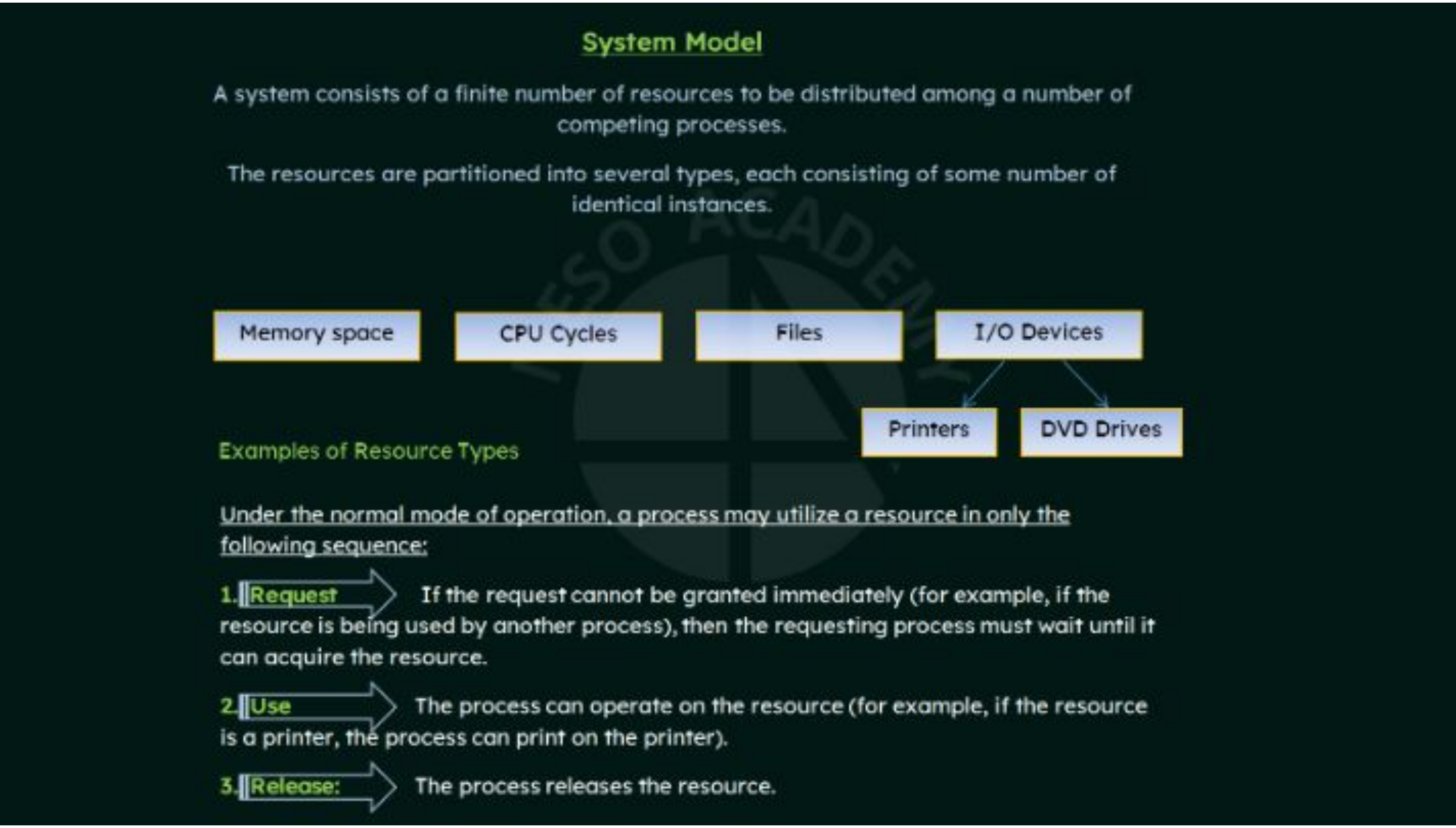
Deadlocks

Deadlocks

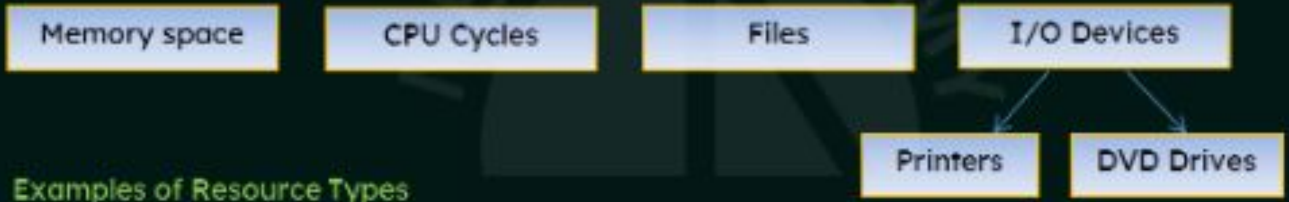
- In a multiprogramming environment, several processes may compete for a finite number of resources.
- A process requests resources; and if the resources are not available at that time, the process enters a waiting state.
- Sometimes, a waiting process is never again able to change state, because the resources it has requested are held by other waiting processes.
- **This situation is called a deadlock.**



System Model

A system consists of a finite number of resources to be distributed among a number of competing processes.

The resources are partitioned into several types, each consisting of some number of identical instances.



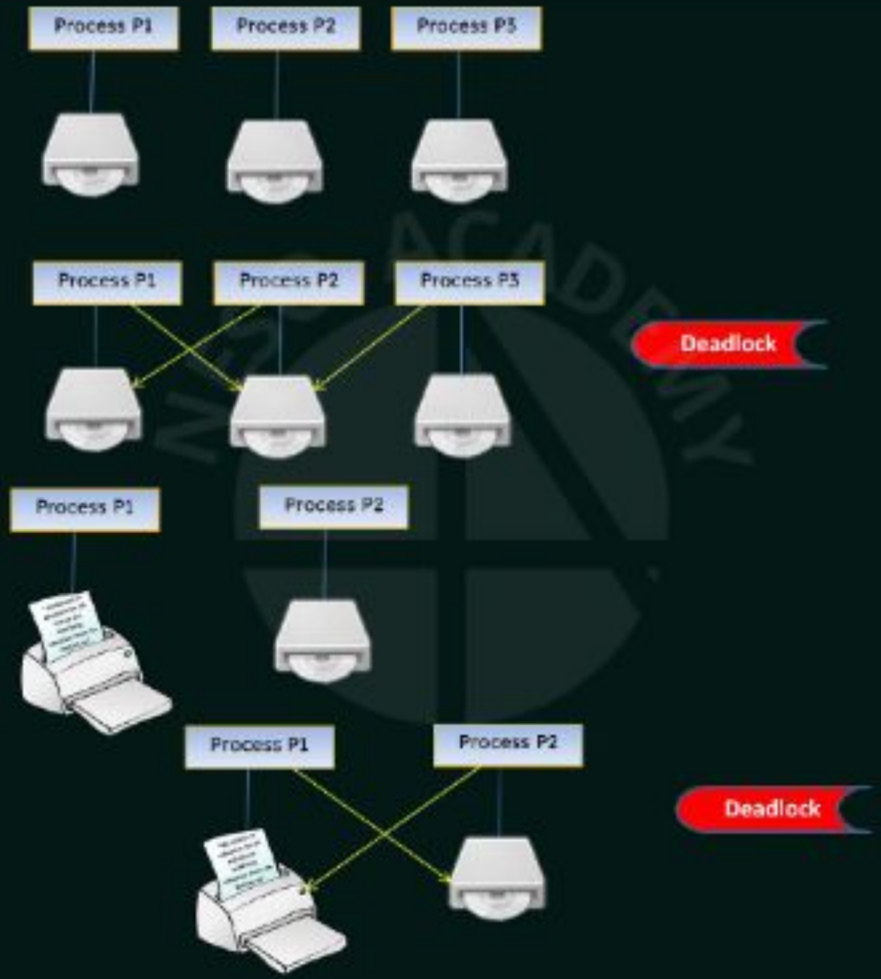
Examples of Resource Types

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

1. **Request** ⇨ If the request cannot be granted immediately (for example, if the resource is being used by another process), then the requesting process must wait until it can acquire the resource.

2. **Use** ⇨ The process can operate on the resource (for example, if the resource is a printer, the process can print on the printer).

3. **Release:** ⇨ The process releases the resource.


Process P1
Process P2
Process P3
Process P1
Process P2
Process P3
Deadlock
Process P1
Process P2
Process P1
Process P2
Deadlock

Deadlock Characterization

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

Features that characterize deadlocks - Necessary Conditions

A deadlock situation can arise if the following four conditions hold simultaneously in a system:

1. **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.

2. **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.

3. 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.

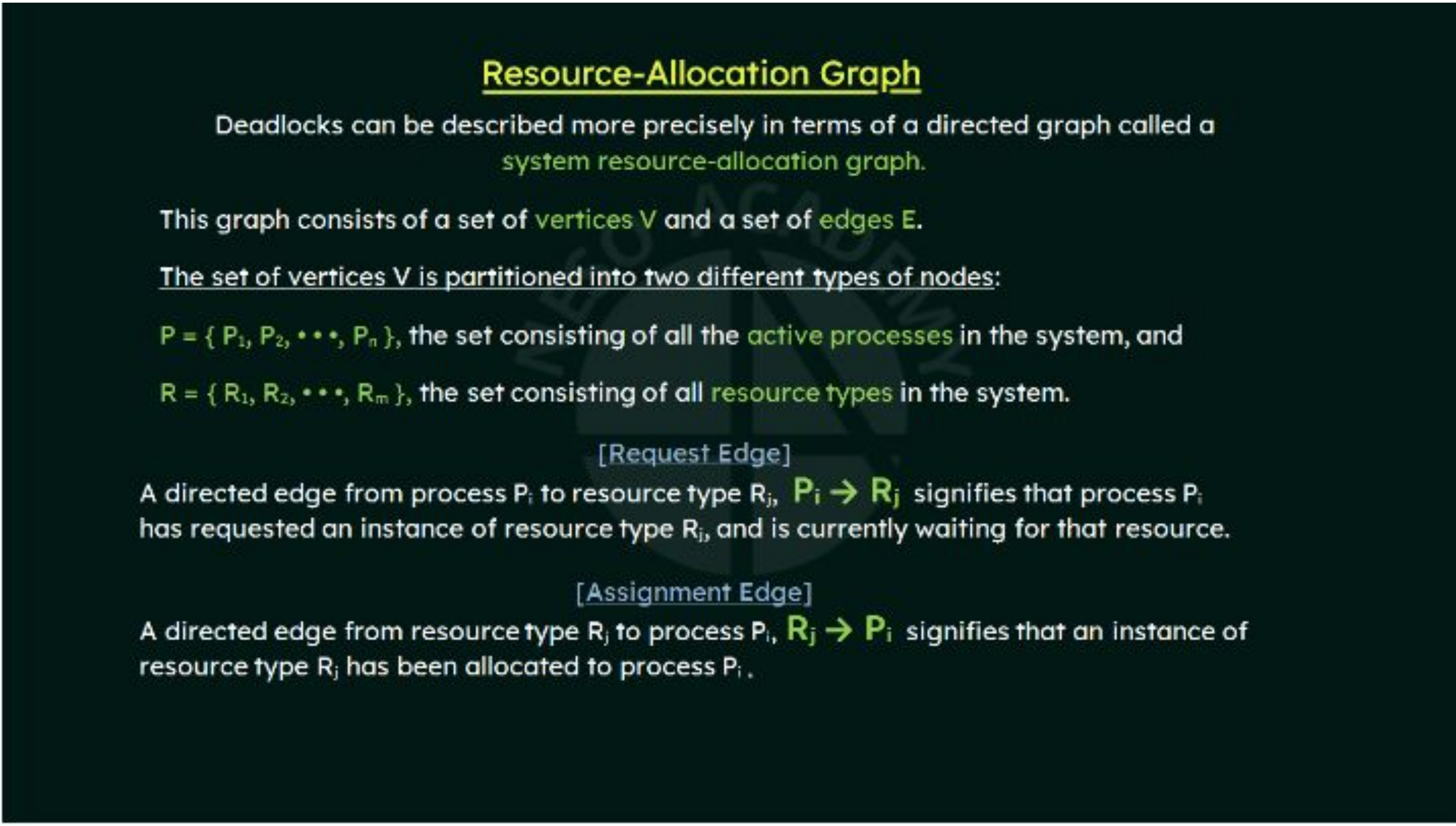
4. Circular wait

A set { $P_0, P_1, \ldots, P_n$ } of waiting processes must exist such that P_0 is waiting for a resource held by P_1, P_1 is waiting for a resource held by P_2, • • •, P_{n-1} is waiting for a resource held by P_n

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

We emphasize that all four conditions must hold for a deadlock to occur.

The circular-wait condition implies the hold-and-wait condition, so the four conditions are not completely independent.



Resource-Allocation Graph

Deadlocks can be described more precisely in terms of a directed graph called a system resource-allocation graph.

This graph consists of a set of vertices V and a set of edges E.

The set of vertices V is partitioned into two different types of nodes:

$P = \{ P_1, P_2, \cdots, P_n \}$, the set consisting of all the active processes in the system, and

$R = \{ R_1, R_2, \cdots, R_m \}$, the set consisting of all resource types in the system.

[Request Edge]

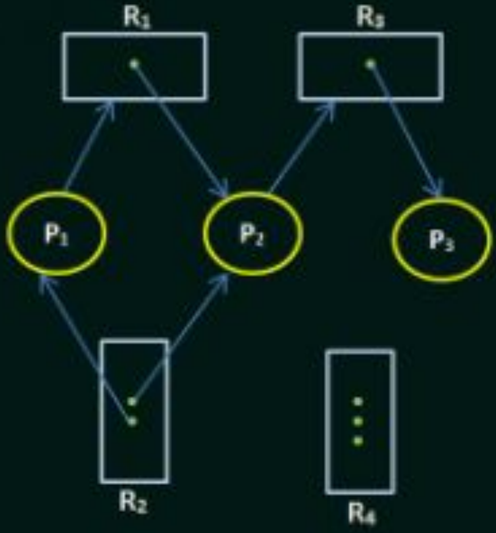
A directed edge from process P_i to resource type R_j, $P_i \rightarrow R_j$ signifies that process P_i has requested an instance of resource type R_j, and is currently waiting for that resource.

[Assignment Edge]

A directed edge from resource type R_j to process P_i, $R_j \rightarrow P_i$ signifies that an instance of resource type R_j has been allocated to process P_i .

Processes are represented using circles

Resources are denoted using rectangles. Since a resource type R_j may have more than one instance, we represent each such instance as a dot within the rectangle.



The sets P, R, and £:

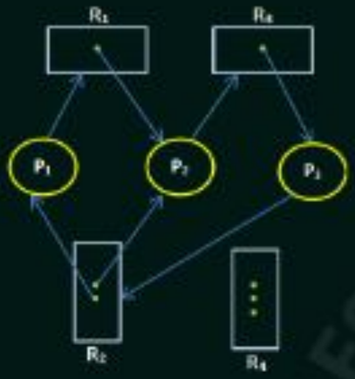
- P = { P1, P2, P3 }
- R = { R1, R2, R3, R4 }
- £ = {P1→R1, P2→R3, R1→P2, R2→P2, R2→P1, R3→P3 }

Resource instances:

- One instance of resource type R1
- Two instances of resource type R2
- One instance of resource type R3
- Three instances of resource type R4

If the graph contains no cycles, then no process in the system is deadlocked.

If the graph does contain a cycle, then a deadlock **may** exist.

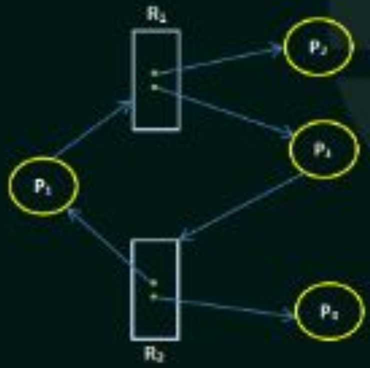


At this point, two minimal cycles exist in the system:

P1 → R1 → P2 → R3 → P3 → R2 → P1

P2 → R3 → P3 → R2 → P2

Processes P1, P2, and P3 are deadlocked.



In this example also we have a cycle:

P1 → R1 → P3 → R2 → P1

However, there is no deadlock.

Observe that process P4 may release its instance of resource type R2.
That resource can then be allocated to P3, breaking the cycle.

If a resource-allocation graph does not have a cycle, then the system is not in a deadlocked state.

If there is a cycle, then the system may or may not be in a deadlocked state.

Methods for Handling Deadlocks

We can deal with the deadlock problem in one of three ways:

1. We can use a protocol to prevent or avoid deadlocks, ensuring that the system will never enter a deadlock state.
2. We can allow the system to enter a deadlock state, detect it, and recover.
3. We can ignore the problem altogether and pretend that deadlocks never occur in the system.

To ensure that deadlocks never occur, the system can use either a deadlock prevention or a deadlock-avoidance scheme

Provides a set of methods for ensuring that at least one of the necessary conditions cannot hold:

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

Requires that the operating system be given in advance additional information concerning which resources a process will request and use during its lifetime.

With this additional knowledge, it can decide for each request whether or not the process should wait.

If a system does not employ either a deadlock-prevention or a deadlock avoidance algorithm, then a deadlock situation may arise.

In this environment, the system can provide an algorithm that examines the state of the system to determine whether a deadlock has occurred and an algorithm to recover from the deadlock (if a deadlock has indeed occurred).

If a system neither ensures that a deadlock will never occur nor provides a mechanism for deadlock detection and recovery, then we may arrive at a situation where the system is in a deadlocked state yet has no way of recognizing what has happened.

In this case, the undetected deadlock will result in deterioration of the system's performance, because resources are being held by processes that cannot run and because more and more processes, as they make requests for resources, will enter a deadlocked state.

Eventually, the system will stop functioning and will need to be restarted manually.

Deadlock Prevention

For a deadlock to occur, each of the four necessary conditions must hold:

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

By ensuring that at least one of these conditions cannot hold, we can prevent the occurrence of a deadlock.

1. Mutual Exclusion

The mutual-exclusion condition must hold for nonsharable resources.

Example: A printer cannot be simultaneously shared by several processes.

Sharable resources, in contrast, do not require mutually exclusive access and thus cannot be involved in a deadlock.

Example: Read-only files. If several processes attempt to open a read-only file at the same time, they can be granted simultaneous access to the file. A process never needs to wait for a sharable resource.

In general, however, we cannot prevent deadlocks by denying the mutual-exclusion condition, because some resources are intrinsically nonsharable.

2. Hold and Wait

To ensure that the hold-and-wait condition never occurs in the system, we must guarantee that, whenever a process requests a resource, it does not hold any other resources.

One protocol that can be used requires each process to request and be allocated all its resources before it begins execution.

An alternative protocol allows a process to request resources only when it has none. A process may request some resources and use them. Before it can request any additional resources, however, it must release all the resources that it is currently allocated.

Both these protocols have two main disadvantages:

1. Resource utilization may be low

2. Starvation is possible

3. No Preemption

To ensure that this condition does not hold, we can use the following protocol:

If a process is holding some resources and requests another resource that cannot be immediately allocated to it (that is, the process must wait), then all resources currently being held are preempted.

Alternatively, if a process requests some resources, we first check whether they are available. If they are, we allocate them. If they are not, we check whether they are allocated to some other process that is waiting for additional resources. If so, we preempt the desired resources from the waiting process and allocate them to the requesting process.

This protocol is often applied to resources whose state can be easily saved and restored later, such as CPU registers and memory space. It cannot generally be applied to such resources as printers and tape drives.

4. Circular Wait

A way to ensure that this condition never holds is to impose a total ordering of all resource types and to require that each process requests resources in an increasing order of enumeration.

Example: Suppose Process P1 is allocated Resource R5. Now if P1 requests for Resources R4 and R3 (which are lesser than R5), such requests will not be granted. Only request for resource greater than R5 will be granted.

Developing an ordering, or hierarchy, in itself does not prevent deadlock. It is up to application developers to write programs that follow the ordering.

Deadlock Avoidance

Banker's Algorithm

An algorithm that can be used for Deadlock Avoidance in resource allocation systems with multiple instances of each resource type.

It is given the name Banker's Algorithm because the algorithm could be used in a banking system to ensure that the bank never allocated its available cash in such a way that it could no longer satisfy the needs of all its customers.

Data Structures used to implement banker's algorithm:

Let 'n' be the number of processes in the system and 'm' be the number of resource types.

Available:

- A 1-D array of size **'m'** indicating the number of available resources of each type.
- Available [j] = k means there are **'k'** instances of resource type $\mathbf{R_j}$

Max:

- A 2-D array of size **'n x m'** that specifies the maximum demand of each process in a system.
- Max[i, j] = k means process $\mathbf{P_i}$ may request at most **'k'** instances of resource type $\mathbf{R_j}$.

Allocation:

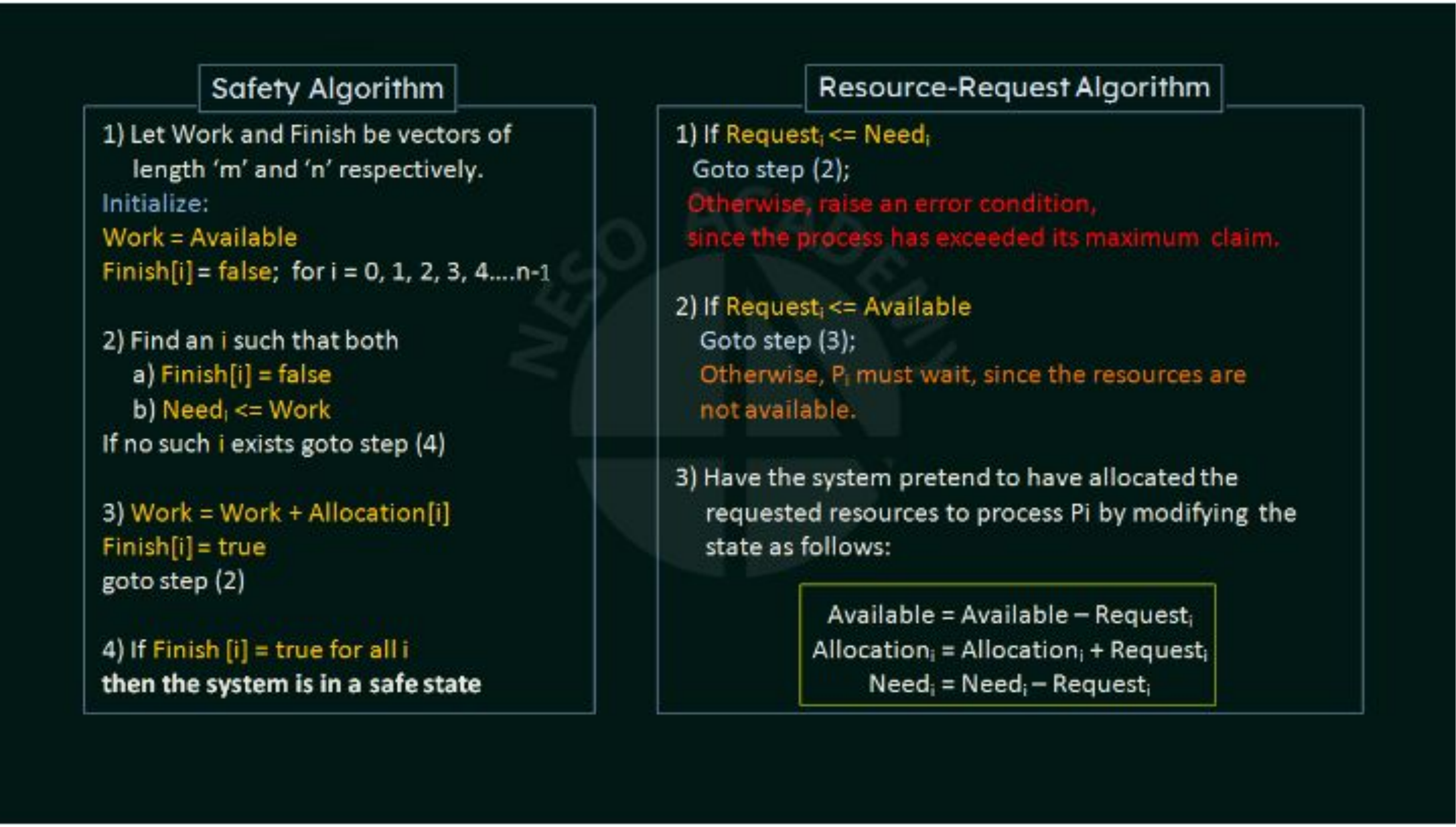
- It is a 2-D array of size **'n x m'** that specifies the number of resources of each type currently allocated to each process.
- Allocation[i, j] = k means process **P_i** is currently allocated **'k'** instances of resource type **R_j**.

Need:

- It is a 2-D array of size **'n x m'** that indicates the remaining resource needs of each process.
- Need [i, j] = k means process **P_i** currently need **'k'** instances of resource type **R_j** for its execution.
- Need [i, j] = Max [i, j] - Allocation [i, j]

Allocation_i specifies the resources currently allocated to process P_i.

Need_i specifies the additional resources that process P_i may still request to complete its task.



Safety Algorithm

```
1) Let Work and Finish be vectors of
   length 'm' and 'n' respectively.
Initialize:
Work = Available
Finish[i] = false;  for i = 0, 1, 2, 3, 4....n-1

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
```

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 Pi by modifying the
   state as follows:

      Available = Available – $Request_i$
      $Allocation_i$ = $Allocation_i$ + $Request_i$
      $Need_i$ = $Need_i$ – $Request_i$
```

Deadlock Avoidance

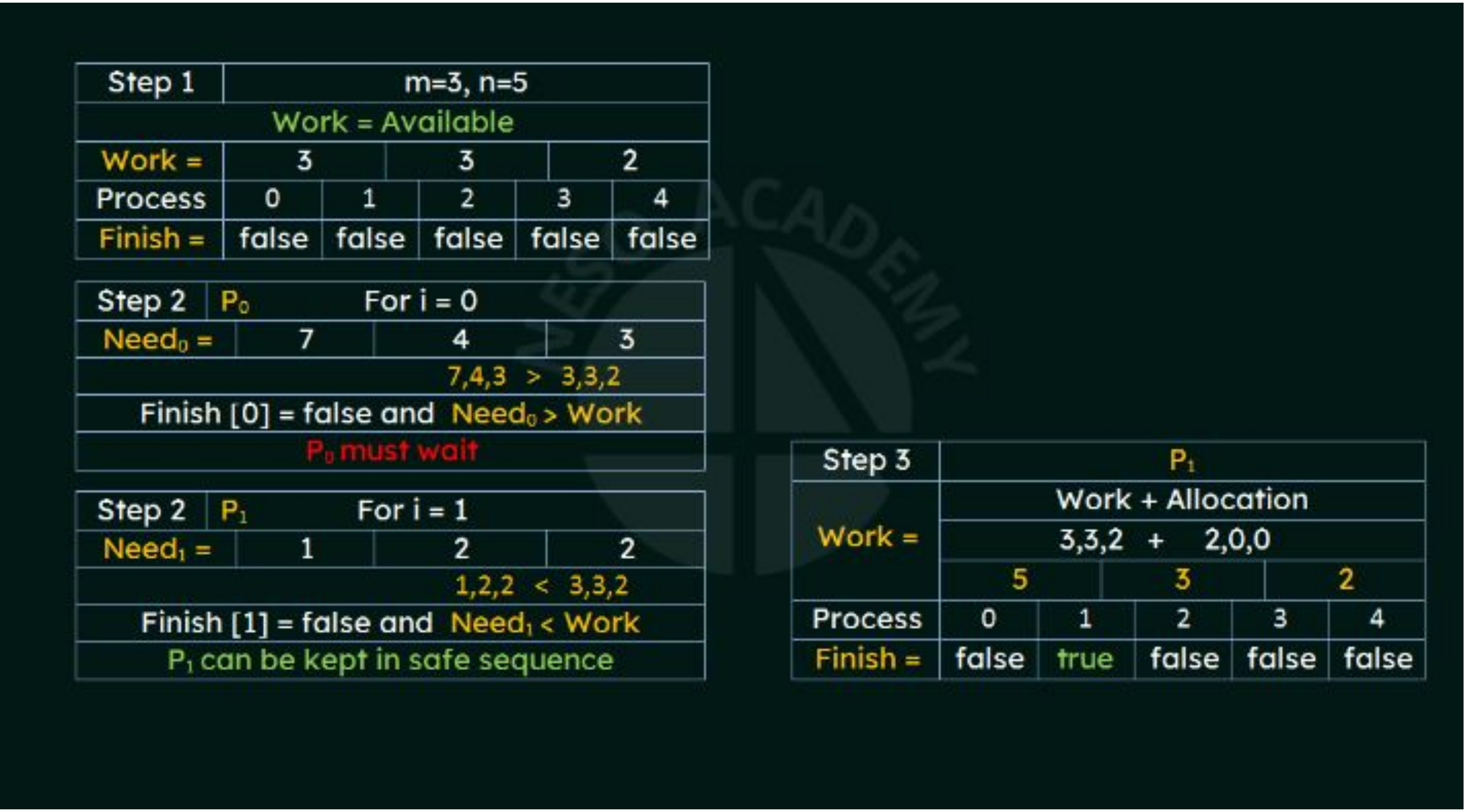
Example of Safety Algorithm in Banker's Algorithm

Consider a system with five processes P_0, P_1, P_2, P_3, P_4 and three resources of type A, B, C. Resource type A has 10 instances, B has 5 instances and C has 7 instances.

Suppose at time T_0 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			

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

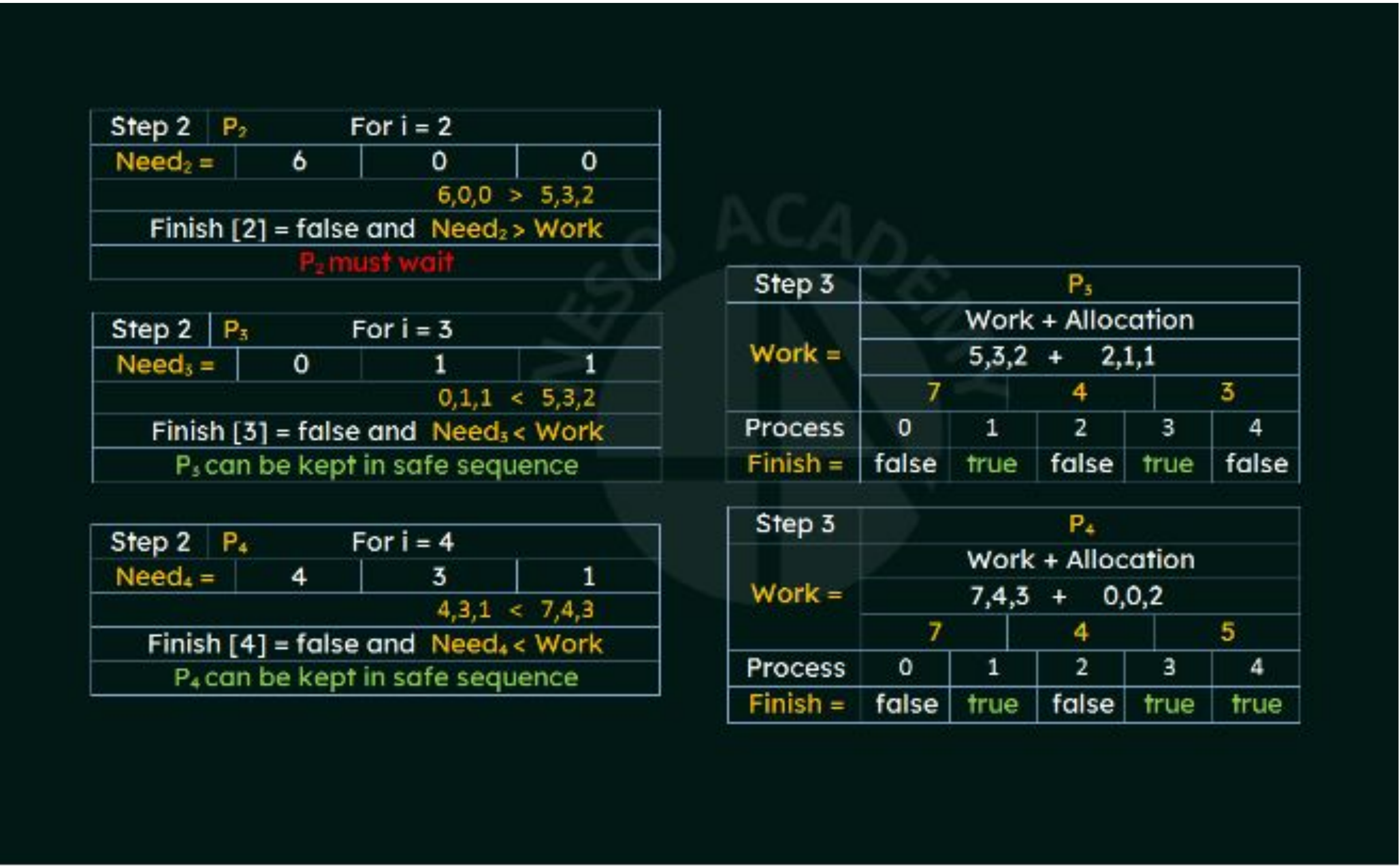


Step 1	m=3, n=5				
Work = Available					
Work =	3		3		2
Process	0	1	2	3	4
Finish =	false	false	false	false	false

Step 2	P_0	For i = 0	
$Need_0$ =	7	4	3
		7,4,3 > 3,3,2	
Finish [0] = false and $Need_0$ > Work			
P_0 must wait			

Step 2	P_1	For i = 1	
$Need_1$ =	1	2	2
		1,2,2 < 3,3,2	
Finish [1] = false and $Need_1$ < Work			
P_1 can be kept in safe sequence			

Step 3	P_1				
Work =	Work + Allocation				
	3,3,2 + 2,0,0				
	5		3		2
Process	0	1	2	3	4
Finish =	false	true	false	false	false



Step 2	P_2	For i = 2	
$Need_2$ =	6	0	0
		6,0,0 > 5,3,2	
Finish [2] = false and $Need_2$ > Work			
P_2 must wait			

Step 2	P_3	For i = 3	
$Need_3$ =	0	1	1
		0,1,1 < 5,3,2	
Finish [3] = false and $Need_3$ < Work			
P_3 can be kept in safe sequence			

Step 2	P_4	For i = 4	
$Need_4$ =	4	3	1
		4,3,1 < 7,4,3	
Finish [4] = false and $Need_4$ < Work			
P_4 can be kept in safe sequence			

Step 3	P_3				
Work =	Work + Allocation				
	5,3,2 + 2,1,1				
	7		4		3
Process	0	1	2	3	4
Finish =	false	true	false	true	false

Step 3	P_4				
Work =	Work + Allocation				
	7,4,3 + 0,0,2				
	7		4		5
Process	0	1	2	3	4
Finish =	false	true	false	true	true

Step 2	P_0	For i = 0	
$Need_4$ =	7	4	3
		7,4,3 < 7,4,5	
Finish [0] = false and $Need_0$ < Work			
P_0 can be kept in safe sequence			

Step 3	P_0				
Work =	Work + Allocation				
	7,4,5 + 0,1,0				
	7		5		5
Process	0	1	2	3	4
Finish =	true	true	false	true	true

Step 2	P_2	For i = 2	
$Need_4$ =	6	0	0
		6,0,0 < 7,5,5	
Finish [0] = false and $Need_2$ < Work			
P_2 can be kept in safe sequence			

Step 3	P_2				
Work =	Work + Allocation				
	7,5,5 + 3,0,2				
	10		5		7
Process	0	1	2	3	4
Finish =	true	true	true	true	true

Now,
Finish [i] = true for all i. So the system is in SAFE STATE

The Safe Sequence is < P_1, P_3, P_4, P_0, P_2 >

Deadlock Avoidance

Example of Resource-Request Algorithm in Banker's Algorithm

Suppose now that process **P_1** requests one additional instance of resource type A and two instances of resource type C, so Request$_1$ = (1,0,2).

Can this request be immediately granted ?

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

Step 1	Check condition
	1, 0, 2 <= 1, 2, 2
	$Request_1$ <= $Need_1$
	Condition Satisfied

Step 2	Check condition
	1, 0, 2 <= 3, 3, 2
	$Request_1$ <= Available
	Condition Satisfied

Step 3

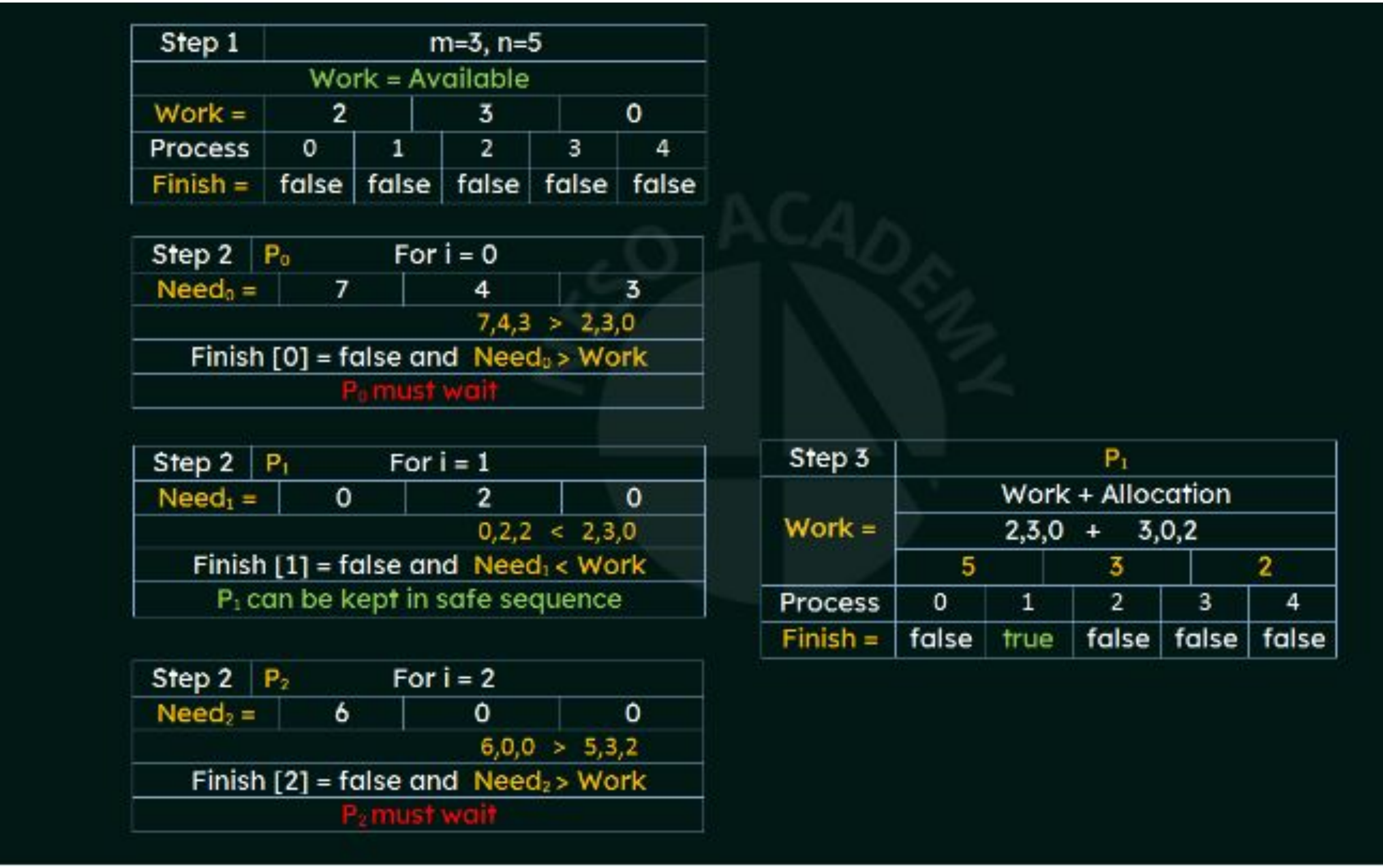
Available = Available - $Request_1$

$Allocation_1$ = $Allocation_1$ + $Request_1$

$Need_1$ = $Need_1$ - $Request_1$

Process	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			

We must determine whether this new system state is safe. To do so, we execute our safety algorithm again and try to find if this system state is safe and if it is safe what is the safe sequence.



Step 1	m=3, n=5				
Work = Available					
Work =	2		3		0
Process	0	1	2	3	4
Finish =	false	false	false	false	false

Step 2	P_0 For i = 0	
$Need_0$ = 7	4	3
7,4,3 > 2,3,0		
Finish [0] = false and $Need_0$ > Work		
P_0 must wait		

Step 2	P_1 For i = 1	
$Need_1$ = 0	2	0
0,2,2 < 2,3,0		
Finish [1] = false and $Need_1$ < Work		
P_1 can be kept in safe sequence		

Step 2	P_2 For i = 2	
$Need_2$ = 6	0	0
6,0,0 > 5,3,2		
Finish [2] = false and $Need_2$ > Work		
P_2 must wait		

Step 3	P_1				
Work =	Work + Allocation				
	2,3,0 + 3,0,2				
	5		3		2
Process	0	1	2	3	4
Finish =	false	true	false	false	false

Step 2	P_3	For i = 3	
$Need_3$ =	0	1	1
		0,1,1 < 5,3,2	
Finish [3] = false and $Need_3 < Work$			
P_3 can be kept in safe sequence			

Step 2	P_4	For i = 4	
$Need_4$ =	4	3	1
		4,3,1 < 7,4,3	
Finish [4] = false and $Need_4 < Work$			
P_4 can be kept in safe sequence			

Step 2	P_0	For i = 0	
$Need_4$ =	7	4	3
		7,4,3 < 7,4,5	
Finish [0] = false and $Need_0 < Work$			
P_0 can be kept in safe sequence			

Step 2	P_2	For i = 2	
$Need_4$ =	6	0	0
		6,0,0 < 7,5,5	
Finish [0] = false and $Need_2 < Work$			
P_2 can be kept in safe sequence			

Step 3	P_3				
Work =	Work + Allocation				
	5,3,2 + 2,1,1				
	7	4	3		
Process	0	1	2	3	4
Finish =	false	true	false	true	false

Step 3	P_4				
Work =	Work + Allocation				
	7,4,3 + 0,0,2				
	7	4	5		
Process	0	1	2	3	4
Finish =	false	true	false	true	true

Step 3	P_0				
Work =	Work + Allocation				
	7,4,5 + 0,1,0				
	7	5	5		
Process	0	1	2	3	4
Finish =	true	true	false	true	true

Step 3	P_2				
Work =	Work + Allocation				
	7,5,5 + 3,0,2				
	10	5	7		
Process	0	1	2	3	4
Finish =	true	true	true	true	true

Now,

Finish [i] = true for all i. So the system is in SAFE STATE

The Safe Sequence is < P_1, P_3, P_4, P_0, P_2 >

So, by applying the Resource-Request Algorithm and by checking the state of the system using the Safety Algorithm, we find that granting the request of Process P_1 still keeps the system in a safe state and hence will not lead to a deadlock.

Hence, we can immediately grant the request of process P_1.

Deadlock Detection

Single Instance of Each Resource Type

In systems where each resources have only a single instance, we can use Wait-For Graph which is an algorithm for deadlock detection and is a variant of the Resource Allocation Graph which we have studied earlier.

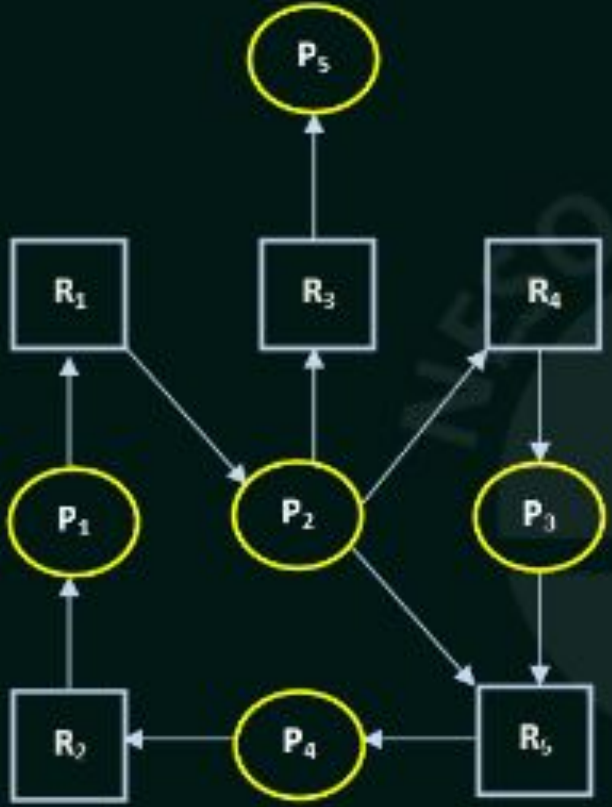
We obtain this graph from the resource-allocation graph by removing the resource nodes and collapsing the appropriate edges.

In a Wait-For Graph:

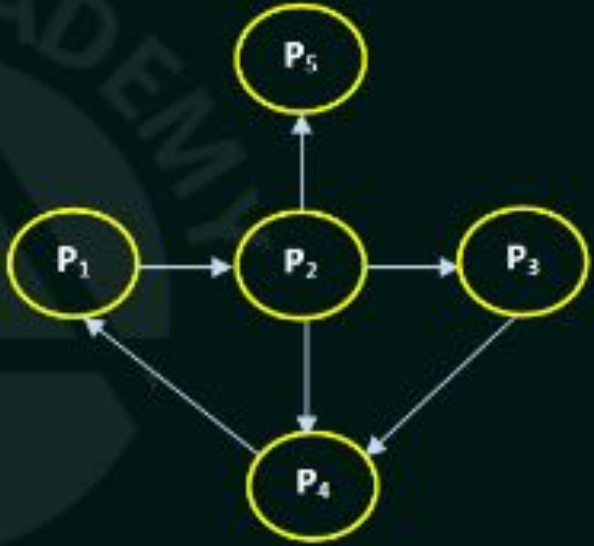
An edge from P_i to P_j indicates that P_i is waiting for P_j to release a resource that is needed by P_i.

In a Resource-Allocation Graph, the same would be denoted as $P_i \rightarrow R_k$ and $R_k \rightarrow P_j$

Deadlock exists in the system
if and only if the wait-for graph contains a cycle



Resource-allocation graph



Corresponding wait-for graph

Deadlock Detection

Multiple Instances of each Resource Type

This algorithm employs several time-varying data structures that are similar to those used in the Banker's Algorithm.

Available: A vector of length m indicates the number of available resources of each type.

Allocation: An n x m matrix defines the number of resources of each type currently allocated to each process.

Request: An n x m matrix indicates the current request of each process.

If Request[i][j] = k, then process P_i, is requesting k more instances of resource type R_j.

The Algorithm:

```
1) Let Work and Finish be vectors of
   length ‘m’ and ‘n’ respectively.
Initialize:
Work = Available
If Allocation ≠ 0 then
Finish[i] = false;  else Finish[i] = true   for i = 0, 1, 2, 3, 4….n-1

2) Find an i such that both
   a) Finish[i] = false
   b) $Request_i$ <= Work
If no such i exists goto step (4)

3) Work = Work + Allocation[i]
Finish[i] = true
goto step (2)

4) If Finish [i] = false for some i, 0<i<n
then the system is in a deadlocked state. $P_i$ is deadlocked
```

Deadlock Detection

Multiple Instances of each Resource Type (Example)

Consider a system with five processes P_0, P_1, P_2, P_3, P_4 and three resources of type A, B, C. Resource type A has 7 instances, B has 2 instances and C has 6 instances.

Suppose at time T_0 following snapshot of the system has been taken:

Process	Allocation			Request			Available		
	A	B	C	A	B	C	A	B	C
P_0	0	1	0	0	0	0	0	0	0
P_1	2	0	0	2	0	2			
P_2	3	0	3	0	0	0			
P_3	2	1	1	1	0	0			
P_4	0	0	2	0	0	2			

Currently Available		
A	B	C
0	0	0
0	1	0
3	1	3
5	2	4
7	2	4
7	2	6

Safe Sequence we find if we excecute our algorithm - $< P_0, P_2, P_3, P_1, P_4 >$

Suppose now that process P_2 makes one additional request for an instance of type C. The Request matrix is modified as follows:

Process	Allocation			Request			Available		
	A	B	C	A	B	C	A	B	C
P_0	0	1	0	0	0	0	0	0	0
P_1	2	0	0	2	0	2			
P_2	3	0	3	0	0	1			
P_3	2	1	1	1	0	0			
P_4	0	0	2	0	0	2			

Currently Available		
A	B	C
0	0	0
0	1	0

Now the System is DEADLOCKED.

Although we can reclaim the resources held by process P_0, the number of available resources is not sufficient to fulfill the requests of the other processes.

Recovery From Deadlock

Process Termination

What to do when a deadlock detection algorithm determines that a deadlock exists?

Possibility 1: Inform the operator that a deadlock has occurred and let the operator deal with the deadlock manually.

Possibility 2: Let the system recover from the deadlock automatically.

How to break from the deadlock?

Option 1: Abort one or more processes to break the circular wait.

Option 2: Preempt some resources from one or more of the deadlocked processes.

Process Termination

To eliminate deadlocks by aborting a process, we use one of two methods:

- Abort all deadlocked processes:

  Will break the deadlock cycle, but at great expense. The deadlocked processes may have computed for a long time, and the results of these partial computations must be discarded and probably will have to be recomputed later.

- Abort one process at a time until the deadlock cycle is eliminated:

  Incurs considerable overhead, since, after each process is aborted, a deadlock-detection algorithm must be invoked to determine whether any processes are still deadlocked.

Aborting a process may not be easy.

Some examples:

- If the process was in the midst of updating a file, terminating it will leave that file in an incorrect state.
- If the process was in the midst of printing data on a printer, the system must reset the printer to a correct state before printing the next job.

If the partial termination method is used:

- We must determine which deadlocked process (or processes) should be terminated.
- This determination is a policy decision, similar to CPU-scheduling decisions.

We should abort those processes whose termination will incur the minimum cost.

Some factors that affect which process(es) are chosen to be aborted:

1. What the priority of the process is.

2. How long the process has computed and how much longer the process will compute before completing its designated task.

3. How many and what type of resources the process has used (for example, whether the resources are simple to preempt)

4. How many more resources the process needs in order to complete.

5. How many processes will need to be terminated.

6. Whether the process is interactive or batch.

Recovery From Deadlock

Resource Preemption

Here we successively preempt some resources from processes and give these resources to other processes until the deadlock cycle is broken.

If preemption is required to deal with deadlocks, then three issues need to be addressed:

1. **Selecting a victim**
2. **Rollback**
3. **Starvation**

Selecting a victim

- Which resources and which processes are to be preempted?

- Determine the order of preemption to minimize cost.

$\downarrow$

Parameters such as the number of resources a deadlocked process is holding and the amount of time the process has thus far consumed during its execution.

<u>Rollback</u>

- What should be done with a process if we prempt its resources ?
- It cannot continue it normal execution as its missing some or all of its needed resources.
- We must roll back the process to some safe state and restart it from that state.

But what is its safe state ?

➢ As it is difficult to determine what a safe state is, the simplest solution is a <u>total rollback</u>.

$\downarrow$

Abort the process and then restart it.

Starvation

- How do we ensure that starvation will not occur?
- How can we guarantee that resources will not always be preempted from the same process?
- If victim selection is based primarily on cost factors, it may happen that the same process is always picked as a victim. As a result, this process never completes its designated task.
  - ➢ We must ensure that a process can be picked as a victim only a (small) finite number of times.

SOLUTION

↓

Include the number of rollbacks in the cost factor.

Deadlocks
Solved Problems (Part-1)

GATE 2014

An operating system uses the Banker's algorithm for deadlock avoidance when managing the allocation of three resource types X, Y, and Z to three processes P0, P1, and P2. The table given below presents the current system state. Here, the Allocation matrix shows the current number of resources of each type allocated to each process and the Max matrix shows the maximum number of resources of each type required by each process during its execution.

	Allocation			Max		
	X	Y	Z	X	Y	Z
P0	0	0	1	8	4	3
P1	3	2	0	6	2	0
P2	2	1	1	3	3	3

There are 3 units of type X, 2 units of type Y and 2 units of type Z still available. The system is currently in a safe state.
Consider the following independent requests for additional resources in the current state:

REQ1: P0 requests 0 units of X, 0 units of Y and 2 units of Z
REQ2: P1 requests 2 units of X, 0 units of Y and 0 units of Z

Which one of the following is TRUE?

(A) Only REQ1 can be permitted.
(B) Only REQ2 can be permitted.
(C) Both REQ1 and REQ2 can be permitted.
(D) Neither REQ1 nor REQ2 can be permitted.

	Allocation			Max			Need			Available		
	X	Y	Z	X	Y	Z	X	Y	Z	X	Y	Z
P0	0	0	1	8	4	3	8	4	2	3	2	2
P1	3	2	0	6	2	0	3	0	0			
P2	2	1	1	3	3	3	1	2	2			

REQ1: P0 requests 0 units of X, 0 units of Y and 2 units of Z

Assume that the request was granted

	Allocation			Max			Need			Available		
	X	Y	Z	X	Y	Z	X	Y	Z	X	Y	Z
P0	0	0	3	8	4	3	8	4	0	3	2	0
P1	3	2	0	6	2	0	3	0	0			
P2	2	1	1	3	3	3	1	2	2			

Now, P1 can be allowed to execute

	Allocation			Max			Need			Available		
	X	Y	Z	X	Y	Z	X	Y	Z	X	Y	Z
P0	0	0	3	8	4	3	8	4	0	3	2	0
P1	3	2	0	6	2	0	3	0	0			
P2	2	1	1	3	3	3	1	2	2			

After P1 executes

Available		
X	Y	Z
6	4	0

With this available count- neither P0 nor P2 can be allowed to execute.

Hence, REQ1 cannot be permitted.

	Allocation			Max			Need			Available		
	X	Y	Z	X	Y	Z	X	Y	Z	X	Y	Z
P0	0	0	1	8	4	3	8	4	2	3	2	2
P1	3	2	0	6	2	0	3	0	0			
P2	2	1	1	3	3	3	1	2	2			

REQ2: P1 requests 2 units of X, 0 units of Y and 0 units of Z

Assume that the request was granted

	Allocation			Max			Need			Available		
	X	Y	Z	X	Y	Z	X	Y	Z	X	Y	Z
P0	0	0	1	8	4	3	8	4	2	1	2	2
P1	5	2	0	6	2	0	1	0	0			
P2	2	1	1	3	3	3	1	2	2			

P1 or P2 can be allowed to execute

	Allocation			Max			Need			Available		
	X	Y	Z	X	Y	Z	X	Y	Z	X	Y	Z
P0	0	0	1	8	4	3	8	4	2	1	2	2
P1	5	2	0	6	2	0	1	0	0			
P2	2	1	1	3	3	3	1	2	2			

If P1 executes

Available		
X	Y	Z
6	4	2

With this available count P2 can be executed

After P2 executes

Available		
X	Y	Z
8	5	3

With this available count P0 can be executed

All processes executed. Safe Sequence <P1, P2, P0>

Hence, REQ2 can be permitted.

Deadlocks
Solved Problems (Part-2)

GATE 2009

Consider a system with 4 types of resources R1 (3 units), R2 (2 units), R3 (3 units), R4 (2 units). A non-preemptive resource allocation policy is used. At any given instance, a request is not entertained if it cannot be completely satisfied. Three processes P1, P2, P3 request the resources as follows if executed independently.

Process P1:
t=0: requests 2 units of R2
t=1: requests 1 unit of R3
t=3: requests 2 units of R1
t=5: releases 1 unit of R2
and 1 unit of R1.
t=7: releases 1 unit of R3
t=8: requests 2 units of R4
t=10: Finishes

Process P2:
t=0: requests 2 units of R3
t=2: requests 1 unit of R4
t=4: requests 1 unit of R1
t=6: releases 1 unit of R3
t=8: Finishes

Process P3:
t=0: requests 1 unit of R4
t=2: requests 2 units of R1
t=5: releases 2 units of R1
t=7: requests 1 unit of R2
t=8: requests 1 unit of R3
t=9: Finishes

Which one of the following statements is TRUE if all three processes run concurrently starting at time t=0?
(A) All processes will finish without any deadlock
(B) Only P1 and P2 will be in deadlock.
(C) Only P1 and P3 will be in a deadlock.
(D) All three processes will be in deadlock.

Process P1:

t=0: requests 2 units of R2
t=1: requests 1 unit of R3
t=3: requests 2 units of R1
t=5: releases 1 unit of R2
and 1 unit of R1.
t=7: releases 1 unit of R3
t=8: requests 2 units of R4
t=10: Finishes

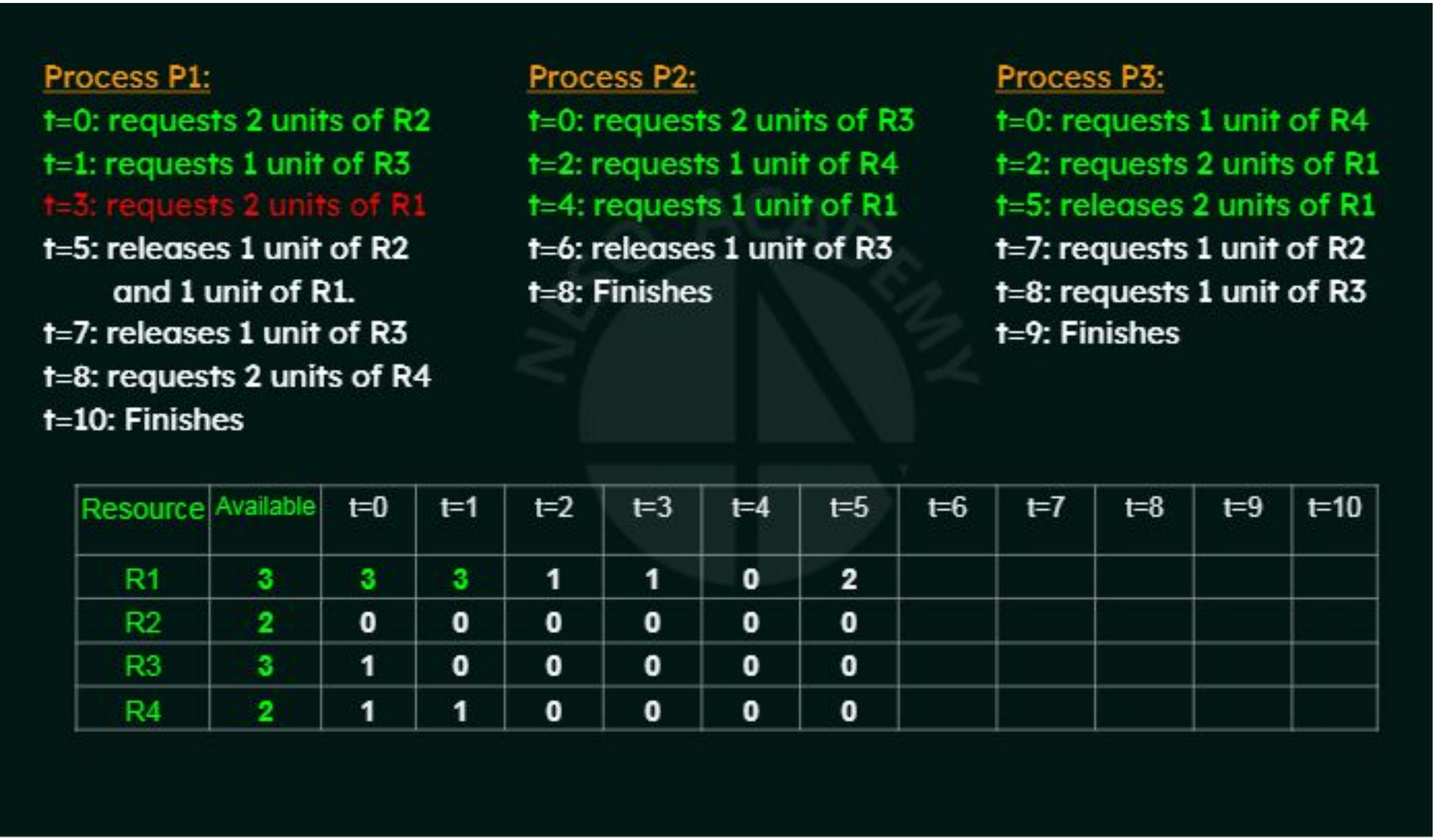
Process P2:

t=0: requests 2 units of R3
t=2: requests 1 unit of R4
t=4: requests 1 unit of R1
t=6: releases 1 unit of R3
t=8: Finishes

Process P3:

t=0: requests 1 unit of R4
t=2: requests 2 units of R1
t=5: releases 2 units of R1
t=7: requests 1 unit of R2
t=8: requests 1 unit of R3
t=9: Finishes

Resource	Available
R1	3
R2	2
R3	3
R4	2



Process P1:

t=0: requests 2 units of R2
t=1: requests 1 unit of R3
t=3: requests 2 units of R1
t=5: releases 1 unit of R2
and 1 unit of R1.
t=7: releases 1 unit of R3
t=8: requests 2 units of R4
t=10: Finishes

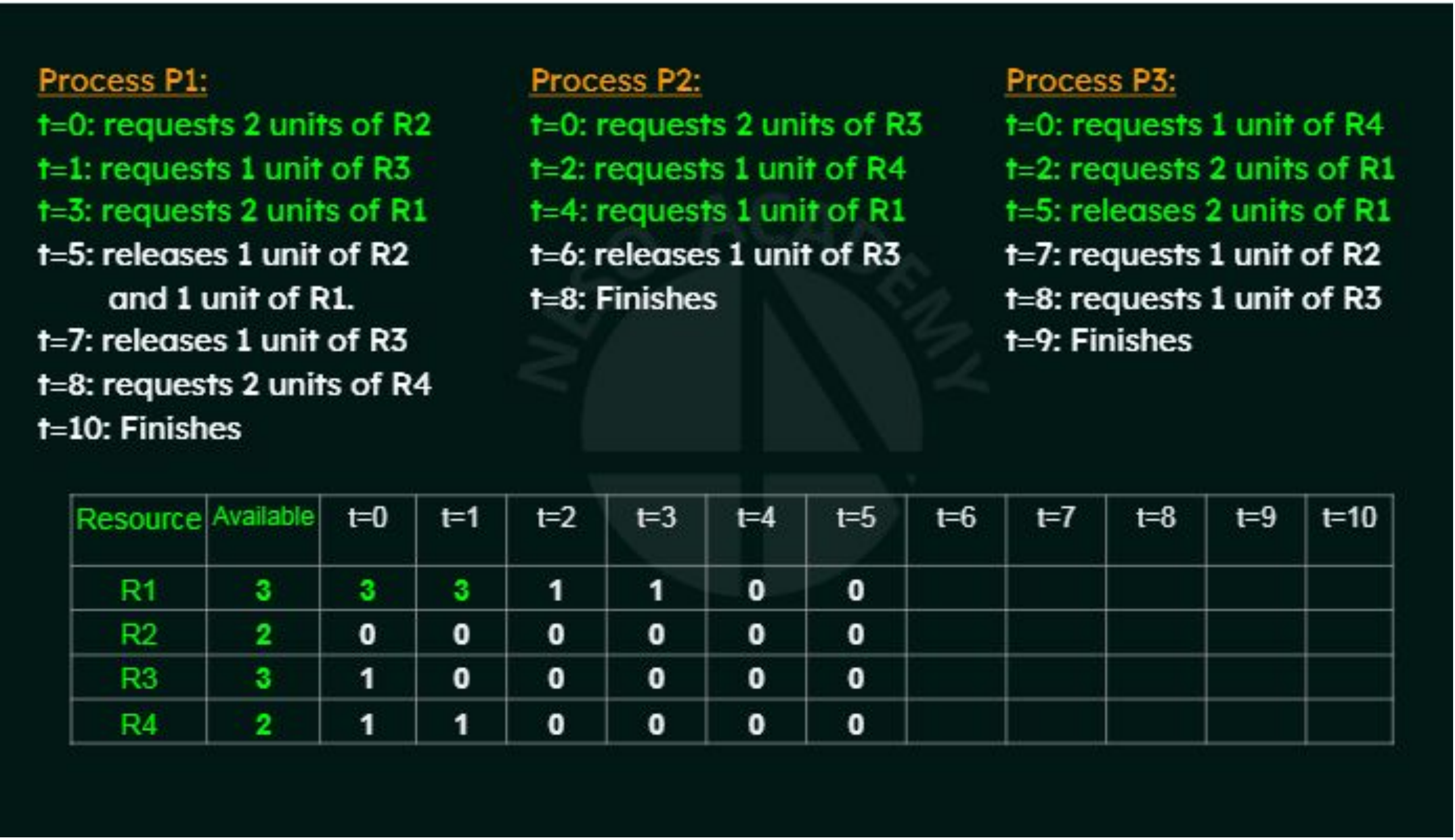
Process P2:

t=0: requests 2 units of R3
t=2: requests 1 unit of R4
t=4: requests 1 unit of R1
t=6: releases 1 unit of R3
t=8: Finishes

Process P3:

t=0: requests 1 unit of R4
t=2: requests 2 units of R1
t=5: releases 2 units of R1
t=7: requests 1 unit of R2
t=8: requests 1 unit of R3
t=9: Finishes

Resource	Available	t=0	t=1	t=2	t=3	t=4	t=5	t=6	t=7	t=8	t=9	t=10
R1	3	3	3	1	1	0	2					
R2	2	0	0	0	0	0	0					
R3	3	1	0	0	0	0	0					
R4	2	1	1	0	0	0	0					



Process P1:

t=0: requests 2 units of R2
t=1: requests 1 unit of R3
t=3: requests 2 units of R1
t=5: releases 1 unit of R2
and 1 unit of R1.
t=7: releases 1 unit of R3
t=8: requests 2 units of R4
t=10: Finishes

Process P2:

t=0: requests 2 units of R3
t=2: requests 1 unit of R4
t=4: requests 1 unit of R1
t=6: releases 1 unit of R3
t=8: Finishes

Process P3:

t=0: requests 1 unit of R4
t=2: requests 2 units of R1
t=5: releases 2 units of R1
t=7: requests 1 unit of R2
t=8: requests 1 unit of R3
t=9: Finishes

Resource	Available	t=0	t=1	t=2	t=3	t=4	t=5	t=6	t=7	t=8	t=9	t=10
R1	3	3	3	1	1	0	0					
R2	2	0	0	0	0	0	0					
R3	3	1	0	0	0	0	0					
R4	2	1	1	0	0	0	0					

Process P1:

t=0: requests 2 units of R2

t=1: requests 1 unit of R3

t=3: requests 2 units of R1

t=5: releases 1 unit of R2
and 1 unit of R1.

t=7: releases 1 unit of R3

t=8: requests 2 units of R4

t=10: Finishes

Process P2:

t=0: requests 2 units of R3

t=2: requests 1 unit of R4

t=4: requests 1 unit of R1

t=6: releases 1 unit of R3

t=8: Finishes

Process P3:

t=0: requests 1 unit of R4

t=2: requests 2 units of R1

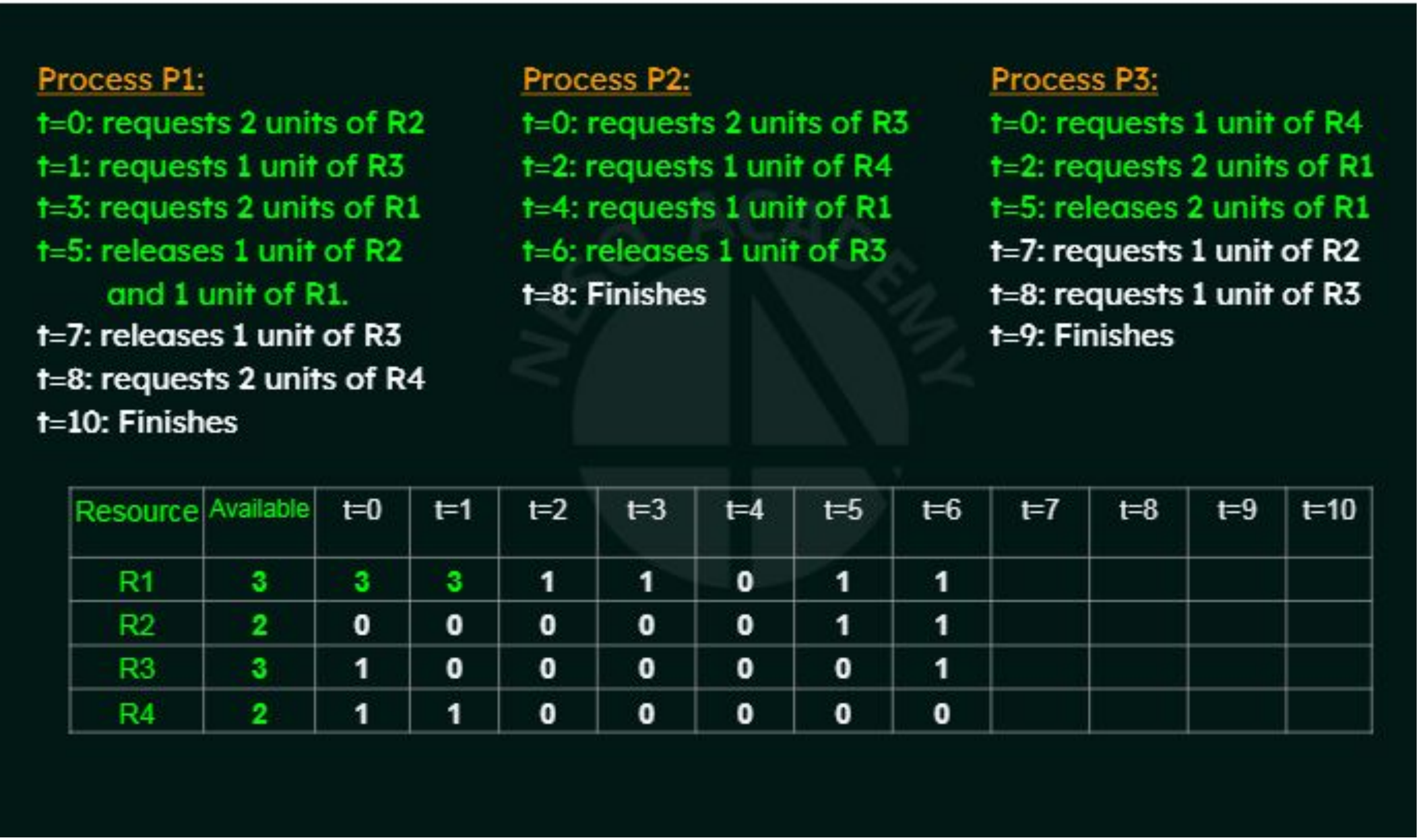
t=5: releases 2 units of R1

t=7: requests 1 unit of R2

t=8: requests 1 unit of R3

t=9: Finishes

Resource	Available	t=0	t=1	t=2	t=3	t=4	t=5	t=6	t=7	t=8	t=9	t=10
R1	3	3	3	1	1	0	1					
R2	2	0	0	0	0	0	1					
R3	3	1	0	0	0	0	0					
R4	2	1	1	0	0	0	0					



Process P1:

t=0: requests 2 units of R2
t=1: requests 1 unit of R3
t=3: requests 2 units of R1
t=5: releases 1 unit of R2
 and 1 unit of R1.
t=7: releases 1 unit of R3
t=8: requests 2 units of R4
t=10: Finishes

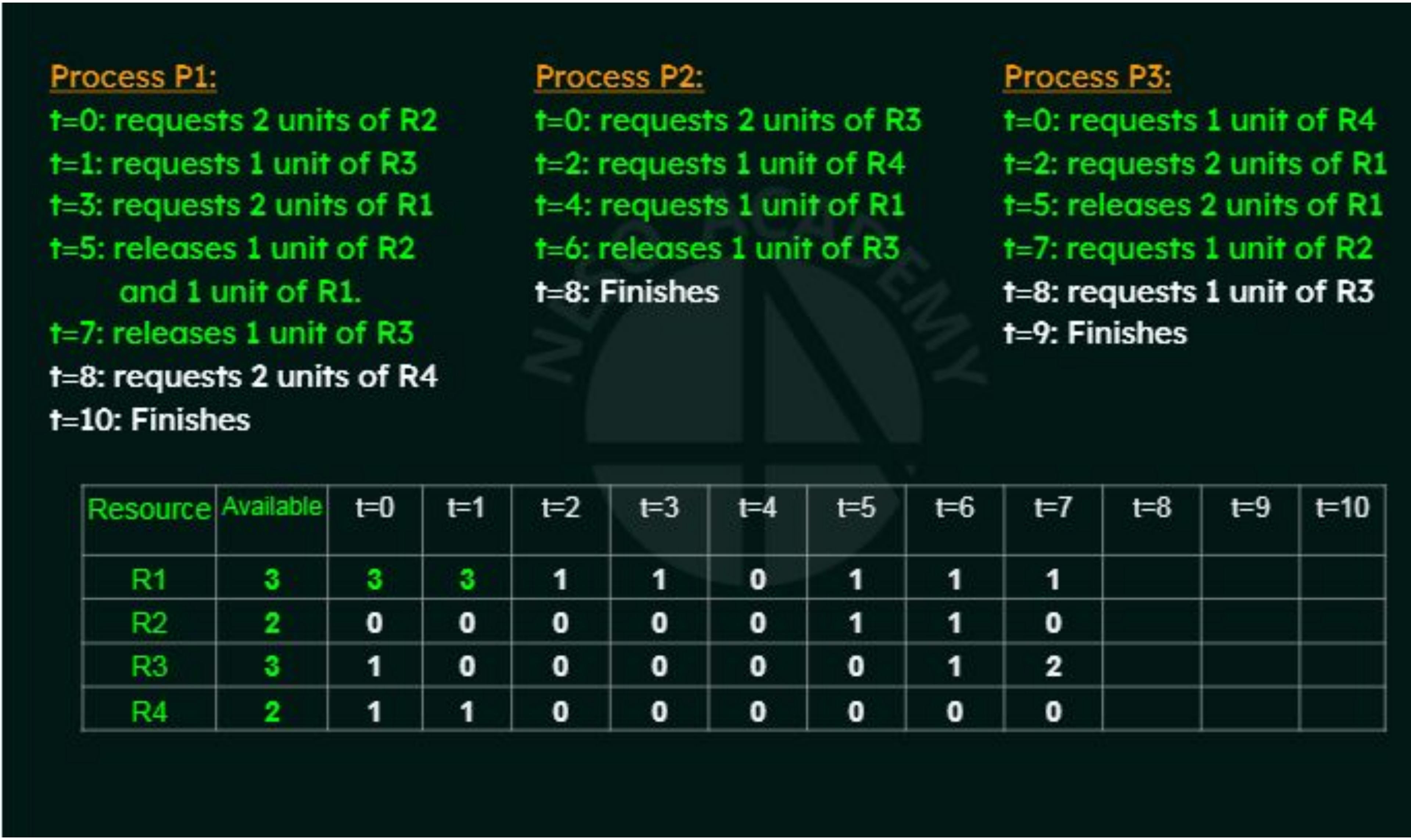
Process P2:

t=0: requests 2 units of R3
t=2: requests 1 unit of R4
t=4: requests 1 unit of R1
t=6: releases 1 unit of R3
t=8: Finishes

Process P3:

t=0: requests 1 unit of R4
t=2: requests 2 units of R1
t=5: releases 2 units of R1
t=7: requests 1 unit of R2
t=8: requests 1 unit of R3
t=9: Finishes

Resource	Available	t=0	t=1	t=2	t=3	t=4	t=5	t=6	t=7	t=8	t=9	t=10
R1	3	3	3	1	1	0	1	1				
R2	2	0	0	0	0	0	1	1				
R3	3	1	0	0	0	0	0	1				
R4	2	1	1	0	0	0	0	0				



Process P1:

t=0: requests 2 units of R2
t=1: requests 1 unit of R3
t=3: requests 2 units of R1
t=5: releases 1 unit of R2
and 1 unit of R1.
t=7: releases 1 unit of R3
t=8: requests 2 units of R4
t=10: Finishes

Process P2:

t=0: requests 2 units of R3
t=2: requests 1 unit of R4
t=4: requests 1 unit of R1
t=6: releases 1 unit of R3
t=8: Finishes

Process P3:

t=0: requests 1 unit of R4
t=2: requests 2 units of R1
t=5: releases 2 units of R1
t=7: requests 1 unit of R2
t=8: requests 1 unit of R3
t=9: Finishes

Resource	Available	t=0	t=1	t=2	t=3	t=4	t=5	t=6	t=7	t=8	t=9	t=10
R1	3	3	3	1	1	0	1	1	1			
R2	2	0	0	0	0	0	1	1	0			
R3	3	1	0	0	0	0	0	1	2			
R4	2	1	1	0	0	0	0	0	0			

Process P1:

t=0: requests 2 units of R2
t=1: requests 1 unit of R3
t=3: requests 2 units of R1
t=5: releases 1 unit of R2
and 1 unit of R1.
t=7: releases 1 unit of R3
t=8: requests 2 units of R4
t=10: Finishes

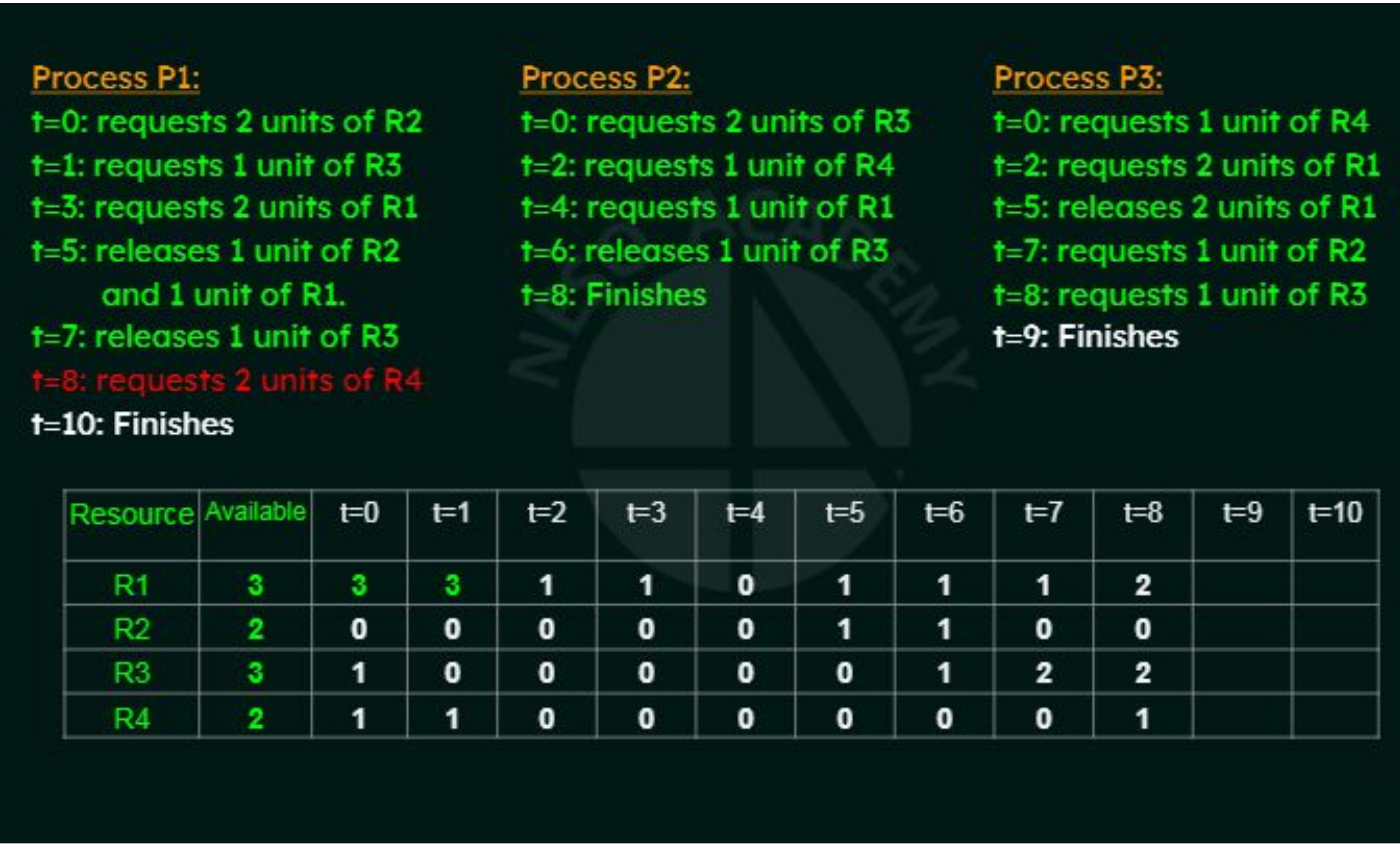
Process P2:

t=0: requests 2 units of R3
t=2: requests 1 unit of R4
t=4: requests 1 unit of R1
t=6: releases 1 unit of R3
t=8: Finishes

Process P3:

t=0: requests 1 unit of R4
t=2: requests 2 units of R1
t=5: releases 2 units of R1
t=7: requests 1 unit of R2
t=8: requests 1 unit of R3
t=9: Finishes

Resource	Available	t=0	t=1	t=2	t=3	t=4	t=5	t=6	t=7	t=8	t=9	t=10
R1	3	3	3	1	1	0	1	1	1	2		
R2	2	0	0	0	0	0	1	1	0	0		
R3	3	1	0	0	0	0	0	1	2	3		
R4	2	1	1	0	0	0	0	0	0	1		



Process P1:

t=0: requests 2 units of R2
t=1: requests 1 unit of R3
t=3: requests 2 units of R1
t=5: releases 1 unit of R2
and 1 unit of R1.
t=7: releases 1 unit of R3
t=8: requests 2 units of R4
t=10: Finishes

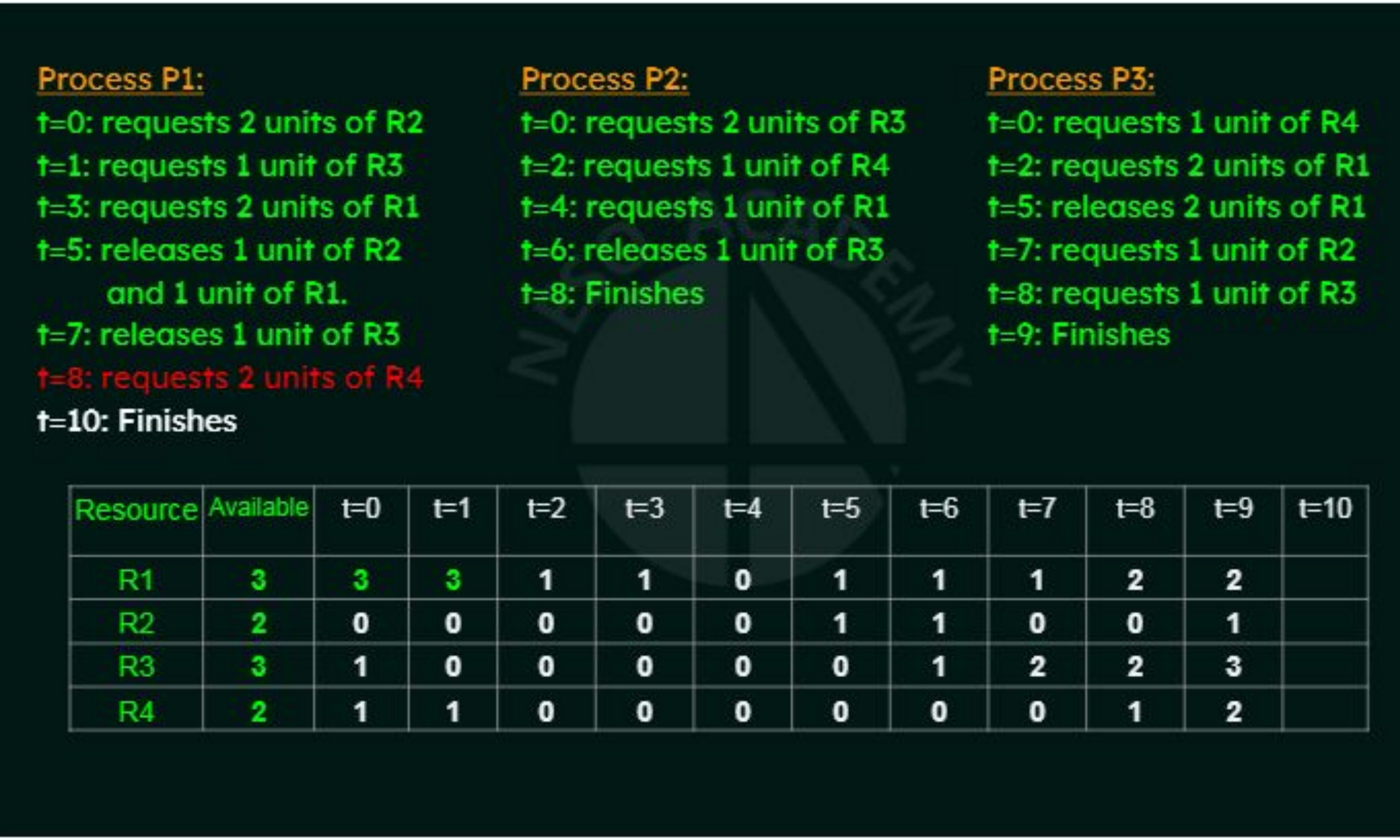
Process P2:

t=0: requests 2 units of R3
t=2: requests 1 unit of R4
t=4: requests 1 unit of R1
t=6: releases 1 unit of R3
t=8: Finishes

Process P3:

t=0: requests 1 unit of R4
t=2: requests 2 units of R1
t=5: releases 2 units of R1
t=7: requests 1 unit of R2
t=8: requests 1 unit of R3
t=9: Finishes

Resource	Available	t=0	t=1	t=2	t=3	t=4	t=5	t=6	t=7	t=8	t=9	t=10
R1	3	3	3	1	1	0	1	1	1	2		
R2	2	0	0	0	0	0	1	1	0	0		
R3	3	1	0	0	0	0	0	1	2	2		
R4	2	1	1	0	0	0	0	0	0	1		



Process P1:

t=0: requests 2 units of R2
t=1: requests 1 unit of R3
t=3: requests 2 units of R1
t=5: releases 1 unit of R2
and 1 unit of R1.
t=7: releases 1 unit of R3
t=8: requests 2 units of R4
t=10: Finishes

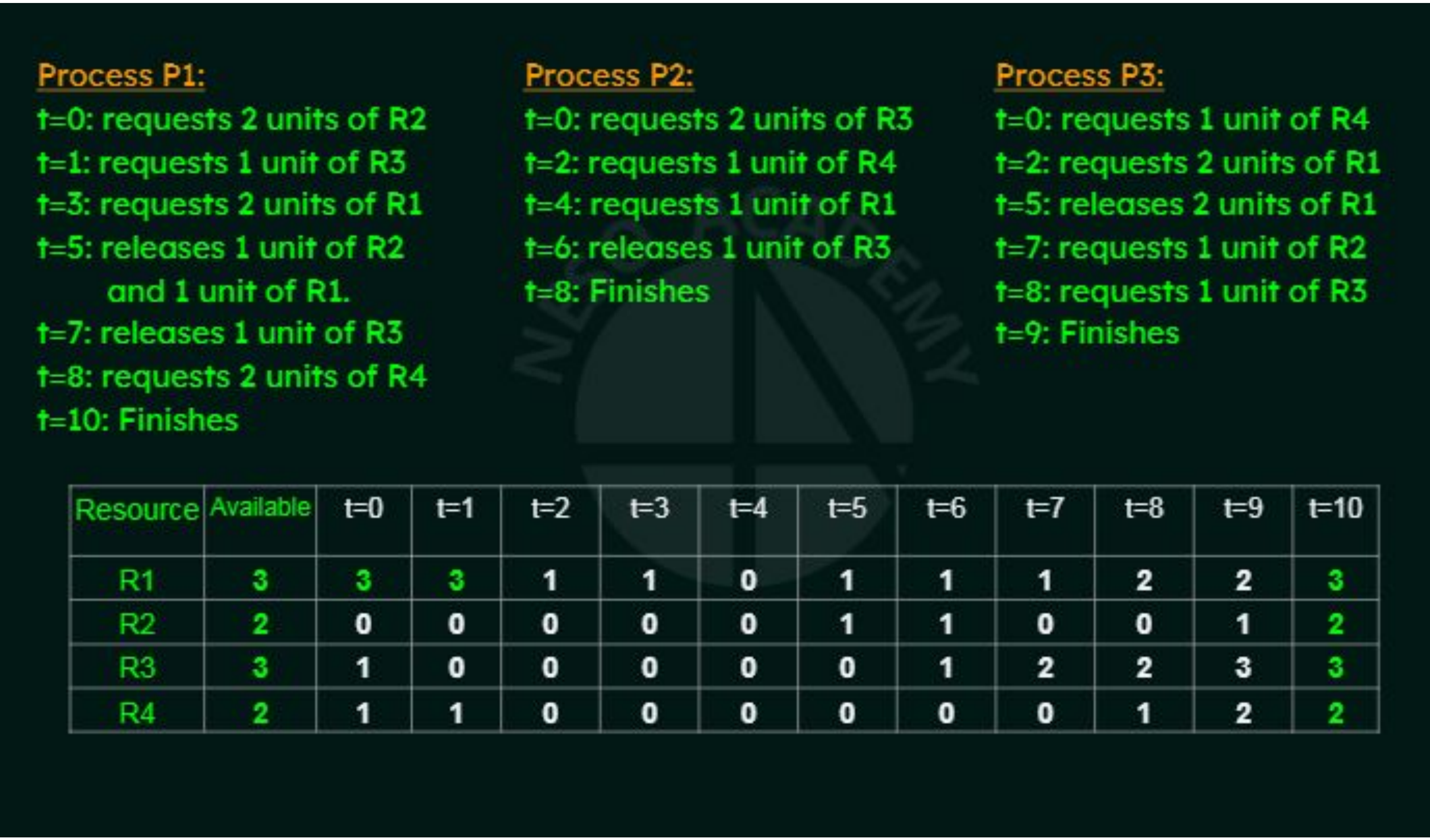
Process P2:

t=0: requests 2 units of R3
t=2: requests 1 unit of R4
t=4: requests 1 unit of R1
t=6: releases 1 unit of R3
t=8: Finishes

Process P3:

t=0: requests 1 unit of R4
t=2: requests 2 units of R1
t=5: releases 2 units of R1
t=7: requests 1 unit of R2
t=8: requests 1 unit of R3
t=9: Finishes

Resource	Available	t=0	t=1	t=2	t=3	t=4	t=5	t=6	t=7	t=8	t=9	t=10
R1	3	3	3	1	1	0	1	1	1	2	2	
R2	2	0	0	0	0	0	1	1	0	0	1	
R3	3	1	0	0	0	0	0	1	2	2	3	
R4	2	1	1	0	0	0	0	0	0	1	2	



Process P1:

t=0: requests 2 units of R2
t=1: requests 1 unit of R3
t=3: requests 2 units of R1
t=5: releases 1 unit of R2
and 1 unit of R1.
t=7: releases 1 unit of R3
t=8: requests 2 units of R4
t=10: Finishes

Process P2:

t=0: requests 2 units of R3
t=2: requests 1 unit of R4
t=4: requests 1 unit of R1
t=6: releases 1 unit of R3
t=8: Finishes

Process P3:

t=0: requests 1 unit of R4
t=2: requests 2 units of R1
t=5: releases 2 units of R1
t=7: requests 1 unit of R2
t=8: requests 1 unit of R3
t=9: Finishes

Resource	Available	t=0	t=1	t=2	t=3	t=4	t=5	t=6	t=7	t=8	t=9	t=10
R1	3	3	3	1	1	0	1	1	1	2	2	3
R2	2	0	0	0	0	0	1	1	0	0	1	2
R3	3	1	0	0	0	0	0	1	2	2	3	3
R4	2	1	1	0	0	0	0	0	0	1	2	2

Deadlocks
Solved Problems (Part-2)

GATE 2009

Consider a system with 4 types of resources R1 (3 units), R2 (2 units), R3 (3 units), R4 (2 units). A non-preemptive resource allocation policy is used. At any given instance, a request is not entertained if it cannot be completely satisfied. Three processes P1, P2, P3 request the resources as follows if executed independently.

Process P1:
t=0: requests 2 units of R2
t=1: requests 1 unit of R3
t=3: requests 2 units of R1
t=5: releases 1 unit of R2
and 1 unit of R1.
t=7: releases 1 unit of R3
t=8: requests 2 units of R4
t=10: Finishes

Process P2:
t=0: requests 2 units of R3
t=2: requests 1 unit of R4
t=4: requests 1 unit of R1
t=6: releases 1 unit of R3
t=8: Finishes

Process P3:
t=0: requests 1 unit of R4
t=2: requests 2 units of R1
t=5: releases 2 units of R1
t=7: requests 1 unit of R2
t=8: requests 1 unit of R3
t=9: Finishes

Which one of the following statements is TRUE if all three processes run concurrently starting at time t=0?
(A) All processes will finish without any deadlock
(B) Only P1 and P2 will be in deadlock.
(C) Only P1 and P3 will be in a deadlock.
(D) All three processes will be in deadlock.

Deadlocks

Solved Problems (Part-3)

GATE 2014

Question 1

A system contains three programs and each requires three tape units for its operation. The minimum number of tape units which the system must have such that deadlocks never arise is __________.

(A)	6
(B)	7
(C)	8
(D)	9

If Tape Units =6:

WORST CASE:

Program 1- Allocated 2 tape units

Program 2- Allocated 2 tape units

Program 3- Allocated 2 tape units

All needs 1 more tape unit to complete execution and end up waiting for each other for 1 more tape unit.

But, if there is 1 more Tape Unit (i.e total number of tape units =7) then, at least one of the programs can be allocated that tape unit and can finish its execution and release all the tape units that it was holding and hence allow the next 2 programs also to complete their execution.

Question 2

GATE 2015

Consider the following policies for preventing deadlock in a system with mutually exclusive resources.

I. Processes should acquire all their resources at the beginning of execution. If any resource is not available, all resources acquired so far are released.

II. The resources are numbered uniquely, and processes are allowed to request for resources only in increasing resource numbers.

III. The resources are numbered uniquely, and processes are allowed to request for resources only in decreasing resource numbers.

IV. The resources are numbered uniquely. A process is allowed to request only for a resource with resource number larger than its currently held resources.

Which of the above policies can be used for preventing deadlock?

(A) Any one of I and III but not II or IV
(B) Any one of I, III and IV but not II
(C) Any one of II and III but not I or IV
(D) Any one of I, II, III and IV

Question 2

GATE 2015

Consider the following policies for preventing deadlock in a system with mutually exclusive resources.

I. Processes should acquire all their resources at the beginning of execution. If any resource is not available, all resources acquired so far are released.

II. The resources are numbered uniquely, and processes are allowed to request for resources only in increasing resource numbers.

III. The resources are numbered uniquely, and processes are allowed to request for resources only in decreasing resource numbers.

IV. The resources are numbered uniquely. A process is allowed to request only for a resource with resource number larger than its currently held resources.

Policy I: Will avoid HOLD AND WAIT

Policy II: Will avoid CIRCULAR WAIT

Policy III: Will avoid CIRCULAR WAIT

Policy IV: Will avoid CIRCULAR WAIT

Deadlocks

Solved Problems (Part-4)

GATE 2018

In a system, there are three types of resources: E, F and G. Four processes P0, P1, P2 and P3 execute concurrently. At the outset, the processes have declared their maximum resource requirements using a matrix named Max as given below. For example, Max[P2, F] is the maximum number of instances of F that P2 would require. The number of instances of the resources allocated to the various processes at any given state is given by a matrix named Allocation.

Consider a state of the system with the Allocation matrix as shown below, and in which 3 instances of E and 3 instances of F are the only resources available.

Allocation			
	E	F	G
P0	1	0	1
P1	1	1	2
P2	1	0	3
P3	2	0	0

Max			
	E	F	G
P0	4	3	1
P1	2	1	4
P2	1	3	3
P3	5	4	1

From the perspective of deadlock avoidance, which one of the following is true?

(A) The system is in safe state.
(B) The system is not in safe state, but would be safe if one more instance of E were available.
(C) The system is not in safe state, but would be safe if one more instance of F were available.
(D) The system is not in safe state, but would be safe if one more instance of G were available.

Allocation			
	E	F	G
P0	1	0	1
P1	1	1	2
P2	1	0	3
P3	2	0	0

Max			
	E	F	G
P0	4	3	1
P1	2	1	4
P2	1	3	3
P3	5	4	1

	Allocation			Max			Need			Available		
	E	F	G	E	F	G	E	F	G	E	F	G
P0	1	0	1	4	3	1	3	3	0	3	3	0
P1	1	1	2	2	1	4	1	0	2			
P2	1	0	3	1	3	3	0	3	0			
P3	2	0	0	5	4	1	3	4	1			

	Allocation			Max			Need			Available		
	E	F	G	E	F	G	E	F	G	E	F	G
P0	1	0	1	4	3	1	3	3	0	6	4	6
P1	1	1	2	2	1	4	1	0	2			
P2	1	0	3	1	3	3	0	3	0			
P3	2	0	0	5	4	1	3	4	1			

Available (3, 3, 0), which can satisfy either process P0 or P2.

If we start with P0:
P0 <3, 3, 0> After completion, Available count = (3, 3, 0) + (1, 0, 1) = (4, 3, 1)

P2 <0, 3, 0> After completion, Available count = (4, 3, 1) + (1, 0, 3) = (5, 3, 4)

P1 <1, 0, 2> After completion, Available count = (5, 3, 4) + (1, 1, 2) = (6, 4, 6)

Deadlocks

Solved Problems (Part-4)

GATE 2018

In a system, there are three types of resources: E, F and G. Four processes P0, P1, P2 and P3 execute concurrently. At the outset, the processes have declared their maximum resource requirements using a matrix named Max as given below. For example, Max[P2, F] is the maximum number of instances of F that P2 would require. The number of instances of the resources allocated to the various processes at any given state is given by a matrix named Allocation.

Consider a state of the system with the Allocation matrix as shown below, and in which 3 instances of E and 3 instances of F are the only resources available.

Allocation			
	E	F	G
P0	1	0	1
P1	1	1	2
P2	1	0	3
P3	2	0	0

Max			
	E	F	G
P0	4	3	1
P1	2	1	4
P2	1	3	3
P3	5	4	1

From the perspective of deadlock avoidance, which one of the following is true?

(A) The system is in safe state.
(B) The system is not in safe state, but would be safe if one more instance of E were available.
(C) The system is not in safe state, but would be safe if one more instance of F were available.
(D) The system is not in safe state, but would be safe if one more instance of G were available.

Deadlocks

Solved Problems (Part-5)

ISRO CS 2017

Question 1

Which of the following is not true with respect to deadlock prevention and deadlock avoidance schemes?

(A) In deadlock prevention, the request for resources is always granted if resulting state is safe.

(B) In deadlock avoidance, the request for resources is always granted, if the resulting state is safe.

(C) Deadlock avoidance requires knowledge of resource requirements a priori.

(D) Deadlock prevention is more restrictive than deadlock avoidance.

Deadlocks

Solved Problems (Part-5)

ISRO CS 2016

Question 2

With single resource, deadlock occurs

(A) if there are more than two processes competing for that resources

(B) if there are only two processes competing for that resources

(C) if there is a single process competing for that resources

(D) none of these

Deadlocks

Solved Problems (Part-5)

ISRO CS 2013

Question 3

Which of the following is not a necessary condition for deadlock?

(A) Mutual exclusion

(B) Reentrancy

(C) Hold and wait

(D) No pre-emption

Deadlocks

Solved Problems (Part-6)

ISRO CS 2011

Question 1

A total of 9 units of a resource type available, and given the safe state shown below, which of the following sequence will be a safe state?

Process	Used	Max
P1	2	7
P2	1	6
P3	2	5
P4	1	4

(A) (P4, P1, P3, P2)

(B) (P4, P2, P1, P3)

(C) (P4, P2, P3, P1)

(D) (P3, P1, P2, P4)

Process	Used	Max
P1	2	7
P2	1	6
P3	2	5
P4	1	4

Total number of units of the resource available = 9

Available = Total available - Total used
= 9 - 6 = 3

Process	Used	Max	Need	Available
P1	2	7	5	3
P2	1	6	5	
P3	2	5	3	
P4	1	4	3	

Total used = 6

Process	Used	Max	Need	Available
P1	2	7	5	4
P2	1	6	5	
P3	2	5	3	
P4	1	4	3	

(A) (P4, P1, P3, P2)

(B) (P4, P2, P1, P3)

(C) (P4, P2, P3, P1)

(D) (P3, P1, P2, P4)

If we start with P4:

P4 <3> After completion, Available count = **3 + 1 = 4**

Now if P1 or P2 tries to execute, their requests cannot be granted as they both need 5 units of the resource which is currently unavailable.

Deadlocks

Solved Problems (Part-6)

Question 2

ISRO CS 2009

When a process is rolled back as a result of deadlock the difficulty which arises is

(A) Starvation

(B) System throughput

(C) Low device utilization

(D) Cycle stealing