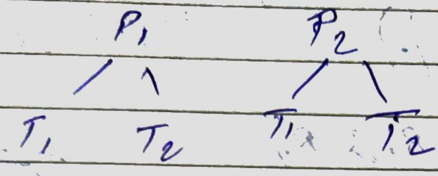
LEC-21 | Deadlock

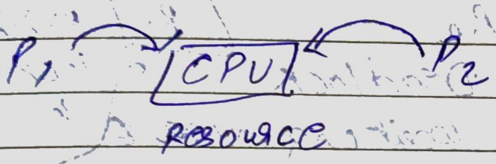
System

R_1 R_2 R_3 R_4	→ Resources

E.g.:- Memory space, CPU, files locks, I/O devices etc.

In deadlock ...

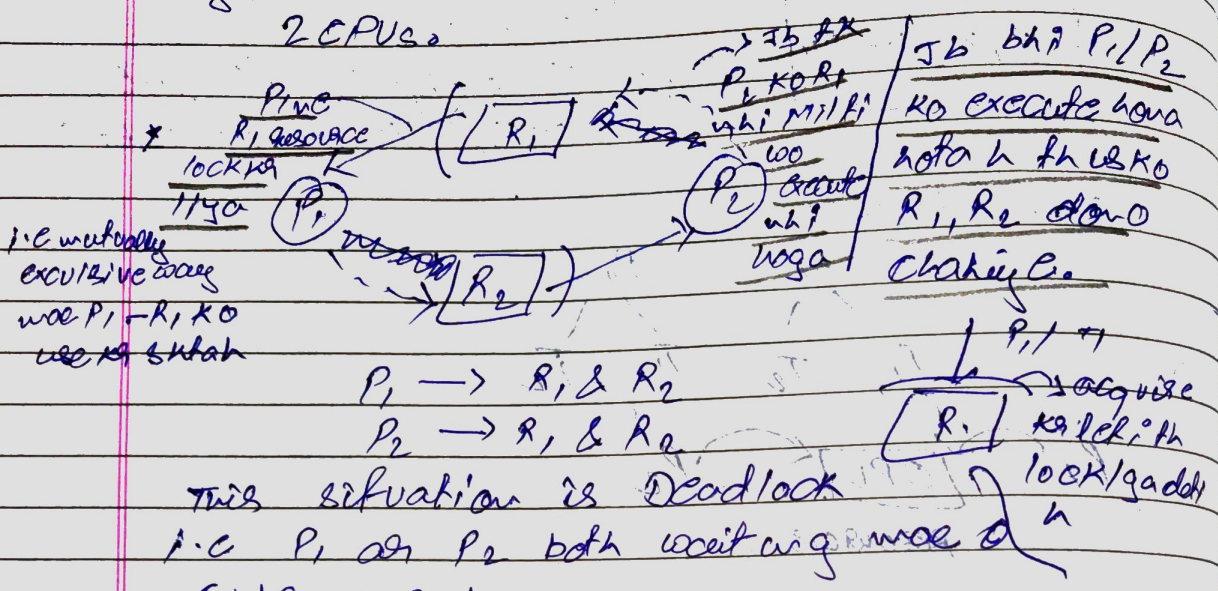




finite no. of resources →
multiple processes → isko aise use karna h taki sara system chlta rhe
so
i.e kahi pe blockage na ho

- In multi programming environment, we have several processes competing for finite no. of resources.
- Process requests a resource (R). if R is not available (taken by other process), process enters in a waiting state. sometimes that waiting process is never able to change its state b/c the resource it has requested is busy (forever), called Deadlock.
- DL is a bug present in the process / thread synchronization method.
  In DL, processes never finish executing & the system resources are tied up, preventing other jobs from starting.

- Single resource can have multiple instances of that E.g., CPU is a resource, and a system can have 2 CPUs.



$P_1 \rightarrow R_1$ & R_2

$P_2 \rightarrow R_1$ & R_2

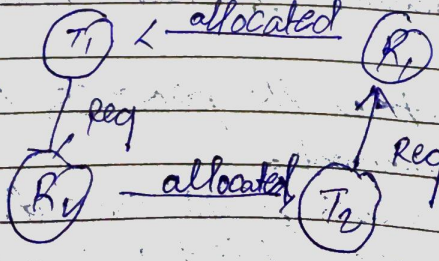
This situation is Deadlock i.e P_1 or P_2 both waiting mode ho chle gye h

* How a process/thread utilize resource?

(a) Request
Req the R_1 if R is free lock it, else wait till it is available.

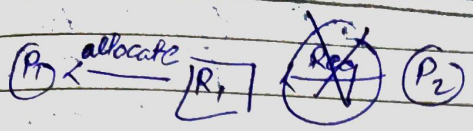
(b) use

(c) Release
Release resource instance & make it available for other processes.

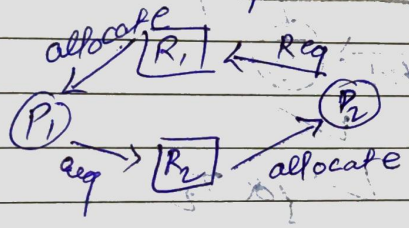


* Deadlock necessary condition : 4 conditions should hold simultaneously.

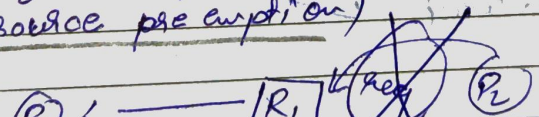
(a) Mutual exclusion
- only 1 process at a time can use the resource, if another process req that resource, the requesting process must wait until the resource has been released.



(b) Hold & wait
- A process must be holding at least one resource & waiting to acquire additional resources that are currently being held by other processes.



(c) No preemption
- resource must be voluntarily released by the process after completion of execution
(No resource preemption)

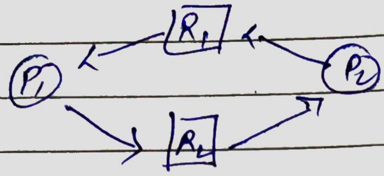


P1 job tk execute nhi hota tb R1 tb release nhi kar sakta
or agar P1 ne req. kiya ↳ R1 ko, tb R2 ko allocate nhi hona chahiye

(d) Circular wait

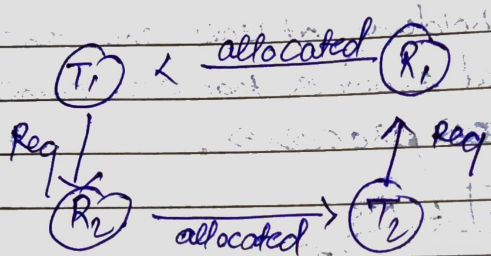
- A set $\{P_0, P_1, \dots P_n\}$ of waiting process must exist such that P_0 is waiting for a resource held by P_1, P_1 is waiting for a resource held by P_2 & so on.

i.e circular way of waiting



P_1 ek aise res. ko le ke baithi h jo P_2 leke baitha h & vice versa

Ex –



1. Mutual exclusion ✓
2. hold & wait ✓ for T_2
3. No preemption ✓

Mera system es example mai kahta h wo R_1 ko chode

R_1 jb tak release nhi hoga jb tk T_1 bole du ja.

4. Circular wait ✓

~~Methods for handling~~

Resource allocation graph.

① vertex — Process vertex (P)
→ Resource vertex [R]

② Edges. — Assign [R1] allocate → (P1)
→ Request (P1) Req [R2]

multiple instance.

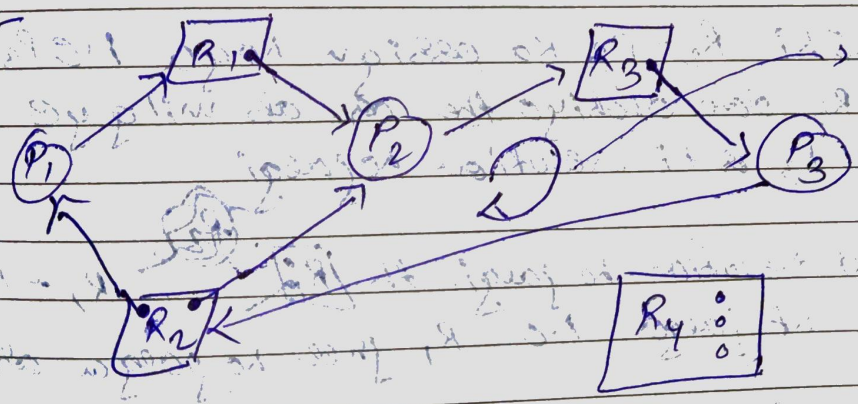
[R0] (dot)
↓
3 instance

CPU
↓ 4 core
4 (CPU)

[R]
↓
single instance

This graph for system representation

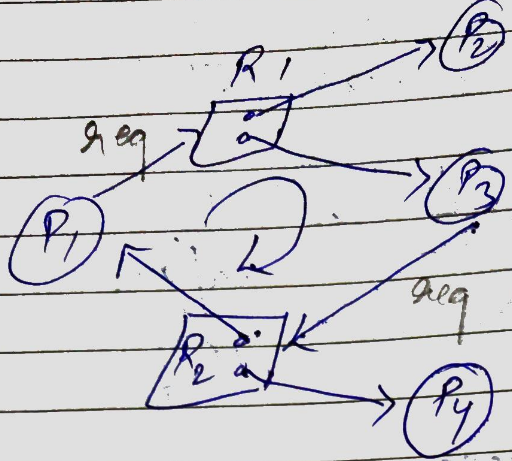
Ex-



Cycle dikh rha h to (may be) deadlock ho agar cycle vhi h to pka deadlock vhi hoga.

By def of RAG
→ No cycle ⇒ No DL
→ RAG Cycle ⇒ May be DL

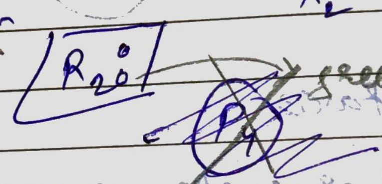
Q - cycle hogi but DL nhi hoga



cycle th h

t=0

after sometime
P4 releases R2 then



R2 - P4 edge ko hta deta h

ar phir R2
(P3) ko assign ho jaega

ar jaise hi R2 - P3 ko assign hoga i.e P3 ko
R1 ar R2 dono chahiye the ab mil gye
are to P3 ki execution ho jaegi

ar (P2) ki execution ho jaegi th [R1] R1 - P2
ka edge ht jaega i.e R1 free ho jaega ar
P1 ko R1 bhi mil jaega.

Cycle was present
But NO DL

Methods for handling Deadlocks.

(a) use a protocol to prevent & avoid DL, ensuring that the system will never enter a deadlocked state.

(b) Allow the system to enter a deadlocked state, detect it and recover it.

(c) ~~Ignore the problem that DL never occur, the system can use either a DL prevention or~~

(c) Ignore the problem altogether and pretend that deadlocks never occur in system. Ostrich algorithm aka DL ignorance.

Deadlock Prevention

by ensuring at least one of the necessary cond^n cannot hold.
agar ek bhi DL cond^n false hogya to DL nhi hoga.

(a) Mutual exclusion → prevent it
- use locks (mutual exclusion) only for non shareable resources
- Shareable resources like read only files can be accessed by multiple processes / threads.

- However, we cannot prevent DLs by denying the mutual exclusion cond^n, b/c some resources are intrinsically non shareable.

i.e avoid karunga mutual exclusion kam se kam jagah pe

=> jo non shareable resource h kewal unhi mae mutual exclusion h use karunga i.e locks ka i.e CS => memory space jaha baari baari se process use karein

Read only file
=> koi write nhi kar rha => inconsistency aaegi nhi
th isse kyu C.S banau, kyu alag reso. banau jo non shareable h
=> Shareable res => ~~no locks~~

=> no locks

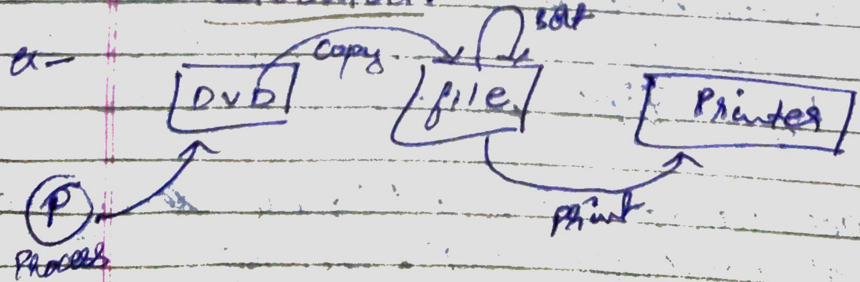
(b) Hold & wait

- To ensure H & W cond" never occur in the system we must guarantee that, whenever a process req. a resource, it doesn't hold any other resource.

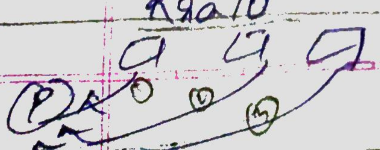
- Protocol (A) can be, each process has to req & be allocated all its resources before its execution.

- Protocol (B) can be, allow a process to request resources only when it has none. It can request any additional resource after it must have released all the resources that it currently allocated.

eg-



start hi mae ye 3 teen ke teen resource allocate kralu
allocation ke baad ye task
copy → sort → print ye tina kaam baad mae dedo



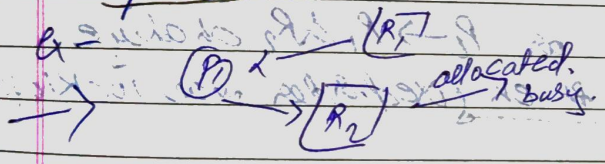
⇒ Jb copy ko phi nogi DVD – file, tb bhi pointer ko le ke baitha hu ko
extra copy wale lag jage, tb bhi pointer wait hold ka ka baitha hu.

Page No.
Date: / /

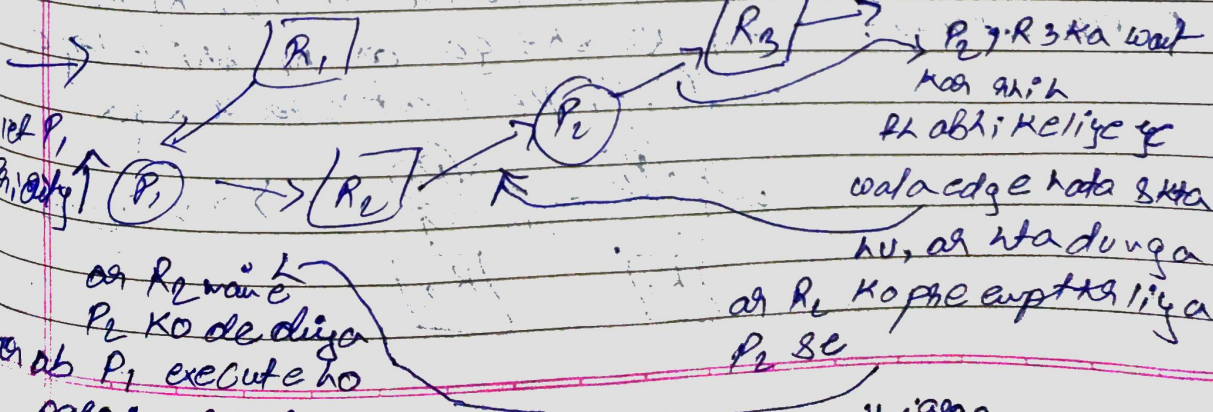
(c) **NO preemption**

- If a process is holding some resources & req. another resource that cannot be immediately allocated to it, then all the resources the process is currently holding are preempted. The process will restart only when it can regain its old resources, as well as the new one that it is requesting. (i.e. Lock may occur).

- If a process request some resources, we first check whether they are available. If yes, we allocate them. If not, we check whether they are allocated to some other process that is waiting process & allocate them to requesting process.



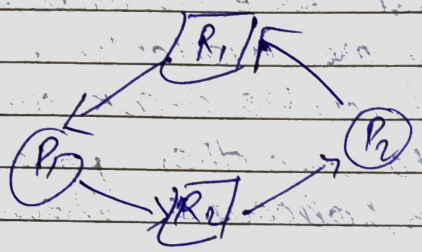
P1 ko ek interrupt jaega abhi R2 jitne bhi P1 ne currently resources hold kiye h (P1 — R1) R1 ko release kr do
⇒ P1, R1 ko release kr dega
or wait karega jb R1 or R2 saath me avail. ho jaenge. Abhi to R2 free hoga
Jb R2 free hoga tb (P1) done hua ko saath me req karega or chahega dono saath me allocate ho jaega



(d) **Circular wait**

- To ensure that this cond^n never holds is impose a proper ordering of resource allocation.

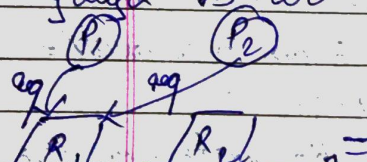
- P_1 and P_2 both require R_1 & R_2, locking on these resources be like, both try to lock R_1 then R_2. by this way which ever process first locks R_1 will get R_2.



Isko rokne keliye ek fixed order lagayenga

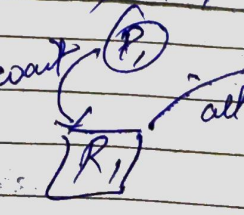
Sol^n $P_1 \rightarrow R_1 \& R_2$ chaiye or $P_2 \rightarrow R_1 \& R_2$ chahiye, Ib P_1 or P_2 lock krne jayenge tb ek fixed order me lock hoga

order phle done thread. R_1 ko lock karenge Ib unko R_1 mil jayega tb woo R_2 pe jayegi. $R_1 \longrightarrow R_2$ (R_1 already)



let assume P_2 ko R_1 phle mil gaya

$\Rightarrow$ (P_1), R_2 ko req nhi karega woo R_2 pe wait karta rhega or ab (P_2) ko R_1 mil chuka woo ab (P_2), R_2 ko req krr ske ga or R_2 bhi le lega or (P_1), R_1 pe wait karta rahega



$\Rightarrow$ No cycle

LEC-22 | Deadlock Avoidance.

Page No.
Date:

DL avoidance:

* Idea is, the kernel be given in advance info concerning which resources will use in its lifetime.
By this system can decide for each req whether the process should wait. To decide whether the current request can be satisfied or delayed, the system must consider the resources currently available, resources currently allocated to each process in the system & the future requests & releases of each process.

DL avoidance ke case wae hme pta hota h
1. Current state → no. of process,
   wat need of R. of each processes
   Currently allocated amount of Res. to each process
   Max amount of each Res.

ab target ye h ki ab kis tarah schedule kare process ko or Res. ko handle kare ki jo system h wo safe state wae rhe
↓
kis tarah sched. or handle karu ki system DL free rhe

- Schedule process and its resources allocation in such a way that DL never occur.

- Safe state: A state is safe if the system can allocate resources to each process (up to its max.) in some order & still avoid DL
A system is safe state only if there exists a safe sequence.

- In an unsafe state, the OS cannot prevent processes from requesting resources in such a way that any DL occur. It is not necessary that all unsafe states are deadlocks; an unsafe state may lead to a DL.

- The main key of DL avoidance method is whenever the req. is made for resources then the req must only be approved only in the case of the resulting state is a safe state.

- In a case, if the system is unable to fulfill the req of all processes, then the state of system is unsafe.

- Scheduling algo using which DL can be avoided by finding safe sat state. (Banker's algo.)

(*) Banker Algorithm

when a process req. a set of resources, the system must determine whether allocating these resources will leave the system in a safe state. If yes, then the resource may be allocated to the process. If not, then the process must wait till other processes release enough resources.

Ex - Check system safe / unsafe

Resource initially

Process (P)	A	B	C	A max need	B	C
P_1	0	1	0	7	5	3
P_2	2	0	0	3	2	2
P_3	3	0	2	9	0	2
P_4	2	1	1	4	2	2
P_5	0	0	2	5	3	3
Total:	7	2	5			

Total already → Given hai

Total res. of A → 10, B → 5, C → 7

Avail at the start = Total res. − Total already

$10 - 7 = 3$ for (A)

$5 - 2 = 3$ for (B)

$7 - 5 = 2$ for (C)

Remaining need = max need − allocated

$7 - 0 = 7$

A	B	C	Available
3	3	2	

A	B	C	Remaining need	
7	4	3		P_1
1	2	2		P_2
6	0	0		P_3
2	1	1		P_4
5	3	1		P_5
				P_5

⇒ P_2 →

P_2 ko allocated res. free hoge, i.e

A	B	C
2	0	0

will be free & can be added to avail res. after P_2 execution

Now Available

A	B	C
3	3	2
+ 2	0	0
5	3	2

P_2 allo. res. now added

A	B	C	Remaining	
7	4	3		P_1
~~1~~	~~2~~	~~2~~		P_2
6	0	0		P_3
~~2~~	~~1~~	~~1~~		P_4
5	3	1		P_5

⇒ $P_2 \rightarrow P_4 \rightarrow$

Now P_4 res. free ⇒ P_4 allo. res. are now added.

P_4 req

Avail

A	B	C
5	3	2
+ 2	1	1
7	4	3

A	B	C	Remaining	
7	4	3		P_1
~~1~~	~~2~~	~~2~~		P_2
6	0	0		P_3
~~2~~	~~1~~	~~1~~		P_4
5	3	1		P_5

⇒ $P_2 \rightarrow P_4 \rightarrow P_5 \rightarrow$

P_5 ko schedule kr skte h

Page No.
Date:

Now P_3 res. free & its all. res. are added

Avail

7	4	3
+0	0	2
7	4	5

rem

~~7 4 3~~ P_1
~~1 2 2~~ P_2
6 0 0 P_3
~~2 1 1~~ P_4
~~5 3 1~~ P_5

⇒ ($P_2 \to P_4 \to P_5 \to P_1$)

⇒ ab P_1 ke res. free hogye.
ab P_1 ke all. res. add honge. → avail mai.

Avail

7	4	5
+0	1	0
7	5	5

rem

~~7 4 3~~ P_1
~~1 2 2~~ P_2
6 0 0 P_3
~~2 1 1~~ P_4
~~5 3 1~~ P_5

⇒ $P_2 \to P_4 \to P_5 \to P_1 \to P_3$

Avail.

A	B	C
3	3	2
5	3	2
7	4	3
7	4	5
7	5	5
+3	0	2
10	5	7

P_3 allo. res

10 5 7 ↓ A B C

Avail bhi total jitne ho gye h mtlb ab scheduling kr di h or ye particular safe state h.

Ye un safe tb hota jb maan 10 ~~7 4 3~~ P_1 → 6 ke jagh hota

Avail

7 5 5

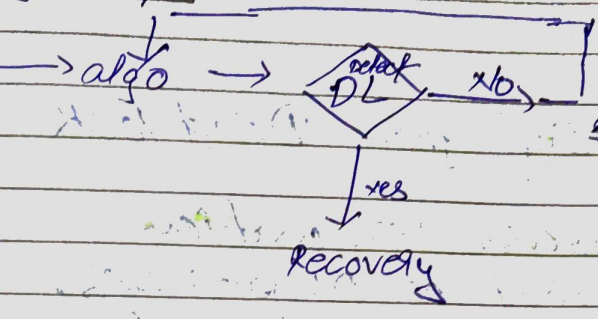
~~7 4 3~~ P_1
~~1 2 2~~ P_2
8 6 0 0 P_3
~~2 1 1~~ P_4
~~5 3 1~~ P_5

ab P_3 schedule nhi ho pati th mai 8 0 1 extra h

schedule which can put us in a safe schedule hai => unsafe state.

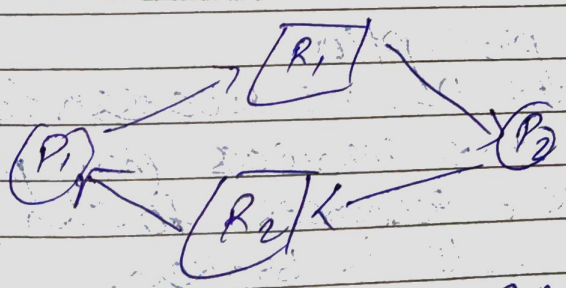
Deadlock Detection

System has ~~not~~ implemented DL prevention & a DL avoidance technique, then they may employ DL detection & recovery technique.



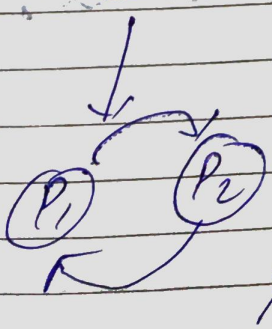
(a) Single instance of Each resource type (wait for graph method).

→ A DL exist in the system if and only if there is a cycle in the wait for graph. In order to detect the DL, the system needs to maintain the wait for graph & periodically system invokes algo that searches for cycle in wait for graph.



version

RAG → wait for graph



wait for graph banane ke

cycle => DL ✓

no cycle => DL ✗

(b) Multiple instances for each resource type.

- Bankers Algorithm

↳ Safe seq avail ⇒ No DL

If, No Safe seq avail ⇒ DL

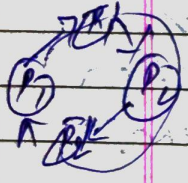
(4) Recovery from Deadlock

(a) Process Termination

- Abort all DL processes.
- Abort one process at a time until DL cycle is eliminated.

Kill P_1 & P_2 of DL cycle

Previous example



jiski 100 priority process usko bhi kill karke

If Kill P_2 ⇒ R_1 free

⇒ $P_1 \rightarrow R_1$

⇒ NO DL

(b) Resource preemption

To eliminate DL, we successively preempt some resources from processes & give these resources to other processes until DL cycle is broken.