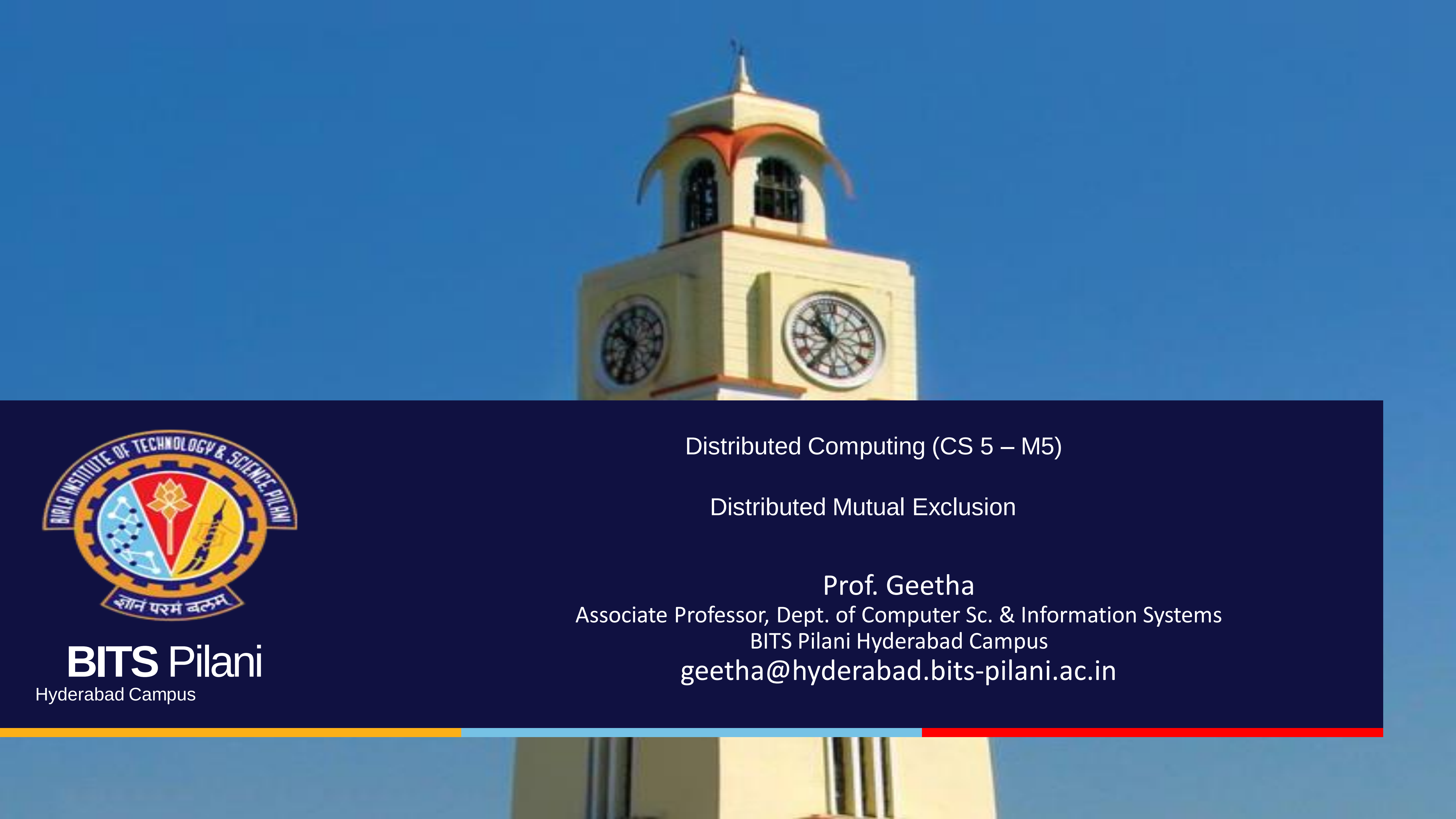


BITS Pilani
Hyderabad Campus

Distributed Computing (CS 5 – M5)

Distributed Mutual Exclusion

Prof. Geetha
Associate Professor, Dept. of Computer Sc. & Information Systems
BITS Pilani Hyderabad Campus
geetha@hyderabad.bits-pilani.ac.in

What is mutual exclusion?

- ❖Mutual Exclusion also known as Mutex was first identified by Dijkstra
- ❖When a process is accessing a shared variable, it is said to be in a critical section (code segment)
- ❖When no two processes can be in Critical Section at the same time, this state is known as Mutual Exclusion which is a property of concurrency control and is used to prevent race condition (happens when the result of multiple thread execution in critical section differs according to the order in which the threads execute)

Mutual Exclusion Devices:-

Locks, recursive locks, semaphores, monitor, message passing etc.,

Mutual exclusion in synchronization

- **Mutual exclusion** is a property of process synchronization which states that "no two processes can exist in the critical section at any given point of time"

- Any process synchronization technique being used must satisfy the property of mutual exclusion, without which it would not be possible to get rid of a race condition

- During concurrent execution of processes, processes need to enter the critical section (or the section of the program shared across processes) at times for execution

- It might so happen that because of the execution of multiple processes at once, the values stored in the critical section become inconsistent; in other words, the values depend on the sequence of execution of instructions – also known as a race condition

- The primary task of process synchronization is to get rid of race conditions while executing the critical section

Mutual Exclusion in Distributed Computing

Mutual exclusion is a concurrency control property which is introduced to prevent race conditions; the requirement that a process can not enter its critical section while another concurrent process is currently present or executing in its critical section i.e., only one process is allowed to execute the critical section at any given instance of time

In single computer system, memory and other resources are shared between different processes. The status of shared resources and the status of users is easily available in the shared memory so with the help of shared variable (For example: Semaphores) mutual exclusion problem can be easily solved

In Distributed systems, we neither have shared memory nor a common physical clock and therefore we can not solve mutual exclusion problem using shared variables. To solve the mutual exclusion problem in distributed systems, message passing is used

A site in distributed system does not have the complete information of the state of the system due to lack of shared memory and a common physical clock

Mutual Exclusion

- for distributed systems –
  - decision as to which process is allowed access to the CS next is made by message passing
  - must deal with unpredictable message delays and incomplete knowledge of the system state

Mutual Exclusion

- ❑ ensures that concurrent access of processes to a shared resource or data is serialized
- ❑ executed in a mutually exclusive manner
- ❑ for distributed system –
  - ❑ only one process is allowed to execute the critical section (CS) at any given time
  - ❑ semaphores or a local kernel cannot be used to implement mutual exclusion

Three approaches for mutual exclusion in distributed computing systems

- ❑ Token based
- ❑ Non-token based
- ❑ Quorum based

Token-Based Approach

- a unique token is shared among the sites
- token is also called PRIVILEGE message
- a site is allowed to enter its CS if
  - it possesses the token
  - it continues to hold the token until the execution of the CS is over
- mutual exclusion is ensured because the token is unique

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 CS when an assertion becomes true

- mutual exclusion is enforced because the assertion becomes true only at one site at any given time

Quorum-Based Approach

- each site requests permission to execute the CS from a subset of sites
- subset of sites is called quorum
- 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
  - this site is responsible to make sure that only one request executes the CS at any time

System Model for Mutual Exclusion

- ✓ system consists of N sites, S_1, S_2,…., S_N
- ✓ without loss of generality, assume that a single process is running on each site
- ✓ process at site S_i is p_i
- ✓ processes communicate asynchronously over an underlying communication network
- ✓ any process wishing to enter the CS
  - ✓ requests all other or a subset of processes by sending REQUEST messages
  - ✓ waits for appropriate replies before entering the CS
- ✓ while waiting the process is not allowed to make further requests to enter the CS

System Model for Mutual Exclusion contd..

- ❑ site can be in one of the following 3 states:
  - ❑ requesting the CS
  - ❑ executing the CS
  - ❑ neither of the 2
- ❑ "requesting the CS" state - site is blocked and cannot make further requests for the CS
- ❑ "idle" state - site is executing outside the CS
- ❑ for token-based algorithms
  - ❑ a site can also be in a state where a site holding the token is executing outside the CS
  - ❑ such state is called *idle token state*

System Model for mutual exclusion contd..

- 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
- nature of channels (FIFO or not) is algorithm specific
- assume that:
  - channels reliably deliver all messages
  - sites do not crash
  - network does not get partitioned
- **timestamps are used to decide the priority of requests in case of a conflict**
- *general rule - smaller the timestamp of a request, the higher its priority to execute the CS*

System Model contd..

Notations:

- ❑ N - number of processes or sites involved in invoking the critical section

- ❑ T - average message delay

- ❑ E - average critical section execution time

Requirements of mutual exclusion algorithms

- ❑ **No Deadlock:**

Two or more sites should not endlessly wait for any message that will never arrive

- ❑ **No Starvation:**

Every site which wants to execute critical section should get an opportunity to execute it in finite time. Any site should not wait indefinitely to execute critical section while other sites are repeatedly executing the critical section

- ❑ **Fairness:**

Each site should get a fair chance to execute the critical section. Any request to execute the critical section must be executed in the order they are made i.e Critical section execution requests should be executed in the order of their arrival in the system

- ❑ **Fault Tolerance:**

In case of failure, it should be able to recognize it by itself in order to continue functioning without any disruption

Properties of Mutual Exclusion Algorithms

A mutual exclusion algorithm should satisfy:

- Safety Property – absolutely necessary
- Liveness Property – important
- Fairness – important

Requirements of Mutual Exclusion Algorithms

- ❑ **Safety property**
  - ❑ at any instant, only one process can execute the critical section
  - ❑ absolutely necessary property
- ❑ **Liveness property**
  - ❑ absence of deadlock and starvation
  - ❑ a site must not wait indefinitely to execute the CS while other sites are repeatedly executing the CS
  - ❑ every requesting site should get an opportunity to execute the CS in finite time

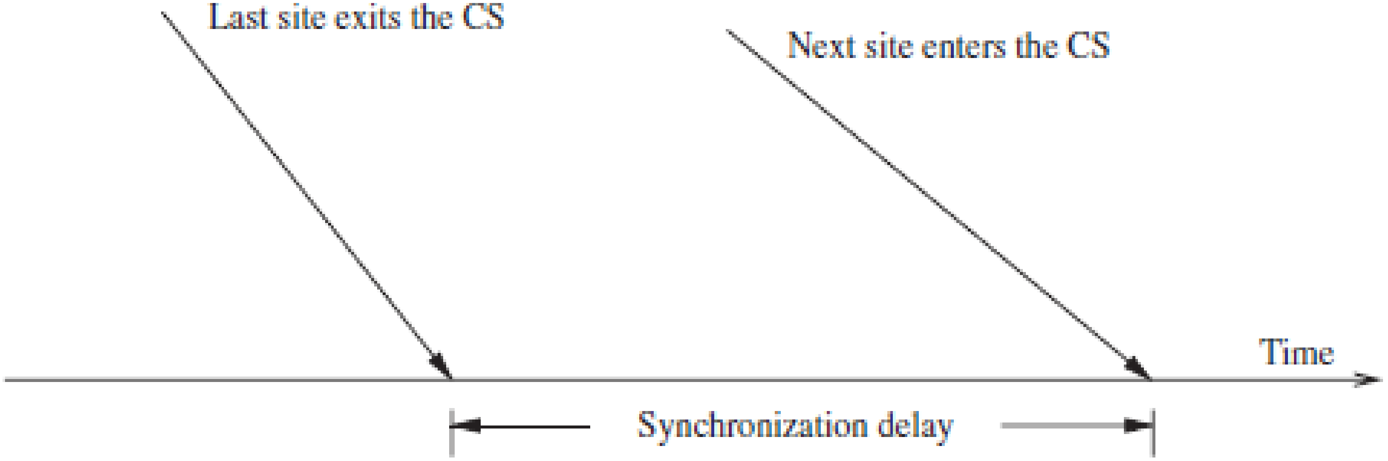
Mutual Exclusion Algorithms

- ❑ **Fairness**

  - ❑ each process gets a fair chance to execute the CS
  - ❑ CS execution requests are executed in order of their arrival in the system
  - ❑ time is determined by a logical clock

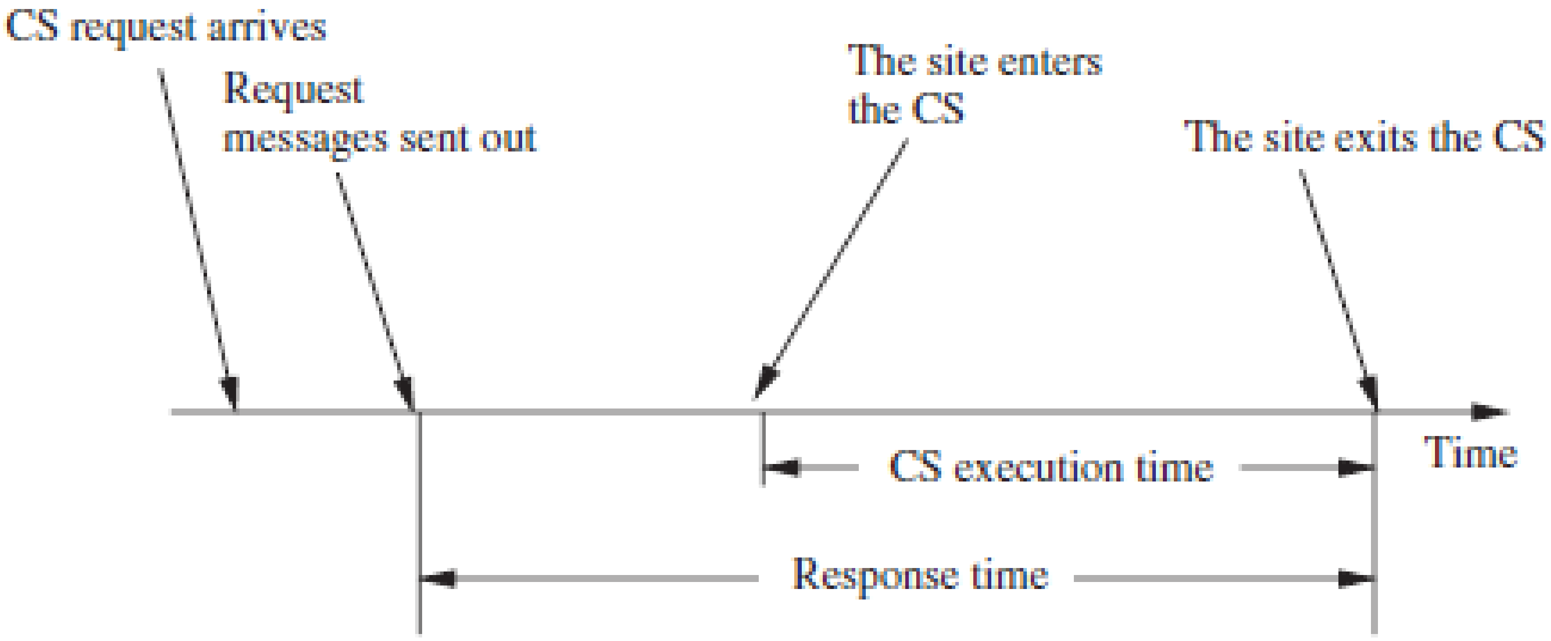
Performance Metrics

- ❑ **Message complexity** - number of messages that are required per CS execution by a site
- ❑ **Synchronization delay** –
  - ❑ after a site leaves the CS, the time required before the next site enters the CS
  - ❑ one or more sequential message exchanges may be required after a site exits the CS and before the next site can enter the CS



Performance Metrics

- ❑ **Response time** – time interval a request waits for its CS execution to be over after its request messages have been sent out
- ❑ **System throughput** -
  - ❑ rate at which the system executes requests for the CS
  - ❑ SD is Synchronization Delay
  - ❑ E is average critical section execution time
  - ❑ **System throughput = 1/(SD+E)**



Performance Metrics

Low and high load performance –

- ❑ "low load" – not more than one request for the critical section present in the system simultaneously
- ❑ "high load" -
  - ❑ there is always a pending request for critical section at a site
  - ❑ after having executed a request, a site immediately initiates activities to execute its next CS request
  - ❑ a site is seldom in the idle state

Lamport’s Algorithm

- time is determined by logical clocks
- when a site processes a request for the CS, it
  - updates its local clock
  - assigns the request a timestamp
- algorithm is fair - executes CS requests in the increasing order of timestamps
- every site S_i keeps a queue, request_queue$_i$
- request_queue$_i$ contains mutual exclusion requests ordered by their timestamps

Lamport’s Algorithm

- algorithm requires communication channels to deliver messages in FIFO order
- when a site removes a request from its request queue
  - its own request may come at the top of the queue
  - enables 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

Lamport's Algorithm

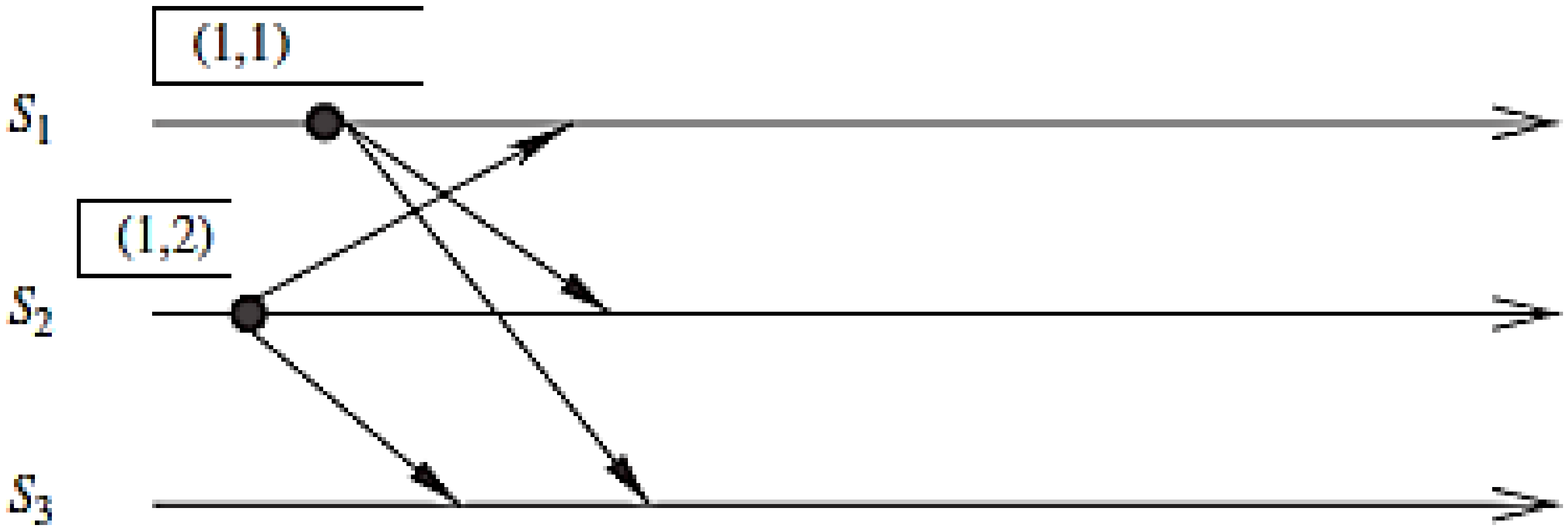
- **Requesting the critical section:**
- When a site S_i wants to enter the CS, it broadcasts a REQUEST(ts_i, i) message to all other sites and places the request on request_queue$_i$. ((ts_i, i) denotes the timestamp of the request)
- When a site S_j receives the REQUEST(ts_i, i) message from site S_i, it places site S_i's request on request_queue$_j$ and returns a timestamped REPLY message to S_i

Lamport’s Algorithm

- ❑ **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 the request_queue$_i$

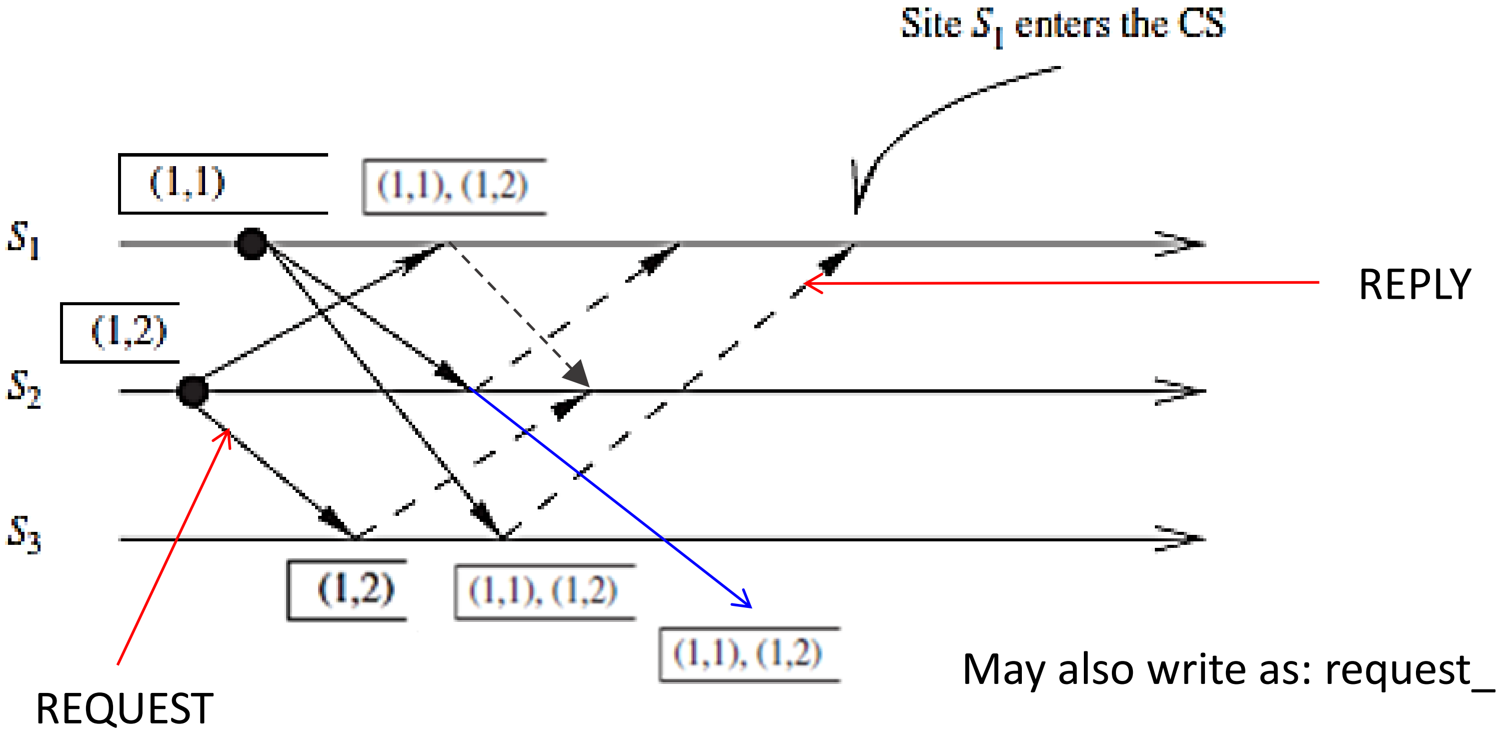
- ❑ **Releasing the critical section:**
- ❑ • Site S_i, upon exiting the CS, removes its request from the top of its request queue and broadcasts a timestamped RELEASE message to all other sites
- ❑ • When a site S_j receives a RELEASE message from site S_i, it removes S_i’s request from its request queue

Lamport’s Algorithm



Sites S_1 and S_2 make requests for the CS

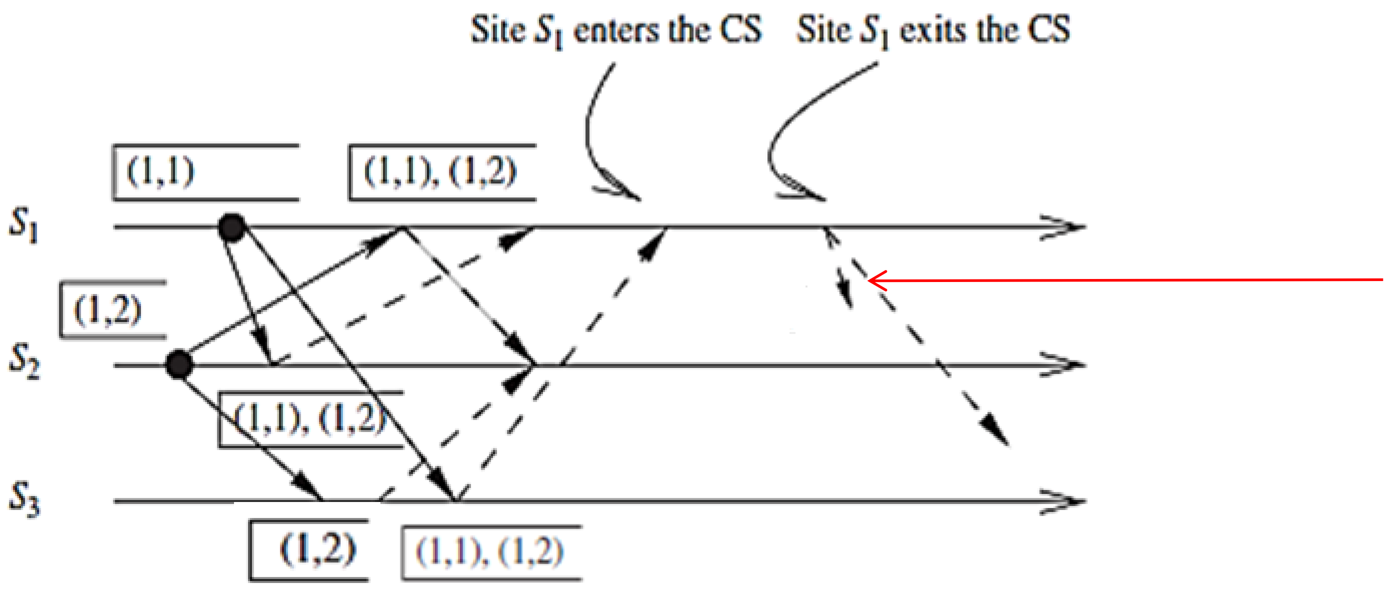
Lamport's Algorithm



May also write as: request_queue$_1$ = ((1, 1), (1, 2))

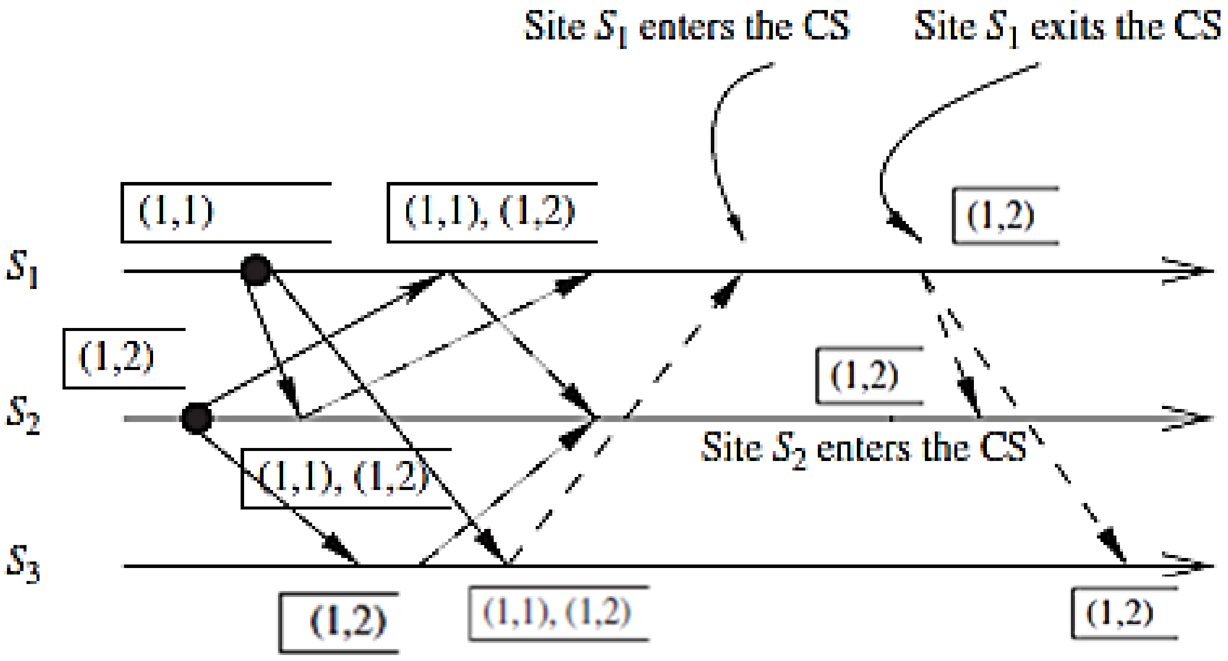
Site S_1 enters the CS

Lamport's Algorithm



Site S_1 exits the CS and sends RELEASE messages

Lamport’s Algorithm



Site S_2 enters the CS

Lamport’s Algorithm

- ❑ **Correctness**
- ❑ *Lamport’s algorithm achieves mutual exclusion*
- ❑ *Lamport’s algorithm is fair - requests for CS are executed in the order of their timestamps*
- ❑ **Performance**
- ❑ For each CS execution, requires (N – 1) REQUEST messages, (N – 1) REPLY messages, and (N – 1) RELEASE messages
- ❑ Requires 3(N – 1) messages per CS invocation
- ❑ Synchronization delay in the algorithm is T (T = avg. message delay)

Ricart–Agrawala Algorithm

- communication channels are not required to be FIFO
- uses two types of messages:
  - REQUEST
  - 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

Ricart–Agrawala Algorithm

- timestamps are used to decide the priority of requests in case of conflict –
  - if a process p_i that is waiting to execute the CS receives a REQUEST message from process p_j , then if the priority of p_j's request is lower, p_i defers the REPLY to p_j and sends a REPLY message to p_j only after executing the CS for its pending request
  - otherwise, p_i sends a REPLY message to p_j immediately, provided it is currently not executing the CS
- if several processes are requesting execution of the CS, the highest priority request succeeds in collecting all the needed REPLY messages and gets to execute the CS

Ricart–Agrawala Algorithm

- each process p_i maintains the request-deferred array, RD_i
- size of RD_i = no. of processes in the system
- initially, ∀i ∀j: $RD_i[j] = 0$
- whenever p_i defers the request sent by p_j, it sets $RD_i[j] = 1$,
- after it has sent a REPLY message to p_j , it sets $RD_i[j] = 0$
- when a site receives a message, it updates its clock using the timestamp in the message
- when a site takes up a request for the CS for processing, it
  - updates its local clock
  - assigns a timestamp to the request
- execution of the CS requests is always in the order of their timestamps

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

Ricart–Agrawala Algorithm

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

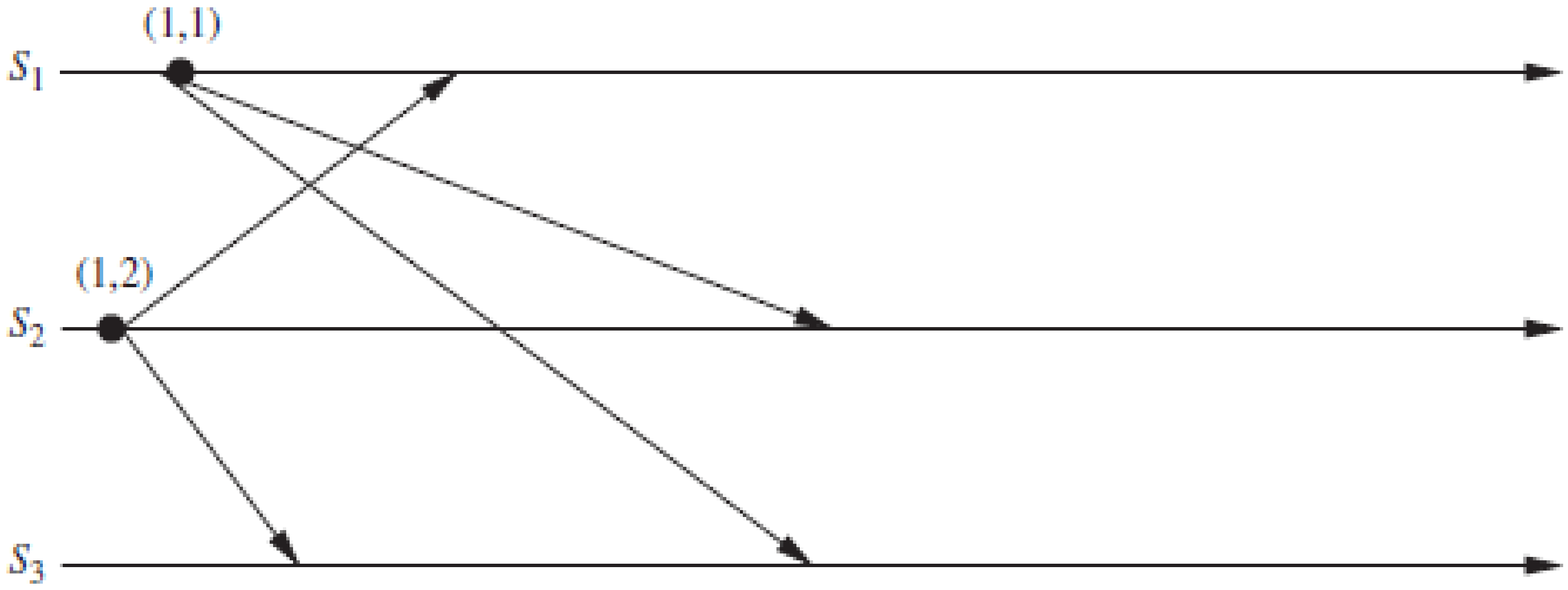
Releasing the critical section:

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

Correctness

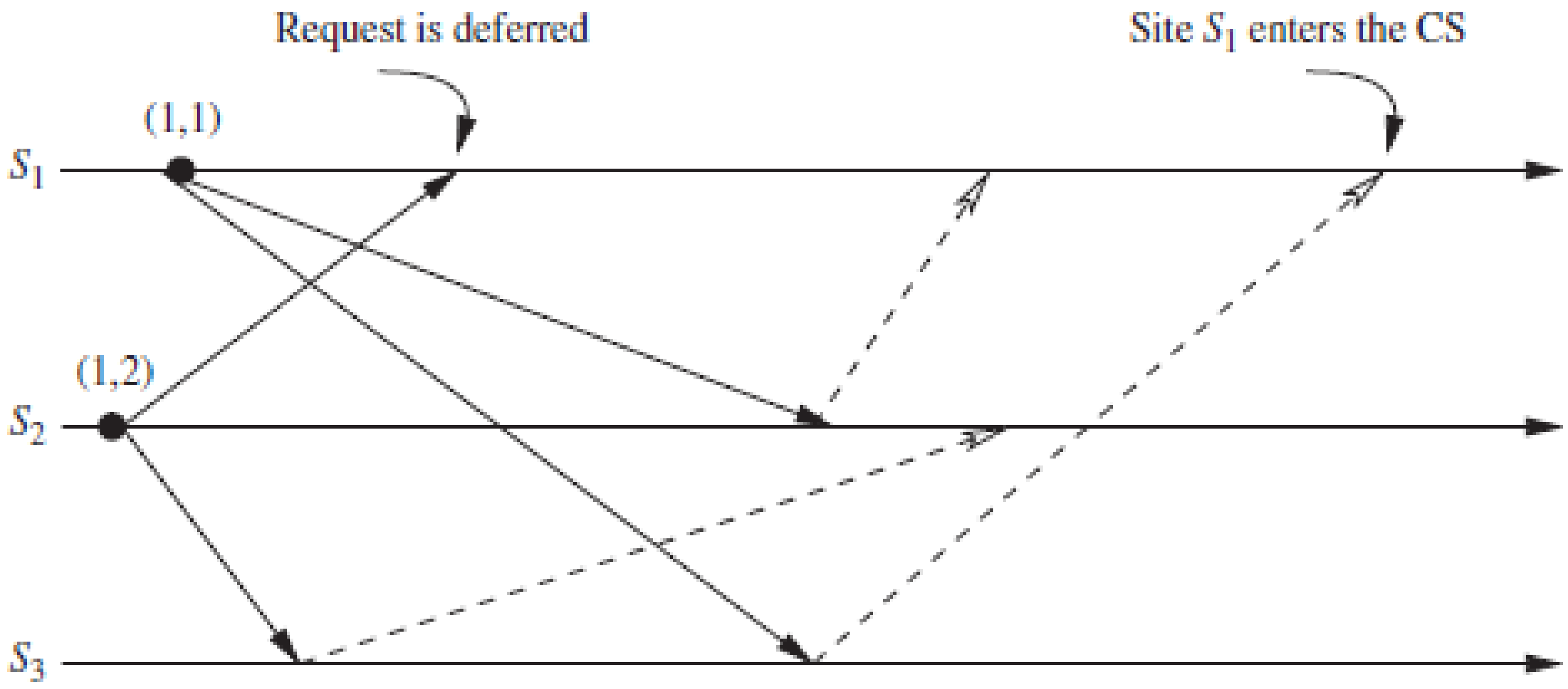
•*Ricart–Agrawala algorithm achieves mutual exclusion*

Ricart–Agrawala Algorithm



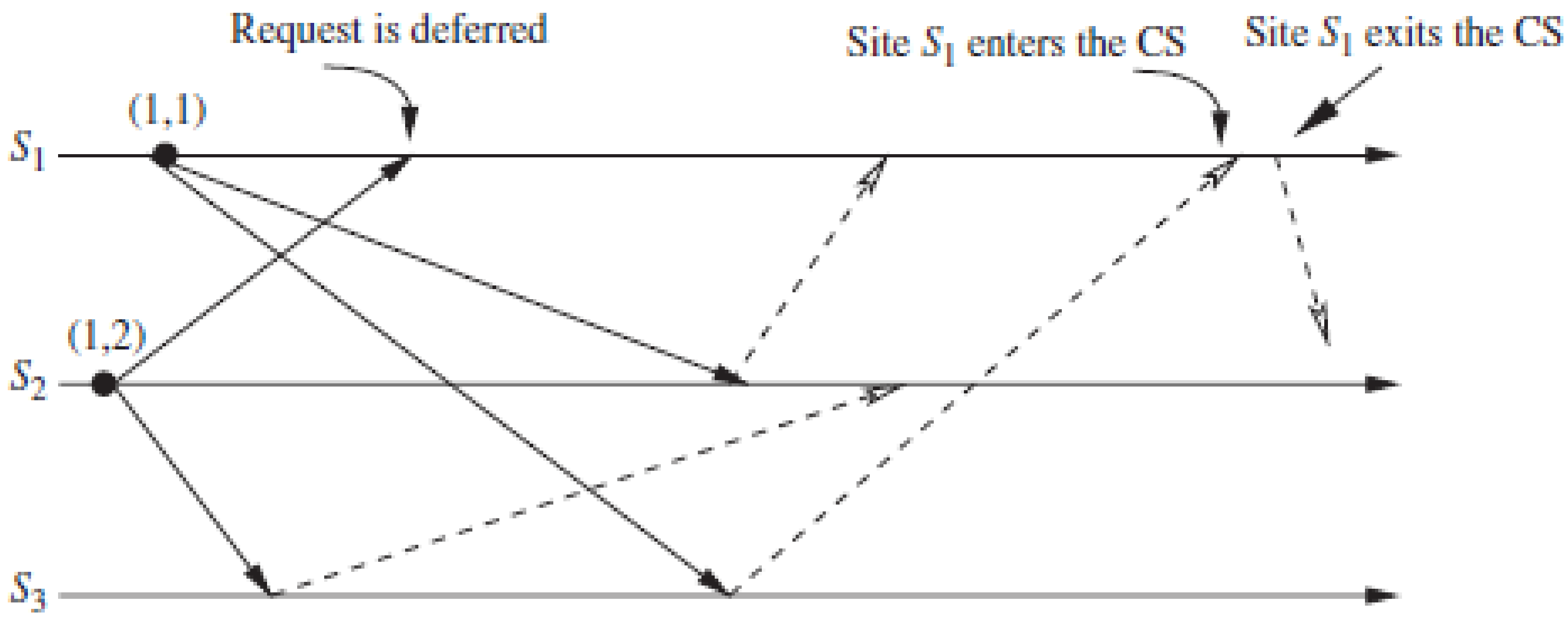
Sites S_1 and S_2 each makes a request for the CS

Ricart–Agrawala Algorithm



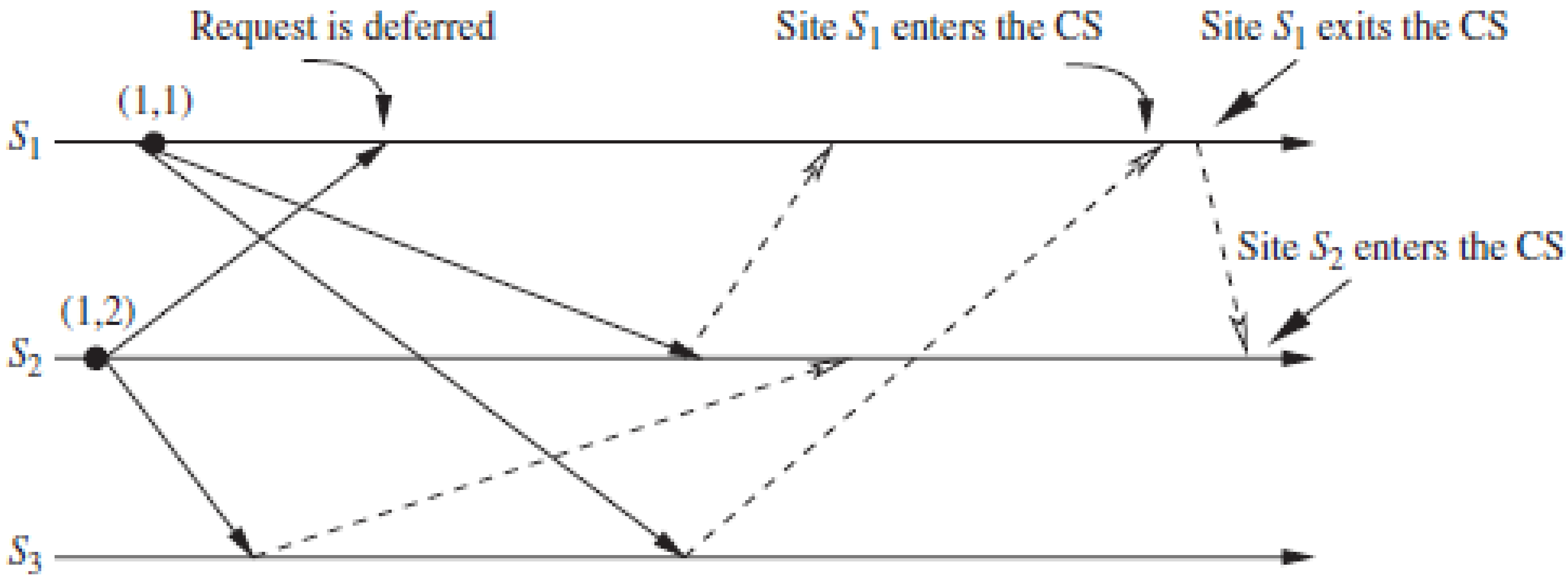
Site S_1 enters the CS

Ricart–Agrawala Algorithm



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

Ricart–Agrawala Algorithm



Site S_2 enters the CS

Ricart–Agrawala Algorithm

Performance

- For each CS execution, requires (N – 1) REQUEST messages and (N – 1) REPLY messages
- requires 2(N – 1) messages per CS execution
- synchronization delay in the algorithm is T

Maekawa’s Algorithm

- quorum-based mutual exclusion algorithm
- request sets for sites (i.e., quorums) are constructed to satisfy the following 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: $(\forall i : 1 \leq i \leq N :: |R_i| = K$ for some $K)$
  - M4: Any site S_j is contained in K number of R_i’s, $1 \leq i, j \leq N$
- Maekawa showed that $N = K(K - 1) + 1$
- This relation gives $|R_i| = K = \sqrt{N}$ (square root of N)

Maekawa’s Algorithm

- ❑ there is at least one common site between the request sets of any two sites (condition M1)
- ❑ every pair of sites has a common site which mediates conflicts between the pair
- ❑ a site can have only one outstanding REPLY message at any time; that is,
  - ❑ it grants permission to an incoming request if it has not granted permission to some other site
  - ❑ mutual exclusion is guaranteed
- ❑ requires delivery of messages to be in the order they are sent between every pair of sites

Maekawa’s Algorithm

- conditions M1 and M2 are necessary for correctness
- M3 ($\forall i : 1 \le i \le N :: |R_i| = K$ for some K) states that the size of the requests sets of all sites must be equal
  - equal amount of work to invoke mutual exclusion
- M4 (Any site S_j is contained in K number of R_i’s, $1 \le i, j \le N$) enforces that exactly the same number of sites should request permission from any site
  - equal responsibility

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.

Maekawa’s Algorithm

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.

Maekawa’s Algorithm

Correctness: *Maekawa’s algorithm achieves mutual exclusion*

Performance:

- size of a request set is √N
- an execution of the CS requires √N REQUEST, √N REPLY, and √N RELEASE messages, resulting in 3√N messages per CS execution
- synchronization delay 2T

Reference

- Ajay D. Kshemkalyani, and Mukesh Singhal, Chapter 9, “Distributed Computing: Principles, Algorithms, and Systems”, Cambridge University Press, 2008 (reprint: 2013).