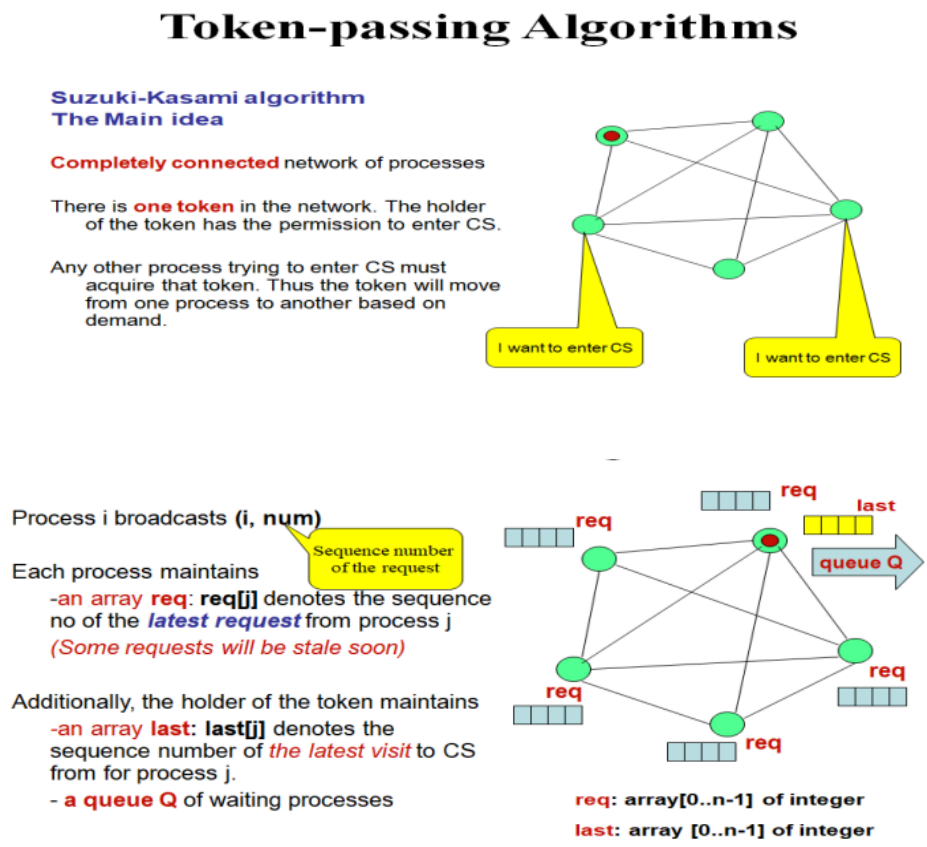
- **Token Based Algorithms: Suzuki-Kasami's Broadcast Algorithms**

Suzuki–Kasami algorithm is a token-based algorithm for achieving mutual exclusion in distributed systems. In token-based algorithms, A site is allowed to enter its critical section if it possesses the unique token.

Non-token-based algorithms use timestamps to order requests for the critical section whereas sequence number is used in token based algorithms. Each request for the critical section contains a sequence number. This sequence number is used to distinguish old and current requests.

Example:



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 which 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.

The basic idea underlying this algorithm may sound rather simple, however, there are the following two design issues must be efficiently addressed:

1. How to distinguishing an outdated REQUEST message from a current REQUEST message:

Due to variable message delays, a site may receive a token request message after the corresponding request has been satisfied. If a site can not determine if the request corresponding to a token request has been satisfied, it may dispatch the token to a site that does not need it. This will not violate the correctness; however, this may seriously degrade the performance by wasting messages and increasing the delay at sites that are genuinely requesting the token. Therefore, appropriate mechanisms should be implemented to determine if a token request message is outdated.

2. How to determine which site has an outstanding request for the CS:

After a site has finished the execution of the CS, it must determine what sites have an outstanding request for the CS so that the token can be dispatched to one of them. The problem is complicated because when a site Si receives a token request message from a site Sj, site Sj may have an outstanding request for the CS. However, after the corresponding request for the CS has been satisfied at Sj, an issue is how to inform site Si (and all other sites) efficiently about it. Outdated REQUEST messages are distinguished from current REQUEST messages in the following manner: A REQUEST message of site Sjhas the form REQUEST(j, n) where n (n=1, 2,...) is a sequence number which indicates that site Sjis requesting its nth CS execution.

A site Si keeps an array of integers RNi[1..N] where RNi[j] denotes the largest sequence number received in a REQUEST message so far from site Sj. When site Si receives a REQUEST (j, n)message, it sets RNi[j]:=max(RNi[j], n). Thus, when a site Si receives a REQUEST (j, n) message, the request is outdated if RNi[j]>n. Sites with outstanding requests for the CS are determined in the following manner: The token consists of a queue of requesting sites, Q, and an array of integers LN[1..N], where LN[j] is the sequence number of the request which site Sj executed most recently. After executing its CS, a site Si updates LN[i]:=RNi[i] to indicate that its request corresponding to sequence number RNi[i] has been executed. Token array LN[1..N] permits a site to determine if a

site has an outstanding request for the CS. Note that at site Si if RNi[j]=LN[j]+1, then site Sj is currently requesting a token. After executing the CS, a site checks this condition for all the j' s to determine all the sites which are requesting the token and placestheir id's in queue Q if these id's are not already present in the Q. Finally, the site sends the token to the site whose id is at the head of the Q.

ALGORITHM

1. Requesting the critical section

- If site Si does not have the token, then it increments its sequence number, RNi[i], and sends a REQUEST(i, sn) message to all other sites. ('sn' is the updated value of RNi[i].)
- When a site Sj receives this message, it sets RNj[i] to max(RNj[i], sn). If Sj has the idle token, then it sends the token to Si if RNj[i]=LN[i]+1.

2. Executing the critical section

- Site Si executes the CS after it has received the token.

3. Releasing the critical section

Having finished the execution of the CS, site Si takes the following actions:

- It sets the LN[i] element of the token array equal to RNi[i].
- For every site Sj whose id is not in the token queue, it appends its id to the token queue if RNi[j]=LN[j]+1.
- If the token queue is nonempty after the above update, Si deletes the top site id from the token queue and sends the token to the site indicated by the id. Thus, after executing the CS, a site gives priority to other sites with outstanding requests for the CS (over its pending requests for the CS). Note that Suzuki-Kasami's algorithm is not symmetric because a site retains the token even if it does not have a request for the CS, which is contrary to the spirit of Ricart and Agrawala's definition of symmetric algorithm: "no site possesses the right to access its CS when it has not been requested".

Correctness

Mutual exclusion is guaranteed because there is only one token in the system and a site holds the token during the CS execution.

Message Complexity:

The algorithm requires 0 message invocation if the site already holds the idle token at the time of critical section request or maximum of N message per critical section execution. This N messages involves

- (N – 1) request messages
- 1 reply message

Performance:

Synchronization delay is 0 and no message is needed if the site holds the idle token at the time of its request.

In case the site does not hold the idle token, the maximum synchronization delay is equal to maximum message transmission time and a maximum of N message is required per critical section invocation.

Drawbacks of Suzuki–Kasami Algorithm:

Non-symmetric Algorithm: A site retains the token even if it does not have requested a critical section. According to definition of symmetric algorithm

"No site possesses the right to access its critical section when it has not been requested."

- **Singhal's Heuristic Algorithm**

Instead of broadcast: each site maintains information on other sites, guess the sites likely to have the token.

Data Structure:

- Si maintains SVi[1……N] and SNi[1……..N] for storing information on other sites: state and highest sequence number.
- Token contains 2 arrays: TSV[1…N] and TSN[1……N].
- States of a site:
  - R: requesting CS
  - E: executing CS
  - H: holding token, idle
  - N: none of the above
- Initialization
  - SVi[j]:=N, for j=N….i; SVi[j]:=R, for j=i-1…..1; SNi[j]:=0,j=1….N. S1(Site 1) is in state H.
  - Token: TSV[j]:=N & TSN[j]:=0, j =1….N

Requesting CS

- If Si has no token and requests CS:
  - SVi[i]:=R. SNi[i]:=SNi[i]+1
  - Send REQUEST (i,sn) to sites Sj for which SVi[j] = R.
- Receiving REQUEST (i,sn): if sn<=SNj[i], ignore.

  Otherwise, update SNj[i] and do:

  - SVj[i]=N->SVj[i]:=R
  - SVj[j]=R-> if SVj[i]!=R, set it to R and send REQUEST (j,SNj[j]) to Si. Else do nothing
  - SVj[j]=E->SVj[i]:=R
  - SVj[i]=H->SVj[i]:=R, TSV[i]:=R, TSN[i]:= sn, SVj[j]=N. Send a token to Si.

Executing CS: after getting a token. Set SVi[i]:=E

Releasing CS

- SVi[i]:=N, TSV[i]:=N. Then, do:
  - For other Sj:if(SNi[j]>TSN[j]), then {TSV[j]:=SVi[j]; TSN[j]:=SNi[j]}
  - Else {SVi[j]:=TSV[j]; SNi[j]:=TSN[j]}
- If SVi[j]=N, for all j, then set SVi[i]:=H. Else send token to a site Sj provided SVi[j]=R

Fairness of algorithm will depend on choice of Si, since no queue is maintained in token.

Arbitration rules to ensure fairness used.

Performance:

- Low to moderate loads: average of N/2 messages.
- High loads: N messages (all site request CS)
- Synchronization delay: T

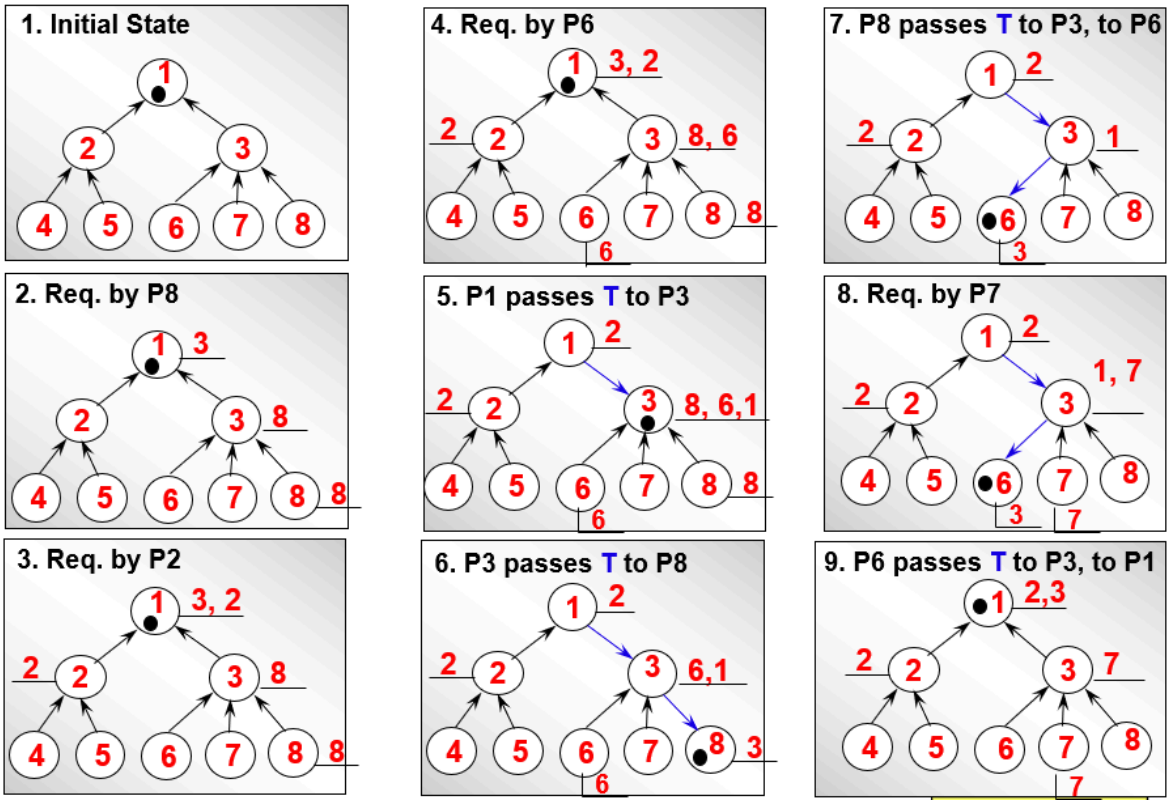
Raymond's Tree based Algorithm

Processes are organized as an un-rooted n-ary tree. Each process has a variable HOLDER, which indicates the location of the token relative to the node itself. Each process keeps a REQUEST_Q that holds the names of neighbors or itself that have sent a REQUEST, but have not yet been sent the token in reply.

- To enter the CS
  - Enqueue self.
  - If a request has not been sent to HOLDER, send a request.
- Upon receipt of a REQUEST message from neighbor x
  - If x is not in the queue, enqueue x.
  - If self is a HOLDER and still in the CS, do nothing further.
  - If self is a HOLDER but exits the CS, then get the oldest requester (i.e., dequeue REQUEST_Q), set it to be the new HOLDER, and send a token to the new HOLDER.
- Upon receipt of a token message
  - Dequeue REQUEST_Q and set the oldest requester to be HOLDER.
  - If HOLDER=self, then hold the token and enter the CS.
  - If HOLDER= some other process, send a token to HOLDER. In addition, if the (remaining) REQUEST_Q is non-empty, send REQUEST to HOLDER as well.
- On exiting the CS

- If REQUEST_Q is empty, continue to hold the token.
- If REQUEST_Q is non-empty, then
  - dequeue REQUEST_Q and set the oldest requester to HOLDER, and send a token to HOLDER.
  - In addition, if the (remaining) REQUEST_Q is non-empty, send REQUEST to HOLDER as well.

Example:



Drawbacks of Raymond's tree based algorithm:

It can cause starvation: Raymond's algorithm uses greedy strategy as a site can executes the critical section on receiving the token even when its request is not on the top of the request queue. This affect the fairness of the algorithm and thus can cause in starvation.

Performance:

- Synchronization delay is (T * log N)/ 2, because the average distance between two sites to successively execute the critical section is (Log N)/2. Here T is maximum message transmission time.
- In heavy load conditions, the synchronization delay becomes T because a site executes the critical section every time the token is transferred.
- The message complexity of this algorithm is O(log N) as the average distance between any two nodes in a tree with N nodes is log N
- Deadlock is impossible

Message Complexity:

In the worst case, the algorithm requires 2 * (Longest path length of the tree) message invocation per critical section entry. If all nodes are arranged in a straight line then the longest path length will be N – 1 and thus the algorithm will require 2 * (N -1) message invocation for critical section entry. However, if all nodes generates equal number of REQUEST messages for the privilege, the algorithm will require approximately 2*N / 3 messages per critical section entry.

• Comparative Performance Analysis

Token Based Algorithms	Non-Token Based Algorithms
In the Token-based algorithm, a unique token is shared among all the sites in Distributed Computing Systems.	In Non-Token based algorithms, there is no token, not even any concept of sharing a token for access.
Here, a site is allowed to enter the Critical Section if it possesses the token.	Here, two or more successive rounds of messages are exchanged between sites to determine which site is to enter the Critical Section next.
The token-based algorithm uses the sequences to order the request for the Critical Section and to resolve the conflict for the simultaneous requests for the	Non-Token based algorithm uses the timestamp (another concept) to order the request for the Critical Section and to resolve the conflict for the simultaneous

System.	requests for the System.
The token-based algorithm produces less message traffic as compared to Non-Token based Algorithm.	Non-Token based Algorithm produces more message traffic as compared to the Token-based Algorithm.
They are free from deadlock (i.e. here there are no two or more processes in the queue in order to wait for messages that will actually can't come) because of the existence of unique tokens in the distributed system.	They are not free from the deadlock problem as they are based on timestamps only.
Here, it is ensured that requests are executed exactly in the order as they are made in.	Here there is no surety of execution order.
Token-based algorithms are more scalable as they can free your server from storing session state and also they contain all the necessary information which they need for authentication.	Non-Token based algorithms are less scalable than the Token-based algorithms because the server is not free from its tasks.
Here the access control is quite Fine-grained because here inside the token roles, permissions and resources can be easily specified for the user.	Here the access control is not so fine as there is no token which can specify roles, permission, and resources for the user.
Token-based algorithms make authentication quite easy.	Non-Token based algorithms can't make authentication easy.
Examples of Token-Based Algorithms are: (i) Singhal's Heuristic Algorithm (ii) Raymonds Tree Based Algorithm (iii) Suzuki-Kasami Algorithm	Examples of Non-Token Based Algorithms are: (i) Lamport's Algorithm (ii) Ricart-Agarwala Algorithm (iii) Maekawa's Algorithm