02-06-2025

Types of Operating System:

Batch operating system
multiprogramming operating system
multitasking / Time-sharing OS
Real-time OS
Embeded OS
Networking OS

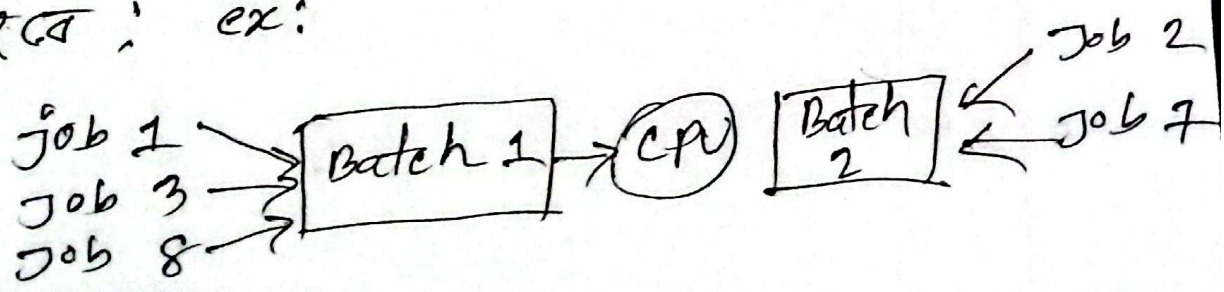
Batch OS :-

user 1 → Job 1
user 2 → Job 2
user 3 → Job 3
⋮
Job n

→ Operator

operator এর কাজ Batch করা

→ Similer type এর job নিয়ে একটি group করে রাখে : ex:



Description:

Disadvantage:

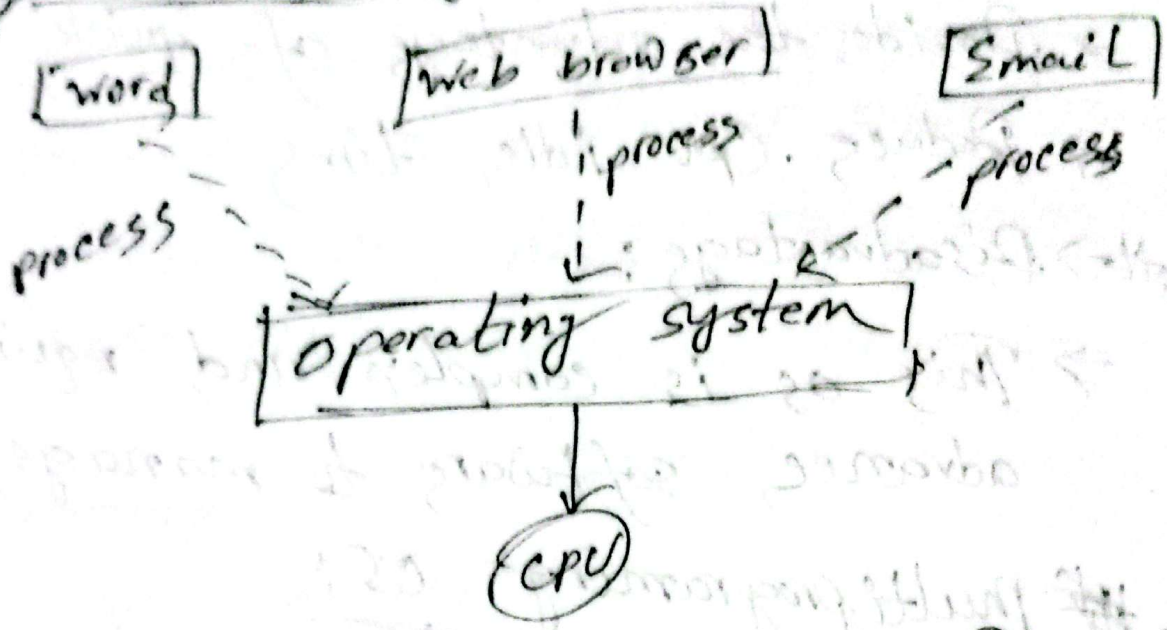
→ Lack of interaction between user & the job

→ CPU is often idle.

→ Difficult to provide the desired priority

Advantage:

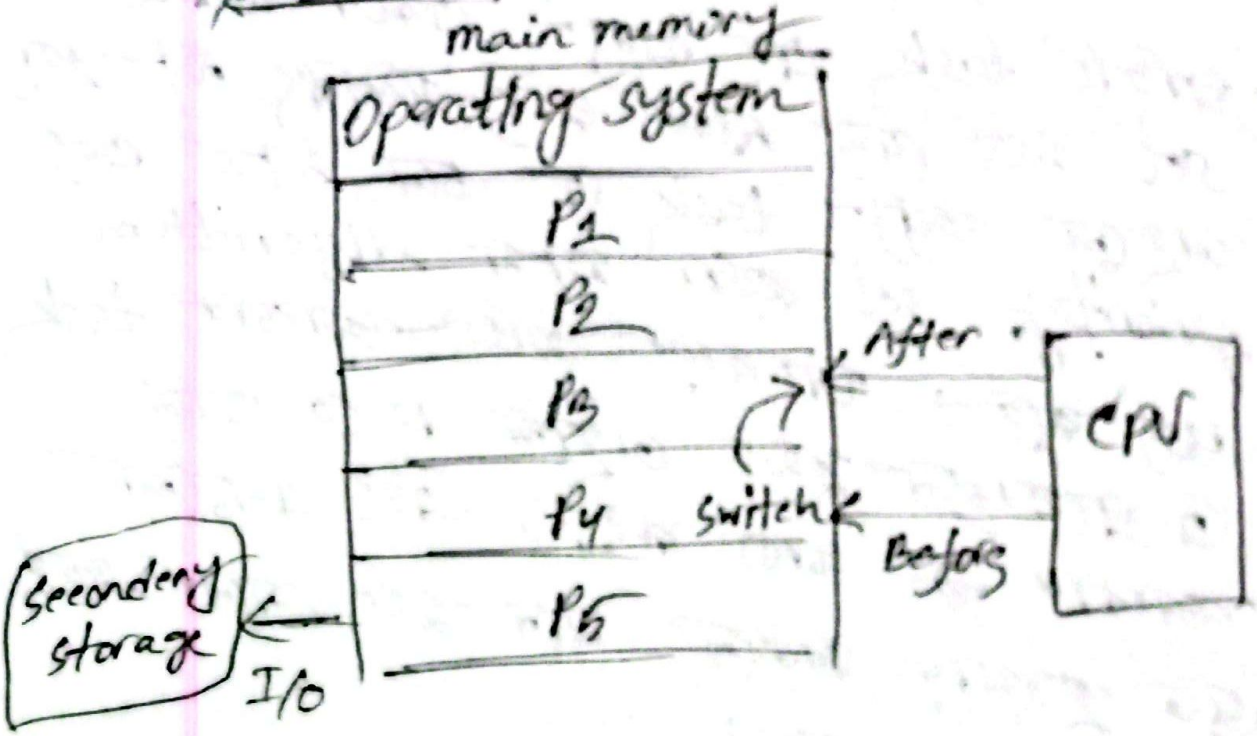
Multitasking OS / Time-sharing OS -



- প্রতিটি task এর জন্য একটা time set করে দেয়া থাকে যদি তা সম্পন্ন না হয়ে থাকলে অন্য task এ চলে যায় এবং priority set করা থাকে Algorithm এর মাধ্যমে। আবার যদি আরো task এ প্রয়োজন হয় তাহলে ফিরে এসে আবার ও কাজ করবে, এবং এতগুলো দ্রুত হওয়ার কারনে মনে হয় যেনো একই সাথে সব হচ্ছে।

* Advantages:
  → Provides the advantages of quick response
  → Reduces CPU idle time
* → Disadvantages:
  → This os is complex and requires advance software to manage

Multiprograming OS:



P4 → A program compiling code
P3 → Reading a file.

Multiprograming এ একটা কাজ শেষ না হওয়া পর্যন্ত অন্য কাজে যায় না, serially কাজ করে। তবে যখন CPU instruction দিচ্ছে তখন switch করতে পারে।

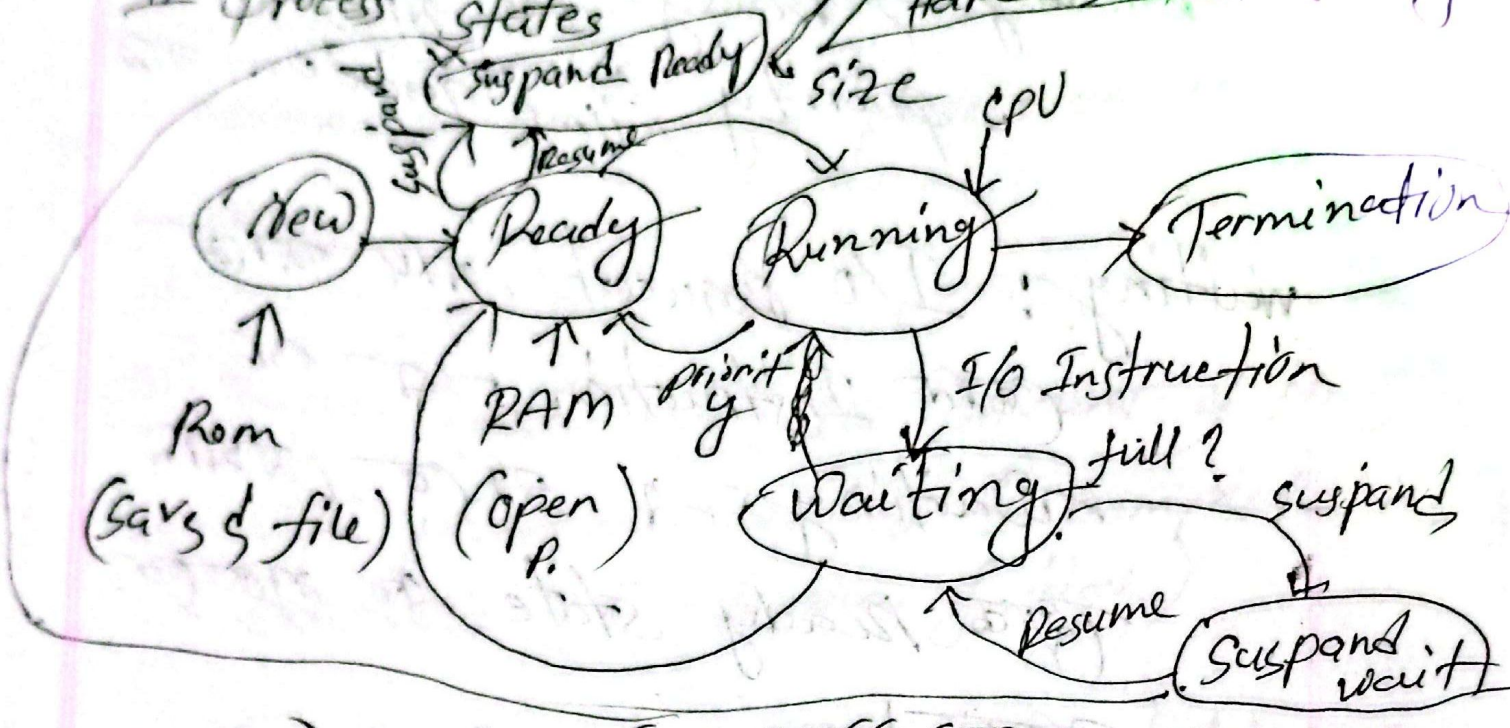
* Advantage:
  - → CPU sharing
  - → Resources Management
  - → Scheduling Algorithm

* Disadvantage:
  - → Response time same

06-2025

RAM
ROM
Hard disk
Primary memory
secondary
Tertiary

Process States



→ New ⇒ Rom ও সাধারণ নতুন, যতটুকু সেভ file বা installed app

→ Ready ⇒ Ram ও সাধারণ নতুন, প্রস্তুত Open করবার সময় scheduling বা dispatch বলে, Ready Queue

→ Running ⇒ CPU → একই time ও একটা কাজ করে, তবে যদি অন্য কোনো priority সম্পন্ন Instruction/প্রস্তুত আসে তাহলে বর্তমানটা Ready ও পাঠিয়ে দেয়।

Running → Ready তে যাওয়ার priority / Time quantum শেষ

Waiting: I/O Request যদি আসে কোন Instruction থেকে তার Waiting এ যাবে, সেখান থেকে Ready state এ যাওয়ার জন্য।

Suspand wait: Waiting যদি full হয়ে যায় তাহলে Suspand wait এ যাবে, Waiting এর সাহায্যার জন্য। যখন Waiting full থাকে তাহলে সেটা আবার Suspand Ready তে চলে যায়।

Suspand Ready: যদি অনেকগুলো file Open থাকে Ready Queue তে তাহলে সাহায্য করার জন্য তা Suspand Ready তে যায়।

Arrival Time :- Arrival time refers to the time at which a process enters the ready queue.

completion time

Burst Time : It is the amount of time the processors requires to complete it's execution. (duration of time)

Turn around time: Turn around time is the total time taken for a process to complete after enters to the system. (actual time)

TA = Completion time − Arrival time

Completion time: It is the time at which a process finishes it's execution.

Waiting time: It is the total time a process spends waiting in the ready queue before it gets cpu execution

WT = ~~TA~~ BT − ~~TA~~ / TA − BT

multi tasking a use 2?

Response time → Response time is the time from when a process arrives in the ready queue to the the first time it gets the CPU. (first CPU access)

RT = first CPU start time − Arrival time

18-06-2025

CPU scheduling:

CPU scheduling is a process that allow one process to use the cpu while another process is delayed (in standby)

Due to unavailability of any resources such as I/O devices, thus making full use of CPU.

* Types —

time sharing multitasking

(I) Preemptive - It is a type of CPU scheduling where the operating system can interrupt current running process and assign the CPU to another process usually based on priority or time.

non-programming

(II) Non-preemptive - It is a type of CPU scheduling where once a process starts executing on the CPU it runs to completion.

12 ** popular

Scheduling Algorithm

preemptive

- SRTF (shortest remaining time first)
- LRTF (Longest remaining time first)
- ** Round Robin
- ** priority based

Non-preemptive

- ** FCFS (First coms first serve
- ** SJF (shortest job fir
- LJF (Longest job fir
- HRRN (Highest Respons Ratio next)
- Multilevel queue

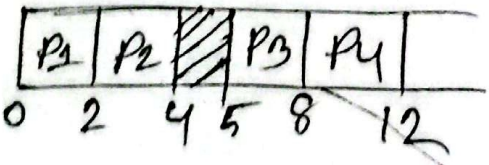
FCFS Algorithm -

process no	Arrival time	Burst time	Completion time	Turn around time	Waiting time	Response time
P_1	0	2	2	2	0	0
P_2	1	2	4	3	1	1
P_3	5	3	8	3	0	0
P_4	6	4	12	6	2	2

TA = CT - AT

Ready Queue | P_1 | P_2 | P_3 | P_4 |

Gantt chart



WT = BT - TA / TA - BT must be (+ve)

RT = first CPU time - AT

Note: Non-preemptive এ Waiting time == Response time

Note: কোন বা একাধিক P যদি একই arrival time এ আসে তাহলে যে আগে যে access পাবে

28/06-2025

Shortest Job first (SJF)

Mode → Non-Premptive

Process no.	Arrival time	Burst time	Completion time	Turn Around time	waiting time	Response time
P1	1	3	6	5	2	2
P2	2	4	10	8	4	4
P3	1	2	3	2	0	0
P4	4	4	14	10	6	6

Ready Queue: P4 | P3 | P2 | P4
2 1 3 4

Gantt Chart: | P3 | P1 | P2 | P4 |
0 1 3 6 10 14

TA = CT − AT
WT = TA − BT

Note: Burst time মাত্র কম যে আগে Gantt Chart এ যাবে, সমান হলে যে আগে Arrive হয়েছে সেটা Running এ যাবে অবশ্যই বাকি সব time এ আসতে (Arrive) or Ready Queue তে থাকতে।

25-06-2025

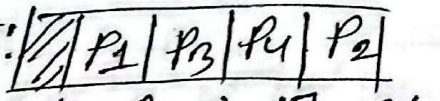
Longest Job first.

process no	Arrival time	Burst time	Completion time	waiting time	Turn Around time	Response time
P_1	1	2	3	0	2	0
P_2	2	4	21	15	19	15
P_3	3	6	9	0	6	0
P_4	4	8	17	5	13	5

Ready Queue: P1 | P2 | P3 | P4

Note: যার Burst time বেশি সে আগে আসবে।

Gantt chart:



TA = CT − AT

WT = TA − BT

Note: Burst time সমান হলে Arrival time এ যে আগে।

Longest Remaining time first (LRTF)

P. NO	AT	BT	CT	TAT	WT	RT
P_1	0	5	10	10	5	0
P_2	1	3	11	10	7	3
P_3	2	4	12	10	6	0
P_4	4	1	13	9	8	8

Ready Queue: | P_1 | P_2 | P_3 | P_4 |

gantt chart:

P_1	P_1	P_3	P_1	P_2	P_3	P_1	P_2	P_3	P_1	P_2	P_3	P_4
0	1	2	3	4	5	6	7	8	9	10	11	12 13

P_1 = 5 4 3 2 1 X

P_2 = 3 2 1 X

P_3 = 4 3 2 1 X

P_4 = X

Preemptive — Algo

Shortest Remaining Time First (SRTF)

let, time = 1 unit

P.NO	AT	BT	CT	TAT	WT	RT
P1	0	5	9	9	4	0
P2	1	3	4	3	0	0
P3	2	4	13	11	7	7
P4	4	1	5	1	0	0
Avg:	1.75	3.25	7.75	6	2.75	1.75

Ready Queue: P1 P2 P3 P4

Gantt chart:

P1	P2	P2	P2	P4	P1	P1	P1	P1	P3
0	1	2	3	4	5	6	7	8	9	10

P3	P3	P3
11	12	13

P1 = 4, 3, 2, 1
P2 = 2, 1
P3 = 4, 3, 2, 1
P4 = 1

TAT = CT − AT
WT = TAT − BT

Note: যার Remaining time / Burst time কম সে আগে আসবে।

RT = first − AT

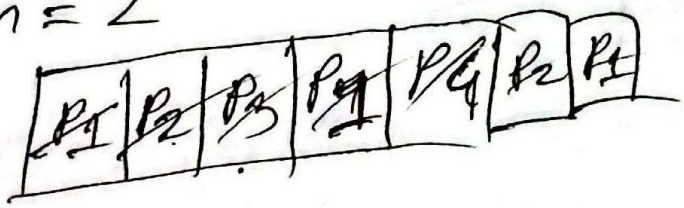
Preemptive

Round Robin Algorithm

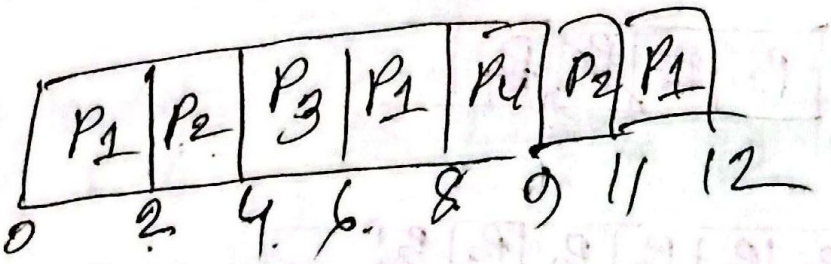
PT	AT	BT	CT	TAT	WT	RT
P_1	0	5	12	12	7	0
P_2	1	4	11	10	6	1
P_3	2	2	6	4	2	2
P_4	4	1	9	5	4	4

Time Quantum = 2

Ready Queue



Gantt chart



$P_1 = 5, 3, 1, 0$

$P_2 = 4, 2$

$P_3 = 2$

$P_4 = 1, 0$

Round Robin এক্ষেত্রে Ready Queue তে যে আগে আসে সে প্রথমে যায়। কিন্তু সে TQ এর সমান কিংবা কম gantt time নেয়। এবং বাকি সময়ে এ time এ Arrive হওয়া process এর পর Ready Queue

Problem 1: (non - premtive)

Priority scheduling

P.no.	AT	BT	Priority	CT	TAT	WT	RT
P_1	0	5	2	5	5	0	0
P_2	1	3	1	8	7	4	4
P_3	2	8	4	22	20	12	12
P_4	3	6	3	14	11	5	5

Ready Queue | P_1 | P_2 | P_3 | P_4 |

gantt chart

P_1	P_2	P_4	P_3

0 5 8 14 22

যার Priority value কম তার priority বেশী / তত; সে তারের gantt chart এ আগে

TAT = CT - AT

WT = BT - TAT - BT

Preemptive scheduling Priority

P.no	AT	BT	Priority	CT	TAT	WT	RT
P_1	0	10	3	16	16	6	0
P_2	2	4	1	6	4	0	0
P_3	3	6	4	22	19	13	13
P_4	5	2	2	8	3	1	1

Quantum time = 2

Ready Queue $P_1 | P_2 | P_3 | P_4$

gantt chart

P_1	P_2	P_2	P_4	P_1	P_3

0 2 4 6 8 16 22

~~$P_1 = 10-8$~~

~~$P_2 = 4-2$~~

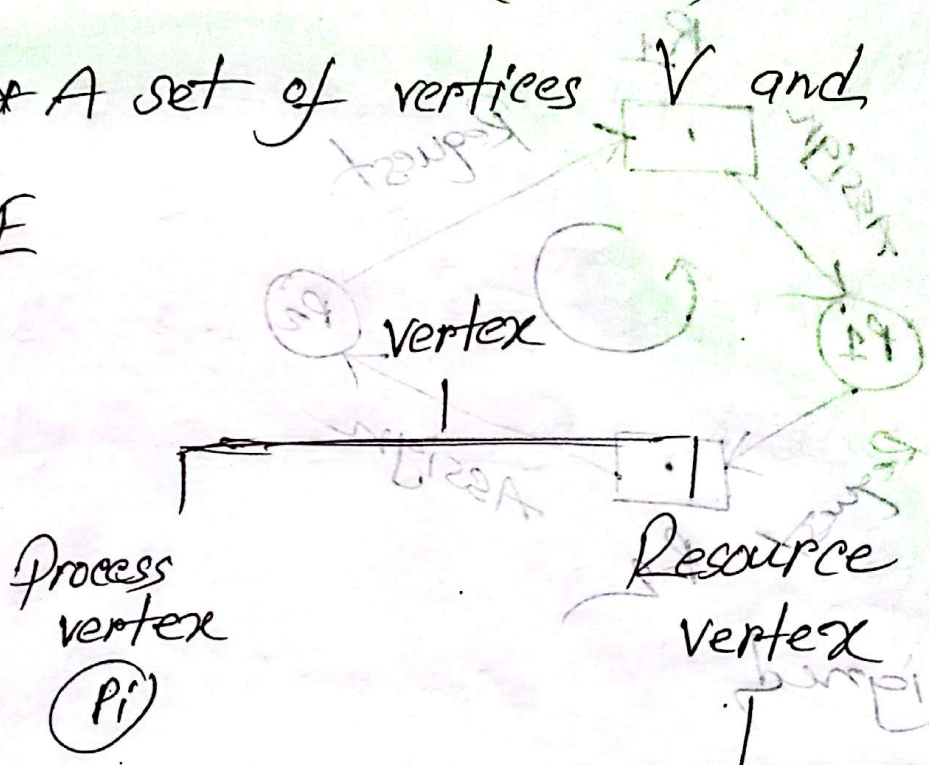
~~$P_3 = 6$~~

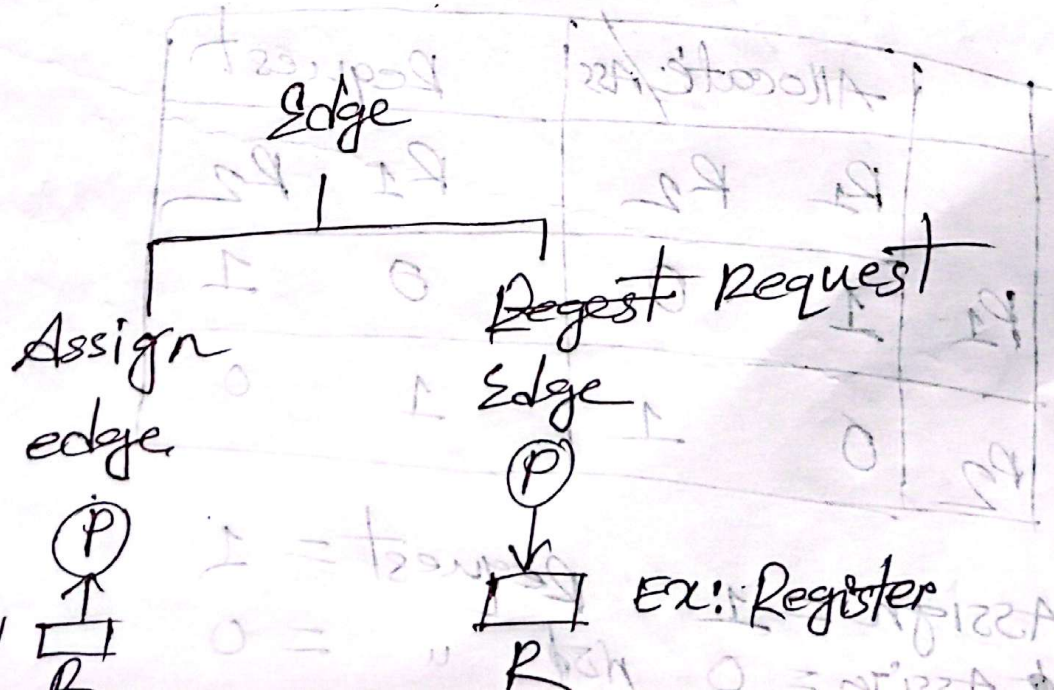
~~$P_4 = 2$~~

Final

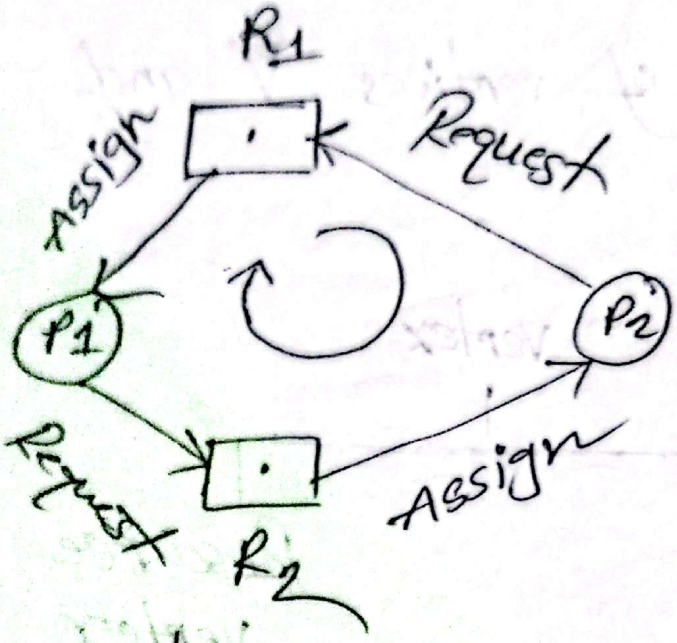
Resource Allocation Graph (RAG)

* A set of vertices V and a set of edges E





ex:



Assigned

$P_1 \leftarrow R_1$

$P_2 \leftarrow R_2$

method to identify deadlock

	Allocate/Ass		Request	
	R_1	R_2	R_1	R_2
P_1	1	0	0	1
P_2	0	1	1	0

Assign = 1 Request = 1

Not Assign = 0 Not " = 0

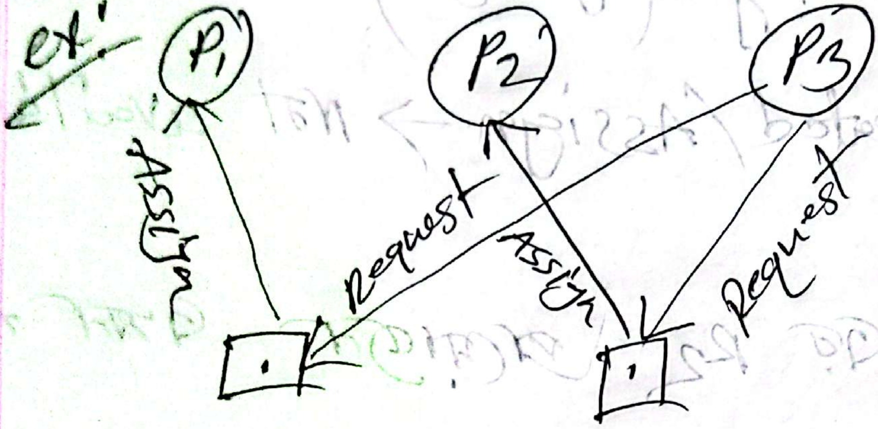
* Availability $\begin{pmatrix} R_1 & R_2 \\ 0 & 0 \end{pmatrix}$

→ Allocated/Assign → Not available = 0

Ex: P_1 এর R_2 প্রয়োজন, এখন Availability তে দেখবো R_2 Available আছে কি না — যদি না থাকে তাহলে ঐ process Complete না।

Note: যদি কোনো process complete হয়ে যায় তাহলে ঐ process এর Resource অন্য কোনো process এর প্রয়োজন হলে তাহলে সে ঐটা use করতে পারবে, যেহেতু Available

Ex:



∴ P1 ← R1 = 0

∴ P2 ← R2 = 0

* Assigned

	R1	R2
Availability	0	0
P1 → Completed	1	0
P2 → Completed	1	1
P3 → completed		

P1 ← R1

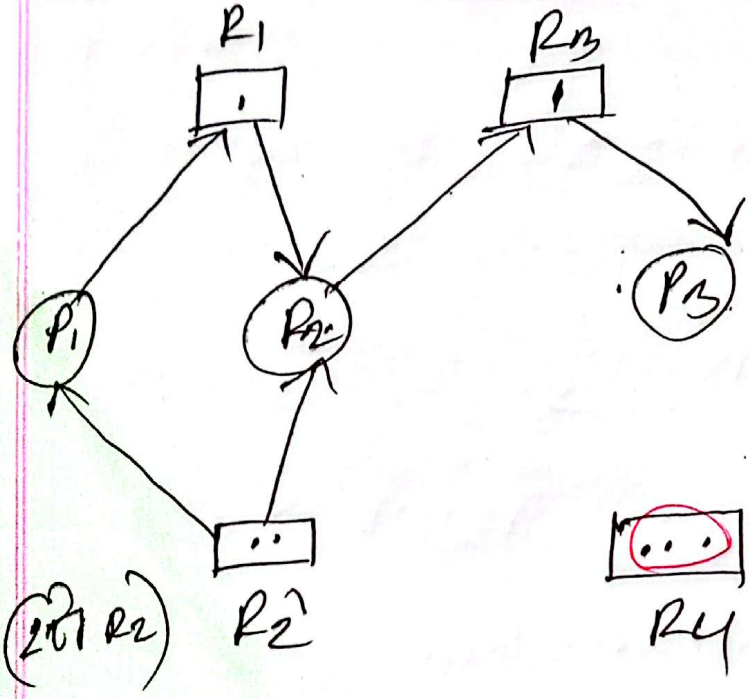
P2 ← R2

~~R3 ←~~

	Allocate R1	Allocate R2	Request R1	Request R2
P1	1	0	0	0
P2	0	1	0	0
P3	0	0	1	1

∴ R1 R2 কারণে P3 এর দরকার ২টি Available, তাই এটা Complete.

* Available $\begin{pmatrix} R_1 & R_2 & R_3 \\ 1 & 1 & \end{pmatrix}$



Ass: $P_1 \leftarrow R_2$

$P_2 \leftarrow R_1, R_2$

$P_3 \leftarrow R_3$

	R_1	R_2	R_3	R_4
	0	0	0	0
P_3	0	0	1	1
P_2	1	1	1	1
P_1	1	1	1	1

Request

Allocate

	R_1	R_2	R_3	R_4	R_1	R_2	R_3	R_4
P_1	0	1	0	0	1	0	0	0
P_2	1	1	0	0	0	0	1	0
P_3	0	0	1	0	0	0	0	0

Available :

R_1	R_2	R_3	R_4	
0	0	0	1	
0	1	1	1	P_1 → completed
1	1	1	1	P_2 → com
0	0	1	1	P_3 → complet

Deadlock Avoidance

In deadlock avoidance, the request for any resource will be granted, if the resulting state of the system doesn't cause deadlock in the system.

Banker's Algorithm (Deadlock Detection)

Safe & unsafe state:

→ A safe state refers to a system where the allocation of the resources to each process ensures the avoid new of deadlock.
avoidence

Total A=10, B=5, C=7

Process	Allocation			max need			Available			Remaining need		
	A	B	C	A	B	C	A	B	C	A	B	C
P_1	0	1	0	7	5	3	3	3	2	7	3	3
P_2	2	0	0	3	2	2	5	3	2	1	2	2
P_3	3	0	2	9	0	2	7	4	3	6	0	0
P_4	2	1	1	4	2	2	7	4	5	2	1	1
P_5	0	0	2	5	3	3	7	5	5	5	3	1
	7	2	5				10	5	7			

∴ safe state

Sequence: $P_2 \rightarrow P_4 \rightarrow P_5 \rightarrow P_1 \rightarrow P_3$

∵ P_1 এর ~~need~~ need 4(B), কিন্তু ~~Available~~ rem 3(B)

3-4=-1 এভাবে যে Complete হবে, এমন অনেক P_2, check করতে পারি, অনেক অন্য কোন Process, যার Sequence Maintain করতে হবে।

Total : A=10, B=5, C=7, D=8

	process	Allocation				max need				Available				Remaining need			
		A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D
✓	P_1	0	1	0	3	7	5	3	4	3	3	2	1	7	4	3	1
✓	P_2	2	0	0	1	3	2	2	2	5	3	2	2	1	2	2	1
	P_3	3	0	2	1	9	0	2	2	7	4	3	2	6	0	0	1
✓	P_4	2	1	1	0	4	2	2	2	7	4	5	4	2	1	1	2
✓	P_5	0	0	2	2	5	3	3	3	7	5	5	7	5	3	1	1
		7	2	5	7					10	5	7	8				

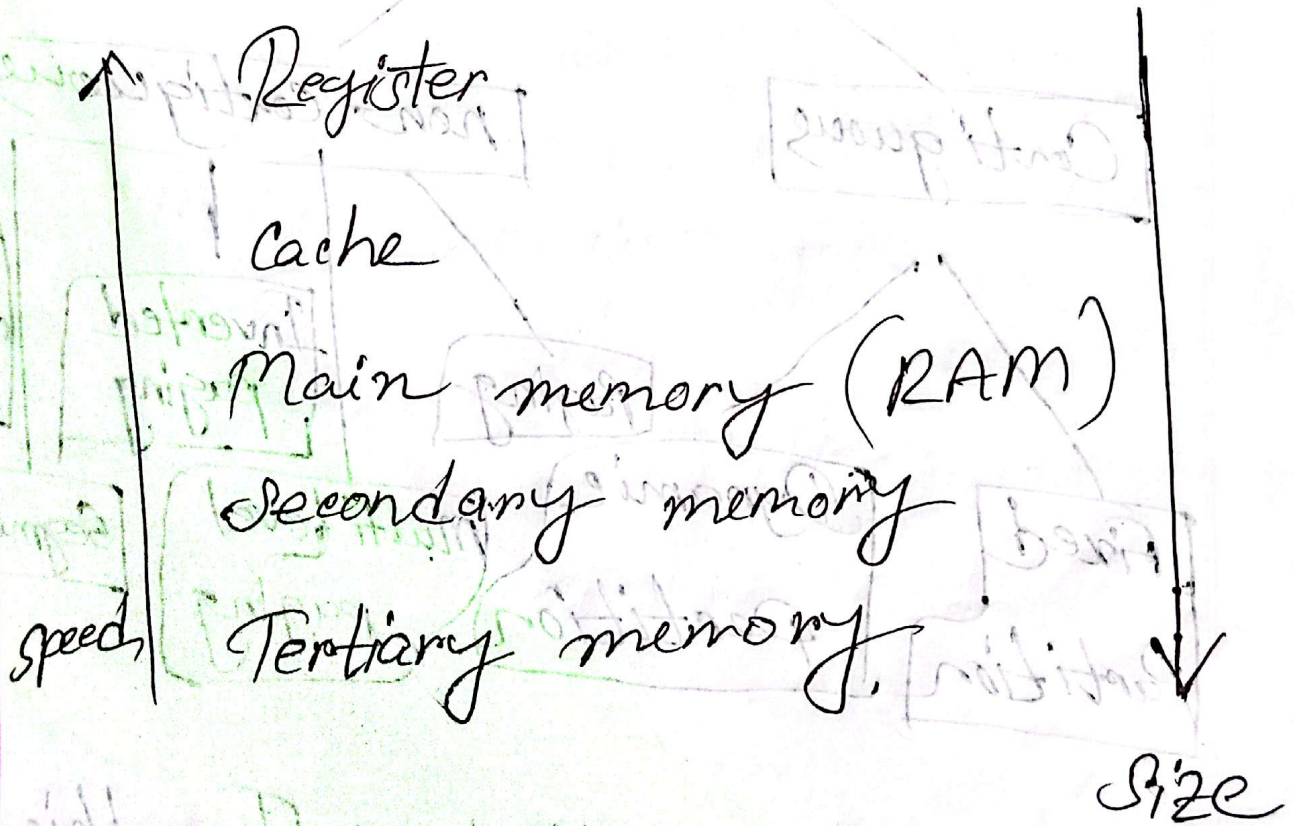
Sequence : $P_2 \rightarrow P_4 \rightarrow P_5 \rightarrow P_1 \rightarrow P_3$

Safe state.

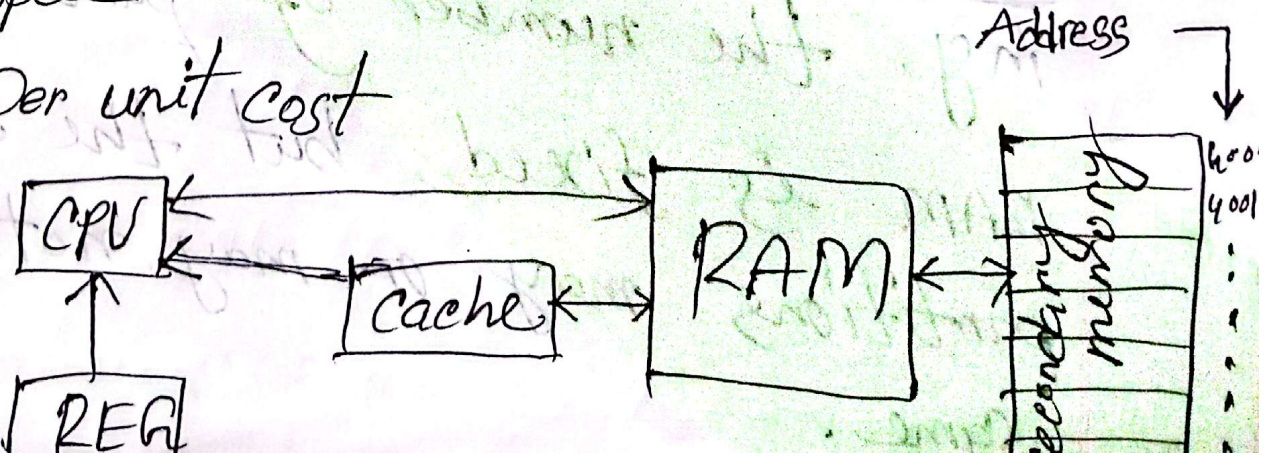
Note: Multiple Resource -এর ক্ষেত্রে Banker's Algorithm Best আবার Single Resource -এর ক্ষেত্রে RAG Best.

Memory Management

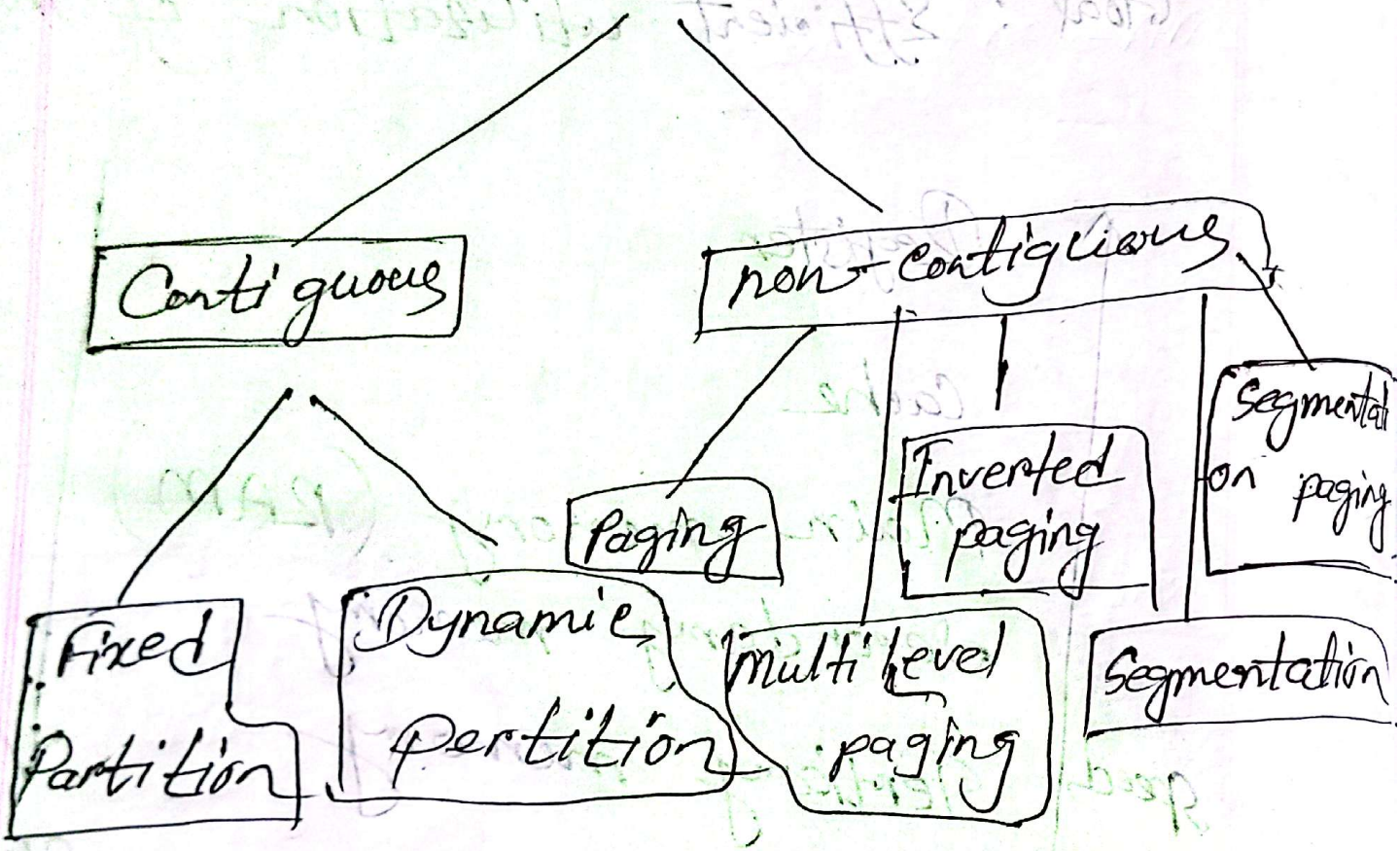
Goal : Efficient utilization of memory



(I) Size
(II) Speed
(III) per unit cost



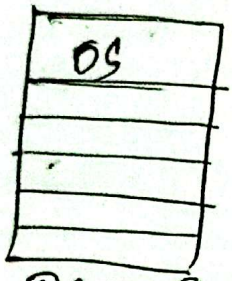
Contiguous & non Contiguous memory allocation:



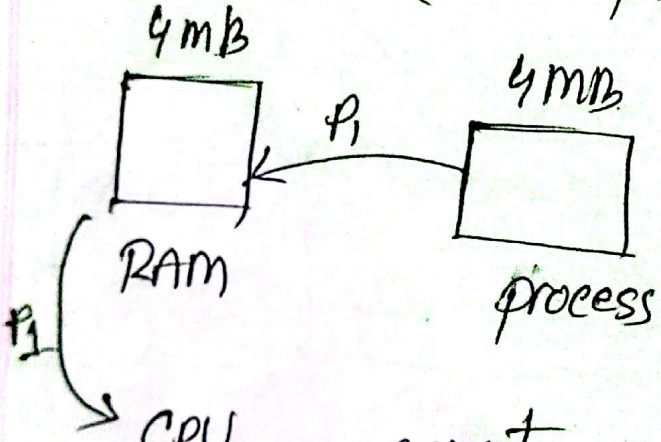
* Fixed size partitioning: In this partitioning, the number of partition in RAM is fixed, but the size of partitions may or may not be the same.

Memory Management

* Degree of multiprograming : Keeping more and process in the RAM.



- RAM (main Memory)



Note: K = time for I/O Request

P_1 request → I/O

∴ CPU

∴ CPU Utilization = 1 − k

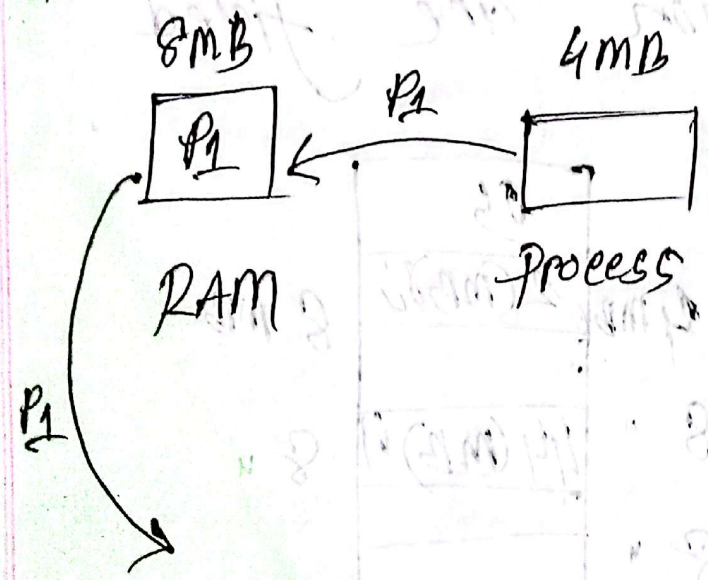
let, K = 70%

∴ 1 − k = 1 − 0.7 = 0.3

= 30%

Note: Dynamic = variable
static = fixed

~~# Dynamic partitioning~~



Total no. of process = 2

$\therefore k \cdot k = k^2$

$\therefore$ CPU Utilization $= 1 - k^2$

$= 1 - 0.49$

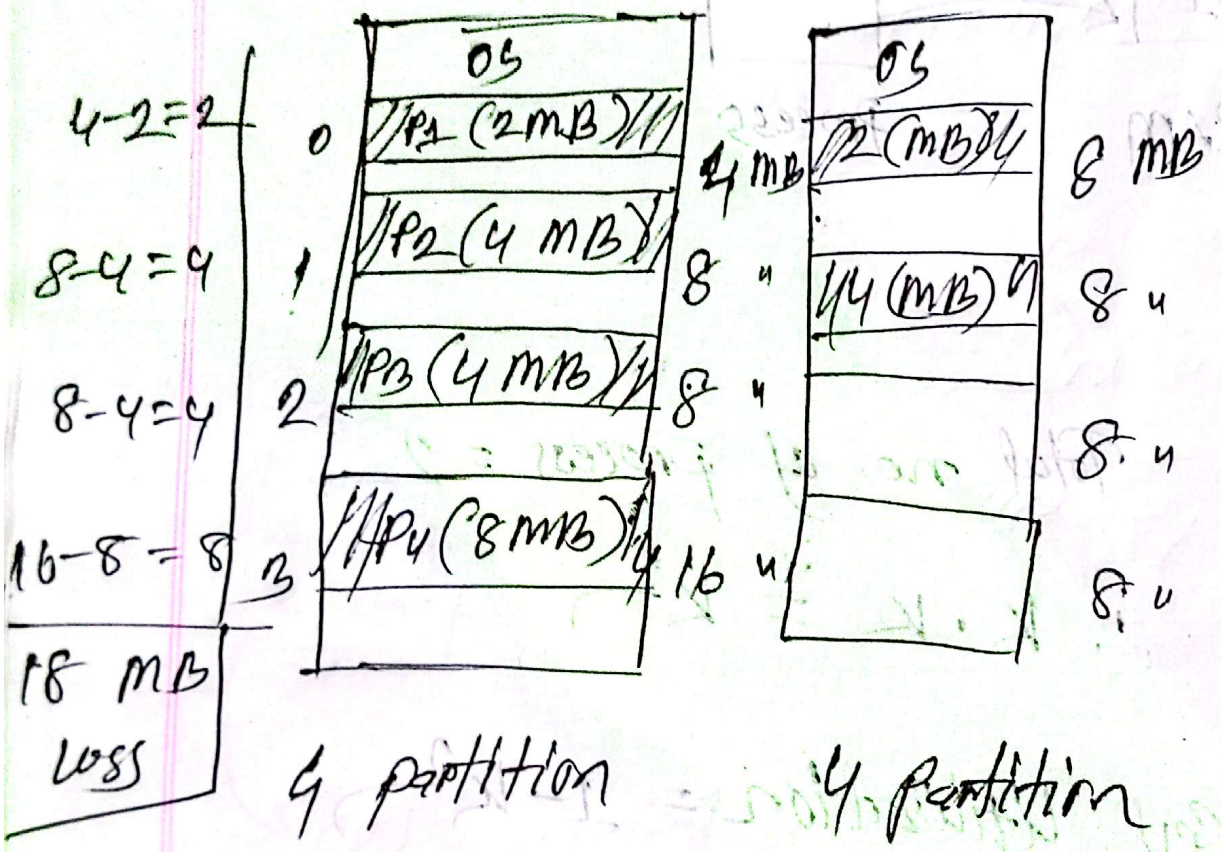
$= 1 - 0.49$

$= 50\%$

Note: Ram size ~~or~~ amount idle time ~~size~~ p(2)

Fixed Partitioning

* No. of partition are fixed



∴ Partition are fixed but the size will vary

→ Size of each partition may or may not be same.

Internal fragementation: যে সমস্যা process < partition.

Limitations:

$P_1 = 2$ MB $\leq$ Partition, $P_2 = 4$ MB

(i) Internal fragmentation

(ii) Degree of multiprogramming is limited (ex: 32 MB (P30))

(iii) External fragmentation

Note: External fragementation হলে সমস্ত Available space সাকার দরুণ use করা যায়না। কারণ process এর task ভাগ হয়না, একটা process একটা partition দখল করে থাকে।

Advantage :

→ Easy to Allocate.

Dynamic Partitioning

In this technique, the partition size is not declared initially. It is declared at the time of process loading.

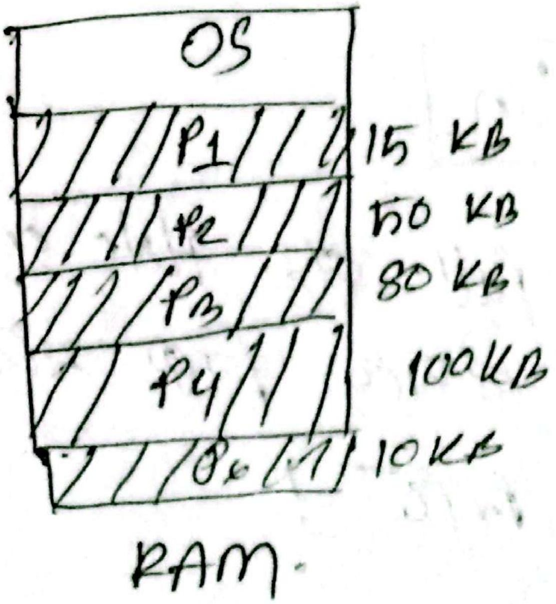
Dynamic / Variable Partitioning

let, $P_1 = 15\,KB$

$P_2 = 50\,KB$

$P_3 = 8\,KB$

$P_4 = 100\,KB$

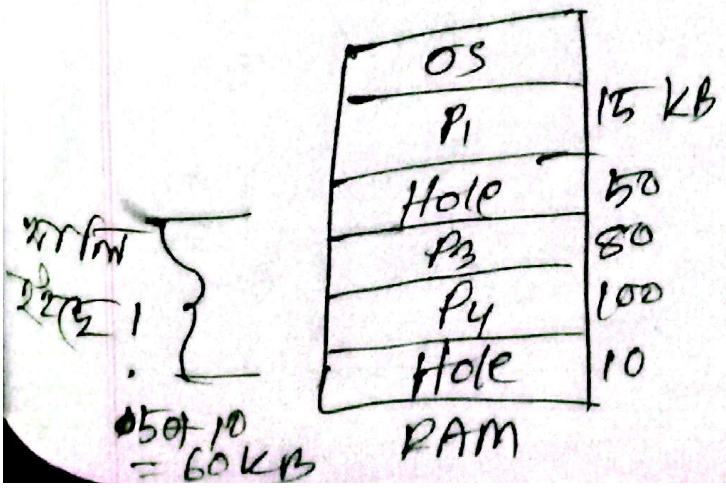


* First unallocated, in runtime will allocated

Advantage:

(I) Internal fragmentation overcome

(II) Degree of multiprograming is high



let, $P_5 = 60\,KB$

∴ ২টা Hole মিলে 60 KB খালি কিন্তু P_5(60) KB কে দিলে তা জায়গা পাবে না।

এই Problem (holes) এর case overcome করার জন্য Algorithm এটি

Compaction

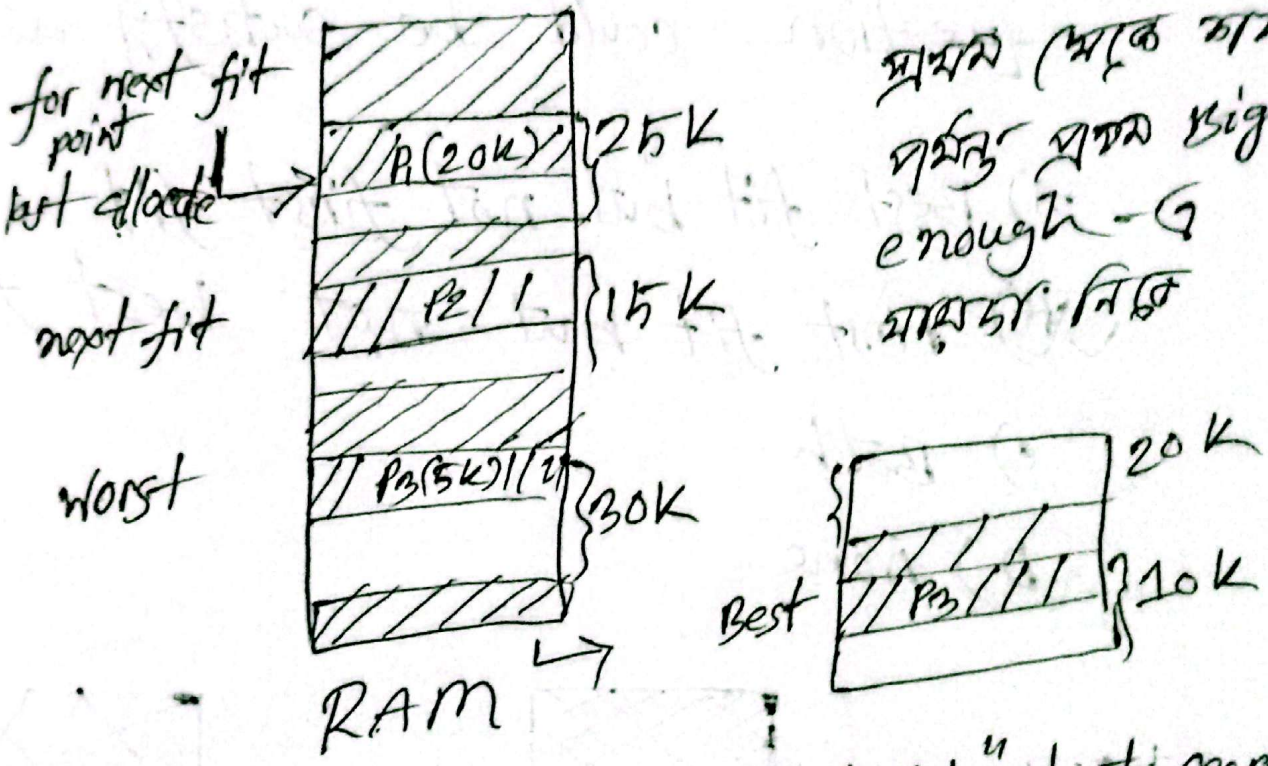
technique → full করে hole গুলো merge করে বেশের দিকে সরানো, বাকিগুলো নিচে।

Dis Advantage:

→ CPU interupt হয়, that makes the system slow

27-08-2025

First case: Allocate the first hole that is big enough (P1 = 25K)



প্রথম থেকে কাছাকাছি প্রথম big enough-এ assign নিচ্ছে

Next fit: "Same as first fit" start search always from the last allocated hole. (P2 = 10K)

Best fit: Allocate the smallest hole that is big enough P3 = 8K

Worst fit: Allocate the largest available hole to process. P4 = 5K

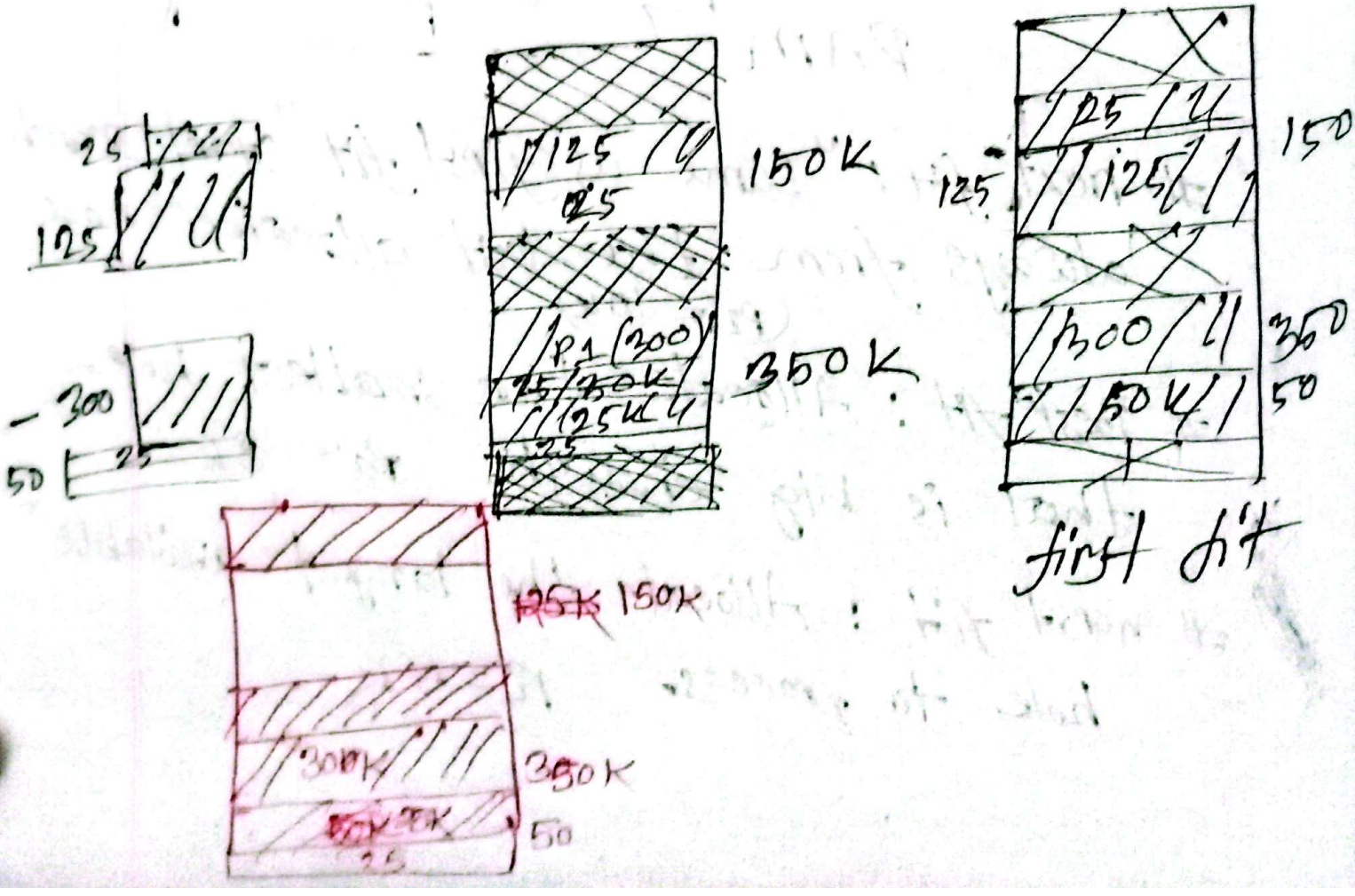
Q: Request from Processes are 300K, 25K, 125K, 50K respectively. The above question could be satisfy with

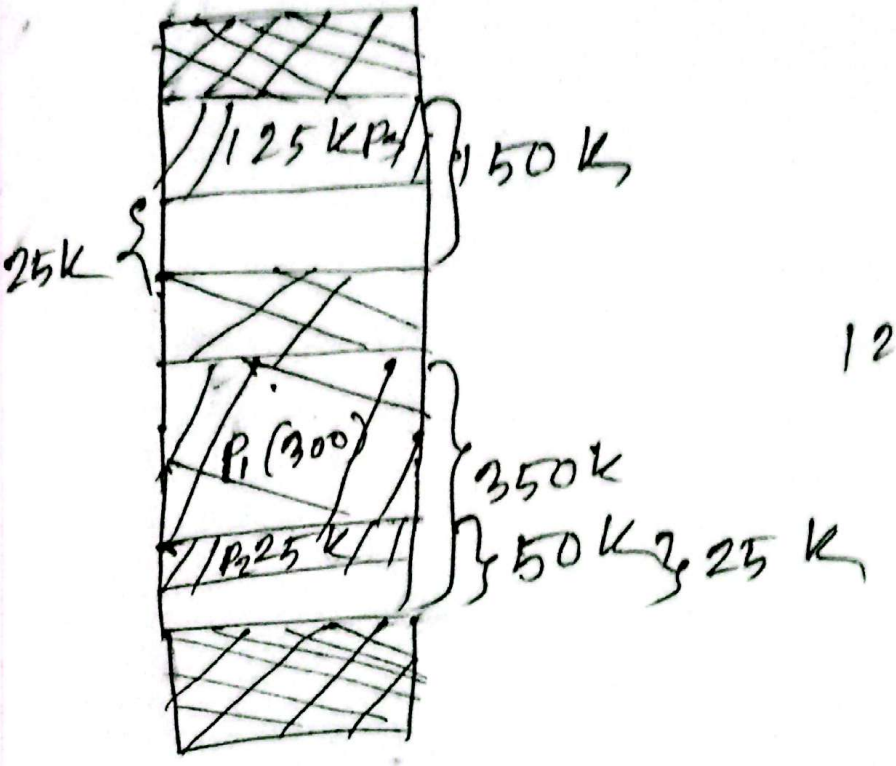
a) Best fit but not first fit

b) first fit but not best fit

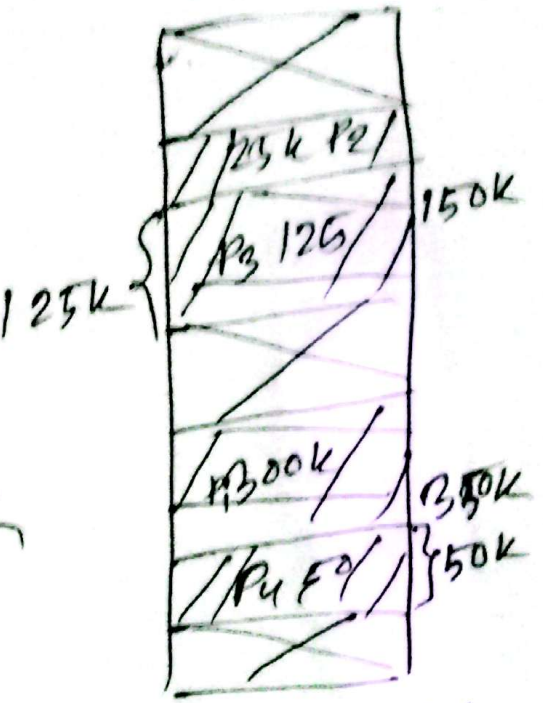
c) Both

d) none





Best fit



first fit