Unit 3

Deadlocks

Main Topics

System Model

Deadlock Characterization

Methods for Handling Deadlocks

Deadlock Prevention

Deadlock Avoidance

Deadlock Detection

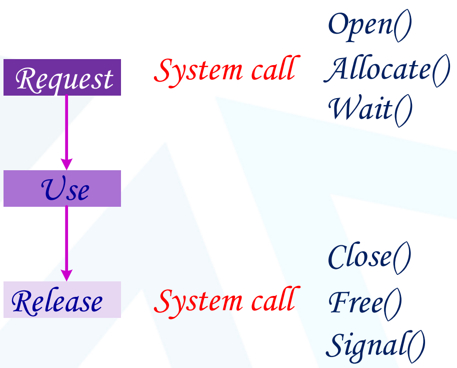
Recovery from Deadlock

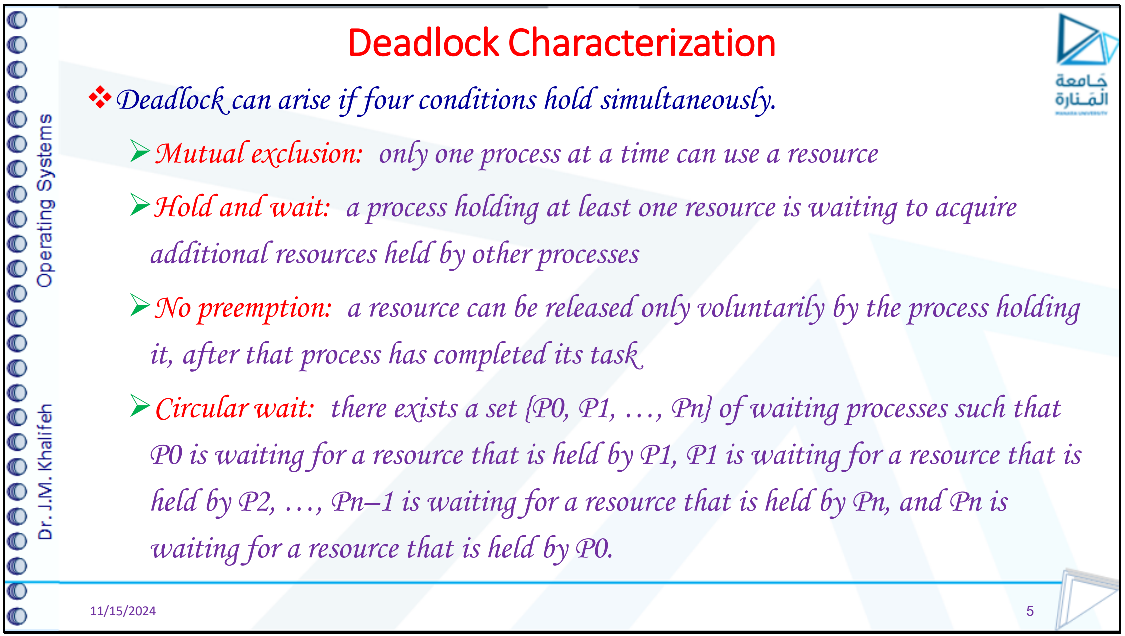
Chapter Objectives

- *To develop a description of deadlocks, which prevent sets of concurrent processes from completing their tasks*
- *To present a number of different methods for preventing or avoiding deadlocks in a computer system.*

System Model

- *System consists of resources*
- *Resource types R1, R2, . . ., Rm*
  - ✓ *CPU cycles, memory space, I/O devices*
- *Each resource type Ri has Wi instances.*
- *Each process utilizes a resource as follows:*
  - *request*
  - *use*
  - *release*

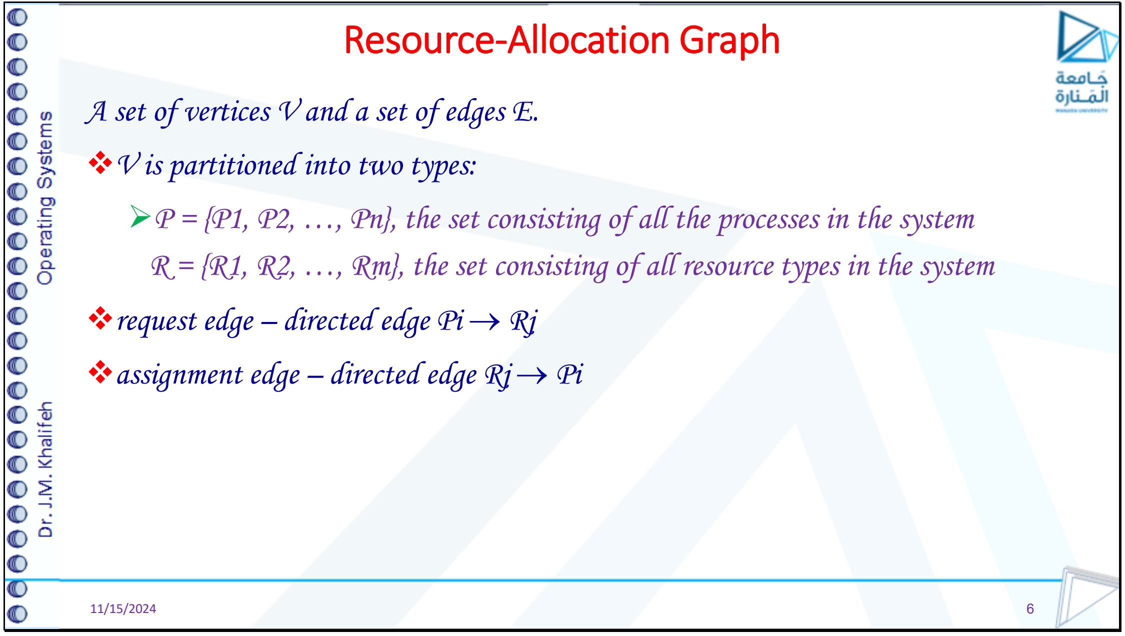




Deadlock Characterization

❖ *Deadlock can arise if four conditions hold simultaneously.*

- *Mutual exclusion: only 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 other processes*
- *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 {P0, P1, …, Pn} of waiting processes such that P0 is waiting for a resource that is held by P1, P1 is waiting for a resource that is held by P2, …, Pn–1 is waiting for a resource that is held by Pn, and Pn is waiting for a resource that is held by P0.*



Resource-Allocation Graph

A set of vertices V and a set of edges E.

❖ *V is partitioned into two types:*

- *P = {P1, P2, …, Pn}, the set consisting of all the processes in the system*

  R = {R1, R2, …, Rm}, the set consisting of all resource types in the system

❖ *request edge – directed edge Pi → Rj*

❖ *assignment edge – directed edge Rj → Pi*

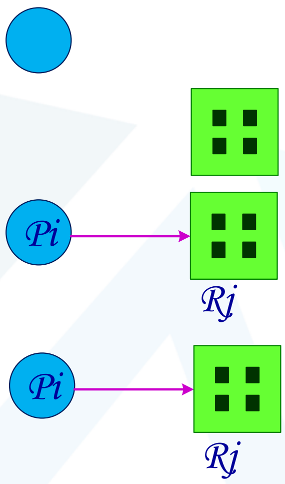
Resource-Allocation Graph (Cont.)

Process

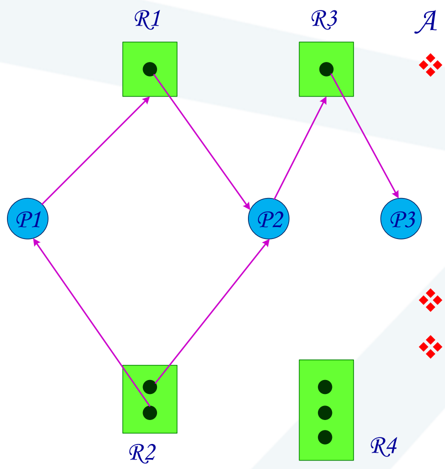
Resource Type with 4 instances

Pi requests instance of Rj

Pi is holding an instance of Rj



Example of a Resource Allocation Graph

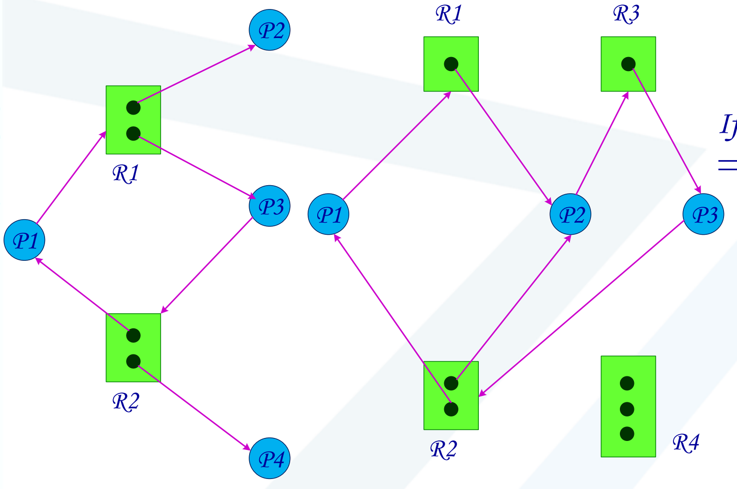


A set of vertices V and a set of edges E.

- *V is partitioned into two types:*
  - *P = {P1, P2, …, Pn}, the set consisting of all the processes in the system*

    R = {R1, R2, …, Rm}, the set consisting of all resource types in the system
- *request edge – directed edge Pi → Rj*
- *assignment edge – directed edge Rj → Pi*
  - *E={P1→R1, P2 → R3, R1 → P2, R2 → P2, R2 → P1, R3 → P3}*

Graph With A Cycle: with Deadlock and with No Deadlock



If graph contains a cycle ⇒

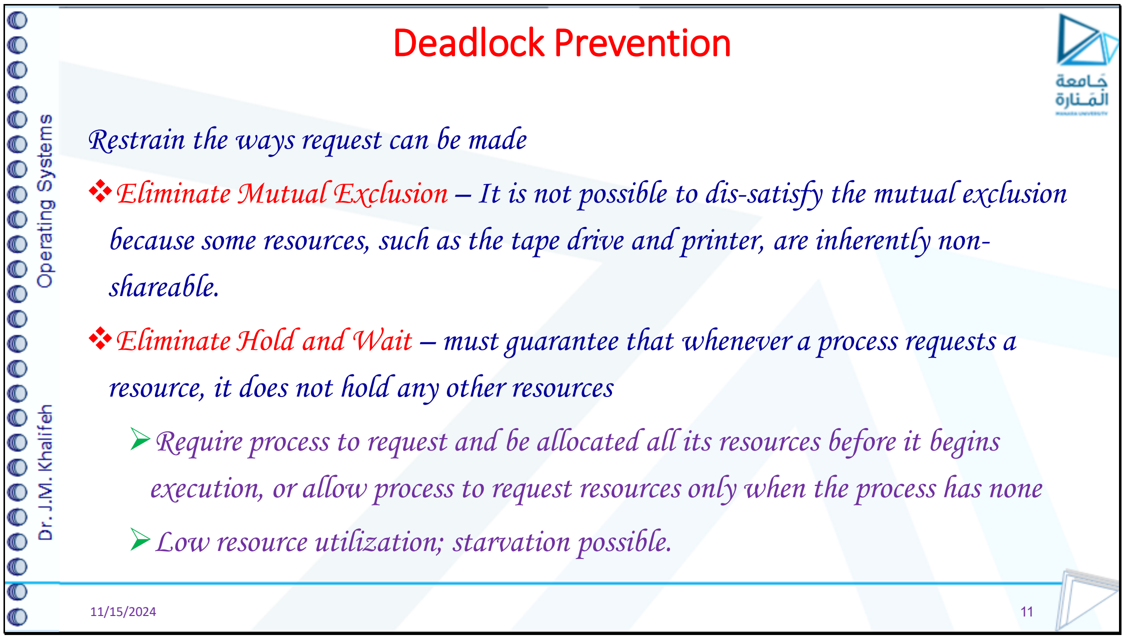
- *if only one instance per resource type, then deadlock*
- *if several instances per resource type, possibility of deadlock*

Methods for Handling Deadlocks

❖ *Ensure that the system will never enter a deadlock state*

❖ *Allow the system to enter a deadlock state and then recover*

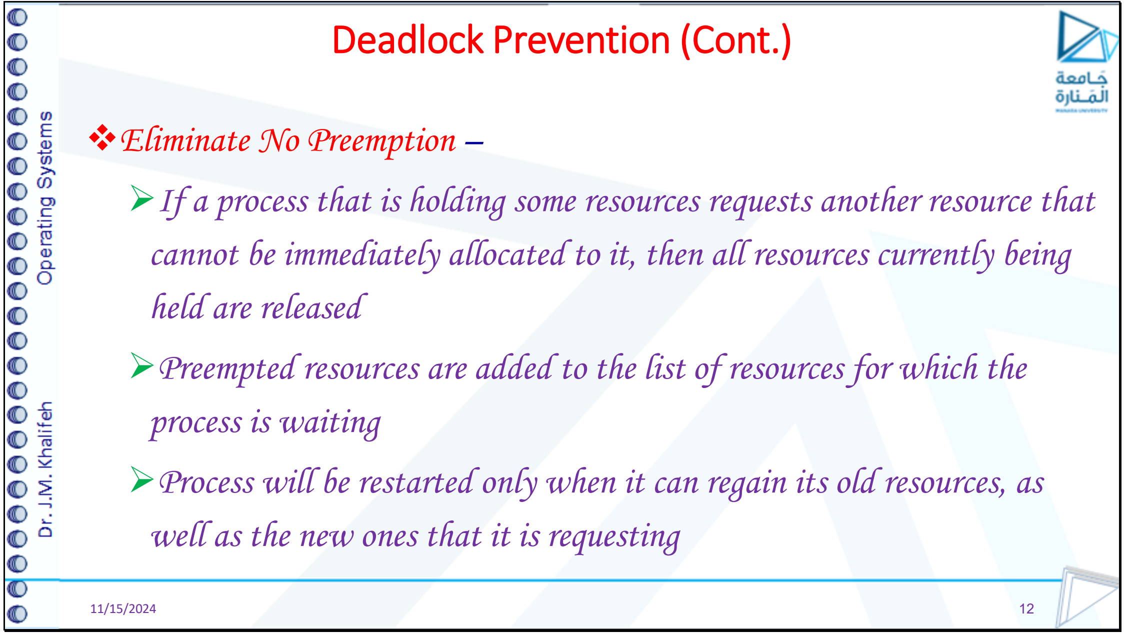
❖ *Ignore the problem and pretend that deadlocks never occur in the system; used by most operating systems, including UNIX*



Deadlock Prevention

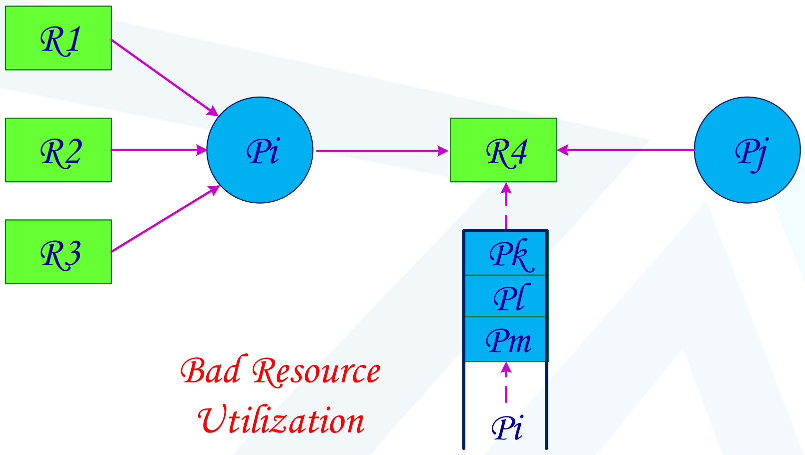
Restrain the ways request can be made

- *Eliminate Mutual Exclusion – It is not possible to dis-satisfy the mutual exclusion because some resources, such as the tape drive and printer, are inherently non-shareable.*
- *Eliminate Hold and Wait – must guarantee that whenever a process requests a resource, it does not hold any other resources*
  - *Require process to request and be allocated all its resources before it begins execution, or allow process to request resources only when the process has none*
  - *Low resource utilization; starvation possible.*



Deadlock Prevention (Cont.)

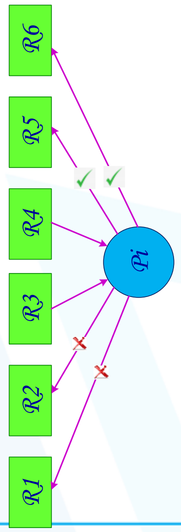
- *Eliminate No Preemption –*
  - *If a process that is holding some resources requests another resource that cannot be immediately allocated to it, then all resources currently being held are released*
  - *Preempted resources are added to the list of resources for which the process is waiting*
  - *Process will be restarted only when it can regain its old resources, as well as the new ones that it is requesting*

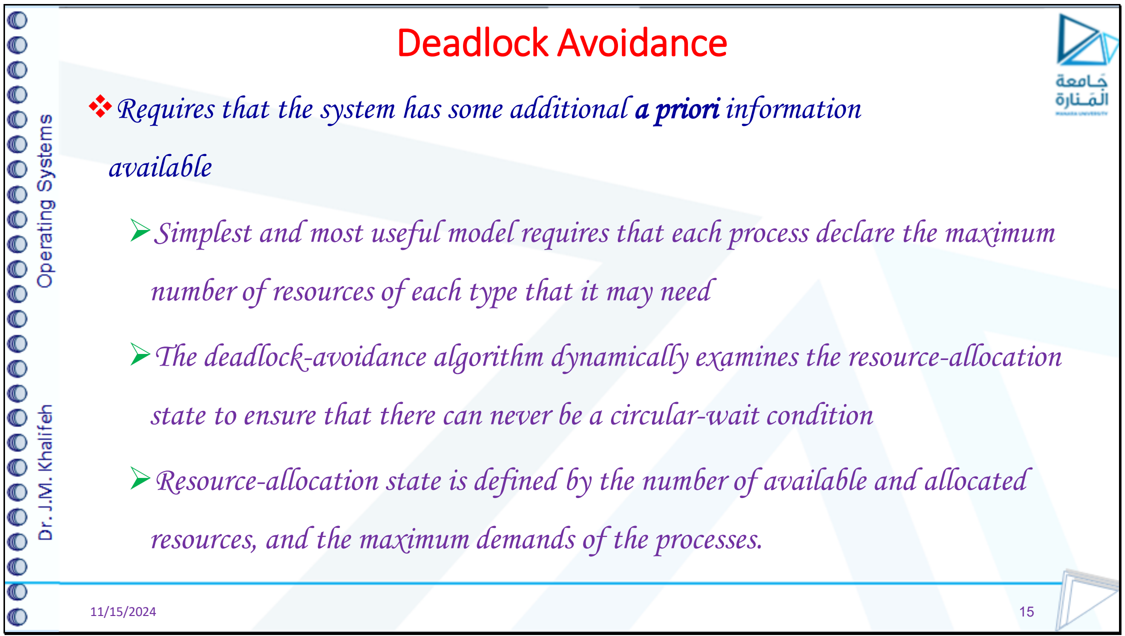


Bad Resource Utilization

Deadlock Prevention (Cont.)

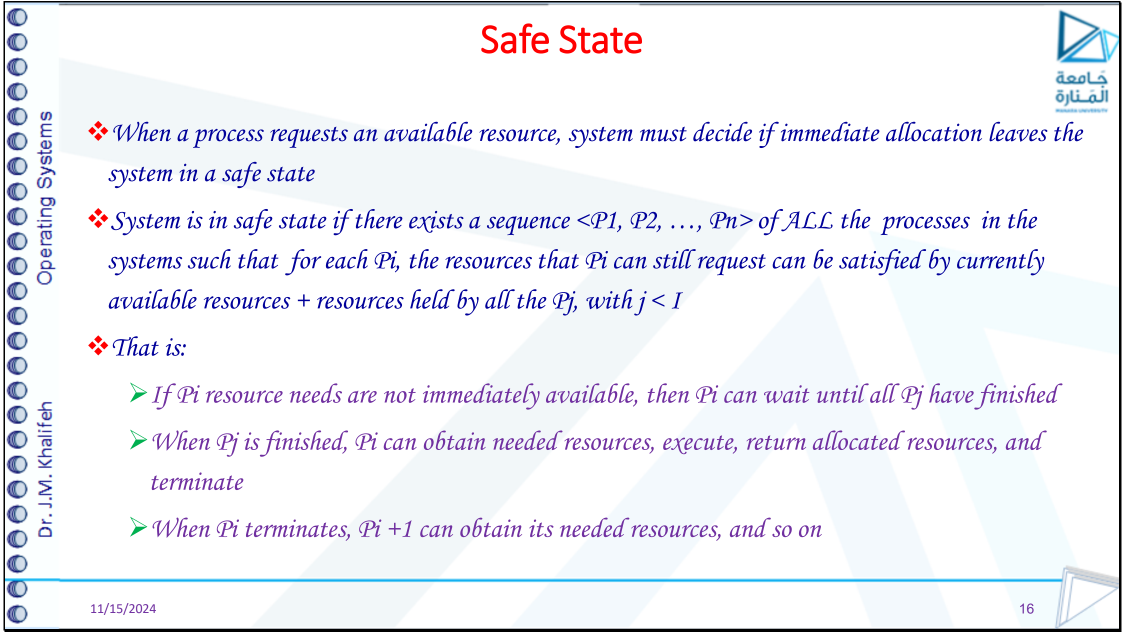
- *Eliminate Circular Wait – impose a total ordering of all resource types, and require that each process requests resources in an increasing order of enumeration.*
  - *Each resource will be assigned a numerical number. A process can request the resources to increase/decrease. order of numbering. For Example, if the P1 process is allocated R3 and R4 resources, now next time if P1 asks for R2, R1 lesser than R3 such a request will not be granted, only a request for resources more than R4 will be granted.*





Deadlock Avoidance

- *Requires that the system has some additional* ***a priori*** *information available*
  - *Simplest and most useful model requires that each process declare the maximum number of resources of each type that it may need*
  - *The deadlock-avoidance algorithm dynamically examines the resource-allocation state to ensure that there can never be a circular-wait condition*
  - *Resource-allocation state is defined by the number of available and allocated resources, and the maximum demands of the processes.*



Safe State

- *When a process requests an available resource, system must decide if immediate allocation leaves the system in a safe state*
- *System is in safe state if there exists a sequence <P1, P2, …, Pn> of ALL the processes in the systems such that for each Pi, the resources that Pi can still request can be satisfied by currently available resources + resources held by all the Pj, with j < I*
- *That is:*
  - *If Pi resource needs are not immediately available, then Pi can wait until all Pj have finished*
  - *When Pj is finished, Pi can obtain needed resources, execute, return allocated resources, and terminate*
  - *When Pi terminates, Pi +1 can obtain its needed resources, and so on*

Basic Facts

- ❖ *If a system is in safe state ⇒ no deadlocks*
- ❖ *If a system is in unsafe state ⇒ possibility of deadlock*
- ❖ *Avoidance ⇒ ensure that a system will never enter an unsafe state.*

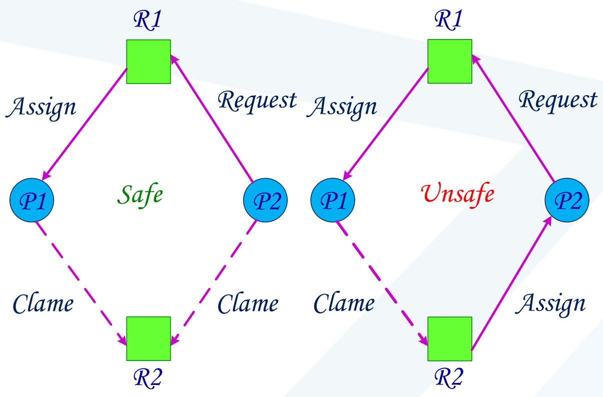
Avoidance algorithms

- ❖ *Single instance of a resource type*
  - ➢ *Use a resource-allocation graph*
- ❖ *Multiple instances of a resource type*
  - ➢ *Use the banker’ s algorithm*

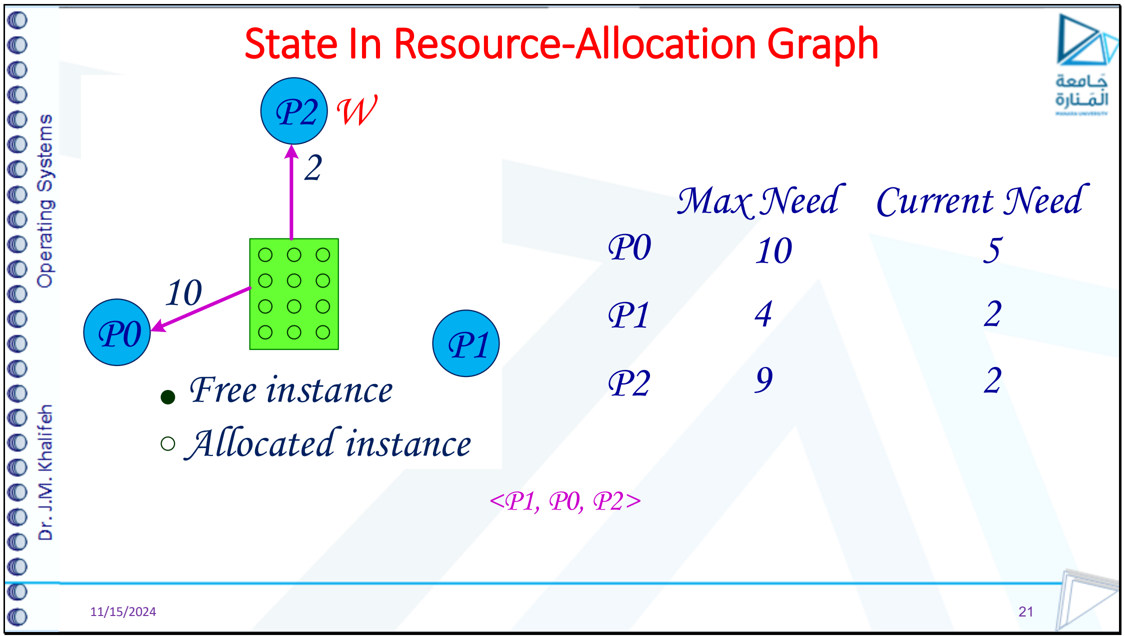
Resource-Allocation Graph Scheme

- ❖ *Claim edge Pi → Rj indicated that process Pj may request resource Rj; represented by a dashed line*
- ❖ *Claim edge converts to request edge when a process requests a resource*
- ❖ *Request edge converted to an assignment edge when the resource is allocated to the process*
- ❖ *When a resource is released by a process, assignment edge reconverts to a claim edge*
- ❖ *Resources must be claimed a priori in the system*

Resource-Allocation Graph



The resource allocation graph (RAG) is used to visualize the system's current state as a graph. The Graph includes all processes, the resources that are assigned to them, as well as the resources that each Process requests. If there are fewer processes, we can quickly spot a deadlock in the system by looking at the graph rather than the tables we use in Banker's algorithm.



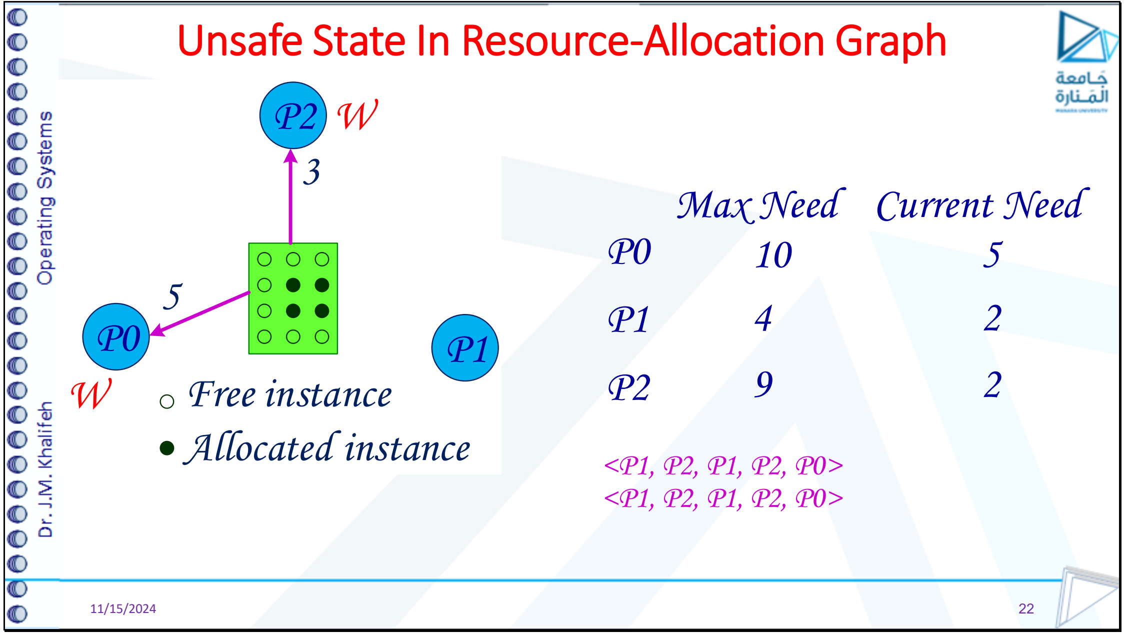
State In Resource-Allocation Graph

● Free instance

○ Allocated instance

	Max Need	Current Need
P0	10	5
P1	4	2
P2	9	2

<P1, P0, P2>



Unsafe State In Resource-Allocation Graph

○ Free instance

● Allocated instance

	Max Need	Current Need
P0	10	5
P1	4	2
P2	9	2

<P1, P2, P1, P2, P0>
<P1, P2, P1, P2, P0>

Resource-Allocation Graph Algorithm

- *Suppose that process Pi requests a resource Rj*
- *The request can be granted only if converting the request edge to an assignment edge does not result in the formation of a cycle in the resource allocation graph*

Banker's Algorithm

Creditors

A	*B*	*C*
x	*y*	*z*

SUM *Cash*

?

Debtor applies for a loan: W=100

Cash-SUM>=W

Banker’s Algorithm

- *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*

Data Structures for the Banker’s Algorithm

Available :*It is 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 Rj

$$Available\,[\,j\,]=\{\,4,\,6,\,2,\,1,\,7\}$$

Data Structures for the Banker’s Algorithm

Resource

$$\text{Process}\quad \begin{array}{c|cccccc|} & R1 & R2 & R3 & R4 & & Ri \\ P1 & 2 & 0 & 5 & 3 & \cdot & 1 \\ P2 & 3 & 2 & 6 & 0 & \vdots & 2 \\ P3 & 1 & 1 & 4 & 3 & \vdots & 7 \\ P4 & 0 & 2 & 2 & 3 & \vdots & 2 \\ & \cdot & \cdot & \cdot & \cdot & \cdot & \\ Pi & 1 & 3 & 2 & 6 & \cdot & 4 \end{array}$$

Max[i, j] = k means process Pi may request at most ‘k’ instances of resource type Rj.

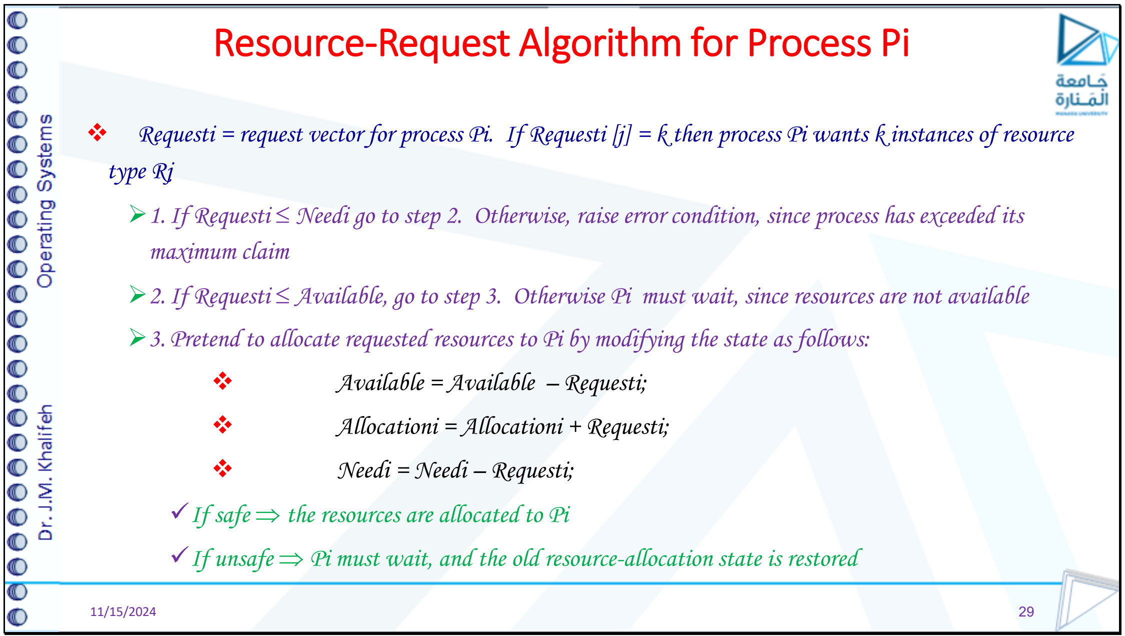
Allocation[i, j] = k means process Pi is currently allocated ‘k’ instances of resource type Rj

Need [i, j] = k means process Pi currently needs ‘k’ instances of resource type Rj

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

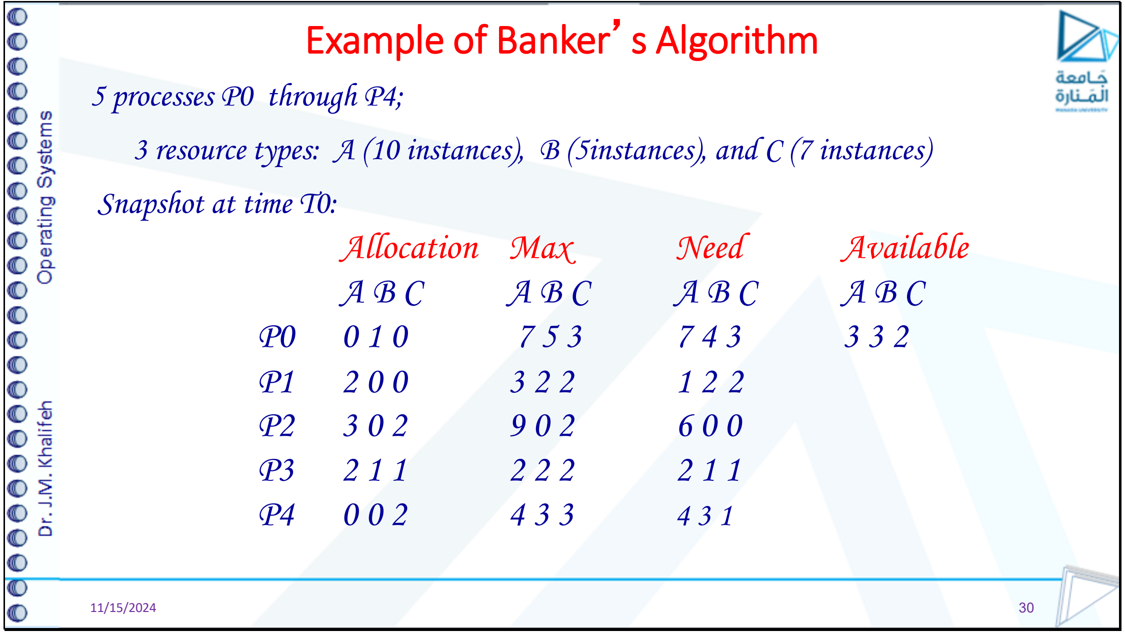
Safety Algorithm

1. *Let Work and Finish be vectors of length m and n, respectively. Initialize:*
   - ➢ *Work = Available*
   - ➢ *Finish [i] = false for i = 0, 1, …, n- 1*
2. *Find an i such that both:*
   - ➢ *(a) Finish [i] = false*
   - ➢ *(b) Needi ≤ Work*
   - ➢ *If no such i exists, go to step 4*
3. *Work = Work + Allocationi*
   - ➢ *Finish[i] = true*
   - ➢ *go to step 2*
4. *If Finish [i] == true for all i, then the system is in a safe state*



Resource-Request Algorithm for Process Pi

- Requesti = request vector for process Pi. If Requesti [j] = k then process Pi wants k instances of resource type Rj
  - 1. If Requesti ≤ Needi go to step 2. Otherwise, raise error condition, since process has exceeded its maximum claim
  - 2. If Requesti ≤ Available, go to step 3. Otherwise Pi must wait, since resources are not available
  - 3. Pretend to allocate requested resources to Pi by modifying the state as follows:
    - Available = Available – Requesti;
    - Allocationi = Allocationi + Requesti;
    - Needi = Needi – Requesti;
  - ✓ If safe ⟹ the resources are allocated to Pi
  - ✓ If unsafe ⟹ Pi must wait, and the old resource-allocation state is restored



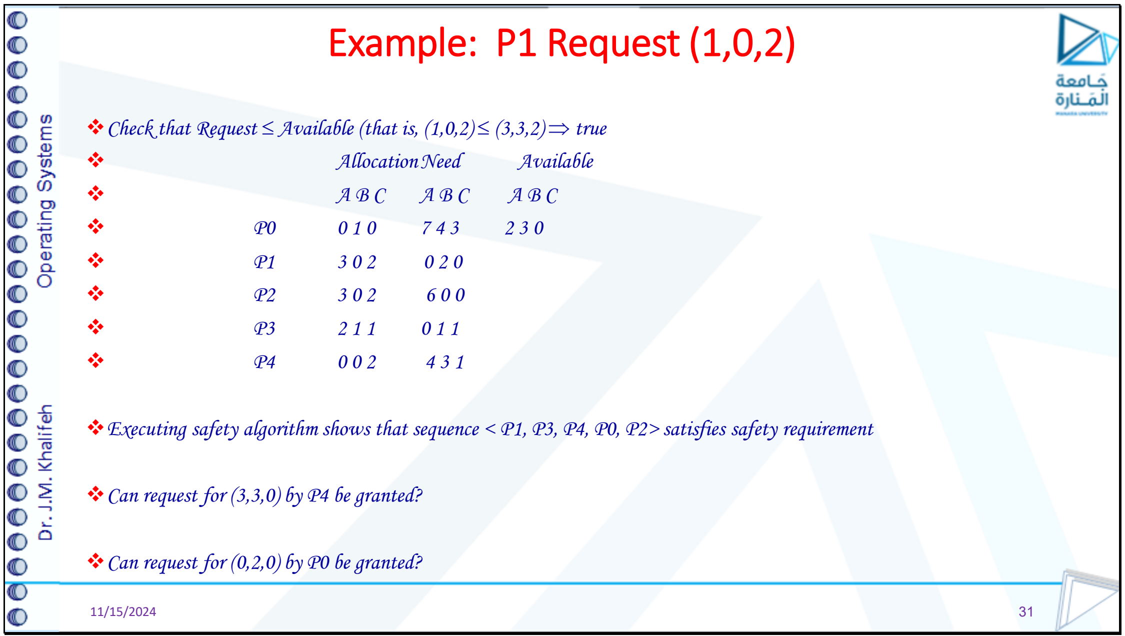
Example of Banker's Algorithm

5 processes P0 through P4;

3 resource types: A (10 instances), B (5instances), and C (7 instances)

Snapshot at time T0:

	Allocation	Max	Need	Available
	A B C	A B C	A B C	A B C
P0	0 1 0	7 5 3	7 4 3	3 3 2
P1	2 0 0	3 2 2	1 2 2	
P2	3 0 2	9 0 2	6 0 0	
P3	2 1 1	2 2 2	2 1 1	
P4	0 0 2	4 3 3	4 3 1	

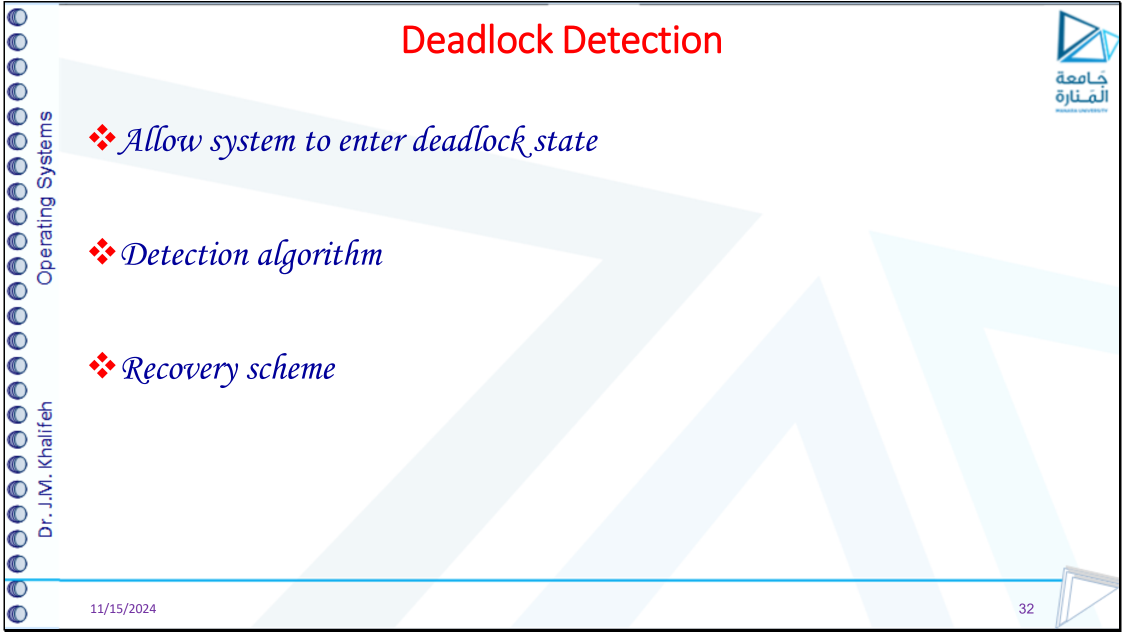


Example: P1 Request (1,0,2)

- Check that Request ≤ Available (that is, $(1,0,2) \le (3,3,2) \Rightarrow$ true

	Allocation	Need	Available
	A B C	A B C	A B C
P0	0 1 0	7 4 3	2 3 0
P1	3 0 2	0 2 0	
P2	3 0 2	6 0 0	
P3	2 1 1	0 1 1	
P4	0 0 2	4 3 1	

- Executing safety algorithm shows that sequence < P1, P3, P4, P0, P2> satisfies safety requirement
- Can request for (3,3,0) by P4 be granted?
- Can request for (0,2,0) by P0 be granted?



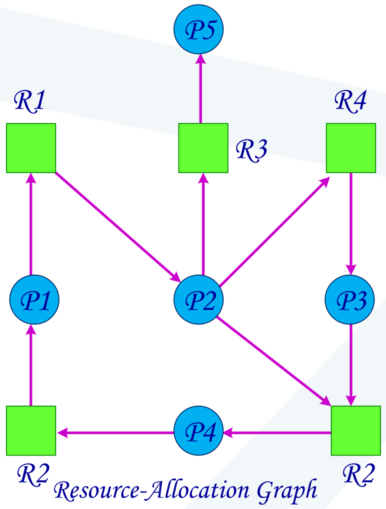
Deadlock Detection

- Allow system to enter deadlock state
- Detection algorithm
- Recovery scheme

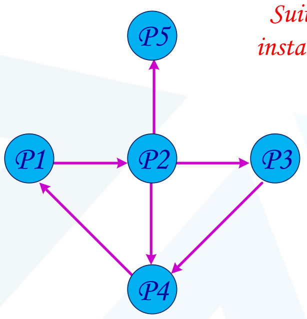
Single Instance of Each Resource Type

- ❖ *Maintain wait-for graph*
  - ➢ *Nodes are processes*
  - ➢ *Pi → Pj if Pi is waiting for Pj*
- ❖ *Periodically invoke an algorithm that searches for a cycle in the graph. If there is a cycle, there exists a deadlock*
- ❖ *An algorithm to detect a cycle in a graph requires an order of n2 operations, where n is the number of vertices in the graph*

Resource-Allocation Graph and Wait-for Graph



Resource-Allocation Graph



Suitable for Single instance of a resource type

Corresponding wait-for graph

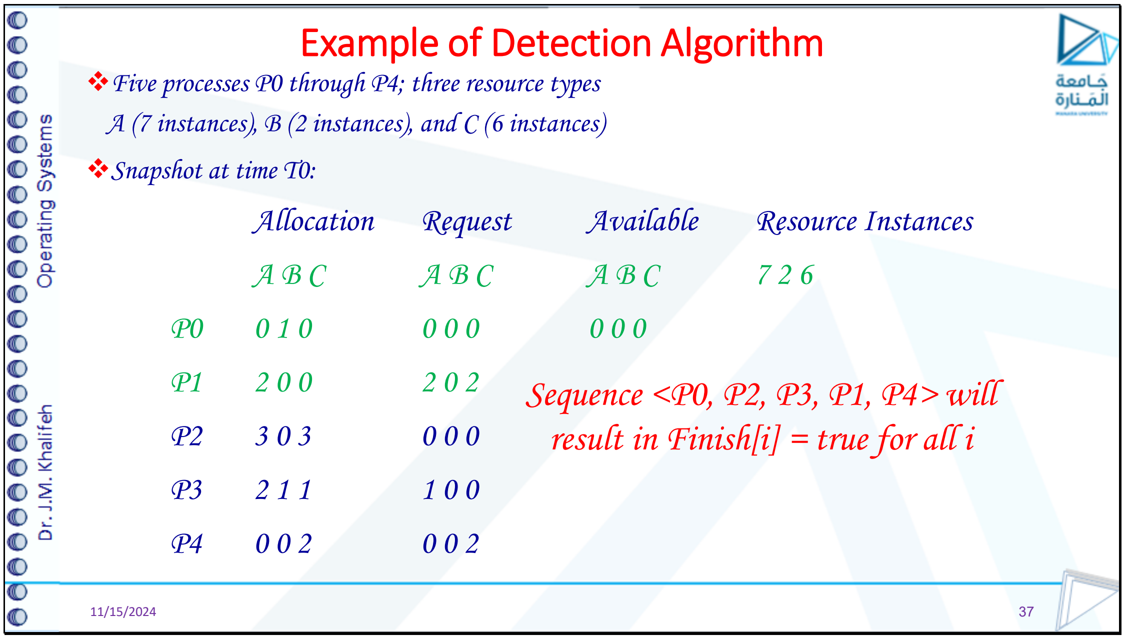
Several Instances of a Resource Type

The algorithm employs several times varying data structures:

- ❖ ***Available:*** *A vector of length m indicates the number of available resources of each type.*
- ❖ ***Allocation:*** *An n*m matrix defines the number of resources of each type currently allocated to a process. The column represents resource and rows represent a process.*
- ❖ ***Request:*** *An n*m matrix indicates the current request of each process. If request[i][j] equals k then process P_i is requesting k more instances of resource type R_j.*

Detection Algorithm

- ❖ *Let Work and Finish be vectors of length m and n respectively. Initialize Work= Available.*
  For i=0, 1, …., n-1 , if $Request_i$ = 0, then Finish[i] = true; otherwise, Finish[i] = false.
- ❖ *Find an index i such that both*
  a) Finish[i] == false
  b) $Request_i$ <= Work
  If no such i exists go to step 4.
- ❖ *Work= Work+ $Allocation_i$*
  Finish[i]= true
  Go to Step 2.
- ❖ *If Finish[i]== false for some i, 0<=i<n, then the system is in a deadlocked state. Moreover, if Finish[i]==false the process P_i is deadlocked.*

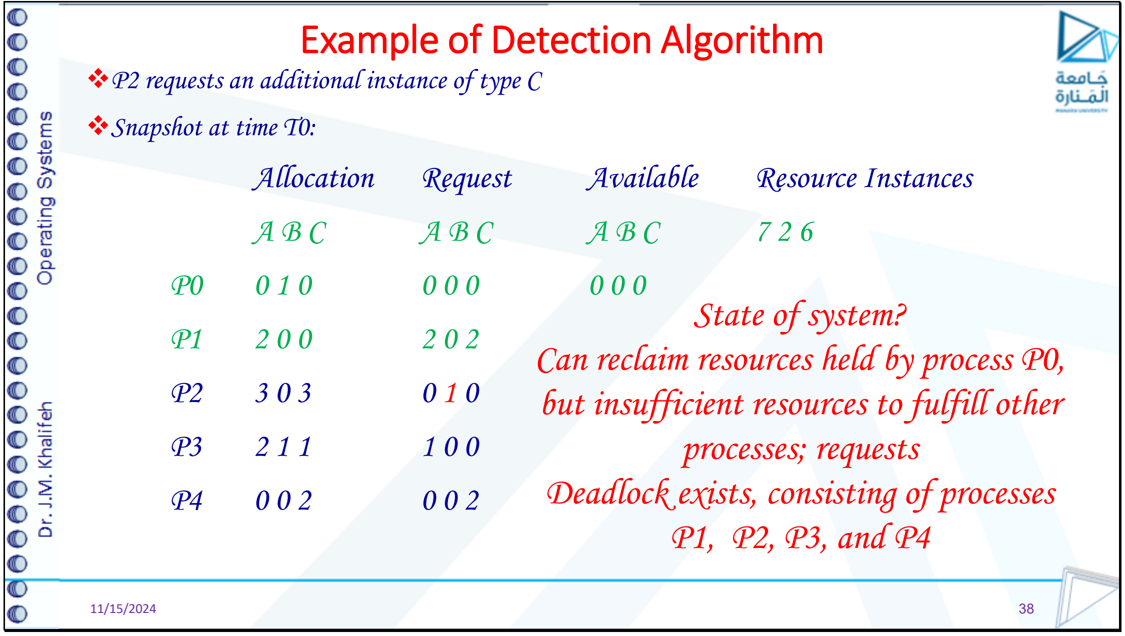


Example of Detection Algorithm

- ❖ Five processes P0 through P4; three resource types A (7 instances), B (2 instances), and C (6 instances)
- ❖ Snapshot at time T0:

	Allocation	Request	Available	Resource Instances
	A B C	A B C	A B C	7 2 6
P0	0 1 0	0 0 0	0 0 0	
P1	2 0 0	2 0 2		
P2	3 0 3	0 0 0		
P3	2 1 1	1 0 0		
P4	0 0 2	0 0 2		

Sequence <P0, P2, P3, P1, P4> will result in Finish[i] = true for all i

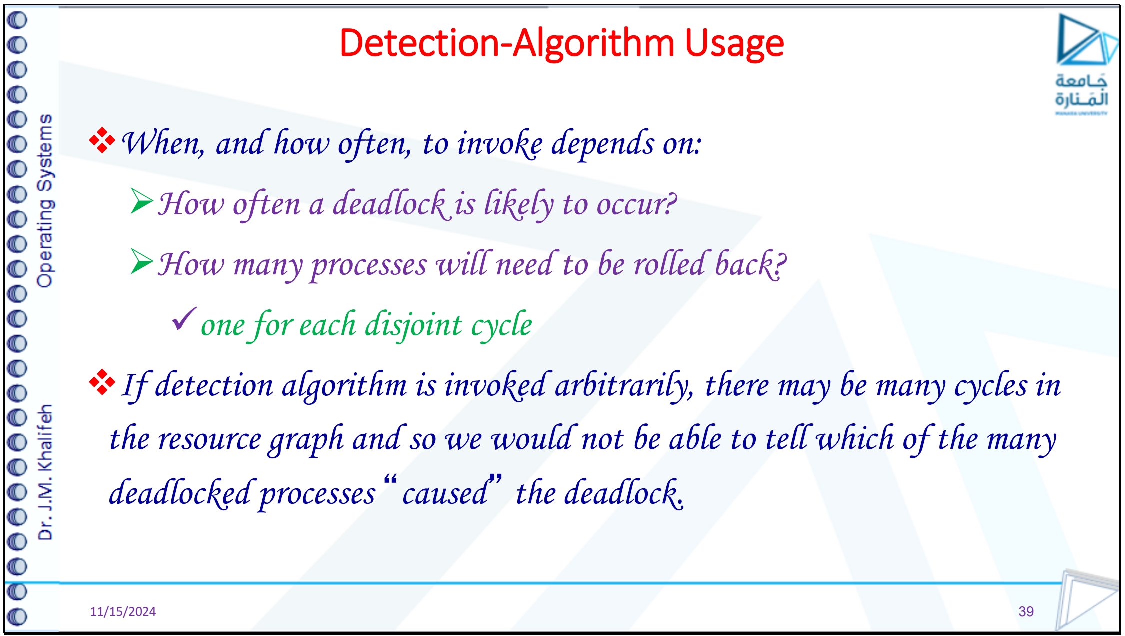


Example of Detection Algorithm

- ❖ P2 requests an additional instance of type C
- ❖ Snapshot at time T0:

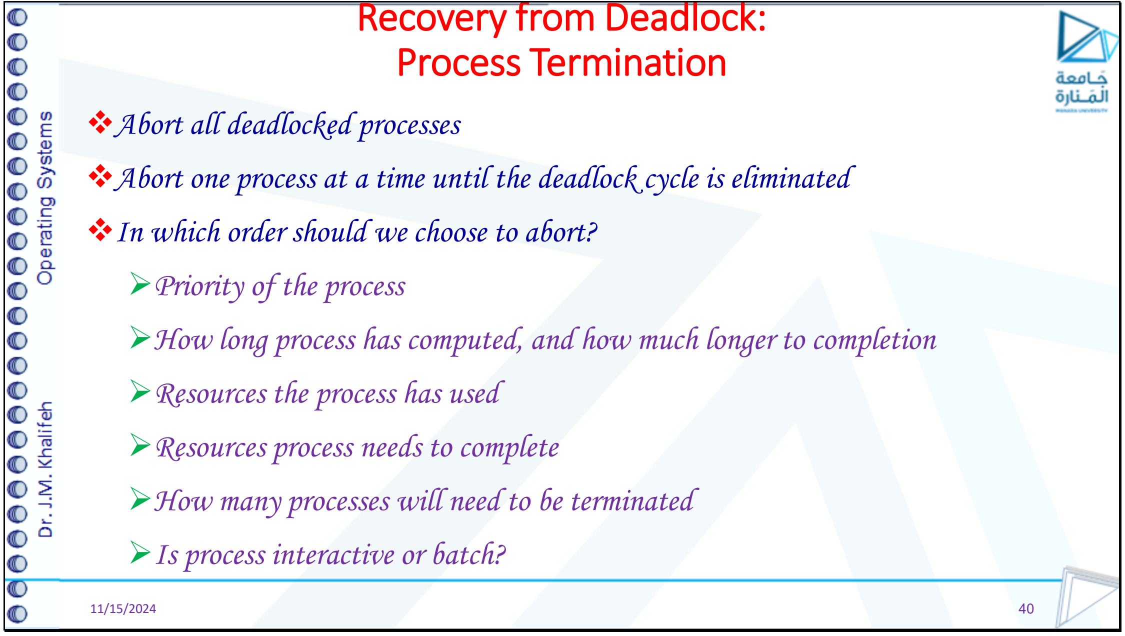
	Allocation	Request	Available	Resource Instances
	A B C	A B C	A B C	7 2 6
P0	0 1 0	0 0 0	0 0 0	
P1	2 0 0	2 0 2		
P2	3 0 3	0 1 0		
P3	2 1 1	1 0 0		
P4	0 0 2	0 0 2		

State of system?
Can reclaim resources held by process P0, but insufficient resources to fulfill other processes; requests
Deadlock exists, consisting of processes P1, P2, P3, and P4



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.*



Recovery from Deadlock:
Process Termination

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

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*