UNIT - II → (About process)

→ process synchronization
→ CPU scheduling
→ Real time scheduling
→ Deadlocks

} Topics covered

Process Synchronization

↳ co-ordinate & execute.
↳ No 2 process can have access to the same shared data & resources.

critical section/region accesses ↳ shared resource

Types of solutions to CS prg

Software
Peterson's solution

H/w
→ Synchronization H/w
— TSL Instruction
→ Compare & Swap Ins

MUTEX Locks
(spin-lock Software too)

Prg Lang. con
→ Semaphores

Peterson's solution

→ applicable for 2 processes

→ Turn & Flag [2]

- Turn → int variable → whose turn is it to access
- Flag [2] → boolean array → indicates if the process is ready to enter CS

flag [i] == true

[Pi is ready to enter.

```
do
{
    flag [i] = true;
    turn = j
    while (flag [j] && turn == j );

    // critical section

    flag [i] = false

    // remainder section

} while (true);
```

P_i	P_j
flag[i] = F P_i → enter CS flag[i] = T turn = j ↓ enters flag[i] = F	flag[j] = F enter P_j CS flag[j] = T turn = i ↓ enters flag[j] = F

Generally i → current process
j → next process

flag[i]	P_i (start)	turn	P_j	flag[j]
				F
F	flag[i] = T	j	(Now P_j wants CS)	
T	turn = j		while ( ) → True	
	while ( ) → False	j	↓ waits	
	↓ access CS			
F	flag[i] = F		while ( ) → False	
			↓ access CS	

→ Mutual exclusion
→ Progress
→ Bounded waiting

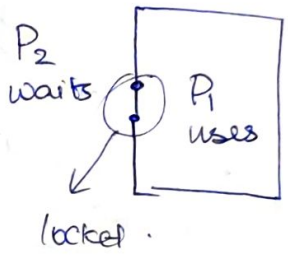
} Solution address 3 conditions to solve CS

Limitation

→ only 2 process
→ Buzy waiting

Hardware Solution

→ idea of locking



→ Uniprocessor
↳ Disable interrupts.
→ No preemption (switching).
→ Not scalable

→ Modern machines
→ Atomic → non-interruptible.
→ Ether test memory word & set value
→ Or swap contents of two memory words.

→ Test & Set Lock

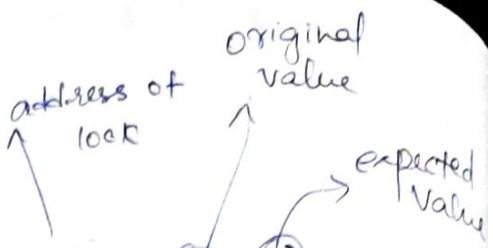
```
do {
    acquire lock
        critical section.
    release lock
        remainder section.
} while (True);
```

Eg:

```
bool TestAndSet (bool *target)
{
    bool rv = *target;
    *target = True
    return rv;
}
```

```
do {
while (TestAndSet (&lock));
        // do nothing.

    // CS

    lock = false

    // remainder section.
} while (True);
```

Compare & swap




```
do {
    while (compareAndSwap(&lock, 0, 1) != 0);
        // Does nothing

    // Critical Section

    lock = 0;

        // remainder section

} while (true);
```

```
compareAndSwap(*lock, orig, exp)
{
    if (*lock == orig)
    {
        *lock = exp;
        return false;
    }

    return true;
}
```

Bounded waiting & Mutual exclusion } with TSL.

```
do{
    waiting[i] = True;
    key = True
    while (waiting[i] && key)
        key = TestAndSet (&lock);
    waiting[i] = False
    // critical section
```

determines next turn:

```
    j = (i+1)%n
    while ((j!=i) && !waiting[j])
        j = (j+1)%n
```

```
    if (j==1)
        lock = False
    else
        waiting[j] = False
    // remainder section
} while (True);
```

Mutex Locks

→ acquire()
→ release()

```
acquire()
{
    while (!available)
        // wait
    available = false;
}
```

```
release()
{
    available = true;
}
```

```
do {
    acquire lock
        critical section
    release lock
        remainder section
}
```

Semaphores

→ No busy waiting
→ Semophore variable, Mutex variable } → S
→ operations → wait(), signal(), wakeup(), block()

```
do
{ wait(S)
    CS
  signal(S)
    remainder
} while (1);
```

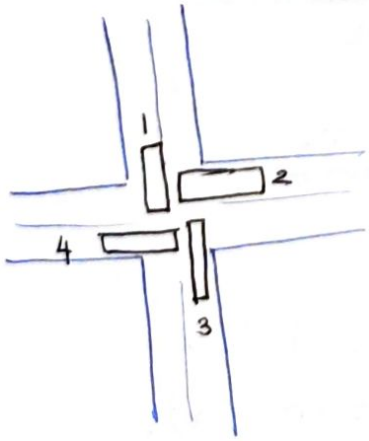
```
wait(S)
{ while(S<=0);
  S--;
}
```

```
signal(S)
{ S++;
}
```

block() — place the process in the waiting queue.

wakeup() — remove one of the processes in the waiting queue it in the ready queue.

Deadlock



Classical probs

- Must avoid deadlock

① ⇒ Bounded - Buffer
② ⇒ Readers & Writers
③ ⇒ Dining Philosophers (monitor)

} Producer / consumer Prob

---

① ⇒

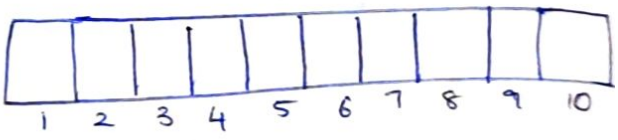
Shared data

Semaphore full
empty
mutex
} semaphore

Initial

full = 1
empty = n = 10
mutex = 1

Buffer



Shared data

Producer

```
do
{
    wait (empty);
    wait (mutex);

    signal (mutex);
    signal (full);
} while (1);
```

Consumer

```
do {
    wait (full);
    wait (mutex);

    signal (mutex);
    signal (empty);
} while (1);
```

⇒ Multiple readers can read
↓
access the shared data

⇒ Single writer can write
↓
access the shared data

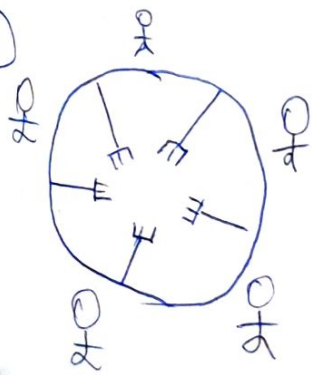
Reader

```
do {
wait (mutex);
readcount++;
if (readcount == 1)
    wait (rw-mutex);
signal (mutex)
// reading performed.
wait (mutex);
readcount --;
if (read count == 0)
    signal (rw-mutex);
signal (mutex);
} while (True);
```

Writer

```
do {
wait (rw-mutex);
// writing is
// performed.
signal (rw-mutex);
} while (True).
```

(3)



🍴 → chop stick

States
- thinking (not eating)
- eating (needs 2 chopstick)

requires

⇒ Eating 2 chopstick

left right

Philosopher

```
do {
    wait (chopstick [i]);
    wait (chopstick [(i+1) % 5]);

        // eat
```

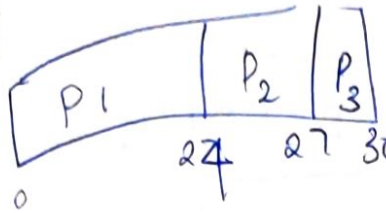
```
    signal (chopstick [i]);
    signal (chopstick [(i+1) % 5]);

        // think
```

```
} while (True)
```

FCFS
First come First Serve

P_1 burst time : 24
P_2 burst time : 3
P_3 burst time : 3

Gantt chart



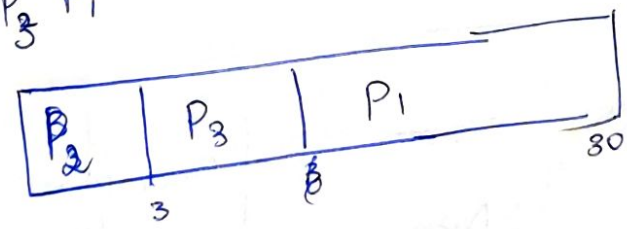
Waiting Time	Completion Time (Turnover time)	
0	24	P_1
24	27	P_2
27	30	P_3

Average $\frac{0+24+27}{3}$ $\frac{24+27+30}{3}$

$= 17$ $= 27$

P_2 P_3 P_1

Gnatt Chart



Average waiting time

$= \frac{0+3+6}{3}$

$= 3$

Average turnaround

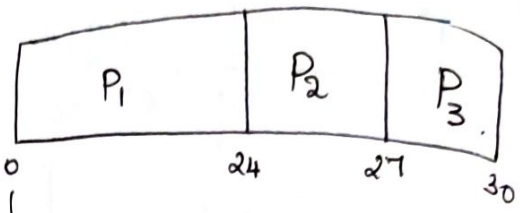
$= \frac{3+6+30}{3}$

$= 13$.

CPU scheduling

FCFS - First Come First Serve → Non preemptive

	Burst Time
P_1	24
P_2	3
P_3	3



→ Assume arrived time = 0ms

Average waiting time = $\frac{0 + 24 + 27}{3} = \frac{51}{3} = 17$ ms

Average turnaround time } = $\frac{24+27+30}{3} = 8+9+10 = 27$ ms

WT = 17ms
TT = 27ms

SJF - Shortest Job First

→ Non preemptive.

Preamptive
↳ switching ✓

Non-preamptive
↳ switching ✗

$WT = \frac{0+3+6}{3} = 3$ ms

$TT = \frac{3+6+30}{3} = 1+2+10 = 13$ ms

WT = 3 ms
TT = 13 ms

Preemptive of SJF is SRT

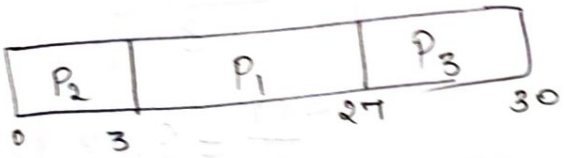
Shortest Run Time Scheduling

Turnaround Time
TT = WT + BT
Waiting Time + Burst Time

Priority

Priority		Burst time
2	P_1	24
1	P_2	3
3	P_3	3

Low number High priority



$WT = \frac{0+3+27}{3} = 10$ ms

$TT = \frac{3+27+30}{3} = 20$ ms

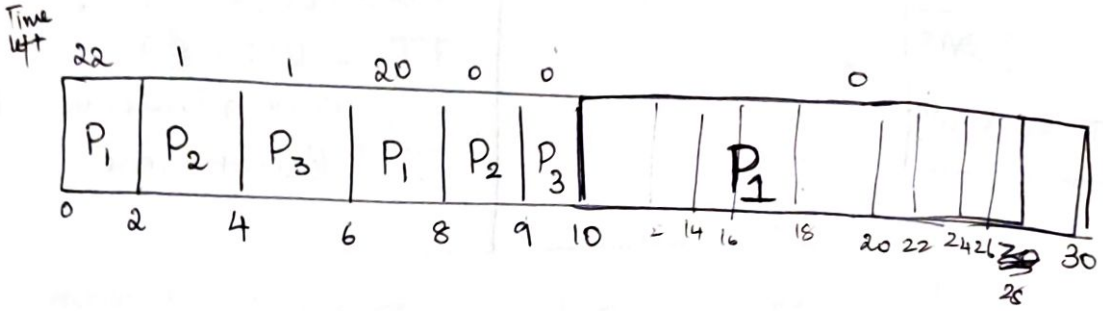
WT = 10 ms
TT = 20 ms

Round Robin → preemptive

preemption

time quantum = 2ms.

P_1	24
P_2	3
P_3	3



$$WT = \frac{0+2+4+6+8+9+10}{3}$$

$$= \frac{39}{3} = 13 ms.$$

$$TT = \frac{39+30}{3} = 23 ms.$$

$$WT = \frac{(2+(8-4)) + (4+(9-6)) + (0+(6-2)+(10-8))}{3}$$

$$= \frac{6+7+6}{3}$$

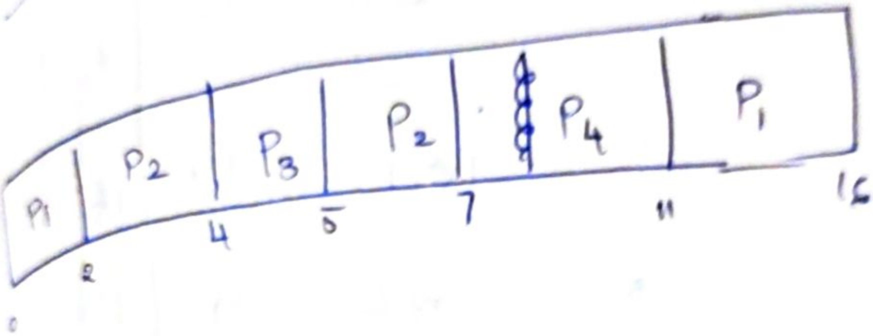
$$= \frac{19}{3} = 6.33 ms.$$

$$TT = \frac{((6+24)+(6+3)+(7+3))}{3}$$

$$= \frac{30+9+10}{3}$$

$$= 16.33 ms$$

	Arrival Time	Burst
P_1	0	7
P_2	2	4
P_3	4	1
P_4	5	4



WT = 1[(0+0+

PID	Priority	Arrival	Burst	Comp.	TT	WT
P1	2 (10to)	0	~~25~~ 4	25	25	21
P2	4	1	~~22~~ 2	22	21	19
P3	6	2	~~21~~ 3	21	19	16
P4	10	3	1~~0~~ 5	12	9	4
P5	8	4	1~~5~~ 1	19	15	14
P6	12 (High)	5	~~9~~3 4	9	4	0
P7	9	6	~~1~~5 6	18	12	6

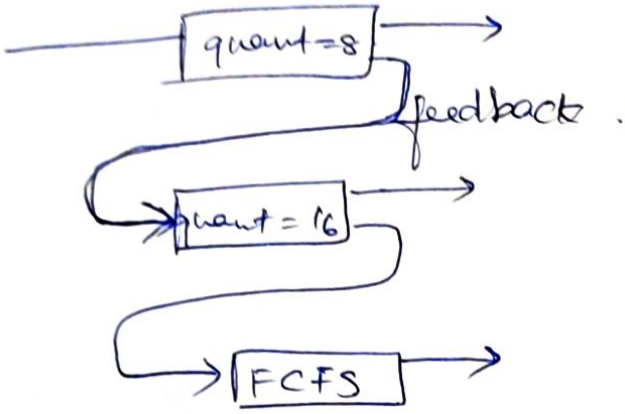
Multi level Queue Scheduling

80% CPU → foreground process.

20% CPU → background process.

Multi level Feedback Queue

Every queue → unique time quantum.



Real Time CPU Scheduling

hard

→ strict requirement

Soft

→ relevant.

Latency (delay)

Interrupt

Dispatch.

Rate Monotonic Scheduling

short time = high priority
long periods = low priority

burst
- periods

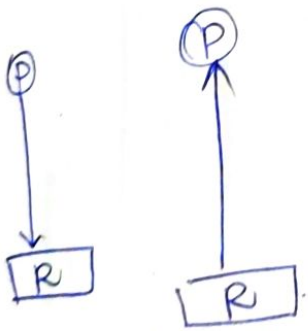
Each process
↳ unique burst time

Instruction
I/O → switching lot
CPU → switching less

Earliest Deadline First
(EDF)

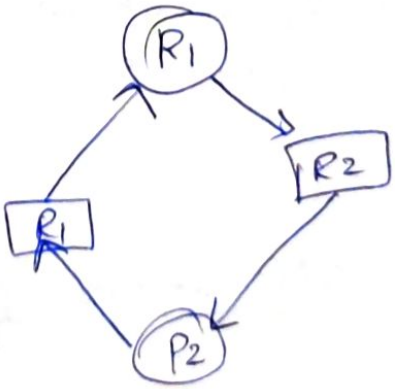
- Not periodic
Not equal burst time per cycle.

Deadlock



Requested Allocated.

Each process waits for another for resource.



Request

↙ ↘

available = 1 → allocate

else → wait

Use → User resourse

Release → release the resour

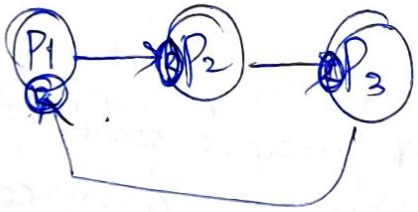
Characterization (4)

→ Mutual exclusion

→ Hold & wait ⟹ P → R_1 Use (Hold), R_2, R_3 } wait

→ No premption → won't release or switch without process completion

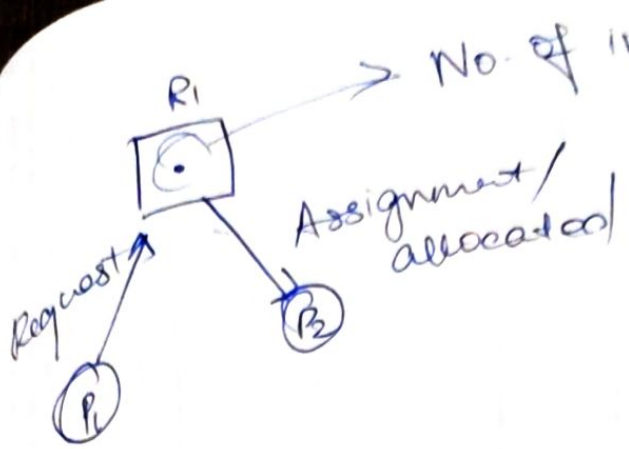
→ Circular wait.

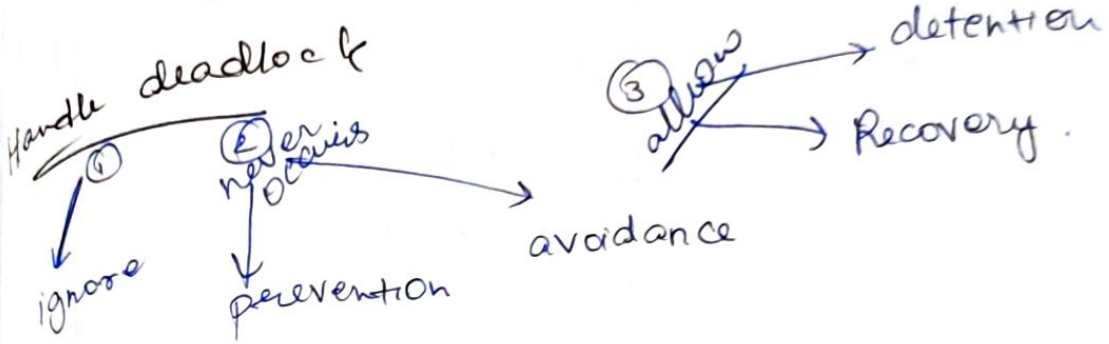


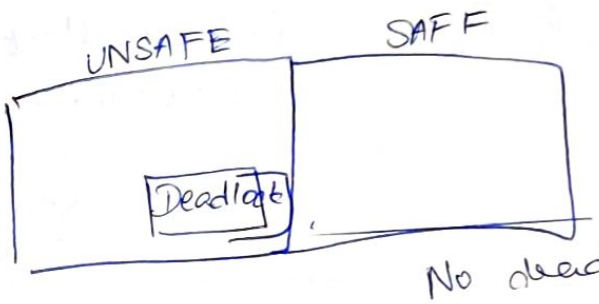
Request for a resource held by another process.

Resource allocation graph.

Request edge $P_i \rightarrow R_j$ | Assignmat edge $R_j \rightarrow P_i$







⇒ Process avoid deadlock/ only with luck

No deadlock
⇒ OS can avoid it.
⇒ doesn't occur.
⇒ there exist a safe sequence.

Deadlock Avoidance

① Banker's Algo

③ Available →

R1	R2	R3
5	4	3

Need

② Max →

	R1	R2	R3
P1	5	7	3
P2	5	6	2

① Allocation →

	R1	R2	R3
P1	1	0	1
P2	2	3	2

Safe sequence : P_1 P_2 (no deadlock in this sequence execution)

Non allocated resources

	Allocation Matrix				Max Matrix				Available Matrix			
	A	B	C	D	A	B	C	D	A	B	C	D
P1	0	1	1	0	0	2	1	0	1	5	2	0
P2	1	2	3	1	1	6	5	2	1	6	3	0
P3	1	3	6	5	2	3	6	6				
P4	0	6	3	2	0	6	5	2				
P5	0	0	1	4	0	6	5	6				
Total	2	12	14	12								

Need = Max − Allocation

$$= \begin{bmatrix} 0 & 2 & 1 & 0 \\ 1 & 6 & 5 & 2 \\ 1 & 3 & 6 & 6 \\ 2 & 6 & 5 & 2 \\ 0 & 6 & 5 & 6 \\ 0 & 6 & & \end{bmatrix} - \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 2 & 3 & 1 \\ 1 & 3 & 6 & 5 \\ 0 & 6 & 3 & 2 \\ 0 & 0 & 1 & 4 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 4 & 2 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 2 & 0 \\ 0 & 6 & 4 & 2 \end{bmatrix}$$

Need

	A	B	C	D
P1	0	1	0	0
P2	0	4	2	1
P3	1	0	0	1
P4	0	0	2	0
P5	0	6	4	2

Condition

(2) Need$_i$ <= Work;

Available

$P_1 \rightarrow (0, 1, 0, 0) <= (1, 5, 2, 0)$ ✓

~~$P_2 \rightarrow (0, 4, 2, 1) <= (1, 5, 2, 0)$~~ ✗

$P_1 \Rightarrow$ Work = Work + Allocation

Work = (1, 5, 2, 0) + (0, 1, 1, 0)

Work = (1, 6, 3, 0)

$P_2 \rightarrow (0, 4, 2, 1) <= (1, 6, 3, 0)$ ✓

$P_2 \Rightarrow$ Work = Work + Allocation

Work = (1, 6, 3, 0) + (1, 2, 3, 1)

Work = (2, 8, 6, 1)

$P_3 \Rightarrow$ ~~Work = Work + A~~

$(1, 0, 0, 1) <= (2, 8, 6, 1)$ ✓

$P_3 \Rightarrow$ Work = Work + Allocation

= (2, 8, 6, 1) + (1, 3, 6, 5)

= (3, 11, 12, 6)

$\Rightarrow (1, 0, 0, 1) <= (1, 6, 3, 0)$

P3

$\Rightarrow (0, 0, 2, 0) <= (1, 6, 3, 0)$ ✓

P4

P1	P4	P5	P2	P3

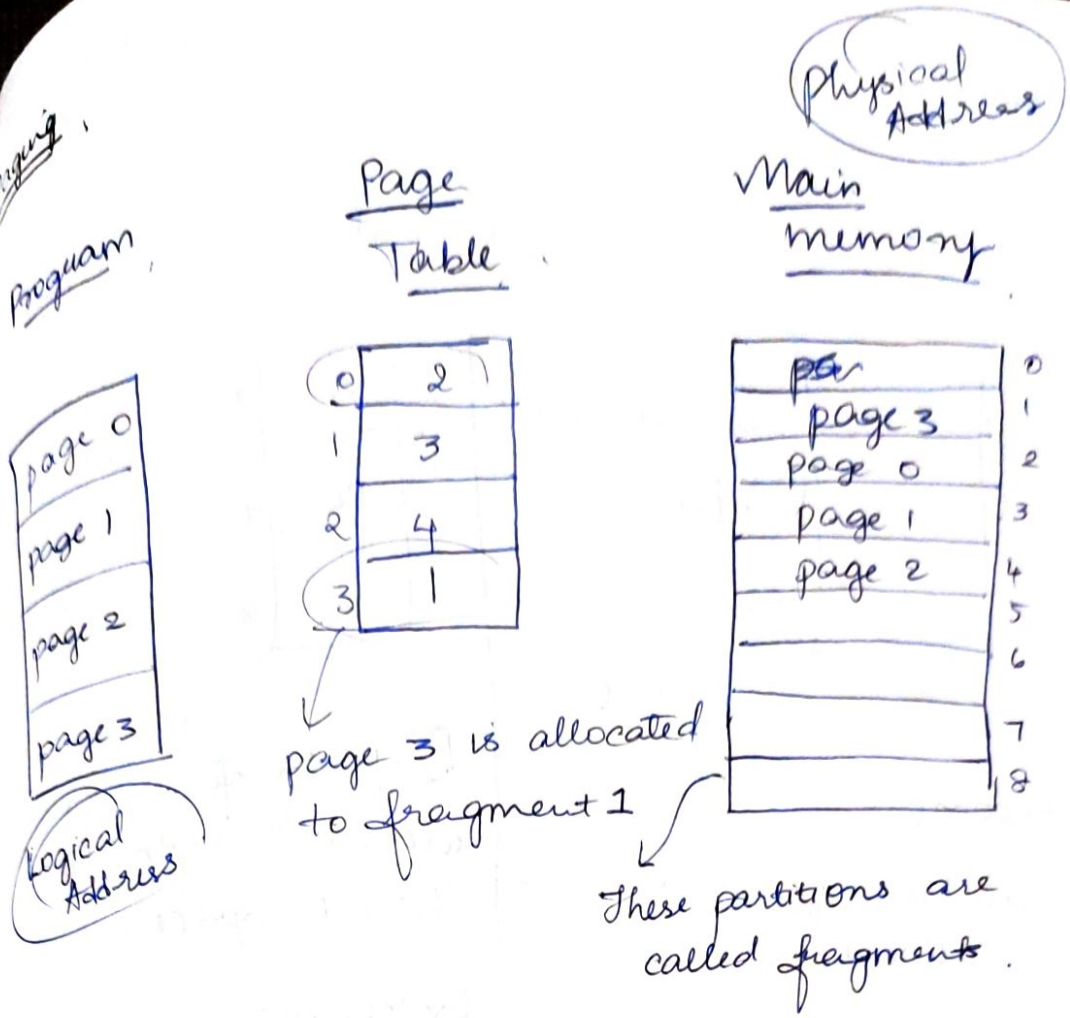
(1)

> vi fi.sh. # Create file.

Save & quit Esc :wq.

> sh fi.sh # Execute.

Paging:



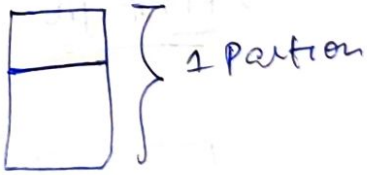
In segmentation, segments are of variable size.
In paging, segments are of equal size.

paged-segments ⇒ In a page there are multiple pages.

Allocation:

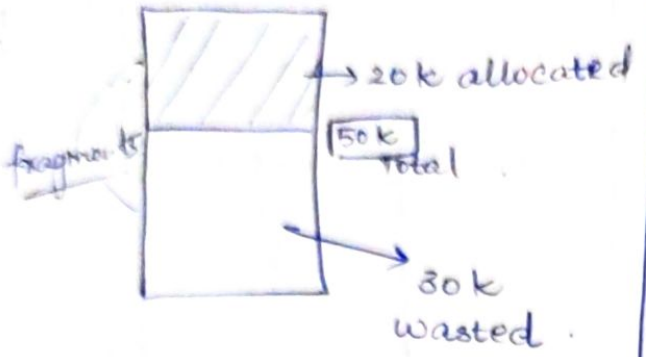
- Contiguous
  - Fixed
    - → Limit size
    - → Internal fragmentation
    - → No multiprog
    - → Not necessary to be equal size
  - Variable.

Fragmentation



Fragmentation

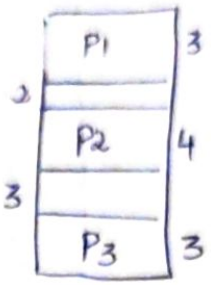
Internal



OVERCOME:

→ allow dynamic allocation

External



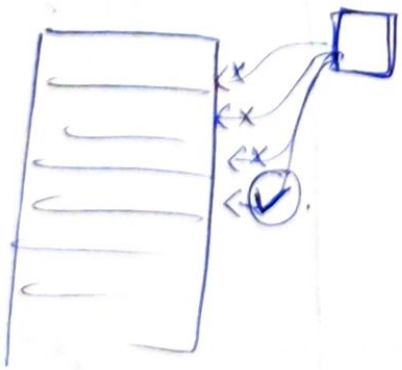
P_4 of size 4 cannot be allocated even though there is space.

OVERCOME:

Move the process to one end & have free space to the other end

Partition Allocation Algorithm

→ First fit

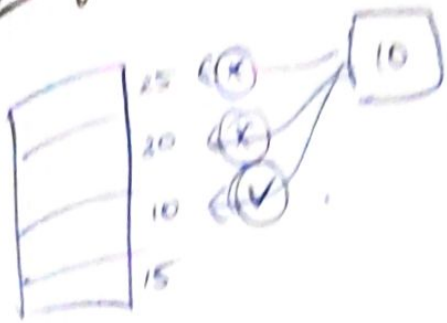


Variable Part. Demerits

→ complex memory allocation

→ External fragmentation

→ Best fit



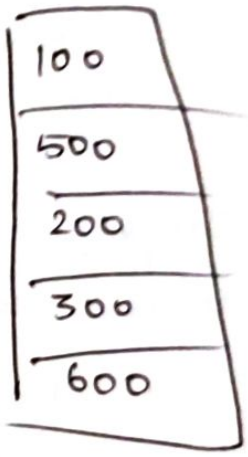
finds the best fit

→ worst fit

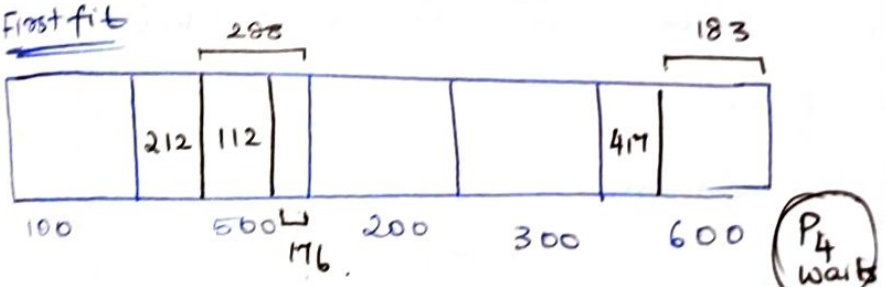
finds the biggest hole to fit

① Problem

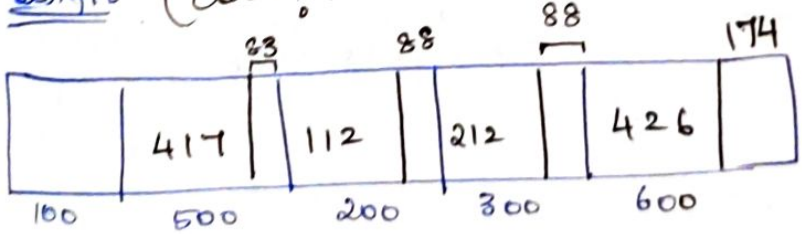
212 P_1 417 P_2 112 P_3 426 P_4 ?



First fit

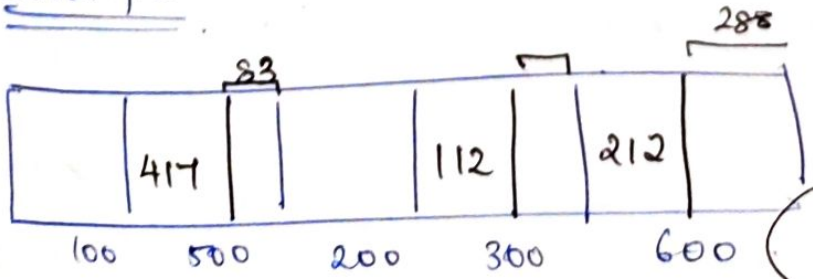


Best fit (cool!)



600 − 212 = 288
600 − 417 = 183
288 − 112 = 176

Worst fit



600 − 212 = 288

Non Contiguous

Paging — Segmentation.

PAGING:

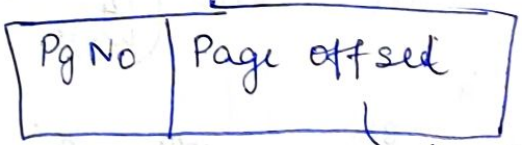
→ virtual physical address or

Logical address (CPU generates) → occupies virtual Address Space.

Physical address (Address in main memory) → occupies physical address space.

The physical address space is conceptually divided into frames.

Page size = frame.

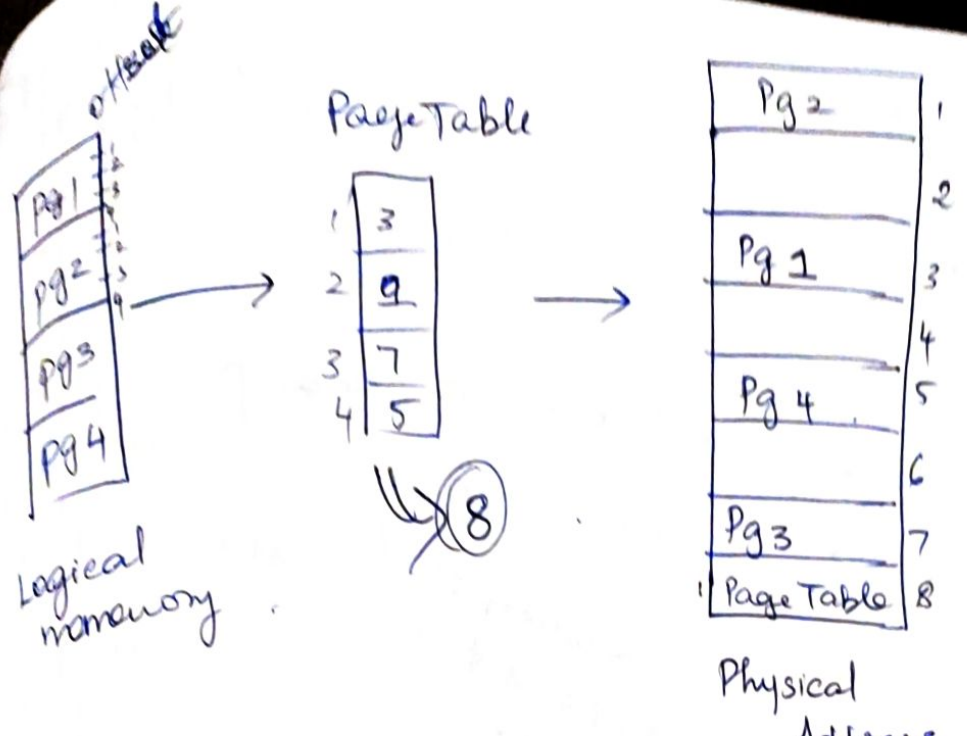


→ refers size.

LOOK-UP

→ Pg No / offset.

→ Generate number.



free frame list } 2 | 4 | 6 | 8

PTBR → Base Register = 8

PTLR → Length Register = 1

PT
(page Table

Problem

→ Each reference requires 2 memory access.
↓ overcome by special hardware (cache).
↓
associative memory (or) Translation look-aside Buffer

Effective Access Time (EAT) $= (1+\varepsilon)\alpha + (2+\varepsilon)(1-\alpha$

$= 2 + \varepsilon - \alpha$

ε — Associative Lookup.

α — Hit ratio.

↙ possibility that the pg is there in the TLB.

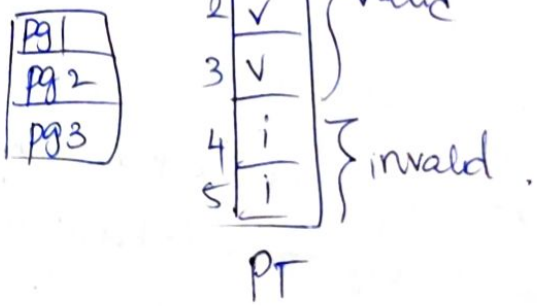
Memory Protection

(1bit)

→ Local memory b/w process

Read Write

Valid (legal) Invalid (illegal).



Page Fault

↳ Not correctly loaded.

⇒ Check valid memory

↙ Yes → Good

↘ No → Terminate.

UNIT 2

Process Synchronization

↳ coordinating execution so no two process can have access to the same shared data and resources

Critical Section

↳ shared resource.

↳ solutions.

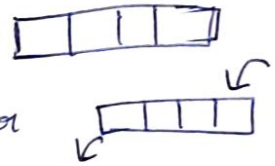
- Software Solution
  - ↳ Peterson's Solution
- Hardware Solutions
  - ↳ TSL Instruction
  - ↳ Compare & Swap Instruction
- MUTEX Locks
  - ↓ Spin lock S/W Tool
- Programm language construc
  - ↓ Sema -phore

S/W Solution

→ applicable for 2 process.

→ Classical Problems.

- ↳ Bounded Buffer.
- ↳ Producer - consumer
- ↳ Dining Philosophers.



Peterson's solution

→ algorithm deals with 2 variables

→ turn (int) and flag (flag).

flag → True : Process is ready to enter.


```
{
    flag [i] = true;
    turn = j;
    while ( flag[j] && turn == j );
                [Critical section]
    flag[i] = false;
                [remainder section]
} while (true);
```

waiting = busy waiting.

Conditions addressed

(i) Mutual exclusion
(ii) Progress
(iii) Bounded waiting

Limitations

→ 2 process
→ Busy waiting.

H/W solution

→ uses h/w
→ idea : locking
→ uniprocessors.
    ↳ No premption.
→ Not Scalable.
→ Modern machines have atomic h/w.

① TSL

- Test and Set Lock

```
do
{
    acquire lock
        critical section
    release lock
        remainder section
} while (True);
```

- No preemption.

```
bool TestAndSet (bool *target)
{
    bool rv = *target;
    *target = True;
    return rv;
}.
```

Improvised version

```
bool waiting[n]
bool lock
```
} initially both are set to false

do {
waiting [i] = True;
key = True;
while (waiting [i] && key)
key = TestAndSet (&lock);
waiting [i] = False;

CS

j = (i+1) % n;
while ((j != i) && !waiting [j])
j = (j+1) % n;

if (i == j)
lock = False;
else
waiting [j] = False

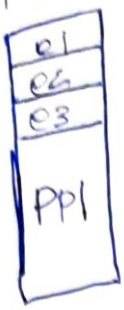
RS

} while (True);

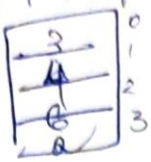
MUTEX LOCKS

Shared Pages

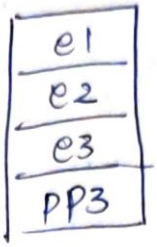
Process 1



PT for process 1

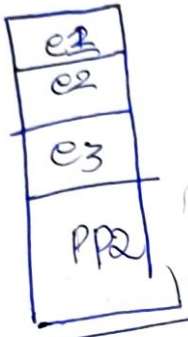


Process 3

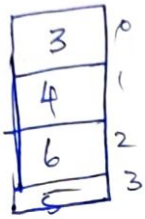


PT – P3

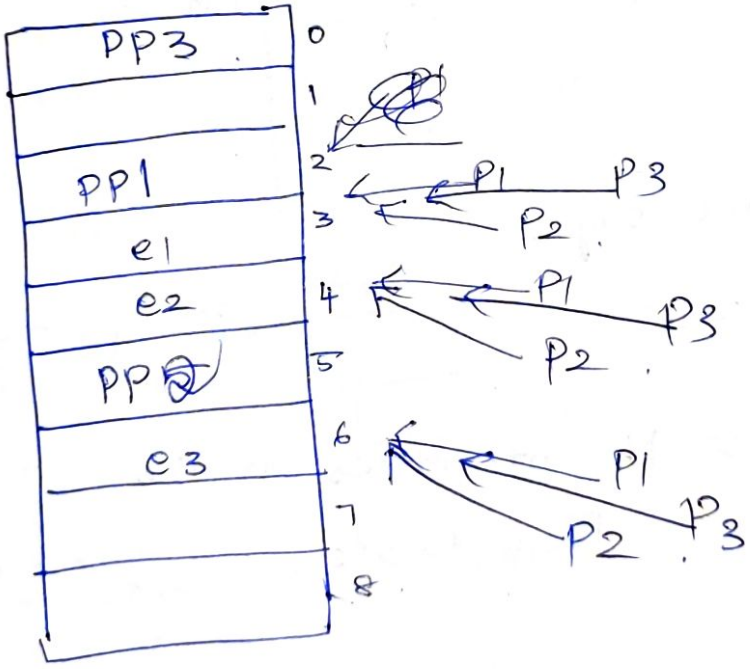
Process 2



PT – P2

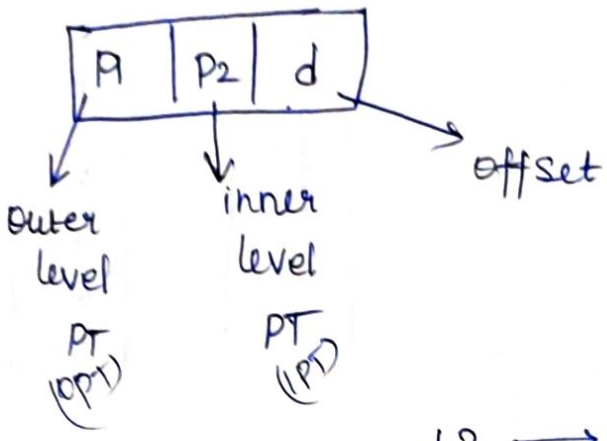


Main memory



Structure of Page Table

1. Two level PT



2^{32} bit address

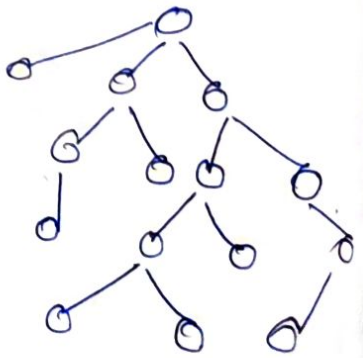
- 12 → PI size.
- 10 → PI size.
- 10 → offset

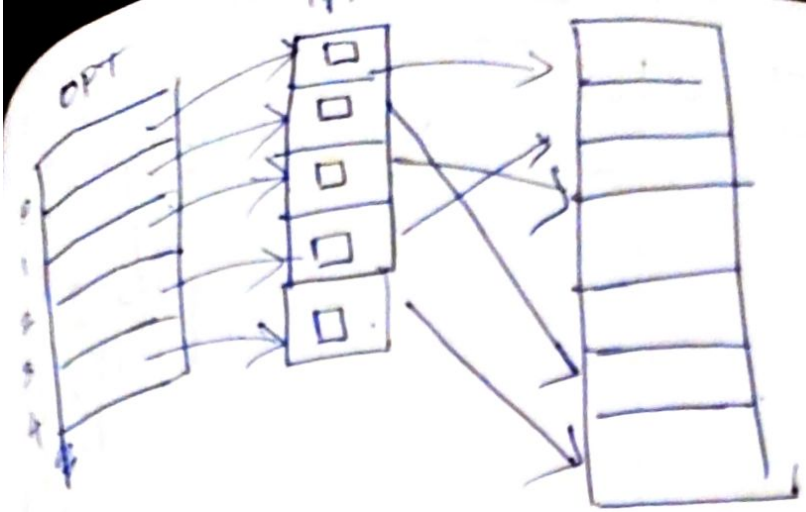
2^{32} → Main memory.

2^{12} → Page size.

$\frac{2^{32}}{2^{12}} = 2^{20}$ ⇒ No of entries in the In the Page table.

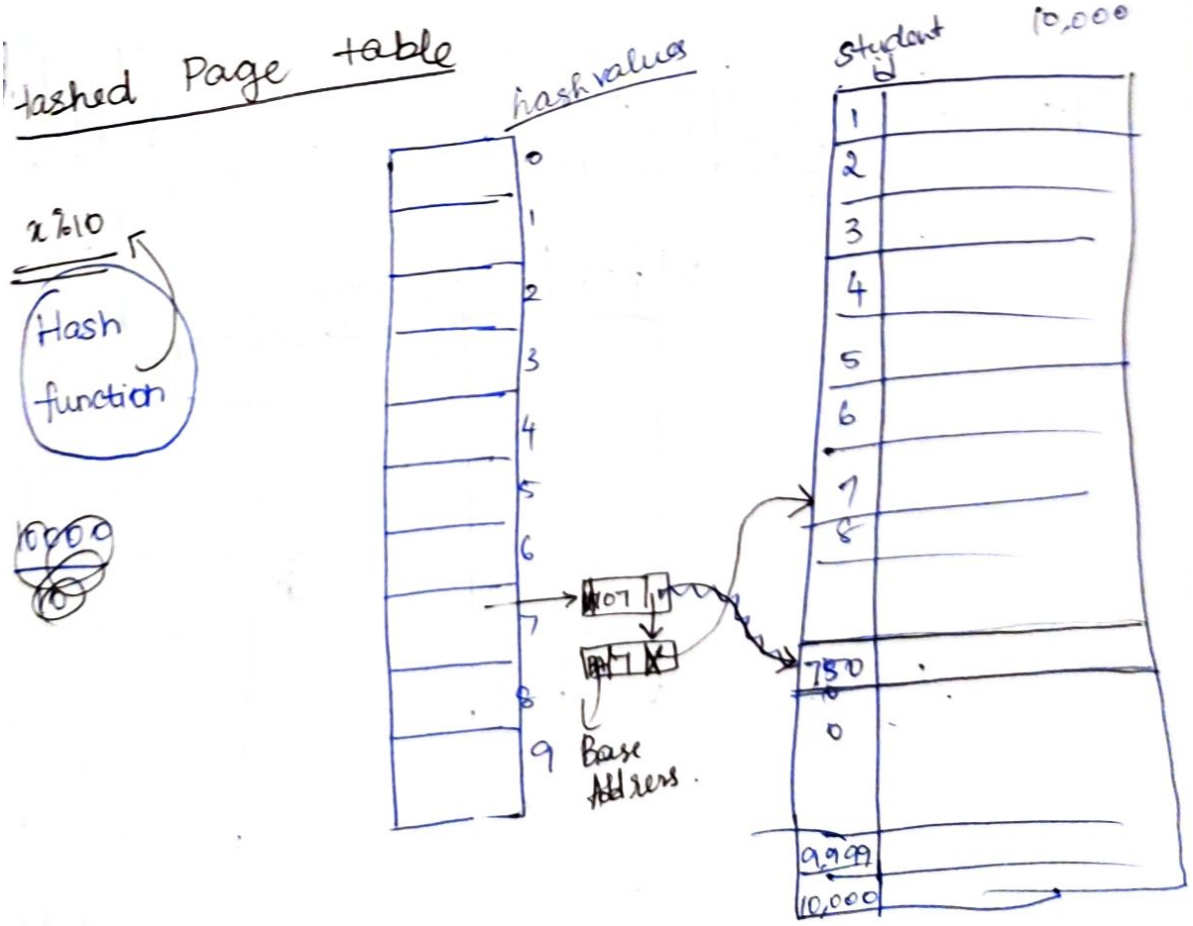
2. Hierarchical

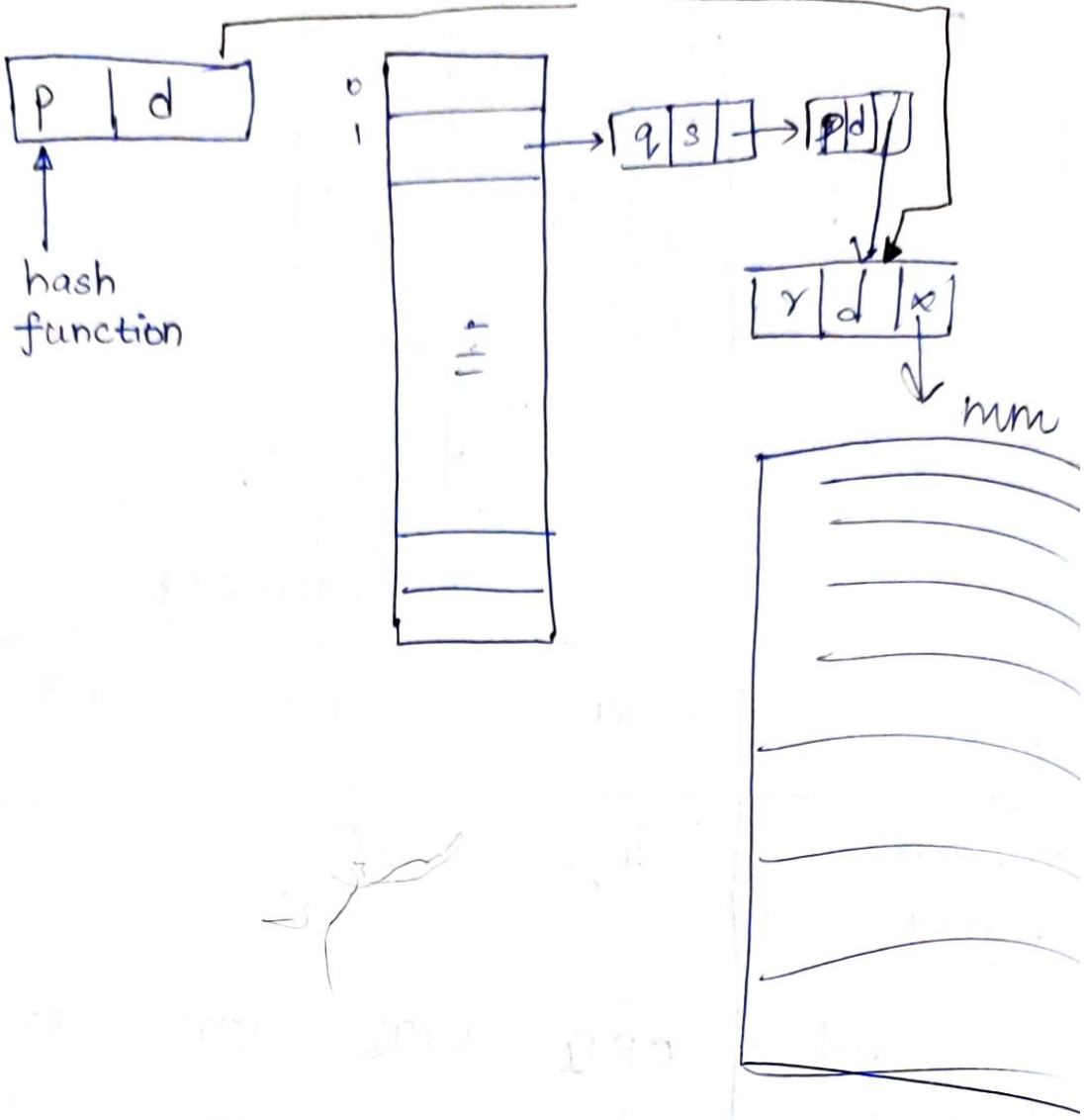




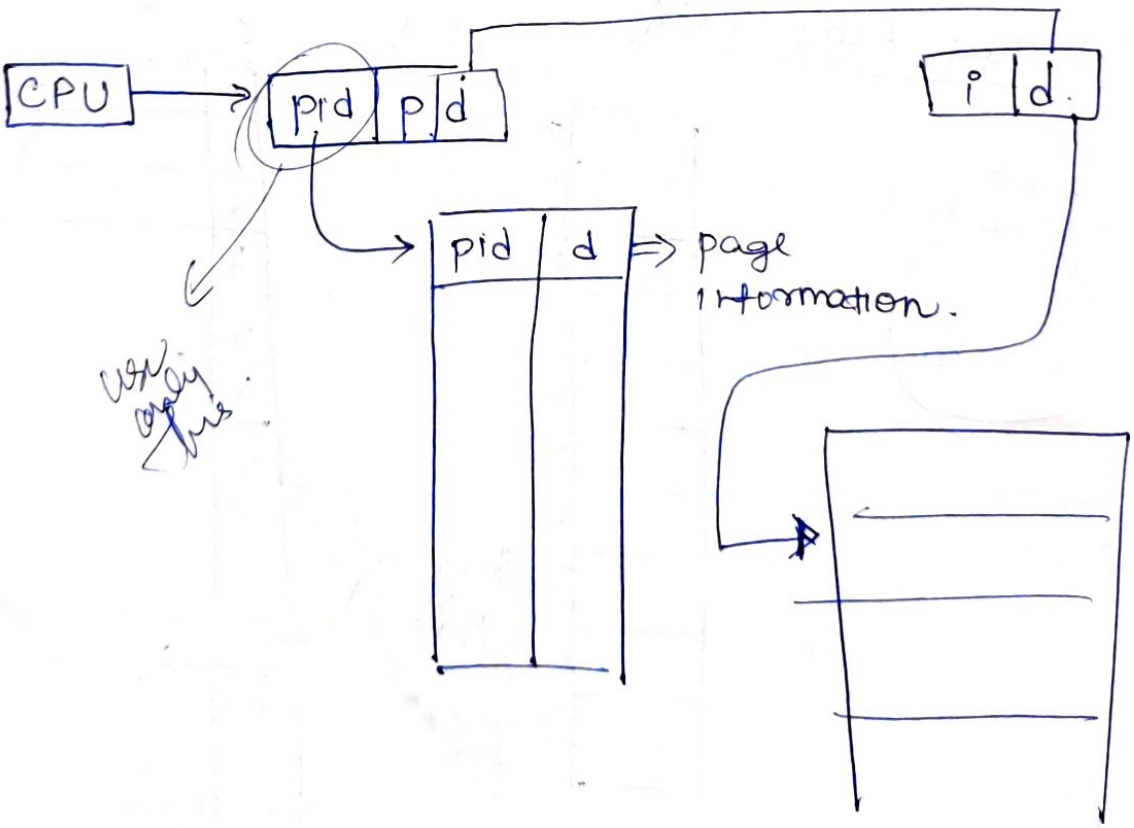
	OPT	IPT	offset
	P1	P2	P3
2^{64}	42	10	12
Two level Paging			
Three level Paging	OPT_1 32	OPT_2 10	IPT 10	offset 12

Hashed Page table



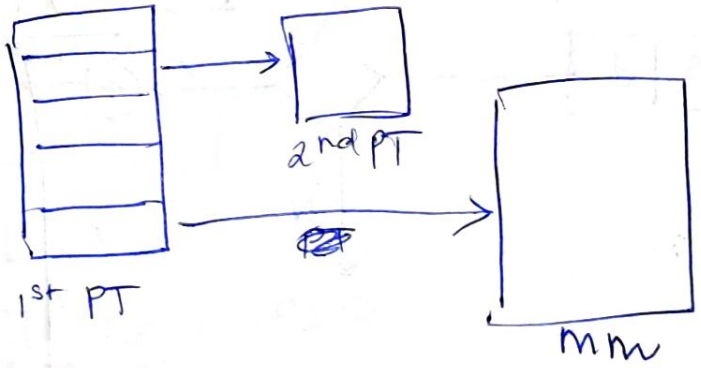


Inverted Page Table



ARM

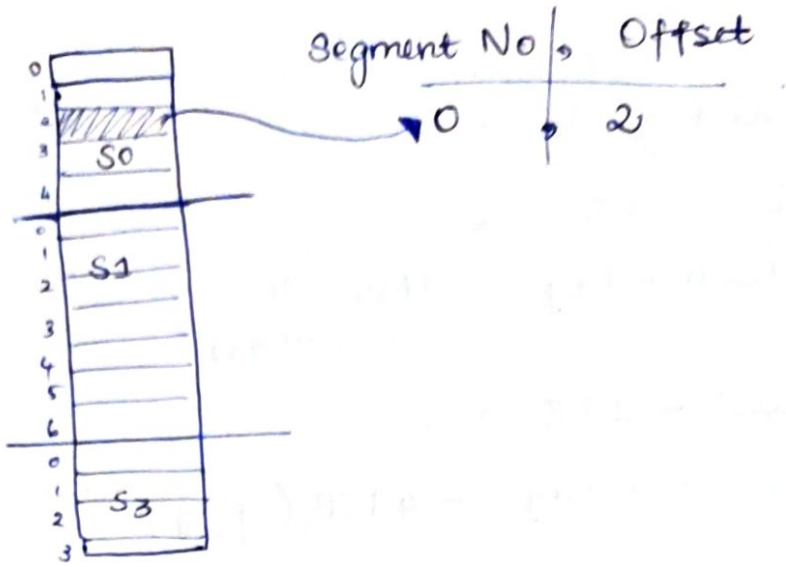
- Uses 2 level Page Table.
- First level – 16 KB size
  each entry – 1 MB size (sections)
- Second Level – 1 KB size
  each entry – 4 KB (pages)
- TLB – 16 MB sections,
  64 KB pages
- ARM V4, V5 ⇒ sub pages
- 4 KB page ⇒ 1 KB sub pages
- 64 page ⇒ 16 KB subpages
- 2 independent pagetables – 16 KB, 64 KB



Segmentation

Paging → pages are fixed equal size. internal fragmentation.

Segmentation → segment, variable size.



STBR – Segment Table Base Register

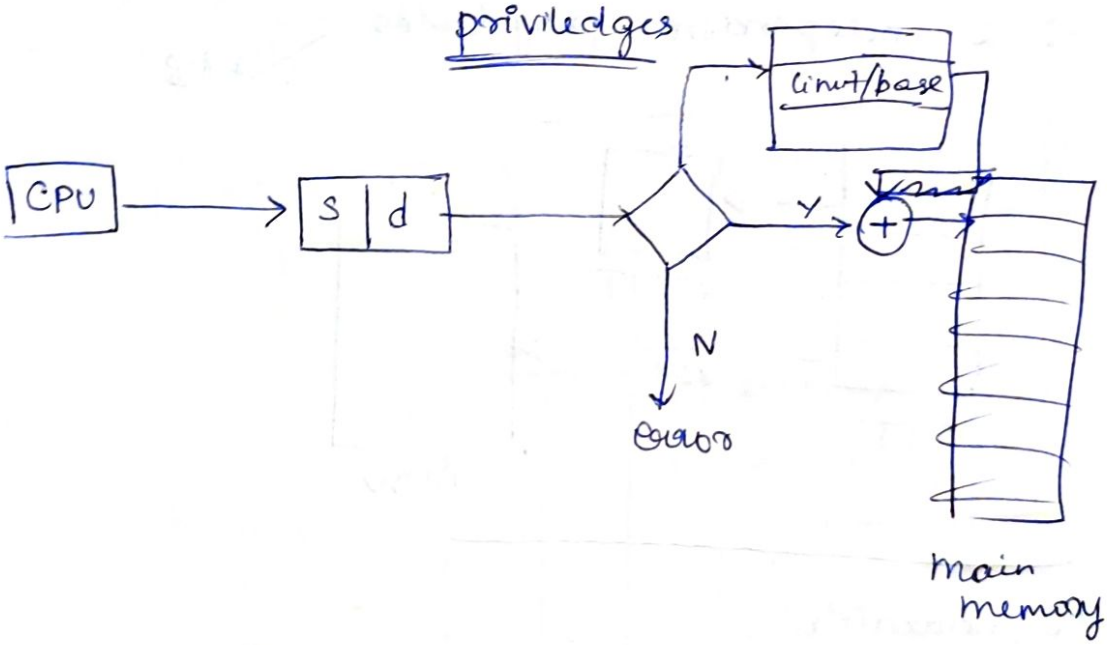
STLR – Segment Table Length Register.

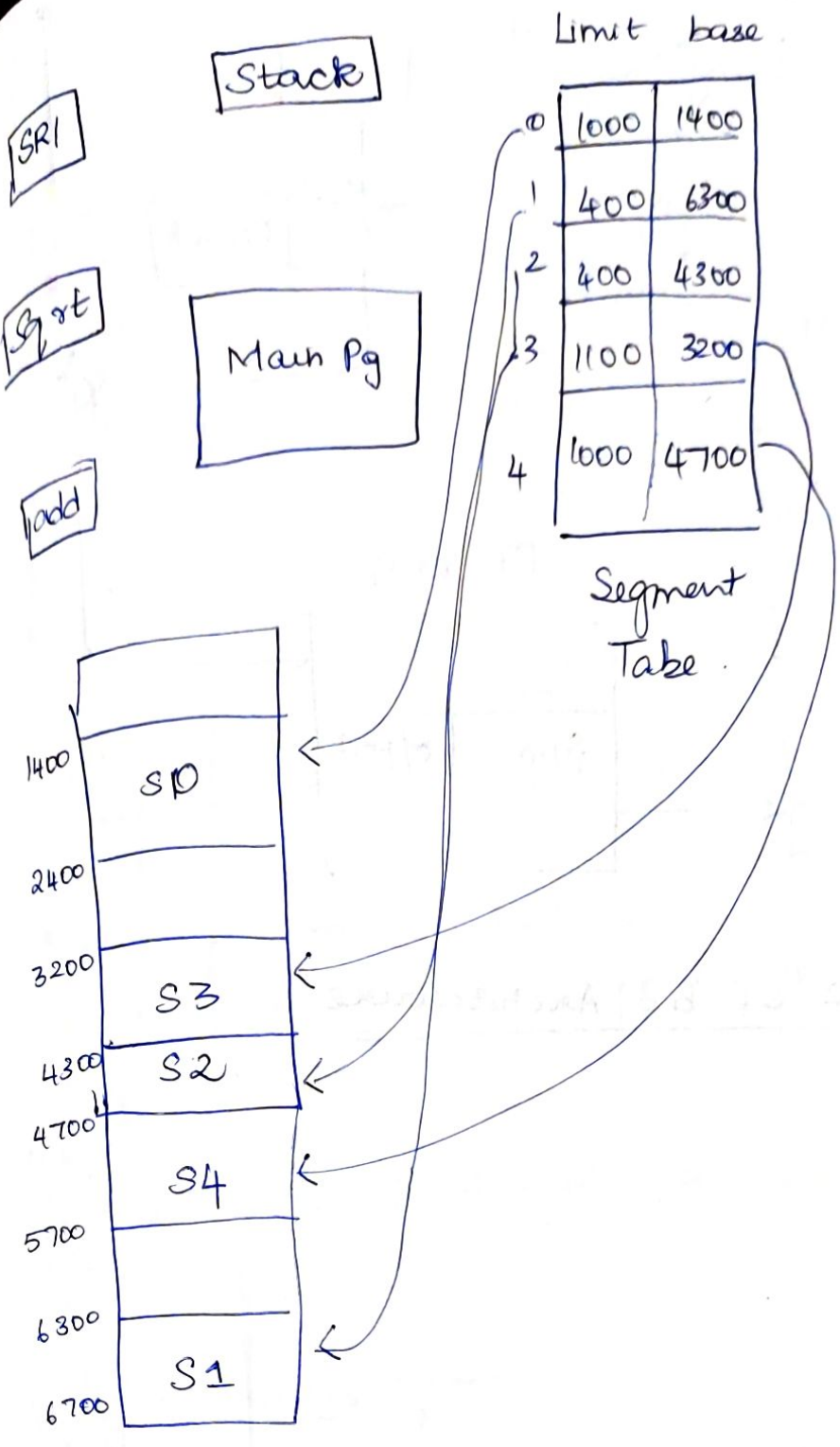
* Protection bit

⇒ 0 – Illegal segment

⇒ 1 – protected ⇒ r/w/e (read / write / execute)

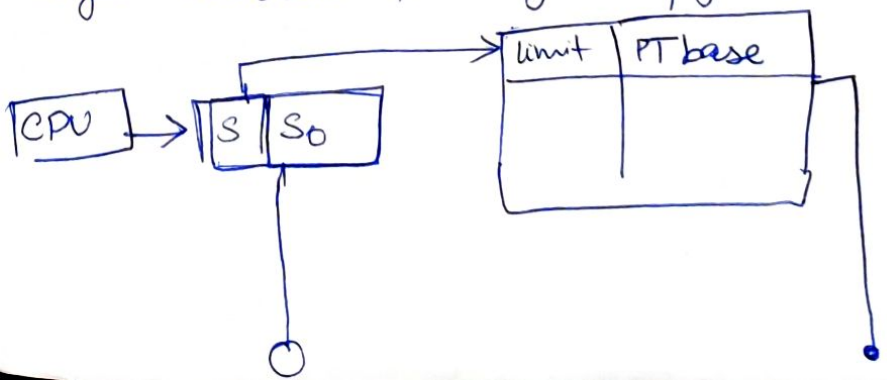
privileges

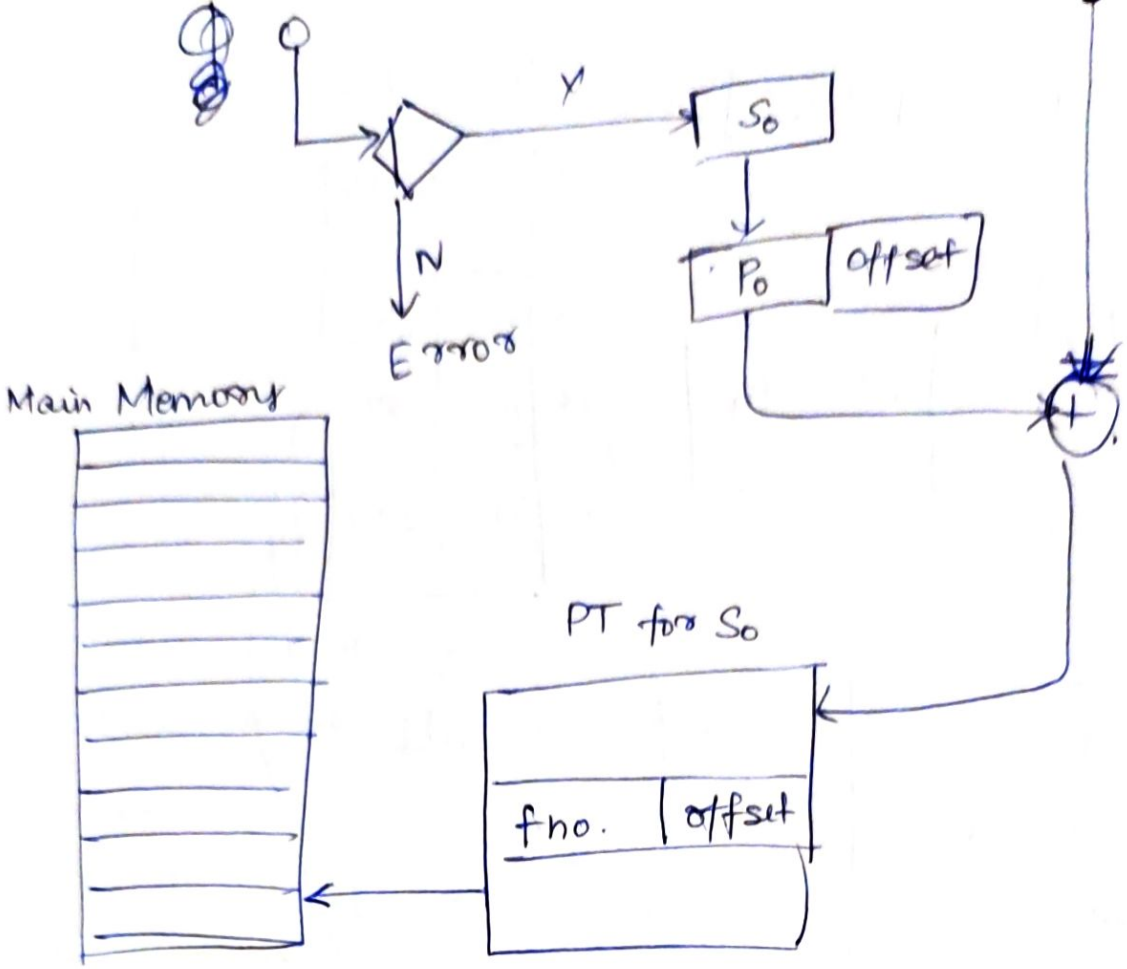




Paged Segmentation

Logical address ⇒ segno, pgno, offset





Intel 32 & 64 bit Architecture

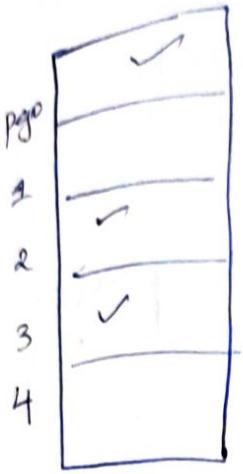
↓

Pentium.

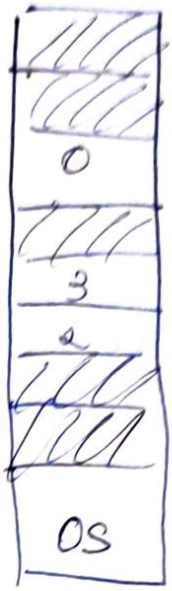
UNIT - 4

Virtual Memory

Need



Physical

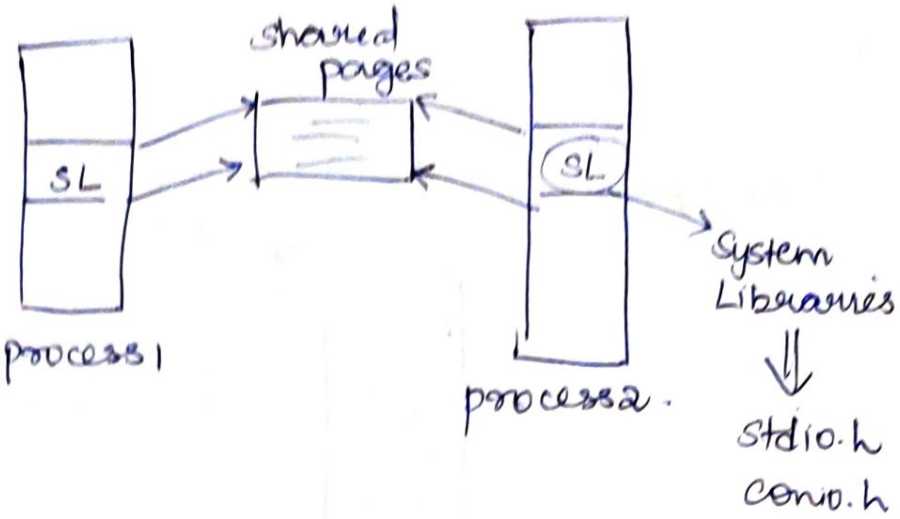


Upon completion of 0, 3, 2
→ 0, 3, 2 are swapped with 1, 4.

Advantage
→ Prg can be > physical memory & still execute.

Virtual Address Space
↳ Set of all virtual address space
→ CPU generated.

Shared Library



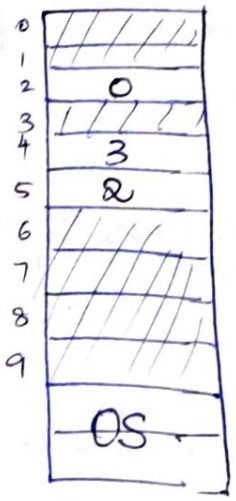
Shared pages reduce size.
Instead of being duplicated for each process

Demand Paging

↳ Pages are allocated to the frames in the main memory according to need or) demand to be executed.

Page Table

		→ valid/invalid bit.
0	2	V
1		I
2	5	V
3	4	V
4		I



V → Valid.
↳ allocated

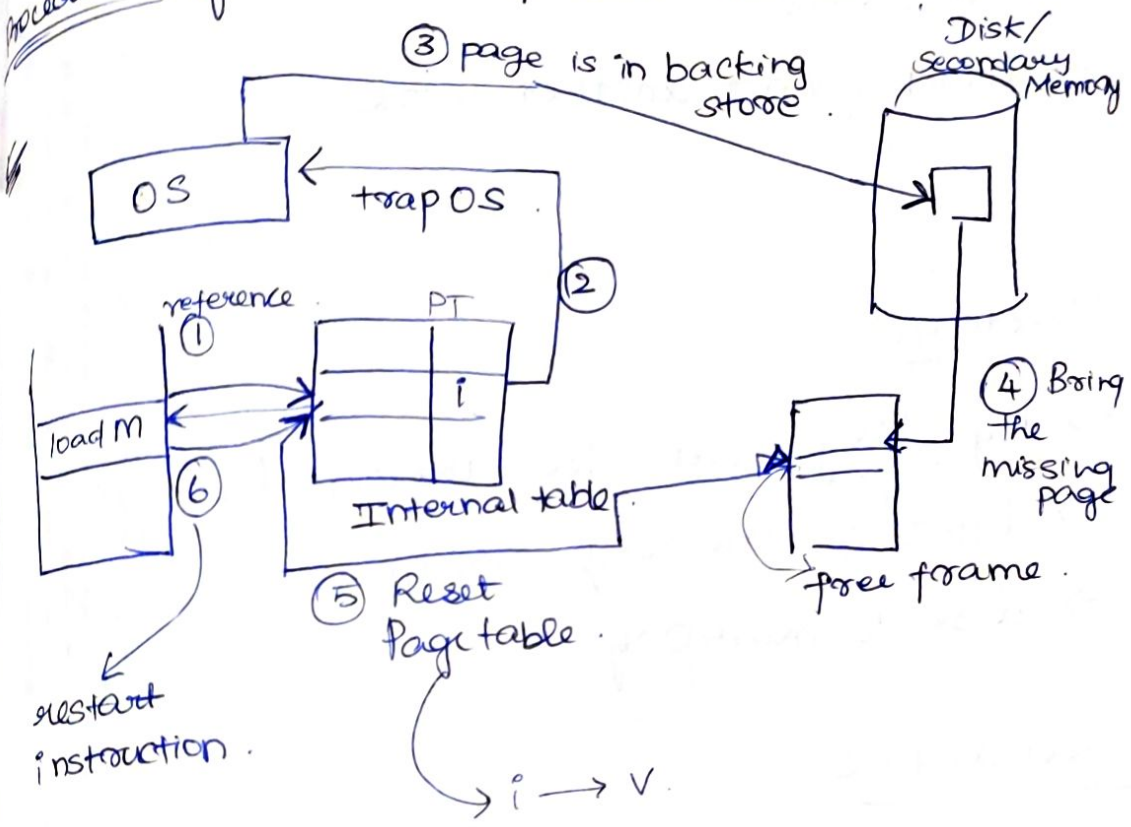
I → Invalid
↳ not allocated

Page Fault

→ User accesses a page that is not loaded in the main memory.

Like: Access of Page 1 → causes Page Fault.

Procedure for handling Page Fault



H/W

Page Table & Disk.

Performance

↓

Memory access time (MAT)

10 to 200 ns

Page Fault time (PFT)

↳ time taken to follow page fault process

0.03

Effective Access Time.

$$EAT = (1-p)ma + p \times pft$$

$0 < p < 1$

↳ like probability.

①

→ Interrupt

→ Save states.

→ Control given to another work.

②

Advantage

→ Prg large can be executed.

→ Effective memory usage.

→ Base to multiprg.

Disadvantage

→ Takes time for rectification.