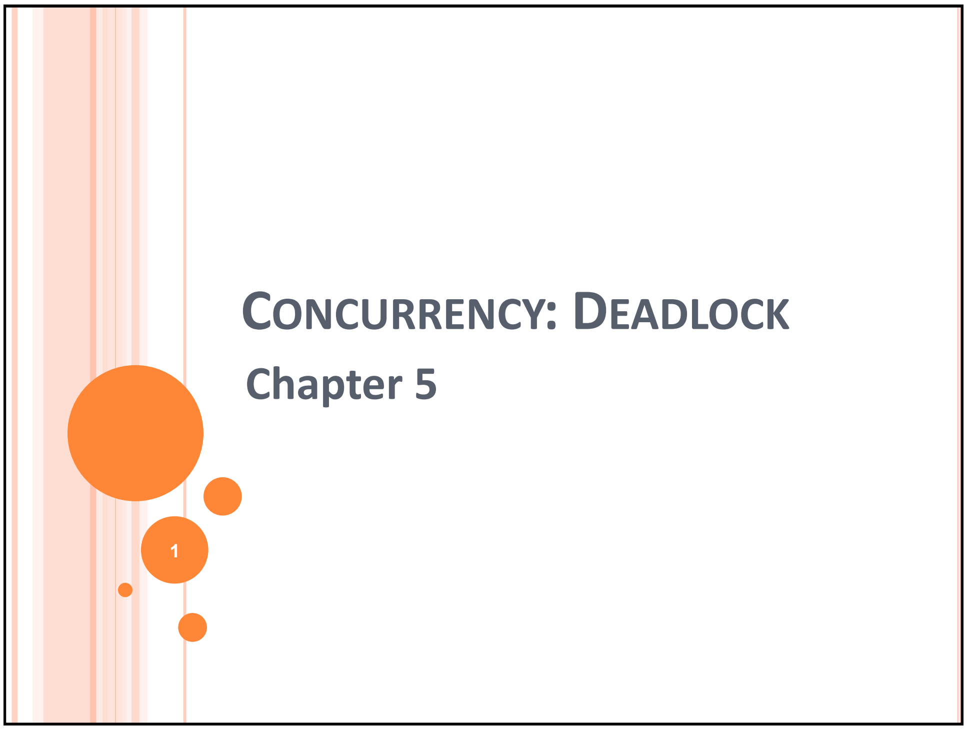
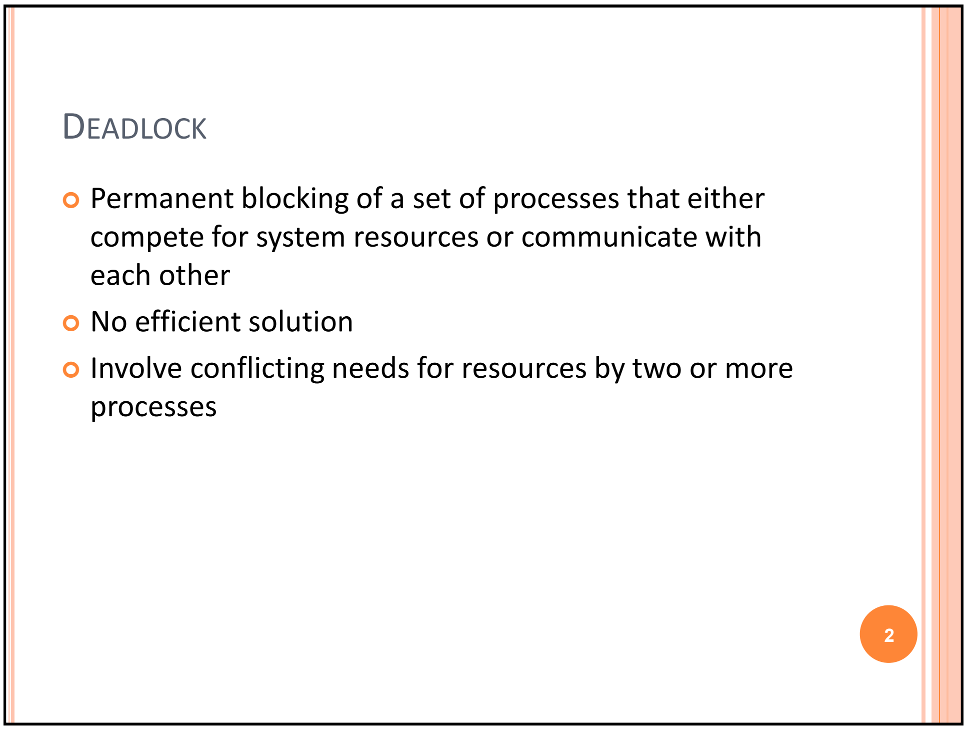


# CONCURRENCY: DEADLOCK

Chapter 5

1

1



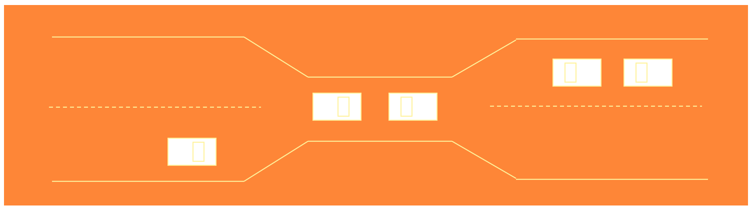
## DEADLOCK

- Permanent blocking of a set of processes that either compete for system resources or communicate with each other
- No efficient solution
- Involve conflicting needs for resources by two or more processes

2

2

## Bridge Crossing Example



- Traffic only in one direction.
- Each section of a bridge can be viewed as a resource.
- If a deadlock occurs, it can be resolved if one car backs up (preempt resources and rollback).
- Several cars may have to be backed up if a deadlock occurs.
- Starvation is possible.

3

3

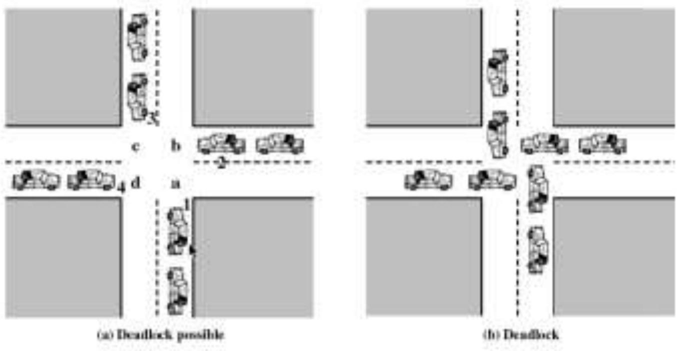
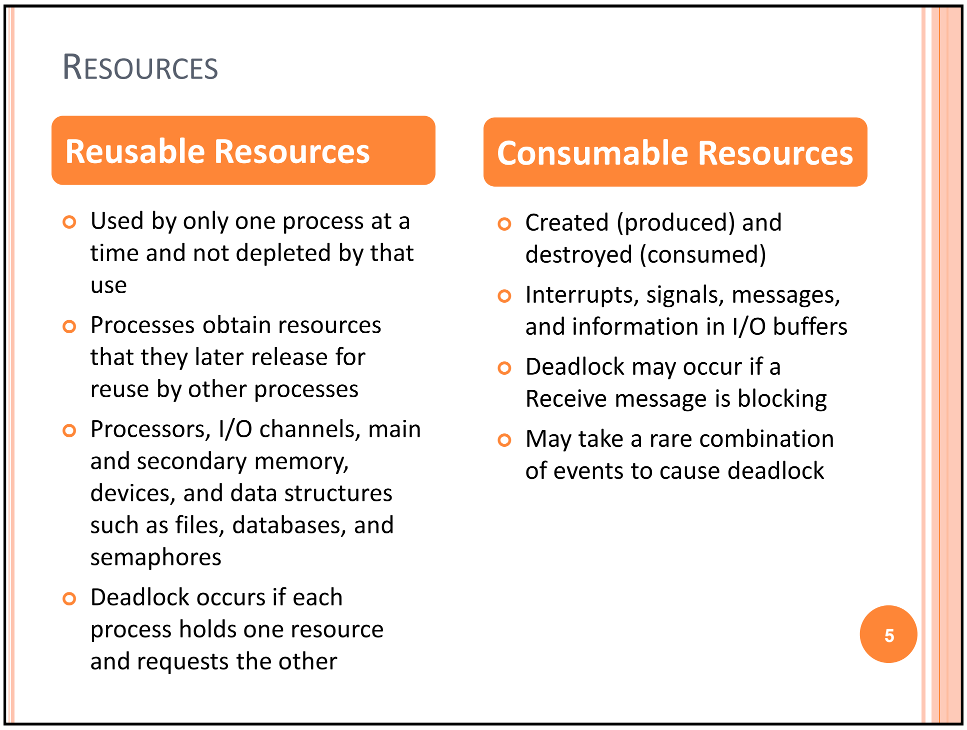


Figure 6.1 Illustration of Deadlock

4

4



## RESOURCES

**Reusable Resources**

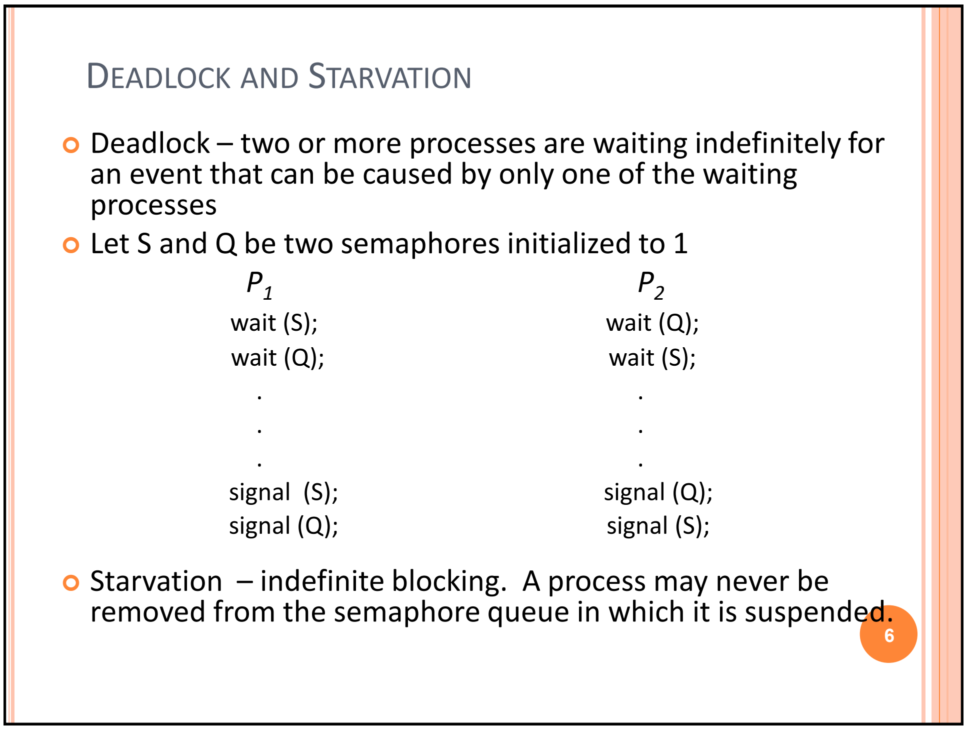
- Used by only one process at a time and not depleted by that use
- Processes obtain resources that they later release for reuse by other processes
- Processors, I/O channels, main and secondary memory, devices, and data structures such as files, databases, and semaphores
- Deadlock occurs if each process holds one resource and requests the other

**Consumable Resources**

- Created (produced) and destroyed (consumed)
- Interrupts, signals, messages, and information in I/O buffers
- Deadlock may occur if a Receive message is blocking
- May take a rare combination of events to cause deadlock

5

5



## DEADLOCK AND STARVATION

- Deadlock – two or more processes are waiting indefinitely for an event that can be caused by only one of the waiting processes
- Let S and Q be two semaphores initialized to 1

```
        P1                      P2
    wait (S);               wait (Q);
    wait (Q);               wait (S);
        .                       .
        .                       .
        .                       .
    signal  (S);            signal (Q);
    signal (Q);             signal (S);
```

- Starvation – indefinite blocking. A process may never be removed from the semaphore queue in which it is suspended.

6

6

## Example of Deadlock

**Process P**

| **Step** | **Action** |
|---|---|
| $p_0$ | Request (D) |
| $p_1$ | Lock (D) |
| $p_2$ | Request (T) |
| $p_3$ | Lock (T) |
| $p_4$ | Perform function |
| $p_5$ | Unlock (D) |
| $p_6$ | Unlock (T) |

**Process Q**

| **Step** | **Action** |
|---|---|
| $q_0$ | Request (T) |
| $q_1$ | Lock (T) |
| $q_2$ | Request (D) |
| $q_3$ | Lock (D) |
| $q_4$ | Perform function |
| $q_5$ | Unlock (T) |
| $q_6$ | Unlock (D) |

**Figure 6.4 Example of Two Processes Competing for Reusable Resources**

What happen when execution of $p_0p_1q_0q_1p_2q_2$ ?

7

## Another Example of Deadlock

- Space is available for allocation of 200Kbytes, and the following sequence of events occur

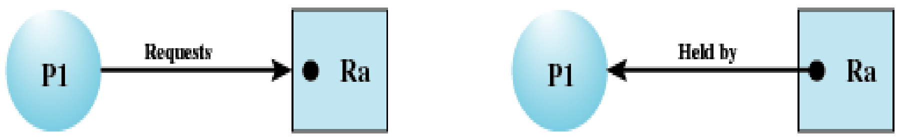
| **P1** | **P2** |
|---|---|
| . . . | . . . |
| **Request 80 Kbytes;** | **Request 70 Kbytes;** |
| . . . | . . . |
| **Request 60 Kbytes;** | **Request 80 Kbytes;** |

- If P1 is given 80Kb and P2 gets 70Kb, then…
- Deadlock occurs if both processes progress to their second request – both processes’ requests cannot be granted as there is insufficient space.
- Each process must wait for the other to release space → deadlock.

8

## Resource Allocation Graphs

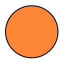
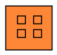
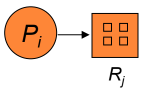
- Directed graph that depicts a state of the system of resources and processes

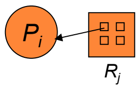


(a) Resouce is requested (b) Resource is held

9

## Resource-Allocation Graph (Cont.)

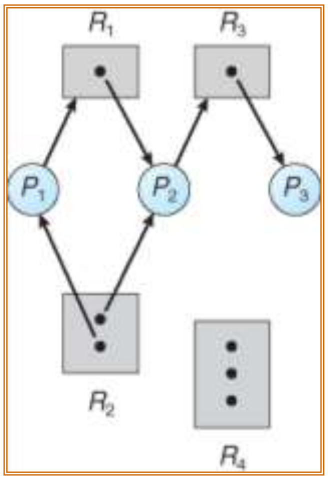
- Process 
- Resource Type with 4 instances 
- $P_i$ requests instance of $R_j$ 

- $P_i$ is holding an instance of $R_j$ 

10

10

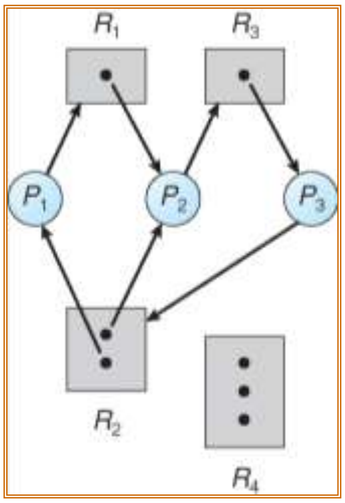
## Example of a Resource Allocation Graph



11

11

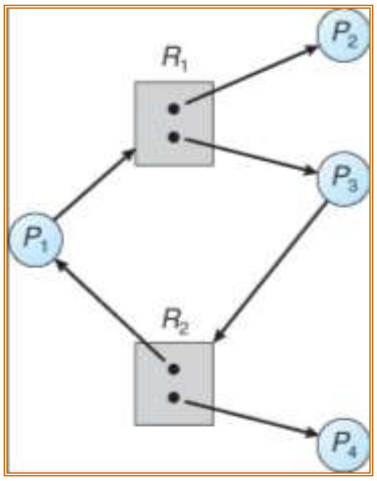
## Resource Allocation Graph With A Deadlock



12

12

## RESOURCE ALLOCATION GRAPH WITH A CYCLE BUT NO DEADLOCK



13

## RESOURCE ALLOCATION GRAPHS

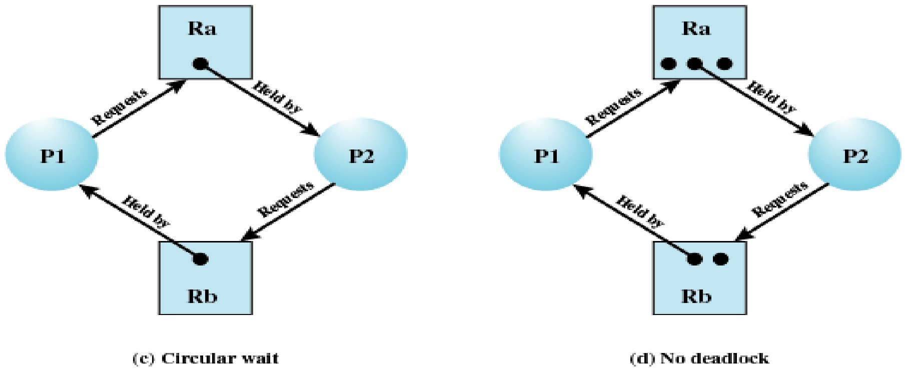


**Figure 6.5 Examples of Resource Allocation Graphs**

14

## Wait-for Graphs

- Simplified resource allocation graphs
- Only shows processes, no resources shown, i.e. which process is waiting for which.

## Wait-for Graphs

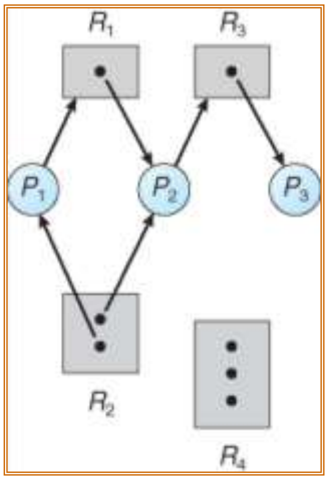
$$\begin{tikzcd}
 & P1 \arrow[dr] & \\
R1 \arrow[ur] & & R2 \arrow[dl] \\
 & P2 \arrow[ul] &
\end{tikzcd}$$

Resource allocation graph

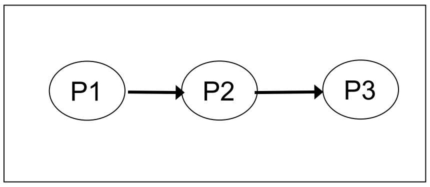
$$\begin{tikzcd}
P1 \arrow[d, bend left] \\
P2 \arrow[u, bend left]
\end{tikzcd}$$

Wait-for graph

## Wait-for Graphs



Resource allocation graph

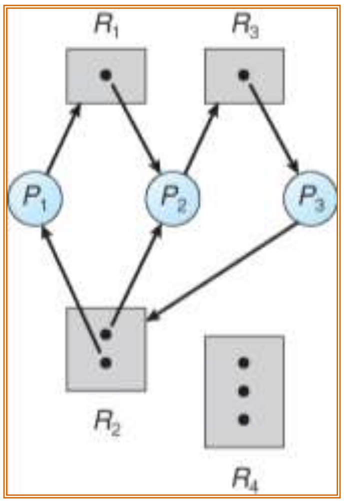


Wait-for graph

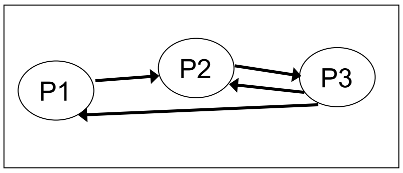
17

17

## Wait-for Graphs



Resource allocation graph



Wait-for graph

18

18

## Resource-Allocation Graph and Wait-for Graph

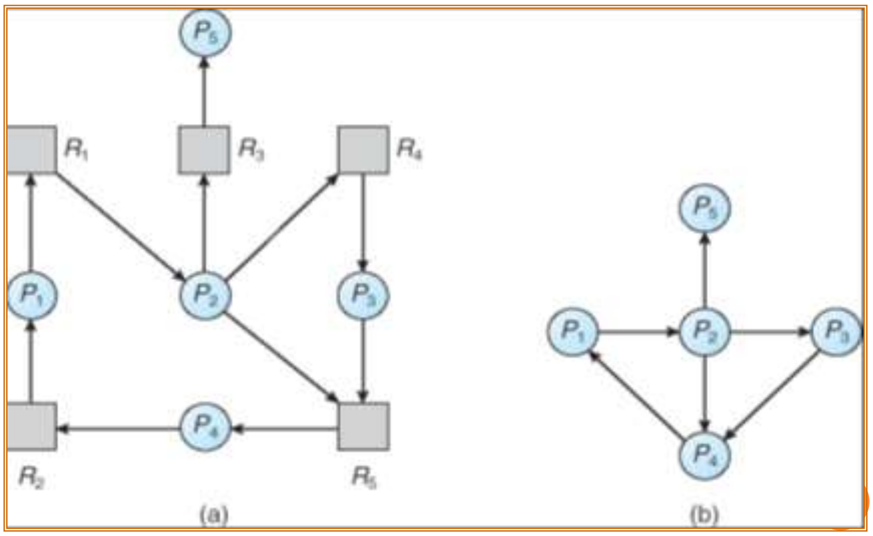
- P1 holds R2, and requests R1
- P2 holds R1, and requests R3, R4 & R5
- P3 holds R4, and requests R5
- P4 holds R5, and requests R2
- P5 holds R3

- Draw the resource allocation graph – combined in ONE graph

19

19

## Resource-Allocation Graph and Wait-for Graph



Resource-Allocation Graph Corresponding wait-for graph

20

## Conditions for Deadlock

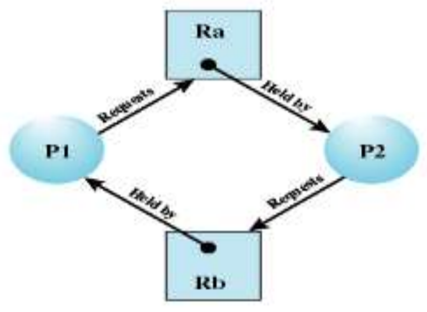
1. Mutual exclusion
   - At least one resource must be held in a non-sharable mode; that is only one process at a time can use the resource.
2. Hold-and-wait
   - A process must be holding at least one resource and waiting to acquire additional resources that are currently being held by other processes.
3. No preemption
   - No resource can be forcibly removed from a process holding it; a resource can be released only voluntarily by the process holding it.

21

21

## Conditions for Deadlock

4. Circular wait
   - A closed chain of processes exists, such that each process holds at least one resource needed by the next process in the chain (P0 → P1, P1→ P2,…., Pn-1 → Pn, and Pn →P0)



(c) Circular wait

22

22

## Possibility of Deadlock

1. Mutual Exclusion
2. No preemption
3. Hold and wait

23

23

## Existence of Deadlock

1. Mutual Exclusion
2. No preemption
3. Hold and wait
4. Circular wait

Deadlock exists

24

24

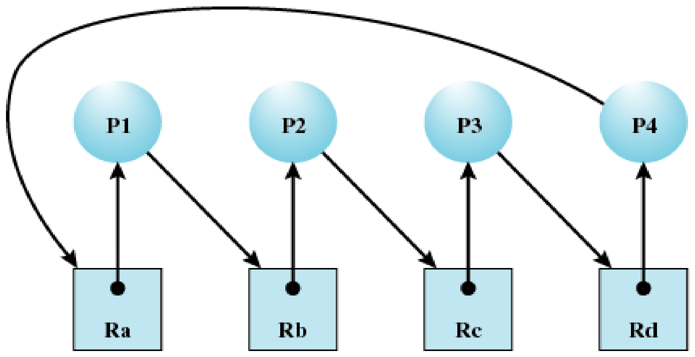


**Figure 6.6 Resource Allocation Graph for Figure 6.1b**

25

25

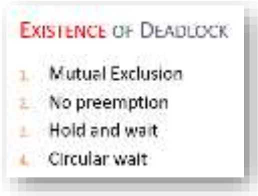
## Three Approaches Dealing with Deadlock

- Prevention
  - Adopting a policy that eliminates 1 of the conditions of deadlock.
- Avoidance
  - Making the appropriate dynamic choices based on the current state of resource allocation.
- Detection
  - Detect the presence of deadlock and take action to recover.

26

26

## Deadlock Prevention


Existence of Deadlock

1. Mutual Exclusion
2. No preemption
3. Hold and wait
4. Circular wait

Restrain the ways request can be made.

1. Mutual Exclusion
   - Must be supported by the operating system
   - not required for sharable resources; must hold for nonsharable resources.

2. Hold and Wait
   - must guarantee that whenever a process requests a resource, it does not hold any other resources.
   - Require a process request all of its required resources at one time, i.e. 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

3. No Preemption
   - Process must release resource and request again
   - Operating system may preempt a process to require it releases its resources
   - 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.

4. Circular Wait
   - impose a total ordering of all resource types, and require that each process requests resources in an increasing order of enumeration.
   - Define a linear ordering of resource types

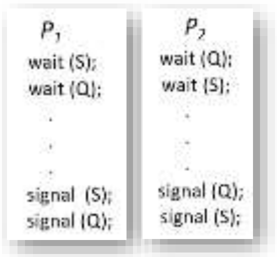
     e.g $R_1$=2 , $R_2$=5 , $R_3$=6 , $R_4$=8

## Deadlock Avoidance

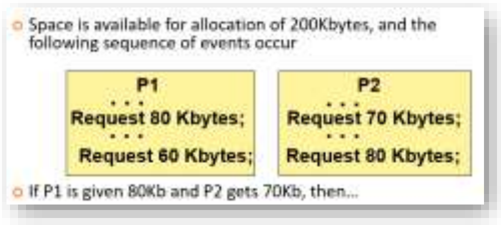
- A decision is made dynamically whether the current resource allocation request will, if granted, potentially lead to a deadlock
- Requires knowledge of future process request

## Two Approaches to Deadlock Avoidance

1. Do not start a process if its demands might lead to deadlock
2. Do not grant an incremental resource request to a process if this allocation might lead to deadlock



```
P1                P2
wait (S);         wait (Q);
wait (Q);         wait (S);
  .                 .
  .                 .
  .                 .
signal (S);       signal (Q);
signal (Q);       signal (S);
```



- Space is available for allocation of 200Kbytes, and the following sequence of events occur

| P1 | P2 |
|---|---|
| . . . | . . . |
| Request 80 Kbytes; | Request 70 Kbytes; |
| . . . | . . . |
| Request 60 Kbytes; | Request 80 Kbytes; |

- If P1 is given 80Kb and P2 gets 70Kb, then...

## Resource Allocation Denial

- Referred to as the banker's algorithm
- State of the system is the current allocation of resources to process
- Safe state is where there is at least one sequence that does not result in deadlock
- Unsafe state is a state that is not safe

31

31

## Safe, Unsafe , Deadlock State

- If a system is in safe state $\Rightarrow$ no deadlocks.
- If a system is in unsafe state $\Rightarrow$ possibility of deadlock.
- Avoidance $\Rightarrow$ ensure that a system will never enter an unsafe state.

32

32

## Determination of a Safe State Initial State

| | R1 | R2 | R3 |
|---|---|---|---|
| P1 | 3 | 2 | 2 |
| P2 | 6 | 1 | 3 |
| P3 | 3 | 1 | 4 |
| P4 | 4 | 2 | 2 |

Claim matrix **C**

| | R1 | R2 | R3 |
|---|---|---|---|
| P1 | 1 | 0 | 0 |
| P2 | 6 | 1 | 2 |
| P3 | 2 | 1 | 1 |
| P4 | 0 | 0 | 2 |

Allocation matrix **A**

| R1 | R2 | R3 |
|---|---|---|
| 9 | 3 | 6 |

Resource vector **R**

| R1 | R2 | R3 |
|---|---|---|
| 0 | 1 | 1 |

Available vector **V**

**(a) Initial state**

33

33

## Determination of a Safe State Initial State

| | R1 | R2 | R3 |
|---|---|---|---|
| P1 | 3 | 2 | 2 |
| P2 | 6 | 1 | 3 |
| P3 | 3 | 1 | 4 |
| P4 | 4 | 2 | 2 |

Claim matrix **C**

| | R1 | R2 | R3 |
|---|---|---|---|
| P1 | 1 | 0 | 0 |
| P2 | 6 | 1 | 2 |
| P3 | 2 | 1 | 1 |
| P4 | 0 | 0 | 2 |

Allocation matrix **A**

| | R1 | R2 | R3 |
|---|---|---|---|
| P1 | 2 | 2 | 2 |
| P2 | 0 | 0 | 1 |
| P3 | 1 | 0 | 3 |
| P4 | 4 | 2 | 0 |

**C – A**

| R1 | R2 | R3 |
|---|---|---|
| 9 | 3 | 6 |

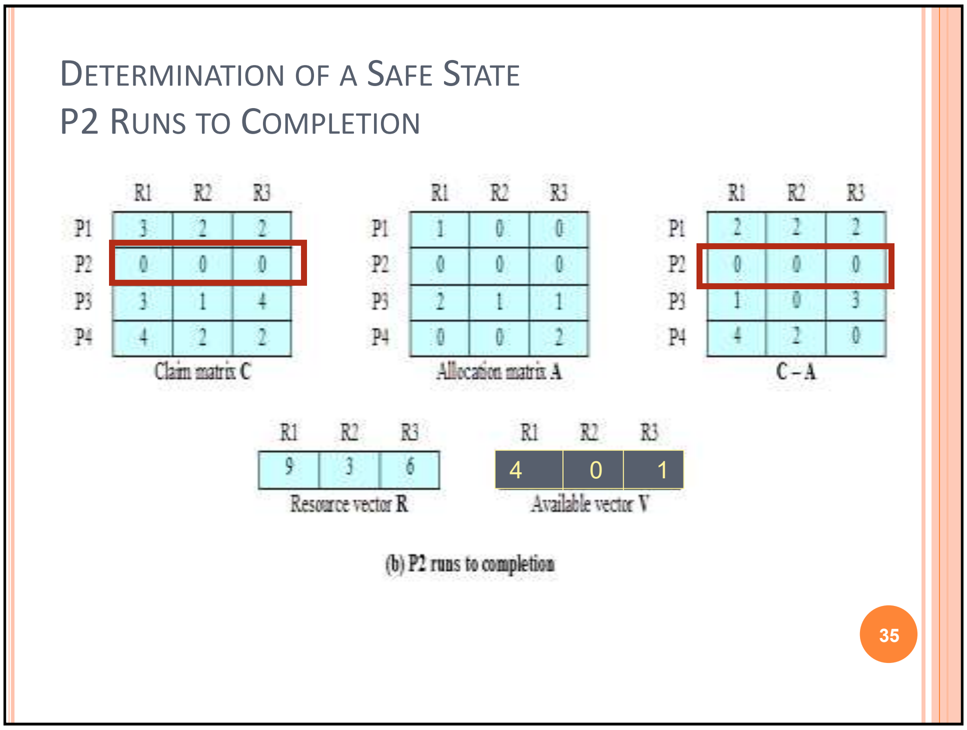
Resource vector **R**

| R1 | R2 | R3 |
|---|---|---|
| 0 | 1 | 0 |

Available vector **V**

**(a) Initial state**

34



## Determination of a Safe State
## P2 Runs to Completion

| | R1 | R2 | R3 |
|---|---|---|---|
| P1 | 3 | 2 | 2 |
| P2 | 0 | 0 | 0 |
| P3 | 3 | 1 | 4 |
| P4 | 4 | 2 | 2 |

Claim matrix C

| | R1 | R2 | R3 |
|---|---|---|---|
| P1 | 1 | 0 | 0 |
| P2 | 0 | 0 | 0 |
| P3 | 2 | 1 | 1 |
| P4 | 0 | 0 | 2 |

Allocation matrix A

| | R1 | R2 | R3 |
|---|---|---|---|
| P1 | 2 | 2 | 2 |
| P2 | 0 | 0 | 0 |
| P3 | 1 | 0 | 3 |
| P4 | 4 | 2 | 0 |

C – A

| R1 | R2 | R3 |
|---|---|---|
| 9 | 3 | 6 |

Resource vector R

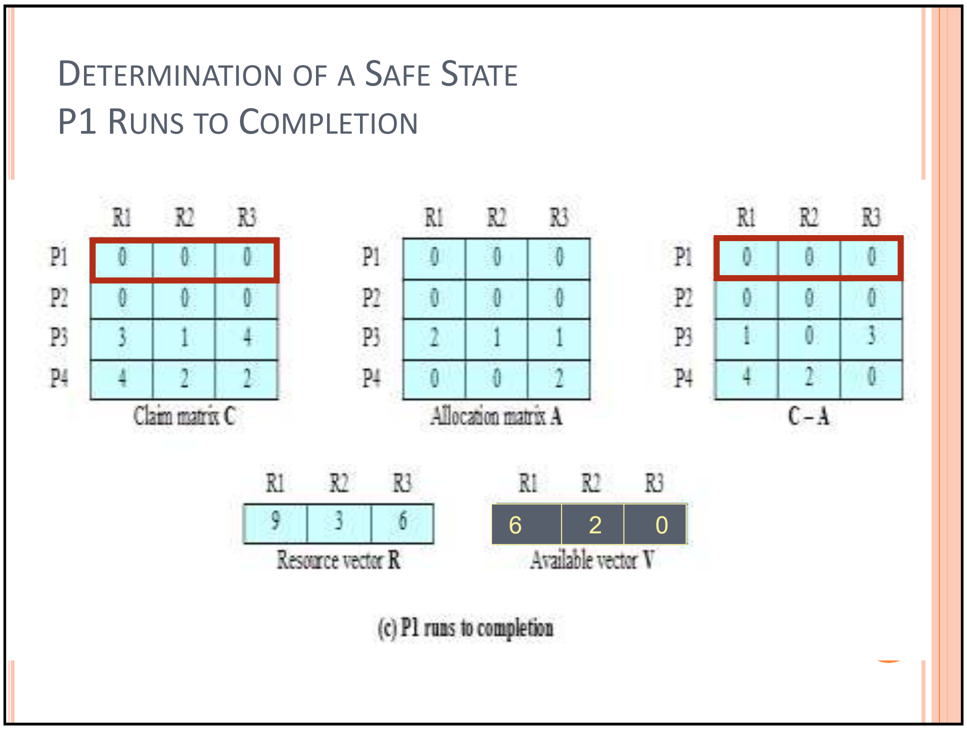
| R1 | R2 | R3 |
|---|---|---|
| 4 | 0 | 1 |

Available vector V

(b) P2 runs to completion

35

35



## Determination of a Safe State
## P1 Runs to Completion

| | R1 | R2 | R3 |
|---|---|---|---|
| P1 | 0 | 0 | 0 |
| P2 | 0 | 0 | 0 |
| P3 | 3 | 1 | 4 |
| P4 | 4 | 2 | 2 |

Claim matrix C

| | R1 | R2 | R3 |
|---|---|---|---|
| P1 | 0 | 0 | 0 |
| P2 | 0 | 0 | 0 |
| P3 | 2 | 1 | 1 |
| P4 | 0 | 0 | 2 |

Allocation matrix A

| | R1 | R2 | R3 |
|---|---|---|---|
| P1 | 0 | 0 | 0 |
| P2 | 0 | 0 | 0 |
| P3 | 1 | 0 | 3 |
| P4 | 4 | 2 | 0 |

C – A

| R1 | R2 | R3 |
|---|---|---|
| 9 | 3 | 6 |

Resource vector R

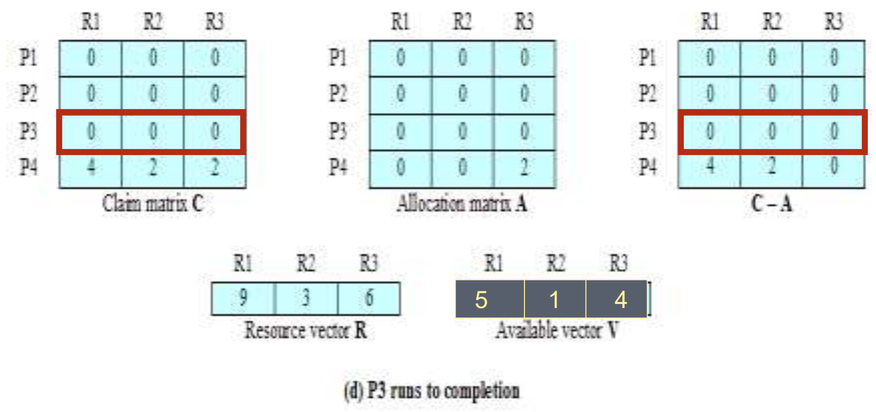
| R1 | R2 | R3 |
|---|---|---|
| 6 | 2 | 0 |

Available vector V

(c) P1 runs to completion

36

## Determination of a Safe State
## P3 Runs to Completion



Claim matrix **C**

| | R1 | R2 | R3 |
|---|---|---|---|
| P1 | 0 | 0 | 0 |
| P2 | 0 | 0 | 0 |
| P3 | 0 | 0 | 0 |
| P4 | 4 | 2 | 2 |

Allocation matrix **A**

| | R1 | R2 | R3 |
|---|---|---|---|
| P1 | 0 | 0 | 0 |
| P2 | 0 | 0 | 0 |
| P3 | 0 | 0 | 0 |
| P4 | 0 | 0 | 2 |

**C – A**

| | R1 | R2 | R3 |
|---|---|---|---|
| P1 | 0 | 0 | 0 |
| P2 | 0 | 0 | 0 |
| P3 | 0 | 0 | 0 |
| P4 | 4 | 2 | 0 |

Resource vector **R**

| R1 | R2 | R3 |
|---|---|---|
| 9 | 3 | 6 |

Available vector **V**

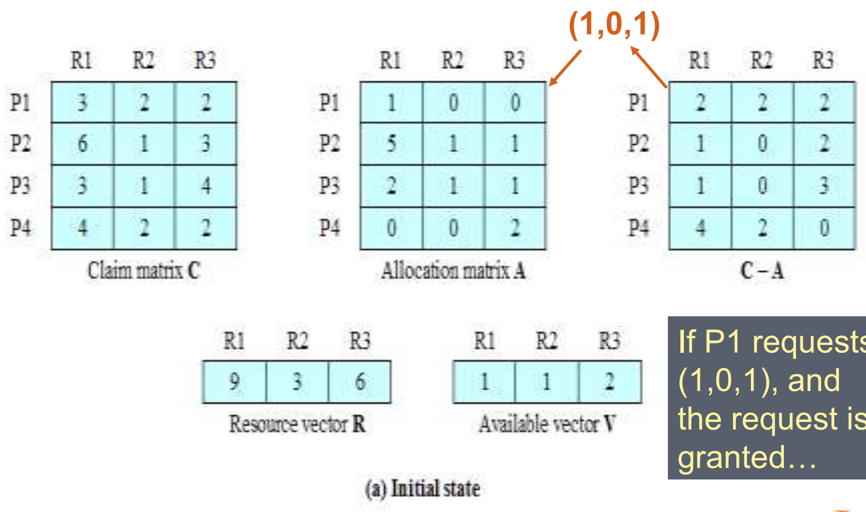
| R1 | R2 | R3 |
|---|---|---|
| 5 | 1 | 4 |

**(d) P3 runs to completion**

37

37

## Determination of an Unsafe State



Claim matrix **C**

| | R1 | R2 | R3 |
|---|---|---|---|
| P1 | 3 | 2 | 2 |
| P2 | 6 | 1 | 3 |
| P3 | 3 | 1 | 4 |
| P4 | 4 | 2 | 2 |

Allocation matrix **A**

| | R1 | R2 | R3 |
|---|---|---|---|
| P1 | 1 | 0 | 0 |
| P2 | 5 | 1 | 1 |
| P3 | 2 | 1 | 1 |
| P4 | 0 | 0 | 2 |

**C – A**

| | R1 | R2 | R3 |
|---|---|---|---|
| P1 | 2 | 2 | 2 |
| P2 | 1 | 0 | 2 |
| P3 | 1 | 0 | 3 |
| P4 | 4 | 2 | 0 |

Resource vector **R**

| R1 | R2 | R3 |
|---|---|---|
| 9 | 3 | 6 |

Available vector **V**

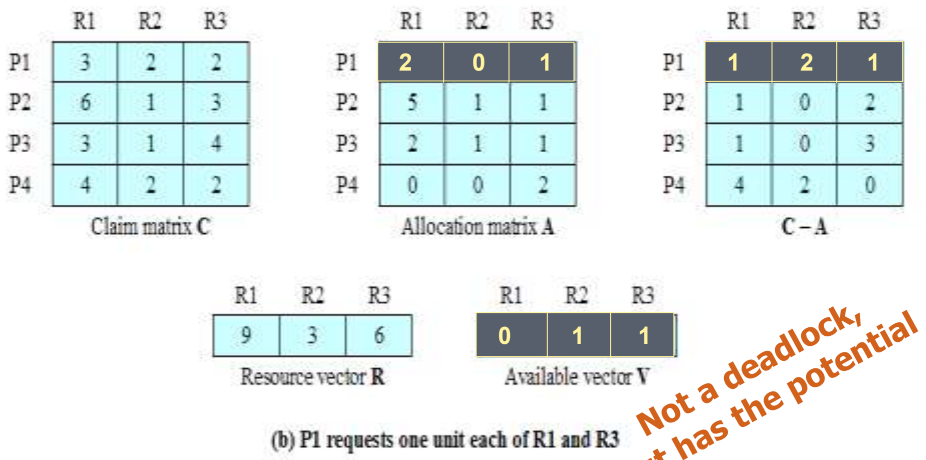
| R1 | R2 | R3 |
|---|---|---|
| 1 | 1 | 2 |

**(a) Initial state**

If P1 requests (1,0,1), and the request is granted…

38

## Determination of an Unsafe State



Claim matrix **C**

| | R1 | R2 | R3 |
|---|---|---|---|
| P1 | 3 | 2 | 2 |
| P2 | 6 | 1 | 3 |
| P3 | 3 | 1 | 4 |
| P4 | 4 | 2 | 2 |

Allocation matrix **A**

| | R1 | R2 | R3 |
|---|---|---|---|
| P1 | 2 | 0 | 1 |
| P2 | 5 | 1 | 1 |
| P3 | 2 | 1 | 1 |
| P4 | 0 | 0 | 2 |

**C** – **A**

| | R1 | R2 | R3 |
|---|---|---|---|
| P1 | 1 | 2 | 1 |
| P2 | 1 | 0 | 2 |
| P3 | 1 | 0 | 3 |
| P4 | 4 | 2 | 0 |

Resource vector **R**

| R1 | R2 | R3 |
|---|---|---|
| 9 | 3 | 6 |

Available vector **V**

| R1 | R2 | R3 |
|---|---|---|
| 0 | 1 | 1 |

**(b) P1 requests one unit each of R1 and R3**

**Not a deadlock, but has the potential**

39

## Deadlock Avoidance

- Maximum resource requirement must be stated in advance
- Processes under consideration must be independent; no synchronization requirements
- There must be a fixed number of resources to allocate
- No process may exit while holding resources

40

## Deadlock Detection

- Deadlock detection do not limit resource access or restrict process action. Resources are allocated whenever demanded.
- Periodically, OS checks for circular wait condition using some sort of algorithm for detection.

41

41

## Deadlock Detection

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

42

42

## Deadlock Detection

- The steps with the assumption that all processes are marked initially indicating not deadlocked:
  1. Mark each process for which all the elements are zeros in the allocation matrix.
  2. Initialize a matrix w equal to the Available matrix.
  3. Recursively, find an index i, such that process i is currently unmarked and the i-th row of Q is less than or equal to W meaning:

     $Q_{ik} \leq W_k \;\; \mathit{for}\ 1 \leq k \leq m$

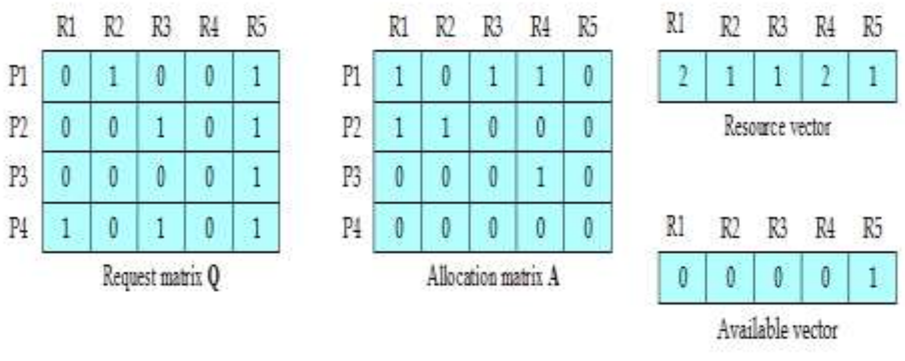
  If true:

     $W_k = W_k + A_{ik} \;\; \mathit{for}\ 1 \leq k \leq m$

  If false:

     Terminate

## Deadlock Detection



Request matrix Q

|  | R1 | R2 | R3 | R4 | R5 |
|---|---|---|---|---|---|
| P1 | 0 | 1 | 0 | 0 | 1 |
| P2 | 0 | 0 | 1 | 0 | 1 |
| P3 | 0 | 0 | 0 | 0 | 1 |
| P4 | 1 | 0 | 1 | 0 | 1 |

Allocation matrix A

|  | R1 | R2 | R3 | R4 | R5 |
|---|---|---|---|---|---|
| P1 | 1 | 0 | 1 | 1 | 0 |
| P2 | 1 | 1 | 0 | 0 | 0 |
| P3 | 0 | 0 | 0 | 1 | 0 |
| P4 | 0 | 0 | 0 | 0 | 0 |

Resource vector

| R1 | R2 | R3 | R4 | R5 |
|---|---|---|---|---|
| 2 | 1 | 1 | 2 | 1 |

Available vector

| R1 | R2 | R3 | R4 | R5 |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 |

**Figure 6.10 Example for Deadlock Detection**

## Deadlock Detection

Let us see an example to demonstrate deadlock detection strategy:

Request Matrix Q

| | R1 | R2 | R3 | R4 | R5 |
|---|---|---|---|---|---|
| P1 | 0 | 1 | 0 | 0 | 1 |
| P2 | 0 | 0 | 1 | 0 | 1 |
| P3 | 0 | 0 | 0 | 0 | 1 |
| P4 | 1 | 0 | 1 | 0 | 1 |

Allocation Matrix

| | R1 | R2 | R3 | R4 | R5 |
|---|---|---|---|---|---|
| P1 | 1 | 0 | 1 | 1 | 0 |
| P2 | 1 | 1 | 0 | 0 | 0 |
| P3 | 0 | 0 | 0 | 1 | 0 |
| P4 | 0 | 0 | 0 | 0 | 0 |

Available Vector

| R1 | R2 | R3 | R4 | R5 |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 |

Resource Vector

| R1 | R2 | R3 | R4 | R5 |
|---|---|---|---|---|
| 2 | 1 | 1 | 2 | 1 |

45

## Deadlock Detection – Step 1

Request Matrix Q

| | R1 | R2 | R3 | R4 | R5 |
|---|---|---|---|---|---|
| P1 | 0 | 1 | 0 | 0 | 1 |
| P2 | 0 | 0 | 1 | 0 | 1 |
| P3 | 0 | 0 | 0 | 0 | 1 |
| ~~P4~~ | ~~1~~ | ~~0~~ | ~~1~~ | ~~0~~ | ~~1~~ |

Allocation Matrix

| | R1 | R2 | R3 | R4 | R5 |
|---|---|---|---|---|---|
| P1 | 1 | 0 | 1 | 1 | 0 |
| P2 | 1 | 1 | 0 | 0 | 0 |
| P3 | 0 | 0 | 0 | 1 | 0 |
| P4 | 0 | 0 | 0 | 0 | 0 |

Mark P4, because no allocated resources

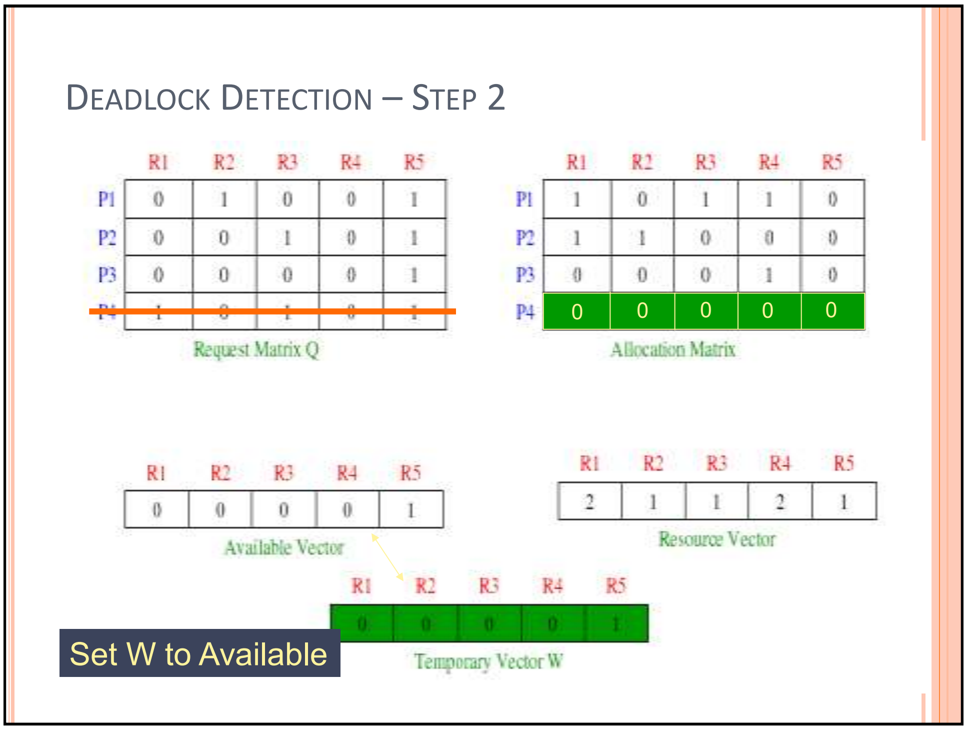
Available Vector

| R1 | R2 | R3 | R4 | R5 |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 |

Resource Vector

| R1 | R2 | R3 | R4 | R5 |
|---|---|---|---|---|
| 2 | 1 | 1 | 2 | 1 |

46



## Deadlock Detection – Step 2

| | R1 | R2 | R3 | R4 | R5 |
|---|---|---|---|---|---|
| P1 | 0 | 1 | 0 | 0 | 1 |
| P2 | 0 | 0 | 1 | 0 | 1 |
| P3 | 0 | 0 | 0 | 0 | 1 |
| ~~P4~~ | ~~1~~ | ~~0~~ | ~~1~~ | ~~0~~ | ~~1~~ |

Request Matrix Q

| | R1 | R2 | R3 | R4 | R5 |
|---|---|---|---|---|---|
| P1 | 1 | 0 | 1 | 1 | 0 |
| P2 | 1 | 1 | 0 | 0 | 0 |
| P3 | 0 | 0 | 0 | 1 | 0 |
| P4 | 0 | 0 | 0 | 0 | 0 |

Allocation Matrix

| R1 | R2 | R3 | R4 | R5 |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 |

Available Vector

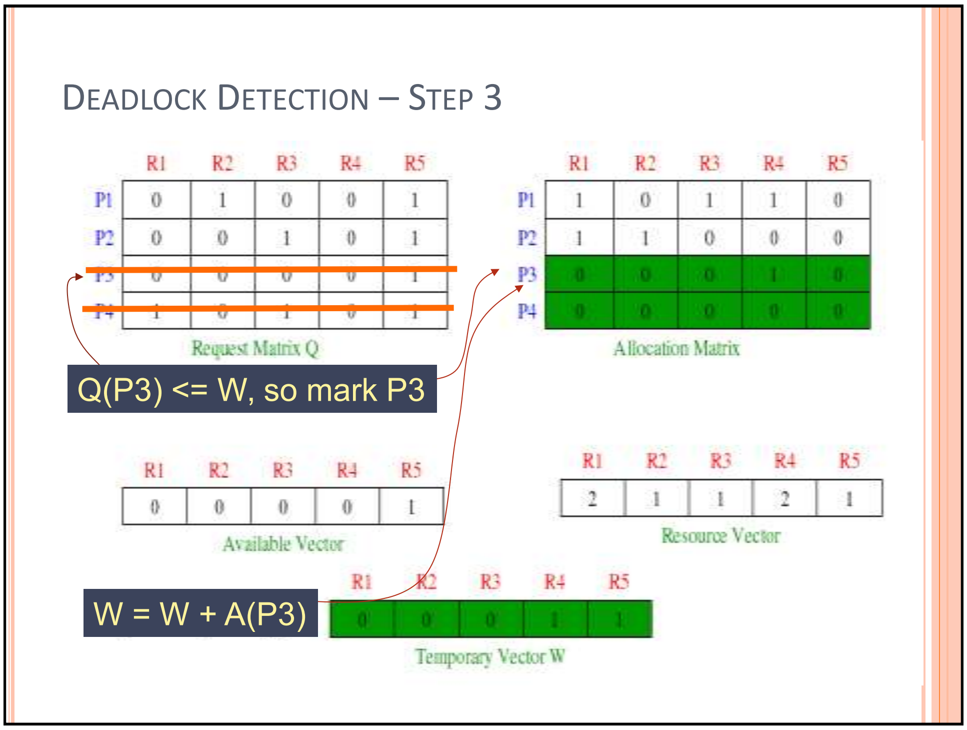
| R1 | R2 | R3 | R4 | R5 |
|---|---|---|---|---|
| 2 | 1 | 1 | 2 | 1 |

Resource Vector

| R1 | R2 | R3 | R4 | R5 |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 |

Temporary Vector W

Set W to Available



## Deadlock Detection – Step 3

| | R1 | R2 | R3 | R4 | R5 |
|---|---|---|---|---|---|
| P1 | 0 | 1 | 0 | 0 | 1 |
| P2 | 0 | 0 | 1 | 0 | 1 |
| ~~P3~~ | ~~0~~ | ~~0~~ | ~~0~~ | ~~0~~ | ~~1~~ |
| ~~P4~~ | ~~1~~ | ~~0~~ | ~~1~~ | ~~0~~ | ~~1~~ |

Request Matrix Q

| | R1 | R2 | R3 | R4 | R5 |
|---|---|---|---|---|---|
| P1 | 1 | 0 | 1 | 1 | 0 |
| P2 | 1 | 1 | 0 | 0 | 0 |
| P3 | 0 | 0 | 0 | 1 | 0 |
| P4 | 0 | 0 | 0 | 0 | 0 |

Allocation Matrix

Q(P3) <= W, so mark P3

| R1 | R2 | R3 | R4 | R5 |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 |

Available Vector

| R1 | R2 | R3 | R4 | R5 |
|---|---|---|---|---|
| 2 | 1 | 1 | 2 | 1 |

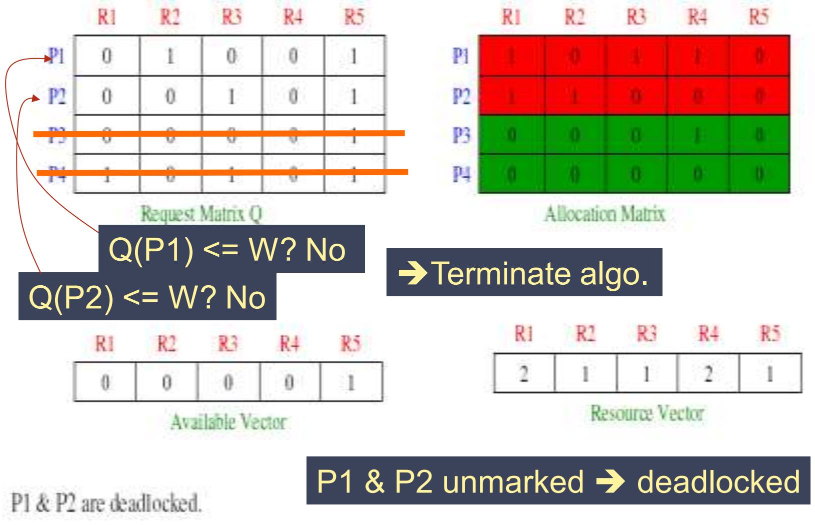
Resource Vector

| R1 | R2 | R3 | R4 | R5 |
|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 1 |

Temporary Vector W

W = W + A(P3)

## Deadlock Detection – Step 4



| | R1 | R2 | R3 | R4 | R5 |
|---|---|---|---|---|---|
| P1 | 0 | 1 | 0 | 0 | 1 |
| P2 | 0 | 0 | 1 | 0 | 1 |
| ~~P3~~ | ~~0~~ | ~~0~~ | ~~0~~ | ~~0~~ | ~~1~~ |
| ~~P4~~ | ~~1~~ | ~~0~~ | ~~1~~ | ~~0~~ | ~~1~~ |

Request Matrix Q

| | R1 | R2 | R3 | R4 | R5 |
|---|---|---|---|---|---|
| P1 | 1 | 0 | 1 | 1 | 0 |
| P2 | 1 | 1 | 0 | 0 | 0 |
| P3 | 0 | 0 | 0 | 1 | 0 |
| P4 | 0 | 0 | 0 | 0 | 0 |

Allocation Matrix

Q(P1) <= W? No

Q(P2) <= W? No

➔Terminate algo.

| R1 | R2 | R3 | R4 | R5 |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 |

Available Vector

| R1 | R2 | R3 | R4 | R5 |
|---|---|---|---|---|
| 2 | 1 | 1 | 2 | 1 |

Resource Vector

P1 & P2 are deadlocked.

P1 & P2 unmarked ➔ deadlocked

49

## Example of Detection Algorithm

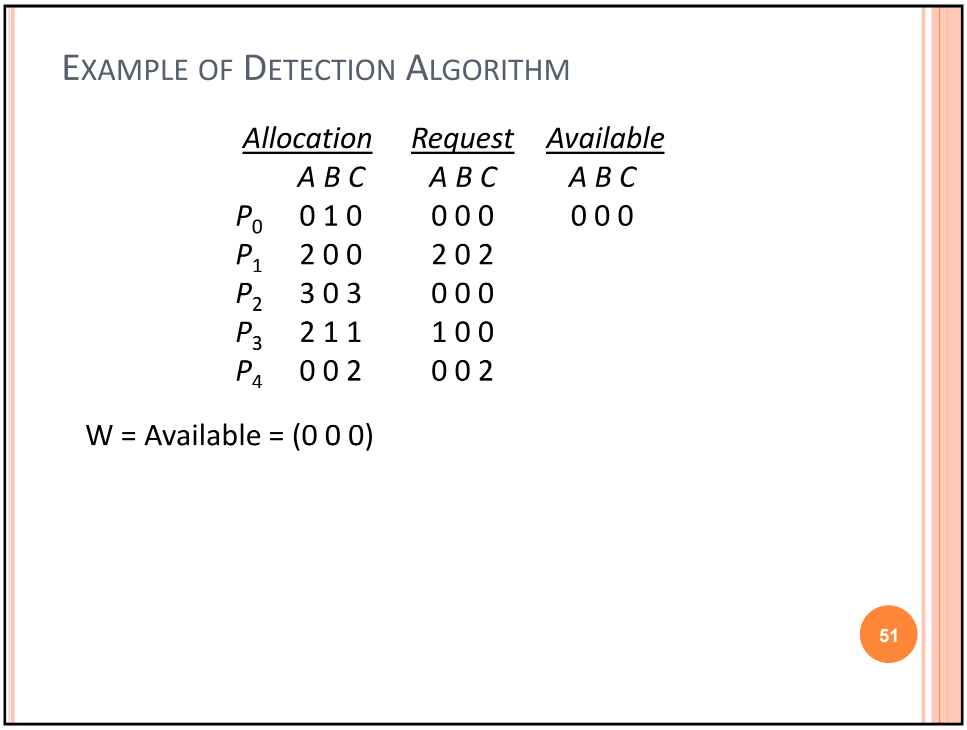
- Five processes $P_0$ through $P_4$; three resource types A (7 instances), *B* (2 instances), and *C* (6 instances).
- Snapshot at time $T_0$:

| | *Allocation* | *Request* | *Available* |
|---|---|---|---|
| | *A B C* | *A B C* | *A B C* |
| $P_0$ | 0 1 0 | 0 0 0 | 0 0 0 |
| $P_1$ | 2 0 0 | 2 0 2 | |
| $P_2$ | 3 0 3 | 0 0 0 | |
| $P_3$ | 2 1 1 | 1 0 0 | |
| $P_4$ | 0 0 2 | 0 0 2 | |

- Sequence <$P_0$, $P_2$, $P_3$, $P_1$, $P_4$> will result in *Finish*[*i*] = true for all *i*.

50

50

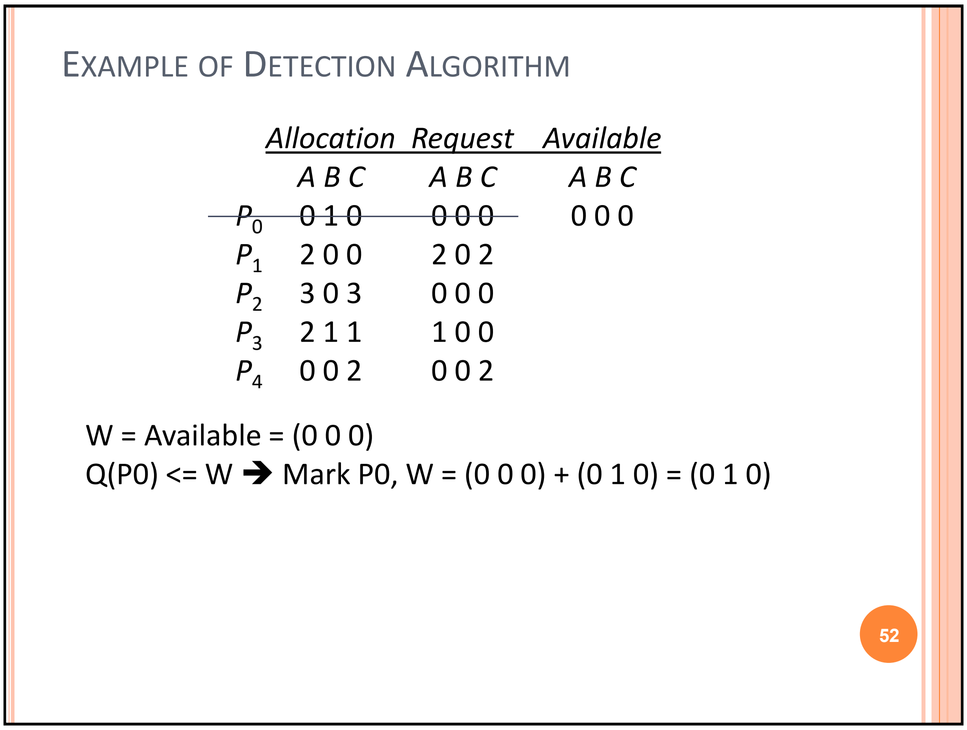


## Example of Detection Algorithm

| | *Allocation* | *Request* | *Available* |
|---|---|---|---|
| | *A B C* | *A B C* | *A B C* |
| $P_0$ | 0 1 0 | 0 0 0 | 0 0 0 |
| $P_1$ | 2 0 0 | 2 0 2 | |
| $P_2$ | 3 0 3 | 0 0 0 | |
| $P_3$ | 2 1 1 | 1 0 0 | |
| $P_4$ | 0 0 2 | 0 0 2 | |

W = Available = (0 0 0)

51



## Example of Detection Algorithm

| | *Allocation* | *Request* | *Available* |
|---|---|---|---|
| | *A B C* | *A B C* | *A B C* |
| ~~$P_0$~~ | ~~0 1 0~~ | ~~0 0 0~~ | 0 0 0 |
| $P_1$ | 2 0 0 | 2 0 2 | |
| $P_2$ | 3 0 3 | 0 0 0 | |
| $P_3$ | 2 1 1 | 1 0 0 | |
| $P_4$ | 0 0 2 | 0 0 2 | |

W = Available = (0 0 0)
Q(P0) <= W ➔ Mark P0, W = (0 0 0) + (0 1 0) = (0 1 0)

52

## Example of Detection Algorithm

| | *Allocation* | *Request* | *Available* |
|---|---|---|---|
| | *A B C* | *A B C* | *A B C* |
| ~~$P_0$~~ | ~~0 1 0~~ | ~~0 0 0~~ | 0 0 0 |
| $P_1$ | 2 0 0 | 2 0 2 | |
| ~~$P_2$~~ | ~~3 0 3~~ | ~~0 0 0~~ | |
| $P_3$ | 2 1 1 | 1 0 0 | |
| $P_4$ | 0 0 2 | 0 0 2 | |

W = Available = (0 0 0)
Q(P0) <= W ➔ Mark P0, W = (0 0 0) + (0 1 0) = (0 1 0)
Q(P2) <= W ➔ Mark P2, W = (0 1 0) + (3 0 3) = (3 1 3)

53

53

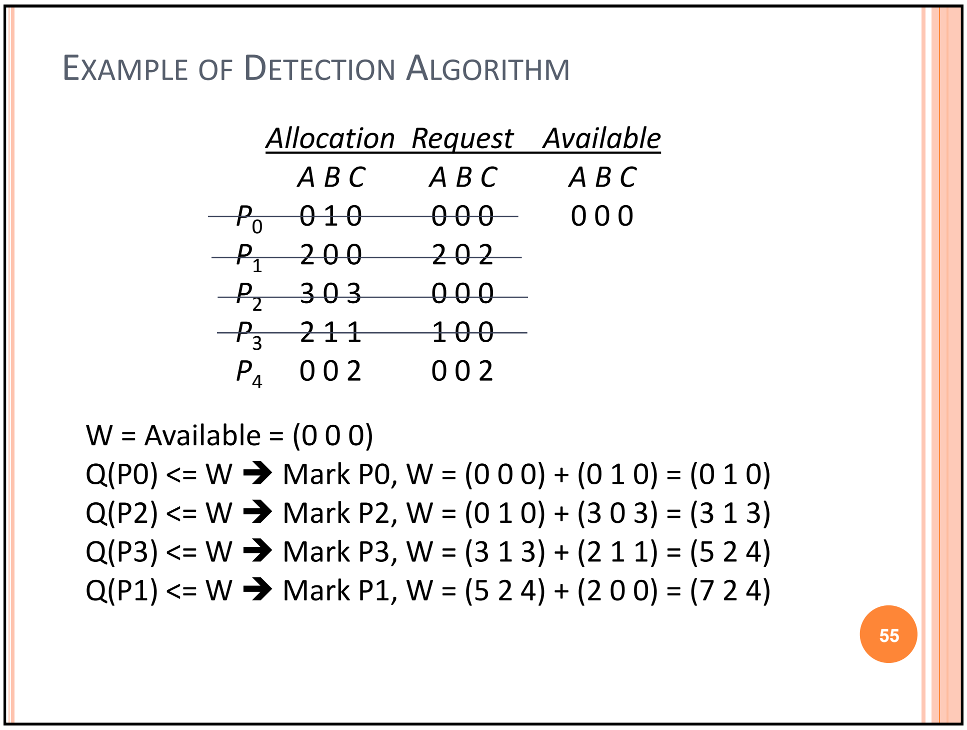
## Example of Detection Algorithm

| | *Allocation* | *Request* | *Available* |
|---|---|---|---|
| | *A B C* | *A B C* | *A B C* |
| ~~$P_0$~~ | ~~0 1 0~~ | ~~0 0 0~~ | 0 0 0 |
| $P_1$ | 2 0 0 | 2 0 2 | |
| ~~$P_2$~~ | ~~3 0 3~~ | ~~0 0 0~~ | |
| ~~$P_3$~~ | ~~2 1 1~~ | ~~1 0 0~~ | |
| $P_4$ | 0 0 2 | 0 0 2 | |

W = Available = (0 0 0)
Q(P0) <= W ➔ Mark P0, W = (0 0 0) + (0 1 0) = (0 1 0)
Q(P2) <= W ➔ Mark P2, W = (0 1 0) + (3 0 3) = (3 1 3)
Q(P3) <= W ➔ Mark P3, W = (3 1 3) + (2 1 1) = (5 2 4)

54

54



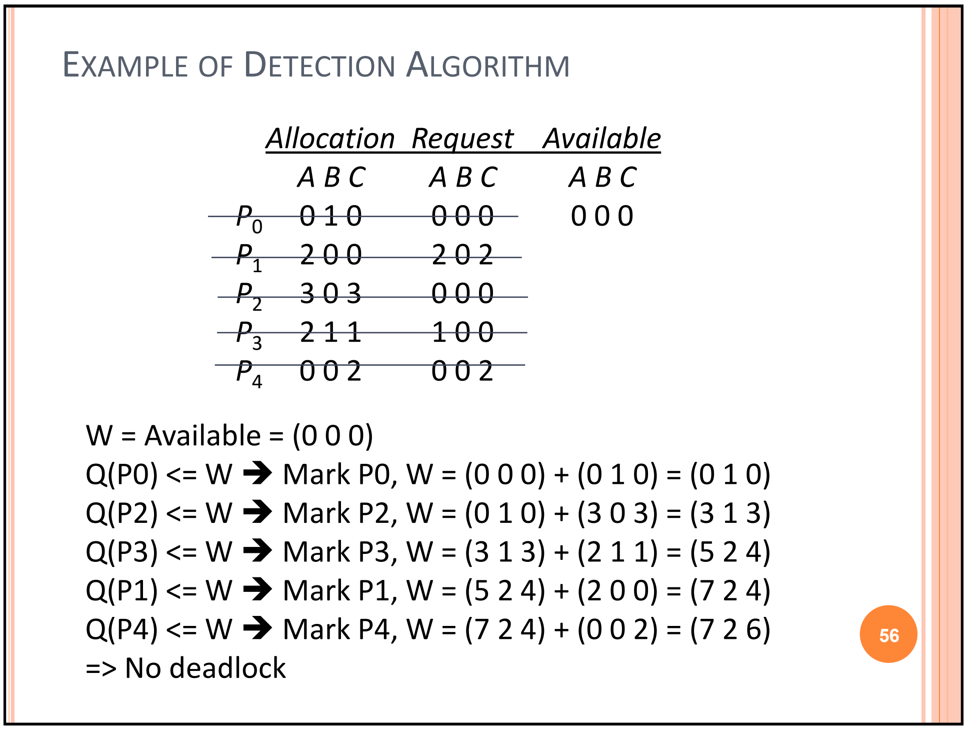
## EXAMPLE OF DETECTION ALGORITHM

| | *Allocation* | *Request* | *Available* |
|---|---|---|---|
| | *A B C* | *A B C* | *A B C* |
| ~~$P_0$~~ | ~~0 1 0~~ | ~~0 0 0~~ | 0 0 0 |
| ~~$P_1$~~ | ~~2 0 0~~ | ~~2 0 2~~ | |
| ~~$P_2$~~ | ~~3 0 3~~ | ~~0 0 0~~ | |
| ~~$P_3$~~ | ~~2 1 1~~ | ~~1 0 0~~ | |
| $P_4$ | 0 0 2 | 0 0 2 | |

W = Available = (0 0 0)
Q(P0) <= W ➔ Mark P0, W = (0 0 0) + (0 1 0) = (0 1 0)
Q(P2) <= W ➔ Mark P2, W = (0 1 0) + (3 0 3) = (3 1 3)
Q(P3) <= W ➔ Mark P3, W = (3 1 3) + (2 1 1) = (5 2 4)
Q(P1) <= W ➔ Mark P1, W = (5 2 4) + (2 0 0) = (7 2 4)

55

55



## EXAMPLE OF DETECTION ALGORITHM

| | *Allocation* | *Request* | *Available* |
|---|---|---|---|
| | *A B C* | *A B C* | *A B C* |
| ~~$P_0$~~ | ~~0 1 0~~ | ~~0 0 0~~ | 0 0 0 |
| ~~$P_1$~~ | ~~2 0 0~~ | ~~2 0 2~~ | |
| ~~$P_2$~~ | ~~3 0 3~~ | ~~0 0 0~~ | |
| ~~$P_3$~~ | ~~2 1 1~~ | ~~1 0 0~~ | |
| ~~$P_4$~~ | ~~0 0 2~~ | ~~0 0 2~~ | |

W = Available = (0 0 0)
Q(P0) <= W ➔ Mark P0, W = (0 0 0) + (0 1 0) = (0 1 0)
Q(P2) <= W ➔ Mark P2, W = (0 1 0) + (3 0 3) = (3 1 3)
Q(P3) <= W ➔ Mark P3, W = (3 1 3) + (2 1 1) = (5 2 4)
Q(P1) <= W ➔ Mark P1, W = (5 2 4) + (2 0 0) = (7 2 4)
Q(P4) <= W ➔ Mark P4, W = (7 2 4) + (0 0 2) = (7 2 6)
=> No deadlock

56

56

## Example (Cont.)

- $P_2$ requests an additional instance of type $C$.

| | *Request* |
|---|---|
| | *A B C* |
| $P_0$ | 0 0 0 |
| $P_1$ | 2 0 1 |
| $P_2$ | 0 0 $\bigcirc$**1** |
| $P_3$ | 1 0 0 |
| $P_4$ | 0 0 2 |

- State of system?
  - Can reclaim resources held by process $P_0$, but insufficient resources to fulfill other processes; requests.
  - Deadlock exists, consisting of processes $P_1$, $P_2$, $P_3$, and $P_4$.

57

57

## Example of Detection Algorithm

| | *Allocation* | *Request* | *Available* |
|---|---|---|---|
| | *A B C* | *A B C* | *A B C* |
| ~~$P_0$~~ | ~~0 1 0~~ | ~~0 0 0~~ | 0 0 0 |
| $P_1$ | 2 0 0 | 2 0 2 | |
| $P_2$ | 3 0 3 | 0 0 1 | |
| $P_3$ | 2 1 1 | 1 0 0 | |
| $P_4$ | 0 0 2 | 0 0 2 | |

W = Available = (0 0 0)
Mark P0, W = (0 0 0) + (0 1 0) = (0 1 0)
⇒ Not enough to satisfy any other request
⇒ Deadlock
⇒ $P_1$, $P_2$, $P_3$, $P_4$ are deadlocked

58

58

## Deadlock Recovery

Strategies once deadlock is detected:

1. Abort all deadlocked processes
2. Back up each deadlocked process to some previously defined checkpoint, and restart all process
   - Original deadlock may occur
3. Successively abort deadlocked processes until deadlock no longer exists
4. Successively preempt resources until deadlock no longer exists

59

59

## Selection Criteria Deadlocked Processes

- Least amount of processor time consumed so far
- Least number of lines of output produced so far
- Most estimated time remaining
- Least total resources allocated so far
- Lowest priority

60

60

## Deadlock Handling Approaches

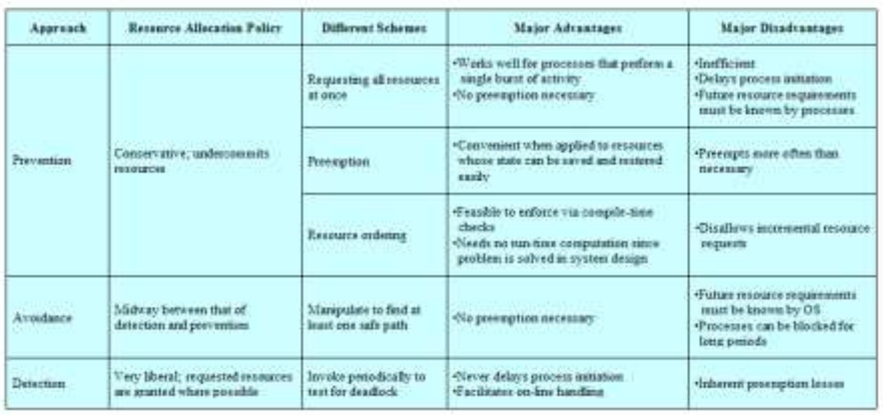
- Prevention
  - do not allow deadlock at all (by eliminating any one of the conditions leading to deadlock)
- Avoidance
  - deadlock may occur, but avoid it (using deadlock avoidance algo). Need to know future resource demands.
- Detection
  - allow deadlock to occur, then detect it, and perform deadlock recovery.

61

61

## Strengths and Weaknesses of the Strategies

**Table 6.1 Summary of Deadlock Detection, Prevention, and Avoidance Approaches for Operating Systems [ISLO80]**



| Approach | Resource Allocation Policy | Different Schemes | Major Advantages | Major Disadvantages |
|---|---|---|---|---|
| Prevention | Conservative; undercommits resources | Requesting all resources at once | •Works well for processes that perform a single burst of activity<br>•No preemption necessary | •Inefficient<br>•Delays process initiation<br>•Future resource requirements must be known by processes |
| | | Preemption | •Convenient when applied to resources whose state can be saved and restored easily | •Preempts more often than necessary |
| | | Resource ordering | •Feasible to enforce via compile-time checks<br>•Needs no run-time computation since problem is solved in system design | •Disallows incremental resource requests |
| Avoidance | Midway between that of detection and prevention | Manipulate to find at least one safe path | •No preemption necessary | •Future resource requirements must be known by OS<br>•Processes can be blocked for long periods |
| Detection | Very liberal; requested resources are granted where possible | Invoke periodically to test for deadlock | •Never delays process initiation<br>•Facilitates on-line handling | •Inherent preemption losses |

62

62

## Dining Philosophers Problem

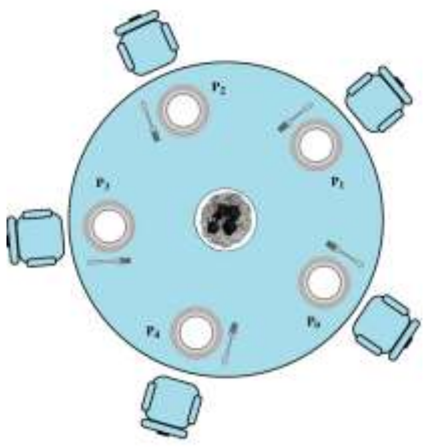


Figure 6.11 Dining Arrangement for Philosophers

63

63

## Dining Philosophers Problem

- Important in mutual exclusion strategy to avoid deadlock and starvation and in coordination of shared resources.
- This problem is standard test for evaluating approaches to synchronization:
  - First attempt could be that each philosopher picks up the fork on his left and then picks the right fork, after eating he puts both forks on the table. Imagine if all the philosophers gets hungry at the same time it, they will sit and each one of them will pick the fork on the left and hence all of them will starve as no second fork.
  - So we can spend more money and bring another set of five forks (expensive solution).
  - Why not provide training so they should be able to eat with just one fork.
  - Another solution could be to have a guard posted at the door who only allows maximum of 4 philosophers at one time, which at least ensure that one of the philosopher will be able to eat. Which ensures deadlock and starvation solution.

64

64