Deadlock

- Deadlock is a situation where a process or a set of processes is blocked on an event that never occurs
- Processes while holding some resources may request for additional allocation of resources which are held by other processes
- Processes are in circular wait for the resources

Deadlock vs Starvation

- **Starvation** occurs when a process waits for a resource that becomes available continuously but is not allocated to a process
- **Two Main Differences**

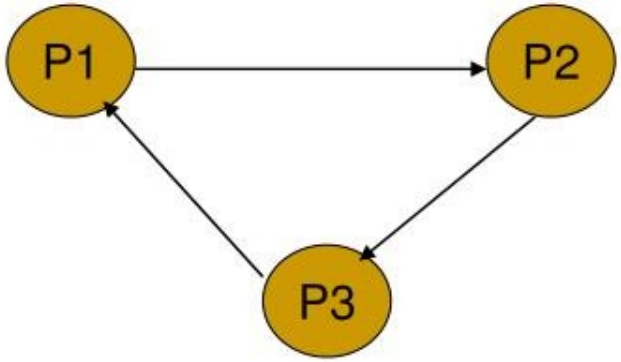
- In starvation it is not certain that a process will ever get the requested resource where as a deadlocked process is permanently blocked because required resource never become available
- In starvation the resource under contention is in continuation use where as this is not true in case of deadlock

Necessary Conditions for Deadlock

- Exclusive access
- Wait while hold
- No Preemption
- Circular wait

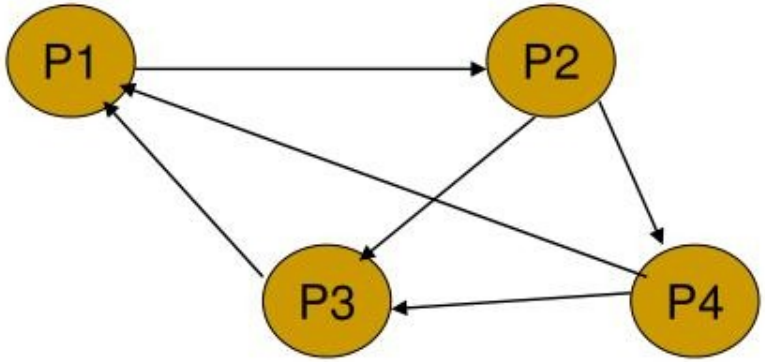
Models of Deadlock

- **Single-Unit Request Model**
- Process is restricted to request only one resource at a time
- Outdegree in WFG is one
- Cycle in WFG means deadlock



Models of Deadlock .....

- **AND Request Model**
- Process can simultaneously request multiple resources
- Process Remain blocked until all the resources are granted
- Outdegree of WFG can be more than 1
- Cycle in WFG means system is deadlocked
- Process can be involved in more than one deadlock



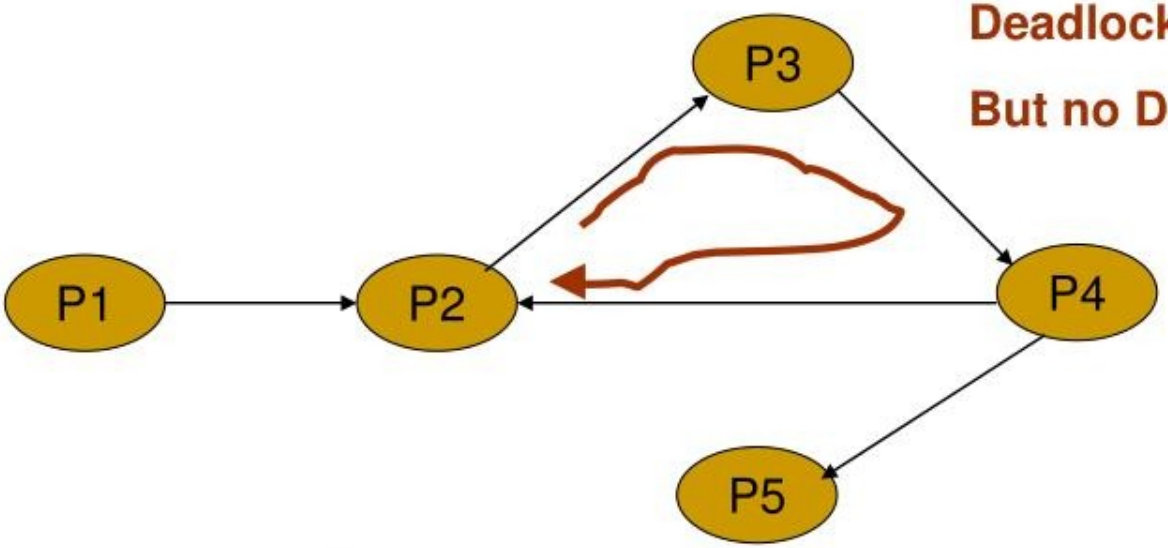
Models of Deadlock .....

- **OR Request Model**
- Process can simultaneously request multiple resources
- Process Remain blocked until it is granted any of the requested resources
- Outdegree of WFG can be more than 1
- **Cycle in WFG is not a sufficient condition for the deadlock**
- **Knot in the WFG is a sufficient condition for deadlock**
- Knot is a subset of graph such that starting from any node in the subset it is impossible to leave the knot by following the edges of the graph

Cycle vs Knot

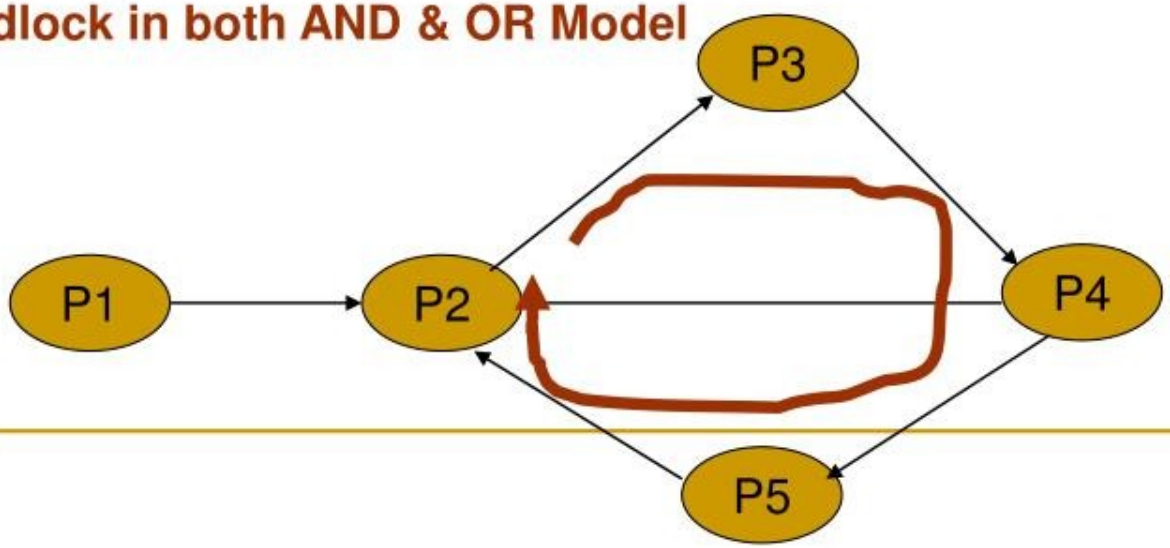
Deadlock in AND Model

But no Deadlock in OR Model



Cycle but no Knot

Deadlock in both AND & OR Model



Cycle & Knot

Resources

- Reusable (CPU, Main-memory, I/O Devices)
- Consumable (Messages, Interrupt Signals

Distributed Deadlock Detection

- **Assumptions:**
  a. **System has only reusable resources**
  b. **Only exclusive access to resources**
  c. **Only one copy of each resource**
  d. **States of a process: running or blocked**
  e. **Running state: process has all the resources**
  f. **Blocked state: waiting on one or more resource**

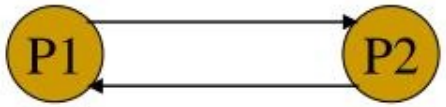
Resource vs Communication Deadlocks

- **Resource Deadlocks**
  - A process needs multiple resources for an activity.
  - Deadlock occurs if each process in a set request resources held by another process in the same set, and it must receive all the requested resources to move further.

- **Communication Deadlocks**
  - Processes wait to communicate with other processes in a set.
  - Each process in the set is waiting on another process's message, and no process in the set initiates a message until it receives a message for which it is waiting.

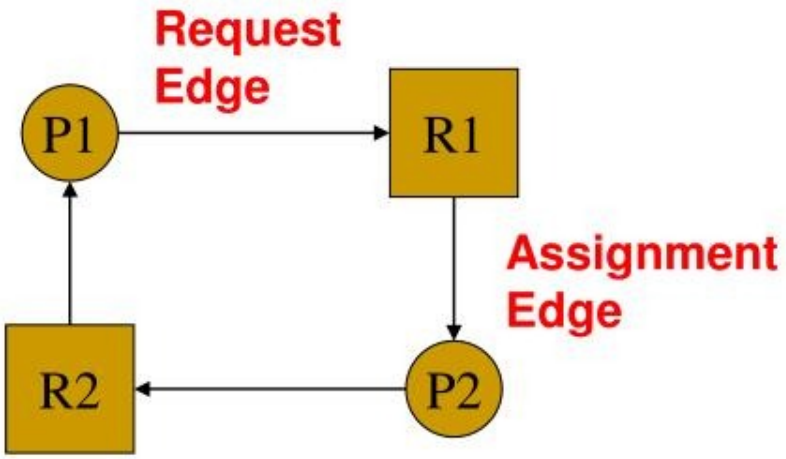
Graph Models

- Nodes of a graph are processes. Edges of a graph the pending requests or assignment of resources.
- Wait-for Graphs (WFG): P1 -> P2 implies P1 is waiting for a resource from P2.
- Transaction-wait-for Graphs (TWF): WFG in databases.
- Deadlock: directed cycle in the graph.
- Cycle example:



Graph Models

- Wait-for Graphs (WFG): P1 -> P2 implies P1 is waiting for a resource from P2.



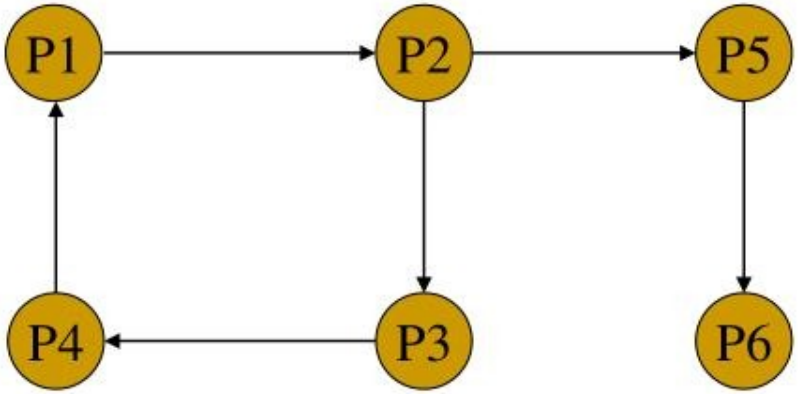
AND, OR Models

- **AND Model**
  - A process/transaction can simultaneously request for multiple resources.
  - Remains blocked until it is granted *all* of the requested resources.

- **OR Model**
  - A process/transaction can simultaneously request for multiple resources.
  - Remains blocked till *any one* of the requested resource is granted.

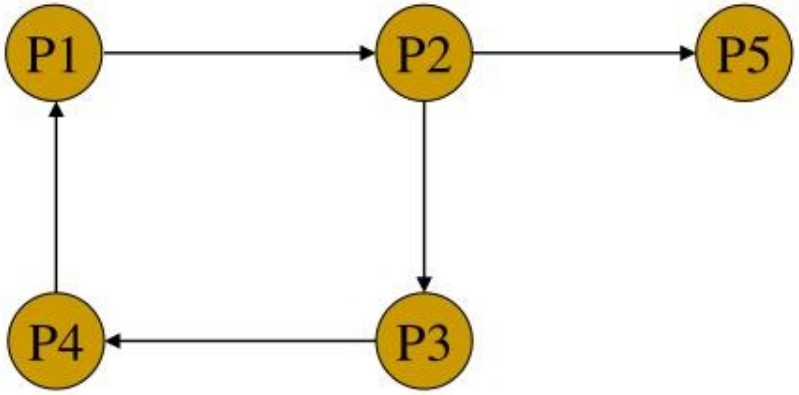
Sufficient Condition

Deadlock ??



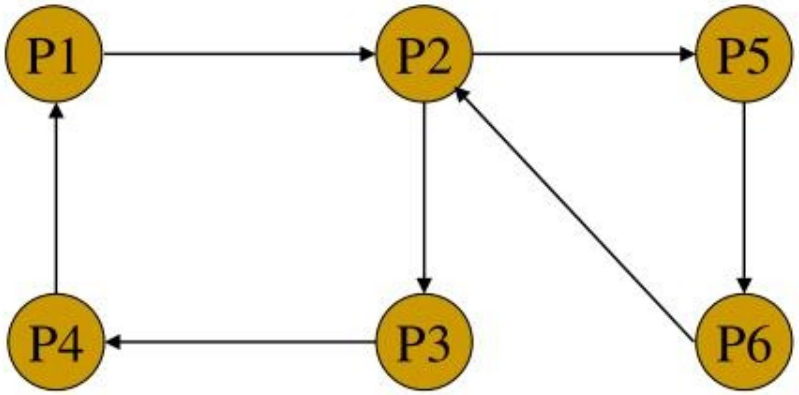
AND, OR Models

- **AND Model**
  - **Presence of a cycle.**



AND, OR Models

- **OR Model**
  - Presence of a knot.
  - Knot: Subset of a graph such that starting from any node in the subset, it is impossible to leave the knot by following the edges of the graph.



Deadlock Handling Strategies

- **Deadlock Prevention:** difficult
- **Deadlock Avoidance:** before allocation, check for possible deadlocks.
  - Difficult as it needs global state info in each site (that handles resources).
- **Deadlock Detection:** Find cycles. Focus of discussion.
- **Deadlock detection algorithms must satisfy 2 conditions:**
  - No undetected deadlocks.
  - No false deadlocks.

Distributed Deadlocks

- **Centralized Control**
  - A *control site* constructs wait-for graphs (WFGs) and checks for directed cycles.
  - WFG can be maintained continuously (or) built on-demand by requesting WFGs from individual sites.
- **Distributed Control**
  - WFG is spread over different sites.Any site can initiate the deadlock detection process.
- **Hierarchical Control**
  - Sites are arranged in a hierarchy.
  - A site checks for cycles only in descendents.

Centralized Algorithms

- **Ho-Ramamurthy 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 Algorithms

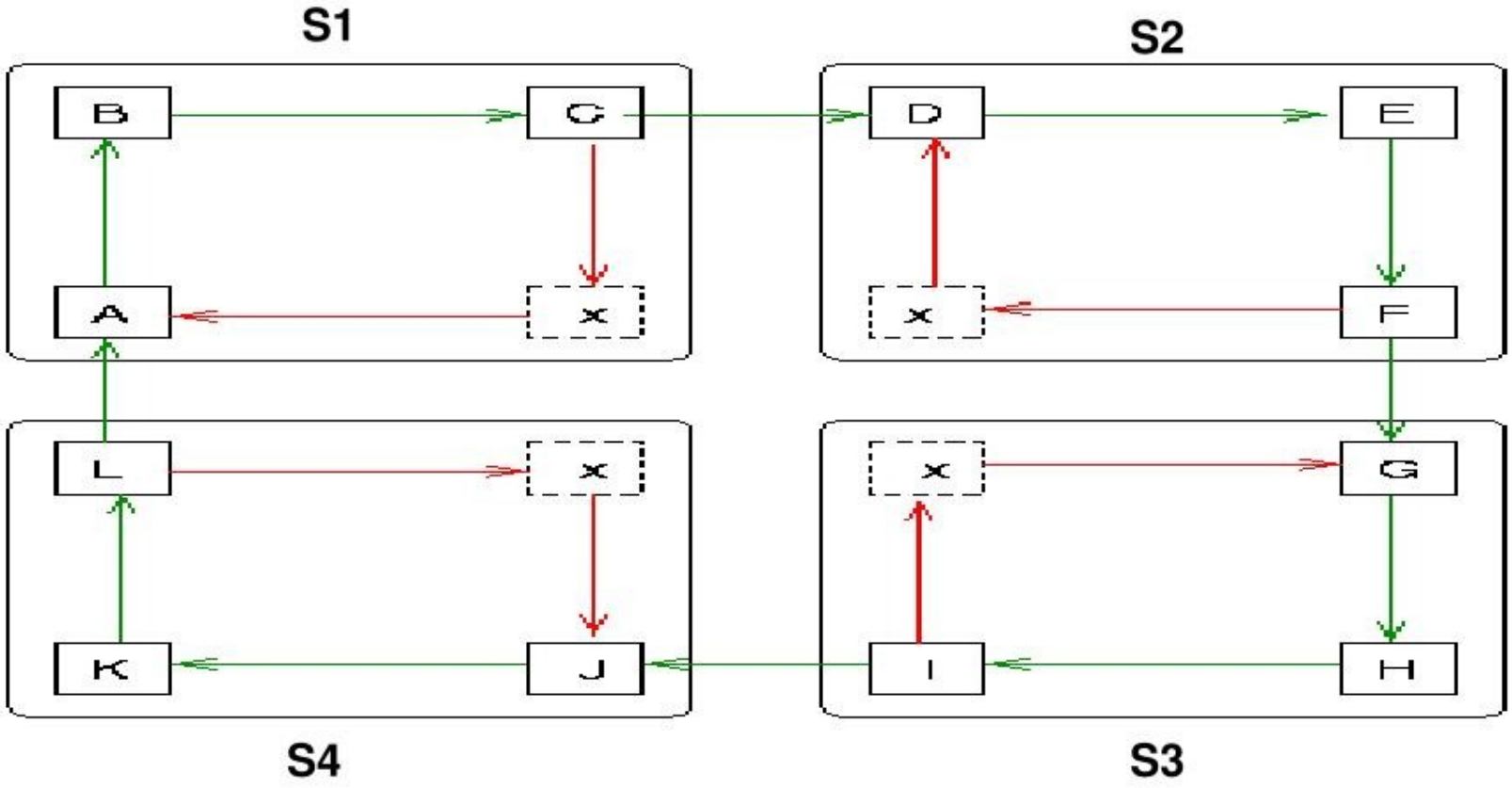
- **Path-pushing**: resource dependency information disseminated through designated paths (in the graph) [Examples : Menasce-Muntz & Obermarck]
- **Edge-chasing**: special messages or probes circulated along edges of WFG. Deadlock exists if the probe is received back by the initiator. [Examples :CMH for AND Model , Sinha-Natarajan]
- **Diffusion computation**: queries on status sent to process in WFG. [Examples :CMH for OR Model, Chandy-Herman]
- **Global state detection**: get a snapshot of the distributed system. [Examples :Bracha-Toueg,Kshemkalyani-Singhal]

Path-pushing

- **Obermarck's Algorithm (AND model)**
  - Path Propagation Based Algorithm
  - Based on a database model using transaction processing
  - Sites which detect a cycle in their partial WFG views convey the paths discovered to members of the (totally ordered) transaction
  - Algorithm can detect ***phantoms*** due to its asynchronous snapshot method

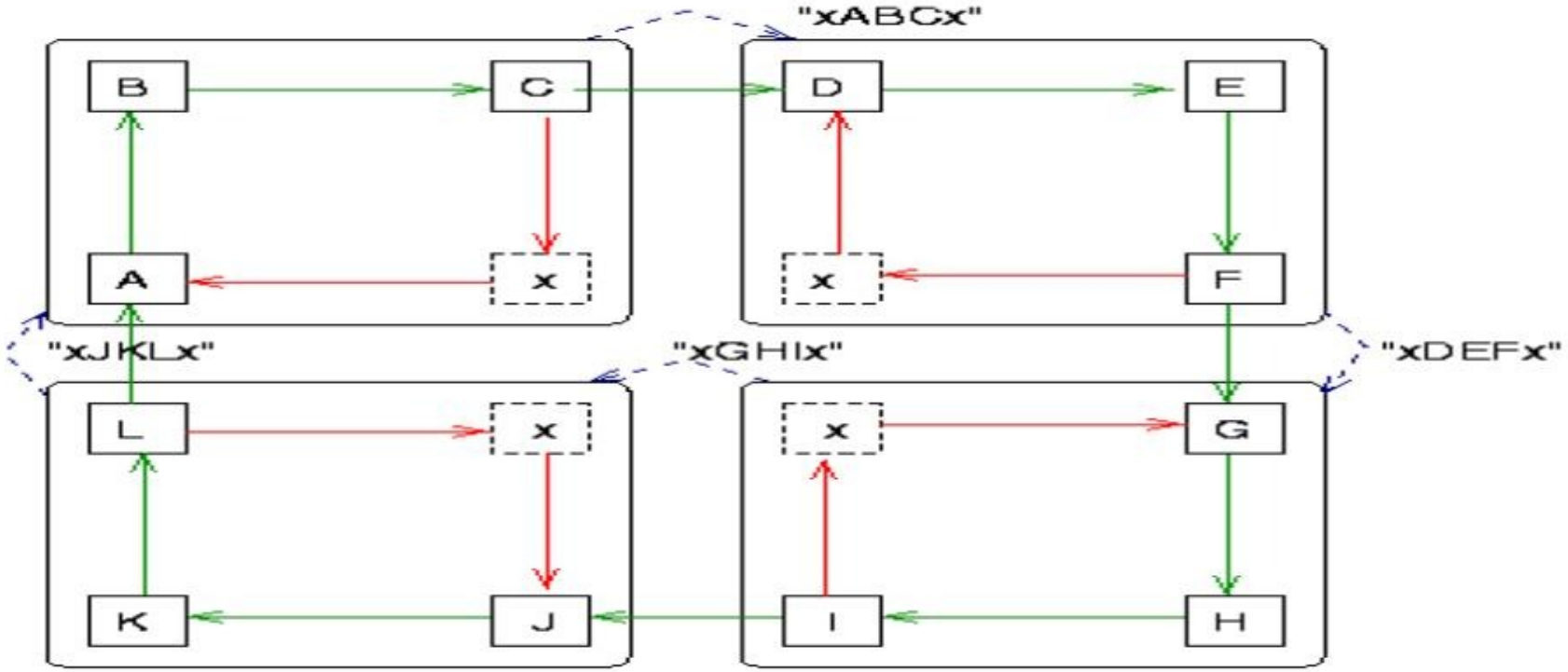
Obermark's Algorithm Example

Intial State



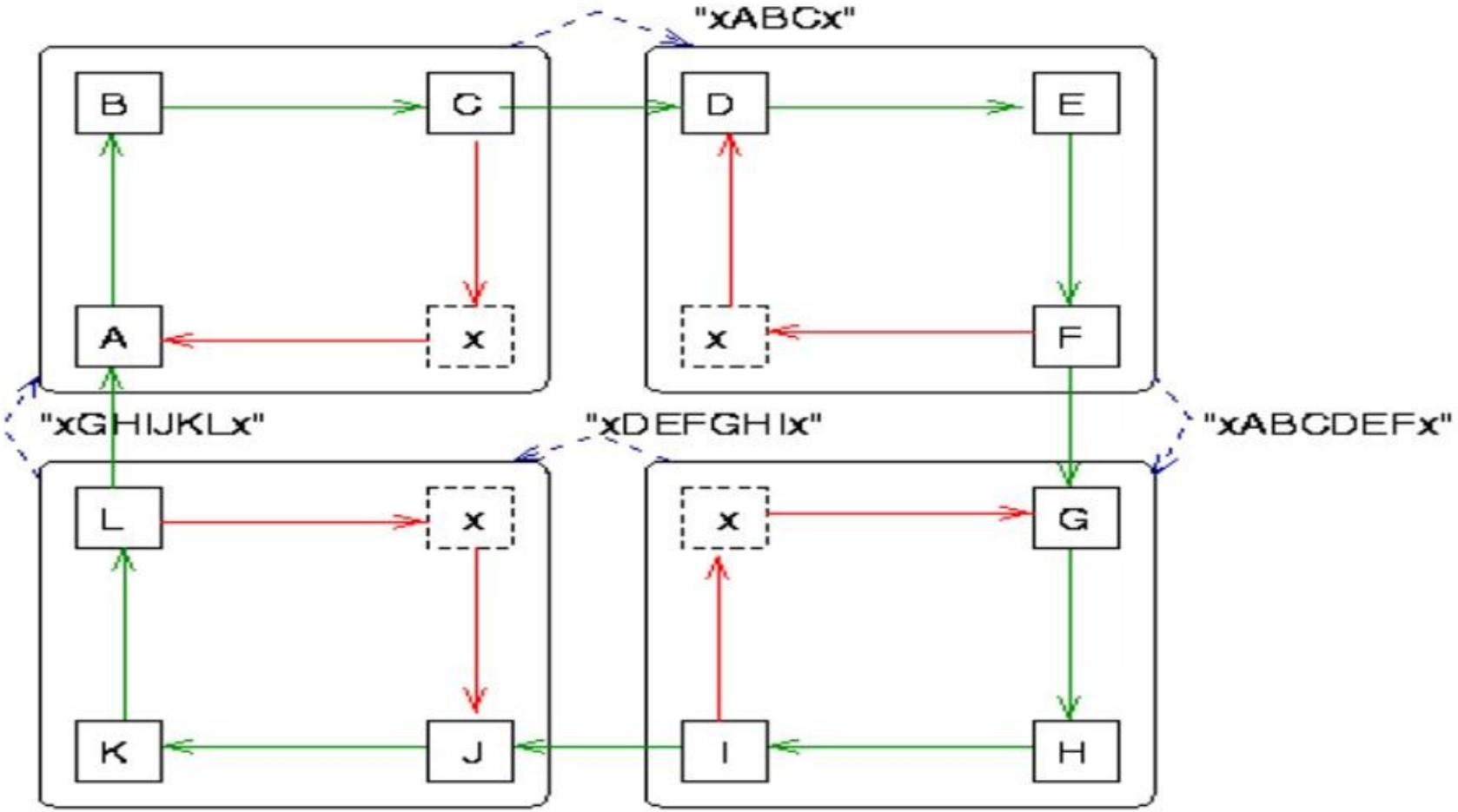
Obermark's Algorithm Example

Iteration 1



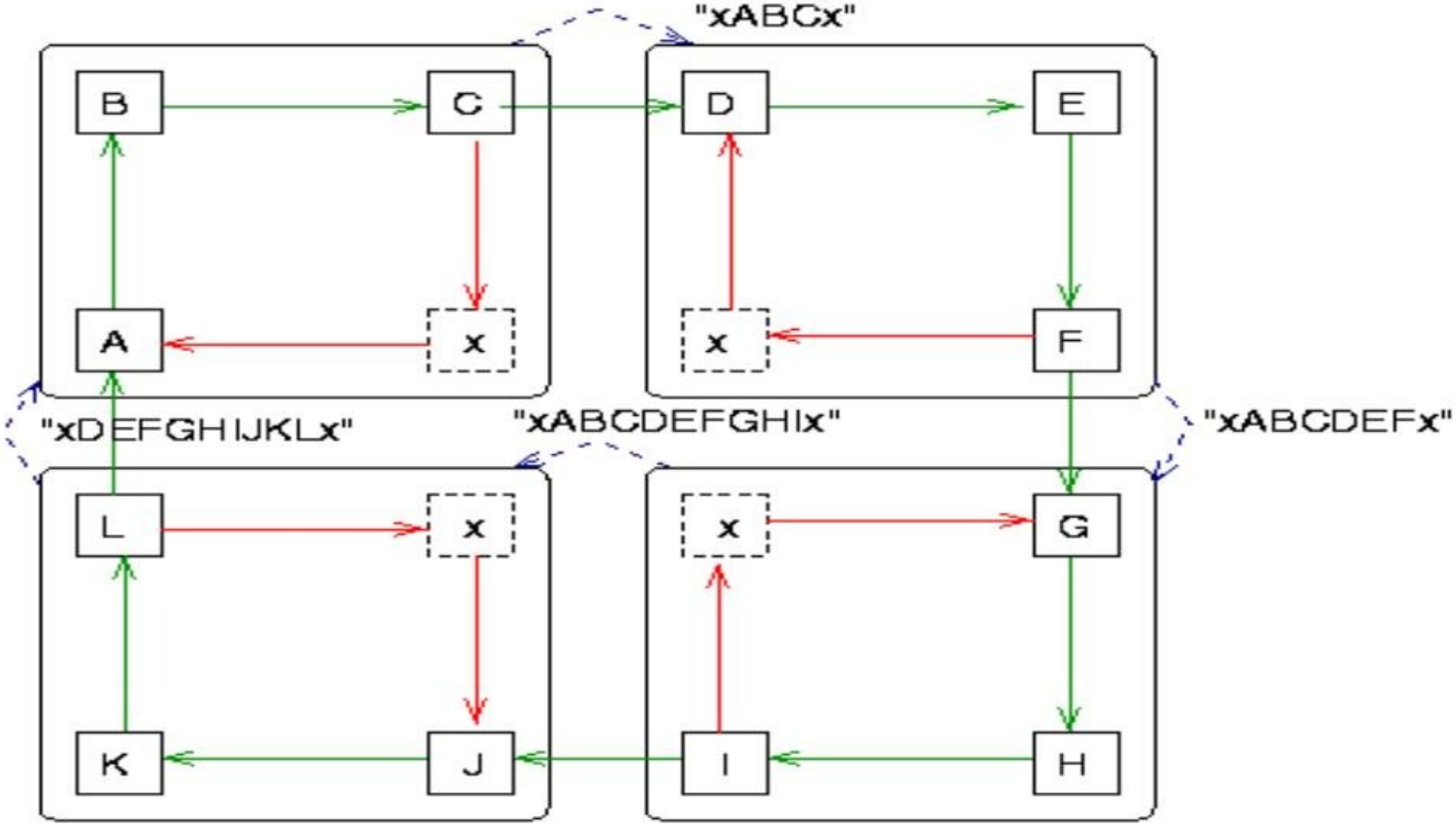
Obermark's Algorithm Example

Iteration 2



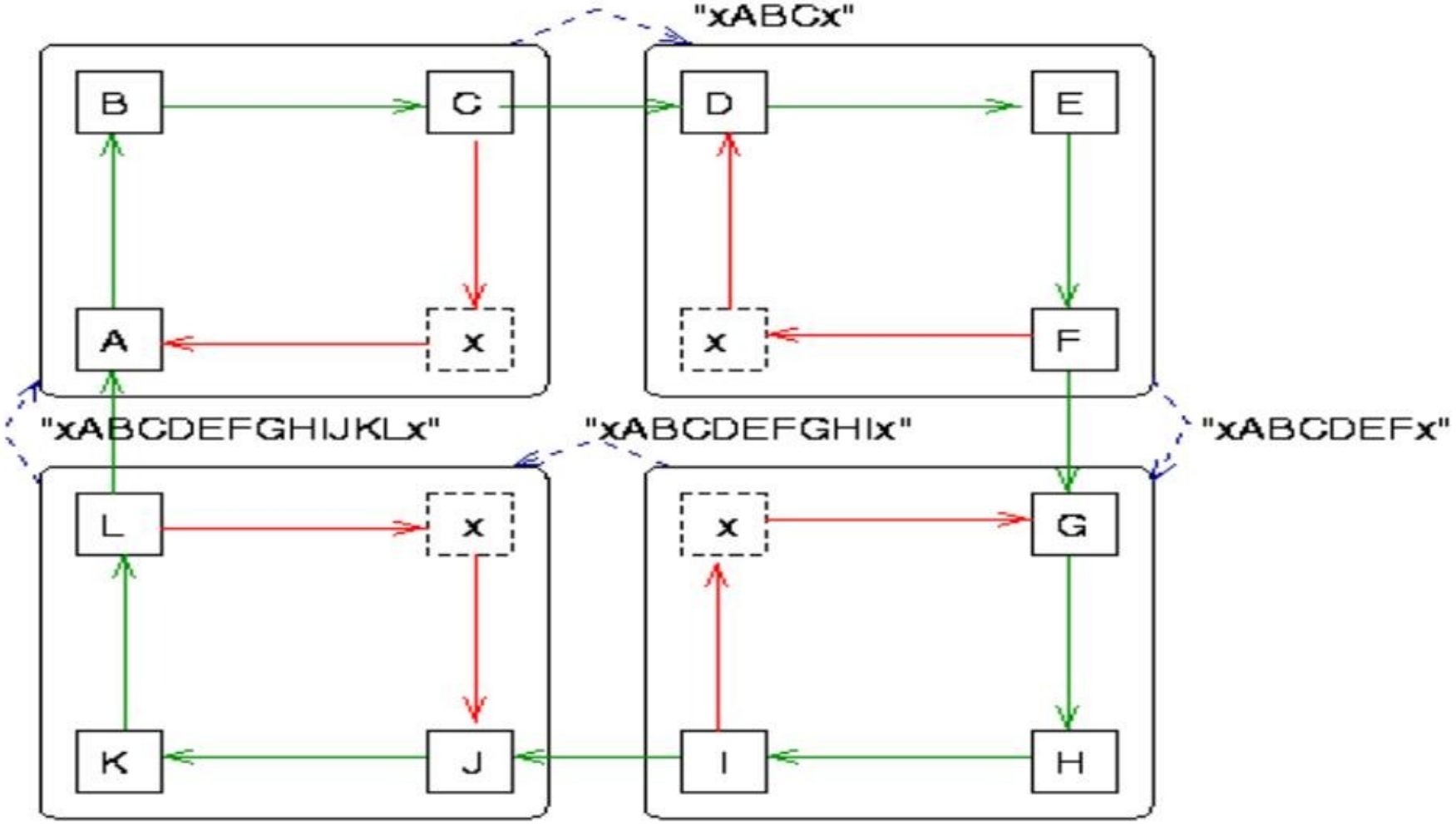
Obermark's Algorithm Example

Iteration 3



Obermark's Algorithm Example

Iteration 4



Edge-Chasing Algorithm

- **Chandy-Misra-Haas's Algorithm (AND MODEL)**:
  - 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.

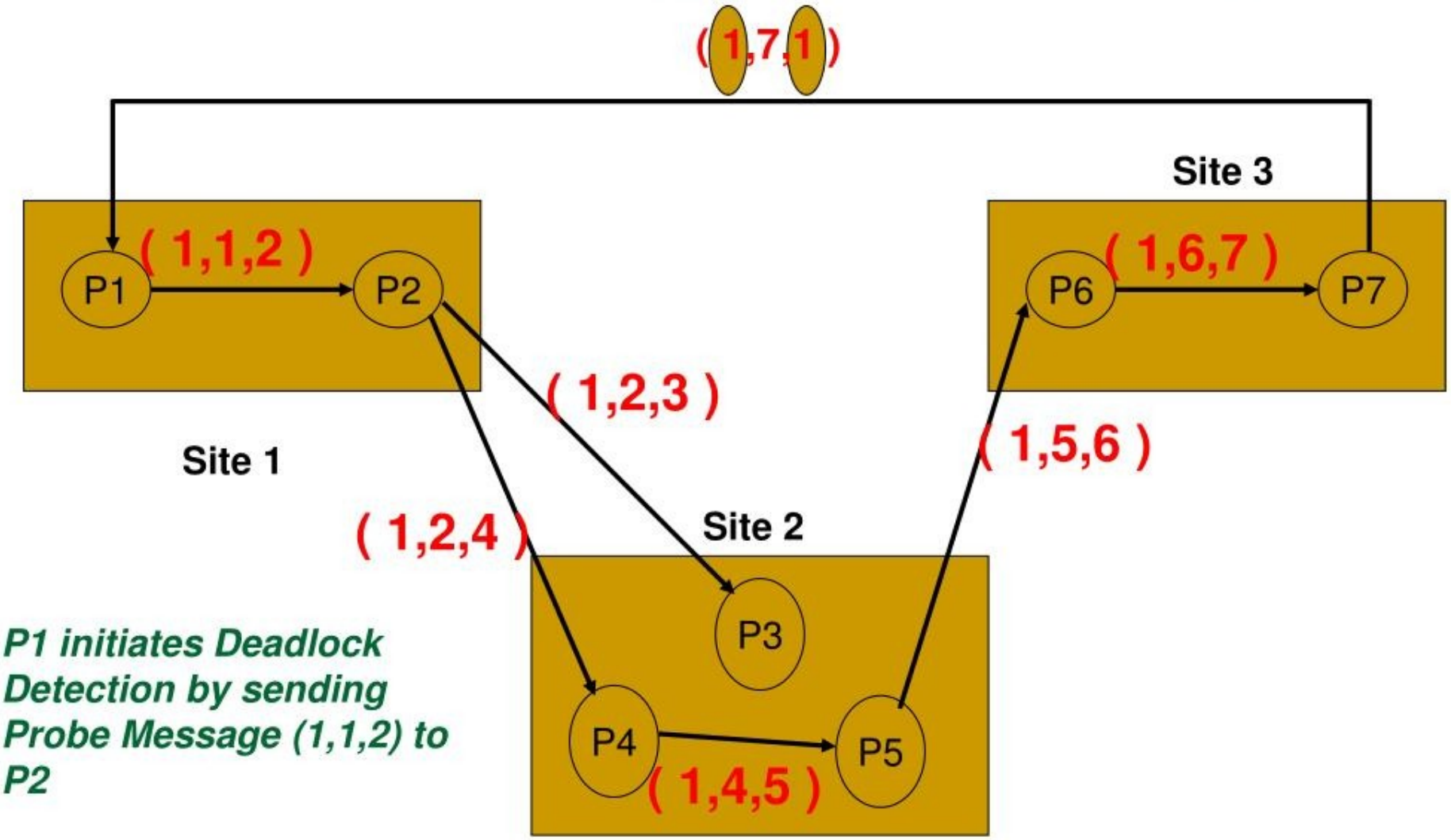
Performance:

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).

C-M-H Algorithm: Example



P1 initiates Deadlock Detection by sending Probe Message (1,1,2) to P2

Diffusion-based Algorithm
CMH Algorithm for OR Model

Initiation by a blocked process Pi:

send query(i,i,j) to all processes Pj in the dependent set DSi of Pi;
num(i) := |DSi|; waiti(i) := true;

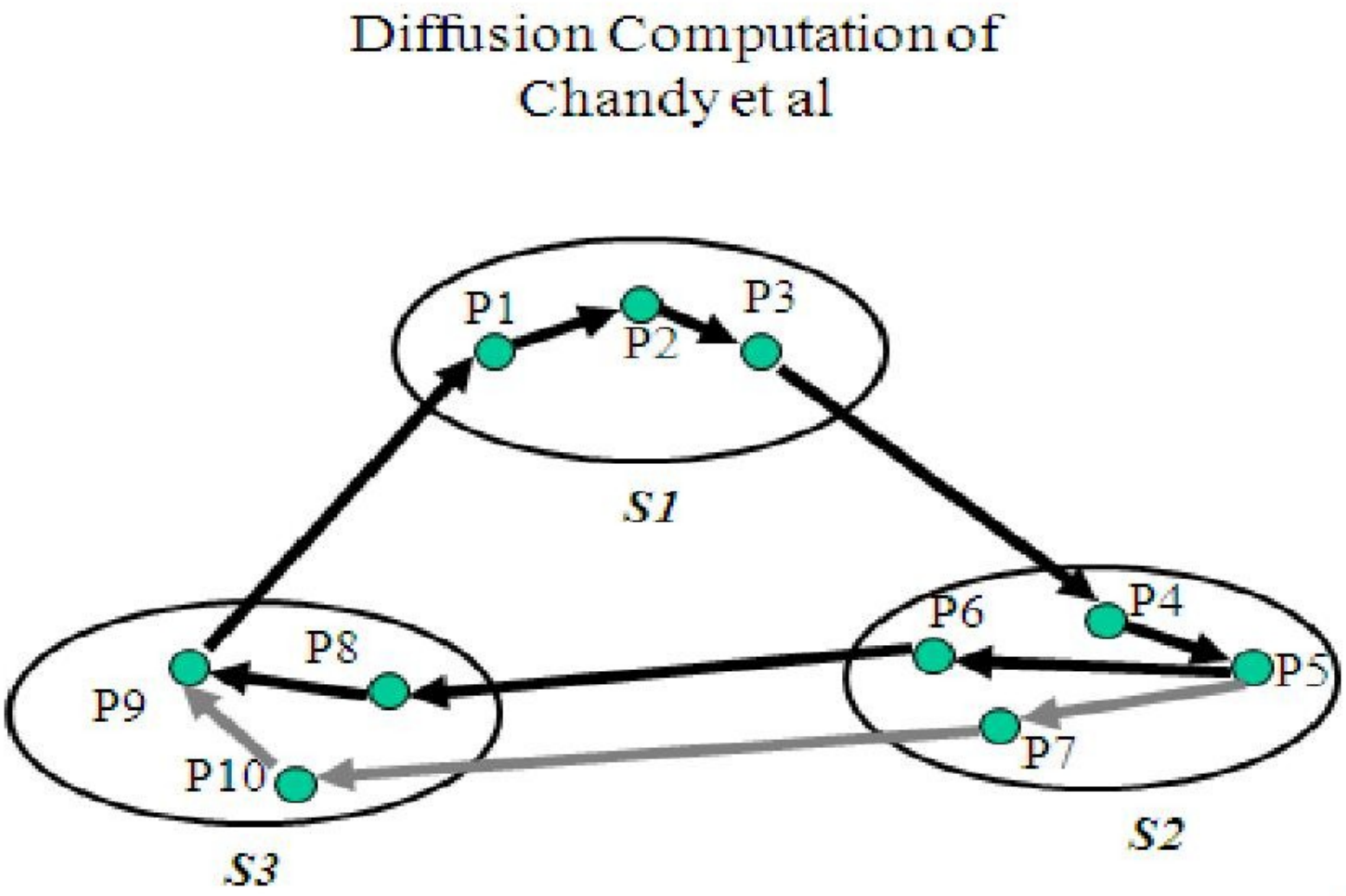
Blocked process Pk receiving query(i,j,k):

if this is *engaging* query for process Pk /* first query from Pi */
    then send query(i,k,m) to all Pm in DSk;
    numk(i) := |DSk|; waitk(i) := true;
else if waitk(i) then send a reply(i,k,j) to Pj.

Process Pk receiving reply(i,j,k)

if waitk(i) then
    numk(i) := numk(i) - 1;
    if numk(i) = 0 then
        if i = k then declare a deadlock.
        else send reply(i, k, m) to Pm, which sent the engaging query.

Diffusion Algorithm: Example



P1 => P2 **message at P2 from P1** (P1, P1, P2)

P2 => P3 **message at P3 from P2** (P1, P2, P3)

P3 => P4 **message at P4 from P3** (P1, P3, P4)

P4 => P5 **ETC.**

P5 => P6

P5 => P7

P6 => P8

P7 => P10

end condition

P8 => P9 (P1, P8, P9), **now reply (P1, P9, P1)**

P10 => P9 (P1, P10, P9), **now reply (P1, P9, P1)**

P8 <= P9 **reply (P1, P9, P8)**

P10<= P9 **reply (P1, P9, P10)**

P6 <= P8 **reply (P1, P8, P6)**

P7 <= P10 **reply (P1, P10, P7)**

P5 <= P6 **ETC.**

P5 <= P7

P4 <= P5

P5 cannot reply until both P6 and P7 replies arrive !

P3 <= P4

P2 <= P3

P1 <= P2 **reply (P1, P2, P1)**

deadlock condition

Engaging Query

- **How to distinguish an engaging query?**
  - query(i,j,k) from the initiator contains a unique sequence number for the query apart from the tuple (i,j,k).
  - This sequence number is used to identify subsequent queries.
  - (e.g.,) when query(1,7,1) is received by P1 from P7, P1 checks the sequence number along with the tuple.
  - P1 understands that the query was initiated by itself and it is not an engaging query.
  - Hence, P1 sends a reply back to P7 instead of forwarding the query on all its outgoing links.

Mitchell-Merritt Algorithm (Edge-Chasing Category)

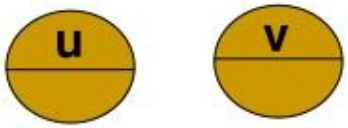
- Each Node has two labels : Public & Private
- Private Label is unique to node but may change
- Initially both private and public label values are same
- Guarantees that only one process will detect the deadlock
- Process/Node/Site responsible for deadlock detection propagates public label in reverse direction
- When a blocked transaction reads the public label of waiting upon process it changes its public label if its own public label value is less than read value.
- When a initiator process reads the message with public label equals to its own then deadlock is detected.

Mitchell-Merritt Algorithm

The algorithm exhibits 4 nondeterministic state transitions

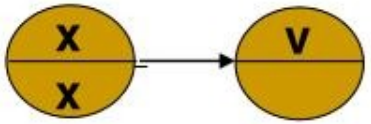
1. Block State

State Before



Outdegree =0

State After

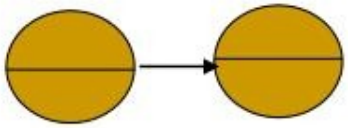


Value x should be computed as per function inc(u,v) i.e. any value which is larger than both u,v

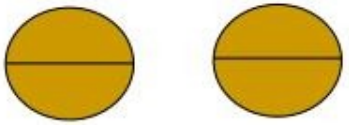
1. This block step occurs when a process begins to wait on some resource held by other [Creates an edge in WFG]
2. Label change occurs in this step for waiting process
3. Both public and private labels of the waiting process are increased to a value greater than their previous values & greater than the public label of the process being waited on.

2. Activate

State Before



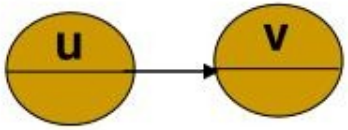
State After



- Earlier there is an edge in the before state, but there will be no edge in the after state
- Edge disappeared [Either process may be allocated resource, or timed out or owner of the resource may have changed]

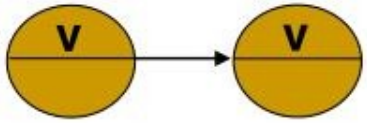
3. Transmit State

State Before



If u < v

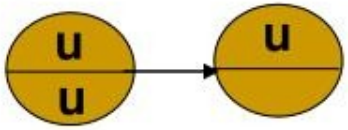
State After



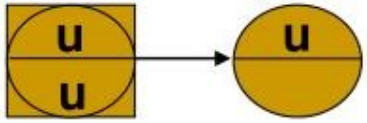
1. When a waiting process reads public variable of waiting upon process
2. If the public label of waiting process is smaller than the public label of the process upon whom it is waiting, then waiting process will change its public label equal to the public label of the process upon whom it is waiting.
3. Waiting process's private label remains unchanged

4. Detect State

State Before

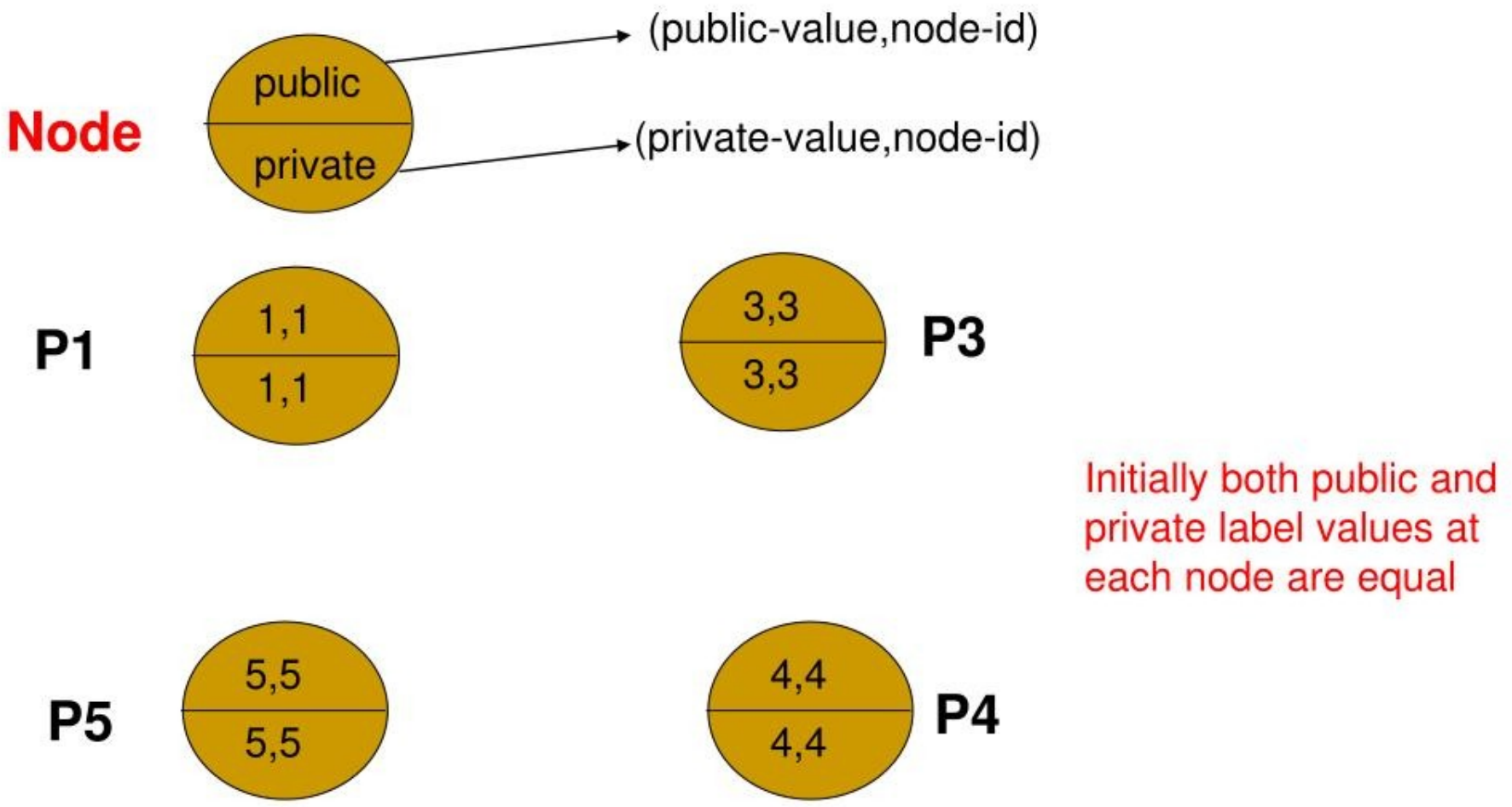


State After



1. When a process sees its own public label comes back to itself
2. When a process reads a public label of the waiting upon process and finds that the public label value of waiting upon process is equals to its own public label value then it determines that a cycle exists and declares deadlock

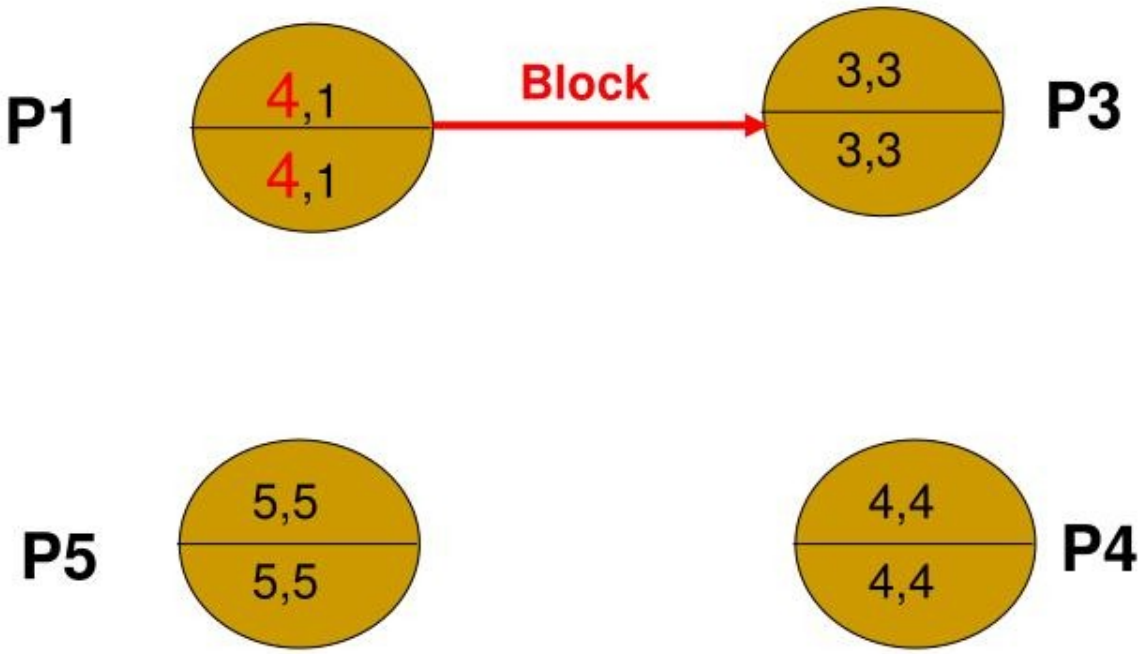
Mitchell-Merritt Algorithm Example



Mitchell-Merritt Algorithm Example cont…

Now suppose P1 is waiting for P3 (P1 → P3)

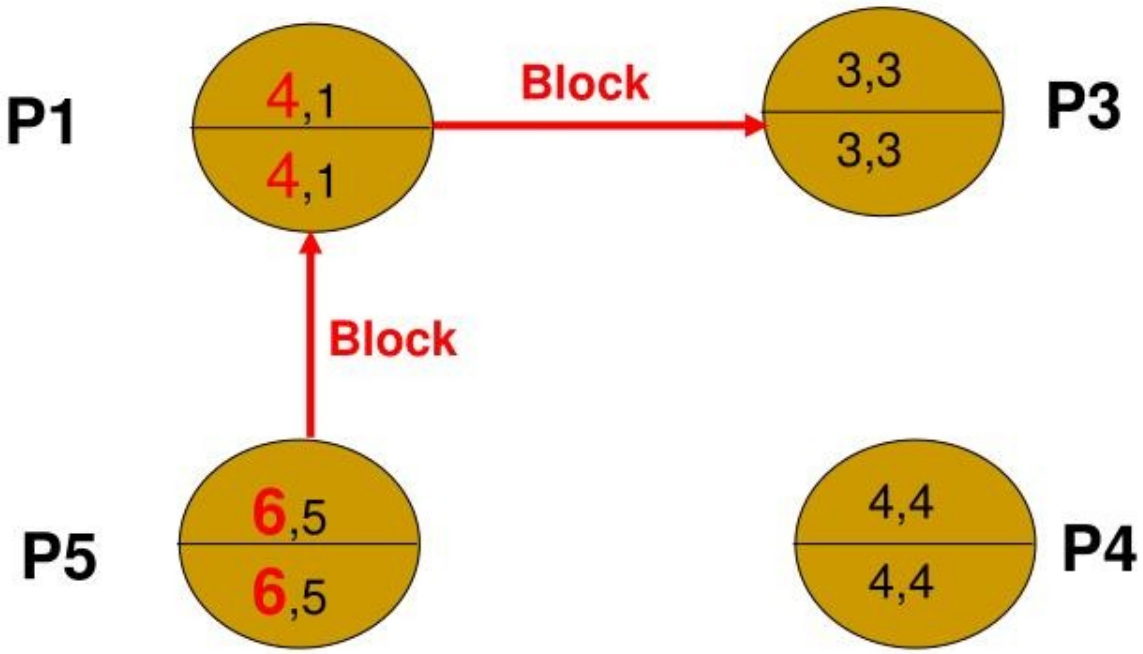
Block state will occur for P1



Mitchell-Merritt Algorithm Example cont…

Now suppose P5 is waiting for P1 (P5 → P1)

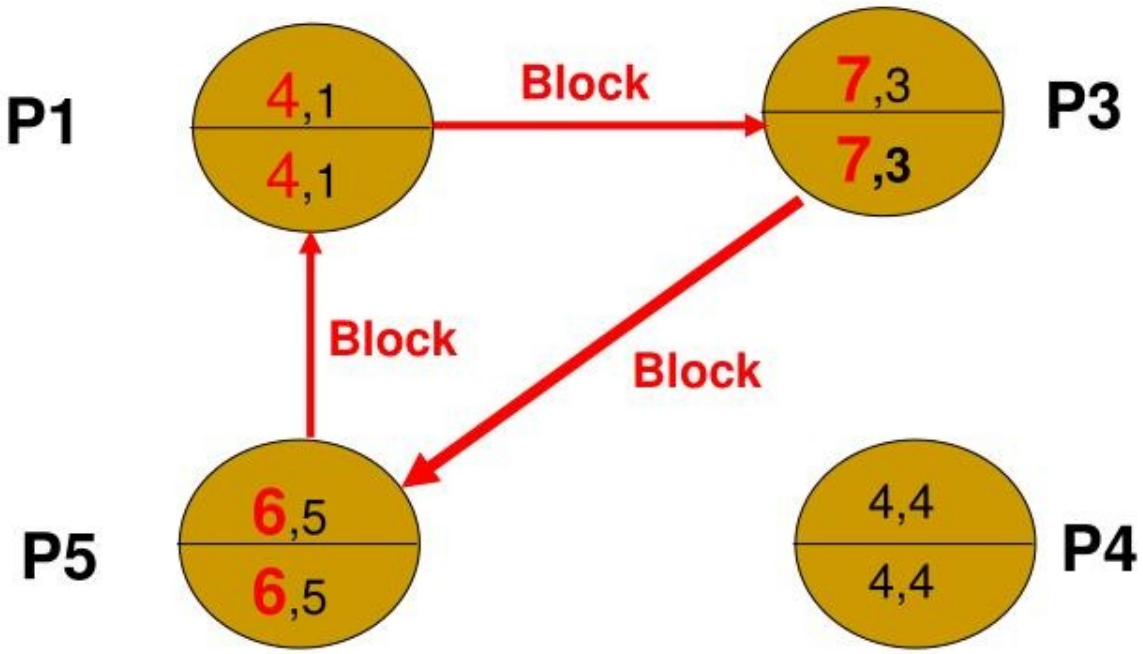
Block state will occur for P5



Mitchell-Merritt Algorithm Example cont…

Now suppose P3 is waiting for P5 (P3 → P5)

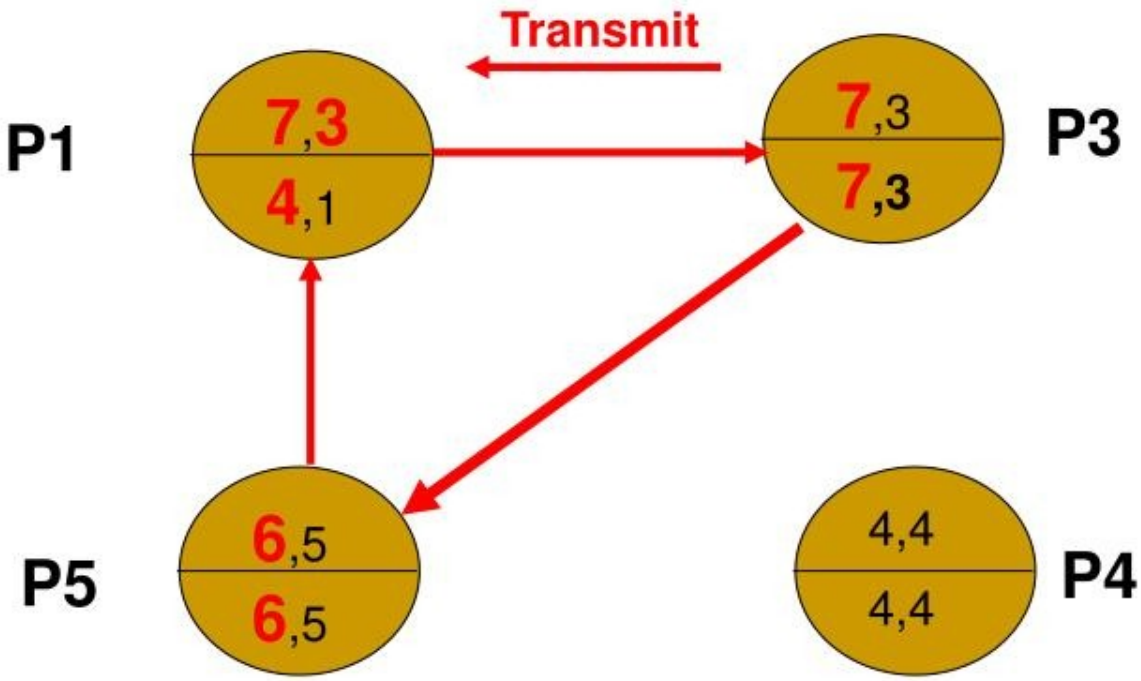
Block state will occur for P3



Mitchell-Merritt Algorithm Example cont…

Now P3 initiates Transmit Phase

P3 will transmit its public label to P1 (Reverse Direction)



Here P1 reads public label of P3

P1's public label =(4,1)

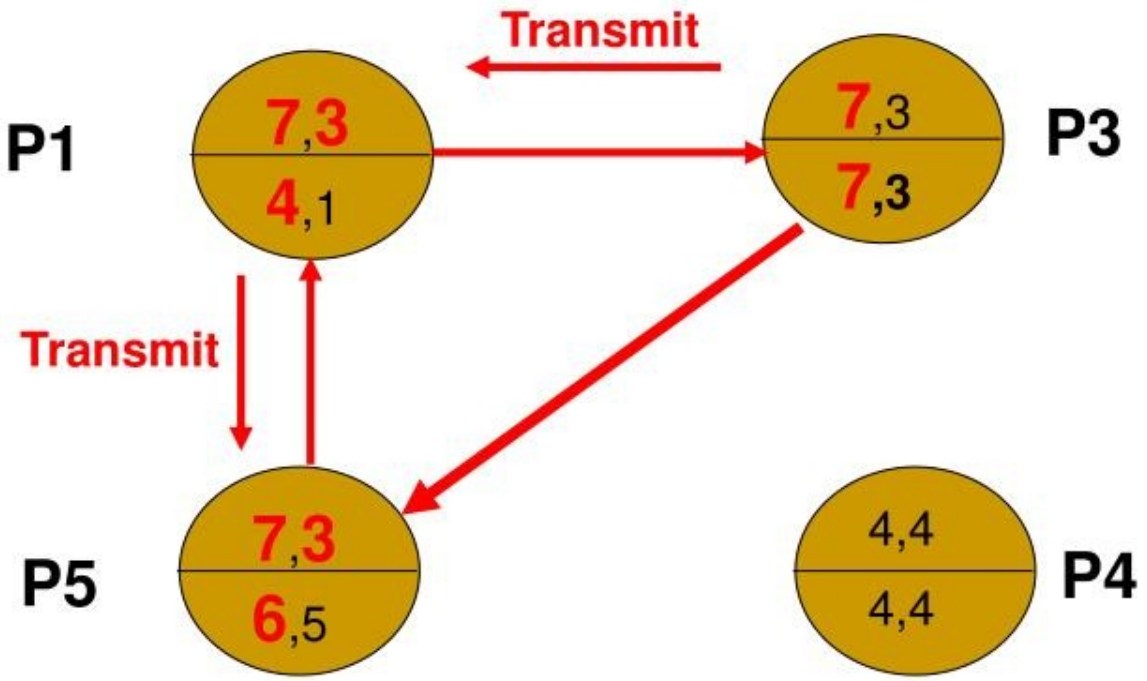
P3's public label =(7,3)

So P1 will change is public label to (7,3)

But No change for private label of P1

Mitchell-Merritt Algorithm Example cont…

P1 will transmit its public label to P5 (Reverse Direction)



P1's public label =(7,3)

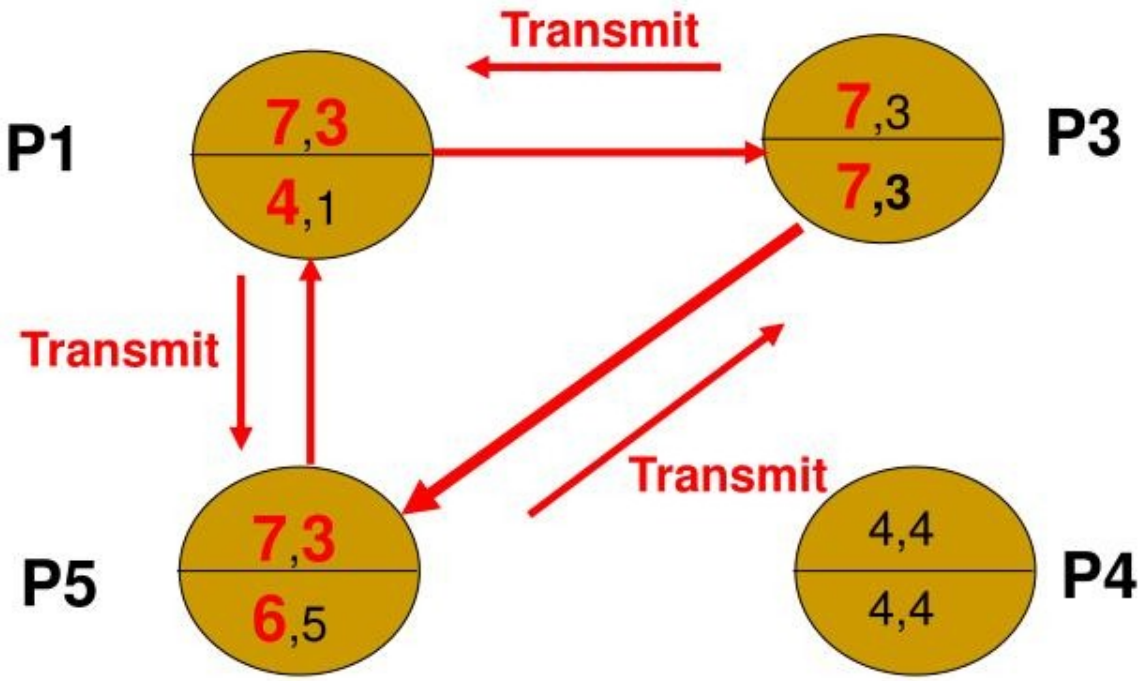
P5's public label =(6,5)

So P5 will change is public label to (7,3)

But No change for private label of P5

Mitchell-Merritt Algorithm Example cont…

P5 will transmit its public label to P3 (Reverse Direction)



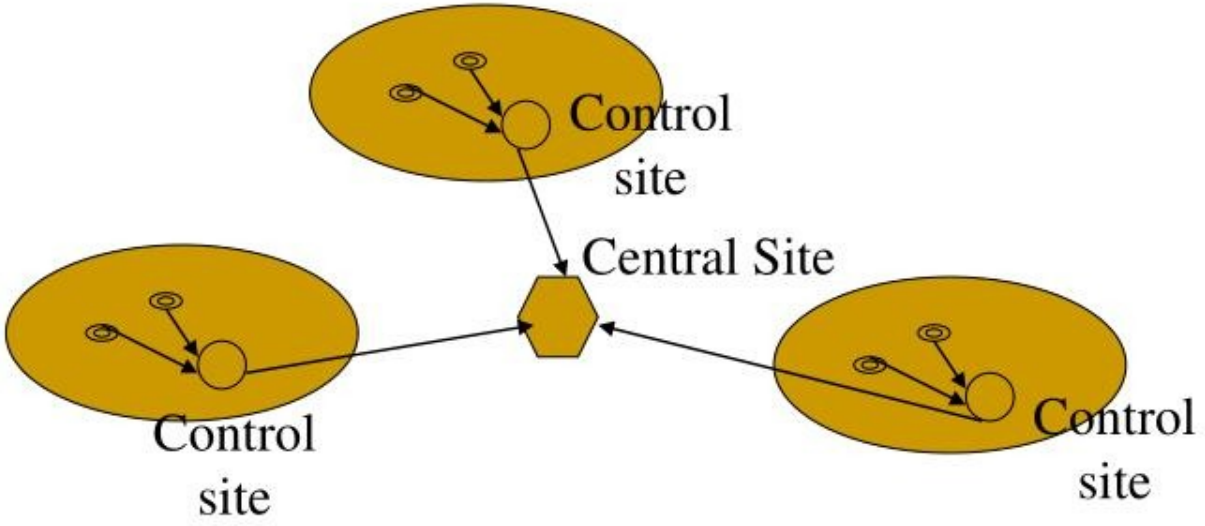
P5's public label =7,3

P3's public label =7,3

So P3 Detects Deadlock

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.