

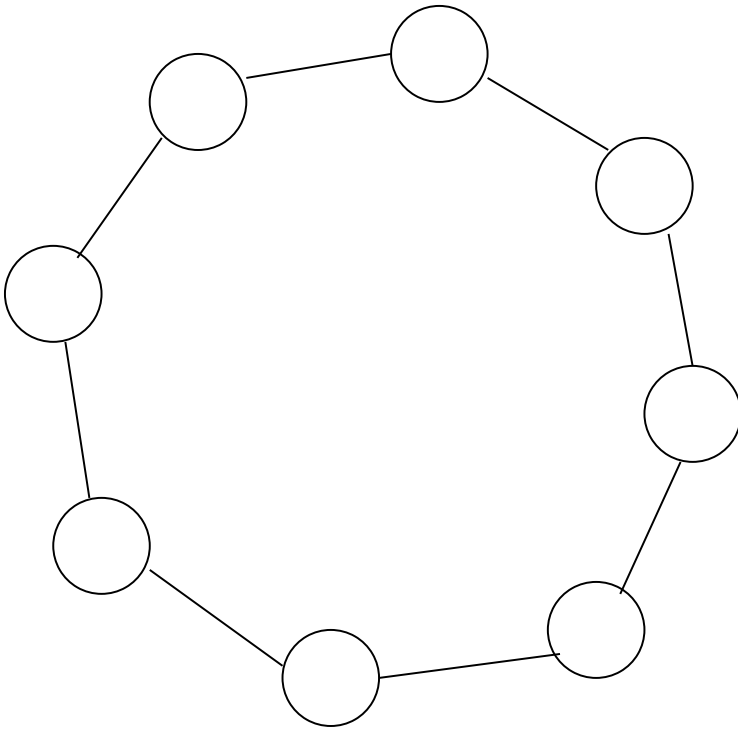
*Algorithms for COOPERATIVE DS: Leader
Election in the MPS model*

Leader Election (\mathcal{LE}) problem

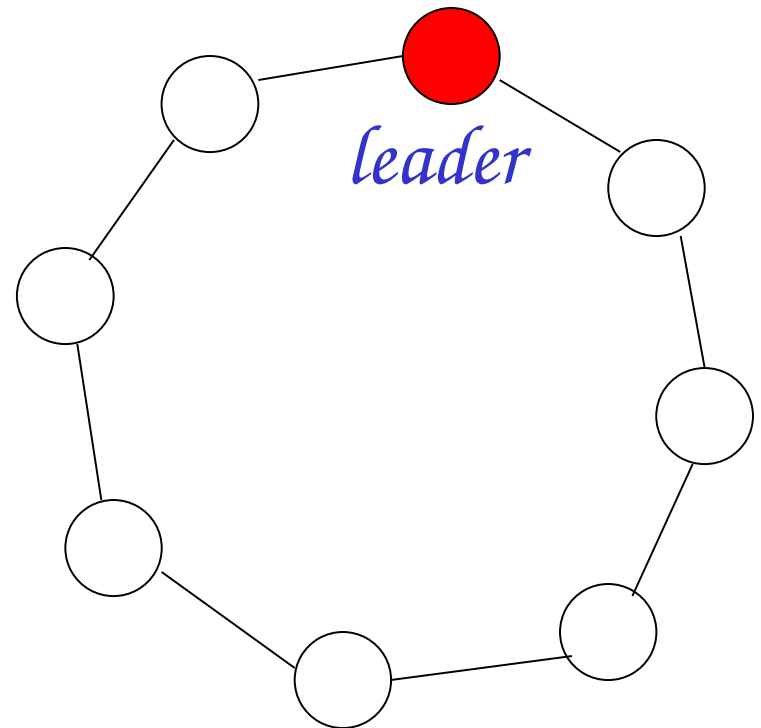
- *In a DS, it is often needed to designate a single processor (i.e., a **leader**) as the coordinator of some forthcoming task (e.g., finding a spanning tree of a graph using the leader as the root)*
- *In a \mathcal{LE} computation, each processor must decide between two internal states: either **elected** (won), or **not-elected** (lost, default **initial** state).*
- *Once an **elected state** is entered, processor will remain forever in an elected state: i.e., irreversible decision*
- ***Correctness:** In every **admissible** execution, **exactly one** processor (the **leader**) must eventually enter in the elected state, while all the remaining processors must remain in the not-elected state*

Leader Election in *Ring Networks*

Initial state
(all not-elected)



Final state

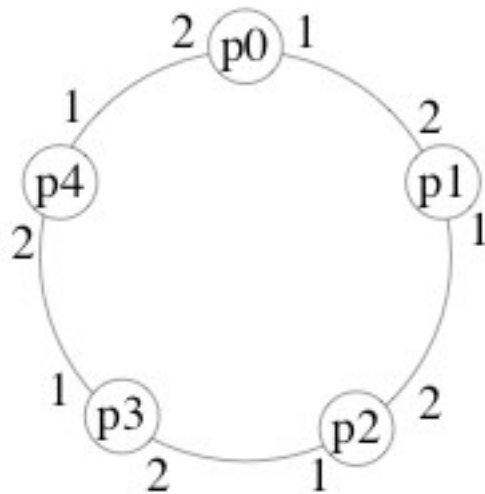


Why Studying Rings?

- *Simple starting point, easy to analyze*
- *Abstraction of a classic LAN topology*
- *Lower bounds and impossibility results for ring topology also apply to arbitrary topologies*

Sense-of-direction in Rings

In an oriented ring, processors have a consistent notion of left and right: notice that this assumption is concerned with the knowledge of a processor



1 = left = clockwise

2 = right = counter-clockwise

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

LE algorithms in rings depend on:

Anonymous Ring

Non-anonymous Ring

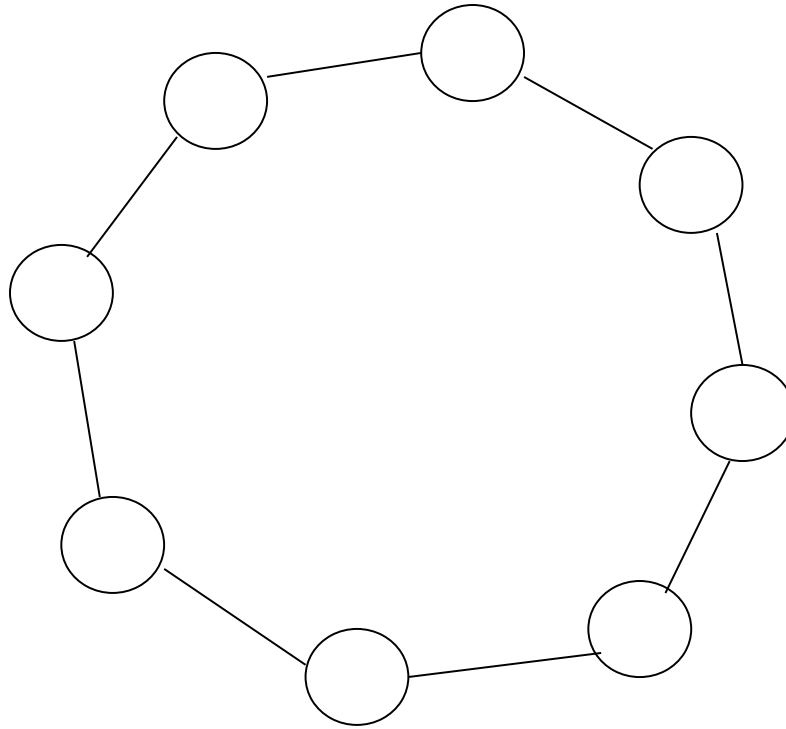
Size of the network n is known (non-unif.)

Size of the network n is not known (unif.)

Synchronous Algorithm

Asynchronous Algorithm

LE in *Anonymous* Rings



*Every processor runs **exactly** the same algorithm*

*Every processor does exactly the same
execution*

Impossibility for Anonymous Rings

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

- the algorithm knows the ring size (non-uniform)
- the algorithm is synchronous

Proof Sketch (for non-unif and sync rings, i.e., the strongest model): It suffices to show an execution in which a hypothetical algorithm will fail:

Round 1: Assume all processors wake-up simultaneously (notice this is a worst-case assumption). Of course, there is no message to read in **inbuf**. So, every processor begins in the same state (**not-elected**), perform the same internal computations, and sends the same outgoing msgs, since it is anonymous!

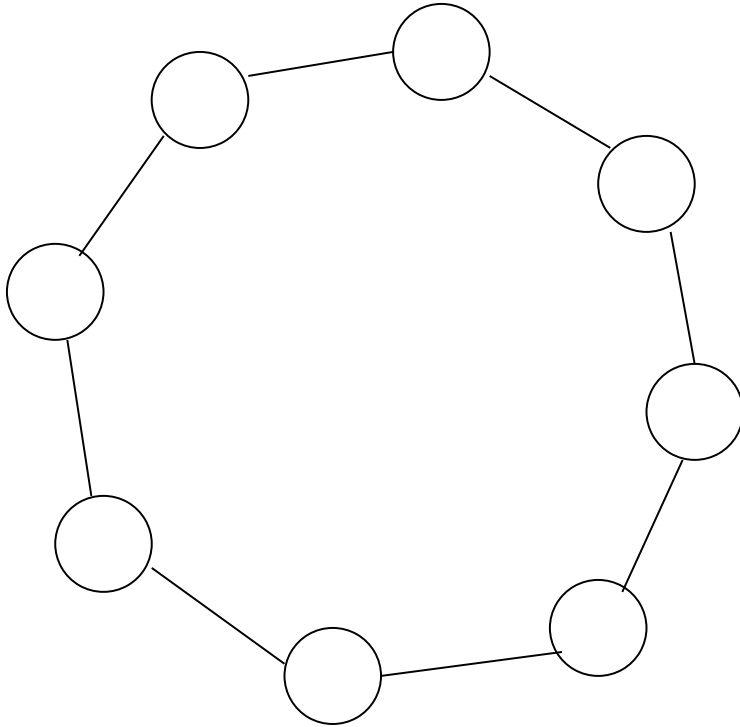
Round 2: Every processor receives same msgs in the **inbuf**, so it will do the same internal computations, and it will send the same msgs.

... and so on and so forth for **Rounds 3, 4, ..., k-1**

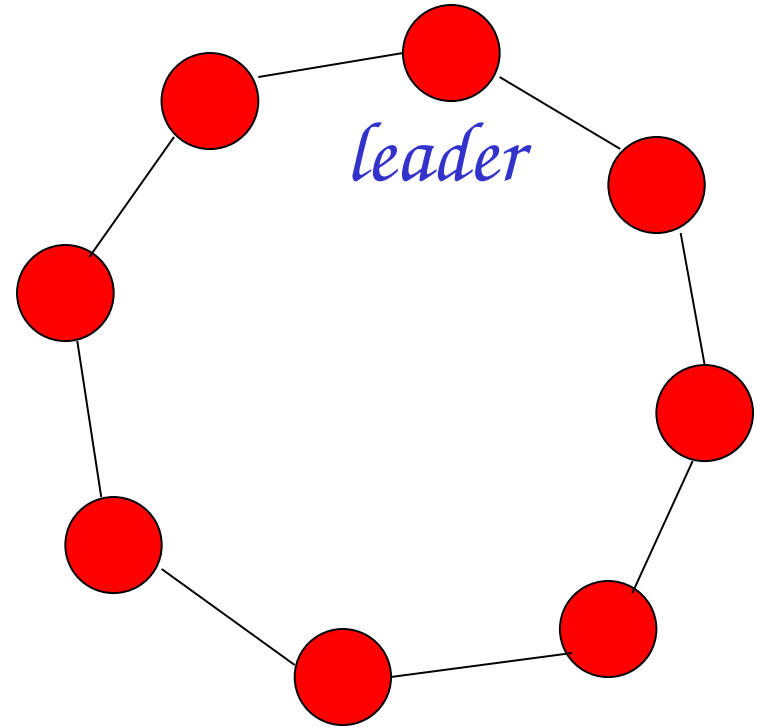
Round k: Eventually some processor is supposed to enter an **elected** state. But then they all would do \Rightarrow **incorrectness**!

Initial state

(all not-elected)



Final state



*If one node is elected leader,
then every node is elected leader*

Impossibility for Anonymous Rings

Since the theorem was proven for non-uniform and synchronous rings, the same result holds for weaker models:

uniform

asynchronous

Rings with Identifiers, i.e., non-anonymous

*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**.*

Overview of \mathcal{LE} in Rings with Ids

There exist algorithms when nodes have unique ids.

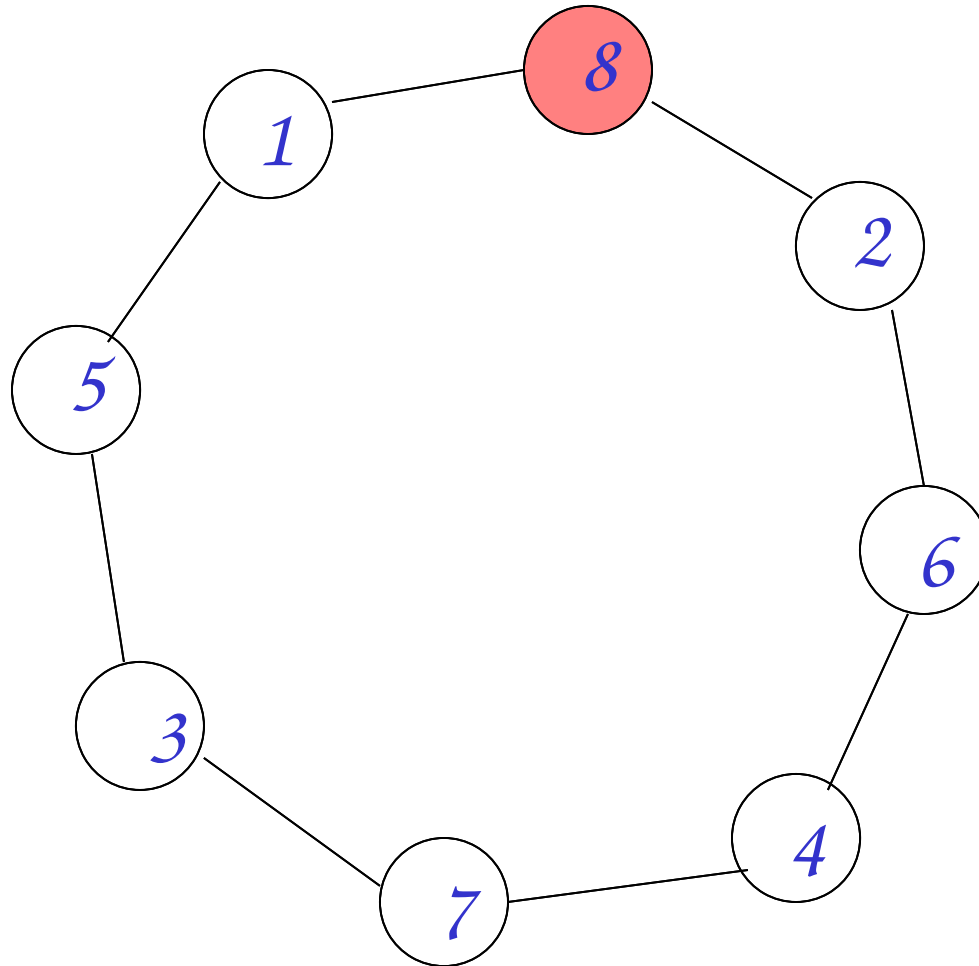
*We will evaluate them according to their **message (and time) complexity**. Best results follow:*

- *asynchronous rings:*
 - $O(n \log n)$ messages
- *synchronous rings:*
 - $\Theta(n)$ messages, time complexity depending on n and on the **magnitude of the smaller identifier in the ring**

*Above bounds are asymptotically tight (though we will not show lower bounds) and hold for **uniform** rings.*

Asynchronous Non-anonymous Rings

W.l.o.g: design an algorithm s.t. the maximum id node is elected leader



An $O(n^2)$ messages *asynchronous* algorithm: the Chang-Roberts algorithm (1979)

- Every processor which wakes-up (either spontaneously or by a message arrival, no *synchronized start* is required) sends a message with its own id to the left
- Every processor forwards to the left any message with an id greater than its own id
- If a processor receives its own id it elects itself as the *leader*, and announces this to the others
- **Remark:** it is *uniform* (number of processors does not need to be known by the algorithm)
- We will show the algorithm requires $O(n^2)$ messages; we use O notation because *not all the* executions of the algorithm costs n^2 , in an asymptotic sense, but only some of them, as we will see

CR algorithm: pseudo-code for a generic processor

```
boolean participant=false;  
int leader_id=null;
```

To initiate an election:

```
send(ELECTION $\langle$ my_id $\rangle$ );  
participant:=true;
```

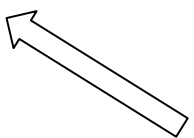
Upon receiving a message **ELECTION** \langle *j* \rangle :

```
if (j > my_id) then send(ELECTION $\langle$ j $\rangle$ );  
if (my_id = j) then send(LEADER $\langle$ my_id $\rangle$ );  
if ((my_id > j)  $\wedge$  ( $\neg$ participant)) then  
    send(ELECTION $\langle$ my_id $\rangle$ );  
participant:=true;
```

Upon receiving a message **LEADER** \langle *j* \rangle :

```
leader_id:=j;  
if (my_id  $\neq$  j) then send(LEADER $\langle$ j $\rangle$ );
```

Remark: this guarantees that the processor with largest id will participate to the election, and so eventually there will be a leader



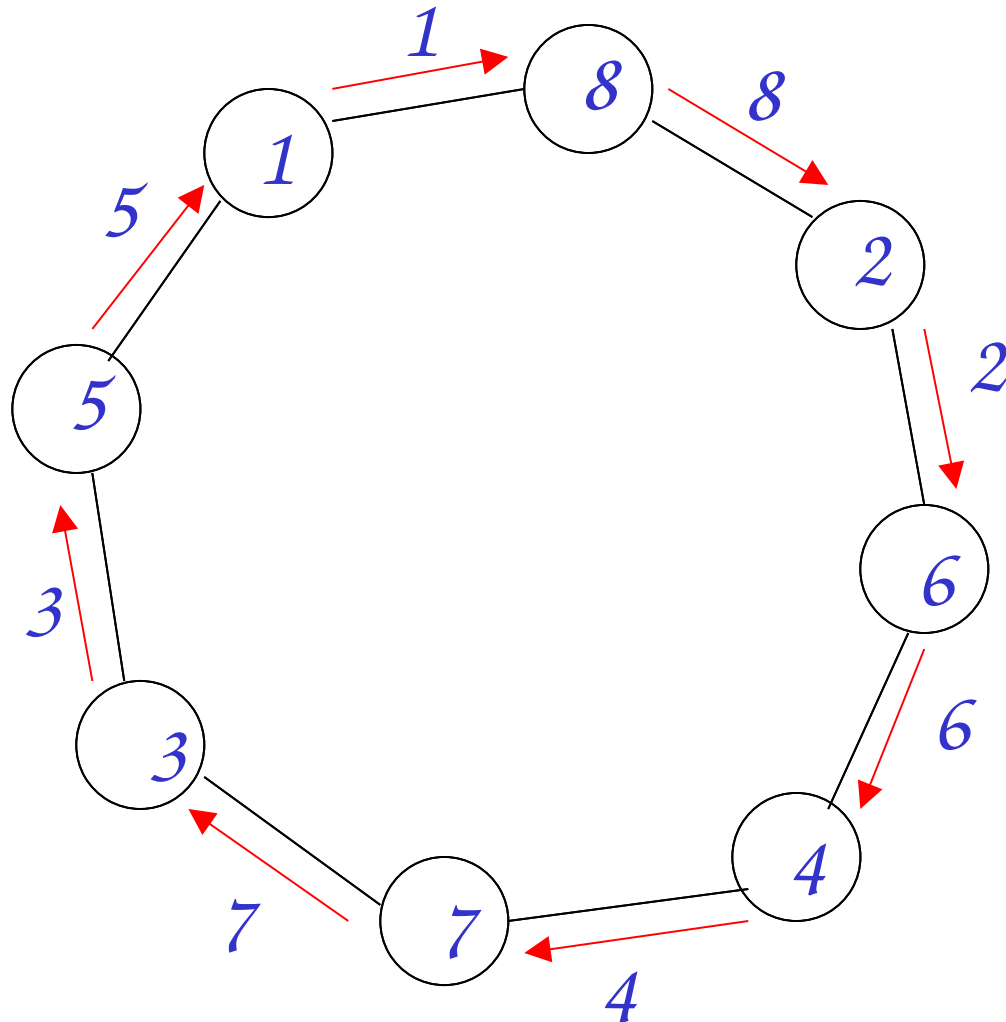
Chang-Roberts algorithm: an execution

(all the nodes start together)

End of Phase 1:

Each node sent a message with its id

to the left neighbor, and all the messages arrived at destination



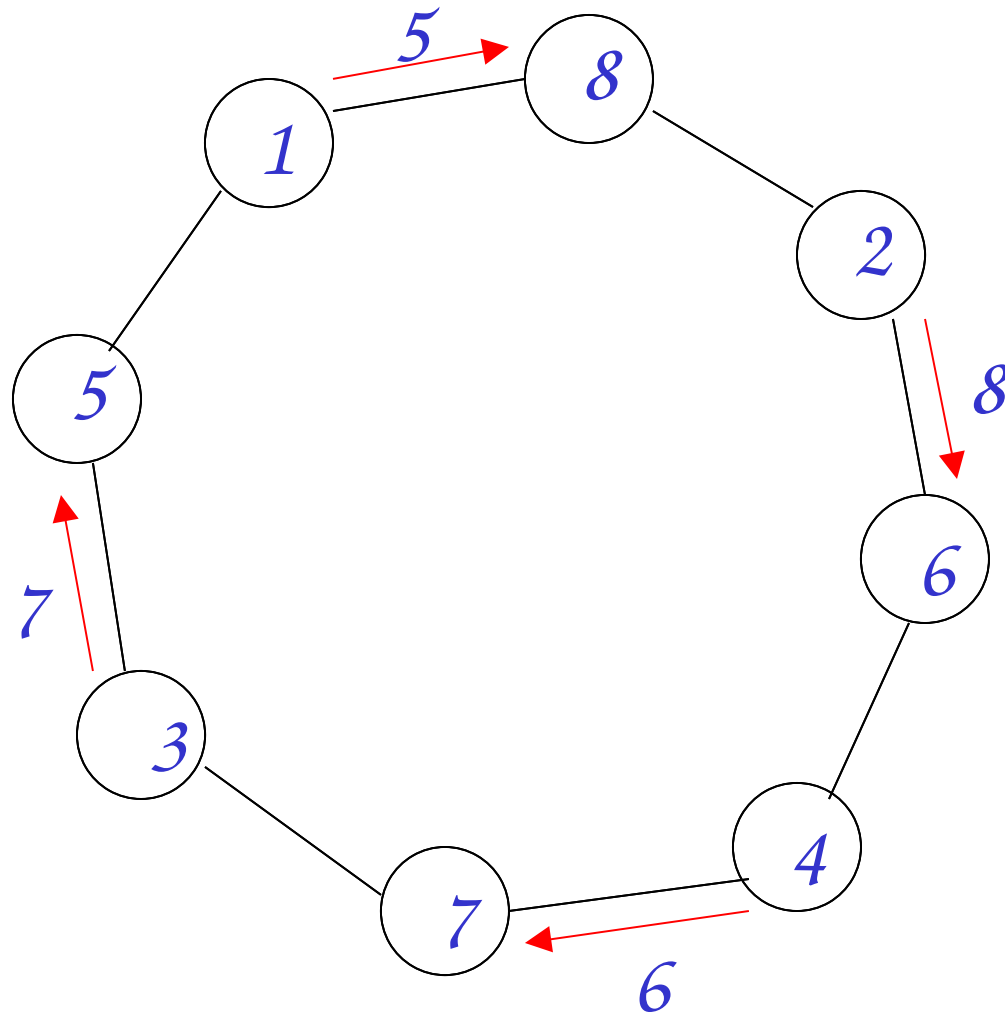
*Beginning of **Phase 2**: at each processor*

If: received id > my own id

Then: forward message \Rightarrow Only ids 5, 6, 7, 8 are forwarded

End of Phase 2:

*5, 6, 7 and 8
arrive at
destination*

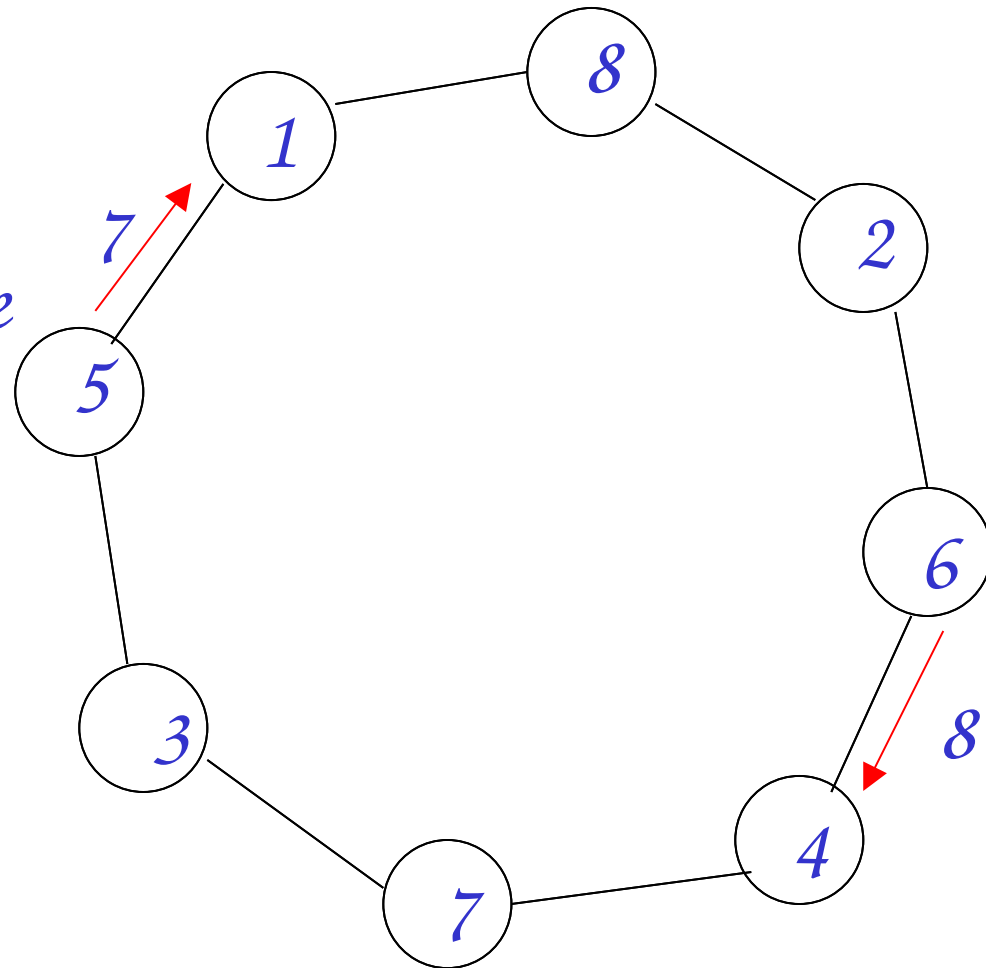


*Beginning of **Phase 3**: at processors 5, 8, 6, 7*

If: received id > my own id

Then: forward message \Rightarrow Only ids 7 and 8 are forwarded

***End of Phase3:**
Ids 7 and 8 arrive
at destination*



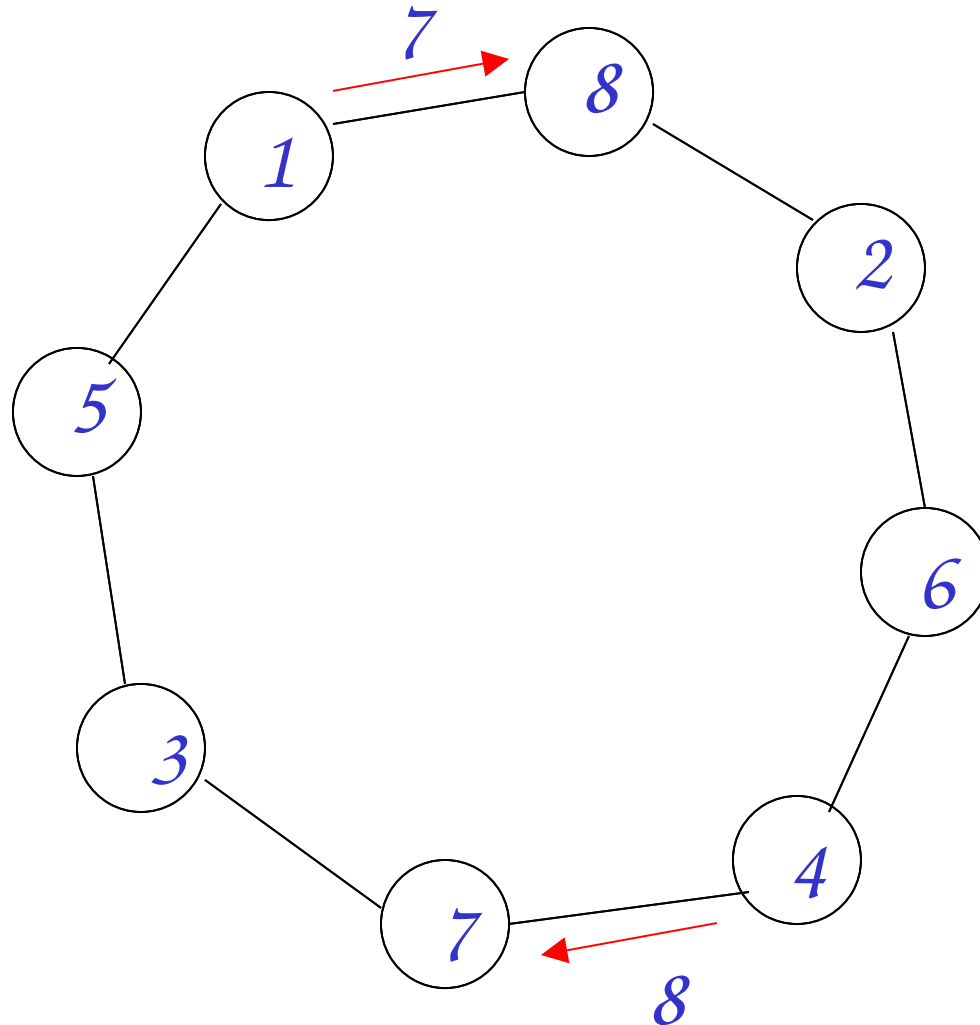
Beginning of Phase 4: at processors 1 and 4

If: received id > my own id

Then: forward message \Rightarrow ids 7 and 8 are forwarded

End of Phase 4:

*Ids 7 and 8
arrive at
destination*



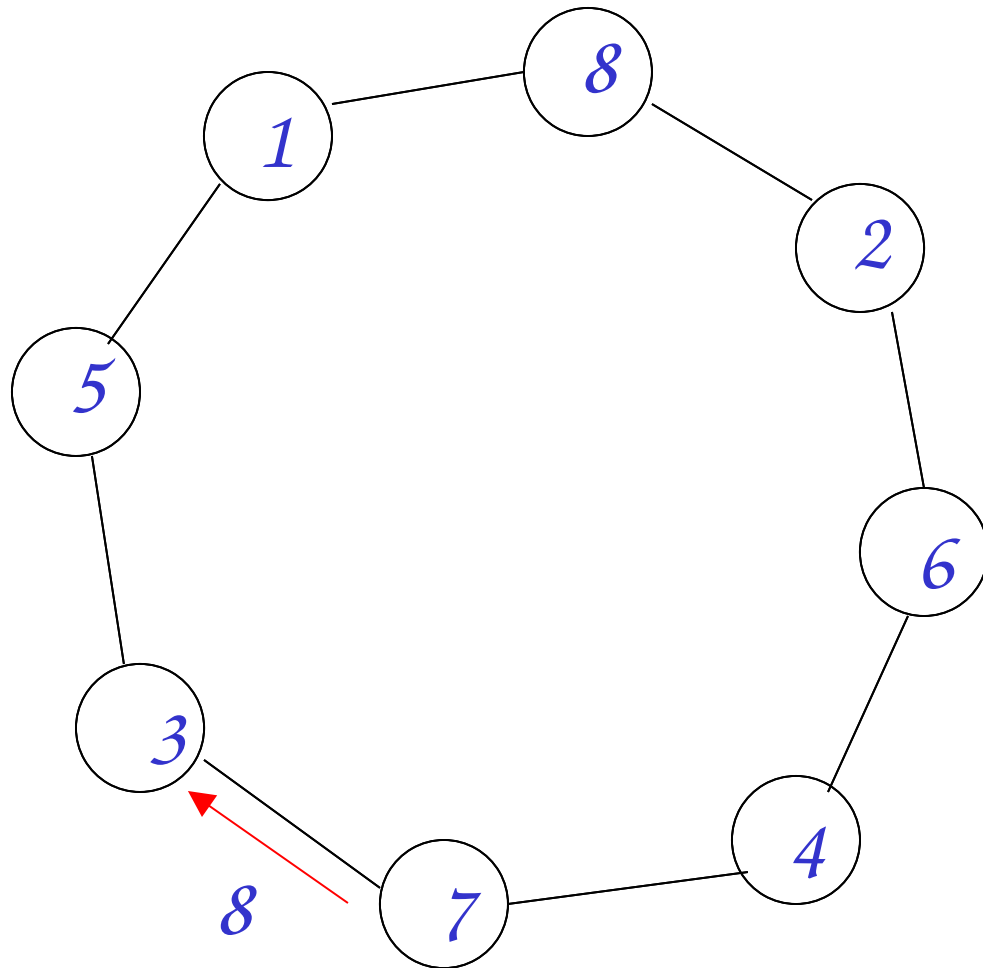
Beginning of Phase 5: at processor 7 and 8

If: received id > my own id

Then: forward message \Rightarrow Only id 8 is forwarded

End of Phase 5:

*Id 8 arrives at
destination*



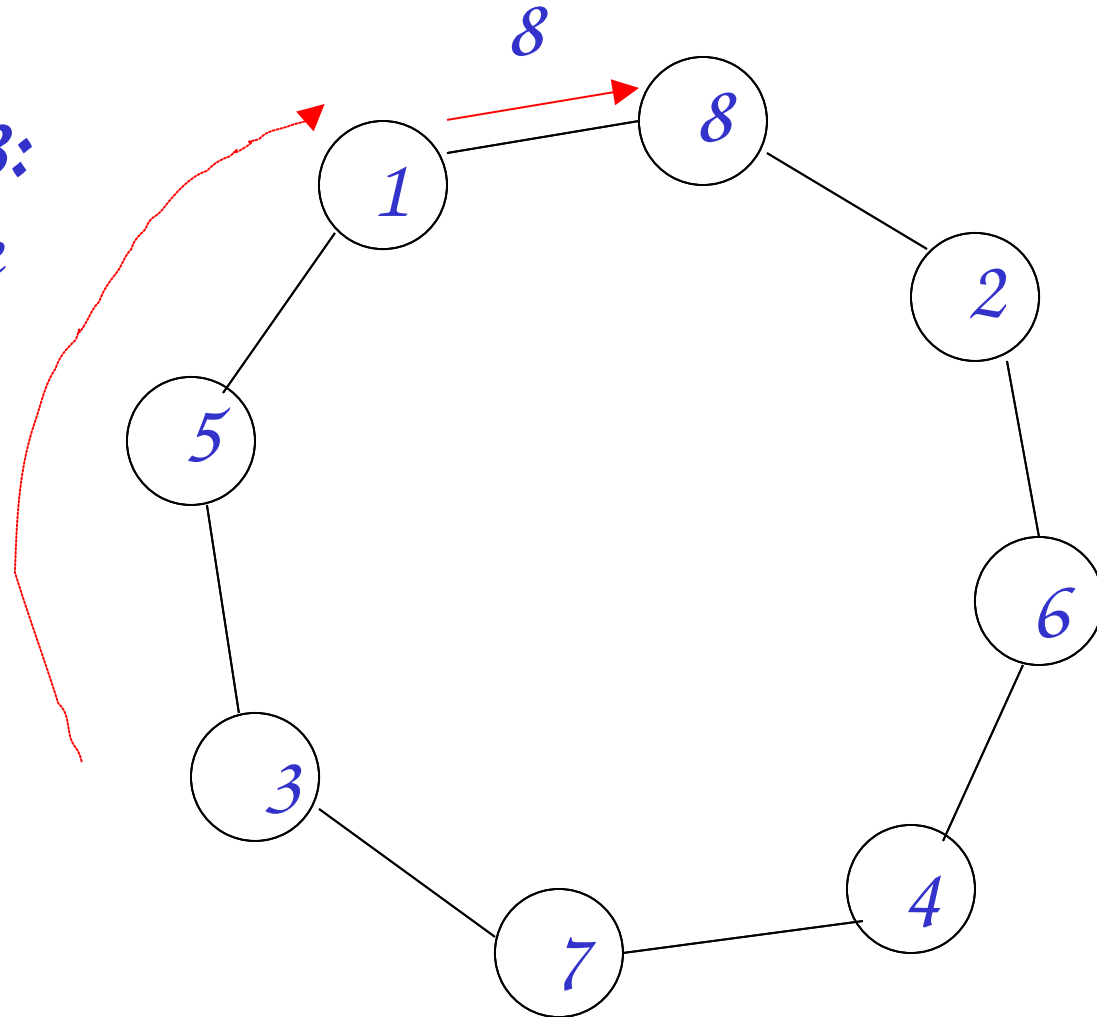
*Beginning of **Phase 8**: at processor 1*

If: received id > my own id

Then: forward message

End of Phase 8:

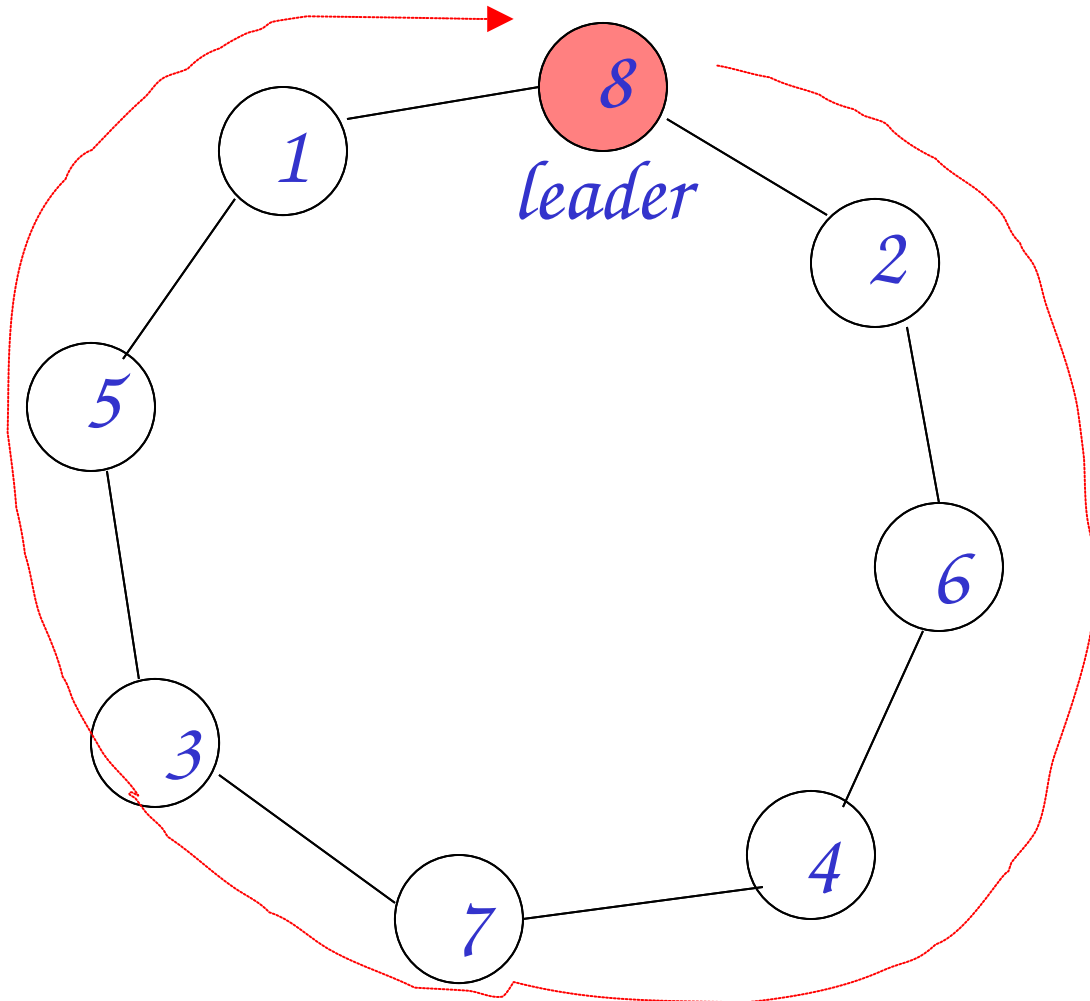
*Id 8 will arrive
to the base*



Beginning of Phase 9: at processor 8

If: a node receives its own message

Then: it elects itself a leader



Leader announcement:

*All the nodes
will be notified
that 8 is the
leader*

Analysis of Chang-Roberts algorithm

Correctness: Elects processor with **largest id**.

Indeed, the message containing the largest id passes through every processor, while all other messages will be stopped somewhere

Message complexity: Depends on 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 either largest or second largest
etc.*

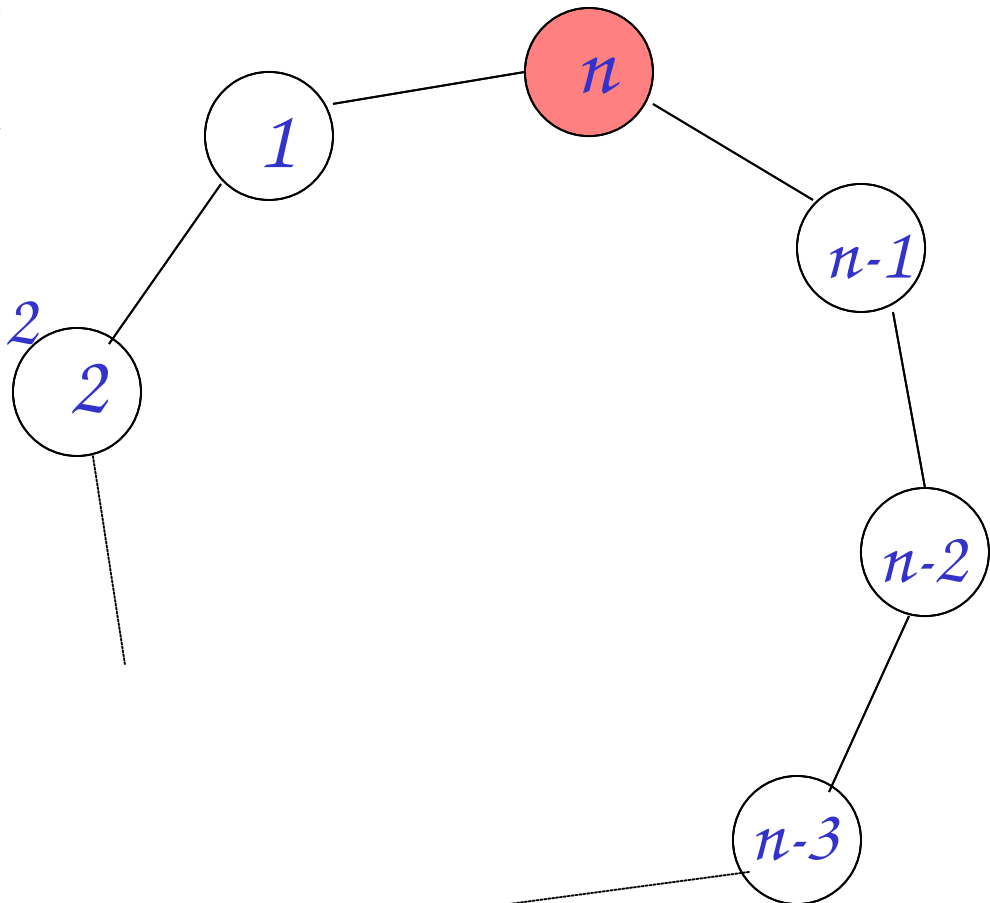
Worst case: $\Theta(n^2)$ messages

*Worst way to arrange the
ids is in decreasing order:*

*2nd largest causes $n - 1$
messages*

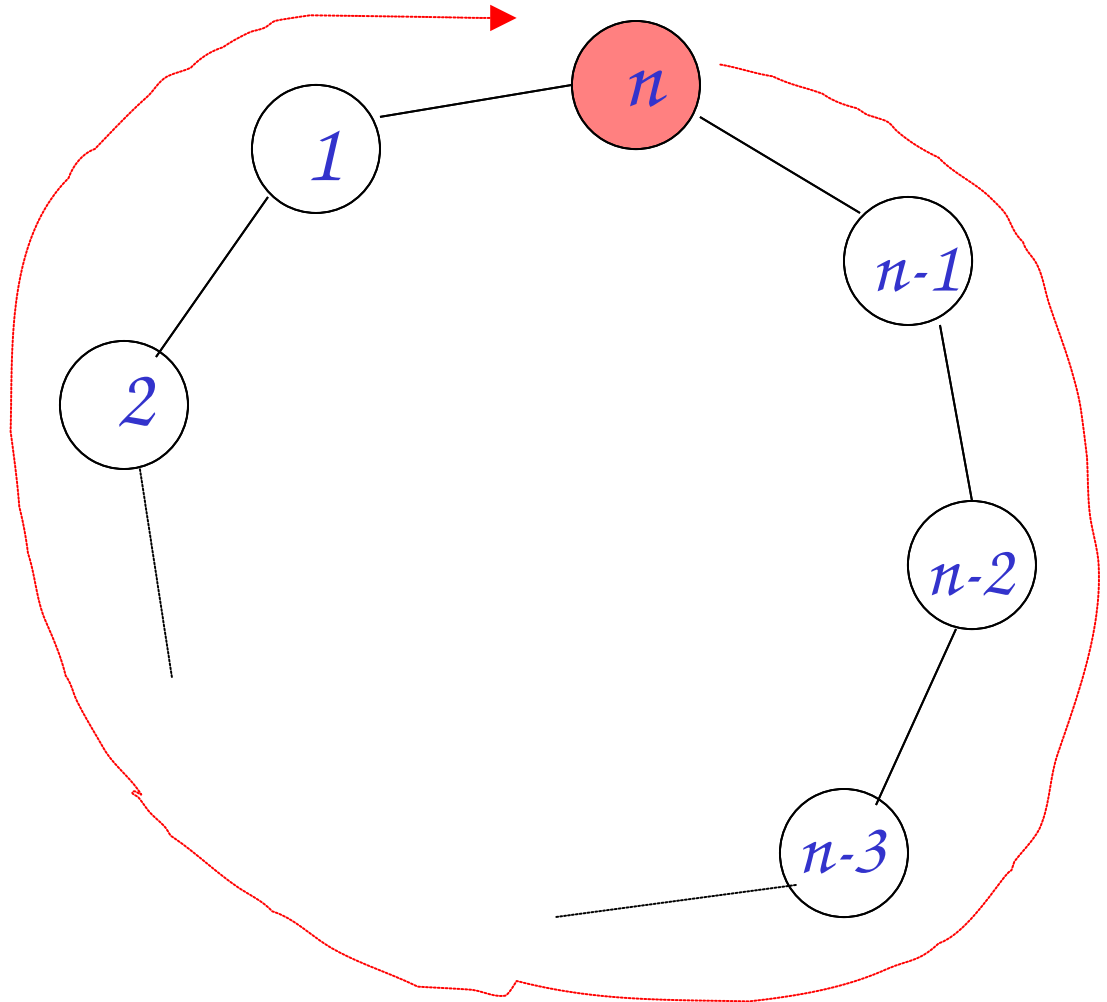
*3rd largest causes $n - 2$
messages*

etc.



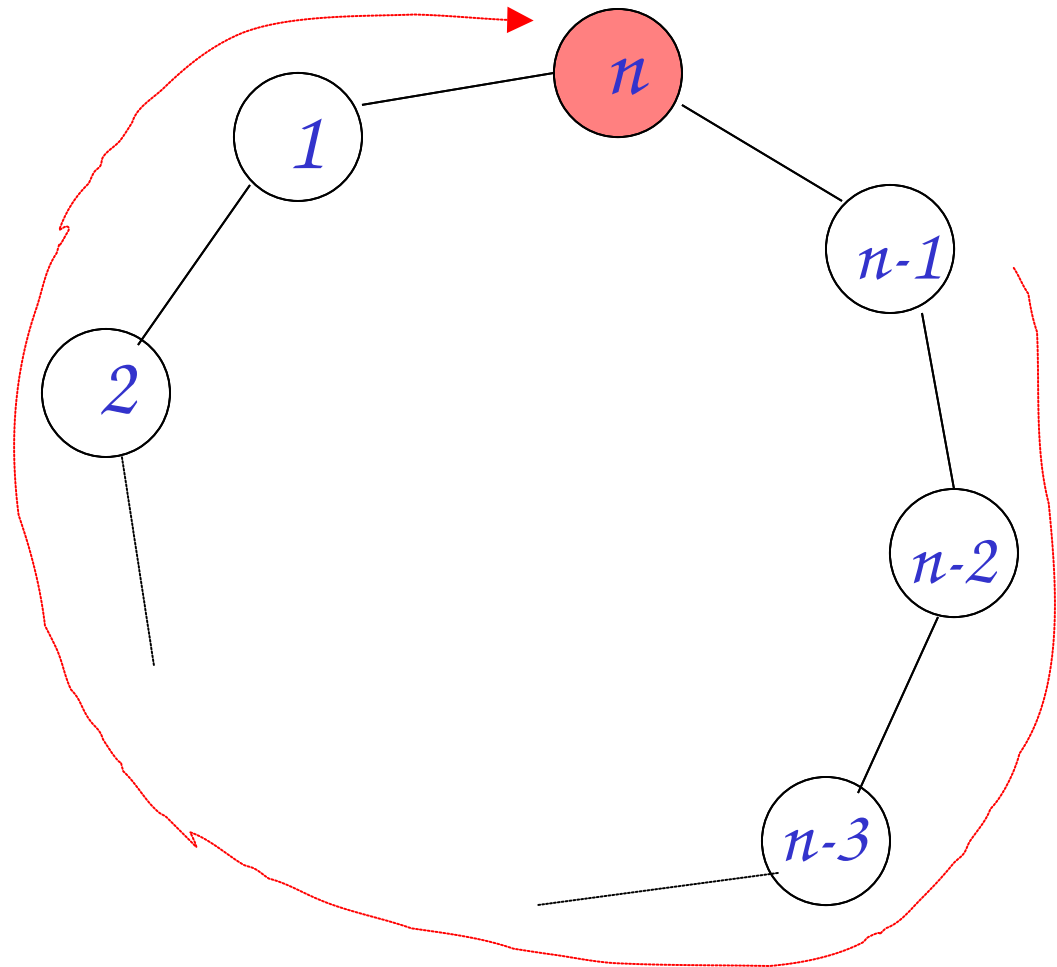
Worst case: $\Theta(n^2)$ messages

n messages



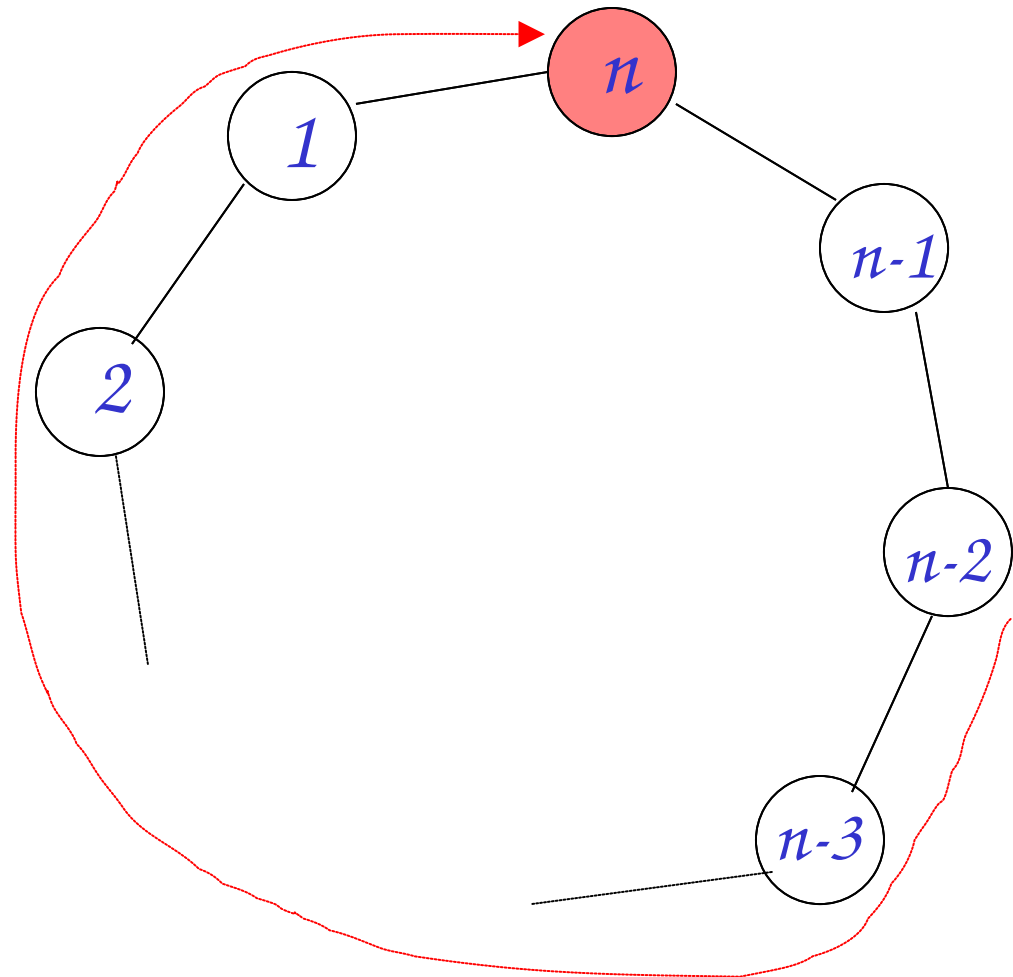
Worst case: $\Theta(n^2)$ messages

$n-1$ messages



Worst case: $\Theta(n^2)$ messages

$n-2$ messages



Worst case: $\Theta(n^2)$ messages

Total messages:

$n +$

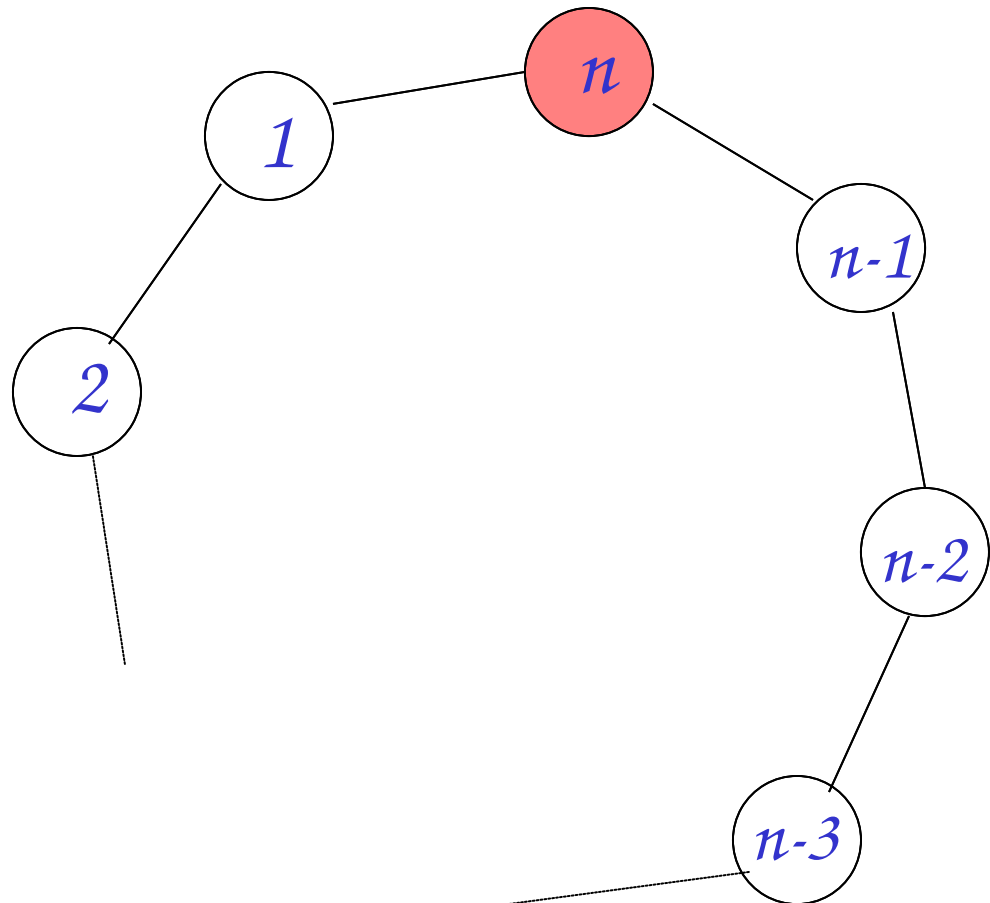
$n - 1 +$

$n - 2 +$

\dots

$2 +$

$1 = \Theta(n^2)$



Best case: $\Theta(n)$ messages

Total messages:

$n +$

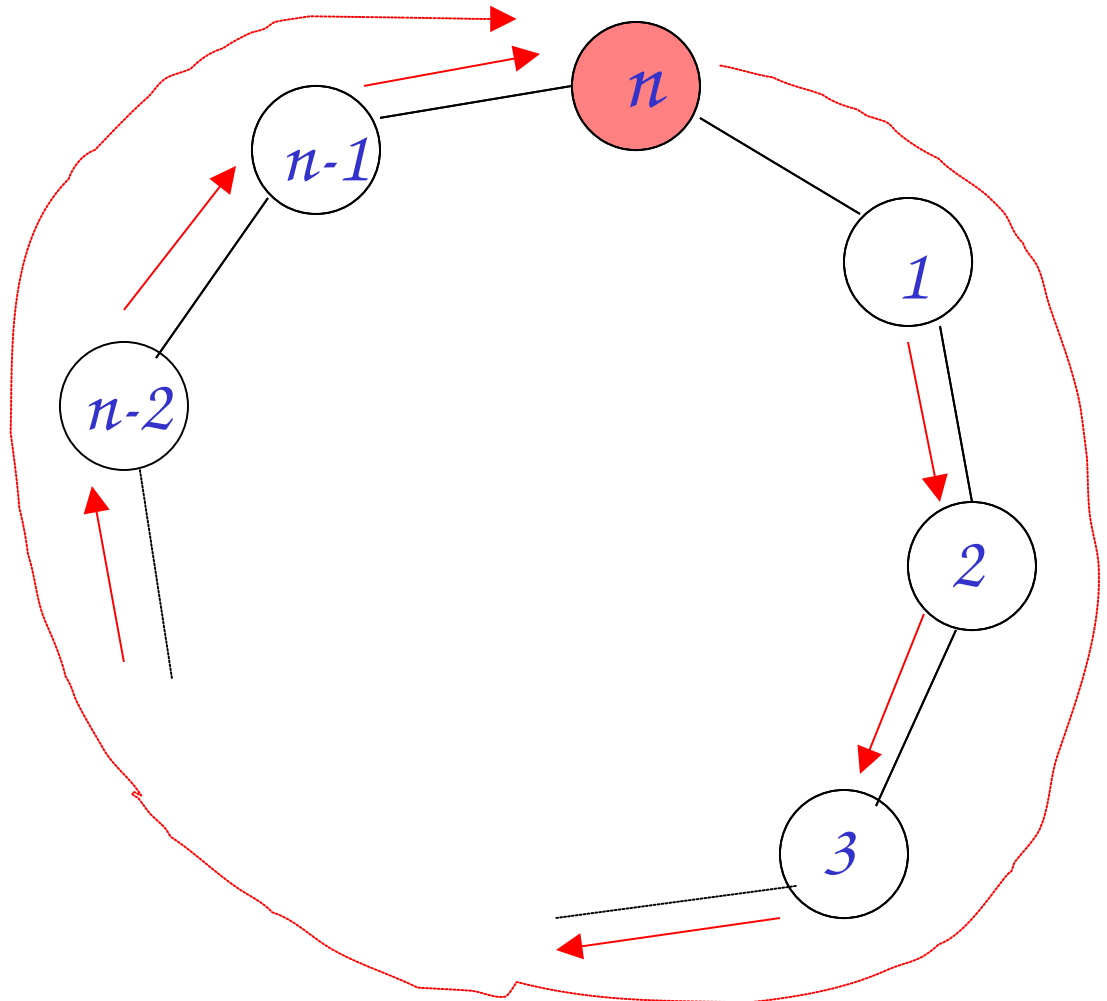
$1 +$

$1 +$

\dots

$1 +$

$1 = \Theta(n)$



Average case analysis CR-algorithm

Theorem: The average message complexity of the CR-algorithm is $\Theta(n \log n)$.

Sketch of proof: Assume all $n!$ rings (all possible permutations) are equiprobable, and assume that all processors wake-up simultaneously:

- Probability that a generic id makes exactly 1 step is equal to the probability it makes at least 1 step minus the probability it makes at least 2 steps: $\text{Prob}(\text{to make exactly 1 step}) = 1 - 1/2 = 1/2$
[indeed, to make at least 2 steps, the id must be greater than the id on its clockwise side, and this happens with probability $1/2$]
- Probability that a generic id makes exactly 2 steps is equal to the probability it makes at least 2 steps minus the probability it makes at least 3 steps: $\text{Prob}(\text{to make exactly 2 steps}) = 1/2 - 1/3 = 1/6$
[indeed, to make at least 3 steps, the id must be greater than the next 2 ids on its clockwise side, and this happens with probability $1/3$]
- ...
- Probability that a generic id makes exactly k steps is equal to the probability it makes at least k steps minus the probability it makes at least $k+1$ steps: $\text{Prob}(\text{to make exactly } k \text{ steps}) = 1/k - 1/(k+1) = 1/k(k+1)$
- ...
- Probability that a generic id makes exactly n steps is just $1/n$

Average case analysis CR-algorithm (2)

⇒ Expected number of steps (i.e., of messages) for each id is

$$\begin{aligned}\mathbb{E}(\# \text{ messages}) &= \sum_{i=1, \dots, n} i \cdot \text{Prob}(\text{to make exactly } i \text{ steps}) = \\ &= 1 \cdot 1/2 + 2 \cdot 1/6 + 3 \cdot 1/12 + \dots + (n-1) \cdot 1/[n(n-1)] + n \cdot 1/n = \\ &= 1/2 + 1/3 + 1/4 + \dots + 1/n + 1 = 1 + \sum_{i=2, \dots, n} 1/i \leq \\ &\leq 1 + \int_1^n 1/i \, di = 1 + \ln n - \ln 1 = 1 + \ln n = \Theta(\log n)\end{aligned}$$

⇒ Average message complexity is:

$\Theta(n \log n)$ (i.e., $\Theta(\log n)$ for each of the n ids – remember that we assumed all were waking up at the same time) + n (for the leader announcement)
= $\Theta(n \log n)$.

•

Can We Use Fewer Messages?

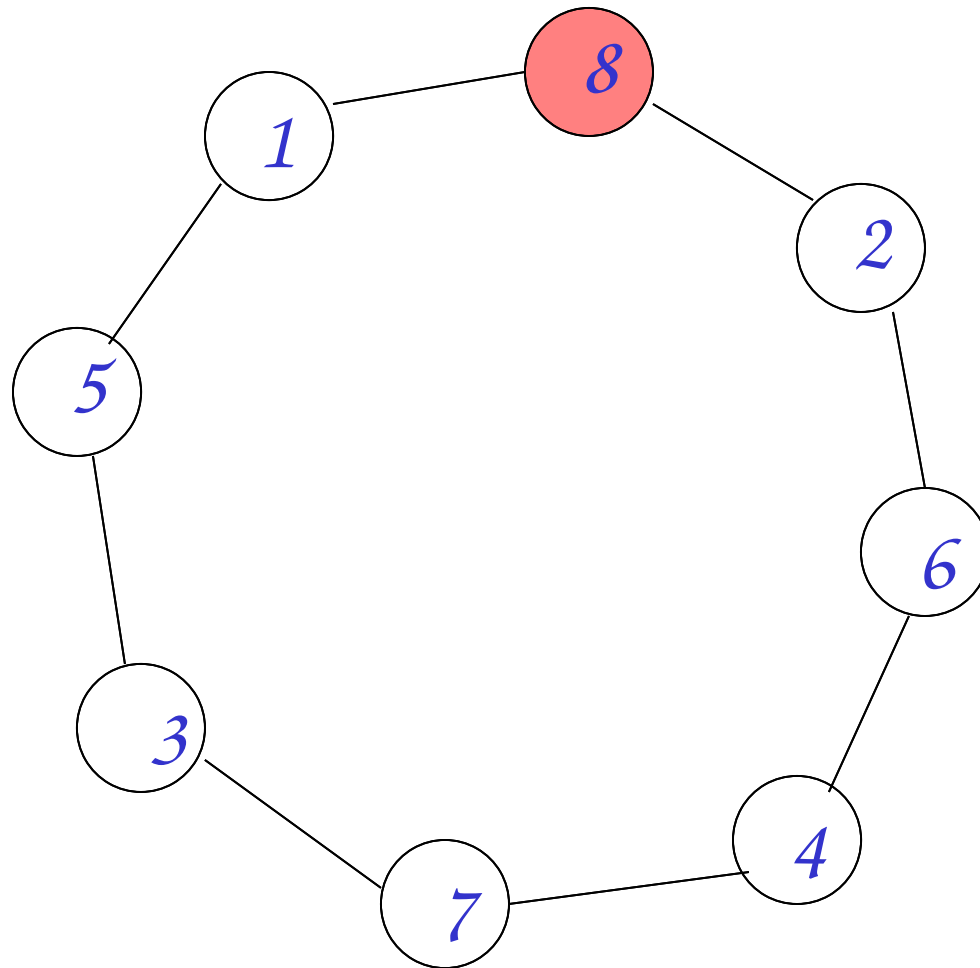
The $O(n^2)$ algorithm is simple and works in both synchronous and asynchronous model.

But can we solve the problem with fewer messages?

Idea:

Try to have msgs containing larger ids travel smaller distance in the ring

An $O(n \log n)$ messages *asynchronous* algorithm:
the Hirschberg-Sinclair algorithm (1980)
Again, the maximum id node is elected leader



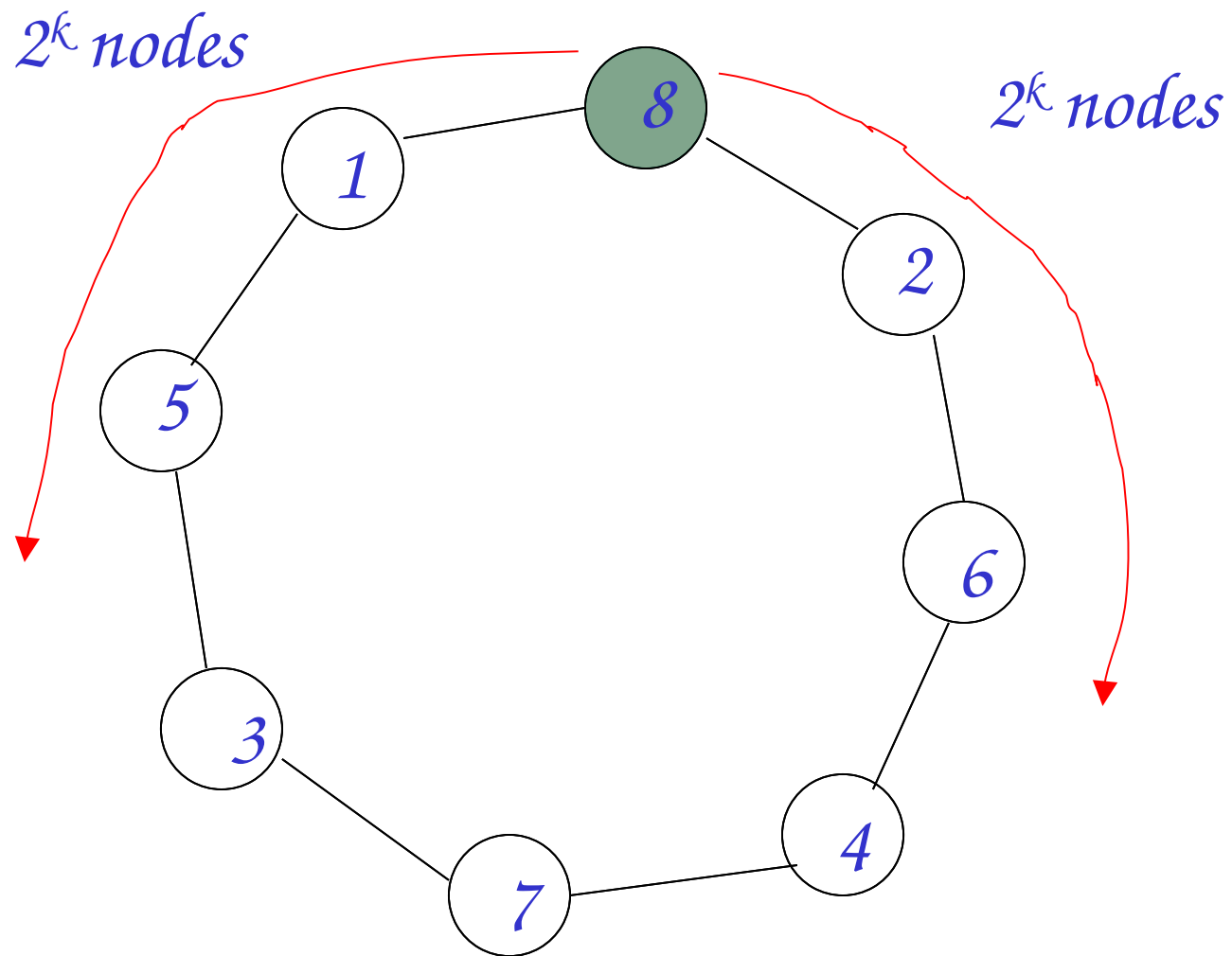
Hirschberg-Sinclair algorithm (1)

- Assume ring is bidirectional
- Carry out elections on *increasingly larger sets*
- Algorithm works in (asynchronous) *phases* $k=0,1,2,\dots$
- No *synchronized start* is required: Every processor which wakes-up (either spontaneously or by a message arrival), tries to elect itself as a *temporary leader* of the current phase to access to the next phase
- p_i becomes a *temporary leader* in phase $k=0,1,2,\dots$ iff it has the *largest* id of its *2^k -neighborhood*, namely of all nodes that are at a distance 2^k or less from it; to establish that, it sends *probing* messages on both sides
- *Probing* in phase k requires at most $4 \cdot 2^k$ messages for each processor trying to become leader

Message types

1. *Probing (or election) message*: it travels from the temporary leader towards the periphery of the actual neighborhood and will contain the fields (*id*, *current phase*, *step counter*); as for the CR-algorithm, a probing message *will be stopped* if it reaches a processor with a larger *id*
2. *Reply message*: it travels from the periphery of the actual neighborhood towards the temporary leader and will contain the fields (*id* (of the temporary leader), *current phase*)

2^k - neighborhood of



Hirschberg-Sinclair algorithm (2)

- *Only processors that win the election in phase k can proceed to phase $k+1$*
- *If a processor receives a **probe** message with its own id, it elects itself as **leader***
- ***Remark:** it is **uniform** (number of processors does not need to be known by the algorithm)*

HS algorithm: pseudo-code for a generic processor

To initiate an election (phase 0):

```
send(ELECTION $\langle my\_id, 0, 1 \rangle$ ) to left and right;
```

Upon receiving a message **ELECTION** $\langle j, k, d \rangle$ from left (right):

```
if  $((j > my\_id) \wedge (d < 2^k))$  then
```

```
    send(ELECTION $\langle j, k, d + 1 \rangle$ ) to right (left);
```

```
if  $((j > my\_id) \wedge (d = 2^k))$  then
```

```
    send(REPLY $\langle j, k \rangle$ ) to left (right);
```

```
if  $(my\_id = j)$  then announce itself as leader;
```

Upon receiving a message **REPLY** $\langle j, k \rangle$ from left (right):

```
if  $(my\_id \neq j)$  then
```

```
    send(REPLY $\langle j, k \rangle$ ) to right (left);
```

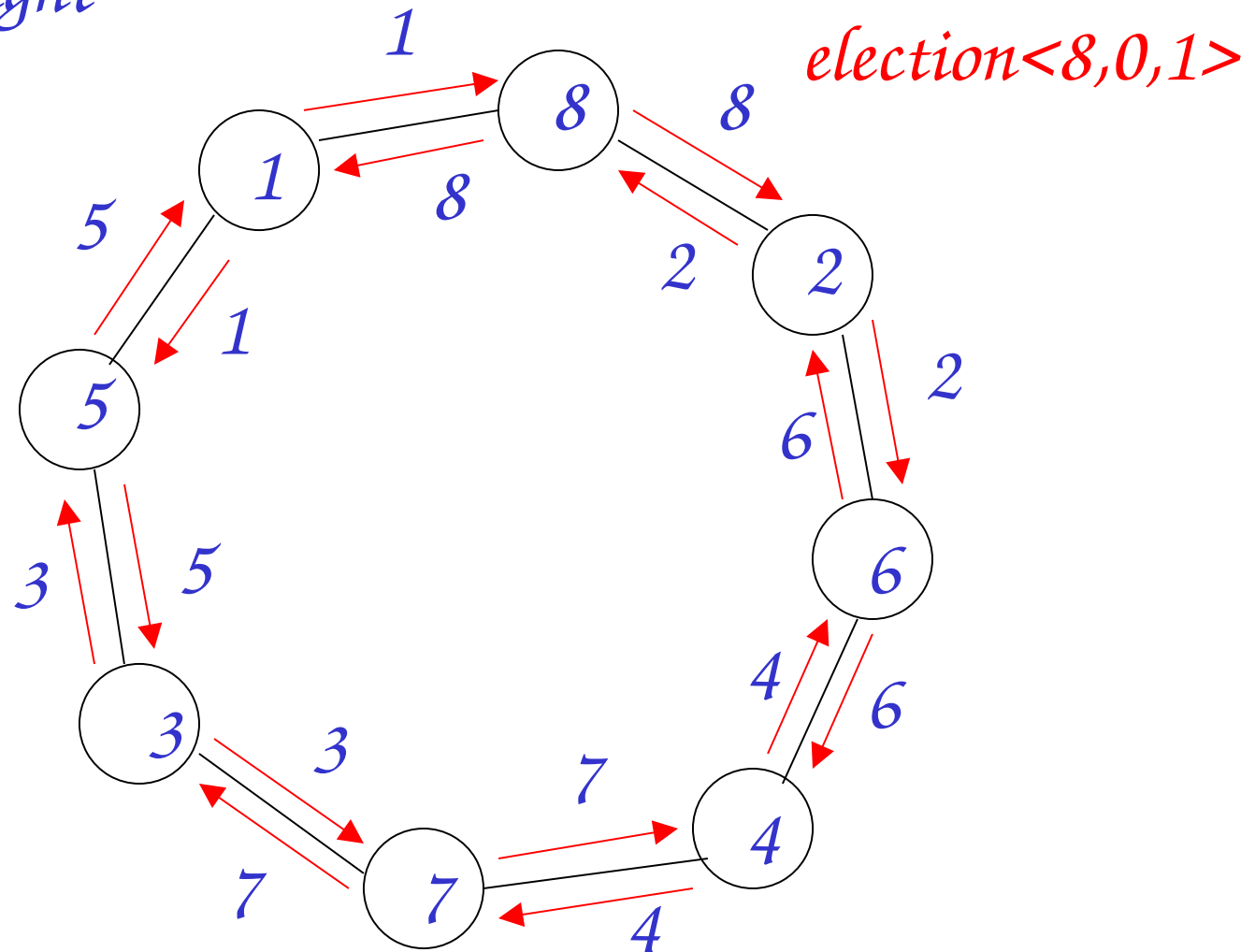
```
else
```

```
    if (already received REPLY $\langle j, k \rangle$ )
```

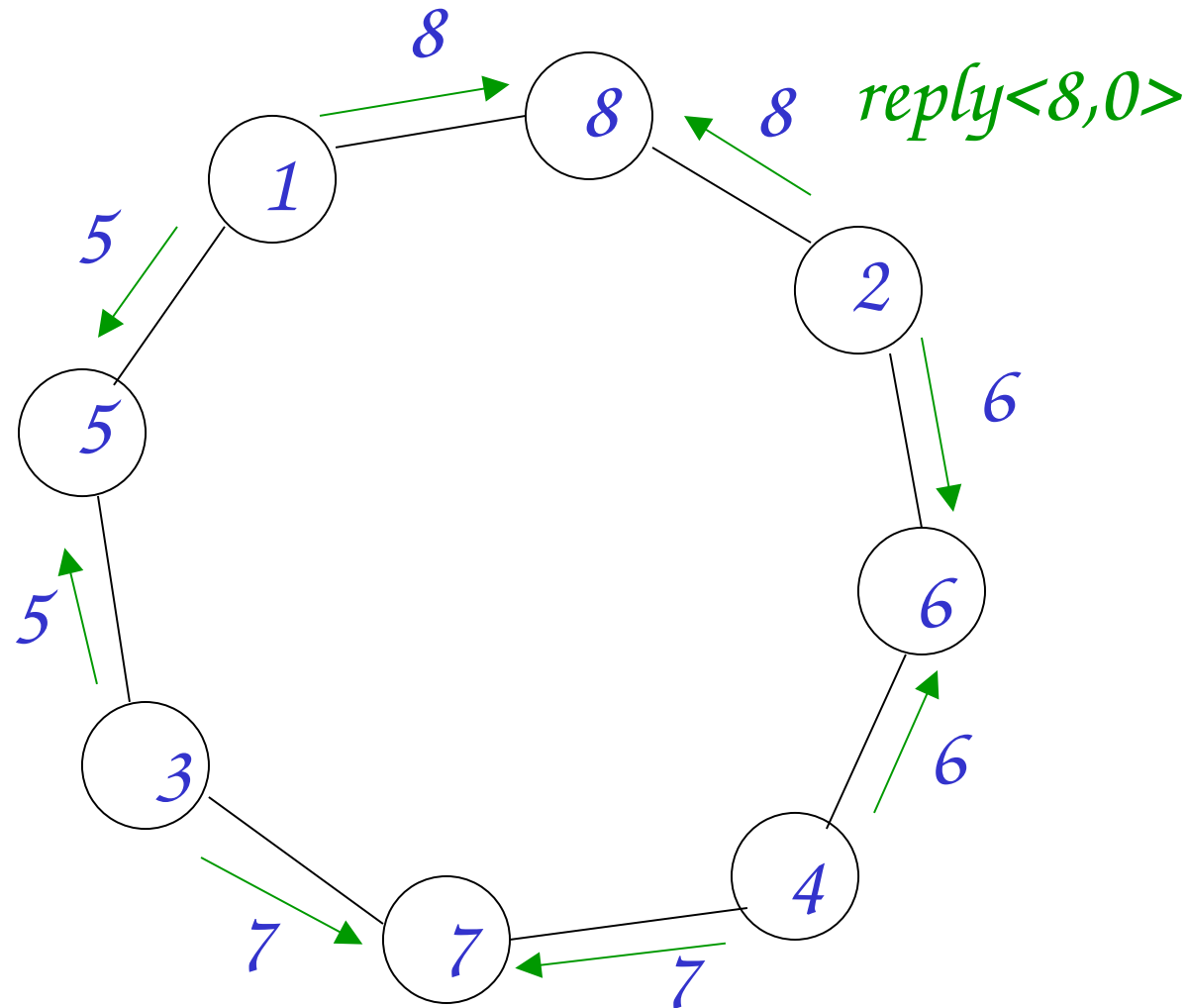
```
        send(ELECTION $\langle j, k + 1, 1 \rangle$ ) to left and right;
```

*Question: do we really need the phase field in a **REPLY** message?*

Phase 0: each node sends a probing message (id, 0, 1) to its $2^0=1$ -neighborhood, i.e., to its left and its right

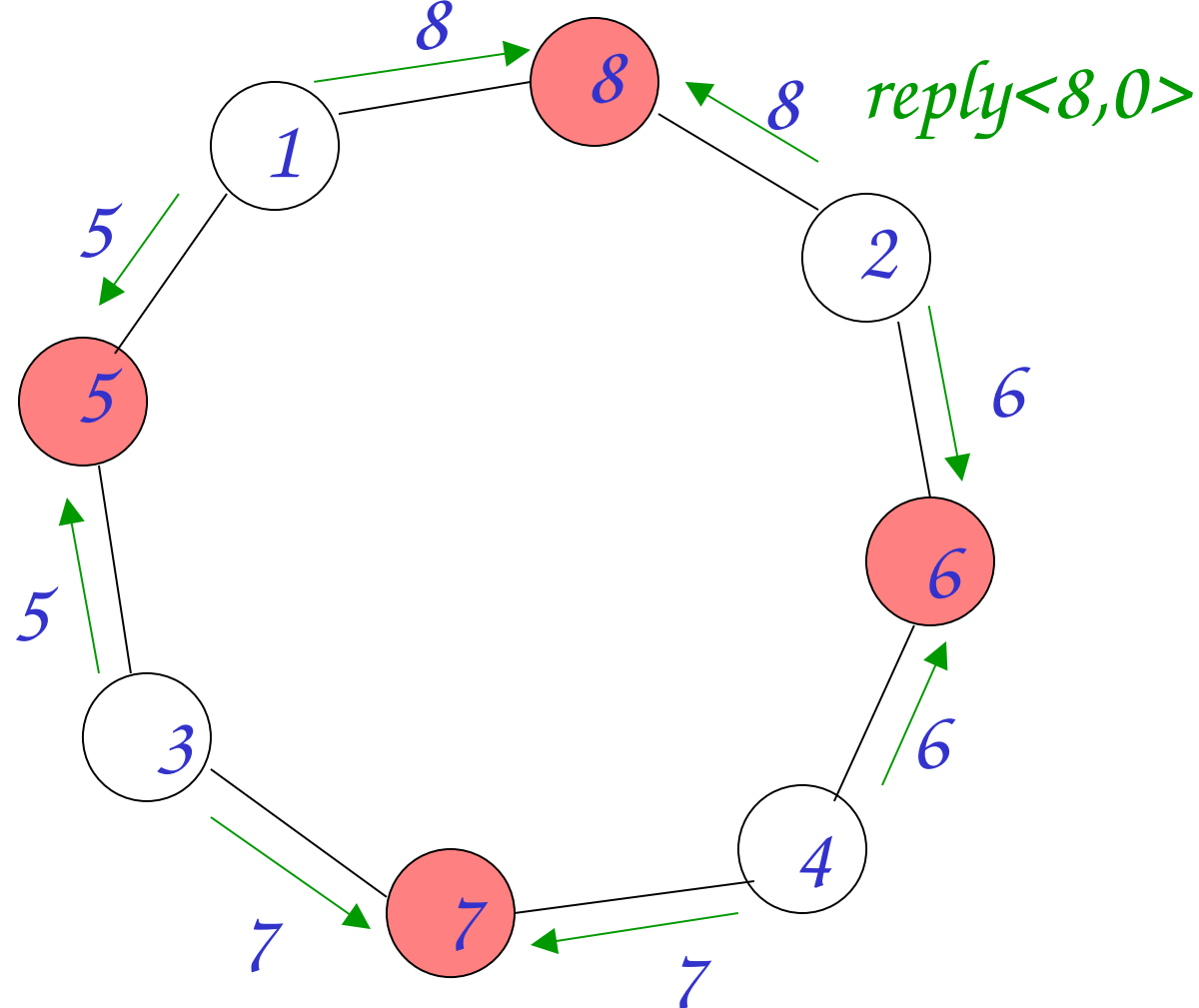


Phase 0: each node receives a probing message (id, 0, 1) from its left and its right, and so it realizes it is the last node of the neighborhood (since $2^0=1$); if the received id is greater than its own id, it sends back a reply message

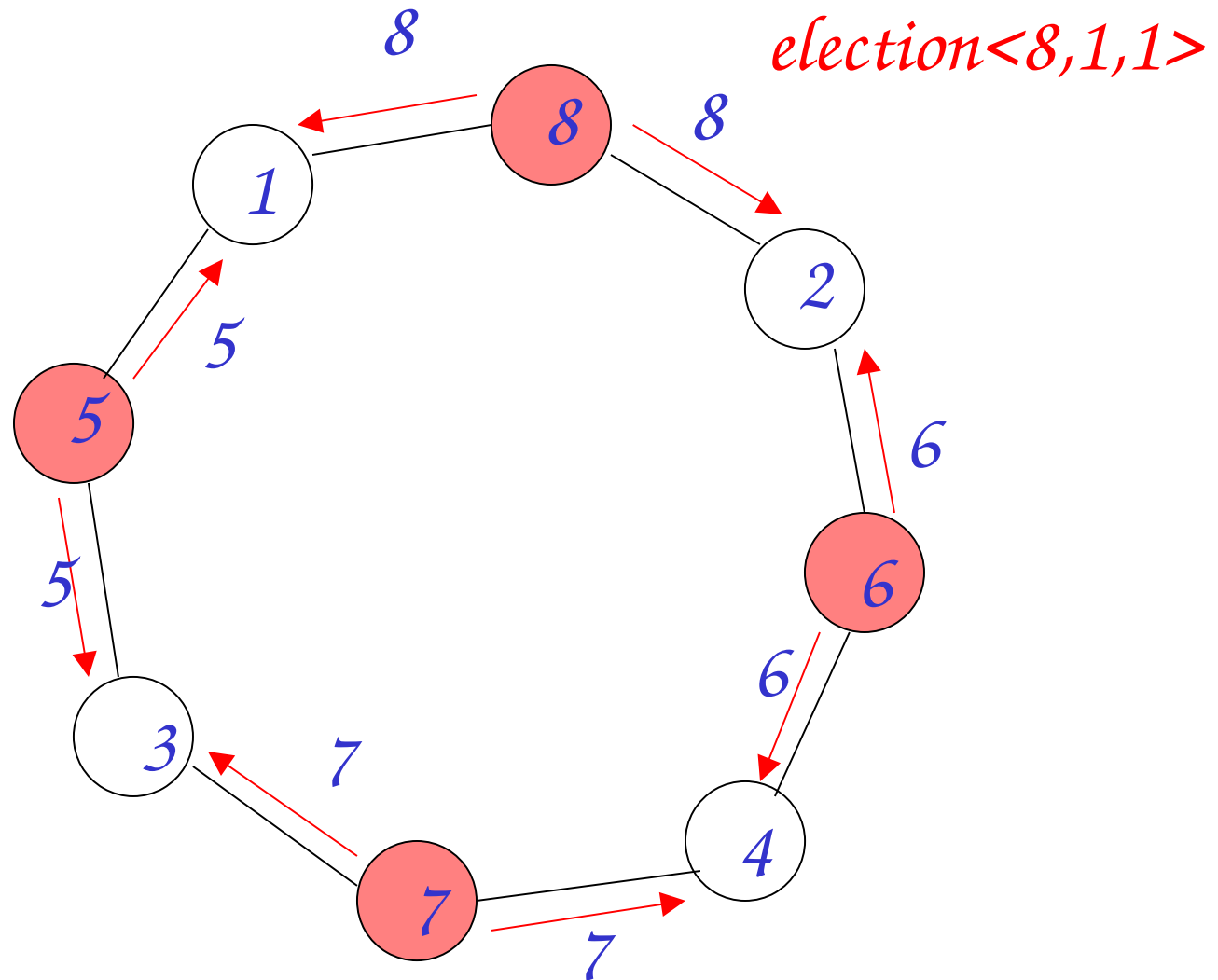


If: a node receives both replies

Then: it becomes a temporary leader and proceeds to the next phase

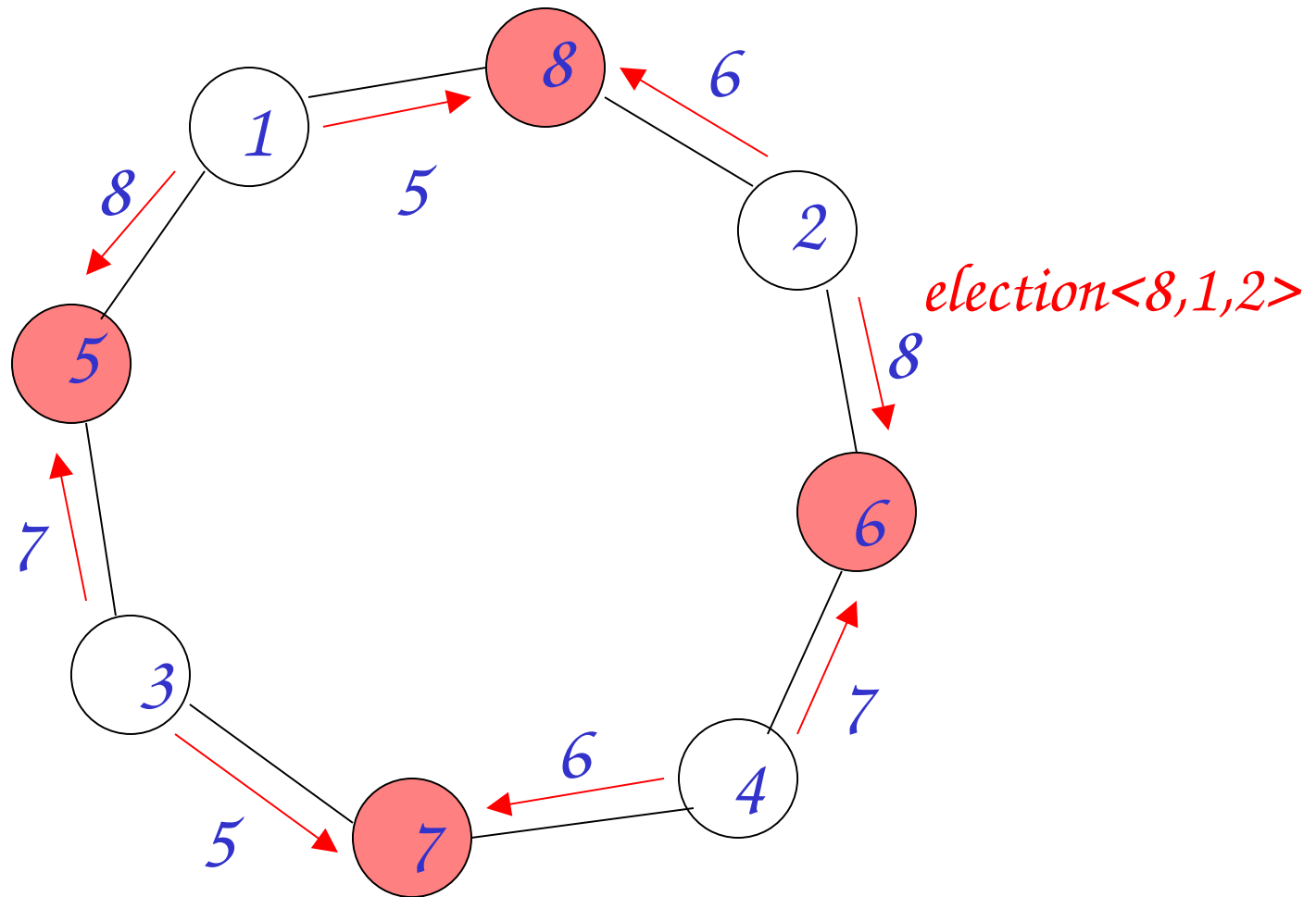


Phase 1: send a probing message $(id, 1, 1)$ to left and right nodes in the 2^1 -neighborhood



If: received id $>$ my own id

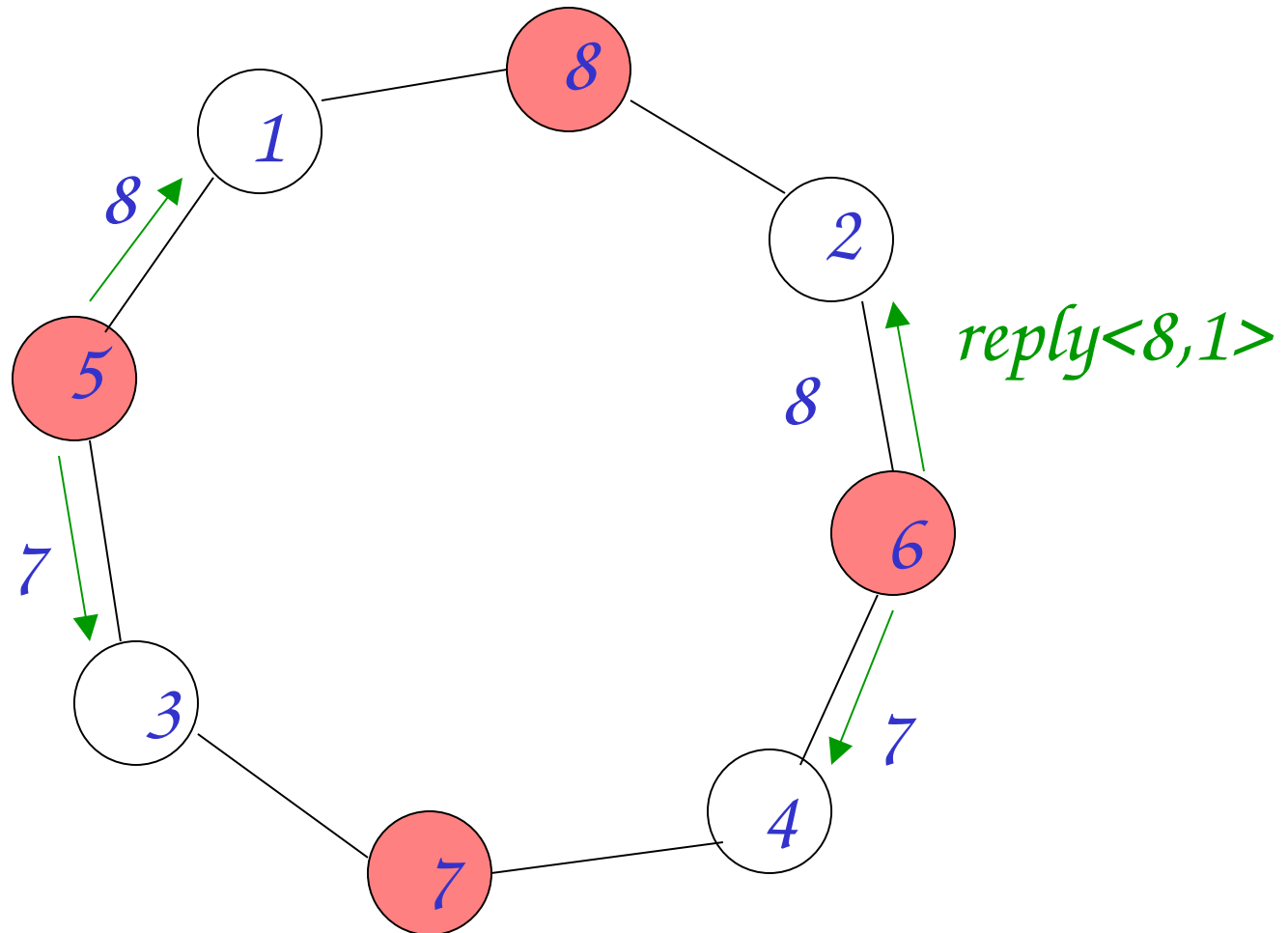
Then: forward the probing message (id,1,2)



At second step: since step counter=2, if a node receive a probing message, it realizes it is on the boundary of the 2-neighborhood

If: received id > my own id

Then: send a reply message

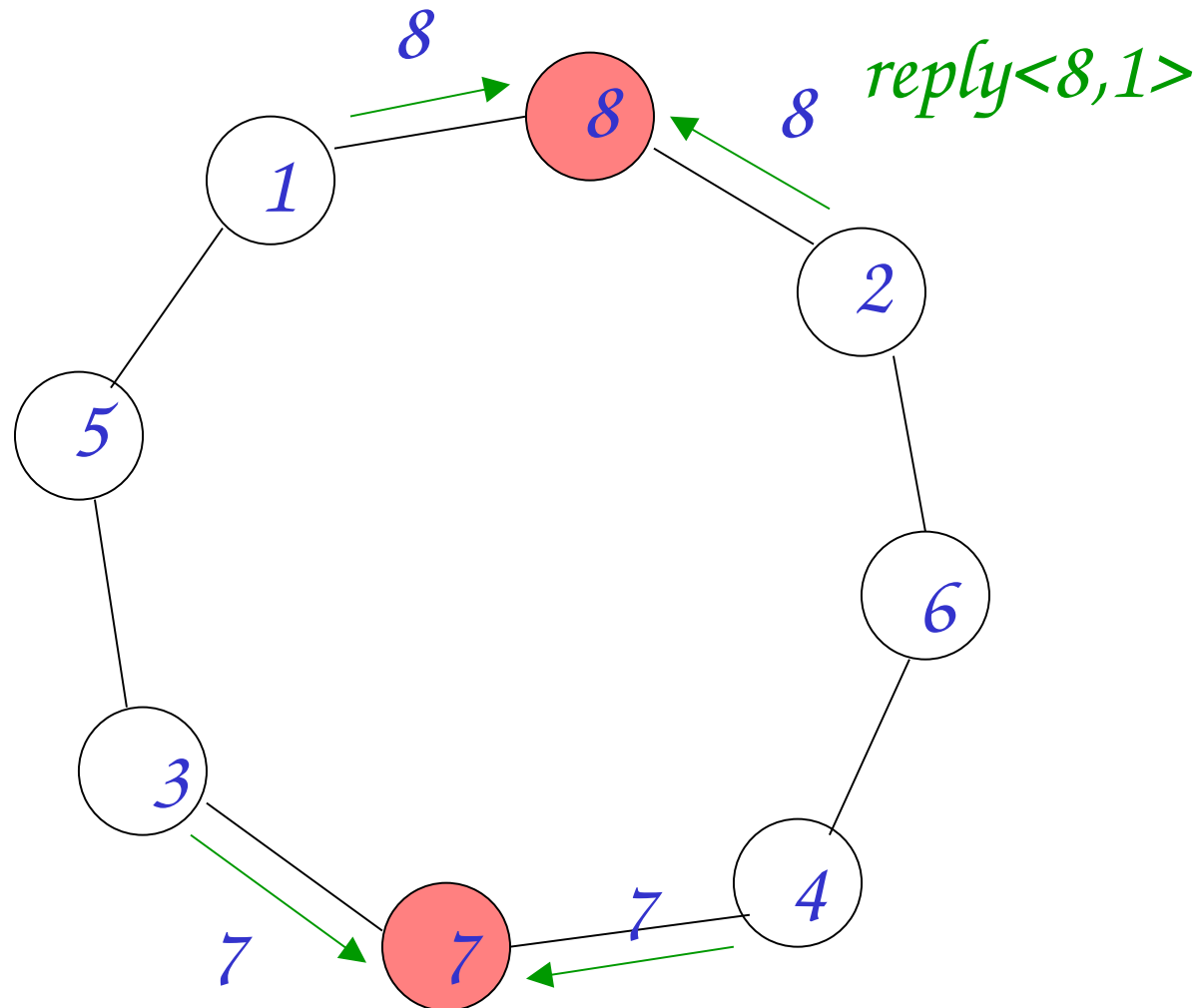


If: a node receives a reply message with another id

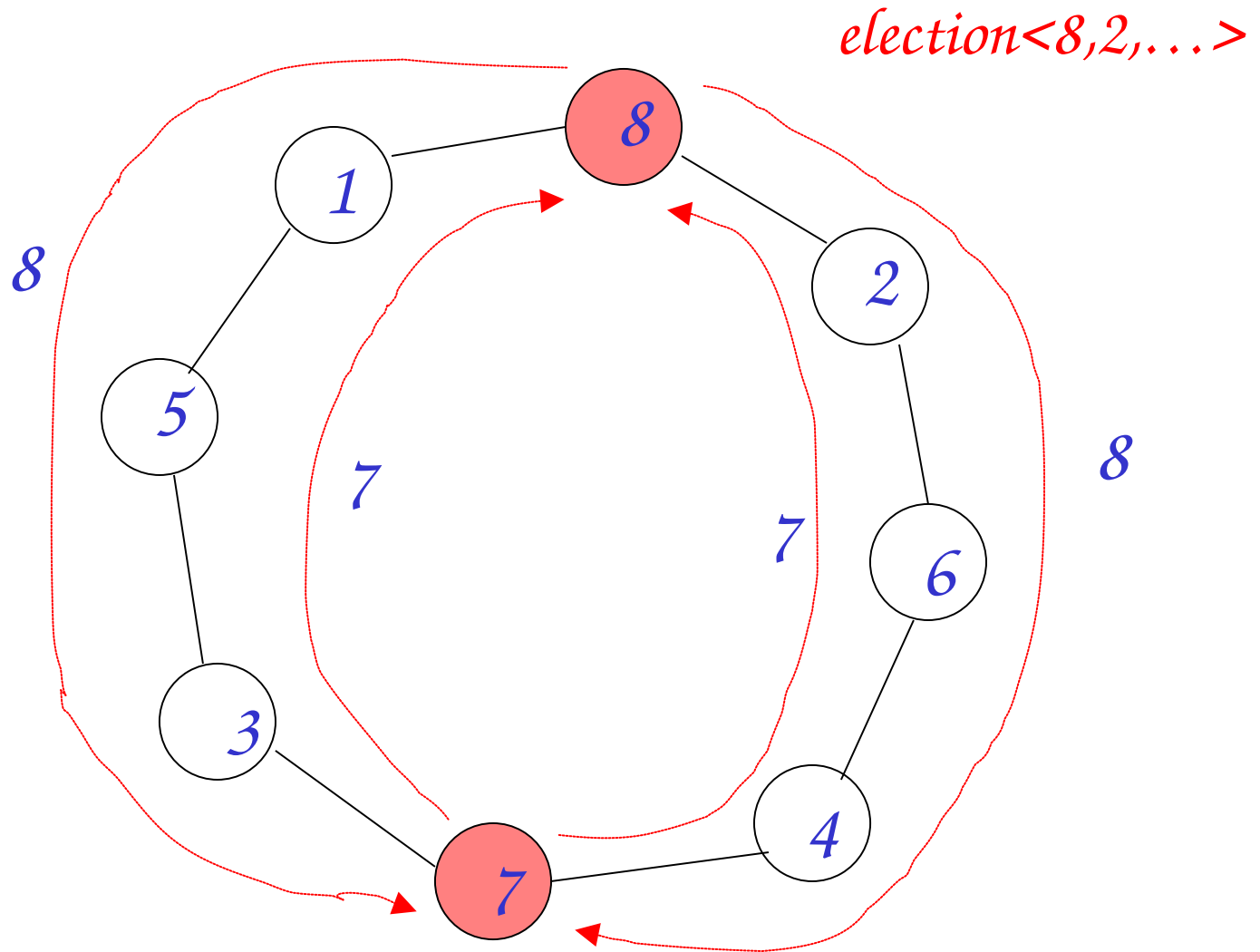
Then: forward it

If: a node receives both replies

Then: it proceed to the next phase



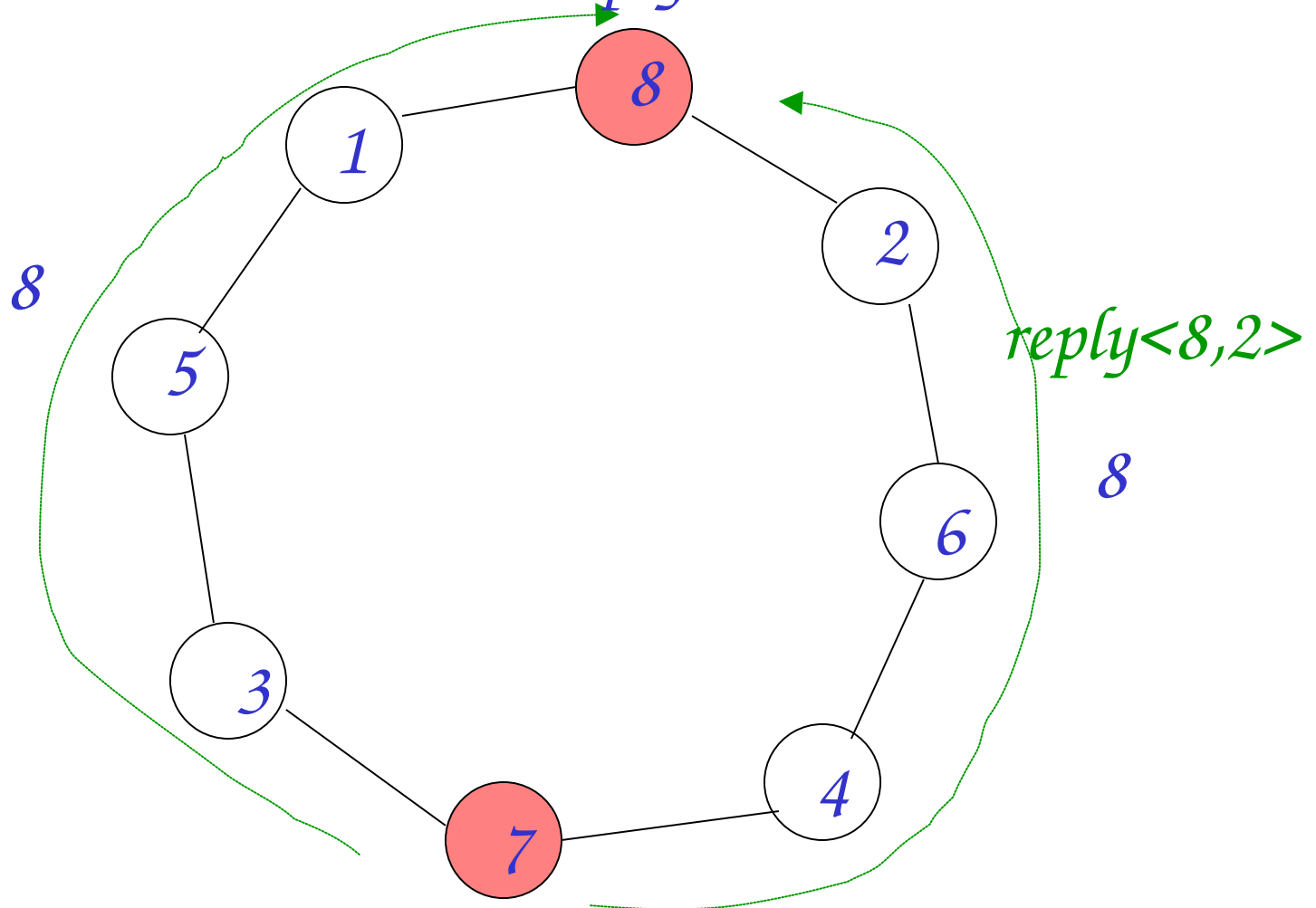
Phase 2: send id to the $2^2=4$ -neighborhood



At the 2² step:

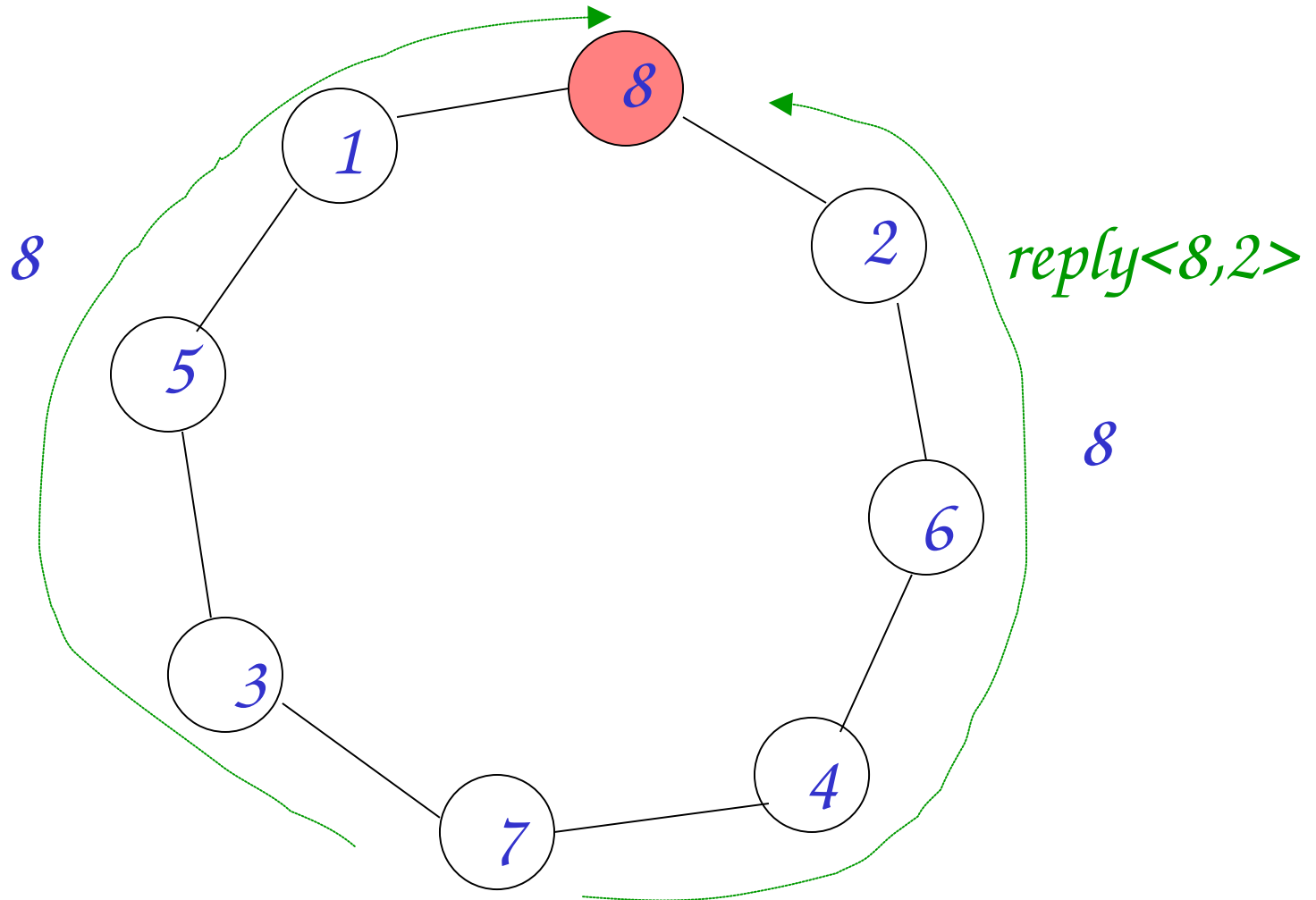
If: received id \geq current id

Then: send a reply



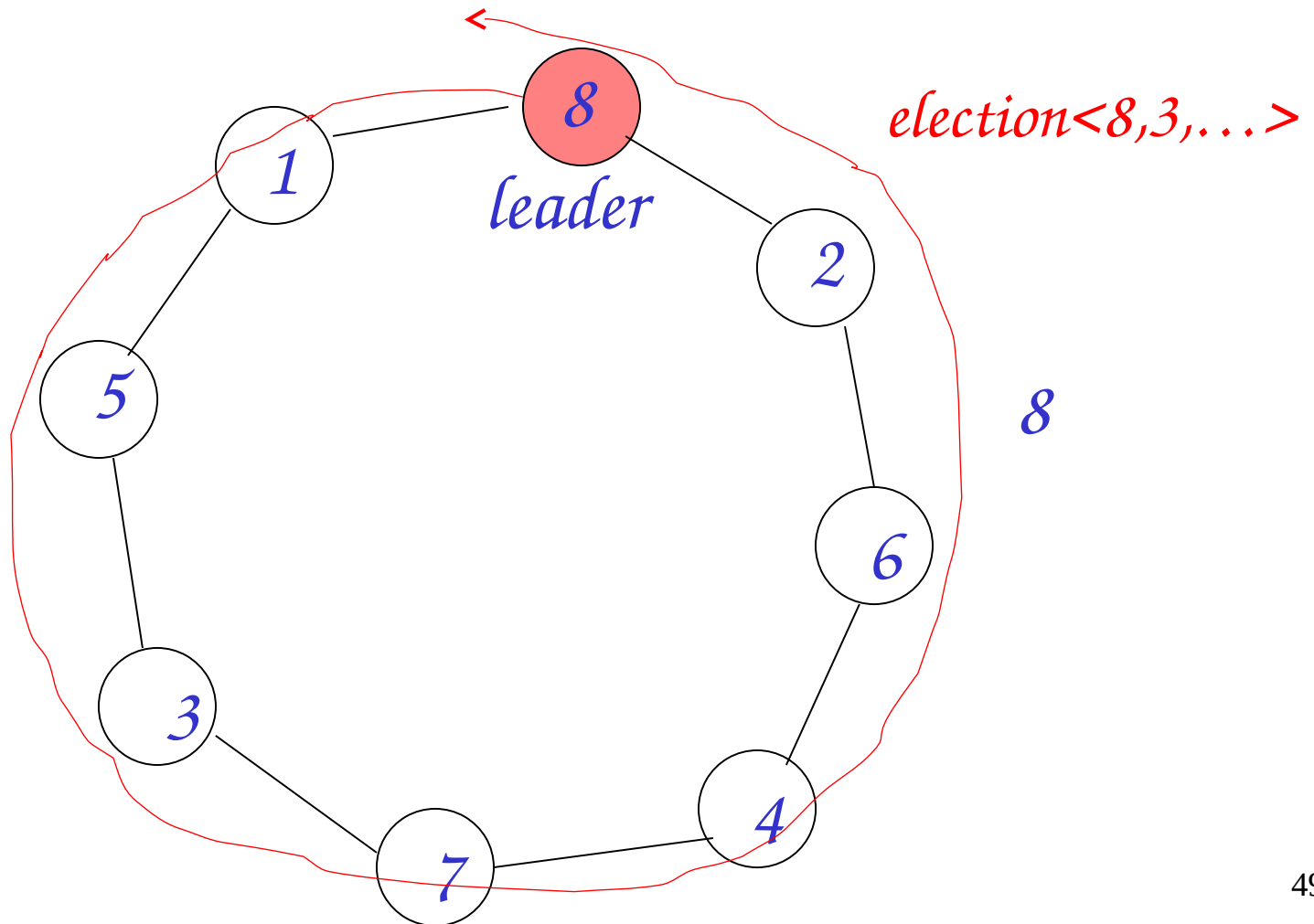
If: a node receives both replies

Then: it becomes temporary leader



Phase 3: send id to $2^3=8$ -neighborhood

*⇒ The node with id 8 will receive its own probe message, and then becomes the **leader**!*

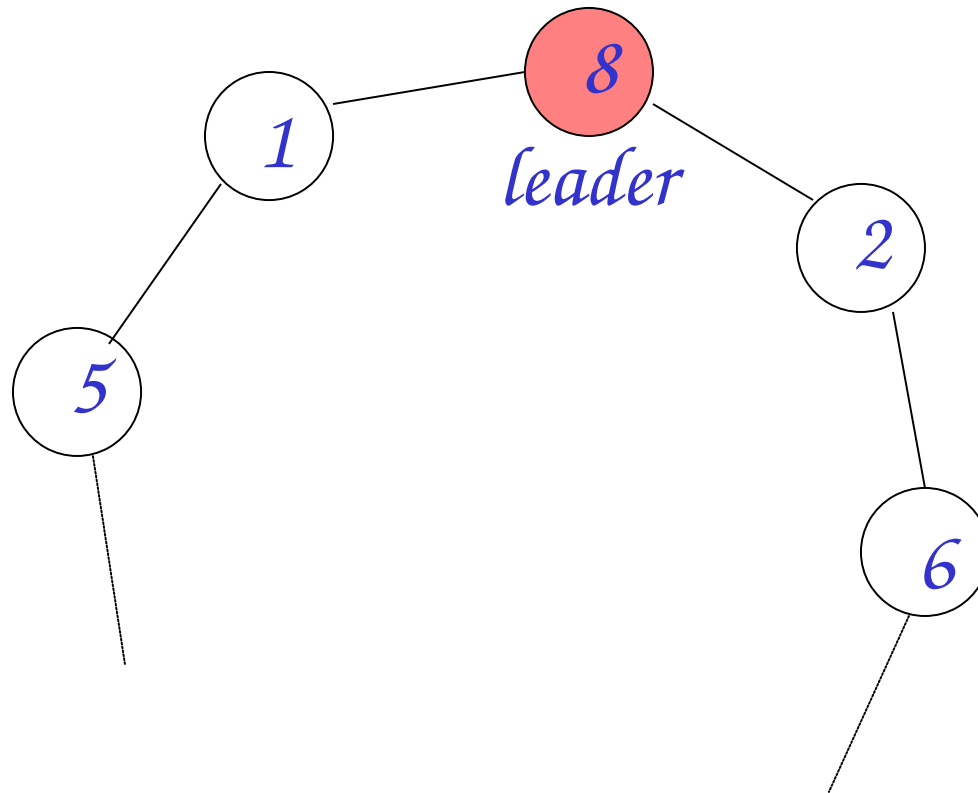


In general:

n nodes



$\Theta(\log n)$ phases



Analysis of HS algorithm

***Correctness:** Similar to CR algorithm (we assume that all processors participate to the election either waking-up spontaneously, or after receiving an ELECTION message – this will guarantee the node with max id will be elected as the leader)*

Message Complexity:

Each msg belongs to a particular phase and is initiated by a particular proc.

Probe distance in phase i is 2^i

*Number of msgs initiated by a processor in phase i is at most $4 \cdot 2^i$
(probes and replies in both directions)*

Message complexity

*Max # messages per each node trying
to become temporary leader*

*Max # nodes trying to become
temporary leader*

Phase 0: 4

n

Phase 1: 8

$n / 2$

...

Phase i : 2^{i+2}

$n / 2^i$

...

Phase $\log n$: $2^{\log n + 2}$

$n / 2^{\log n}$

Message complexity

Max # messages per each node trying
to become temporary leader

Max # nodes trying to become
temporary leader

$$\text{Phase 0: } 4 \quad \times \quad n \quad = 4n$$

$$\text{Phase 1: } 8 \quad \times \quad n / 2 \quad = 4n$$

...

$$\text{Phase } i: 2^{i+2} \quad \times \quad n / 2^i \quad = 4n$$

...

$$\text{Phase } \log n: 2^{\log n + 2} \quad \times \quad n / 2^{\log n} \quad = 4n$$

Total messages: $O(n \cdot \log n)$

Can we do better?

- *The $O(n \log n)$ algorithm is more complicated than the $O(n^2)$ algorithm but uses fewer messages in the worst case.*
- *It works in both the synchronous and the **asynchronous** case (and no **synchronized start** is required), and it is **uniform***
- *Can we reduce the number of messages even more? Not in the very same model:*

***Thr:** Any **asynchronous uniform** LE algorithm on a ring requires $\Omega(n \log n)$ messages.*

Questions:

- 1. What about a best case for HS?*
- 2. Can you see an instance of HS which will use $\Theta(n \log n)$ messages?*
- 3. Do we need the left/right notion in a processor when running HS?*
- 4. What about a variant of HS in which probing messages are sent only along one direction (for instance, on the left side)?*

Solution of the homework (1/3)

1. What about a best case for HS? Imagine a situation in which ids are given in increasing order:

Message complexity

Max # messages per each node trying to become temporary leader		Max # nodes trying to become temporary leader	Total # of messages
Phase 0: 4	\times	n	$= 4n$
Phase 1: 8	\times	1	$= 8 = (4n)/(n/2)$
...			
Phase i : 2^{i+2}	\times	1	$= 2^{i+2} = (4n)/(n/2^i)$
...			
Phase $\log n$: $2^{\log n + 2}$	\times	1	$= 2^{\log n + 2} = (4n)/(n/2^{\log n})$

Total messages: $4n + 4(1/2 + 1/4 + \dots + n/2 + n) \approx 4n + 8n = \Theta(n)$

Solution of the homework (2/3)

2. Can you see an instance of HS which will use $\Theta(n \log n)$ messages? Yes, the one we gave in our example!

3. Do we need the left/right notion in a processor when running HS? No, definitively!

4. What about a variant of HS in which probing messages are sent only along one direction (for instance, on the left side)? No problem, but nodes passing a phase may be more now... check it out how many messages can be generated in the worst case:

Solution of the homework (3/3)

Imagine a situation in which ids are given in decreasing order: then, at the end of Phase 0, only one node will die; at the end of Phase 1, again only one node will die; at the end of Phase 2, 2 nodes will die; at the end of phase 3, 2^2 nodes will die; in general; at the end of Phase i , 2^{i-1} nodes will die. Then:

Message complexity

Max # messages per each node trying to become temporary leader		Max # nodes trying to become temporary leader	Total # of messages
Phase 0: 2	×	n	$= 2n$
Phase 1: 4	×	$n-1$	$= 4(n-1)$
...			
Phase i : 2^{i+1}	×	$n-2^{i-2}-\dots-1-1=n-2^{i-1}$	$= 2^{i+1} (n-2^{i-1})$
...			
Phase $\log n$: $2^{\log n+1} = 2n$	×	$n/2$	$= n^2$

Total messages: $2n + 4(n-1) + \dots + n^2 = \Theta(n^2)$.

A $\Theta(n)$ -messages *Synchronous* Algorithm

Requirements: n must be known (i.e., it is *non-uniform*), and all the processors must *start together at the very beginning* (this assumption could be easily relaxed)

Reminder: At each *round* each processor, in order:

- Reads the incoming messages buffer;
- Makes some internal computations;
- Sends messages which will be read in the next round.

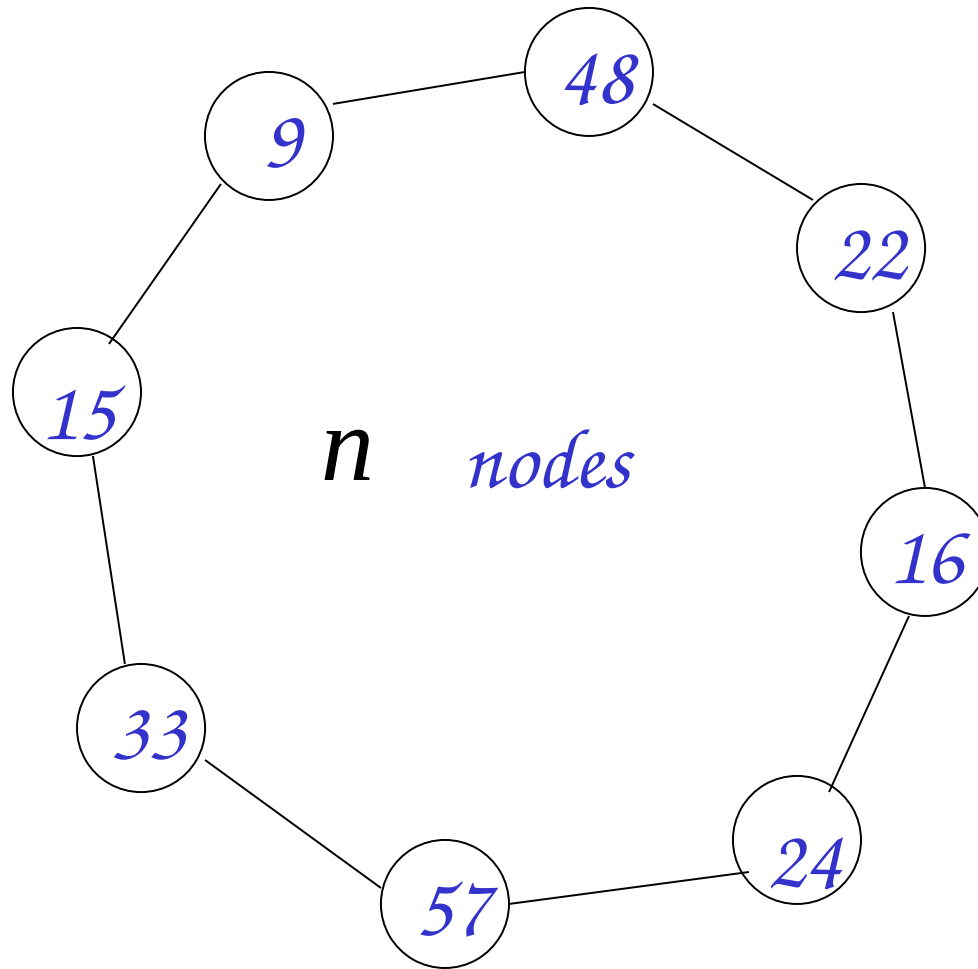
*Rounds are grouped in **phases**: each phase consists of n rounds:*

If at the beginning of phase $k=0,1,\dots$ there is a node with id k

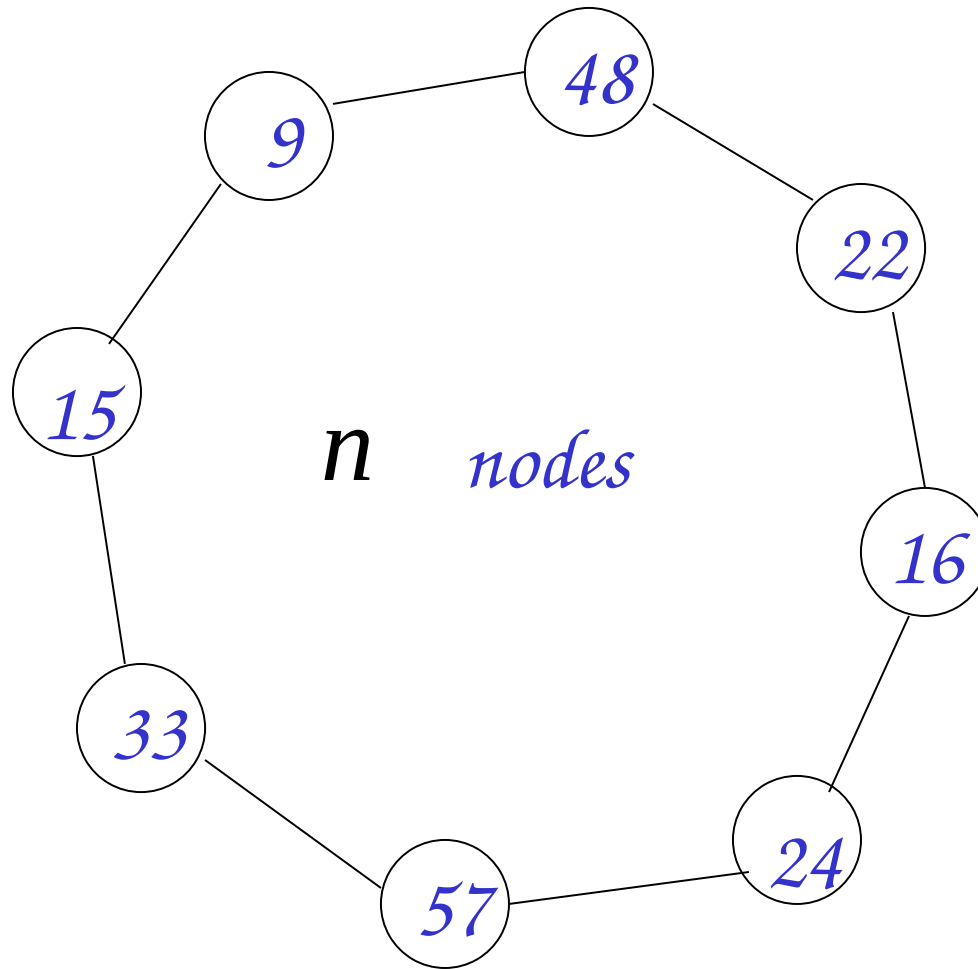
- *it elects itself as the **leader**;*
- *it **notifies** all the other nodes it became the leader;*
- *the algorithm terminates.*

Remark: The node with smallest id is elected leader

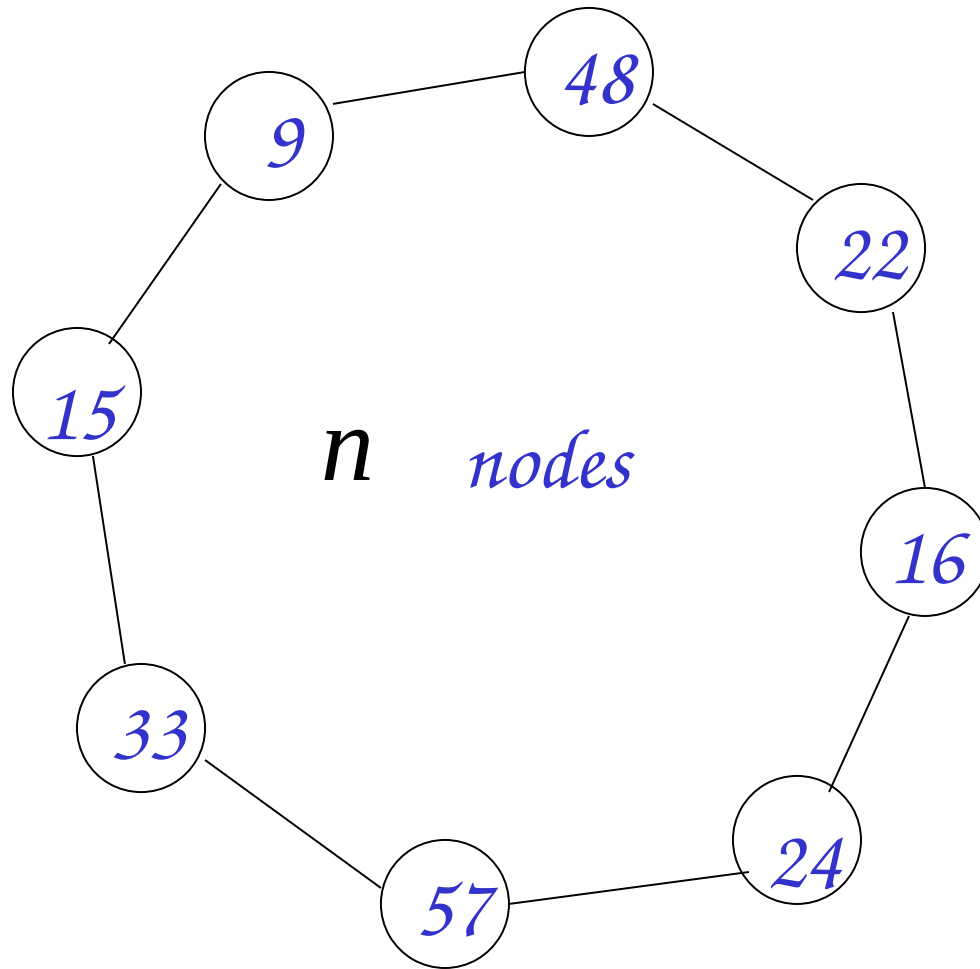
Phase 0 (rounds 1, 2, ..., n=8): no message sent



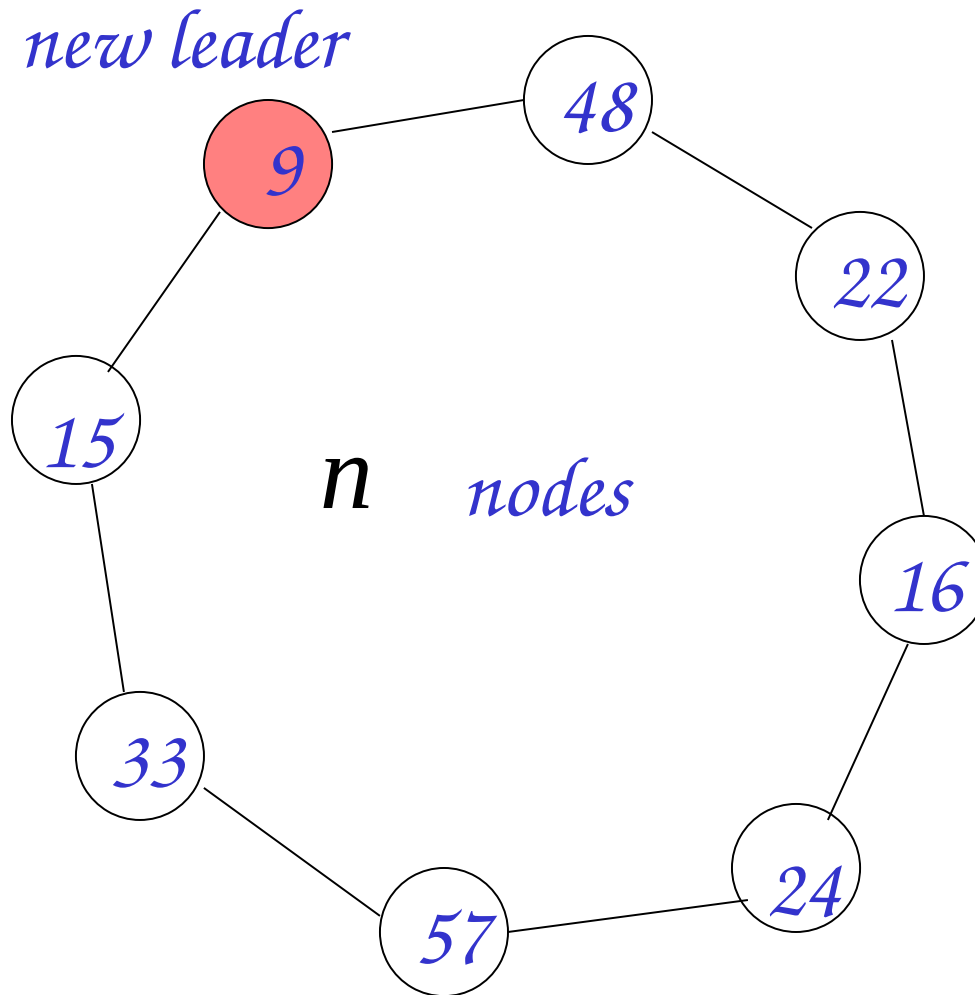
Phase 1 (rounds $n+1=9, \dots, 2n=16$): no message sent



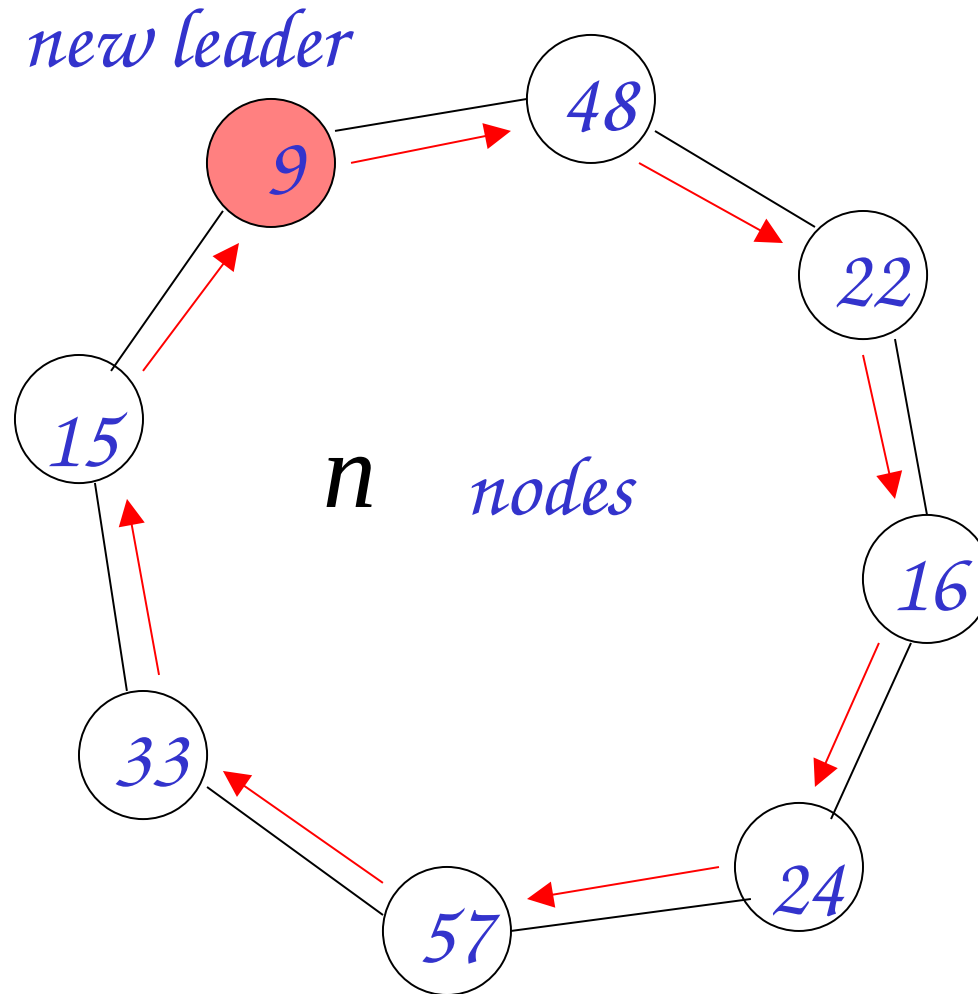
...Phase 8 (rounds $8 * 8 + 1 = 65, \dots, 8 * 9 = 72$): *no message sent*



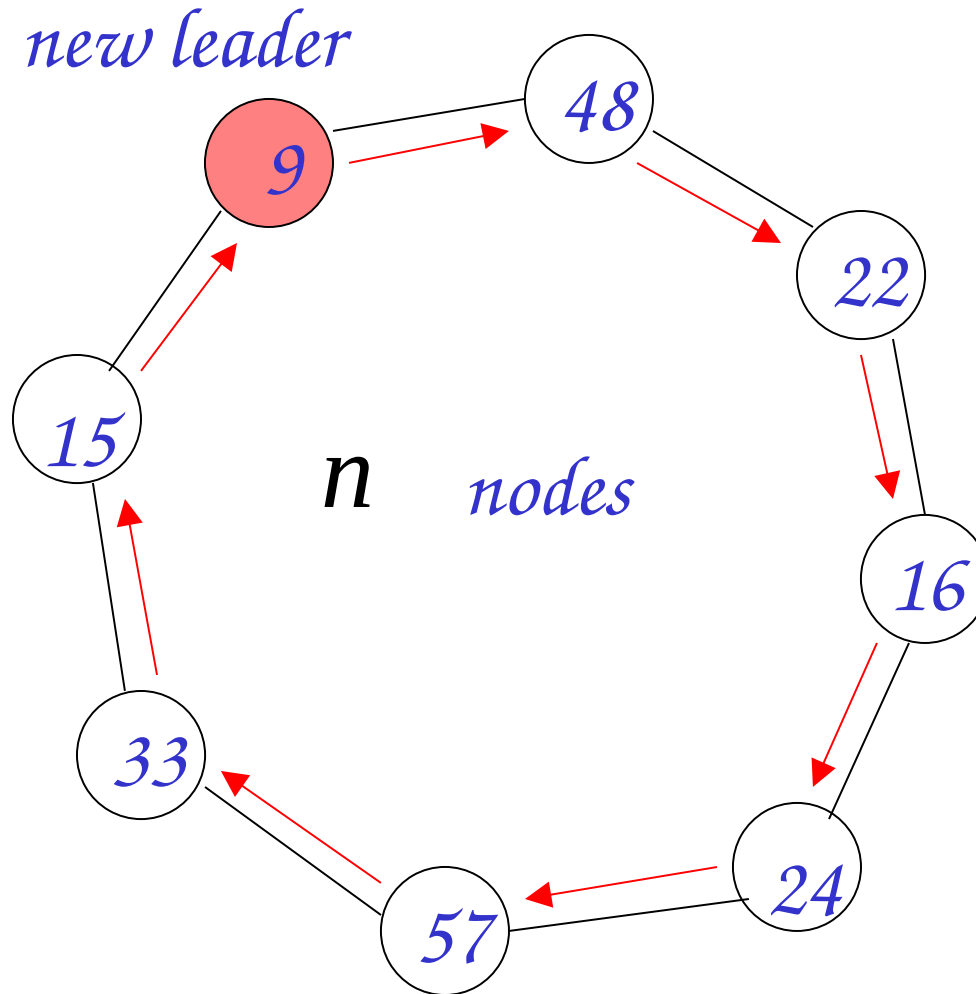
Phase $k=9$: at the beginning of round 73 (which is equal to $k \cdot n + 1 = 9 \cdot 8 + 1$), node with $id=9$ realizes phase 9 has started, so it elect itself as the leader, by announcing it with a clock-wise message (that will take the next $n-1=7$ rounds to reach all the nodes in the ring)



Phase 9 (n rounds): n messages sent

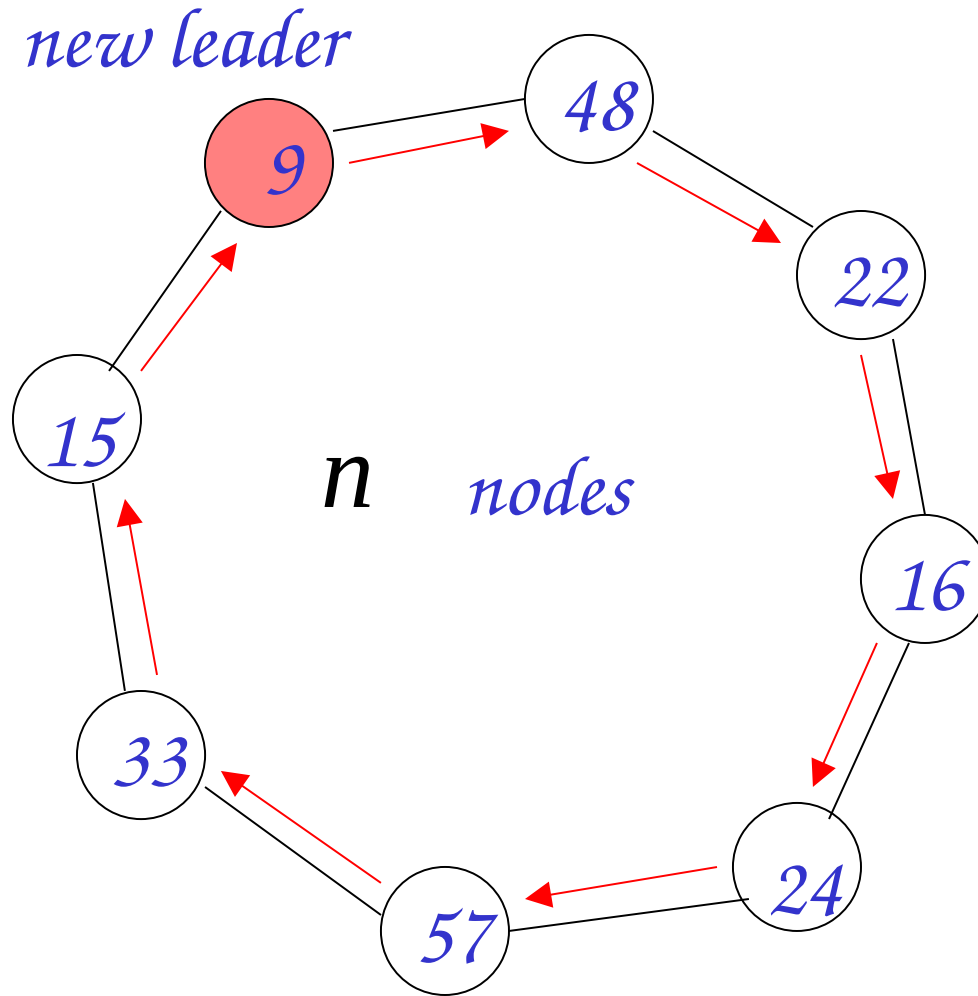


Phase 9 (n rounds): n messages sent



Algorithm Terminates

Phase 9 (n rounds): n messages sent



Total number of messages: n (Question: can we reduce it to $n-1$?)

Algorithm analysis

Correctness: Easy to see

Message complexity: $n = \Theta(n)$, which can be shown to be optimal

*Time complexity (# rounds): $\Theta(n \cdot \mathcal{L})$, where \mathcal{L} is the leader (i.e., smallest) id in the ring \Rightarrow not bounded by any function of n
 \Rightarrow it is not **strongly polynomial** in n . Notice however that it is commonly assumed that $\mathcal{L} = O(n^k)$, $k = O(1)$*

Other disadvantages:

- Requires synchronous start (not really!)*
- Requires knowing n (**non-uniform**)*

Homework

Show that the algorithm can be modified in order to be correct also if not all the processors wake up at round 1.

Solution

It suffices to notice that since the system is non-uniform, then if a processor with id x wakes up at round $k > 1$, then it will simply compute $(k-1)/n$, and the following cases are possible:

- 1. $(k-1)/n$ returns a remainder **equal to 0**, and $\lfloor k/n \rfloor = x$: in this case, the processor immediately **elects itself as the leader**;*
- 2. $(k-1)/n$ returns a remainder other than 0, and $\lfloor k/n \rfloor < x$: in this case, the processor starts participating to the election process;*
- 3. $(k-1)/n$ returns a remainder other than 0, and $\lfloor k/n \rfloor \geq x$: in this case, the processor **does not** participate to the election process.*

*A **uniform** $\Theta(n)$ -messages Synchronous Algorithm: the Frederickson & Lynch algorithm (1987)*

*Works in a **weaker** model than the previous synchronous algorithm:*

- **uniform** (does not rely on knowing n)*
- processors **need not start at the same round**; a processor either wakes up spontaneously or when it first gets a message*
- IDEA: messages travel at different “speeds” (the leader’s one is the fastest)*

***Reminder:** At each **round** each processor, in order:*

- Reads the incoming messages buffer;*
- Makes some internal computations;*
- Sends messages which will be read in the next round.*

Slow and fast messages

- A processor that wakes up spontaneously is **active**; sends its id in a **fast** message (one edge per round) in a clockwise direction
- A processor that wakes up when receiving a msg is **relay**; **it does not enter ever in the competition to become leader**
- A processor only forwards a message whose id is **smaller** than **any other competing id** it has **seen so far**, including its own one in case it is active (notice this is different from CR algorithm)
- A **fast** message carrying id **m** that reaches an **active** processor becomes **slow**: it starts traveling at **one edge** every **2^m rounds** (i.e., a processor that receives it at round **r**, i.e., it was sent at round **r-1**, will forward it at round **$r+2^m-1$**)
- If a processor gets its own id back, it elects itself as **leader**

Algorithm analysis

***Correctness:** convince yourself that the active processor with smallest id is elected.*

***Message complexity:** Winner's msg is the fastest. While it traverses the ring, other messages are slower, so they are overtaken and stopped before too many messages are sent.*

Message Complexity

A message will contain 2 fields: (id, 0/1 (slow/fast))

Divide msgs into four types:

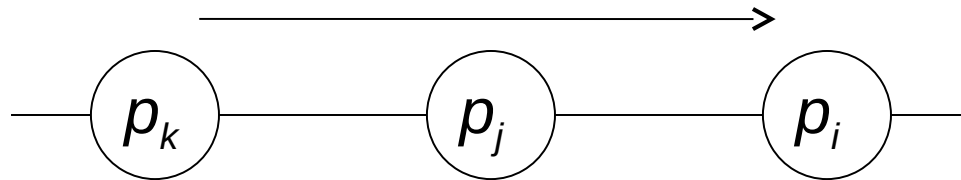
- 1. fast msgs*
- 2. slow msgs sent while the leader's msg is fast*
- 3. slow msgs sent while the leader's msg is slow*
- 4. slow msgs sent while the leader is sleeping*

Next, count the number of each type of msg.

Number of Type 1 Messages

(fast messages)

Show that no processor forwards more than one fast msg (by contradiction):



Suppose p_i forwards p_j 's fast msg and p_k 's fast msg. But when p_k 's fast msg arrives at p_j :

1. either p_j has already sent its fast msg, so p_k 's msg becomes slow (contradiction)
2. p_j has not already sent its fast msg, so it never will (contradiction) since it is a **relay**

Number of type 1 msgs is $O(n)$.

Number of Type 2 Messages

(slow msgs sent while leader's msg is fast)

Leader's msg is fast for at most n rounds

by then it would have returned to leader

Slow msg i is forwarded $n/2^i$ times in n rounds

Max. number of msgs is when ids are as small as possible (0 to $n-1$ and leader is 0)

Number of type 2 msgs is at most

$$\sum_{i=1}^{n-1} n/2^i \leq n$$

Number of Type 3 Messages

(slow msgs sent while leader's msg is slow)

*Maximum number of rounds during which leader's msg is slow is $n \cdot 2^{\mathcal{L}}$
(\mathcal{L} is leader's id).*

No msgs are sent once leader's msg has returned to leader

Slow msg i is forwarded $n \cdot 2^{\mathcal{L}} / 2^i$ times during $n \cdot 2^{\mathcal{L}}$ rounds.

Worst case is when ids are \mathcal{L} to $\mathcal{L} + n - 1$ (independently of \mathcal{L} , and so in particular, when $\mathcal{L} = 0$)

Number of type 3 msgs is at most

$$\sum_{\substack{i=\mathcal{L} \\ \mathcal{L}+n-1}} n \cdot 2^{\mathcal{L}} / 2^i = \sum_{\substack{i=0 \\ n-1}} n / 2^i \leq 2n$$

Number of Type 4 Messages

(slow messages sent while leader is sleeping)

Claim: Leader sleeps for less than n rounds.

Proof: Indeed, it can be shown that the leader will awake after at most $k \leq n$ rounds, where k is the distance in the ring between the leader and the closest counter-clockwise active processor which woke-up at round 1 (prove by yourself by using induction)

- Slow message i is forwarded $n/2^i$ times in n rounds
- Max. number of messages is when ids are as small as possible (0 to $n-1$ and leader is 0)
- Number of type 4 messages is at most

$$\sum_{i=1}^{n-1} n/2^i \leq n$$

Total Number of Messages

We showed that:

number of type 1 msgs is at most n

number of type 2 msgs is at most n

number of type 3 msgs is at most $2n$

number of type 4 msgs is at most n

\Rightarrow total number of msgs is at most $5n=O(n)$, and of course is at least n , and so the message complexity is $\Theta(n)$

Time Complexity

*Running time is $O(n \cdot 2^{\mathcal{L}})$, where \mathcal{L} is the leader (i.e., smallest) id.
Even worse than previous algorithm, which was $O(n \cdot \mathcal{L})$. This algorithm is **polynomial** in n only if we assume that the smallest identifier is $O(\log n)$ (which is realistic, though)*

\Rightarrow The advantage of having a linear number of messages is paid by both the synchronous algorithms with a number of rounds which depends on the minimum id

Summary of \mathcal{LE} algorithms on rings

- *Anonymous rings: no any algorithm*
- *Non-anonymous asynchronous rings:*
 - $O(n^2)$ algorithm (unidirectional rings)
 - $O(n \log n)$ messages (*optimal*, bidirectional rings)
- *Non-anonymous synchronous rings:*
 - $\Theta(n)$ messages (*optimal*), $O(n \cdot \mathcal{L})$ rounds (non-uniform)
 - $\Theta(n)$ messages (*optimal*), $O(n \cdot 2^{\mathcal{L}})$ rounds (uniform)

\mathcal{LE} algorithms on general topologies

INPUT: a MPS $G=(\mathcal{V},\mathcal{E})$ with $|\mathcal{V}|=n$ and $|\mathcal{E}|=m$

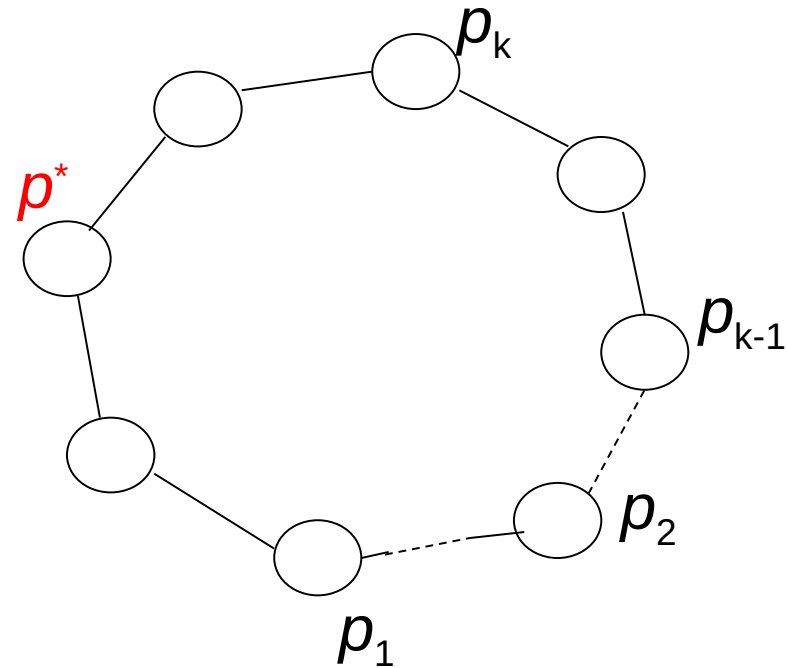
- *Anonymous: no any algorithm (of course...)*
- *Non-anonymous asynchronous systems:*
 - *$O(m+n \log n)$ messages*
- *Non-anonymous synchronous systems:*
 - *$O(m+n \log n)$ messages, $O(n \log n)$ rounds*
(notice that this could be used on a ring by returning an $O(n \log n)$ messages and rounds algo; can you guess whether we have already given –at least implicitly- such a result?)
- *Homework: think to complete graphs...*

Proof of the Claim that the leader sleeps for less than n rounds.

Proof: Let p^* denote the leader, and let p_1, p_2, \dots, p_k be the processors which are active when p^* wakes up, in counter-clockwise order w.r.t. p^* .

Let $r(p) \geq 1$ be the round at which an active processor p wakes up. We show by induction that

$r(p^*) < r(p_i) + |p_i p^*|$, for $i=1, \dots, k$ where $|p_i p^*| < n$ is the clockwise distance (i.e., number of edges) separating p_i and p^* in clockwise order in the ring.

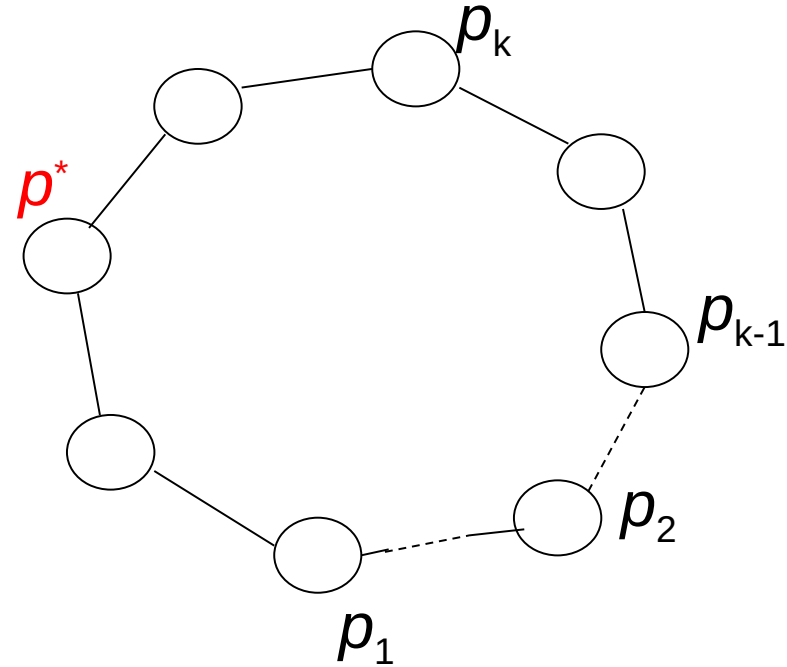


Proof of the Claim (follows)

- $i=1$: trivially, $r(\mathbf{p}^*) < r(p_1) + |p_1 \mathbf{p}^*|$, since otherwise p^* cannot be active;
- Assume that $r(\mathbf{p}^*) < r(p_{i-1}) + |p_{i-1} \mathbf{p}^*|$; we have to show that $r(\mathbf{p}^*) < r(p_i) + |p_i \mathbf{p}^*|$; but trivially

$$r(p_{i-1}) < r(p_i) + |p_i p_{i-1}|$$

since otherwise p_{i-1} cannot be active; then, by the inductive hypothesis

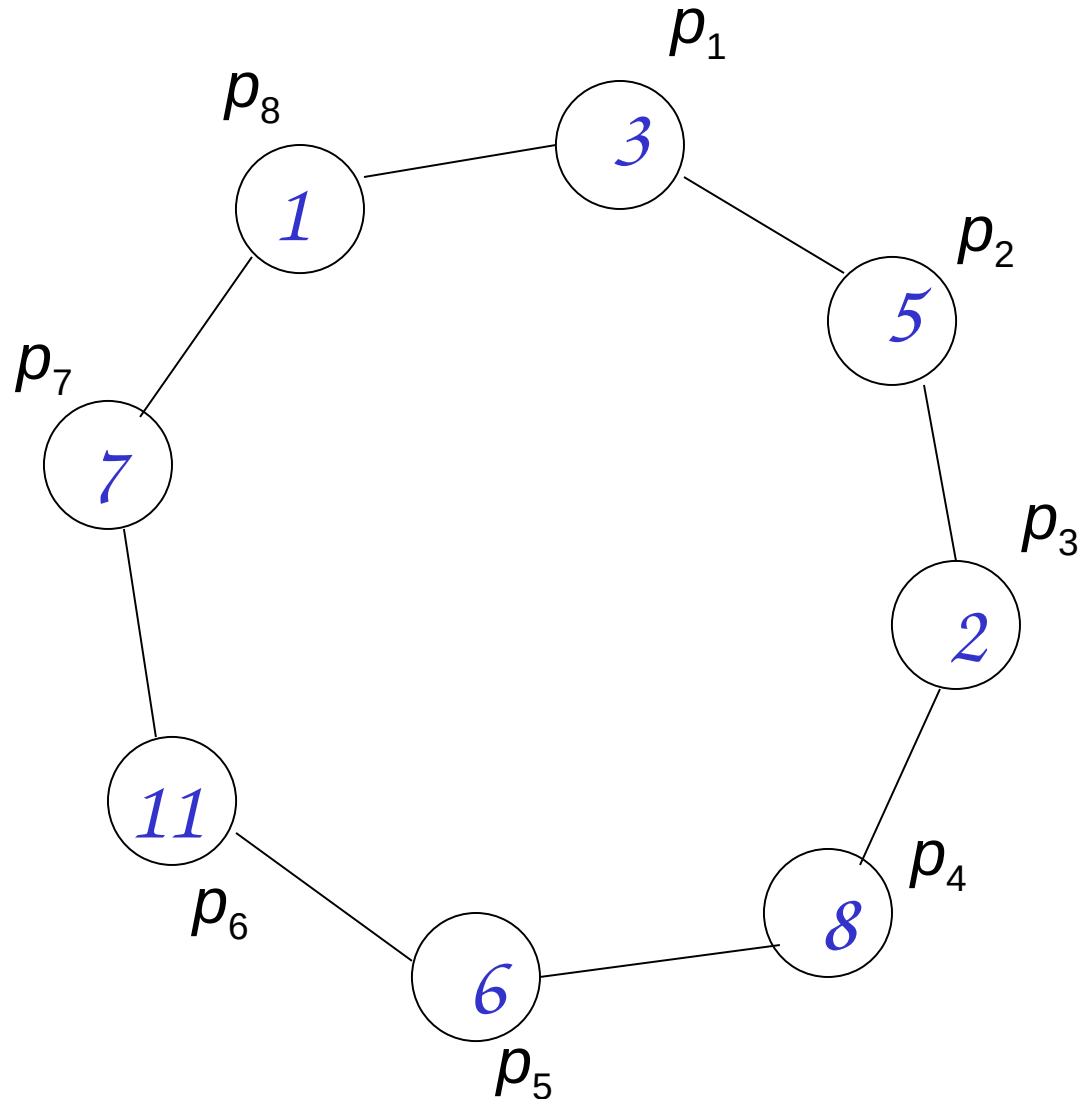


$$r(\mathbf{p}^*) < r(p_{i-1}) + |p_{i-1} \mathbf{p}^*| < r(p_i) + |p_i p_{i-1}| + |p_{i-1} \mathbf{p}^*| = r(p_i) + |p_i \mathbf{p}^*|.$$

Thus, since there is at least an active processor in p_1, p_2, \dots, p_k that woke up at round 1, say p_j , and since $|p_j \mathbf{p}^*| \leq n-1$, it follows that

$$r(\mathbf{p}^*) < r(p_j) + |p_j \mathbf{p}^*| \leq n.$$

Homework: Write the pseudo-code and execute the slow-fast algorithm on the following ring, assuming that p_1, p_5, p_8 will awake at round 1, and p_3 will awake at round 2.



Pseudocode

```

TYPE MSG/
    int ID
    boolean SPEED // 0=SLOW; 1=FAST

PROGRAM MAIN{//Start at any round either spontaneously or
    after receiving a message
    STATE:=Non_Leader
    SMALLER_ID:=+∞
    R:= current round //taken from the universal clock
    IF(IN_BUFFER=Empty){
        SMALLER_ID:=MY_ID
        MSG.ID:=MY_ID
        MSG.SPEED:=1
        SEND(MSG)
        REPEAT(ACTIVE_CASE)
    } ELSE REPEAT(RELAY_CASE)
}
    
```

PROCEDURE ACTIVE_CASE{//This is repeated in any round following the waking-up round

```

    R:= current round
    IF(IN_BUFFER=Non-Empty){
        RECEIVE(MSG) //This makes the IN_BUFFER empty
        IF(MSG.ID=MY_ID){
            STATE:=Leader
            EXIT}
        IF(MSG.ID < SMALLER_ID){
            SMALLER_ID:=MSG.ID
            TIMEOUT:=R+(2^MSG.ID)-1
            MSG.SPEED:=0;
            OUT_BUFFER:=MSG //The buffer may be overwritten
        }
        IF(R=TIMEOUT) SEND(OUT_BUFFER)
    }
    
```

PROCEDURE RELAY_CASE{//This is repeated in any round since the waking-up round

```

    R:= current round
    IF(IN_BUFFER=Non-Empty){
        RECEIVE(MSG) //This makes the IN_BUFFER empty
        IF(MSG.ID < SMALLER_ID){
            SMALLER_ID:=MSG.ID
            OUT_BUFFER:=MSG
            IF(MSG.SPEED=1) TIMEOUT:=R
            ELSE TIMEOUT:=R+(2^MSG.ID)-1}}
        IF(R=TIMEOUT) SEND(OUT_BUFFER)
    }
    
```