# Operating System Project

# Raymond's DME algorithm for Synchronization

Presented by:-

- Akshay Singh (16114011)
- Priyesh Kumar (16114051)
- Tarun Kumar (16114066)
- Ankur Parihar (16114015)

# Mutual Exclusion

- It is concurrent access of processes to a shared resource or data in a mutually exclusive manner so as to synchronize them.
- Only one process is allowed to execute the critical section (CS) at any given time.
- Message passing is the sole means for implementing distributed mutual  exclusion.

# Introduction

- We must deal with unpredictable message delays in distributed systems.
- There are two basic approaches for distributed mutual exclusion:
  - Token based approach
  - Non-token based approach (Permission based)
- Token-based approach:
  - A unique token is shared among the sites to ensure mutual exclusion.
  - A site is allowed to enter its CS only if it has the token.

  E.g. Suzuki-Kasami's algorithm, Raymond's Algorithm
- Non-token based approach:
  - A node needs to get the permission of all other nodes to execute its CS.

  E.g. Lamport's algorithm, Maekawa's algorithm

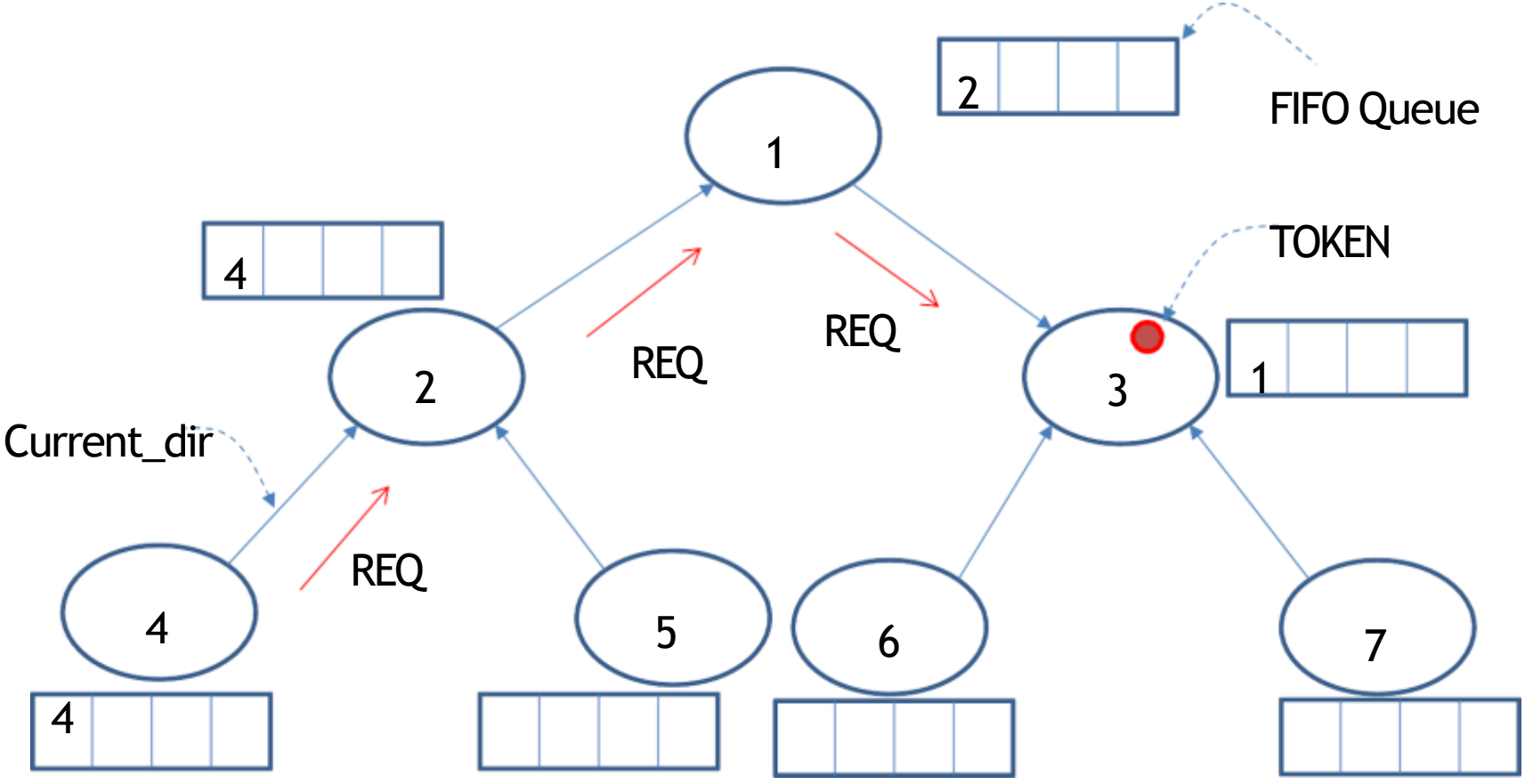
# System Model

- The system consists of N sites, $S_1$, $S_2$, ..., $S_N$.
- We assume that a single process is running on each site.
- A site can be in one of the following three states: requesting the token, executing the CS, or neither requesting nor executing i.e. idle.
- In the ‘requesting the token’ state, the site is blocked and can not make further requests for token.
- At any instant, a site may have several pending requests for CS.

# Raymond's Tree-Based Algorithm

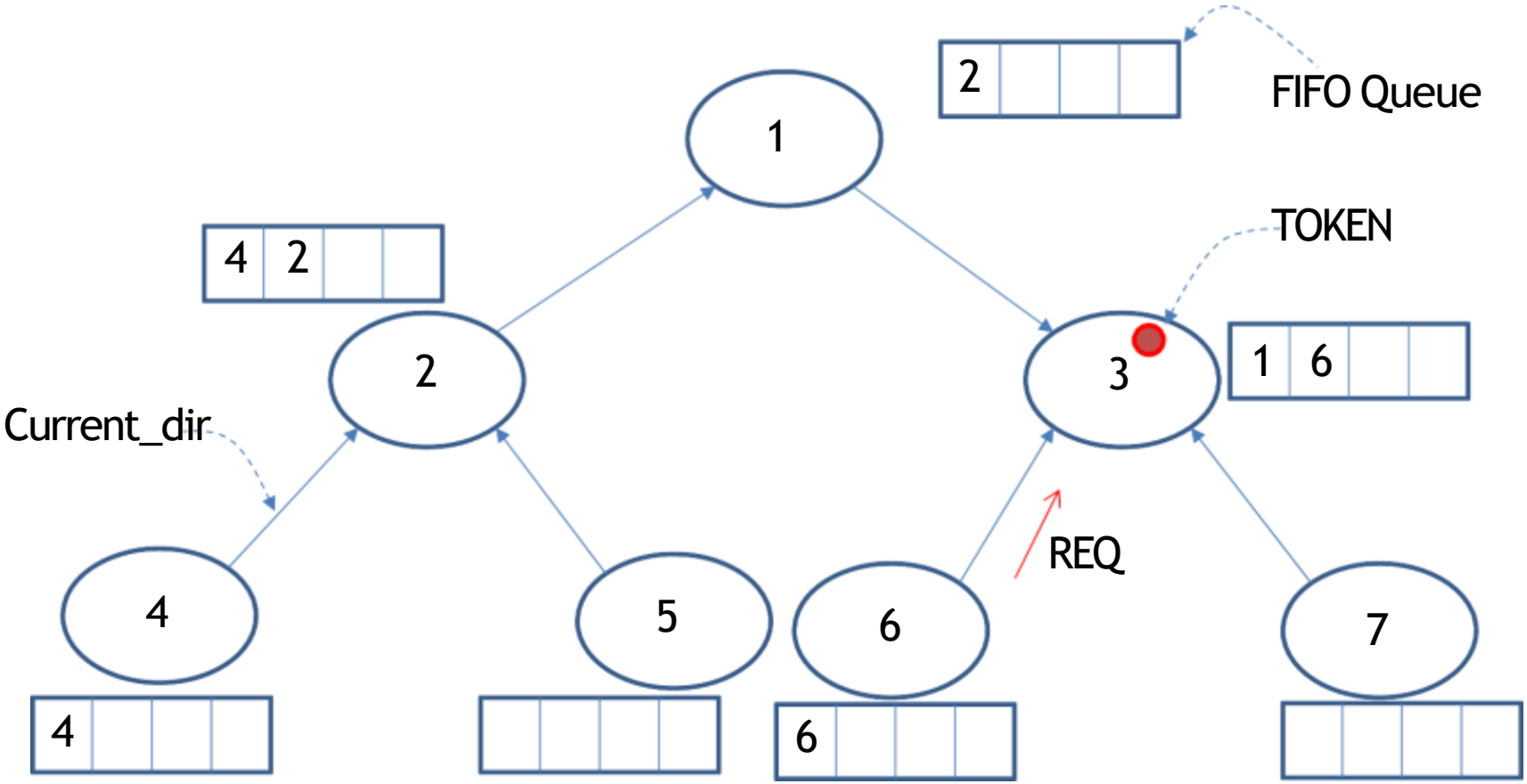
- The algorithm operates on a minimal spanning tree of the network topology.
- A node needs to know about and communicate only to its immediate-neighboring nodes.
- Only one node can have the privilege/token (called the privileged node) at any time, except when the token is transferred from one node to another via message.
- When there are no nodes requesting for the privilege, it remains in possession of the node that last used it.

# Raymond's Tree based DE Algorithm Example




Node 4 made the request to enter its CS.
Its REQUEST will be sent to node 3 (TOKEN holder) via node 2 and node 1 after storing the node ids in the respective FIFO Queue of nodes.

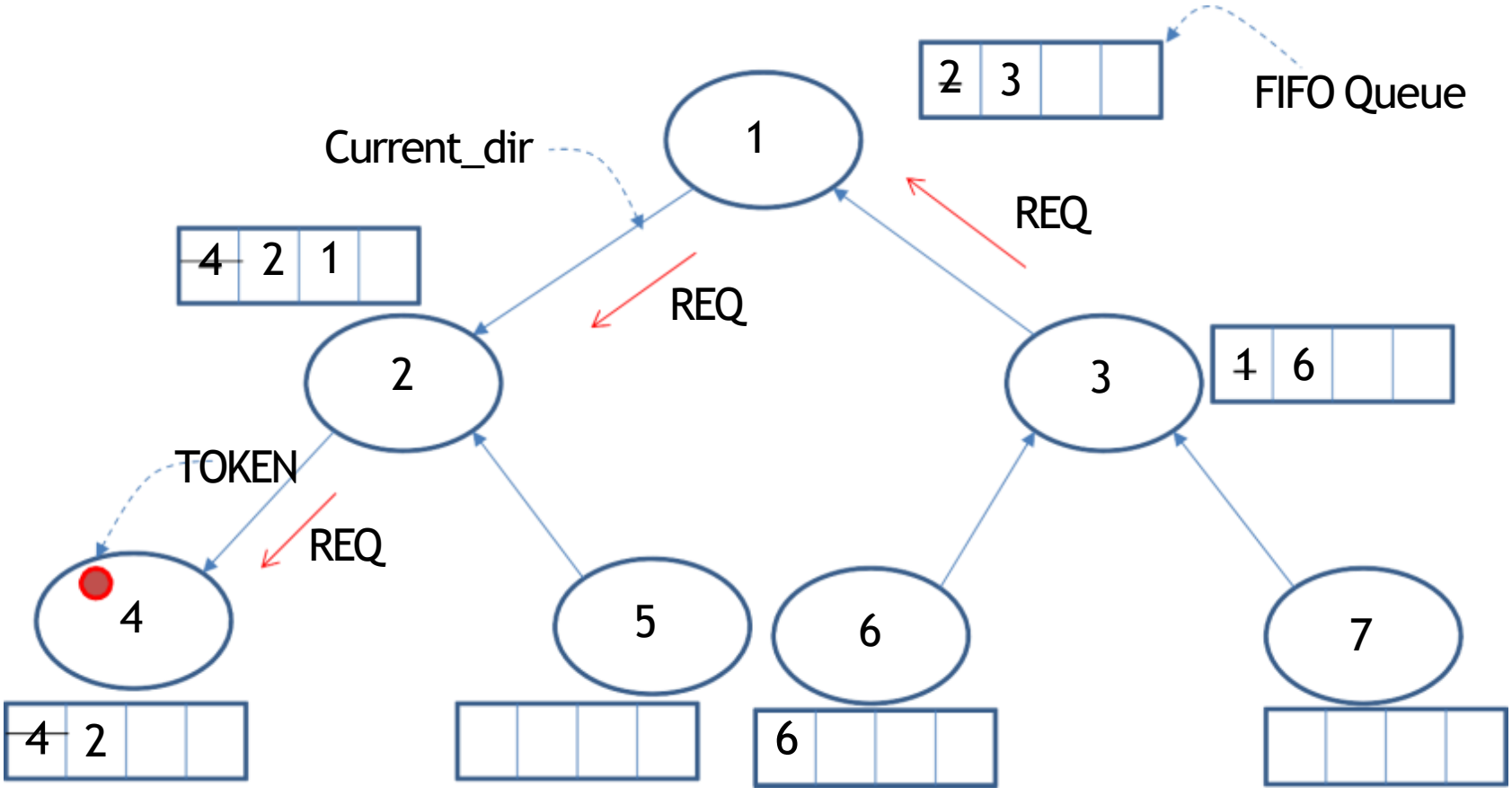
# Raymond's Tree based DME Algorithm Example




After node 4, node 2 and node 6 made a request.

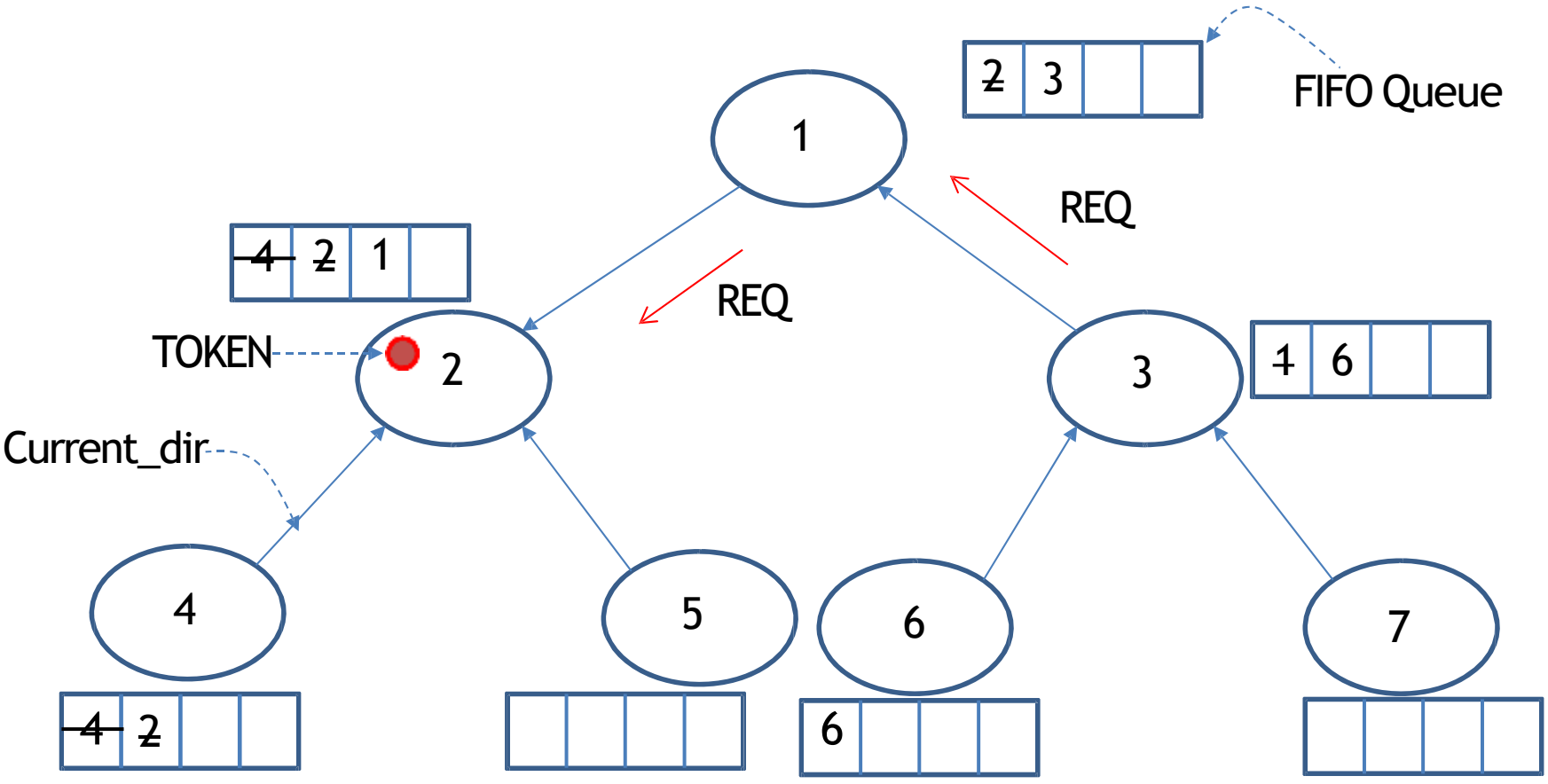
The request of node 2 is not propagated to node 1 since its already there, instead it is stored in its FIFO Queue .However, REQUEST of node 6 is propagated to node 3 after storing it in its FIFO Queue. Node 3 also stores the id of node 6 in its FIFO Queue.

# Raymond's Tree based DME Algorithm Example



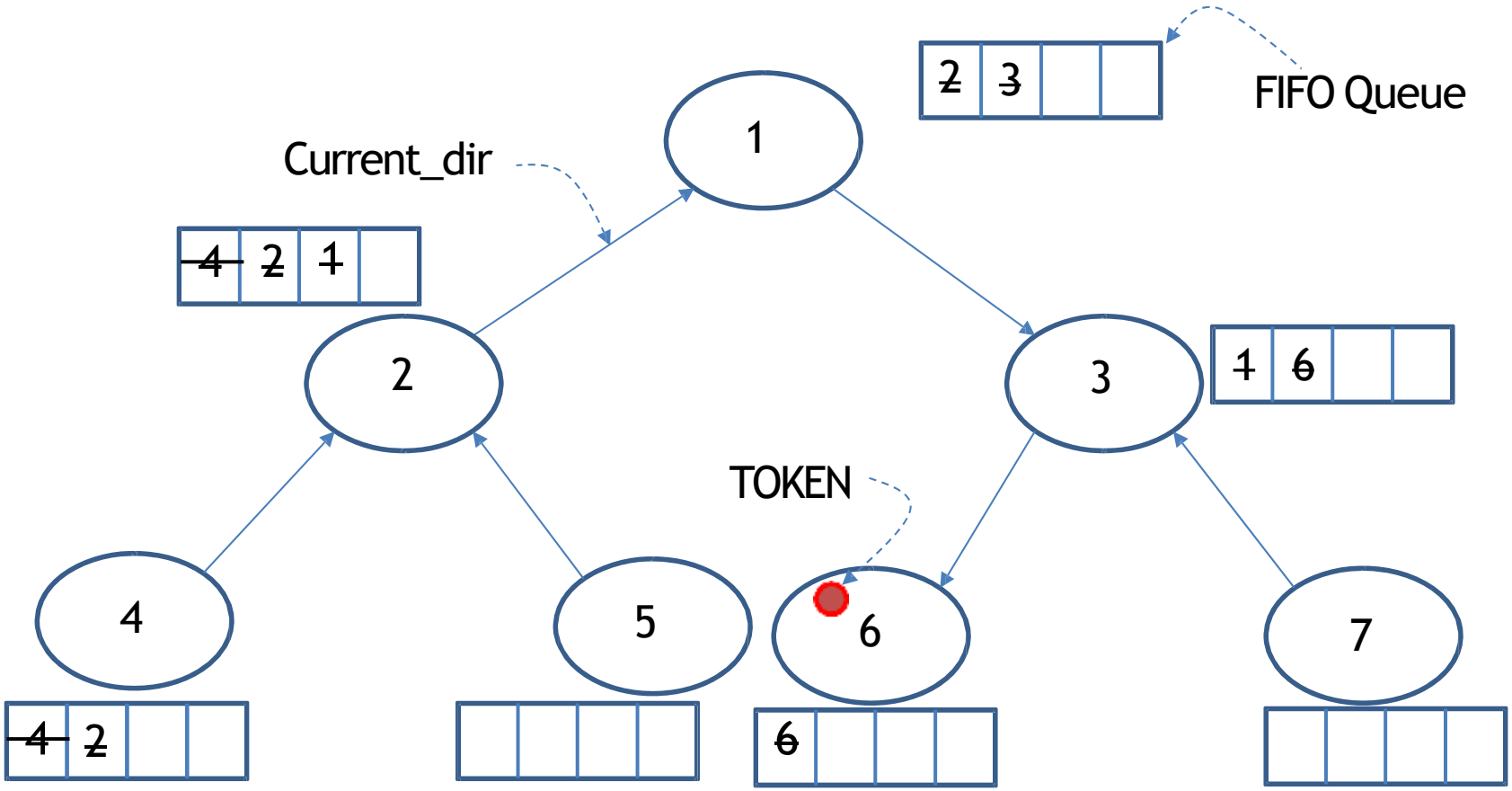

After completion of execution of CS by node 3, it will send the TOKEN to node 1. Since its FIFO Queue is not empty, it also forwards REQUEST to node 1. Node 1 will forward the TOKEN and REQUEST (since its FIFO Queue is not empty) to node 2 and then subsequently to node 4. Finally node 4 will be the TOKEN holder and its FIFO Queue contains id 2 .

# Raymond's Tree based DME Algorithm Example




After completion of execution of CS by node 4, it sends the token to node 2 and once node 2 gets the token and its request is at the head of FIFO Queue, it enters its CS.

# Raymond's Tree based DME Algorithm Example




After completion of execution of CS by node 2, it sends the token to node 1, node 1 will send TOKEN to node 3 and finally it reaches node 6. Note that there will not be any REQUEST messasges from node 2 or node 1 or node 3 since their FIFO Queue are empty. Once node 6 gets the TOKEN and its request is at the head of FIFO Queue, it enters its CS.

# Data Structures
## Each node to maintains the following variables:

| Variable Name | Possible Values | Comments |
|---|---|---|
| HOLDER | "self" or the identity of one of the immediate neighbours. | Indicates the location of the privileged node in relation to the current node. "self" means current node is privileged. |
| USING | True or false. | Indicates if the current node is executing the critical section. |
| REQUEST Q | A FIFO queue that could contain "self" or the identities of immediate neighbors as elements. | The REQUEST Q of a node consists of the identities of those immediate neighbors that have requested for privilege but have not yet been sent the privilege . Max size is neighbors+1. |
| ASKED | True or false. | Indicates if node has sent a request for the privilege. |

# The Algorithm

It consists of the following routines for each node:-

ASSIGN PRIVILEGE

MAKE REQUEST

**ASSIGN PRIVILEGE:**

This routine sends a PRIVILEGE message if:-

- The current node holds the privilege but is not using it.
- its REQUEST_Q is not empty.
- the element at the head of its REQUEST_Q is not “self.”
- When “self” is at the head of REQUEST_Q, the node will enter into its critical section after removing “self” from the head of REQUEST_Q.
- Also, the variable ASKED is set to false since the currently privileged node will not have sent a request for the PRIVILEGE message.

# MAKE REQUEST

This routine sends a REQUEST message if:-

- it does not hold the privilege,
- its REQUEST Q is not empty, i.e., it requires the privilege.
- it has not sent a REQUEST message already.
- The variable ASKED of a node will be true when it has sent REQUEST_Q message to an immediate neighbor and has not received a response.
- A node does not send any REQUEST messages if ASKED is true. Thus the variable ASKED makes sure that unnecessary REQUEST messages are not sent from the unprivileged nodes.
- This makes the REQUEST_Q of any node bounded, even when operating under heavy load.

# Message handling section of each node

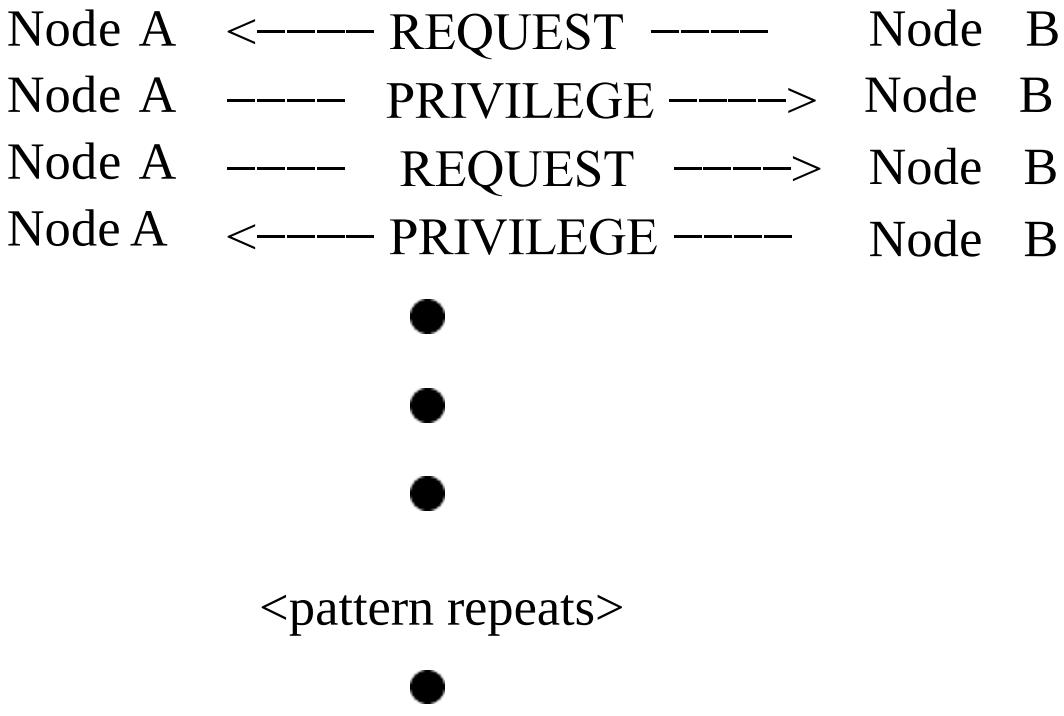
Below we show four events that constitute the algorithm.

| Event | Algorithm Functionality |
|---|---|
| A node wishes to execute critical section. | Enqueue(REQUEST_Q, self); ASSIGN_PRIVILEGE; MAKE_REQUEST |
| A node receives a REQUEST message from one of its immediate neighbors X. | Enqueue(REQUEST_Q, X); ASSIGN_PRIVILEGE; MAKE_REQUEST |
| A node receives a PRIVILEGE message. | HOLDER := self; ASSIGN_PRIVILEGE; MAKE_REQUEST; |
| A node exits the critical section. | USING := false; ASSIGN_PRIVILEGE; MAKE_REQUEST |

# Key features of Raymond's algorithm

## 1. Mutual Exclusion is guaranteed

The algorithm ensures that at any instant of time at most one node holds the privilege. During transit stage, there are no privileged nodes.




Logical pattern of message flow between neighboring nodes A and B.

# 2. Message Overtaking has no effect

- This algorithm does away with the use of sequence numbers since message flow between any two neighboring nodes sticks to a logical pattern.
- If at all message overtaking occurs between the nodes A and B, it can occur when a PRIVILEGE message is sent from node A to node B closely followed by a REQUEST message.
- Such a message overtaking will not affect the operation of the algorithm.
- If node B receives the REQUEST message from node A before receiving the PRIVILEGE message from node A, A's request will be queued in REQUEST $Q_B$. Since B is not a privileged node, it will not be able to send a privilege to node A in reply.

# 3. Deadlock is Impossible

When the critical section is free, and one or more nodes want to enter the critical section but are not able to do so, a deadlock may occur. In some DME algorithms deadlock may happen due to any of the following scenarios:

- The privilege cannot be transferred to a node because no node holds the privilege. This can't happen since we have assigned privilege to one node initially which keeps on passing.
- The node in possession of the privilege is unaware that there are other nodes requiring the privilege. This can't happen since the request message is bound to reach the privileged node via proxies.
- The PRIVILEGE message does not reach the requesting unprivileged node. This is also not possible in this algorithm.

# 4. Starvation is Impossible

- When a node A holds the privilege, and another node B requests for the privilege, the identity of B or the id's of proxy nodes for node B will be present in the REQUEST Qs of various nodes in the path connecting the requesting node to the currently privileged node.
- So depending upon the position of the id of node B in those REQUEST Qs, node B will sooner or later receive the privilege.So starvation is impossible.

# Performance Analysis of Raymond's algorithm

- In the worst-case, the algorithm requires (2 * longest path length of the tree) messages per critical section entry.
- This happens when the privilege is to be passed between nodes at either ends of the longest path of the minimal spanning tree.
- The longest path length of tree is typically O(log N) where this algorithm involves exchange of O(log N) messages per CS execution in worst case. Thus it is one of the most efficient DME algorithms available.
- The worst possible network topology for this algorithm is where all nodes are arranged in a straight line. In that case the longest path length will be N – 1, and thus the algorithm will exchange 2 * (N – 1) messages per CS execution in worst case.
- Under heavy load, the algorithm exhibits an interesting property: "As the number of nodes requesting for the privilege increases, the number of messages exchanged per critical section entry decreases." In heavy load, the algorithm requires exchange of only four messages per CS execution.

Thank You