Week 5: Part 2
Leader Election
Leader Election

• Purpose
  – Need to pick one process to act as coordinator

• Assumptions
  – Processes have no distinguishing characteristics
  – Each process has a unique ID to identify itself
  – Reliable message delivery
Bully algorithm

• Select process with largest ID as the leader

• When process P detects a dead leader:
  – Send *election* message to all processes with higher IDs
    • If nobody responds, *P wins* and takes over
    • If any process responds, P’s job is done
  – Optional: Let all nodes with lower IDs know an election is taking place

• If process receives an election message
  – Send *OK* message back
  – Hold election (unless it is already holding one)
Bully algorithm

• A process announces victory:
  – Sends all processes a message telling them that it is the new leader

• If a dead process recovers
  – It holds an election to find the leader
Bully algorithm

Rule: highest # process is the leader

Suppose $P_5$ dies

$P_2$ detects $P_5$ is not responding
Bully algorithm

P₂ starts an election
Contacts all higher-numbered systems
Everyone who receives an *ELECTION* message responds
… and holds their own election, contacting higher # processes

Example: P₃ receives the message from P₂
  Responds to P₂
  Sends *ELECTION* messages to P₄ and P₅
P₄ responds to P₃ and P₂'s messages
… and holds an election
Bully algorithm

Nobody responds to $P_4$

After a timeout, $P_4$ declares itself the leader
Ring election algorithm

Ring arrangement of processes

• If any process detects failure of leader
  – Construct election message with process ID and send to next process
  – If successor is down, skip over
  – Repeat until a running process is located

• Upon receiving an election message
  – Process forwards the message, adding its process ID to the body
Eventually message returns to originator

- Process sees its ID on list
- Multicasts a leader message announcing the new leader
  - E.g., highest numbered process
Assume $P_1$ discovers that the leader, $P_5$, is dead

$P_1$ starts an election

Election: $\{P_1\}$
Ring algorithm

Election: \{P_1, P_2\}
Ring algorithm

Election: \{P_1, P_2, P_3\}
Election: \{P_1, P_2, P_3, P_4\}

\textit{Fails: P_5 is dead}
Election: \{P_1, P_2, P_3, P_4\}

*Skip to* $P_0$
Ring algorithm

Election: \{P_1, P_2, P_3, P_4, P_0\}
P₂ receives the election message that it initiated
P₂ now picks a leader (e.g., highest ID)

Election: \{P₁, P₂, P₃, P₄, P₀\}

Because P₁ sees its ID at the head of the list, it knows that this is the election that it started

We might have multiple concurrent elections. Everyone needs to pick the same leader. Here, we agree to pick the **highest ID** in the list.

This is me! Winner!
P₁ announces that P₄ is the new leader to the group

Many other election algorithms that target other topologies: mesh, torus, hypercube, trees, …
Optimize the ring
- Message always contains *one* process ID
- Avoid multiple circulating elections

- If a process sends a message, it marks its state as a *participant*
  - This allows it to cut off extra elections
- Assume highest # PID is the winner
Upon receiving an election message:

If \( \text{PID(message)} > \text{PID(process)} \) – *higher ID will always win over a lower one*
forward the message

If \( \text{PID(message)} < \text{PID(process)} \) AND process is not a *participant* – *we are a higher ID number; use it*
replace PID in message with PID(process) and forward the new message
set the process state to *participant*

If \( \text{PID(message)} < \text{PID(process)} \) AND process is *participant* *discard* the message
– *we’re already circulating our ID and it’s a higher number than this one*

If \( \text{PID(message)} == \text{PID(process)} \)
the process is now the leader
– *message fully circulated to the one who started: announce winner*
Network Partitions

• Network **partitions** (segmentation)
  – Multiple nodes may decide they’re the leader
  – Leads to multiple groups, each with a leader & diverging data among them → **split brain**

• Dealing with partitions
  – Insist on a majority → if no majority, the system will not function
    • **Quorum** = minimum # of participants required for a system to function
  – Rely on alternate communication mechanism to validate failure
    • Redundant network, shared disk (but that can also fail)
The End