Leader Election

Chapter 3

Observations

Election in the Ring

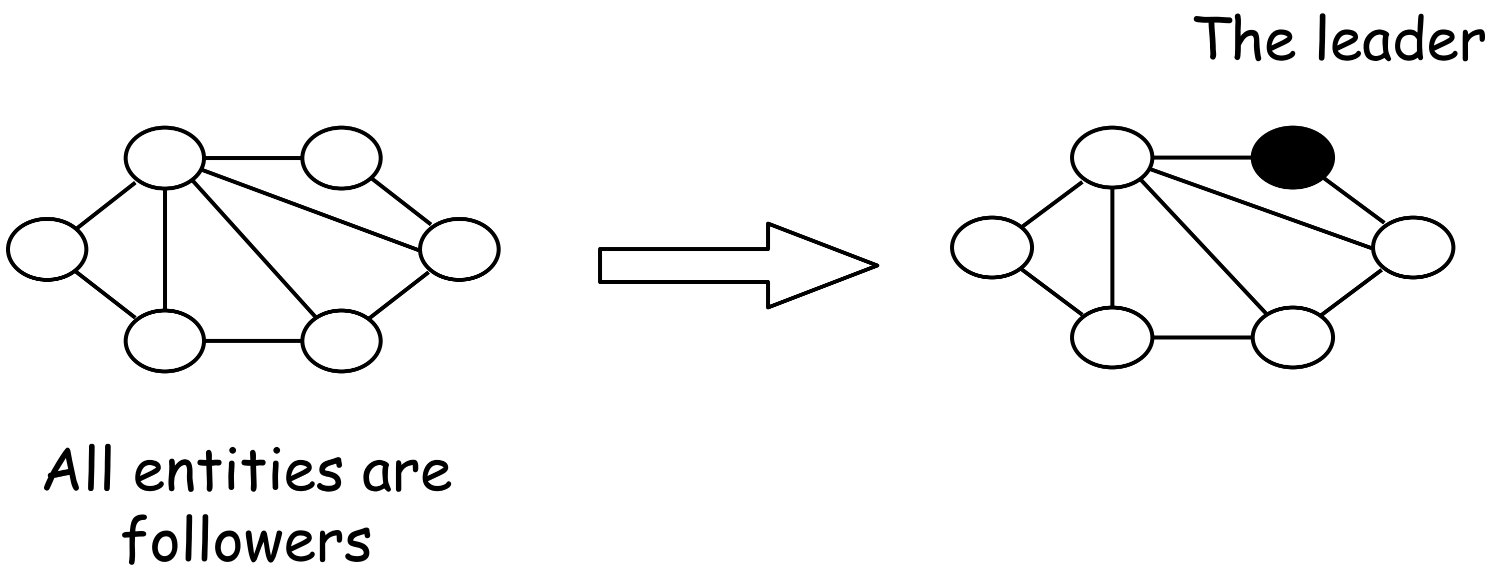
Election in the Mesh

Election in an arbitrary graph

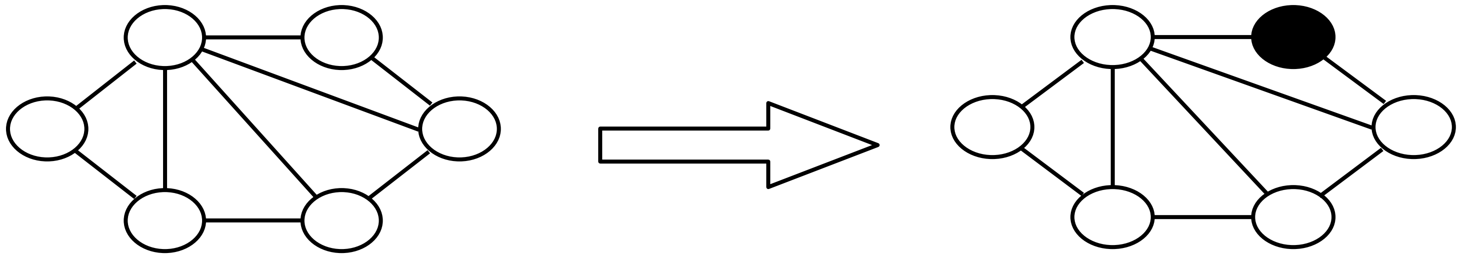
Election

The problem

Move the system from an initial configuration where all entities are in the same state into a final configuration where all entities are in the same state except one, namely the leader.



Election: Impossibility Result



Under the standard restrictions **R** (BL, CN, TR)

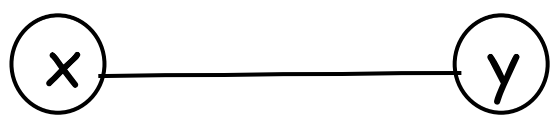
Theorem
The election problem cannot be generally solved if the entities do not have different identities.

Election: Impossibility Result

Under the standard restrictions **R** (BL, CN, TR)

Proof (Sketch)
By contradiction.

- Consider a system with two entities, x and y, in the same initial state
- If P solves the problem, it works in any conditions and delays
- Take a scenario synchronous and where x and y start simultaneously
- They have the same rules, thus they perform the same steps
- Thus, at each moment, they are doing the same thing
- If one becomes leader, even the other one (there cannot be a unique leader)



Election: Additional Restriction

We need to extend the standard restrictions **R** (BL, CN, TR)

The origin of the problem
Entities have all the same behaviour (behaviroural symmetry)
- same set of rules
- same initial state

The idea
Break the symmetry making each entity unique (in same way)

Initial distinct values
Each entity has a unique identifier

Election: Solution Strategy

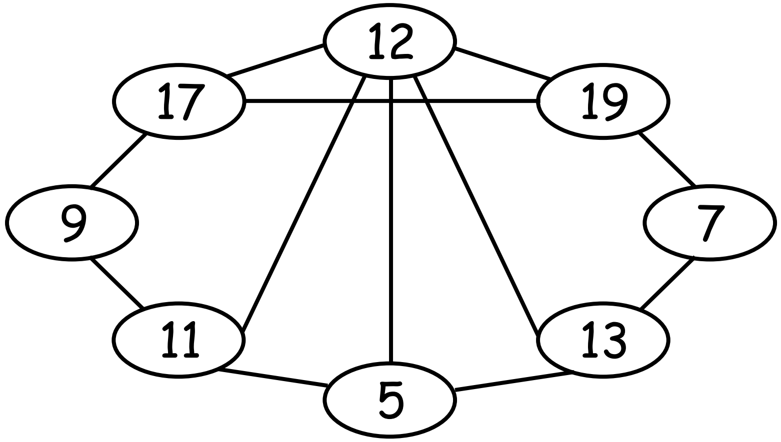
Strategy elect minimum
1. find the smallest value
2. the leader is the entity with that value

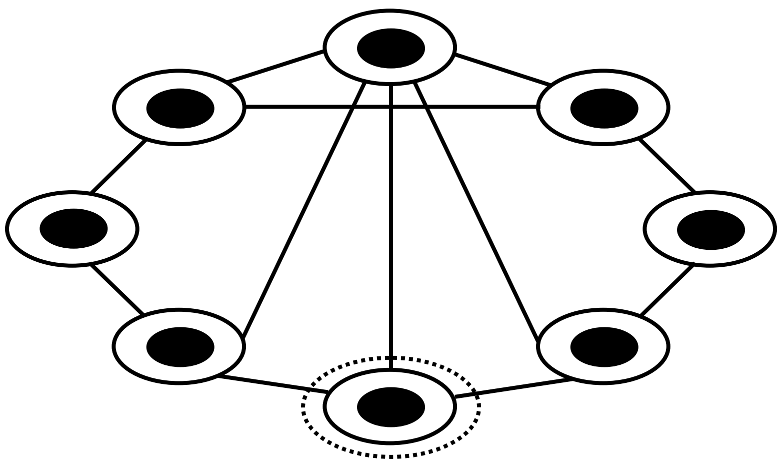
Strategy elect minimum initiator
1. find the smallest value among the initiators
2. the leader is the entity with that value

Strategy elect root
1. build a rooted spanning tree
2. the leader is the root of the tree

We can also consider taking the maximum

Note: with distinct Ids Minimum Finding is an election



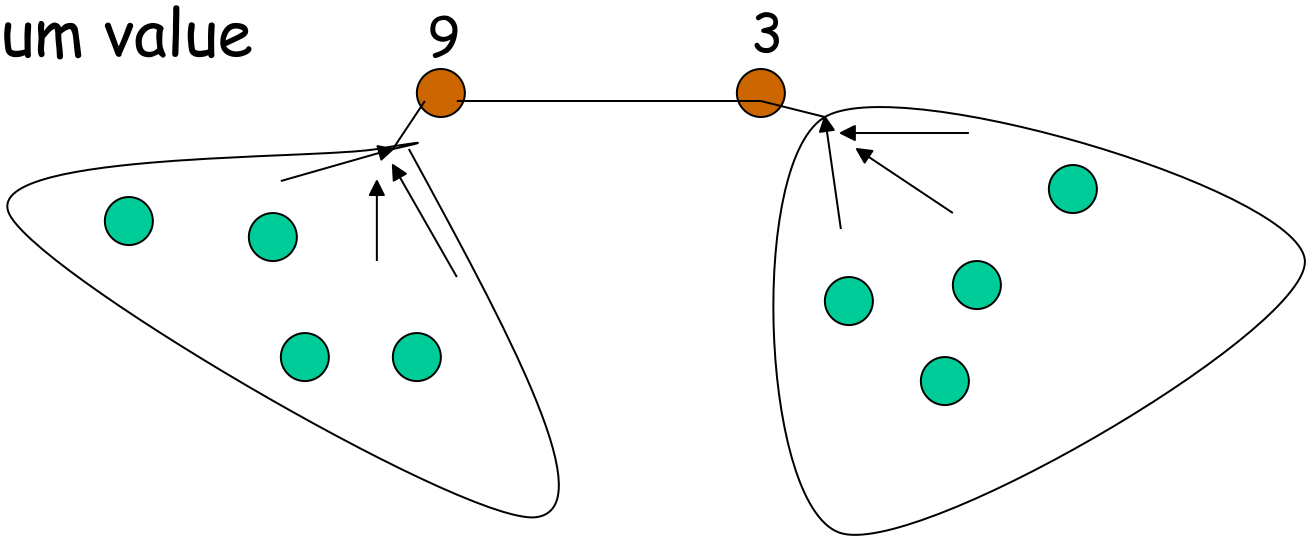


Election in the Tree

To each node x is associated a distinct identifier v(x)

A simple algorithm:

1) Execute the saturation technique,
2) Choose the saturated node holding the minimum value



rules to add to Saturation

SATURATED
Receiving(Election, id)*

```
send("Termination") to N(x) - parent
if id(x) <= id* then
    become LEADER
else
    become FOLLOWER
```

PROCESSING
Receiving("Termination")

```
send("Termination") to N(x) - parent
become FOLLOWER
```

Resolve

```
send("Election", id(x)) to parent
become SATURATED
```

Elect_Min vs Elect_Root in Trees

Complexity (n. messages)

Elect_min: $3n + k^* - 4$

Elect_root: $3n + k^* - 2$

Can we have a better estimate?

Bit Complexity

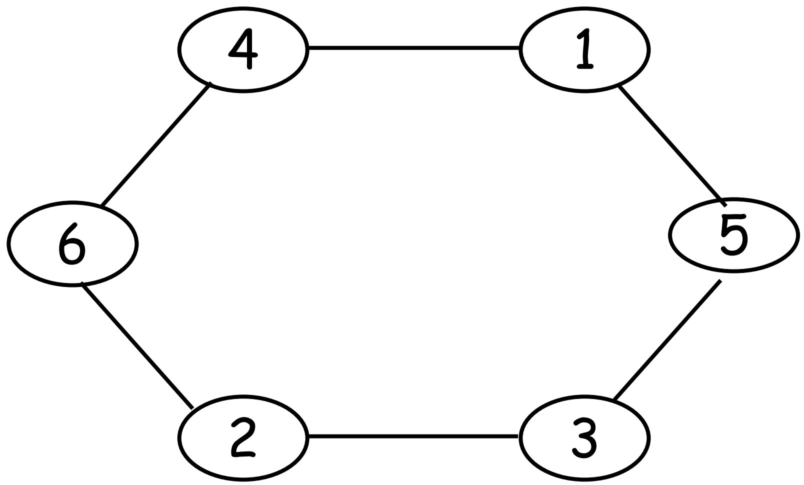
Elect_min: $n(c + \log id) + c(2n + k^* - 4)$

Elect_root: $2(c + \log id) + c(3n + k^* - 2)$

c = bit of header

$\log id$ = bit of payload

Ring

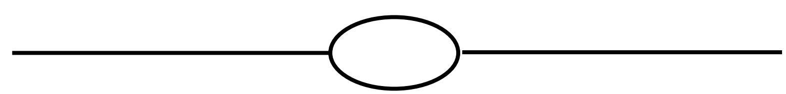


- n entities
- m= n links

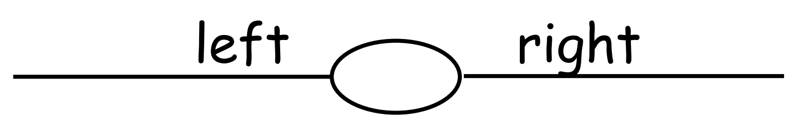
- n. of entities = n. of links

- Symmetric topology

- Each entity has two neighbors



When there is sense of direction:



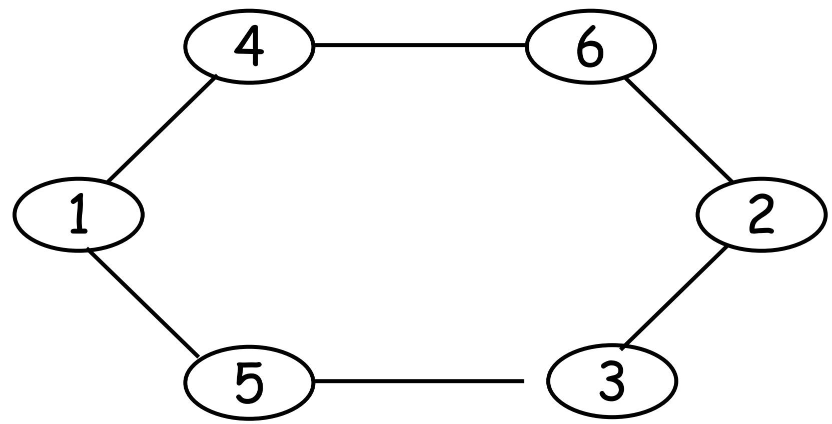
Election Algorithms in Rings

- All the way
- As Far
- Controlled distance
- Electoral stages
  --- bidirectional version
- Alternating steps

Electing the minimum

All the way

Basic Idea: Each id is fully circulated in the ring.
---> each entity sees all identities.



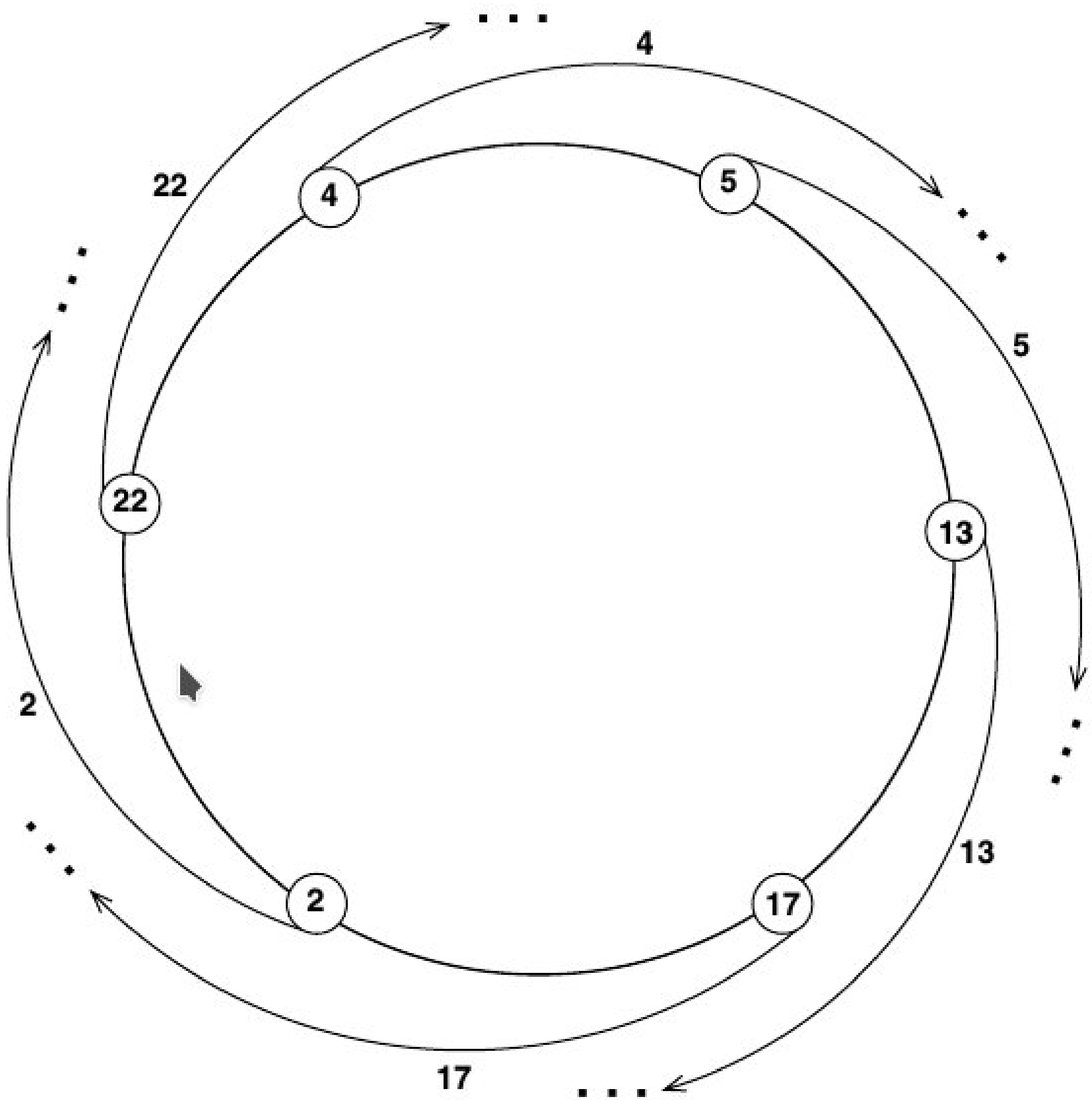
ASSUMPTIONS

- Two versions: unidirectional/bidirectional links.
- Local orientation (i.e. not necessarily a sense of direction)
- Distinct identities.

Basic Behaviour

1. Initiators start by selecting one neighbor and sending their ids

2. When a message arrives, an entity replies sending its id, forwarding the received message and storing the minimum id seen so far



Correctness and Termination

The communication activities can terminate because entities do not forward messages with their ids.

Question: how can we make aware an entity of termination?

To terminate we need further assumptions:
- FIFO assumption (termination when an entity receives its id back)
- until the number of messages reach the size of network (knowledge of the size of the network)

Note: knowledge of n can be acquired

States: S={ASLEEP, AWAKE, FOLLOWER, LEADER}
S INIT={ASLEEP};
S_TERM={FOLLOWER, LEADER}.
IR;Ring

ASLEEP

Spontaneously
INITIALIZE
become AWAKE

Receiving(``Election'', value, counter)
INITIALIZE;
send(``Election'', value, counter+1)
to other
min:= Min{min, value}
count:= count+1
become AWAKE

INITIALIZE

count:= 0
size:= 1
known:= false
send("Election",id(x),size) **to right**;
min:= id(x)

AWAKE

Receiving ("Election", value, counter)

```
If value <> id(x) then
    send ("Election",value,counter+1)  to  other
    min:= MIN{min,value}
    count:= count+1
    if  known = true   then
                    CHECK
    endif
else
        ringsize:= counter
        known:=   true
        CHECK
endif
```

```
CHECK
if count = ringsize then
        if  min = id(x) then
                become LEADER
        else
                become  FOLLOWER
        endif
endif
```

Complexity

Each identity crosses each link --> n^2

The size of each message is log(*id* + counter)

$O(n^2)$ messages
$O(n^2 \log(MaxMsg))$ bits

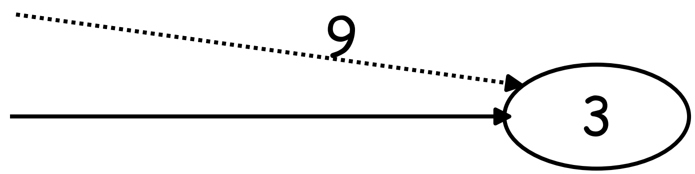
Observations:

1. The algorithm also solves the data collection problem

2. It also works for unidirectional/bidirectional.

AsFar (as it can)

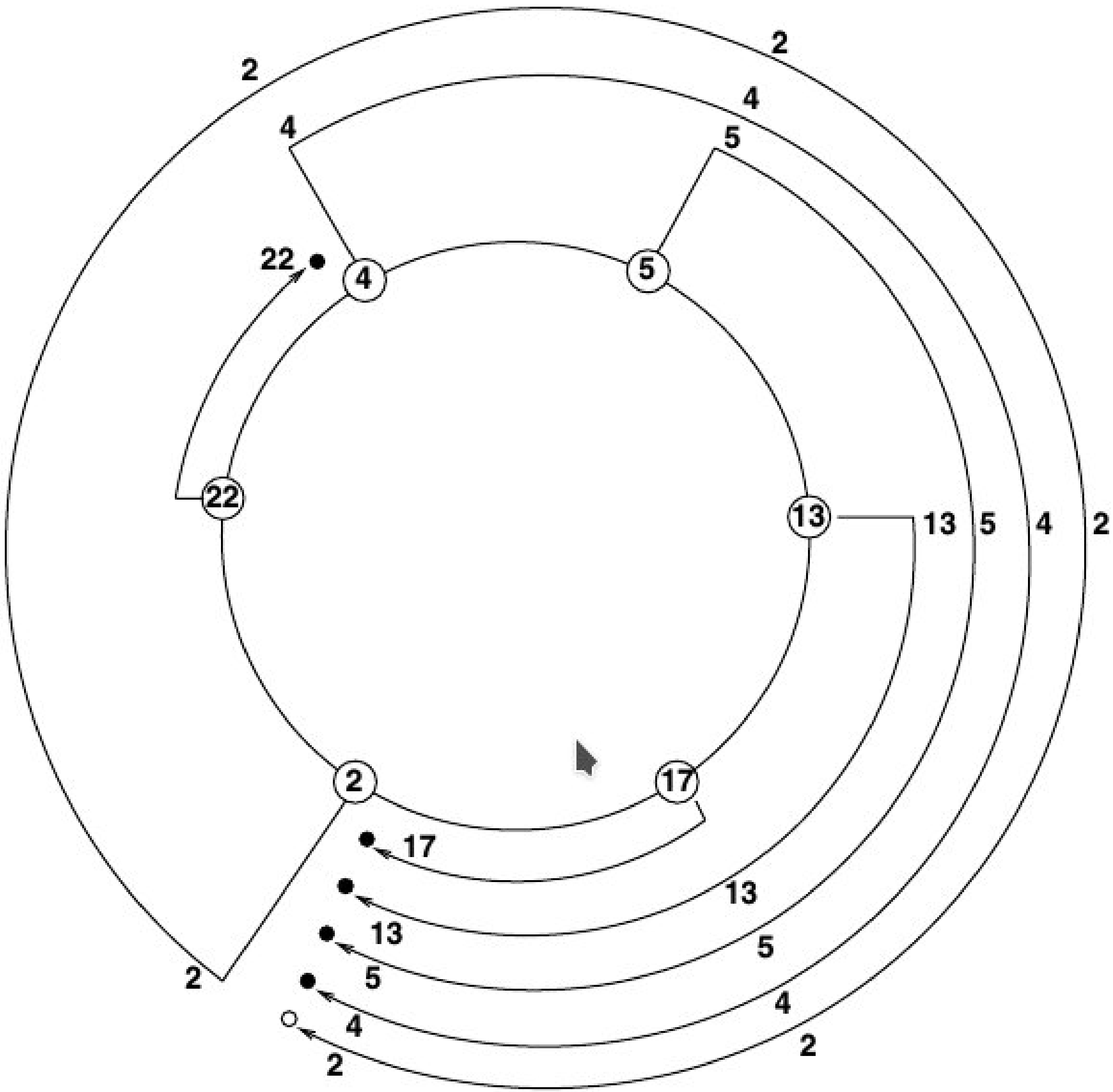
Basic Idea: It is not necessary to send and receive messages with larger *ids* than the *ids* that have already been seen.

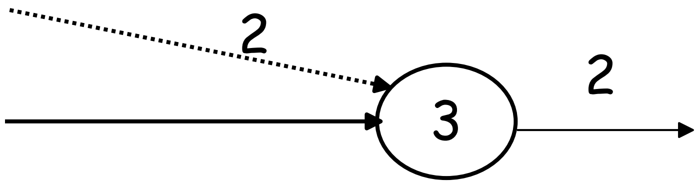


ASSUMPTIONS

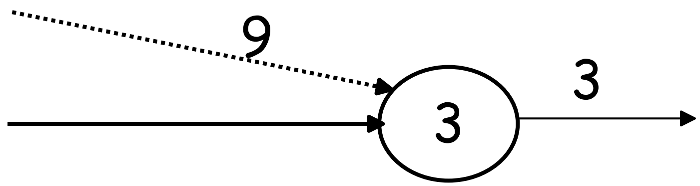
- Unidirectional/bidirectional ring
- Different *ids*
- Local orientation

The Basic Idea



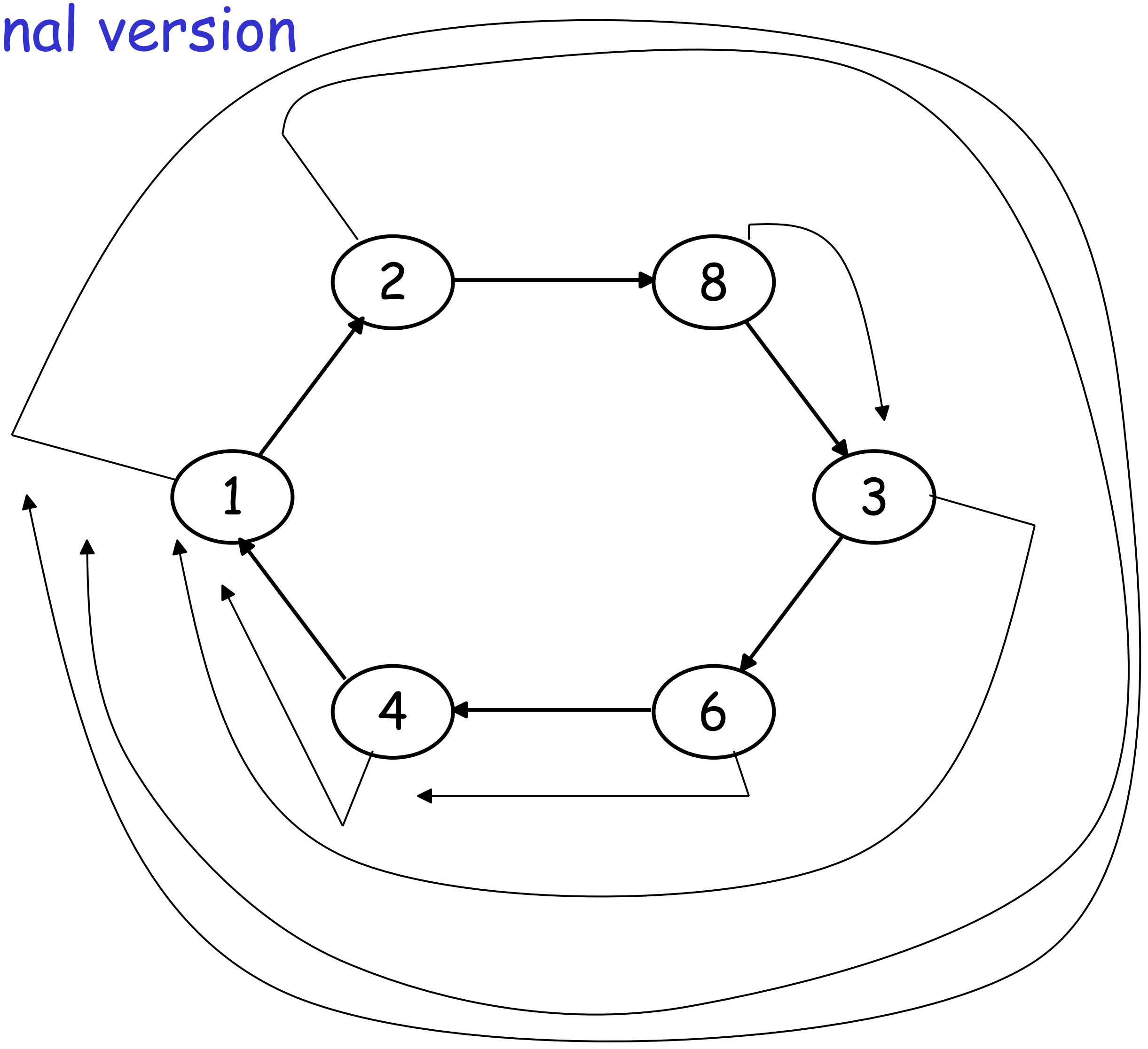


Receiving y smaller-than me
send(y) to other neighbour

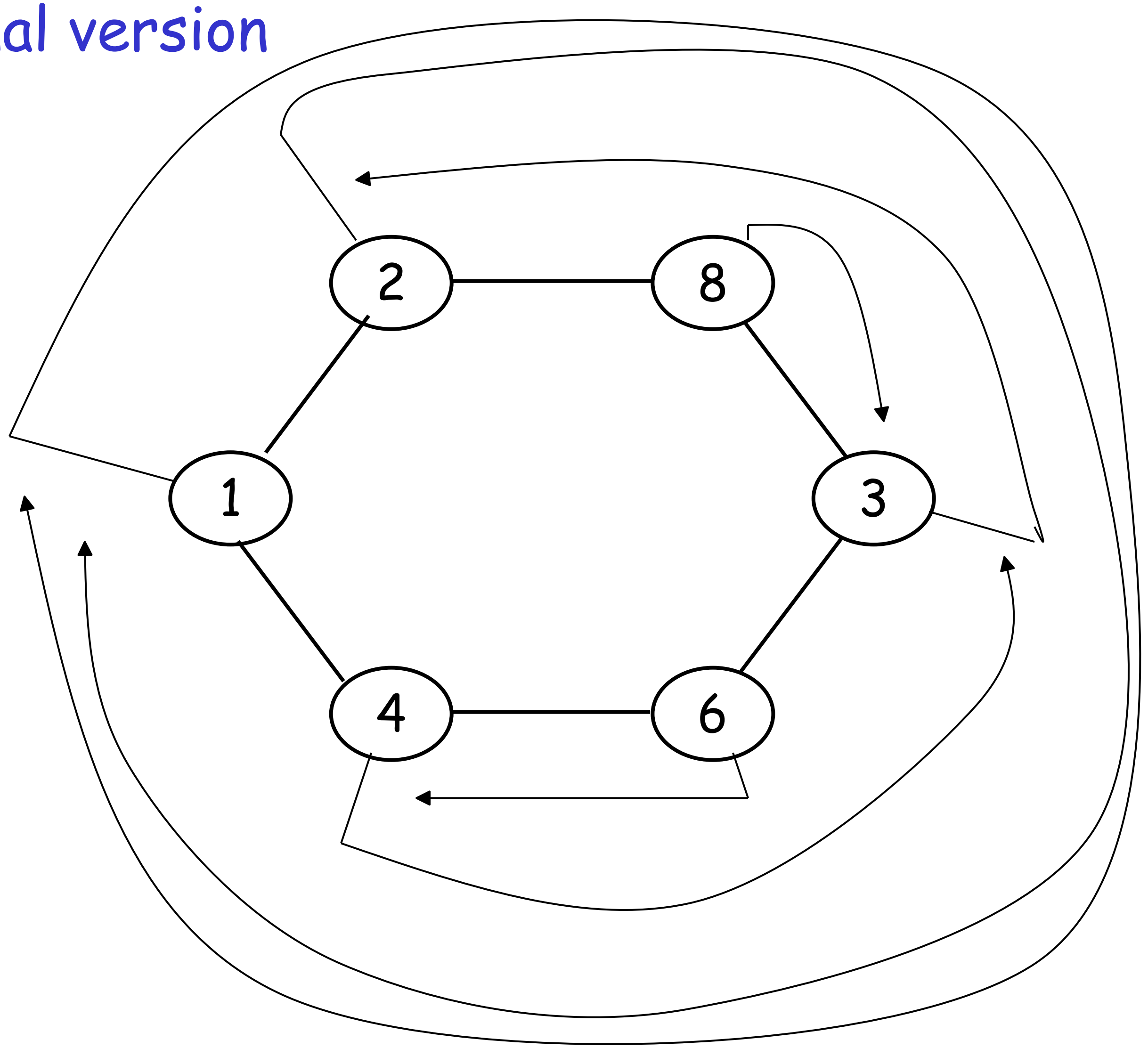


Receiving y bigger-than me
send(x) to other neighbour
(if not sent already)

unidirectional version



bidirectional version



States: S={ASLEEP, AWAKE, FOLLOWER, LEADER}
S_INIT={ASLEEP}
S_TERM={FOLLOWER, LEADER}

--- unidirectional version

ASLEEP

Spontaneously

    send("Election",id(x)) **to right**
    min:= id(x)
    become AWAKE

Receiving("Election", value)

    send("Election",id(x)) **to right**
    min:= id(x)
    If value < min then
        send("Election", value) **to other**
        min:= value
    endif
    become AWAKE

/* this could be avoided if id(x)>value

AWAKE

Receiving("Election'", value)

```
if  value < min then
        send("Election", value)  to other
        min:= value
else
   If value = id(x) then NOTIFY endif
endif
```

Receiving(Notify)

```
send(Notify)  to other
become  FOLLOWER
```

<u>NOTIFY</u>

```
send(Notify) to   right
become LEADER
```

Correctness and **Termination**

The leaders knows it is the leader when it receives its message back.

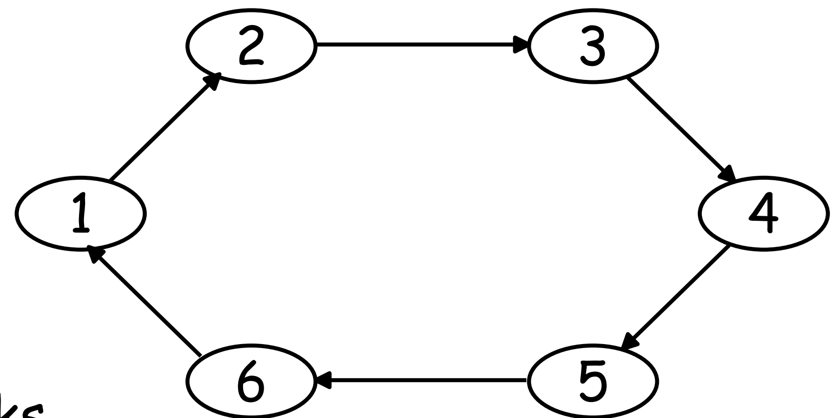
When do the other know ?

Notification is necessary !

Observations:

- Bidirectional version

Worst-Case Complexity (Unidirectional Version)

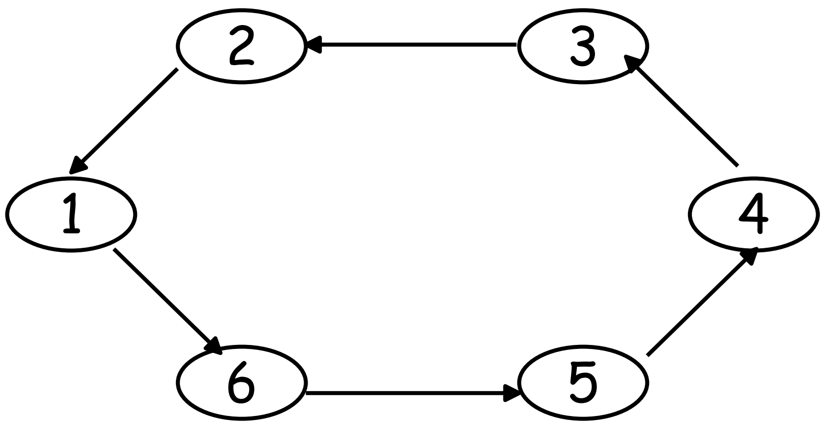


1 ---> n		links
2 ---> n - 1		links
3 ---> n - 2		links
...	...	...
n ---> 1		link

$$n + (n - 1) + (n - 2) + \ldots + 1 = \sum_{i=1}^{n} i = (n+1)(n) / 2$$

Total: $n(n+1) / 2 + n =$ **$O(n^2)$**
Last n: notification

Best-Case Complexity (Unidirectional Version)



1 ---> n links
for all $i \neq 1$ ---> 1 link (--> total = n - 1)

Total: n + (n - 1) + n = **O(n)**
Last n: notification

Average-Case Complexity

Entities are ordered in an equiprobable manner.

$O(n \log n)$

Controlled Distance

Basic idea: Operate in stages. An entity maintains control on its own message.

ASSUMPTIONS

- Bidirectional ring
- Different *ids*
- Local orientation

sense of direction only for simplicity - not needed

Ingredients

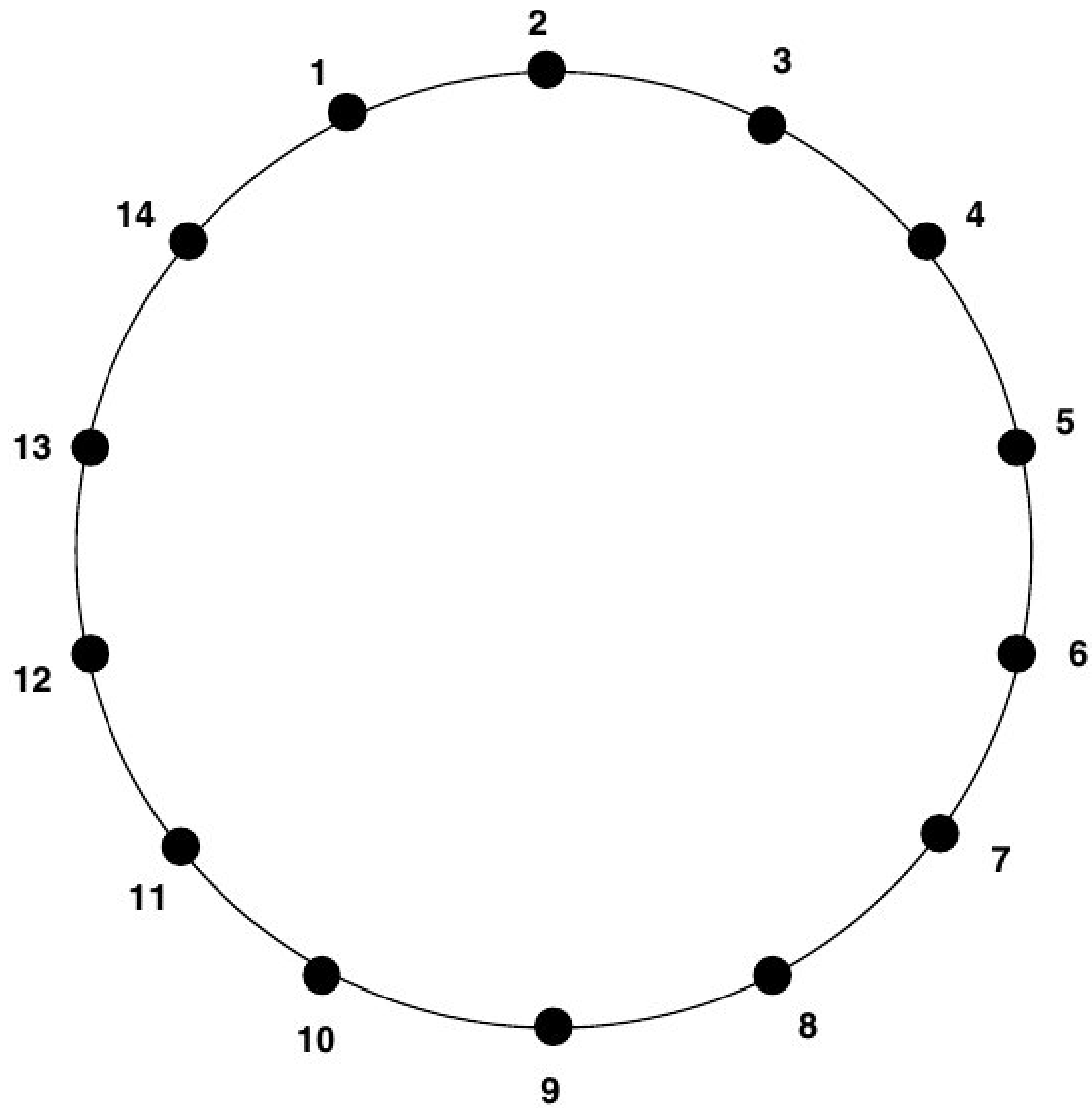
1) Limited distance (to avoid big msgs to travel too much)

Ex: stage i: distance 2^{i-1}

2) Return messages (if seen something smaller does not continue)

3) Check both sides

4) Smallest always win (regardless of stage number)


1
2
3
4
5
6
7
8
9
10
11
12
13
14

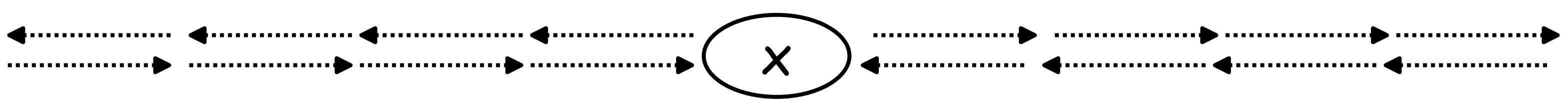
Candidate entities begin the algorithm.

Stage i:

- Each *candidate* entity sends a message with its own *id* in both directions

- the msg will travel until it encounters a smaller Id or reaches a certain distance

- If a msg does not encounters a smaller Id, it will return back to the originator



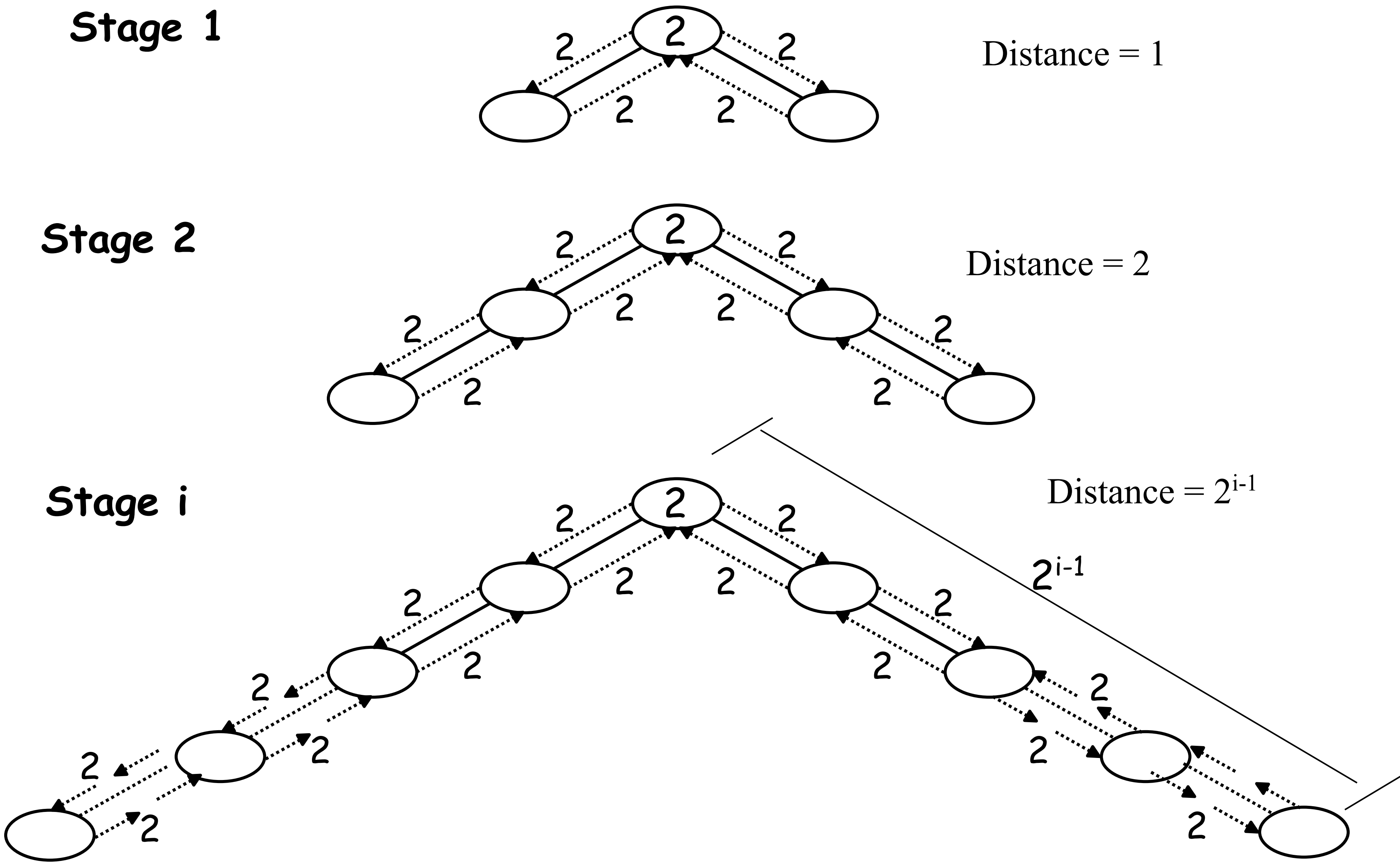
- A candidate receiving its own msg back from both directions survives and start the next stage

Entities encountered along the path read the message and:

- Each entity *i* with a *greater* identity Id_i becomes defeated (passive).

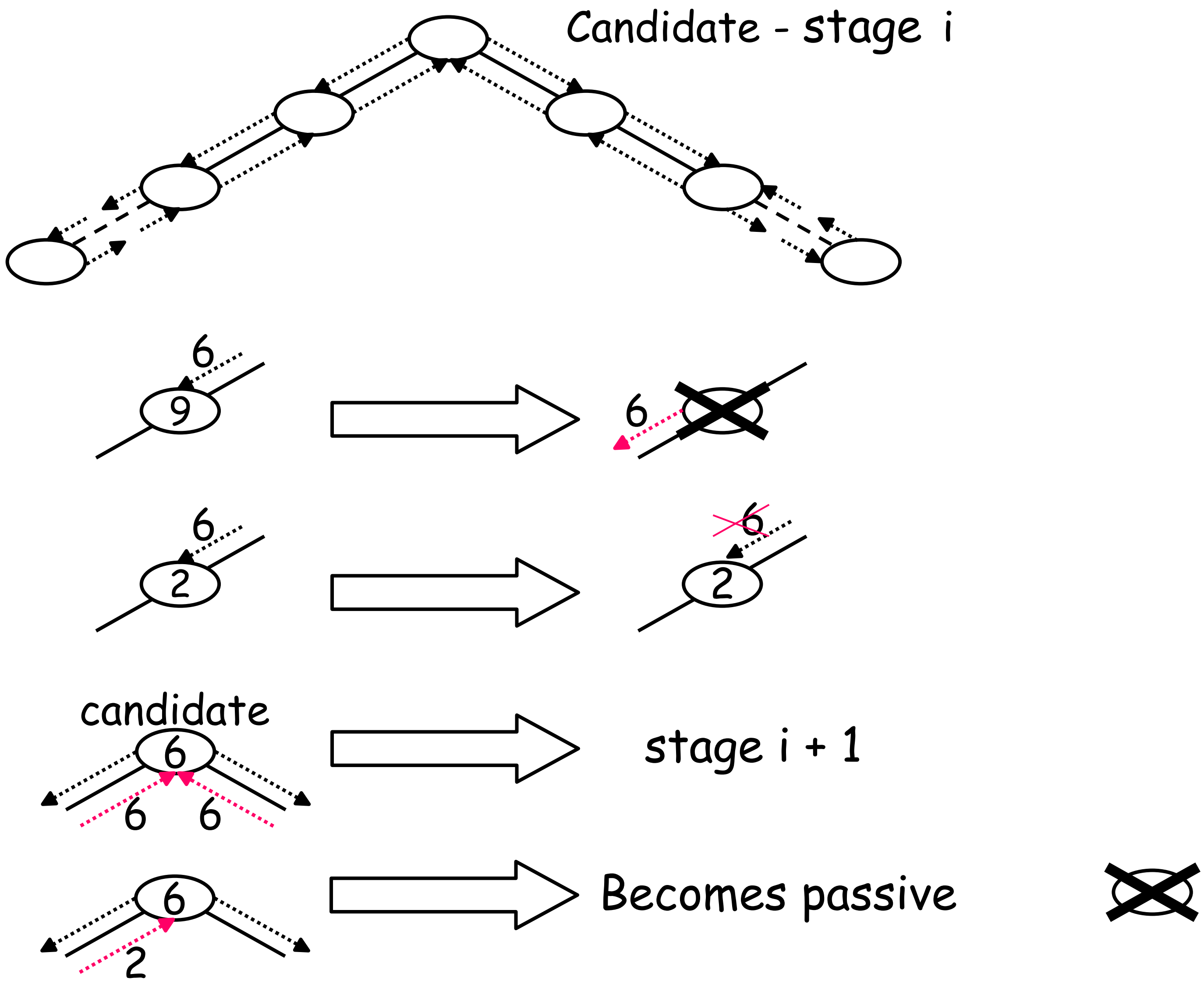
- A defeated entity forwards the messages originating from other entities, if the message is a notification of termination, it terminates

More...



Stage 1
Distance = 1
Stage 2
Distance = 2
Stage i
Distance = 2^{i-1}
2^{i-1}

More...


Candidate - stage i
6
9
6
6
2
6
2
candidate
6
6
6
stage i + 1
6
2
Becomes passive

States: S={ASLEEP, CANDIDATE, DEFEATED, FOLLOWER, LEADER}
S_INIT={ASLEEP}
S_TERM={FOLLOWER, LEADER}

ASLEEP
Spontaneously
INITIALIZE
become CANDIDATE

Receiving("Forth", id, stage*, limit*)*
if $id^* < id(x)$ **then**
PROCESS_MESSAGE
become DEFEATED
else
INITIALIZE
become CANDIDATE

CANDIDATE

```
Receiving("Forth", id*, stage*, limit*)
        if  id* < id(x)  then
            PROCESS_MESSAGE
            become DEFEATED
        else
            if  id* = id(x) then NOTIFY

Receiving("Back", id*)
        if  id* = id(x) then CHECK

Receiving("Notify")
        send("Notify") to other
        become  FOLLOWER
```

DEFEATED

```
Receiving(*)
        send(*) to other
        if * = Notify then
             become  FOLLOWER
```

```
INITIALIZE
        stage := 1
        limit := dis(stage)
        count := 0
        send("Forth", id(x), stage, limit) to N(x)

PROCESS-MESSAGE
        limit* := limit* - 1
        if limit* = 0 then
            send("Back", id*, stage*) to sender
        else
            send("Forth", id*, stage*, limit*) to other

CHECK
        count := count + 1
        if count = 1 then
          count := 0
          stage := stage + 1
          limit := dis(stage)
          send("Forth", id(x), stage, limit) to N(x)
NOTIFY
        send("Notify") to right
        become LEADER
```

Correctness and Termination

If a candidate receives its message from the opposite side it sent it, it becomes the leader and notifies.

- The smallest id will always travel the max distance defeating every entity it encounters

- The distance monotonically increases eventually becoming greater than n

- The leader will eventually receive its message from the opposite directions

Note: we do not need message ordering.
What happens if an entity receives a message from a higher stage ?

Complexity

Messages

When the distance is doubled at each stage i.e., dis(i) = 2^{i-1}, the worst case complexity is

$$O(n \log n)$$

Time (ideal)

- 2dis(i) time units required by the message with the smaller id to cover the distance
- 2n times unit are need to wake-ups & final notifications

$$2n + \sum_{i=1} 2\,dis(i) = O(n)$$

Stages

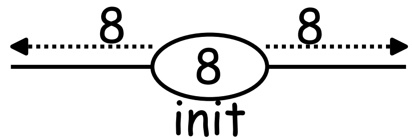
Basic idea:

A message will travel until it reaches another candidate
A candidate will receive a message from both sides

ASSUMPTIONS

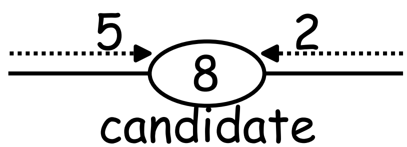
- Distinct *ids*
- Bidirectional ring (+ unidirectional version)
- Local orientation
- **Message ordering** (for simplicity only: not needed)

Each *candidate* sends its own *Id* in both directions.



When a *candidate i* receives two messages Id_j (from the right) and Id_k (from the left), it determines if it becomes *passive* (= it is not the smallest), or if it remains *candidate* (= it is the smallest).

When a *candidate i* receives two messages Id_j (from the right) and Id_k (from the left),



After receiving the first message:
`close-port` (enqueue messages possibly arriving later)

After receiving the second message, perform the action and **`re-open-port`**

States: S={ASLEEP, CANDIDATE, WAITING, DEFEATED, FOLLOWER, LEADER}
S_INIT={ASLEEP}
S_TERM={FOLLOWER, LEADER}

ASLEEP

Spontaneously

    INITIALIZE
    become CANDIDATE

Receiving("Election", id, stage*)*

    INITIALIZE
    min := MIN(id*, min)
    close-port(sender)
    become WAITING

```
CANDIDATE
Receiving("Election", id*, stage*)
        if id* <> id(x) then
            min := MIN(id*, min)
            close-port(sender)
            become WAITING
        else
            send(Notify) to N(x)
            become LEADER

WAITING
Receiving("Election", id*, stage*)
        open(other)
        stage := stage + 1
        min := MIN(id*, min)
        if min = id(x) then
            send("Election", id(x), stage) to N(x)
            become CANDIDATE
        else
            become DEFEATED
```

DEFEATED

Receiving(*)

send(*) to other

if * = Notify **then**

become FOLLOWER

INITIALIZE

stage := 1

count := 0

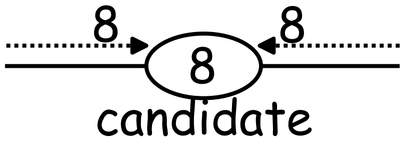
min := id(x)

send("Election", id(x), stage) to N(x)

Correctness and termination

The minimal entity will never cease to send messages.

When an entity knows that it is the *leader*

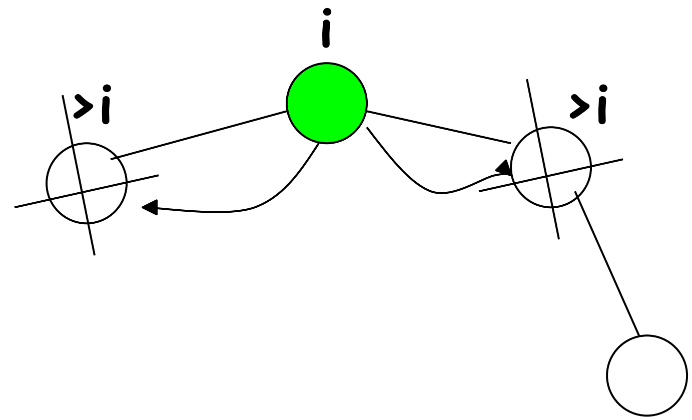


it sends a *notification* message which
travels around the ring.

Complexity - Worst Case

At each step: At least half the entities became passive.

$$n_{i+1} \le \frac{n_i}{2}$$



$n_0 = n$

$n_1 = n/2$

$n_i = n/2^i$

$n/2^k = 1$

when

$k = \log n$

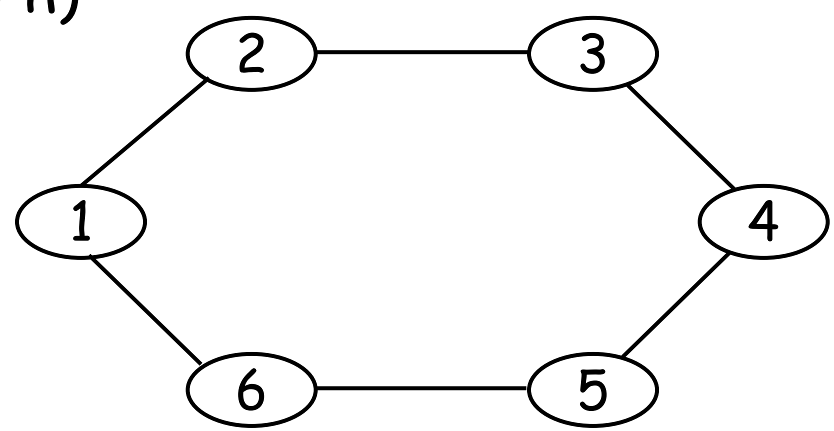
 # steps: At most (log *n*)

Each entity sends or resends 2 messages.
 # messages: 2n
 # bits: 2*n* * (log *n*)

Last entity: 2n messages to understand that it is the last active entity, then n notification messages.

Total: 2n * (log *n*) + 3 n = O(n log n)

Best Case ?



Removing Message Ordering

The ordering assumption ensures that messages received by a candidate in stage i are originated by candidates at the same stage

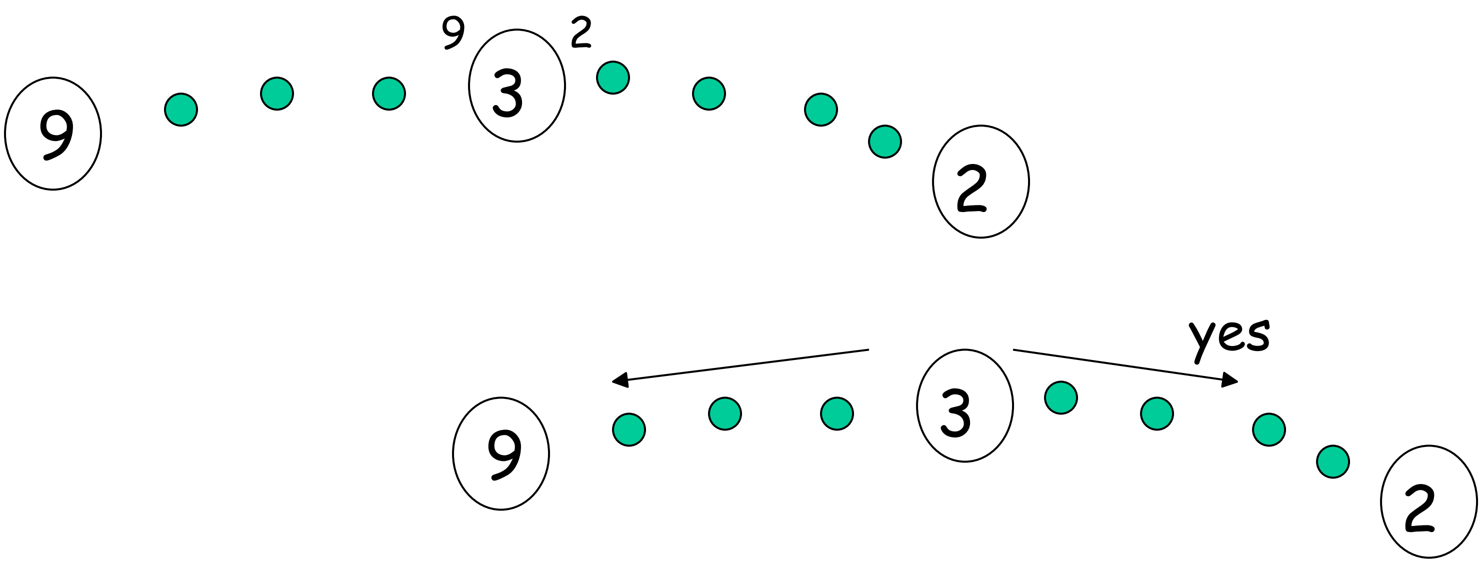
We can remove this assumption and modify the protocol to enforce that messages are processed in order

1. Each message carries the stage number of the entity originating it

2. When a candidate x in stage i receives a message M with stage $j > i$, x stores M and processes it after all messages with stage $i + 1, \ldots, j - 1$ arrive

Stages with Feedback

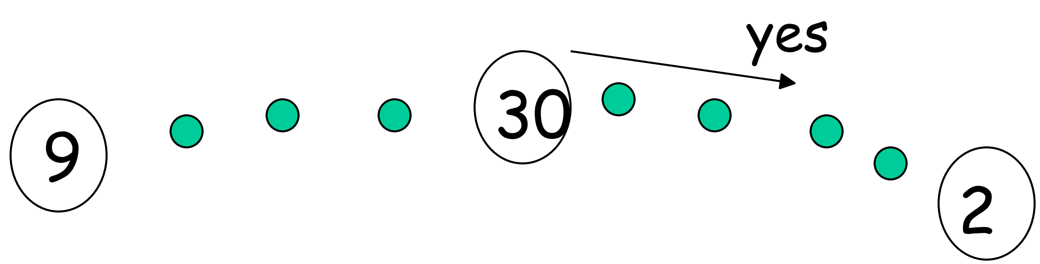
A feedback is sent back to the originator of the message



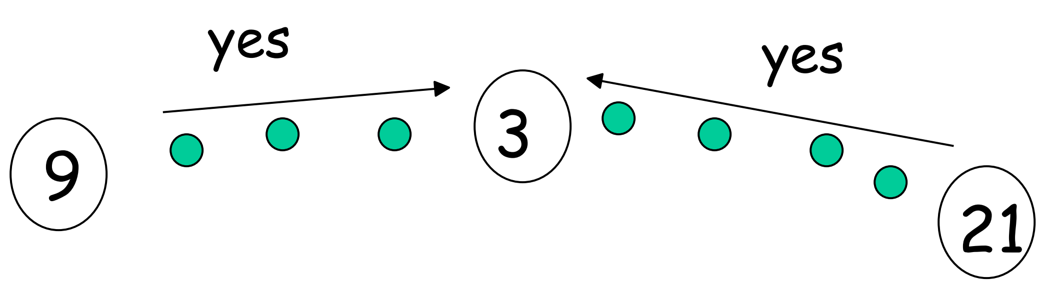
send YES to the smallest of the two IF it is smaller than me

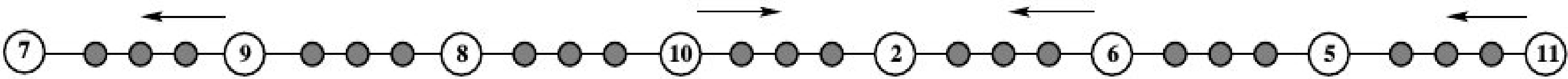
A candidate x receives two messages from two neighboring candidates y=r(i,x) and z=l(i,x)

A positive feedback is sent to
- y if id(y) < MIN(id(x), id(z))
- z if id(z) < MIN(id(x), id(z))



A candidate survives a stage if it receives positive feedback from its neighboring candidates





Candidates 8 and 5 do not receive any feedback

Candidate 2 is the only one that survives this stage

Question
How can they do they didn't survive this stage?

Answer
They do not step to the next stage, and become defeated when they receive the election message from 2 at the next stage (it works like a negative feedback).

Correctness & Termination

Consider the entity with x_{min} that will be the leader

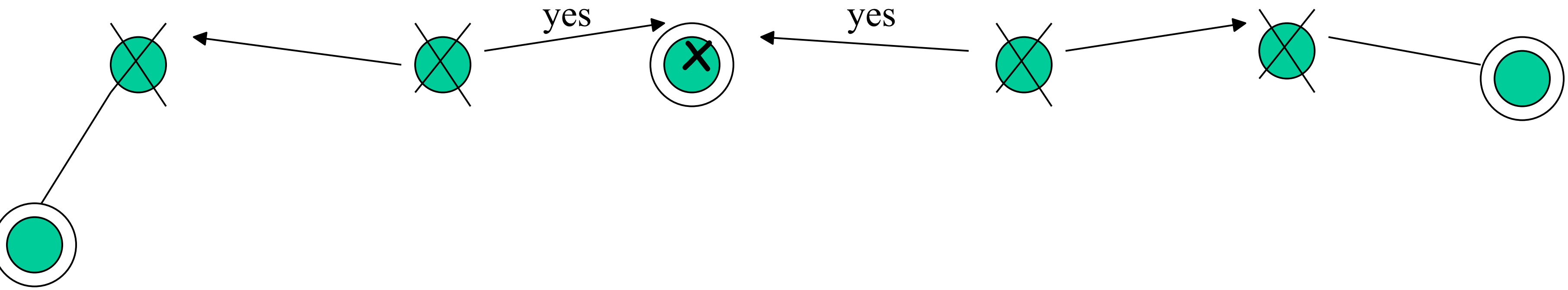
It never sends a positive feedback and always receives two positive ones

This means that each neighboring candidate at each stage does not survive

The number of candidates at each stage is monotonically decreasing, until x_{min} will be the only one

Number of stages

If x survives at , it must have received a feedback from both neighbouring candidates $l(i,x)$ and $r(i,x)$ Moreover, $l^2(i,x)$ and $r^2(i,x)$ do not survive too.



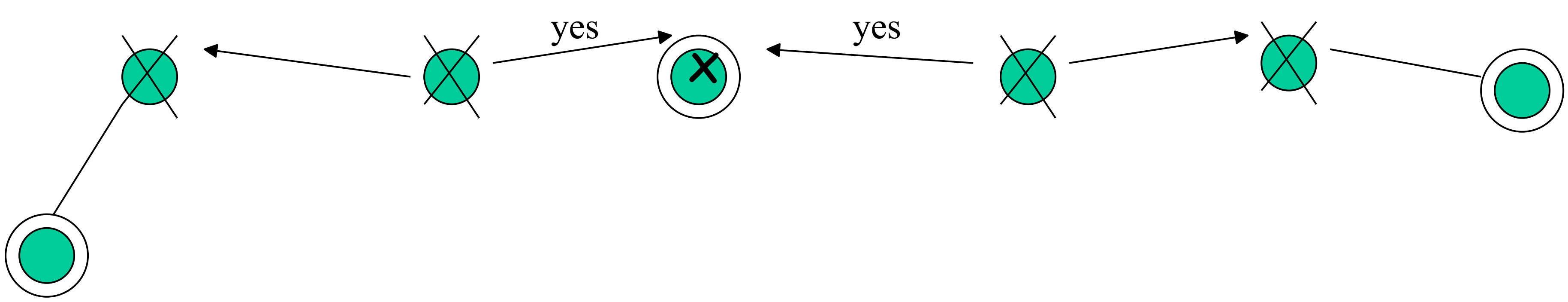
$n_{i+1} <= \frac{n_i}{3}$ survivors at each stage

n. stages $\log_3 n$

Number of messages

At stage i, there are
- 2 n election messages
- n feedback messages

Tot = 3 n messages per stage



Messages for all stages
$3n + \log_3 n = O(n \log n)$

Alternating Steps

Basic idea: Alternating directions.

- Different *ids*.
- Bidirectional ring and sense of direction.
- Local orientation.
- Message ordering.

send-left
begin-to-defeat (if possible)
send-right

Algorithm

1. **Each entity** sends a message **to its right**. This message contains the entity's own *id*.

2. Each entity compares the *id* it received from its left to its own *id*.

3. If its own *id* is greater than the received *id*, the entity becomes passive.

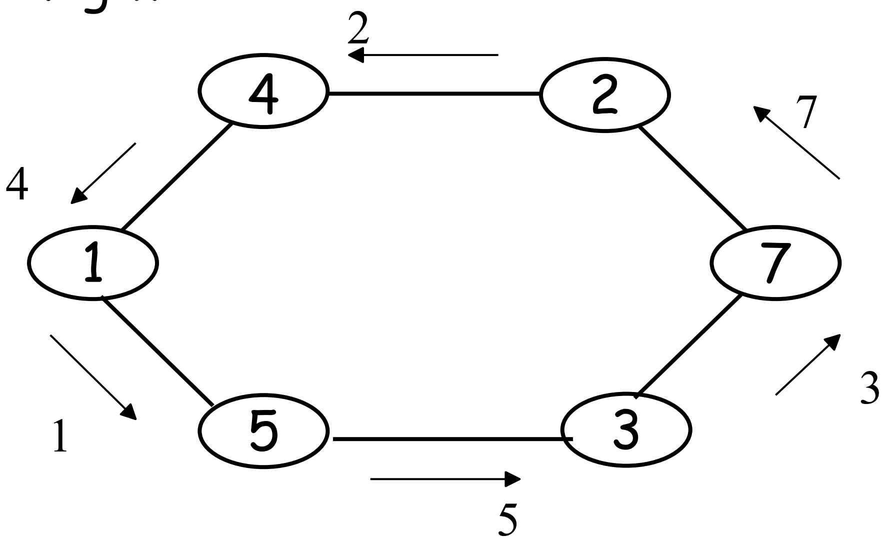
4. All entities that remained active (surviving) send their *id*s **to their left**.

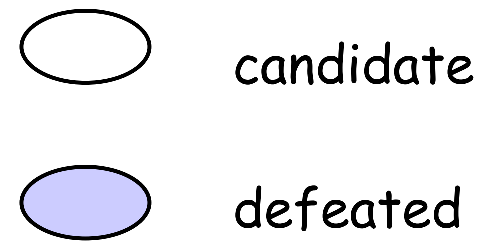
5. A surviving entity compares the *id* it received from its right with its own *id*.

6. If its own *id* is greater than the *id* it received, it becomes passive.

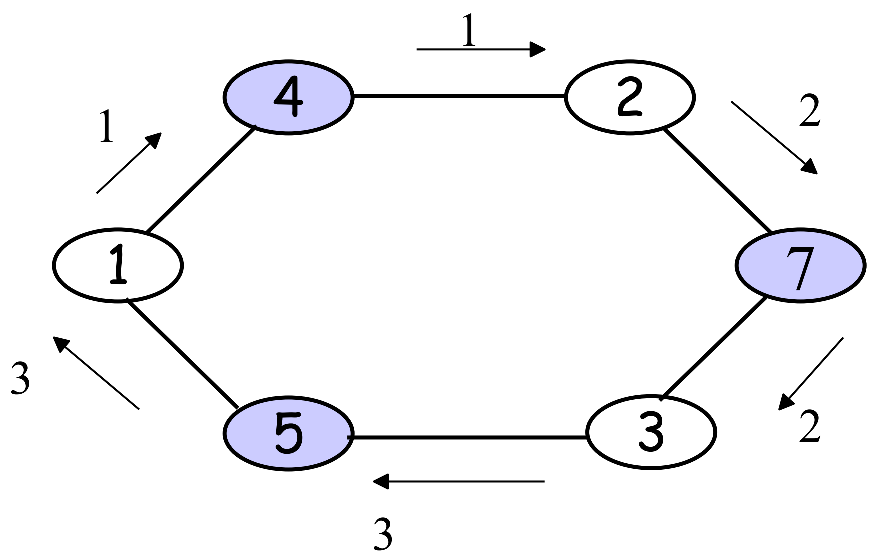
7. Go back to step 1 and repeat until an entity receives its own *id* and becomes *leader*.

Step 1: right





Step 2: left



States: S={ASLEEP, CANDIDATE, WAITING, DEFEATED, FOLLOWER, LEADER}
S_INIT={ASLEEP}
S_TERM={FOLLOWER, LEADER}
Restrictions: **IR;**Oriented Ring; Message Ordering

ASLEEP
Spontaneously
    INITIALIZE
    become CANDIDATE

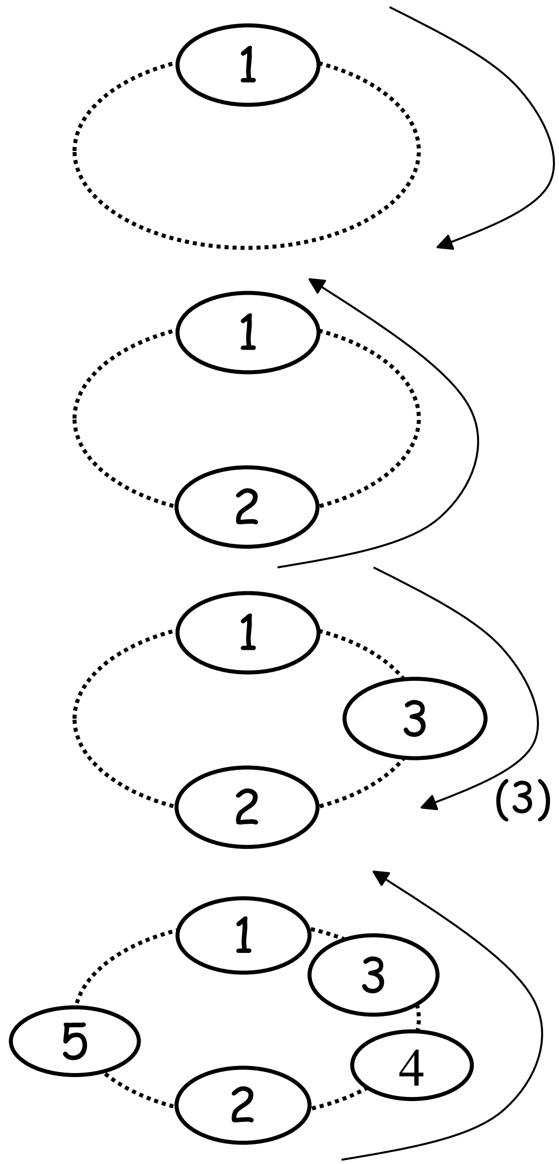
Receiving("Election", id, step*)*
    INITIALIZE
    PROCESS_MESSAGE
    become CANDIDATE

```
CANDIDATE
Receiving("Election", id*, step*)
        if id* <> id(x) then
            PROCESS_MESSAGE
        else
            send(Notify) to N(x)
            become LEADER

DEFEATED
Receiving(*)
        send(*) to other
        if * = Notify then
            become FOLLOWER
```

```
INITIALIZE
step := step + 1
min := id(x)
send("Election", id(x), step) to right
close-port(right)

PROCESS_MESSAGE
    if id* < min then
        open(other)
        become DEFEATED
    else
        step := step + 1
        send("Election", id(x), step) to sender
        close-port(sender)
        open(other)
```

Complexity

Analyze # of steps in worst case:



Last phase k

1 active entity

Phase k - 1

at least 2 active entities

(2) will become passive at the next step.

Phase k - 2

at least 3 active entities

(3) must be there; otherwise, (2) would be killed.

Phase k - 2

at least 5 active entities

1 2 3 5 8 13 21 ....

 # steps = index of the lowest Fibonacci number >= n

F_1 = 1
F_2 = 2
F_3 = 3
F_4 = 5
F_5 = 8
. . .

F_k = i = ?
= approx. $1.45 \log_2 n$

 # Messages = n for each step

Total = approx. $1.45 \, n \log_2 n$

Conjecture:

In unidirectional rings,
the worst case complexity is (n^2);
to have a complexity of $O(n \log n)$ messages,
bidirectionality is necessary.

The Conjecture is false.

Unidirectional version

Simulation of the bidirectional algorithm with the same complexity.

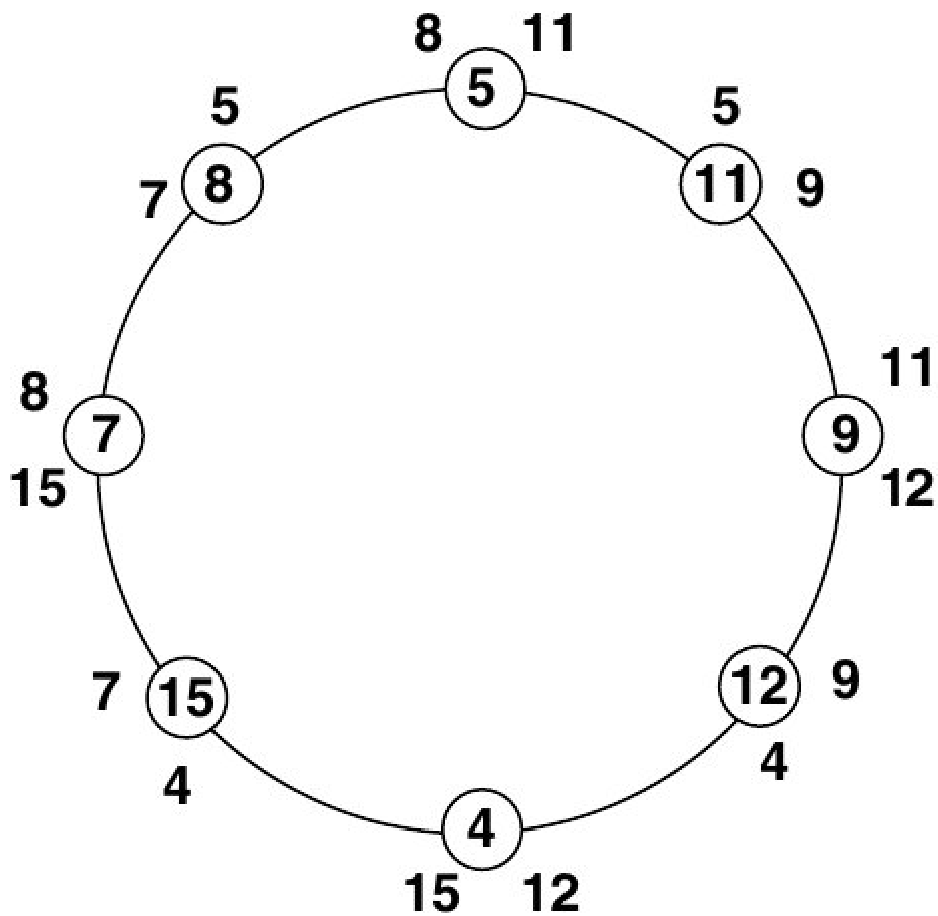
Example: Unidirectional stages

Decompose the the operation of send in both directions in two steps

- Send the id(x) in the only possible direction
- Send also what you receive

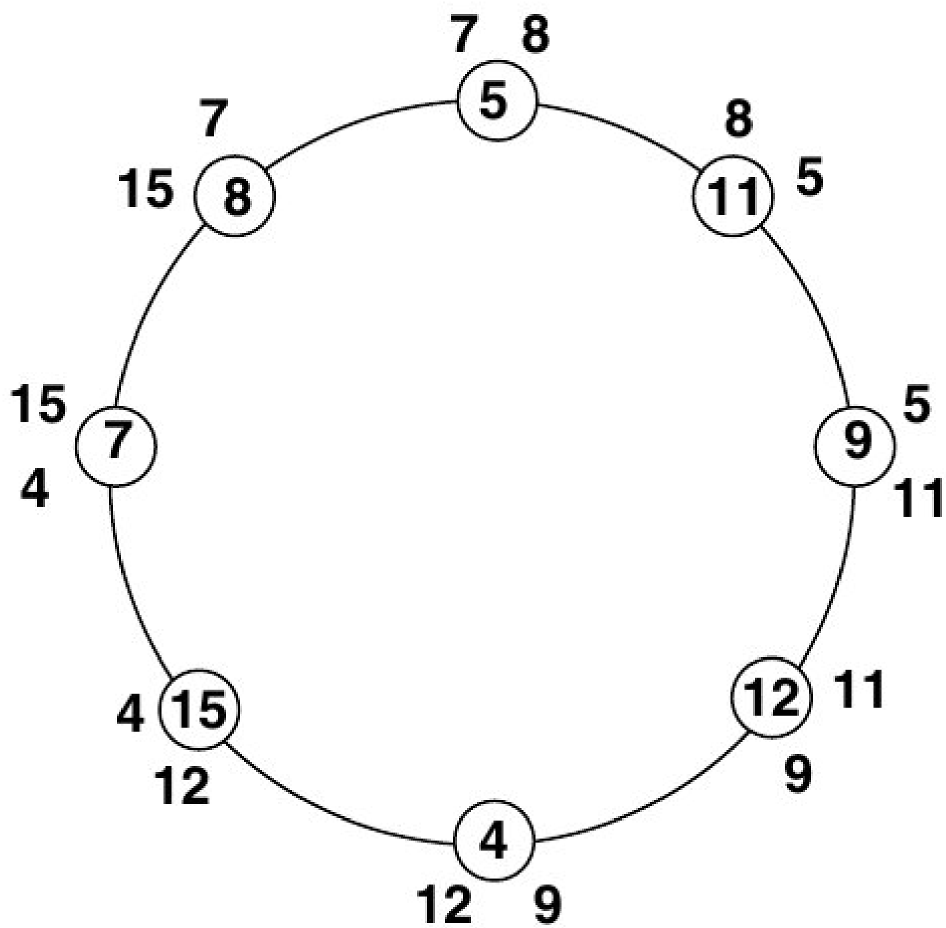
The entities have exactly the same information as in the case of bidirectional links but shifted to the next cadidate

Recall with bidirectional links



After sending the id

Unidirectional Links



The same information but shift to next

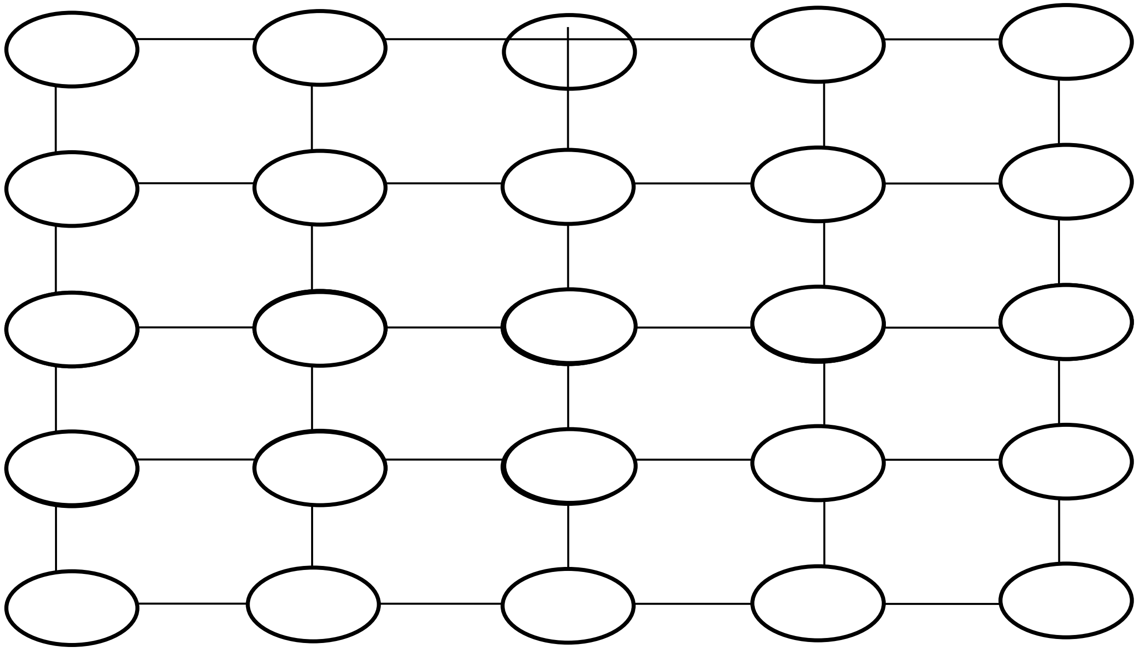
Election in Bidirectional Links (a recap)

bidirectional	*worst case*	*average*	*notes*
All the Way	n^2	n^2	
AsFar	n^2	$0.69n\log n + O(n)$	
ProbAsFar	n^2	$0.49n\log n + O(n)$	
Control	$6.31n\log n + O(n)$		
Stages	$2n\log n + O(n)$		
StagesFbk	$1.89n\log n + O(n)$		
Alternate	$1.44n\log n + O(n)$		oriented ring
BiMinMax	$1.44n\log n + O(n)$		
lower bound		$0.5n\log n + O(n)$	$n = 2^p$ known

Election in Unidirectional Links

unidirectional	*worst case*	*average*	*notes*
All the Way	n^2	n^2	
AsFar	n^2	$0.69n \log n + O(n)$	
UniStages	$2n \log n + O(n)$		
UniAlternate	$1.44n \log n + O(n)$		
MinMax	$1.44n \log n + O(n)$		
MinMax+	$1.271n \log n + O(n)$		
lower bound		$0.69n \log n + O(n)$	
lower bound		$0.25n \log n + O(n)$	$n = 2^p$ known

Mesh



If it is square mesh: n nodes = $n^{\frac{1}{2}} \times n^{\frac{1}{2}}$

$m = O(n)$

Asymmetric topology

corners

border

internal

Idea: Elect as a leader one of the four corners

Three phases:

1) Wake up

2) Election (on the border) among the corners

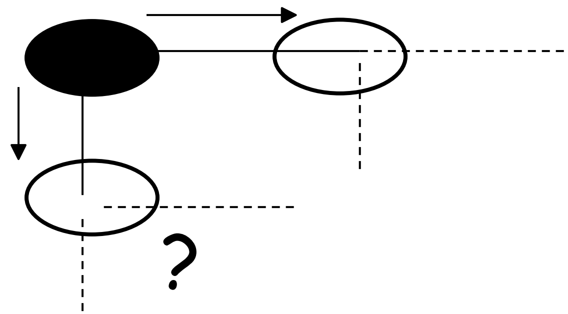
3) Notification

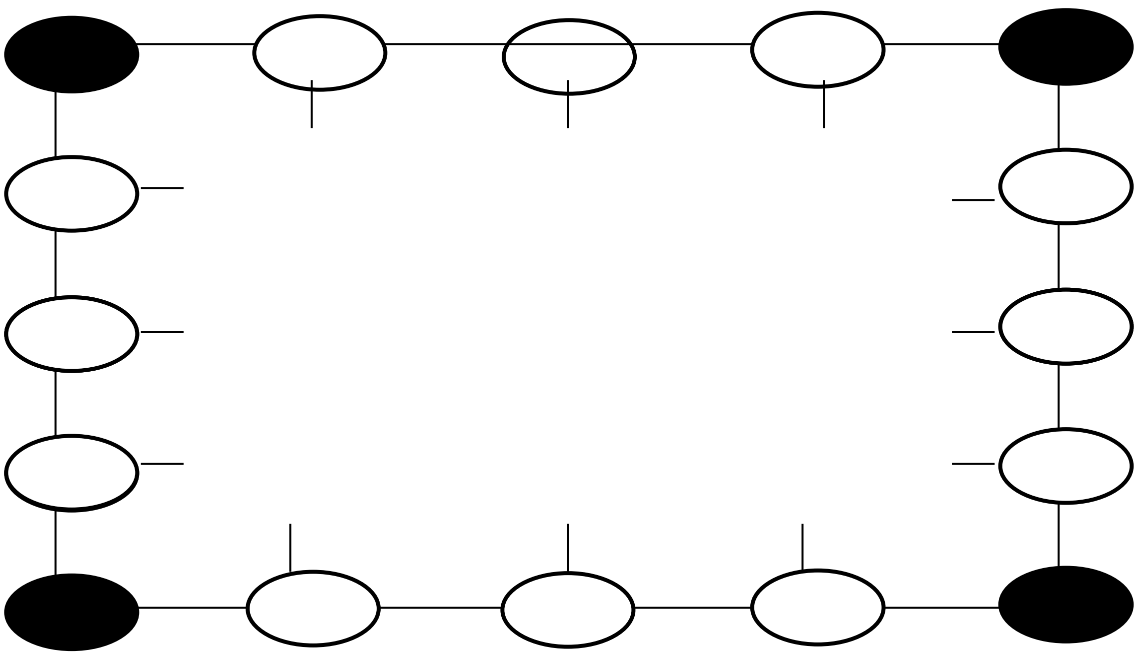
- Wake up

- Each initiator send a wake-up to its neighbours
- A non-initiator receiving a wake up, sends it to its other neighbours

$$O(m) = O(n)$$

2) Election on the border started by the corners





O(n)

3) Notification by flooding

$O(m) = O(n)$

TOT: $O(n)$