Operating System

BIM IV Semester

Credits: 3

Lecture Hours:48

Er. Santosh Bhandari,
(Master Computer Science)

Unit-3

Deadlock

Deadlock

A process in operating system uses resources in the following way.

1. Requests a resource
2. Use the resource
3. Releases the resource

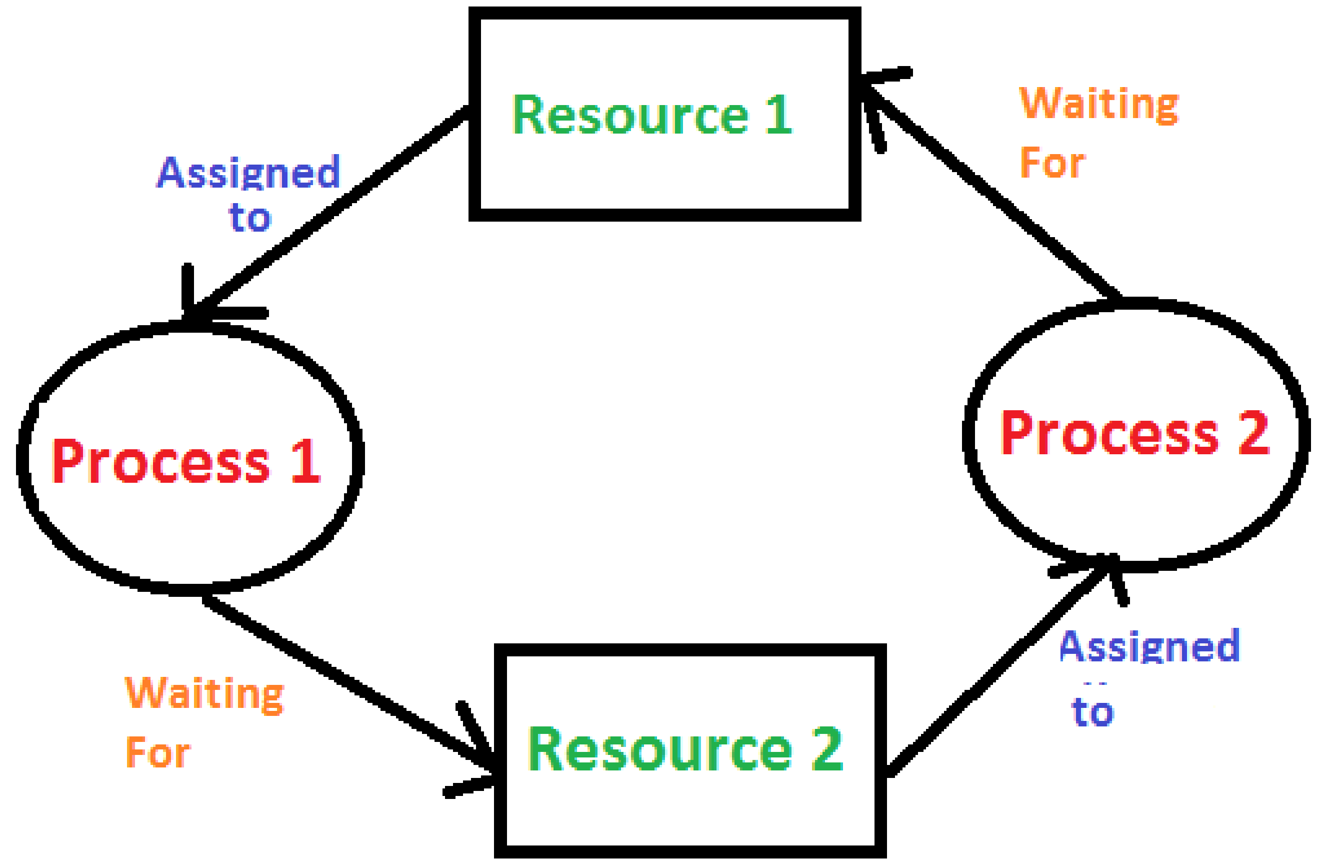
What is deadlock?

A Deadlock is a situation where each of the computer process waits for a resource which is being assigned to some another process. In this situation, none of the process gets executed since the resource it needs, is held by some other process which is also waiting for some other resource to be released.

Deadlock

A deadlock happens in operating system when two or more processes need some resource to complete their execution that is held by the other process.

In the diagram, the process 1 has resource 1 and needs to acquire resource 2. Similarly, process 2 has resource 2 and needs to acquire resource 1. Process 1 and process 2 are in a deadlock as each of them needs the other's resources to complete their execution but neither of them is willing to relinquish their resources.



Deadlock

Example

- ✦ System has 2 tape drives.
- ✦ P1 and P2 each hold one tape drive and each needs another one.

Example

- ✦ semaphores A and B, initialized to 1

P0

wait (A); wait(B)

P1

wait (B); wait(A)

Necessary Conditions for Deadlock/Characteristics

Coffman Conditions

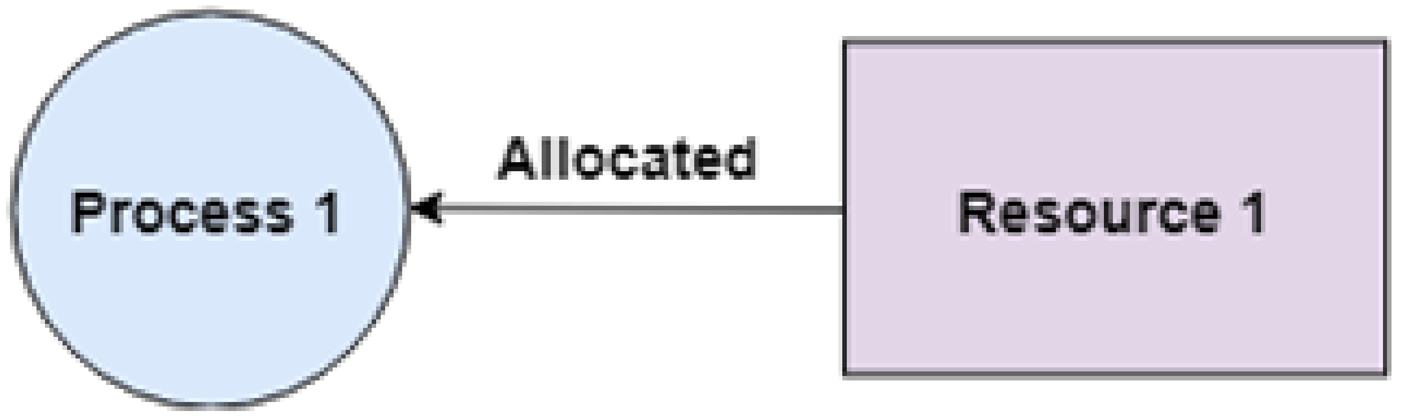
A deadlock occurs in four different scenarios. Those four conditions are also known as **the Coffman conditions**. **Coffman (1971)**

In other words, a deadlock occurs if four Coffman conditions are true.

1. **Mutual exclusion**
2. **Hold and wait**
3. **No preemption**
4. **Circular wait**

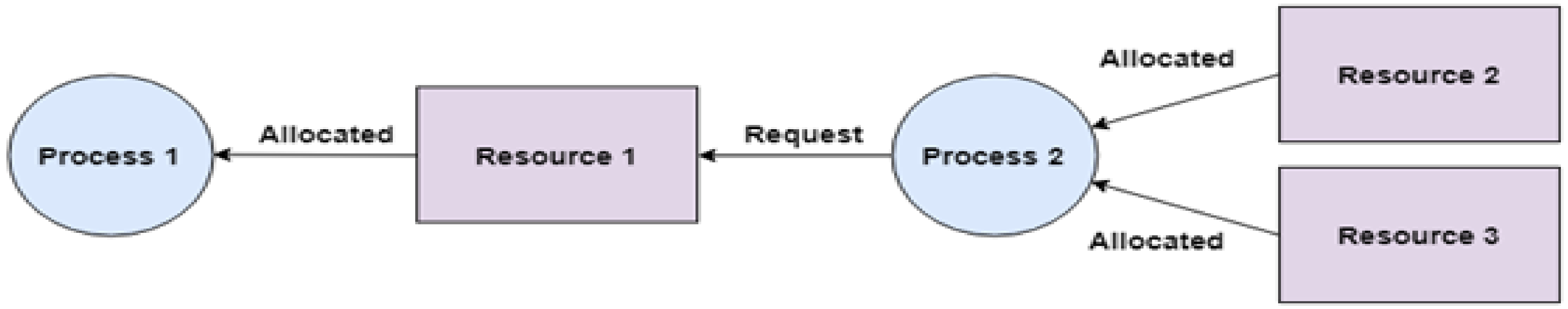
Necessary Conditions for Deadlock

Mutual Exclusion: Two or more resources are non-shareable (Only one process can use at a time)
There should be a resource that can only be held by one process at a time. In the diagram below, there is a single instance of Resource 1 and it is held by Process 1 only.



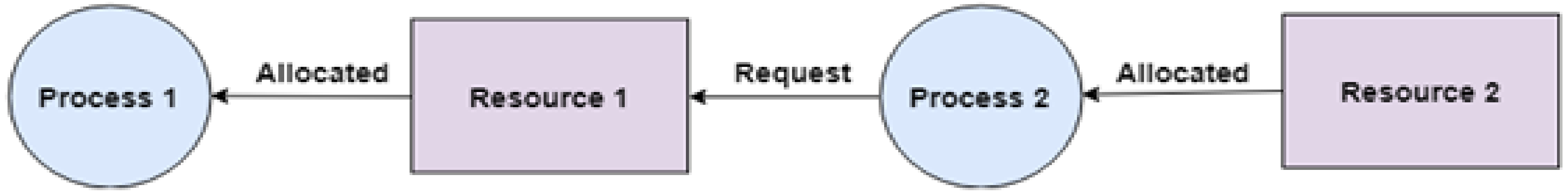
Necessary Conditions for Deadlock

2. Hold and Wait: A process is holding at least one resource and waiting for resources.
A process can hold multiple resources and still request more resources from other processes which are holding them. In the diagram given below, Process 2 holds Resource 2 and Resource 3 and is requesting the Resource 1 which is held by Process 1.



Necessary Conditions for Deadlock

3. No Preemption: A resource cannot be taken from a process unless the process releases the resource.
A resource cannot be preempted from a process by force. A process can only release a resource voluntarily. In the diagram below, Process 2 cannot preempt Resource 1 from Process 1. It will only be released when Process 1 relinquishes it voluntarily after its execution is complete.

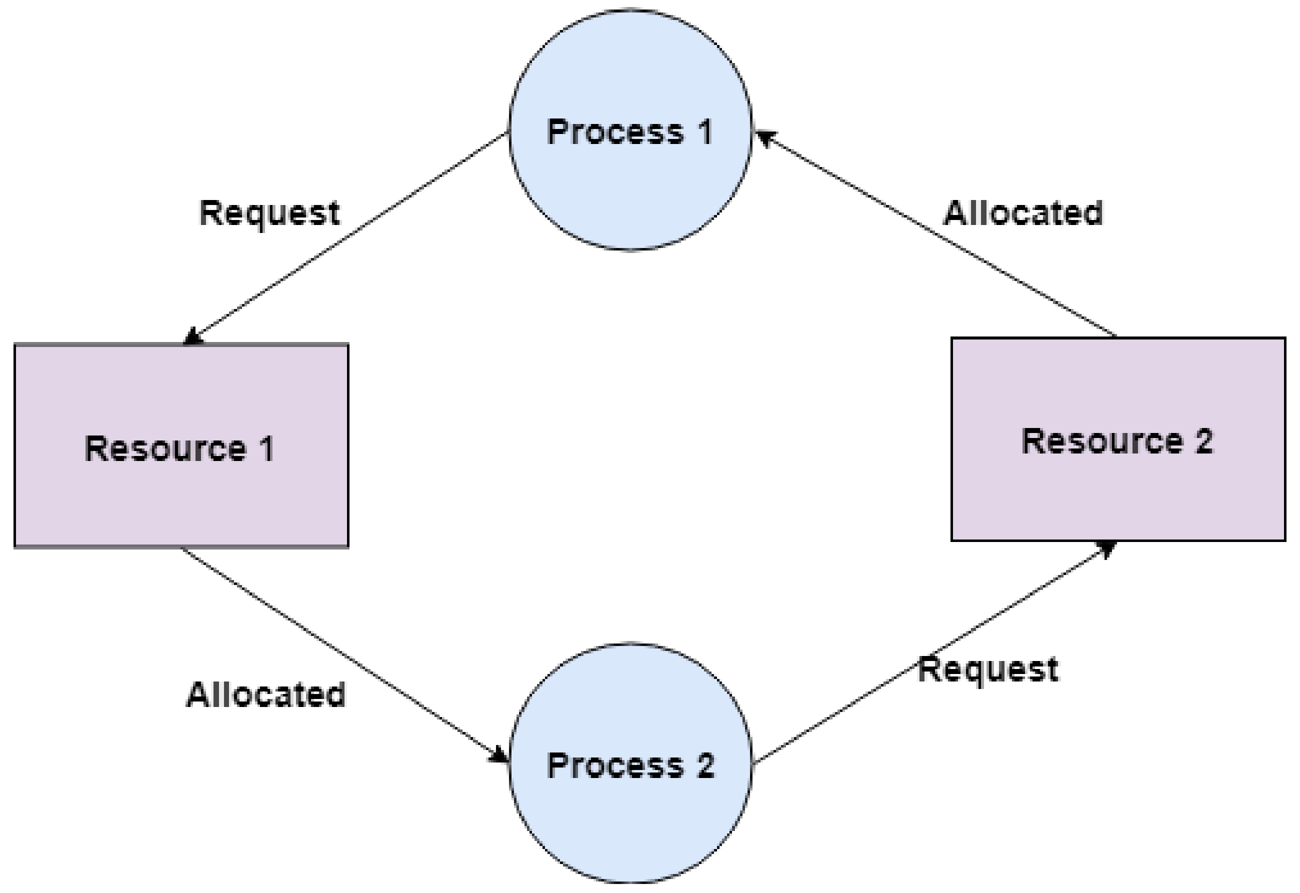


Necessary Conditions for Deadlock

4. Circular Wait: A set of processes waiting for each other in circular form. A process is waiting for the resource held by the second process, which is waiting for the resource held by the third process and so on, till the last process is waiting for a resource held by the first process. This forms a circular chain.

Necessary Conditions for Deadlock

For example: Process 1 is allocated Resource2 and it is requesting Resource 1. Similarly, Process 2 is allocated Resource 1 and it is requesting Resource 2. This forms a circular wait loop.



Deadlock vs. Starvation

Deadlock	Starvation
Deadlock is a situation where no process got blocked and no process proceeds	Starvation is a situation where the low priority process got blocked and the high priority processes proceed.
Deadlock is an infinite waiting.	Starvation is a long waiting but not infinite.
Every Deadlock is always a starvation.	Every starvation need not be deadlock.
The requested resource is blocked by the other process.	The requested resource is continuously be used by the higher priority processes.
Deadlock happens when Mutual exclusion, hold and wait, No preemption and circular wait occurs simultaneously.	It occurs due to the uncontrolled priority and resource management.

System resources

A computer has many different resources. A process needing one of these resources can use any of them.

There are two types of system resources:

1. Preemptable
2. Non-Preemptable

System resources

Preemptable: A preemptable resource is one which can be allocated to a given process for a period of time, then be allocated to another process and then be reallocated to the first process without any effects.

Examples

Memory, Buffers, CPU, Array Processor

System resources

Non Preemptable: A non preemptable resource cannot be taken from one process and given to another without any side effects.
Example: Printer, certainly we would not want to take the printer away from one process and give it to other in the middle of a print job.

Resource Allocation Graph (RAG)

A resource allocation graph shows which resource is held by which process and which process is waiting for a resource of a specific kind.

The resource allocation graph explained following:
what is the state of the system in terms of processes and resources?
How many resources are available?
How many are allocated?
What is the request of each process?

Resource Allocation Graph (RAG)

Types of Vertices in RAG: two types

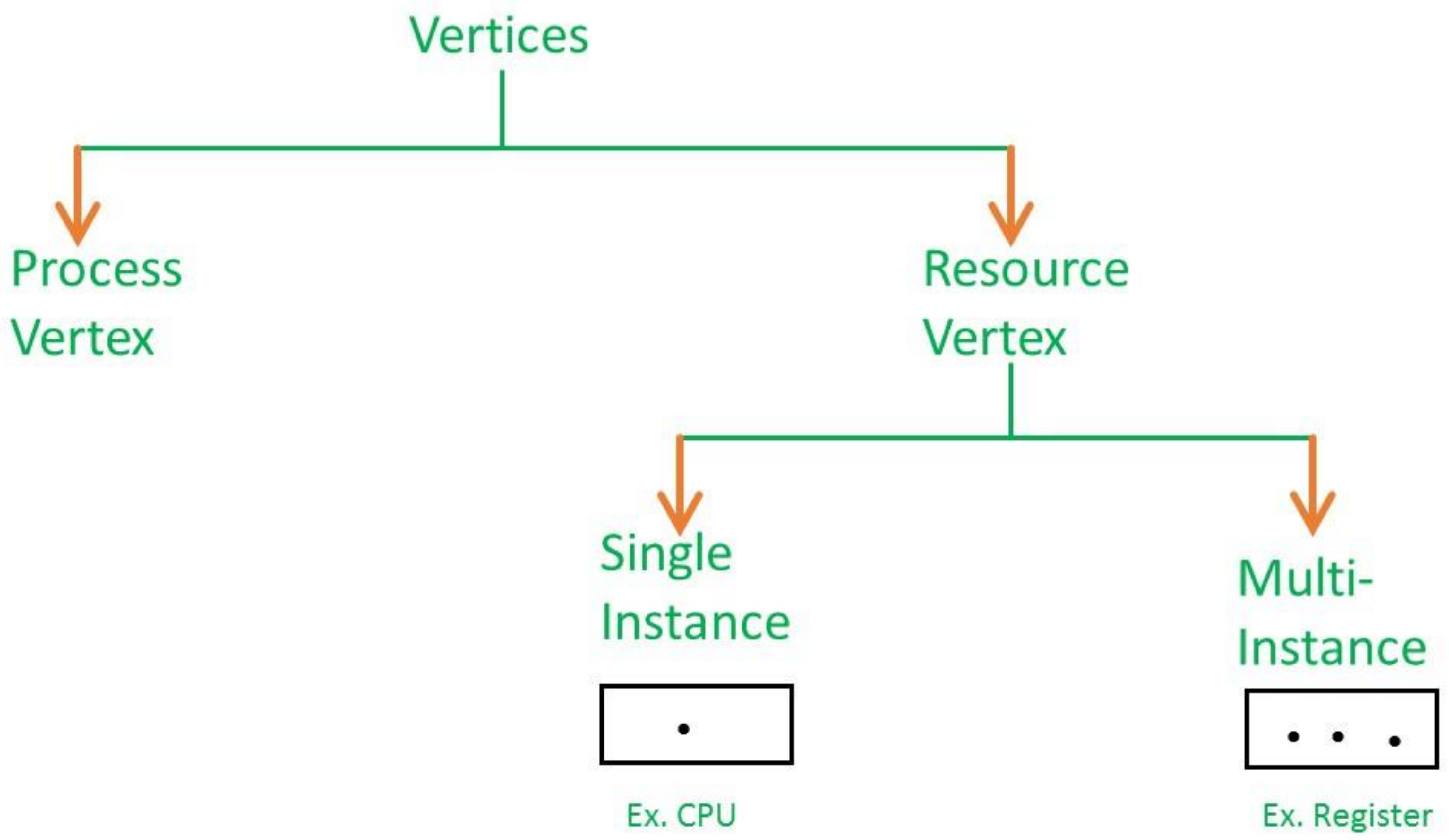
1. Process Vertex: Every process will be represented as a process vertex. Generally, the process will be represented with a circle.

2. Resource Vertex: Every resource will be represented as a resource vertex. It is also two types:

Single instance type resource: It represents a box, inside the box, there will be one dot. So the number of dots indicates how many instances are present of each resource type.

Multi-resource instance type resource: It also represents as a dot, inside the box, there will be many dots present.

Resource Allocation Graph (RAG)

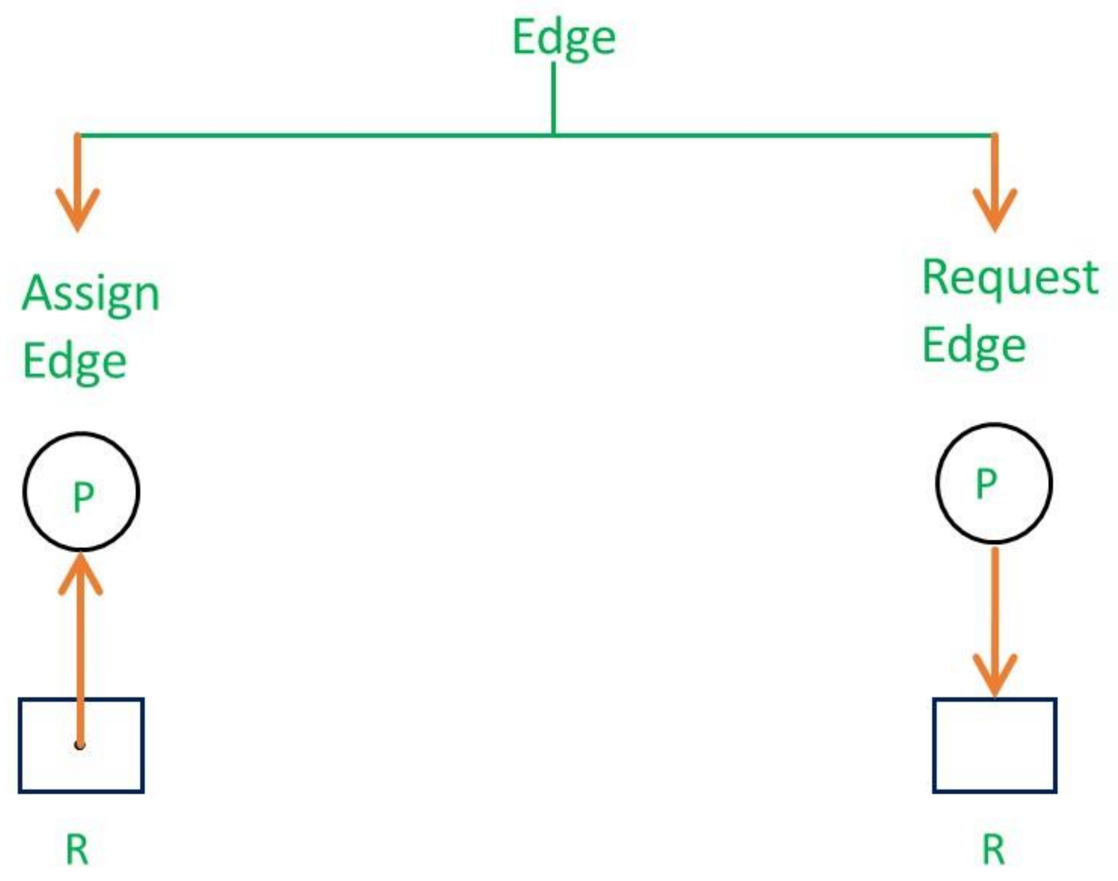


Resource Allocation Graph (RAG)

Types of Edges in RAG?: two

1. Assign Edge: If you already assign a resource to a process then it is called Assign Edge.

2. Request Edge: It means in the future the process might want some resource to complete the execution, that is called request edge.



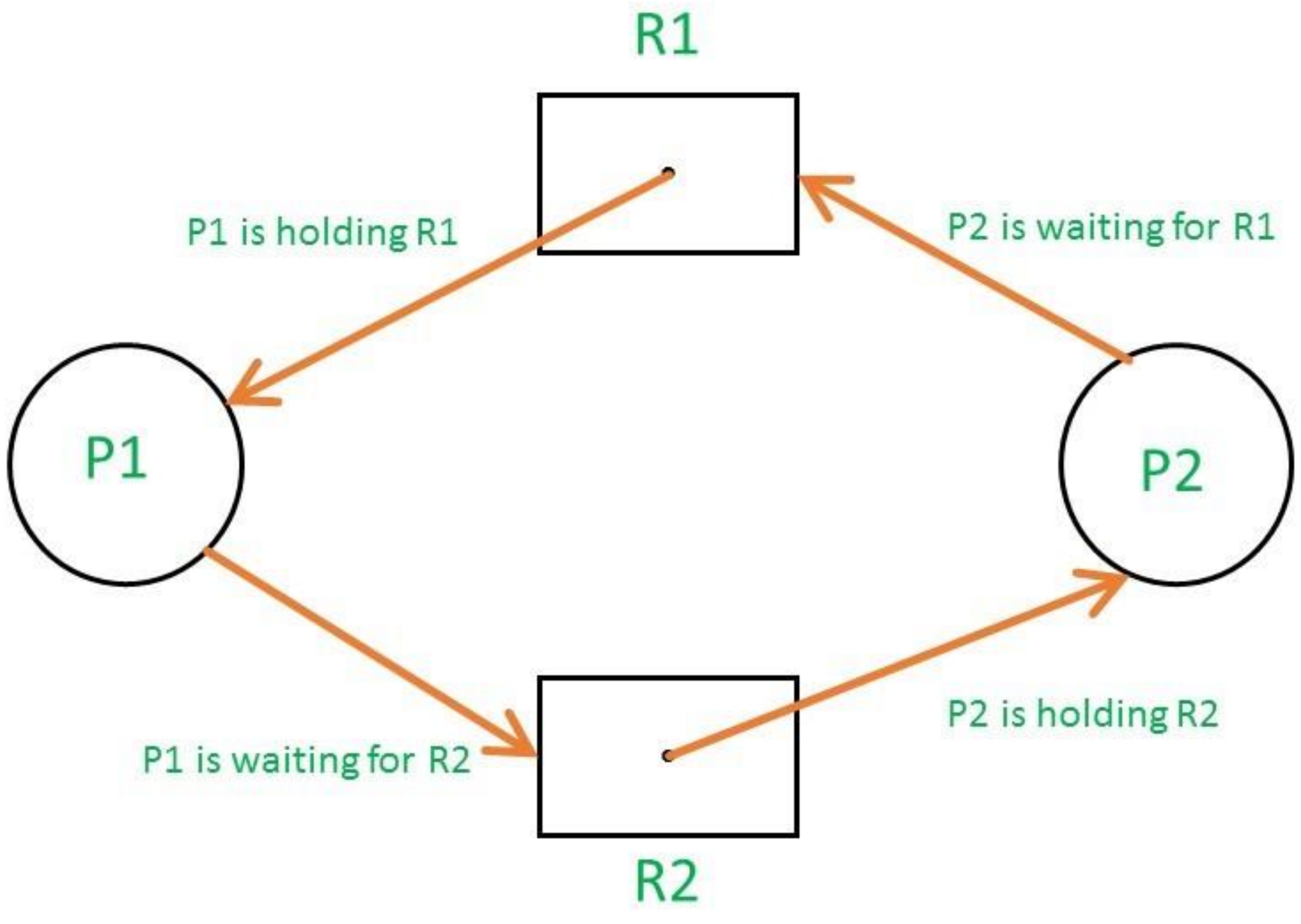
Arrow R->P: if a process is using a resource

Arrow P->R: If a process is requesting a resource

Example 1: Single instances RAG

With deadlock

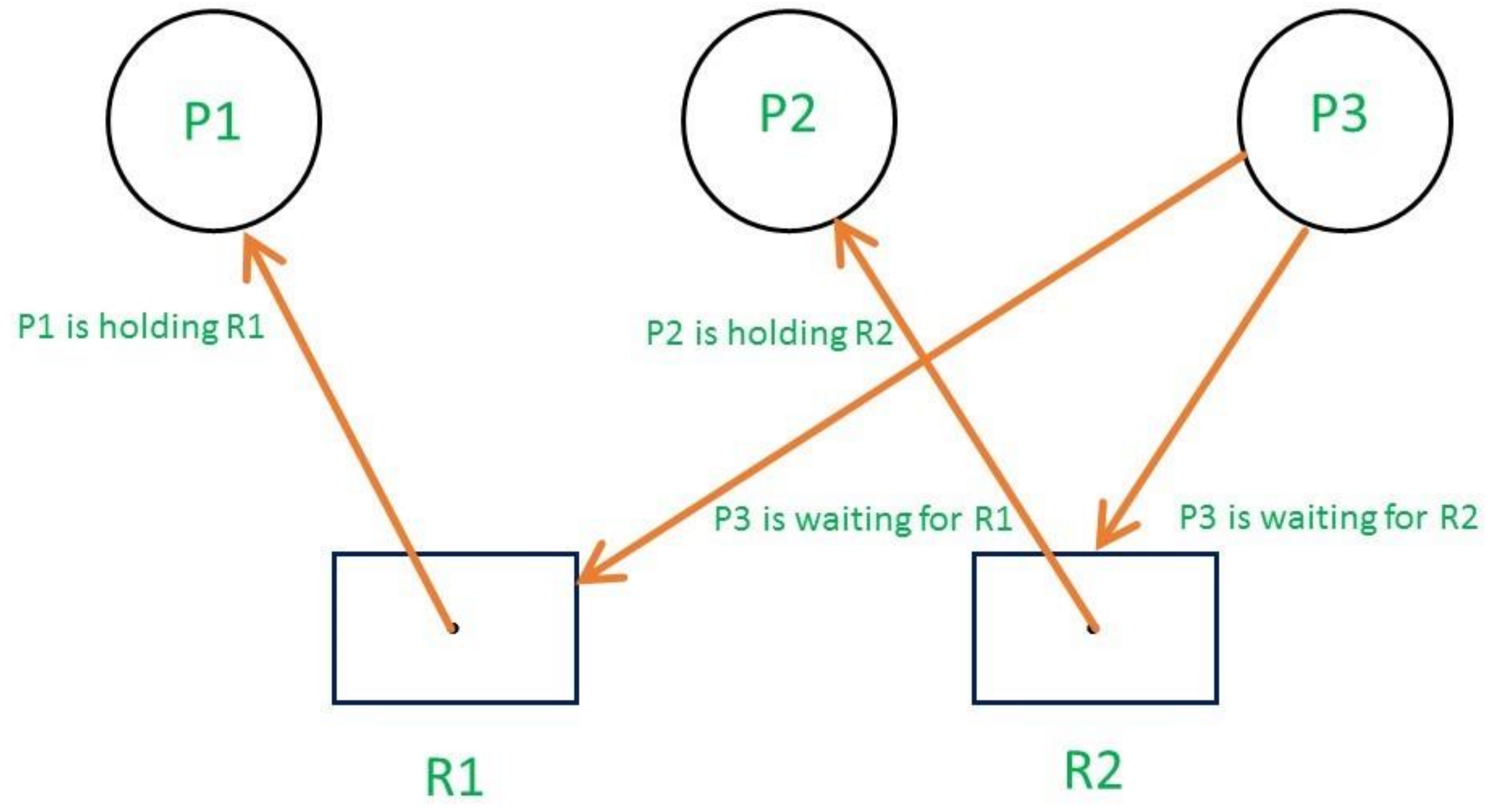
If there is a cycle in the Resource Allocation Graph and each resource in the cycle provides only one instance, then the processes will be in deadlock. For example, if process P1 holds resource R1, process P2 holds resource R2 and process P1 is waiting for R2 and process P2 is waiting for R1, then process P1 and process P2 will be in deadlock.



SINGLE INSTANCE RESOURCE TYPE WITH DEADLOCK

Example 1: Single instances RAG

Without Deadlock:
Processes P1 and P2 acquiring resources R1 and R2 while process P3 is waiting to acquire both resources. In this example, there is no deadlock because there is no circular dependency.



SINGLE INSTANCE RESOURCE TYPE WITHOUT DEADLOCK

Handling Deadlocks

Methods of handling deadlocks:

There are four approaches to dealing with deadlocks.

1. Deadlock Ignorance (Ostrich Method)

2. Deadlock Prevention

-Mutual exclusion, hold and wait, No-preemption, Circular wait

3. Deadlock avoidance (Banker's Algorithm)

4. Deadlock detection & recovery

Deadlock Ignorance (Ostrich Method)

Stick your head in the sand and pretend there is no problem at all, this method of solving any problem is called Ostrich Algorithm. This Ostrich algorithm is the most widely used technique in order to ignore the deadlock and also it used for all the single end-users uses. If there is deadlock in the system, then the OS will reboot the system in order to function well.

Including UNIX and WINDOWS, all the operating system ignore the deadlock.

Deadlock Ignorance (Ostrich Method)

Advantages:

- Simplicity: Ignoring the possibility of deadlock can make the design and implementation of the operating system simpler and less complex.
- Performance: Avoiding deadlock detection and recovery mechanisms can improve the performance of the system, as these mechanisms can consume significant system resources.

Deadlock Ignorance (Ostrich Method)

Disadvantages:

- Unpredictability: Ignoring deadlock can lead to unpredictable behavior, making the system less reliable and stable.
- System crashes: If a deadlock does occur and the system is not prepared to handle it, it can cause the entire system to crash, resulting in data loss and other problems.
- Reduced availability: Deadlocks can cause processes to become blocked, which can reduce the availability of the system and impact user experience.

Deadlock Prevention

-Try to false necessary conditions of deadlock (Mutual exclusion, hold and wait, No-preemption, Circular wait).

-To prevent deadlock we try to false any of the four or all four conditions.

We can prevent a Deadlock by eliminating any of the four conditions.

1. Removal of mutual exclusion
2. Eliminate Hold and wait
3. Eliminate No Preemption
4. Eliminate Circular Wait

Deadlock Prevention

1. Removal of mutual exclusion
A resource can never be used by more than one process simultaneously which is fair enough but that is the main reason behind the deadlock. If a resource could have been used by more than one process at the same time then the process would have never been waiting for any resource.
2. Eliminate Hold and Wait: Allocate all required resources to the process before the start of its execution, this way hold and wait condition is eliminated but it will lead to low device utilization.

Deadlock Prevention

3. Eliminate No Preemption:

In certain situations, deadlocks can be prevented by resource preemption. If a process requests a resource that is currently allocated to another process, the operating system can preempt (temporarily revoke) the resource from the current process and allocate it to the requesting process.

Deadlock Prevention

4. Eliminate Circular Wait:

To violate circular wait, we can assign a priority number to each of the resource. A process can't request for a lesser priority resource. This ensures that not a single process can request a resource which is being utilized by some other process and no cycle will be formed.

-This ensures that processes never enter a circular wait state, breaking the circular wait condition and preventing deadlocks.

Deadlock Prevention

Among all the methods, eliminating Circular wait is the only approach that can be implemented practically.

Deadlock Avoidance

Banker's algorithm

-It is used for deadlock avoidance

-It is also used for deadlock detection.

Deadlock Avoidance: Banker's algorithm

-The banker's algorithm is used in the banking system to check whether a loan can be sanctioned to a person or not.

-Suppose there are n number of account holders in a bank and the total sum of their money is S.

-If a person applies for a loan then the bank first subtracts the loan amount from the total money that the bank has and if the remaining amount is greater than S then only the loan is sanctioned.

Deadlock Avoidance: Banker’s algorithm

Three things to consider in the banker's algorithm :

1. Request: How much each process can request for each resource. (i.e. [MAX] request).

2. Allocated: How much each process is currently holding each resource. ([ALLOCATED] resource).

3. Available: The number of each resource currently available. ([AVAILABLE] resource).

Deadlock Avoidance: Banker’s algorithm

The data structure used in Bankers' algorithm:

Available: It is an array of length 'm' that defines each type of resource available in the system. When Available[j] = K, means that 'K' instances of Resources type R[j] are available in the system.

Max: It is a [n x m] matrix that indicates each process P[i] can store the maximum number of resources R[j] (each type) in a system.

Deadlock Avoidance: Banker’s algorithm

The data structure used in Bankers' algorithm:

Allocation: It is a matrix of m x n orders that indicates the type of resources currently allocated to each process in the system. When Allocation [i, j] = K, it means that process P[i] is currently allocated K instances of Resources type R[j] in the system.

Need: It is an M x N matrix sequence representing the number of remaining resources for each process. When the Need[i] [j] = k, then process P[i] may require K more instances of resources type Rj to complete the assigned work.

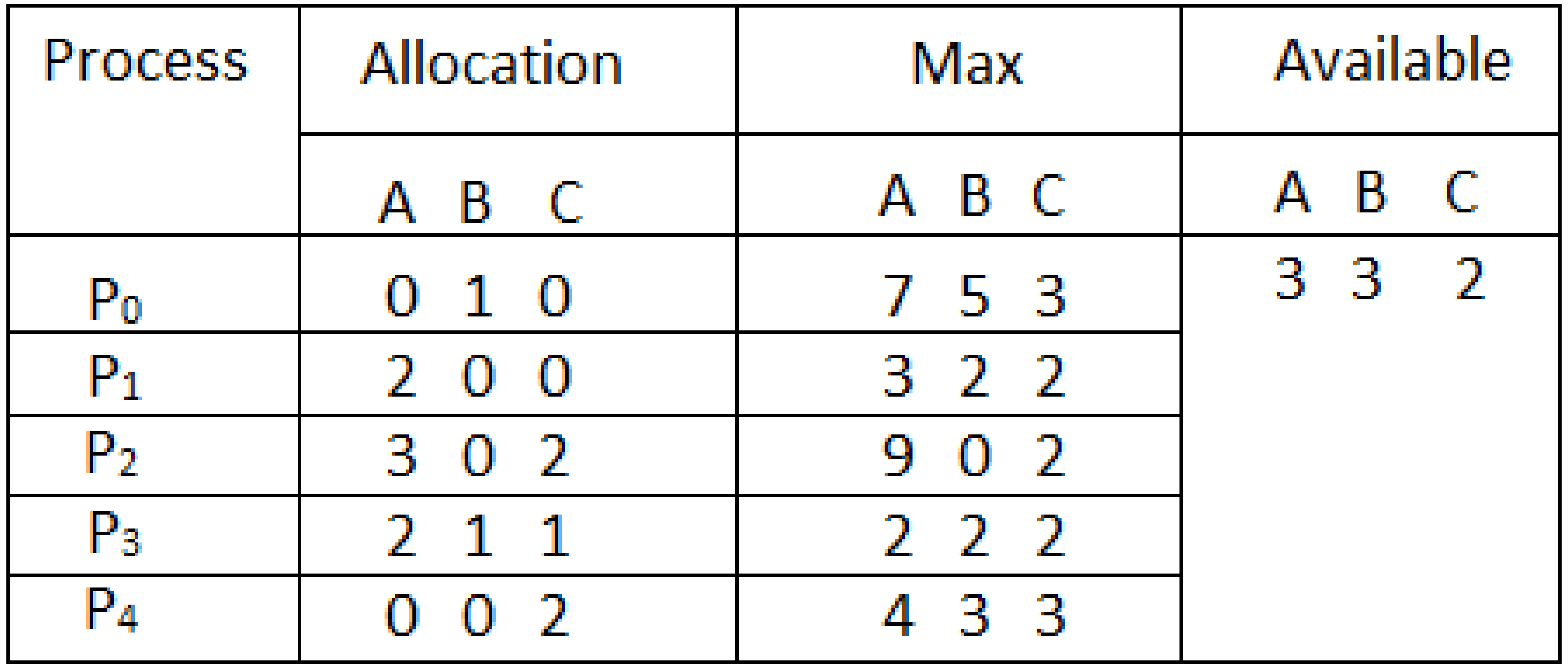
Nedd[i][j] = Max[i][j] - Allocation[i][j].

Deadlock Avoidance: Banker’s algorithm

The data structure used in Bankers' algorithm:
Finish: It is the vector of the order m. It includes a Boolean value (true/false) indicating whether the process has been allocated to the requested resources, and all resources have been released after finishing its task.

Deadlock Avoidance: Banker's algorithm

Problem: Considering a system with five processes P0 through P4 and three resources of type A, B, C. Resource type A has 10 instances, B has 5 instances and type C has 7 instances. Suppose at time t0 following snapshot of the system has been taken:



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			

Calculate:

1. Need matrix
2. Is there a deadlock?
3. If not, calculate the safe sequence.

Deadlock Avoidance: Banker’s algorithm

Total Resources: A=10, B = 5, C = 7

Available = Total resource – Total of allocation (A: 10-7=3, B: 5-2=3, C: 7-5=2)

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

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			

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

Safe sequence: P1-p3-p4-p0-p2

Deadlock Avoidance: Banker’s algorithm

Step 1: For Process P1:

Need <= Available

7, 4, 3 <= 3, 3, 2 condition is false.

So, we examine another process, P2.

Step 2: For Process P2:

Need <= Available

1, 2, 2 <= 3, 3, 2 condition true

New available = available + Allocation

(3, 3, 2) + (2, 0, 0) => 5, 3, 2

Similarly, we examine another process P3.

Step 3: For Process P3:

P3 Need <= Available

6, 0, 0 < = 5, 3, 2 condition is false.

Similarly, we examine another process, P4.

Deadlock Avoidance: Banker’s algorithm

Step 4: For Process P4:

P4 Need <= Available

0, 1, 1 <= 5, 3, 2 condition is true

New Available resource = Available + Allocation

5, 3, 2 + 2, 1, 1 => 7, 4, 3

Similarly, we examine another process P5.

Step 5: For Process P5:

P5 Need <= Available

4, 3, 1 <= 7, 4, 3 condition is true

New available resource = Available + Allocation

7, 4, 3 + 0, 0, 2 => 7, 4, 5

Now, we again examine each type of resource request for processes P1 and P3.

Deadlock Avoidance: Banker’s algorithm

Step 6: For Process P1:

P1 Need <= Available

7, 4, 3 <= 7, 4, 5 condition is true

New Available Resource = Available + Allocation

7, 4, 5 + 0, 1, 0 => 7, 5, 5

So, we examine another process P2.

Step 7: For Process P3:

P3 Need <= Available

6, 0, 0 <= 7, 5, 5 condition is true

New Available Resource = Available + Allocation

7, 5, 5 + 3, 0, 2 => 10, 5, 7

Hence, we execute the banker's algorithm to find the safe state and the safe sequence like **P2, P4, P5, P1 and P3.**

Deadlock Avoidance: Banker’s algorithm

Safety 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) Needi <= 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
```

Detecting deadlock using resource allocation graph

1. Using Single instance RAG
2. Using Multiple Instance RAG

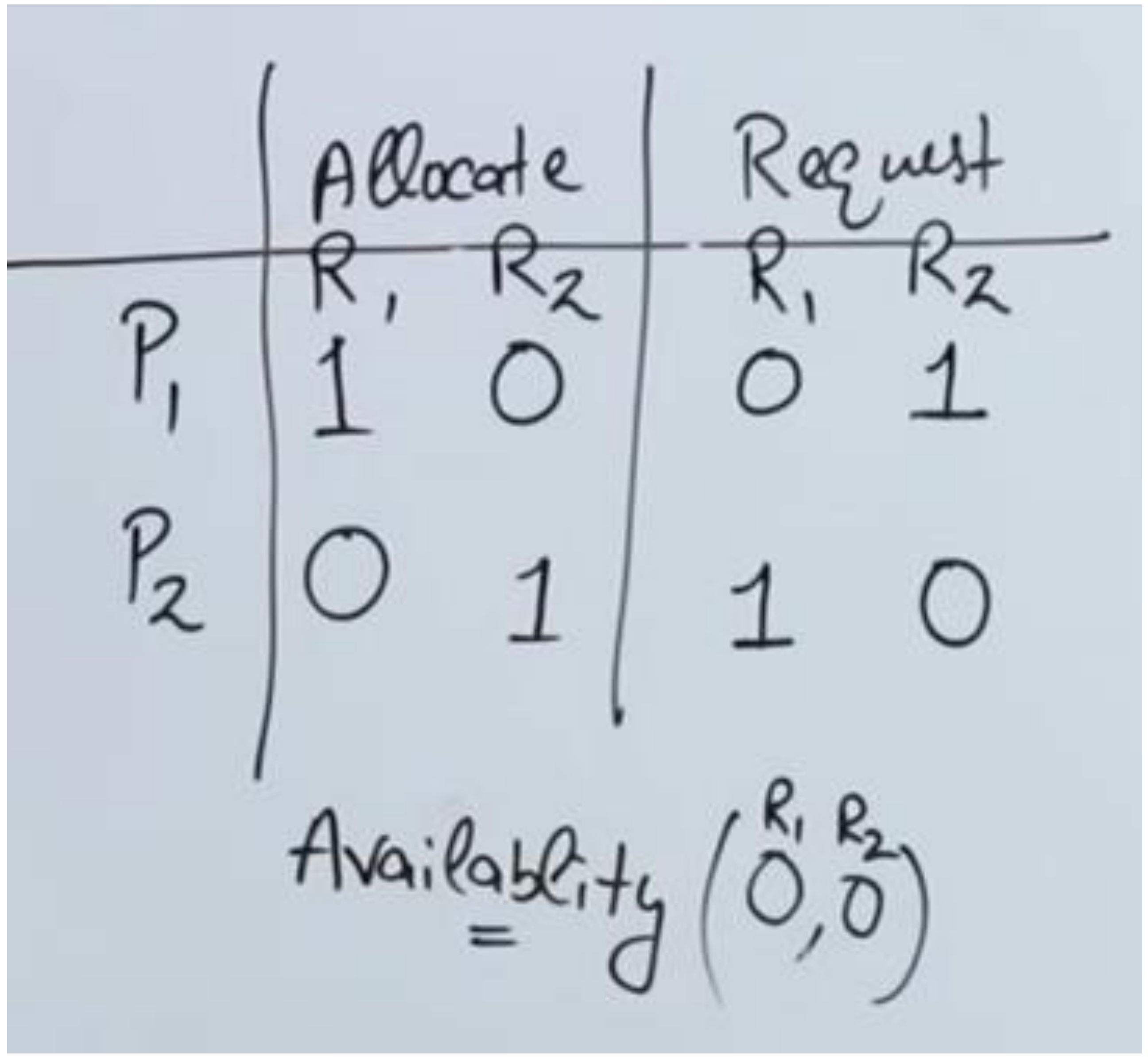
Deadlock detection using Single instances RAG

Problem: Check if deadlock occurs:

-There is a cycle so deadlock occurs.

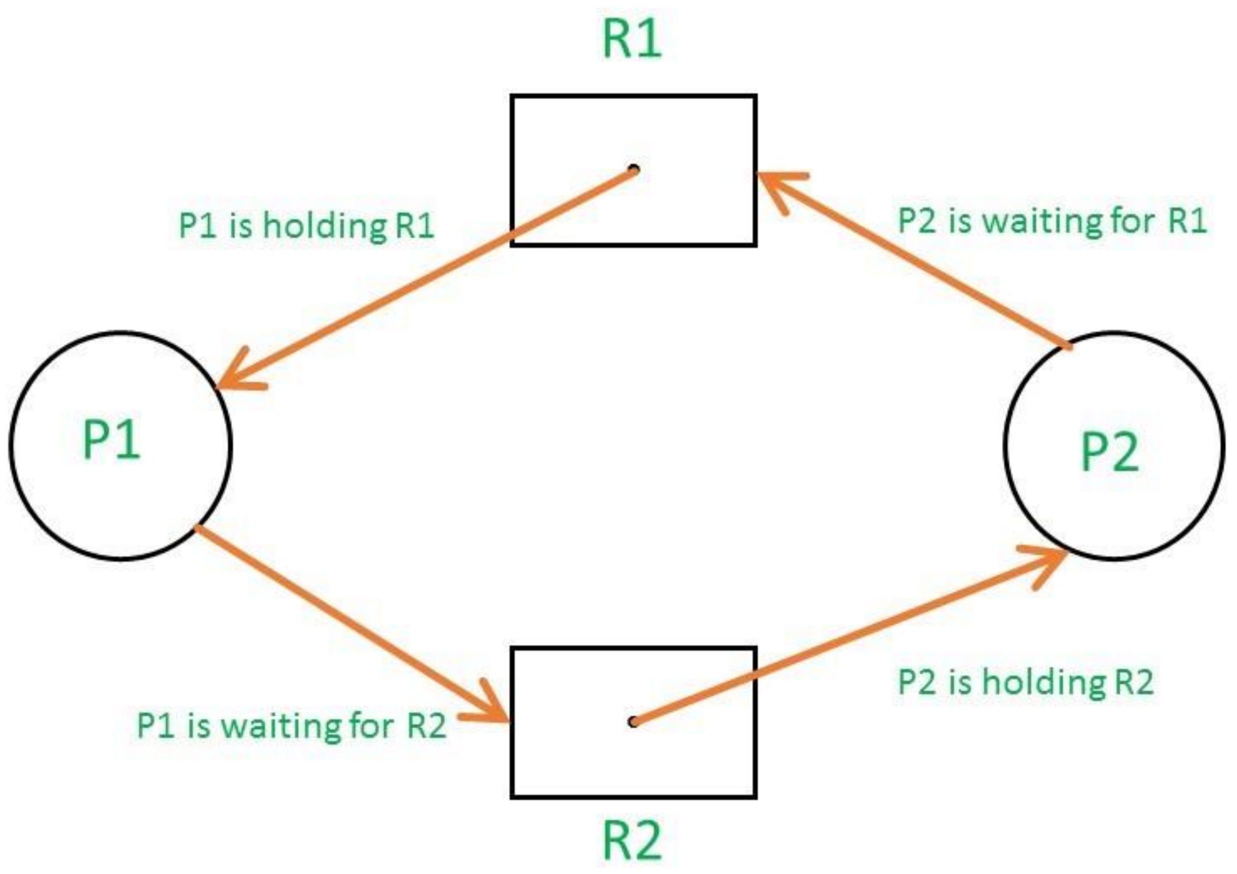
To verify:

From the table, We can't fulfill the request from availability. So, There is a deadlock



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

Availability = $\begin{pmatrix} R_1 & R_2 \\ 0, & 0 \end{pmatrix}$



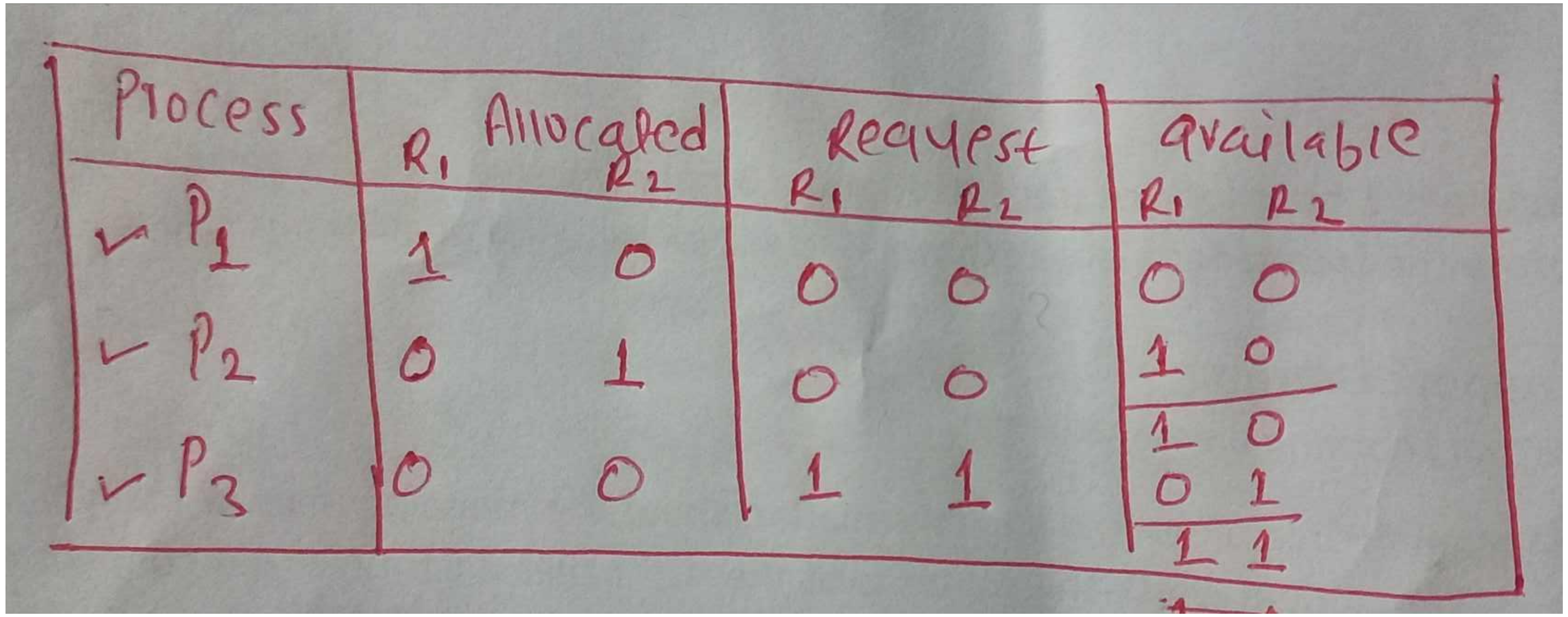
SINGLE INSTANCE RESOURCE TYPE WITH DEADLOCK

Deadlock detection using Single instances RAG

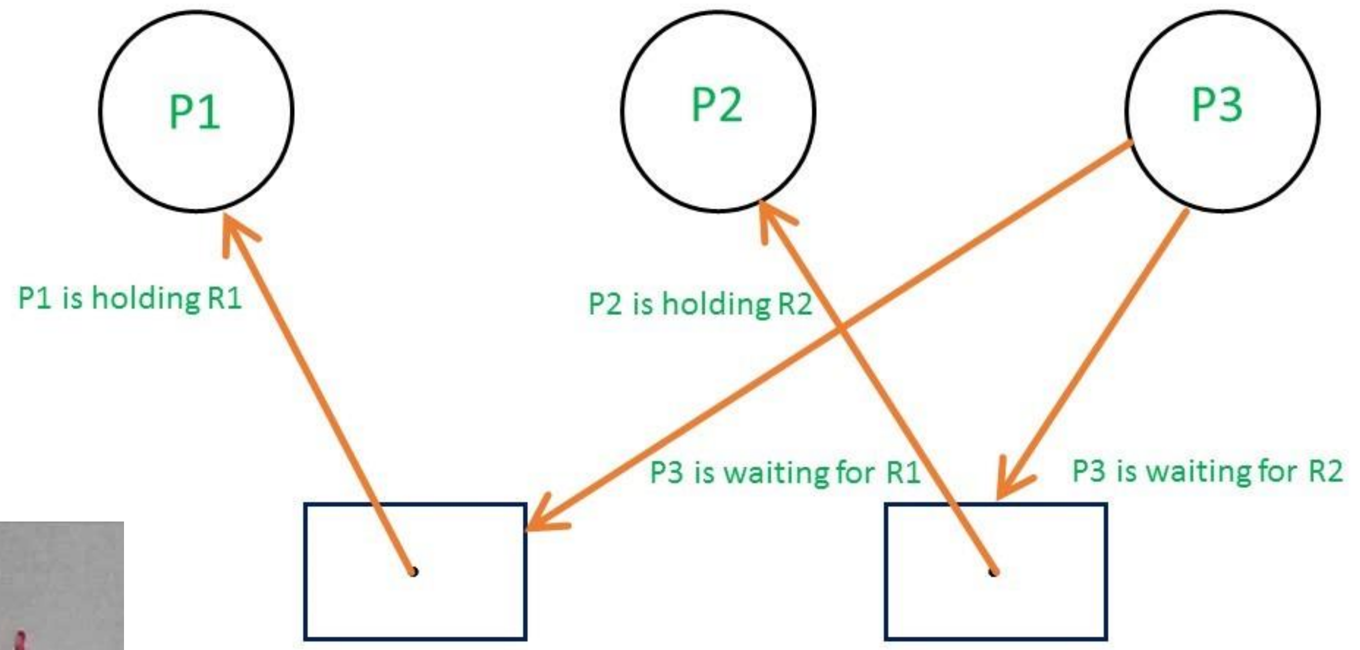
Problem: Check if deadlock occurs:

-There is no complete cycle so No deadlock.

From the table, We can fulfill the request from availability. So, There is NO deadlock .



Process	Allocated R1	Allocated R2	Request R1	Request R2	available R1	available R2
✓ P1	1	0	0	0	0	0
					1	0
✓ P2	0	1	0	0	1	0
					0	1
✓ P3	0	0	1	1	1	1



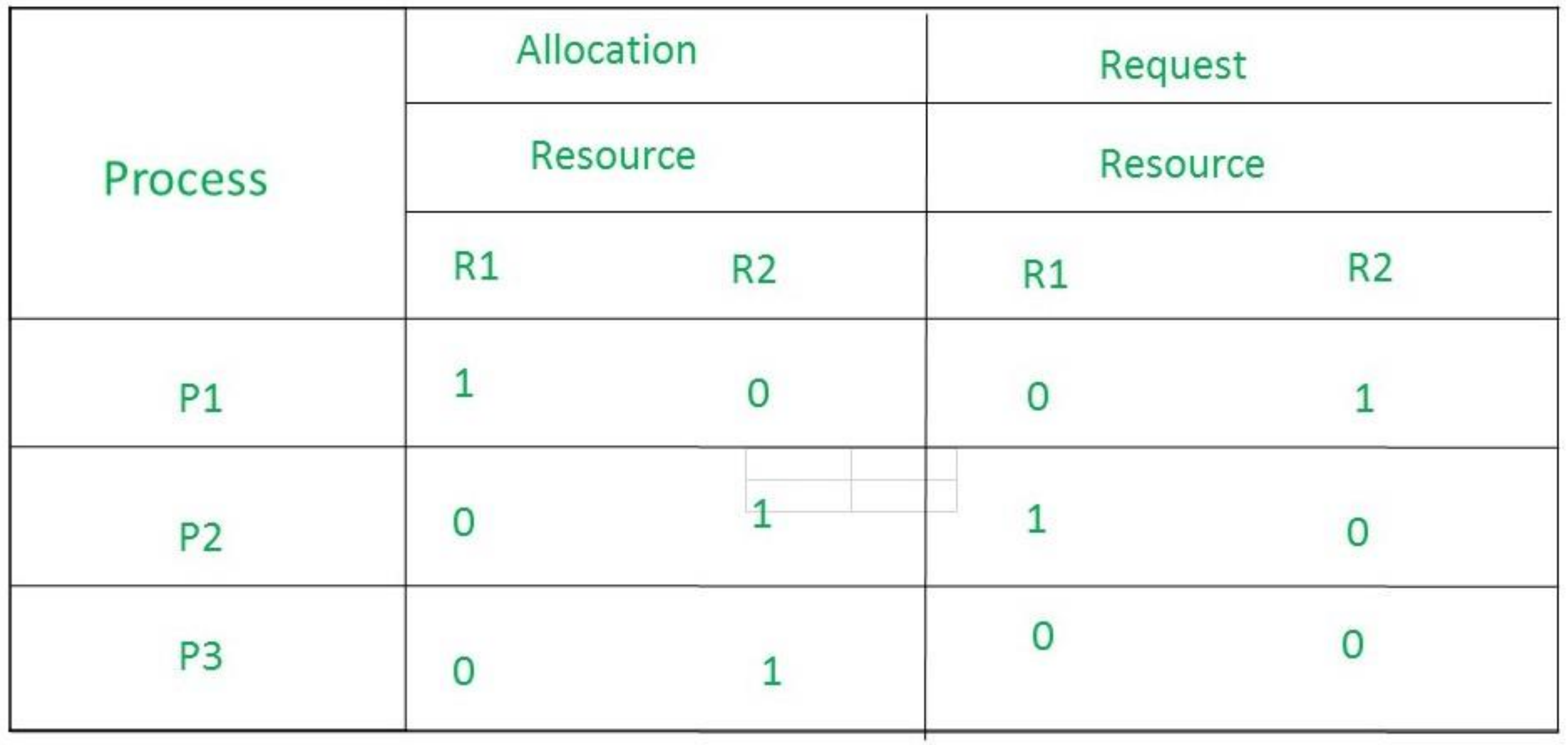
SINGLE INSTANCE RESOURCE TYPE WITHOUT DEADLOCK

Deadlock detection using MULTIPLE instances RAG

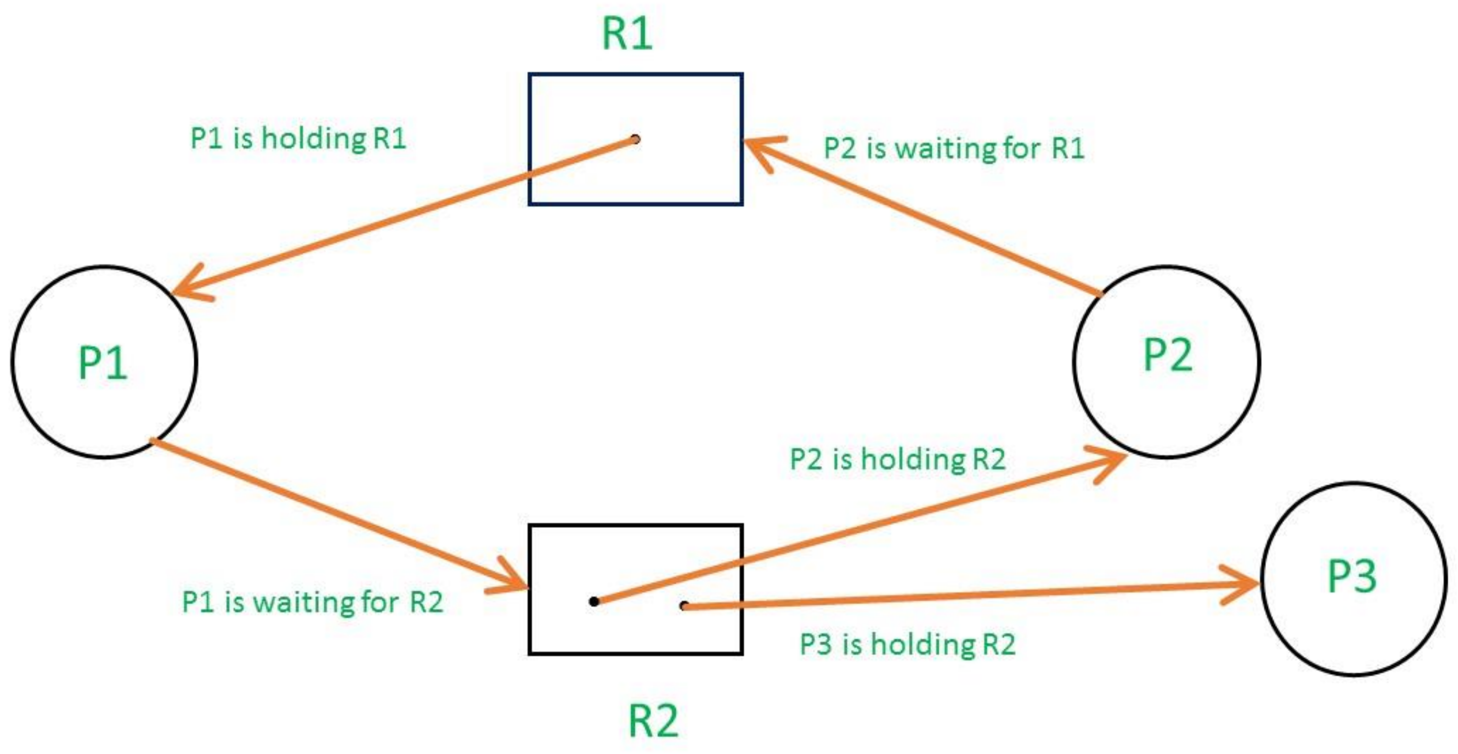
Problem: Check if deadlock occurs:

-There is no complete cycle so No deadlock.

From the table, We can fulfill the request from availability. So, There is NO deadlock.



Process	Allocation		Request	
	Resource		Resource	
	R1	R2	R1	R2
P1	1	0	0	1
P2	0	1	1	0
P3	0	1	0	0



MULTI INSTANCES WITHOUT DEADLOCK

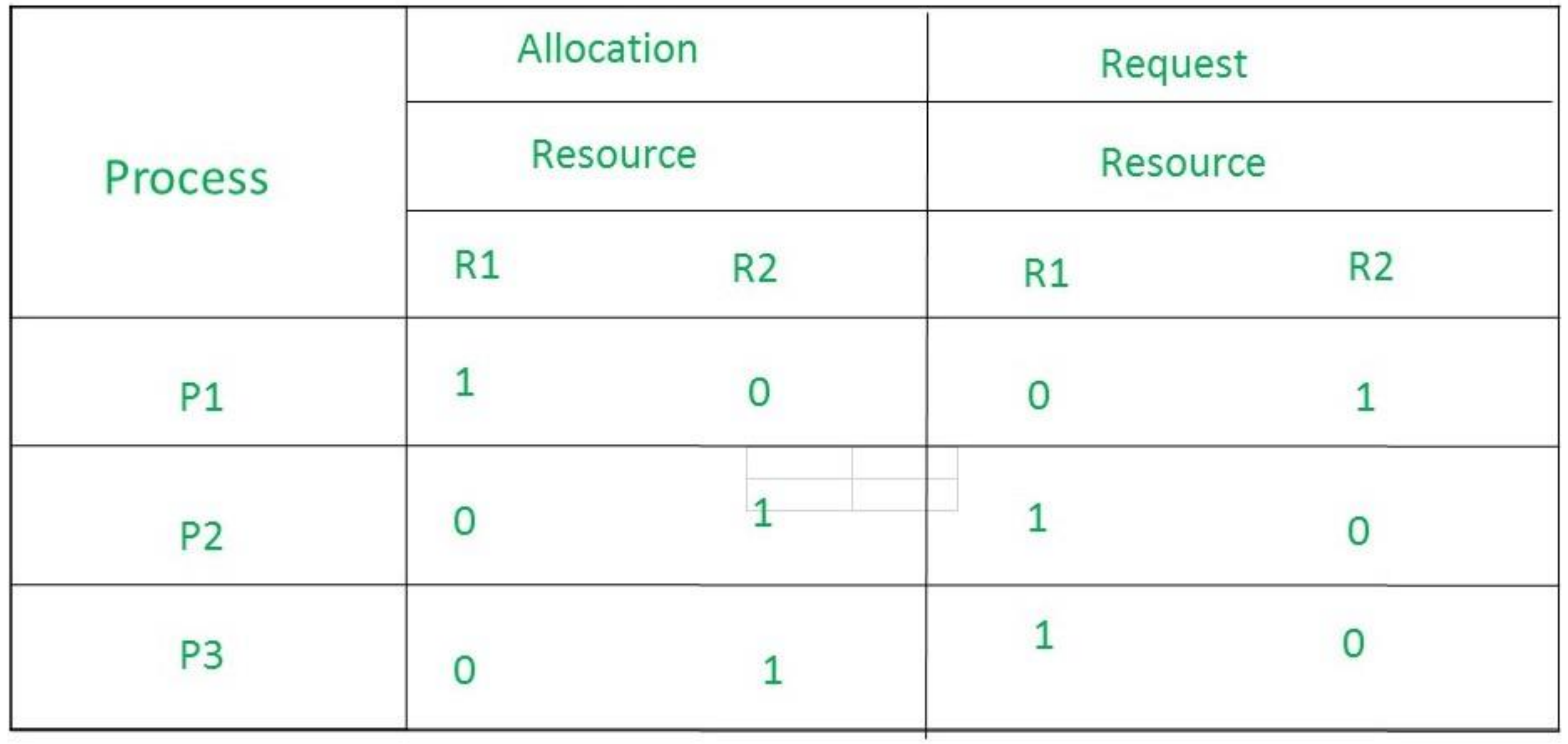
Available:
R1: 0
R2: 0

Deadlock detection using MULTIPLE instances RAG

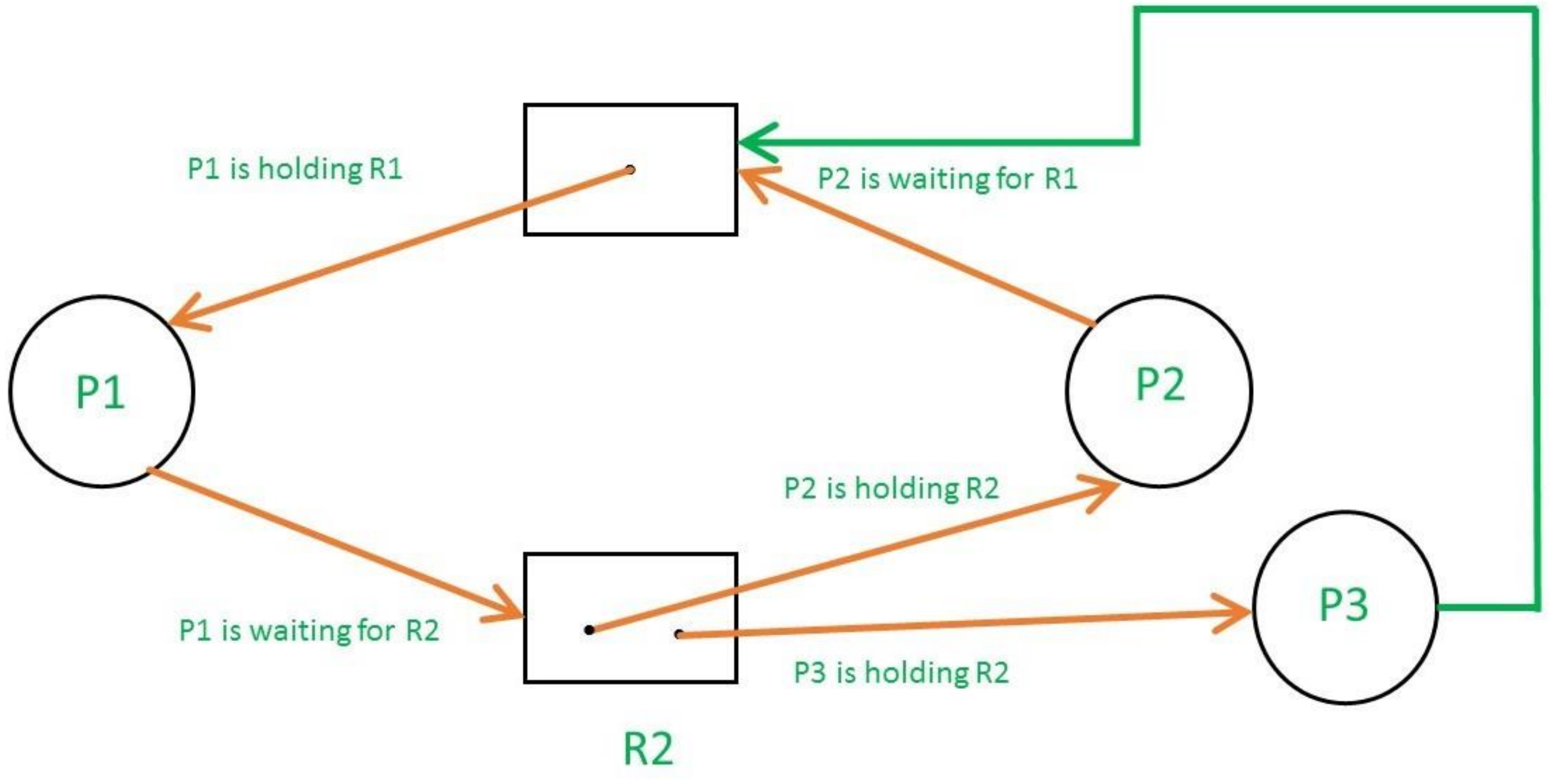
Problem: Check if deadlock occurs:

-There is complete complete cycle so deadlock.

From the table, We can't fulfill the request from availability. So, There is deadlock.



Process	Allocation		Request	
	Resource		Resource	
	R1	R2	R1	R2
P1	1	0	0	1
P2	0	1	1	0
P3	0	1	1	0



MULTI INSTANCES WITH DEADLOCK

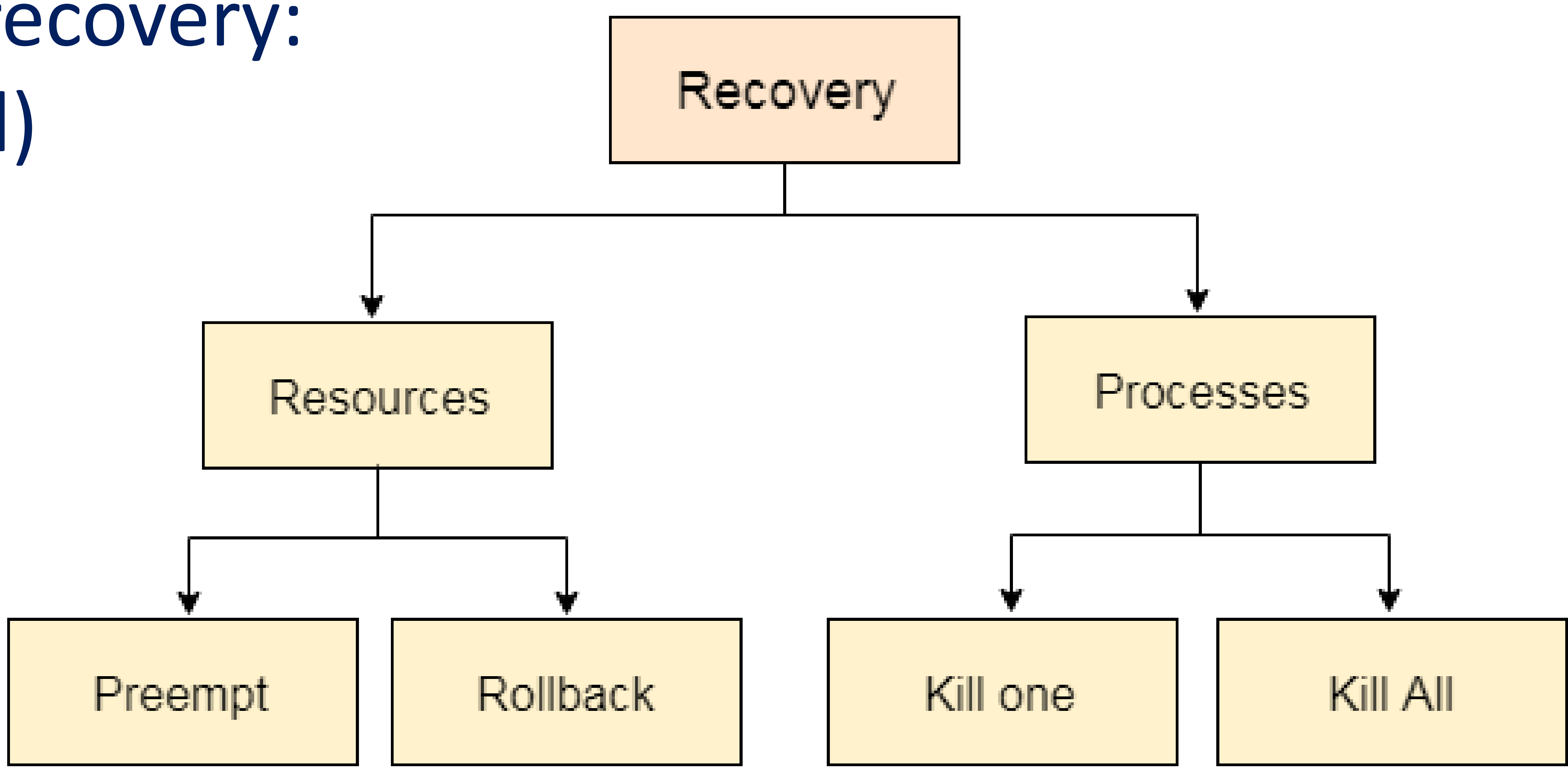
Available:
R1: 0
R2: 0

Deadlock Recovery

Deadlock recovery is a critical process that is initiated after a deadlock has been detected in a computer system.

Four methods of deadlock recovery:

1. Process termination (kill)
2. Priority inversion
3. Resource pre-emption
4. Resource Rollback.



Deadlock Recovery

Process Termination:
-In this method, the operating system identifies the processes involved in the deadlock and terminates one or more processes.
-This frees up the resources held by the terminated processes, which can be used by the remaining processes to continue their execution.

Drawbacks: loss of data, abrupt termination of processes, and inconsistency in the system.

Deadlock Recovery

Priority Inversion

-In this method, the priority of the processes is changed to avoid deadlock situations.

-The process holding the required resources is given a higher priority, and the process waiting for the resources is given a lower priority.

-this method can also lead to starvation of lower priority processes, as higher priority processes can keep preempting the resources.

Deadlock Recovery

Resource Pre-emption

-It is a complex method for resolving deadlocks.

-In this method, the operating system identifies the resources involved in the deadlock and selects one or more resources to be pre-empted.

-The resources are then taken away from the process holding them and allocated to the waiting processes.

-The pre-empted process is suspended until the required resources become available again.

Deadlock Recovery

Rollback

-Rollback is a method for resolving deadlocks that is commonly used in database systems.

-In this method, the system rolls back the transactions of the involved processes to a previous state where they were not deadlocked.

-This method can cause significant delays in the execution of the transactions and can result in a loss of data.

Find me

9851083215

Santosh.it288@mail.com

www.phtechno.com

Kathmandu