Operating System

- Silber shetz Galvin
- OS a concept based approach by DM Dhamdhere
- OS by willam stalling.

- OS is a prog. use to execute other prog. efficienty & convinently.
- OS act as a interface b/w user & hardware
- Turnaround time → Is the time elapsed since submission to job is done.

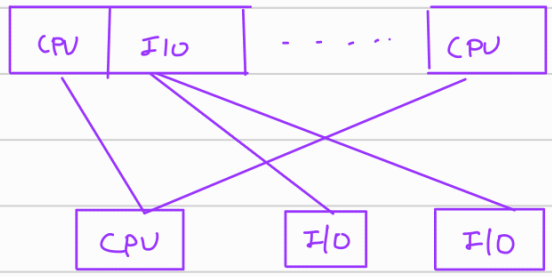
Types of OS →

- batchprocessing OS → In this user submit job to comp. operator. now comp. operator group this jobs to form a batch.
  - Their is 1 special prog. batch moniter prog. in comp. (RAM)
  - operator submit batch with control info in comp. to initiate execution of 1st job.
  - initiate mean assign address to prog. counter
  - Sequencing is done by batch prog
  - Overall turnaround time reduces.

- In batch processing user submit their job to computer operator. comp. operator will then grp these job to group in batch
- this batch of job is submit to comp with some other con-trol info.
- Their is a resident prog called as batch moniter prog that always stays in the computer system.
- This batch moniter prog. initiates execution of 1st job. when this job terminates, control branches back to batch moniter prog.
- batch moniter prog will then initiate job
- The main obj. of batch moniter system is automating seq.
- Adv → Reduce turnaround time.

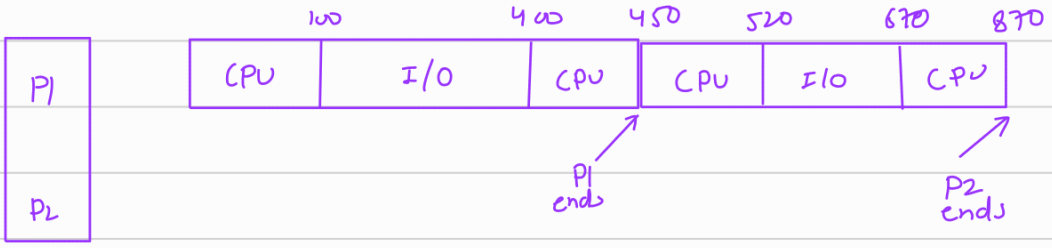
- Theoritically a prog. start & end with CPU burst.

Uni program → (Automated sequences)



- At any instance of time only 1 program progress
- After completion of 1st another can execute
- In uni programing their may be multiple prog in main memory. OS select some program to execute over CPU only after termination of this prog can another prog. executes
- For ex. consider 2 prog P1 & P2 having following execution trace

Prog.	CPU	I/O	CPU	(in ms)
P1	100	300	50	
P2	70	150	200	



Turnaround time P1 = 450

——11—— P2 = 870

Avg turnaround time = $\frac{P1 + P2}{2} = \frac{450+870}{2} = 660$

- CPU utilization is the fraction of time CPU is utilized i.e total CPU time/total turnaround time.

$$\text{CPU utilization} = \frac{420}{870} = 0.482$$

- I/O utilization = fraction of time I/O utilized

I/O ut. = 51.8%

- Throughput is no of prog completed per unit time

$$\text{throughput} = \frac{2}{870} \text{ prog/millisecond}$$

$$= \frac{2000}{870} \text{ prog/sec} = 2.2 \text{ prog/sec}$$

$$= \text{i.e } 2 \text{ prog/sec}$$

- progress coefficient - It is a fraction of time a prog progresses

$$\text{Prog. coff. (P1)} = 1 \quad \text{(After 450 P1 ends)}$$

$$\text{Prog. coff (P2)} = \frac{420}{870}$$

Multiprogramming →

- In multiprog. their may be multiple prog in main memory but generally multiple prog reside in main memory.
- I/O - I/O may overlap CPU - I/O overlap
- Unless stated we have to made maximum overlap.

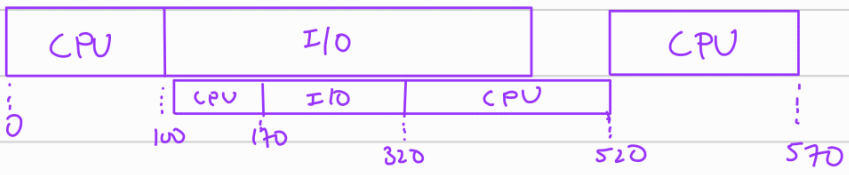
- In multiprogramming their may be multiple prog in the main memory operating system select some prog. to execute over CPU (depending on cpu scheduling algorithm).
- As long as this prog performs CPU brust, CPU will be allocated to the prog. but when it wishes to perform I/O activity, CPU will be allocated to some other prog which requires it if any
- In short overlap of CPU act or I/O act. & overlap of I/O & I/O act. or both.

Ex

Prog.	CPU	I/O	CPU
P1	100	300	50
P2	70	150	200

Assume first come first serve →

P1 :



- By this total turnaround time ↓, CPU utilization increase

$$\text{CPU utilization (in \%)} = \frac{420}{570} \times 100 = 73.6\%$$

$$\text{I/O utilization (in \%)} = \frac{300}{570} \times 100 = 52.6\%$$

$$\text{CPU ideal time (in \%)} = \frac{150}{570} \times 100 = 26.4\%$$

$$\text{throughput} = \frac{2}{570} \times 1000 \ \frac{\text{prog}}{\text{sec}} = \left\lfloor \frac{200}{57} \right\rfloor \text{ prog/sec}$$

$$= 3 \text{ prog/sec}$$

$$\text{prog. coff (P1)} = \frac{450}{570}$$

$$\text{prog. coff (P2)} = \frac{420}{520}$$

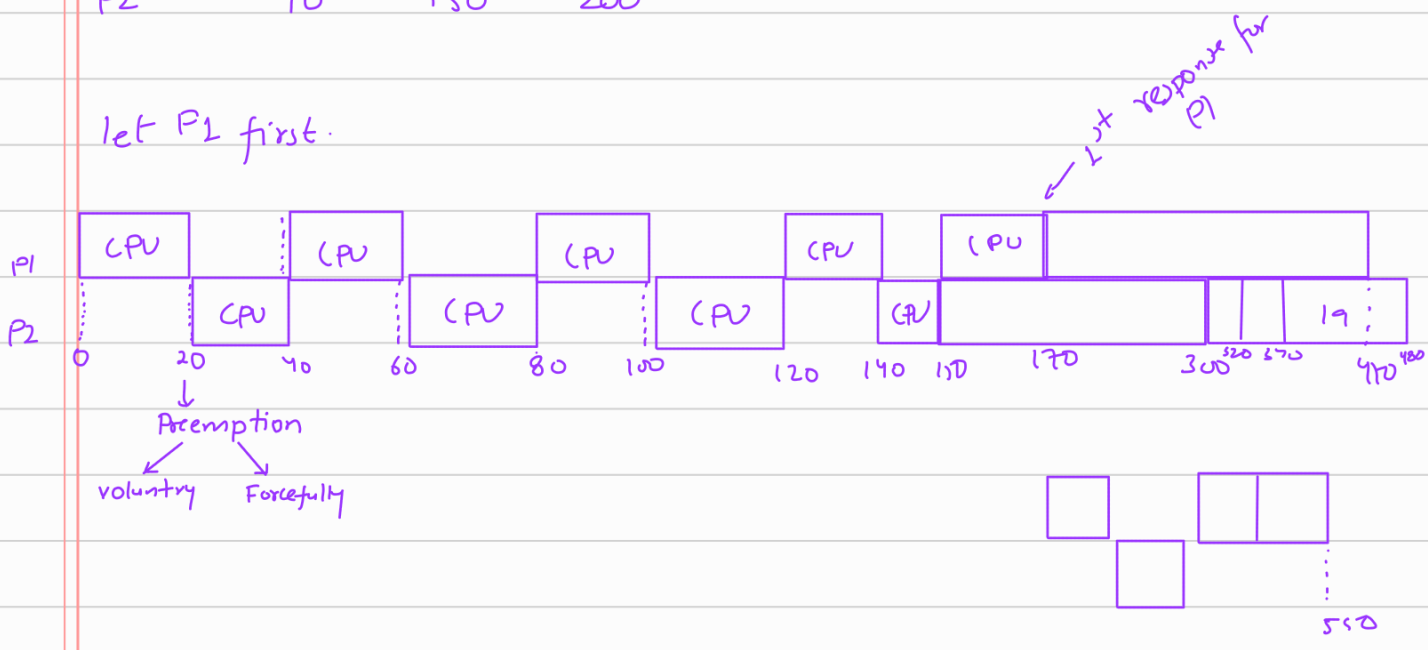
- <u>Time Sharing →</u>
- Time sharing is extension of multiprogramming
- Their is predefined value of time known as time quantum or time slice. (time quantum is defined by OS designed)
- A process stops either if CPU brust end or time quantum finishes if one a process is selected over CPU to be process
- Then next process is selected for execution over CPU.

Ex-

Prog.	CPU	I/O	CPU (in ms)
P1	100	300	50
P2	70	150	200

TQ = 20ms

let P1 first.



CPU utilization = $\frac{\quad}{550}$ = 76.3%

- CPU utilization might ↑, ↓ or remain const compared to multiprog.
- Main aim of multiprogramming is to get fair response time.
- Response time is time elapsed since submission of prog till its first response appear excluding the time it take to generate the response
  response → I/O

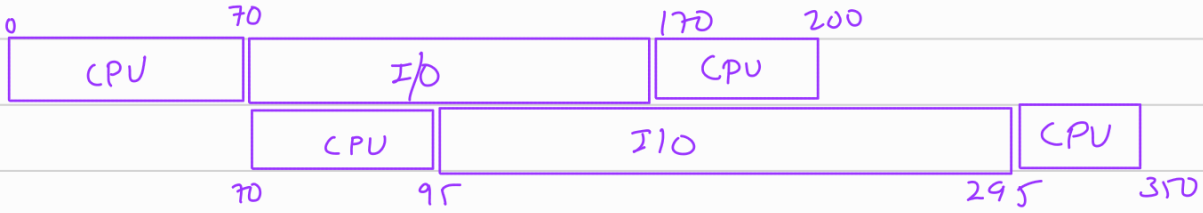
Q. Consider following set of processes assume to be arrived at time 0 & in order, following is the exhibition trace for these processes all the Calculate CPU utilization, I/O utilization and throughput for

(i) Uniprogramming
(ii) Multi programming
(iii) Time sharing TQ = 10 m-sec

Progress	CPU	I/O	CPU
P1	70	100	30
P2	25	200	55

(i) CPU ut. = $\frac{180}{480}$ I/O ut. = $\frac{300}{480}$ throughput = $\frac{2}{480}$ ×1000

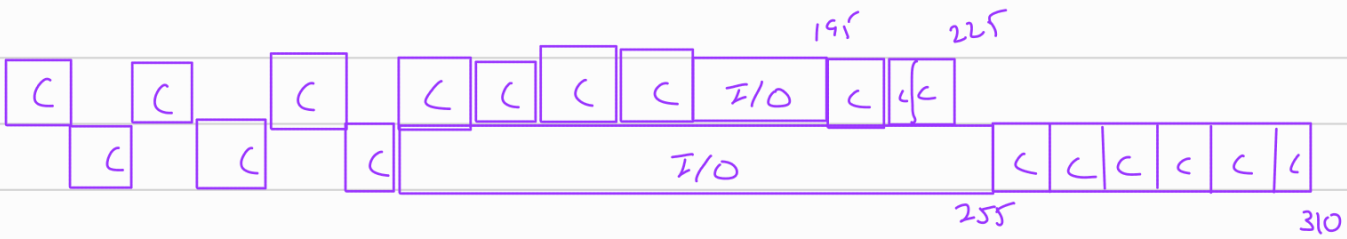
(ii)



$$\text{CPU Ut.} = \frac{70+25+30+55}{350} = \frac{180}{350}$$

$$\text{I/O ut} = \frac{225}{350} \qquad \text{throughput} = \frac{2}{350} \times 1000$$

(iii)



$$\text{CPU Ut.} = \frac{180}{310}$$

$$\text{I/O Ut} = \frac{200}{310}$$

$$\text{Throughput} = \frac{2}{310} \times 1000$$

Q. Consider 3 clocks Job1, Job2, Job3 assumed to bearrived at t=0 initially in order following are attribute of these jobs

Attribute	Job1	Job2	Job3
Type of Job	Heavy compute	Heavy I/O	Heavy I/O
Duration	10 min	5 min	15 min
Need printer	No	Yes	No
Need disk	No	No	Yes
Memory	150KB	200KB	100KB

Calculate CPU utilization, IO utilization, Printer utilization, disk utilization, memory utilization.

- $\text{CPU} = \frac{10}{30}$ $\quad \text{I/O} = \frac{20}{30}$ $\quad \text{printer} = \frac{5}{30}$ $\quad \text{disk} = \frac{15}{30}$

$$\text{Memory} = \frac{150+200+100}{3\,(1000)} \rightarrow \text{Given}$$

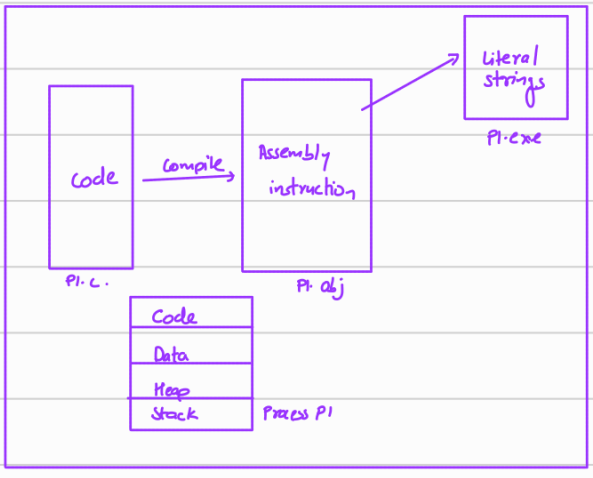
Multiprog

- P1: 450/1000
  P2: 300/1000
  P3: 100/1000

  Avg ut. $= \frac{850}{1000} \times \frac{1}{3}$

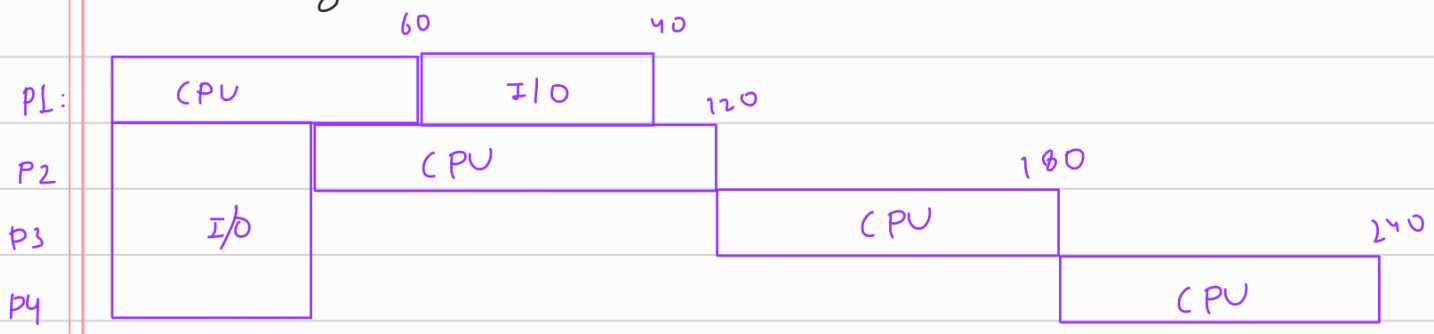
Process →

- Every info (file) is stored in secondary storage disk.
- When we compile a file a executable file is developed in secon-dary storage disk
- Executable file is of lower level lang. corresponding to high level lang. known as assembly level instruction
- Apart from this it contain some predefine header statement about exe itself which is later on needed to execute
- While we execute this prog. a process corresponding to the prog. is being formed & is initially stored in disk itself.
- The process contains 4 section namely
  ① Code ② Data ③ Heap ④ Stack



- Code contain all machine level instr.
- Size of code = $x * y * z$ (Assuming uniform)
  (x ↓ HLL, y ↓ LLL, z ↓ storage)
- Data section will be storing static & global variable.
- Both code section & data section is compile time constant.
- At the time of compilation storage is provided to code & data section
- Heap & stack are not compile time constant, actual space will be consumed at runtime.
- These two section grow at runtime toward each other
- Heap section is used to store dynamically allocated variable
- Stack contain info. about local variable of the program former parameter and some other information

Q. Consider a multiprog. operating system having 4 processes where each process spends 40% of time in I/O utilization, calculate effective CPU utilization?

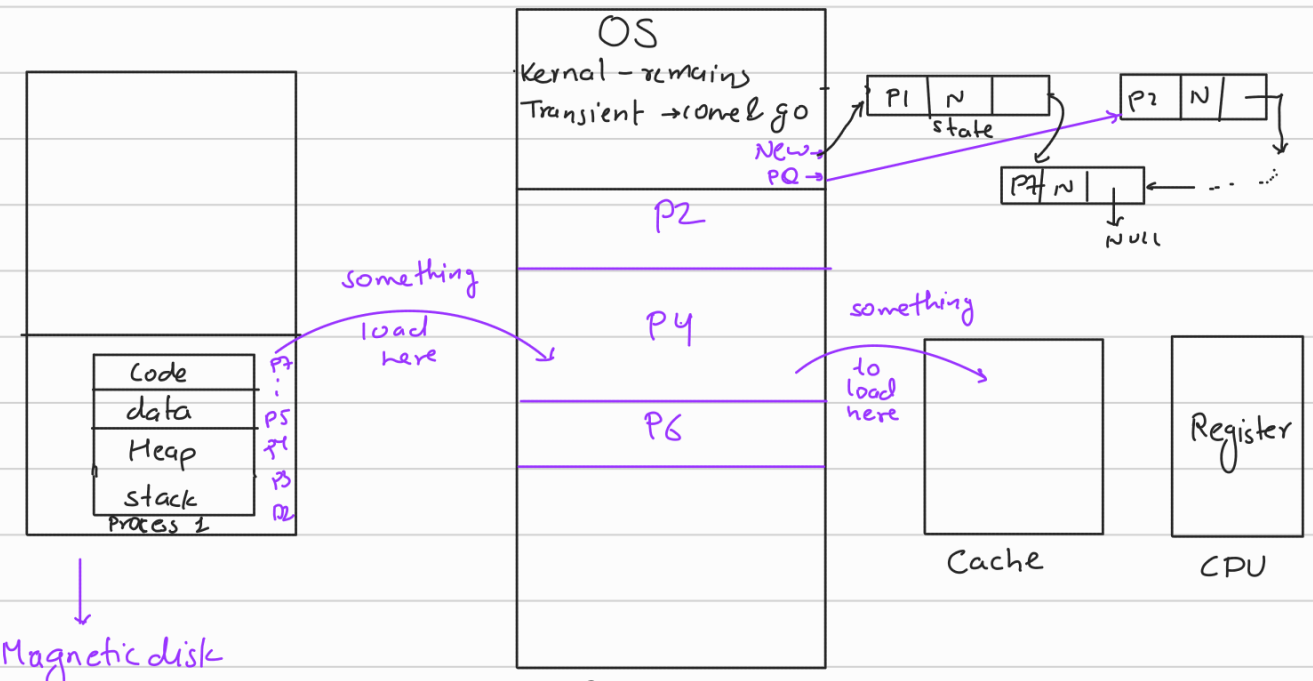


$$\text{CPU utilization} = \frac{240}{240} = 1$$

- Nothing given ∴ Max overlap.
- This is wrong

Prob of all I/O together = $(0.4)^4$

∴ prob. of CPU utilization = $1-(0.4)^4 = 0.976$ i.e 97.6%



Magnetic disk
- slow but large

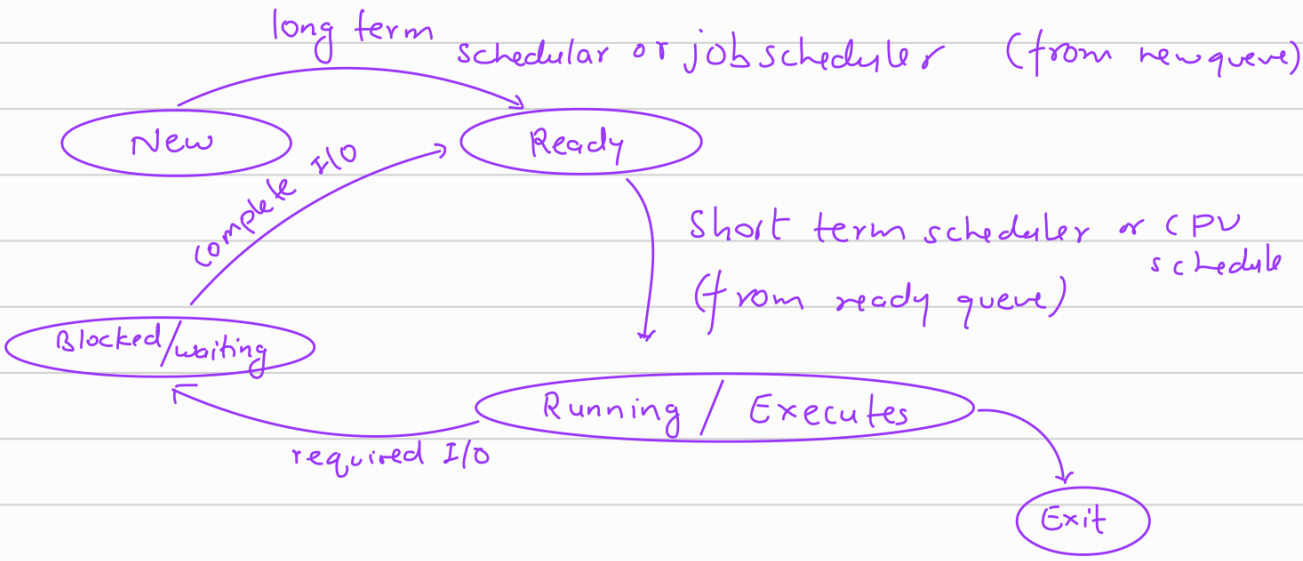
- Fast
- Relative small size

- larger the storeage it is slower

OS → Resident → Kernal
OS → Transient → Come & go

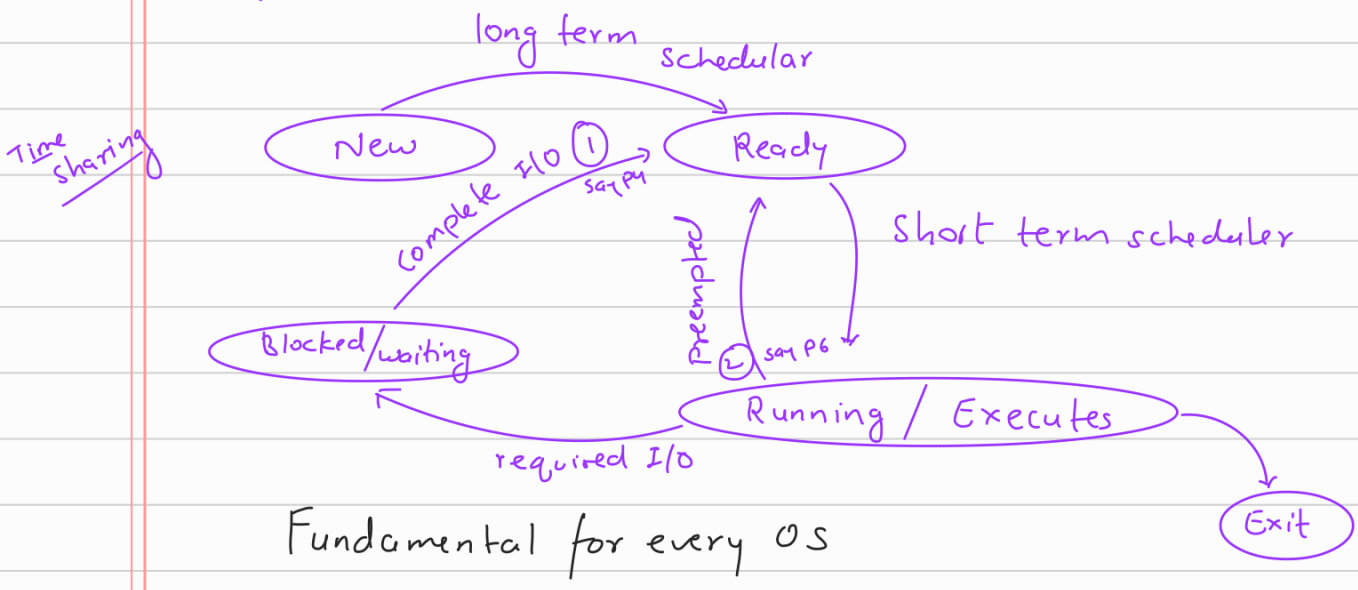
- If state of process is New (N) it mean process is created but yet to load in main memory
- Among these 'N' process which process is to select we use scheduling.

Process state transition Diagram



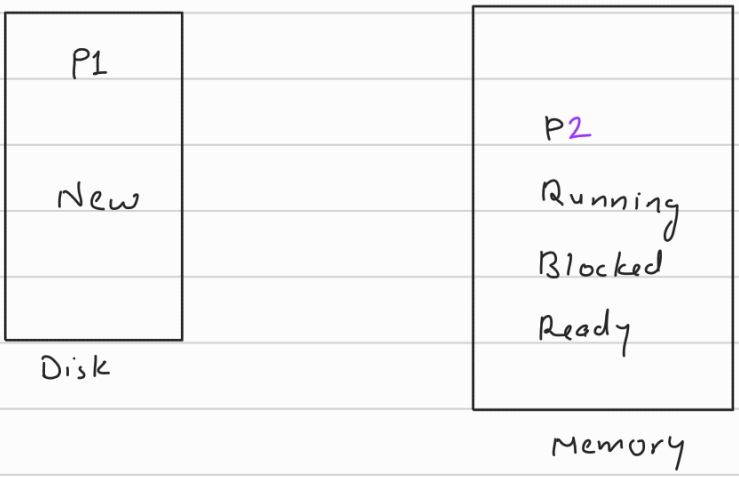
- The process that are loaded in memory their state will become ready & pointer changes
- The process in ready state are waiting for CPU
- depending of CPU scheduling 1 process from ready queue is selected by CPU to execute.
- The process which is scheduled by CPU should move out from ready queue
- Only 1 process will be in running state on CPU
- The process running is performing CPU burst after that it need to perform I/O burst, for this process comes under waiting state and it will be waiting in waiting queue
- Two cond^n when state = blocked → doing I/O, waiting for I/O
- At this time scheduler will invoke & select another process from ready-queue to run over CPU
- Now when prv process complete I/O it will join ready queue to use CPU again at last position
- When all I/O & CPU burst are over then it state change from running to exit.
- Exit state is important for garbage collection

- In time sharing if process is preempted process state again go to ready state



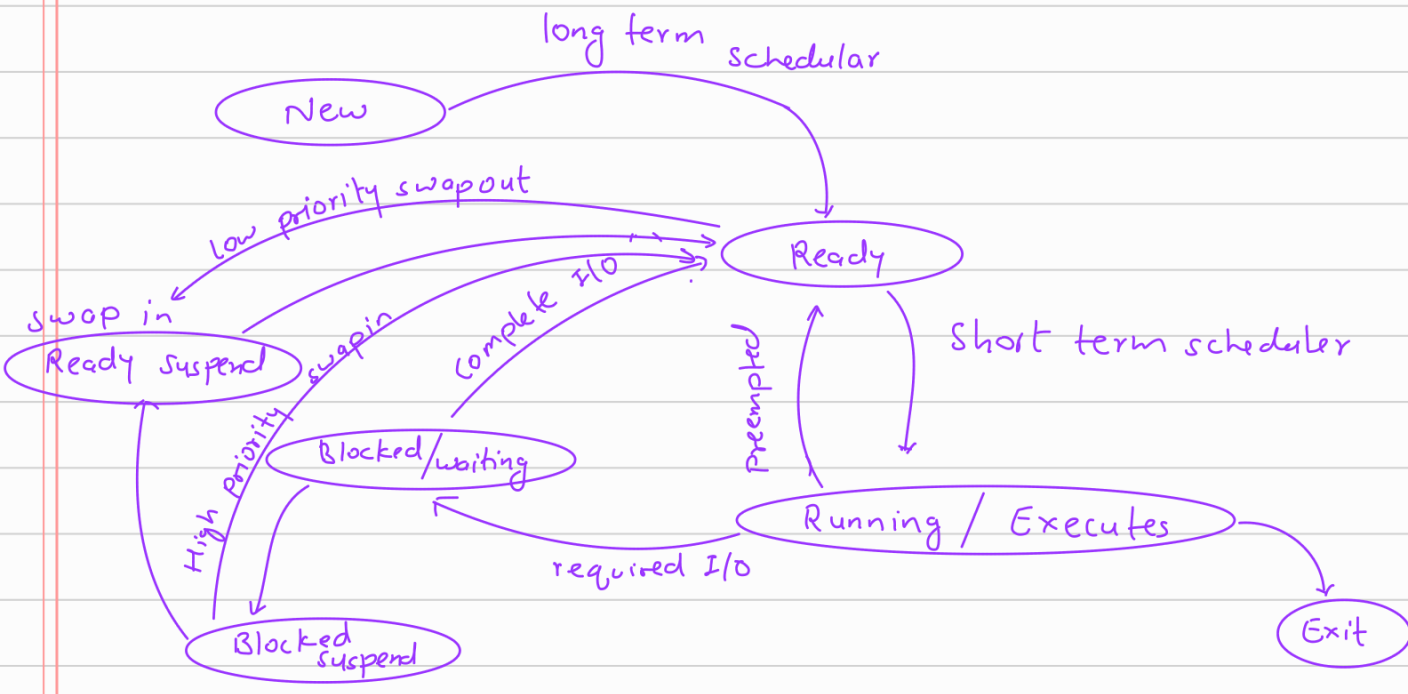
Fundamental for every OS

- If ① & ② occor symultaneously then who will join ready queue first ??
- We can go with P4, the last CPU process for it arrive earlier.
- New → process is created yet to brought in main memory



- OS say why to keep a process performing I/O in main memory (very fast) & I/O are slow ∴ OS swap process in blocked state back to disk
- This way more memory is avalible for CPU ∴ CPU utilization may increase.
- Degree of multiprogramming = No. of prog. at main memory
- After swapping state of prog trans. back to disk will change it state to blocked suspend
- After performing I/O state will change to ready suspend as it

directly can't placed in ready state



- When a high priority process complete I/O in Blocked suspend & need to run immediately then it directly move to ready state & it swap out a low priority process from ready state to ready suspend state.

Short term Scheduling / CPU Scheduling →

- There may be several process in ready queue (RQ), CPU scheduling governs the way, the process is being selected from RQ & fundamental CPU scheduling algorithm are -

1. FCFS → First come First serve → Non premptive
2. SJF → Shortest Job first → Non premptive
3. SRTF → Shortest Remaining Time first → SJF with pre-emptive
4. Priority Scheduling → Non Pre-emptive
5. Pre-emptive priority scheduling → Premptive
6. Round-Robin Scheduling → Always premptive (FCFS with premption)

Non Fundamental →

7. Multilevel queue scheduling
8. Multilevel feedback queue scheduling

- Non preemptive cpu schedule algo. CPU scheduler is invoked under 2 cond^n
  ①. When a process wish to perform I/O operation & therefore a CPU scheduler is invoked to select next new process
  ②. When a process is terminate then CPU scheduler is invoked to select next new process

- In preemptive CPU schedule algo CPU scheduler is invoked under 3 cond^n
  ①. When a process wish to perform I/O operation & therefore a CPU scheduler is invoked to select next new process
  ②. When a process is terminate then CPU scheduler is invoked to select next new process
  ③. When a process is preempted then CPU scheduler is invoked to schedule next new process

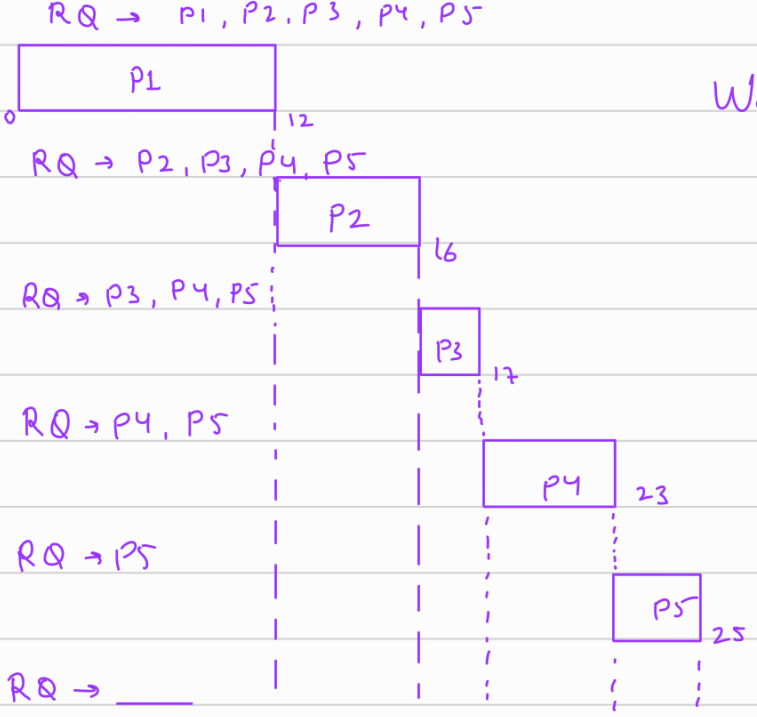
First come First serve →

- The process that arrive in ready queue first will be selected first
- New process join at tail at ready queue
- Implemented uses queue (one of possibility)
- Time comp. will be O(1)
- Also it is easy to impleament

Example

- Consider following set of processes with given arrival time & CPU brust processes

Process	Arrival time (in ms)	CPU burst (in ms)
P1	0	12
P2	2	4
P3	3	1
P4	6	6
P5	8	2



Waiting time ??

P1 = 0-0 = 0 P2 = 12-2 = 10

P3 = 16-3 = 13 P4 = 17-6 = 11

P5 = 25-8 = 17

Avg = 49/5 = 9.8 ms

Turnaround time ??

P1 = 12-0 P2 = 16-2

P3 = 17-3 P4 = 23-6

P5 = 25-8

Avg = 74/5 = 14.8

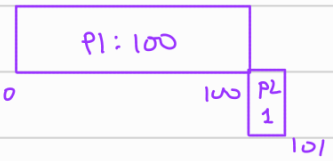
- Waiting time of process is the time elapsed by the process waiting in the ready queue for CPU
- Turnaround time is time b/w^n submission till its completion

Advantage

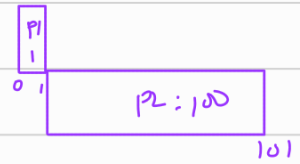
- Its advantage is that it is simple to implement in term of time & space complexity.
- FCFS ensures fairness for process to get a chance

Disadvantage

- large file if comes first and a very small file is waiting for very large amount of time, if small file is being process first then average waiting time can be reduced.



Avg waiting time = $\frac{0+100}{2} = 50$



Avg waiting time = $\frac{0+1}{2} = 0.5$

- FCFS generally suffer from large average waiting time & large turnaround time.
- This effect is known as Convoy effect.
- Consider a scenario where there is a CPU bound process & following this by many I/O bound process, once a CPU bound process selected to run over CPU, it will spend large time over CPU

then during this I/O bound process waiting for it, meanwhile the I/O devie are ideal, once CPU bound is over, some I/O burst will be performed & I/O will be busy follow by I/O bound start & at that time CPU will be mostly ideal this is convoy effect.

- This will cause poor resource utilization

- CPU bound → A process having very large CPU brusts although few in number. (Process spends large time on CPU)
- I/O bound → A process having very short CPU brust although large in number.

- For a proper CPU utilization ∃ a good mix of CPU bound & I/O bound process

Shortest Job First →

- Process having shortest CPU burst is selected from the ready-queve for execution over CPU
- Time complexity is more than FCFS ∴ its quite hard to implement.

priority ∝ $\frac{1}{\text{CPU first.}}$

Process	Arrival Time	CPU burst
P1	0	12
P2	2	4
P3	3	1
P4	6	6
P5	8	2

Turnaround time

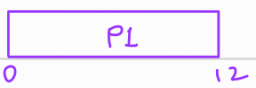
P1: 12-0 = 12 P3: 13-3 = 10

P2: 19-2 = 17 P4: 25-6 = 19

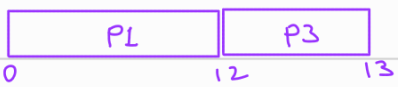
P5: 15-8 = 07

Avg = 13ms

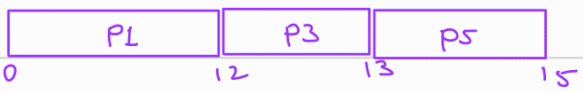
t=0 - RQ → P1



t=12 - RQ → P2, P3, P4, P5



t=13 - RQ → P2, P4, P5



Wating time

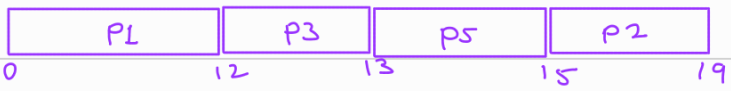
P1: 0-0=0 P3 = 12-3 = 9

P2: 15-2 = 13 P4 = 19-6 =

P5 = 13-8 = 5

Avg = 8

t=15 - RQ → P2, P4



t=19- RQ → P4

- No other scheduling algorithm can have avg. waiting time less then SJF ∴ it is optimal algorithm
- This algorithm is used for benchmarking of other algorithm
- Idea for selection CPU brust, but CPU burst can't be computed earlier, it can only be computed once a burst run over CPU
- Also CPU burst depends on machine.

Q. Consider following set of processes & calculate avg wating time & avg turnaro-und time for FCFS, SJF.

Process	Arrival time	CPU burst
P1	0	5
P2	2	7
P3	4	8
P4	20	1
P5	30	2

1. FCFS

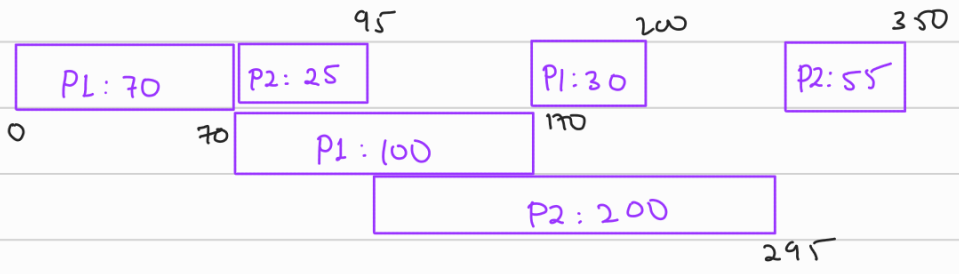
	Turnaround → Complete - arrival	Waiting → begin - arrival
P1	5 - 0 = 5	0 - 0 = 0
P2	12 - 2 = 10	5 - 2 = 3
P3	20 - 4 = 16	12 - 4 = 8
P4	21 - 20 = 1	20 - 20 = 0
P5	32 - 30 = 2	30 - 30 = 0
Avg :-	6.8	2.2

SJF is also same

Q. For given instance calculate Avg. waiting time, Avg. turnaround time cpu utilization & I/O utilization for FCFS, SJF

Progress	CPU	I/O	CPU
P1	70	100	30
P2	25	200	55

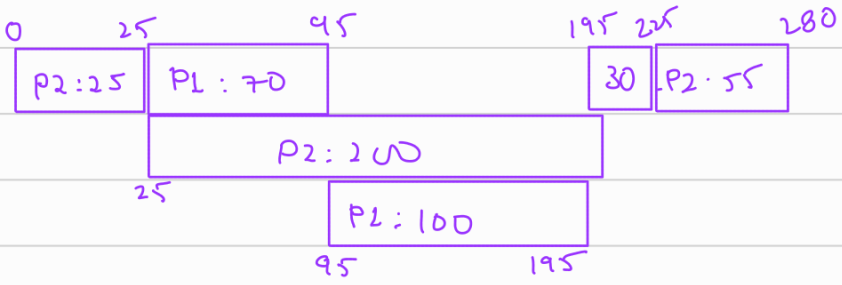
1. FCFS →



	Turnaround	Waiting
P1. CPU 1	70 - 0 = 70	0 - 0 = 0
P1 CPU 2	95 - 0 = 95	170 - 170 = 0
P2 CPU 1	200 - 0 = 200	70 - 0 = 70
P2 CPU 2	350 - 0 = 350	295 - 295 = 0
Avg:	143/2	35

Job ready hai tabhi aaye hai

2. SJF →



	Turnaround	Waiting
P1. CPU 1	95 - 0	25 - 0 = 25
P1 CPU 2	25 - 0	195 - 195 = 0
P2 CPU 1	225 - 0	195 - 195 = 0
P2 CPU 2	280 - 0	225 - 225 = 0

Avg: 125/2 = 62.5 25/2 = 12.5

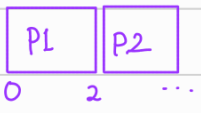
Shortest remaining time first →

- In Shortest remaining time algo., a process having shortest CPU burst is selected from ready queue mean while if there arrives a new proces whose CPU burst is found to be shorter than then CPU burst of currently executing process then newly arrived process will preempt current process takes hold of CPU.

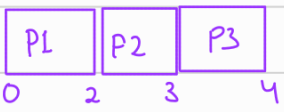
Process	Arrival time	CPU burst
P1	0	12
P2	2	4
P3	3	1
P4	6	6
P5	8	2

t=0 RQ → P1

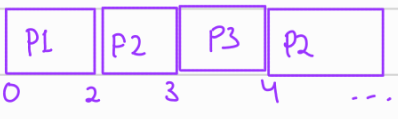
t=2 RQ → P2



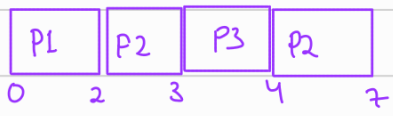
t=3 RQ → P1(10), P3(1)



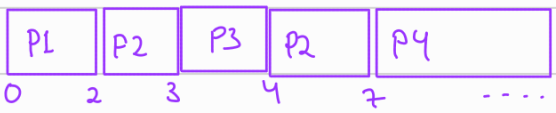
t=4 RQ → P1(10), P2(3)



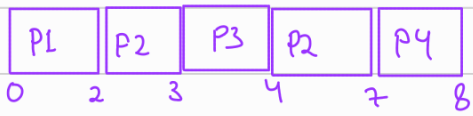
t=6 RQ → P1(10), P2(1), P4(6)



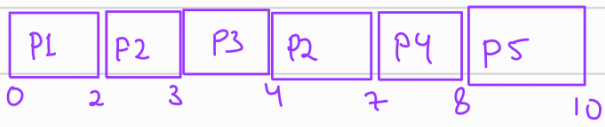
t=7 RQ → P1(10), P4(6)



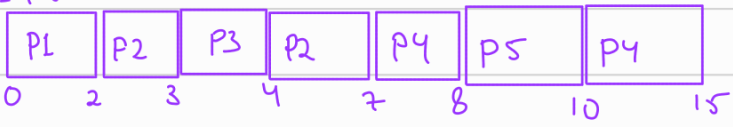
t=8 RQ → P1(10), P4(5), P5(2)



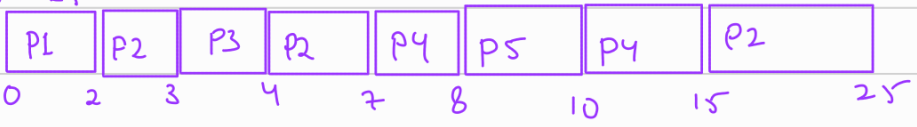
t=10



t=15 RQ → P1(10), P4(5)



t=25 RQ → P1(10)



Waiting P1: (0-0)+(15-2) = 13
P2: 2-2 + 4-3 = 1
P3: 3-3 = 0
P4: (7-6)+(10-8) = 2
P5: 8-8 = 0

Avg = $\frac{13+1+}{5}$ = 3.4

- It suffers from problem of starvation → Indefinate (not ∞) waiting
- In real process keeps of comming, if we implement SRTF or SJF then some large CPU burst will be in starvation.

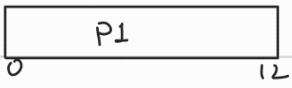
Priority Scheduling →

- Each process assigned a value stored in process control block of corr. process
- In general high no. mean higher priority & low no. mean low priority unless stated.

Process	Arrival time	CPU burst	Priority
P1	0	12	0
P2	2	4	1
P3	3	1	2
P4	6	6	3
P5	8	2	4 (Highest)

- Select a process from Ready Queue ∵ It is non preemptive ∴ process leave CPU after completion.

t=0 RQ → P1 (0, 12)

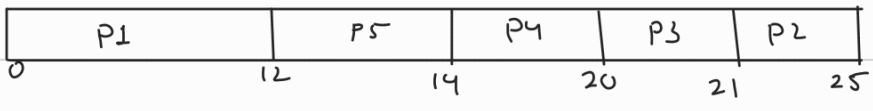


t=2 RQ → P2 (1, 4)

t=3 RQ → P2 (1, 4), P3 (2, 1)

t=6 RQ → P2 (1, 4), P3 (2, 1), P4 (3, 6)

t=8 RQ → P2 (1, 4), P5 (2, 1), P4 (3, 6), P5 (4, 2)



Waiting time :-

P1: 0 - 0 = 0

P2: 21 - 2 = 19

P3: 20 - 3 = 17

P4: 14 - 6 = 8

P5: 12 - 8 = 4

Avg : 9.6

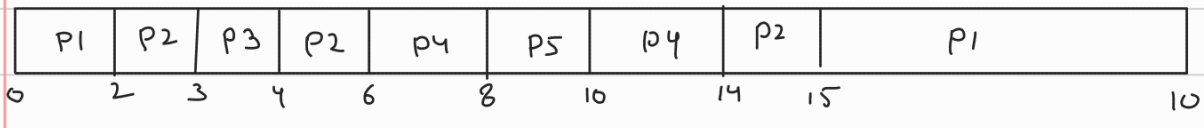
Turnaround time →

Avg : 14.6

Premptive Priority Scheduling

- Each process assigned a value stored in process control block of corr. process
- In general high no. mean higher priority & low no. mean low priority unless stated.
- Select a process from Ready Queue ∵ It is preemptive ∴ high priority process prempt the low priority process & highest priority process run & low priority process join ready queue again.

Process	Arrival time	CPU burst	Priority
P1	0	12	0
P2	2	4	1
P3	3	1	2
P4	6	6	3
P5	8	2	4 (Highest)



Waiting time:-

P1: (0-0) + (15-2) = 13

P2: (2-2) + (4-2) + (14-6) = 9

P3: (3-3) = 0

P4: (6-6) + (10-8) = 2

P5: (8-8) = 0

Avg : 24/5 = 4.8

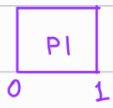
- Aging → the priority of low priority process gradually ↑ after certain duration so at some time it become high priority process to solve problem of starvation
- At a time only lowest priority process aging can be applied and in case of multiple lowest process FCFS can be done for aging.

Round robin Scheduling →

Process	Arrival time	CPU burst
P1	0	12
P2	2	4
P3	3	1
P4	6	6
P5	8	2

Assuming time quantum TQ = 1 ms

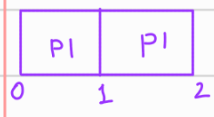
t = 0 RQ → P1 Running →



premptive

t = 1 RQ → Running → P1

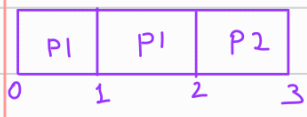
t = 1⁺ RQ → P1 Running →



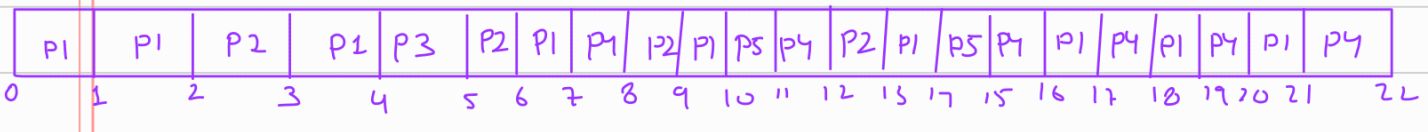
t = 2⁻ RQ → P2 Running → P1

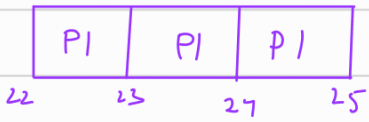
t = 2 RQ → P1, P2 Running →

- Selection from head
- Running join RQ at end
- Process prempt after TQ



t = 3 RQ → P3, P1 Running → P2 Preemp





Q. Calculate Avg waiting time using TQ = 1ms for given instance

Process	Arrival time	CPU burst
P1	0	5
P2	1	2
P3	3	6
P4	6	3
P5	8	4

Multi level queue scheduling →

i.e ready queue partion in seperate several queue

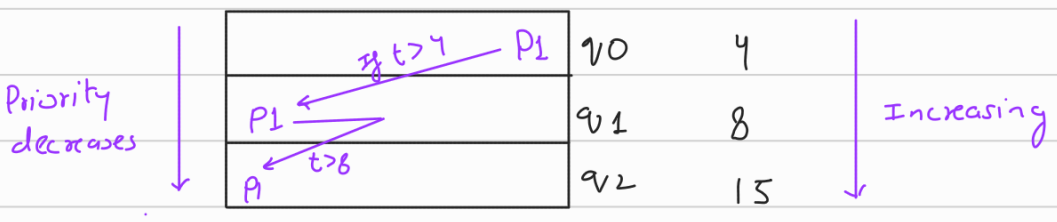
- There are multiple level of queue, i.e multiple ready queue
- Process partion is done on basis of nature of process (Background process, foreground process, priority based process, system process)
- Why not to make several seprate queue for different type of process

Background	q0	let FCFS
Foreground	q1	let Round robing
system process	q2	let Premptive priority
etc		

- All queue can have their own CPU scheduling algorithm
- System process have complete priority over other queue
- These queues themselves can be scheduled premptive priority by assigning absolute priority to the queues

Multi level feedback queue scheduling →

- Ready queue is partitioned in several seprate queues
- Each queue has got some specified value of time associated to each queue

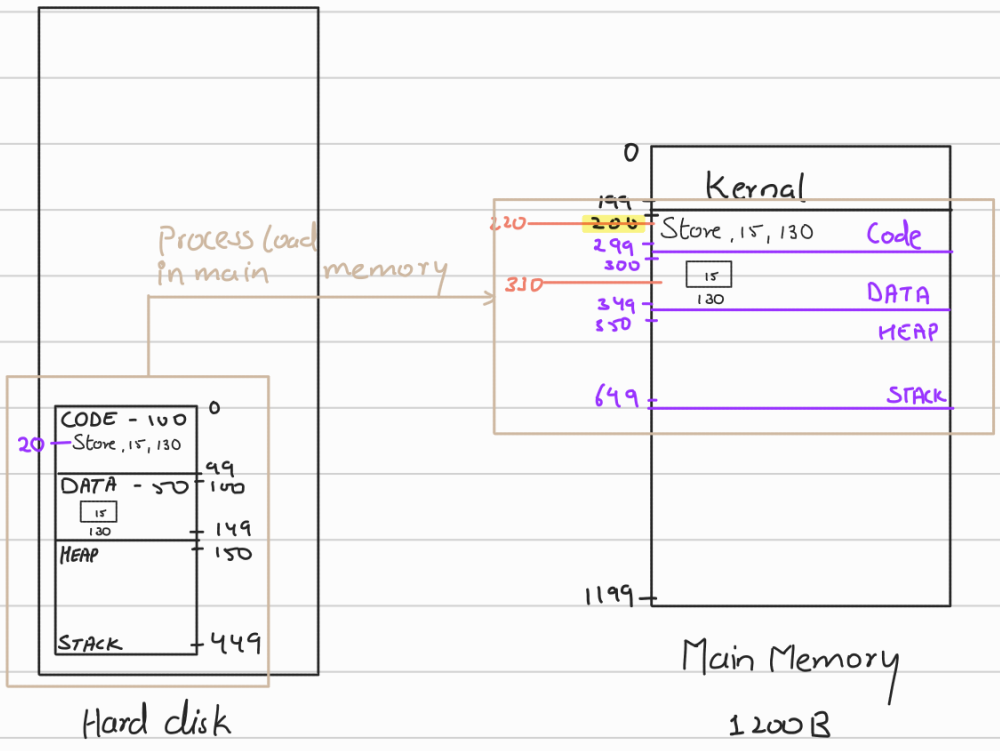


- Each queue has a fixed priority or 1/time
- If a process arrive with time ≤ 4 it joins q0, else if time

is more than 4 ms will give feedback to another priority queue & if it is more than 8 ms it will give feedback & push process to next queue

- This will align lesser CPU burst process at higher priority
- Every process joins at largest priority queue. If CPU burst req. of the process is found to be larger than time value associated to queue it will then the feedback at the tail of lower priority queue (The idea is to schedule processes having short CPU burst with larger priority

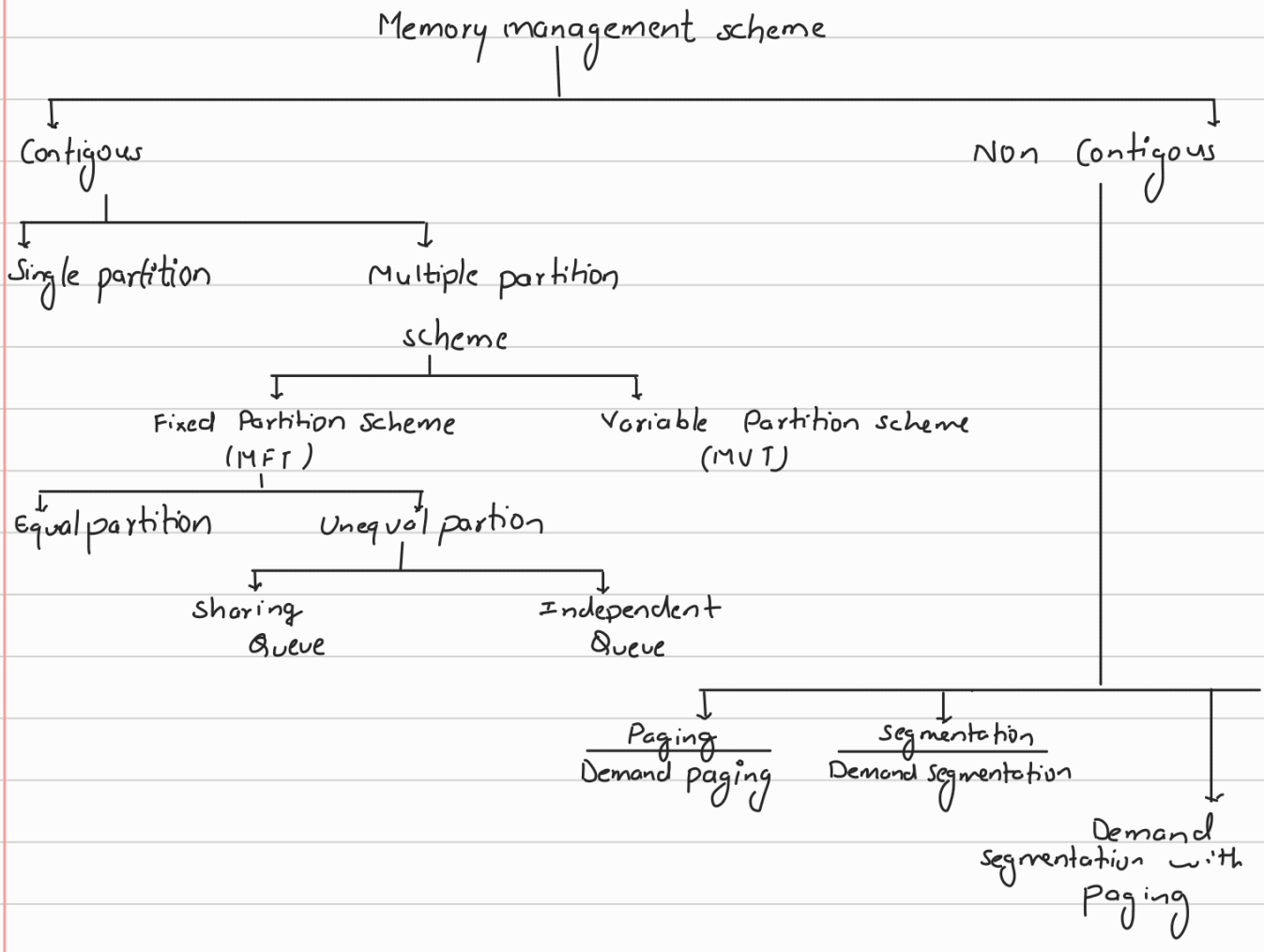
Memory management →



- logical address need to convert to physical address in main memory add (20) in hard disk process convert to 220 in Main Memory & 130 add of process changes to 200+130 = 330

- Before conversion of logical address we need to ensure that it is a valid logical address to ensure memory protection.

- We need to take care how process need to handled in main memory

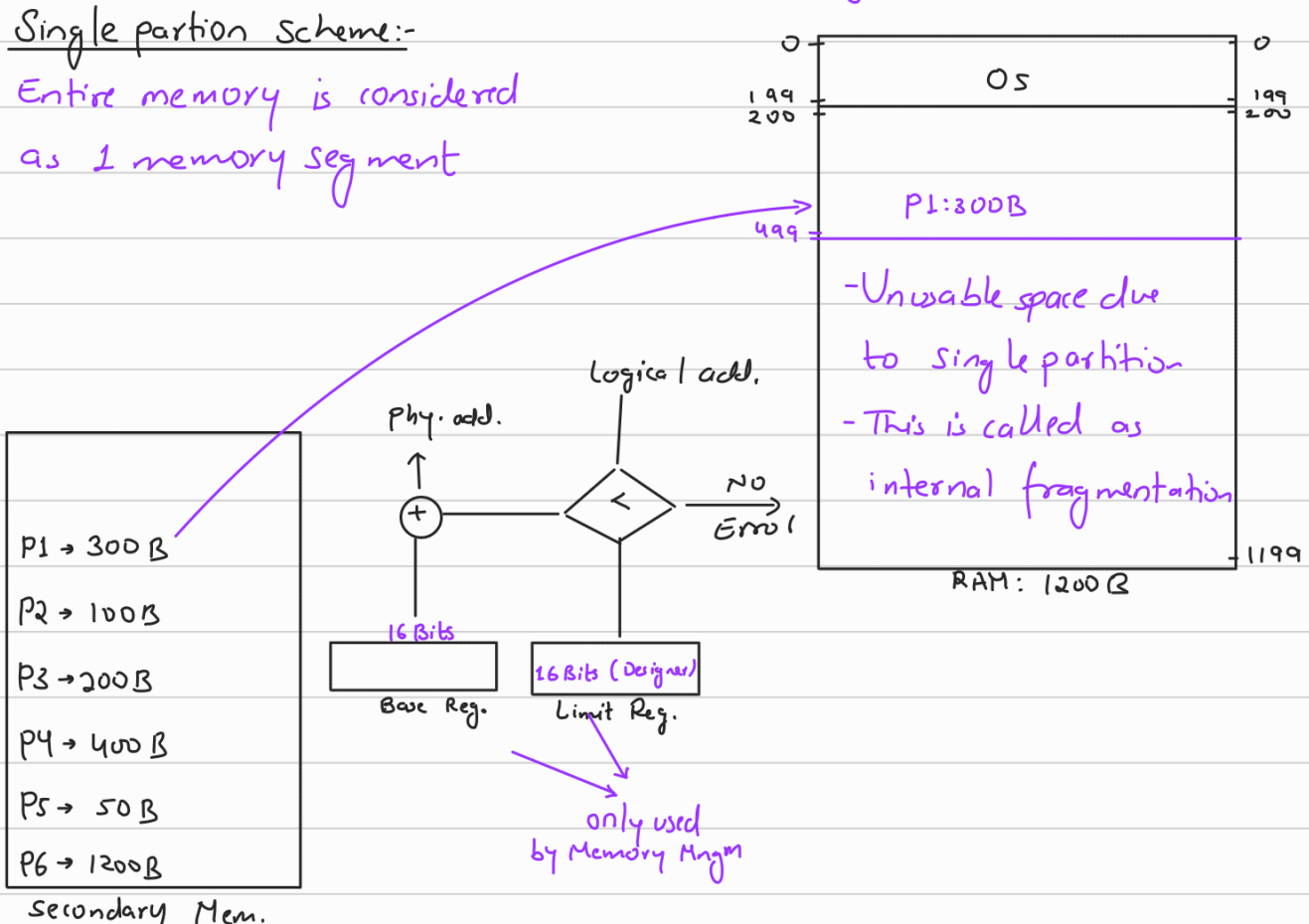


Contigous →

- In this Memory Management scheme process start up from a byte address and then extend it from there till completion of process

Single partion Scheme:-

- Entire memory is considered as 1 memory segment



Base reg :- Base address of main memory ∴ size = $\lceil \text{size ram} \rceil_{2^n} = \lceil 1200 \rceil_{2^n}$

$= \lceil 2^{11} \rceil$ ∴ 11 bit

but reg are in power of 2 ∴ it is 16 bit

- For P1 : Base reg : 200
  : Limit reg: 300

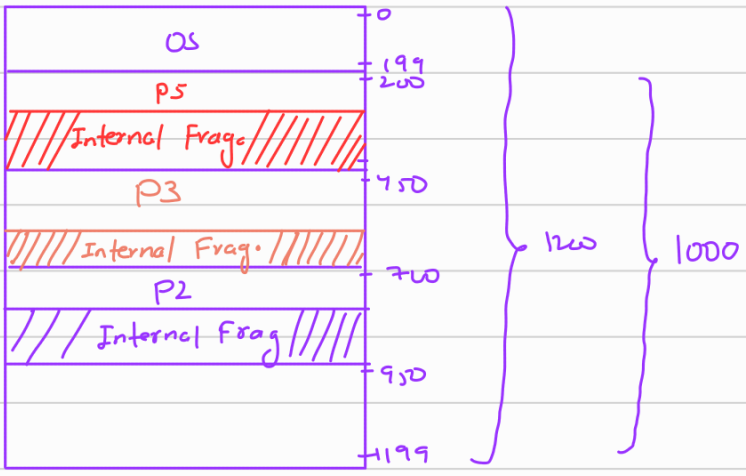
- In single portion scheme entire user space is regarded as single availible largest partion that can accomodate only 1 process
- A process will start from base address of user space in main memory & extend up from there
- Two seprate and dedicated registers called as base and limit will be reserved in the hardware to keep base adds of process in main memory & it size respectively
- For logical to physical add. translation, generated every logical add. will be first compared with the limit field if it is less then base address will be added to generate corre. physical address other wise error will be reported
- The advantage is that it is simple → less info to store ∴ less time for context switching
- The disadvantage is that it is suitable for uniprogramming environment only.
- One more disadvantage : It suffer from large internal fragmentation
- When some process is allocated space in some partition rest of the space within partition is wasted this form of waste is called as internal fragmentation
- We can't directly execute a bigger size process (> user size) unless we use overlays programming techniques
- If programmer realize that a prog can be divided in module such that some of module can be ignored & at any instance size of req. module <=user segment size then we can load req. modules only when required & remove after use this way prog with size > user partition size can be executed.

- Due to this large internal fragmentation we move to mutiple partition scheme

Multiple partition scheme →

- Main memory is divided into multiple
- size of partition can be fix or variable
- The size come under number, ie no. of partition are fix or not
- Considering fix no. partition with fix size

P1 : 300B
P2 : 100B
P3 : 200B
P4 : 400B
P5 = 50B
P6 = 1200B



- P1 can't be loaded directly without overlays
- P2 can be loaded in any partition. (let 700-950) then internal fragmentation = 250 - 100 = 150
- ∵ We need to store data corr. to process base add. & limit add. of diff. partition ∴ a table is made

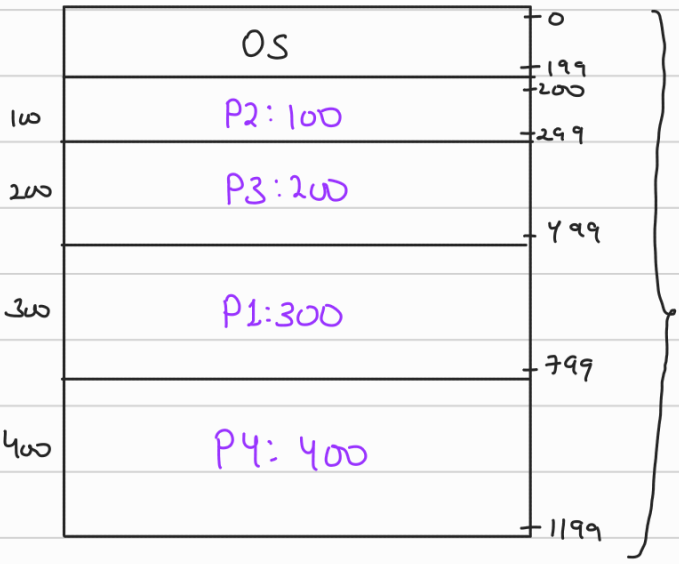
PNo	Base	Limit
P2	700	100
P3	450	200
P4	200	50
-	-	-

- Now P3 can run on any segment left (say 450-700) ∴ Internal frag = 250 - 200 = 50
- Now P4 can't be executed without overlays ∵ size > 250
- Now P5 can be executed.
- Now P6 can't be executed without overlays ∵ size > 250

- Advantage is that extent of internal fragmentation is less.
- Disadvantage → Degree of multiprogramming is fix = No. of partition
  → Some big process can't run due to short size but small process make large Internal fragmentation

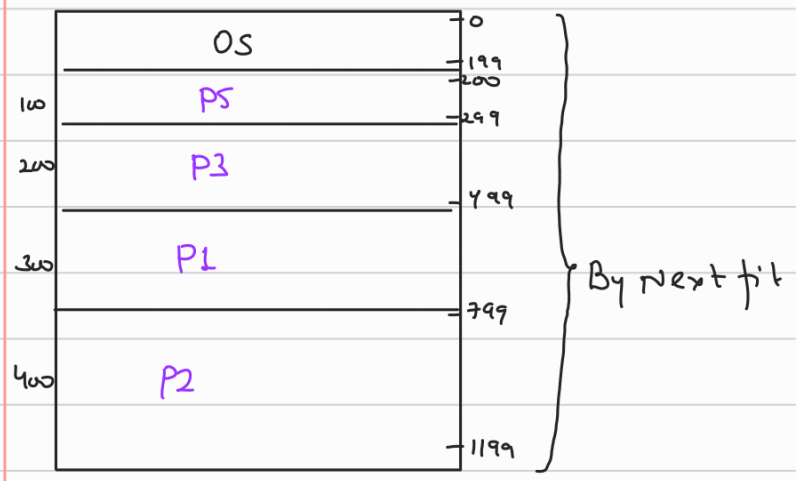
- Considering fix partition with variable size →

P1 : 300B
P2 : 100B
P3 : 200B
P4 : 400B
P5 = 50B
P6 = 1200B

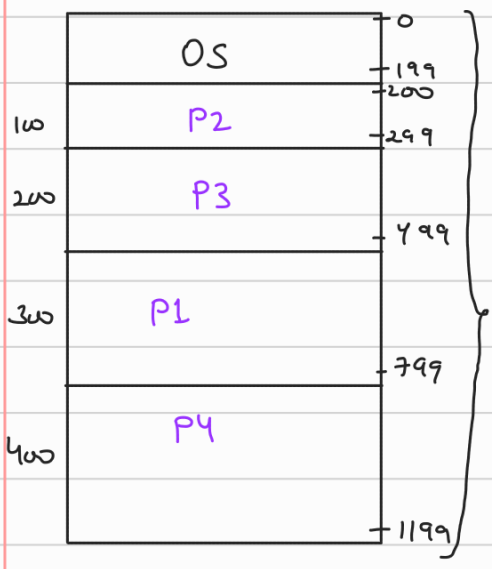


- Now partition need to be selected by some algorithm as all partition are different
- There are 4 partition selection strategy :-
  First fit, Next fit, Best fit, Worst fit

First fit	Next fit	Best fit	Worst fit
Starting from 1st partition scan & select 1 which is vacant & sufficient	It scans from Next to from which prev req. satisfies (In cycle)	selects the smallest partion ie vacant & Sufficient	Select the largest partition sufficient & vacant



For P4 it wait when P2 ends P4 will load in last partition.

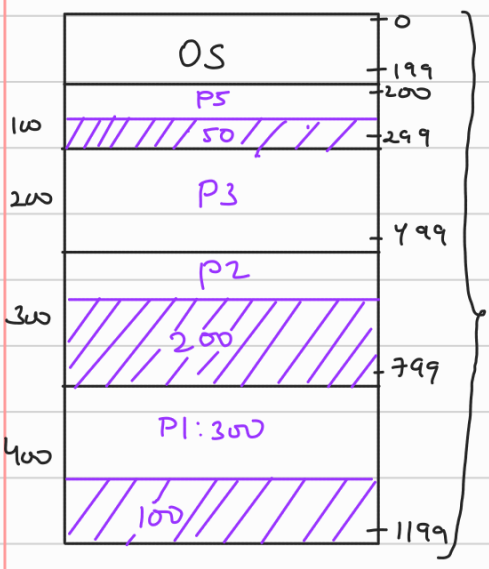


By Best fit

First fit } Better in term of time
Next fit }

Best fit } Better in term of space not good wrt time

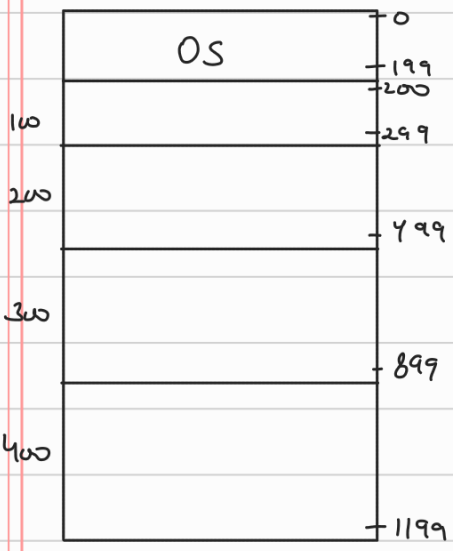
Worst fit } Better in nothing.



By Worst fit

- Now let each partition has it own scheduling queue
- All process size b/w^n certain limit will be scheduled to each partition

Ex:

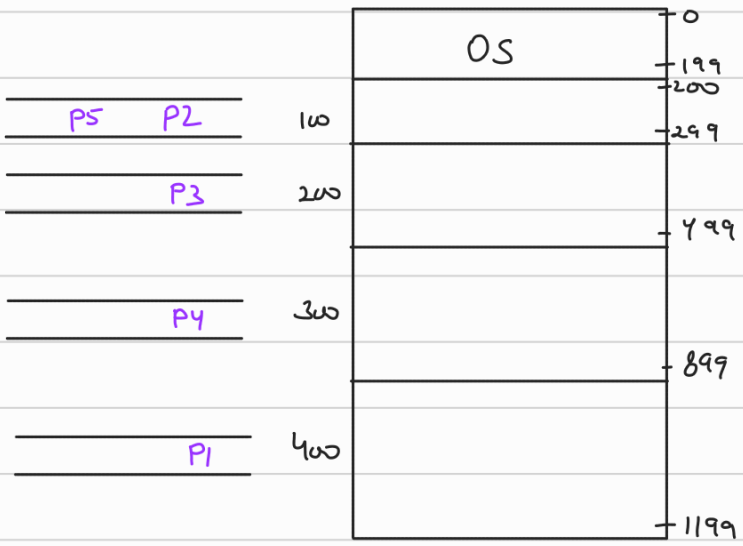


⇒ All process $0 \le sz \le 100$ in queue for this seg.

⇒ All process $100 \le sz \le 200$

⇒ All process $300 \le sz \le 400$

⇒ All process $200 \le sz \le 300$



P1 : 300B
P2 : 100B
P3 : 200B
P4 : 400B
P5 = 50B

- Compile time binding →
- When we are compiling a prog. & at that time we know where is going to be stored ie logical add. assigned to physical add. in compile time is called as compile time binding

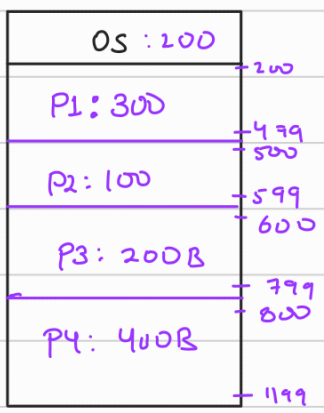
- In above case also when process is created we know where the process is going to be created ∴ Compile time binding is possible due to this compile time may be little higher but execution will be faster

- If compile time binding is possible then we can push it upto load time bind and can be push upto runtime binding.

MVT →

- Dynamic partion scheme
- Multiprogramming with variable no of task
- I a single largest free space called whole
- When process arrived the required size partition created dynamically

Pid	Base	Limit
P1	200	300
P2	500	100
P3	600	200
P4	800	400



P1 : 300B
P2 : 100B
P3 : 200B
P4 : 400B
P5 = 50B

- After this say P2 & P4 terminate the ∃ 2 wholes then ∃ 4 strategy Best fit, Next fit, First fit & Worst fit then next process select 1 of the whole

- Say a new process arrived whose size = Σ free size in RAM but are not contiguous then process can't load ∴ It is called external fragmentation

- If two whole are consequtive due to termination of processes they will merge to form single large whole

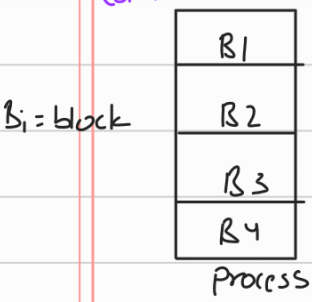
- Compection may be used to resolve external fragmentation problem but it is rarely opted
- Compection → sarri process jo ho rahi ek taraf le jao khali ek traf kardo
- But during compection process will be extremely slowdown

- Sometime if a process of 100, 98 B process came and 100 wholes non contiguous free up then instead of making 2B, 100 wholes which can't hold process (may be decided by OS designer) therefore OS designer to avoid multiple entries in whole table that ↑ search time the OS designer may allocate these 98 B processes a 100 B space to make Internal fragmentation, to avoid increase in search time of whole.

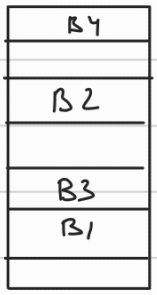
- All contiguous management has a restriction of contineous partition ∴ We move to non-contiguous memory management

Non contiguous memory management scheme →

- A process is allowed to be broken down into block which can be stored seprately in main memory



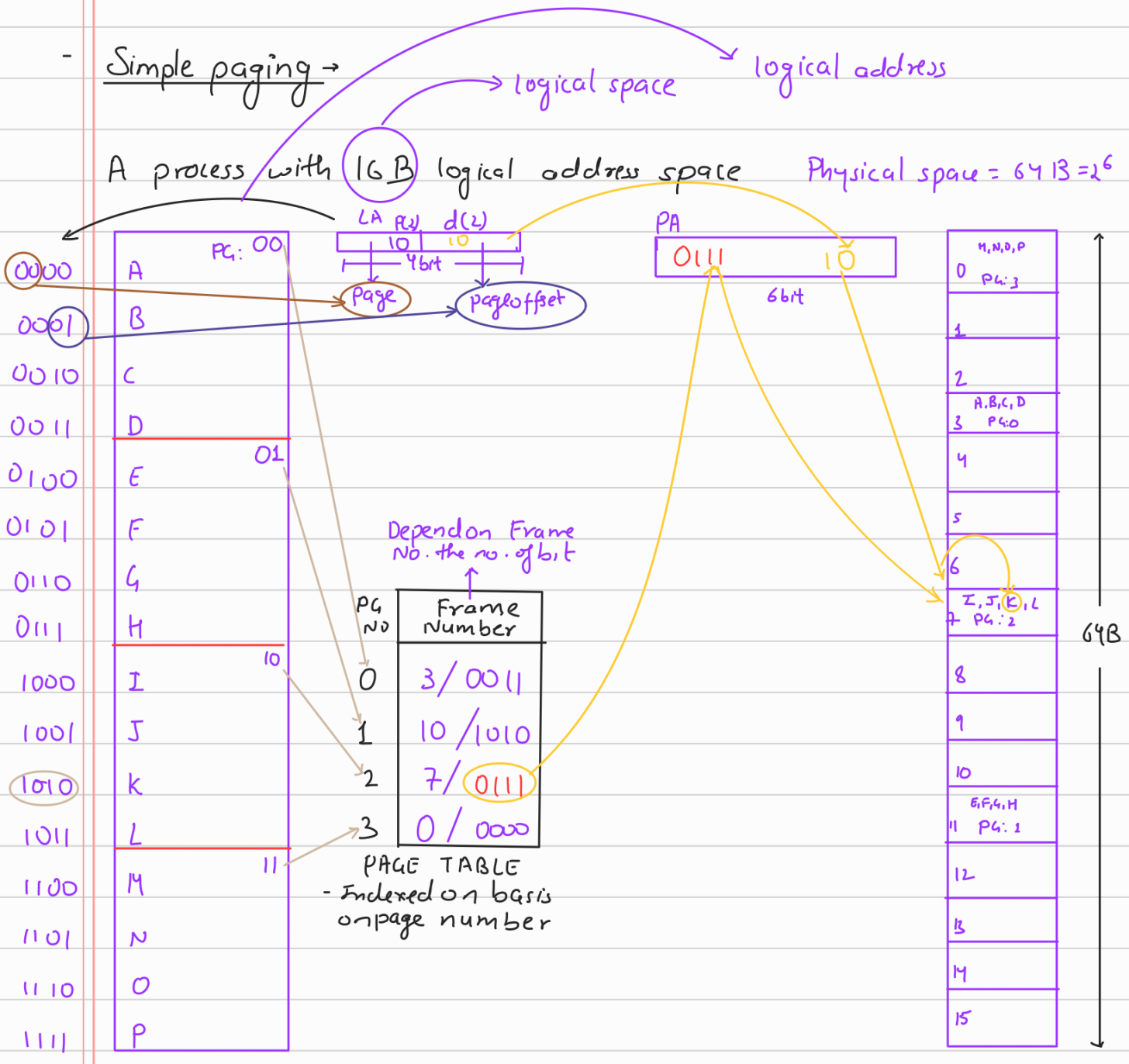
B_i = block



1 entry for each block need to store ∴ May suffer from high context switch time

- PAGING →

- Simple paging →



Consider Aj = 1B

let page size = 4B

No. of pages in L.Add. space = $\frac{16}{4}$ = 4 pages = (2 bits) Add.

No. of frames in PAS = $\frac{64}{4}$ = 16 frames = (4 bits) Add.

- logical space of process divide in fix size block called page & physical address space is also divided in fix block size called frame also page size should be equal to frame size.
- Each page of process is loaded in any of the frame in main memory

Organization of page table →

1. Page table of process is stored in set of dedicated registors
   - Accesing page table will be fast (generally 10ns)
   - Effective main memory access time will be 10 ns + 100 ns storage access is 10 times approx
   - Set are dedicated chips are limited in numbers & are limited in size ∴ large page table can't be stored
   - ∴ large process can't be executed ∵ it will have large page table
   - It suffer from large context switch time ∵ entire page table is saved in process control block corresponding to process & new process table need to be created.

2. Page table is stored in the main memory itself and its base address is stored in a dedicated registor called as PTBR (page table base reg.)
   - let PTBR ref to frame 5 then its add in. main memory is 0101 00 → PTBR value
   - When page no added with PTBR we got required value & now frame no. is obtained & page offset is concatenated to generate physical address
   - Effective address time = 100ns + 100 ns = 200ns
     (100 ns: ↓ M.M Page table; 100 ns: ↓ Byte itself in M.M)
   - Only 1 register ∴ we don't consider its time
   - Context switch time is less compared prev one
   - Memory access time is too low

If not stated use this Pg. table

3. Page table is stored in main memory itself & some of entries of page table are stored in a fast content addrosable memory called as TLB (Translation Lookaside Buffer)
   - Is always for accomodating page table entries of processes

Pid	Page no	Frame no
p1	3	f1
⋮	⋮	⋮

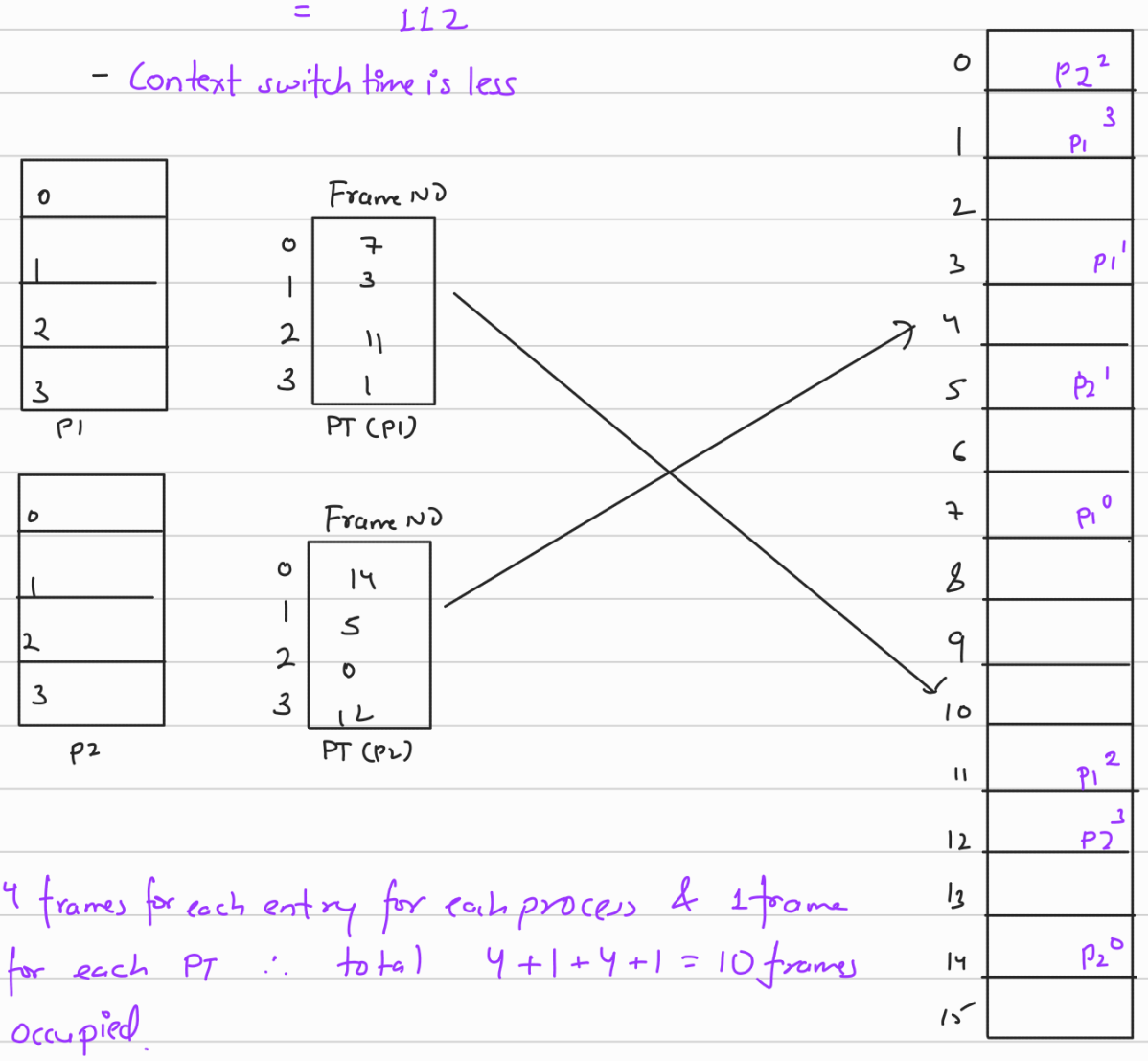
- First pg no. & pid matched in TLB if found we got frame no from it (Almost 98% time it happen) else general methode continue.

- $\therefore$ Time = $0.98\,(\underbrace{10+100}_{\text{TLB hit}}) + .02\,(\underbrace{10+100+100}_{\text{TLB miss}})$

$= 107.8 + 4.2$

$= 112$

- Context switch time is less



4 frames for each entry for each process & 1 frame for each PT $\therefore$ total $4+1+4+1 = 10$ frames occupied.

Pid	Pg	frames
P1	3	f1
⋮	⋮	⋮

TLB

only pg can't be stored overlap b/w diff process

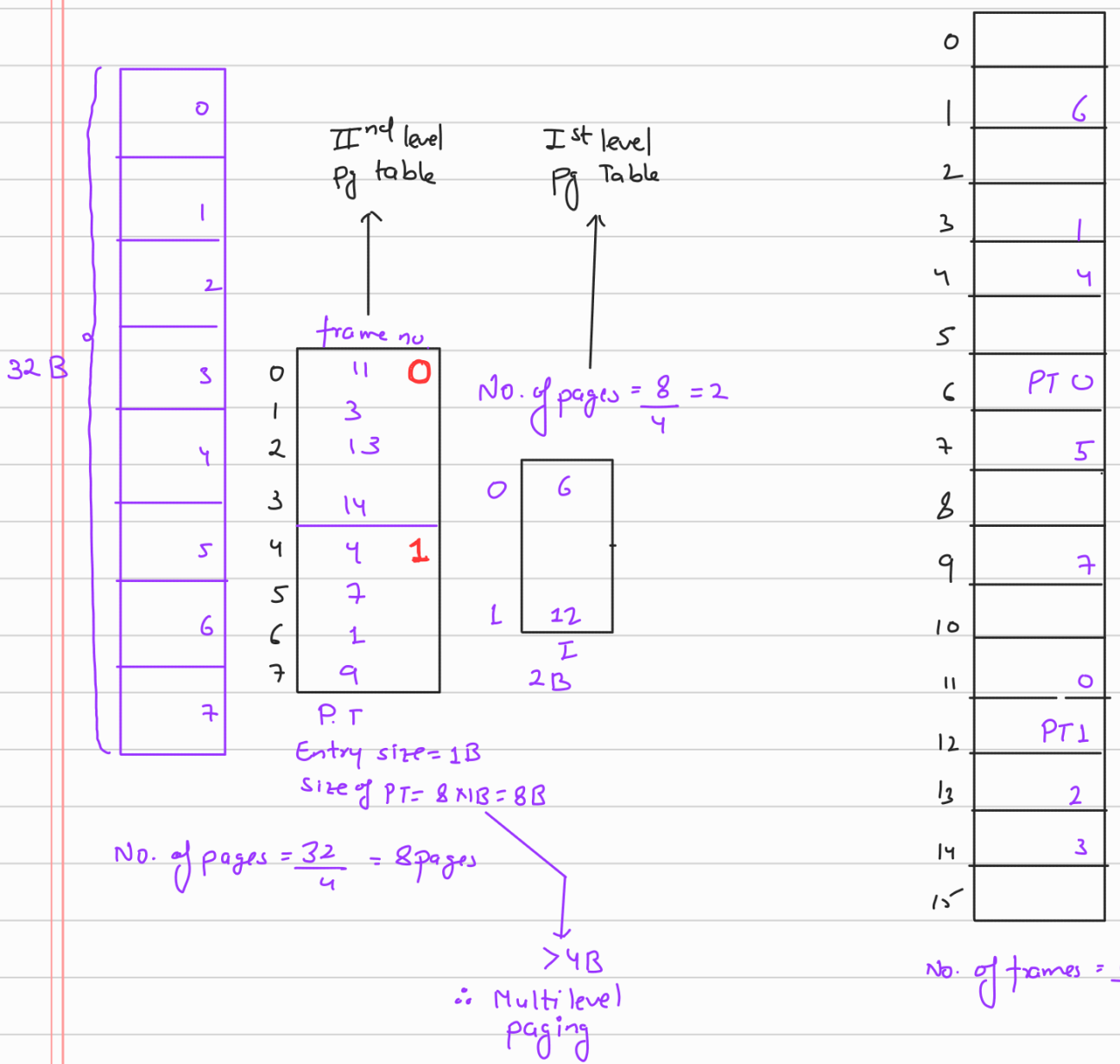
- If TLB is full & a new entry found then it will overwrite

Multilevel paging →

logical address space = 32B

page size = 4B

Physical add. space : 64B



- Each page of page table store in any of frame in main memory
- If size of i^{th} level > page size it will further break into another level of page table until size become ≤ page size

Assignment

Q. Consider a 4GB process with page size 2KB each entry in page table is 4B wide calculate main memory is 8GB calculate

(i) No. of bits used for indexing each level of page table and page offset

(ii) Size of each level of page table and total size of page tables

(iii) No. of entries in each level of page table

(iv) No of entries per page in each level of page table

(v) Total amount of internal fragmentation if any

(vi) Number of frames occupied by process and its page tables

(vii) Assuming some of the entry of page table is stored in fast content add. memory called as TLB having hit ratio 98% & access time 10ns main memory has an access time 100ns calculate effective access time

Process = 4GB = 4×2^{30} B = 2^{32}B LA = 32 bit

page size = 2×2^{10} B = 2^{11} B

entry size of pg table = 4 B = 2^2 B

Main Memory = 8×2^{30} B = 2^{33} B

No. of pages = $\frac{4 \times 2^{30}}{2 \times 2^{10}} = 2 \times 2^{20} = 2^{21}$

Each entry is 4B wide ∴ size = $2 \times 2^{20} \times 4 = 2^{23}$ B

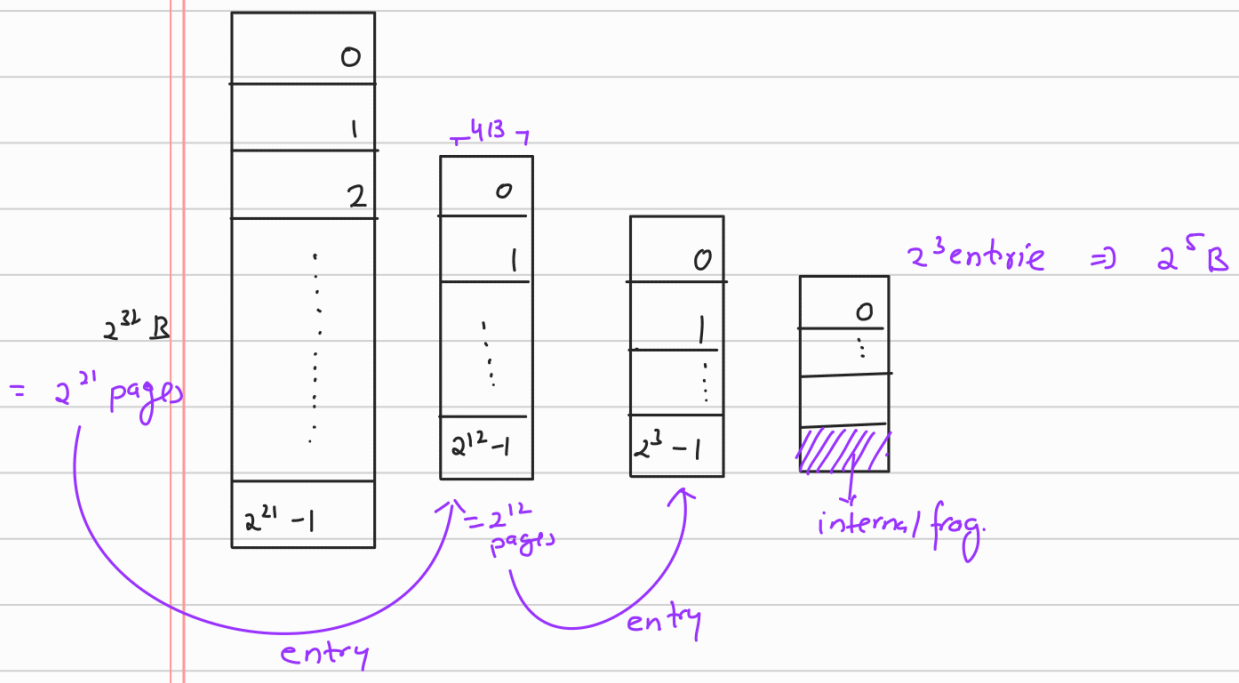
∵ sz > 2×2^{10} ∴ we page the page table

No. of pages = $\frac{2^{23}}{2^{11}} = 2^{12}$

sz = 2^{14} B > 2^{11} ∴ we agin page this level

No. of pages = $\frac{2^{14}}{2^{11}} = 2^3$

∴ sz = 2^5 < 2^{11} ∴ this is last level



(i) 3 bit lvl 1, 9 bit lvl 2, 9 bit lvl 3, 11 bit page off.

(ii) 32 B, 2^{14} B, 2^{23} B

(iii) 2^3, 2^{12}, 2^{21}

(iv) 2^3, $2^9 = \left(\frac{2^{12}}{2^3}\right)$, 2^9

(v) 204 B - 32

(vi) $2^{21} + 2^{12} + 2^3 + 1$

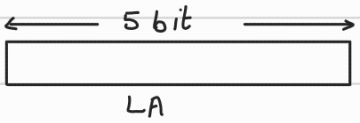
(vii) $0.98(10+100) + 0.02(10 + \overbrace{100 + 100 + 100}^{\text{lvl}} + 100)$

$= 0.98(110) + 0.02(410)$

$= 116$

Valid/Invalid bit (in page table) →

let process size = 20 B



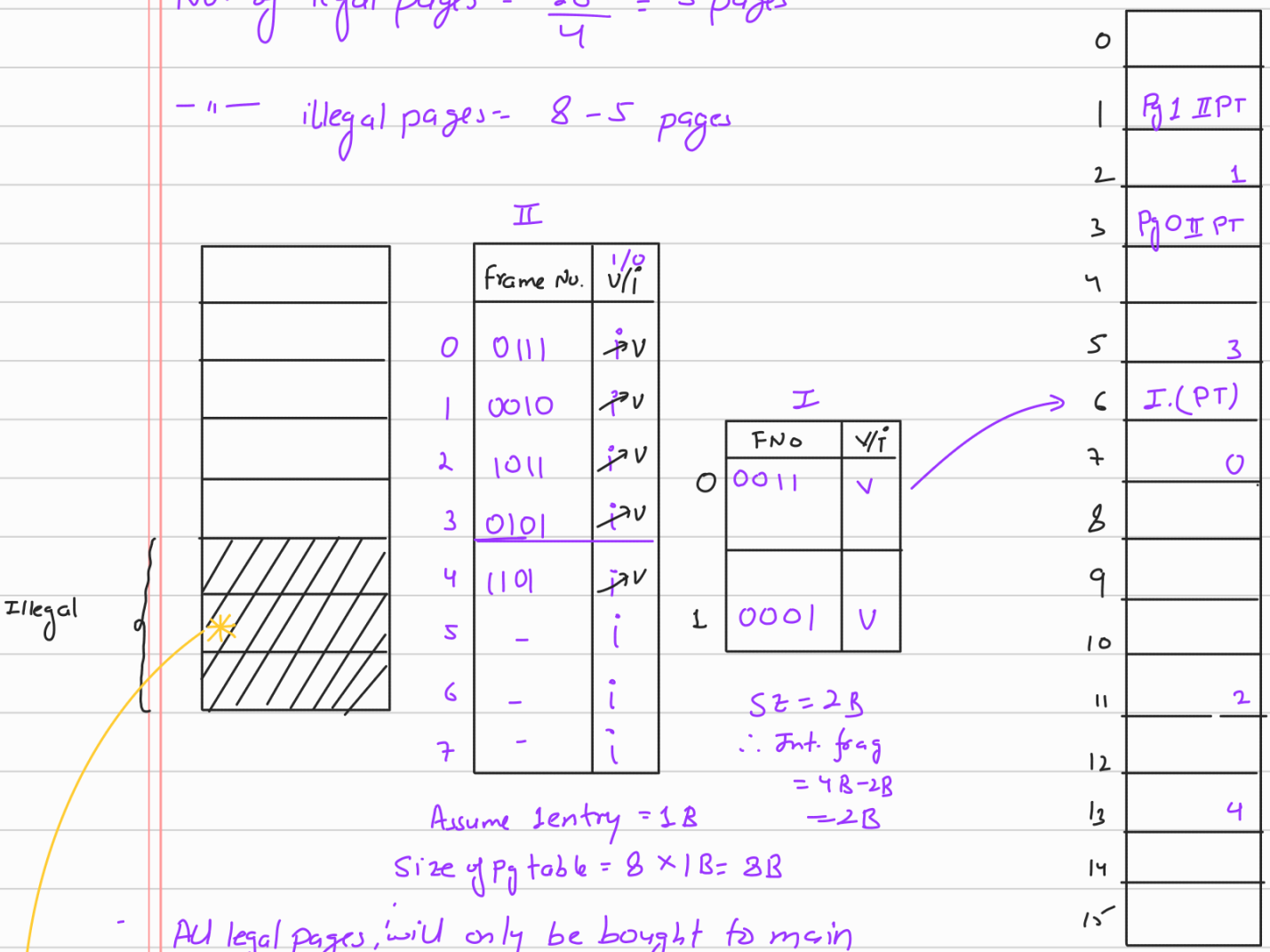
page size = 4 B, mm = 32 B

Total no. of pages = $\frac{32\text{ B}}{4\text{ B}}$ = 8 pages

All pages will not be valid pages.

No. of legal pages = $\frac{20}{4}$ = 5 pages

— " — illegal pages = 8 − 5 pages



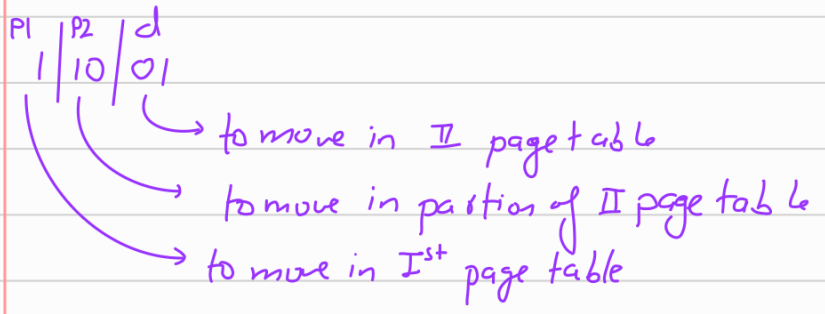
Assume 1 entry = 1B

Size of Pg table = 8 × 1B = 8B

- All legal pages will only be bought to main memory

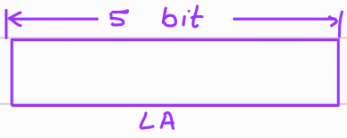
No of pages of page table = $2^3/2^4 = 2^1$

- If a single entry in page table is valid then that entry is valid



Virtual memory / Demand paging →

- Virtual memory is a concept implemented wing demand paging
- by this way a process larger than main memory can also be executed
- CPU = $1 - p^n$ P = prob. of I/O n↑ CPU↑



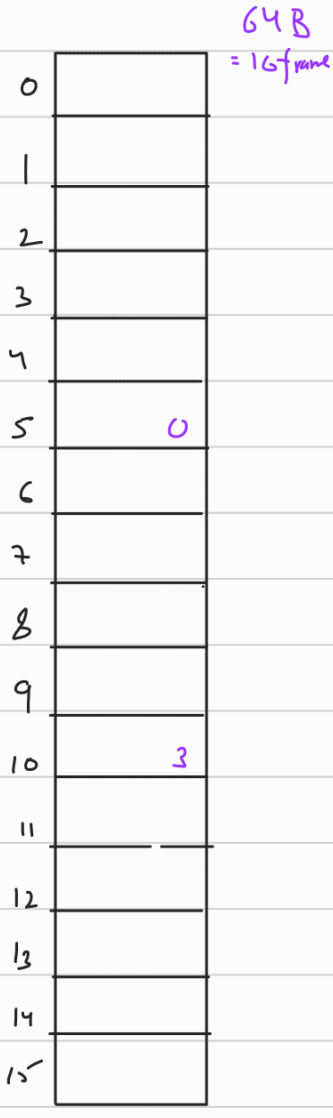
process = 20B ≈ 32B = 2^5

let page sz = 4B

Total no. of pages = 32/4 = 8 pages

legal pages = 5 pg.

	frame	v/i
0	0101	v
1	–	i
2	–	i
3	1010	v
4	–	i
5	–	i
6	–	i
7	–	i



- Principle of locality persist ∴ process can run without loaded fully in main memory for significant amount of time

- In demand paging v/i bit has got 2 meaning :
  - if it is valid corr. to page in page table & it is legal & is loaded in main memory
  - else it can either be invalid or not loaded in main memory

OS can identify why it is invalid & if it is due to page fault & main memory has some free frame the page will be loaded & PT is avalible & if no frames avalible then page replacement take place

- In demand paging not all pages of process are required to brought into main memory., a process can start its execution by having fewer no. of pages in main memory doing so has 2 adv. (legal)

  - A process of size > M.M can also be executed
  - More no. of frames will be left vacant for the use of other process i.e degree of multiprog. may be increased. If degree of multi prog ↑ then CPU utilization will ↑ (Except thrashing)

The drawback is :-

- If a page ie not present in MM is required by the process then it causes a page fault. the faulted page will then be loaded from secondary storage to main memory. If some frame is vacant then faulted page will be loaded to any of vacant frame but if no frame is vacant then some page from the main memory need to be replaced acc. to page replacement algorithm

We can add one more detail to page table i/e modified/Unmodified

	frame	v/i	Modified/ Unmodify
0	0101	v	u
1	–	i	u
2	–	i	u
3	1010	v	u
4	–	i	u
5	–	i	u
6	–	i	u
7	–	i	u

→ Not mandatory but req for efficiency

initially 'u', if a new page come to a frame by replacement then only it is changed to 'm'

while replacement is their if before it the bit is 'u' we can replace by only 1 pg transfer, & if it is 'M' we can replace by 2 pg transfer.

Q. Consider a demand paging system in which a regular memory takes 100ns. 1 out of 10000 pages ref. causes pagefault. page fault service time is 25 ms calculate effective MM access time.

P(No pg fault) = 0.9999

100 ns X 0.9999 + 0.0001 X 25 ms

2.6 µs

- **Modify / Unmodify bit →**
- This bit is set to be unmodified corr. to some page in page table if that page has not been modified since its arrival in main memory. But if the page has been modified since its arrival in the main mem. then this bit is said to be modified.

During page replacement : if it is found that the page to be replaced has not been modified then only 1 pg transfer is req. to bring in faulted pg. from disk to MM but if the pg to be rep. is found to be modified then 2 pg transfer will be required. the replaced pg. will update its copy in the disk & the faulted pg. is brought into the frame of replaced page

Q. Consider a 1000 byte process with page size 100 B, main memory has 4 frames initially all vacant following are the memory addresses generated during execution trace of this prog 502, 409, 615, 416, 805, 55, 215, 620, 815, 419, 690, 415, 88, 317, 214, 813, 615, 415, 211, 720, 109 144, 612, 218, 512, 430, 809, 912, 915

Calculate page fault ratio for the following page replacement algo.

(i) FIFO

(ii) Optimal page replacement algo.

(iii) LRU

0-99	0
100-199	1
	2
	3
1000	4
	5
	6
	7
	8
900-99	9

502, 409, 615, 416, 805, 55, 215, 620, 815, 419, 690, 415, 88, 317

PG:

5	4	6	4	8	0	2	6	8	4	6	4	0	3
5	5	5	5	5	0	0	0	0	0	0	0	0	3
	4	4	4	4	4	2	2	2	2	2	2	2	2
		6	6	6	6	6	6	6	4	4	4	4	4
				8	8	8	8	8	8	6	6	6	6

214, 813, 615, 415, 211, 720, 109, 144, 612, 218, 512, 430, 809, 912, 915

2	8	6	4	2	7	1	1	6	2	5	4	8	9	9
3	3	3	3	3	3	1	1	1	1	1	1	8	8	8
2	8	8	8	8	8	8	8	6	6	6	6	6	9	9
4	4	4	4	2	2	2	2	2	2	5	5	5	5	5
6	6	6	6	6	7	7	7	7	7	7	4	4	4	4

- In FIFO **Belady's Anamoly** is observed.

Our expectation is No. of frame ↑ ⇒ page fault ↓

but in fifo No. of frame ↑ may ⇒ page fault ↑

Q. Consider following page reference storing 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5 calculate no. of page fault assuming main memory has

(i) 4 Frames

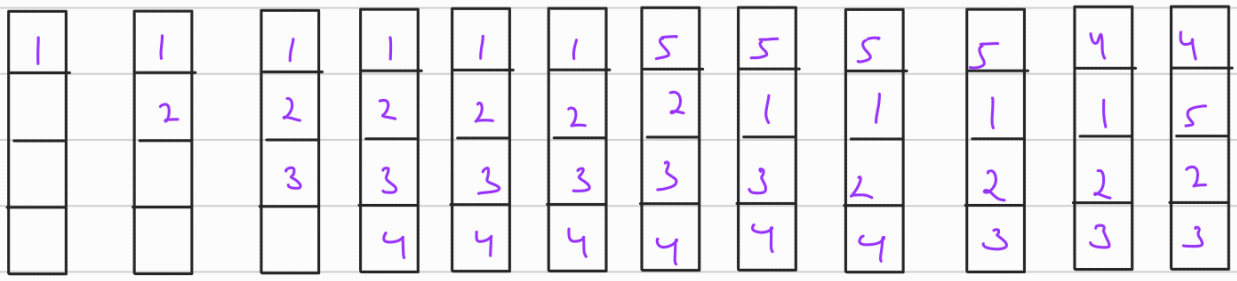
(ii) 5 Frames

(iii) 6 Frames

Assuming FIFO as page replacement algorithm

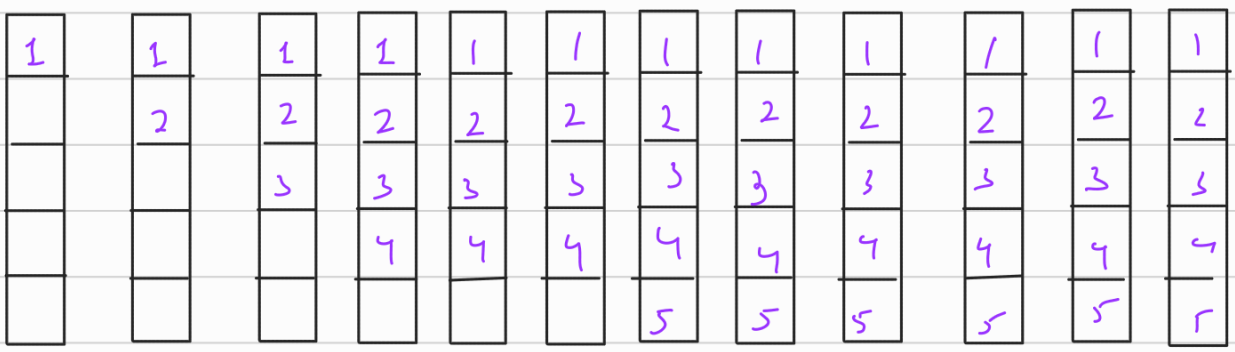
1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

(i)



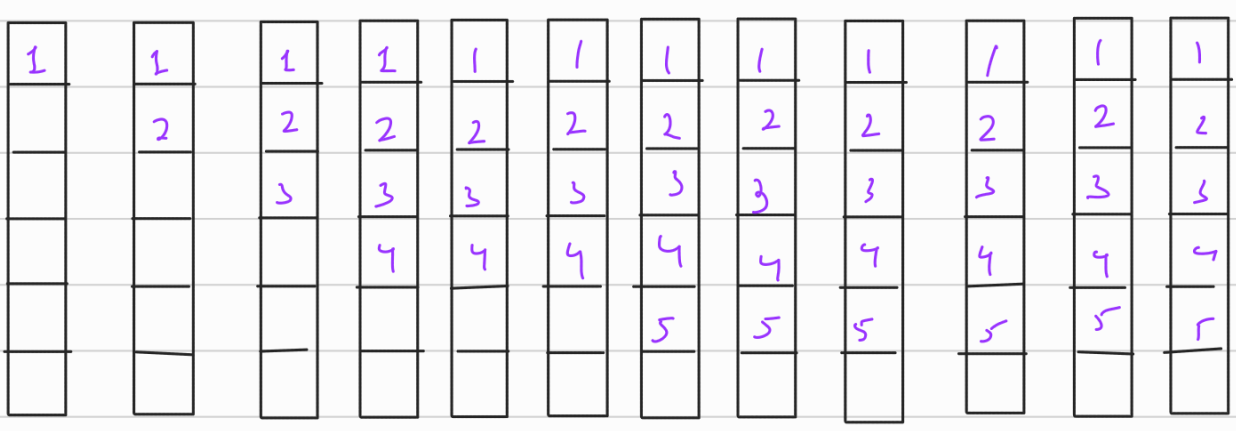
page fault = 10/12

(ii)

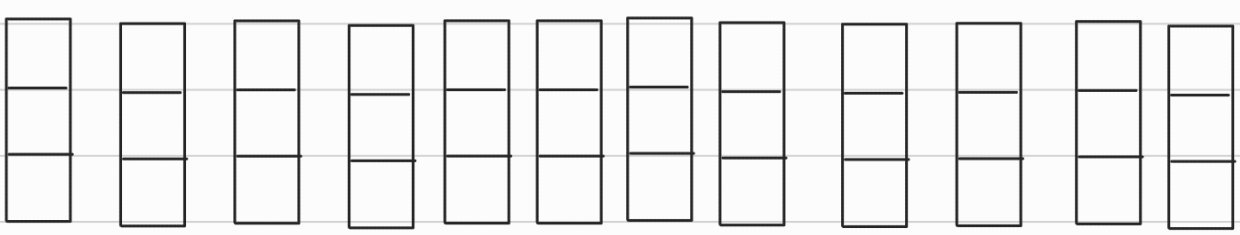


page fault = 5/12

(iii)



(iv) 3 frames



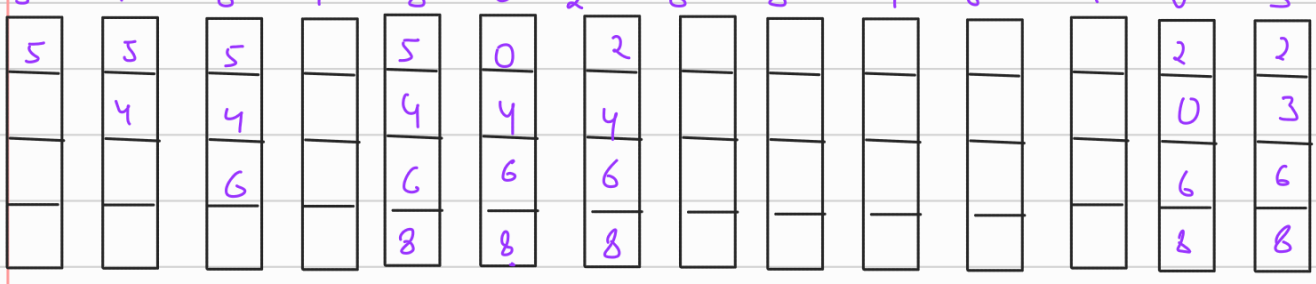
page fault =

Optimal page replacement algo →

- In optimal page replacement the page that is not used for longest period of time will be replaced. (in future)

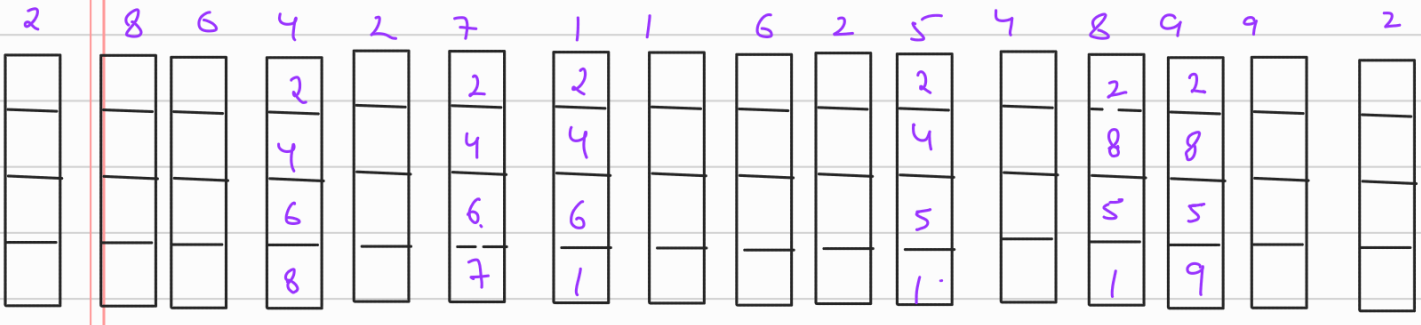
502, 409, 615, 416, 805, 55, 215, 620, 815, 419, 690, 415, 88, 317

PG: 5 4 6 4 8 0 2 6 8 4 6 4 0 3



214, 813, 615, 415, 211, 720, 109, 144, 612, 218, 512, 430, 809, 912, 915, 215

2 8 6 4 2 7 1 1 6 2 5 4 8 9 9 2

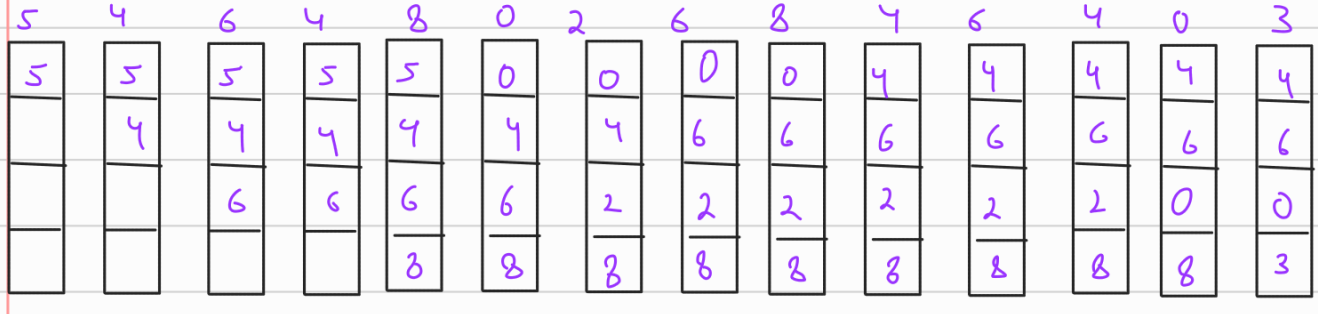


LRU →

- The page that was not used for longest period of time is replaced

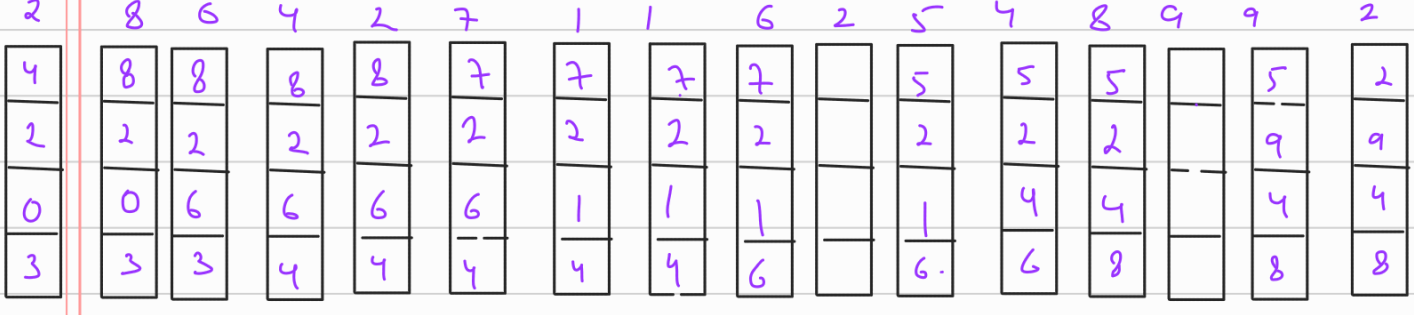
502, 409, 615, 416, 805, 55, 215, 620, 815, 419, 690, 415, 88, 317

PG: 5 4 6 4 8 0 2 6 8 4 6 4 0 3



214, 813, 615, 415, 211, 720, 109, 144, 612, 218, 512, 430, 809, 912, 915, 215

2 8 6 4 2 7 1 1 6 2 5 4 8 9 9 2



page faut = $\frac{22}{30}$