A New Leader Election Implementation

Hans Svensson

Thomas Arts

Leader Election

- Within a set of participating processes
  - Algorithm ensures exactly one leader
  - All (active) participants know this leader
- Erlang behavior gen_leader
- Original implementation is broken *[ACS04]*
- We re-implemented gen_leader

[ACS04] T. Arts, K. Claessen, and H. Svensson. *Semi-formal development of a fault-tolerant leader election protocol in Erlang.* In: Lecture Notes in Computer Science, vol. 3395, p. 140-154, Springer, Feb 2005.

What was broken

- Two leaders elected at the same time
  - Incorrect modification of Singh's algorithm
- Dead-lock situation without leader
  - Overlooked critical message sequence

Note: The original algorithm (Singh) is not broken, just the implementation

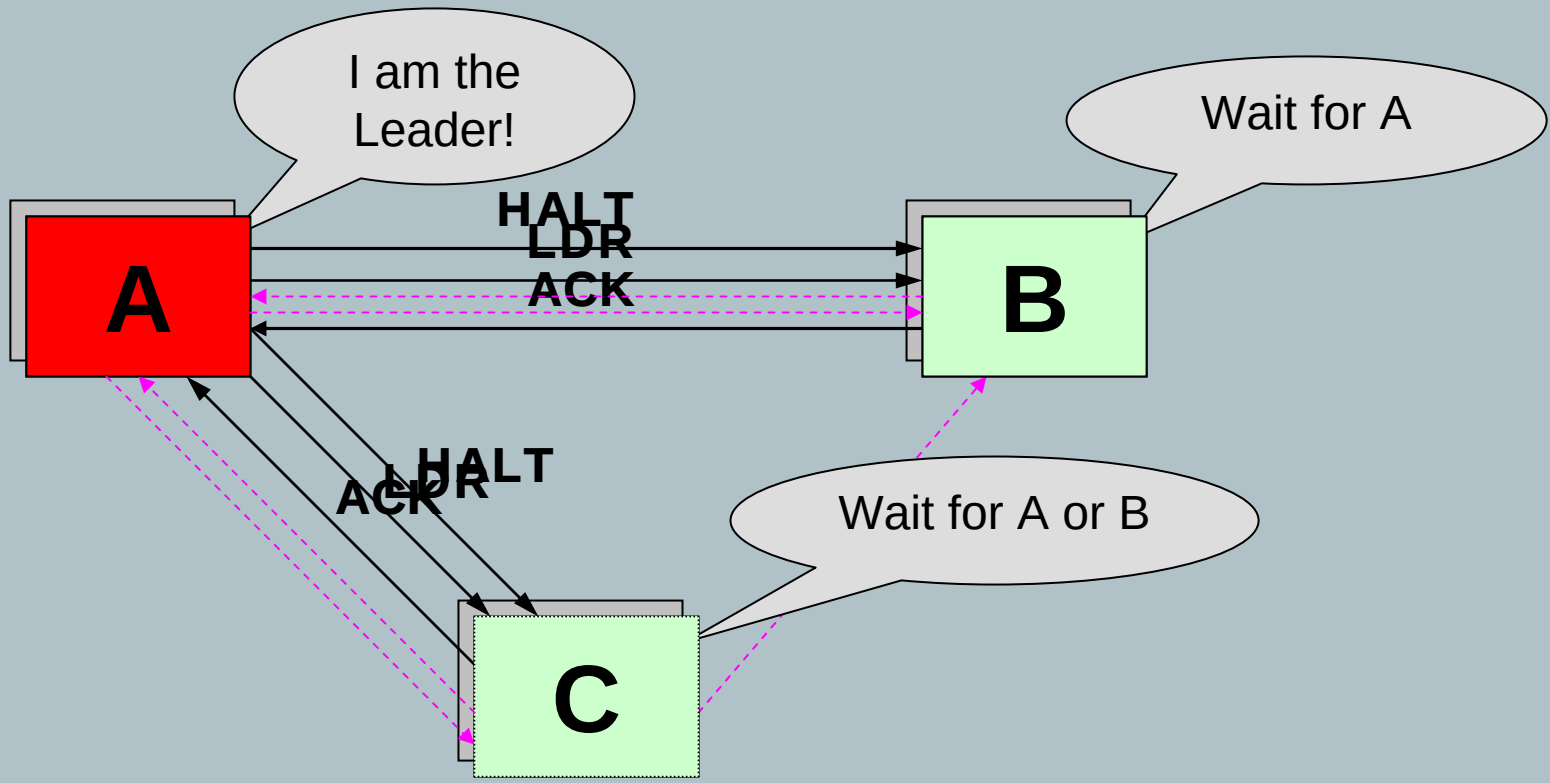
Why not fix it?

- Already substantially modified
  - Not a very good algorithm choice
  - Semantic mismatch
  - Risk of introducing new errors
- Choose a more suitable algorithm
  - Non-trivial task
  - Can we expect a good match?

'Leader Election in Distributed Systems with Crash failure' - S. Stoller

When a process is started, it first checks whether a process with **higher priority** is **active**. If such a process exist, the process simply **waits** for one of those processes to become the leader. If, on the other hand, the present process is the process with **highest priority**, the process itself **tries to become** the **leader**. Becoming the leader is done by making sure that all processes with lower priority either are **aware of its existence** or are **inactive**. When all processes with lower priority are informed, the process announces itself as the leader. **Periodically**, the elected leader **polls** the **inactive processes**, if one of the inactive processes is activated, the **election** process is **restarted**. Processes supervise each other with **failure detectors**.

'Leader Election in Distributed Systems with Crash failure' - S. Stoller

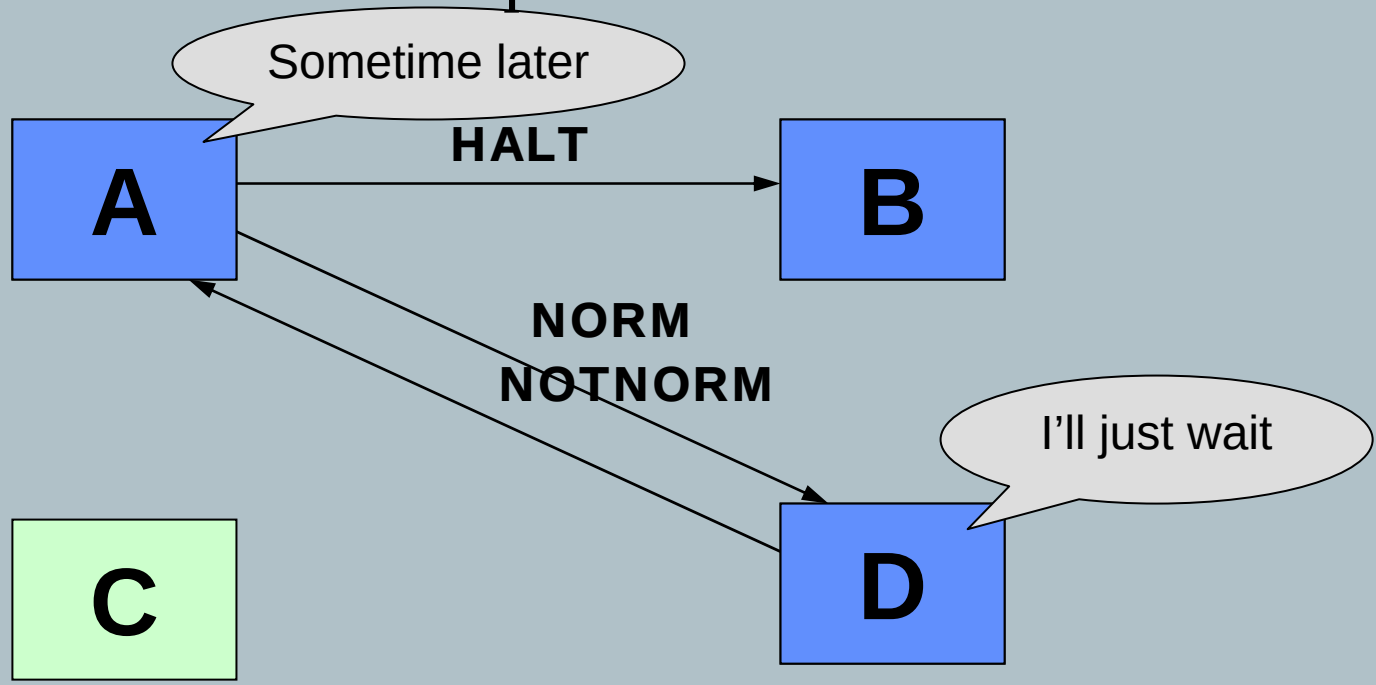


A > B > C

Failure detector

Changing the behavior

- Re-election every time a process is activated
  - Inefficient
  - Does not match our requirements



Failure detectors omitted

Adapting the Algorithm

Assume that we have an elected leader

- A process with lower priority is activated
  - The leader informs the new process
- A process with higher priority is activated
  - Tries to start a new election
  - The others should not accept a ‘HALT’
  - Anyone can inform the new process
  - The new process confirms the leadership

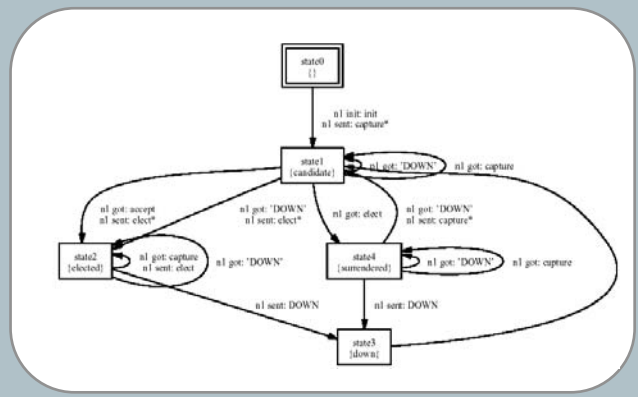
Failure detection is VERY important here

Implementation

- Translates well into Erlang
- Failure detection is done by Monitors
- Same interface as original gen_leader

Testing the implementation

- Tracing and Abstraction
  - Randomly activates/deactivates processes
  - Randomly delays messages
  - Abstract traces can be model checked
- Erlang QuickCheck
  - Random testing technique
  - Influence the scheduler



Features

- Fault tolerant leader election
- No unnecessary elections
- Implemented as Erlang behavior
- Correct?

http://www.cs.chalmers.se/~hanssv/leader_election