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.
- 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
- **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.
- **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 . Without loss of generality, we assume that 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

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.

- In the 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 refereed to as 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.

Distributed mutual exclusion algorithms

- We do not make any assumption regarding communication channels if they are FIFO or not.
- This is algorithm specific. We assume that channels reliably deliver all messages, sites do not crash, and the network does not get partitioned

Requirements of mutual exclusion algorithms

A mutual exclusion algorithm should satisfy the following 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.

Distributed mutual exclusion algorithms

Two or more sites should not endlessly wait for messages that will never arrive.

In addition, 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.

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 property generally means that the CS execution requests are executed in order of their arrival in the system

Distributed mutual exclusion algorithms

Performance metrics

The performance of mutual exclusion algorithms is generally measured by the following four metrics:

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

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 in the sense that a request for CS are executed in the order of their timestamps and time is determined by logical clocks.

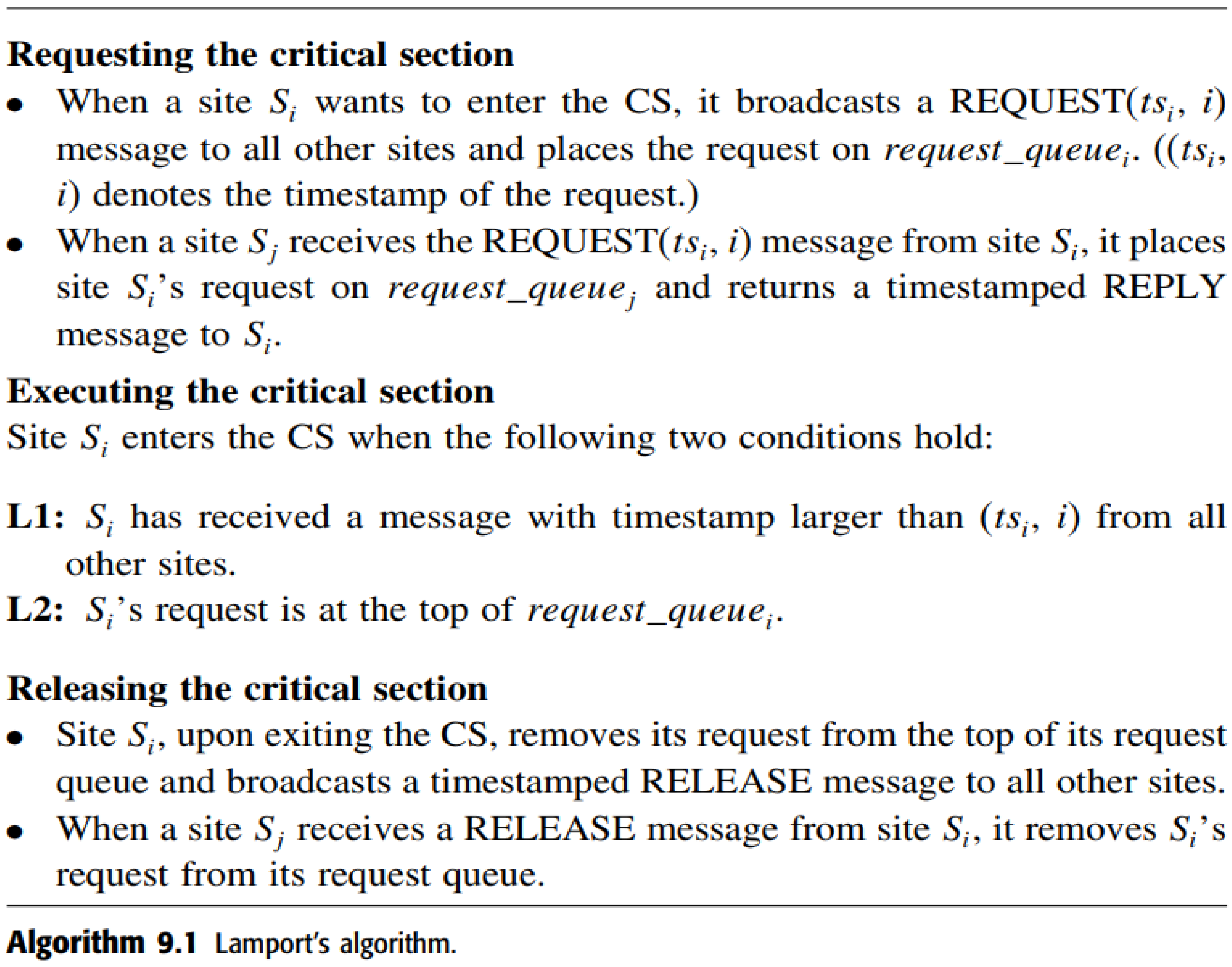
When a site processes a request for the CS, it updates its local clock and assigns the request a timestamp.

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.

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,

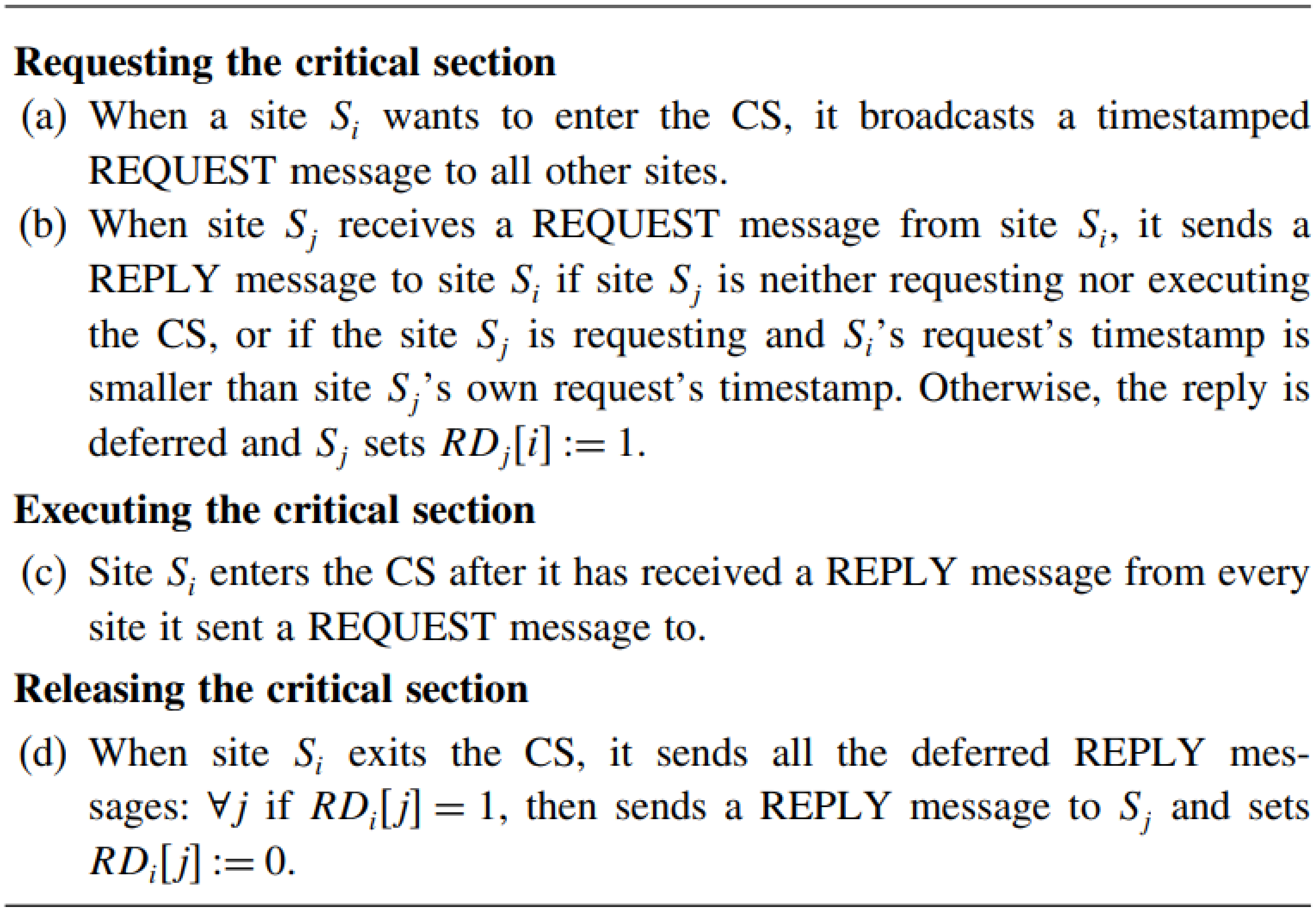
- then if the priority of pj's request is lower, pi defers the REPLY to pj and sends a REPLY message to pj only after executing the CS for its pending request.

- 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, the size of which is the same as the number of processes in the system.

- Initially, $\forall i\ \forall j$: Rdi[j] = 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.

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.
- 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 significantly reduce the message complexity of invoking mutual exclusion by having sites ask permission from only a subset of sites.

- Since these algorithms are based on the notion of "Coteries" and "Quorums," we first describe the idea of coteries and quorums.

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

- Intersection property

- Minimality property

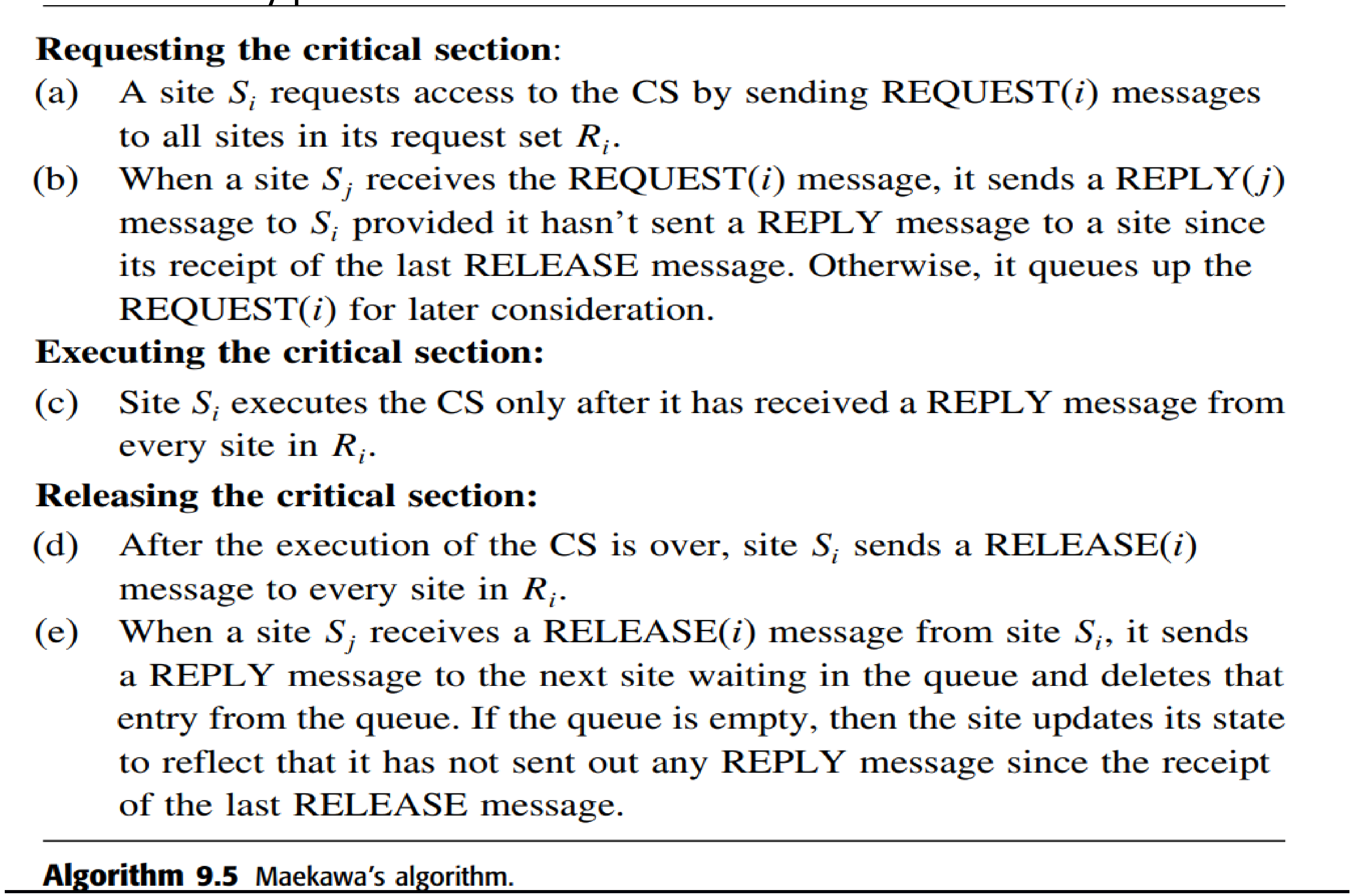
- Coteries and quorums can be used to develop algorithms to ensure mutual exclusion in a distributed environment

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.”
- Minimality property ensures efficiency

Maekawa's algorithm

- Maekawa's algorithm was the first quorum-based mutual exclusion algorithm.
- This algorithm requires delivery of messages to be in the order they are sent between every pair of sites.



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.

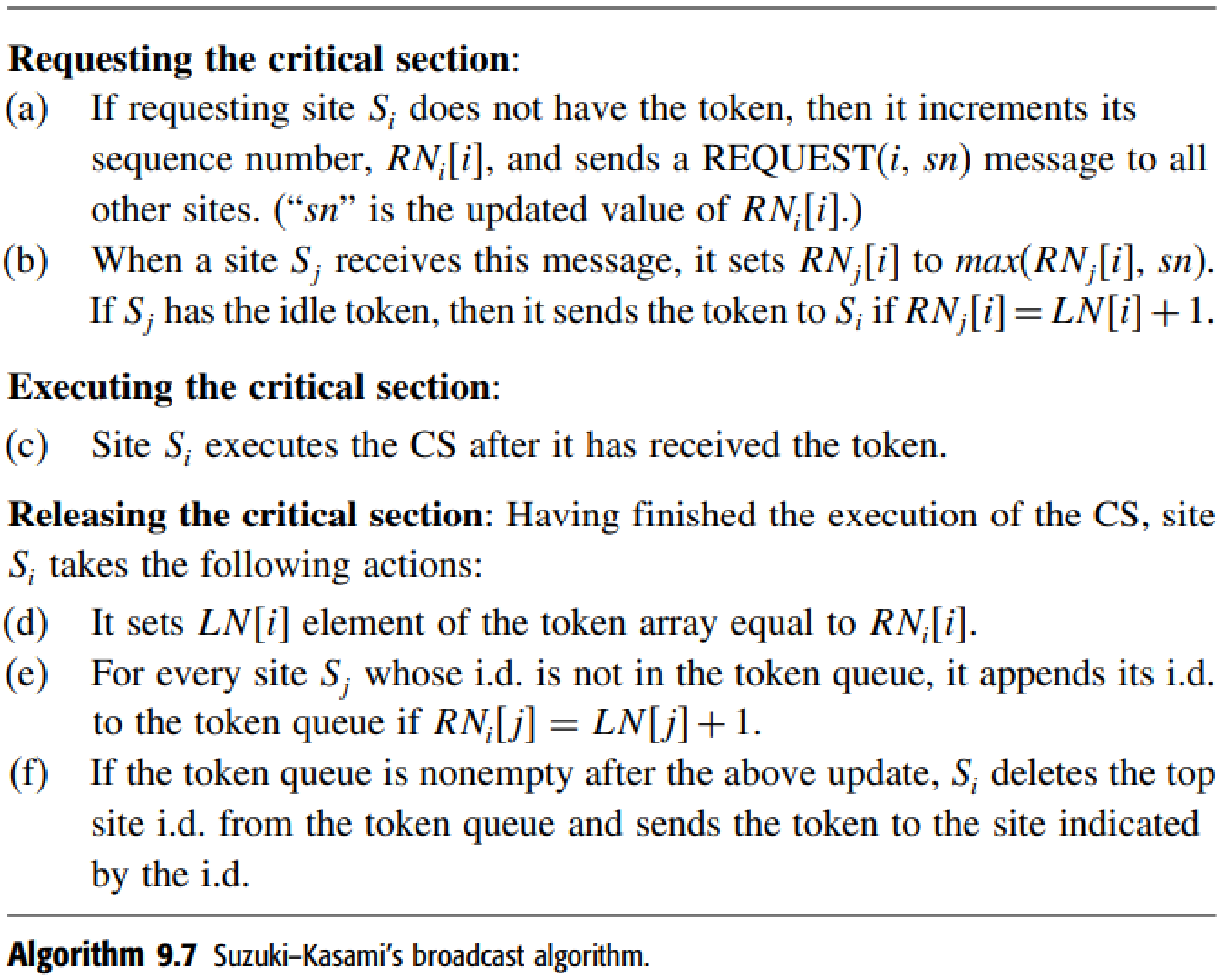
Token-based algorithms

- 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.

Deadlock detection in distributed systems

- Deadlocks are a fundamental problem in distributed systems
- In distributed systems, a process may request resources in any order, which may not be known a priori, and a process can request a resource while holding others.
- If the allocation sequence of process resources is not controlled in such environments, deadlocks can occur.
- A deadlock can be defined as a condition where a set of processes request resources that are held by other processes in the set.

Deadlock detection in distributed systems

- Deadlocks can be dealt -> three strategies: **deadlock prevention, deadlock avoidance, and deadlock detection.**
- **Deadlock prevention** →achieved by either having a process acquire all the needed resources simultaneously before it begins execution or by pre-empting a process that holds the needed resource.
- In the **deadlock avoidance** approach to distributed systems, a resource is granted to a process if the resulting global system is safe.
- **Deadlock detection** requires an examination of the status of the process–resources interaction for the presence of a deadlock condition.
- To resolve the deadlock, we have to abort a deadlocked process.

Deadlock detection in distributed systems

System model

- A distributed system consists of a set of processors that are connected by a communication network.
- The communication delay is finite but unpredictable.
- A distributed program is composed of a set of n asynchronous processes P1, P2, , Pi, , Pn that communicate by message passing over the communication network.
- Each process is running on a different processor.
- The processors do not share a common global memory and communicate solely by passing messages over the communication network.

Deadlock detection in distributed systems

- There is no physical global clock in the system to which processes have instantaneous access.
- The communication medium may deliver messages out of order, messages may be lost, garbled, or duplicated due to timeout and retransmission, processors may fail, and communication links may go down.
- The system can be modeled as a directed graph in which vertices represent the processes and edges represent unidirectional communication channels.

Deadlock detection in distributed systems

Assumptions:

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

- A process can be in two states→ **running** (active state) or **blocked**.

- In the running state→ a process has all the needed resources and is either executing or is ready for execution.

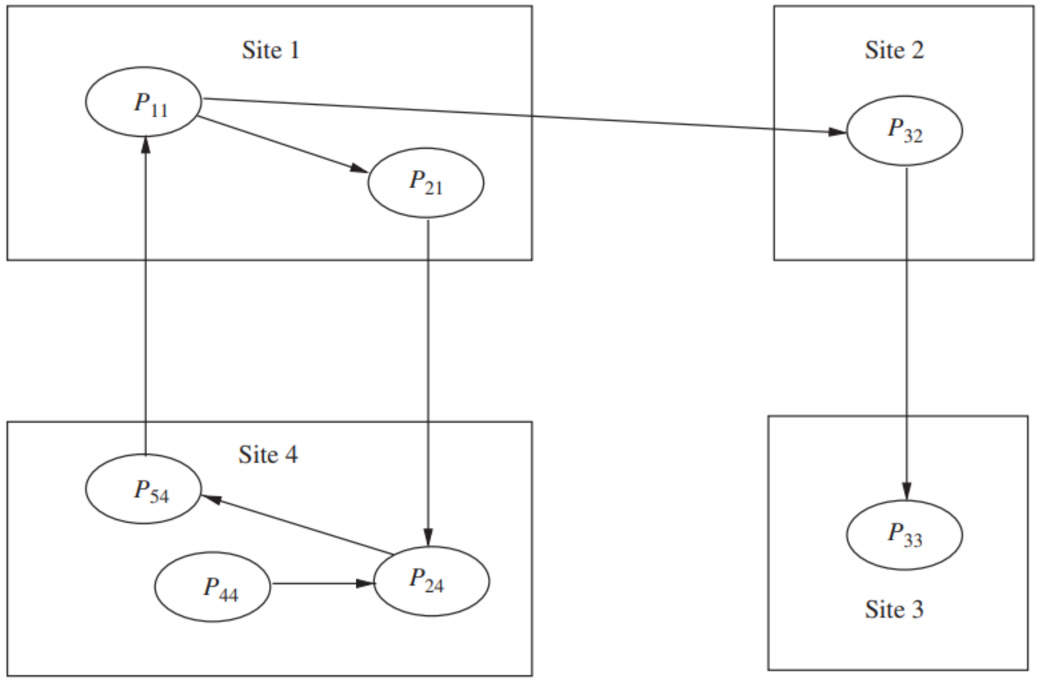
- In the blocked state→ a process is waiting to acquire some resource.

Deadlock detection in distributed systems

Wait-for graph (WFG)

- In distributed systems, the state of the system can be modeled by directed graph, called a wait-for graph (WFG).
- In a WFG, nodes are processes and there is a directed edge from node P1 to node P2 if P1 is blocked and is waiting for P2 to release some resource.
- A system is deadlocked if and only if there exists a directed cycle or knot in the WFG

Figure 10.1 Example of a WFG.



Deadlock detection in distributed systems

Deadlock handling strategies

3 Strategies for handling deadlocks,

- deadlock prevention,
- deadlock avoidance,
- deadlock detection.

Handling of deadlocks→ highly complicated in distributed systems

- no site has accurate knowledge of the current state of the system
- every inter-site communication involves a finite and unpredictable delay

Deadlock prevention →

- a process acquire all the needed resources simultaneously before it begins executing or by pre-empting a process that holds the needed resource

- highly inefficient and impractical in distributed systems

Deadlock detection in distributed systems

In **deadlock avoidance** approach to distributed systems

- a resource is granted to a process if the resulting global system state is safe
- deadlock avoidance is impractical in distributed systems

Deadlock detection

- requires an examination of the status of process– resource interactions for the presence of cyclic wait

- Deadlock detection in distributed systems seems to be the best approach to handle deadlocks in distributed systems.

Issues in Deadlock detection in distributed systems

Issues in deadlock detection :2 basic issues:

- detection of existing deadlocks
- resolution of detected deadlocks.

Detection of deadlocks : 2 Issues

- maintenance of the WFG and
- searching of the WFG for the presence of cycles

- In distributed systems, a cycle or knot may involve several sites, the search for cycles greatly depends upon how the WFG of the system is represented across the system.
- Depending upon the way WFG information is maintained and the search for cycles is carried out →Centralized, Distributed, and Hierarchical algorithms for deadlock detection

Deadlock detection in distributed systems

Correctness criteria

A deadlock detection algorithm must satisfy the following two conditions:

1. **Progress (no undetected deadlocks) :** The algorithm must detect all existing deadlocks in a finite time.

after all wait-for dependencies for a deadlock have formed, the algorithm should not wait for any more events to occur to detect the deadlock

2. Safety (no false deadlocks) : The algorithm should not report deadlocks that do not exist (called phantom or false deadlocks).

In distributed systems where there is no global memory and there is no global clock, it is difficult to design a correct deadlock detection algorithm because sites may obtain an out-of-date and inconsistent WFG of the system.

As a result, sites may detect a cycle that never existed

Deadlock detection in distributed systems

Resolution of a detected deadlock

Deadlock resolution involves breaking existing wait-for dependencies between the processes to resolve the deadlock.

It involves rolling back one or more deadlocked processes and assigning their resources to blocked processes so that they can resume execution

Deadlock detection in distributed systems

Models of deadlocks

- Distributed systems allow many kinds of resource requests.
- A process might require a single resource or a combination of resources for its execution
- Models of deadlocks introduces a hierarchy of request models starting with very restricted forms to the ones with no restrictions

Models of deadlocks

1. **The single-resource model**

- Simplest resource model in a distributed system
- a process can have at most one outstanding request for only one unit of a resource.

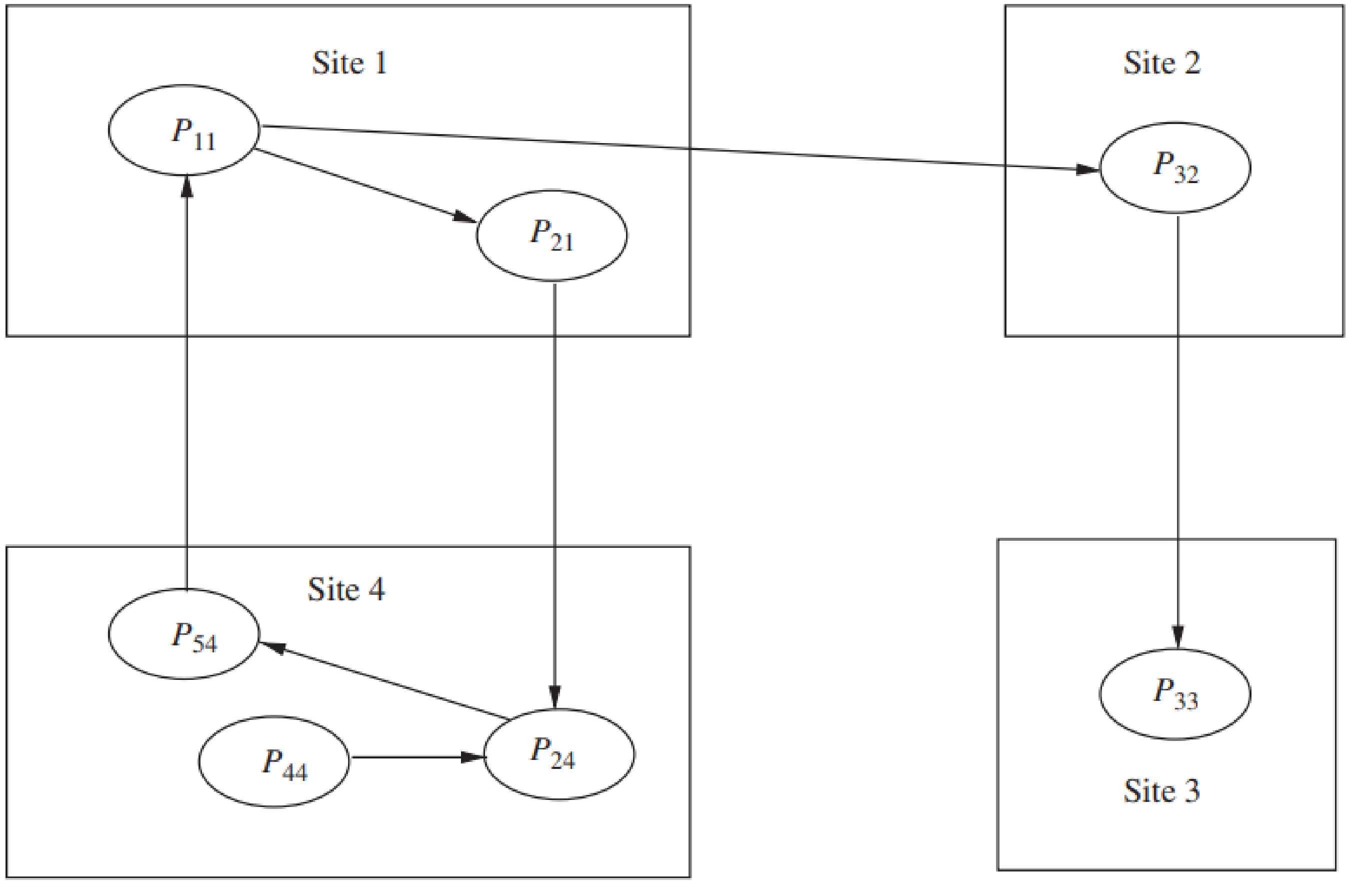
- maximum out-degree of a node in a WFG for the single resource model can be 1
- the presence of a cycle in the WFG shall indicate that there is a deadlock

Models of deadlocks

2. The AND model

- In the AND model, a process can request more than one resource simultaneously and the request is satisfied only after all the requested resources are granted to the process.
- The requested resources may exist at different locations.
- The out degree of a node in the WFG for AND model can be more than 1.
- The presence of a cycle in the WFG indicates a deadlock in the AND model.
- In the AND model, if a cycle is detected in the WFG, it implies a deadlock but not vice versa
- Each node in WFG ->AND node

Figure 10.1 Example of a WFG.



Models of deadlocks

3. The OR model

- In the OR model, a process can make a request for numerous resources simultaneously and the request is satisfied if any one of the requested resources is granted.
- The requested resources may exist at different locations.

- If all requests in the WFG are OR requests, then the nodes are called OR nodes.

- Presence of a cycle in the WFG of an OR model does not imply a deadlock in the OR model.
- In an OR model, a blocked process P is deadlocked if it is either in a knot or it can only reach processes on a knot

The OR model contd…

- In the OR model, the presence of a **knot** indicates a deadlock
- In a WFG, a vertex v is in a knot if for all u :: u is reachable from v : v is reachable from u.
- No paths originating from a knot shall have dead ends.

The OR model contd…

- **A deadlock in the OR model**
- A process Pi is blocked if it has a pending OR request to be satisfied.
- With every blocked process, there is an associated set of processes called dependent set.
- A process shall move from an idle to an active state on receiving a grant message from any of the processes in its dependent set.
- A process is permanently blocked if it never receives a grant message from any of the processes in its dependent set.
- A set of processes S is deadlocked if all the processes in S are permanently blocked.

The OR model contd…

- To formally state that a set of processes is deadlocked, the following conditions hold true:

➢ 1. Each of the process is the set S is blocked.

➢ 2. The dependent set for each process in S is a subset of S.

➢ 3. No grant message is in transit between any two processes in set S.

Models of deadlocks

3. The AND-OR model

A generalization of the previous two models (OR model and AND model) is the AND-OR model.

In the AND-OR model, a request may specify any combination of and and or in the resource request.

For example, in the ANDOR model, a request for multiple resources can be of the form x and (y or z).

Models of deadlocks

4 The $\binom{p}{q}$ model

- Another form of the AND-OR model is the (p q)model (called the P-out-of-Q model), which allows a request to obtain any p available resources from a pool of q resources.

- Both the models are the same in expressive power.

- model lends itself to a much more compact formation of a request

- Every request in the model can be expressed in the AND-OR model and vice-versa

Models of deadlocks

5. Unrestricted model

- In the unrestricted model, no assumptions are made regarding the underlying structure of resource requests.

- Only one assumption that the deadlock is stable is made and hence it is the most general model.

- This model helps separate concerns:
- Concerns about properties of the problem (stability and deadlock) are separated from underlying distributed systems computations (e.g., message passing versus synchronous communication).