

Lecture: 08

Quorum Based Distributed Mutual Exclusion Algorithms



Rajiv Misra

Dept. of Computer Science & Engineering
Indian Institute of Technology Patna
rajivm@iitp.ac.in

Preface

Recap of Previous Lecture:

- In the previous lecture, we have presented the **Concept of Distributed Mutual Exclusion Algorithms** for distributed computing systems.
- We have also discussed about **Non-Token based approaches** *i.e.* **Lamports algorithm** and **Ricart-Agrawala algorithm**

Preface

Content of this Lecture:

- In this lecture, we will discuss about the **Quorum based approaches** and
- Also discuss its few approaches namely **Maekawa's Algorithm** and **Agarwal-El Abbadi Quorum-Based Algorithm**

(ii) Quorum Based Approaches

Introduction

- 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.

Quorum-Based Mutual Exclusion Algorithms

Quorum-based mutual exclusion algorithms are different in the following two ways:

1. 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 $\forall i \forall j : 1 \leq i, j \leq N : R_i \cap R_j \neq \Phi$. Consequently, every pair of sites has a site which mediates conflicts between that pair.
2. 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.

Contd...

Since these algorithms are based on the notion of '**Coteries**' and '**Quorums**', we next 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:** 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.

Contd...

Coterie and **quorum** can be used to develop algorithms to ensure mutual exclusion in a distributed environment. A simple protocol works as follows:

- Let 'a' is 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.

Note that the **Minimality property ensures efficiency** rather than correctness.

(i) Maekawa's Algorithm

- Maekawa's algorithm was the **first quorum-based mutual exclusion algorithm**. The **request sets for sites** (i.e., **quorums**) in Maekawa's algorithm 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: $(\forall i : 1 \leq i \leq N :: |R_i| = K)$

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

Maekawa used the theory of projective planes and showed that $N = K(K - 1) + 1$. This relation gives $|R_i| = \sqrt{N}$

Maekawa's Algorithm

- Conditions **M1** and **M2** are necessary for correctness; whereas conditions **M3** and **M4** provide other desirable features to the algorithm.
- Condition **M3** states that the size of the requests sets of all sites must be equal implying that all sites should have to do equal amount of work to invoke mutual exclusion.
- Condition **M4** enforces that exactly the same number of sites should request permission from any site implying that all sites have **“equal responsibility”** in granting permission to other sites.

The Algorithm

A site S_i executes the following steps to execute the CS.

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 .

The 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.

Correctness

Theorem: *Maekawa's algorithm achieves mutual exclusion.*

Proof:

- Proof is by contradiction. Suppose two sites S_i and S_j are concurrently executing the CS.
- This means site S_i received a **REPLY** message from all sites in R_i and concurrently site S_j was able to receive a **REPLY** message from all sites in R_j .
- If $R_i \cap R_j = \{S_k\}$, then site S_k must have sent **REPLY** messages to both S_i and S_j concurrently, which is a contradiction.

Correctness

- Since the size of a request set is $\sqrt{N}$, an execution of the CS requires $\sqrt{N}$ REQUEST, $\sqrt{N}$ REPLY, and $\sqrt{N}$ RELEASE messages, resulting in $3\sqrt{N}$ messages per CS execution.
- **Synchronization delay** in this algorithm is $2T$. This is because after a site S_i exits the CS, it first releases all the sites in R_i and then one of those sites sends a REPLY message to the next site that executes the CS.

Problem of Deadlocks

- **Maekawa's algorithm can deadlock** because a site is exclusively locked by other sites and requests are not prioritized by their timestamps.
- Assume three sites **S_i , S_j , and S_k simultaneously invoke mutual exclusion.**
- Suppose $R_i \cap R_j = \{S_{ij}\}$, $R_j \cap R_k = \{S_{jk}\}$, and $R_k \cap R_i = \{S_{ki}\}$.

Consider the following scenario:

1. S_{ij} has been locked by S_i (forcing S_j to wait at S_{ij}).
 2. S_{jk} has been locked by S_j (forcing S_k to wait at S_{jk}).
 3. S_{ki} has been locked by S_k (forcing S_i to wait at S_{ki}).
- This state represents a deadlock involving sites S_i , S_j , and S_k .

Handling Deadlocks

- Maekawa's algorithm handles deadlocks by requiring a site to yield a lock if **the timestamp of its request is larger than the timestamp of some other request** waiting for the same lock.
- A site **suspects a deadlock** (and initiates message exchanges to resolve it) whenever a higher priority request arrives and waits at a site because the site has sent a **REPLY message to a lower priority request**.

Handling Deadlocks

Deadlock handling requires **three types of messages**:

FAILED: A FAILED message from site S_i to site S_j indicates that S_i can not grant S_j 's request because it has currently granted permission to a site with a higher priority request.

INQUIRE: An INQUIRE message from S_i to S_j indicates that S_i would like to find out from S_j if it has succeeded in locking all the sites in its request set.

YIELD: A YIELD message from site S_i to S_j indicates that S_i is returning the permission to S_j (to yield to a higher priority request at S_j).

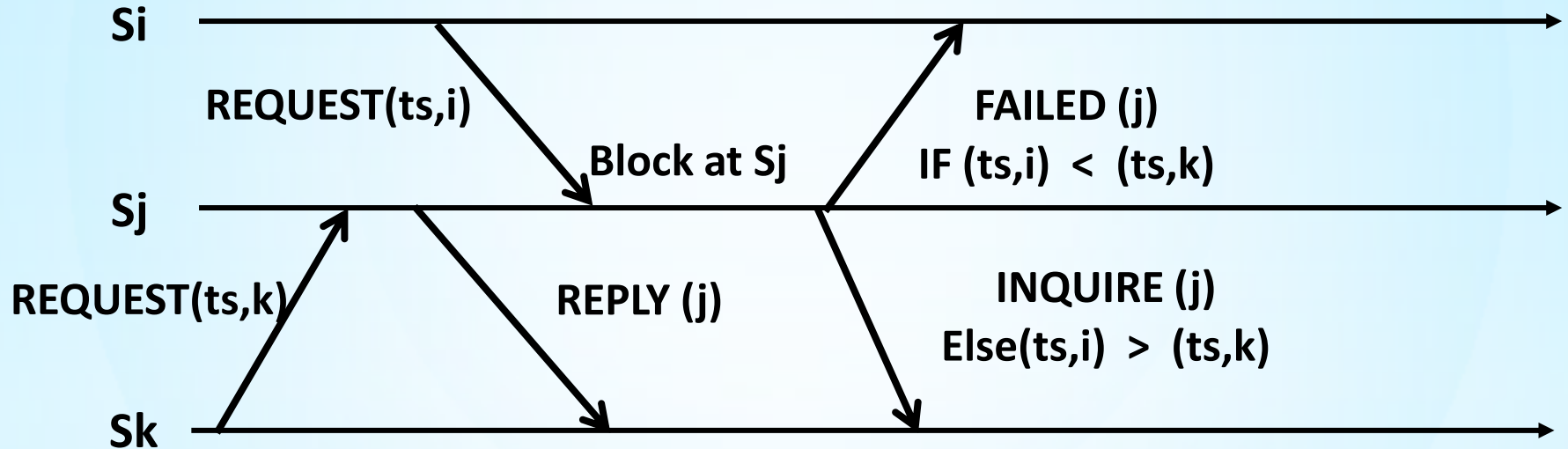
Handling Deadlocks

Maekawa's algorithm handles deadlocks as follows:

- When a **REQUEST**(ts, i) from site S_i blocks at site S_j because S_j has currently granted permission to site S_k , then S_j sends a **FAILED**(j) message to S_i if S_i 's request has lower priority. Otherwise, S_j sends an **INQUIRE**(j) message to site S_k .
- In response to an **INQUIRE**(j) message from site S_j , site S_k sends a **YIELD**(k) message to S_j provided S_k has received a **FAILED** message from a site in its request set or if it sent a **YIELD** to any of these sites, but has not received a new **GRANT** from it.
- In response to a **YIELD**(k) message from site S_k , site S_j assumes as if it has been released by S_k , places the request of S_k at appropriate location in the request queue, and sends a **GRANT**(j) to the top request's site in the queue. Maekawa's algorithm requires extra messages to handle deadlocks
- **Maximum number of messages** required per CS execution in this case is $5\sqrt{N}$

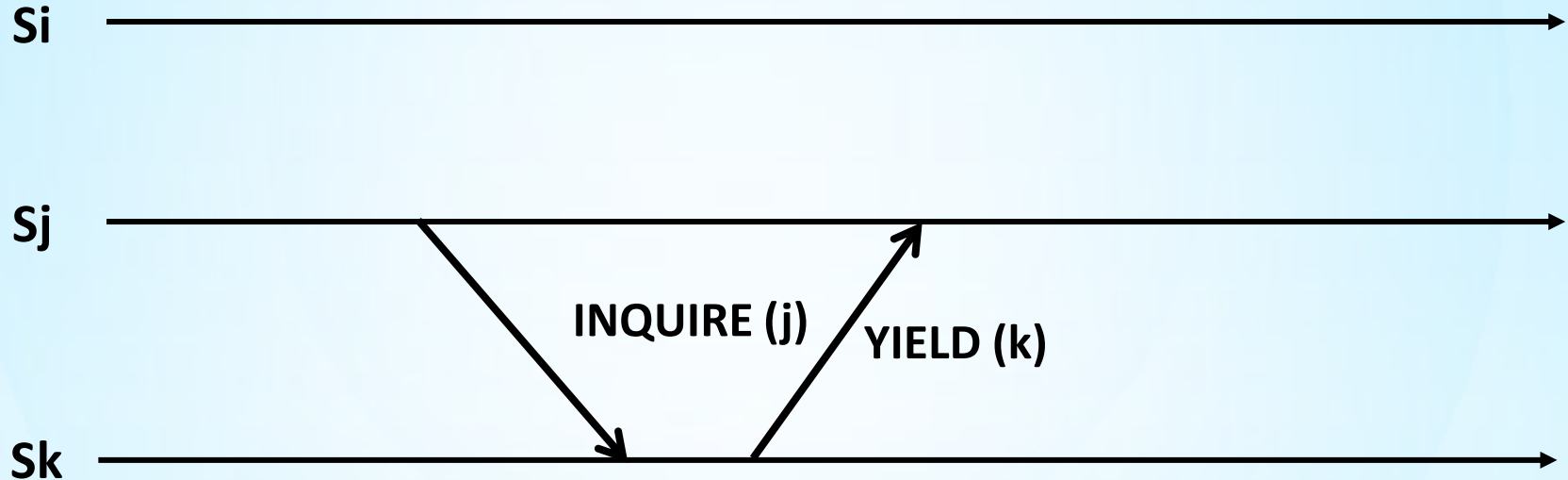
Handling Deadlocks: Case-I

When a **REQUEST**(ts, i) from site S_i blocks at site S_j because S_j has currently granted permission to site S_k , then S_j sends a **FAILED**(j) message to S_i if S_i 's request has lower priority. Otherwise, S_j sends an **INQUIRE**(j) message to site S_k .



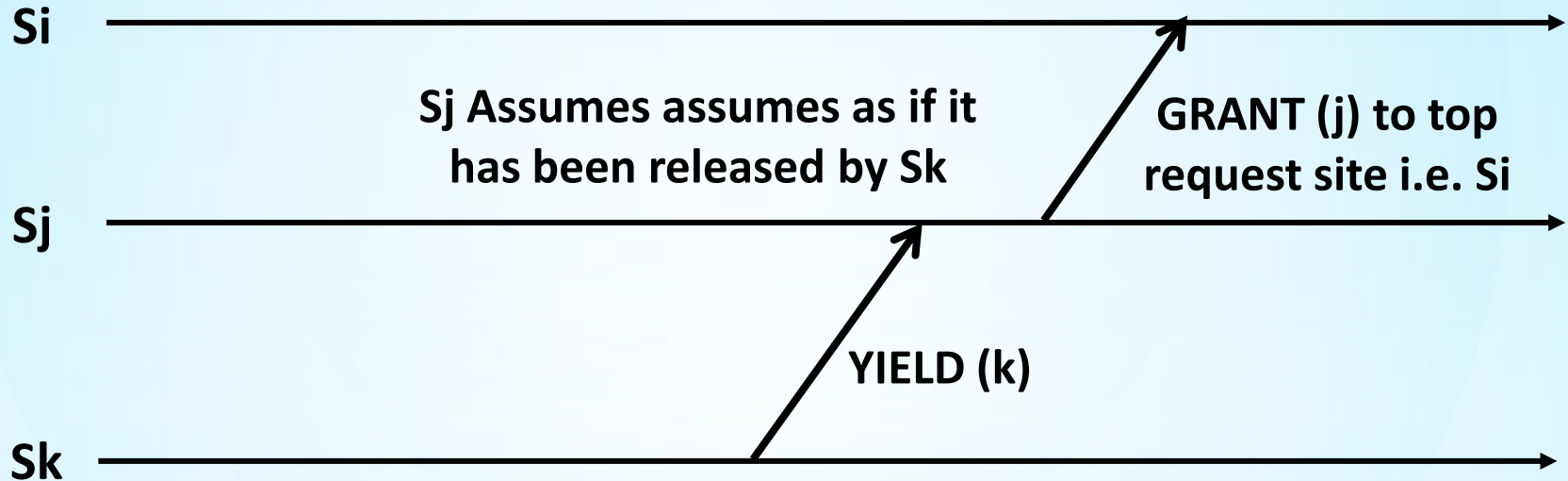
Handling Deadlocks: Case-II

In response to an **INQUIRE(j)** message from site S_j , site S_k sends a **YIELD(k)** message to S_j provided S_k has received a **FAILED** message from a site in its request set or if it sent a **YIELD** to any of these sites, but has not received a new **GRANT** from it.



Handling Deadlocks: Case-III

In response to a **YIELD(k)** message from site S_k , site S_j assumes as if it has been released by S_k , places the request of S_k at appropriate location in the request queue, and sends a **GRANT(j)** to the top request's site in the queue.



(ii) Agarwal-El Abbadi Quorum-Based Algorithm

- **Agarwal and El Abbadi** developed a simple and efficient mutual exclusion algorithm by introducing **tree quorums**.
- They gave a novel algorithm for constructing tree-structured quorums in the sense that it uses **hierarchical structure of a network**.
- The mutual exclusion algorithm is independent of the underlying topology of the network and there is **no need for a multicast facility** in the network.
- However, such facility will improve the performance of the algorithm.
- The mutual exclusion algorithm **assumes** that **sites in the distributed system can be organized into a structure such as tree, grid, binary tree, etc. and**
- There exists a **routing mechanism to exchange messages** between different sites in the system.

(ii) Agarwal-El Abbadi Quorum-Based Algorithm

- Agarwal-El Abbadi quorum-based algorithm uses '**tree-structured quorums**'
- All the sites in the system are logically organized into a **complete binary tree**.
- For a complete binary tree with **level 'k'**, we have **$2^{k+1} - 1$ sites** with its root at level k and leaves at level 0.
- The number of sites in a path from the root to a leaf is equal to the level of the tree k+1 which is equal to **$O(\log n)$** .
- A **path in a binary tree is the sequence** $a_1, a_2, \dots, a_i, a_{i+1}, \dots, a_k$ such that a_i is the parent of a_{i+1} .

Algorithm for constructing a tree-structured quorum

```
FUNCTION GetQuorum (Tree: NetworkHierarchy): QuorumSet;  
  VAR left, right : QuorumSet;  
  BEGIN  
    IF Empty (Tree) THEN  
      RETURN ({});  
    ELSE IF GrantsPermission(Tree↑.Node) THEN  
      RETURN ((Tree↑.Node) ∪ GetQuorum (Tree↑.LeftChild));  
    OR  
      RETURN ((Tree↑.Node) ∪ GetQuorum (Tree↑.RightChild));(*line 9*)  
    ELSE  
      left←GetQuorum(Tree↑.left);  
      right←GetQuorum(Tree↑.right);  
      IF (left = ∅ ∨ right = ∅) THEN  
        (* Unsuccessful in establishing a quorum *)  
        EXIT(-1);  
      ELSE  
        RETURN (left ∪ right);  
      END; (* IF *)  
    END; (* IF *)  
  END; (* IF *)  
END GetQuorum
```

Algorithm for constructing a tree-structured quorum: Explanation

- The algorithm for constructing tree-structured quorums uses two functions called **GetQuorum(Tree)** and **GrantsPermission(site)** and assumes that there is a well-defined root for the tree.
- **GetQuorum** is a recursive function that takes a **tree node “x”** as the parameter and calls **GetQuorum** for its child node provided that the **GrantsPermission(x)** is true.
- The **GrantsPermission(x)** is **true** only when the **node “x”** agrees to be in the quorum. If the **node “x”** is down due to a failure, then it may not agree to be in the quorum and the value of **GrantsPermission(x)** will be **false**.

Algorithm for constructing a tree-structured quorum:

Explanation

- The algorithm tries to construct quorums in a way that each quorum represents **any path from the root to a leaf**. *i.e.*, in this case the (no failures) quorum is any set $a_1, a_2, \dots, a_i, a_{i+1}, \dots, a_k$, where a_1 is the root and a_k is a leaf, and for all $i < k$, a_i is the parent of a_{i+1} .
- If it fails to find such a path (say, because node 'x' has failed), the control goes to the **ELSE** block which specifies that the failed node 'x' is substituted by two paths both of which start with the left and right children of 'x' and end at leaf nodes.
- If the leaf site is down or inaccessible due to any reason, then the quorum cannot be formed and the algorithm terminates with an error condition.
- The sets that are constructed using this algorithm are termed as **tree quorums**.

Analysis of the algorithm for constructing tree-structured quorums

- The **best case scenario** of the algorithm takes $O(\log n)$ sites to form a tree quorum. There are certain cases where even in the event of a failure, $O(\log n)$ sites are sufficient to form a tree quorum.
- For example, if the site that is parent of a leaf node fails, then the **number of sites that are necessary for a quorum will be still $O(\log n)$** . Thus, the algorithm requires very **few messages** in a relatively **fault-free environment**.
- It can tolerate the failure up to $n - O(\log n)$ sites and still form a tree quorum.
- **In the worst case**, the algorithm requires the majority of sites to construct a tree quorum and the number of sites is same for all cases (faults or no faults). The **worst case tree quorum size** is determined as $O((n+1)/2)$ by **induction**.

Examples of Tree-Structured Quorums

When there is no node failure, the number of quorums formed is equal to the number of leaf sites.

- Consider the tree of height 3 shown in Figure 8.1, constructed from 15 ($=2^{3+1}-1$) sites.
- In this case 8 quorums are formed from 8 possible root-leaf paths: 1-2-4-8, 1-2-4-9, 1-2-5-10, 1-2-5-11, 1-3-6-12, 1-3-6-13, 1-3-7-14 and 1-3-7-15.
- If any site fails, the algorithm substitutes for that site two possible paths starting from the site's two children and ending in leaf nodes.
- For example, when node 3 fails, we consider possible paths starting from children 6 and 7 and ending at leaf nodes. The possible paths starting from child 6 are 6-12 and 6-13, and from child 7 are 7-14 and 7-15.
- So, when node 3 fails, the following eight quorums can be formed:
 $\{1,6,12,7,14\}$, $\{1,6,12,7,15\}$, $\{1,6,13,7,14\}$, $\{1,6,13,7,15\}$, $\{1,2,4,8\}$,
 $\{1,2,4,9\}$, $\{1,2,5,10\}$, $\{1,2,5,11\}$.

8 quorums are

formed :

1-2-4-8,
1-2-4-9,
1-2-5-10,
1-2-5-11,
1-3-6-12,
1-3-6-13,
1-3-7-14 and
1-3-7-15

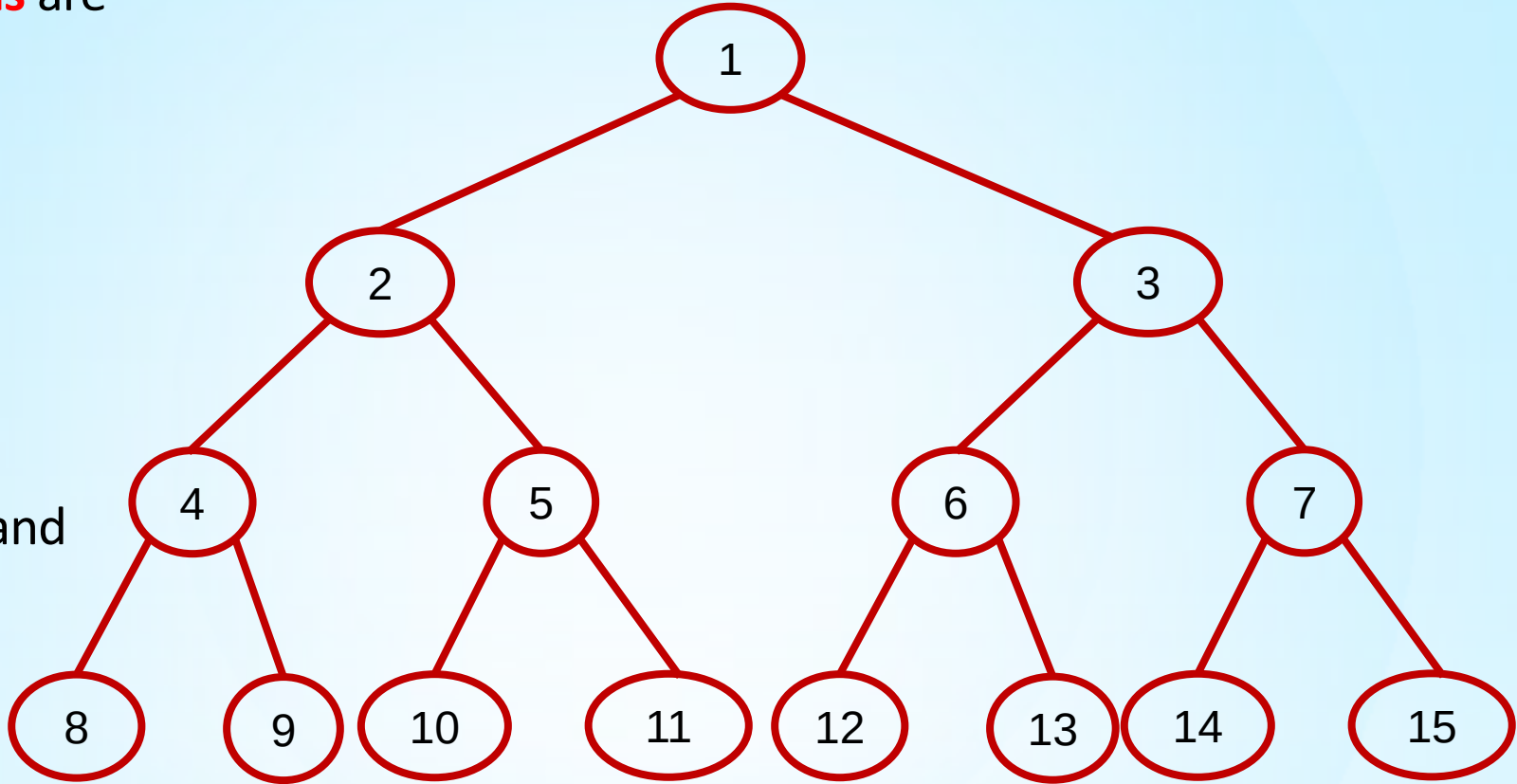


Figure 8.1: A tree of 15 sites.

So, **when node 3 fails**, quorums can be formed:

$\{1,6,12,7,14\},$
 $\{1,6,12,7,15\},$
 $\{1,6,13,7,14\},$
 $\{1,6,13,7,15\},$
 $\{1,2,4,8\},$
 $\{1,2,4,9\},$
 $\{1,2,5,10\},$
 $\{1,2,5,11\}.$

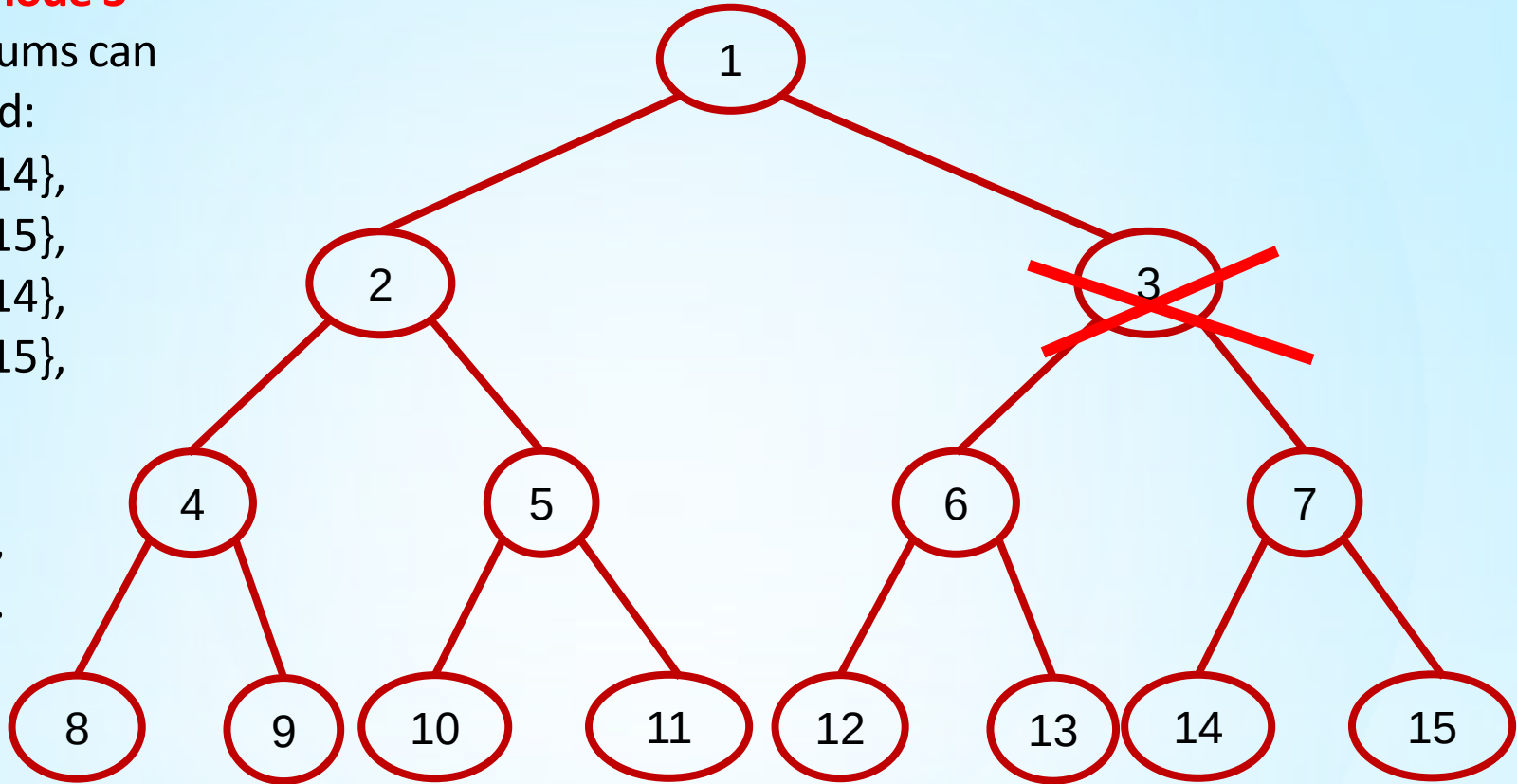


Figure 8.1: A tree of 15 sites.

Property of 'Graceful degradation'

- Since the number of nodes from root to leaf in an '**n**' node complete tree is **$\log n$** , the best case for quorum formation, i.e, **the least number of nodes needed for a quorum is $\log n$** .
- When the number of node failures is greater than or equal to **$\log n$** , the algorithm may not be able to form tree-structured quorum.
- So, as long as the number of site failures is less than **$\log n$** , the tree quorum algorithm guarantees the formation of a quorum and it exhibits the property of '**graceful degradation**'.

Mutual Exclusion Algorithm

A site s enters the critical section (CS) as follows:

- Site s sends a '**Request**' message to all other sites in the structured quorum it belongs to.
- Each site in the quorum stores incoming requests in a *request queue*, ordered by their timestamps.
- A site sends a '**Reply**' message, indicating its consent to enter CS, only to the request at the head of its *request queue*, having the lowest timestamp.
- If the site s gets a '**Reply**' message from all sites in the structured quorum it belongs to, it enters the CS.
- After exiting the CS, s sends a '**Relinquish**' message to all sites in the structured quorum. On the receipt of the '**Relinquish**' message, each site removes s 's request from the head of its *request queue*.

Mutual Exclusion Algorithm

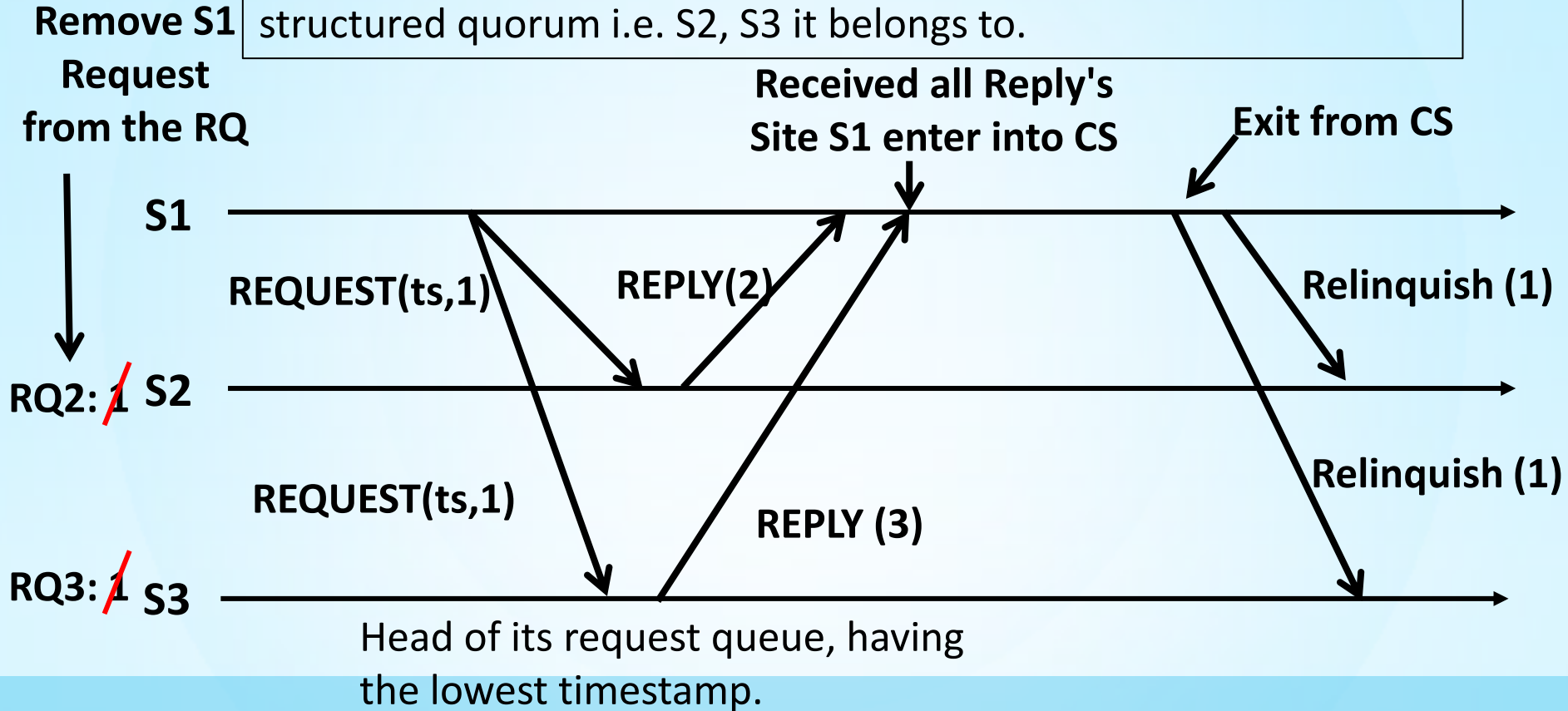
- If a new request arrives with a timestamp smaller than the request at the head of the queue, an **'Inquire'** message is sent to the process whose request is at the head of the queue and waits for a **'Yield'** or **'Relinquish'** message.
- When a site s receives an **'Inquire'** message, it acts as follows: If s has acquired all of its necessary replies to access the CS, then it simply ignores the **'Inquire'** message and proceeds normally and sends a **'Relinquish'** message after exiting the CS.

If s has not yet collected enough replies from its quorum, then it sends a **'Yield'** message to the inquiring site.

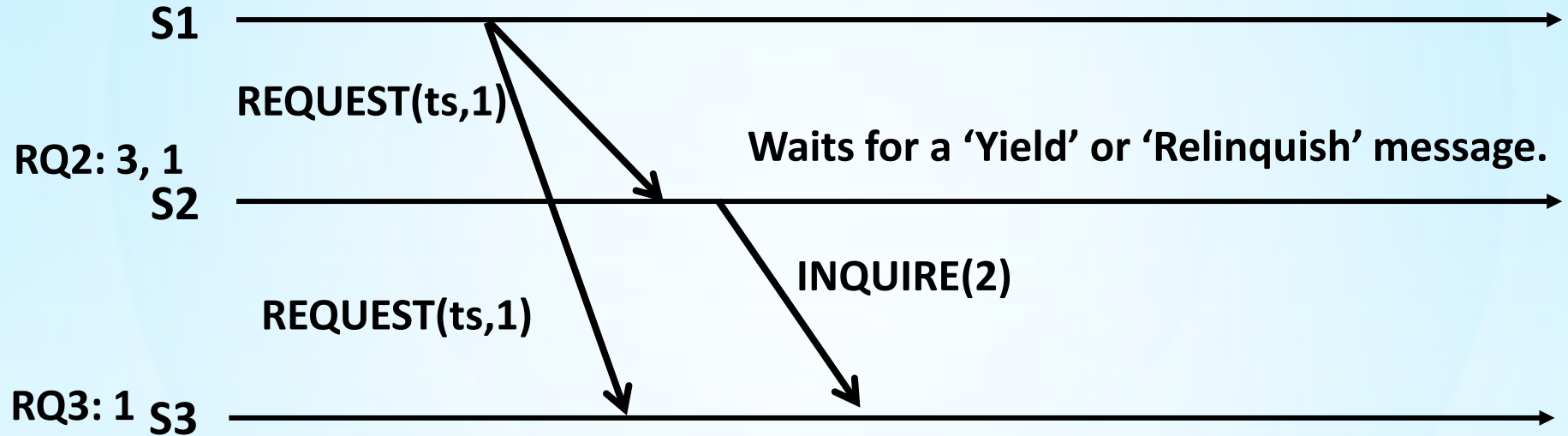
- When a site gets the **'Yield'** message, it puts the pending request (on behalf of which the **'Inquire'** message was sent) at the head of the queue and sends a **'Reply'** message to the requestor.

Example:

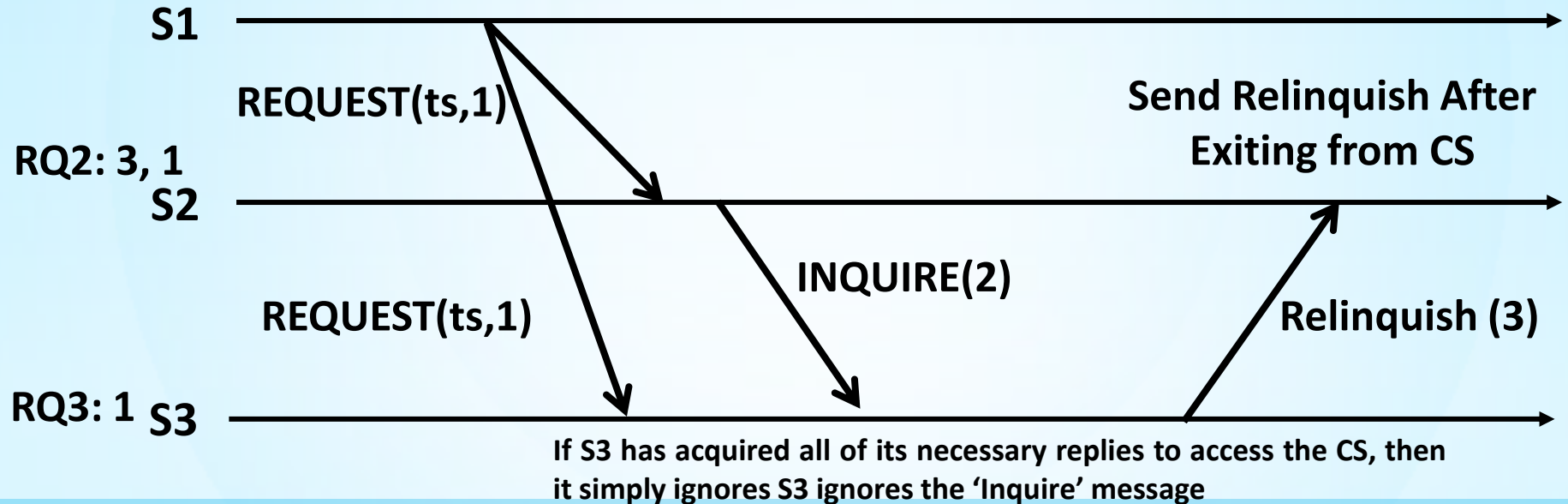
Site S1 sends a '**Request**' message to all other sites in the structured quorum i.e. S2, S3 it belongs to.



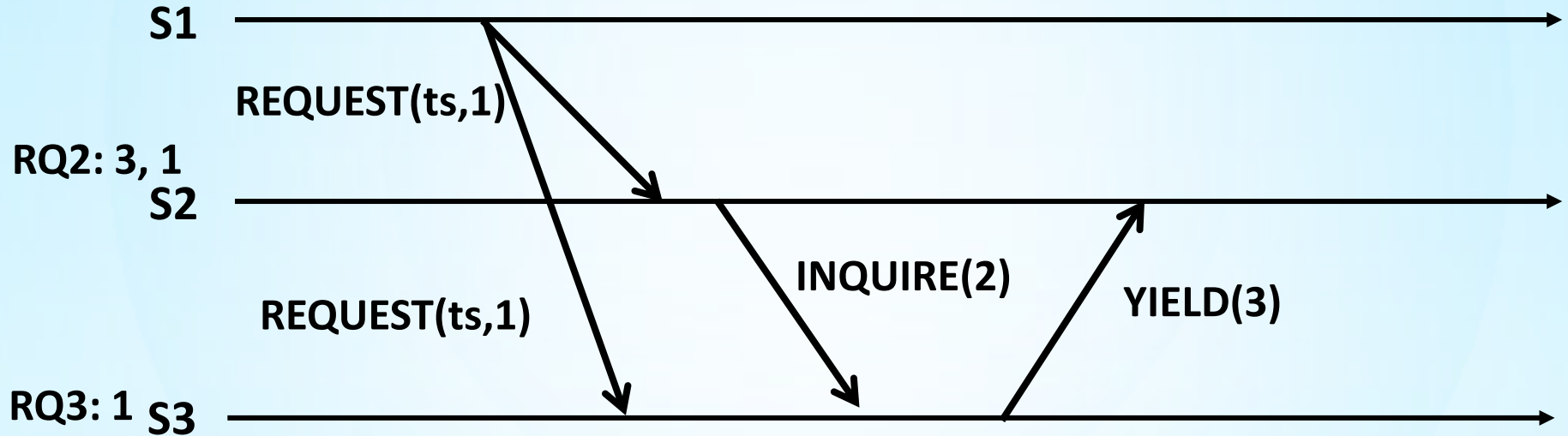
If a new request arrives with a timestamp smaller than the request at the head of the queue, an **'Inquire'** message is sent to the process whose request is at the head of the queue and waits for a **'Yield'** or **'Relinquish'** message.



When a site S3 receives an '**Inquire**' message, it acts as follows: If S3 has acquired all of its necessary replies to access the CS, then it simply ignores the '**Inquire**' message and proceeds normally and sends a '**Relinquish**' message after exiting the CS.

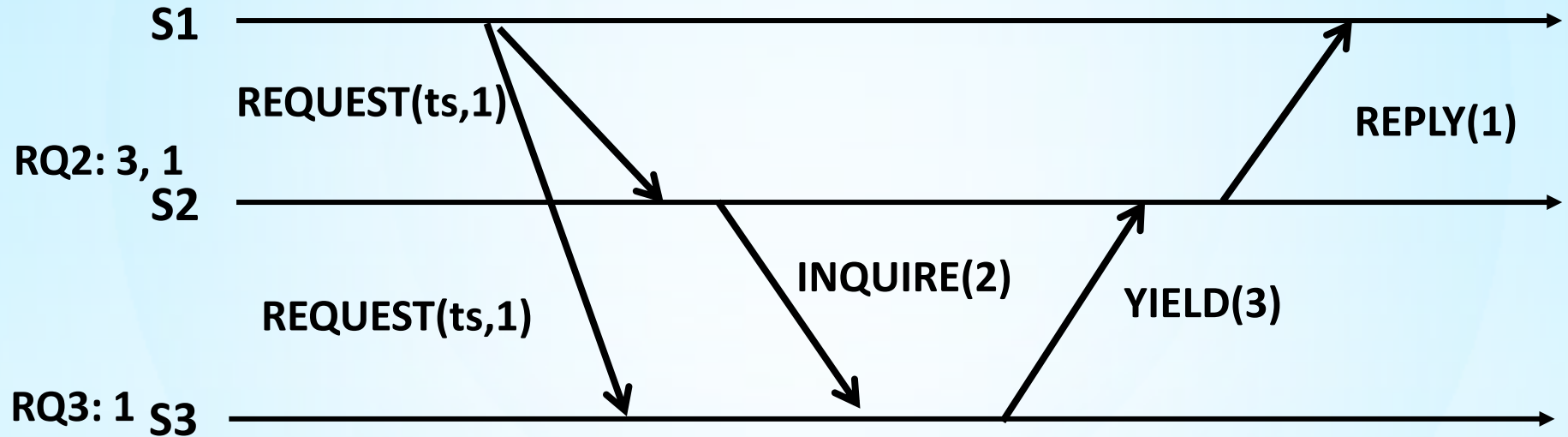


If S3 has not yet collected enough replies from its quorum, then it sends a **'Yield'** message to the inquiring site.



If S3 has not yet collected
enough replies from its quorum

When a site gets the '**Yield**' message, it puts the pending request (on behalf of which the '**Inquire**' message was sent) at the head of the queue and sends a '**Reply**' message to the requestor.



If S3 has not yet collected enough replies from its quorum

Correctness proof

- **Mutual exclusion** is **guaranteed** because the set of quorums **satisfy the Intersection property**.
- Consider a coterie C which consists of quorums $\{1,2,3\}$, $\{2,4,5\}$ and $\{4,1,6\}$.
- Suppose nodes 3, 5 and 6 want to enter CS, and they send requests to sites (1, 2), (2, 4) and (1, 4), respectively.
- Suppose site 3's request arrives at site 2 before site 5's request. In this case, site 2 will grant permission to site 3's request and reject site 5's request.
- Similarly, suppose site 3's request arrives at site 1 before site 6's request. So site 1 will grant permission to site 3's request and reject site 6's request.
- Since sites 5 and 6 did not get consent from all sites in their quorums, they do not enter the CS.
- Since site 3 alone gets consent from all sites in its quorum, it enters the CS and mutual exclusion is achieved.

Conclusion

- In this lecture, we have discussed about quorum based approaches. i.e. **Maekawa's Algorithm and Agarwal-El Abbadi Quorum-Based Algorithm**
- There exists a variety of quorums and a variety of ways to construct quorums. For example, **Maekawa used the theory of projective planes** to develop quorums of size $\sqrt{N}$ and Agarwal-El Abbadi quorum-based algorithm uses '**tree-structured quorums**.'
- In upcoming lecture, we will discuss about '**Token based approaches**' i.e. Suzuki-Kasami's Broadcast Algorithm and Raymond's Tree-Based Algorithm .