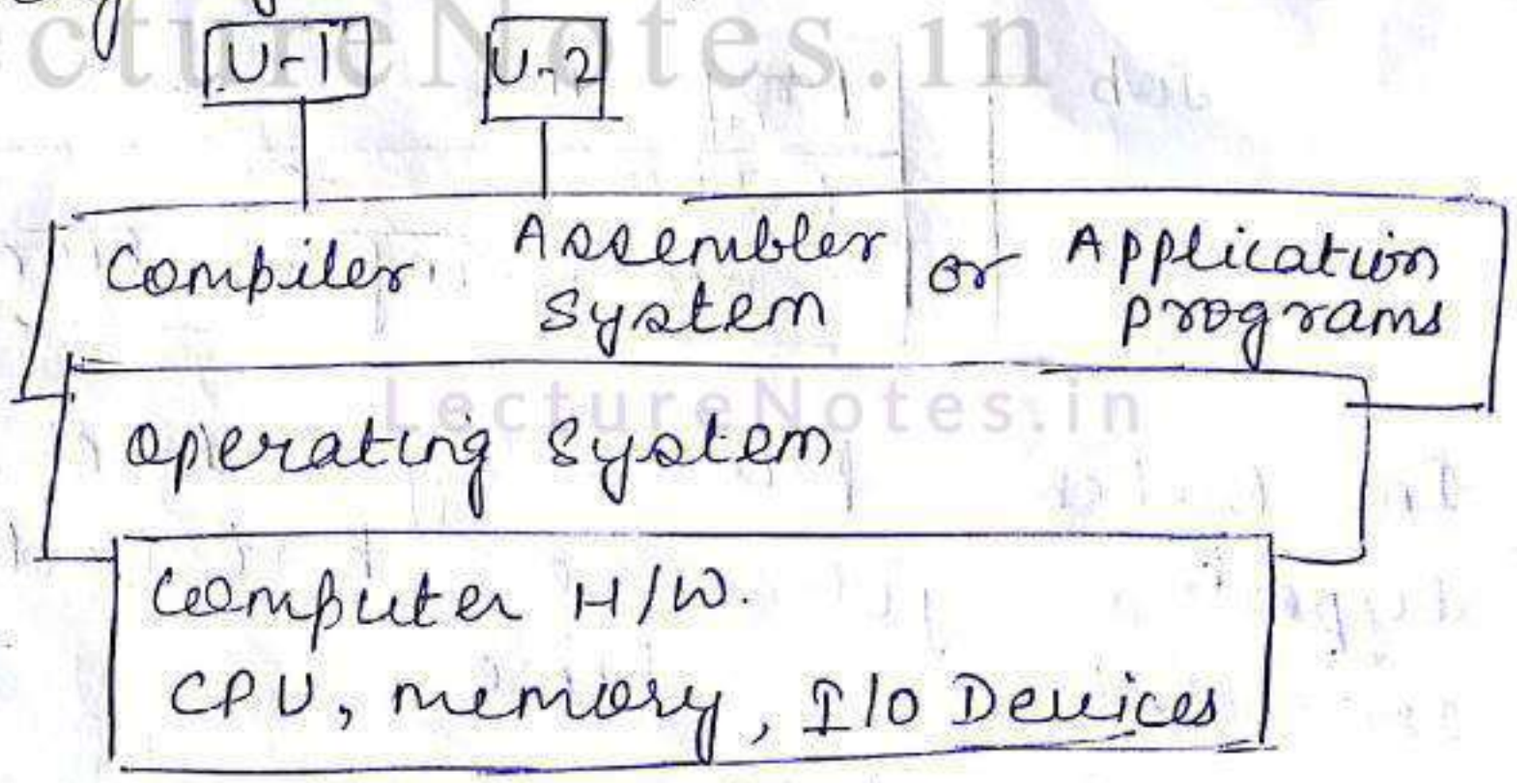
11/7/17

Resource manager / allocator.

1.) Introduction
2.) Process management :- storage and execution of program.
3.) process synchronization
4.) Deadlock : tackling problems.
5.) Memory management
6.) File management.

- An Operating system (OS) is a set of program that act as an interface between the user and the computer hardware.
- The purpose of OS is to provide an environment in which a user can execute programs in a convinient and efficient manner.
- OS is a resource manager or Resource allocator. examples of resources are: CPU, memory, I/O Devices
- OS also behaves as a service provider.

→ what operating system do?



① user-view
↳ PC: maximise work.
↳ mainframe (-MFC)
: maximise resource utilization

② System view
- Resource allocation

Scanned by CamScanner

12/7/17

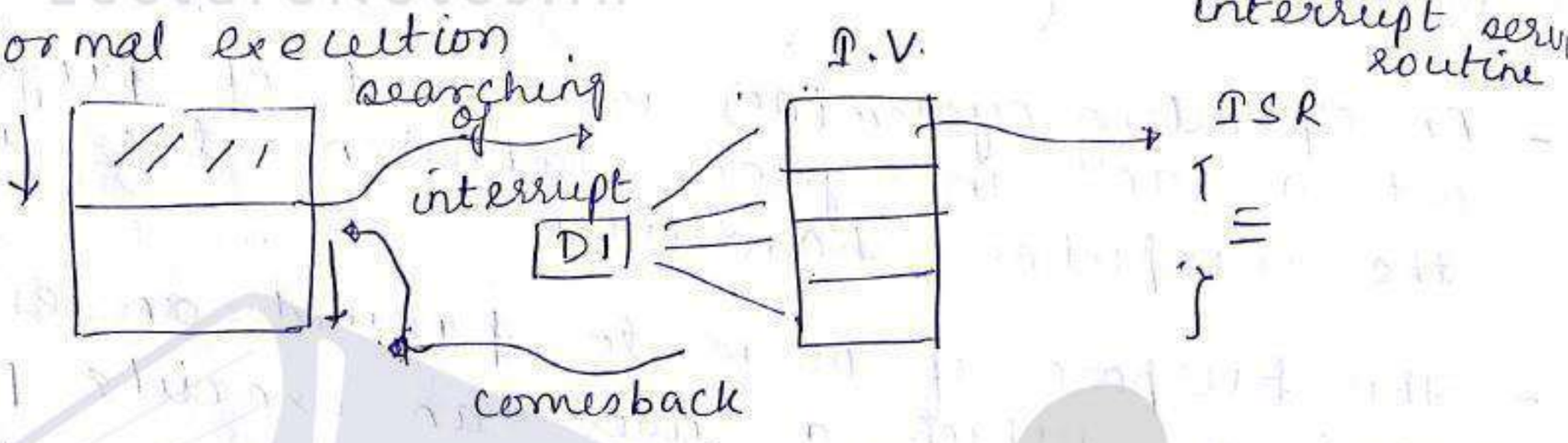
→ computer system operation.

i) initialize : All devices are connected or not p
ii) OS will be loaded.

↳ kernel will be loaded in RAM.
↳ (core part of processor)

Interrupt : unwanted signal

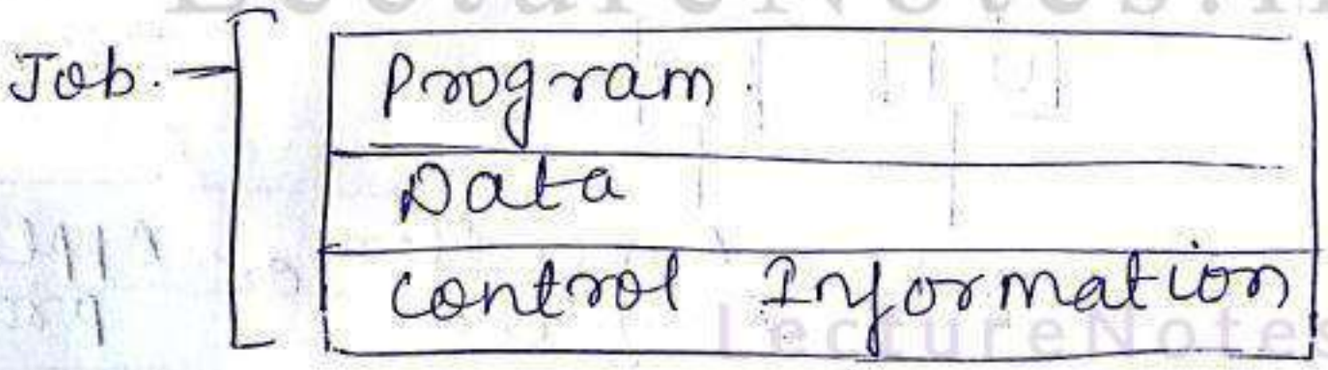
Interrupt vector : interrupts are stored in array.



system call : deviating from main program and executing interrupt and again coming back and executing main program is called system call.

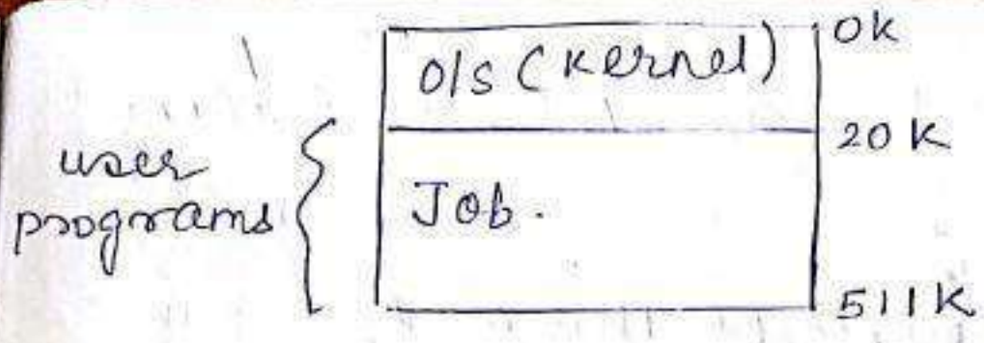
→ Evolution of operating system.

i) Batch processing system.



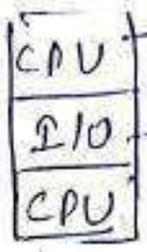
- In Batch processing technique similar types of jobs are batched together and executed at a time.
- The major task of OS was to transfer control automatically from one job to the next job.

Scanned by CamScanner



→ memory layout of batch processing system.

Job < CPU bound.
I/O bound.

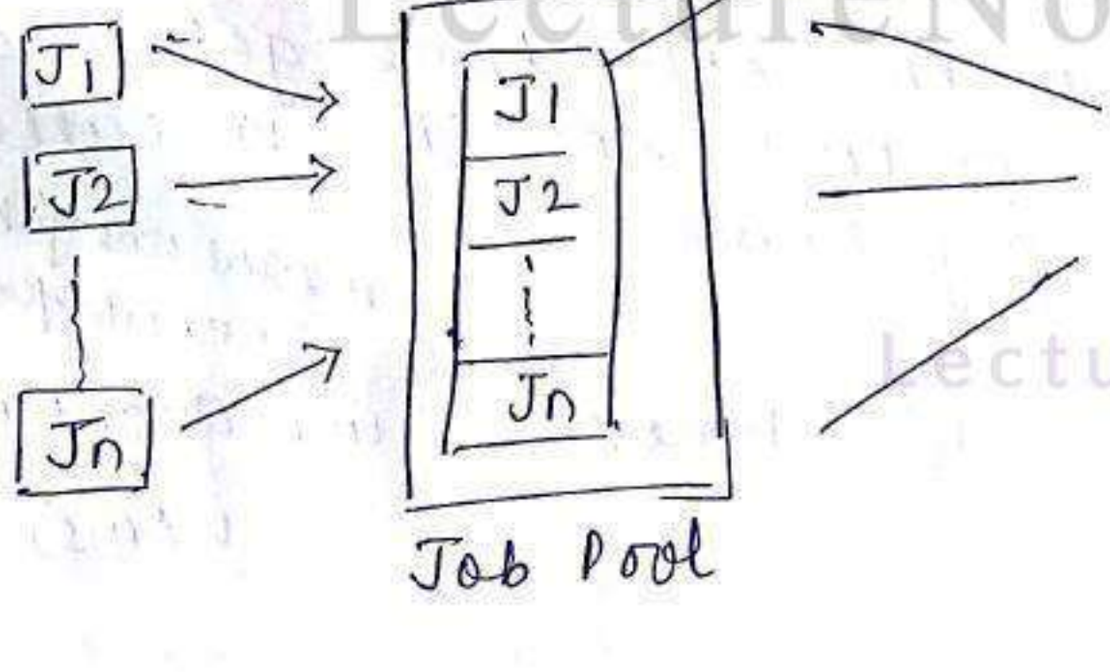
eg: 

→ CPU bound (CPU is busy)
└ CPU is idle I/O is busy.

CPU should always be busy so as to increase performance.

- Advantage of Batch processing system
  i.) processing is faster
- Disadvantage
  i.) CPU is often idle.
  ii.) user cannot interact with the job when it is executing.

ii.) Multiprogramming System :-
To avoid (CPU is often idle) disadvantage this system was introduced.
Multiprogramming is a technique to execute multiple programs simultaneously by a single processor.

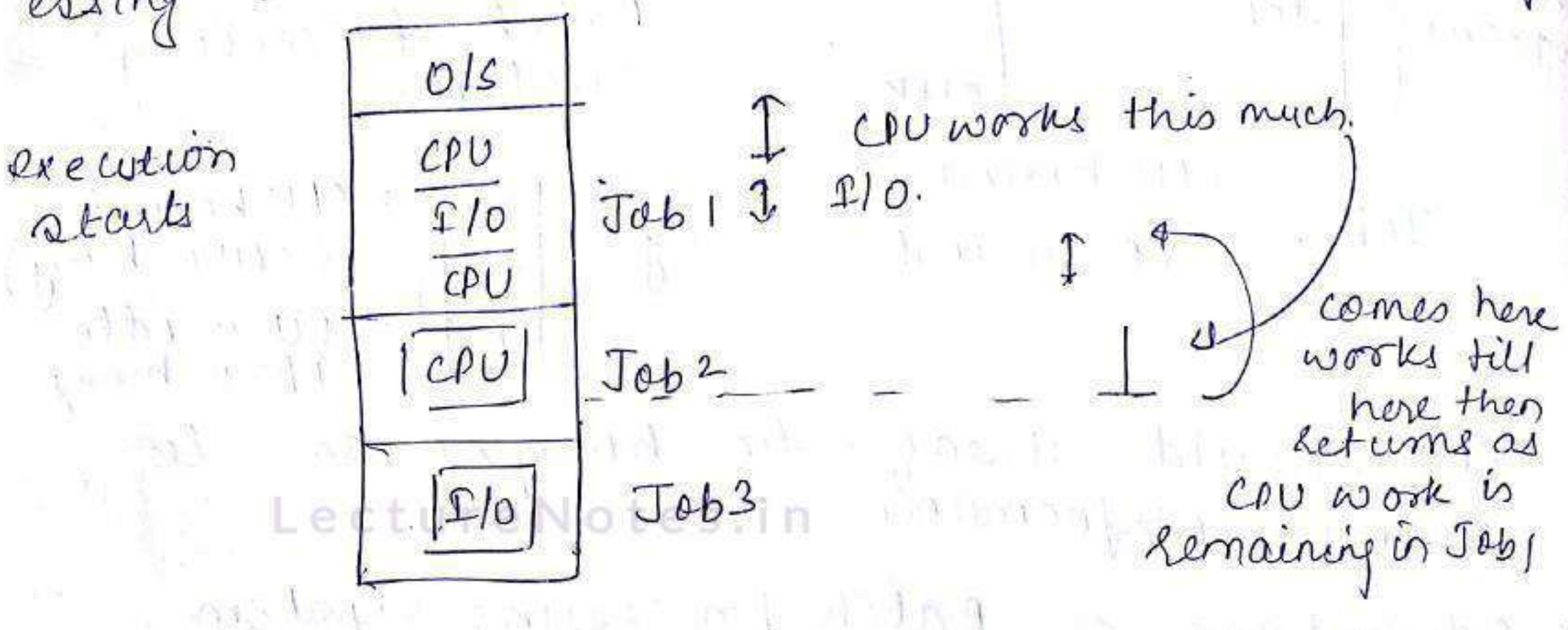


Ready Queue Queue which contains jobs and those jobs are ready for execution. CPU can any time start its execution.

Job Queue : Here Jobs may or may not be ready.

Scanned by CamScanner

Selecting the job from Ready Queue and processing it in CPU is based on CPU scheduling.



→ Advantage

i.) efficient memory utilization as multiple jobs are stored.
ii.) CPU is never idle
iii.) Throughput of the CPU may also increase
iv) waiting time is limited.

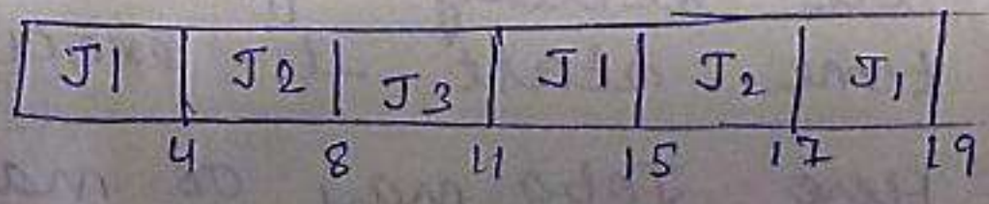
18/7/17

(iii.) Time-sharing operating system

- Time-sharing or multitasking is a logical extension of multiprogramming
- Multiple jobs are executed by the CPU switching between them
- In this method the CPU time gets shared by different programs so it is called time sharing system

(means one job can execute for max. of 4ms: Time Quantum 4(ms))

Program	Execution time.
J1	10.
J2	6.
J3	3



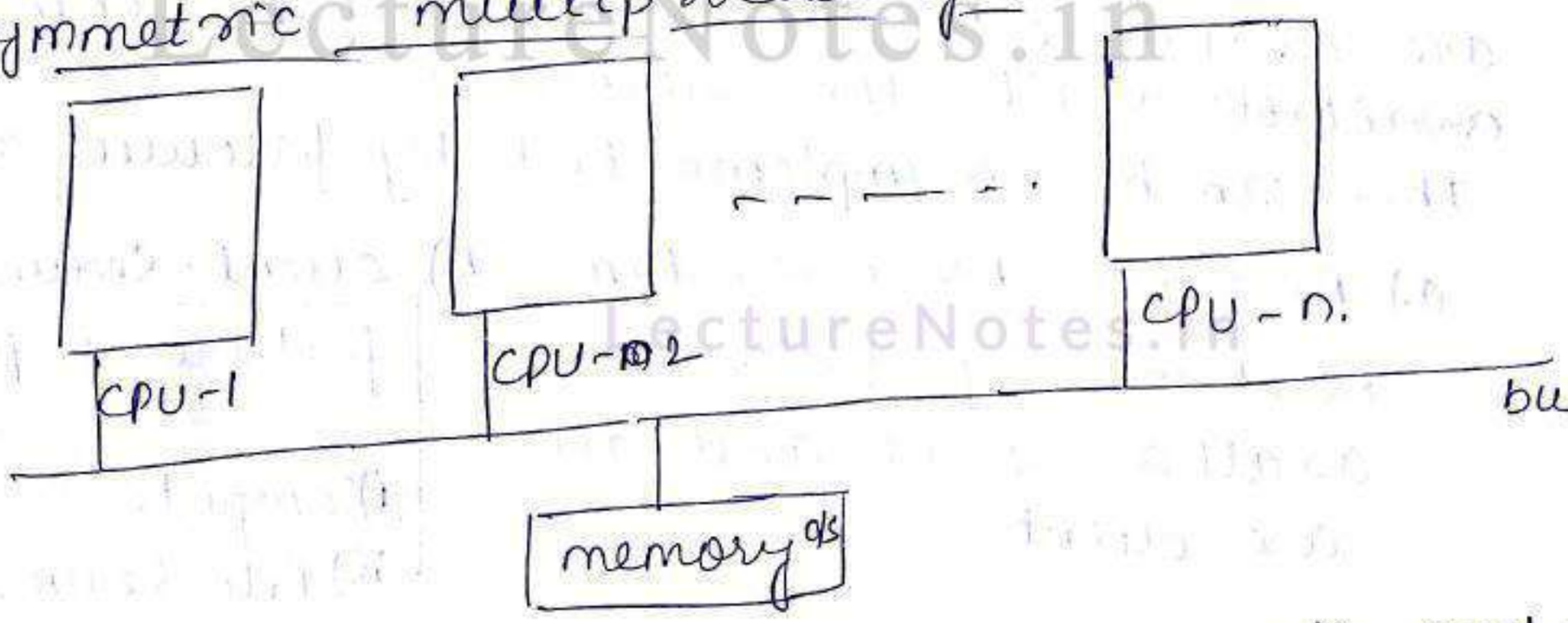
Scanned by CamScanner

- Advantage:
i.) user can interact with the job when it is executing.
ii.) The efficient use of CPU i.e CPU utilization.

IV.) MULTIPROCESSOR SYSTEM or PARALLEL SYSTEM or
a.) Tightly coupled system.
Multiprocessor system have more than one processors will share computer bus, memory, peripheral devices etc. then this type of system are called multiprocessor system.

Advantage
i.) increased throughput (programs executed in particular time period: throughput).
ii.) It is economical. {(eg: 5 CPU in 5 comp. & 5 CPU in one)
iii.) increased reliablity.
a.) The ability to continue providing service proportional to the level of surviving hardware is called graceful degradation.
{graceful degradation
max. no. of failure ϕ in which system can operate efficiently.
b.) some systems go beyond the graceful degradation are called fault tolerance.

There are two types of multiprocessor system.
i.) Symmetric multiprocessing (smp)



- Each processor runs a copy of the o/s and these copies communicate with one-another when needed
- In SMP all the processors are peers

Scanned by CamScanner

(ii.) Asymmetric multiprocessing system:

- In this case each processor is assigned a specific task
- A master processor controls the whole syste and other processors follow the master processor's instructions
- the master processor schedules and allocat the work to the slave processor.

b.) Distributed system or loosely coupled system.

⇒ In Distributed systems, the processors canno share memory, clock and peripheral devices. each processor has its own local memory and they are communicated among themsel ves through high speed buses (LAN/WAN/Internet).

Advantage:

i.) Resource sharing.
ii.) computation speedup.
iii.) Reliability.

1917|17

v.) Network operating system.

A Network has a number of nodes and a node has many resources. When we want to use the resources, we need to know where the resource are available and connect that machine to ~~availabl~~ avail the resources.

This can be implemented by following architectures

a.) Peers-to-Peers System:
No chance of failure as all are server and all are client.

b.) Client-Server system.
failure is possible
i.) Compute-server system.
ii.) File-server system.

vi.) Real-time operating system.

- A Real-time system is used when a rigid time requirement have been placed on the operation of a processor or the flow of data

Scanned by CamScanner

- A real-time system has well-defined fixed time constraint. Processing must be done within the defined constraint otherwise the system will fail.
- There are two types of real time system.
  a.) Hard real time system: This system guarantees that critical tasks are completed on time.
  b.) Soft real time system: It means that only the precedence and the sequence for the task operations are defined.

→ COMPONENTS OF OPERATING SYSTEM (LS) Pg-24.

i.) Process management:
program in execution is called a process.
program is passive entity whereas process is an active entity.
when a process is executed/created, a resource is allocated to it.
when execution is over the process releases all the resource.
One which manages all program and processes is done by process management

ii.) Main-memory management
main memory: large collection of semiconductor cells capable of storing one bit of information either 0 or 1
each cell has a physical address.
Data can be stored/accessed using that physical address.

CPU generates operation.
CPU → logical address
↓
memory management unit
↓
physical address

In main memory randomly we can access data and time to access any address is same.

iii.) File management
File: collection of information.
Folder: collection of file. All Folders make: superfolder type

File
- character
- numeric
- alphanumeric
- Binary

Scanned by CamScanner

20/7/17

4. I/O system management:

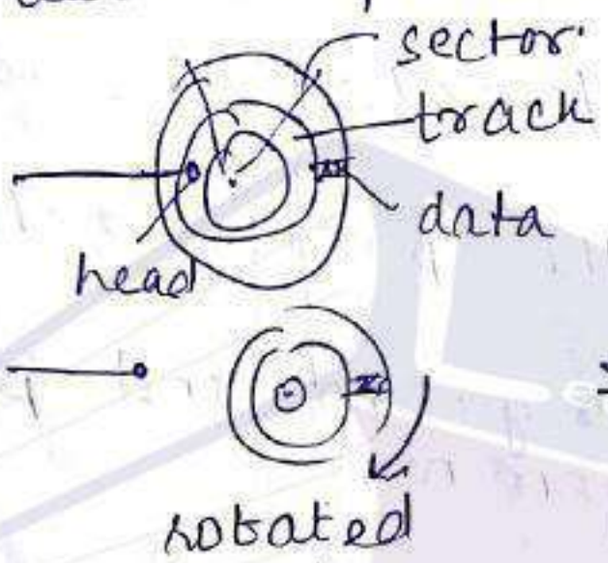
I/O are electromechanical device. It cannot be connected directly due to speed mismatch. so some buffer/cache memory is put in low-speed device (eg: printer)
Here buffer is I/O subsystem.
some predefined softwares / drivers are also installed sometimes.

5. Secondary storage management.

main m/m : volatile (power on → works).

80

disc is partitioned into tracks. It is further divided into sector.



To read head is used.

and data is read.

fragmentation
- free space one side
- data on other side

defragmentation

6. Network. LAN, WAN, Internet.

program → program needs some resource that is not with me → then I need to access that resource from a resource system. This is provided in this Network system.

7. Protection System

If two processes are running both should be protected from each other

2 types

i) user process : we write program → execution → process

ii) system process : when we access some program from our we need some system program. and if we access it then it becomes system process.

Scanned by CamScanner

8. Command Interpreter system :-

Command Interpreter :
(eg : in dos dir ↵ will show list of directory
↳ this is a program to show list.
↳ interface b/w^n user and OS

→ Operating System SERVICES. (LQ). Pg-54.
OS provides services to user, system etc.

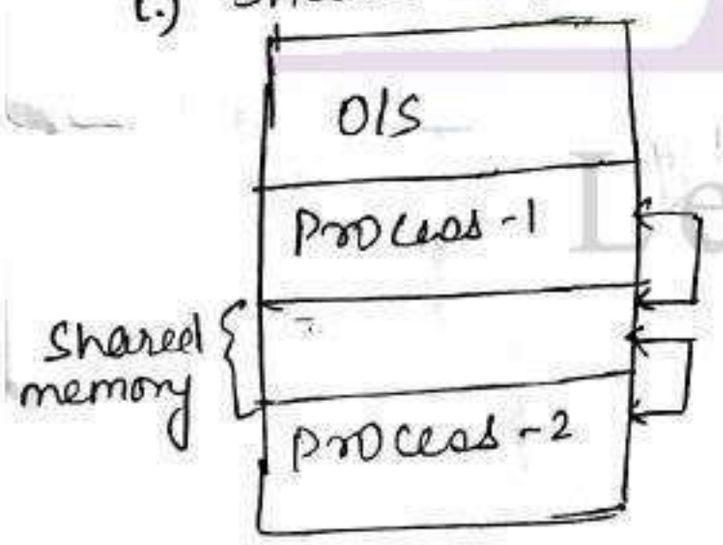
1. Program Execution
program executes normally → O/P
abnormally → error

2. I/O operation.
while executing program we give input using keyboard, mouse, LAN, some file..

3. File System Manipulation.
Reading data from file → write/append etc in file.

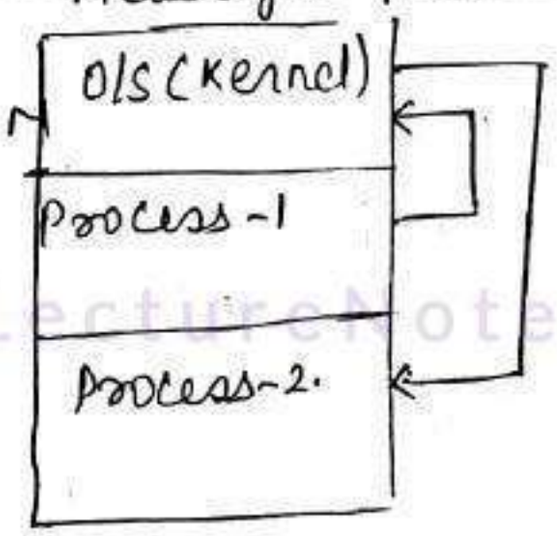
4. Communication
2 processes are running → They may communicate among themselves : i) shared m/m concept ii) message passing
Two process within same system:
Two process in diff. system:

i) Shared m/m



both process can send or receive data

Message passing



P-1 P-2 commu

P-1 ~~comm~~ comm. with O.S then O.S communic-ates with P-2

5.) Error detection
Errors generated by I/O, CPU, memory, program is handled by this.

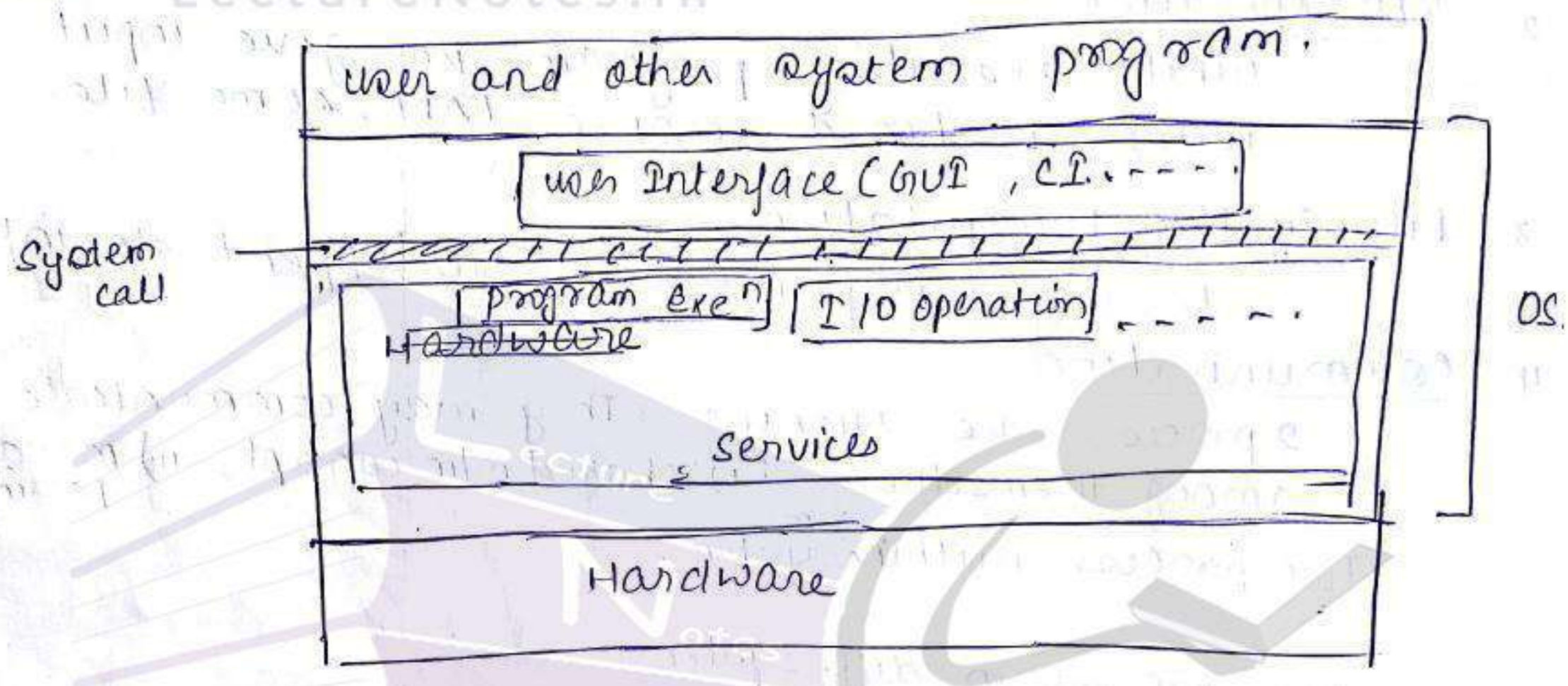
6. Resource Allocation.
Allocating resources to process or a system at time of execution.

Scanned by CamScanner

7.) Accounting :-
TO keep track of how many users use how many resources

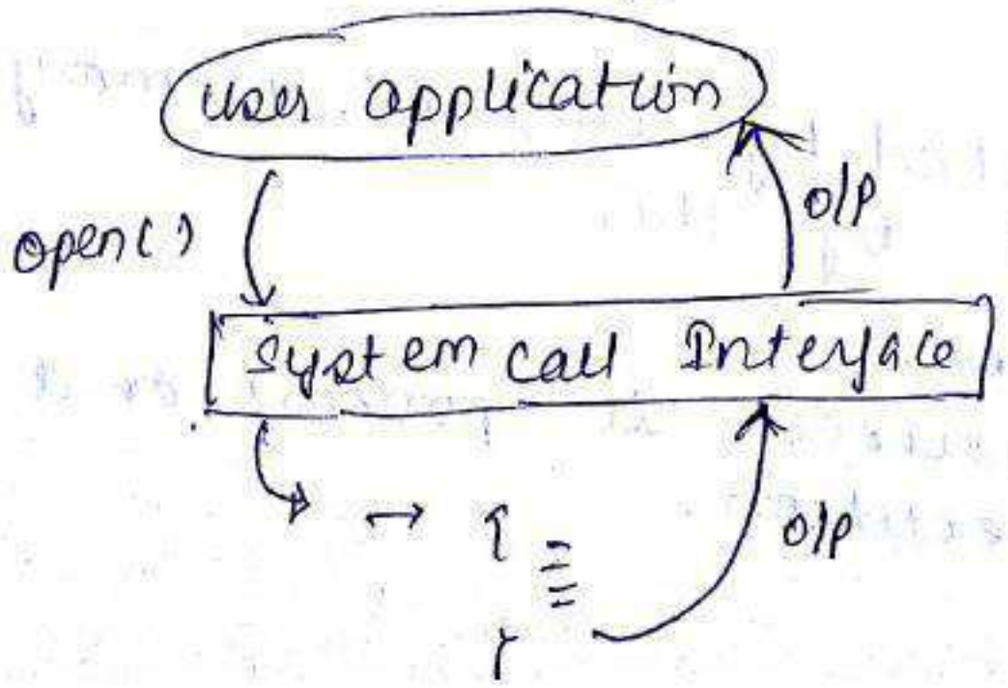
8.) protection and security :-
protection : protecting one process to another
security : whether or not you are authourised person to use system. TO test it userid and password is used.

21/7/17



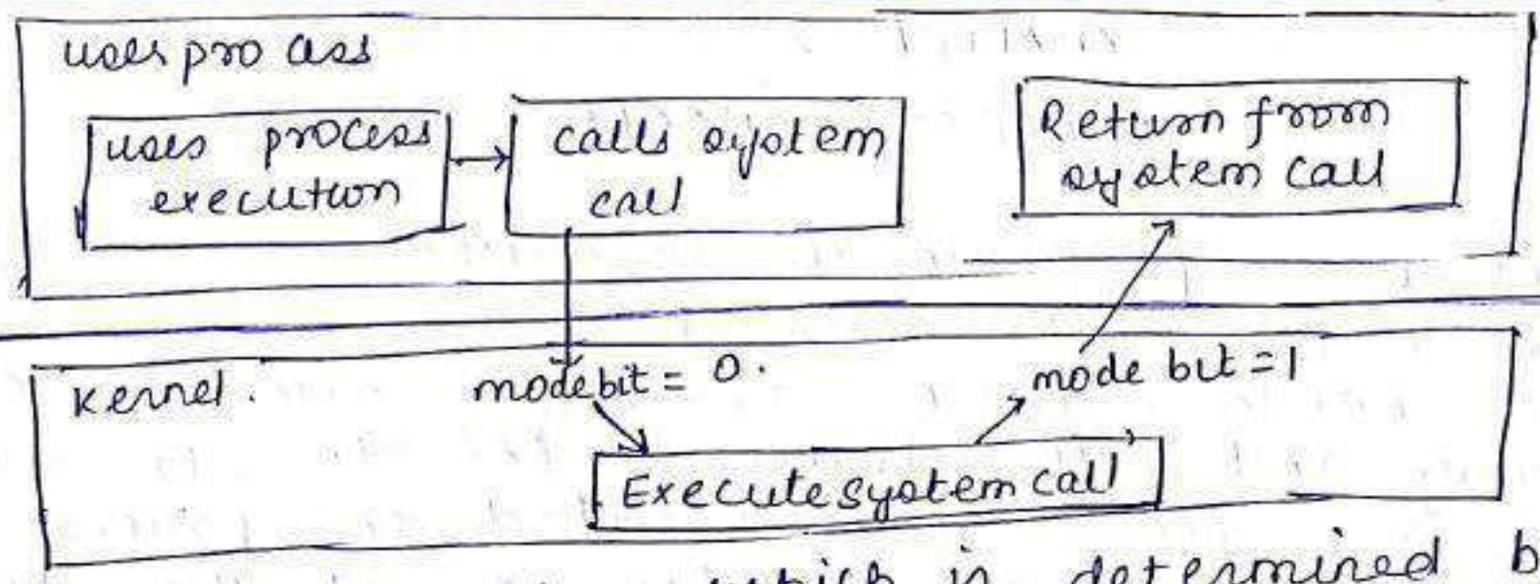
→ System call : provides the interface between the process and the O/S. (Pg-61)
Types of system call : there are 6 types.
i.) process control
ii.) file manipulation.
iii.) Device manipulation
iv.) information maintainance
v.) communications.
vi.) protection. (Pg 65)

eg :-



Scanned by CamScanner

Mode bit



System runs in 2 modes which is determined by bit
if bit is 1 then system is in user mode
0 " " " " kernel " /supervisor mode
mode bit is changed using interrupt and interrupt is executed by system call.
This is how user and kernel process work without hampering each other.

LectureNotes.in

Scanned by CamScanner

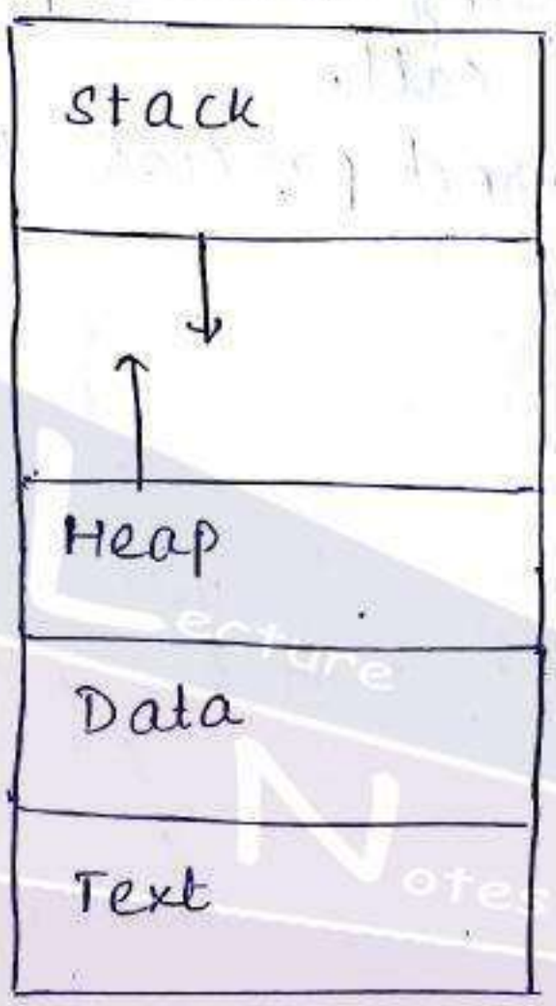
21/7/17.

CHAPTER - 2

PROCESS CONCEPT

- Process is a program in execution.
- Process is a unit of work.
- Process needs certain resources such as CPU time, memory and I/O devices to do its work. These resources are allocated to process either at the time of creation or at the time of execution.
- Process : main m/m Program : secondary m/m.

Internal structure of process :



func^n Arguments
func^n return type.
Local variables.

Dynamic memory allocation.

Global variables

Instructions.

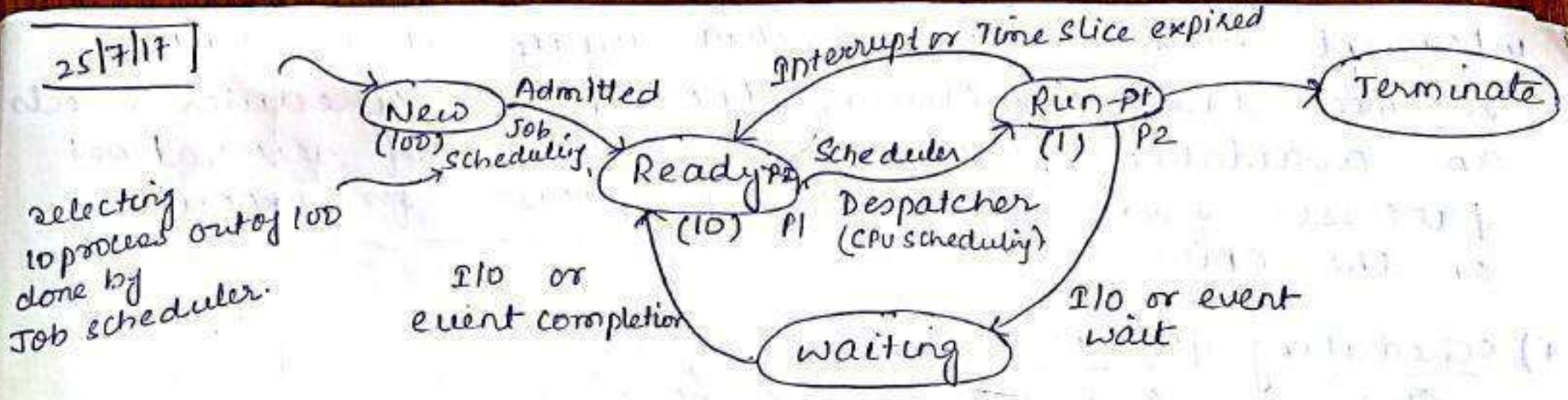
→ Process state

- When a process executes it changes the state
- The state of the process is determined by current activity of the process.
- The states are
  1. New : The process is being created.
  2. Running : The instructions are being executed.
  3. Waiting : The process is waiting for some event to occur.
  4. Ready : The process is waiting to be assigned to a processor.
  5. Terminate : The process has finished its execution.

1 → 4 → 2 → 3 → 4 → 2 → 5.

Scanned by CamScanner

25/7/17



out of 10 process CPU can run a single process at a time } done by CPU & scheduler process: process / CPU scheduling.

[working system of process state]

→ Process control Box Block (PCB).

Each process is represented in the OS by its process control block or Task control block.

Pointer	state
Process Number	
Program counter	
Registers	
memory limits	
list of open files	
list of open devices	
≡	

PCB.

pointer : holds address of each pcb.

state : pcb is in which state is specified here.

process number : every process is identified by its no.

program counter : holds address of next instruction to be executed.

Registers : eg: Accumulator : for holding temp. results ; GPRS R0-R7

, IR : holds current instruction.

m/m limit : base limit ....etc.

eg. condition code / flags : status register. eg numlock

ON — 1
OFF — 0 Num

→ Process Scheduling -

- The objective of multiprogramming is to have some process running at all times to maximize the CPU utilization.
- The objective of time sharing system is to switch the CPU among processes so frequently that user can

Scanned by CamScanner

interrupt with each program while it is running
- To meet these objectives, the process scheduler selects an available process from a set of several available processes from the ready queue for execution on the CPU.

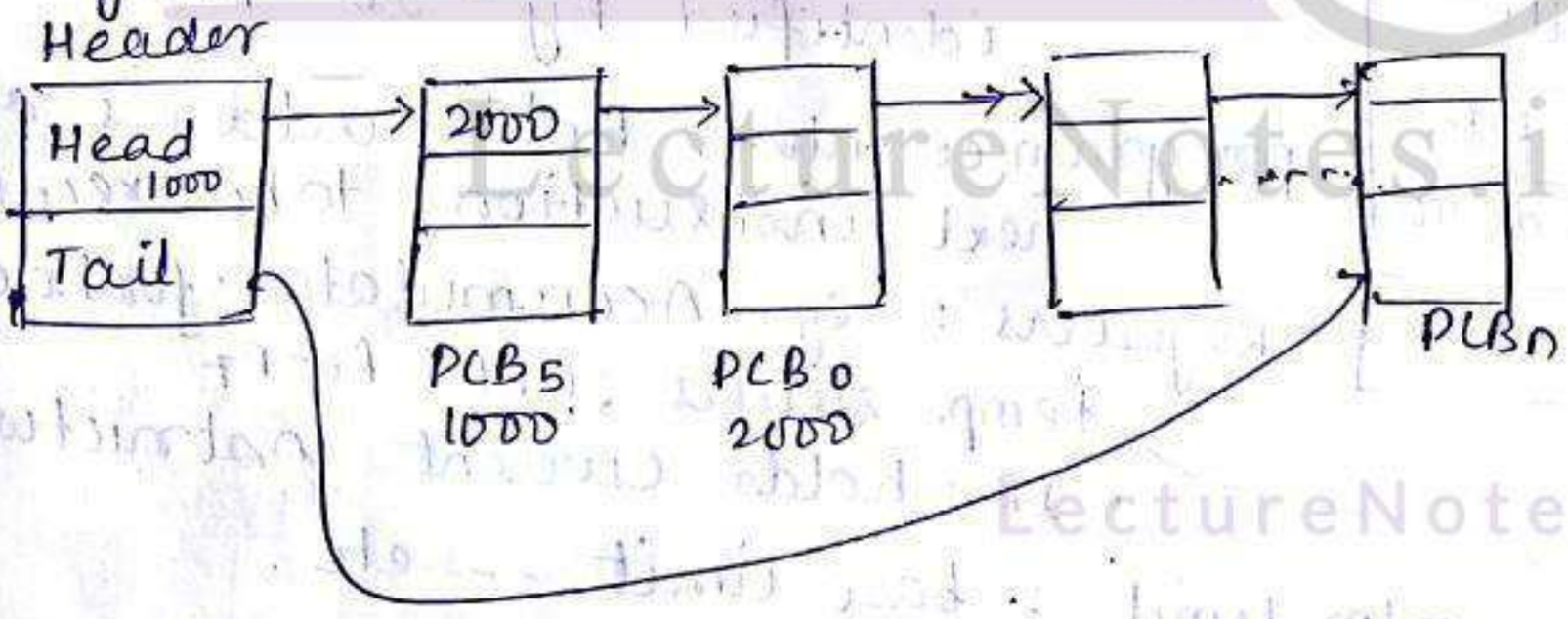
i) <u>Scheduling queues</u>.

- Job queue (newly created process)
- Ready queue
- Device queue (every device maintains a queue)

In process scheduling (Job queue is not used)
└ Ready, Device queue.

- Job <u>queue</u> : when the process enters the system, they are put into a job queue. Job queue consist of all processes in the system.
- <u>Ready queue</u> : The processes that are residing in the main memory and are ready and waiting to execute are kept on a list called Ready queue.
- Device <u>queue</u> : The list of processes waiting for a particular i/o device is called device queue.

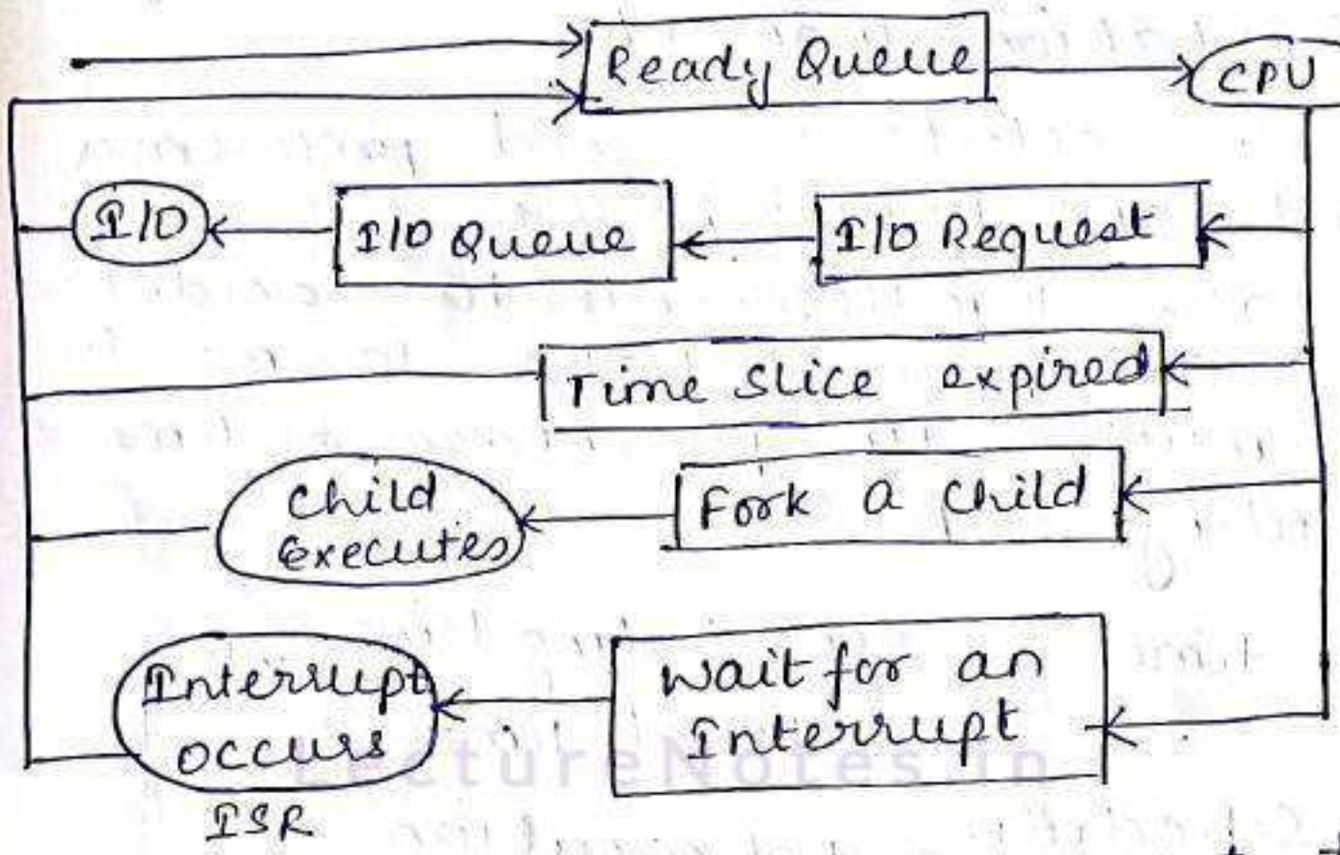
Every Queue is maintained in the form of linked list.



26/7/17

→ <u>Queuing Diagram Representation of process scheduling</u>:

Scanned by CamScanner



- each rectangular box represents the queue.
- Here two queues are used (Ready and Device).
- The circle represents the resources that serve the queues.

→ Scheduler.

- The process migrates among various scheduling queues throughout its lifetime.
- The OS must select a process from these queues in some fashion. This selection process is carried out by a scheduler.
- There are 3 types of schedulers
  i.) long-term scheduler or Job scheduler:
  ii.) short-term scheduler or CPU scheduler
  iii) Medium-term scheduler

i.) Long-term or Job scheduler.
It selects a process from Job pool and loads them into main memory (Ready Queue). for execution.

ii.) short-term
It selects a process from Ready Queue that are ready to execute and allocates the CPU to one of them.

- The number of processes in the memory is called degree of multiprogramming.
- I/O bound process is one that spends more of its time doing I/O then computations

Scanned by CamScanner

- A CPU bound process is one that spends more of its time in computation than I/O.

Note* Long term scheduler selects a good process mix of I/O bound and CPU bound process.

eg: Q: If a scheduler takes 10 milliseconds to decide to execute a process. The process takes 100 ms to execute. How much ~~time~~ percentage of time is taken for scheduling a process.

% of scheduling time = $\frac{\text{scheduling time}}{\text{Total time}} \times 100$

Total time = Scheduling time + Execution time

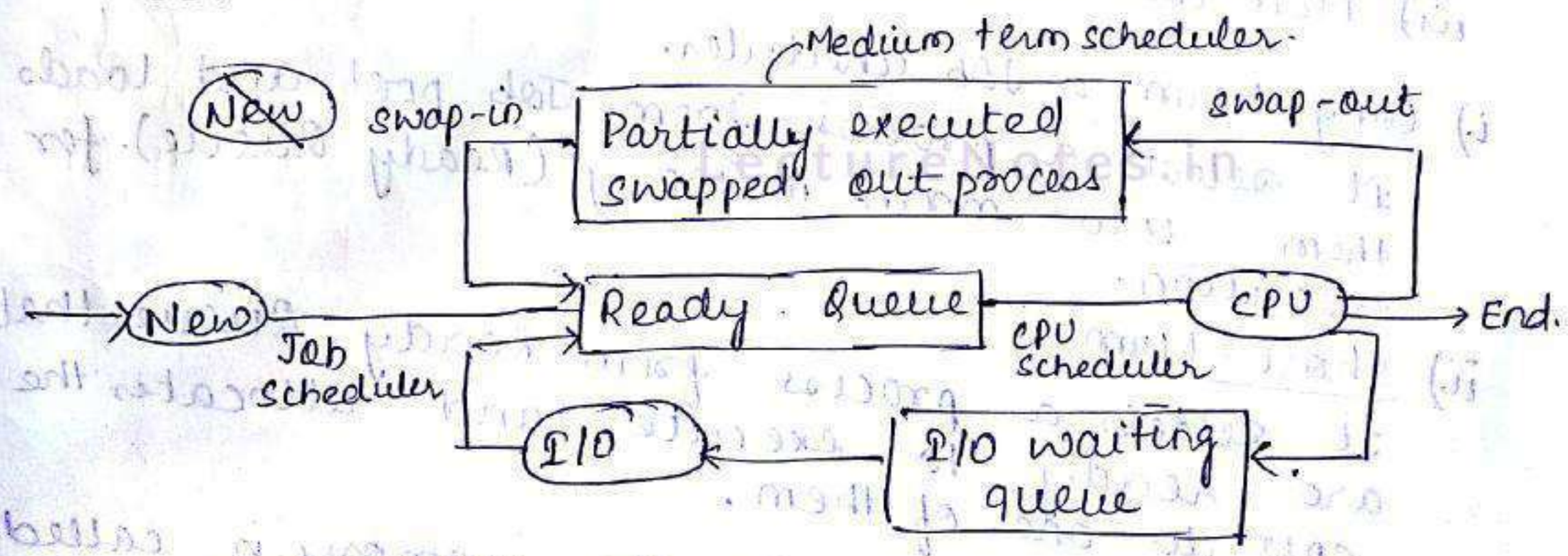
$\therefore$ % of scheduling time = $\frac{10}{110} \times 100$

$\frac{100}{11} = 9\%$

$\therefore$ 9% of total CPU time is wasted for scheduling a process.

iii.) Medium term scheduler:

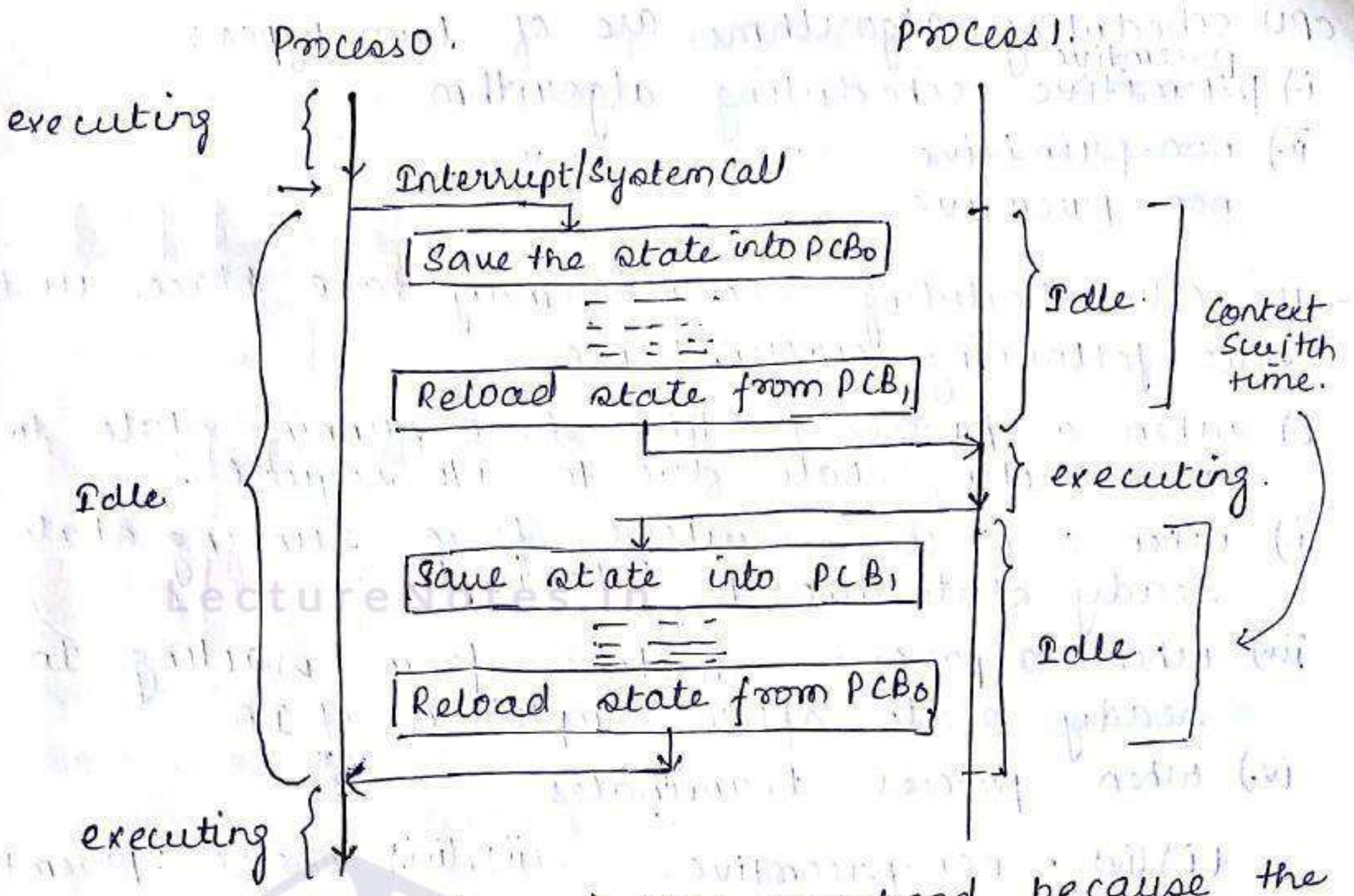
- In time sharing system, we are removing a running process from CPU to the ready queue after time slice expires. So Removing a proces from CPU to ready queue is done by a scheduler called medium term scheduler.



→ Context Switch

Switching the CPU from one process to another requires saving the state of the old process and loading the saved state for the new process is known as context ...

Scanned by CamScanner



Context switch time is pure overhead because the system does no useful work while switching

→ CPU / process scheduling Algorithms

27/7/17

- The CPU scheduling algorithm determines how the CPU will be allocated to the process.
- CPU scheduler : The CPU scheduler selects a process from several available processes in memory (Ready Queue) that are ready to execute and allocates the CPU to one of ~~the~~ them
- Dispatcher : The Dispatcher is a module that gives control of the CPU to the process selected by the CPU scheduler.
  The main function of the scheduler is switching the CPU from one process to another process.

The Dispatcher should be fast because it is invoked during each and every process switch. The time it takes for the dispatcher to stop one process and start another process is known as dispatch latency/delay

Scanned by CamScanner

CPU scheduling algorithms are of two types:
i.) ~~primitive~~ preemptive scheduling algorithm.
ii.) ~~non-primitive~~ " "
non-preemtive

- The CPU scheduling decision may take place under four following circumstances.
i.) when a process switches from running state to the waiting state due to I/O request.
ii.) when a process switches from running state to ready state due to interrupt.
iii.) when a process switches from waiting to ready state after completion of I/O.
iv.) when process terminates.

(i) (iv) : non-preemtive. (ii), (iii.) scheme : preemtive

- In preemtive scheduling the CPU can release the process even in the middle of the execution.
- In non-preemtive scheduling, once the CPU has been alloted to a process, the process keeps the CPU until it releases the CPU either by terminating or by switching to the waiting state

→ Scheduling criteria :

- The criteria determines the best algorithm among several best algorithm.
i.) CPU utilization : It is the percentage of time that the processor is busy.
ii.) Throughput : no. of processes completed per time unit is called throughput.
iii.) Turn around time : The interval from the time of submission of process to the time of completion is called turn around time.

Turn around time = time spent in Job queue
+ time spent in Ready queue
+ execution time of process
+ doing I/O.

Turn around time = finish time - arrival time

Scanned by CamScanner

iv.) waiting time : It is the amount of time that a process spends waiting in the ready queue.
waiting time = starting time - arrival time.

v.) Response time : The time interval from submission of a request until the first response is known as Response time.
Response time = First response - Arrival time.

i.) ii.) should be ↑ iii) iv) v) should be ↓

→ Algorithm

i.) First come first serve

Q. Let a set of processes P1 to P5 arrives in following order Find the schedule.

P.NO.	AT (Arrival time)	BT (Burst time)
1	0	4
2	1	3
3	2	1
4	3	2
5	4	5

: criteria : AT
Mode : non-preemptive

Gantt

P1	P2	P3	P4	P5
0	4	7	8	10	15

FR : First response

P NO	AT	BT	CT (completion time)	TAT (turnaround time) CT-AT	WT = TAT-BT	RT = FR-AT
1	0	4	4	4	0	0
2	1	3	7	6	3	3
3	2	1	8	6	5	5
4	3	2	10	7	5	5
5	4	5	15	11	6	6

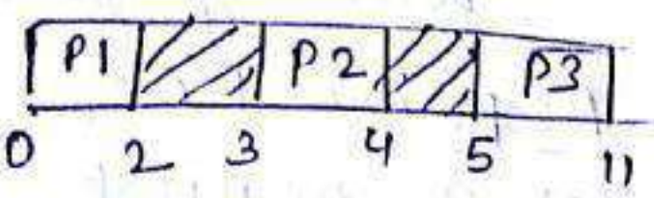
Avg. Turn around time = (4+6+6+7+11)/5 =
Avg WT = (0+3+5+5+6)/5 =
Avg RT = (0+3+5+5+6)/5 =

Scanned by CamScanner

28/7/17

Q

P NO.	AT	BT
1	0	2
2	3	1
3	5	6



PNO	AT	BT	CT	TAT = CT - AT	WT = TAT - BT	RT = FR - AT
1	0	2	2	2	0	0
2	3	1	4	1	0	0
3	5	6	11	6	0	0

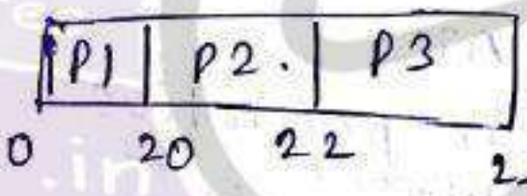
Here WT = 0 means CPU is waiting for process whereas in previous eg. there was waiting time means processes were waiting for CPU.

→ convoy effect : all other process wait for one big process to get off the CPU. This effect results in lower CPU and device utilization

eg { If arrival time is not given all process enter the system at time = 0

Q

PNO.	AT	BT
1	0	20
2	1	2
3	1	1

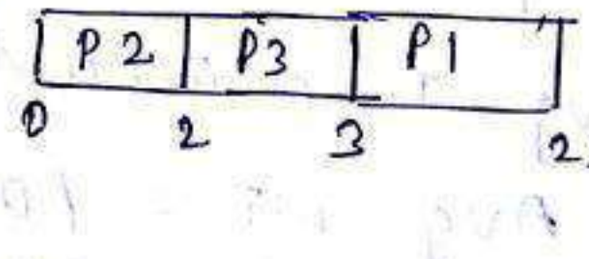


PNO.	AT	BT	CT	TAT = CT - AT	WT = TAT - BT	RT = FR - AT
1	0	20	20	20	0	0
2	1	2	22	21	19	19
3	1	1	23	22	21	21

Avg. TAT = $\frac{63}{3}$ Avg WT = $\frac{40}{3}$

Q

P NO.	AT	BT
1	1	20
2	0	2
3	0	1



PNO.	AT	BT	CT	TAT = CT - AT	WT = TAT - BT	RT = FR - AT
1	1	20	23	22	2	
2	0	2	2	2	0	
3	0	1	3	3	2	

Avg TAT = $\frac{27}{3}$ WT = $\frac{4}{3}$

Scanned by CamScanner

whenever you are executing process you should not execute big process first so as to keep waiting time less.
convoy effect : increase in time - - -.

→ ii) Shortest Job First (SJF)

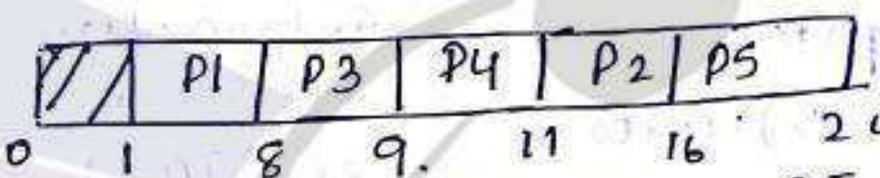
- The process having smallest execution time will be assigned to the processor next.
- If 2 processes having same execution time then you apply first come first serve to break the tie

Here! This algo works according to burst time (less). mode : non-preemtive.

eg:-

P.NO.	AT	BT
1	1	7
2	2	5
3	3	1
4	4	2
5	5	8

use SJF algo to find avg. waiting time, - - - -.



P NO.	AT	BT	CT	TAT = CT-AT	WT = TAT-BT	RT = FR-AT
1	1	7	8	7	0	0
2	2	5	16	14	9	9
3	3	1	9	6	5	5
4	4	2	11	7	5	5
5	5	8	24	19	11	11

Avg. WT = $\frac{30}{5}$ = 6

Avg. RT = 10.6

Avg. TAT = $\frac{53}{5}$ = 10.6

- shortest Job first with prediction of burst/execution time.

Advantage of SJF.
- maximum Throughput
- min. avg. waiting time and turn around time

Disadvantage of SJF
- It leads to starvation (long process waits due to many small ones)
- It is not implementable because burst time of the process cannot be known in advance in real situations

The solution is SJF with prediction / predicted burst time.

Scanned by CamScanner

Prediction Technique

- Static
  - process size
  - process type
- Dynamic
  - simple average
  - exponential average

1/8/17

→ Process size (Byte)

P_{old} = 200 KB ⇒ 20 units
P_{new} = 201 ⇒ 20 units
} × not advisable to use.

→ Process type

- Kernel process
  eg: scheduler dispatcher
  (3-5) units
- user process
  - Interactive process
    ex: games
    (5-8)
  - foreground process
    (10-15) units of time
  - Background process
    (15-20) units

Dynamic

- simple average

let n processes starting from $(P_1 - \cdots P_n)$
let t_i be the etc: attual burst time
τ_i denotes predicted burst time of i process.

so. $\boxed{\tau_{i+1} = \frac{1}{n}\sum_{i=1}^{n} t_i}$

- exponential average.

$\tau_{n+1} = \alpha t_n + (1-\alpha)\tau_n$ —— ①, $0 \le \alpha \le 1$

predicted time of (n+1)th process

α : const : smoothing factor.

$\tau_n = \alpha t_{n-1} + (1-\alpha)\tau_{n-1}$ —— ②

put ② in ①

Scanned by CamScanner

$\tau_{n+1} = \alpha t_n + (1-\alpha)[\alpha t_{n-1} + (1-\alpha)\tau_{n-1}]$

$\tau_{n+1} = \alpha t_n + (1-\alpha)\alpha t_{n-1} + (1-\alpha)^2 \tau_{n-1}$

$\tau_{n+1} = \alpha t_n + (1-\alpha)\alpha t_{n-1} + \cdots\cdots + (1-\alpha)^j \alpha t_{n-j} + \cdots\cdots + (1-\alpha)^{n+1} \tau_n$

ex $\alpha = 0.5 \quad \tau_1 = 10$ Actual BT (P1, P2, P3, P4) = (4, 8, 6, 7)

$\tau_5 = ?$

$\tau_2 = \alpha t_1 + (1-\alpha)\tau_1$

$= 0.5 \times 4 + 0.5 * 10.$

$= 7.$

$\tau_3 = \alpha t_2 + (1-\alpha)\tau_2$

$= 0.5 \times 8 + 0.5 \times 7 = 7.5$

$\tau_4 = \alpha t_3 + (1-\alpha)\tau_3$

$= 0.5 \times 6 + 0.5 \times 7.5$

$= 3 + 3.75$

$= 6.75$

$\tau_5 = 6.875$

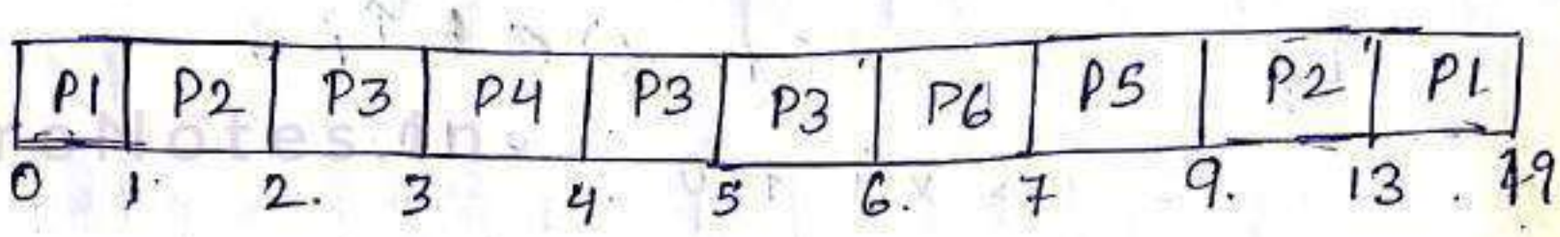
→ shortest remaining time first scheduling time algorithm

- In this algorithm CPU scheduler always selects the process that has the shortest remaining processing time.
- when a new process enters the ready queue, the short term scheduler (CPU scheduler) compares remaining time of executing process with the newly entered process
- If the new process has least executing time, the scheduler selects that process for execution. Otherwise continues with the old process execution.

Scanned by CamScanner

P.NO.	Arrival time AT	Burst/execution time BT	completion time CT	~~TAT~~ TAT = CT-AT	WT = TAT - BT
1	0	~~7~~ 6 0	19	19	12.
2	1	~~5~~ 4 0	13	12.	7
3	2.	~~3~~ 2 ~~1~~ 0	6.	4	1
4	3	1 0	4	1	0.
5	4	2	9	5	3.
6	5	1 0.	7	2	1

schedule



when we apply fcfs

criteria : shortest remaining time

mode : preemtive

for completion time move right to left.

→ Priority scheduling algorithm

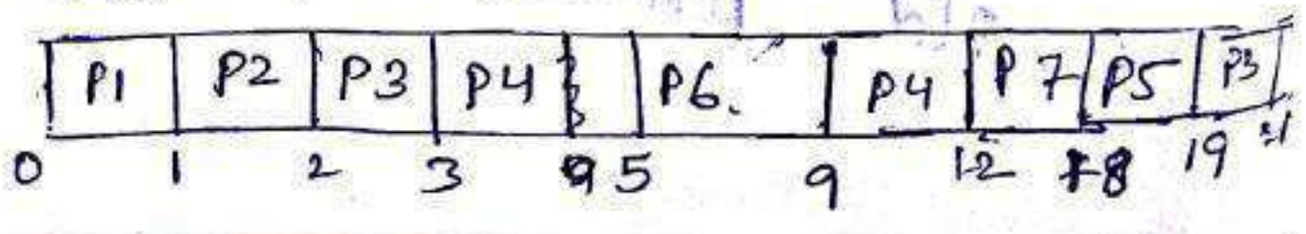
- A priority is associated with each process and the CPU is allocated to the process with the highest priority
- equal priority processes are scheduled in fcfs order (first come first serve).
- priority can be defined internally or externally.

nothing given: smallest no. — more priority

2/8/17

Q. Preemtive priority scheduling algorithm.

P.NO.	Priority	AT	BT	CT	TAT = CT - AT	WT = TAT - BT	RT = FR - AT
1	2 (low)	0	~~4~~ 3	25	25	21	0
2	4	1	~~2~~ 1	22	21	19	0
3	6	2	~~3~~ ~~2~~	21	19	16.	5
4	10	3	~~5~~ ~~4~~ 3	12	9	4	0
5	8	4	~~1~~	19	15	14	14
6	12 (high)	5	~~4~~	9	4	0.	0
7	9	6	~~6~~	18	12	6.	6



Scanned by CamScanner

Avg. WT = ? Avg. TAT = ? Avg. RT = ?

Throughput = Total job / Total time = 7/25

→ Non-preemtive priority scheduling :=

WT = RT in case of non-preemtive

Q.

P.NO.	Priority	AT	BT	CT	TAT	WT	RT
1	2 (low)	0	4	4	4	0	0
2	4	1	2	25	24	22	22
3	6	2	3	23	21	18	18
4	10	3	5	9	6	1	1
5	8	4	1	20	16	15	15
6	12 (high)	5	4	13	8	4	4
7	9	6	6	19	13	7	7

Avg. WT = ?
" TAT = ?
" RT = ?
Throughput = ?

Schedule

Note* In non-preemtive scheduling waitting time is equal to response time

- The major problem with priority scheduling algo is indefinite blocking or starvation.
- Starvation means only higher priority processes are executing whereas lower priority processes are waiting for the CPU for long period of pe time.
- A solution to this problem : aging
  aging is a technique to gradually increase the priority of the processes that are waiting in the syster for longer period of time.

ROUND ROBIN ALGORITHM. (RR)

- The RR algorithm is designed for time-sharing systems. In this case
- It is similar to fcfs algo but the preemption is added to switch blw the processes.
- A small time unit called time quantum or time slice is defined by OS.
- In this algorithm the CPU switches b/w the processes when the time quantum expires

P2 | P1
22 25

Scanned by CamScanner

Eg:-

P.NO.	AT	BT	CT	TAT	WT	RT
1	0	~~4~~ 2.	8	8	4.	
2	1	~~5~~ ~~3~~ 1	18	17	12	
3	2	~~2~~	6.	4	2	
4	3	1	9	6.	5	
5	4	~~6~~ ~~4~~ 2.	21	17	11	
6.	6	~~3~~ ~~1~~.	19	13.	10.	

TQ = 2.

preemtive

Queue: | P1 | P2 | P3 | P1 | P4 | P5 | P2 | P6 | P5 | P2 | P6 | P5 |

Schedule:

P1	P2	P3	P1	P4	P5	P2	P6	P5	P2	P6	P5.

0 2 4 6. 8 9 11 13 15 17 18 19 21

3 | 8 | 17

Q =

P.NO.	AT	BT	CT	TAT = CT - AT	WT = TAT - BT	RT = FR - AT
1	5	~~5~~ 2	32	27	22.	10.
2	4	~~6~~ ~~3~~.	27	23.	17	5.
3	3	~~7~~ ~~4~~ 1	33	30.	23.	3.
4	1	~~9~~ ~~6~~ ~~3~~.	30	29	20.	0.
5	2	~~2~~.	6.	4	2.	2.
6.	6.	~~3~~	21	15	12	12.

TQ = 3.

Queue: | P4 | P5 | P3 | P2 | P4 | P1 | P6 | P3 | P2 | P4 | P1 | P3 |

Schedule:

	P4	P5	P3	P2	P4	P1	P6	P3	P2	P4	P1	P3

0 1 4 6. 9 12 15 18 21 24 27 30 32 33

time quantum increase → fcfs

time quantum decrease → in ms → then process just jumps from one to another.

- The performance of round robin algorithm depends on the size of time quantum
- If the time quantum is very large, the RR algorithm behaves same as fcfs algorithm.
- If time quantum is very small (1ms), the RR algorithm is called processor sharing.

Process time = 10.

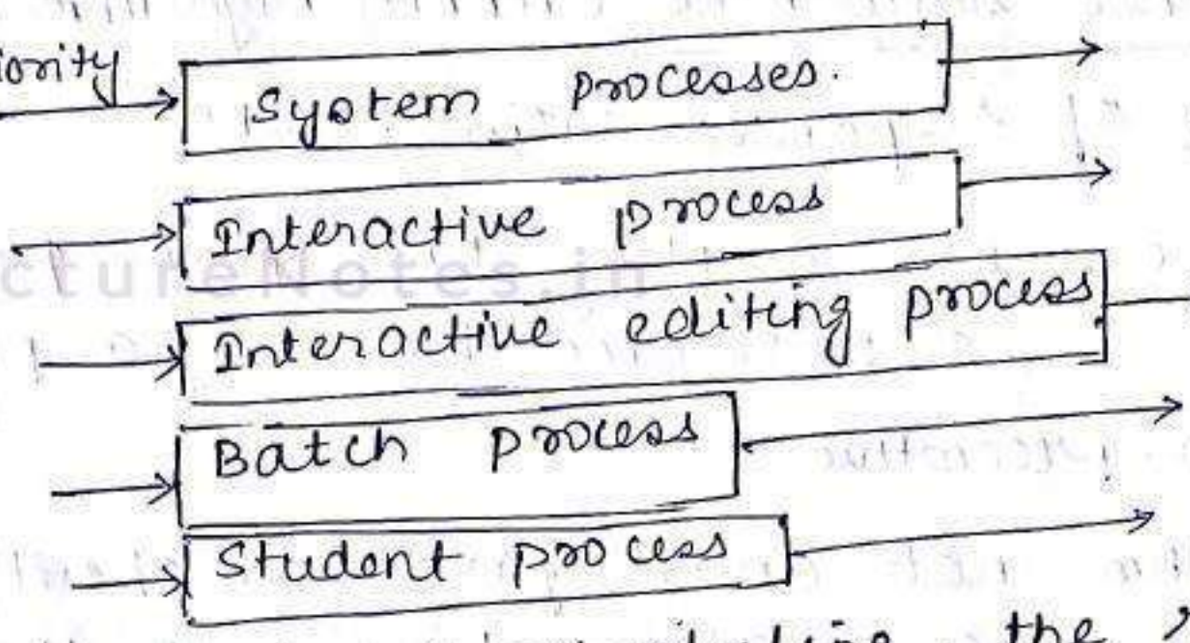
Time Quantum	context switch.
12	0
6	1
1	9.

Scanned by CamScanner

- smaller the time quantum, context switch will be more.

Multilevel Queue scheduling Algorithm :

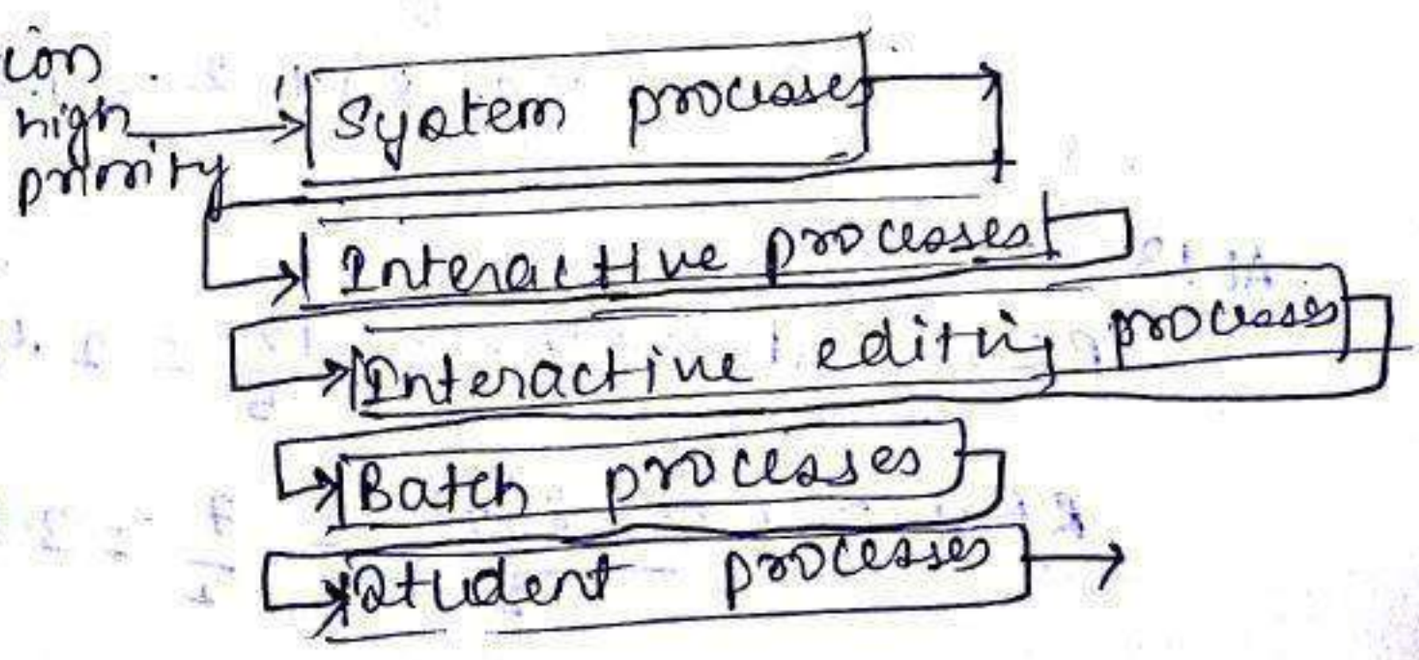
- Processes are classified into different groups such as - foreground - background -.. etc.
- These processes have different response time so they need different scheduling schemes.



- In multilevel queue scheduling, the ready queue is divided into several separate queues.
- The processes are permanently assigned to 1 queue based on the priority of the process.
- Each queue has its own scheduling algorithms.

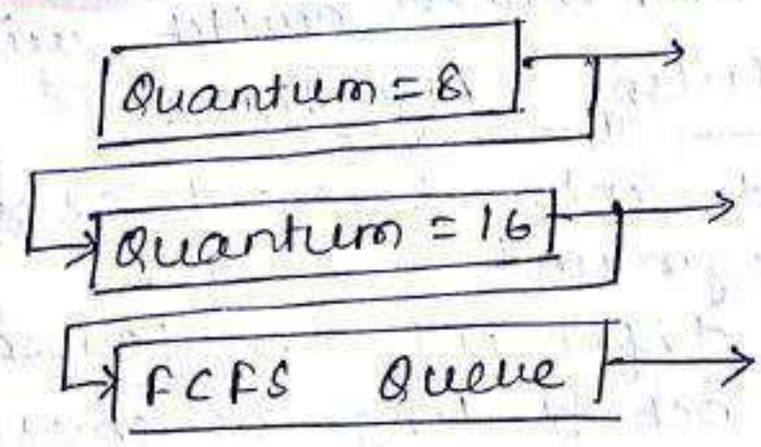
Multilevel feedback queue-scheduling algorithm :

- In multilevel queue scheduling, the processes are permanently assigned to a queue on entry to the system.
- Processes do not move between the Queue (multilevel queue).
- In multilevel feedback Queue scheduling allows a process to move between queue.
- If a process uses too much CPU time, it will move to the lower priority queue.
- It prevents starvation.



Scanned by CamScanner

eg:-



Highest Response ratio next (HRRN) Algorithm :

- The criteria of response ratio = $\frac{W+S}{S}$.

where, W = waiting time for a process so far
S = service time of a process or BT

- Mode : Non-preemtive
- HRRN algorithm not only favours shortest job but also limits the waiting time of the longer jobs.

4/8/17

P.NO.	AT	BT	CT	TAT = CT - AT	WT = TAT - BT	RT = FR - AT
0	0	3	3	3	0	0
1	2	6	9	7	1	1
2	4	4	13	9	5	5
3	6	5	15	9	4	9
4	8	2	20	12	10	5

schedule

P0	P1	P2	P4	P3
0	3	9	13	15	20

At 9

$RR_2 = \frac{(9-4)+4}{4} = 2.25$

$RR_3 = \frac{(9-6)+5}{5} = 1.6$.

$RR_4 = \frac{(9-8)+2}{2} = 1.5$

once whose response is more will go for execution. i.e P_2.

At 13

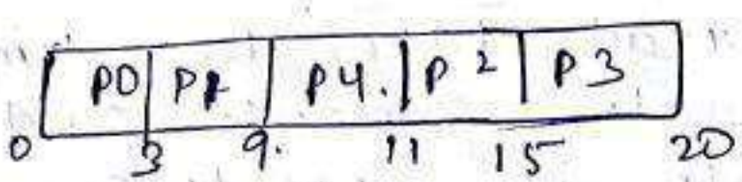
$RR_3 = \frac{(13-6)+5}{5} = \frac{12}{5} = 2.4$

$RR_4 = \frac{(13-8)+2}{2} = \frac{7}{2} = 3.5$

P_4 will be executed.

Scanned by CamScanner

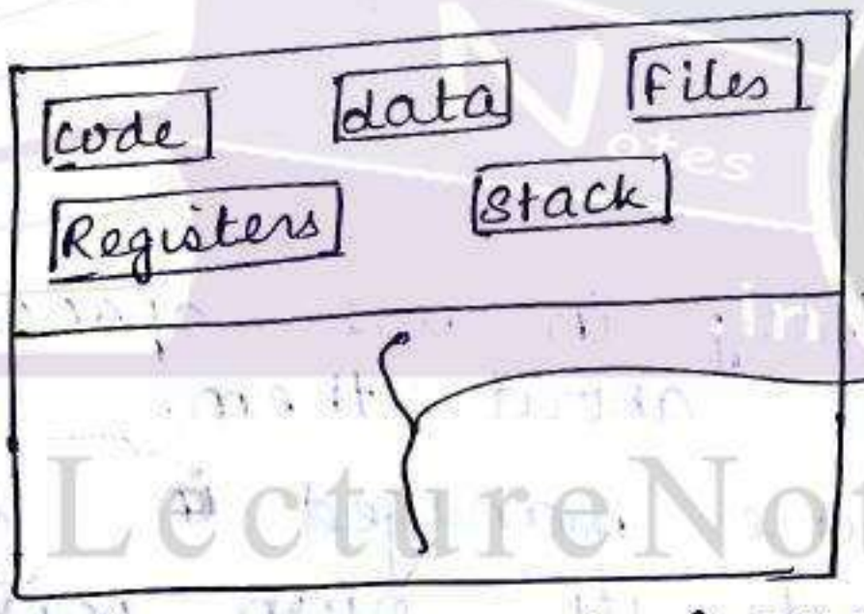
SJF schedule



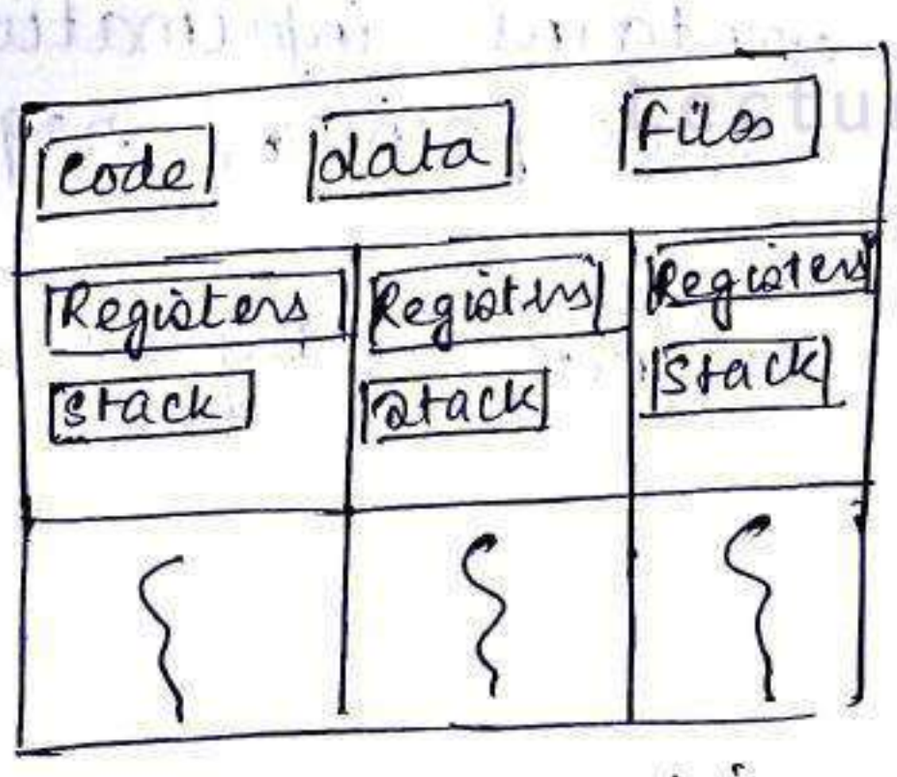
PNO.	AT	BT
0	0	3
1	2	6
2	4	4
3	6	5
4	8	2

→ THREAD.

- Process is divided into number of smaller task called Thread.
- A Thread sometimes called light weight process (LWP).
- It is a basic unit of CPU utilization.
- Thread has Thread id, program counter (PC), Register set and stack etc....
- NO. of threads within a process execute at a time called multithreading.



Single Thread Process



Multithreading process.

LectureNotes.in

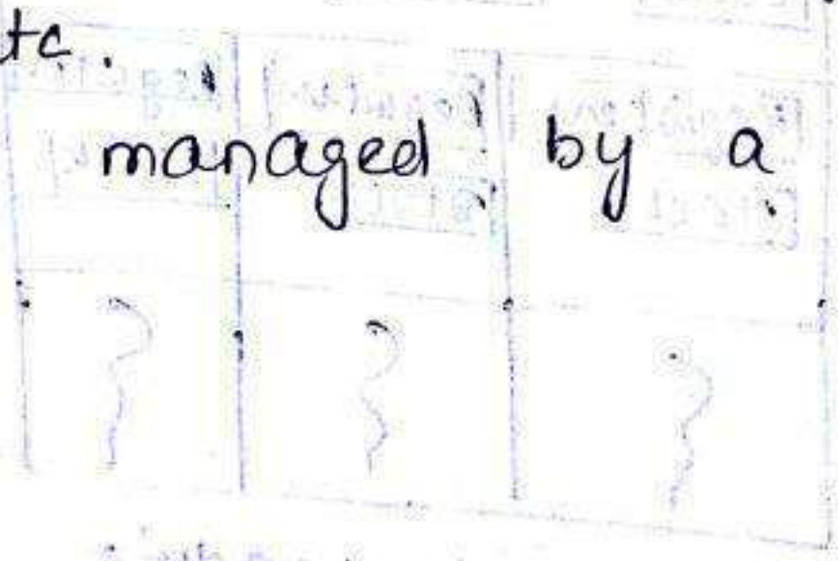
Scanned by CamScanner

process	Thread
i) Process takes more time to create, execute and complete its operation	i) Thread takes less time to create, execute and complete its operation.
ii) It takes more time to switch b/wn processes	ii) It takes less time to switch b/wn threads.
iii) Process communication is more difficult	iii) Thread communication is easier.
iv) Processes are loosely coupled	iv) Threads are tightly coupled
v) It requires more no. of resources to execute	v) It requires fewer no. of resources to execute i.e it is light weight
vi) Processes are not suitable for parallel activities	vi) Threads are suitable for parallel activities.

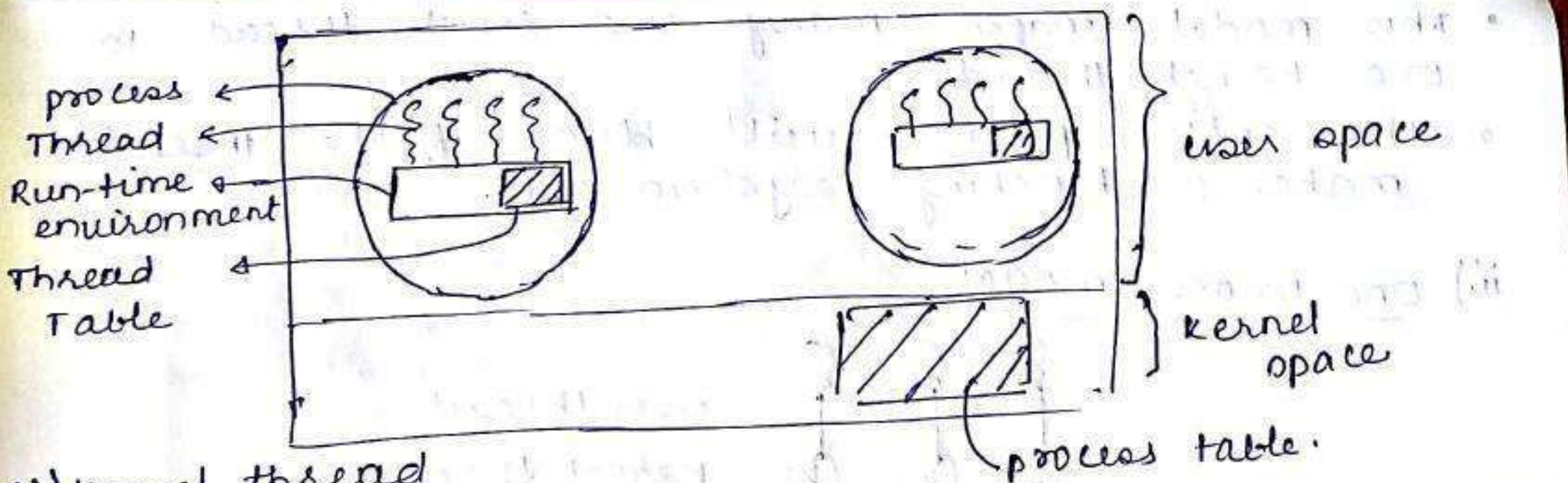
- There are 2 types of thread.
  - i) user thread
  - ii) kernel thread

→ i) user thread.

- are loaded entirely in user space, kernel knows / nothing about them.
- when threads are managed in user space the process needs its own private thread table.
- The thread table contains information about program counter, stack pointer, registers and state (of thread) etc.
- Thread table is managed by a run-time environment

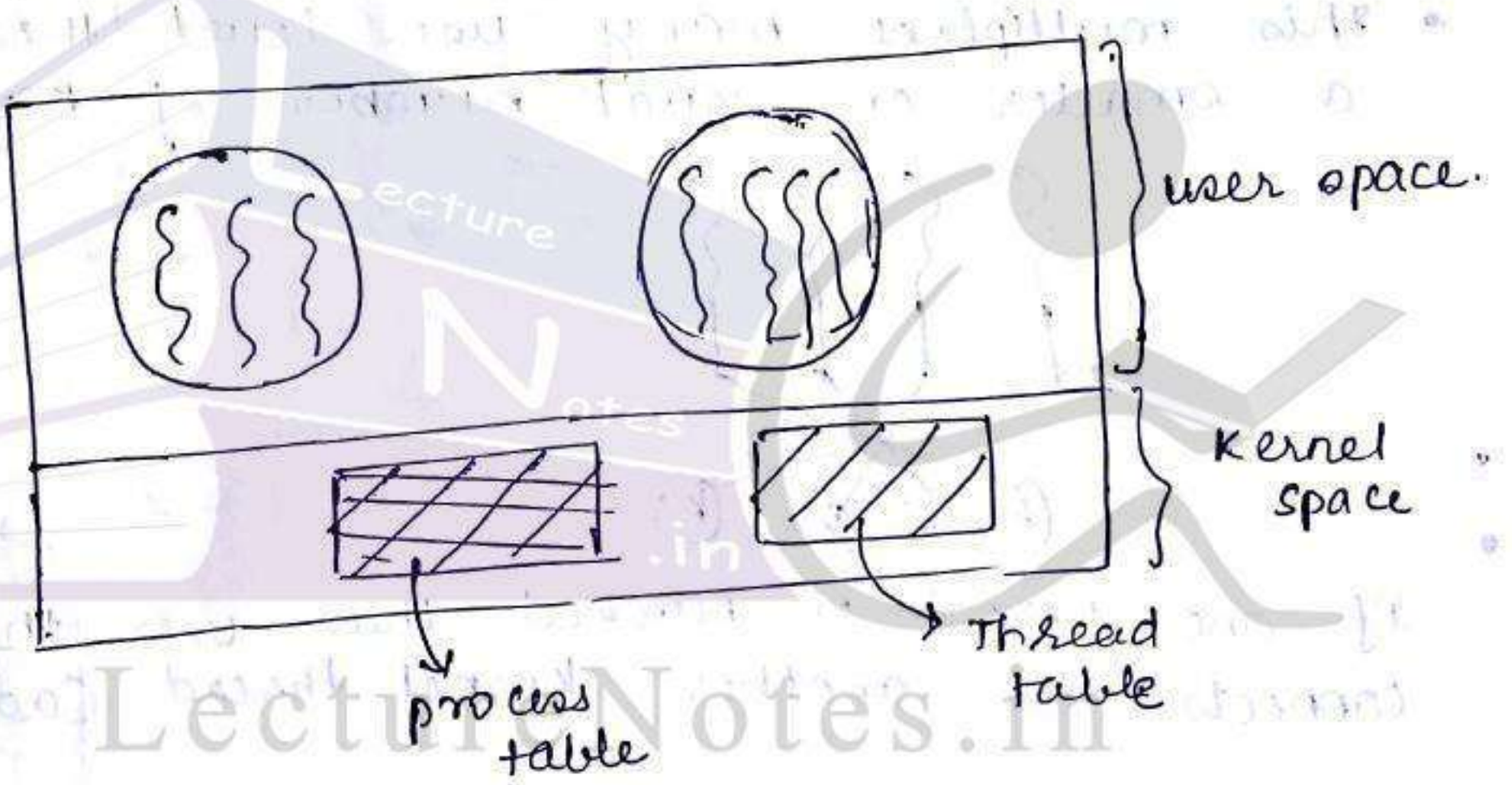


Scanned by CamScanner



ii) Kernel thread

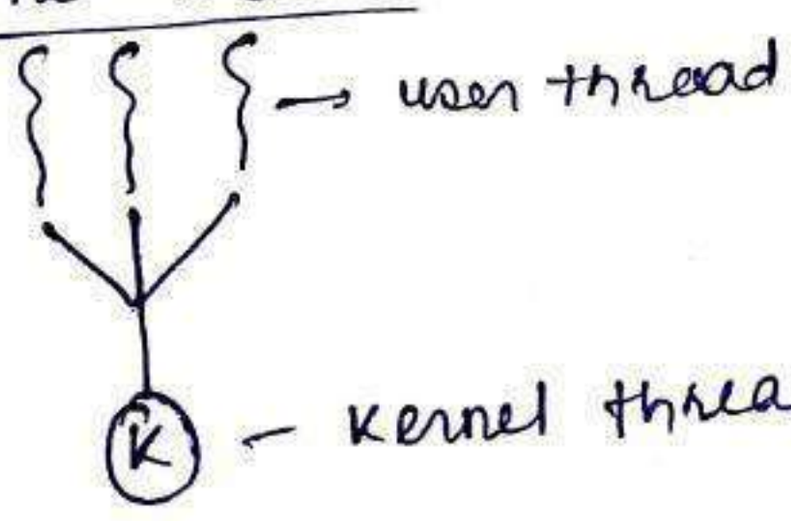
- In kernel thread, kernel does total work of thread management
- There is no thread table in each process
- kernel has a thread table that keeps track of all the threads in the system.



→ Multi-threading model

- many system provide support for both user and kernel thread.
- Three types of multithreading implementation

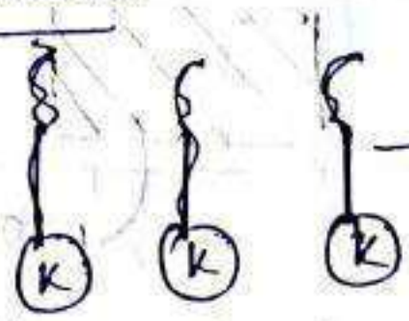
i) many to one model



Scanned by CamScanner

- This model maps many user level thread to one kernel thread.
- The entire process will block if the thread makes a blocking system call.

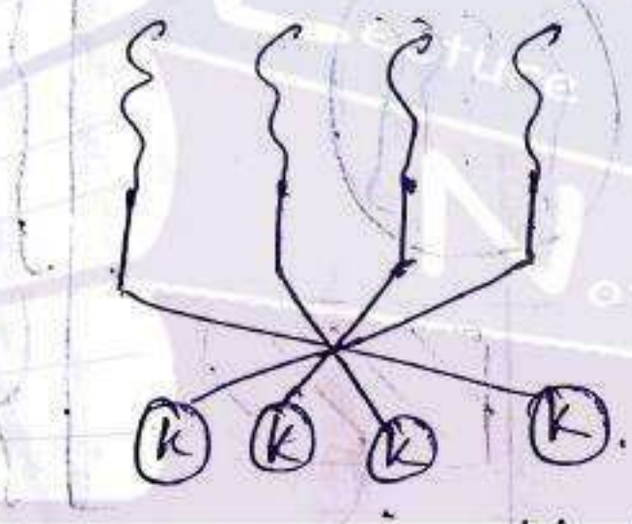
ii.) One to one model



- In this maps each user thread to a kernel thread
- one blocked yet others work. (advantage)

iii.) many to many model

- This multiplexes many user level threads to a smaller or equal number of kernel threads



- If one kernel is blocked then user thread connects to another kernel thread. (advantage)

LectureNotes.in

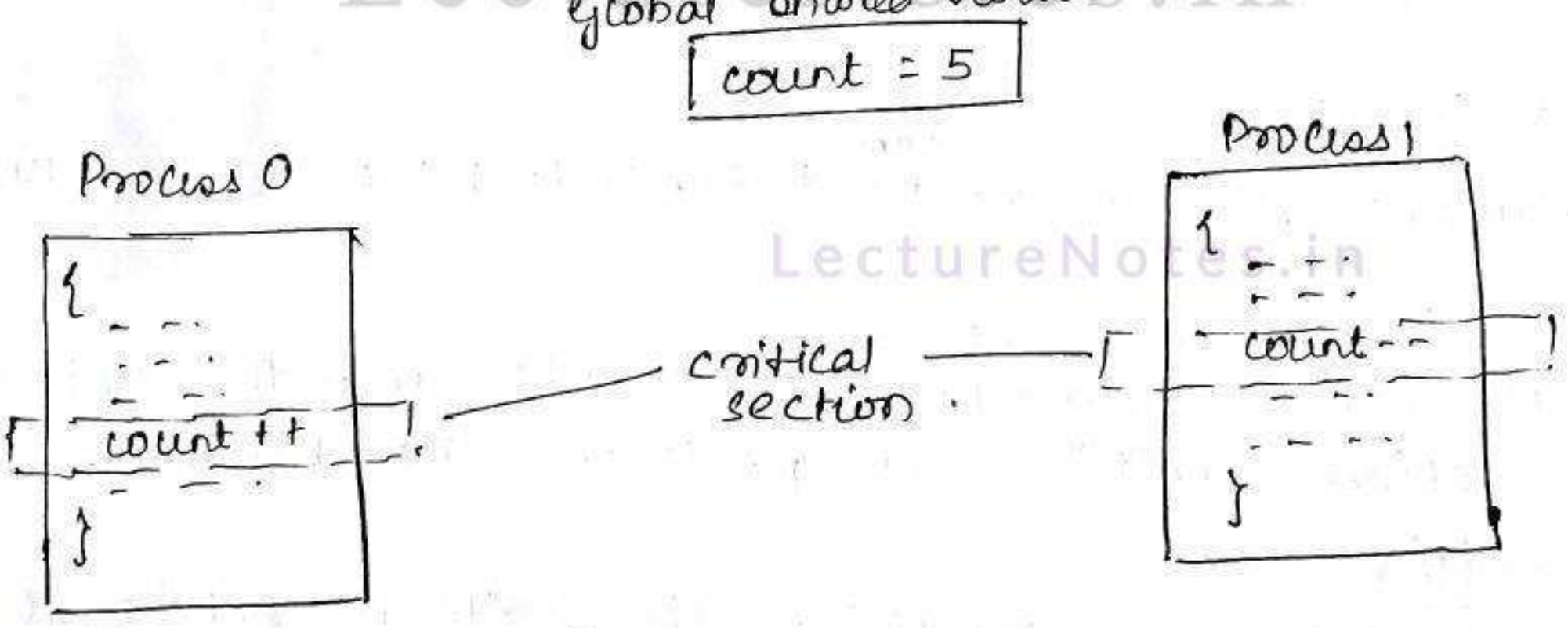
Scanned by CamScanner

8/8/17

#: PROCESS SYNCHRONIZATION

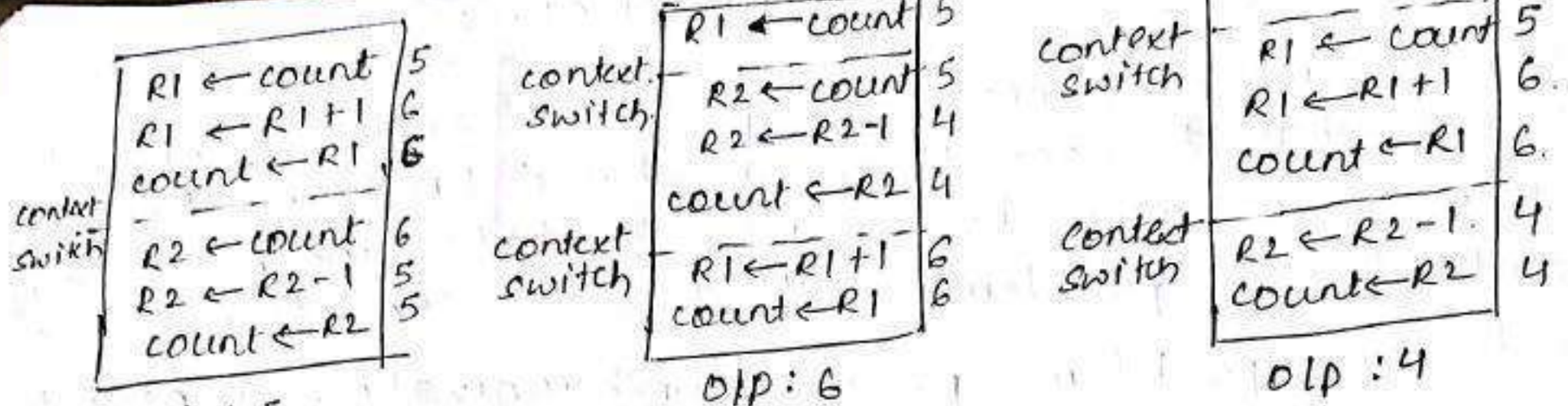
- When two or more co-operating processes execute a shared variable, a shared record or a shared file simultaneously, then there may be chances of data inconsistency. This leads to a problem called race-condition.
- To avoid this problem process synchronization is needed.
- There are two kinds to synchronization.
  - i) Data-access synchronization : It ensures that shared data does not get loose consistency, when they are updated by several co-operating processes.
  - ii) control-synchronization : It ensures that co-operating processes perform their actions in desired order.

- Race-condition
  - A situation where several processes access and manipulate the same data concurrently and the outcome of the execution depends on the particular order in which the access takes place.
  - A race condition occurs when multiple processes read or write data items so that the final result depends on the order of the executions of instructions in the multiple processes.



expected o/p: 5 ✓
if they execute simultaneously 4, 6. X.

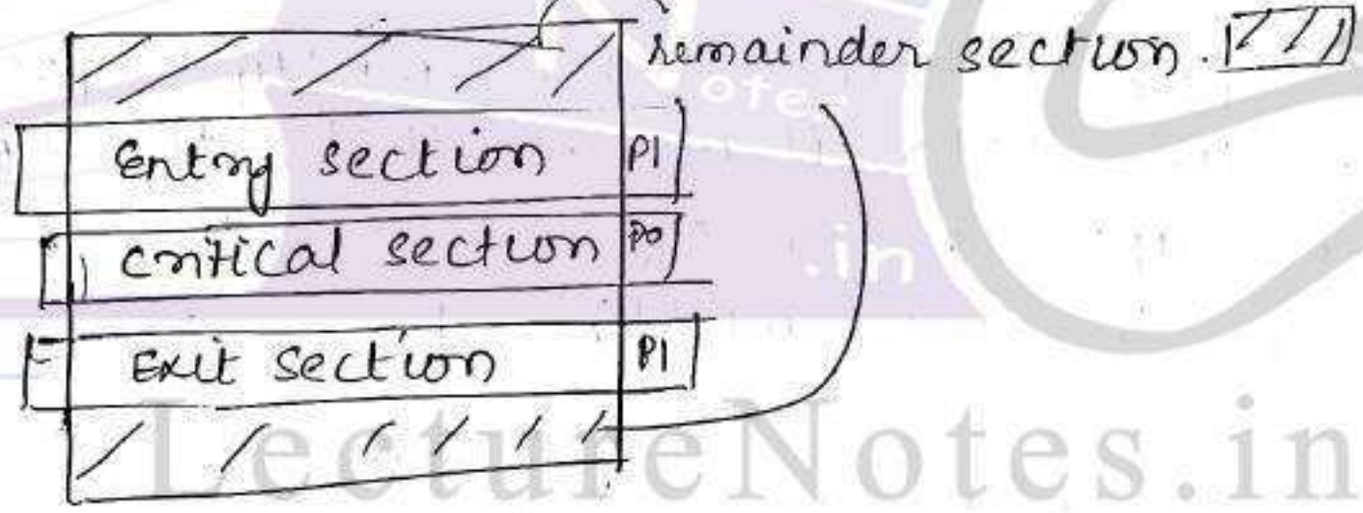
Scanned by CamScanner



Race condition

Due to two co-operating process which have a shared value

- Race condition can be prevented by synchronization.
- Synchronization ensures that only one process at a time manipulate the critical data.
- ~~The outcome depends on the~~
- A situation where the several processes access and manipulate the critical data is called critical section.
- The outcome of the critical section depends on the order in which the access takes place.



- critical section:
  - only one process ~~should~~ can execute here at a time
- entry section
  - It allows only one process to enter into the critical section and rest processes are blocked.
- exit-section.
  - Once a process completes its critical section, it enters into the exit section.
  - In exit section, it allows another process to enter into the critical section.

LectureNotes.in

Scanned by CamScanner

- remainder section:
  - The rest of the codes are called remainder sectio
- A solution to the critical section problem must satisfy
  i.) mutual exclusion
  ii.) progress
  iii.) bounded-waiting

i.) mutual exclusion.
- It allows only one process to execute in critical section at a given instant of time.

ii.) progress
a.) when no process is in critical section, any proces that requests entry into the critical section must be permitted within finite amount of time.
b.) A decision who will enter into the critical section must be taken by considering only those processes who wants to enter into the critical section and these decision must be taken in finite time.

iii.) bounded-waiting
- there is an upper bound on the number of times a process enters the critical section, while another process is waiting.

9/8/17

If the above sections are satisfied then it is said to be synchronized. (process)

→ LOCKS and UNLOCKS

- All critical section problems use locking and unlocking. Suppose I have a shared variable.

shared variable
```
int count = 5;
lock_t L;
```

Process 0
```
{
=
lock(L);      entry-section
c++;          C.S.
unlock(L)     exit section
```

Process 1
```
{
=
lock(L);
count--;
unlock(L);
=
```

Scanned by CamScanner

- LOCK(L) : aquire LOCK(L) exclusively only one process with L can access the critical section
- unlock(L): release exclusive access to LOCK(L) permitting other processes to access the critical section.

Note*: locking and unlocking should be designed in such a way that the three requirements of critical section should be satisfied.

→ using interrupt

LectureNotes.in

shared variable

```
int count=5;
```

Process 0

```
{
  ≡
  disable interrupt (); entry
  count++;              → critical
  enable interrupt      → exit
  ≡
}
```

Process 1

```
{
  ≡
  disable interrupt(); E.S
  count--;             E.S
  enable interrupt;    E.S
  ≡
}
```

- i.) It is simple : when interrupts are disabled, context switch will not happen
- ii.) It requires privelege: user processes generally cannot disable the interrupts.

→ software solutions {for this critical section}

attempt 1.

shared variable

```
int ~~item~~ turn = 1;
```

Process 1

```
while (1)
{ ≡
  while(turn=2);
  Critical section
  turn=2;
} ≡
```

Process 2.

```
while(1)
{
  ≡
  while(turn==1);
  Critical section.
  turn = 1
  ≡
}
```

Scanned by CamScanner

{ while(cond^n); ≡ statements } { if cond^n true then nothing executes; if " false then statements execute }

P1 → P2 → P1 → P2 - - - mutual exclusion ↑ as at a time only one process executes at a time.

P2 can't proceed till P2 executes so progress is not satisfied. satisfied. (if P2 doesn't want to execute)

- It achieves mutual exclusion
- Here processes are executed alternatively in critical section P1 → P2 → P1 → P2 - - -
- Busy waiting : waste of power and time (frequently P1 checks value of turn and while statement - - -)

Problems for attempt 1

- It had a common turn flag that was modified by both the processes.
- This required processes to alternate
- Due to this reason progress also not satisfied.

Solution:

- It should have two flags : one for each process

→ Software solution (attempt 2)

10|8|17

- The problem with attempt 1 is that it does not retain sufficient information about the state of the process. It remembers only which process is allowed to enter into the critical section.
- To avoid this problem we can replace 'turn' variable with the following array.

```
Boolean flag[2];
flag[0] = flag[1] = false
```

Scanned by CamScanner

shared variable.

flag [0] = flag [1] = false.

Process 1

```
while (1)
{
    while (flag [1]);
    flag [0] = true;
    critical section
    flag [0] = false;
    ==
}
```

Process 2

```
while (1)
{
    while (flag [0]);
    flag [1] = true;
    critical section
    flag [1] = false;
    ==
}
```

context switch

- Entry section
- Exit section

(mutual exclusion fails over here ... and both process enter into critical section when context switch takes place

- It does not guarantee the mutual exclusion
- It means process 1 and process 2 are in critical section at same time.

- problem with attempt 2

The flags are set after we break from the while loop

→ software solution (attempt 3)

shared variable

flag [0] = flag [1] = false

Process 1

```
while (1)
{
    flag [0] = True;
    while (flag [1]);
    critical section
    flag [0] = false;
    ==
}
```

Process 2

```
while (1)
{
    flag [1] = True;
    while (flag [0]);
    critical section
    flag [1] = false;
    ==
}
```

C.S. deadlock

LectureNotes.in

Scanned by CamScanner

- It achieves mutual exclusion
- It does not achieve progress as these two processes are deadlocked.

→ Peterson's solution

shared variable

```
turn : 1 or 2.
flag[0] = flag[1] = false
```

Process1

```
while(1)
{
   flag[0] = True;
   turn = 2;
   while(flag[1] AND turn == 2);
   critical section
   flag[0] = false;
}
```

Process 2

```
while(1)
{
   flag[1] = True;
   turn = 1;
   while(flag[0] AND
         turn == 1);
   critical section
   flag[1] = false;
}
```

{ flag[0] = true : means process1 is interested to enter into critical section.
turn = 2 : means turn of process 2.

- The deadlock is broken over here due to 'turn' variable because turn can be either 1 or 2 therefore the tie is broken. only one process enters into the critical section at a time so it solves critical section problem for two processes.
It also satisfies all the three required cond^n s.

11/8/17

→ SEMAPHORE

It is a special kind of integer variable which can be initialized and can be accessed only by two atomic operations. such as P() or wait() operation and V() or signal() operation.
atomic means no context switch in b/wn.

Scanned by CamScanner

- semaphore s ;

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

V
```
Signal (s)
  {
    s++;
  }
```

{s is a variable of semaphore type it can do P() , V() wait() or signal() opn.

- semaphore mutex ;
  mutex = 1 ;

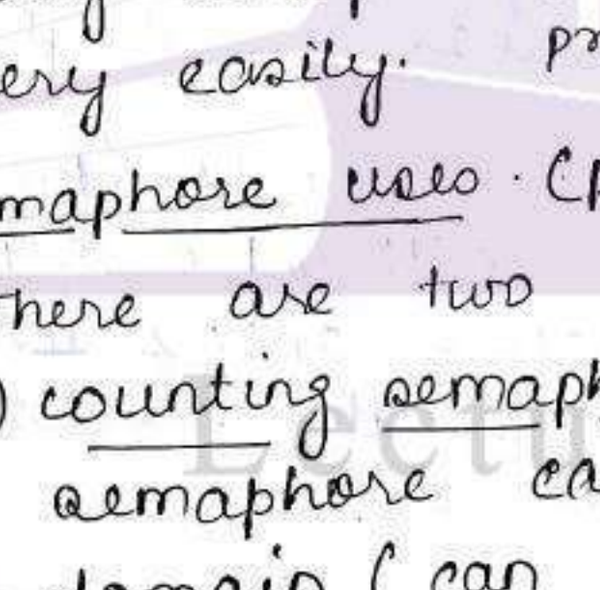
P_i :
```
P(mutex)
critical section
V(mutex)
Remainder section
```

P_j :
```
P(mutex)
critical section
V(mutex)
Remainder section
```

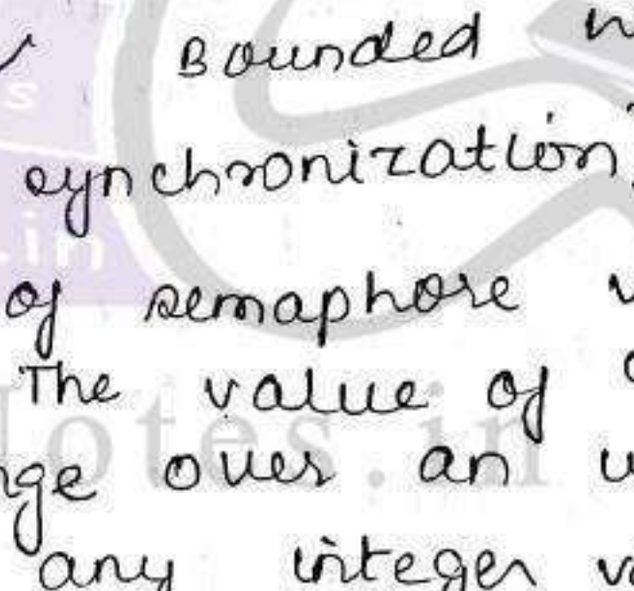
- using semaphore, we can implement mutual exclusion very easily. progress ✓ Bounded waiting ✗

- semaphore uses (process synchronization).
  - There are two types of semaphore variable
    i) counting semaphore : The value of counting semaphore can range over an unrestricted domain (can store any integer value).
    ii) binary semaphore : The value of binary semaphore can range only between 0 and 1.

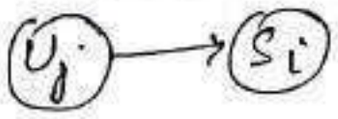
Program1



Program2



Scanned by CamScanner



Restriction
Uj must be executed before Si for right output

```
semaphore sync;
sync = 0;
```

Program-1

```
≡
P(sync)
Si;
≡
```

Prog-2

```
≡
Uj
V(sync)
≡
```

Proj 1

```
S1
S2
P(sync)
Si
SiH
≡
Sn
```

Proj 2

```
U1
U2
i
Uj
V(sync);
UjH
Um
```

- semaphore implementation (to save time & power of CPU due to busy waiting)

- Busy waiting : it is consuming CPU time without doing any useful work.
- This busy waiting can be eliminated by modifying the semaphore structure.

Now we define semaphore structure with two fields.

i) integer ii) list

i) integer : it contains the value of semaphore variable.

ii) list : it contains no. of processes that are waiting.

c:-

```
typedef struct
{
    int value;
    struct process *list;
}semaphore;
```

Scanned by CamScanner

- wait() semaphore operation can be defined as:

```
wait (semaphore *s)
{
    s→value--;
    if (s→value < 0)
    {
        add this process to s→list;
        block();
    }
}
```

// Now process is not in running state but in block state.

- signal() semaphore operation can be defined as:

```
signal (semaphore *s)
{
    s→value++;
    if (s→value >= 0)
    {
        remove a process P from s→list;
        wakeup(P);   // block state to running state.
    }
}
```

16/8/17

→ Synchronization hardware.

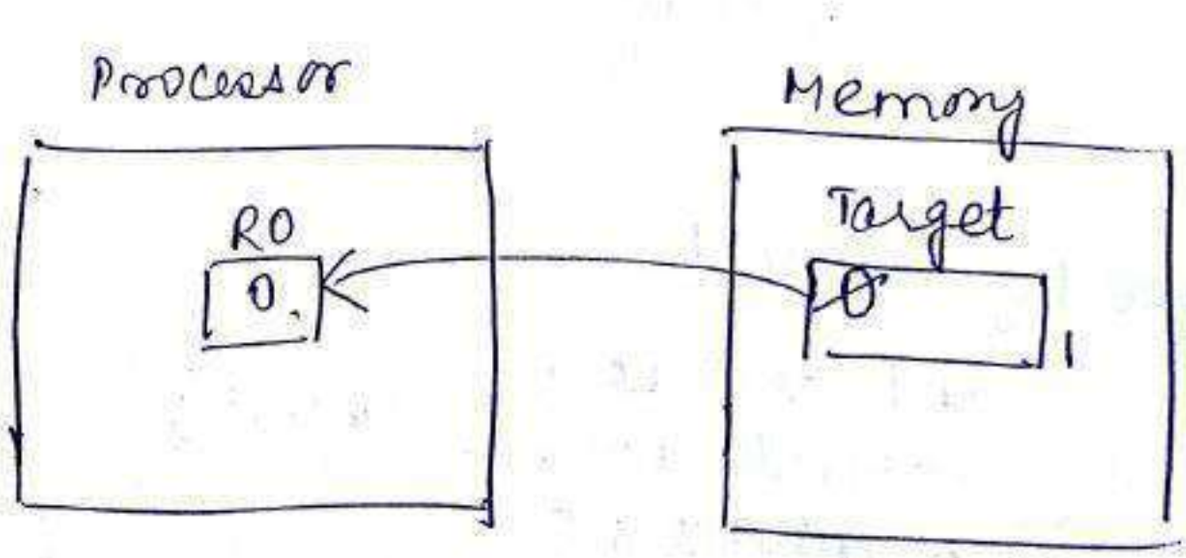
- Some hardware instructions can be used effectively in solving the critical section problem.

```
boolean test_and_set (boolean * target)
{
    boolean rev = *target;
    *target = True;
    return rev;
}
```

First times

Return

0.
1
1
1
:



Scanned by CamScanner

first time it returns 0 rest time 1.

- Mutual exclusion implementation using test-and-set function.

Process 1 lock = 0

```
while (1)
  {
     while( test_and_set (&lock) == 1);
     critical section;
     lock = 0.
  }
```

achieves mutual exclusion but not others.

Note* It Intel processor does not support test-and-set function.

```
int compare_and_swap (int *value, int expected, int new_value).
                      0                         1
{
    int temp = *value;
    if (*value == expected)
        *value = newvalue;
    return temp;
}
```

atomic

1st value = 0 expected = 0
new_value = 1.
temp = 0.
value = 1

memory
value
01.

2nd

- mutual exclusion implementation with compare_and_swap function

Process 1

```
While (1).
  {
     while (compare_and_swap (&lock, 0, 1) != 0);
      critical section:
      lock = 0;
  }
```

mutual exclusion satisfied
rest not satisfied

Scanned by CamScanner

- Another algorithm using test_and_set function (instruction that satisfies all the three critical section requirements.

Global variables.

```
boolean waiting[n] = false;
boolean lock = false
```

Pi:

```
while (1).
{
waiting[i] = true;   // Pi is waiting to enter into the C.S
key = true;  // local variable
While(waiting[i] && key)
    key = test_and_set (& lock);
waiting[i] = false;
```

C.S

critical section.

```
j = i+1 % n;
while ((j != i) && !waiting[j]).
    j = (j+1) % n;
if (j == i)
lock = false; // coming back to initial state(false)
else
waiting[j] = false; //
}.
```

allows next process to enter into critical section

These two will execute when processes are interested

P0 P1 P2 P3 P4

P0 P1 P2 P3 P4 not interested

Pj :

waiting[i] = false Initially

true false

key = true.

Now Pj wants to enter while we are in C.S of Pi

waiting[j] = true key = true

Scanned by CamScanner

It satisfies all three condn's for critical section.

17/8/17

fork()

→ Mutex locks

Mutex
Mutual exclusion.

- The hardware based sol^n to all the criticial section problem are generally inaccessible to application programmers.
- To avoid this problem, best designer build a software tools to solve critical section problem which is called mutex lock.
- Mutex locks is used to protect critical section and also prevents race condition.
- A process must acquire the lock before entering to the critical section and release the lock when it exits from the critical section.

```
while (1)
{
    acquire lock
        critical section
    release lock
}
```

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

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

due to busy waiting other process seem to be spinning so it is also called spin lock.

Scanned by CamScanner

- The main disadvantage of this implementation is busy waiting.
- while a process is in its critical section, any other process that tries to enter its critical section must loop continuoely in the call to acquire().
- These types of mutex lock is called spin lock. VVI*
- because the process spins while waiting for the lock to become available

- Deadlock and starvation due to semaphore.

- A deadlock problem will arise when two or more processes are waiting indefinitely for an event that can be caused only by one of the waiting process.

```
semaphore S, Q;
S = 1; Q = 1;
```

P0.

```
wait (S);
wait (Q);
- - - -
- - - -
- - - -
signal (S);
signal (Q);
```

critical section

P1.

```
wait (Q);
wait (S);
- - - -
- - - -
- - - -
signal (Q);
signal (S);
```

Rough.

S = 1
Q = 1
wait (S) S=0
(Q) Q=0.
C.S enters
signal (S) S=1.
Q=1

Q = 0
S = 0.
P1 blocked.
then P1 can execute.

Rough

wait (S).
context switch. S=0.
Now execute P0 or P1
wait (Q). busy waiting
deadlock

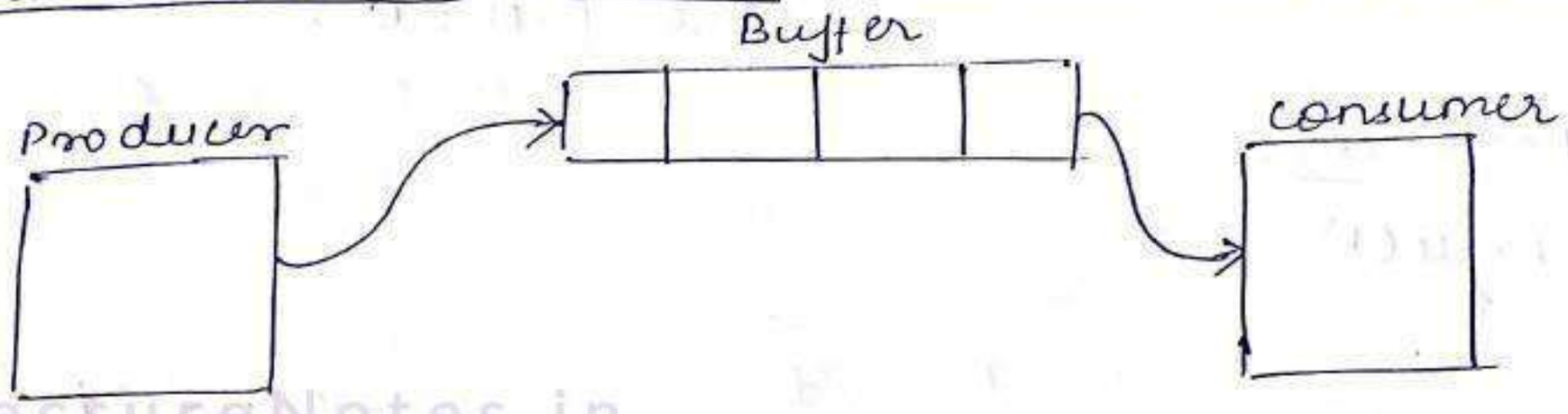
wait (Q). Q=0.
wait (S)
busy waiting

Scanned by CamScanner

{ if two signal may could switch together then 2 processes may enter into C.S. }

→ classic problems of synchronization .(VVI)

i) producer-consumer problem / Bounded-buffer problem.



consumer

- It is also known as Bounded-buffer problem as buffer size is bound
- producer produces an item and stores in buffer, consumer consumes the item from buffer
- It has 2 problems
  i.) producer is interested to produce but the buffer is full
  ii.) consumer is interested to consume but the buffer is empty.
- Producer and consumer share the following data-structure.

```
int n;
Semaphore mutex = 1; // mutual exclusion accessed
                        to the buffer.
Semaphore empty = n; // initially all the n positi
                        -ns of buffer are empty.
Semaphore full = 0; // zero buffer is full mea
                        all are empty.
```

Producer

```
{ ....
```

LectureNotes.in

Scanned by CamScanner

18/8/17

```
int n;
semaphore mutex = 1;
semaphore empty = n;
semaphore full = 0;
```

Producer:

```
while(1)
{
  ---
  /* produce an item in next_produced */
  ---
  wait(empty);
  wait(mutex);
  /* add next_produced to the buffer */
  signal(mutex);
  signal(full);
}
```

entry section

c.s

E.s

Consumer

```
while(1)
{
  wait(full);
  wait(mutex);
  /* remove an item from buffer to new_consumed */
  signal(mutex);
  signal(empty);
  /* consume the item in next_produced */
  ---
}
```

22/8

(2.)

LectureNotes.in

Scanned by CamScanner

22/8/17

(2.) Reader-writer problem

1. R1, R2, R3 --- Rn. ✓
2. R1, W1 ×
3. W1, R1 ×
4. W1, W2 ×

```
semaphore wrt = 1;
semaphore mutex = 1;
int readcount = 0;
```

writer

```
while(1)
{
    wait(wrt);
    /* writing is performed */
    signal(wrt);
}
```

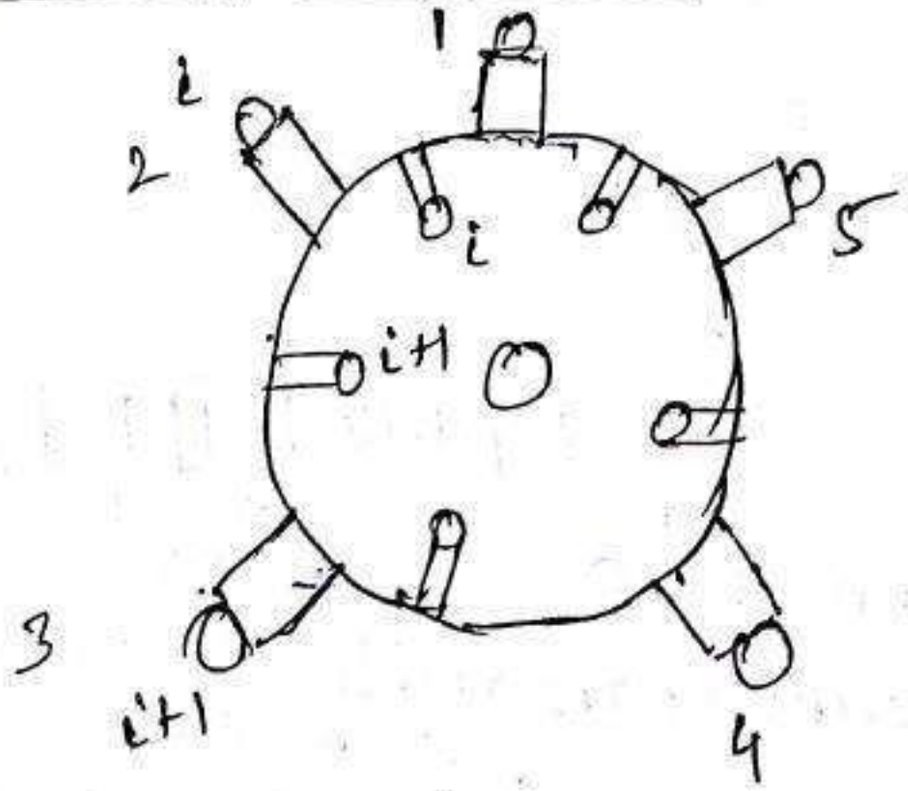
C.S. writer gets blocked (ii) case

Reader:

```
while(1)
{
    wait(mutex);
    read_count++;
    if(read_count==1)
    wait(wrt);
    signal(mutex);
    C.S. /* Reading is performed */
    wait(mutex);
    read_count--;
    if(readcount==0)
    signal(wrt);
    signal(mutex);
}
```

atomic

(3.) The Dining Philosopher problem



Scanned by CamScanner

- when a philosopher thinks, she does not interact with her ~~colleagues~~ colleagues.
- From time to time a philosopher gets hungry and tries to pick off the two chopsticks that are closest to her (left and right)
- A philospher may pick up only one chopstick at a time
- She cannot pick up a chopstick that is already in the neighbour's hand.
- when a hungry philospher has both chopsticks at the sametime, she eats without releasing the chopsticks
- when she finishes eating, she puts down both the chopsticks and start thinking again.
- A philospher pick up the chopstick by executing wait() operation and releases the chopstick using signal() operation.

```
Semaphore chopstick[5]; // Initialize to 1.
do
{
    wait (chopstick[i]);
    wait (chopstick[(i+1)%5]);
    - - -
C.S. | /* eat for a while */ |
    - - -
    signal [chopstick[i]);
    signal [chopstick[(i+1)%5]);
    - - -
    /* think for a while */
} while(true)
```

Chopstick

1	1	1	1	1
0	1	2	3	4

- This guarantees that no two neighbours are eating simultaneously.

Scanned by CamScanner

- Suppose that all five processes become hungry at the same time and each grabs her left chopstick and the right chopstick is not available for any philosopher. This leads to deadlock.
- possible remedies to avoid deadlock are
  i.) allow atmost four philosphers to sit simultaneously on the table
  ii.) allow a philospher to pick up her left chopstick only if both chopsticks are available.
  iii.) an odd no. of philospher picks up her left chopstick first and then right chopstick whereas even-numbered philospher picks up her right chopstick first and then left chopstick.

MONITORS.

MONITORS.

Although semaphore provides a convienient and efficient mechanism for process synchronization but missuing the semaphore results in error.

1.) wait(mutex) ⇒ signal(mutex)
c.s c.s
signal(mutex) wait(mutex)

problem: several process enter into the critical section simultaneously, violating mutual exclusion requirement.

2.) wait(mutex)
c.s
wait(mutex)

problem: deadlock occurs.

3) Suppose that a process omits wait(mutex) or signal(mutex) or both, then either mutual exclusion is violated or deadlock will occur.

Scanned by CamScanner

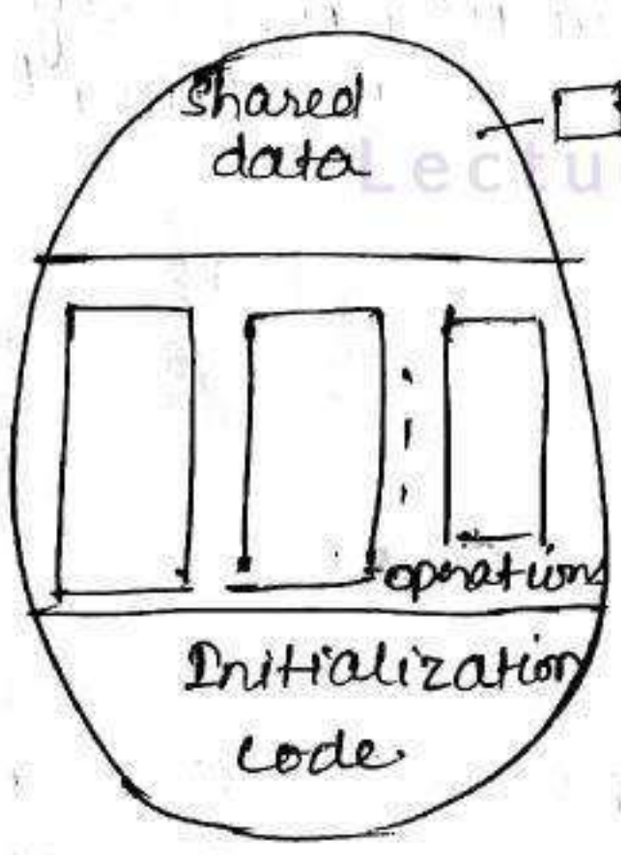
4) To avoid these problems, monitor is introduced.

```
monitor monitor_name
{
    /* shared variable declaration */
    function P1( )
    {
    }
    function P2( )
    {
    }
    ==
    ==
function Pn( )
    {
    }
    initialization_code(-)
    {
    }
}
```

Syntax of monitor

→ Monitor usage

A monitor type is an ADT that includes set of programmer defined operations that are provided with mutual exclusion within the monitor.



* Schematic view of * monitor

Scanned by CamScanner

- A programmer who needs to write synchronization scheme can also define variables of type 'condition'.

  condition x, y;

- The operation x.wait() means that the process involved in this operation is suspended under θ until another invokes. i.e x.signal()

The fork() system call.

23/8/17

Parent

```
{
fork();
pid = ....;
}
```

returns CPID.

Child - returns 0.

check { n processes → 2^n processes (fork)

child process $2^n - 1$

system call fork() is used to create process. It takes no arguments and returns a process ID.

The purpose of fork() is to create a new process which becomes the child process of the caller.

see pdf.

LectureNotes.in

Scanned by CamScanner

Absent days:-

After Midsem

Deadlocks.

A process request for resources. If the resources are not available at that time, the process enters a wait state. Waiting processes may never again change state, because the resources that have requested are hold by other waiting processes This situation is called deadlock.

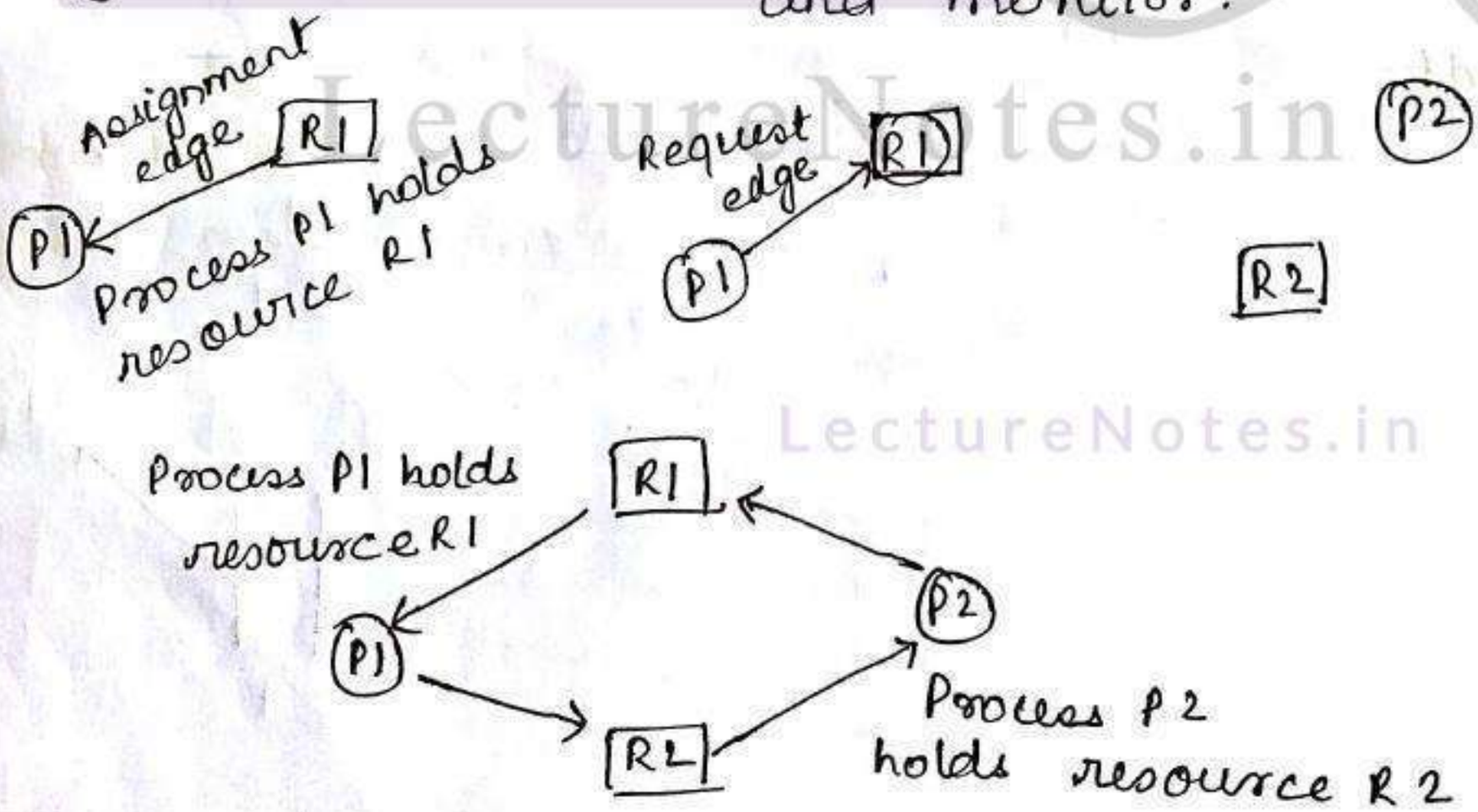
(P1) ⟶ [R1]

example of resources are - CPU cycles, memory space, IO devices etc.
A process must request a resource before using it. and must release the resource after using it.
A process may utilize a resource in following sequence -

1. Request (System call)
2. Use
3. Release (System call)

There are two types of resource -

1. Physical Resource - CPU, Memory, I/O.
2. Logical resource - Read only files, semaphore. and monitor.



If there is a cycle in the graph, there may be chances of deadlock.

Scanned by CamScanner

Necessary conditions for deadlock.-

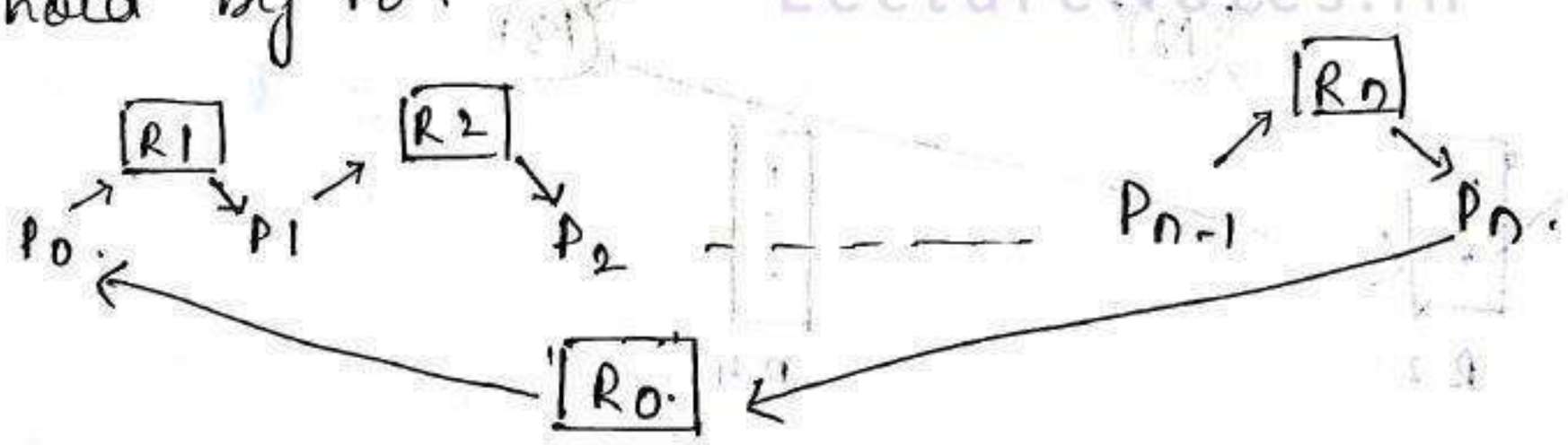
A deadlock situation can occur if the following 4 conditions hold simultaneously in the system-

1. Mutual exclusion. Only one process may use the resource at a time.

2. Hold and wait. A process must be holding atleast one resource and waiting to acquire additional resources that are currently being hold by the other processes.

3. No preemption: Resources cannot be preempted in the middle of the execution. A resource can be released only if the process has completed its task.

4. Circular wait: A set $\{P_0, P_1 \text{---} P_{n-1}, P_n\}$ of waiting process must exist such that P_0 is waiting for a resource that is hold by P_1, P_1 is waiting for the resource that is hold by P_2, --- P_{n-1} is waiting for a resource that is hold by P_n and P_n is waiting for a resource hold by P_0.



Scanned by CamScanner

Resource Allocation graph.

Deadlock can be described in terms of a directed graph called resource allocation graph. This graph consists of set of vertices 'V' and set of edges 'E'.

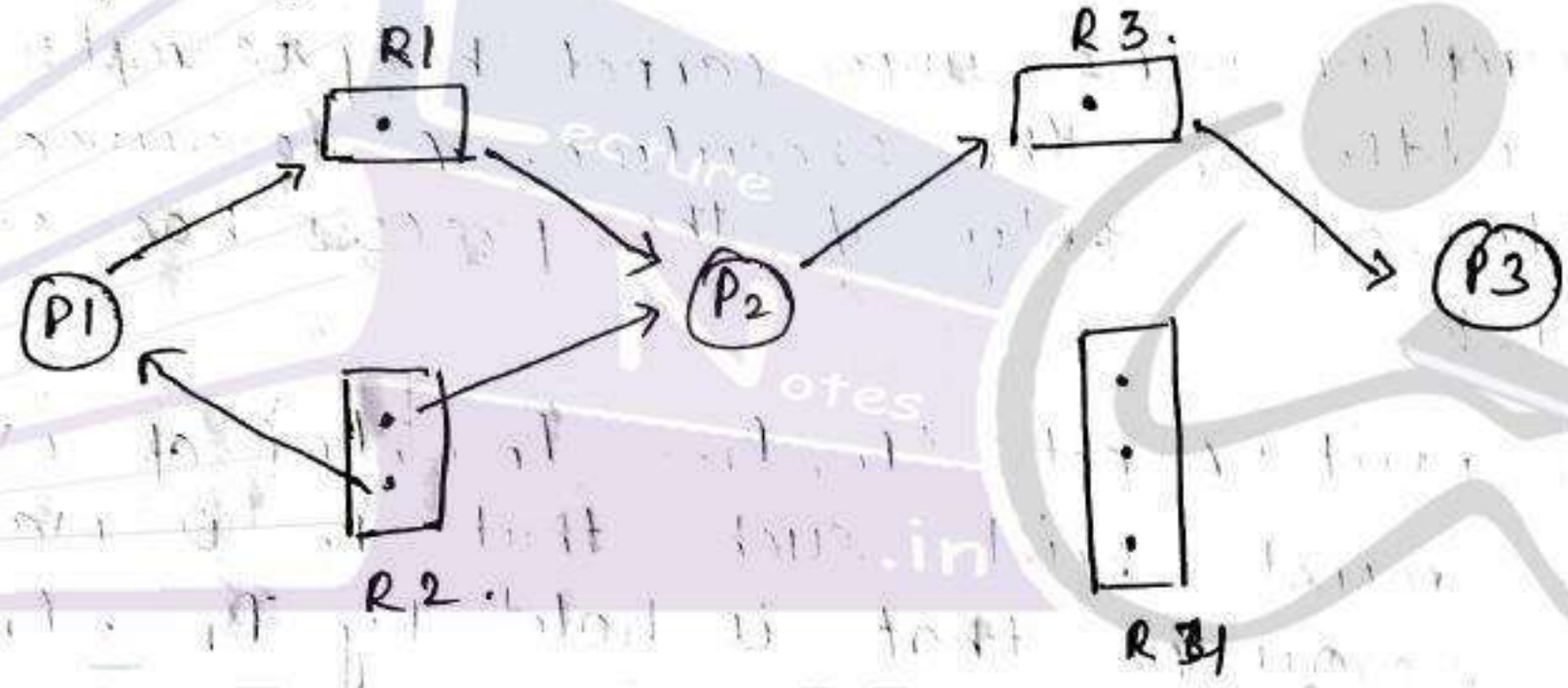
vertices are of two types -

1. ○ process
2. □ resource.

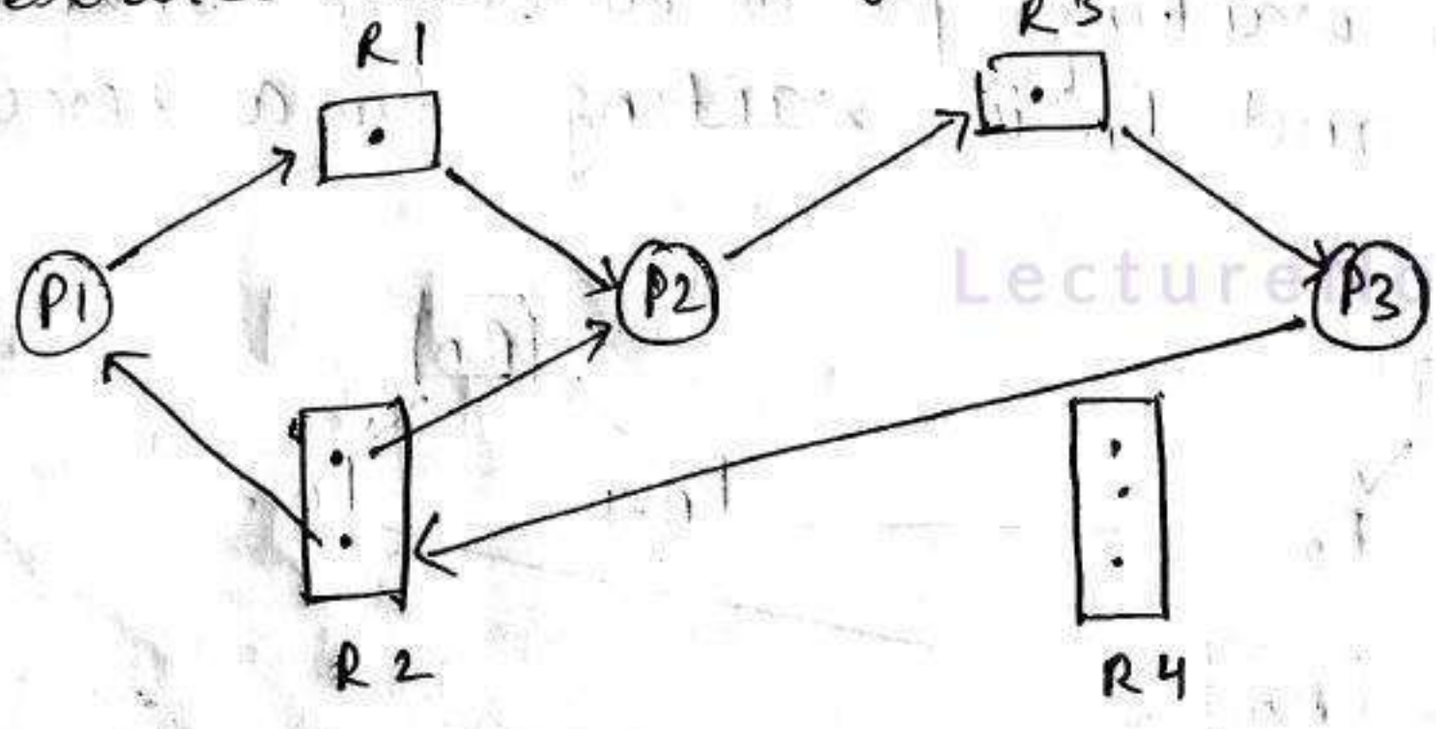
edges of two types -

1. (Pi) → [Rj] Request edge
2. (Pi) ← [Rj] Assignment edge.

example of Resource Allocation graph



Resource allocation graph without deadlock.

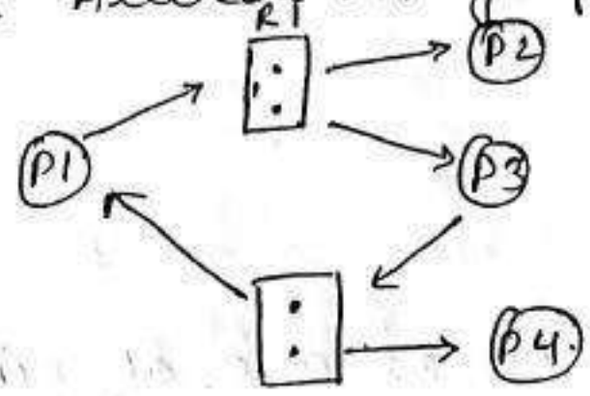


1. P1 → R1 → P2 → R3 → P3 → R2 → P1
2. P2 → R3 → P3 → R2 → P2.

Deadlocked processes are P1, P2, P3.

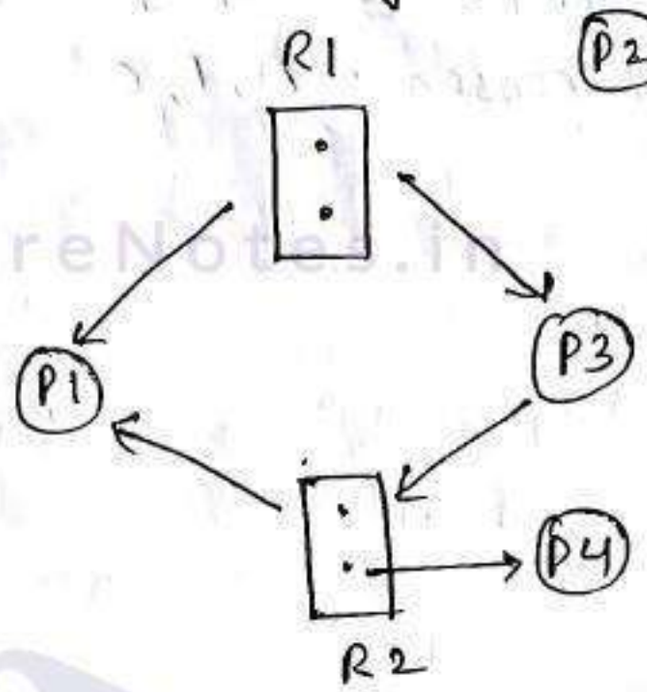
Scanned by CamScanner

Resource Allocation graph with deadlock.



Not in deadlock

Resource allocation graph without deadlock



→ If the graph contains no cycle then the processes are not in deadlock.

→ If the graph contains a cycle then a deadlock may exist.

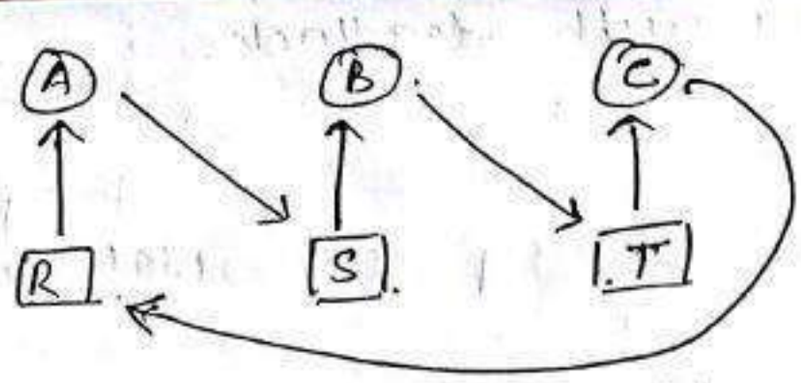
→ If each resource type has exactly one instance and a cycle exist in resource allocation graph implies deadlock.

→ If each resource type has several instances, then a cycle does not neccessarily imply that a deadlock has occured.

So, here a cycle in the graph is neccessary but not a sufficient condition for the existence of deadlock.

Q.) 1. (A) requests [R]
2. (B) " [S]
3. (C) " [T]
4. (A) " [S]
5. (B) " [T]
6. (C) " [R].

Scanned by CamScanner

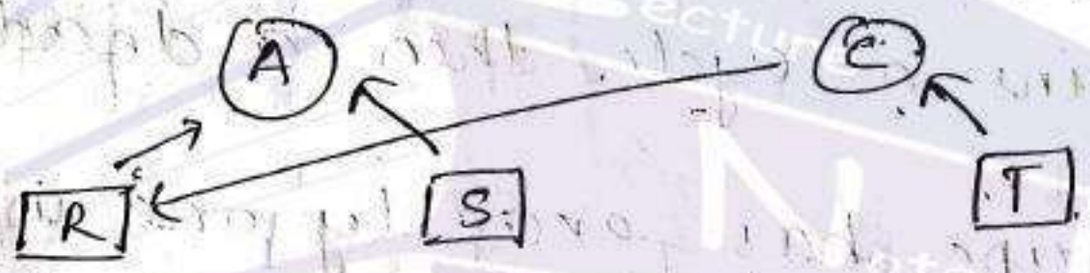


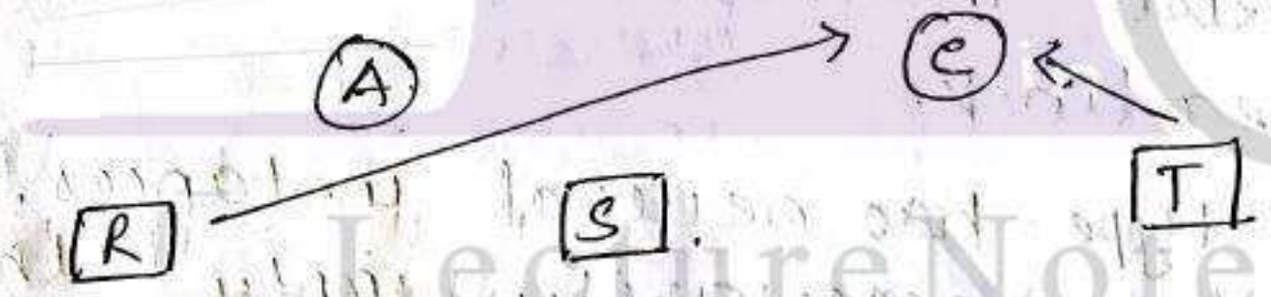
There is cycle and all resources are of single instance hence deadlock.

cycle : (A) → [S] → (B) → [T] → (C) → [R] → (A)

Deadlocked processes : A, B, C.

Q. 1. (A) requests [R]
2. (C) " [T]
3. (A) " [S]
4. (C) " [R]
5 (A) releases [R]
6. (A) releases [S]





No cycle hence no deadlock.

Methods for handling deadlock -
we can deal with the deadlock problem in one of the three ways

1. Deadlock prevention.
2. Deadlock avoidance.
3. Deadlock detection and recovery.

LectureNotes.in

Scanned by CamScanner

1.) Deadlock prevention - One can prevent deadlock by avoiding a policy that eliminates one of the conditions such as mutual exclusion, hold and wait, no preemption and circular wait.

2.) Deadlock avoidance - Deadlock avoidance requires the advanced additional information of resource allocation of the processes. By additional knowledge it can be determined whether there is a deadlock or not.

3) Deadlock detection and recovery - we can allow the system to enter into the deadlock state by holding 4 conditions - mutual exclusion, hold n wait, no preemption and circular wait and take action to recover it.

→ Deadlock prevention

Deadlock prevention is a set of methods for ensuring that ~~atleas~~ atleast one of the four conditions ME, hold and wait, no preemtion and circular wait cannot hold.

- Mutual exclusion - Mutual exclusion means only one process can use the resource at a time. It means resources are not shared by no. of processes at a time. so the resources are non-shareable in nature. TO avoid this condition use protocol which convert non-shareable resource to shareable resource

ex : Non shareable - Printer
shareable - Read only files.

Printer is not shared by no. of processes at a time, so we cannot convert non-shareable to shareable node.

Disadvantage - we cannot avoid deadlock by converting non-sharable to shareable for all resources

Scanned by CamScanner

12/9/17

- Hold and wait - To avoid hold and wait we must ensure that it does not hold any other resou
  Two protocols are used -
  1.) It allows a process to request and allocate all its resource before the execution begins.
  2.) Allows a process to request only when the process has none.
  It means a process can request any additional resource by releasing all the resources that is currently allocated. ex - printer, pendrive etc.
  Disadvantage :- Resource utilization is low and starvation may occur.

- No preemption - No preemption means resources are not released in the middle of the execution
  Two protocols required are -
  1.) If a process is holding some resource and requi other resources that cannot be immediately allocated to it, then all the resources currently being held are preempted.
  2.) If a process requests some resources we first check whether they are available or not. If they are available we allocate them. If the resources are not available we check whether they are allocated to some other resources that is waiting for additional reso. urce. If so we preempt the desired resource from the waiting process and allocate them to the request process.

  Disadvantage - It cannot be applied to resou ces like printer, cd drivers, tape drivers etc.

Scanned by CamScanner

must
resour
ocate
gins.
the
ional
that
etc.
d
are
tion
request
ly
rist
f
f
e
esO-
urce
-
oour-

12/9/17

- circular wait condition never holds ordering of all the resources in an increasing order of enumeration

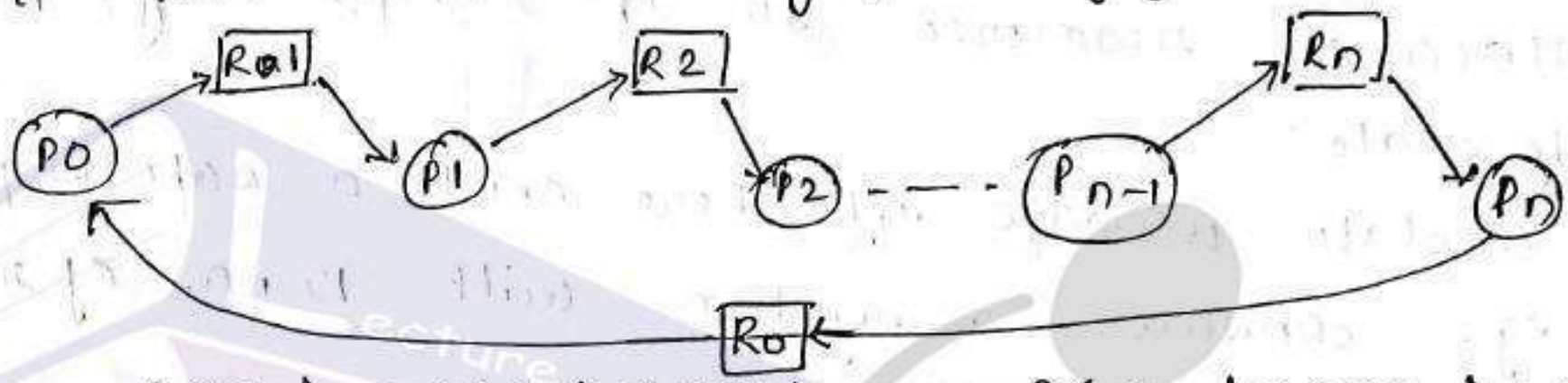
To prevent deadlock following protocols are used.

i.) each process can request resources only in an increasing order of enumeration

ii) a process can request resources of type R_j if and only if $F(R_j) > F(R_i)$

ii.) whenever a process request an instance of resource type R_j, it has released only resources ~~arrived~~ R_i such that $F(R_i) \geq F(R_j)$

$\{F(R_j)$: Resources hold by process $j\}$



$F(R_0) < F(R1) < F(R2) < \dots F(R_{n-1}) < F(R_n) < F(R_0)$

enumeration value of R_0.

$F(R_0) < F(R_1)$ ✓

$F(R0) < F(R2)$ ✓

$F(R0) < F(R_{n-1})$ ✓

$F(R_0) < F(R_n)$ ✓

$F(R_0) < F(R_0)$ X Here circular wait condition breaks and becomes false.

By transitivity we get

$F(R_0) < F(R_0)$ which is impossible so there can be no circular wait

disadvantage :- circular wait prevention may be inefficient, slowing down processes and denying resource access unnecessarily.

Scanned by CamScanner

2.) Deadlock avoidance.

- Deadlock avoidance algorithm dynamically examines the resource allocation state to ensure that a circular wait condition can never exist.
- resource allocation state is defined by no. of available and allocated resources and the maximum demand of the processes.
- If a process request for a resource the avoidance algorithm checks before the allocation of resources about the state of the system.
- If the state is safe, system allocate resources to the requesting process.
- If the state is unsafe, then the system do not allocate resource to the requesting process.

→ a.) safe state

a state is safe iff there exist a safe sequence

eg:- consider a system with 12 no. of resources and 3 processes

Processes.	Max. need.	Hold.	Need.
P_0	10	5	5. (unsafe)
P_1	4	2	2
P_2.	9	2	7 (unsafe)

currently available resources = Total no. of resources - Total no. of currently holding resources

$12 - 9 = 3$.

At time t_0, granting 3 resources to P_0 and P_2 will leave the system in unsafe state.

At time t_1, 2 resources are allocated to process P_1, the currently available = $3 + 2 = 5$.

$3 - 2 = 1$.

$1 + 4 = 5$ ← when all resources are released by P_1.

Scanned by CamScanner

At time t2, 5 resources are allocated to P0. It becomes safe and get executed.

After execution currently available = 5+5
= 10.

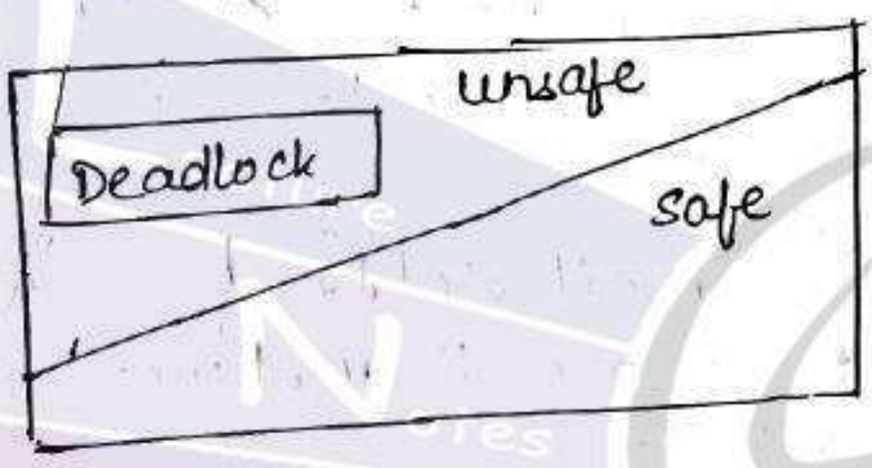
At time t3, 7 resources are allocated to P2, it becomes safe and get executed.

After execution 9+3. = 12.

currently available

The safe sequence is <P1, P0, P2>

- If there is safe sequence in your system, your system is safe so no deadlock.
- A safe state is not a deadlock state.
- A deadlock state is an unsafe state but not all unsafe states are deadlocks.
- An unsafe state may lead to a deadlock.



13/9/17.

b.) Resource allocation graph algorithm.

For deadlock avoidance, resource allocation graph contains a resource allocation system with only one instance of each resource type.

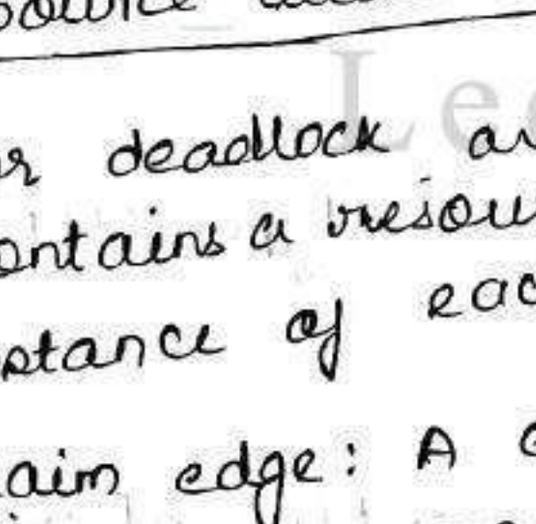
claim edge: A claim edge Pi indicates that process Pi may request resource Rj at some time in future to:

t_1 : Pi ⇢ Rj (CE)

t_1 : Pi → Rj (RE)

t_2 : Pi ← Rj (AE)

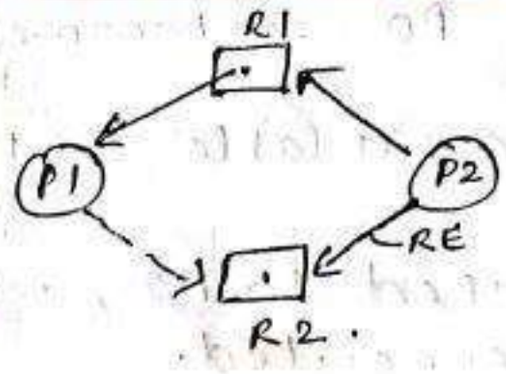
t_3 : Pi ⇢ Rj (CE)

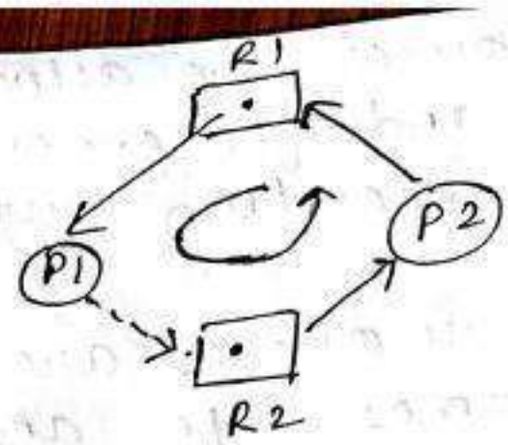


{P1, P2 may request R2 in future}

Resource allocation graph for deadlock avoidance.

Scanned by CamScanner





If P1 requests R2. then there will be cycle. this happens due to R2 → P2 this is unsafe state

i) Saf
Th
u
•
•

→ Banker's Algorithm.

- Resource allocation graph algorithm is not applicable to resource allocation system with multiple instances of each resource type.
- Let n be the no. of processes and m be the no. of resource time.
- Available : it is a vector of length m indicates no. of available resources of each type.

Available [j] = k.

it means there are k instances of resource type Rj are available

- Max : max [i, j] = k

it means process Pi request atmost k instances of resource type Rj

- Allocation : no. of resources of each type currently allocated to each process.

Allocation [i, j] = k

it means process Pi is currently allocated with k instances of resource type Rj

- Need : It indicates the remaining resources needed by each process

Need [i, j] = k.

it means process Pi may need k more instances of resource type Rj to complete its task

ii

Need [i, j] = Max [i, j] - Allocation [i, j]

Scanned by CamScanner

i) safety algorithm.

This algorithm finds whether the system is safe or unsafe.

- step 1.

  let work and finish be vectors of length m and n.

  work := Available

  Finish[i] = false for i = 1, 2 ---- n.

- step 2.

  Find an i such that

  a.) Finish[i] = False {// process must not be completed

  b.) Needi ≤ work. {// need should be ≤ available

  if not such i is found, go to step 4.

- step 3.

  work := work + Allocation; {// we need to release when process executes completely.

  Finish[i] = True.

  goto step-2 {// for other processes here only 1 process gets completed.

- step 4.

  if Finish[i] = true for all i then the system is in safe state

ii.) resource request algorithm.

when a request for a resource is made by process Pi then following actions are taken :-

- step 1.

  If Requesti ≤ Needi, goto step 2 otherwise an error will generate

- step 2.

  If Requesti ≤ Available, goto step 3 otherwise Pi must wait till the resources are available

- step 3.

  The system allocate the requested resources to process Pi by modifying the following :-

  Available := Available - Requesti

  Allocation := Allocation i + Requesti

  Needi := Needi - Requesti

Scanned by CamScanner

If resource allocation state is safe
is completed and process Pi is allocated with resource

If the state is unsafe then process Pi must wait for request i

example

consider a system with 5 processes P0 to P4 and 3 resource type A, B and C. Resource type A has 10 instances, B has 5 instances and C has 7 instances.

i.) check the system is safe or unsafe
ii.) Suppose process P1 request ~~one~~ 1 more instance of resource type A and 2 more instances of resource type C. Can the request be granted?

14|9|17.

At time T_0

Processes	Allocation (Hold)			Max			Available			Need.		
	A	B	C	A	B	C	A	B	C	A	B	C.
P_0 (False)	0	1	0	7	5	3	3	~~3~~	2	7	4	3 X✓
										1	2	2 ✓
P_1 (F)(T)	2	0	0	3	2	2				6	0	0 X'
P_2 (F)	3	0	2	9	0	2				0	1	1 ✓
P_3 (F)(T)	2	1	1	2	2	2				4	3	1 ✓
P_4 (F)(T)	0	0	2	4	3	3						

Ans:-

i) Available = Total - allocation
(10 5 7) - (7 2 5)
= (3 ~~3~~ 2)

ii) Apply safety algorithm

step1 work = available = (3 3 2)
finish[i] = false

step2 find i such that Needi ≤ work & finish[i]=
P0 X
P1 ✓ Needi ≤ work.
Need of P1 (1 2 2) ≤ (3 3 2)

Scanned by CamScanner

work = work + Allocation
= (3 3 2) + (2 0 0)
work = (5 3 2) Finish[i] = true

Need P_3 (0 1 1) ≤ work (5 3 2)
work = work + Allocation.
(5 3 2) + (2 1 1)
work = (7 4 3)

Need P_4 (4 3 1) ≤ work (7 4 3)
work = work + Allocation
= (7 4 3) + (0 0 2)
work = (7 4 5)

Next loop.

Need P_0 (7 4 3) ≤ work (7 4 5)
work = work + allocation
= (7 4 5) + (0 1 0)
work = (7 5 5)

Need P_2 (6 0 0) ≤ work (7 5 5)
work = work + allocation
(7 5 5) + (3 0 2)
= (10 5 7).

Safe sequence: < P1 P3 P4 P0 P2 >.
or
< P1, P3, P4, P2, P0 >
is in safe state.

Hence the system

ii.)

	A	B	C.
P1	1	0	2

Request. Resource, request algorithm.

1.) Request P_1 ≤ Need P1.
(1 0 2) ≤ (1 2 2) ✓

2.) Request P_1 ≤ Available
(1 0 2) ≤ (3 3 2) ✓

If these cond^n^s are false then request will not be granted here granted.

LectureNotes.in

Scanned by CamScanner

available = available - Request$_{P1}$.

(3 3 2) - (1 0 2)

= (2 3 0)

allocation$_{P1}$ = allocation$_{P1}$ + request$_{P1}$

(2 0 0.) + (1 0 2)

= (3 0 2)

Need$_{P1}$ = Need$_{P1}$ - Request$_{P1}$

= (1 2 2) - (1 0 2)

= (0 2 0).

The new state of the system will be:-

Process.	Allocation			Max			Available			Need		
	A	B	C.	A	B	C	A	B	C	A	B	C
P_0	0	1	0	7	5	3	2	3	0	7	4	3
P_1	3	0	2	3	2	2				0.	2	0
P_2	3	0	2	9	0	2				6	0	0
P_3	2	1	1	2	2	2				0	1	1
P_4	0	0	2.	4	3	3.				4	3.	1

Now you must determine whether this new system is safe or not. To do so we execute safety algorithm again and find the safe sequence. as. this: ~~safe~~ <P1 P3 P4 P0 P2> which satisfies our safety requirement. Hence we can grant the request for process P_1

Scanned by CamScanner

15/9/17

3.) Deadlock detection.

If a system does not employ either deadlock prevention or deadlock avoidance algorithm then a deadlock may occur so a system must provide.

i.) an algorithm that examines the deadlock has occured or not (deadlock detection algorithm).

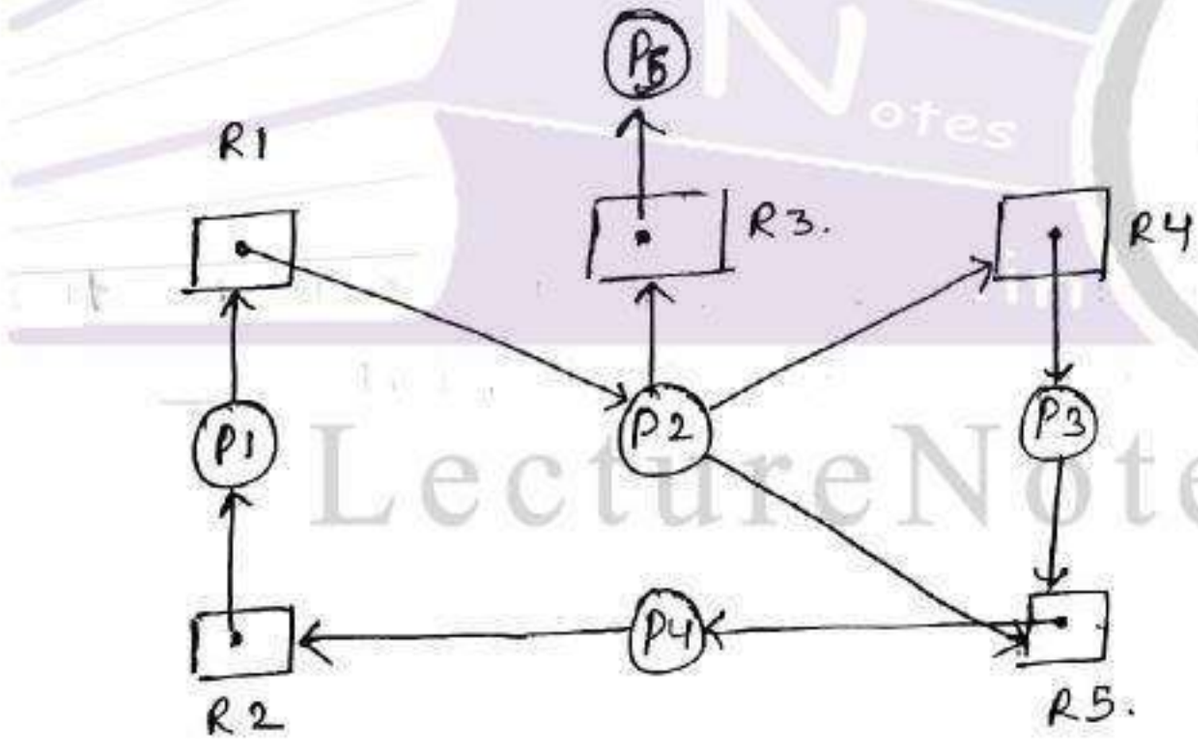
ii.) an algorithm to recover from deadlock.

a). • single instance of each resource type.

If all the resources have only a single instance then wait-for graph is used as deadlock detection algorithm.

We obtain wait-for graph ~~from~~ from resource allocation graph by removing resource nodes and collapsing the appropriate edges.

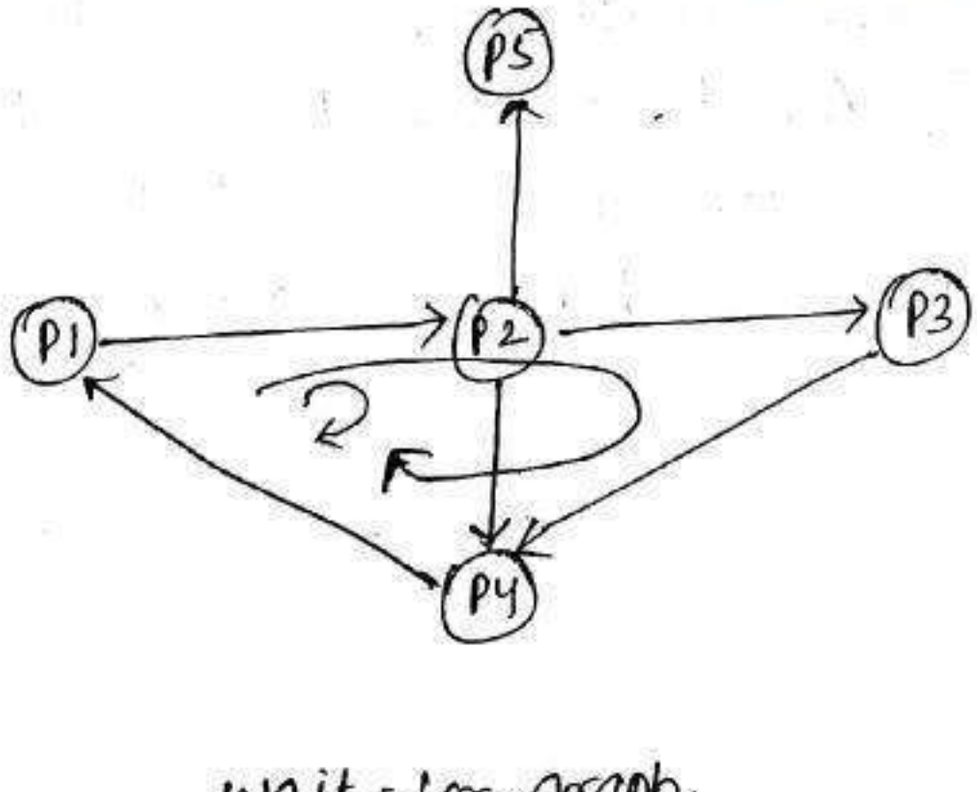
A deadlock exist in the system iff the wait for graph ~~consist~~ contains a cycle.



(Resource allocation graph).

$P_1 \rightarrow R1 \quad R1 \rightarrow P2$

$\therefore P1 \rightarrow P2$



wait-for-graph.

Scanned by CamScanner

b.) • multiple instances of a resource type.

The deadlock detection algorithm is applicable to a system with multiple instances of each resource type.

deadlock detection algorithm

step1. : Let work & finish be vectors of length m and n, respectively.

work := Available for i = 1, 2 ---- n.

if Allocation$_i$ ≠ 0, then Finish[i] = false.

otherwise, Finish[i] = True.

step 2. : For an index i such that

a) Finish[i] = false

b) Request$_i$ ≤ work

if no such i exist, go to step 4.

step3.

work := work + Allocation$_i$

Finish[i] = True

goto step 2.

step4 if Finish[i] = False for some i, $1 \le i \le n$ then the system is in deadlock state.

~~example~~

example

we consider a system with 5 processes P_0 to P_4 and these resources are A, B and C. Resource type A has 7 instances, B has 2 instances and C has 6 instances. Suppose at time T_0 we have following resource allocation state.

~~Process~~

LectureNotes.in

Scanned by CamScanner

Process	Allocation			Request			Available		
	A	B	C	A	B	C	A	B	C
P_0	0	1	0	0	0	0	0	0	0
P_1	2	0	0	2	0	2			
P_2	3	0	3	0	0	0			
P_3	2	1	1	1	0	0			
P_4	0	0	2	0	0	2			

10/10/17.

Step 1 = work = 000 = available = 0 0 0.

Step 2 = Request$_0$ (0 0 0) $\leq$ work (0 0 0) ✓

Step 3 = work$_0$ = work$_0$ + allocation$_0$.

000 + 010.

Request$_1$ (2 0 2) $\leq$ work (0 1 0) ✗

Request$_2$ (0 0 0) $\leq$ work (0 1 0) ✓

010 + 3 0 3 = 3 1 3.

Request$_3$ (1 0 0) $\leq$ work (3 1 3) ✓

work = 3 1 3 + 2 1 1. = 5 2 4.

Request$_4$ (0 0 2) $\leq$ work (5 2 4) ✓.

work = 5 2 4 + 0 0 2 = 5 2 6.

Request$_1$ (2 0 2) $\leq$ work (5 2 6) ✓.

work = 5 2 6 + 2 0 0 = 7 2 6.

safe sequence : $\langle P_0, P_2, P_3, P_4, P_1 \rangle$

so the system is not in the deadlock state.

→ suppose the process P2 makes an additional request for an instance of resource type C then find out which processes are in deadlock state.

	Request
=P2	0 0 1

work = available = 0 0 0.

Request$_0$ (0 0 0) $\leq$ work (0 0 0) ✓

work = 0 0 0 + 0 1 0 = 0 1 0.

Request$_1$ (2 0 2) $\leq$ work (0 1 0) ✗. P0 is True rest all false.

Request$_2$ (0 0 1) $\leq$ work (0 1 0) ✗.

Request$_3$ (1 0 0) $\leq$ work (0 1 0) ✗

Request$_4$ (0 0 2) $\leq$ work (0 1 0) ✗

Scanned by CamScanner

Now the system is in deadlock because after execution of P_0, the no. of available resources is not sufficient to fulfill the requests of other processes. Thus the process P_1, P_2, P_3, P_4 are deadlocked.

→ RECOVERING FROM DEADLOCK.

- when detection algorithm determines that deadlock exist, then we can deal with deadlock manually or automatically
- TO break the deadlock either we apply
  i) process termination
  ii) Resource preemtion.

i) process termination.

TO eliminate deadlock by aborting a process and releasing all its resources. We basically use two methods for process termination.

a.) abort all deadlock processes
b.) abort one process at a time until the deadlock cycle is eliminated.

criteria for termination
- priority
- duration of computation
- no. and type of resources allocated.
- type no. of process.

ii.) Resource preemtion.

we preempt some resources from process and give this to other processes until the deadlock cycle is broken.

preemtion of resources should follow these issues.

a.) selecting a victim. : we must determine which resources and which processes are to be preemted so that it will minimize the cost.

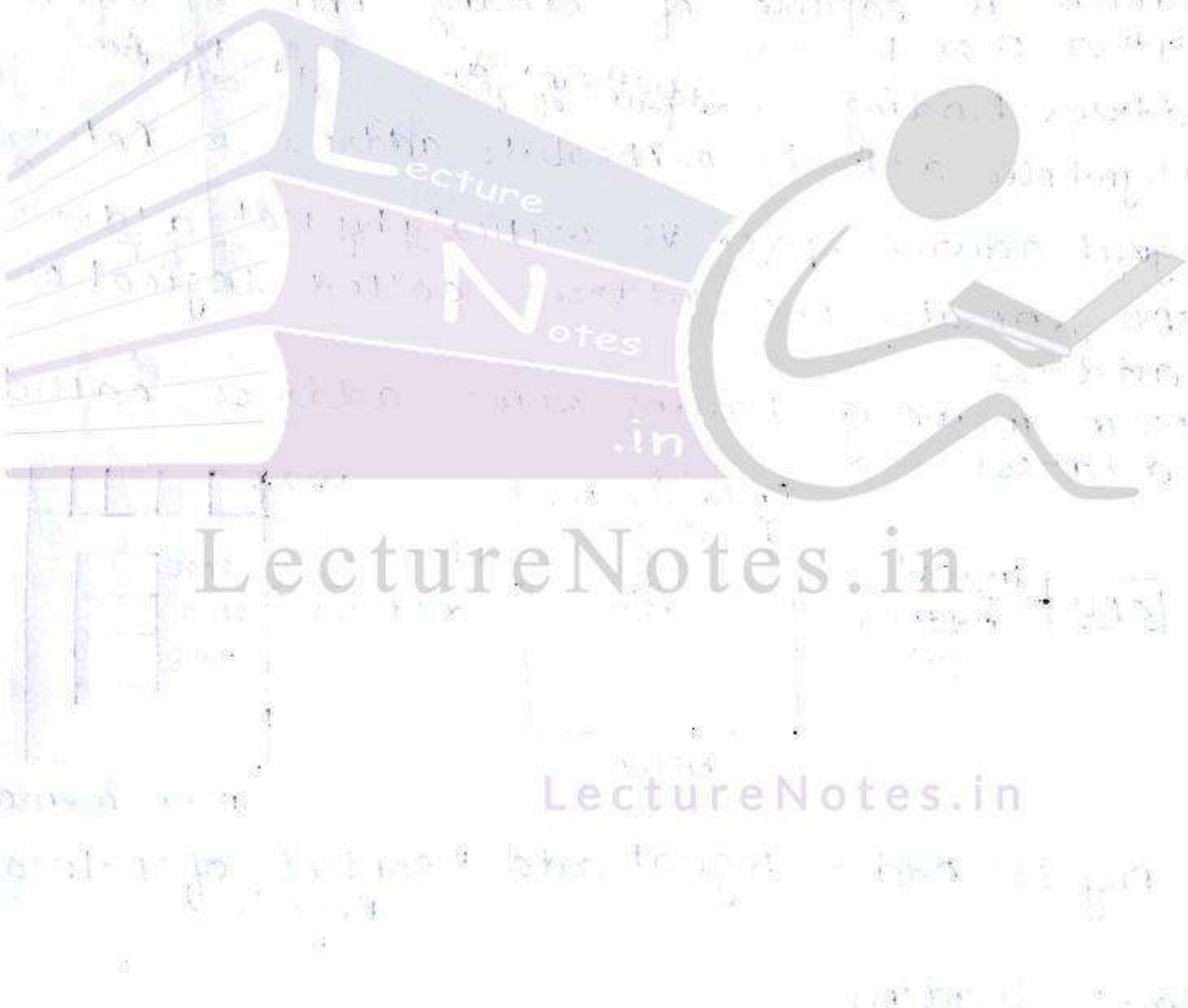
b.) rollback. : since the resources are preemted from a process, it cannot continue its normal execution thus we must rollback the process to some safe state and restart it from that state.
Generally it is difficult to determine a safe state so we go for total rollback.

Scanned by CamScanner

c) starvation :- to avoid starvation we should take care that resources ~~should~~ will not always be preemted from same process.
we must ensure that a process can be picked as a victim only for a finite no. of times.

Deadlock complete.

LectureNotes.in



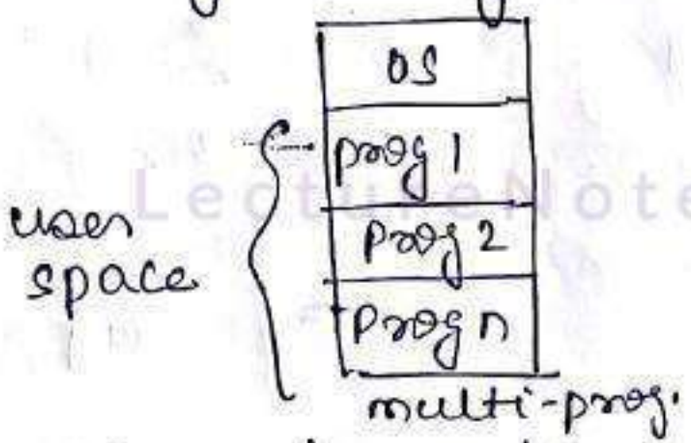
LectureNotes.in
LectureNotes.in

Scanned by CamScanner

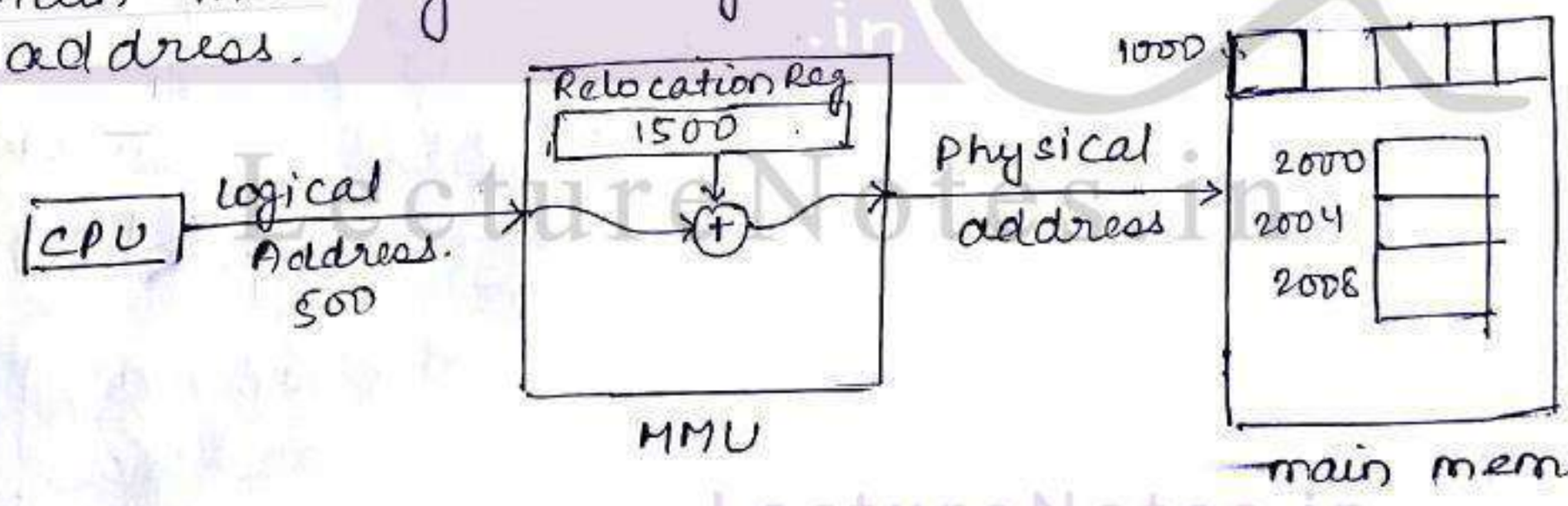
11/10/17

CHAPTER : MEMORY MANAGEMENT

- In uniprogramming main memory contains OS and a single program.
- In multi-programming main m/m contains OS and multiple program.
- The dynamic partition of memory by the OS is called memory management.



- memory - is a large collection of semi-conductor cells which is capable of storing 1 bit of information either 0 or 1
- Address binding -
  1. Symbolic address (address found in prog. eg: goto) 2. Absolute address (at the time of compilation) 3. Relocatable address (at the time of execution)
- Logical address space Vs physical address space
  - CPU generates one address called logical or virtual address.
  - Main memory having some address called physical address.



Physical Add = logical add + content of Relocation Register

- Dynamic loading -
  - Loading means load the program from secondary memory to main memory.
  - Types 1. static or compile time loading
    2. dynamic or run time loading.
  - If all the ~~loading~~ routines (functions) are loaded in the main memory at the time of compilation

Scanned by CamScanner

is called static loading.

- If all the ~~load~~ routines are loaded in the main memory at the time of execution then it is called dynamic loading.
- In dynamic loading, routines are not loaded until it is called.
- Advantage (dynamic loading)
  i) an unused routine is never loaded. so we can obtain better memory utilization.

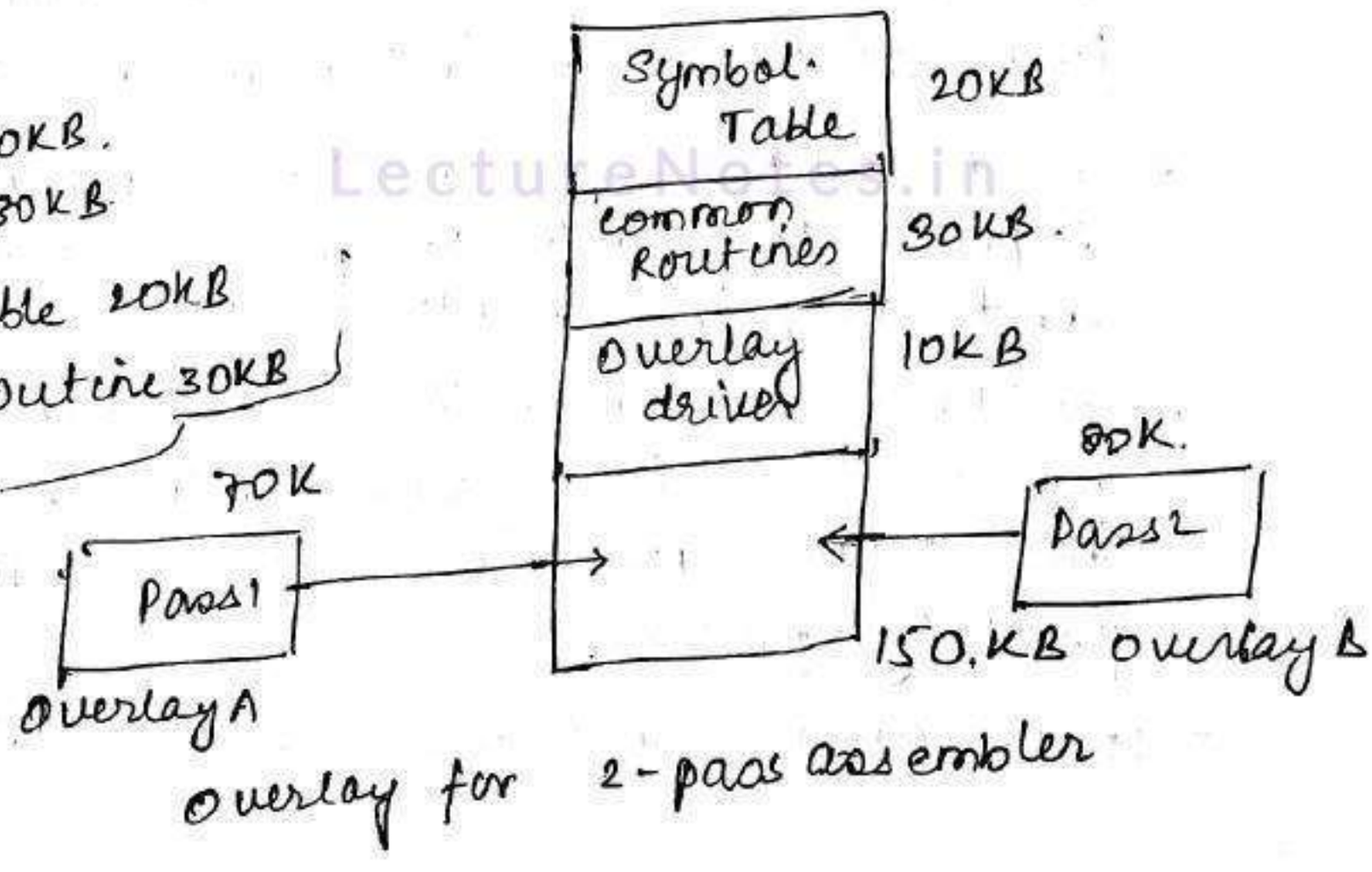
- Dynamic linking.
  - linking of library files before execution time is called static linking.
  - linking library files at the time of execution is called dynamic linking.

- Overlays.
  - To enable a process to be larger than the amount of memory allocated to it then we use overlays.
  - The meaning of overlays is to keep in memory only those instructions or data that are needed. and other ~~instructions~~ ~~loaded~~ after ~~the~~ ~~when~~ other
  - when other instructions are needed, they are loaded into m/m space occupied previously by the instructions. which are no longer needed.

12/10/17

{ Pass 1 70KB.
  Pass 2. 80KB

Symbol Table 20KB
Common Routine 30KB
Total : 200



overlay for 2-pass assembler

Scanned by CamScanner

To load everything at once we require 200 KB of memory but if we have only 150 KB available, we cannot run our process so we define two overlays

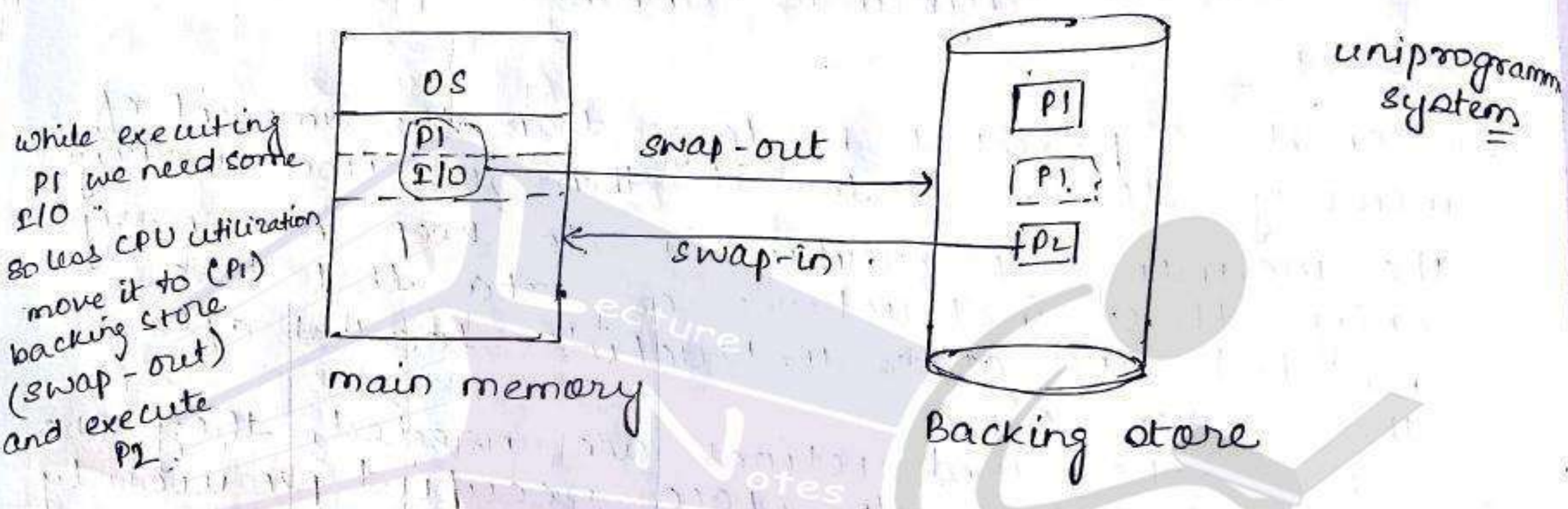
Overlay A contains system table, common routines and Pass1 similarly overlay B contains system table common routine and Pass 2.

size of overlay A is 20 + 30 + 70 = 120 KB

size of overlay B is 20 + 30 + 80 = 130 KB

{Disadvantage : slower due to division}.

- Swapping —



uniprogramming system

- swapping is a method to improve the main memory utilization
- switching a process from main memory to disc is called swap-out and switching a process from disc to main m/m is swap-in.
- In Round Robin algorithm when time quantum expires the m/m manager swap out one process and swap-in another process
- similarly in priority scheduling, o if high-priority process arrives the m/m manager can swap-out the lower priority process and swap in high priority process.
- It is also sometimes called roll-in-roll-out.

Scanned by CamScanner

- contiguous memory allocation:
  - Main memory must accomodate OIs and user-process
- Memory protection :> protecting the OIs from the user process and protecting one user procss from another user-process is called memory protection
  > we can provide protection using 2 registers : relocation register and limit register
  > relocation register is base register and
  > Limit register contains range of logical address.

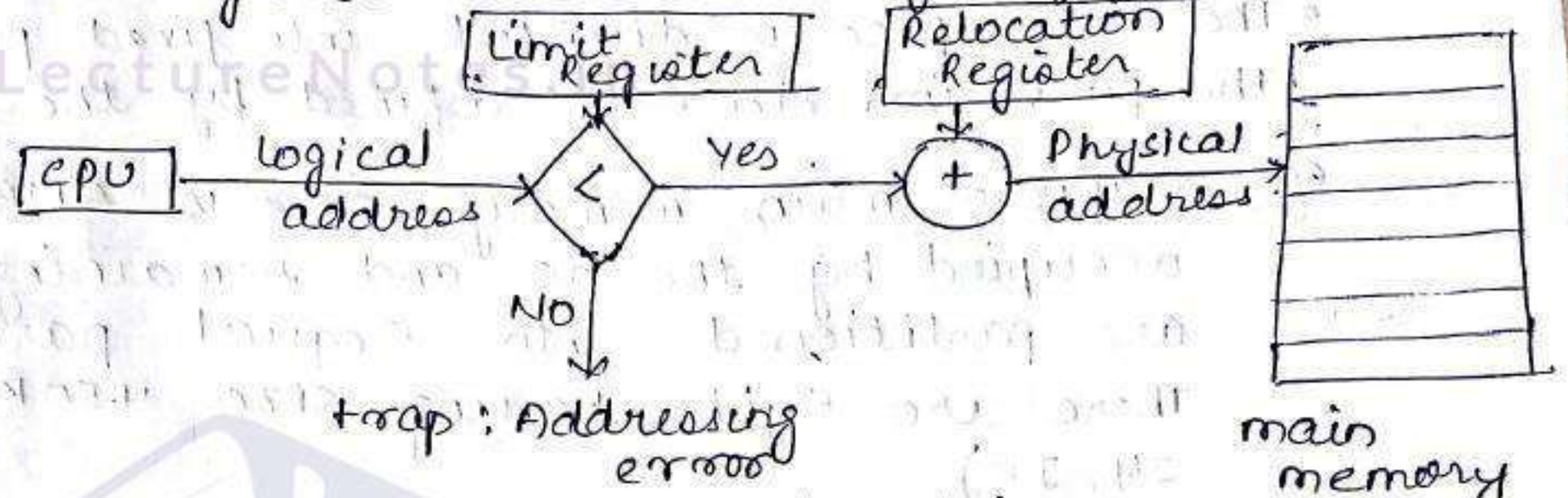
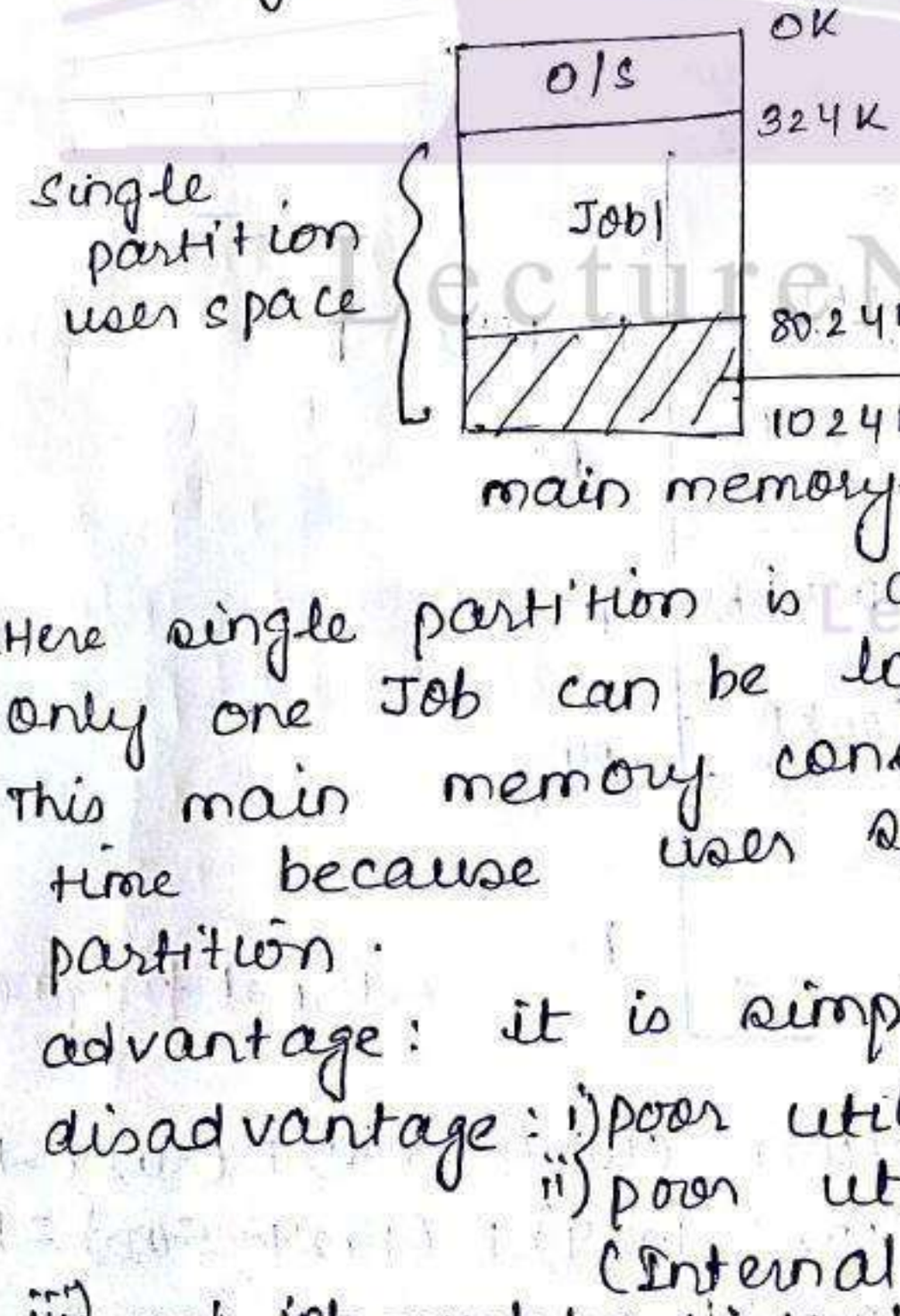


Fig: Hardware support for relocation and limit register

- memory allocation:
  > There are many methods for memory allocation.
  i) single partition allocation:



Here single partition is available for user space. only one Job can be loaded in this user space This main memory consist of only one processat a time because user space is treated as a single partition.
advantage: it is simple
disadvantage: i) poor utilization of processor.
ii) poor utilization of memory.
(Internal fragmentation - wastage of m/m)
iii) user's job must be limited to size of available main m/m

Scanned by CamScanner

ii.) multiple partitions allocation.

13/10/17

This method is implemented in 3 ways
a.) fixed equal multiple partitions
b.) fixed variable multiple partition
c.) dynamic multiple partition.

a.) fixed equal multiple partitions

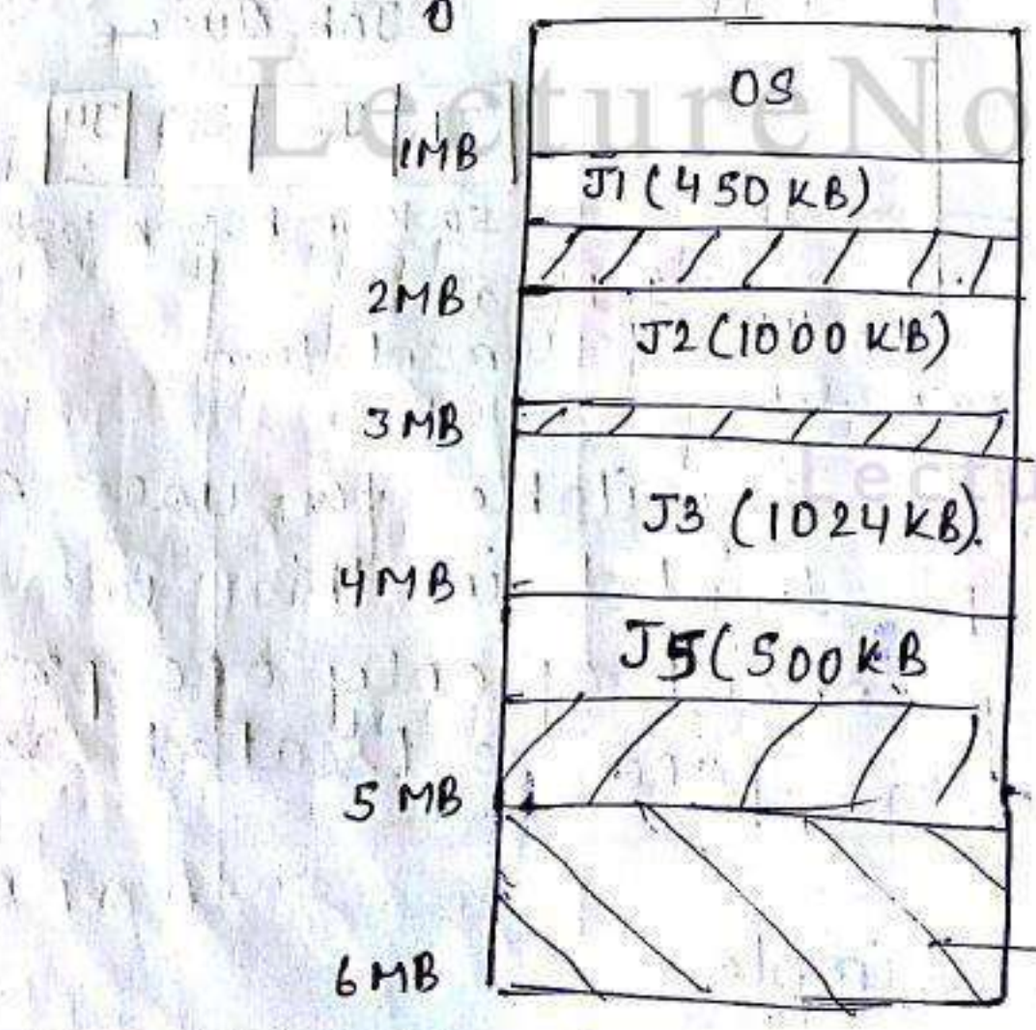
- The user space is divided into fixed partitions.
- The partition size are defined by the OS.

eg:- suppose main memory size is 6MB, 1MB occupied by the OS and remaining 5MB are partitioned into 5 equal partitions. There are 5 jobs ~~hatting~~ ~~size~~ ~~450KB~~ (J1, J2, J3, J4, J5)

Job Queue

J1	450 KB
J2	1000 KB
J3	1024 KB
J4	1500 KB
J5	500 KB

Find the internal and external fragmentation.



Total internal fragmentation (1024 - 450) + (1024 - 1000) + (1024 - 1024) + (1024 - 500) = 1122KB.
external fragmentation = 1024KB

Scanned by CamScanner

{ Total space is (1122+1024) > 1500 but still cannot be allocated as contiguous space is needed.

Advantage

i) supports multiprogramming.
ii) efficient utilization of CPU.
iii) simple and easy to implement

Disadvantage

i) supports internal and external fragmentation.
ii) The single free area may not be large enough for partition a single partition.

b) fixed variable multiple partition:

- In this technique the user space of main memory is divided into no. of partitions but the partition size are different length.

eg :-

Job Queue

Job	Size (kB)	Arrival Time
J1	825	10
J2	600	5
J3	1200	20
J4	450	30
J5	650	15

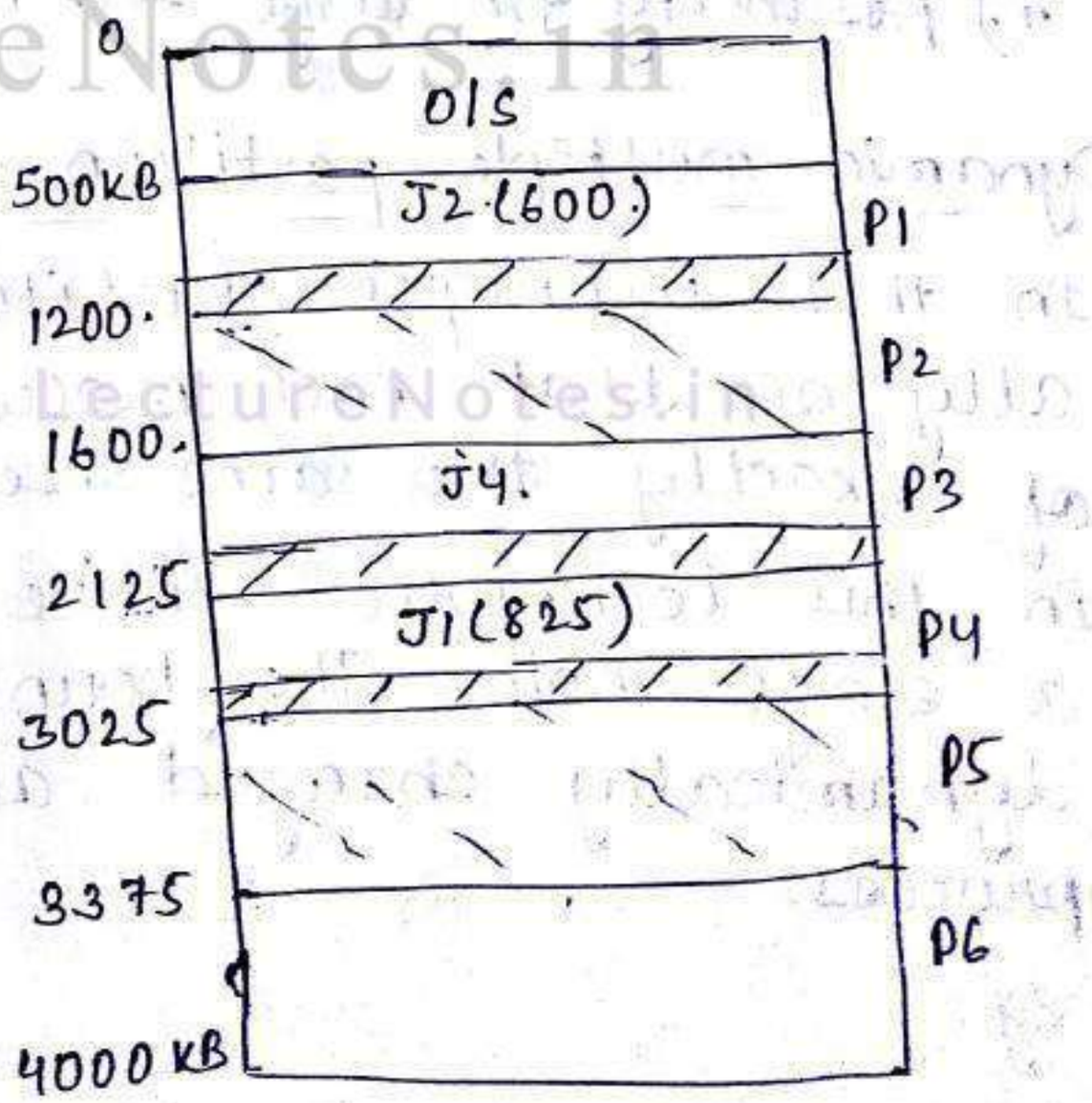
Partition	Size (KB)
P1	700
P2	400
P3	525
P4	900
P5	350
P6	625

Find internal and external fragmentation using this technique.

:- | J4 | J3 | J5 | J1 | J2 | → (x)

Job Queue

← According to arrival time



Total internal fragmentation
= (700 - 600) + (900 - 825)
+ (525 - 450)
= 250 KB

Total external fragmentation
= 400 + 350 + 625
= 1375 KB

Scanned by CamScanner

17/10/17

problem : The size of J2 is 600 KB, it is loaded in partition 1. so internal fragmentation is 100 KB. But if Job2 is loaded in partition 6 then internal fragmentation is 25 KB

To avoid this problem we use

i.) first-fit algorithm :
allocate the partition that is big enough starting from the first till the end. we can stop searching as soon as, we find a free hole (partition) that is large enough.

ii.) best-fit algorithm
allocate the smallest hole or partition that is big enough

iii.) worst-fit algorithm-
allocate the job to the largest hole.

advantage of fixed variable multiple partition.

i.) It supports multiprogramming.
ii.) efficient utilization of CPU and memory. (best & first fit)
iii) simple and easy to implement

disadvantage.

i.) This technique supports internal and external fragmentation.
ii.) possiblity for large external fragmentation (worst-fit)

c.) Dynamic multiple partition.

- In this technique partitions are created dynamically so that each process is loaded into partition of exactly the same size as that of process size.
- In this technique entire user space is treated as a single hole. The boundaries of partition are dynamically changed and depend on size of the process.

eg:

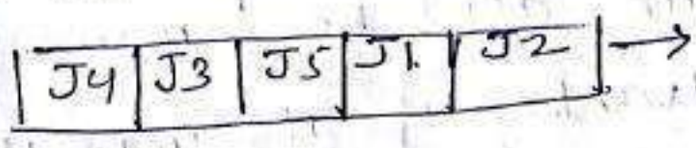
Scanned by CamScanner

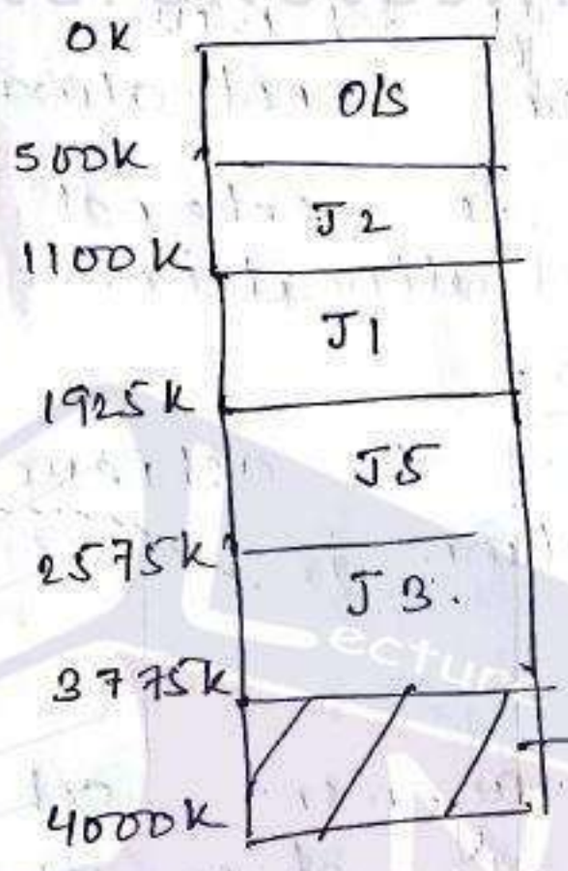
19/10/17

Job Queued

Job	size	Arrival Time
J1	825	10
J2	600	5
J3	1200	20
J4	450	30
J5	650	15

Job Queue

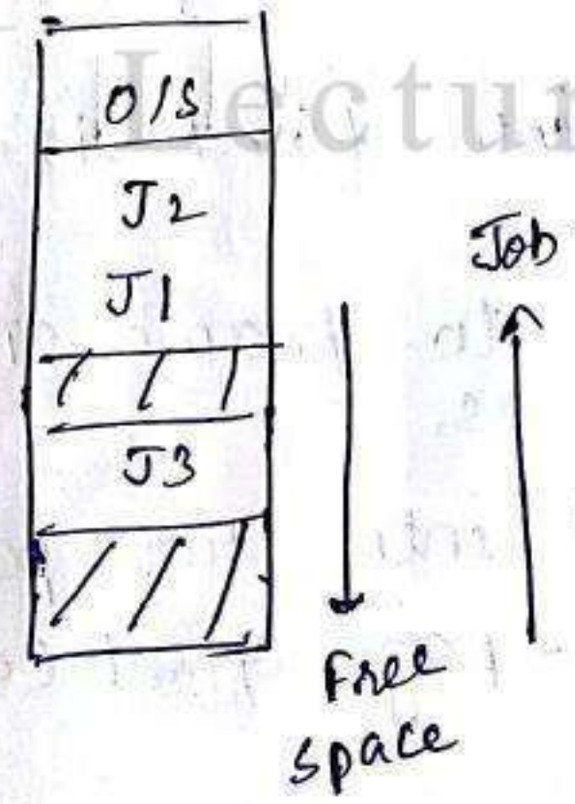


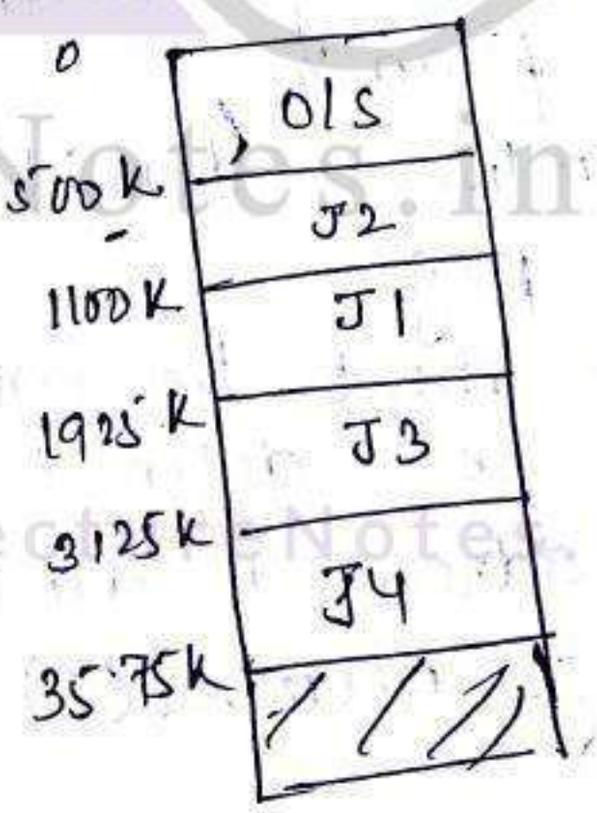


After all job allocation, only 225KB of memory space is available so we cannot allocate J4 into it.

Fragmentation = 225K

Suppose after sometime J5 completes its operation and release its memory. so total free space = 225+650 = 875K





Advantage:

- It does not supports internal fragmentation because partitions are dynamically changed.
- efficient utilization of mm and CPU.

Scanned by CamScanner

Disadvantage

- It supports external fragmentation.

→ compaction.

solution to the problem of external fragmentation is called compaction. The goal of compaction is to place all free memory space in one large block

advantage: It improves memory utilization
it improves performance

disadvantage: It is very expensive
compaction is not always possible.

Note*: Another solution to external fragmentation is non-contiguous allocation.

→ Non-Contiguous allocation:-

There are two techniques to achieve this
i) paging ii) segmentation.

{ address of ram - Physical
" " process - Logical

- Paging

Paging is a mm management scheme that permits physical address space of a process to be non-contiguous

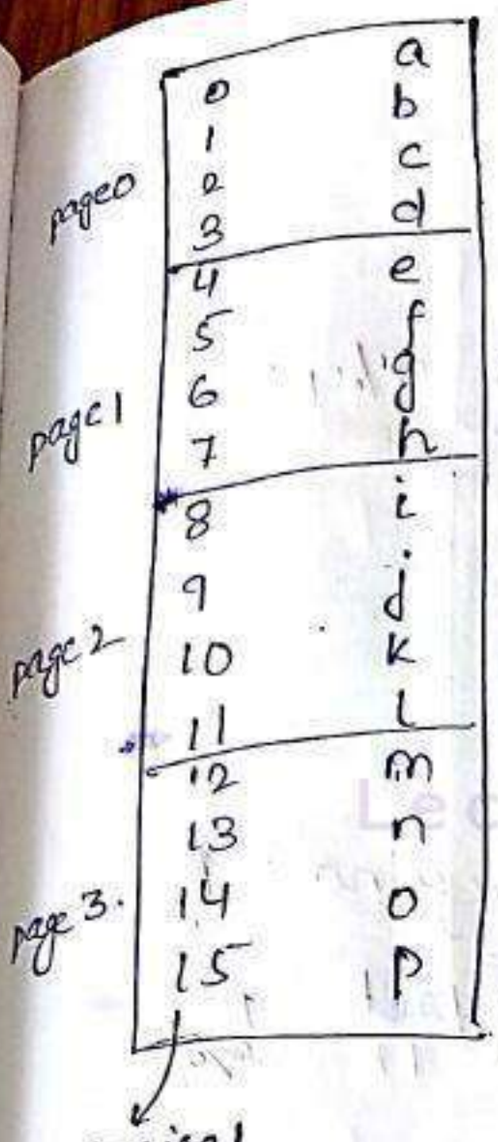
Physical memory is divided into fixed size blocks called frames.

Logical m/m is divided into fixed size blocks called pages.

Page size must be equal to frame size which should be defined by OS.

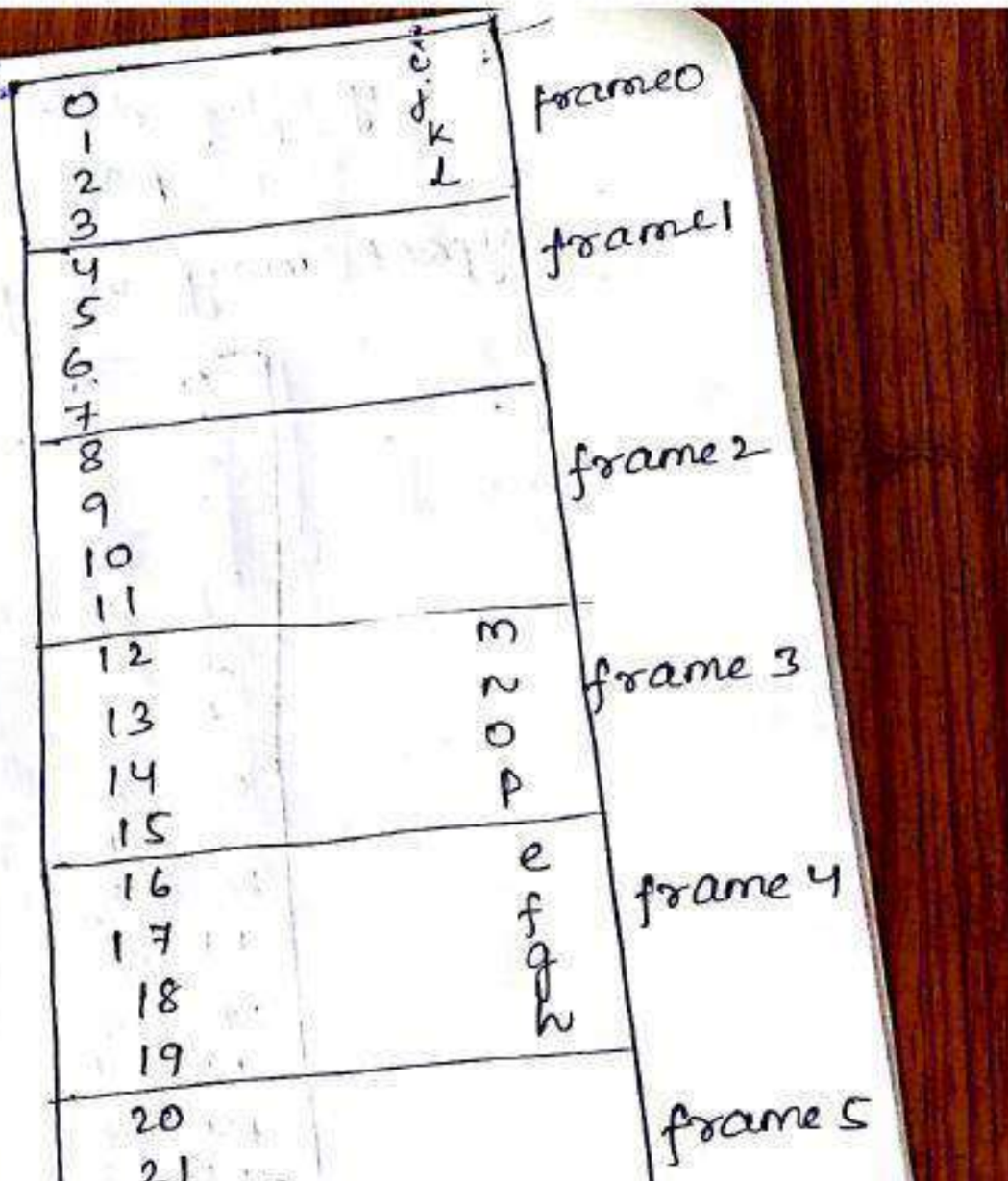
Logical address is divided into two parts
- page number(P)
- page offset(d)

Scanned by CamScanner



logical address

logical memory.



Physical address

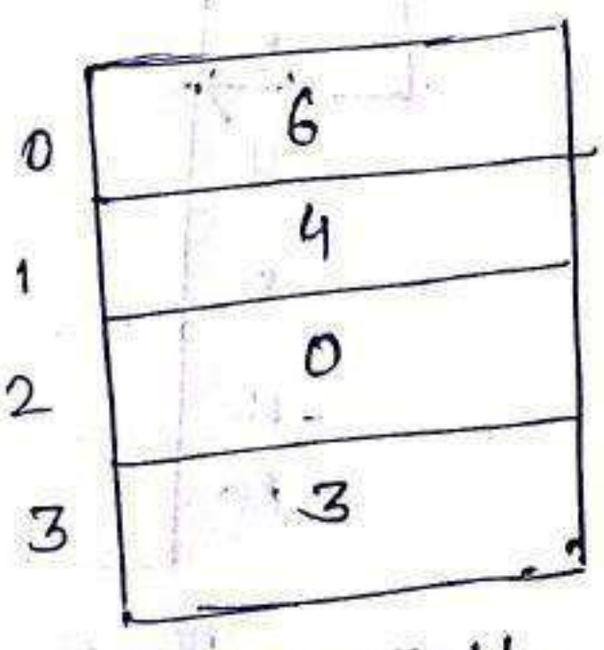
Physical memory

Page no. = $\frac{\text{logical address}}{\text{page size}}$

offset = logical address % Page size

Page size.

→ Page 0 stored in frame 6.



Page Table

- Page number is used as an index into the page table
- Page table contains the base address of each page of the physical m

LectureNotes.in
Scanned by CamScanner

if log. add = 9

page no. = $\frac{9}{4} = 2$.

offset may vary from 0 to 3

0	0	a
1	1	b
2	2	c
3	3	d
4	0	e
5	1	f
6	2	g
7	3	h
8	0	i
9	1	j
10	2	k
11	3	l
12	0	m
13	1	n
14	2	o
15	3	p

offset = 9%4 = 1

eg. 15

page no = $\frac{15}{4} = 3$

offset $\frac{15}{4}$ = 3

→ H

PA = frame no. × page size + offset

PA = 0 × 4 + 1 = 1

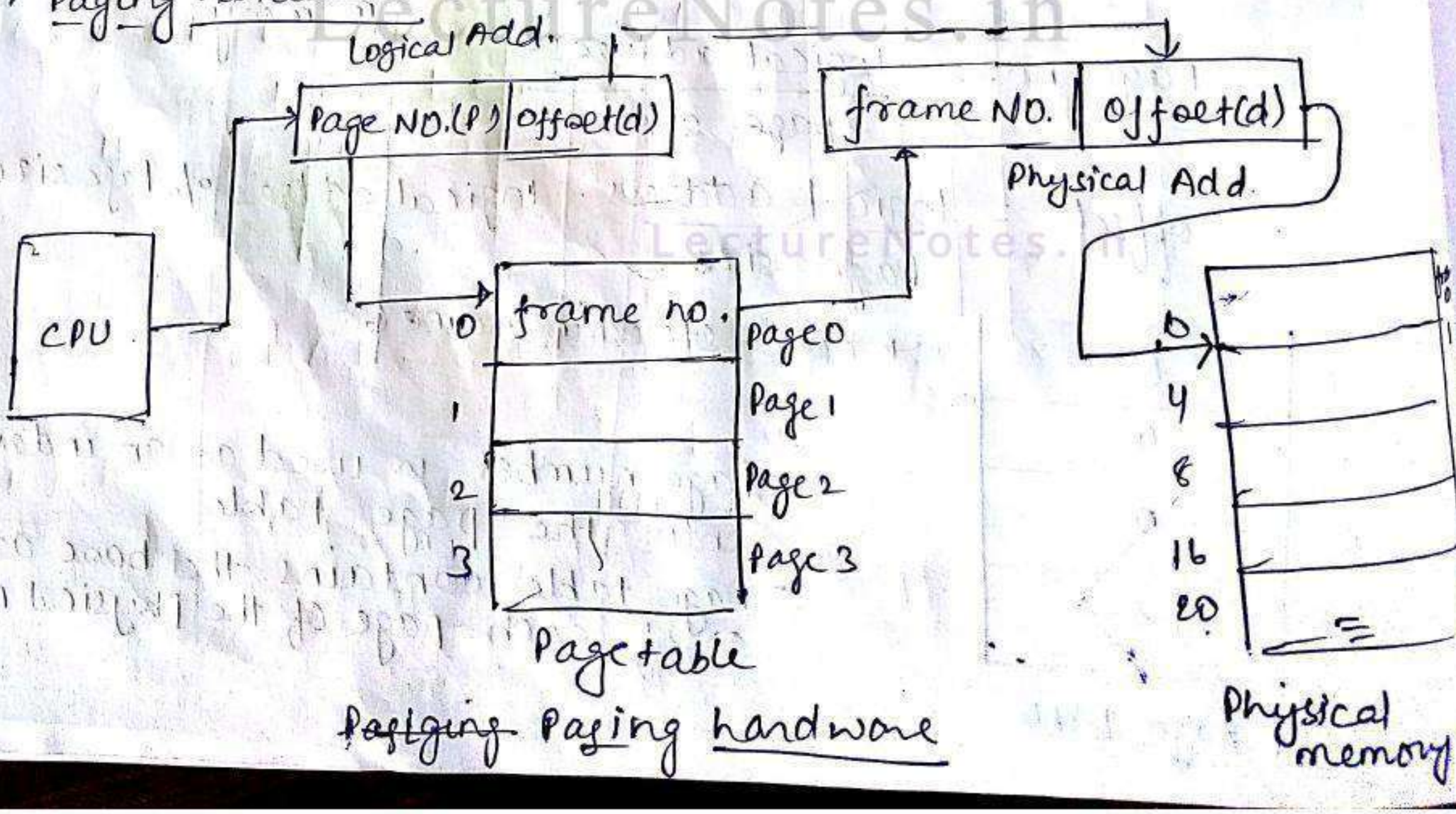
13 PA = 3 ~~× 4 + 1 = 13~~

3 × 4 + 13 % 4

= 12 + 1 = 13

24|10|17

→ Paging hardware :-



Paging hardware

Scanned by CamScanner

Advantage of paging:

i) no external fragmentation but we may have some internal fragmentation.
ii) It supports time sharing system.
iii) It supports virtual memory.
iv) sharing of common code is possible
v) To avoid internal fragmentation small page size are desirable.

Disadvantage.

i) It supports some amount of internal fragmentation.
ii) If the no. of pages are high it is difficult to maintain the page table.

→ Hardware support of page table :-

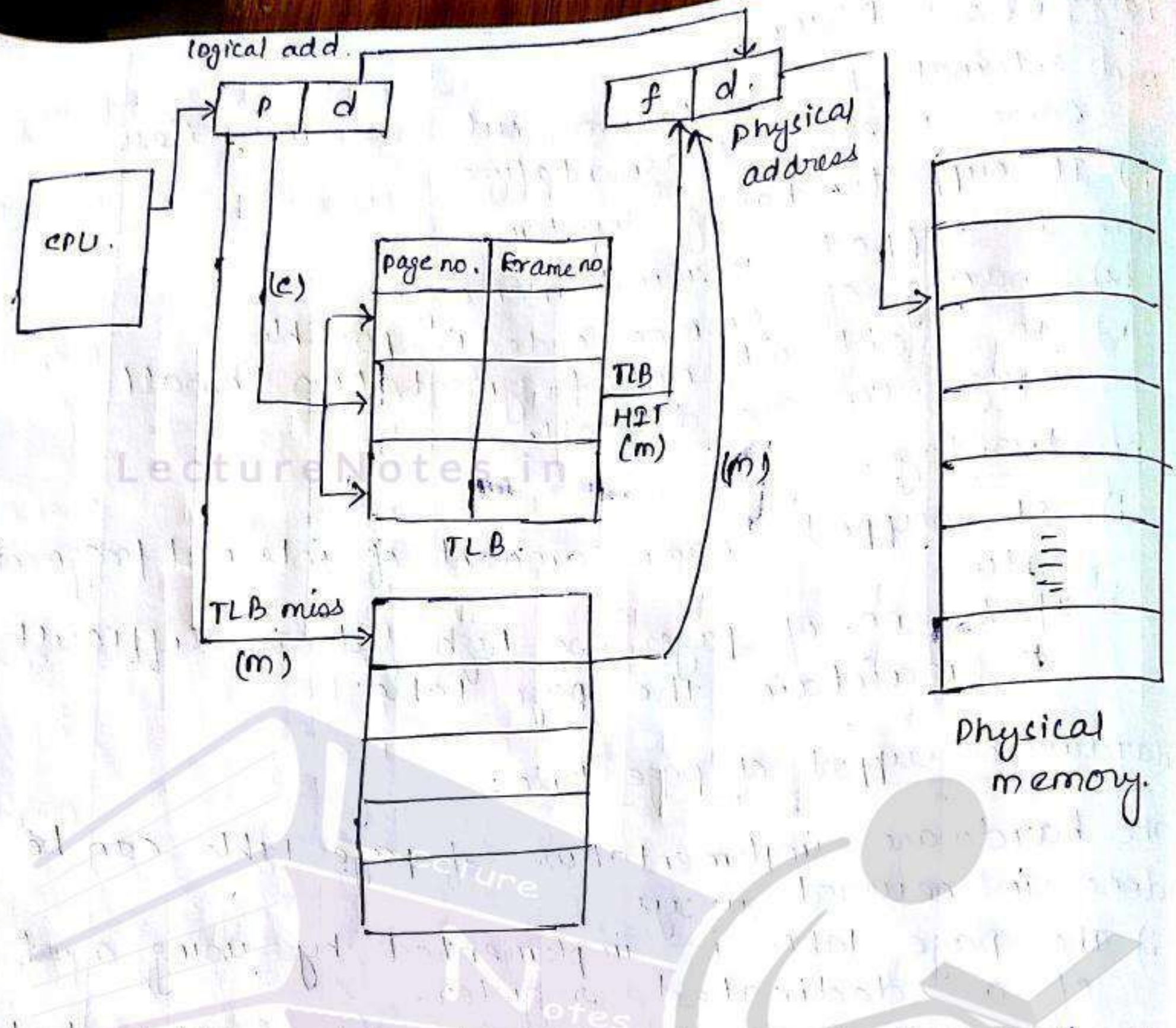
The hardware implementation of page table can be done in several ways

i) The page table is implemented by using a set of a dedicated register.
advantage : accessing the page table will be faster.
disadvantage: design of large page table using registers are not feasible.

ii) keep the page table in main memory.
advantage: we can create a large size page table.
disadvantage: It takes more time to access the memory location (by a factor of 2).

iii) The solution to this problem is translation look aside buffer (TLB) : It is a special, small, fast hot cache hardware called TLB.
This TLB is a associative memory or high speed memory. (content adressible mim (CAM) principle)
eg: when you search some part of name of a file and file comes)

Scanned by CamScanner



let h be the TLB hit ratio, c is the cache access time and m is memory access time then effective memory access time =

$$\boxed{EMAT = h*(c+m) + (1-h)*(c+2M)}$$

Q: A TLB having 80% hit ratio 20 nanosec to search TLB, 100 ns to access the memory. Find the effective memory access time.

$h = 0.8 \qquad c = 20\ ns \qquad m = 100\ ns$

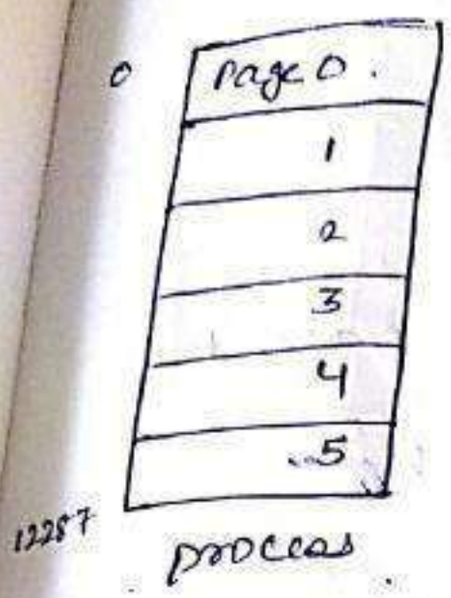
$EMAT = 0.8 \times (120) + 0.2 \times (20 + 200)$

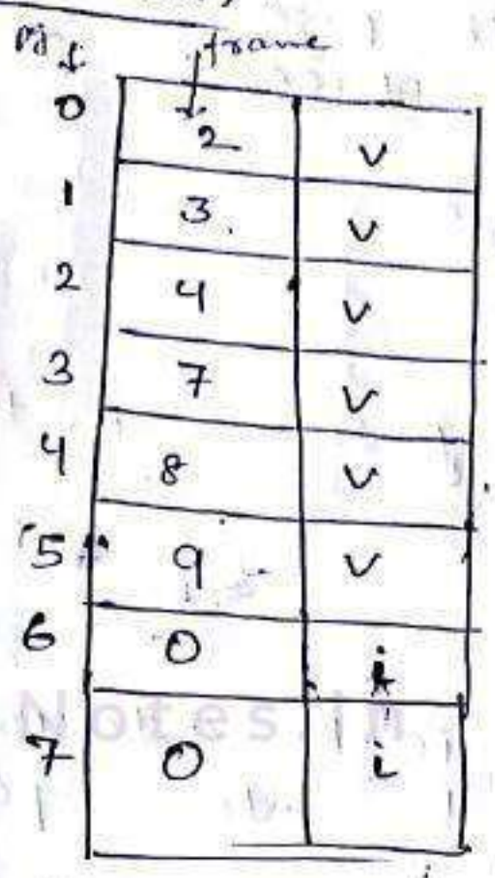
$= 96 + 44$

$= 140\ ns$

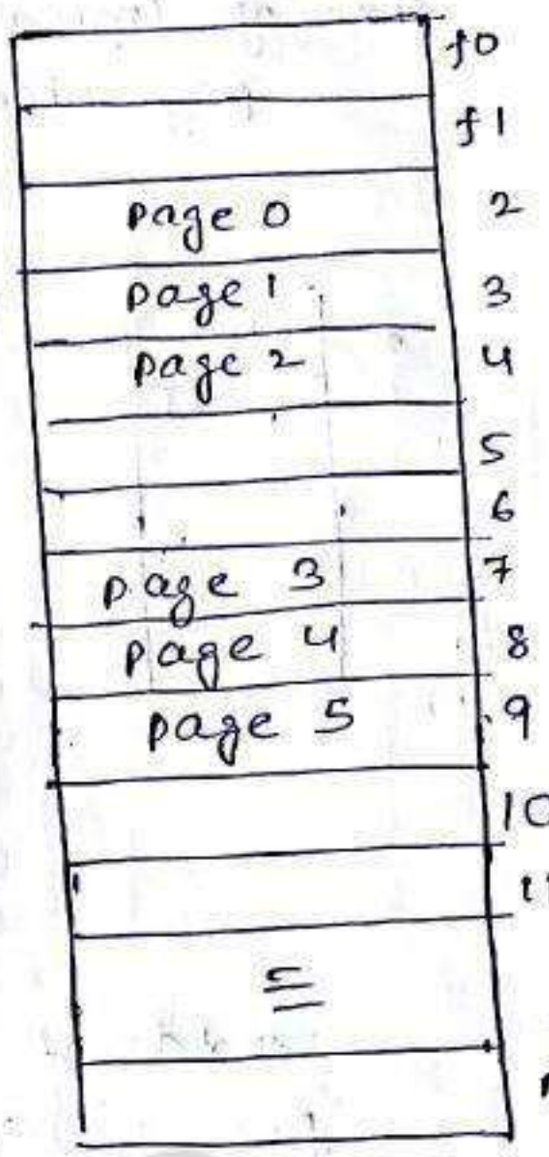
Scanned by CamScanner

→ memory protection.





page table.

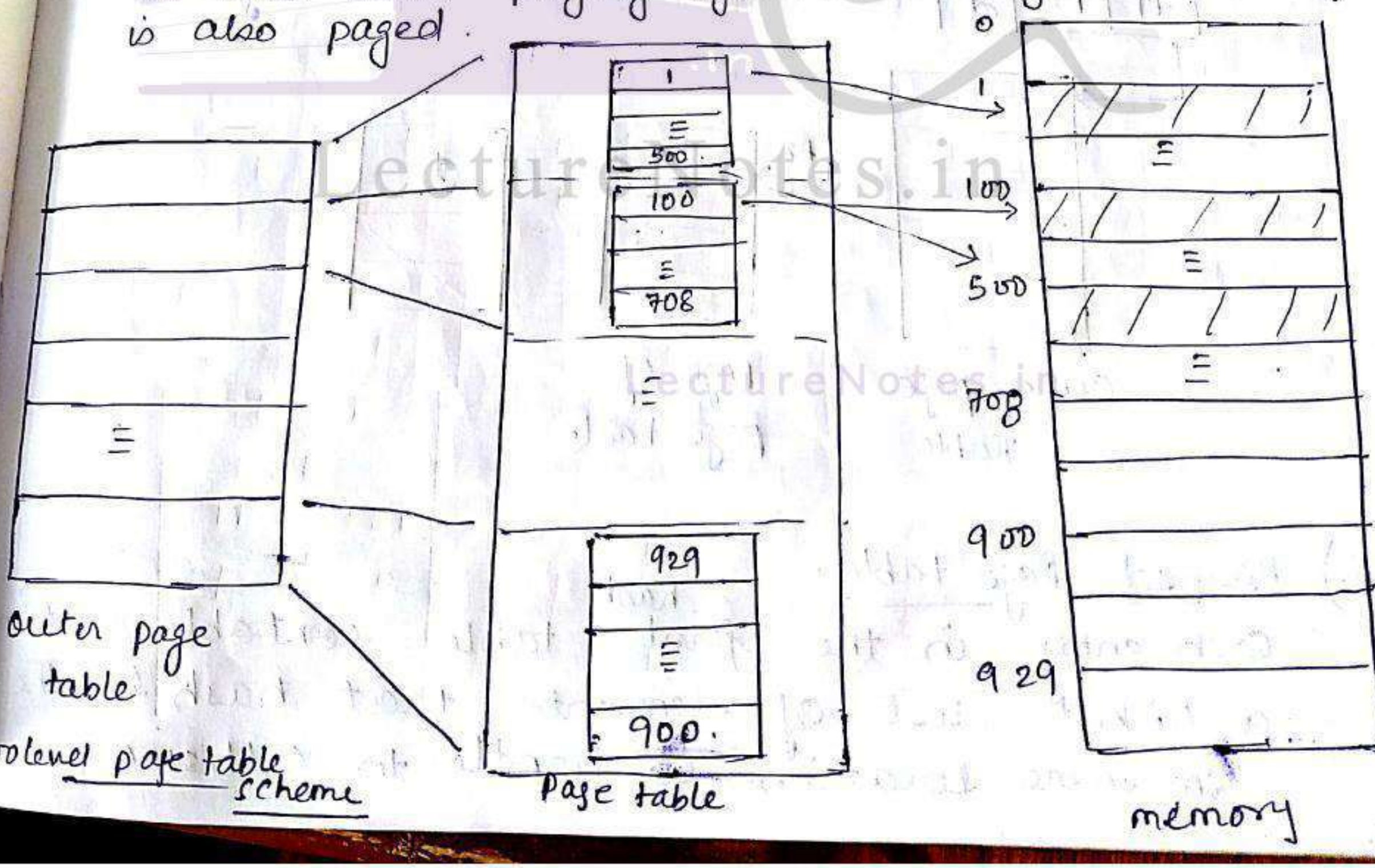


physical memory.

→ structure of the page table :-

i) Hierarchical paging :-

In two level paging system, the page table itself is also paged.

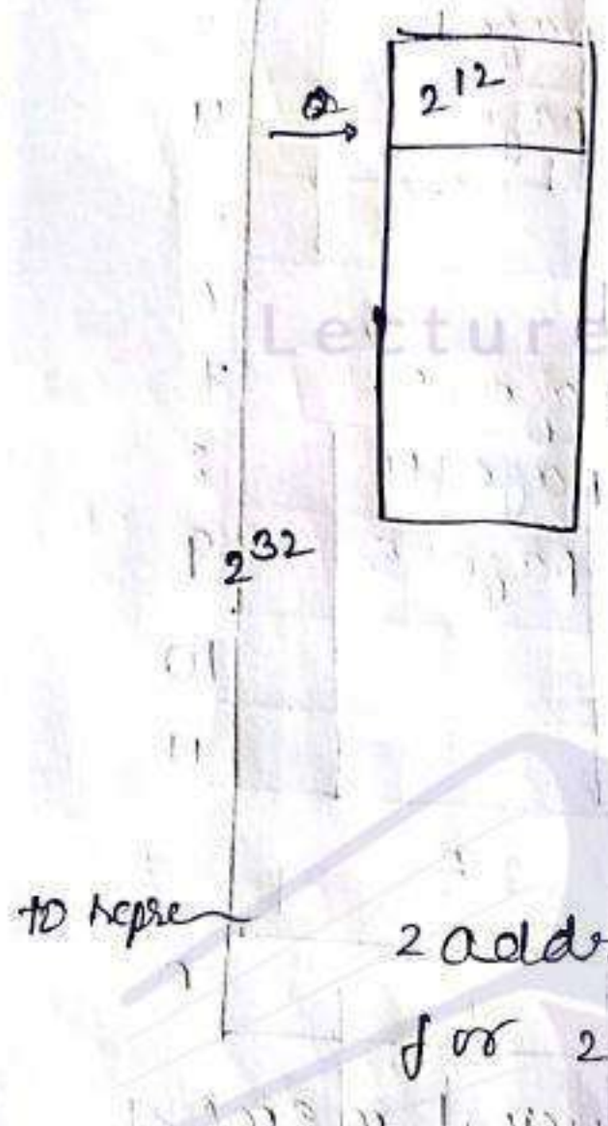


Two level page table scheme

Scanned by CamScanner

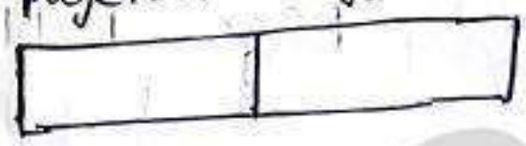
eg:- A 32 bit machine with page size of 4 KB.
size of logical address space is 2^{32}

page size = 4 KB
$= 2^2 \times 2^{10}$
$= 2^{12}$

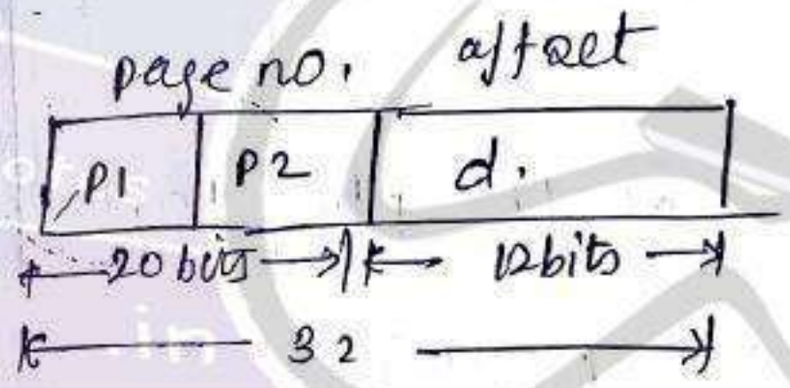


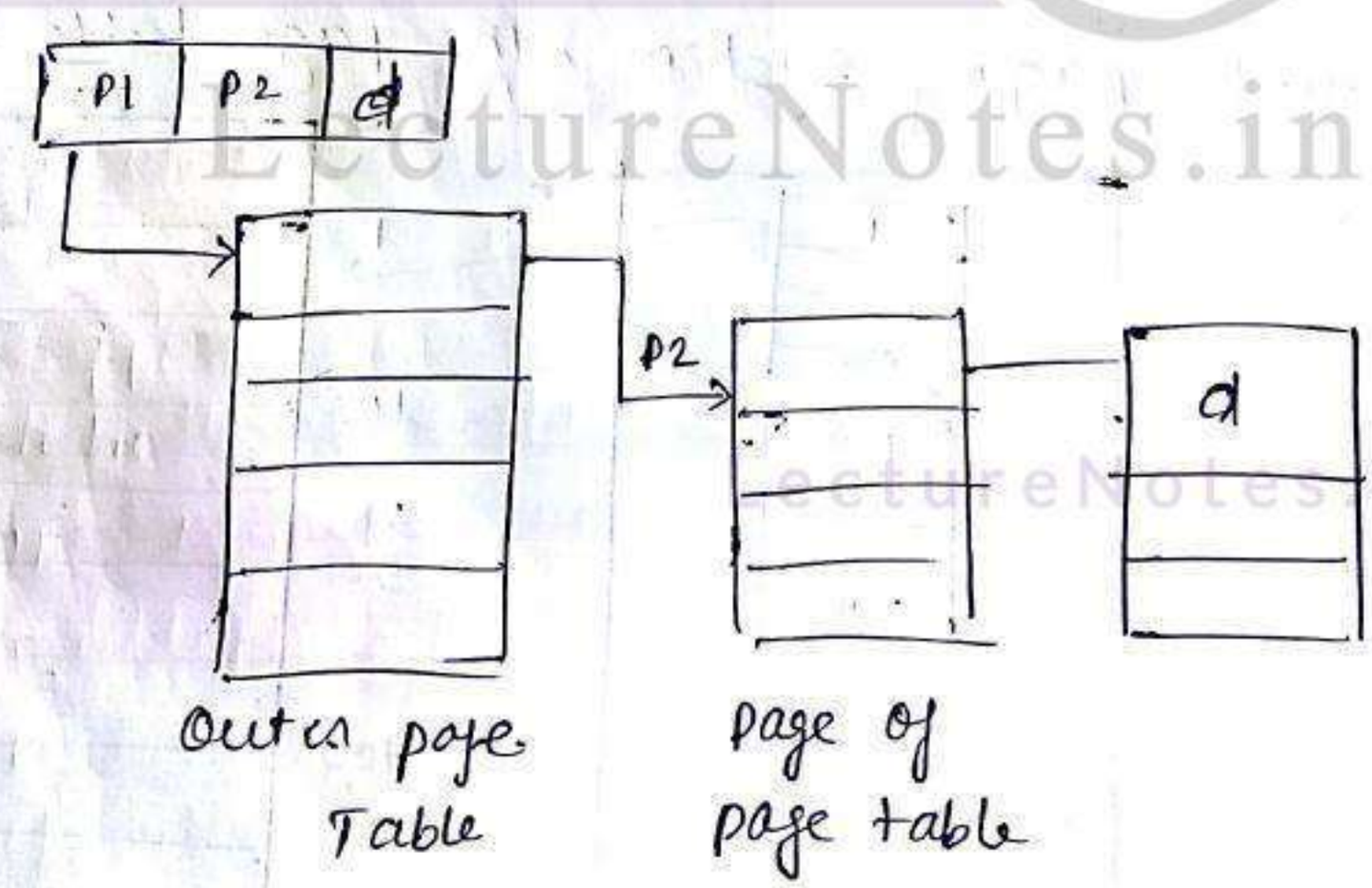
No. of entries in the page table
$= \frac{2^{32}}{2^{12}} = 2^{20}$

This logical address can be divided into 2 parts
offset



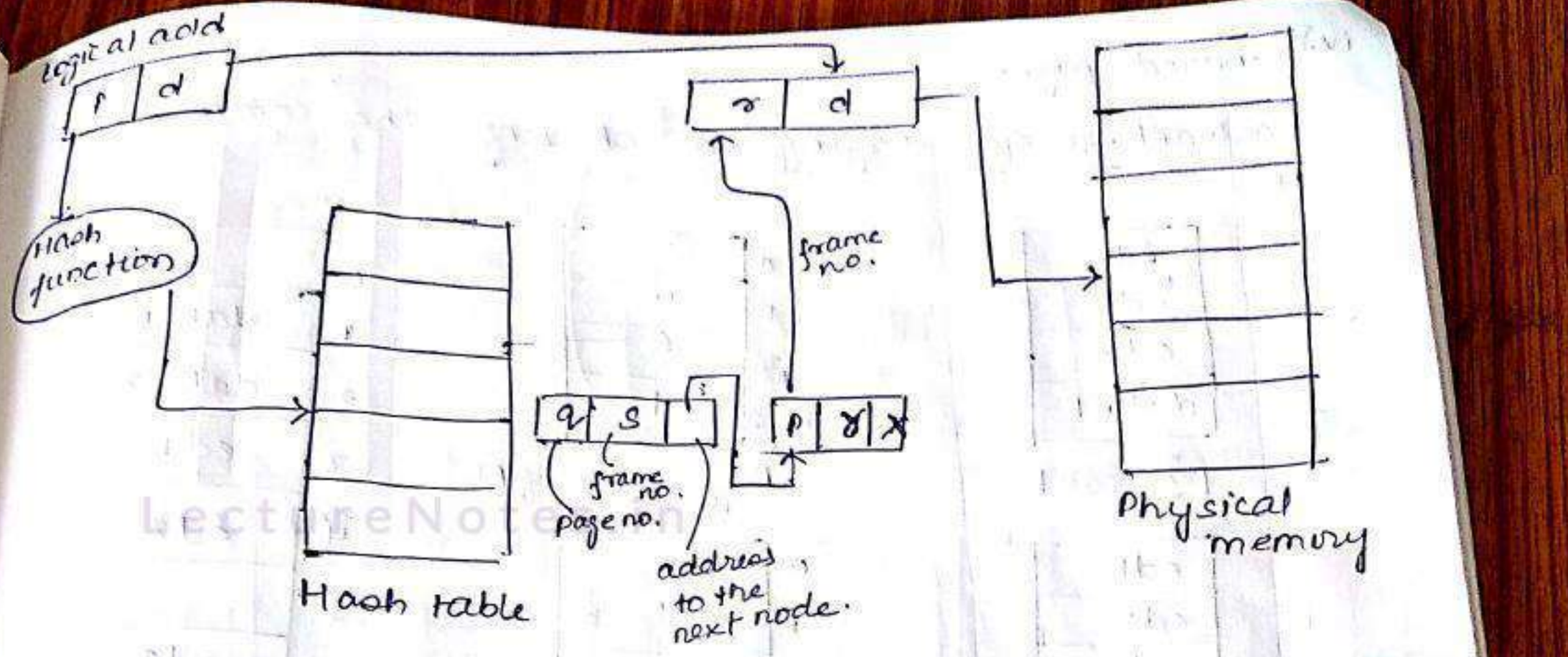
to represent 2 address — 1 bit
for 2^{12} add → 12 bits reqd.





ii.) Hashed Page table.
Each entry in the ~~first~~ hash table ~~contable~~ contains a linked list of elements that hash to the same location to handle to collision

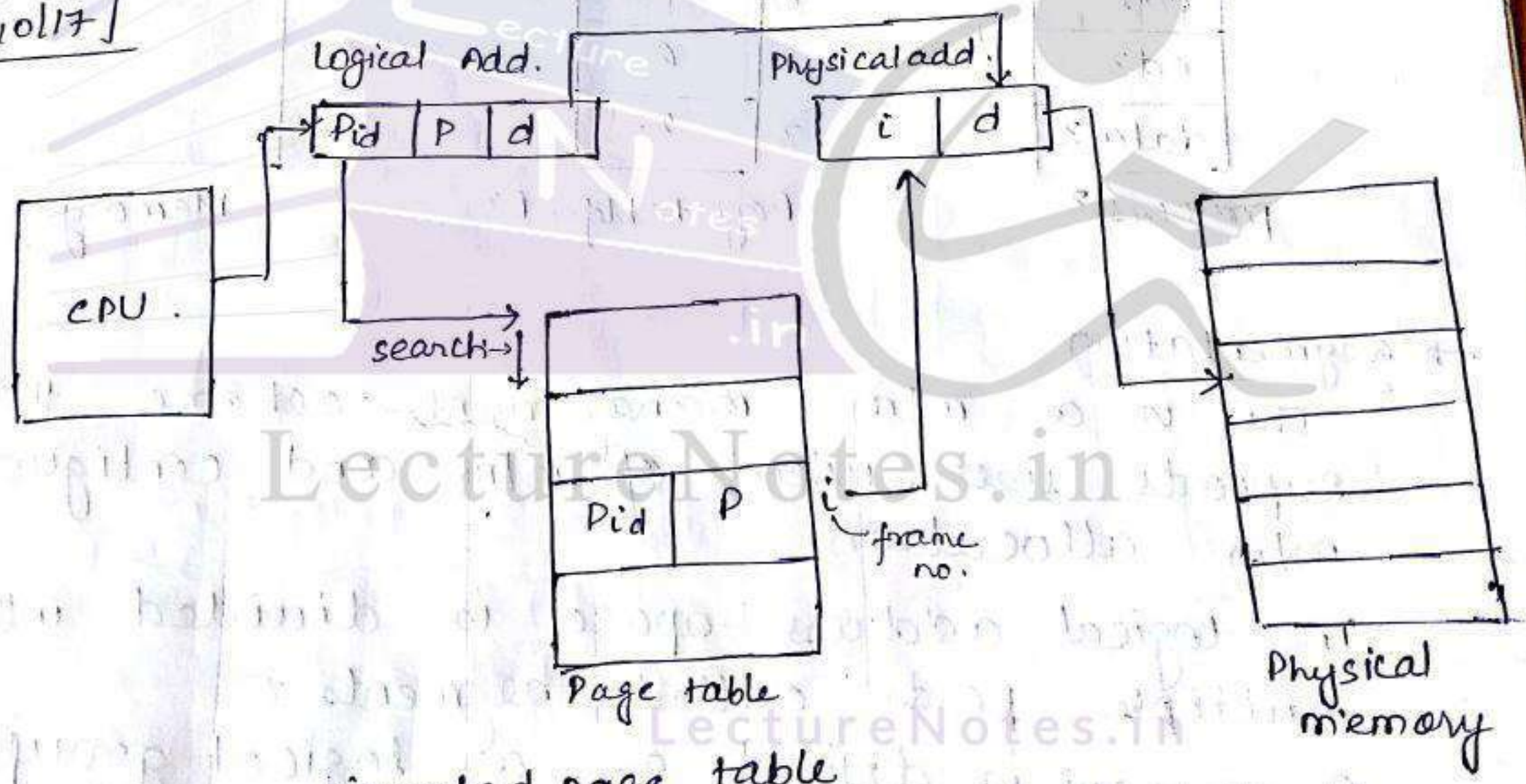
Scanned by CamScanner



iii) Inverted page table.

each process has a page table associated with it. An inverted page table has one entry for each page or frame

25/10/17



Inverted page table

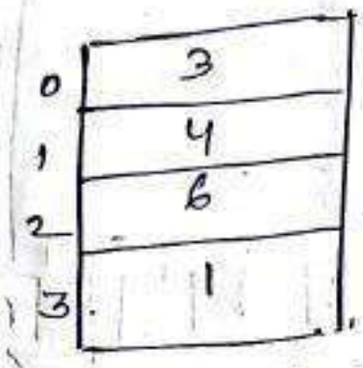
In this case only 1 page table is available in the system. Each virtual address consist of a triplet <Processid, pageno., offset>. In each inverted page table entry contains <process id, page no.>

Scanned by CamScanner

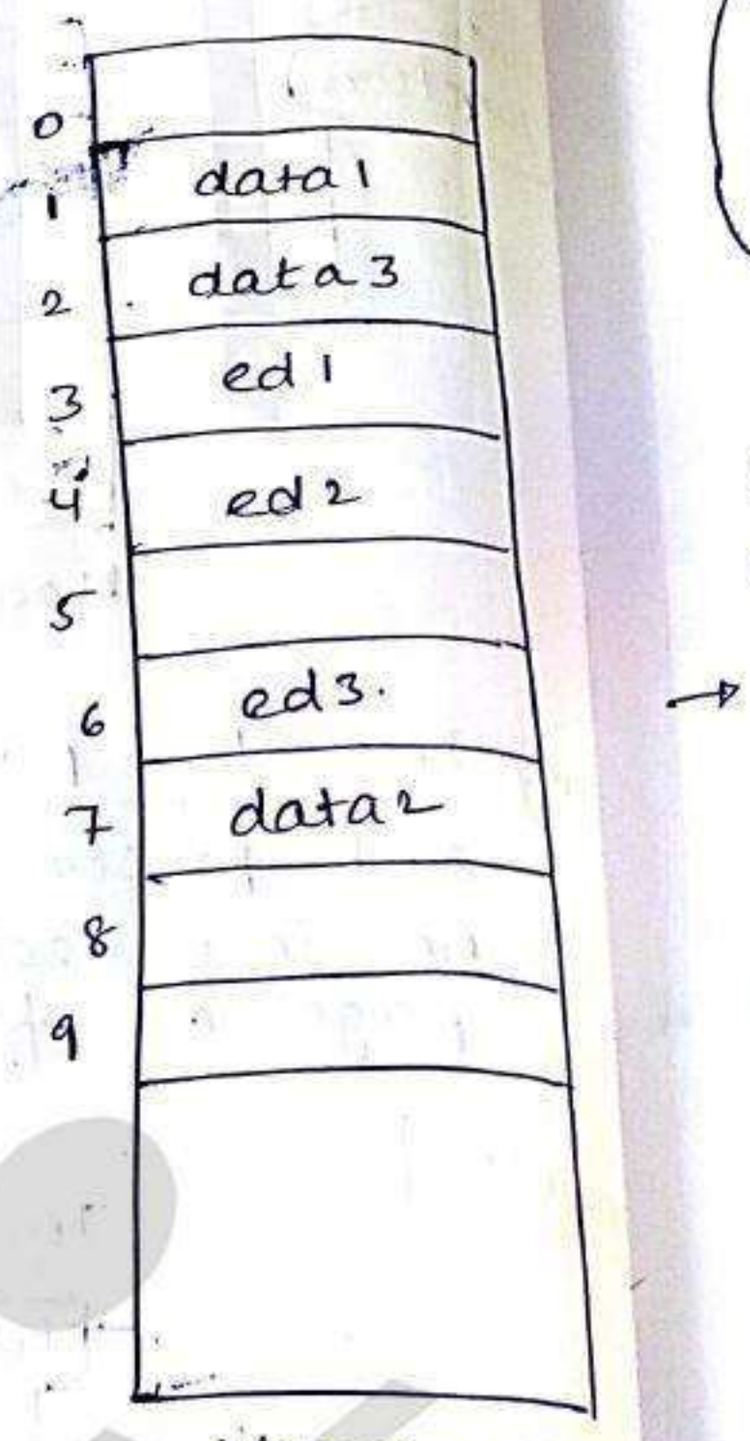
iv) shared pages.

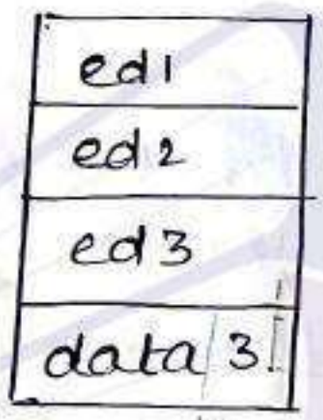
advantage of paging is sharing the common code.

process 1



Page table of P1.





process 2

page table for P2

process 3.

Page table for P3

Memory.

→ Segmentation

This is a m/m management scheme that supports user view of m/m and contiguous m/m allocation.

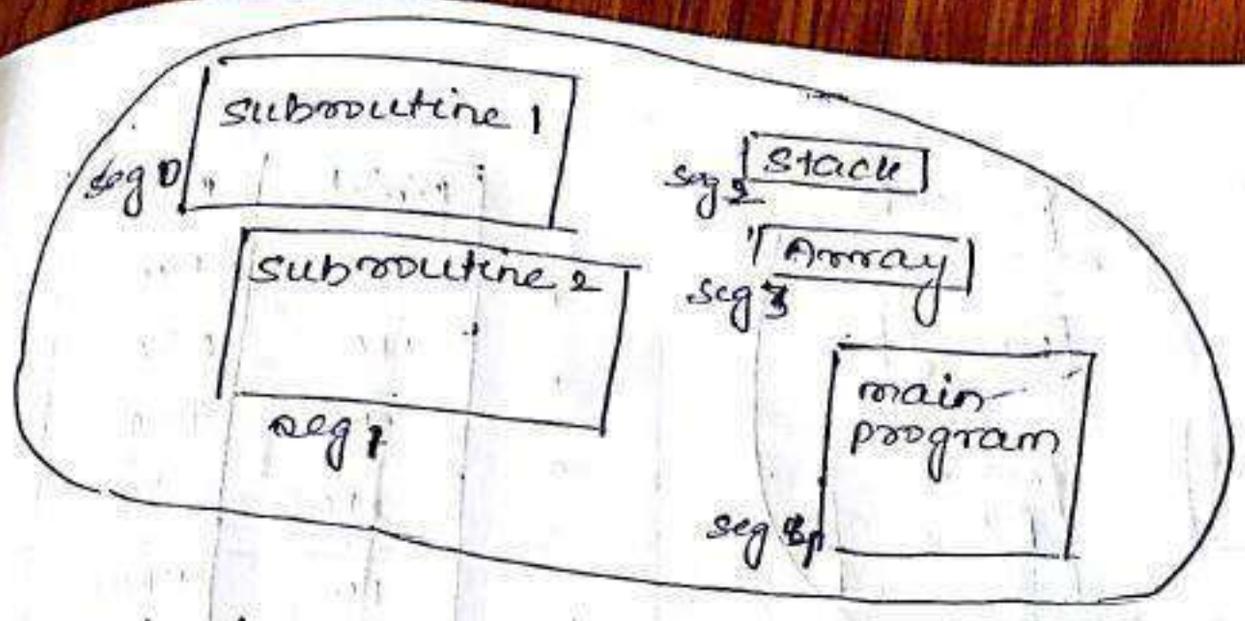
The logical address space is divided into multiple parts called segments.

A segment is defined as a logical grouping of instructions such as ~~stubg~~ subroutine, data area, stack, array, etc.

Each segment has a name and a length.

Every program is a collection of segments segmentation is a technique of managing these segments.

Scanned by CamScanner



A logical address consist of two tuples <segment no., offset>!

→ segmentation hardware

A segment table is used to map two-dimensional user defined address into one-dimensional physical address.

each entry in the segment table has segment base and segment limit. Index of the segment table is the segment number.

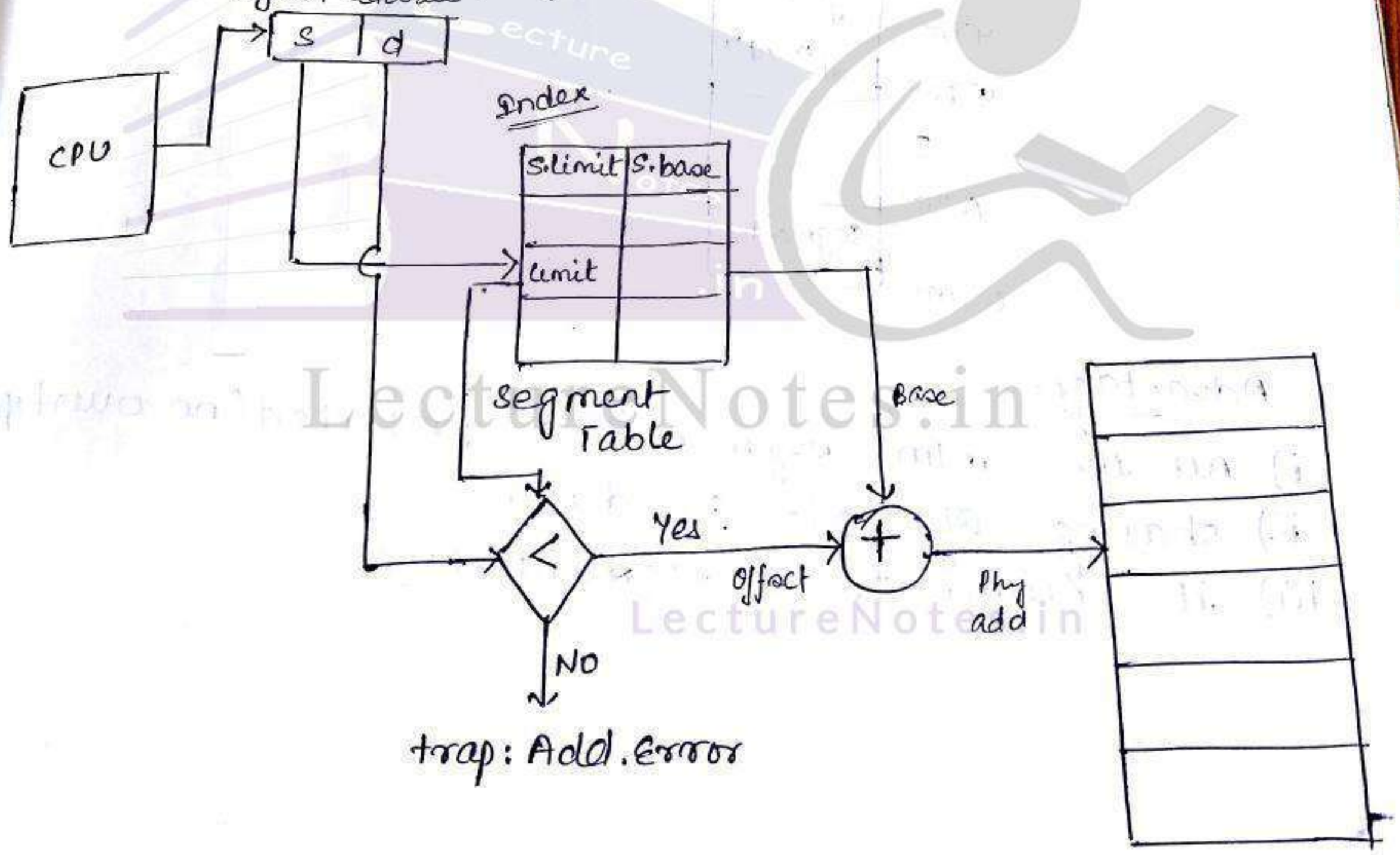
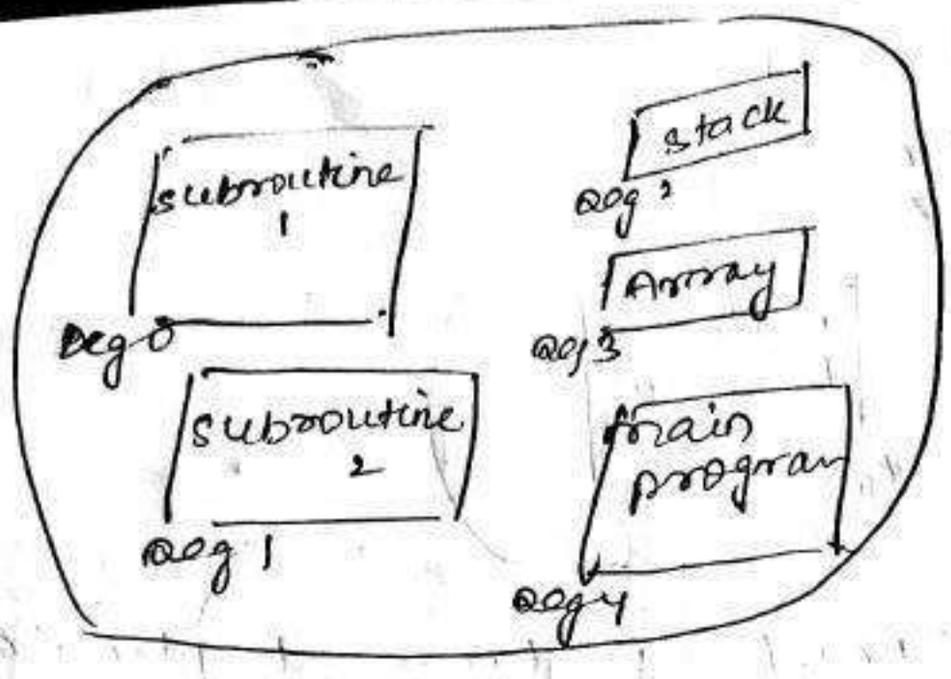


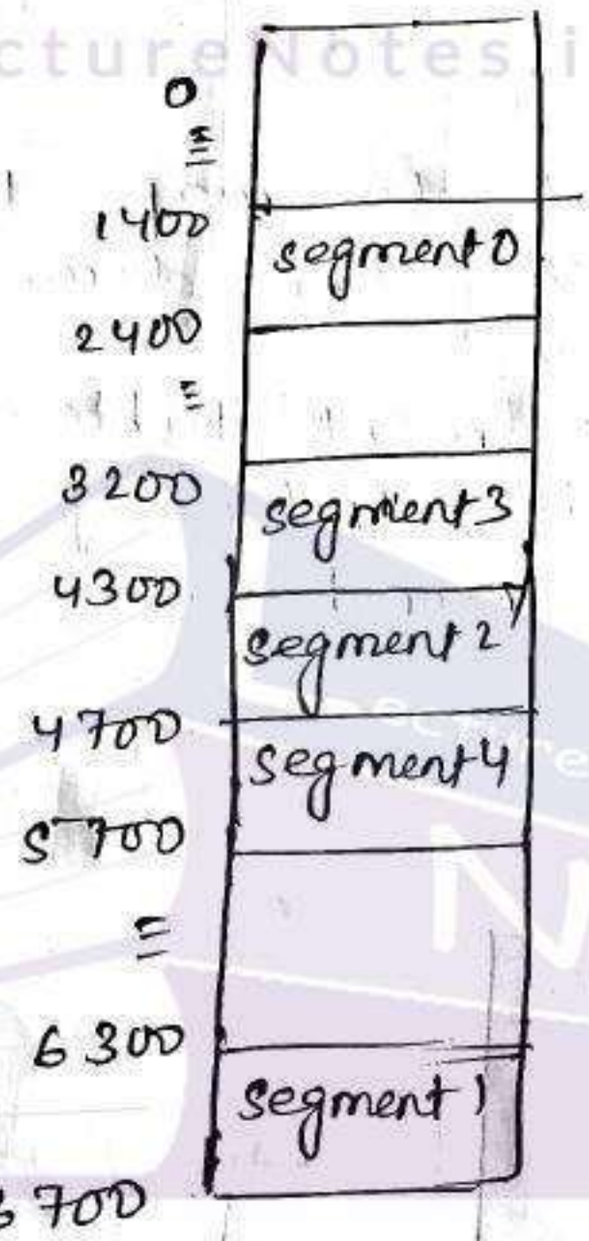
Fig: segmentation hardware

Scanned by CamScanner



	Limit	Base
0	1000	1400
1	400	6300
2	400	4300
3	1100	3200
4	1000	4700

Segment table.



{ 3500 valid address
{ 2500 invalid address

Advantages.

i.) All the m/m segments are protected (no overlapping)
ii.) sharing of code or data.
iii.) it eliminate fragmentation.

LectureNotes.in

Scanned by CamScanner

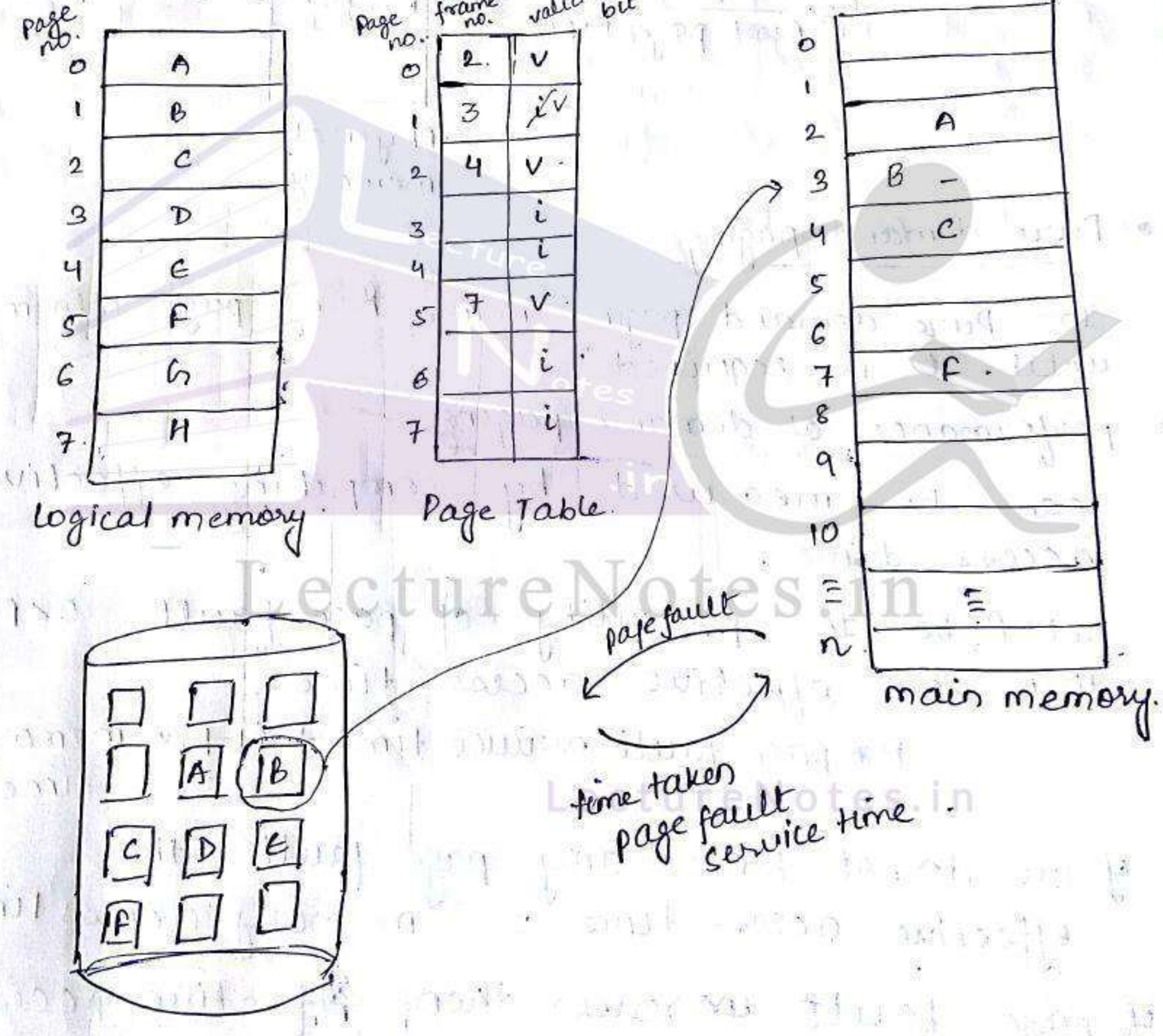
26/10/12

VIRTUAL MEMORY.

virtual m/m is implemented by using demand paging. It is similar to paging system with swapping. In demand paging a page is not loaded into the main m/m from the secondary m/m until it is needed. So a page is loaded into the main memory by demand. This technique is called demand paging. Here a lazy swapper is used to swap the pages b/w main m/m and secondary m/m.

A lazy swapper never swaps a page into the memory unless that page will be needed.

- Demand Paging hardware :-



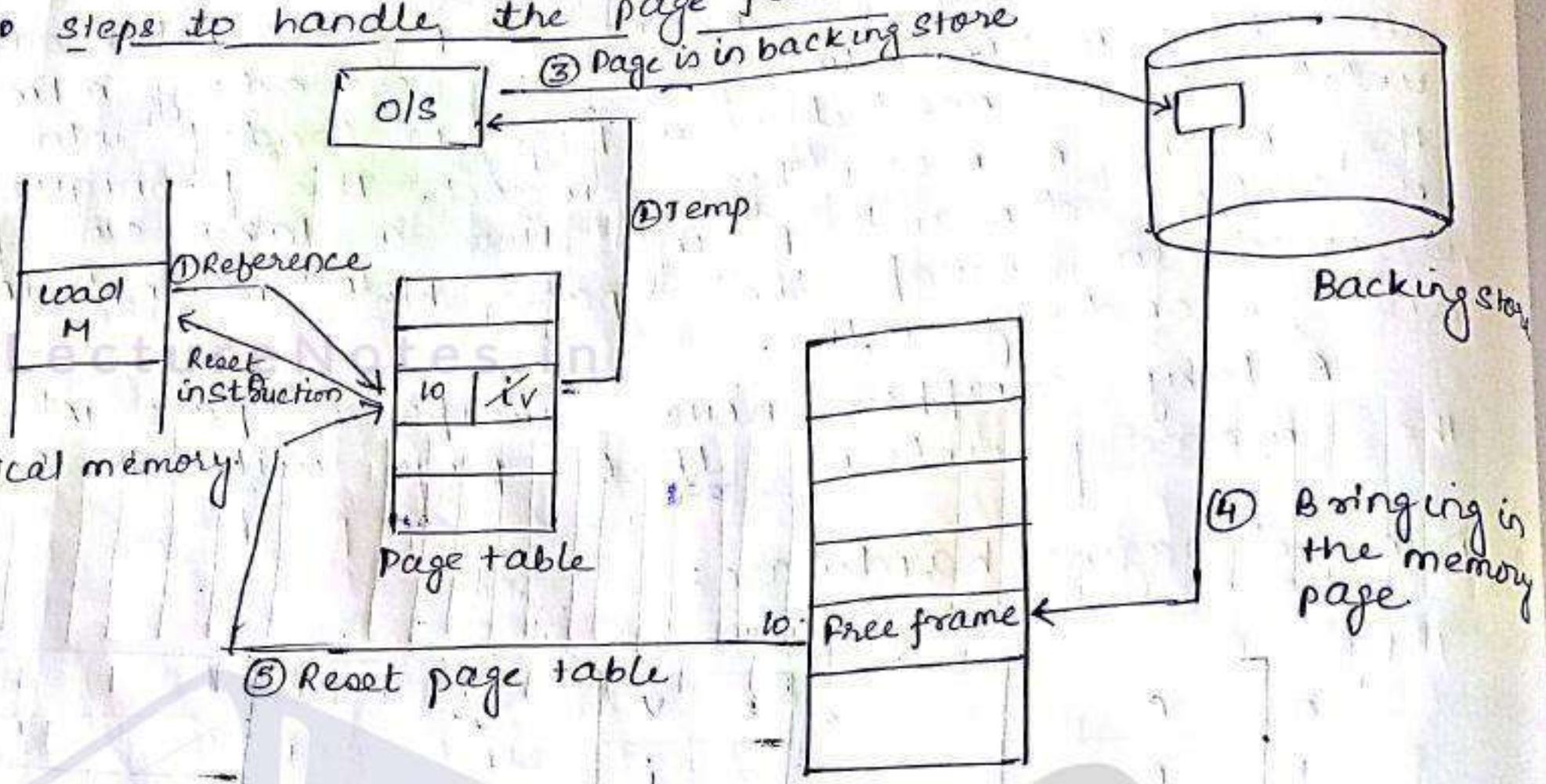
- Page Fault.

when the process needs to execute a particular page, that page is not available in memory this

Scanned by CamScanner

situation is called page fault. It means the OS fails to bring the required / desired page into memory.

- Steps to handle the page fault



- Pure demand paging

In Pure demand paging never bring page into memory until it is required

- performance of demand paging

can be measured by computing effective access time.

let P be the probablity of page fault $0 \le P \le 1$

then, the effective access time =

P * page fault service time + (1-P) * memory access time

if we do not have any page fault the effective access time = memory access time (i.e. P=0)

if page fault increases then effective access time also increases.

Scanned by CamScanner

- page replacement Algorithm

when page fault occurs, the page replacement is needed. Page replacement means select a victim page in main memory and replace that page with required page from the backing store.

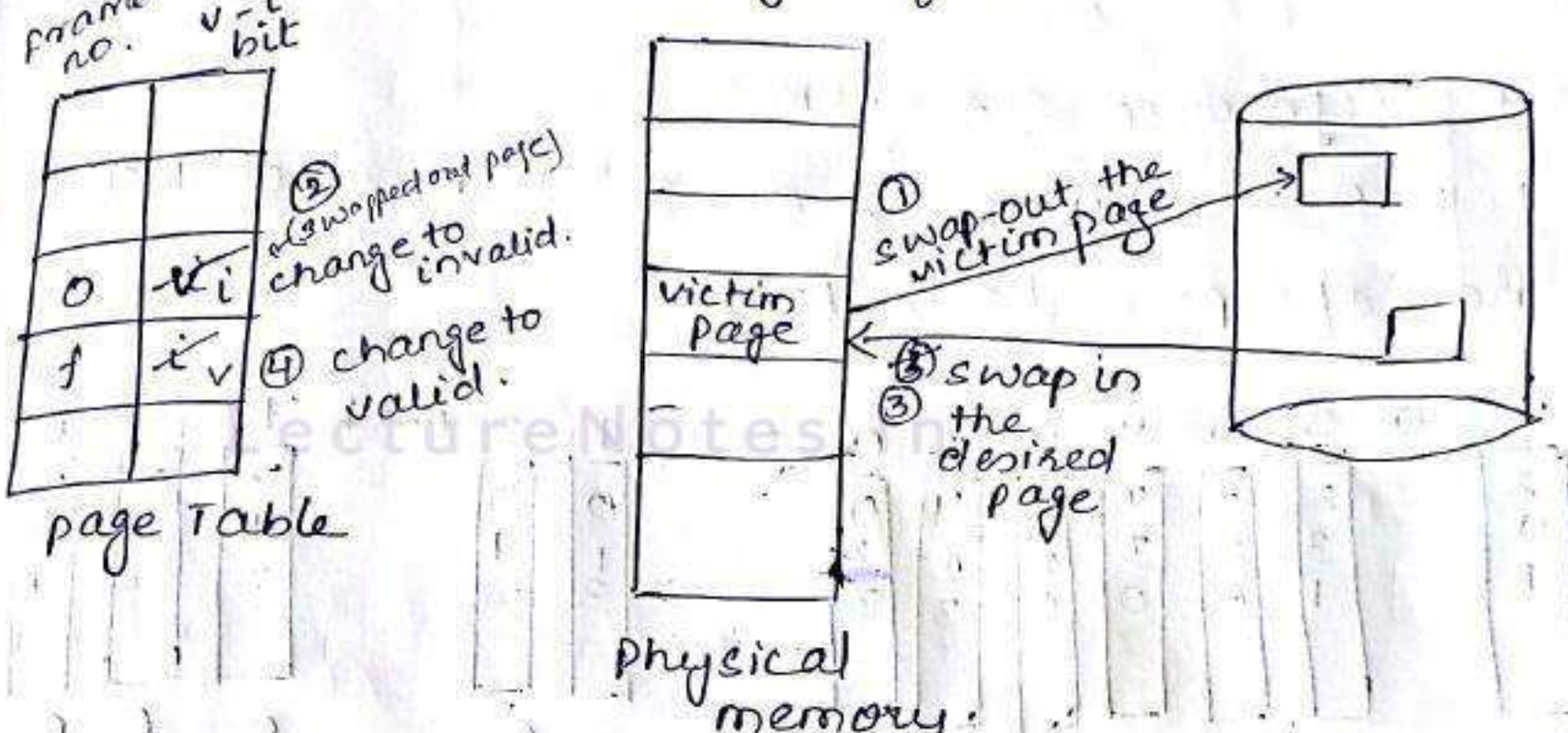
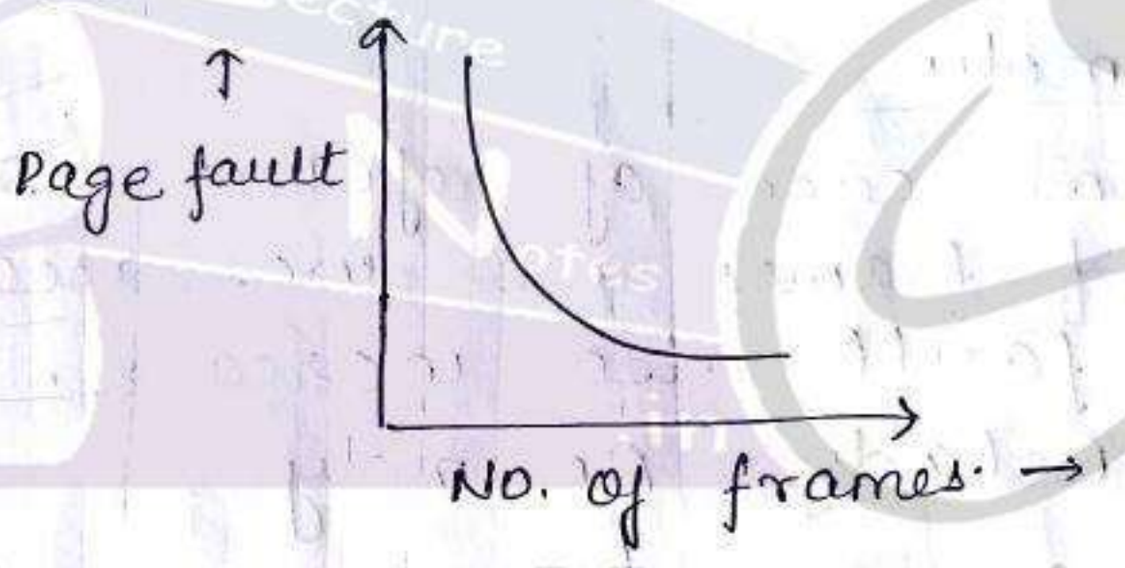


Fig: Page replacement working principle.

If the no. of frames increases the number of page fault decreases.



LectureNotes.in

Scanned by CamScanner

FIFO Replacement Algorithm

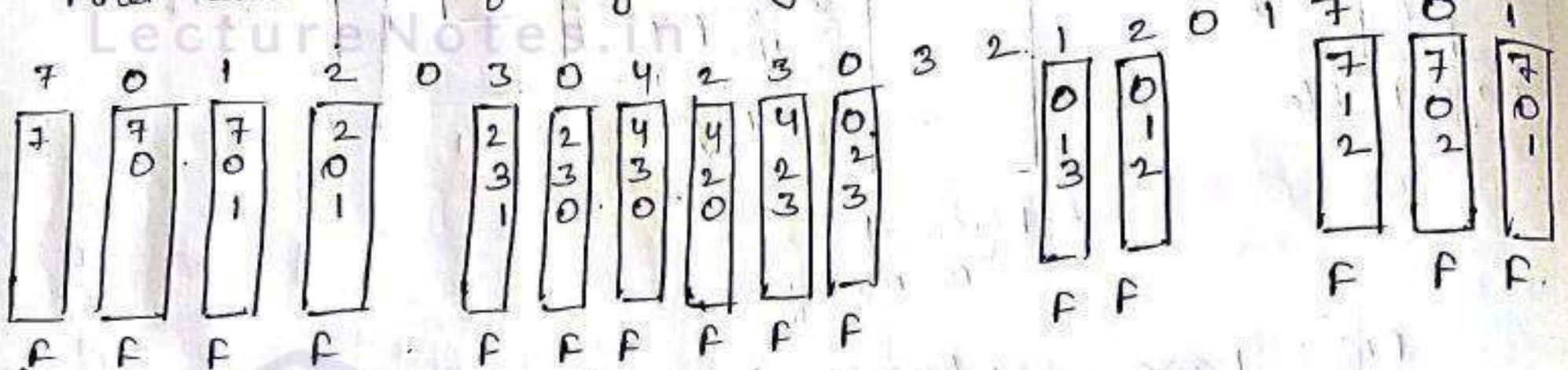
Replace the page of main memory on the principle of FIFO. It means replace the page which comes first.

Example -

Suppose reference string is

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

Find the no. of page layouts -



fault.

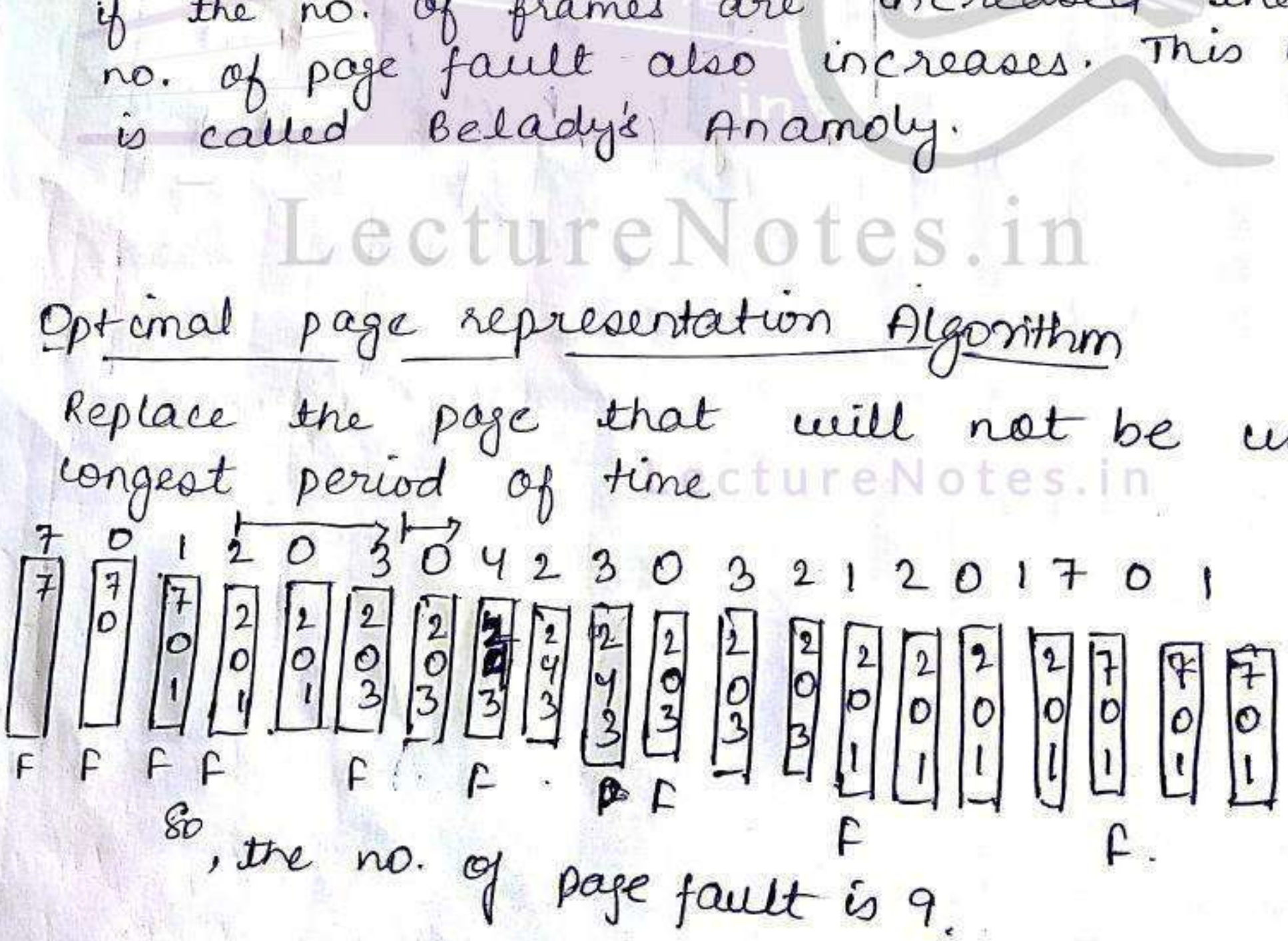
So, no. of page fault is 15.

Belady's Anamoly

This is some case of reference string is which if the no. of frames are increased then the no. of page fault also increases. This situation is called Belady's Anamoly.

Optimal page representation Algorithm

Replace the page that will not be used for longest period of time

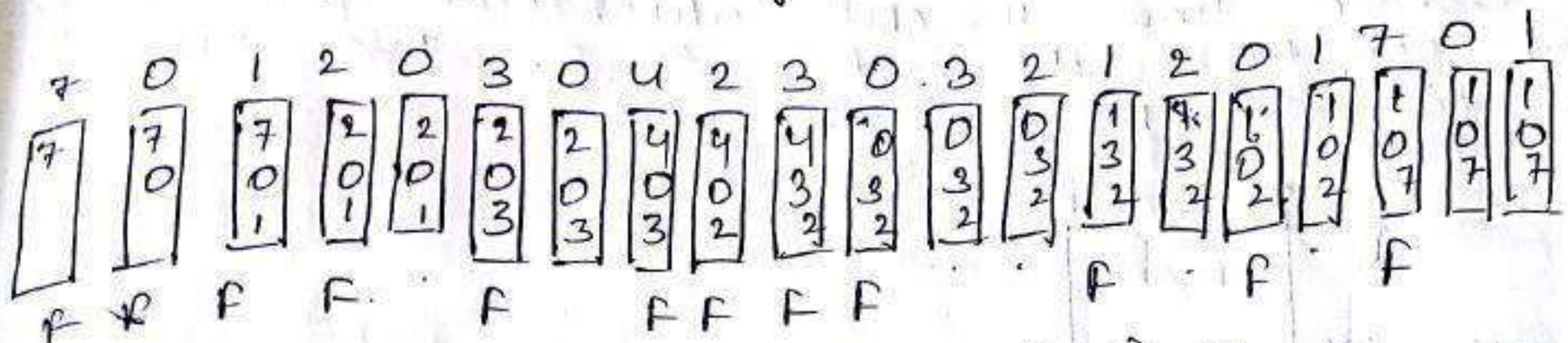


So, the no. of page fault is 9.

LectureNotes.in

Scanned by CamScanner

least Recently used (LRU) Page replacement Algorithm
Replace the page that has not been used for the longest period of time.



So the no. of page fault is 12.

LectureNotes.in

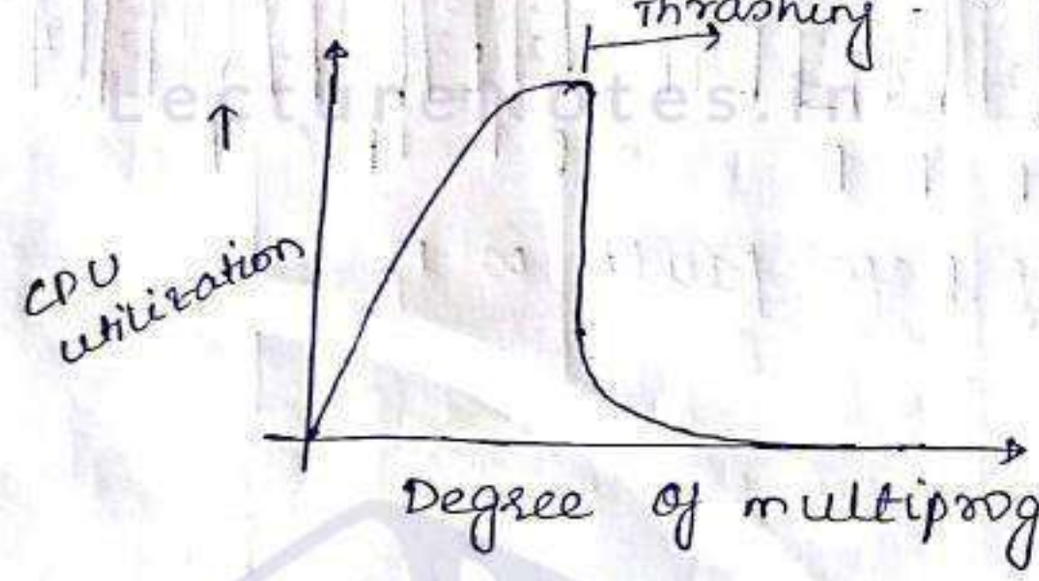
LectureNotes.in

LectureNotes.in

Scanned by CamScanner

VVI.

Thrashing :- If the no. of processes submitted to the CPU for execution are increased, the CPU utilization also increases but increasing the process continuously at certain time the CPU utilization falls sharply and sometimes it reaches to 0. This situation is said to be thrashing.



If the process does not have required no. of frames, it will cause quickly page fault. At this point, it must replace some page. since all its pages are in active use, it must replace a page that will not be needed again. As a result, it will I give fault again and again. This process continues to fault, replacing the pages for which it hit then faults and bring back the it in right away. This high paging activity is called thrashing.

A page is thrashing if it is spending more time in paging than executing.

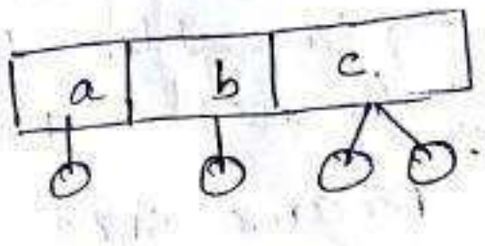
memory & virtual m/m complete.

Scanned by CamScanner

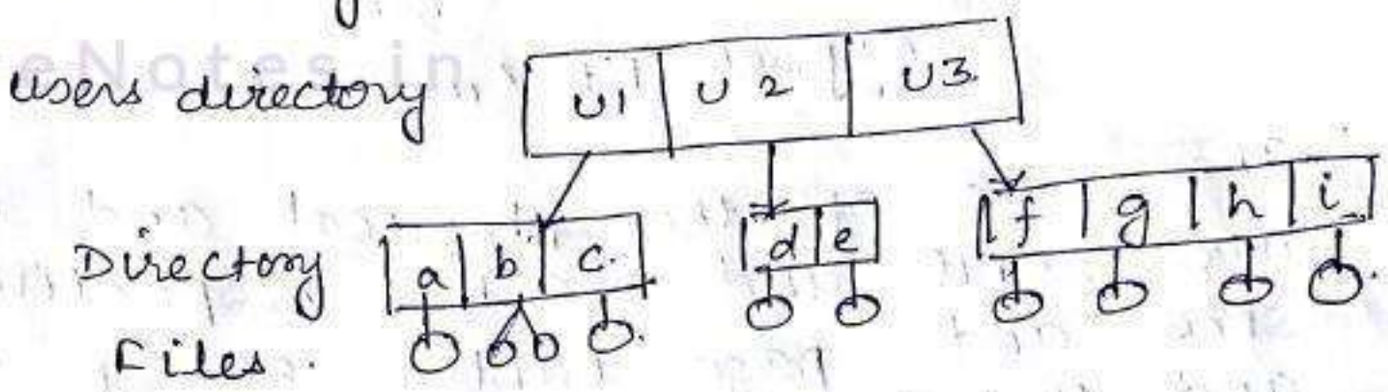
11/11/2017

File System Interface

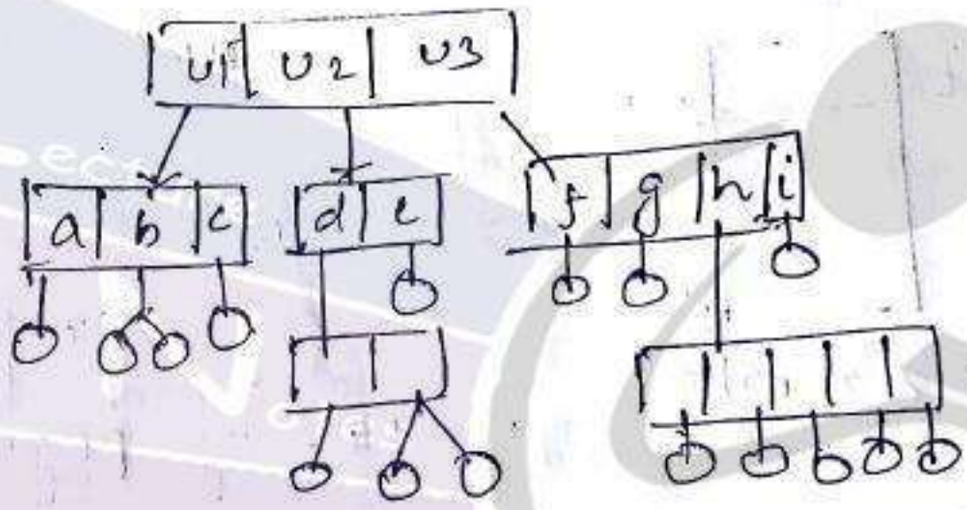
- File access method (LQ).
  i) sequential access ii) Direct access iii) Index sequential access.
- Directory structure (LQ)
  i) single level directory.



ii) two level directory



iii) Tree structured directory



LectureNotes.in

Scanned by CamScanner

→ problems (memory).

Q: calculate the page offset (bits) if the physical address is 24 bits and main memory having 128 frames.

→ $0 - 2^{24}-1$ size $= 2^{24}$

Physical address = 24 bits

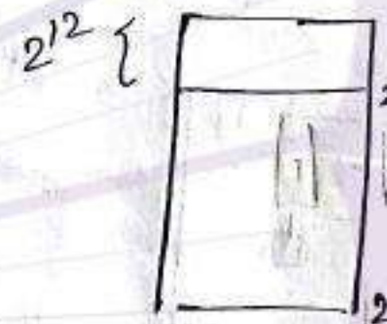
∴ Addressable locations $= 2^{24}$

No. of frames $= 128 = 2^7$

Page size = Frame size $= \frac{2^{24}}{2^7} = 2^{17} = 17$ bits.

Offset = 17 (max).

Q: The size of the physical and logical addresses are 25 bits and 29 bits respectively. If the frame size and page table entry size are 4KB and 4 byte respectively. what is the size of the page table.



4KB $= 2^{12}$

No.

Page size = frame size = 4KB

No. of pages $= \frac{2^{29}}{2^{12}} = 2^{17}$

There is an entry for every page in page table

∴ No. of entries in page table $= 2^{17}$.

size of page table $= 2^{17} \times 4$ B

$= 512$ KB

LectureNotes.in

Scanned by CamScanner

9/11/17

Ch - File System Implementation.

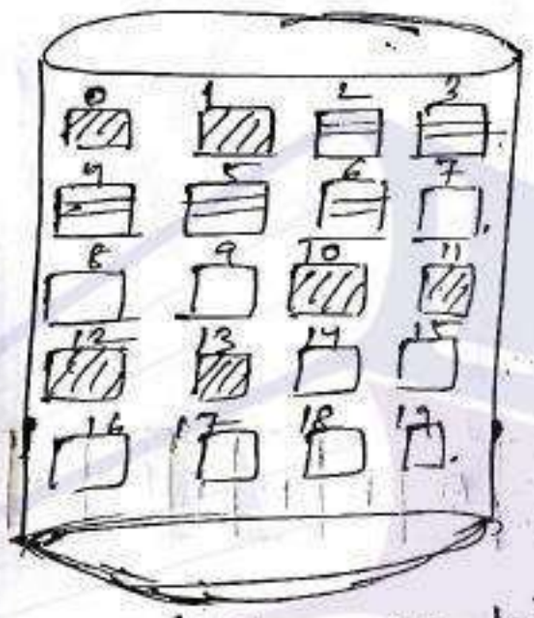
Application program
↓
Logical file system
↓
File organization Module
↓
Basic file system.
↓
I/O control.
↓
Device.

File system organization (LQ)

Allocation methods. (LQ)

i.) contiguous allocation
ii.) linked allocation
iii.) indexed allocation

i.)



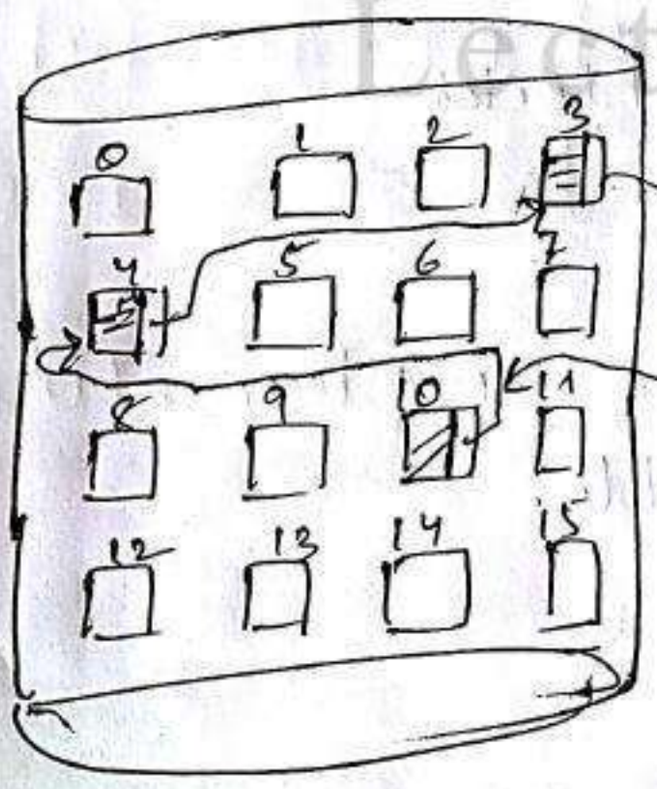
Directory.

File	start	length
test.c	0	2.
add.cpp	10	4.
sub.java	2	5.

Disadvantage

i.) size of bit space should be available from before
ii.) when size of file increases or decreases then fragmentation eliminated

ii.) Linked allocation.



Directory.

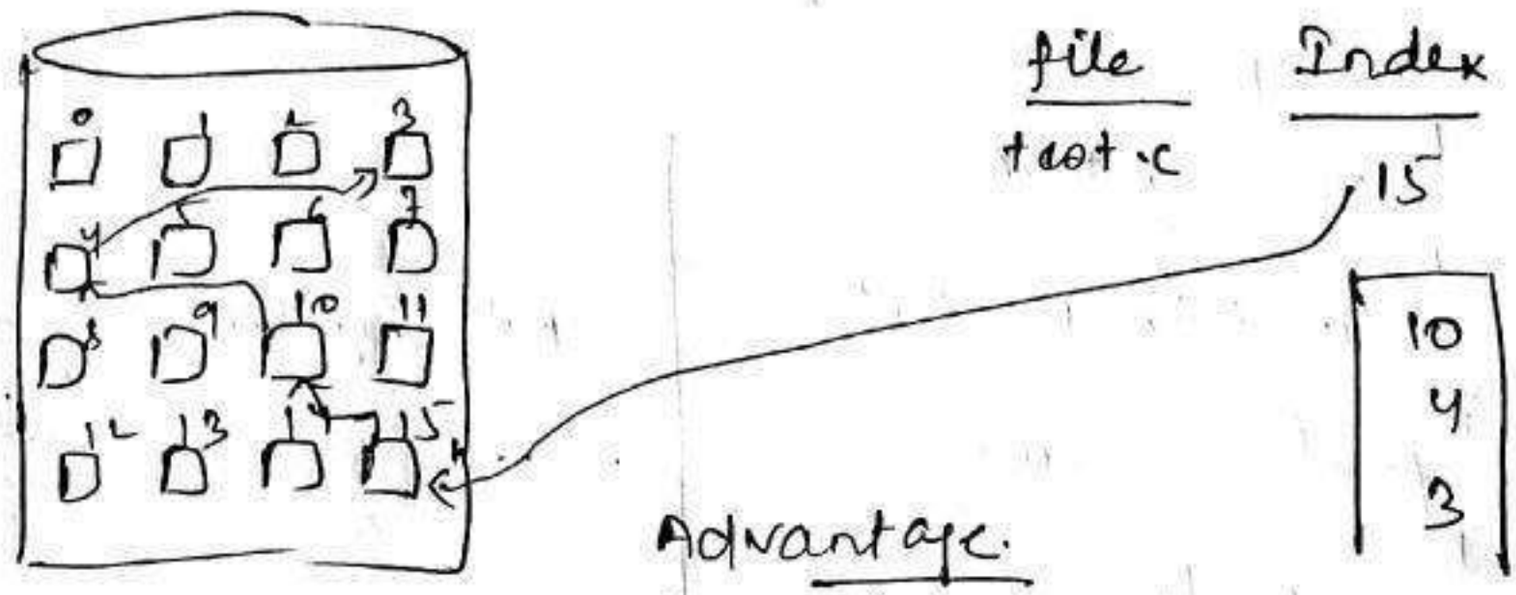
file.	start	end.
Test.c	10	3

Disadvantage

i.) for eg:- 4th block can't be accessed directly. u have to go through 10.
ii.) If block 4 misses then u can't move forward.
iii.) m/m occupied due to storing of address.

Scanned by CamScanner

iii) indexed allocation :-

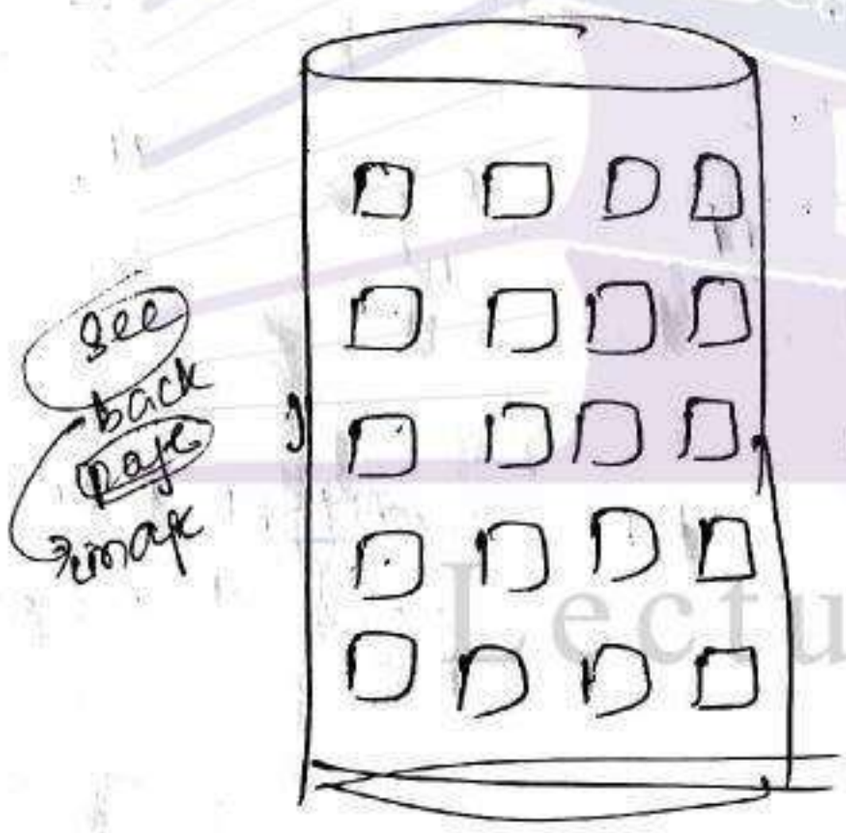


Advantage.
Even if one address misses then we can use index to go to next block

→ Performance consideration.

i.) free space management.

ii.) Bit vector. (2 marks)



iii.) linked list

iv.) counting (count the free blocks)

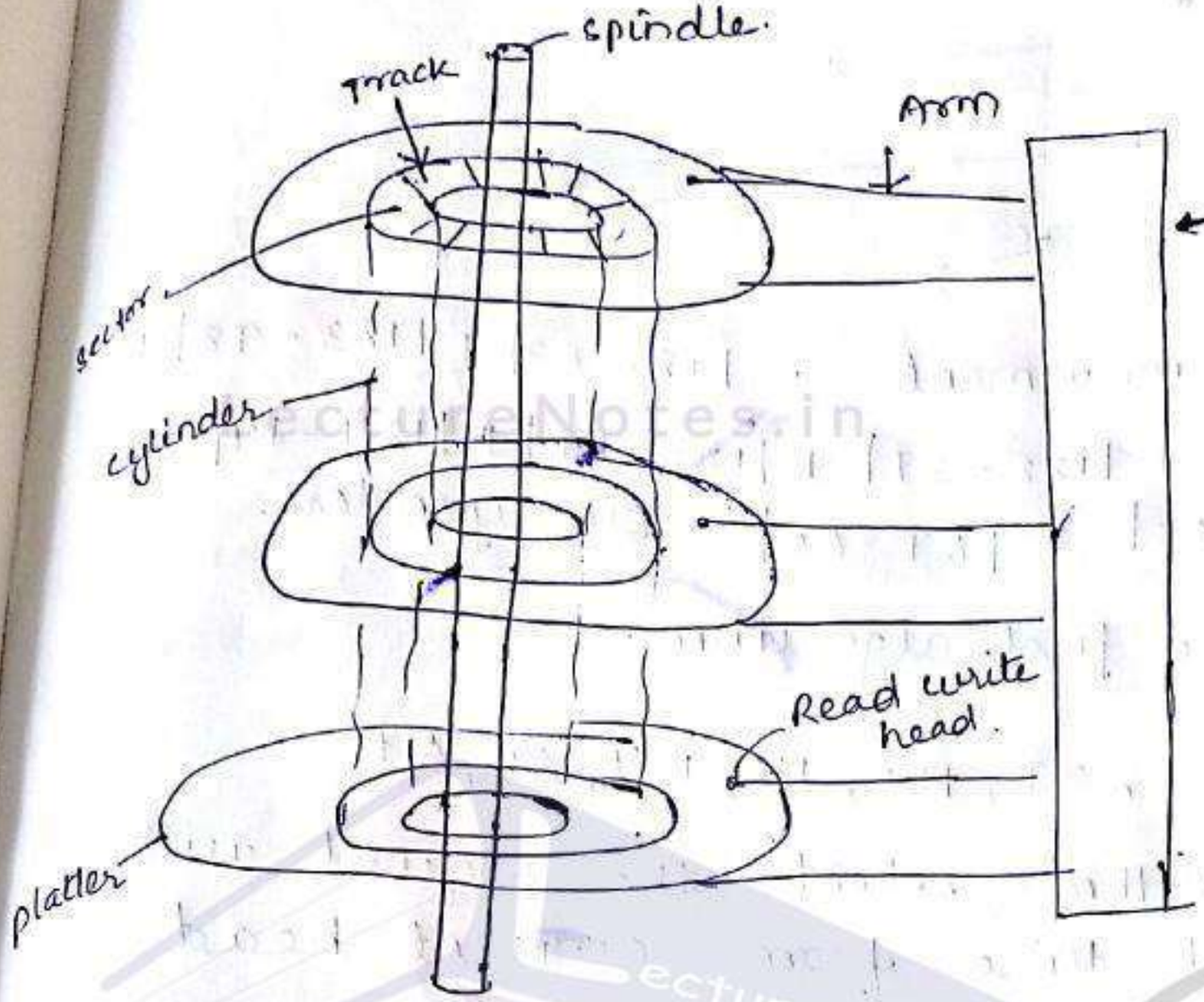
→ Directory implementation.
There are two methods for implementing directory
linear list and hash table.

LectureNotes.in
Scanned by CamScanner

Ahead 3/11/17

Mass storage structure

1. Magnetic Disc :-



1.) Transfer rate : It is the rate at which data flow b/w drive and computer.

2.) Access time consist of two parts :-

a.) seek time :- The time required to move the disc arm to the desired cylinder.

b.) Rotational latency :- The time required for the desired sector to rotate the disc head.

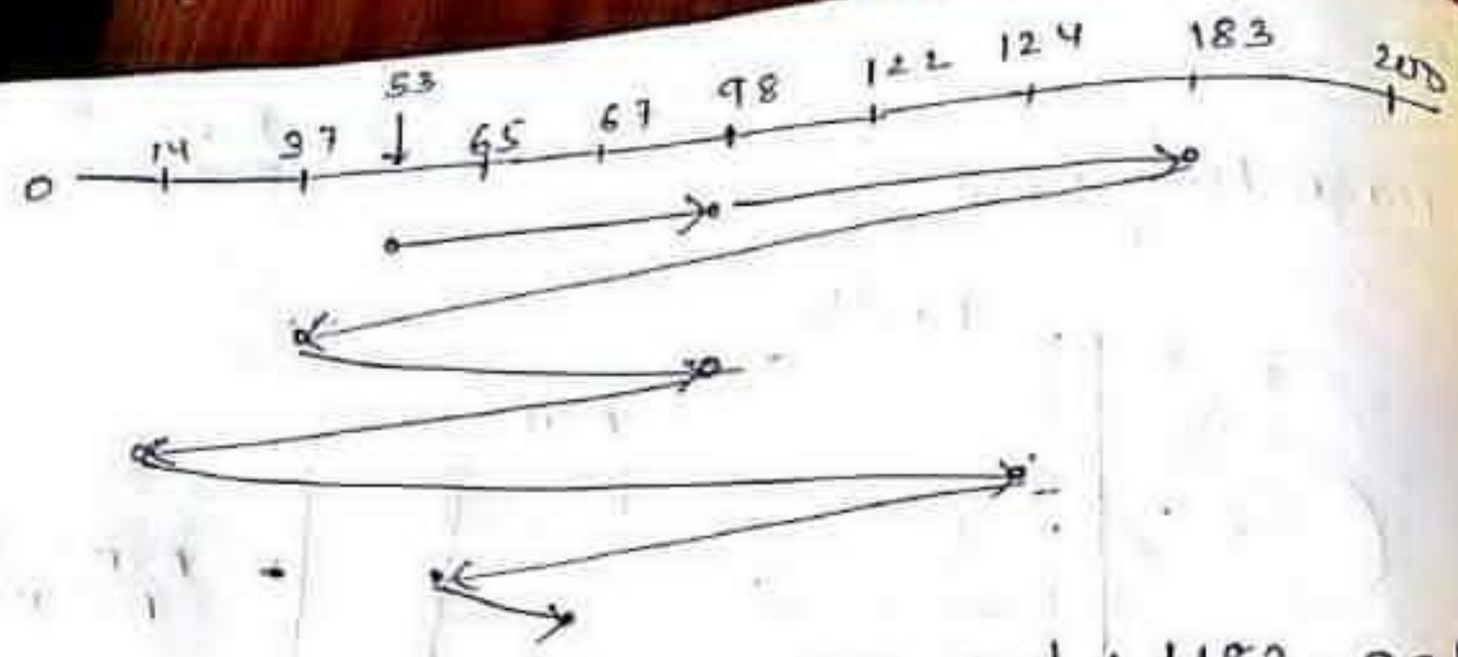
- Disc Scheduling :-
- FCFS scheduling algorithm :-

A disc queue request the blocks on cylinders of following order.

Queue : 98, 183, 37, 122, 14, 124, 65, 67

Head start at : 53. Find the total head movement.

Scanned by CamScanner

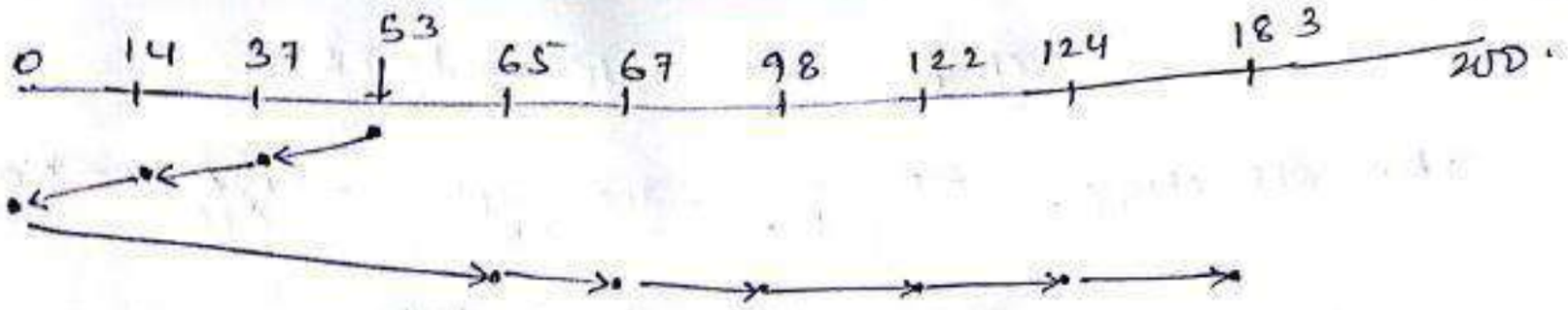


∴ Total head movement = |98 - 53| + |183 - 98| + |183 - 37| + |122 - 37| + |122 - 14| + |124 - 14| + |124 - 65| + |67 - 65| = 640 cylinders.

- shortest seek time first algorithm.

queue : 98, 183, 37, 122, 14, 124, 65, 67

The SSTF algorithm selects the request with

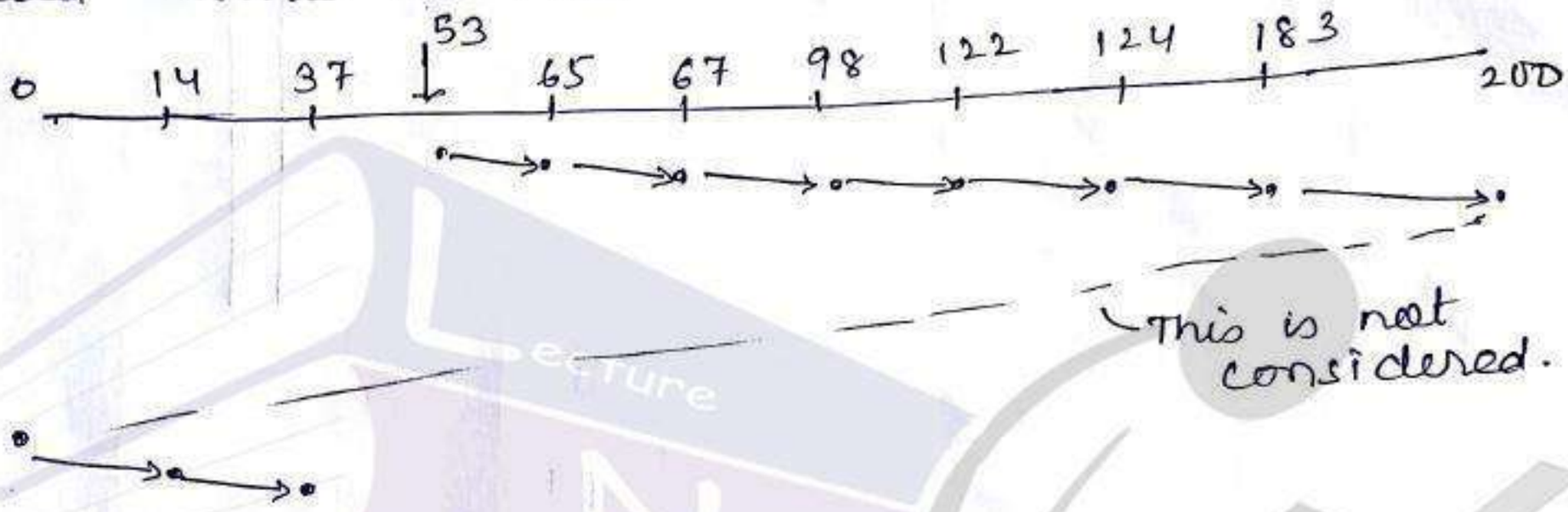


Total head movement = |53-37| + |14-37| + |14-0|
+ |65-0| + |67-65| + |98-67| +
|122-98| + |124-122| + |183-124| = 236.

Circular-Scan (C-SAN) Algorithm

Queue : 98, 183, 37, 122, 14, 124, 65, 67.

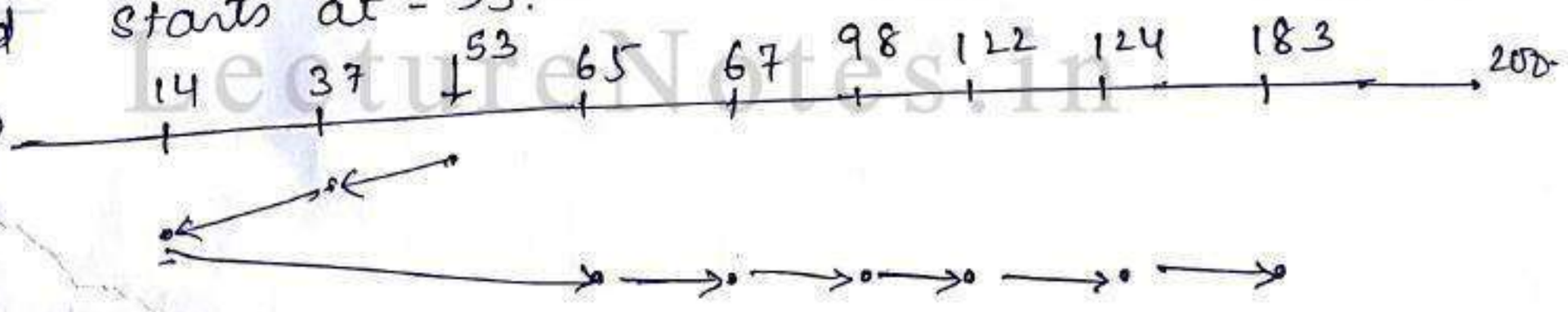
Head starts at : 53



LOOK Scheduling

Queue : 98, 183, 37, 122, 14, 124, 65, 67

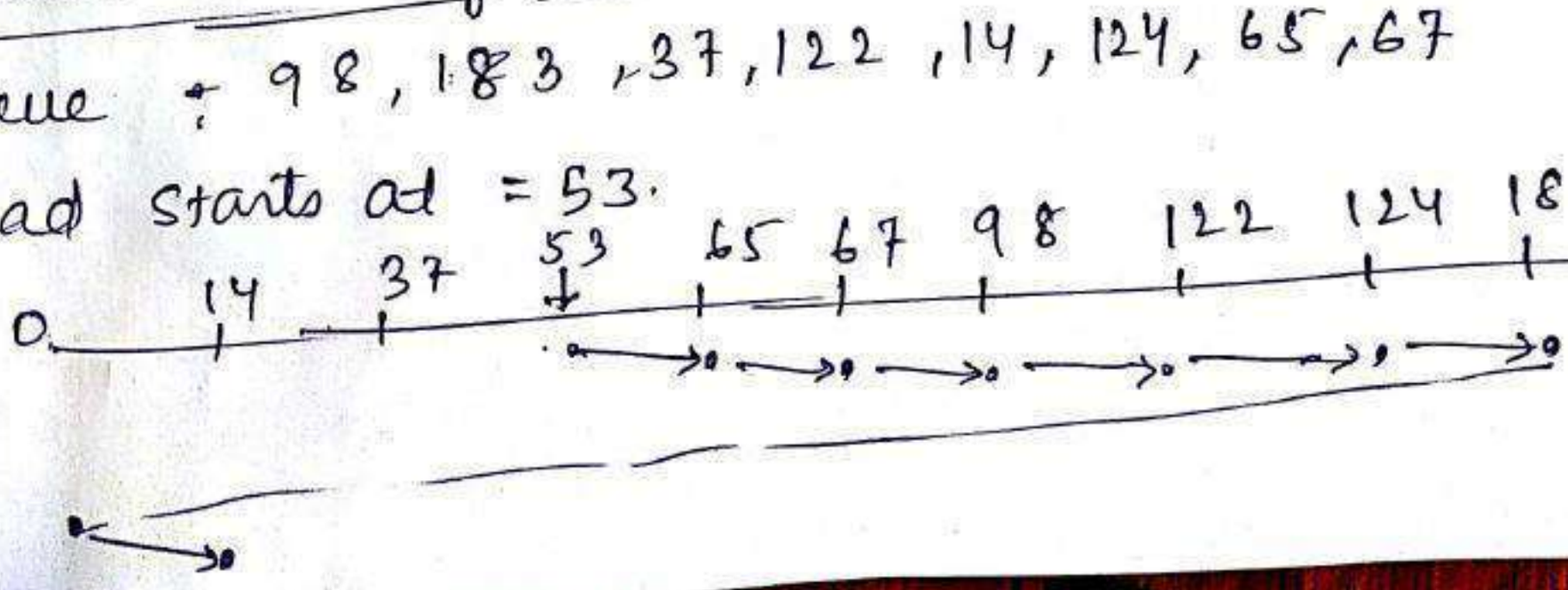
Head starts at = 53.



Circular LOOK algorithm (C-LOOK)

Queue : 98, 183, 37, 122, 14, 124, 65, 67

Head starts at = 53.



Scanned by CamScanner

7/11/17.

chapter :13 I/O systems.

- bus structure (LQ)
- Polling (SQ)
- Interrupt (LQ)
- DMA (LQ)
- I/O request to hardware operation (LQ)

chapter :14 protection.

- Domain structure.
- Access matrix.
- Access matrix of figure A with domains as objects.
- Access matrix with copy rights.
- Access matrix with owner rights.

ch15: security

virus worm.

Friday - Test (mid)

LectureNotes.in

Scanned by CamScanner

LectureNotes.in

File System

Dr. Ajay Kumar Jena
School of Computer Engineering
KIIT University, Bhubaneswar

LectureNotes.in
LectureNotes.in

LectureNotes.in

ACKNOWLEDGEMENT

I am very much thankful to say that some contents of these slides are taken from the book

Operating System Concepts
by
Silberschatz, Galvin and Gagne.

LectureNotes.in

By Ajay Kumar Jena, KIIT Deemed to be University, Bhubaneswar

LectureNotes.in

File Concept

- It is a collection of related information that is recorded in the secondary storage
- Contiguous logical address space (programs i.e. source and object forms)
- File control bloc: storage structure consisting of information about a file
- Types:
  - Data
    - Numeric, alphanumeric
    - Character, alphabetic
    - binary
  - Program
- Contents defined by file's creator
  - Many types
    - Consider **text file, source file, executable file**
- A file is a sequence of bits, bytes, lines, records.

LectureNotes.in

LectureNotes.in

By Ajay Kumar Jena, KIIT Deemed to be University, Bhubaneswar

LectureNotes.in

File Attributes

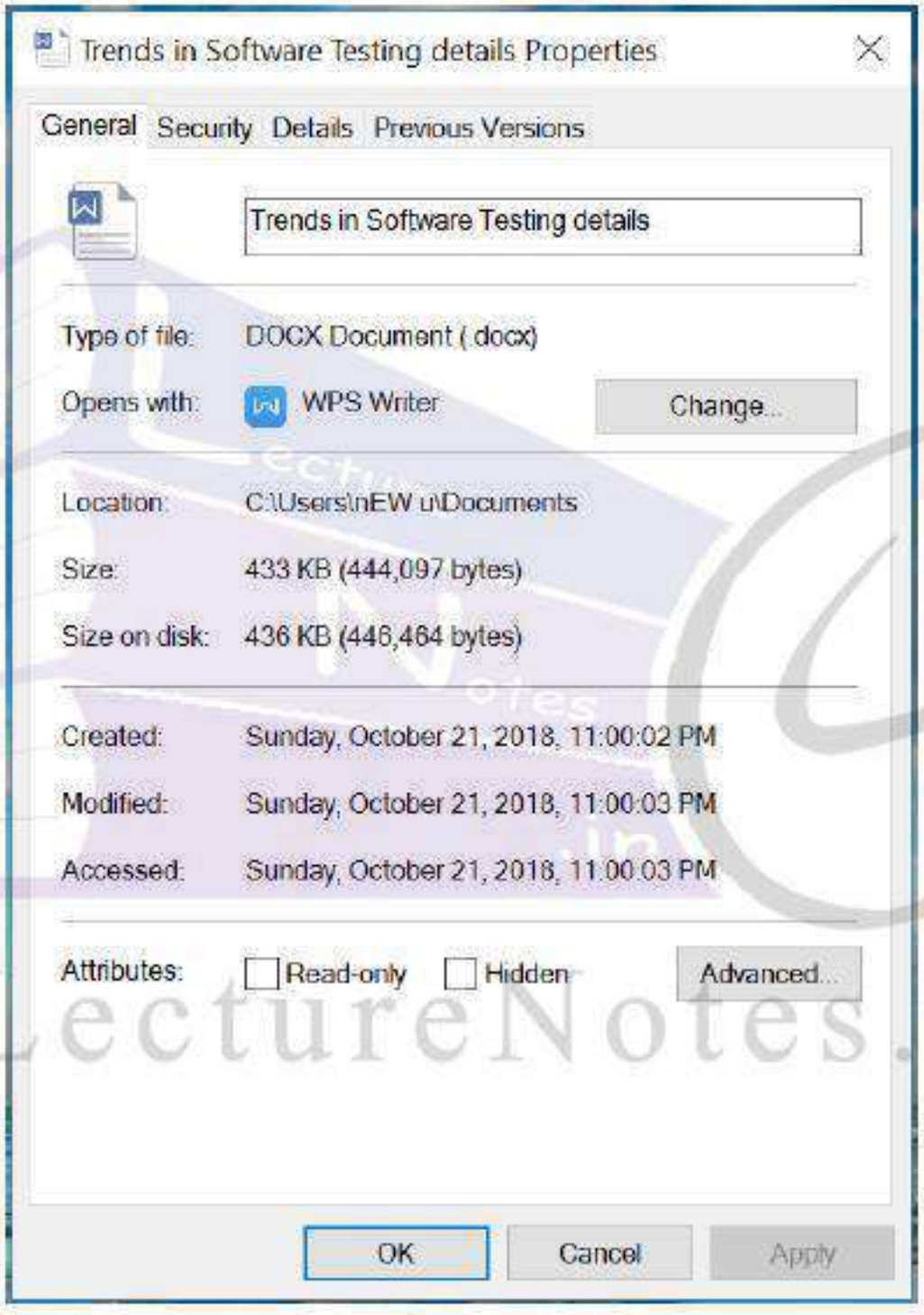
- A file is named, for the convenience of its users and is referred to you by name
- **Name** – symbolic file name is the only information kept in human-readable form
- **Identifier** – unique tag (number) identifies file within file system
- **Type** – needed for systems that support different types
- **Location** – pointer to file location on device
- **Size** – current file size
- **Protection** – controls who can do reading, writing, executing
- **Time, date, and user identification** – data for protection, security, and usage monitoring
- Information about files are kept in the directory structure, which is maintained on the disk
- Many variations, including extended file attributes such as file checksum
- Information kept in the directory structure

LectureNotes.in

LectureNotes.in

By Ajay Kumar Jena, KIIT Deemed to be University, Bhubaneswar

File info Window on Mac OS X



LectureNotes.in

By Ajay Kumar Jena, KIIT Deemed to be University, Bhubaneswar

LectureNotes.in

File Operations

- File is an **abstract data type**
- **Create** – Space in file system must be found, an entry for the new file will me made in the directory
- **Write** – To make a system call to write at **write pointer** location
- **Read** – at **read pointer** location
- **Reposition within file** - **seek**
- **Delete**
- **Truncate:** User may erase the content of the file but wants to hold the attributes
- ***Open(F_i)*** – search the directory structure on disk for entry F_i, and move the content of entry to memory
- ***Close (F_i)*** – move the content of entry F_i in memory to directory structure on disk

LectureNotes.in

By Ajay Kumar Jena, KIIT Deemed to be University, Bhubaneswar

File Types – Name, Extension

file type	usual extension	function
executable	exe, com, bin or none	ready-to-run machine-language program
object	obj, o	compiled, machine language, not linked
source code	c, cc, java, pas, asm, a	source code in various languages
batch	bat, sh	commands to the command interpreter
text	txt, doc	textual data, documents
word processor	wp, tex, rtf, doc	various word-processor formats
library	lib, a, so, dll	libraries of routines for programmers
print or view	ps, pdf, jpg	ASCII or binary file in a format for printing or viewing
archive	arc, zip, tar	related files grouped into one file, sometimes compressed, for archiving or storage
multimedia	mpeg, mov, rm, mp3, avi	binary file containing audio or A/V information

LectureNotes.in

By Ajay Kumar Jena, KIIT Deemed to be University, Bhubaneswar

LectureNotes.in

File Structure

- None - sequence of words, bytes
- Simple record structure
  - Lines
  - Fixed length
  - Variable length
- Complex Structures
  - Formatted document
  - Relocatable load file
- Can simulate last two with first method by inserting appropriate control characters
- Who decides:
  - Operating system
  - Program

LectureNotes.in

By Ajay Kumar Jena, KIIT Deemed to be University, Bhubaneswar

LectureNotes.in

Access Methods

- **The information in the file can be accessed in several ways.**
- **Sequential Access.** The most simplest method. Information accessed sequentially one record after another

```
read next
write next
```

no read after last write (rewrite)

- **Direct Access –** When particular record/s are needed to be accessed out of many record. Randomly access any file block from HDD.

```
read n
write n
position to n
read next
write next
rewrite n
```

n = relative block number

- **Index Sequential Access:** Combination of sequential access and direct access. This access method involves maintaining an index. Te index is a pointer to the block.

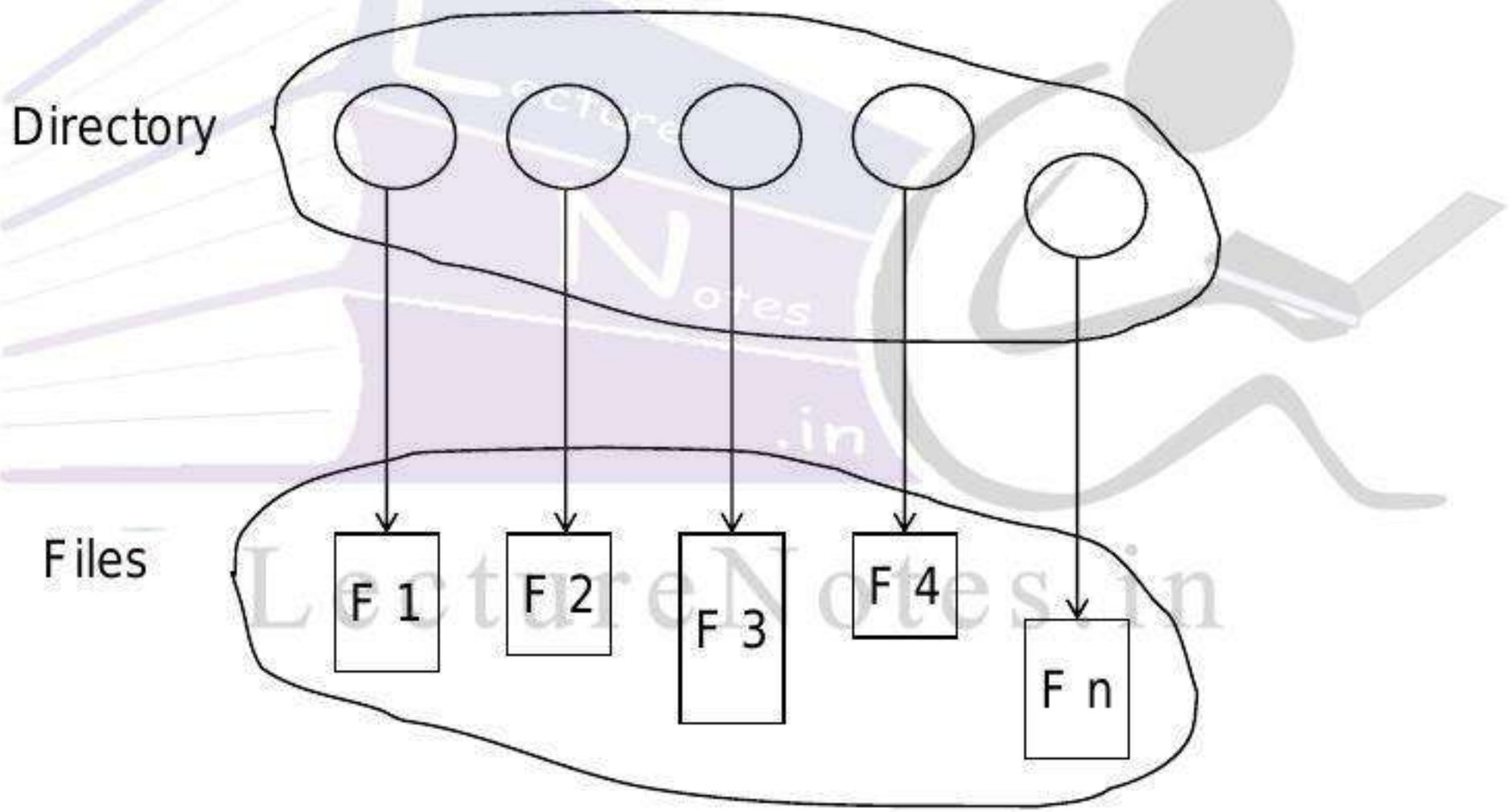
LectureNotes.in

By Ajay Kumar Jena, KIIT Deemed to be University, Bhubaneswar

LectureNotes.in

Directory Structure

- Files are to be organised. How to store the file.
- A collection of nodes containing information about all files



Both the directory structure and the files reside on disk

LectureNotes.in

By Ajay Kumar Jena, KIIT Deemed to be University, Bhubaneswar

LectureNotes.in

Disk Structure

- Disk can be subdivided into **partitions**
- Disks or partitions can be **RAID** protected against failure
- Disk or partition can be used **raw** – without a file system, or **formatted** with a file system
- Partitions also known as minidisks, slices
- Entity containing file system known as a **volume**
- Each volume containing file system also tracks that file system's info in **device directory** or **volume table of contents**
- As well as **general-purpose file systems** there are many **special-purpose file systems**, frequently all within the same operating system or computer

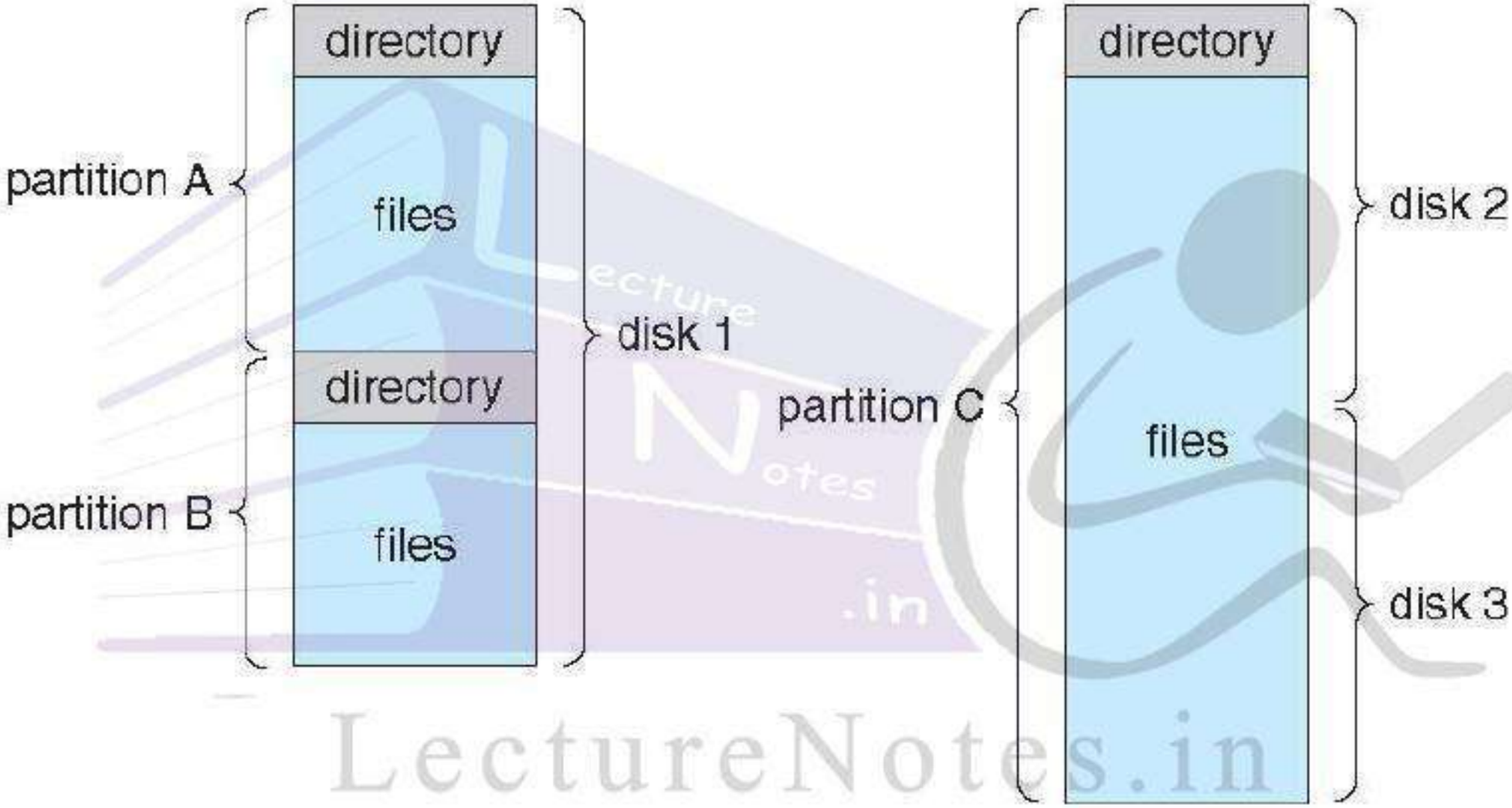
LectureNotes.in

LectureNotes.in

By Ajay Kumar Jena, KIIT Deemed to be University, Bhubaneswar

LectureNotes.in

A Typical File-system Organization



LectureNotes.in
LectureNotes.in

By Ajay Kumar Jena, KIIT Deemed to be University, Bhubaneswar

LectureNotes.in

Operations Performed on Directory

- Search for a file
- Create a file
- Delete a file
- List a directory
- Rename a file
- Traverse the file system

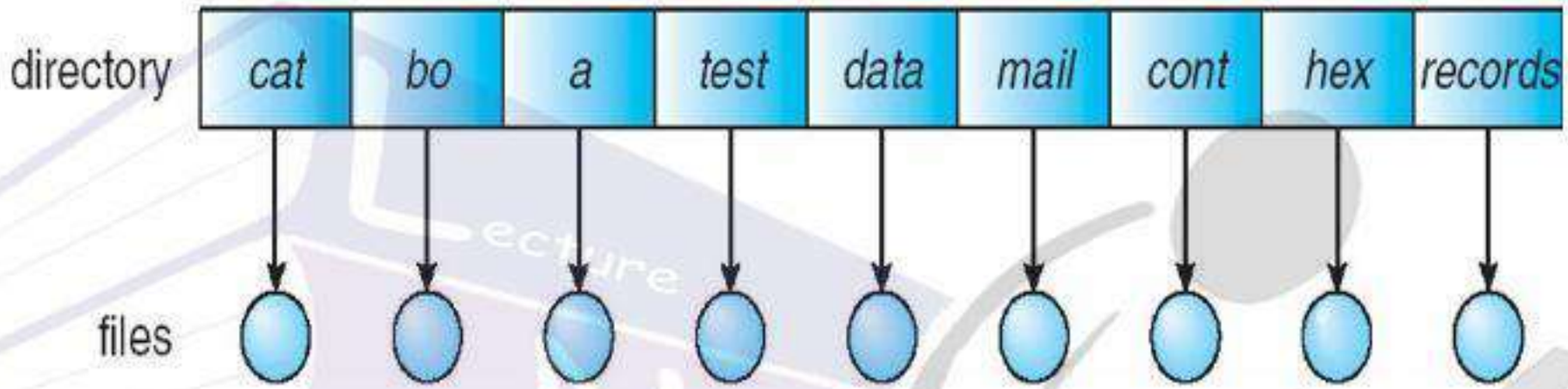
LectureNotes.in

LectureNotes.in

By Ajay Kumar Jena, KIIT Deemed to be University, Bhubaneswar

Single-Level Directory

- The simplest directory structure is a single directory for all users.
- All files reside in one and the same directory.



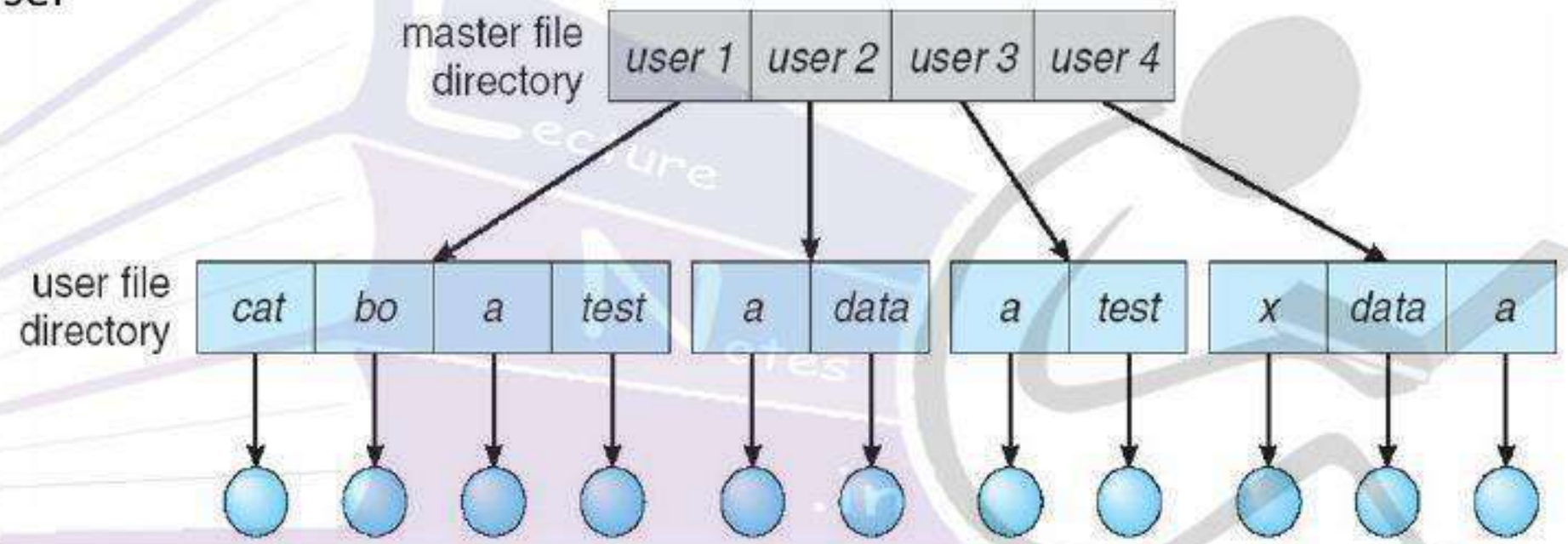
- Naming problem (No two files having same name)
- Grouping problem

LectureNotes.in

By Ajay Kumar Jena, KIIT Deemed to be University, Bhubaneswar

Two-Level Directory

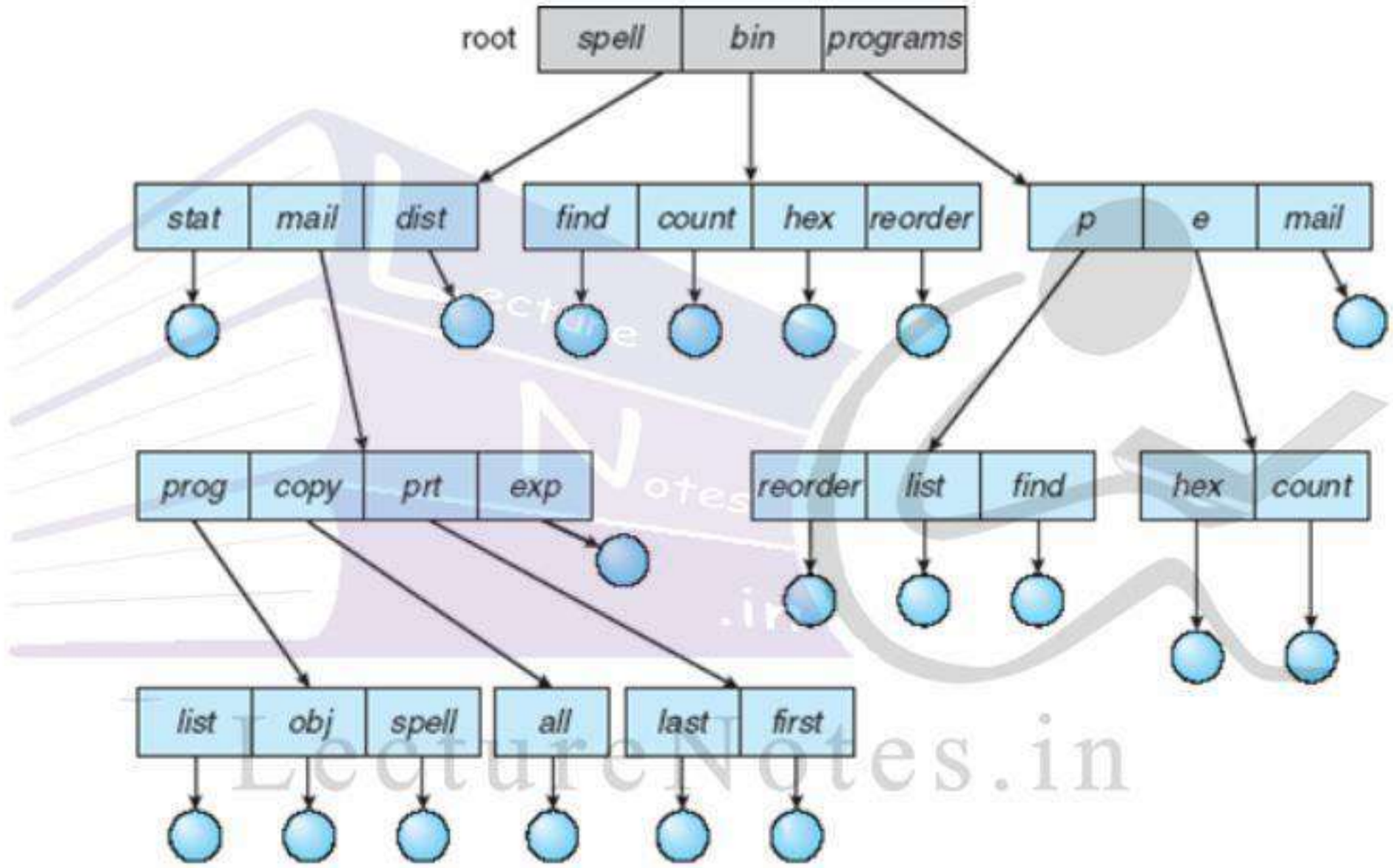
- The main drawback of single-level is to have unique file names by different users. It can be solved by creating separate directories.
- Each user has its own user file directory (UFD). Separate directory for each user



- One directory exclussive for each user. Path name
- Can have the same file name for different user
- Efficient searching
- No grouping capability

LectureNotes.in

Tree-Structured Directories



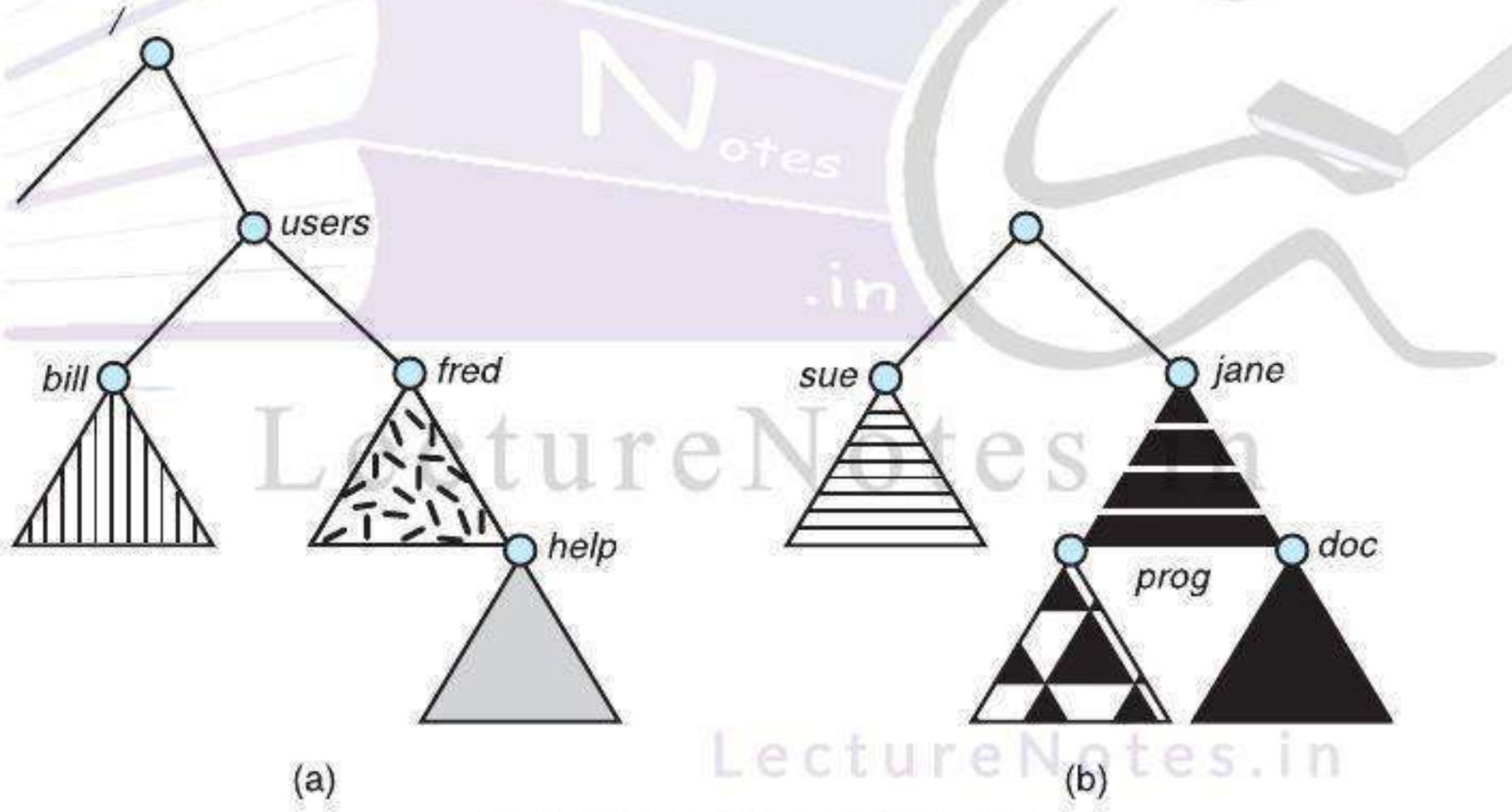
LectureNotes.in

LectureNotes.in

By Ajay Kumar Jena, KIIT Deemed to be University, Bhubaneswar

File System Mounting

- As a file must open before it is used, a file system must be **mounted** before it can be accessed.
- The location within the file system at which to attach a new file system is known as mount point
- A unmounted file system (i.e., Fig. (b)) is mounted at a **mount point**

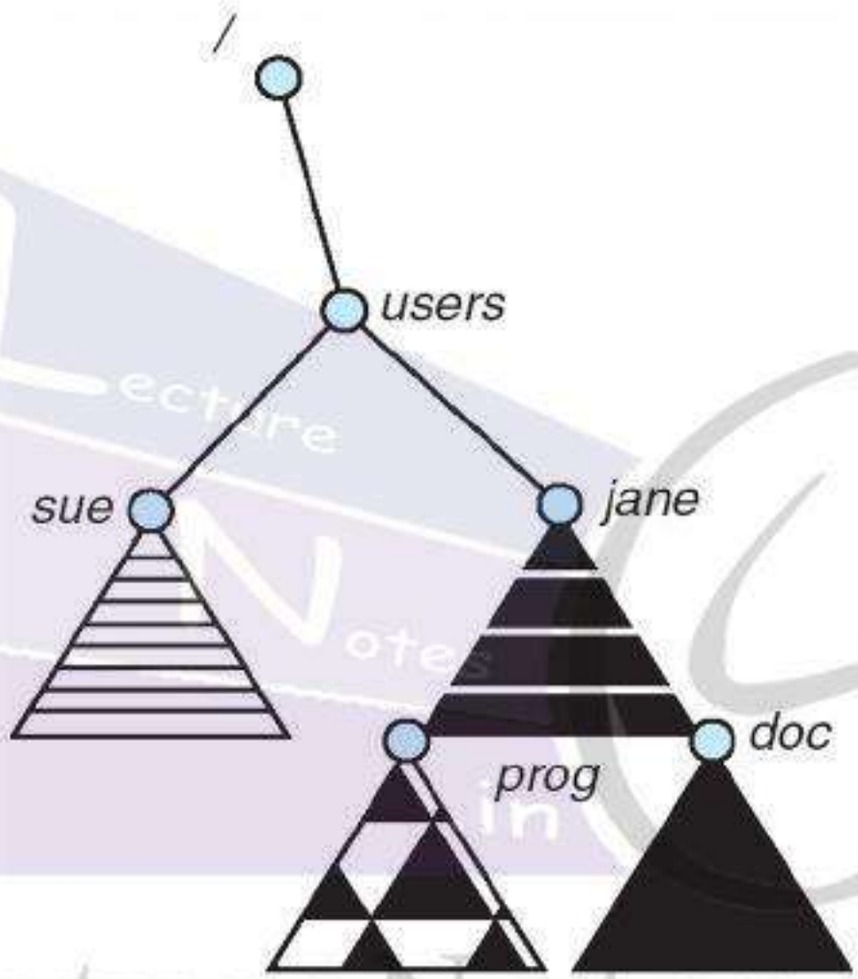


LectureNotes.in

By Ajay Kumar Jena, KIIT Deemed to be University, Bhubaneswar

LectureNotes.in

Mount Point



LectureNotes.in
LectureNotes.in

By Ajay Kumar Jena, KIIT Deemed to be University, Bhubaneswar

File-System Structure

- File is the most important part in computer. Bcoz all the data stores in files. It is the part of operating system. IO,Process, everything is a file. It is a sw which is managing all these files. It stores data and program.
- File structure
  - Logical storage unit, Collection of related information
  - Allocating disk space
  - Recovering the free space
- **File system** resides on secondary storage (disks)
  - Provided user interface to storage, mapping logical to physical
  - Provides efficient and convenient access to disk by allowing data to be stored, located retrieved easily
- Disk provides in-place rewrite and random access
  - I/O transfers performed in **blocks** of **sectors** (usually 512 bytes)
- **File control block** – storage structure consisting of information about a file
- **Device driver** controls the physical device
- File system organized into layers

LectureNotes.in
Dr. Ajay Kumar Jena, KIIT Deemed to be University, Bhubaneswar

Layered File System

By designing ay SW we use layring approach.
We are going to divide all the task in to groups.
When one try to modify one layer it will affect the other layer.
Application program desires for a file i.e create, modify, append and read it will ask the request to the logical file system . LFS has meta data (who is the use, permission, when created with directory structure)
FOM: File is divided into logical blocks. Mapping of LB=Phy. Block.) also free spae mgt.
BFS: Read/Write. Read one block. issue the command to the IO control to do it.
IOC: consists of device drivers, interrupt handler
Devices: IO control will access the device.

LectureNotes.in

File System Layers

- **Device drivers** manage I/O devices at the I/O control layer
  - Given commands like "read drive1, cylinder 72, track 2, sector 10, into memory location 1060" outputs low-level hardware specific commands to hardware controller
- **Basic file system** given command like "retrieve block 123" translates to device driver
- Also manages memory buffers and caches (allocation, freeing, replacement)
  - Buffers hold data in transit
  - Caches hold frequently used data
- **File organization module** understands files, logical address, and physical blocks
  - Translates logical block # to physical block #
  - Manages free space, disk allocation

LectureNotes.in

By Ajay Kumar Jena, KIIT Deemed to be University, Bhubaneswar

File System Layers (Cont.)

- **Logical file system** manages metadata information
  - Translates file name into file number, file handle, location by maintaining file control blocks (**inodes** in UNIX)
  - Directory management
  - Protection
- Layering useful for reducing complexity and redundancy, but adds overhead and can decrease performanceTranslates file name into file number, file handle, location by maintaining file control blocks (**inodes** in UNIX)
  - Logical layers can be implemented by any coding method according to OS designer

LectureNotes.in

LectureNotes.in

By Ajay Kumar Jena, KIIT Deemed to be University, Bhubaneswar

LectureNotes.in

File System Layers (Cont.)

- Many file systems, sometimes many within an operating system
  - Each with its own format (CD-ROM is ISO 9660; Unix has **UFS**, FFS; Windows has FAT, FAT32, NTFS as well as floppy, CD, DVD Blu-ray, Linux has more than 40 types, with **extended file system** ext2 and ext3 leading; plus distributed file systems, etc.)
  - New ones still arriving – ZFS, GoogleFS, Oracle ASM, FUSE

LectureNotes.in

LectureNotes.in

By Ajay Kumar Jena, KIIT Deemed to be University, Bhubaneswar

File-System Implementation

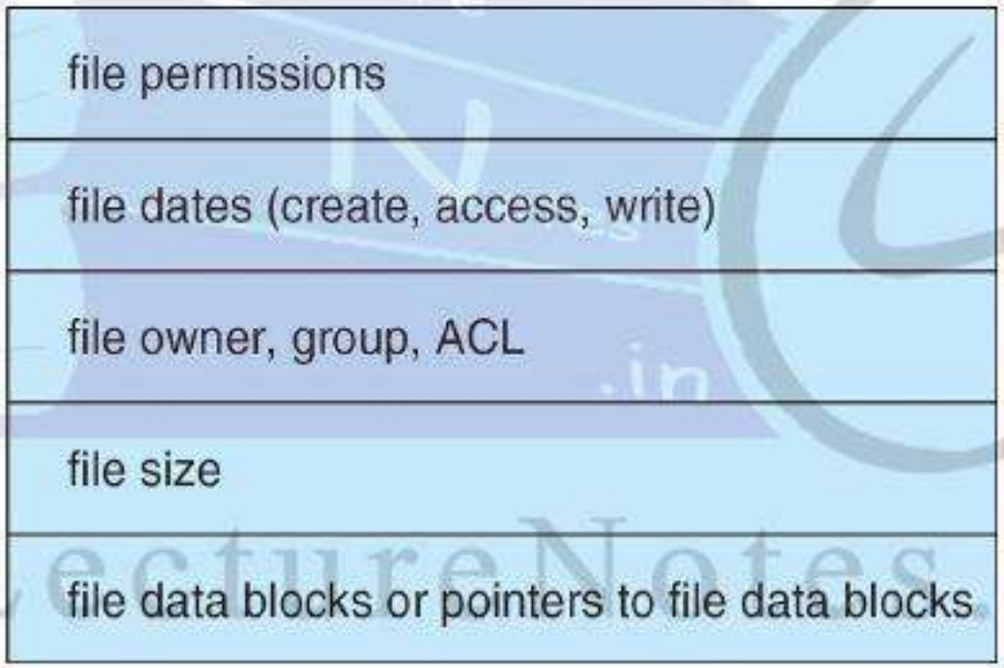
- We have system calls at the API level, but how do we implement their functions?
  - On-disk and in-memory structures
- **Boot control block** contains info needed by system to boot OS from that volume
  - Needed if volume contains OS, usually first block of volume
- **Volume control block** (**superblock, master file table**) contains volume details
  - Total # of blocks, # of free blocks, block size, free block pointers or array
- Directory structure organizes the files
  - Names and inode numbers, master file table

LectureNotes.in

By Ajay Kumar Jena, KIIT Deemed to be University, Bhubaneswar

File-System Implementation (Cont.)

- Per-file **File Control Block (FCB)** contains many details about the file
  - inode number, permissions, size, dates
  - NFTS stores into in master file table using relational DB structures



LectureNotes.in

By Ajay Kumar Jena, KIIT Deemed to be University, Bhubaneswar

LectureNotes.in

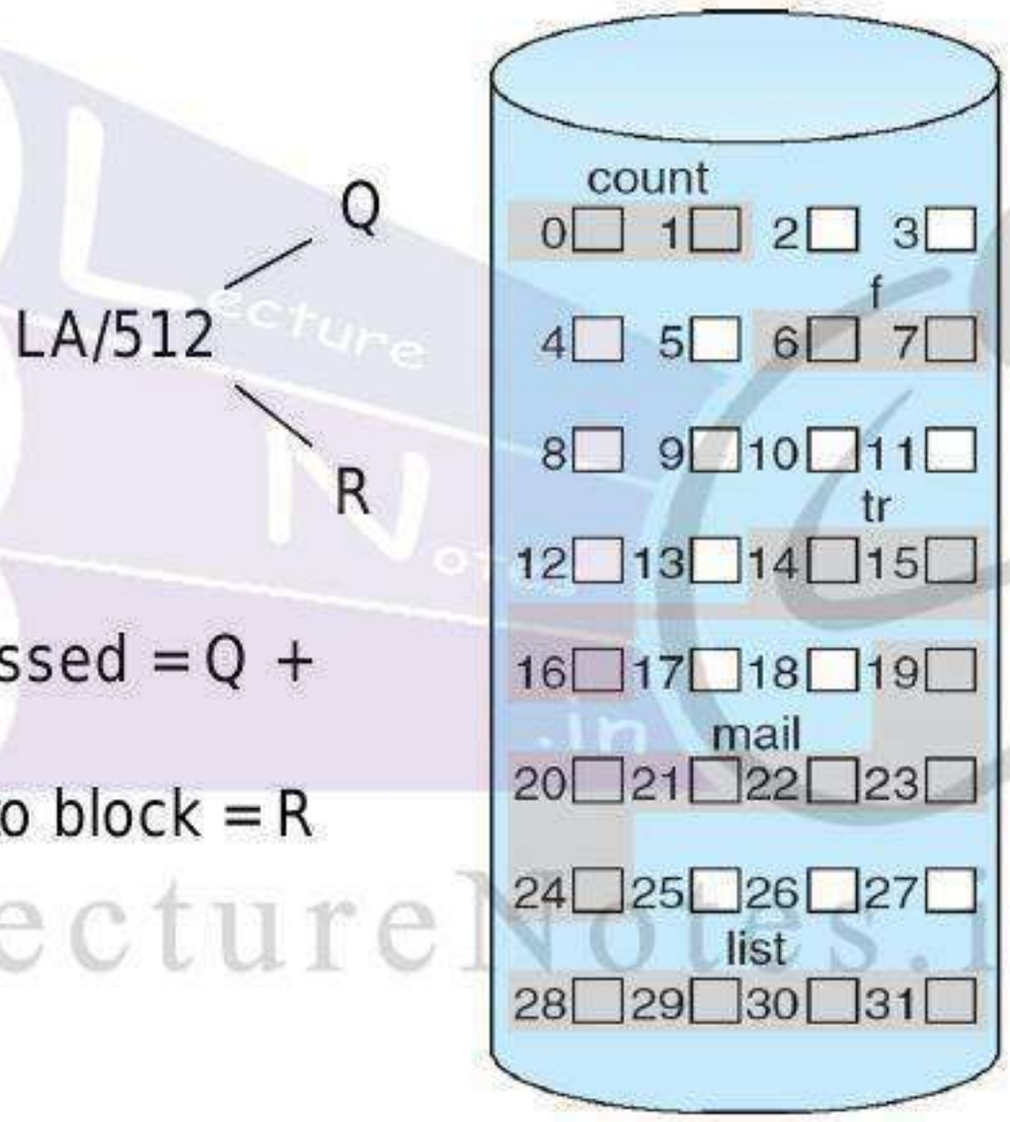
Allocation Methods - Contiguous

- An allocation method refers to how disk blocks are allocated for files:
- The disk space should be utilized efficiently and files can access quickly
- **Contiguous allocation** – each file occupies set of contiguous blocks
  - Best performance in most cases
  - Simple – only starting location (block #) and length (number of blocks) are required
  - Problems include finding space for file, knowing file size, external fragmentation (suppose a file is deleted and in its place a new file to be inserted, due to lack of space it may not be possible), need for **compaction off-line** (**downtime**) or **on-line**

By Ajay Kumar Jena, KIIT Deemed to be University, Bhubaneswar

Contiguous Allocation

- Mapping from logical to physical



directory

file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

Block to be accessed = Q + starting address
Displacement into block = R

LectureNotes.in

LectureNotes.in

By Ajay Kumar Jena, KIIT Deemed to be University, Bhubaneswar

Extent-Based Systems

- Many newer file systems (i.e., Veritas File System) use a modified contiguous allocation scheme
- Extent-based file systems allocate disk blocks in extents
- An **extent** is a contiguous block of disks
  - Extents are allocated for file allocation
  - A file consists of one or more extents

LectureNotes.in

By Ajay Kumar Jena, KIIT Deemed to be University, Bhubaneswar

LectureNotes.in

Allocation Methods – Linked Non-contiguous

- **Linked allocation** – each file a linked list of blocks
  - File ends at nil pointer
  - No external fragmentation
  - Each block contains pointer to next block
  - No compaction, external fragmentation
  - Free space management system called when new block needed
  - Improve efficiency by clustering blocks into groups but increases internal fragmentation
  - Reliability can be a problem
  - Locating a block can take many I/Os and disk seeks

LectureNotes.in

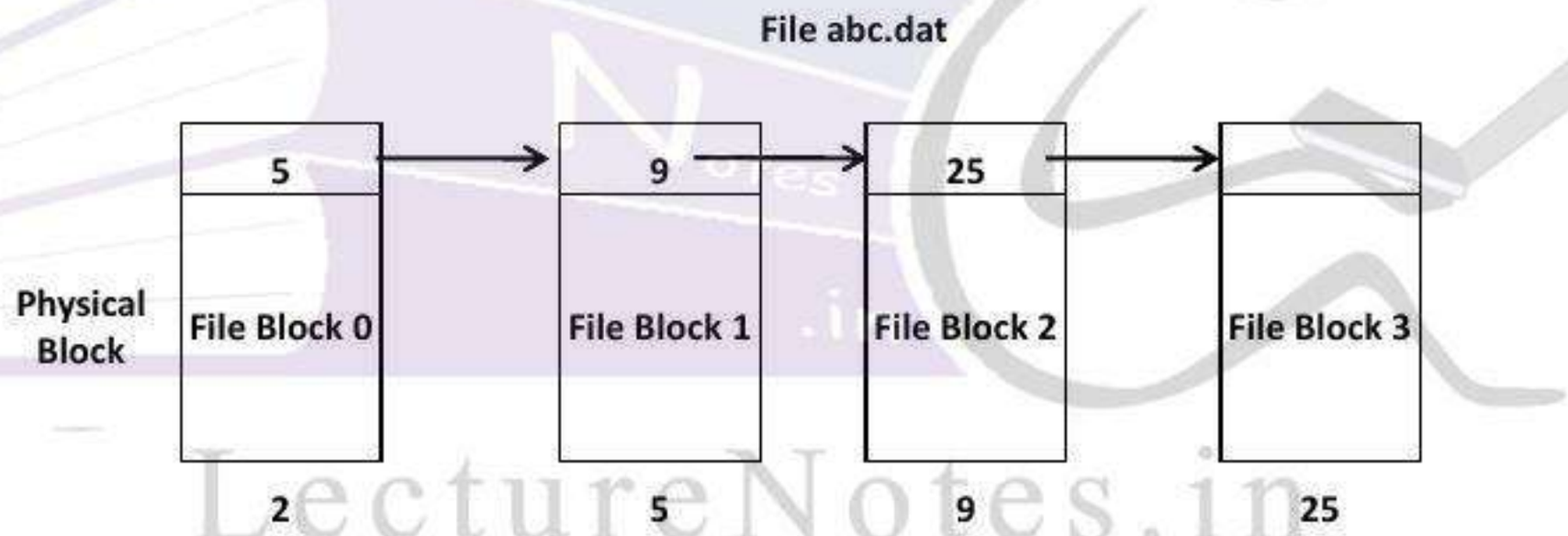
LectureNotes.in

By Ajay Kumar Jena, KIIT Deemed to be University, Bhubaneswar

LectureNotes.in

Allocation Methods – Linked (Cont.)

- FAT (File Allocation Table) variation
  - Beginning of volume has table, indexed by block number
  - Much like a linked list, but faster on disk and cacheable
  - New block allocation simple



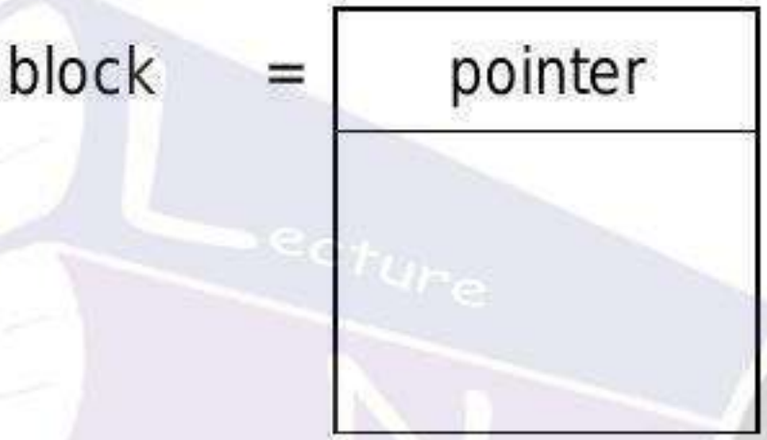
Unlike contiguous allocation every disk block can be used
No space is lost due to fragmentation
External fragmentation avoided
Disadvt: Random access is very slow
Pointer also takes some space

LectureNotes.in
LectureNotes.in

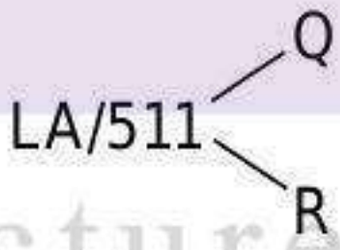
LectureNotes.in

Linked Allocation

- Each file is a linked list of disk blocks: blocks may be scattered anywhere on the disk



- Mapping



Block to be accessed is the Qth block in the linked chain of blocks representing the file.

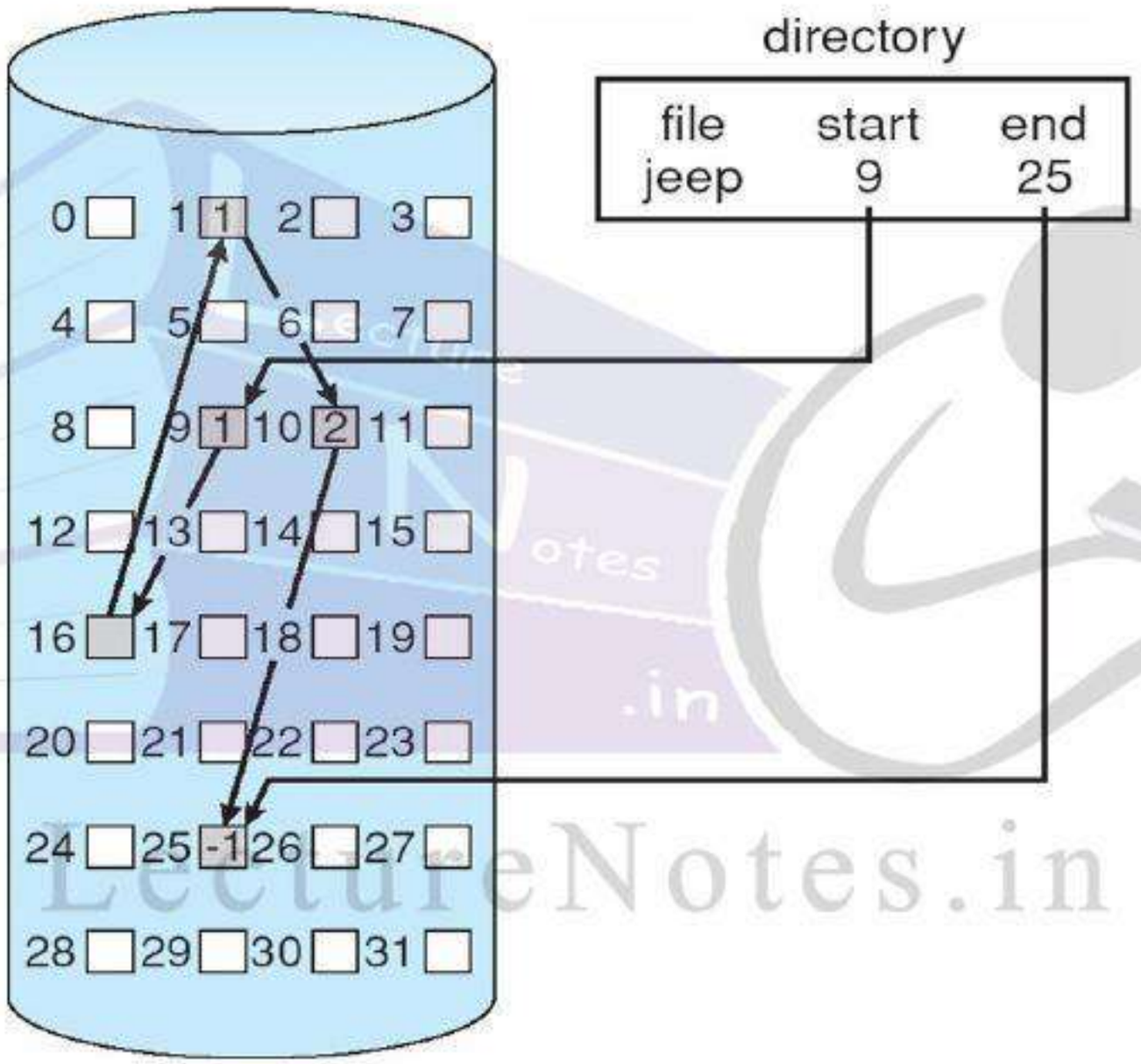
Displacement into block = R + 1

LectureNotes.in

By Ajay Kumar Jena, KIIT Deemed to be University, Bhubaneswar

LectureNotes.in

Linked Allocation

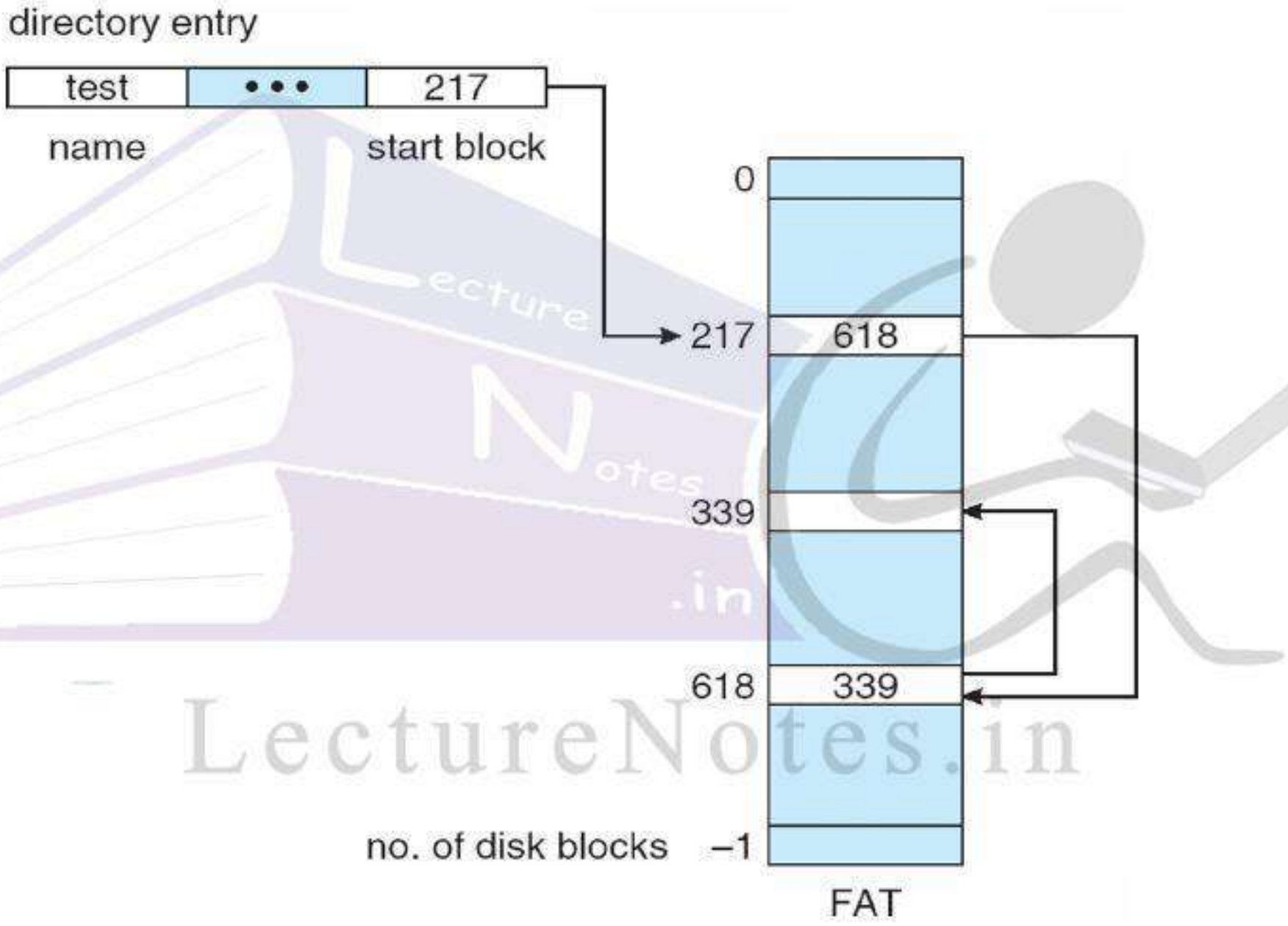


LectureNotes.in

LectureNotes.in

By Ajay Kumar Jena, KIIT Deemed to be University, Bhubaneswar

File-Allocation Table



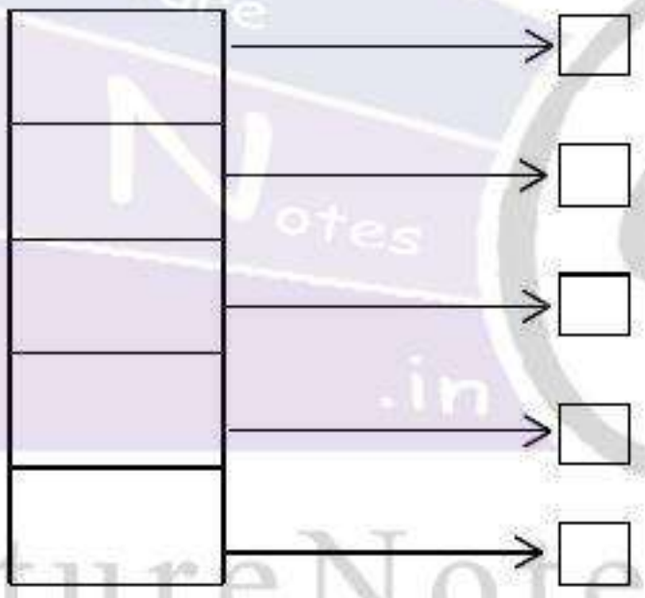
LectureNotes.in

By Ajay Kumar Jena, KIIT Deemed to be University, Bhubaneswar

Allocation Methods - Indexed

- **Indexed allocation**
  - Each file has its own **index block**(s) of pointers to its data blocks

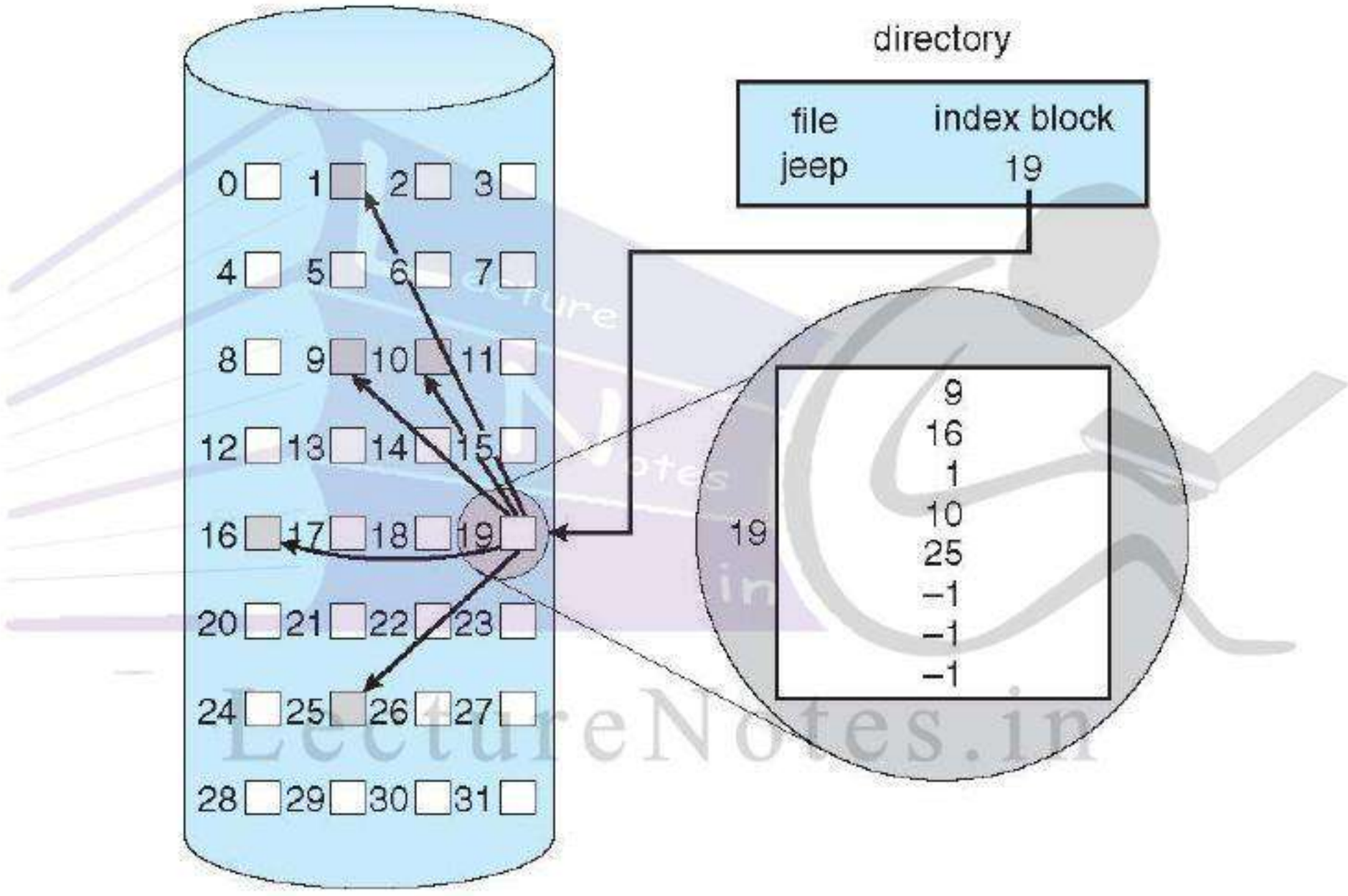
- Logical view



LectureNotes.in

By Ajay Kumar Jena, KIIT Deemed to be University, Bhubaneswar

Example of Indexed Allocation

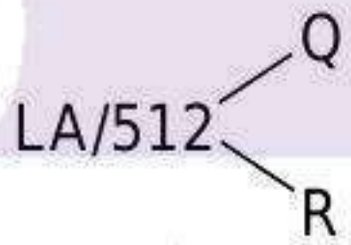


LectureNotes.in

By Ajay Kumar Jena, KIIT Deemed to be University, Bhubaneswar

Indexed Allocation (Cont.)

- Need index table
- Random access
- Dynamic access without external fragmentation, but have overhead of index block
- Mapping from logical to physical in a file of maximum size of 256K bytes and block size of 512 bytes. We need only 1 block for index table



Q = displacement into index table
R = displacement into block

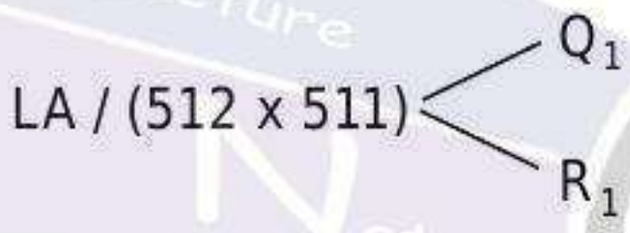
LectureNotes.in

By Ajay Kumar Jena, KIIT Deemed to be University, Bhubaneswar

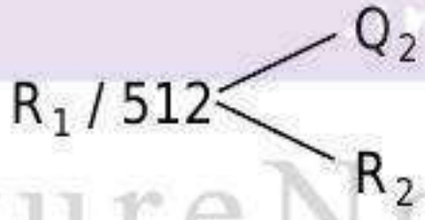
Indexed Allocation – Mapping (Cont.)

- Mapping from logical to physical in a file of unbounded length (block size of 512 words)

- Linked scheme – Link blocks of index table (no limit on size)



Q_1 = block of index table
R_1 is used as follows:



Q_2 = displacement into block of index table
R_2 displacement into block of file:

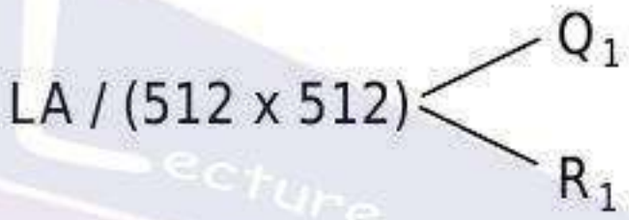
LectureNotes.in

LectureNotes.in

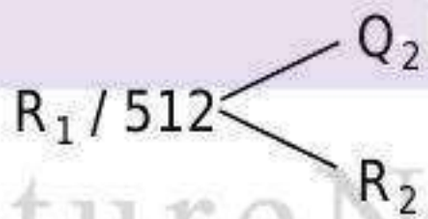
By Ajay Kumar Jena, KIIT Deemed to be University, Bhubaneswar

Indexed Allocation – Mapping (Cont.)

- Two-level index (4K blocks could store 1,024 four-byte pointers in outer index -> 1,048,567 data blocks and file size of up to 4GB)



Q_1 = displacement into outer-index
R_1 is used as follows:



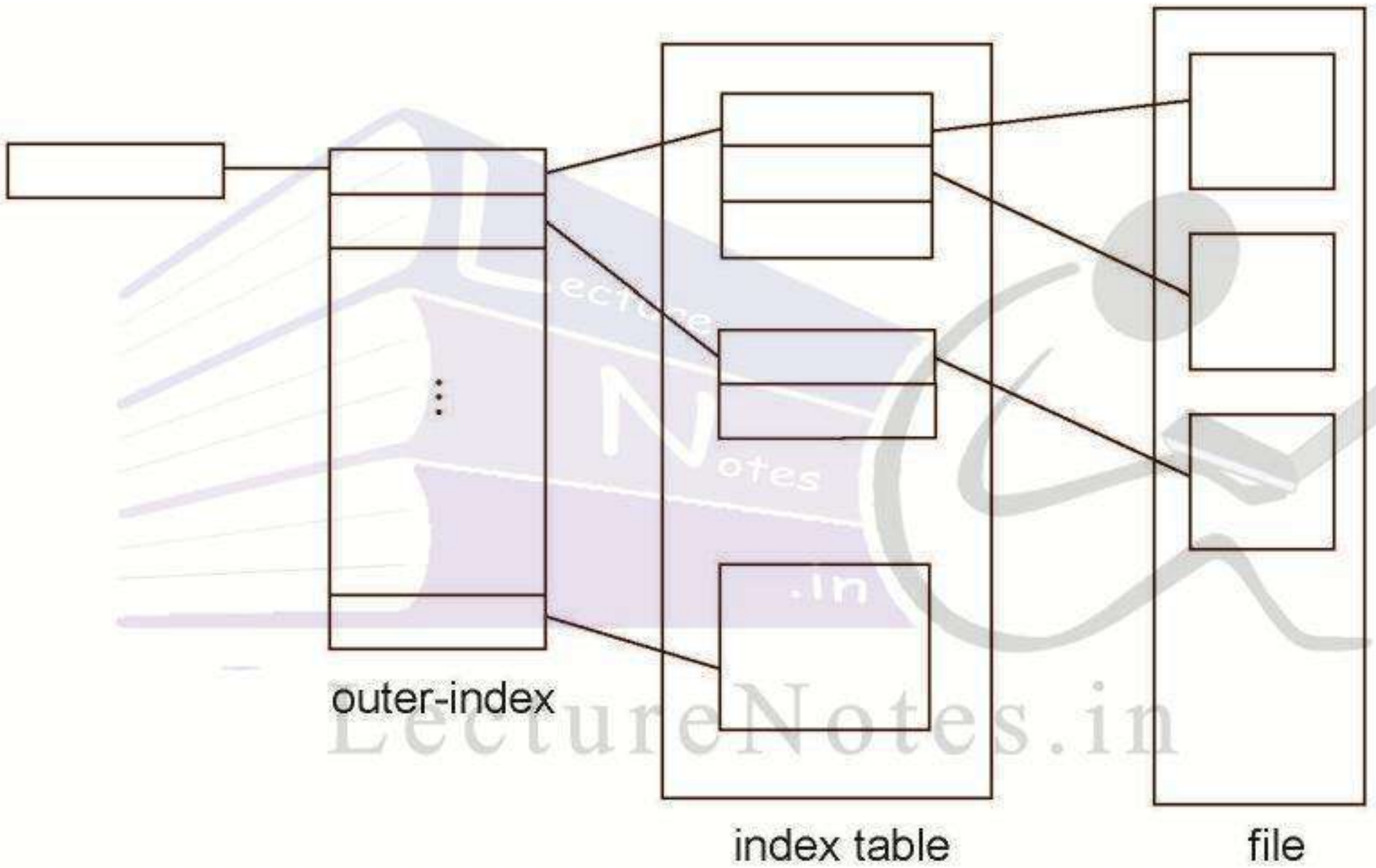
Q_2 = displacement into block of index table
R_2 displacement into block of file:

LectureNotes.in

LectureNotes.in

By Ajay Kumar Jena, KIIT Deemed to be University, Bhubaneswar

Indexed Allocation – Mapping (Cont.)

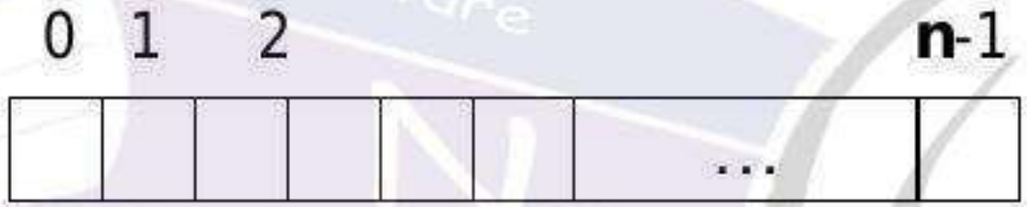


LectureNotes.in

By Ajay Kumar Jena, KIIT Deemed to be University, Bhubaneswar

Free-Space Management

- The main responsibility of the file system is to allocate block to each file and to keep track. Other responsibility is to know how many free space is there.
- File system maintains **free-space list** to track available blocks/clusters
  - (Using term "block" for simplicity)
- **Bit vector** or **bit map** (***n*** blocks) (for every block a bit is maintained)



$$bit[i] = \begin{cases} 1 \Rightarrow block[i] \text{ free} \\ 0 \Rightarrow block[i] \text{ occupied} \end{cases}$$

Block number calculation

(number of bits per word) *
(number of 0-value words) +
offset of first 1 bit

CPUs have instructions to return offset within word of first "1" bit

LectureNotes.in
By Ajay Kumar Jena, KIIT Deemed to be University, Bhubaneswar

LectureNotes.in

Free-Space Management (Cont.)

- Bit map requires extra space
  - Example:

    block size = 4KB = 2^{12} bytes

    disk size = 2^{40} bytes (1 terabyte)

    $n = 2^{40}/2^{12} = 2^{28}$ bits (or 32MB)

    if clusters of 4 blocks -> 8MB of memory
- Easy to get contiguous files

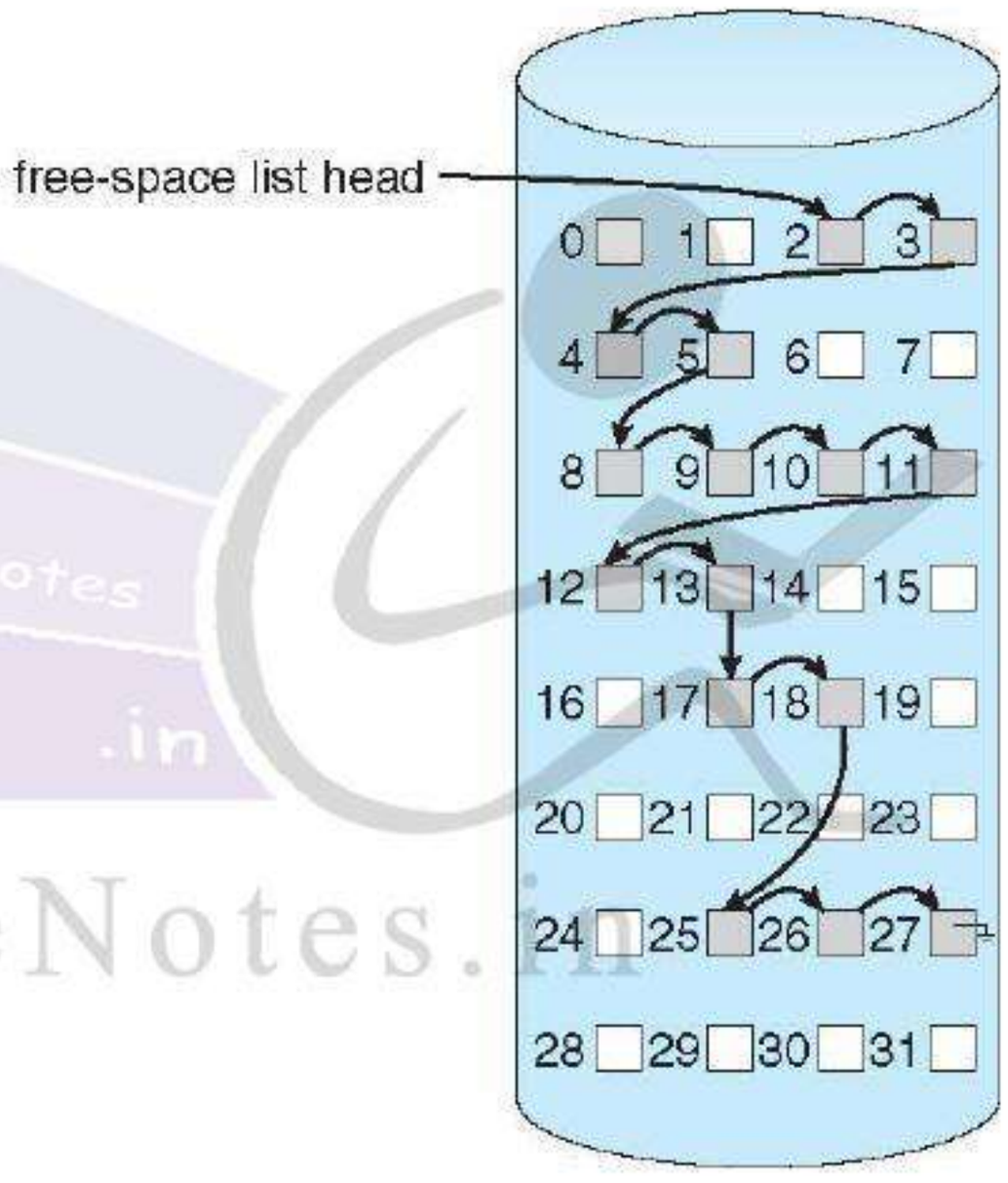
LectureNotes.in

LectureNotes.in

By Ajay Kumar Jena, KIIT Deemed to be University, Bhubaneswar

Linked Free Space List on Disk

- Linked list (free list)
  - Cannot get contiguous space easily
  - No waste of space
  - No need to traverse the entire list (if # free blocks recorded)



LectureNotes.in

By Ajay Kumar Jena, KIIT Deemed to be University, Bhubaneswar

LectureNotes.in

Free-Space Management (Cont.)

- Grouping
  - Modify linked list to store address of next *n-1* free blocks in first free block, plus a pointer to next block that contains free-block-pointers (like this one)
- Counting
  - Because space is frequently contiguously used and freed, with contiguous-allocation allocation, extents, or clustering
    - Keep address of first free block and count of following free blocks
    - Free space list then has entries containing addresses and counts

LectureNotes.in

LectureNotes.in

By Ajay Kumar Jena, KIIT Deemed to be University, Bhubaneswar

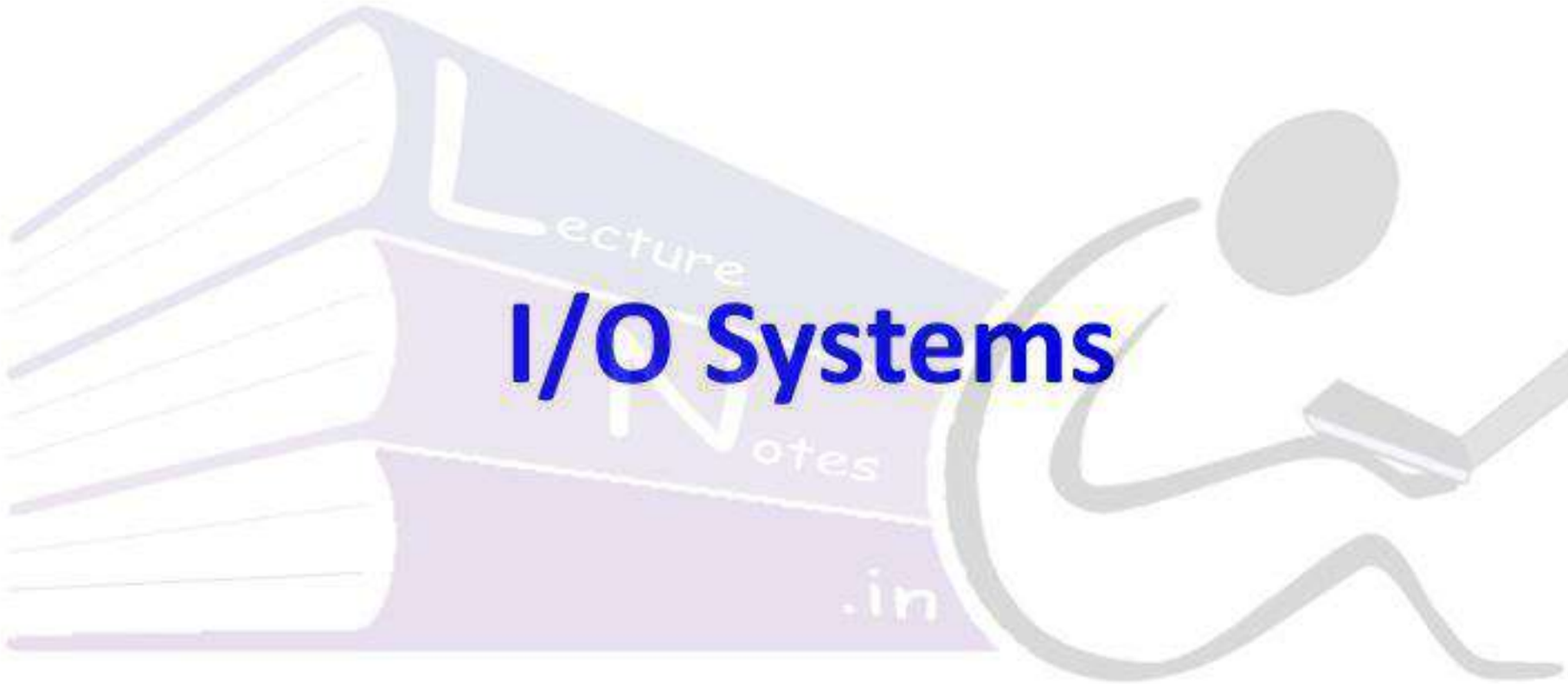
Free-Space Management (Cont.)

- Space Maps
  - Used in **ZFS**
  - Consider meta-data I/O on very large file systems
    - Full data structures like bit maps couldn't fit in memory -> thousands of I/Os
  - Divides device space into **metaslab** units and manages metaslabs
    - Given volume can contain hundreds of metaslabs
  - Each metaslab has associated space map
    - Uses counting algorithm
  - But records to log file rather than file system
    - Log of all block activity, in time order, in counting format
  - Metaslab activity -> load space map into memory in balanced-tree structure, indexed by offset
    - Replay log into that structure
    - Combine contiguous free blocks into single entry

LectureNotes.in

By Ajay Kumar Jena, KIIT Deemed to be University, Bhubaneswar

LectureNotes.in



I/O Systems

LectureNotes.in

LectureNotes.in

By Ajay Kumar Jena, KIIT Deemed to be University, Bhubaneswar

LectureNotes.in

Overview

- I/O management is a major component of operating system design and operation
- It transfers the information between the internal storage (CPU, MM) and the External storage (HDD)
  - Important aspect of computer operation
  - I/O devices vary greatly
  - Various methods to control them
  - Performance management
  - New types of devices frequent
- Ports, busses, device controllers connect to various devices
- **Device drivers** encapsulate device details
  - Present uniform device-access interface to I/O subsystem

LectureNotes.in

LectureNotes.in

By Ajay Kumar Jena, KIIT Deemed to be University, Bhubaneswar

LectureNotes.in

- Computer operates on a great many kind of devices
  - General category storage device: HDD, tapes
  - Transmission devices: N/W card, modem, router etc.
  - Human interface devices: KB, Mouse, screen etc.
- A device communicates with the system by sending signals over a cable or even through air
- The device communicates with the machine via a connection point (port). Serial, Parallel, USB
- A bus is a set of wires and a rigidly defined protocol that specifies a set of messages that can be sent on wires. Buses are used widely in computer architecture.
- **Device drivers** encapsulate device details
  - Present uniform device-access interface to I/O subsystem

LectureNotes.in

By Ajay Kumar Jena, KIIT Deemed to be University, Bhubaneswar

LectureNotes.in

I/O Hardware

- Incredible variety of I/O devices
  - Storage Transmission Human-interface
- Common concepts – signals from I/O devices interface with computer for which special communication tracks are necessary.
  - **Port** – connection point for device

    The device communicates with the machine via a communication point (port).
  - **Bus** – It is a set of wires and a rigidly defined protocol that specifies a set of messages that can be sent on wires. Buses are used widely in computer architecture.
    - **PCI** bus common in PCs and servers, PCI Express (**PCIe**)
    - **expansion bus** connects relatively slow devices
  - **Controller** (**host adapter**) – electronics that operate port, bus, device
    - Sometimes integrated
    - Sometimes separate circuit board (host adapter)
    - Contains processor, microcode, private memory, bus controller, etc
      - Some talk to per-device controller with bus controller, microcode, memory, etc

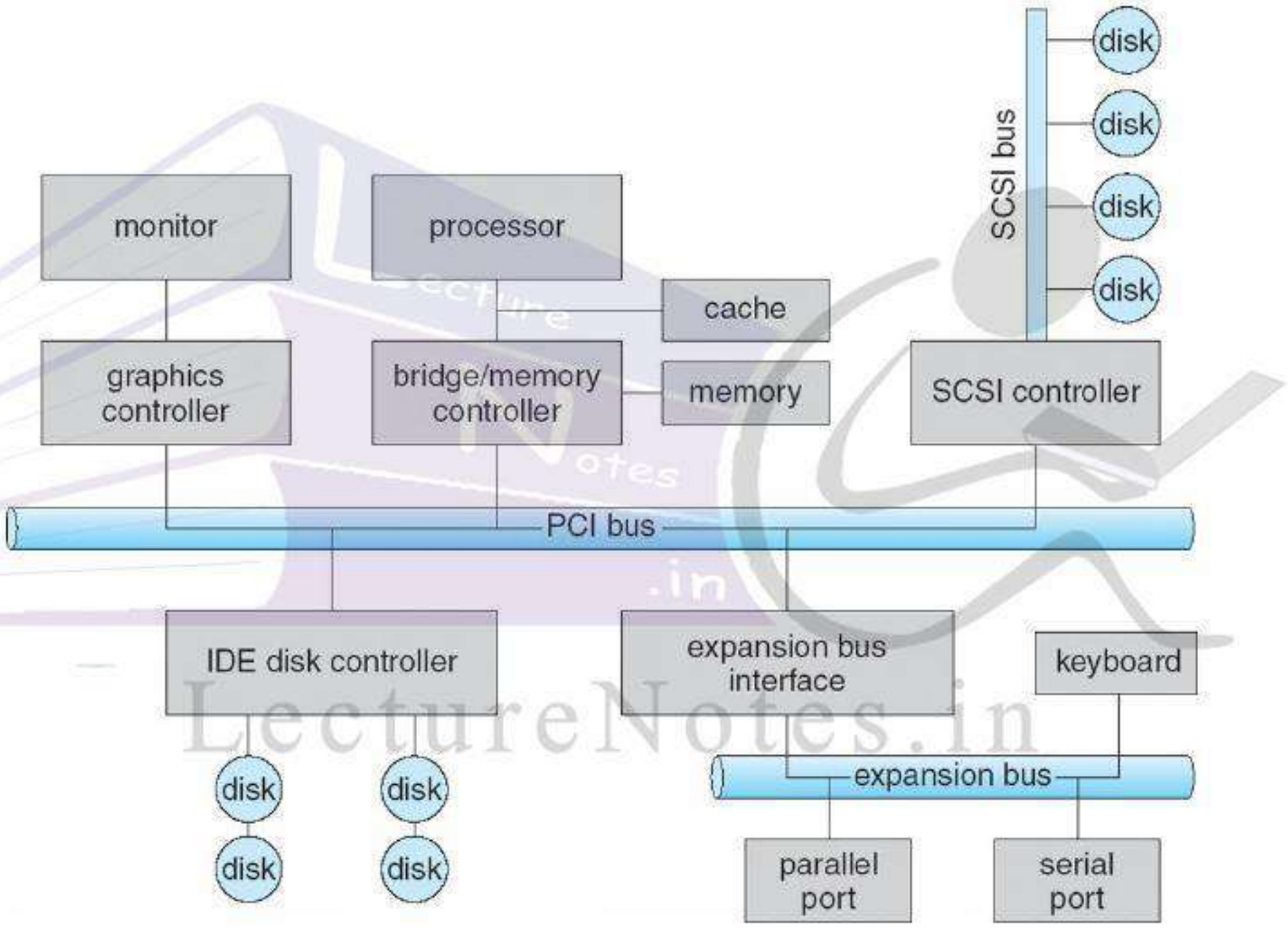
LectureNotes.in

LectureNotes.in

By Ajay Kumar Jena, KIIT Deemed to be University, Bhubaneswar

LectureNotes.in

A Typical PC Bus Structure



A typical Peripheral Component Interconnection Structure

By Ajay Kumar Jena, KIIT Deemed to be University, Bhubaneswar

LectureNotes.in

LectureNotes.in

I/O Hardware (Cont.)

- I/O instructions control devices
- Devices usually have registers where device driver places commands, addresses, and data to write, or read data from registers after command execution
  - Data-in register, data-out register, status register, control register
  - Typically 1-4 bytes, or FIFO buffer
- Devices have addresses, used by
  - Direct I/O instructions
  - **Memory-mapped I/O**
    - Device data and command registers mapped to processor address space
    - Especially for large address spaces (graphics)

LectureNotes.in

LectureNotes.in

By Ajay Kumar Jena, KIIT Deemed to be University, Bhubaneswar

LectureNotes.in

- I/O devices have two components.
  - Mechanical component (the device itself i.e. mouse, KB)
  - Electronic component of devices which controls the device is called device controller. The circuit inside the KB and mouse is the controller)
  - Tasks of controller
    - It converts serial bit stream to block of bytes
    - Performs error correction if necessary
- Ports, busses, device controllers connect to various devices
- **Device drivers** encapsulate device details
  - Present uniform device-access interface to I/O subsystem

LectureNotes.in

LectureNotes.in

By Ajay Kumar Jena, KIIT Deemed to be University, Bhubaneswar

LectureNotes.in

Device I/O Port Locations on PCs (partial)

I/O address range (hexadecimal)	device
000–00F	DMA controller
020–021	interrupt controller
040–043	timer
200–20F	game controller
2F8–2FF	serial port (secondary)
320–32F	hard-disk controller
378–37F	parallel port
3D0–3DF	graphics controller
3F0–3F7	diskette-drive controller
3F8–3FF	serial port (primary)

LectureNotes.in

LectureNotes.in

By Ajay Kumar Jena, KIIT Deemed to be University, Bhubaneswar

LectureNotes.in

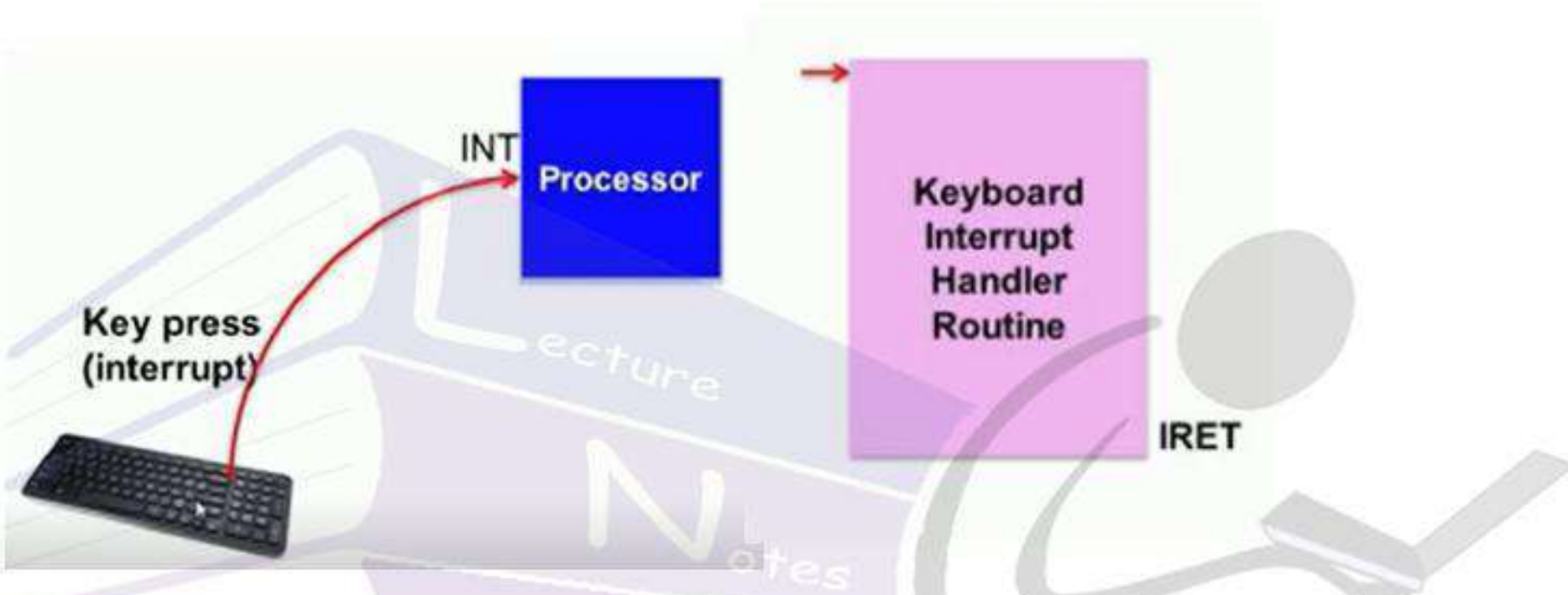
Interrupts

- Data transfer between the I/O and the system can be done
  - Programmed I/O
    - CPU has to wait for the I/O module which is a disadvantage
  - Interrupt controlled I/O
    - CPU does not wait for I/O. When I/O ready it interrupts CPU
  - DMA
- The H/w mechanism that enables a device to notify the CPU is called as interrupt.
- CPU **Interrupt-request line** triggered by I/O device
  - Checked by processor after each instruction
- **Interrupt handler** receives interrupts
  - **Maskable** to ignore or delay some interrupts
- **Interrupt vector** to dispatch interrupt to correct handler
  - Context switch at start and end
  - Based on priority
  - Some **nonmaskable**
  - Interrupt chaining if more than one device at same interrupt number

LectureNotes.in
By Ajay Kumar Jena, KIIT Deemed to be University, Bhubaneswar

LectureNotes.in

HARDWARE INTERRUPTS



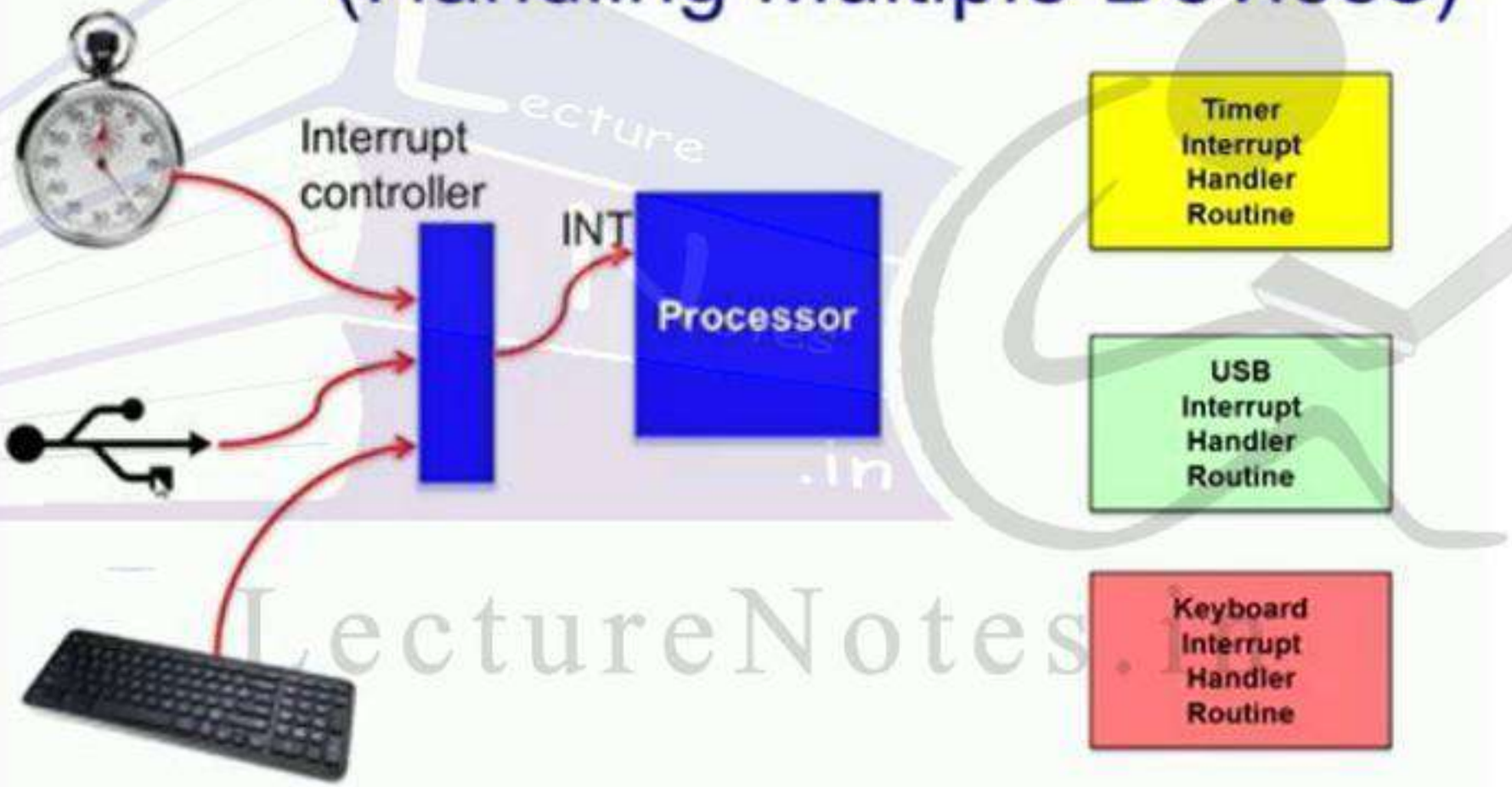
Now-a-days processors have dedicated pins known as INT pin or INTR pin. Devices like KB connects to the processor by INT pin. When a key is pressed in the KB an interrupt is generated in the CPU. The KB interrupt handler routine will be invoked.

LectureNotes.in

By Ajay Kumar Jena, KIIT Deemed to be University, Bhubaneswar

LectureNotes.in

Hardware Interrupts
(Handling Multiple Devices)

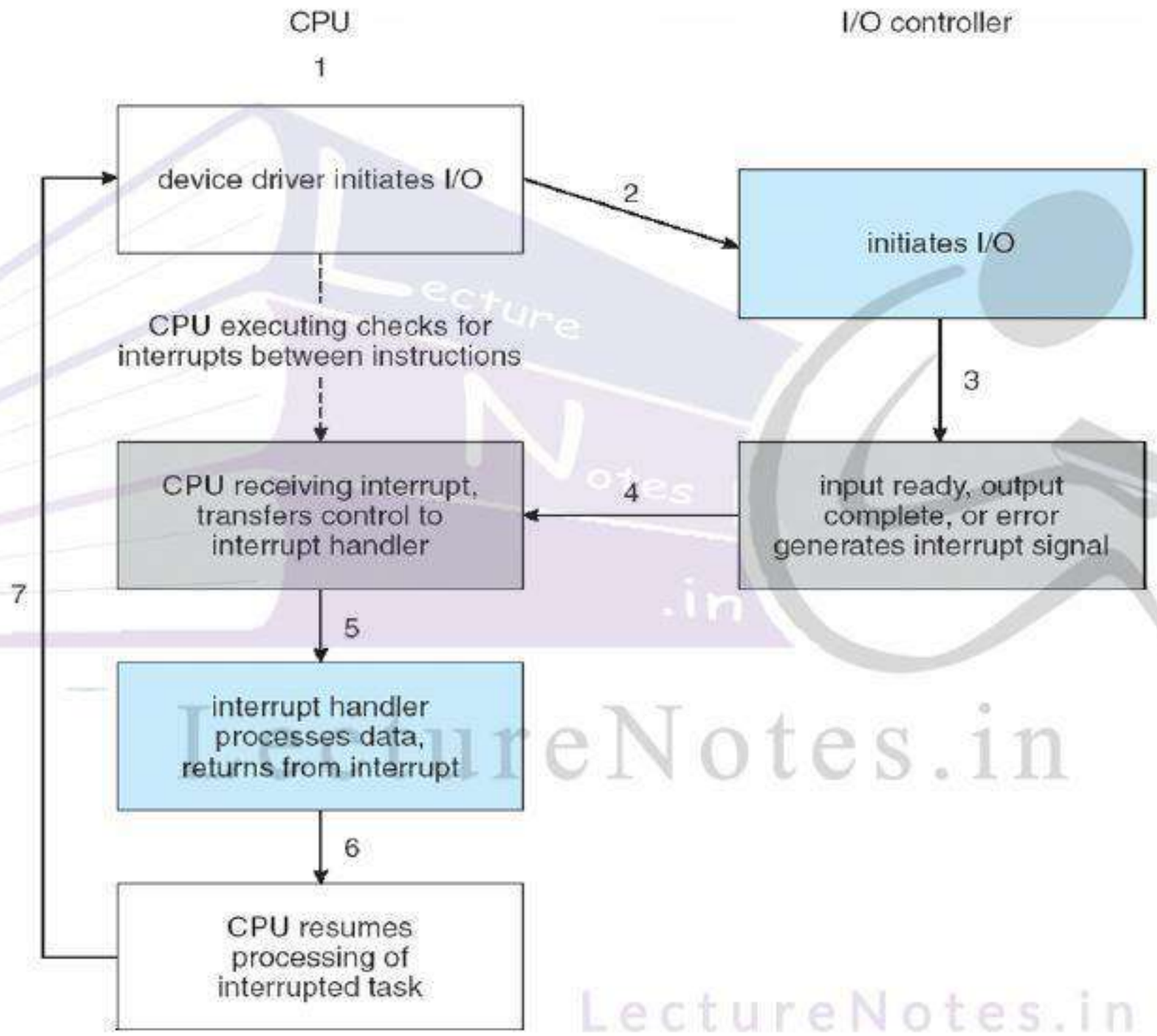


LectureNotes.in

By Ajay Kumar Jena, KIIT Deemed to be University, Bhubaneswar

LectureNotes.in

Interrupt-Driven I/O Cycle



LectureNotes.in
LectureNotes.in

By Ajay Kumar Jena, KIIT Deemed to be University, Bhubaneswar

LectureNotes.in

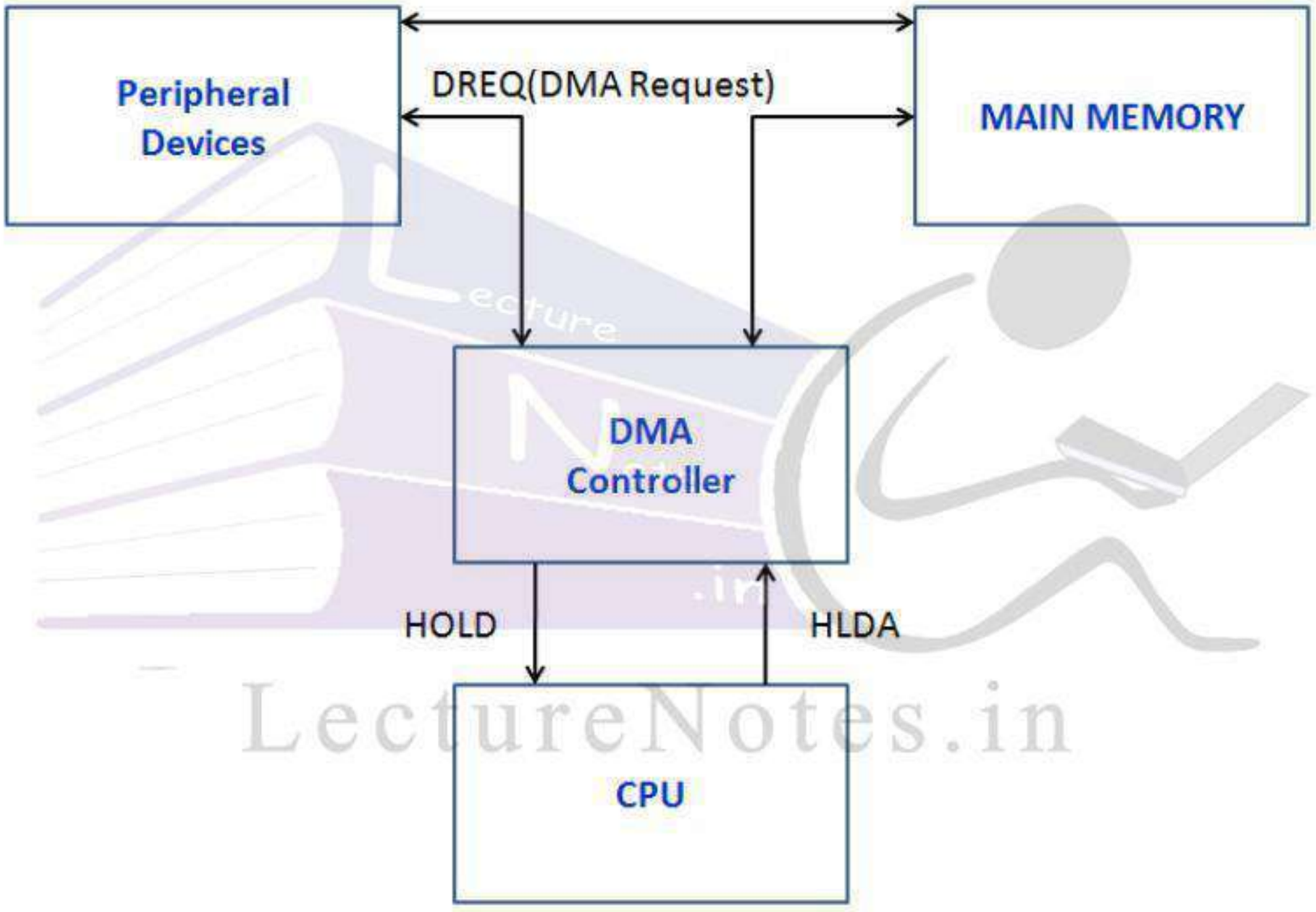
Direct Memory Access

- Used to avoid **programmed I/O** (one byte at a time) for large data movement
- The IO devices will not be synchronized with CPU for transferring the data as IO devices are very slow.
- Requires **DMA** controller is having four parameters i.e. source address, destination address, no of blocks, Read/Write
- Bypasses CPU to transfer data directly between I/O device and memory
- OS writes DMA command block into memory
  - Source and destination addresses
  - Read or write mode
  - Count of bytes
  - Writes location of command block to DMA controller
  - Bus mastering of DMA controller – grabs bus from CPU
    - **Cycle stealing** from CPU but still much more efficient
  - When done, interrupts to signal completion

LectureNotes.in

By Ajay Kumar Jena, KIIT Deemed to be University, Bhubaneswar

LectureNotes.in

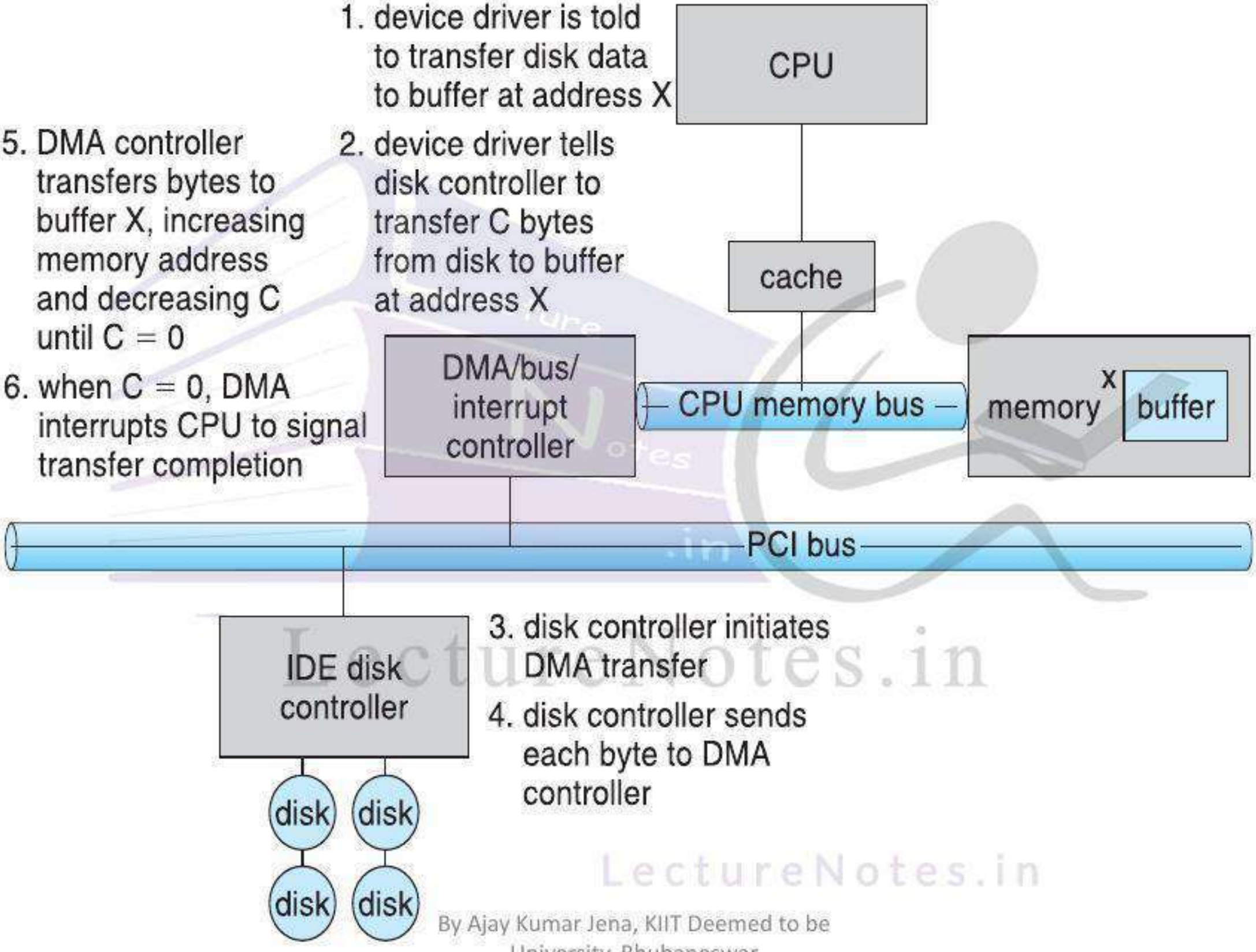


LectureNotes.in
LectureNotes.in

By Ajay Kumar Jena, KIIT Deemed to be University, Bhubaneswar

Six Step Process to Perform DMA Transfer

1. device driver is told to transfer disk data to buffer at address X
2. device driver tells disk controller to transfer C bytes from disk to buffer at address X
3. disk controller initiates DMA transfer
4. disk controller sends each byte to DMA controller
5. DMA controller transfers bytes to buffer X, increasing memory address and decreasing C until C = 0
6. when C = 0, DMA interrupts CPU to signal transfer completion



LectureNotes.in
By Ajay Kumar Jena, KIIT Deemed to be University, Bhubaneswar

LectureNotes.in

Application I/O Interface

- I/O system calls encapsulate device behaviors in generic classes
- Device-driver layer hides differences among I/O controllers from kernel
- New devices talking already-implemented protocols need no extra work
- Each OS has its own I/O subsystem structures and device driver frameworks
- Devices vary in many dimensions
  - **Character-stream** or **block**
  - **Sequential** or **random-access**
  - **Synchronous** or **asynchronous** (or both)
  - **Sharable** or **dedicated**
  - **Speed of operation**
  - **read-write, read only,** or **write only**

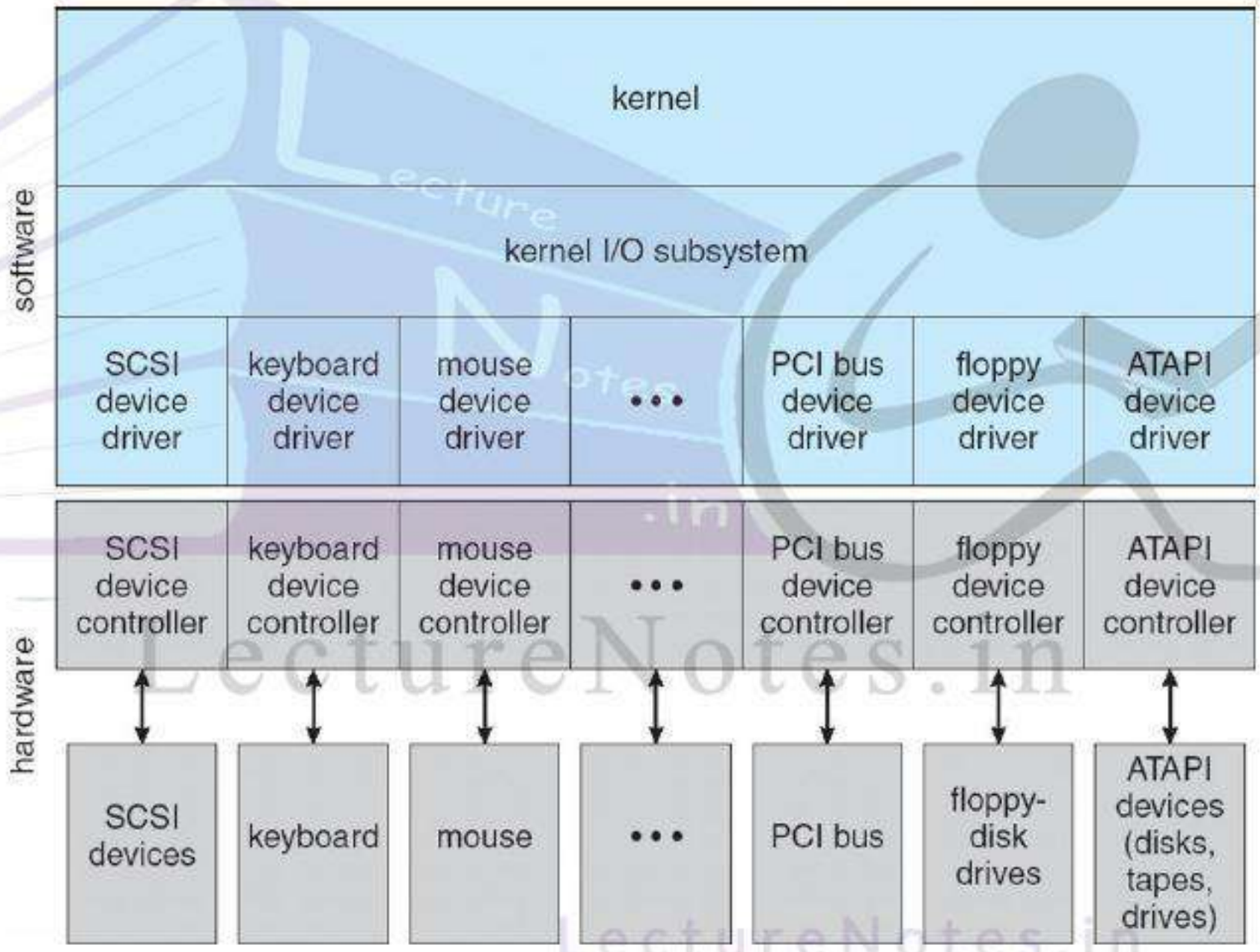
LectureNotes.in

LectureNotes.in

By Ajay Kumar Jena, KIIT Deemed to be University, Bhubaneswar

LectureNotes.in

A Kernel I/O Structure



By Ajay Kumar Jena, KIIT Deemed to be University, Bhubaneswar

LectureNotes.in

Characteristics of I/O Devices

aspect	variation	example
data-transfer mode	character block	terminal disk
access method	sequential random	modem CD-ROM
transfer schedule	synchronous asynchronous	tape keyboard
sharing	dedicated sharable	tape keyboard
device speed	latency seek time transfer rate delay between operations	
I/O direction	read only write only read–write	CD-ROM graphics controller disk

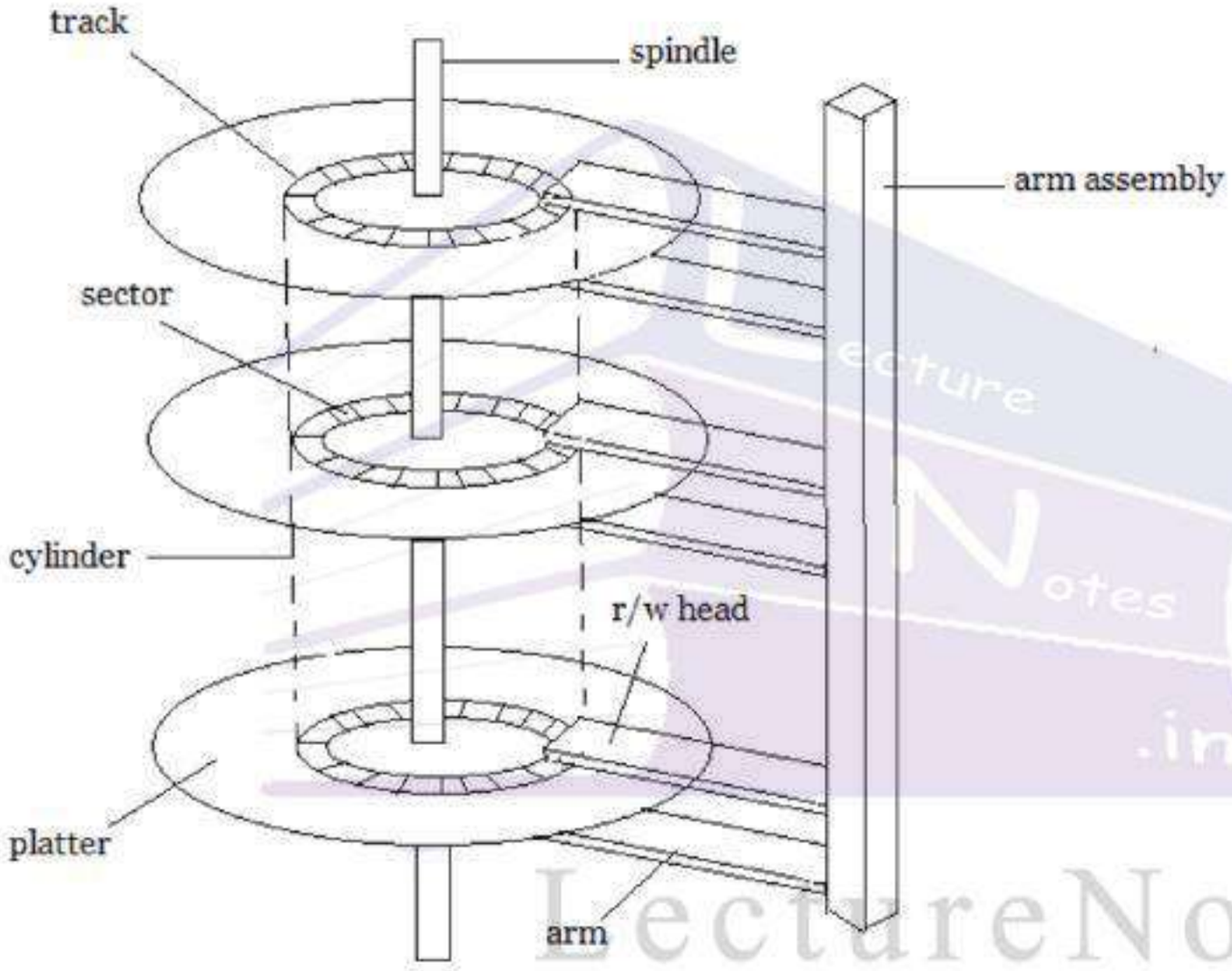
LectureNotes.in

LectureNotes.in

By Ajay Kumar Jena, KIIT Deemed to be University, Bhubaneswar

LectureNotes.in

Disk Structure



Structure of a magnetic disk

- Secondary storage devices are those devices whose memory is non volatile
- Secondary storage is also called auxiliary storage.
- Secondary storage is less expensive when compared to primary memory like RAMs.
- The speed of the secondary storage is also lesser than that of primary storage.
- Hence, the data which is less frequently accessed is kept in the secondary storage.
- A few examples are magnetic disks, magnetic tapes, removable thumb drives etc.

LectureNotes.in

By Ajay Kumar Jena, KIIT Deemed to be University, Bhubaneswar

LectureNotes.in

Disk Scheduling

- A magnetic disk contains several **platters**. Each platter is divided into circular shaped **tracks**.
- The length of the tracks near the centre is less than the length of the tracks farther from the centre.
- Each track is further divided into **sectors**.
- Tracks of the same distance from centre form a cylinder.
- A read-write head is used to read data from a sector of the magnetic disk.
- The speed of the disk is measured as two parts:

- **Transfer rate:** This is the rate at which the data moves from disk to the computer.
- **Random access time:** It is the sum of the seek time and rotational latency.
- **Seek time** is the time taken by the arm to move to the required track.
- **Rotational latency** is defined as the time taken by the arm to reach the required sector in the track.

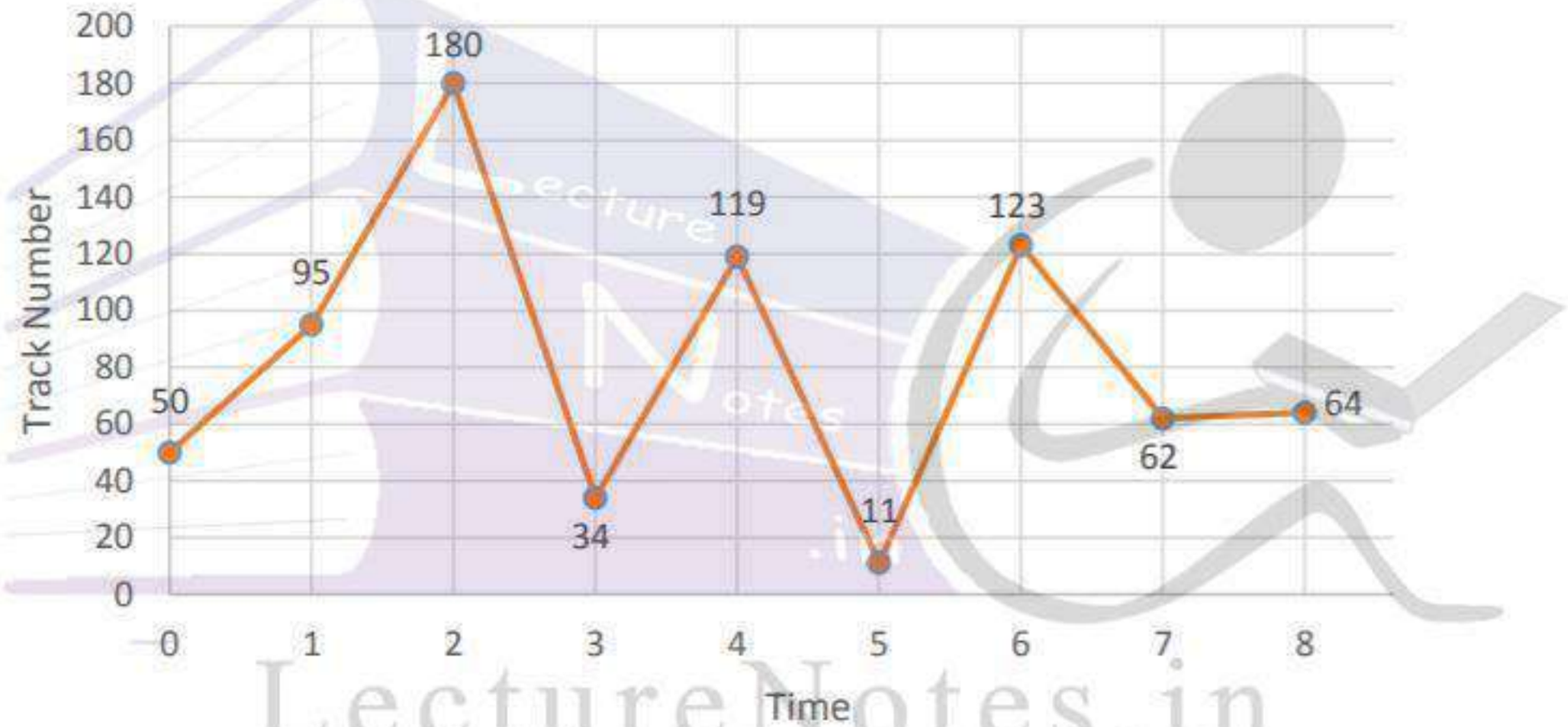
LectureNotes.in

By Ajay Kumar Jena, KIIT Deemed to be University, Bhubaneswar

LectureNotes.in

FCFS Disk scheduling algorithm:

Given the following queue -- 95, 180, 34, 119, 11, 123, 62, 64 with the Read-write head initially at the track 50 and the tail track being at 199.



LectureNotes.in

For this case it went from 50 to 95 to 180 and so on. From 50 to 95 it moved 45 tracks. If you tally up the total number of tracks you will find how many tracks it had to go through before finishing the entire request. The total head movement of 644 tracks.

THM= (180 - 50) + (180-34) + (119-34) + (119-11) + (123-11) + (123-62) + (64-62)

= 130 + 146 + 85 + 108 + 112 + 61 + 2 = 644 tracks

LectureNotes.in
By Ajay Kumar Jena, KIIT Deemed to be University, Bhubaneswar

SSTF (Shortest Seek Time First) Disk scheduling algorithm:

Given the following Request queue – 82, 170, 43, 140, 24,16, 190 with the Read-write head initially at the track 50 and the tail track being at 199.

LectureNotes.in

Here the position which is closest to the current head position is chosen first. The request is serviced according to next shortest distance. Find the shortest seek time. To move the desired track. From 50, the next shortest distance would be 43. From 43 to 24 track. By moving accordingly we have 43,24,16,82,140,170,190.

THM = (50-16)+ (190-16) = 34+174=208 tracks

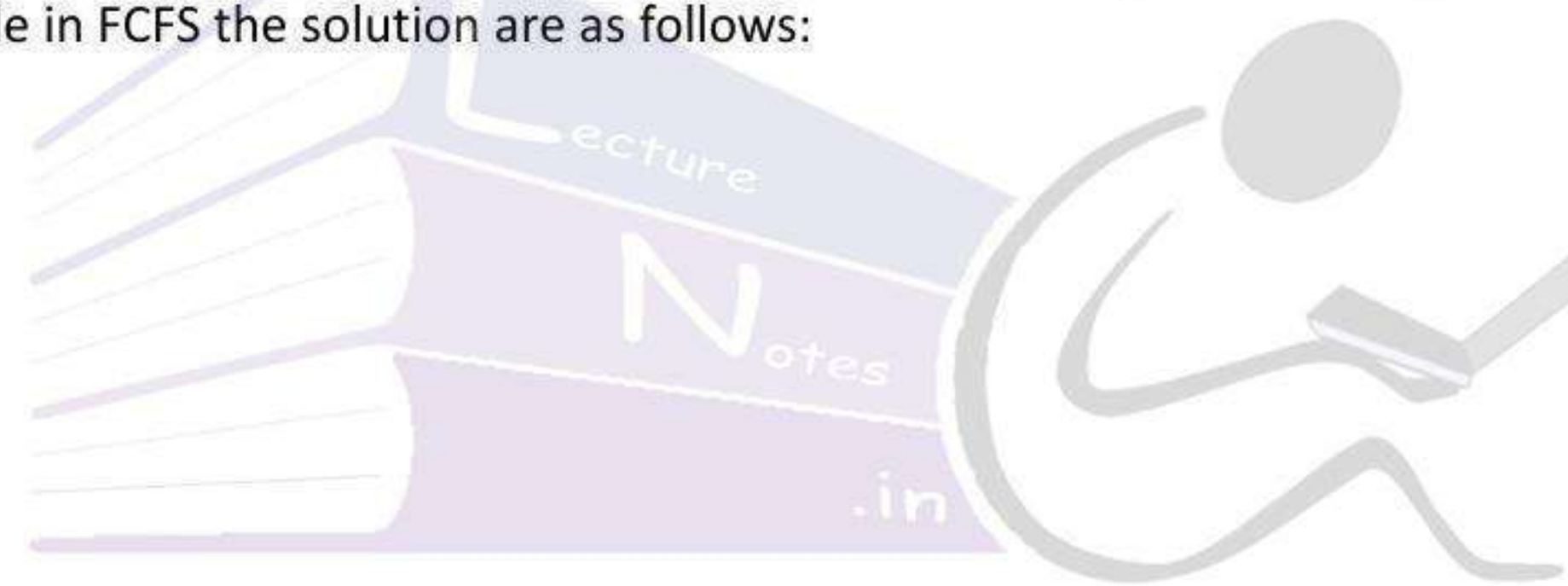
LectureNotes.in

By Ajay Kumar Jena, KIIT Deemed to be University, Bhubaneswar

LectureNotes.in

SCAN Scheduling Algorithm

This algorithm is performed by moving the R/W head back-and-forth to the innermost and outermost track. As it scans the tracks from end to end, it process all the requests found in the direction it is headed. This will ensure that all track requests, whether in the outermost, middle or innermost location, will be traversed by the access arm thereby finding all the requests. This is also known as the Elevator algorithm. Using the same sets of example in FCFS the solution are as follows:



LectureNotes.in

LectureNotes.in

By Ajay Kumar Jena, KIIT Deemed to be University, Bhubaneswar

LectureNotes.in

Protection & Security

LectureNotes.in

LectureNotes.in

LectureNotes.in

Protection

- A computer system is a collection of processes and objects.
- HW objects such as CPU, memory segment, disk, tape drives.
- SW objects like files, programs, semaphores
- Different operations are performed on different objects.
- CPU can only execute, Memory segment read, write tape drives can have read, write, rewound
- Data files can be created, opened, read, written, closed, deleted
- Program files can read, written, executed and deleted.
- A process should be allowed to access only those resources for which it has authorization.
- So the process operates within a protection domain which specifies that the resources the process may access.
- Each domain is defined a set of objects and the type of operations that may be invoked on each objects.
- The ability to execute an operation on the object is an access right.

LectureNotes.in

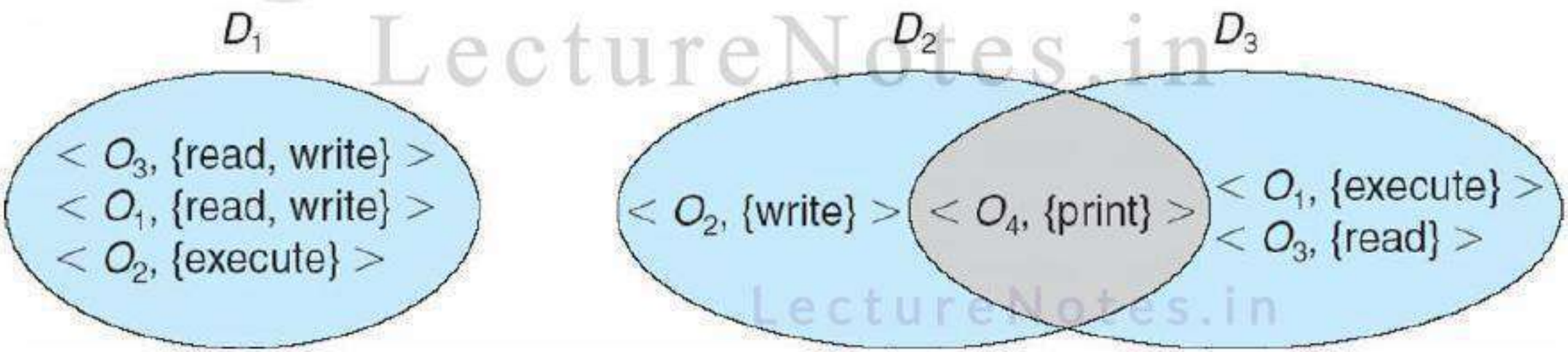
By Ajay Kumar Jena, KIIT Deemed to be University, Bhubaneswar

LectureNotes.in

Domain Structure

- Protection Domain
  - User (user id)
  - Process (process id)
  - Procedures

- Access-right = <*object-name*, *rights-set*> It is a order pair where *rights-set* is a subset of all valid operations that can be performed on the object
- Domain = collection of set of access-rights



LectureNotes.in

LectureNotes.in

Access Matrix

- View protection as a matrix (**access matrix**)
- Rows represent domains (Users/groups/subjects)
- Columns represent objects (Resources)
- **`Access(i, j)`** is the set of operations that a process executing in Domain D_i can invoke on Object O_j. **Access control matrix** defines what access rights user u has for object O.

	object / domain	F_1	F_2	F_3	printer
User1	D_1	read		read	
User2	D_2				print
User3	D_3		read	execute	
User4	D_4	read write		read write	

LectureNotes.in
LectureNotes.in
By Ajay Kumar Jena, KIIT Deemed to be University, Bhubaneswar

LectureNotes.in

Use of Access Matrix

- If a process in Domain D_i tries to do "op" on object O_j, then "op" must be in the access matrix
- User who creates object can define access column for that object
- Can be expanded to dynamic protection
  - Operations to add, delete access rights
  - Special access rights:
    - *owner of* O_i
    - *copy op from* O_i *to* O_j *(denoted by "* "*)*
    - *control –* D_i *can modify* D_j *access rights*
    - *transfer – switch from domain* D_i *to* D_j
  - *Copy* and *Owner* applicable to an object
  - *Control* applicable to domain object

LectureNotes.in

LectureNotes.in

By Ajay Kumar Jena, KIIT Deemed to be University, Bhubaneswar

LectureNotes.in

Use of Access Matrix (Cont.)

- **Access matrix** design separates mechanism from policy
  - Mechanism
    - Operating system provides access-matrix + rules
    - If ensures that the matrix is only manipulated by authorized agents and that rules are strictly enforced
  - Policy
    - User dictates policy
    - Who can access what object and in what mode
- But doesn't solve the general confinement problem

LectureNotes.in

LectureNotes.in

By Ajay Kumar Jena, KIIT Deemed to be University, Bhubaneswar

LectureNotes.in

Access Matrix of Figure A with Domains as Objects

domain \ object	F_1	F_2	F_3	laser printer	D_1	D_2	D_3	D_4
D_1	read		read			switch		
D_2				print			switch	switch
D_3		read	execute					
D_4	read write		read write		switch			

LectureNotes.in

LectureNotes.in

By Ajay Kumar Jena, KIIT Deemed to be University, Bhubaneswar

LectureNotes.in

Access Matrix with *Copy* Rights

object / domain	F_1	F_2	F_3
D_1	execute		write*
D_2	execute	read*	execute
D_3	execute		

(a)

object / domain	F_1	F_2	F_3
D_1	execute		write*
D_2	execute	read*	execute
D_3	execute	read	

(b)

LectureNotes.in
LectureNotes.in
By Ajay Kumar Jena, KIIT Deemed to be University, Bhubaneswar

LectureNotes.in

Access Matrix With *Owner* Rights

object / domain	F_1	F_2	F_3
D_1	owner execute		write
D_2		read* owner	read* owner write
D_3	execute		

(a)

object / domain	F_1	F_2	F_3
D_1	owner execute		write
D_2		owner read* write*	read* owner write
D_3		write	write

(b)

LectureNotes.in

By Ajay Kumar Jena, KIIT Deemed to be University, Bhubaneswar

LectureNotes.in

Modified Access Matrix of Figure B

domain \ object	F_1	F_2	F_3	laser printer	D_1	D_2	D_3	D_4
D_1	read		read			switch		
D_2				print			switch	switch control
D_3		read	execute					
D_4	write		write		switch			

LectureNotes.in

LectureNotes.in

By Ajay Kumar Jena, KIIT Deemed to be University, Bhubaneswar

LectureNotes.in

Security

- The Security Problem
- Program Threats
- System and Network Threats
- Cryptography as a Security Tool
- User Authentication
- Implementing Security Defenses
- Firewalling to Protect Systems and Networks
- Computer-Security Classifications
- An Example: Windows 7

LectureNotes.in

By Ajay Kumar Jena, KIIT Deemed to be University, Bhubaneswar

LectureNotes.in

The Security Problem

- **Threats:** It is the potential security violation
- When a program created by an user is used by another user there may be the chances of misuse.
- Examples: **Trojan horse:** It sits idle and transmits all user information to the attacker.
- Trap doors, stack/buffer overflow
- **System threats:** The OS files or resources are misused.
- Examples: WORMS: Small programs that replicates themselves. It contains malicious codes that cause major damage to the files of OS. They consume maximum resources thus denying the service to the user.
- **VIRUS: Small programs that replicates and modification to the files.**
- **Intruders** (**crackers**) attempt to breach security
- **Threat** is potential security violation
- **Attack** is attempt to breach security
- Attack can be accidental or malicious
- Easier to protect against accidental than malicious misuse

LectureNotes.in

By Ajay Kumar Jena, KIIT Deemed to be University, Bhubaneswar

LectureNotes.in

Security Violation Categories

- **Breach of confidentiality**
  - Unauthorized reading of data
- **Breach of integrity**
  - Unauthorized modification of data
- **Breach of availability**
  - Unauthorized destruction of data
- **Theft of service**
  - Unauthorized use of resources
- **Denial of service (DOS)**
  - Prevention of legitimate use

LectureNotes.in
LectureNotes.in

By Ajay Kumar Jena, KIIT Deemed to be University, Bhubaneswar

LectureNotes.in

Security Violation Methods

- **Masquerading** (breach **authentication**)
  - Pretending to be an authorized user to escalate privileges
- **Replay attack**
  - As is or with **message modification**
- **Man-in-the-middle attack**
  - Intruder sits in data flow, masquerading as sender to receiver and vice versa
- **Session hijacking**
  - Intercept an already-established session to bypass authentication

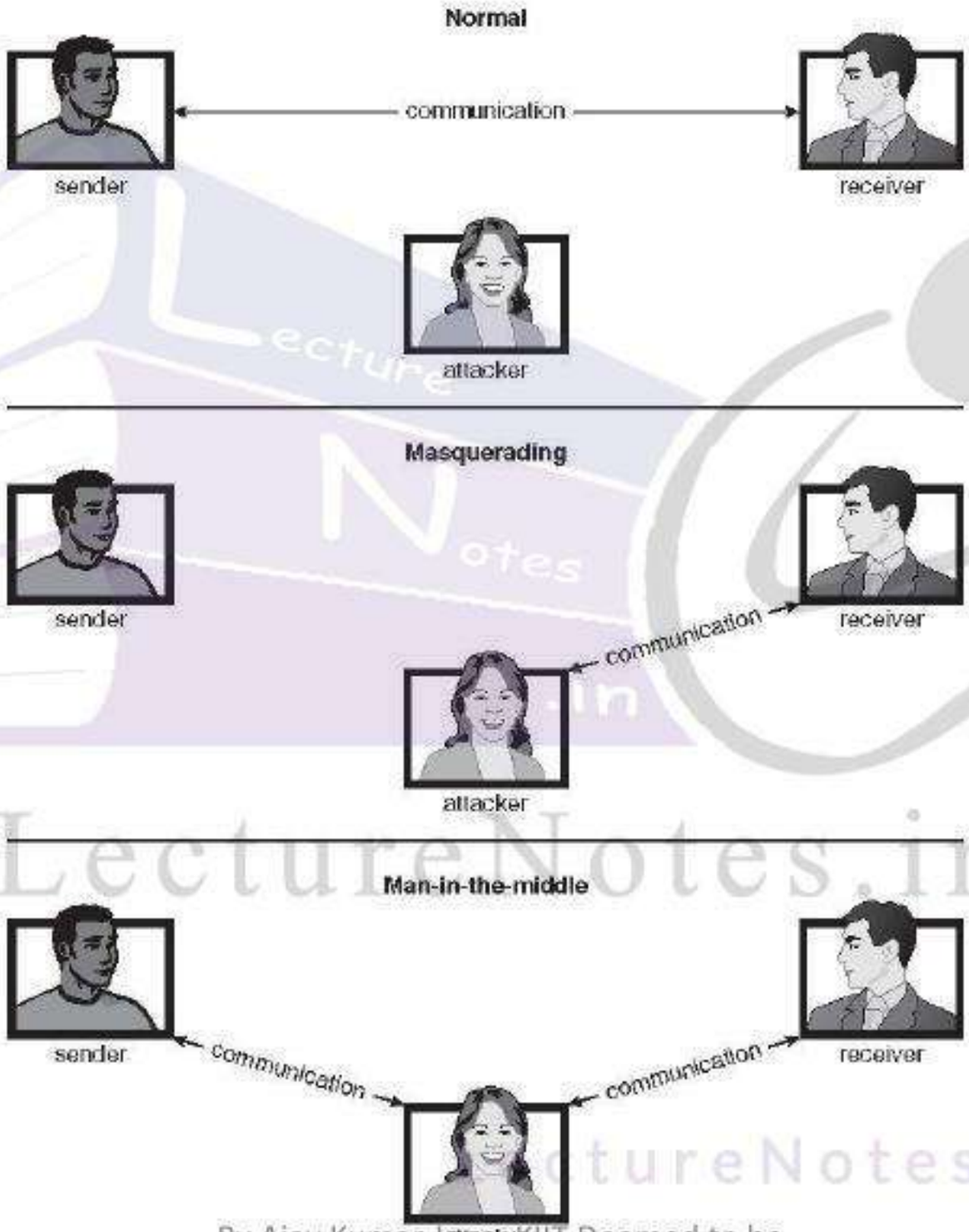
LectureNotes.in

LectureNotes.in

By Ajay Kumar Jena, KIIT Deemed to be University, Bhubaneswar

LectureNotes.in

Standard Security Attacks



LectureNotes.in
LectureNotes.in
By Ajay Kumar Jena, KIIT Deemed to be University, Bhubaneswar

LectureNotes.in

Security Measure Levels

- Impossible to have absolute security, but make cost to perpetrator sufficiently high to deter most intruders
- Security must occur at four levels to be effective:
  - **Physical**
    - Data centers, servers, connected terminals
  - **Human**
    - Avoid **social engineering**, **phishing**, **dumpster diving**
  - **Operating System**
    - Protection mechanisms, debugging
  - **Network**
    - Intercepted communications, interruption, DOS
- Security is as weak as the weakest link in the chain
- But can too much security be a problem?

LectureNotes.in

By Ajay Kumar Jena, KIIT Deemed to be University, Bhubaneswar

LectureNotes.in

Program Threats

- Many variations, many names
- **Trojan Horse**
  - Code segment that misuses its environment
  - Exploits mechanisms for allowing programs written by users to be executed by other users
  - **Spyware, pop-up browser windows, covert channels**
  - Up to 80% of spam delivered by spyware-infected systems
- **Trap Door**
  - Specific user identifier or password that circumvents normal security procedures
  - Could be included in a compiler
  - How to detect them?

LectureNotes.in

LectureNotes.in

By Ajay Kumar Jena, KIIT Deemed to be University, Bhubaneswar

LectureNotes.in

Program Threats (Cont.)

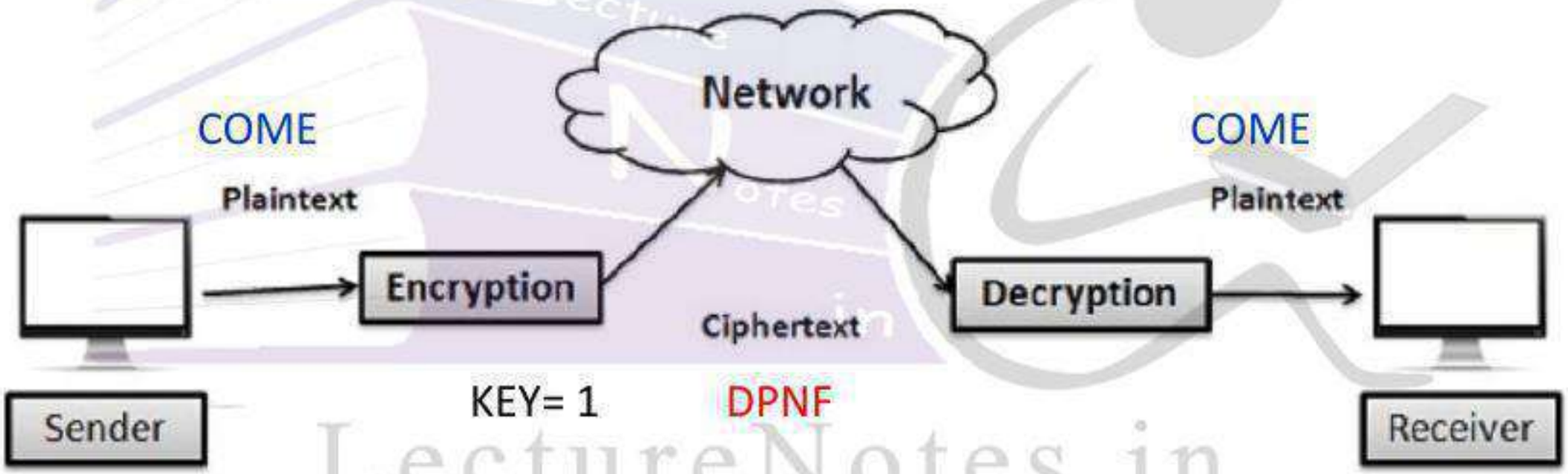
- **Logic Bomb**
  - Program that initiates a security incident under certain circumstances
- **Stack** and **Buffer Overflow**
  - Exploits a bug in a program (overflow either the stack or memory buffers)
  - Failure to check bounds on inputs, arguments
  - Write past arguments on the stack into the return address on stack
  - When routine returns from call, returns to hacked address
    - Pointed to code loaded onto stack that executes malicious code
  - Unauthorized user or privilege escalation

LectureNotes.in

LectureNotes.in

Encryption

- It is a technique that enables the sender of a message to encrypt the message, so that only a legitimate receiver can decrypt and read the message.
- Caesar Cipher



LectureNotes.in

LectureNotes.in

By Ajay Kumar Jena, KIIT Deemed to be University, Bhubaneswar

LectureNotes.in

ACKNOWLEDGEMENT

I am very much thankful to say that some contents of these slides are taken from the book

Operating System Concepts
Silberschatz, Galvin and Gagne.

LectureNotes.in

LectureNotes.in

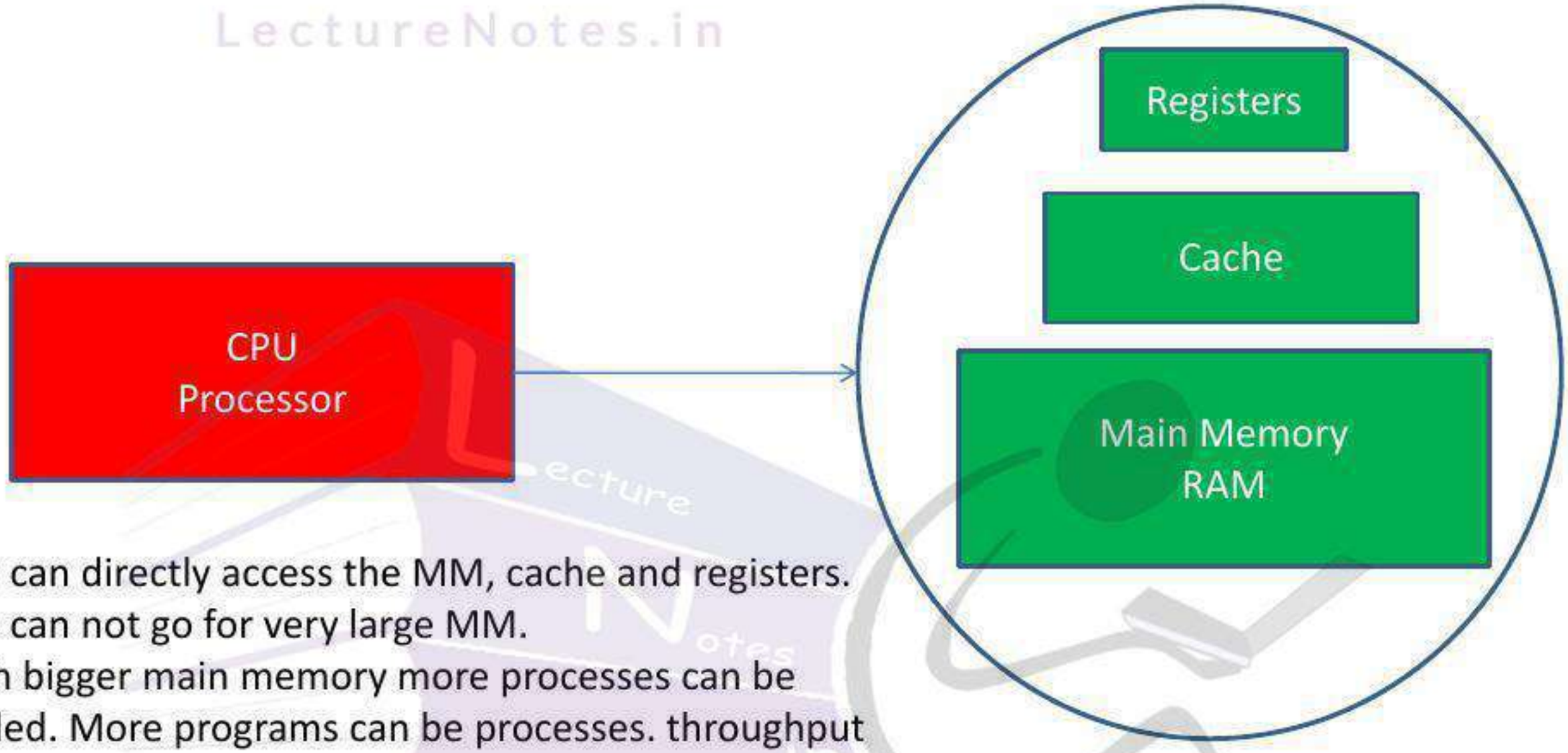
LectureNotes.in

MEMORY MANAGEMENT

Dr. Ajay Kumar Jena
School of Computer Engineering
KIIT University, Bhubaneswar

LectureNotes.in
LectureNotes.in

LectureNotes.in



CPU can directly access the MM, cache and registers.
One can not go for very large MM.
With bigger main memory more processes can be loaded. More programs can be processes. throughput more

LectureNotes.in

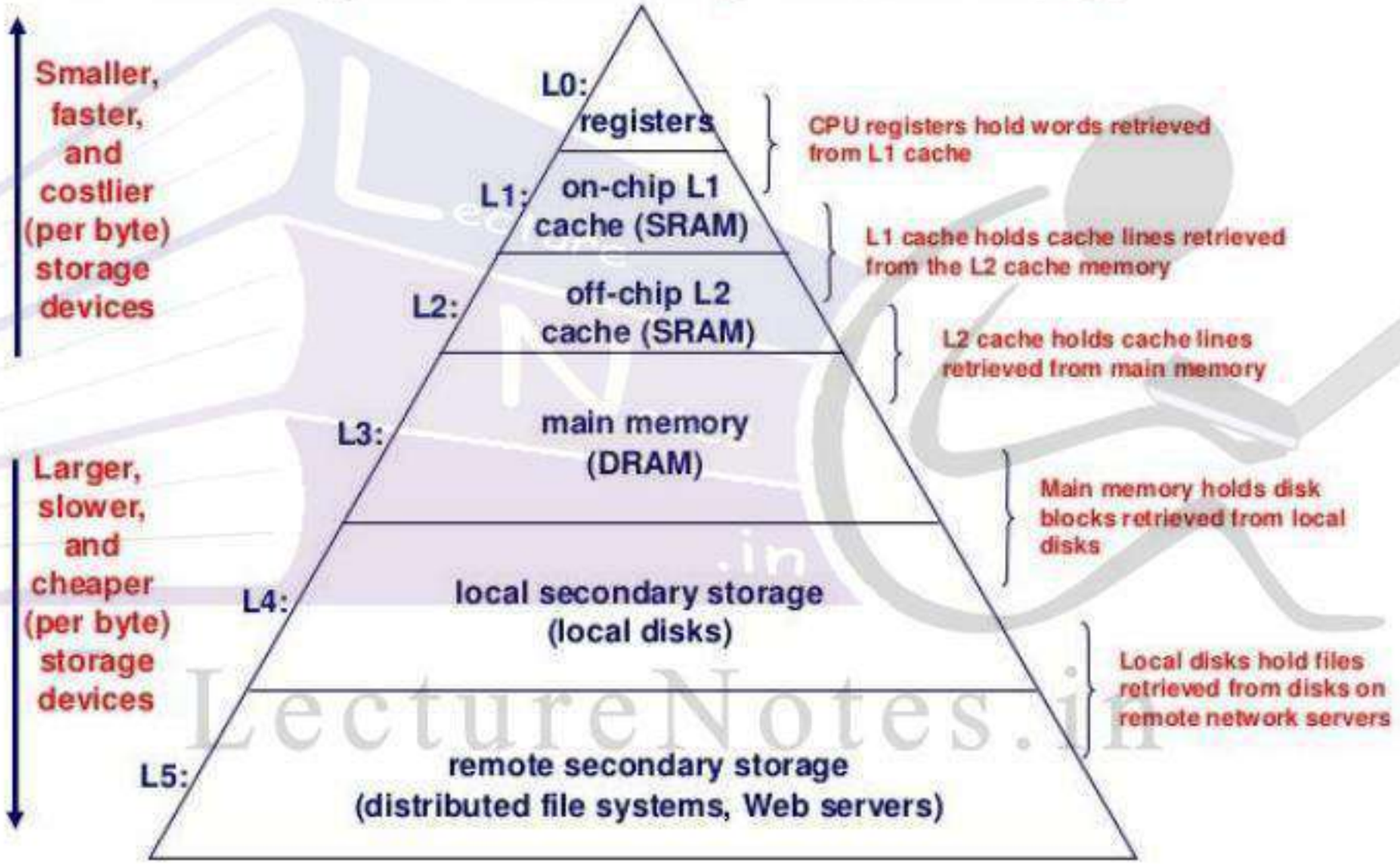
Criteria

1. Size (More)
2. Access Time (less)
3. Per unit cost (less)

LectureNotes.in

LectureNotes.in

An Example Memory Hierarchy



LectureNotes.in

LectureNotes.in

LectureNotes.in

- In a uni-programming system, main memory is divided into two parts. One part for the OS(kernel) and another part for the program currently being executed.

- In a multiprogramming system, the "user" part of memory must be subdivided further to accommodate multiple processes.

- The task of subdivision is carried out dynamically by the OS is known as memory management.

- To improve the utilization of the CPU and speed of computer, we must keep several processes in memory i.e we must share memory

LectureNotes.in

LectureNotes.in

Overview of Memory

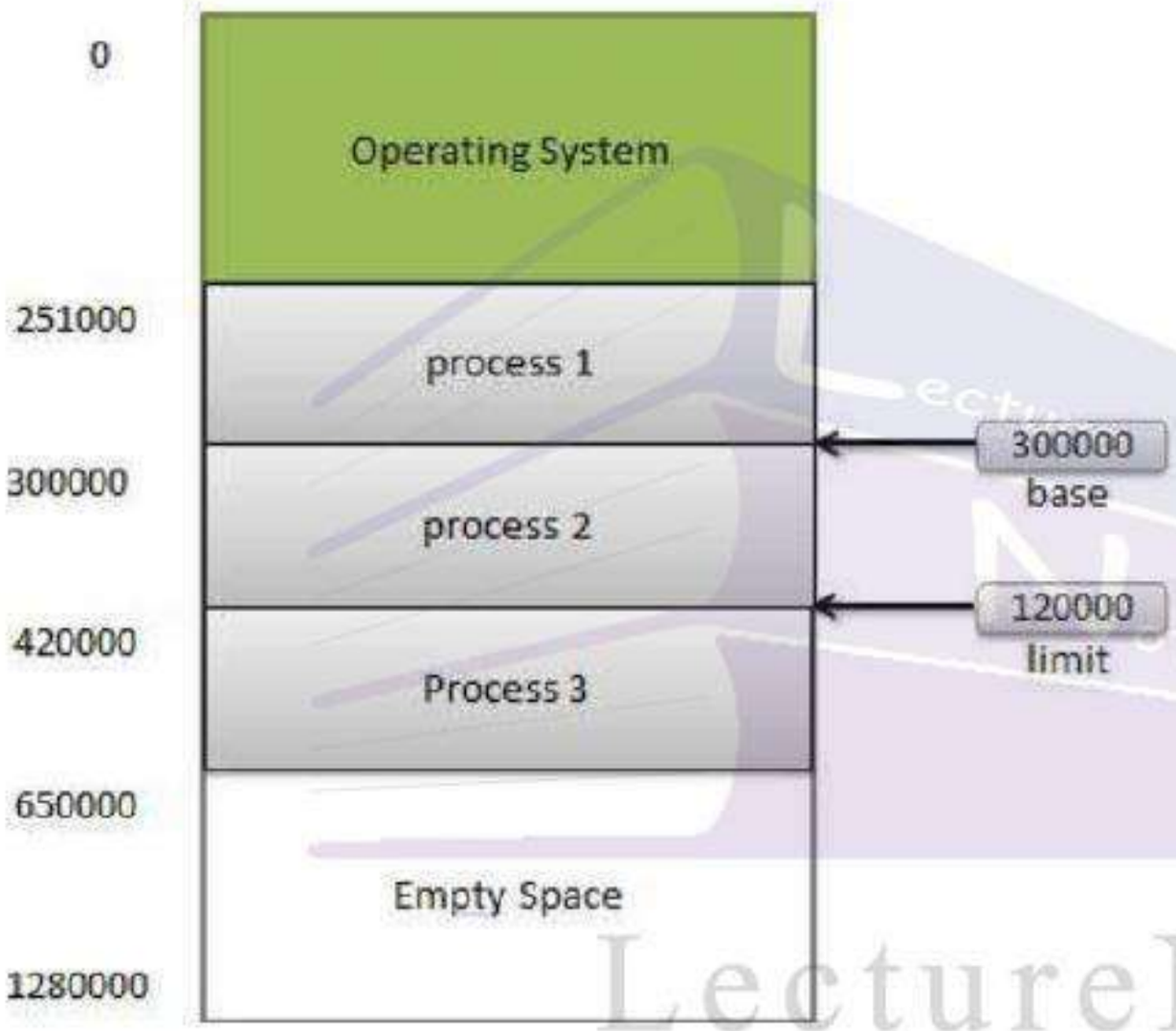
- Memory consists of large array of words or bytes, each with its own address.
- The CPU fetches instructions from memory according to the value of program counter
- Instruction execution cycle consists of, first fetch an instruction from memory. The instruction is then decoded and operands fetched from memory. The instruction has been executed on the operands, results may be stored back in memory.

LectureNotes.in

LectureNotes.in

LectureNotes.in

Memory Protection



Base and limit register defines a logical address space

- Each process has to store in a separate memory space.
- For concurrent execution multiple processes must be loaded to memory.
- To separate memory spaces and to ensure that a process can access its legal address space base and limit registers are used.
- Here the program can legally access all address from 300000 through 420000.

LectureNotes.in
LectureNotes.in

LectureNotes.in

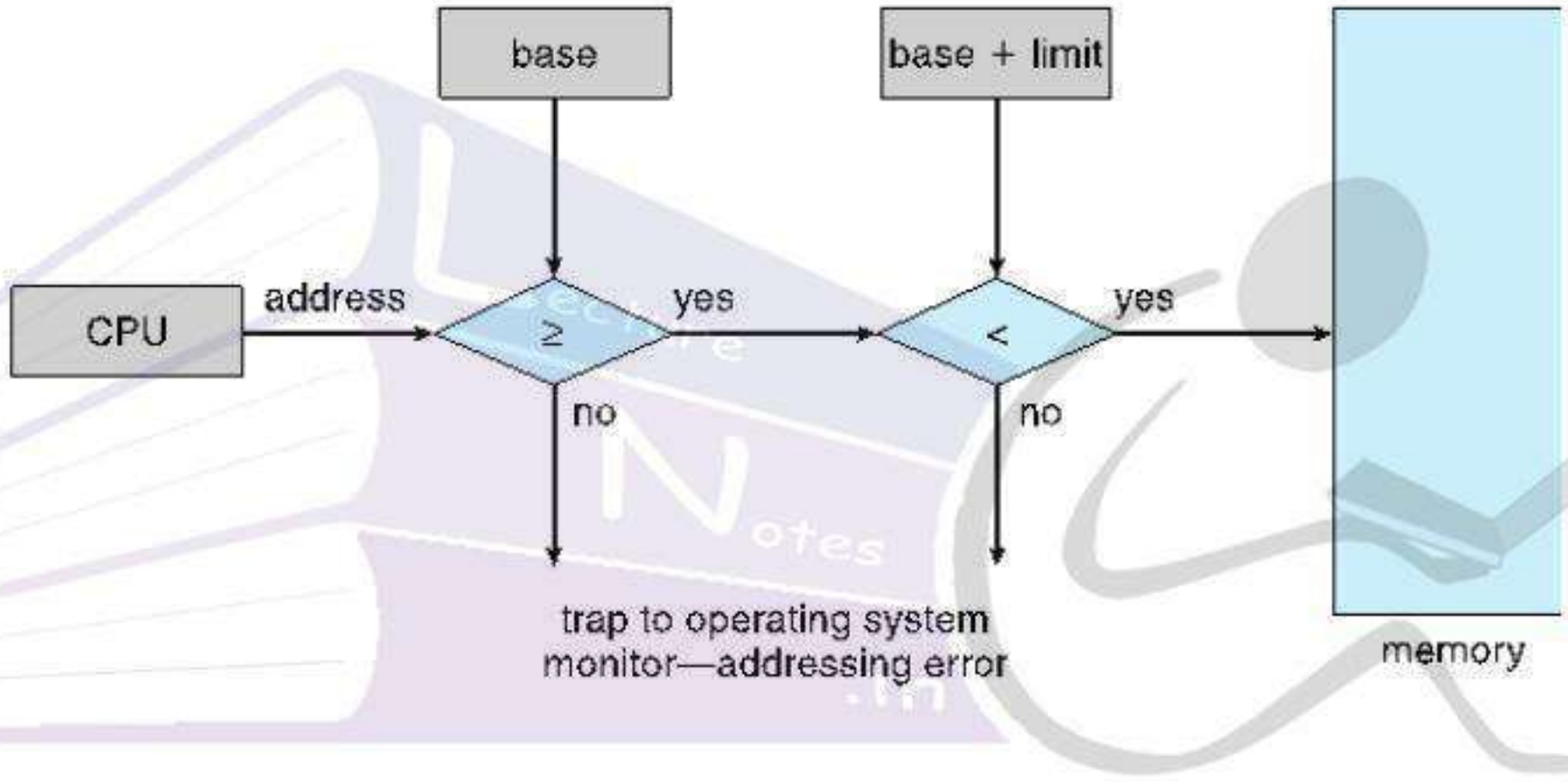
Memory Protection

- Protecting the O/S from user processes and protecting the user process from one another the base and limit registers are used.
- The base register contains the value of the smallest physical address and the limit register contains the range of the logical address.
- In this ex. base register holds 300000 and the limit register holds 120000, then the program can legally access all addresses from 300000 through 420000.

- Each logical address must be less than the limit register. The memory management unit (MMU) maps the logical address dynamically by adding the value in the base register. This mapped address is sent to the memory.

LectureNotes.in

LectureNotes.in



Hardware address protection with base and limit register

LectureNotes.in

LectureNotes.in

Address Binding

- A program resides on a disk as a binary executable file. The program must be brought into memory and placed within a process for it to be executed.

- The collection of processes on the disk that is waiting to be brought into memory for execution forms the input queue.

- Select one of the processes from the input queue and to load that process into memory. When the process is executed, it accesses instructions and data from memory. After execution, the process terminates and its memory space is free.

- Address in the source program are symbolic. The compiler will bind these symbolic addresses to relocatable addresses.

LectureNotes.in

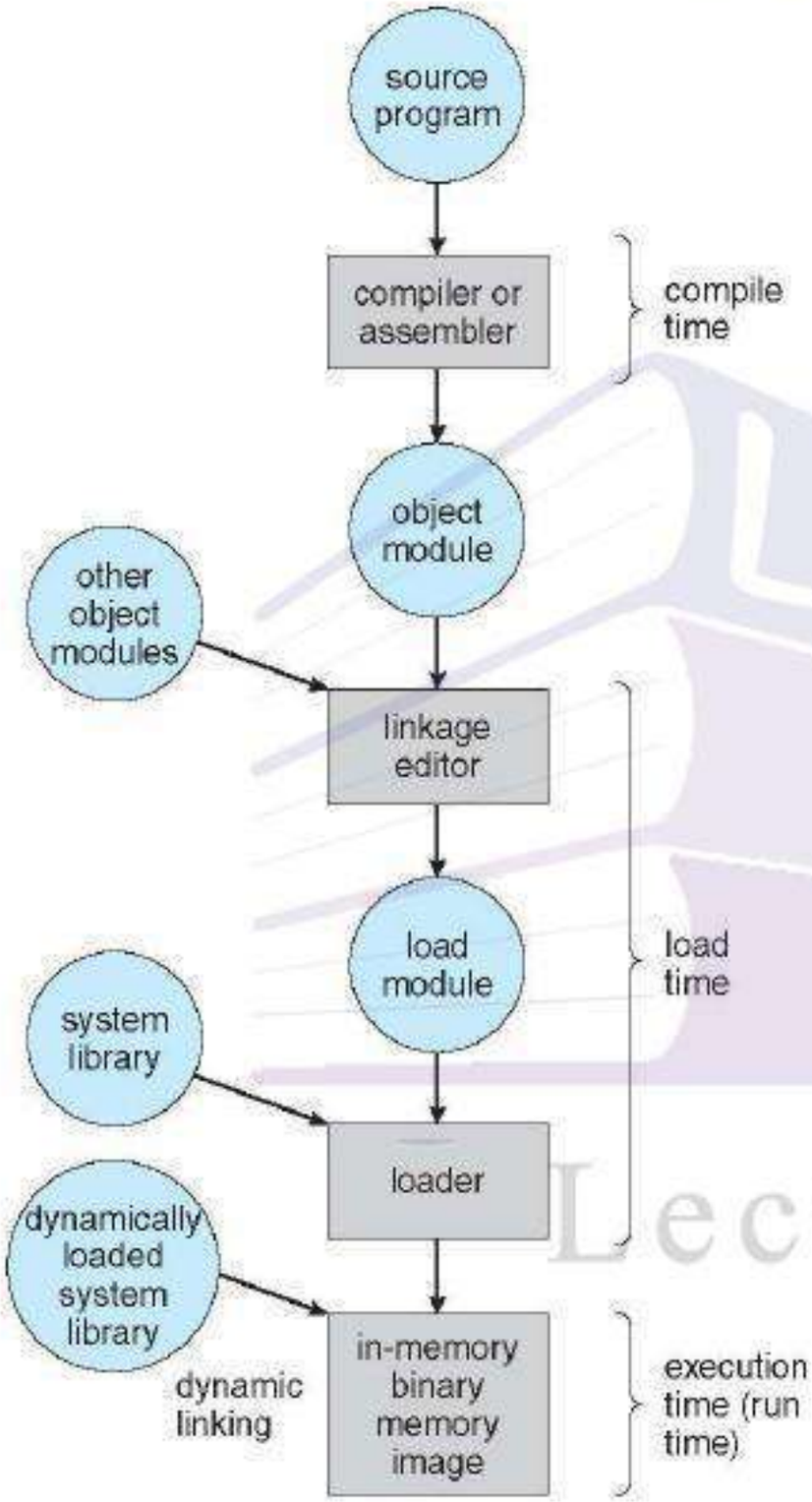
- The linkage editor or loader will bind these relocatable addresses to absolute addresses.
- Binding of instructions and data to memory addresses can be done:

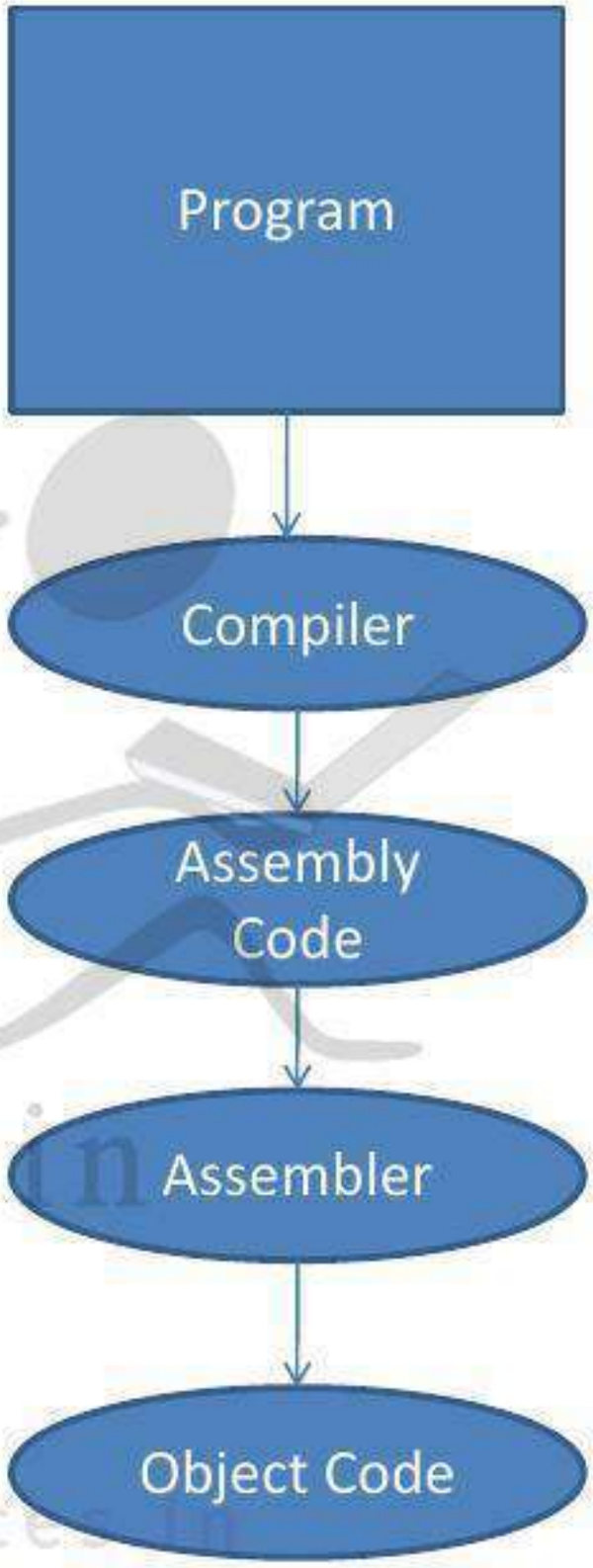
  Compile time: If memory location known a priori (known at the compile time where the process can reside), absolute code can be generated; must recompile code if starting location changes

  Load time: If address is not known at compiler time, it must generate relocatable code

  Execution time: Binding delayed until run time if the process can be moved during its execution from one memory segment to another

LectureNotes.in





Multistep processing of a user program

LectureNotes.in

Logical and Physical Address Space

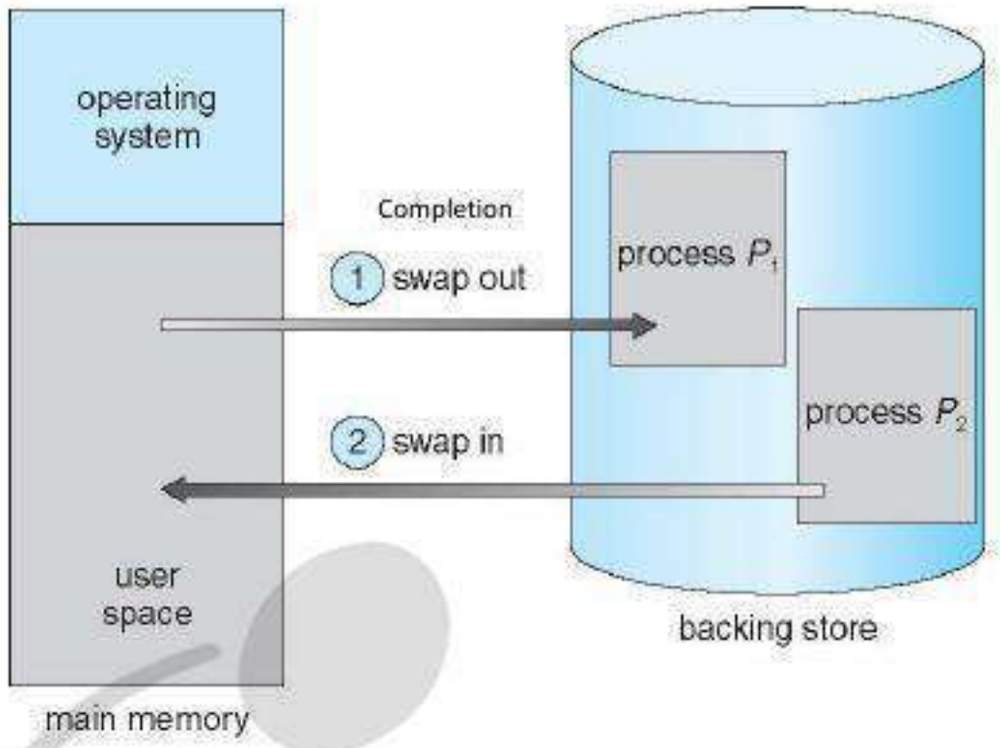
- An address generated by the CPU is called as logical address.
- An address seen by the memory unit i.e. the one loaded into the memory address register of the memory is referred as a physical address.

LectureNotes.in

LectureNotes.in

LectureNotes.in

- Swapping is a method to improve the memory utilization.
- A process needs to be in memory for execution. A process can be swapped temporarily out of memory to a backing store and then brought back into memory for continued execution.
- Suspended due to I/O, terminated itself after completed all of its operation and abnormal termination at the time of trap.
- Switching a process from the main memory to disk is said to be “swap out” and switching from disk to main memory is said to be “swap in”.



In RR scheduling, when a quantum expires, the memory manager will start to swap out the process that just finished and to swap in another process.

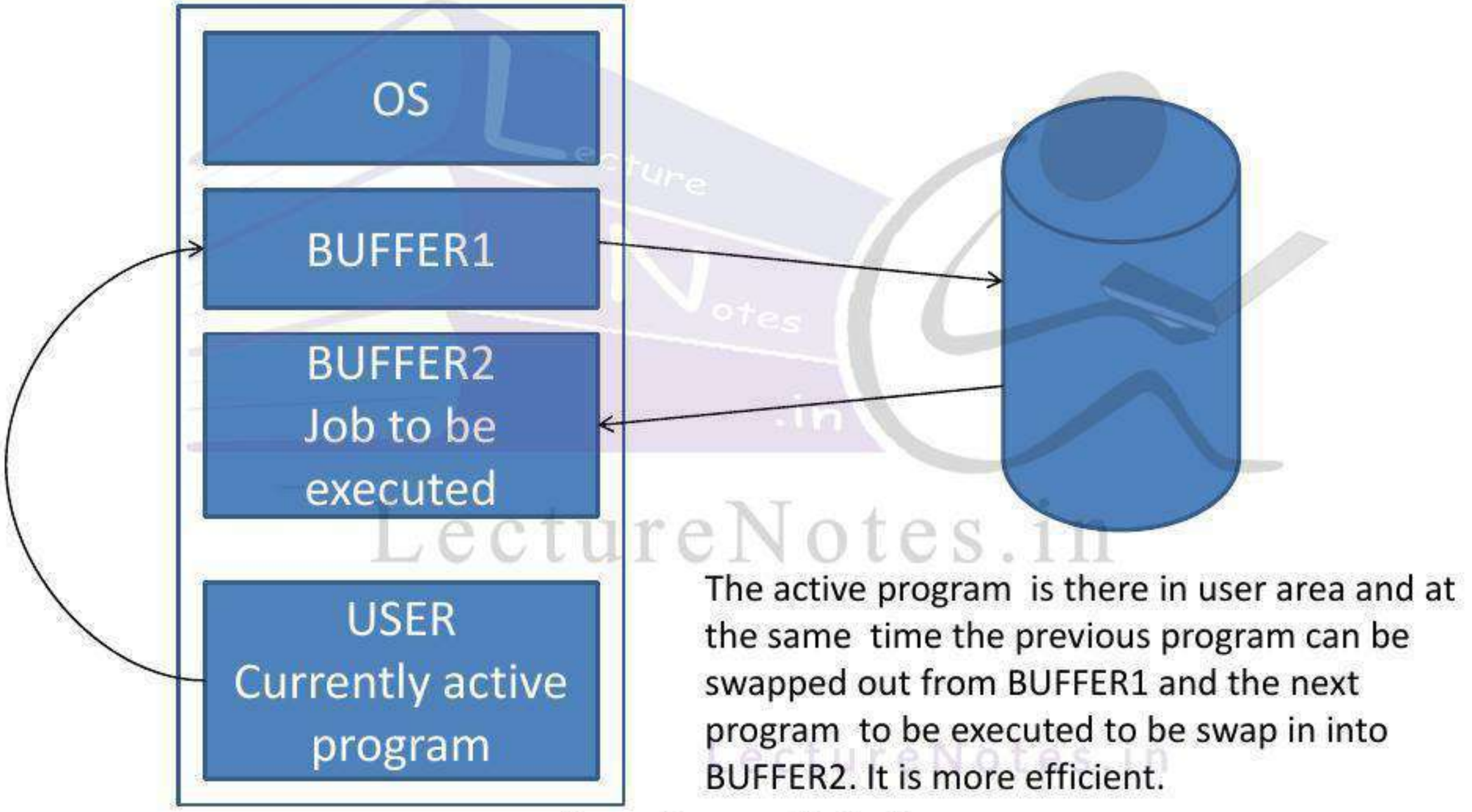
- Suppose the main memory composed of 10 processes, assume this is the maximum capacity the CPU and currently process 9 is executing. In the middle suppose P9 needs some I/O, then CPU switch to another process and P9 is moved to the disk and another process is loaded in main memory in place of P9. After completion of the I/O of P9 it will be moved to main memory from the disk.

LectureNotes.in
LectureNotes.in
LectureNotes.in
Memory Management by Ajay Sir

LectureNotes.in

• In every swap out and swap in performs some I/O operation. Which takes a huge amount of time. The CPU utilization is getting very poor if more number swap will be there. CPU cannot start until the swap in and swap out operation is completed.

• To improve the CPU efficiency



The active program is there in user area and at the same time the previous program can be swapped out from BUFFER1 and the next program to be executed to be swap in into BUFFER2. It is more efficient.

LectureNotes.in

Dynamic Loading

- Loading means load program or module from secondary storage device (disks) to main memory.
- Mainly divided into two types
  - Compile time loading / static loading
  - Runtime loading / dynamic loading
- All the routines are loaded in the main memory at the time of compilation is said to be static loading or compile time loading.
- The routines are loaded in the main memory at the time of execution or running is said to be dynamic loading.
- Only the main program will be loaded and the functions will be there in the sec. storage.

LectureNotes.in

LectureNotes.in

Dynamic Linking

- **Static linking** – system libraries and program code combined by the loader into the binary program image
- Dynamic linking –linking postponed until execution time
- Small piece of code, **stub**, used to locate the appropriate memory-resident library routine
- Stub replaces itself with the address of the routine, and executes the routine
- Operating system checks if routine is in processes' memory address
  - If not in address space, add to address space
- Dynamic linking is particularly useful for libraries
- System also known as **shared libraries**
- Consider applicability to patching system libraries
  - Versioning may be needed

LectureNotes.in

LectureNotes.in

Contiguous Memory Allocation

- The main memory must accommodate both the OS and the user processes.
- The memory is divided into two partitions i.e. Resident of OS and User processes
- The OS may place either in low or high memory. Since the interrupt vector is often in low memory, programmers usually place the OS in low memory.
- Generally, we want several user processes to reside in main memory at the same time. So, in contiguous memory allocation, each process is contained inn a single contiguous part of memory.

LectureNotes.in

LectureNotes.in

LectureNotes.in

Memory Management Technique

- Contiguous (ex. array)
    - Fixed Partition (Static)
    - Variable Partition (Dynamic)
- Non-contiguous (linked list)
    - Paging
    - Multilevel Paging
    - Inverted Paging
    - Segmentation
    - Segmented Paging

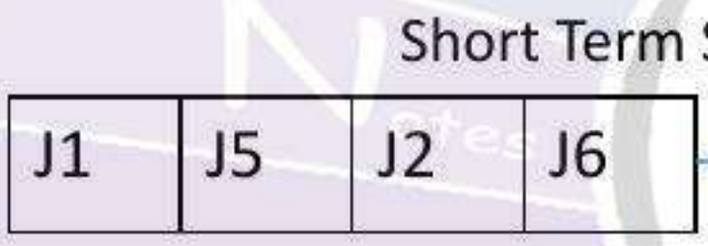
LectureNotes.in

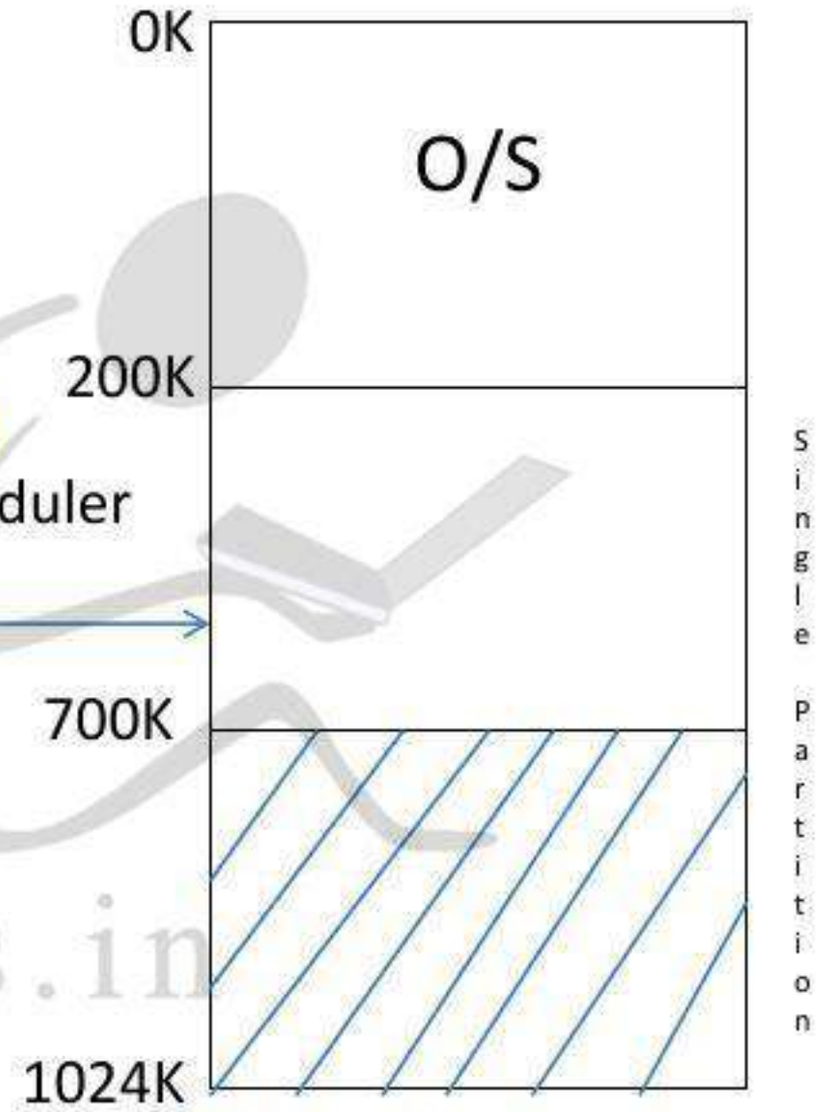
LectureNotes.in

LectureNotes.in

Single Partition Allocation

- The OS resides in the low memory and the remaining memory is treated as a single partition. This is available for user space. Only one job can be loaded in this user space.
- The main memory consists of only one process at a time because the user space is treated as a singe partition.





- ADVT: It is simple and doesn't require great expertise.
- DISADVT: Poor utilization of memory
- Poor utilization of process waiting for I/O
- User's job is limited to the size of available main memory.

LectureNotes.in

LectureNotes.in

Multiple Partitions memory Management

- This method can be implemented in three ways.
- Fixed equal multiple partitions memory mgt.
- Fixed variable multiple partitions memory mgt.
- Dynamic multiple partitions memory mgt.

LectureNotes.in

LectureNotes.in

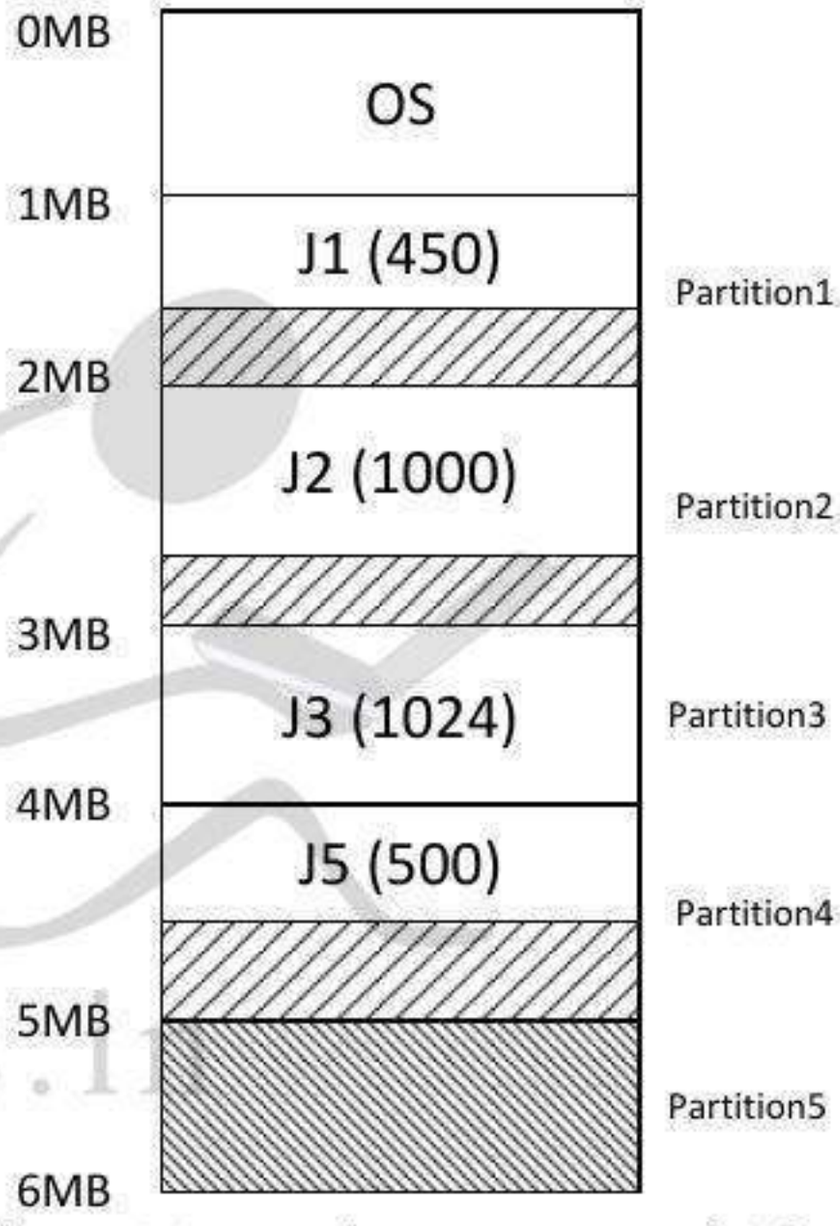
LectureNotes.in

Fixed equal multiple partitions/ static partition

- The user space is divided into fixed partitions. The partition sizes are depending upon OS.

Total MM is of size 6MB. 1MB occupied by OS.
Remaining is partitioned into 5 equal fixed partitions.
J1,J2,J3,J4,J5 are 5 jobs to be loaded in to MM.

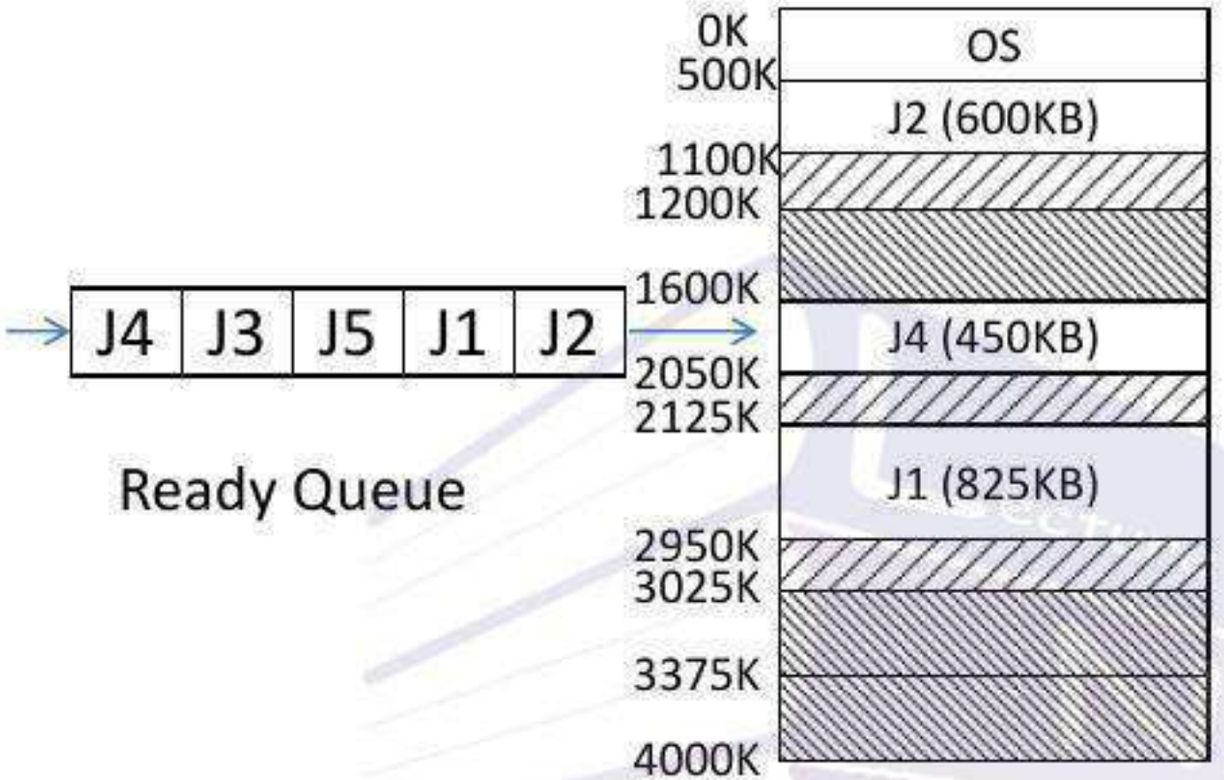
Job	Sizes
J1	450KB
J2	1000KB
J3	1024KB
J4	1500KB
J5	500KB



- Job1 is loaded into partition1. The size of par1 is 1024KB. Size of J1 is 450KB.
- Waste space= 1024-450=574KB
- This known as internal fragmentation.

There is no enough space to load J4 bcoz the size of J4 is GT all partitions. The memory which is not allotted to any process is external fragmentation i.e. 1024KB.
Total internal fragmentation is = (1024-450)+(1024-1000)+(1024-500)=1172KB
A partition of MM wasted within a partition is internal frag. And the wastage of an entire partition is external fragmentation.

Fixed variable multiple partitions



Partitions	
Partition	Sizes
P1	700KB
P2	400KB
P3	5254KB
P4	900KB
P5	350KB
P6	625KB

Job Queue		
Job	Sizes	Arival Time
J1	825KB	10ms
J2	600KB	5ms
J3	1200KB	20ms
J4	450KB	30ms
J5	650KB	15ms

- In this the user space is divided into a number of partitions but the partition sizes are different.
- The OS keeps track which partitions are occupied and which are free.
- J2 arrives first and after searching from low to high loaded to Partion1.
- J1 arrives next. P4 is large enough to accommodate. So J1 into P4 leaves 75KB free.
- J5 arrives no enough space to load that job. J5 has to wait till gets free memory.
- J3 arrives next, there is no large enough partition to load that.
- J4 arrives last the size is 450KB, partition P3 is large enough to lad that one, loaded to P3.
- Total external Fragmentation= 400+350+625=1375
- Total internal fragmentation= 100+75+75=250 (J2 can be loaded P6 for efficient use)

LectureNotes.in

LectureNotes.in

To avid this problem, we use (partition allocation policy)

1. First Fit algorithm: It allocates the process in the fist free large enough partition.
2. Best Fit: It allocate the process in the smallest possible memory partition
3. Worst Fit: It searches for the largest possible memory partition

Example: There are five different size memory partitions of 100K,500K,200K,300K,600K (in sequence). By using first, best and worst fit algo place the processes of 250,455,189,465K (in order).

LectureNotes.in
LectureNotes.in

LectureNotes.in

A table called Partition Description Table (PDT) keeps information about the partitions.

Let us consider we have 1 MB of primary memory (RAM). Four partitions are marked as 200K, 250K, 150K and 500K respectively.

To keep track of all partitions, a table is created as follows

P. No	P. Base	P. Size	P. Status
1	0K	200	FREE
2	200K	250	FREE
3	450K	150	FREE
4	600K	500	FREE

LectureNotes.in

Dynamic Partitions Memory Management

•In this method the partitions are created dynamically, so that each process will be loaded in the partition exactly the same size of the process.

•In this scheme the entire use space is treated as a “big hole”. The boundaries of the partitions are dynamically changed and the boundaries are depend on the size of the process.

Job Queue		
Job	Sizes	Arival Time
J1	825KB	10ms
J2	600KB	5ms
J3	1200KB	20ms
J4	450KB	30ms
J5	650KB	15ms

• The jobs J2,J1, J5,J3 are loaded. The last job J4, the size is 450K, but the available memory is 225K (4000-(500+600+825+650+1200)). So the job J4 has to wait until the memory is available.

ADVT: This scheme support multiprogramming.
Efficient utilization of the processor and I/O devices.
Partitions are changes dynamically, so it does not suffer from internal fragmentation.

DISADVT:
This scheme suffers from internal as well as external fragmentation.

LectureNotes.in
LectureNotes.in
Memory Management by Ajay Sir

LectureNotes.in

Compaction

The solution to external fragmentation is compaction.
The goal of compaction is to place all free memory together in one large block.

- Compaction is not possible if relocation is static.
- Compaction is possible if relocation is dynamic done at execution time.
- The simplest compaction algorithm is to move all processes towards one end of memory so that all holes will be moved to the other end producing one large hole of available memory.

ADVT: This improves memory utilization and performance of system.
DISADVT:
This scheme is very expensive.
It is not always possible.

LectureNotes.in

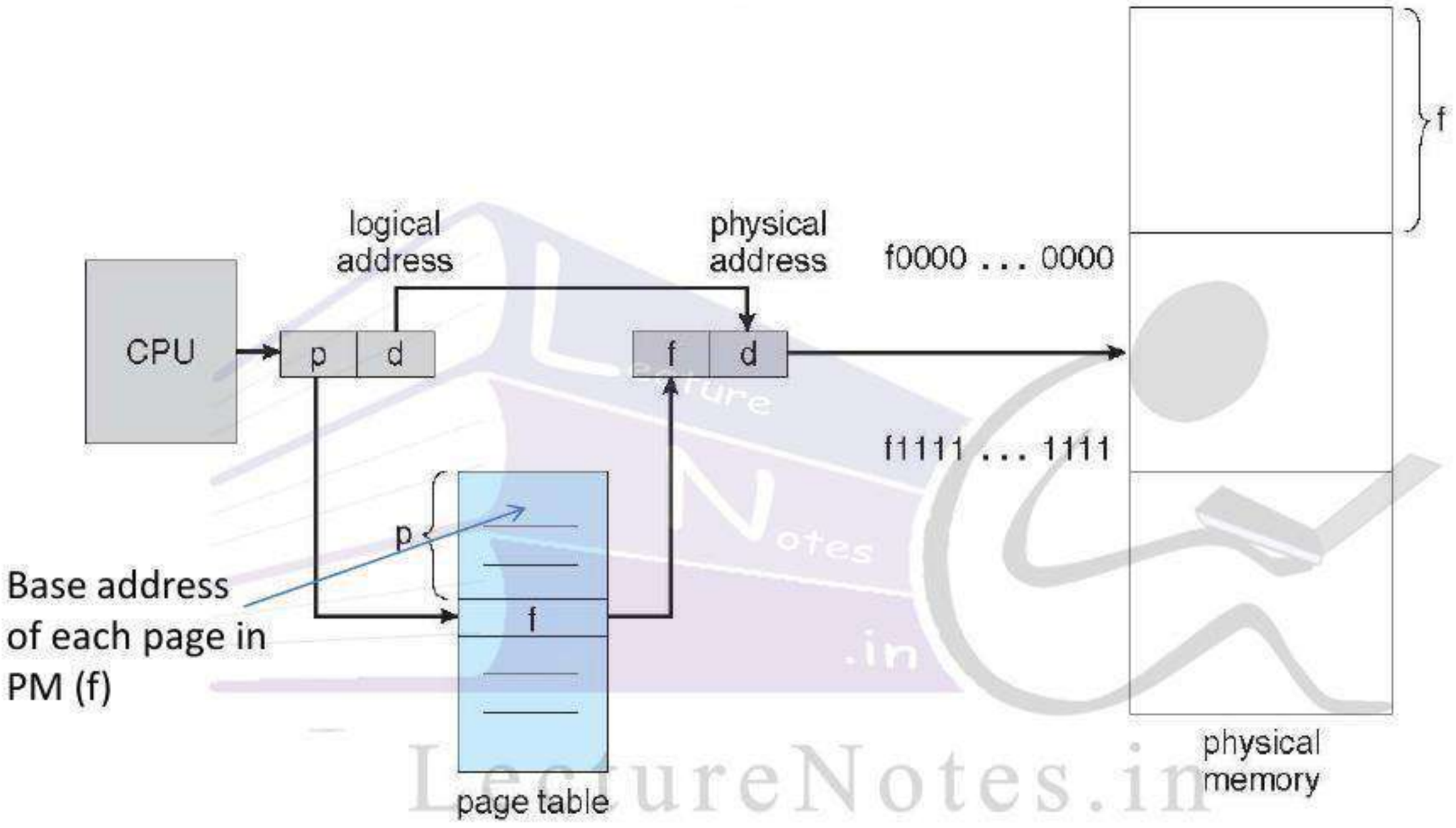
LectureNotes.in

Non-contiguous Memory Allocation
Paging

- A contiguous memory may not be available always.
- For execution of a process no consecutive memory is required.
- A process is divided to a number of parts, which may be loaded into different area of RAM / primary memory.
- A process is divided to number of equal size blocks called pages.
- Primary memory or physical memory into same sized blocks called **frames.**
- **Size of the page = Size of the frame**
- A page of a process is loaded into any of the available frame.
- A Page Table keeps track of which page of a process is loaded into which frame of primary memory.
- For every process there is a page table.

LectureNotes.in

Paging Hardware



p= page no

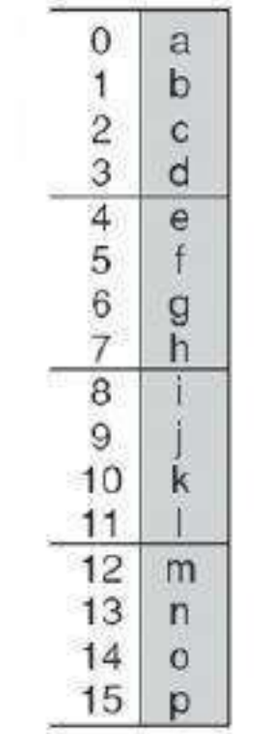
d= instruction offset within that page/ displacement

Page table holds the base address of every page.

Page table is a metadata. It is a penalty. 555=5+55

LectureNotes.in

Paging Example



logical memory

page table

physical memory

Mapping of logical address to physical address.
A page size =4 bytes and physical memory = 32 bytes(8 pages)
Logical address of 0 is page 0, offset 0
Logical address of 1 is page 0, offset 1
Logical address of 2 is page 0, offset 2
Logical address of 3 is page 0, offset 3
Logical address of 4 is page 0, offset 0
Logical address of 5 is page 0, offset 1
.
.
.

Logical address of 0 is page=0, offset=0
Physical Address= Frame no X page size +offset
Physical addr of logical addr 0 is 20= 5X4+0
Physical addr of logical addr 3 is 23= 5X4+3
Physical addr of logical addr 4 is 24= 6X4+0
Physical addr of logical addr 13 is 9= 2X4+1

n=2 and m=4 32-byte memory and 4-byte pages

LectureNotes.in

Advt. and dis-advantages of Paging

- No external fragmentation
- Internal fragmentation bcoz only the last page of the process is not filled completely. On an average the last page of the process is half full, so it is called ½ of the page.
- Sharing of pages can be done.
- It supports virtual memory

Dis-advantages:

- Extra space is required for sharing the page table
- Takes more time for address translation
- It suffers from page break. Suppose logical address space is 17KB, page size is 4KB, So this job requires 5 frames, the 5th frame consists of only 1KB, so reamaining 3KB wasted. Therefore, it is called page break.
- If the number of pages are high, it is difficult to maintain page table.

LectureNotes.in

Segmentation

- Segmentation is a method in which a process will be divided into parts and put it in the MM looking to the related data i.e function, arrays…..
- Pagging also same but in fixed size. (Example of C program, Inst1…..Inst99)
- Segmentation divided the program in variable sizes. It is not diving the process without looking into it.
- The user view is not same as the actual physical memory.
- Memory-management scheme that supports user view of memory contiguous memory allocation.
- A logical address space is a collection of segments. It is defined as logical grouping of instructions such as :

main program	procedure
function	method
object	local variables, global variables
common block	stack
symbol table	arrays

- Each segment has a name and length. The addresses specify both the segment name and the offset within the segment.

LectureNotes.in
Memory Management by Ajay Sir

LectureNotes.in

Segmentation Architecture

- CPU generates logical address
- Logical address consists of a two tuple:
- <segment-number, offset>,

- **Segment table** – maps two-dimensional physical addresses; each table entry has:
  - **base** – contains the starting physical address where the segments reside in memory
  - **limit** – specifies the length of the segment

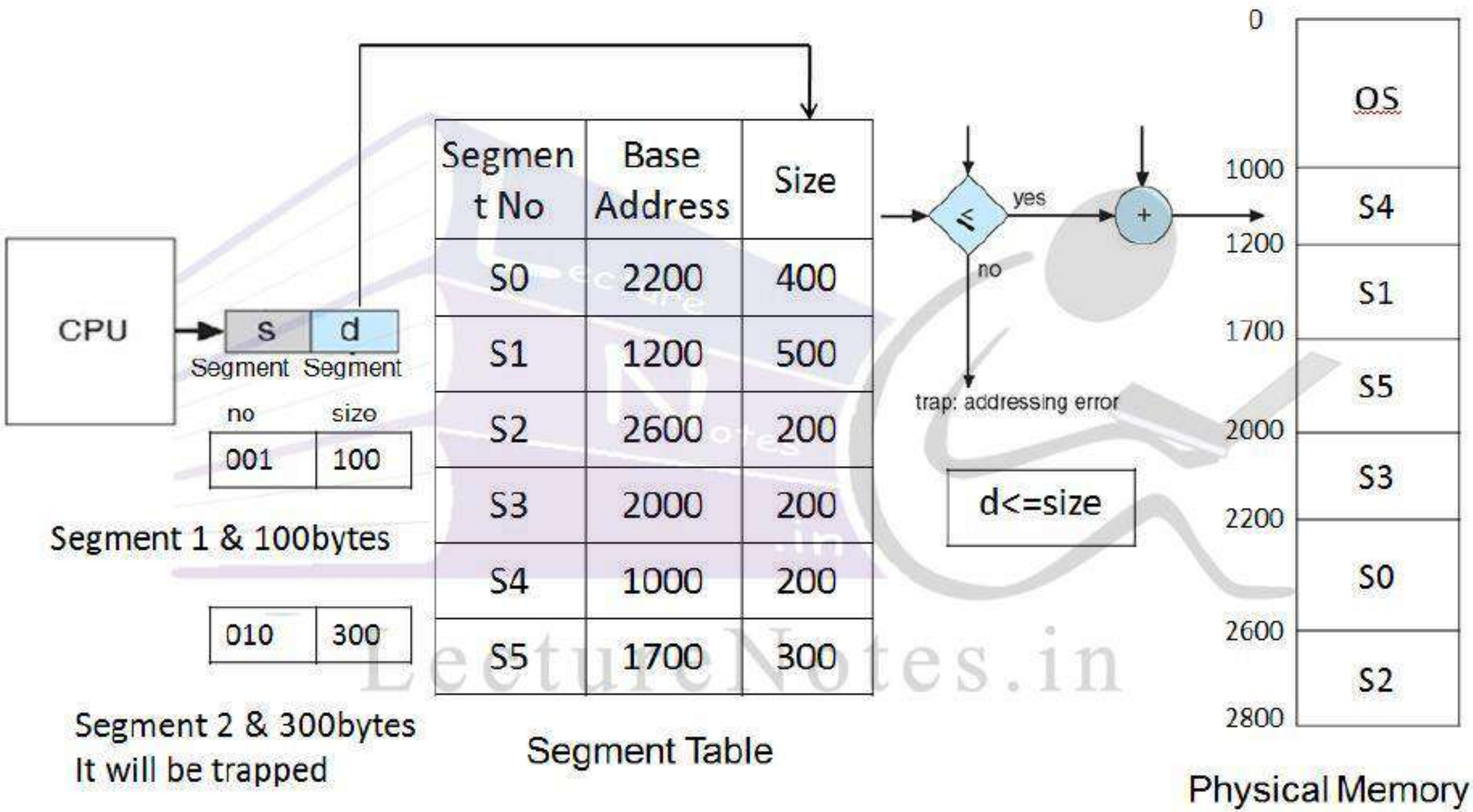
- **Segment-table base register (STBR)** points to the segment table's location in memory

- **Segment-table length register (STLR)** indicates number of segments used by a program;
- segment number ***s*** is legal if ***s*** < **STLR**

LectureNotes.in

LectureNotes.in

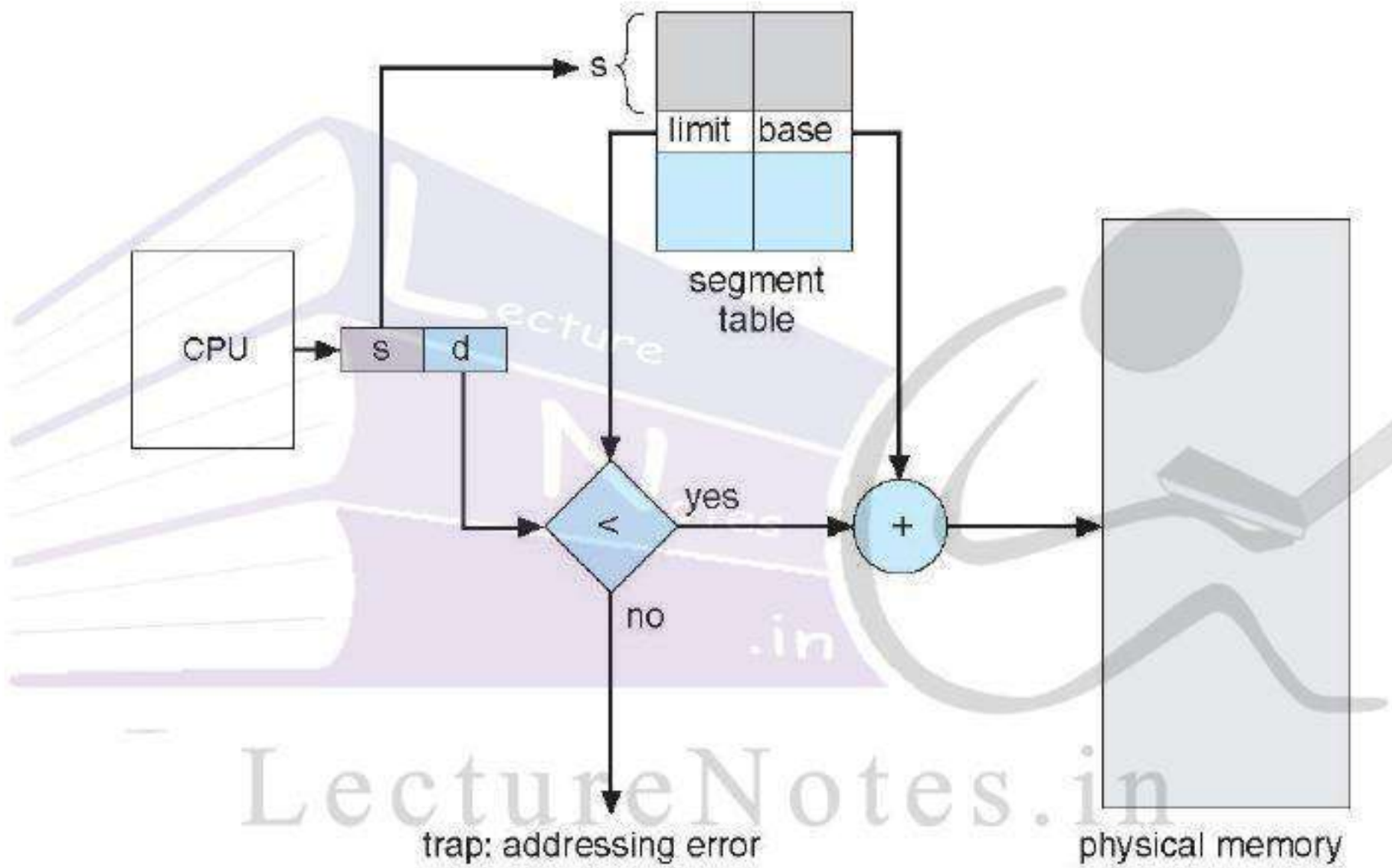
Segmentation Hardware



LectureNotes.in

LectureNotes.in

Segmentation Hardware



LectureNotes.in

LectureNotes.in

LectureNotes.in

Comparison between Paging and Segmentation

Paging

- Logical memory is partitioned into pages
- The logical address space is divided into pages by MMU
- Suffering from internal fragmentation
- O/S maintains a page table to map between frames and page.
- It doesn't support the user's view of memory.
- Page no. and displacement used to calculate absolute address(p,d).
- Multilevel paging is possible.
- O/ S maintains a free frame list, so no need to search for free frame.

Segmentation

- Logical memory is partitioned into segments.
- The logical address space is divided into segments specified by programmers.
- Suffering from external fragmentation
- O/S maintains a segment table for mapping.
- It supports the user's view of memory.
- Segment no. and offset is used to calculate absolute address(s,d).
- Multilevel segmentation is possible but no use
- O/ S maintains the particulars of available memory.

LectureNotes.in

Paging using Translation Look-aside Buffer

- Page table is data structure implemented as a set of dedicated registers. The use of registers for small page table is efficient. But large page table these are not feasible.
- The page tables are kept in main memory. In the main memory where the program stores the page table also stores there.
- By accessing a logical address by the CPU is to access the page table in the main memory is same as accessing the page table first and the physical address next.
- So the main memory will be accesses twice or by factor 2.
- It is a penalty of time. To reduce this time.
- The memory access is slow. (trade off speed and time)

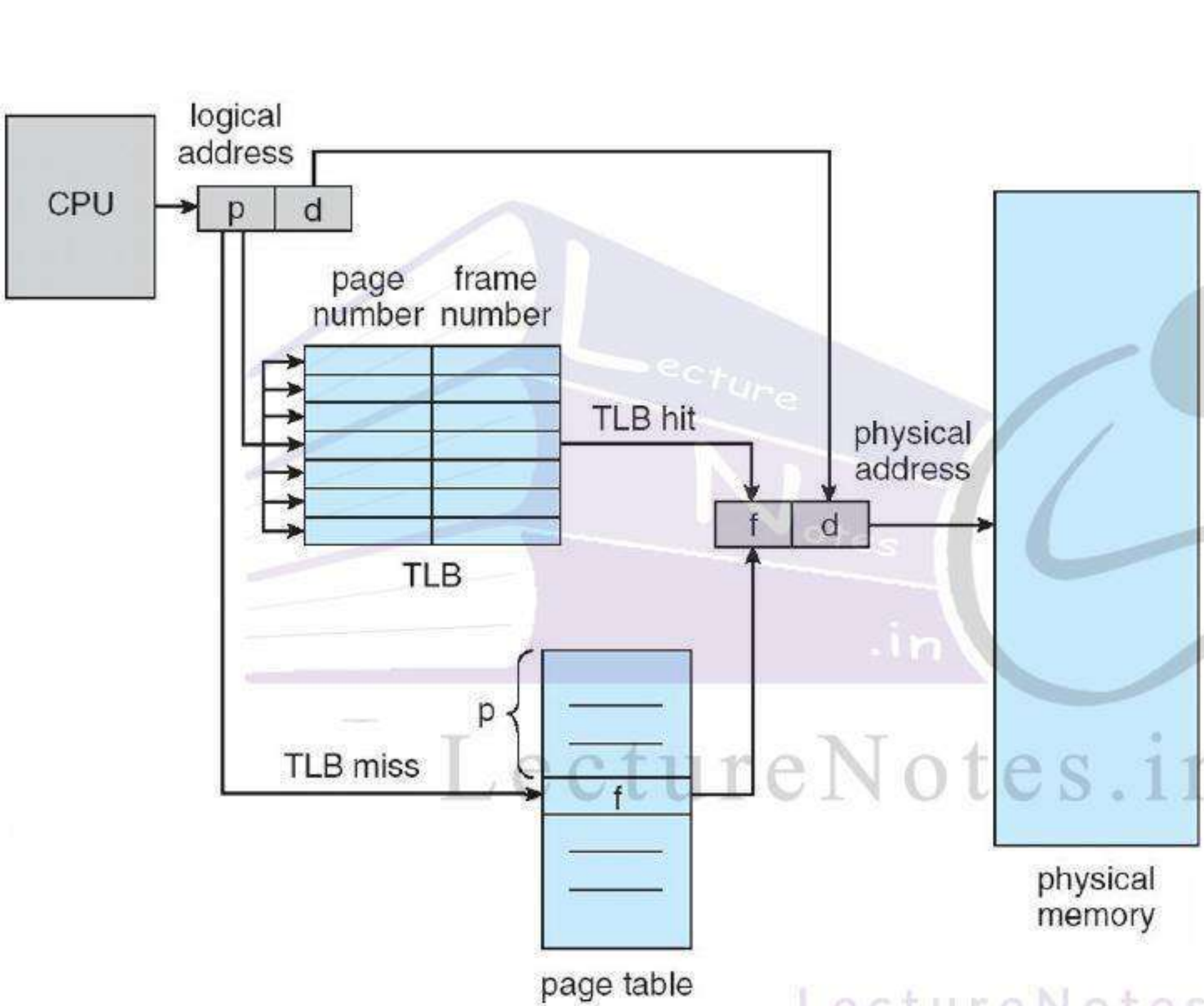
LectureNotes.in

The solution is

- A special small fast look up h/w cache called Translation Look-aside Buffer (TLB).
- While context switch one has clear the TLB so that next process can start fresh.

LectureNotes.in

Paging Hardware With TLB



P1
Inst1
Inst2
Inst3
.....
Inst50

P2
Inst1
Inst2
Inst3
.....
Inst50

LectureNotes.in

LectureNotes.in

Inverted Page Table

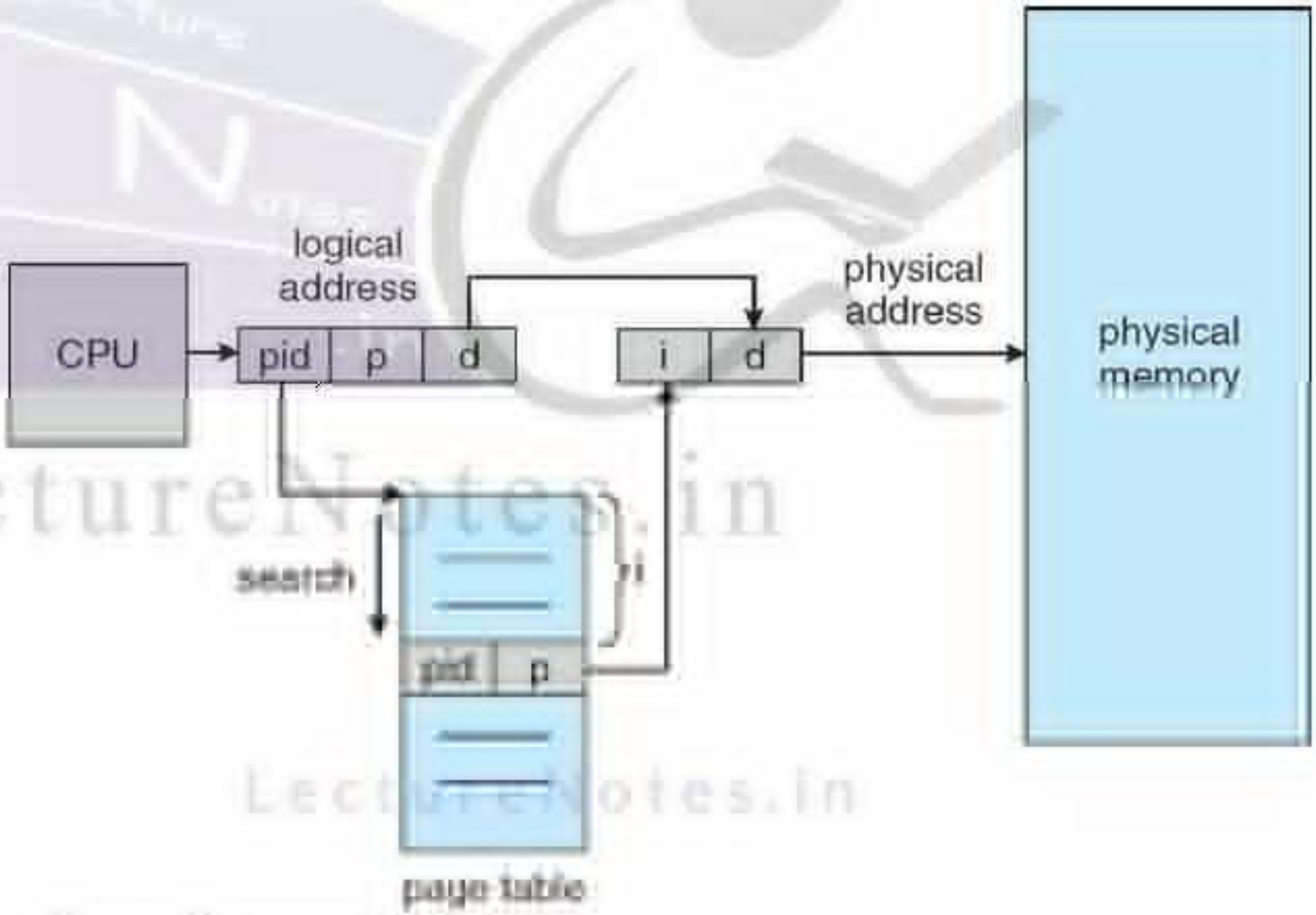
- Each process is having a page table which keep track of all possible logical pages, track all physical pages
- Each page table may consists of millions of entries. These tables may consume large amount of physical memory because for each process it creates a page table. As the MM is limited so frames are limited. So we have to think something
- To reduce the memory one can use inverted page table.
- Instead of keeping all the page table for all the processes, one can create a global page table which will handle all the page table.
- Here the searching time is more.

LectureNotes.in
LectureNotes.in

Inverted Page Table Architecture

Global Page Table

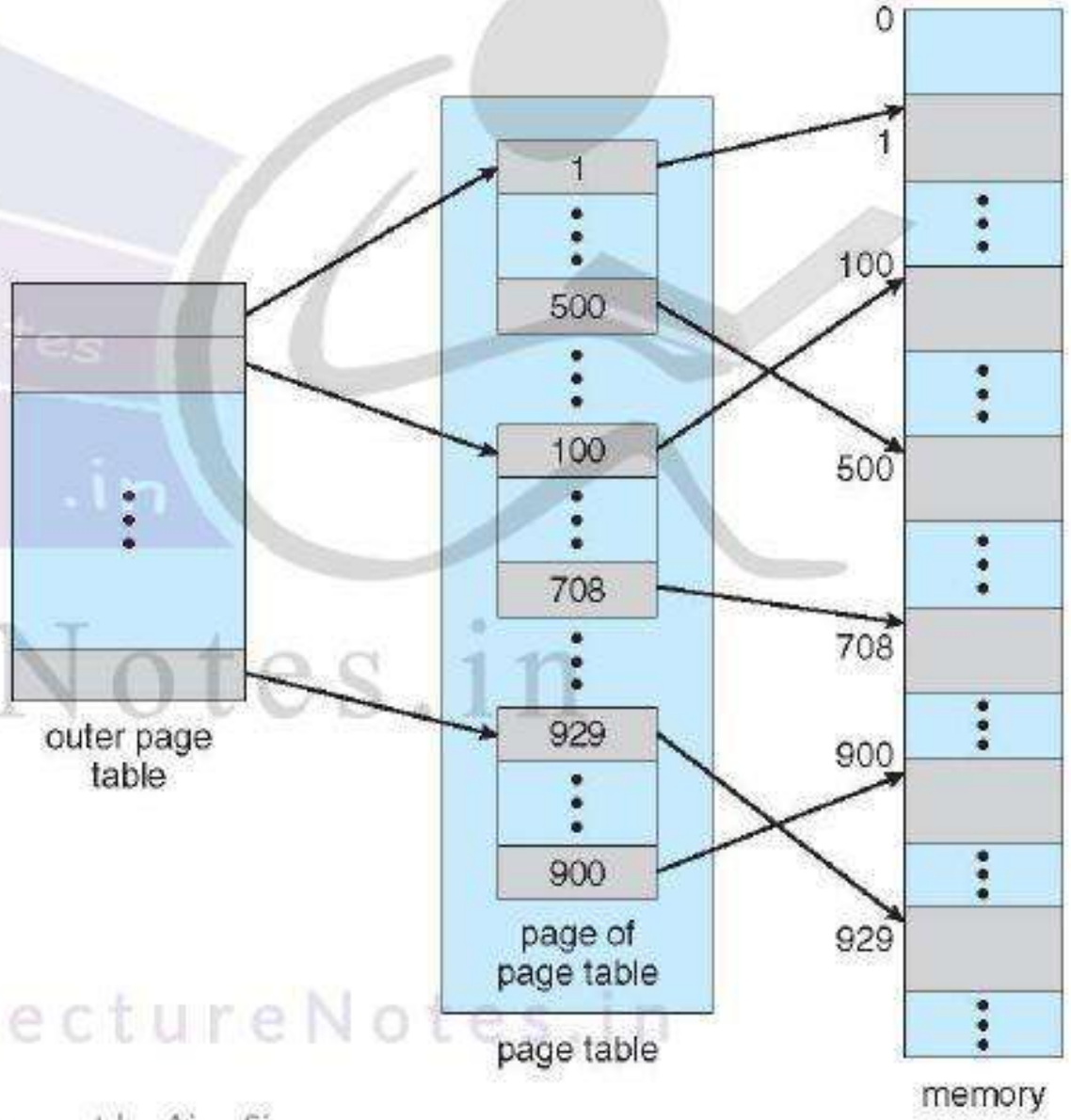
Frame No	Page No	Process ID
0	Page0	P1
1	Page3	P2
2	Page2	P3
3	Page2	P1
4	Page1	P3
5	Page0	P2



LectureNotes.in

HIERARCHICAL PAGING

- Most of the modern computer system supports a large logical-address space(2^{32} to 2^{64})
- So the page table becomes exclusively large.
- Example: Consider a system of 32-bit logical address space. If the page size of the system is 4KB(2^{12}), then a page table will consists of 1 million entries ($2^{32}/2^{12}$).



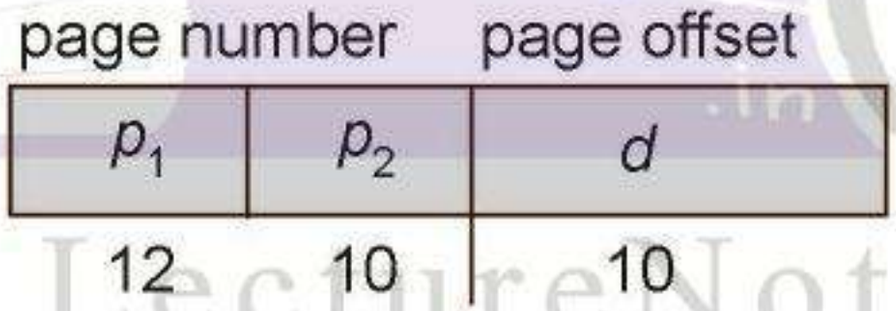
Two-Level Page-Table Scheme

LectureNotes.in
Memory Management by Ajay Sir

LectureNotes.in

Two-Level Paging Example

- A logical address (on 32-bit machine with 1K page size) is divided into:
  - a page number consisting of 22 bits
  - a page offset consisting of 10 bits
- Since the page table is paged, the page number is further divided into:
  - a 12-bit page number
  - a 10-bit page offset
- Thus, a logical address is as follows:

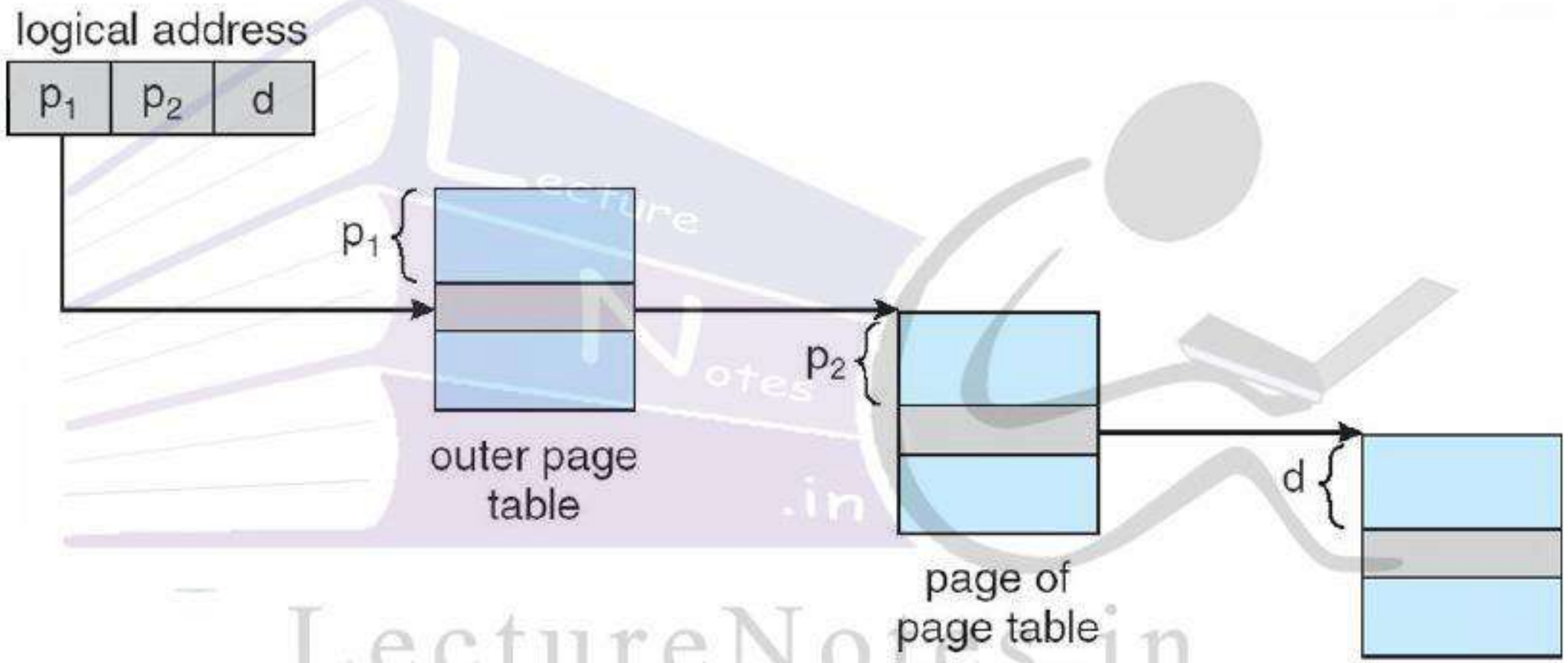


- where p_1 is an index into the outer page table, and p_2 is the displacement within the page of the inner page table
- Known as **forward-mapped page table**

LectureNotes.in
LectureNotes.in

LectureNotes.in

Address-Translation Scheme



LectureNotes.in
LectureNotes.in

Virtual Memory

- Whenever the process size is more than the MM the OS helps us in executing the process.
- Virtual memory is a technique that allows the execution of the processes that may not be completely in the memory.
- Programs can be executed even program size (logical address space) is greater than the physical available memory.
- The main advantage that the more larger programs can be executed in the physical memory which size is less.
- Ex. Suppose a program size is 15MB but available memory is 12MB. So,, 12MB is loaded in main memory, remaining 3MB loaded in secondary memory. When this 3MB will be needed fro the execution then swap out 3MB swap out 3MB from main memory and swap in 3MB from secondary memory to main memory.
- Virtual memory is a separation of user logical memory from physical memory. This allows an extremely large VM to be provided for programmer when only a small physical memory is available.

LectureNotes.in

Advantages of Virtual Memory

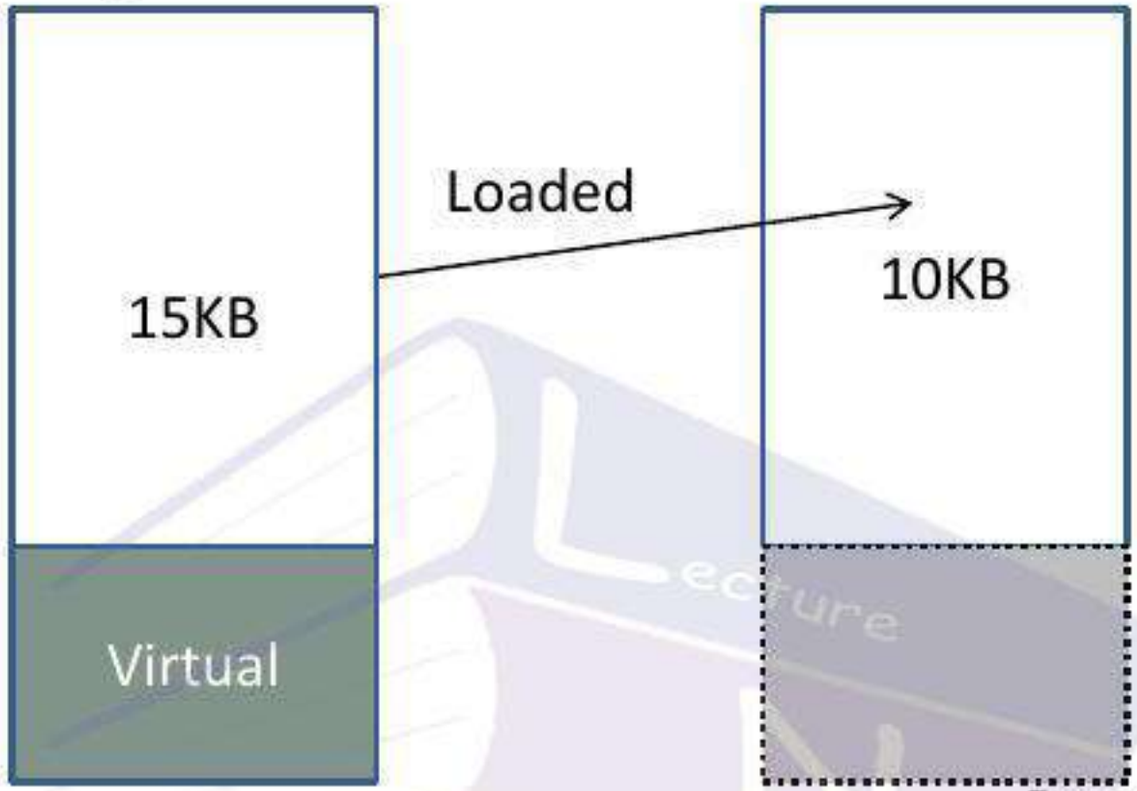
- Efficient main memory utilization
- Ability to execute a program that is partially available in the main memory.
- More programs can be loaded to the main memory so that degree of multiprogramming will be achieved and with more CPU utilization.
- Less I/O will be needed to load or swap each user program into memory. So, each user program would run faster.
- It allows processes to easily share files address spaces(logical memory) through page sharing. Due to this, it provides efficient mechanism for process creation.
- Virtual memory is implemented by demand paging.
- Here both the system and user will be benefited.

LectureNotes.in

LectureNotes.in

Program Size

RAM

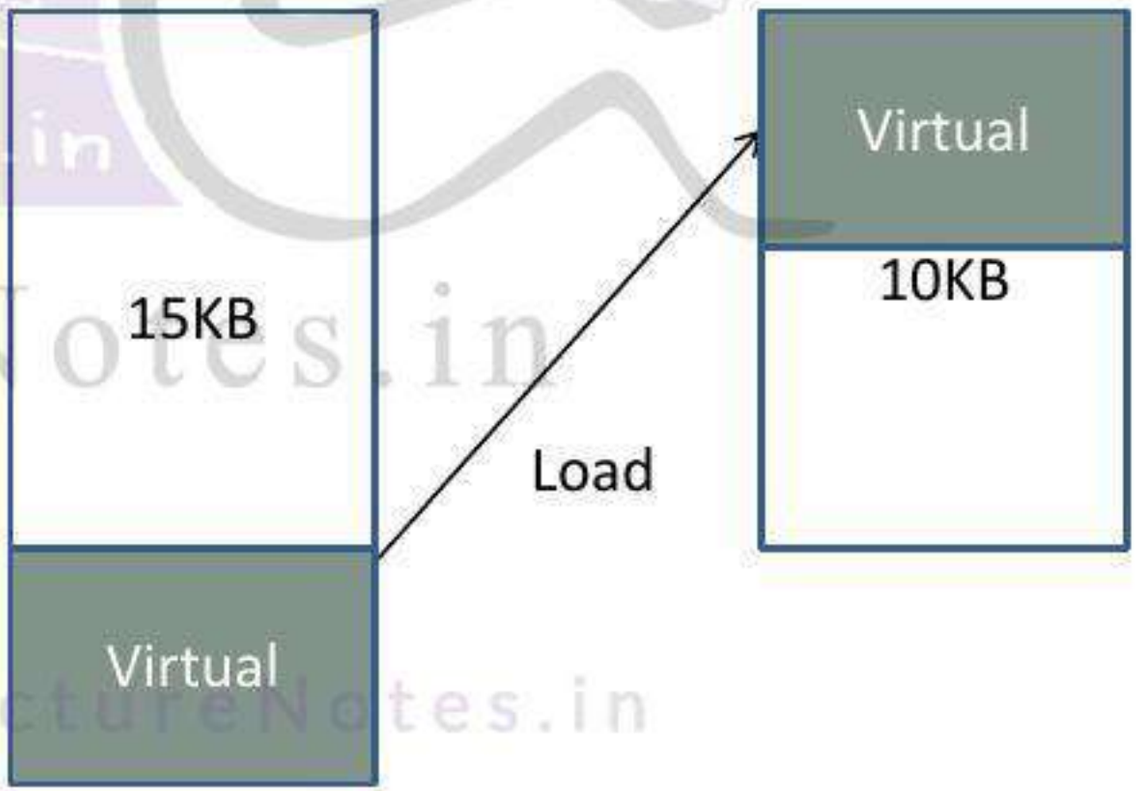


Physical Execution
Logical Execution

Seems to be
extended by CPU

Program Size

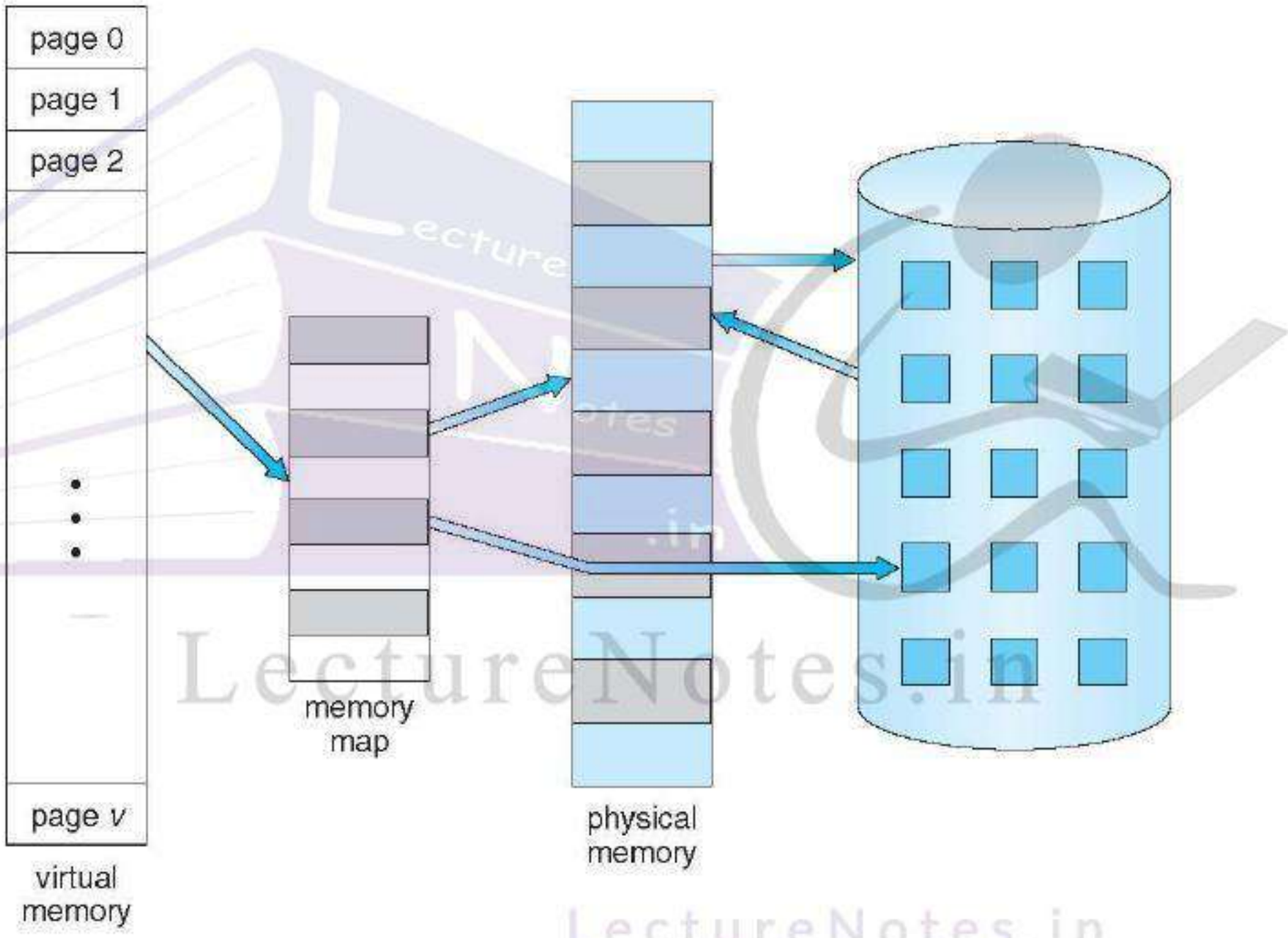
RAM



LectureNotes.in
LectureNotes.in

LectureNotes.in

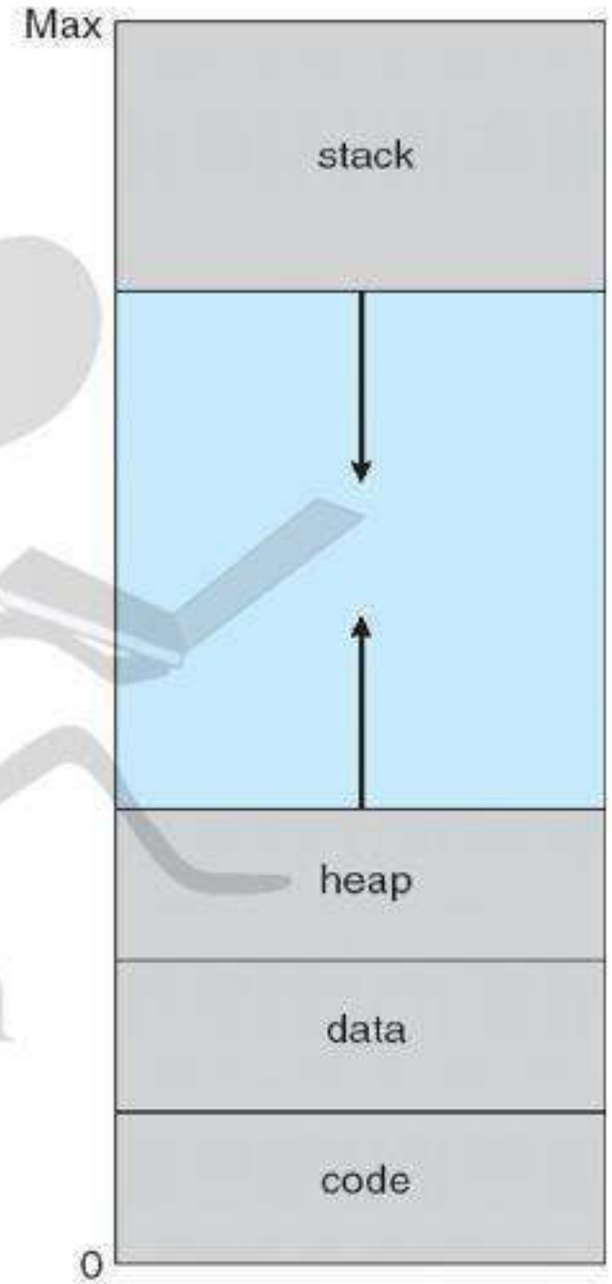
Virtual Memory That is Larger Than Physical Memory



LectureNotes.in
LectureNotes.in

Virtual-address Space

- The virtual address space of a process refers to the logical (or virtual) view of how a process is stored in memory.
- Usually design logical address space for stack to start at Max logical address and grow "down" while heap grows "up"
  - Maximizes address space use
  - Unused address space between the two is hole
    - No physical memory needed until heap or stack grows to a given new page
- Enables **sparse** address spaces with holes left for growth, dynamically linked libraries, etc
- System libraries shared via mapping into virtual address space
- Shared memory by mapping pages read-write into virtual address space
- Pages can be shared during `fork()`, speeding process creation



LectureNotes.in

Demand Paging

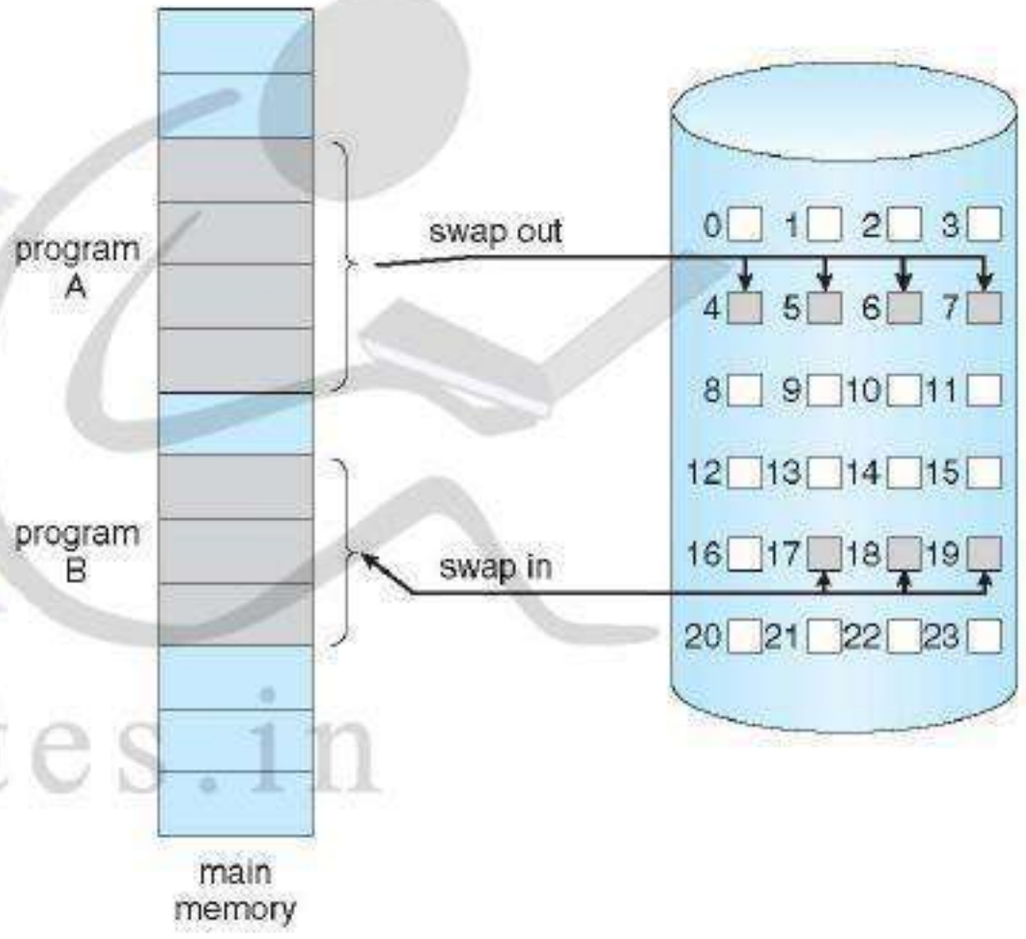
- The virtual memory is implemented by demand paging.
- The criteria of demand paging is “A page will not be loaded into the main memory from the SSD until it is needed”.
- The page will be loaded only when they are demanded during execution.
- A demand paging system is similar to a paging system with swapping where processes reside in secondary memory.
- Generally by using a lazy swapper a page loaded only when it is needed.

LectureNotes.in
LectureNotes.in

LectureNotes.in

Demand Paging

- Bring a page into memory only when it is needed
  - Less I/O needed, no unnecessary I/O
  - Less memory needed
  - Faster response time
  - More users
- Similar to paging system with swapping



- Page is needed ⇒ reference to it
  - invalid reference ⇒ abort
  - not-in-memory ⇒ bring to memory

- **Lazy swapper** – never swaps a page into memory unless page will be needed
  - Swapper that deals with pages is a **pager**

LectureNotes.in

LectureNotes.in

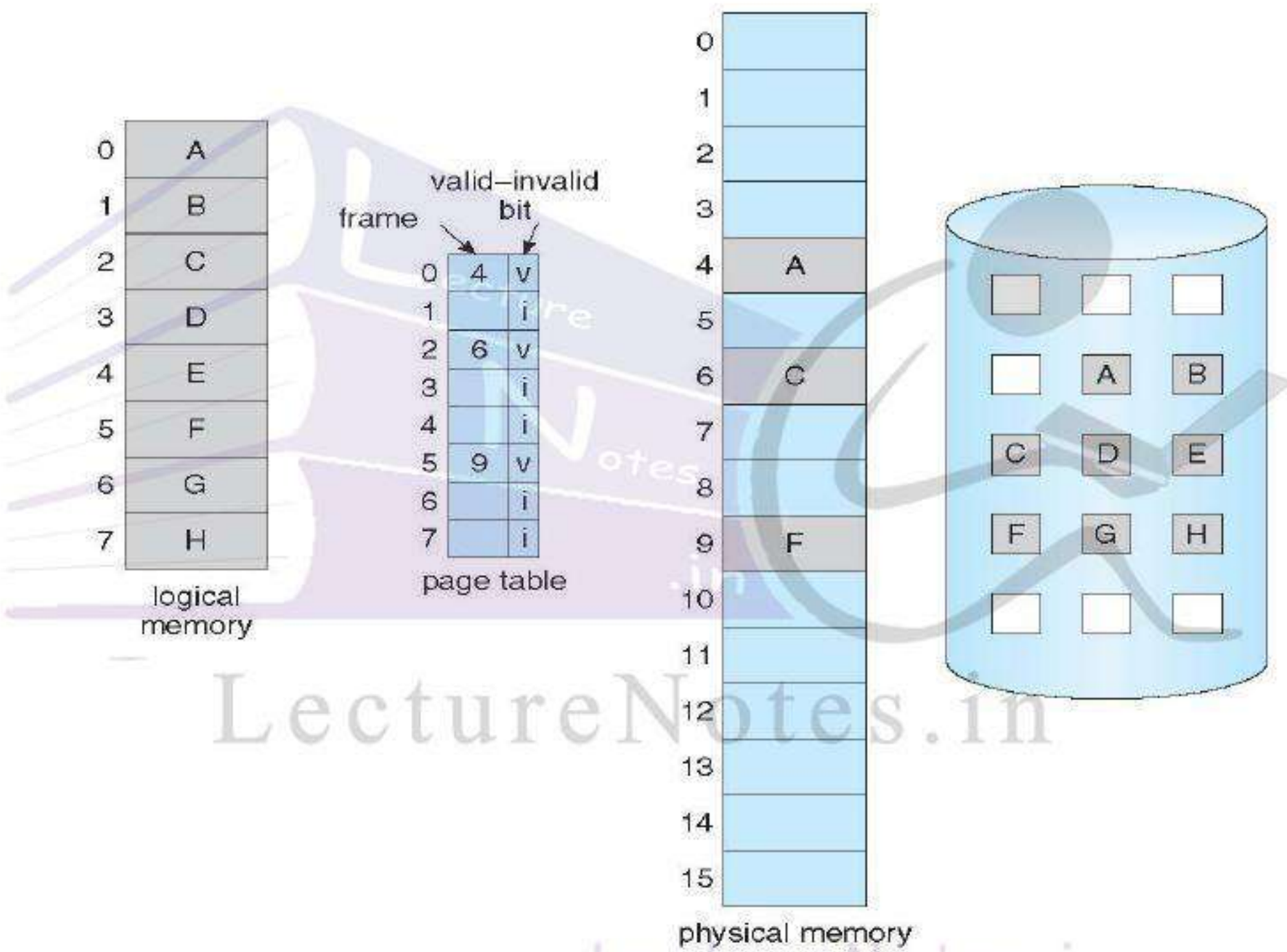
Basic Concepts

- With swapping, pager guesses which pages will be used before swapping out again
- Instead, pager brings in only those pages into memory
- We need hardware support to distinguish between those pages that are in main memory and those pages that are in in the disk.
- This can be done by using a “valid” and “invalid” bit.
- When this bit is “valid” this value indicates that the associated page is both legal and in memory.
- If the bit is “invalid” this value indicates that the page either not valid (i.e. not in the logical address space of the process) or is valid but is currently on the disk.

LectureNotes.in

LectureNotes.in

Page Table When Some Pages Are Not in Main Memory



LectureNotes.in

LectureNotes.in

LectureNotes.in

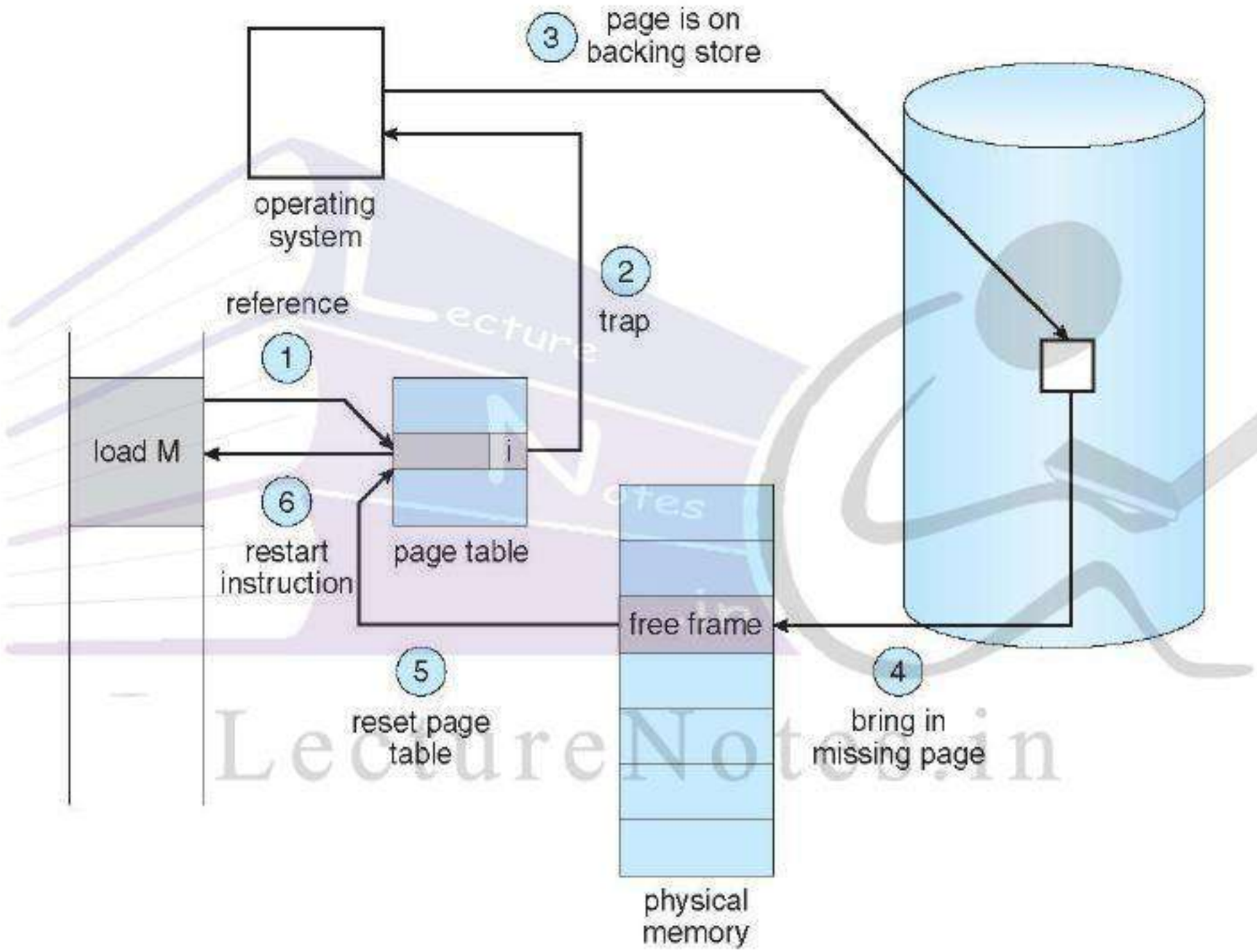
Page Fault

- When a process needs to execute a particular page, that page is not available in the main memory then there will be page fault trap or interrupt.
- It means OS fails to bring the desired page into the memory.
- The procedure for handling the page fault is as follows:
  - The memory address requested is first checked, to make sure it was a valid memory request.
  - If the reference was invalid, the process is terminated. Otherwise, the page must be paged in.
  - A free frame is located, possibly from a free-frame list.
  - A disk operation is scheduled to bring in the necessary page from disk. (This will usually block the process on an I/O wait, allowing some other process to use the CPU in the meantime.)
  - When the I/O operation is complete, the process's page table is updated with the new frame number, and the invalid bit is changed to indicate that this is now a valid page reference.
  - The instruction that caused the page fault must now be restarted from the beginning, (as soon as this process gets another turn on the CPU.)

LectureNotes.in

LectureNotes.in

Steps in Handling a Page Fault



LectureNotes.in

LectureNotes.in

LectureNotes.in

Pure Demand Paging

- When OS sets the instruction pointer to the first instruction of the process, which is on a non-memory resident page, the process immediately faults for the page.

LectureNotes.in

Performance of Demand Paging

- Stages in Demand Paging (worse case)

1. Trap to the operating system
2. Save the user registers and process state
3. Determine that the interrupt was a page fault
4. Check that the page reference was legal and determine the location of the page on the disk
5. Issue a read from the disk to a free frame:
    1. Wait in a queue for this device until the read request is serviced
    2. Wait for the device seek and/or latency time
    3. Begin the transfer of the page to a free frame
6. While waiting, allocate the CPU to some other user
7. Receive an interrupt from the disk I/O subsystem (I/O completed)
8. Save the registers and process state for the other user
9. Determine that the interrupt was from the disk
10. Correct the page table and other tables to show page is now in memory
11. Wait for the CPU to be allocated to this process again
12. Restore the user registers, process state, and new page table, and then resume the interrupted instruction

LectureNotes.in

LectureNotes.in

Performance of Demand Paging (Cont.)

- Three major activities
  - Service the interrupt – careful coding means just several hundred instructions needed
  - Read the page – lots of time
  - Restart the process – again just a small amount of time
- Page Fault Rate $0 \le p \le 1$ (probability of page fault occur =p)
  - if p = 0 no page faults
  - if p = 1, every reference is a fault
- Effective Access Time (EAT)

  EAT = (1 – p) x main memory access time (in nano sec.) + p (page fault overhead + swap page out + swap page in) (In mili second)

1 mili seceond = 10,00000 nano second

LectureNotes.in

LectureNotes.in

Copy-on-Write

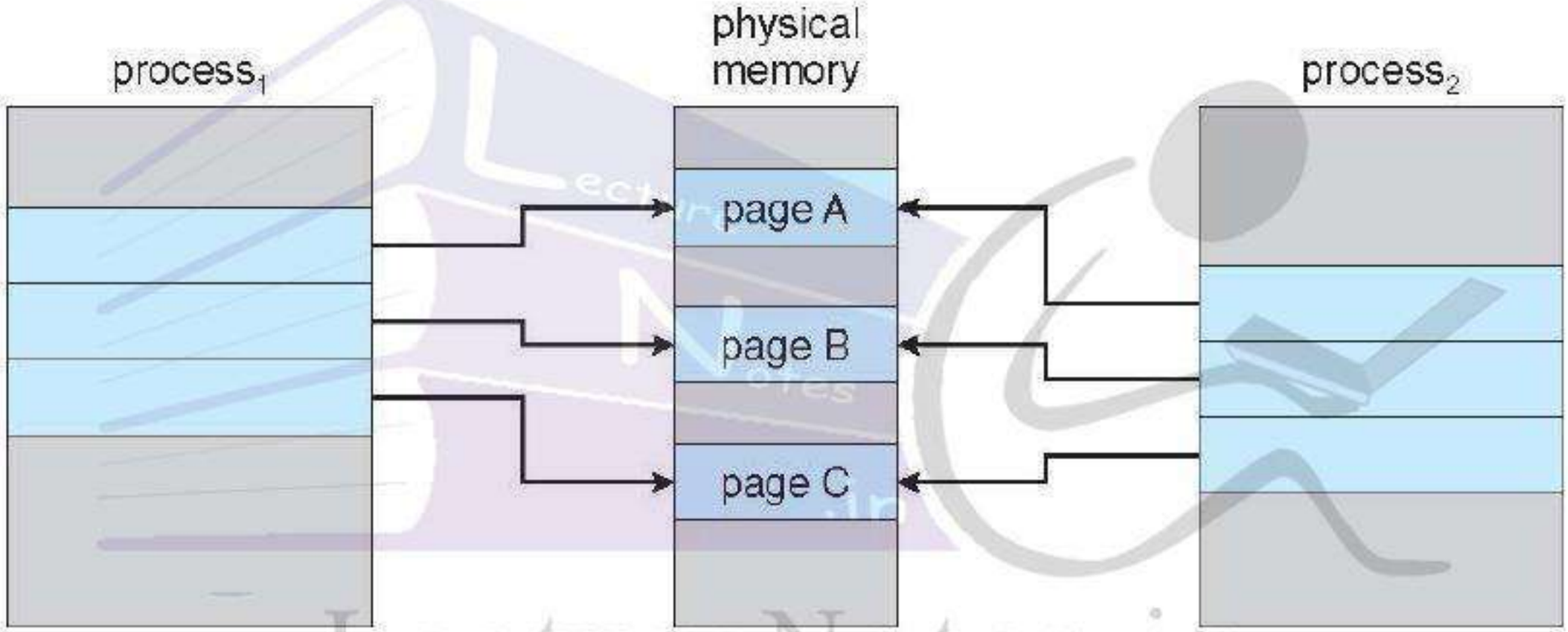
- **Copy-on-Write** (COW) allows both parent and child processes to initially ***share*** the same pages in memory
  - If either process modifies a shared page, only then is the page copied
- COW allows more efficient process creation as only modified pages are copied
- In general, free pages are allocated from a **pool** of **zero-fill-on-demand** pages
  - Pool should always have free frames for fast demand page execution
    - Don't want to have to free a frame as well as other processing on page fault
  - Why zero-out a page before allocating it?
- `vfork()` variation on `fork()` system call has parent suspend and child using copy-on-write address space of parent
  - Designed to have child call `exec()`
  - Very efficient

LectureNotes.in

LectureNotes.in

LectureNotes.in

Before Process 1 Modifies Page C



LectureNotes.in
LectureNotes.in