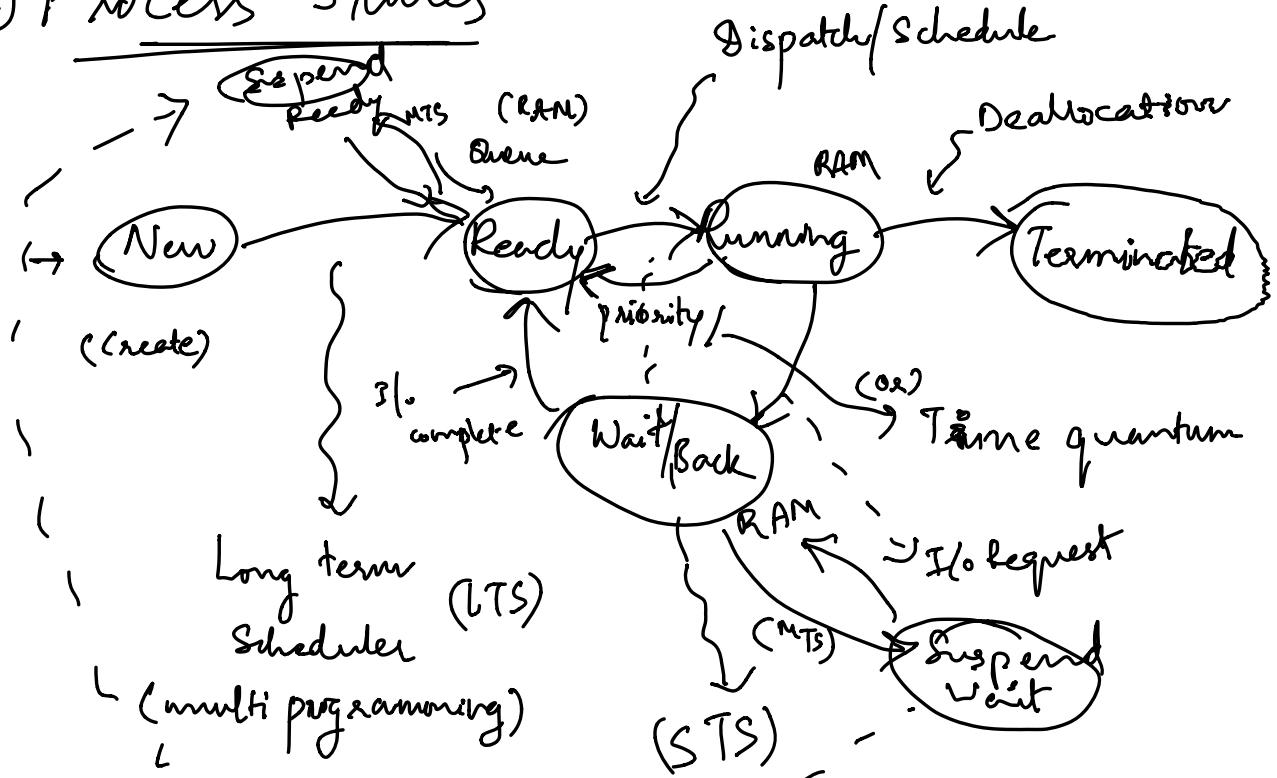
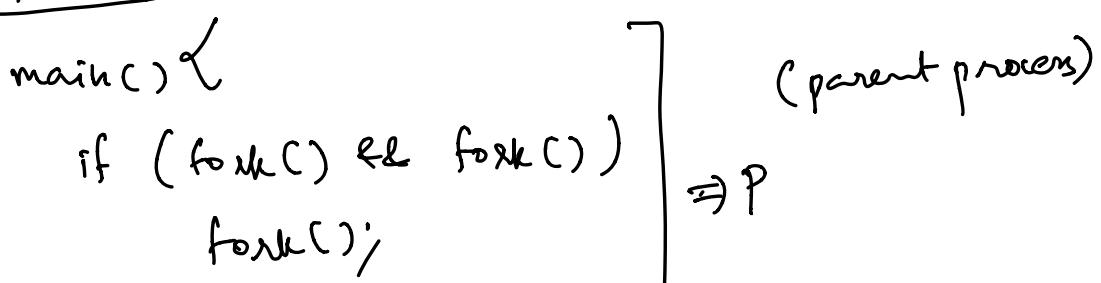


① Process States



Non-preemptive \rightarrow if CPU runs a process completely without terminating i.e., no priority / time quantum stops else preemp five

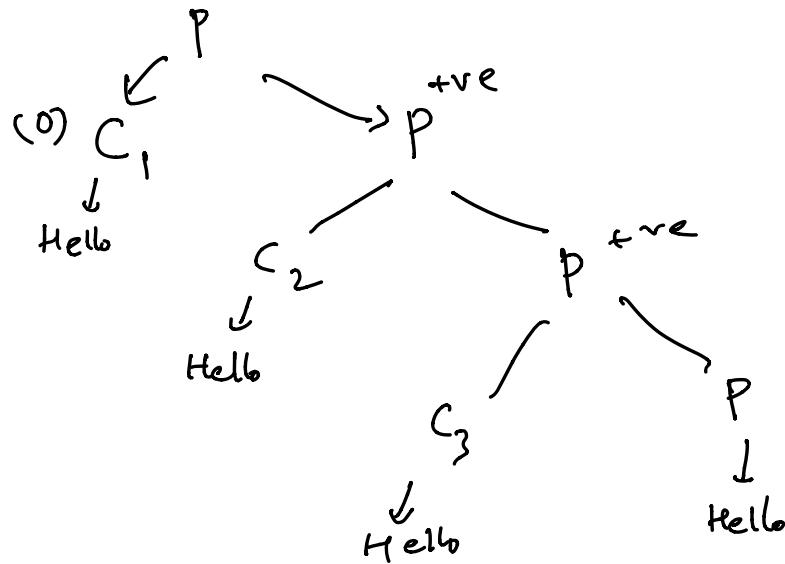
② Fork



```

    pf ("Hey");
    return 0;
}

```



③ CPU Scheduling

- Arrival: when ps enters ready state
- Burst: time required by a ps to get execute on CPU
- Completion time: time at which ps completes its execution.
- turnaround time:
 $(\text{completion} - \text{arrival})$

→ waiting time:

(turnaround - burst)

Ps have CPU bound & I/O bound

ps → CPU → I/O req → CPU

→ response time:

(time at which ps gets CPU first time
- arrival time)

1) FCFS → criteria: Arrival time
mode : non-preemptive

2) SJF → criteria: Burst-time
mode : non-preemptive

3) SRTF → criteria: Burst-time
mode : pre-emptive

III
(SJF with pre-emption)

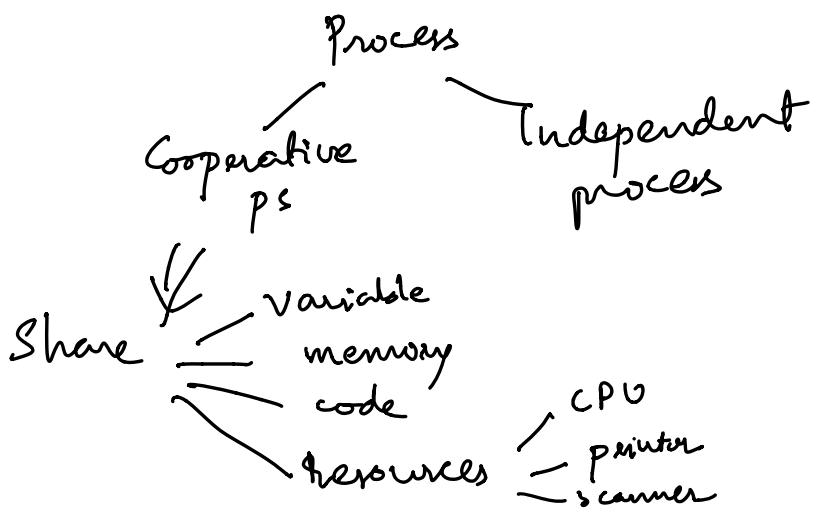
(gantt chart = running queue)

4) Round Robin → criteria: time-quantum
mode: preemptive

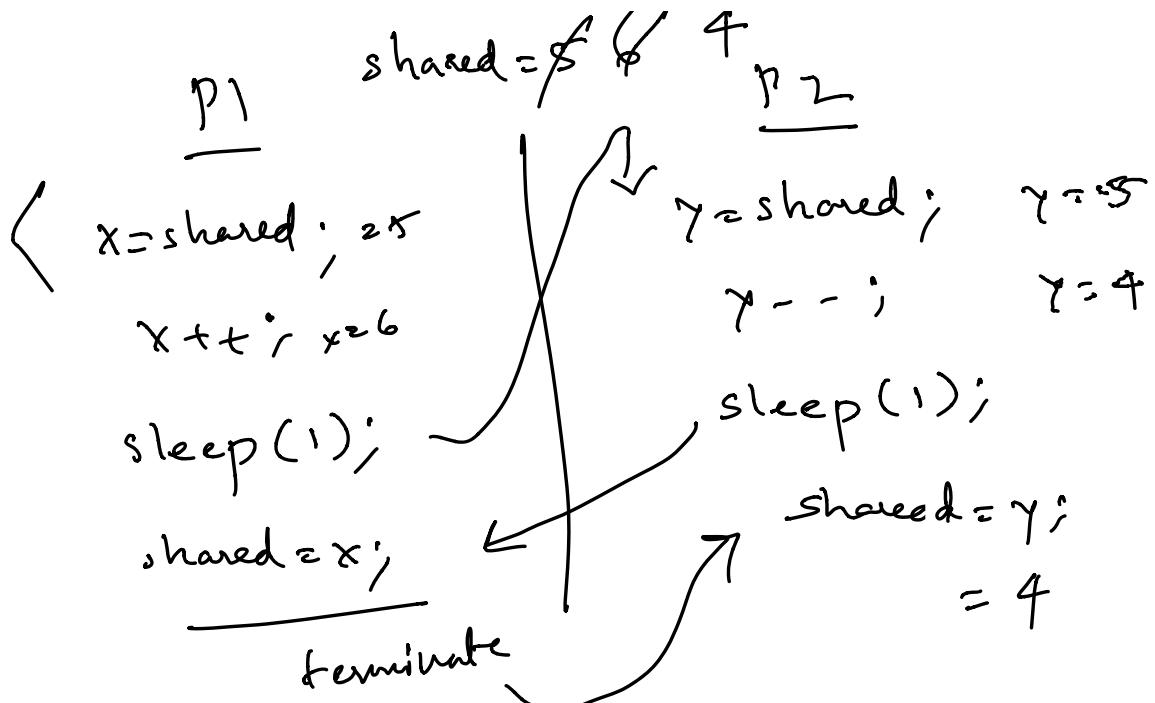
context-switching → save running pro
cess and bring a new pr (load)

5) pre-emptive priority
→ criteria: "priority"
mode: preemptive

④ Process Synchronization

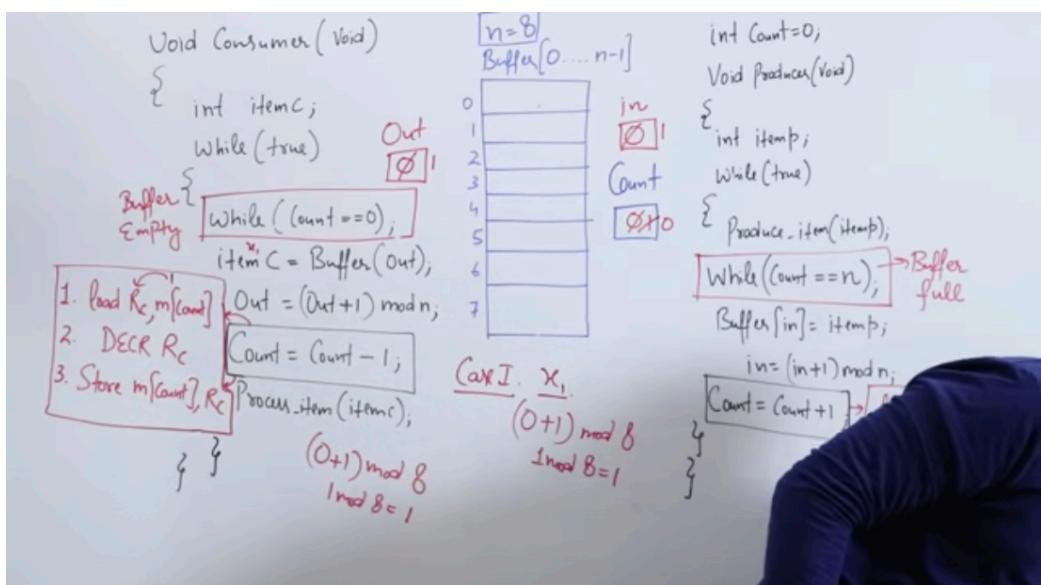


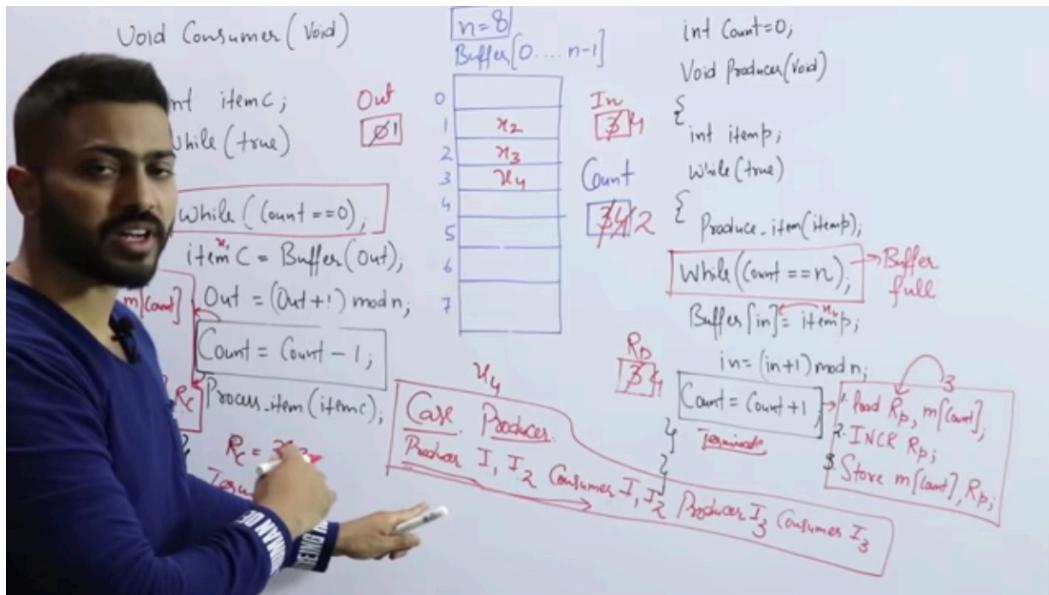
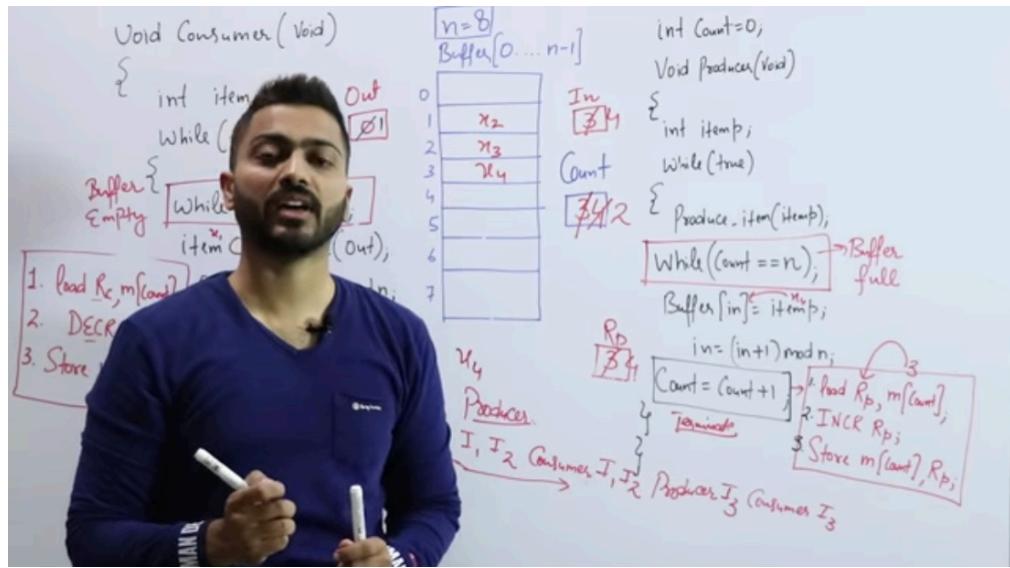
Ex: Race condition



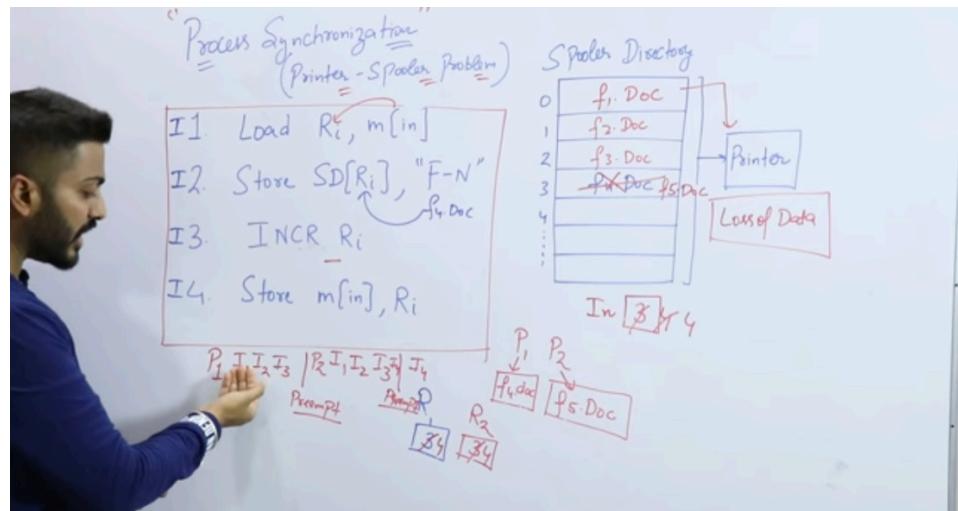
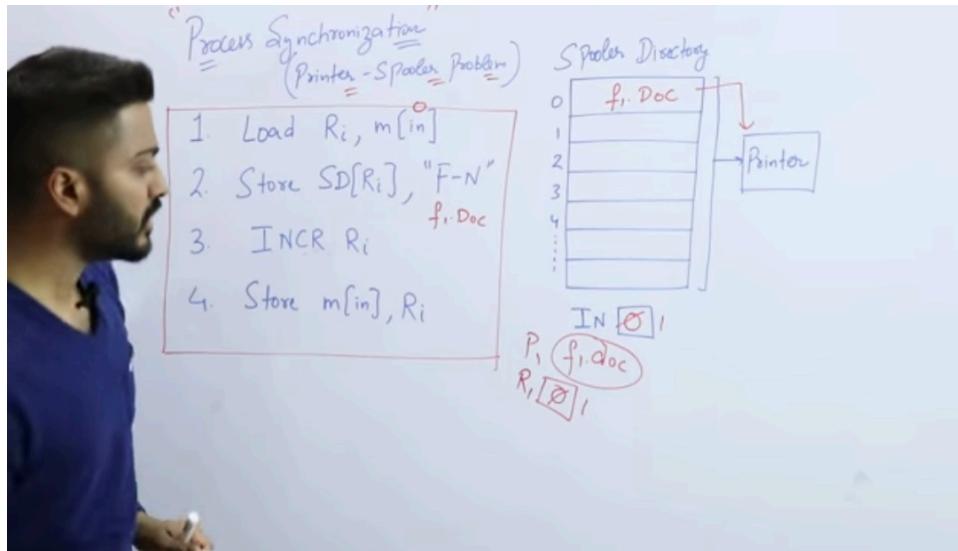
But ans should be 5.

Q) Producer - Consumer prob





ii) Printer-Spooler Problem



iii) Critical Section

→ is the part of the program where shared resources are accessed by various processes.

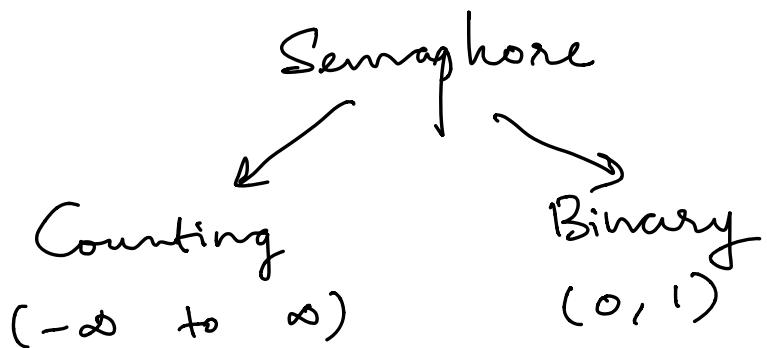
② is a place where shared variables, resources are placed.

Synchronization Mechanism

4 Conditions :

- 1) Mutual Exclusion } Primary
 } mandatory
- 2) Progress
- 3) Bounded Wait
- 4) No assumption related to t/w speed.

(iv) Semaphores → method to prevent race condition & achieve process synchronization (prevent deadlock, loss of data etc).



→ Semaphore is an integer variable which is used in mutual exclusive manner by various concurrent cooperative processes in order to achieve synchronization.

→ Operations : $p()$ & $v()$

$p()$ ↓
1) down, 2) wait
(entry)

$v()$
1) up, 2) signal,
post, release.
(Exit)

Wait (Semaphore S)

```
{   S.Value = S.Value - 1;  
   if (S.Value < 0)  
   {     put process (PCB) in  
         suspend list sleep(),  
   }  
   else return;  
}
```

Signal (Semaphore S)

```
{   S.Value = S.Value + 1;  
   if (S.Value ≤ 0)  
   {     Select a process from  
         Suspend list wake up(); }  
}
```

A) Binary Semaphore $\xrightarrow{0}$ $\xrightarrow{1}$

Wait (Semaphore S)

{ if (S.Value == 1)
 { S.Value = 0; }

else

{ // Block the ps & put in suspend list
 Sleep();

}
 }

Signal (Semaphore S)

{ if (Suspend list is Empty)
 { S.Value = 1; }

else

{ // Select a ps from suspend-list
 wake-up();

}
 }

Here in wait, signal if "if" section is executed then it's a successful operation

→ soln of Producer-Consumer
(using Binary Semaphore)

counting semaphore → full = 0
Binary ~ $\rightarrow S = 1$ → Empty = N

<p>Produce (item p)</p> <pre>{ down (Empty); down (S); Buffer[IN] = item p; IN = (IN+1) % N up (S); up (full); }</pre>	<p>Consumer (item c)</p> <pre>{ down (full); down (S); item c = Buffer[out]; out = (out+1) % N; up (S); up (Empty); }</pre>
--	---

→ soln of reader-writers problem

R-W → problem; W-R → problem

W-W → problem; R-R → no prob.

pseudo code

int rc = 0

semaphore rs = 1;

semaphore w = 1;

void Reader() {

 while (true)

 down(rs);

 rc = rc + 1;

 if (rc == 1) down(db);

 up(rs);

CS

 down(rs);

 rc = rc - 1;

```
if (rc == 0) up(db);  
up(rs);  
process-data  
} } }
```

```
void writer() {
```

```
while (true) {  
    down(db);
```

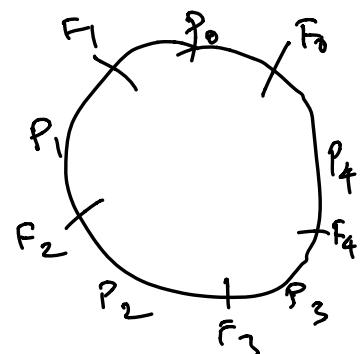
cs

```
} } } up(db);
```

Dining Philosophers

Prob:

```
void Philosopher(void)  
{  
    while (true)
```



```

    {
        Thinking();
        take-fork(i); ← left-fork
        take-fork((i+1)%N)
        Eat(); ← right-fork
        put-fork(i);
        put-fork((i+1)%N);
    }

```

If P_0 comes first
say $P_0 \rightarrow$ fork 0
next P_1 comes & takes fork 1
 P_0 has to wait for P_1 to eat
and put-fork back. To address
this issue we use array of
semaphores $S[N]$; N is no.
of forks.

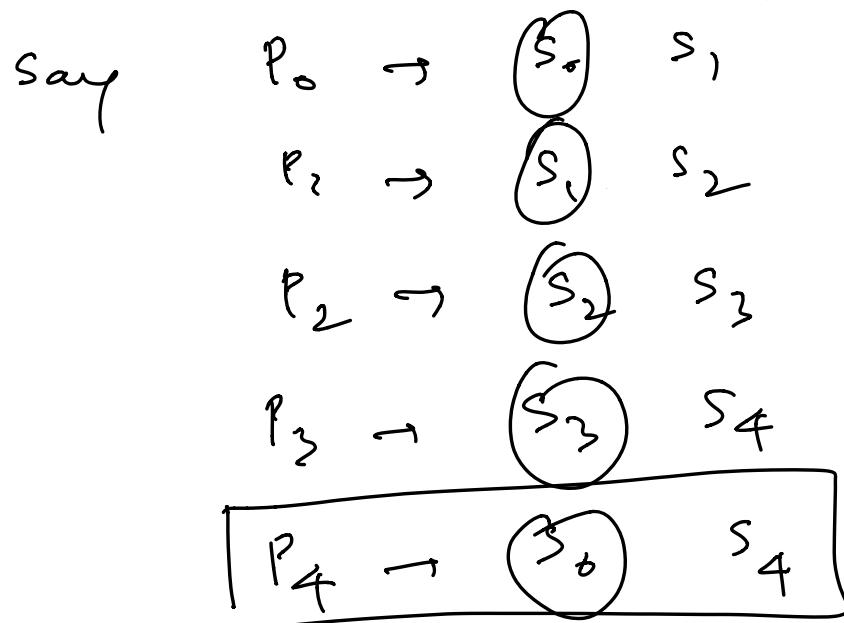
$$S[S] = \{1, 1, 1, 1, 1\}$$

solution:

```
void philosopher (void)
{
    while (true)
    {
        Think();
        wait (take-fork(Si));
        wait (take-fork(Si+1..N));
        EAT();
        signal (put-fork (Si+1..N));
        signal (put-fork (Si));
    }
}
```

But here, there's a deadlock situation say if every philosopher takes his left spoon & waiting for other spoon.

→ To solve this, just change the orientation for any 1 philosopher



for P_4 reverse the condn, hence

P_4 waits for S_0 meanwhile P_3 picks S_4 and eats & so on.

Deadlock will be resolved.

Deadlock

→ If two or more process are waiting on happening of some event, which never happens, then we say these processes are stuck in deadlock,

causing for deadlock (Necessary)

- 1) Mutual Exclusion.
- 2) No preemption.
- 3) Hold & Wait.
- 4) Circular wait.

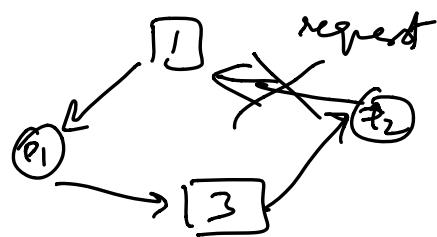
→ Various methods to handle it

- ↳ 1) Deadlock Ignorance (Ostrich method)
2) Deadlock prevention
3) Deadlock avoidance (Banker's Algo)
4) Deadlock detection & recovery

① → Ignorance : Cuz of rare occurrence of deadlock in OS we ignore it in many OS we use i.e., system hangs & we restart it. If we've extra code it might slow down os's speed.

② Prevention → try to remove one of the 4 causes of deadlock (eg: to remove circular wait, say that a process can request resource only in the increasing order

- i.e.
1. Printer
 2. Scanner
 3. CPU
 4. Register



Banker's Algo (deadlock avoidance)

(also used for dl. detection)

Ex: Total A=10, B=5, C=7

Ps	Allocation			Max Need			Rem. Need		
	(A)	(B)	(C)	(A)	(B)	(C)	(A)	(B)	(C)
P1	0	1	0	7	5	3	7	4	3
P2	2	0	0	3	2	2	1	2	2
P3	3	0	2	9	0	2	6	0	0
P4	2	1	1	4	2	2	2	1	1
P5	0	0	2	5	3	3	5	2	2
	7	2	5						

Remaining \rightarrow 3 3 2
 ↓
 P1 ✗
 ↓
 P2 ✓

rem \rightarrow 5 3 2
 ↓
 P3 ✗
 ↓
 P4 ✓

rem \rightarrow 7 4 3 \rightarrow P5 ✓

rem \rightarrow 7 4 5 \rightarrow P1 ✓

Sequence
 P2 \rightarrow P4 \rightarrow P5
 \rightarrow P1 \rightarrow P3
 (Safe sequence)

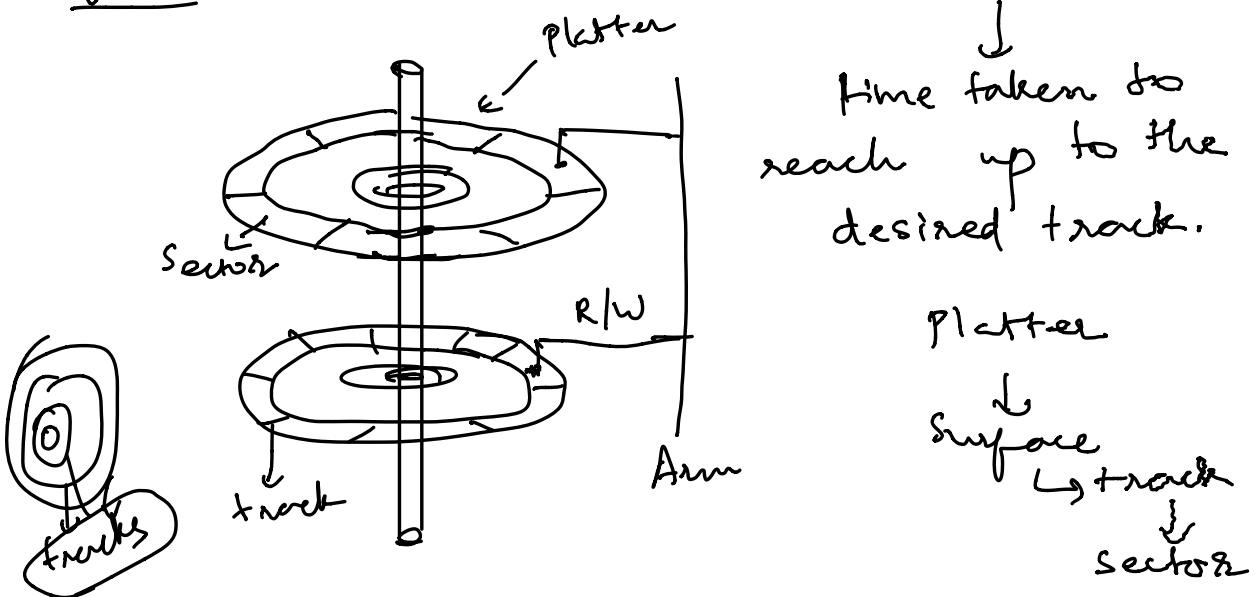
rem \rightarrow 7 5 8 \rightarrow P3 ✓

rem \rightarrow 10 5 7 = total ✓

If at any situation you can't
alloc the resources to any ps
then there's a deadlock.

⑤ DISK SCHEDULING ALGS

Goal: to minimize the seek time



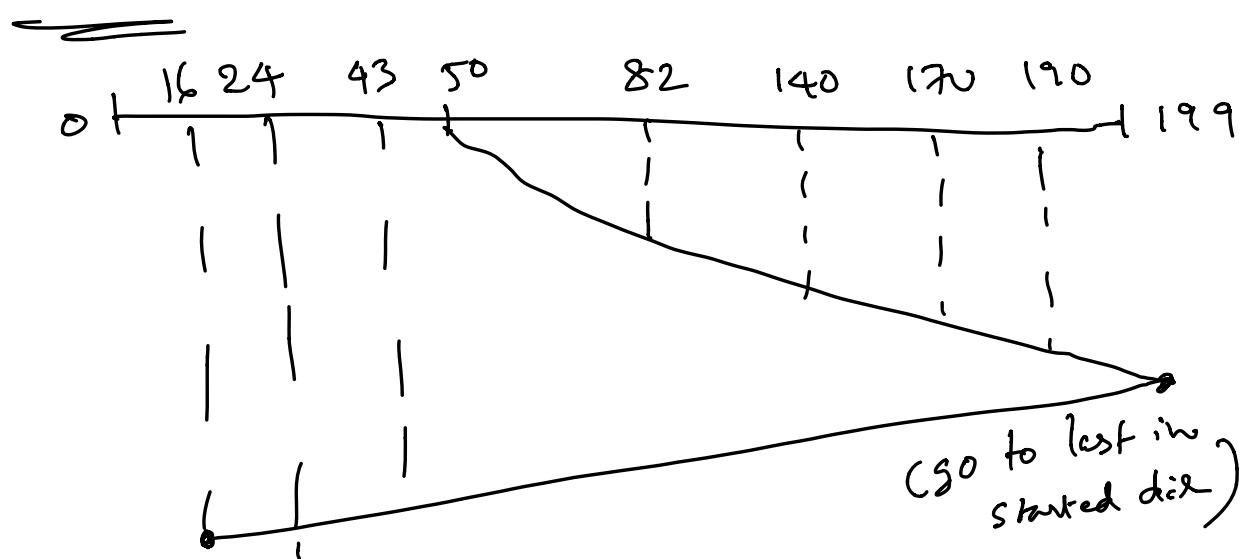
FCFS, SSTF (shortest seek time first), SCAN,
LOOK, CSCAN (circular scan), CLOCK.

FCFS ex: disk \rightarrow 200 tracks . request
 que \rightarrow 82, 170, 43, 140, 24, 16, 190
 R/W head = 50

total no. of tracks movement by R/W

$$\text{head} = (82-50) + (170-82) + (170-43) \\ + (140-43) + (140-24) + (24-16) + (90-16)$$

SCAN!



$$\text{moves} \rightarrow (199-50) + (199-16)$$

In look just go till 190.

In C-SCAN, go from 50 to 199 & then start from '0' again.

$$\text{moves} = (199 - 50) + (199 - 0) + (43 - 0)$$

In C-Look, go from 50 to 190 ?
then start from lowest value '16'

again

$$\text{moves} = (190 - 50) + (190 - 16) + (43 - 16)$$

In SSTF, from curr point to next point
such that seek time is min (greedy)
(prob of starvation)

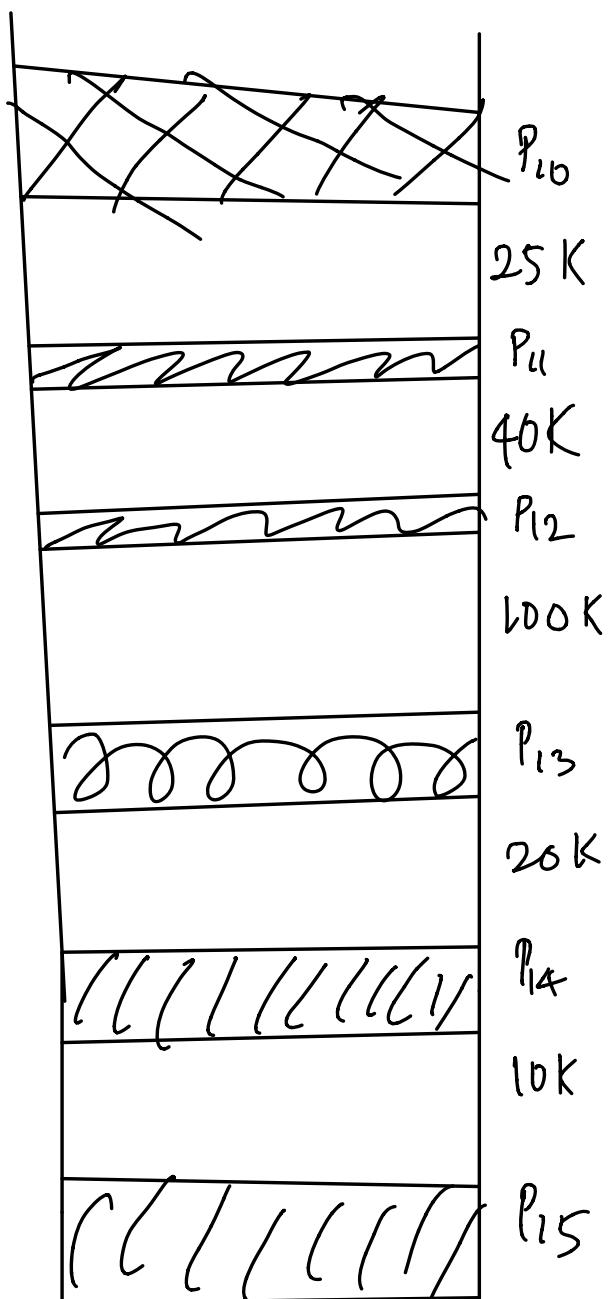
⑥ Memory Management

First fit: Allocate the first hole that is big enough

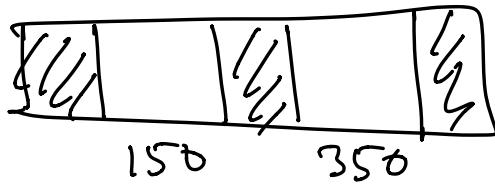
Next fit: Same as first fit but start search always from last allocated hole

Best fit: Allocate the smallest hole
that is big enough

Worst fit: Allocate the largest hole.



Σx_i



Request from ps \rightarrow 300k, 25k, 125k, 50k.
the above req could be satisfied by

- 1) Best \nsubseteq not first
- 2) Best not \nsubseteq first yes
- 3) Both
- 4) None

$\xrightarrow{\text{Sol:}}$ First

~~300~~ \hookrightarrow 150, 50

25 \hookrightarrow 125, 50

125 \hookrightarrow 0, 50

50 \hookrightarrow 0, 0

Best
300 \hookrightarrow 150, 50

25 \hookrightarrow 150, 25

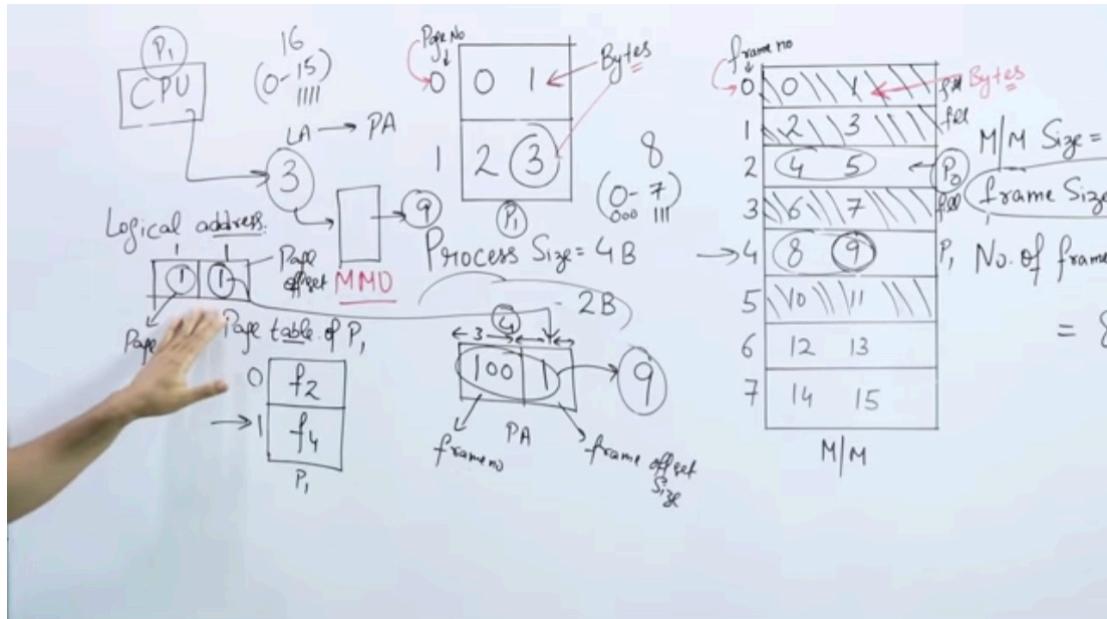
125 \hookrightarrow 25, 25

50 \hookrightarrow X

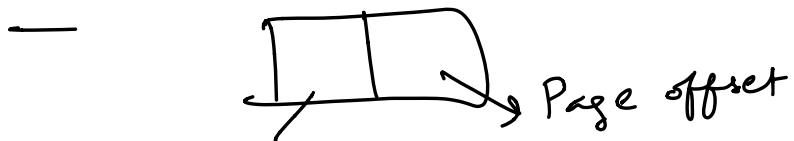
(External fragmentation)

\rightarrow worst fit works too

⑦ Paging $[2^{10} \rightarrow 1K, 2^{20} \rightarrow 1M, 2^{30} \rightarrow 1G, 2^{40} \rightarrow 1T]$



Q: Logical address LASpace



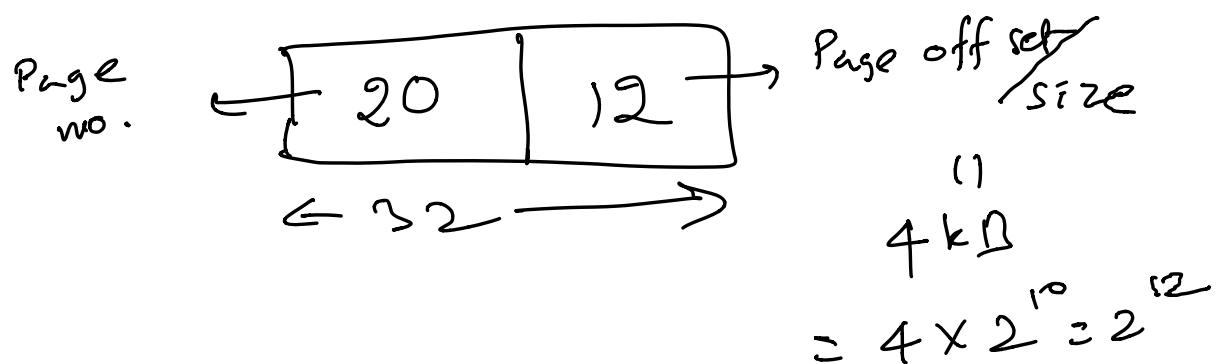
Given: LAS = 64GB

PAS = 64 MB

Page size = 4 kB

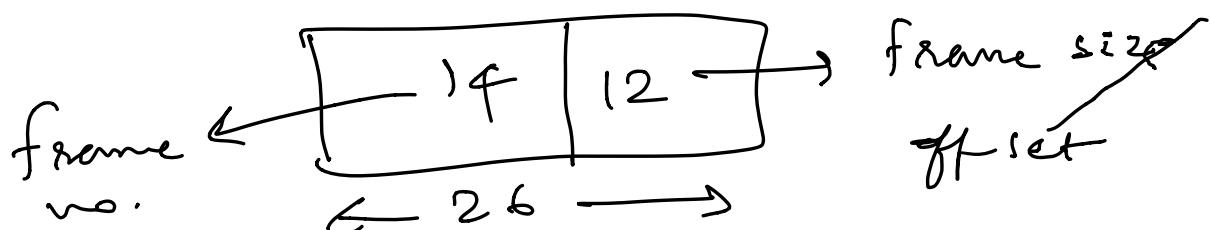
Q: No. of pages, No. of frames,
No. of entries in page table, size of page).

Sol: LAS \rightarrow $19B \rightarrow (10^3) \sim \cancel{2^{30}} 2^{30}$ bits
 $44B \rightarrow 2^2 \times 2^{30} \rightarrow 2^{32}$ bits



\Rightarrow 20 bits are used to represent page no. \Rightarrow no. of pages $= 2^{20} = 2048$

\rightarrow Frames calc from physical addrs.
 space (PAS) $= 64 MB = 2^6 \times 2^{20}$

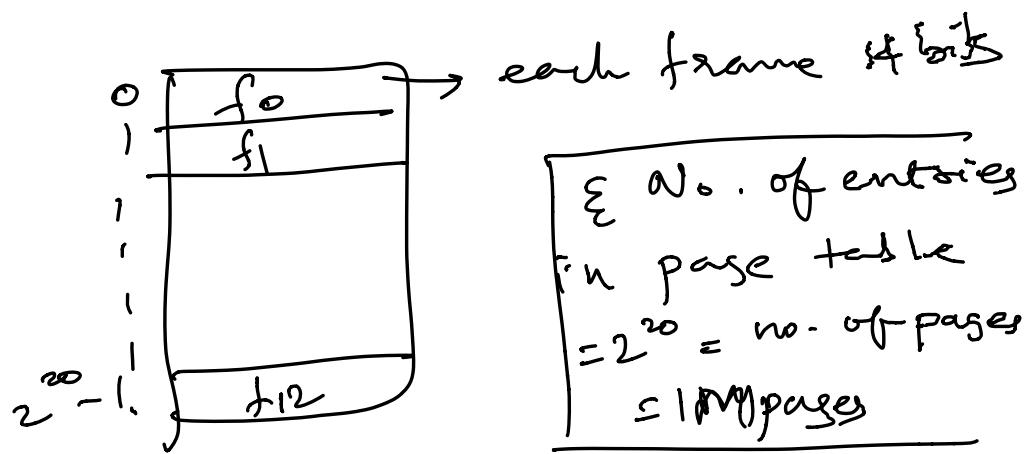


\rightarrow always frame size \equiv page size

so as to perfectly fit a page
in a frame

⇒ 14 bits for representing frame
no ⇒ 2^{14} frames

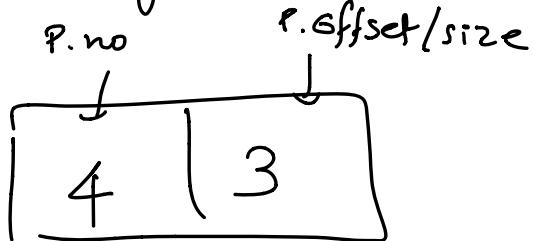
⇒ In page table we have
frame no.s for each page



$$\therefore \text{Page table size} = 2^{20} \times 14 \text{ bits}$$

Q: LA = 7 bits, PA = 6 bits, page size = 8 words. Calculate no. of pages & no. of frames

Soln: LA



$\leftarrow 7 \rightarrow$ 2^7 $(0-7)$ to rep.

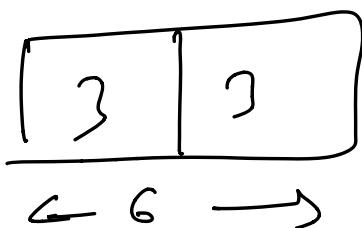
8 words = 8 bytes

In 32-bit system (1 word = 4 bytes)

$$\text{no. of pages} = 2^7 = 16$$

$= n$ - of entries
in page table

PA

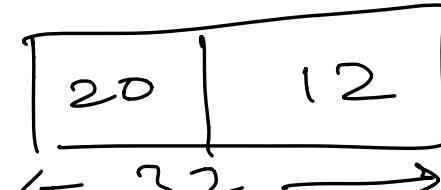


$\leftarrow 6 \rightarrow$

$$\text{no. of frames} = 2^6 = 8$$

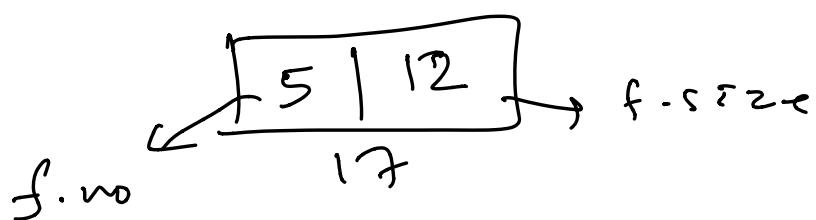
Q: LAS \rightarrow 32 bits, Page size = 4 kB
— RAM \rightarrow 128 kB. Then what will be ratio of page table & inverted page table size if each entry in both is of size 4B?

Sol: LAS/VAS \rightarrow



$$\text{Page size} = 4 \times 2^{10} = 2^{12}$$

$$\text{PAS} \rightarrow 128 \text{ kB} \rightarrow 2^7 \times 2^{10} = 2^{17}$$



$$\text{Page table size} = 2^{20} \times 4B$$

(as size of each entry is given)

In inverted page table, no. of entries are = no. of frames
= 2^5

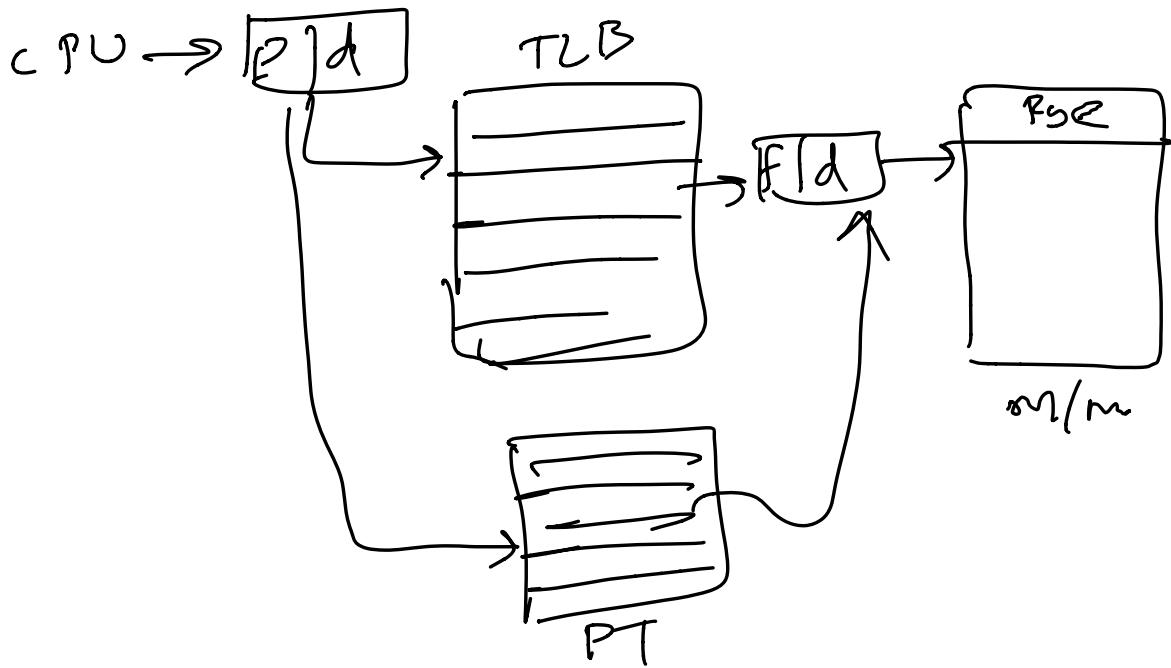
; Ratio $\Rightarrow 2^{20} : 2^5 \Rightarrow 2^{15} : 1$

Q: TLB access $\rightarrow 10$ ns & main memory access $\rightarrow 50$ ns. What is Effective machine time (in ns) if TLB hit ratio = 90% & no page faults.

Sol: EAT = $(0.9) \times (50 + 10)$
+ $(0.1) \times (10 + 100)$

$$\begin{aligned} &= 0.9 \times 60 + 11 \\ &= 54 + 11 = 65 \text{ ns} \end{aligned}$$

$TLB \rightarrow$ Translation Lookaside Buffer



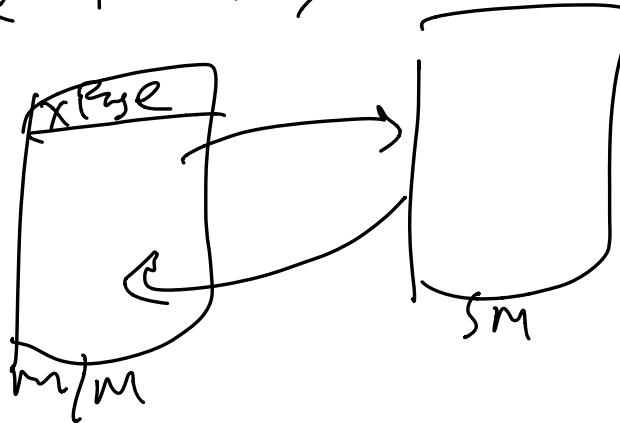
$$\Sigma MAT = \text{Hit}(TLB + M/M) + \text{Miss}(TLB + PT + M/M)$$

\downarrow

\equiv

m/m

if page fault, get it from secondary memory



which page fold service time (\times)

→ LRU Page Replacement Algorithm

$f_4 \ 2$

$f_3 \ \times \ f_1^1$

$f_2 \ 0$

$f_1 \ \times \ 3 \ \checkmark \ 7$

\downarrow fault \downarrow fault \downarrow fault \downarrow fault \downarrow hit \downarrow fault
 \downarrow hit \downarrow hit \downarrow hit \downarrow hit \downarrow hit \downarrow hit

Ref. String → 7, 0, 1, 2, 0, 3, 0, 4, 2,

3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1
↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑
hit hit hit hit hit hit hit hit hit hit

fault

fault

page faults → 8

Page flits → 12