Operating System

Unit – 4
(Part-A)
Deadlock

Mayank Mishra
School of Electronics Engineering
KIIT-Deemed to be University

Introduction

- In a multiprogramming environment, several processes may compete for a finite number of resources.

- A process requests resources; if the resources are not available at that time, the process enters a waiting state.

- Sometimes, a waiting process can never again change state, because the resources it has requested are held by other waiting processes. This situation is called a **deadlock**.

 # Perhaps the best illustration of a deadlock can be drawn from a law passed by the Kansas legislature early in the 20th century. It said, in part: "When two trains approach each other at a crossing, both shall come to a full stop and neither shall start up again until the other has gone."

System Model

- A system consists of a finite number of resources to be distributed among a number of competing processes.
- The resources may be partitioned into several types (or classes), each consisting of some number of identical instances.
- CPU cycles, files, and I/O devices (such as network interfaces and DVD drives) are examples of resource types.

System Model (Contd.)

Under the normal mode of operation, a process may utilize a resource in only the following sequence:

- **Request:** The process requests the resource. If the request cannot be granted immediately (for example, if a mutex lock is currently held by another process), then the requesting process must wait until it can acquire the resource.
- **Use:** The process can operate on the resource (for example, if the resource is a printer, the process can print on the printer).
- **Release:** The process releases the resource.

Deadlock Characterization

- In a deadlock, processes never finish executing, and system resources are tied up, preventing other jobs from starting.

- A deadlock situation can arise if the following four conditions hold simultaneously in a system:
  - Mutual exclusion.
  - Hold and wait.
  - No pre-emption.
  - Circular wait.

Deadlock Characterization (Contd.)

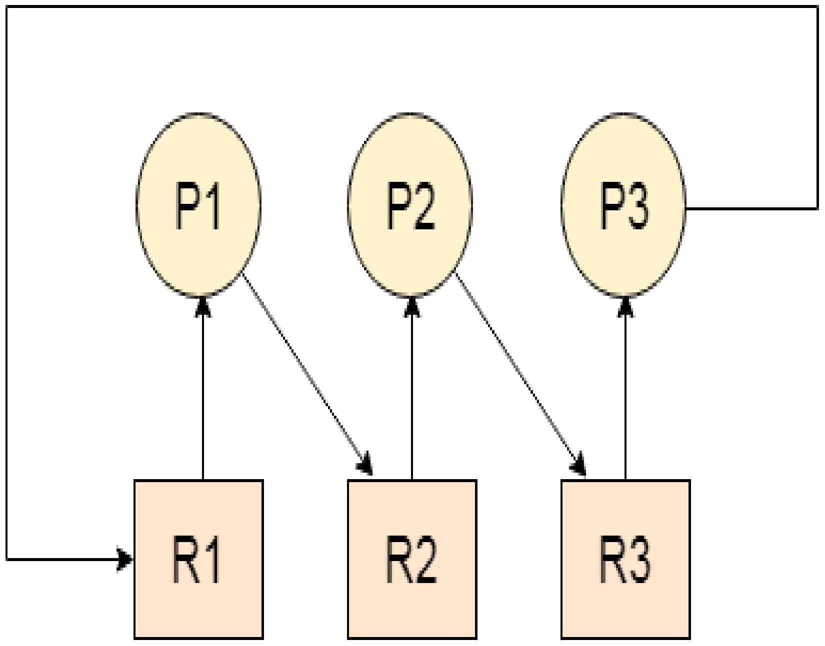
- ***Mutual exclusion:*** At least one resource must be held in a non-sharable mode; that is, only one process at a time can use the resource. If another process requests that resource, the requesting process must be delayed until the resource has been released.

- ***Hold and wait:*** A process must be holding at least one resource and waiting to acquire additional resources that are currently being held by other processes.

- ***No preemption:*** Resources cannot be preempted; that is, a resource can be released only voluntarily by the process holding it, after that process has completed its task.

Deadlock Characterization (Contd.)

- ***Circular wait:*** A set{P0,P1,...,Pn} of waiting processes must exist such that P0 is waiting for a resource held by P1, P1 is waiting for a resource held by P2, ..., Pn−1 is waiting for a resource held by Pn, and Pn is waiting for a resource held by P0.
  - Let us assume that there are three processes P1, P2 and P3. There are three different resources R1, R2 and R3. R1 is assigned to P1, R2 is assigned to P2 and R3 is assigned to P3.
  - After some time, P1 demands for R1 which is being used by P2. P1 halts its execution since it can't complete without R2. P2 also demands for R3 which is being used by P3. P2 also stops its execution because it can't continue without R3. P3 also demands for R1 which is being used by P1 therefore P3 also stops its execution.
  - In this scenario, a cycle is being formed among the three processes. None of the process is progressing and they are all waiting. The computer becomes unresponsive since all the processes got blocked.



Resource Allocation Graph (RAG)

- A **Resource Allocation Graph (RAG)** is a visual way to understand how **resources** are assigned in an **operating system**.

- Instead of using only tables to show which resources are allocated, requested, or available, the RAG uses nodes and edges to clearly illustrate relationships between **processes** and their required resources.

- RAGs primarily help in detecting deadlocks by visually representing the relationships between processes and resources, making it easier to identify potential deadlock conditions.

RAG (Contd.)

Types of Vertices in RAG

In a Resource Allocation Graph, there are two types of vertices

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

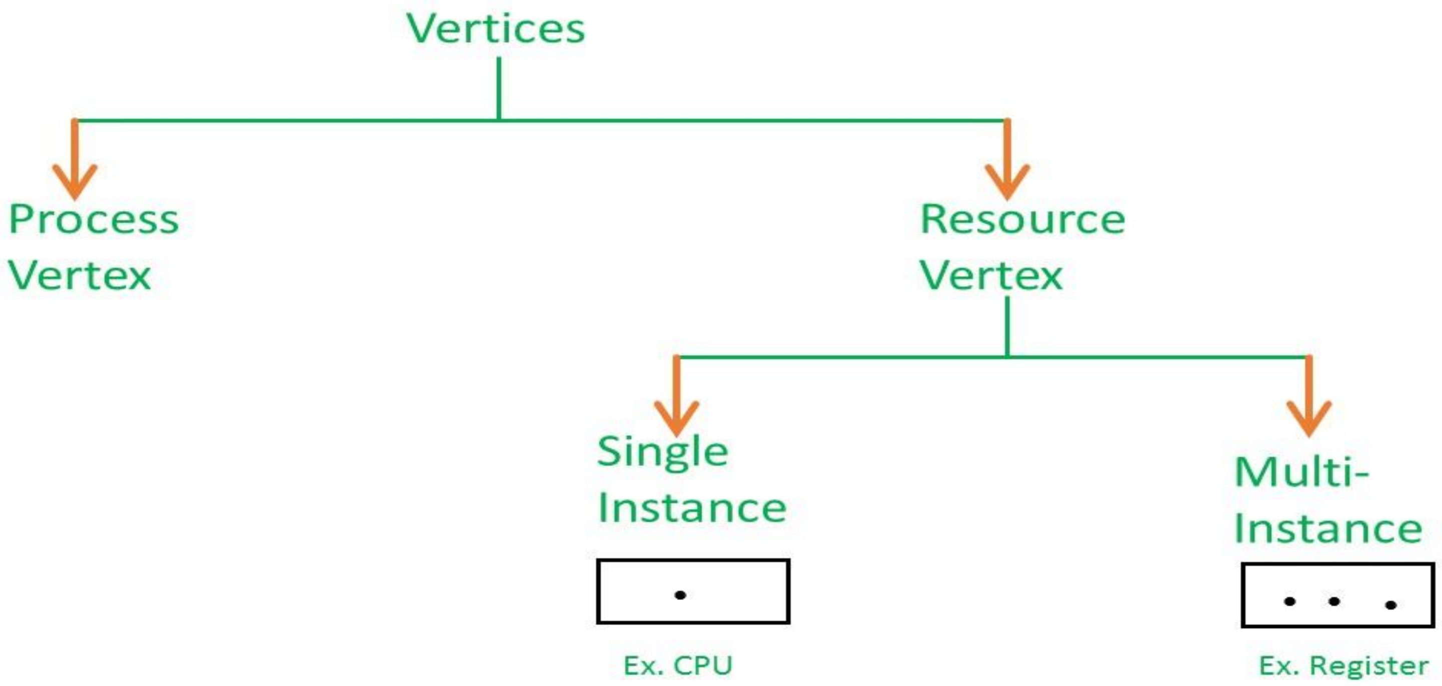
2. Resource Vertex: Every resource will be represented as a resource vertex. It is also two types:

- **Single Instance Type Resource:** A Single Instance Resource refers to a type of resource in the system that has only one available instance or copy. In the Resource Allocation Graph (RAG), a single instance resource is represented as a single resource node, and only one process can be assigned to this resource at any time.

- **Multi-Resource Instance Type Resource:** A Multi Instance Resource refers to a type of resource that has multiple instances available. In a Resource Allocation Graph (RAG), a multi-instance resource has multiple resource nodes connected to process nodes, allowing multiple processes to request and hold instances of that resource simultaneously.

RAG (Contd.)

Types of Vertices in RAG

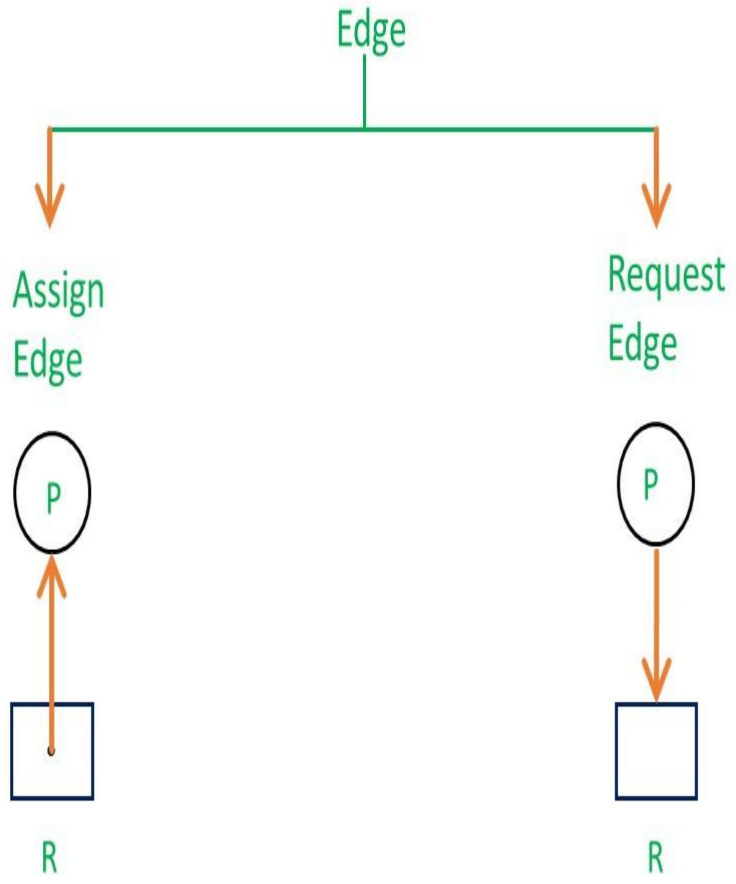


RAG (Contd.)

Types of Edges in RAG

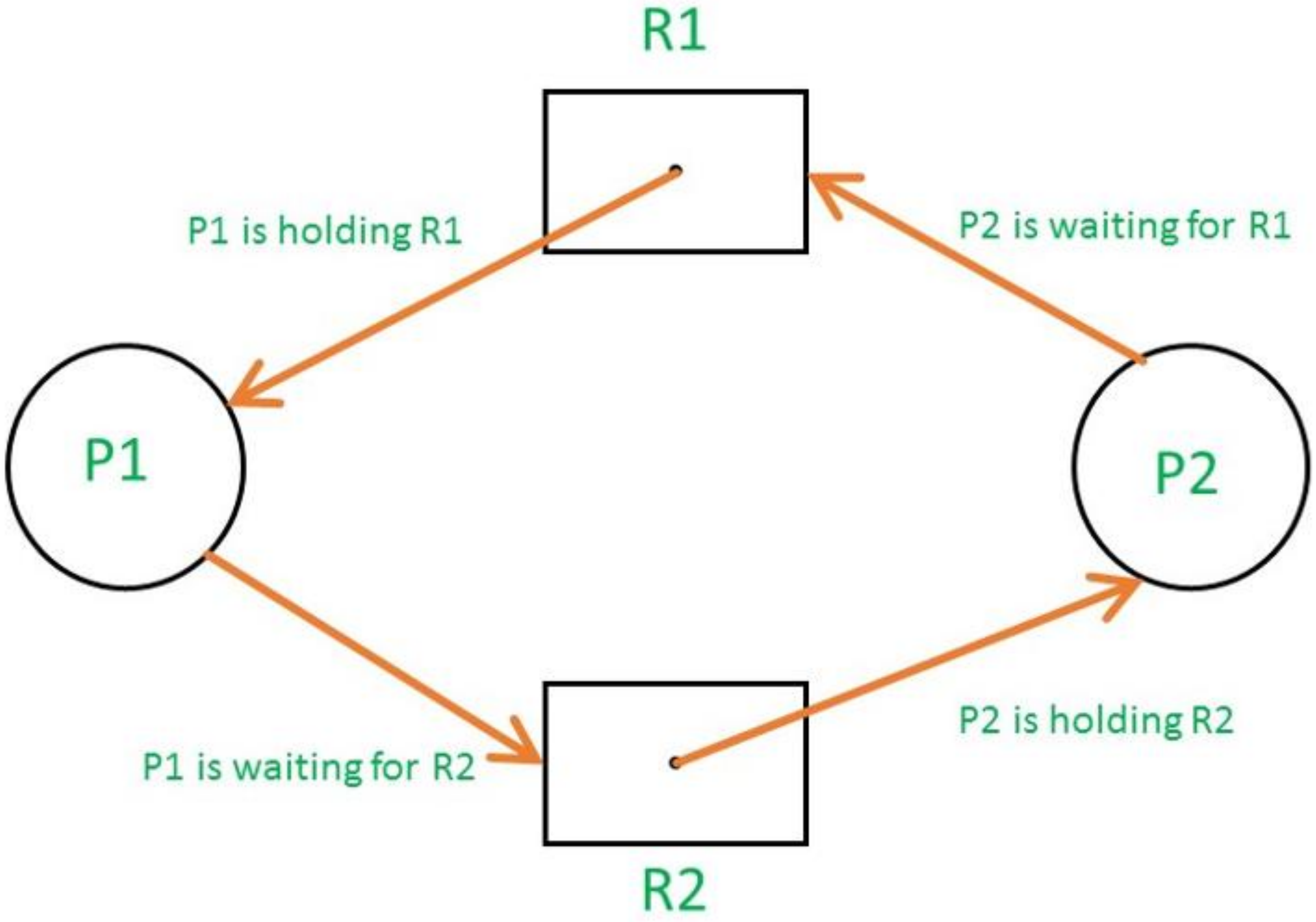
There are two types of edges in RAG:

- **Assign Edge:** If you already assign a resource to a process then it is called Assign edge. This is shown by an arrow from the resource vertex to the process vertex.
- **Request Edge**: A request edge represents that a process is currently requesting a resource. This is shown by an arrow from the process vertex to the resource vertex.



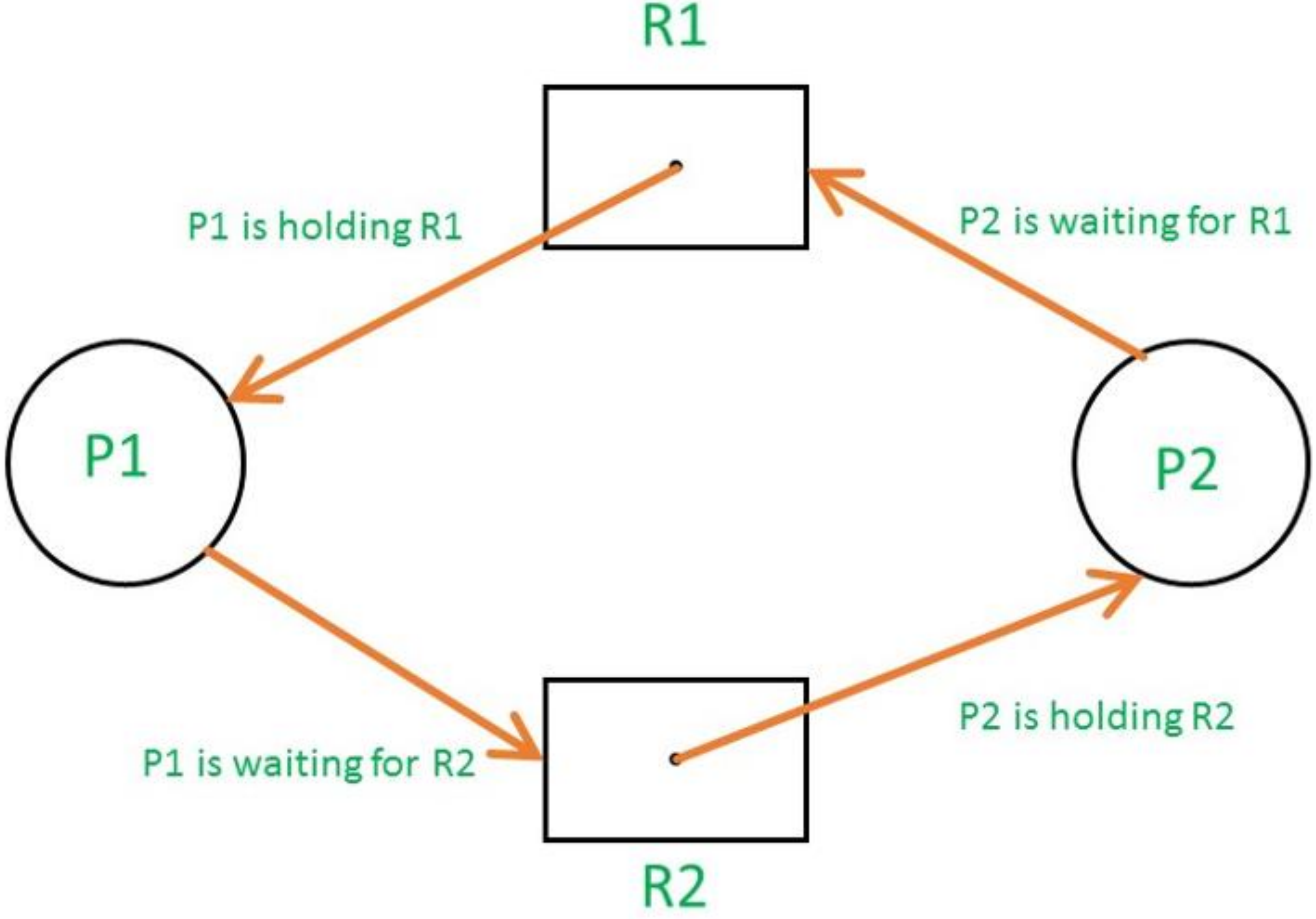
 # If a process is using a resource, an arrow is drawn from the resource node to the process node. If a process is requesting a resource, an arrow is drawn from the process node to the resource node.

Example 1 (Single Instances RAG)



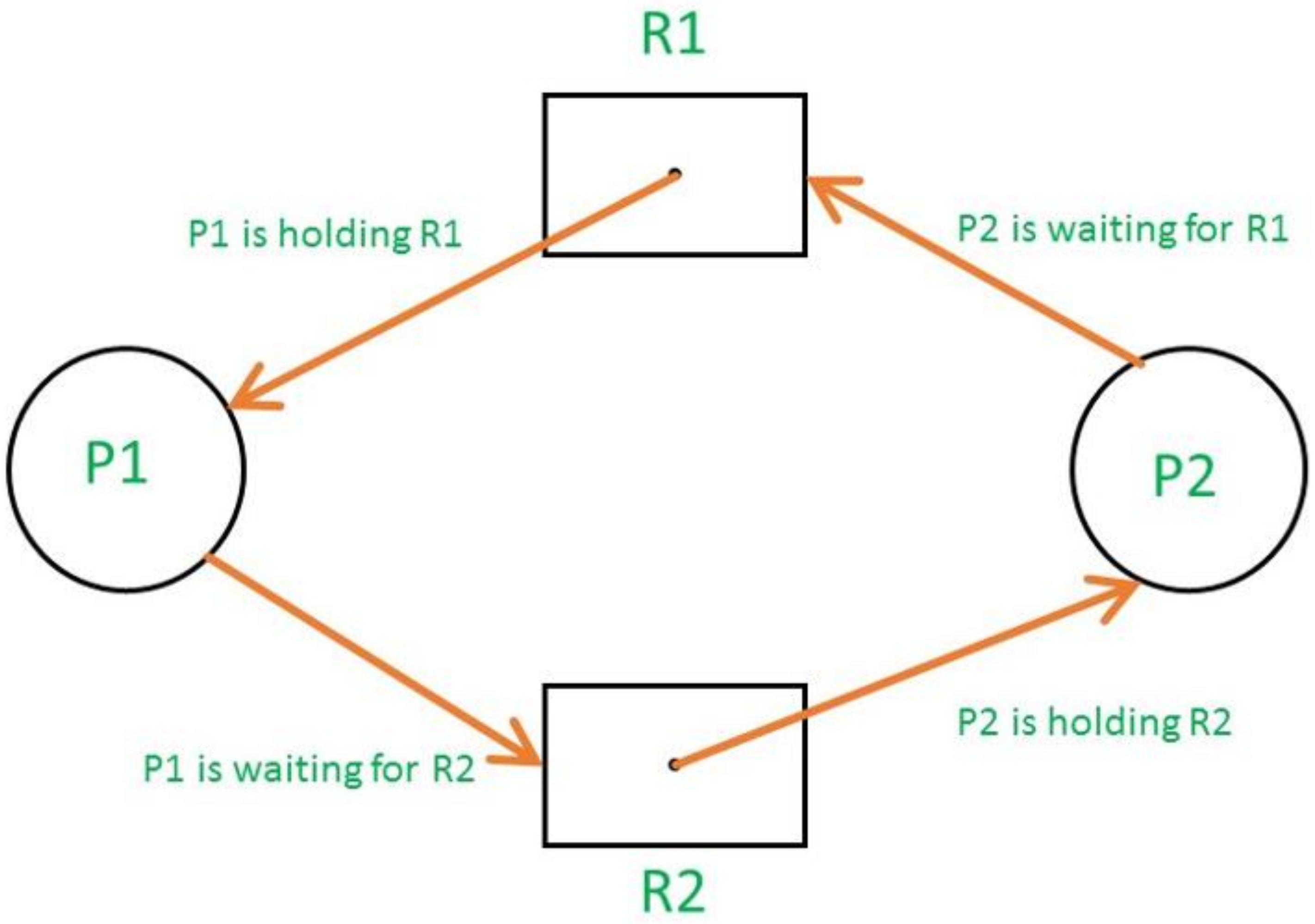
Does it contains Deadlock?

Example 1 (Single Instances RAG)



Let’s see, whether this RAG has cycle:______?

Example 1 (Single Instances RAG)



Let's see, whether this RAG has cycle: YES

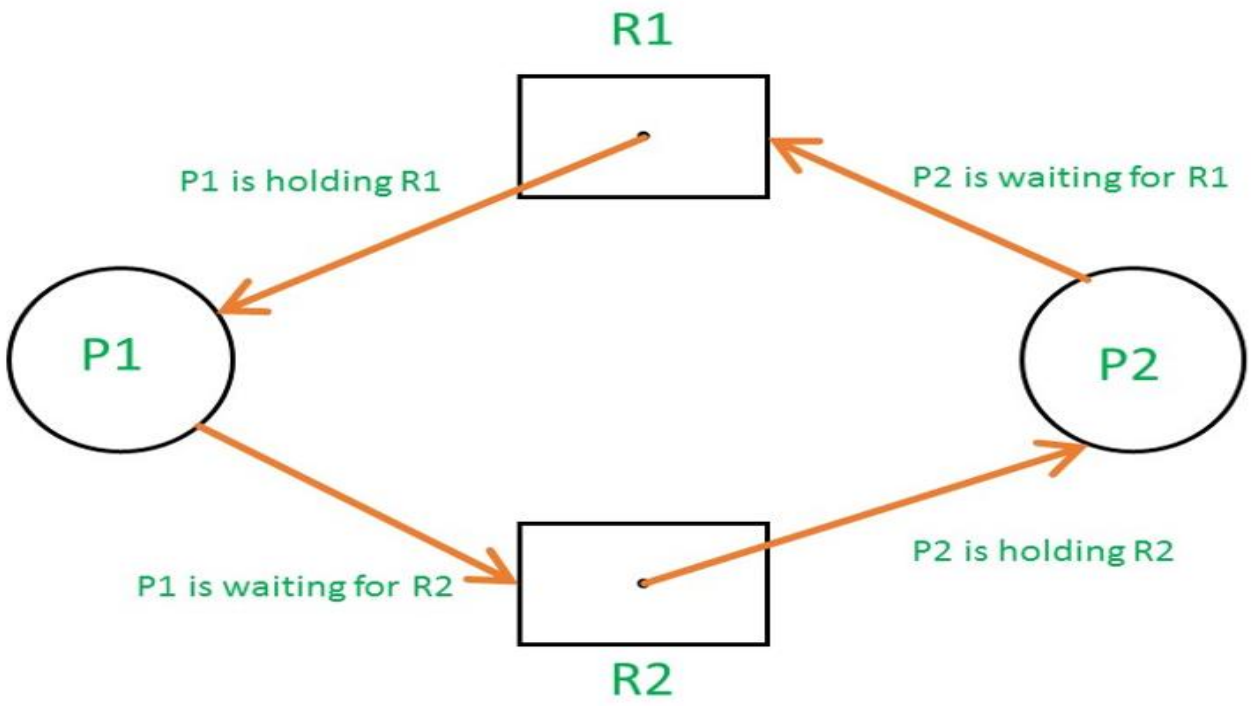
Example 1 (Single Instances RAG)

- If RAG has circular wait (cycle), it will always have Deadlock.

 # Above statement is only true for Single Instance Scenario

Example 1 (Single Instances RA

Other approach to check



Process	Allocation		Request	
	R1	**R2**	**R1**	**R2**
P1	1	0	0	1
P2	0	1	1	0

Example 1 (Single Instances RAG)

Other approach to check

Process	Allocation		Request	
	R1	**R2**	**R1**	**R2**
P1	1	0	0	1
P2	0	1	1	0

Availability = (R1, R2) = (0,0)

Can the request of P1 and P2 be fulfilled?

Example 1 (Single Instances RAG)

Other approach to check

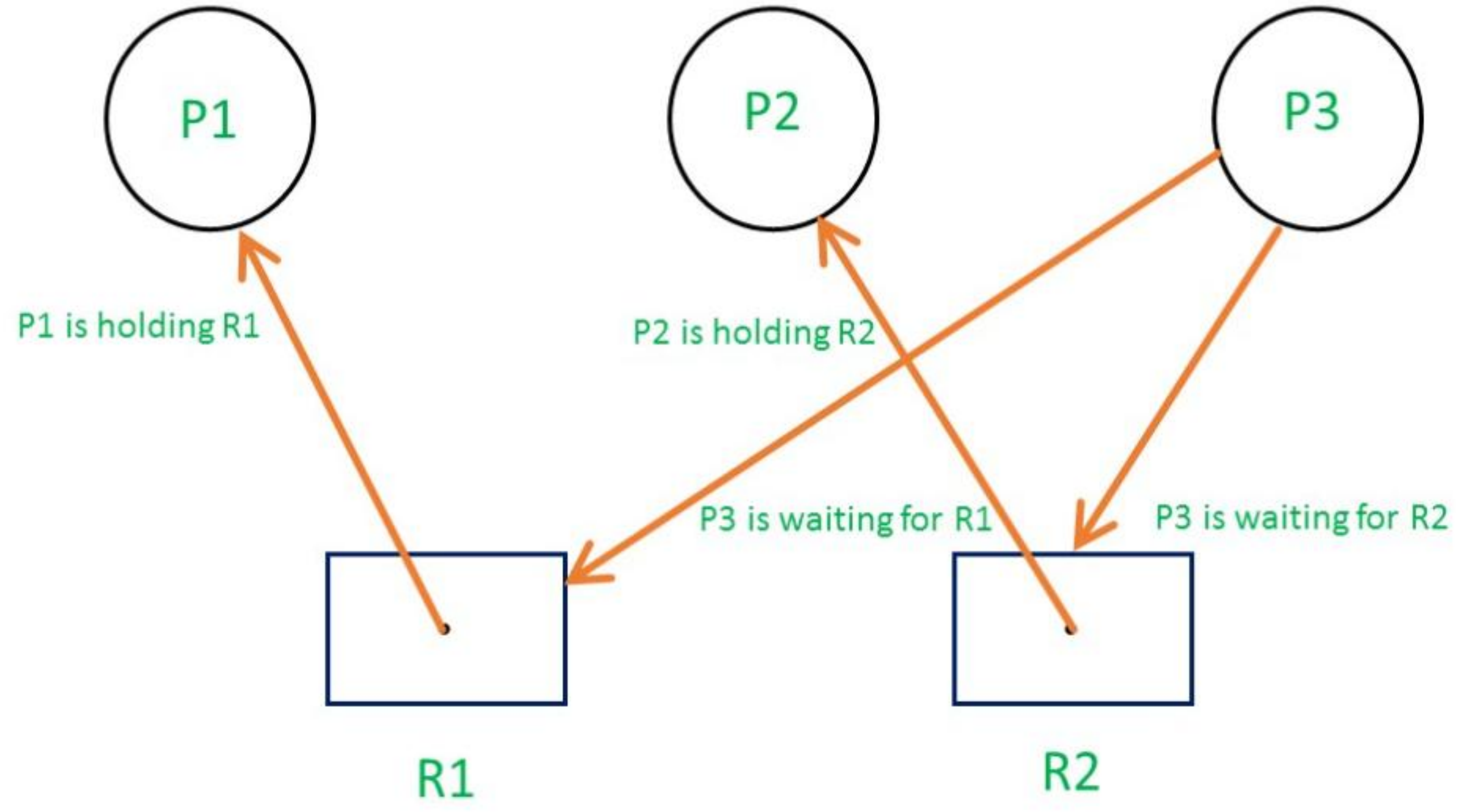
Process	Allocation		Request	
	R1	**R2**	**R1**	**R2**
P1	1	0	0	1
P2	0	1	1	0

Availability = (R1, R2) = (0,0)

Can the request of P1 and P2 be fulfilled? - **No**

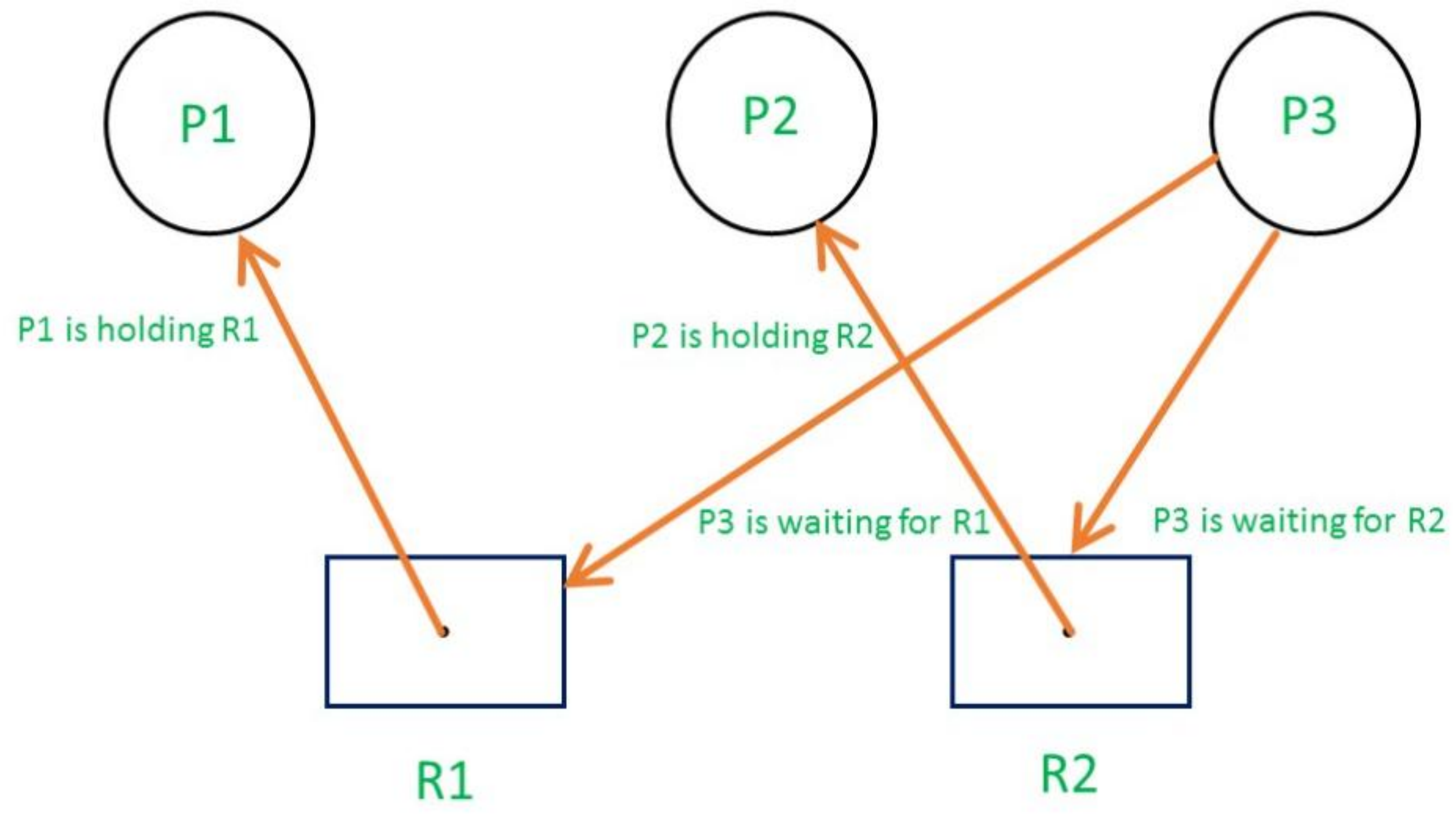
In this RAG, Deadlock is present.

Example 2 (Single Instances RAG)



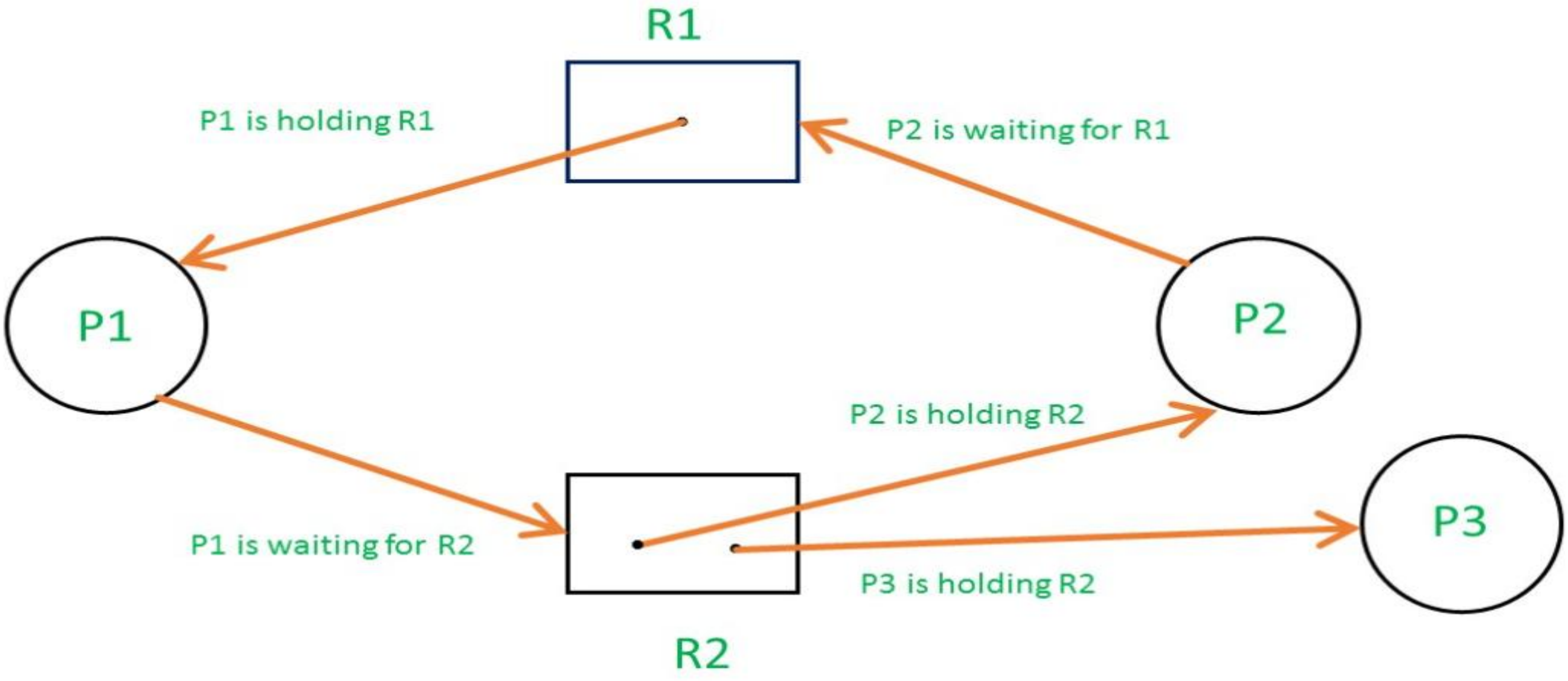
Does it contains Deadlock?

Example 2 (Single Instances RAG)

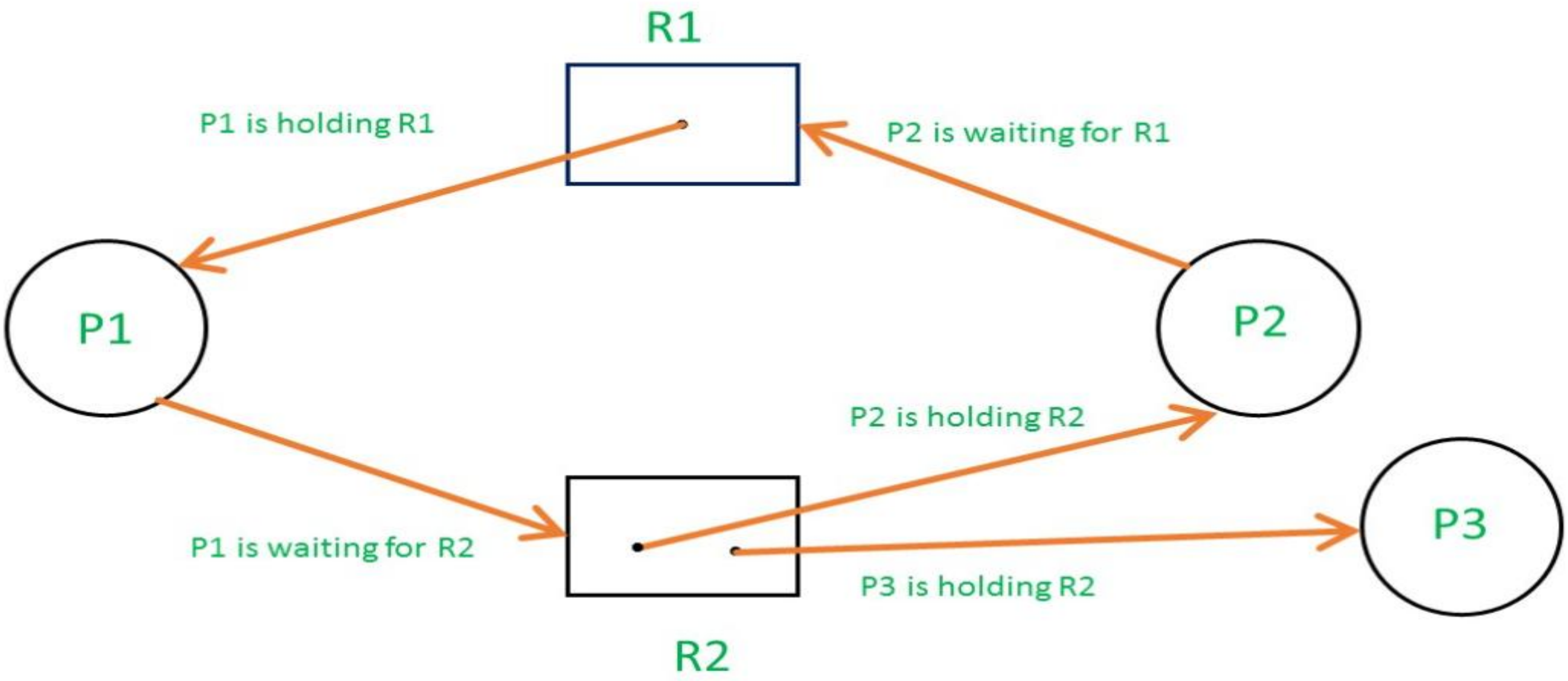


Does it contains Deadlock?-- NO

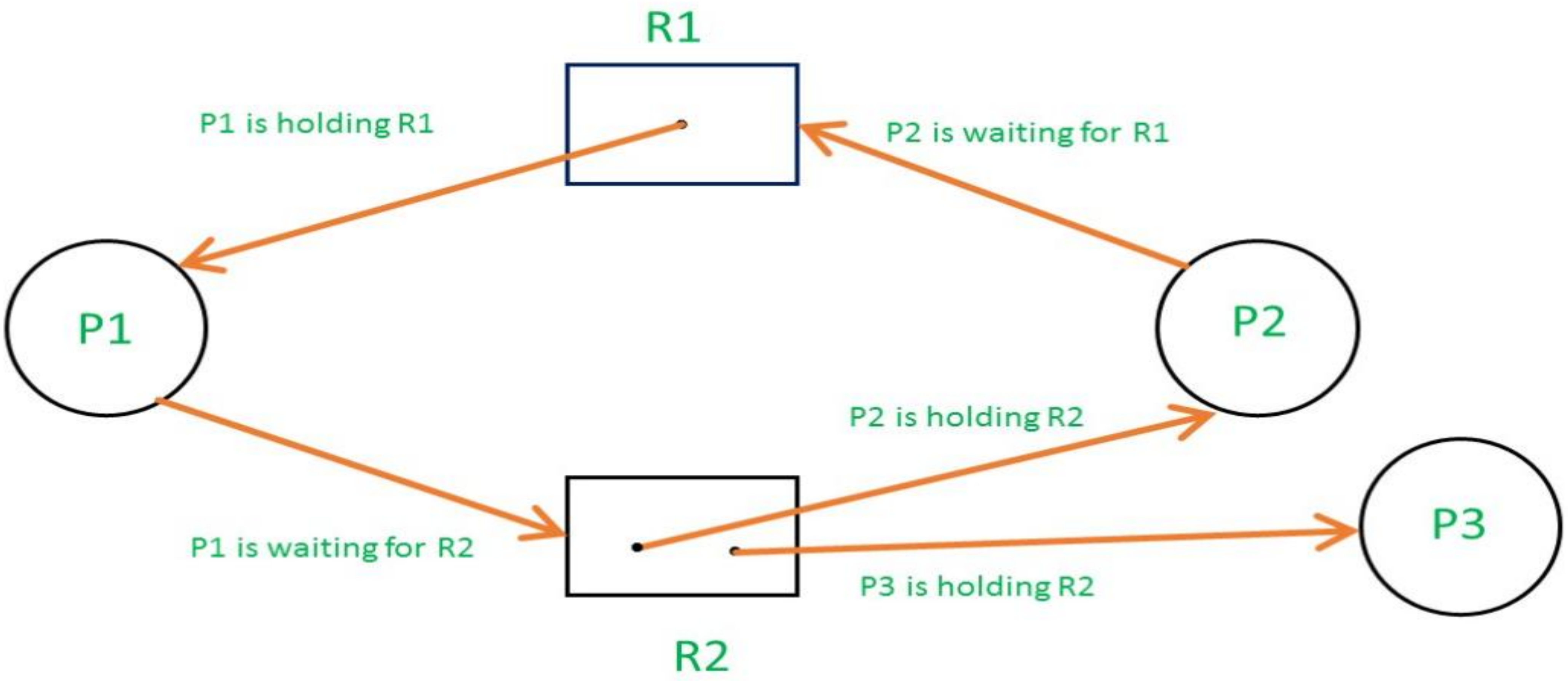
Example 3 (Multi Instances RAG)



Does it contains Deadlock?



Let’s see, whether this RAG has cycle:______?



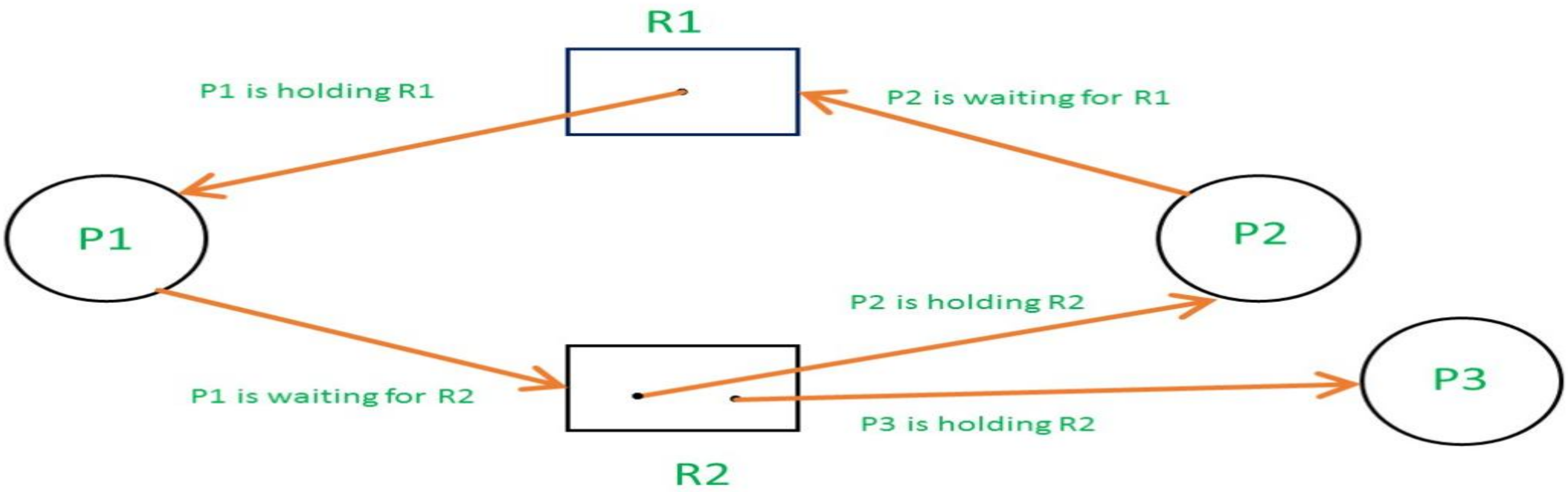
Let’s see, whether this RAG has cycle: YES

Let’s see, whether this RAG has cycle: YES

Can there will be Deadlock?

 # As observed in the Single Instance Scenario

Let’s check with the other approach!!!!



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

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

Available = 0 0 (As P3 does not require any extra resource to complete the execution and after
P3 0 1 completion P3 release its own resource)

New Available = 0 1 (As using new available resource we can satisfy the requirment of process P1
P1 1 0 and P1 also release its prevous resource)

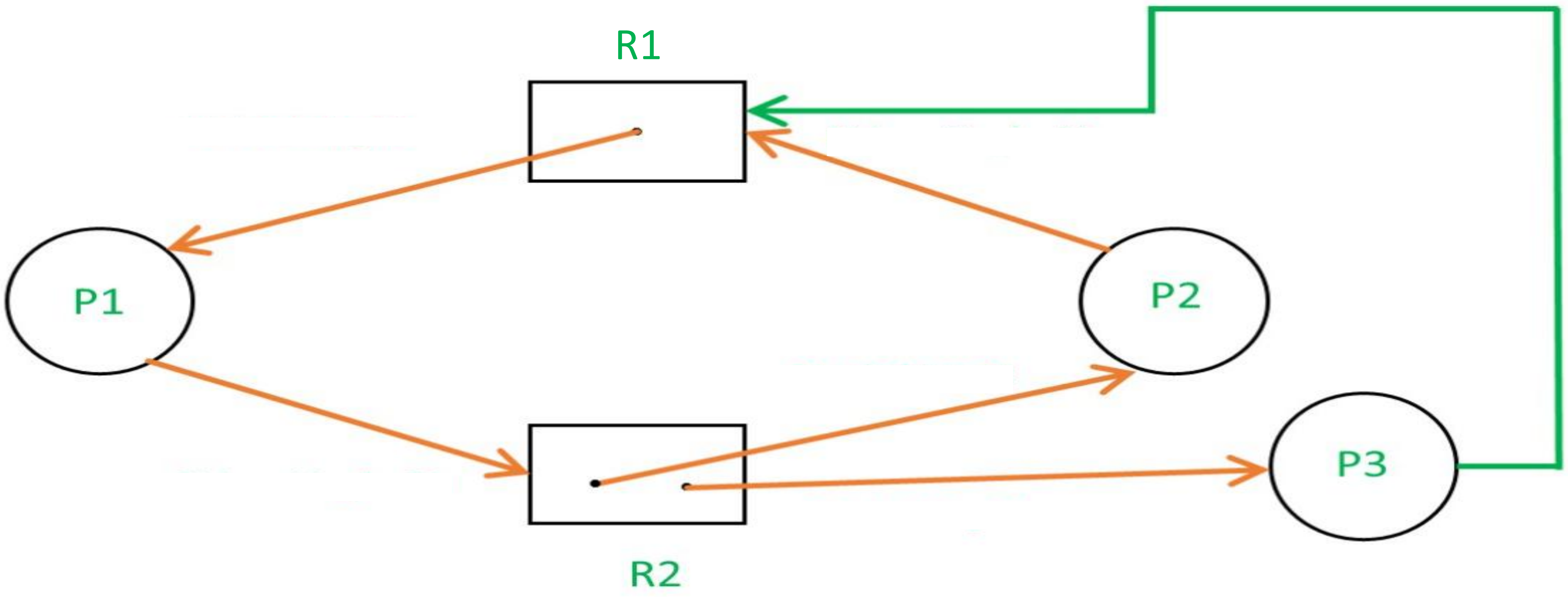
New Available = 1 1 (Now easily we can satisfy the requirement of process P2)
P2 0 1

New Available = 1 2

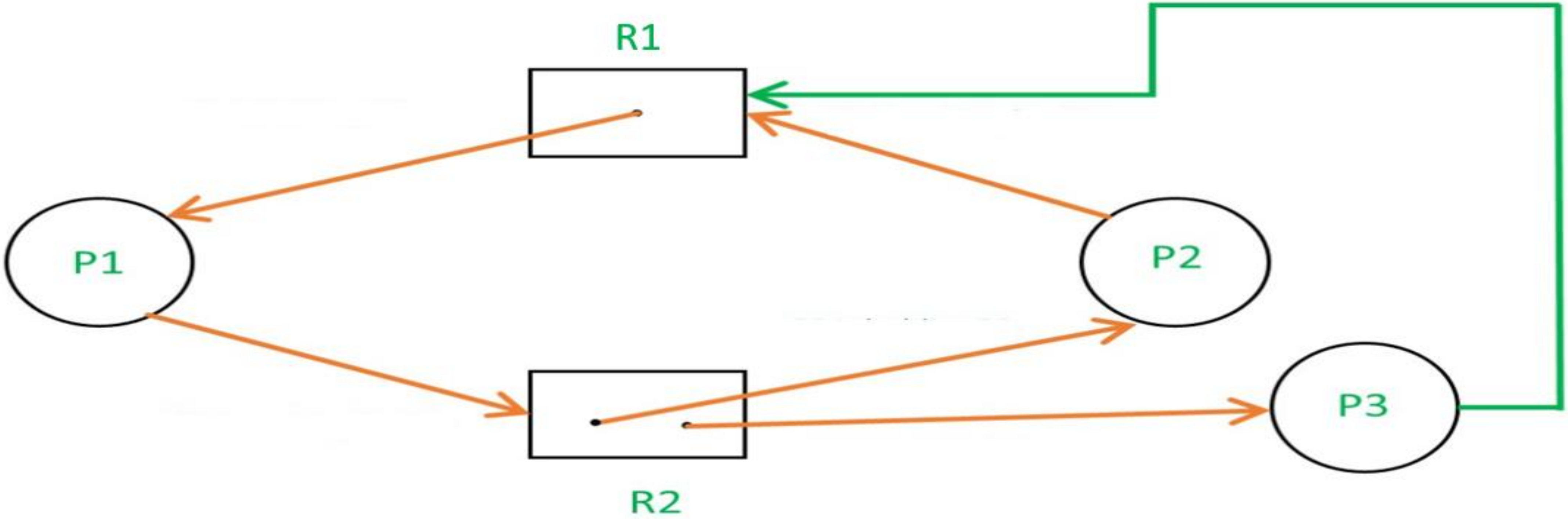
No Deadlock

- There is no deadlock in this RAG. Even though there is a cycle, still there is no deadlock. Therefore, in multi-instance resource cycle is not sufficient condition for deadlock.

Example 4 (Multi Instances RAG)



Does it contains Deadlock?

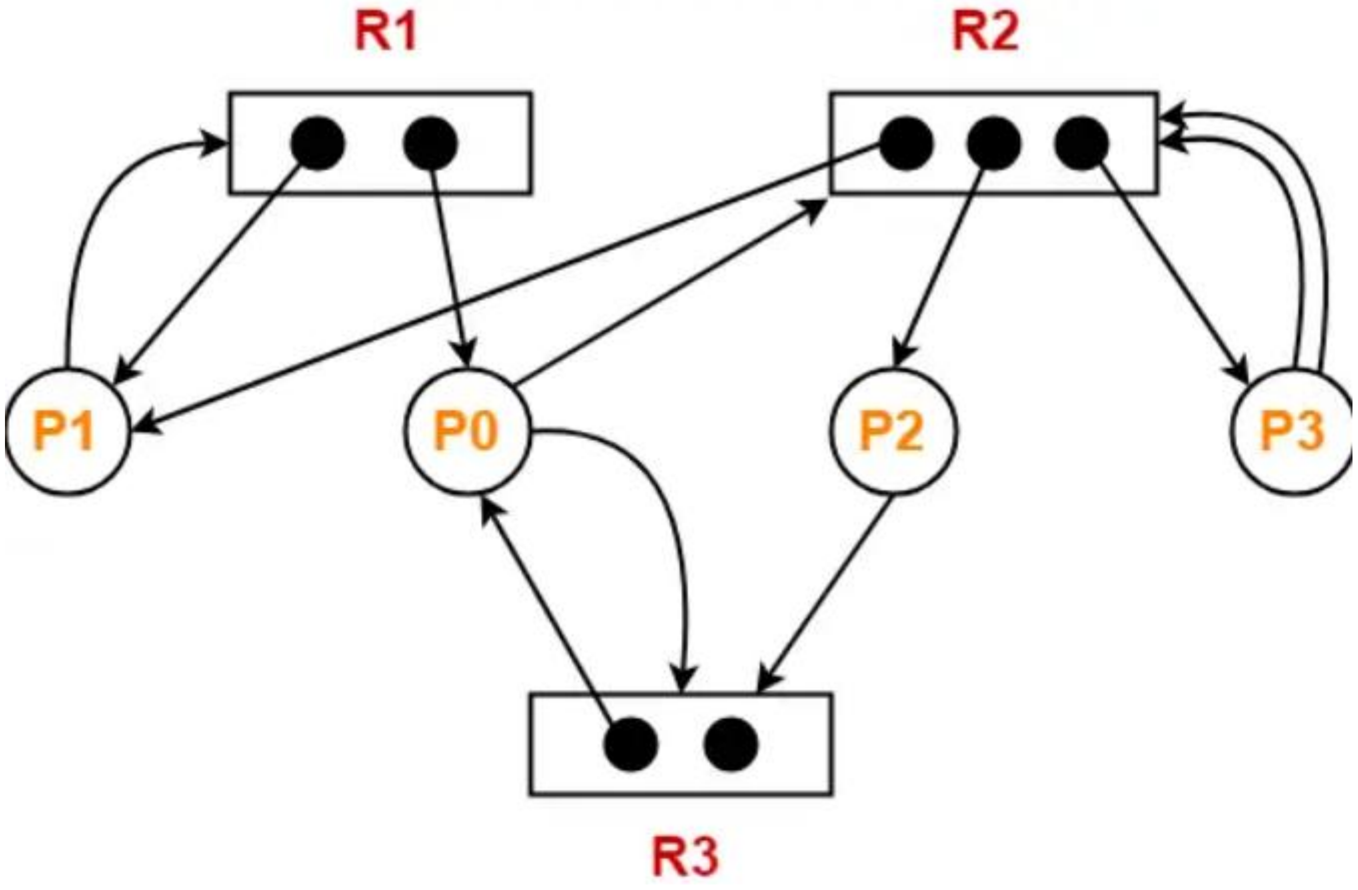


Process	Allocation		Request	
	Resource		Resource	
	R1	R2	R1	R2
P1	1	0	0	1
P2	0	1	1	0
P3	0	1	1	0

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

- So, the Available resource is = (0, 0), but requirement are (0, 1), (1, 0) and (1, 0). So, you can't fulfill any one requirement. Therefore, it is in deadlock. Therefore, every cycle in a multi-instance resource type graph is not a deadlock. If there has to be a deadlock, there has to be a cycle. So, **in case of RAG with multi-instance resource type, the cycle is a necessary condition for deadlock but not sufficient.**

Example 5 (Multi Instances RAG)



Find if the system is in a deadlock state otherwise find a safe sequence.?

	Allocation			Need		
	R1	R2	R3	R1	R2	R3
Process P0	1	0	1	0	1	1
Process P1	1	1	0	1	0	0
Process P2	0	1	0	0	0	1
Process P3	0	1	0	0	2	0

Available = [R1 R2 R3] = [0 0 1]

Available

= [0 0 1] + [0 1 0]

= [0 1 1]

- Process P2 is allocated the requested resources.
- It completes its execution and then free up the instances of resources held by it.

Available

= [0 1 1] + [1 0 1]

= [1 1 2]

- Process P0 is allocated the requested resources.
- It completes its execution and then free up the instances of resources held by it.

Available

= [1 1 2] + [1 1 0]

= [2 2 2]

- Process P1 is allocated the requested resources.
- It completes its execution and then free up the instances of resources held by it.

Available

= [2 2 2] + [0 1 0]

= [2 3 2]

- Process P3 is allocated the requested resources.
- It completes its execution and then free up the instances of resources held by it.

Thus,

- There exists a safe sequence P2, P0, P1, P3 in which all the processes can be executed.
- So, the system is in a safe state.

References

1. Abraham Silberschatz, Peter Baer Galvin, Greg Gagne, “Operating System Concepts,” Eleventh Edition (Willey).

2. Andrew S. Tanenbaum, “Modern Operating Systems”, Fourth Edition (Pearson Publications), 2014.

3. https://www.geeksforgeeks.org/

4. https://www.javatpoint.com/

5. https://www.tutorialspoint.com/

6. https://www.nesoacademy.org/

7. https://www.baeldung.com/

8. https://www.educative.io/

9. https://prepinsta.com