OPERATING SYSTEMS

Deadlock Avoidance

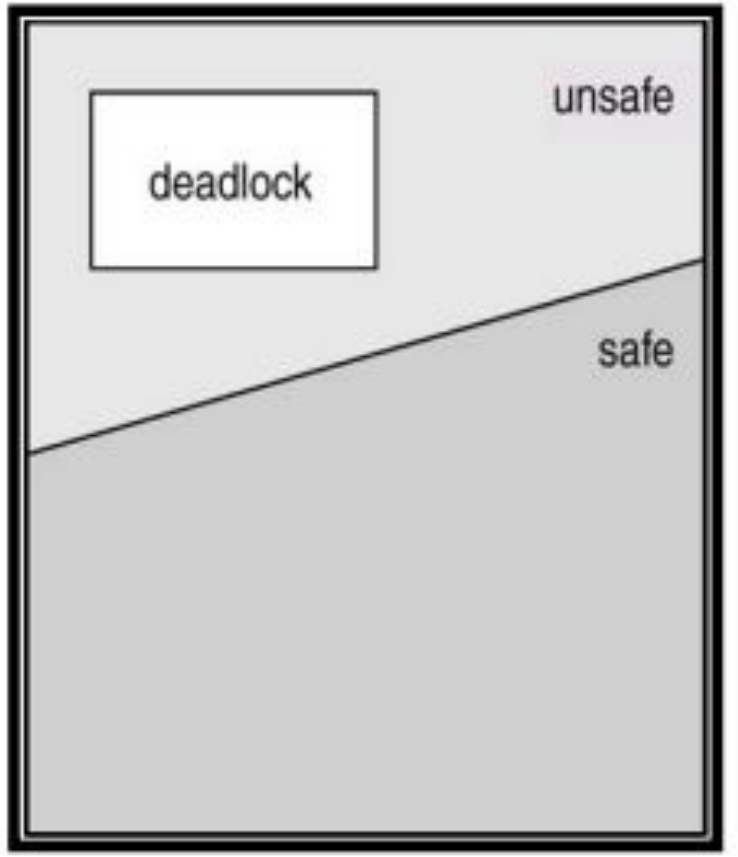
- One method for avoiding deadlocks is to require additional information about how resources may be requested.
- Each request for resources by a process requires that the system consider the resources currently available, the resources currently allocated to the process, and the future requests and releases of each process, to decide whether the current request can be satisfied or must wait to avoid a possible future deadlock.
- The simplest and most useful model requires that each process declare the maximum number of resources of each type that it may need.
- Given a priori information about the maximum number of resources of each type that may be requested by each process, it is possible to construct an algorithm that ensures that the system will never enter a deadlocked state.
- A deadlock avoidance algorithm dynamically examines the resource-allocation state to ensure that a circular wait condition can never exist.

Safe State

- A state is safe if 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.
- A sequence of processes is a safe sequence for the current allocation state if, for each Pi, the resources that Pi can still request can be satisfied by the currently available resources plus all the resources held by all the Pj with $j < i$.
- In this situation, if the resources that Pi needs are not immediately available, then Pi can wait until all Pj have finished.
- When they have finished, Pi can obtain all of its needed resources, complete its designated task, return its allocated resources and terminate.
- When Pi terminates, Pi+1 can obtain its needed resources and terminate.
- If no such sequence exists, then the system is said to be unsafe.

Safe State

- If a system is in a safe state, there can be no deadlocks.
- An unsafe state is not a deadlocked state; a deadlocked state is conversely an unsafe state.
- Not all unsafe states are deadlocks, however an unsafe state may lead to a deadlock state.
- Deadlock avoidance makes sure that a system never enters an unsafe state.



Safe State (Example)

- There is a system with 12 tape drives and three processes <P0,P1,P2>.

Process	Max Need	Allocated
P0	10	5
P1	4	2
P2	9	2

- Therefore, system is currently in a safe state, with the safe sequence <P1,P0,P2>.

Safe State (Example)

- Now, consider that P2 requests and is allocated one more tape drive. Assuming that the tape drive is allocated to P2, the new system state will be:

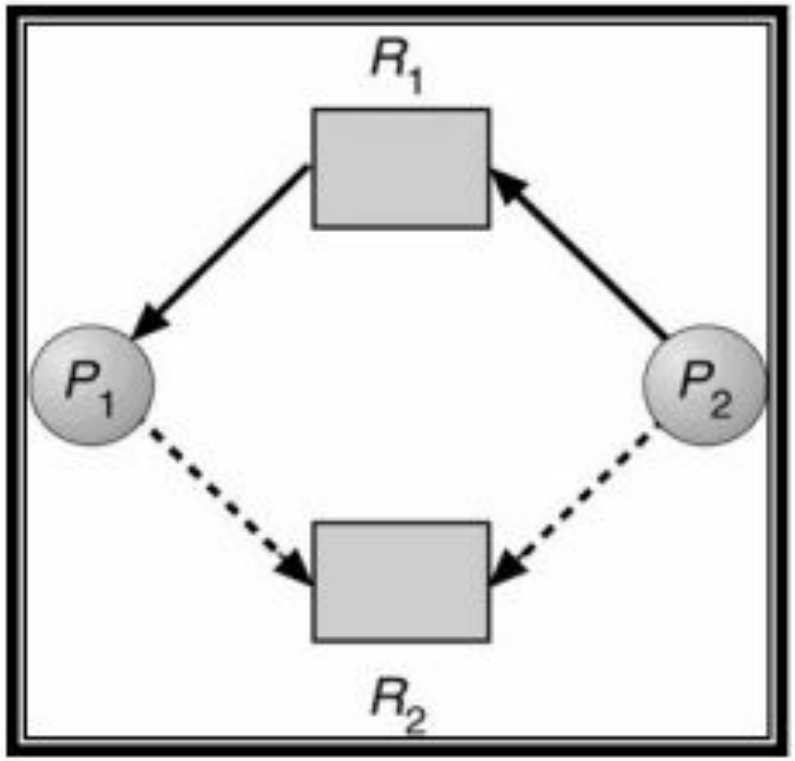
Process	Max Need	Allocated
P0	10	5
P1	4	2
P2	9	3

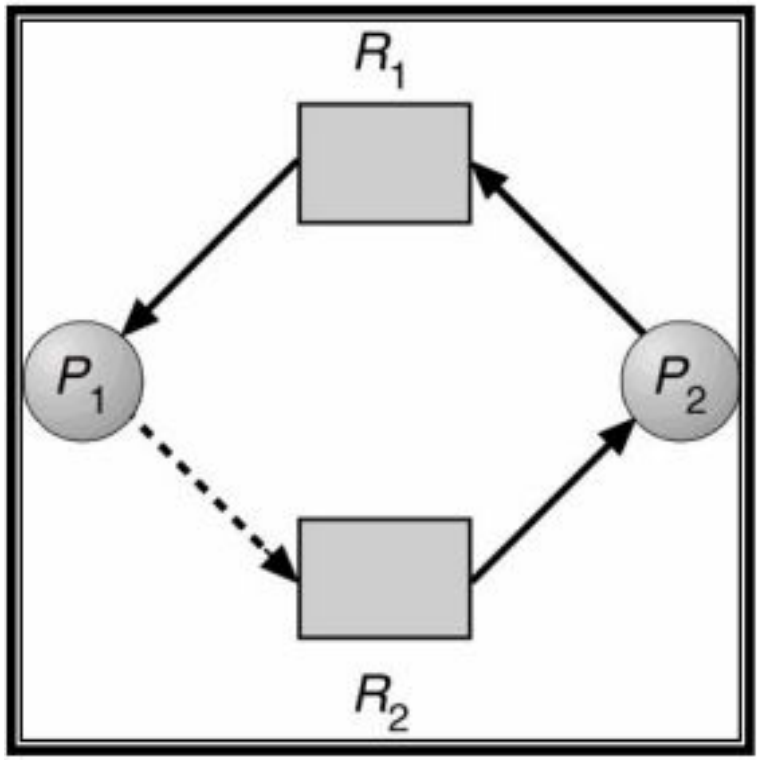
- This new system is not safe. P1's maximum remaining future need can be satisfied but neither P0's nor P2's maximum future needs can be satisfied.

Resource Allocation Graph

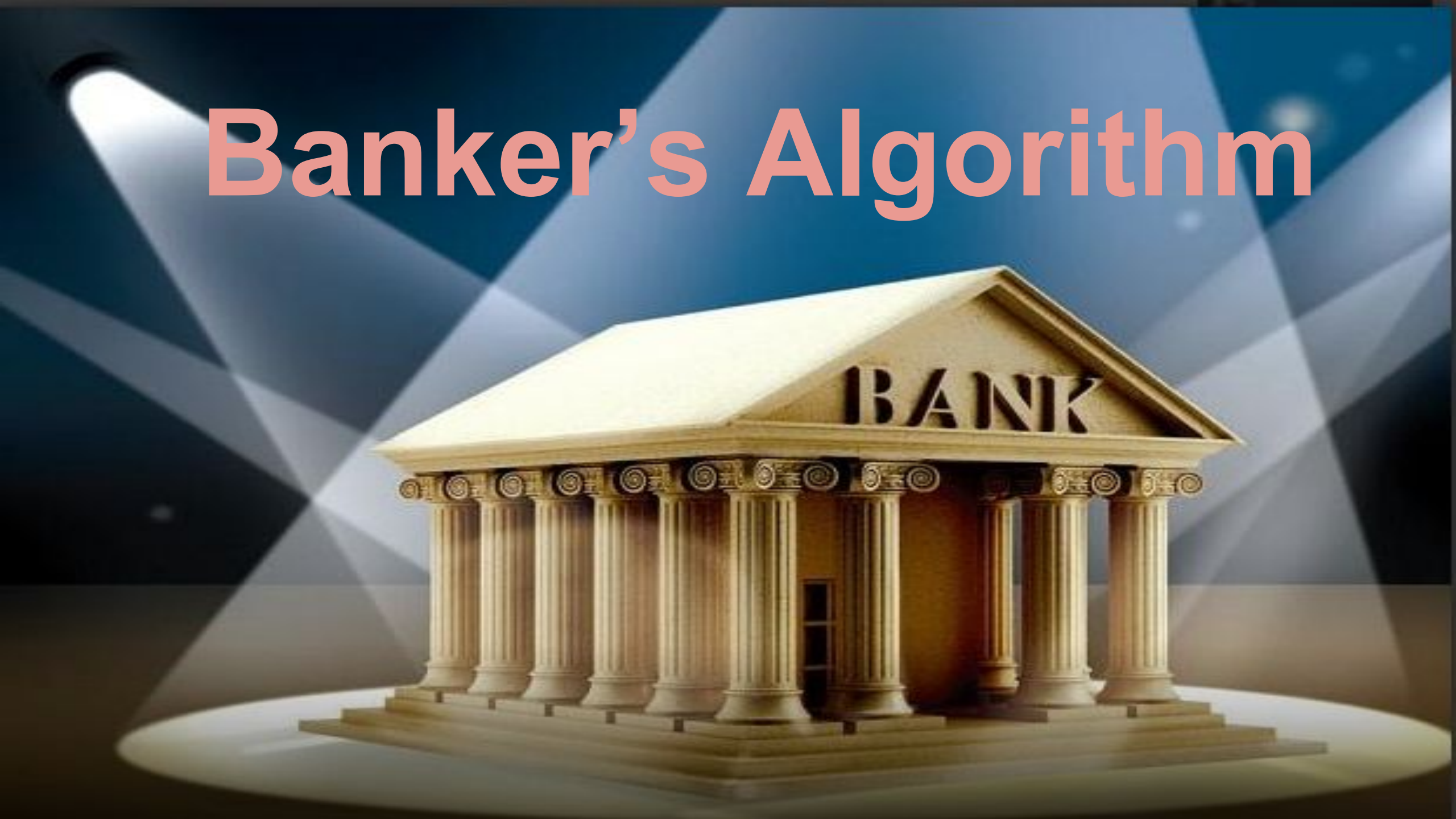
- In addition to the request and assignment edges, we introduce a new type of edge called a claim edge to resource allocation graphs.
- A claim edge Pi →Rj indicates that process Pi may request resource Rj at some time in the future.
- A dashed line is used to represent a claim edge. ----->
- When Pi requests resource Rj the claim edge is converted to a request edge.
- Suppose that Pi requests resource Rj. The request can be granted only if converting the request edge Pi →Rj into an assignment edge Rj →Pi does not result in the formation of a cycle.
- If no cycle exists, then the allocation of the resource will leave the system in a safe state.
- If a cycle is found, then the allocation will put the system in an unsafe state.

Resource Allocation Graph





Banker's Algorithm



Banker's Algorithm

- When a new process enters the system, it must declare the maximum number of instances of each resource type that it may need, i.e., each process must a priori claim maximum use of various system resources.
- This number may not exceed the total number of instances of resources in the system, and there can be multiple instances of resources.
- When a process requests a set of resources, the system must determine whether the allocation of these resources will leave the system in a safe state.
- If it will, the resources are allocated; otherwise the process must wait until some other process releases enough resources.
- We say that a system is in a safe state if all of the processes in the system can be executed to termination in some order; the order of process termination is called safe sequence.
- When a process gets all its resources, it must use them and return them in a finite amount of time.

Banker's Algorithm

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

Data structures:
Available: A vector of length m indicates the number of available instances of resources of each type.
Available[j] = = k means that there are k available instances of resource Rj.
Max: An n x m matrix defines the maximum demand of resources of each process.
Max[i,j] = = k means that process Pi may request at most k instances of resource Rj. *f*

Banker's Algorithm

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

Allocation[i,j] = = k means that Pi is currently allocated k instances of resource type Rj.

Need: An n x m matrix indicates the remaining resource need of each process.

Need[i,j] = = k means that Pi may need k more instances of resource type Rj to complete its task. Note that Need[i,j] = = Max[i,j] - Allocation[i,j].

Safety Algorithm

- The algorithm for finding out whether or not a system is in a safe state can be described as follows:

1. Let Work and Finish be vectors of length m and n, respectively. Initialize Work = Available and Finish[i] = false for i = 1, 2, …, n.
2. Find an i such that both a) Finish[i] = = false b) $Need_i$ <= 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] = = true for all i, then the system is in a safe mode.

- This algorithm may require an order of m x n^2 operations to decide whether a state is safe

Resource Request Algorithm

- Let $Request_i$ be the request vector for process Pi. If $Request_i[j]=k$, then process Pi wants k instances of resource Rj. When a request for resources is made by process Pi the following actions are taken:

1. If $Request_i <= Need_i$ go to step 2. Otherwise, raise an error condition since the process has exceeded its maximum claim.
2. If $Request_i <= Available$, go to step 3. Otherwise Pi 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:

$$Availabe = Available - Request_i ;$$
$$Allocation_i = Allocation_i + Request_i ;$$
$$Need_i = Need_i - Request_i ;$$

- Invoke the Safety algorithm. If the resulting resource allocation graph is safe, the transaction is completed. Else, the old resource allocation state is restored and process Pi must wait for $Request_i$..

Example 1

- Consider a system with five processes P0 through P4 and three resource types: A, B, C.

	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			

Example 1

- The content of the matrix **Need** is defined to be Max- Allocation and is:

	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

Example 1

	Allocation			Need			Available		
	A	B	C	A	B	C	A	B	C
P_0	0	1	0	7	4	3	3	3	2
P_1	2	0	0	1	2	2	5	3	2
P_2	3	0	2	6	0	0			
P_3	2	1	1	0	1	1			
P_4	0	0	2	4	3	1			

Safe Sequence: $< P_1 >$

Example 1

	Allocation			Need			Available		
	A	B	C	A	B	C	A	B	C
P_0	0	1	0	7	4	3	3	3	2
P_1	2	0	0	1	2	2	5	3	2
P_2	3	0	2	6	0	0	7	4	3
P_3	2	1	1	0	1	1			
P_4	0	0	2	4	3	1			

Safe Sequence: $< P_1, P_3 >$

Example 1

	Allocation			Need			Available		
	A	B	C	A	B	C	A	B	C
P_0	0	1	0	7	4 5	3	3	3	2
P_1	2	0	0	1	2	2	5	3	2
P_2	3	0	2	6	0	0	7	4	3
P_3	2	1	1	0	1	1	7	4	5
P_4	0	0	2	4	3	1			

Safe Sequence: $< P_1, P_3, P_4 >$

Example 1

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

Safe Sequence: $< P_1, P_3, P_4, P_2, P_0 >$

Example 2

- P0 requests (0,2,0).
- Should this request be granted?
- In order to answer this question, we again follow Banker's algorithm

✔ Steps:

1. Is Request0 ≤ Need0?

- (0,2,0) ≤ (7,4,3) ⇒ true

2. Is Request0 ≤ Available?

- (0,2,0) ≤ (3,3,2) ⇒ true

Example 2

	Need			Allocation			Available		
	A	B	C	A	B	C	A	B	C
P_0	7	4	3	0	1	0	3	3	2
P_1	1	2	2	2	0	0			
P_2	6	0	0	3	0	2			
P_3	0	1	1	2	1	1			
P_4	4	3	1	0	0	2			

Example 2

- The following is the updated system state. We run the Safety algorithm on this state now and show the steps of executing the algorithm.

	Need			Allocation			Work		
	A	B	C	A	B	C	A	B	C
P_0	7	2	3	0	3	0	3	1	2
P_1	1	2	2	2	0	0	5	2	3
P_2	6	0	0	3	0	2			
P_3	0	1	1	2	1	1			
P_4	4	3	1	0	0	2			

Safe Sequence: $<P_3>$

Example 2

- The following is the updated system state. We run the Safety algorithm on this state now and show the steps of executing the algorithm.

	Need			Allocation			Work		
	A	B	C	A	B	C	A	B	C
P_0	7	2	3	0	3	0	3	1	2
P_1	1	2	2	2	0	0	5	2	3
P_2	6	0	0	3	0	2	7	2	3
P_3	0	1	1	2	1	1			
P_4	4	3	1	0	0	2			

- Safe Sequence: <P3,P1>

Example 2

	Need			Allocation			Work		
	A	B	C	A	B	C	A	B	C
P_0	7	2	3	0	3	0	3	1	2
P_1	1	2	2	2	0	0	5	2	3
P_2	6	0	0	3	0	2	7	2	3
P_3	0	1	1	2	1	1	10	2	5
P_4	4	3	1	0	0	2			

Safe Sequence: $<P_3, P_1, P_2>$

Example 2

	Need			Allocation			Work		
	A	B	C	A	B	C	A	B	C
P_0	7	2	3	0	3	0	3	1	2
P_1	1	2	2	2	0	0	5	2	3
P_2	6	0	0	3	0	2	7	2	3
P_3	0	1	1	2	1	1	10	2	5
P_4	4	3	1	0	0	2	10	5	5

Safe Sequence: $<P_3, P_1, P_2, P_0, P_4>$

Practice Example

- Note that safe sequence is not necessarily a unique sequence.
- There are several safe sequences for the above example.

Practice Example

- If P1 requests (1,0,2), can this request may be granted immediately?