Lecture 22

- Outline
  - Leader Election
  - Complexity measure of distributed computation
    - Bits
    - Messages
    - Rounds
  - Ring with IDs, no faults – leader election
    - Chang – Roberts Algorithm with Participants/Non-Participants

LeLann's ring election

- Whenever a node receives back its id, it has seen every other initiators id
  - Assuming FIFO channels

- Let every node keep a list of every identifier seen ($list_p$)
  - If non-initiator, $state=lost$ immediately
  - If initiator, when own id received:
    - $state=leader$ if $min\{list_p\}=p$
    - $state=lost$ otherwise
- Followed by a ELECTED message, any process in an election cannot initiate a new one until it has received an ELECTED message.

LeLann's Algorithm

$$
\begin{array}{l}
\textbf{var } List_p \quad : \text{set of } \mathcal{P} \quad \textbf{init } \{p\} \; ; \\
\quad state_p \; ; \\
\\
\textbf{begin if } p \text{ is initiator } \textbf{then} \\
\quad \textbf{begin } state_p := cand \; ; \text{send } \langle \textbf{tok}, p \rangle \text{ to } Next_p \; ; \text{receive } \langle \textbf{tok}, q \rangle \; ; \\
\quad\quad \textbf{while } q \neq p \textbf{ do} \\
\quad\quad\quad \textbf{begin } List_p := List_p \cup \{q\} \; ; \\
\quad\quad\quad\quad \text{send } \langle \textbf{tok}, q \rangle \text{ to } Next_p \; ; \text{receive } \langle \textbf{tok}, q \rangle \\
\quad\quad\quad \textbf{end} \; ; \\
\quad\quad \textbf{if } p = \min(List_p) \textbf{ then } state_p := leader \\
\quad\quad\quad \textbf{else } state_p := lost \\
\quad \textbf{end} \\
\quad \textbf{else while } true \textbf{ do} \\
\quad\quad \textbf{begin } \text{receive } \langle \textbf{tok}, q \rangle \; ; \text{send } \langle \textbf{tok}, q \rangle \text{ to } Next_p \; ; \\
\quad\quad\quad \textbf{if } state_p = sleep \textbf{ then } state_p := lost \\
\quad\quad \textbf{end} \\
\textbf{end}
\end{array}
$$

Initially only know myself

Send my id, and wait

Repeat forwarding and collecting ids until we receive our id

***Termination**:*

did we win or lose

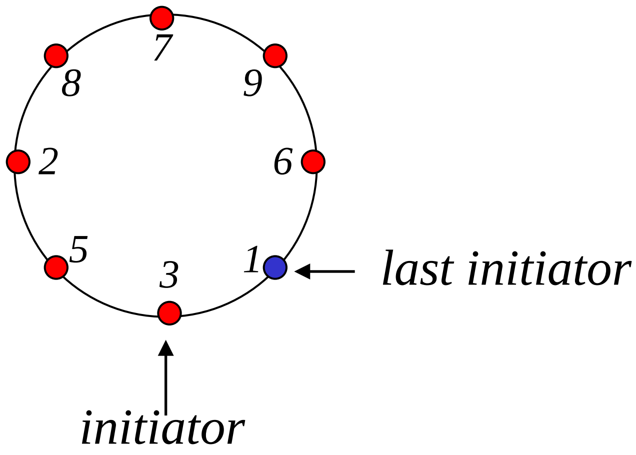
Non-initiators just forward and lose

Message Complexity

- Worst case is every node is initiator (N)
  - Every initiator sends N messages
- Gives a total of N^2 messages

Time Complexity

- Assume last initiator f starts at time N-1
  - f terminates after its token has circulated the ring, N steps



- Time complexity $2N$-1

Chang-Roberts - An improvement

- Chang and Roberts came up with a small improvement

- Idea:
  - When a node receives a token with smaller id than itself, why should it keep forwarding it?
  - It is a waste, we know that that id will never win!
  - Lets drop tokens with smaller ids than ourselves!

How to make it work

- But if we drop a token with id s, node s will be waiting forever on its token to come back
- How do we solve that?

- If node s token token is dropped, it will anyway receive another token with lower id
  - In that case, s should set *state=lost*

Chang and Roberts Algorithm

```
var state_p ;

begin if p is initiator then
        begin state_p := cand ; send ⟨tok, p⟩ to Next_p ;
            while state_p ≠ leader do
                begin receive ⟨tok, q⟩ ;
                    if q = p then state_p := leader
                    else if q < p then
                        begin if state_p = cand then state_p := lost ;
                              send ⟨tok, q⟩ to Next_p
                        end
                end
        end
    else while true do
        begin receive ⟨tok, q⟩ ; send ⟨tok, q⟩ to Next_p ;
              if state_p = sleep then state_p := lost
        end
end
(* Only the leader terminates the program. It floods a message to all
processes to inform them of the leader's identity and to terminate *)
```

Initiator sends its token

While not leader, keep receiving tokens

*If my token comes, then I **won***

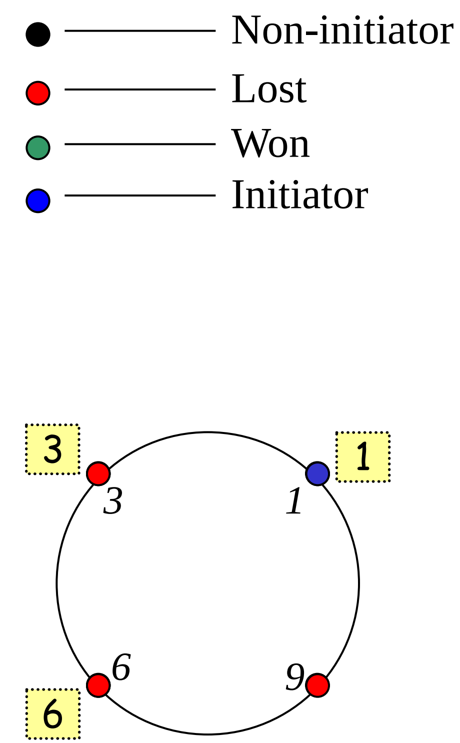
*Otherwise, only forward if it's lower id than me and **lose***

*Non-initiators just forward and **lose***

Chang Roberts - Example

- Nodes 1, 3, 6 are initiators

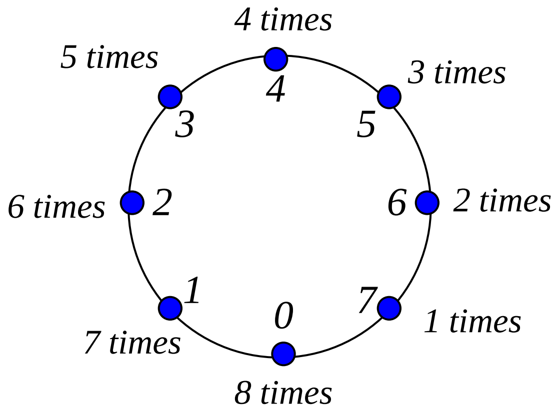
```
var state_p ;

begin if p is initiator then
        begin state_p := cand ; send ⟨tok, p⟩ to Next_p ;
              while state_p ≠ leader do
                    begin receive ⟨tok, q⟩ ;
                          if q = p then state_p := leader
                          else if q < p then
                                begin if state_p = cand then state_p := lost ;
                                      send ⟨tok, q⟩ to Next_p
                                end
                    end
        end
     else while true do
             begin receive ⟨tok, q⟩ ; send ⟨tok, q⟩ to Next_p ;
                   if state_p = sleep then state_p := lost
             end
end
```



Chang Roberts Analysis

- Worst case complexity same as LeLann's
  - Time Complexity: $2N-1$
  - Message Complexity: $O(N^2)$
    - Considered a sorted ring with N initiators

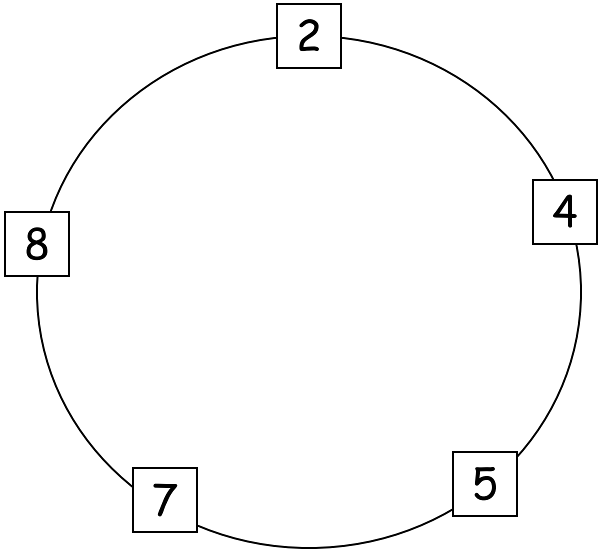
$$\sum_{i=0}^{N-1}(N-i) = N - \sum_{i=0}^{N-1} i = N - \frac{(N-1)N}{2} = \frac{(N+1)N}{2}$$



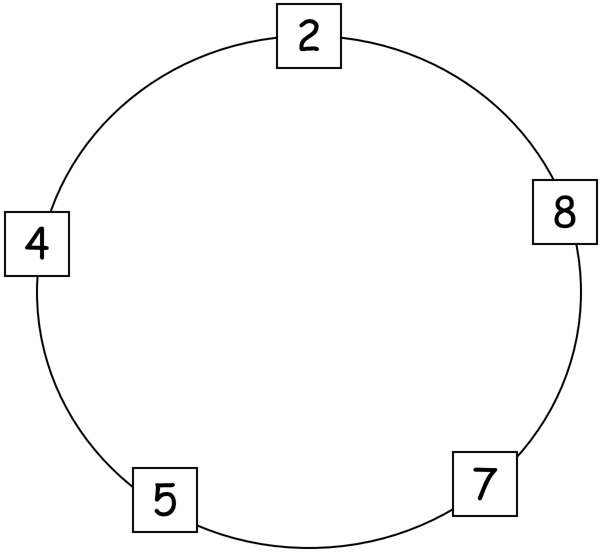
Chang-Roberts Algorithm: Best/Worst Performance

For distributed systems, communication is the bottleneck. Performance thus is often described as message complexity.

- Best case:
  - 2n-1 messages



- Worst case:
  - (n-1)/2 messages



Further improvements?

var $state_p$;

begin if p is initiator **then**
    begin $state_p := cand$; send $\langle\, \mathbf{tok}, p \,\rangle$ to $Next_p$;
        while $state_p \neq leader$ **do**
            begin receive $\langle\, \mathbf{tok}, q \,\rangle$;

Break the loop if state!=cand to avoid blocking receive

                if $q = p$ **then** $state_p := leader$

Do not just compare with your id, compare with the lowest id you have seen!

                else if $q < p$ **then**
                    begin if $state_p = cand$ **then** $state_p := lost$;
                        send $\langle\, \mathbf{tok}, q \,\rangle$ to $Next_p$
                    end
            end
    end
  else while $true$ **do**

Let the non-initiators also filter lower ids

      begin receive $\langle\, \mathbf{tok}, q \,\rangle$; send $\langle\, \mathbf{tok}, q \,\rangle$ to $Next_p$;
          if $state_p = sleep$ **then** $state_p := lost$
      end
end
(* Only the leader terminates the program. It floods a message to all processes to inform them of the leader's identity and to terminate *)

Summary

- Deterministic election algorithms
  - For rings, LeLanns and Chang/Roberts
- Anonymous networks
  - Network size can be computed if leader exists
  - Unique names can be distributed if leader exists
  - Impossible to construct a deterministic election algorithm without a leader
  - Probabilistic algorithms using random numbers
    - Using the tree algorithm to elect a leader

Relationship to Mutual Exclusion

- Like Mutual exclusion
  - Only, one the leader enters the critical section
- Unlike Mutual exclusion
  - Do not worry about fairness
  - Must inform others
  - Need to worry about coordinator failing

Faults

- Nodes cooperate
- System software works
- Every message recv'ed was sent
- Nodes have safe-storage
- Nodes can only halt
- Messages are not corrupted
- Messages are FIFO ordered
- Messages have time-out (considered failed)
- Nodes respond immediately

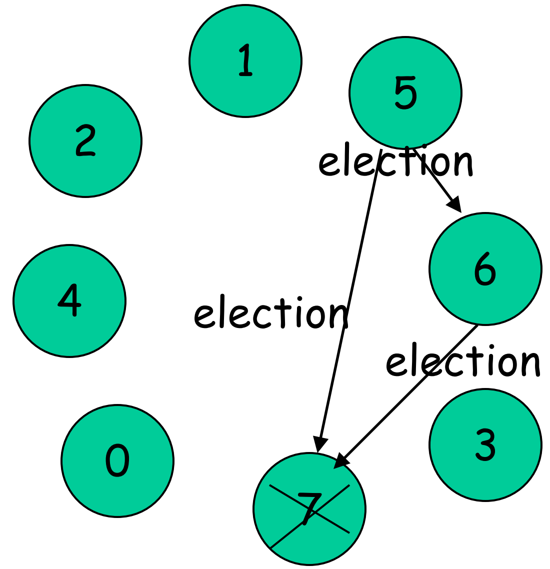
Bully Algorithm

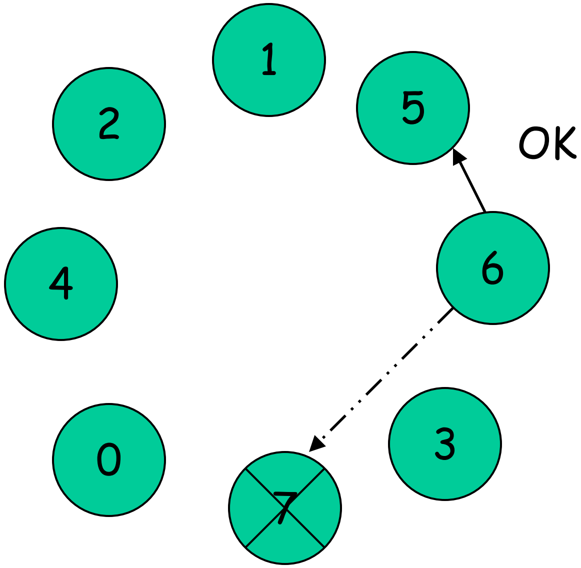
1. Send *election* message *(I want to be the leader)* to processes with *larger id*
2. Give up your bid if a process with *larger id* sends a *reply* message *(means no, you cannot be the leader).* In that case, wait for the *leader* message (*I am the leader*). Otherwise elect yourself the leader and send a *leader* message
3. If *no reply is received*, then elect yourself the leader, and broadcast a *leader* message.
4. If you receive a reply, but later don't receive a *leader* message from a process of larger id (i.e the leader-elect has crashed), then re-initiate election by sending *election* message.

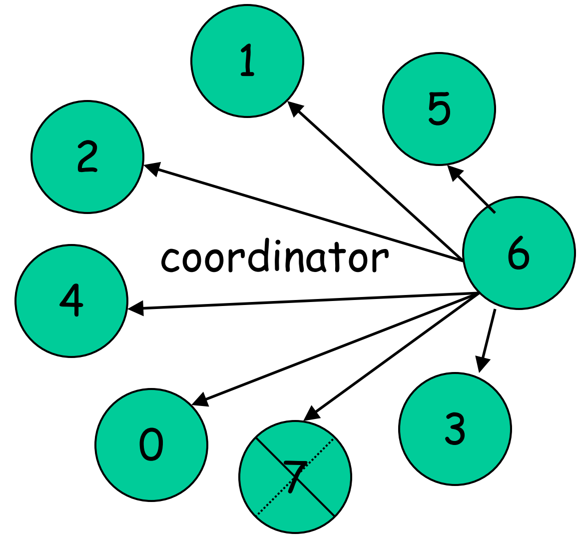
The process *q* now calls an election (if it has not already done so).

Repeat until no higher-level process responds. The last process to call an election "wins" the election.

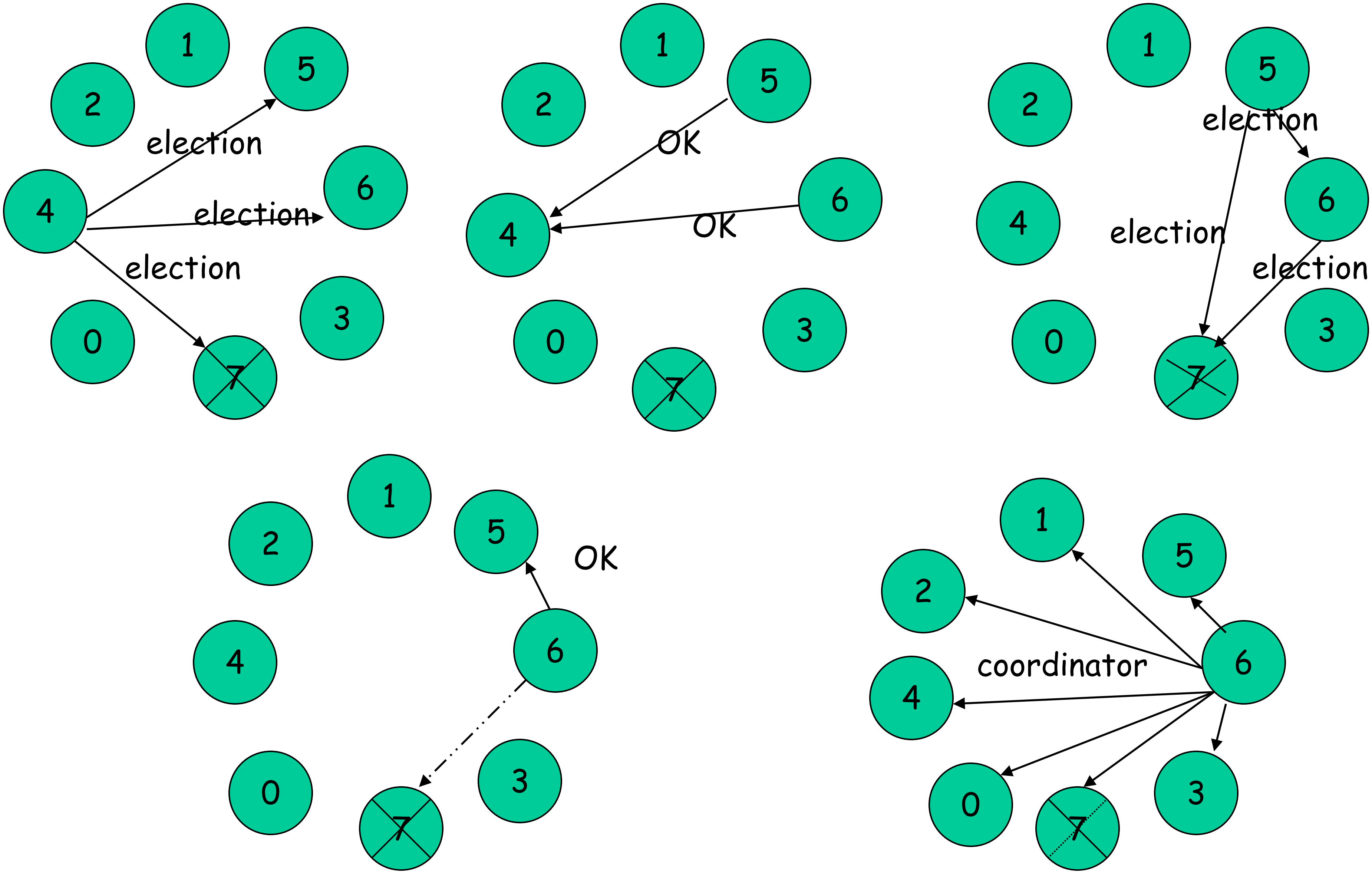
The winner sends a message to other processes announcing itself as the new coordinator.







If 7 comes back on line, it will call an election



election
election
election
OK
OK
election
election
election
OK
coordinator