# Introduction

Hardware

Mechanical
Electrical
Electronic
} Parts

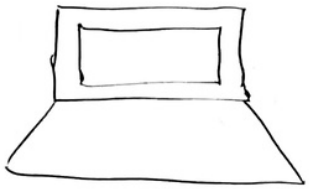
Software

set of programs

- Application (Spreadsheet, Db)
- System (OS)

User

Interface
Intermediatory



System H/W

↓
Eg ↓

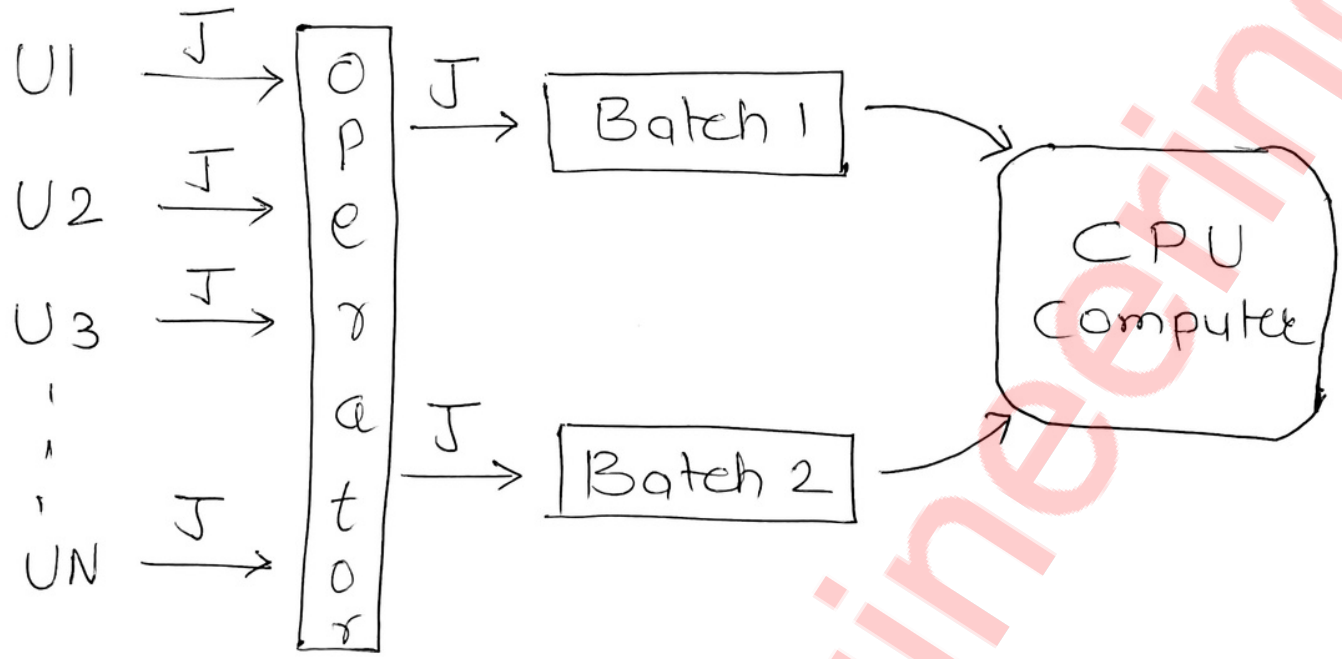
Mac, Windows, Android, Pos, Linux

- OS Goals
  - Primary : Convenience / Ease feel (Easy to use)
  - Secondary : Efficiency, Reliable Management

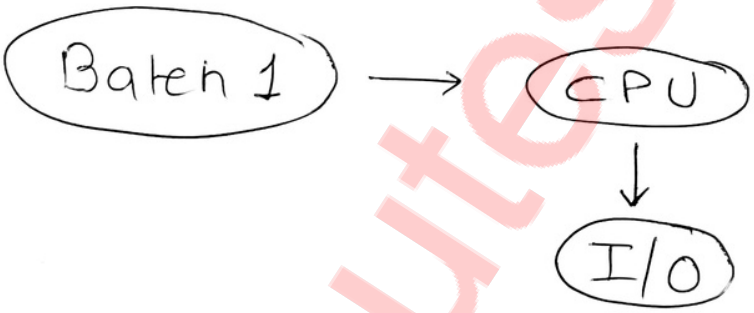
- OS functions
  1. Booting: start the computer & make it ready to work
  2. Process Mgmt
  3. Memory Mgmt
  4. File Mgmt
  5. I/O Device Mgmt
  6. Security

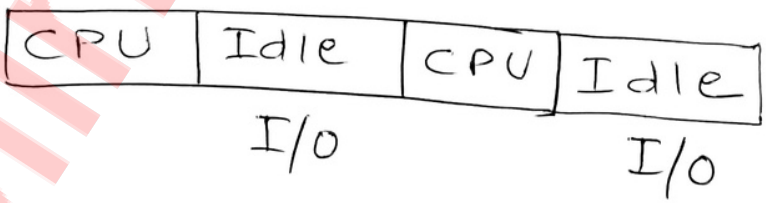
- Types of OS:
  - → Batch
  - → Multiprogrammed
  - → Multitasking
  - → Clustered (1 aur 1 = 11)
  - → Distributed (Environment / schema)
    - ↳ Not centered at point
  - → Embedded (Specific / fixed / narrow system functionalities)
    - ↳ AC, washing m/c.

# Batch OS







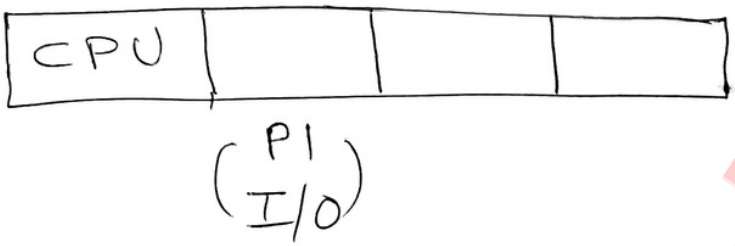



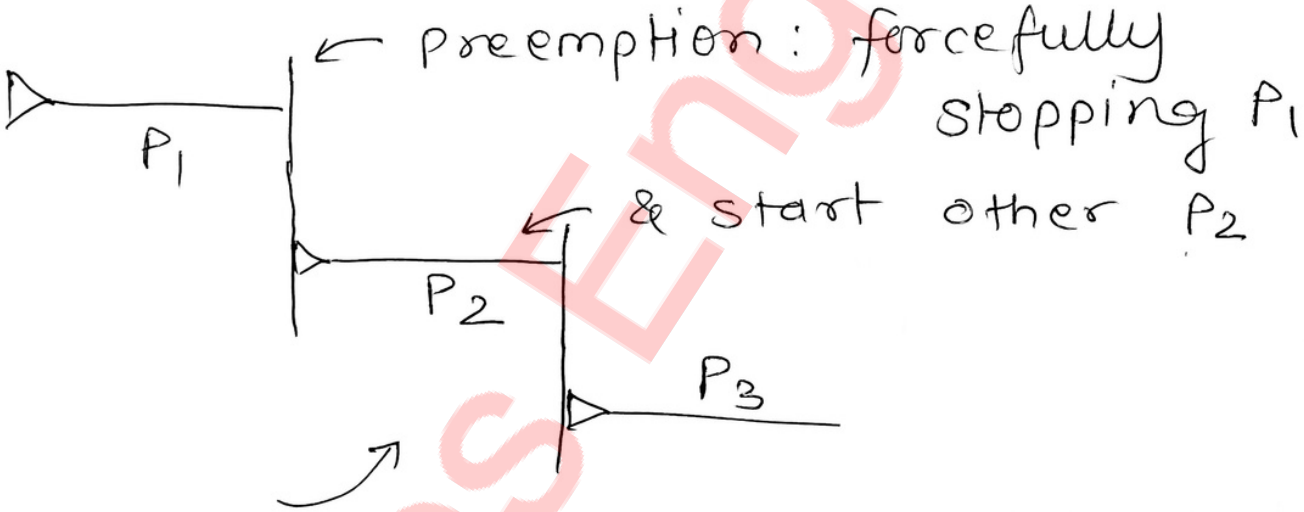

CPU Utilization ki Band Baji Padi hai

- Multiprogramming OS

→ No preemption [focus on more than 1 process]

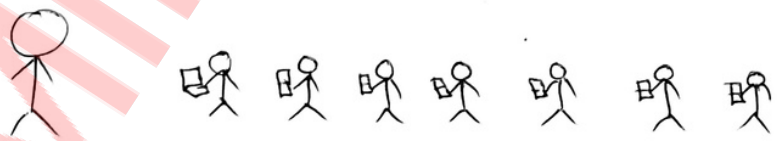
(P1) (P2) (P3)




- Multitasking (Preemption allowed)

Eg: Selfiee

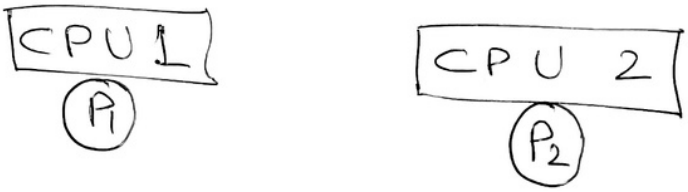


(10 selfiee each)

MP: Take 10 selfiee with each (P).

MT: Take 5 selfiee then next.

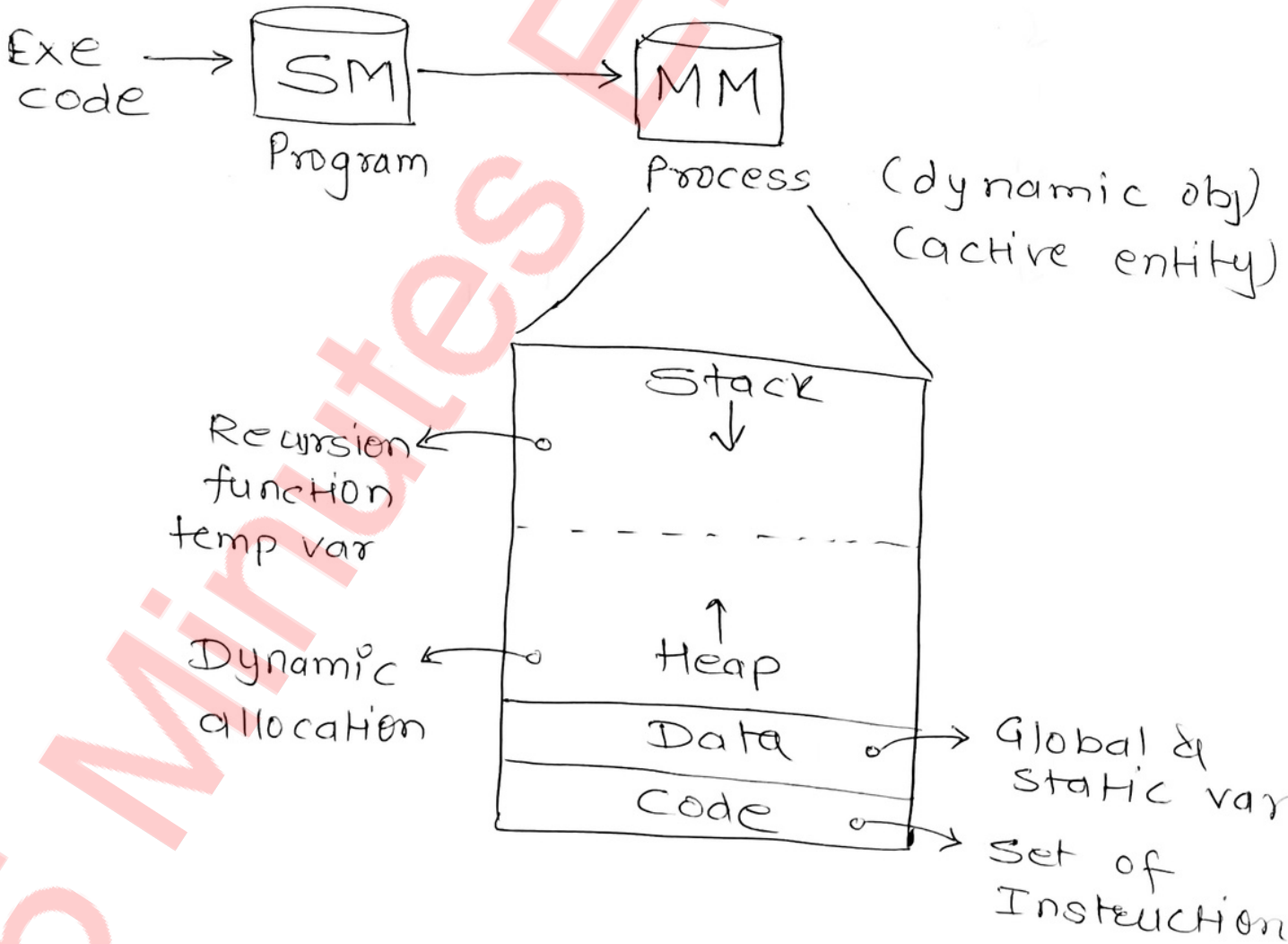
- Multiprocessing OS



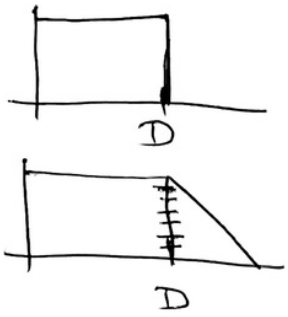

(If we need many Processors ← NO extra 'M' 'I/O')

→ Parallelism Increased
→ Efficiency Improved
→ faster processing
→ Reliability

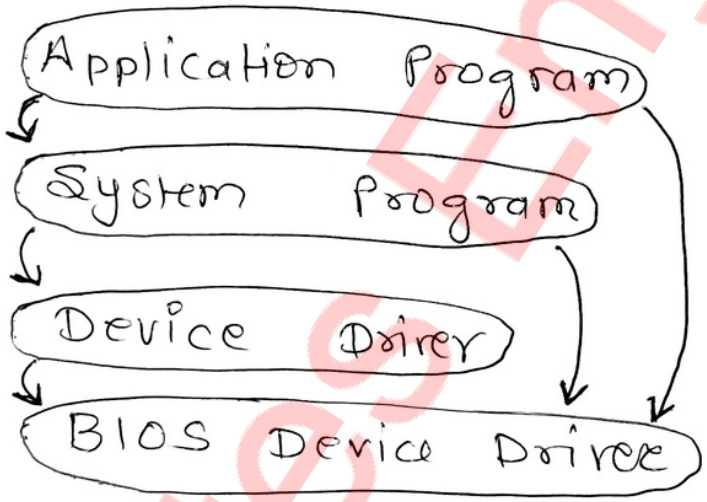
- Process Vs Program



- Real Time OS [Time limit] [Delay x]
  - → Hard → steict
  - → soft → less steict




- Structure of OS [Components of OS are organized & interconnected]
  - → simple

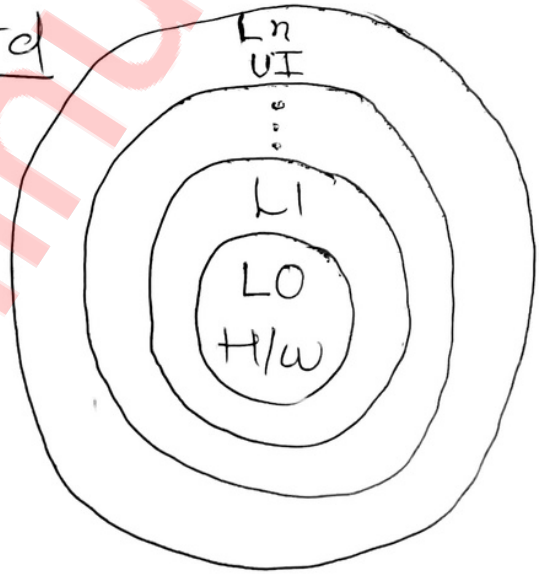



Easy

But

OS crashes (as user prg fails)

→ Layerd



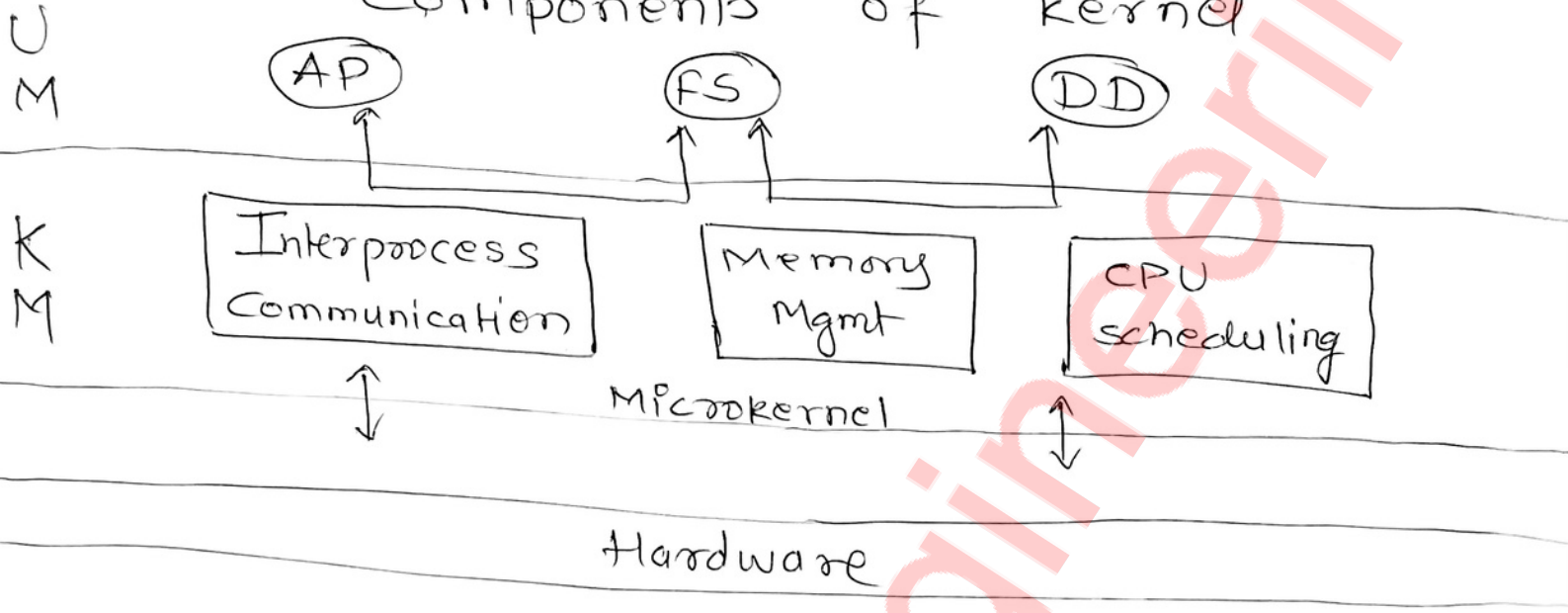

→ Each Layer has a specific task

→ Hierarchy: Layer can use the services provided by lower L

→ Independent

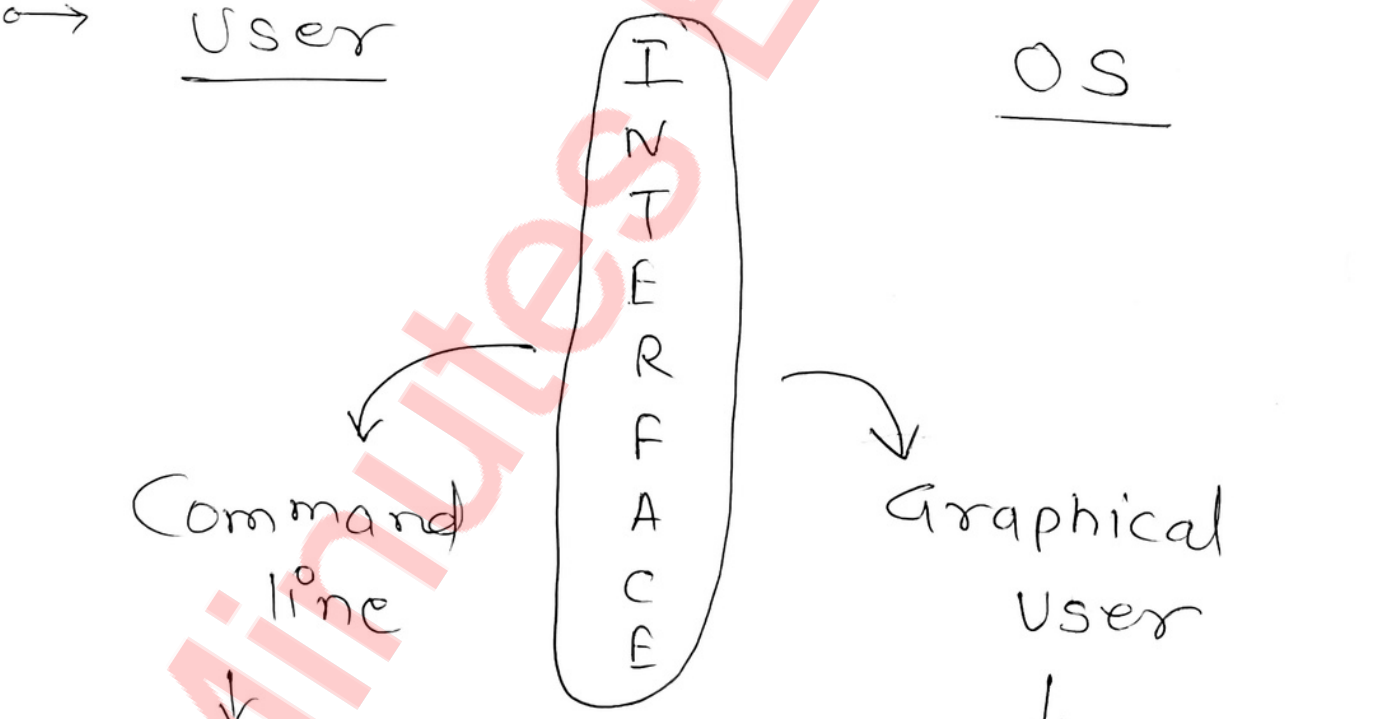
- **Micro kernel OS**

  ↳ Removing all non-essential components of kernel

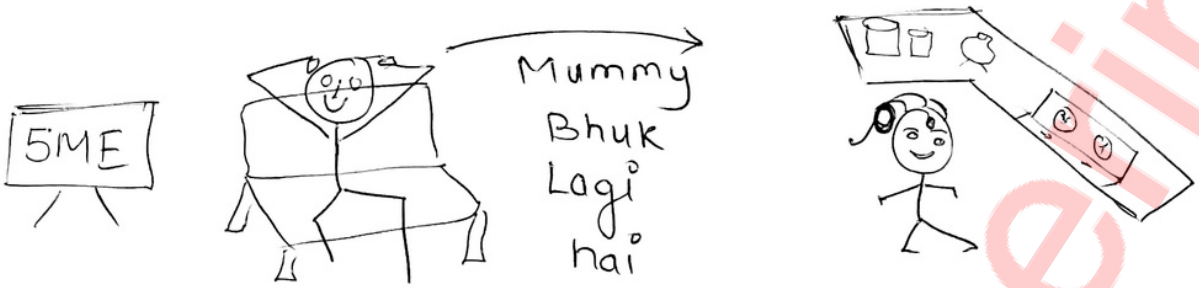



→ Complex. [kernel + Interface = OS]

→ **User** **OS**



# • System Call




User Program → SC → OS → [H/W

→ Process control
{end, abort, load execute, create process, terminate process, wait for time, "fork", wait event, allocate memory & free memory}

→ ~~File System~~
{ Create, open, close, delete, read, write, set & get file attrs }

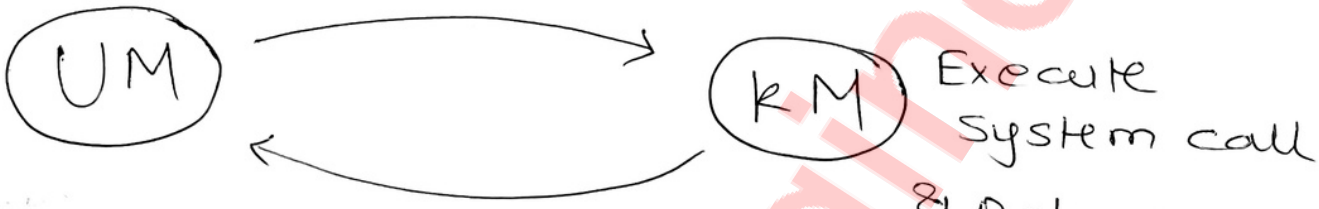
→ Device Mgemt
{ Read, write, request, release }
Reposition

→ Information { get system time & data }
{ get PId, get attr }

→ Communication { Creat, delete connection, pipe }
{ Send, receive }

Mode → User (m = 1)
(Applications operate here)
(Slave / Unprivileged mode)

→ Kernel (m = 0)
(Core OS functions)
(Master / privileged / system mode)




System call to execute from kernel

Context Switching

Printf()

write()


FIR

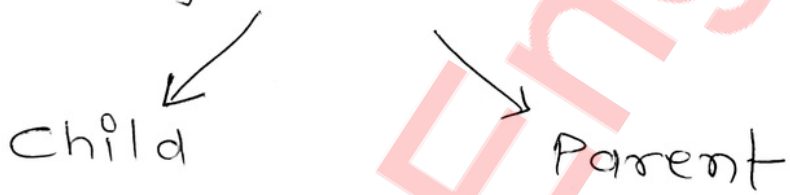
° fork System call

→ 0 : child process

→ 1 : Parent process

```
P()
{
  fork()
}
```

1 fork()

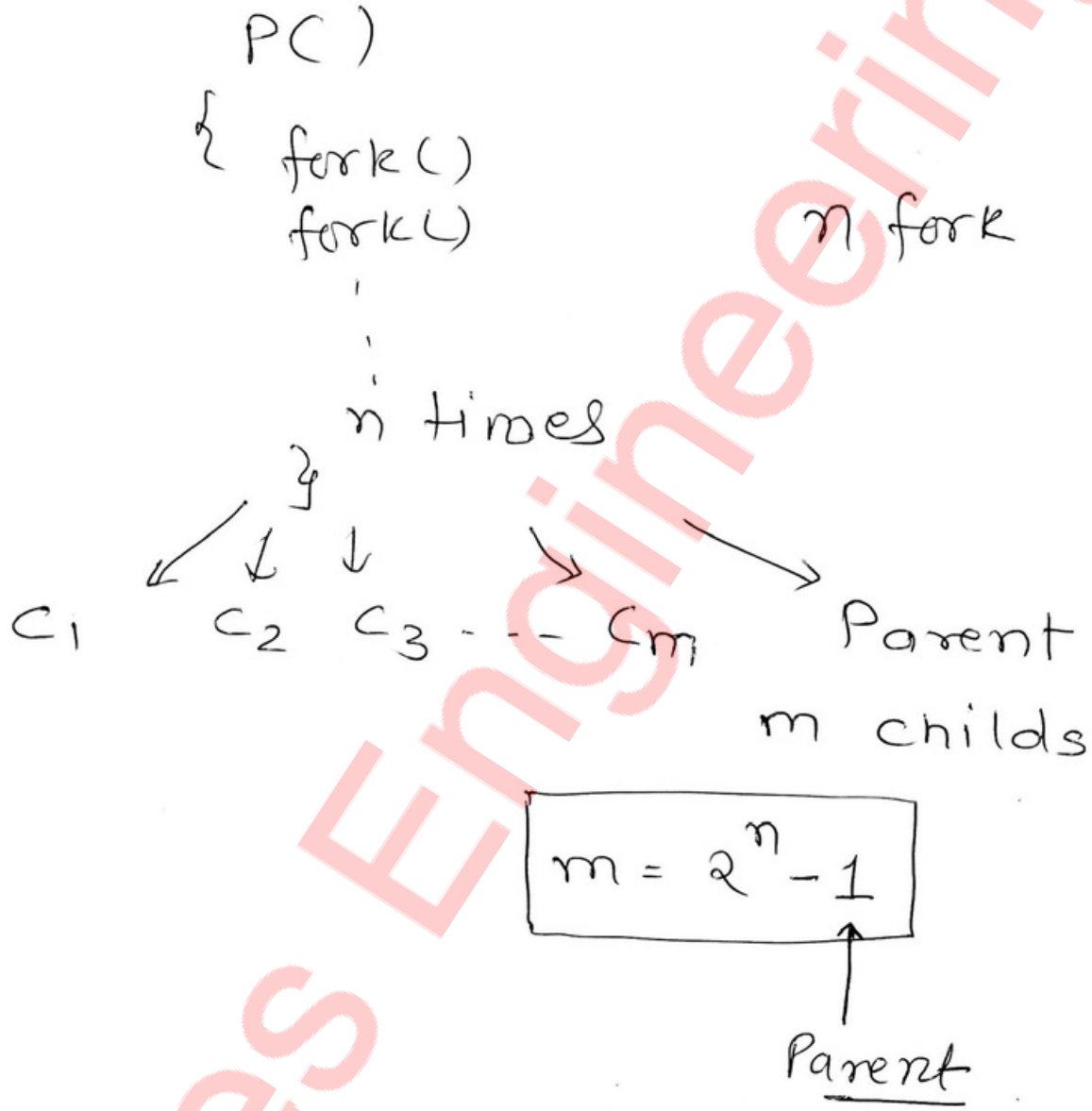



1 child

```
P()
{
  fork()
  fork()
}
```

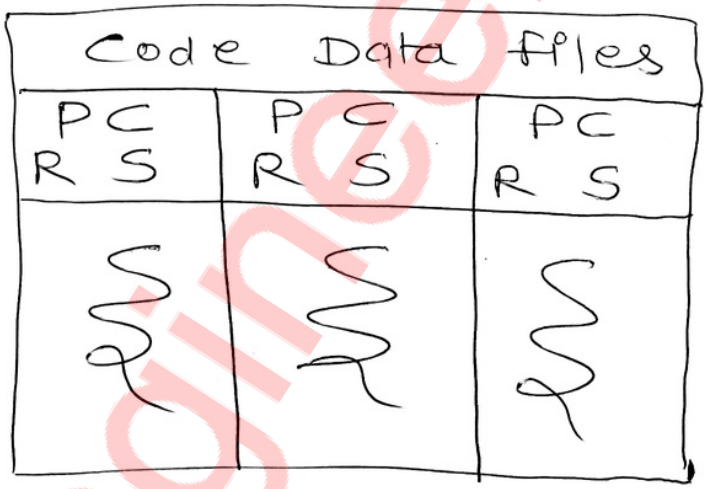
2 fork()


3 child



# Threads

- Single
- Multi


↑ Single Process (Multiple tabs in a Browser)

**Need** :- 1) Creating new thread in current process requires less time & effort than creating a new process.

2) Threads can share common data
3) Context switching is quick.
4) Can be terminated in less time.

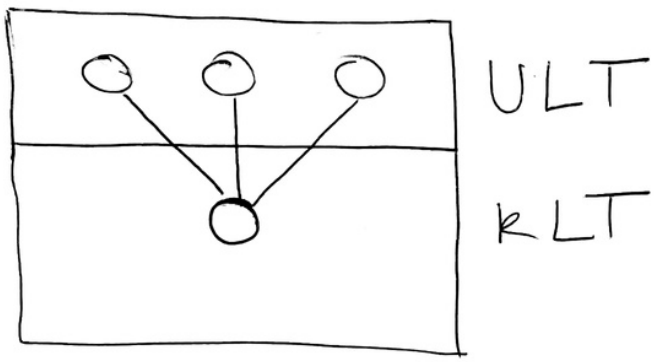
- User level thread
  - ↳ As the name suggest
  - → threads at user level
    i.e these threads are managed by user/appln
  - → faster (CS)
  - → ULT gets blocked entirely (Process)
- kernel level thread
  - ↳ Threads are managed by Operating system.
  - → Slower (CS)

CS

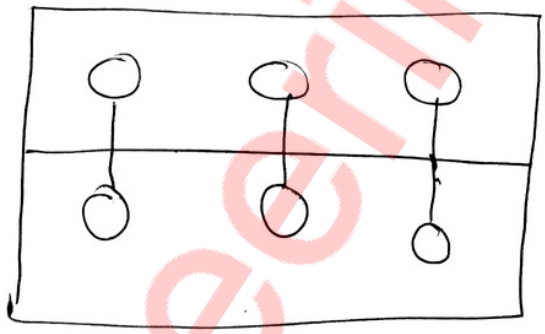
ULT < KLT << Process

→ NO effect of one KLT to other.

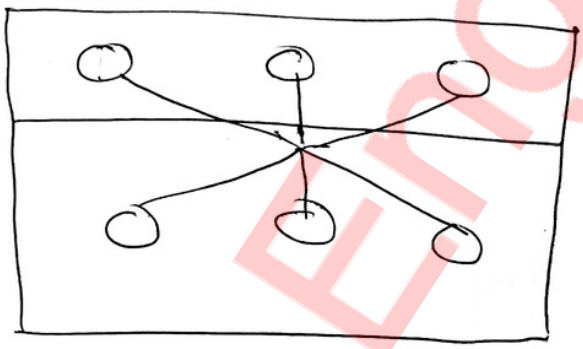
Many to one



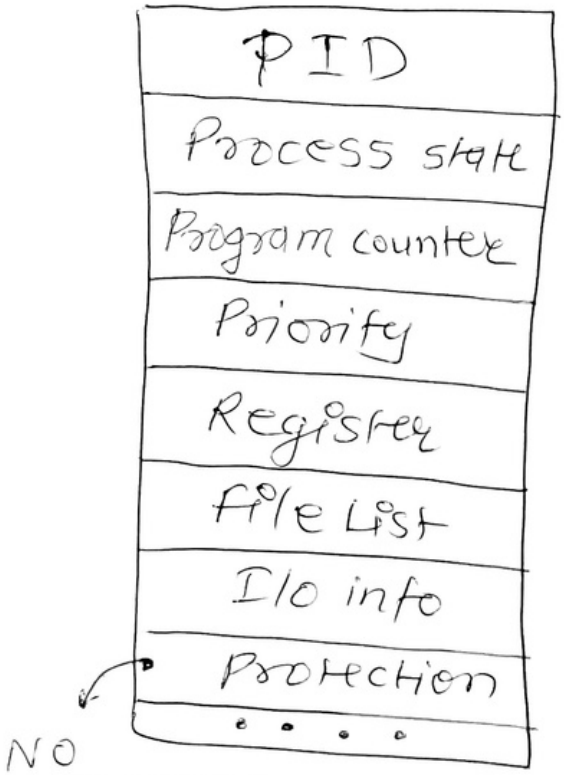

One to One



Many to Many

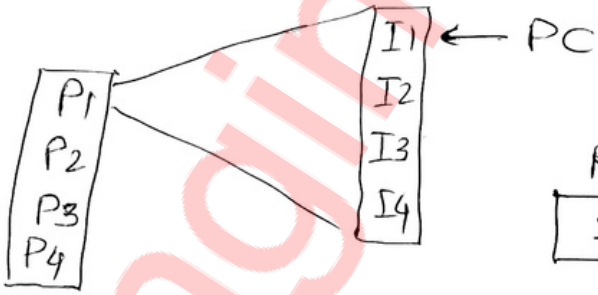


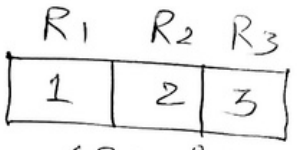
# Process Control Block :




Each process has
↓ it contains all
Info regarding
a process

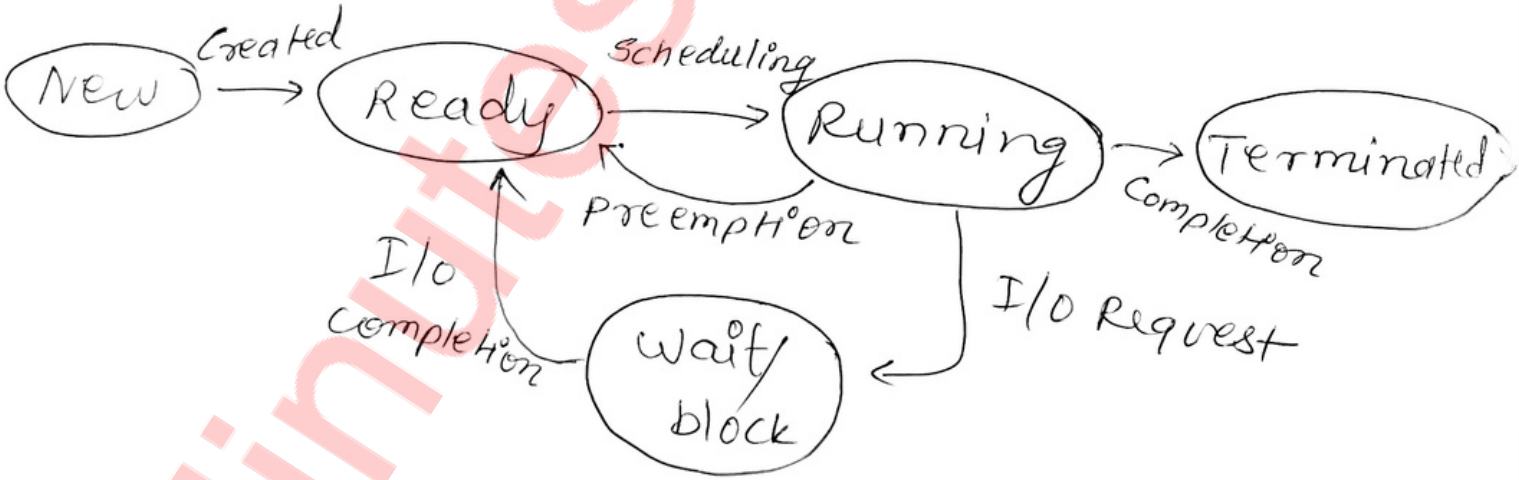
NO process Interfer with other (P)




KP > UP
Priority

# Process States



# Schedulers

- Long Term
- Short Term
- Medium Term

"Job scheduler"

new → Ready

Ready → (dispatching) → Run

LT →

Process Time → CPU (P1) (P2) (P3) } starvation

Process Time → I/O (P1) (P2) (P3) } Idle

ST →

Ready → (scheduling / Dispatching) → Run

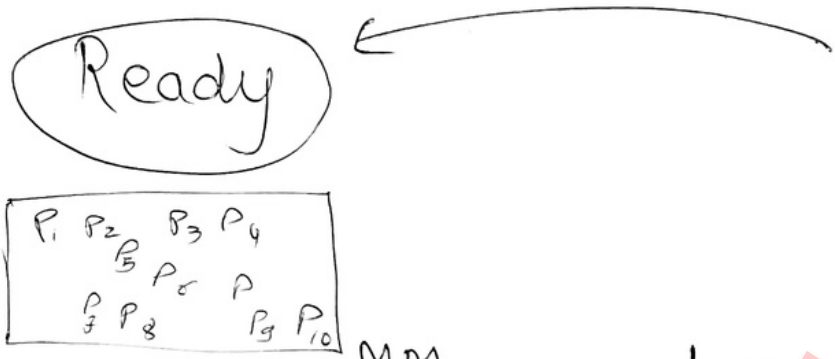
CPU scheduler (fast)

MT → Swapping scheduler.

wait/block (MM) → Suspend wait (SM)

Ready (MM) → Suspend Ready (SM)

# Degree of Multiprogramming




DOM = 10

LTS: Control DoM

STS: Lesser Control over DoM

MTS: It Reduces the DoM

- **CPU Scheduling** (which Ⓟ in Ready Q is allotted to CPU)
  - Preemptive
    - SRTF
    - LRTF
    - Round Robin
    - Priority
  - Non Preemptive
    - FCFS
    - SJF
    - LJF
    - HRRN

→ Shortest Remaining Time first
→ Longest Remaining Time first
→ First come first serve
→ Shortest job first
→ Longest job first
→ Highest Response Ratio next

- Time Quantum ['t' units for each Ⓟ]

① Arrival Time: when Ⓟ enter the Ready state
② Burst Time: Time required by Ⓟ to execute.
③ Waiting Time: [TAT − BT]
④ Turn around Time: [CT − AT]
⑤ Completion Time: when Ⓟ complete its execution
⑥ Response Time: [(Ⓟ get CPU 1st time) − AT]

- Non-Preemptive

## ① FCFS

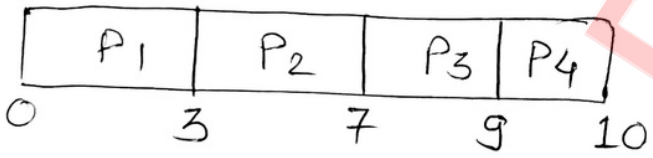
Eg:-

| Pno | AT | BT | CT | TAT | WT |
|---|---|---|---|---|---|
| P1 | 0 | 3 | 3 | 3 | 0 |
| P2 | 1 | 4 | 7 | 6 | 2 |
| P3 | 2 | 2 | 9 | 7 | 5 |
| P4 | 3 | 1 | 10 | 7 | 6 |

TAT = CT − AT

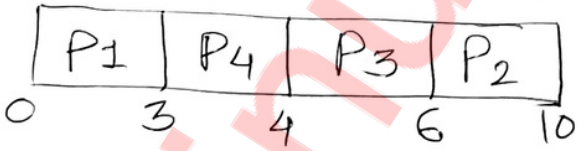
WT = TAT − BT

CT (Traverse Right to left)




Problem : Starvation

## ② SJF (Consider shortest BT first)

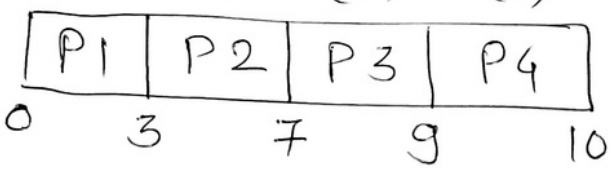



| Pno | CT | TAT | WT |
|---|---|---|---|
| P1 | 3 | 3 | 0 |
| P2 | 10 | 9 | 5 |
| P3 | 6 | 4 | 2 |
| P4 | 4 | 1 | 0 |

## ③ LJF (Consider Longest BT first)

| Pno | AT | BT | CT | TAT | WT |
|---|---|---|---|---|---|
| P1 | 0 | 3 | 3 | 3 | 0 |
| P2 | 1 | 4 | 7 | 6 | 2 |
| P3 | 2 | 2 | 9 | 7 | 5 |
| P4 | 3 | 1 | 10 | 7 | 6 |

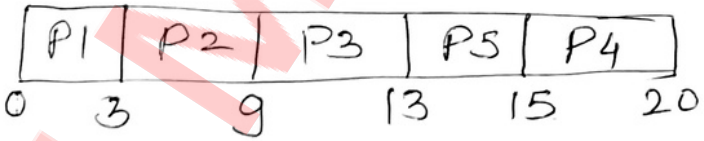
(4) (2) (1)




## ④ HRRN

$$\text{Response Ratio} = \frac{WT + BT}{BT}$$

| Pno | AT | BT |
|---|---|---|
| P1 | 0 | 3 |
| P2 | 2 | 6 |
| P3 | 4 | 4 |
| P4 | 6 | 5 |
| P5 | 8 | 2 |




At $t = 9$ (P2) completes

P3, P4, P5 is available

WT → P3 = (9 − 4) = 5
WT → P4 = (9 − 6) = 3
WT → P5 = (9 − 8) = 1

RR → P3 = $\frac{5+4}{4}$ = 2.25
RR → P4 = $\frac{3+5}{5}$ = 1.6
RR → P5 = $\frac{1+2}{2}$ = 1.5

(P3) selected ✓

At $t = 13$

WT → P4 = (13 − 6) = 7

WT → P5 = (13 − 8) = 5

RR → P4 = $\frac{7+5}{5}$ = 2.4

RR → P5 = $\frac{5+2}{2}$ = 3.5

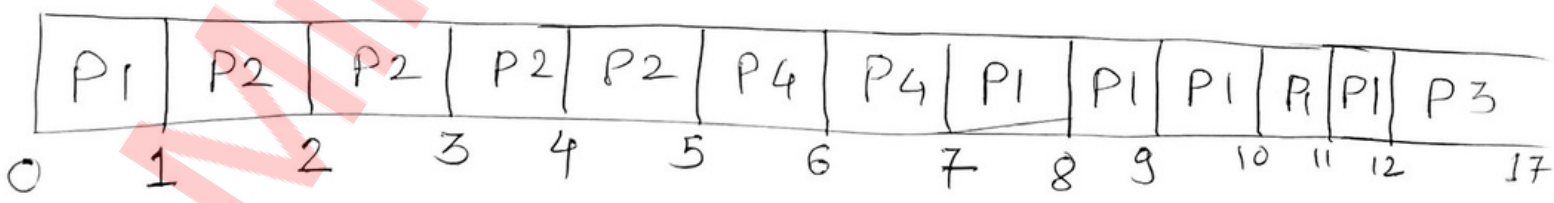
(P5) selected ✓

- Preemptive nature.

① SRTF

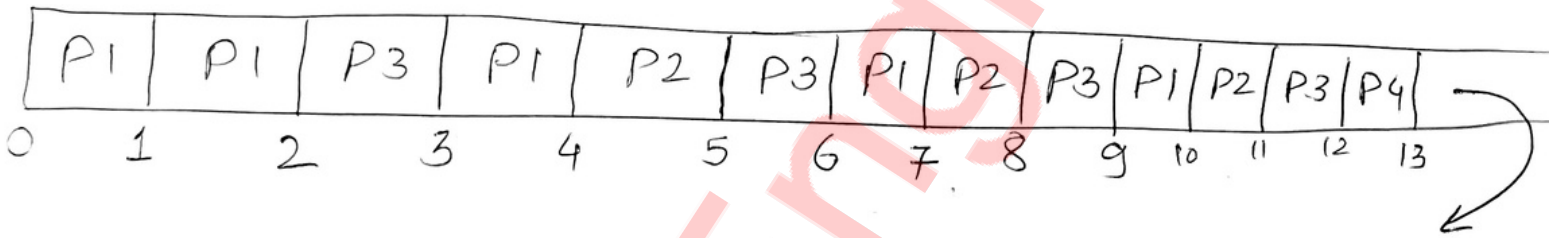
Eg

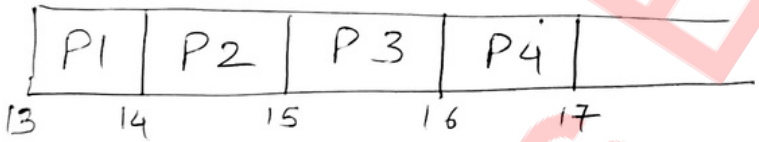
| Pno | AT | BT | CT | TAT | WT | RT |
|---|---|---|---|---|---|---|
| P1 | 0 | 6 | 12 | 12 | 6 | 0 |
| P2 | 1 | 4 | 5 | 4 | 0 | 0 |
| P3 | 2 | 5 | 17 | 15 | 10 | 10 |
| P4 | 4 | 2 | 7 | 3 | 1 | 1 |



## ② LRTF

| Pno | AT | BT | CT | TAT | WT | RT |
|---|---|---|---|---|---|---|
| P1 | 0 | 6 | 14 | 14 | 8 | 0 |
| P2 | 1 | 4 | 15 | 14 | 10 | 3 |
| P3 | 2 | 5 | 16 | 14 | 9 | 0 |
| P4 | 4 | 2 | 17 | 13 | 11 | 8 |

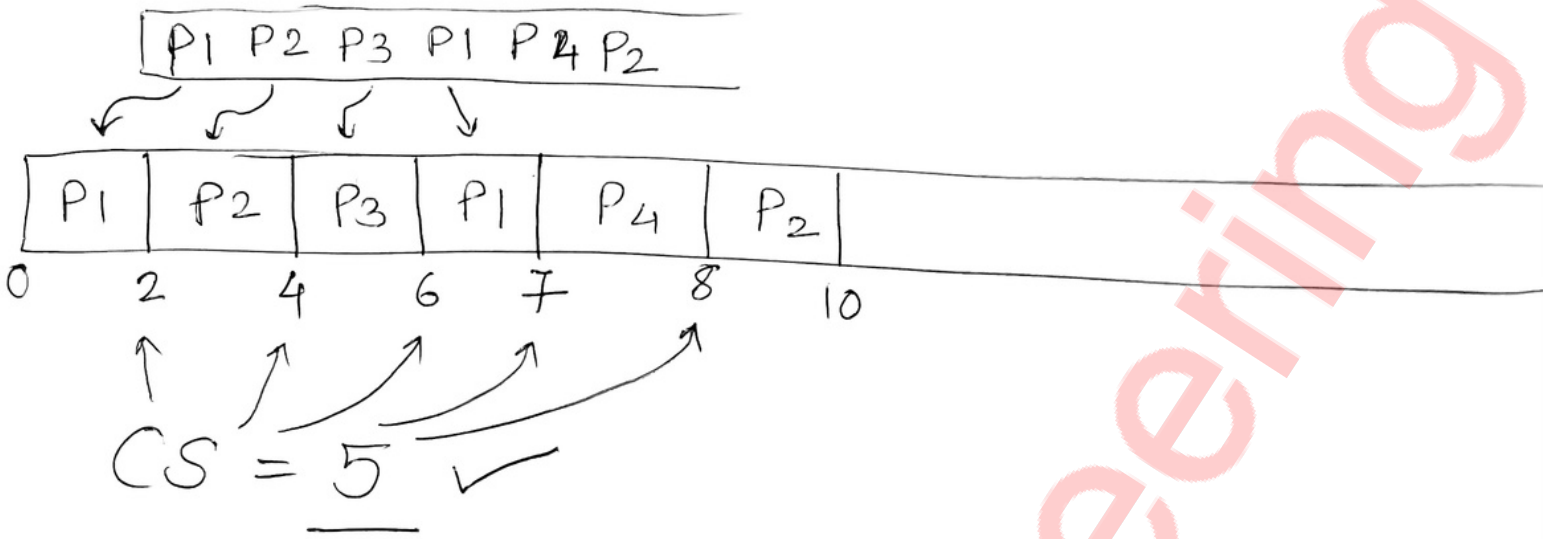



## ③ Round Robin (AT + TQ)

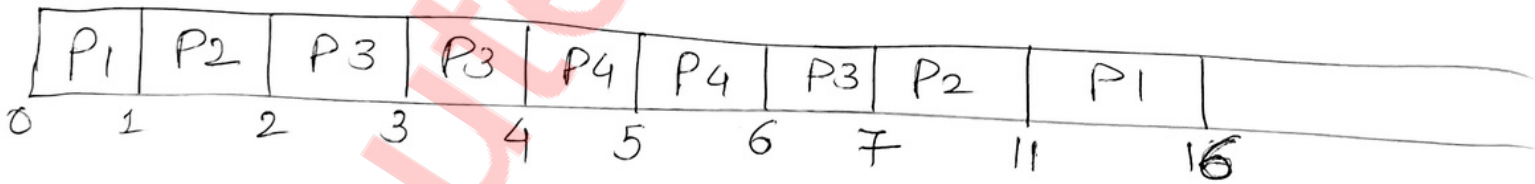
Min(BT, Q) ↳ Q = 2 units

| Pno | AT | BT | CT | TAT | WT |
|---|---|---|---|---|---|
| P1 | 0 | 3 | 7 | 7 | 4 |
| P2 | 1 | 4 | 10 | 9 | 5 |
| P3 | 2 | 2 | 6 | 4 | 2 |
| P4 | 3 | 1 | 8 | 5 | 4 |

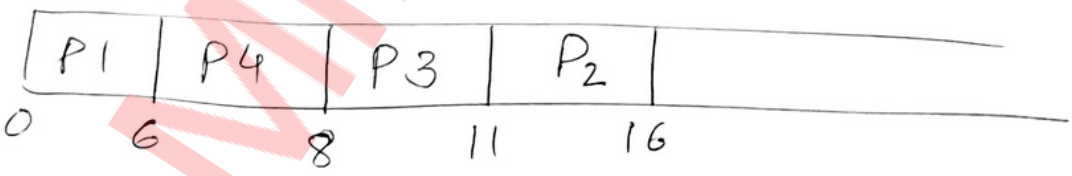



④ Priority (Preemptive)

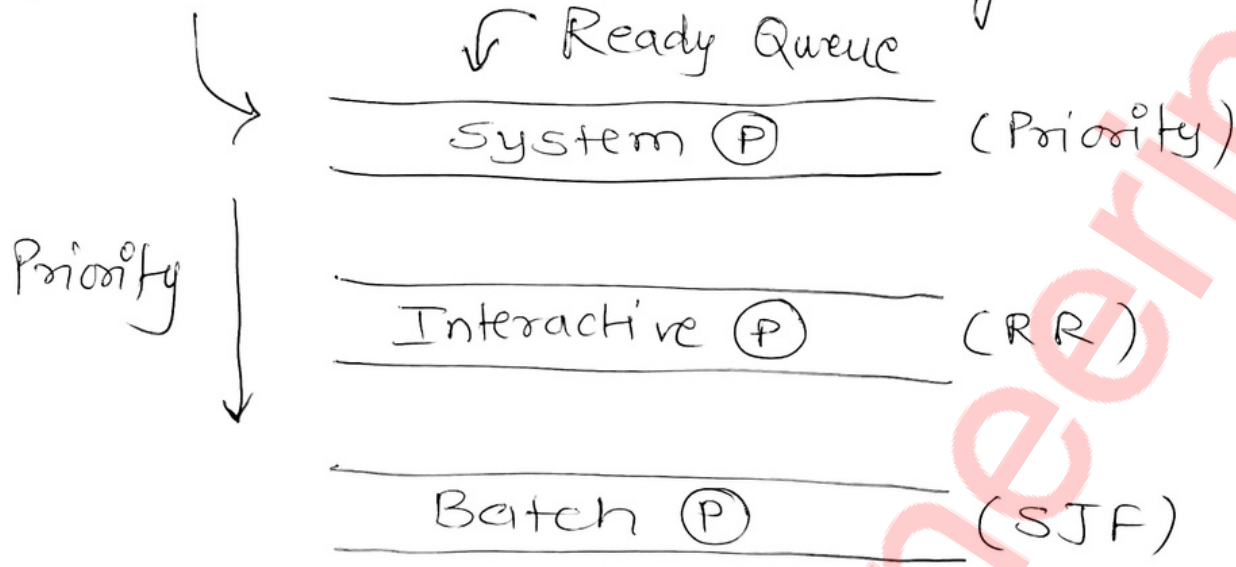
| Pno | Priority | AT | BT | CT | TAT | WT |
|---|---|---|---|---|---|---|
| P1 | 2 | 0 | 6 5 | 16 | 16 | 10 |
| P2 | 4 | 1 | 5 4 | 11 | 10 | 5 |
| P3 | 6 | 2 | 3 1 | 7 | 5 | 2 |
| P4 | 8 | 4 | 2 | 6 | 2 | 0 |



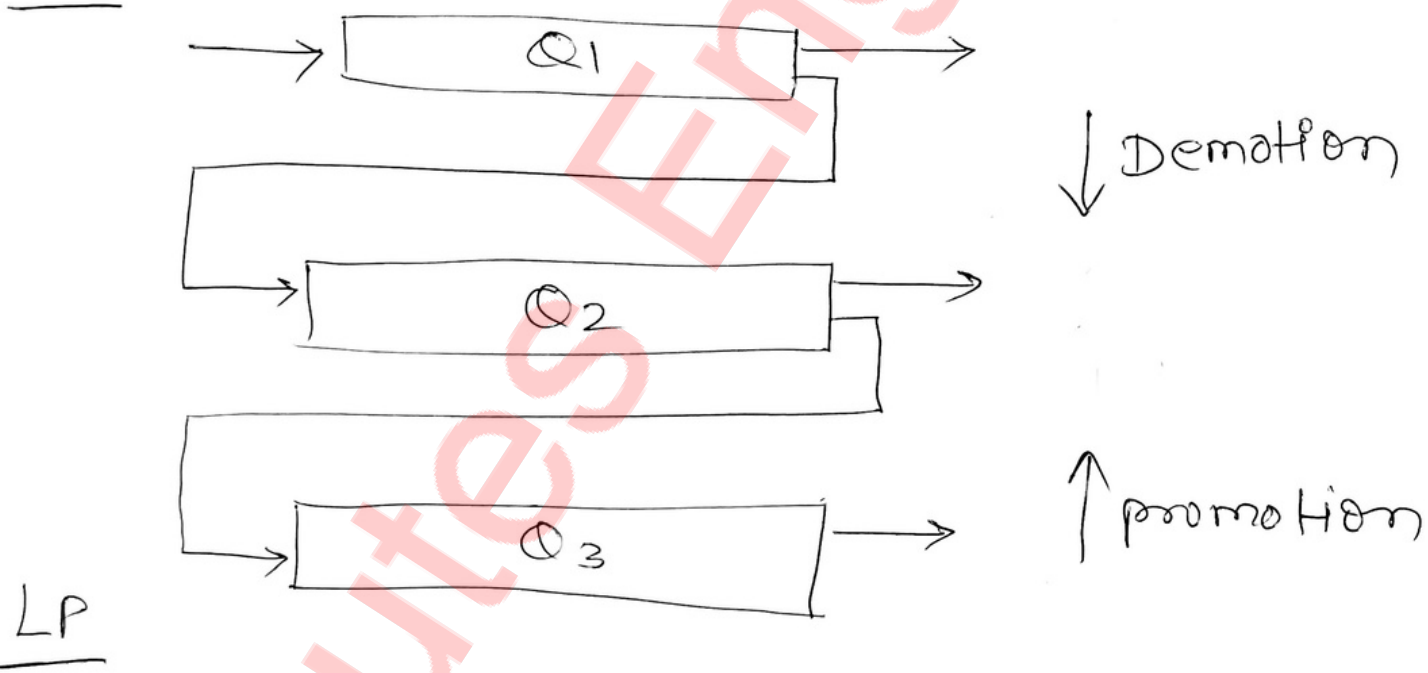

→ (Non preemptive)



# Multi Level Queue Scheduling




## feedback



# Process Synchronization

→ Multiple processes can access shared resources without interfering with each other & maintain data consitency.

Process → Independent (No affect)

→ Cooperative (affect other or get affected by other)

Common → data, memory, Resource

## Race Condition

data = 100

| $P_1$ | $P_2$ |
|---|---|
| a = data | b = data |
| a = a + 50 | b = b + 5 |
| — Preempt — | — Preempt — |
| data = a | data = b |

data = 150

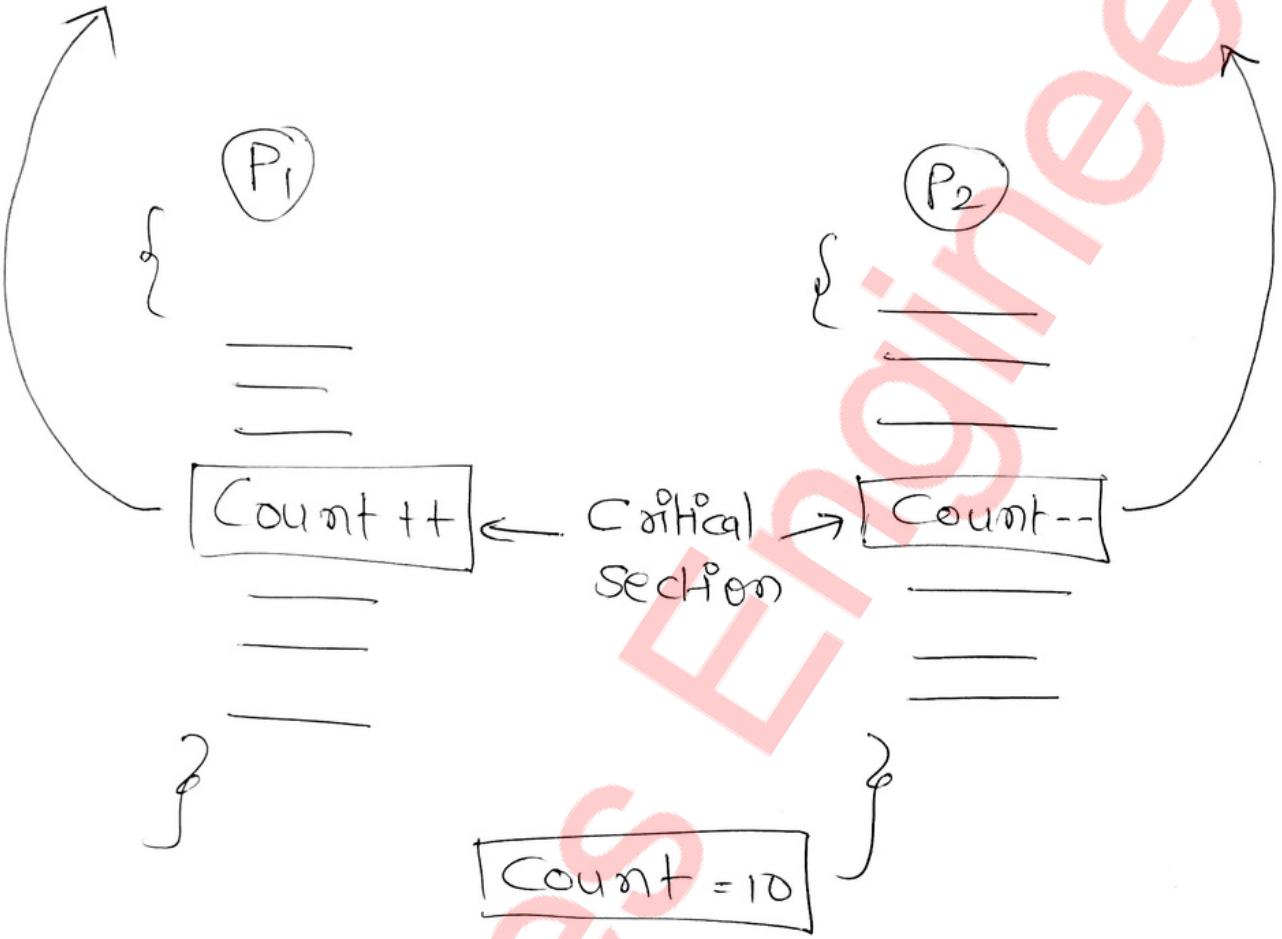
data = 105

Eg Count++:

① write R1, count
② INC R1
③ write count, R1

Count--;

① write R2, count
② DEC R2
③ write count, R2




Entry section

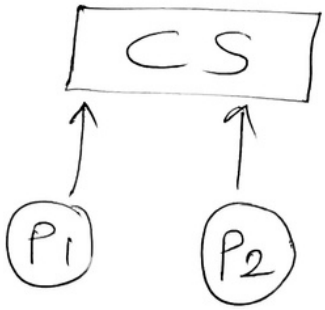
Critical section

Exit section

# Critical section Problem (soln)

## Conditions

### ① Mutual Exclusion




{ at a time (P1) or (P2) }
only → (1) ← Process

### ② Progress

Interested Process can go → CS
Not Interested (P) should not stop interested (P).

### ③ Bounded waiting (Not mandatory)

A Bound or limit should exist for a process to how many times a process can enter their CS.

CS

(✓) 1 2 3 (P1) (P2)
(✗) 1 2 - - ∞

- Solutions

## ① Two Process Solution

### a) Turn Variable

P1

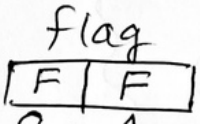
```
while (1)
{
    while(Turn != 0);
        CS
    Turn = 1
        NCS
}
```

P2

```
while (1)
{
    while(Turn != 1);
        CS
    Turn = 0;
        NCS
}
```

- Mutual Exclusion ✓
- Progress ✗ [P2 Must go to CS then only P1 can visit CS]

### b) Array [Flag]

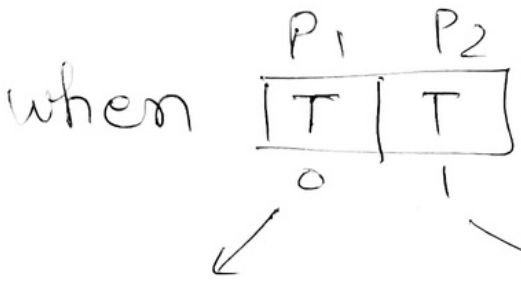



P1

```
while (1)
{ F[0] = True
  while (F[1])
      CS
  F[0] = False
      NCS
}
```

P2

```
while(1)
{
  F[1] = True
  while (F[0])
      CS
  F[1] = False
  NCS
}
```




NO process can enter in CS as we encountered "Deadlock"

## c) Peteeson's solution

→ use [Turn variable] + [Flag Array]

P1 P2
F F
0 1

T = 0

**P1**

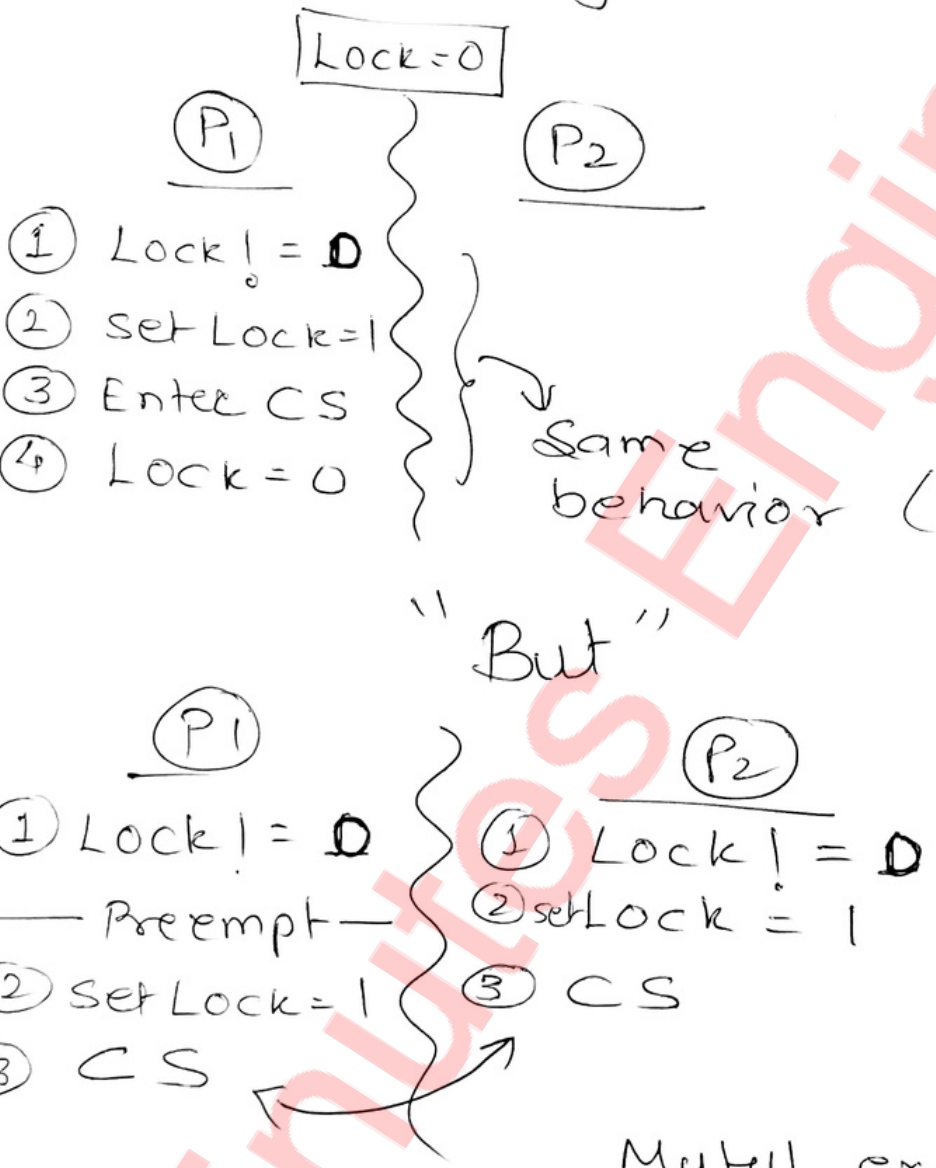
```
while(1)
{
  F[0] = True
  Turn = 1
  while(Turn==1 && F[1]==True)
  CS
  F[0] = False
  NCS
}
```

**P2**

```
while(1)
{
  F[1] = True
  Turn = 0
  while(Turn==0 && F[0]==True)
  CS
  F[1] = False
  NCS
}
```

# Lock variable

① while ( Lock == 1 ); → or "!= 0" } Entry
② Lock = 1
③ CS
④ Lock = 0 } Exit




Mutul exclusion (x)

- Test & set

Here we combine ① & ② and make it an atomic instruction.

① Test : ( LOck != 0 )
&
② Set : LOck = 1

] combine hence the name Test & set.

```
[while ( Test&set(&Lock)); ]
  CS
  Lock = false
  NCS
```

```
Boolean Test&set (Boolean *target)
{
    Boolean s = *target
    *target = True
    return = s;
}
```

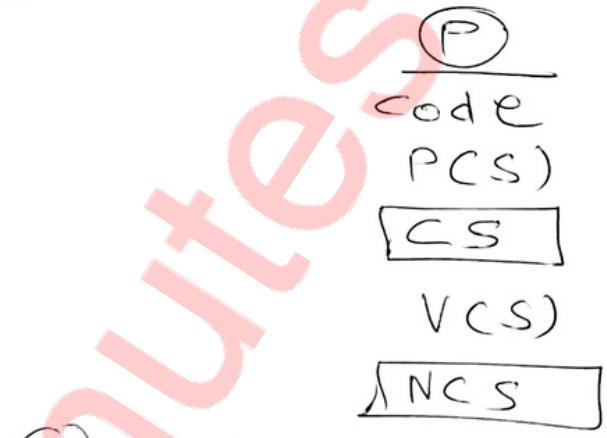
- Semaphores
  - Dijkstra (1965)
  - Managing concurrent processes by use of a Integer value (variable)

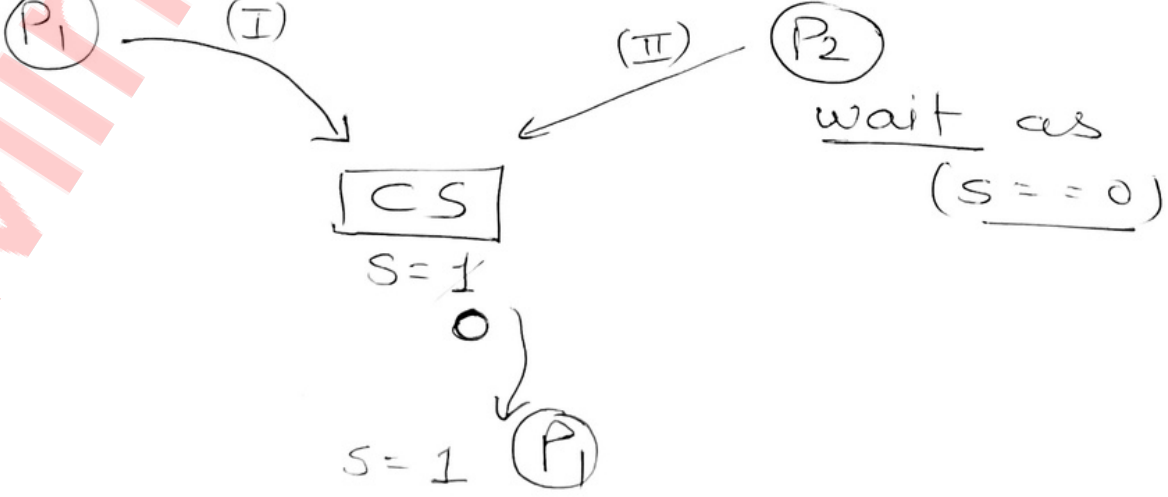
Binary semaphore ('0' or '1')

P(S) { wait, sleep or down)

V(S) { signal, wake-up or up)

```
P ( Semaphore S)          V (Semaphore S)
{ while (S==0);           {
      S--;                      S++;
}                         }
```



- Counting Semaphore ($-\infty$ to $\infty$) ⇐ int variable value


```
P(Semaphore S)
{
   S--
  if (S<0)
    {
      Put (P) Pn
      Block list sleep();
    }
    else
    return
}
```

```
V(semaphore S)
{
  S = S+1
  if (S<=0)
    { select (P)
    from Block list
     wake up();
    }
}
```

S → -ve (how many (P) are in Block list)

S → 0 (No (P) in Block list)

# Producer Consumer Problem.

```
Producer() {
    while (True) {
        // Produce an item
        wait (Empty);
        wait (mutex);
        Insert();
        signal (mutex);
        signal (full);
    }
}
```

```
consumer() {
    while (True) {
        wait (full);
        wait (mutex);
        consume();
        signal (mutex);
        signal (Empty);
    }
}
```

Semaphore
- full : '0' filled positions
- Empty : 'N' empty spaces
- mutex : for mutual exclusion.

# Reader-Writer Problem

Can have many → Readers / Writers

(R-R) ✓ allowed.

But at the same time

(W-W) & (W-R) & (R-W)

NOT allowed.

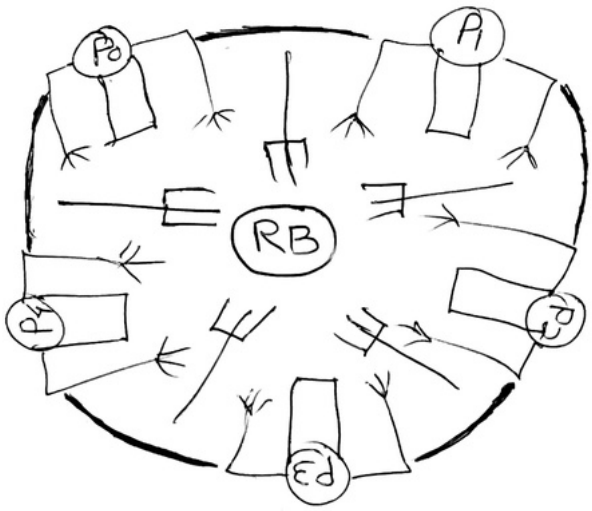
```
Reader() {
wait(mutex)
RC++
if (RC==1)
    wait(write)
signal(mutex)
Read() //CS
wait(mutex)
RC--
if(RC==0)
    signal(write)
signal(mutex)
}
```

```
write() {
wait(write)
write() //CS
signal(write)
}
```

mutex = 1
write = 1
RC = 0

Initially.

# Dining Philosopher Problem




forks is "shared Resource"

```
void Philosopher (void)
{
    while (True)
    {
        Thinking();
        down (F[i]);
        down (F[(i+1) % N]);
        Eat();
        up (F[i]);
        up (F[(i+1) % N]);
    }
}
```

Left 'F' ← Take fork

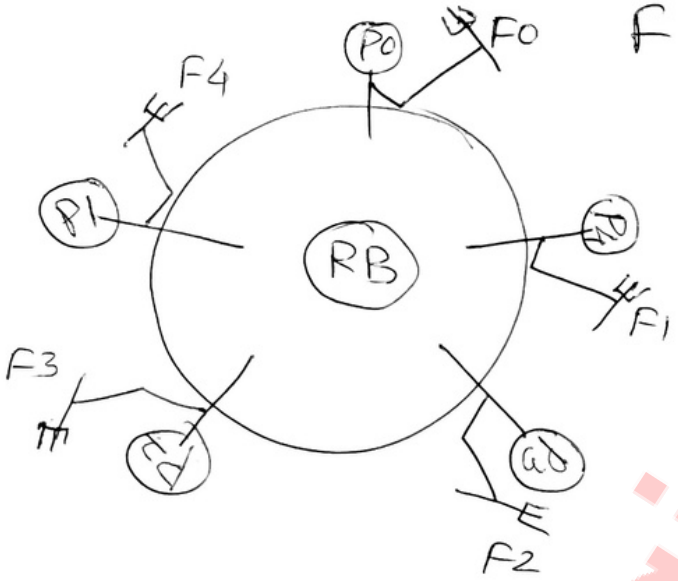
Right 'F' ← Take fork

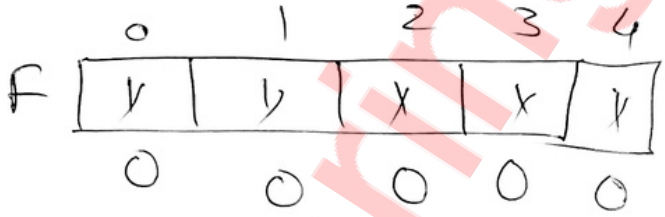
Keep/put fork

| F | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| | 1 | 1 | 1 | 1 | 1 |

| | 'L' Fork | 'R' |
|---|---|---|
| P0: | 0 | 1 |
| P1: | 1 | 2 |
| P2: | 2 | 3 |
| P3: | 3 | 4 |
| P4: | 4 | 0 |

# Deadlock situation.



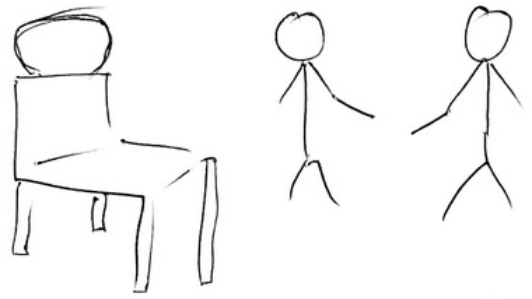

all have "left fork"

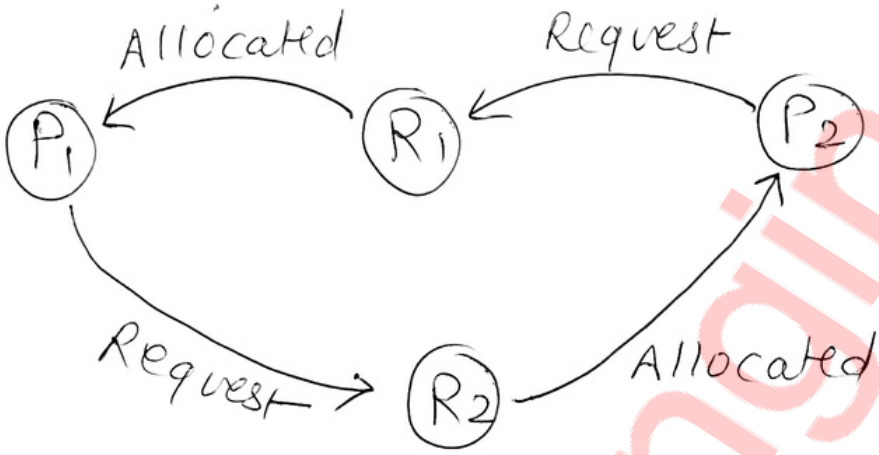
for (Pi) we can reverse the order of picking fork ('R' then 'L')

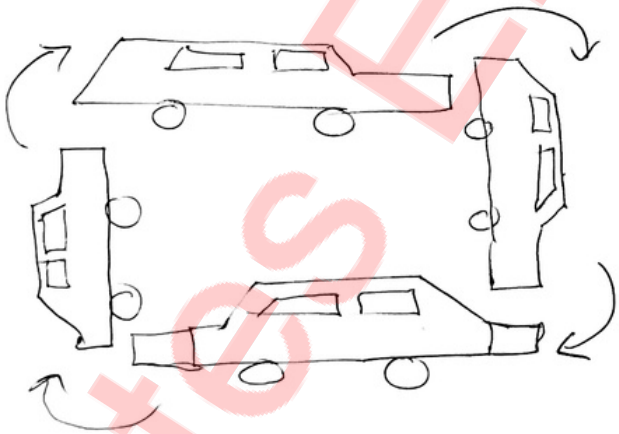
→ wait/down(F[(i+1) % N])
wait/down(F[i])

⇒ (Pi) gets blocked.

# Deadlock → "Aap pehle"





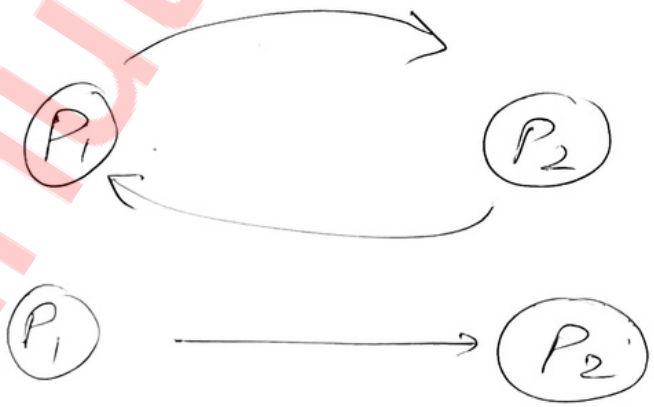
- Conditions for deadlock
  - → Mutual Exclusion
  - → Hold & wait
  - → NO preemption
  - → Circular wait

- Mutual Exclusion: Only 1 (P) can use a Reasource at a time (Non-shareable)

- Hold & wait: (Pi) holding at least One Resource & wait to get more resources that are acquired by other (Pj).

- NO Preemption: (P) can only release the Resource voluntorily after completing the process.

- Circular wait:




All conditions must be holding.

- Deadlock Handling Methods

① Prevention : Its always better than cure.

→ Deadlock should not happen. "very strict"

"Drive slow to prevent accident"

- Mutual Exclusion : Not all resources can be made shareable.
- Hold & wait : can voilate it.

↓

Process should have all required resource before Running. otherwise Not allowed to Run.

To request for additional Resource first release the Resource that is being held.

Time out on waiting.

- NoPreemption : Can voilate it
  → Preemption can be done based on priority.
- Circular Wait : can voilate it
  → Process can request resource in higher/increasing order.

- **Avoidance** : If you can't prevent try to avoid it.
  (A bit less strict)

→ Here we try to provide all info to system.

- **Banker's Algorithm**

Total = (5, 5, 5) — (R1) & (R2) & (R3)

Allocated = (5, 4, 3)

Available = (0, 1, 2)

| | R1 | R2 | R3 | R1 | R2 | R3 |
|---|---|---|---|---|---|---|
| P0 | 1 | 2 | 1 | 1 | 0 | 3 |
| P1 | 2 | 0 | 1 | 0 | 1 | 2 |
| P2 | 2 | 2 | 1 | 1 | 2 | 0 |
| | [Allocated] | | | [Need / Requested] | | |

So we can fullfill the need of P1

Need → 0 1 2 available → 0 1 2

∴ New Available = 2 0 1
+ 0 1 2
= 2 1 3

Next (P0)

∴ NA = 1 2 1
+ 2 1 3
= 3 3 4

Next (P2)

NA = 3 3 4
+ 2 2 1
= 5 5 5

(P1) → (P0) → (P2) Safe sequence

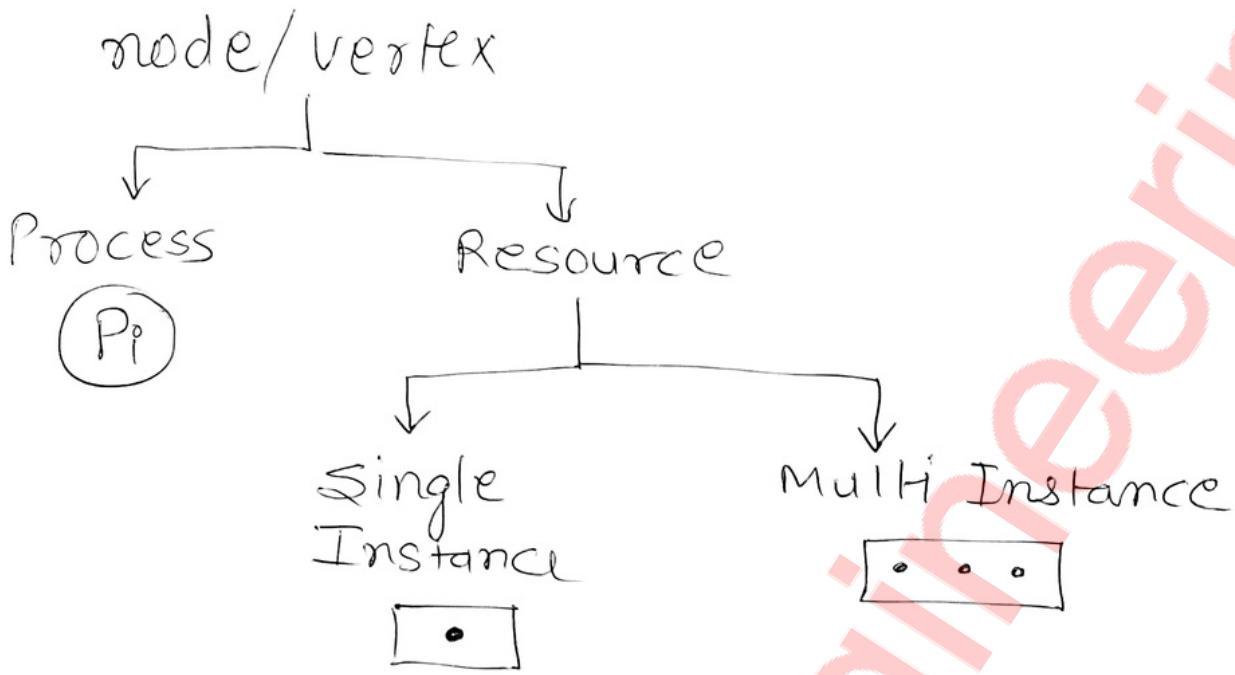
- Detection & Recovery

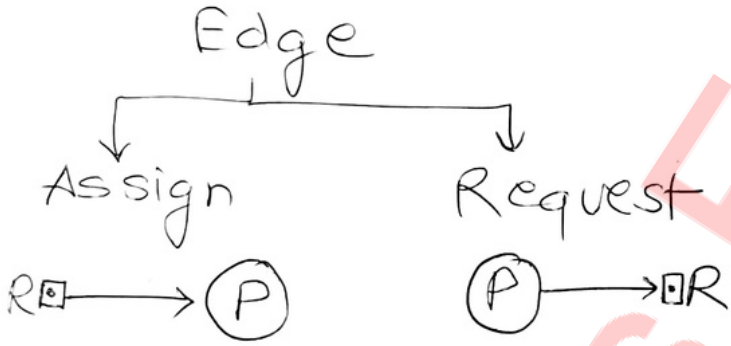
  ↓ first detect (Banker's Algo) (Resource Allocation Graph)

  ↓ then Recovery
  - → Process Termination
  - → Resource Preemption

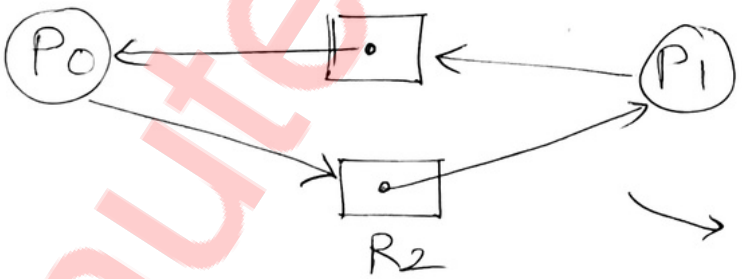
- Ignorance [Osteich Algorithm]
  - → No concept of deadlock exists.
  - → Windows, mac follow this approach to handle deadlock by Ignoring it.
  - → Its a Rare event.... and the cost required to handle Deadlock is high.

- Resource Allocation Graph.




Eg:



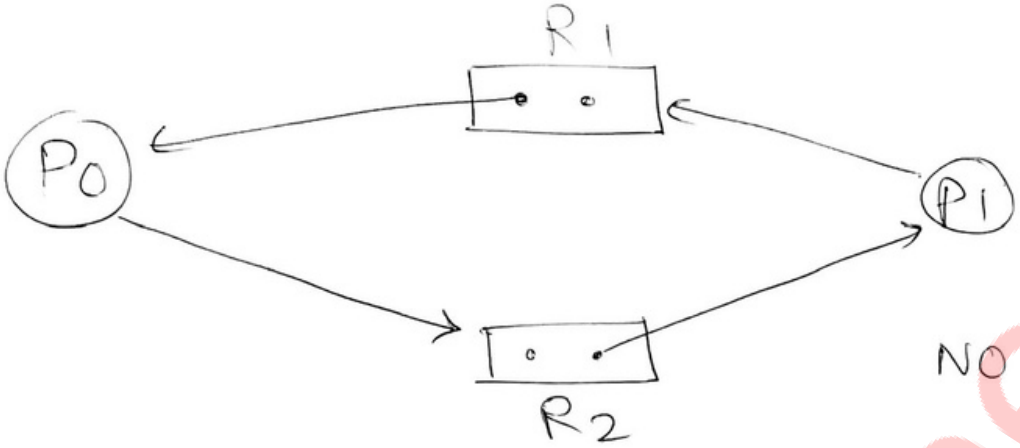

If cycle exists Deadlock May / May Not be present

If cycle doesn't exist then Deadlock doesn't exist

| Pno | Allocated (R1) | (R2) | Request (R1) | (R2) |
|---|---|---|---|---|
| P0 | 1 | 0 | 0 | 1 |
| P1 | 0 | 1 | 1 | 0 |

Available [0, 0] { P0 P1 } X

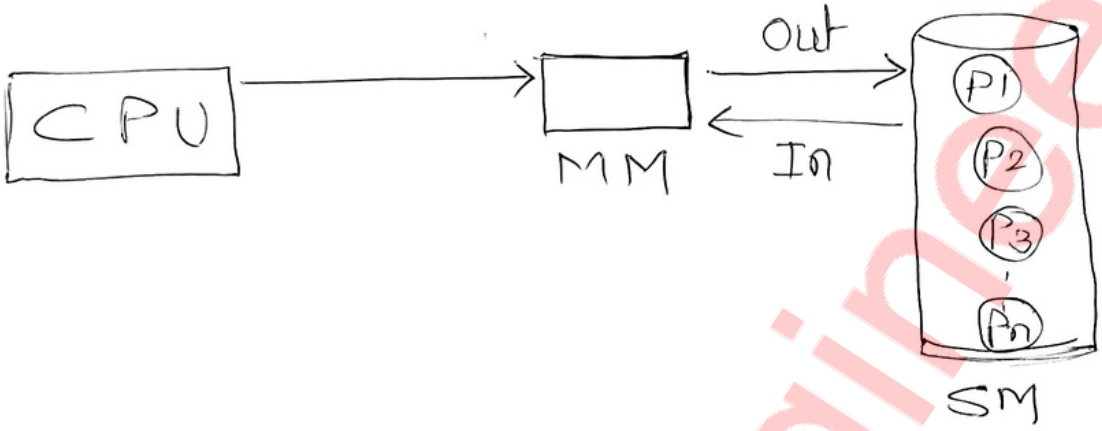
It is a deadlock situation.



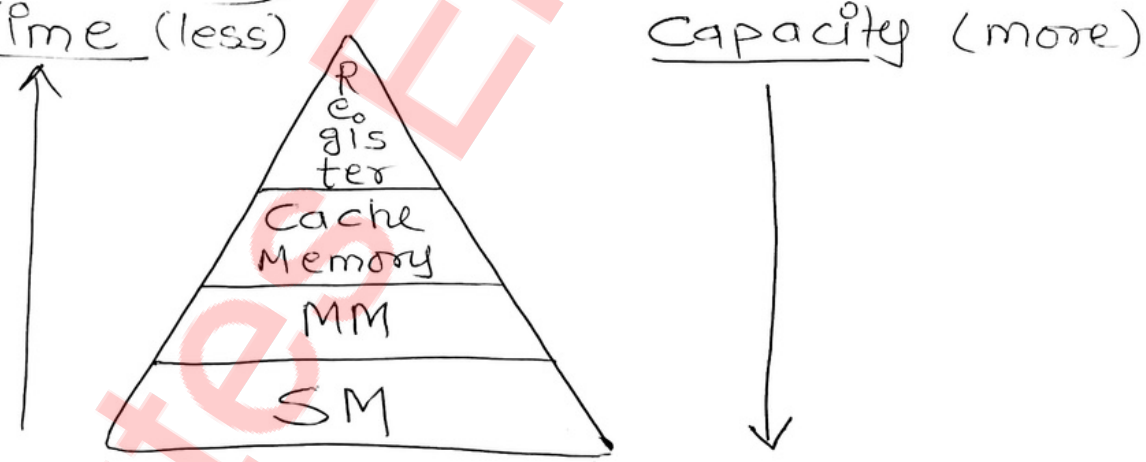

NO deadlock

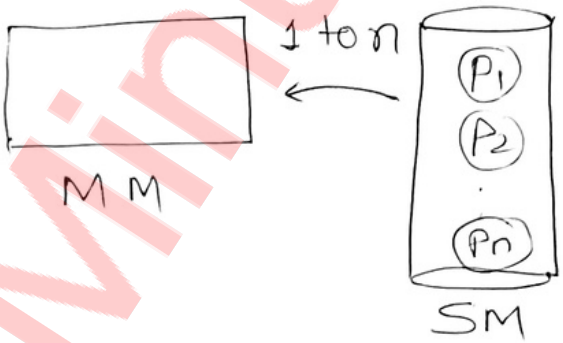
Available (1, 1) [ P0 P1 ] ✓

Memory Management : 1) Proper / efficient / optimal utilization of memory
↳ ka
2) Multiprogramming




"Memory hierarchy"

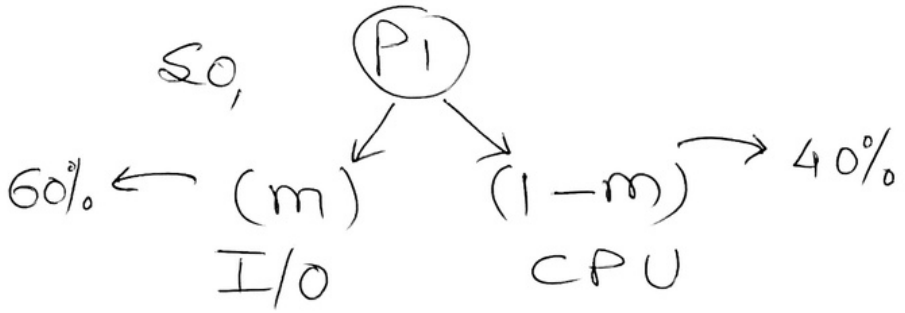



Process → CPU
↳ I/O

Degree of MP ∝ CPU utilization

- no of process in MM = (1)




NOW Lets consider 2 process in MM

So, CPU utilization = $(1-m^2)$

$= 1-(0.60)^2$

$= 1-0.36$

$= 0.64 \rightarrow 64\%$

NOW 3 process in MM

So, CPU Utilization = $(1-m3)$

$= 1-0.216$

$= 0.78 \rightarrow 78\%$

Now 10 process in MM

So, CPU utilization = $(1-m^{10})$

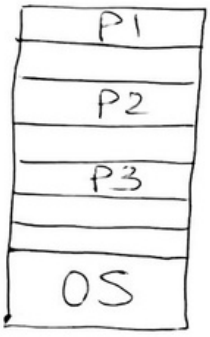
$= 1-0.0060466$

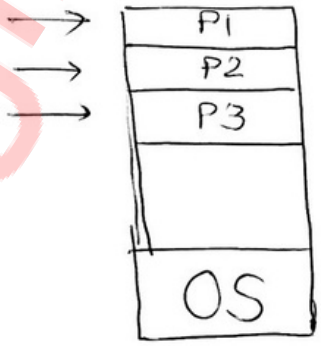
$= 0.9939 \rightarrow 99\%$

So, → $\boxed{(1-m^n)}$

# Memory Mgmt Techniques

- Non-Contiguous
  - Paging
  - Segmentation
  - Segmented Paging
- Contiguous
  - Fixed Partition
  - Variable partition

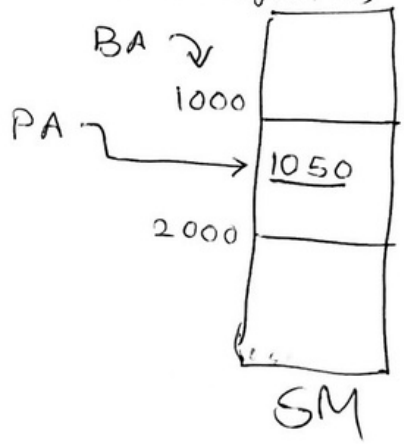



→ Contiguous [Addr. Traslation]

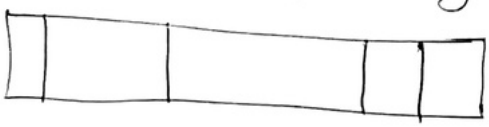
$PA = LA + BA$

- PA: Physical Addr. (o/p)
- LA: Logical Addr (Given by CPU)
- BA: Base Addr (Relocation Register)

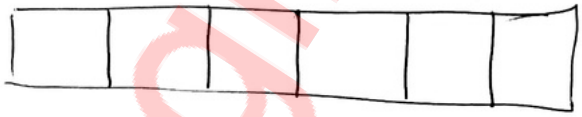


- Fixed partitioning

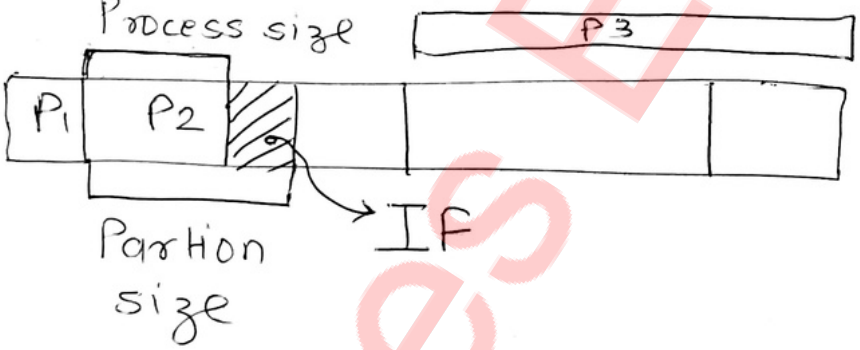
  - Fixed no. of partitions
  - Partition size may be different
  - Spanning Not Allowed.
  - Wastage of memory. (IF)
  - Size limitation. (Not above max Par.)



Unequal Partitioning



Equal Partitioning.



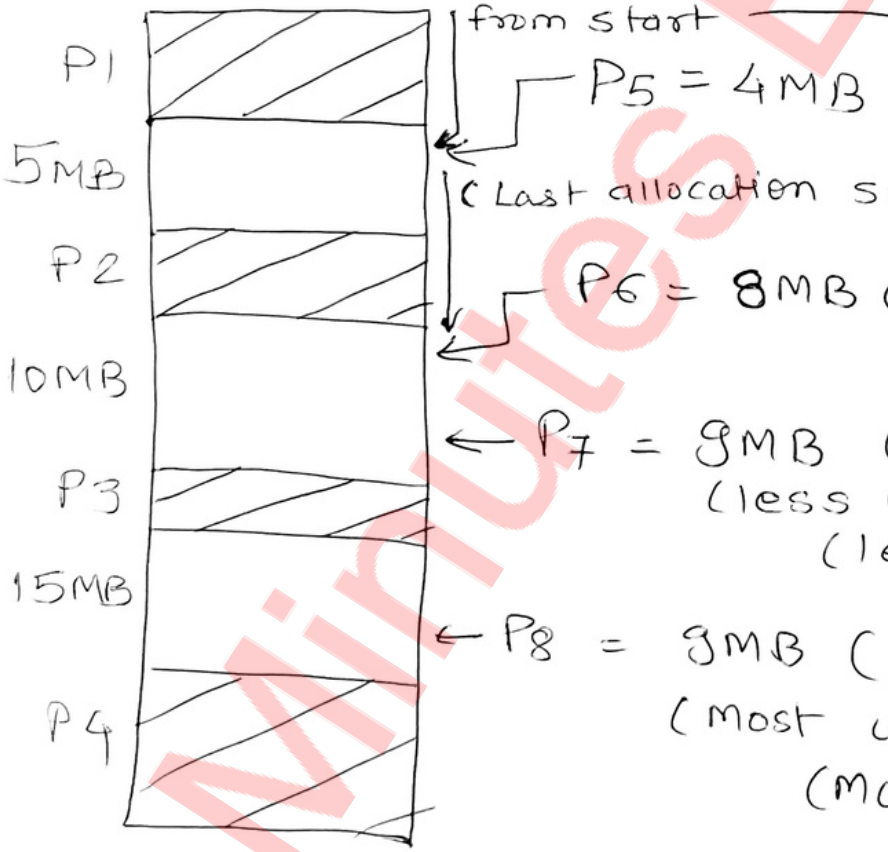

⇒ Degree of Multiprogramming is affected here.

⇒ External fragmentation exists (Have space but can't allocate to (P)) Due to contiguous condition

# Variable Partitioning

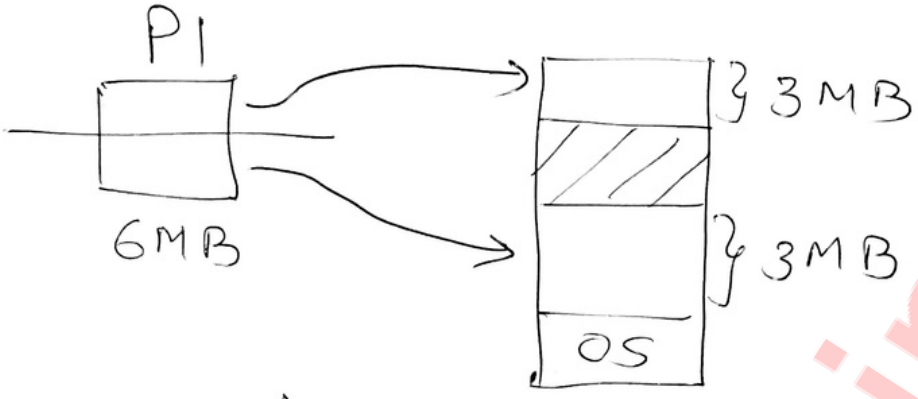
- Variable size Partitions (Dynamic Nature)
- No internal fragmentation
- External Fragmentation Exist
- No effect on DOM.
- No Restriction on Process size.

⇝ Now to Remove the EF problem.
[Process shifting bw- expensive]



- Non-Contiguous Memory Allocation

↳ Used to fix (EF).




↑

So here always we need to analyze the no. of holes & their size to partition the Process accordingly.


$$P.S = F.S$$

PNO → P1

| | | | | |
|---|---|---|---|---|
| 0 | 0 | 1 | 2 | 3 |
| 1 | 4 | 5 | 6 | 7 |
| 2 | 8 | 9 | 10 | 11 |
| 3 | 12 | 13 | 14 | 15 |

FNO → MM

| | | | | | |
|---|---|---|---|---|---|
| 0 | 0 | 1 | 2 | 3 | |
| 1 | 4 | 5 | 6 | 7 | P[0] |
| 2 | 8 | 9 | 10 | 11 | |
| 3 | 12 | 13 | 14 | 15 | P[1] |
| 4 | 16 | 17 | 18 | 19 | |
| 5 | 20 | 21 | 22 | 23 | P[2] |
| 6 | 24 | 25 | 26 | 27 | |
| 7 | 28 | 29 | 30 | 31 | P[3] |

Process size = 16 B

Page size = 4 B = frame size

MM size = 32 B

∴ No of pages = $\frac{16}{4}$ = 4 pages

∴ No. of frames = $\frac{32}{4}$ = 8 frames.

Logical Addr

| PNO | offset |
|---|---|
| 2 bit | 2 bit |

P[0] ← 0 0 : 0 0 → 1st B

P[1] ← 0 1 : 0 1 → 2nd B

P[2] ← 1 0 : 1 0 → 3rd B

P[3] ← 1 1 : 1 1 → 4th B

Translation →

Physical Addr

| fNO | offset |
|---|---|
| 3 bit | 2 bit |

F[0] 0 0 0

F[7] 1 1 1

CPU → 5th Byte → Logical addr

| | |
|---|---|
| 0 1 | 0 1 |

MMU

PT

| | |
|---|---|
| 0 | F1 |
| 1 | F3 |
| 2 | F5 |
| 3 | F7 |

Physical addr

| | |
|---|---|
| 0 1 1 | 0 1 |

frame no: 03 ← 2nd Byte

Page Table → It a Data Structure & every process has a Separate PT, stored in MM

→ No. of Pages = No Enteies of In PT

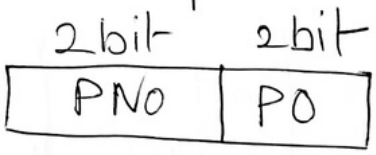
→ PageTable entry size = No. of frames.


Problem: ① Need 2 Access
- → Page Table
- → Actual byte data

② Storage overhead (To store PT)

③ Traslation is complex task

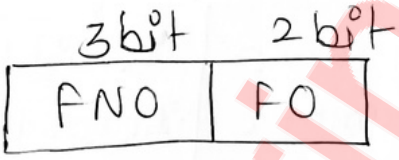
- Logical Addr Space




$\Rightarrow LAS = 2^4$

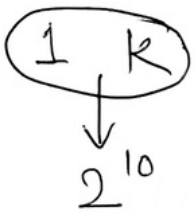
$= 16\,B$

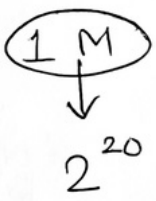
- Physical Addr Space



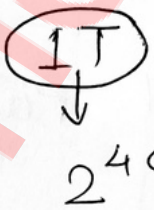

$\Rightarrow PAS = 2^5$

$= 32\,B$





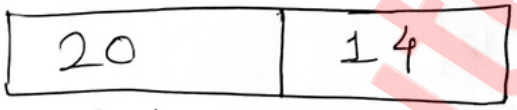



$\Rightarrow LAS = 16\,GB$

$2^4 \times 2^{30}$

$2^{34}$

34 bit LA




$= 34 - 14$

$= 20$ bit

No. of pages

$2^{20}$

1M

depends on PS/FS

$= 16\,KB$

$2^4 \times 2^{10}$

$2^{14} \rightarrow 14$ bit

$\Rightarrow PAS = 32\,GB$

$2^5 \times 2^{30}$

$2^{35}$

35 bit PA


$= 35 - 14$

$= 21$ bit

14 bit

No. of frames

$2^{21}$

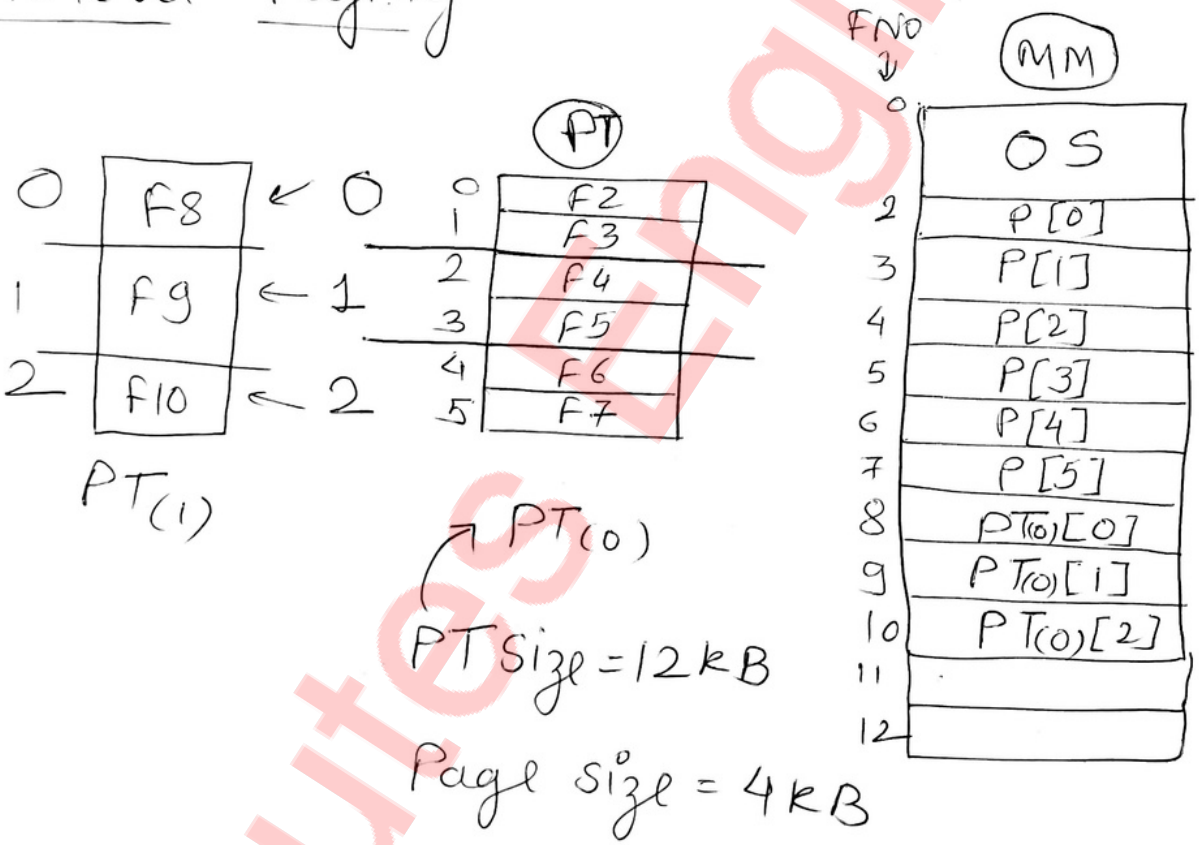
$2^1 \times 2^{20}$

$\Rightarrow 2M$

- Page Table Entry

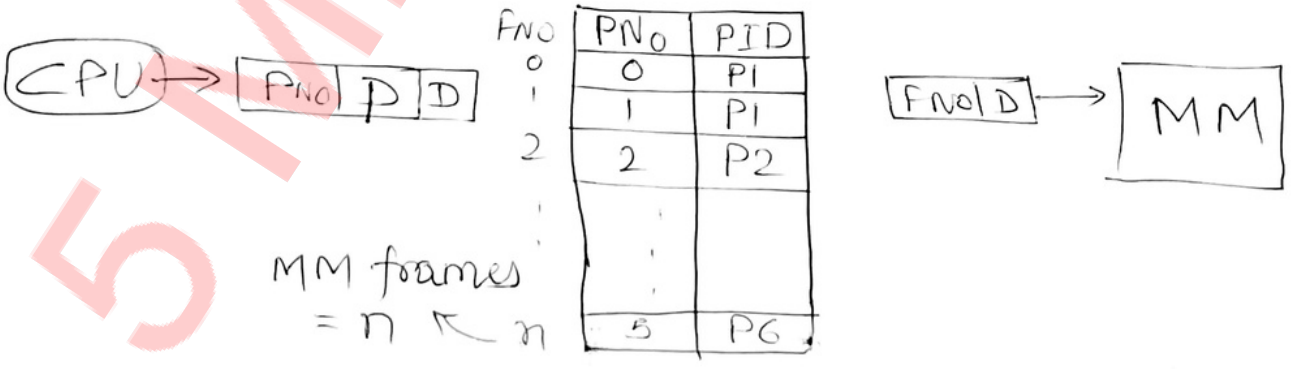
| Frame Number | Present Absent | Protection [Read write Execute] | Reference First Time / Not First Time | Caching E/D | Dirty Update |
|---|---|---|---|---|---|
| Mandatory part | | Optional part | | | |

- Multilevel Paging

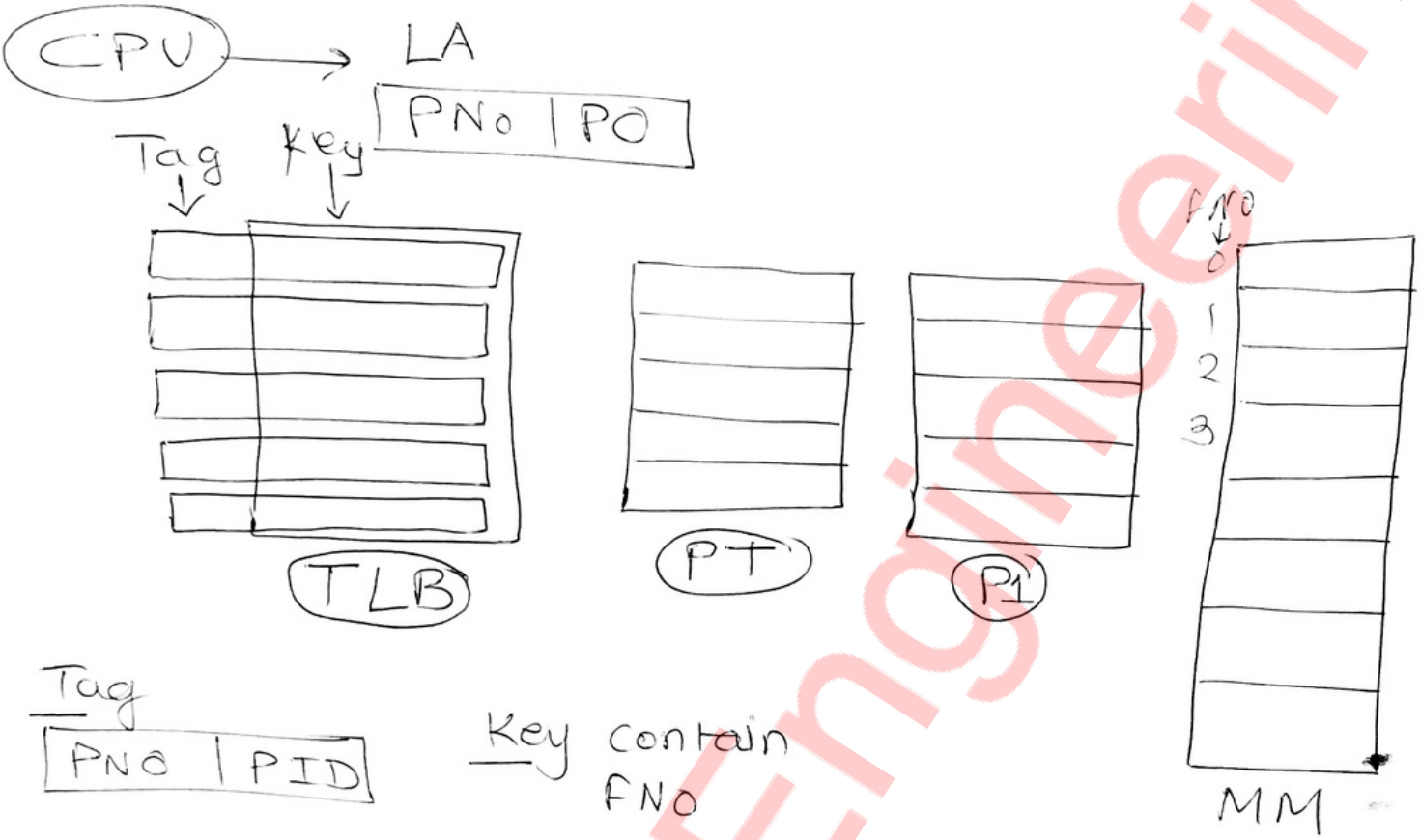



- Inverted Paging

→ Local PT (x) Global PT (✓)



# Translation Lookaside Buffer [TLB]




Tag

| PNO | PID |
|---|---|

Key contain FNO

| 02 | P1 |
|---|---|

→ F3 → found [TLB Hit]

→ Not found [TLB Miss]

$$EMAT = p(t+m) + (1-p)(t+2m)$$

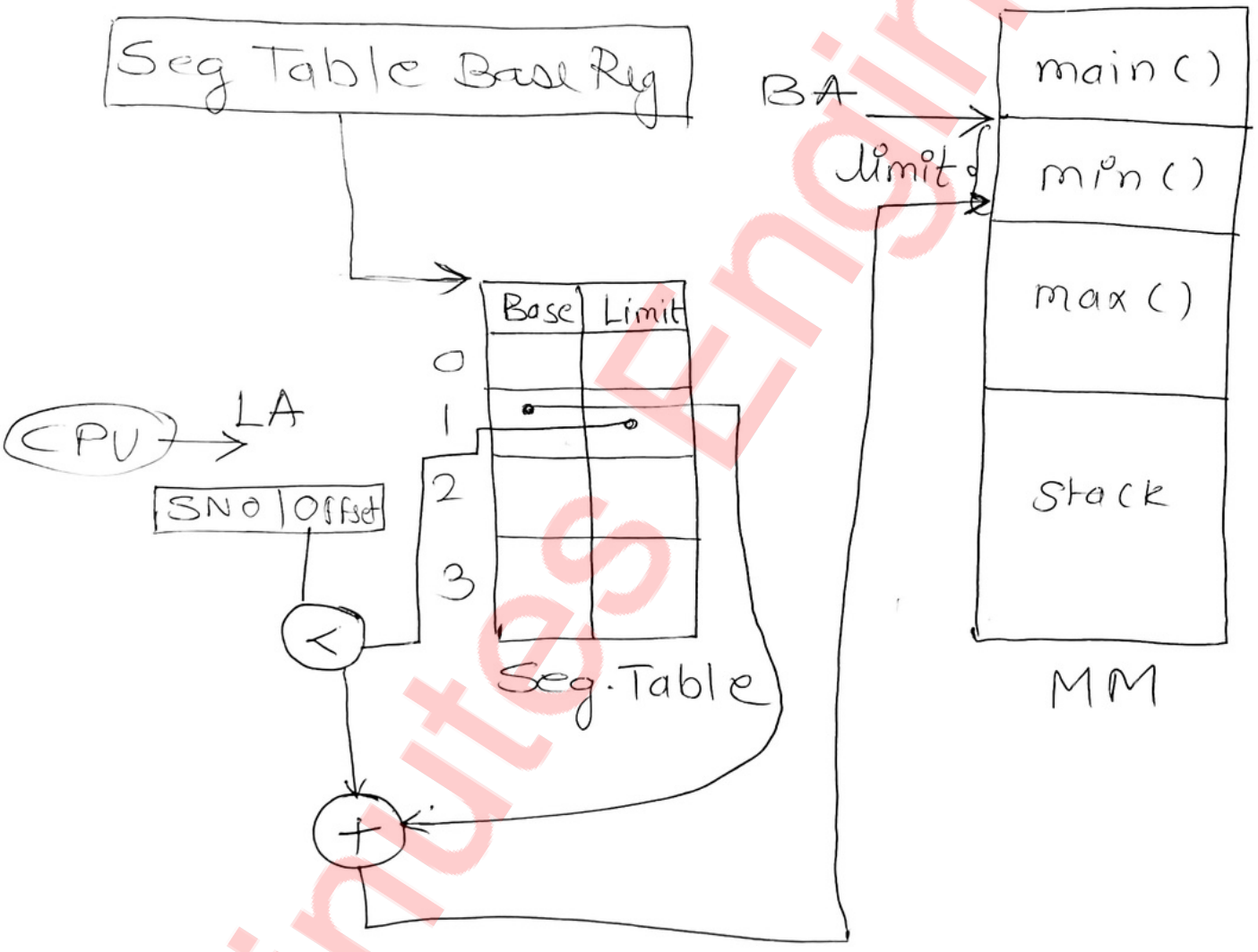
$p(t+m)$: TLB hit

$(1-p)(t+2m)$: TLB miss

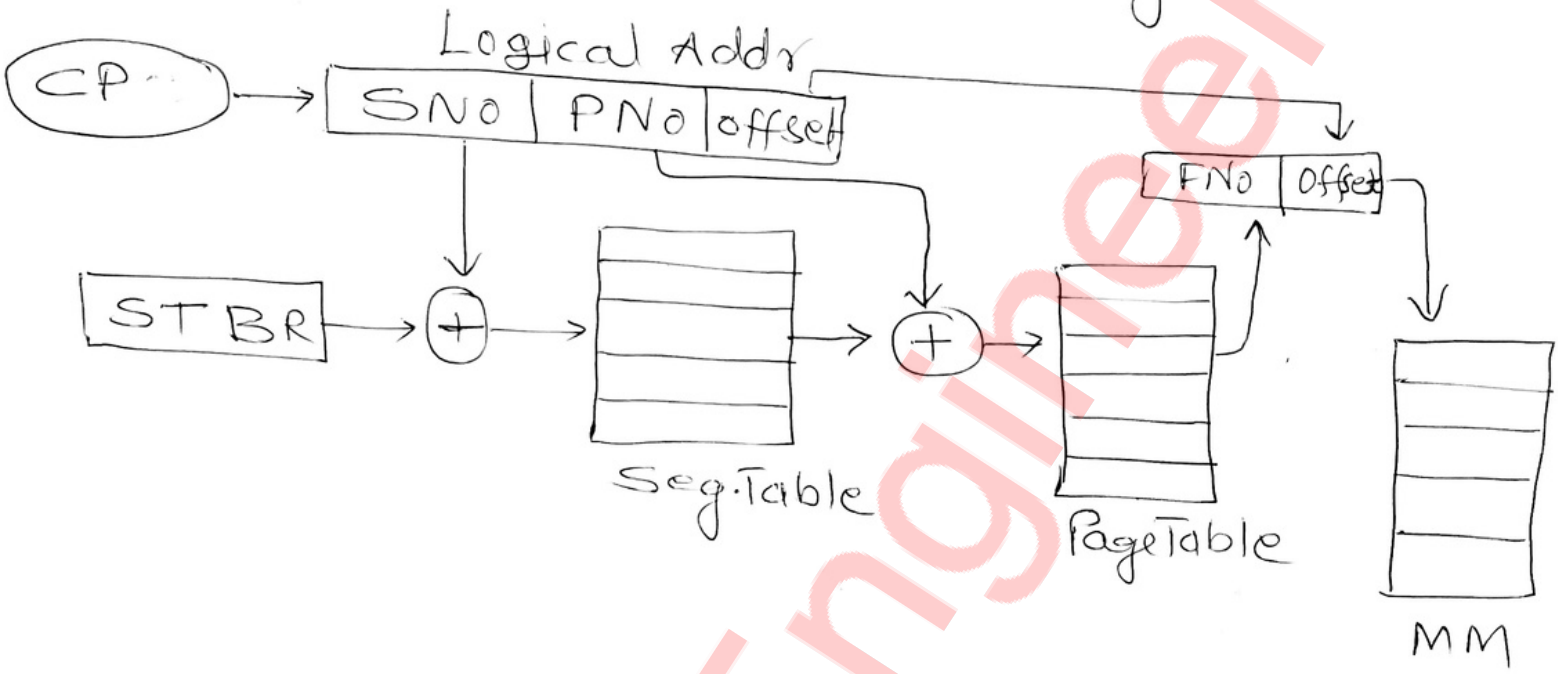
# Segmentation

→ Paging was near to OS view But away from User View

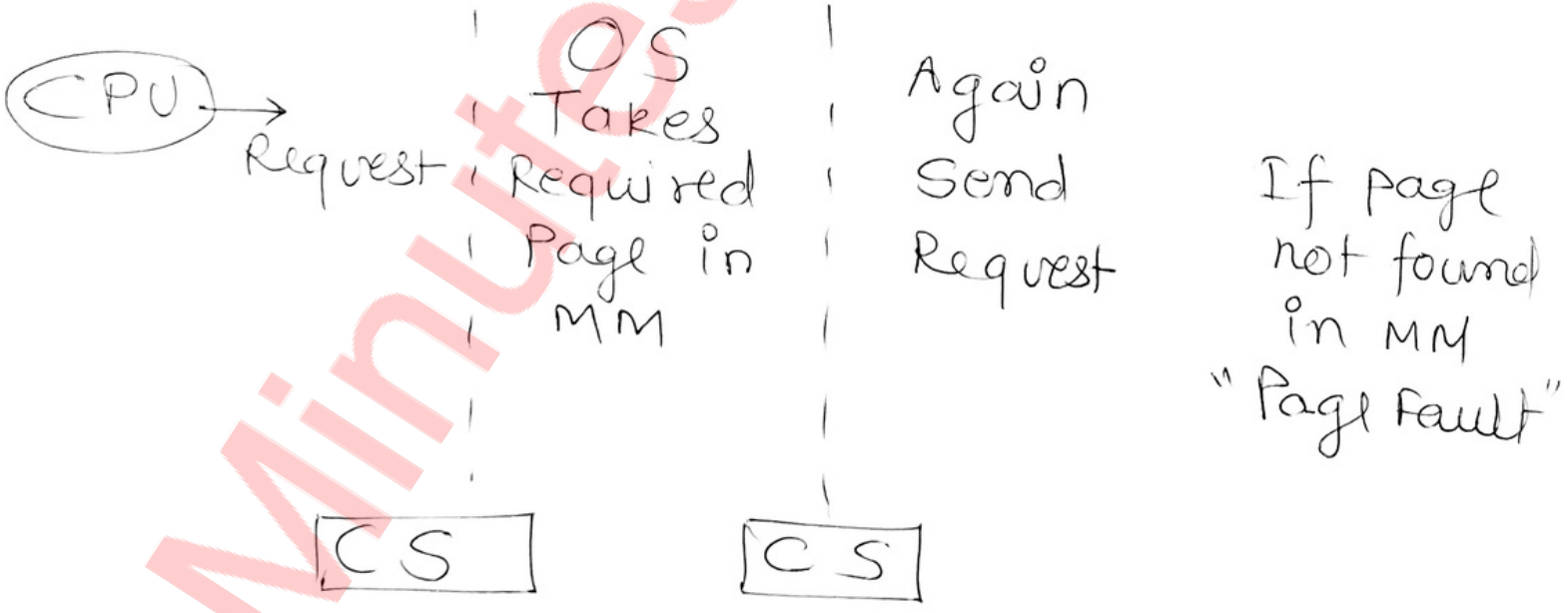
→ Segmentation support User view of Memory



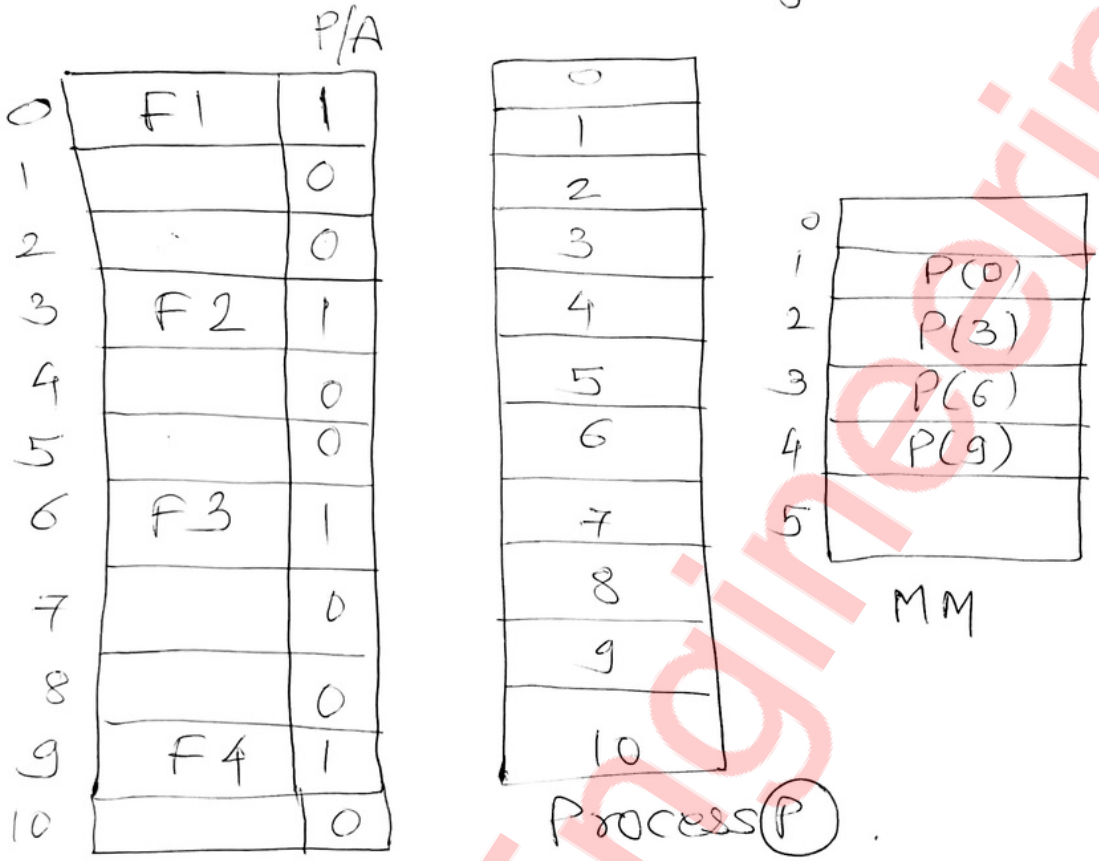
- Segmentation with Paging
  - ↳ Dividing Segment into Pages



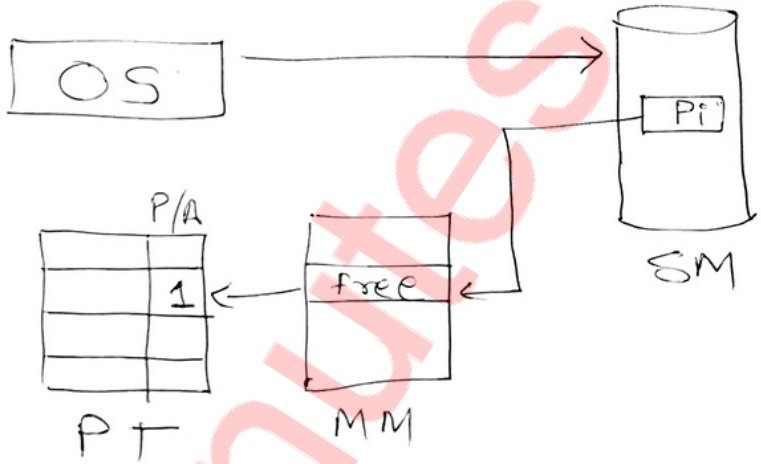

- Demand Paging & Page fault
  - ↳ Don't load until requested/Required



- Virtual Memory → Frame allocation
  → Page Replacement




"How to handle Page fault"




→ If frame is not free
then → check dirty bit

- 1 (modified) [Replace The Page in SM]
- 0 (NOT Modified) [Discard it]

- Frame allocation. No. of pages of (P) to load in MM.

  - (Min) — Min no. of pages req. to execute a Instruction
  - (Max) — All pages of (P)

- Equal Allocation.

  (P1) = 20 Pages
  (P2) = 100 Pages
  (P3) = 100 Pages

  60 frames

  $\frac{60}{3} = 20$

  P1 = 20
  P2 = 20
  P3 = 20.

- Weighted Allocation

  (P1) = 30
  (P2) = 40
  (P3) = 30

  No. of frames In MM = (10)

  (P1) = $\left(\frac{30}{100}\right) \times 10 = 3$

  (P2) = $\left(\frac{40}{100}\right) \times 10 = 4$

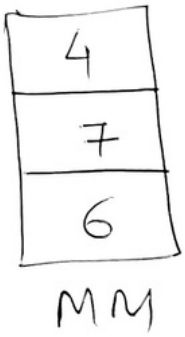
  (P3) = $\left(\frac{30}{100}\right) \times 10 = 3$

# ∘ Page Replacement Algorithm

## ① FIFO

⇒ 4, 7, 6, 1, 7, 6, 1, 2, 7, 2

No. of frames = 3

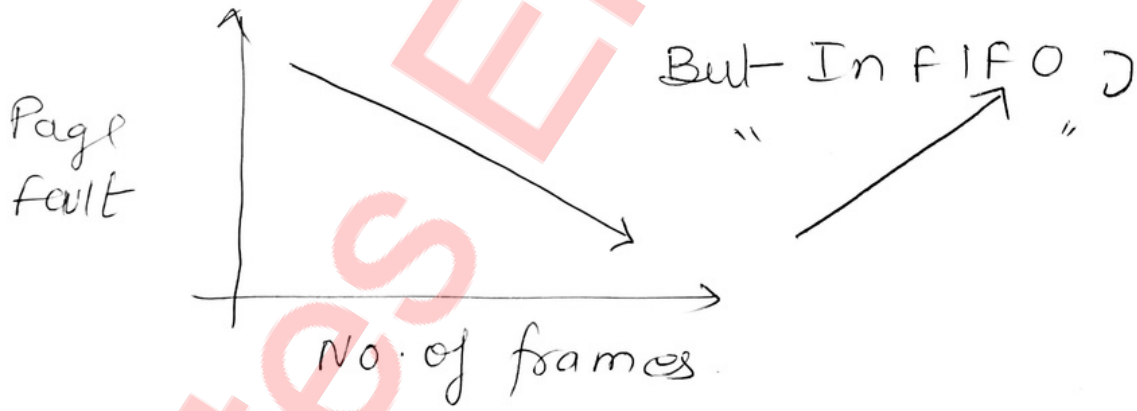



MM

4 7 6

Page fault = 6

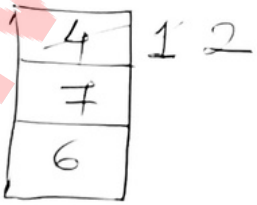
∘ Belady's Anamoly:




## ② Optimal "future"

⇒ 4, 7, 6, 1, 7, 6, 1, 2, 7, 2

frames = 3




Page fault = 5

## ③ LRU : "Past"

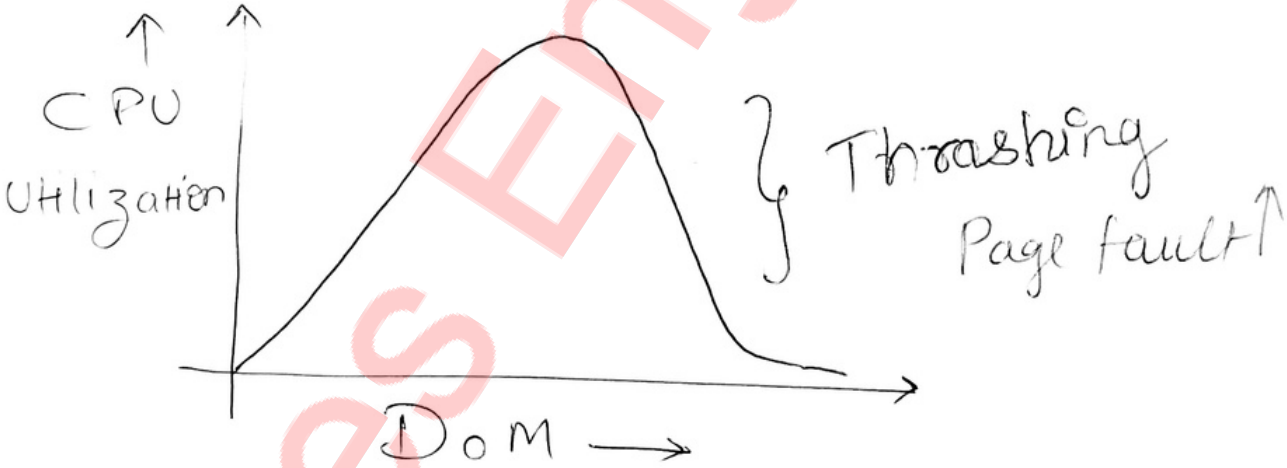
⇒ 4, 7, 6, 1, 7, 6, 1, 2, 7, 2

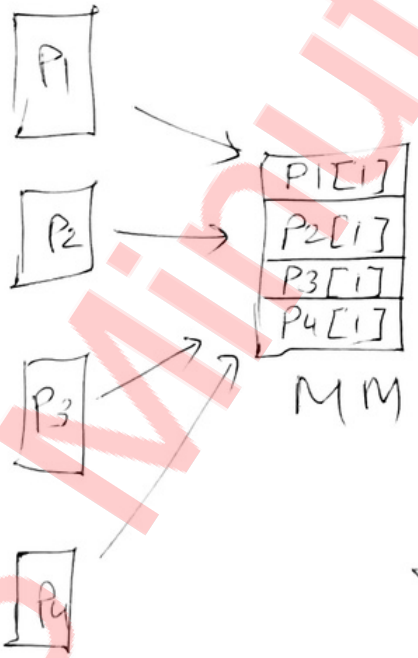
f = 3

| | |
|---|---|
| 4 | 1 |
| 7 | 2 |
| 6 | 7 |

Page fault = 6

- Thrashing



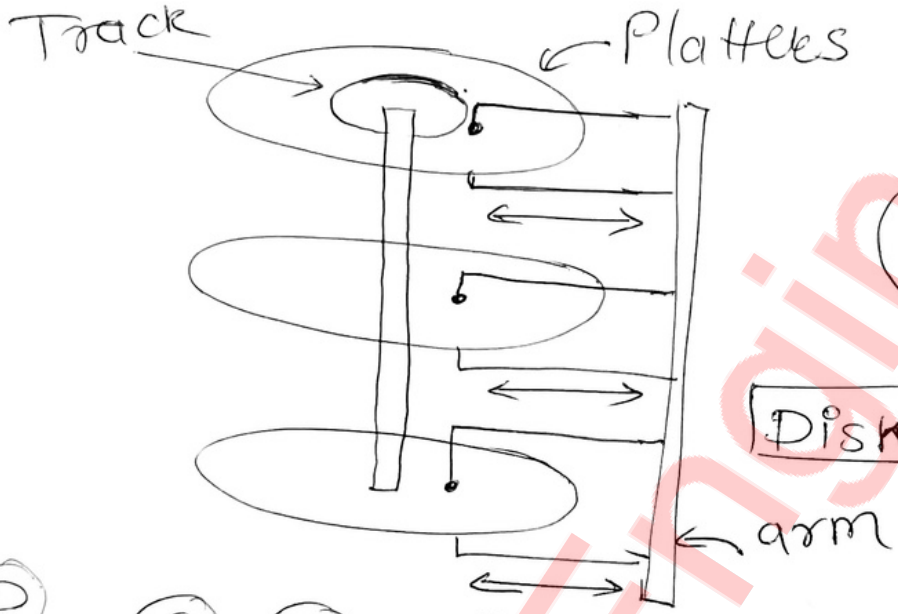

CPU needs
P1[2] P2[3] P3[2]
P4[3] {Page fault}

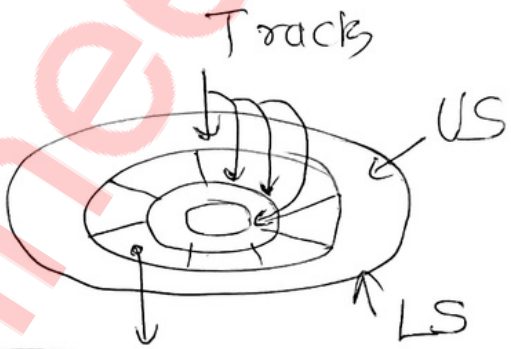
Soln: Refere to Past request of (P)

# Disk Mgmt

↳ ka ↗

## → Disk Architecture




Disk → sector → Platter → Surface → Track → Sector → Data

(P) (S) (T) (S) (D)

$8 \times 2 \times 256 \times 512 \times 512\ B$

$2^3 \times 2^1 \times 2^8 \times 2^9 \times 2^9\ B$

Disk Size $= 2^{30} B = 1MB$

# Disk access Time = ST + RL + TT

① **Seek Time:** Time required for R/W head to move on targeted track.

$$ST = n \times S$$

- $n$ → no. of track head move.
- $S$ → ST/track

② **Rotational Latency:** Time taken by R/W head to move to target sector.

$$RL = \frac{A}{F}$$

- $A$ → Angle bet$^n$ current & target 's'
- $F$ → Rotational frequency.

$= \frac{1}{2}$ full Rotation

③ **Transfer Time:** The time taken to R/W the data.

$$TT = \frac{B}{R}$$

- $B$ → Size of data in Bytes
- $R$ → Data transfer rate.

$$R = [\text{No. of head} \times \text{Capacity of one Track} \times \text{No. of Rotation in 1 sec}]$$

**Q →** Surfaces = 16
Tracks/surface = 128
Sectors/Track = 256
Data/sector = 512 B
Rotation = 3600 RPM

Find: 1) Disk capacity
2) No of bits to addr sector
3) Data Transfer Rate

① Capacity of Disk

→ 16 x 128 x 256 x 512 Bytes

↓ ↓ ↓ ↓

$= 2^4 \times 2^7 \times 2^8 \times 2^9$

$= 2^{28}$ bytes

$= 256$ MB

② No. of bits Required to addr sector

→ 16 x 128 x 256

↓ ↓ ↓

$= 2^4 \times 2^7 \times 2^8$

$= 2^{19}$ sectors

$= \underline{19 \text{ bits}}$

③ Data Transfer Rate:

↓

No. of head (surface) × Capacity of 1 track × No. of Rotation in one sec.

↓ ↓ ↓

16 × [256 x 512B] × [3600/60]

↓ ↓

$= 2^4 \times 2^8 \times 2^9 \times$ 60 bytes/sec

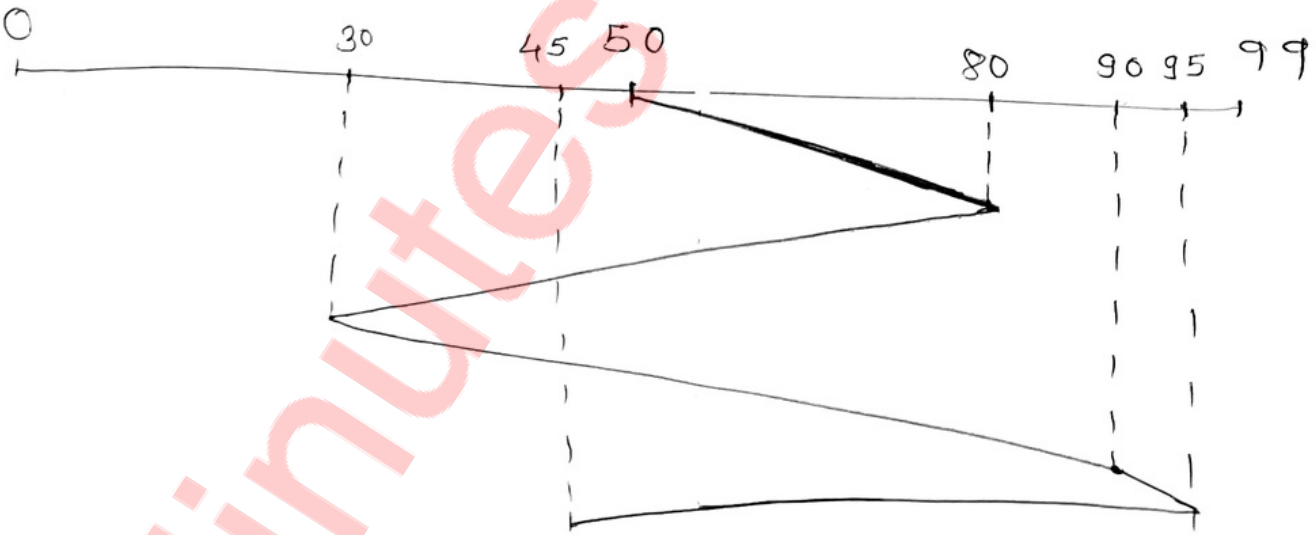
$= 60 \times 2^{21}$ B/sec = 120 MBps

- Disk scheduling Algorithm
{To minimize the seek Time}

① FCFS ② SSTF ③ SCAN
④ LOOK ⑤ C-SCAN ⑥ C-LOOK

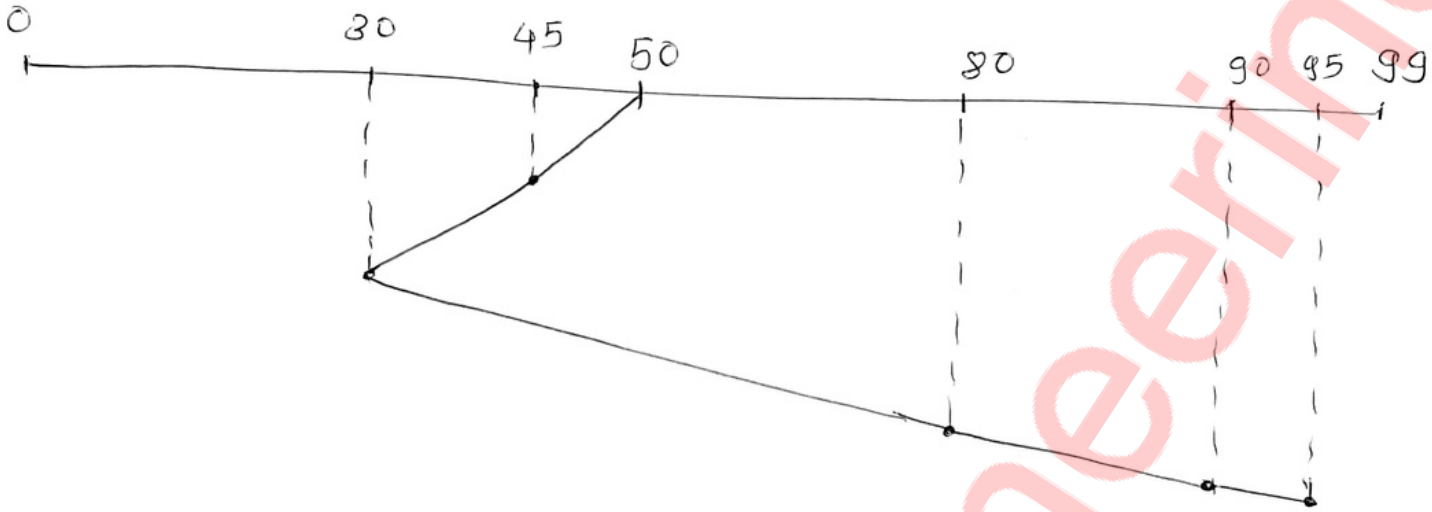
1) First Come First Serve [FCFS]

→ 100 Tracks [0-99]
Request Queue has Track Numbers:
{80, 30, 90, 95, 45}
Current track = 50.




⇒ (80-50) + (80-30) + (90-30) + (95-90) + (95-45)
⇒ 30 + 50 + 60 + 5 + 50 = 195

- Shortest Seek time First [SSTF]

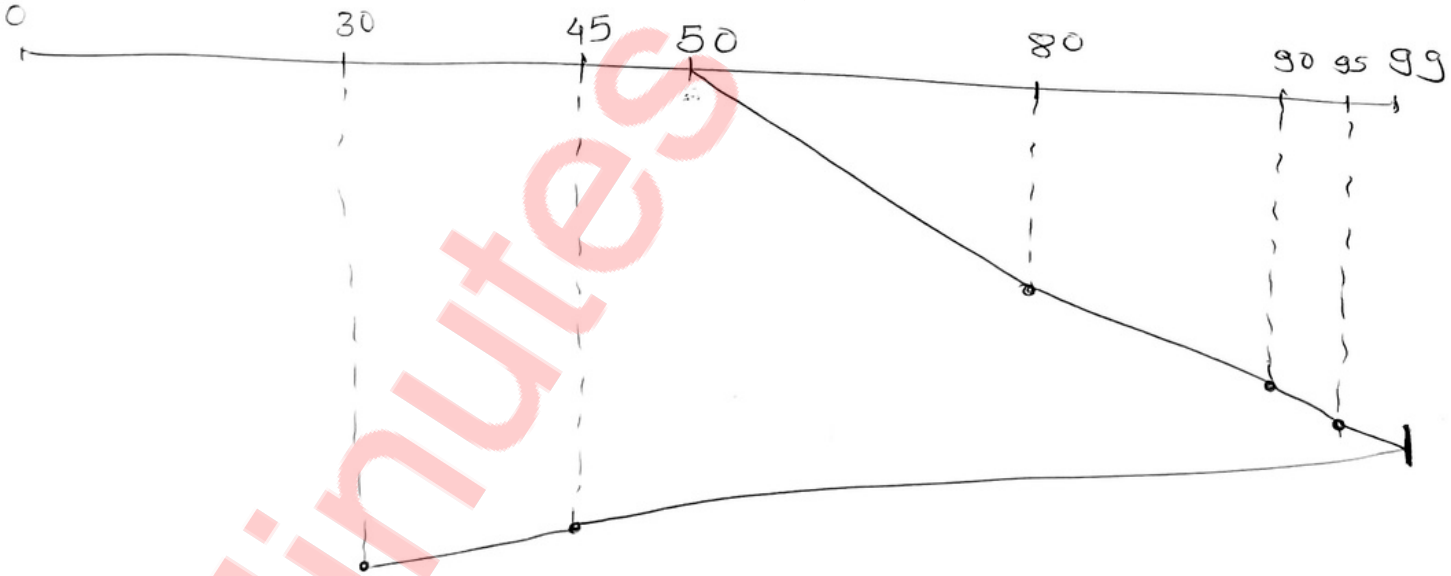



$\Rightarrow (50-30)+(95-30)$

$\Rightarrow 20+65$

$= 85$

- SCAN [Move to large value]

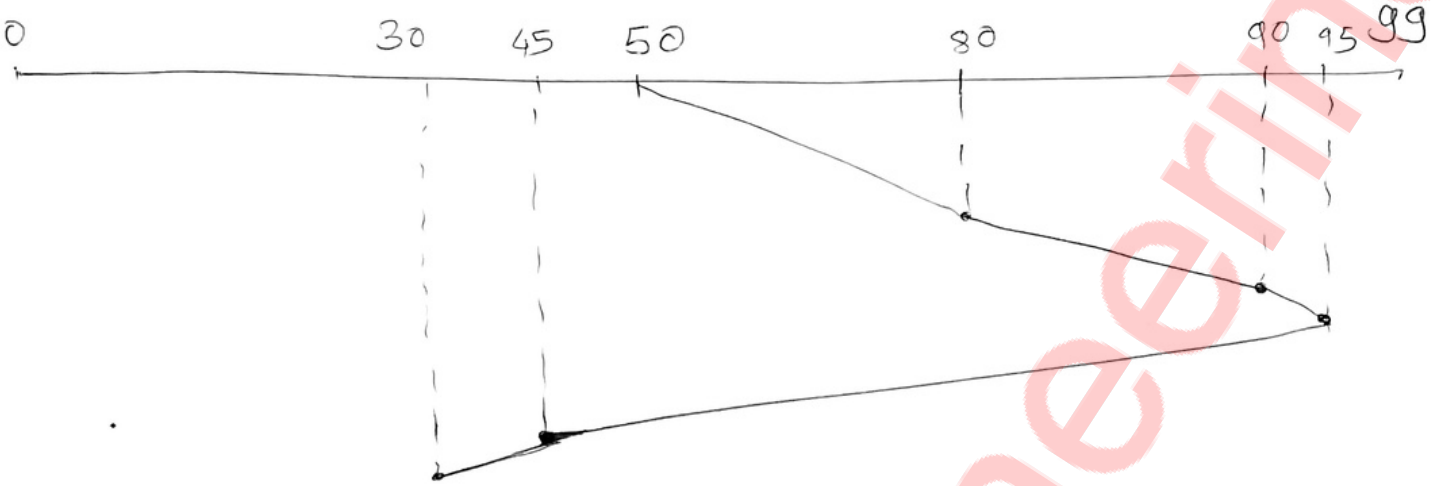



$\Rightarrow (99-50)+(99-30)$

$\Rightarrow 49+69$

$\Rightarrow 118$

- **LOOK** [Dont go to extreme ends]

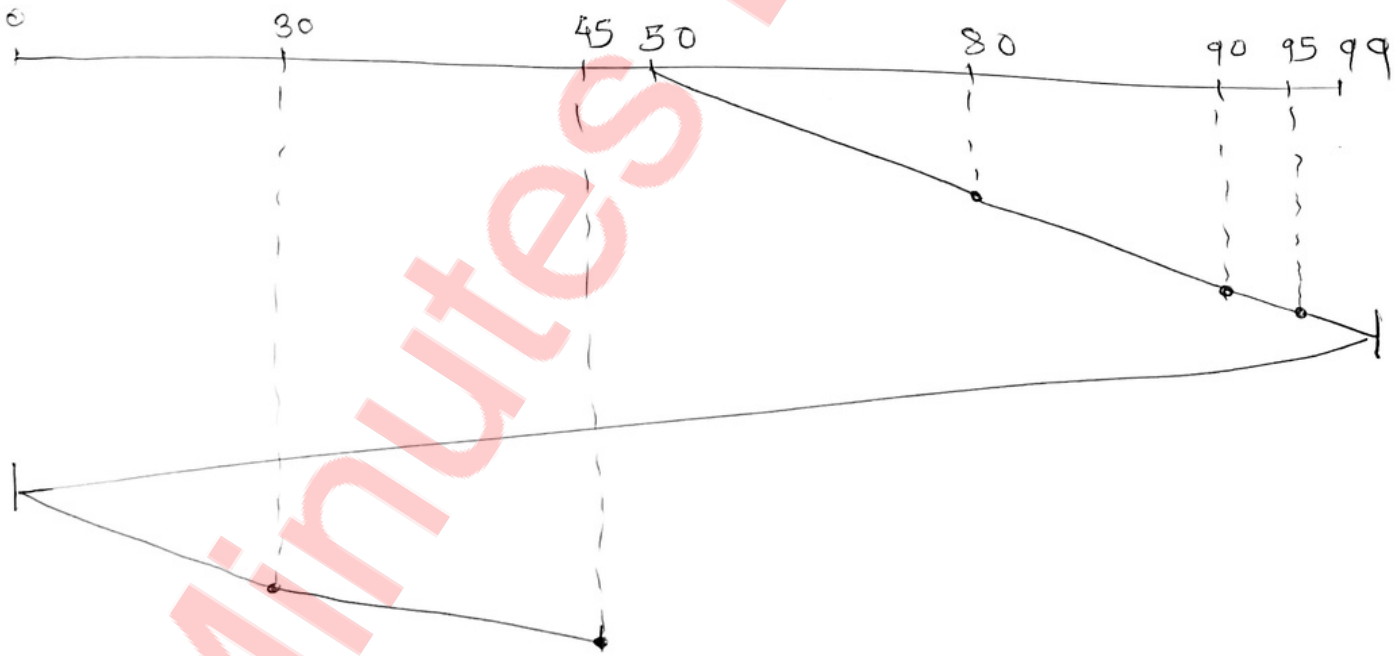



$\Rightarrow (95-50)+(95-30)$

$\Rightarrow 45+65$

$\Rightarrow 110$

- **C-SCAN** : [Move In large direction]

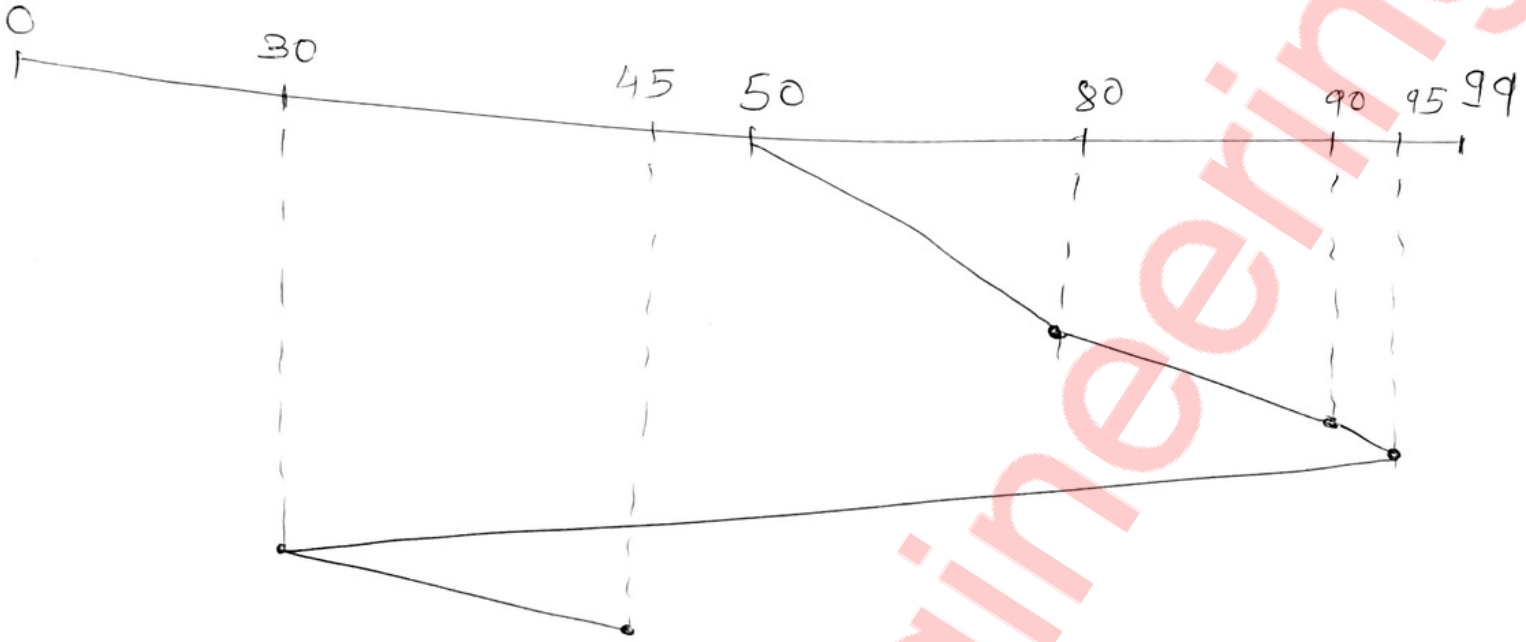



$\Rightarrow (99-50)+(99-0)+(45-0)$

$\Rightarrow (49+99+45)$

$\Rightarrow 193$

° C-LOOK [Move in large direction]

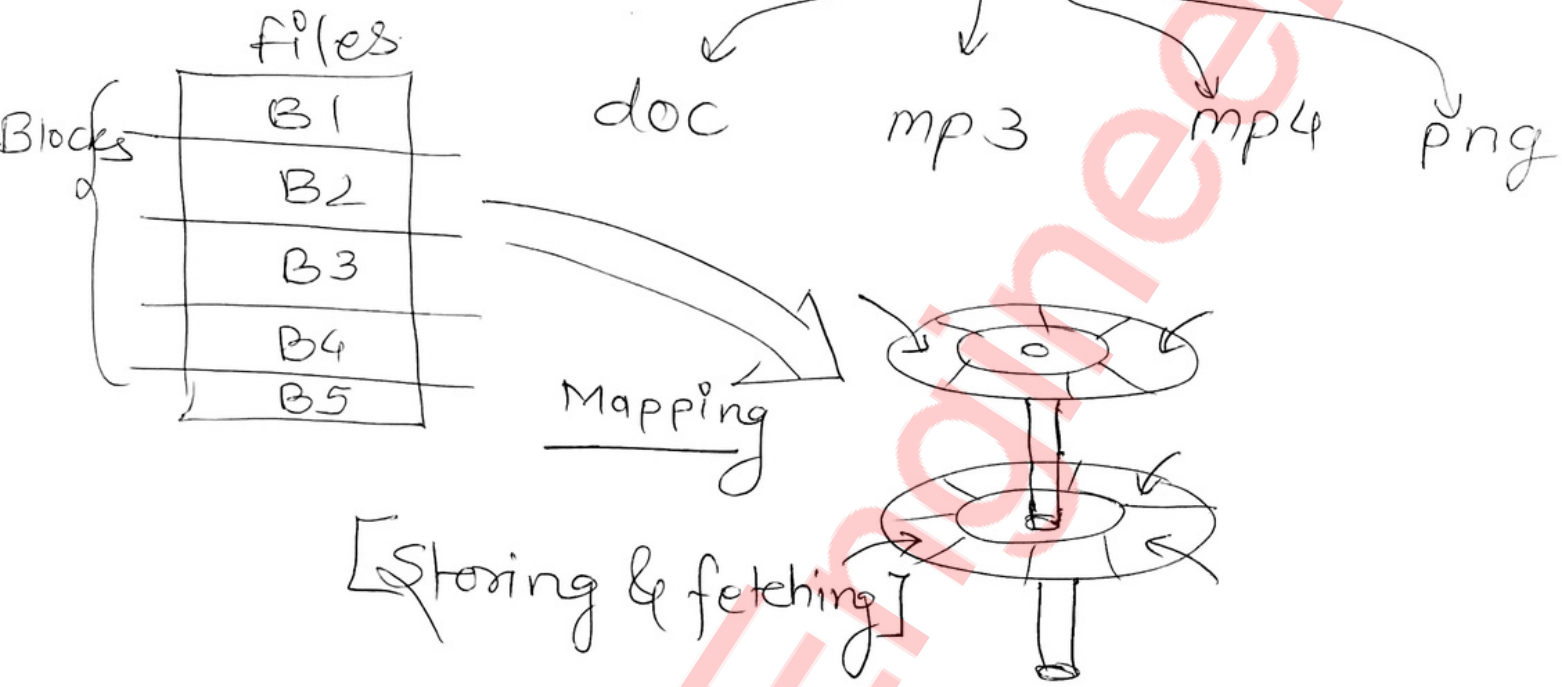



⇒ (95 − 50) + (99 − 30) + (45 − 30)

⇒ 45 + 69 + 15

⇒ 129

- File Systems : User operate/work/ Interact / modify the "Files"






[Storing & fetching]

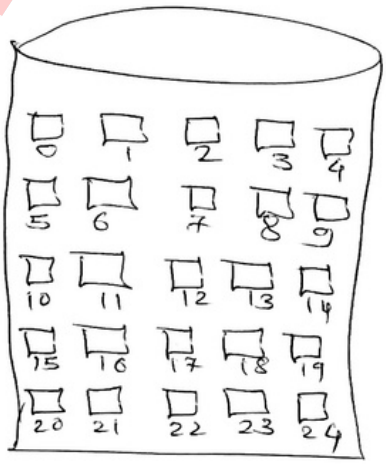
- File Operations
  - → Create
  - → Read
  - → write
  - → Truncate
  - → Delete
  - → Reposition

"Get Info" → ~~Attributes~~
- → Name
- → size
- → Type [.mp3, .mp4]
- → Create date
- → last modified date
- → Permission (Read only, R/W)

- Allocation Methods
  - Contiguous
  - Non Contiguous
    - Linked List
    - Indexed
- Disk utilization ↑
- Access ↑

folder/ Directory

| file | start | length |
|---|---|---|
| X | 0 | 5 |
| Y | 10 | 10 |
| Z | 20 | 5 |




→ Easy & fast

→ Sequential Access ✓

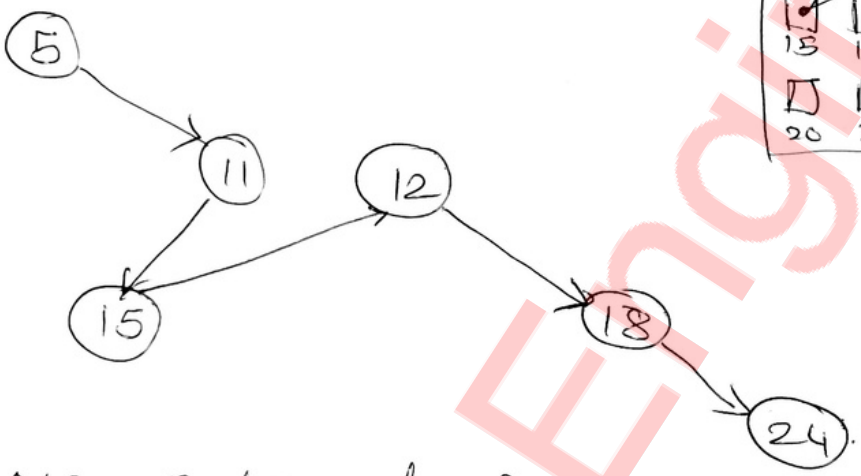
Direct Access ✓ [Block access] (direct)

- External fragmentation ✓

- Linked List → Non-Contiguous

Folder / Directory

| file | start | end |
|---|---|---|
| X | 5 | 24 |



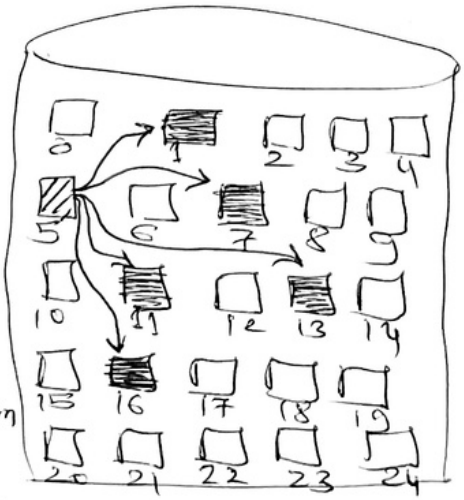

⇒ NO External fragmentation

⇒ Seek time ↑

- Indexed file → Non-Contiguous.

Folder / Directory

| file | Index Block |
|---|---|
| X | 5 |



⇒ NO External fragmentation

⇒ Random Access easy.