Concurrency: Deadlock and Starvation

Deadlock

- Permanent blocking of a set of processes that either compete for system resources or communicate with each other
- No efficient solution
- Involve conflicting needs for resources by two or more processes

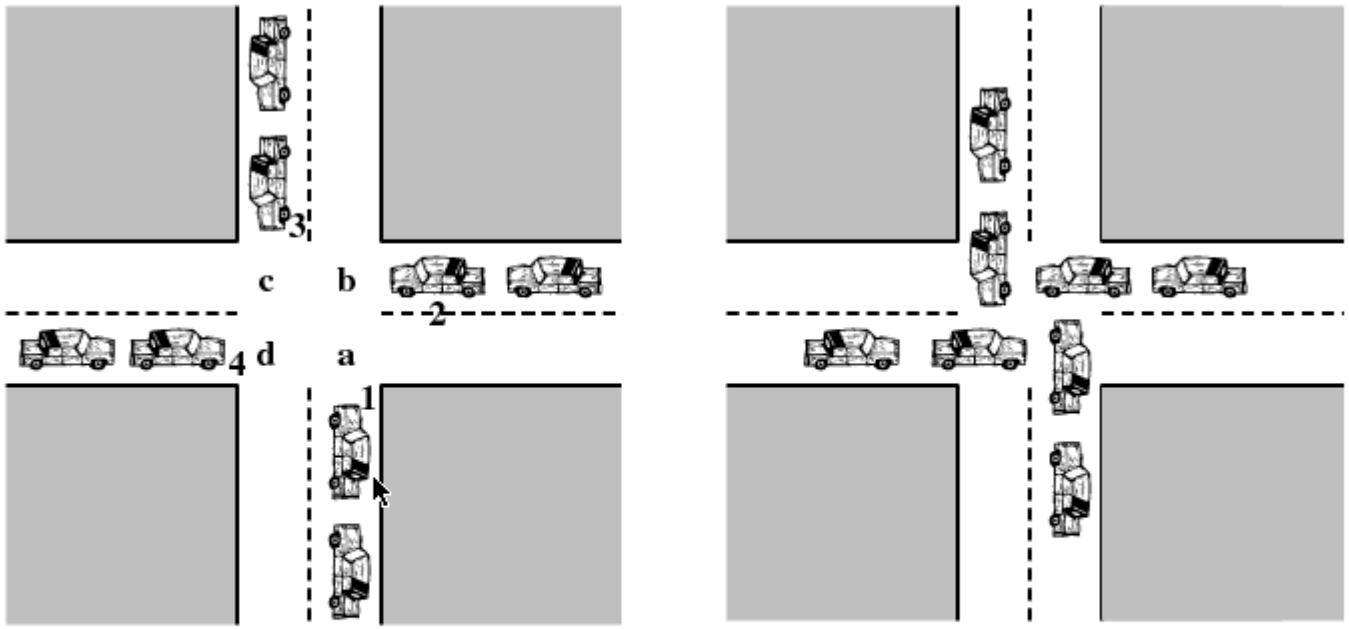
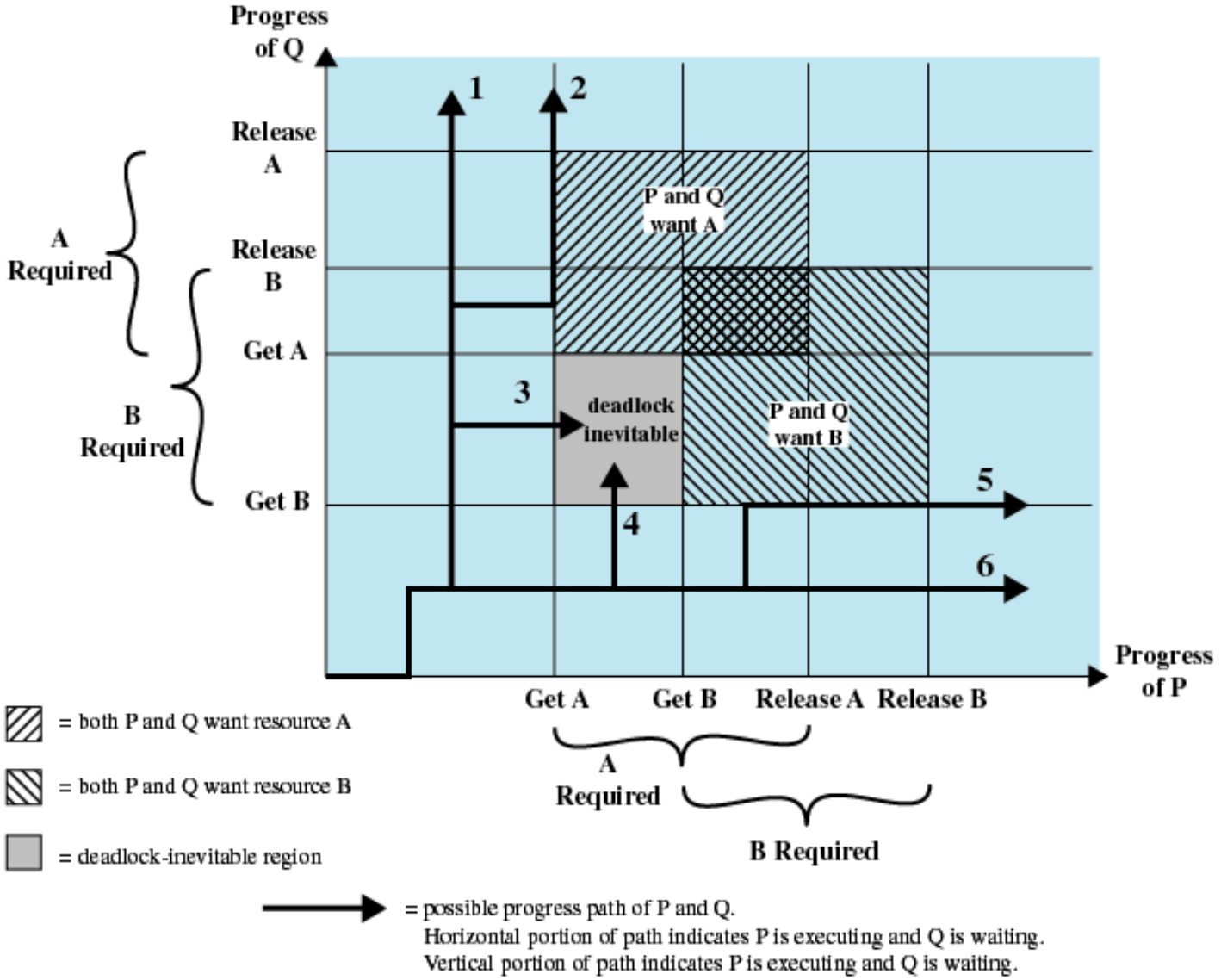


Figure 6.1 Illustration of Deadlock

Deadlock & Fatal Region



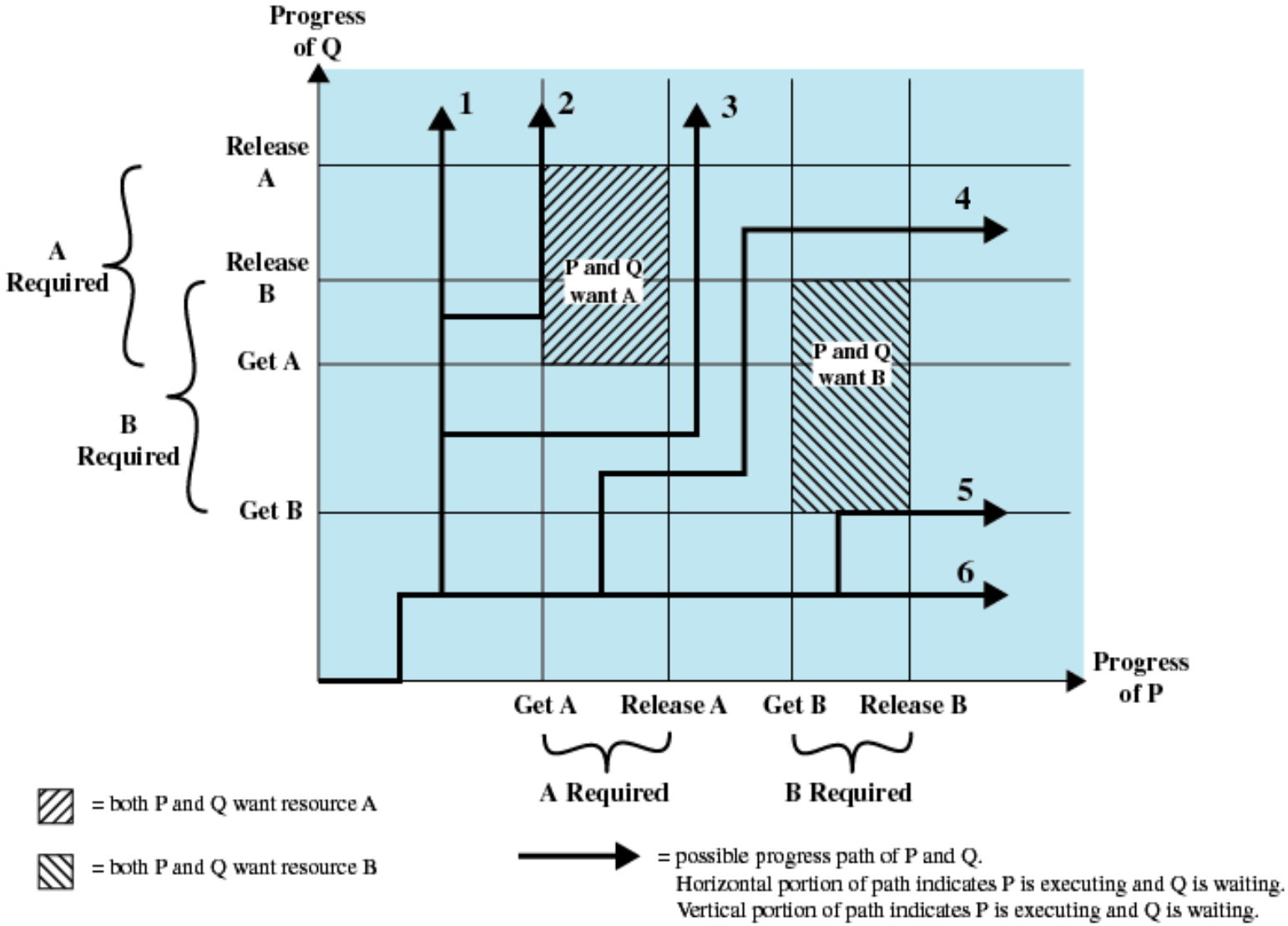


Figure 6.3 Example of No Deadlock [BACO03]

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
- 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

Example of Deadlock

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)

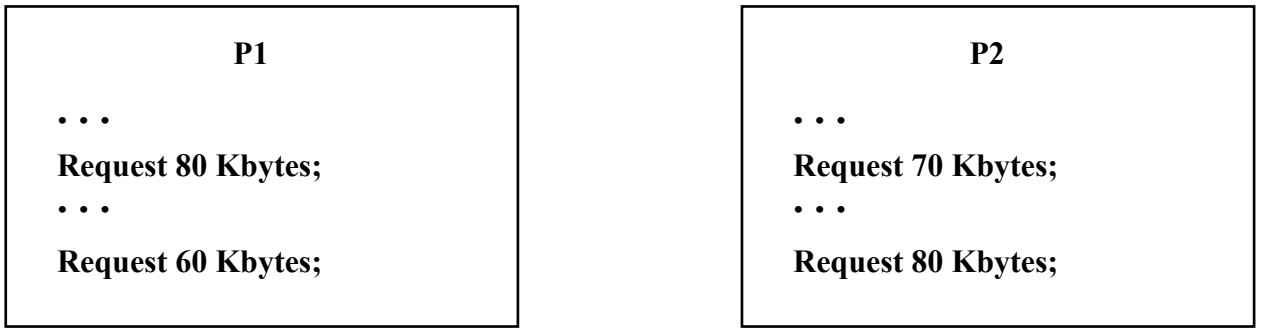
Process Q

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)

Figure 6.4 Example of Two Processes Competing for Reusable Resources

Another Example of Deadlock

- Space is available for allocation of 200Kbytes, and the following sequence of events occur



- Deadlock occurs if both processes progress to their second request

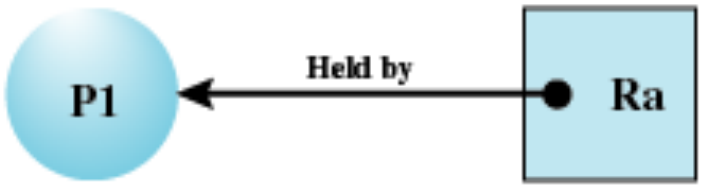
Consumable Resources

- Created (produced) and destroyed (consumed)
- Interrupts, signals, messages, and information in I/O buffers
- Deadlock may occur if a Receive message is blocking; e.g.:
  - P1: receive(P2, M); P2:receive(P1, M); P1&P2:send
- May take a rare combination of events to cause deadlock

Resource Allocation Graphs

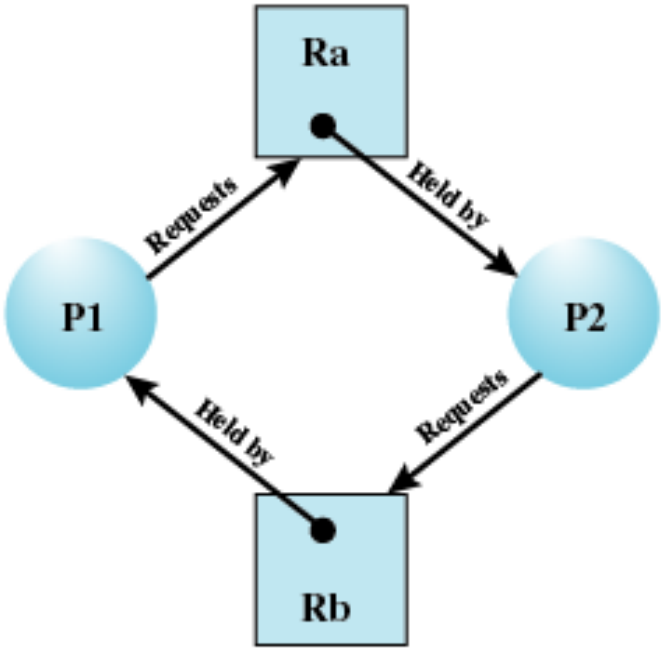
- Directed graph that depicts a state of the system of resources and processes

(a) Resouce is requested

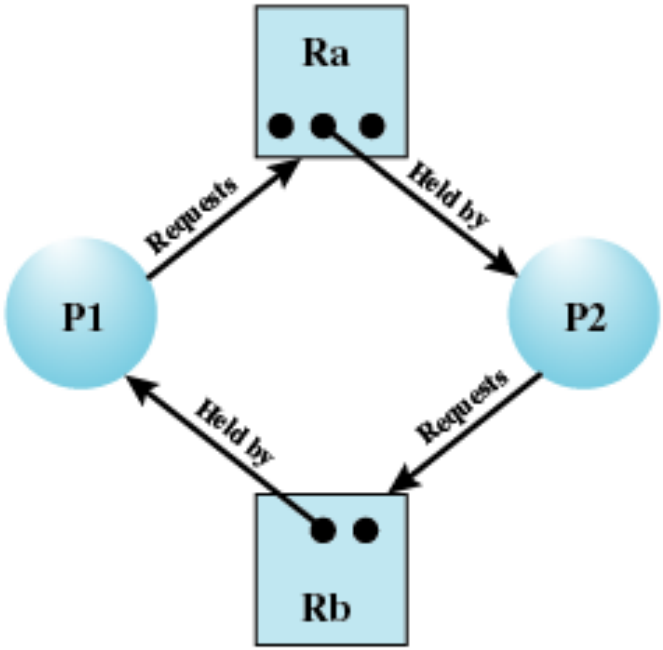


(b) Resource is held

Resource Allocation Graphs



(c) Circular wait



(d) No deadlock

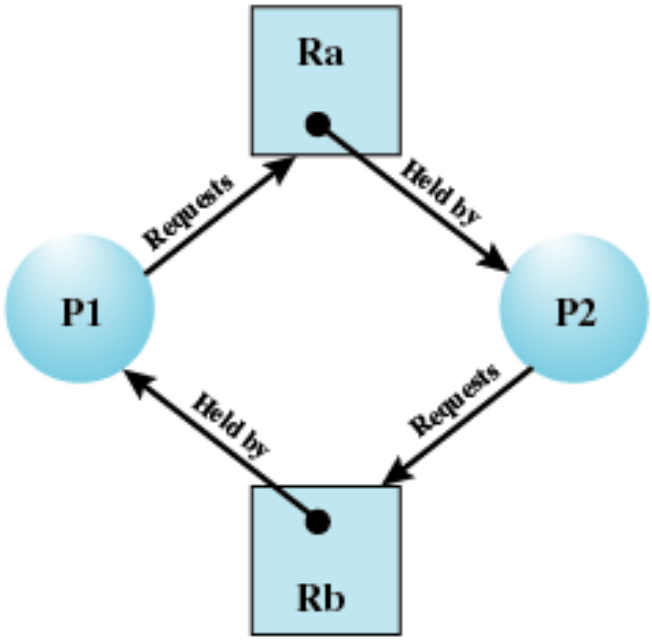
Figure 6.5 Examples of Resource Allocation Graphs

Conditions for Deadlock

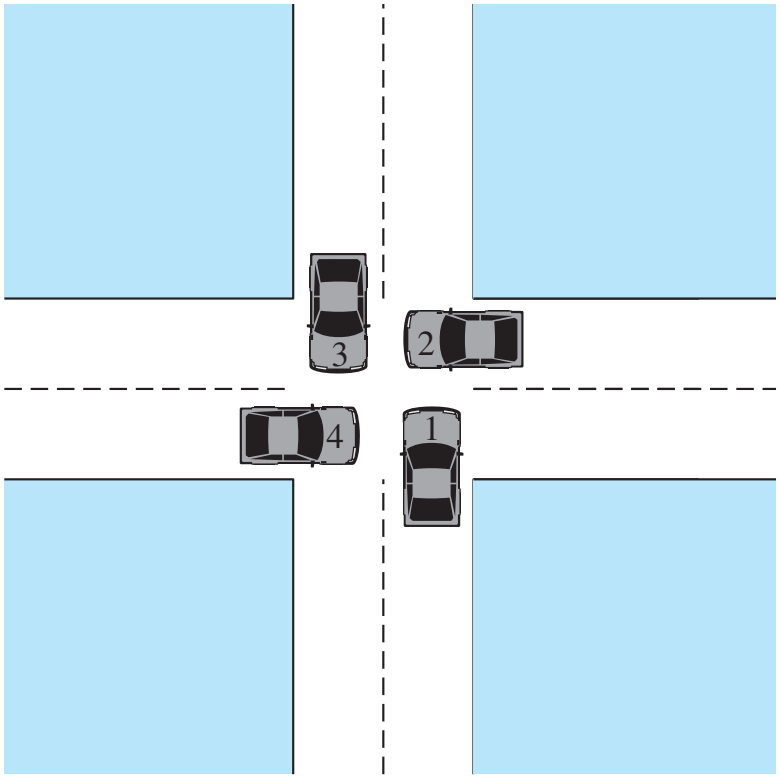
- Mutual exclusion
  - Only one process may use a resource at a time
- Hold-and-wait
  - A process may hold allocated resources while awaiting assignment of others
- No preemption
  - No resource can be forcibly removed form a process holding it

Conditions for Deadlock

- 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



(c) Circular wait



(b) Deadlock

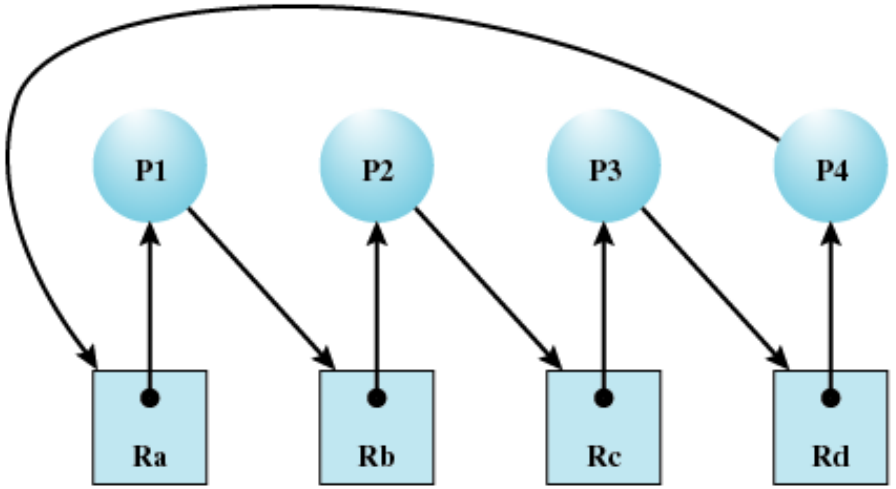


Figure 6.6 Resource Allocation Graph for Figure 6.1b

Possibility of Deadlock (necessary condition)

- Mutual Exclusion
- No preemption
- Hold and wait

Existence of Deadlock (sufficient condition)

- Mutual Exclusion
- No preemption
- Hold and wait
- Circular wait (fatal region has been reached)

What to do:

- Prevention: make sure that one of the 4 sufficient conditions is not met
- Avoid: make dynamic choices based on system and resource allocation state
- Detect: mechanisms to understand that it has happened

Deadlock Prevention

- Mutual Exclusion
  - Must be supported by the operating system
- Hold and Wait: Require a process request all resources at one time
  - Not efficient & difficult (cannot know in advance)
- No Preemption: Process must release resource and request again (priorities)
  - Operating system may preempt a process to require it releases its resources (handle same priorities)
- Circular Wait: Define a linear ordering of resource types
  - Enforce requests only in one direction of the ordering (either increasing or decreasing order)

Deadlock Avoidance

- A decision is made dynamically whether the current resource allocation request will, if granted, potentially lead to a deadlock
- Requires knowledge of future process request

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

Process Initiation denial

[Resource = $\mathbf{R} = (R_1, R_2, \ldots, R_m)$	total amount of each resource in the system
Available = $\mathbf{V} = (V_1, V_2, \ldots, V_m)$	total amount of each resource not allocated to any process
Claim = $\mathbf{C} = \begin{bmatrix} C_{11} & C_{12} & \cdots & C_{1m} \\ C_{21} & C_{22} & \cdots & C_{2m} \\ \vdots & \vdots & \vdots & \vdots \\ C_{n1} & C_{n2} & \cdots & C_{nm} \end{bmatrix}$	C_{ij} = requirement of process i for resource j
Allocation = $\mathbf{A} = \begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1m} \\ A_{21} & A_{22} & \cdots & A_{2m} \\ \vdots & \vdots & \vdots & \vdots \\ A_{n1} & A_{n2} & \cdots & A_{nm} \end{bmatrix}$	A_{ij} = current allocation to process i of resource j

1. $R_j = V_j + \sum_{i=1}^{n} A_{ij}$, for all j — All resources are either available or allocated.
2. $C_{ij} \leq R_j$, for all i, j — No process can claim more than the total amount of resources in the system.
3. $A_{ij} \leq C_{ij}$, for all i, j — No process is allocated more resources of any type than the process originally claimed to need.

- Start a new process P(n+1) only iff:
  - Rj >= C(n+1)j + Sum(i=1->n) Cij for all j.