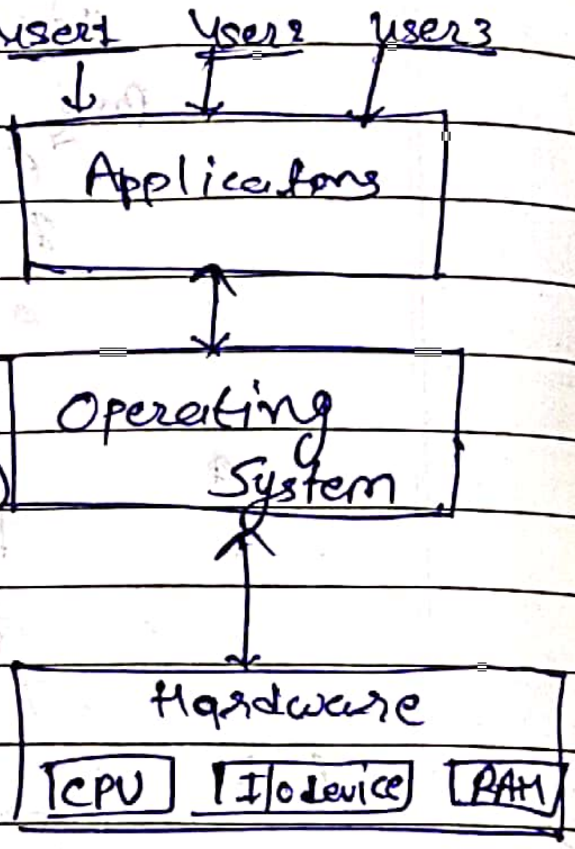
DATE: \ \
PAGE: 1

'Operating System & its Functions' :-

Lec-1:



Functionalities of operating system :-

1) Resource Management.
2) Process Mgmt. (CPU Scheduling)
3) Storage mgmt. (HD)
4) Memory Mgmt. (RAM)
5) Security

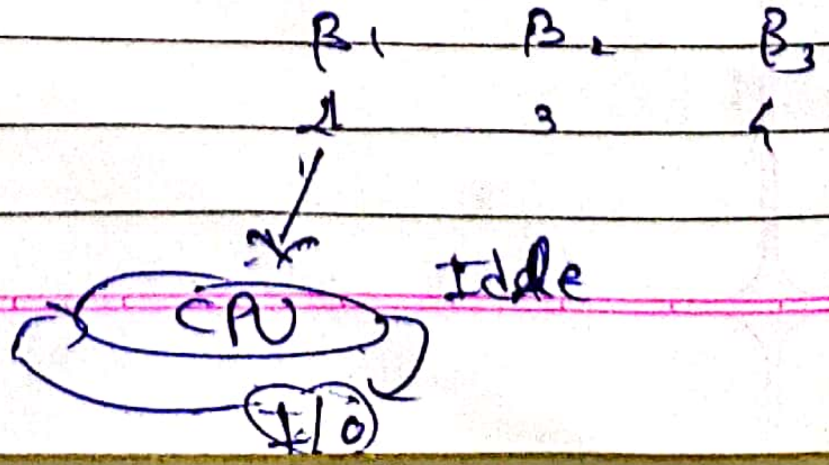
Windows acquised almost complete market
It provide great ~~convinence~~. Convinience.

Lec-2:

Batch Operating System :-

~~1) Batch~~
~~2) Mul~~

Punch cards
paper tape
mag. tape
⟶ Operator
↓
executes



DATE: / /
PAGE: 2

e.g. Fortran
IBSYS 709x
1960

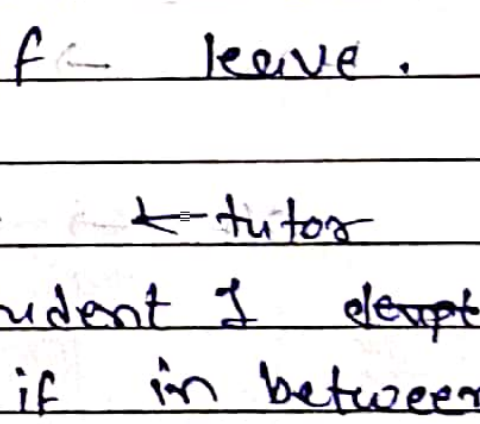
Lec-13 Multiprogrammed OS (non Preemptive)

Her complete whole process and then go for next step for process

. The CPU will not serve to any other process until & and Unless process which is on working it self leave.

e.g. student I ← tutor
tutor complete student I except doubt then reach to s2. but if in between s1 leave for some reason then also tutor will reach to s2.

Multitasking / Timesharing OS :-
(Preemptive)



→ It provides equal amount of time to every processes.
→ e.g 5 second P1, then P2, ... and repeat until all processes are processed.

tutor solve S1 1 que then go to S2 and solve his 1 que and so on. repeate until all ~~que~~ students' 5 ques solved.

Benefit over multi~~processing~~ programming :-

Response time increases.

Lec-1.4

a) Real time OS → Hard
→ soft

(time constraints)

Hard → missile Launch where time is important

soft → video streaming where significance is comparatively less

Clustered System

1) ~~Distributed System~~ :-

More than one computers make one computer.

2) ~~Clustered~~ Distributed :-

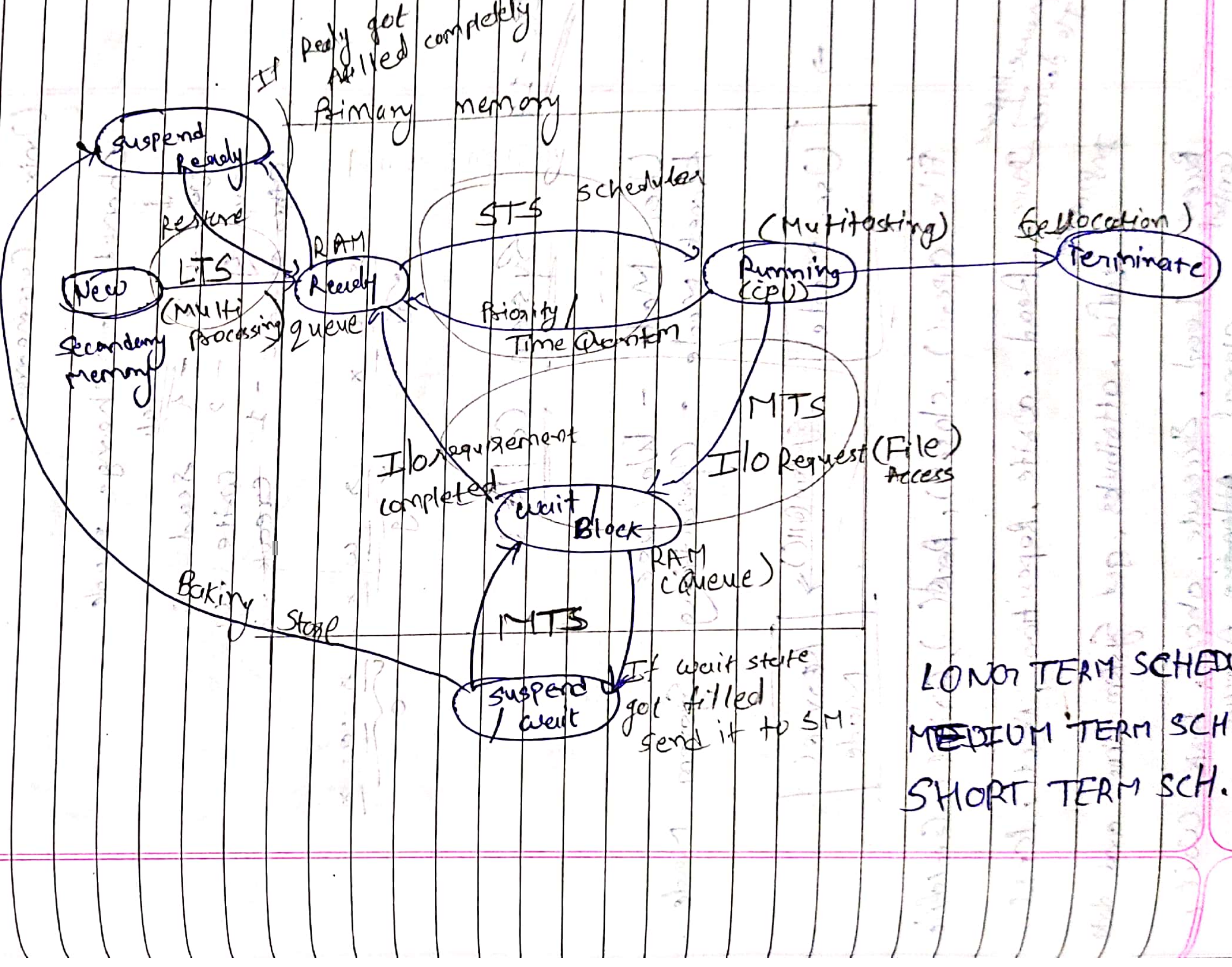
Devices are distributed, access multiple geographical regions.

3) Embedded System :-

micro oven, washing machine

DATE: / /
PAGE: 4

LEC-1.5 Process States



LONG TERM SCHEDUL
MEDIUM TERM SCH.
SHORT TERM SCH.

DATE: / /
PAGE: 3

Lec 1.6 Ques:-

Unix Commands:

Chmod → change mode
↓ octals
User - u — 4 Read - r
group - g — 2 write - w
Other - o — 1 execute - x

r w x r w x r w x
u g o

Lec-1.7 System Calls:

User Mode - We use
Kernel Mode - Operating System Mode.

→ [User Mode] —System Call()→ [Kernel Mode]

mouse, keyboard I/O devices

File → Open(), close(), Read(), Write(), Createfile
Device → Read, write, Reposition, ioctl, fcntl
Information → getid, attributes, get System time and date
Process control → Load, Execute, abort, Fork, Wait, signal, ~~Allocation~~ Allocate

DATE: / /
PAGE: 6

Communication → Pipe(), Create/delete connections, shmget()

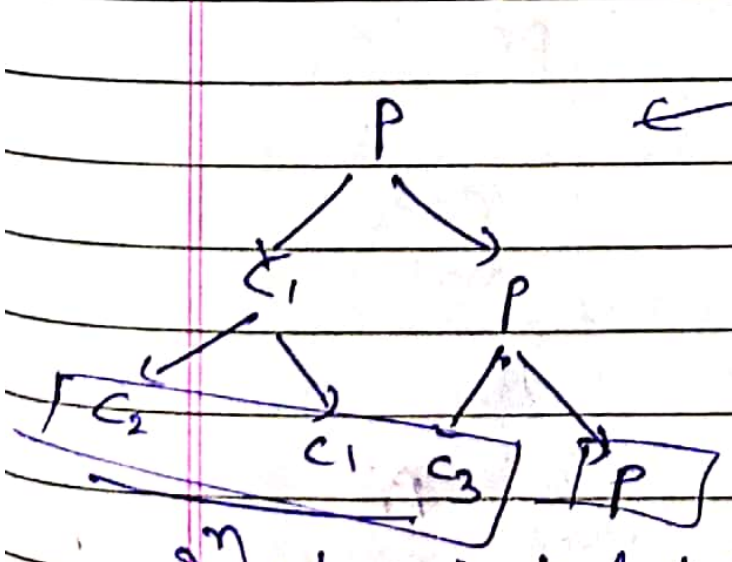
Protection → chmod

Lec-1.8

Fork()

0 → child
+1 → Parent
-1 → when child is not generated

```
main()
{
fork();
printf("hello");
}
```



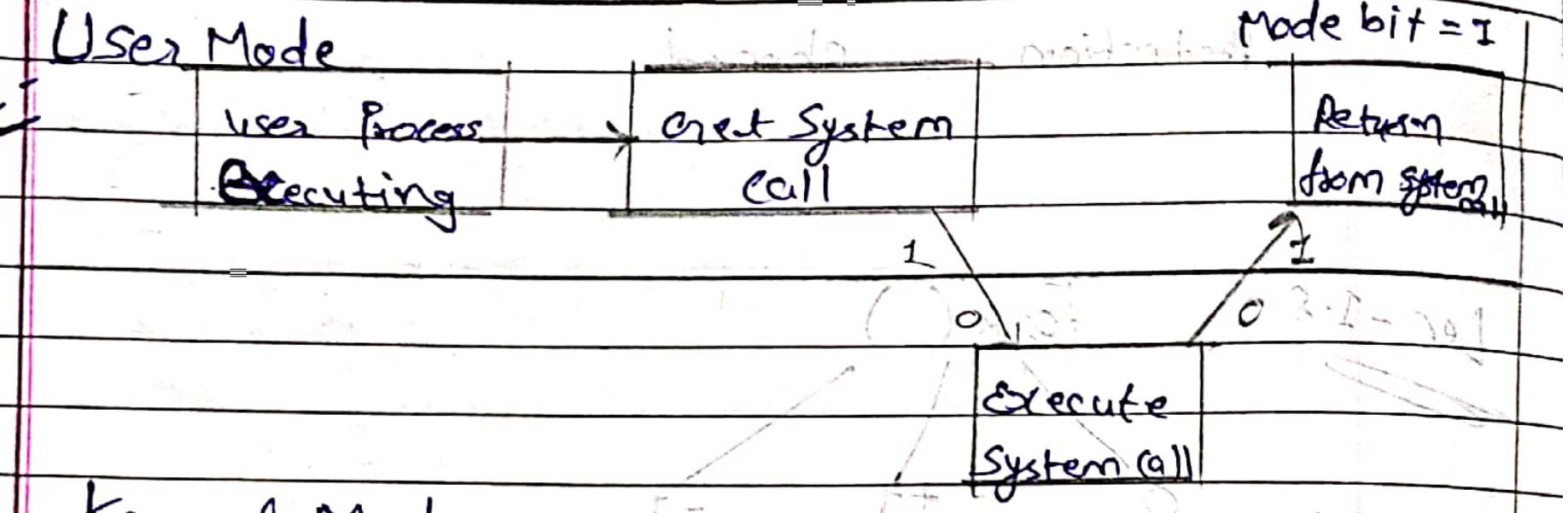
$2^n - 1$ ⇒ ~~test~~ total child processes

2^n ⇒ total ~~child~~ processes

DATE: / /
PAGE: 7

Lec-1.10

User Mode Vs Kernel Mode



HD (Hard Disk)

Lec-1.11

Process Vs Threads

DATE: / /
PAGE: 8

	Process	Threads
1)	System Calls involved in process .eg fork()	There is no system call involved.
2)	Os treats different processes differently	All user level threads treated as single task for Os.
3)	Different process have different copies of data, files, code	Threads share same copy of code and data.
4)	Content switching is slower	Content switching is faster.
5)	Blocking a process will not block another process.	Blocking a thread will block entire process
6)	Independent	Interdependent

Lec - 1.12

	User Level Thread	Kernel level Thread
1)	User level threads are managed by User level library	kernel level threads are managed by Os, system calls.
2)	User level threads are typically fast	kernel level threads are slower then User level.

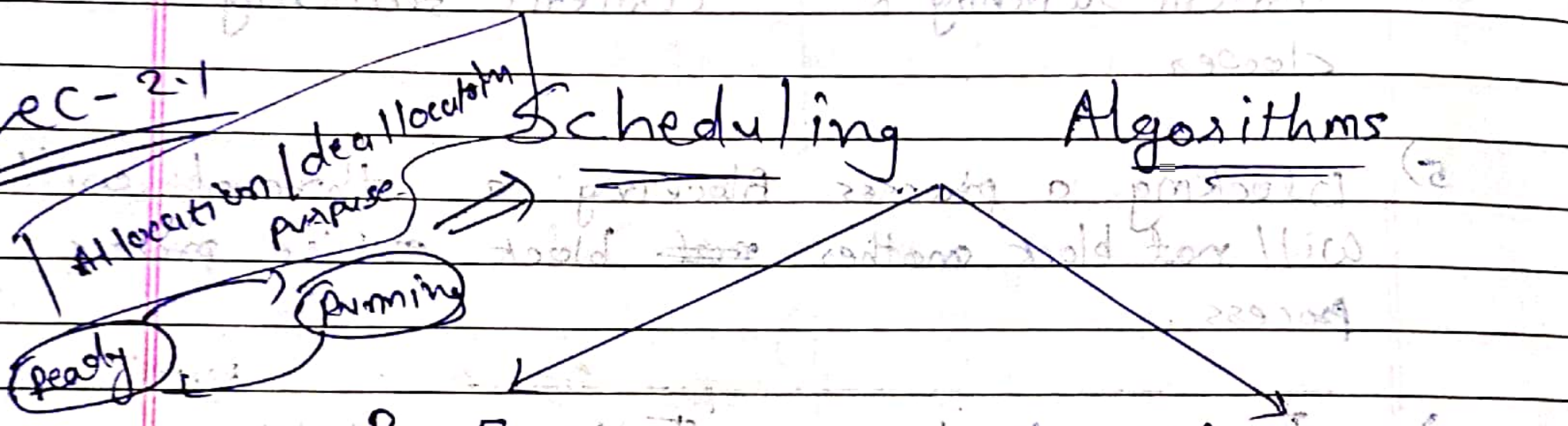
3) Context switching is faster.	Context switching is slow.
4) If one user level threads perform blocking operation then entire process get blocked.	If one kernel level thread blocked, No affects on others.

CT

Process > KLT > ULT

Lec-21

Scheduling Algorithms



Pre Emptive	Non Pre emptive
→ SRTF (shortest Remaining time first)	→ FCFS (first come first serve)
→ LRTF (Longest Remaining time first)	→ SJF (shortest Job first)
→ Round Robin	→ LJF (Longest Job first)
→ Priority based	→ HRRN (Highest Response Ratio Next)
	→ Multilevel Queue

Lec-2.2

"CPU scheduling"

(bank Entry time) Point of time

→ Arrival Time :- The time at which process enters the Ready Queue or state.

(when my work was helding) (duration)

→ Burst Time :- (Duration) or time required by a process to get execute on CPU.

(when I leave bank) (point of time)

→ Completion Time :- Point of which at process complete its execution.

(whole time from entering the bank to leaving bank)

→ Turn around Time = {Completion time - Arrival time}

(wait until process get started) (duration)

→ Waiting time = TAT - BT.

(time when I enter to the point of time whe my work started)

→ Response Time = The time at which a process gat CPU for First time - Arrival time

DATE: / /
PAGE: M

Lec-2.3

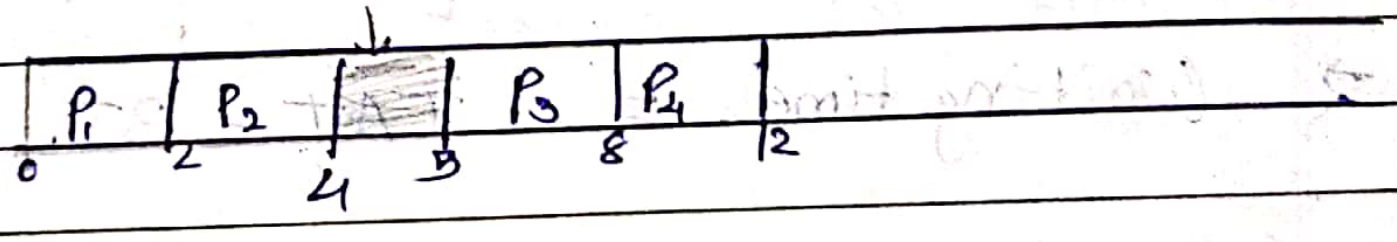
First come First Serve (FCFS) :-

{TAT-BT}
{AT-CT} ~~{BT-AT}~~;

Process No.	Arrival Time	Burst Time	Competion time	TAT	WT	RT
~~P_1~~	0	2	2	2	0	0
~~P_2~~	1	2	4	3	1	1
~~P_3~~	5	3	8	3	0	0
~~P_4~~	6	4	12	6	2	2

Criteria = "Arrival Time"

Gantt chart:-



Lec-2.4

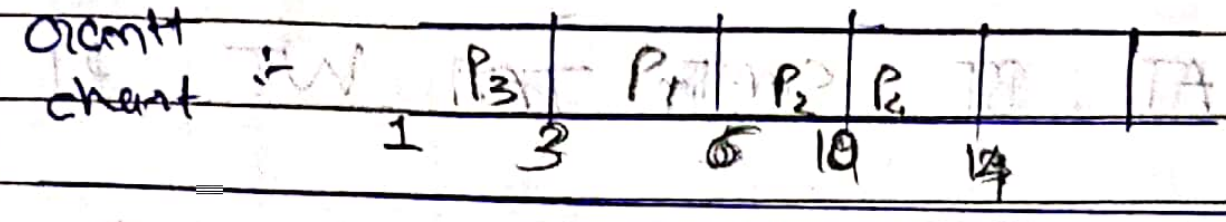
Shortest Job First (SJF)

P. No.	AT	BT	CT	TAT	WT	RT
~~P_1~~	1	3	6	5	2	2
~~P_2~~	2	4	10	8	4	4
~~P_3~~	1	2	3	2	0	0
~~P_4~~	4	4	14	10	6	6

DATE:
PAGE: 12

criteria = "Burst Time"
Mode = "Non premptive"

GANTT CHART :-

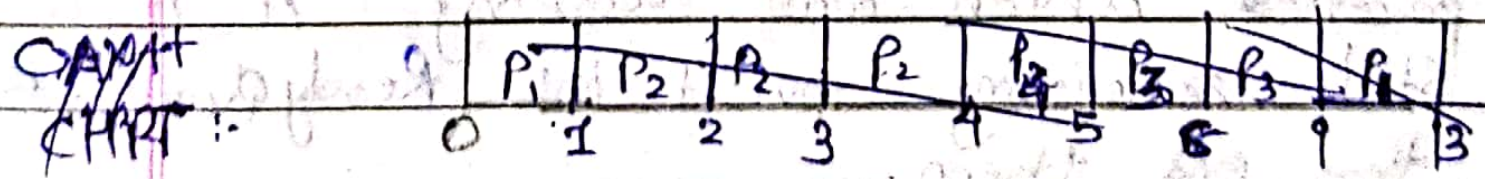


LEC - 2.5

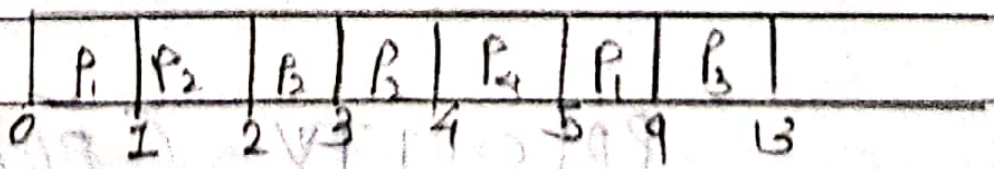
SORTEST REMAINING TIME FIRST:

P.NO.	AT	BT	CT	TAT	WT	RT
P1	0	8-5=3	9	9	6	0
P2	1	5-1=4	4	3	0	0
P3	2	4	13	11	7	7
P4	4	1-0	5	1	0	0

GANTT CHART :-



GANTT CHART :-



Dec-2.6

Round Robin

Criteria = "Time Quantom". Here TQ = 2

P.No	AT	BT	Cmp. Time	TAT	WT	RT
P_1	0	5 3 1	12	12	7	0
P_2	1	4 2 0	11	10	6	1
P_3	2	2 0	6	4	2	2
P_4	4	1 0	9	5	4	4

Ready Queue : ~~P_3 P_4 P_1 P_4 P_2 P_1~~

GANTT CHART :

P_1	P_2	P_3	P_1	P_4	P_2	P_1

0 2 4 5 8 9 11 12

first enter the eligible processes into ready queue then check the last processes which was working if it is still remaining put it into Ready Queu after ~~the end~~ them.

Lec-2.7

PRIORITY (PRE-EMPTIVE)

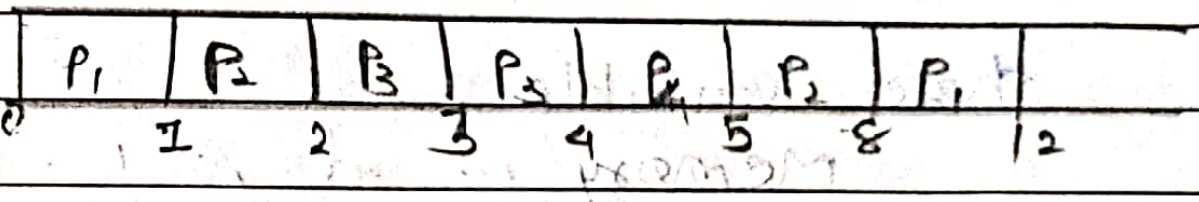
Higher the numer
Higher the priority

DATE: / /
PAGE: 14

Priority	Process No.	Arrival Time	Burst time	Completion time	TAT	WT
10	P_1	0	5 ~~4~~ 0	12	12	7
20	P_2	1	~~4~~ 3 0	8	~~7~~	3
30	P_3	2	~~2~~ 1 0	4	2	0
40	P_4	4	~~1~~ 0	5	1	0

criteria = "Priority"

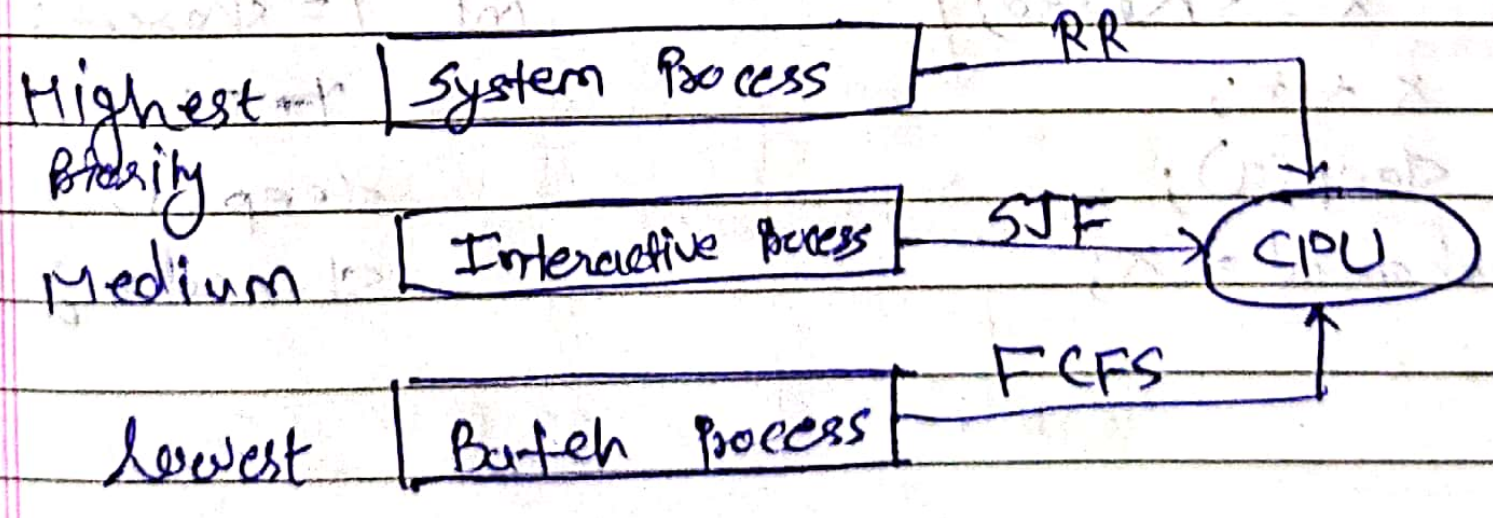
GANTT CHART :-



Lec-2.10

Multilevel Queue :-

→ Provide different type of Process to different Queues.



DATE: / /
PAGE: 13

→ what if we have a big amount of system processes in such cases starvation problem may occurs As a part of solution use Multilevel Feedback Queue.

Lec-31

Process Synchronization.

Processes

- Cooperative processes
- Independent processes

share Variable
Memory
code

Resources

- CPU
- Printer
- Scanners

int shared = 5

P_1	P_2
int x = shared;	int y = shared;
x++;	y++;
sleep(1);	sleep(1);
shared = x;	shared = y;
Terminate	terminate

switching context

x = 6

shared = 6

y = 6

shared = 7

Ans actully should be 5 because we are

DATE: / /
PAGE: 10

That values complete dat them it's called Race condition :-

mixing change of I in each process

→ ~~At the end~~ Result or value of shared variable depends on order of execution.

→ If we start from P2 it will be shared = 5

→ Such scenario is called "Race condion".

LEC-3.2 PRODUCER-CONSUMER :-

Buffer [0 ... n-1]

x_1
x_2
x_3

int Count=0;

```
void Consumer()
{
    int itemC;
    while(true)
    {
        while(Count==0)
        itemC = Buffer(out);
        out = (Out+1) mod n;
        Count = Count - 1;
        Process_item(itemC)
    }
}
```

```
void Producer(void)
{
    int itemP;
    while(true)
    {
        while
        I/O ⇒ Process_item(itemP)
        while(true)
        Buffer[in] = itemP
        in = (in+1) % n
        Count++;
    }
}
```

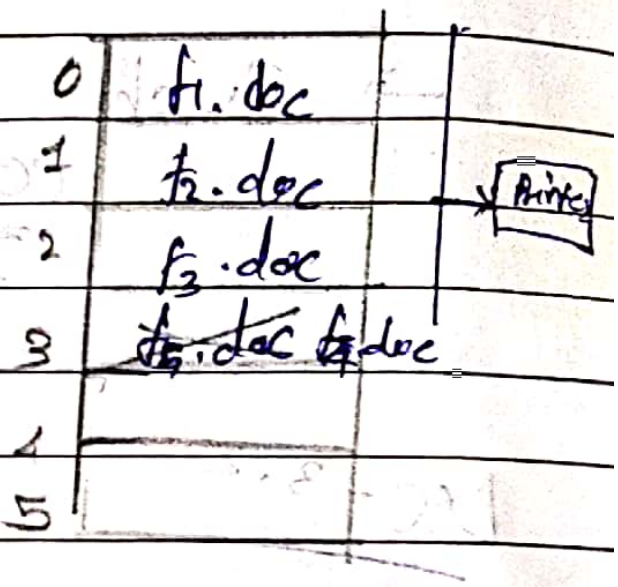
Producer I_1, I_2 consumer I_1, I_2 Producer I_3 consumer I_3

DATE: / /
PAGE: 17

Lec-3.3

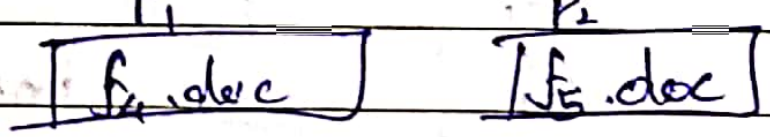
Printer - Spooler Problem :-

1. Load Ri, m[in]
2. Store SD[Ri], "F-N"
3. INCR Ri
4. Store m[in], Ri



→ first 3 files will be stored without any issue.

→ Now we have two processes simultaneously trying to add data in spooler.



→ P1 I1, I2 I3

R3 = 3
SD[R3] = f4.doc
R3 = 4

Context switch

→ P2 I1 I2 I3 I4

R4 = 3
SD[R4] = f5.doc
R4 = 4
Store m[4] = f5.doc

→ I4 (of P1)

Store m[4] = f4.doc

DATE: \ \
PAGE: 18

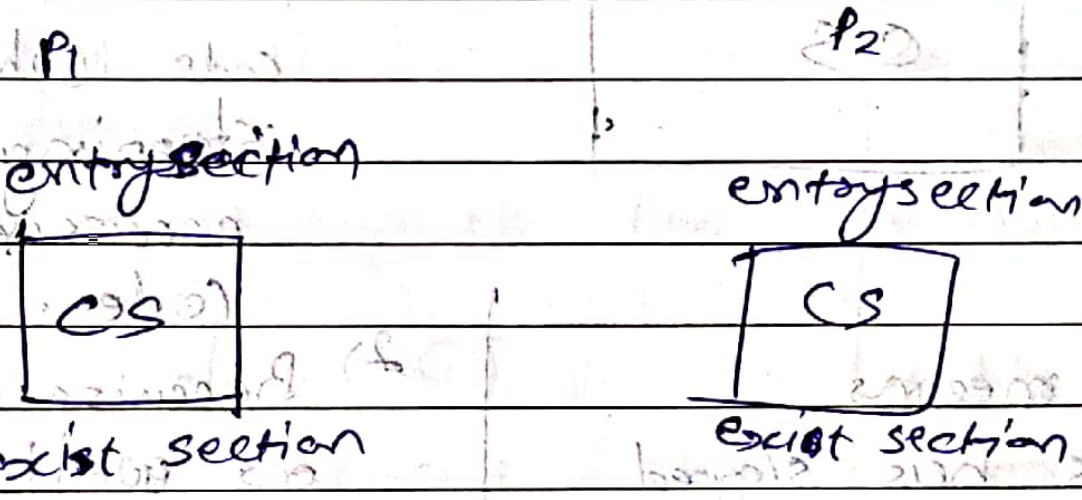
→ because of Asynchronization data loss occured.

"Critical Section"

Lec-3.4

" The program where shared resources are accessed by various processes".

→ place where shared variables, resources are placed.

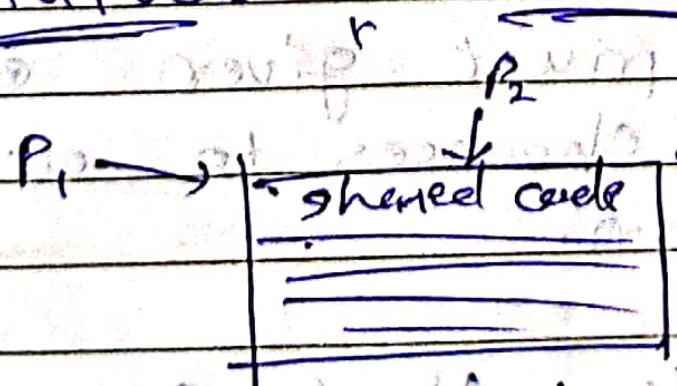


Synchronization Mechanism :-

primary { 1) Mutual Exclusion
2) Progress

secondary { 3) Bounded Wait
4) No assumption related to H/w speed

1) Mutual Exclusion :-



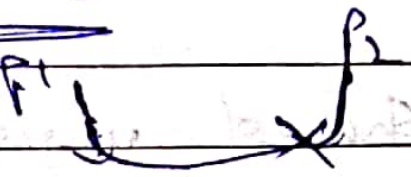
→ If P_1 enters into shared code P_2 or P_3 are not allowed to enter.

DATE: \ \
PAGE: 19

~~Mutex~~

→ At a time only one process can ~~can~~ access shared code.

2) Progress



1) P_2 want to access a shared code but here P_1 ~~because~~ hug some code which is ~~not~~ stopping P_2 from accessing shared code.

2) Because of this P_1 as well as P_2 both ~~can't~~ are not able to access a code.

3) which means progress has stopped because of their fight

→ e.g Socity people don't want to work ~~as~~ and also don't let some one to progress.

3) Bound Wait

→ all processes must given equal opportunity for chances to access Critical section.

→ It must not like treat any process

is ~~or~~ accessing for infinite time and other ~~other~~ is waiting for infinite.

4) No assumption for H/w

solution must work to any H/w and it must portable / universal.

Lec - 3.5

```
do {
    acquire lock
        CS
    release lock
}
```

* Execute in User Mode
* Multiprocess Solution
* No mutual Exclusion Guarantee

pseudo code

```
1. while (Lock == 1);
2. Lock = 1
3. Critical section
4. Lock = 0
```

Entry code

exit code

case-1

P_1	P_2
I - 1 2 3 4	P_2 x

Lock = 0 Varem = I full

case-2

P_1	P_2	P_1
I *	1 2 3	2 3
↑	CS	CS
preemtion		

other mutual exclusion fulfilling both

both are in critical section

DATE: / /
PAGE: 20

Lec-3.6

Critical Section Solution using "Test_and_Set" instruction.

```
while (test_and_set(&lock));
    Critical section
lock = false
```

→ Merge a two lines from user code to prevent preemption

```
boolean test_and_set(boolean *target)
{
    boolean x = *target;
    *target = TRUE;
    return x;
}
```

Lec-3.7

Turn Variable (Strict Alteration)

* 2 Process Solution
* Run in User Mode

Process "P0"	Process "P1"
Non CS	Non CS
while(turn != 0);	while(turn != 1);
Critical section	Critical section
turn = 1;	turn = 0;

→ Mutual exclusion achieved
→ Progress " Not achieved
→ ~~BW~~ BW achieved since ~~Hmm~~ turn variable at the end of code.

Semaphores → $-\infty$ to $+\infty$
→ 0 & 1

Lec-36

P() , Down(), Wait
V() , Up() , Signal

S=0 there is no value inside suspended queue and also no one is allowed to enter into critical section.

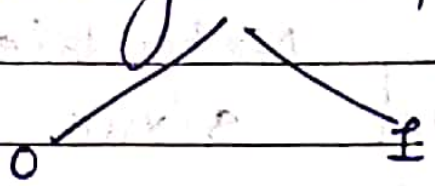
→ Semaphore is an Integer variable which is used in mutual ~~exclusion~~ exclusive manner by various concurrent Cooperative processes in order to achieve synchronization.

```
Down(Semaphore S)
{
  S.value = S.value - 1;
  if(S.value < 0)
  {
    put process(PCB) in
    suspended list sleep();
  }
  else {
    return;
  }
}
```

```
Up(Semaphore S)
{
  S.value = S.value + 1;
  if(S.value >= 0)
  {
    Select a process
    from suspended list
    wake up();
  }
}
```

LEC-39

Binary Semaphore



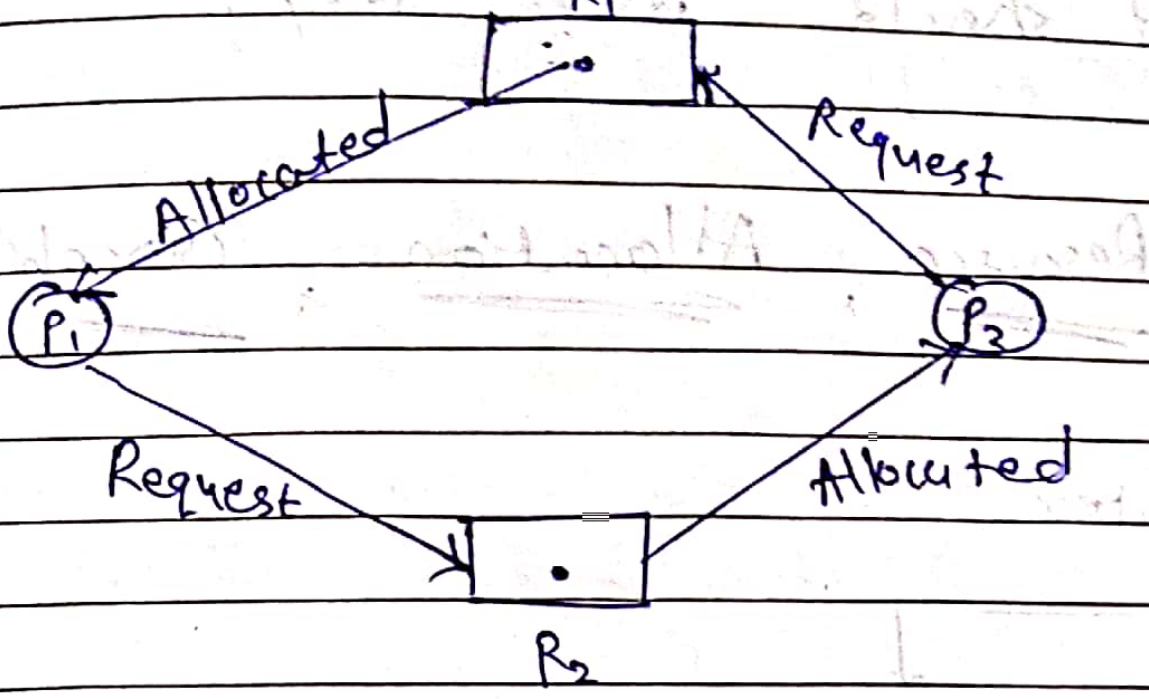

```
Down(Semaphore S)
{
  if(S.value == 1)
  {
    S.value = 0;
  }
  else
  {
    Block this process
    and place in
    suspend list
    Sleep();
  }
}
```

```
Up(Semaphore S)
{
  else if(Suspend list is empty)
  {
    S.value = 1;
  }
  else
  {
    Select a process
    from suspend list
    and wakeup();
  }
}
```

LEC-40

Deed Lock :-

→ If two or more processes are waiting on happening of some event, which never happens, then we say these processes are involved in deadlock then that state is called.



⇒ Necessary conditions for Deadlock:

1) Mutual Exclusion
2) No-Preemption (No context switching)
3) ~~Hold~~ Hold & wait
4) Circular wait

1) Mutual Exclusion
⇒ More than one processes can't hold resources at a same time.

2) No-Preemption
⇒ must not leave resources inbetween or switch in between.

3) Hold & wait
⇒ Process must hold some resources and also waiting for some resources tobe allocated.

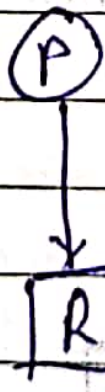
4) Circular wait

DATE: / /
PAGE: 25

→ Wait should make loop.

LeC-4-2 Resource Allocation Graph(RAG)

- Vertex
  - Process Vertex
    - Single Instance
      - e.g CPU, Monitor
    - Multi Instance
      - Register
  - Resource Vertex
- Edge
  - Assign Edge
  - Request Edge
- Waiting
  - Finite (starvation)
  - Infinite (Deadlock)

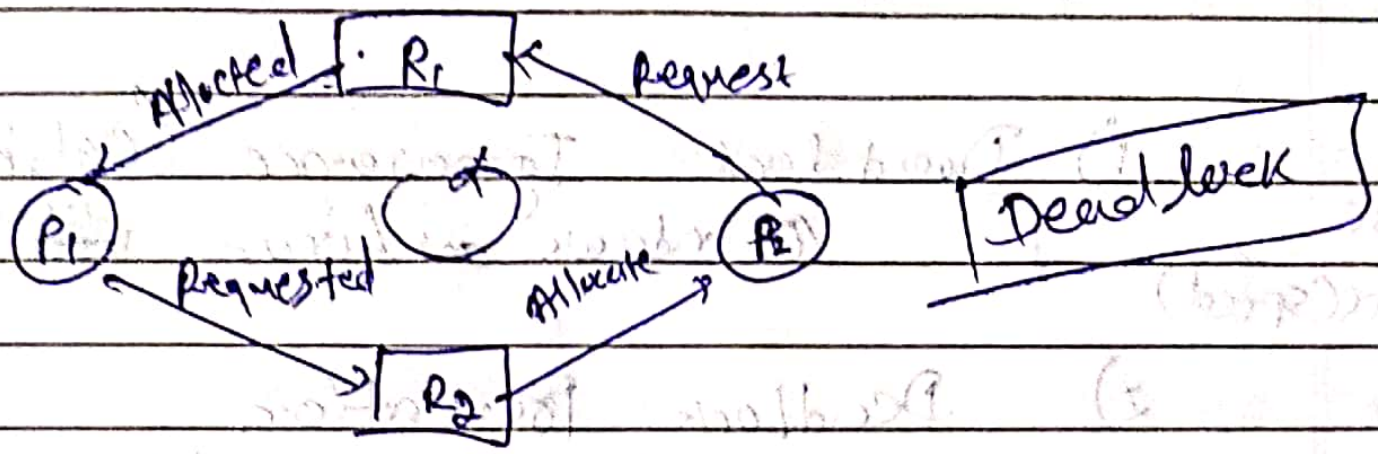


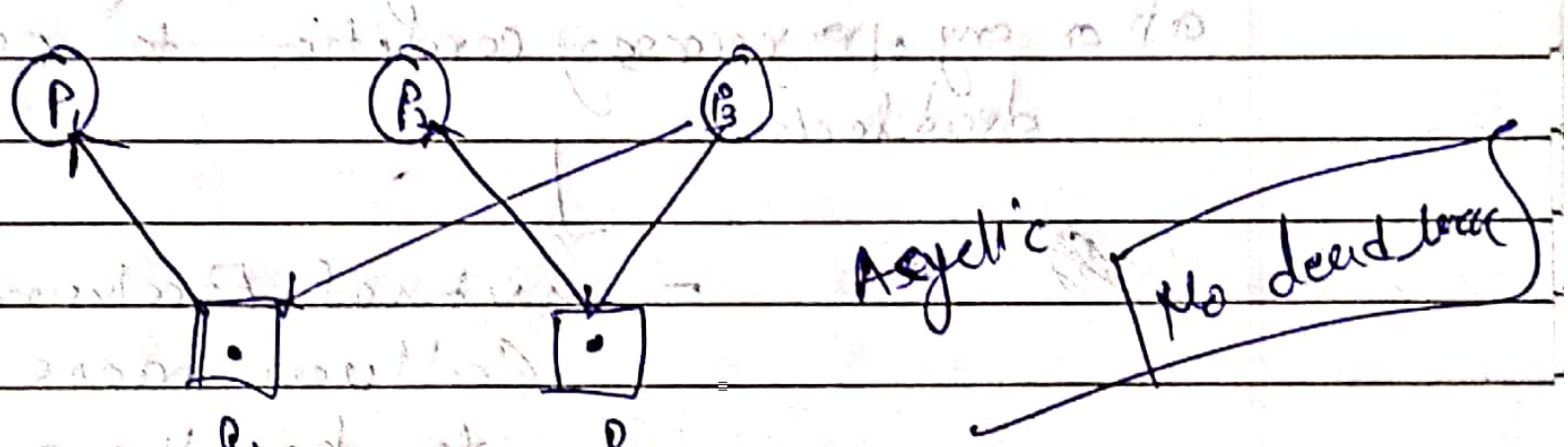
DATE: \ \
PAGE: 26

Checking method:

(1) try to find cycle (only in sigle Instance Resource)

(2) Availibility Matrix





	Allocated R_1	Allocated R_2	Request R_1	Request R_2
✓ P_1	1	0	0	0
✗ P_2	0	1	0	0
✓ P_3	0	0	1	1

Availibility (0, 0) P_1
(1, 0)
(1, 1) P_3

DATE: \ \
PAGE: 27

Lec-44

Deadlock Handling Methods and Deadlock Prevention

1) Deadlock Ignorance (Ostrich method)
(Windows and Linux use this method)
increase performance (speed)

2) Deadlock Prevention
↳ try to discard or remove all or any of the necessary condition to occur a deadlock.

- Mutual Exclusion
(allow more than process to handle a single resource at a time)
- No-Preemption
(We can make processes premptive)
- Hold & wait
(assign some time quantum to processes)
- Circular wait
(give numbers to resources)

3) Deadlock Avoidance (Banker's Algo
(check before assigning resources to processes whether it will create a deadlock or not)

DATE: / /
PAGE: 08

4) Deadlock Detection And Recovery
1) kill the protection
2) Resource Preemption

Lec-4.5 DeadLock Avoidance (Banker's Algo)

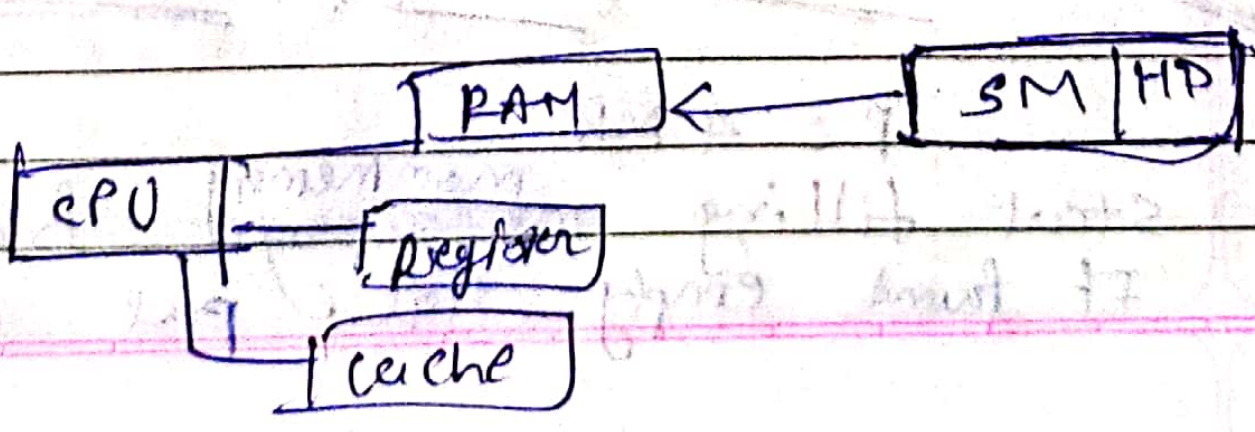
total A = 10 , B = 5 , C = 7
$-\frac{7}{3}$ $-\frac{2}{3}$ $-\frac{5}{2}$ (MaxNeed - Alloc)

	Allocation			Max Need			Available			Remaining Need		
R	A	B	C	A	B	C	A	B	C	A	B	C
P_1	0	1	0	7	5	3	3	3	2	7	4	3
P_2	2	0	0	3	2	2	5	3	2	1	2	2
P_3	3	0	2	9	0	2				6	0	0
P_4	2	1	1	4	2	2	7	4	3	2	1	1
P_5	0	0	2	5	3	3	7	4	5	5	3	1
	7	2	5									

find a safe sequence for the same.

$P_2 \rightarrow P_4 \rightarrow P_5 \rightarrow P_1 \rightarrow P_3$

Lec-5.1 Memory Management: Methods of managing Primary memory

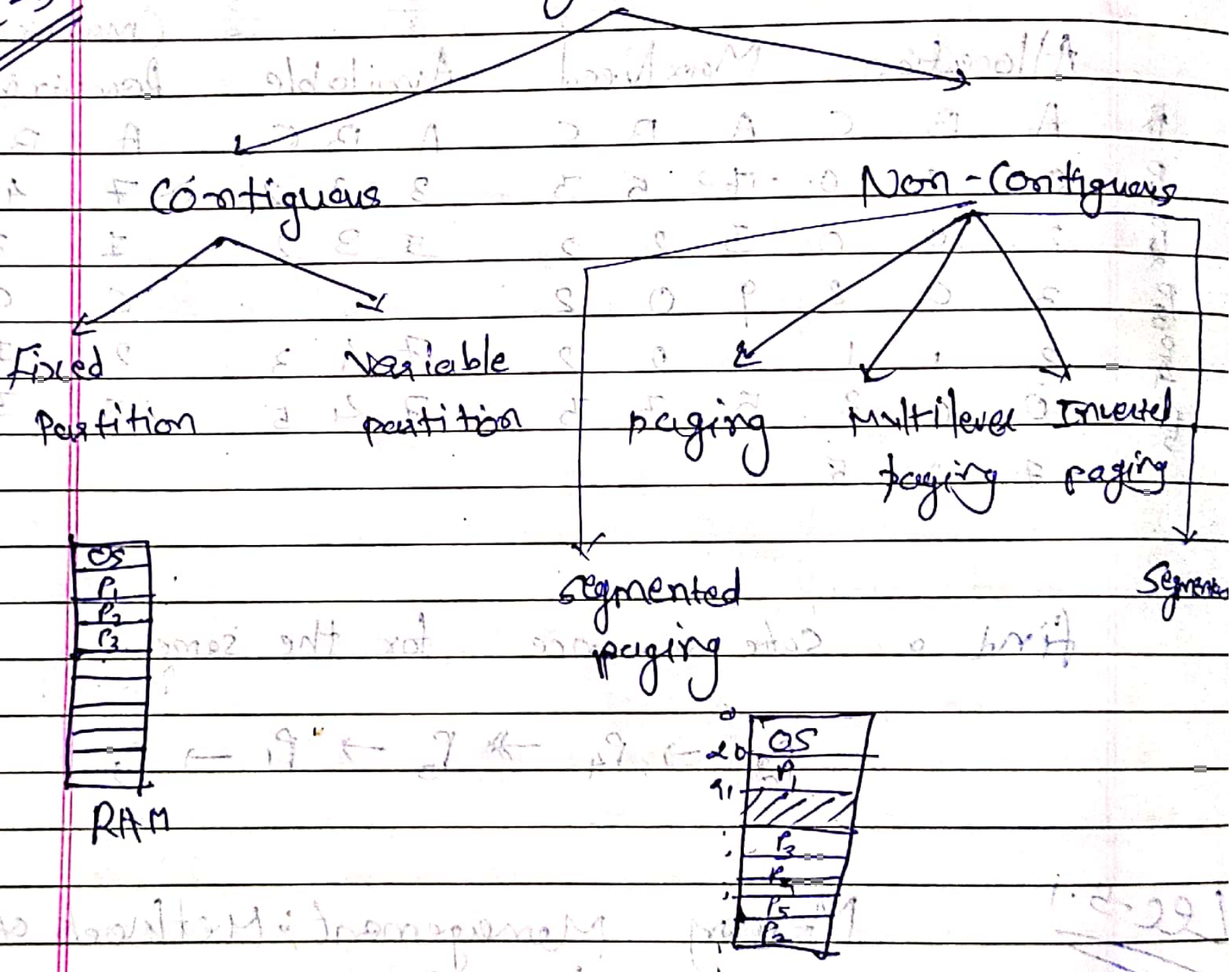


DATE: / /
PAGE: 29

→ Memory large array of bytes
→ By putting maximum number of processes into RAM or increasing size of RAM we one can increase the performance of system.

Lec-5.2

Memory Management Techniques



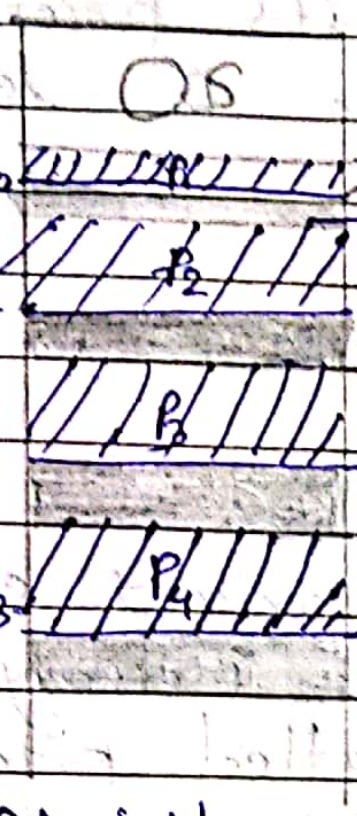
Lec-5.3

Fixed Size Partitioning

p = 4 MB

→ Start filling main memory from up to down
If found empty slot put process over there

DATE: / /
PAGE: 30



$P_1 = 2MB$

Internal fragmentation = 2MB

1) Internal fragmentation
2) Limit in Process size (32 MB) can't be added)
3) Limitation in multiprogramming
4) External Fragmentation
↓
new process came after all slots ~~fulled~~ filled then ~~last~~ ~~pro~~ that process's size is external fragmentation

→ Used in ~~fixed~~ main frames.

Variable size fixed partitioning

	4MB
	6MB
	8MB
	8MB

fixed size fixed partitioning

OS	
P_1	2MB
P_2	4MB
P_3	8MB
P_4	2MB
P_5	2MB
P_6	4MB

Lec-5.4

Variable Partitioning or Dynamic Partitioning

1) No internal fragmentation
2) No limitation on no. of process
3) No limitation on Process size

→ What if P_2 completes its process and leave in between. and also P_4 leave in between.

Now we have 6 MB size empty in RAM.

→ P_7 = 6MB come but we can not allocated due to contigeous memory allocation this is called External fragmentation.

⇒ Solution is compaction. Copy process from below and put them to empty cells which are above them.

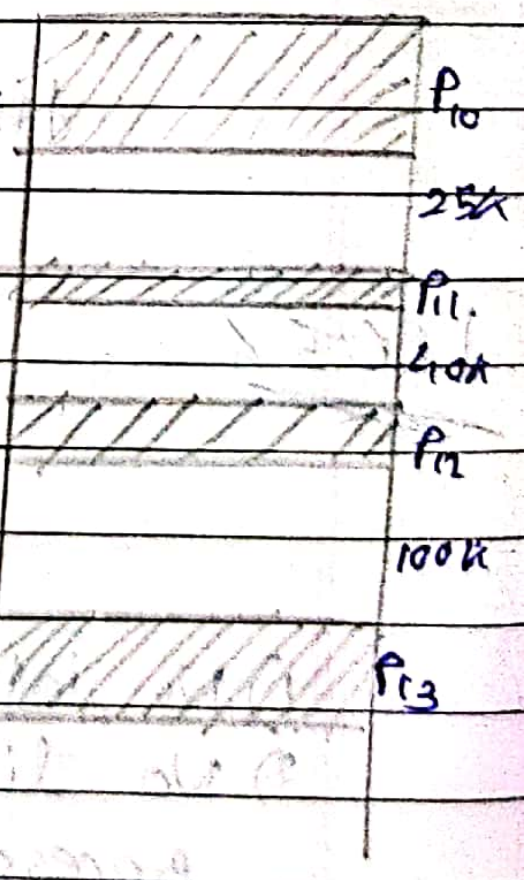
Lec-5.5 Algorithms For External fragmentation

simple/fast

* 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
  - internal frag. will be less
  - slow

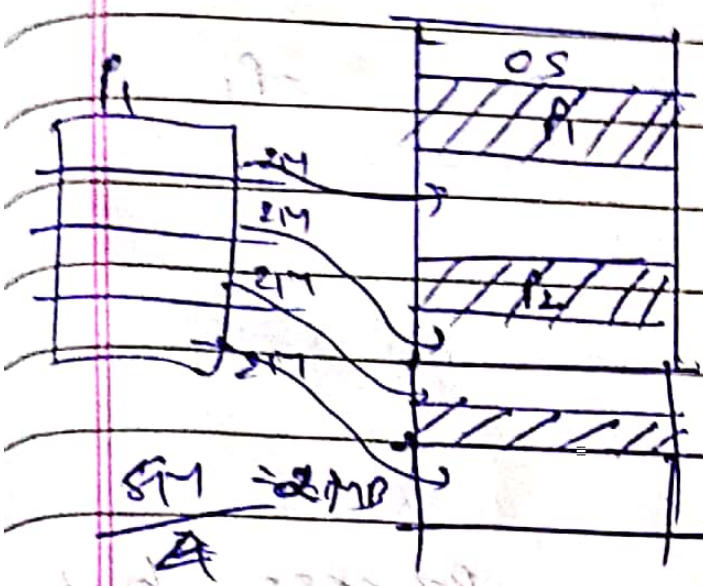


DATE: / /
PAGE: 32

worst fit: Allocate the largest hole
↳ slow

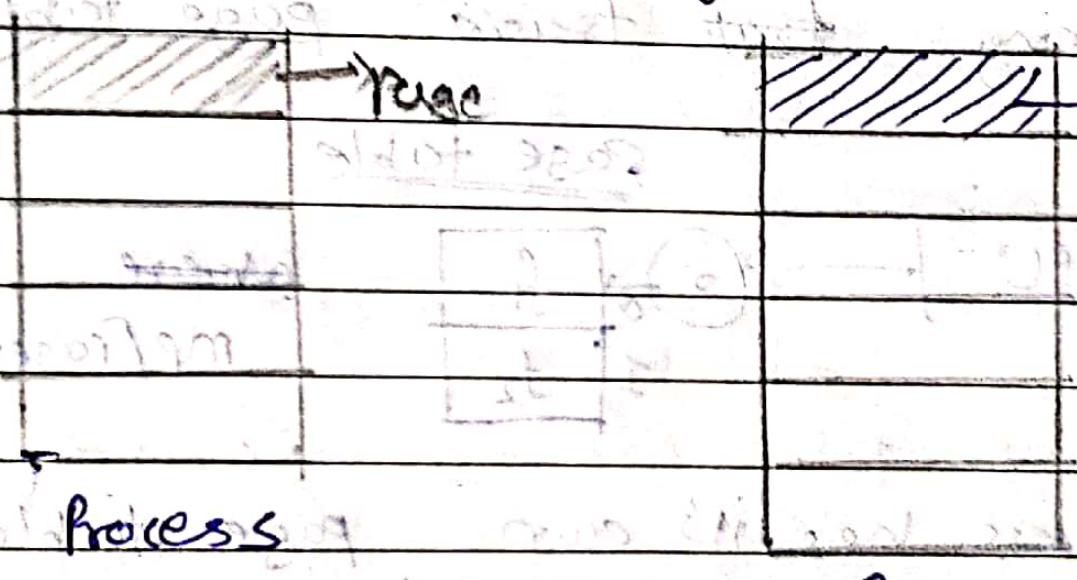
Lec-55

Non Contigeous Memory Allocation



→ check whole memory and decide how much size is left then divide whole process into parts then Allocate it into RAM.
→ costly operation.

Paithan Paithan use this



Pagesize = frame size

every fraction of process can fit into frame

DATE: / /
PAGE: 33

Lec-59

Paging :

Pageno.	Bytes	
0	0	1
1	2	3

P_1 (process)

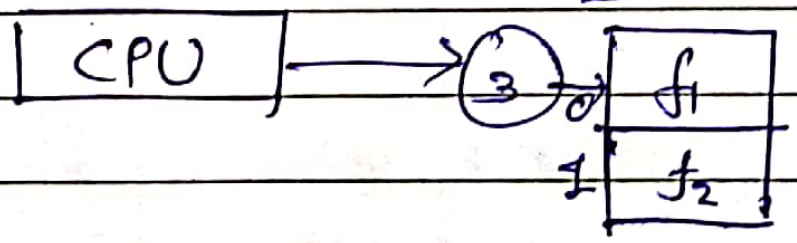
0	0	1	
1	2	3	
2	4	5	P_0
3	6	7	
4	8	9	P_1
5	10	11	
6	12	13	
7	14	15	

M.M size = 16 Bytes
frame size = 2 B
No of fra = $\frac{16}{2}$ = 8 Bytes

→ CPU wants to access Process but don't know where it is stored into Main memory.

→ For making ~~task~~ tasks - page table is used.

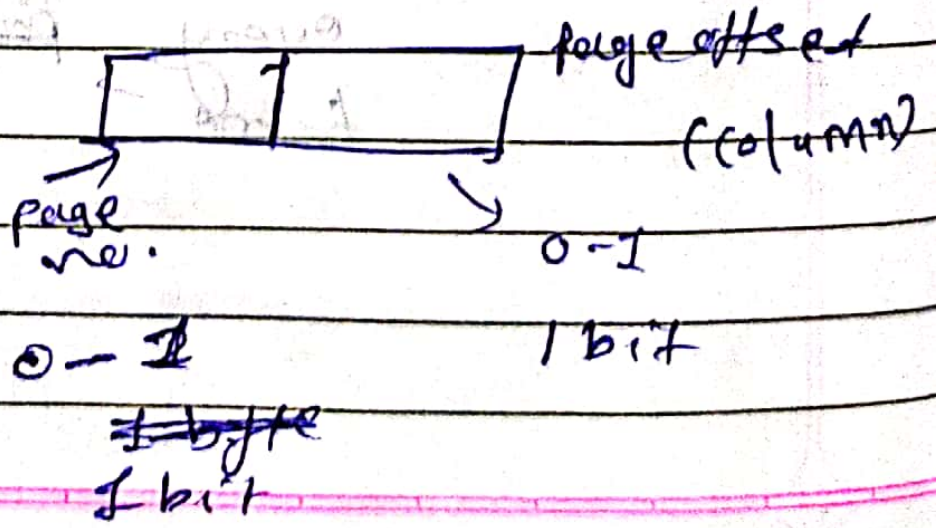
Page table



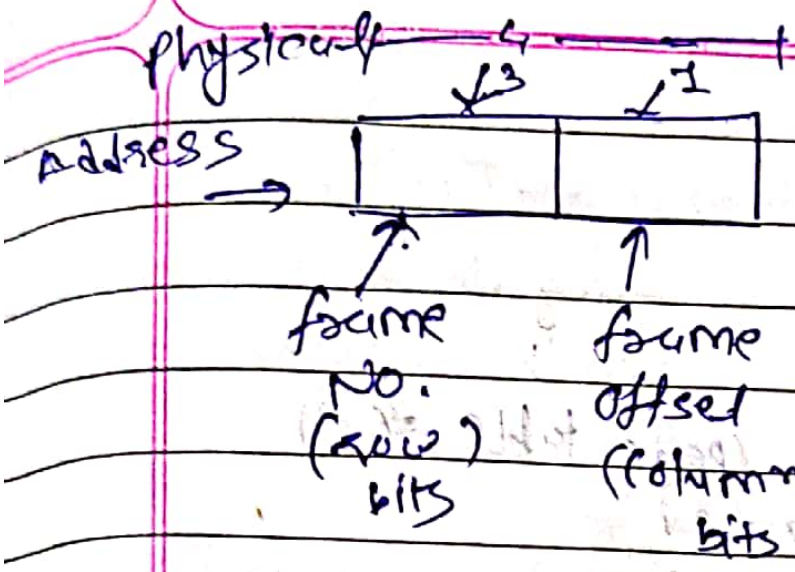
mp[pageno] = frame

→ every process has it's own page table.
→ This thing is handled by Memory management Unit.

Logical Address ⇒ (CPU uses)



DATE: \ \
PAGE: 34



CPU generates Number → Logical Address e.g 3 → check into Page table

Pageno.	Pageoffset
1	1

0	f_2
1	f_4

Check into physical Address

eg

framene.	frame offset
100	1

1001

take 9 address detail and send it to CPU

CPU processes on detail.

frameoffset = pageoffset

Lec-5.10

LAS = 4 GB
PAS = 64 MB
Page size = 4 KB

DATE: / /
PAGE: 35

No. of pages = ?
No. of frames = ?

No. of entries on page table = (?)
Size of page table = ?

total = $2^6 \times 2^{20} = 2^{26}$
page size = $2^2 \times 2^{10} = 2^{12}$

No. of pages = $2^{(26-12)}$
frame no. = 14 = 2^{14}

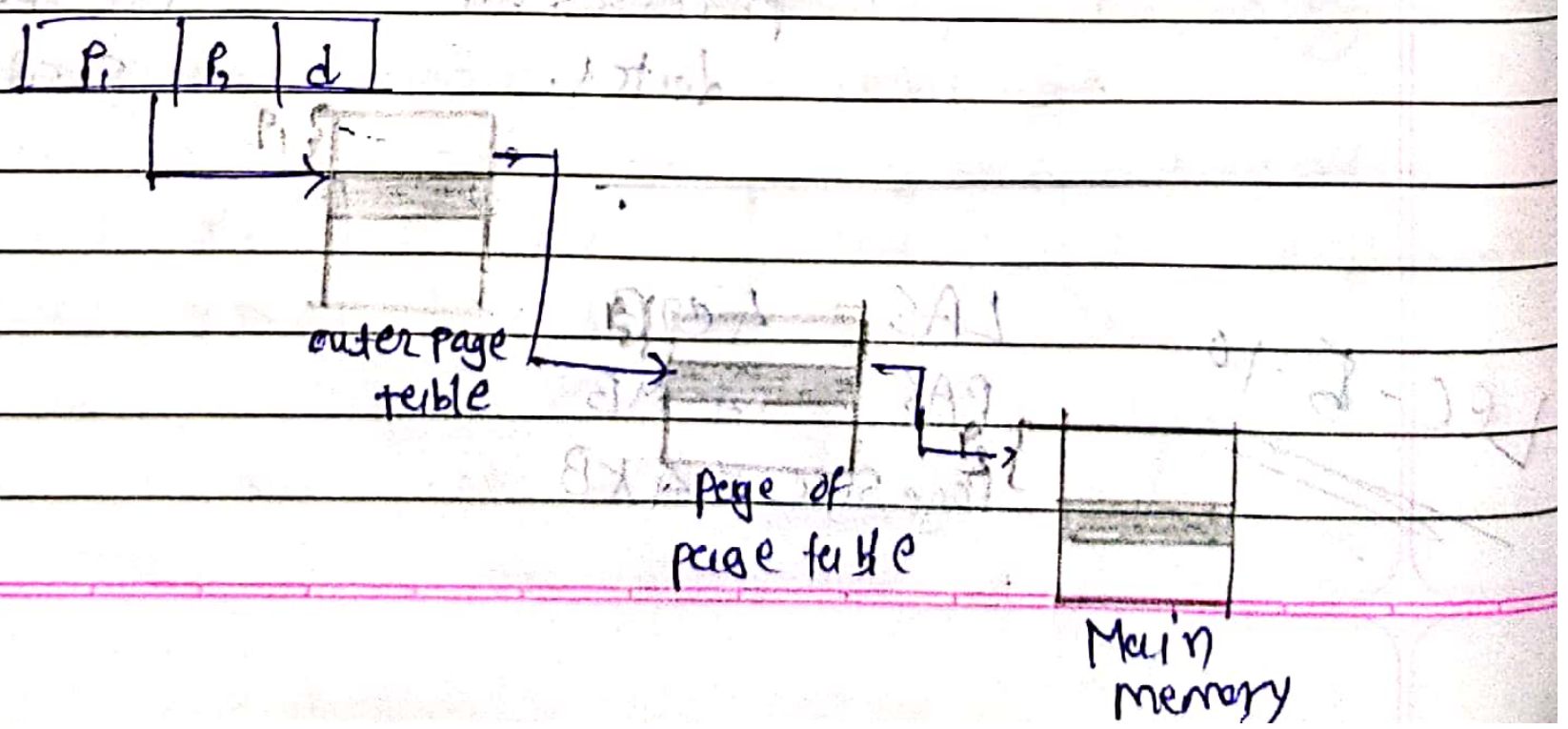
total = $2^2 \times 2^{30} = 2^{32}$
page no. = 32 - 12 = 20 = 2^{20} P

No. of pages = No. of entries = 2^{12}
size = 14×2^{20}

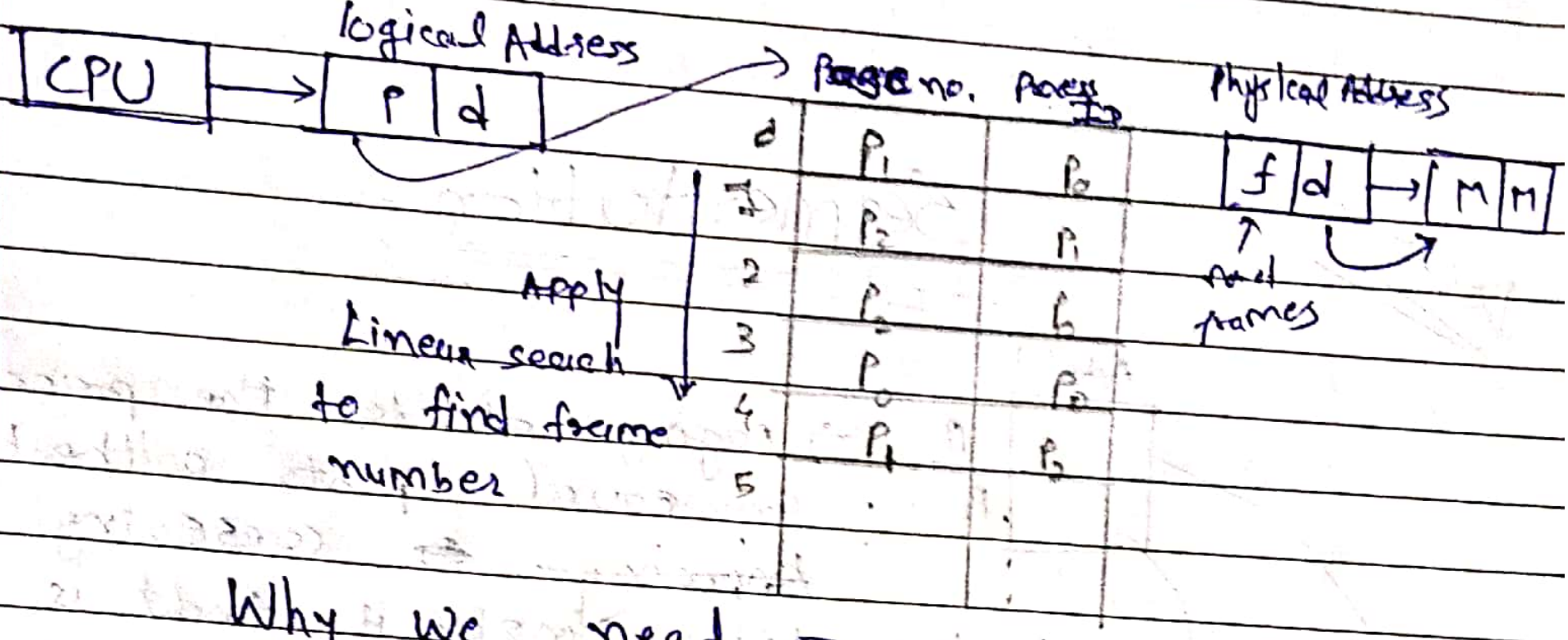
Lec-5.13

Address - Translation Scheme.

Page table size > frame size



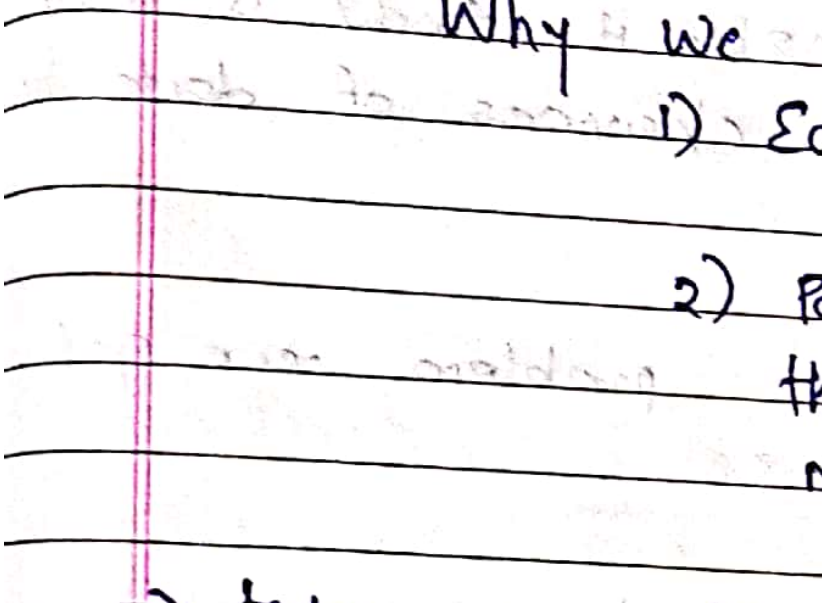
Inverted Paging

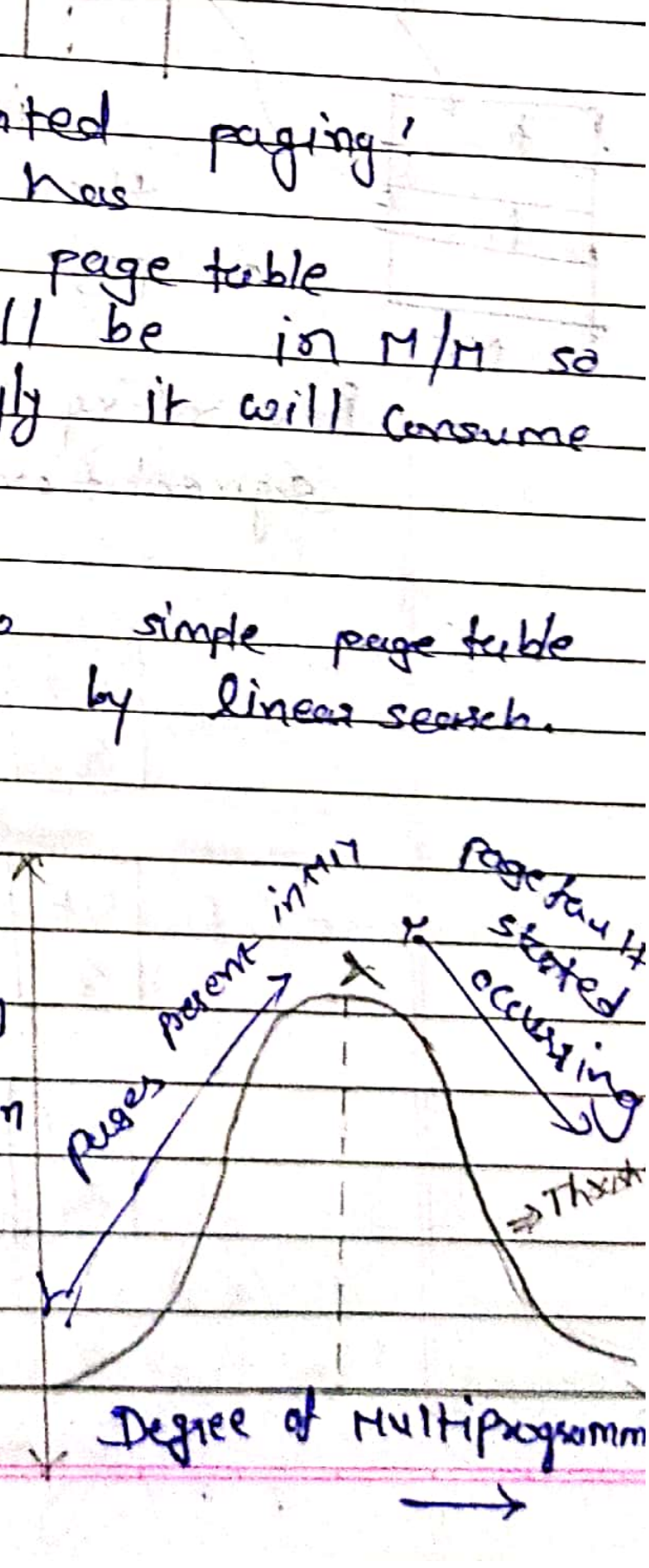


Why we need Inverted paging?

1) Each process has its own page table
2) Page table will be in M/M so that unnecessarily it will consume more space.

→ take more time compared to simple page table as searching is performed by linear search.





pagefault: whenever page not found in ~~Main~~ Main memory

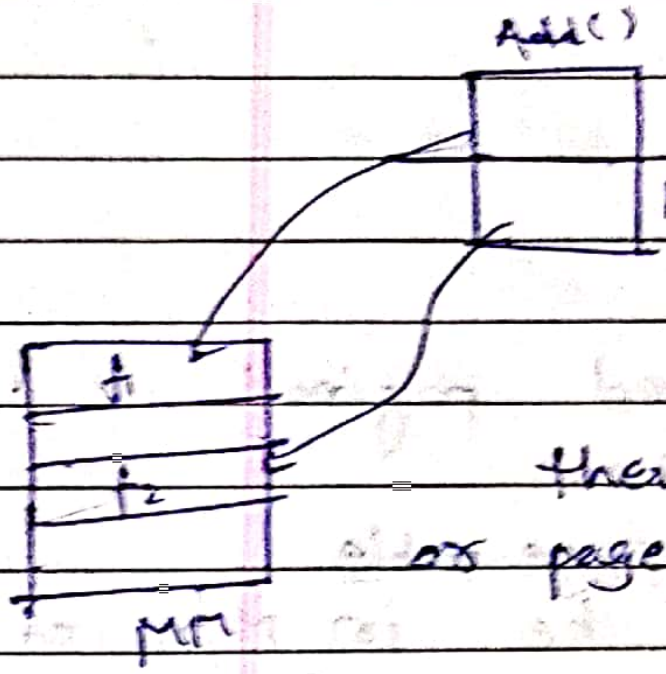
DATE: / /
PAGE: 27

solution:-

1) Increased main Memory size
2) Decrease Long term Scheduler speed (dispatch less process in Ready state).

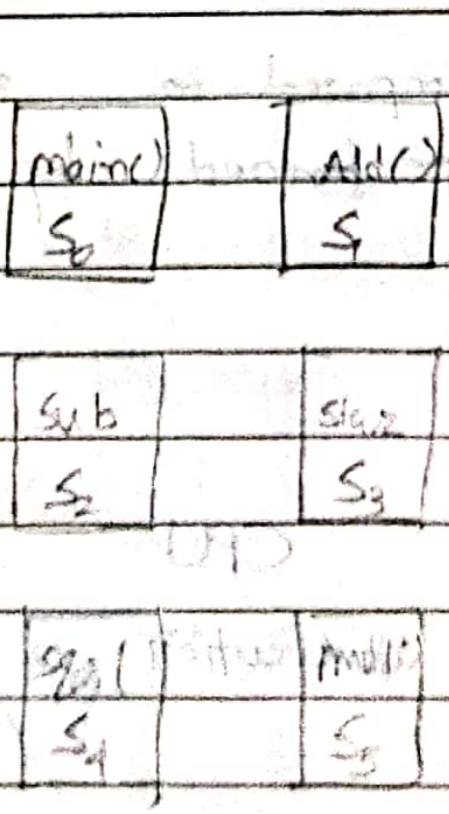
Dec-5-19

Segmentation:-



→ paging divides the process in equal parts without ~~knowing~~ knowing what is going inside it. that is why there are higher chances of data loss or page fault.

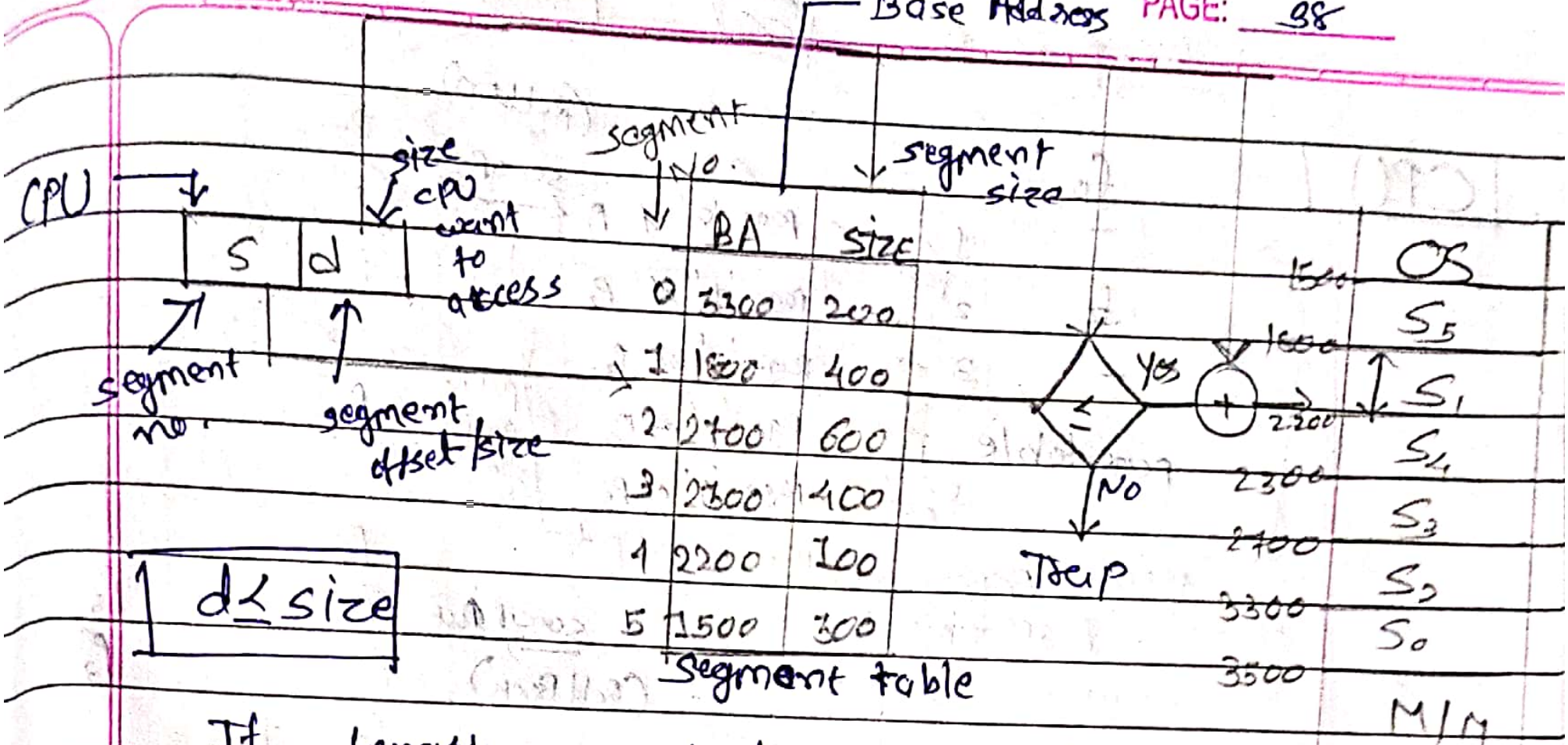
→ Preventing from this problem we get segmentation.



→ divides code or process into segments as per requirements.
→ size can be different of all segments.

DATE: / /

PAGE: 38



If length asked by CPU is smaller or equal to size of segment then it will execute else it will give an error or segment fault.

Lec-5.18

Overlay

→ main memory size < process size

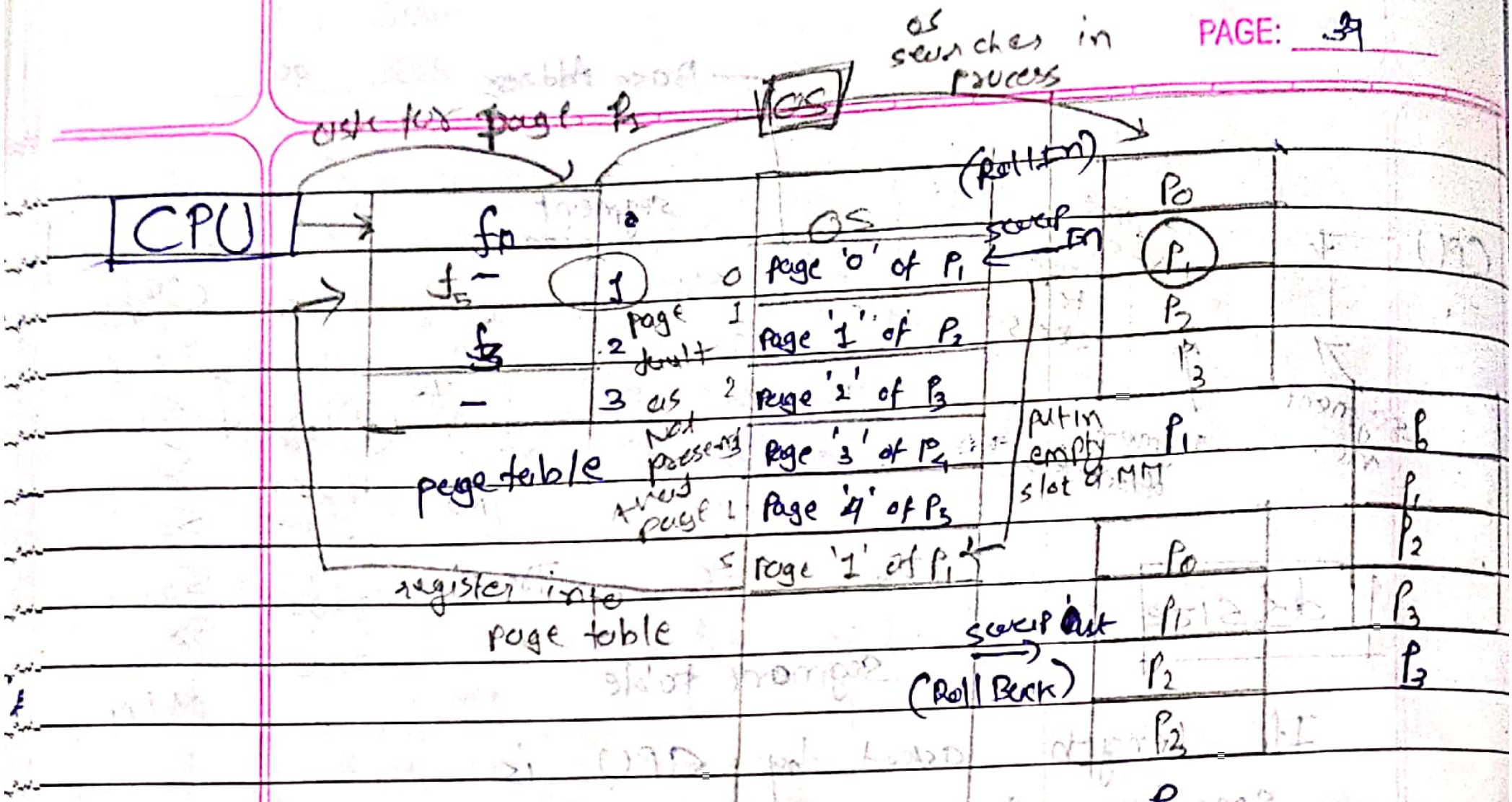
→ used in embedded systems

→ load / accomodate ~~data~~ Pages ~~processes~~ into main memory as per user requirement.

Lec 5.19

Virtual Memory:

→ It creates illusion that all pages are there in main memory.



Effective memory Access time

$$\text{Page fault EMAT} = p\left(\begin{matrix}\text{Page fault}\\ \text{service time}\\ \text{(milliseconds)}\end{matrix}\right) + (1-p)\left(\begin{matrix}\text{memory}\\ \text{manag.}\\ \text{Access time}\\ \text{(nano seconds)}\end{matrix}\right)$$

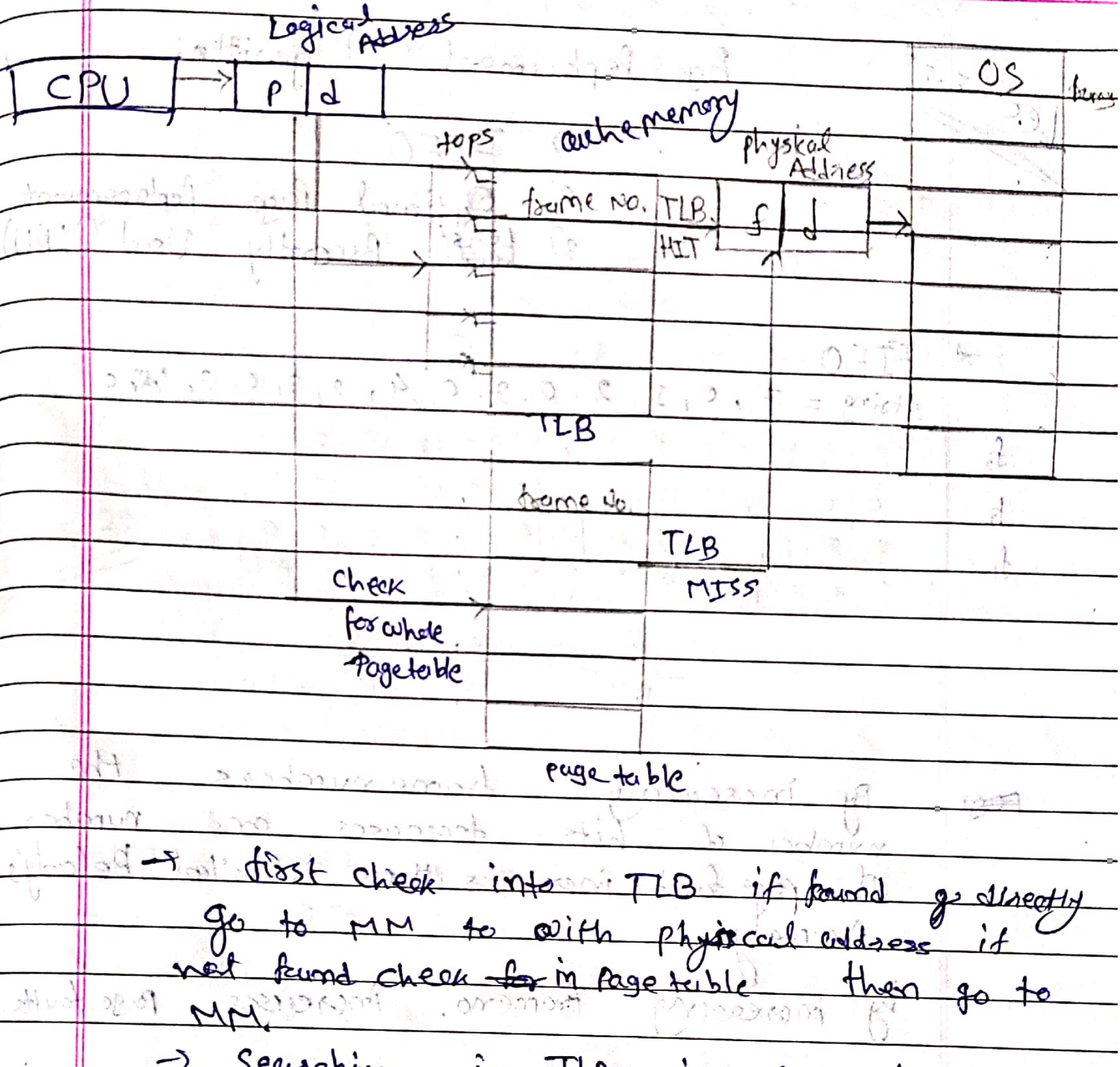
Let - 5.20

Translation Lookaside Buffer:

⇒ It's an effort to decrease overall searching time.

⇒ We search whole memory for 2 times
1) while searching for page table
2) while searching for page it self.

DATE: / /
PAGE: 40



→ first check into TLB if found go directly go to MM to with physical address if not found check ~~for~~ in Page table then go to MM.

→ Searching in TLB is fast to since cache is fast compared to ~~the~~ DRAM.

$$EMAT = \underset{\text{Ratio}}{\text{Hit}} (TLB + x) + \underset{\text{Ratio}}{\text{miss}} (TLB + x + x)$$

Hit ratio must greater than miss to have high benefit.

DATE: / /
PAGE: 41

Lec-5.22

Page Replacement Algorithm:-

1) FIFO
2) Optimal Page Replacement
3) Least Recently Used (LRU)

FIFO

String = 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 1, 2, 0

f_3			1	1	1	3	0	0	0	3	3	3	3	2	2
f_2		0	0	0	0	3	3	3	2	2	2	2	1	1	1
f_1	7	7	7	2	2	2	2	4	4	4	0	0	0	0	0
	*	*	*	*	Hit	*	*	*	*	*	Hit	*	*	Hit	

$$\frac{\text{hit}}{\text{miss}} = \frac{3}{15} = \frac{1}{5}$$

By increasing frame numbers the number of hits decreases and number of page fault increases this is called Belady's Anomaly.

by increasing frame no. increases, page faults.

Lec-5-24

Optimal Page Replacement

(Replace the page which is not used in longest dimension of time in future)

Reference String → 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

go to future take last present from already present process

DATE: / /
PAGE: 112

f_4				2	2	2	2	2	2	2	2	2	2	2	2	2	7	7	7
f_3			1	1	1	1	4	4	4	4	4	4	4	4	4	4	4	4	4
f_2		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
f_1	7	7	7	7	3	3	3	3	3	3	3	3	1	1	1	1	1	1	1
	*	*	*	*	*	hit	*	hit	hit	hit	hit	*	hit	hit	hit	*	hit	hit	

(farthest entry in right direction from current)
If there is no demand in future you can ~~select one~~ replace any of them.

5.25
LRU (Least Recently Used)
(Replace the least recently used page in Past)

f_1				2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
f_3			1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
f_2		0	0	0	0	0	0	4	4	4	4	4	4	1	1	1	1	1	1
f_1	7	7	7	7	7	3	3	3	3	3	3	3	3	3	3	3	3	5	7
	*	*	*	*	hit	*	hit	*	hit	hit	*	hit	hit	*	hit	hit	hit	*	+2 hit

7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

←

go into past and check which is the least used from present into frames.
(farthest entry from current in left direction)

5.26
Most Recently Used
(Repeat Replace the most recently used page in past)
(nearest entry in left from current)

DATE: / /
PAGE: 43

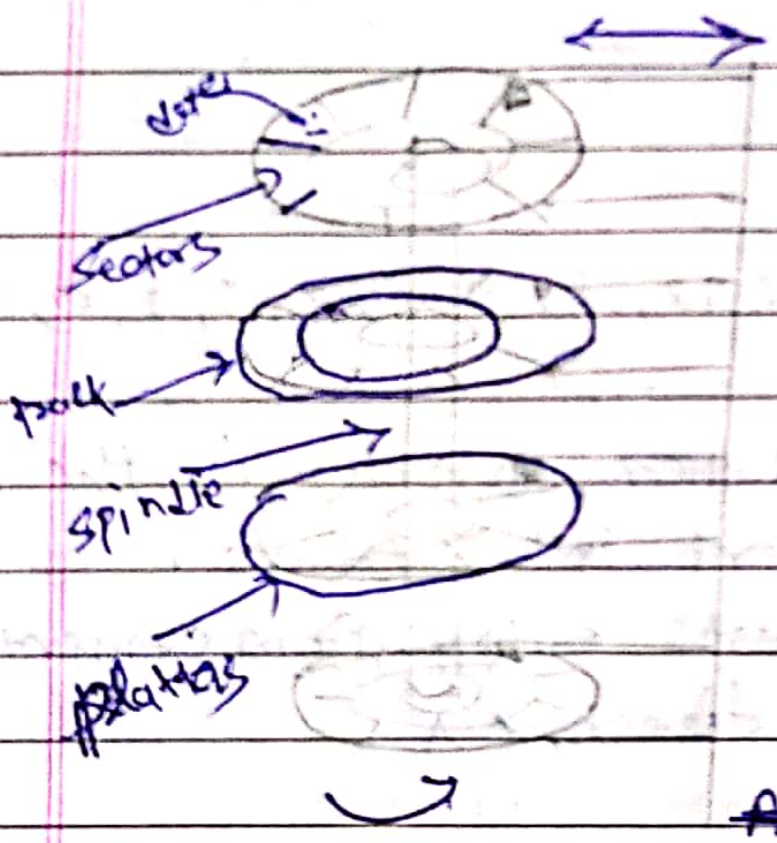
7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

f_2				2	2	2	2	2	2	3	0	3	2	2	2	0	0	0	
f_1			1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
f_0		0	0	0	0	0	0	4	4	4	4	4	4	4	4	4	4	4	
d_1	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	
	miss	miss	miss	miss	miss	miss	miss	miss	miss	hit	hit	hit	hit	hit	hit	miss	hit	hit	

f_2	0	0	hit = 8
f_1	1	1	miss = 12
f_0	4	4	
d_1	7	7	
	hit	hit	

Lec-6.1

Hard Disk Architecture :-



platters → Surface → tracks
→ ~~data~~ sectors → data

~~Architecture~~

Actuator

ASM

DATE: / /
PAGE: 44

* Top Interview ~~Question~~ Questions :

1) Process States & PCB

→ A process Control Block (PCB) is a data structure used by computer operating systems to store all the information about a process.

→ Process states : New, Ready, Running, Wait / Block, Suspended Ready, Suspended Block / wait, Terminated

→ A thread has three : Running, Ready, Blocked states.

2) Mutex vs Binary semaphore

- A Mutex is different from a semaphore as it is a locking mechanism while a semaphore is a signal mechanism.
- A Binary semaphore can be used as a Mutex but a Mutex can never be used as semaphore.

3) Process vs Thread

4) Monolithic Vs Microkernel

→ The microkernel runs user and kernel services in different adress spaces. On the other hand, the monolithic kernel runs both kernel and user services in the same address space.

→ In Monolithic Microkernels, only essential processes like IPC, memory mgmt, and scheduling take place in kernel space.

5) System call ?

6) Demand paging

→ It says keep all pages of the frames in the secondary memory until they are required. It means do not load any page in main memory until it is required.

7) chmod command

→ changemod command

8) Paging vs Segmentation

Paging → fixed size division

Segmentation → division into variable sizes.

DATE: \ \
PAGE: 46

10) Internal segmentation vs External Fragmentation

11) Preemptive vs Non-preemptive Scheduling

12) What is deadlock.?

13) Multiprogramming vs Multitasking

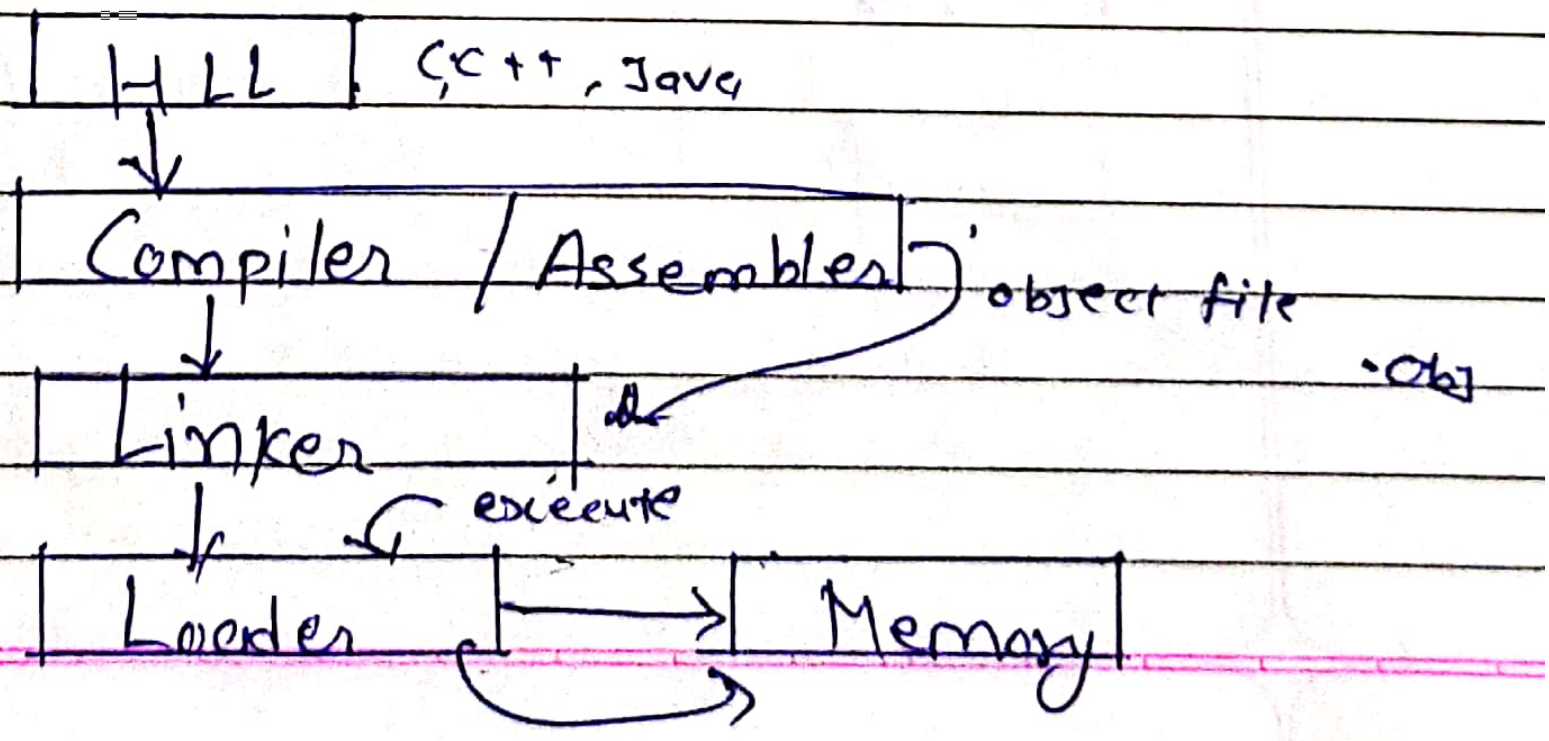
14) Spatial Locality Vs Temporal locality

→ Spatial locality means all those instructions which are stored nearby to the recently executed instruction have high chances of execution.

→ Temporal locality means that a instruction which is recently executed have high chances of execution again.

15) Virus, Worm, Trojan, Malware, Spyware

Linker & Loader :-



DATE: \ \
PAGE: 47

Object file :

- Header
- Text Section (code collection)
- Data Section (variables)
- BSS Section (uninitialised variables)
- Symbol Section :
- ~~Redo~~ Relocation Information Section (gives a relevent value)
- Debugging Information :

Scanned with CamScanner