CISC 3320

C24c Deadlock Avoidance

Hui Chen

Department of Computer & Information Science

CUNY Brooklyn College

Acknowledgement

- These slides are a revision of the slides provided by the authors of the textbook via the publisher of the textbook

Outline

- Deadlock Avoidance
  - by carefully allocating (non-sharable) resources
- Deadlock Detection
- Recovery from Deadlock

Deadlock Avoidance

- Carefully allocates (non-sharable) resources
  - The deadlock-avoidance algorithm dynamically examines the *resource-allocation state* to ensure that there can never be a *circular-wait condition*, i.e., in a safe state

Information Known *A* Priori

- Requires that the system has some additional ***a priori*** information available
  - Simplest and most useful model requires that each process declare the ***maximum number*** of resources of each type that it may need

Resource-Allocation State

- Resources available (the numbers of instances of and the types of resources available)
- Resource allocated (the numbers of instances of and the types of resources allocated)
- Maximum demands (the number of instances of and types) of resources of the threads

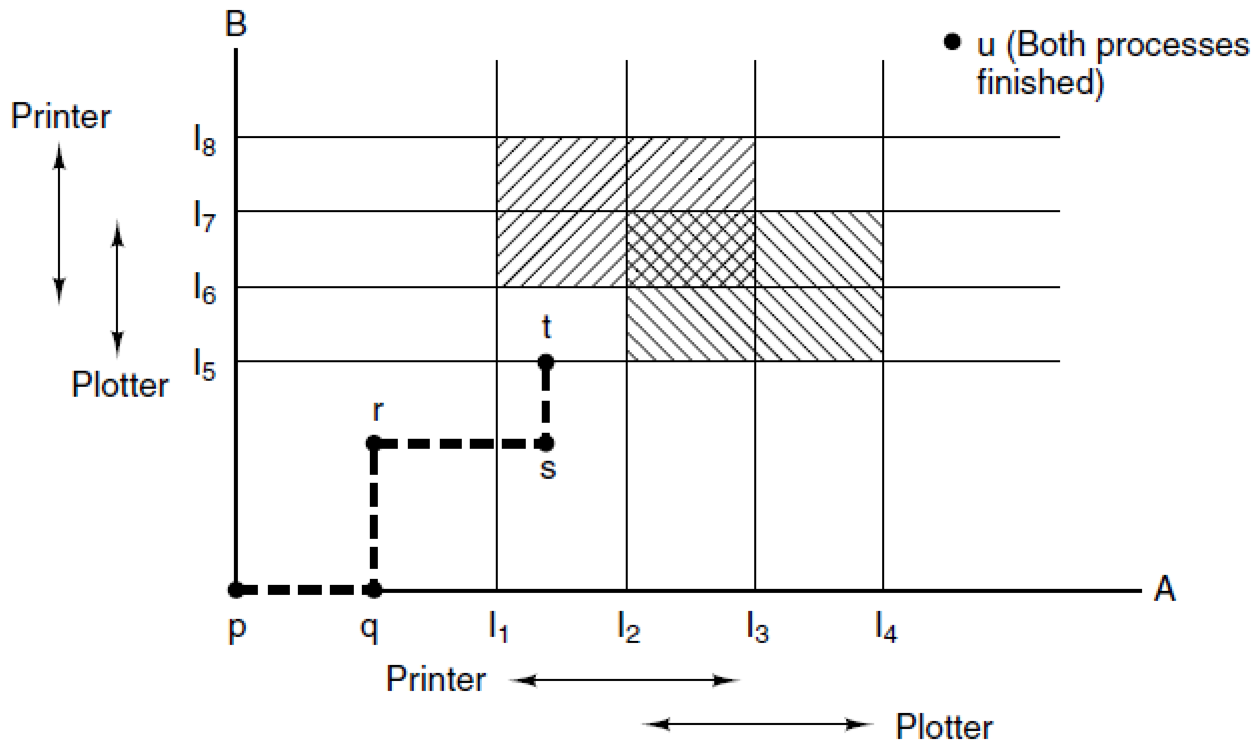
Define Safe State

- System is in **safe state** if there exists a sequence <P_1, P_2, …, P_n> of ALL the processes in the systems such that for each P_i, the resources that P_i can still request can be satisfied by currently available resources + resources held by all the P_j, with $j < I$

Define Safe State: Scenarios

- That is:
  - If P_i resource needs are not immediately available, then P_i can wait until all P_j have finished
  - When P_j is finished, P_i can obtain needed resources, execute, return allocated resources, and terminate
  - When P_i terminates, P_{i+1} can obtain its needed resources, and so on

Resource Trajectory



- [Figure 6-8 in Tanenbaum & Bos, 2014]

Safe and Unsafe State

- Safe state
  - The system can allocate resources to each process in some order and still avoid a deadlock
  - A safe state is not a deadlocked state
- Unsafe state
  - A deadlocked state is an unsafe state
  - An unsafe state may not be a deadlock state
  - An unsafe state is a state that <u>may</u> lead to a deadlock

Safe State: Example

- A resources has 10 instances
- Does exist a scheduling order of processes A, B, C, and allow all of them to complete?
    - The following sequence shows that (a) is safe

	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)

Unsafe State: Example

- A resources has 10 instances
- Does exist a scheduling order of processes A, B, C, and allow all of them to complete?
  - (b) is unsafe: you can run B to completion, but no sufficient resources for A or C to complete

	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)

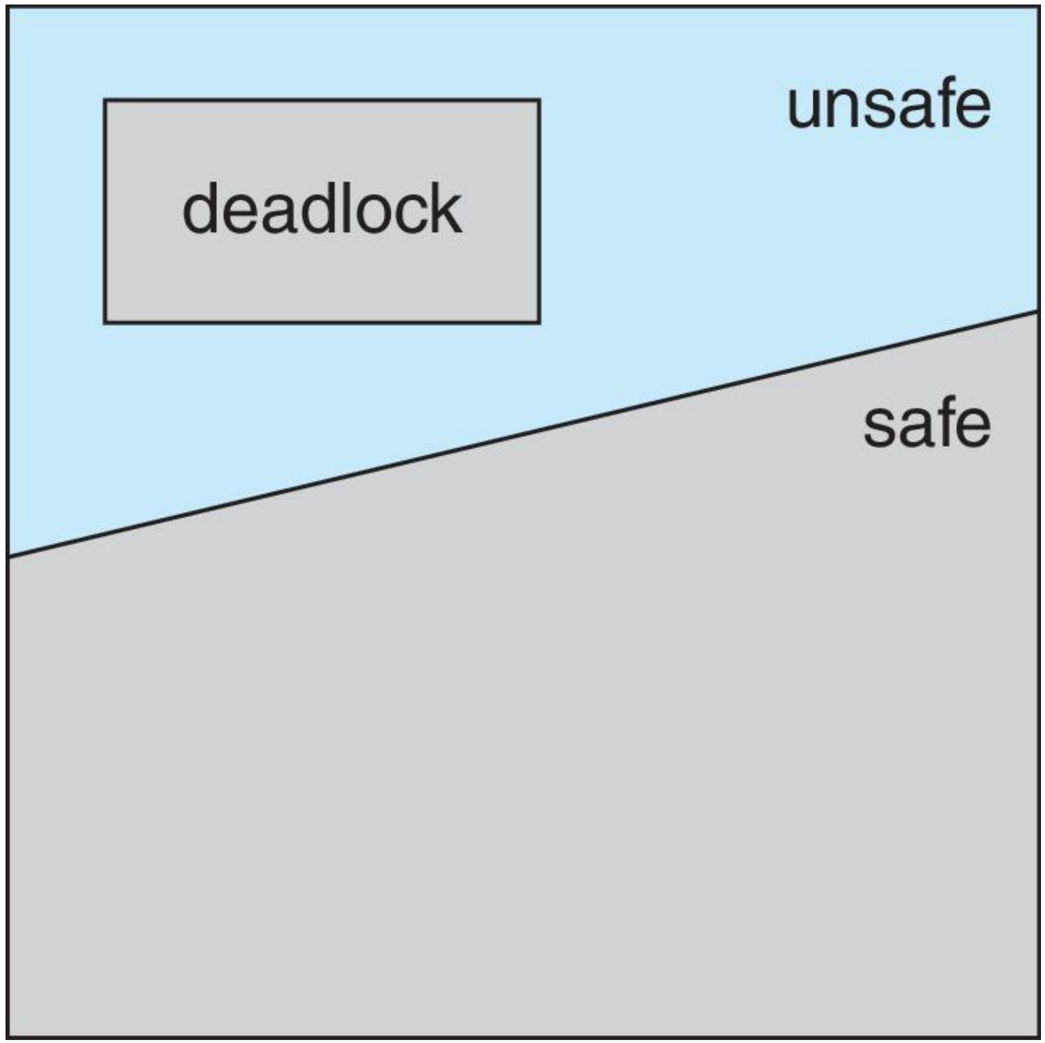
	Has	Max
A	4	9
B	—	—
C	2	7

Free: 4
(d)

Safe State and Deadlocks

- 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.

Safe, Unsafe, Deadlock State



Deadlock Avoidance Algorithms

- If the system enters an unsafe state when the system grants the resource request

- Single instance of a resource type
  - Use a resource-allocation graph
- Multiple instances of a resource type
  - Use the Banker's Algorithm

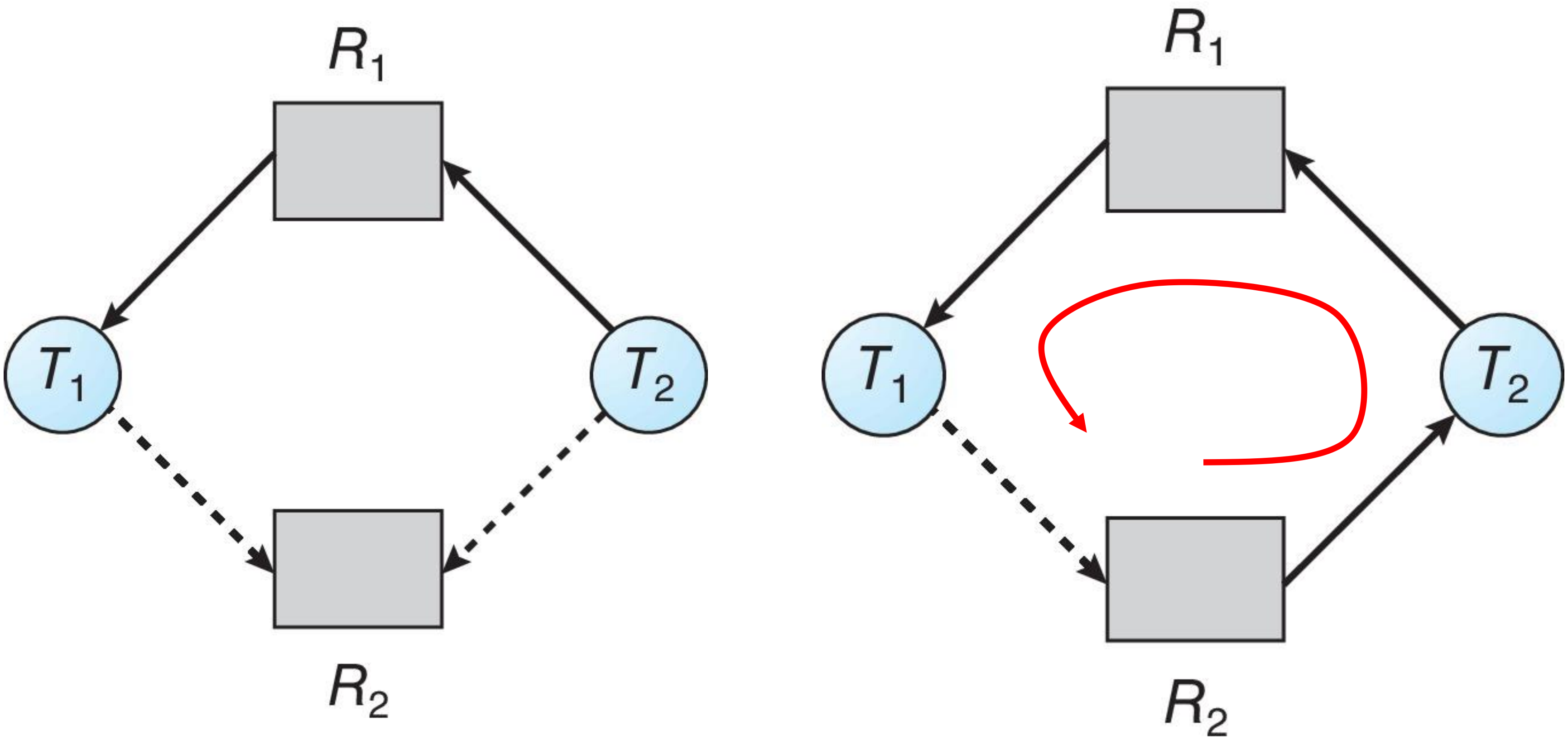
Questions?

- Deadlock avoidance
  - Resource allocation
  - Resource allocation state
  - Safe and unsafe sates
- When to use?
  - The resource allocation graph
  - The Banker's algorithm

Resource-Allocation Graph Scheme

- **Claim edge** P_i --> R_j indicated that process P_j may request resource R_j; represented by a dashed line
- Claim edge converts to request edge $P_i \rightarrow R_j$ when a process requests a resource
- Request edge converted to an assignment edge $P_i \leftarrow R_j$ when the resource is allocated to the process
- When a resource is released by a process, assignment edge reconverts to a claim edge
- Resources must be claimed *a priori* in the system

Resource-Allocation Graph Scheme: Example



Resource Allocation Graph Algorithm:

- Suppose that process P_i requests a resource R_j
- The request can be granted only if converting the request edge to an assignment edge does not result in the formation of a cycle in the resource allocation graph
  - For each node in the graph if the request granted,
    - Do a depth first search, check if cycle exists
  - Complexity of the algorithm: $O(N^2)$ (N: the number of processes)

Questions?

- Single instance of resources
- Resource allocation graph algorithm
- Safe and unsafe state?
- How about a resource has multiple instances?

Banker's Algorithm: Assumptions

- Multiple instances of resources
- Each process must *a priori* claim maximum use
- 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

Data Structures for the Banker's Algorithm

- Let n = number of processes, and m = number of resources types.
- **Available**: Vector of length m. If available $[j] = k$, there are k instances of resource type R_j available
- **Max**: $n \times m$ matrix. If $Max[i,j] = k$, then process P_i may request at most k instances of resource type R_j
- **Allocation**: $n \times m$ matrix. If $Allocation[i,j] = k$ then P_i is currently allocated k instances of R_j
- **Need**: $n \times m$ matrix. If $Need[i,j] = k$, then P_i may need k more instances of R_j to complete its task

$$Need[i,j] = Max[i,j] - Allocation[i,j]$$

Safety Algorithm

1. Let ***Work*** and ***Finish*** be vectors of length *m* and *n*, respectively. Initialize

   Work = Available

   ***Finish* [*i*] = *false* for *i* = 0, 1, …, *n*- 1**

2. Find an ***i*** such that both

   (a) ***Finish* [*i*] = *false***

   (b) ***Need$_i$*** $\leq$ ***Work***

   If no such ***i*** exists, go to step 4

3. Work = Work + Allocation$_i$

   Finish*[*i*] = *true
   go to step 2

4. If ***Finish* [*i*] == *true*** for all ***i***, then the system is in a safe state

Resource-Request Algorithm for Process P_i

Request**$_i$* = request vector for process P_i. If ***Request**$_i$* **[*j*]** = ***k then process P_i wants ***k*** instances of resource type R_j

1. If ***Request**$_i$* $\leq$ ***Need**$_i$* go to step 2. Otherwise, raise error condition, since process has exceeded its maximum claim
2. If ***Request**$_i$* $\leq$ ***Available***, go to step 3. Otherwise P_i must wait, since resources are not available
3. Pretend to allocate requested resources to P_i by modifying the state as follows:

$$Available = Available - Request_i;$$

$$Allocation_i = Allocation_i + Request_i;$$

$$Need_i = Need_i - Request_i;$$

- If safe $\Rightarrow$ the resources are allocated to P_i
- If unsafe $\Rightarrow$ P_i must wait, and the old resource-allocation state is restored

Banker's Algorithm for Multiple Resources

1. Look for a row in R (i.e., Need), whose unmet resource needs are all smaller than or equal to *A* (i.e., *Available*). If no such row exists, system will eventually deadlock.
2. Assume the process of row chosen requests all resources needed and finishes. Mark that process as terminated, add its resources to the *A* vector.
3. Repeat steps 1 and 2 until either all processes are marked terminated (safe state) or no process is left whose resource needs can be met (deadlock)

Banker's Algorithm: Example

- 5 processes P_0 through P_4;

  3 resource types:

  A (10 instances), *B* (5 instances), and *C* (7 instances)

- Snapshot at time T_0:

	Allocation	Max	Available
	A B C	A B C	A B C
P_0	0 1 0	7 5 3	3 3 2
P_1	2 0 0	3 2 2	
P_2	3 0 2	9 0 2	
P_3	2 1 1	2 2 2	
P_4	0 0 2	4 3 3	

- The content of the matrix ***Need*** is defined to be ***Max* – *Allocation***

	Need
	A B C
P_0	7 4 3
P_1	1 2 2
P_2	6 0 0
P_3	0 1 1
P_4	4 3 1

- The system is in a safe state since the sequence $< P_1, P_3, P_4, P_2, P_0>$ satisfies safety criteria

Example: P_1 Request (1,0,2)

- Check that Request ≤ Available (that is, $(1,0,2) \le (3,3,2) \Rightarrow$ true

	Allocation	*Need*	*Available*
	A B C	*A B C*	*A B C*
P_0	0 1 0	7 4 3	2 3 0
P_1	3 0 2	0 2 0	
P_2	3 0 2	6 0 0	
P_3	2 1 1	0 1 1	
P_4	0 0 2	4 3 1	

- Executing safety algorithm shows that sequence < P_1, P_3, P_4, P_0, P_2> satisfies safety requirement
- Can request for (3,3,0) by P_4 be granted?
- Can request for (0,2,0) by P_0 be granted?

Questions?

- When to use the Banker's algorithm?
- Data structures?
- Algorithm?