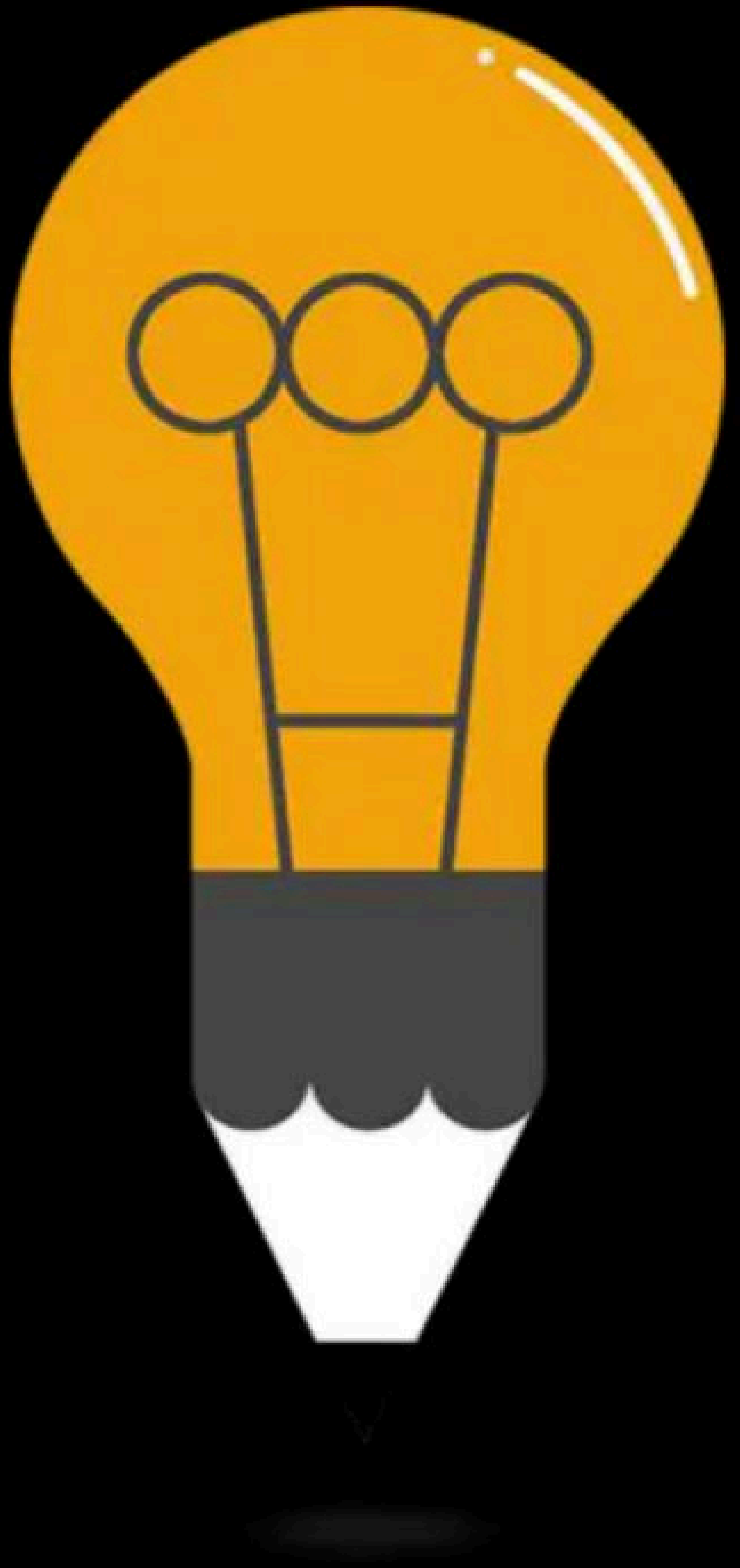
Deadlock Avoidance

Comprehensive Course on Operating System for GATE - 2024/25

Vishvadeep Gothi • Lesson 20 • May 11, 2023



Operating System
Deadlock

By: Vishvadeep Gothi

1 • Asked by Kumar

Sir why ans race condition

When the result of a computation depends on the speed of the processes involved there is said to be

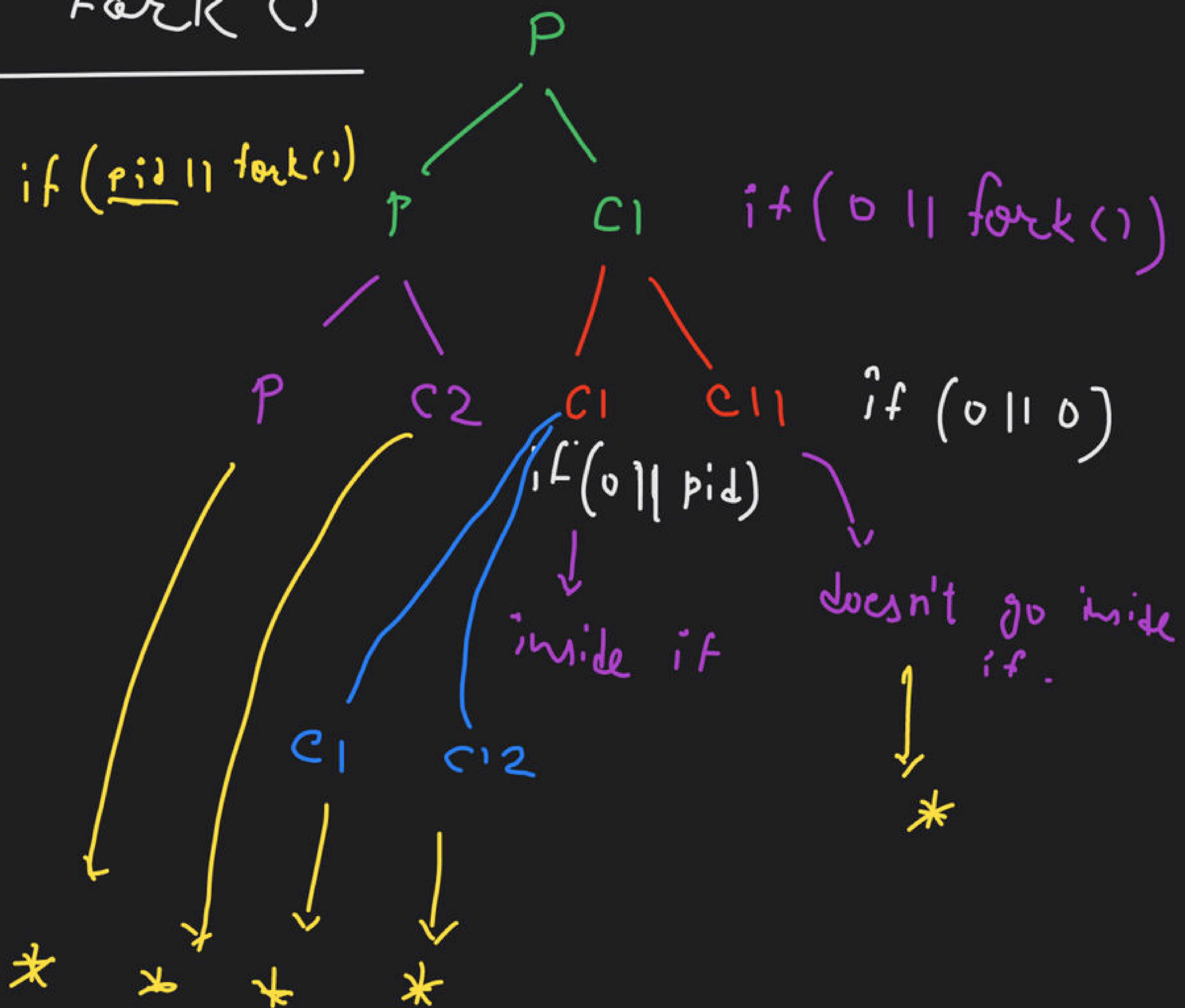
(a) cycle stealing (b) race condition

(c) a time lock (d) a deadlock

DPP fork()

Parent

```
void main()
{
    if ( fork() || fork() )
    {
        fork();
    }
    printf("*");
}
```



5 times * printed.

Operations on Resources → H/w , s/w

3 operations on resources:

1. Request
2. Use ← allocated
3. Release → done by process when process is done with use of resource.

if any process request for a resource to OS.

OS can allocate the resource to process if it is available.

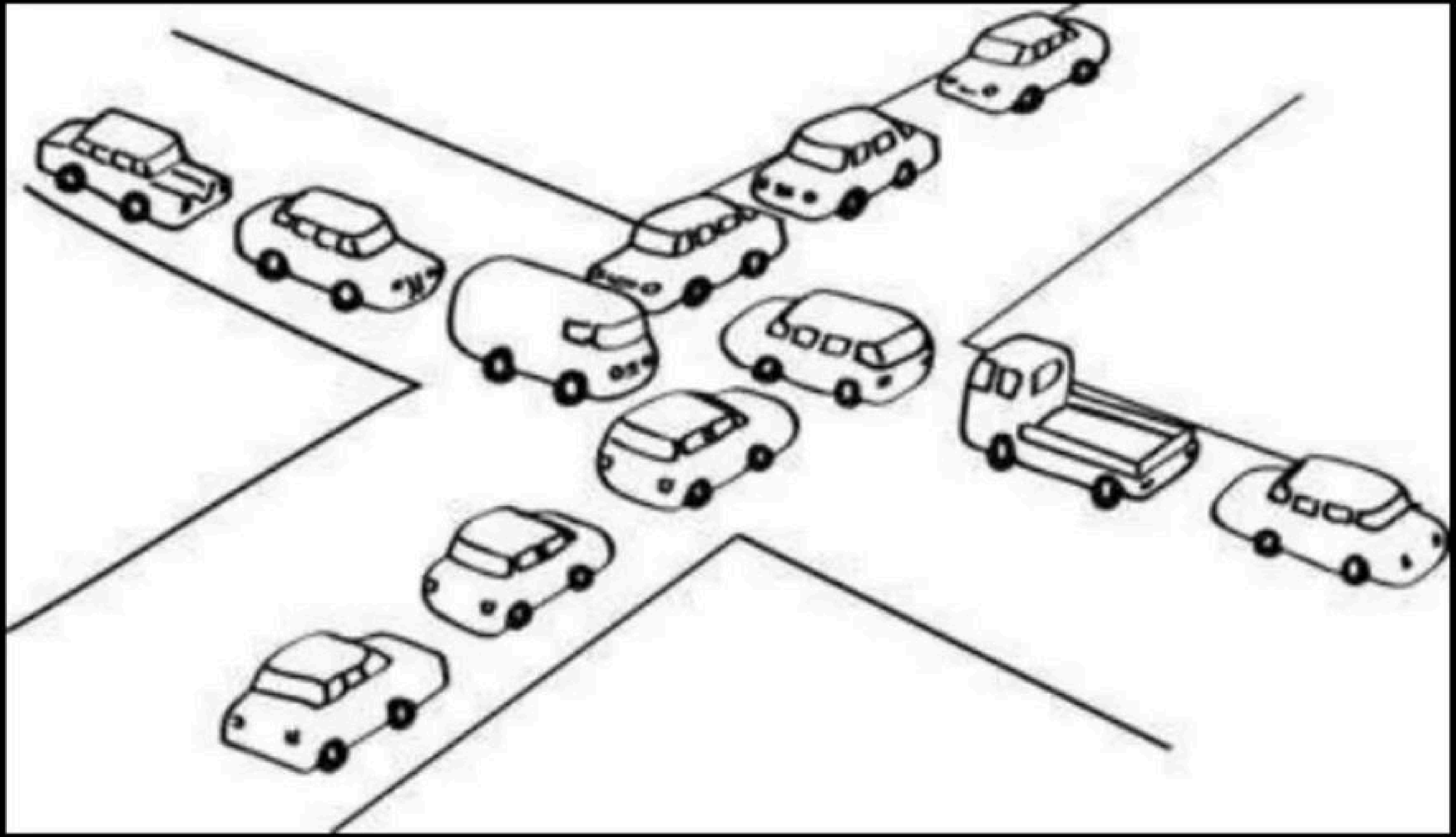
Deadlock

If two or more processes are waiting for such an event which is never going to occur

	Holds	wait
P1	keyboard	Hard disk
P2	Harddisk	Printer
P3	Printer	keyboard

Deadlock for P1, P2, P3

Deadlock



Starvation

Indefinite wait

There is a chance that the wait will be over

Deadlock

Permanent wait

No chance that the wait will be over.

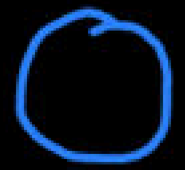
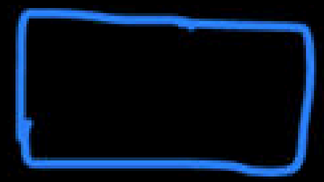
Necessary Conditions for Deadlock

Deadlock can occur only when all following conditions are satisfied:

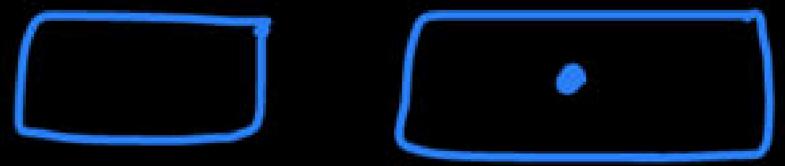
1. Mutual Exclusion → a resource can be used by 1 process at a time.
2. Hold & Wait → all deadlocked processes should hold atleast one resource & should wait for atleast one resource.
3. No-preemption → allocated resources must not be preempted from processes.
4. Circular Wait → all deadlocked processes must wait for each-other in circular manner

Resource Allocation Graph

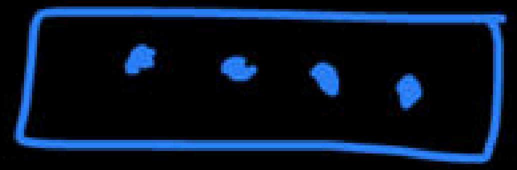
Nodes

1. Process 
2. Resource 

single instance of resource



multiple instances of a resource

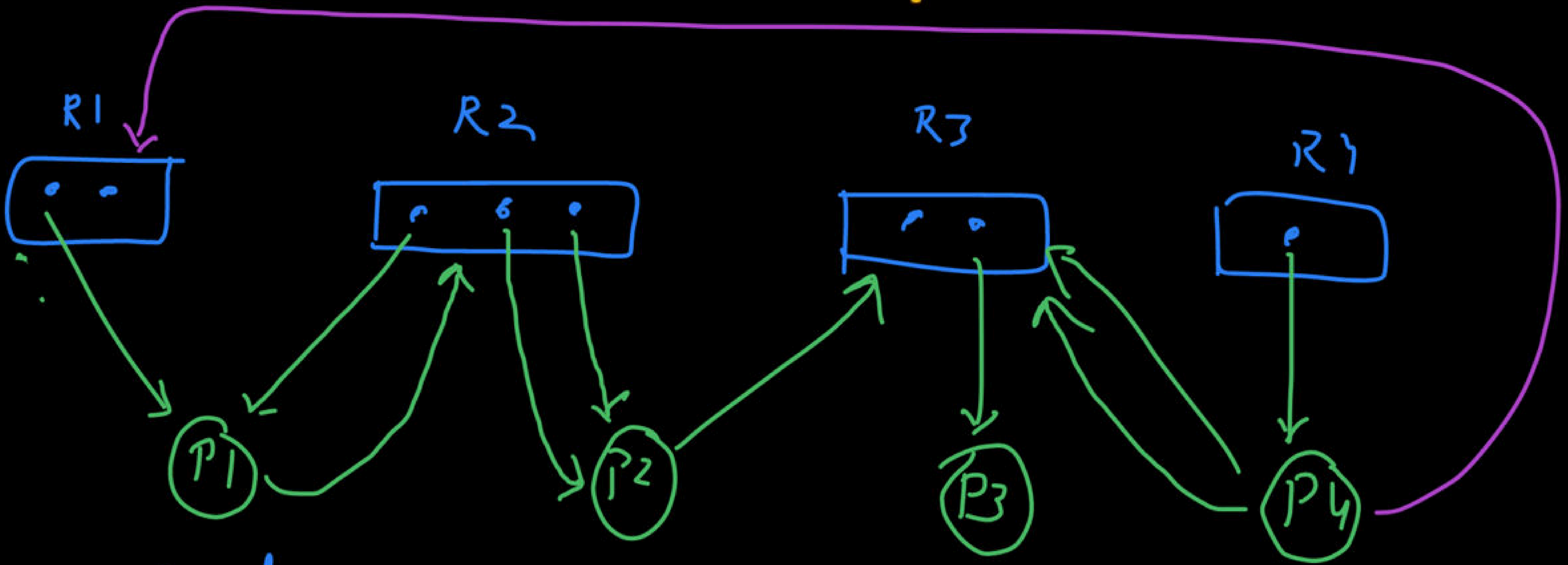


4 instances

Edges

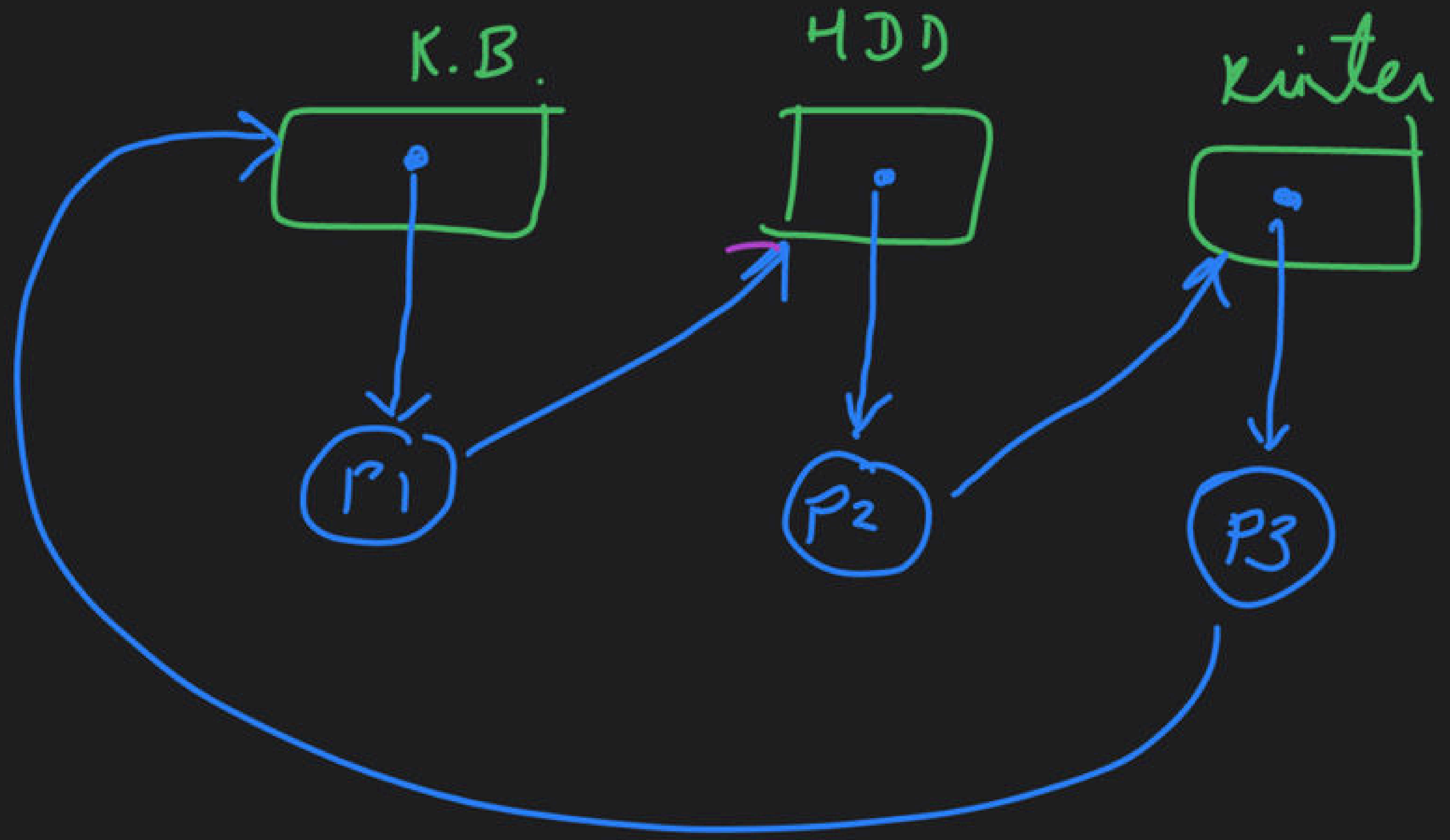
1. Allocat^n :- from resource instance to process
2. Request :- from process to resource

Resource Allocation Graph



Process	Allocation				Request			
	R1	R2	R3	R4	R1	R2	R3	R4
P1	1	1	0	0	0	1	0	0
P2	0	2	0	0	0	0	1	0
P3	0	0	1	0	0	0	0	0
P4	0	0	0	1	1	0	2	0

	Hold	wait
P1	K.B.	HDD
P2	HDD	Printer
P3	Printer	K.B.



Recovery From Deadlock

1. Make Sure that deadlock never occur
   - Prevent the system from deadlock or avoid deadlock
2. Allow deadlock, detect and recover
3. Pretend that there is no any deadlock

Deadlock Prevention :-

Does not allow system to satisfy one of the 4 necessary conditions for deadlock.

→ **Mutual Exclusion :-**

① Make all processes independent ⇒ not possible practically

② Increase no. of resources, so that each process can have their own resource

→ Hold and wait :-

A process should either wait or hold but should not do both together.

A process must acquire all resources together if all available, or else must wait for all.

⇒ Decreased resource utilization.

⇒ Possibility of starvation

No preemption :-

OS tries to preempt resources from processes.

→ Process may be in unstable state after resource preemption.

Circular wait :-

Give numbers to each resource $R_1, R_2, R_3, \ldots, R_n$

A process can request a resource R_i, while holding a resource R_j, only when $i > j$

Ex: ① A process holds R4, request R6 ⇒ allowed

② A process holds R3, Request R1 ⇒ Release R3, try to acquire R1 first then R3.

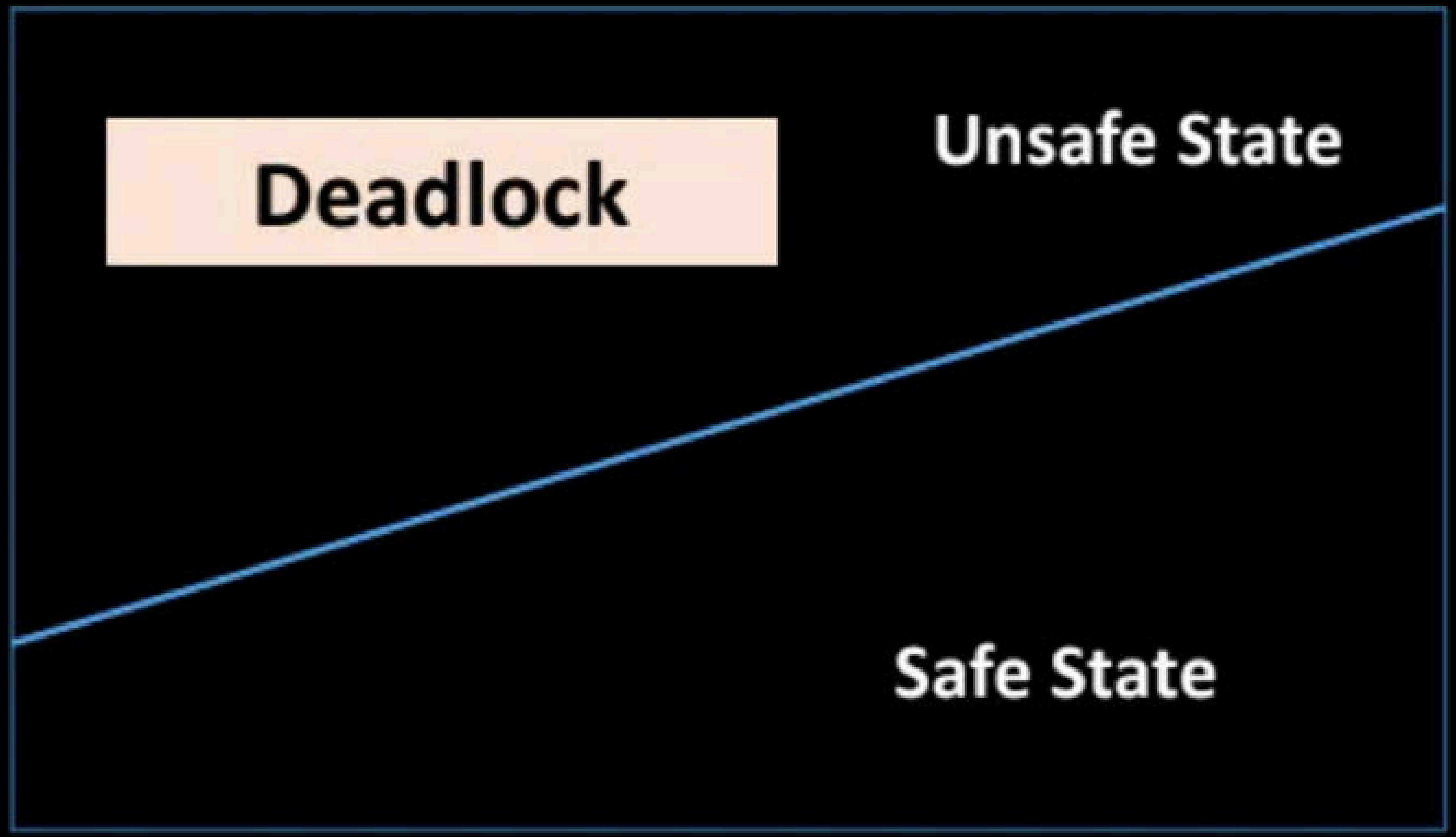
③ A process holds R4 and R7, requests for R5 ⇒ Release R7, try to acquire R5, first then R7.

Deadlock Avoidance

In deadlock avoidance, the OS tries to keep system in safe state

Deadlock Avoidance

In deadlock avoidance, the OS tries to keep system in safe state



if system is in unsafe state, then possibility of deadlock.

Deadlock Avoidance

In deadlock avoidance, the request for any resource will be granted if the resulting state of the system doesn't cause deadlock in the system.

→ In deadlock avoidance, each process must declare to OS that for which resource how many instances at max the process will require.

Banker's Algorithm

The banker's algorithm is a resource allocation and deadlock avoidance algorithm that tests for safety

① safety algorithm ⇒ checks if system is in safe state or not

② Resource Request algo ⇒ when a request comes from process, then OS checks if the request can be granted or not.

granted means system will be in safe state after allocation.

Banker's Algorithm

Process	Allocation	Max	Available
P1	1	3	1
P2	5	8	
P3	3	4	
P4	2	7	

Banker's Algorithm

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

Banker's Algorithm

1. Allocation:
2. Max:
3. Need:
4. Available:

Banker's Algorithm

1. Let Work and Finish be vectors of length 'm' and 'n' respectively.
   Initialize: Work = Available
   Finish[i] = false; for i=1, 2, 3, 4….n
2. Find an i such that both
   (a) Finish[i] = false
   (b) $Need_i$ <= Work
   if no such i exists goto step (4)
3. Work = Work + Allocation[i]
   Finish[i] = true
   goto step (2)
4. if Finish [i] = true for all i
   then the system is in a safe state

Question

Process	Allocation				Max				Available			
	A	B	C	D	A	B	C	D	A	B	C	D
P1	0	0	1	2	0	0	1	2	1	5	2	0
P2	1	0	0	0	1	7	5	0				
P3	1	3	5	4	2	3	5	6				
P4	0	6	3	2	0	6	5	4				
P5	0	0	1	4	0	6	5	6				

Banker's Algorithm

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

Question

What will happen if process P1 requests one additional instance of resource type A and two instances of resource type C?

Banker's Algorithm

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

Question

What will happen if process P0 requests one additional instance of resource type A and two instances of resource type C?

Question

What will happen if process P3 requests one additional instance of resource type B?

Resource Request Algorithm

1. *If $Request_i <= Need_i$*
   Goto step (2) ; otherwise, raise an error condition, since the process has exceeded its maximum claim.
2. *If $Request_i <= Available$*
   Goto step (3); otherwise, P_i must wait, since the resources are not available.
3. *Have the system pretend to have allocated the requested resources to process P_i by modifying the state as follows:*
   Available = Available – Requesti
   $Allocation_i = Allocation_i + Request_i$
   $Need_i = Need_i – Request_i$

Happy Learning.!