Dt
Pg B+

Software : set of tested programs + documentation

- Software
  - System Software (low-level software) [runs in background]
    - operating S.S.
    - Device Drivers
    - Translators
      - Compiler
      - Interpreter
      - Assembler
    - Utility
      - antivirus
      - cleanup tools
      - defragmenters
  - Application software (end-user software) [runs above system software]
    - Word Processors
      - Notepad, MS Word
    - Database software
      - MySQL, MS Access
    - Multimedia software
      - VLC Media Player, Windows M.P.
    - Graphics software
      - Adobe Photoshop, Blender
    - Web Browsers
      - Google Chrome, Opera, UC

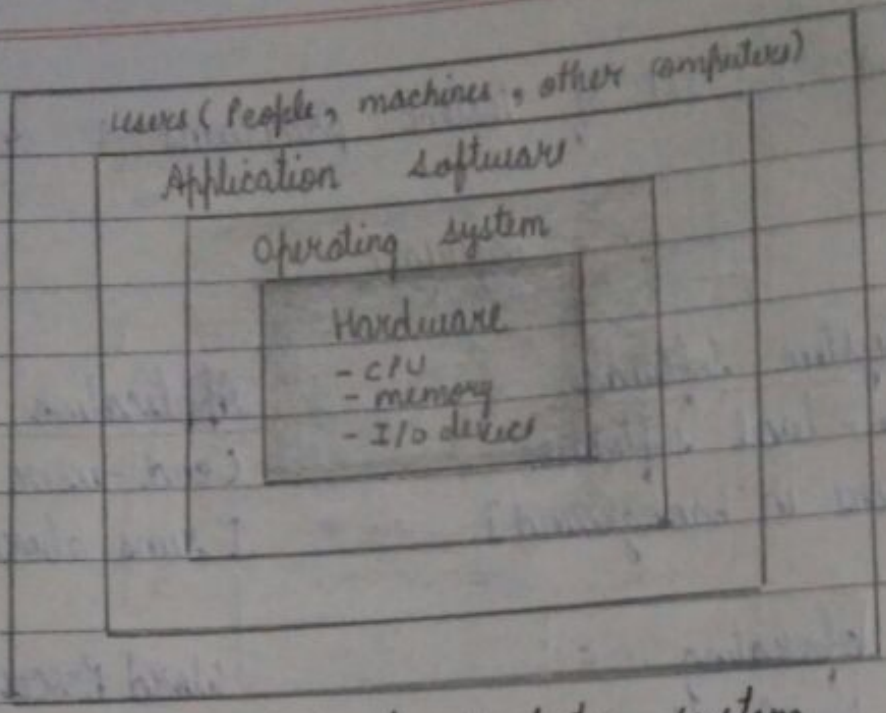
OPERATING SYSTEM SOFTWARE

[kernel - core program] + [some more system p/g]

User's View :
It is an interface between the user and the hardware [CPU, I/O devices, memory]

System's View : control program
It is a resource Manager

Dt
Pg



components of computer system

Interface :

- CUI [Character User Interface]
  eg : DOS
- GUI [Graphical User Interface]
  eg : Windows

Dt
Pg B+

FUNCTIONS

- Resource Management
- CPU Scheduling / Process Management
- Memory Management (Mainly RAM)
- I/O device Management
- Storage Management [disk / file system]
- Security & Protection passwords etc.
- Process synchronization
- Coordination between other softwares & users eg: translators

GOALS

- Efficient utilization of resources
- Convenience (user-friendly)

Dt _____ B+
Pg _____

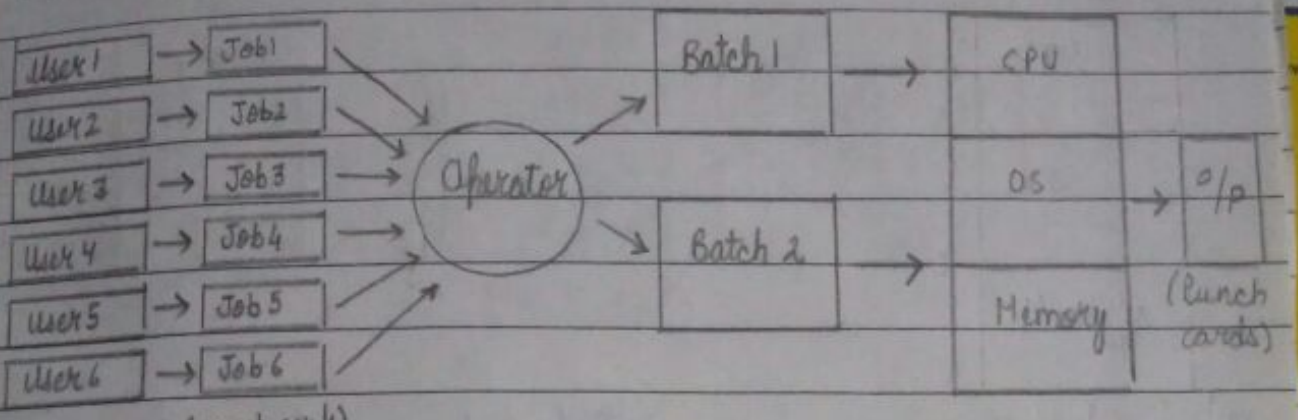
TYPES

eg: job

Calculator Program	
Input : 12, 2	instructions: division

12/2

1) Batch Operating System

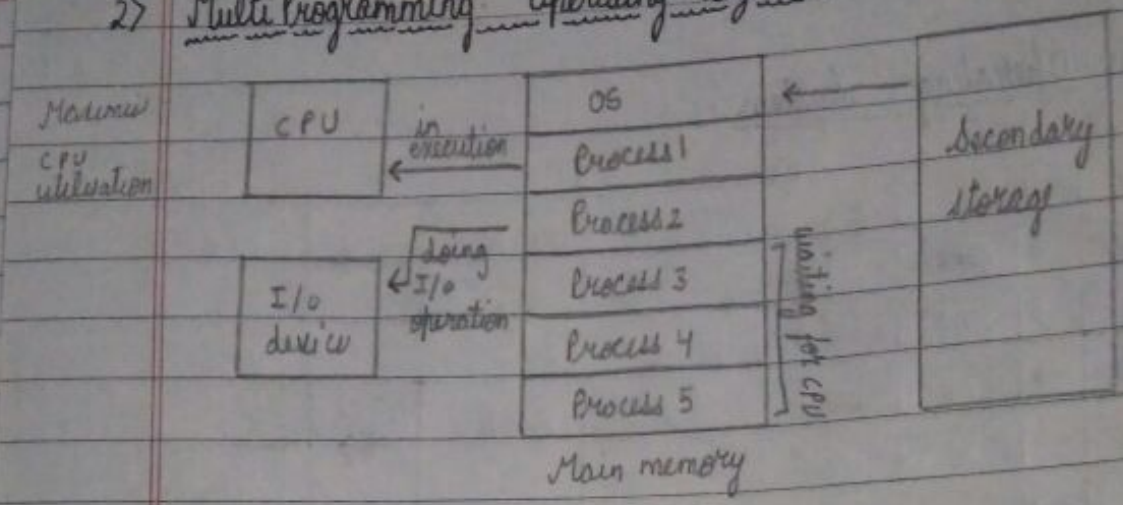


Job: Program + input data + control instructions
Batch: group of similar kind of jobs

→ Each user prepares its job on an off-line device (punch cards and submits it to the computer operator.
→ Computer operator then groups similar kind of jobs to speed up the processing. [All the jobs in a batch required similar resources] → time of allocation/deallocation reduced
→ Batches are fed to the computer one by one. One job is executed at a time. First come first served.
→ Then output is obtained (in the form of punch cards) and returned to the user.

Advantages	Disadvantages
→ Saves time that was being wasted earlier (context switching)	→ CPU remain idle for long time
→ No manual interventon required	→ May lead to starvation
→ Multiple users can share batch system	→ Priority can't be set for jobs
	→ lack of interaction b/w user & job

2) Multiprogramming Operating System



Multiprogramming : Interleaved execution of multiple processes by the same computer

Note : Only one process is executed by a processor at a time

→ In multiprogramming, multiple processes are loaded into the main memory which are ready to execute.
→ If running process performs I/O operations etc., CPU doesn't sit idle rather picks another process for execution.
→ Therefore CPU never sits idle [except at the time of context switching or if no process is ready for execution]

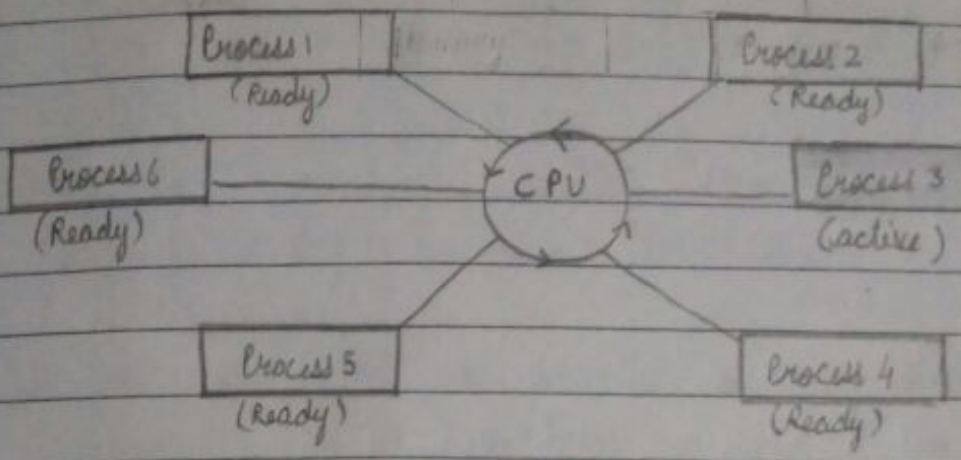
Advantages
- High CPU utilization
- Waiting time is less
- Increased throughput
- priority can be assigned
- Multi-user

Disadvantages
- Scheduling is required
- Main memory managment is required [Memory fragmentation, paging]
- CPU is allocated to another process only if it is idle
- Context switching is required

Dt
Pg B+

3) Multi Tasking Operating System

(Time sharing / fair share / Multiprogramming with Round - Robin)



Multitasking : execution of multiple tasks by sharing processor's time

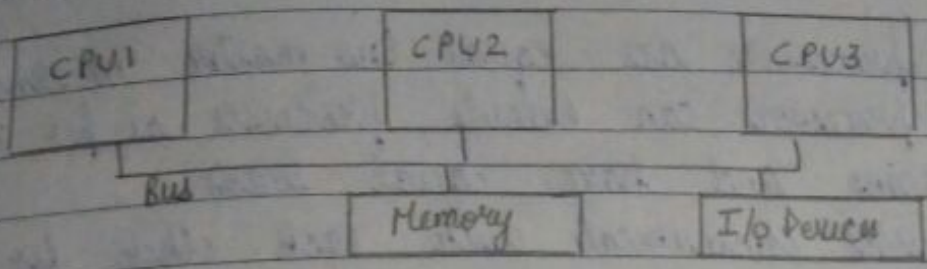
Time quatum / slice / slot : short period of time during which a task gets user CPU attention. (10 - 100 millise

→ Multiple processes are loaded into the memory and CPU switches between them in fixed time intervals (time quantums)
→ Time slot is decided by the OS
→ Context switching is frequent enough to give an expression that all tasks are being executed simultaneo
→ Reduces response time (main goal)

Advantages	Disadvantages
• response time ↓	• context switching overhead
• High CPU efficiency	• Data communication proble
• Throughput ↑	• system must have higher
• Multi - user & user friendly	• specification

Dt
Pg B+

Multiprocessing Operating Systems



Multiprocessing: executing multiple processes simultaneously using multiple processors.

→ Multiple processors work in parallel sharing the resources like memory, bus, peripheral devices, system clock etc (tightly coupled systems)
→ Applications designed for the use in multiprocessing are threaded, so they can be broken into smaller routines that are executed on multiple processors independently
→ Each CPU contains a copy of the OS and these copies communicate with one another to coordinate operations.
→ Used when large data is to be processed at high speed eg: satellite control, weather forcasting etc.

Types:
- Symmetric Multiprocessing OS (SMP OS)
- Asymmetric Multiprocessing OS (ASMP OS)

Adv.:- high speed, time ↓ throughput ↑
- if one CPU fails, others handle the execution
- cost effective [shared resources]

Disadv.:- Process sync required
- Memory management
- CPU scheduling

i) Symmetric Multiprocessing OS

→ All processors are equal (no master - no slave)
→ All processors can execute processes as per their decisions and have equal loads.
→ Processors communicate with each other for the execution if required.

advantages	Disadvantages
- If one processor fails, other handles the execution - No dependency.	- difficult to design

ii) Assymmetric Multiprocessing OS

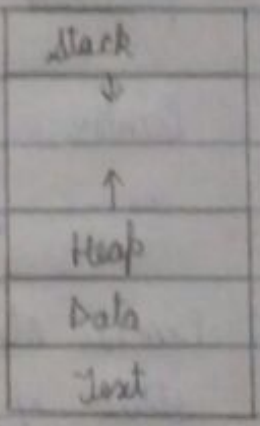
→ It uses the concept of Master-Slave architecture
→ Only one processor work as a primary or master processor which controls the other slave processors.
→ Some processors are assigned maintenance task while others perform application tasks (asymetry)

advantages	Disadvantages
- easy to design.	- If master processor fails, execution stops.

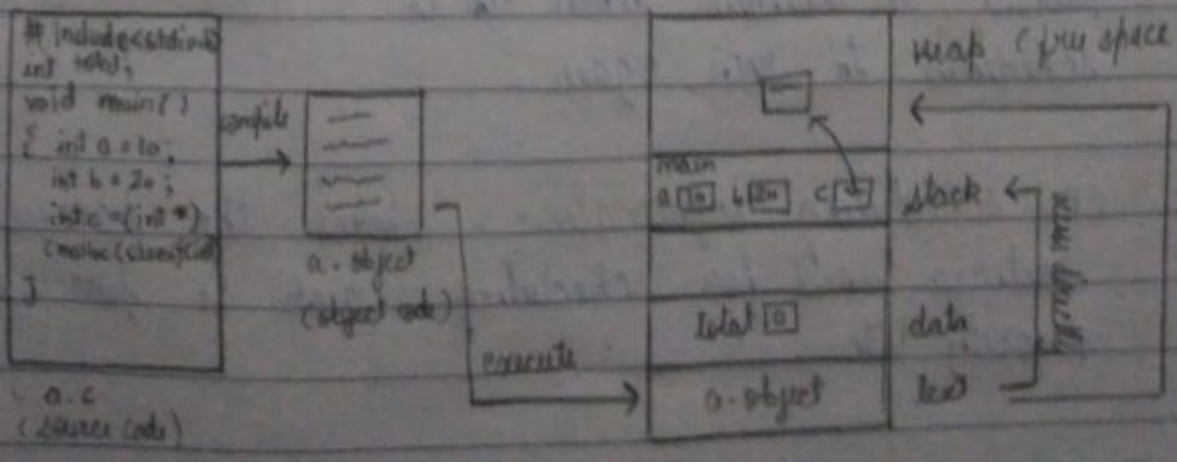
PROCESSES

program (passive unit) : set of instructions that perform a specific task.

Process (active unit) : a program in execution
- It is an instance of a computer program that is being executed by one or many threads.



- **Stack**: contains the temporary data such as function parameters, local variables, return addresses etc.
  [fixed size] (can be accessed directly) — size is determined during compilation & allocated when process begins to execute
  (implemented using stack DS)
- **Heap**: Dynamically allocated memory to a process during runtime [via new, malloc etc.]
  [dynamic size] (accessed using pointers) — size is determined during runtime & allocated during execution
  (NOT implemented using stack DS)
- **Data**: global and static variables
- **Text**: compiled program code [current activity represented by the value of program counter & contents of processor's registers]



Dt
Pg B+

PCB (Process Control Block / Task Control Block)

PCB is a data structure used by OS to maintain all the information about a process
[Brain of a Process]

- Process ID : Unique Id / number provided to each process by the OS
- Pointer : points to parent process
- Process state represents current state of the process : new, ready, running etc.
- Process counter : represents the address of next instruction to be executed
- CPU registers : contains state info of the process which allows the process to continue when its is scheduled to run again (accumulators, index registers, stack pointer etc)
- CPU scheduling information : stores the parameters of criteria used for scheduling such as process priority etc.

Process ID
Pointer
Process state
Process Counter
CPU registers
CPU Scheduling
Memory managment info
Accounting info
I/O status info
Process privileges

- Memory Management info: stores limit register info, page and segment tables etc.
- Accounting info keeps bookkeeping data such as CPU time used, time limits, job numbers etc.
- I/o status info: list of I/o devices allocated to the process, list of open files.
- process priveleges: access-permissions for resources of the system.

Pg B+

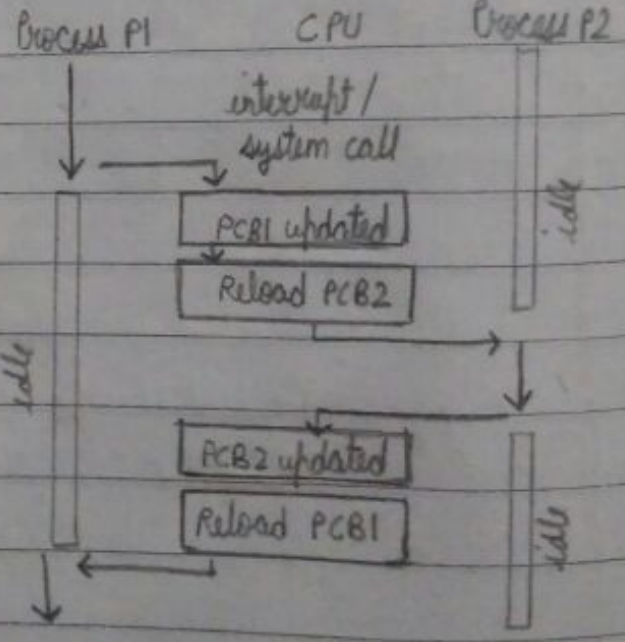
Context Switching remove PCB of running process & place PCB of next process in queue

- It is a process in which CPU switches from one process or task to another
- The state of currently running process is saved (so that it can resume execution later) and another process is alloted to the CPU
- It is used for multitasking
- Context switching occurs so fastly that it gives an illusion that all processes are being executed at the same time

(state of running process is switched)

Steps involved:

- Context of the process P1 (running currently) is saved in the PCB of process 1
- PCB1 is moved to relevant queue (ready queue, I/O queue, waiting queue etc.)
- New process that is to be executed is selected from ready queue
- PCB of process 2 is updated.



Dt
Pg

B+

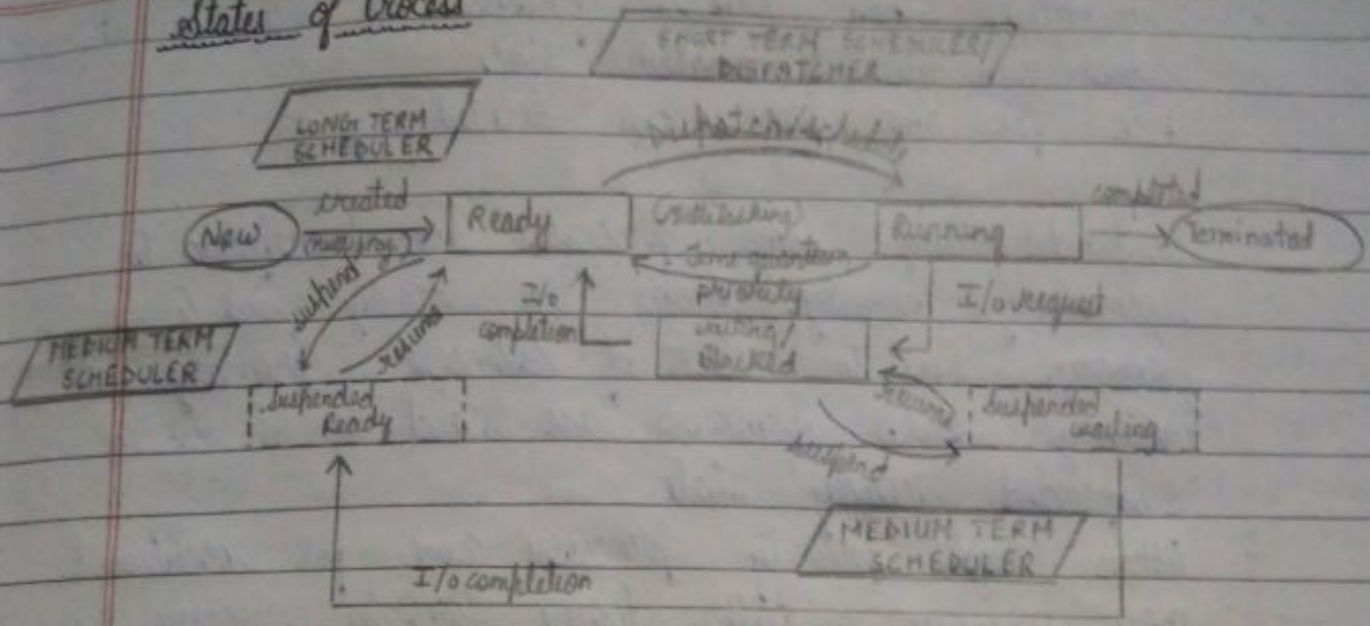
→ Triggers for context switching
- Multitasking
- Interrupt Handling
- User and kernel Mode switching

Advantages
- used for multitasking / multiprograming.
Only 1 CPU is required to run multiple processes.

Disadvantage
- context - switch time is pure overhead (No useful task is performed during context switching)
(takes few milli sec. / switch)

Job Queue
Ready Queue
Waiting Queue

B+



- New: The program which is to be picked up by the OS for execution (from the secondary memory) is said to be in 'new' state
  At a time - multiple processes can be in new state
- Ready: After creation, process is transfered to main memory in ready queue & is said to be in 'ready' state.
  A process in ready state waits for the CPU to be assigned
  At a time - multiple processes can be in ready state
- Running: Once a process is chosen by OS for execution from the ready queue, it enters into 'running' state
  At a time - only 1 process can be in running state (for single processor)
  - n process — (for n processors)
- Waiting / Blocked Whenever a process requires other resources (I/o devices etc.), it enters in 'waiting' or 'blocked' state & waits for the resources (in waiting queue)
  At a time - multiple processes can be in waiting state

- Terminated: When process finishes its execution, it comes in the terminated state. reallocation of resources PCB for the process is deleted by OS.

- Suspended ready: Some processes are transferred to secondary memory from the ready queue (main memory) due to lack of resources (primary memory). Such processes are said to be ~~stt~~ in 'suspended ready' state.
  - When high priority processes come for execution & main memory is full, lower priority processes are transfered to sec. memory.
  - Suspended ready processes wait in the sec memory until space gets available in the main memory & are then transfered to the main memory again.

- Suspended waiting: When processes are transfered to sec. memory from the waiting or blocked state, they are said to be in 'Suspended waiting' state (due to lack of space in main memory)
  A process in suspended waiting state still executes & is transfered to suspended ready state in case I/O op. is completed [& if still not completed & space gets available in waiting queue → returns to the waiting queue]

throughput : turnaround time
waiting time response time

Dt
Pg B+

→ CPU bound processes : processes which spends most of their time on executing the code i.e. take more CPU time.
eg : • complex mathematical algorithms.
• searching / sorting algorithms.
• encryption decryption algorithms

→ I/o - Bound processes : processes which spend most of their time submitting I/o requests and waiting for the response i.e. takes more I/o time.
eg : word processor
[more time is spent in waiting for input & less on CPU - spell checking, auto suggestion etc.]

→ Throughput : Number of processes completed per unit time

point of time → Arrival time : The point of time at which process enters the ready queue / state.

duration → Burst time : time required by a process to get executed on CPU
(execution time / running time / CPU time)

point of time → Completion time : The time at which process completes its execution (exit time)

duration → Turnaround time : Total time spent by a process in the system
TA time = Completion / exit time - arrival time
= Burst time + waiting time

Tutorial

→ Waiting time : time spent by process in ready queue waiting for the CPU.
waiting time = TA time - Burst time

→ Response time : Time between a process enters ready queue and gets scheduled on the CPU for the first time.
RT = time at which process gets CPU for the first time - arrival time.

→ Starvation (indefinite blocking)
Phenomenon in which a process is ready to run but has to wait indefinitely due to lower priority [when CPU is busied]

Dt
Pg B+

Schedulers special system software used to handle process scheduling

1) Long Term Scheduler
selects processes from secondary memory & loads them into main memory (in ready queue) for execution
- controls the degree of multiprogramming [no of processes that can be in ready queue]
- LT scheduler must select a careful mixture of I/O bound & CPU bound processes
  - if CPU bound processes ↑ → I/O devices remain idle
  - if I/O bound processes ↑ → CPU remains idle.

2) Short Term Scheduler
selects a process from the ready queue for execution based on some scheduling algorithm. CPU scheduling
- executes most frequently.
- ensures that, there is no starvation due to high burst time proc

3) Medium Term Scheduler
swaps out a process from main memory [from ready queue / waiting queue state] & places into sec. memory [suspended ready / suspended waiting state] & later swaps in.
- responsible for suspending & resuming a process.
for maintaining balance b/w I/O & CPU bound processes
dec degree of multiprog.

Dispatcher responsible for loading the process selected by ST scheduler on the CPU from ready queue
- context switching

frequency : ST > MT > LT

Dt. _____ B+
Pg. _____

PROCESS - SCHEDULING

It is an OS task which involves allocation of resources and time to the processes in multiprogramming systems.

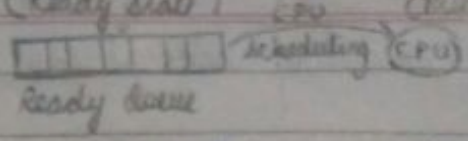
- handles the removal of the running process from the CPU & selection of another process on the basis of a particular algorithm.
- schedules processes in different states - ready, running and waiting

- OS maintains separate queue for each of the process state:

- Job Queue: keeps all the processes ~~tnt~~ in system
- Ready Queue: keeps all the processes residing in main memory, ready to execute
- Device Queue: keeps all the processes which are waiting for I/o devices.

Dt. B+

CPU Scheduling



Process of determining which process will use the CPU while another process is on hold (for maximum CPU utilisation) based on different algorithms [using ST scheduler]

Criteria

- CPU utilization should be maximum $\left(\frac{\text{expected time}}{\text{actual time}} \leq 1\right)$
  ideally - 100%, practically - 40% - 90%
- Throughput should be high
  10/ second
- Avg Turnaround time should be minimum
- Avg waiting time minimum
- Avg response time minimum

CPU Scheduling Algorithms

1> Preemptive Scheduling	2> Non-Preemptive Scheduling
• CPU can be allocated to another process of high priority (or due to time quantum) in between the execution of currently running process.	• CPU can't be allocated to another process until current process completes its execution.
• efficient CPU utilization	• Poor CPU utilization
• WT & RT is less	• WT & RT is high
• overhead of switching the process from ready state to running state & vice versa.	• No such overhead
• CPU is allocated to a process for specific time (may be less than its BT)	• CPU is allocated to a process until it terminates (equal to its BT)
• Round Robin, SRTF, preemptive priority.	• FCFS, non-preemptive SJF, non-preemptive priority
low priority processes may face starvation	may lead to convoy effect

PU Sched. lgo → FCFS, SJF, LJF, Multilevel Queue
SRTF, LRTF, Round Robin, Priority

→ CPU sched ✗
A1 → —— ✗
P2 → —— ✓

B+

FIRST COME FIRST SERVE (FCFS)

[available for batch OS]

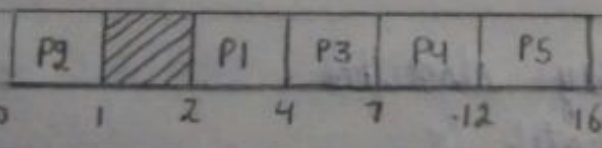
(simplest algorithm)
assigns CPU to the process which arrives first into the ready queue
- Non-preemptive scheduling (WT = RT)
[as soon as a process gets response - doesn't have to wait anymore - completes & exits]

Example:
(Assumption : I/o time is not required for any process)

Process	Arrival time (AT)	Burst time (BT)	completion time (CT)	CT - AT Turnaround time (TAT)	TAT - BT Waiting time (WT)	Response time (RT)
P1	2	2	4	4-2 = 2	2-2=0	0
P2	0	1	1	1-0 = 1	1-1=0	0
P3	2	3	7	7-2 = 5	5-3=2	2
P4	3	5	12	12-3 = 9	9-5=4	4
P5	4	4	16	16-4 = 12	12-4=8	8
				29	14	14

*[if more than 1 process arrive at same time - any of them can be taken]

Gantt chart :



→ average TAT : 29/5 = 5.8
→ average WT : 14/5 = 2.8
→ average RT : 14/5 = 2.8
→ throughput : 5/16

Convey Effect

It is a phenomenon associated with FCFS in which processes of higher burst time comes before the processes with lower burst time resulting in high avg waiting time.

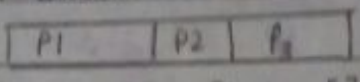
a CPU bound process followed by I/o bound processes

Case 1

Process	AT	BT	TAT	WT
P_1	0	50	50	0
P_2	1	1	50	49
P_3	1	2	52	50
				99

starvation: wait due to priority

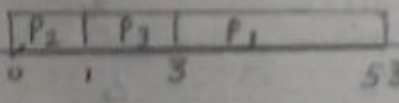
gantt chart:



avg waiting time = 99/3 = 33

Process	AT	BT	TAT	WT
P_1	1	50	52	2
P_2	0	1	1	0
P_3	0	2	3	1
				3

gantt chart:



avg waiting time = 3/3 = 1

Case 1 suffers from convey effect.

Adv: - simple & easy algorithm
- suitable for batch OS

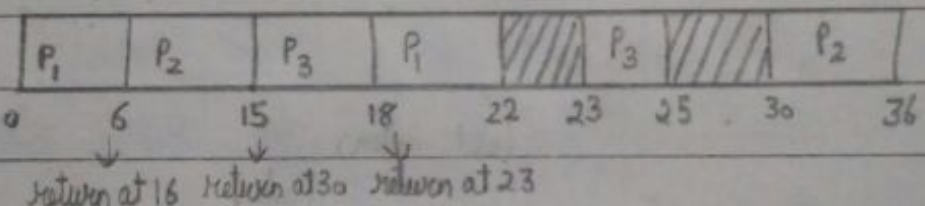
Disadv: - convey effect
- non-preemptive nature - CPU utilisation is low.
- doesn't respond according to priority of process

Dt
Pg B+

→ Three processes P_1, P_2, P_3 with process time 20, 30, 10 respectively. Each process uses first 30% of its process time on CPU then 50% in I/O & last 20% on CPU.
Find avg WT, TAT, RT if FCFS scheduling is followed.

Process	PT	AT	BT	I/OT	BT	CT	TAT	WT	RT
P_1	20	0	6	10	4	22	22	12	0
P_2	30	0	9	15	6	36	36	21	6
P_3	10	0	3	5	2	25	25	20	15
							83	53	21

gantt chart:



avg. RT = 21/3 = 7
avg. WT = 53/3 = 17.66̄
avg. TAT = 83/3 = 27.6̄
throughput = 3/36 = 1/12
CPU utilization factor = $\frac{30}{36}$ = 0.833 = 83.3%

CPU idle time factor = $\frac{6}{36} = \frac{1}{6}$ = 16.66%

Dt
Pg B+

SHORTEST JOB FIRST (SJF)

assigns CPU to the process which has smallest burst time out of all the processes present in ready queue
- effectively reduces the average waiting time.

- Preemptive SJF (SRTF)
- non-preemptive SJF (normal)

Non-preemptive SJF

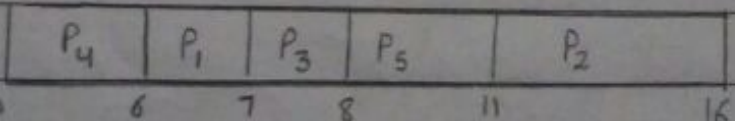
Once CPU is assigned to a process, it can't be snatched by another process, even if its burst time is smaller.

Example:
* I/o time X

Process	AT	BT	CT	TAT	WT	RT
P_1	2	1	7	5	4	4
P_2	1	5	16	15	10	10
P_3	4	1	8	4	3	3
P_4	0	6	6	6	0	0
P_5	2	3	11	9	6	6
				39	23	23

* [if more than one process have same BT → follow FCFS] P_4, P_1, P_3, P_5

gantt chart:



throughput : 5/16
avg waiting time : 23/5 = 4.6
avg response time : 4.6
avg TAT : 39/5 = 7.8

Tutorial

Preemptive SJF
(SRTF: Shortest Remaining Time First)

CPU can be allotted to another process if its burst time is smaller than currently running process the remaining time of

Example:

Process	AT	BT	CT	TAT	WT	RT
P_1	2	1	3	1	0	0
P_2	1	5	16	15	10	10
P_3	4	1	5	1	0	0
P_4	0	6 5 4	11	11	5	0
P_5	2	3 2	7	5	2	1
				33	17	11

P_4, P_2, P_1, P_5, P_3

gantt chart:

P_4	P_4	P_1	P_5	P_3	P_5	P_4	P_2

0 1 2 3 4 5 7 11 16

avg. TAT = 33/5 = 6.6
avg. WT = 17/5 = 3.4
avg. RT = 11/5 = 2.2
throughput = 5/16

If arrival time is same for all processes
non preemptive SJF ≈ preemptive SJF

Dt. _____ B+
Pg. _____

Adv. :
- min. avg. WT & TAT (SRTF) than all other algo
- better response time
- max. throughput (SRTF)
- provides a standard for other algo [for WT] used in batch OS

optimal when arrival time is same for all processes. (no convey effect)

Disadv. :
- can not be implemented bcz it depends on BT of the processes which is usually not known beforehand. (approx BT can be determined)
- may lead to starvation of the processes with larger burst time. [Larger BT process may have to wait for ∞
- convey effect in non-preemptive SJF
- priorities can't be set.
- poor response time for processes with larger BT

Prediction of burst time

To implement SJF algo, BT needs to be known before. Following techniques are used to predict BT:

→ Static Techniques

- Process size
  If P_{old} (100 kB) - 5 units
  ∴ P_{new} (102 kB) ≃ 5 units
  As their size is approximately same, their BT is predicted to be equal.

- Process type
  - OS process (system process)
  [schedulers, compilers, program managers etc.]
  lower BT - 3 to 5 units

  - User process
    - interactive process I/O bound
      lower BT
    - foreground process I/O bound + CPU bound
      slightly higher BT MS office, editors
    - background process CPU bound
      higher BT key logger

→ Dynamic Techniques

- Simple Averaging

P_1 P_2 P_3 P_4 P_5

$$P_6 = \frac{t_1 + t_2 + t_3 + t_4 + t_5}{5}$$

predicted burst time of $(i+1)$th process $= \frac{1}{n}\sum_{I=1}^{n} T(i)$

- Exponential Averaging

$$\boxed{\varepsilon_{i+1} = \alpha A_i + (1-\alpha)\varepsilon_i}$$

ε_{i+1} = expected time for process $i+1$

ε_i = expected time for process i

A_i = Actual burst time of process i

α = smoothening factor

$0 \leq \alpha \leq 1$

- If $\alpha = 0$ $\quad \varepsilon_{i+1} = \varepsilon_i$ expected BT of previous process
- If $\alpha = 1$ $\quad \varepsilon_{i+1} = A_i$ Actual BT of previous process
- If $\alpha = 1/2$ equally weighted - ε_i & A_i

eg: $\alpha = 0.5$ $\quad \varepsilon_1 = 5$ $\quad \varepsilon_5 = ?$

Process	BT	ε_i
P_1	4	5
P_2	8	4.5
P_3	5	6.25
P_4	6	5.625

$E_2 = 0.5(4) + 0.5(5)$

$= 2.0 + 2.5 = 4.5$

$\varepsilon_3 = 0.5(8) + 0.5(4.5)$

$= 4.0 + 2.25 = 6.25$

$E_4 = 0.5(5) + 0.5(6.25)$

$2.5 + 3.125 = 5.625$

$\varepsilon_5 = 0.5(6) + 0.5(5.625)$

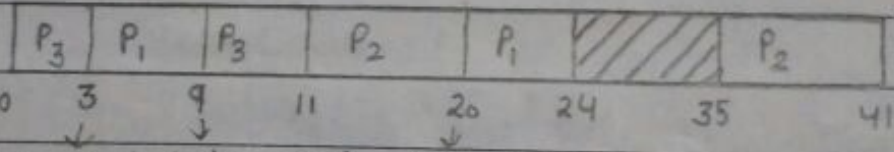
$3.0 + 2.8125 = 5.8125$

Dt _____ B+
Pg _____

→ Three processes P_1, P_2, P_3 with process time 20, 30, 10 respectively. Each process uses its first 30% of the process time on CPU, then 50% in I/O & last 20% on CPU. Find avg WT, TAT, RT, CPU utilisation % if system follows SJF scheduling non-preemptive by default

Process	PT	AT	BT	I/O T	BT	TAT	WT (TAT - BT)	RT
P_1	20	0	6	10	4	24	14	3
P_2	30	0	9	15	6	41	26	11
P_3	10	0	3	5	2	11	6	0
						76	46	14

gantt chart:



avg WT = 46/3 = 15.3

avg TAT = 76/3 = 25.3

avg RT = 14/3 = 4.6

CPU utilization factor = $\frac{\text{expected CPU time}}{\text{actual CPU time}} = \frac{30}{41} = 0.7317 \approx 73\%$ 100%

CPU idle time factor = $\frac{\text{CPU idle time}}{\text{total CPU time}} = \frac{11}{41} = 0.2683 \approx 27\%$

throughput = 3/41

Dt
Pg

B+

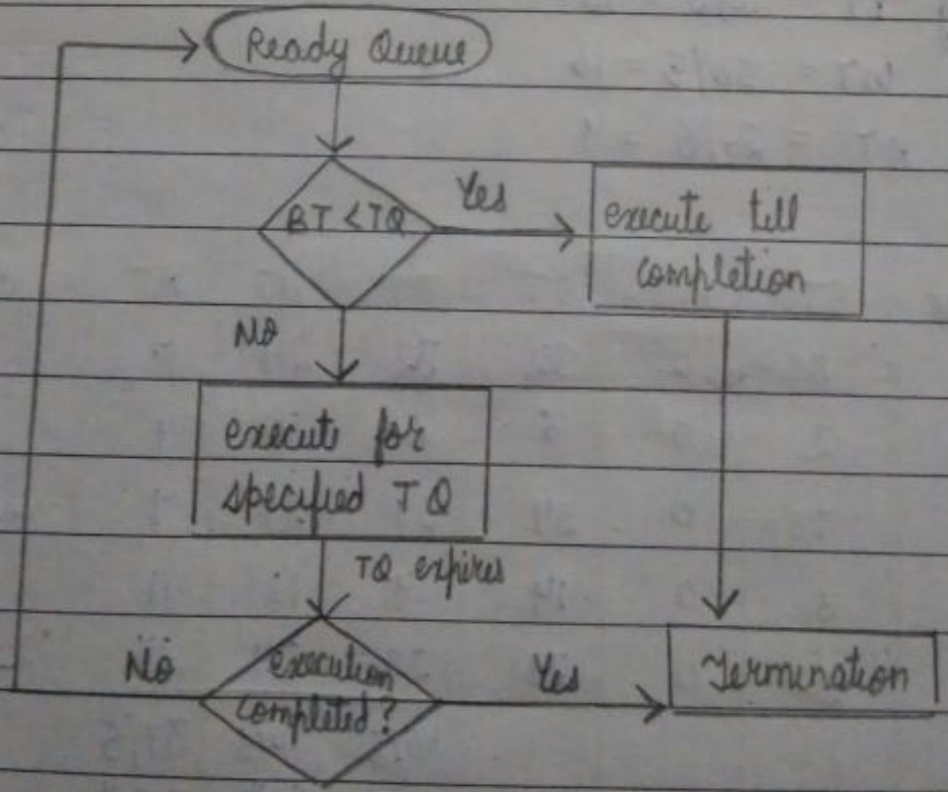
ROUND ROBIN [FCFS + time quantum]

assigns time slices to each process in equal portions (in circular order) [starvation free] [response T ↓]

- no priorities
- suitable for time sharing OS
- preemptive

Time quantum: period of time for which process is allowed to execute uninterruptedly in preemptive multitasking (10 - 100 ms)

- Time quantum must be selected carefully
  - if too small - overhead of context switching
    - waiting time ↑
  - if too large - ends up as FCFS scheduling.



Example:

Process	AT	BT	CT	TT	WT	RT
✓ P_1	0	8 5 2	22	22	14	0
✓ P_2	5	2	11	6	4	4
✓ P_3	1	7 4 1	23	22	15	2
✓ P_4	6	3	14	8	5	5
✓ P_5	8	5 2	25	17	12	9
				75	50	20

TQ = 3

P_1	P_3	P_1	P_2	P_4	P_3	P_5	P_1	P_3	P_5

Ready Queue

gantt chart:

P_1	P_3	P_1	P_2	P_4	P_3	P_5	P_1	P_3	P_5

0 3 6 9 11 14 17 20 22 23 25

avg TT = 75/5 = 15

avg WT = 50/5 = 10

avg RT = 20/5 = 4

Process	BT	AT	CT	TT	WT	RT
✓ P_1	8 5 2	0	32	32	24	0
✓ P_2	2	0	6	6	4	4
✓ P_3	7 4 1	0	34	34	27	7
✓ P_4	3	0	14	14	11	11
✓ P_5	5 2	0	29	29	24	15
				115/5 = 23	90/5 = 18	37/5 = 7.4

- context switch time = 1 unit
- TQ = 3 units
- no. of context switches = 9
- CPU utilization: $\frac{25}{34} = 0.735$

= 73.5 %

Ready Queue:

P_1	P_2	P_3	P_4	P_5	P_1	P_3	P_5	P_1	P_3

gantt chart:

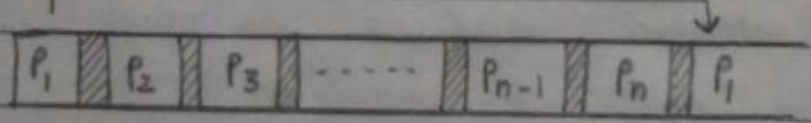
P_1		P_2		P_3		P_4		P_5		P_1		P_3		P_5		P_1		P_3

0 3 4 6 7 10 11 14 15 18 19 22 23 26 27 29 30 32 33 34

Adv. : - easy & simple to implement BT not required as in SJF
- each process gets a fair share of CPU (same priority for all processes)
- starvation free
- no convoy effect
- diminishes response time
- suitable for multitasking

Disadv. : - deciding size of TQ is very tough
(if not decided properly may lead to WT ↑, TAT ↑, RT ↑, throughput ↓, CPU utilization ↓)
- overhead of context switching
- priorities can't be set.

→ Let there are n processes in a system. Context switch time is S seconds. What should be the max time quantum so that any process has to wait for atmost T seconds to enter in CPU again



$$[t_q * (n-1) + n * S] \leq T$$

$$t_q \leq \frac{T - nS}{n-1}$$

Dt
Pg B+

PRIORITY SCHEDULING

when loaded in ready queue

A priority is associated with each process. CPU is assigned to the process having higher priority.
- Priority is decided based on memory requirements, time requirements etc.
- In case of tie, FCFS is followed.
- Priorities can be
  - static : doesn't change throughout the execution
  - dynamic : changes after some fixed interval of time

Preemptive Priority Scheduling:

CPU can be assigned to a process with higher priority in the middle of execution of a process with lower priority.

Example:

Lower the no, higher the priority

Process	Priority	AT	BT	CT	TT	WT	RT
P_1	3	0	~~8~~ 3	15	15	7	0
P_2	4	1	2	17	16	14	14
P_3	4	3	4	21	18	14	14
P_4	5	4	1	22	18	17	17
P_5	2	5	~~6~~ 1	12	7	1	0
P_6	6	6	5	27	21	16	16
P_7	1	10	1	11	1	0	0
					96/7	69/7	61/7
					=13.7	=9.8	=8.7

gantt chart :

P_1	P_1	P_1	P_1	P_5	P_5	P_7	P_5	P_1	P_6	P_3	P_4	P_6
0	1	3	4	5	6	10	11	12	15	17	21	22	27

Tutorial

Non-Preemptive Priority Scheduling:

Lower no → higher priority

Process	Priority	AT	BT	CT	TT	WT	RT
P_1	3	0	8	8	8	0	0
P_2	4	1	2	17	16	14	14
P_3	4	3	4	21	18	14	14
P_4	5	4	1	22	18	17	17
P_5	2	5	6	14	9	3	3
P_6	6	6	5	27	21	16	16
P_7	1	10	1	15	5	4	4
			(27)		95/7	68/7	68/7
					13.5	9.7	9.7

P_1 P_2 P_3 P_4 P_5 P_6 P_7

gantt chart:

P_1	P_5	P_7	P_2	P_3	P_4	P_6

0 8 14 15 17 21 22 (27)

Adv. priorities can be set

Disadv. - starvation of processes with lower priority.
- deciding priorities is tough

once in 1967, IBM 7094 used priority sch. - a process waited till 1973

AGING

method to prevent starvation
Gradually priority of processes is inc. after fixed intervals of time.

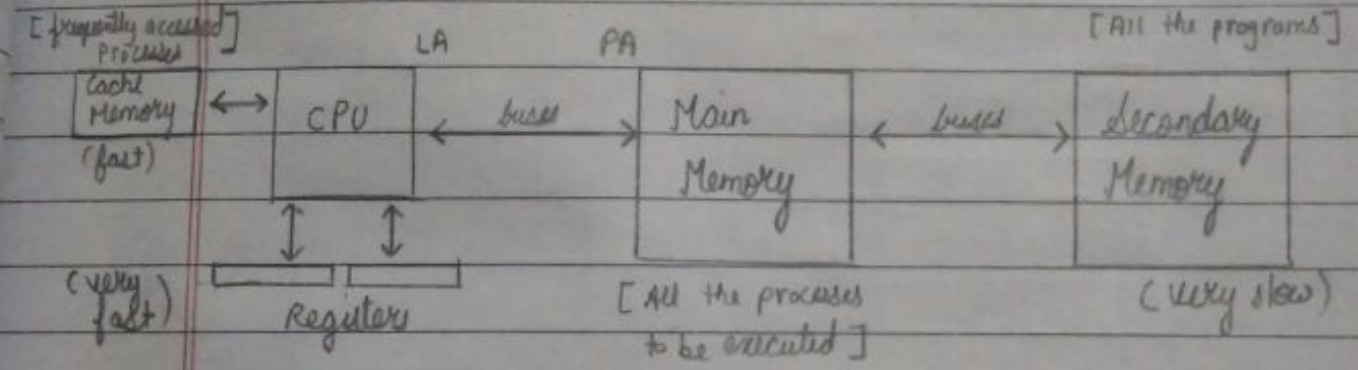
Dt
Pg B+

MEMORY MANAGEMENT

- It is a functionality provided by operating system which manages the primary memory of computer system

Tasks
- Moves processes back & forth between main memory & disk
- keeps track of memory allocation / deallocation
- Security: prohibits user program to enter in os area & other user area.

goals
- maximise memory utilization
- maximise CPU utilization
- security to user programs & OS



degree of multiprogramming
no. of processes that can be accommodated in main memory at a time for execution.
- degree of multiprog. ↑ , CPU utilization ↑
  depends on size of RAM

Example:

RAM size = 4MB Process size = 4MB
degree = 1 idle time = k (75%)
CPU utilisation = 1 − k (25%)

RAM size = 8MB Process size = 4MB
degree = 2 idle time = k^2
CPU utilisation = $1-k^2$ 76%

RAM size = 16MB Process size = 4MB
degree = 4 idle time k^4
CPU utilisation = $1-k^4$ (94%)

Tutorial

Memory Allocation Techniques

- Contiguous
  - Fixed partitioning
    - equal sized
    - variable sized
  - Dynamic partitioning
- Non Contiguous
  - Paging
  - Multilevel Paging
  - Inverted Paging
  - Segmentation
  - Segmented Paging

Contiguous

- Contiguous memory allocation allocates consecutive blocks of memory to a process
- Faster execution of process
- Not ~~one~~ much address translations ∴ less overhead
- process need not be divided into blocks
- A table is maintained by OS that contains all available & occupied partitions in memory
- Wastage of memory

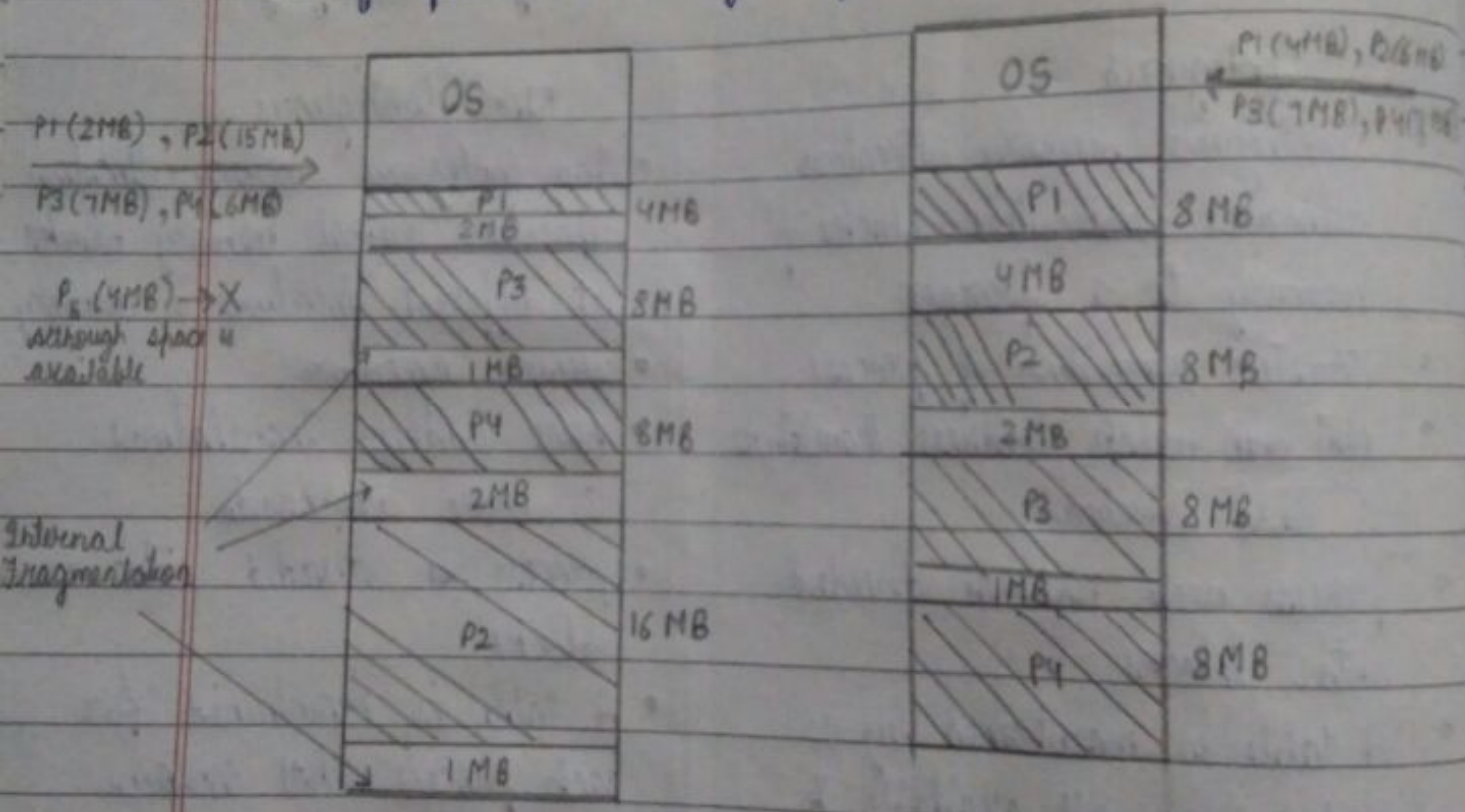
Non-Contiguous

- Non Contiguous memory allocation allocates several memory blocks at different locations in memory
- slower execution
- More address translations ∴ more overheads
- process is divided into various blocks
- A table is maintained for each process that contains base addresses of each block acquired by the process in memory
- less wastage

Fixed Partioning

no. of size of partitions is decided in advance before the process is loaded

- oldest & simplest technique
- Memory is divided into fixed-size partitions.
- No of partitions are fixed
- Size of partition may or may not be same
- spanning is not allowed
- A partition is allocated to a single process only
- No of partitions = degree of multiprogramming



Equal size Partitioning Unequal size Partitioning

Internal Fragmentation (memory wastage)

When the process is allocated to a memory block whose size is more than that of process & due to which some part of the memory is left unused.

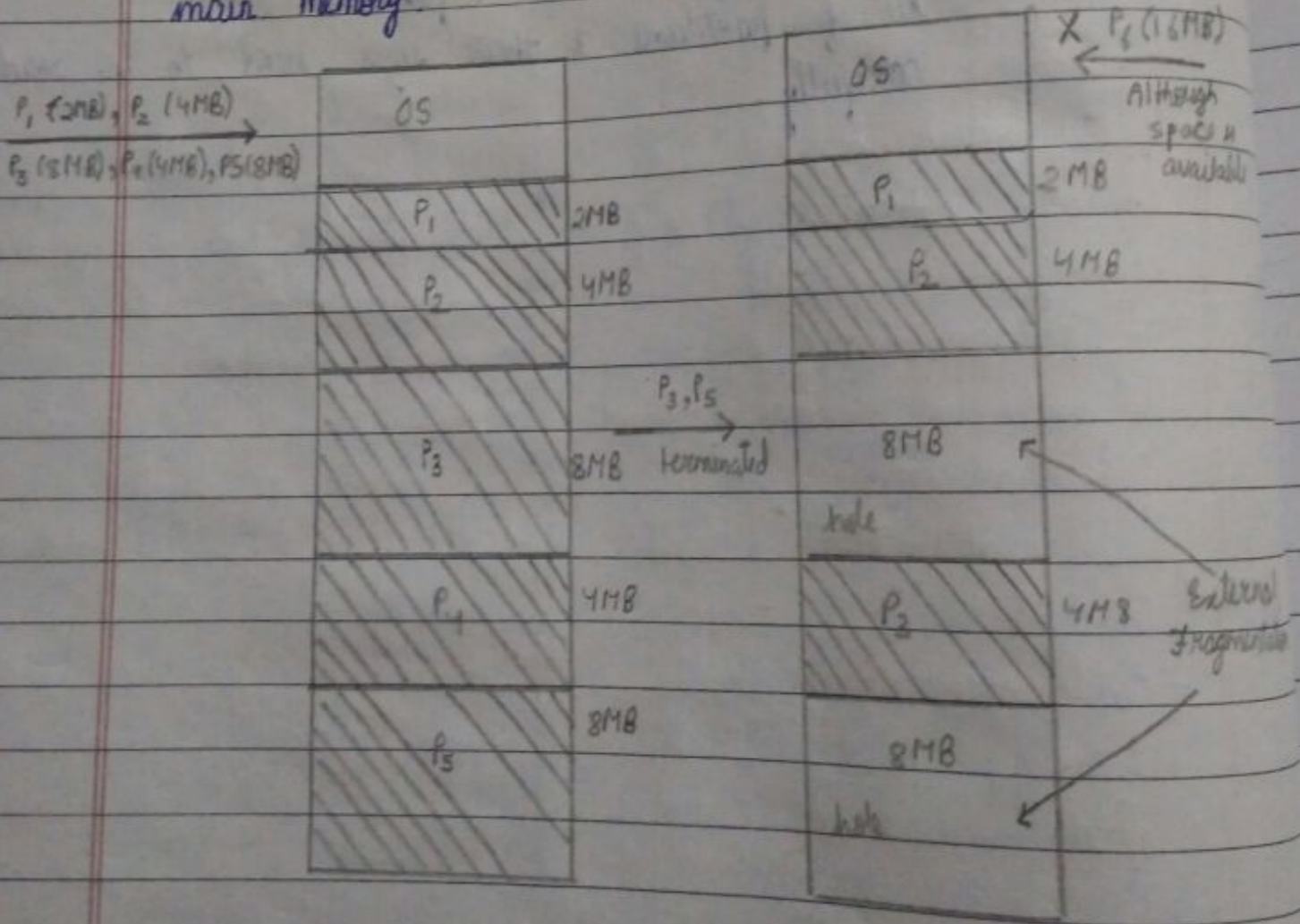
Dt
Pg B+

Adv. • Easy to implement

Disadv. • Internal Fragmentation → external fragmentation
• Limitation to size of the process
(processes which are larger than the size of partitions are not allocated the memory)
• limitation to the degree of multiprogramming
(= no. of partition → fixed)
• no of partitions & their sizes need to be decided carefully.

Dynamic Partitioning

- Initially RAM is not partitioned.
- Partitions are created at the time of process loading.
- Size of partition is equal to the incoming process (this avoids internal fragmentation)
- No of ~~processes~~ or size of partitions is not fixed & depends on the size of incoming processes & size of main memory.



External fragmentation

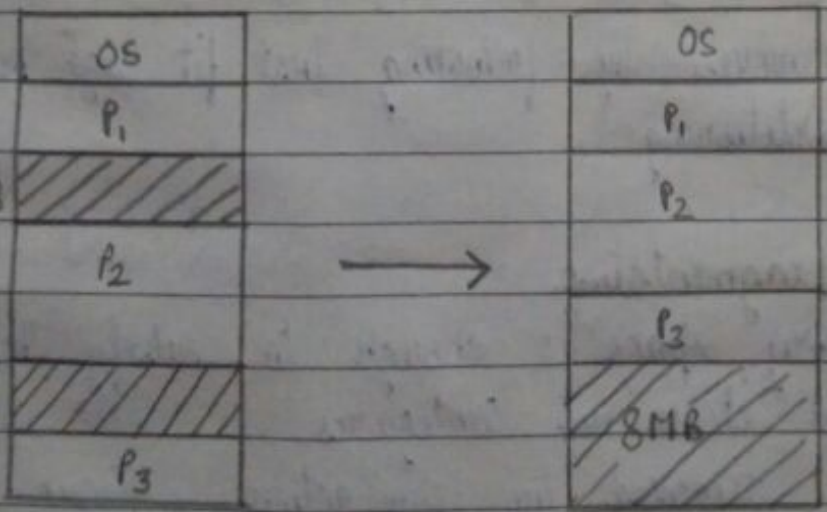
When the memory space is sufficient but can't be utilised because it is not contiguous.

holes are present in memory & their size ↓ over the time

Adv.
- No internal fragmentation
  (bcz partition size = process size)
- No limitation on size of processes
  (bcz partitions are made dynamically in accordance with the size of process)
- No limitation on degree of multiprogramming
  (bcz no of processes aren't fixed)

Disadv.
- External fragmentation
- Allocation & deallocation of memory is complex.

Compaction (defragmentation)
It is a process of combining all the free spaces together to make it contiguous & avoid external fragmentation.



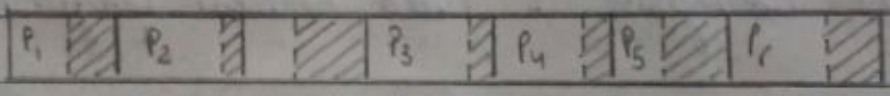
drawback: efficiency of the system is decreased. Huge amount of time is invested & CPU remains idle for this time.

Fragmentation

When processes are loaded & removed from the memory, the free memory space is broken into smaller pieces. Due to the small size, these memory fragments can't be allocated which leads to memory wastage. This is called fragmentation

→ Internal Fragmentation allocated space is wasted

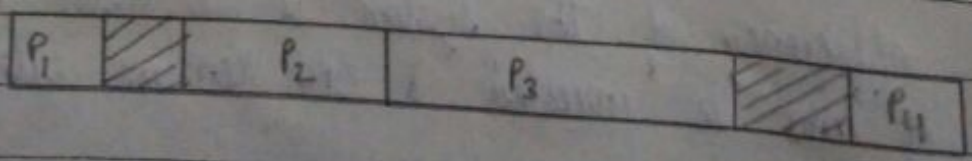
- When memory block assigned to a process is bigger than the process size, left over portion of the block remains unused.
- It usually happens when a block / partition is allocated to a single process & spanning is not allowed.



- It can be removed by following best fit algo or dynamic partitioning.

→ External Fragmentation unallocated space is wasted

- Total memory space is enough to satisfy the memory request but it is not contiguous.
- It can be removed by compaction, paging or segmentation



B+

Algorithms (for unequal fixed size / dynamic partitioning)

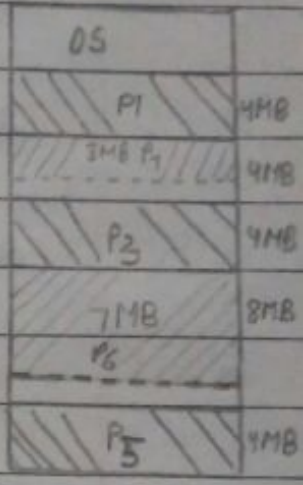
In fixed & dynamic partitioning, free partitions and holes are allocated processes based on some algorithms.

1) **First - Fit**

Process is allocated the first available memory ~~with~~ (unallocated partition / hole) with space more than or equal to it's size.

adv: fast

disadv: memory wastage (not efficient)

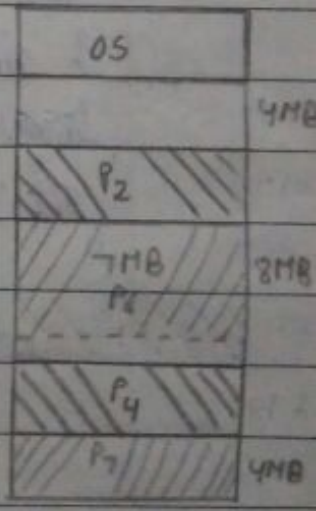


2) **Next - Fit** (modified - first-fit)

Process is allocated the first available memory with space more than or equal to its size, starting from the last allocated memory space.

- uses a moving pointer to track
- last allocated memory space.

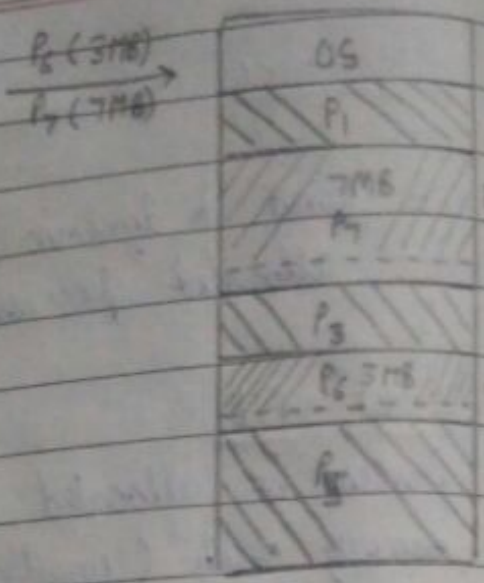
faster.



3) Best Fit

searches the whole memory acc to the size of process & allocates it to the closest-fitting free partition / hole in the memory

- keeps a list of free memory (smallest - largest)

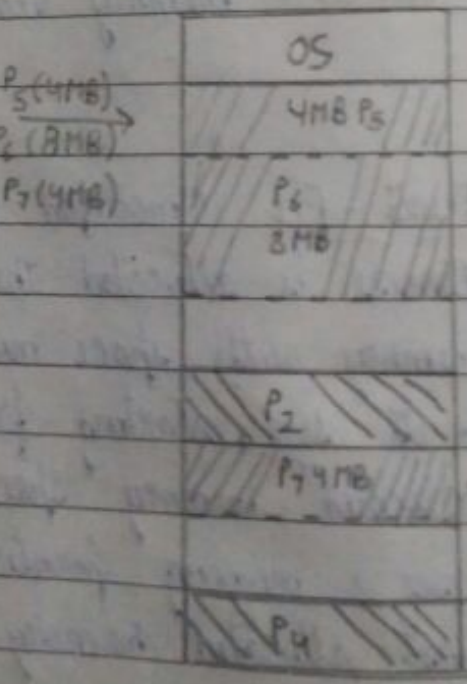


adv: • memory efficient
(less memory is wasted)

disadv: • slow
• reduces holes to much smaller size which can are very rarely occupied afterwards.

4) Worst Fit

searches the whole memory & allocates the process to largest sufficient partition among all the freely available partitions



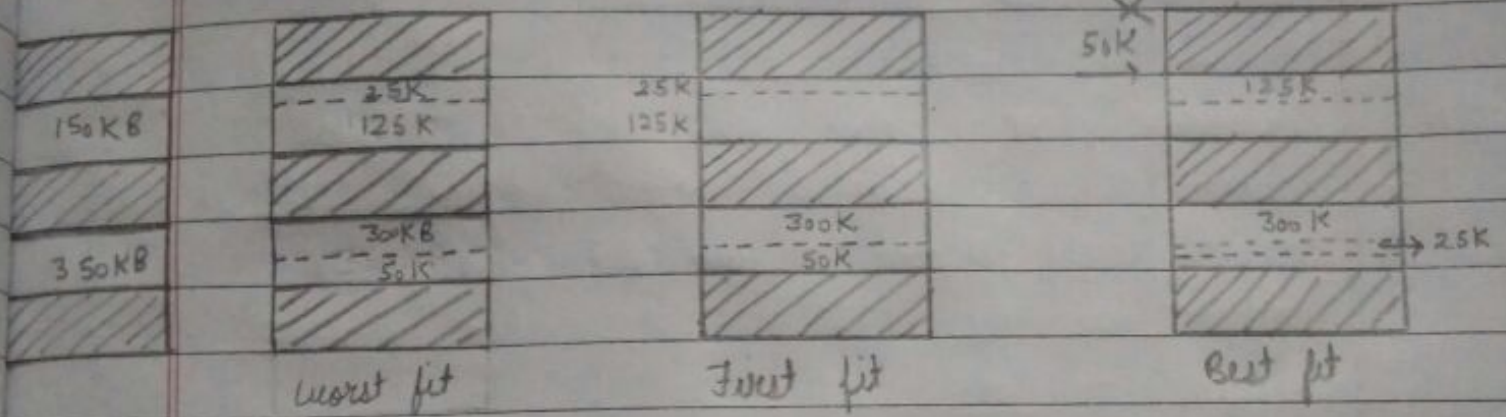
adv: remaining holes are large enough to accomodate other processes.

disadv: if large process comes later, it can't be accommodated.

Dt
Pg B+

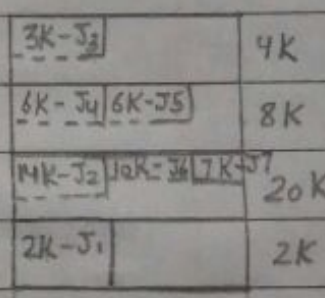
Questions:

→ 300K, 25K, 125K, 50K are requested respectively
Above requirement could be satisfied with
a) Best Fit only b) First fit only c) both d) None



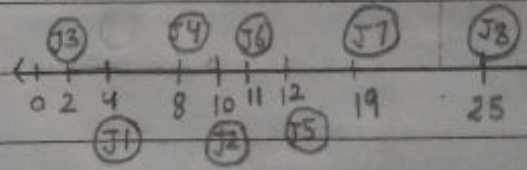
→

enter time	0	0	0	0	8	10	11	
Request No	J1	J2	J3	J4	J5	J6	J7	J8
Request Size	2K	14K	3K	6K	6K	10K	7K	20K
Usage Time	4	10	2	8	4	1	8	6



"best fit" arrival time = 0

Calculate the time at which J_7 will be completed a) 17 b) 20 c) 19 d) 37



Tutorial

B+

DL
PG

PAGING

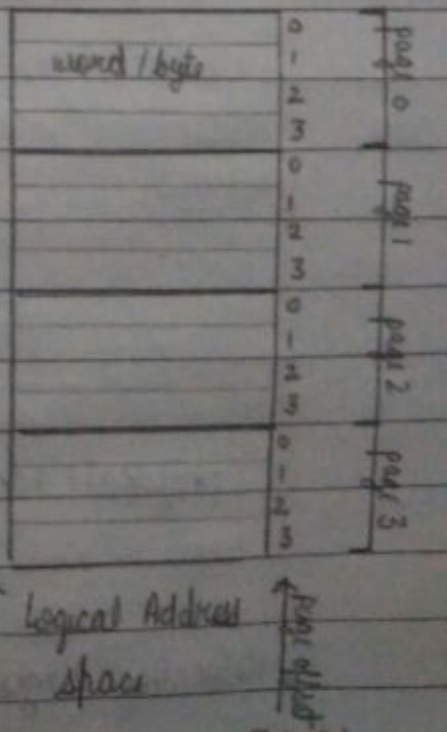
- It is a non-contiguous memory allocation technique
- In this technique,
  - logical address space is divided into pages (fixed-size)
  - physical address space is divided into frames (fixed-size)
  - size of page = size of frame
    (for max memory utilisation & avoid fragmentation)
- A process is divided into pages & stored in frames in main memory
  (Pages can be stored at different locations in non-contiguous frames)

(in reference to the secondary memory)

→ Logical Address space
- set of all the logical addresses generated by the CPU
- It is a virtual / Conceptual space which doesn't exist in memory

Logical address: It is a virtual address generated by CPU during program execution.
It is used as a reference to access the physical memory

- Logical address: | page no. (p) | page offset (d) |
- For n bit page no., no. of pages = 2^n ; $n = 2$, 4 pages
- For m bit page offset, no. of words / page = 2^m ; $m = 2$, 4 words/page
- no. of bits in logical address = no. of bits in page no. + no. of bits in p offset = $n + m$
  size of logical address space = 2^{n+m} (total words) ; $2^{2+2} = 16$ words



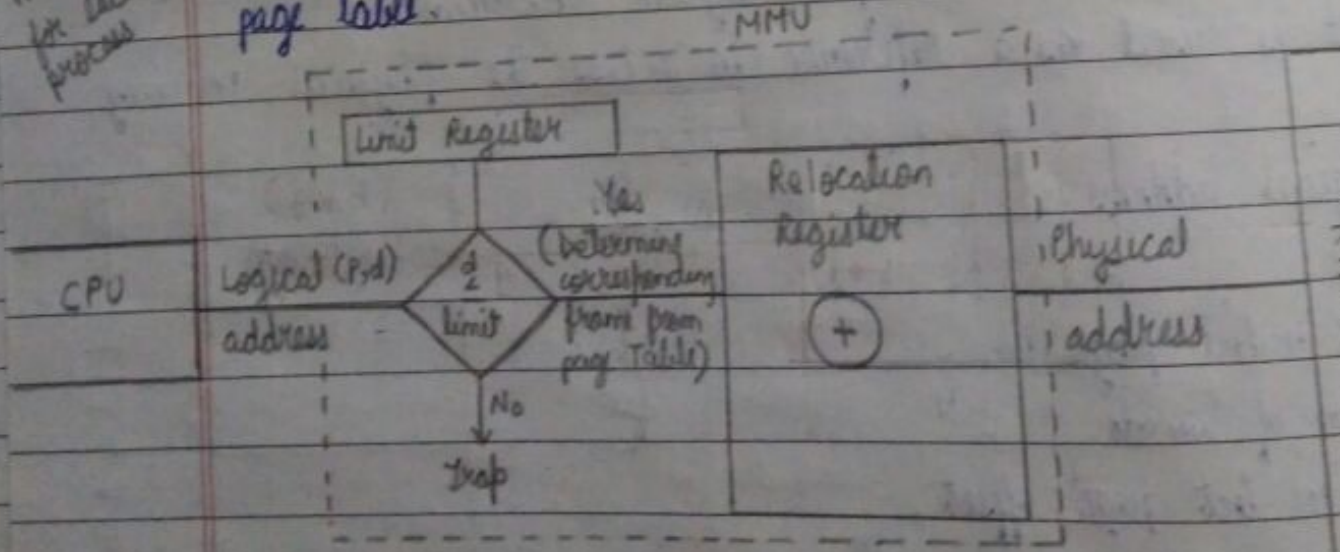
→ Physical Address Space (In reference to the ...)
set of all the physical addresses corresponding to logical addresses in logical space
It actually exists in the memory

physical address | frame no (f) | frame offset (d)

Note: The user never directly deals with the physical address. CPU generates the logical address & program runs on the logical addresses. But actually logical addresses are mapped to the corresponding physical addresses by MMU

MMU (Memory Management Unit)
It is a hardware device responsible for converting logical addresses into physical addresses using a page table.

A page table is maintained for each process



physical address = frame no * size of frame + offset
= f * n + d

Relocation Register: special register in CPU used to map logical addresses to physical addresses

Dt
Pg
B+

Example

→ Logical Memory Space

0	a	0
	b	1
	c	2
	d	3
1	e	0
	f	1
	g	2
	h	3
2	i	0
	j	1
	k	2
	l	3

Page Table

Page No.	Frame No.
0	0
1	2
2	4

Physical Memory Space

0	0	a	0
	1	b	1
	2	c	2
	3	d	3
1	4		0
	5		1
	6		2
	7		3
2	8	e	0
	9	f	1
	10	g	2
	11	h	3
3	12		0
	13		1
	14		2
	15		3
4	16	i	0
	17	j	1
	18	k	2
	19	l	3

(limit)
frame size = page size = 4 words
Memory size = 20 words
no. of frames = $\frac{20}{4} = 5$

process size = 12 words
no of pages = $\frac{12}{4} = 3$

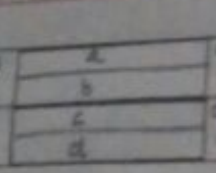
logical address = 2 bits + 2 bits = 4 bits
physical address = 3 bits + 2 bits = 5 bits

$(p,d) = (1,2)$ [g] in logical space
$2 \leq 4$ ✓
$f = 2$ $(f,d) = (2,2)$
physical address = $2 * 4 + 2 = 10$ [g] in physical space

$(p,d) = \underbrace{(2,0)}_{\text{logical add}} \rightarrow 0 \leq 4 \xrightarrow{\text{True}} \underbrace{(4,0)}_{\text{physical add}} \rightarrow 4 * 4 + 0 = 16$

$2^2 = 4$ $2^7 = 128$ $2^{20} = 1M$
$2^3 = 8$ $2^8 = 256$ $2^{30} = 1G$
$2^4 = 16$

→



Logical address space

page no	frame no
0	2
1	4

page table

physical address space

frame size = page size = 2
memory size = 16
no. of frames 16/2 = 8
process size = 4
no. of pages = 4/2 = 2

1bit 1bit 3bits 1bit

1	1	→	1	0	0	1

(2 → d) (4 → d)

0	1	→	0	1	0	1

(1 → b) (5 → b)

Logical address = 1 bit + 1 bit
= 2 bits

Physical address = 3 bit + 1 bit
= 4 bits

→ LAS = 4 GB
PAS = 64 MB
Page size = 4KB

no of pages? no of frames? size of LA? PA?
no. of entries in page table?
size of page table?

LA

20B	12B

PA

14B	12B

no. of pages = $\frac{\text{logical address space}}{\text{page size}} = \frac{4GB}{4KB} = \frac{2^{2+30}\ B}{2^{2+10}\ B} = 2^{20}$

no. of bits req. for page no = 20 bits

no. of frames = $\frac{\text{physical address space}}{\text{frame size}} = \frac{64MB}{4KB} = \frac{2^{6+20}\ B}{2^{2+10}\ B} = 2^{14}$

no. of bits req for frame no = 14 bits

no. of bits req for logical address ⇒ $4GB = 2^{32}$ → 32 bits

or bits for page no. + bits for offset, $4KB = 2^{12}$ → 12 bits

20 + 12 = 32 bits

assumption - each word = 1 byte

size of physical address => 64 MB = 2^{26} 26 bits
or bits for frame no + bits of offset
14 + 12 = 26 bits.

no. of entries in page table = no. of pages = 2^{20}
size of page table = no. of pages x size of each entry
$(2^{20} \times 14)$ bits
↳ page table stores frame no.

→ LA = 7 bits PA = 6 bits page size = 8 words / byte
no. of pages? no. of frames? size of page table?
no. of bits req. for offset ⇒ $8 = 2^3$ 3 bits
no. of bits req. for page no. = size of LA - size of offset
= 7 - 3 = 4

LA: | 4B | 3B |

no. of bits req. for frame no. = size of PA - size of offset
6 - 3 = 3

PA: | 3B | 3B |

no. of pages = 2^4 = 16 pages
no. of frames = 2^3 = 8 frames
size of page table = no. of entries x size of an entry
= no. of pages x size of frame no.
= 16 x 3 = 48

size of LAS = 2^7 = 128 Bytes / words
size of PAS = 2^6 = 64 Bytes / words

Page Table

- It is used to keep all the information about the page
- It's base address is stored in PTBR (Page Table Base Register)

Page table entries:

Frame No.	Valid/ Invalid	Protection (RWX)	Reference (0/1)	Caching	Dirty
	1-bit				1
1 (optional)					

Modify

- Frame no: frame no. corresponding to every page. no. of bits depend on no. of frames.
- Valid / Invalid bit: 0 - if present, 1 - if absent
  If a page is absent → page fault
  (It is used to support virtual memory)
- Protection bit: R - Read, W - Write, X - execute
- Referenced bit: 0 - if page is not accessed in last clk cycle
  1 - if accessed.
  (used in page replacement algo LRU)
- Caching: disabled - if page changes frequently & fresh page is req. (cache store old info)
  enabled - if caching is required
- Dirty bit: 1 - if modified (i.e. write-access)
  0 - if not modified
  (so that changes can be saved when page moves out)

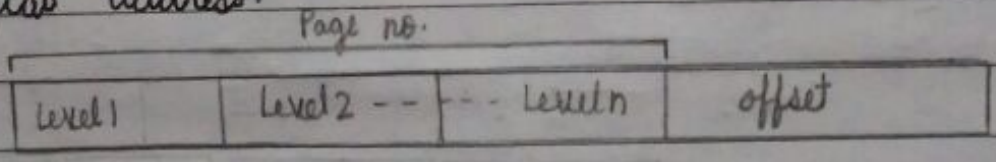
Dt
Pg

B+

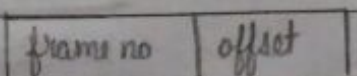
MULTILEVEL PAGING

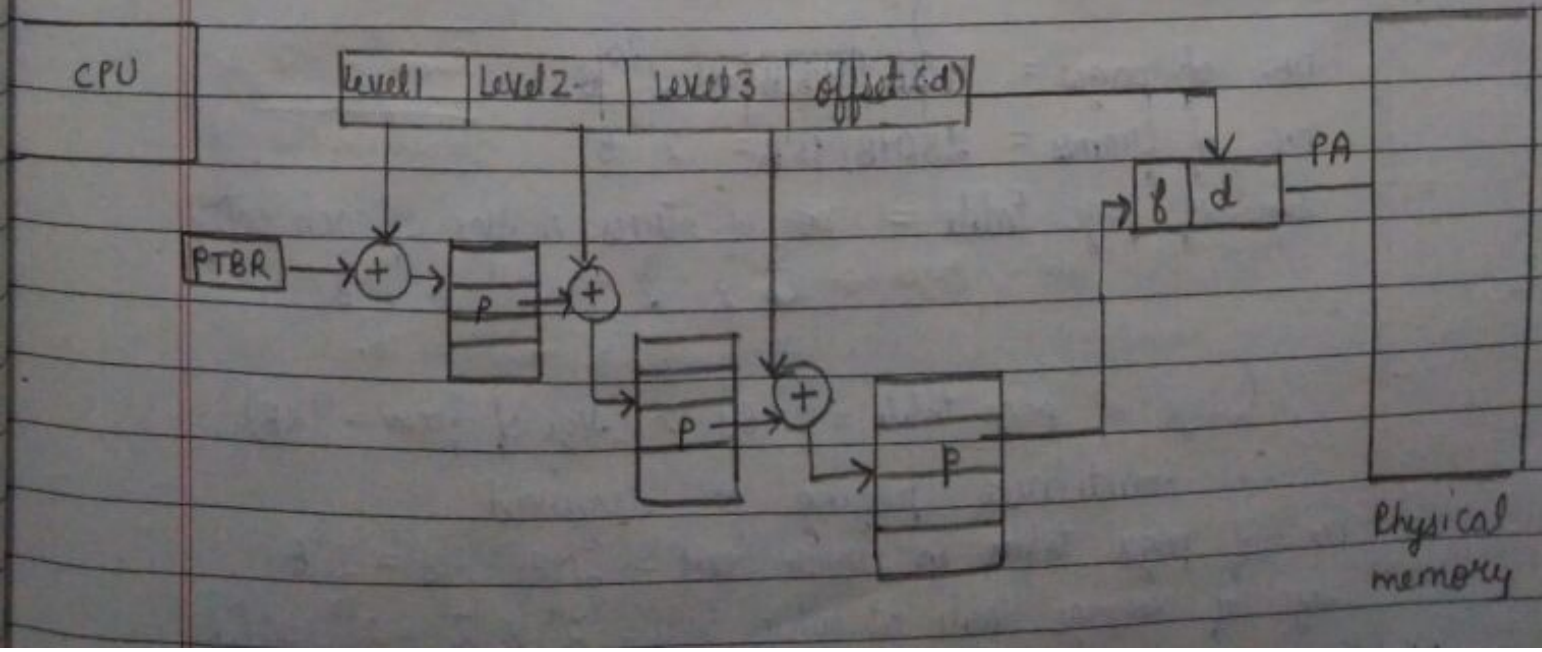
- It is a paging technique in which multiple levels of page tables are used (bcz size of page table > size of a frame ∴ page table too needs to be divided)
- The last level page tables contains actual frame info
- First level contains only 1 page table & its address is stored in PTBR
- One access to each level is required to find the actual frame.
- Each page table (except the last level) contains frame no. of next level page tables

Logical address:



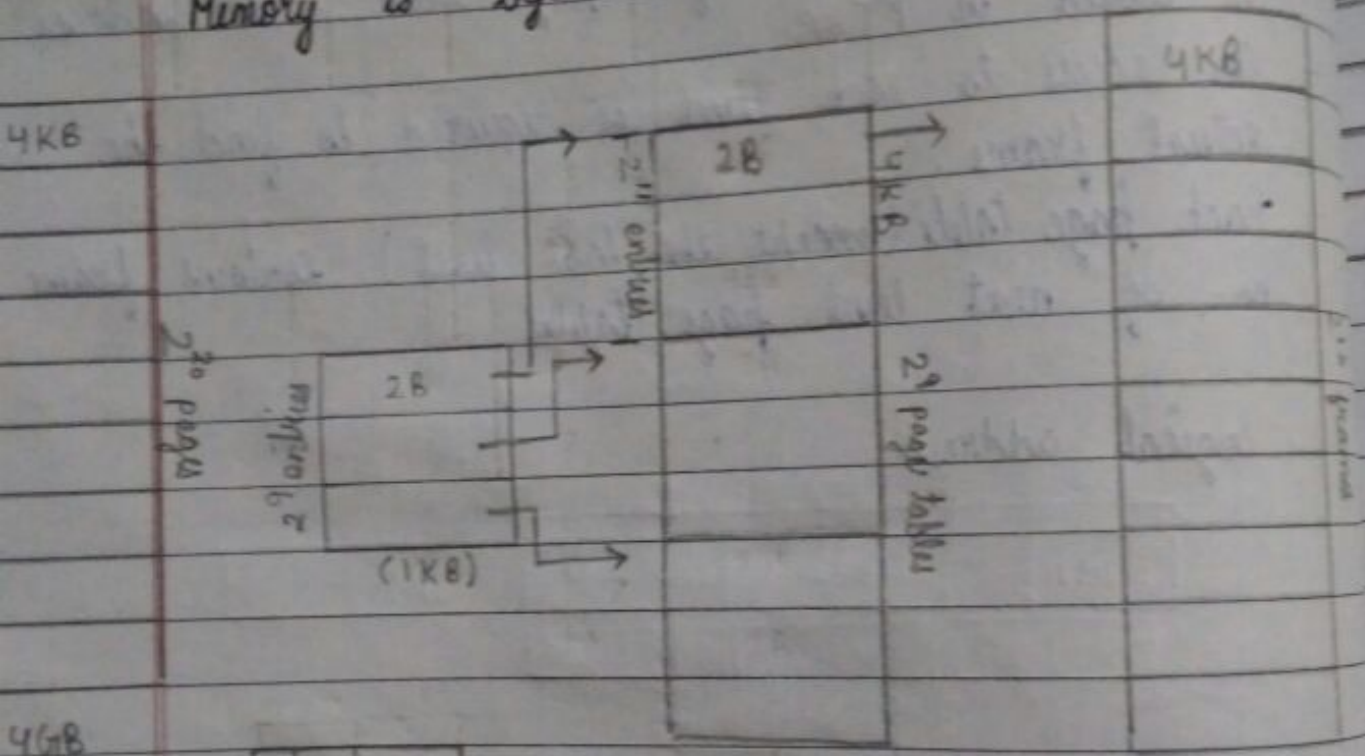
Physical address:





Example:

Physical address space = 256 MB 2^{28} B
Logical address space = 4GB 2^{32} B
Frame size = 4KB 2^{12} B
Page table entry = 2B
Memory is byte addressable



LA = | 9 | 11 | 12 |
Level1 Level2 offset

no of pages = 4GB/4KB = 2^{20} B
no of frames = 256MB/4KB = 2^{16} B

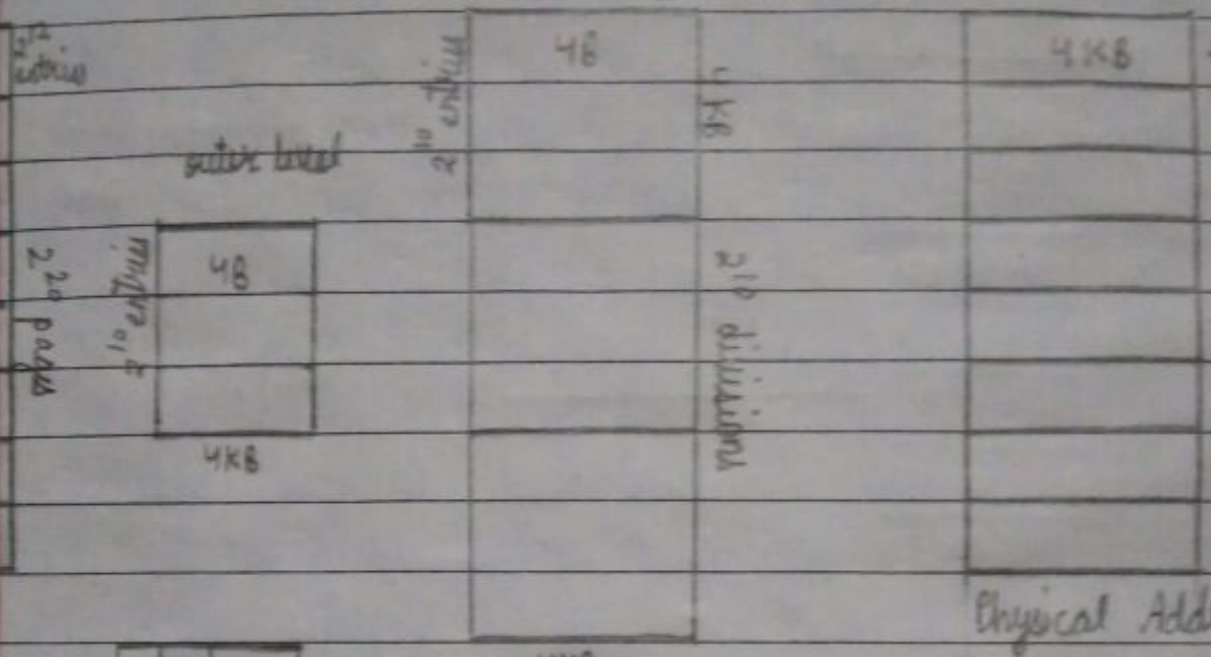
20 bits	12 bits	LA
16 bits	12 bits	PA

→ size of page table = no of entries × size of each entry
no of pages ← 2^{20} × 2 = 2^{21} = 2MB
no of bits for frame no.

(size of page table = 2MB) > (size of frame = 4KB)
∴ multilevel paging is required

no of page tables in inner level = 2MB/4KB = 2^9 B
→ size of page table at outer level = 2^9 × 2 = 2^{10} = 1KB
1KB < 4KB ∴ no further level

Example LA = 32 bit
Page size = 4KB 2^{12}B, Page table entry = 4B 2^2B
(2^{32}B)



No of pages = (Size of logical memory) / (size of a page)
= $2^{32} / 2^{12} = 2^{20}$ pages

20	12

Inner level
no of entries in page table = 2^{20}
size of each entry = 4B
∴ size of inner page table = $2^{20} \times 2^2 = 2^{22}$B = 4M

4MB > 4KB i.e. (size of page table > frame size)
∴ inner page table needs to be divided
No. of divisions of page table = 4MB/4KB = 2^{10}

Outer level
no. of entries = 2^{10}, size of each entry = 4B
∴ size of outer page table = $2^{10} \times 4B = 2^{12}B = 4KB$
4KB = 4KB i.e. (size of outer page table) = (size of frame)

Rarely used due to ↑ search time

Dt. Pg. B+

INVERTED PAGING

In this paging technique, an inverted page table / frame table is maintained for the frames present in physical memory rather than maintaining separate page tables for each process.

*

Inverted page table / frame table

- no of entries = no of frames
- Contains page no & page id (both combined refers to a unique page)

frame	Page No	Page Id
0	–	–
1	–	–
2	–	–
3	–	–

0	f_0
1	X
2	f_2
3	X

Page table of P1

0	X
1	f_3
2	f_5
3	X

Page table of P3

0	X
1	f_1
2	X
3	f_4

Page table of P2

fNo	Page No	Page Id
0	P_0	P_1
1	P_1	P_2
2	P_2	P_1
3	P_1	P_3
4	P_3	P_2
5	P_2	P_3

frame table / inverted page table

→ Each process has its own P.T
→ If pages of n processes are not in memory, then n P.T are not in the memory (memory overhead)

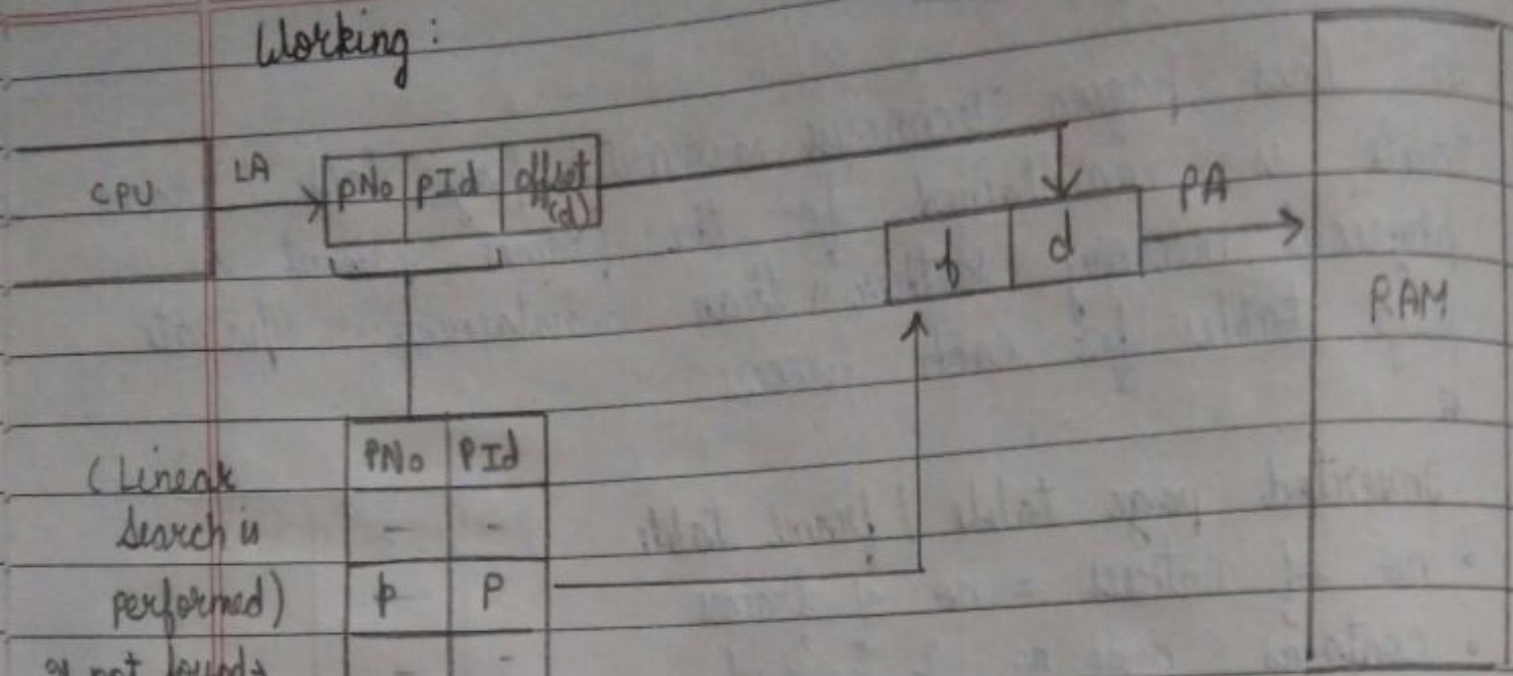
Paging

→ Only one inverted PT is not in the memory.
→ Linear search is performed to find the frame no. (search time ↑)

Inverted paging

Tutorial

Working:



(Linear search is performed)

If not found → segmentation fault

Advantages	Disadvantages
reduced memory space	search time ↑

Advantages of Paging

- No external fragmentation
- Swapping of pages is easy as pages are of same size.
- Efficient use of memory

Disadvantages of Paging

- Access time is high (LA → PA)
- Internal fragmentation in last page of the process.
- Large memory space is required as page tables are also stored

Dt. B+
Pg.

VIRTUAL MEMORY

It is a memory allocation scheme in which secondary memory can be used as a part of main memory at the times of shortage of RAM

- All memory references must be logical addresses and must be dynamically translated into physical addresses at run time. [∴ parts of the process can change locations during execution ∃]
- Process must be divided into pieces.

In this scheme,

→ Only required parts of the process is loaded into the main memory & rest of the parts are stored in a secondary memory & fetched when required.
→ This section of secondary memory is termed as virtual memory.
→ It gives an illusion to the user that complete processes are loaded into the main memory and therefore it is large enough but in actual, real memory is limited.

Features:

- Processes which are larger than RAM can be executed
- Reduces external fragmentation (As entire process need not be loaded at the same time)
- The amount of space used by the process (in real memory) in main memory can be varied during execution
- Location of various processes may be varied during execution
- Degree of multiprogramming increases.
- It is implemented by 'Demand Paging'
- It can also be implemented by 'segmentation' & 'paged segmentation' technique but it is quite complex.

Dt
Pg B+

Demand Paging [paging concept + virtual memory concept]

- The process of loading a page into the main memory only when it is demanded. Lazy Swapper
- A page may or may not be present in the memory when it is demanded.
  - If the page is present in main memory, it is accessed (page bit)
  - If page is not present in main memory, page interrupt occurs & control is shifted from user to OS - Page Fault

PMT

Page Map Table (PMT)

- It contains frame no, valid / invalid bit.
- If page is present in main memory, corresponding frame no. is entered & valid bit = 1
- If page is not present, frame no is empty & valid bit = 0

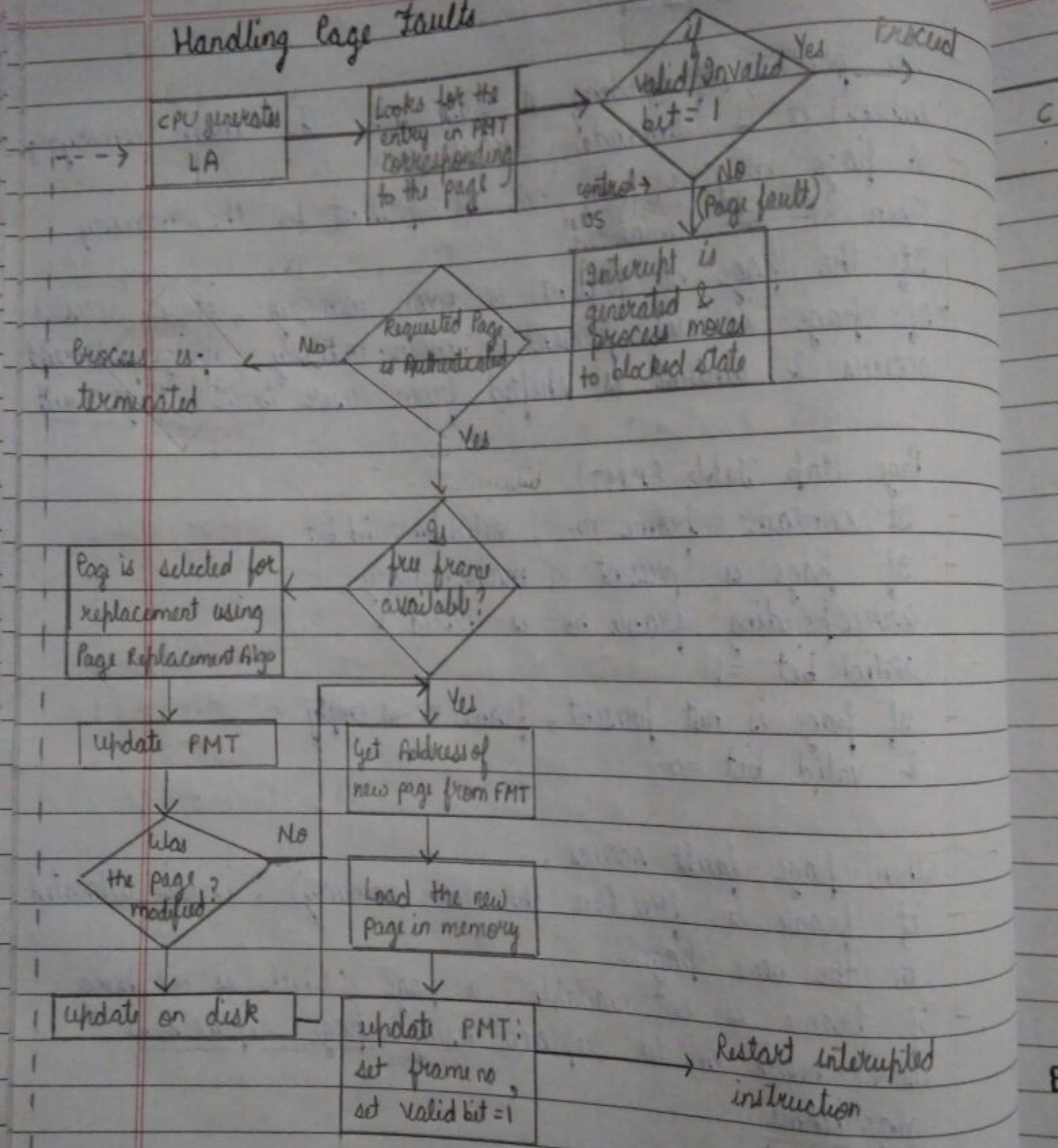
	f no.	v/i
0	X	0
1	3	1
2	X	0
3	8	1

PMT

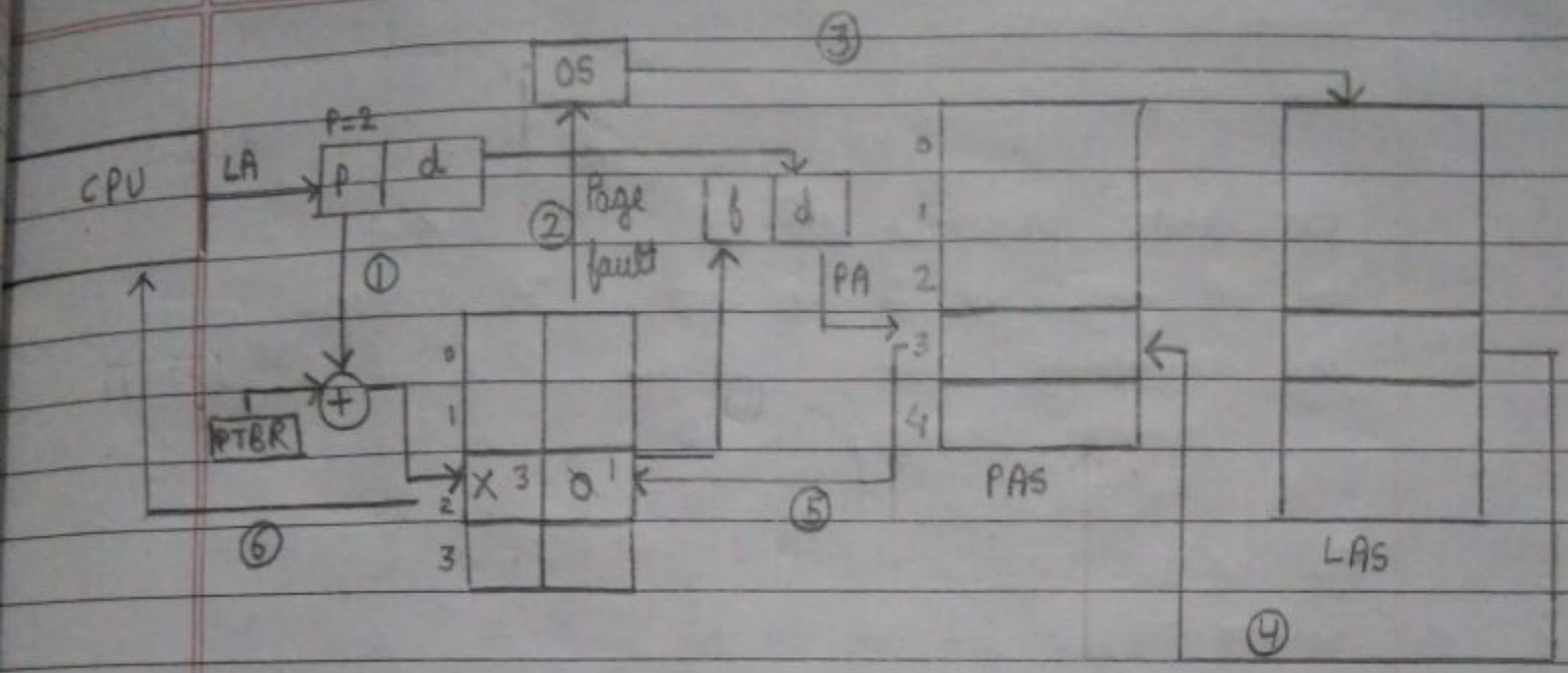
When page fault occurs,

- if frame is free (in the main memory), it is allocated to the new page.
- if frame is not available, a page (which is not being used) needs to be replaced using page replacement algorithms.

Handling Page Faults



B+



Effective Memory Access Time (EMAT)

page fault rate = p [$0 \le p \le 1$]
(probability of page fault occurance)
page fault service time = S (usually in millisec)
(time taken to handle page fault)
memory access time = m (usually in nanosec)
(time taken to access the memory)

$$EMAT = p*(S+m) + (1-p)*m$$
$$= p*S + (1-p)*m \quad [\because p\,m << S]$$

Example: MM Access time = 100 u sec, Page fault rate = 40%, S = 4m sec

$$EMAT = 0.4\,(4 \times 10^{-3}) + (0.6)(100 \times 10^{-6})$$
$$= 1.6 \times 10^{-3} + 0.6 \times 10^{-4}$$
$$= 1.6 \times 10^{-3} + 0.06 \times 10^{-3} = 1.66 \times 10^{-3} \text{ sec}$$
$$= 1660 \text{ u sec}$$

Pg B+

Page Replacement Algorithms

Algorithms that are used to decide which page needs to be replaced by the new page that is demanded, in case page fault occurs

1) FIFO (First In First Out)

The page that comes first in the main memory, is replaced first

OS maintains a queue for all the pages, front → oldest page

Example:

Reference String	3	2	1	3	4	1	6	2	4	3	4	2	1	4	5	2	1	3	4
f_1	3	3	3	3	4	4	4	4	4	3	3	3	3	3	5	5	5	5	4
f_2		2	2	2	2	2	6	6	6	6	4	4	4	4	4	2	2	2	2
f_3			1	1	1	1	1	2	2	2	2	2	1	1	1	1	1	3	3
	M	M	M	H	M	H	M	M	H	M	M	H	M	H	M	M	H	M	M

no of hits = 6 hit ratio = (6/19) [x 100]

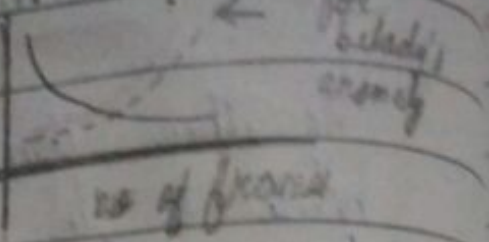
no of miss = 13 miss ratio = (13/19) [x 100]

Reference String	7	0	1	2	0	3	0	4	2	3	0	3	1	2	0
f_1			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_3	7	7	7	2	2	2	2	4	4	4	0	0	0	0	0
	x	x	x	x	✓	x	x	x	x	x	x	✓	x	x	✓

no of hits = 3 hit ratio = 3/15 = 1/5 = 20%

miss = 12 miss ratio = 12/15 = 4/5 = 80%

Belady's Anomaly : (in FIFO)



For some reference strings no of page faults increase with increase in ~~by increasing~~ no of frames. This is known as Belady's Anomaly.

Example :

Reference String	1	2	3	4	1	2	5	1	2	3	4	5
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	5
	x	x	x	x	x	x	x	✓	✓	x	x	✓

Hits = 3
Miss = 9

Reference String	1	2	3	4	1	2	5	1	2	3	4	5
f_4				4	4	4	4	4	4	3	3	3
f_3			3	3	3	3	3	3	2	2	2	2
f_2		2	2	2	2	2	2	1	1	1	1	5
f_1	1	1	1	1	1	1	5	5	5	5	4	4
	x	x	x	x	✓	✓	x	x	x	x	x	x

Hits = 2
Miss = 10

Last replaced page is the one demanded next

bcz future requests are not known
practically not possible, acts a theoretical benchmark for other algos

DL for Pg B+

2) Optimal Page Replacement
(Best) for all ref. str

The page which is not used for longest period of time in the future is replaced.

follow FIFO

Example:

Reference String	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
f_4				2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
f_3			1	1	1	1	1	4	4	4	4	4	4	1	1	1	1	1	1	1
f_2		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
f_1	7	7	7	7	7	3	3	3	3	3	3	3	3	3	3	3	3	7	7	7
	x	x	x	x	✓	x	✓	x	✓	✓	✓	✓	✓	x	✓	✓	✓	x	✓	✓

page hits = 12 page hit ratio = 12/20 = 3/5
page miss = 8 page miss ratio = 8/20 = 2/5

3) Least Recently Used (LRU)

Hard → The page that is least recently used is replaced
page that is not used for long time

Example:

Reference String	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0
f4				2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
f3			1	1	1	1	1	4	4	4	4	4	4	1	1	1	1	1	1
f2		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
f1	7	0	7	7	7	3	3	3	3	3	3	3	3	3	3	3	3	7	7
	X	X	X	X	✓	X	✓	X	✓	✓	✓	✓	✓	X	✓	✓	✓	X	✓

page hit = 12
page miss = 8

hit ratio = 12/20 = 3/5
miss ratio = 8/20 = 2/5

4) Most Recently Used (MRU)

The page that is most recently used is replaced

Reference String	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
f4				2	2	2	2	2	2	3	0	3	2	2	2	0	0	0	0	0
f3			1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
f2		0	0	0	0	3	0	4	4	4	4	4	4	4	4	4	4	4	4	4
f1	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7
	X	X	X	X	✓	X	X	X	✓	X	X	X	X	✓	✓	X	✓	✓	✓	✓

page hit = 8
page miss = 12

hit ratio = 8/20 = 2/5
miss ratio = 12/20 = 3/5

SEGMENTATION

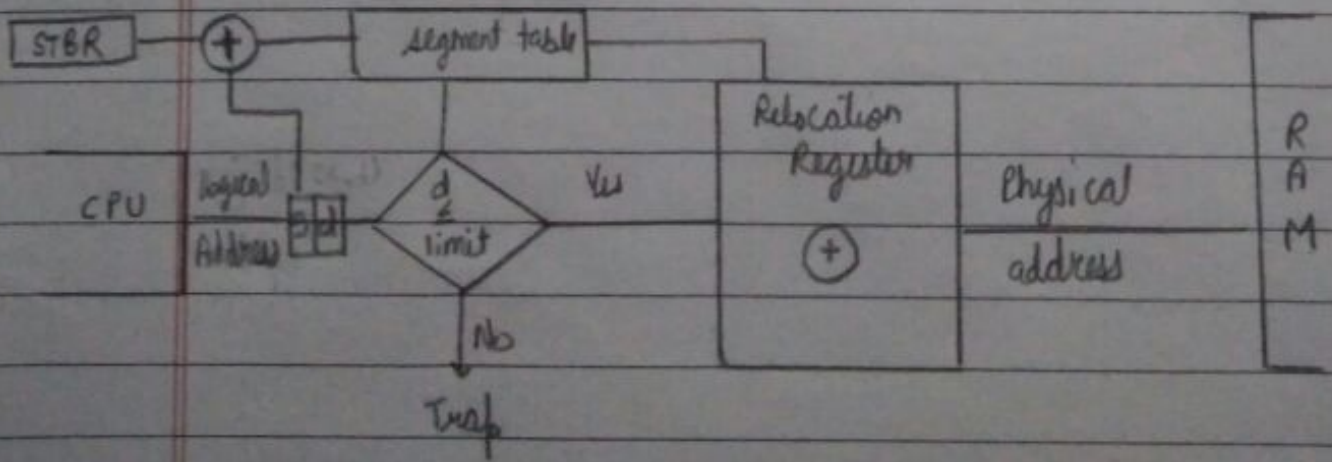
- non - contiguous memory allocation technique
- process is divided into segments which are not necessarily all of the same sizes.
- Segments are made as per the user's view. Each segment contains same of type of function eg main function may constitute a segment while library function can be included in another segment.
- A segment table is maintained for each process

Base address	Limit/ offset
-	-
-	-
-	-

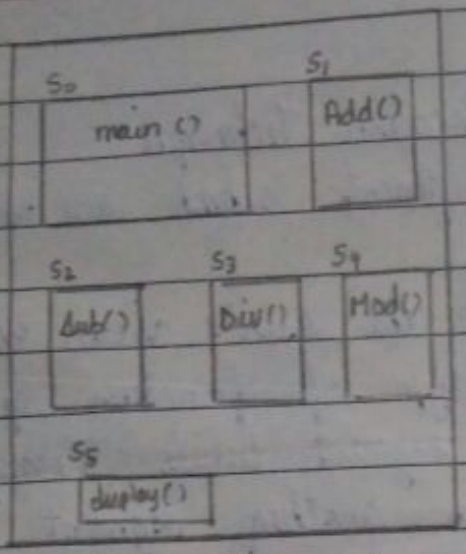
- Base address: contains the starting physical address where the segments reside in the physical memory.
- Limit: specifies length of segment

Logical address: | Segment no (s) | Limit (d) |

Physical address: | Base address (b) | Limit (d) |



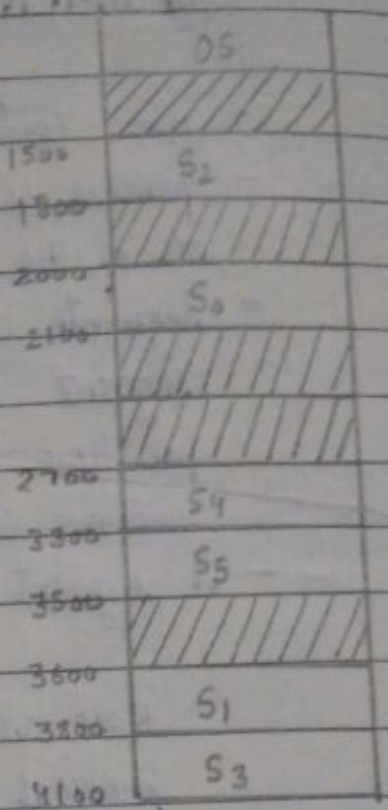
Physical Address = base address + limit / offset



Logical address space

Segment	BA	Limit
0	2000	100
1	3600	200
2	1500	300
3	3800	300
4	2700	600
5	3300	200

Segment Table



Physical Address Space

→ (1, 100)

$100 \leq 200$

(1, 100) → (3600, 100) ✓

physical address = 3600 + 100 = 3700

→ (2, 400)

$400 \not\leq 300$ Trap

Dt
Pg
B+

Adv. :
- No internal fragmentation
- Segment Table consumes less space in comparison to page table in paging
- average segment size is larger than page size
- easier to relocate segments
- segments are of variable size as per the user's view

Disadv :
- external fragmentation
- costly memory management algos.
- maintenance of segment table for each process leads to overhead

NAME : PAYAL
CLASS : CSE-2 (IV Sem.)
ROLL No. : 23609

Dt
Pg

B+

PAGING WITH SEGMENTATION

- This is a non-contiguous memory allocation technique.
- In this technique process is divided into variable-size segments. Further each segment is divided into fixed-size pages. Therefore pages are always smaller than the segments.
- Every process has a segment table and multiple page tables corresponding to different segments.
- Main memory is divided into frames.
  size of page = size of frame.

Segment Table

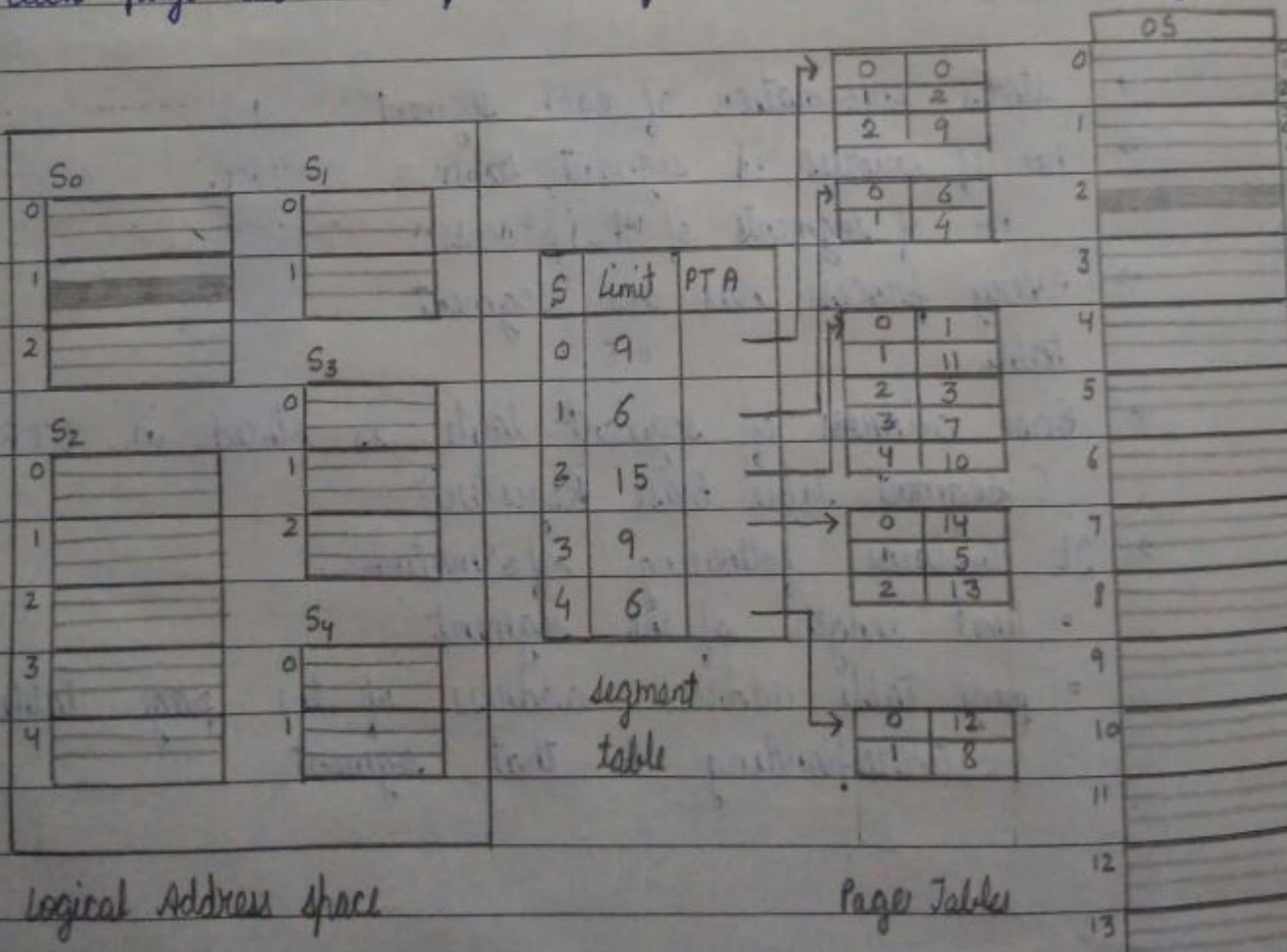
Limit	Page Table Address
-	-
-	-
-	-

→ stores information of each segment.
→ no of entries in segment table = no of segments of the process.
→ every process has one segment table
→ Base address of segment table is stored in STBR (Segment Table Base Register)
→ It contains following information :
- limit : length of the segment
- page table address : address of the page table corresponding to that segment.

Page Table

Page No	Frame No

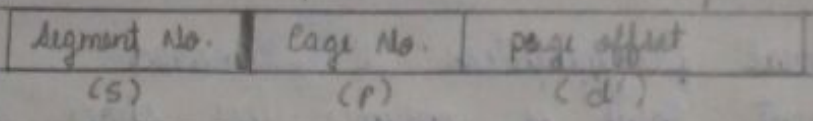
→ Stores information of each page of a segment.
→ Each segment has its own page table. Therefore,
→ no of page tables in a process = no of segments
→ No of entries in a page table = no of pages in the corresponding segment
→ It contains the frame no corresponding to each page.
→ Each page table occupies one frame in the main memory



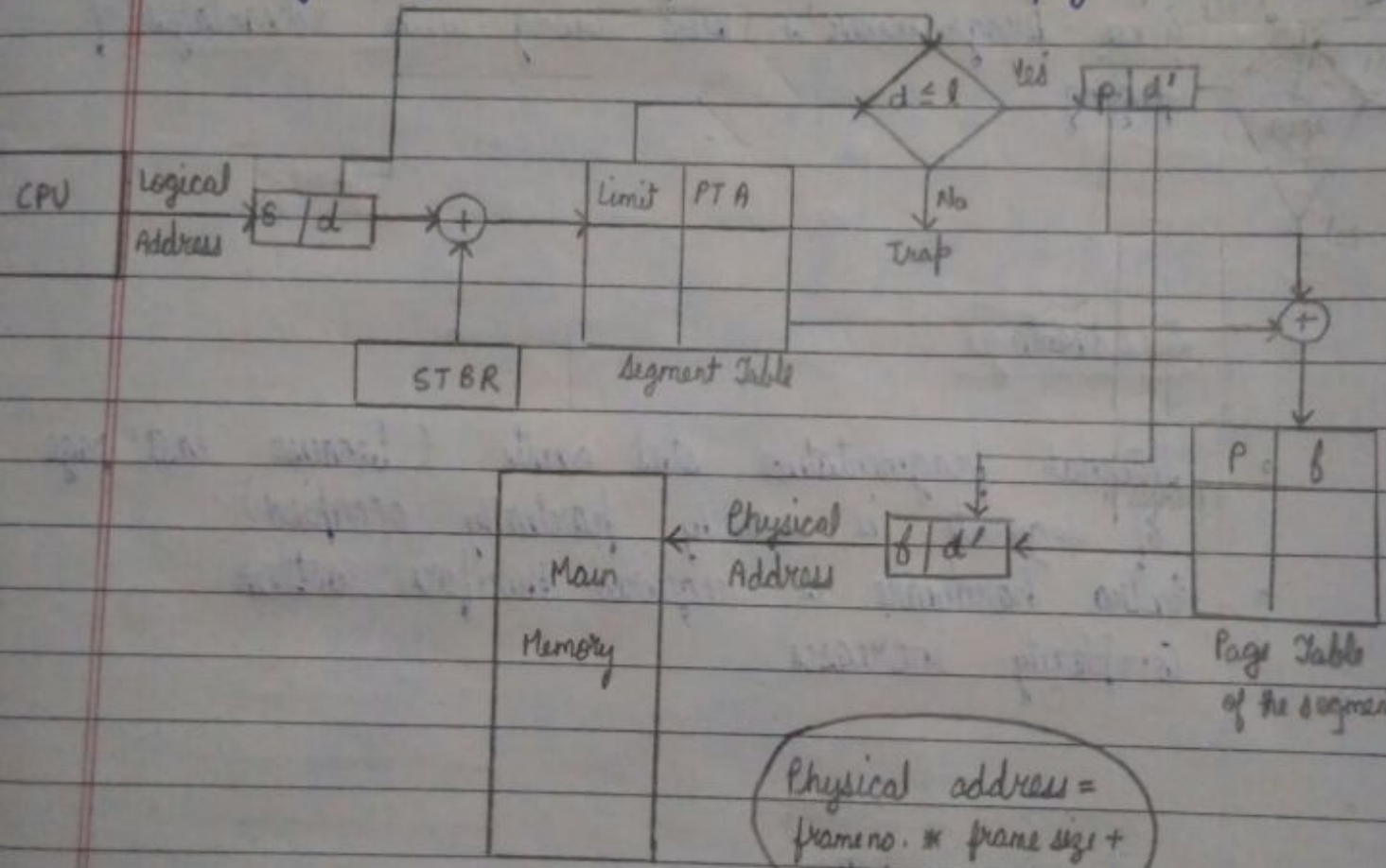
Dt
Pg

B+

Logical address virtual address generated by the CPU



MMU (Memory Management Unit) is responsible for mapping logical addresses into the physical addresses.



Physical address = frame no. * frame size + offset

Example:

S=0 P=1 d'=1

0	0	0	0	1	0	1

d = 5

$5 \leq 9$ True

f = 2 d' = 1

0	0	1	0	0	1

$2 * 3 + 1 = 7$

Physical Address

Dt
Pg B+

Advantages

- External fragmentation is removed
- Segment table is consumes less space as no of segments are limited
- Page size is limited by segment size
- Gives programmer's view along with advantages of paging

Disadvantages

- Internal fragmentation still exists (because last page of segments is usually partially occupied)
- Extra hardware is required therefore costlier.
- Complexity increases.