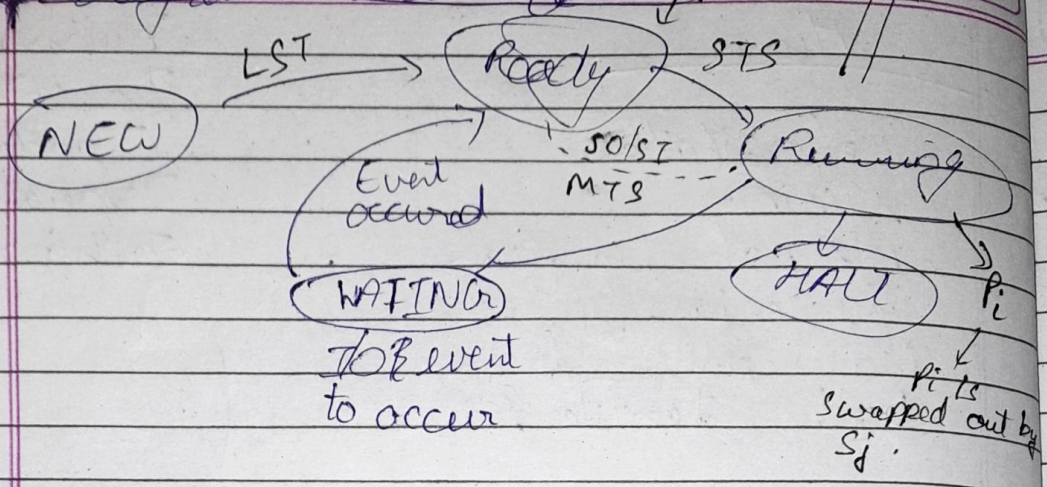
SO: Swapped out
SI: Swapped In

Context Switching

PAGE NO.:
DATE: / /

Program execution



New state has a queue of initialed program, so is stored in secondary storage

* Swap out & Swap in:
If some program that was being executed, gets pushed out by a higher priority program; it is said to have been swapped out, and the higher prior prog is said to have been swapped in.

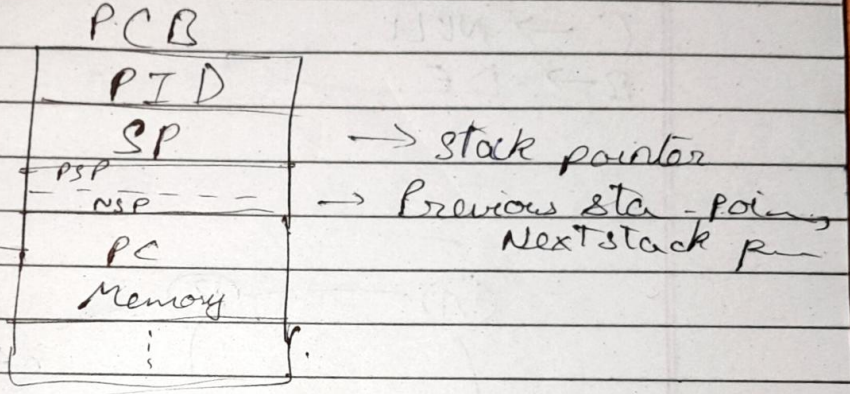
Scheduling:
Preemptive
Non-Preemptive.

Long term Scheduler
Short term Scheduler
Mid term Scheduler

* The OS assigns PID (process IDs) to all the processes to uniquely

identify them.

A PCB (Program Control Block) stores the state of execution of the process. It is unique for each process, and has same PID as the process.



Q switching from one process to other is done in the Running state and is called context switching. (swap in & swap out is context switching)

Non-preemptive: If a process is running, MTS will not stop it in between the exec to push another program

Preemptive: MTS can swap out a running program.

Process	Time	Priority
P_1	10	1
P_2	20	2
P_3	15	2
P_4	5	2
P_5	10	1

OS

PAGE NO.:
DATE: / /

CPU Scheduling
- FCFS
- SJF / SJNF : (shortest Job first / shortest job next)
- priority Scheduling
- SRTF : (shortest remaing time first)
- Round Robin (RR)
- Multilevel Queue Schedu (MLQ)
- " Feedback " (MLFQ)

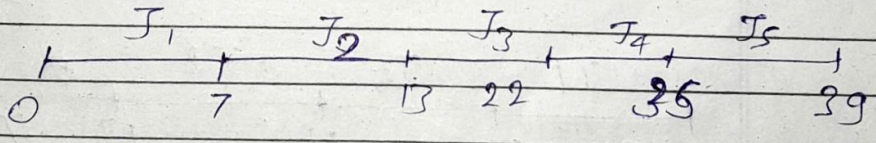
Goals: (of CPU schedu)
- Maximise CPU utilization, Throughput
- Minimize AWT, WTAT.
  (Average waiti.t) (Turnaround time)

FCFS

Eg: RQ → {J1, J2, J3, J4, J5}
⇒ Estimated Execution time (Ji, Pi)

Arrival time.	AT	EET
J1	0	7
J2	1	6
J3	1	9
J4	2	13
J5	5	4

GC:



Then total time taken =>
Max (completion time (T_i)), i=1,2--n

WT (J_1) = 0

J_2 came to RQ at the instance t = 1s. and J_1 takes 7 s of execution

So T_2 waited for $7-1=6s$

$\therefore WT(T_2) = 6$

Similarly

$WT(T_3) = 18-1 = 12$

$WT(T_4) = 22-2 = 20$

$WT(T_5) = 35-5 = 30$

$\therefore AWT = \frac{0+6+12+20+30}{5} = 13.6$

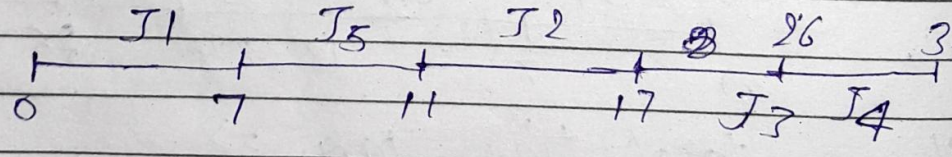
$ATAT = \sum (\text{Time of end} - \text{Time of initiation})$

$= \frac{(7-0)+(13-1)+(22-8)+(35-2)+(39-5)}{5}$

$= 21.4$

SJF / SJN

We need to make sure that the CPU is not idle. So at $t=0$ T_1 is executed.



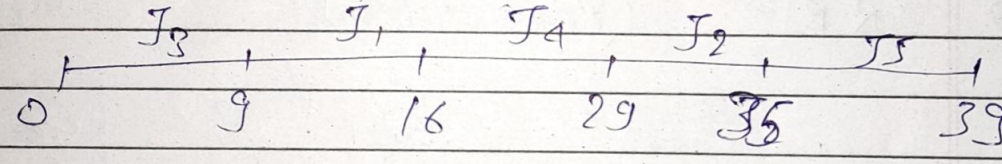
$AWT = \frac{1}{5}\left(\underset{T_1}{0} + \underset{T_5}{(7-5)} + \underset{T_2}{(11-1)} + \underset{T_3}{(17-1)} + \underset{T_4}{(26-22)}\right) = 10.4$

ATAT = 18.2

Priority Scheduling

	AT	EET	PV
J_1	0	7	2
J_2	1	6	3
J_3	0	9	1
J_4	2	13	2
J_5	5	4	9

Let's say the scheduler looks at the Ready Queue (RQ) in batches. So, the Gr. chart is:



$$AWT = \frac{1}{5}\left(\underset{J3}{0} + \underset{J1}{9} + \underset{J4}{(16-2)} + (29-1) + \underset{J2}{(35-5)} + \underset{J5}{}\right)$$

$$= 16.2$$

* Starvation problem: Higher priority process get executed, overlooking the lower priority jobs.

To solve this, after the successful execution of the job, the priority value of each job will be incremented by 1.

The system ~~only~~ will also set a threshold priority value (say 10 in above case). Once a job reaches this value, it will be give prior of 1 in next job, and will be given the CPU in the next iteration.
This is called ageing.

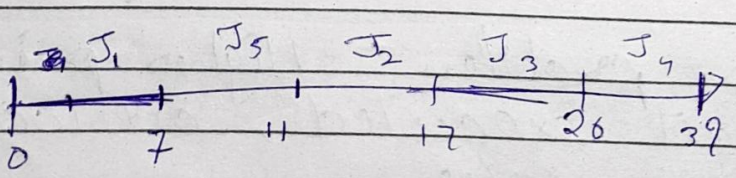
Preemptive Scheduling –

1. SRTF (Shortest Remaining ~~Task~~ Time first)
2. RR (Round Robin)

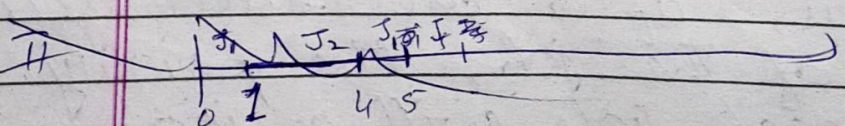
(1)

	AT	EET	Case II
J_1	0	7	→ 6 → 5
J_2	1	6	→ ~~6~~ 4 ✓
J_3	1	9	→ 9
J_4	2	13	→ 6 ✓
J_5	5	~~4~~ 4	→ ~~3~~ ✓

Case I

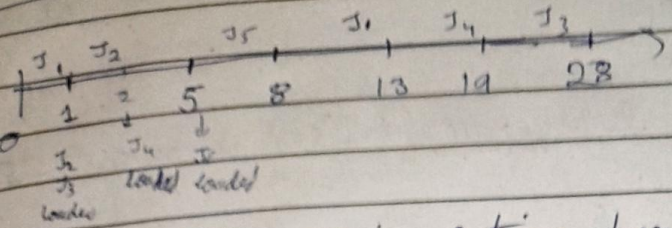


Case II



PAGE NO.:
DATE: / /

Case II



SRTF → After each time period if a new job is loaded, then the program with lowest time remaining (in execution) will be done.

$WT(J_1) = 7$ { loaded = 0, finish = 13, running = 6 | 13-6=7 }

only 1 content switching → (when one job is stopped for other)

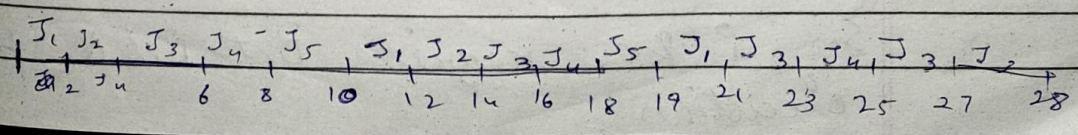
$WT(J_2) = 0$, $WT(J_3) = 18$
$WT(J_4) = 11$, $WT(J_5) = 0$

$WT_{avg} = \frac{7+0+18+11+0}{5} = \frac{36}{5} = 7.2$

(2) RR
Round Robin.

Time quanta / time slice. → (10 - 100) ms

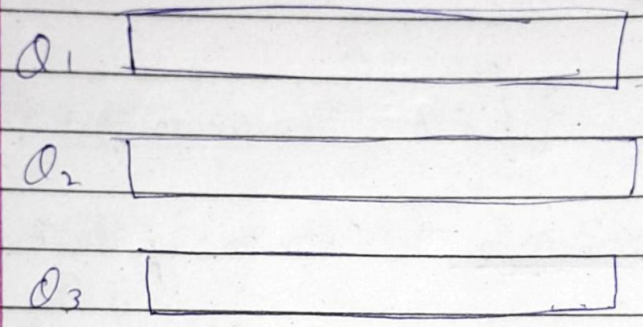
Same example (case II) → with time slice = 2.



RR → removes starvation but too many content switching. Also WT is a lot

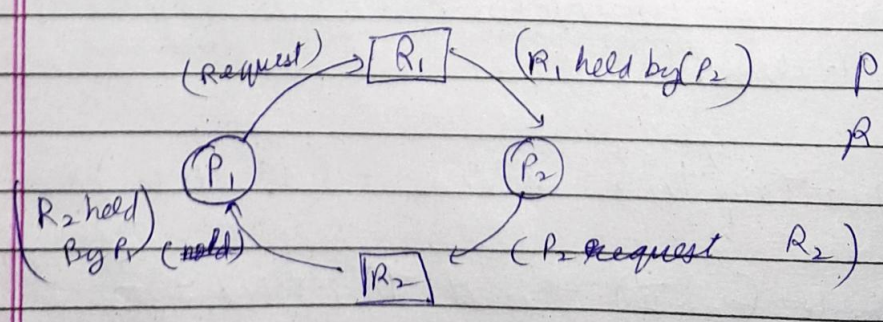
Effective & Usefull. (Used in most OS)

MLQ (Multilevel Queue)



{Based on some cond" jobs are categorised in diff queues. And diff queues are using diff scheduling techs}

Process Syncronization & Communication



Say, P1 & P2 require Both R1 & R2 for execution

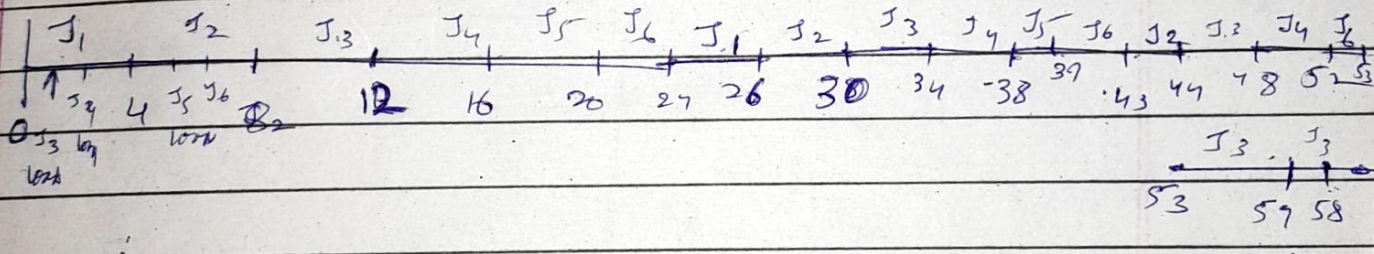
PAGE NO.:
DATE: / /

Programs are in deadlock ..

	AT	CPU	Priority
J_1	0	6	3
J_2	0	9	2
J_3	1	17	4
J_4	2	12	1
J_5	5	5	2
J_6	6	9	5

Implement RR (t=4), Priority, SRTF.

RR

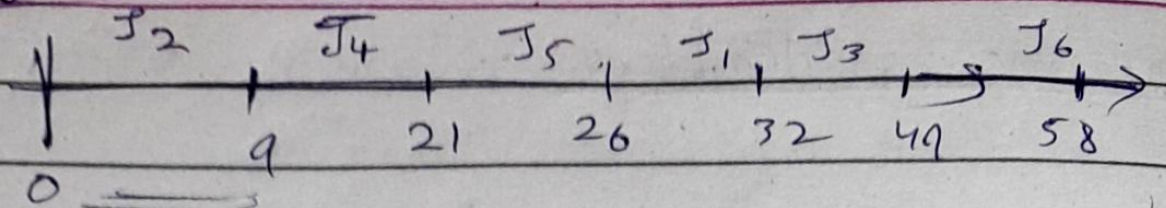


WT(J_1) = 20, WT(J_2) = 35, WT(J_3) = 40, WT(J_4) = 38
WT(J_5) = 29, WT(J_6) = 38

Avg WT = 33.3 Switches = 14

TAT(J_1) = 26, TAT(J_2) = 44, TAT(J_3) = 57
TAT(J_4) = 50, TAT(J_5) = 34, TAT(J_6) = 53 - 6 = 47

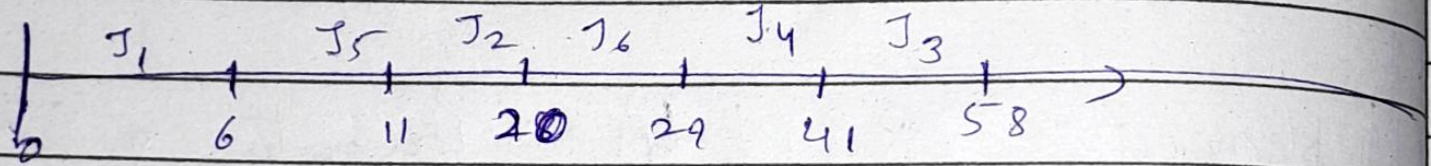
Priority



$WT(J_1) = 26$, $WT(J_2) = 0$, $WT(J_3) = 31$
$WT(J_4) = 7$, $WT(J_5) = 16$, $WT(J_6) = 43$

Avg WT = 20.5

SRTF

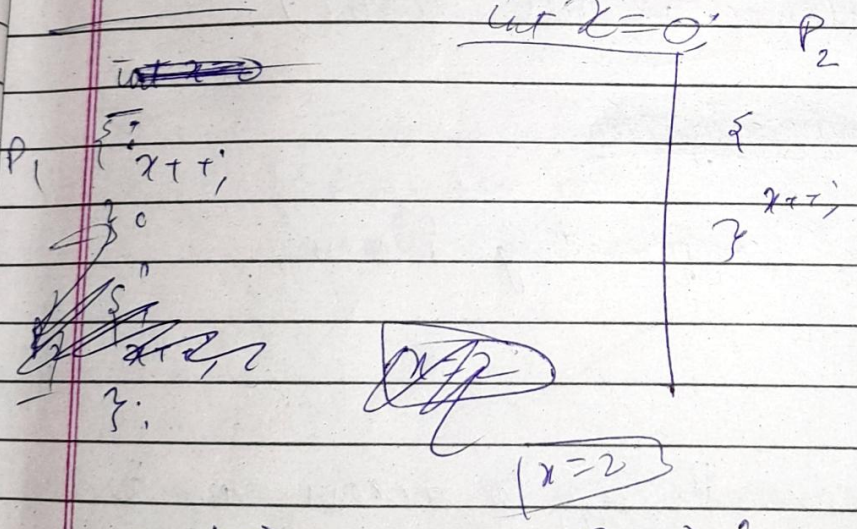


$$\text{Avg TAT} = \frac{6 + 9 + 20 + 20 + 39 + 57}{6}$$

Process Synchronization

- Independent Process
- ~~Depen~~ Cooperating process

- Race condition
- Critical Section
  - Mutual Exclusion
  - Progress
  - Bounded Waiting
- Mutex lock
- Same place
  - Binary
- Semaphore
  - Binary
  - Continous



x++; ⇒

```
LW    R1, x;
ADD   R1, R1, 1;
SW    x, R1
```

Assume P_1 loads x to R_1 & then the process finds P_2 must execute first,

P_1	P_2
LW R_1, X	
context switch	
	LW R_2, X
	ADD R_2, R_2, 1
	SW X, R_1
	context switch
ADD R_1, R_1, 1	
SW X, R_1	
x = 1	x = 1

- This is not the right result.
(bank balance is not right)

Lack of synchronization

- P_1 & P_2 are cooperating Process

Race Condn

No. of process trying to access same section of code / memory.

Such sections are called critical section

do {
entry section
critical section
exit section
remainder section
} while (True)

give the CS to other program only when one is ~~first~~ finished accessing CS

→ Mutual Exclusion → Programs accessing CS must be mutually exclusive

→ Bounded Waiting → It should not be the case that one program is waiting for ∞ time.

Mutex lock

access CS → lock {flag = 1} → Completion ← lock {flag = 0} ← other program can access CS

15/2/24

Deadlock

Necessary & Sufficient Cond^n

→ Not Mutual Exclusion
→ Hold & Wait
→ No - Preemption
→ Circular wait

Deadlock Sol^n

→ Deadlock Prevention
→ " Avoidance
→ " Detection & Recovery

→ (Until one task is fulfilled the resource must not be allocated to other program)

Request
Use
Release

Resource Allocation Graph

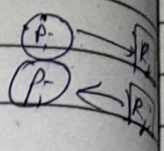
G(V, E)

Edges

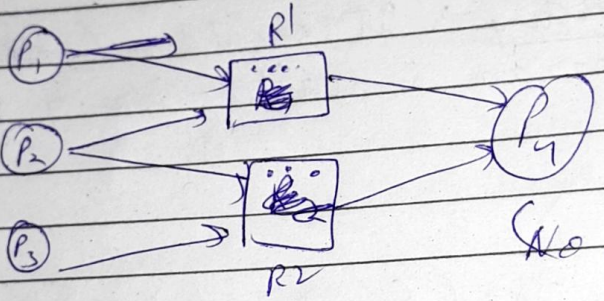
- Process → P_i
- Resource → R_i

→ Requesting

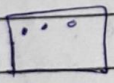
→ Allocation

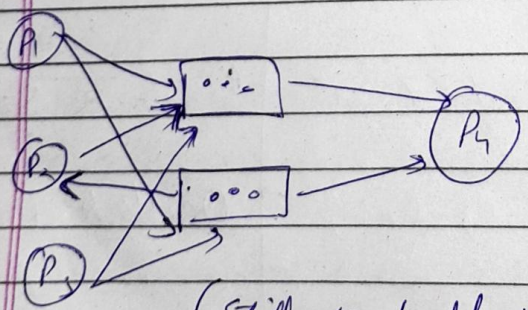


- diff types of resources
- Each r_i has 'm' no. of identical Resources

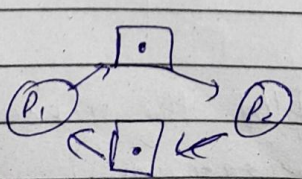


(No deadlock)

 → (...) dots represent no. of identical instances / resources



(Still no deadlock)



This is deadlock

If there is a cycle in graph & all the resources have 1 instance then deadlock will occur

In case of multiple instance → may or may not

$A = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$.

$\boxed{\min \text{card}(S)}$ if $S \subseteq A$, such that $a+b=9$ for some $a, b \in S$.

minimum cardinality of S.

There exists $a, b \in S$, such that $a+b=9$

So we select all the odd numbers.

$\therefore S = \{1, 3, 5, 7, 9, \_\_\_\}$

The last remaining number must be some even no., which on sum, gives 9.

$\therefore$ min cardin $= \underline{6}$

Difference of a polynomial

nth difference of an nth degree poln is constant and (n+1)th diff is 0

$$f(x) = a_0 x^n + a_1 x^{n-1} + \cdots a_{n-1} x + a_n$$

$$\Delta f(x) = f(x+h) - f(x)$$

$$= a_0\left((x+h)^n - x^n\right) + a_1\left((x+h)^{n-1} - x^{n-1}\right) + \cdots a_n - a_n$$

Deadlock Avoidance
Banker's Algorithm

n - processes at some time
m - resource types

To handle all the resources, we use a vector.

Available m : Ava [i] = k

↑
k resources available for ith process

$Max^{*}_{n \times m}$: Max [i, j] = k

k no. of jth resources instances are required for successful execution of ith process.

$Allocation_{n \times m}$: Allo [i, j] = k instances

k resources of j type allocated to P_i

$Needed_{n \times m}$: Need [i, j] = k

k resource instances of j type are still required by P_i

Request $_i$: stores the no. of resources required by P_i

PAGE NO.:
DATE: / /

The requested amount of resource must be less than equal to Need.

① If $Request_i \leq Need_i$, then goto step ②, else error
② If $Request_i \leq Available$ then goto step ③, else wait.

③ $Avail = Avail - Request_i$;
$Allocate_i = Alloc_i + Request_i$;
$Need = Need - Request$.

④ Check if the new state is safe or not. This is done using a safety algorithm.

Safety algo:

① $Work_m = Available$;
Finish = False;

② Find an i such that
Finish[i] = F and $Need_i \leq Work_i$
if no such i is found goto step 4

③ Work + = ~~Available~~ Allocated.
Finish[i] = T;
goto step 2

④ Finish[i] = True $\forall i$
Safe state of system

Qus

A	B	C
10	5	7

(three resource types)

No. of processes: 5 ; {P_0, ---P_4}

Allocation $_{5\times3}$:

	A	B	C
P_0	0	1	0
P_1	2	0	0
P_2	3	0	2
P_3	2	1	1
P_4	0	0	2

Max $_{5\times3}$:

	A	B	C
P_0	7	5	3
P_1	3	2	2
P_2	9	0	2
P_3	2	2	2
P_4	4	3	3

Available :

A	B	C
3	3	2

[Available = [10 5 7] - np. sum (Allocation, axis=0)
columns

Need $_{5\times3}$:

	A	B	C
P_0	7	4	3
P_1	1	2	2
P_2	6	0	0
P_3	0	1	1
P_4	4	3	1

(Need = Max - Al

DATE: / /

Now we use safety Algo:

Work: [3 3 2]
Finish = [False] × 5

we find P_i which can be finished with rema resources. i=1 is found.

∴ Finish[1] = T
$Work_i$ = Work + Allo$_1$
⇒ [3 3 2] + [2 0 0]
= [5 3 2]

Next i=3
P_3: F[3] = T
work += Allo$_3$
= [7 4 3]

P_4: F[4] = T
work += Allo$_4$
= [7 4 5]

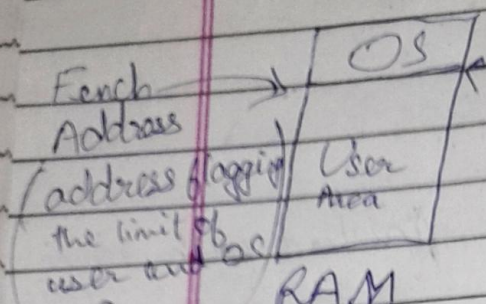
P_0: F[0] = T
work += Allo$_0$
= [7 5 5]

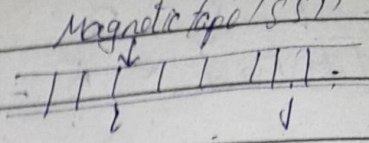
P_2: F[2] = T
work += Allo$_2$
= [10 5 7]

∴ The safe sequence is: P_1, P_3, P_4, P_0, P_2

22/09/24

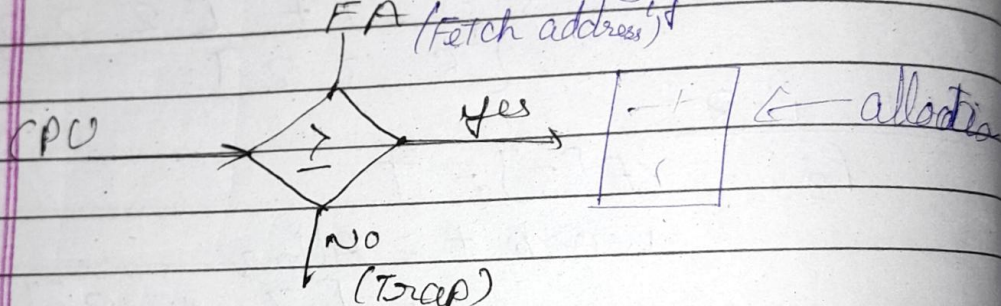
Memory Management





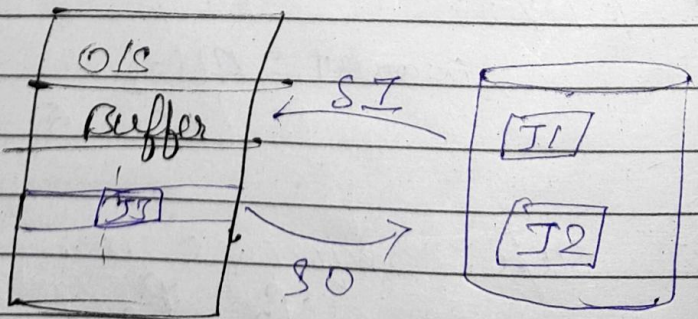
to go from i to j th memory block in MT and S_{ij} : No. of blocks (cost

but in case of RAM $S_{ij} \simeq 0$



(If the requested add is less than Fench address)

Swap in / swap out:

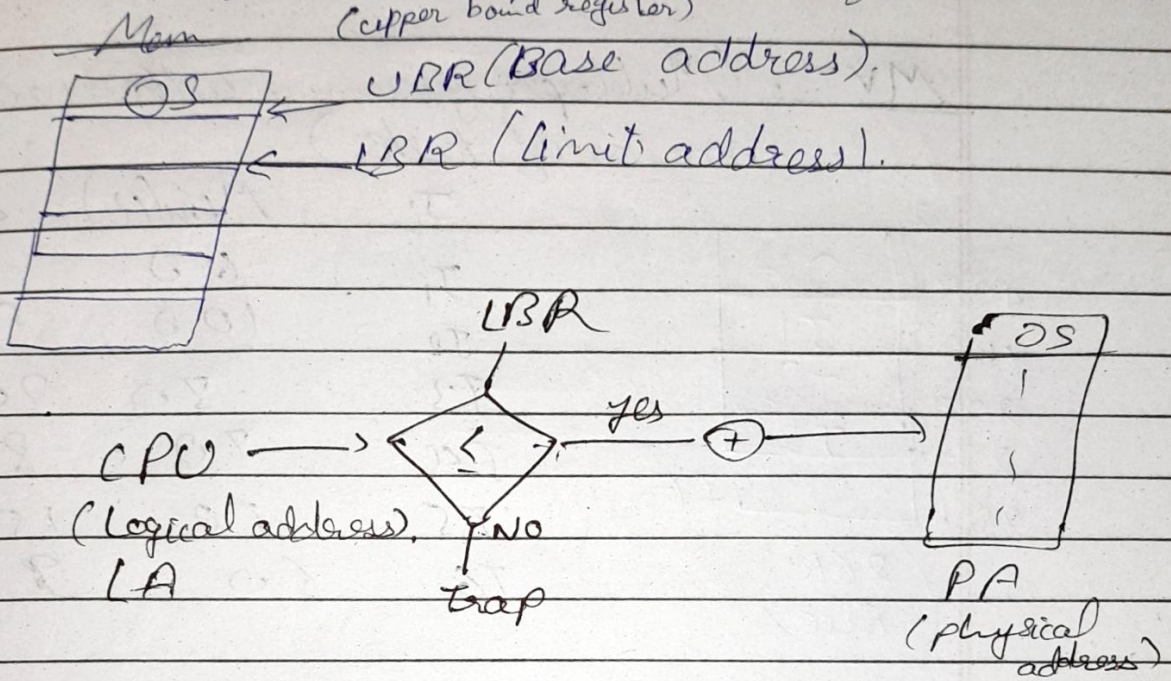


Instead of swapping in/out of the primary memory and secondary storage

The buffer is used in the RAM for this purpose, which is faster.

MFT: (multiprogramming with fixed no of tasks)

The no. of tasks or processes is fixed.



memory allocation can be of three types

① First fit: allocate the address wherever space enough is found

② Best fit: allocate with least waste

③ Worst fit: allocate in the largest block.

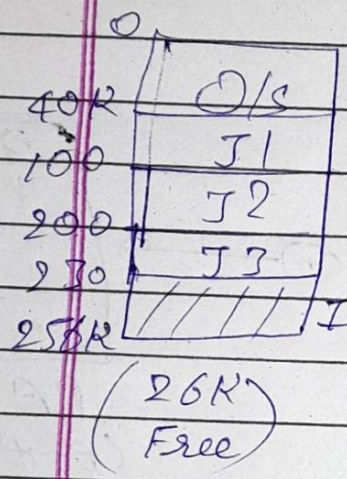
Internal fragmentation: empty or wasted space inside the allocated space.

External fragmentation: Fragmented space is available but can't be used as it is not in a single block, so the OS can't allocate it to some other job.

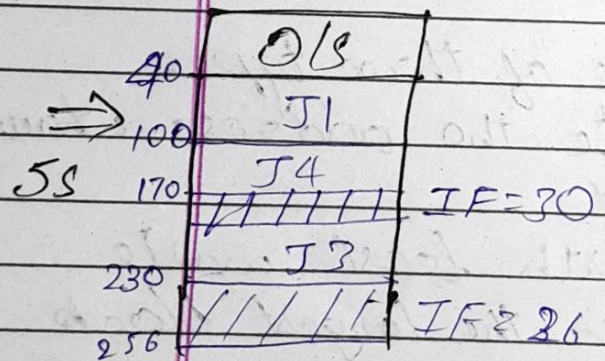
$$\sum_{i=1} IF_i \geq \text{New Job}$$

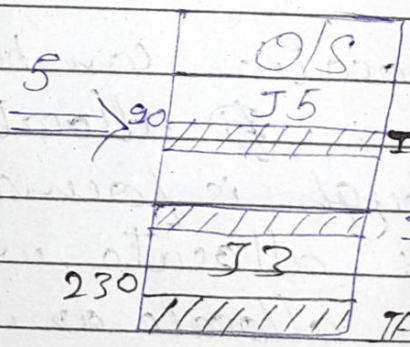
This unused space can't be allocated and leads to External fragmentation.

MVT: (Multiprogramming with variable number of tasks)

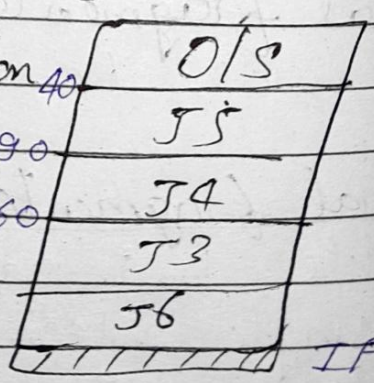


J_i	Mem(i)	ΔT_i
J_1	60	10
J_2	100	5
J_3	30	20
J_4	70	8
J_5	50	15
J_6	60	9



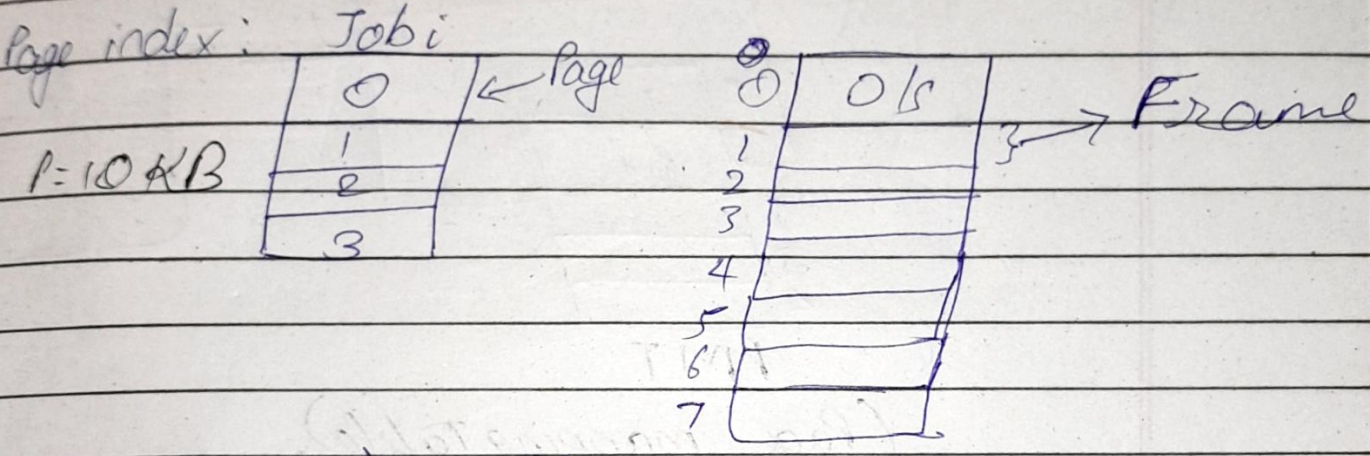


Compaction



MVT results in lots of internal fragmentation. This can be avoided/ rectified using a technique called Compaction. Where all the unused (fragmented) memory is stored in one block.

Paged Memory management



$\forall i, j, \; s(P_i) = s(b_j)$

also any $P_i \longrightarrow b_j$ iff $b_j = 0$

P = page size.

$L = p + d$

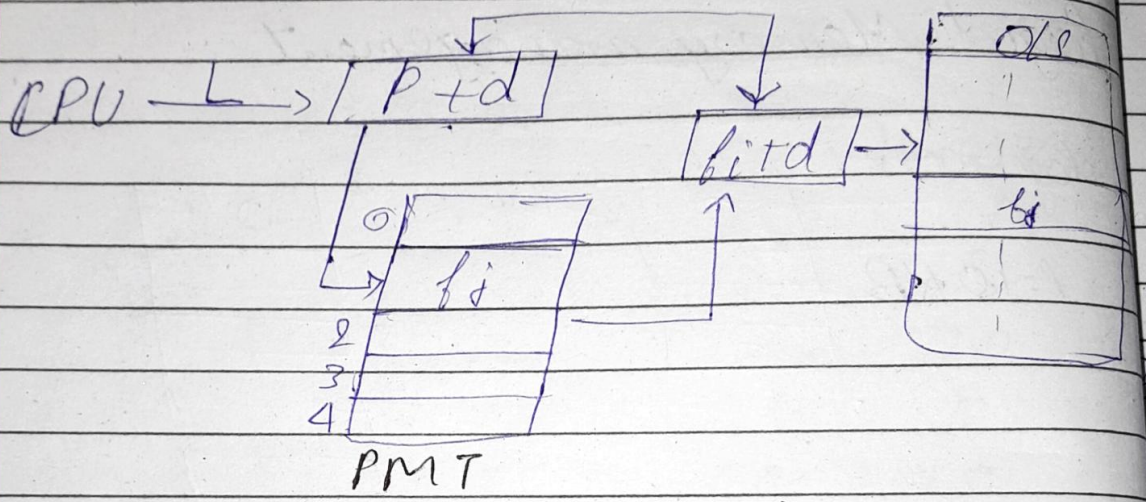
- L: logical address
- p: page no.
- d: offset with the page (offset as the space in the page)

page no. $p = L / P$

$d = L \% P$

* Any page can be fitted to any frame so we need to find some mapping from pages to frames.

The CPU generates logical address which is given by p+d



(Page mapping Table).

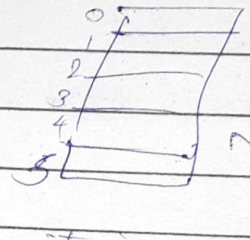
* (In MFT & MVT the memory was allocated sequentially, but in Paging, it is not necessarily sequential)

Memory Management

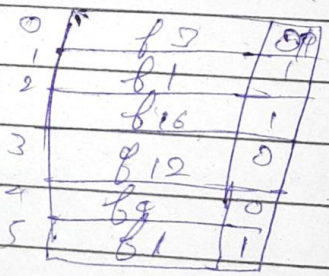
- MFT (continuous m...)
- MVT
- Paging Memory management (Non-continuous memory allocation)
- Segmented
- Page ...
- Demand/virtual

Page map table

PMT:



it is mapped to frames given in PMT



The system calculates the (valid/invalid entry) probability of some page being used in near future called locality of reference.

V/I: → 0 implies the page is available in main memory. So if the system needs that page in future the page is checked for validity

and then accessed.
We also need to check if the page (say P2) has been updated in main memory. If it has, the change must be reflected in secondary memory.

Say the memory is filled with pages, and the CPU needs a page that is not in RAM. So we need to select a page in the main memory and replace it with the required page.

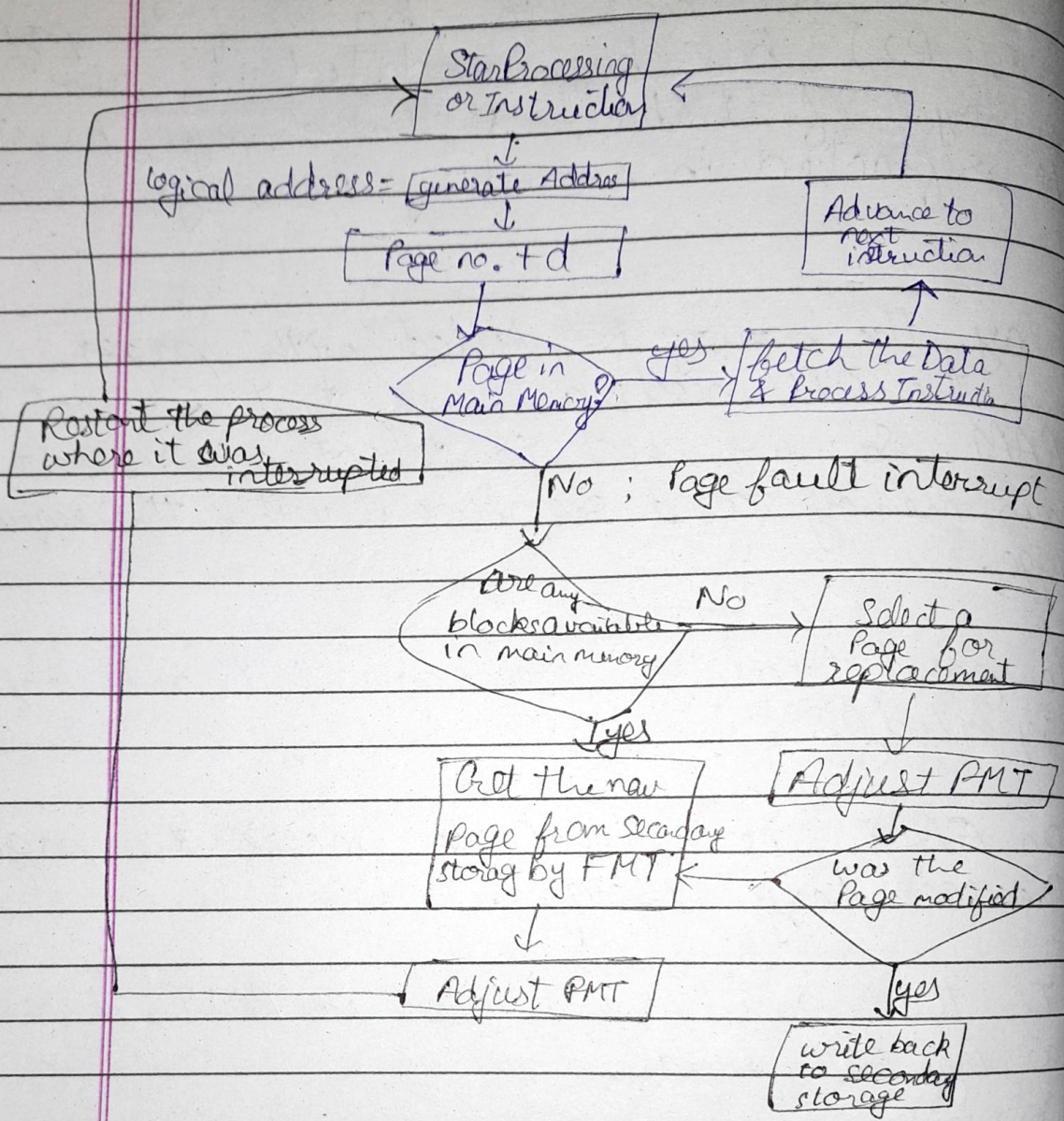
Temporal locality: for pages that are looped repeatedly.

an PMT is used to access the page in the secondary storage.

If the page is not listed in PMT, it throws a Page Fault error.

PMT is stored in the CPU cache (L1/L2)

Master's Flowchart



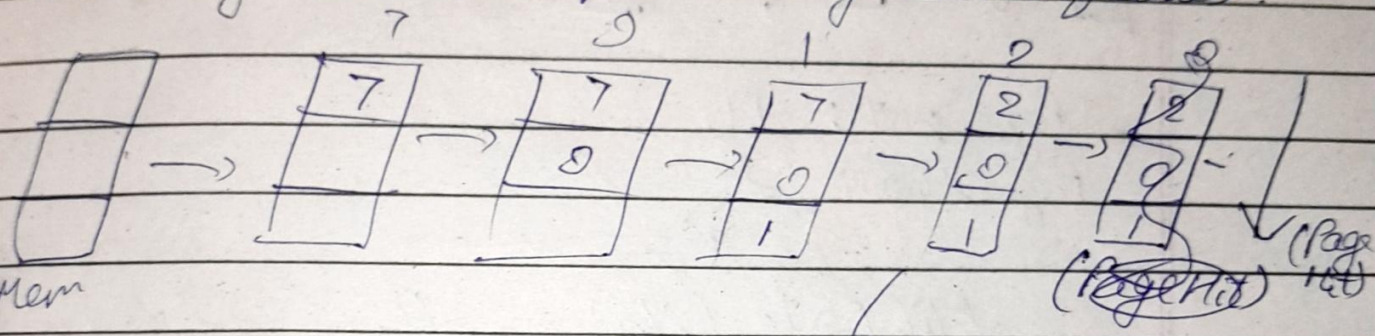
If a page is updated, the change stays in the main memory with the page. This change is reflected in secondary storage only when it is replaced from main memory.
To select a page we use:
① FIFO ② optimal algo
③ LRU

Page Replacement Algorithm

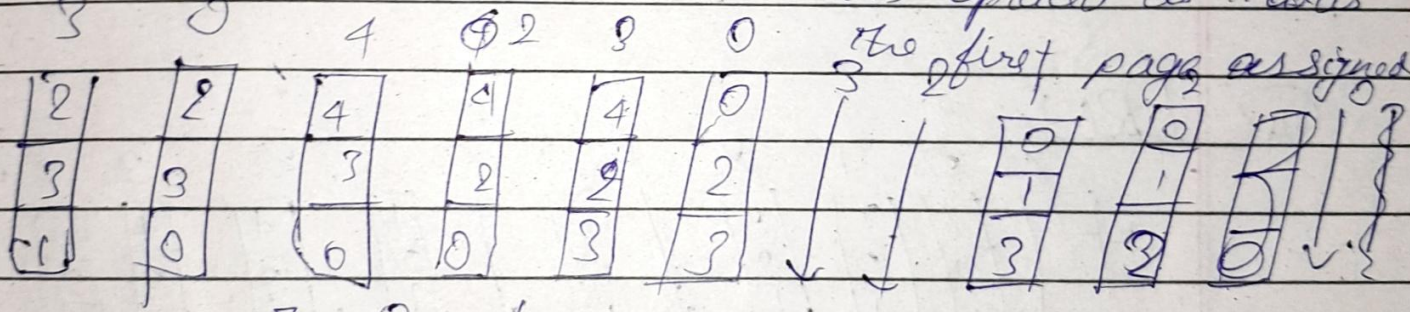
1. First come First serve
2. Optimal
3. Least Recent Used (LRU)

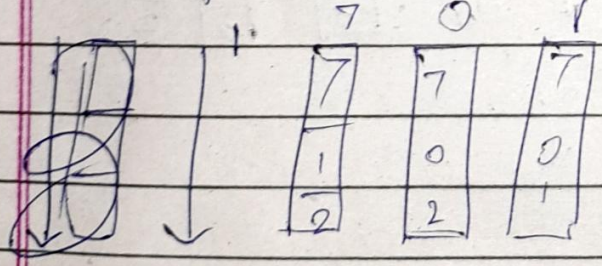
Ex: 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1 (Page numbers)

Let's say that our memory has 3 frames.



7 is replaced as it was the first page assigned





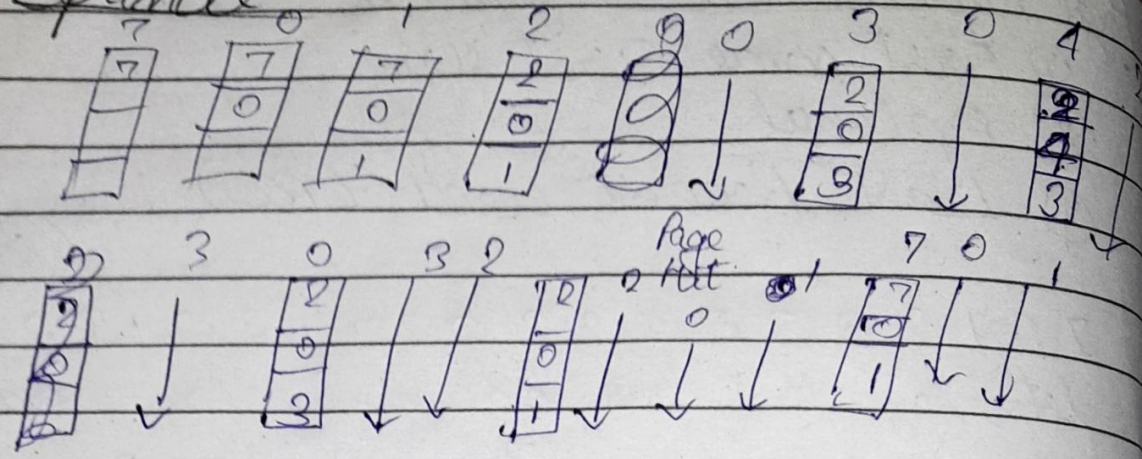
Page Hits : 5

Page Miss : 15 (count after all frames are filled)

Page Hit ratio : $\frac{5}{20} = \frac{H_1}{N}$

Page Miss ratio : $\frac{M_2}{N} = \frac{15}{20}$

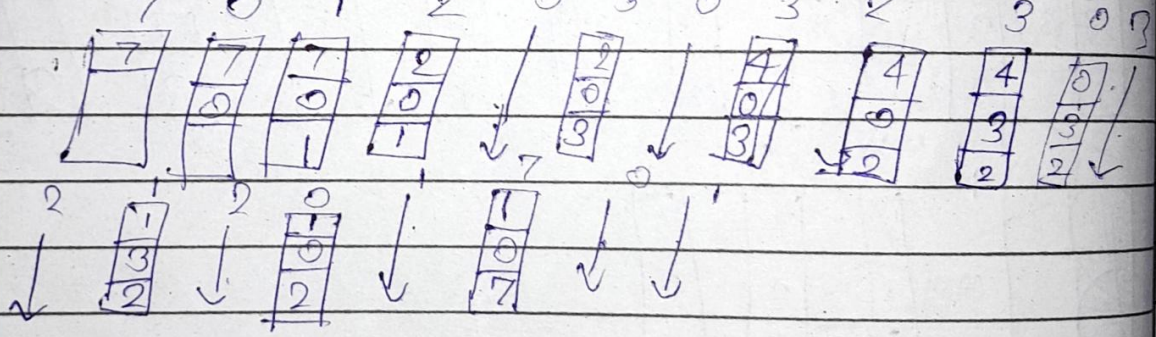
(ii) Optimal



Page Hit : 11
Page Miss : 9

Hit Rate (Hp) = $\frac{11}{20}$ = 55%
Miss rate (Mp) = $\frac{9}{20}$ = 45%

(iii) LRU

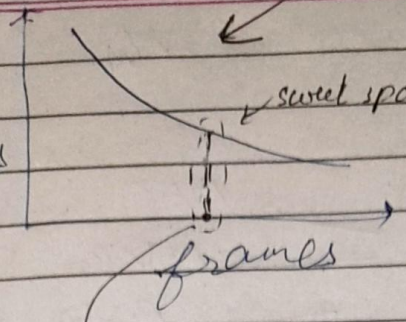


PH. → 8

PM. 12

HR = 8/20
MR = 12/20

Parachor curve.



N/2 ; frames = $\frac{1}{2}$ · No. of pages.
(This result is given by an Approximation algorithm).

two parameters play imp role in Memory manag

t_{a_1} → Access time in main memory
t_{a_2} → Access time in secondary storage

$t_{a_1} << t_{a_2}$

↓ ↓

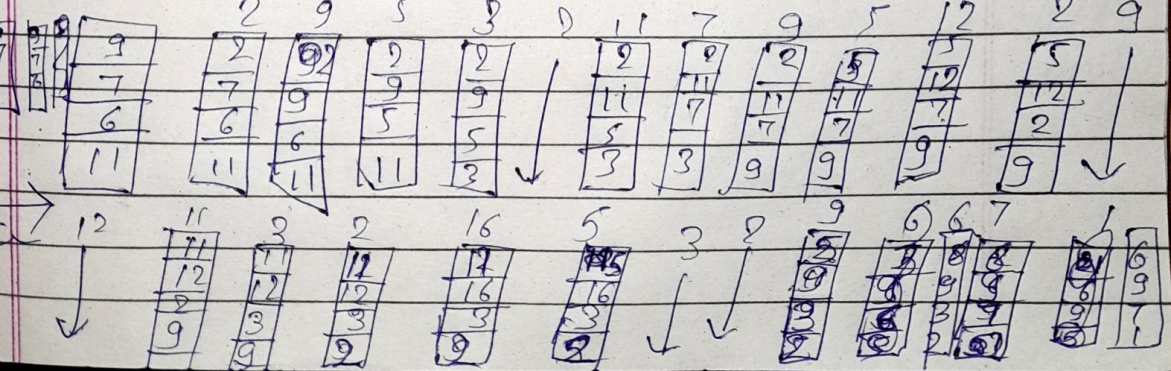
associated with P_H P_M

Then Effective Access time = $t_{eff} = (H \cdot t_{a_1} + M \cdot t_{a_2})$

our objective is to minimize t_{eff}.

Ex: (LRU)

9 7 6 11 2 9 5 3 2 11 7 9 5 12 2 9 12
11 3 2 16 5 3 2 9 6 7 1



H_A = 5/28 H_M = 20/28

Summary

Memory mana

① Continuous → MFT, MVT

② Non-Continuous → Paging, segmented, Page-Segmented, Demand page

→ logical address = Page no. + offset

→ physical address → Memory in frames.

eff. $s(p) = s(f_i)$ // PMT and FMT

→ Master Diagram

→ Write back / write through policy.

(Updation in secondary storage of pages) → (simply replacing the page)

→ Page Replacement algorithm: FIFO, Optimal, LRU.

→ Memory access time (t_{a_1})

→ Secondary storage time (t_{a_2})

→ Effective access time

DATE: / /

Q. In a system, there are three types of resources E, F and G. Four processes P_1, P_2, P_3, & P_4; The processes have declared their max resource requirement max as Max. Eg:

Max(P_2, F) is the maximum no. of instances of F that P_2 would require. The Allocation matrix is:

Allo

	E	F	G
P_1	1	0	1
P_2	1	1	2
P_3	1	0	3
P_4	2	0	0
	5	1	6

Max

	E	F	G
P_1	4	3	1
P_2	2	1	4
P_3	1	3	5
P_4	5	4	1

Availa

E	F	G
3	3	0

E	F	G
8	4	6

Q. From the perspective of deadlock avoidance, which of the following is true.

(i) System is in safe state

(ii) System is not in safe state, but would be, if one more instance of E was availab

(iii) " " " " " "

if one instance of F is available

(iv) " " " " " "

if one instance of G was availabl

[3 3 0] →

		E	F	G
Need →	P_1	3	3	0
	P_2	1	0	2
	P_3	0	3	2
	P_4	3	4	1

(P1)

Av → [3 3 0] + [1 0 1]

[4 3 1].

∴ if G is added

P_2 [4 3 2]

[5 4 4] → P_3 [6 4 7]

↓ P_4

[8 8 4 7]

∴ Resource G is required

Q. Exec in Ubuntu / Linux.

```
#include <stdio.h>
int main()
{
int i;
for(i=0; i<10; i++)
  {
    if (i%2==0)
      fork();
  }
return 0;
}
```

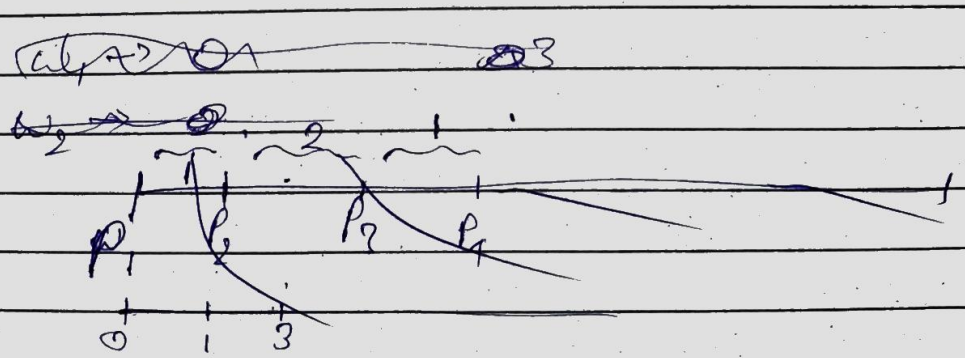
PAGE NO :
DATE : / /

Eg: Consider the processes with arrival time (in ms) and their CPU bursts are as follows:

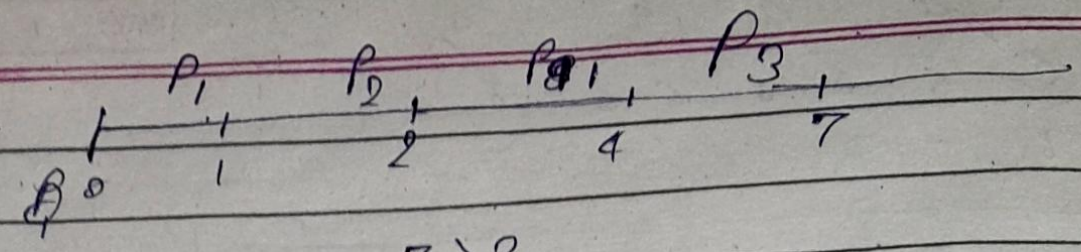
	AT	BT
P_1	0	3 2
P_2	1	1
P_3	3	3
P_4	4	7

They are executing by SRTF (Preemptive) if the avg waiting time is 1 ms. Then value of Z is:

	BT + AT
$W_1 \rightarrow 0$	3
$W_2 \rightarrow 2$	4
$W_3 \rightarrow 1$	7
$W_3 \rightarrow 3$	7+Z



Case 1



$z > 3$

$w_1 = 1$

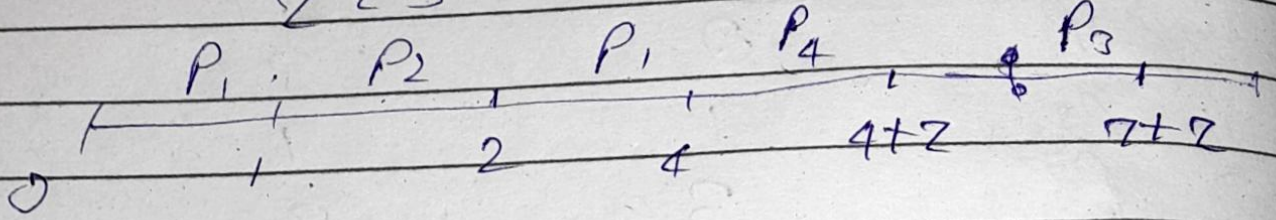
$w_2 = 0$

$w_3 = 1$

$w_4 = 3$.

Case 2:

$z < 3$



$w_1 \rightarrow 1$

$w_2 = 0$

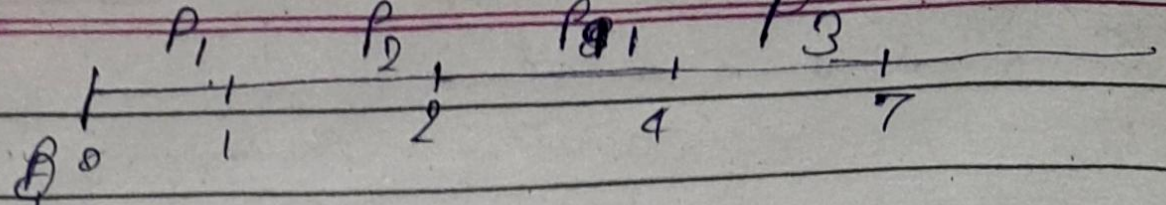
$w_3 = 1 + z$

$P_4 = 0$

$$\frac{2+z}{4} = 1,$$

$$\boxed{z = 2}$$

Case 1



$z > 3$

$w_1 = 1$

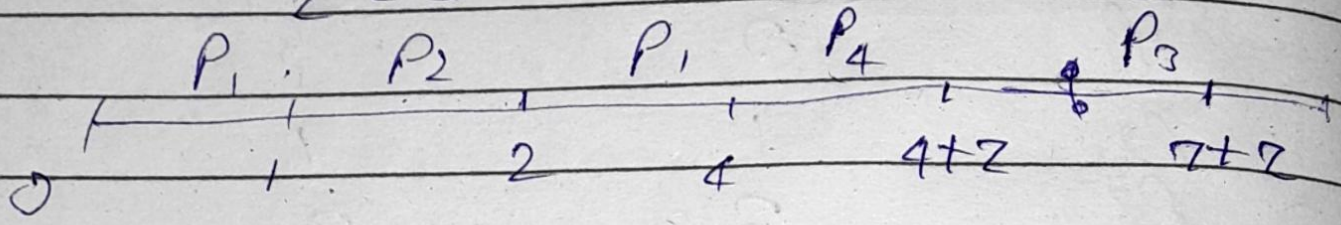
$w_2 = 0$

$w_3 = 1$

$w_4 = 3$.

Case 2:

$z < 3$



$w_1 \rightarrow 1$

$w_2 = 0$

$w_3 = 1 + z$

$P_4 = 0$

$$\frac{2+z}{4} = 1,$$

$$z = 2$$

7/8/24

PAGE NO.:
DATE: / /

OS

Secondary storage:

Storage types:-

- RAM (Random Access Memories)
- Serial Access Storage Device (SASD)
- Direct Access storage Device (DASD)

} divided based on access time

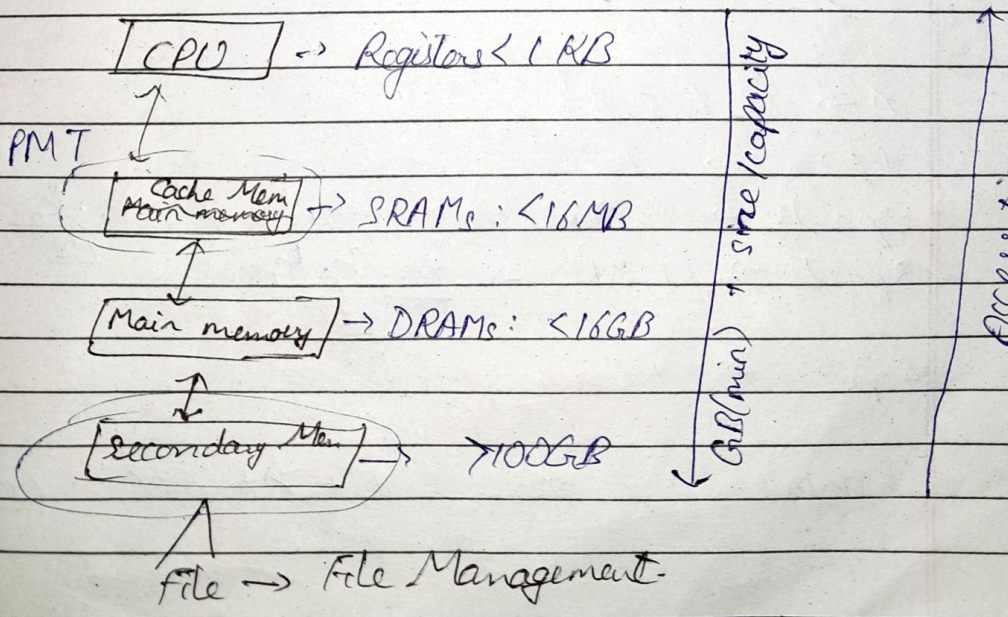
i ---------- jth (memory blocks)

Say we need to find t_{ij}. (access time from block i to j).

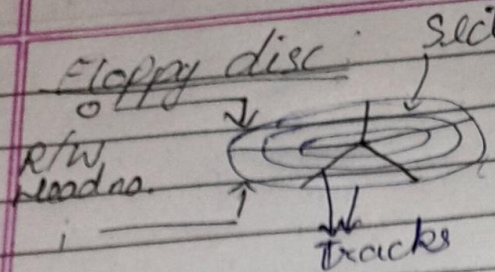
then if $T_{ij} \simeq 0$ → RAM.

for SASD → T_{ij} → very high (Eg: Magnetic tape)

For DASD → T_{ij} : {0 — very high}
(Eg: Hard disk; floppy disk)



Floppy disc



< track no., sector no. >

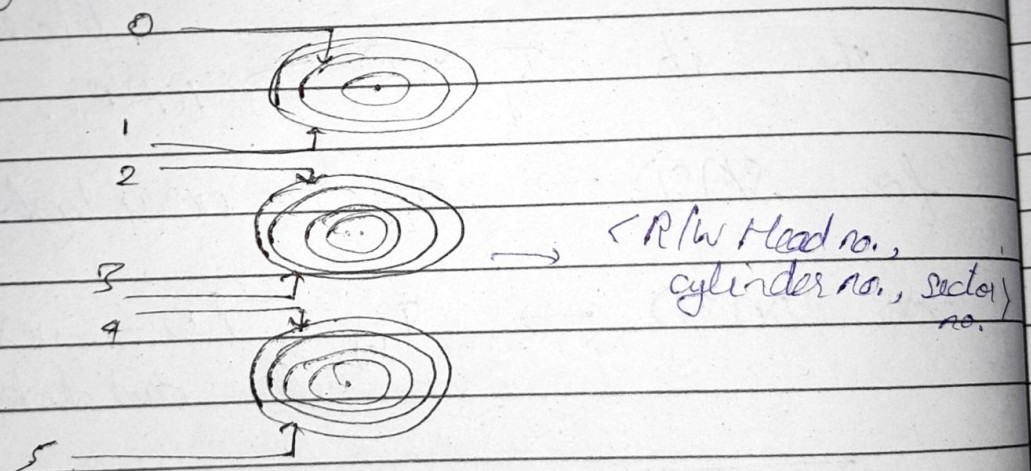
⇓

Block of Memory

if the disc is two sided:

< R/W Head no., track no., sector no. >

if there are multiple discs, then



A → {a, b, c, d}.

1 part → 1 way

2 parts → (1,3) or (2,2) → $4_{C_3} \cdot 3_{C_3} + \frac{4_{C_2} \cdot 3_{C_2}}{2!}$

3 parts → $(1,2,1)_2 \rightarrow \frac{4_{C_2} \cdot 3_{C_1} \cdot 1_{C_1}}{2!}$

4 parts → 1 way;

∴ total no. of ways = 1 + 7 + 6 + 1 = 15