DD2451
Parallel and Distributed Computing

FDD3008
Distributed Algorithms

Lecture 8

Leader Election

Mads Dam

Autumn/Winter 2011

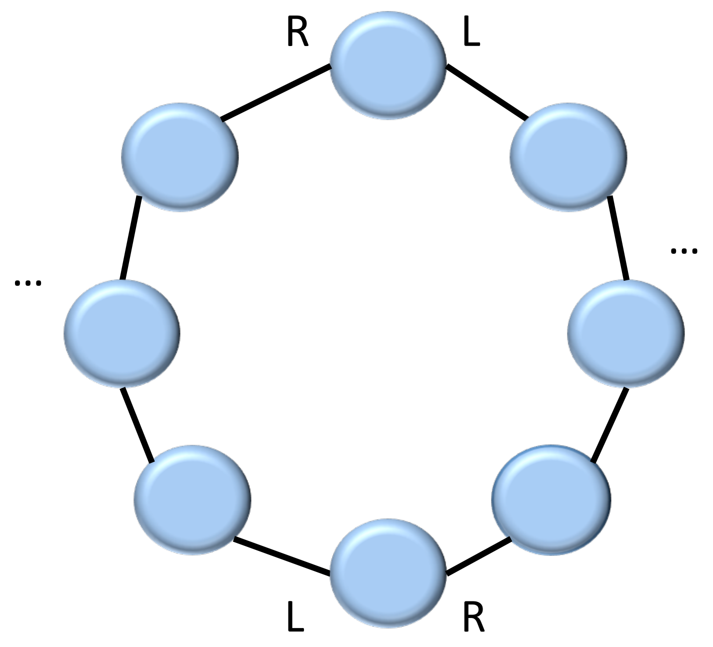
Previously . . .

- Consensus for message passing concurrency
- Crash failures, byzantine failures
- Lower bounds and upper bounds
- Algorithms using authentication and randomization

- Today:
- How to elect a leader in a ring of processes
- Another version of agreement
- Sometimes useful
- Some interesting techniques
- Key theme: Symmetry breaking

Process Rings

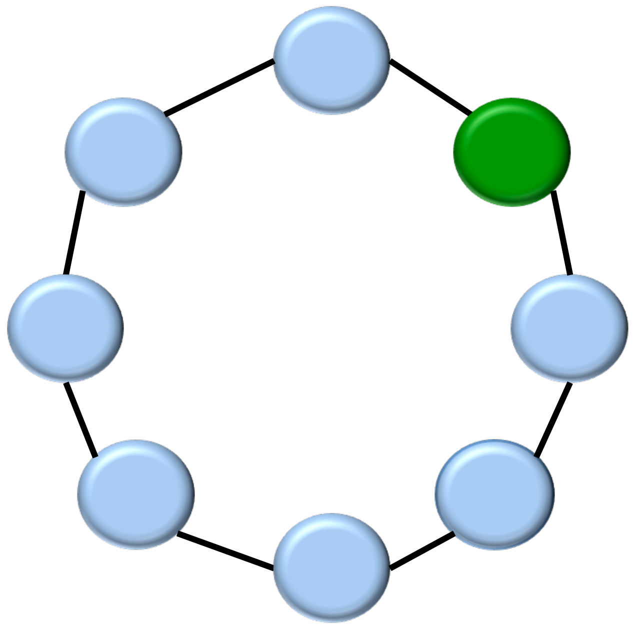
- Process organized in ring
- Each process has left and right neighbour
- Orientation is consistent for all processes in ring



Reading: Attiya, Welch ch 3, Wattenhofer ch 2 in the course page

Task

- Each process in one of three states
  - {undecided, leader, not leader}
- Initially, each process is undecided
- On termination:
  - Each process is in either leader or not-leader state
  - There must be exactly one leader



Anonymous Rings

Node identifiers can be used to break symmetries

- So we initially assume anonymity
- Ring is *anonymous,* if nodes do not have unique identifiers
  - Each node in a given ring executes the same control program
- Ring is *uniform*, if the size of the ring is not known in advance
  - There is a single control program executed by all nodes, for any ring size

Impossibility of Leader Election

Theorem 1

Leader election on an anonymous ring using a deterministic control program on each node is impossible

Lemma 1

After *k* rounds of any deterministic algorithm on an anonymous synchronous ring, each node is in the same state s_k

Proof: All nodes start in the same state

One round consists of sending, receiving, and performing local computation. Left and right are treated symmetrically by all nodes. So they send and receive same messages, perform same computation, ends up in same state.

Impossibility of Leader Election

Theorem 1

Leader election on an anonymous ring using a deterministic control program on each node is impossible

Proof:

If one nodes decides to become leader, all nodes do so, by lemma 1.

Since this holds for non-uniform, synchronous systems it holds for uniform, alt. asynchronous systems as well

Sense of direction does not help

Asynchronous Rings

Drop now the anonymity assumption

```
Code for node v:

 upon waking up {
    Send own identifier to the left neighbour}

 upon receiving a message w {
    if w > own identifier {
       forward w to the left neighbour ;
       decide not to be leader ; }
    else {if w = own identifier {
             decide to be leader} } }
```

Left or right? Ok to just send a message to the “other” neighbour

Correctness

- Let *v* be node with max identifier
- *v* sends it identifier to the left
- All other nodes will eventually receive *v* from right, decide to be non-leader, and forward *v* to the left
- Eventually *v* will receive own identifier from right and decide to be leader

Time and Message Complexity

How to measure time complexity in asynchronous networks?

Def. *Time complexity*: Number of time units taken from start of execution to completion in the worst case (for any legal input and any execution scenario), assuming a message delay of 1.

Above applies to asynchronous algorithms only

Def. *Message complexity*: Number of messages exchanged, for any legal input, for any execution scenario

Applies to both synchronous and asynchronous algorithms

Complexity of Alg. 1

Message complexity is $O(n^2)$

- n nodes, n identifiers
- Each node forwards at most n messages

Time complexity of alg. 1 is $O(n)$:

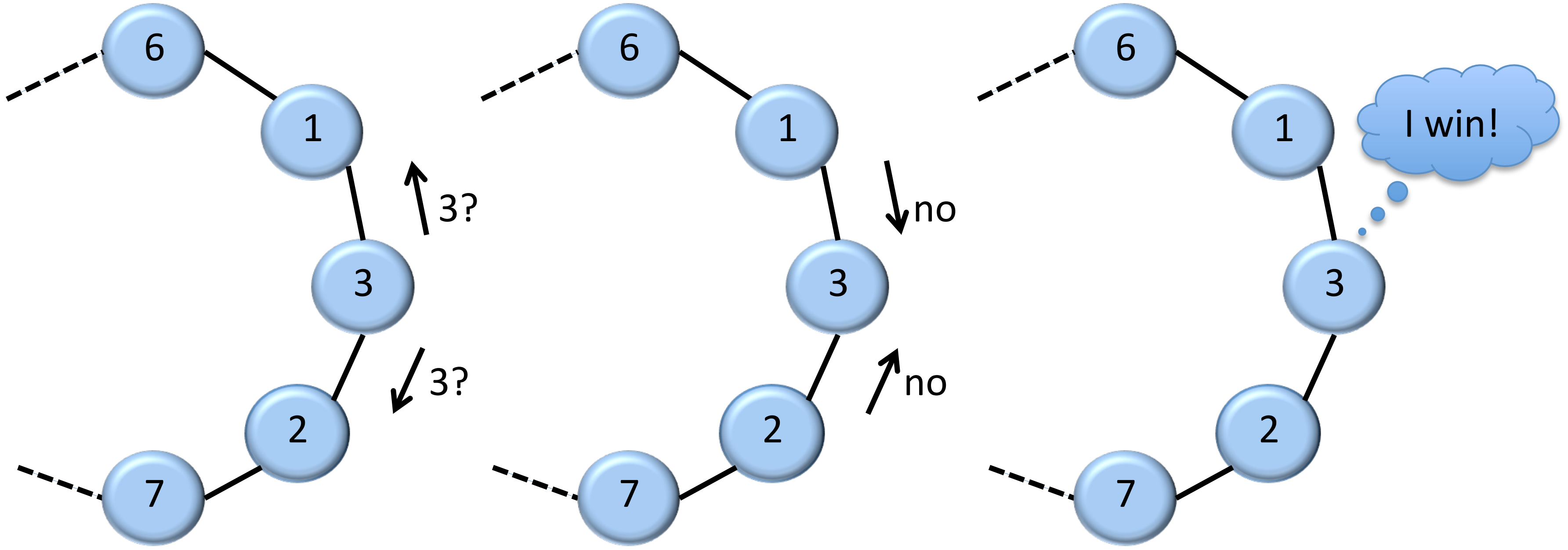
- Start when first node starts sending
- After at most $n - 1$ time units the leader-to-be v is reached
- Routing v:s identifier around the ring takes n steps
- After $2n - 1$ steps all told all nodes are decided

Can we do better?

Radius Growth

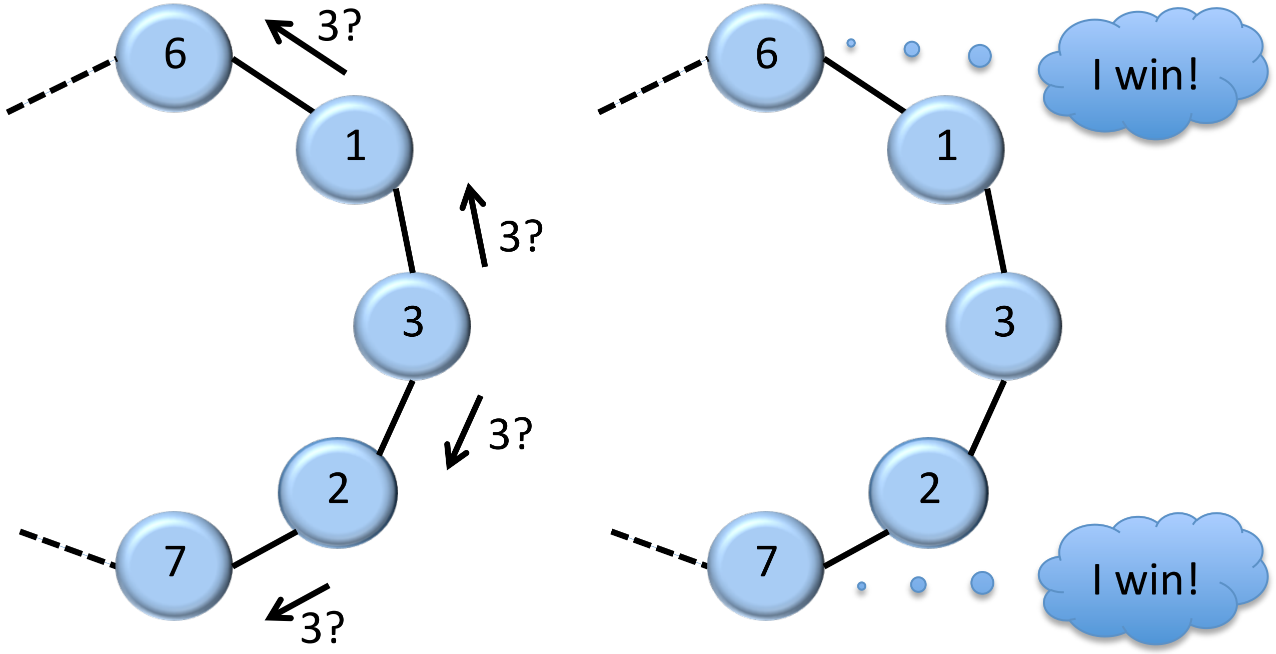
Idea:

- Find a winner node for each *k*-neighbourhood in the ring
- Let *k* grow exponentially
- When the *k*-neighbourhood is the entire ring, winner is leader
- Winner sends termination message around the ring



Radius Growth

When the probe (3?) hits a node with larger index the probe is swallowed



Radius Growth

Algorithm 2

```
Code for node v with identifier id:

upon waking up send probe(id,0,1) left and right

upon receiving probe(j,k,d) from left (right)
  if j = id terminate as leader
  if j > id and d < $2^k$ {
         send probe(j,k,d+1) to right (left)}
  if j > id and d >= $2^k$ {
         send reply(j,k) to left (right)}

upon receiving reply(j,k) from left (right)
  if j != id {send reply(j,k) to right (left)}
  else {
   if already received reply(j,k) from right (left)
         send probe(id,k+1,1) left and right }
```

Correctness

- Probe of node with maximal index will never be swallowed
- Receiving own probe causes max index node to terminate as leader
- Probes of other nodes will be swallowed
- Once max index node has determined it is leader it sends termination message around the ring

Complexity

Time complexity is $O(n)$:

- The leader-to-be sends messages around the ring with round trip times 2, 4, 8, … , $2 \cdot 2^k$ with $k <= \log(n + 1)$
- The termination costs n sequential message transmissions

NB “Rounds” does not mean the algorithm is synchronous, it is not

Complexity

Message complexity is O(n log n):

- If node survives round r then no other node in 2^r neighbourhood survives round r
- So, less than $n/2^r$ nodes are active in round $r + 1$
- Being active in round r costs $2 \cdot 2 \cdot 2^r$ messages
  - 2^r probe messages
  - 2^r reply messages
  - Going left and right along the ring
- So round r costs at most $2 \cdot 2 \cdot 2^r \cdot n/(2^r\text{-}1) = 8n$ messages
- There are $O(\log n)$ rounds

NB "Rounds" does not mean the algorithm is synchronous, it is not

Lower Bounds

Goal: Prove that algorithm 2 is asymptotically optimal

Rules of the game:

- Recall execution model of last lecture
- Nodes wake up at the latest when receiving first message
- Algorithms must be uniform
- We assume we have an algorithm and show it cannot complete faster than O(*n* log *n*) time
- It needs to do this *regardless* how messages are scheduled
  - And when nodes wake up
  - Otherwise it is not a solution
- But communication links must be FIFO

Open Schedules

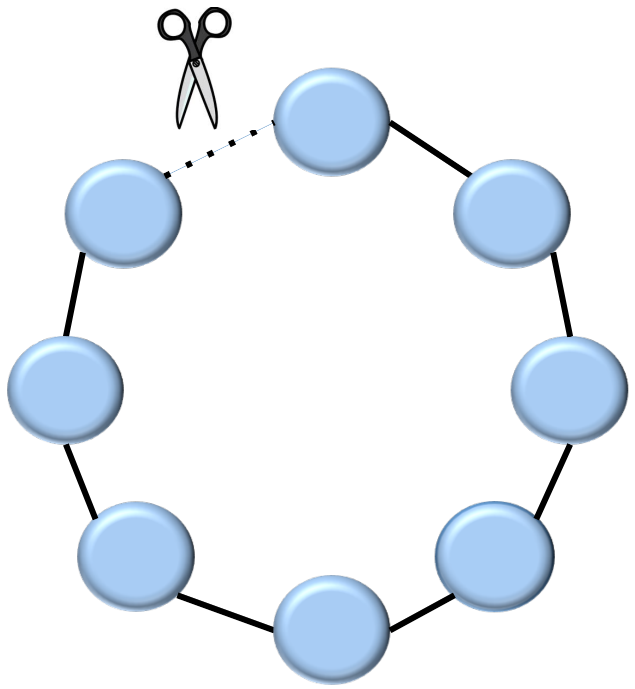
Schedule: Execution chosen by the scheduler

Open schedule:

- Schedule with an open edge / communication link

Open edge:

- Edge along which no message has yet been scheduled



Base Case

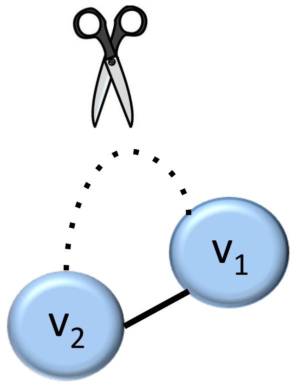
Lemma 2

Given a ring with two nodes we can construct an open schedule in which at least one message is received

Proof: Suppose $v_1 < v_2$.

Node v_1 must learn of v_2 so at least one message is received

Schedule this edge and cut the other edge

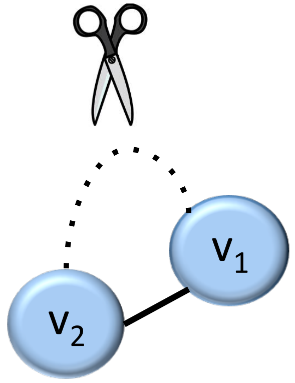


Base Case

Lemma 3

The nodes cannot distinguish the two-node schedule from one of a larger ring connection at the open edge

Proof: Noone can observe the difference between this



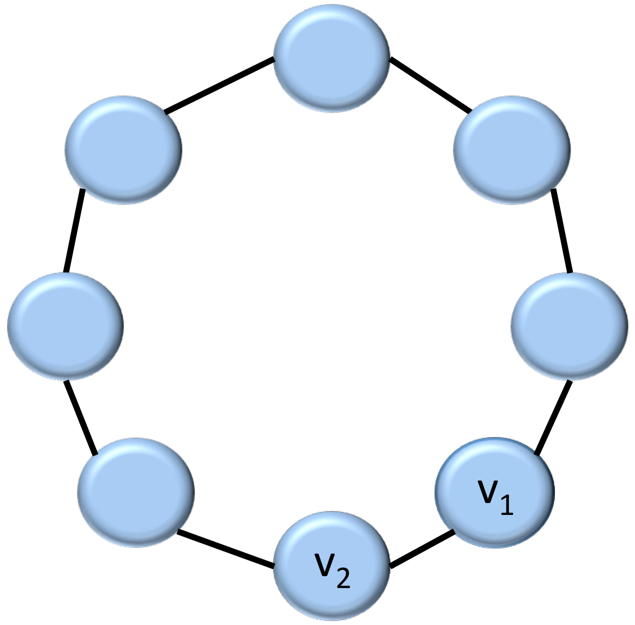
Base Case

Lemma 4

The nodes cannot distinguish the two-node schedule from one of a larger ring connection at the open edge

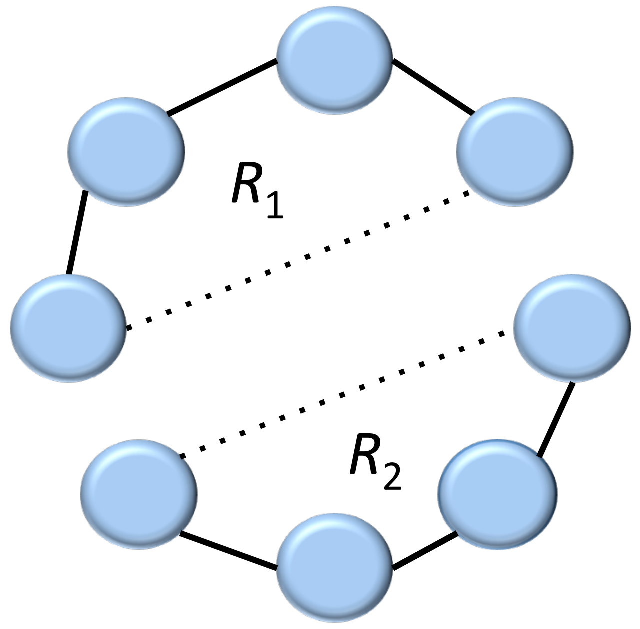
Proof: Noone can observe the difference between this and this

(as long as no messages have been schedule from the remainder of the ring)



Induction Step

Assume two rings of size $n/2$ with open schedules



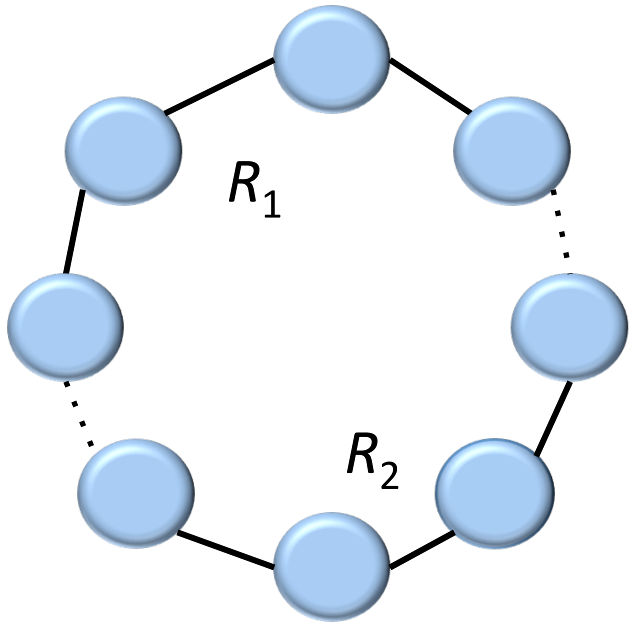
Induction Step

Assume two rings of size *n*/2 with open schedules

We can construct an open schedule on a ring of size *n*

If M(*n*/2) is number of messages can construct schedule with 2M(*n*/2) without scheduling either of the two open edges

(Remember: We decide when edges are scheduled and when nodes wake up)



Induction Step

Each node in – say – R_1 must learn of at least one node in R_2

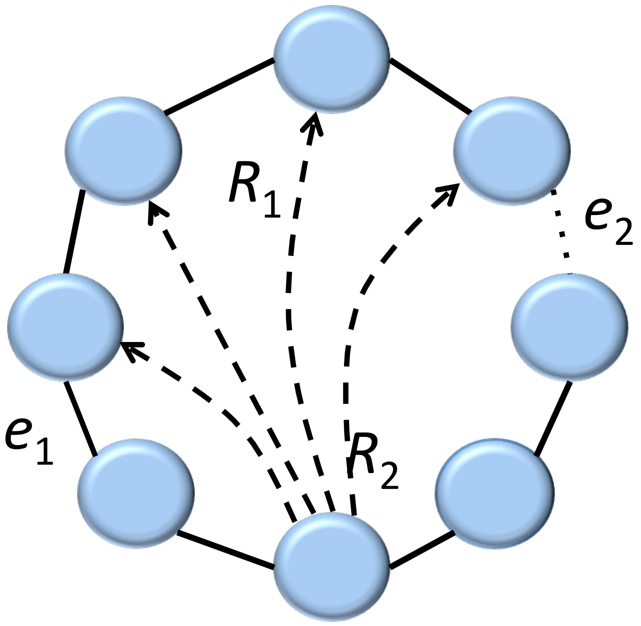
At least n/2 messages must be passed from R1 to R2

But some messages use e_1 and other use e_2 – won't do

Closing one of the edges will cause at least n/4 messages to be passed

(not necessarily over the closed edge though)

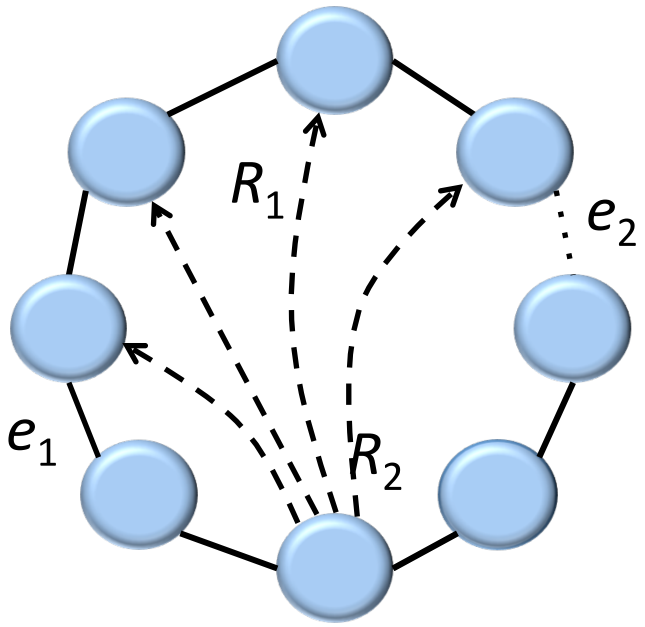
Schedule this edge and leave the other open



So We Have Shown

Lemma 5

Gluing together *R*1 and *R*2, at least 2M(*n*/2) + *n*/4 messages must be exchanged to solve leader election on the ring of size *n*



Finishing the Job

Theorem 2

Any uniform leader election algorithm for synchronous rings has at least message complexity

$$M(n) >= n/4(\log n + 1) = \Omega(n \log n)$$

Like O(n log n) but an asymptotic lower bound

Proof:

Assume n is power of 2

Base case: $M(2) >= 1 = 2/4(\log 2 + 1)$

Induction step:

$$\begin{aligned} M(n) &= 2 \cdot M(n/2) + n/4 \\ &>= 2 \cdot (n/8 (\log (n/2) + 1)) + n/4 \\ &= n/4 \log n + n/4 = n/4(\log n + 1) \end{aligned}$$

Synchronous Rings

Can we do better for synchronous rings?

Let's assume the ring size is known to be *n*

The rounds are synchronized

Unique identifiers, minimum identifier wins

Algorithm 3:

```
Code for node v with identifier id:

Initially round number is 0

Each round:
   increment round number ;
   if round number = id and no message received {
      decide to be leader ;
      broadcast “id is leader”}
```

Synchronous Leader Election: Analysis

- Message complexity is *n*
- Time complexity is id·*n*

- Solution is non-uniform
- Uniform solutions are also available, check Attiya-Welch, ch 3