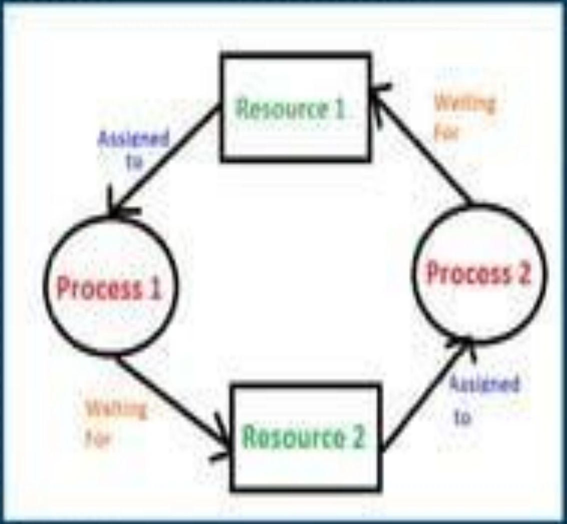


What is a Dead-Lock ?

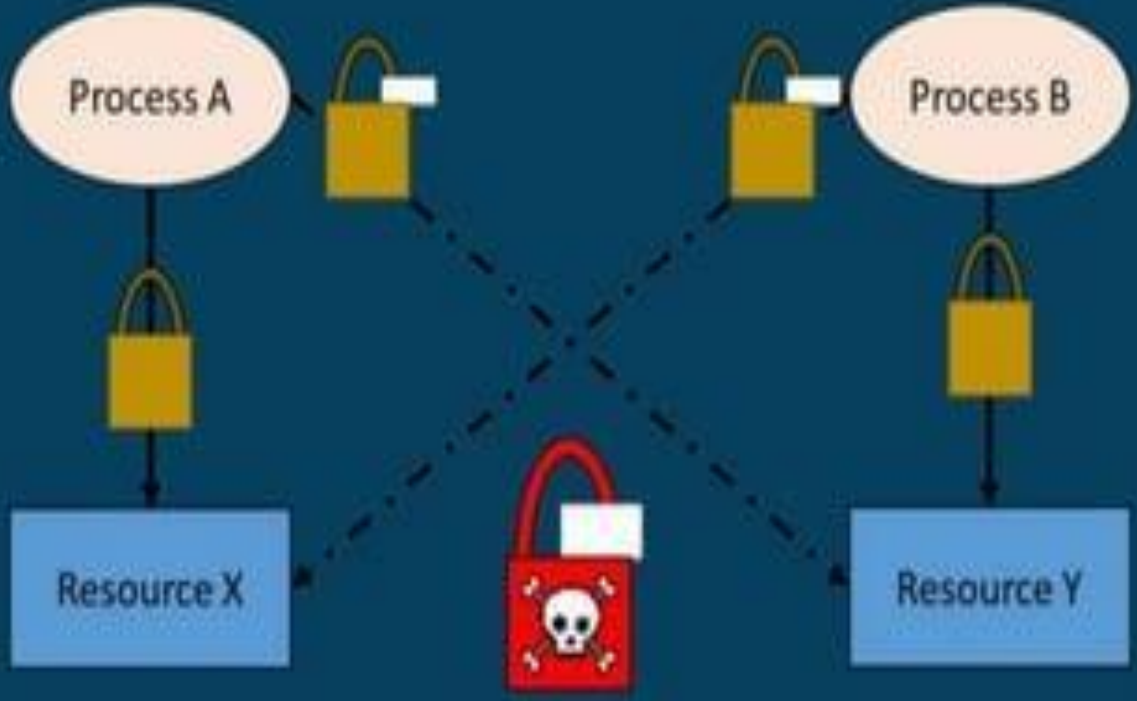
Deadlock is a **situation** where a set of processes are blocked because each process is holding a resource and waiting for another resource acquired by some

Realtime-Example of Deadlock

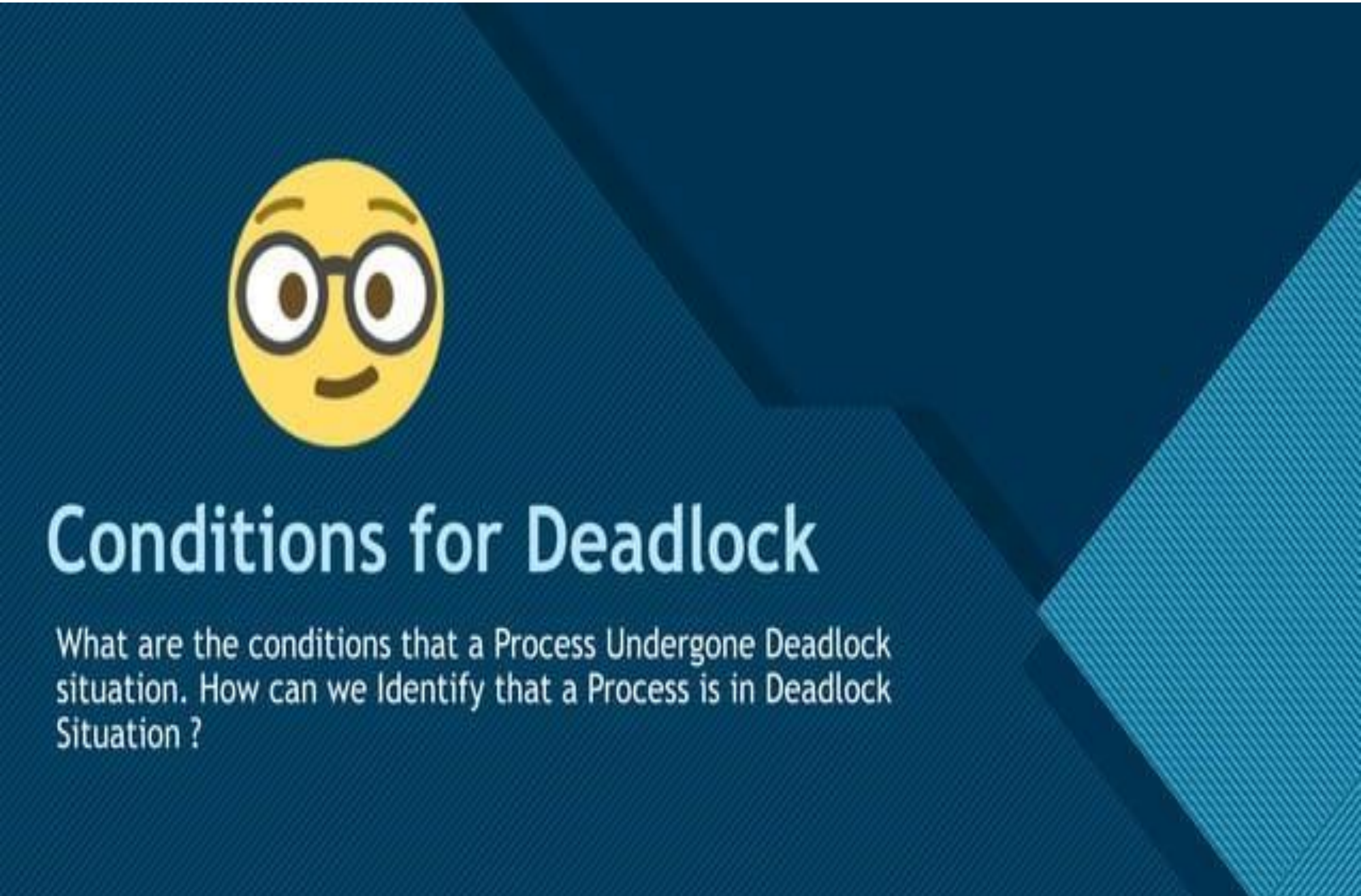
- Two Trains are Travelling on different Tracks. If a Crossing came, A Train Must Hold or wait for sometime to continue the Journey. That waiting situation Is Referred as "Deadlock" Situation.

- While playing Chess , CHECKMATE is the deadlock situation. Where, th

Deadlock Situation in Data-Base

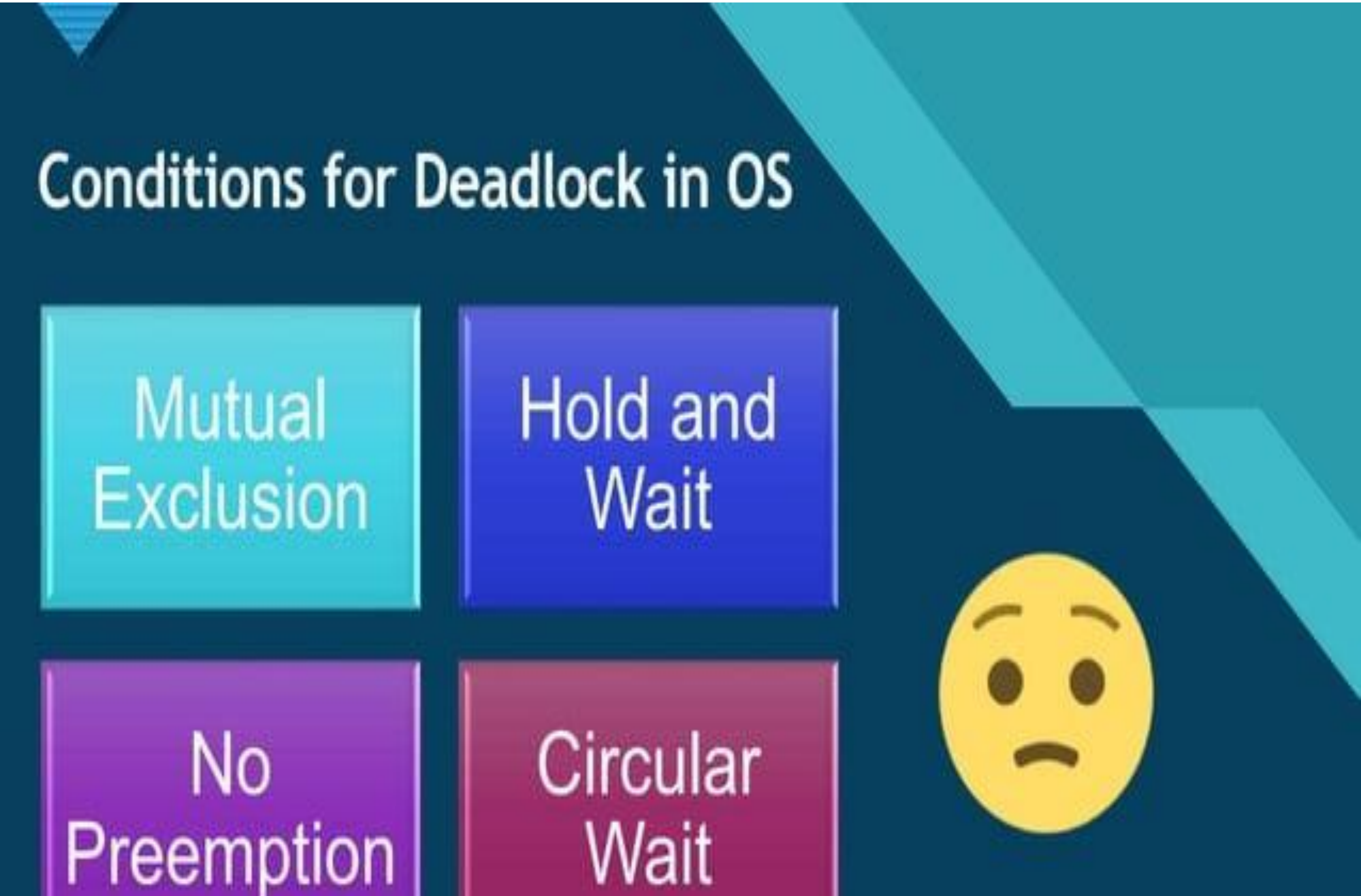


- No Two Transactions takes place at a time in the Database. It hold on
transaction In the Queue to Proceed.



Conditions for Deadlock

What are the conditions that a Process Undergone Deadlock situation. How can we Identify that a Process is in Deadlock Situation ?



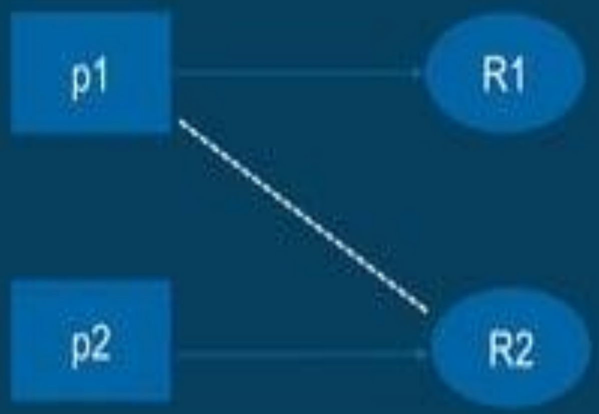
Conditions for Deadlock in OS

Mutual Exclusion	Hold and Wait
No Preemption	Circular Wait

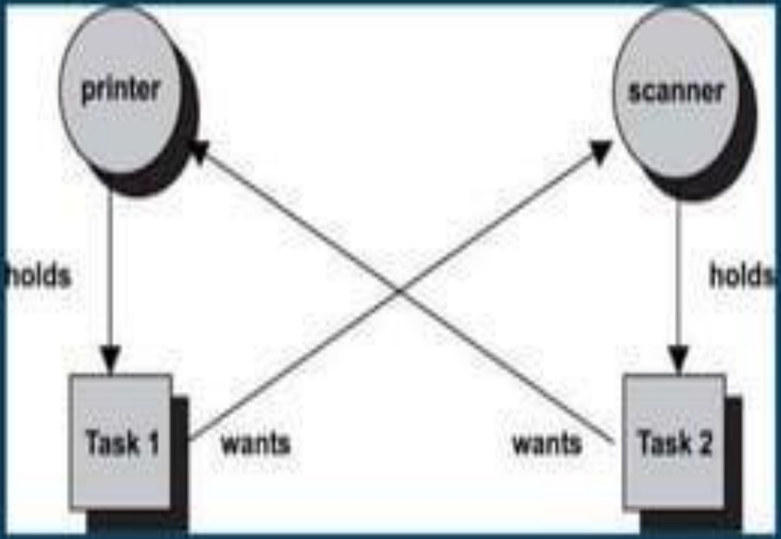
Mutual-Exclusion

- When a Process is Accessing a Shared Variable, the Process is said to be in a CRITICAL SITUATION

- One Process at a time can Use a resource.



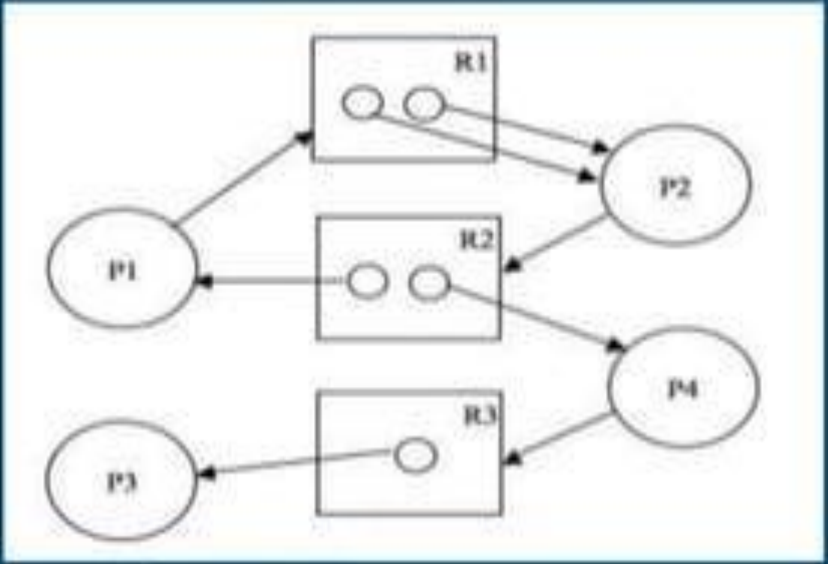
Hold and Wait



- A Process holding at least one resource is waiting to acquire additional resources held by another processes.

- Printer needs scanned copy to Print and scanner wants a copy to

No Preemption

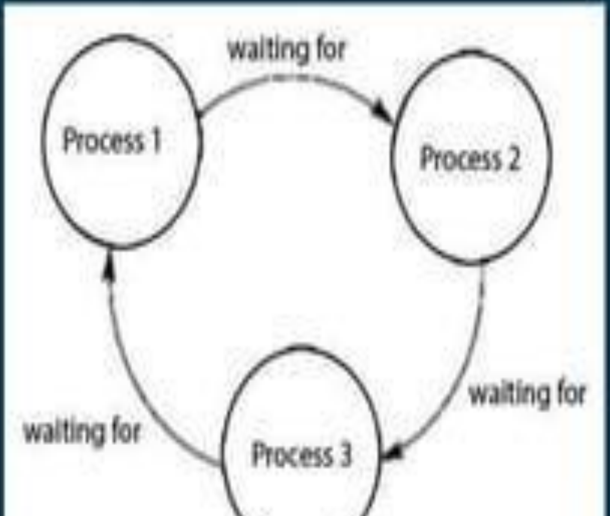


- A Resource can be released only voluntarily by the process holding it, after that process has completed its task.

Circular Wait

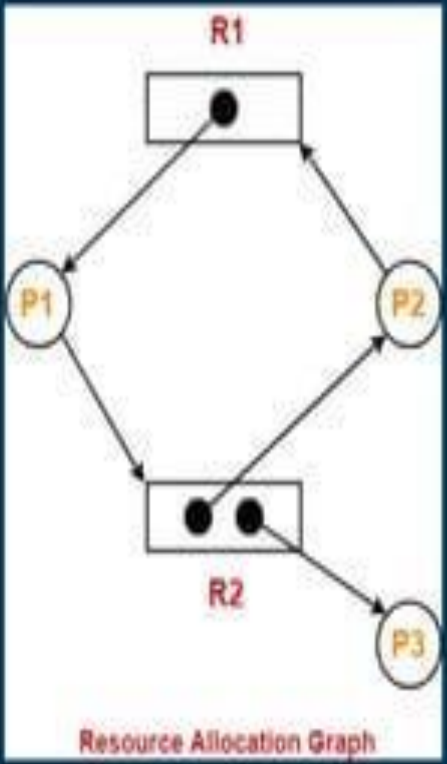
- There exists a set { p1,p2,p3…pn } of waiting processes such that p1 is waiting for a resource that Is held by p2, p2 is waiting for a resource that is held by p3.

- P_n is waiting for the resource p_{n-1}



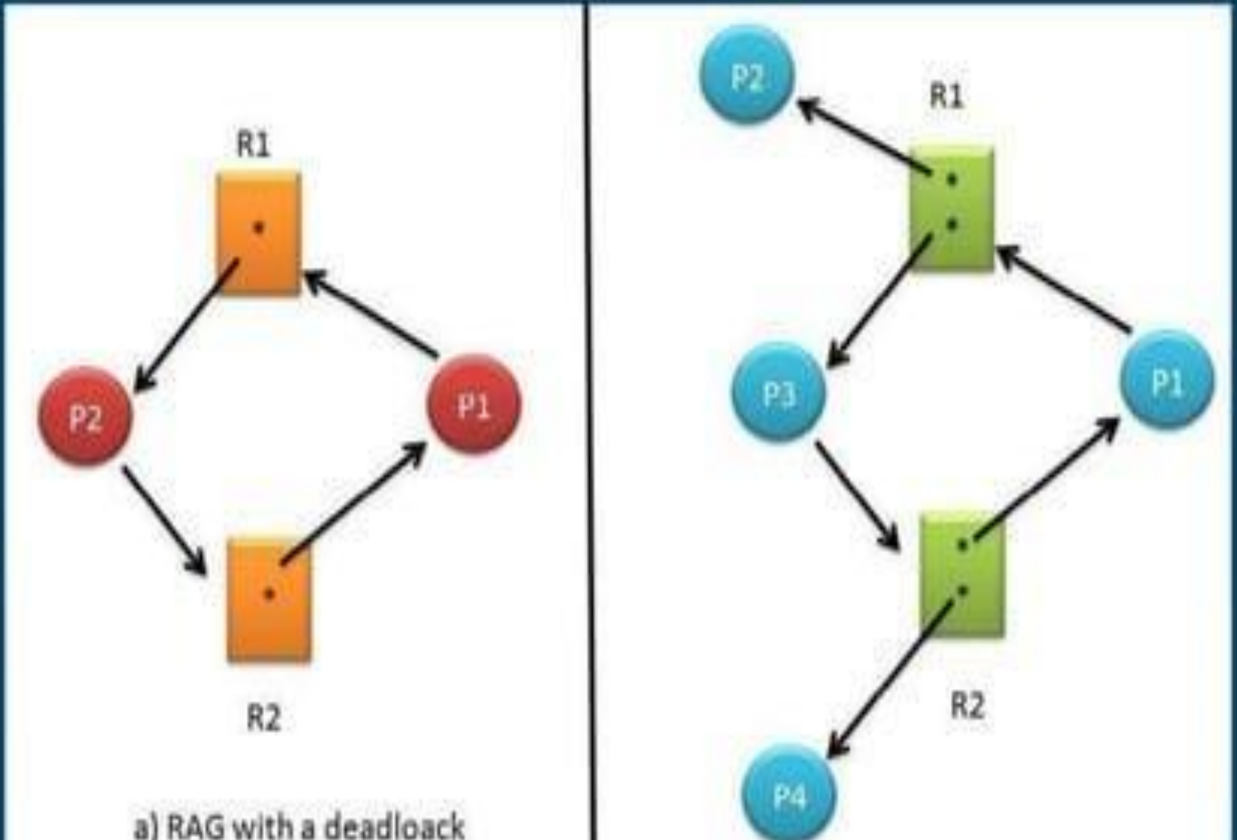
Resource-Allocation Graph

Resource Allocation Without cycles



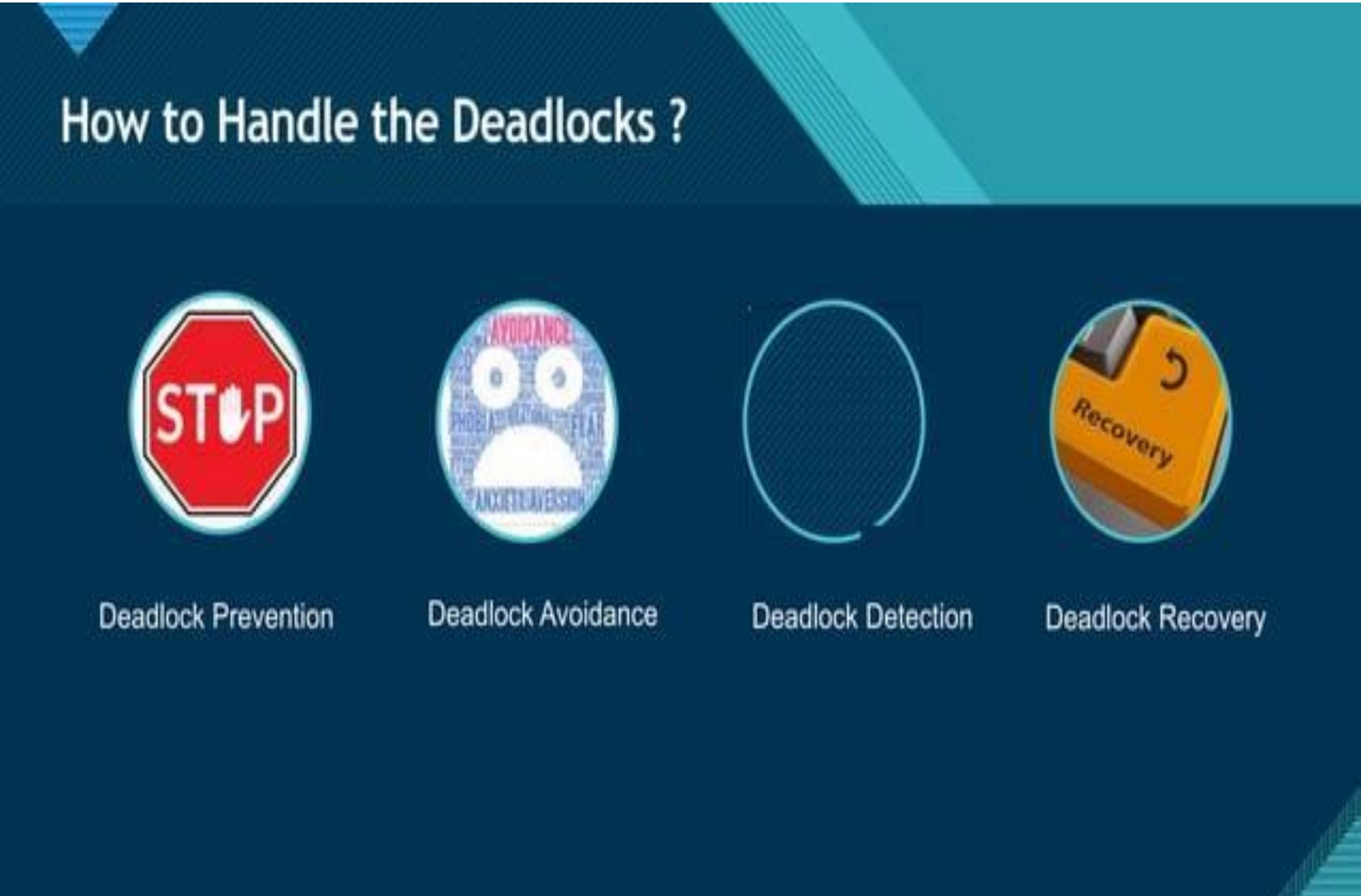
- If Graph contains no cycles → No Deadlock
- If graph contains a Cycle
- → If only one instance per resource type, then deadlock.
- → If several Instances per resource type, possibility of dead lock.

Resource Allocation Graph



a) RAG with a deadloack

→ Resource Allocation graph contains cycles & but, no deadlock



How to Handle the Deadlocks ?

Deadlock Prevention

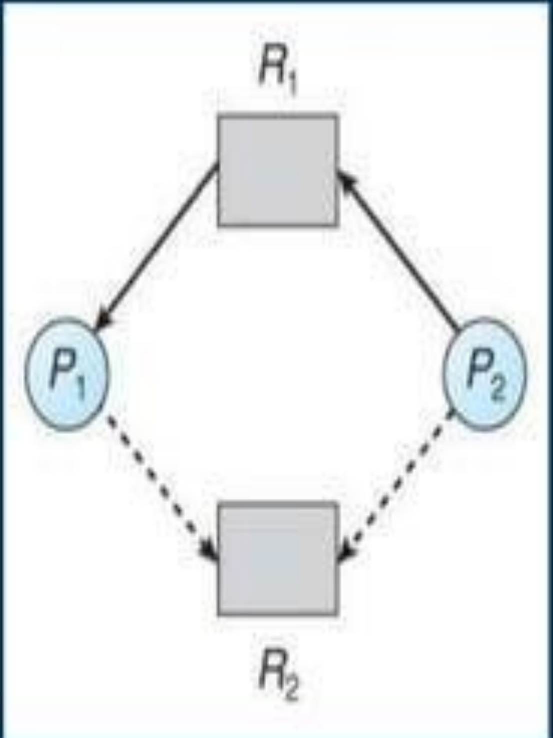
Deadlock Avoidance

Deadlock Detection

Deadlock Recovery

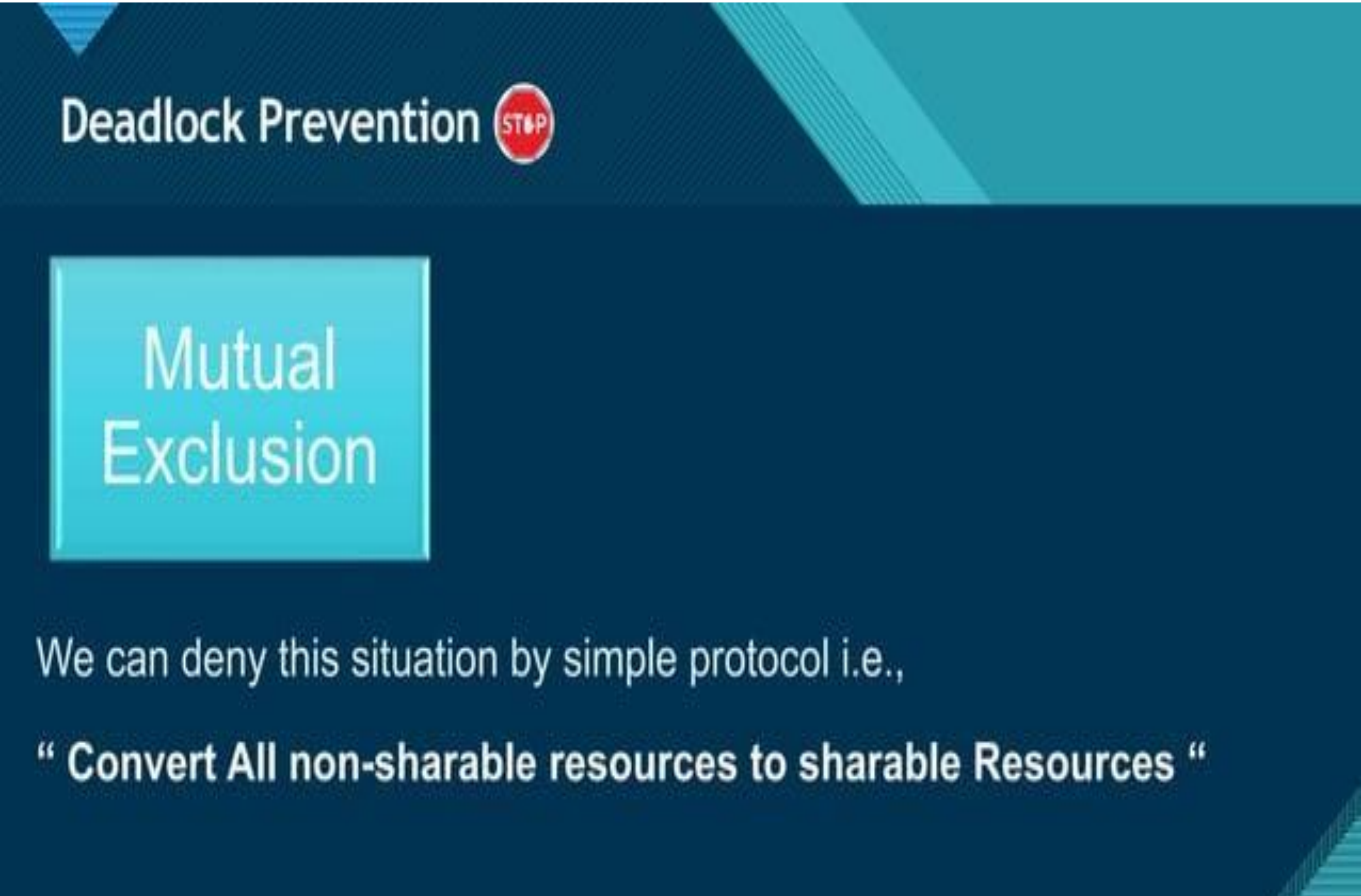
Deadlock Prevention

At least one of 4 deadlock conditions is never satisfied.



Elimination of

→ Mutual Exclusion
→ Hold & Wait Condition
→ No Preemption Condition
→ Circular Wait

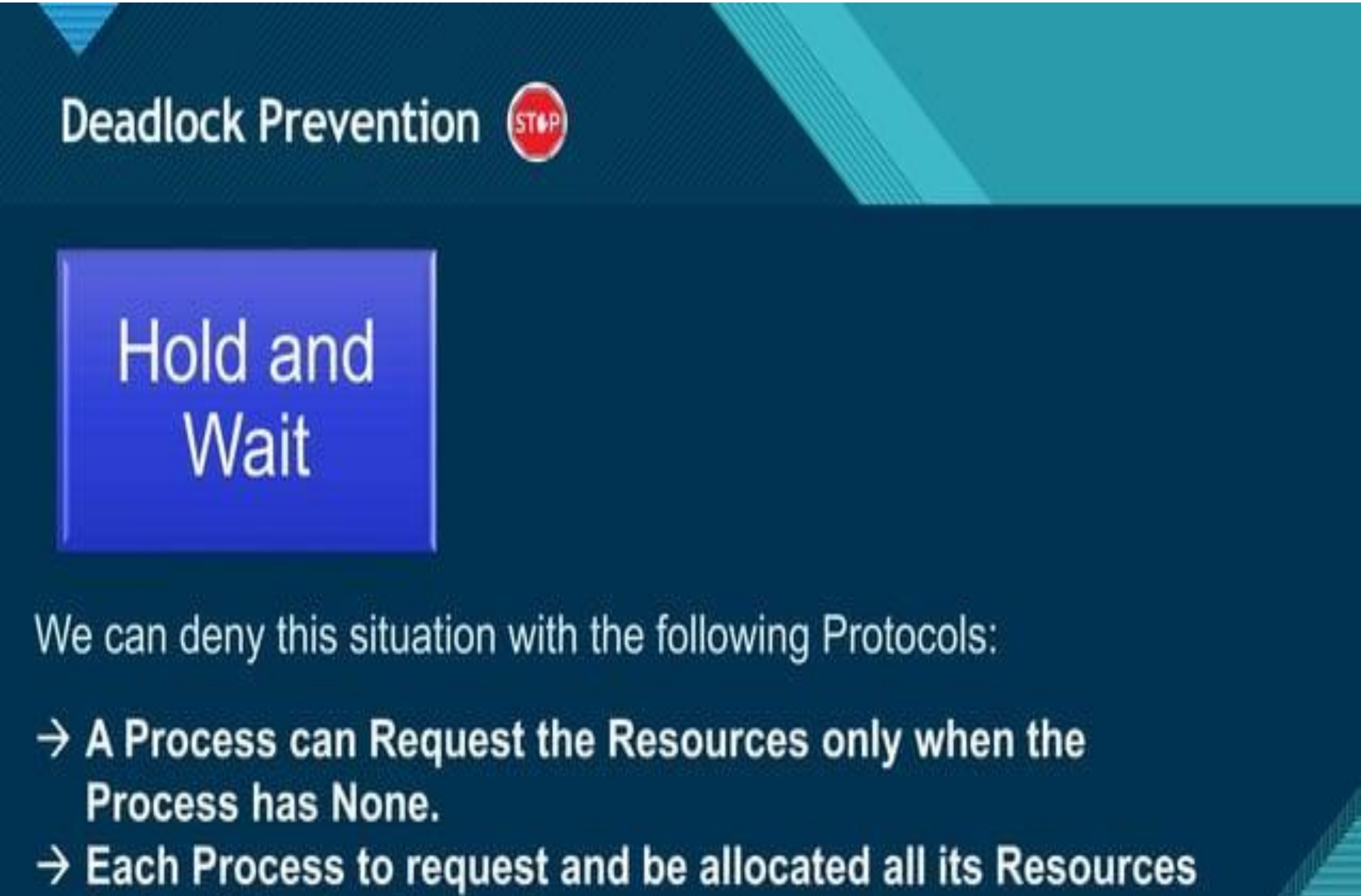


Deadlock Prevention

Mutual Exclusion

We can deny this situation by simple protocol i.e.,

“ Convert All non-sharable resources to sharable Resources “



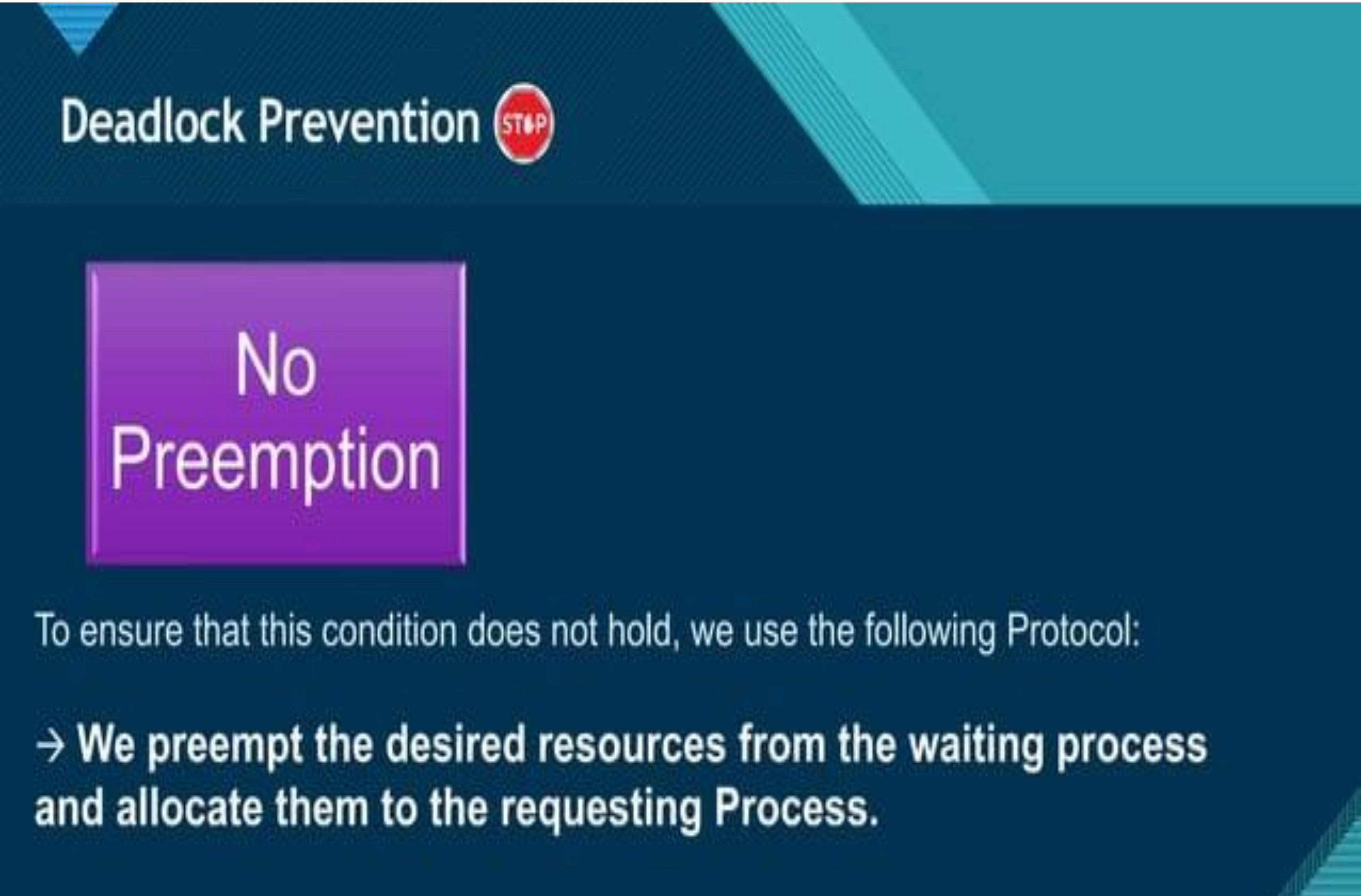
Deadlock Prevention

Hold and Wait

We can deny this situation with the following Protocols:

→ **A Process can Request the Resources only when the Process has None.**

→ **Each Process to request and be allocated all its Resources**

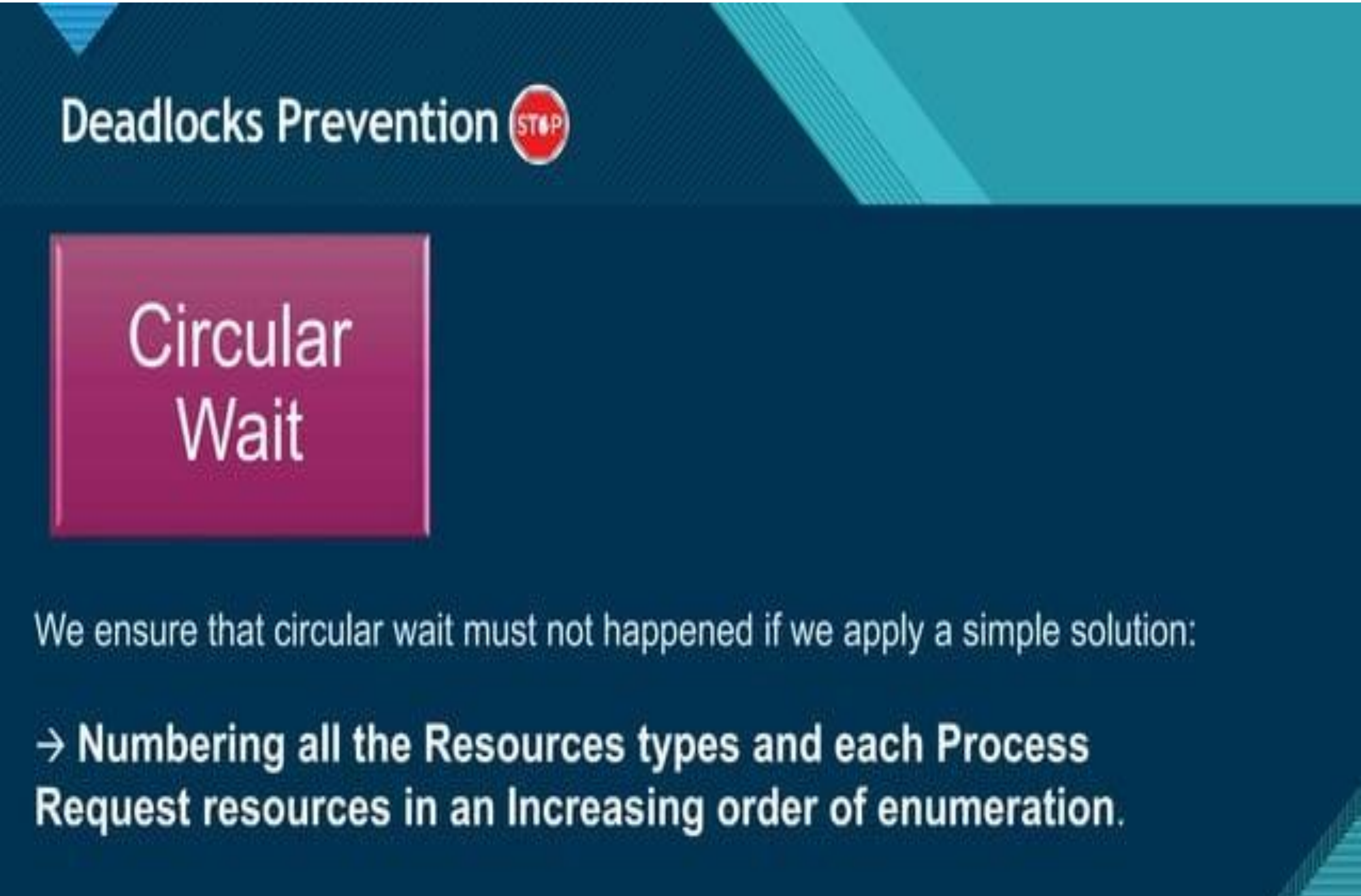


Deadlock Prevention

No Preemption

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

→ **We preempt the desired resources from the waiting process and allocate them to the requesting Process.**



Deadlocks Prevention

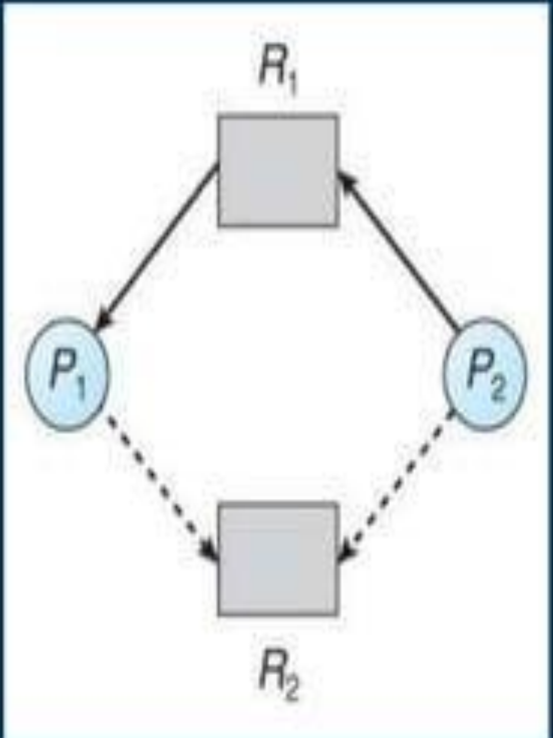
Circular Wait

We ensure that circular wait must not happened if we apply a simple solution:

→ **Numbering all the Resources types and each Process Request resources in an Increasing order of enumeration.**

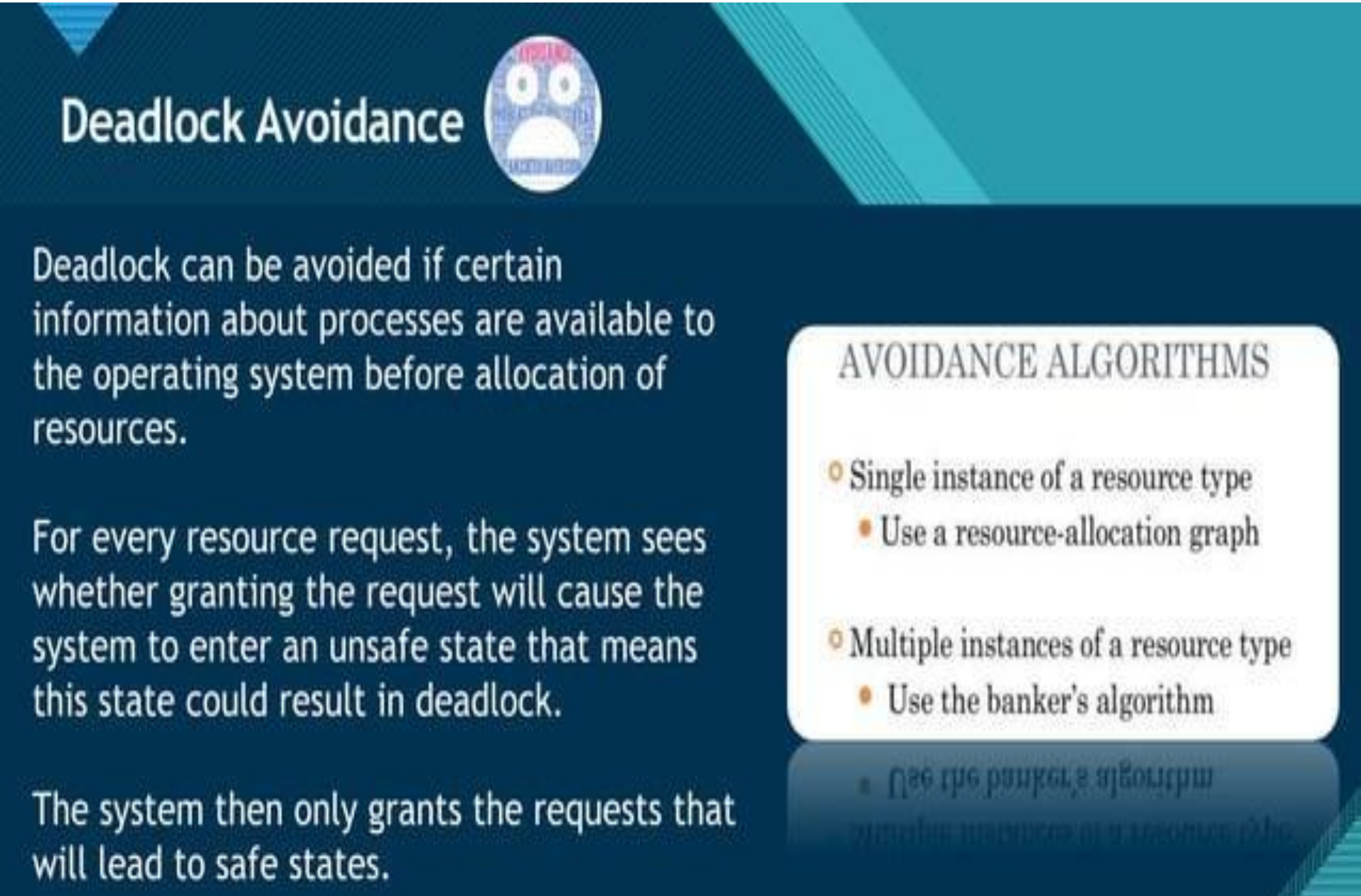
Deadlock Avoidance

At least one of 4 deadlock conditions is never satisfied.



Elimination of

→ Mutual Exclusion
→ Hold & Wait Condition
→ No Preemption Condition
→ Circular Wait



Deadlock Avoidance

Deadlock can be avoided if certain information about processes are available to the operating system before allocation of resources.

For every resource request, the system sees whether granting the request will cause the system to enter an unsafe state that means this state could result in deadlock.

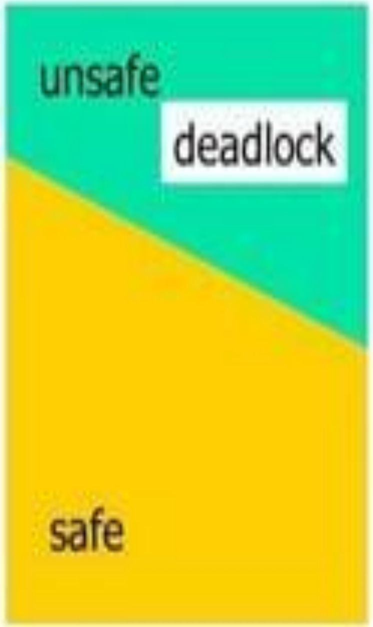
The system then only grants the requests that will lead to safe states.

AVOIDANCE ALGORITHMS

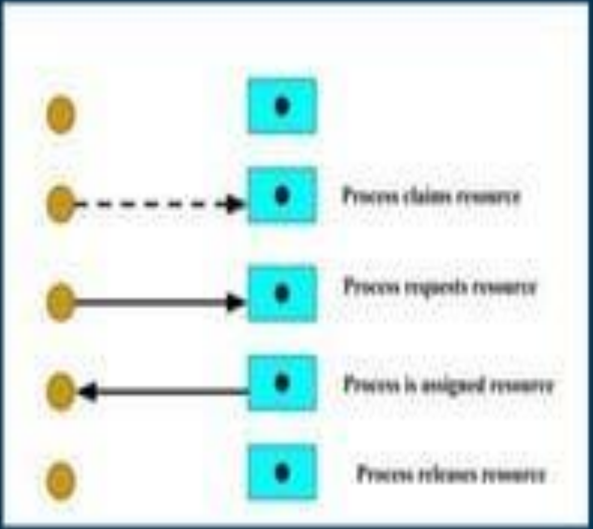
- Single instance of a resource type
  - Use a resource-allocation graph
- Multiple instances of a resource type
  - Use the banker's algorithm

Deadlock Avoidance

Avoidance – Safe/Unsafe states



- A state is safe is the system can allocate resources to each process in some order and still avoid a deadlock. More formally, a system is in a safe state only if there exists a safe sequence.
- If no safe sequence exists, then the state is said to be unsafe.


Process claims resource
Process requests resource
Process is assigned resource
Process releases resource

Deadlock Avoidance

Banker's Algorithm (cont.)

- Always keep so many resources that satisfy the needs of at least one client
- Multiple instances.
- Each process must a priori claim maximum use.
- When a process requests a resource it may have to wait.
- When a process gets all its resources it must return them in a finite amount of time.

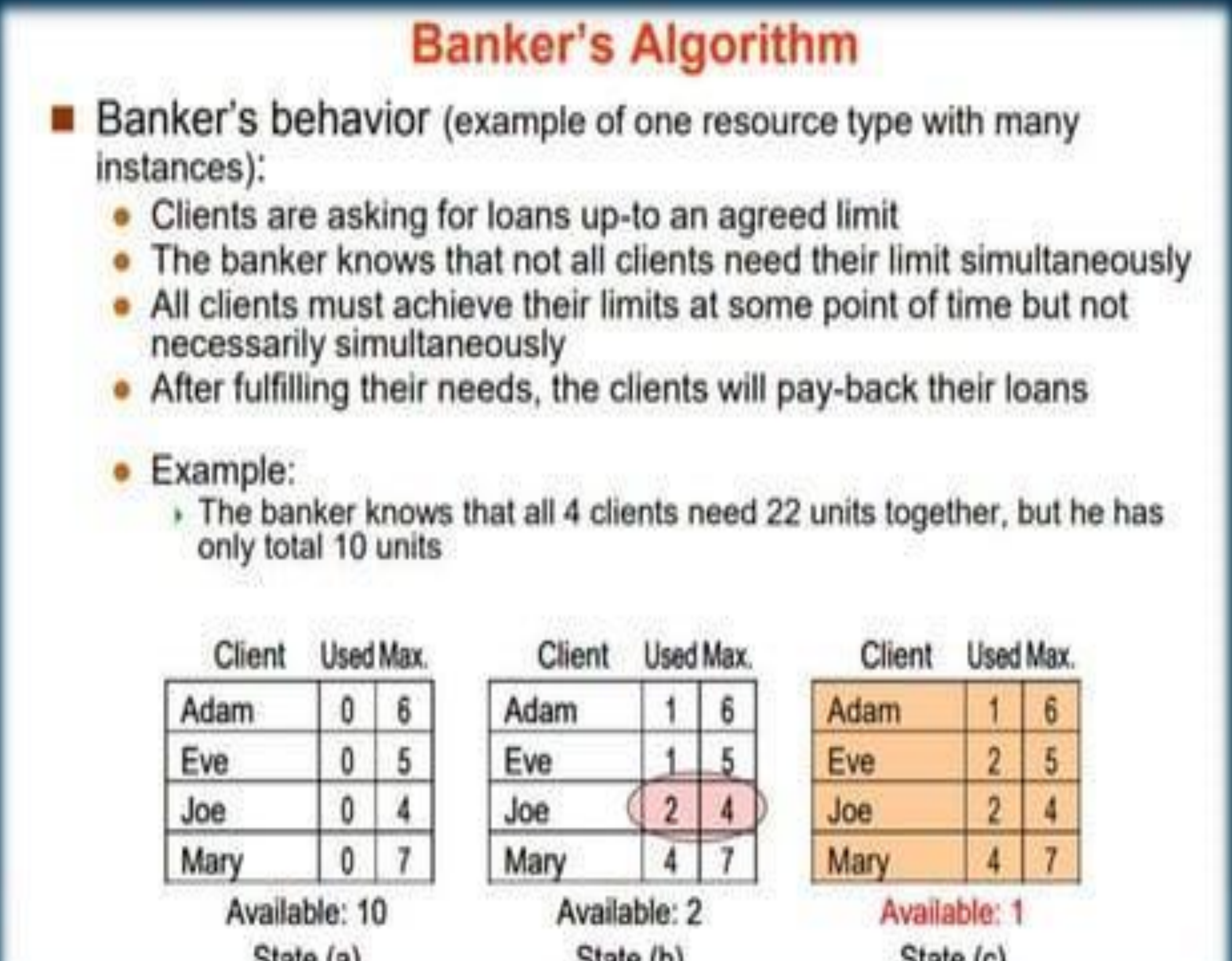
Example of Banker's Algorithm

- 5 processes P_0 through P_4; 3 resource types *A* (10 instances), *B* (5instances, and *C* (7 instances).
- Snapshot at time T_0:

	Allocation	*Max*	*Need*	*Total*
	A B C	*A B C*	*A B C*	*A B C*
P_0	0 1 0	7 5 3	7 4 3	10 5 7
P_1	2 0 0	3 2 2	1 2 2	*Allocated*
P_2	3 0 2	9 0 2	6 0 0	7 2 5
P_3	2 1 1	2 2 2	0 1 1	*Available*
P_4	0 0 2	4 3 3	4 3 1	3 3 2

- The system is in a safe state since the sequence $< P_1, P_3, P_4, P_2, P_0>$ satisfies safety criteria

Deadlock Avoidance



Banker's Algorithm

- Banker's behavior (example of one resource type with many instances):
  - Clients are asking for loans up-to an agreed limit
  - The banker knows that not all clients need their limit simultaneously
  - All clients must achieve their limits at some point of time but not necessarily simultaneously
  - After fulfilling their needs, the clients will pay-back their loans
  - Example:
    - The banker knows that all 4 clients need 22 units together, but he has only total 10 units

Client	Used	Max.
Adam	0	6
Eve	0	5
Joe	0	4
Mary	0	7

Available: 10

State (a)

Client	Used	Max.
Adam	1	6
Eve	1	5
Joe	2	4
Mary	4	7

Available: 2

State (b)

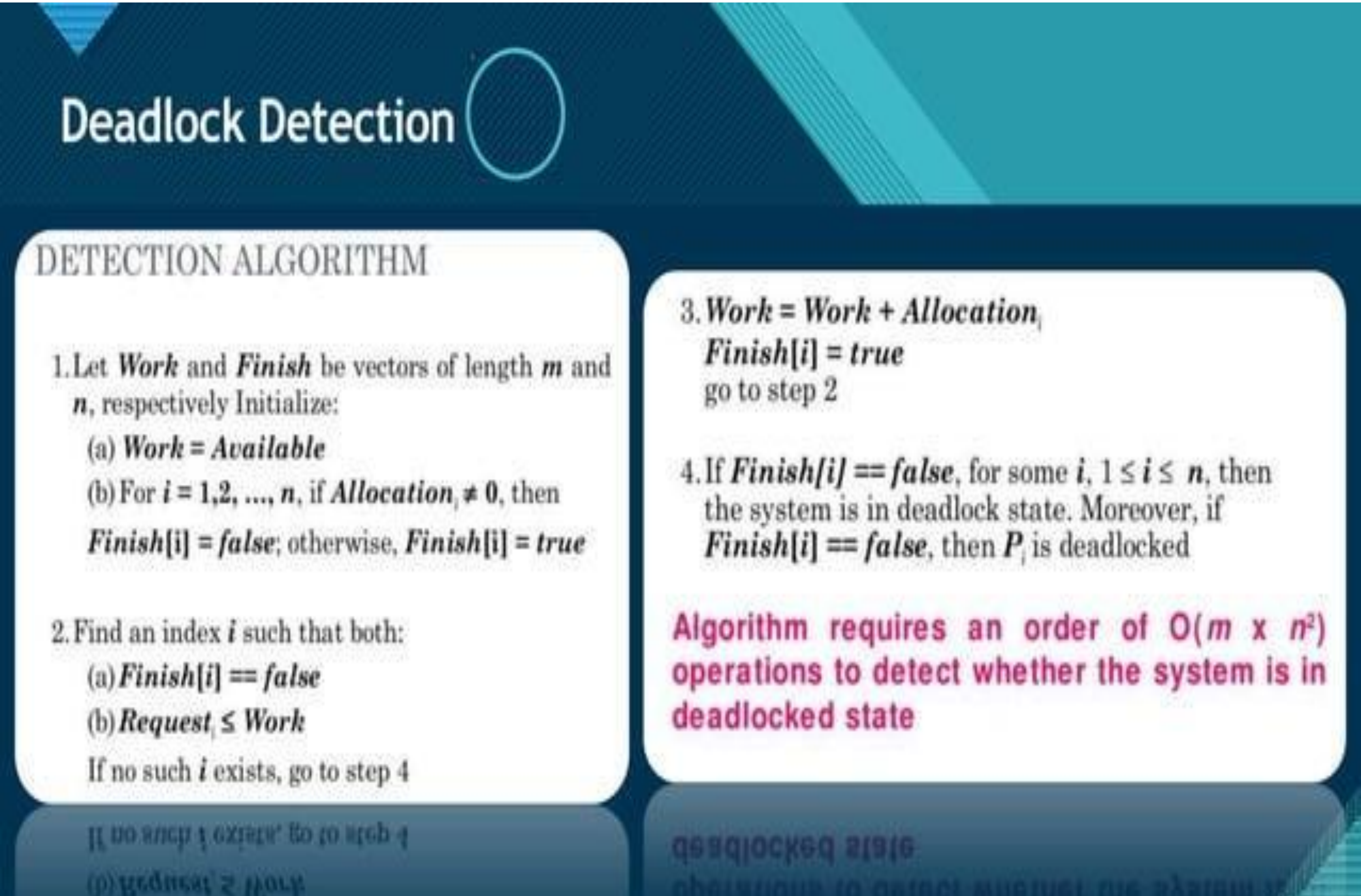
Client	Used	Max.
Adam	1	6
Eve	2	5
Joe	2	4
Mary	4	7

Available: 1

State (c)

Deadlock Detection

- Allow System to enter deadlock state
- Detection algorithm
- Recovery Scheme



Deadlock Detection

DETECTION ALGORITHM

1. Let ***Work*** and ***Finish*** be vectors of length ***m*** and ***n***, respectively Initialize:
   (a) ***Work* = *Available***
   (b) For ***i* = 1,2, ..., *n***, if ***Allocation***$_i$ **≠ 0**, then ***Finish*[i] = *false***; otherwise, ***Finish*[i] = *true***

2. Find an index ***i*** such that both:
   (a) ***Finish*[*i*] == *false***
   (b) ***Request***$_i$ **≤ *Work***
   If no such ***i*** exists, go to step 4

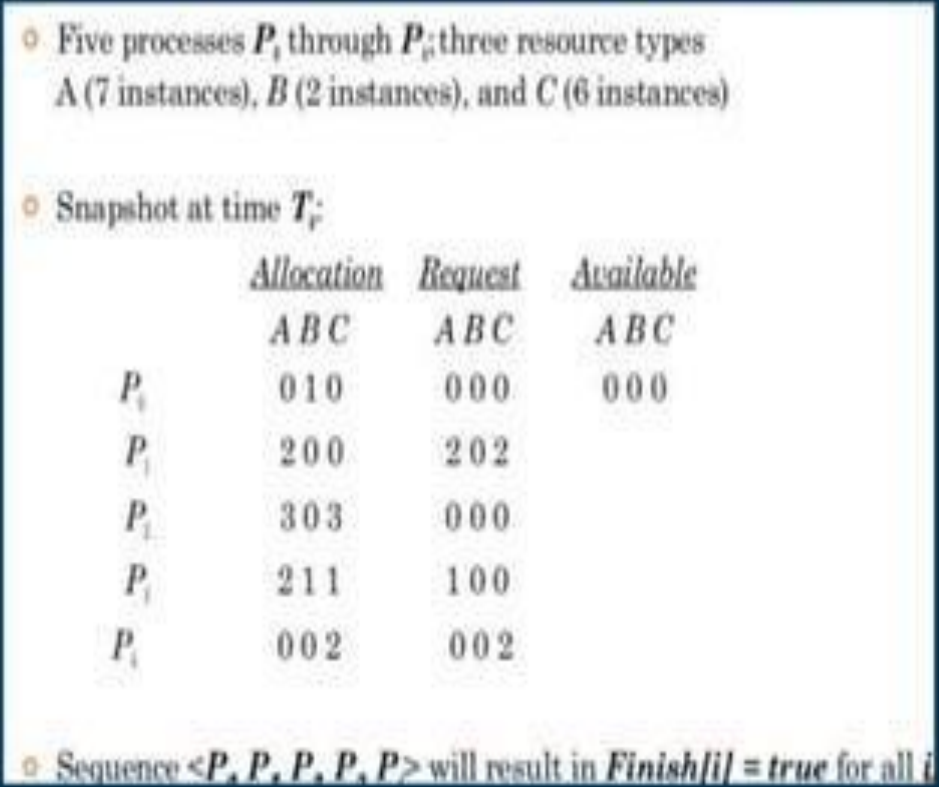
3. ***Work* = *Work* + *Allocation***$_i$
   Finish*[*i*] = *true
   go to step 2

4. If ***Finish[i]* == *false***, for some ***i***, 1 ≤ ***i*** ≤ ***n***, then the system is in deadlock state. Moreover, if ***Finish*[i] == *false***, then ***P***$_i$ is deadlocked

Algorithm requires an order of O(*m* x *n*^{2}) operations to detect whether the system is in deadlocked state

Deadlock Detection

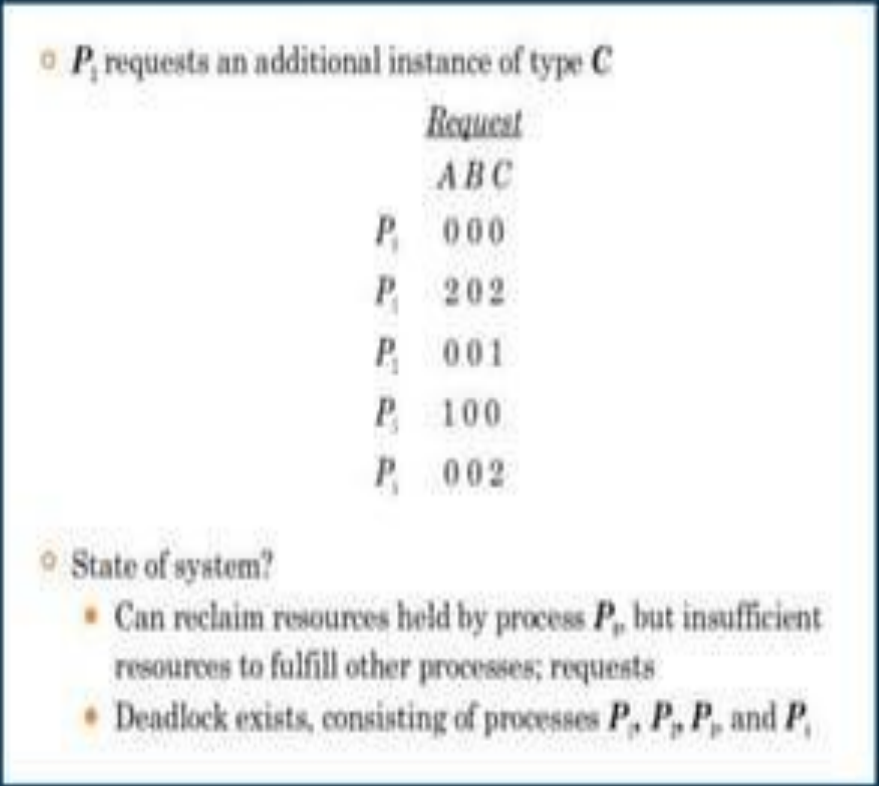
Example for Detection Algorithm



- Five processes P_0 through P_4; three resource types A (7 instances), B (2 instances), and C (6 instances)
- Snapshot at time T_0:

	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	

- Sequence $<P_0, P_2, P_3, P_1, P_4>$ will result in ***Finish[i] = true*** for all i

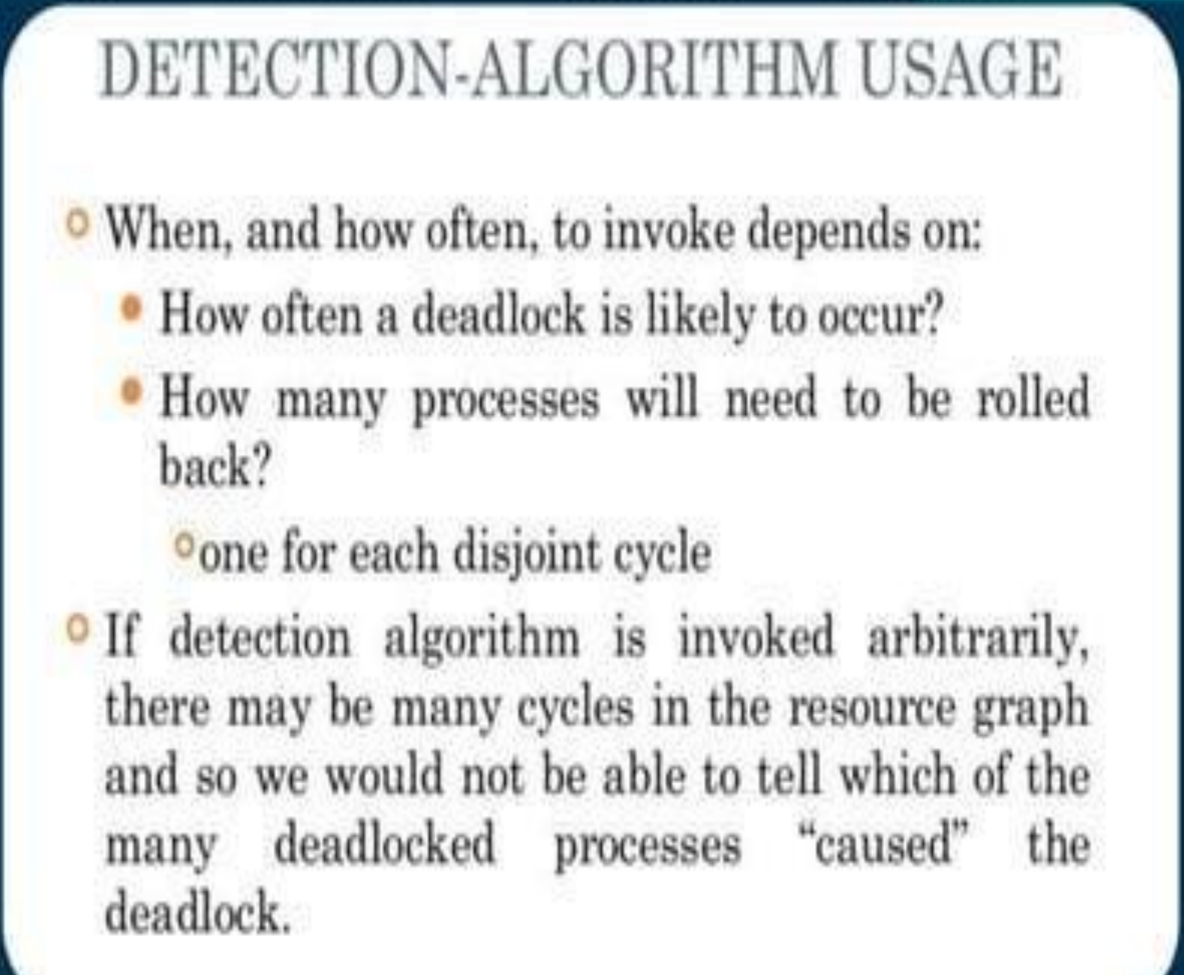


- P_2 requests an additional instance of type C

	Request
	A B C
P_0	0 0 0
P_1	2 0 2
P_2	0 0 1
P_3	1 0 0
P_4	0 0 2

- State of system?
  - Can reclaim resources held by process P_0, but insufficient resources to fulfill other processes; requests
  - Deadlock exists, consisting of processes P_1, P_2, P_3, and P_4

Deadlock Detection

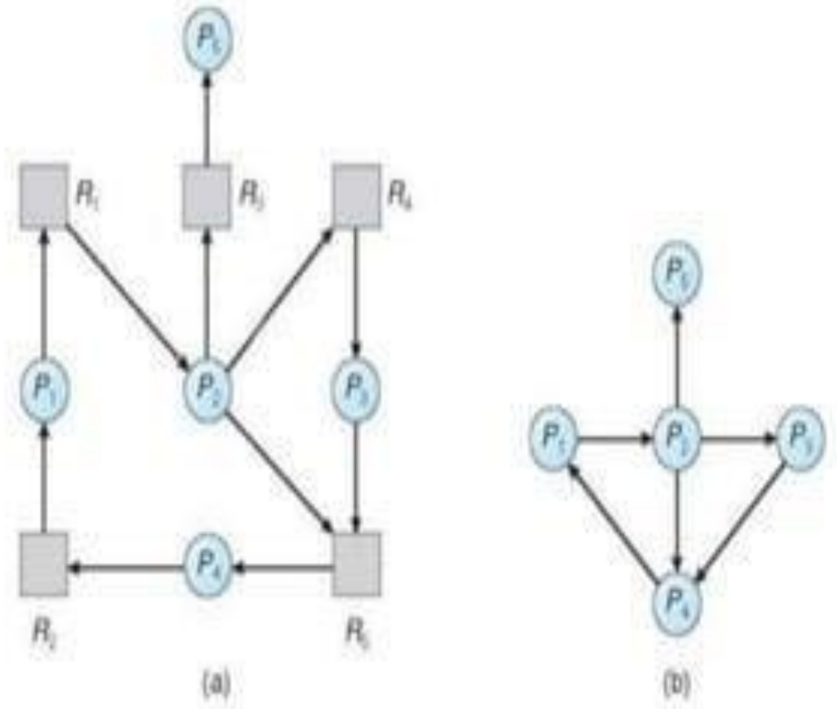


DETECTION-ALGORITHM USAGE

- When, and how often, to invoke depends on:
  - How often a deadlock is likely to occur?
  - How many processes will need to be rolled back?
    - one for each disjoint cycle
- If detection algorithm is invoked arbitrarily, there may be many cycles in the resource graph and so we would not be able to tell which of the many deadlocked processes "caused" the deadlock.

Deadlock Detection

RESOURCE-ALLOCATION GRAPH AND WAIT-FOR GRAPH



Resource-Allocation Graph   Corresponding wait-for graph

Once Deadlock has been detected, Some Strategy is needed for recovery. The various approaches of recovering from deadlocks are:

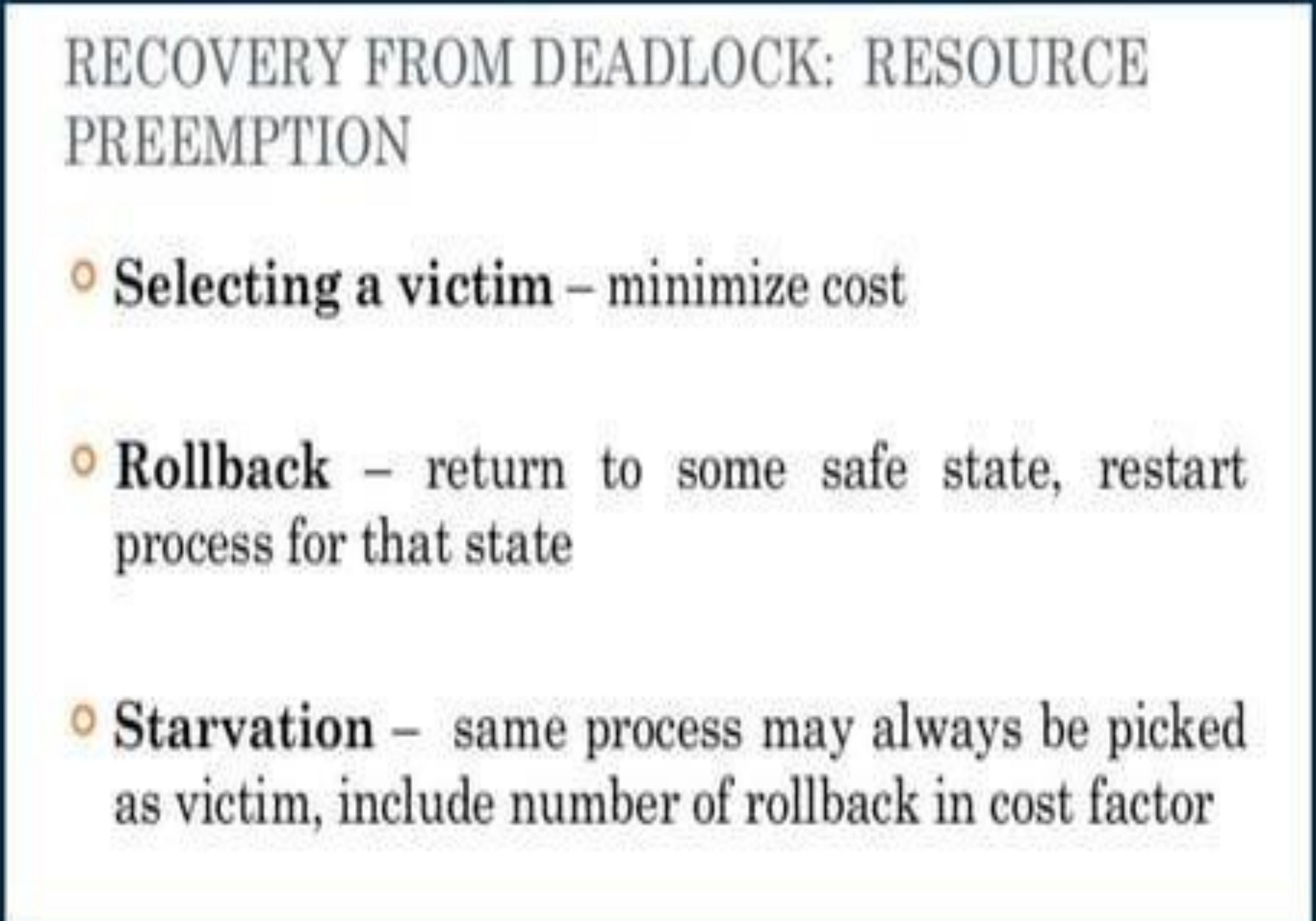
- **Process Termination**
- **Resource Preemption**

Deadlock Recovery

- Abort all deadlocked processes
- Abort one process at a time until the deadlock cycle is eliminated
- In which order should we choose to abort?
  1. Priority of the process
  2. How long process has computed, and how much longer to completion
  3. Resources the process has used
  4. Resources process needs to complete
  5. How many processes will need to be terminated
  6. Is process interactive or batch?

Process termination occurs when the **process** is terminated The exit() system call is used by most operating systems for **process termination.** This **process** leaves the processor and releases all its resources.

Deadlock Recovery



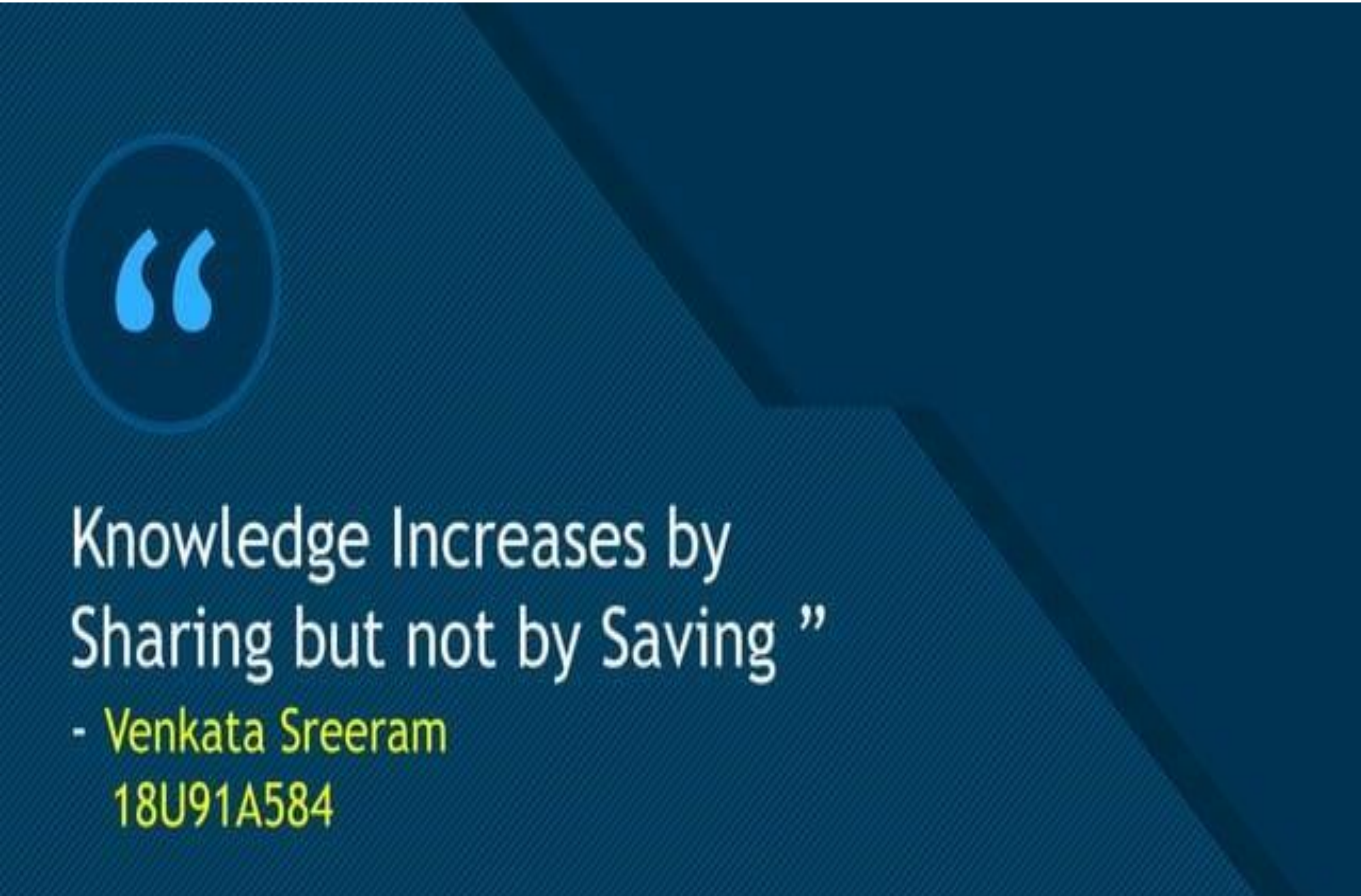
RECOVERY FROM DEADLOCK: RESOURCE PREEMPTION

- **Selecting a victim** – minimize cost
- **Rollback** – return to some safe state, restart process for that state
- **Starvation** – same process may always be picked as victim, include number of rollback in cost factor

Deadlocks-Concept Overview

Beyond the concepts of Deadlocks, It was very easy to understand with Realtime concepts where the resources has been allocated to the processors while performing operations in Operating System.

We must ensure that the recovery Strategies of deadlocks and prevention, avoidance Algorithms well by giving suitable examples.



“

Knowledge Increases by Sharing but not by Saving ”

- Venkata Sreeram
  18U91A584

😁Thank you !