Distributed Deadlock Detection

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 resource(s) is waiting to acquire additional resources held by other processes.
- **No preemption:** a resource can be released only voluntarily by the process holding it upon its task completion.
- **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_n is waiting for a resource that is held by P_0.

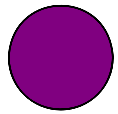
System Model

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

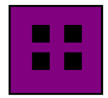
  CPU cycles, memory space, I/O devices
- Each resource type R_i has W_i instances.
- Each process utilizes a resource as follows:
  - request
  - use
  - release

Resource Allocation Graph

- Process

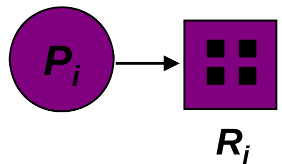


- Resource Type with 4 instances



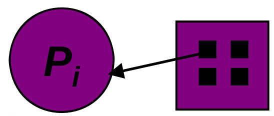
The sequence of Process's recourse utilization

- P_i requests instance of R_j



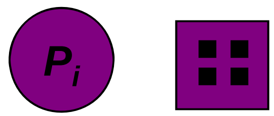
Request edge

- P_i is holding an instance of R_j



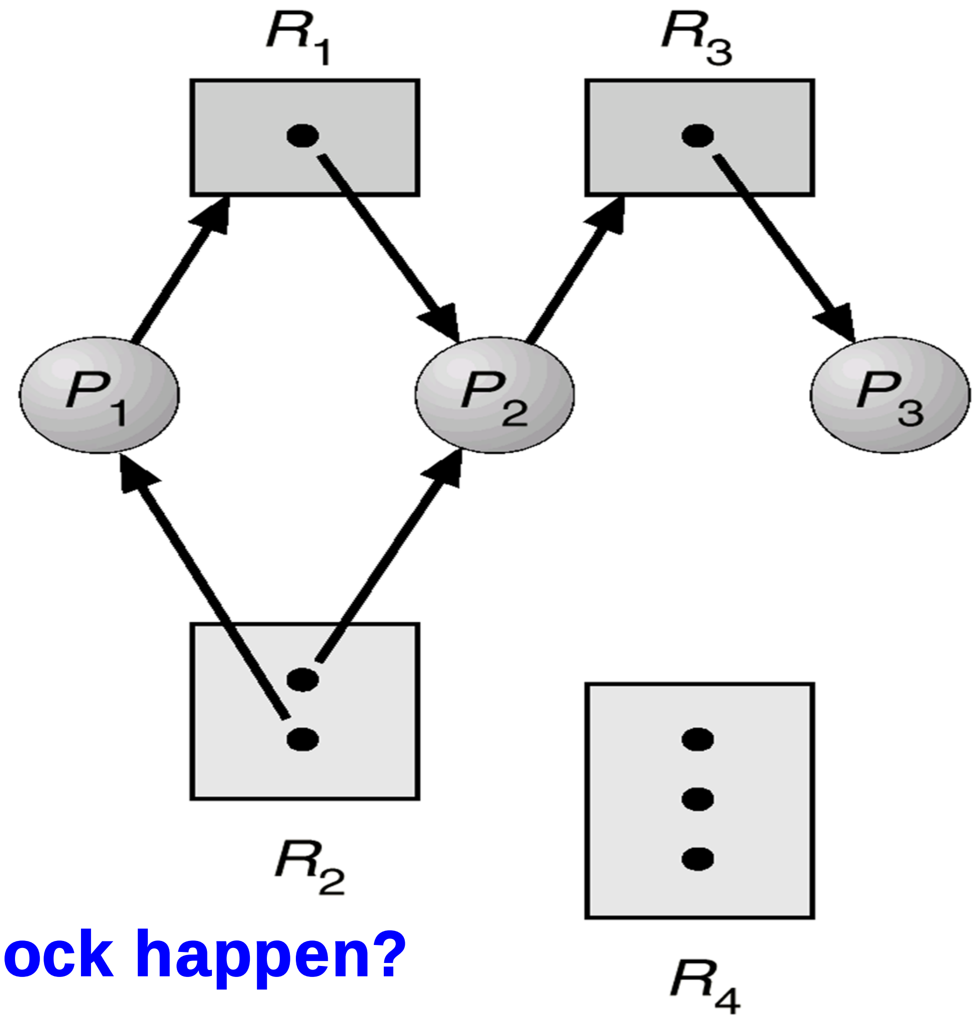
Assignment edge

- P_i releases an instance of R_j



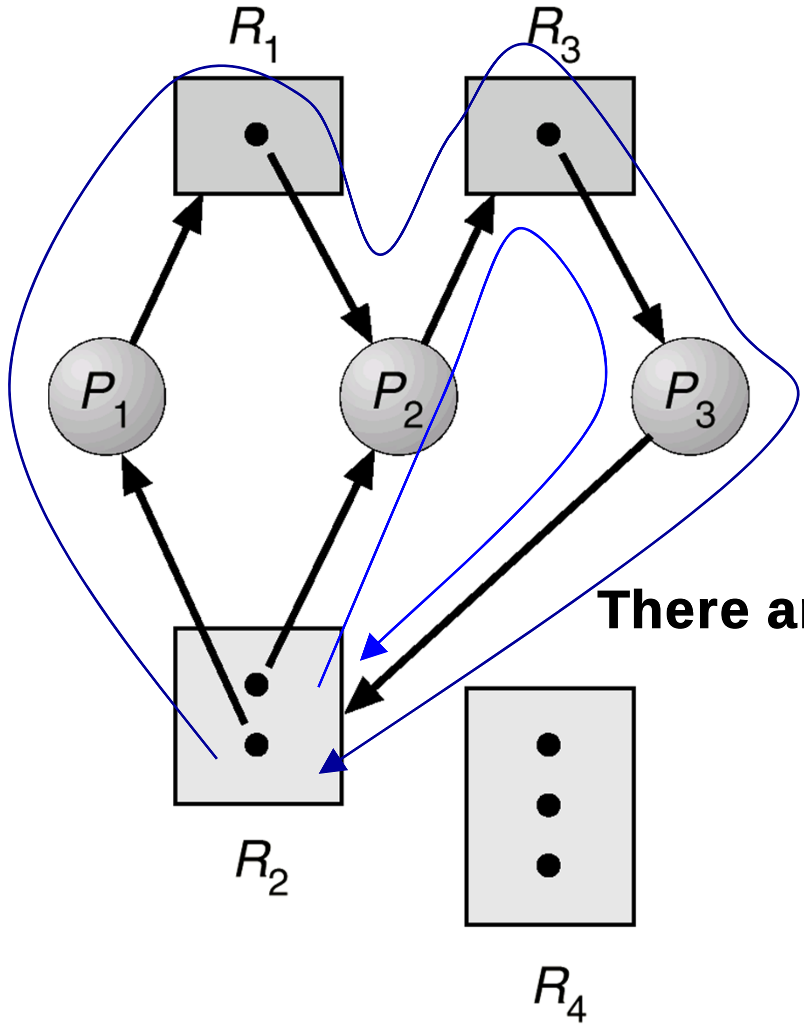
R_j

Resource-allocation graph



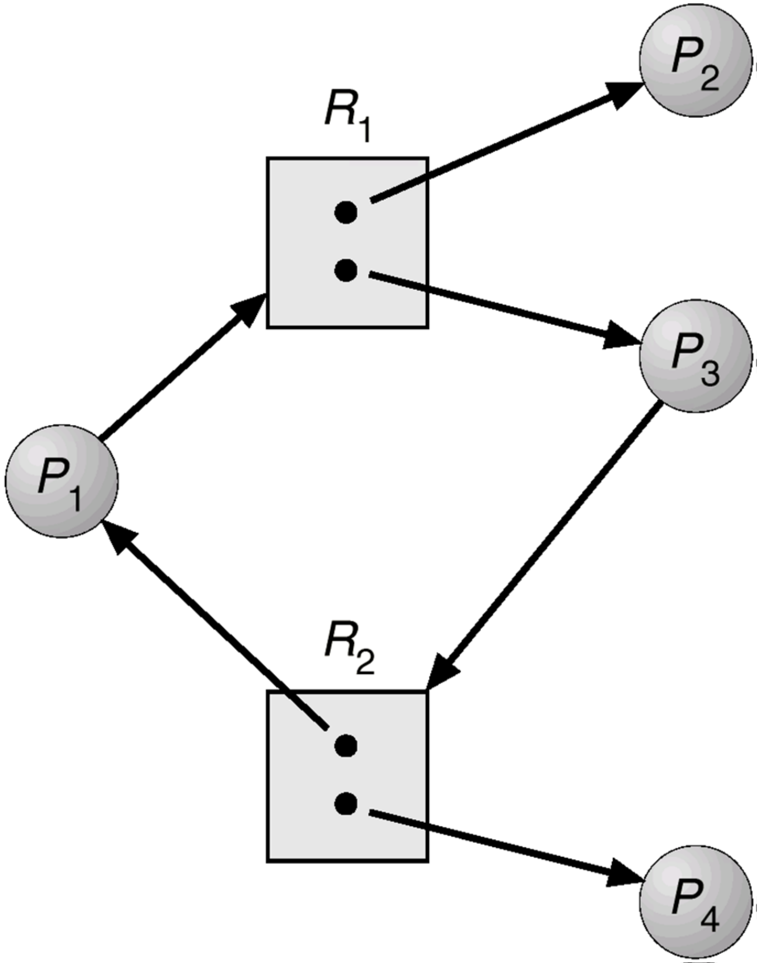
Can a deadlock happen?

Resource Allocation Graph With A Deadlock



There are two cycles found.

Resource Allocation Graph With A Cycle But No Deadlock



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

Two types of deadlocks

- Resource deadlock: uses AND condition.

  AND condition: a process that requires resources for execution can proceed when it has acquired all those resources.
- Communication deadlock: uses OR condition.

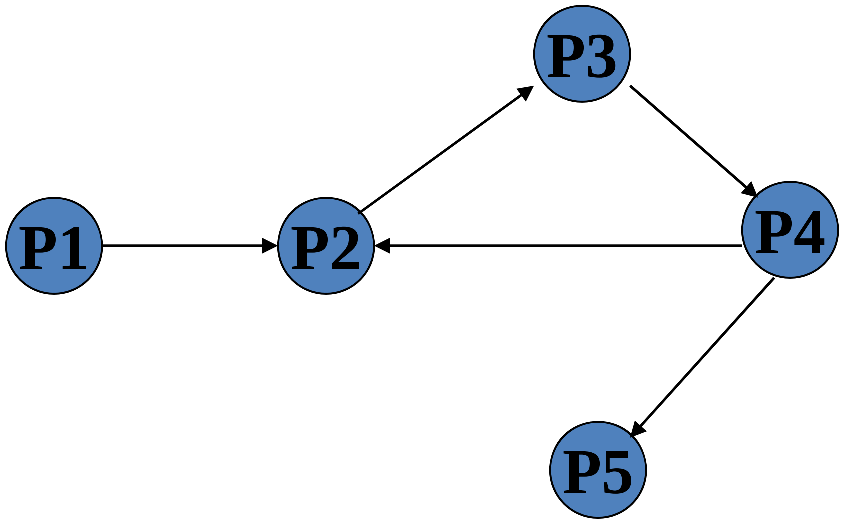
  OR condition: a process that requires resources for execution can proceed when it has acquired at least one of those resources.

Deadlock conditions

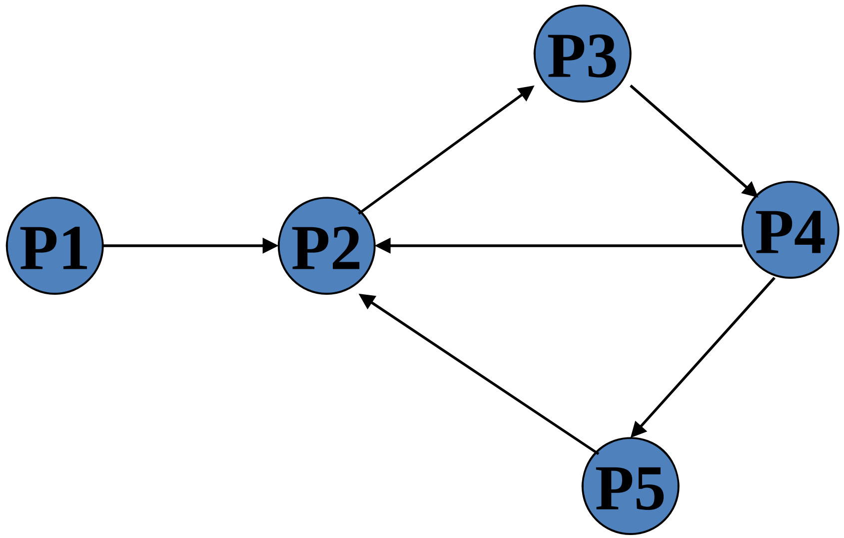
- The condition for deadlock in a system using the AND condition is the existence of a *cycle*.
- The condition for deadlock in a system using the OR condition is the existence of a *knot*.

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

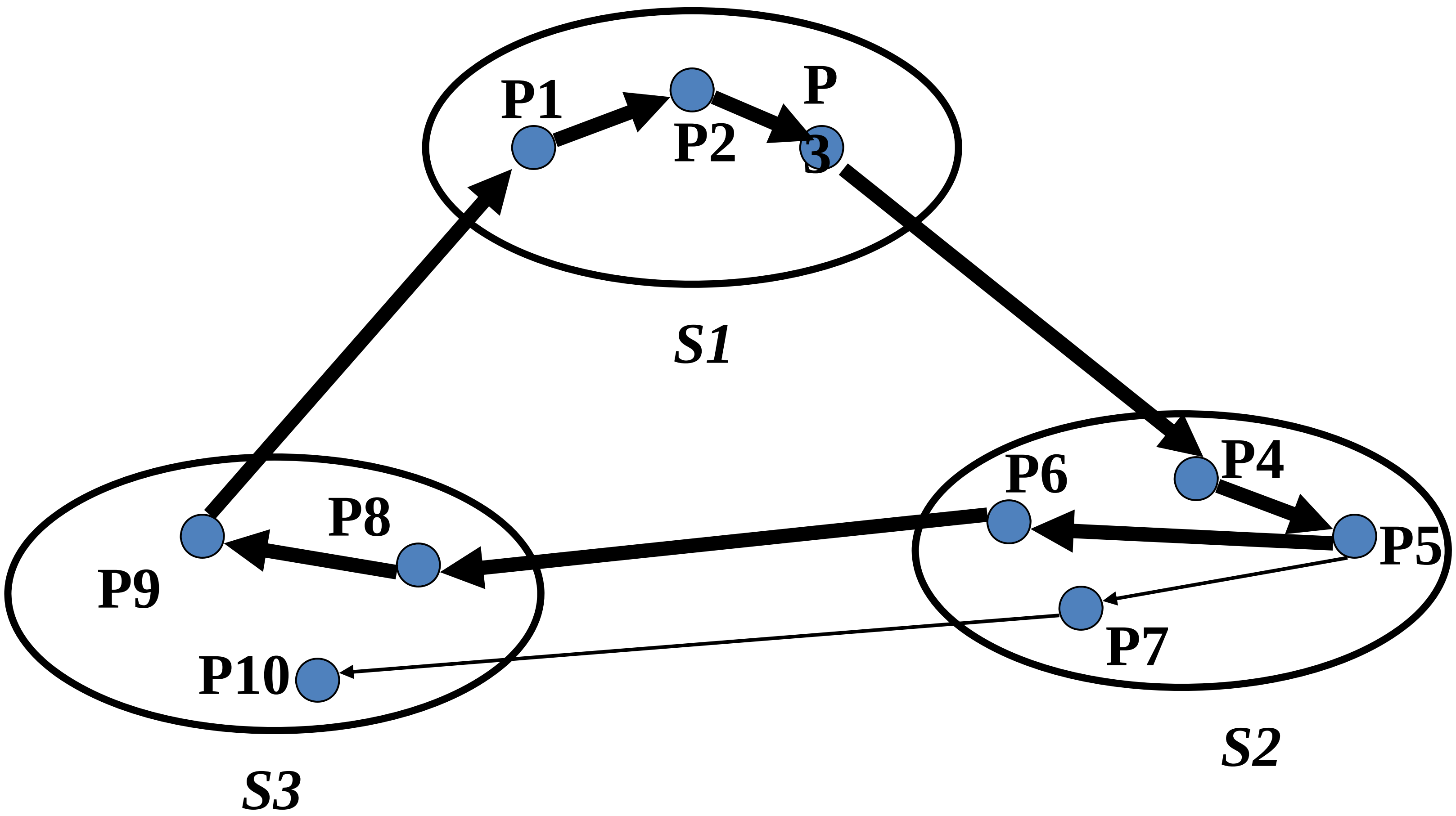
DS Deadlock Detection

- Bi-partite graph strategy modified
  - Use Wait For Graph (WFG or TWF)
    - All nodes are processes (threads)
    - Resource allocation is done by a process (thread) sending a request message to another process (thread) which manages the resource (client - server communication model, RPC paradigm)
  - A system is deadlocked IFF there is a directed cycle (or knot) in a global WFG

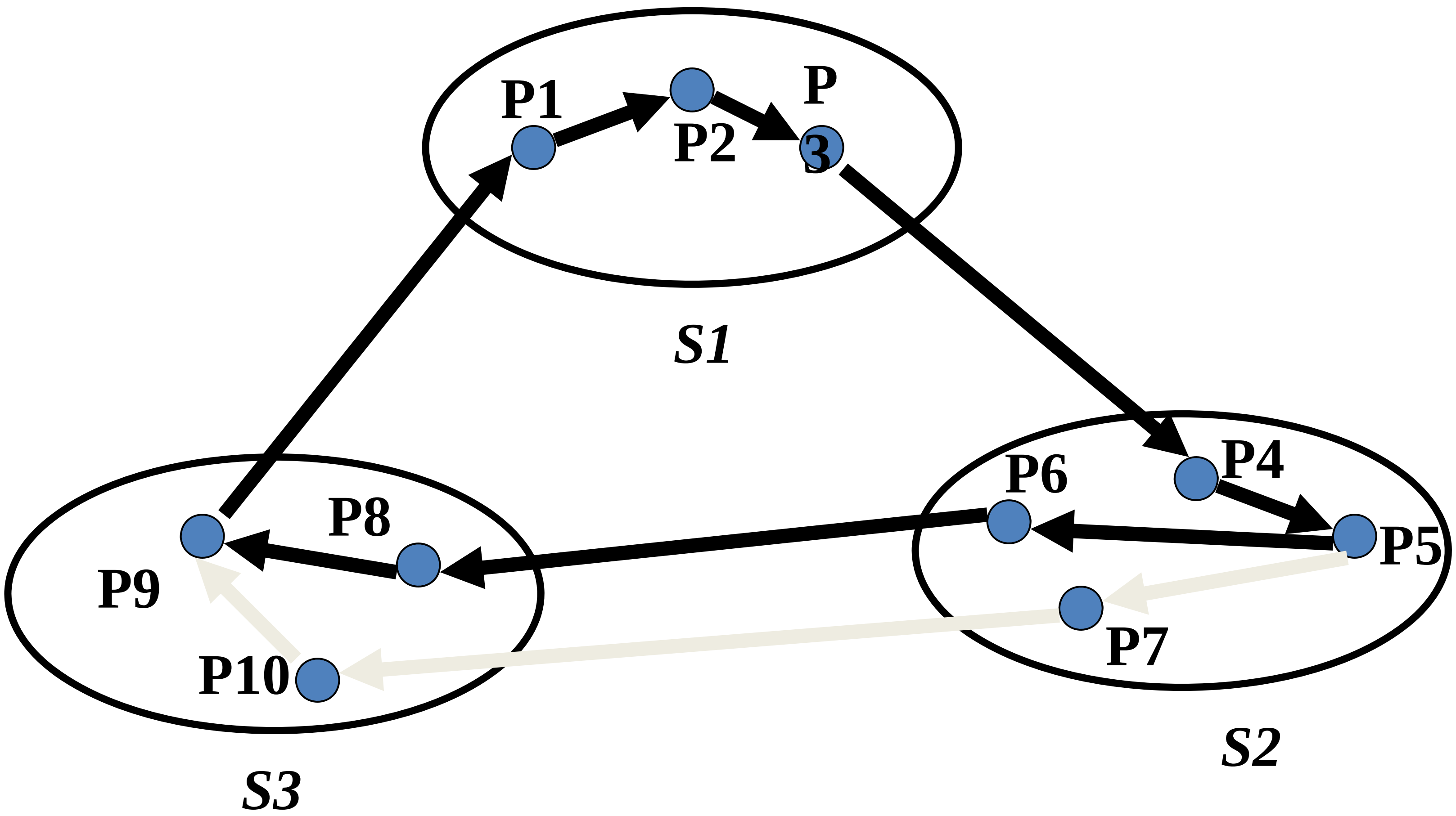
DS Deadlock Detection, Cycle vs. Knot

- The AND model of requests requires all resources currently being requested to be granted to un-block a computation
  - A **cycle is sufficient** to declare a deadlock with this model
- The OR model of requests allows a computation making multiple different resource requests to un-block as soon as any are granted
  - A **cycle** is a **necessary** condition
  - A **knot** is a **sufficient** condition

Deadlock in the AND model; there is a cycle but no knot
No Deadlock in the OR model



Deadlock in both the AND model and the OR model; there are cycles and a knot



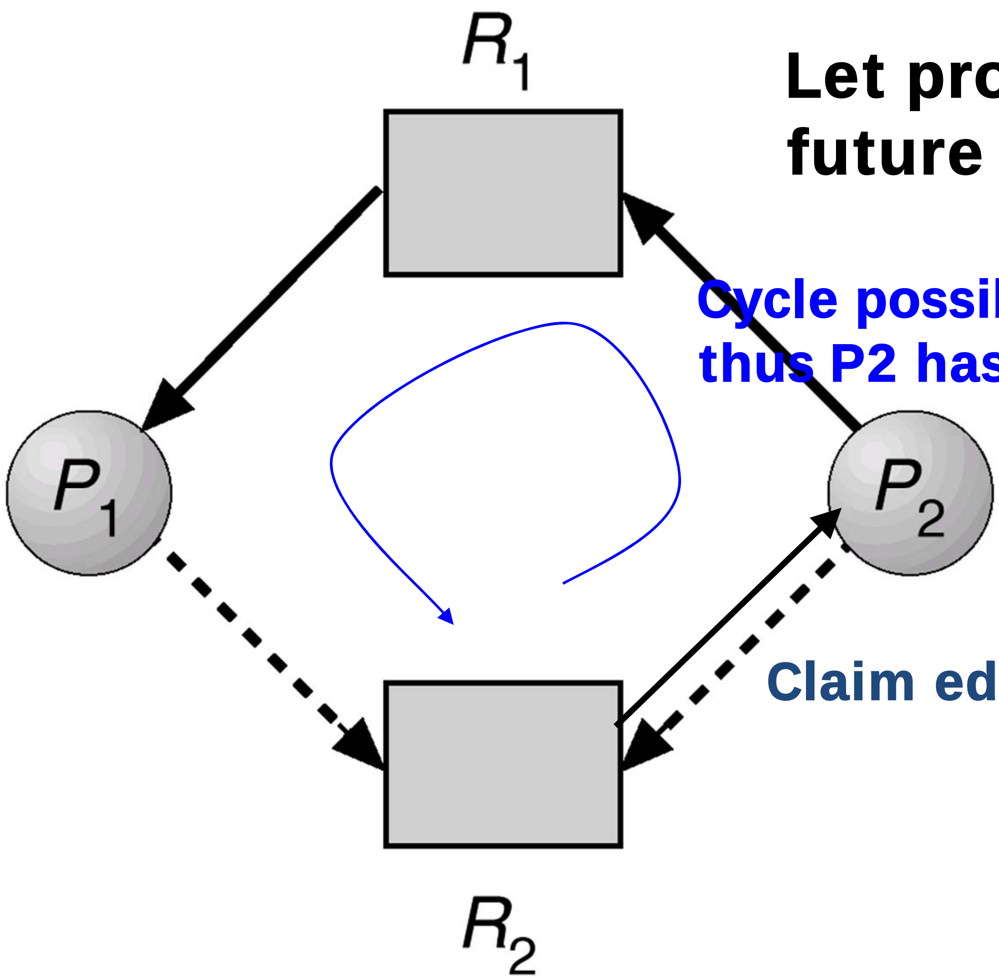
Deadlock Handling Strategies

- Deadlock Prevention
- Deadlock Avoidance
- Deadlock Detection

Distributed Deadlock Prevention

- A method that might work is to order the resources and require processes to acquire them in strictly increasing order. This approach means that a process can never hold a high resource and ask for a low one, thus making cycles impossible.
- With global timing and transactions in distributed systems, two other methods are possible -- both based on the idea of assigning each transaction a global timestamp at the moment it starts.
- When one process is about to block waiting for a resource that another process is using, a check is made to see which has a larger timestamp.
- We can then allow the wait only if the waiting process has a lower timestamp.
- The timestamp is always increasing if we follow any chain of waiting processes, so cycles are impossible --- we can used decreasing order if we like.
- It is wiser to give priority to old processes because
  - they have run longer so the system have larger investment on these processes.
  - they are likely to hold more resources.
  - A young process that is killed off will eventually age until it is the oldest one in the system, and that eliminates starvation.

Deadlock Avoidance



Let processes supply OS with future resource requests

Cycle possibly formed (unsafe state), thus P2 has to wait for a safe state

Claim edge (future request)

Control Organization for Deadlock Detection

- Centralized Control
- Distributed Control
- Hierarchical Control

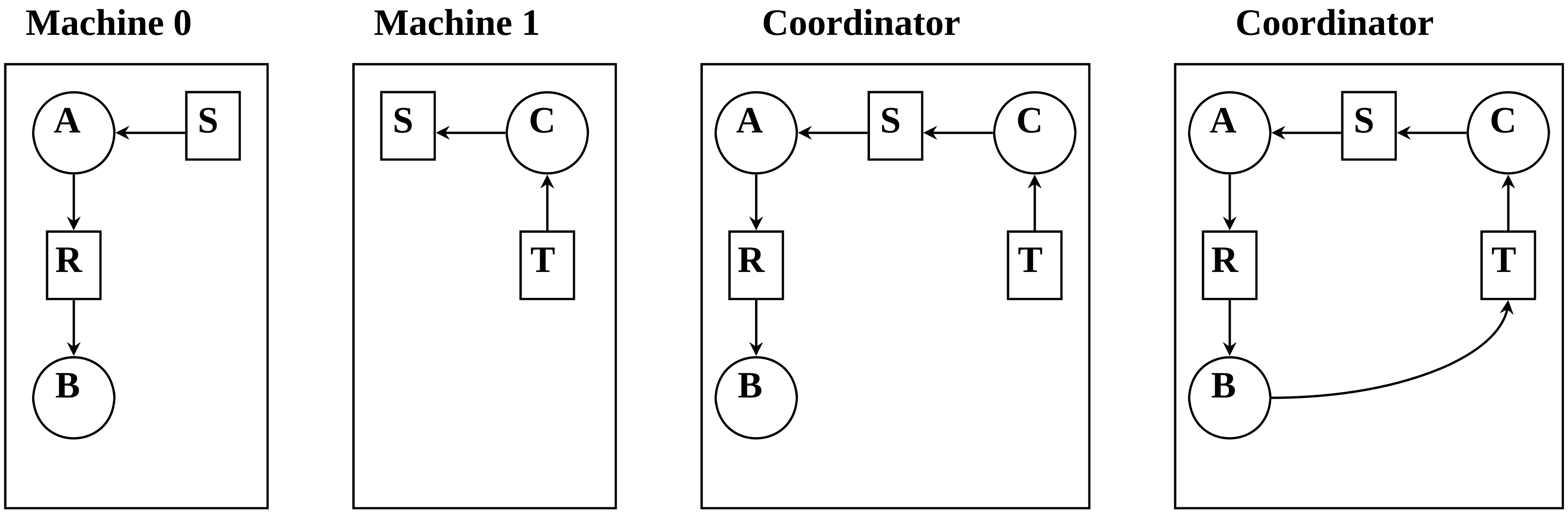
Issues in Deadlock Detection & Resolution

- **Detection**
  - **Progress: No undetected deadlocks**
  - **Safety: No false deadlocks**
- **Resolution**

Centralized Deadlock Detection

- We use a centralized deadlock detection algorithm and try to imitate the non-distributed algorithm.
  - Each machine maintains the resource graph for its own processes and resources.
  - A centralized coordinator maintain the resource graph for the entire system.
  - When the coordinator detect a cycle, it kills off one process to break the deadlock.
  - In updating the coordinator's graph, messages have to be passed.
    - Method 1) Whenever an arc is added or deleted from the resource graph, a message have to be sent to the coordinator.
    - Method 2) Periodically, every process can send a list of arcs added and deleted since previous update.
    - Method 3) Coordinator ask for information when it needs it.

False Deadlocks



B release R, and ask for T

- One possible way to prevent false deadlock is to use the Lamport's algorithm to provide global timing for the distributed systems.
- When the coordinator gets a message that leads to a suspect deadlock:
  - It send everybody a message saying "I just received a message with a timestamp T which leads to deadlock. If anyone has a message for me with an earlier timestamp, please send it immediately"
  - When every machine has replied, positively or negatively, the coordinator will see that the deadlock has really occurred or not.

Centralized Deadlock-Detection Algorithms

- The Ho-Ramamoorthy Algorithms
  - The Two-Phase Algorithm
  - The One-phase Algorithm

Centralized Algorithms

- Ho-Ramamoorthy 2-phase Algorithm
  - Each site maintains a status table of all processes initiated at that site: includes all resources locked & all resources being waited on.
  - Controller requests (periodically) the status table from each site.
  - Controller then constructs WFG from these tables, searches for cycle(s).
  - If no cycles, no deadlocks.
  - Otherwise, (cycle exists): Request for state tables again.
  - Construct WFG based *only* on common transactions in the 2 tables.
  - If the same cycle is detected again, system is in deadlock.
  - Later proved: cycles in 2 consecutive reports *need not* result in a deadlock. Hence, this algorithm detects false deadlocks.

Centralized Algorithms...

- Ho-Ramamoorthy 1-phase Algorithm
  - Each site maintains 2 status tables: *resource status* table and *process status* table.
  - Resource table: transactions that have locked or are waiting for resources.
  - Process table: resources locked by or waited on by transactions.
  - Controller periodically collects these tables from each site.
  - Constructs a WFG from transactions common to both the tables.
  - No cycle, no deadlocks.
  - A cycle means a deadlock.

Distributed Deadlock-Detection Algorithms

- A Path-Pushing Algorithm
  - The site waits for deadlock-related information from other sites
  - The site combines the received information with its local TWF graph to build an updated TWF graph
  - For all cycles ‘EX -> T1 -> T2 -> Ex’ which contains the node ‘Ex’, the site transmits them in string form ‘Ex, T1, T2, Ex’ to all other sites where a sub-transaction of T2 is waiting to receive a message from the sub-transaction of T2 at that site

Edge-Chasing Algorithm

- Chandy-Misra-Haas's Algorithm:
  - A probe(i, j, k) is used by a deadlock detection process Pi. This probe is sent by the home site of Pj to Pk.
  - This probe message is circulated via the edges of the graph. Probe returning to Pi implies deadlock detection.
  - Terms used:
    - Pj is *dependent* on Pk, if a sequence of Pj, Pi1,.., Pim, Pk exists.
    - Pj is *locally dependent* on Pk, if above condition + Pj,Pk on same site.
    - Each process maintains an array *dependenti: dependenti(j)* is true if Pi knows that Pj is dependent on it. (initially set to false for all i & j).

Chandy-Misra-Haas's Algorithm

Sending the probe:

if Pi is locally dependent on itself then deadlock.
else for all Pj and Pk such that
(a) Pi is locally dependent upon Pj, and
(b) Pj is waiting on Pk, and
(c) Pj and Pk are on different sites, send probe(i,j,k) to the home site of Pk.

Receiving the probe:

if (d) Pk is blocked, and
(e) *dependentk(i)* is false, and
(f) Pk has not replied to all requests of Pj,
then begin
dependentk(i) := true;
if k = i then Pi is deadlocked
else ...

Chandy-Misra-Haas's Algorithm

Receiving the probe:

…….

else for all Pm and Pn such that

(a') Pk is locally dependent upon Pm, and

(b') Pm is waiting on Pn, and

(c') Pm and Pn are on different sites, send probe(i,m,n) to the home site of Pn.

end.

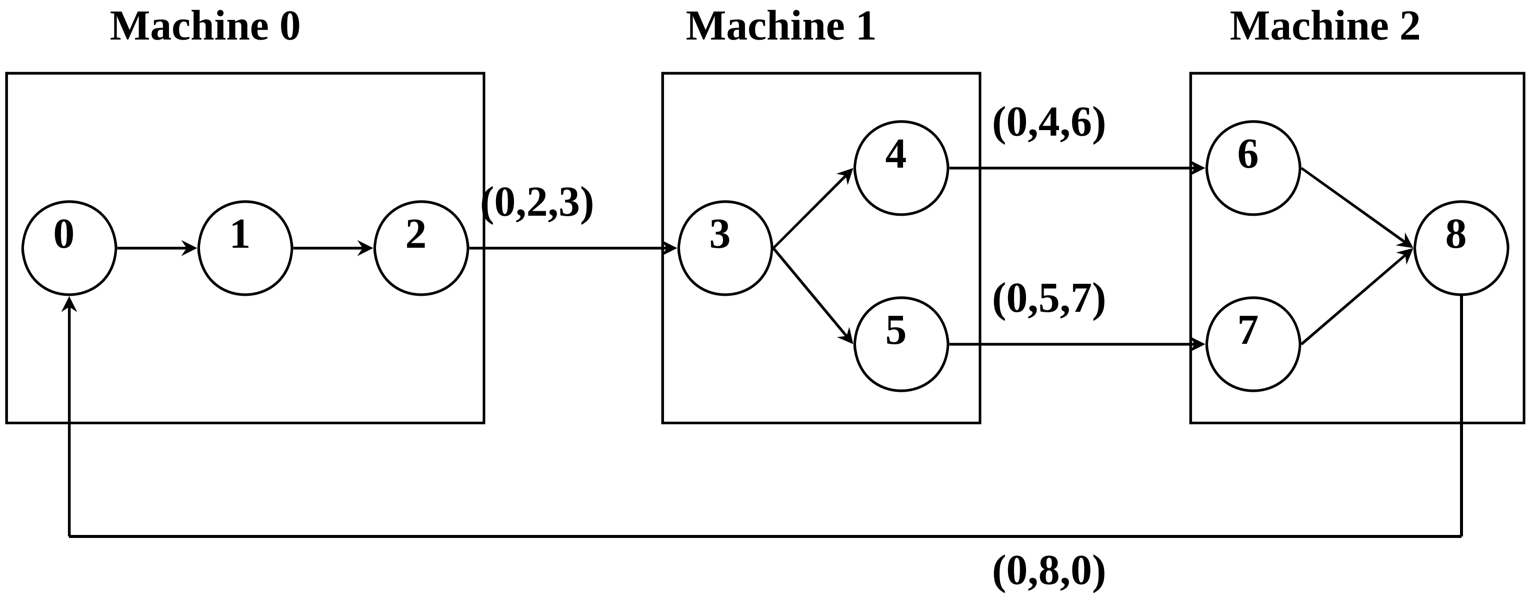
Performance:

For a deadlock that spans m processes over n sites, m(n-1)/2 messages are needed.

Size of the message 3 words.

Delay in deadlock detection O(n).

Chandy-Misra-Haas Algorithm



- There are several ways to break the deadlock:
  - The process that initiates commit suicide -- this is overkilling because several process might initiates a probe and they will all commit suicide in fact only one of them is needed to be killed.
  - Each process append its id onto the probe, when the probe come back, the originator can kill the process which has the highest number by sending him a message. (Even for several probes, they will all choose the same guy)

Other Edge - Chasing Algorithms

- The Mitchell – Merritt Algorithm
- Sinha – Niranjan Algorithm

Chandy et al.'s Diffusion Computation Based Algo

- Initiate a diffusion computation for a blocked process P_i:

  send query (*i, i, j*) to each process P_j in the

  dependent set DS_i of P_i;

  $num_i(i) := |DS_i|; wait_i(i) := true$

- When a blocked process P_k receives a query (*i, j, k*):

  if this is the engaging query for process P_k then

  send *query (i, k, m)* to all P_m in its dependent set DS_k;

  $num_k(i) := |DS_k|; wait_k(i) := true$

  else if $wait_k(i)$ then send a *reply (i, k, j)* to P_j.

Chandy et al.'s Algo. Contd.

- When a process P_k receives a reply (i, j, k):

  if $wait_k(i)$ then begin $num_k(i) := num_k(i) - 1;$

  if $num_k(i) = 0$

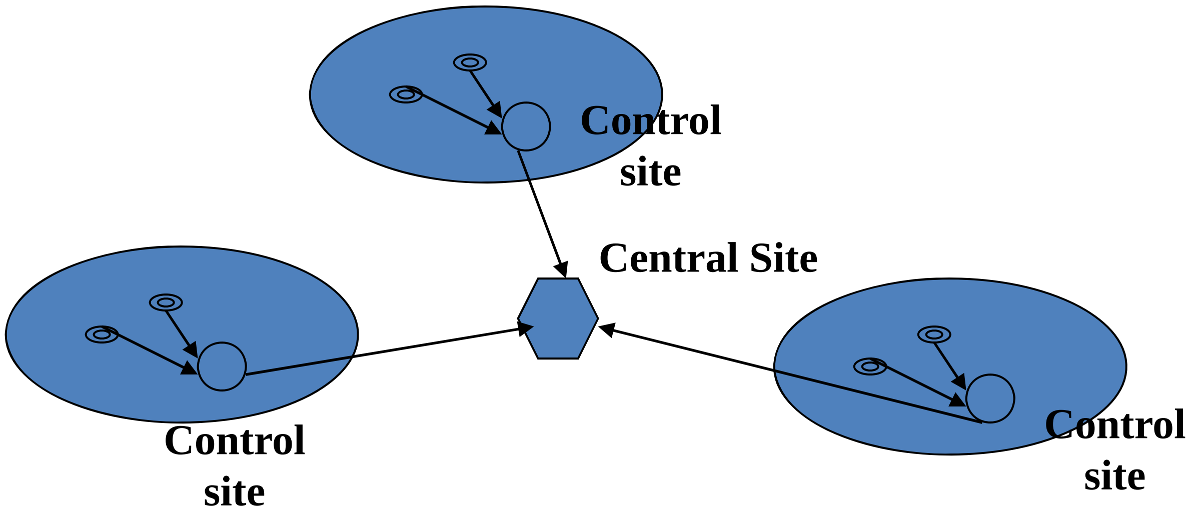
  then if $i = k$ then *declare a deadlock*

  else send *reply (i, k, m)* to the process P_m which

  sent the engaging query

Hierarchical Deadlock Detection

- Follows Ho-Ramamoorthy's 1-phase algorithm. More than 1 control site organized in hierarchical manner.
- Each control site applies 1-phase algorithm to detect (intracluster) deadlocks.
- Central site collects info from control sites, applies 1-phase algorithm to detect intracluster deadlocks.



Persistence & Resolution

- Deadlock persistence:
  – Average time a deadlock exists before it is resolved.
- Implication of persistence:
  – Resources unavailable for this period: affects utilization
  – Processes wait for this period unproductively: affects response time.
- Deadlock resolution:
  – Aborting at least one process/request involved in the deadlock.
  – Efficient resolution of deadlock requires knowledge of all processes and resources.
  – If every process detects a deadlock and tries to resolve it independently -> highly inefficient ! Several processes might be aborted.

Deadlock Resolution

- Priorities for processes/transactions can be useful for resolution.
  - Consider priorities introduced in Obermarck's algorithm.
  - Highest priority process initiates and detects deadlock (initiations by lower priority ones are suppressed).
  - When deadlock is detected, lowest priority process(es) can be aborted to resolve the deadlock.
- After identifying the processes/requests to be aborted,
  - All resources held by the victims must be released. State of released resources restored to previous states. Released resources granted to deadlocked processes.
  - All deadlock detection information concerning the victims must be removed at all the sites.

The End / OR is it deadlock?

- We are now entering the *idle state* , waiting for a message from any of the other processes in the room !
- Don’t make us send out probes!