# Deadlock

Deadlock Avoidance

#### Deadlock

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

# Approaches to deadlock handling

- Ignore Deadlock (Ostrich approach)
  - If infrequent enough and result is not serious
- Deadlock Prevention
  - Prevent one of the necessary/sufficient conditions
- Deadlock Avoidance
  - Allow the 3 necessary conditions
  - Dynamically make choices to avoid deadlock
    - decide based on knowledge of future requests
    - i.e., find a <u>safe path</u>

# Approaches to deadlock handling

- Deadlock Detection
  - Periodically run algorithm to detect circular waiting
  - After detecting deadlock,
    - run a <u>recovery algorithm</u> to remove deadlock

### Deadlock avoidance

- Require system has *a priori* information
- Each process declares the maximum number of resources of each type that it may need
- The deadlock-avoidance algorithm *dynamically* examines the resource-allocation *state* to ensure that there can never be a circular-wait condition
- Resource-allocation state is defined by the number of available and allocated resources, and the maximum demands/claims of the processes

# Two deadlock avoidance strategies

- Do not start a process if its demands will result in deadlock
  - Process initiation denial
- Do not grant an *incremental* resource request if this allocation could result in deadlock
  - Resource allocation denial

### Process initiation denial

- Conditions that must hold
  - All resources are either available or allocated
  - No process can claim more than the total amount of resources
  - No process can claim or hold more resources than it originally claimed
- Start a process only if *all the needs* of the current processes and the new process can be met
- *Pessimistic approach* as the assumption is that all processes will require their maximum claims at the same time

### Resource allocation denial

- Banker's Algorithm
  - Multiple instances of each resource type
  - Each process must claim its maximum resource usage *a priori* :
    - can not exceed the total number of resources in the system
  - 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

### System state

- Safe State:
  - there is at least one resource request sequence in which all processes can run to completion
- Unsafe State:
  - There is only a potential for deadlock
- Always ensure the system is in a safe state
  - *Request*: Update system state as if it is granted
  - If state is safe, grant the request; else block the process until it is safe to grant request
- When a process gets all of its resources, it must return them in finite time

### Determination of an unsafe state

	R1	R2	R3
P1	3	2	2
P2	6	1	3
P3	3	1	4
P4	4	2	2
Claim matrix C			

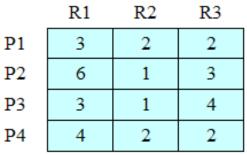
	R1	R2	R3
P1	1	0	0
P2	5	1	1
P3	2	1	1
P4	0	0	2
Allocation matrix A			

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

R1	R2	R3
9	3	6
Resource vector R		

(a) Initial state

# Determination of an unsafe state (2)



Claim matrix C

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

Allocation matrix A

	R1	R2	R3
P1	1	2	1
P2	1	0	2
<b>P</b> 3	1	0	3
P4	4	2	0
		~ .	

 $\mathbf{C} - \mathbf{A}$ 

R1	R2	R3
9	3	6

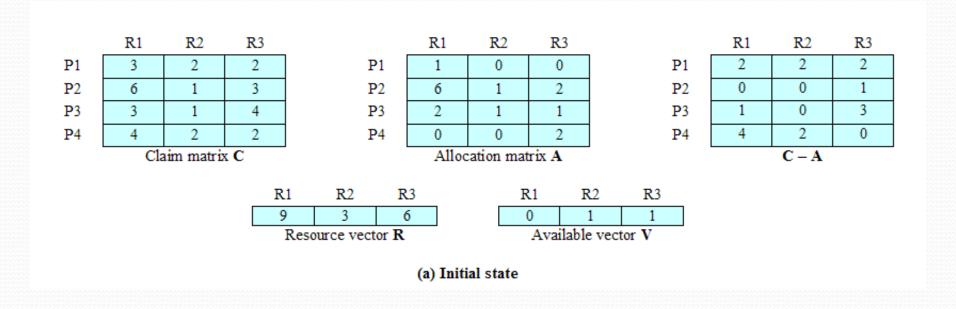
Resource vector R

R1	R2	R3
0	1	1

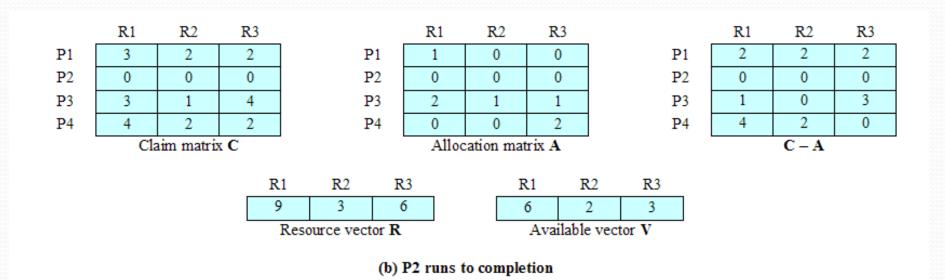
Available vector V

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

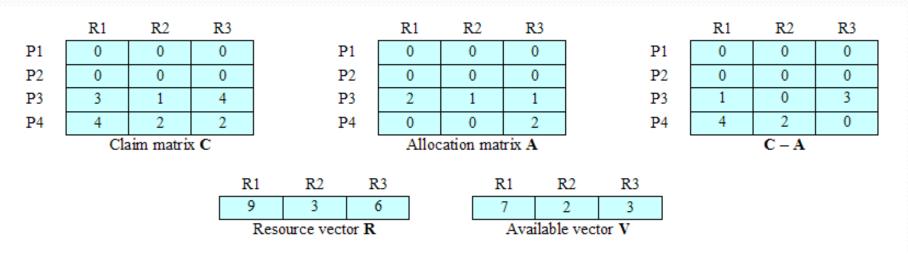
### Determination of a safe state



# Determination of a safe state (2)



# Determination of a safe state (3)



(c) P1 runs to completion

# Determination of a safe state (4)

