Distributed Systems

CS425/ECE428

Feb 22 2023

Instructor: Radhika Mittal

Acknowledgements: Indy Gupta and Nikita Borisov

Today's agenda

- **Leader Election**
  - Chapter 15.3
- **Goal:**
  - What is leader election in distributed systems?
  - How do we elect a leader?
  - To what extent can we handle failures when electing a leader?

Examples of leader election?

- The root server in a group of NTP servers.
- The master in Berkeley algorithm for clock synchronization.
- In the sequencer-based algorithm for total ordering of multicasts, the "sequencer" = leader.
- The central server in the "central server algorithm" for mutual exclusion.
- Other systems that need leader election: Apache Zookeeper, Google's Chubby.

Why Election?

- Example: Your Bank account details are replicated at a few servers, but one of these servers is responsible for receiving all reads and writes, i.e., it is the leader among the replicas
  - What if servers disagree about who the leader is?
  - What if there are two leaders per customer?
  - What if the leader crashes?

    Each of the above scenarios leads to inconsistency

Leader Election Problem

- In a group of processes, elect a *Leader* to undertake special tasks
  - And *let everyone know* in the group about this Leader
- What happens when a leader fails (crashes)
  - Some process detects this (using a Failure Detector!)
  - Then what?
- Focus of this lecture: Election algorithm. Its goal:
  1. Elect one leader only among the non-faulty processes
  2. All non-faulty processes agree on who is the leader

Calling for an Election

- Any process can call for an election.
- A process can call for at most one election at a time.
- Multiple processes are allowed to call an election simultaneously.
  - All of them together must yield only a single leader
- The result of an election should not depend on which process calls for it.

Election Problem, Formally

- A run of the election algorithm must always guarantee:
  - **Safety**: For all non-faulty processes *p*:
    - *p* has elected:
      - (q: a particular non-faulty process with the *best attribute value*)
      - or Null
  - **Liveness**: For all election runs:
    - election run terminates
    - & for all non-faulty processes *p*: *p*'s elected is not Null
- At the end of the election protocol, the non-faulty process with the best (highest) election attribute value is elected.
  - Common attribute : leader has highest id
  - Other attribute examples: leader has highest IP address, or fastest cpu, or most disk space, or most number of files, etc.

System Model

- *N* processes.
- Messages are eventually delivered.
- Failures may occur during the election protocol.
- **Each process has a unique id.**
  - Each process has a unique attribute (based on which Leader is elected).
  - If two processes have the same attribute, combine the attribute with the process id to break ties.

Classical Election Algorithms

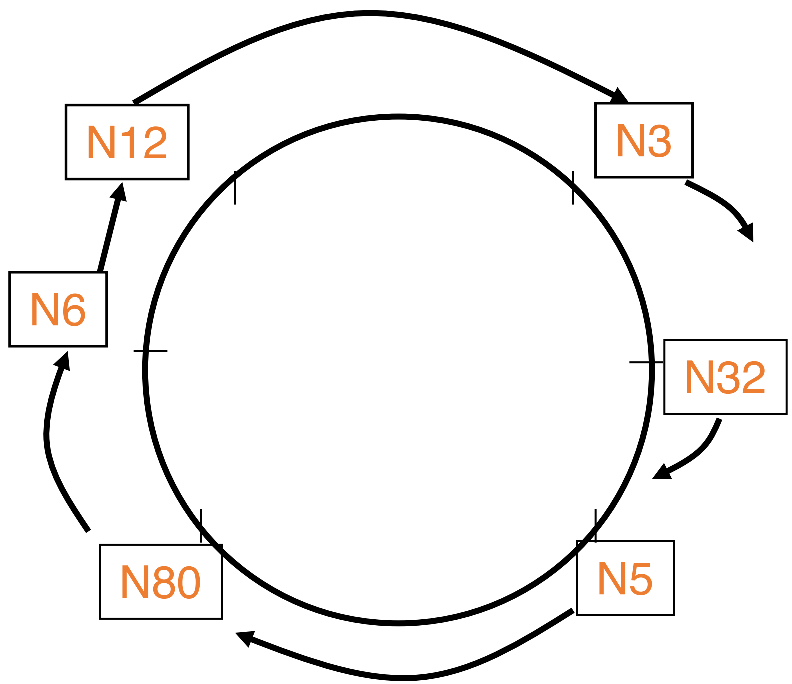
- Ring election algorithm
- Bully algorithm

Classical Election Algorithms

- **Ring election algorithm**
- Bully algorithm

Ring Election Algorithm

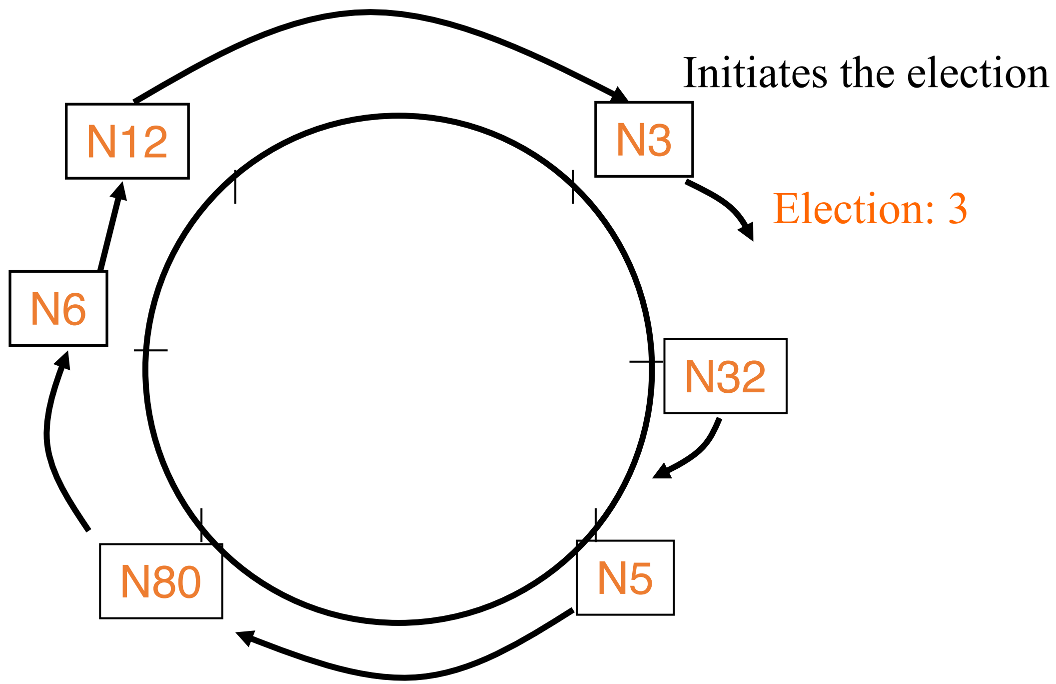
- *N* processes are organized in a logical ring
  - All messages are sent clockwise around the ring.



Ring Election Protocol (basic version)

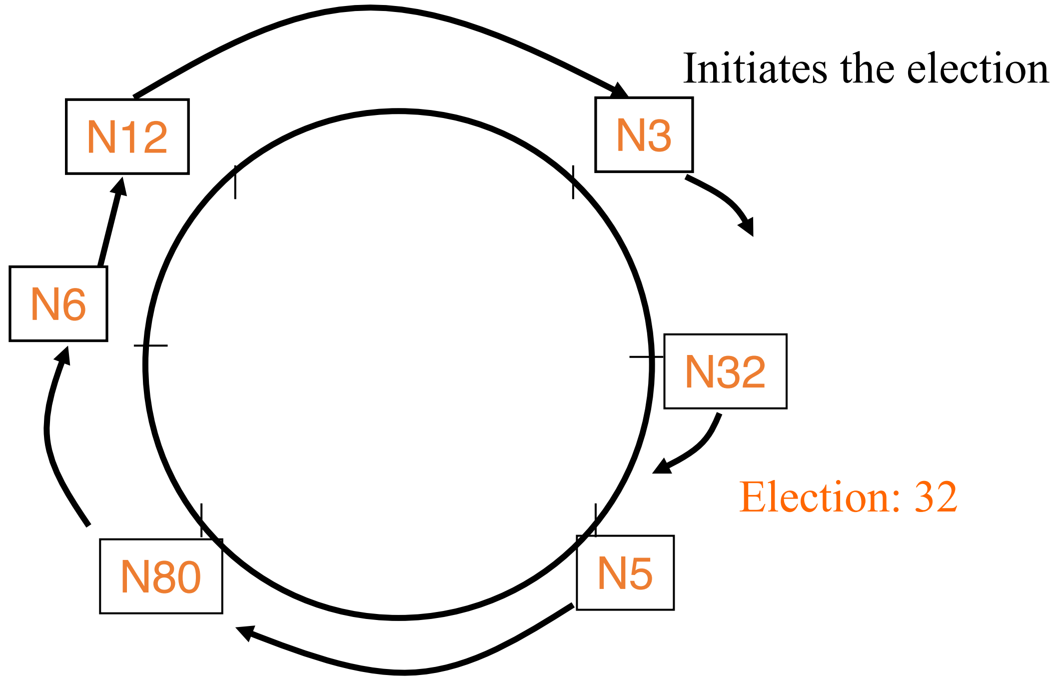
- When Pi start election
  - send election message with Pi's $<attr_i, i>$ to ring successor.
- When Pj receives message (election, $<attr_x, x>$) from predecessor
  - If $(attr_x, x) > (attr_j, j)$:
    - forward message (election, $<attr_x, x>$) to successor
  - If $(attr_x, x) < (attr_j, j)$
    - send (election, $<attr_j, j>$) to successor
  - If $(attr_x, x) = (attr_j, j)$: Pj is the elected leader (why?)
    - send elected message containing Pj's id.
- elected message forwarded along the ring until it reaches the leader.

Ring Election: Example



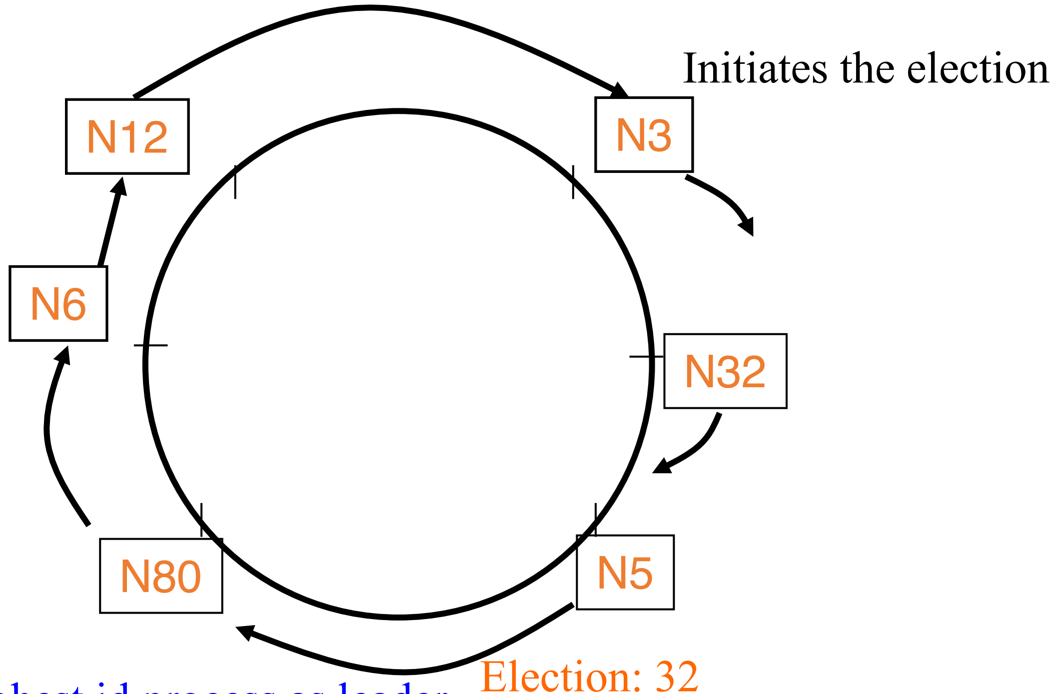
Goal: Elect highest id process as leader

Ring Election: Example



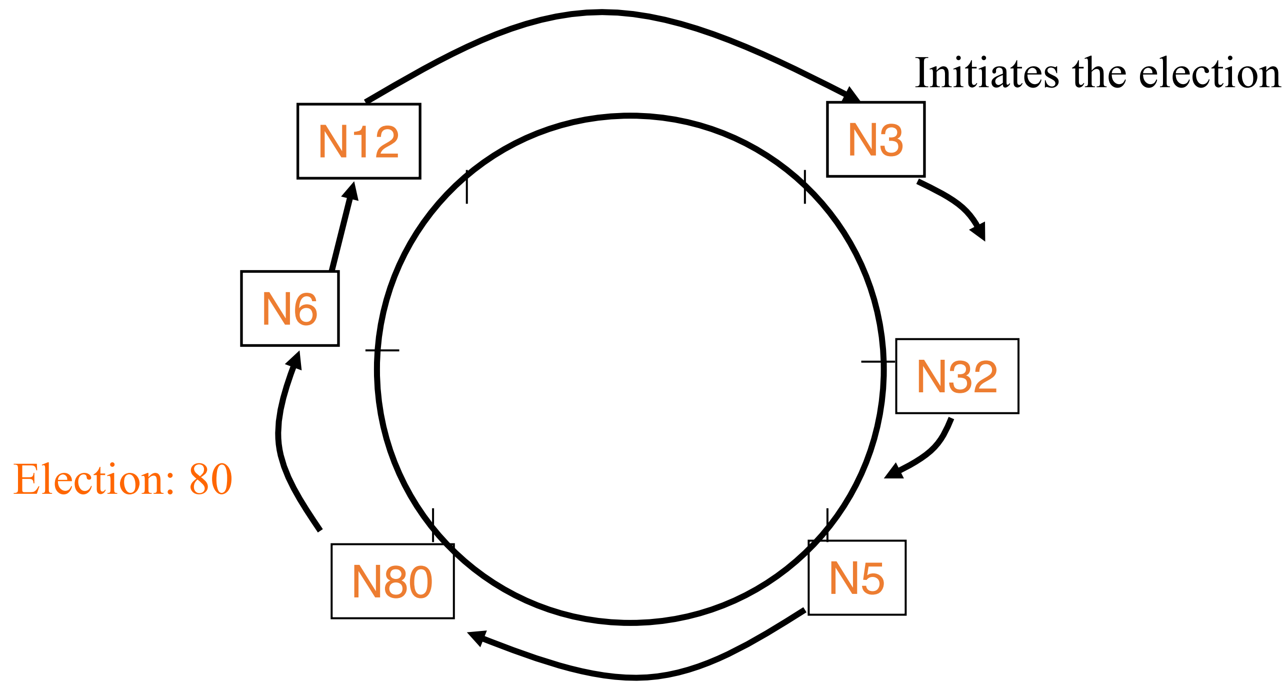
Goal: Elect highest id process as leader

Ring Election: Example



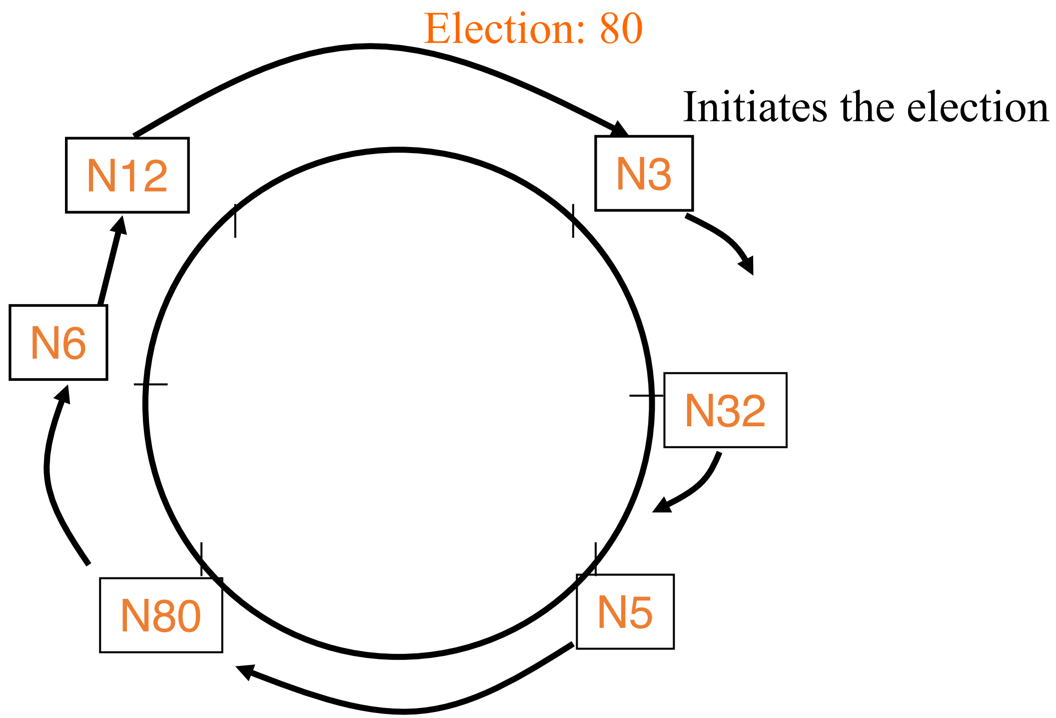
Goal: Elect highest id process as leader

Ring Election: Example



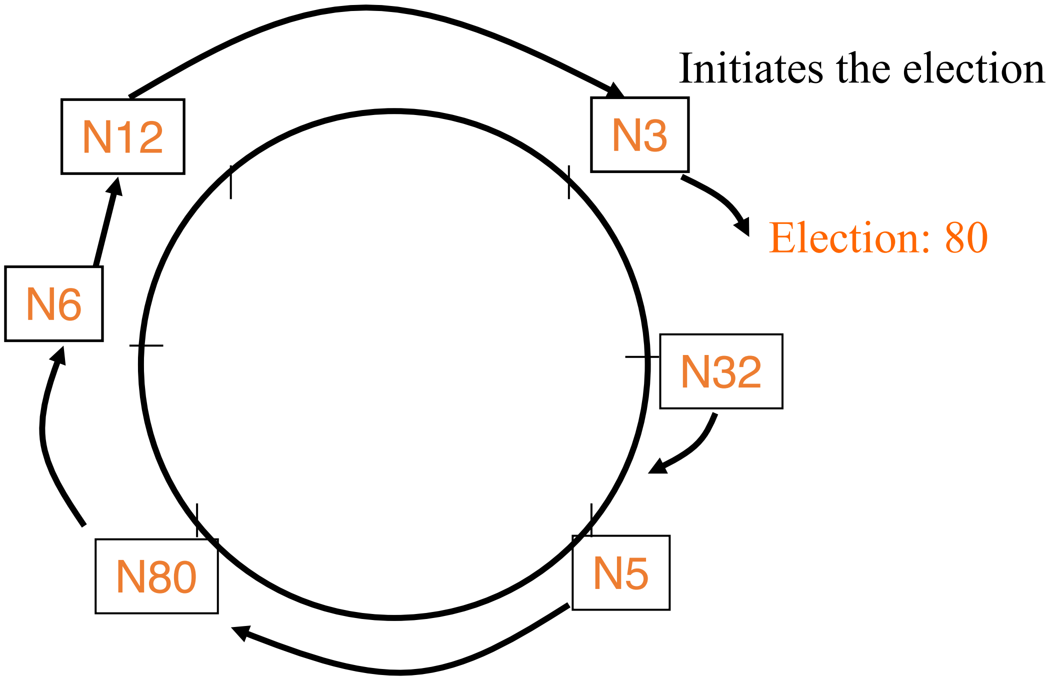
Goal: Elect highest id process as leader

Ring Election: Example



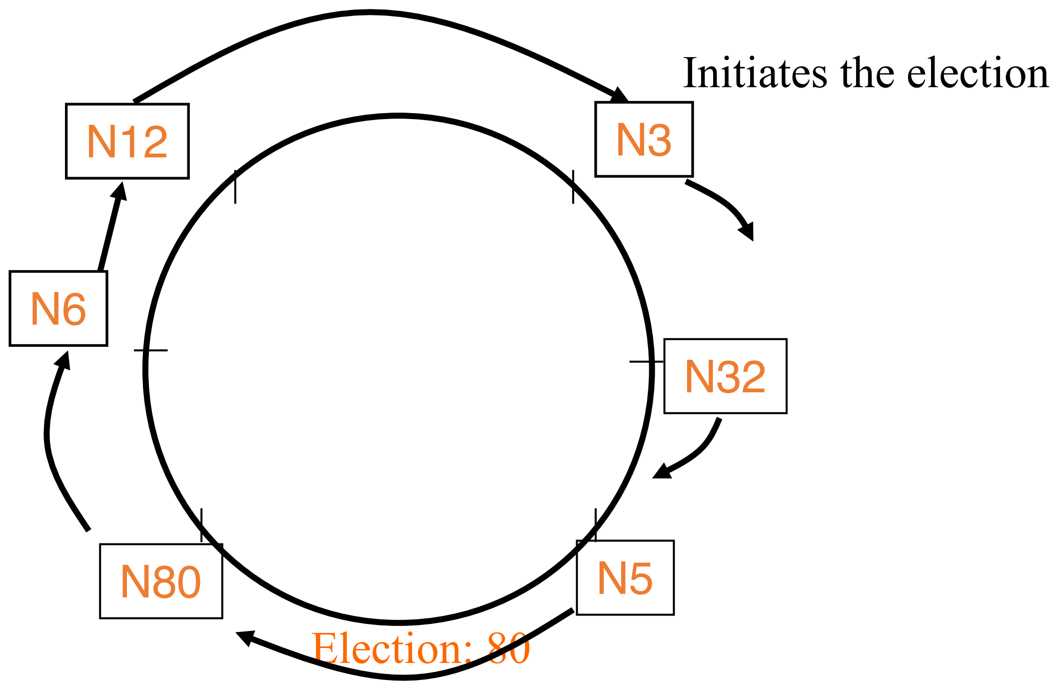
Goal: Elect highest id process as leader

Ring Election: Example



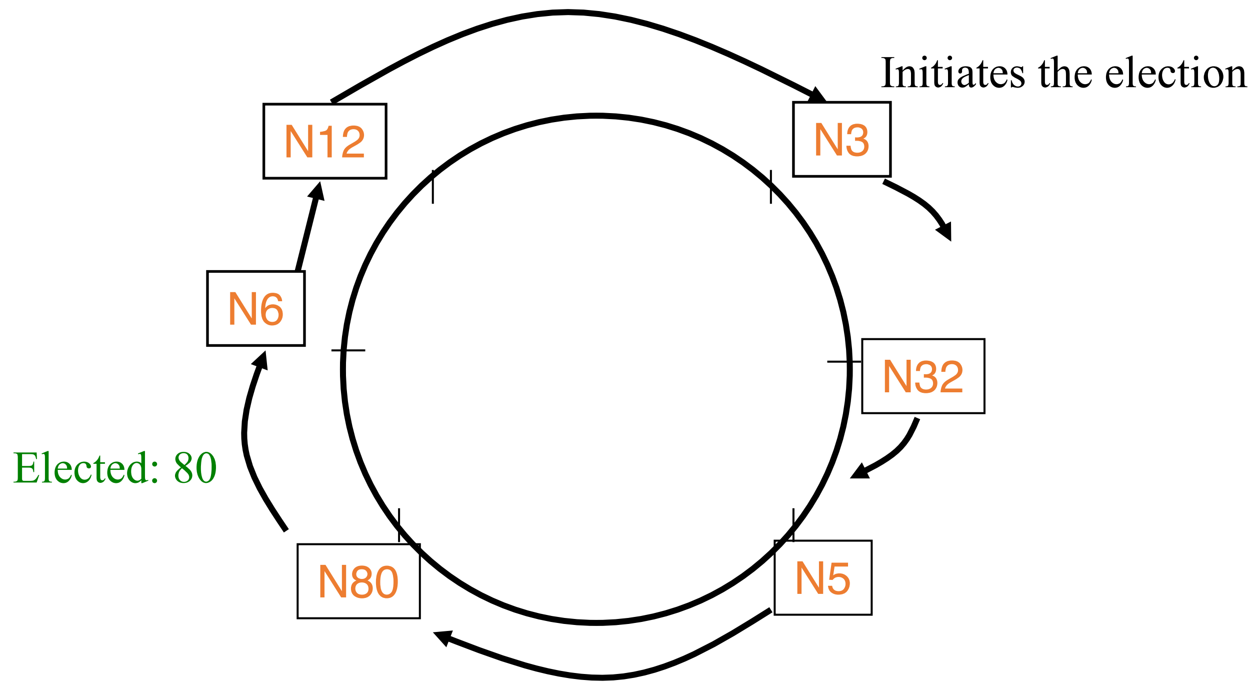
Goal: Elect highest id process as leader

Ring Election: Example



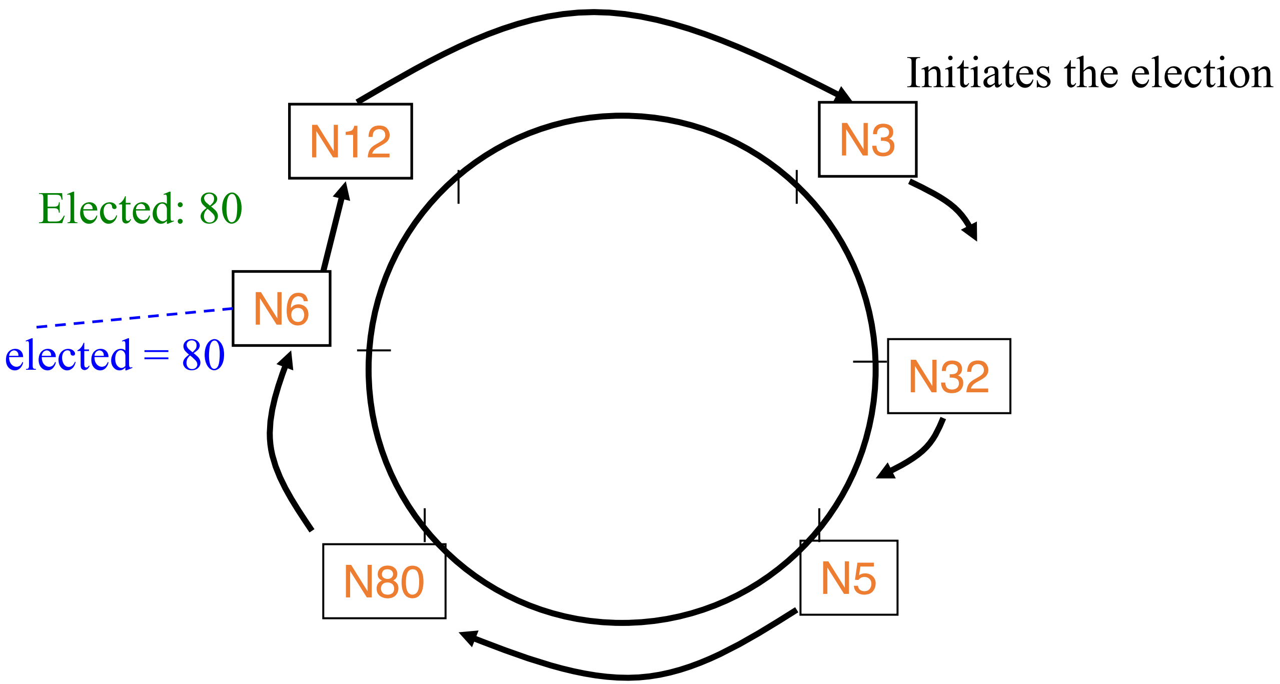
Goal: Elect highest id process as leader

Ring Election: Example



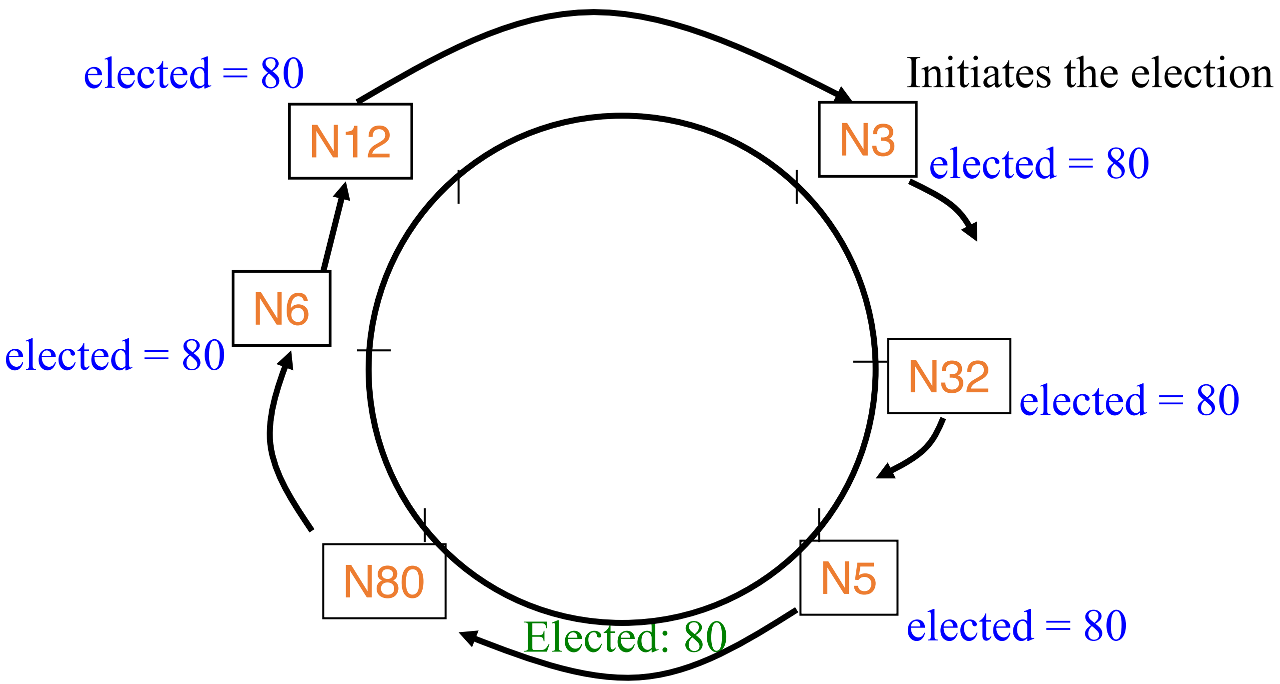
Goal: Elect highest id process as leader

Ring Election: Example



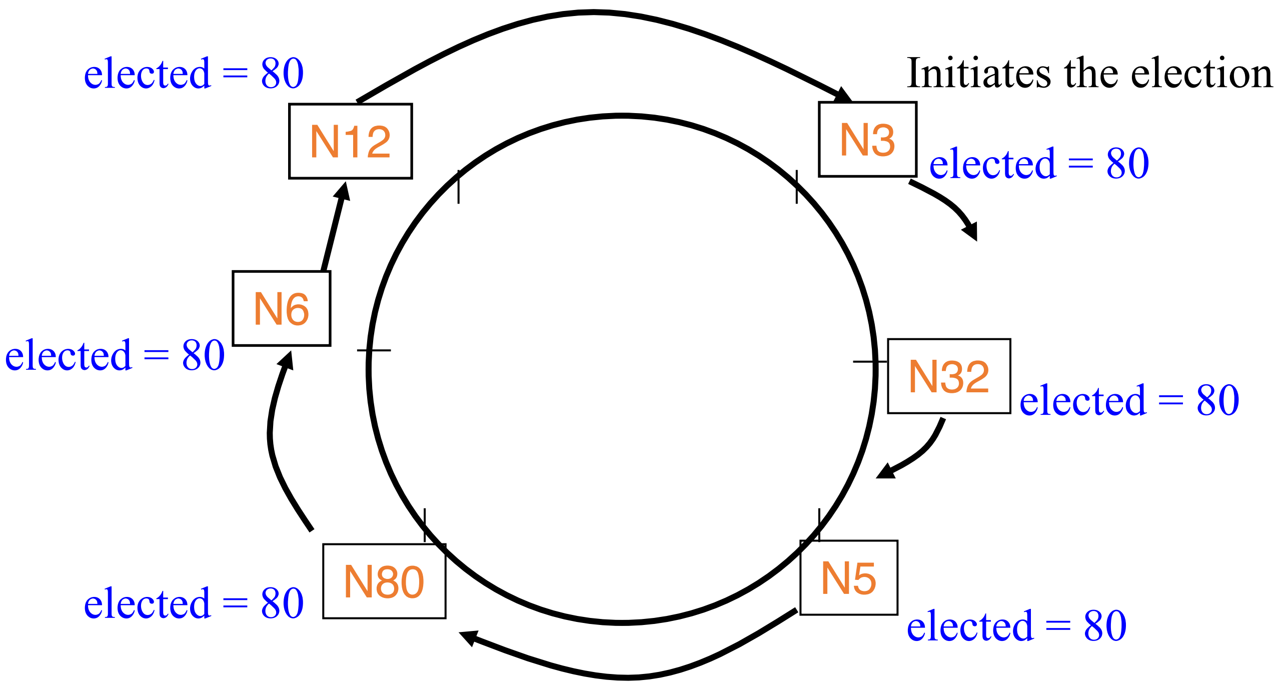
Goal: Elect highest id process as leader

Ring Election: Example



Goal: Elect highest id process as leader

Ring Election: Example



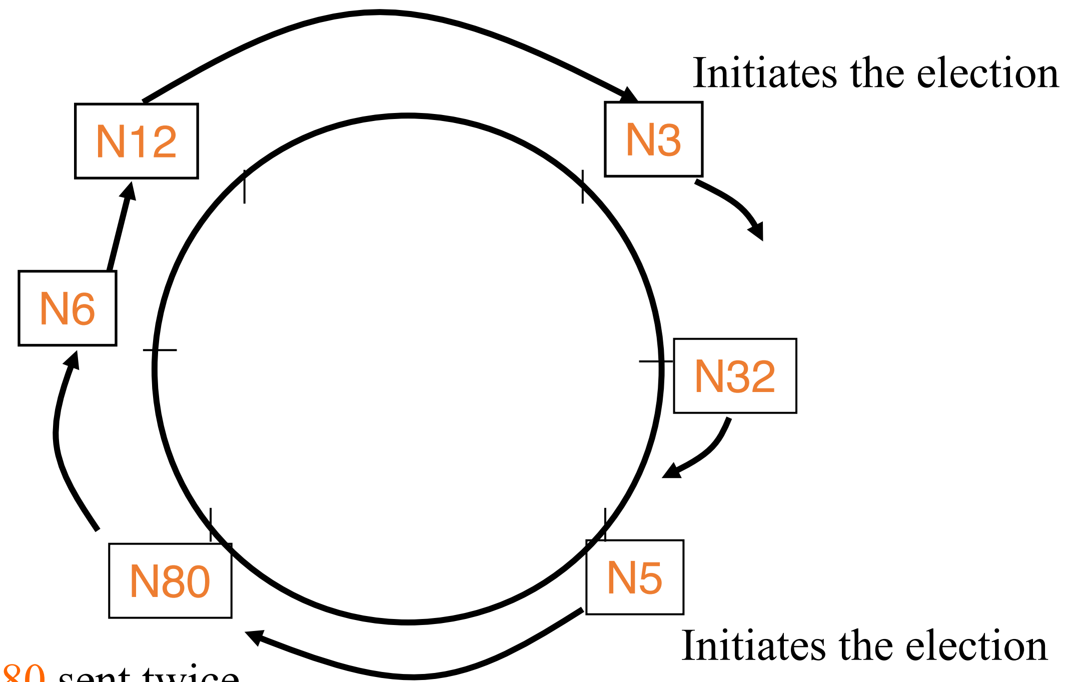
Goal: Elect highest id process as leader

Ring Election Protocol (basic version)

- When Pi start election
  - send election message with Pi's <$attr_i$, i> to ring successor.
- When Pj receives message (election, <$attr_x$, x>) from predecessor
  - If $(attr_x, x) > (attr_j, j)$:
    - forward message (election, <$attr_x$, x>) to successor
  - If $(attr_x, x) < (attr_j, j)$
    - send (election, <$attr_j$, j>) to successor
  - If $(attr_x, x) = (attr_j, j)$: Pj is the elected leader (why?)
    - send elected message containing Pj's id.
- elected message forwarded along the ring until it reaches the leader.

What happens when multiple processes call for an election?

Ring Election: Example

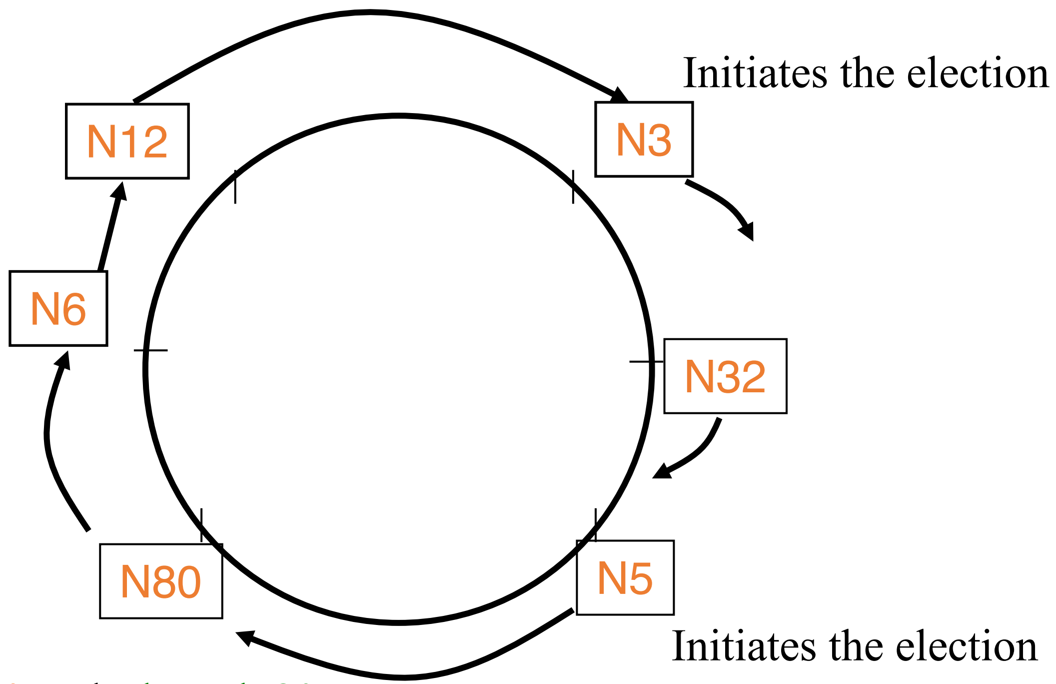


Election: 80 sent twice.
Elected: 80 also sent twice.

Ring Election Protocol [Chang & Roberts'79]

- When Pi start election
  - send election message with Pi's $<attr_i, i>$ to ring successor.
  - set state to participating
- When Pj receives message (election, $<attr_x, x>$) from predecessor
  - If $(attr_x, x) > (attr_j, j)$:
    - forward message (election, $<attr_x, x>$) to successor
    - set state to participating
  - If $(attr_x, x) < (attr_j, j)$
    - If (not participating):
      - send (election, $<attr_j, j>$) to successor
      - set state to participating
  - If $(attr_x, x) = (attr_j, j)$: Pj is the elected leader (why?)
    - send elected message containing Pj's id.
- elected message forwarded along the ring until it reaches the leader.
  - Set state to not participating when an elected message is received.

Ring Election: Example

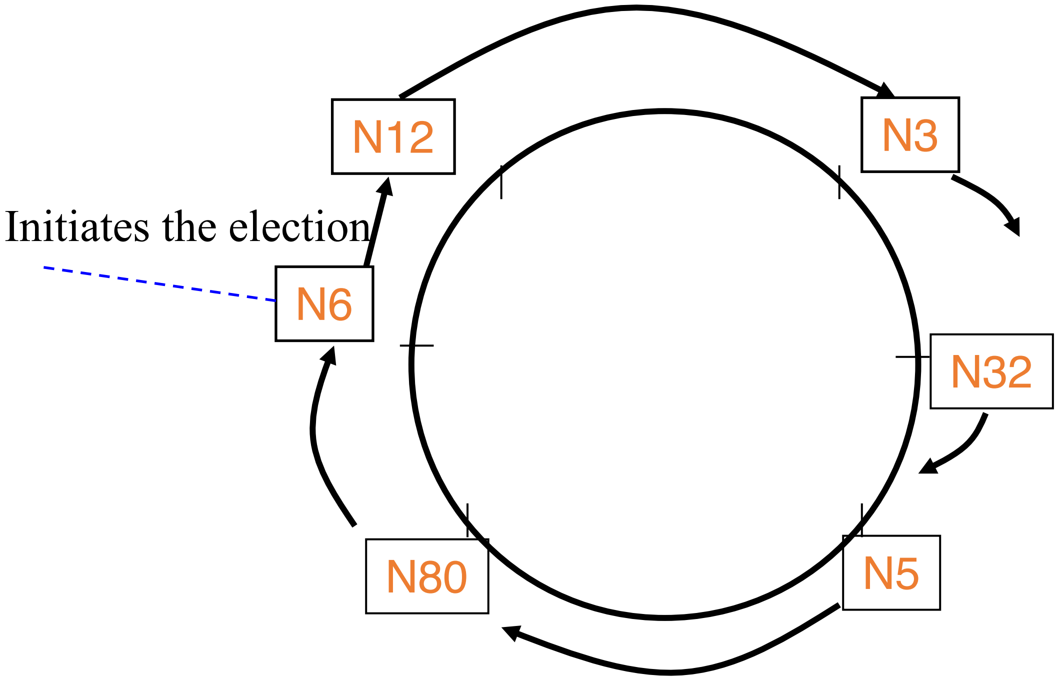


Election: 80 and Elected: 80
sent only once.

Performance Analysis

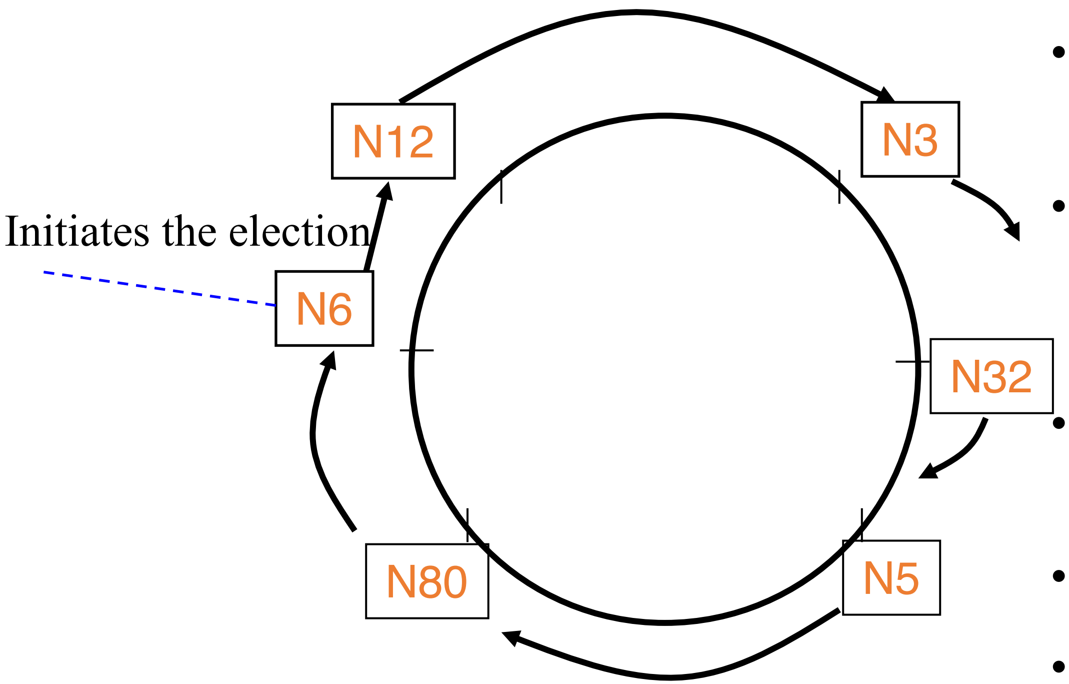
- Let's assume no failures occur during the election protocol itself, and there are *N* processes.
- Let's also assume that only one process initiates the algorithm
- **Bandwidth usage:** Total number of messages sent.
- **Turnaround time:** The number of serialized message transmission times between the initiation and termination of a single run of the algorithm.

Worst-case



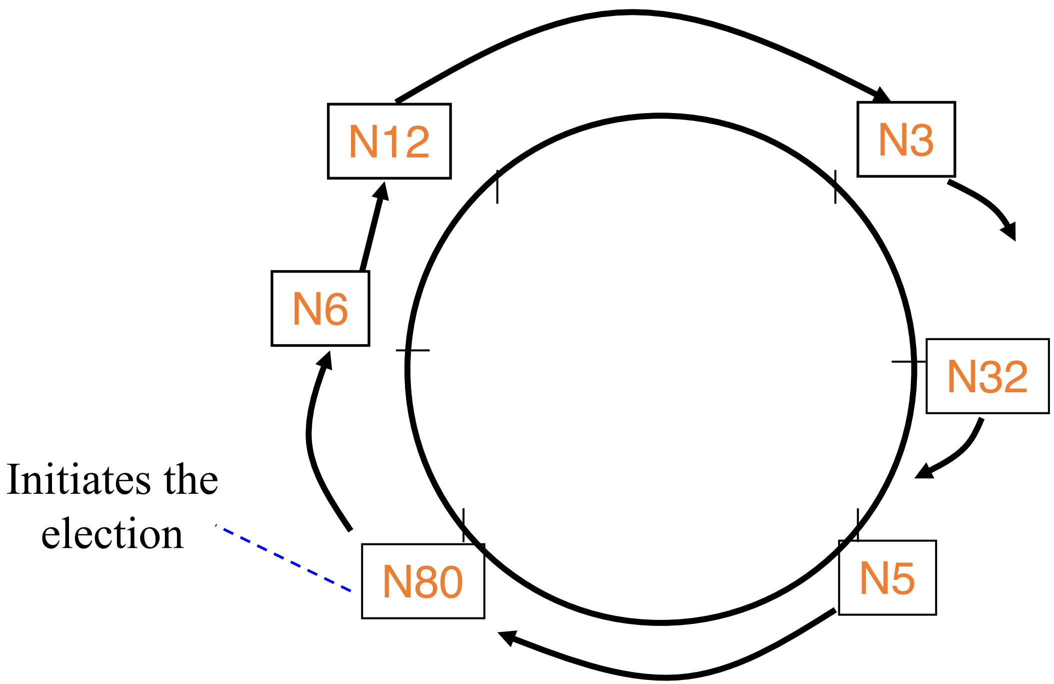
When the initiator is the ring successor of the would-be leader.

Worst-case



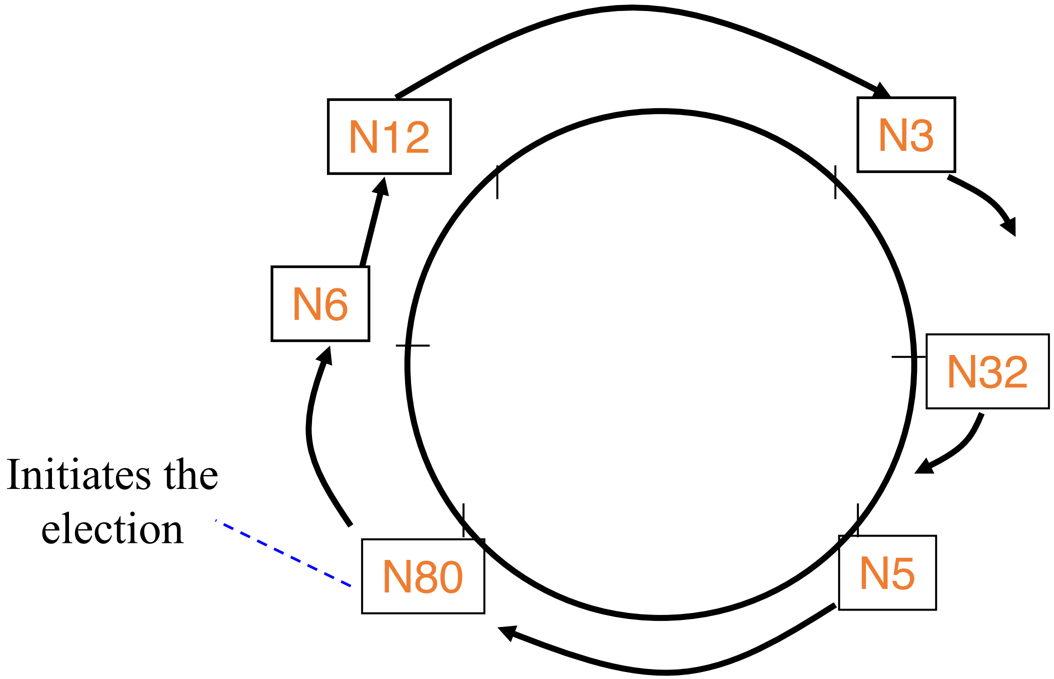
- (N-1) messages for Election message to get from N6 to N80.
- N messages for Election message to circulate around ring without message being changed.
- N messages for Elected message to circulate around the ring
- **No. of messages: (3N-1)**
- **Turnaround time: (3N-1) message transmission times**

Best-case



When the initiator is the would-be leader.

Best-case



When the initiator is the would-be leader.

No. of messages: 2N

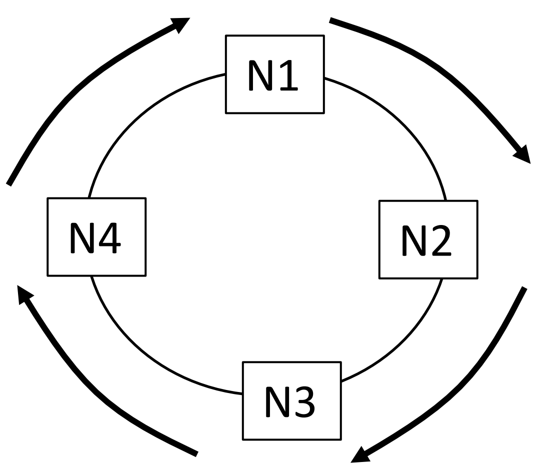
Turnaround time:
2N message transmission times

Performance Analysis

- Let's assume no failures occur during the election protocol itself, and there are *N* processes.
- Let's also assume that only one process initiates the algorithm
- Bandwidth usage (total number of messages)
  - O(N): Worst case = 3N -1; Best case = 2N.
- O(N) turnaround time.

Performance Analysis

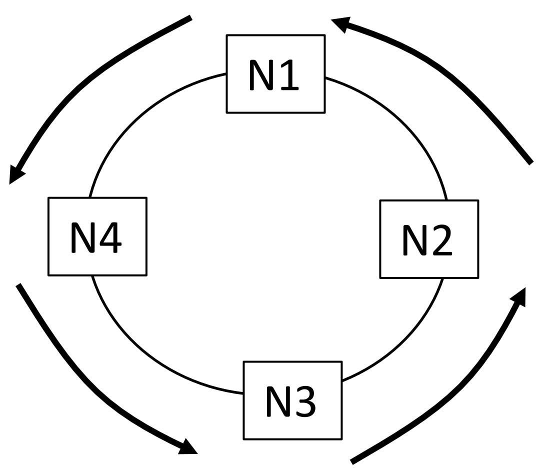
- Let's assume no failures occur during the election protocol itself, and there are *N* processes.
- When each process initiates the algorithm?
  - O(N) messages in best-case.



- N election messages generated at the start of algorithm.
- Only one survives, and completes a full round.
  - N-1 more messages.
- One round for the elected message
  - N messages.
- Total: 3N-1 messages

Performance Analysis

- Let's assume no failures occur during the election protocol itself, and there are *N* processes.
- When each process initiates the algorithm?
  - O(N) messages in best-case.
  - $O(N^2)$ in worst-case.



- N election messages generates at the starts of algorithm.
- N - 1 survive the next time step.
- N – 2 survive the next time step.
- ….

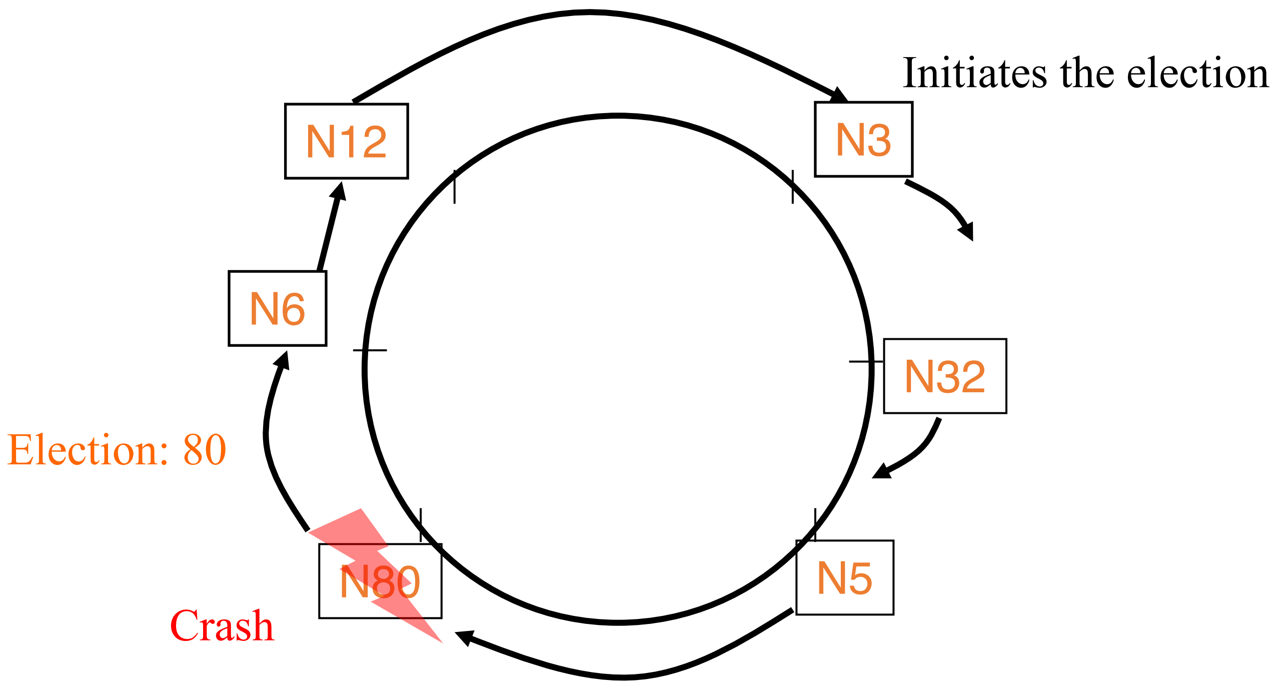
Performance Analysis

- Let's assume no failures occur during the election protocol itself, and there are *N* processes.
- When each process initiates the algorithm?
  - O(N) messages in best-case.
  - $O(N^2)$ messages in worst-case.
  - O(N) turnaround time.

Correctness

- Assuming no process fails.
- Safety:
  - Process with highest attribute elected by all nodes.
- Liveness:
  - Election completes within 3N – 1 message transmission times.

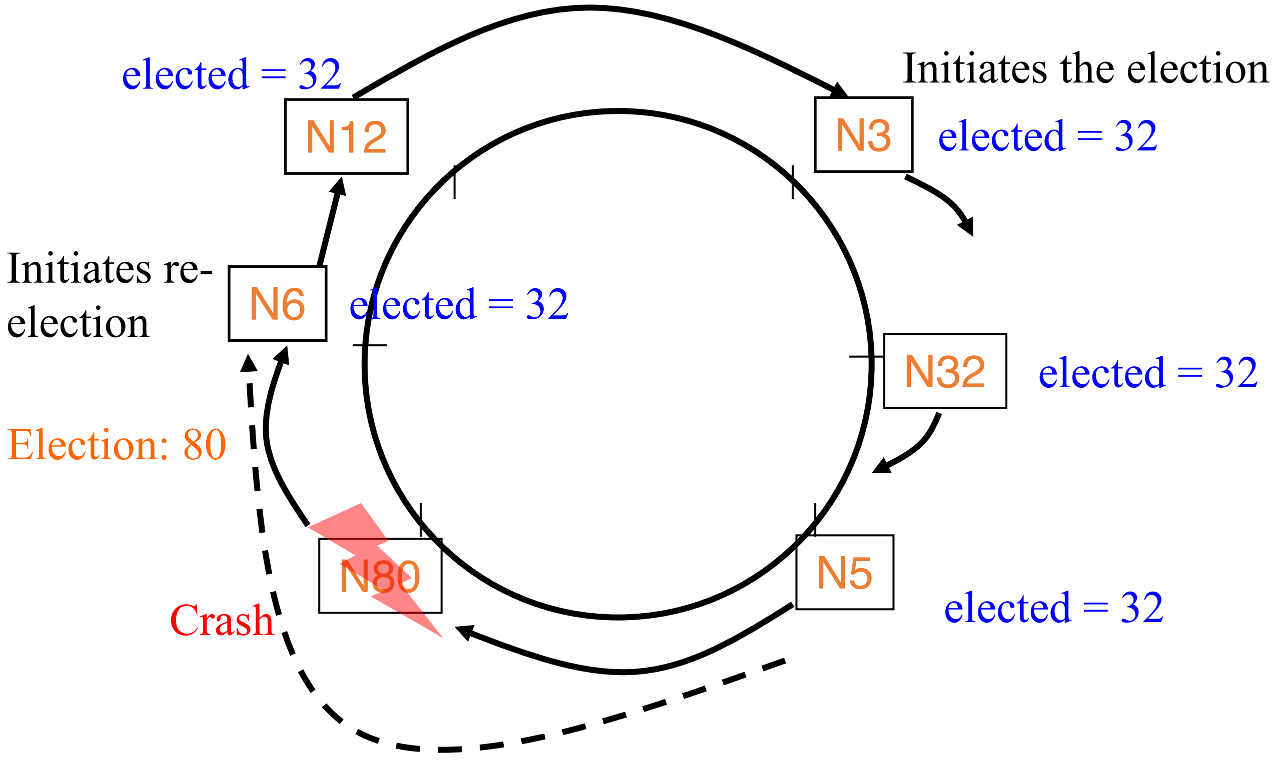
Handling Failures



Handling failures

- Use the failure detector.
- A process can detect failure of N80 via its own local failure detector:
  - Repair the ring.
  - Stop forwarding Election:80 message.
  - Start a new run of leader election.

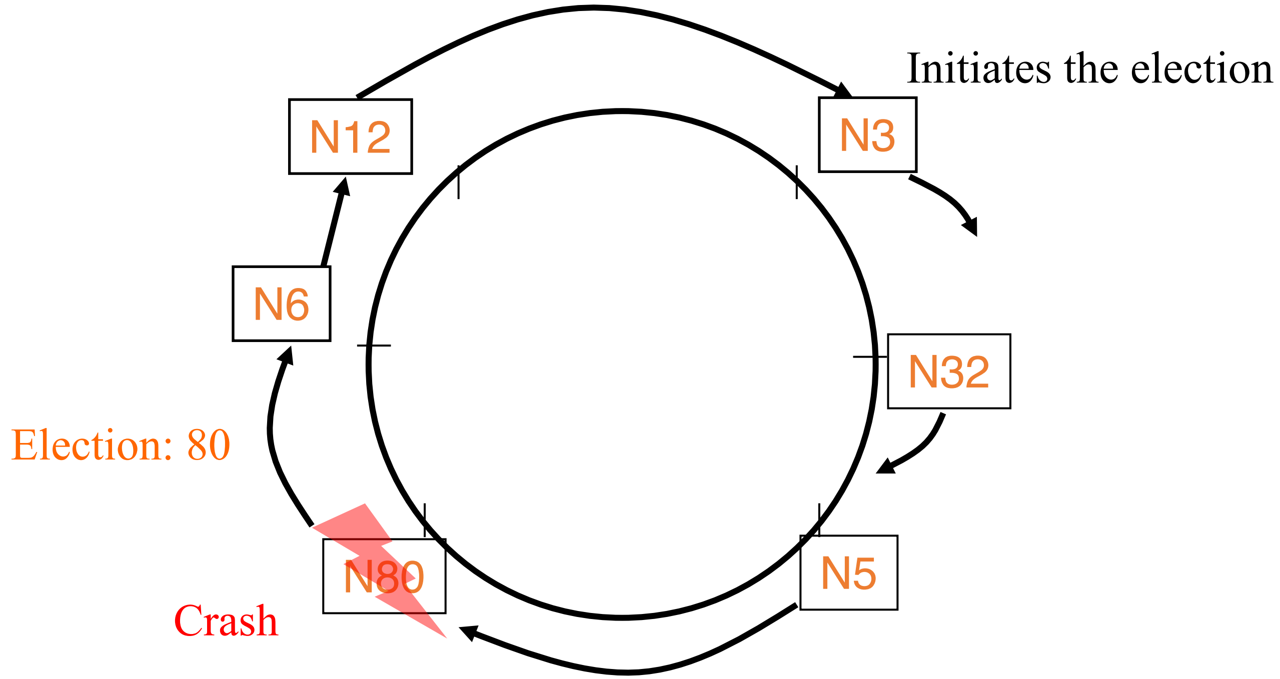
Handling Failures



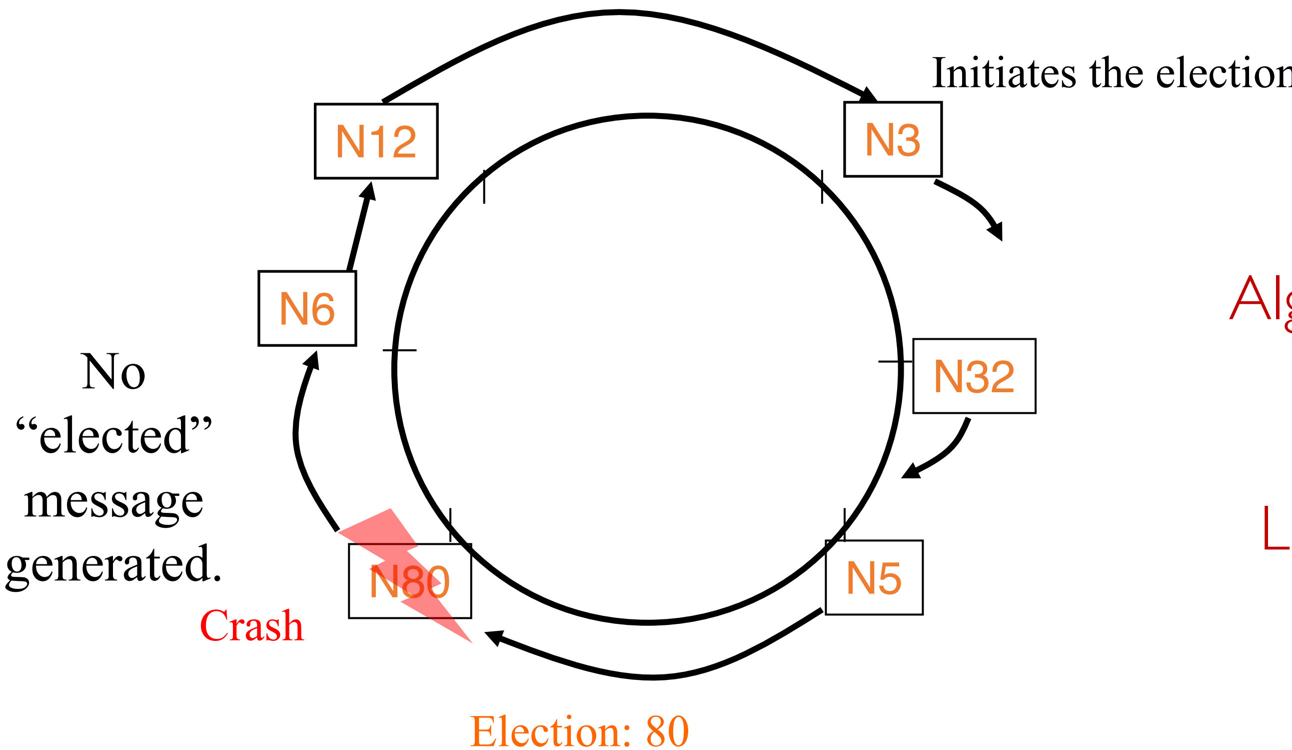
Handling failures

- Use the failure detector.
- A process can detect failure of N80 via its own local failure detector:
  - Repair the ring.
  - Stop forwarding Election:80 message.
  - Start a new run of leader election.
- But failure detectors cannot be both complete and accurate.
  - Incomplete FD => N80's failure might be missed .

What happens if a process failure is undetected?



What happens if a process failure is undetected?



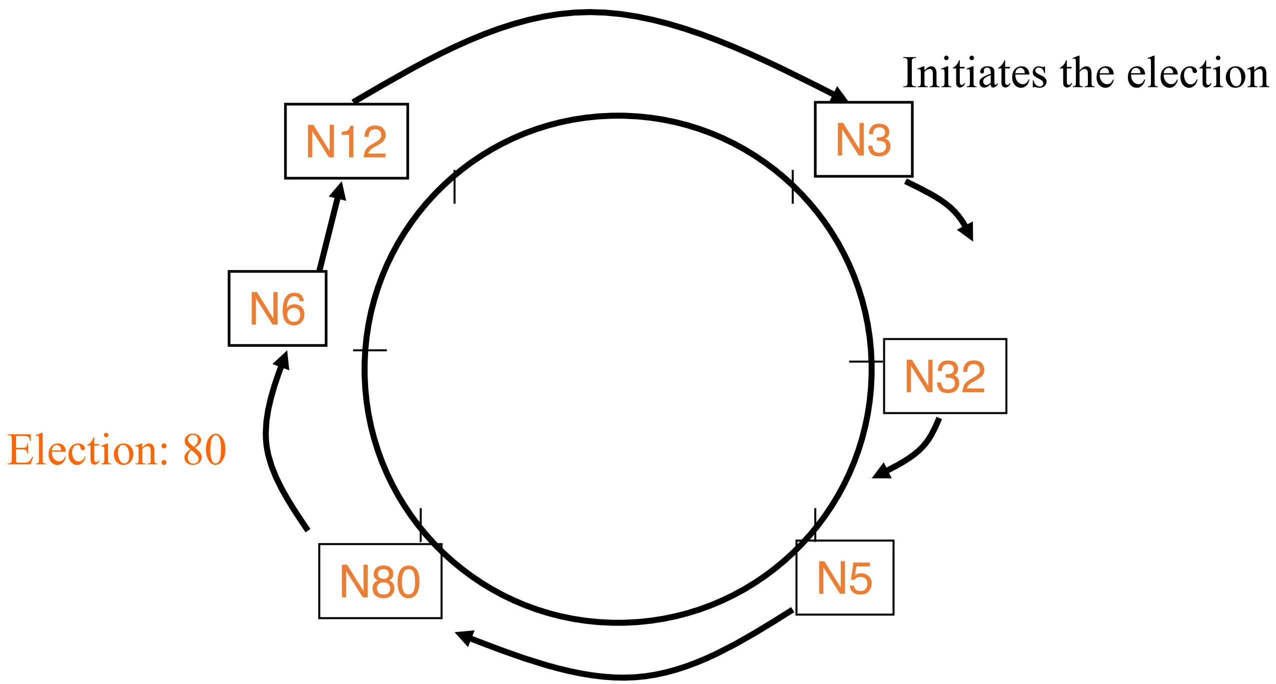
Algorithm does not terminate.

Liveness violated.

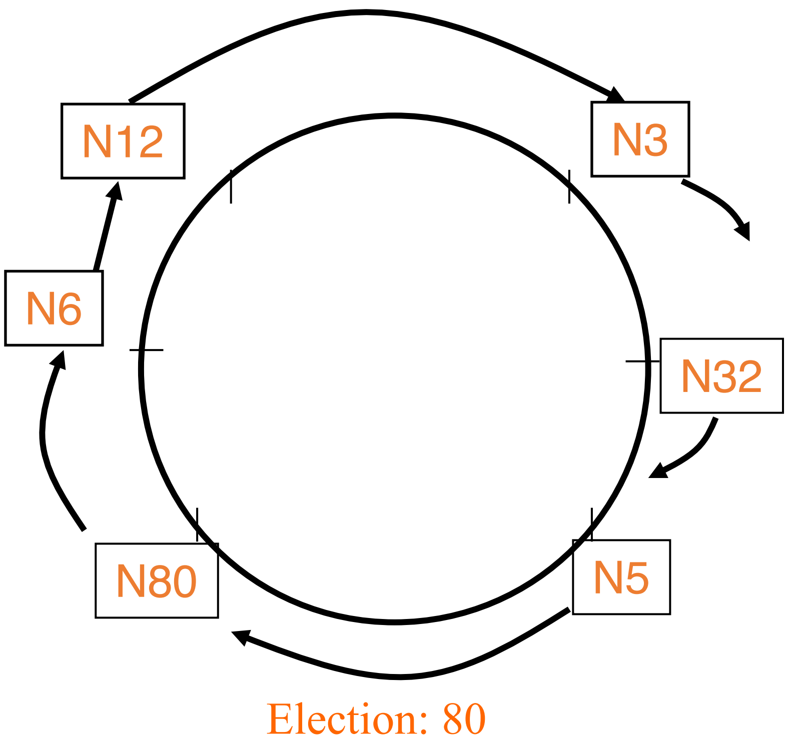
Handling failures

- Use the failure detector.
- A process can detect failure of N80 via its own local failure detector:
  - Repair the ring.
  - Stop forwarding Election:80 message.
  - Start a new run of leader election.
- But failure detectors cannot be both complete and accurate.
  - Incomplete FD => N80's failure might be missed
    - violation of liveness.
  - Inaccurate FD => N80 mistakenly detected as failed

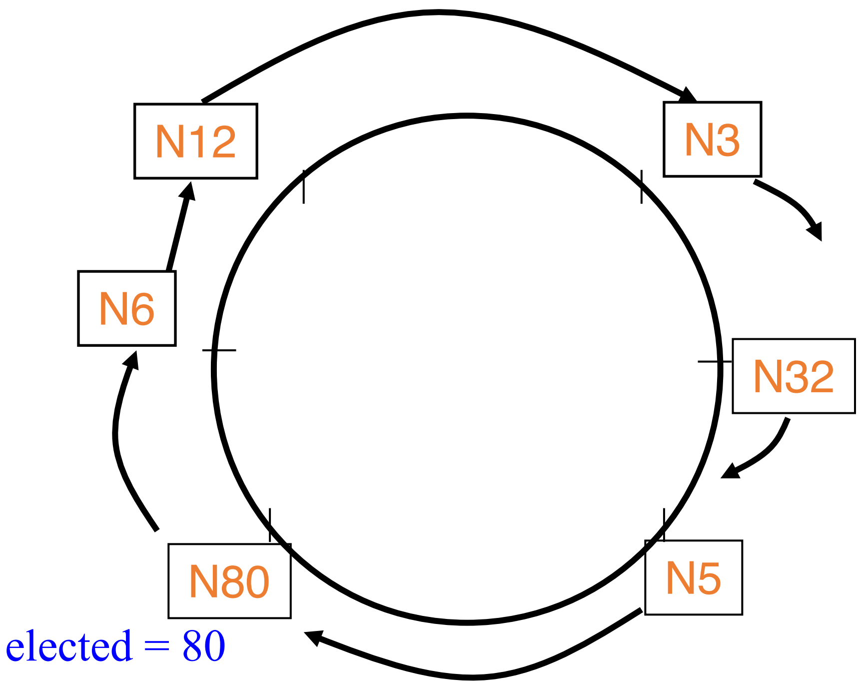
What can happen if an alive process is detected as failed?



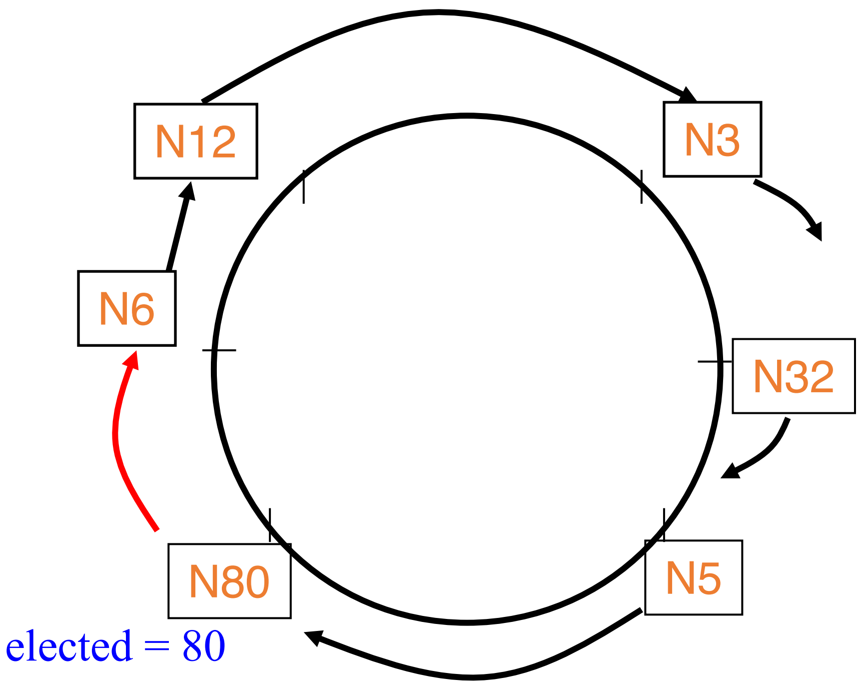
What can happen if an alive process is detected as failed?



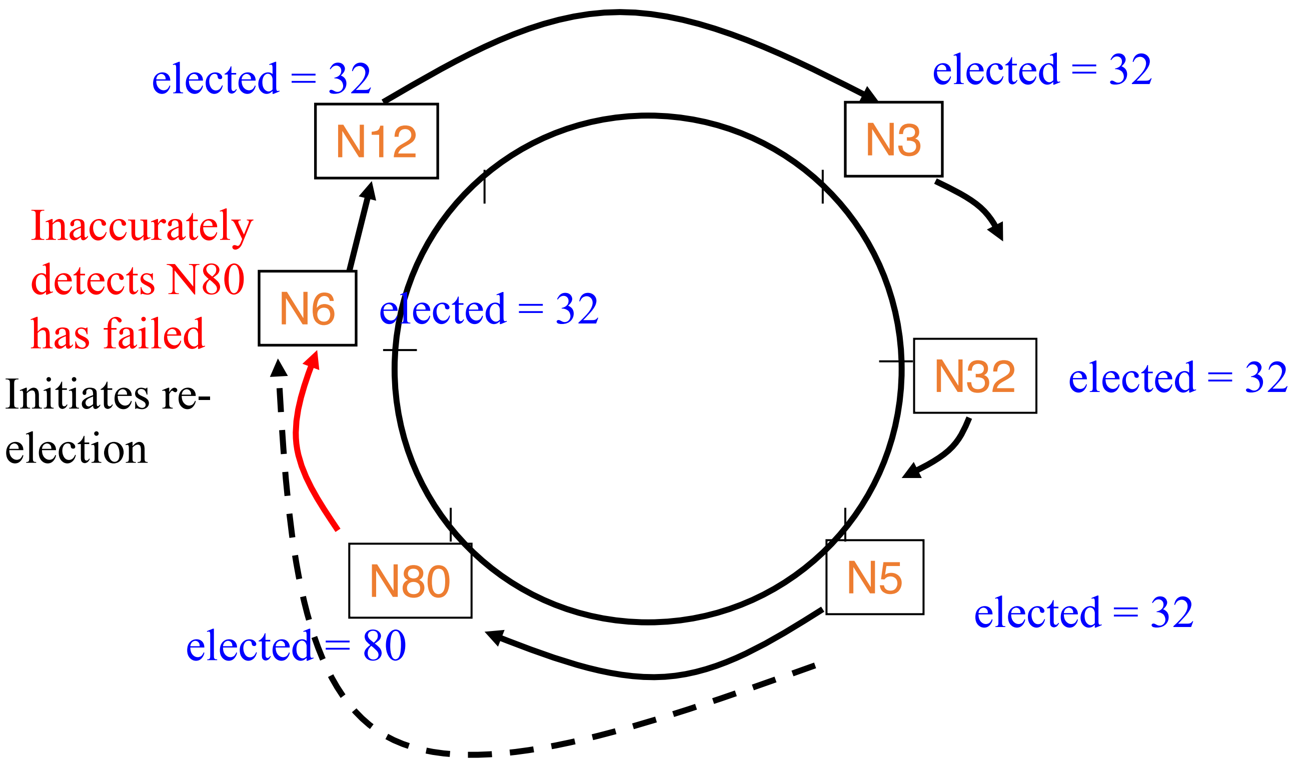
What can happen if an alive process is detected as failed?



What can happen if an alive process is detected as failed?



What can happen if an alive process is detected as failed?



Safety has been violated.

Fixing for failures

- Use the failure detector.
- A process can detect failure of N80 via its own local failure detector:
  - Repair the ring.
  - Stop forwarding Election:80 message.
  - Start a new run of leader election.
- But failure detectors cannot be both complete and accurate.
  - Incomplete FD => N80's failure might be missed
    - violation of liveness.
  - Inaccurate FD => N80 mistakenly detected as failed
    - new ring will be constructed without N80.
    - a process with lower attribute will be selected.
    - violation of safety.

Classical Election Algorithms

- Ring election algorithm
- **Bully algorithm**

Bully algorithm

- Faster turnaround time than ring election.
- Explicitly build in the notion of timeouts into the algorithm.
- Let's assume (for simplicity of exposition) that the attribute based on which leader is elected is the process id.
- Before discussing Bully algorithm, let's first discuss a simpler (related) algorithm…..

Multicast-based algorithm

- Start an election
  - Multicast <election, my ID> to all processes
  - If receive <agree> from all processes, then elected
    - Multicast <coordinator, my ID>
  - If receive <disagree> from any process
    - Give up election

- Receive <election, *ID*> from process *p*
  - If ID > my ID
    - Send <agree> to *p* (unicast)
  - If ID < my ID
    - Send <disagree> to *p*
    - Start election (if not already running)
- What about failures?

Multicast-based algorithm

- Start an election
  - Multicast <election, my ID> to all processes
  - If receive <agree> from all processes or timeout, then elected
    - Multicast <coordinator, my ID>
  - If receive <disagree> from any process
    - Give up election

- Receive <election, *ID*> from process *p*
  - If ID > my ID
    - Send <agree> to *p* (unicast)
  - If ID < my ID
    - Send <disagree> to *p*
    - Start election (if not already running)
- Can we improve on this?

Multicast-based algorithm

- Start an election
  - Multicast <election, my ID> to all processes
  - If ~~receive <agree> from all processes or~~ timeout, then elected
    - Multicast <coordinator, my ID>
  - If receive <disagree> from any process
    - Give up election
- Receive <election, *ID*> from process *p*
  - ~~If ID > my ID~~
    - ~~Send <agree> to *p* (unicast)~~
  - If ID < my ID
    - Send <disagree> to *p*
    - Start election (if not already running)
- Can we improve on this?

Bully Algorithm

- All processes know other process' ids.
- Do not need to multicast election to all processes.
- Only to processes with higher id.

Bully Algorithm

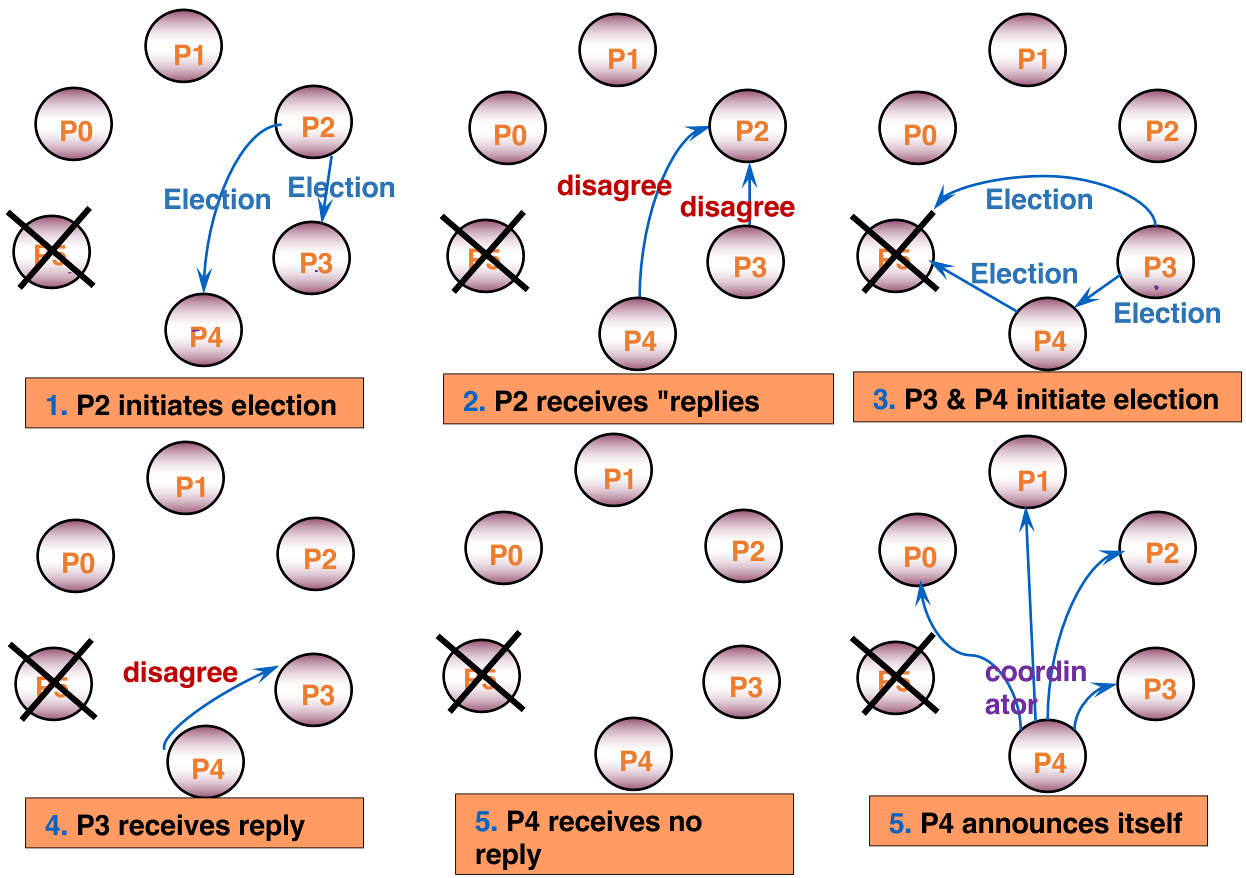
- When a process wants to initiate an election
  - **if** it knows its id is the highest
    - it elects itself as coordinator, then sends a *Coordinator* message to all processes with lower identifiers. Election is completed.
  - **else**
    - it initiates an election by sending an *Election* message
    - (contd.)

Bully Algorithm (2)

- **else** it initiates an election by sending an *Election* message
  - Sends it to only processes that have a *higher id than itself*.
  - **if** receives no answer within timeout, calls itself leader and sends *Coordinator* message to all lower id processes. Election completed.
  - **if** an answer received however, then there is some non-faulty higher process => so, wait for coordinator message. If none received after another timeout, start a new election run.
- A process that receives an *Election* message replies with *disagree* message, and starts its own leader election protocol (unless it has already done so).

Bully Algorithm: Example

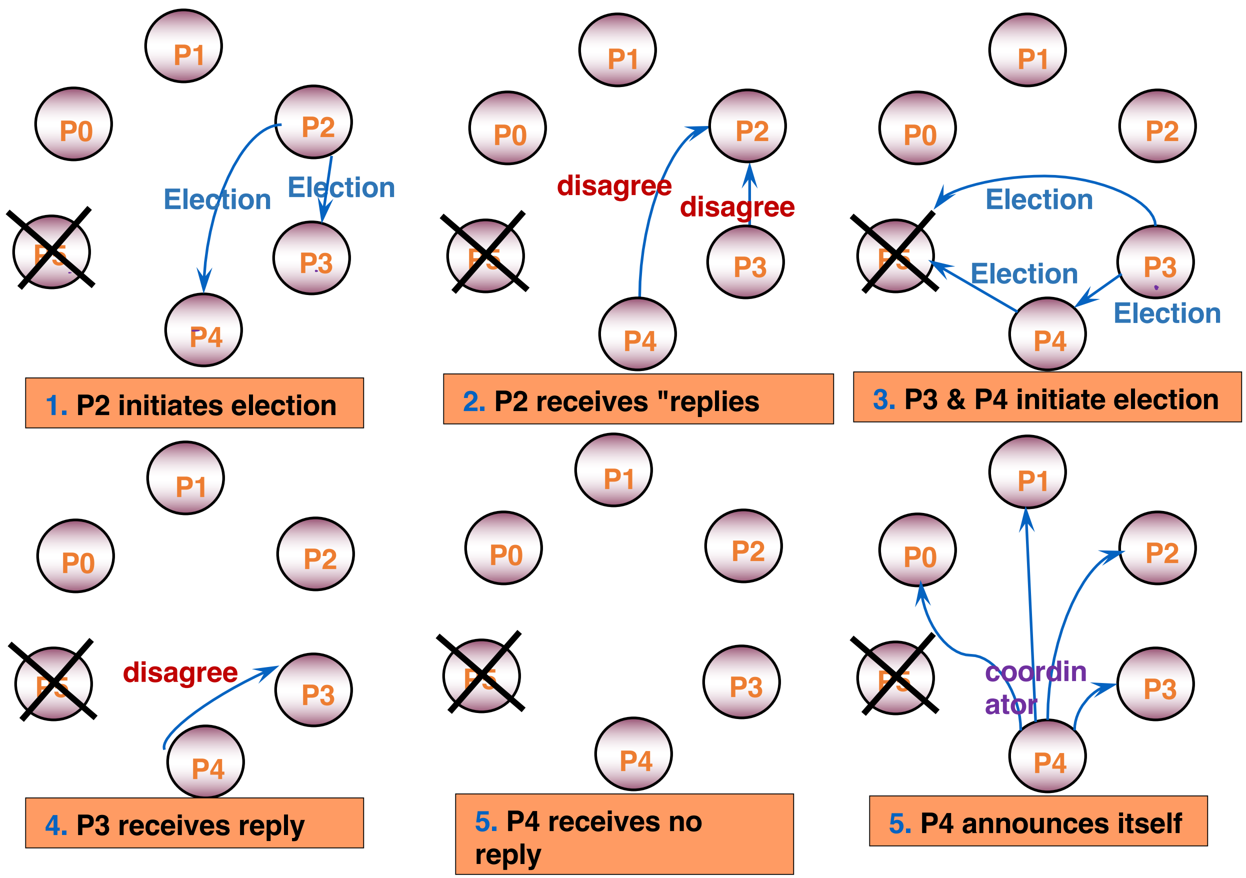
P2 initiates election after detecting P5's failure.



What if P4 fails after step 3?

Bully Algorithm: Example

P2 initiates election after detecting P5's failure.



What if P4 fails after step 4?

Bully Algorithm (2)

- **else** it initiates an election by sending an *Election* message
  - Sends it to only processes that have a *higher id than itself*.
  - **if** receives no answer within **timeout**, calls itself leader and sends *Coordinator* message to all lower id processes. Election completed.
  - **if** an answer received however, then there is some non-faulty higher process => so, wait for coordinator message. If none received after another **timeout**, start a new election run.
- A process that receives an *Election* message replies with *disagree* message, and starts its own leader election protocol (unless it has already done so).

Timeout values

- Assume the one-way message transmission time (T) is known.

- First timeout value (when the process that has initiated election waits for the first response)
  - Must be set as accurately as possible.
    - If it is too small, a lower id process can declare itself to be the coordinator even when a higher id process is alive.
  - What should be the first timeout value be, given the above assumption?
    - $2T + (\text{processing time}) \approx 2T$

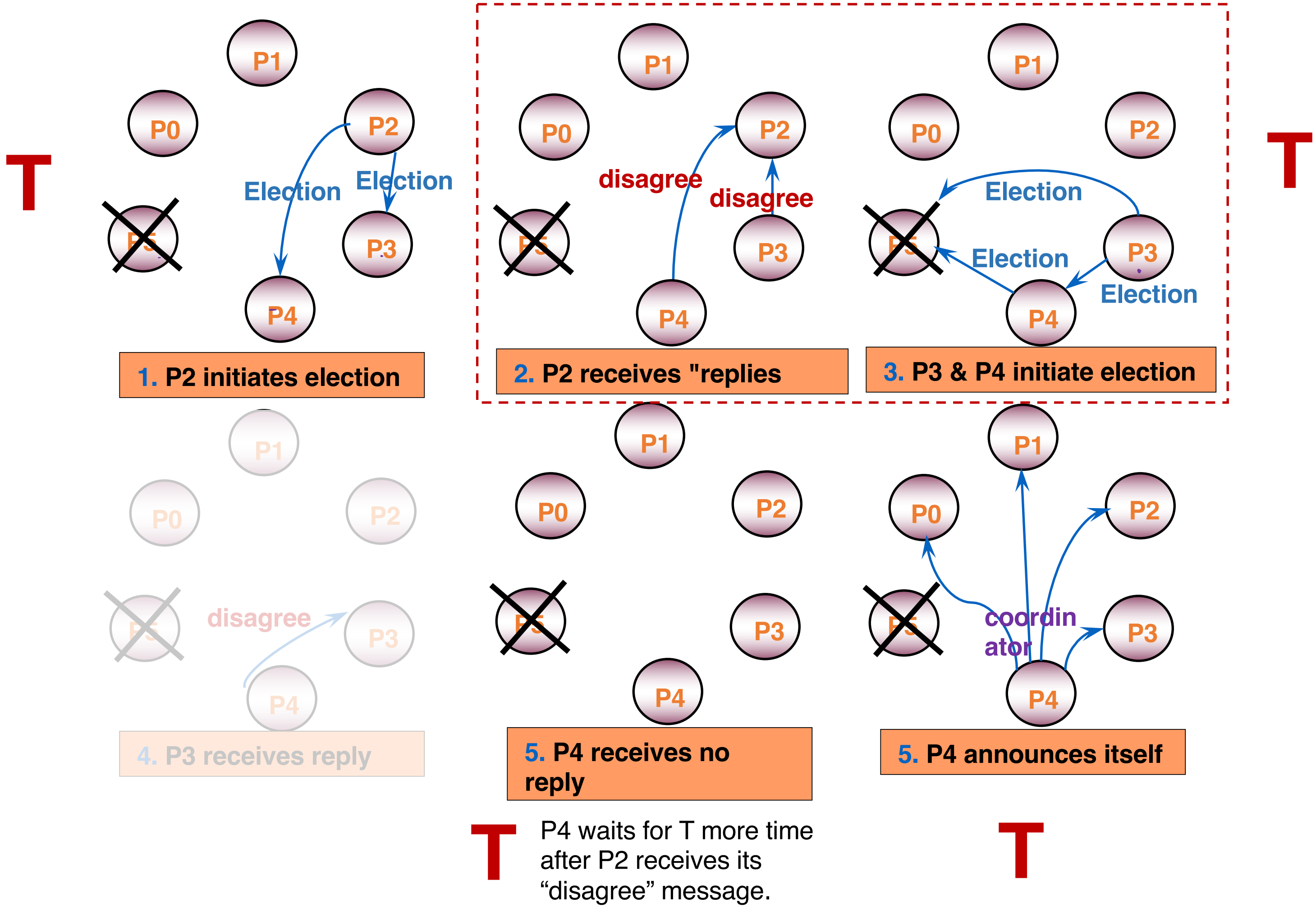
- When the second timeout happens (after 'disagree' message), election is re-started.
  - A very small value will lead to extra "Election" messages.
  - A suitable option is to use the worst-case turnaround time.

Performance Analysis

- Best-case
  - Second-highest id detects leader failure
    - Highest remaining id initiates election.
  - Sends (N-2) Coordinator messages
  - Turnaround time: 1 message transmission time (T)
- Worst-case: For simplicity, assume no failures after a process calls for election.
  - Turnaround time: 4 message transmission times (4T)
    - if any lower id process detects failure and starts election.

Bully Algorithm: Example

P2 initiates election after detecting P5's failure.



Analysis

- Best-case
  - Second-highest id detects leader failure
    - Highest remaining id initiates election.
  - Sends (N-2) Coordinator messages
  - Turnaround time: 1 message transmission time
- Worst-case: For simplicity, assume no failures after a process calls for election.
  - Turnaround time: 4 message transmission times
    - if any lower id process detects failure and starts election.
    - Election + (disagree & Election) + (Timeout – T) + Coordinator
  - When the process with the lowest id in the system detects failure.
    - (N-1) processes altogether begin elections, each sending messages to processes with higher ids.
    - i-th highest id process sends (i-1) election messages
    - Number of Election messages

      $= N-1 + N-2 + \ldots + 1 = (N-1)*N/2 = O(N^2)$

Correctness

- In synchronous system model:
  - Set timeout accurately using known bounds on network delays and processing times.
  - Satisfies safety and liveness.

- In asynchronous system model:
  - Failure detectors cannot be both accurate and complete.
  - Either liveness and safety is violated.

Why is Election so hard?

- Because it is related to the consensus problem!
- If we could solve election, then we could solve consensus!
  - Elect a process, use its id's last bit as the consensus decision.
- But (as we will see in next week's class) consensus is impossible in asynchronous systems, so is election!

Summary

- Leader election is an important problem in distributed system.
  - Crucial for implementing any centralized algorithm.
- Two classical algorithms:
  - Ring election algorithm and Bully algorithm
- Hard to guarantee correctness in an asynchronous system with failures.