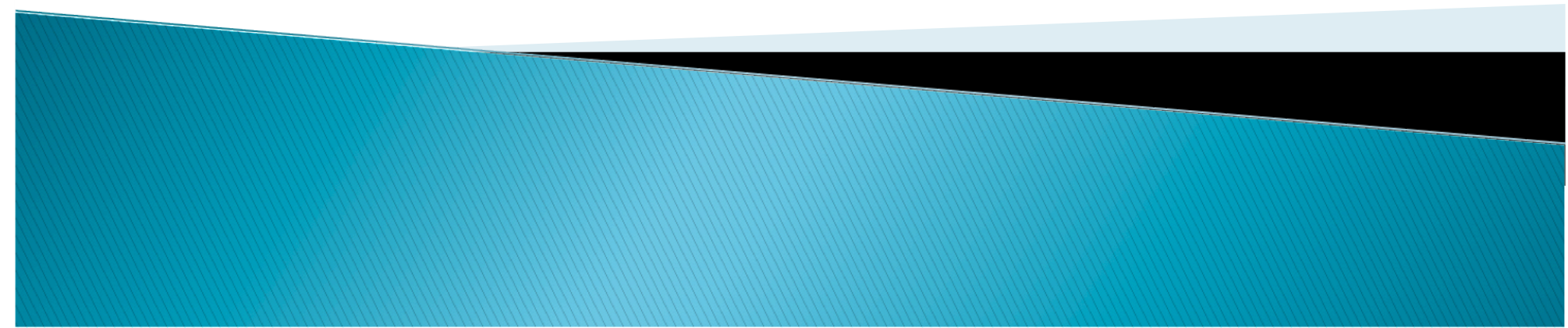


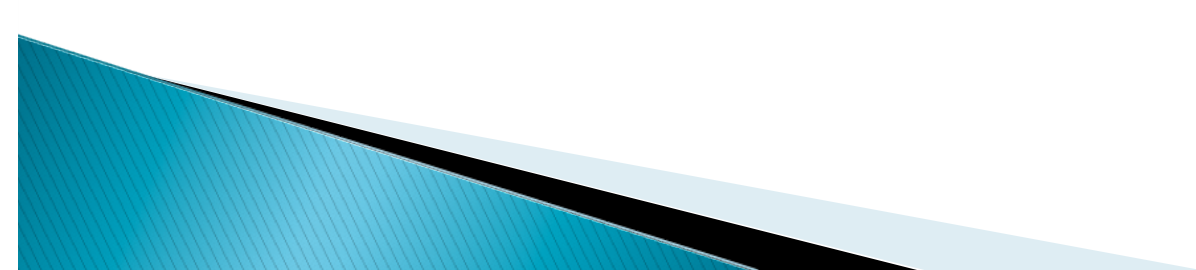
# Chapter 7: Deadlocks

---

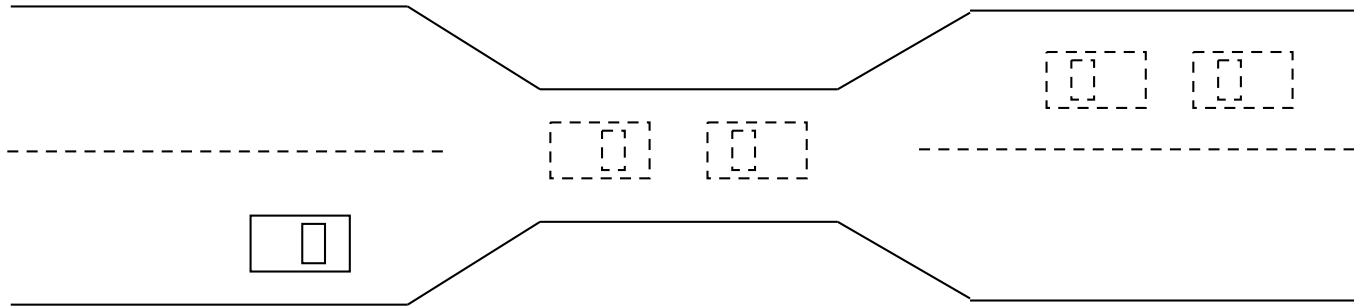


# The Deadlock Problem

- A set of blocked processes each holding a resource and waiting to acquire a resource held by another process in the set.
- Example
  - Consider a system with three CD RW drives.
  - Suppose each of three processes holds one of these CD RW drives.
  - If each process now requests another drive, the three processes will be in a deadlock state.
  - Each is waiting for the event "CD RW is released," which can be caused only by one of the other waiting processes.



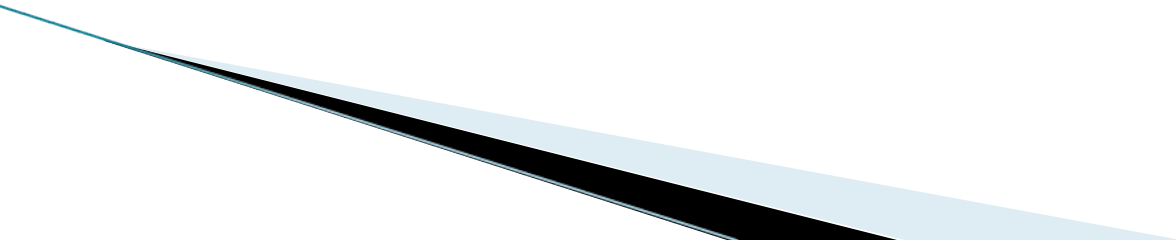
# Bridge Crossing Example



- ❑ Traffic only in one direction.
- ❑ The bridge can be viewed as a resource and the cars as processes.
- ❑ If a deadlock occurs, it can be resolved if one car backs up (preempt resources and rollback).
- ❑ Several cars may have to be backed up if a deadlock occurs.
- ❑ Starvation is possible.

# Necessary conditions for deadlock

A deadlock situation can arise if the following four conditions hold simultaneously in a system:

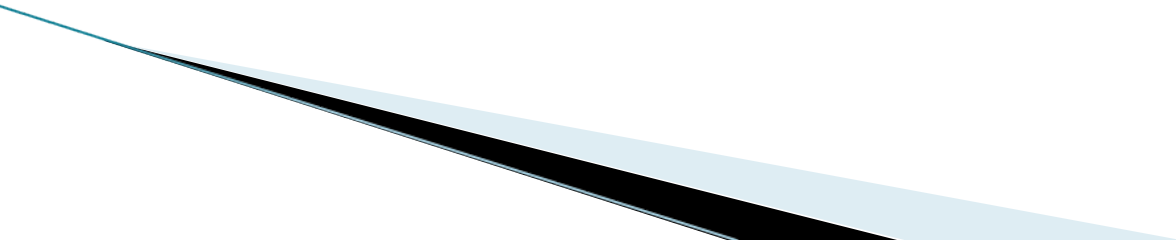
- 1. Mutual exclusion.** At least one resource must be held in a non-sharable mode; that is, only one process at a time can use the resource. If another process requests that resource, the requesting process must be delayed until the resource has been released.
  - 2. Hold and wait.** A process must be holding at least one resource and waiting to acquire additional resources that are currently being held by other processes.
  - 3. No preemption.** Resources cannot be preempted.; that is, a resource can be released only voluntarily by the process holding it, after that process has completed its task.
- 

## Necessary conditions for deadlock(cntd)

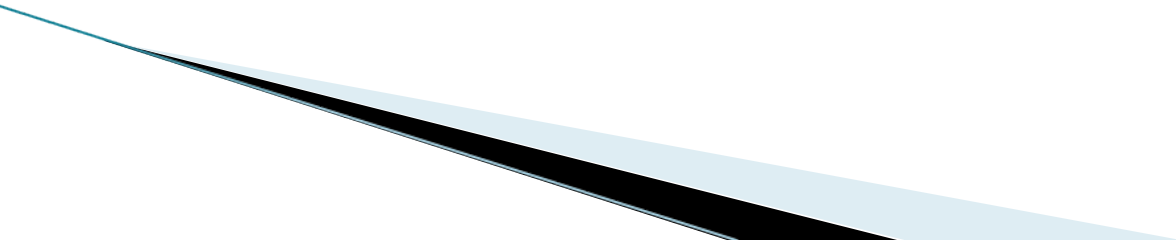
4. **Circular wait.** A set  $\{P_i, \dots, P_n\}$  of waiting processes must exist such that  $P_0$  is waiting for a resource held by  $P_1$ ,  $P_1$  is waiting for a resource held by  $P_2$ , ...,  $P_{n-1}$  is waiting for a resource held by  $P_n$ , and  $P_n$  is waiting for a resource held by  $P_0$ .

We emphasize that all four conditions must hold for a deadlock to occur. The circular-wait condition implies the hold-and-wait condition, so the four conditions are not completely independent. However, that it is useful to consider each condition separately.

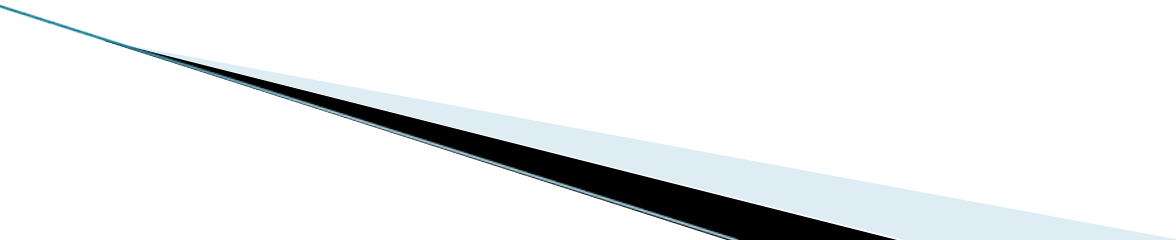
# Banker's Algorithm

- Banker's algorithm is a **resource allocation** and **deadlock avoidance** algorithm developed for the safe allocation of predetermined maximum possible amounts of all resources.
  - The Banker's algorithm is run by the operating system whenever a process requests resources.
  - The algorithm avoids deadlock by denying or postponing the request if it determines that accepting the request could put the system in an unsafe state (one where deadlock could occur).
- 

# Working

- For the Banker's algorithm to work, it needs to know three things:
    1. How much of each resource each process could possibly request [Max]
    2. How much of each resource each process is currently holding [ALLOCATED]
    3. How much of each resource the system currently has available [AVAILABLE]
  - Resources may be allocated to a process only if it satisfies the following conditions:
    - $\text{request} \leq \text{max}$ , else set error condition as process has crossed maximum claim made by it.
    - $\text{request} \leq \text{available}$ , else process waits until resources are available.
- 

# Safe and Unsafe States

- A state is considered safe if it is possible for all processes to finish executing.
  - Since the system cannot know when a process will terminate, or how many resources it will have requested by then, the system assumes that all processes will eventually attempt to acquire their stated maximum resources and terminate soon afterward.
  - The algorithm determines if a state is safe by trying to find a hypothetical set of requests by the processes that would allow each to acquire its maximum resources and then terminate (returning its resources to the system). Any state where no such set exists is an unsafe state.
- 



# 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  $i = 1, 3, \dots, n$ .
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.

# Example of Banker's Algorithm

- 5 processes  $P_0$  through  $P_4$ ; 3 resource types  $A$  (10 instances),  $B$  (5 instances, and  $C$  (7 instances).
- Snapshot at time  $T_0$ :

	<u>Allocation</u>			<u>Max</u>			<u>Available</u>		
	$A$	$B$	$C$	$A$	$B$	$C$	$A$	$B$	$C$
$P_0$	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.

<u>Need</u>			
	<i>A</i>	<i>B</i>	<i>C</i>
<i>P0</i>	7	4	3
<i>P1</i>	1	2	2
<i>P2</i>	6	0	0
<i>P3</i>	0	1	1
<i>P4</i>	4	3	1

- The system is in a safe state since the sequence  $\langle P1, P3, P4, P2, P0 \rangle$  satisfies safety criteria. HOW?

- **For P1**

- Available resources – P1(Need)

- $\langle 3, 3, 2 \rangle - \langle 1, 2, 2 \rangle = \langle 2, 1, 0 \rangle$

- So P1's request is satisfied and after using the resources it returns the resources to system.

- Available =  $\langle 2, 1, 0 \rangle + P1(\text{Max}) = \langle 2, 1, 0 \rangle + \langle 3, 2, 2 \rangle = \langle 5, 3, 2 \rangle$

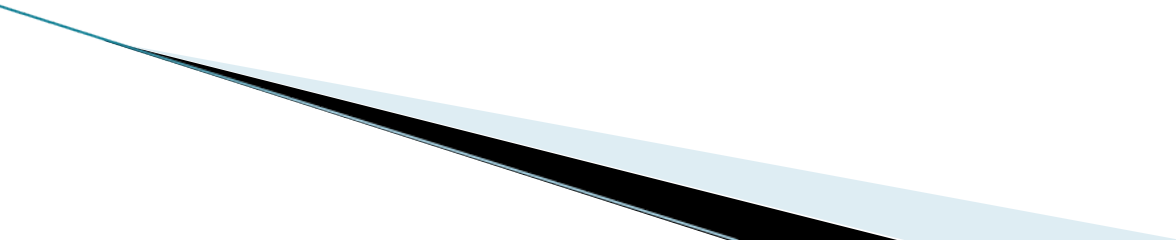
- **For P3**

- Available resources – P3(Need)

- $\langle 5, 3, 2 \rangle - \langle 0, 1, 1 \rangle = \langle 5, 2, 1 \rangle$

- So P3's request is satisfied and after using the resources it returns the resources to system.

- Available =  $\langle 5, 2, 1 \rangle + P3(\text{Max}) = \langle 5, 2, 1 \rangle + \langle 2, 2, 2 \rangle = \langle 7, 4, 3 \rangle$



- **For P4**

- Available resources – P4(Need)

- $\langle 7, 4, 3 \rangle - \langle 4, 3, 1 \rangle = \langle 3, 1, 2 \rangle$

- So P4's request is satisfied and after using the resources it returns the resources to system.

- Available =  $\langle 3, 1, 2 \rangle + P4(\text{Max}) = \langle 3, 1, 2 \rangle + \langle 4, 3, 3 \rangle = \langle 7, 4, 5 \rangle$

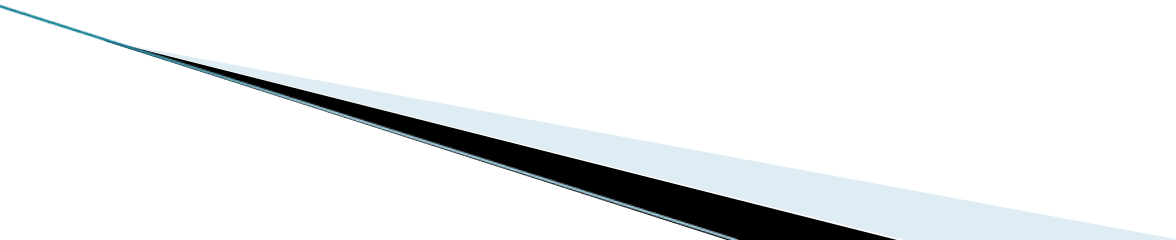
- **For P2**

- Available resources – P2(Need)

- $\langle 7, 4, 5 \rangle - \langle 6, 0, 0 \rangle = \langle 1, 4, 5 \rangle$

- So P2's request is satisfied and after using the resources it returns the resources to system.

- Available =  $\langle 1, 4, 5 \rangle + P2(\text{Max}) = \langle 1, 4, 5 \rangle + \langle 9, 0, 2 \rangle = \langle 10, 4, 7 \rangle$



- **For P0**

- Available resources – P0(Need)
- $\langle 10, 4, 7 \rangle - \langle 7, 4, 3 \rangle = \langle 3, 0, 4 \rangle$

- So P0's request is satisfied and after using the resources it returns the resources to system.
- Available =  $\langle 3, 0, 4 \rangle + P0(\text{Max}) = \langle 3, 0, 4 \rangle + \langle 7, 5, 3 \rangle = \langle 10, 5, 7 \rangle$

- Suppose now that process P1 requests one additional instance of resource type A and two instances of resource type C, so Request =  $\langle 1, 0, 2 \rangle$ .
- First we check that Request  $<$  Available i-e  $\langle 1, 0, 2 \rangle < \langle 3, 3, 2 \rangle$ , which is true.
- We then pretend that this request has been fulfilled, and we arrive at the following new state:

	<u>Allocation</u>			<u>Need</u>			<u>Available</u>		
	A	B	C	A	B	C	A	B	C
P0	0	1	0	7	4	3	2	3	0
P1	3	0	2	0	2	0			
P2	3	0	1	6	0	0			
P3	2	1	1	0	1	1			
P4	0	0	2	4	3	1			



- We must determine whether this new system state is safe.
  - To do so, we execute our safety algorithm and find that the sequence  $\langle P1, P3, P4, P0, P2 \rangle$  satisfies the safety requirement. Hence, we can immediately grant the request of process P1.
  - However when the system is in this state, a request for  $(3,3,0)$  by P4 cannot be granted, since the resources are not available.
  - Furthermore, a request for  $(0,2,0)$  by P0 cannot be granted, even though the resources are available, since the resulting state is unsafe.
- 