# Lecture 21

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

# Motivation

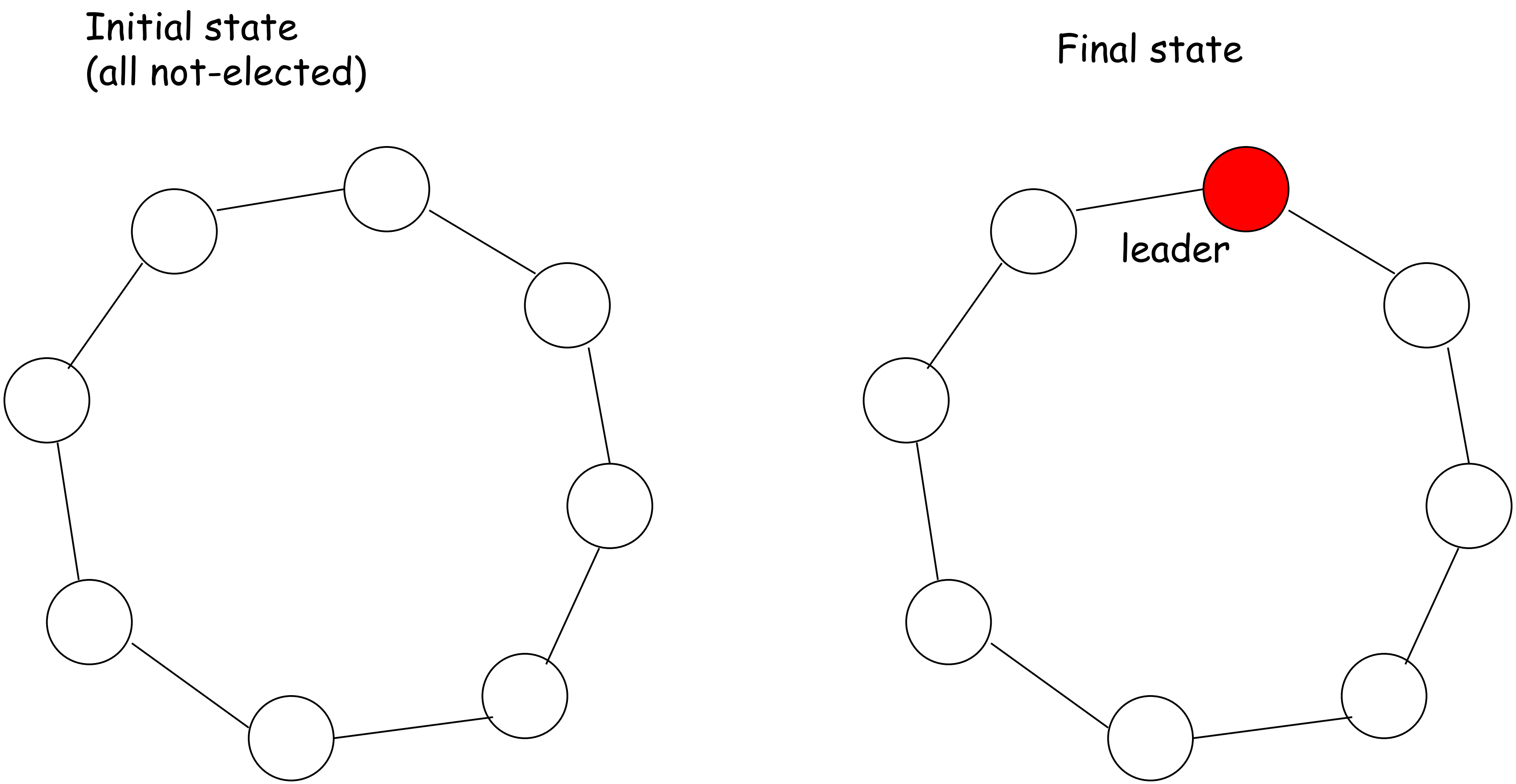
- We often need a coordinator in distributed systems
  - Leader, distinguished node/process

- If we have a leader, mutual exclusion is trivially solved
  - The leader determined who enters CS
  - Inefficient though…

# Election Algorithms

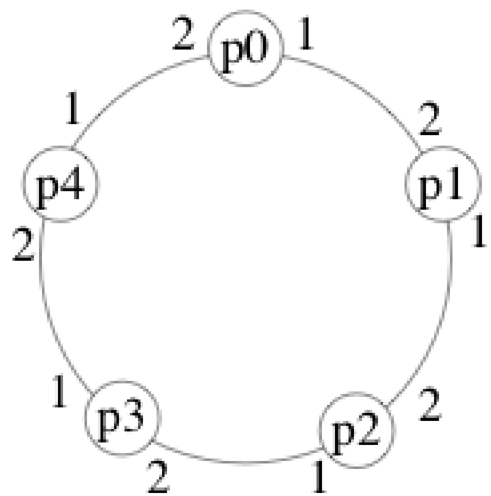
- Any process can serve as coordinator
- Any process can "call an election" (initiate the algorithm to choose a new coordinator).
  - There is no harm (other than extra message traffic) in having multiple concurrent elections.
- Elections may be needed when the system is initialized, or if the coordinator crashes or retires.

# Leader Election in Ring Networks



# Sense-of-direction in Rings

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




1 = left = clockwise

2 = right = counter-clockwise

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

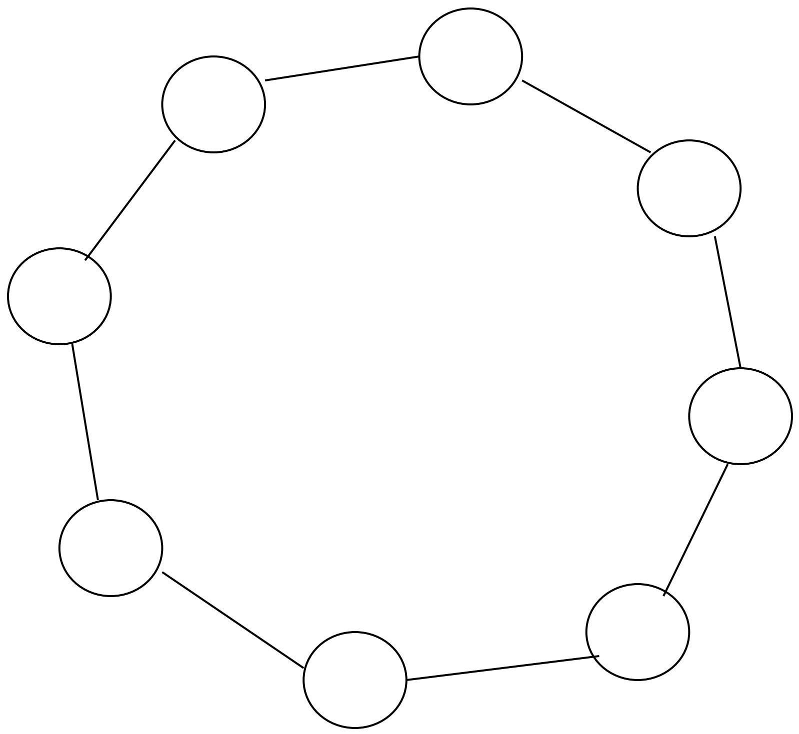
# Properties of Rings and Algorithms

Anonymous Ring (no identifiers)
Non-anonymous Ring

Size of the network n is known (non-uniform)
Size of the network n is not known (uniform)

Synchronous Algorithm
Asynchronous Algorithm

# Leader Election in Anonymous Rings



Every processor runs 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)
  - in the synchronous model
- **Proof Sketch (for non-uniform and synchronous rings):**
  - Every processor begins in same state (not-elected) with same outgoing msgs (since anonymous)
  - Every processor receives same msgs, does same state transition, and sends same msgs in round 1
  - And so on and so forth for rounds 2, 3, …
  - Eventually some processor is supposed to enter an elected state. But then they all would.

# Definition of an Election Algorithm

- Formally, an algorithm is an election algorithm if and only if:
  - Each process has the same local algorithm
  - The algorithm is decentralized
    - It can be initiated by any number of processes in the system
  - It reaches a terminal configuration, and in each reachable terminal configuration one process is in state *leader* and the rest are in the state *lost*

# Assumptions

- Fully asynchronous system
  - No global clock, transmission times arbitrary
- Every process has a unique identity
  - Identities are totally ordered
  - Because we have finite number of processes:
    - *Max* identity and *Min* identity available (extreme values)

# 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

# LeLann's Algorithm

Initially only know myself

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

*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*