Part 6

Deadlocks

6.1. Resource
6.2. Introduction to deadlocks
6.3. The ostrich algorithm
6.4. Deadlock detection and recovery
6.5. Deadlock avoidance
6.6. Deadlock prevention
6.7. Other issues

Chapter Objectives

- To develop a description of deadlocks, which prevent sets of concurrent processes from completing their tasks
- To present a number of different methods for preventing or avoiding deadlocks in a computer system.

6.1. Resource

Resources(1)

- Examples of computer resources
  - printers
  - tape drives
  - tables
- Processes need access to resources in reasonable order
- Suppose a process holds resource X and requests resourceY
  - at same time another process holds Y and requests X
  - both are blocked and remain so

Resources (2)

- Deadlocks occur when …
  - processes are granted exclusive access to devices
  - we refer to these devices generally as resources
- Preemptable resources
  - can be taken away from a process with no ill effects
- Nonpreemptable resources
  - will cause the process to fail if taken away

Resources (3)

- Sequence of events required to use a resource
  1. request the resource
  2. use the resource
  3. release the resource
- Must wait if request is denied
  - requesting process may be blocked
  - may fail with error code

Resources (4)

- Example request/release as system call
  - request/release device
  - open/close file
  - allocate/free memory
  - wait/signal

6.2. Introduction to deadlocks

Introduction to Deadlocks

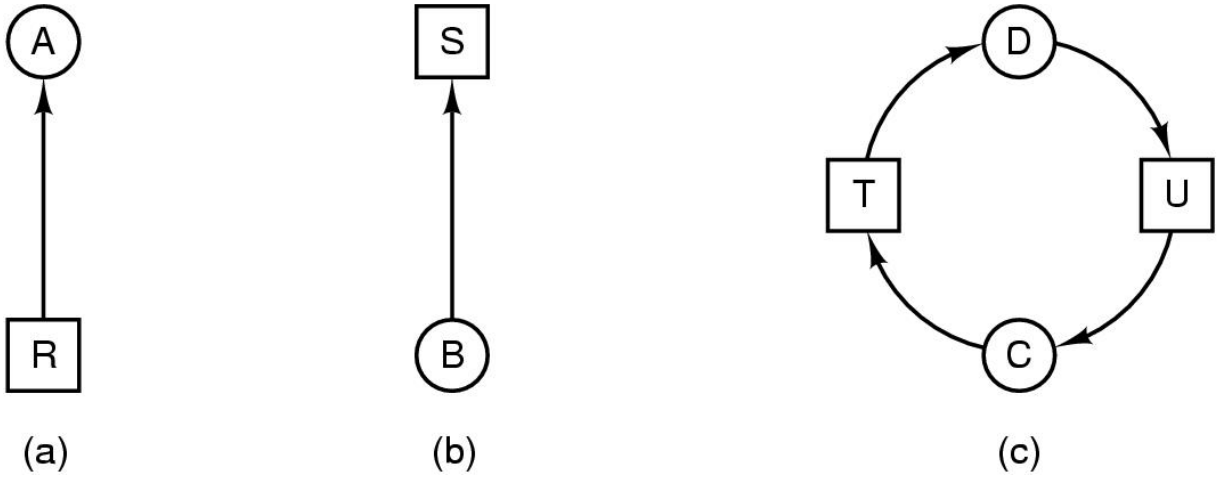
- Formal definition :
  A set of processes is deadlocked if each process in the set is waiting for an event that only another process in the set can cause
- Usually the event is release of a currently held resource
- None of the processes can …
  - run
  - release resources
  - be awakened

Four Conditions for Deadlock

1. Mutual exclusion condition
   - each resource assigned to 1 process or is available
2. Hold and wait condition
   - process holding resources can request additional
3. No preemption condition
   - previously granted resources cannot forcibly taken away
4. Circular wait condition
   - must be a circular chain of 2 or more processes
   - each is waiting for resource held by next member of the chain

Deadlock Modeling (1)

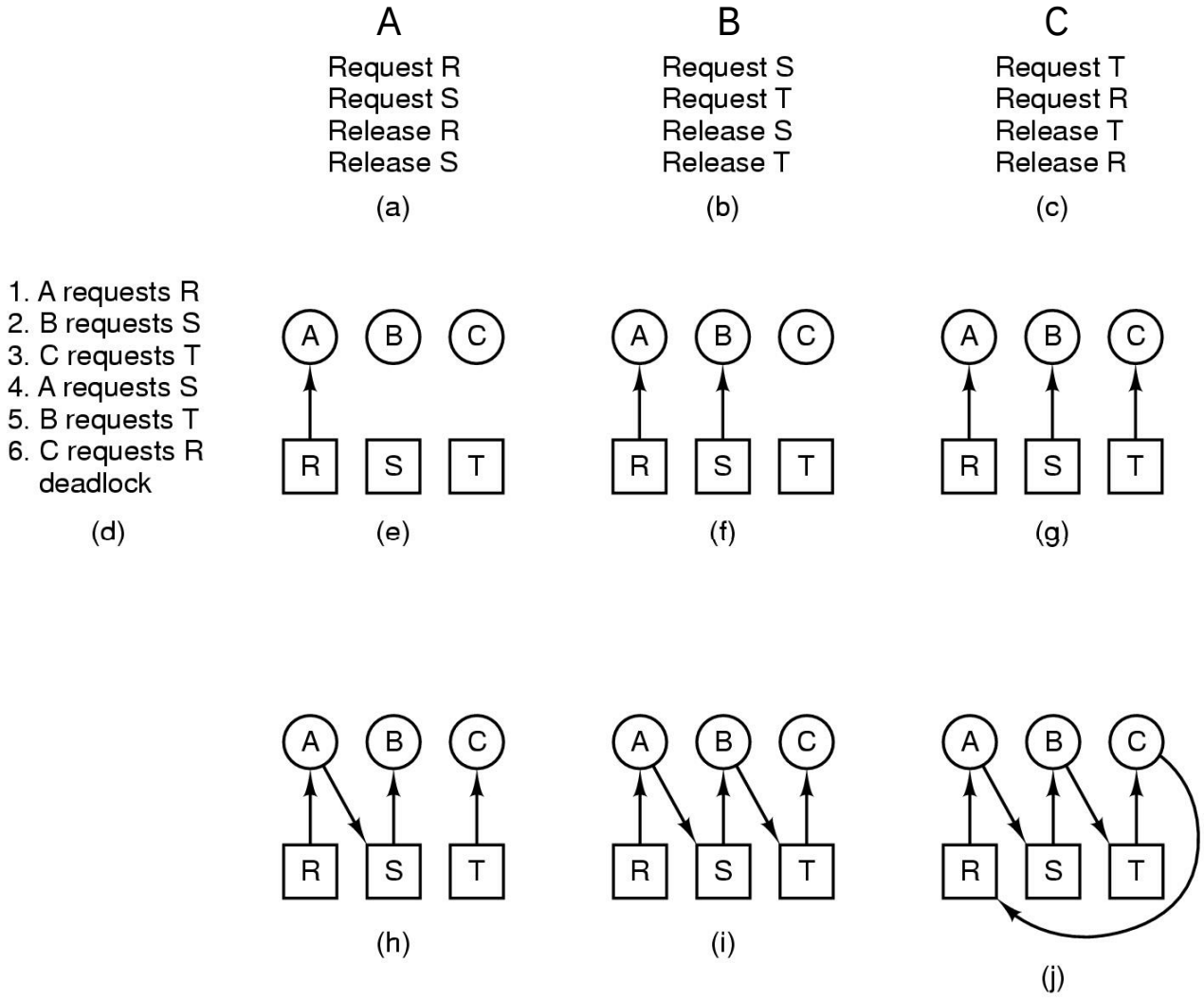
- Modeled with directed graphs
- Resource-Allocation Graph (RAG)



(a) (b) (c)

  - resource R assigned to process A
  - process B is requesting/waiting for resource S
  - process C and D are in deadlock over resources T and U

Deadlock Modeling (2)

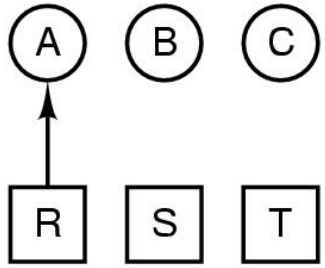


How deadlock occurs

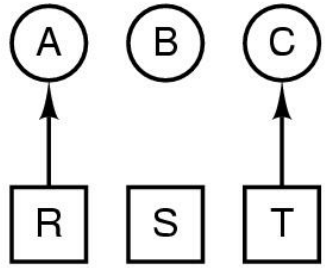
Deadlock Modeling (3)

1. A requests R
2. C requests T
3. A requests S
4. C requests R
5. A releases R
6. A releases S
   no deadlock

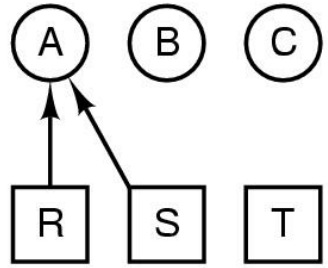
(k)



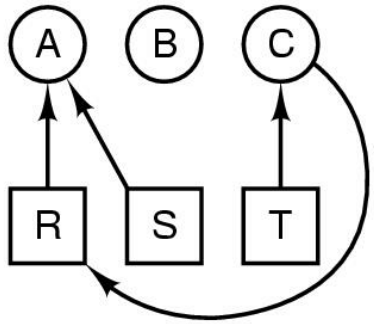
(l)



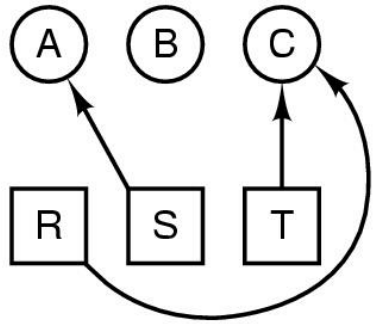
(m)



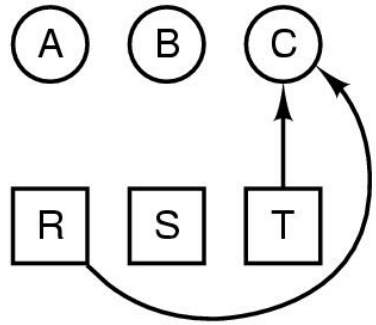
(n)



(o)



(p)



(q)

How deadlock can be avoided

Strategies for dealing with Deadlocks

1. just ignore the problem altogether
2. detection and recovery
3. dynamic avoidance
   - careful resource allocation
4. prevention
   - negating one of the four necessary conditions

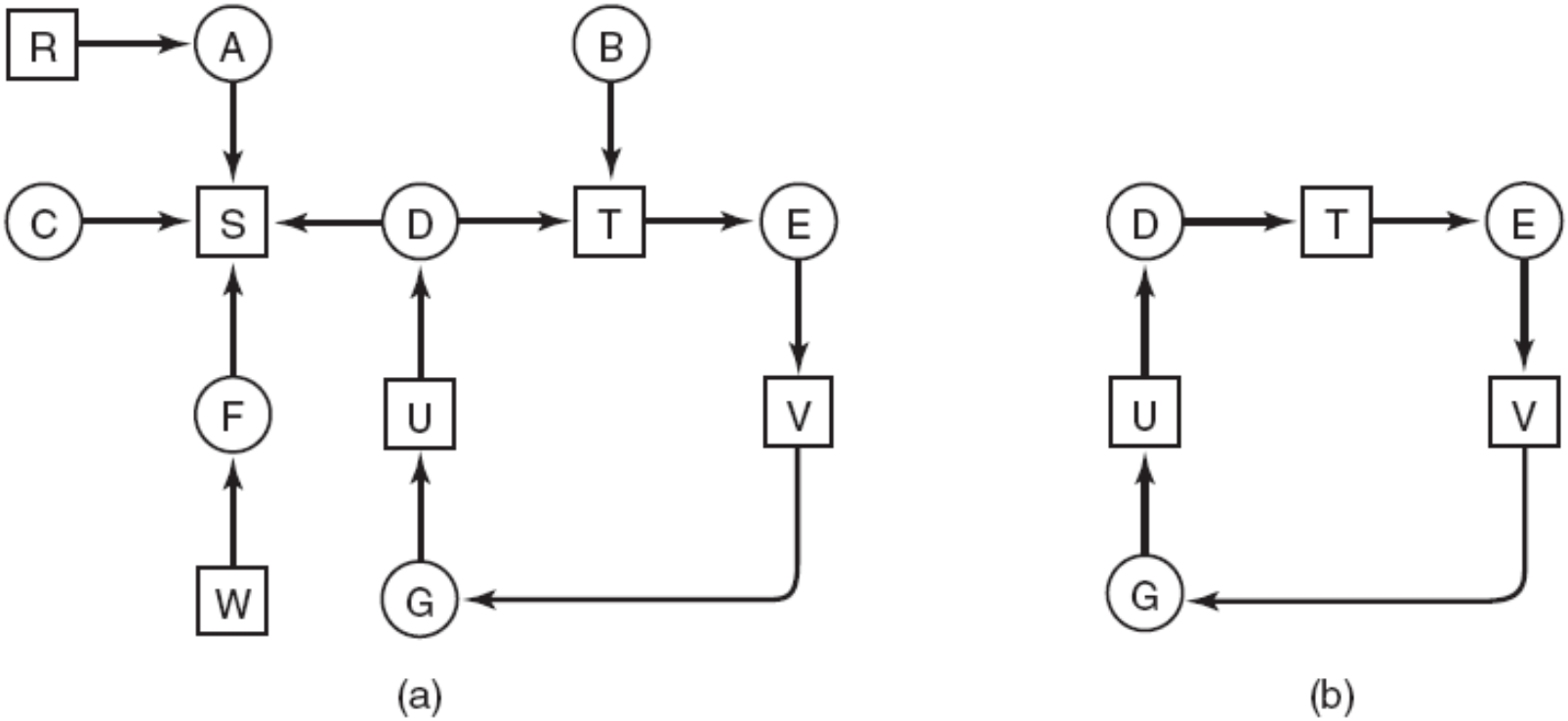
6.3. The ostrich algorithm

The Ostrich Algorithm

- Pretend there is no problem
- Reasonable if
  - deadlocks occur very rarely
  - cost of prevention is high
- UNIX and Windows takes this approach
- It is a trade off between
  - convenience
  - correctness

6.4. Deadlock detection and recovery

Deadlock Detection with One Resource of Each Type (1)



Deadlock Detection with One Resource of Each Type (2)

Algorithm for detecting deadlock:

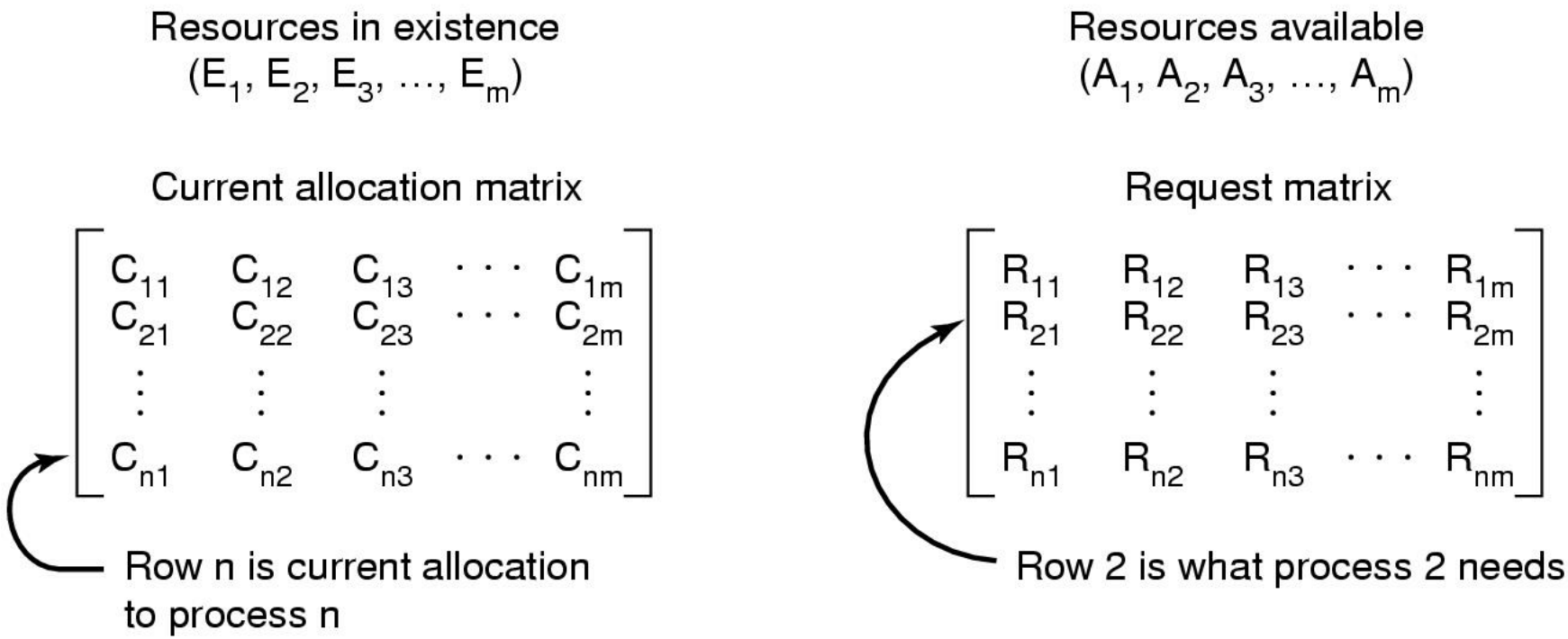
1. For each node, N in the graph, perform the following five steps with N as the starting node.
2. Initialize L to the empty list, designate all arcs as unmarked.
3. Add current node to end of L, check to see if node now appears in L two times. If it does, graph contains a cycle (listed in L), algorithm terminates.

   …

Deadlock Detection with One Resource of Each Type (3)

4. From given node, see if any unmarked outgoing arcs. If so, go to step 5; if not, go to step 6.
5. Pick an unmarked outgoing arc at random and mark it. Then follow it to the new current node and go to step 3.
6. If this is initial node, graph does not contain any cycles, algorithm terminates. Otherwise, dead end. Remove it, go back to previous node, make that one current node, go to step 3.

Detection with Multiple Resource of Each Type (1)



Data structures needed by deadlock detection algorithm

Detection with Multiple Resource of Each Type (2)

The deadlock detection algorithm:

1. Look for unmarked process, P_i, for which the i-th row of R is less than or equal to A
2. If such process is found, add the i-th row of C to A , mark the process and go back to step 1
3. If no such process exists, the algorithm terminates.

When algorithm terminates, any unmarked processes are known to be dealocked

Detection with Multiple Resource of Each Type (3)

	Tape drives	Plotters	Scanners	CD Roms
E = (	4	2	3	1)

	Tape drives	Plotters	Scanners	CD Roms
A = (	2	1	0	0)

Current allocation matrix

$$C = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 \\ 0 & 1 & 2 & 0 \end{bmatrix}$$

Request matrix

$$R = \begin{bmatrix} 2 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 2 & 1 & 0 & 0 \end{bmatrix}$$

An example for the deadlock detection algorithm

After first cycle A=(2 2 2 0),

After second cycle A=(4 2 2 1)

Recovery from Deadlock (1)

- Recovery through preemption
  - take a resource from some other process
  - depends on nature of the resource
- Recovery through rollback
  - checkpoint a process periodically
  - use this saved state
  - restart the process if it is found deadlocked

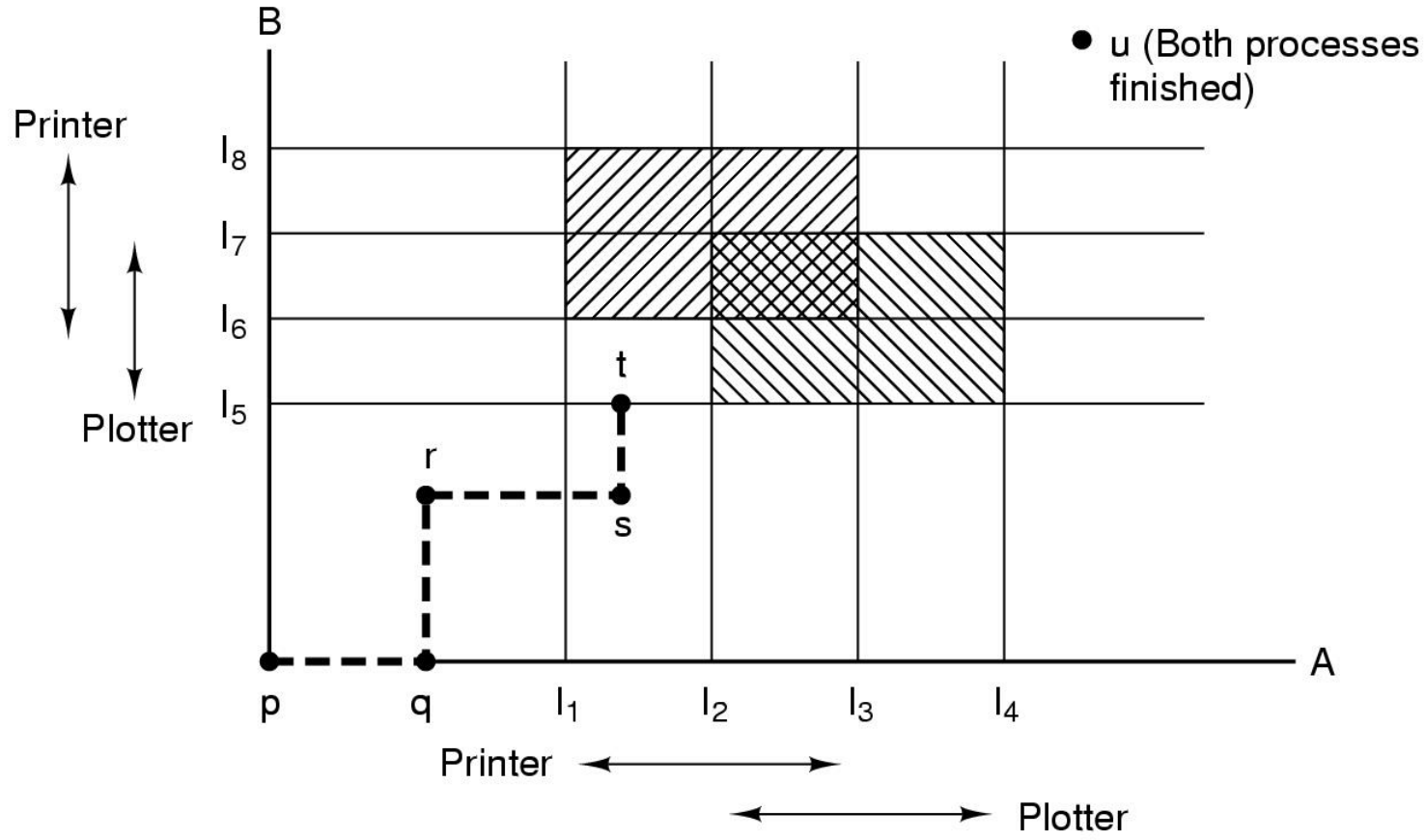
Recovery from Deadlock (2)

- Recovery through killing processes
  - crudest but simplest way to break a deadlock
  - kill one of the processes in the deadlock cycle
  - the other processes get its resources
  - choose process that can be rerun from the beginning

6.5. Deadlock avoidance

Deadlock Avoidance

Resource Trajectories



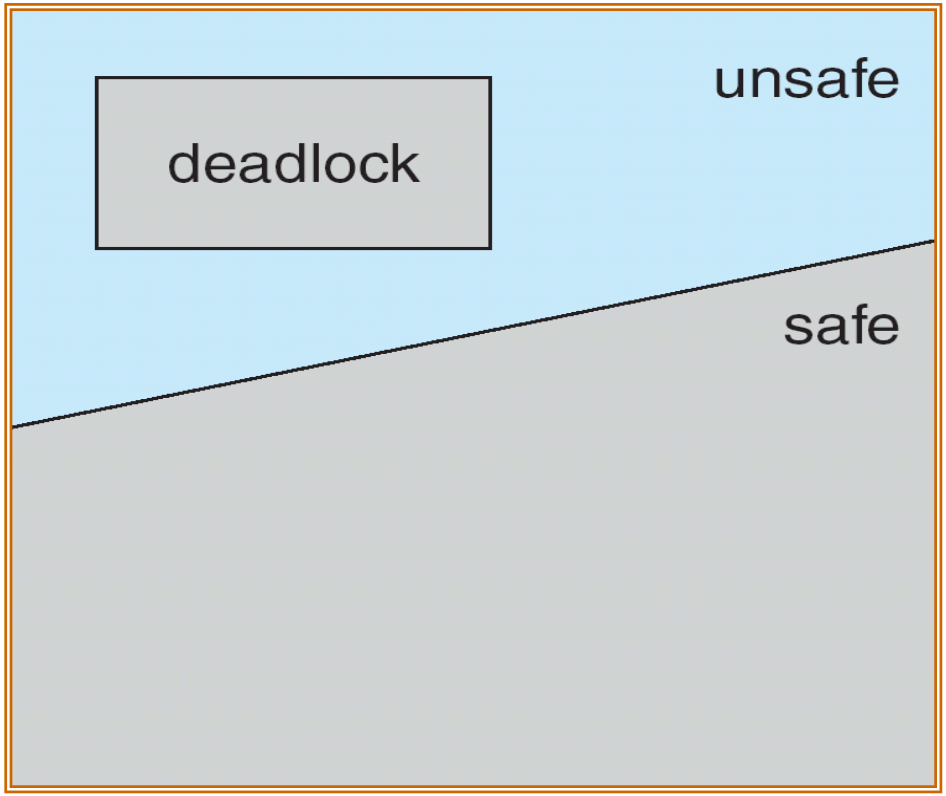
Two process resource trajectories

Deadlock Avoidance
Basic Facts

- At any instant of time, current state of system consisting of E (Resources in Existance), A (Resource Available), C (Current allocation matrix), R (Request matrix)
- 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.

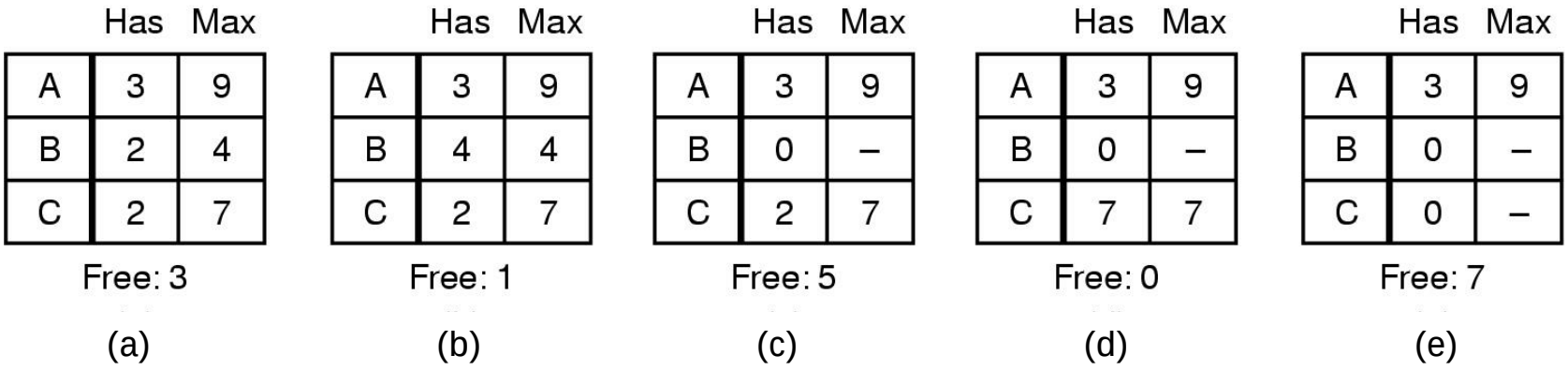
Deadlock Avoidance

Safe, Unsafe , Deadlock State



Deadlock Avoidance

Safe and Unsafe States (1)



	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)

- Example: 3 processses A, B, C using one resource with total 10 instances, 7 already allocated, 3 available
- Demonstration that the state in (a) is safe

Deadlock Avoidance

Safe and Unsafe States (2)

	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)

Demonstration that the state in b is not safe

Deadlock Avoidance

The Banker's Algorithm for a Single Resource (1)

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

Free: 10

(a)

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

Free: 2

(b)

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

Free: 1

(c)

- Three resource allocation states
  - (a) safe
  - (b) safe
  - (c) unsafe

Deadlock Avoidance

The Banker's Algorithm for a Single Resource (2)

- The banker’s algorithm considers each request as it occurs, and see if granting it leads to a safe state.
- If it does, the request is granted; otherwise, it is postponed until later.
- To see if a state is safe, the banker checks to see if he has enough resources to satisfy some customer.
- If so, those loans are assumed to be repaid, and the customer now closest to the limit is checked, and so on. If all loans can eventually be repaid, the state is safe and the initial request can be granted.

Deadlock Avoidance

Banker's Algorithm for Multiple Resources (1)

Process	Tape drives	Plotters	Scanners	CD ROMs
A	3	0	1	1
B	0	1	0	0
C	1	1	1	0
D	1	1	0	1
E	0	0	0	0

Resources assigned

Process	Tape drives	Plotters	Scanners	CD ROMs
A	1	1	0	0
B	0	1	1	2
C	3	1	0	0
D	0	0	1	0
E	2	1	1	0

Resources still needed

E = (6342)
P = (5322)
A = (1020)

Example of banker's algorithm with multiple resources

If order is D, E, A, B, C, Vector A will be (2121), (2121), (5132), (5232), (6342)

Deadlock Avoidance

Banker's Algorithm for Multiple Resources (2)

- The algorithm for checking to see if a state is safe can be stated.

1. Look for a row, R, whose unmet resource needs are all smaller than or equal to A. If no such row exists, the system will eventually deadlock since no process can run to completion.
2. Assume the process of the row chosen requests all the resources it needs and finishes. Mark that process as terminated and add all its resources to the A vector.
3. Repeat steps 1 and 2 until either all processes are marked terminated, in which case the initial state was safe, or until a deadlock occurs, in which case it was not.

6.6. Deadlock prevention

Deadlock Prevention

Attacking the Mutual Exclusion Condition

- Some devices (such as printer) can be spooled
  - only the printer daemon uses printer resource
  - thus deadlock for printer eliminated
- Not all devices can be spooled
- Principle:
  - avoid assigning resource when not absolutely necessary
  - as few processes as possible actually claim the resource

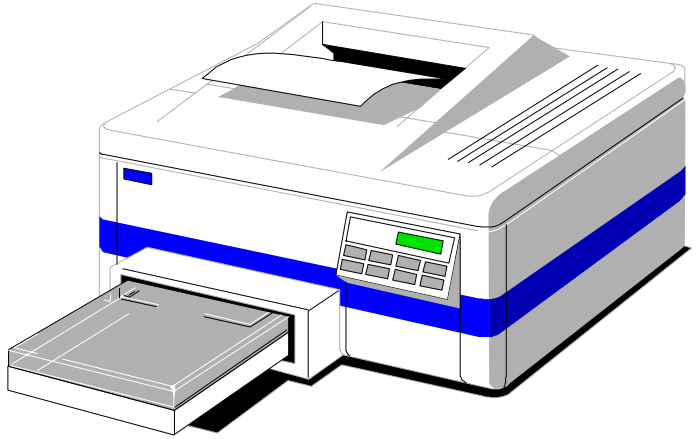
Deadlock Prevention

Attacking the Hold and Wait Condition

- Require processes to request resources before starting
  - a process never has to wait for what it needs
- Problems
  - may not know required resources at start of run
  - also ties up resources other processes could be using
- Variation:
  - process must give up all resources
  - then request all immediately needed

Deadlock Prevention
Attacking the No Preemption Condition

- This is not a viable option
- Consider a process given the printer
  - halfway through its job
  - now forcibly take away printer
  - !!??

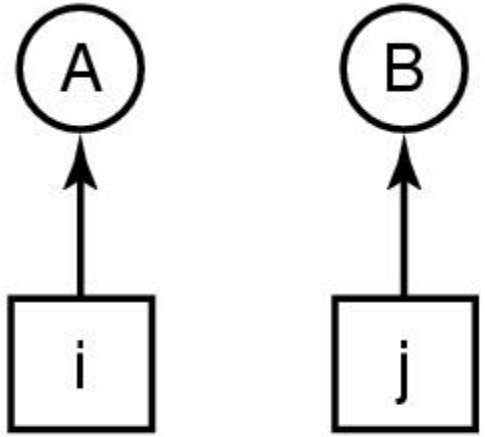


Deadlock Prevention

Attacking the Circular Wait Condition (1)

1. Imagesetter
2. Scanner
3. Plotter
4. Tape drive
5. CD Rom drive

(a)



(b)

- Normally ordered resources
- A resource graph

Deadlock Prevention

Condition	Approach
Mutual exclusion	Spool everything
Hold and wait	Request all resources initially
No preemption	Take resources away
Circular wait	Order resources numerically

Summary of approaches to deadlock prevention

6.7. Other issues

Other Issues
Two-Phase Locking

- Phase One
  - process tries to lock all records it needs, one at a time
  - if needed record found locked, start over
  - (no real work done in phase one)
- If phase one succeeds, it starts second phase,
  - performing updates
  - releasing locks
- Note similarity to requesting all resources at once
- Algorithm works where programmer can arrange
  - program can be stopped, restarted

Nonresource Deadlocks

- Possible for two processes to deadlock
  - each is waiting for the other to do some task
- Can happen with semaphores
  - each process required to do a *down()* on two semaphores (*mutex* and another)
  - if done in wrong order, deadlock results

Starvation

- Algorithm to allocate a resource
  - may be to give to shortest job first
- Works great for multiple short jobs in a system
- May cause long job to be postponed indefinitely
  - even though not blocked
- Solution:
  - First-come, first-serve policy