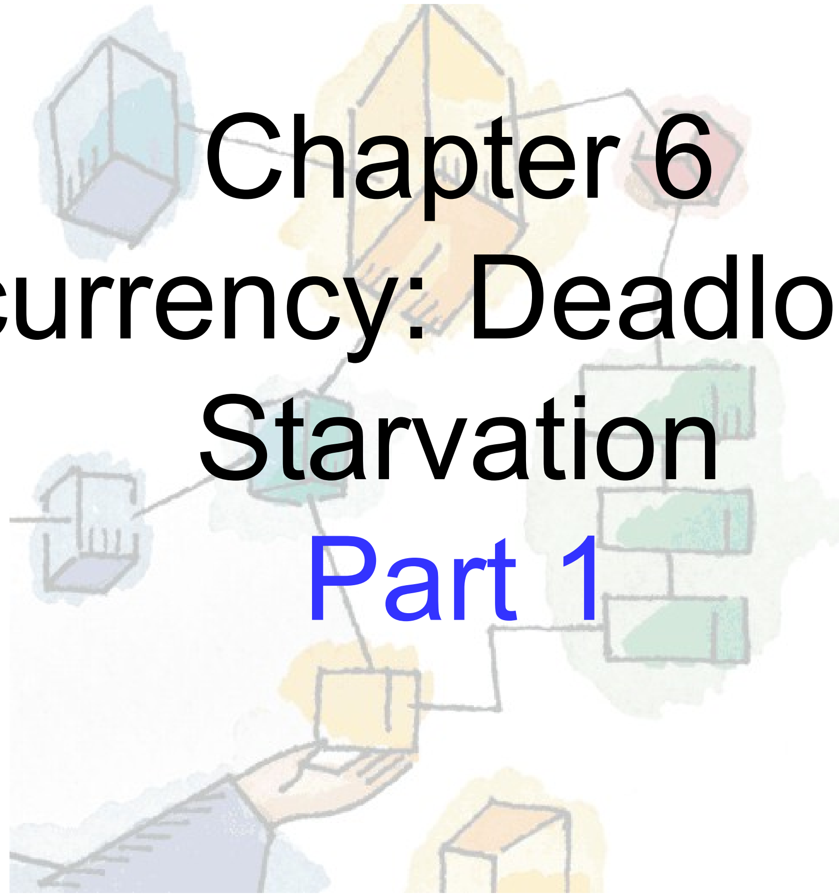
Operating Systems:
Internals and Design Principles, 8/E
William Stallings



Chapter 6 Concurrency: Deadlock and Starvation

Part 1

Patricia Roy
Manatee Community College, Venice, FL

Outline

- Principle of deadlock
- Deadlock Prevention
- Deadlock avoidance
- Deadlock Detection
- Integrated Deadlock Strategy
- Dining philosophers Problem
- UNIX concurrency Mechanisms
- Linux Kernel Concurrency Mechanism

The Anology

When two trains approach each other at a crossing, both shall come to a full stop and neither shall start up again until the other has gone. Statute passed by the Kansas State Legislature, early in the 20th century.

—*A TREASURY OF RAILROAD FOLKLORE,*
B. A. Botkin and Alvin F. Harlow

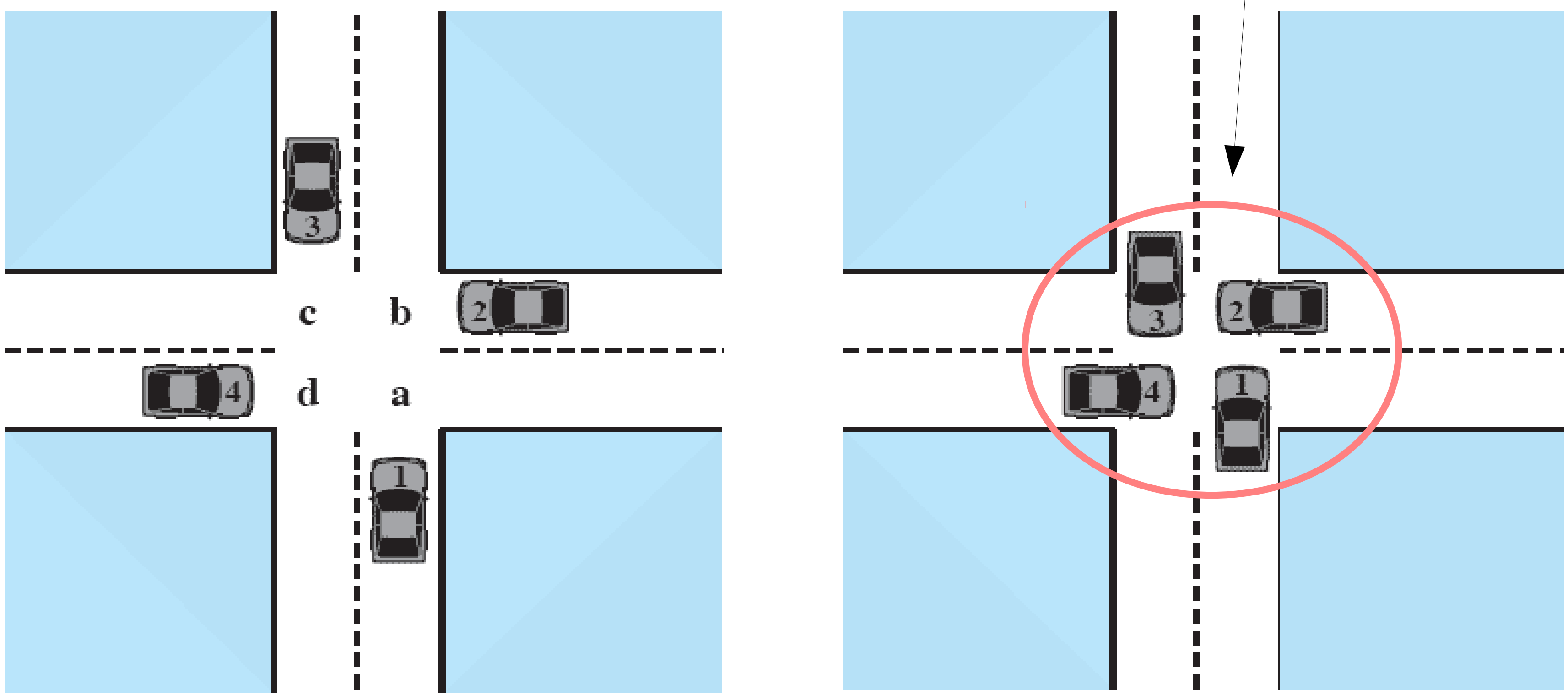
Deadlock

- 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
- Unlike other problem, there is No efficient solution
- Involve conflicting needs for resources by two or more processes

Deadlock

Resources are available

Ignore the rules and proceed



(a) Deadlock possible

(b) Deadlock

All resources are seized

Figure 6.1 Illustration of Deadlock

Deadlock

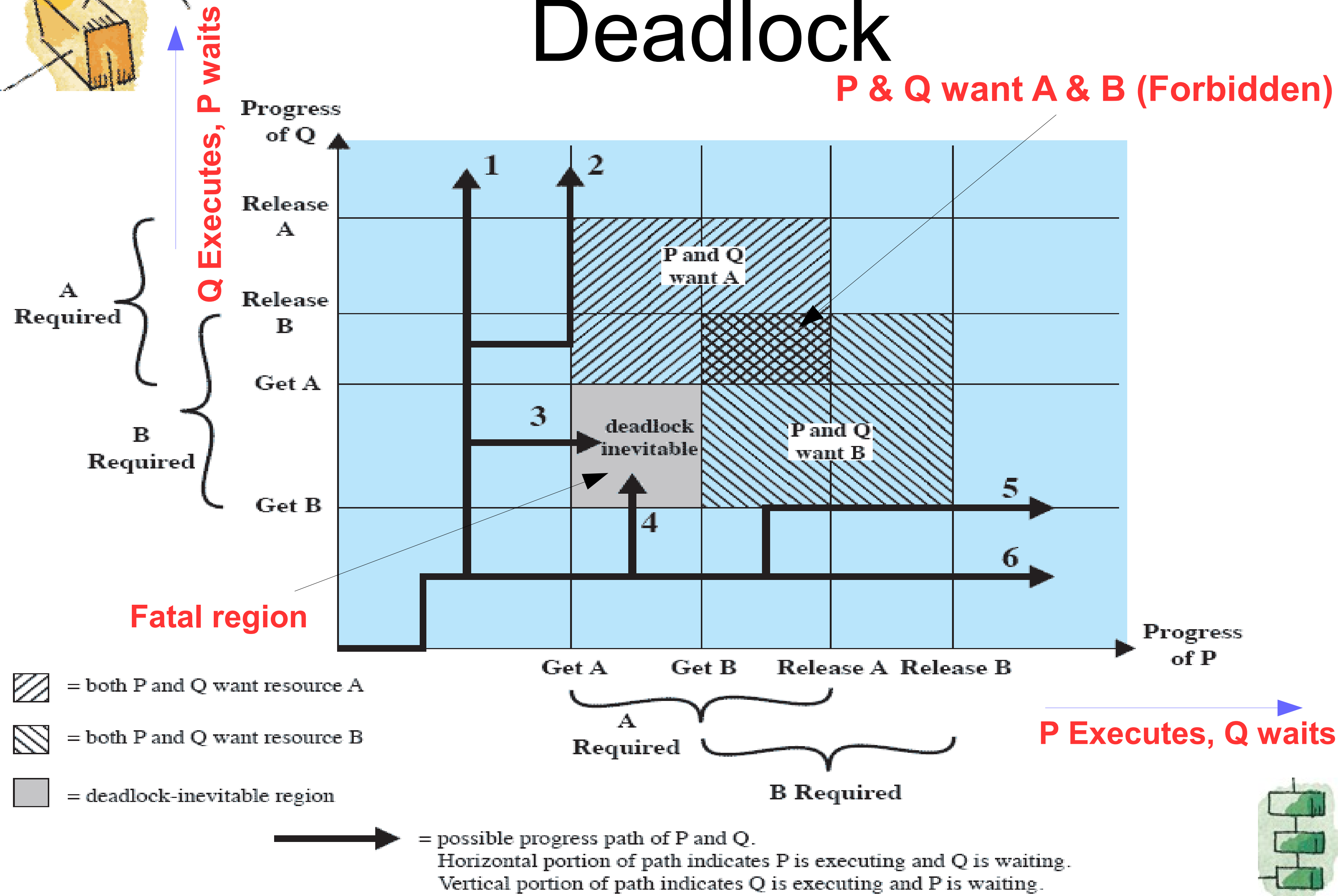


Figure 6.2 Example of Deadlock

6 different execution paths

- Q acquires B and then A and then releases B and A. When P resumes execution, it will be able to acquire both resources
- Q acquires B and then A. P executes and blocks on a request for A. Q releases B and A. When P resumes execution, it will be able to acquire both resources
- Q acquires B and then P acquires A. Deadlock is inevitable because as execution proceeds, Q will block on A and P will block on B
- P acquires A and then Q acquires B. Deadlock is inevitable because as execution proceeds, Q will block on A and P will block on B
- P acquires A and then B. Q executes and blocks on a request for B. P releases A and B. When Q resumes execution, it will be able to acquire both resources
- P acquires A and then B and then releases A and B When Q resumes execution, it will be able to acquire both resources

Deadlock

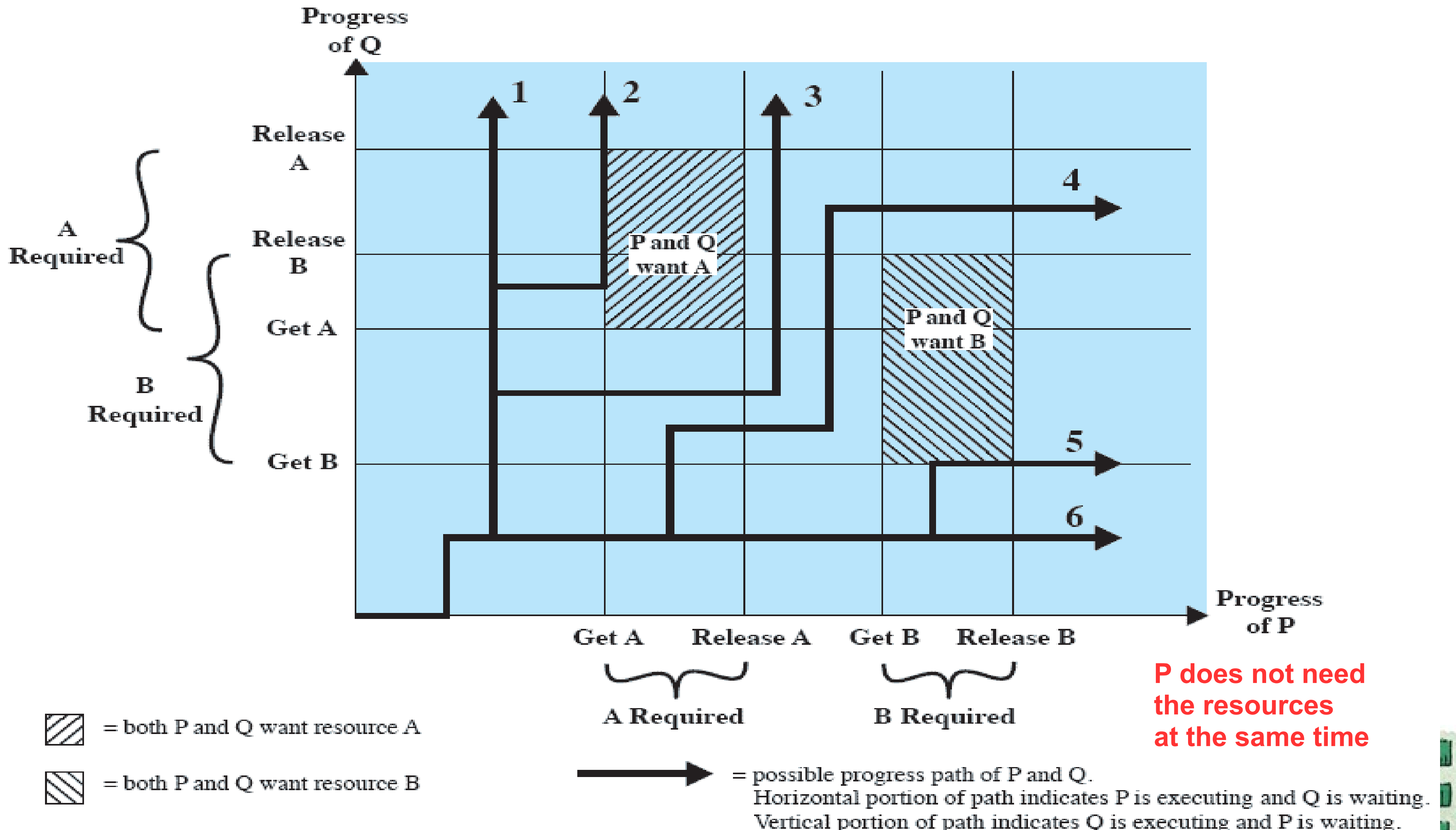


Figure 6.3 Example of No Deadlock [BACO03]

P does not need the resources at the same time

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
- E.g: 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

Reusable Resources

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)

D = Disk File   T = Tape Drive

Figure 6.4 Example of Two Processes Competing for Reusable Resources

Reusable Resources

- 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;**

- Deadlock occurs if both processes progress to their second request

Consumable Resources

- Created (produced) and destroyed (consumed)
- When acquired by a process, it is ceases to exist
- E.g: Interrupts, signals, messages, and information in I/O buffers

Example of Deadlock

- Each process attempts to receive a message from the other process, then send a message to the other one.

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

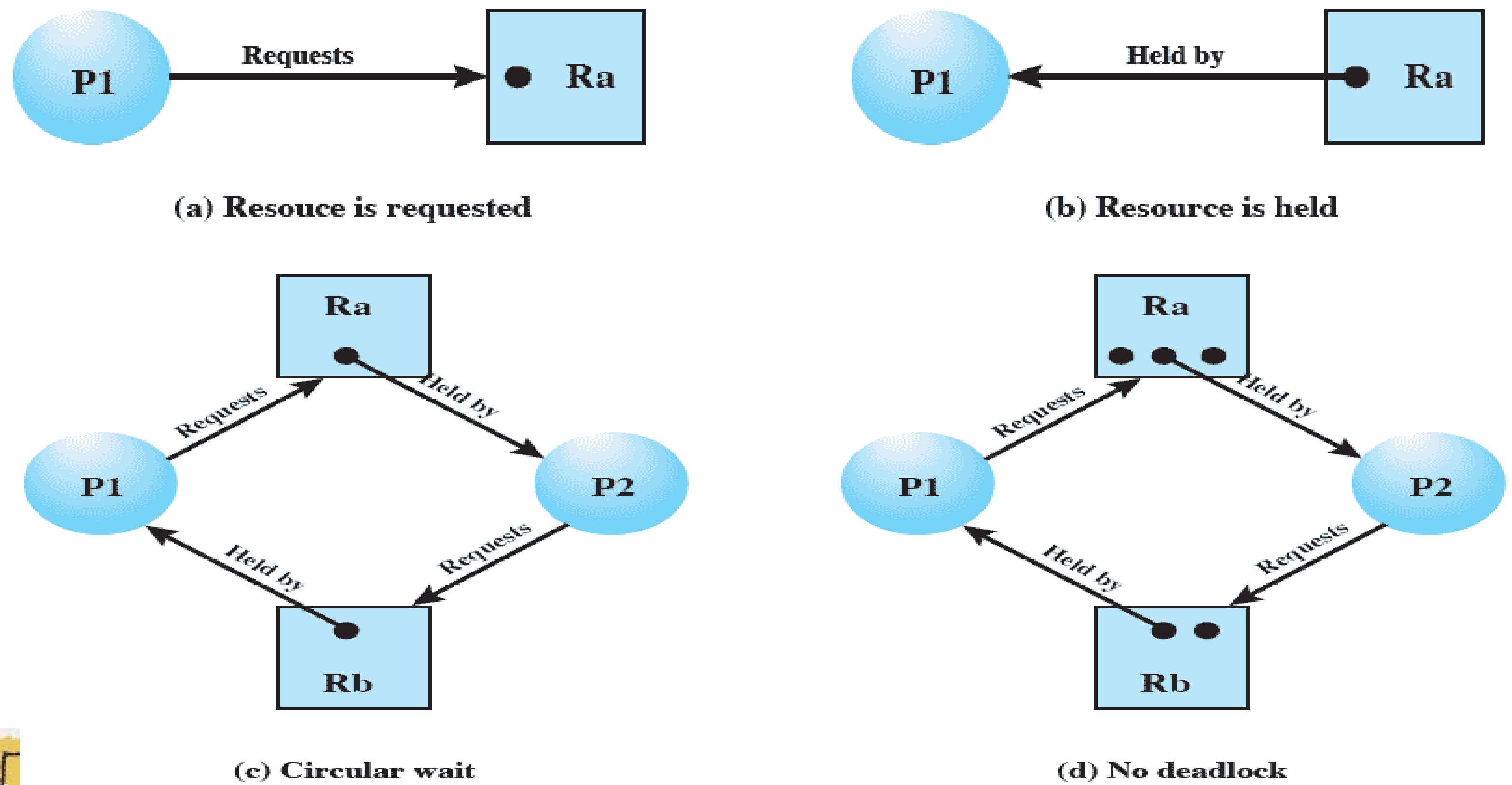
```
P2
. . .
Receive(P1);
. . .
Send(P1, M2);
```

- Deadlock may occur if a Receive message is blocking (until the message is received)
- Design error – cause deadlock, difficult to detect
- May take a rare combination of events to cause deadlock

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

Resource Allocation Graphs

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



Deadlock

Many resources

Resource Allocation Graphs

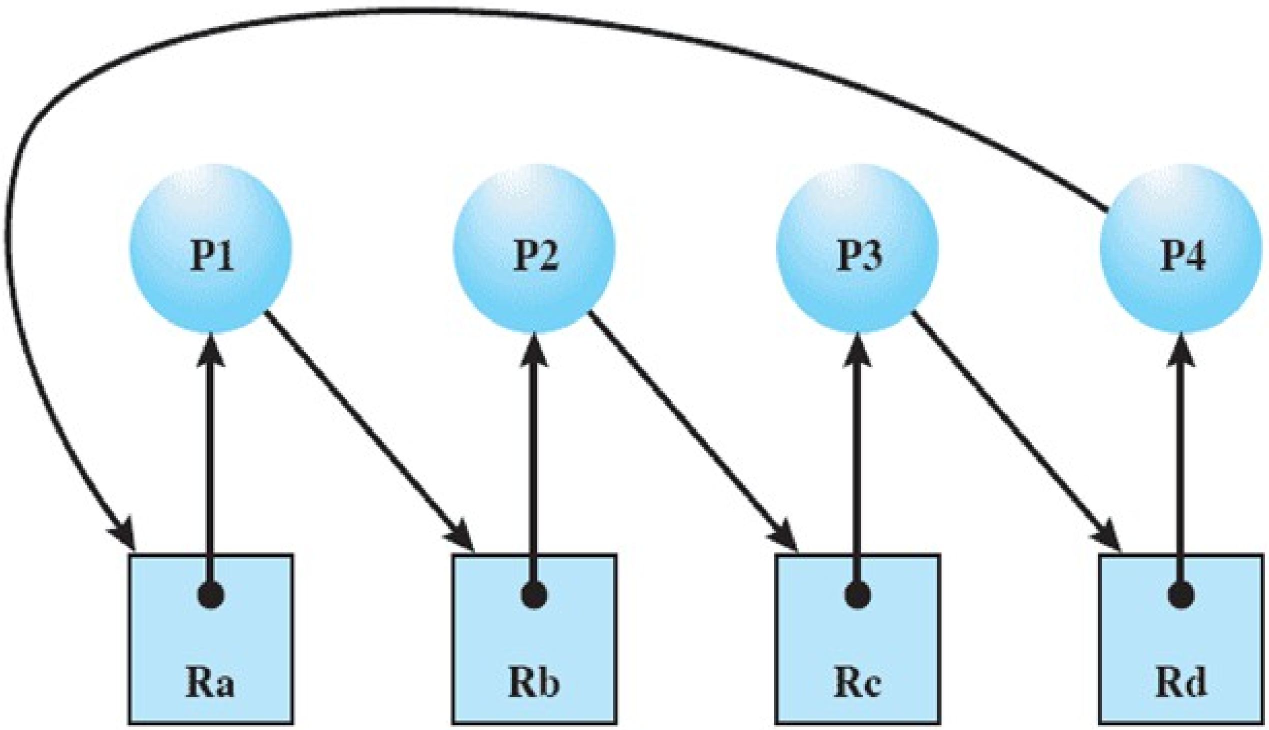


Figure 6.6 Resource Allocation Graph for Figure 6.1b

Conditions for Deadlock

Necessary, but not sufficient

- Mutual exclusion
  - Only one process may use a resource at a time
- Hold-and-wait
  - A process may hold allocated resources while awaiting assignment of others
- No preemption
  - No resource can be forcibly removed from a process holding it
- 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

For a deadlock to actually take place

Possibility of Deadlock	Existence of Deadlock
Mutual Exclusion	Mutual Exclusion
No preemption	No preemption
Hold and wait	Hold and wait
	Circular Wait

Unresolvable

Deadlock Prevention

- **Mutual Exclusion (ME)**
  - If access to resource require ME, it must be supported by the OS. Eg: file: multiple read, single write
- **Hold and Wait**
  - Require a process request all of its required resources at one time. Inefficient: longtime waiting for all resources, remain unused, denied to other processes.
- **No Preemption**
  - First: if a process holding certain resources is denied a further request, that process must release its original resources and request again
  - If the resources are held by others, OS may preempt the 2nd process to require it releases its resources (if no 2 processes possessed the same priority)
- **Circular Wait**
  - Define a linear ordering of resource types

Deadlock Avoidance

- Differs subtly from deadlock prevention
- Allows the 3 necessary conditions, but make intelligent choices to assure that deadlock point is never reached
  - *Is the current resource alloc. will lead to potential deadlock?*
- Thus, it allows more concurrency than prevention
- A decision is made dynamically whether the current resource allocation request will, if granted, potentially lead to a deadlock
- Thus, requires knowledge of future process requests

Two Approaches to Deadlock Avoidance

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