Mutual Exclusion in a Distributed Environment (Review)

- Mutual exclusion
  - Centralized algorithms
    - Central physical clock
    - Central coordinator
  - Distributed algorithms
    - Time-based event ordering
      - Lamport's algorithm (logical clocks)
      - Ricart & Agrawala's algorithm (" ")
      - Suzuki & Kasimi's algorithm (broadcast)
    - Token passing
      - Le Lann's token-ring algorithm (logical ring)
      - Raymond's tree algorithm (logical tree)
    - Sharing K identical resources
      - Raymond's extension to Ricart & Agrawala's time-based algorithm
  - Atomic transactions (later in course)
- Related — self-stabilizing algorithms, election, agreement, deadlock

Election Algorithms

- In a distributed system, many algorithms require a permanent or temporary leader:
  - Distributed mutual exclusion:
    - Central coordinator algorithm requires a coordinator
    - Token-ring algorithm, Suzuki-Kasami's broadcast algorithm, and Raymond's tree algorithm require an initial token holder
  - Distributed deadlock detection — maintainer of a global wait-for graph
- If leader fails, must *elect* a new leader
  - Election algorithms assume there is a unique priority number for each thread
  - Goal: elect the highest-priority thread as the leader, tell all active threads
  - Second goal: allow a recovered leader to re-establish control (or at least, to identify the current leader)

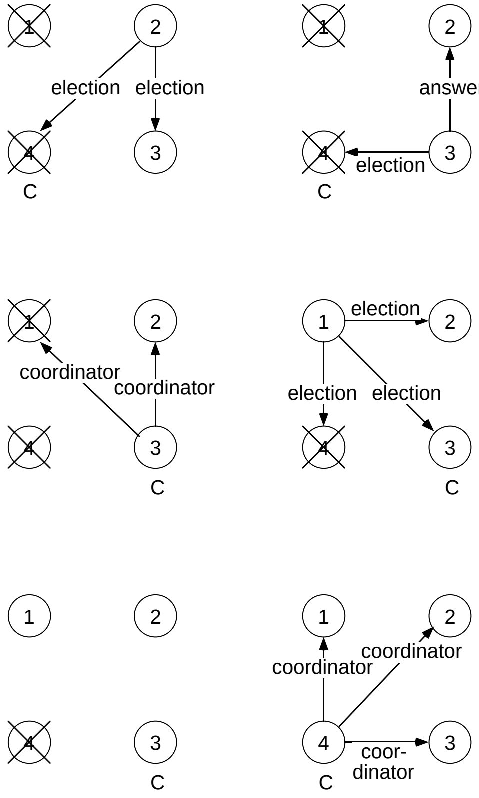
Garcia-Molina's Bully Algorithm (1993)

- 3 types of messages:
  - *Election* —announce an election
  - *Answer* — acknowledge election msg.
  - *Coordinator* — announce new coordinator
- The election:
  - A thread begins an election when it notices the coordinator has failed
    - To do so, it sends *election* messages to all threads with a higher priority
  - It then awaits an *answer* message (from a live thread with a higher priority)
    - If none arrives within a certain time, it declares itself the coordinator, and sends a *coordinator* message to all threads with a lower priority
    - If an *answer* message does arrive, it waits a certain time for a *coordinator* message to arrive from the new coordinator
      - If none arrives, it begins another election

Garcia-Molina's Bully Algorithm (cont.)

- Result of the election:
  - If a thread receives a *coordinator* message, it accepts the new coordinator
- Participating in an election:
  - If a thread receives an *election* message:
    - It sends back an answer message
    - It begins another election (with its higher-ups) unless it has already begun one
- Failed threads:
  - When one restarts, it begins an election
    - Unless it knows it has the highest priority, in which case it just sends out *coordinator* messages to re-establish control
- Evaluation:
  - N–2 messages in best case
  - $O(N^2)$ messages in worst case

Garcia-Molina’s Bully Algorithm (cont.)



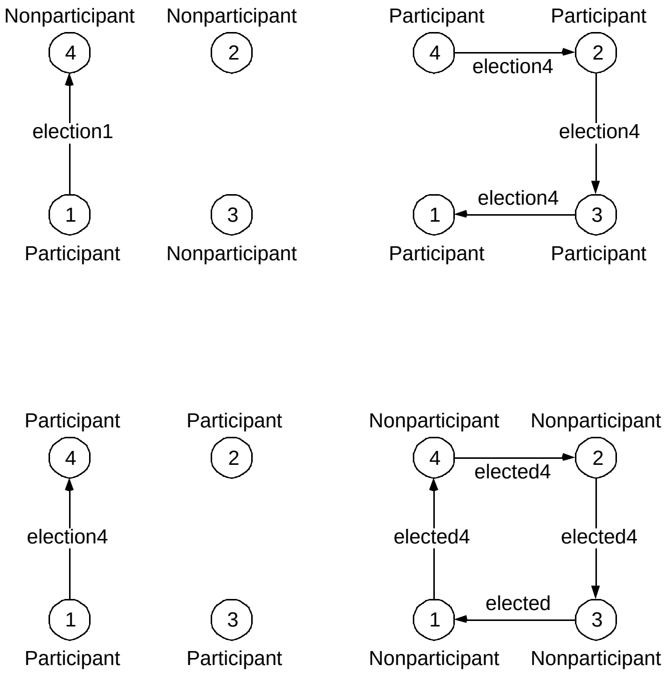
Chang and Roberts’ Ring Algorithm (1979)

- Threads are arranged in a logical ring
  - Every thread is initially a *non-participant*
- The election:
  - A thread begins an election by
    - Marking itself as a *participant*
    - Sending an *election* message (containing its identifier) to its neighbor
  - When a thread receives an *election* message, it compares the identifier that arrived in the message to its own:
    - If the arrived identifier is greater, then it:
      - If it is not a *participant*, it:
        - Marks itself as a *participant*
      - Forwards the message to its neighbor
    - If the arrived identifier is smaller:
      - If it is not a *participant*, it:
        - Marks itself as a *participant*
        - Substitutes its own identifier in the *election* message and sends it on
      - If it is already a *participant*, it does nothing

Chang and Roberts’ Ring Algorithm (cont.)

- The election:
  - When a thread receives an *election* message, it compares…:
    - If the arrived identifier is that of the receiving thread, then its identifier is the largest, so it becomes the coordinator
      - It marks itself as a *non-participant* again,
      - It sends an *elected* message to its neighbor, announcing the results of the election and its identity
  - When a thread receives an *elected* message, it
    - Marks itself as a *non-participant*, and
    - Forwards the message to its neighbor
- Evaluation:
  - 3N–1 messages in worst case
    - N–1 *election* messages to reach immediate neighbor in wrong direction, N *election* messages to elect it, then N *elected* messages to announce result

Chang and Roberts’ Ring Algorithm (cont.)


Nonparticipant
Nonparticipant
4
2
election1
1
3
Participant
Nonparticipant
Participant
Participant
4
election4
2
election4
election4
1
3
Participant
Participant
Participant
Participant
4
2
election4
1
3
Participant
Participant
Nonparticipant
Nonparticipant
4
elected4
2
elected4
elected4
elected
1
3
Nonparticipant
Nonparticipant