Distributed Mutual Exclusion Algorithms

Gaurav Parashar

September 6, 2020

Table of Contents

Introduction

- Mutual exclusion: Concurrent access of processes to a shared resource or data is executed in mutually exclusive manner.

Introduction

- Mutual exclusion: Concurrent access of processes to a shared resource or data is executed in mutually exclusive manner.
- Only one process is allowed to execute the critical section (CS) at any given time.

Introduction

- Mutual exclusion: Concurrent access of processes to a shared resource or data is executed in mutually exclusive manner.
- Only one process is allowed to execute the critical section (CS) at any given time.
- In a distributed system, shared variables (semaphores) or a local kernel cannot be used to implement mutual exclusion.

Introduction

- Mutual exclusion: Concurrent access of processes to a shared resource or data is executed in mutually exclusive manner.
- Only one process is allowed to execute the critical section (CS) at any given time.
- In a distributed system, shared variables (semaphores) or a local kernel cannot be used to implement mutual exclusion.
- Message passing is the sole means for implementing distributed mutual exclusion.

Introduction

- Distributed mutual exclusion algorithms must deal with unpredictable message delays and incomplete knowledge of the system state.

Introduction

- Distributed mutual exclusion algorithms must deal with unpredictable message delays and incomplete knowledge of the system state.
- Three basic approaches for distributed mutual exclusion:

Introduction

- Distributed mutual exclusion algorithms must deal with unpredictable message delays and incomplete knowledge of the system state.
- Three basic approaches for distributed mutual exclusion:
  - Token based approach.

Introduction

- Distributed mutual exclusion algorithms must deal with unpredictable message delays and incomplete knowledge of the system state.
- Three basic approaches for distributed mutual exclusion:
  - Token based approach.
    - SUZUKI-KASAMI'S Broadcast Algorithm

Introduction

- Distributed mutual exclusion algorithms must deal with unpredictable message delays and incomplete knowledge of the system state.
- Three basic approaches for distributed mutual exclusion:
  - Token based approach.
    - SUZUKI-KASAMI'S Broadcast Algorithm
    - SINGHAL HEURUSTIC Algorithm

Introduction

- Distributed mutual exclusion algorithms must deal with unpredictable message delays and incomplete knowledge of the system state.
- Three basic approaches for distributed mutual exclusion:
  - Token based approach.
    - SUZUKI-KASAMI'S Broadcast Algorithm
    - SINGHAL HEURUSTIC Algorithm
    - RAYMOND's tree based Algorithm

Introduction

- Distributed mutual exclusion algorithms must deal with unpredictable message delays and incomplete knowledge of the system state.
- Three basic approaches for distributed mutual exclusion:
  - Token based approach.
    - SUZUKI-KASAMI'S Broadcast Algorithm
    - SINGHAL HEURUSTIC Algorithm
    - RAYMOND's tree based Algorithm
  - Non-token based approach.

Introduction

- Distributed mutual exclusion algorithms must deal with unpredictable message delays and incomplete knowledge of the system state.
- Three basic approaches for distributed mutual exclusion:
  - Token based approach.
    - SUZUKI-KASAMI'S Broadcast Algorithm
    - SINGHAL HEURUSTIC Algorithm
    - RAYMOND's tree based Algorithm
  - Non-token based approach.
    - LAMPORT'S Algorithm

Introduction

- Distributed mutual exclusion algorithms must deal with unpredictable message delays and incomplete knowledge of the system state.
- Three basic approaches for distributed mutual exclusion:
  - Token based approach.
    - SUZUKI-KASAMI'S Broadcast Algorithm
    - SINGHAL HEURUSTIC Algorithm
    - RAYMOND's tree based Algorithm
  - Non-token based approach.
    - LAMPORT'S Algorithm
    - RICART-AGRAWALA Algorithm

Introduction

- Distributed mutual exclusion algorithms must deal with unpredictable message delays and incomplete knowledge of the system state.
- Three basic approaches for distributed mutual exclusion:
  - Token based approach.
    - SUZUKI-KASAMI'S Broadcast Algorithm
    - SINGHAL HEURUSTIC Algorithm
    - RAYMOND's tree based Algorithm
  - Non-token based approach.
    - LAMPORT'S Algorithm
    - RICART-AGRAWALA Algorithm
    - MAEKAWA'S Algorithm

Introduction

- Token-based approach:

Introduction

- Token-based approach:
  - A unique token is shared among the sites.

Introduction

- Token-based approach:
  - A unique token is shared among the sites.
  - A site is allowed to enter its CS if it possesses the token.

Introduction

- Token-based approach:
  - A unique token is shared among the sites.
  - A site is allowed to enter its CS if it possesses the token.
  - Mutual exclusion is ensured because the token is unique.

Introduction

- Non-token based approach:

Introduction

- Non-token based approach:
  - Two or more successive rounds of messages are exchanged among the sites to determine which site will enter the CS next .

System Model

- The system consists of N sites, S1, S2, ..., SN .

System Model

- The system consists of N sites, S1, S2, ..., SN .
- We assume that a single process is running on each site. The process at site Si is denoted by pi.

System Model

- The system consists of N sites, S1, S2, ..., SN .
- We assume that a single process is running on each site. The process at site Si is denoted by pi.
- A site can be in one of the following three states: requesting the CS, executing the CS, or neither requesting nor executing the CS (i.e., idle).

System Model

- The system consists of N sites, S1, S2, ..., SN .
- We assume that a single process is running on each site. The process at site Si is denoted by pi.
- A site can be in one of the following three states: requesting the CS, executing the CS, or neither requesting nor executing the CS (i.e., idle).
- In the 'requesting the CS' state, the site is blocked and can not make further requests for the CS. In the 'idle' state, the site is executing outside the CS.

System Model

- The system consists of N sites, S1, S2, ..., SN .
- We assume that a single process is running on each site. The process at site Si is denoted by pi.
- A site can be in one of the following three states: requesting the CS, executing the CS, or neither requesting nor executing the CS (i.e., idle).
- In the 'requesting the CS' state, the site is blocked and can not make further requests for the CS. In the 'idle' state, the site is executing outside the CS.
- In token-based algorithms, a site can also be in a state where a site holding the token is executing outside the CS (called the idle token state)

System Model

- The system consists of N sites, S1, S2, ..., SN .
- We assume that a single process is running on each site. The process at site Si is denoted by pi.
- A site can be in one of the following three states: requesting the CS, executing the CS, or neither requesting nor executing the CS (i.e., idle).
- In the 'requesting the CS' state, the site is blocked and can not make further requests for the CS. In the 'idle' state, the site is executing outside the CS.
- In token-based algorithms, a site can also be in a state where a site holding the token is executing outside the CS (called the idle token state)
- At any instant, a site may have several pending requests for CS. A site queues up these requests and serves them one at a time.

Requirements of Mutual Exclusion Algorithms

Safety Property : At any instant, only one process can execute the critical section.

Requirements of Mutual Exclusion Algorithms

Safety Property : At any instant, only one process can execute the critical section.

Liveness Property : This property states the absence of deadlock and starvation. Two or more sites should not endlessly wait for messages which will never arrive.

Requirements of Mutual Exclusion Algorithms

Safety Property : At any instant, only one process can execute the critical section.

Liveness Property : This property states the absence of deadlock and starvation. Two or more sites should not endlessly wait for messages which will never arrive.

Fairness : Each process gets a fair chance to execute the CS. Fairness property generally means the CS execution requests are executed in the order of their arrival (time is determined by a logical clock) in the system.

Performance Metrics

The performance is generally measured by the following four metrics:

Message complexity : The number of messages required per CS execution by a site.

Synchronization delay : After a site leaves the CS, it is the time required and before the next site enters the CS (see Figure 1).

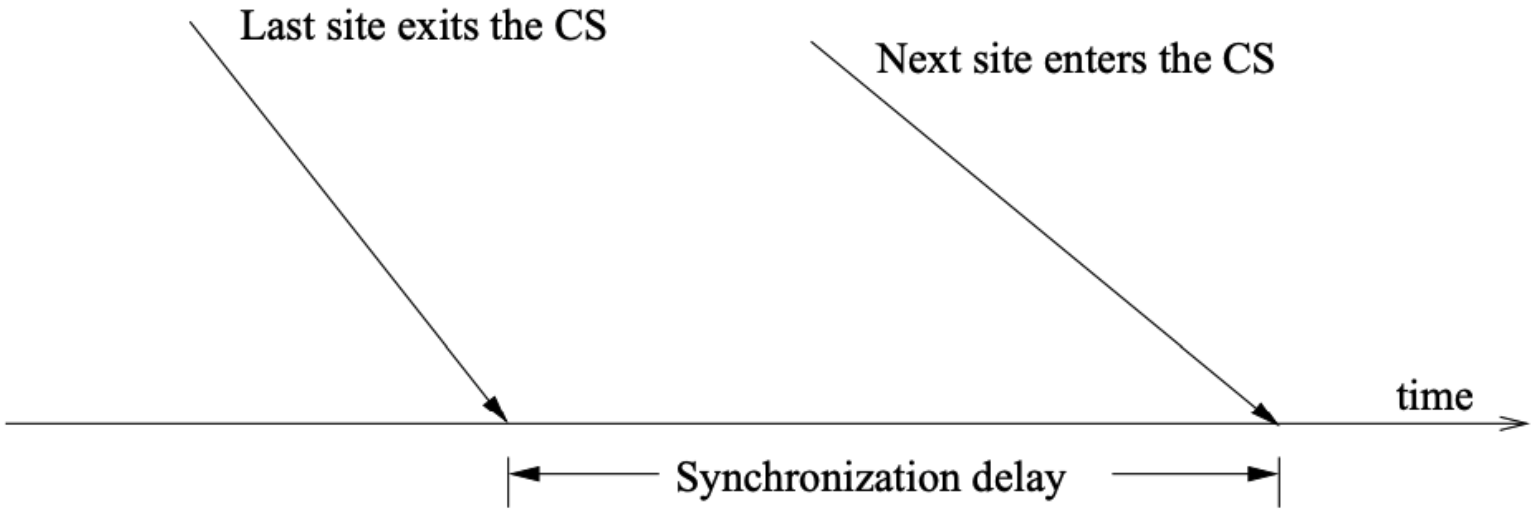


Figure: Synchronization Delay

Performance Metrics

Response time : The time interval a request waits for its CS execution to be over after its request messages have been sent out (see Figure 2).

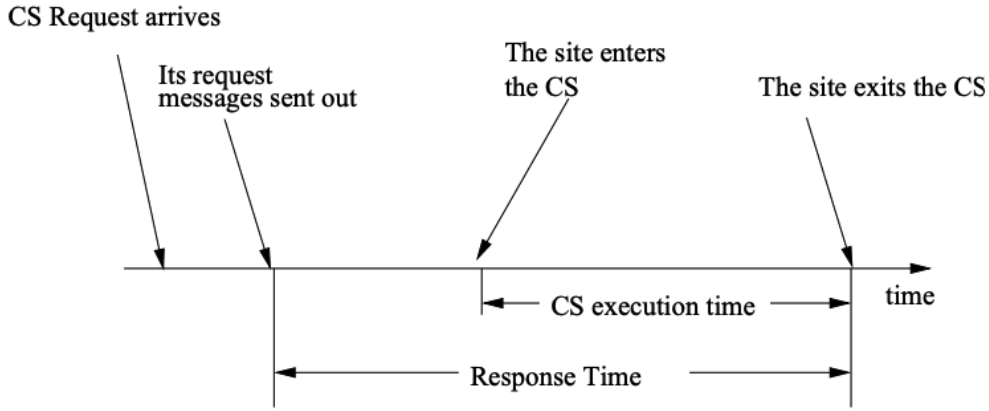


Figure: Response Time

System Throughput : The rate at which the system executes requests for the CS. System Throughput=

$$\frac{1}{SD + E}$$

where SD is the synchronization delay and E is the average critical section execution time

Performance Metrics

Low and High Load Performance

- We often study the performance of mutual exclusion algorithms under two special loading conditions, viz., "low load" and "high load".

Performance Metrics

Low and High Load Performance

- We often study the performance of mutual exclusion algorithms under two special loading conditions, viz., "low load" and "high load".
- The load is determined by the arrival rate of CS execution requests.

Performance Metrics

Low and High Load Performance

- We often study the performance of mutual exclusion algorithms under two special loading conditions, viz., "low load" and "high load".
- The load is determined by the arrival rate of CS execution requests.
- Under *low load* conditions, there is seldom more than one request for the critical section present in the system simultaneously.

Performance Metrics

Low and High Load Performance

- We often study the performance of mutual exclusion algorithms under two special loading conditions, viz., "low load" and "high load".
- The load is determined by the arrival rate of CS execution requests.
- Under *low load* conditions, there is seldom more than one request for the critical section present in the system simultaneously.
- Under heavy load conditions, there is always a pending request for critical section at a site

Difference between Token based and Non Token based Algorithms

Token-based algorithm	Non-Token based algorithm
unique token	messages are exchanged
enter if it has token	2 or more messages are exchanged
uses the sequences to order	uses the timestamp
They are free from deadlock	not free from deadlock
less message traffic	produces more message traffic
are scalable	are less scalable

Table: Difference between Token and Non Token based Algorithms

Introduction

Lamport's Algorithm

- Requests for CS are executed in the increasing order of timestamps and time is determined by logical clocks.

Introduction

Lamport's Algorithm

- Requests for CS are executed in the increasing order of timestamps and time is determined by logical clocks.
- Every site S_i keeps a queue, $request\_queue_i$ which contains mutual exclusion requests ordered by their timestamps.

Introduction

Lamport's Algorithm

- Requests for CS are executed in the increasing order of timestamps and time is determined by logical clocks.
- Every site S_i keeps a queue, $request\_queue_i$ which contains mutual exclusion requests ordered by their timestamps.
- This algorithm requires communication channels to deliver messages the FIFO order.

Algorithm

Requesting the critical section:

- When a site S_i wants to enter the CS, it broadcasts a REQUEST(ts_i, i) message to all other sites and places the request on $request\_queue_i$ ((ts_i, i) denotes the timestamp of the request.).

Executing the critical section: Site S_i enters the CS when the following two conditions hold:

L1 S_i has received a message with timestamp larger than (ts_i, i) from all other sites.

Algorithm

Requesting the critical section:

- When a site S_i wants to enter the CS, it broadcasts a REQUEST(ts_i, i) message to all other sites and places the request on $request\_queue_i$ ((ts_i, i) denotes the timestamp of the request.).
- When a site S_j receives the REQUEST(ts_i, i) message from site S_i ,places site S_i's request on request $queue_j$ and it returns a timestamped REPLY message to S_i.

Executing the critical section: Site S_i enters the CS when the following two conditions hold:

L1 S_i has received a message with timestamp larger than (ts_i, i) from all other sites.

L2 S_i's request is at the top of $request\_queue_i$.

Algorithm

Releasing the critical section:

- Site S_i , upon exiting the CS, removes its request from the top of its request queue and broadcasts a timestamped RELEASE message to all other sites.

When a site removes a request from its request queue, its own request may come at the top of the queue, enabling it to enter the CS.

Algorithm

Releasing the critical section:

- Site S_i , upon exiting the CS, removes its request from the top of its request queue and broadcasts a timestamped RELEASE message to all other sites.
- When a site S_j receives the RELEASE message from site S_i, it removes S_i's request from its request queue.

When a site removes a request from its request queue, its own request may come at the top of the queue, enabling it to enter the CS.

correctness

Theorem: Lamport's algorithm achieves mutual exclusion. Proof:

- Proof is by contradiction. Suppose two sites S_i and S_j are executing the CS concurrently. For this to happen conditions L1 and L2 must hold at both the sites concurrently

correctness

Theorem: Lamport's algorithm achieves mutual exclusion. Proof:

- Proof is by contradiction. Suppose two sites S_i and S_j are executing the CS concurrently. For this to happen conditions L1 and L2 must hold at both the sites concurrently
- This implies that at some instant in time, say t, both S_i and S_j have their own requests at the top of their request queues and condition L1 holds at them. Without loss of generality, assume that S_i 's request has smaller timestamp than the request of S_j.

correctness

Theorem: Lamport's algorithm achieves mutual exclusion. Proof:

- Proof is by contradiction. Suppose two sites S_i and S_j are executing the CS concurrently. For this to happen conditions L1 and L2 must hold at both the sites concurrently
- This implies that at some instant in time, say t, both S_i and S_j have their own requests at the top of their request queues and condition L1 holds at them. Without loss of generality, assume that S_i 's request has smaller timestamp than the request of S_j.
- From condition L1 and FIFO property of the communication channels, it is clear that at instant t the request of Si must be present in $request\_queue_j$ when S_j was executing its CS. This implies that S_j's own request is at the top of its own request queue when a smaller timestamp request, S_i's request, is present in the $request\_queue_j$ – a contradiction!

Performance

- For each CS execution, Lamport's algorithm requires (N - 1) REQUEST messages, (N - 1) REPLY messages, and (N - 1) RELEASE messages.

Performance

- For each CS execution, Lamport's algorithm requires (N - 1) REQUEST messages, (N - 1) REPLY messages, and (N - 1) RELEASE messages.
- Thus, Lamport's algorithm requires 3(N - 1) messages per CS invocation.

Performance

- For each CS execution, Lamport's algorithm requires (N - 1) REQUEST messages, (N - 1) REPLY messages, and (N - 1) RELEASE messages.
- Thus, Lamport's algorithm requires 3(N - 1) messages per CS invocation.
- Synchronization delay in the algorithm is T.

Introduction

Ricart-Agrawala Algorithm

- The Ricart-Agrawala algorithm assumes the communication channels are FIFO. The algorithm uses two types of messages: REQUEST and REPLY.

Introduction

Ricart-Agrawala Algorithm

- The Ricart-Agrawala algorithm assumes the communication channels are FIFO. The algorithm uses two types of messages: REQUEST and REPLY.
- A process sends a REQUEST message to all other processes to request their permission to enter the critical section. A process sends a REPLY message to a process to give its permission to that process.

Introduction

Ricart-Agrawala Algorithm

- The Ricart-Agrawala algorithm assumes the communication channels are FIFO. The algorithm uses two types of messages: REQUEST and REPLY.
- A process sends a REQUEST message to all other processes to request their permission to enter the critical section. A process sends a REPLY message to a process to give its permission to that process.
- Processes use Lamport-style logical clocks to assign a timestamp to critical section requests and timestamps are used to decide the priority of requests.

Introduction

Ricart-Agrawala Algorithm

- The Ricart-Agrawala algorithm assumes the communication channels are FIFO. The algorithm uses two types of messages: REQUEST and REPLY.
- A process sends a REQUEST message to all other processes to request their permission to enter the critical section. A process sends a REPLY message to a process to give its permission to that process.
- Processes use Lamport-style logical clocks to assign a timestamp to critical section requests and timestamps are used to decide the priority of requests.
- Each process P_i maintains the Request-Deferred array, RD_i , the size of which is the same as the number of processes in the system.

Introduction

Ricart-Agrawala Algorithm

- The Ricart-Agrawala algorithm assumes the communication channels are FIFO. The algorithm uses two types of messages: REQUEST and REPLY.
- A process sends a REQUEST message to all other processes to request their permission to enter the critical section. A process sends a REPLY message to a process to give its permission to that process.
- Processes use Lamport-style logical clocks to assign a timestamp to critical section requests and timestamps are used to decide the priority of requests.
- Each process P_i maintains the Request-Deferred array, RD_i , the size of which is the same as the number of processes in the system.
- Initially, $\forall I\ \forall j$: RD_i [j]=0. Whenever p_i defer the request sent by p_j , it sets RD_i[j]=1 and after it has sent a REPLY message to p_j, it sets RD_i[j]=0.

Resource vs Communication Deadlock

Resource Deadlock:

- Process can simultaneously send requests for resources
- Processes can hold resources and send requests, simultaneously

Network Deadlock:

- Process wait to communicate with other process among set of processes.
- a waiting process can get unblocked on receiving a communication from any one of the processes

"Wait to acquire resource" vs "Wait to communicate"

System Model

System Model

- The systems have only reusable resources
- Processes are allowed only exclusive access to resources
- there is only one copy of each resource

A process can be in two states running or blocked. A Graph-Theoritic Model

- The state of process-resource interaction in DS can be modelled by a bi-partite directed graph called as resource allocation graph.
- Nodes of this graph are processes & resources of a system are edges, which depicts pending or assignment state
- A system is in deadlock if it is's resource allocation graph contains a directed cycle

Deadlock handling strategies

- Prevention
- Avoidance
- Detection

Deadlock Detection & Resolution entails addressing two basic issues:-

- Detection of existing deadlocks
- Resolution of distributed deadlock

The Detection of Deadlock involves tow issues:-

- Maintenance of WFG
- Search of the WFG for presence of cycles

In distributed systems, a cycle may involve several sites, so the search for cycles greatly depends upon the WFG of the entire DS. A correct deadlock detection algorithm must satisfy the following two conditions:-

- Progress - No undetected deadlocks
  - The algorithm must detect all existing deadlocks in finite time.

In distributed systems, a cycle may involve several sites, so the search for cycles greatly depends upon the WFG of the entire DS. A correct deadlock detection algorithm must satisfy the following two conditions:-

- Progress - No undetected deadlocks
  - The algorithm must detect all existing deadlocks in finite time.
  - Once a deadlock had occurred, the deadlock detection activity should continuously progress until the deadlock is detected.
- Safety - No false deadlock
  - The algorithm should not report any deadlocks which are non-existent called as Phantom deadlocks.

Resolution

- Deadlock Resolution involves breaking the existing wait for dependencies in the system WFG
- It involves either rolling back one or more processes that are deadlocked & assigning their resources to blocked processes in the deadlock so that they can resume execution.
- When WFG dependency is broken, the corresponding info should be cleared immediately, otherwise will result in Phantom deadlock

Deadlock Detection Algorithms

Control Organization for Distributed Deadlock Detection

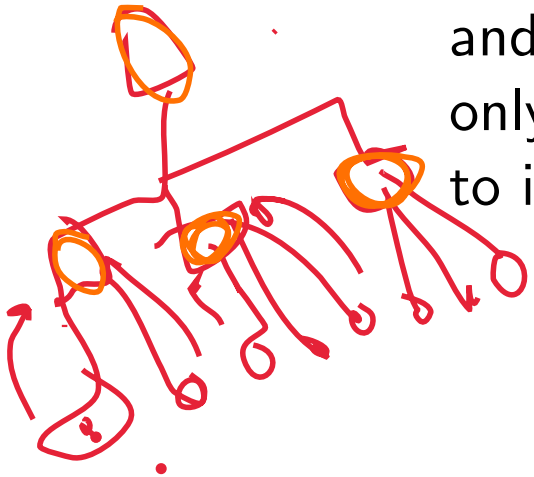
- Centralized control -(Ho-Ramamoorthy's one and two phase algorithms)
- Distributed control (Obermarck's Path Pushing Algorithm, Chandy-Mishra-Haas Edge Chasing Algorithm , Diffusion Computation, Global state detection)
- Hierarchical control (Menasce-Muntz Algorithm, Ho-Ramammoorthy' Algorithm)

Control Organization for Distributed Deadlock Detection

Centralized Control : Control site has the responsibility of constructing the global WFG & searching it for cycles.

Distributed Control Responsibility is shared equally among all the nodes. WFG is spread over several sites and participates in deadlock detection.

Hierarchical Control It combines the benefits of both centralised and distributed approach. Sites detect deadlock in only child sites & resultant WFG or RAG is forwarded to its parent site.



Ho-Ramamoorthy's one and two phase algorithms

One Phase Algorithm

- Each site maintains two states table: process and resource table.
- One of the site becomes the central control site.
- The central control site periodically asks for the states table.
- control site builds WFG using the states table.
- control site analyzes WFG and resolves any present cycles.

Short comings

- Phantom Deadlocks.

Ho-Ramamoorthy's one and two phase algorithms

One Phase Algorithm

- Each site maintains two states table: process and resource table.
- One of the site becomes the central control site.
- The central control site periodically asks for the states table.
- control site builds WFG using the states table.
- control site analyzes WFG and resolves any present cycles.

Short comings

- Phantom Deadlocks.
- High storage and communication costs

Phantom Deadlock

- System A has Process P1, holding resource S and waiting for T which is held by P2 in System B.
- Here process P1 sends 2 messages, to control site, namely release of resource and need of resource T.
- Coordinator finds a cycle and would indicate the system is in deadlock state. Refer Figure: 3

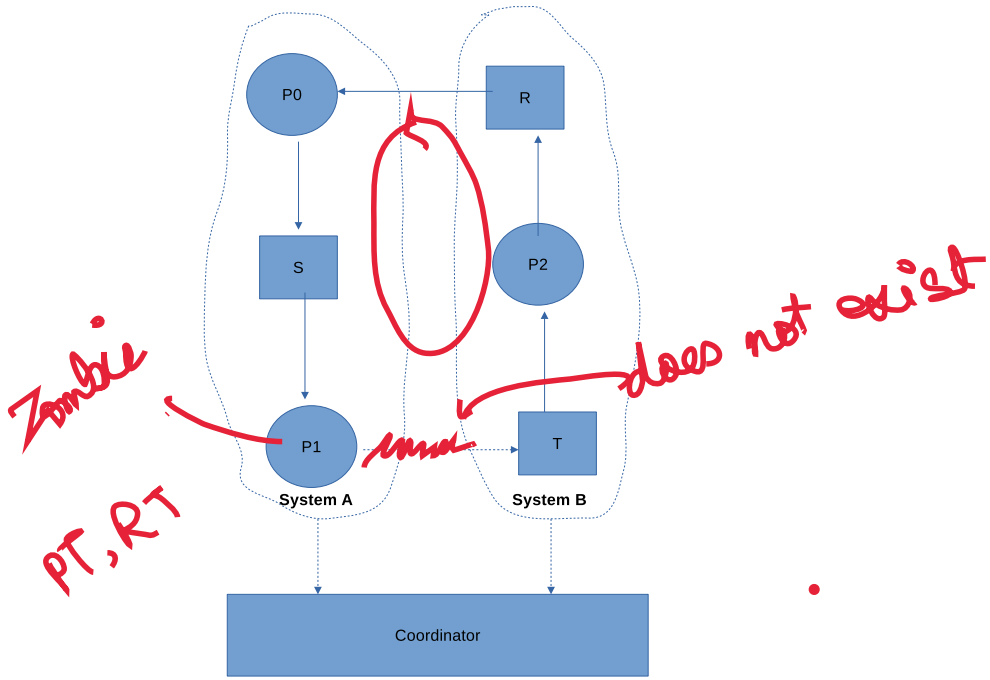
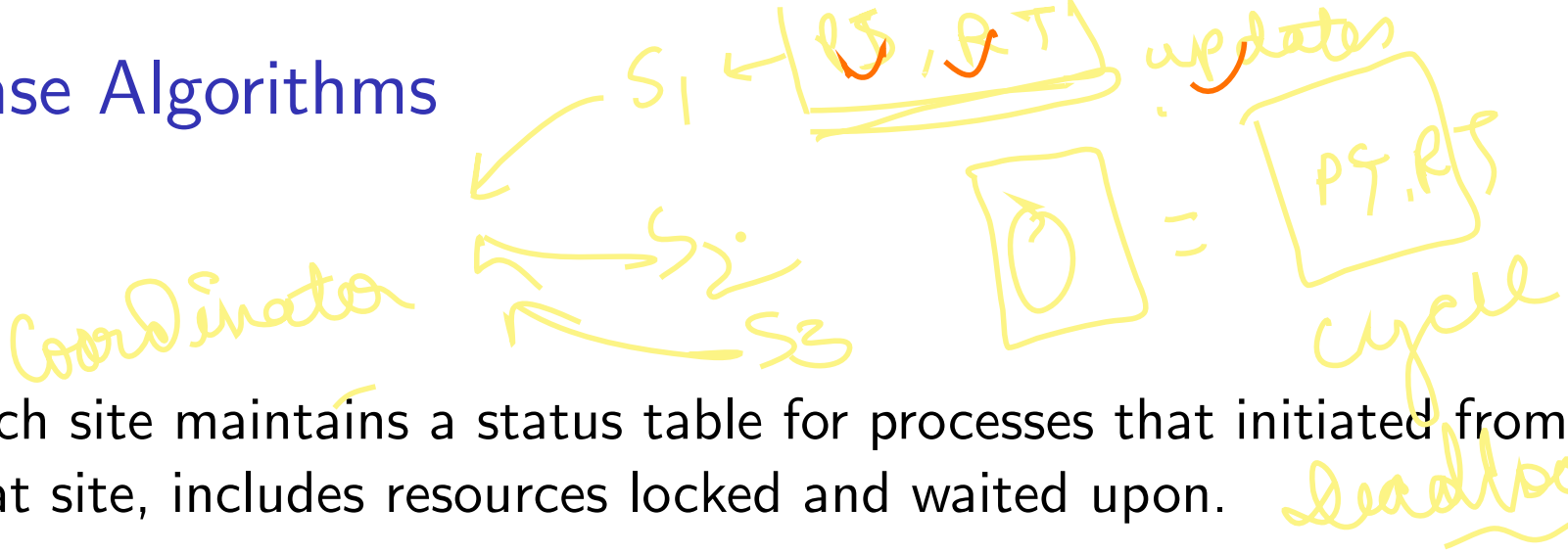


Figure: Phantom Deadlock

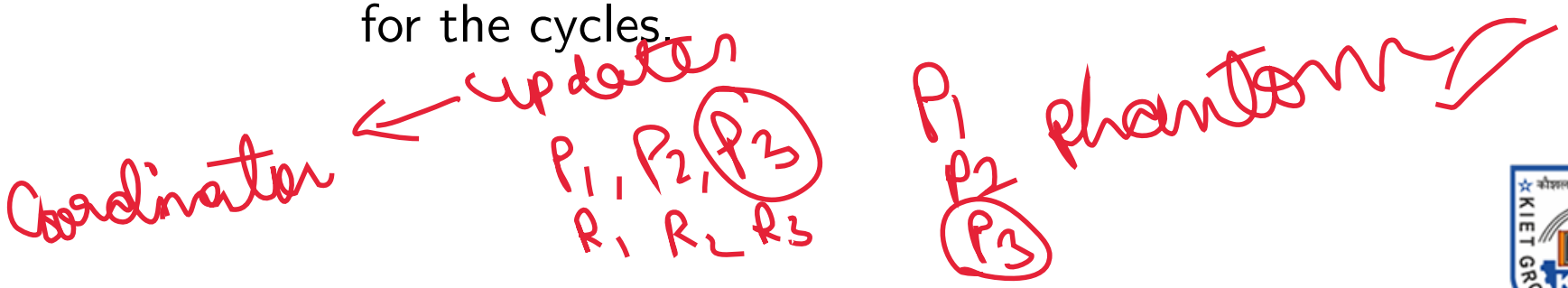
2-Phase Algorithms



Each site maintains a status table for processes that initiated from that site, includes resources locked and waited upon.

Phase 1 : control site periodically ask for these locked and waited tables. It then teaches for cycles in these tables.

Phase 2 : If cycle is found then, it control site makes 2nd request for the tables. If the details are found common in both the table. Requests will be analysed for the cycles.



Obermarks's Path Pushing Algorithm

- Information about the wait-for dependencies are propagated in the form of paths.
- The site wait for deadlock related information from other sites
- the site combines received information with its local TWF(Transaction Wait For Graph) to build an updated TWF graph.
- messages are sent from one site to another in the form of string using paths.

Refer Figure: 4, Figure: 5, Figure: 6 and Figure: 7.

Obermarks's Path Pushing Algorithm

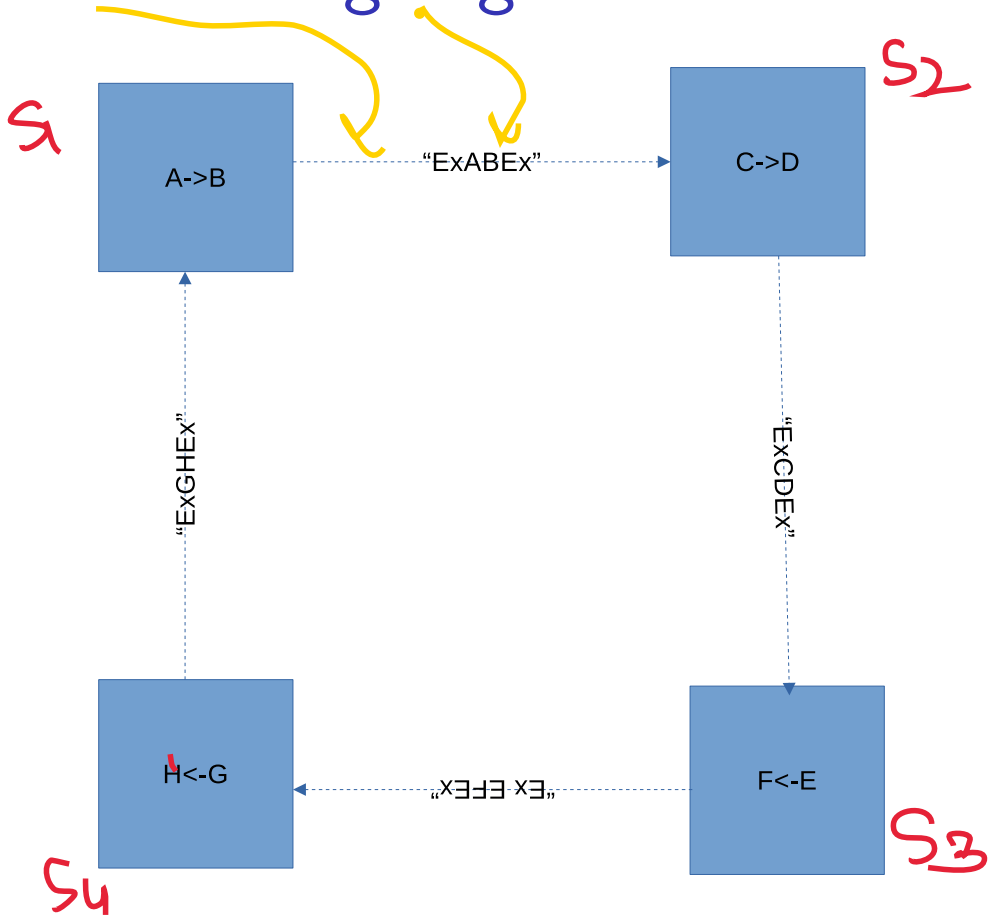


Figure: Step 1

Obermarks's Path Pushing Algorithm

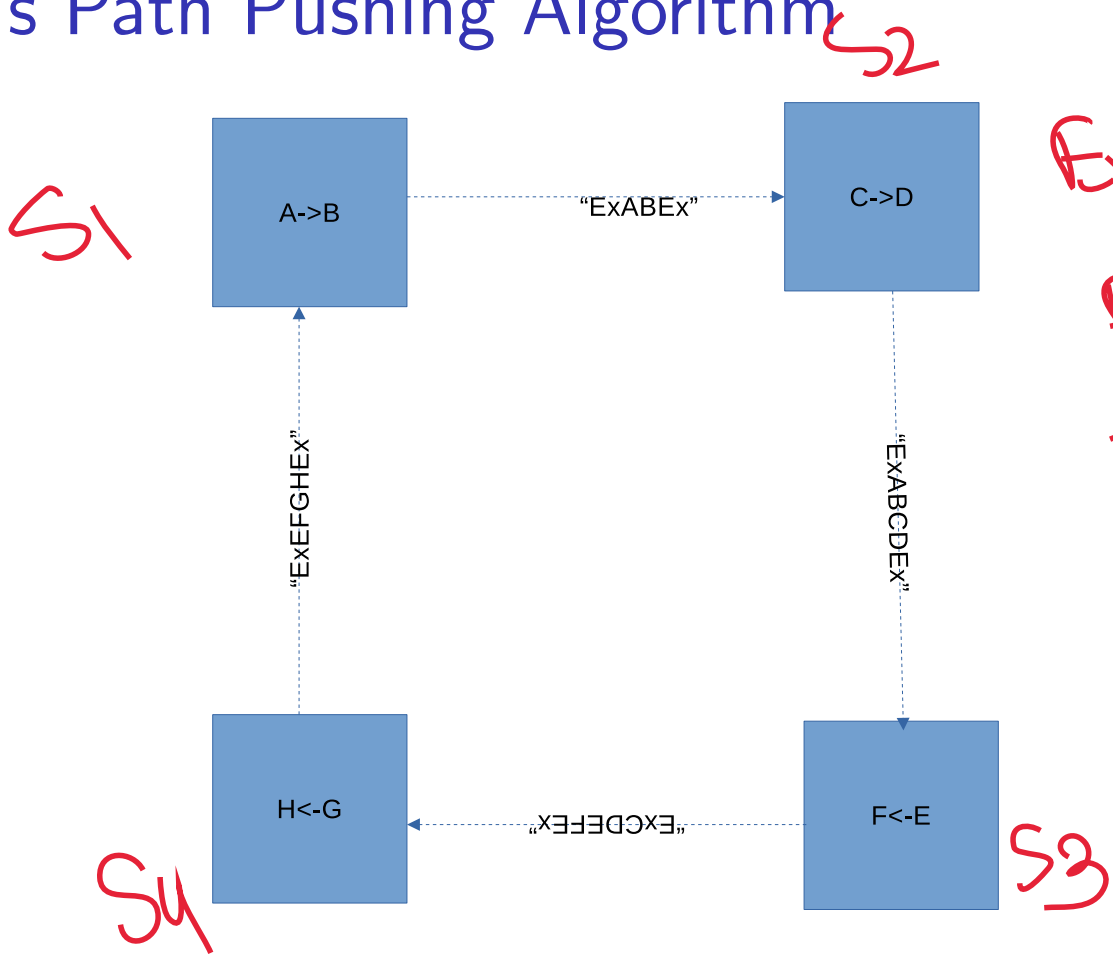


Figure: Step 2

Obermarks's Path Pushing Algorithm

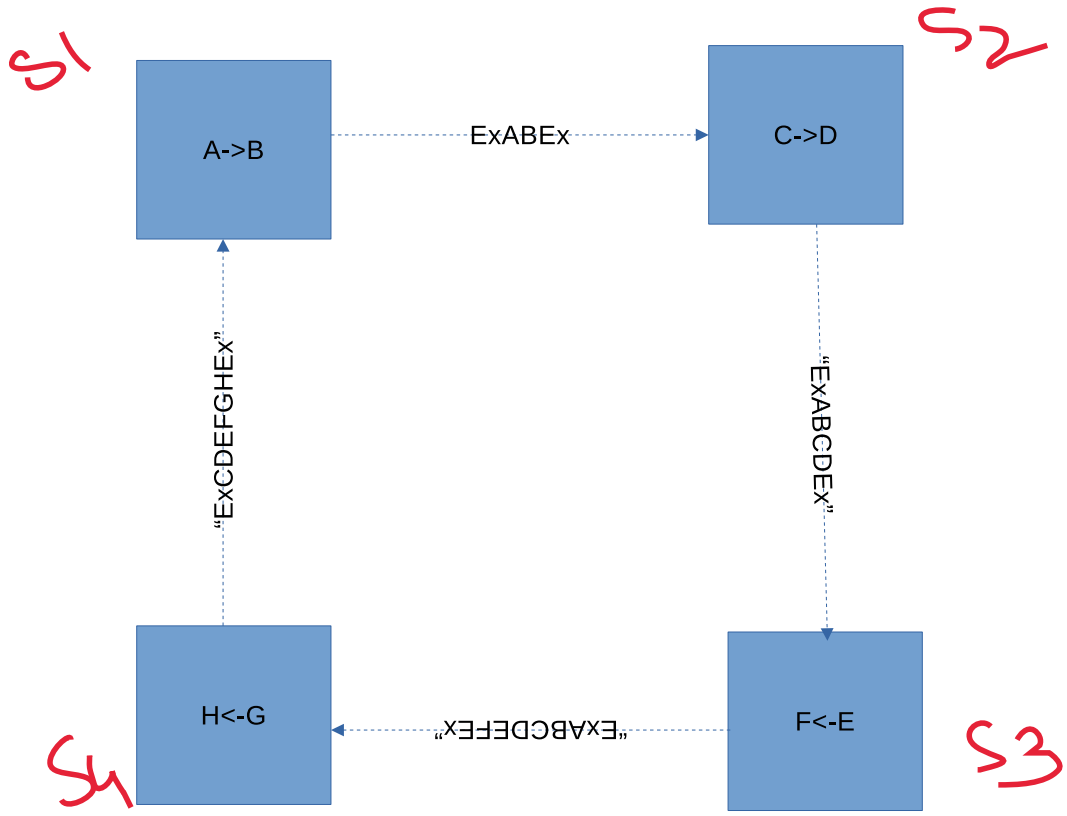


Figure: Step 3

Obermarks's Path Pushing Algorithm

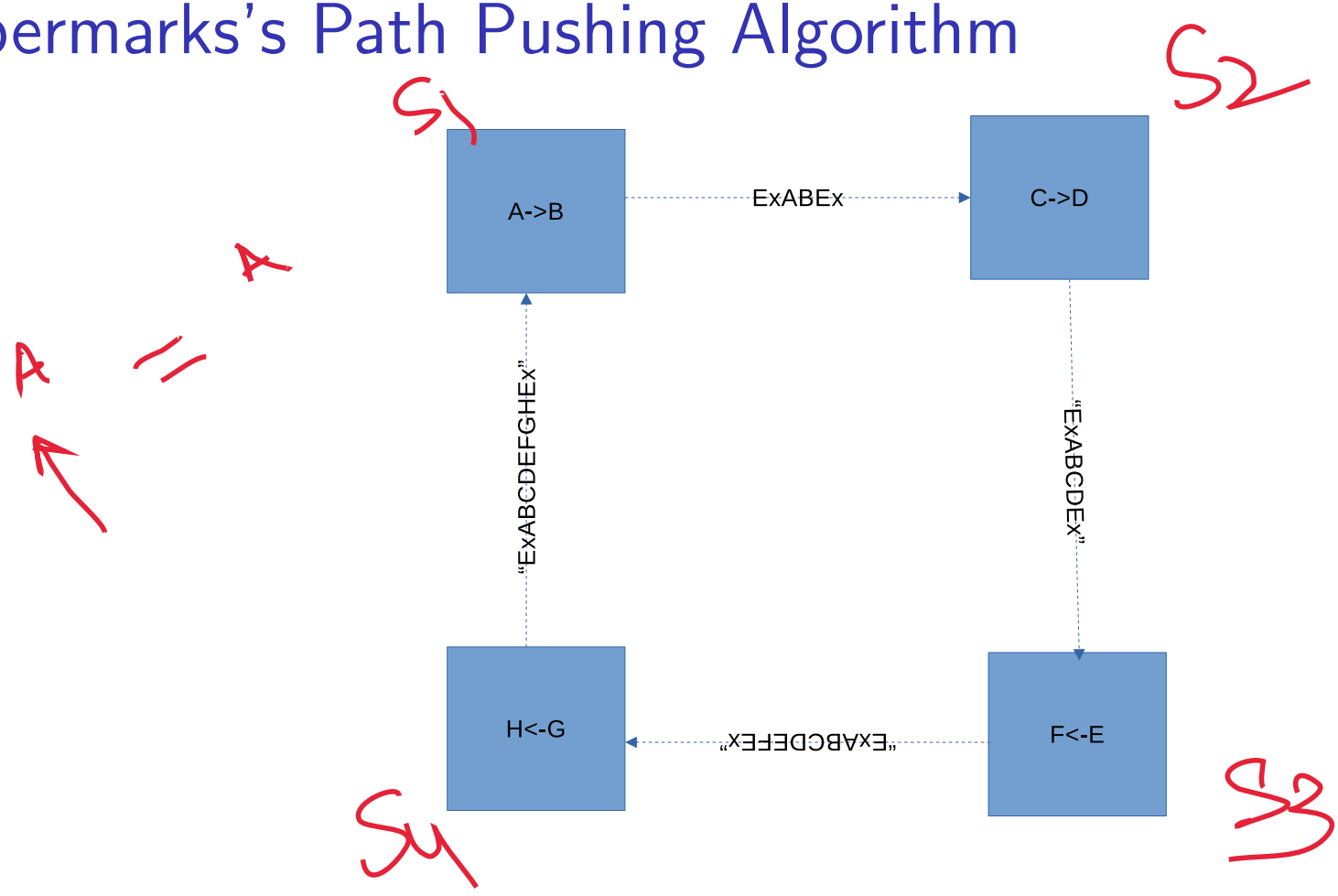
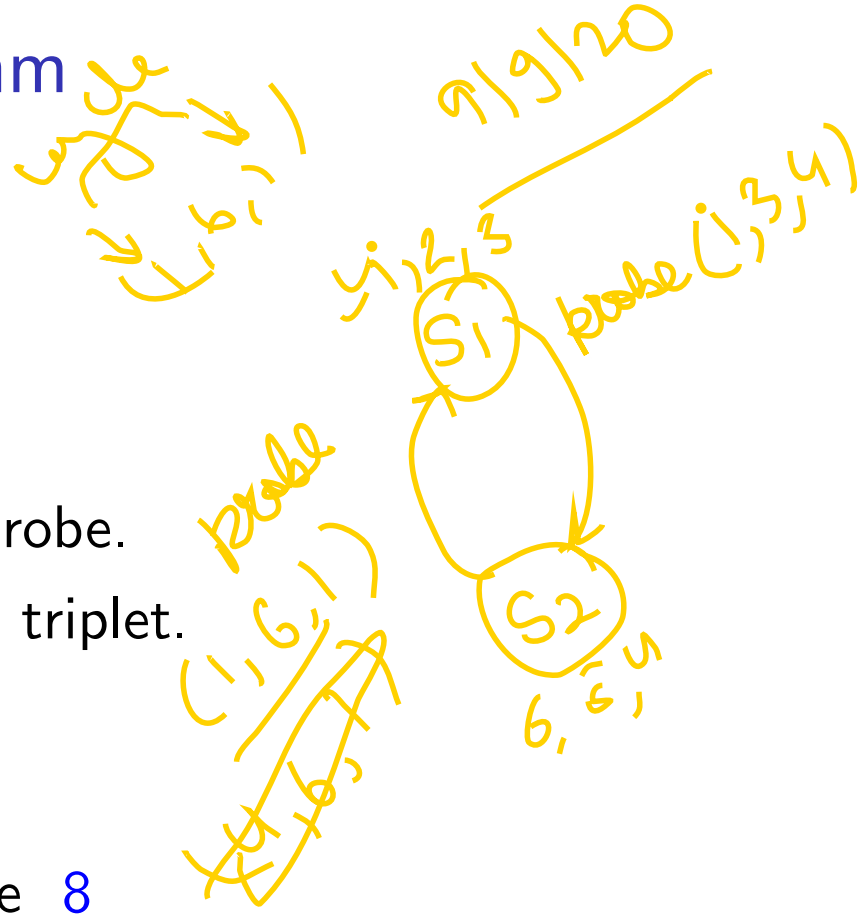


Figure: Step 4

Chandy-Mishra Haas's Algorithm



- Messages are sent named as Probe.
- Probe syntax: Probe(i,j,k) is a triplet.
- $P_i \rightarrow$ Initiate
- $P_j \rightarrow$ Site sent message to P_k

The algorithm is explained in Figure 8

Chandy-Mishra Haas's Algorithm

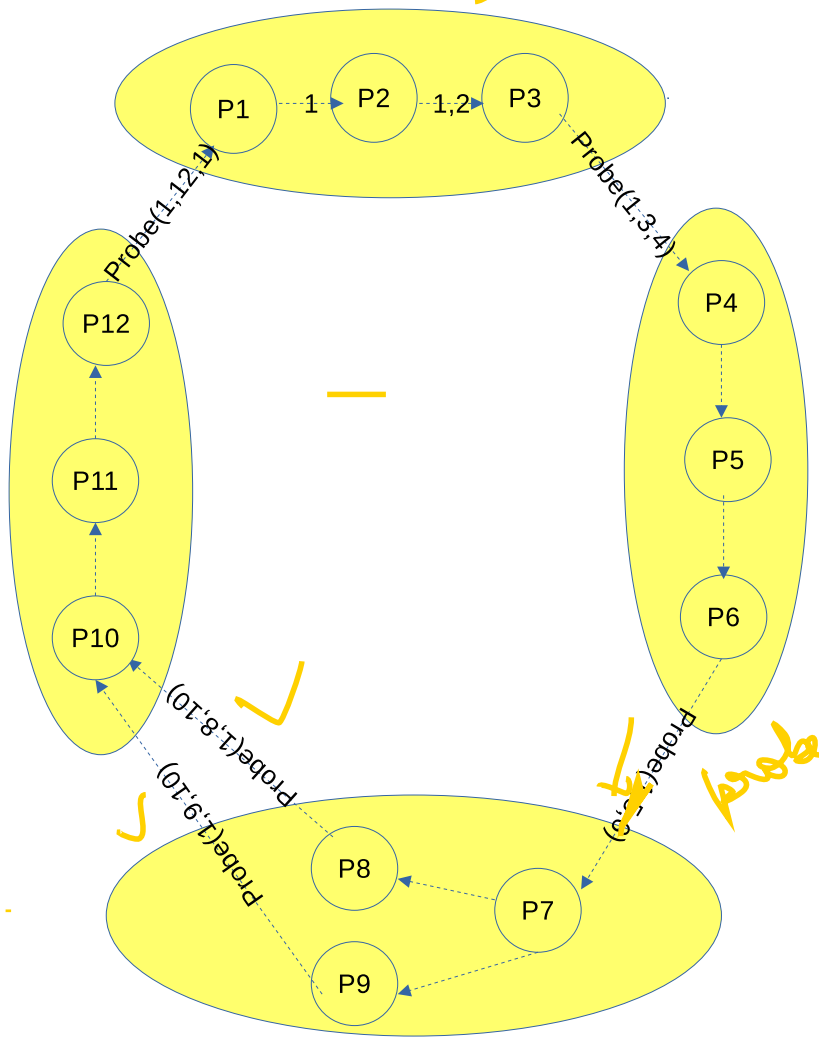


Figure: Chandy-Mishra Haas's Algorithm