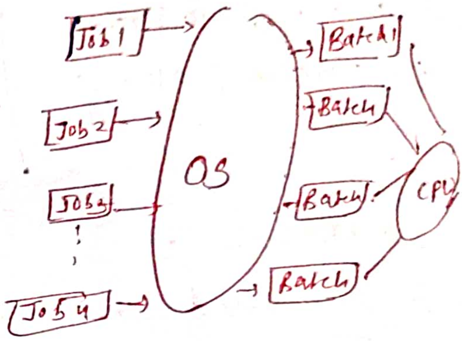
OS → CPU & disk schedu...
→

→ platform required for various software.
→ Acts as a bridge between the user of computer and Hardware
→ Purpose is to provide an environment in which user can execute Program efficiently & conveniently.

Types →

① Batch OS →

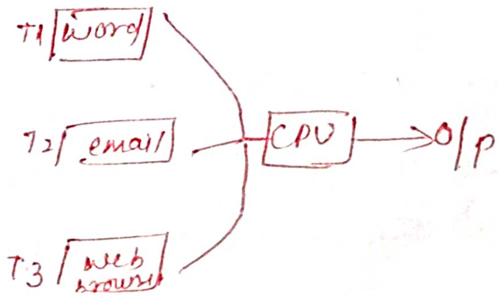


→ Doesn't interact with the computer directly.
→ An operator, take similar job, have same requirement
→ Multitasking is not possible.
→ And OS gives it to CPU for further process
→ ex - Bank statements, payroll system etc.

Advantage
① Easy to manage
② multiple user can use
③ Idle time is very less.

DisAdvantage
① Hard to debug
② Sometimes costly
③ Operator should know batch system.

② Time sharing OS: —



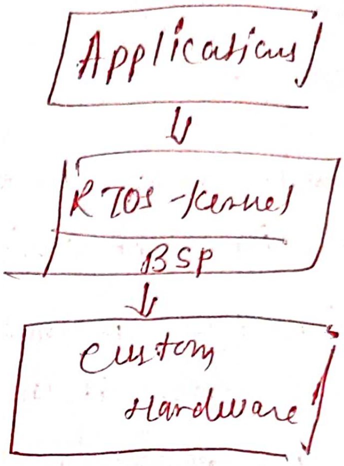
→ Each task given time to execute, so all tasks work smoothly
→ Each user gets time of CPU, use single system
→ Multitasking is possible.
→ Task can be from single user or different user.
→ Time that each task gets executed quantum time.
→ After Time interval OS switch to next task
Ex → Multics, unix etc.

Advantage
① Each task get equal opportunity
② Idle time can be reduces.
③ Fewer chance of duplication

Disadvantage
① Data communication problem
② Reliability problem.

③ Real time OS →

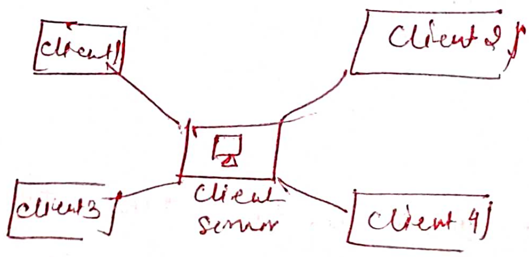
→ Time play ~~an~~ major role here.
→ OS, serve real time system
⇒ The time interval required to process or response I/P very small.
→ used when time requirement are very strict.
robots, missile system, air traffic control etc.
→ Two type → Hard → OS for application Time constraints are very strick, airbags
soft → Time constrains are less strict.



Advantage	Disadvantage
① Error free system	① costly
② Memory allocation is best managed	② complex algorithm.

④ Network OS →

→ Also known as window server
→ System runs on a server.
→ provide capability to manage data, user, security, groups, application and other networking system function.
→ client server model.



⑤ Distributed OS →

→ Independent systems posses their own CPU & memory unit
→ Systems processors differs in size & function

Benefit of working with this OS., always possible that one user can access the files or S/W which is not ~~on~~ present on ~~the~~ his system. but on some other system.

Process / Program

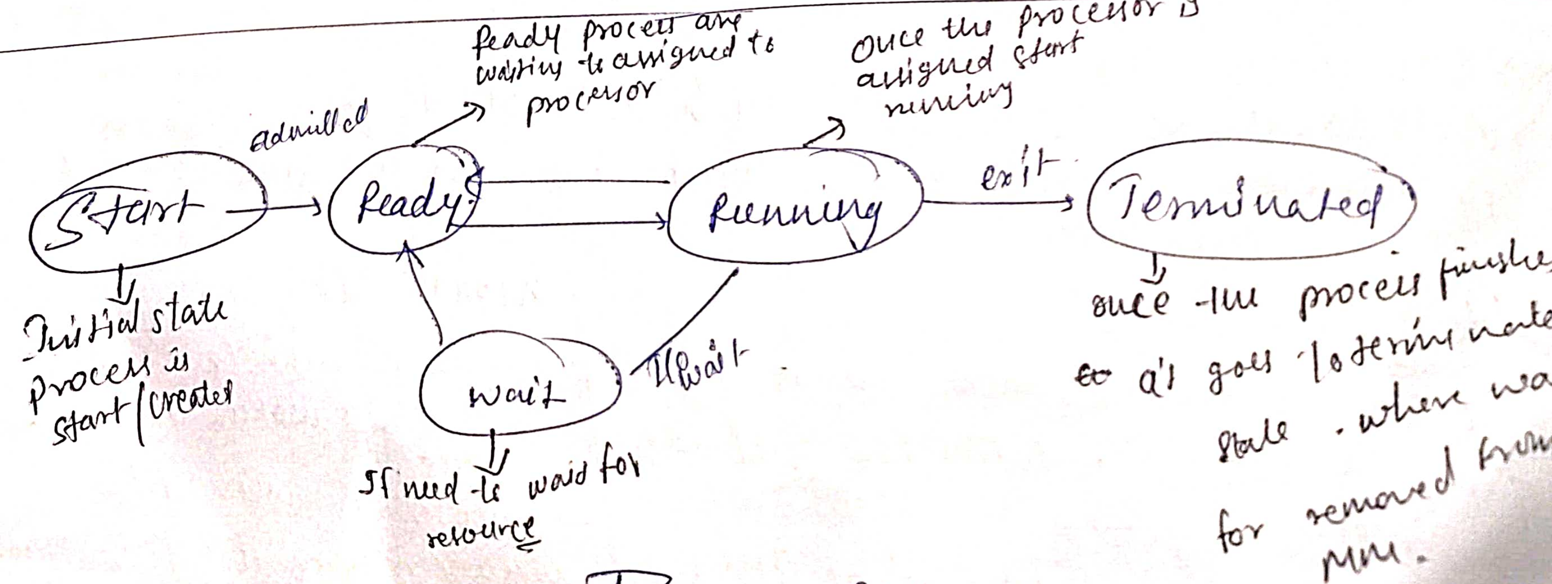
Program in execution
Program ready to execute.
i) Execution of process in sequential fashion
Easy language, computer program in text file, execute it become process and perform all the tasks in program.

→ Program is loaded in memory and it become a process.

4 type
- stack → temporary data functions, parameter, local variable.
- ↕
- Heap → Dynamically allocated memory during run time
- text → include current activity represented by value of PC
- data → Global & static variable.

→ Program is a piece of code can be single line or multiple line
→ Set of instructions designed to complete a specific task.

Process	Program
→ Dynamic entity	→ Static entity
→ Instance of an executing program	→ Set of instruction designed to complete a specific task
→ active entity, created during execution and load in MM.	→ passive entity resides in secondary memory.



Process life cycle

[PCB → Process control Block:]

→ Data structure, maintained by OS for every proce[ss]

→ Identified by integer Process ID.

→ Task control Block.

→ Each process is represented in OS by PCB.

→ Single program can have multiple process & each is identified by PCB

→ contains various info such as / –

unique indentification - of each process ←	Process ID
current state, ready, running, waiting ←	State
To parent process ←	Pointer
	Priority
Pointer to Address the next instruction to be executed ←	PC
	CPU registers
I/O devices alloted to process ←	I/O info
Amount of CPU used for process execution. ←	Accounting info
	etc.

[Kernel]

→ Central component of OS.

→ manages operation of computer and hardware

→ Basically manage operation of memory & CPU time

→ Core component.

→ Acts as bridge b/w applications & data processing

→ performed at hardware level using inter process communication & system calls.

Micro kernel

→ kernel type that implement an OS by providing methods, including low level address space management

→ Provides minimal services of process & memory management.

Monolithic Kernel

→ is an OS architecture where entire OS is working in kernel space.

→ Basic OS in which file, memory, device, process management directly controlled in kernel itself.

Process Scheduling →

- Activity of process manager.
- Handles removal of running process from CPU and selection of another process on basis of particular stratergy.
- essential part of multiprogramming OS

[Schedulers]

→ Special ~~system~~ software Handles process scheduling
→ Main task select job submitted into the system
→ Decide which process to run.

① Long term → Job scheduler,

→ select process from queue and load them into memory for execution
→ Provide balanced mix of job
→ Controls the degree of multiprogramming — stable, avg rate of process creation = avg. departure rate of process leaving the system
→ Time sharing OS → no long term schedule
→ changes the state from new to ready.

② Short Term :—

→ CPU scheduler, dispatcher.
→ faster than long term.
→ Increase the system performance.
→ Decision which process to execute next
→ ready to running state

③ Medium Term :—

also called context switching.

→ Part of swapping
→ reduce degree of multiprogram.
→ removes process from memory
→ charge of handling swapped out processes
→ Process suspended if it make I/O request.

Scheduling Algorithms

→ **FCFS** → → Simplest of all OS scheduling algorithm
→ Process request CPU first alloted to the CPU first
→ Implemented by using FIFO queue.
→ Supports non-preemptive & pre-emptive CPU scheduling
→ Easy to implement & use
→ Task are, executed as FCFS
→ Waiting time is high
→ Not efficient in performance.

→ **SJF** → → process having smallest execution time
→ may or may not be pre-emptive.
→ Best approach to reduce WT for other processes
→ Easy to implement in Batch system, CPU time is known
→ Suffers starvation problem.
→ Better ~~than~~ than FCFS
→ used for long term scheduling

→ **Priority** → It can be pre-emptive & non preemptive both
→ work based on priority of the process
→ The high priority or most Imp process executed first
→ If both have same priority they work on basis of FCFS.

→ **SRTF** → → Pre-emptive version of SJF.
→ Faster than SJF.
→ process with smallest amount of time remaining until completion is select to execute.
→ Contex switching is done alot.

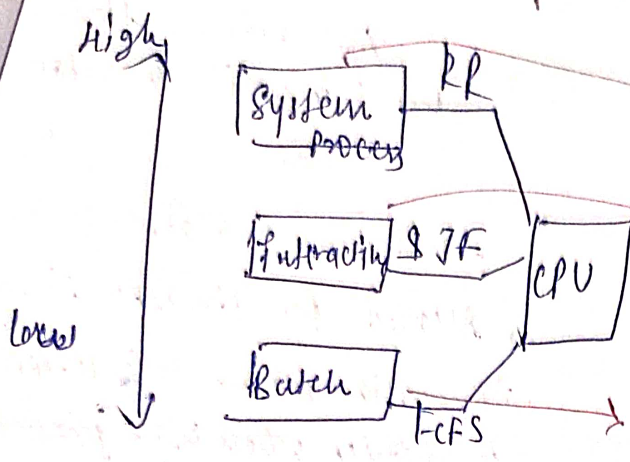
→ **RR** → → One of the most popular Sched. algo in OS.
→ pre-emptive version of FCFS
→ focus on time sharing technique
→ every process gets executed in cyclic way.
quantum time is alloted
each process as present in queue for that quantum time.

Multilevel ... process in ... class ... It can ... high

Multilevel queue! –

Process in ready queue divided into different classes
Each class has its own scheduling algorithm.

It can be divided into foreground process (interactive), Background process (Batch)



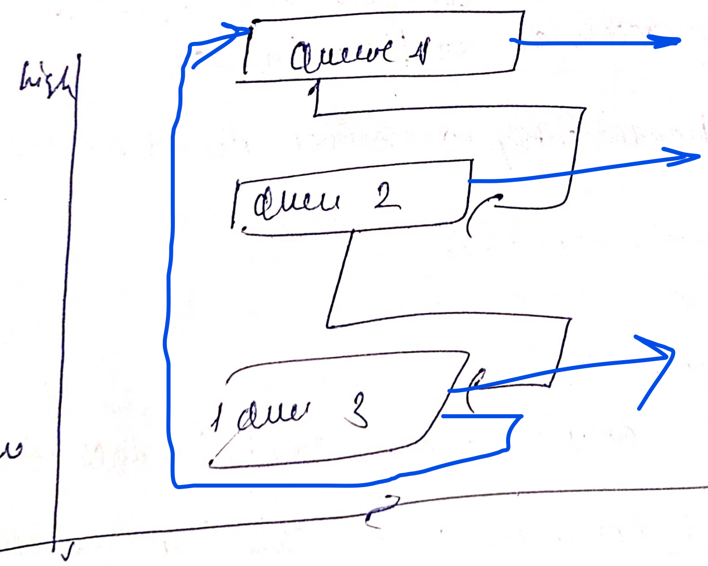
CPU Has its own process

It should have same type of interaction.

Collects data and info in the form of batch before processing.

→ MFQS →

→ Same as multilevel queue.
→ But in this process can move b/w Queue



1) Dining philosopher
2) sleeping barber
3) Bounded Buffer (Process consumer)
4) Readers writes

→ Semaphore →

→ Its a signalling mechanism
→ A thread is waiting on a semaphore can be signalled by another thread
→ Different than mutex, mutex can be signalled only be one thread, wait function

→ Two atomic operation, wait and signal for process synchronization

Non-pre-emptive

→ Once the cpu allocated to process hold it till it reaches waiting state or terminated

→ Running to ready state

→ CPU is alloted till it terminate or waiting state

→ can't be interrupted in middle

⇛ .

→ Put in ready
→ Process with the time starts
→ forcefully takes out from running state keep it in wait & allot to another process

→ running to waiting

→ Alloted for limited time

→ Execution process is interrupted in middle when high priority come

→ Critical Section :—

→ Is a code segment.

→ where shared variable can be access

→ Atomic action required.

→ more than one process access the same code segment.

→ Only one process can access at a time in CS.

```
do {
    entry section → handle entry → resources needed for execute
    critical section .
    exit section → release the resource
    Remainder section,
} while (TRUE)
```

Solution to CS must satisfy

① Mutual exclusion — only one process can be inside CS at any time

② Progress → Process is not using CS then can't stop another process

③ Bounded waitings → Each process must wait for limited time

Deadlock

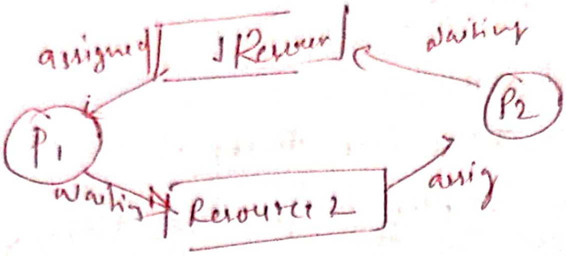
Deadlock is a situation where set of processes are blocked. Each process is holding a resources & waiting for another resource acquired by some other process.

→ further Processing is not possible.

→ Occurs because we want to execute multiple process concurrently.

→

Ex - two trains coming towards each other on the same track and there only one track



Necessary condition :- (4 conditions)

1. Mutual exclusion :- Two or more resource are non-sharable.
2. Hold and wait → A process is holding resource & waiting for resource.
3. No-pre-emption → A resource can't be taken from process ~~until the~~ unless it release the resource.
4. Circular wait → set of process waiting for each other in circular form.

Methods for handling deadlock :- Three ways.

① Deadlock prevention or avoidance :-

→ The idea is to not let the system into deadlock state.

→ This system will make sure the 4 condition does arise.

→ This technique are costly.

→ For this it has 4 different ways.

1. Eliminate mutual exclusion
2. Solve Hold & wait
3. Allow - pre-emption
4. Solve circular wait.

Avoidance :-

→ It is futuristic
→ using the strategy 'Avoidance' we have to make attempts
→ All the info or resource process need is known to us by execu..
→ we use Banker's algorithm for this.

② Deadlock Detection & recovery :-

→ It's done it two phase
→ In first phase, we examine states of process if deadlock is in system or not
→ If found then we apply algorithm for recovery.

③ Deadlock ignorance :-

→ very rare
→ let it happen and reboot the system
→ Ostrich algorithm is used.

→ Create need matrix [Required − Allocation]
[Max]

→ Need matrix = Request also
→ check Available ≥ need, execute.
→ Add allocate to available.

Race condition :-

→ Occur inside critical section
→ Happens when multiple thread executing in critical section differ. order of thread execution.
→ Can be avoid when CS is treated as an atomic instruction
→ A proper thread syncronisation using locks., atomic variable prevent Race condition.

[Banker's Algorithm]

Banker's algorithm is a resource allocation and deadlock avoidance algorithm.

Banker's algorithm is named so because it is used in banking system to check whether loan can be sanctioned to a person or not.

Also known

→ deadlock avoidance or deadlock detection in OS.

→ n number → account holder

Total sum of their money → S

→ Person applies loan → subtracts loan money → Total money

If

→ remaining money ≥ S then only loan is sanctioned.

→ Bank would try to be in safe state always

[RAG]

Resource Allocation Graph :-

→ Deadlock can be described more precisely in terms of directed graph

→ The set of vertices consists node for P = {P1 P2 P3 - - } consist of all activity process in the system.

→ and R = {R1, R2, R3 - - - } set of consisting all the resources.

→ How many resource are available

How many are allocated, what is the request of each process can be represented in terms of diagram.

RAG contains vertices & edges

1. Process vertex → Every process will be represented as.
2. resource vertex → Every resource is represented

↓

Single instance → Represent as box inside box only one dot.

multiple → many dots.

1. Assign edge → If Resource Ri is allocated to Pi

[R] → (P)

2. Request edge → If Pi request and instance R1

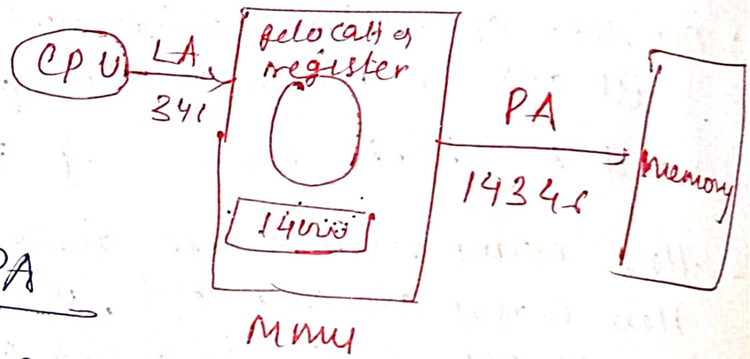
(P1) → [R]

Logical Address →

→ generated by CPU, which program is running
Time → execution of program
→ Virtual address → doesn't exist physically.
→ used as a refrence to access physical memory location by

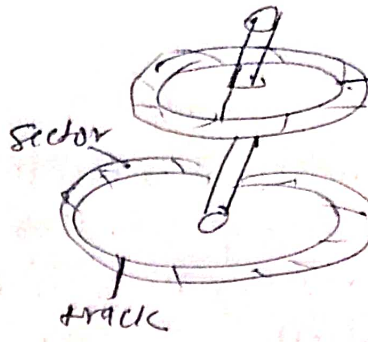
Physical Address :-

→ identifies physical location of required data in the memory
→ user ~~never~~ never directly deal with PA but access by its LA
→ user ~~generated~~ program generates LA and thinks program is running in LA
→ Program need physical memory for execution
→ LA mapped to PA by MMU



LA	PA
Generated by CPU	Computed by MMU
Access → user can use LA to access PA	→ can't access directly
Visibility → user can view LA of a program	Can never view
editable → can be change	~~Can't~~ will not change
Virtual Address	read address

[Disk scheduling]



Platter
↓
surface
↓
sector
↓
track

Done by OS
I/O scheduling
I/O request arrive on disk ← schedules
Goal is to minimize $\underbrace{\text{seek time}}_{\text{time taken to reach desired disk}}$

→ multiple I/O request can] arrive but only 1 serve

→ Two or more request far from each other result in greater disk arm movement

→ FCFS → → Simplest of all DSA.
→ Requests are addressed in ordered they arrive in disk queue.

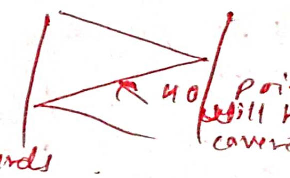
Advantage	Disadvantage
① Every request gets fair chance.	① Not provide best possible service
② No indefinite postponement	② Does not try to optimize seek time

→ SSTF → Track closer to current disk head position served first.
→ Seek time is calculated in advance & they are scheduled according to that.

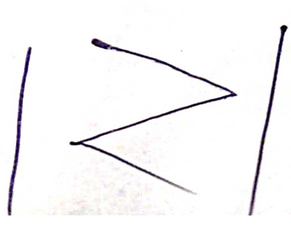
→ SCAN → → Disk arm moves in particular direction and serve requests in that path.
→ After end of disk it reverse its direction and serve in that direction
→ works as an elevator   pre → current
→ elevator algorithm

→ CSCAN :- → Disk moves in a particular direction serve request until it reaches last. ~~to~~ cylinder
→ Then it jumps to last of the opposite direction without serving any request and then start from there

→ LOOK → Look similar as scan to some extent but it doesn't do the extra movement outwards.



→ CLOOK → same as CSCAN but don't go for outwards.



[Memory allocation]

Contiguous MA

→ when a process is request the memory . a single contiguous section of memory block is allotted depending on the requirement.

→ Process can't be divided & placed in different location's due to which external fragmentation proble arrives.

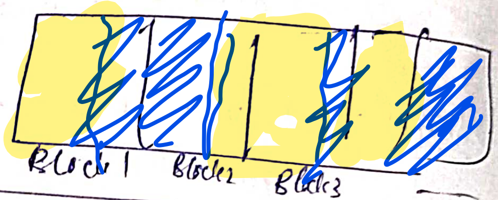
Non-Contiguous

→ To remove solve external fragmentation.

① **First Fit** ! → Very simple & fast
→ Allocate the first block which is big enough.

② **Best Fit** → slow
→ It will search entire list and search the block which lead to minimum internal fragmentation

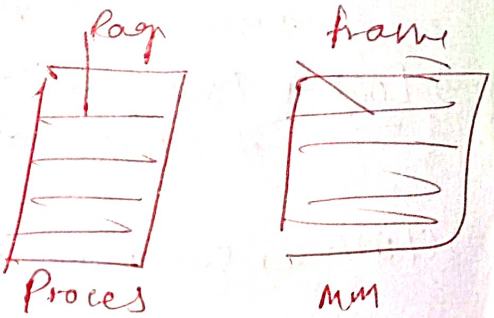
③ **Worst fit** → opposite of best fit.



[Paging]

→ Memory management scheme.
→ Eliminate the need for contiguous allocation of physical memory
→ Process of retriving processes in form of page from secondary to main memory.

→ In paging we divide process into pages.
→ size of page = size of Fram of MM
→ Divide MM into frames
→ so that page can easily fit in frame



The mapping is getting done by MMU.
MMU converts the address generated by CPU into Absolute address.
Use page table.

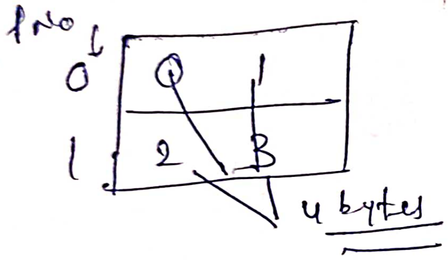
→ $\boxed{\text{Page table}}$ → Contains frame no., which tells where the page exactly present in the MM.

→ Every process has its own page table.

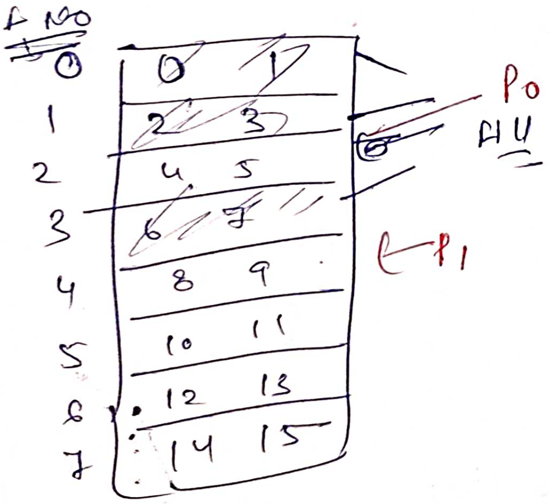
→ LA | PNO | Page offset |

Eg → assume process size 4B and page size 2B.

→ No. of page = ~~Page~~ $\frac{\text{Process size}}{\text{Page size}} = \frac{4}{2} = 2$



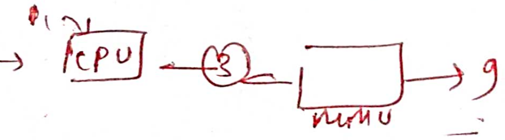
→ RAM = 16 B → frame size = 2 B No of frame = $\frac{16}{2} = 8$



Suppose 0, 1, 3 are fill then Po goes to 2 frame and P1 goes to F4

0 → 4
1 → 5
2 → 8
$\boxed{3 \to 9}$

want something that'll convert 3 to 9



Page
0	F2
1	F4

(Cache) TLB (Translation lookaside Buff

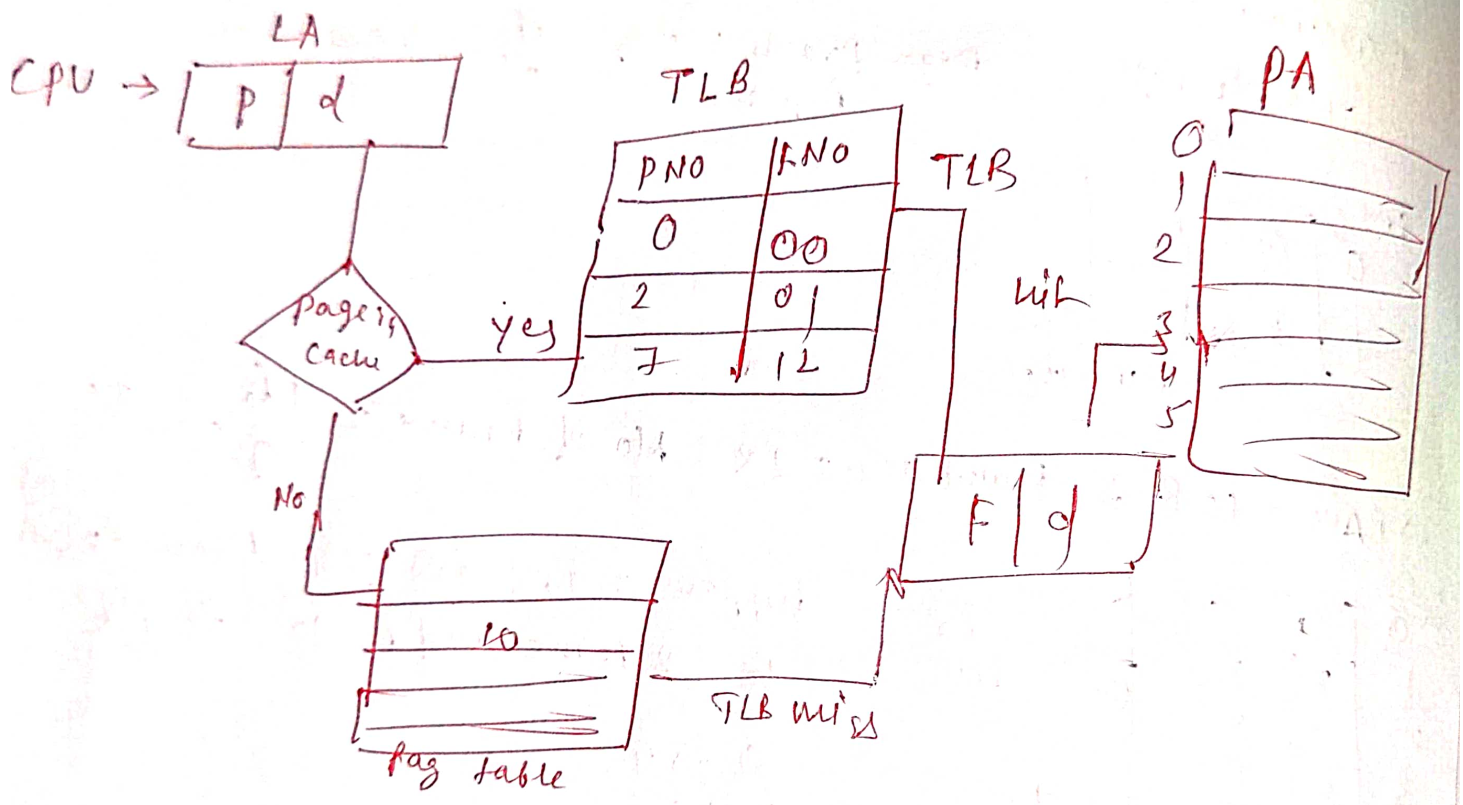
→ Suppose the time to access RAM is 7, and then there page table inside memory need to access so it become 2x

→ So cache memory is faster than RAM.

→ To overcome the problem, high speed cache memory is set for page table which is called TLB.

→ TLB is nothing but a special cache used to keep track of recently used transactions

→ If the Page table is present in TLB its TLB hit otherwise miss.



TLB hit → ~~Check~~ CPU generat LA

→ check TLB if present

→ FNo, tell in MM where page lies

TLB miss → CPU generate LA

→ Check TLB (not) present

→ Now check the Page table.

~~The mapping is getting~~

Page replacement Algorithm

→ FIFO → the frame which is filled first will be replaced first.

Page fault → CPU is searching for page and its not present into the MM so its page fault.

Belady's ~~Algorith~~ Anomaly :– as we know if no. of frame increase than page fault decreased

→ But According to Belady's Anomaly in FIFO is no. of frame increased then page fault also increase sometimes

→ Optimal → replace page which is not used in longest dimension of time in future

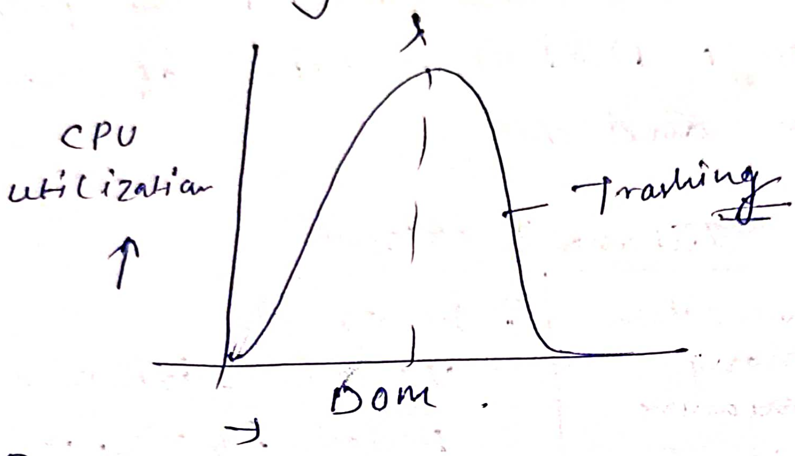
check →

→ LRU → Replace page which is least recently used in past

check ←

Thrashing :-

There comes so many page faults at one point and page hit is less. So all the time of CPU went on taking the service of page from the hard disk to main memory.



→ High DOM

→ Segmentation →

→ Program are divided into parts or segments and then ~~where~~ we put them into the MM.

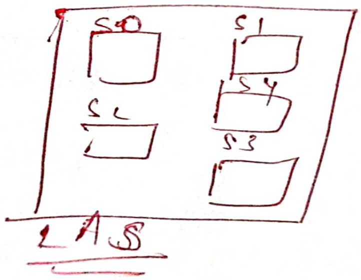
→ In this chunks the programs are divided not necessarily all of them of same size.

→ Its gives the user point of view to process

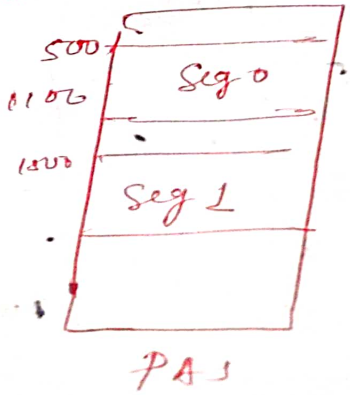
→ The table stores the info of all the segment are segment table.

→ Base address — Starting PA where segment resides

limit → length of segment



	BA	limit
0	500	600
1	1500	400
2	2500	300
3	3000	200



Virtual memory

→ Provides illusion to the programmer that a pr whose size is larger than the size of MM can be execute.

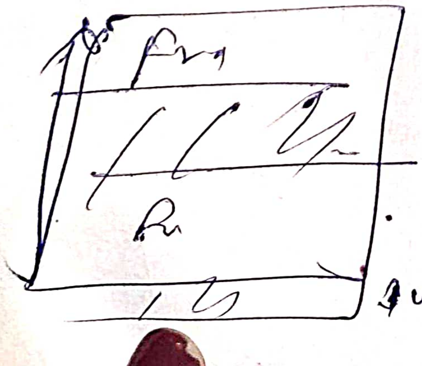
→ Storage allocation schem

→ In this instead of taking the whole process in MM we take page that are required of that process

→ and do swip In and out

Internal & external

Internal	External
① In Internal fragmentation fixed sized memory blocked square measure appointed to process	① In ex variable size memory, blocked, square, measure appointed to method
② Happns when method and process is smaller than memory	② Method or process is removed
③ Solution Best-fit block	③ Paging
④ Occurs memory is divided into fixed size partion	④ Variable size partion
⑤ Occurs in worst fit MA	Best & first fit



user