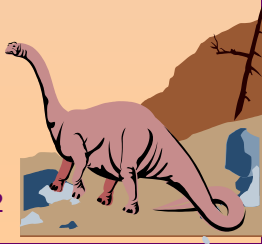
Deadlock Avoidance

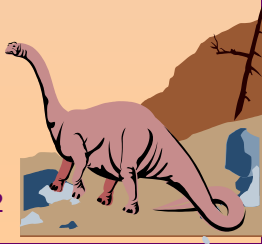
Requires that the system has some additional information about how the resources are to be requested

- Simplest and most useful model requires that each process declare 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 of the processes.



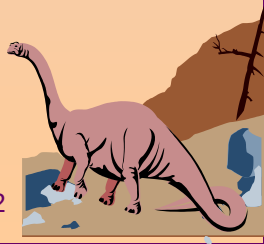
Safe State

- When a process requests an available resource, system must decide if immediate allocation of those resources leaves the system in a *safe state*.
- System is in safe state if there exists a safe sequence of all processes.
- Sequence $<P_1, P_2, \ldots, P_n>$ is safe if for each P_i, the resources that *Pi* can still request can be satisfied by currently available resources + resources held by all the P_j
  - 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.



Basic Facts

- If a system is in safe state ⇒ no deadlocks.
- If a system is in unsafe state ⇒ possibility of deadlock.
- Avoidance ⇒ ensure that a system will never enter an unsafe state.



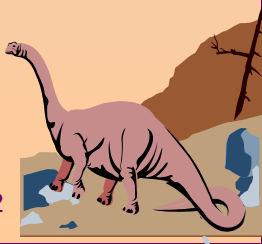
Safe State

- System with 12 tape drives

	Maximum Needs	Current Needs
P0	10	5
P1	4	2
P2	9	2

At time t0: Safe sequence…<P1,P0,P2>

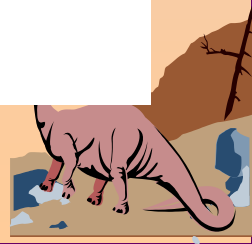
At time t1: Unsafe sequence if 1 more tape drive is allocated to P2



- Example 1: – 12 instances of a resource
- – At time t0, P0 holds 5, P1 holds 2, P2 holds 2
- – Available = 3 free instances

- Example 1:
  - 12 instances of a resource
  - At time t0, P0 holds 5, P1 holds 2, P2 holds 2
  - Available = 3 free instances

processes	max needs	allocated
P0	10	5
P1	4	2
P2	9	2



- Example 1 (cont)
  - Is the system in a safe state? Can I find a safe sequence?
  - Yes, I claim the sequence <P1, P0, P2> is safe.
    - P1 requests its maximum (currently has 2, so needs 2 more) and holds 4, then there is only 1 free resource
    - Then P1 then releases all of its held resources, so that there are 5 free
    - Next, suppose P0 requests its maximum (currently has 5, so needs 5 more) and holds 10, so that there are 0 free
    - Then P0 releases all of its held resources, so that there are 10 free
    - Next P2 requests its max of 9, leaving 3 free and then releases them all
  - Thus the sequence <P1, P0, P2> is safe, and is able in the worst-case to request maximum resources for each process in the sequence, and release all such resources for the next process in the sequence

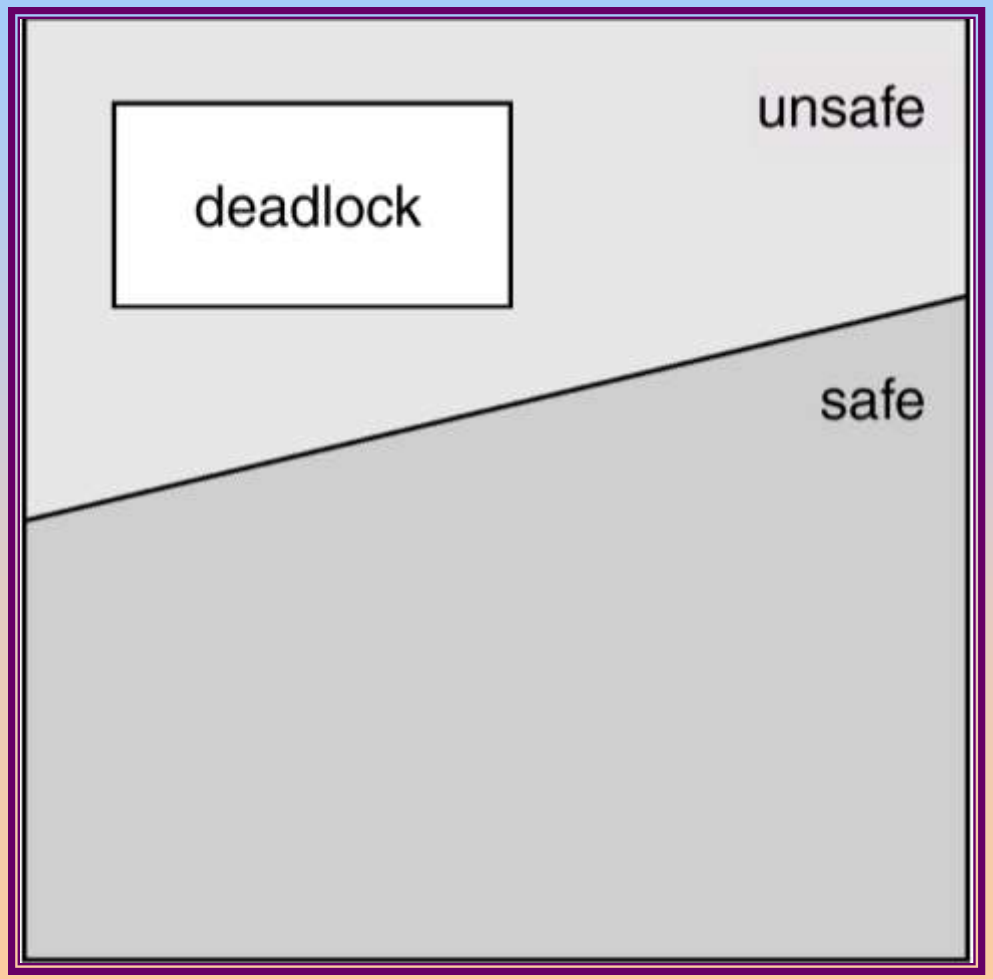
- Example 2:
  - at time t1, process P2 requests and is given 1 more instance of the resource, then

processes	max needs	allocated
P0	10	5
P1	4	2
P2	9	3

  - Available = 2 free instances
  - Is the system in a safe state?

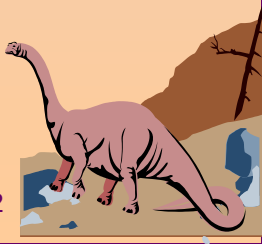
- Example 2 (cont)
  - P1 can request its maximum (currently holds 2, and needs 2 more) of 4, so that there are 0 free
  - P1 releases all its held resources, so that available = 4 free
  - Neither P0 nor P2 can request their maximum resources (P0 needs 5, P2 needs 6, and there are only 4 free)
    - Both would have to wait, so there could be deadlock
  - The system is deemed unsafe
    - The mistake was granting P2 an extra resource at time t1
    - Forcing P2 to wait for P0 or P1 to release their resources would have avoided potential deadlock

Safe, Unsafe , Deadlock State

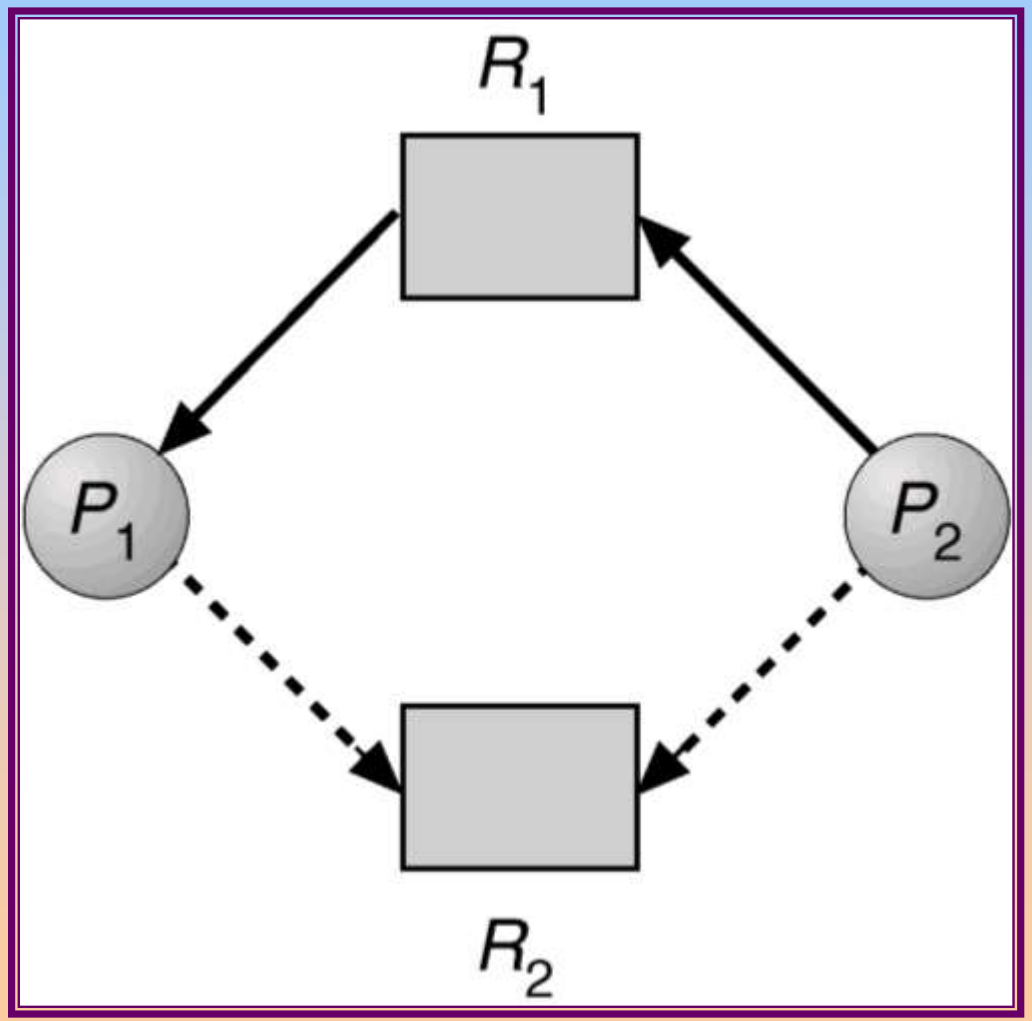


Resource-Allocation Graph Algorithm

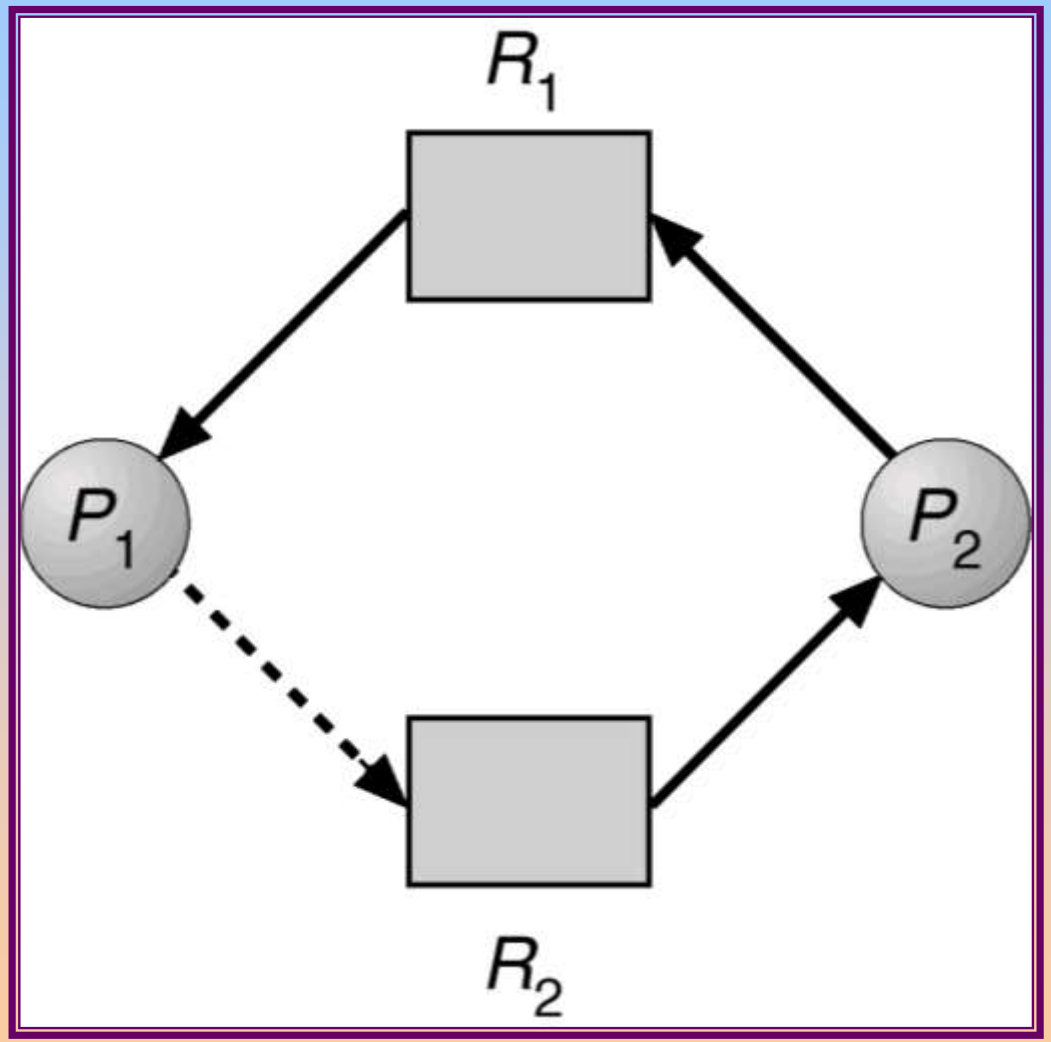
- **SO 2: Demonstrate the resource allocation graph algorithm**
- *Claim edge* $P_i \rightarrow R_j$ indicated that process P_i may request resource R_j; represented by a dashed line.
- Claim edge converts to request edge when a process requests a resource.
- When a resource is released by a process, assignment edge reconverts to a claim edge.
- Resources must be claimed with *a priori information* in the system.



Resource-Allocation Graph For Deadlock Avoidance



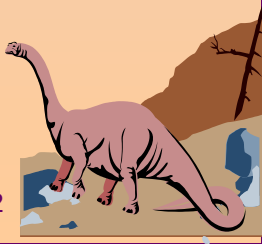
Unsafe State In Resource-Allocation Graph



Banker's Algorithm

- Multiple instances.
- When a process enters into a system it should declare maximum resource needs in all resource types
- This number should not exceed total number of resources in the system.
- When a process requests a set of resources, the system must determine whether the allocation of these resources will leave the system in a safe state.
- If it will, the resources are allocated, otherwise it has to wait until some other process releases enough resources

.

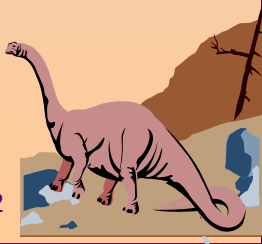


Data Structures for the Banker's Algorithm

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 x m matrix. If *Max* [i,j] = k, then process P_i may request at most k instances of resource type R_j.
- *Allocation:* n x m matrix. If Allocation[i,j] = k then P_i is currently allocated k instances of R_j.
- *Need:* n x 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* =1,2,3, …, *n.*

2. Find an *i* such that both:

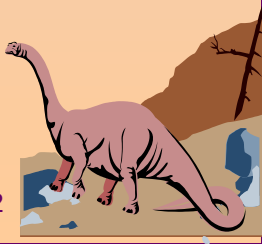
   (a) *Finish* [*i*] = *false*

   (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

Let $Request_i$ be the request vector for process P_i. If $Request_i[j] = k$ then process P_i wants k instances of resource type R_j.

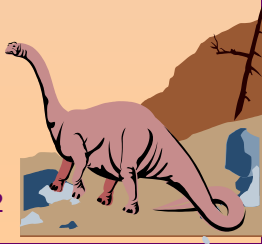
1. If $Request_i \leq Need_i$ go to step 2. Otherwise, raise error condition, since process has exceeded its maximum claim.
2. If $Request_i \leq Available$, go to step 3. Otherwise P_i must wait, since resources are not available.
3. Pretend to allocate requested resources to P_i by modifying the state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

$$Available = Available - Request_i;$$

$$Allocation_i = Allocation_i + Request_i;$$

$$Need_i = Need_i - Request_{i;;}$$

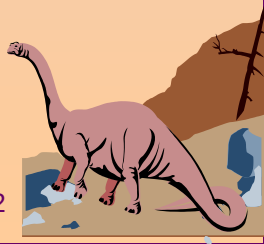
- *If safe* $\Rightarrow$ *the resources are allocated to* P_i.
- *If unsafe* $\Rightarrow$ P_i *must wait, and the old resource-allocation state is restored*



Example of Banker's Algorithm

- 5 processes P_0 through P_4; 3 resource types *A* (10 instances),
  B (5instances, 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	

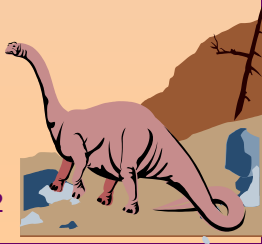


Example (Cont.)

- 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.



Solution: Banker's Algorithm

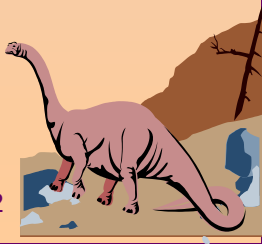
Step 1:

- Safety for process P_0
- $need_0 = (7, 4, 3)$
- If $need_0 \leq$ Available
- if $[(7, 4, 3) \leq (3,3,2)]$ (false)
- Process P_0 must wait.

Step 2:

- Safety for process P_1
- $need_1 = (1, 2, 2)$
- if $need_i \leq$ Available
- if $[(1, 2, 2) \leq (3,3,2)]$
- P_i will execute.
- Available = Available + Allocation
- = (3, 3, 2) + (2, 0, 0)
- = (5, 3, 2)

	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

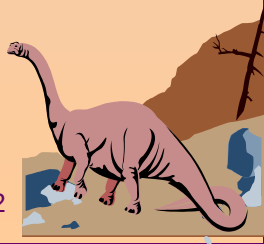


Step 3:

- Safety for process P_2
- $need_2 = (6, 0, 0)$
- if $need_2 \leq Available$
- if $[(6, 0, 0) \leq (5, 3, 2)]$ (false)
- P_3 will execute.
- Available = Available + Allocation
- = (5, 3, 2) + (2, 1, 1)= (7, 4, 3)

Step 5:

- Safety for process P_4
- $need_4 = (4, 3, 1)$
- If $need_4 \leq Available$
- If $[(4, 3, 1) \leq (6, 4, 3)]$
- P_4 will execute.
- Available = Available + Allocation
- = (7, 4, 3) + (0, 0, 2)
- = (7, 4, 5)



Step 6:

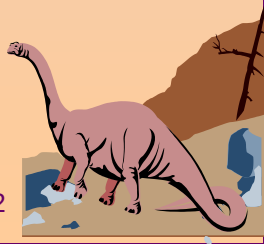
- Safety for process P_0

  $need_0 = (7, 4, 3)$
- if $need_0 \leq Available$
- if $[(7, 4, 3) \leq (7,4,5)]$
- P_0 will execute.
- Available = Available + Allocation
- = (7, 4, 5) + (0, 1, 0)= (7, 5, 5)

Step 7:

- Safety for process P_2
- $need_2 = (6, 0, 0)$
- if $need_2 \leq Available$
- if $[(6, 0, 0) \leq (7,5,5)]$
- P_2 will execute.
- Available = Available + Allocation
- = (7, 5, 5) + (3, 0, 2)= (10, 5, 7)

Safety Sequence = <P1, P3, P4, P0, P2>

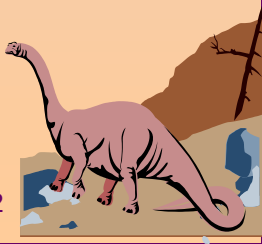


Example P_1 Request (1,0,2) (Cont.)

- Check that Request ≤ Available (that is, (1,0,2) ≤ (3,3,2) ⇒ *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 1	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.



Problems

Consider the following snapshot of a system.

	Allocation	Max	Available
	A B C D	A B C D	A B C D
P0	0 0 1 2	0 0 1 2	1 5 2 0
P1	1 0 0 0	1 7 5 0	
P2	1 3 5 4	2 3 5 6	
P3	0 6 3 2	0 6 5 2	
P4	0 0 1 4	0 6 5 6	

Answer the following questions using the Banker's algorithm:

a. What is the content of the matrix *Need?*

b. Is the system in a safe state?

c. If a request from process P1 arrives for (0,4,2,0), can the request be granted immediately?