University of Nottingham
UK | CHINA | MALAYSIA

# Operating Systems and Concurrency

## Lecture 14: Deadlocks (cont.)

University of Nottingham, Ningbo China, 2024

# Recap

- Definition of deadlocks
- Deadlock occurrence
    - Minimum conditions
- Approaches to dealing with deadlocks
    - Deadlock detection algorithms
    - Deadlock recovery approaches

# Overview
## Today class

- Deadlock Prevention

- Deadlock Avoidance
    - Banker algorithm

# Deadlock Prevention

## Restrain one of the following four conditions

- Deadlock Prevention Restrain one of the following four conditions:
  - Mutual Exclusion
  - Hold and Wait
  - No Preemption
  - Circular Wait

# Deadlock Prevention
## Restrain one of the four conditions

- Deadlock Prevention Restrain one of the following four conditions:
  - Mutual Exclusion:
    - Prevention Strategy: If resources can be shared (e.g., read-only files), they don't require mutual exclusion.
      - However, certain resources (like printers or files that need write access) inherently require mutual exclusion, so this approach doesn't work for all resources.
    - Limitation: This condition cannot be fully eliminated for non-sharable resources, so it's only partially effective in deadlock prevention.

# Deadlock Prevention
## Restrain one of the four conditions

- Deadlock Prevention Restrain one of the following four conditions:

  - Hold and Wait:
    - Description: Prevention Strategy:
      - Option 1: Require processes to request and be allocated all required resources before they begin execution.
        - Pros: Simple to implement.
        - Cons: Leads to low resource utilization since resources may be held for longer than necessary.

      - Option 2: Allow processes to request resources only when they currently hold none.
        - Pros: Reduces the risk of deadlock.
        - Cons: Starvation may occur, as processes might wait indefinitely if they're repeatedly unable to acquire all resources needed.

- Limitation: Both methods have practical drawbacks, like low efficiency or potential starvation.

# Deadlock Prevention
Restrain one of the four conditions

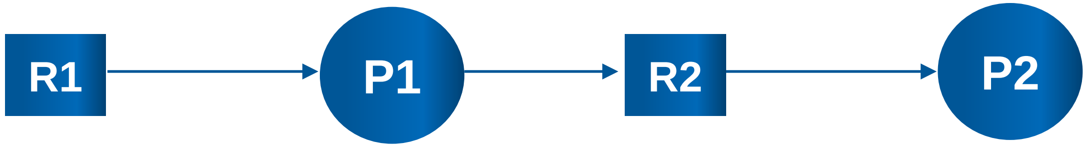
- Deadlock Prevention Restrain one of the following four conditions:

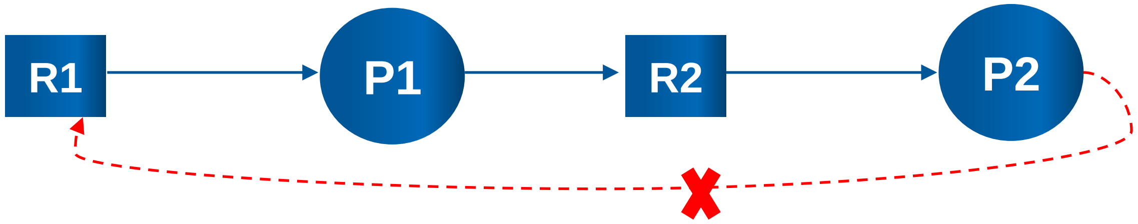
  - No Preemption:
  - Prevention Strategy:
    - If a process holding resources requests additional resources and those resources are not immediately available, the process must release all currently held resources.
      - Example: If Process $P_1$ requests a resource $R_1$ currently held by Process $P_2$, which is waiting for another resource $R_2, R_1$ could be reallocated to $P_1$.


  - Pros: Helps prevent indefinite waiting and potential deadlocks.
  - Cons: May result in process starvation or significant process rollback (restarting from scratch), as processes may be forced to release resources frequently.

# Deadlock Prevention
## Restrain one of the four conditions

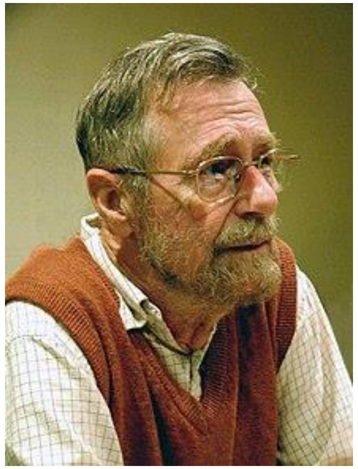
- Deadlock Prevention Restrain one of the following four conditions:
  - Circular Wait:
  - Prevention Strategy: Impose a strict ordering of resources and require that each process requests resources in a specific increasing order.
    - Example: If there are resources $R_1, R_2, \ldots, R_n$ each process must request resources in ascending order (e.g., first $R_1$, then $R_2$, and so on).




  - Pros: Effectively eliminates circular waiting, breaking the cycle required for deadlock.
  - Cons: Processes may need to be restructured to follow the ordering, which can complicate programming and may not always align with the logical flow of tasks.

# Deadlock Avoidance
## Banker's Algorithm (BA)



- Dijkstra's Banker's Algorithm, developed by Edsger Dijkstra in 1965.

- The name "Banker's Algorithm" comes from its analogy to a banking system, where a banker must ensure they retain enough funds to satisfy the needs of all clients while processing loan requests.

- It is modeled on the way a small-town banker might deal with a group of customers to whom he has granted lines of credit. (Years ago, banks did not lend money unless they knew they could be repaid.)

# Deadlock Avoidance

## Key Points of the Banker’s Algorithm in this Banking Analogy

1. Loan limits (maximum needs) help the bank gauge worst-case demands and manage its cash flow.

2. The safe state is crucial: Before lending, the bank checks if it can meet all customers’ maximum demands safely to avoid a shortage.

3. Sequential repayment and reallocation of funds enable the bank to serve multiple customers over time, ensuring no one is denied due to insufficient funds.

# Banker's Algorithm (BA)
## Example: Banking Analogy

What the algorithm does is check to see if granting the request leads to an unsafe state. If so, the request is denied. If granting the request leads to a safe state, it is carried out.

| | Has | Max |
|---|---|---|
| A | 0 | 6 |
| B | 0 | 5 |
| C | 0 | 4 |
| D | 0 | 7 |

Free: 10

(a)

This state is safe

| | Has | Max |
|---|---|---|
| A | 1 | 6 |
| B | 1 | 5 |
| C | 2 | 4 |
| D | 4 | 7 |

Free: 2

(b)

This state is safe because with two units left, the banker can delay any requests except C's, thus letting C finish and release all four of his resources. With four units in hand, the banker can let either D or B have the necessary units, and so on.

| | Has | Max |
|---|---|---|
| A | 1 | 6 |
| B | 2 | 5 |
| C | 2 | 4 |
| D | 4 | 7 |

Free: 1

(c)

Consider what would happen if a request from B for one more unit were granted in (b). We would have situation (c), which is unsafe.

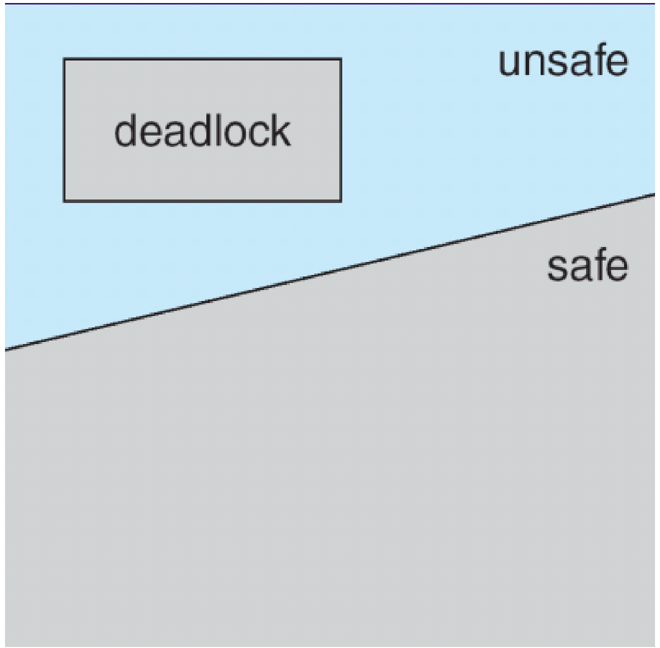
- The banker's algorithm considers each request as it occurs, seeing whether granting it leads to a safe state. If it does, the request is granted; otherwise, it is postponed until later.

# Banker's Algorithm (BA)

- When a new process enters the system, it must declare the maximum number of instances of each resource type that it may need.
- This number may not exceed the total number of resources in the system.
- When a user 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

# Deadlock Avoidance
# Banker's Algorithm (BA): Safe state, Basic facts

- Safe State: A system state where it's possible to find a safe sequence for process execution that guarantees no deadlock.
- 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.



# Banker Algorithm
# Safe State

- Consider the following example for a single resource, assuming that 10 entities exist (E=10) for the given resource (Has (allocation), Max, ):

Allocation

Available

| | Has | Max |
|---|---|---|
| A | 3 | 9 |
| B | 2 | 4 |
| C | 2 | 7 |

Free: 3

(a)

| | Has | Max |
|---|---|---|
| A | 3 | 9 |
| B | 4 | 4 |
| C | 2 | 7 |

Free: 1

(b)

| | Has | Max |
|---|---|---|
| A | 3 | 9 |
| B | 0 | – |
| C | 2 | 7 |

Free: 5

(c)

| | Has | Max |
|---|---|---|
| A | 3 | 9 |
| B | 0 | – |
| C | 7 | 7 |

Free: 0

(d)

| | Has | Max |
|---|---|---|
| A | 3 | 9 |
| B | 0 | – |
| C | 0 | – |

Free: 7

(e)

Fig: Demonstration that the state in (a) is safe (Tanenbaum)

# Banker Algorithm
# Safe State

- Consider the following example (A gets an **extra resource** in (b)), again assuming that 10 instances exist for the given resource:
  - The state in *b* is **not yet deadlocked**, but **will deadlock eventually.**

| | Has | Max |
|---|---|---|
| A | 3 | 9 |
| B | 2 | 4 |
| C | 2 | 7 |

Free: 3
(a)

| | Has | Max |
|---|---|---|
| A | 4 | 9 |
| B | 2 | 4 |
| C | 2 | 7 |

Free: 2
(b)

| | Has | Max |
|---|---|---|
| A | 4 | 9 |
| B | 4 | 4 |
| C | 2 | 7 |

Free: 0
(c)

| | Has | Max |
|---|---|---|
| A | 4 | 9 |
| B | — | — |
| C | 2 | 7 |

Free: 4
(d)

Fig: Demonstration that the state in (b) is not safe. (Tanenbaum)

# Banker's Algorithm (BA) Data Structure to implement BA

- We need the following data structures, where n is the number of processes in the system and m is the number of resource types:
  - **Allocation**. An n × m matrix defines the number of resources of each type currently allocated to each process. If Allocation[i][j] equals k, then process Pi is currently allocated k instances of resource type Rj .
  - **Available**. A vector of length m indicates the number of available resources of each type. If Available[j] equals k, then k instances of resource type Rj are available.
  - **Max**. An n × m **matrix defines the maximum demand of each process.** If Max[i][j] equals k, then process Pi may request at most k instances of resource type Rj .
  - **Need**. An n × m matrix indicates the remaining resource need of each process. If Need[i][j] equals k, then process Pi may need k more instances of resource type Rj to complete its task. Note that Need[i][j] equals Max[i][j] − Allocation[i][j]

- Let say a system
  - has n=2 processes ($p_0$,$p_1$)
  - m=2 resources (printer and scanner), printer has 5 instance and scanner has 4 instance.
    - E=[5,4] Available= [2,1]

| | Allocation | | Max | |
|---|---|---|---|---|
| | Pri | Sca | Pri | Sca |
| $P_0$ | 3 | 2 | 4 | 3 |
| $P_1$ | 0 | 1 | 1 | 4 |

| | Need | |
|---|---|---|
| | Pri | Sca |
| $P_0$ | 1 | 1 |
| $P_1$ | 1 | 3 |

# Deadlock avoidance

## Banker's Algorithm: Steps in the Banker's Algorithm

**1.Initialization**:

1. Each process declares its maximum needs for each resource.
2. The system knows the total resources available and calculates which resources are allocated and which remain available.

**3. Safety Check**:

1. The system verifies if, after the allocation, there exists a safe sequence (a possible order for all processes to complete).
2. If a safe sequence exists, the allocation is confirmed. If not, the process must wait.

**2. Resource Request Handling**:

1. When a process requests resources, the system checks:
   1. **Request ≤ Need**: The requested resources must be within the declared maximum needs.
   2. **Request ≤ Available** : The resources must be available.
2. If these checks pass, the system temporarily allocates the resources

# Baker Algorithm
# Resource request Algorithm

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

1) *If* $Request_i <= Need_i$
   *Goto step (2) ; otherwise, raise an error condition, since the process has exceeded its maximum claim.*
2) *If* $Request_i <= Available$
   *Goto step (3); otherwise,* $P_i$ *must wait, since the resources are not available.*
3) *Have the system pretend to have allocated the requested resources to process* $P_i$ *by modifying the state as follows:*

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

- *If safe : the resources are allocated to Pi*
- *If unsafe:* $P_i$ *must wait, and the old resource-allocation state is restored*

# Banker Algorithm
# Safety Algorithm

- A safe state is one where there is a feasible order for the processes to finish without deadlock.

```
1) Let Work and Finish be vectors of length ‘m’ and ‘n’ respectively.
   Initialize: Work = Available
   Finish[i] = false; for i=1, 2, 3, 4….n
2) Find an i such that both
   a) Finish[i] = false
   b) Need_i <= Work
   if no such i exists goto step (4)
3) Work = Work + Allocation[i]
   Finish[i] = true
   goto step (2)
4) if Finish [i] = true for all i then the system is in a safe state
```

# Banker Algorithm
# An Illustrative Example

- Consider a system with five processes P0 through P4 and three resource types A, B, and C.
- Resource type A has ten instances, resource type B has five instances, and resource type C has seven instances.
- Suppose that, at time T0, the following snapshot of the system has been taken.
  - What will be the content of the Need matrix?
  - Is this state safe?
  - What will happen if process P1 requests one additional instance of resource type A and two instances of resource type C?

| | Allocation | | | Max | | | Need | | | Available | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | 3 | 3 | 2 |
| P1 | 0 | 1 | 0 | 7 | 5 | 3 | | | | | | |
| P2 | 2 | 0 | 0 | 3 | 2 | 2 | | | | | | |
| P3 | 3 | 0 | 2 | 9 | 0 | 2 | | | | | | |
| P4 | 2 | 1 | 1 | 2 | 2 | 2 | | | | | | |
| P5 | 0 | 0 | 2 | 4 | 3 | 3 | | | | | | |

# Banker Algorithm
# An Illustrative Example: Need Matrix

Need [i, j] = Max [i, j] – Allocation [i, j] So, the content of Need Matrix is:

| | Allocation | | | Max | | | Need | | | Available | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | 3 | 3 | 2 |
| P0 | 0 | 1 | 0 | 7 | 5 | 3 | 7 | 4 | 3 | | | |
| 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 | 0 | 1 | 1 | | | |
| P4 | 0 | 0 | 2 | 4 | 3 | 3 | 4 | 3 | 1 | | | |

# Banker Algorithm
# An Illustrative Example: Is it safe?

| | Allocation | | | Max | | | Need | | | Available | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | 3 | 3 | 2 |
| P0 | 0 | 1 | 0 | 7 | 5 | 3 | 7 | 4 | 3 | | | |
| 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 | 0 | 1 | 1 | | | |
| P4 | 0 | 0 | 2 | 4 | 3 | 3 | 4 | 3 | 1 | | | |

**Step 1 of Safety Algo**

m=3, n=5

Work = Available

Work = | 3 | 3 | 2 |

| | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| Finish = | false | false | false | false | false |

**Step 2**

For i = 0

$Need_0$ = 7, 4, 3

Finish [0] is false and $Need_0$ > Work (7,4,3 > 3,3,2)

So $P_0$ must wait

But Need ≤ Work

**Step 2**

For i = 1

$Need_1$ = 1, 2, 2

Finish [1] is false and $Need_1$ <= Work (1,2,2 <= 3,3,2)

So $P_1$ must be kept in safe sequence

**Step 3**

Work = Work + $Allocation_1$ (3, 3, 2 + 2, 0, 0)

| | A | B | C |
|---|---|---|---|
| Work = | 5 | 3 | 2 |

| | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| Finish = | false | true | false | false | false |

**Step 2**

For i = 2

$Need_2$ = 6 , 0, 0

Finish [2] is false and $Need_2$ > Work (6, 0, 0 > 5,3, 2)

So $P_2$ must wait

**Step 2**

For i=3

$Need_3$ = 0, 1, 1

Finish [3] = false and $Need_3$ < Work (0, 1, 1 < 5, 3, 2)

So $P_3$ must be kept in safe sequence

**Step 3**

Work = Work + $Allocation_3$ (5, 3, 2 + 2, 1, 1)

| | A | B | C |
|---|---|---|---|
| Work = | 7 | 4 | 3 |

| | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| Finish = | false | true | false | true | false |

**Step 2**

For i = 4

$Need_4$ = 4, 3, 1

Finish [4] = false and $Need_4$ <= Work (4, 3, 1 <= 7, 4, 3)

So $P_4$ must be kept in safe sequence

**Step 3**

Work = Work + $Allocation_4$ (7, 4, 3 + 0, 0, 2)

| | A | B | C |
|---|---|---|---|
| Work = | 7 | 4 | 5 |

| | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| Finish = | false | true | false | true | true |

**Step 2**

For i = 0

$Need_0$ = 7, 4, 3

Finish [0] is false and Need <= Work (7, 4, 3 <= 7, 4, 5)

So $P_0$ must be kept in safe sequence

**Step 3**

Work = Work + $Allocation_0$ (7, 4, 5 + 0, 1 , 0)

| | A | B | C |
|---|---|---|---|
| Work = | 7 | 5 | 5 |

| | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| Finish = | true | true | false | true | true |

**Step 2**

For i = 2

$Need_2$ = 6 , 0, 0

Finish [2] is false and $Need_2$ <= Work (6, 0, 0 <= 7, 5, 5)

So $P_2$ must be kept in safe sequence

**Step 3**

Work = Work + $Allocation_2$ (7, 5, 5 + 3, 0, 2)

| | A | B | C |
|---|---|---|---|
| Work = | 10 | 5 | 7 |

| | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| Finish = | true | true | true | true | true |

**Step 4**

Finish [i] = true for 0 ≤ i ≤ n

Hence the system is in Safe state

The safe sequence is $P_1$, $P_3$ , $P_4$ , $P_0$, $P_2$

# Banker Algorithm
# An Illustrative Example: What will happen if process P1

- What will happen if process P1 requests one additional instance of resource type A and two instances of resource type C?

| | Allocation | | | Max | | | Need | | | Available | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | 2 | 3 | 0 |
| P0 | 0 | 1 | 0 | 7 | 5 | 3 | 7 | 4 | 3 | | | |
| P1 | 3 | 0 | 2 | 3 | 2 | 2 | 0 | 2 | 0 | | | |
| P2 | 3 | 0 | 2 | 9 | 0 | 2 | 6 | 0 | 0 | | | |
| P3 | 2 | 1 | 1 | 2 | 2 | 2 | 0 | 1 | 1 | | | |
| P4 | 0 | 0 | 2 | 4 | 3 | 3 | 4 | 3 | 1 | | | |

$$Request_1 = \overset{A}{1}, \overset{B}{0}, \overset{C}{2}$$

To decide whether the request is granted we use Resource Request algorithm

Step 1

$\overset{1,\,0,\,2}{Request_1} <= \overset{1,\,2,\,2}{Need_1}$ ✔

Step 2

$\overset{1,\,0,\,2}{Request_1} <= \overset{3,\,3,\,2}{Available}$ ✔

Step 3

Available = Available – $Request_1$

$Allocation_1 = Allocation_1 + Request_1$

$Need_1 = Need_1 - Request_1$

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

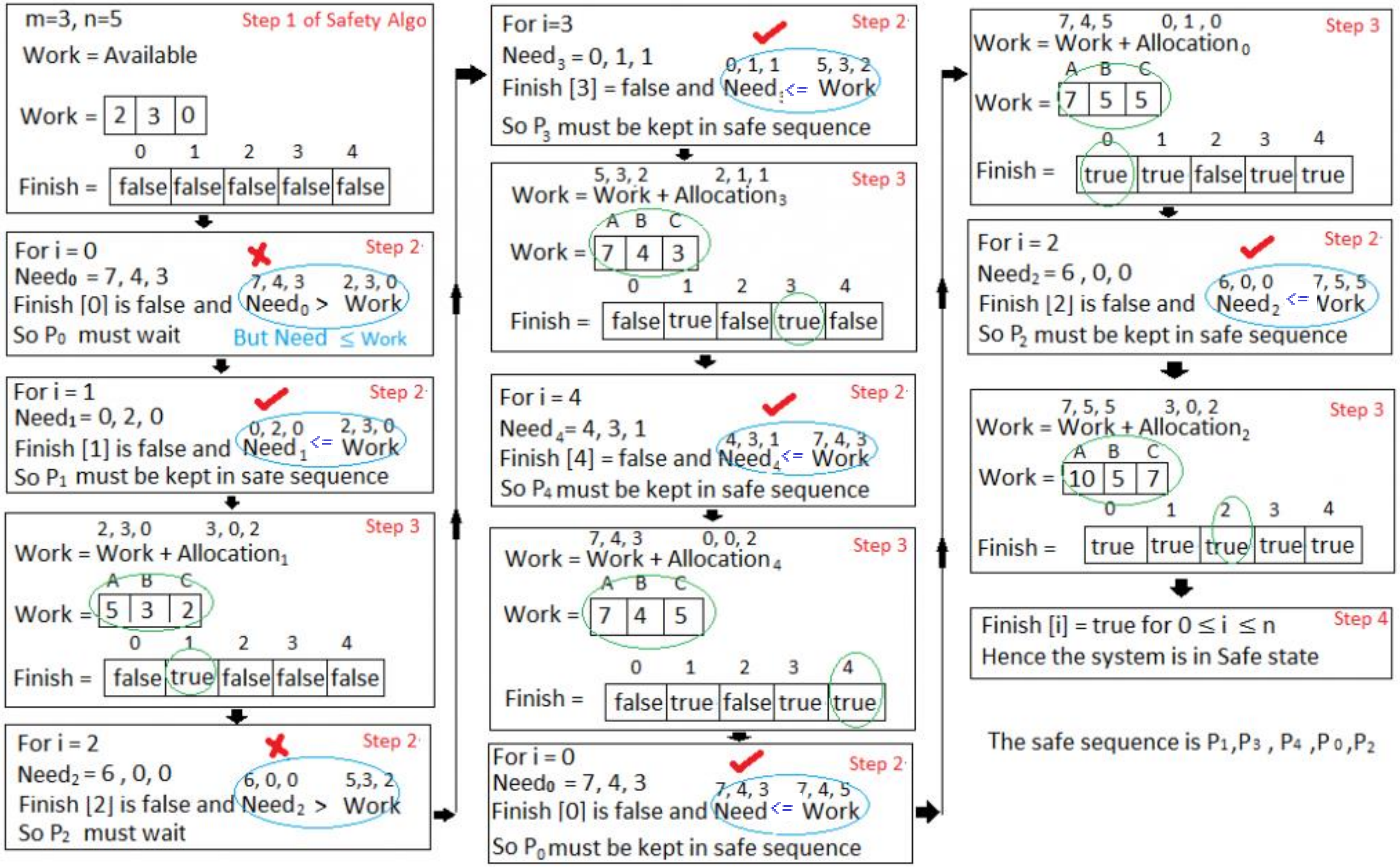
[ 3,3,2] – [1,0,2]= [2,3,0]

[ 2,0,0] + [1,0,2]= [3,0,2]

[ 1,2,2] – [1,0,2]= [0,2,0]

# Banker Algorithm
# An Illustrative Example: What will happen if process P1

- We must determine whether this new system state is safe. To do so, we again execute Safety algorithm on the above data structures.




| | Allocation | | | Max | | | Need | | | Available | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | 2 | 3 | 0 |
| P0 | 0 | 1 | 0 | 7 | 5 | 3 | 7 | 4 | 3 | | | |
| P1 | 3 | 0 | 2 | 3 | 2 | 2 | 0 | 2 | 0 | | | |
| P2 | 3 | 0 | 2 | 9 | 0 | 2 | 6 | 0 | 0 | | | |
| P3 | 2 | 1 | 1 | 2 | 2 | 2 | 0 | 1 | 1 | | | |
| P4 | 0 | 0 | 2 | 4 | 3 | 3 | 4 | 3 | 1 | | | |

# Homework

- When the system is in this state if P4 request for R(3,3,0).Can it be granted?

| | Allocation | | | Max | | | Need | | | Available | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | 2 | 3 | 0 |
| P0 | 0 | 1 | 0 | 7 | 5 | 3 | 7 | 4 | 3 | | | |
| P1 | 3 | 0 | 2 | 3 | 2 | 2 | 0 | 2 | 0 | | | |
| P2 | 3 | 0 | 2 | 9 | 0 | 2 | 6 | 0 | 0 | | | |
| P3 | 2 | 1 | 1 | 2 | 2 | 2 | 0 | 1 | 1 | | | |
| P4 | 0 | 0 | 2 | 4 | 3 | 3 | 4 | 3 | 1 | | | |

- P4 cannot be granted, since the resources are not available.

# Homework

- A request for (0,2,0) by P0 cannot be granted, even though the resources are available, since the resulting state is unsafe.  **Show it. Home Work!**

| | Allocation | | | Max | | | Need | | | Available | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | 2 | 3 | 0 |
| P0 | 0 | 1 | 0 | 7 | 5 | 3 | 7 | 4 | 3 | | | |
| P1 | 3 | 0 | 2 | 3 | 2 | 2 | 0 | 2 | 0 | | | |
| P2 | 3 | 0 | 2 | 9 | 0 | 2 | 6 | 0 | 0 | | | |
| P3 | 2 | 1 | 1 | 2 | 2 | 2 | 0 | 1 | 1 | | | |
| P4 | 0 | 0 | 2 | 4 | 3 | 3 | 4 | 3 | 1 | | | |

# Recommended readings

- Modern Operating Systems (Tanenbaum): **Chapter 6(6.5)**
- Operating System Concepts (Silberschatz): **Chapter 7(7.5)**
- Operating Systems: Internals and Design Principles (Starlings): **Chapter 6(6.3)**