Deadlocks

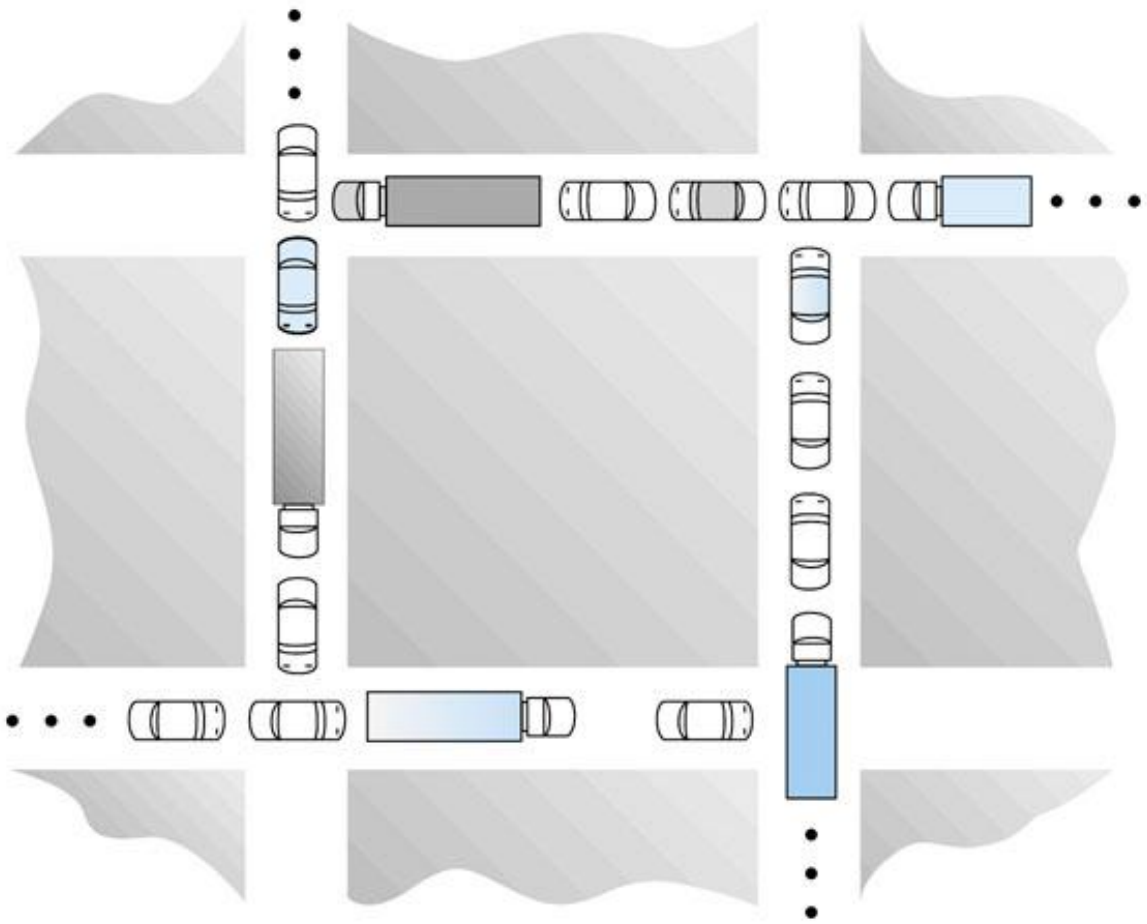
Jin-Soo Kim (jinsookim@skku.edu)
Computer Systems Laboratory
Sungkyunkwan University
http://csl.skku.edu

Today's Topics

- **What is the deadlock problem?**
- **Four conditions for deadlock**
- **Handling deadlock**
  - Prevention
  - Avoidance
  - Detection & Recovery
  - Ignorance

Deadlock (1)

- **Traffic deadlock**



Deadlock (2)

- **Example 1**
  - System has 2 disk drives.
  - P1 and P2 each hold one disk drive and each needs another one.

- **Example 2**
  - Semaphores A and B, initialized to 1.

P1	P2
wait (A);	wait (B);
wait (B);	wait (A);

Deadlock (3)

- **Deadlock problem**
  - A set of blocked processes each holding a resource and waiting to acquire a resource held by another process in the set.
  - Processes need access to resources in reasonable order.
    - Processes sometimes require more than one resource.
    - Order in which the resources are allocated is important.
  - Deadlocks can occur across machines.
    - Shared devices between a number of machines
    - Printers, scanners, CD recorders, etc.

Deadlock (4)

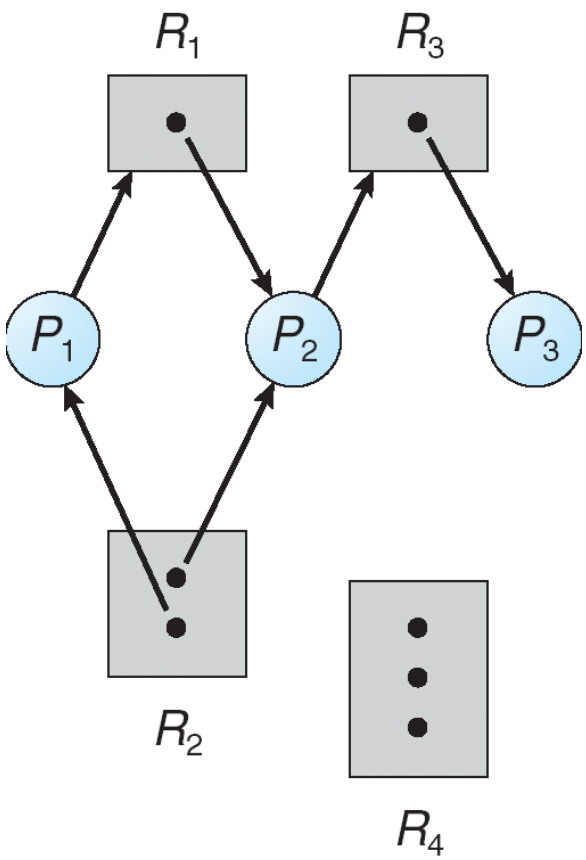
- **Resources**
  - Can only be used by one process at a time.
  - Can be both hardware and software.
    - e.g., tape drives, printers, system tables, database entries, memory locations
  - Request → Allocate → Use → Release
- **Preemptable resources**
  - Can be taken away from a process with no ill effects.
- **Nonpreemptable resources**
  - Will cause the process to fail if taken away.
  - Generally deadlocks involve nonpreemtable resources.

Conditions for Deadlock

- **Four necessary conditions**
  - **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 must exist a set {P0,P1,…,Pn,P0} of waiting processes such that P0 is waiting for a resource that is held by P1, P1 is waiting for a resource held by P2, etc.

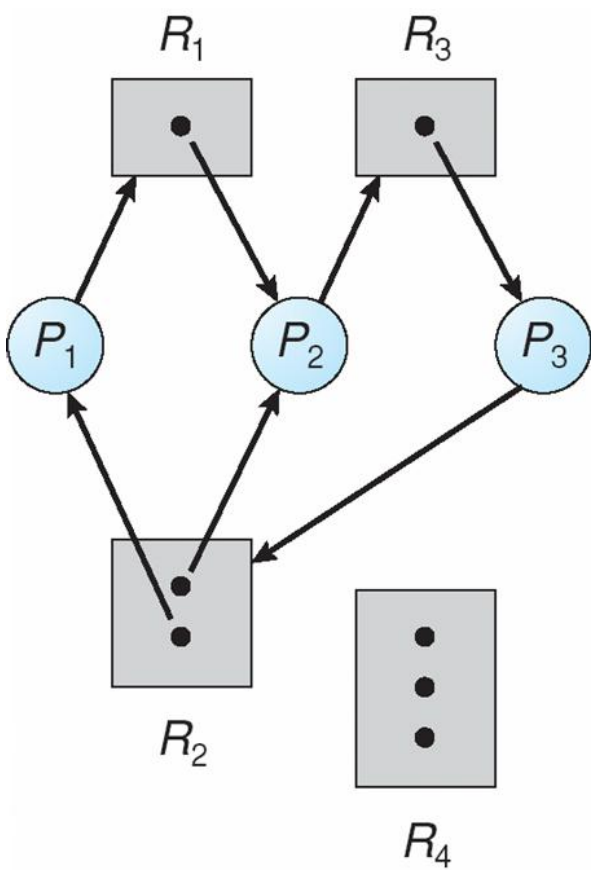
Deadlock Modeling (1)

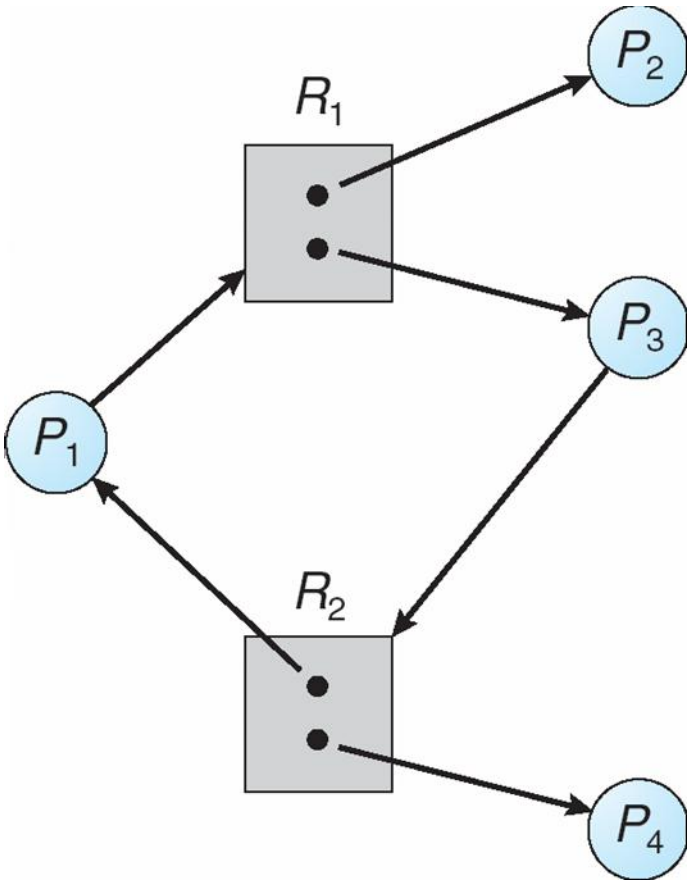
- **Resource-allocation graph**
  - A directed graph
  - Vetices
    - Processes: P = {P1, P2, …, Pn}
    - Resources: R = {R1, R2, …, Rm}
  - Edges
    - Request edge: Pi → Rj
    - Assignment edge: Rj → Pi



Deadlock Modeling (2)

- **Examples**





Deadlock Modeling (3)

- **Basic facts**
  - Having a cycle is a necessary condition for deadlock.
    - Deadlock can arise if four conditions hold simultaneously.
  - 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.

Handling Deadlocks

- **Strategies for handling deadlocks**
  - **Deadlock prevention**
    - Restrain how requests are made.
    - Ensure that at least one necessary condition cannot hold.
  - **Deadlock avoidance**
    - Require additional information about how resources are to be requested.
    - Decide to approve or disapprove requests on the fly.
  - **Deadlock detection and recovery**
    - Allow the system to enter a deadlock state and then recover.
  - **Just ignore the problem altogether!**

Deadlock Prevention (1)

- **Mutual exclusion**
  - Not required for sharable resources, but must hold for non-sharable resources
  - Some devices (e.g., printer) can be spooled
    - Only the printer daemon uses printer resource
    - Not all devices can be spooled.
    - Competition for disk space for spooling can itself lead to deadlock.
  - Principles
    - Avoid assigning resource when not absolutely necessary.
    - Make as few processes as possible actually claim the resource.

Deadlock Prevention (2)

- **Hold and wait**
  - Must guarantee that whenever a process requests a resource, it does not hold any other resources.
    (1) Require process to request and be allocated all its resources before it begins execution.
    (2) Allow process to request resources only when the process has none.
  - Problems
    – May not know required resources at start of run.
    – Low resource utilization.
    – Starvation is possible.

Deadlock Prevention (3)

- **No preemption**

  (1) If a process must wait for another resource, all resources currently being held are implicitly preempted.

  (2) If the requesting resources are allocated to some other process that is waiting for additional resources, preempt the desired resources from the waiting process.

  - Can be applied to resources whose state can be easily saved and restored later.
    - e.g., CPU registers, memory

Deadlock Prevention (4)

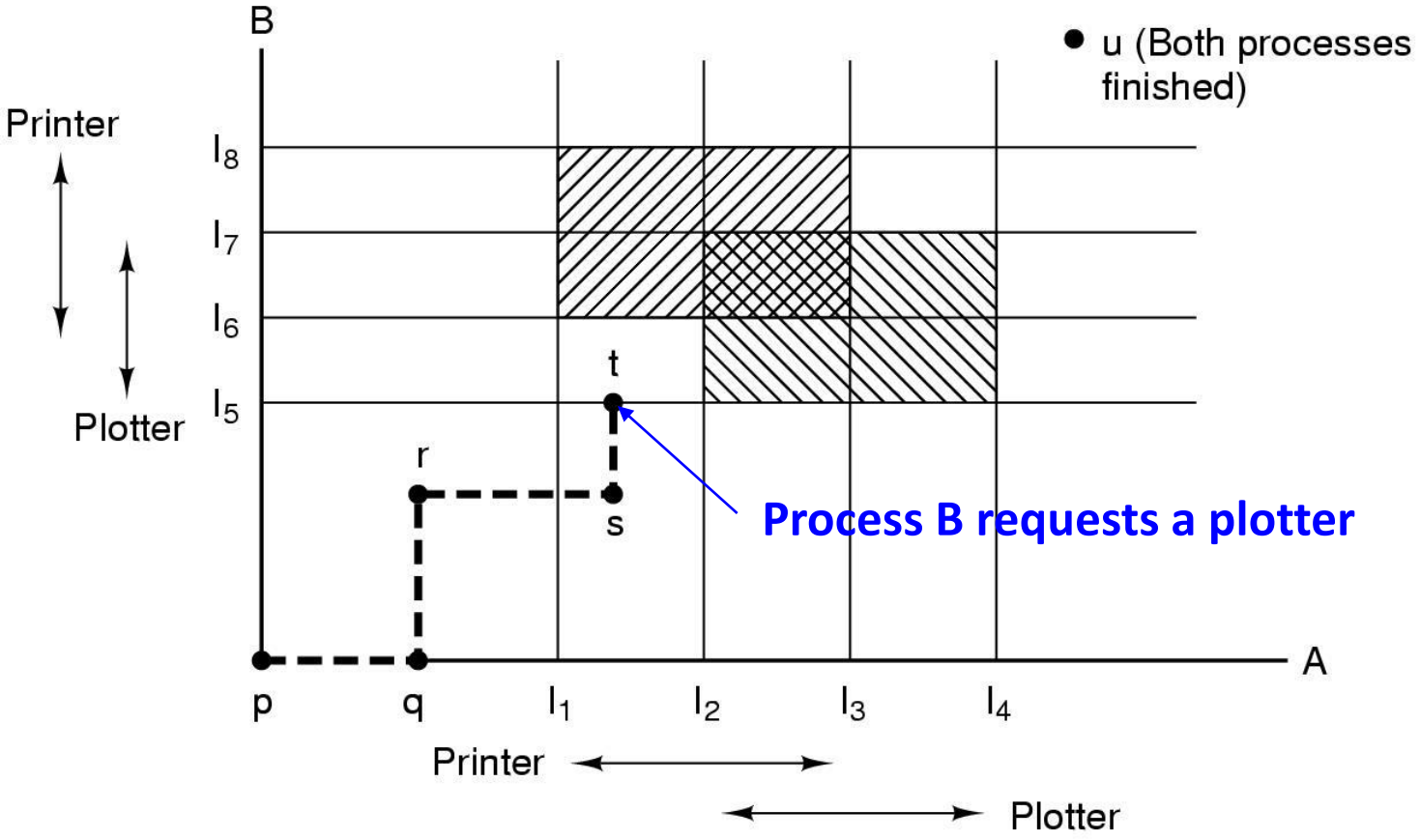
- **Circular wait**
  - Impose a total ordering of all resource types, as a one-to-one function *F.*
    - *F: R* $\rightarrow$ *N,* where R = {R1, R2, …, Rn} is the set of resource types and N is the set of natural numbers.
    - e.g., *F*(tape drive)=1, *F*(disk)=5, *F*(printer)=8, etc.
  - Each process requests resources in an increasing order of enumeration.
  - Whenever a process requests an instance of Rj, it has released any resources Ri such that *F*(Ri) >= *F*(Rj)
  - *F* should be defined according to the normal order of usage of the resources in a system.

Deadlock Avoidance (1)

- **Avoiding deadlocks**
  - Can the system decide whether granting a resource is safe or not and only make the allocation when it is safe?
  - YES! – only if certain information is available in advance.
    - e.g., the maximum number of resources of each type
  - Dynamically examines the resource allocation state to ensure that a circular wait can never exist.

Deadlock Avoidance (2)

- **Resource trajectories**



Deadlock Avoidance (3)

- **Safe state**
  - A state is safe if the system can allocate resources to each process in some order and still avoid a deadlock.
  - A sequence of processes <P1,P2,..Pn> is a safe sequence if, for each Pi, the resources that Pi can still request can be satisfied by the currently available resources + the resources held by all the Pj, with j < i.
    - If the resources that Pi needs are not immediately available, then Pi waits until all Pj have finished.
    - When Pi terminates, Pi+1 can obtain its needed resources, and so on.

Deadlock Avoidance (4)

- Examples

Total: 10units

	Has	Max
P0	3	9
P1	2	4
P2	2	7

Free: 3

P1 requests 2 units — SAFE

P1 gets 2 units

P0	3	9
P1	4	4
P2	2	7

Free: 1

P1 completes

P0	3	9
P1	0	-
P2	2	7

Free: 5

P2 gets 5 units

P0	3	9
P1	0	-
P2	7	7

Free: 0

P2 completes

P0	3	9
P1	0	-
P2	0	-

Free: 7

P0 requests 1 unit — UNSAFE

P0 gets 1 unit

P0	4	9
P1	2	4
P2	2	7

Free: 2

P1 gets 2 units

P0	4	9
P1	4	4
P2	2	7

Free: 0

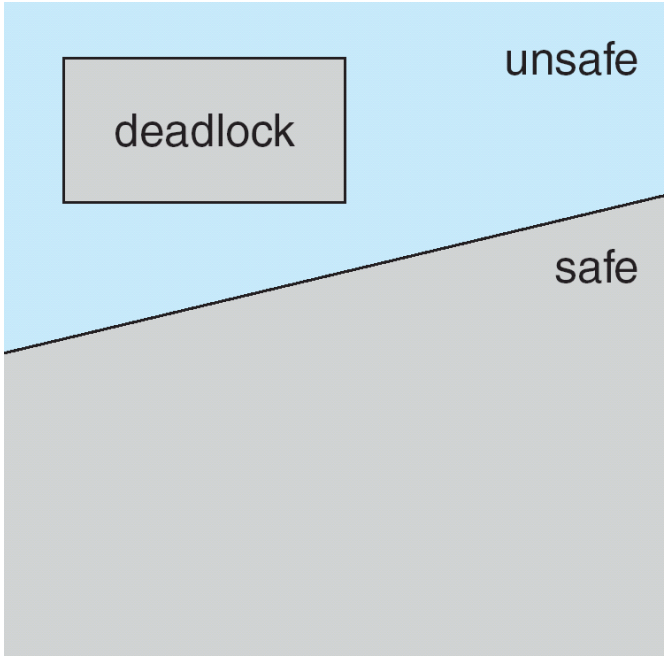
P1 completes

P0	4	9
P1	0	-
P2	2	7

Free: 4

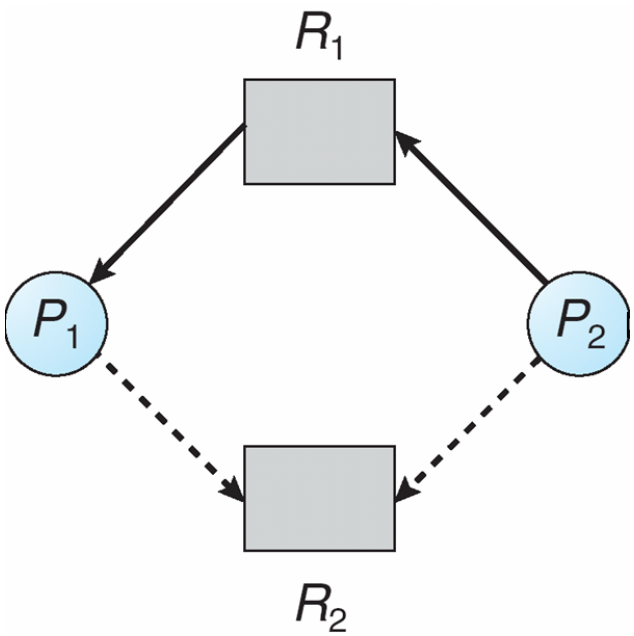
Deadlock Avoidance (5)

- **Basic facts**
  - If a system is in safe state
    $\Rightarrow$ No deadlocks
  - If a system is in unsafe state
    $\Rightarrow$ possibility of deadlock
  - Avoidance: ensure that a system will never enter an unsafe state.



Deadlock Avoidance (6)
Resource allocation graph algorithm

- **Algorithm**
  - Claim edge: Pi → Rj
    - Pi requests Rj in the future.
    - When Pi requests resource Rj, the claim edge (Pi → Rj) is converted to a request edge .
    - When Rj is released by Pi, the assignment edge (Rj → Pi) is reconverted to a claim edge (Pi → Rj).



  - The request can be granted only if converting the request edge to an assignment edge does not result in a cycle.

Deadlock Avoidance (7)
Resource allocation graph algorithm

- **Notes**
  - Resources must be claimed a priori in the system.
  - Applicable when each resource type has only one instance.
  - Detecting a cycle in the graph requires $O(n^2)$ operations, where n is the number of processes.

Deadlock Avoidance (8)
Banker's algorithm

- **Banker's algorithm**
  - Dijkstra, 1965.
  - For resources having multiple instances.
  - 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.

Deadlock Avoidance (9)
Banker's algorithm

- **Details**
  - Check every time if granting a request can lead to an unsafe state.
  - Safety checking algorithm
    - Choose a process whose resource needs are all smaller than or equal to the available resources.
    - If there is no such process, the state is unsafe.
    - If there is, assume the process requests all the resources it needs and finishes.
    - Mark that process as terminated and add all its resources to the available resources.
    - Repeat until all processes are marked terminated.

Deadlock Avoidance (10)
Banker's algorithm

- **Example**

Total Resources

A	B	C
10	5	7

Available Resources

A	B	C
3	3	2

Allocated Resources

	A	B	C
P0	0	1	0
P1	2	0	0
P2	3	0	2
P3	2	1	1
P4	0	0	2

Max Resources

	A	B	C
P0	7	5	3
P1	3	2	2
P2	9	0	2
P3	2	2	2
P4	4	3	3

Needed Resources

	A	B	C
P0	7	4	3
P1	1	2	2
P2	6	0	0
P3	0	1	1
P4	4	3	1

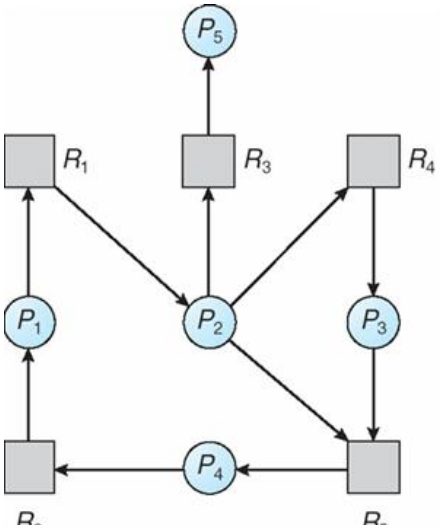
Currently safe: <P1, P3, P4, P0, P2> is a safe sequence

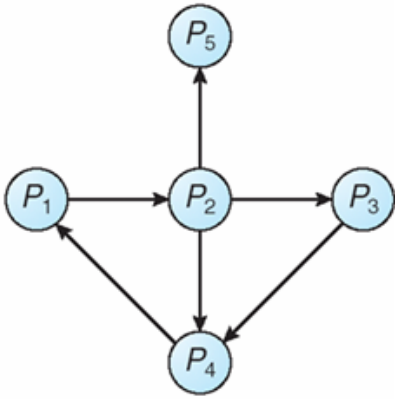
Deadlock Avoidance (11)
Banker's algorithm

- **Notes**
  - Safety checking algorithm requires $O(mn^2)$ operations:
    - m is the number of resource types
    - n is the number of processes
  - Processes rarely know in advance what their maximum resource needs will be.
  - The number of processes is not fixed, but dynamically varying as new users log in and out.
  - Resources that were thought to be available can suddenly vanish.
    - e.g., tape drives or disk drives
  - In practice, few, if any, existing systems use the banker's algorithm for avoiding deadlocks.

Deadlock Detection (1)

- **Single instance / resource type**
  - Maintain a wait-for graph
    - A variant of the resource allocation graph.
    - Pi → Pj if Pi is waiting for Pj
  - Periodically invoke an algorithm that searches for a cycle in the graph.
  - Requires $O(n^2)$ operations





Deadlock Detection (2)

- **Multiple instances / resource type**
  - Very similar to the safety checking in Banker's algorithm

Total Resources

A	B	C
7	2	6

Available Resources

A	B	C
0	0	0

Allocated Resources

	A	B	C
P0	0	1	0
P1	2	0	0
P2	3	0	3
P3	2	1	1
P4	0	0	2

Requested Resources

	A	B	C
P0	0	0	0
P1	2	0	2
P2	0	0	0
P3	1	0	0
P4	0	0	2

What if this becomes 1?

Currently not in a deadlock: <P0, P2, P3, P1, P4> will work.

Deadlock Recovery (1)

- **Process termination**
  - Reclaims all resources allocated to the terminated processes
    - Abort all deadlocked processes
    - Abort one process at a time until the deadlock cycle is eliminated
  - Crudest but simplest way to break a deadlock
  - Not easy since the system may become inconsistent.
    - e.g., files, printing jobs, etc.

Deadlock Recovery (2)

- **Resource preemption**
  - Selecting a victim
    - Minimize cost
  - Rollback
    - Roll back the process to some safe state and restart
  - Starvation
    - Ensure that a process can be picked up as a victim only a finite number of times.
    - Include the number of rollbacks in the cost factor.

Deadlock Ignorance

- **The Ostrich algorithm**
  - Just put your head in the sand and pretend there is no problem at all.
  - Reasonable if
    - Deadlocks occur very rarely
    - Cost of prevention is high
  - UNIX and Windows take this approach.
    - Manual deadlock detection and resolution!
  - It is a trade-off between
    - Convenience
    - Correctness