EE6253 - Operating Systems & Network Programming

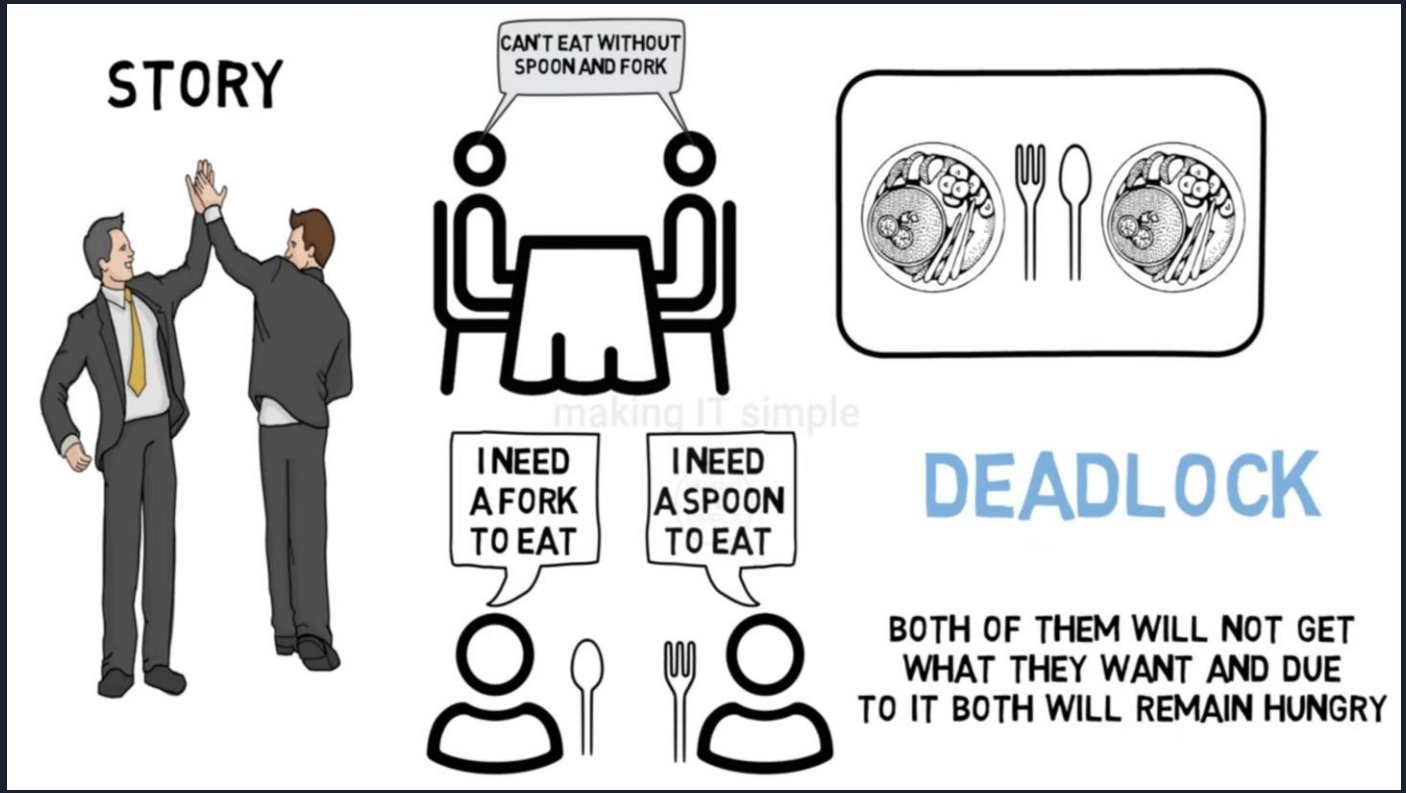
Yugani Gamlath
Department of Electrical and Information Engineering
Faculty of Engineering
University of Ruhuna

Lecture 05 – Deadlocks

Intended Learning Outcomes

- Introduction to Deadlock
- Conditions for Deadlock
- Banker's Algorithm, Safety Algorithm, Safe Sequence
- Resource Allocation Graph (RAG) & Wait For Graph (WFG)
- Methods of Handling Deadlocks

Real-World Story



Introduction to Deadlock

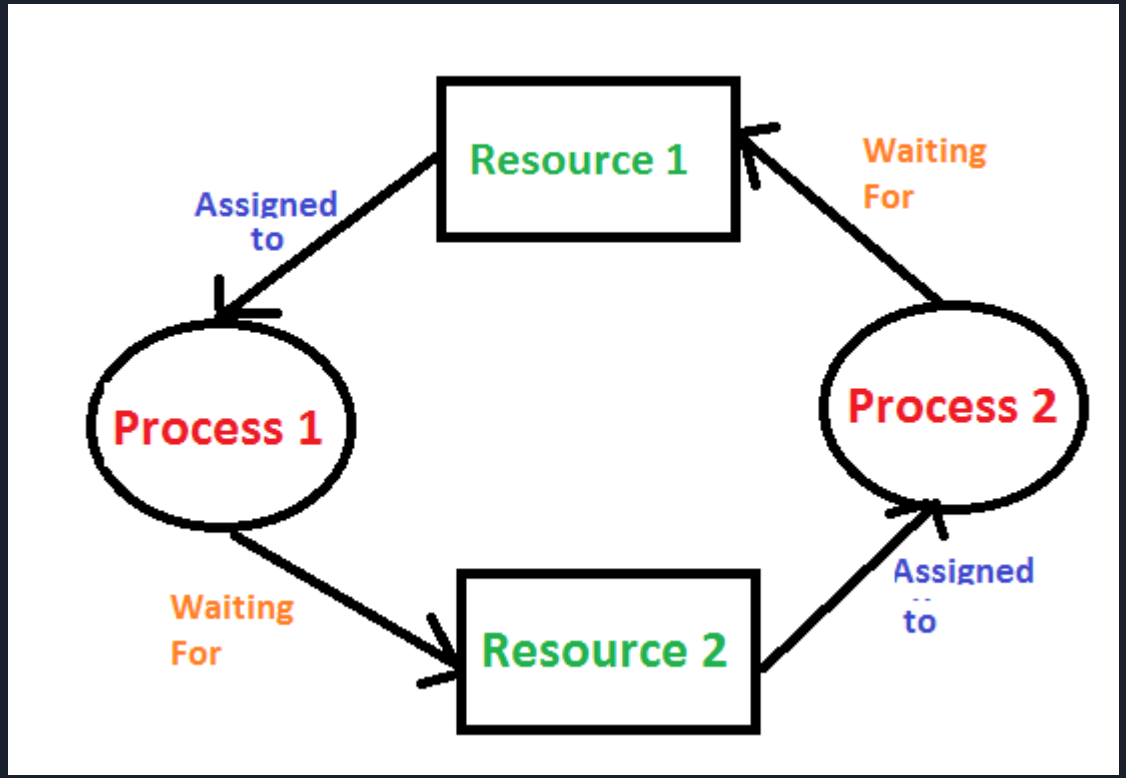
- A deadlock is a situation where a set of processes are blocked because each process is holding a resource and waiting for another resource acquired by some other process.

Example:

- When two trains are coming toward each other on the same track and there is only one track, none of the trains can move once they are in front of each other.

- A similar situation occurs in OS when there are two or more processes that hold some resources and wait for resources held by other(s).

Introduction to Deadlock



Introduction to Deadlock

Both processes begin execution at the same time.

Process 1 obtains Resource 1
Process 2 obtains Resource 2

Current situation:

Process 1 possesses Resource 1
Process 2 possesses Resource 2

What happens next:

Process 1 requires Resource 2 to continue execution but is unable to do so because Process 2 is currently holding Resource 2.

Introduction to Deadlock

Similarly, Process 2 requires Resource 1 to continue execution but is unable to do so because Process 1 is currently holding Resource 1.

Both processes are now stuck in a loop:

Process 1 is awaiting Resource 2 from Process 2
Process 2 is awaiting Resource 1 from Process 1

We have a deadlock because neither process can release the resource it is holding until it completes its task, and neither can proceed without the resource the other process is holding.

Both processes are deadlocked, unable to move forward.

Conditions for Deadlock

- **Mutual Exclusion:**
  Only one process at a time can use the resource.

- **Hold and Wait:**
  Processes hold resources already allocated to them while waiting for other resources.

- **No preemption:**
  Resources are released by processes holding them only after that process has completed its task.

- **Circular wait:**
  A circular chain of processes exists in which each process waits for one or more resources held by the next process in the chain.

Conditions for Deadlock
Mutual Exclusion

This condition requires that at least one resource be held in a non-shareable mode, which means that only one process can use the resource at any given time.

Both Resource 1 and Resource 2 are non-shareable in our scenario, and only one process can have exclusive access to each resource at any given time.

As an example:

- Process 1 obtains Resource 1
- Process 2 acquires Resource 2

Conditions for Deadlock
Hold and Wait

Hold and wait condition specifies that a process must be holding at least one resource while waiting for other processes to release resources that are currently held by other processes.

In example,

- Process 1 has Resource 1 and is awaiting Resource 2.
- Process 2 currently has Resource 2 and is awaiting Resource 1.
- Both processes hold one resource while waiting for the other, satisfying the hold and wait condition.

Conditions for Deadlock
No Preemption

- Preemption is the act of taking a resource from a process before it has finished its task.
- According to the no preemption condition, resources cannot be taken forcibly from a process a process can only release resources voluntarily after completing its task.
- In our scenario, neither Process 1 nor Process 2 can be forced to release the resources in their possession. The processes can only release resources voluntarily.

Conditions for Deadlock
Circular wait

- This condition implies that circular processes must exist, with each process waiting for a resource held by the next process in the chain.
- In our scenario, Process 1 is waiting for Resource 2, which is being held by Process 2.
- Process 2 is awaiting Resource 1 from Process 1.
- This circular chain of dependencies causes a deadlock because neither process can proceed, resulting in a system shutdown.

Banker's Algorithm

- The banker’s algorithm is a resource allocation and deadlock avoidance algorithm.
- Test for safety by simulating the allocation for the predetermined maximum possible amounts of all resources.
- Makes an “s-state” check to test for possible conditions for all other pending activities, before deciding whether allocation should be allowed to continue.

Banker's Algorithm

- The banker’s algorithm is named so because it is used in the banking system to check whether a loan can be sanctioned to a person or not.
- Suppose there are n number of account holders in a bank and the total sum of their money is S.
- If a person applies for a loan then the bank first subtracts the loan amount from the total money that the bank has and if the remaining amount is greater than S then only the loan is sanctioned.
- It is done because if all the account holders come to withdraw their money then the bank can easily do it.

Banker's Algorithm

- It also helps the OS to successfully share the resources between all the processes.
- It is called the banker's algorithm because bankers need a similar algorithm, they admit loans that collectively exceed the bank's funds and then release each borrower's loan in installments.
- This algorithm uses the notation of a safe allocation state to ensure that granting a resource request cannot lead to a deadlock either immediately or in the future.
- Bank would never allocate its money in such a way that it can no longer satisfy the needs of all its customers. The bank would try to be in a safe state always.

Banker’s Algorithm Reference

Banker's Algorithm Visual Explanation

Banker's Algorithm
Data structures

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

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

Banker's Algorithm
Data structures

Max

- It is a 2-d array of size 'n*m' that defines the maximum demand of each process in a system.
- Max[i,j] = k means process Pi may request at most 'k' instances of resource type Rj.

Banker's Algorithm
Data structures

Allocation

- It is a 2-d array of size 'n*m' that defines the number of resources of each type currently allocated to each process.
- Allocation[i,j] = k means process Pi is currently allocated 'k' instances of resource type Rj

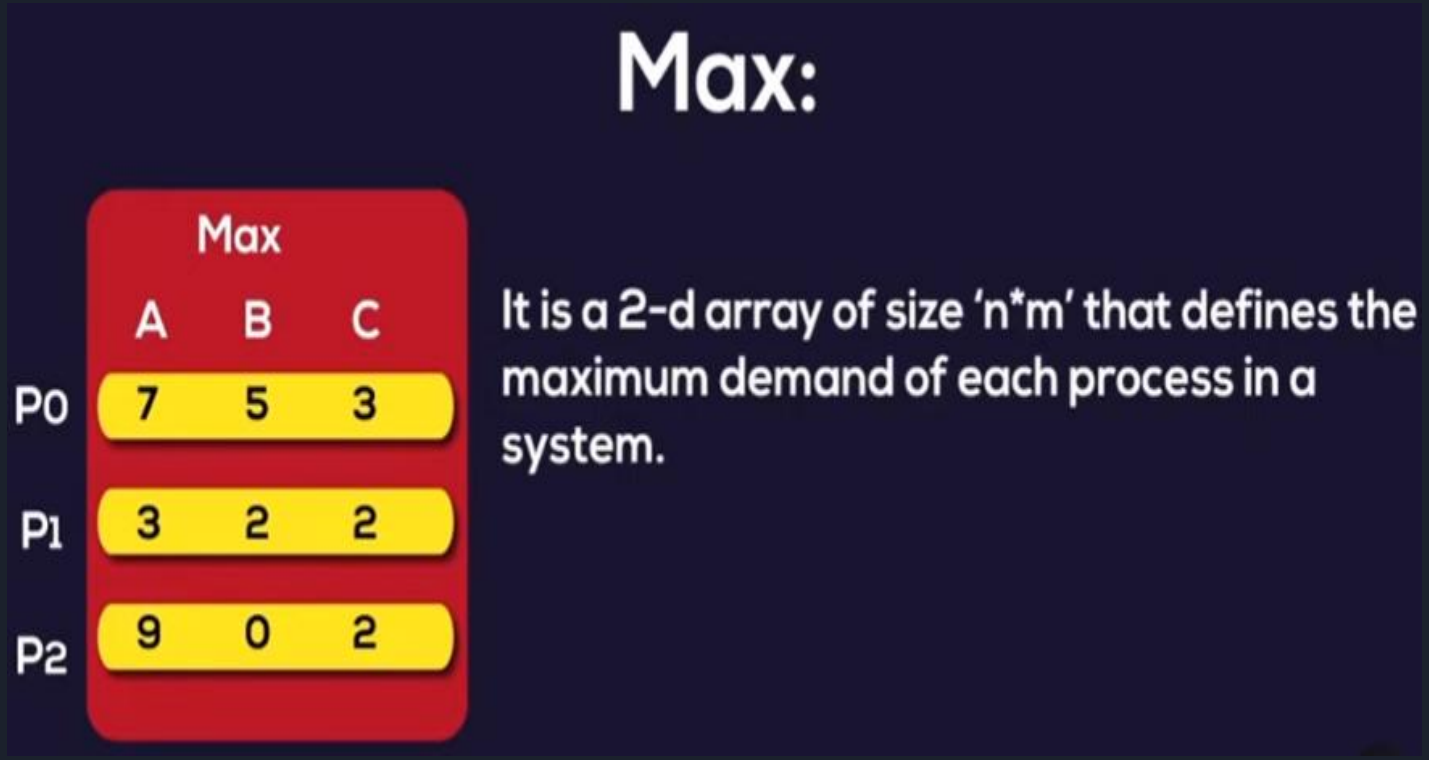
Banker's Algorithm
Data structures

Need

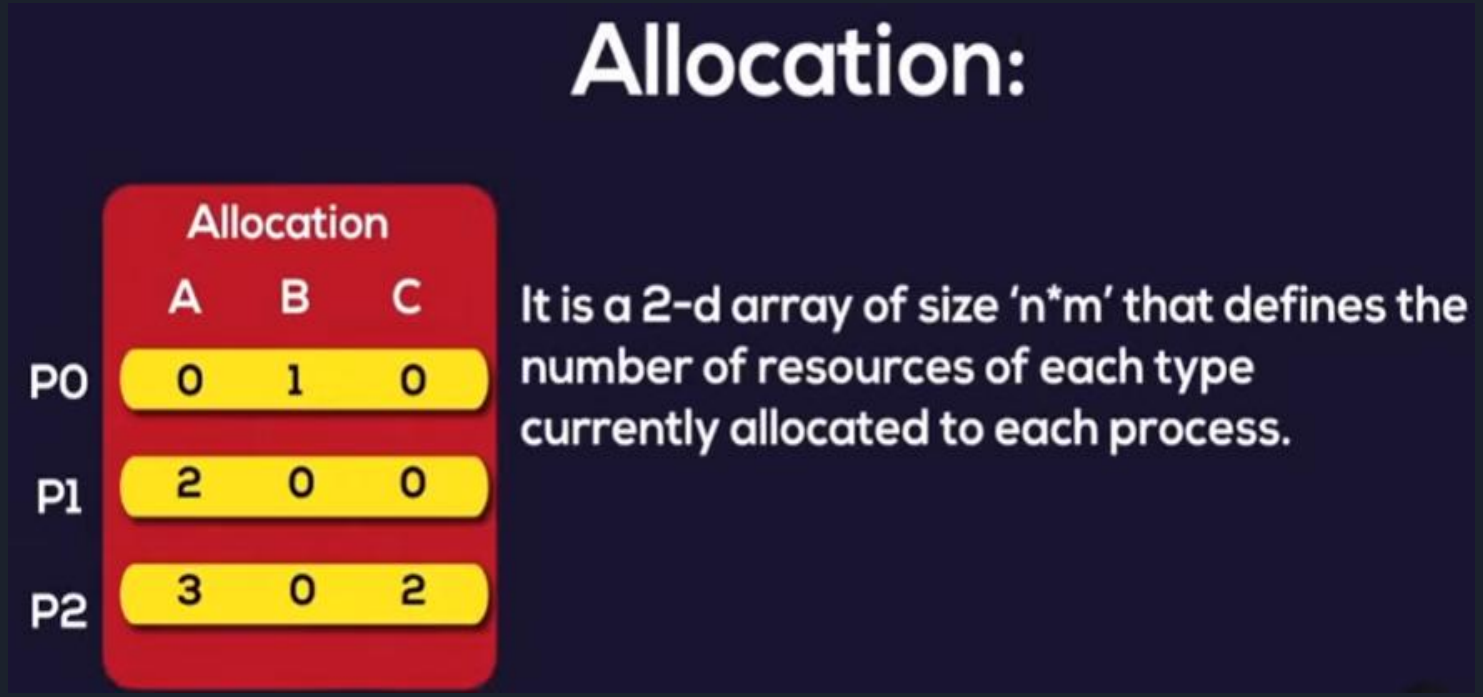
- It is a 2-d array of size 'n*m' that indicates the remaining resource need of each process.
- Need [i,j] = k means process Pi currently needs 'k' instances of resource type Rj
- Need [i,j] = Max [i,j] – Allocation [i,j]

Allocation specifies the resources currently allocated to process Pi and Needi specifies the additional resources that process Pi may still request to complete its task.

Banker's Algorithm

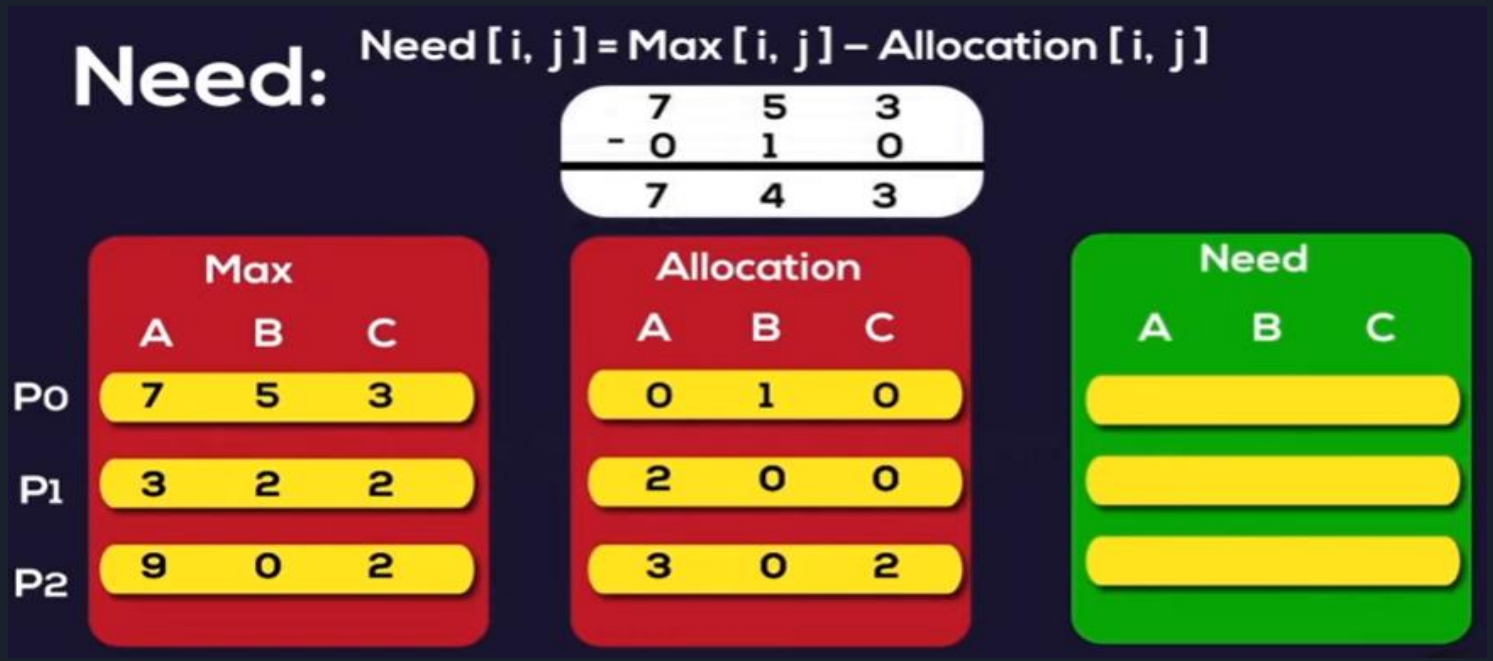


Banker's Algorithm

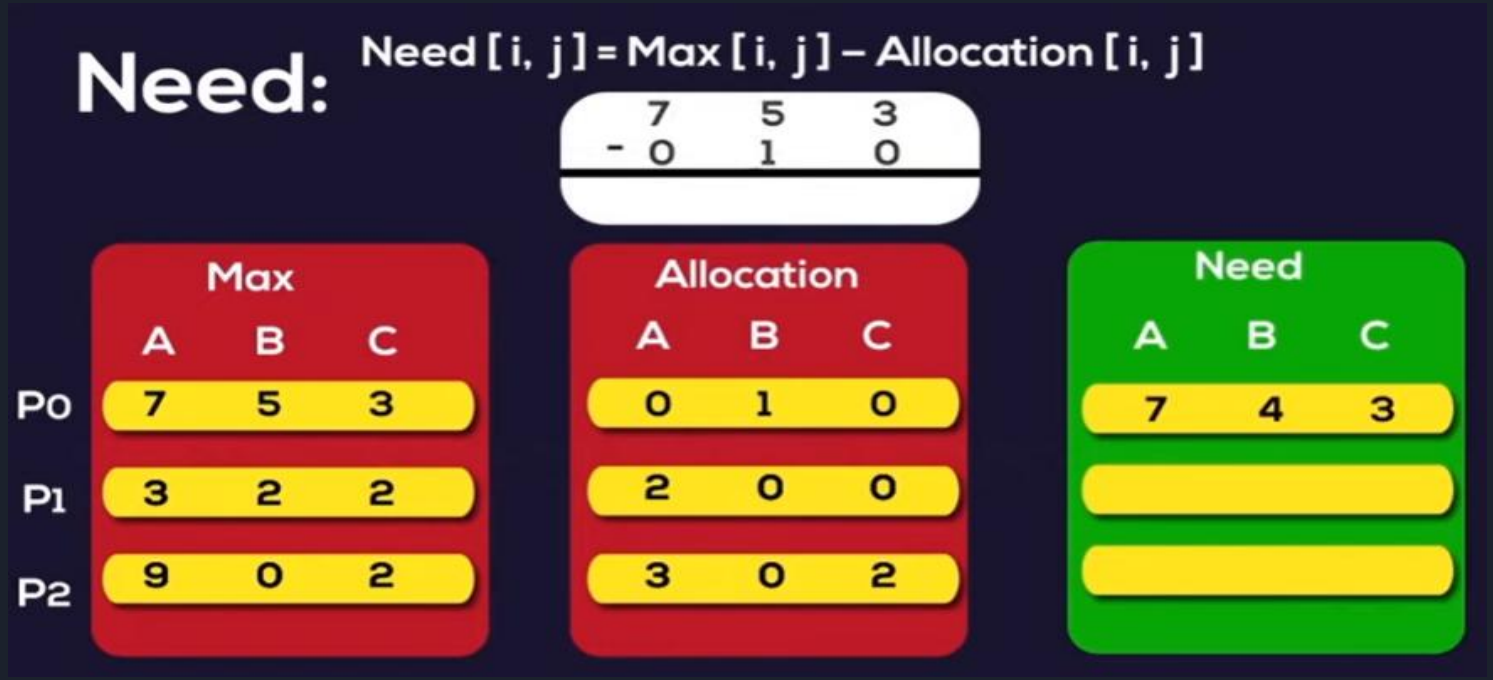


Banker's Algorithm

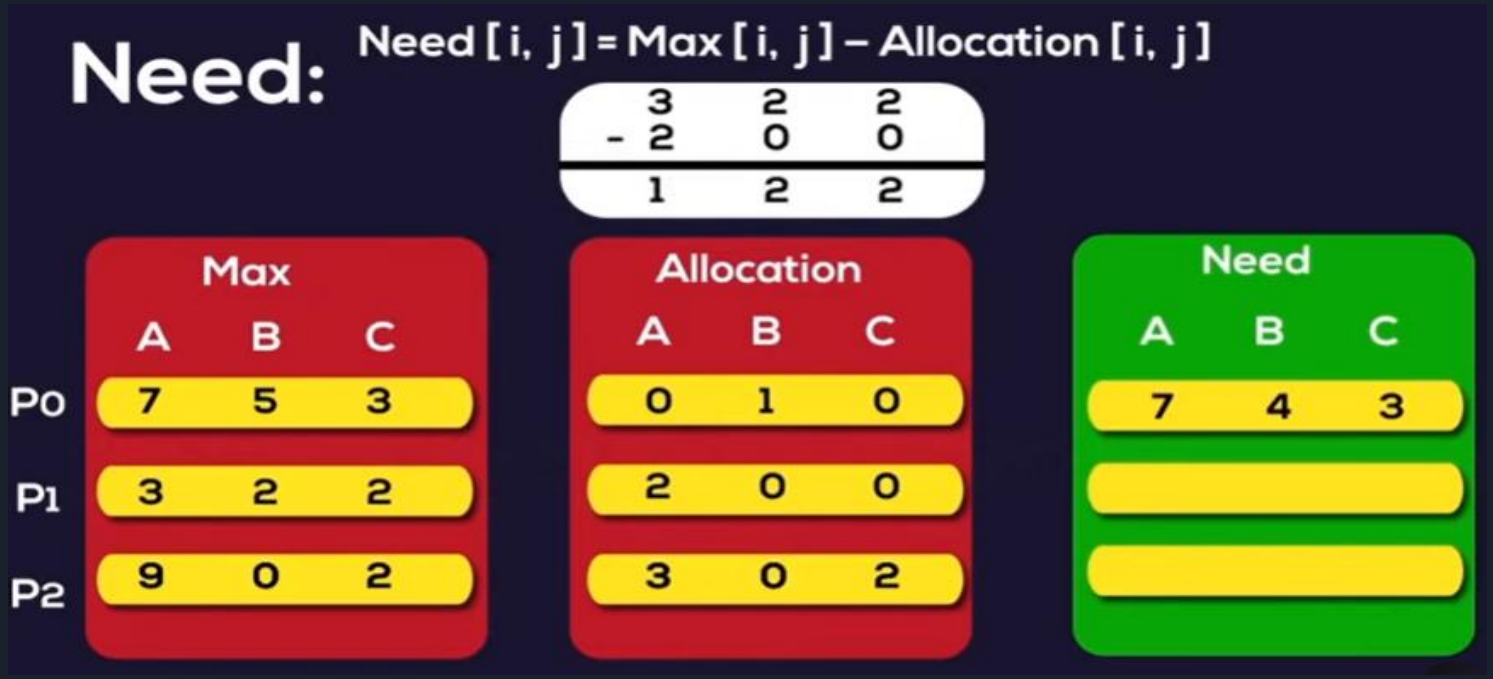
Banker’s Algorithm



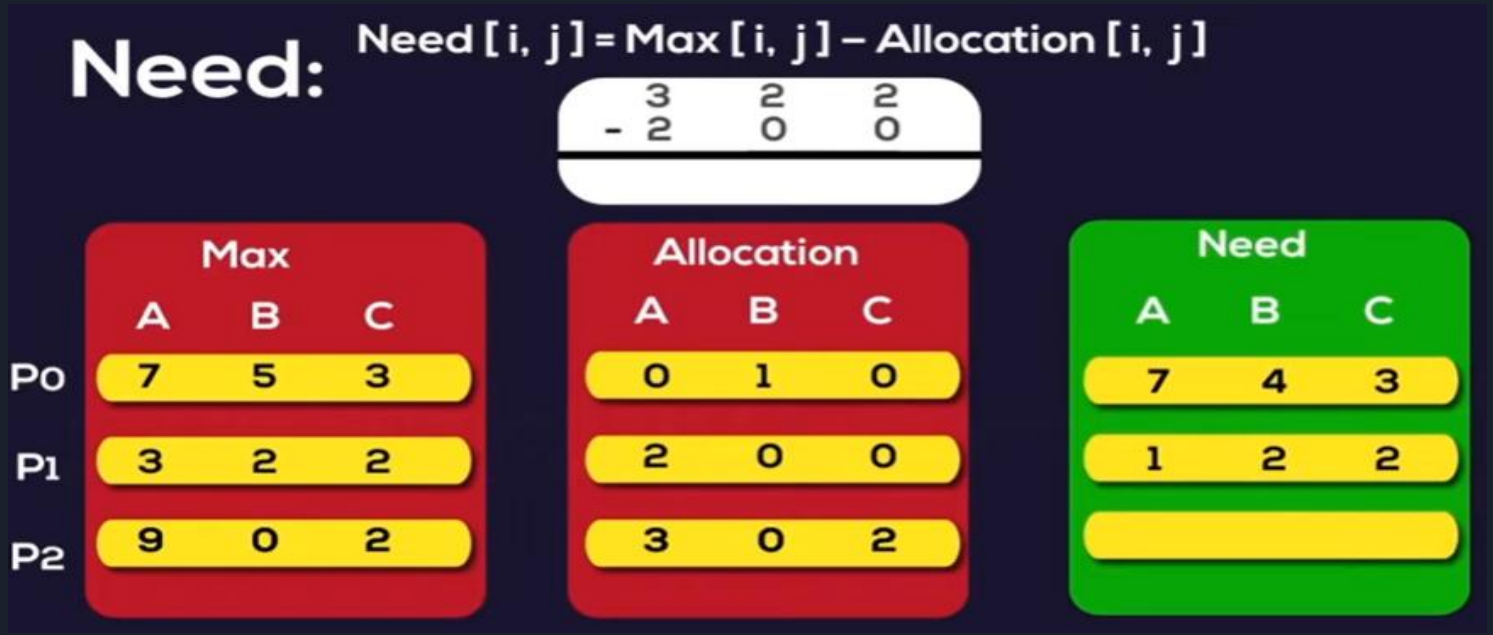
Banker's Algorithm



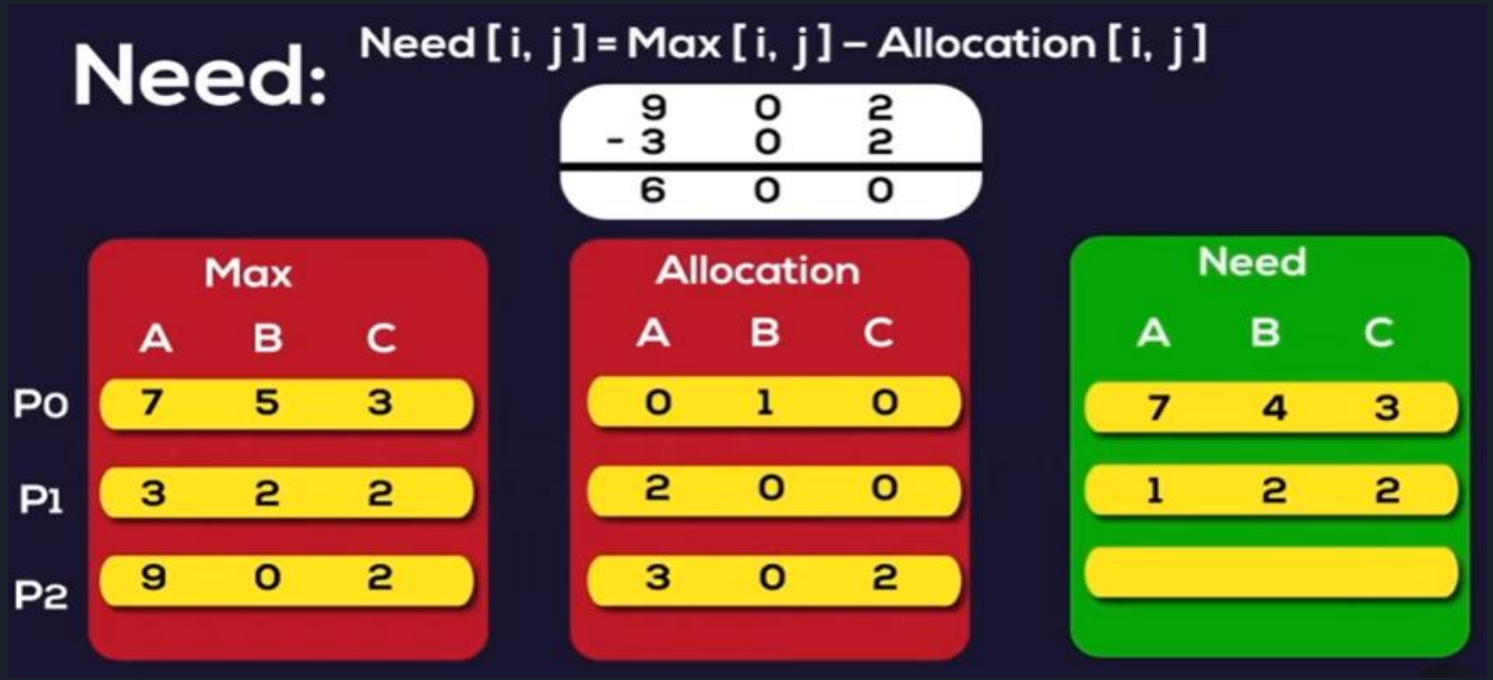
Banker’s Algorithm



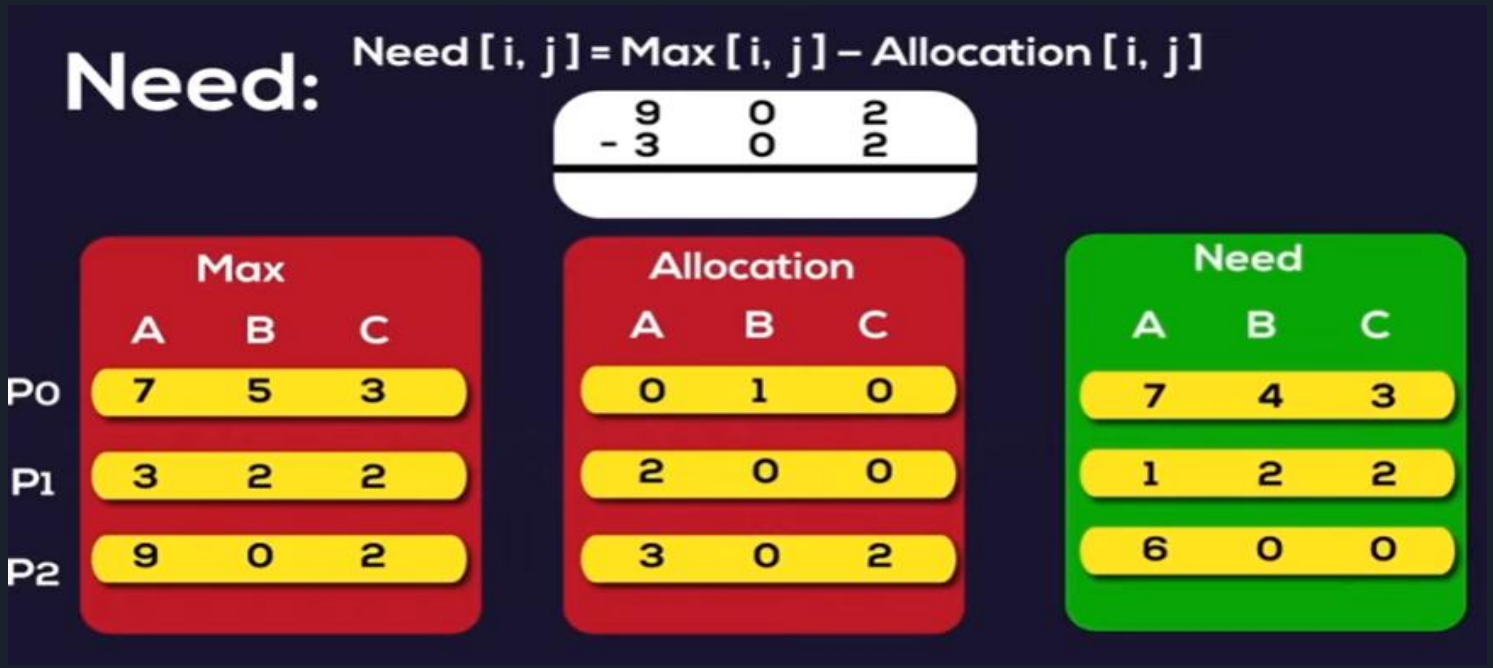
Banker’s Algorithm



Banker’s Algorithm



Banker’s Algorithm



Banker's Algorithm

Banker's Algorithm

Banker's Algorithm

5 1

Banker's Algorithm

Banker’s Algorithm

Banker's Algorithm

Banker’s Algorithm

Safety Algorithm

Finish Array

- Finish[i] represents status of ith process.
- Finish[i] = True means process is completed.
- Finish[i] = False means process is unfinished.

Safety Algorithm

Work Array

- This array represents resources that have been used and given back by processes on their completion.

- Work[i] = K means K instance of Resources (i) have been returned.

This algorithm is used for determining whether or not a particular state is safe.

Safety Algorithm

Let's see safety algorithm implemented on our example

In our case No. of processes (N) is 3 and No. of type of resources (M) is also 3

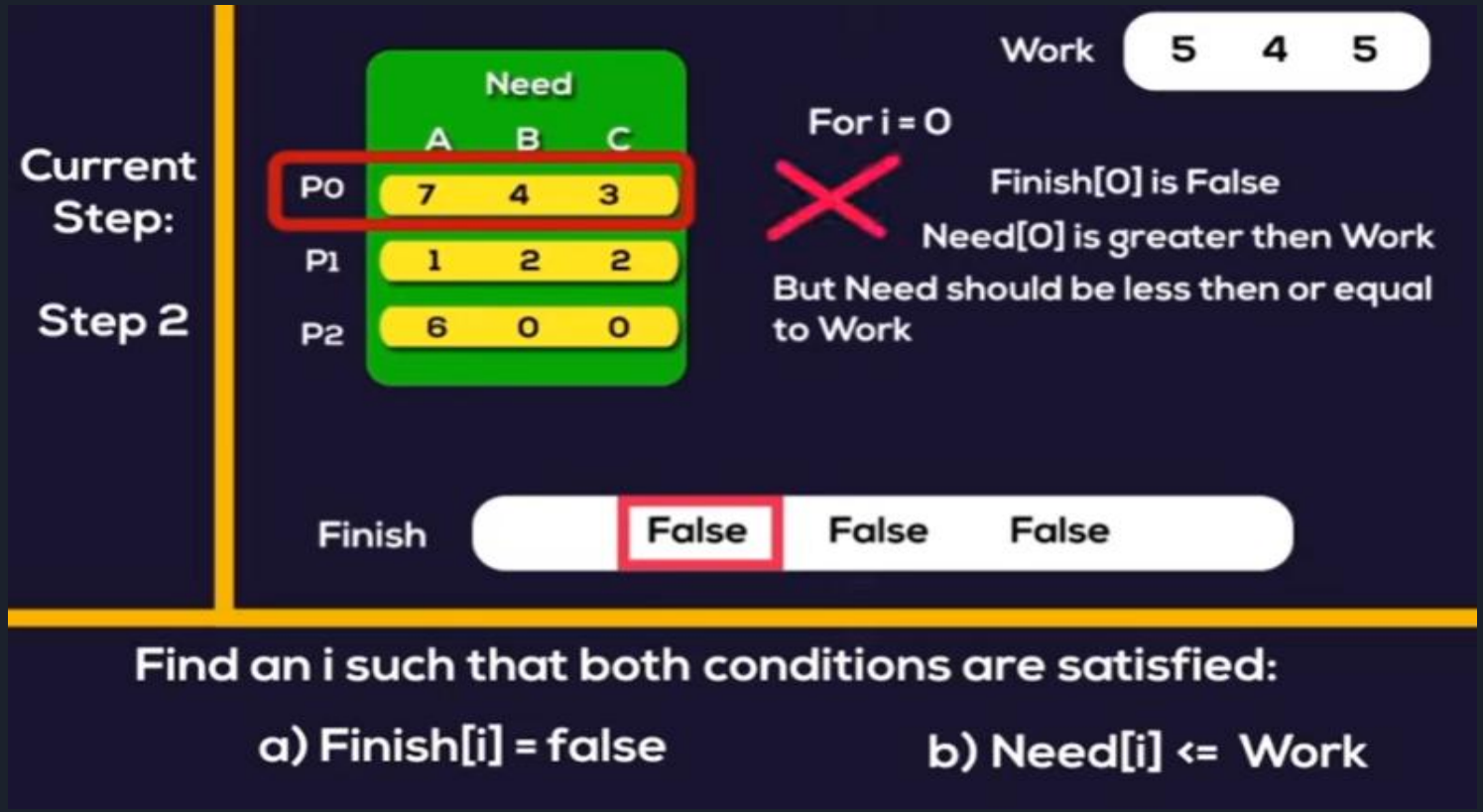
Available array in our case was 5 4 5

Thus Work array will also be 5 4 5

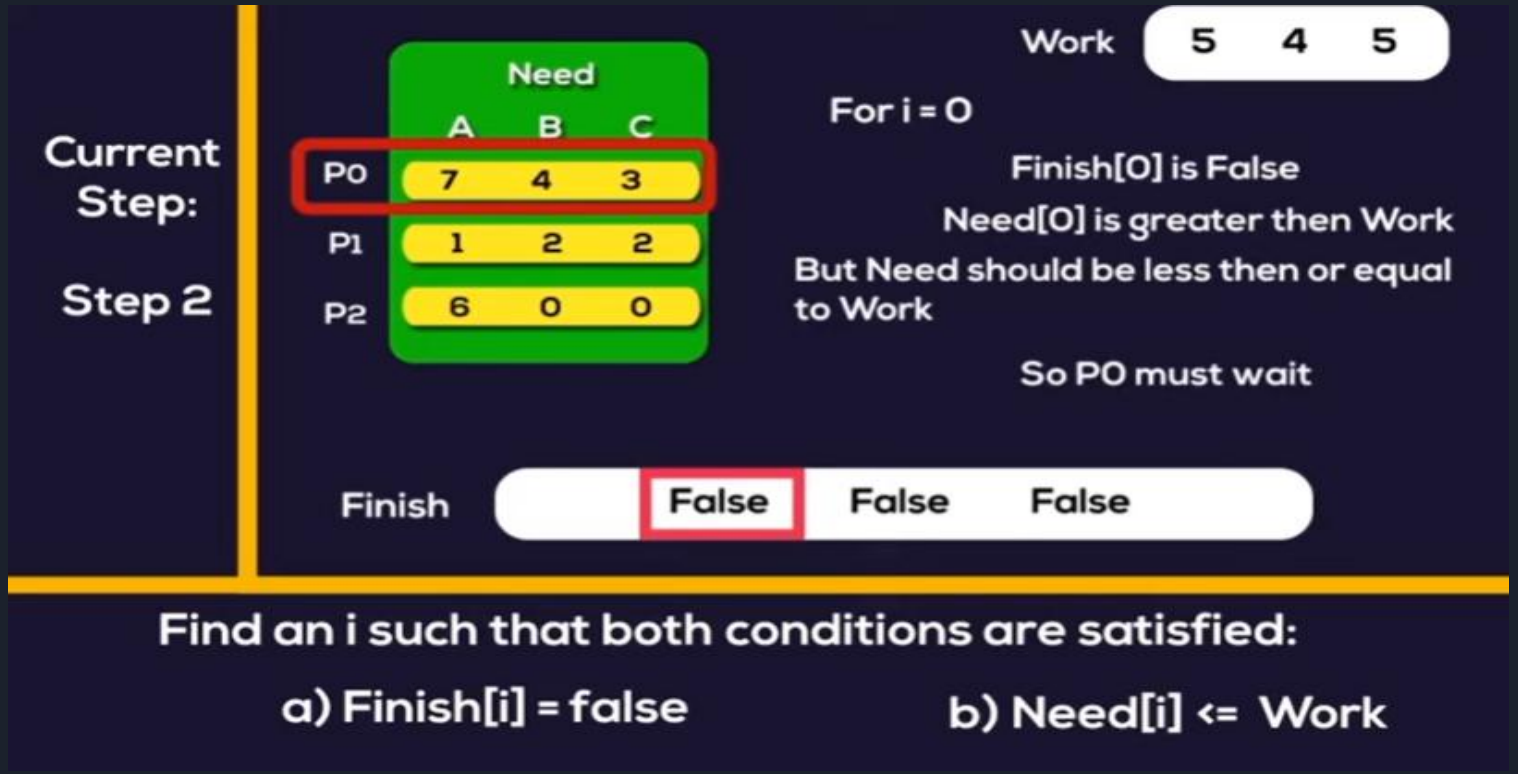
Finish array will be:

False False False

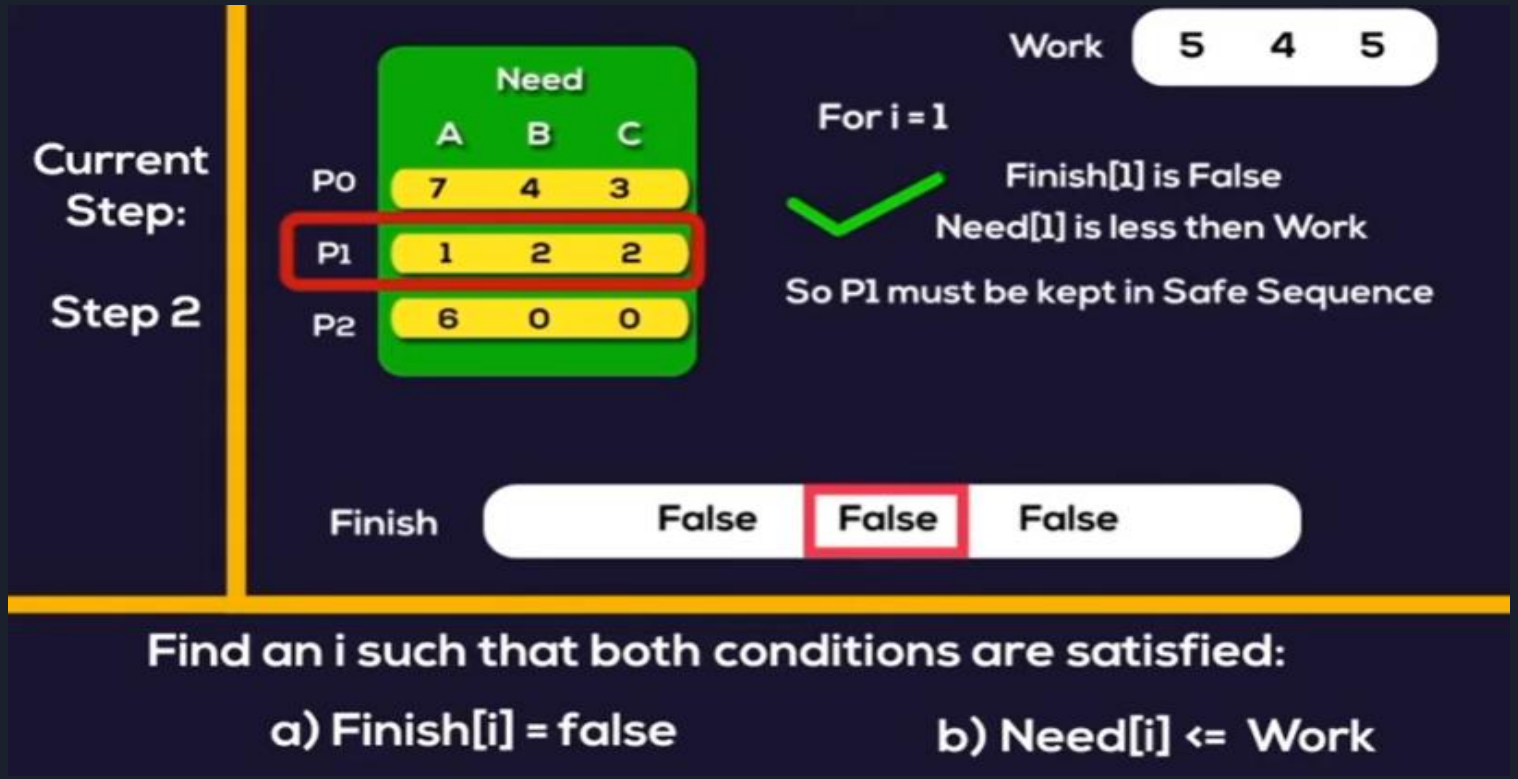
Safety Algorithm



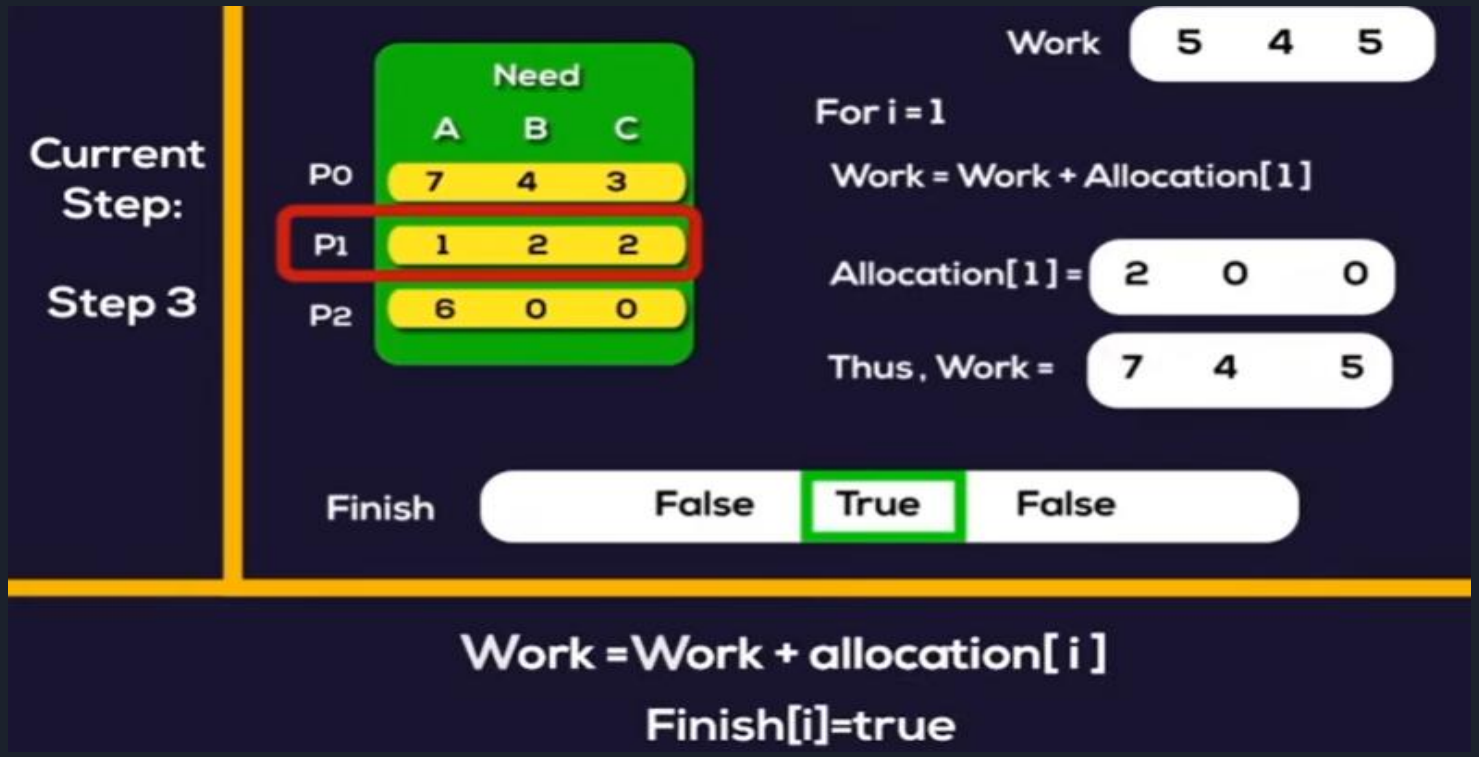
Safety Algorithm



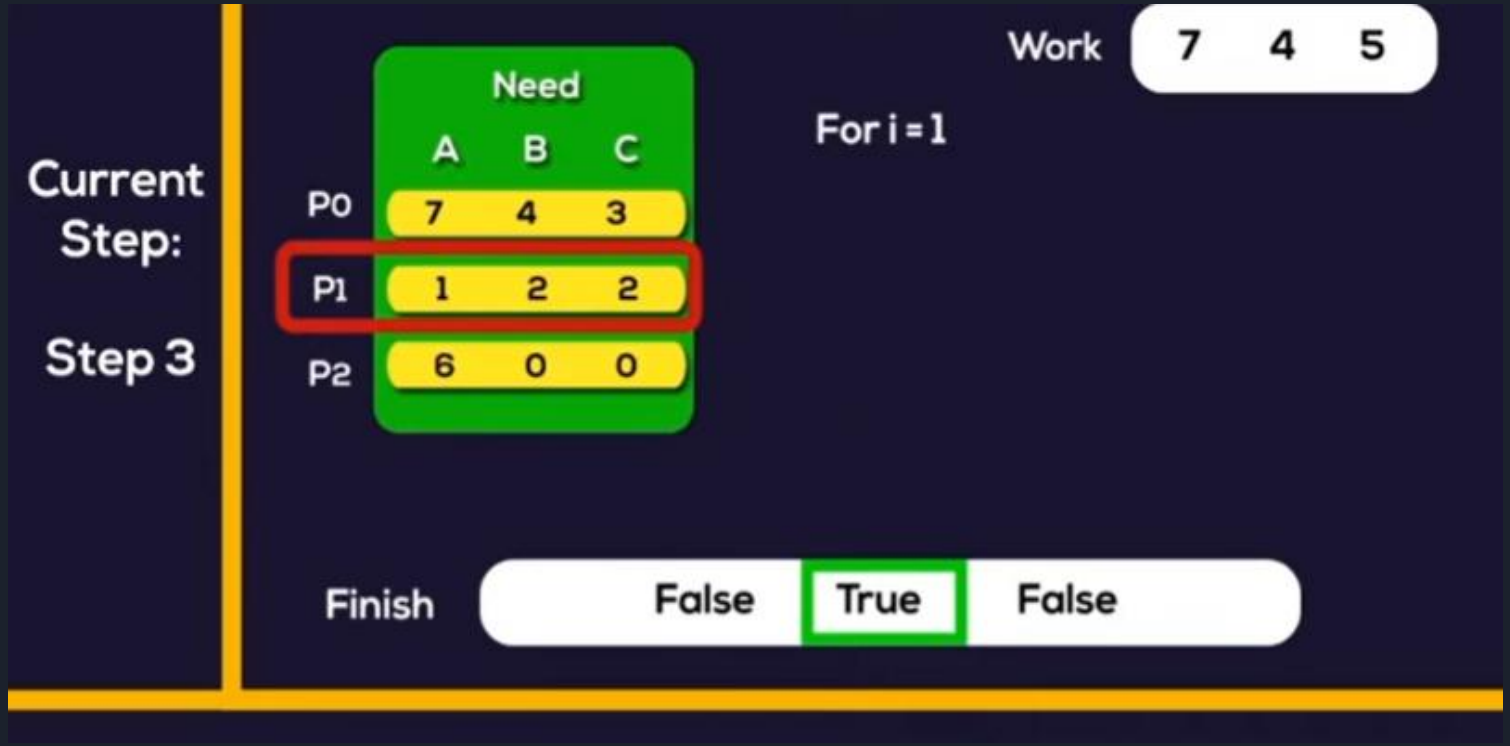
Safety Algorithm



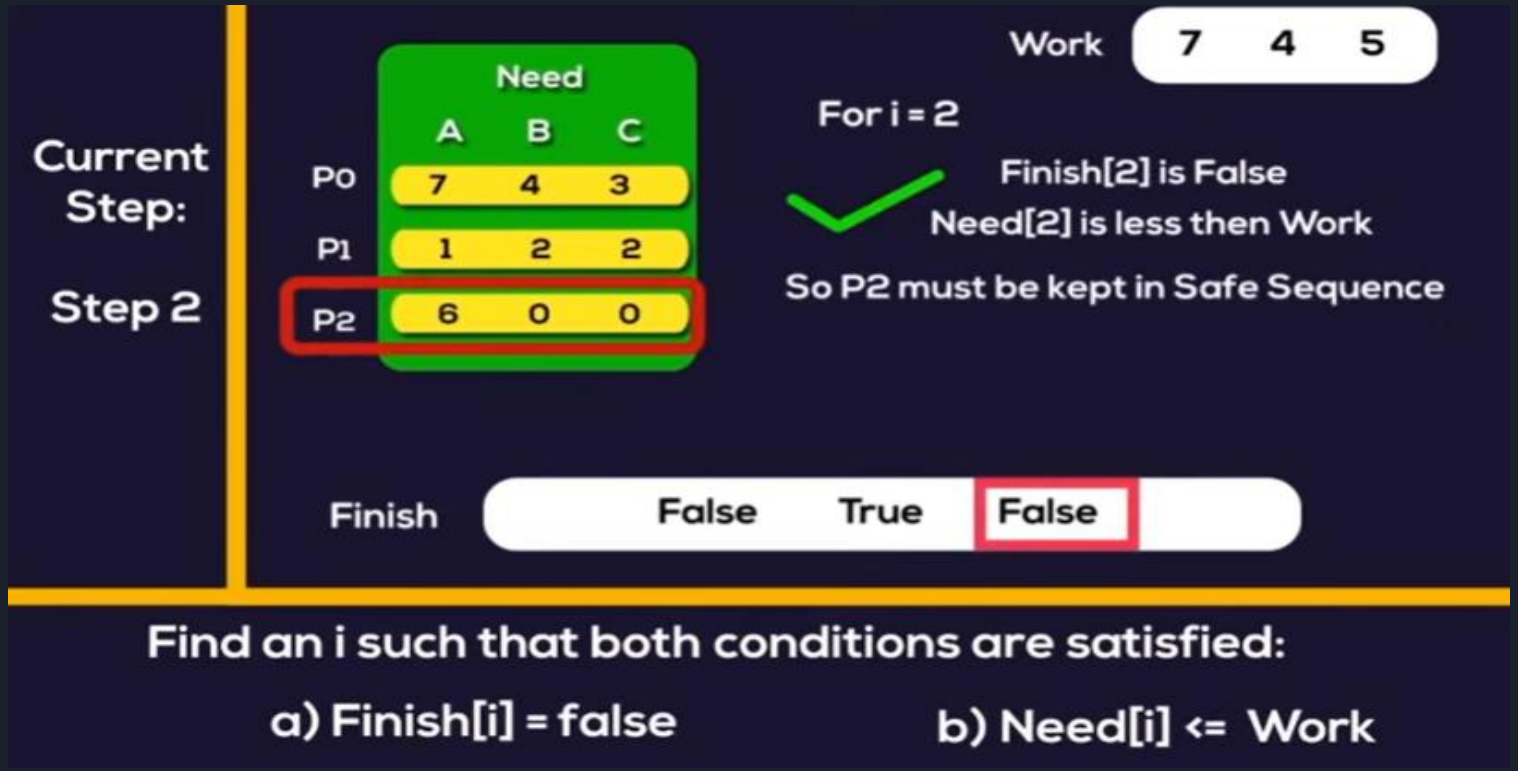
Safety Algorithm



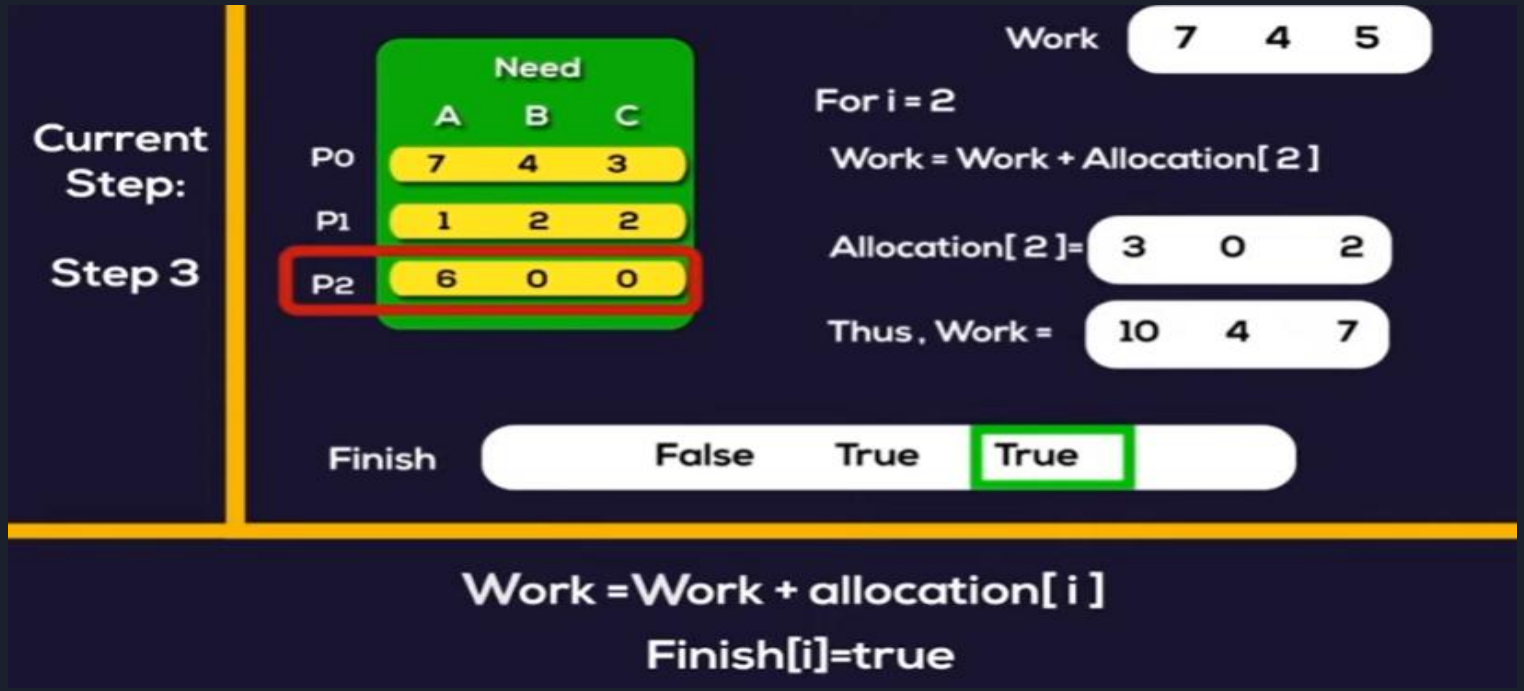
Safety Algorithm



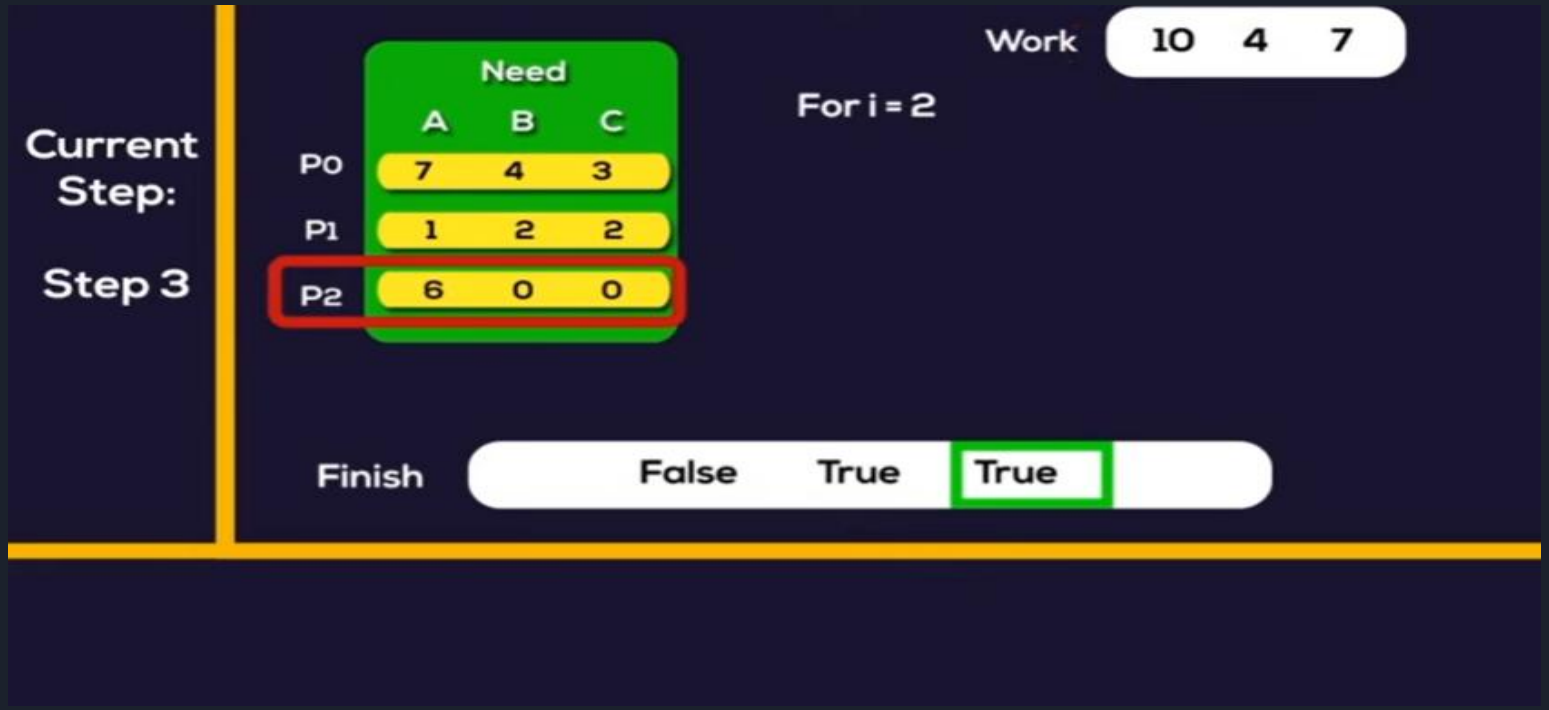
Safety Algorithm



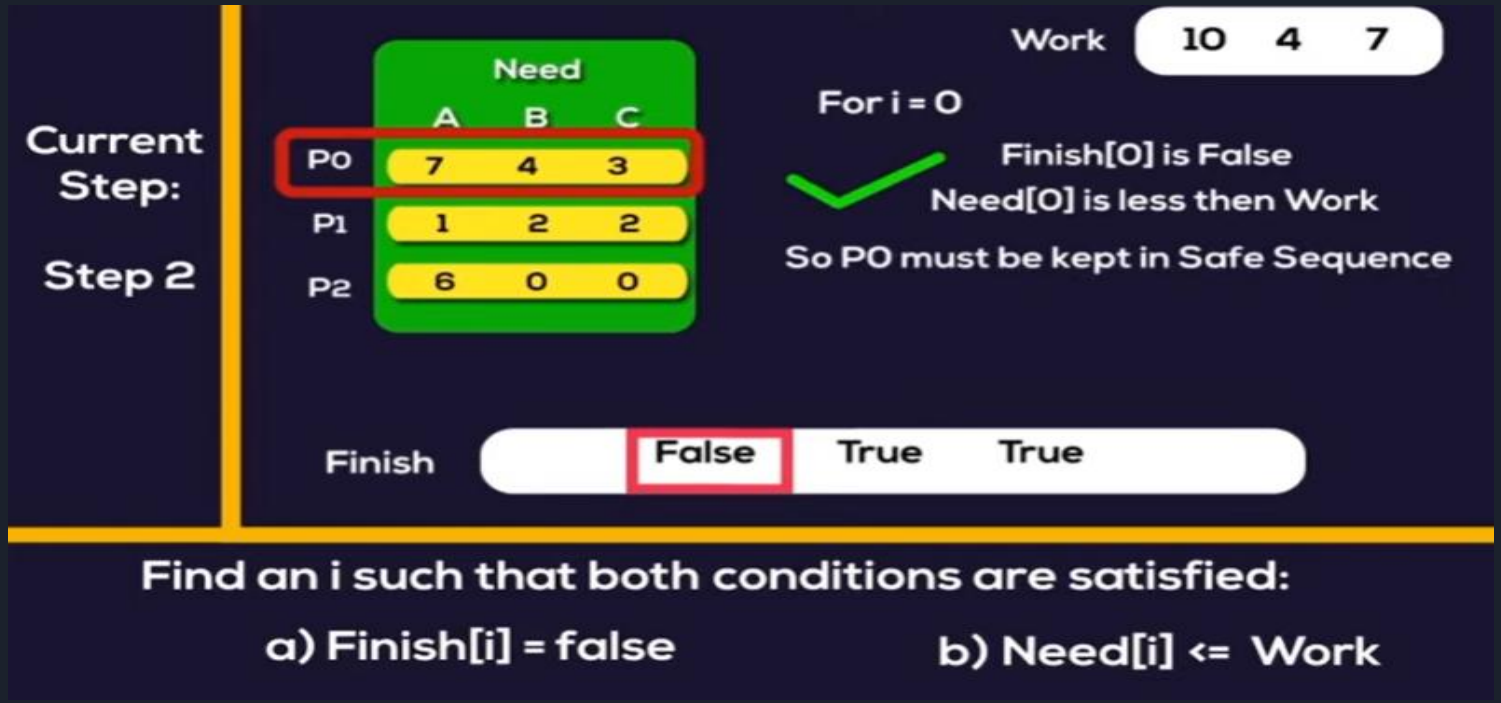
Safety Algorithm



Safety Algorithm

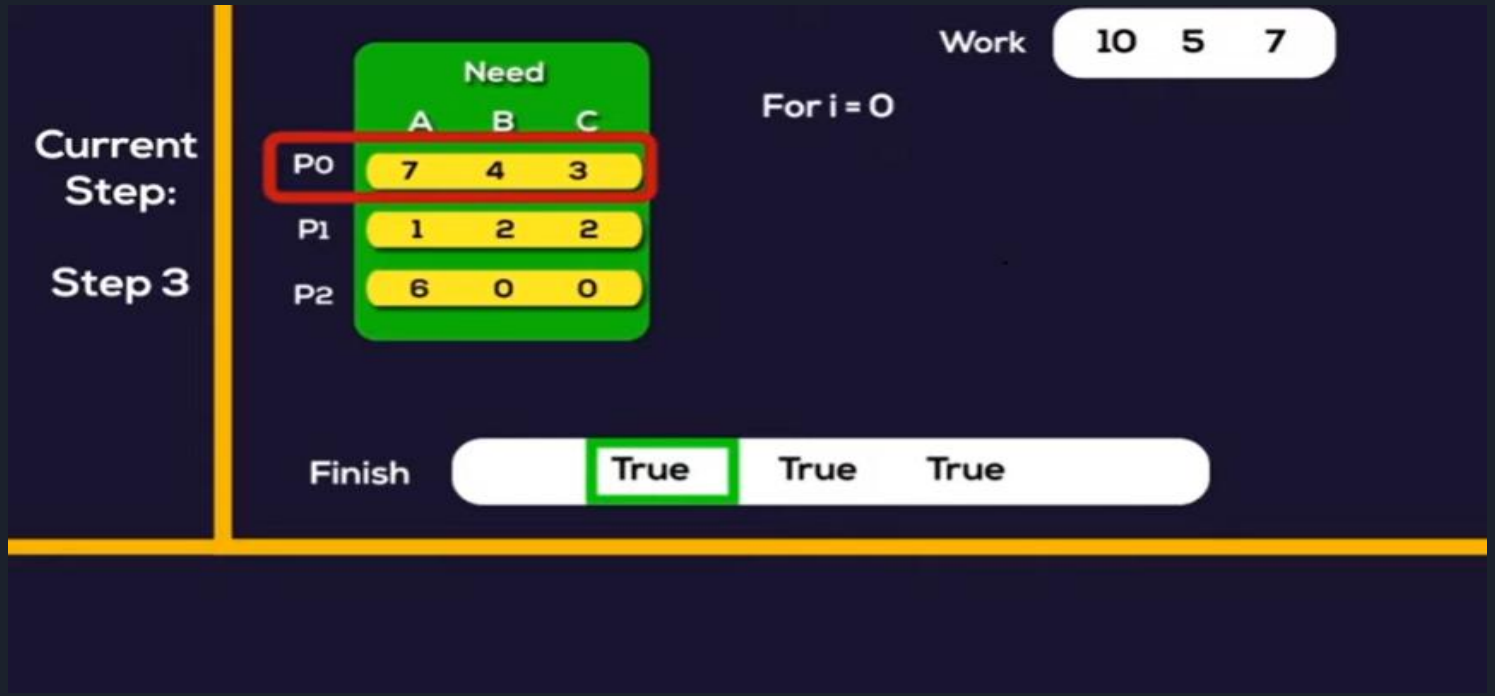


Safety Algorithm



Safety Algorithm

Safety Algorithm



Safety Algorithm
Order of Safe Sequence

<P1,P2,P0>

Resource Allocation Graph (RAG)

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

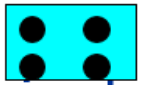
- V is partitioned into 2 types
  - P = {P1, P2,...,Pn} - the set of processes in the system
  - R = {R1, R2,...,Rn} - the set of resource types in the system

- Two kinds of edges
  - Request edge - Directed edge Pi ---> Rj
  - Assignment edge - Directed edge Rj ----> Pi

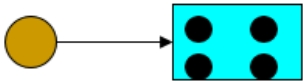
Resource Allocation Graph (RAG)

- Process

- Resource type with 4 instances

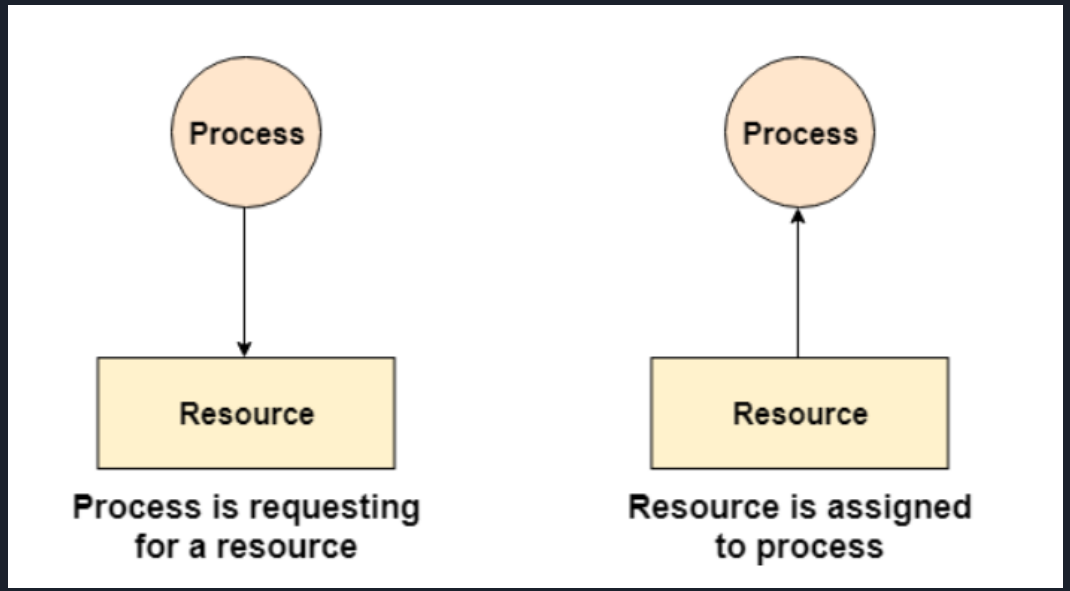


- Pi requests instance of Rj



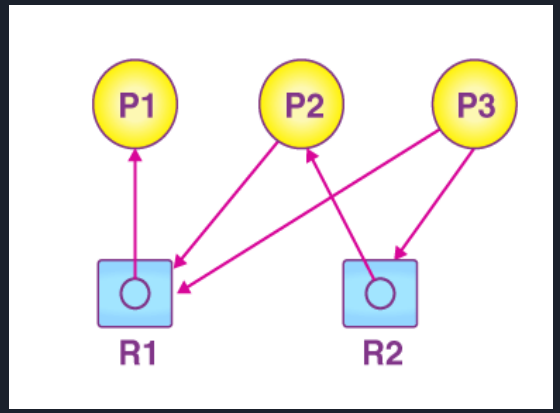
- Pi is holding an instance of Rj

Resource Allocation Graph (RAG)



Resource Allocation Graph (RAG) - Example

- Consider three processes, P1, P2, and P3, as well as two resource types, R1 and R2. Each resource has a single instance.
- According to the graph, P1 is using R1, P2 is holding R2 while waiting for R1, and P3 is waiting for both R1 and R2.
- Because no cycle is generated in the graph, there is no deadlock.



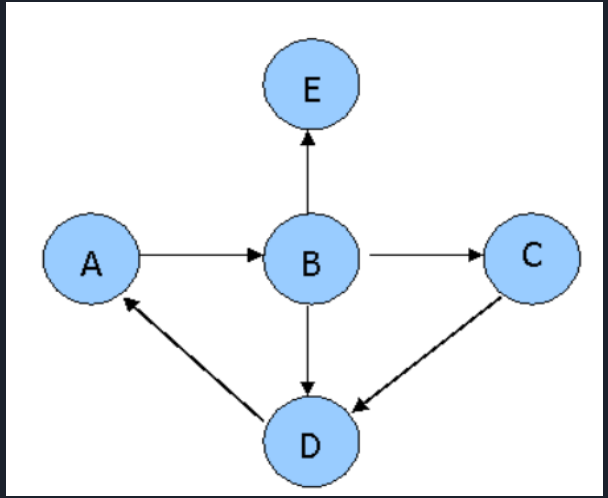
Wait For Graph (WFG)

The Wait For Graph contains,

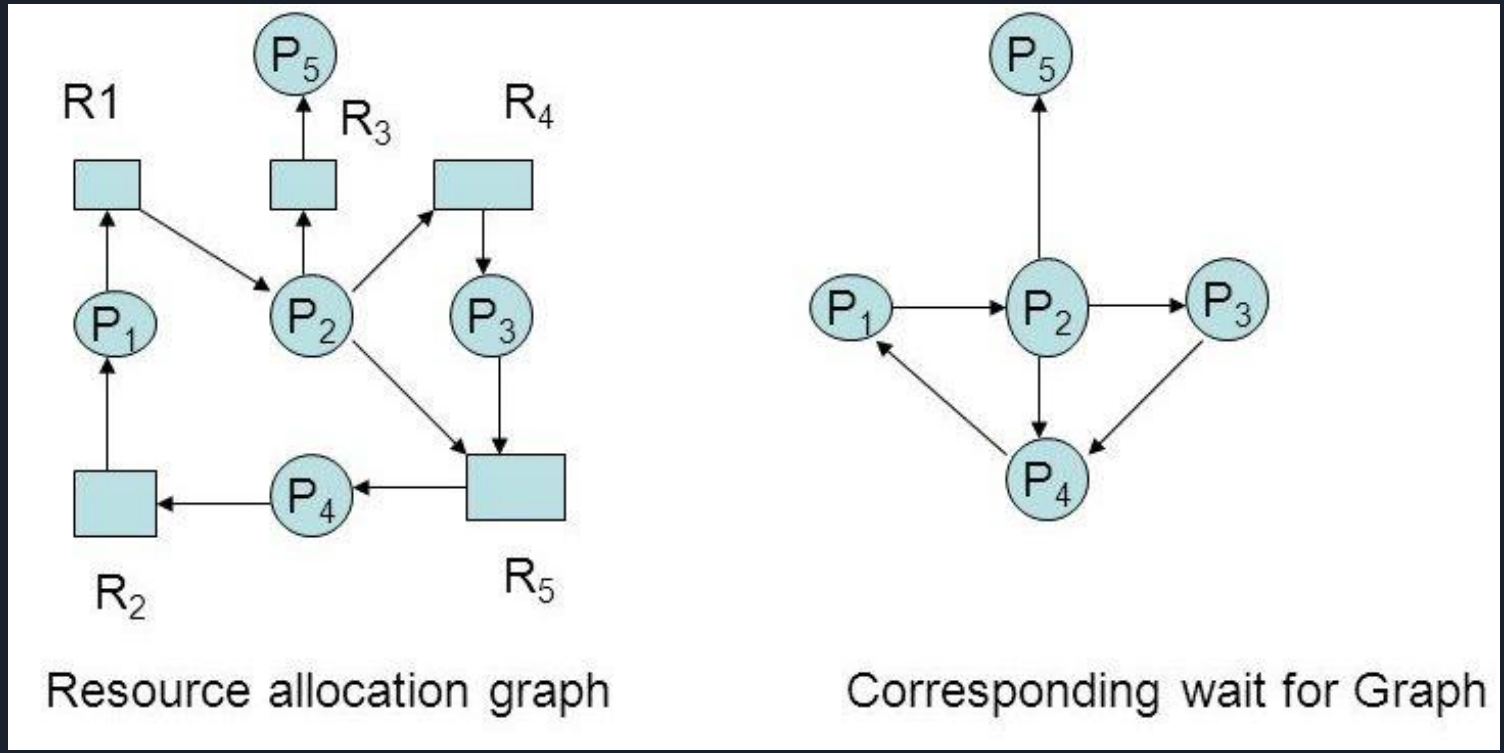
- Nodes, corresponding to processes
- Directed edges, corresponding to a resource held by one process and desired by the other.

- A waits for B,
- B waits for D,
- D waits for A

DEADLOCK



Wait For Graph (WFG)

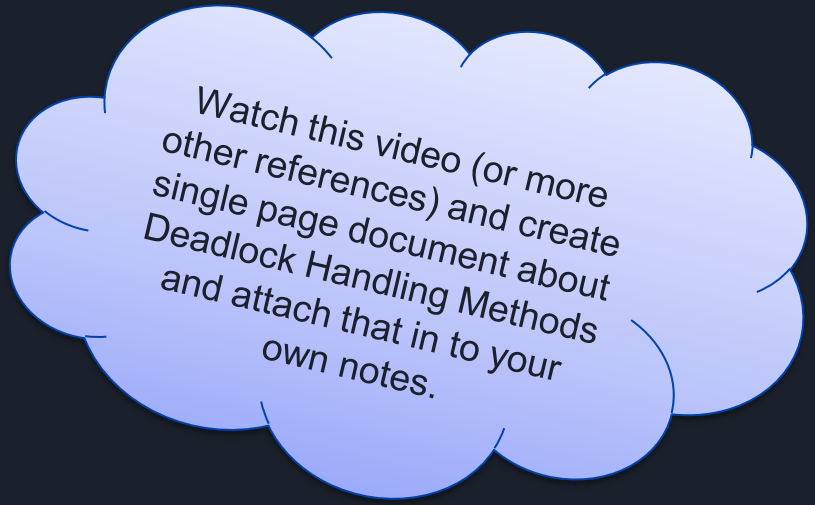


Handling Deadlocks
Methods of Handling Deadlocks

- Deadlock Prevention
- Deadlock Avoidance
- Deadlock Detection and Recovery
- Deadlock Ignorance

Handling Deadlocks
References – Self Learning Activity

Deadlock Handling Methods



Thank You...