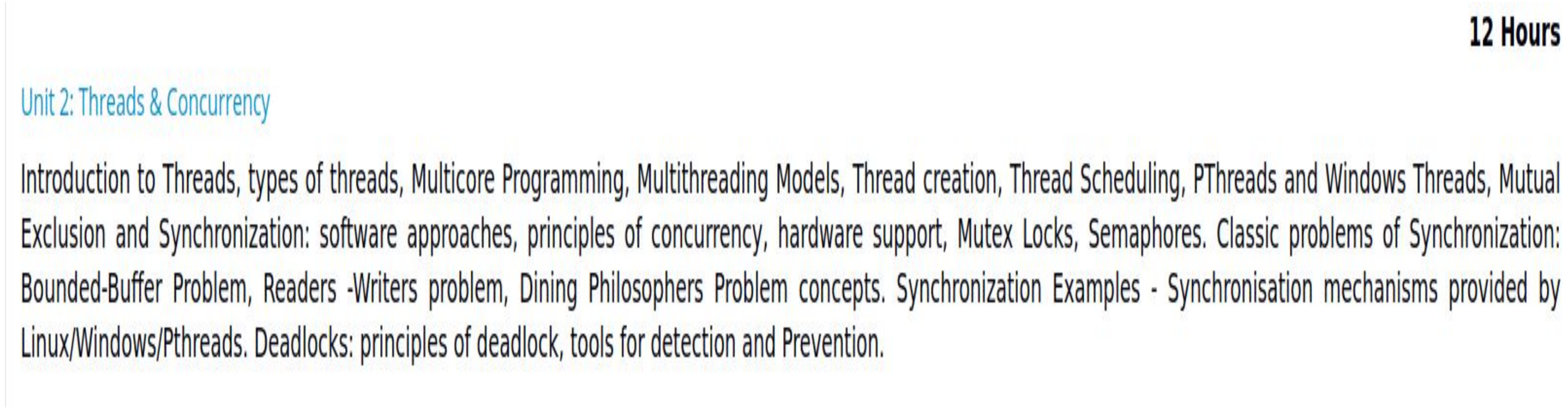
OPERATING SYSTEMS

Deadlocks - 2

Nitin V Pujari
Faculty, Computer Science
Dean - IQAC, PES University

OPERATING SYSTEMS

Course Syllabus - Unit 2



12 Hours

Unit 2: Threads & Concurrency

Introduction to Threads, types of threads, Multicore Programming, Multithreading Models, Thread creation, Thread Scheduling, PThreads and Windows Threads, Mutual Exclusion and Synchronization: software approaches, principles of concurrency, hardware support, Mutex Locks, Semaphores. Classic problems of Synchronization: Bounded-Buffer Problem, Readers -Writers problem, Dining Philosophers Problem concepts. Synchronization Examples - Synchronisation mechanisms provided by Linux/Windows/Pthreads. Deadlocks: principles of deadlock, tools for detection and Prevention.

OPERATING SYSTEMS

Course Outline

13	Introduction to Threads, types of threads, Multicore Programming, Multithreading Models	4.1 - 4.3	42.8
14	Thread creation, Thread Scheduling	5.4	
15	Pthreads and Windows Threads	4.4	
16	Mutual Exclusion and Synchronization: software approaches,	6.1-6.2	
17	principles of concurrency, hardware support	6.3-6.4	
18	Mutex Locks, Semaphores	6.5, 6.6	
19	Classic problems of Synchronization: Bounded-Buffer Problem, Readers-Writers problem	6.7-6.8	
20	Dining-Philosophers Problem	6.8	
21	Synchronization Examples: Synchronisation mechanisms provided by Linux/Windows/Pthreads.	6.9	
22	Deadlocks: principles of deadlock, Deadlock Characterization	7.1-7.3	
23	Deadlock Prevention, Deadlock example	7.4-7.5	
24	Deadlock Detection, Algorithm	7.6	

Topic Outline

- RAG - Deadlock Detection and Recovery
- Deadlock Avoidance & Deadlock Detection
- Deadlock recovery

Deadlocks: RAG - Deadlock Detection and Recovery

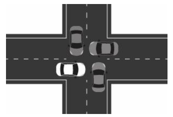
- Allow system to enter deadlock state
- Detection algorithm
- Recovery scheme

Deadlocks: RAG - Deadlock Detection and Recovery

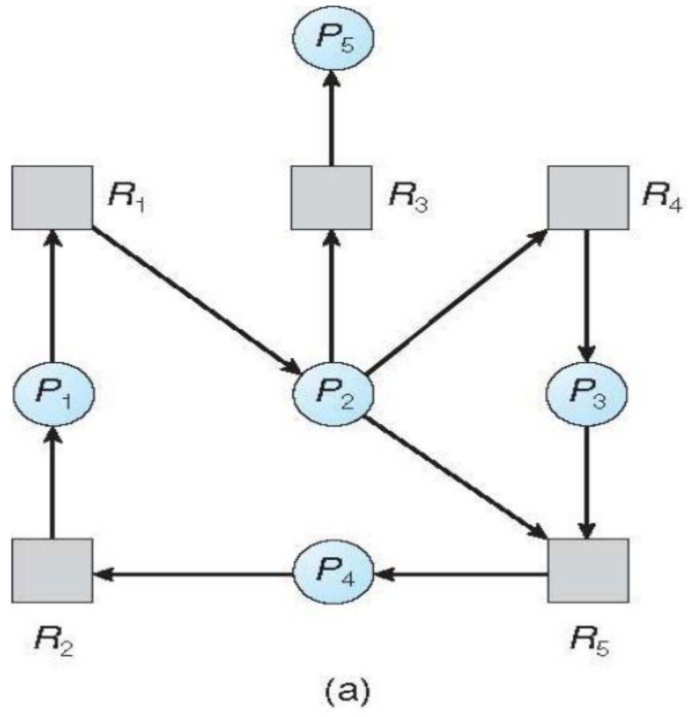
- Maintain wait-for graph
  - Nodes are processes
  - Pi ⟶ Pj if Pi is waiting for Pj

- Periodically invoke an algorithm that searches for a cycle in the graph. If there is a cycle, there exists a deadlock

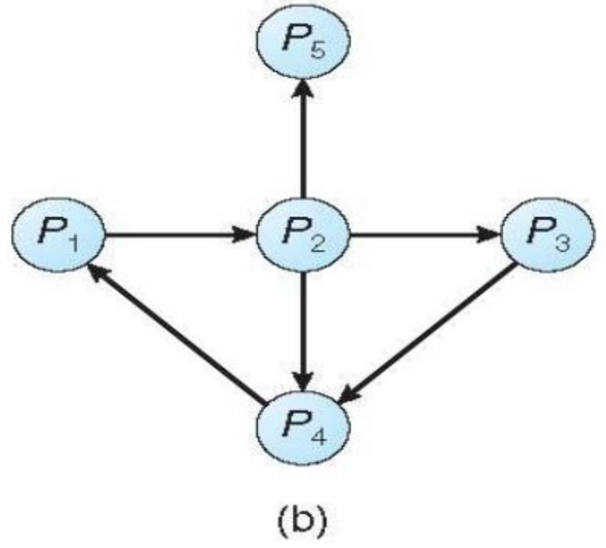
- An algorithm to detect a cycle in a graph requires an order of n**2 operations, where n is the number of vertices in the graph



Deadlocks: RAG - Deadlock Detection and Recovery



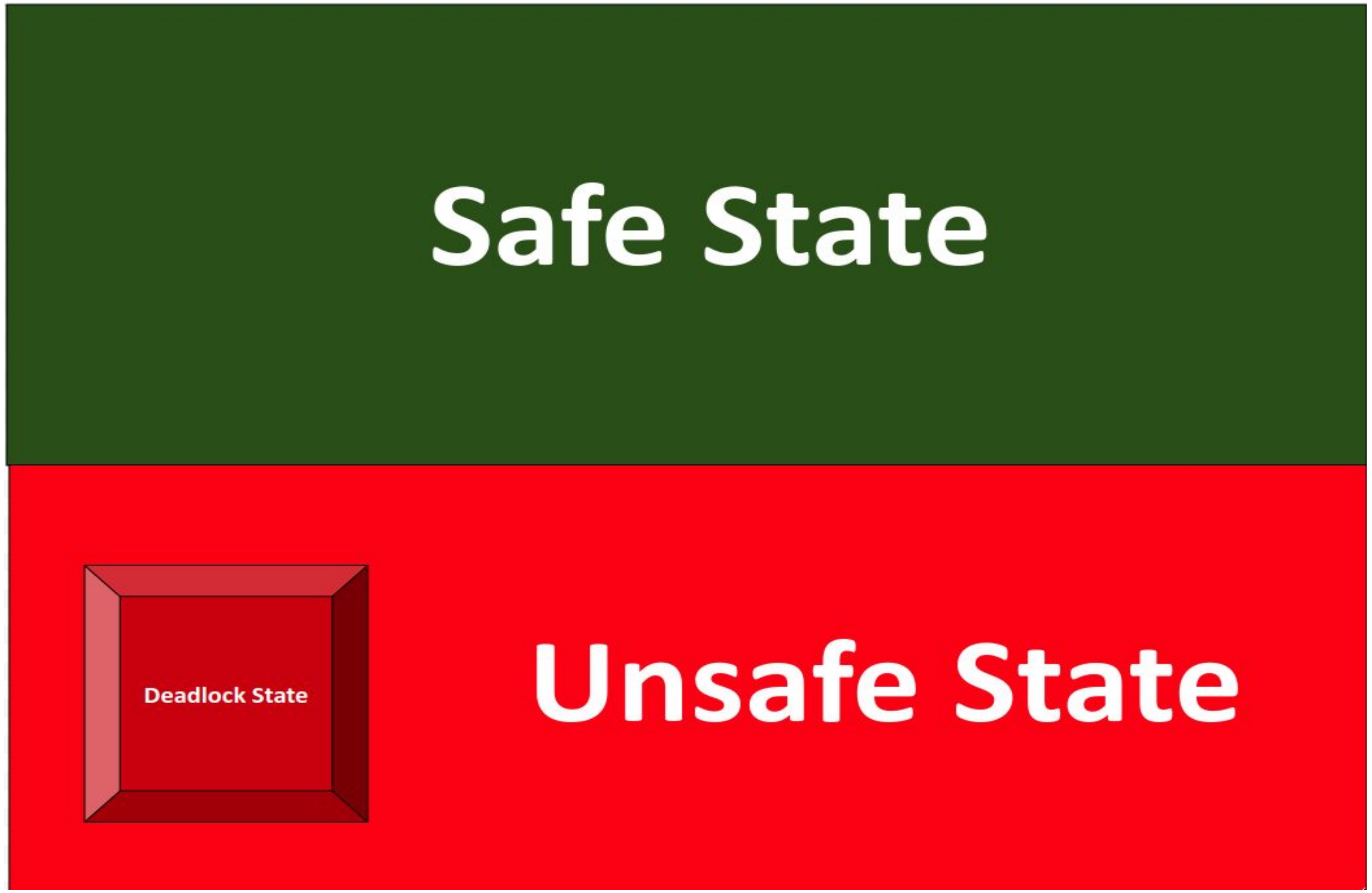
Resource-Allocation Graph



Corresponding wait-for graph

Deadlocks: Deadlock Avoidance & Detection

Safe and Unsafe State



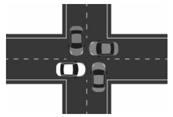
Deadlocks: Deadlock Avoidance & Detection

Safe State

- **Safe state:** A safe state is a state in which all the processes can be executed in some arbitrary order with the available resources such that no deadlock occurs.

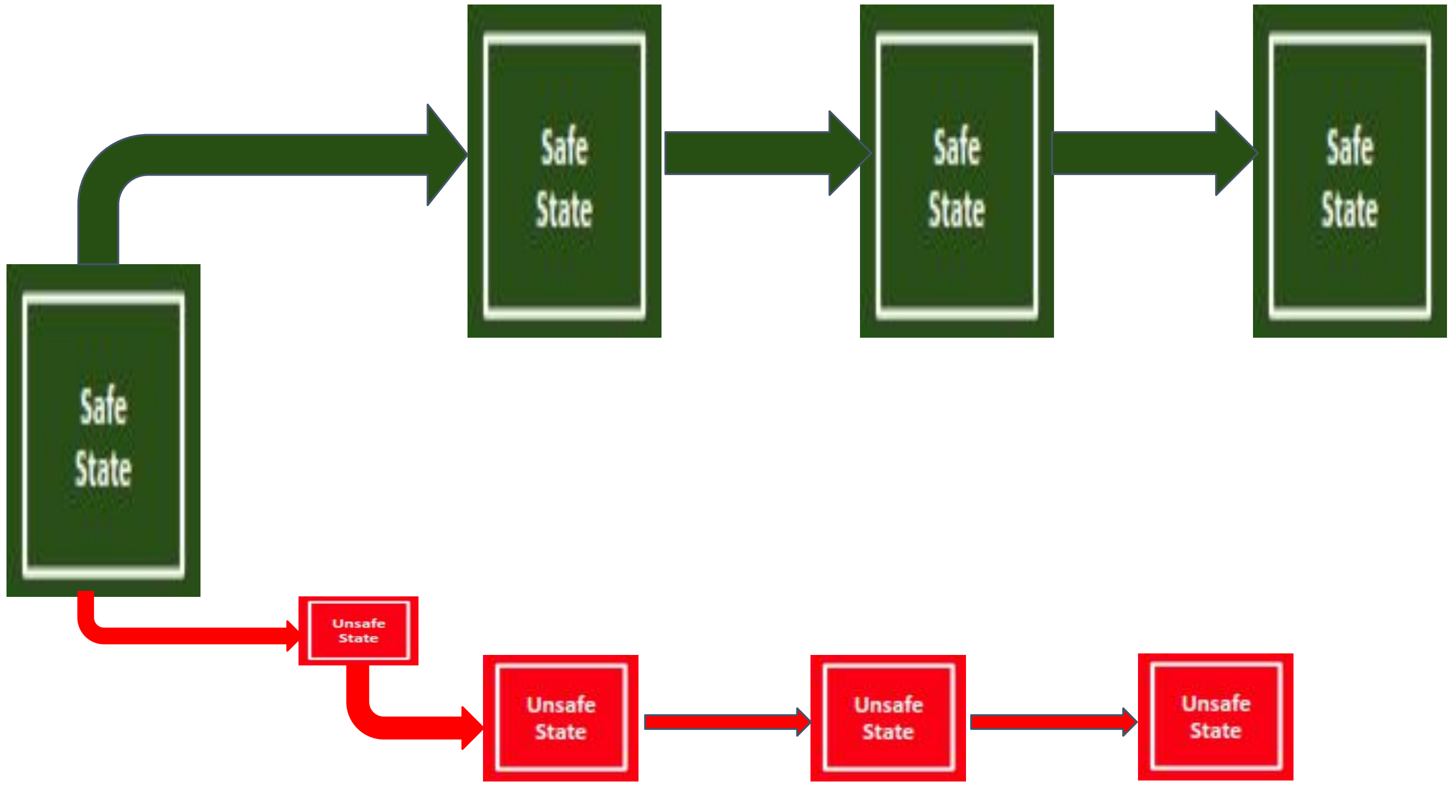
1. If it is a safe state, then the requested resources are allocated to the process in actual.

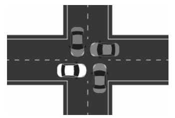
2. If the resulting state is an unsafe state then it rollbacks to the previous state and the process is asked to wait longer.



Deadlocks: Deadlock Avoidance & Detection

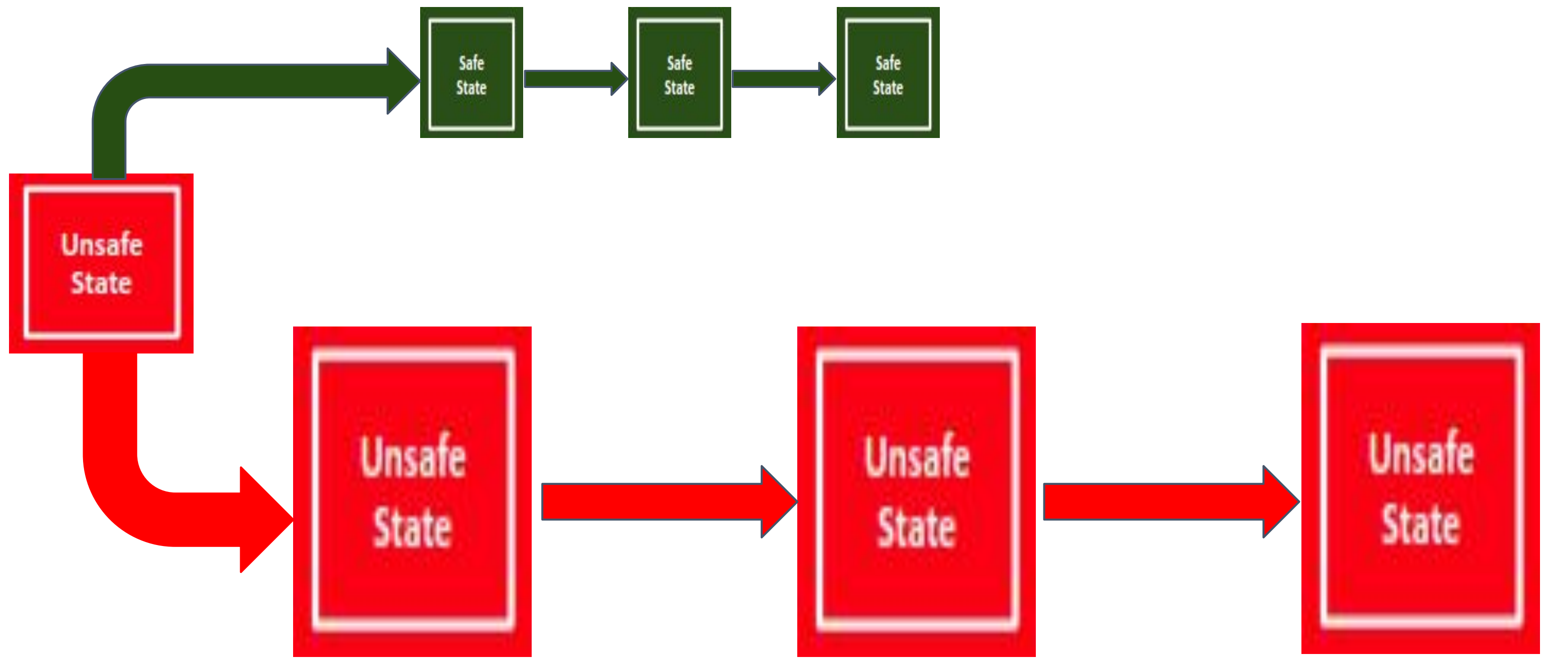
Safe to Unsafe State

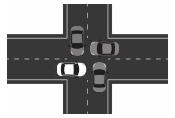




Deadlocks: Deadlock Avoidance & Detection

Unsafe State to Safe State

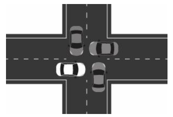




Deadlocks: Deadlock Avoidance & Detection

Banker's Algorithm

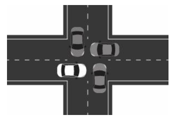
- Banker’s Algorithm is a **Deadlock Avoidance algorithm**.
- It is also used for **Deadlock Detection**.
- This algorithm tells that if any system can go into a deadlock or not by analyzing the currently allocated resources and the resources required by it in the future.
- This algorithm can be used when several instance of resources are present, which is typically cannot be handled by RAG
- There are various data structures which are used to implement this algorithm



Deadlocks: Deadlock Avoidance & Detection

Banker's Algorithm

- Multiple instances.
- Each process must apriori 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.

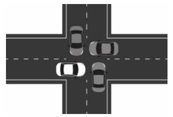


Banker’s Algorithm

The Bankers Algorithm consists of the following two algorithms

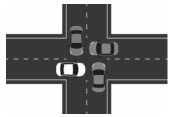
1. Safety Algorithm

2. Request-Resource Algorithm



Deadlocks: Deadlock Avoidance & Detection

- **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.



Deadlocks: Deadlock Avoidance & Detection

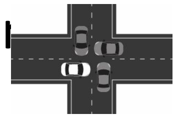
Data Structures used to implement Banker's Algorithm

- **Available:** It is a 1-D matrix that tells the number of each resource type (instance of resource type) currently available.

  Example: Available[R1]= A, means that there are A instances of R1 resources are currently available.

- **Max:** It is a 2-D matrix that tells the maximum number of each resource type required by a process for successful execution.

  Example: Max[P1][R1] = A, specifies that the process P1 needs a maximum of **A** instances of resource R1 for complete executio



Deadlocks: Deadlock Avoidance & Detection

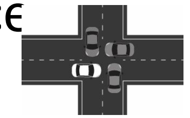
Data Structures used to implement Banker's Algorithm

- **Allocation:** It is a 2-D matrix that tells the number of types of each resource type that has been allocated to the process.

  Example: Allocation[P1][R1] = A, means that **A** instances of resource type R1 have been allocated to the process P1.

- **Need:** It is a 2-D matrix that tells the number of remaining instances of each resource type required for execution.

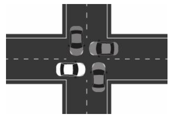
  Example: Need[P1][R1]= A tells that **A** instances of R1 resource type are required for the execution of process P1.



Deadlocks: Deadlock Avoidance & Detection

Data Structures used to implement Banker's Algorithm

Need[i][j]= Max[i][j] - Allocation[i][j], where i corresponds any process P(i) and j corresponds to any resource type R(j)



Deadlocks: Deadlock Avoidance & Detection

RMax		
A	B	C
10	5	7

Available=>Rmax-Allocated		
A	B	C
3	3	2

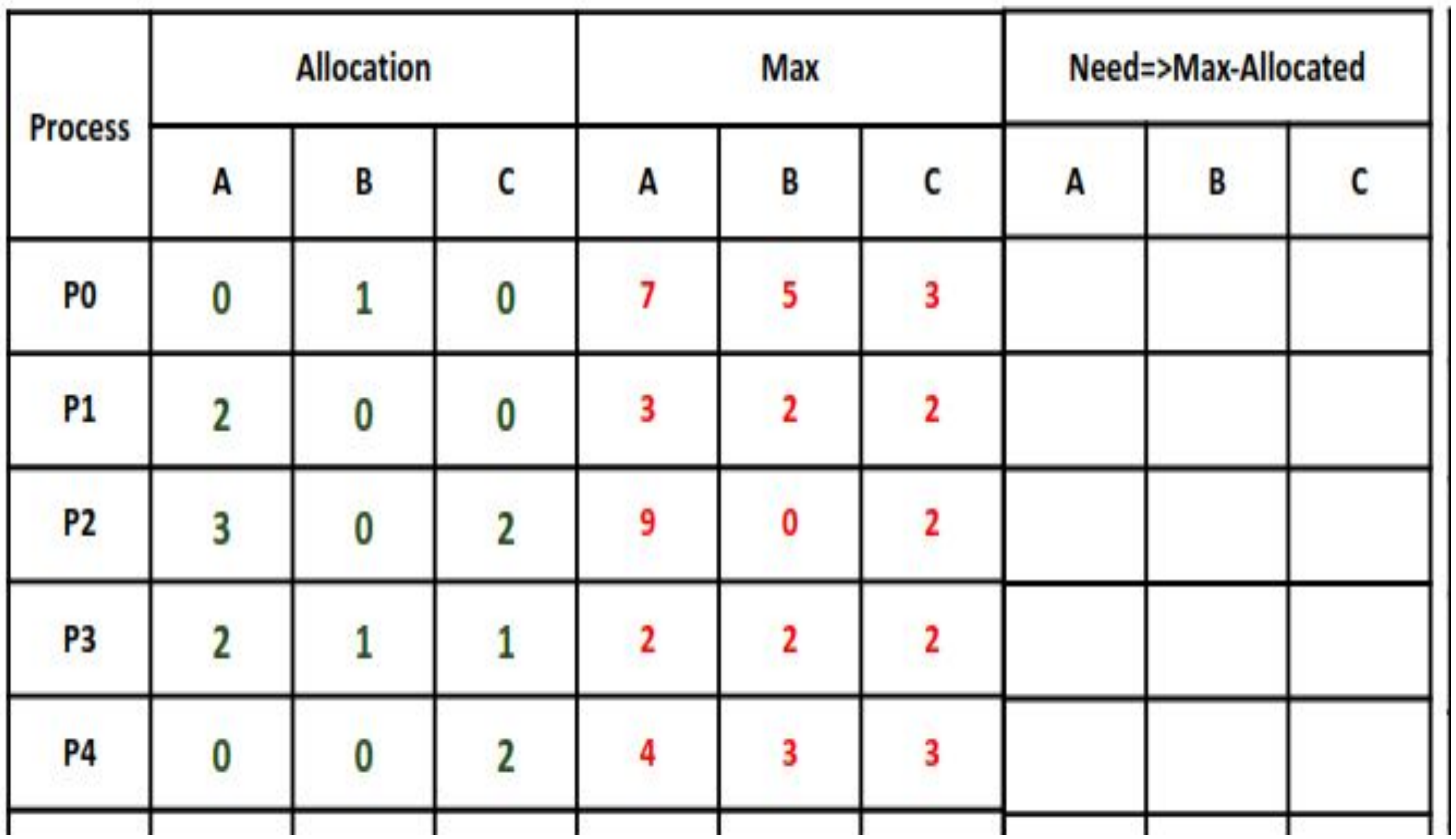
work<= Available => (Rmax-Allocated)		
A	B	C
3	3	2

i	Not Initialised

Process	Flag
P0	False
P1	False
P2	False
P3	False
P4	False

Safe Sequence				

Deadlocks: Deadlock Avoidance & Detection



Process	Allocation			Max			Need=>Max-Allocated		
	A	B	C	A	B	C	A	B	C
P0	0	1	0	7	5	3			
P1	2	0	0	3	2	2			
P2	3	0	2	9	0	2			
P3	2	1	1	2	2	2			
P4	0	0	2	4	3	3			

Deadlocks: Deadlock Avoidance & Detection

Banker's Algorithm: Safety Algorithm

Step1: Let Work and Finish be vectors of length m and n, respectively.

Initialize: Work = Available

Finish [i] =false for i= 0,1,2,..n-1.

Step2: Find an i such that both:

(a) Finish[i] = false

(b) $Need_i \leq Work$

If no such i exists, go to step 4.

Step 3: Work= Work + $Allocation_i$

Finish[i] =true

go to step 2.

Step 4: If Finish[i] == true for all i, then the system is in a safe state.

Deadlocks: Deadlock Avoidance & Detection

Banker's Algorithm: Resource Request 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***$_i \neq$ **0**, then ***Finish*[i] = *false***; otherwise, ***Finish*[i] = *true***

2. Find an index ***i*** such that both:

   (a) ***Finish*[*i*] == *false***

   (b) ***Request***$_i \leq$ ***Work***

   If no such ***i*** exists, go to step 4

Deadlocks: Deadlock Avoidance & Detection

Banker's Algorithm: Resource Request Algorithm

3. $Work = Work + Allocation_i$
   $Finish[i] = true$
   go to step 2

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

Algorithm requires an order of $O(m \times n^2)$ operations to detect whether the system is in deadlocked state

Banker's Safety Algorithm: Detailed Four Steps

The safety algorithm is applied to check whether a state is in a safe state or not.

This algorithm involves the following four steps:

Step 1: Suppose currently all the processes are to be executed. Define two data structures as work and finish as vectors of length m(where m is the length of **Available** vector)and n(is the **number of processes** to be executed).

Work = Available

Finish[i] =false for i = 0, 1, … , n-1.

Step 2: This algorithm will look for a process that has **Need** value less than or equal to the **Work**. So, in this step, we will find an index i such that

Finish[i] ==false &&

Need[i]<= Work

If no such 'i' is present then go to **Step 4** else to **Step 3.**

Deadlocks: Deadlock Avoidance & Detection

Banker's Safety Algorithm: Detailed Four Steps

Step 3: The process '**i**' selected in the above step runs and finishes its execution. Also, the resources allocated to it gets free. The resources which get free are added to the Work and Finish(i) of the process is set as true. The following operations are performed:

Work = Work + Allocation

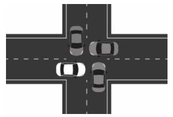
Finish[i] = true

After performing the 3rd step go to step 2.

Step 4: If all the processes are executed in some sequence then it is said to be a safe state. Or, we can say that if

Finish[i]==true for all i,

then the system is said to be in a **Safe State**.



Deadlocks: Deadlock Avoidance & Detection

Banker's Resource- Request Algorithm : Detailed Four Steps

Whenever a process makes a request of the resources then this algorithm checks that if the resource can be allocated or not.

It includes three steps:

1. The algorithm checks that if the request made is valid or not. A request is valid if the number of requested resources of each resource type is less than the **Need(**which was declared previously by the process**).** If it is a valid request then step 2 is executed else aborted.

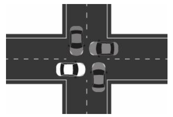
2. Here, the algorithm checks that if the number of requested instances of each resource is less than the available resources. If not then the process has to wait until sufficient resources are available else go to step 3.

3. Now, the algorithm assumes that the resources have been allocated and modifies the data structure accordingly.

   Available = Available - Request(i)

   Allocation(i) = Allocation(i) + Request(i)

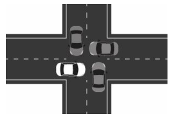
   Need(i) = Need(i) - Request(i)



Deadlocks: Deadlock Avoidance & Detection

Banker's Resource- Request Algorithm : Detailed Four Steps

- After the allocation of resources, the new state formed may or may not be a safe state. So, the **safety algorithm** is applied to check whether the resulting state is a safe state or not.



Deadlocks: 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?
    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?

PES UNIVERSITY ONLINE

THANK YOU

Nitin V Pujari
Faculty, Computer Science
Dean - IQAC, PES University

nitin.pujari@pes.edu

For Course Deliverables by the Anchor Faculty click on www.pesuacademy.com