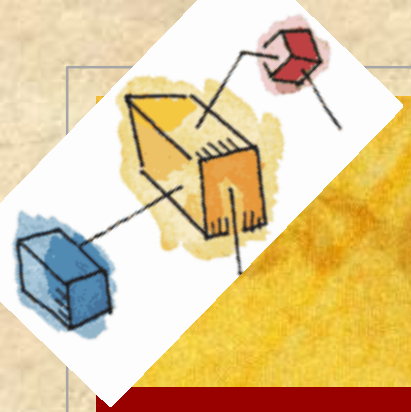
IT 540 Operating Systems ECE519 Advanced Operating Systems

Prof. Dr. Hasan Hüseyin BALIK

(6th Week)

(Advanced) Operating Systems

6. Concurrency: Deadlock and Starvation

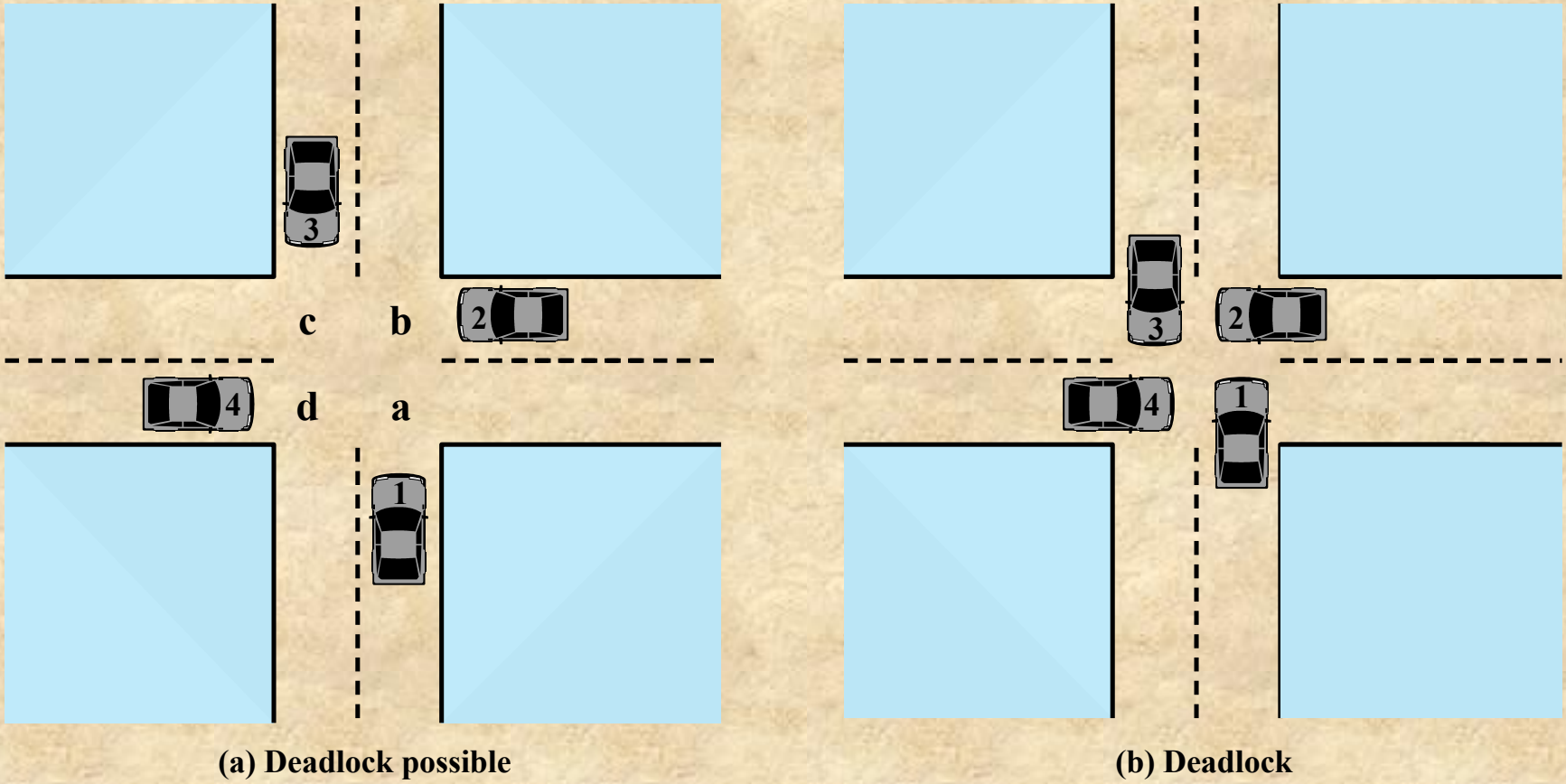


6. Outline

- Principles of Deadlock
- Deadlock Prevention
- Deadlock Avoidance
- Deadlock Detection
- An Integrated Deadlock Strategy

Deadlock

- The permanent blocking of a set of processes that either compete for system resources or communicate with each other
- A set of processes is deadlocked when each process in the set is blocked awaiting an event that can only be triggered by another blocked process in the set
- Permanent
- No efficient solution



(a) Deadlock possible

(b) Deadlock

Figure 6.1 Illustration of Deadlock

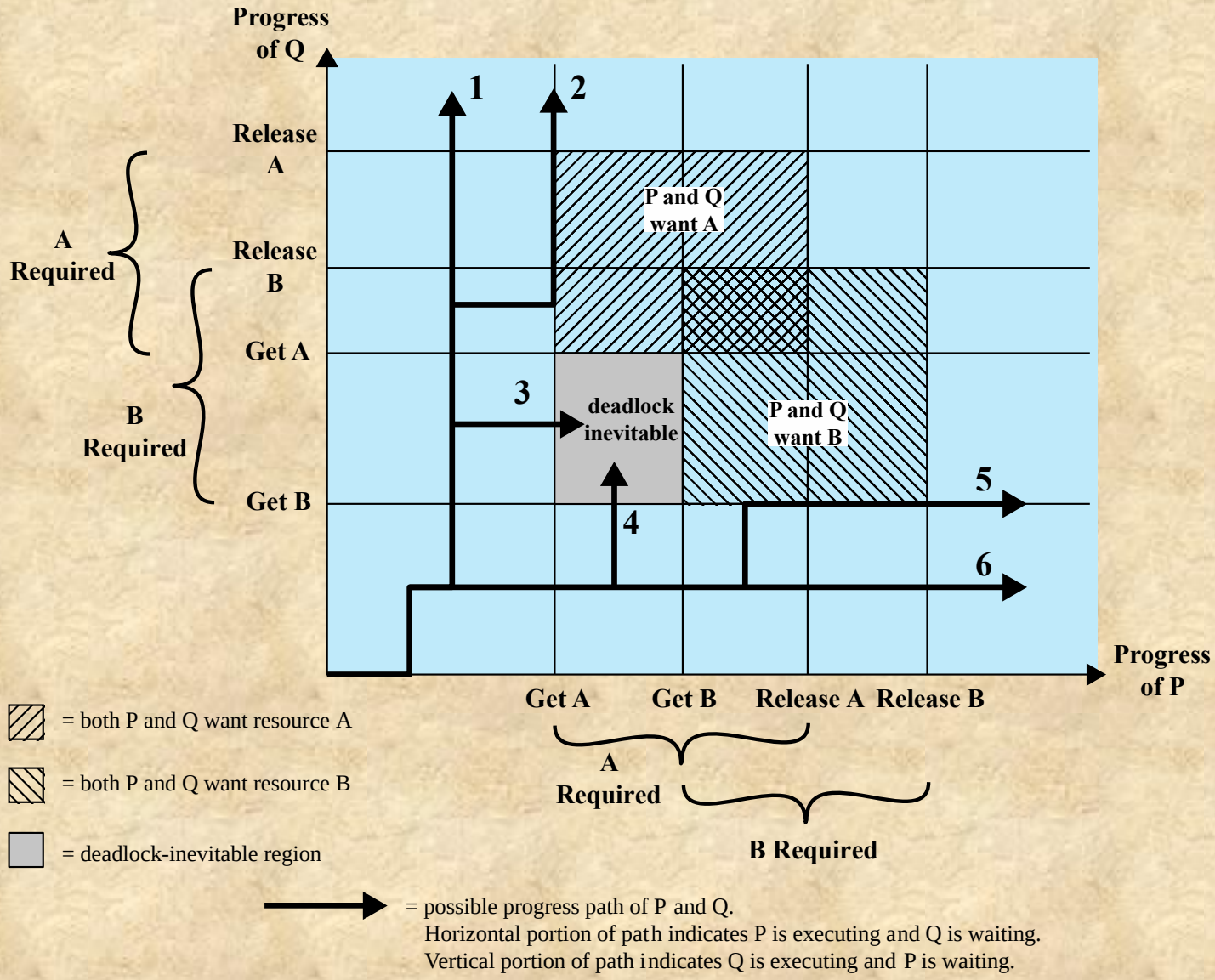


Figure 6.2 Example of Deadlock

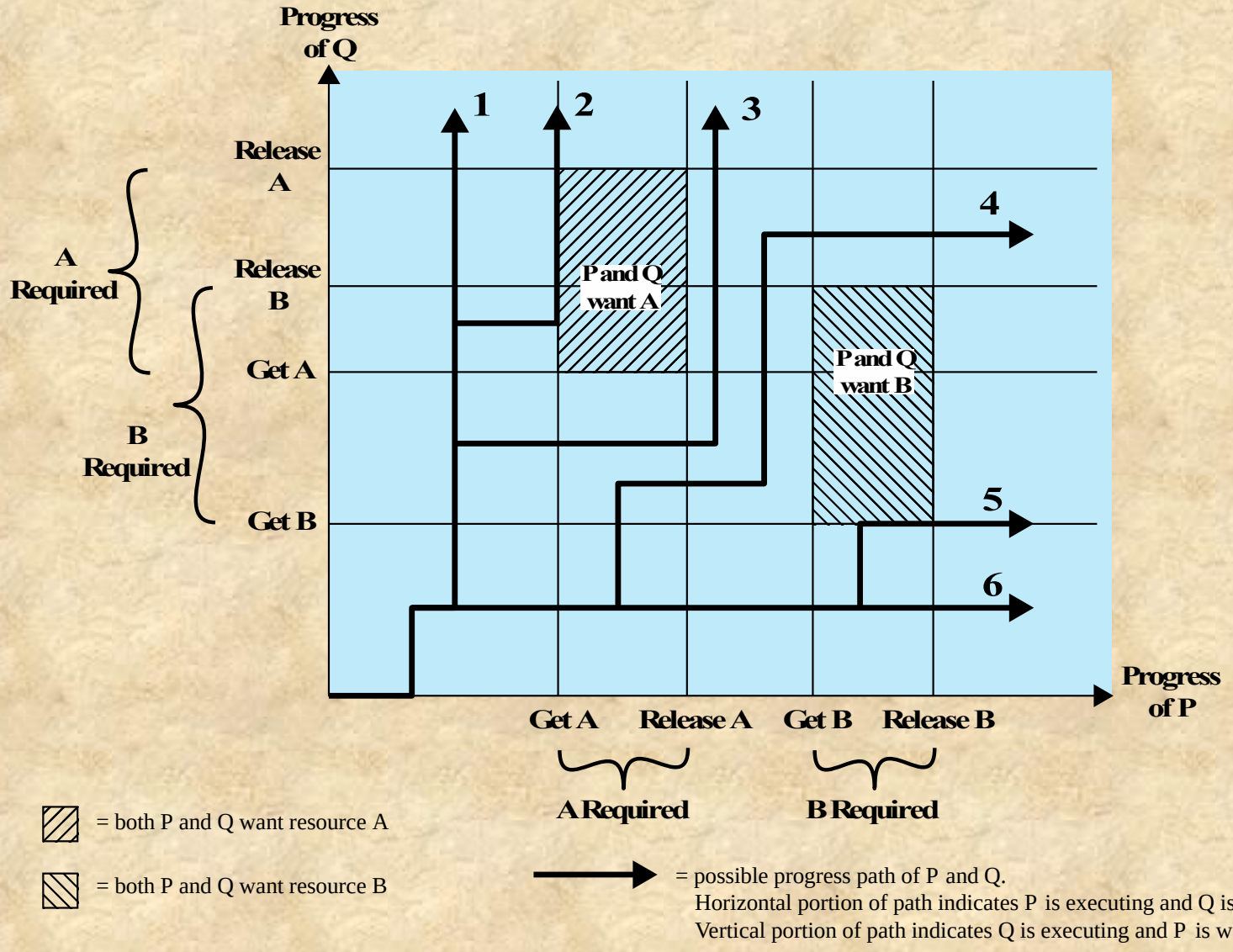


Figure 6.3 Example of No Deadlock

Resource Categories

Reusable

- can be safely used by only one process at a time and is not depleted by that use
  - processors, I/O channels, main and secondary memory, devices, and data structures such as files, databases, and semaphores

Consumable

- one that can be created (produced) and destroyed (consumed)
  - interrupts, signals, messages, and information
  - in I/O buffers

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)

Example of Two Processes Competing for Reusable Resources

Deadlock occurs if the multiprogramming system interleaves the execution of the two processes as follows: p 0 p 1 q 0 q 1 p 2 q 2

Example 2: Memory Request

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

P1

. . .

Request 80 Kbytes;

. . .

Request 60 Kbytes;

P2

. . .

Request 70 Kbytes;

. . .

Request 80 Kbytes;

- Deadlock occurs if both processes progress to their second request

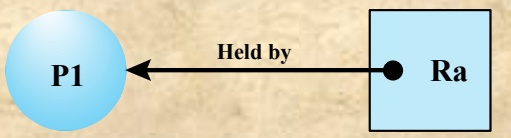
Consumable Resources Deadlock

- Consider a pair of processes, in which each process attempts to receive a message from the other process and then send a message to the other process:

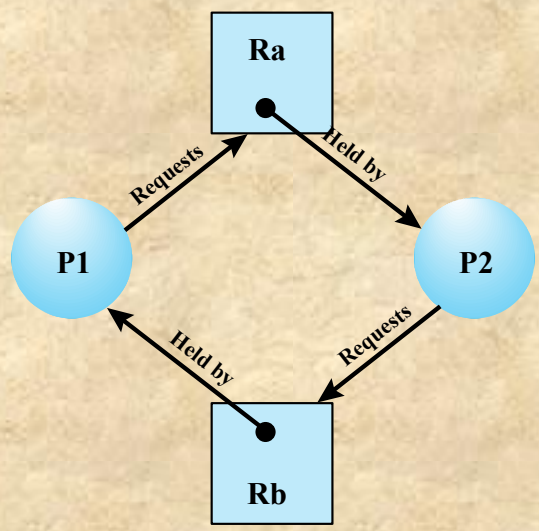
P1	P2
...	...
Receive (P2);	Receive (P1);
...	...
Send (P2, M1);	Send (P1, M2);

- Deadlock occurs if the Receive is blocking

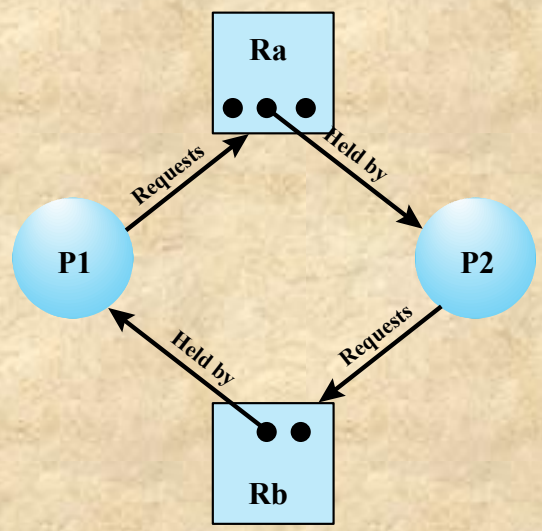
(a) Resouce is requested



(b) Resource is held



(c) Circular wait



(d) No deadlock

Figure 6.5 Examples of Resource Allocation Graphs

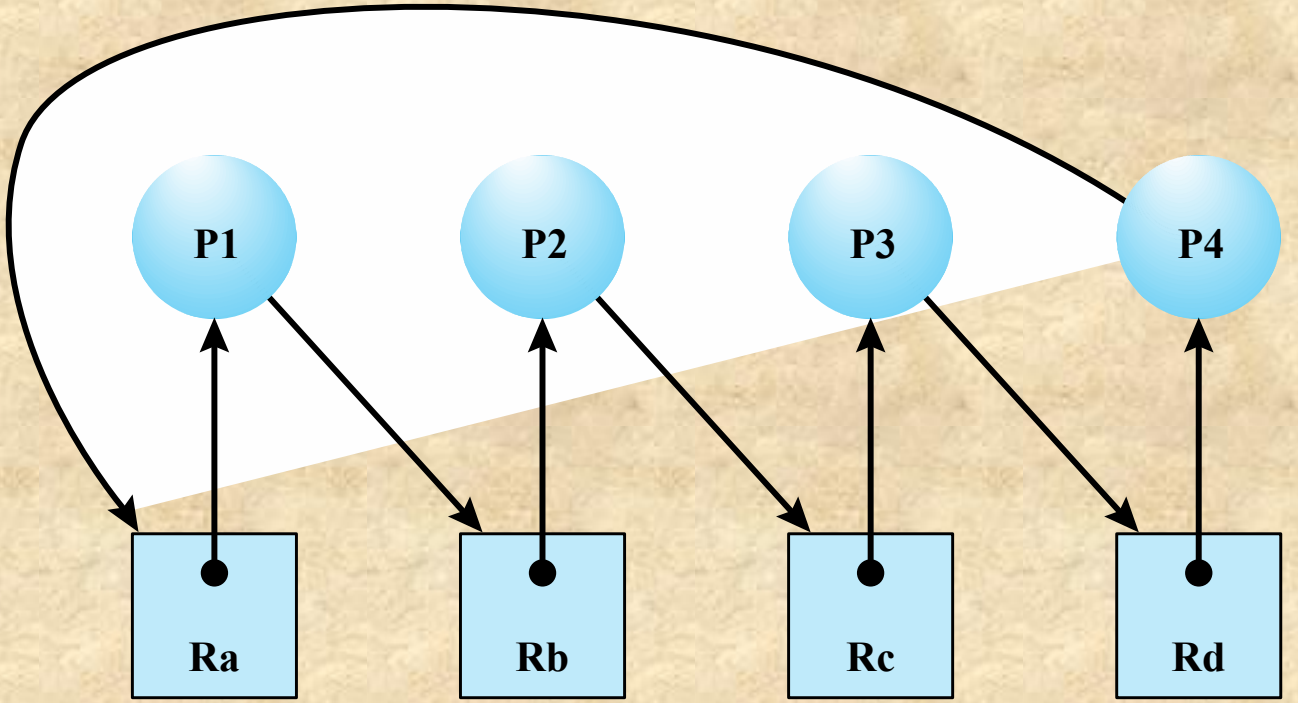


Figure 6.6 Resource Allocation Graph for Figure 6.1b

Conditions for Deadlock

Mutual Exclusion	Hold-and-Wait	No Pre-emption	Circular Wait
• only one process may use a resource at a time • No process may access a resource unit that has been allocated to another process.	• a process may hold allocated resources while awaiting assignment of others	• no resource can be forcibly removed from a process holding it	• a closed chain of processes exists, such that each process holds at least one resource needed by the next process in the chain

Dealing with Deadlock

- Three general approaches exist for dealing with deadlock:

Prevent Deadlock

- adopt a policy that eliminates one of the conditions

Avoid Deadlock

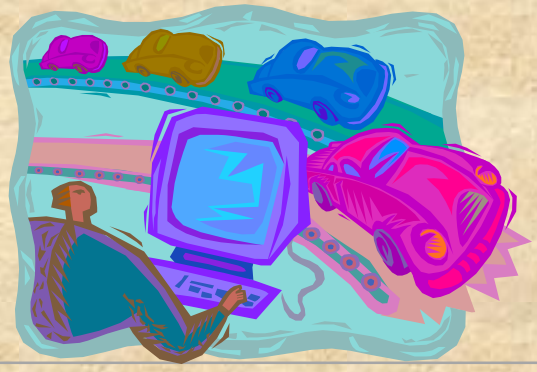
- make the appropriate dynamic choices based on the current state of resource allocation

Detect Deadlock

- attempt to detect the presence of deadlock and take action to recover

Deadlock Prevention Strategy

- Design a system in such a way that the possibility of deadlock is excluded
- Two main methods:
  - Indirect
    - prevent the occurrence of one of the three necessary conditions
  - Direct
    - prevent the occurrence of a circular wait



Deadlock Condition Prevention

Mutual Exclusion

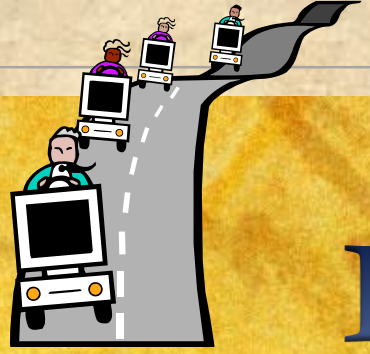
if access to a resource requires mutual exclusion then it must be supported by the OS

Hold and Wait

require that a process request all of its required resources at one time and blocking the process until all requests can be granted simultaneously

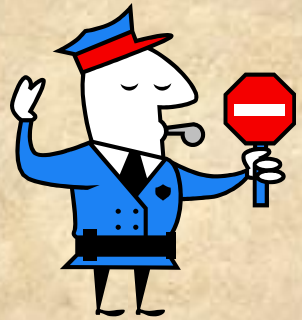
Deadlock Condition Prevention

- No Preemption
  - if a process holding certain resources is denied a further request, that process must release its original resources and request them again
  - OS may preempt the second process and require it to release its resources
- Circular Wait
  - define a linear ordering of resource types



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 requests

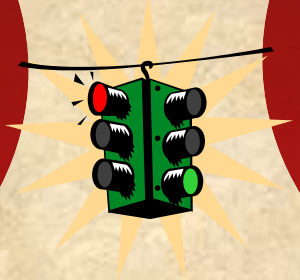


Two Approaches to Deadlock Avoidance

Deadlock Avoidance

Resource Allocation Denial

- do not grant an incremental resource request to a process if this allocation might lead to deadlock

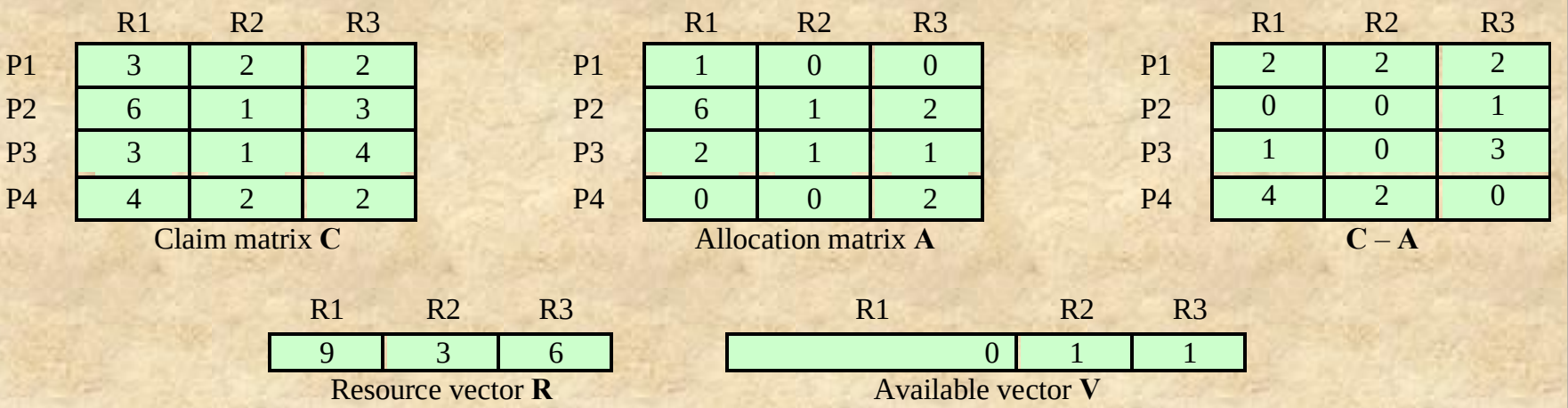


Process Initiation Denial

- do not start a process if its demands might lead to deadlock

Resource Allocation Denial

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



(a) Initial state

Figure 6.7 Determination of a Safe State

	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
6	2	3

Available vector **V**

(b) P2 runs to completion

Figure 6.7 Determination of a Safe State

	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

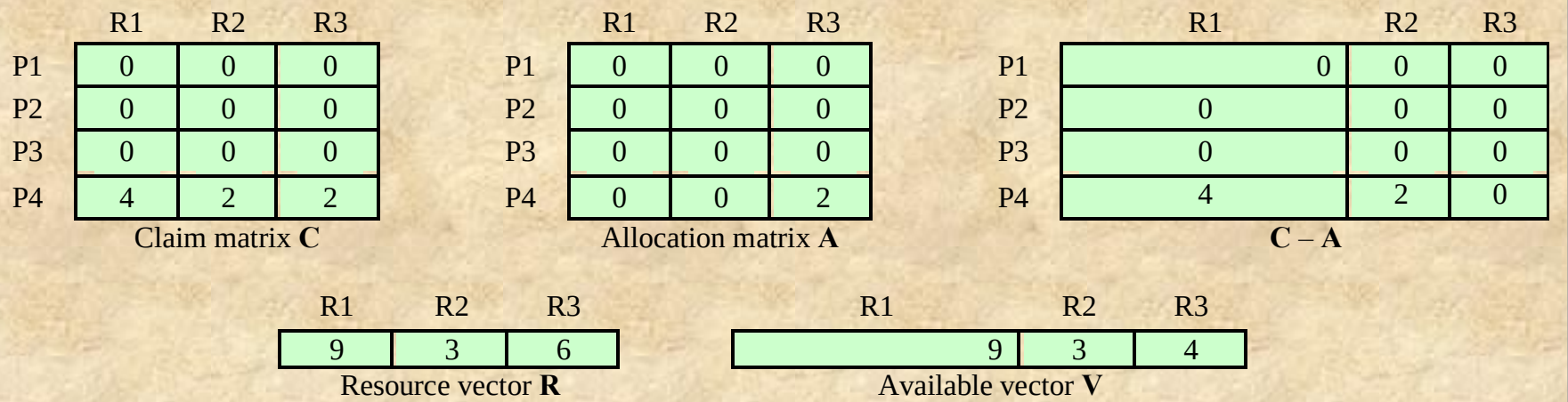
Resource vector **R**

R1	R2	R3
7	2	3

Available vector **V**

(c) P1 runs to completion

Figure 6.7 Determination of a Safe State



(d) P3 runs to completion

Figure 6.7 Determination of a Safe 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 – A

R1	R2	R3
9	3	6

Resource vector **R**

R1	R2	R3
1	1	2

Available vector **V**

(a) 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	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
P3	1	0	3
P4	4	2	0

C – A

R1	R2	R3
9	3	6

Resource vector **R**

R1	R2	R3
0	1	1

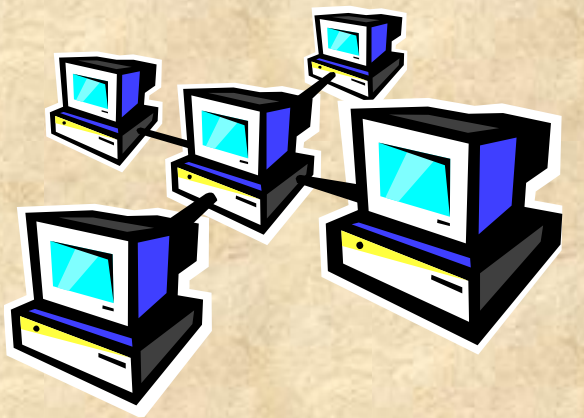
Available vector **V**

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

Figure 6.8 Determination of an Unsafe State

Deadlock Avoidance Advantages

- It is not necessary to preempt and rollback processes, as in deadlock detection
- It is less restrictive than deadlock prevention



Deadlock Avoidance Restrictions

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

Deadlock Strategies

Deadlock prevention strategies are very conservative

- limit access to resources by imposing restrictions on processes

Deadlock detection strategies do the opposite

- resource requests are granted whenever possible

Deadline Detection Algorithms

- A check for deadlock can be made as frequently as each resource request or, less frequently, depending on how likely it is for a deadlock to occur

Advantages:

- it leads to early detection
- the algorithm is relatively simple

Disadvantage

- frequent checks consume considerable processor time

Recovery Strategies

- Abort all deadlocked processes
- Back up each deadlocked process to some previously defined checkpoint and restart all processes
- Successively abort deadlocked processes until deadlock no longer exists
- Successively preempt resources until deadlock no longer exists

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 •No preemption necessary	•Inefficient •Delays process initiation •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 •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 •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 •Facilitates online handling	•Inherent preemption losses

Summary of Deadlock Detection, Prevention, and Avoidance Approaches for Operating Systems