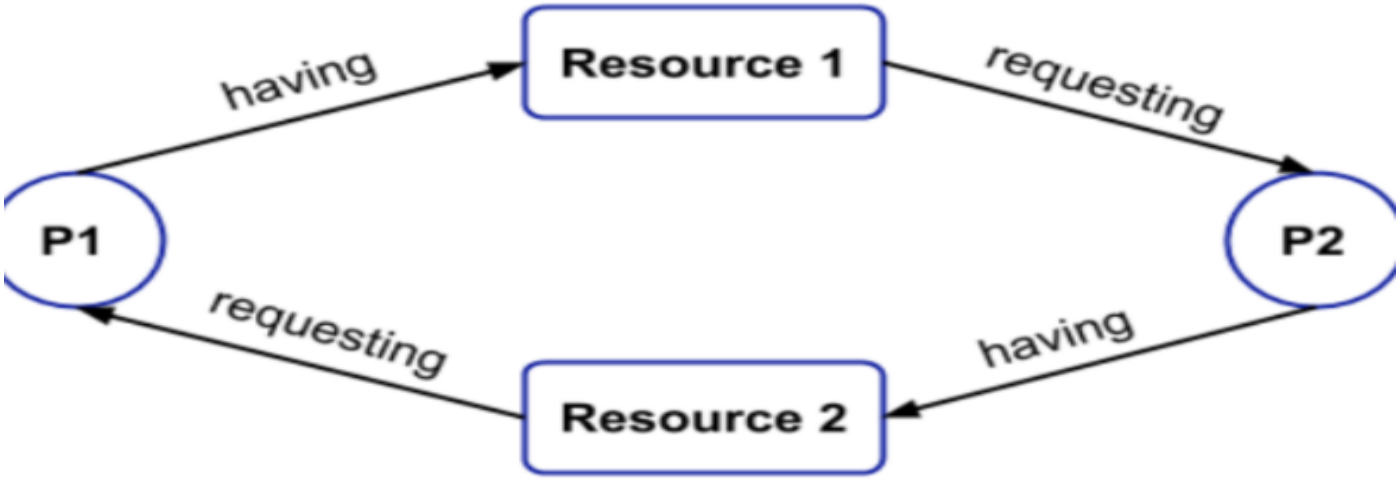
# Deadlock Detection Algorithms

Presented By:
Abhisekh Khanal

Nepal College Of Information Technology (NCIT)

# Deadlock

- A deadlock is a situation in which more than one process is blocked because it is holding a resource and also requires some resource that is acquired by some other process.
- Therefore, none of the processes gets executed.




**For example,** let us assume, For example, let us assume,

- We have two processes P1 and P2. Now, process P1 is holding the resource R1 and is waiting for the resource R2.
- At the same time, the process P2 is having the resource R2 and is waiting for the resource R1.
- So, the process P1 is waiting for process P2 to release its resource and
- At the same time, the process P2 is waiting for process P1 to release its resource. And no one is releasing any resource.
- So, both are waiting for each other to release the resource.
- This leads to infinite waiting and no work is done here.
- This is called **Deadlock.**

# Deadlock Conditions:

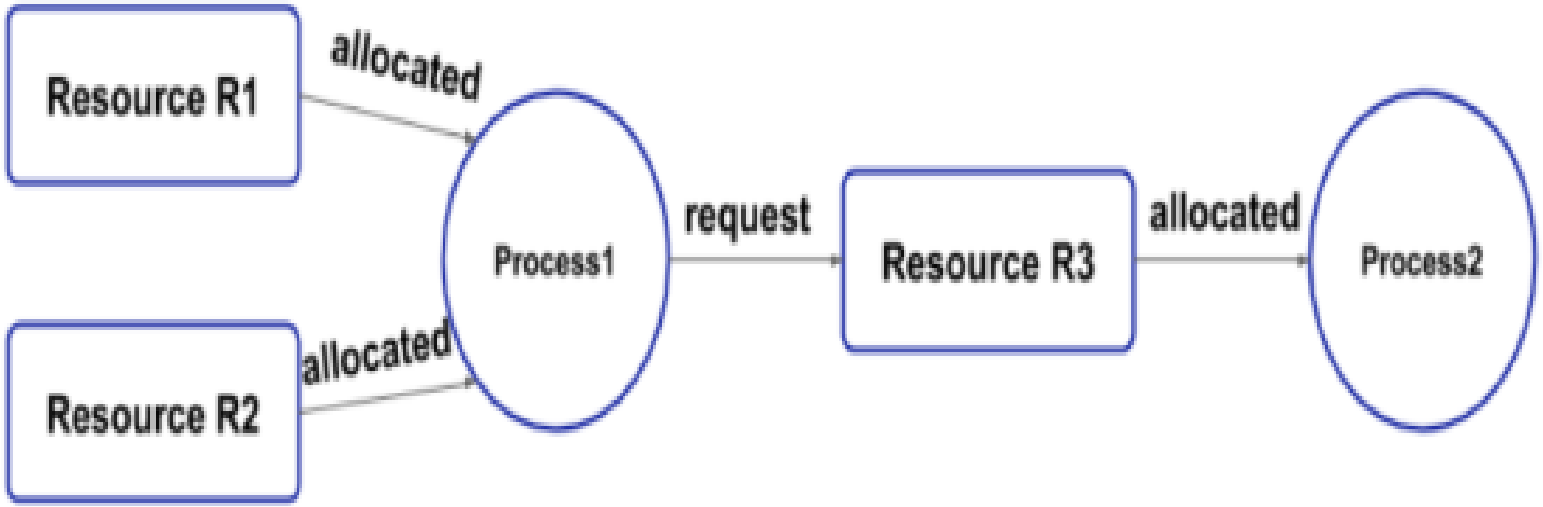
- Mutual Exclusion
- Hold and Wait
- No preemption
- Circular wait

# Mutual Exclusion:

- A resource can be held by only one process at a time.
- In other words, if a process P1 is using some resource R at a particular instant of time, then some other process P2 can't hold or use the same resource R at that particular instant of time.
- The process P2 can make a request for that resource R but it can't use that resource simultaneously with process P1.

# Hold and Wait:

- A process can hold a number of resources at a time and at the same time, it can request for other resources that are being held by some other process.
- For example, a process P1 can hold two resources R1 and R2 and at the same time, it can request some resource R3 that is currently held by process P2.



# No preemption:

- A resource can't be preempted from the process by another process, forcefully.
- For example, if a process P1 is using some resource R, then some other process P2 can't forcefully take that resource.
- If it is so, then what's the need for various scheduling algorithm. The process P2 can request for the resource R and can wait for that resource to be freed by the process P1.

# Circular Wait:

- Circular wait is a condition when the first process is waiting for the resource held by the second process, the second process is waiting for the resource held by the third process, and so on. At last, the last process is waiting for the resource held by the first process.
- So, every process is waiting for each other to release the resource and no one is releasing their own resource. Everyone is waiting here for getting the resource. This is called a circular wait.

# Deadlock Handling

1. Deadlock Ignorance.
2. Deadlock Prevention.
3. Deadlock Avoidance.
4. Deadlock Detection and Recovery

# Deadlock Detection

- There is possibility of deadlock if neither of prevention or avoidance method is applied in a system.
- In such case algorithm must be provided for recovering the system from deadlock.

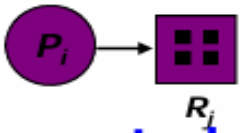
# Resource Allocation Graph

- Process

- Resource Type with 4 instances

**The sequence of Process's recourse utilization**

- $P_i$ requests instance of $R_j$




**Request edge**

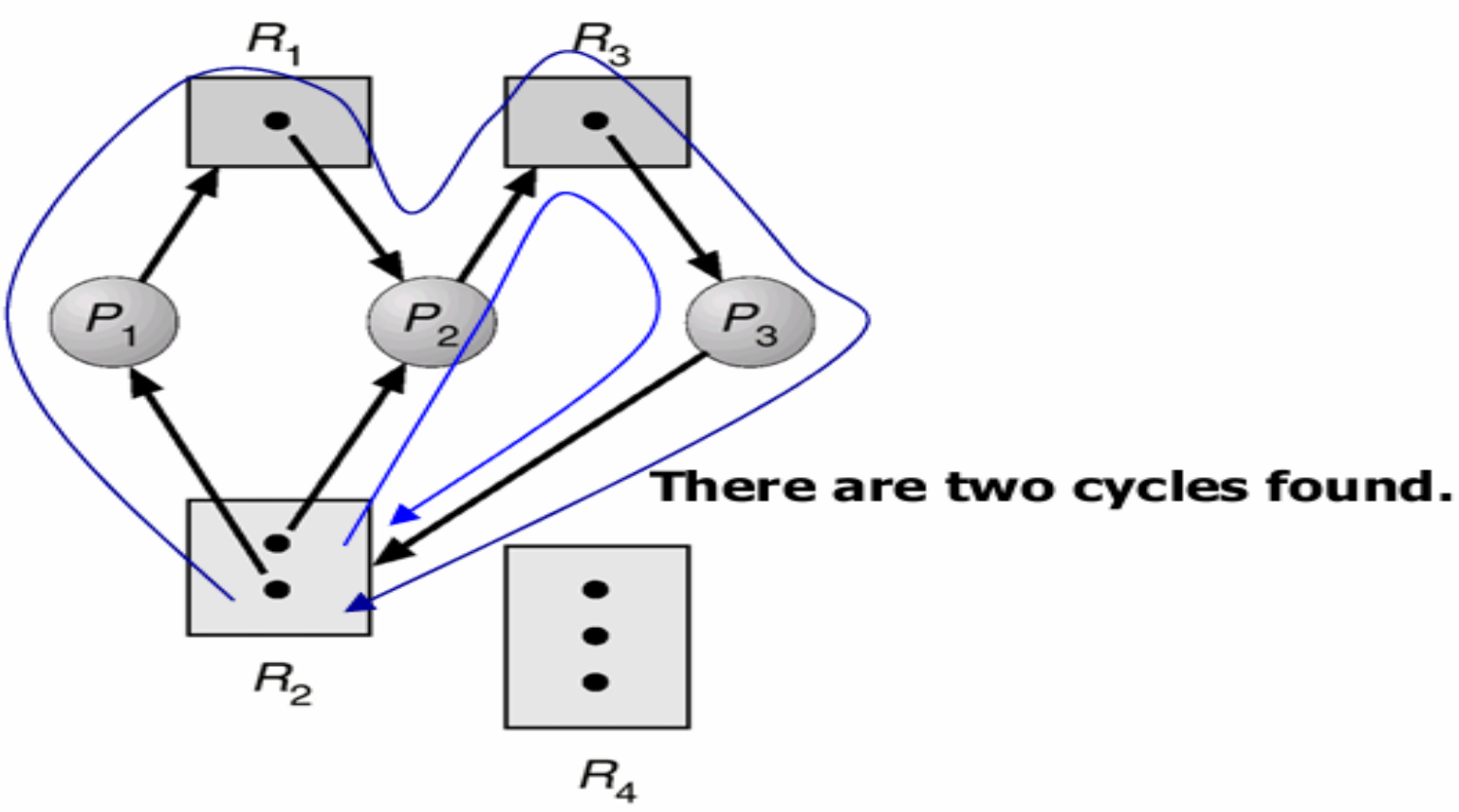
- $P_i$ is holding an instance of $R_j$


**Assignment edge**

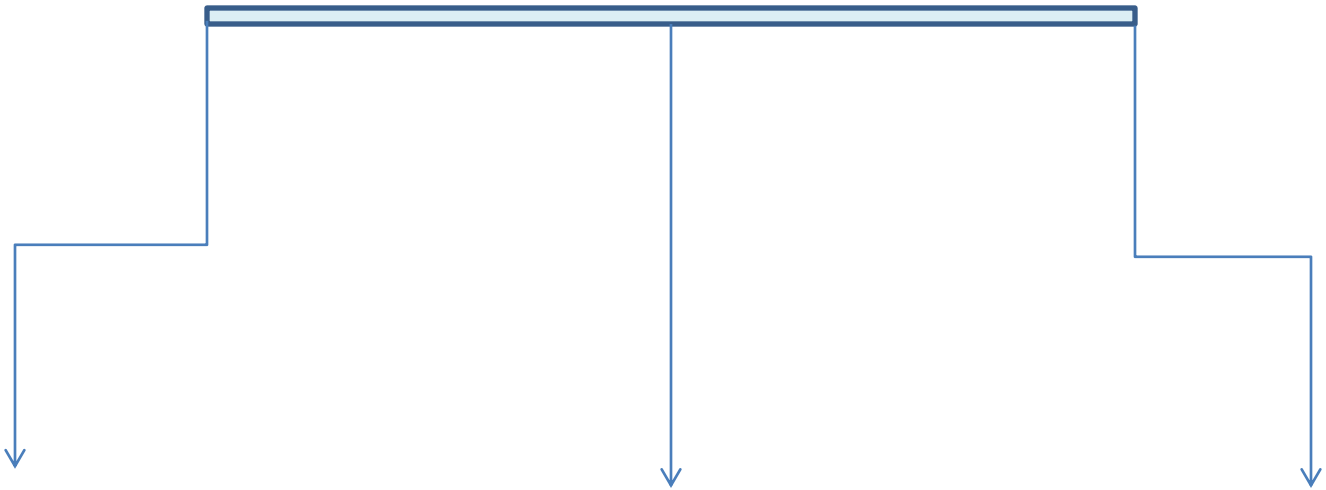
- $P_i$ releases an instance of $R_j$


$R_j$

# Resource Allocation Graph With A Deadlock



# Deadlock Detection Algorithms



Centralized Deadlock Detection

Distributed Deadlock Detection

Hierarchical Deadlock Detection

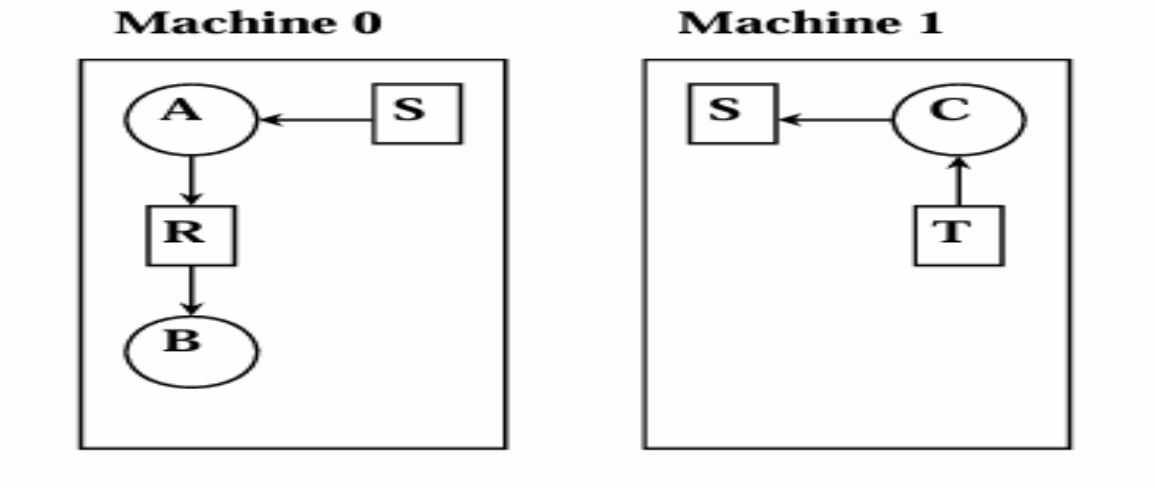
# Centralized Deadlock Detection

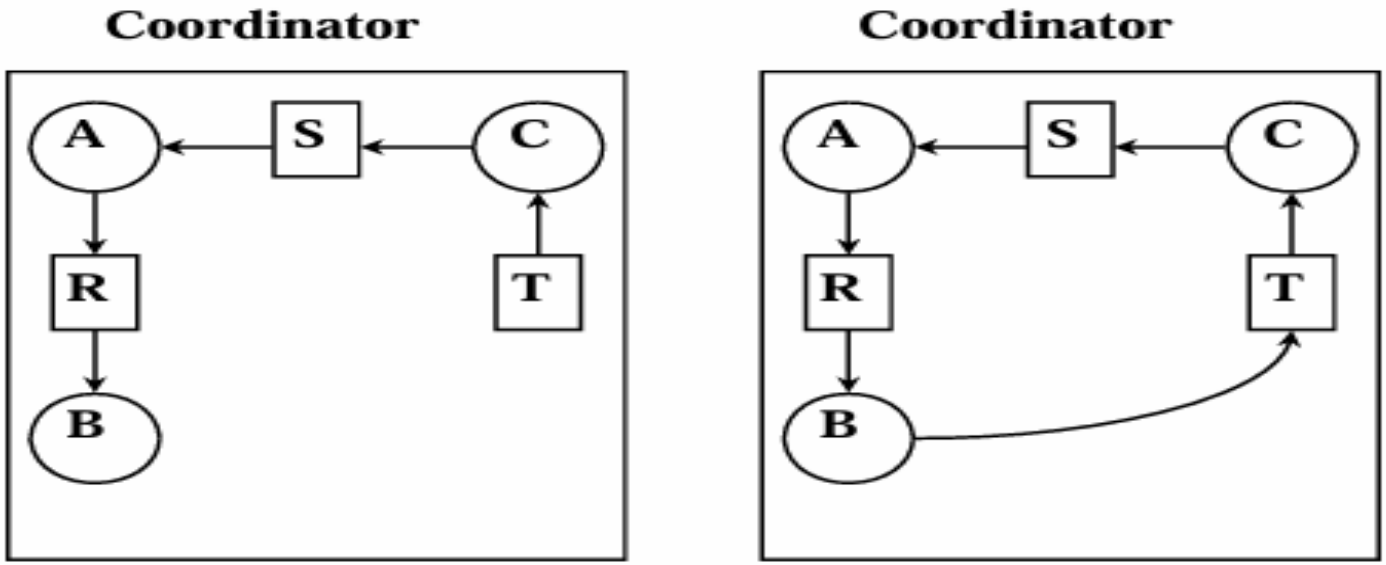
- Control site responsible for constructing the WFG
- Searches for cycles
- Maintains WFG or built it whenever DD to be carried out.
- Simple and easy to implement
- Single point of failure
- Congested near the control site because of messages from all other sites.

# 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***

# False Deadlocks

- One possible way to prevent false deadlock is to us e 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 me ssage with a timestamp T which leads to deadlock. If an yone has a message for me with an earlier timestamp, p lease send it immediately"
  - When every machine has replied, positively or negatively , the coordinator will see that the deadlock has really oc curred or not.

# Centralized Deadlock Detection Algorithms

- The Ho-Ramamoorthy Algorithms

  - The Two-Phase Algorithm

  - The One-Phase Algorithm

# 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

# Ho-Ramamoorthy 1-phase Algorithm

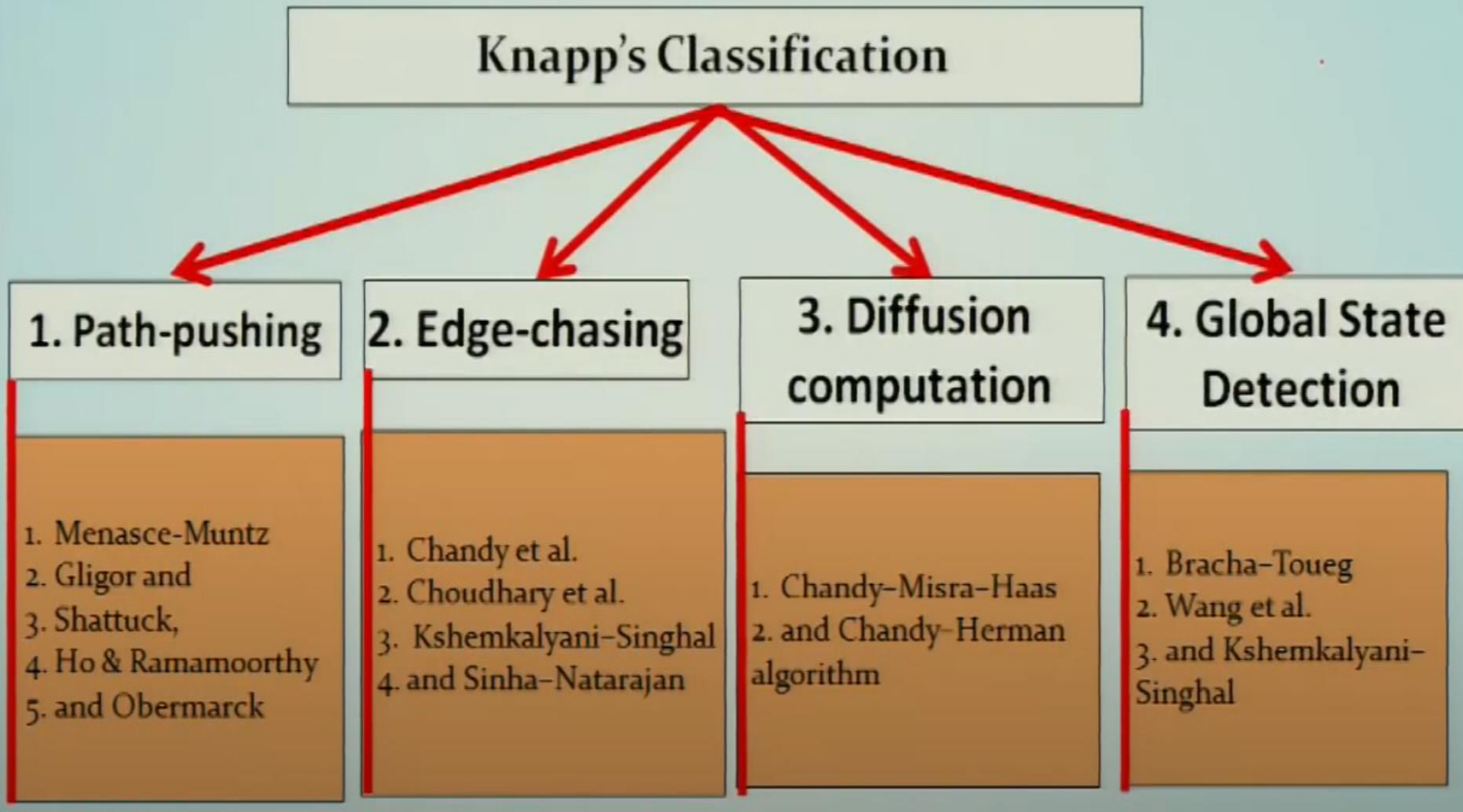
- Each site maintains 2 status tables: resource status table and process status table.
- Resource table: transactions that have locked or are waiting for resources.
- Process table: resources locked by or waited on by transactions.
- Controller periodically collects these tables from each site.
- Constructs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

# Distributed Detection Algorithm

- Responsibility is shared equally among all sites.
- Global state graph is spread over many sites.
- Several site participate in detecting deadlock
- Detection is initiated by a waiting process suspended to be part of a cycle.
- Difficult to design due to lack of globally shared memory.
- Several sites may initiate detection for the same deadlock
- Proof of correctness is difficult for these algorithms.
- Resolution is cumbersome – several sites detect same deadlock and not aware of other sites involved.

# Knapp's Classification

- **E. Knapp (1987)** has classified the Distributed deadlock detection algorithms into four classes:



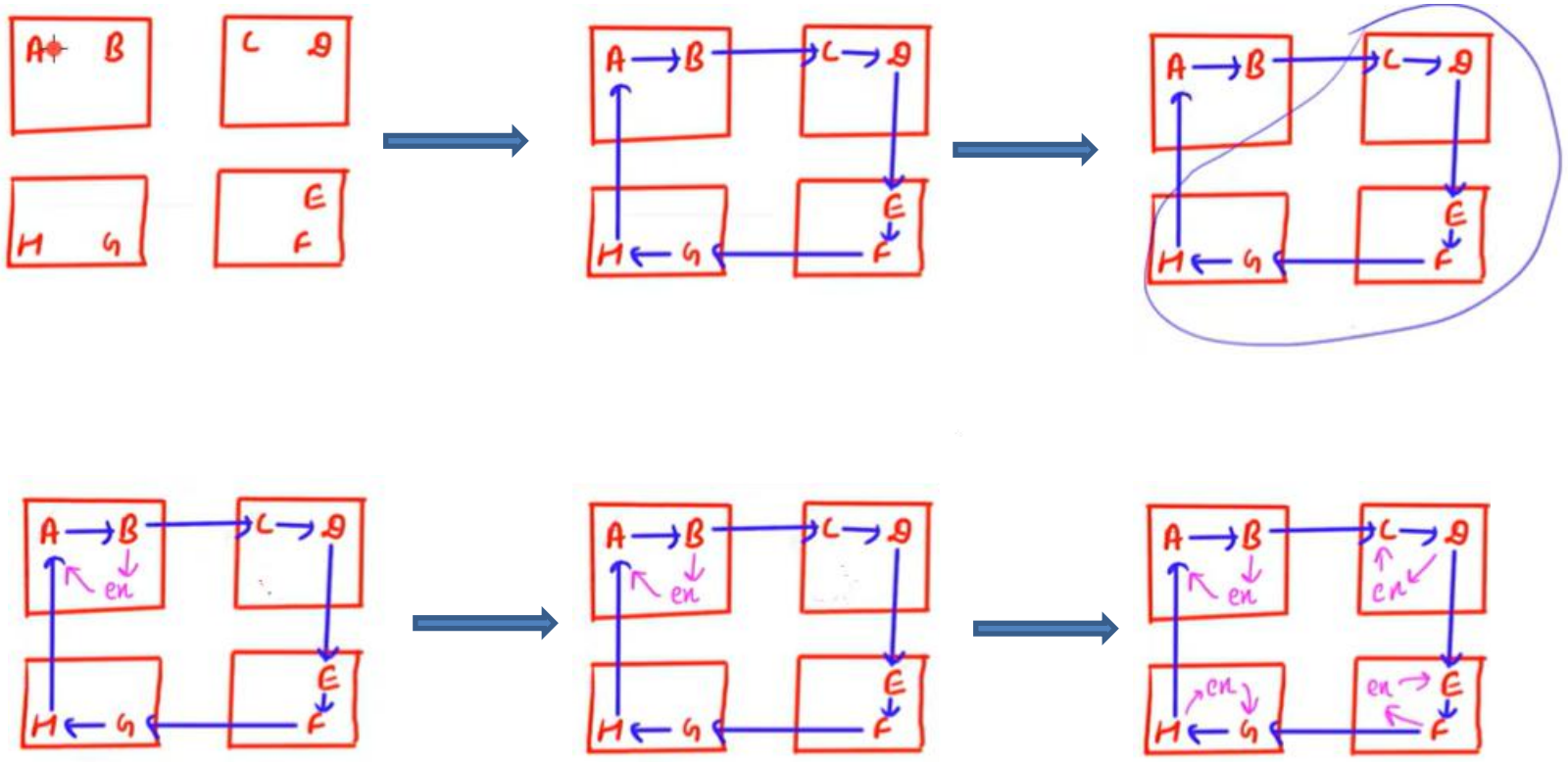
# Distributed Deadlock Detection

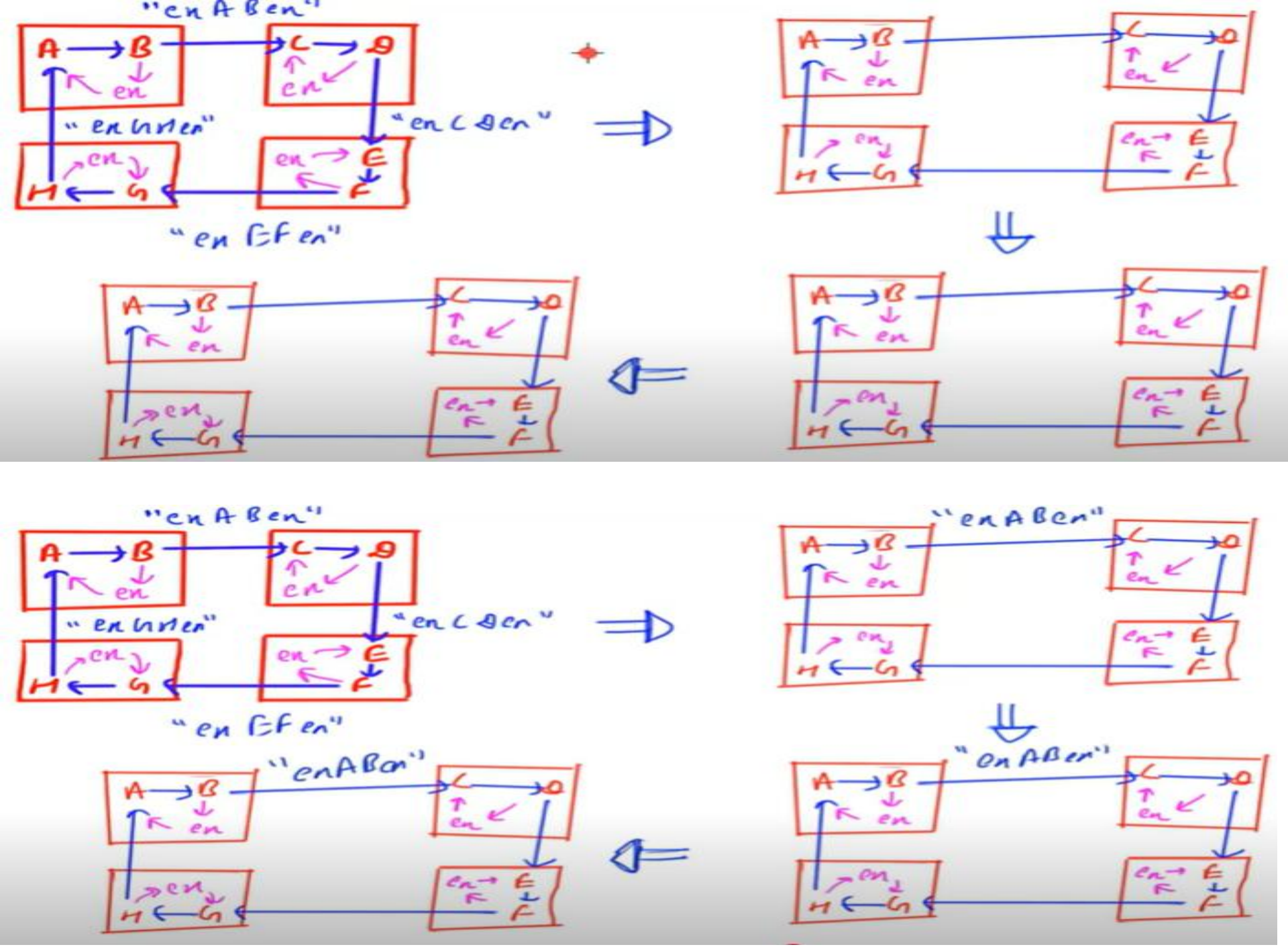
- Four common models are used in building distributed deadlock control algorithms:–
  - Path-pushing
    - ✓ path info sent from waiting node to blocking node
  - Edge-chasing
    - ✓ probe messages are sent along graph edge
  - Diffusion computation
    - ✓ echo messages are sent along graph edges
  - Global state detection
    - ✓ sweep-out, sweep-in (weighted echo messages); WFG construction and reduction

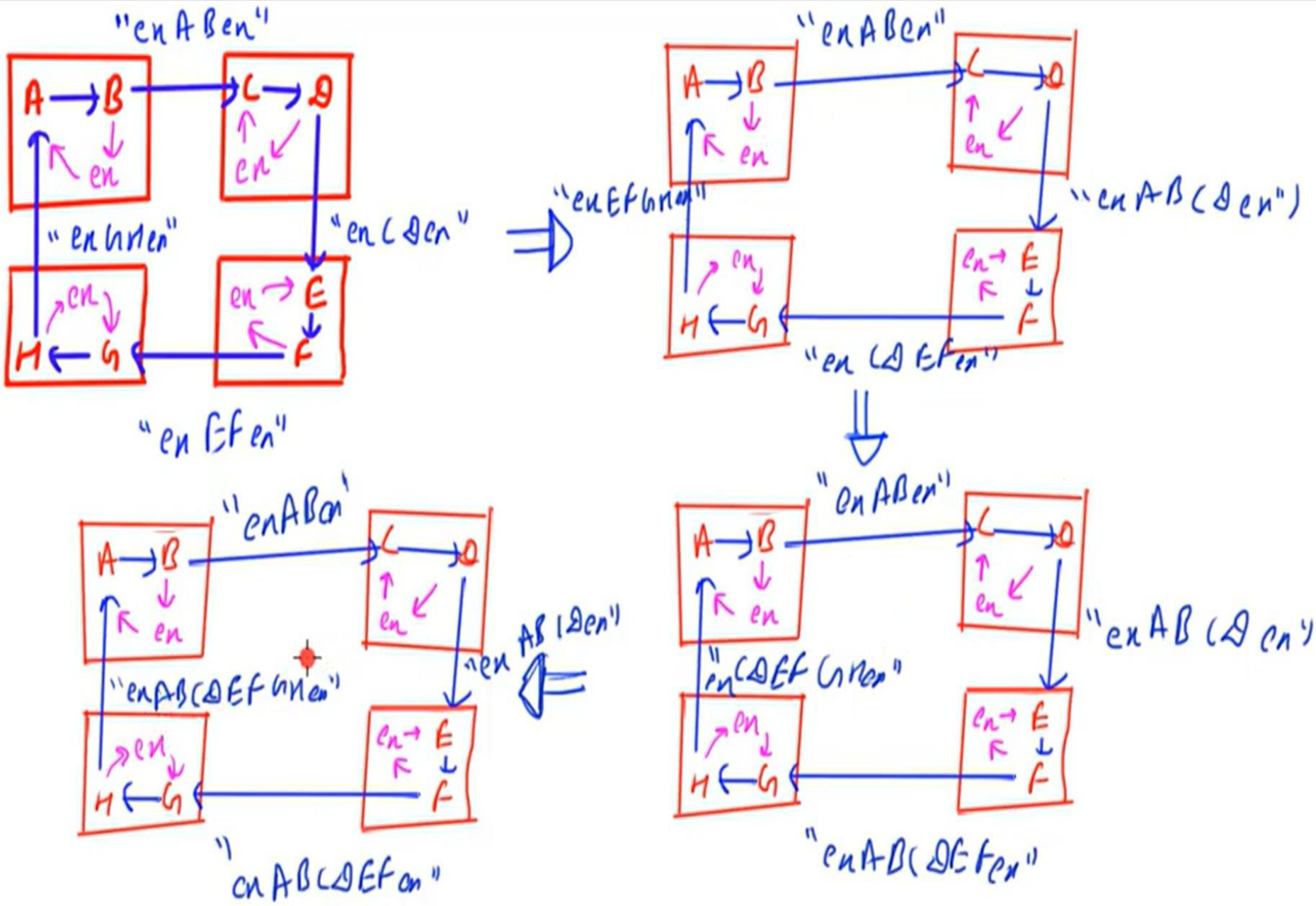
# Path-Pushing Algorithms

- In path-pushing algorithms, distributed deadlocks are detected by maintaining an explicit global WFG.
- The basic idea is to build a global WFG for each site of the distributed system.
- In this class of algorithms, at each site whenever deadlock computation is performed, it sends its local WFG to all the neighboring sites.
- After the local data structure of each site is updated, this updated WFG is then passed along to other sites, and the procedure is repeated until some site has a sufficiently complete picture of the global state to announce deadlock or to establish that no deadlocks are present.
- This feature of sending around the paths of global WFG has led to the term path-pushing algorithms.

# Path Pushing Algorithm




"en A B en"
A
B
en
C
D
en
"en C D en"
"en H G en"
H
G
en
en
E
F
"en E F en"
"en A B en"
"en A B en"
"en A B en"


"enABen"
A
B
en
C
D
en
"enCDen"
"enABCDen"
H
G
en
E
F
"enEFen"
"enABCDEFGHen"
"enABCDEFGHen"
"enABCDEFGHen"
"enABCDEFGHen"

# Edge-Chasing Algorithm

- In an edge-chasing algorithm, the presence of a cycle in a distributed graph structure is verified by propagating special messages called probes, along the edges of the graph.
- These probe messages are different than the request and reply messages.
- The formation of cycle can be detected by a site if it receives the matching probe sent by it previously.
- Whenever a process that is executing receives a probe l message, it discards this message and continues.
- Only blocked processes propagate probe messages along their outgoing edges.
- Main advantage of edge-chasing algorithms is that probes are fixed size messages which is normally very short.

# Edge Chasing Algorithms

- Chandy-Misra-Haas Algorithm
  - It uses a special message called probe.
  - A probe messages M(i, j, k) is initiated
  - Initiated by Pj for Pi and sent to Pk
  - A probe message travels along the edges of the global TWF graph,
  - And deadlock is discovered when a probe message returns to its initiating process.

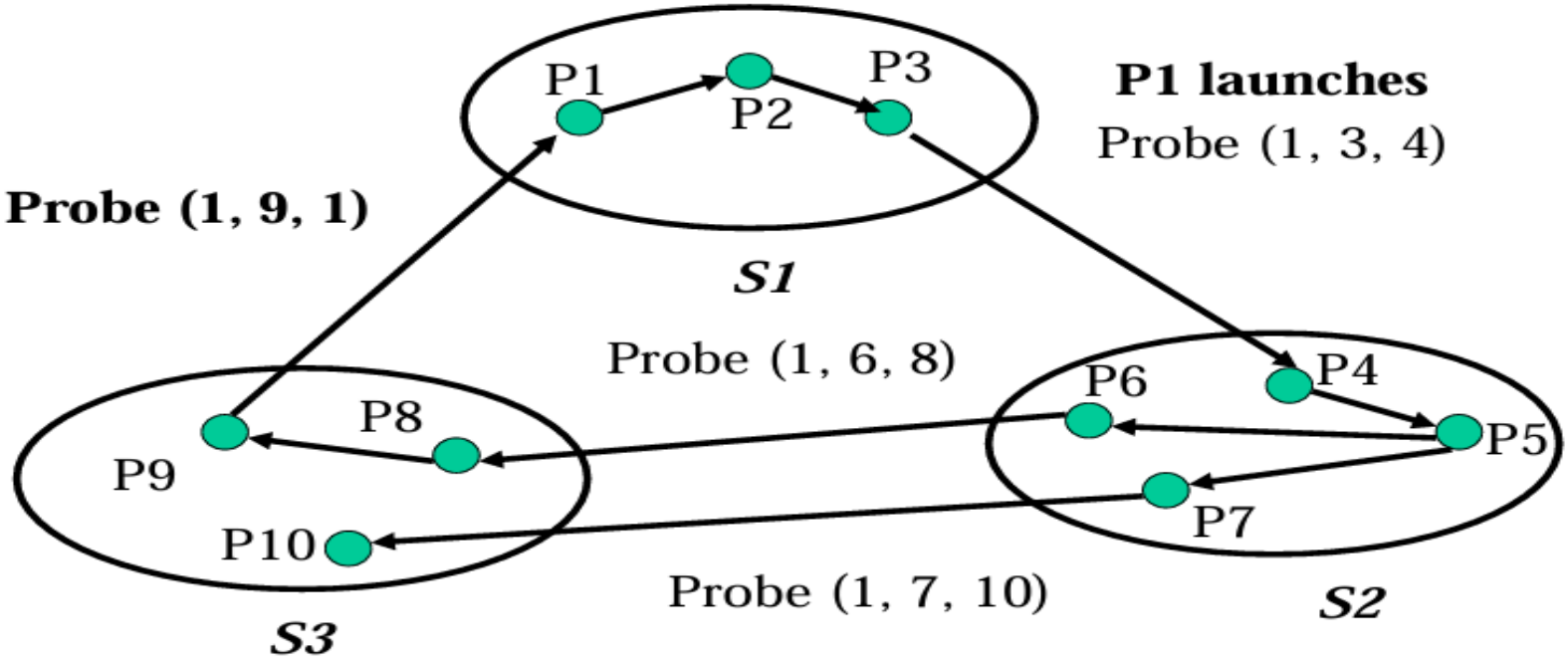


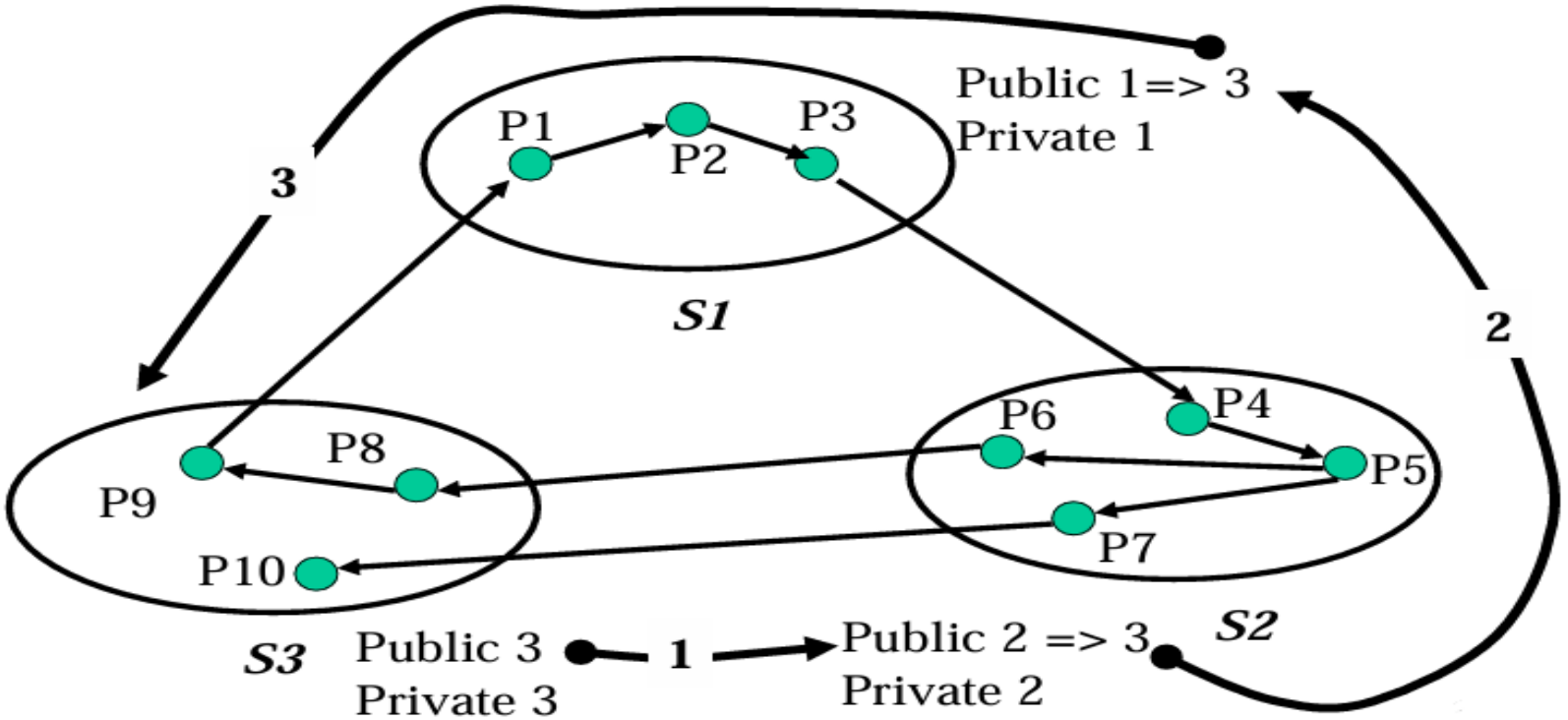

Fig: Chandy-Misra-Haas Algorithm

- If process PI initiates deadlock detection, it sends probe (I, 3,4) to the controller C2 at site S2.
- Since P6 is waiting for P8 and P7 is waiting for P10.
- C2 sends probes (1, 6, 8) and (1, 7, 10) to C3, which in turn sends probe (1, 9,1) to C1
- On receipt of probe (1, 9, 1), C1 declares that
- P1 is deadlocked.

# Mitchell-Merit algorithm

- Propagates messages in the reverse direction.
- Use public-private labeling of messsages.
- Messages may replace their labels  at each site.
- When a message arrives at a site with a matching public label, a deadlock is detected (by only the process with the largest public label in the cycle) which normally does by self-destruct

- Eg:
- P6 initially asks P8 for its public label and changes its own 2 to 3.
- P3 asks P4 and changes its public label 1 to 3.
- P9 asks P1 and finds its own public label 3 and thus detects the deadlock
- P1 -> P2 -> P3 -> P4 -> P5 -> P6 -> P8 -> P9 -> P1

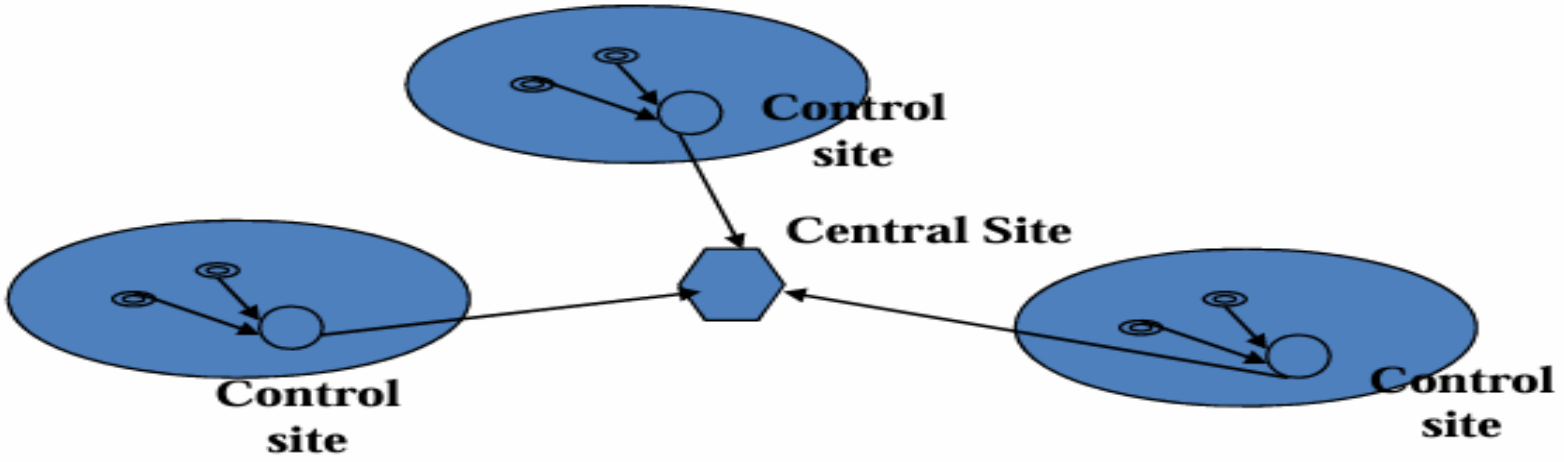


# Hierarchical control

- Sites are arranged in hierarchical fashion
- Sites detects deadlock involving only its descendant sites.
- Get the best of both Centralized and Decentralized control.
- Requires special care while arranging the sites in hierarchical order.
- Objective is defeated if deadlock span several clusters

# Hierarchical Deadlock Detection

- These algorithms represent a middle ground between fully centralized and fully distributed.
- 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.



# The Ho-Ramamoorthy Algorithm

- Uses only 2 levels
  - Master control node
  - Cluster control nodes
- Cluster control nodes are responsible for detecting deadlock among their members and reporting dependencies outside their cluster to the Master control node (they use the one phase version of the Ho-Ramamoorthy algorithm discussed earlier for centralized detection)
- The Master control node is responsible for detecting intercluster deadlocks
- Node assignment to clusters is dynamic

- Uses **only two levels**:
- **Master Control Node** (detects deadlocks **between clusters**).
- **Cluster Control Nodes** (detect **intra-cluster** deadlocks using **one-phase centralized detection**).
- **Cluster assignments are dynamic**, meaning nodes can be reassigned for load balancing.