CST 402 - DISTRIBUTED COMPUTING

Module - III

Module – III
Lesson Plan

- **L1: Distributed mutual exclusion algorithms – System model, Lamport's algorithm**
- **L2: Ricart–Agrawala algorithm**
- **L3: Quorum-based mutual exclusion algorithms – Maekawa's algorithm**
- **L4: Token-based algorithm – Suzuki–Kasami's broadcast algorithm.**
- **L5:** Deadlock detection in distributed systems – System model, Deadlock handling strategies, Issues in deadlock detection**.**
- **L6:** Models of deadlocks

Distributed mutual exclusion algorithms

- Mutual exclusion is a fundamental problem in distributed computing systems.
- Mutual exclusion ensures that concurrent access of processes to a shared resource or data is serialized, that is, executed in a mutually exclusive manner.
- Mutual exclusion in a distributed system states that only one process is allowed to execute the critical section (CS) at any given time
- There are three basic approaches for implementing distributed mutual exclusion:

1. Token-based approach.
2. Non-token-based approach
3. .Quorum-based approach.

Distributed mutual exclusion algorithms

In the token-based approach,

- a unique token is shared among the sites (PRIVILEGE MESSAGE).

- A site is allowed to enter its CS if it possesses the token and it continues to hold the token until the execution of the CS is over.

- Mutual exclusion is ensured because the token is unique

Distributed mutual exclusion algorithms

- **In the 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.
- A site enters the critical section (CS) when an assertion, defined on its local variables, becomes true.
- Mutual exclusion is enforced because the assertion becomes true only at one site at any given time.

Distributed mutual exclusion algorithms

- **In the quorum-based approach**,

➢ each site requests permission to execute the CS from a subset of sites (called a quorum).

➢ The quorums are formed in such a way that when two sites concurrently request access to the CS, at least one site receives both the requests and this site is responsible to make sure that only one request executes the CS at any time.

Distributed mutual exclusion algorithms

- **System model**

- The system consists of N sites, S1, S2,... Sn ,a single process is running on each site
- The process at site Si is denoted by pi.
- All these processes communicate asynchronously over an underlying communication network.
- A process wishing to enter the CS requests all other or a subset of processes by sending REQUEST messages, and waits for appropriate replies before entering the CS
- While waiting the process is not allowed to make further requests to enter the CS.
- A site can be in one of the following three states: requesting the CS, executing the CS, or neither requesting nor executing the CS (Idle state)

Distributed mutual exclusion algorithms

- In the "requesting the CS" state, the site is blocked and cannot make further requests for the CS.
- In the "idle" state, the site is executing outside the CS.
- idle token state :In the token-based algorithms, a site can also be in a state where a site holding the token is executing outside the CS.
- 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.

- communication channels →if they are FIFO or not→depends on the algm
- assume that channels reliably deliver all messages, sites do not crash, and the network does not get partitioned

Requirements of mutual exclusion algms

Satisfy 3 properties

1. **Safety property:**

- The safety property states that at any instant, only one process can execute the critical section.
- This is an essential property of a mutual exclusion algorithm.

2. **Liveness property :**

- This property states the absence of deadlock and starvation.
- Two or more sites should not endlessly wait for messages that will never arrive.
- A site must not wait indefinitely to execute the CS while other sites are repeatedly executing the CS.

- That is, every requesting site should get an opportunity to execute the CS in finite time.

Requirements of mutual exclusion algms contd…

3. Fairness :

- Fairness in the context of mutual exclusion means that each process gets a fair chance to execute the CS.

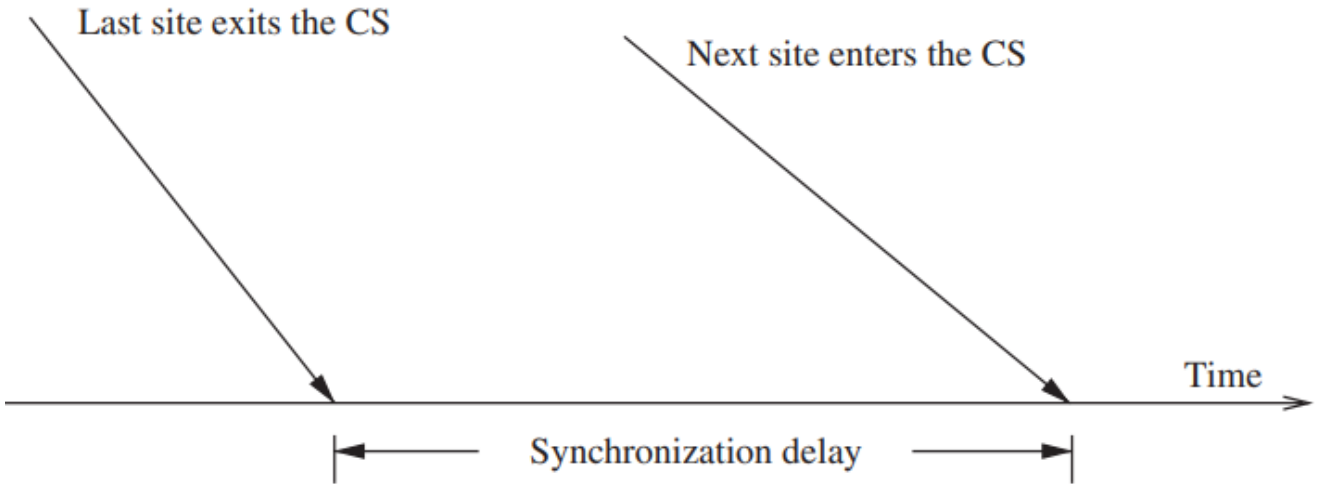
- In mutual exclusion algorithms, the fairness means that the CS execution requests are executed in order of their arrival in the system

Distributed mutual exclusion algorithms

Performance metrics - 4

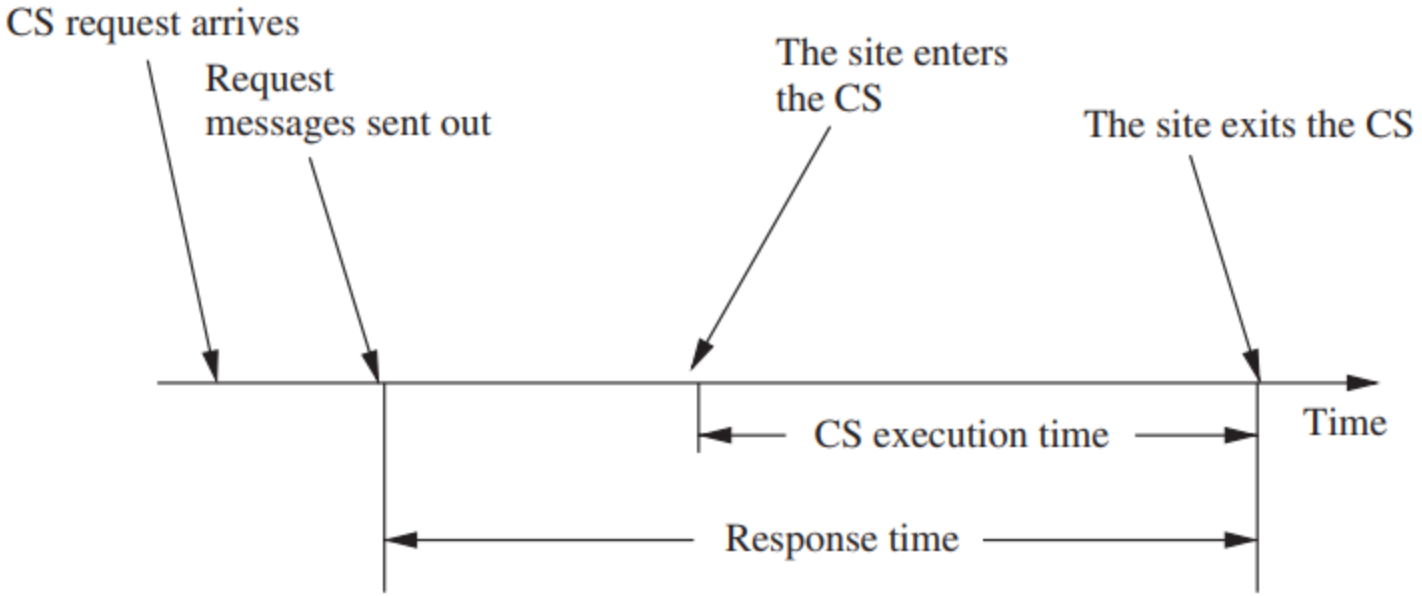
- **Message complexity :** This is the number of messages that are required per CS execution by a site

- **Synchronization delay :** After a site leaves the CS, it is the time required and before the next site enters the CS



Distributed mutual exclusion algorithms

- **Response time :** This is the time interval a request waits for its CS execution to be over after its request messages have been sent out



Distributed mutual exclusion algorithms

Performance metrics

System throughput This is the rate at which the system executes requests for the CS. If SD is the synchronization delay and E is the average critical section execution time, then the throughput is given by the following equation:

$$\text{System throughput} = \frac{1}{(SD + E)}.$$

Lamport's algorithm

- Lamport developed a distributed mutual exclusion algorithm as an illustration of his clock synchronization scheme

- The algorithm is fair → a request for CS are executed in the order of their timestamps and time is determined by logical clocks.

- When a site processes a request for the CS, it updates its local clock and assigns the request a timestamp.

- The algorithm executes CS requests in the increasing order of timestamps.

- Every site Si keeps a queue, request_queuei, which contains mutual exclusion requests ordered by their timestamps.

- This algorithm requires communication channels to deliver messages in FIFO order.

Lamport's 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, it places site S_i's request on $request\_queue_j$ and 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 $request\_queue_i$.

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 a RELEASE message from site S_i, it removes S_i's request from its request queue.

Algorithm 9.1 Lamport's algorithm.

Lamport's algorithm

- 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.
- When a site receives a REQUEST, REPLY, or RELEASE message, it updates its clock using the timestamp in the message
- **Performance :**

➢ 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.

Lamport's algorithm

- sites S1 and S2 are making requests for the CS and send out REQUEST messages to other sites. The timestamps of the requests are (1,1) and (1,2), respectively

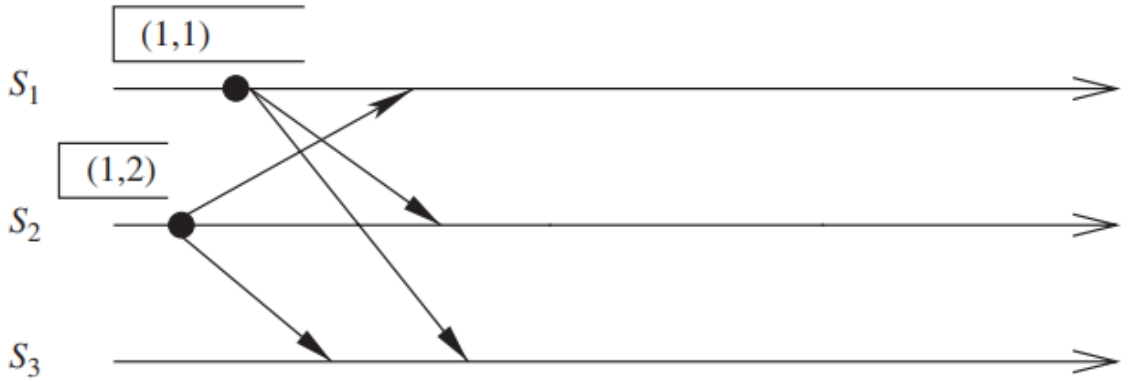
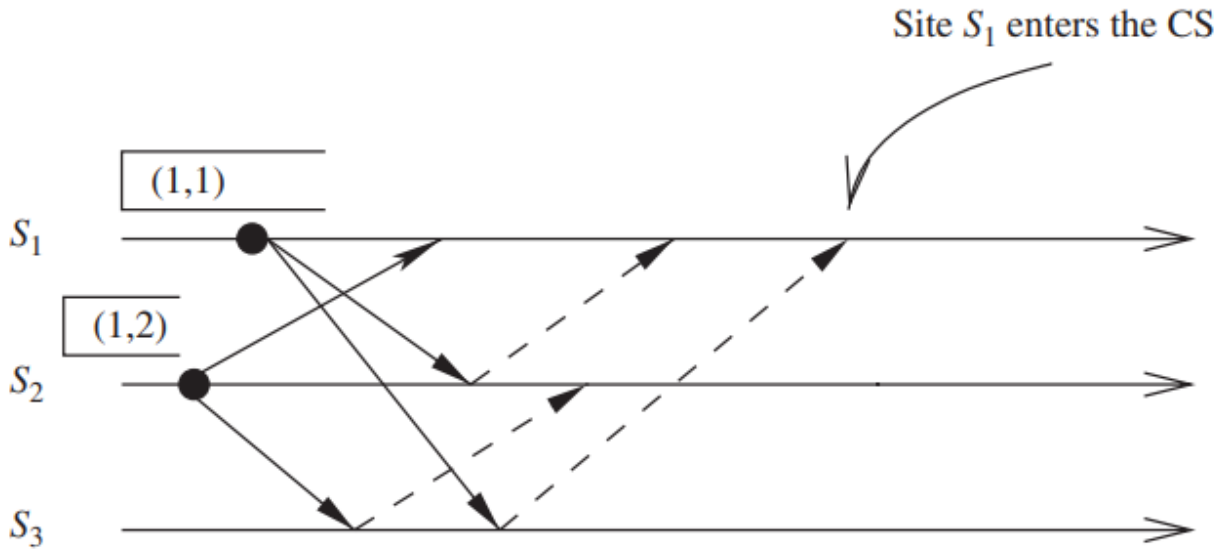


Figure 9.3 Sites S_1 and S_2 are Making Requests for the CS.

Lamport's algorithm

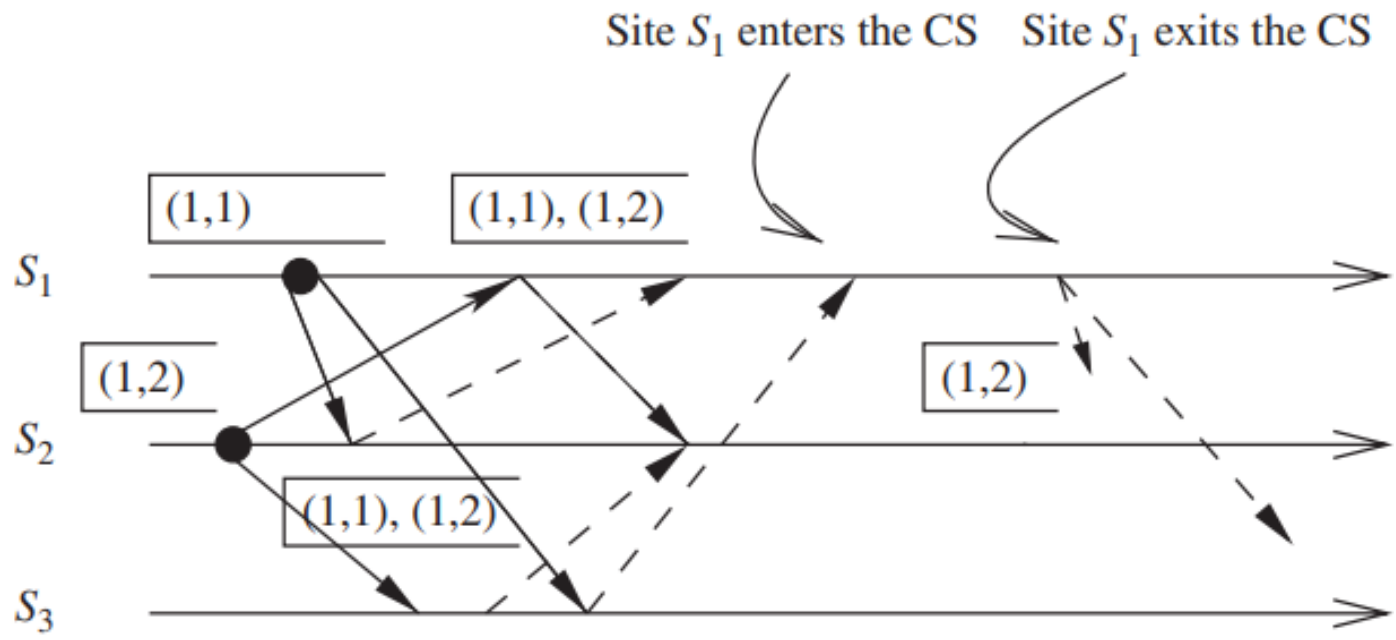
- both the sites S1 and S2 have received REPLY messages from all other sites.
- S1 has its request at the top of its request_queue but site S2 does not have its request at the top of its request_queue.
- Consequently, site S1 enters the CS

Figure 9.4 Site S_1 enters the CS.



Lamport's algorithm

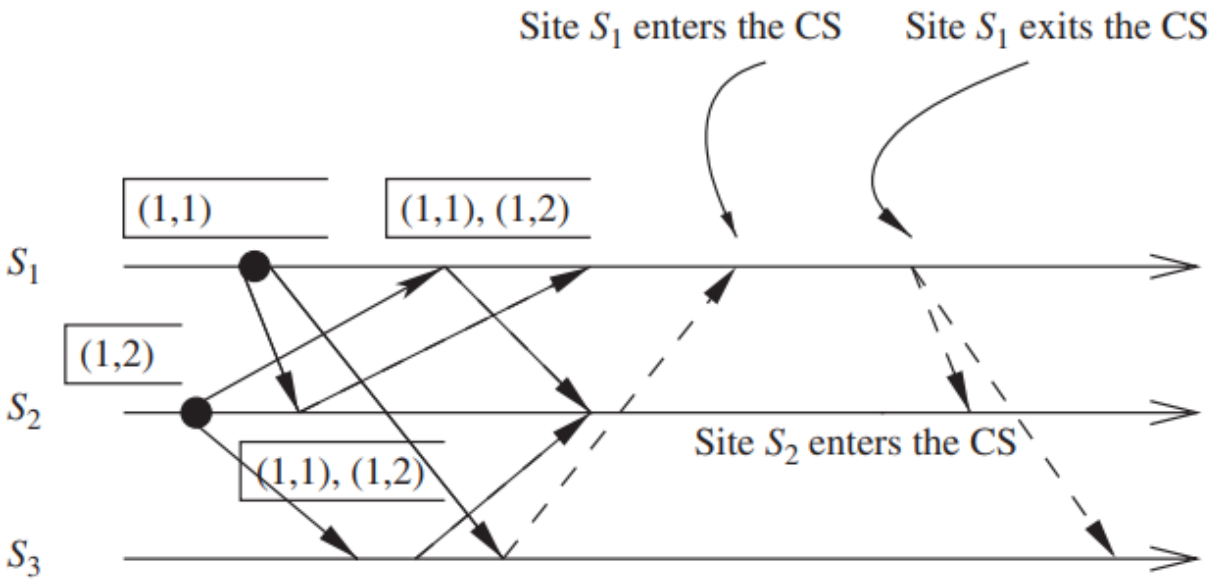
Figure 9.5 Site S_1 exits the CS and sends RELEASE messages.



Lamport's algorithm

- site S2 has received REPLY from all other sites and also received a RELEASE message from site S1.
- Site S2 updates its request_queue and its request is now at the top of its request_queue. Consequently, it enters the CS next

Figure 9.6 Site S_2 enters the CS.



Ricart–Agrawala algorithm

- The Ricart–Agrawala algorithm assumes that 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.
- Timestamps are used to decide the priority of requests in case of conflict

Ricart–Agrawala algorithm

- if a process pi that is waiting to execute the critical section receives a REQUEST message from process pj,

  - then if the priority of pj's request is lower, pi **defers the REPLY** to pj and sends a REPLY message to pj only after executing the CS for its pending request.

  - Otherwise, pi sends a REPLY message to pj immediately, provided it is currently not executing the CS.

- Each process pi maintains the **request-deferred array, RDi, t**he size of which is the same as the number of processes in the system.

- Initially, ∀i ∀j: Rdi[j] = 0. Whenever pi defers the request sent by pj, it sets RDij = 1, and after it has sent a REPLY message to pj, it sets RDij = 0.

Ricart–Agrawala algorithm

Requesting the critical section

(a) When a site S_i wants to enter the CS, it broadcasts a timestamped REQUEST message to all other sites.

(b) When site S_j receives a REQUEST message from site S_i, it sends a REPLY message to site S_i if site S_j is neither requesting nor executing the CS, or if the site S_j is requesting and S_i's request's timestamp is smaller than site S_j's own request's timestamp. Otherwise, the reply is deferred and S_j sets $RD_j[i] := 1$.

Executing the critical section

(c) Site S_i enters the CS after it has received a REPLY message from every site it sent a REQUEST message to.

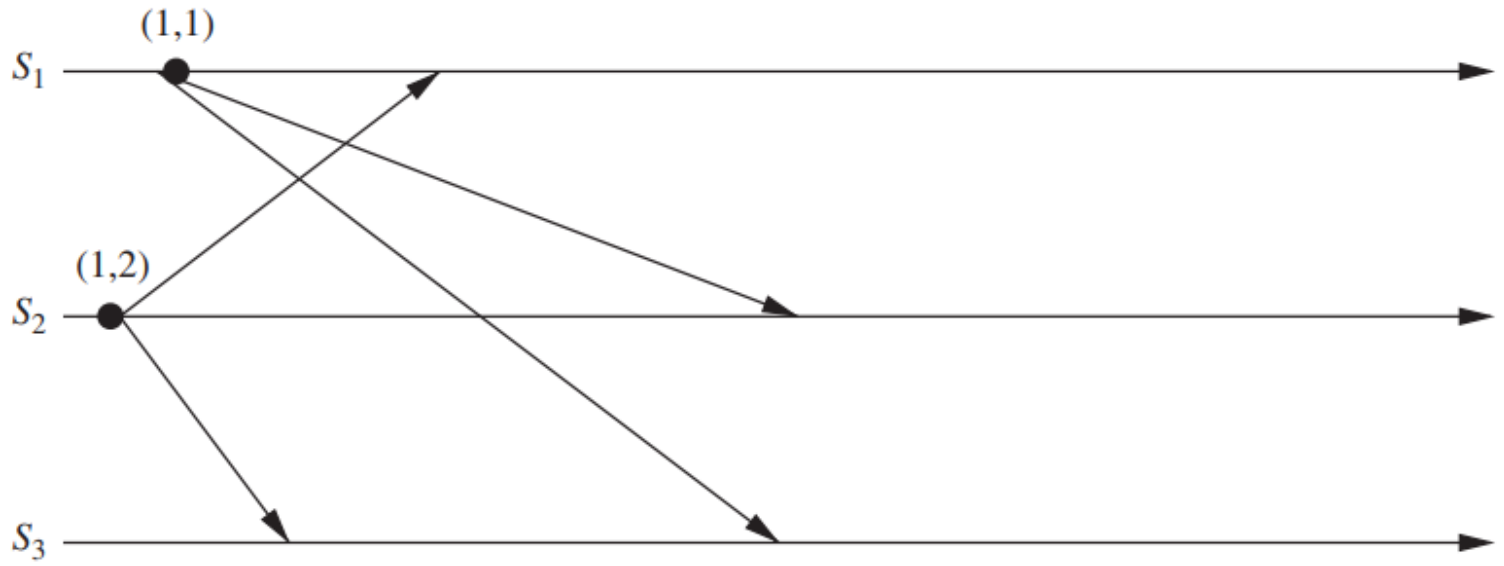
Releasing the critical section

(d) When site S_i exits the CS, it sends all the deferred REPLY messages: $\forall j$ if $RD_i[j] = 1$, then sends a REPLY message to S_j and sets $RD_i[j] := 0$.

Algorithm 9.2 The Ricart–Agrawala algorithm.

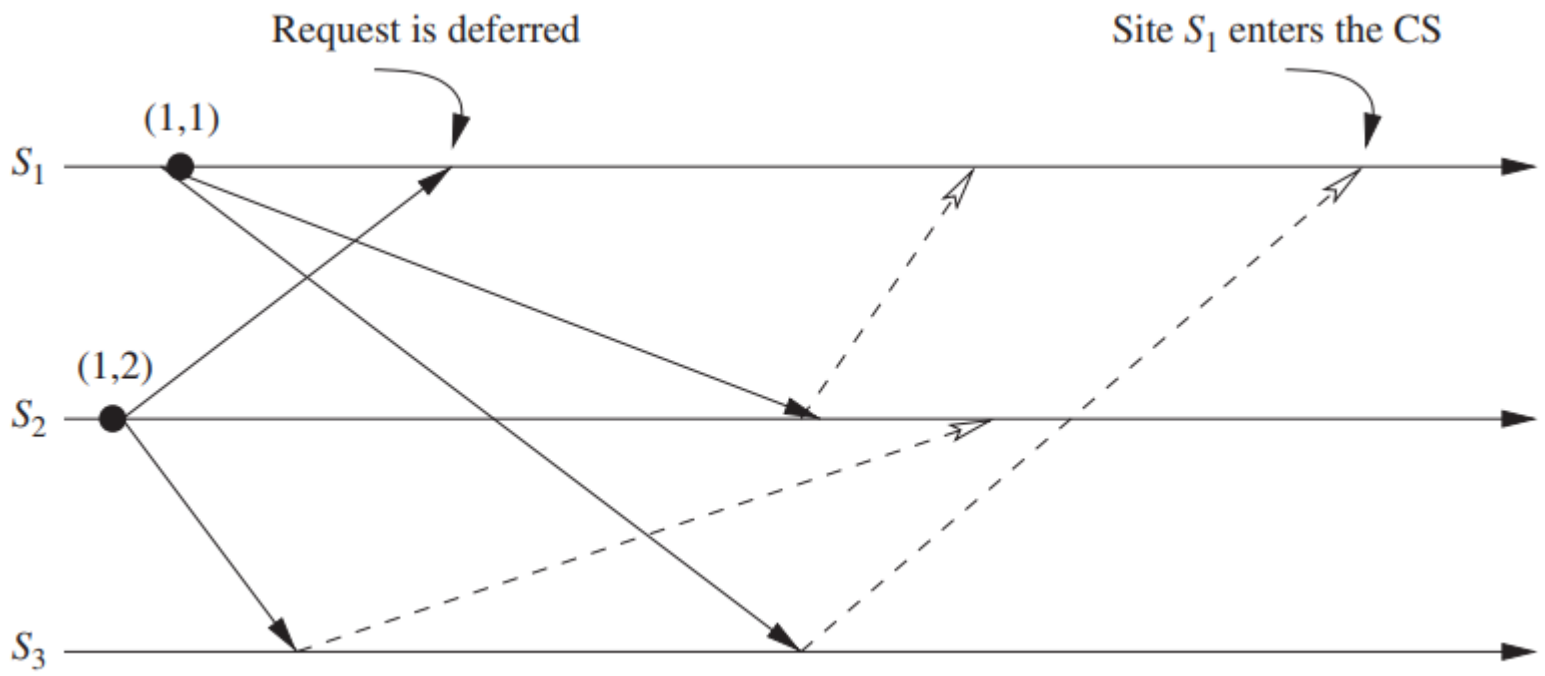
Operation of the Ricart–Agrawala algorithm

Figure 9.7 Sites S_1 and S_2 each make a request for the CS.



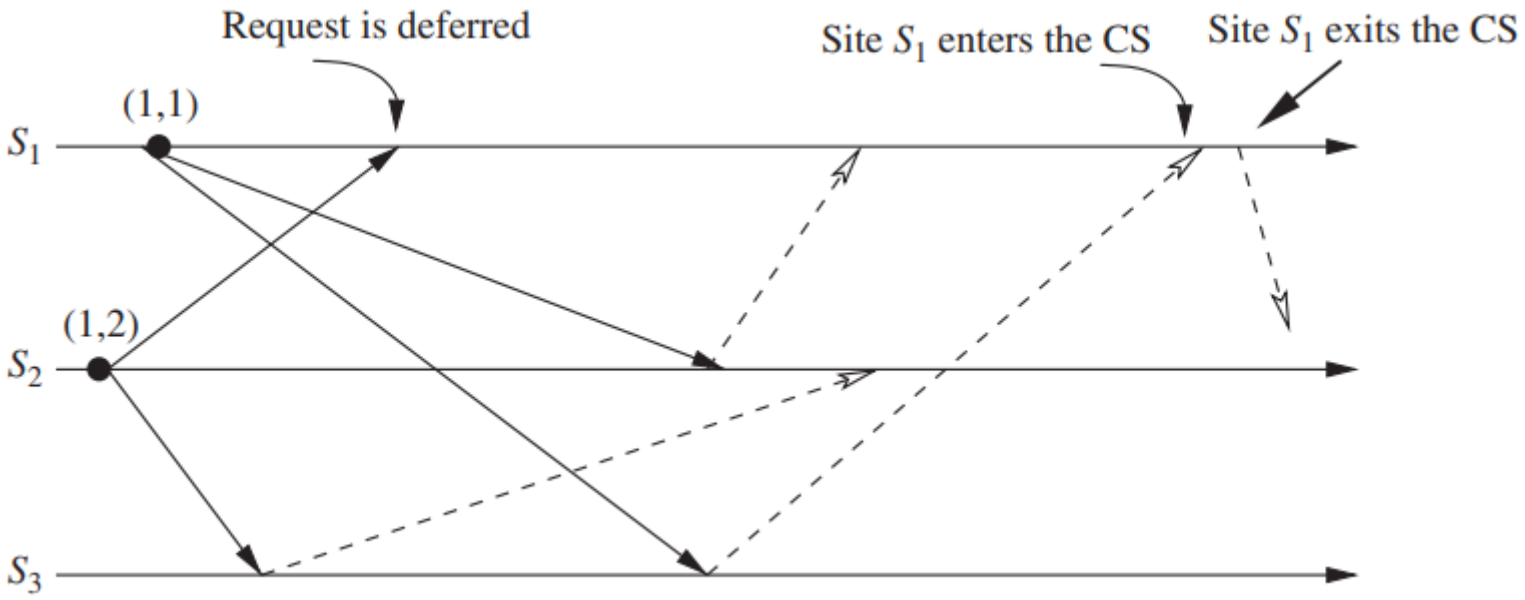
Operation of the Ricart–Agrawala algorithm

Figure 9.8 Site S_1 enters the CS.



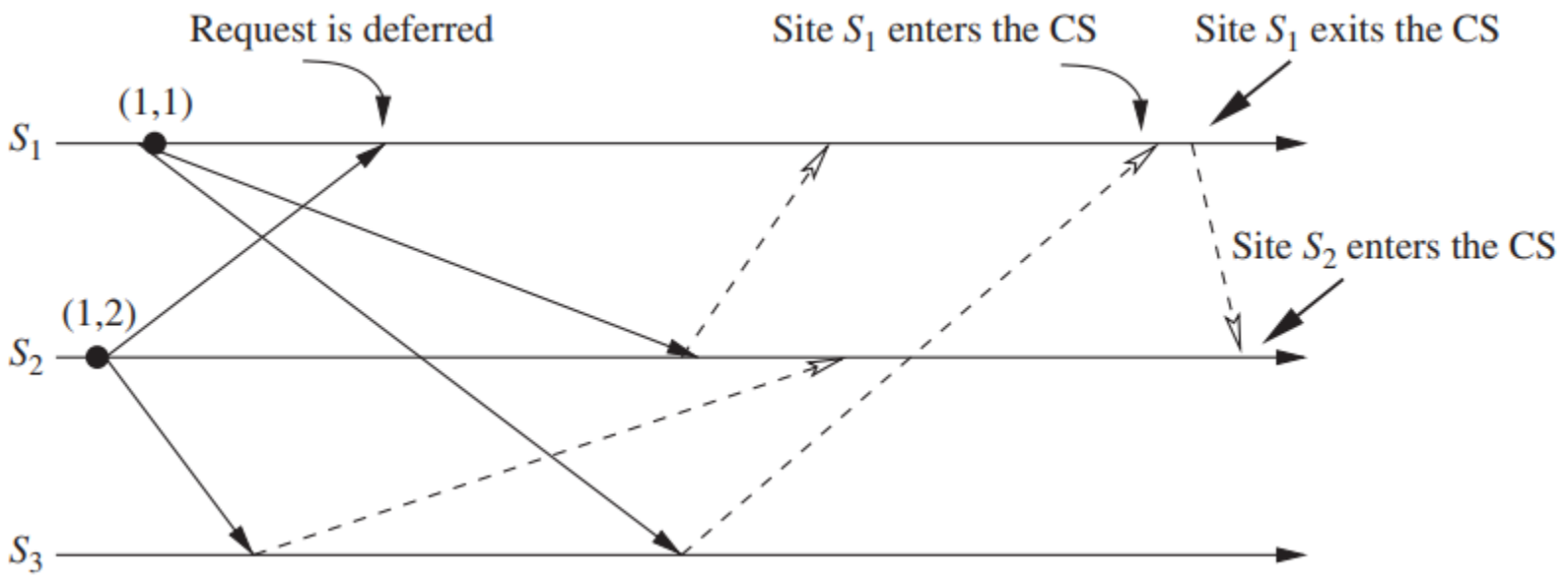
Operation of the Ricart–Agrawala algorithm

Figure 9.9 Site S_1 exits the CS and sends a REPLY message to S_2's deferred request.



Operation of the Ricart–Agrawala algorithm

Figure 9.10 Site S_2 enters the CS.



Ricart–Agrawala algorithm

- In the Ricart–Agrawala algorithm, for every requesting pair of sites, the site with higher priority request will always defer the request of the lower priority site.
- At any time only the highest priority request succeeds in getting all the needed REPLY messages
- For each CS execution, the Ricart–Agrawala algorithm requires N − 1 REQUEST messages and N − 1 REPLY messages.
- Thus, it requires 2(N −1) messages per CS execution

Quorum-based mutual exclusion algorithms

- Quorum-based mutual exclusion algorithms respresented a departure from the trend in the following two ways:

- A site does not request permission from all other sites, but only from a subset of the sites.

the request set of sites are chosen such that

$$\forall i \; \forall j : 1 \leq i, j \leq N :: R_i \cap R_j \neq \Phi$$

- This is a radically different approach as compared to the Lamport and Ricart–Agrawala algorithms, where all sites participate in conflict resolution of all other sites

- In quorum-based mutual exclusion algorithm, a site can send out only one REPLY message at any time.

- A site can send a REPLY message only after it has received a RELEASE message for the previous REPLY message.

- Therefore, a site Si locks all the sites in Ri in exclusive mode before executing its CS.

Quorum-based mutual exclusion algorithms

- Quorum-based mutual exclusion algorithms→ reduce the message complexity of invoking mutual exclusion →sites ask permission from only a subset of sites
- These algorithms are based on the notion of “Coteries” and “Quorums”.

- A **coterie** C is defined as a set of sets, where each set g ∈C is called a **quorum**. The **properties** for quorums in a coterie:

  - **Intersection property** For every quorum g, $h \in C$, $g \cap h \neq \emptyset$.
    For example, sets {1,2,3}, {2,5,7}, and {5,7,9} cannot be quorums in a coterie because the first and third sets do not have a common element.
  - **Minimality property** There should be no quorums g, h in coterie C such that $g \supseteq h$. For example, sets {1,2,3} and {1,3} cannot be quorums in a coterie because the first set is a superset of the second.

- Coteries and quorums can be used to develop algorithms to ensure mutual exclusion in a distributed environment

Quorum-based mutual exclusion algorithms

- A simple protocol works as follows: let “a” be a site in quorum “A.”

- If “a” wants to invoke mutual exclusion, it requests permission from all sites in its quorum “A.”
- Every site does the same to invoke mutual exclusion. Due to the Intersection property, quorum “A” contains at least one site that is common to the quorum of every other site. These common sites send permission to only one site at any time. Thus, mutual exclusion is guaranteed

- Minimality property ensures efficiency
- Quorums are formed as sets that contain a majority of sites.
- **Maekawa** used the theory of projective planes to develop quorums of **size** $\sqrt{N}$

Maekawa's algorithm

- First quorum-based mutual exclusion algm
- The request sets for sites (i.e., quorums) are constructed →conditions:

M1 $(\forall i\ \forall j : i \neq j, 1 \leq i, j \leq N :: R_i \cap R_j \neq \phi)$.

M2 $(\forall i : 1 \leq i \leq N :: S_i \in R_i)$.

M3 $(\forall i : 1 \leq i \leq N :: |R_i| = K)$.

M4 Any site S_j is contained in K number of R_is, $1 \leq i, j \leq N$.

Maekawa's algorithm

- Conditions M1 and M2 are necessary for correctness
- Condition M3 states that the size of the requests sets of all sites must be equal
  - Site->equal amount of work to invoke Mutual exclusion

- Condition M4 enforces that exactly the same number of sites should request permission from any site, which implies that all sites have "equal responsibility" in granting permission to other sites
- This algorithm requires delivery of messages to be in the order they are sent between every pair of sites.

Maekawa's algorithm

Requesting the critical section:

(a) A site S_i requests access to the CS by sending REQUEST(i) messages to all sites in its request set R_i.

(b) When a site S_j receives the REQUEST(i) message, it sends a REPLY(j) message to S_i provided it hasn't sent a REPLY message to a site since its receipt of the last RELEASE message. Otherwise, it queues up the REQUEST(i) for later consideration.

Executing the critical section:

(c) Site S_i executes the CS only after it has received a REPLY message from every site in R_i.

Releasing the critical section:

(d) After the execution of the CS is over, site S_i sends a RELEASE(i) message to every site in R_i.

(e) When a site S_j receives a RELEASE(i) message from site S_i, it sends a REPLY message to the next site waiting in the queue and deletes that entry from the queue. If the queue is empty, then the site updates its state to reflect that it has not sent out any REPLY message since the receipt of the last RELEASE message.

Algorithm 9.5 Maekawa's algorithm.

Token-based algorithms

- In token-based algorithms, a unique token is shared among the sites.
- A site is allowed to enter its CS if it possesses the token.
- A site holding the token can enter its CS repeatedly until it sends the token to some other site.
- Depending upon the way a site carries out the search for the token, there are numerous token-based algorithms
- token-based algorithms use sequence numbers instead of timestamps.
- Every request for the token contains a sequence number

Suzuki–Kasami's broadcast algorithm

- In Suzuki–Kasami's algorithm if a site that wants to enter the CS does not have the token, it broadcasts a REQUEST message for the token to all other sites.
- A site that possesses the token sends it to the requesting site upon the receipt of its REQUEST message.
- If a site receives a REQUEST message when it is executing the CS, it sends the token only after it has completed the execution of the CS
- Although the basic idea underlying this algorithm may sound rather simple, there are two design issues that must be efficiently addressed:
- **1.** How to distinguishing an outdated REQUEST message from a current REQUEST message
- **2.** How to determine which site has an outstanding request for the CS

Suzuki–Kasami's broadcast algorithm

Requesting the critical section:

(a) If requesting site S_i does not have the token, then it increments its sequence number, $RN_i[i]$, and sends a REQUEST(i, sn) message to all other sites. ("sn" is the updated value of $RN_i[i]$.)

(b) When a site S_j receives this message, it sets $RN_j[i]$ to $max(RN_j[i], sn)$. If S_j has the idle token, then it sends the token to S_i if $RN_j[i] = LN[i] + 1$.

Executing the critical section:

(c) Site S_i executes the CS after it has received the token.

Releasing the critical section: Having finished the execution of the CS, site S_i takes the following actions:

(d) It sets $LN[i]$ element of the token array equal to $RN_i[i]$.

(e) For every site S_j whose i.d. is not in the token queue, it appends its i.d. to the token queue if $RN_i[j] = LN[j] + 1$.

(f) If the token queue is nonempty after the above update, S_i deletes the top site i.d. from the token queue and sends the token to the site indicated by the i.d.

Algorithm 9.7 Suzuki–Kasami's broadcast algorithm.