Deadlock Prevention, Avoidance, and Detection

PRESENTED BY

SHILPA KHURANA

A.P CSE DEPT.

The Deadlock problem

- In a computer system deadlocks arise when members of a group of processes which hold resources are blocked indefinitely from access to resources held by other processes within the group.

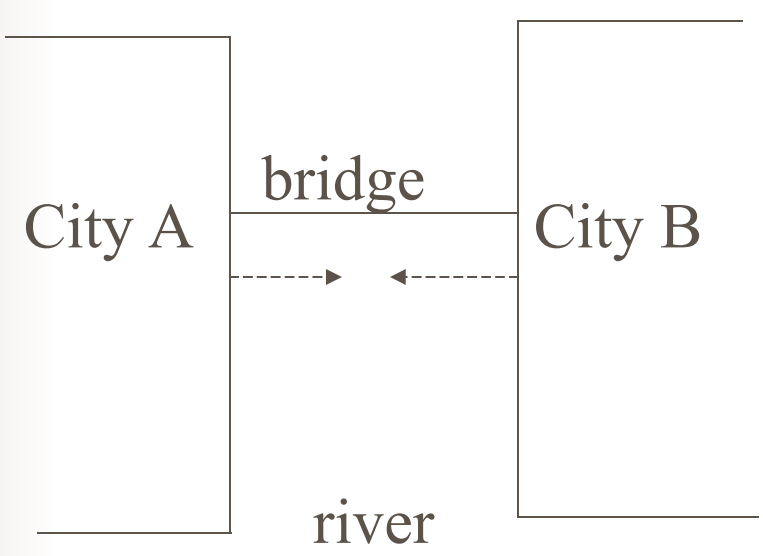
Deadlock example

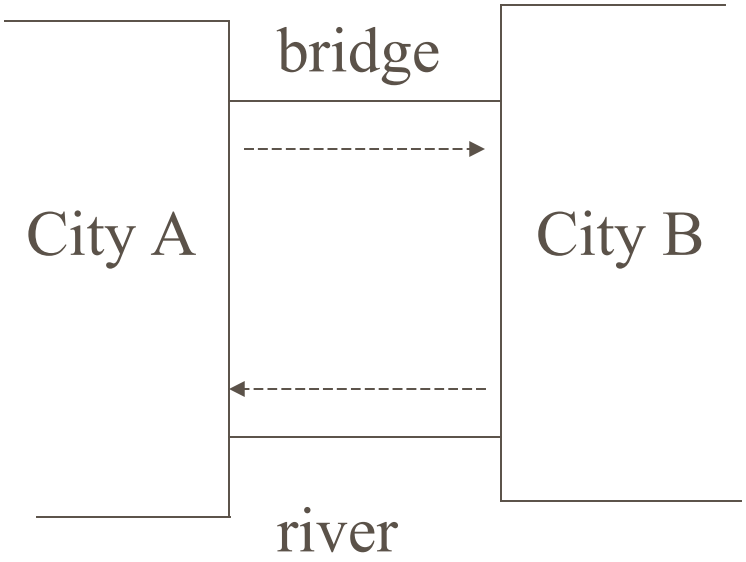
- P_i requests one I/O controller and the system allocates one.
- P_j requests one I/O controller and again the system allocates one.
- P_i wants another I/O controller but has to wait since the system ran out of I/O controllers.
- P_j wants another I/O controller and waits.

Conditions for deadlocks

- *Mutual exclusion*. No resource can be shared by more than one process at a time.
- *Hold and wait*. There must exist a process that is holding at least one resource and is waiting to acquire additional resources that are currently being held by other processes.
- *No preemption*. A resource cannot be preempted.
- *Circular wait*. There is a cycle in the wait-for graph.

An example



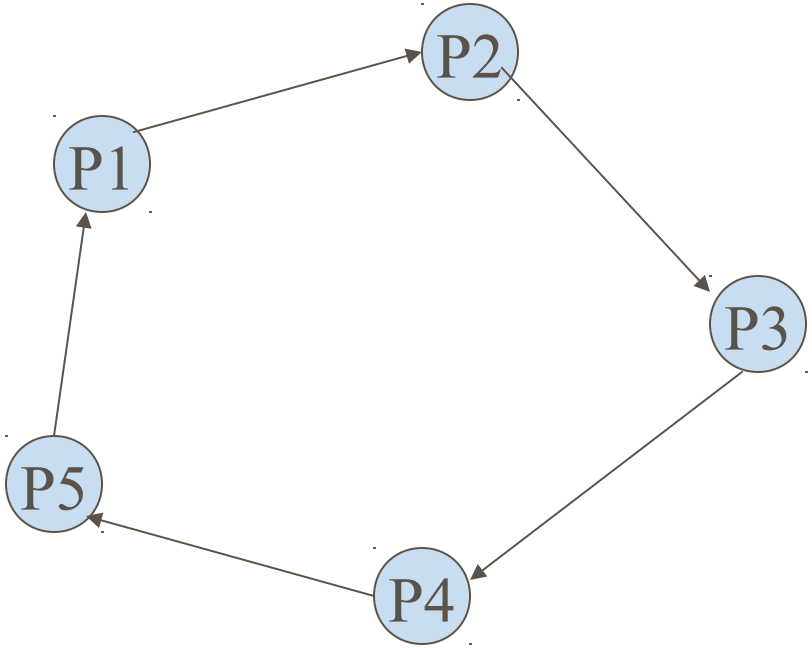


Graph-theoretic models

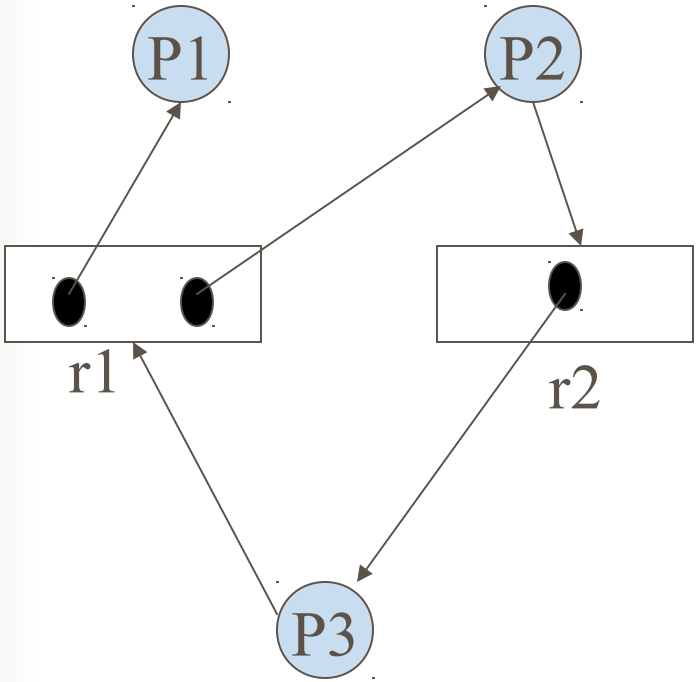
- Wait-for graph.

- Resource-allocation graph.

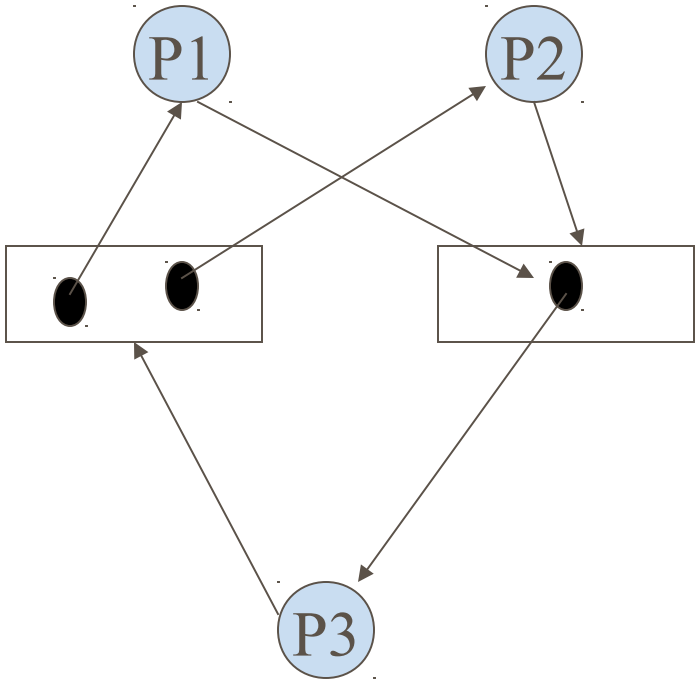
Wait-for graph



Resource allocation graph



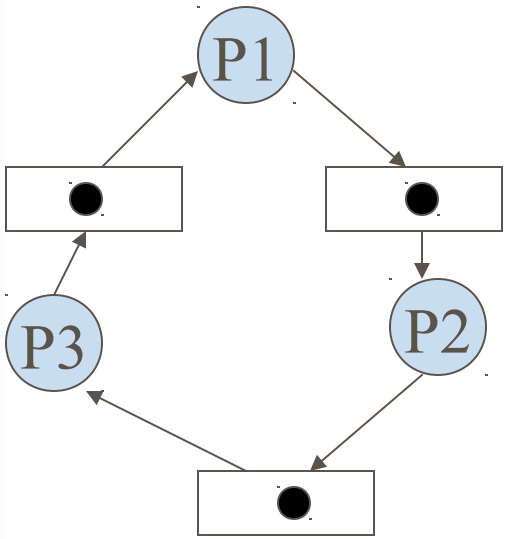
Resource allocation graph
Without deadlock

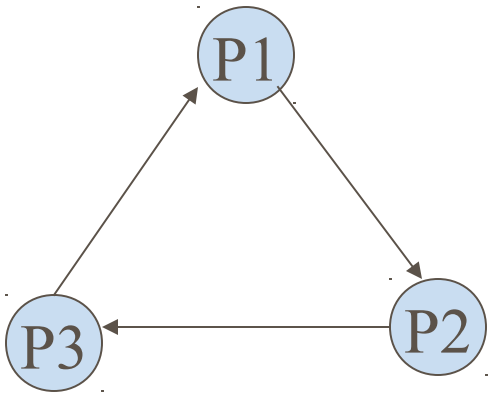


With deadlock

Wait-for graph and Resource-allocation graph conversion

- Any resource allocation graph with a single copy of resources can be transferred to a wait-for graph.



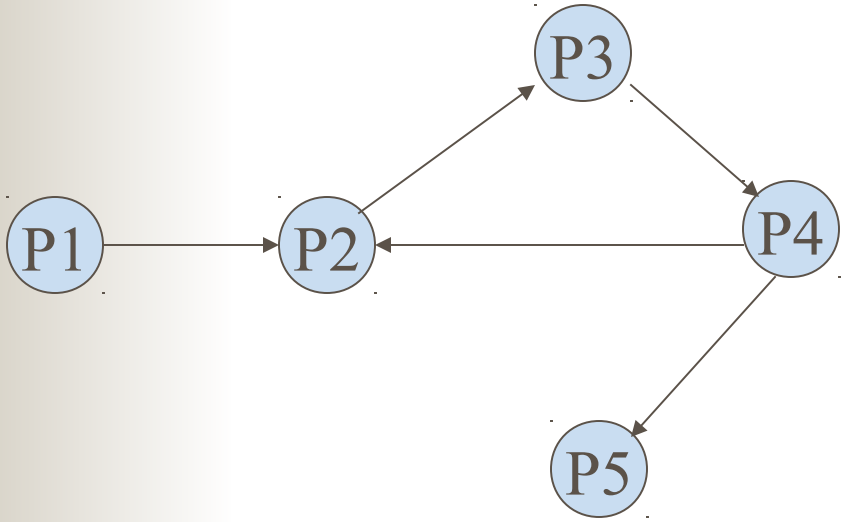


Deadlock conditions

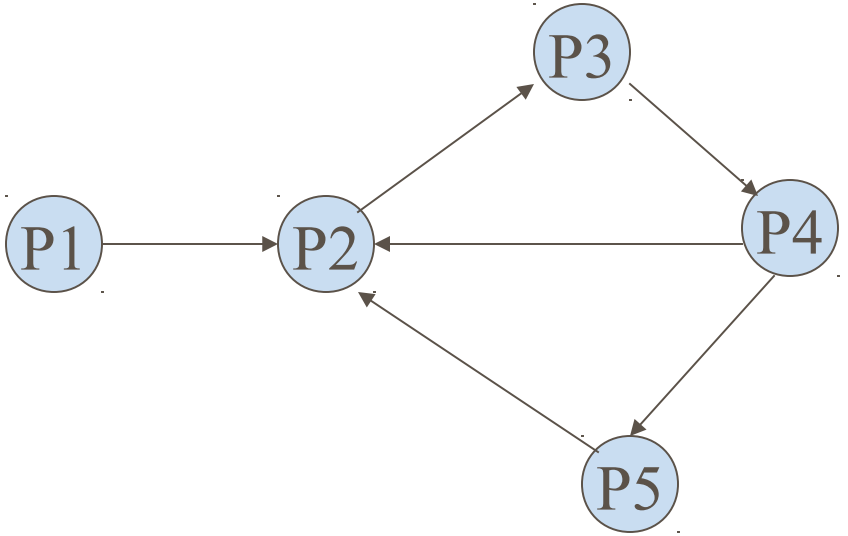
- The condition for deadlock in a system using the AND condition is the existence of a *cycle*.
- The condition for deadlock in a system using the OR condition is the existence of a *knot*.

A knot (K) consists of a set of nodes such that for every node a in K, all nodes in K and only the nodes in K are reachable from node a.

Example: OR condition



No deadlock



Deadlock

Deadlock Prevention

- 1. A process acquires all the needed resources simultaneously before it begins its execution, therefore breaking the hold and wait condition.
- E.g. In the dining philosophers' problem, each philosopher is required to pick up both forks at the same time. If he fails, he has to release the fork(s) (if any) he has acquired.
- Drawback: over-cautious.

- 2. All resources are assigned unique numbers. A process may request a resource with a unique number I only if it is not holding a resource with a number less than or equal to I and therefore breaking the circular wait condition.
- E.g. In the dining philosophers problem, each philosopher is required to pick a fork that has a larger id than the one he currently holds. That is, philosopher P5 needs to pick up fork F5 and then F1; the other philosopher Pi should pick up fork Fi followed by Fi-1.
- Drawback: over-cautions.

- 3. Each process is assigned a unique priority number. The priority numbers decide whether process Pi should wait for process Pj and therefore break the non-preemption condition.
- E.g. Assume that the philosophers' priorities are based on their ids, i.e., Pi has a higher priority than Pj if i <j. In this case Pi is allowed to wait for Pi+1 for I=1,2,3,4. P5 is not allowed to wait for P1. If this case happens, P5 has to abort by releasing its acquired fork(s) (if any).
- Drawback: starvation. The lower priority one may always be rolled back. Solution is to raise the priority every time it is victimized.
- 4. Practically it is impossible to provide a method to break the mutual exclusion condition since most resources are intrinsically non-sharable, e.g., two philosophers cannot use the same fork at the same time.

A Deadlock Prevention Example

Wait-die

- Wants Resource Hold Resource
- Old process -----→ Young process
- 10 20
- Waits

- Wants resource — Holds resource
- Young process 20 — Old process 10

- Dies

- Wait-die is a non-preemptive method.

- **Wound-wait**
- Wants resource Hold resource
- Old process 10 Young process 20
- Preempts

- Wants resource Hold resource
- Young process 20 Old process 10
- Waits

An example

Process id	priority	1st request time	length	Retry interval
P1	2	1	1	1
P2	1	1.5	2	1
P3	4	2.1	2	2
P4	5	3.3	1	1
P5	3	4.0	2	3

Deadlock Avoidance

Four resources ABCD. A has 6 instances, B has 3 instances, C Has 4 instances and D has 2 instances.

Process	Allocation ABCD	Max ABCD
P1	3011	4111
P2	0100	0212
P3	1110	4210
P4	1101	1101
P5	0000	2110

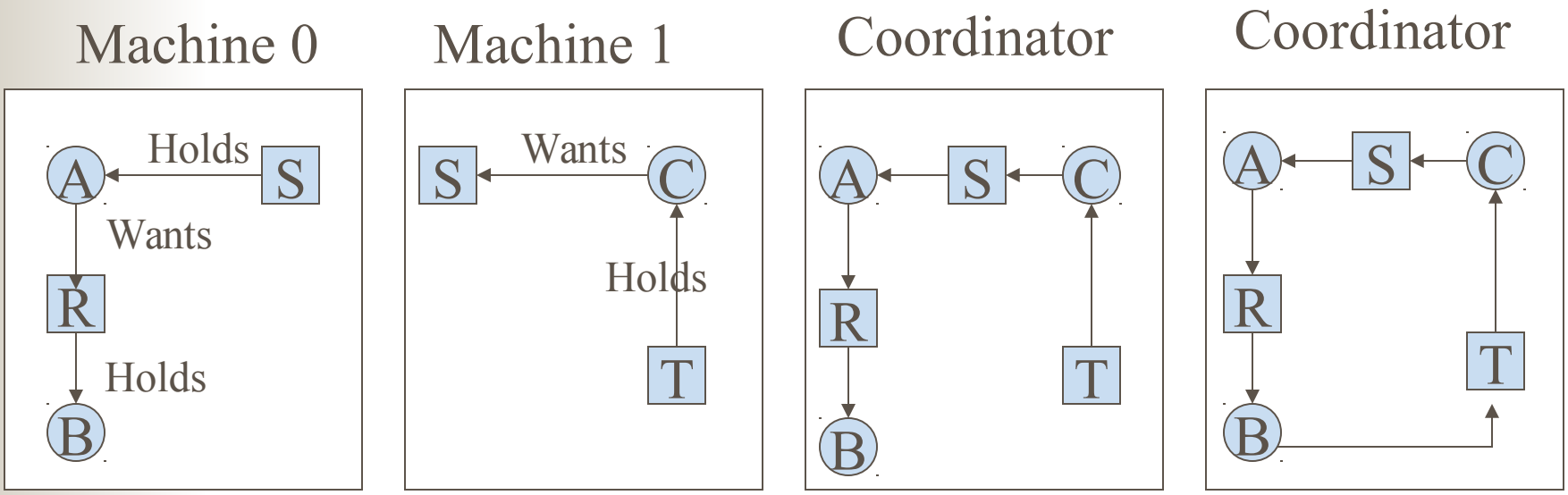
Is the current state safe?

If P5 requests for (1,0,1,0), can this be granted?

Deadlock Detection and Recovery

- Centralized approaches
- Distributed approaches
- Hierarchical approaches

Centralized approaches



B releases R and then B wants T.
But B wants T reaches coordinator first and results in false deadlock.

Distributed approaches

- A copy of the global wait-for graph is kept at each site with the result that each site has a global view of the system.
- The global wait-for graph is divided and distributed to different sites.