Deadlock Module 3

Deadlock is a important concept of operating system.
First will understand what is this deadlock.
In real life lak ur house, throw the key inside the house. Now unless u get key cant open the door, unless u open the door we cannot get the key.
another example when a trains approach each other while crossing, both will come to a full stop & neither shall again until other has gone.

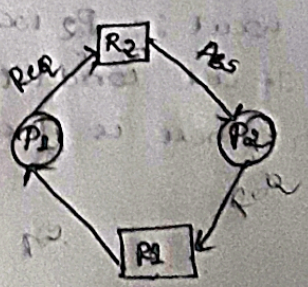
Try to understand, if e want a resource which is held by some other person & that person is using it, so after completing he will return the resource. but he is waiting for the resource which is with me then both of us will end up waiting.

A	B
drink soup	drink soup
plate	spoon

deadlock

We know that our operating system is a multiprogramming os. where multiple process will compete for limited no of resources. when a process wants to execute or run it request for resources & it does not get the resource it enters into a waiting state.
The process which entered into waiting state may not be able to come back to running state (change its state) because the resource which it is waiting for is held by another process which is in waiting state. This situation is called as deadlock.

$P_0 \longrightarrow P_1$ (running)
$P_0 \longrightarrow P_3$ (waiting)
may be waiting for resource which is with P_0.
waiting is not going to end.



$P_1 \rightarrow R_2$
$P_2 \rightarrow R_1$

$P_1 \rightarrow R_2 \rightarrow P_2 \rightarrow R_1 \rightarrow P_1$

Every process follows a system model consist of three process.

1) Request :- Every process request a resource.
2) use :- once get it use
3) Release :- after using release it.

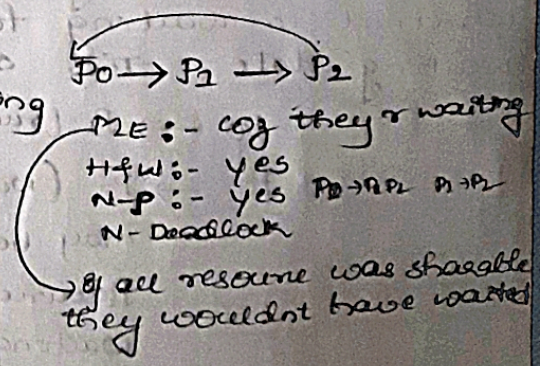
* Necessary conditions coz of which (there exist) deadlock occure if all these conditions occurs at once.

1) Mutual Exclusion :- Only one process can use the resource at a time. If another process request for the same resource then it has to wait.
If the wait is definate no problem. If wait is indefinate then more chances are der. (If the resource is in non sharable mode ex printer or cpu. and more than one process wants to use it)

2) Hold & Wait : Simple example where A ~~want~~ got bowl & B got spoon. watever is available take & then request for another resource. We'll not say tell i get both ill wait.
Same with system also, what ever is available hold it & then wait for other resources.
5 resource → 3 hold & wait 2.
If : wait for all then there may be starvation.

3) No Preemption :- One process cannot snatch the resource of another process. when process completes it releses.

4) Cirular wait :- P2 waiting for P0 but P0 waiting for P1. If all conditions are true at same time then there is deadlock. If there is cycle.



Resource Allocation graph

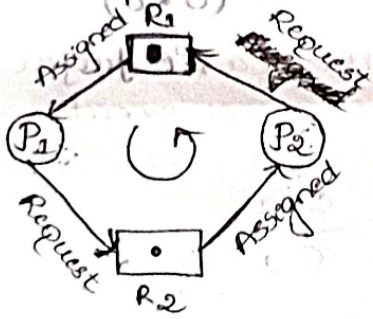
We can say there is deadlock or not by using RAG.
This graph consist of edges & vertices (Set of edges E & vertices V)
Vertices are partitioned into two different types of nodes:
a) $P = \{P_1, P_2, \ldots P_n\}$ Set consisting of active process.
b) $R = \{R_1, R_2, \ldots R_n\}$ set consisting of active Resources types in the system.

A directed edge from Process to Resource i.e $P \rightarrow R$ means process is requesting for resources. This is known as request edge.
A directed edge from Resource to process i.e $R \rightarrow P$ means Resource has be allocated to process. This is known as assignment edge.

- ○ → Represents Process
- □ → Represents Resource
- ⊡ → One instance of Resource
- ⁝ → two instance of Resource

By using RAG we can check is der a deadlock for single instance

1)



$P_1 \rightarrow R_2 \rightarrow P_2 \rightarrow R_1 \rightarrow P_1$

Circular wait.
From where we started we reached the same point.
There exist a cycle. Hence there is a deadlock. (For single instance)

There exist circular wait, & hold & wait (P_1 has R_2 but waiting for R_1) No preemption & Mutual exclusion.

2nd Method

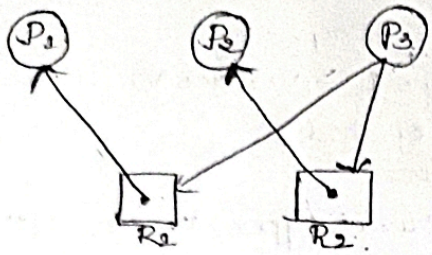
	Allocate		Request	
	R_1	R_2	R_1	R_2
P_1	1	0	0	1
P_2	0	1	1	0

Availability : R_1 R_2
(how many resources are currently available) (0, 0)

Seeing availability we need to check can we fullfill the process need.
Available is (0,0) & P_1 wants (0,1) & P_2 wants (1, 0). So we cant fulfill the need. Hence deadlock

Allocate: the resource assigned to the process.

2)



	Allocate		Request		
	R1	R2	R1	R2	
P1	1	0	0	0	✓
P2	0	1	0	0	✓
P3	0	0	1	1	

Available = (R1 R2)
0 0

P3 can't fulfill P1 yes so it executes & termenates. Now available will be (1,0)

Can't fulfill P2? yes so it executes & termenates. Now available will be (1,1)

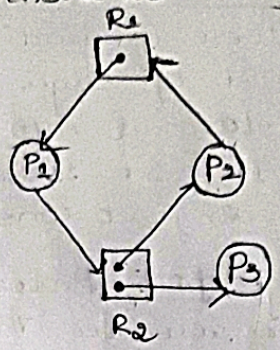
Can't fulfill P3? yes.

No deadlock.

→ There is no cycle or circular wait. Acyclic graph.

If there is circular wait in RAG then always there exist a deadlock. for a single instance Resource.

3) Multiinstance RAG



	Allocate		Request		
	R1	R2	R1	R2	
P1	1	0	0	1	✓
P2	0	1	1	0	
P3	0	1	0	0	

total: R1 = 1, R2 = 2

Available : R1, R2
(0,0)

With this can't fulfill P1? No P2? No P3 yes. So it executes & releases.

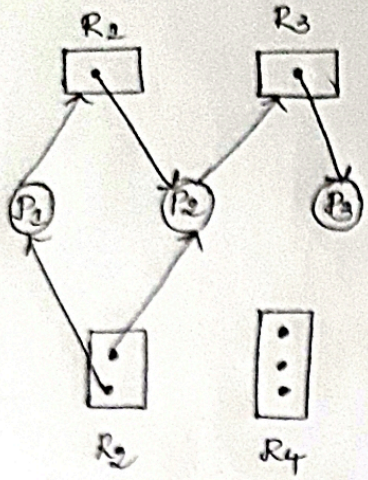
i,e 0 , 0
\+ 0 1
0 1

Now P1 can execute 0, 1
\+ 1 0
1 1

Now P2 can execute yes.

⟨P3, P1, P2⟩ No deadlock.

4)



	Allocate			Request		
	R1	R2	R3	R1	R2	R3
P1	0	1	0	1	0	0
P2	1	1	0	0	0	1
P3	0	0	1	0	0	0

Available : (0, 0, 0)

Can P3 execute ? yes.

```
 0 0 0
 0 0 1
-------
 0 0 1
```

Can P2 execute? yes

```
 0 0 1
 1 1 0
-------
 1 1 1
```

Can P1 execute? yes.

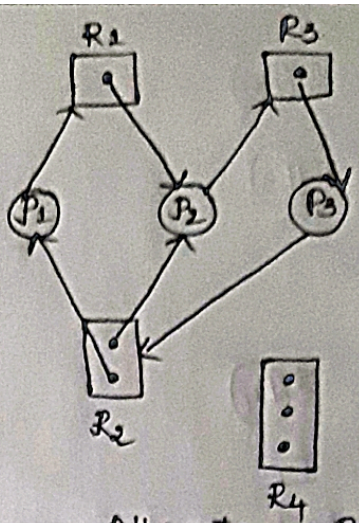
```
 1 1 1
 0 1 0
-------
 1 2 1
```

No deadlock.

P1 → R1 → P2 → R3 → P3

No cycle.

5)



	Allocate			Request		
	R1	R2	R3	R1	R2	R3
P1	0	1	0	1	0	0
P2	1	1	0	0	0	1
P3	0	0	1	0	1	0

Available :- (0, 0, 0)

with this none of the process need can be satisfied. Deadlock.

Manually if u can see there exist a cycle or circular wait.

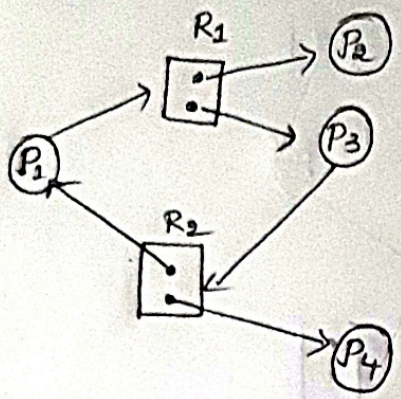
P1 → R1 → P2 → R3 → P3 → R2 → P1

P2 → R3 → P3 → R2 → P2

Note:- For a multi instance graph it is not mandatory to have deadlock if there exist a cycle.

It is true for single instance graph

6)



$P_1 \rightarrow R_1 \rightarrow P_3 \rightarrow R_2 \rightarrow P_1$

There is a cycle but no deadlock.

	Allocate		Request	
	R_1	R_2	R_1	R_2
P_1	0	1	1	0
P_2	1	0	0	0
P_3	1	0	0	1
P_4	0	1	0	0

Available : (0, 0)

Can P_2 execute? yes.

0, 0
1, 0
1, 0

Can P_4 execute? yes.

1, 0
+ 0, 1
1 1

Can P_1 execute? yes

1, 1
0, 1
1 2

Can P_3 execute? yes

1, 2
1, 0
2, 2