# Autonomic Computer Systems CS321 Distributed Self-* Algorithms I: Leader Election, Self-Stabilizing Algos

September 22, 2011

Prof. Dr. Christian Tschudin

Departement Mathematik und Informatik, Universität Basel

## Overview 2011-09-22

Selected "Self-" Algorithms (I)

- Introduction to coordination algorithms
- Leader election
- Notes on message passing models
- Self-stabilization (example: mutual exclusion)
- Logical clocks
- Distributed snapshot

## Introduction: Distributed Systems

Characterization:

- nodes interconnected through a network
- loosely coupled (message exchange)
- independent activity (concurrency)
- unreliable (lossy channels, node failures)

It's hard to write reliable distributed programs.
– Many impossibility results, too!
– Even "autonomics" can not help, then.

## Need for a Coordinator

Hypothetical task: keep a system wide counter.
How to cope with node crashes, network partitioning?

- Easiest is to have a single "coordinator" (server), all other nodes in the net are clients.
- Not relevant which node is coordinator.
- In case of server crash: pick another node.
- In case of network partition: add a 2nd server.

"Leader election" needed. In general: a consensus finding problem

## Overview

Selected “Self-” Algorithms (I)

- Introduction to coordination algorithms
- **Leader election**
- Notes on message passing models
- Self-stabilization (example: mutual exclusion)
- Logical clocks
- Distributed snapshot

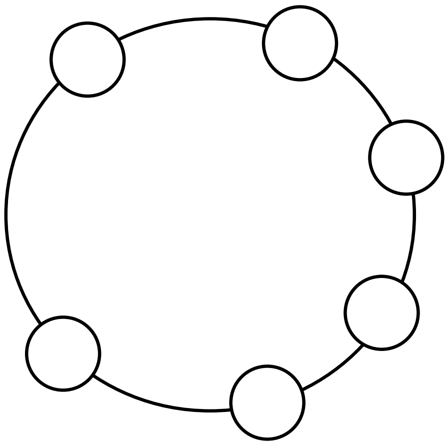
## Leader Election (LE)

Problem description:

- Nodes have status, with one “elected” (winning) and “not-elected” (loosing) state, as well as other states.
- After being elected, a node stays elected (same for not-elected)
- Required properties of a LE algorithm:
  – eventually, every node has either won or lost
  – **exactly one node enters a winning state.**

## Topology – Ring

In proofs for distributed systems, a ring topology is often used instead of arbitrary graphs.



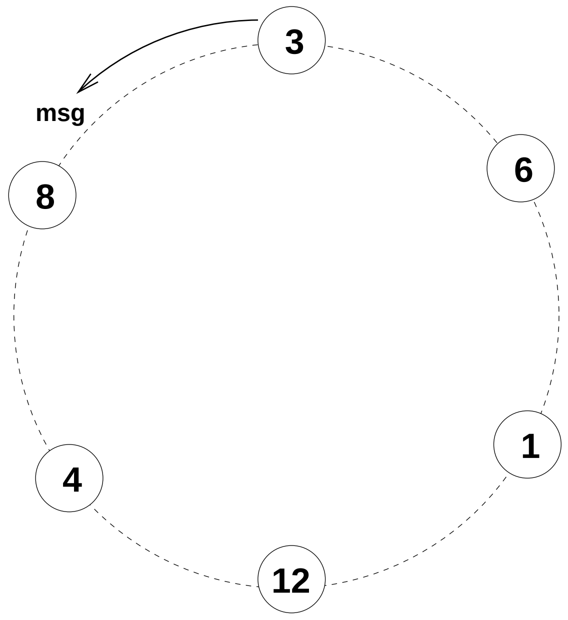
- Ring looks like a toy case, but is complex enough for some impossibility results.
- More variants of ring topologies:
  – **anonymous ring** vs
  – nodes having a unique identifier
  – uniform / non-uniform algos ...
  etc

## Leader Election: Theory Results

- There is **no** leader election algorithm for anonymous rings (no IDs), even if:
  – algorithm knows the ring size
  – we use the simpler synchronous model (see later)
- Proof sketch:
  – consider config with all proc in same start state
  – all will send same message to neighbor
  – all will receive same message, do same transition
  – all will enter same "win" or "loose" state: violation.

Theorem: There exist LE algorithms when nodes have unique IDs.

## Leader Election Algorithm in a Ring with IDs



- All nodes have identical code, but unique IDs
- How to pick a winner?
- How to make sure that everybody agrees on the winner?

## Leader Election Algorithm (Le Lann, 1977)

```
- send value of own ID to the left
- when receiving an id j (from the right):
  o  if j > ID then forward to the left (we loose)
  o  if j = ID then elect self (we win)
  o  if j < ID then do nothing
```

- Intuitive: IDs traveling around, filters max value
- Works in asynchronous and synchronous model
- Above is not optimal: $O(n^2)$ messages (and n time steps)
- Not possible to do better than O(n log n) in asynch model

## Discussion on Coordination

- Leader Election (as shown above) is simplified:
  – no message loss
  – all processors start at the same time
  – no processor fails
  – no processor cheats
  – special ring topology

“Extrema finding”:
– also key element in Garcia-Molinas “Bully Algorithm” (1982)
– but now for arbitrary topologies?

## Leader Election: Constructing a Spanning Tree

“Prepare the network” before the election (logical ring, spanning tree)

- In a static graph (theorem):
  if all nodes have unique IDs, a single *minimum spanning tree* can be constructed. (All nodes agree on this tree, even if the construction is started at arbitrary places)
- Once a MST is available, do “scatter/gather”:
  – send out LE request, scatter-wave goes to the leaf nodes,
  – elect (somehow) leader in each subtree,
  – elect next-level-leader as gather-wave travels towards the root.

What about wireless?

## Leader Election: MANET and Flooding

Wireless networks permit different spanning tree construction using flooding:

- All nodes can initiate a “Campaign-For-Leader” (CFL message)
- Several CFL waves can coexist,
  – waves are tagged with a timestamp,
  – the timestamp with the lowest timestamp eventually wins
  (this is again “extrema-finding”)
- Although not minimal, this tree can be used for leader election (or simply: the root is the leader).

## Leader Election: Network Partitions etc

What if topology changes during the LE computation? Many other problems. Example of a MANET research paper (2003)

```
We consider the problem of secure leader election and propose two
cheat-proof election algorithms : Secure Extrema Finding Algorithm
(SEFA) and Secure Preference-based Leader Election Algorithm (SPLEA).

Both algorithms assume a synchronous distributed system in
which the various rounds of election proceed in a lock-step fashion.
SEFA assumes that all elector-nodes share a single common evaluation
function that returns the same value at any elector-node when applied
to a given candidate-node. When elector-nodes can have different
```

```
preferences for a candidate-node, the scenario becomes more
complicated. Our Secure Preference-based Leader Election Algorithm
(SPLEA) deals with this case. Here, individual utility functions at
each elector-node determine an elector-node's preference for a given
candidate-node.

We relax the assumption of a synchronous distributed system in our
Asynchronous Extrema Finding Algorithm (AEFA) and also allow the
topology to change during the election process. In AEFA, nodes can
start the process of election at different times, but eventually
after topological changes stop long enough for the algorithm to
terminate, all nodes agree on a unique leader. Our algorithm has been
proven to be "weakly" self-stabilizing.
```

Vasudevan, DeCleene, Immerman, Kurose and Towsley:
*"Leader Election Algorithms for Wireless Ad Hoc Networks" (2003)*

## LE-related Concepts Introduced Above

- cheat-proof algorithm
  (cheating as a "failure mode")
- synchronous/asynchronous distributed system
- weakly self-stabilizing algorithm

Some of these concepts will be explored in this lecture series.

## Overview

Selected “Self-” Algorithms (I)

- Introduction to coordination algorithms
- Leader election
- **Notes on message passing models**
- Self-stabilization (example: mutual exclusion)
- Logical clocks
- Distributed snapshot

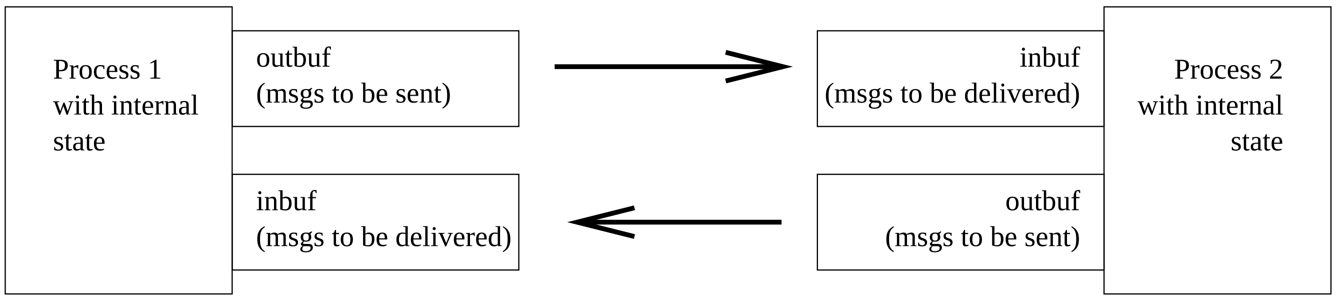
## Some Formalization of Distributed Systems

Networks course: we had implicit assumption of **asynchronous message exchange**.

- Reflects what we have in reality.
- Other models, for better theoretical treatment:
  – **synchronous** message passing
  – asynchronous **shared memory**
- Plus: Topology also plays an important role.
- Plus: failures

Many algo versions: “under this model and this topology, we have ...”

## Message Exchange and Processor State



– Instead of “nodes”, we talk about “processes”
– each process has internal state
– each process has message queues (in and out)
– “channel” is empty (all messages are in queues)

## Configuration

How to describe a system?

- Configuration
  = vector of processor states
  which includes channel content (out and in)
- It’s a kind of “snapshot”, god’s system view
- In reality: global state is not known,
  and in most circumstances **cannot be known**!

## Events: Communication, Computation

In our system modeling, we have two event types:

- **Communication** event:
  move a message from out to in
- **Computation** event:
  execute the process' finite state machine (transducer)
  a) read input and state,
  b) determine next state and output
  c) do the output and stat change)

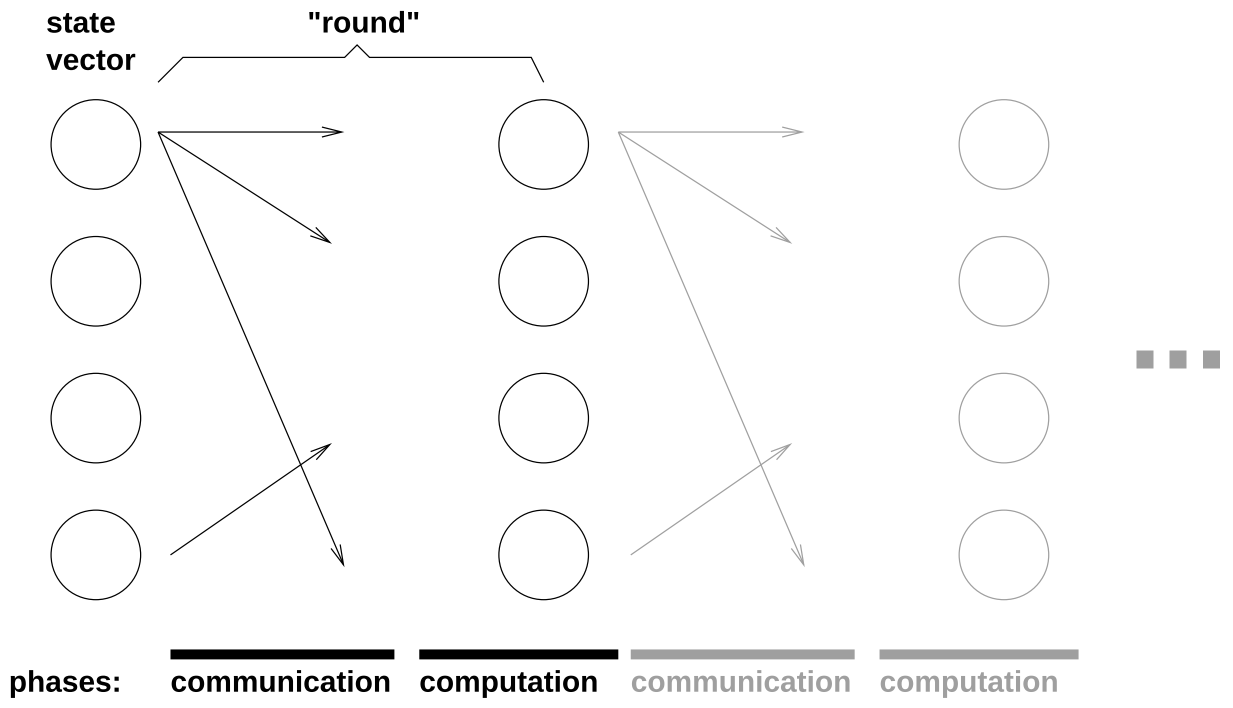
## Two Execution Models (without loss or failure)

Execution = infinite sequence of *config, event, config,* etc

- asynchronous execution:
  – arbitray message delay, different processor speeds
  – infinite sequence of *delivery or exec* events
- synchronous execution:
  – all messages in out-buffer delivered *at once*
  – followed by one exec event *in all processors*
  – infinite sequence of such **rounds**

Synchronous model is "simpler" than asynchronous model.

## Synchronous (lock-step) Execution Model



## Failure Modes

Different classes of why a system may fail:

- Crash
  faulty processor just stops. Idealization of reality
- Byzantine (arbitrary)
  conservative assumption, fits when failure model is unknown or malicious
- Self-stabilization:
  algorithm automatically recovers from transient corruption of state; appropriate for long-running apps

(from J. Welch)

## Overview

Selected "Self-" Algorithms (I)

- Introduction to coordination algorithms
- Leader election
- Notes on message passing models
- **Self-stabilization** (example: mutual exclusion)
- Logical clocks
- Distributed snapshot

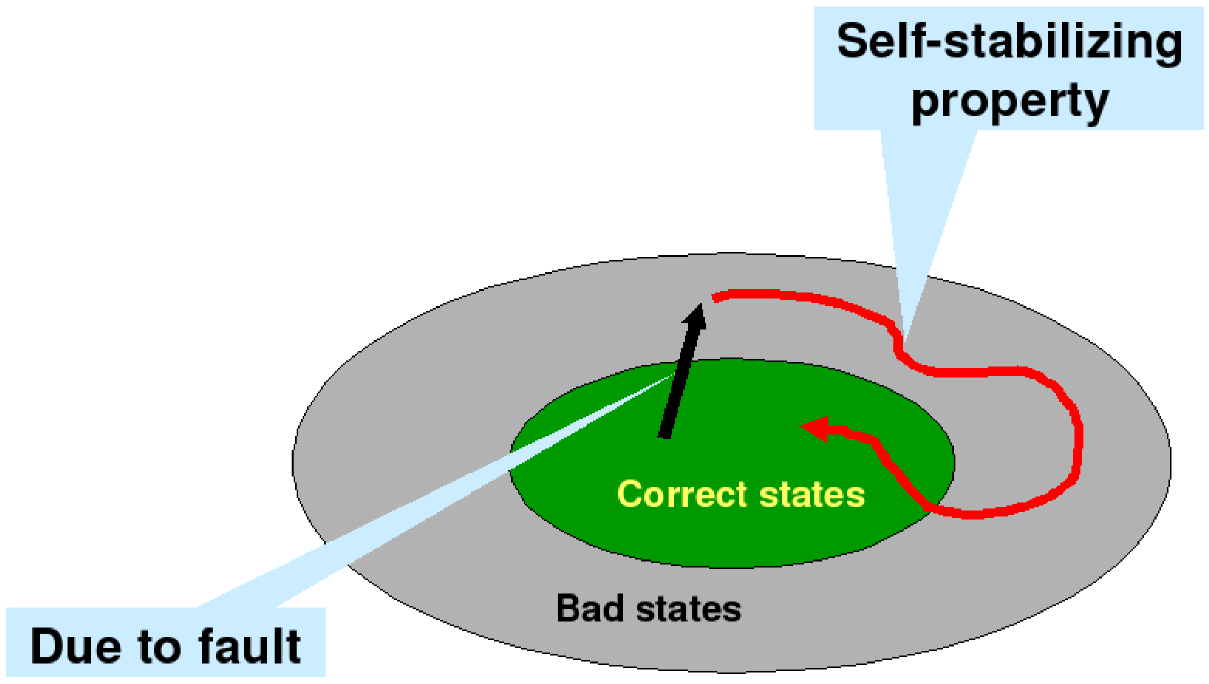
## Intro Self-Stabilization

Definition

- An algorithm is self-stabilizing, if it terminates in a correct state no matter what starting state it initially had.
- Assume continously running algo, and transient errors:
  The system will always go to a "legitime state" despite transient faults.

In reality, "any starting state" can be: value of variables, state of processes (crashed), messages in the channel etc

## Self-Stabilization, State Space View


Self-stabilizing property
Correct states
Bad states
Due to fault