Leader Election in Rings (Classical Distributed Algorithms)

Dr. Rajiv Misra
Associate Professor
Dept. of Computer Science & Engg.
Indian Institute of Technology Patna
rajivm@iitp.ac.in

Preface

Content of this Lecture:

- In this lecture, we will discuss the leader election problem in message-passing systems for a ring topology, in which a group of processors must choose one among them to be a leader.

- We will present the different algorithms for leader election problem by taking the cases like **anonymous/ non-anonymous rings, uniform/ non-uniform rings and synchronous/ asynchronous rings etc.**

Leader Election (LE) Problem: Introduction

- The leader election problem has several variants.
- LE problem is for each processor to decide that either it is **the leader or non-leader**, subject to the constraint that exactly one processor decides to be the leader.
- LE problem represents a general class of ***symmetry-breaking*** problems.
- For example, when a deadlock is created, because of processors waiting in a cycle for each other, the deadlock can be broken by electing one of the processor as a leader and removing it from the cycle.

Leader Election: Definition

- Each processor has a set of **elected (won)** and **not-elected (lost)** states.

- Once an elected state is entered, processor is always in an elected state (and similarly for not-elected): i.e., irreversible decision

- In every admissible execution:
  - every processor eventually enters either an elected or a not-elected state
  - exactly one processor (the **leader**) enters an elected state

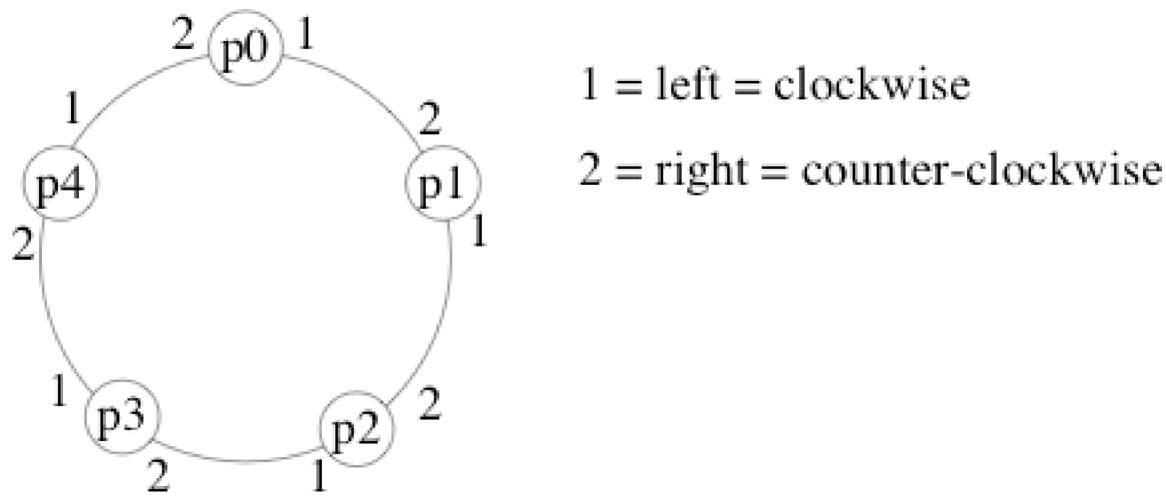
Uses of Leader Election

- A leader can be used to coordinate activities of the system:
  - find a ***spanning tree*** using the leader as the root
  - reconstruct a ***lost token*** in a token-ring network

- In this lecture, we will study the leader election in rings.

Definition: (1)Ring Networks

- In an **oriented ring**, processors have a consistent notion of left and right



- For example, if messages are always forwarded on channel 1, they will cycle clockwise around the ring

Definition: (2) Anonymous Rings

- How to model situation when **processors do not have unique identifiers?**

- **First attempt:** require each processor to have the same state machine

Definition: (3) Uniform (Anonymous) Algorithms

- A **uniform** algorithm **does not use the ring size** (same algorithm for each size ring)
  - Formally, every processor in every size ring is modeled with the same state machine

- A **non-uniform** algorithm **uses the ring size** (different algorithm for each size ring)
  - Formally, for each value of n, every processor in a ring of size n is modeled with the same state machine A_n .

- Note the lack of unique ids.

Impossibility: Leader Election in Anonymous Rings

Theorem: There is ***no* leader election algorithm for anonymous rings,** even if

algorithm knows the ring size (non-uniform) and synchronous model

Proof Sketch:

- Every processor begins in same state with same outgoing messages (since anonymous)
- Every processor receives same messages, does same state transition, and sends same messages in round 1
- Ditto for rounds 2, 3, …
- Eventually some processor is supposed to enter an elected state. But then they all would.

Leader Election in Anonymous Rings

- Proof sketch shows that either safety (never elect more than one leader) or liveness (eventually elect at least one leader) is violated.

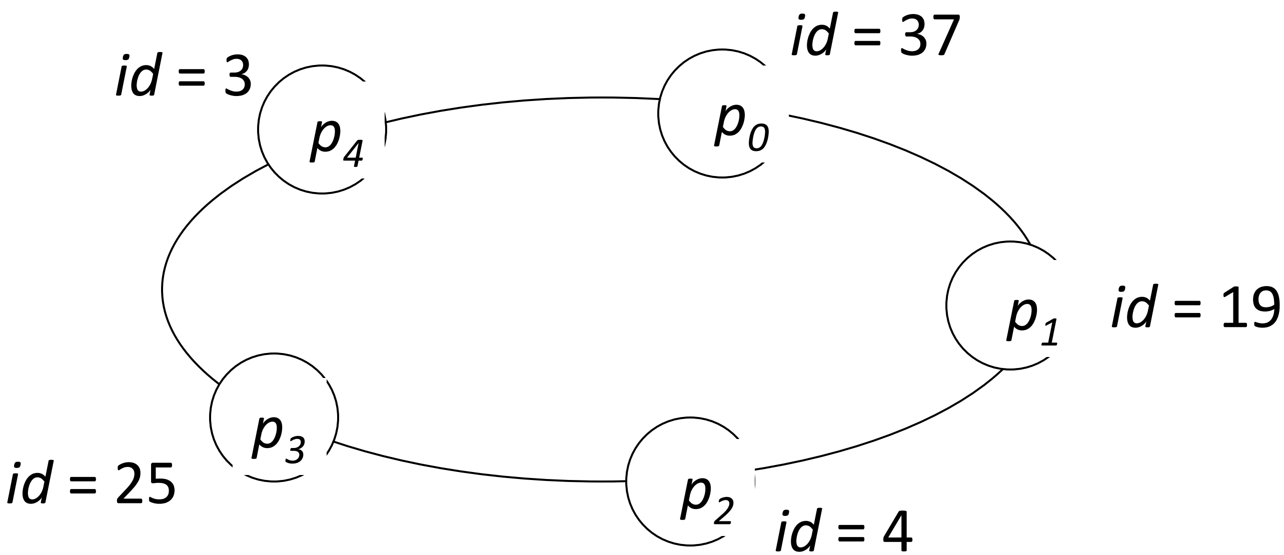
- Since the theorem was proved for non-uniform and synchronous rings, the same result holds for weaker (less well-behaved) models:
  - uniform
  - asynchronous

Rings with Identifiers

- Assume each processor has a unique id.
- Don't confuse indices and ids:
  - **indices** are 0 to $n - 1$; used only for analysis, not available to the processors
  - **ids** are arbitrary nonnegative integers; are available to the processors through local variable *id.*

Specifying a Ring

- Start with the smallest id and list ids in clockwise order.



- Example: 3, 37, 19, 4, 25

Uniform (Non-anonymous) Algorithms

- **Uniform** algorithm: there is one state machine for every id, no matter what size ring
- **Non-uniform** algorithm: there is one state machine for every id and every different ring size
- These definitions are tailored for leader election in a ring.

O(n^2) Messages LE Algorithm: LeLann-Chang-Roberts (LCR) algorithm

- send value of own *id* to the left
- when receive an id *j* (from the right):
  - if $j > id$ then
    - forward *j* to the left (this processor has lost)
  - if $j = id$ then
    - elect self (this processor has won)
  - if $j < id$ then
    - do nothing

Analysis of $O(n^2)$ Algorithm

Correctness: Elects processor with largest id.

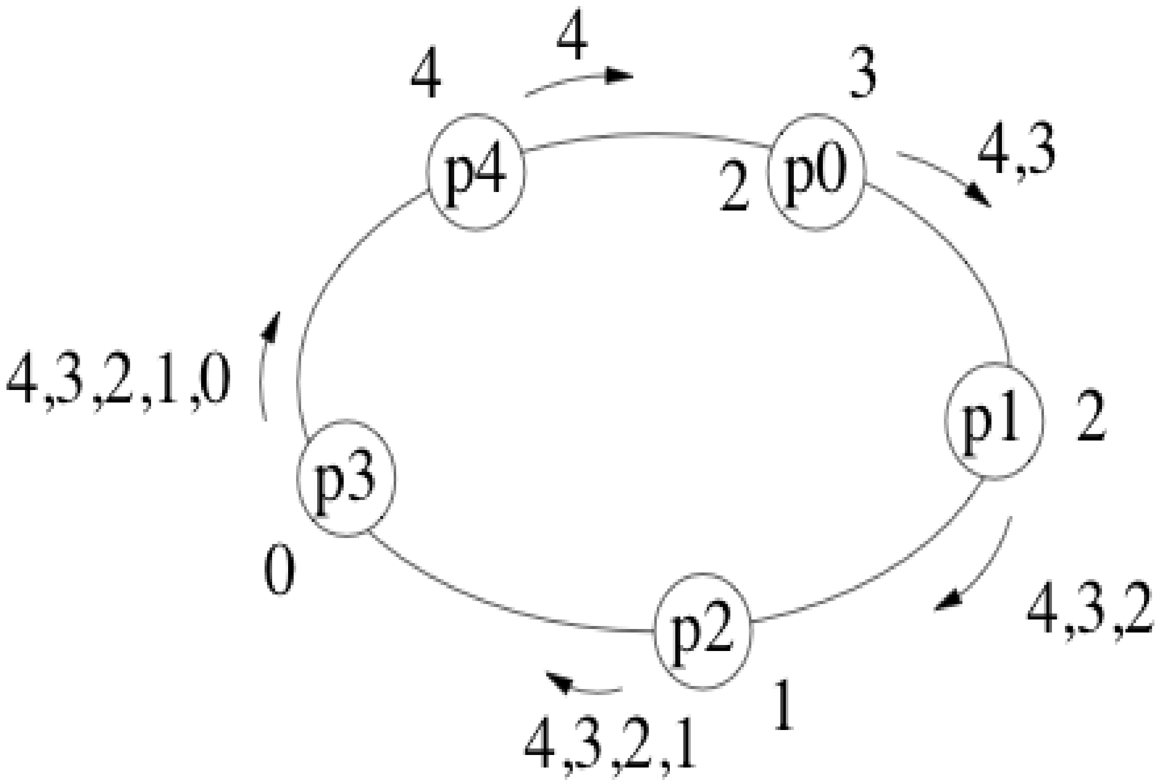
- message containing largest id passes through every processor

Time: $O(n)$

Message complexity: Depends how the ids are arranged.

- largest id travels all around the ring (n messages)
- 2nd largest id travels until reaching largest
- 3rd largest id travels until reaching largest or second largest etc.

Analysis of $O(n^2)$ Algorithm



Analysis of $O(n^2)$ Algorithm

- Worst way to arrange the ids is in decreasing order:
  - 2nd largest causes $n - 1$ messages
  - 3rd largest causes $n - 2$ messages etc.

- **Total number of messages is:**

$$n + (n-1) + (n-2) + \ldots + 1 = \Theta(n^2)$$

Analysis of $O(n^2)$ Algorithm

- Clearly, the algorithm never sends more than $O(n^2)$ messages in any admissible execution. Moreover, there is an admissible execution in which the algorithm sends $\Theta(n^2)$ messages; Consider the ring where the identifiers of the processor are *0,……, n-1* and they are ordered as in Figure 3.2. In this configuration, the message of processor with identifier i is send exactly $i+1$ times, Thus the total number of messages, including the n termination messages, is

$$n + \sum_{i=0}^{n-1} (i+1) = \Theta(n^2)$$

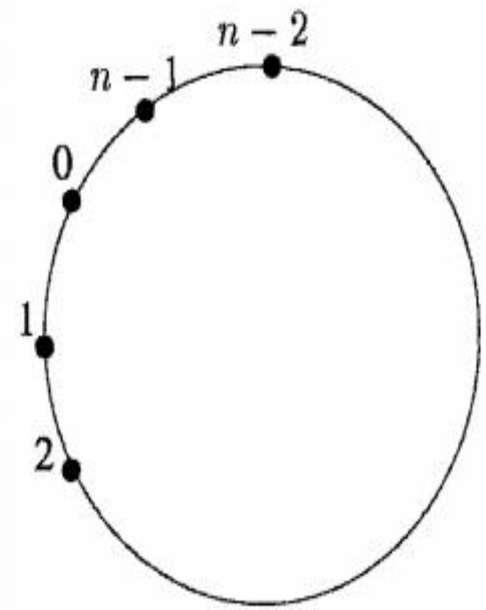


Fig. 3.2 Ring with $\Theta(n^2)$ messages.

Clockwise Unidirectional Ring

Can We Use Fewer Messages?

- The $O(n^2)$ algorithm is simple and works in both synchronous and asynchronous model.
- But can we solve the problem with fewer messages?

Idea:

- Try to have messages containing smaller ids travel smaller distance in the ring

O(*nlogn*) Messages LE Algorithm: The Hirschberg and Sinclair (HS) algorithm

- To describe the algorithm, we first define the ***k-neighbourhood*** of a **processor p_i** in the ring to be the set of processors that are at **distance at most *k* from p_i in the ring** (either to the left or to the right). Note that the ***k-neighbourhood*** of a processor includes exactly ***2k+1*** processors.
- The algorithm operates in phases; it is convenient to start numbering the phases with 0. In the ***kth* phase** a processor tries to become a ***winner*** for that phase; to be a winner, it must have the **largest id** in its ***2^k-neighborhood***. Only processors that are winners in the **kth phase** continue to compete in the ***(k+1)*-st phase**, Thus fewer processors proceed to higher phases, until at the end, only one processor is a winner and it is elected as the leader of the whole ring.

The HS Algorithm: Sending Messages

Phase 0

- In more detail, in phase 0, each processor attempts to become a phase 0 winner and sends a **<probe>** message containing its identifier to its ***1-neighborhood***, that is, to each of its two neighbors.
- If the identifier of the neighbor receiving the probe is greater than the identifier in the probe, it swallows the probe; otherwise, it sends back a **<reply>** message.
- If a processor receives a reply from both its neighbors, then the processor becomes a phase 0 winner and continues to phase 1.

The HS Algorithm: Sending Messages

Phase k

- In general, in phase *k*, a processor p_i that is a phase ***k-1* winner** sends **<probe>** messages with its identifier to its ***2^k-neighborhood*** (one in each direction). Each such message traverses ***2^k processors*** one by one, A probe is swallowed by a processor if it contains an identifier that is smaller than its own identifier.
- If the probe arrives at the last processor on the neighbourhood without being swallowed, then that last processor sends back a **<reply>** message to p_i. If p_i receives replies from both directions, it becomes a phase k winner, and it continues to phase ***k+1***. A processor that receives its own **<probe>** message terminates the algorithm as the leader and sends a termination message around the ring.

Algorithm 5 Asynchronous leader election: code for processor p_i, $0 \le i < n$.

```
Initially, asleep = true

upon receiving no message:
    if asleep then
        asleep := false
        send ⟨probe,id,0,1⟩ to left and right

upon receiving ⟨probe,j,k,d⟩ from left (resp., right):
    if j = id then terminate as the leader
    if j > id and d < 2^k then                          // forward the message
        send ⟨probe,j,k,d + 1⟩ to right (resp., left)   // increment hop counter
    if j > id and d ≥ 2^k then                          // reply to the message
        send ⟨reply,j,k⟩ to left (resp., right)
                                                        // if j < id, message is swallowed

upon receiving ⟨reply,j,k⟩ from left (resp., right):
    if j ≠ id then send ⟨reply,j,k⟩ to right (resp., left)   // forward the reply
    else                                                     // reply is for own probe
        if already received ⟨reply,j,k⟩ from right (resp., left) then
            send ⟨probe,id,k + 1,1⟩                          // phase k winner
```

The HS Algorithm

- The pseudocode appears in **Algorithm 5**. Phase *k* for a processor corresponds to the period between its sending of a **<probe>** message in line 4 or 15 with third parameter k and its sending of a **<probe>** message in line 4 or 15 with third parameter ***k+1***. The details of sending the termination message around the ring have been left out in the code, and only the leader terminates.

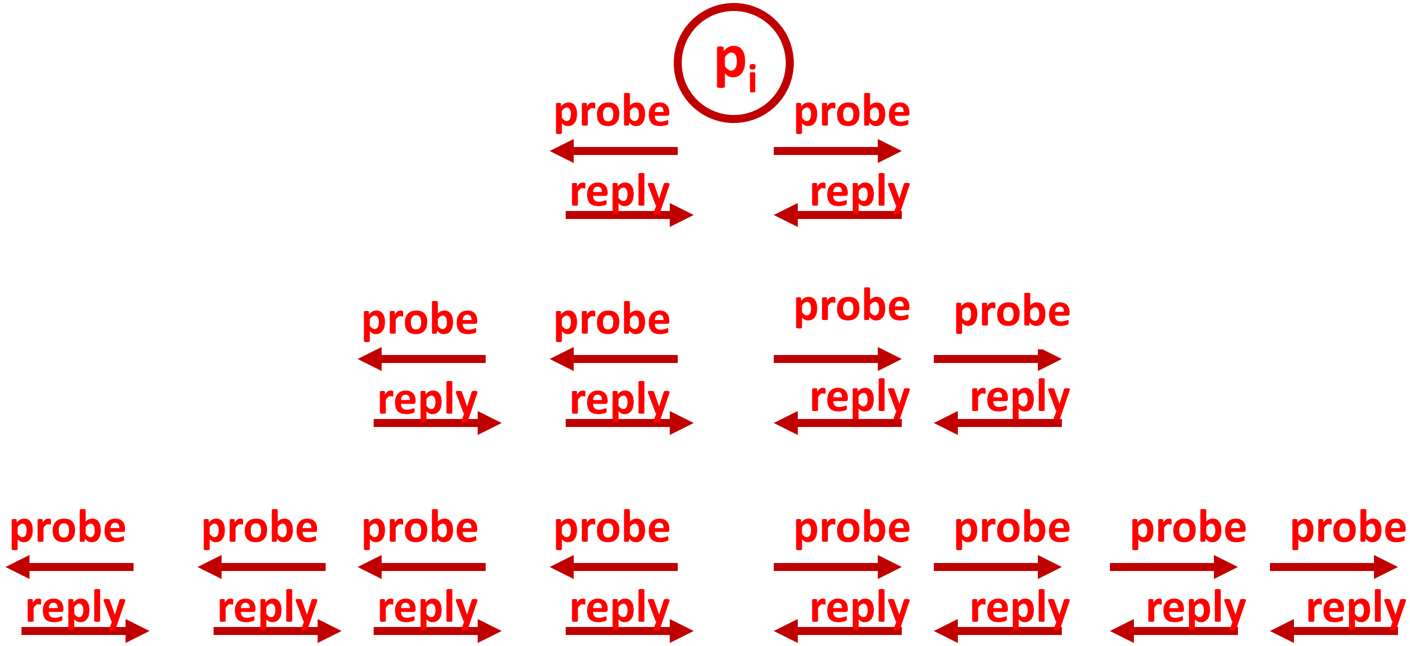
- The correctness of the algorithm follows in the same manner as in the simple algorithm, because they have the same swallowing rules.

- It is clear that the probes of the processor with the maximal identifier are never swallowed; therefore, this processor will terminate the algorithm as a leader. On the other hand, it is also clear that no other **<probe>** can traverse the whole ring without being swallowed. Therefore, the processor with the maximal identifier is the only leader elected by the algorithm.

O(n log *n)* Leader Election Algorithm

- Each processor tries to probe successively larger neighborhoods in both directions
  - size of neighborhood ***doubles*** in each phase
- If probe reaches a node with a larger id, the probe stops
- If probe reaches end of its neighborhood, then a reply is sent back to initiator
- If initiator gets back replies from both directions, then go to next phase
- If processor receives a probe with its own id, it elects itself

$O(n \log n)$ Leader Election Algorithm



Analysis of $O(n \log n)$ Leader Election Algorithm

Correctness:

- Similar to $O(n^2)$ algorithm.

Message Complexity:

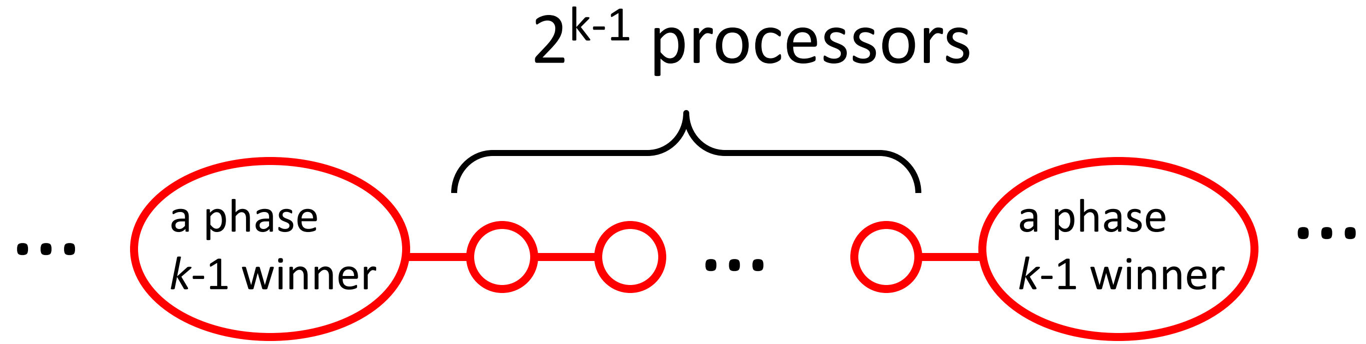
- Each message belongs to a particular phase and is initiated by a particular processor
- Probe distance in phase k is 2^k
- Number of messages initiated by a processor in phase k is at most $4*2^k$ (probes and replies in both directions)

Analysis of *O(n* log *n)* Leader Election Algorithm

- How many processors initiate probes in phase k ?
- For $k = 0$, every processor does
- For $k > 0$, every processor that is a "winner" in phase $k - 1$ does
  - "winner" means has largest id in its 2^{k-1} neighborhood

Analysis of *O(n* log *n)* Leader Election Algorithm

- Maximum number of phase *k - 1* winners occurs when they are packed as densely as possible:



- **Total number of phase *k - 1* winners is at most**

  $n/(2^{k-1} + 1)$

Analysis of $O(n \log n)$ Leader Election Algorithm

- How many phases are there?
- At each phase the number of (phase) winners is cut approx. in half
  - **from $n/(2^{k-1} + 1)$ to $n/(2^k + 1)$**
- So after approx. $\log_2 n$ phases, only one winner is left.
  - more precisely, **max phase is $\lceil \log(n-1) \rceil + 1$**

Analysis of *O(n log n)* Leader Election Algorithm

- Total number of messages is sum, over all phases, of number of winners at that phase times number of messages originated by that winner:

$$\leq 4n + n + \boxed{\sum_{k=1}^{\lceil \log(n-1) \rceil + 1} 4 \bullet 2^k \bullet n/(2^{k-1}+1)}$$

$$< 8n(\log n + 2) + 5n$$

$$= O(n \log n)$$

phase 0 msgs

termination msgs

msgs for phases 1 to $\lceil \log(n-1) \rceil+1$

Can We Do Better?

- The $O(n \log n)$ algorithm is more complicated than the $O(n^2)$ algorithm but uses fewer messages in worst case.
- Works in both synchronous and asynchronous case.
- Can we reduce the number of messages even more?
- Not in the asynchronous model…

Lower bound for LE algorithm

But, can we do better than O(n log n)?

Theorem: Any leader election algorithm for asynchronous rings whose size is not known a priori has $\Omega(n \log n)$ message complexity (holds also for unidirectional rings).

- Both LCR and HS are comparison-based algorithms, i.e. they use the identifiers only for comparisons (<; >;=).
- In synchronous networks, O(n) message complexity can be achieved if general arithmetic operations are permitted (non-comparison based) and if time complexity is unbounded.

Overview of LE in Rings with Ids

- There exist algorithms when nodes have unique ids.
- We have evaluated them according to their ***message complexity***.
- **Asynchronous ring:**
  - $\Theta(n \log n)$ messages
- **Synchronous ring:**
  - $\Theta(n)$ messages under certain conditions
  - otherwise $\Theta(n \log n)$ messages
- All bounds are asymptotically tight.

Conclusion

- This lecture provided an in-depth study of the leader election problem in message-passing systems for a ring topology.

- We have presented the different algorithms for leader election problem by taking the cases like **anonymous/non-anonymous rings, uniform/non-uniform rings and synchronous/ asynchronous rings**