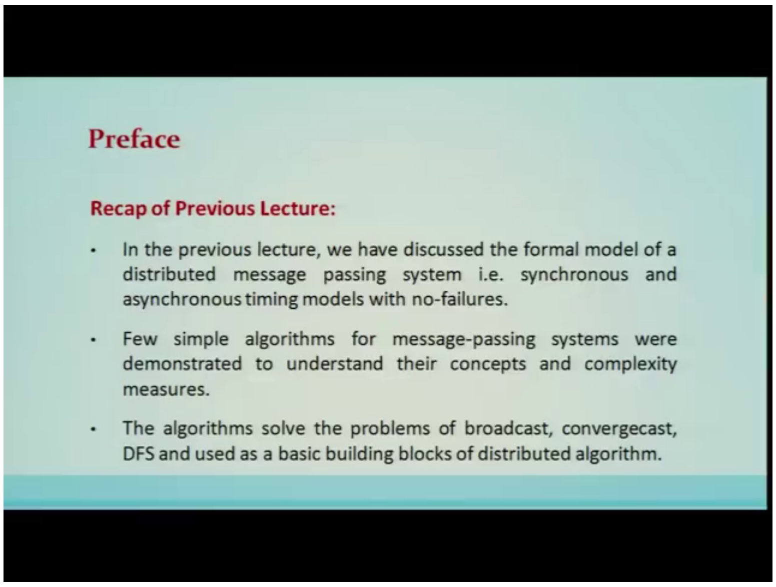
Distributed Systems

Dr. Rajiv Misra
Department of Computer Science and Engineering
Indian Institute of Technology, Patna

Lecture - 03
Leadre Election in Rings

Lecture 3: leader election in the rings preface recap of previous lecture.

(Refer Slide Time: 00:24)



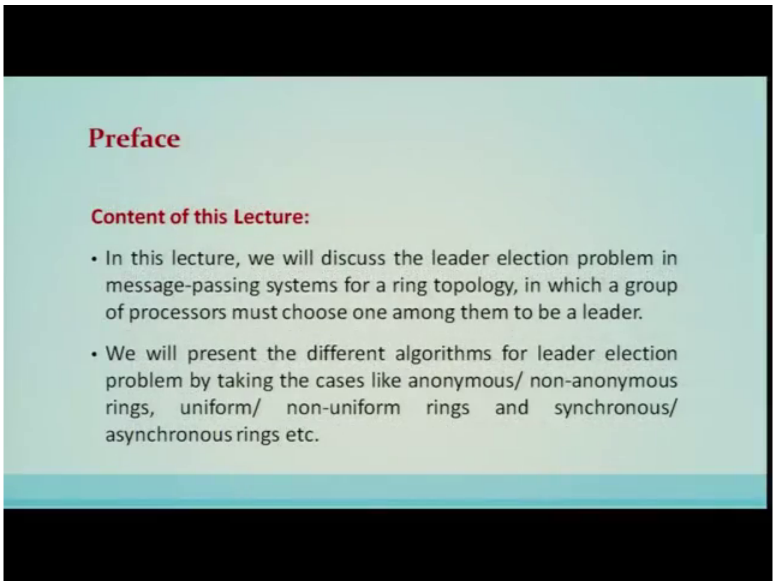
Preface

Recap of Previous Lecture:

- In the previous lecture, we have discussed the formal model of a distributed message passing system i.e. synchronous and asynchronous timing models with no-failures.
- Few simple algorithms for message-passing systems were demonstrated to understand their concepts and complexity measures.
- The algorithms solve the problems of broadcast, convergecast, DFS and used as a basic building blocks of distributed algorithm.

In previous lecture we have discussed the formal model of distributed message passing system that is synchronous and asynchronous timing models with no failures. We have seen in the previous lecture, a few simple algorithms for message passing systems and these algorithms were to understand the concepts and the complexity measures in the distributed algorithm design the algorithms which we have seen in the previous lecture used to solve the problem of broadcast, converge cast, DFS and is being used as the basic building block for the distributed algorithms.

(Refer Slide Time: 01:06)



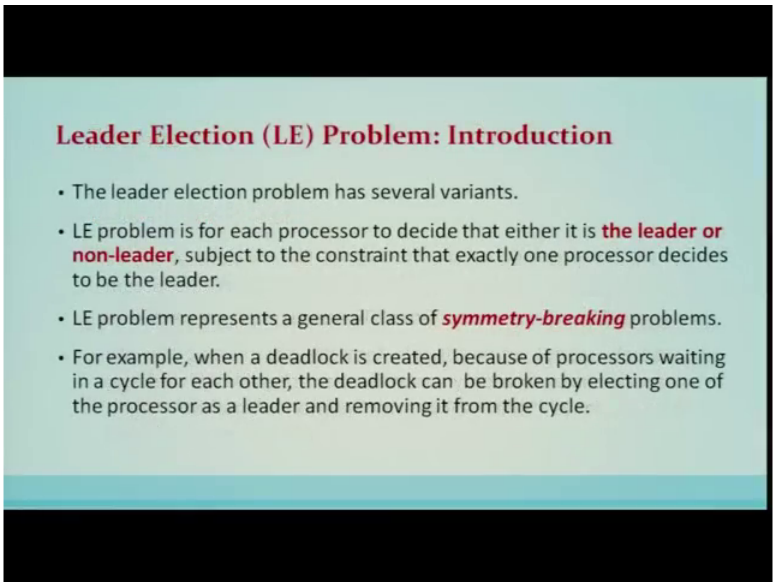
Preface

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.

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

(Refer Slide Time: 01:37)



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.

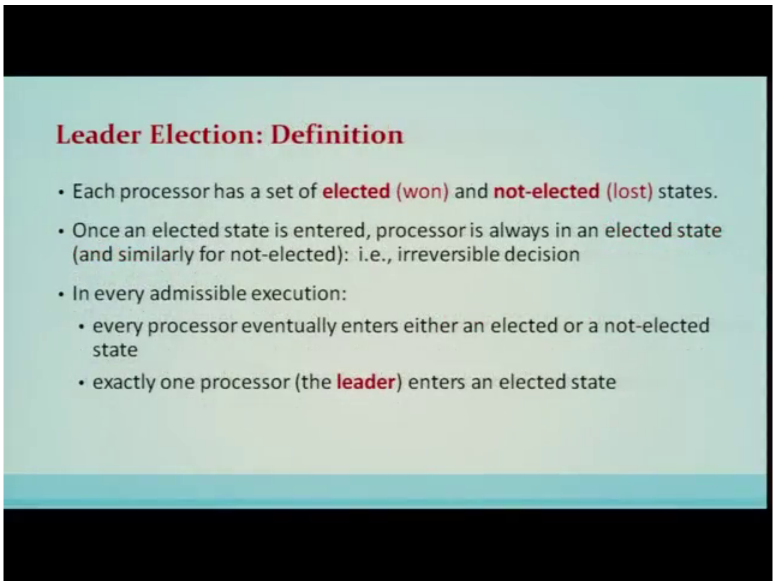
We will present the different algorithms for leader election problems by taking cases like anonymous or a non anonymous rings uniform or non uniform rings and synchronous and asynchronous rings.

So, let us begin this particular introduction of leader election problems. So, in this particular lecture we are considering the topology of a message passing system as a ring. So, ring is a convenient topology and is basically resembles to the physical token ring and corresponds to the ring structure of a physical ring and is easy to understand or design the algorithms in this particular setting that is basically the ring structure.

So, the leader election problem has several variants leader election problem is for each processor to decide that either it is a leader or a non leader subject to the constraint that exactly one processor decides to be a leader. So, leader election problem represents a general class of symmetry breaking problems; so, for example, when a deadlock is created because of the processors waiting in a cycle for each other.

This particular deadlock can be broken by electing one of the waiting processors as the leader and removing it from the cycle and thus breaking the deadlock. So, this is called this particular example is a example where the symmetry breaking is applied in the form of a leader election algorithm.

(Refer Slide Time: 03:05)

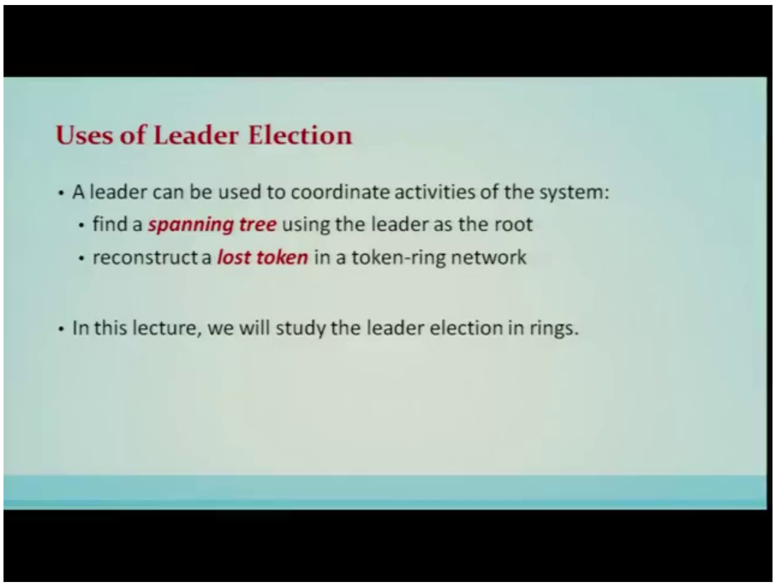


Leader Election: Definition

- 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

So, leader election definition the processor each processor has a set of elected that is one and non-elected states. So, once a elected state is entered the processor is always in an elected state and basically similarly for the non elected states. So, in every admissible execution every processor eventually enters an elected or a non elected state and exactly one processor that is the leader enters an elected state; so, again the use of leader election algorithm.

(Refer Slide Time: 03:47)



Uses of Leader Election

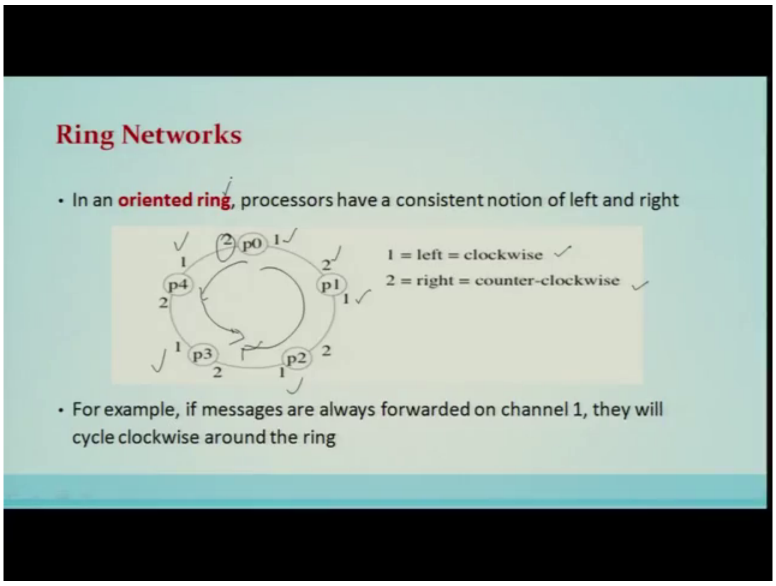
- A leader can be used to coordinate activities of the system:
  - find a ***spanning tree*** using the leader as the root
  - reconstruct a ***lost token*** in a token-ring network
- In this lecture, we will study the leader election in rings.

We are going to a stress upon this use to motivate the basically construction of the leader election algorithm.

So, leader election can be used to coordinate activities in a distributed system; so, for example, finding a spanning tree using the leader as the route. So, we have seen that it becomes easy to basically construct a spanning tree that is DFS, if the route is given and to identify the route in a network leader election algorithm can be used.

Similarly, if in a token ring network if a token is lost, then to reconstruct the lost token a leader election algorithm can be of great help. So, leader election algorithm will identify one of the node and that particular node will recreate the token and will be start resume the token ring a structure. So, in this lecture, we will study the leader election in the message passing system that is called the ring structure.

(Refer Slide Time: 04:50)



Ring Networks

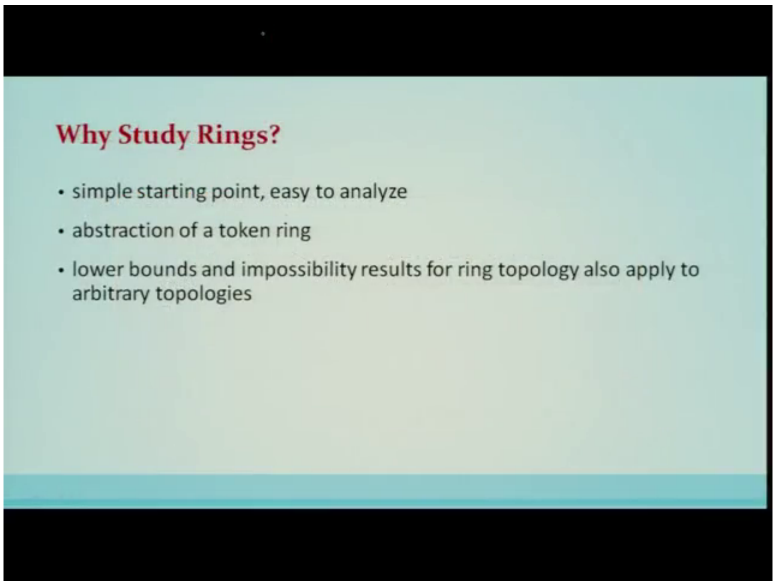
- In an **oriented ring**, processors have a consistent notion of left and right

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

So, we are going to now describe about the ring topology. So, in an oriented ring processor have a consistent notion of left and right. So, for example, this particular diagram which is illustrated over here you can see the processors p 0, p 1, p 2, p 3 and p 4; they have p 0 and p 1; they have the channel. So, p 0 is connected to p 1 on the left side, it is numbered as 1 and p 1 is connected to p 0 with a number that is 2 that is called the right side. So, if you see all the numbers as 1, then the orientation of the ring is basically nothing, but the orientation of a ring is a clockwise. So, if we basically keep on navigating over the number 1.

Similarly, if we are navigating over number 2 that is p 0 is basically communicating with p 4 and p 4 is p 3 that is through the number 2, then basically it is a counter clockwise. So, this is called oriented ring. So, orientation of a ring can be basically formed using these a numbering of a channel and that is that to be done by the processors at its own level there is a local level with a local knowledge.

(Refer Slide Time: 06:31)



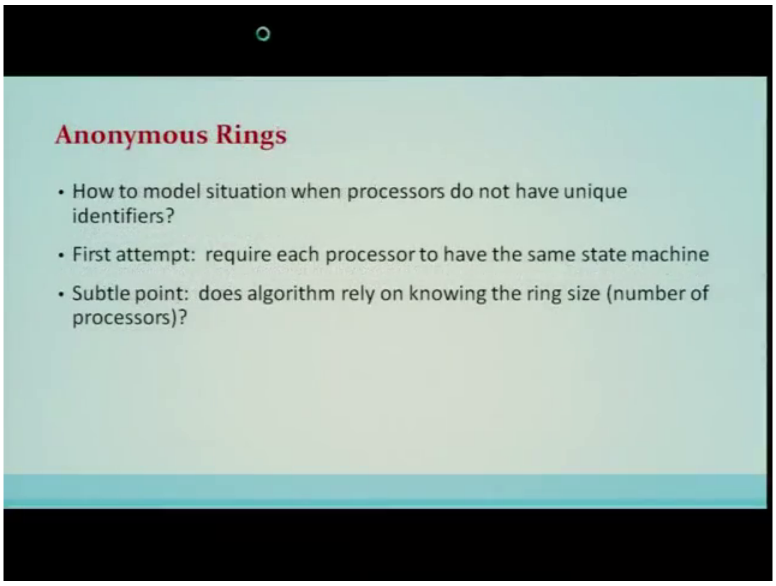
Why Study Rings?

- simple starting point, easy to analyze
- abstraction of a token ring
- lower bounds and impossibility results for ring topology also apply to arbitrary topologies

So, for example, if the messages are always forwarded on a channel one there will be cycle clockwise around the ring. So, why we study the ring because in a message passing system which is basically used as a ring structure or a ring topology gives a starting point and also easy to analyze the algorithm and design the algorithm in this particular setting that we are going to see that is the leader election algorithm.

This abstraction of this particular ring is a token ring also. So, and the lower bounds and impossibility results for the ring topology also applies to an arbitrary topologies. So, whatever algorithms we are a design in this particular setting can also be applied as well in arbitrary topologies.

(Refer Slide Time: 07:13)



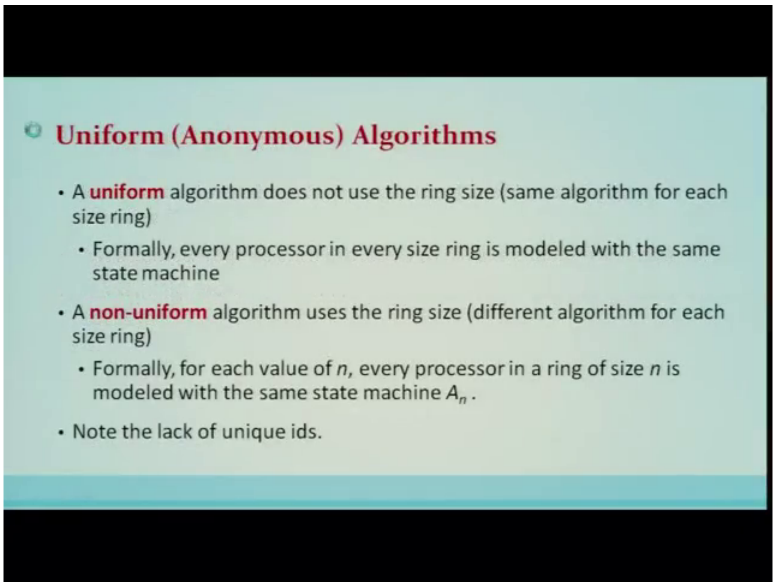
Anonymous Rings

- How to model situation when processors do not have unique identifiers?
- First attempt: require each processor to have the same state machine
- Subtle point: does algorithm rely on knowing the ring size (number of processors)?

So, the different kinds of rings we are now going to start and discuss the first type is called anonymous rings in anonymous rings the processors do not have the unique identifiers; that means, all the processors are anonymous and thus the ring form out of these processors without having any ids unique ids they are called anonymous rings.

So, in this particular setting, each processor will have the same state machine and also the algorithm relies on knowing the ring size that is if that is how many number of processors are there then.

(Refer Slide Time: 08:03)



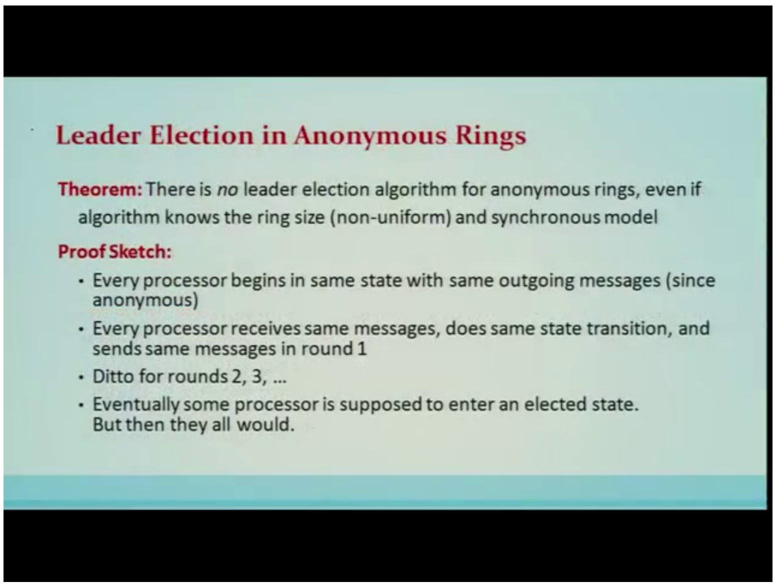
Uniform (Anonymous) Algorithms

- 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.

There is another kind of ring structure and that is called basically a uniform ring.

So, uniform ring will have n nodes, but these particular numbers of nodes are not known then it is called uniform because all the processors are not uniform the algorithm which uses this information that is without knowing the ring size called uniform algorithm. So, uniform algorithm does not use the ring size formally every processor in every size ring is modelled with the same state machine here and that is why it is called uniform algorithm or uniform ring structure a non uniform algorithm uses the size of ring in the algorithm design. So, formally each value of n every processor in the ring of size n is modelled with the same state machine that is of an so; that means, for a different ring size of value n different algorithms or different state machine will represent by A n.

(Refer Slide Time: 09:17)



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.

Now, we are going to see if the ring is anonymous that is the unique ids are not given to the processors. So, about in this particular setting about the structure that is a ring leader election algorithm, how it will work in this particular setting. So, there is a theorem which says that there is no leader election algorithm for anonymous rings even if the algorithm knows the ring size that is if it is non uniform and also, it is basically following a synchronous model.

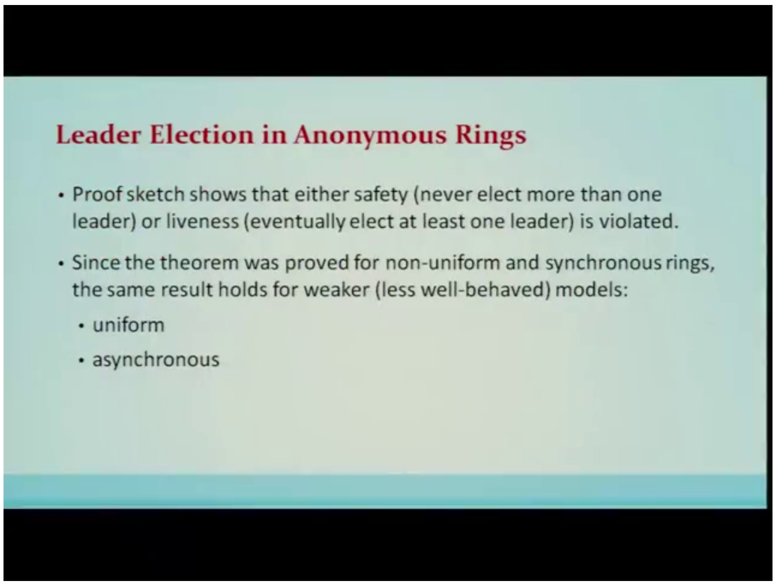
So, this result is called impossibility result; that means, no leader election is possible if the ring is anonymous that is if the IDs are not given we are going to see the proof and then this impossibility results will be used in as basically a information to develop the leader election algorithm.

So, the proof is sketch goes like this every processor now begins in the same state with the same outgoing message since they are anonymous. So, every processor receives the same message does the same state transition and sends the same message in the round one. Now you may ask if a processor is sending the message to whom it is sending how it will you know to whom, it is going to send or that who is the destination.

Then we have seen that the channel numbers that is if it is oriented ring; that means, if it is a clockwise then it will always basically send to the channel 1 and so on. So, basically the channel numbers will be used instead of if the IDs are not known and this particular structure will represent the same state machines. So, same message will be transmitted in same state transition will happen after the receipt of a message.

So, eventually some processor is supposed to enter in a elected state bun then, but then all would enter into a elected state.

(Refer Slide Time: 11:24)

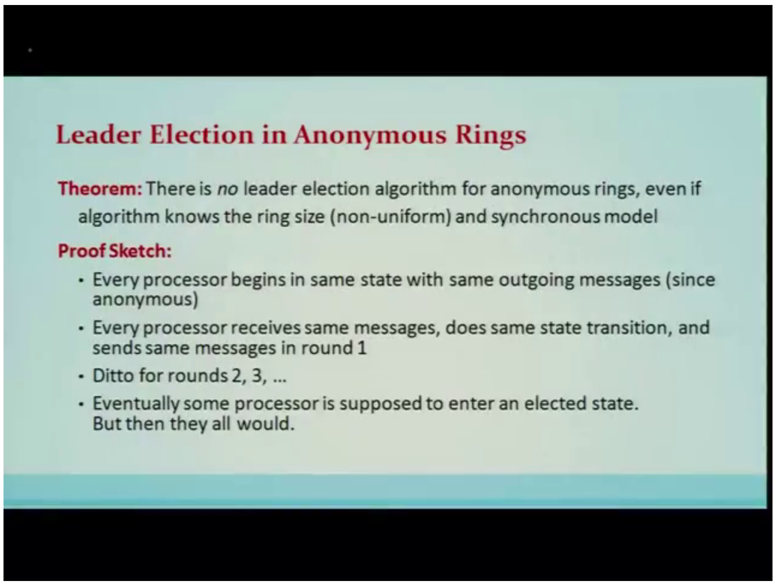


Leader Election in Anonymous Rings

- 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

So, in this particular setting if we analyze using the safety property that never elects more than one leader here every node is elected as a leader or liveness eventually elect at least one leader is also violated. So, in this particular proof we have seen that there is no leader election algorithm for anonymous rings.

(Refer Slide Time: 11:41)



Leader Election in Anonymous Rings

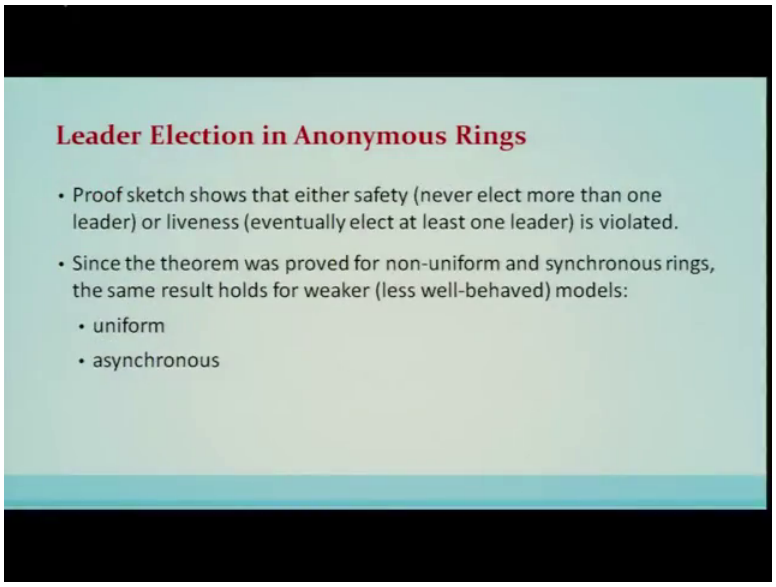
Theorem: There is *no* leader election algorithm for anonymous rings, even if algorithm knows the ring size (non-uniform) and synchronous model

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.

Now, since we have proved this theorem for since the theorem was proof for non uniform and synchronous rings.

(Refer Slide Time: 11:44)

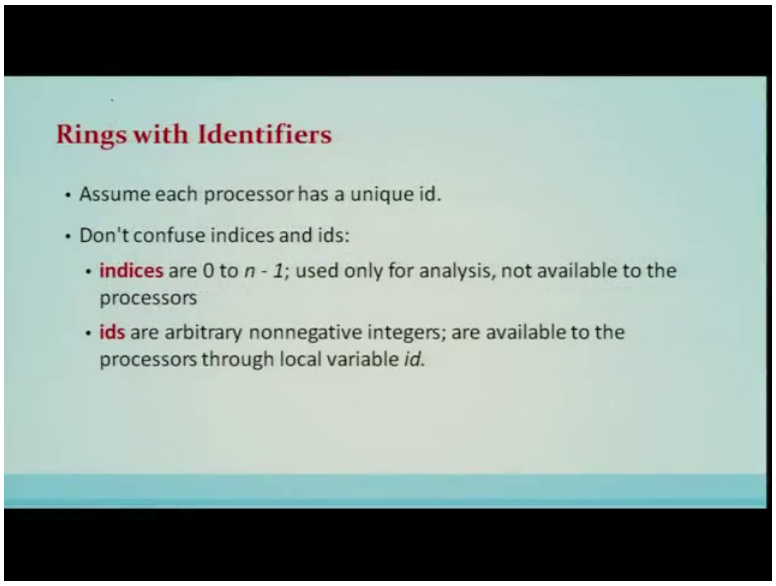


Leader Election in Anonymous Rings

- 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

And the same result will hold for weaker models weaker models in the sense; if it is uniform means the value of n is not known and also a weaker one that is called asynchronous timing model then also this particular theorem holds and leader election is not possible for anonymous rings.

(Refer Slide Time: 12:13)

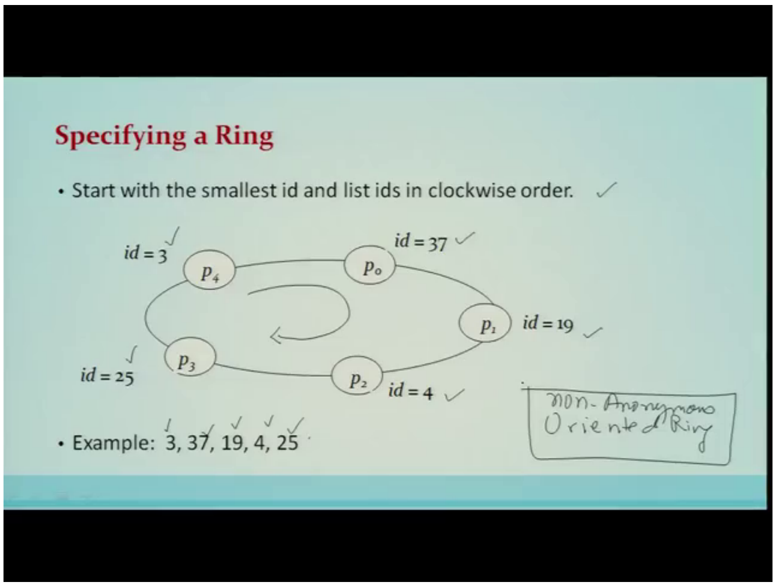


Rings with Identifiers

- 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*.

Now, we are going to discuss the rings with ids because if the leader election is not possible in basically the anonymous rings, then we are going to consider the ring structure with the unique ids. So, ids are assigned out of the natural numbers and so, each processor has a unique id now.

(Refer Slide Time: 12:45)



Specifying a Ring

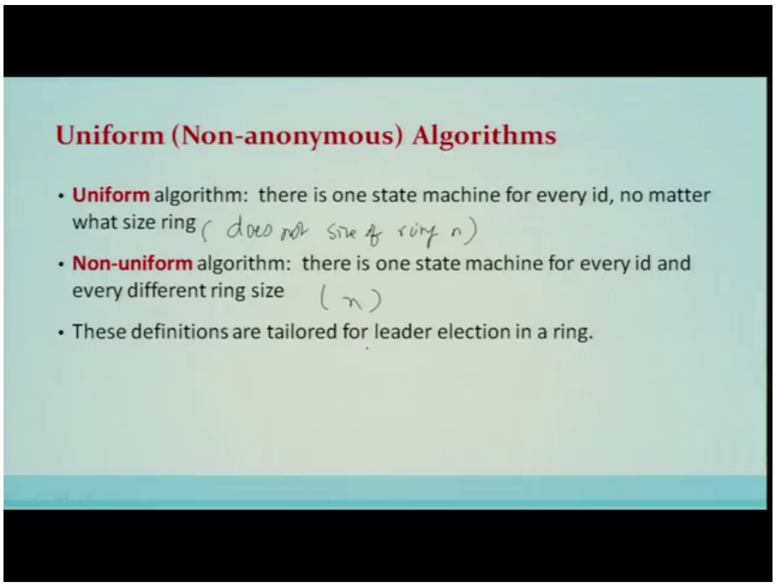
- Start with the smallest id and list ids in clockwise order. ✓

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

So, now we are going to discuss about that ids; how the ids are assigned. So, is specifying a non anonymous ring where each processor is assigned a unique id is illustrated in this particular diagram and I will be explaining you through this particular example. So, you start with the a smallest id from here smallest id is 3. So, let us note down this particular ring in this manner and list ids in a clockwise order. So, clockwise order goes like this.

So, first id is 3, then the next id is 37 and furthermore the next id is nineteen then 4 and finally, 25. So, just see that this particular structure if this rule is followed, then it will give a oriented ring and this will be a non anonymous oriented ring structure which we have seen as an example because and this particular structure we will be used to design the leader election algorithm.

(Refer Slide Time: 13:54)

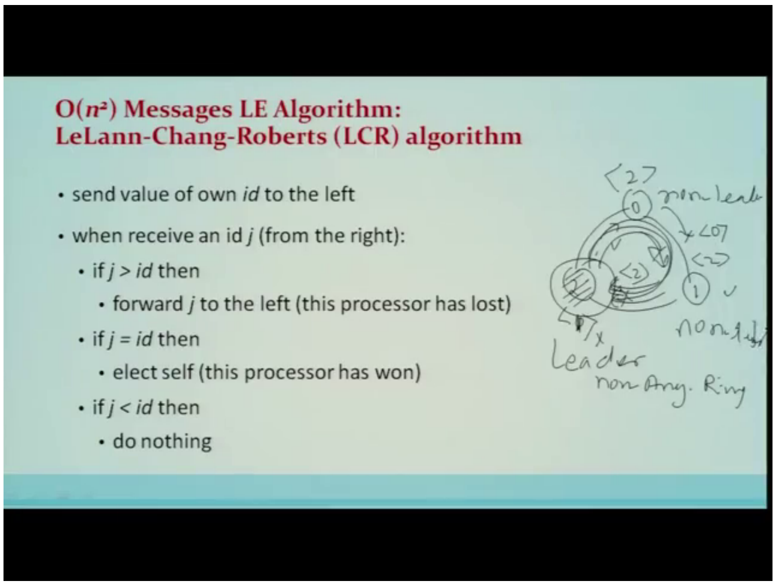


Uniform (Non-anonymous) Algorithms

- **Uniform** algorithm: there is one state machine for every id, no matter what size ring (does not size of ring n)
- **Non-uniform** algorithm: there is one state machine for every id and every different ring size (n)
- These definitions are tailored for leader election in a ring.

So, non anonymous algorithm if it is uniform; that means, there is one state machine for every ids and no matter what the size of the ring; that means, the algorithm in this particular setting does not know the size of the ring that is n is not known non anonymous it knows it knows the value of n there is one state machine for every id and every different ring size and basically, it will form an non uniform non anonymous algorithm.

(Refer Slide Time: 14:42)



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

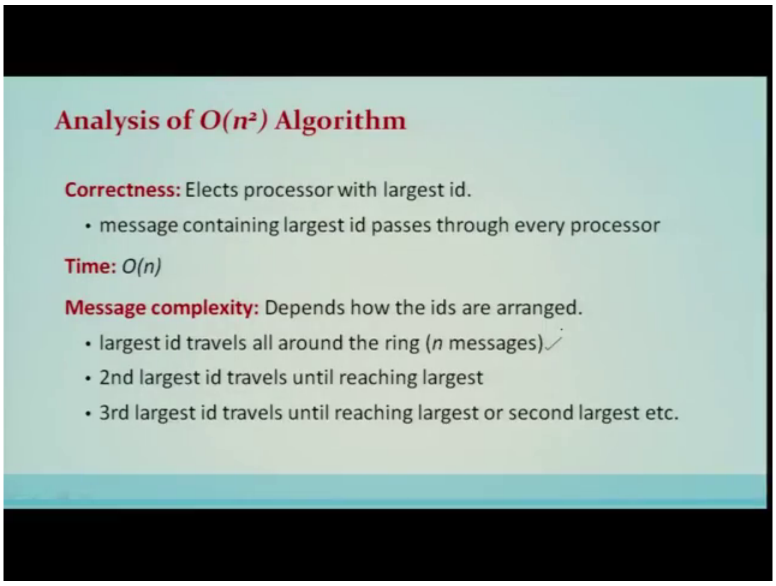
- send value of own *id* to the left
- when receive an id *j* (from the right):
  - if *j* > *id* then
    - forward *j* to the left (this processor has lost)
  - if *j* = *id* then
    - elect self (this processor has won)
  - if *j* < *id* then
    - do nothing

<2>
0
non leader
<0>
<2>
1
<2>
non leader
Leader
non Any. Ring

So, these definitions are tailored for leader election in the ring. So, now we are going to discuss the first leader election algorithm with the message complexity of the order n square and this algorithm is called LeLann Chang Roberts algorithms that is LCR algorithm. So, it is a simplest algorithm and it will give by start point to understand the leader election algorithm and design in the distributed system. So, here every node every processor will send the value of its id to the left in a form of a message. So, we can see this particular example. So, every processor will have the ids and that is unique id because it is a non anonymous ring. Now send the value of its id. So, it will send a message with its id 0 to the left.

Similarly, this also we will send and everyone will send its values now when an id j here from the right from the right it will receive the message and if the j that is the idea of incoming message is greater than its id of a receiving process, then it will forward to the left here in this case this situation is not considered this; this situation is not happening the second j is equal to id that is also not satisfying and if j is less than id here, then it will do nothing; that means, 0 will not be this particular message will be swallowed and this particular structure will go on. So, you see that in this structure here this will also be swallowed this message will be swallowed this message will be swallowed, but this will continue and this again will be continued over here. So, 2 message of size 2 will continue over here again and this will also forwarded. So, 2 will come again over here from where it is being originated.

(Refer Slide Time: 17:13)



Analysis of $O(n^2)$ Algorithm

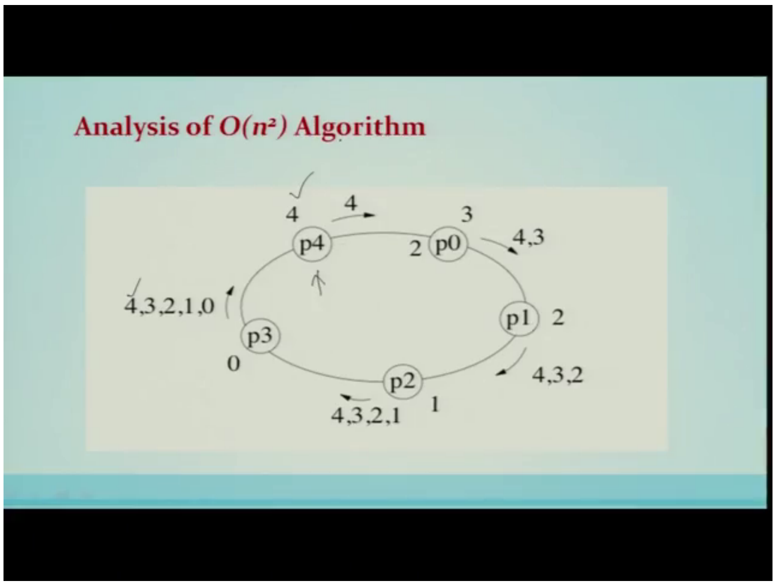
Correctness: Elects processor with largest id.
- message containing largest id passes through every processor

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.

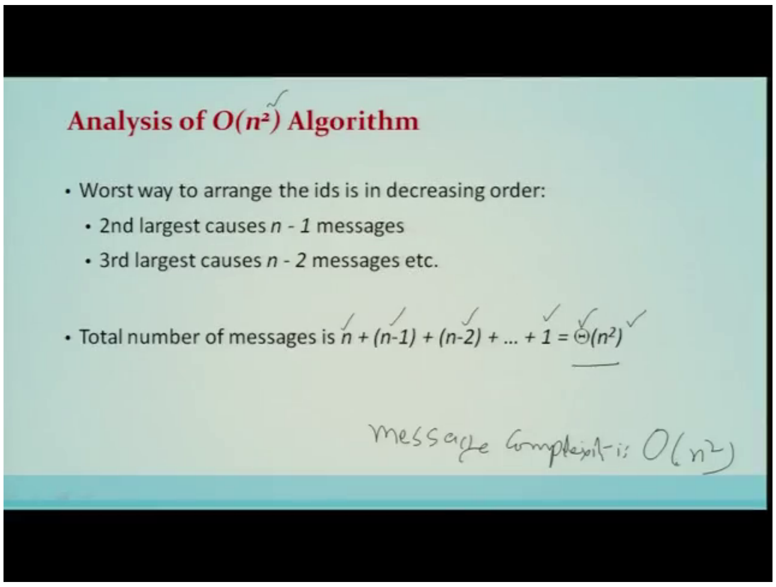
So, the correctness of the algorithm goes like this that it will always elect a processor with the largest id and the message containing largest id passes through every processor and basically comes over here again as I told you the message of a processor id with the highest id will goes along the entire ring and comes back to it and it will be elected as a leader and all other nodes then after electing a leader it will send a message that is called termination message and all other node will become a non leader . So, the time which is taking here in this algorithm is of the order n and the message complexity if will analyze it depends on how the ids are arranged. So, the largest id travels all around the ring that is it will basically result in into n different propagation of the messages second largest id will travel until reaching the largest one third largest id will travel until reaching the largest or a second largest and so on.

(Refer Slide Time: 18:31)


Analysis of $O(n^2)$ Algorithm

So, we can understand using this particular example the message complexity that is order n e square. So, just see that here the id which is that is the highest id that is the id which is number 4 will be propagated along all the ring and will come back over here again similarly the second lowest id that is 3 will propagate and it will not pass through p 4 why because it is having the highest id, it will swallow this particular message.

(Refer Slide Time: 19:12)

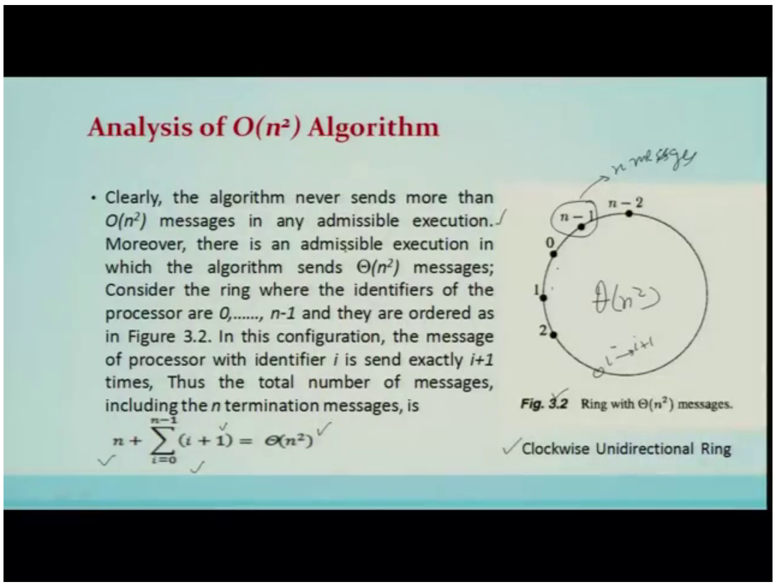

Analysis of $O(n^2)$ Algorithm

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

message complexity is $O(n^2)$

So, these particular messages are being flowing. So, if we see if we count how many messages are total number of messages is being flown you can see that the highest id will basically result into n different messages the next largest id will have n minus 1 n minus 2 n; so, on up to n. So, if we if we sum up it will become the n square. So, this particular topology will incur into n square number of messages. So, that is why the message complexity of this algorithm is of the order n square. So, we go of n square; that means, this particular topology this example of a topology, we have seen where exactly n square messages are required maybe in other topologies, it will require even less than n square, but in no situation the number of messages will exceed n square and that is why the this particular algorithm guarantees that it will take n square number of messages.

(Refer Slide Time: 20:22)



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 $0, \ldots\ldots, n-1$ 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

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

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

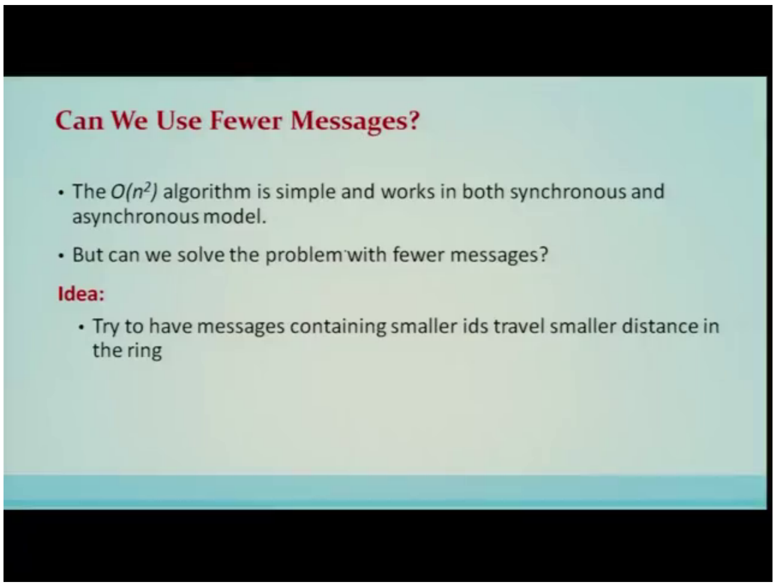
✓ Clockwise Unidirectional Ring

So, again this particular analysis we can see from this particular diagram that in any admissible execution, it will send no one more than order n square messages that is return over here more over there is an admissible execution in which the algorithm sends theta n square messages that we have seen in the previous example, here also in this example you can see consider a ring.

Where the identifiers of the processors are 0, 1, 2 and so on; they are being numbered and they are ordered in figure in this particular figure. So, in this configuration the messages of the processor with identifier i is send exactly i plus 1 times and the total number of messages including the n termination message is n plus n is the termination total number of termination message and this is the total number of messages which is send by the other processor to elect a leader and in this particular ring can see this particular structure that a processor that is an id n minus 1 will basically navigate for n different messages will propagate n different messages. So, a particular processor I will basically propagate total number of exactly identifier is propagated i plus 1 times.

So, if we sum up this particular formula again it will give theta n square and this theta n square; that means, in this particular topology, it will basically incur n square messages.

(Refer Slide Time: 22:12)



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?

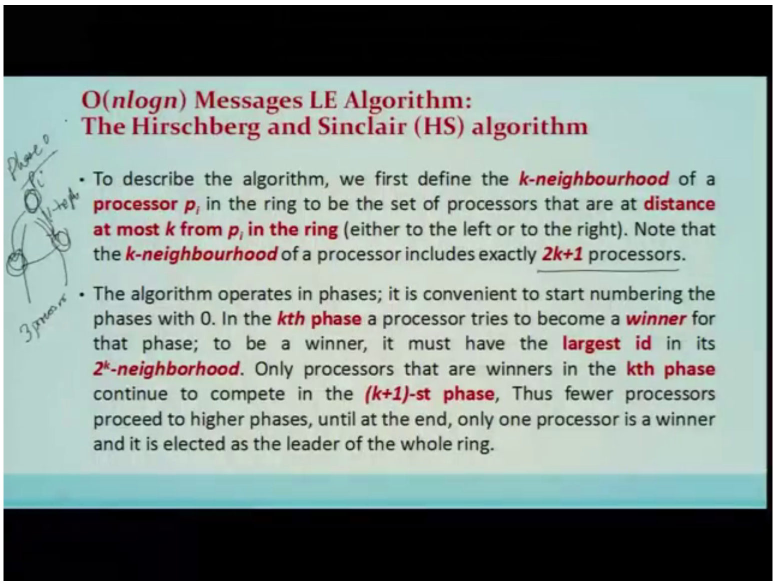
Idea:

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

So, this particular algorithm never incurs more than n square in any admissible execution now the question is can we develop an algorithm which will use fewer than these number of messages that is order n square message. So, that is an idea and we are going to see another algorithm. So, the next algorithm will be based on this idea is that why. So, many messages are used can we reduce the number of messages. So, here it is the idea is saying they try to have messages containing smaller idea travel smaller distances in the ring.

So; that means, smaller ideas are not going to elect as a leader; so, why they are travelling more distances they can be contained in a smaller region and only the ids with a higher ids or larger ids are allowed to travel across all the ring. So, this way we can see in number of messages.

(Refer Slide Time: 23:13)



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

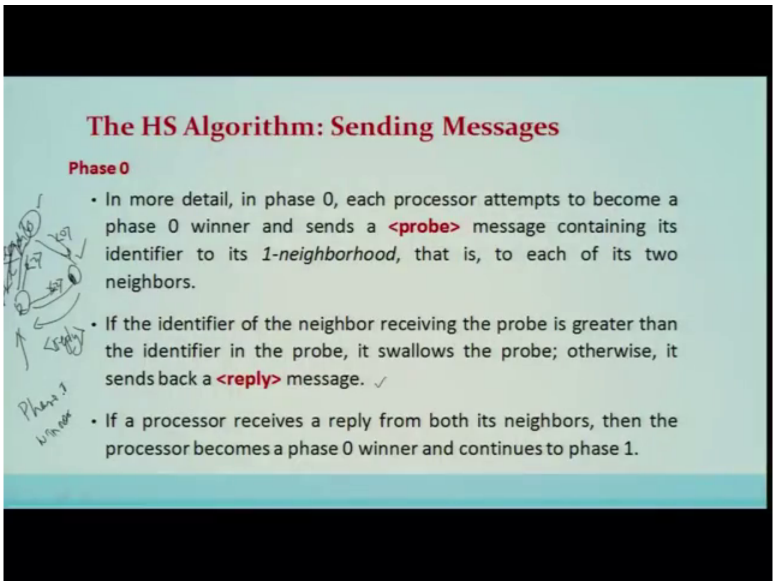
- To describe the algorithm, we first define the ***k-neighbourhood*** of a **processor p_i** in the ring to be the set of processors that are at **distance at most k from p_i in the ring** (either to the left or to the right). Note that the ***k-neighbourhood*** of a processor includes exactly ***2k+1*** 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.

In this particular algorithm and with this idea an another algorithm, we are going to see that is called big O of n log n messages that is leader election algorithm is given by Hirschberg and Sinclair HS algorithm, it is well known algorithm. So, we are not going to describe this algorithm in more details. So, to describe this algorithm we first define the k neighbourhood of the processor pi in a ring to be the set of processors that are at a distance at most k from pi in the ring that is either to the left or to the right.

Note that the k neighbourhood of a process includes exactly 2 k plus 1 processor to understand this through a diagram. So, you can see a ring of 3 nodes in this particular node the k neighbourhood of this particular processor pi one neighbourhood of this process is nothing, but the left and the right only 2 processors will be there similarly if this particularly instead of 1, if it is k neighbourhood. So, many number of k number of processors will be on the left k number of processors on the right. So, this means that a k neighbourhood of a processor includes 2 k plus 1. So, here one neighbourhood will include 3 processors. So, this algorithm that is big O of n log n messages leader election algorithm will require the knowledge of k neighbourhood of a particular processor pi for all the processors.

So, the algorithm operates in the phases. So, it is convenient to start numbering the phases with 0. So, in kth phases; so, 0 phase we have seen. So, in kth phase it is convenient to start the numbering with the phase 0 and kth phase a processor tries to become a winner of that phase to be the winner it must have the largest id in its 2 k neighbourhood only the processors that are winner in the kth phase continue to compete in the k plus oneth phase thus fewer processors exceed to a higher phase until at the end only one processor is the winner and is elected as the leader of the whole ring.

(Refer Slide Time: 25:48)



The HS Algorithm: Sending Messages

Phase 0

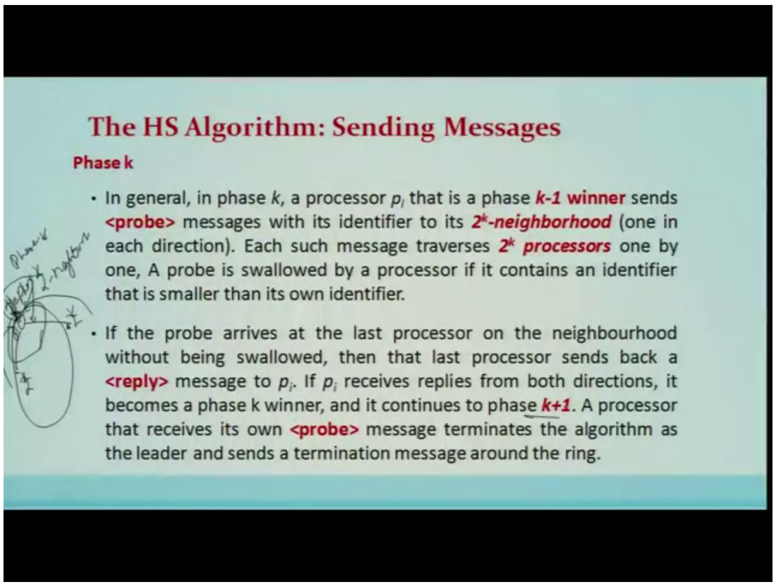
- In more detail, in phase 0, each processor attempts to become a phase 0 winner and sends a **<probe>** message containing its identifier to its *1-neighborhood*, that is, to each of its two neighbors.
- If the identifier of the neighbor receiving the probe is greater than the identifier in the probe, 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.

So, let us see more detail about phase 0 and then you will see about phase k. So, in phase 0 in more detail phase 0 each processor attempt to become a phase 0 winner take the same example.

So, in phase 0 every processor will initiate and try to become a phase 0 winner let us have these ids 0, 1 and 2. So, to become each processor attempts to become a phase 0 winner and sends a probe message containing its id to its one half neighbourhood that is to each of its 2 neighbours left and right if the identifier of the neighbour receiving the probe is greater than the id oh in the probe it swallows the probe otherwise it sends back the reply message. So, here in this case it is one and it is 2. So, the id of the processor receiving the probe message with the id 0 as having higher id, it will swallow it similarly here also it will swallow; it will not give back the reply, but here in this case if node 2 sends the probe message.

Both the message will send back the reply sends back the reply if the processor received the reply from both its neighbours, then the processor becomes the phase 0 winner. So, here processor 2 becomes phase 0 winner because it has received the replies from both of its neighbours.

(Refer Slide Time: 27:47)



The HS Algorithm: Sending Messages

Phase k

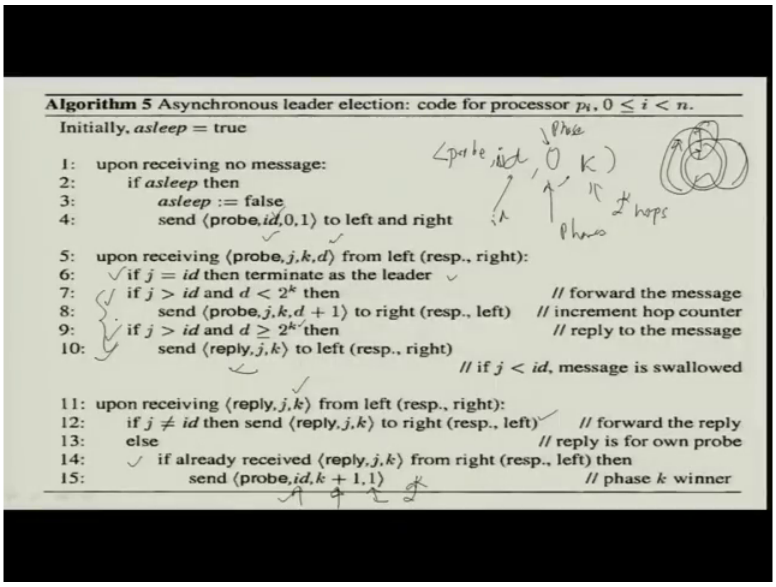
- In general, in phase *k*, a processor p_i that is a phase ***k-1* winner** sends **<probe>** messages with its identifier to its ***2^k-neighborhood*** (one in each direction). Each such message traverses ***2^k processors*** one by one, A probe is swallowed by a processor if it contains an identifier 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.

And we will continue to phase 1. So, we are going to describe about if a processor is winner in a phase k minus 1 and now it is eligible to be participating in phase k. So, phase k that is in general in phase k a processor pi that is the phase k minus 1 winner sends the probe message with its ids 2 raised power k neighbourhood that is one in each direction each such message traverses 2 raise power k processors one by one a probe is swallowed by a processor if it contains an id that is smaller than its own id that we have also seen in the phase k.

If a probe arrives at the last processor on the neighbourhood without being swallowed then that last processor sends back a reply message to pi if pi received a reply from both the direction it becomes a phase k winner and it continues to the phase k plus one. So, the processor that receives its own probe message terminates the algorithm as a leader and sends the termination message around the ring. So, if we can see if this is the ring. So, every every node will have 2 raise power k neighbourhood for a phase k. So, 2 raise power k neighbourhood on the left 2 raise power k neighbours on the on the right if that particular message goes through the last node in each of these 2 power k neighbourhood on the left and right; if it is basically going through up to the last one.

Then it will send back the replies and it will the winner of k phase of the algorithm it will be eligible to proceed to the k plus oneth phase.

(Refer Slide Time: 29:53)



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

```
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 $d \geq 2^k$ 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 $j \neq id$ 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
```

So, that I have explained you the phase k of this particular algorithm the entire algorithm is explained over here. So, here you can see that the probe structure we will have 3 different probe will have 3 different information the first information is called id the id of the node which has initiated on a particular phase and in which phase it is initiating.

So, that phase number is given and the hop count because it has to travel through to 2 raised power k neighbourhood. So, here it is let us say 2 raised power k hops how many hops and this is the phase number which phase message it is it belongs to and the ids which particular. So, in this particular send message you can see that the probe has one 2 3 different arguments 3 arguments means id of that particular processor.

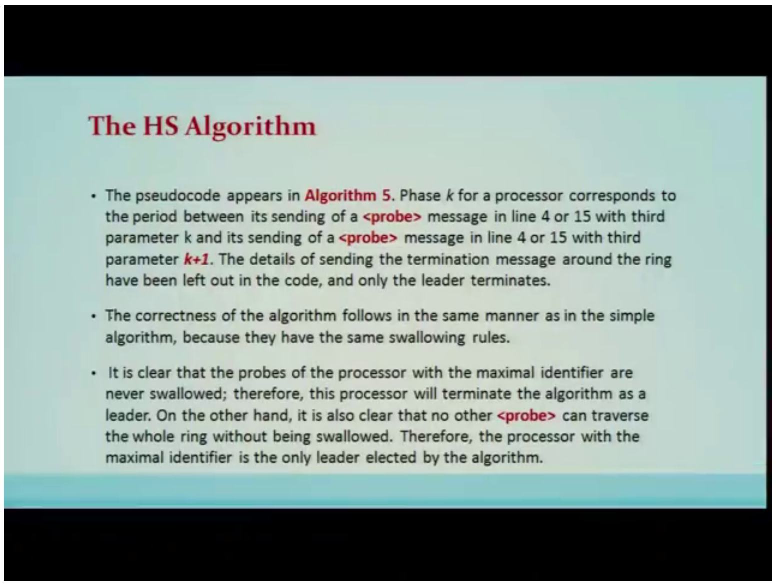
Which has initiated that particular leader election in phase 0 and it is going to send to one hop neighbourhood to the left and right that we have seen now upon receiving this particular probe with jk and d j means the idea of the probe message k means the kth phase and d means the number of hops it has traverse so far. So, upon receiving this probe from the left and it will also receive from the right it will perform 3 different cases now if j is equal to id; that means, the message which has traverse along all along all the nodes or a processors of the ring and came back to the same point.

Then that mode will be terminated and elected as the leader. Now if j that is the receiving message his id or his id is greater than the node which is receiving this particular message and also the number of hops is less than 2 raised power k then it will send the probe to the to the to the left and to the right as well depending upon from where the message it is going to received. Now if the number of hops is reached to 2 raised power k on the left side or on the right side that will send the replies message; so, it will send the reply in that case because it has reached 2 power k hop neighbours, then it will send the reply message. So, it has to decide based on whether it has reached 2 power k the last node of 2 power k neighbourhood.

If it has then basically it will send a reply otherwise it will keep on forwarding the probe. So, all 3 cases are given now upon receiving the reply from the left and the right if j is not equal to id then it will send the reply to the right; that means, it will keep on forwarding the replies if it is not equal to id and if it is id then the reply will be will be reached if already receive already received reply from j from the right and from the left then it will be the winner of phase k and it will start the probe for k plus oneth. So, the structure will be the same node which is the winner of kth phase will go and send the probe at k plus oneth phase or initiate the k plus oneth phase and basically the number of hops, it will basically begin and this hop count is initialize to or is being sent to 1 and keeps on incrementing till it reaches 2 raised power k and there.

These 2 conditions are being checked whether the probe is to be move forwarded or it is to be returned back to the originator. So, this algorithm is explained which is called a leader in O big O of n log n leader election algorithm in an in a ring.

(Refer Slide Time: 34:32)



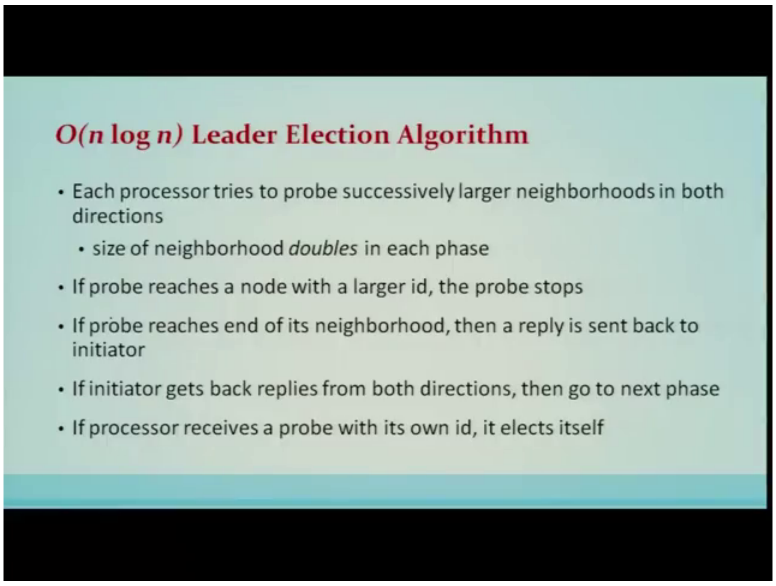
The HS Algorithm

- 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.

So, the pseudo-code which appears in the algorithm 5 we have just seen that algorithm 5 the phase k for a processor corresponds to the period between its ending of a probe message in line 4 or a line 15 that I have explained with the third parameter k and it sending of a probe message in line 4 or 15 with the third parameter k plus one 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 that is over n square algorithm because they have the same swallowing rules.

It is clear that the probes of the processor with a maximal ids are never swallowed therefore, this processor will terminate the algorithm as the 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 this particular algorithm.

(Refer Slide Time: 35:48)

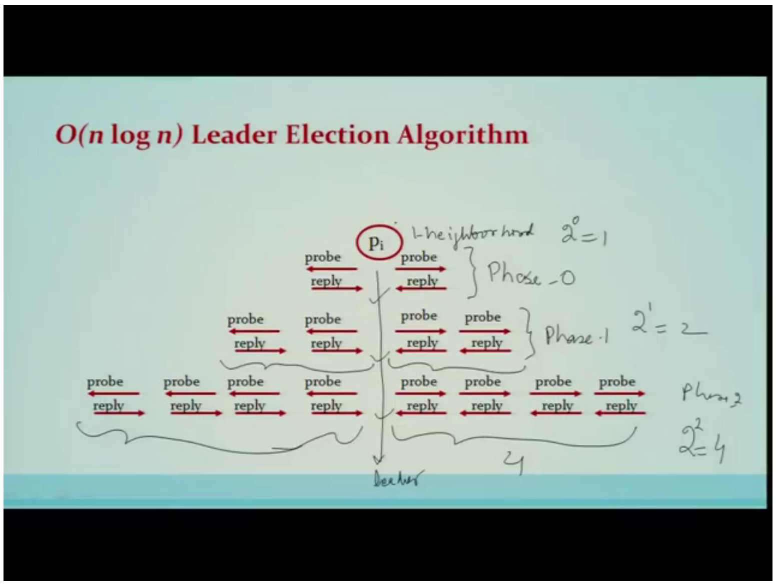


O(n log n) Leader Election Algorithm

- 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

So, again more description about this algorithm; so, each processor now here in this algorithm tries to probe successively larger neighbourhoods in both the directions that is the size of the neighbourhood doubles in each phase. Now if the probe reaches the node with the largest id the probe starts, if the stops if the probe reaches end of the neighbourhood, then the reply is sent back to the initiator that have I explained you in the algorithm.

(Refer Slide Time: 36:29)

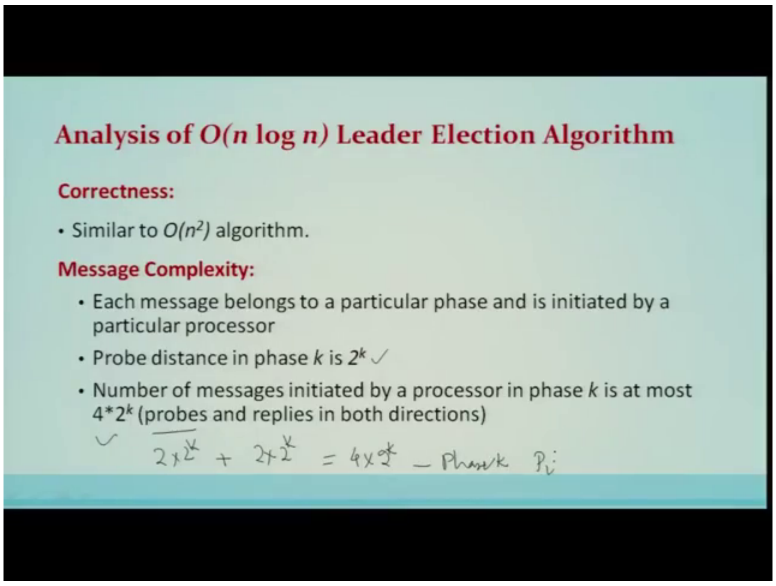


O(n log n) Leader Election Algorithm

If the initiator gets back the replies from both direction, then it goes to the next phase that also we explained in the algorithm if the processor receives a probe with its own id then it will elect itself a leader the same thing is explained here in this particular picture. So, this is now you can very easily identify this is phase 0; phase 0 means this is the one neighbourhood one hop neighbourhood because 2 raised power 0 is 1 this is the phase 1. So, here 2 raised power one that is 2 so; that means, here this particular neighbourhood; that means, it is basically 2 hop neighbourhood in both the directions in the phase 1. Similarly this is the phase 2 in the phase 2 ra 2 raised power 2 is equal to 4.

So, here you just see that this is 4 neighbours on the left 4 neighbours on the right and the probe will go the 4 different hops and the replies has to come back then only pi has to be winner if the pi has to be the leader then it has to be elected as a leader in phase 0 then it has to be elected in phase 1 it has to be elected in a in a phase 2. So, if it is a ring of this size then pi will be the leader elected in this particular algorithm; this is explained over here.

(Refer Slide Time: 38:05)



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

Correctness:

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

Message Complexity:

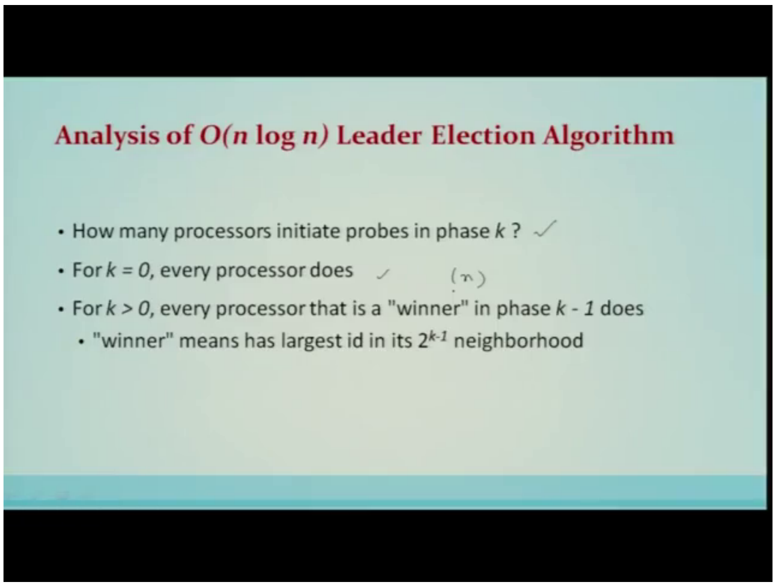
- Each message belongs to a particular phase and is initiated by a particular processor
- Probe distance in phase k is 2^k ✓
- Number of messages initiated by a processor in phase k is at most $4*2^k$ (probes and replies in both directions)

✓ $\overline{2\times 2^k} + 2\times 2^k = 4\times 2^k$ — Phase k P_i

Now, this correctness is similar to big O of n square that I have explained you now message complexity of this algorithm the message complexity of this algorithm belongs to a particular phase and is initiated by a particular processor. So, now, we are going to count how many different messages are being used to elect a leader here in this case and how it becomes order of n log n.

Now, the probe distance in a particular phase k is 2 raised to power k that you know already. So, the number of messages initiated by a processor in phase k is at most 4 times 2 raised power k why because 2 into 2 raised power k probes messages and 2 into 2 raised power k replies messages if you count it becomes 4 times 2 raised power k different messages in a phase k which is being initiated by a processor pi. So, we have to count from any such processors will be there in a phase k.

(Refer Slide Time: 39:15)



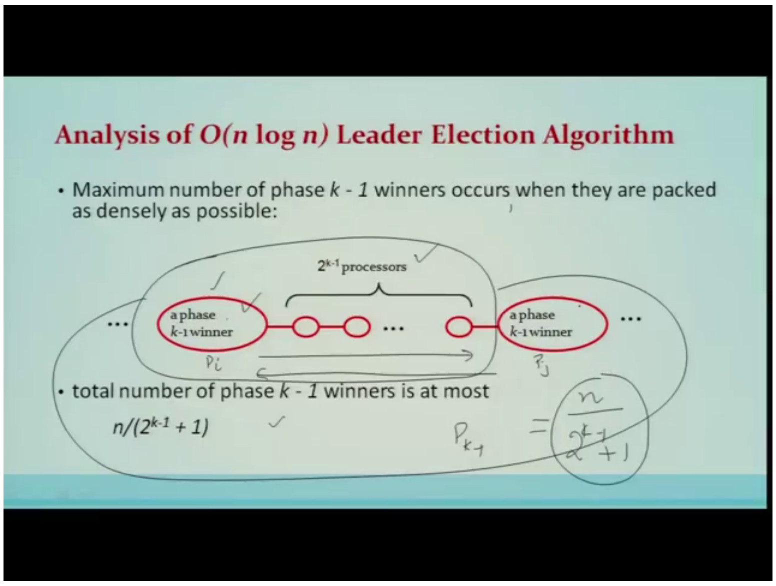
Analysis of $O(n \log n)$ Leader Election Algorithm

- How many processors initiate probes in phase k ? ✓
- For $k = 0$, every processor does ✓ (n)
- 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

So, how many processors will be initiating in a particular phase k that we are going to compute now for phase for k is equal to 0 that is for phase 0 every processor will basically initiate. So, it becomes n; n different processors are there now if the phase is not one is more than or more than 0, then every processor that is the winner in a phase k minus 1.

Does the winner means the largest id in its 2 k minus 1 neighbourhood.

(Refer Slide Time: 39:59)



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:

2^{k-1} processors
a phase k-1 winner
a phase k-1 winner
P_i
P_j

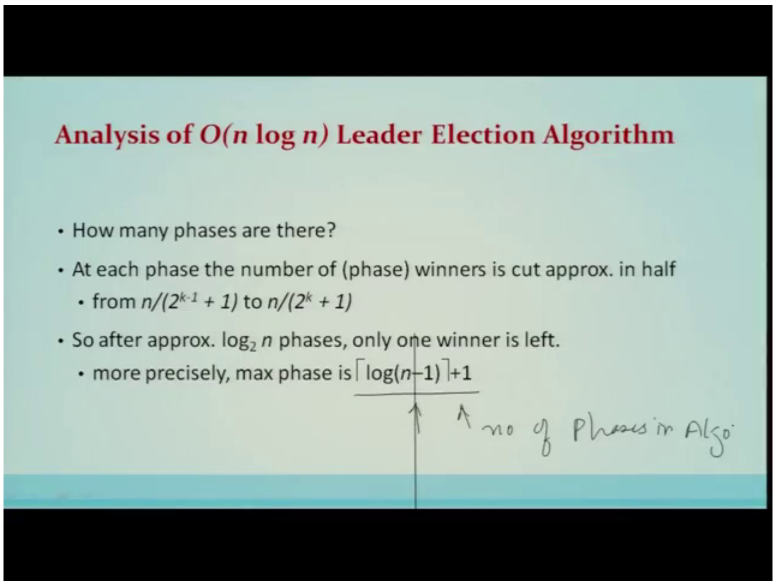
- total number of phase $k - 1$ winners is at most $n/(2^{k-1} + 1)$

$P_{k-1} = \frac{n}{2^{k-1}+1}$

So, that we are going to comput. So, how many num maximum number of phase k minus 1 winner occurs when they are occurs. So, we can see this when these particular phase k minus 1 winners are basically packed densely in this particular manner. So, what is basically? So, if the phase k winner between 2 phase k minus 1 winner let us say pi on the left it has 2 k minus 1 processors. So, another processor pj on it is right, again, it will have 2 k minus 1 processors. So, so if this kind of packing is done all around the ring structure.

So, then we can count how many different phase k minus 1 winner will be there. So, that particular count if we see the total number of phase k n minus 1 winner will be at most n divided by 2 raised power k minus 1 plus 1 plus 1 means including this and 2 k plus one this. So, total number of winners in a phase k minus 1 will be this particular figure that is explained over here using this formula.

(Refer Slide Time: 41:27)



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

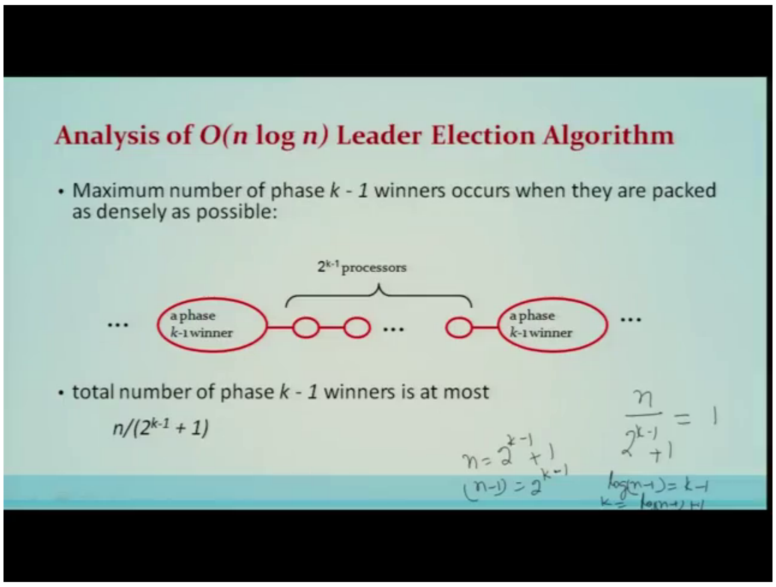
- How many phases are there?
- At each phase the number of (phase) winners is cut approx. in half
  - from $n/(2^{k-1} + 1)$ to $n/(2^k + 1)$
- So after approx. $\log_2 n$ phases, only one winner is left.
  - more precisely, max phase is $\lceil \log(n-1) \rceil + 1$

no of Phases in Algo

Now, next thing is what happened just pause start. So, how many phases are there that we are going to now find out now. So, at each phase the number of phase winners is cut approximately in half from n divided by 2 k minus 1 plus 1 2 n divided by 2 k plus 1.

So, after the approximating log n phases only one winner is left out that is precisely max phases log n minus 1 plus one. So, if you want to understand; this particular how many different phases will be there. So, you can see that how many. So, it will be if we go back. So, total number of phase k minus 1 winner is at most n raised power.

(Refer Slide Time: 42:53)



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

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

$$\frac{n}{2^{k-1}+1} = 1$$

$$n = 2^{k-1} + 1$$

$$(n-1) = 2^{k-1}$$

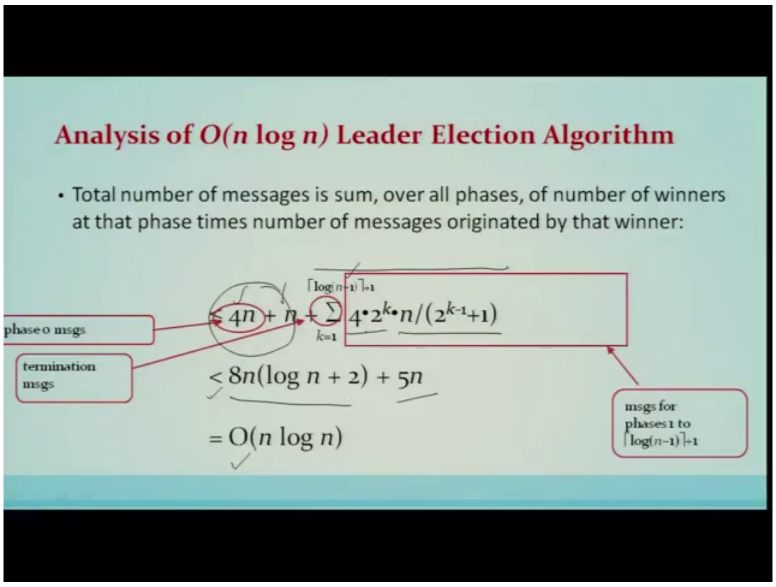
$$\log(n-1) = k-1$$

$$k = \log(n-1) + 1$$

So, total number of phase k minus 1 winners is at most n raised power of 2 k minus 1 plus one. So, these are total number of phase k minus 1 winners. So, in the end there will be only one winner. So, this will become n is equal to 2 to the power k minus 1 plus one when total number of winners is only one and if we move this one over here it will become n minus 1 is equal to 2 raised power k minus 1. So, if you take log out of it both the side.

So, that becomes k is equal to total number of phases will be will be log n minus 1 plus one. So, this particular expression we have explained you how this particular expression came and this will represent how many number of phases the total number of phases in the algorithm; we have computed.

(Refer Slide Time: 44:16)



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:

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

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

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

phase 0 msgs

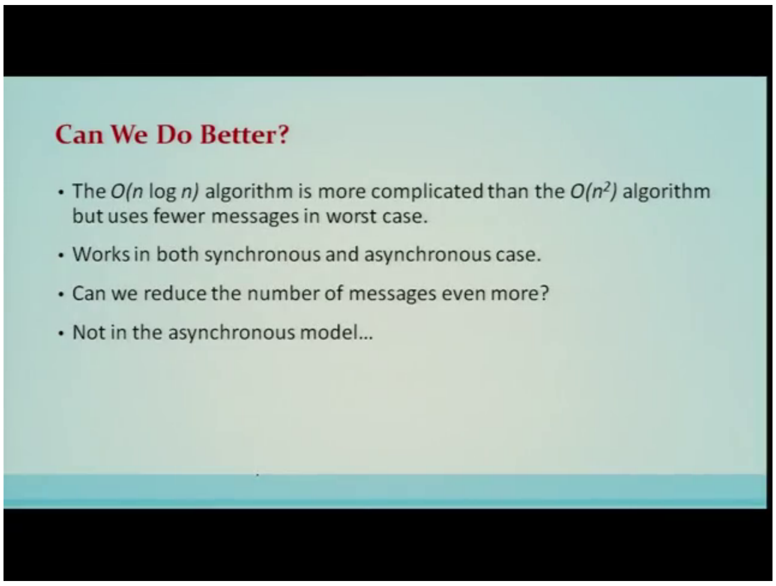
termination msgs

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

Now, we are going to count how many messages are required in this particular algorithm; how many total number of messages are flown to decide a leader in this particular algorithm. So, we can see here 4 n messages will be required in the phase 0 then when the leader is decided then it will send a termination message that will be n different messages in all other cases. So, it will be the summation like this. So, 4 into 2 raised power k different messages will be there that will be a probe and reply message and it is bidirectional.

So, basically 4 times 2 raised power k that we have seen in the previous formula also and these are basically the number of phase 2 k minus 1 winners and total number of phases will be the summation one 2 log n minus 1 that we have seen. So, if we count total number of messages. So, that is what I have explained total number of messages it comes out to be this figure is 5 n and this can be approximated as 8 n log n plus 2 which will be of the order n log n.

(Refer Slide Time: 45:52)



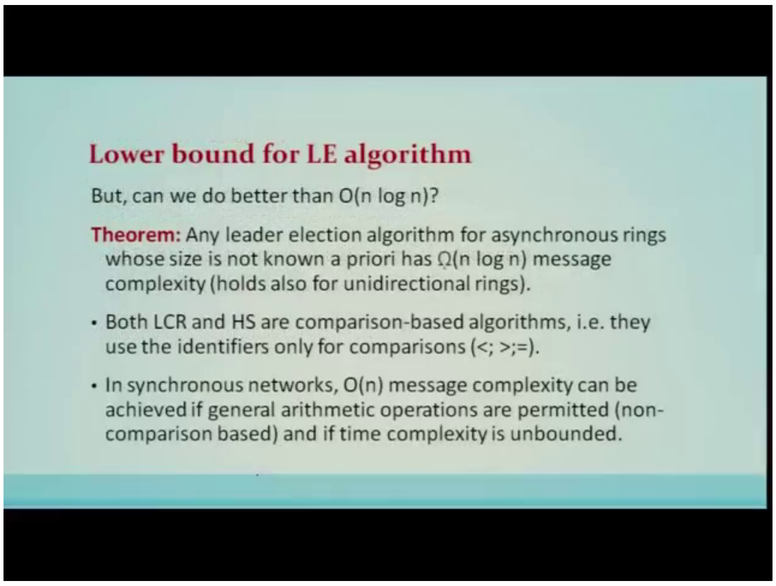
Can We Do Better?

- 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...

Now, the order n log n algorithm is more complicated we have seen than order n square algorithm why because; but it uses a fewer messages in the worst case.

So, it works both in the synchronous and asynchronous case. So, can we reduce the number of messages even further than n log n? So, we can see here that it is not possible why because there is a lower bound which is proved as n log n in this particular model which is called asynchronous model. So, in asynchronous model this algorithm is optimal because n log n is the lower bound which is proved here in asynchronous model.

(Refer Slide Time: 46:36)



Lower bound for LE algorithm

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

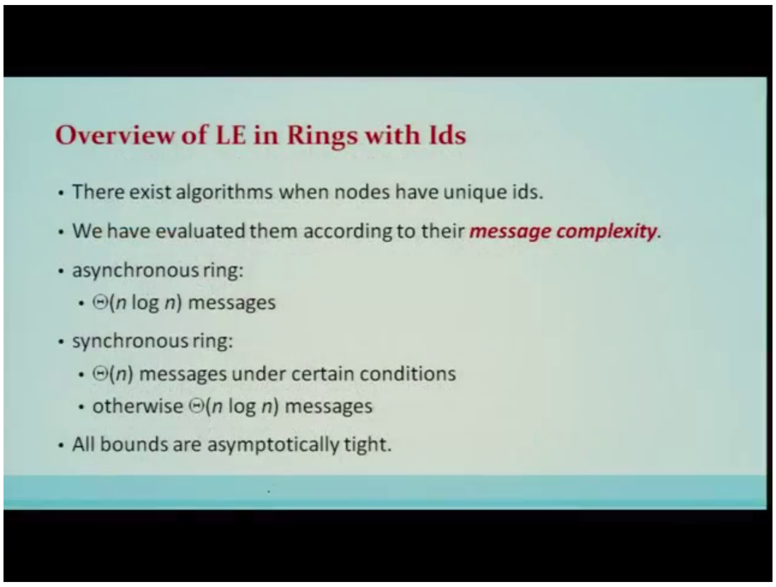
Theorem: Any leader election algorithm for asynchronous rings whose size is not known a priori has $\Omega(n \log n)$ message complexity (holds also for unidirectional rings).

- 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.

So, it is the theorem says that any leader election algorithm for asynchronous rings whose size is not known a priori has the lower bound omega big omega n log n message complexity holds also for the unidirectional rings. So, both LCR; LCR means big O of n square algorithm and HS; HS means oh big O of n log n are compared are companion based algorithm that they use identifies only for comparisons.

In synchronous networks big O of n message complexity can be achieved if general automatic operations are permitted and if the time complexity is unbounded.

(Refer Slide Time: 47:20)



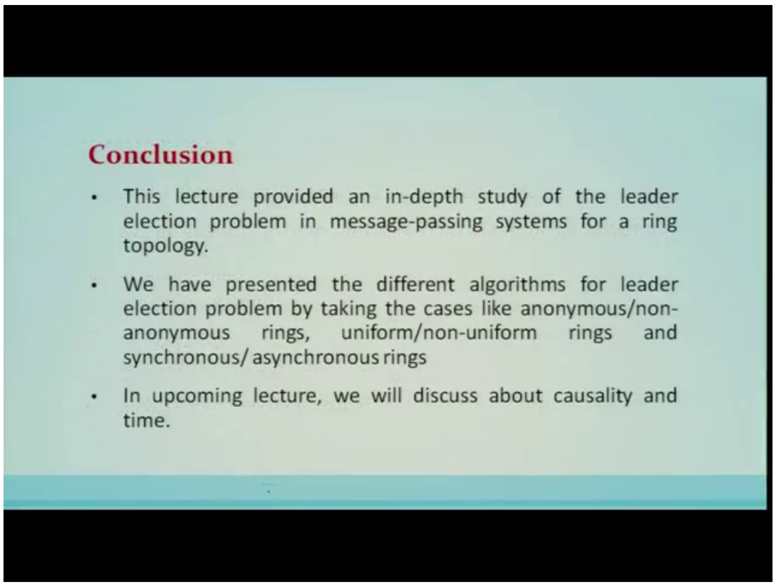
Overview of LE in Rings with Ids

- 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.

So, to summarize of the leader election in the message passing system which is the ring with the non anonymous ring having the distinct ids assigned the we can see the complete scenario as an overview like this that there exist algorithm and the nodes have unique ids that we have seen after seeing the impossibility results then we have evaluated them according to their message complexities that is in the; if it is a synchronous ring that we will take n log n messages if it is synchronous ring.

Then; it will take order n theta n messages under certain conditions otherwise it will be theta n log n messages all bounds are asymptotically tight.

(Refer Slide Time: 48:13)



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
- In upcoming lecture, we will discuss about causality and time.

So, the conclusion; so, this particular lecture provided in depth study of the leader election problem in the message passing system for a ring topology we have presented different algorithm for leader election problem by taking there are different cases like anonymous non anonymous rings uniform and non uniform ring synchronous and asynchronous ring. So, in the upcoming lectures we will discuss about causality and the time concept in the distributed system why because we have already seen that distributed system is not having a common global clock yet how the events are to be ordered for that; we are going to see in the next lecture more details about it.

Thank you.