Locking protocol to enforce order

- Shared: Transaction can read but not write
- Exclusive: Transaction can read and write

- Two phase protocol to ensure serializability:
  - Growing phase - transaction can obtain but not release locks
  - Shrinking phase - transaction can release lock but not acquire new ones

  - Ensures conflict serializability but is not free from deadlocks

Timestamp-based Protocols

- Timestamp transactions: Can be real wall clock time or logical clock
- The timestamp determines the serializability order
- For each data item (Q), associate two timestamps
  - W-timestamp denotes largest timestamp of any transaction that successfully executed write(Q).
  - R-timestamp for read(Q)
- Suppose Ti issues read(Q):
  - If TS(Ti) < W-timestamp(Q), rollback Ti
  - If TS(Ti) >= W-timestamp(Q), execute Ti, R-timestamp = maximum (R-timestamp(Q) and TS(Ti))
- Similarly for Ti issuing write(Q):

Summary

- So far, we covered primitives for process synchronization
- Next, we investigate deadlocks

Chapter 7: Deadlocks

- 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.

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
  - System has 2 tape drives.
  - P_1 and P_2 each hold one tape drive and each needs another one.
- Example
  - semaphores *A* and *B*, initialized to 1

P_0	P_1
wait (A);	*wait(B)*
wait (B);	*wait(A)*

System Model

- Resource types R_1, R_2, . . ., R_m

  CPU cycles, memory space, I/O devices
- Each resource type R_i has W_i instances.
- Each process utilizes a resource as follows:
  - request
  - use
  - release

Deadlock Characterization

Deadlock can arise if four conditions hold simultaneously.

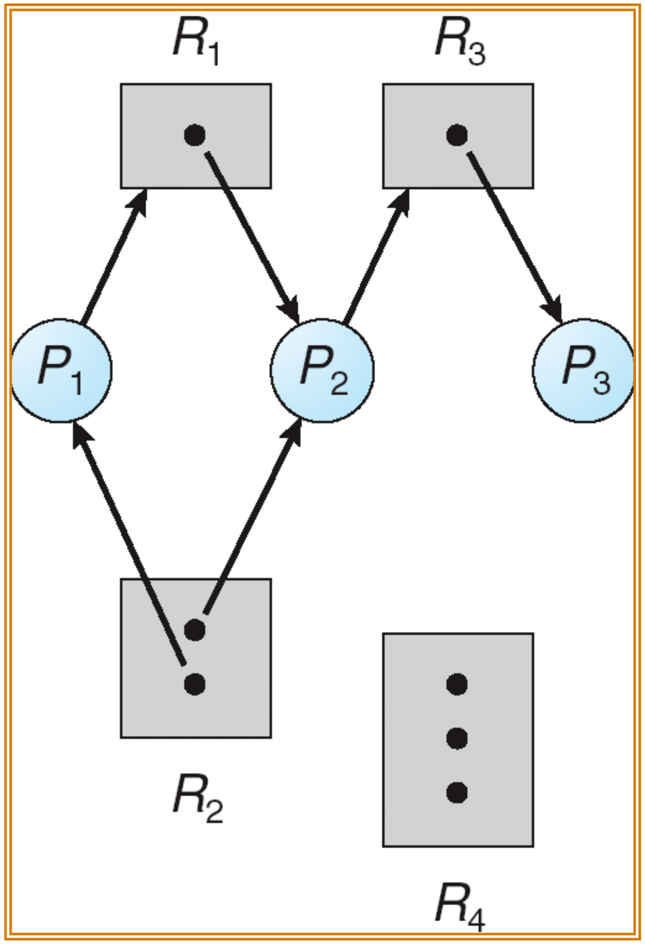
- **Mutual exclusion:** only one process at a time can use a resource.
- **Hold and wait:** a process holding at least one resource is waiting to acquire additional resources held by other processes.
- **No preemption:** a resource can be released only voluntarily by the process holding it, after that process has completed its task.
- **Circular wait:** there exists a set $\{P_0, P_1, \ldots, P_0\}$ of waiting processes such that P_0 is waiting for a resource that is held by P_1, P_1 is waiting for a resource that is held by P_2, …, P_{n-1} is waiting for a resource that is held by P_n, and P_0 is waiting for a resource that is held by P_0.

Resource-Allocation Graph

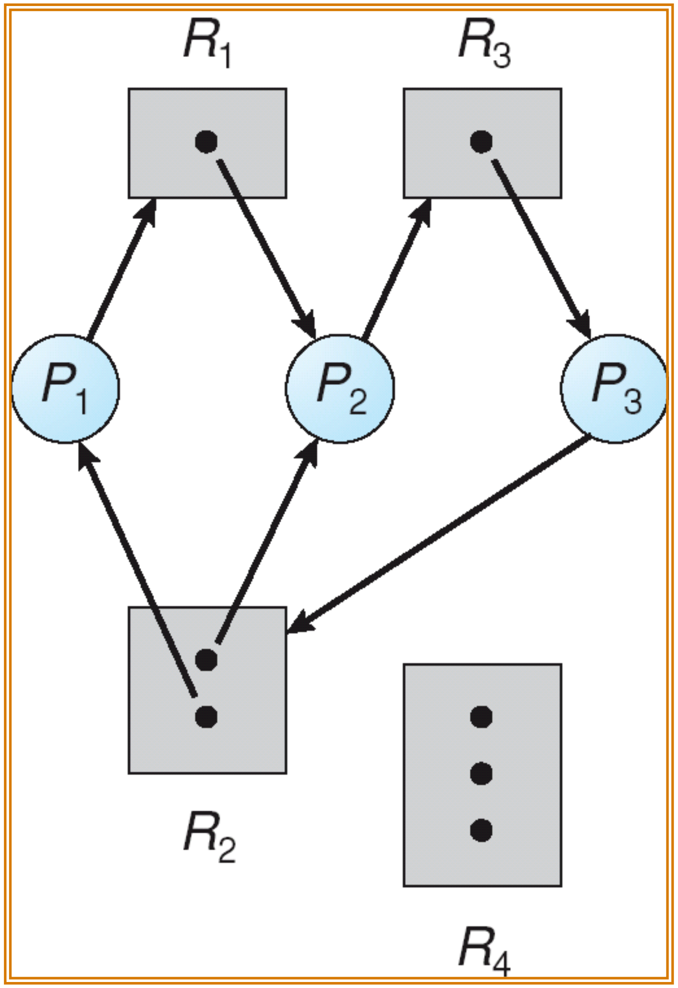
A set of vertices *V* and a set of edges *E*.

- V is partitioned into two types:
  - $P = \{P_1, P_2, \ldots, P_n\}$, the set consisting of all the processes in the system.
  - $R = \{R_1, R_2, \ldots, R_m\}$, the set consisting of all resource types in the system.
- request edge – directed edge $P_1 \rightarrow R_j$
- assignment edge – directed edge $R_j \rightarrow P_i$

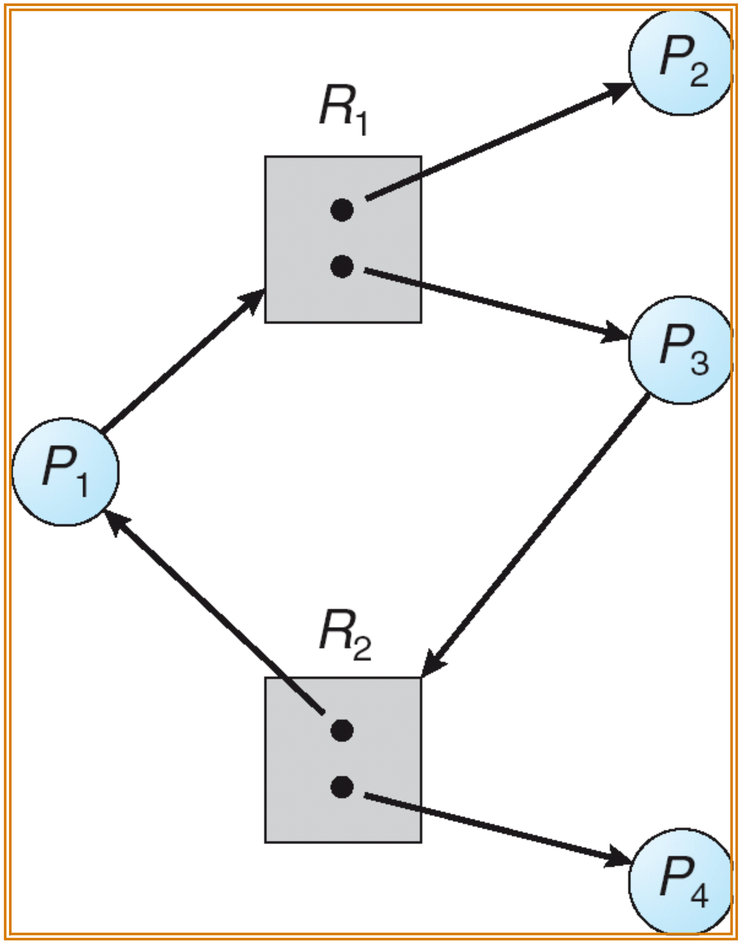
Example of a Resource Allocation Graph



Resource Allocation Graph With A Deadlock



Resource Allocation Graph With A Cycle But No Deadlock



Basic Facts

- If graph contains no cycles ⇒ no deadlock.
- If graph contains a cycle ⇒
  - if only one instance per resource type, then deadlock.
  - if several instances per resource type, possibility of deadlock.

Methods for Handling Deadlocks

- Ensure that the system will *never* enter a deadlock state.
- Allow the system to enter a deadlock state and then recover.
- Ignore the problem and pretend that deadlocks never occur in the system; used by most operating systems, including UNIX.

Deadlock Prevention

Restrain the ways request can be made.

- Mutual Exclusion – not required for sharable resources; must hold for nonsharable resources.
- Hold and Wait – must guarantee that whenever a process requests a resource, it does not hold any other resources.
  - Require process to request and be allocated all its resources before it begins execution, or allow process to request resources only when the process has none.
  - Low resource utilization; starvation possible.

Deadlock Prevention (Cont.)

- **No Preemption** –
  - If a process that is holding some resources requests another resource that cannot be immediately allocated to it, then all resources currently being held are released
  - Preempted resources are added to the list of resources for which the process is waiting
  - Process will be restarted only when it can regain its old resources, as well as the new ones that it is requesting
- **Circular Wait** – impose a total ordering of all resource types, and require that each process requests resources in an increasing order of enumeration

Deadlock Avoidance

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
- 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 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 P_i can still request can be satisfied by currently available resources + resources held by all the P_j, with j<I
  - 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, Unsafe , Deadlock State