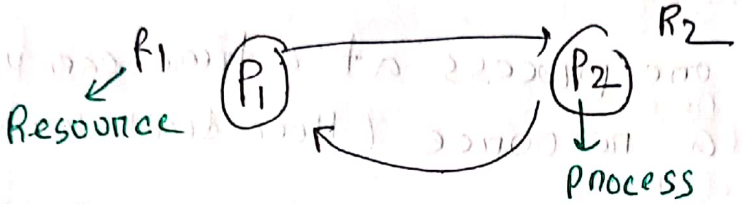
what is deadlock ?

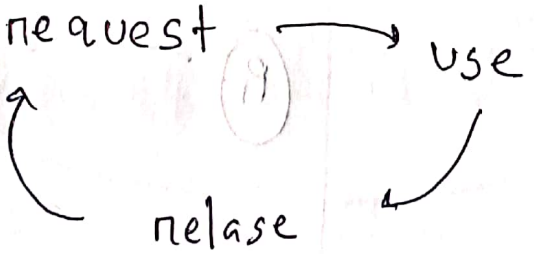
→ deadlock is a situation that occurs in OS when any process enters a waiting state because another waiting processes is holding demanded resource



two or more process waiting for each other.

Resource type
(R)
→ cpu cycle
→ memory space
→ I/O device
→ semaphones

what process do with resource ?



When Deadlocks occurs ?

Deadlock Detection

→ if the is four condition arise

(1) Mutual exclusion :-

only one process at a time can use a resource. (Here deadlock can occurs)

(if resource are shane able with multiple process deadlock never happens)

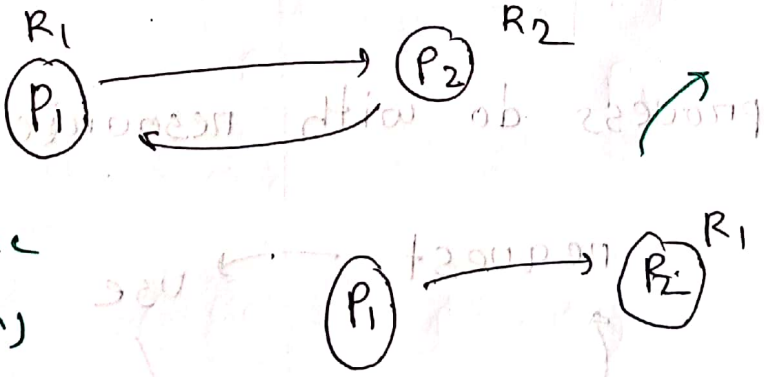
(2) Hold and wait :-

a process holding atleast one resource is waiting to acquire additional resources held by another process

Here, deadlock occurs

Hold and wait

Here P_1 Holding resource R_1 and waiting for resource R_2



No deadlock

No Hold only wait.

Here P_1 holding no resource but waiting for resource R_1

(3) No preemption :-

a nesource can be nelased only by the process holding it, aflen the process has complete.

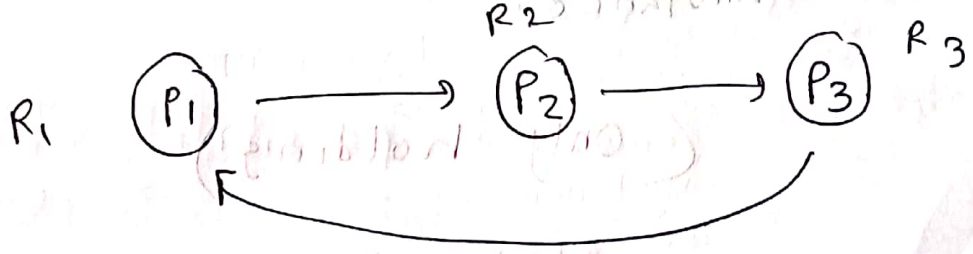
P_1

∴ P_1 cannot nelase R_1 untill finish. but P_1 will not finish because itn waiting R_2. Hene deadlock can occurs.

∴ if pneemptive, No deadlock, ez P_1 can nelase R_1 with finishing its job

(4) Circulan wait :-

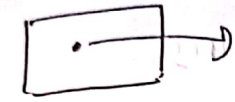
one process is waiting for the nesource is held by seond process, which also waiting fon the nesource held by thind process. This waiting continve untill last process is waiting for a nesource held by finst process. This is called cinculan wait



Resource Allocation Graph

Process node :- ○

resource node :- □

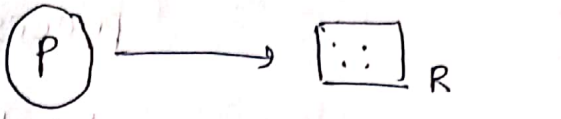


resource node

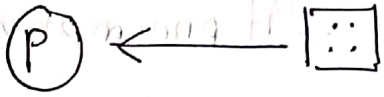
instance of resource node

Here; 1

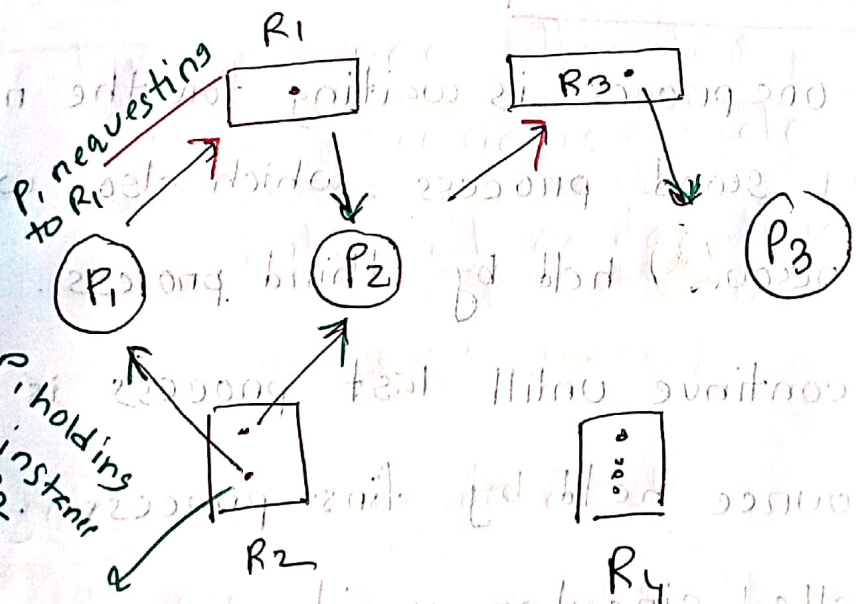
→ Here instance is 1



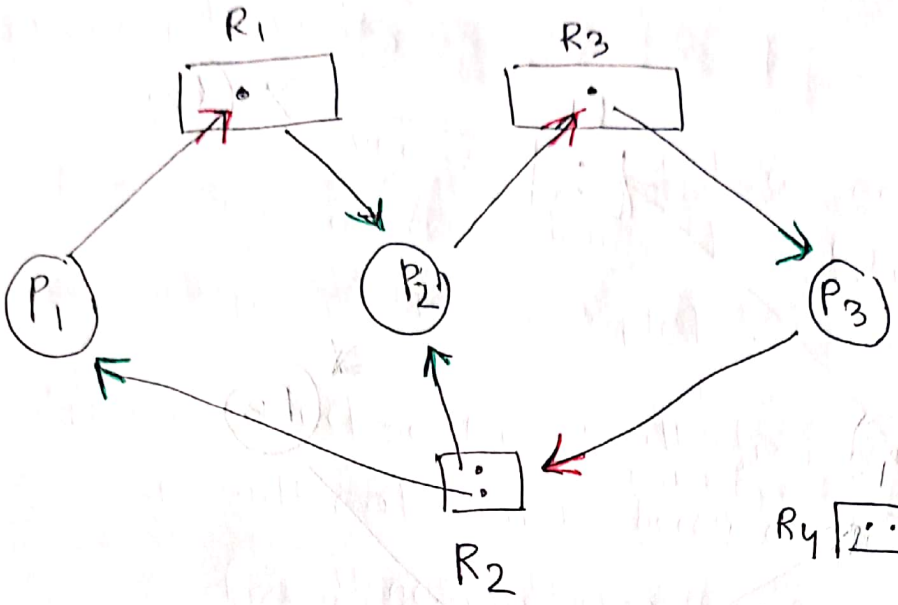
Process, P requesting instance of R



Process, P holding an instance of R



(Only holding)



cycle & deadlock

→ requesting
→ holding

(Hold and wait)

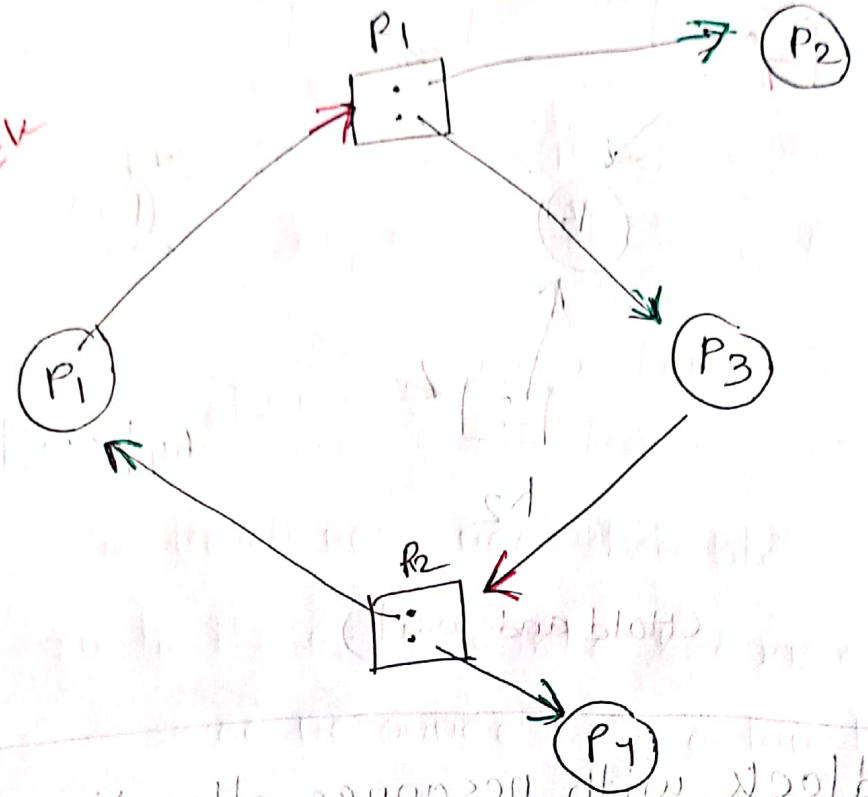
Deadlock with resource allocation Graph

P_1 waiting for instance of R_1, while holding an instance of R_2

P_2 waiting for instance of R_3, while holding an instance of from both R_1, R_2

P_3 waiting for instance of R_2, while holding an instance of R_3

cycle but no deadlock



In this resource allocation graph, there is a cycle but still no deadlock

no cycle ⟶ no deadlock

if cycle $\xrightarrow{\text{one instance resource}}$ deadlock

$\xrightarrow{\text{several instance resource}}$ possibility of deadlock (not sure)

Methods for handling deadlocks

1. deadlock prevention (Never enten deadlock)
2. deadlock avoidance (")
3. allows a system to enten deadlock then recover.
4. Ignone problem & pretend deadlock never occured.
   (LinuX)

Deadlock prevention

∴ if we can prevent any of the deadlock detection then there will be no deadlock.

Pratically prevention is not possible

① Preventing Mutual Exclusion :-

make all resource shanable

∴ pratically not possible because some resource are naturally non-shaneable (write only files)

② Hold & write :-

must guarantee that whenever a process request a resource, it does not hold any other resource

low resource utilization

method-1

require process to request and be allocated all its resource before it begins execution

method-2

allow process to request resource only when the process has none alloacted to it

③ No preemption :-

→ if a process is holding some resources and requests for another resource, at first the process has to release all allocated / holding resource, the requested resource will be allocated to it.

→ Process will be restarted only it can regain its old resources as well as new ones that is requesting.

→ cannot apply to locks & semaphores

④ Circular wait :-

impose a total ordering of all resource types & require that each process request resource in an increasing number.

Deadlock avoidance

(1) Each process declares the maximum number of resources of each type that it may need.

(2) Resource allocation state is defined by the number of available & allocated process and maximum demands of the process.

(3) deadlock avoidance algorithm examines resource allocation state so that there can never be a circular wait condition

what is safe state?

→ a state of system is called safe if the system can allocate all the resources requested by all the processes without entering into deadlock.

→ system is in safe state if exist a sequence $\langle P_0, P_1, P_2 \ldots P_n \rangle$ of all process in the system such that for each P_i, the resource that P_i can still requested can be satisfied by currently available resources + resources held by P_j with $j < i$

one resource with 12 instance

Process	Maximum need	current allocation or instance	need
P_0	10	5	5
P_1	4	2	2
P_2	9	2	7

∴ available resource = $12 - (5+2+2) = 3$

now available resource = 3

∴ P_1 will enter first (P_1 need (2) $\leq$ available resource (3))

∴ After P_1 finish. it will release it resources

∴ now total available resource = $3+2 = 5$

∴ P_0 will enter. (P_0 need 5 (5) $\leq$ available resource (5))

∴ after P_0 finish it will leave its resource

∴ now total available resource.

$= 5 + 5 = 10$

∴ now P_2 enters; P_2 needs (7) $\leq$ total available resource (10)

∴ P_2 terminates.

∴ All processes got their desire resource without deadlock.

∴ state sequence = $\langle P_1, P_0, P_2 \rangle$

∴ system is safe here

one nesounce with 12 instance

Process	Maximum need	cunnent allocation	need
P_0	10	5	5
P_1	4	2	2
P_2	9	3	6

∴ available nesounce $= 12 - (5+2+3)$
$= 2$

∴ Now P_1 entens cz (P_1 needs ≤ available nesounce)
2 ≤ 2

after P_1 finish; total available nesounce $= 2+2 = 4$

∴ Now none P_0 on P_2 can enten

because

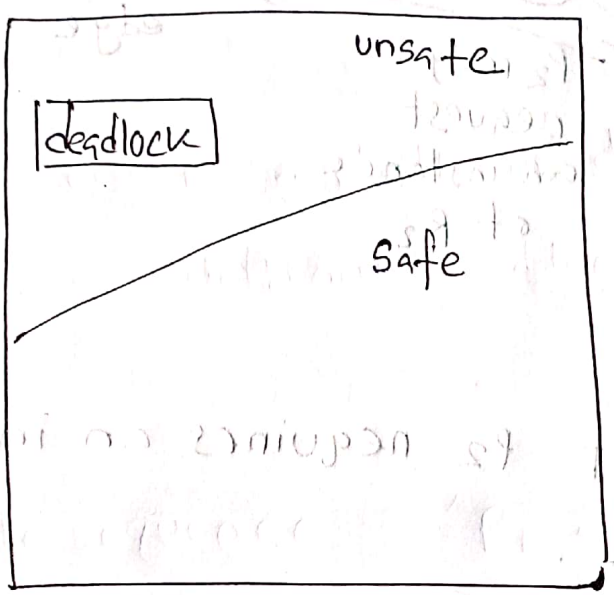
P_0/P_2 needs > available nesounce

∴

Unsafe state

safe state → no deadlock

unsafe state → possibility of deadlock



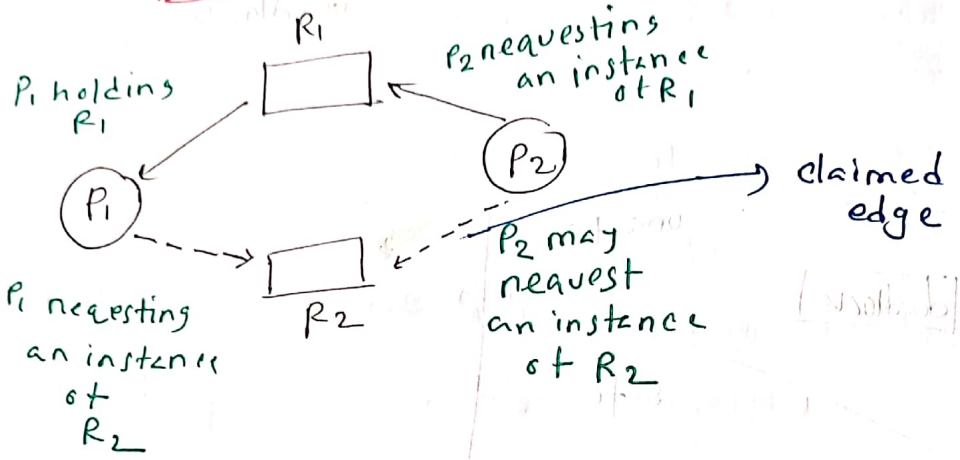
Avoidance Algorithm

single instance → Using resource allocated graph

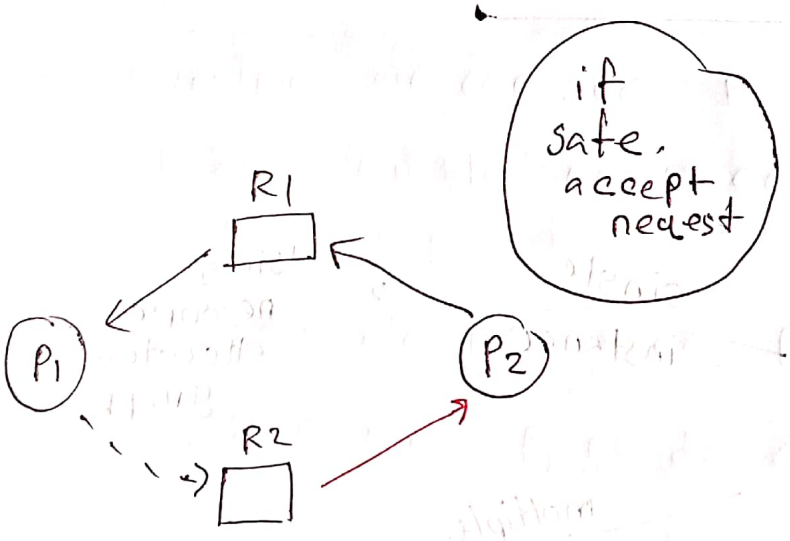
multiple instance → Banker's Algorithm

avaiding deadlock using Resource - Allocation graph

(no cycle - deadlock free)



what happen if P2 requires an instance of R2.



cycle, deadlock possibility

∴ unsafe

deny the request of P2 for R2

① we will temporarily accept the request and give it to P2

② Now check the system is safe or unsafe

③ if unsafe, will deny the request & undo change

Deadlock avoidance with Banker's Algorithm

n = number of processes

m = number of resource type

(Given all)

1. Available
2. Max
3. Allocation
4. need = max − allocation

Q) 5 process P_0 through P_4

∴ 3 resource type

A (10 instance) B (5 instance) C (7 instance)

Process	Allocation			Max			(max − allocation) Need			Available		
	A	B	C	A	B	C	A	B	C	A	B	C
P_0	0	1	0	7	5	3	7	4	3	3	3	2
P_1	2	0	0	3	2	2	1	2	2			
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			

∴ Find safe state sequence?

(Multiple ans; but ~~successively~~ find ans for actual/convention ans)

Ans:- safe sequence :-

$\langle P_1, P_3, P_0, P_2, P_4 \rangle$

3 3 2

Process	Allocation	Need	Available
	A B C	A B C	A B C
P_1	2 0 0	1 2 2	5 3 2
P_3	2 1 1	0 1 1	7 4 3
P_0	0 1 0	7 4 3	7 5 3
P_2	3 0 2	6 0 0	10 5 5
P_4	0 0 2	4 3 1	10 5 7

= [(A B C)] resource type in given

Explain;

In Question, we can see P_0 is in complete

∴ it has available resource, A(3) B(3) C(2)

∴ In this avikble resource, P_1 can complete its job, cause P_1 needs < available resource

1 2 2 < 3 3 2

∴ P_1 enters.

∴ after P_1 finishits job

∴ it releases its allocated resources.

∴ Available resource =

	A	B	C	
	3	3	2	(available past)
+	2	0	0	(P_1 allocation)
	5	3	2	

∴ P_3 enters cause P_3 needs ≤ available resources

0 1 1 ≤ 5 3 2

∴ after P_3 finish its releases its resources.

	A	B	C	
∴ Now available resource =	5	3	2	
+	2	1	1	(P_3 alloc.)
	7	4	3	

∴ Now P_0 can enter cause P_0 needs ≤ available resource

7 4 3 ≤ 7 4 3

Now Question why P_0. why not P_4 cause we are traversing serially?

Ans:- it take P_4 after P_4 finish available resource

after P_0 finish. available resource

A	B	C
7	4	3
+0	+1	+0
7	5	3

NOW, P_2 enters cause P_2 needs < Available resource

6 0 0 < 7 5 3

∴ after P_2 finish. available resource

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

NOW P_4 enters cause P_4 needs < available resource

NOW available resource

A	B	C
~~7~~	~~5~~	
10	5	5
0	0	+2
(10)	(5)	(7)

there are multiple ans :-

< P_1, P_3, P_4, P_0, P_2 > ~~(serially best/ans)~~

< P_3, P_1, P_0, P_2, P_4 >

< P_3, P_1, P_4, P_0, P_2 >

actual / convention

why coose this seqence

P_1, P_3, P_0, P_2, P_4

if P_1, P_3 can both got chance on their needs < available resource

∴ we will choose P_1 first cause it comes first in process list

same P_0, P_4

∴ if P_0 / P_4 needs < available resource

∴ we choose P_0 first.

Q2 :- In the diagram

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

(1) can request for (1,0,2) by P1 be granted?

(2) can request for (3,3,0) by P4 be granted?

(3) can request for (0,2,0) by P0 be granted.

(1) Ans:- $P_1(1, 0, 2)$

First update this values:

Allocation of P_1 :-

$$\begin{array}{r} 2\ 0\ 0 \text{ (past)} \\ +1\ 0\ 2 \text{ (new)} \\ \hline 3\ 0\ 2 \end{array}$$

Need of P_1 :-

$$\begin{array}{r} 1\ 2\ 2 \text{ past} \\ -1\ 0\ 2 \text{ newest} \\ \hline 0\ 2\ 0 \end{array}$$

available :-

$$\begin{array}{r} 3\ 3\ 2 \text{ past} \\ -1\ 0\ 2 \text{ newest} \\ \hline 2\ 3\ 0 \end{array}$$

Now the diagram:

	Allocation			Need			Available		
	A	B	C	A	B	C	A	B	C
P_0	0	1	0	7	4	3	2	3	0
P_1	3	0	2	0	2	0			
P_2	3	0	2	6	0	0			
P_3	2	1	1	0	1	1			
P_4	0	0	2	4	3	1			

Now check there any safe sequence exist on not. if exists the request of P_1 will be granted.

∴ Yes there is a safe sequence

$\langle P_1, P_3, P_0, P_2, P_4 \rangle$

∴ so the request of P_1 will be granted.

2 3 0

Process	Allocation	Need	Available
	A B C	A B C	A B C
P_1	3 0 2	0 2 0	5 3 2
P_3	2 1 1	0 1 1	7 4 3
P_0	0 1 0	7 4 3	7 5 3
P_2	3 0 2	6 0 0	10 5 5
P_4	0 0 2	4 3 1	10 5 7

P_4 (3, 3, 0) Diagram-1

continue from last available 2, 3, 0

(2)

Allocation P_4 :- 0 0 2
+ 3 3 0
3 3 2

Need P_4 :- 4 3 1
- 3 3 0
1 0 1

available of P_4 :- 2 3 0
-3 3 0
-1 0 0

∴ available ≤ request

∴ so we find the system is unsafe

∴ we rolled back to old state

(3)

P_0 (0, 2, 0)

Write
=
continue
from last
table
=

allocation :-

$$\begin{array}{r} 0\ 1\ 0 \\ +\ 1\ 0\ 2\ 0 \\ \hline 0\ 3\ 0 \end{array}$$

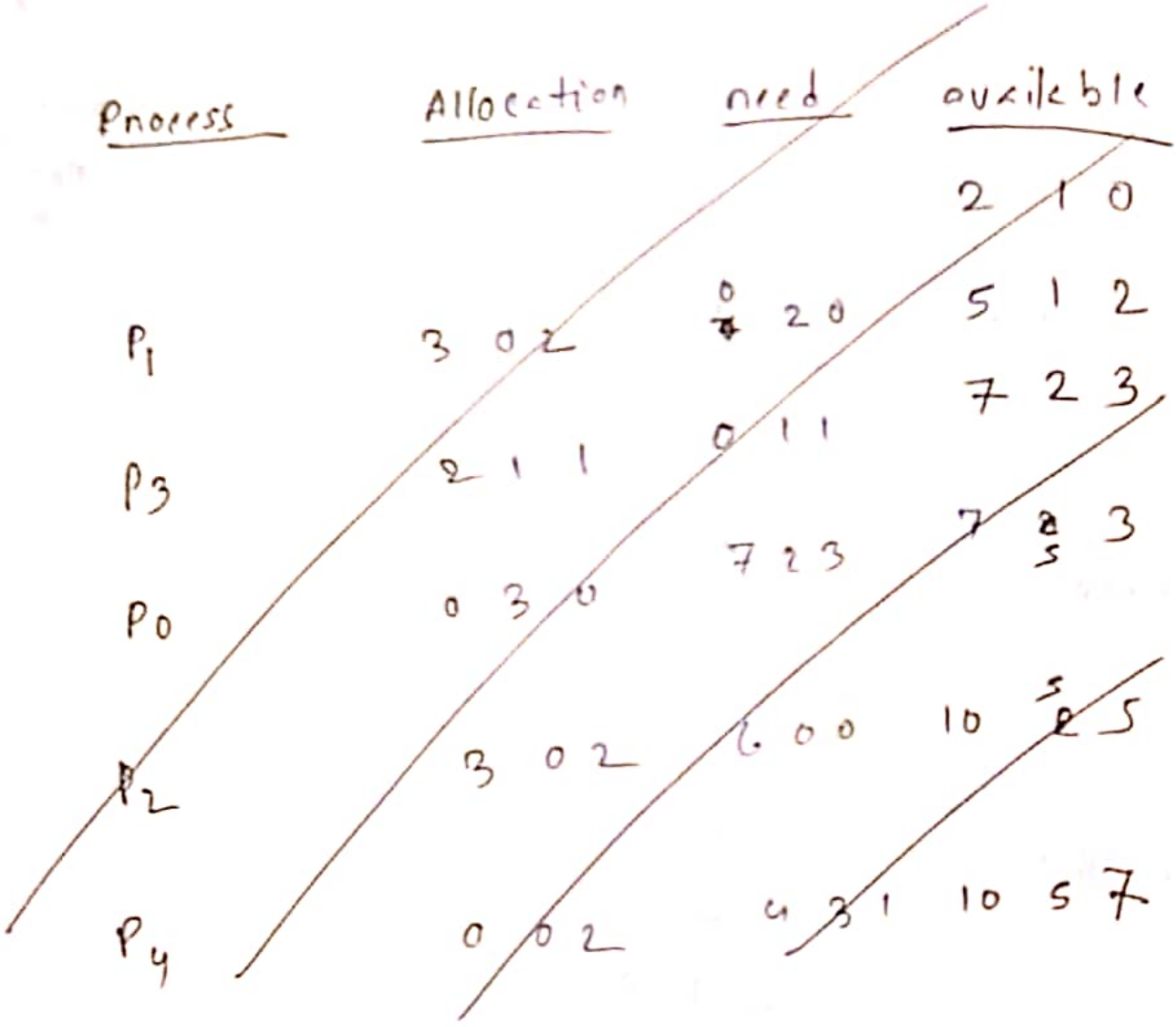
need :

$$\begin{array}{r} 7\ 4\ 3 \\ -\ 0\ 2\ 0 \\ \hline 7\ 2\ 3 \end{array}$$

available :-

$$\begin{array}{r} 2\ 3\ 0 \\ -\ 0\ 2\ 0 \\ \hline 2\ 1\ 0 \end{array}$$

Process	Allocation	need	available
	0 3 0	7 2 3	2 1 0
P_0	3 0 2	0 2 0	
P_1	3 0 2	6 0 0	
P_2	2 1 1	0 1 1	
P_3	0 0 2	4 3 1	
P_4			



Process	Alloc	need ~~avaible~~	available
			2 1 0
P_0	0 3 0	7 2 3 —	Not found enough resources
P_1	3 0 2	0 2 0 —	"
P_2	3 0 2	6 0 0 —	"
P_3	2 1 1	0 1 1 —	"
P_4	0 0 2	4 3 1 —	"

Unsafe (No safe sequence)

Pretend the allocated resource to P_i by modifying the state as follows:

avaiable = available − request

allocation = allocation + request

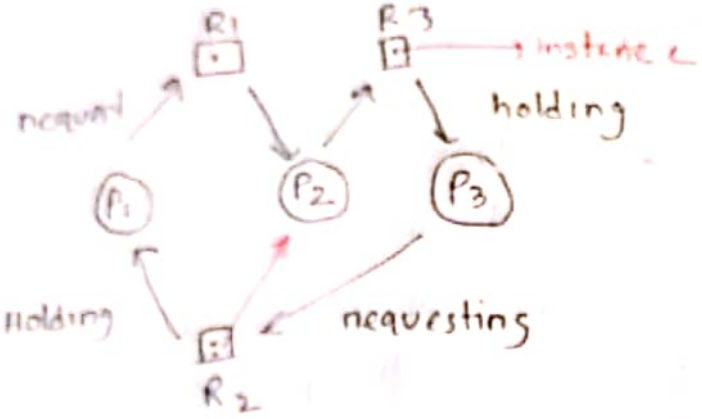
Need = need − request

∴ if safe → then resource allocated to P_i

∴ in unsafe → P_i must wait, old resource - allocation state is restored.

∴ complexity = $O(n^2m)$

Deadlock is a situation that occurs in OS when any processes enter a waiting state because another waiting process holding the demanded resource



Resource allocation graph

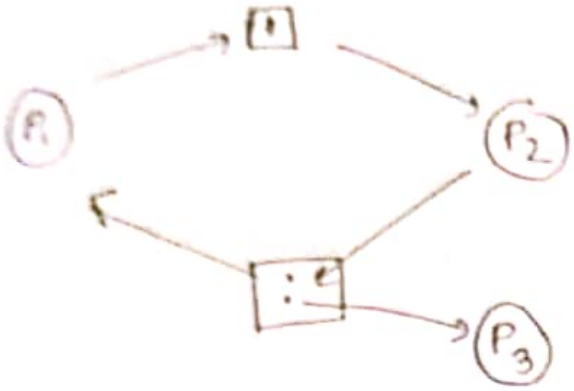
Dead lock Detection

No preemption
(can release resource only after complete job, can't release in middle)

mutual exclusion
(one process share one resource at a time)

Circular wait
(one process waiting for resource held by 2nd process which also waiting for resource held by 3rd process, this waiting continue untill last process waiting for resource held by first process)

Hold and wait
(one process holding more than one or atleast one resource waiting for another resource from another process)



cycle but no deadlock

Handling Deadlocks

- Deadlock prevention
- Deadlock avoidance
- allow to have deadlock then prevent it
- Ignore problem assume deadlock never happened

Deadlock Prevention

(1) (mutual Exclusion) make resource shareable
one resource → multiple process

(2) (Hold & wait) (i) Process will request what resource one needed for execution at first and all at resource will be allocated at the beginning of execution

(ii) Process can request for resource only if it is not holding any resource.

③ (No preemption) if Process request for any resource first it have to release all resource it is holding allocated

④ (circular wait) Order resource and process have to request for resource in an increasing number

---

A state of system is called safe system if the system can allocate all its resources requested by all the process in system without entering deadlock.

request ≤ available resource → safe

request > available → unsafe → possibility of deadlock

deadlock avoidance

single instance

Resource allocation graph

→ temporarily accept the request

→ check any loop on safe/unsafe

→ if unsafe decline the request

multiple instance

Banker's Algorithm

allocation = allocation + request

available = available - request

need = need - request

: after that if we found a safe sequence

→ accept the request

: if unsafe → process have to wait and old resource allocation state is recovered.