CSE 486/586 Distributed Systems
Leader Election

Steve Ko
Computer Sciences and Engineering
University at Buffalo

Recap: Mutual Exclusion

- Centralized
- Ring-based
- Ricart and Agrawala's
- Maekawa's

Why Election?

- Example 1: sequencer for TO multicast
- Example 2: leader for mutual exclusion
- Example 3: group of NTP servers: who is the root server?

What is Election?

- In a group of processes, elect a *leader* to undertake special tasks.
- What happens when a leader fails (crashes)
  - Some process detects this (how?)
  - Then what?
- Focus of this lecture: election algorithms
  - 1. Elect one leader only among the non-faulty processes
  - 2. All non-faulty processes agree on who is the leader
- We'll look at 3 algorithms

Assumptions

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

Problem Specification

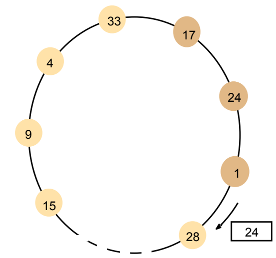
- At the end of the election protocol, the non-faulty process with the best (highest) election attribute value is elected.
  - Attribute examples: CPU speed, load, disk space, ID
  - Must be unique
- Each process has a variable *elected*.
- A run (execution) of the election algorithm must always guarantee at the end:
  - Safety: $\forall$ non-faulty p: (p's *elected* = (q: a particular non-faulty process with the best attribute value) or $\perp$)
  - Liveness: $\forall$ election: (election terminates) & $\forall$ p: non-faulty process, p's *elected* is eventually not $\perp$

Algorithm 1: Ring Election [Chang & Roberts'79]

- N Processes are organized in a logical ring
  - p_i has a communication channel to $p_{i+1 \text{ mod } N}$.
  - All messages are sent clockwise around the ring.
- To start election
  - Send *election* message with my ID
- When receiving message (*election*, id)
  - If id > my ID: forward message
    - » Set state to *participating*
  - If id < my ID: send (*election*, my ID)
    - » Skip if already *participating*
    - » Set state to *participating*
  - If id = my ID: I am elected (why?) send *elected* message
    - » *elected* message forwarded until it reaches leader

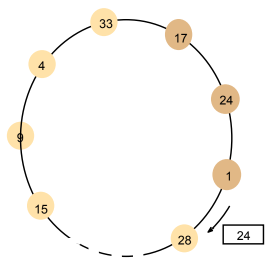
Ring-Based Election: Example

- The worst-case scenario occurs when?
  - the counter-clockwise neighbor (@ the initiator) has the highest attr.
- In the example:
  - The election was started by process 17.
  - The highest process identifier encountered so far is 24
  - (final leader will be 33)



Ring-Based Election: Analysis

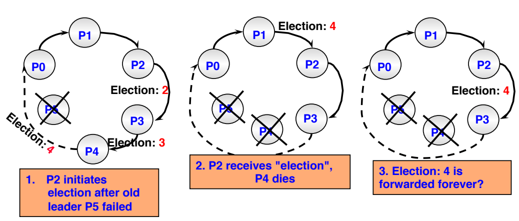
- In a ring of N processes, in the worst case:
  - N-1 *election* messages to reach the new coordinator
  - Another N *election* messages before coordinator decides it's elected
  - Another N *elected* messages to announce winner
- Total Message Complexity = 3N-1
- Turnaround time = 3N-1



Correctness?

- Safety: highest process elected
- Liveness: complete after 3N-1 messages
  - What if there are failures during the election run?

Example: Ring Election



May not terminate when process failure occurs during the election!

Consider above example where attr==highest id

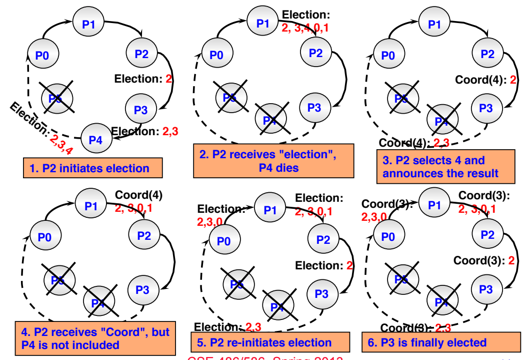
CSE 486/586 Administrivia

- PA2 due this Friday
- Midterm: 3/12 (Wednesday) in class
  - Everything up to today

Algorithm 2: Modified Ring Election

- *election* message tracks *all* IDs of nodes that forwarded it, not just the highest
  - Each node appends its ID to the list
- Once message goes all the way around a circle, new *coordinator* message is sent out
  - Coordinator chosen by highest ID in *election* message
  - Each node appends its own ID to *coordinator* message
- When *coordinator* message returns to initiator
  - Election a success if coordinator among ID list
  - Otherwise, start election anew

Example: Ring Election



Modified Ring Election

- How many messages?
  - 2N
- Is this better than original ring protocol?
  - Messages are larger
- Reconfiguration of ring upon failures
  - Can be done if all processes "know" about all other processes in the system
- What if initiator fails?
  - Successor notices a message that went all the way around (how?)
  - Starts new election
- What if two people initiate at once
  - Discard initiators with lower IDs

What about that Impossibility?

- Can we have a totally correct election algorithm in a fully asynchronous system (no bounds)
  - No! Election can solve consensus
- Where might you run into problems with the modified ring algorithm?
  - Detect leader failures
  - Ring reorganization

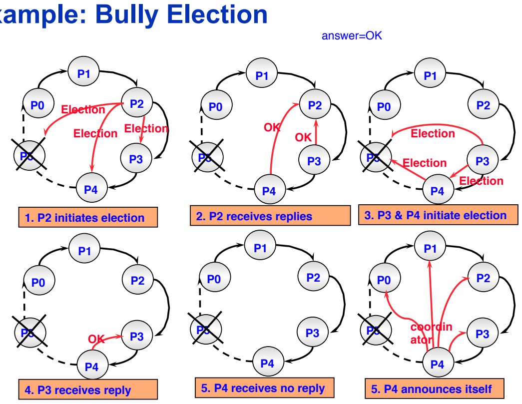
Algorithm 3: Bully Algorithm

- Assumptions:
  - Synchronous system
  - attr=id
  - Each process knows all the other processes in the system (and thus their id's)

Algorithm 3: Bully Algorithm

- 3 message types
  - *election* – starts an election
  - *answer* – acknowledges a message
  - *coordinator* – declares a winner
- Start an election
  - Send *election* messages *only* to processes with higher IDs than self
  - If no one replies after timeout: declare self winner
  - If someone replies, wait for *coordinator* message
    - » Restart election after timeout
- When receiving *election* message
  - Send *answer*
  - Start an election yourself
    - » If not already running

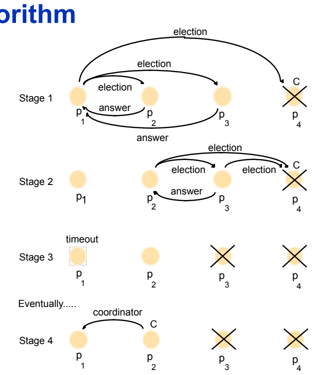
Example: Bully Election



The Bully Algorithm

The coordinator p_4 fails and p_1 detects this

p_3 fails



Analysis of The Bully Algorithm

- Best case scenario?
- The process with the second highest id notices the failure of the coordinator and elects itself.
  - N-2 *coordinator* messages are sent.
  - Turnaround time is one message transmission time.

Analysis of The Bully Algorithm

- Worst case scenario?
- When the process with the lowest id in the system detects the failure.
  - N-1 processes altogether begin elections, each sending messages to processes with higher ids.
  - The message overhead is $O(N^2)$.

Turnaround time

- All messages arrive within T units of time (synchronous)
- Turnaround time:
  - *election* message from lowest process (T)
  - Timeout at 2nd highest process (X)
  - *coordinator* message from 2nd highest process (T)
- How long should the timeout be?
  - $X = 2T + T_{process}$
  - Total turnaround time: $4T + 3T_{process}$

Summary

- Coordination in distributed systems sometimes requires a leader process
- Leader process might fail
- Need to (re-) elect leader process
- Three Algorithms
  - Ring algorithm
  - Modified Ring algorithm
  - Bully Algorithm

Acknowledgements

- These slides contain material developed and copyrighted by Indranil Gupta (UIUC).