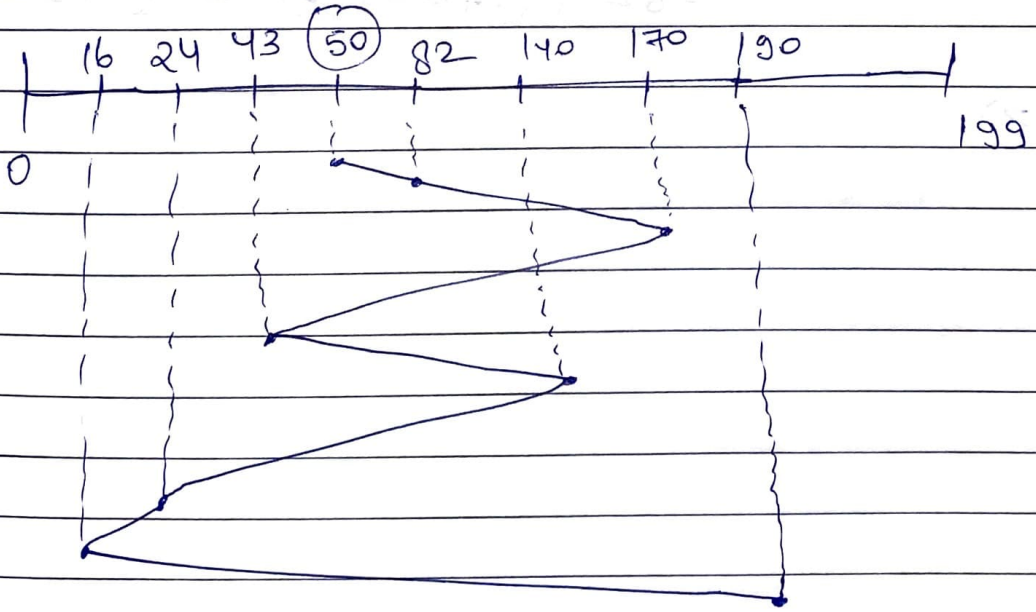
1) Seek time: Time taken by R/W head to reach desired track
2) Rotation time: Time taken for one full rotation (360°)
3) Rotational latency: Time taken to reach to desired sector (half of rotation time)

4) Transfer time: Data to be transfer

Transfer rate: {No. of Heads × Capacity of one Tracks × No. of rotations on one second}

Disk Access time ⇒ ST + RT + TT + QT + CT

FCFS (first come first serve) —



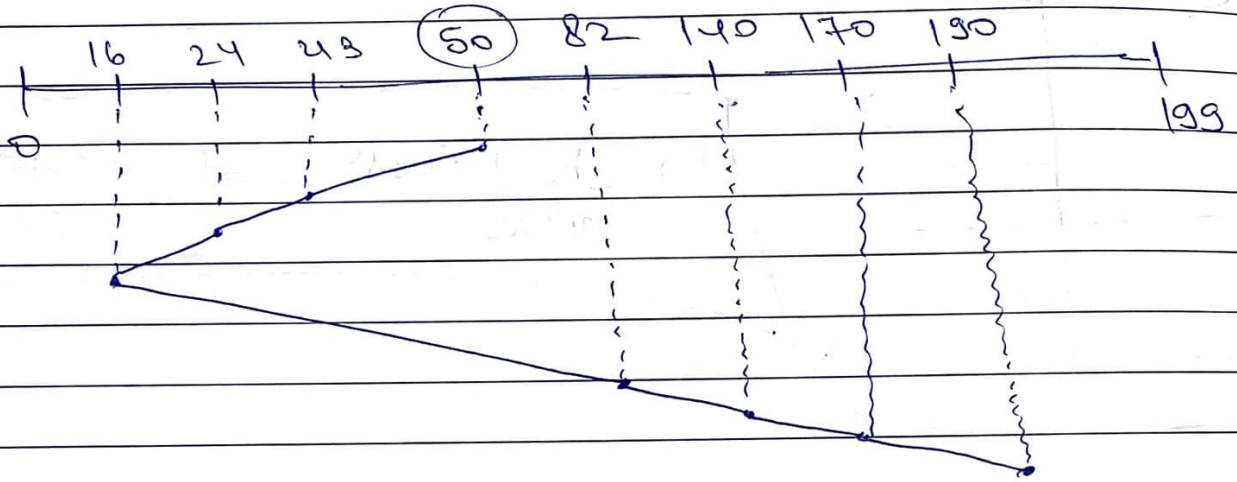
Request track ⇒ 82, 170, 43, 140, 24, 16, 190
current position of R/W head = 50
calculate total no. of tracks movement of head.

⇒ (82-50) + (170-82) + (170-43) ——— + (190-16)

SSTF (Shortest Seek time first) —

Request queue → 82, 170, 43, 140, 24, 16, 190

Curr. pos. of R/W head = 50



⇒ (50-43) + (43-24) + (24-16) + (82-16)
+ (140-82) + (170-140) + (190-170)

⇒

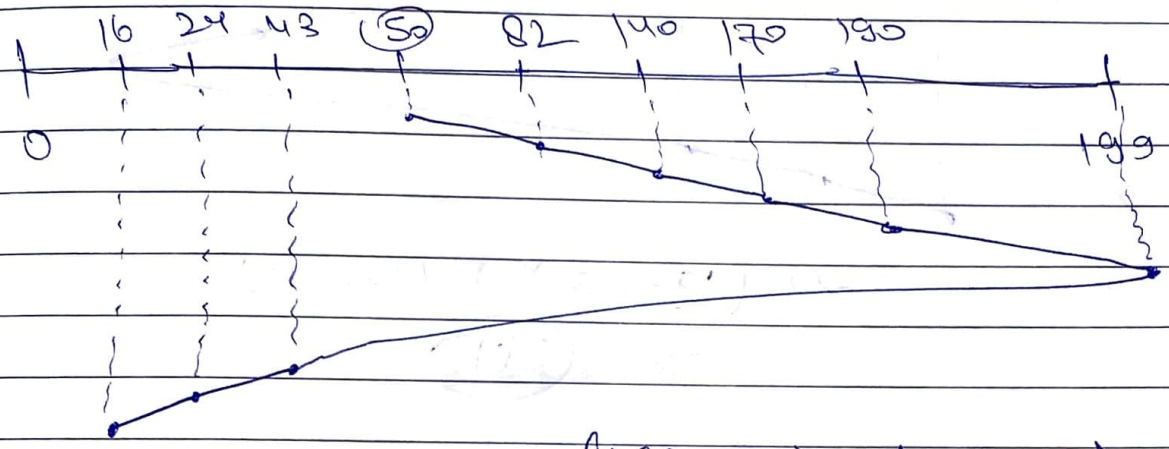
SCAN Algorithm —

Request → 82, 170, 43, 140, 24, 16, 190

Queue

Cur. pos. → 50

move towards high value.



(It do not go at last value)

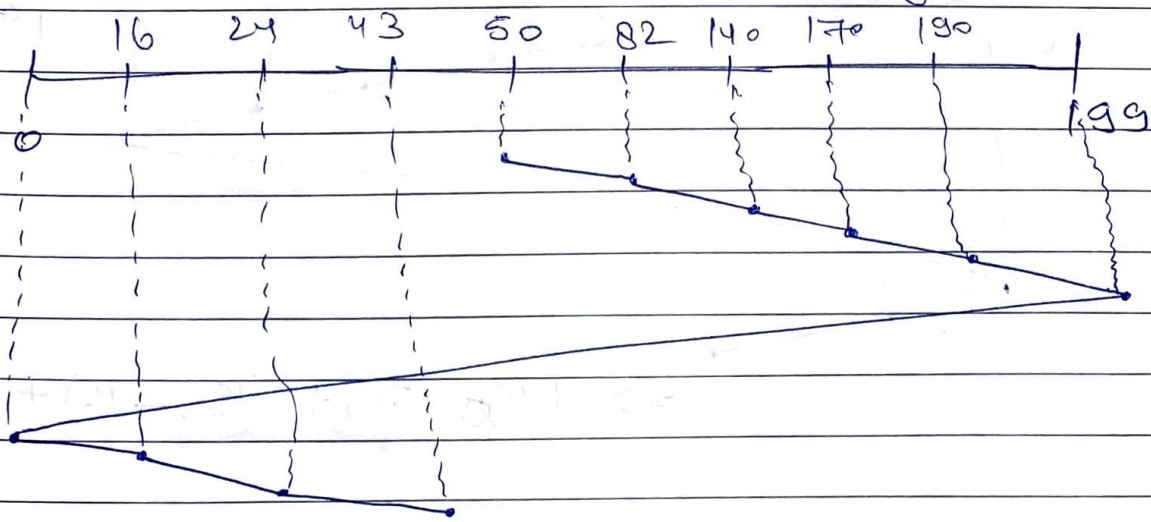
(199-50) + (199-16)

= 332

Q. If R/w head takes 1 ns to move from one track to another then total time taken = 332 x 1 ns

C-SCAN Algorithm —

Direction towards larger value.

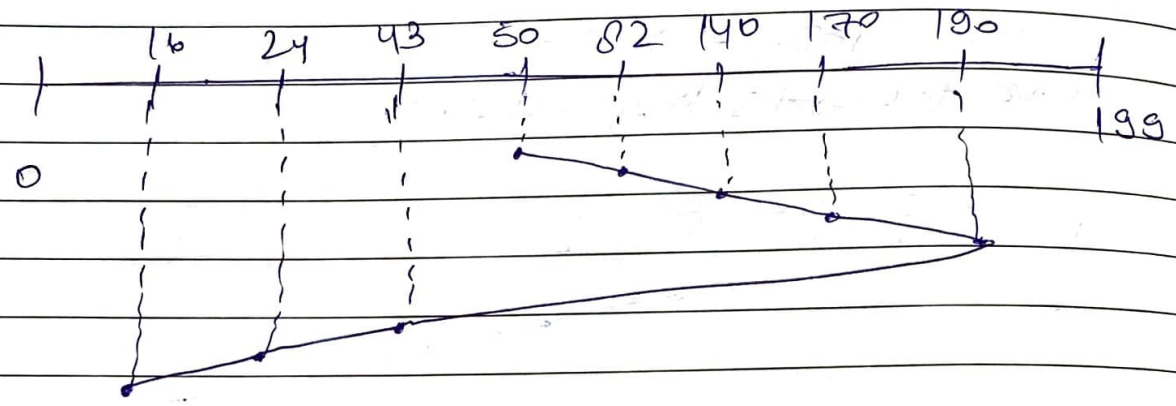


(199-50) + (199-0) + (43-0)

→ 391

LOOK Algorithm —

Direction is towards larger value.

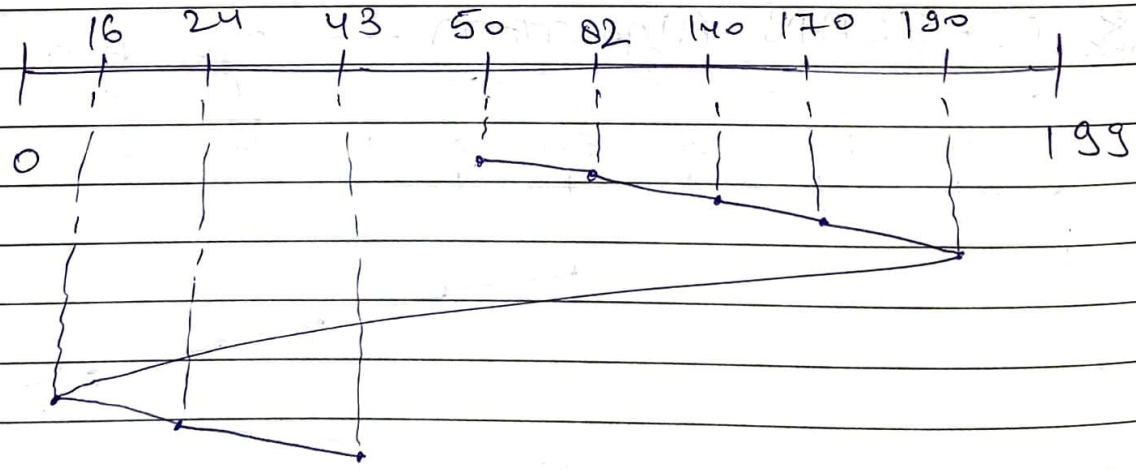


⇒ (190−50) + (190−16)

⇒ 314

C-LOOK Algorithm —

Direction is towards large value



⇒ (190−50) + (190−16) + (43−16)

RAID (Redundant Array of Independent Disks)

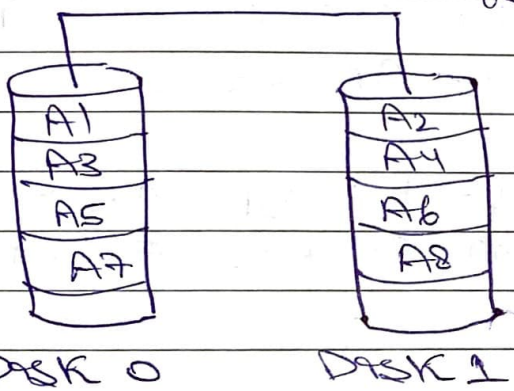
→ Is a way of storing the same data in different places to protect data in case of a drive failure.

RAID 0

(data striping)

(data ko todte hain)

(performance sabse zyada)



RAID - 1

(Data mirroring)

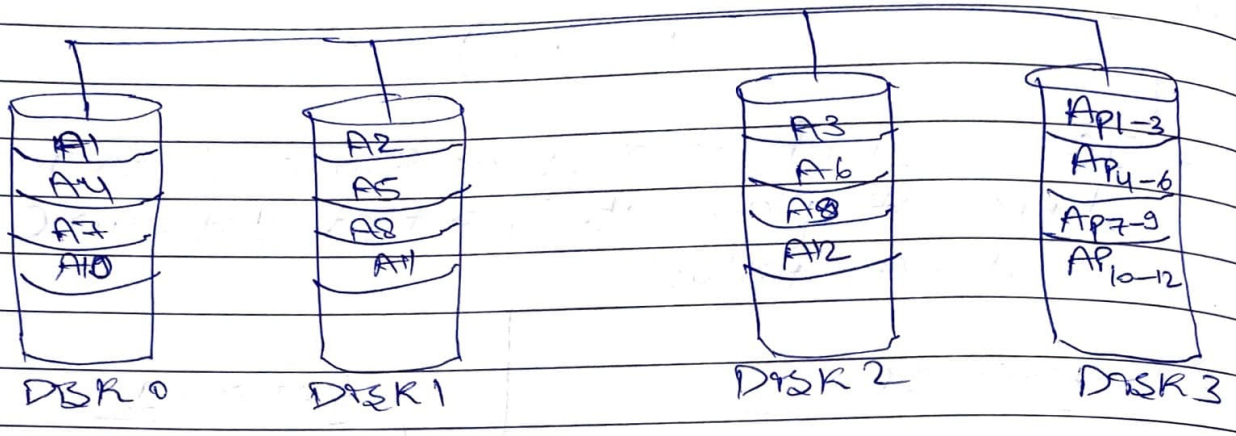
(data ka copy)

(Data ko secure karna)

RAID 1+0

(Data mirroring + Data Striping)

RAID-3 (Ap = Parity)



Ex- $A1 = 1$, $A2 = 2$, $A3 = 3$

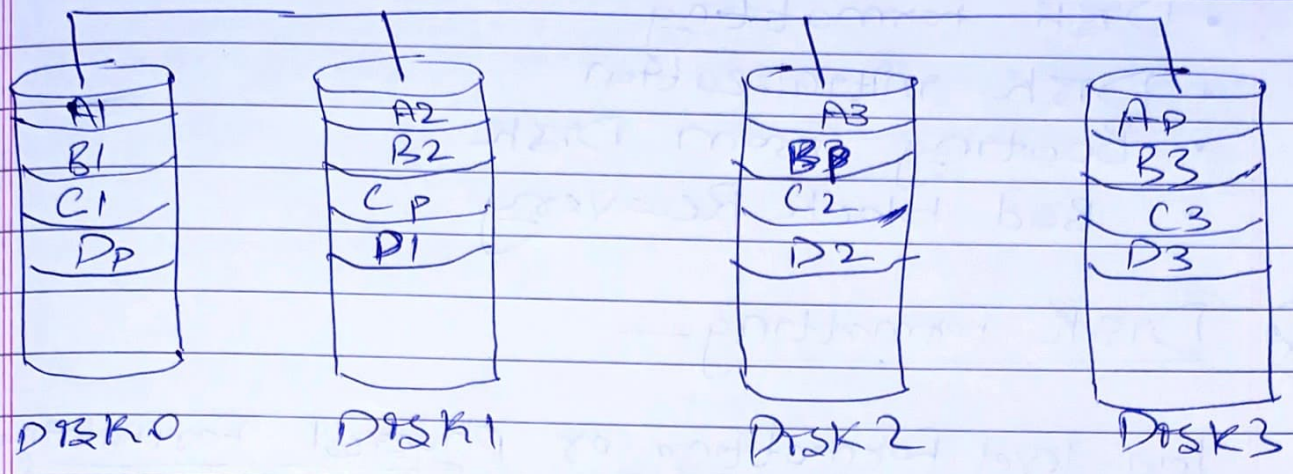
So $Ap_{(1-3)} = 6$ [1+2+3]

So if A1 is lost we can calculate it by using parity $Ap_{(1-3)}$.

⇒ Here DISK 3 will be used very much so it will cause a problem so RAID 5 will solve the problem.

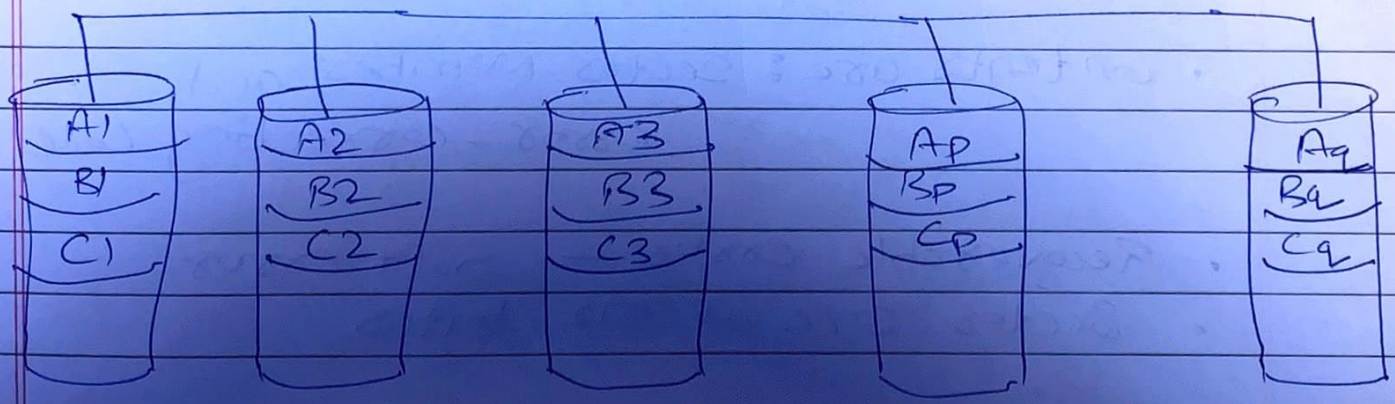
RAID-4 (Same as RAID-3)

RAID - 5



So ab koi ek DISK over utilize nhi hogi ab saari disk equally utilize hogi

RAID - 6



Disk Management —

- Disk formatting
- Disk initialization
- Booting from Disk
- Bad block Recovery

⊛ Disk Formatting —

low-level Formatting or Physical Formatting:
Before a disk can store data it must be divided into sectors.

low-level Formatting → contains data structure for each sector → This data structure contains a header & trailer → These are used by disk controller

- contents are: Sector Number and error-correcting code.

- Recoverable error — Soft error
- Sector size of 512 bytes

- Before storing files, OS, two steps
  - partitioning of the Disk
  - logical Formatting (creation of File system FAT)

✱ Boot Block —

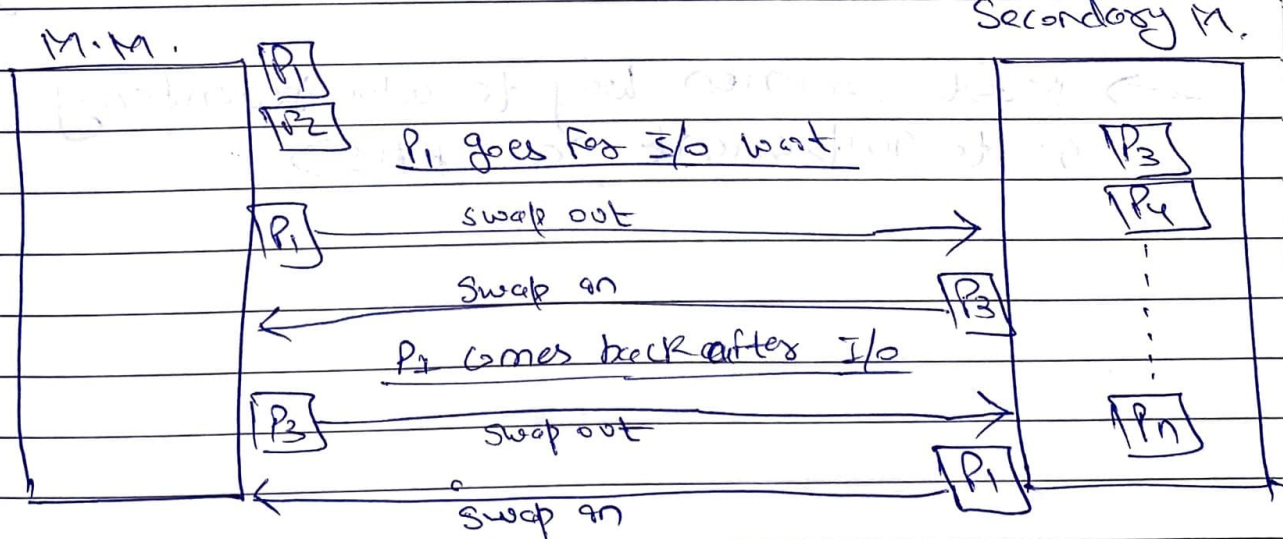
Blocks where the full Bootstrap program is stored on the disk.

✱ Bad blocks —

- If blocks go bad during normal operation, then some special programs must be run manually to search for bad blocks.

✱ Swap-Space Management —

A process can be swapped temporarily out of M.M to secondary memory later again swaps back to M.M.



- Swap- Swap management is another low-level task of O.S.
- Goal of Swap-Space implemen. is to provide best throughput.
- If we use disk space, it will decrease system performance.

(*) Disk Reliability-

- Reliability -

Ability of disk system to accomodate a single or multiple failure and still remain available to others.

→ Adding redundancy ⇒ ↑ reliability of disk system.

→ Most common way to add redundancy is to implement a RAID.