Distributed Systems
Master of Science in Engineering in Computer Science

AA 2019/2020

LECTURE 5: LEADER ELECTION

Recap on Timing Assumptions

Synchronous

- timing assumptions are <u>explicit</u> either on
  - Bounds on process executions and communication channels, or
  - Existence of a common global clock, or
  - Both

Asynchronous

- there are no timing assumptions

Recap on Timing Assumptions

Partial synchrony requires abstract timing assumptions (after an unknown time t the system becomes synchronous)

Two choices:

1. Put assumption on the system model (including links and processes)
2. Create a separate abstractions that encapsulates those timing assumptions

Hide the complexity

Note: manipulating time inside a protocol/algorithm is complex and the correctness proof may become very involved and sometimes prone to errors

An alternative

Sometimes, we may be interested in knowing one process that is alive instead of monitoring failures

- E.g., Need of a coordinator

We can use a different oracle (called *leader election* module) that reports a process that is alive

Leader Election Specification

Module 2.7: Interface and properties of leader election

Module:

Name: LeaderElection, **instance** *le*.

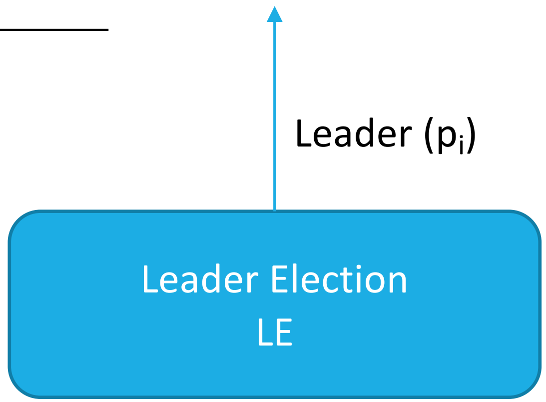
Events:

Indication: $\langle$ *le*, *Leader* | p $\rangle$: Indicates that process p is elected as leader.

Properties:

LE1: *Eventual detection:* Either there is no correct process, or some correct process is eventually elected as the leader.

LE2: *Accuracy:* If a process is leader, then all previously elected leaders have crashed.



Leader Election Implementation

Algorithm 2.6: Monarchical Leader Election

Implements:
  LeaderElection, **instance** *le*.

Uses:
  PerfectFailureDetector, **instance** $\mathcal{P}$.

upon event $\langle$ *le*, *Init* $\rangle$ **do**
  suspected := $\emptyset$;
  leader := $\perp$; → default value.

upon event $\langle$ $\mathcal{P}$, *Crash* | *p* $\rangle$ **do**
  suspected := *suspected* $\cup$ $\{p\}$;

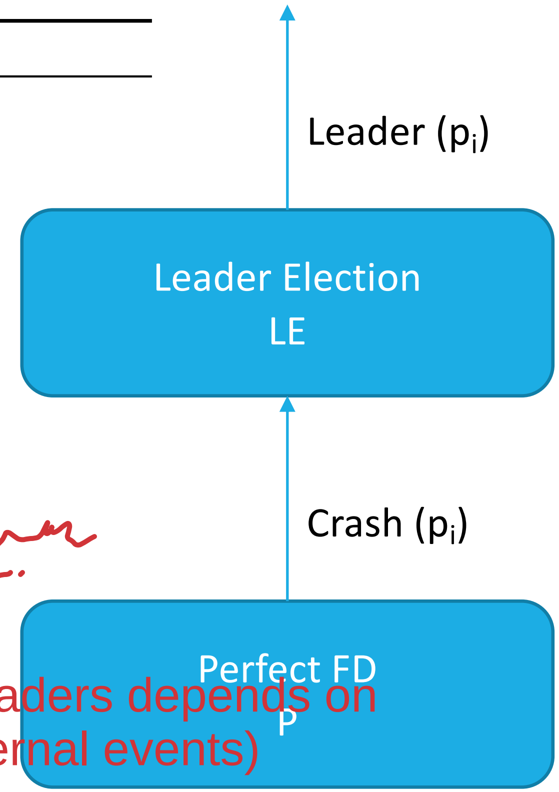
→ up to the designer, [illegible]

upon *leader* $\neq$ maxrank(Π \ *suspected*) **do**
  leader := maxrank(Π \ *suspected*);
  trigger $\langle$ *le*, *Leader* | *leader* $\rangle$;

We need a leader

The speed on checking new leaders depends on implementation details(e.g: internal events)

We are checking the Alive process only when a leader dies



(2) Seen at 10:48

Correctness

What if the Failure detector is not perfect?

It can introduce some mistakes, because of partially synchronous

I can lose accuracy property

You may change leader even if it’s alive

Eventual leader election (Ω)

Module 2.9: Interface and properties of the eventual leader detector

Module:

Name: EventualLeaderDetector, **instance** Ω.

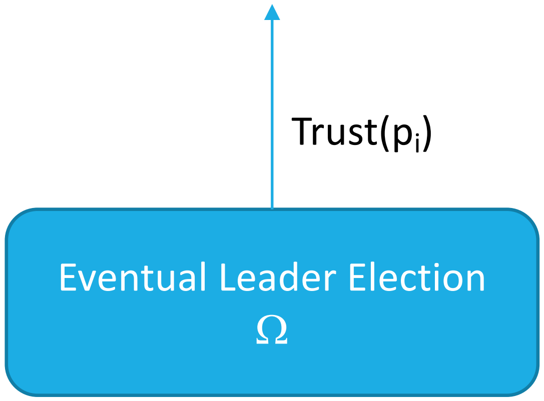
Events:

Indication: $\langle\ \Omega,\ \mathit{Trust} \mid p\ \rangle$: Indicates that process p is trusted to be leader.

Properties:

ELD1: *Eventual accuracy:* There is a time after which every correct process trusts some correct process.

ELD2: *Eventual agreement:* There is a time after which no two correct processes trust different correct processes.



Observation on Ω

Ω ensures that *eventually* correct processes will elect the same correct process as their leader

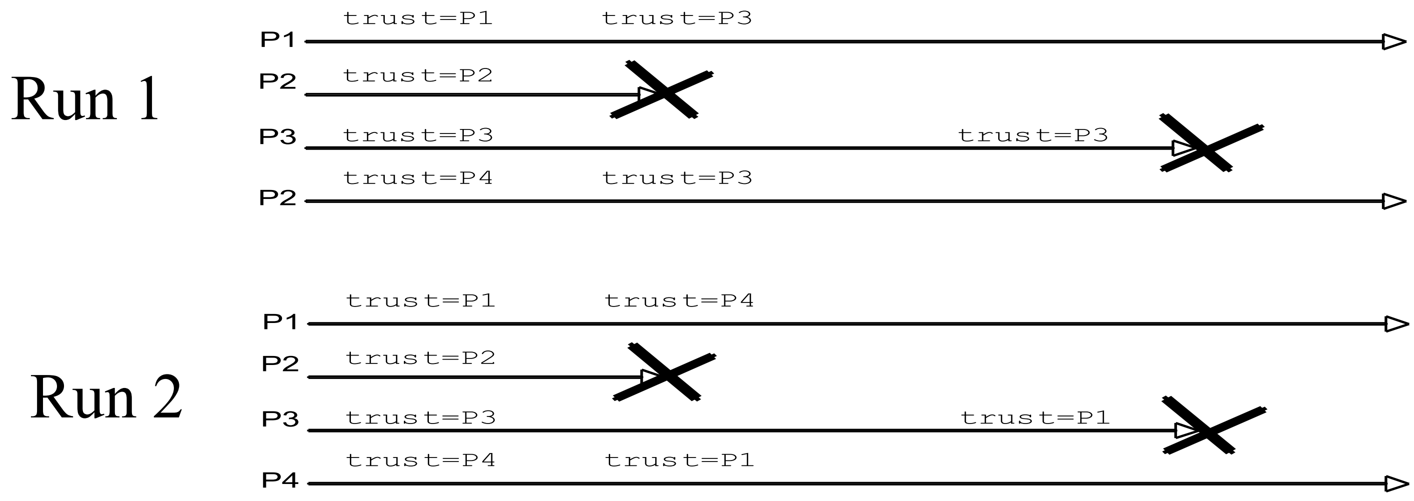
you will converge to the same leader.

Ω does not guarantee that

- Leaders change in an arbitrary manner and for an arbitrary period of time
- many leaders might be elected during the same period of time without having crashed

Once a unique leader is determined, and does not change again, we say that the leader has *stabilized*

Study of Properties



	Run 1	Run 2
Eventual Accuracy	Not verified	Verified
Eventual Agreement	Verified	Not verified

Eventual leader election (Ω)

Using Crash-stop process abstraction

- Obtained directly by <>P by using a deterministic rule on processes that are not suspected by <>P
- trust the process with the highest identifier among all processes that are not suspected by <>P

Assume the existence of a correct process (otherwise Ω cannot be built)

The role of the leader depends lamentation, it can be someone who have to manage something important.

Seen at 11:39
and 11:46

Ω Implementation

Algorithm 2.8: Monarchical Eventual Leader Detection

Implements:
EventualLeaderDetector, **instance** Ω.

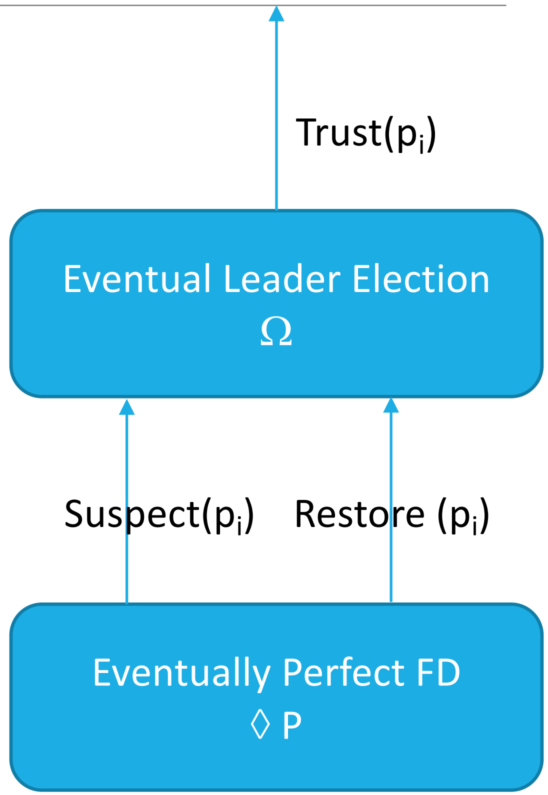
Uses:
EventuallyPerfectFailureDetector, **instance** $\Diamond\mathcal{P}$.

upon event $\langle\ \Omega, Init\ \rangle$ **do**
$suspected := \emptyset$;
$leader := \perp$; default.

upon event $\langle\ \Diamond\mathcal{P}, Suspect \mid p\ \rangle$ **do**
$suspected := suspected \cup \{p\}$;

upon event $\langle\ \Diamond\mathcal{P}, Restore \mid p\ \rangle$ **do**
$suspected := suspected \setminus \{p\}$;

upon $leader \neq \text{maxrank}(\Pi \setminus suspected)$ **do**
$leader := \text{maxrank}(\Pi \setminus suspected)$;
trigger $\langle\ \Omega, Trust \mid leader\ \rangle$;



Eventual leader election (Ω)

System model

- Crash-Recovery
- Partial synchrony

NO USE OF ORACLE

Under this assumption, a correct process means:

1. A process that does not crash or
2. A process that crashes, eventually recovers and never crashes again

Processes unstable will not be considered correct.

Ω With crash-recovery, fair lossy links and timeouts

Algorithm 2.9: Elect Lower Epoch

Implements:
  EventualLeaderDetector, **instance** Ω.

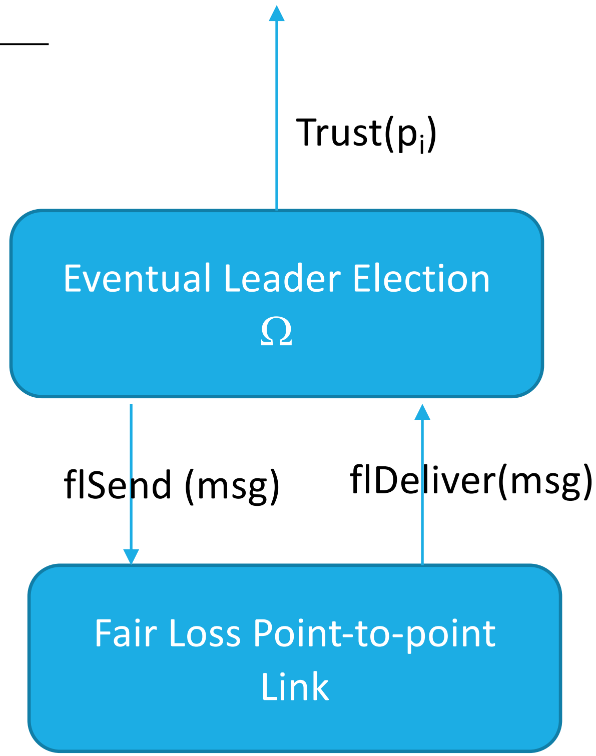
Uses:
  FairLossPointToPointLinks, **instance** *fll*.

upon event $\langle$ Ω, *Init* $\rangle$ **do**
  epoch := 0;
  store(*epoch*);
  candidates := $\emptyset$;
  trigger $\langle$ Ω, *Recovery* $\rangle$;

keeps track of how many times the process crashed and recovered

upon event $\langle$ Ω, *Recovery* $\rangle$ **do**
  leader := maxrank(Π);
  trigger $\langle$ Ω, *Trust* | *leader* $\rangle$;
  delay := Δ;
  retrieve(*epoch*);
  epoch := *epoch* + 1;
  store(*epoch*);
  forall $p \in \Pi$ **do**
    trigger $\langle$ *fll*, *Send* | p, [HEARTBEAT, *epoch*] $\rangle$;
  candidates := $\emptyset$;
  starttimer(*delay*);

Stable stored.



Stream at 12:05

Ω With crash-recovery, fair lossy links and timeouts

Algorithm 2.9: Elect Lower Epoch

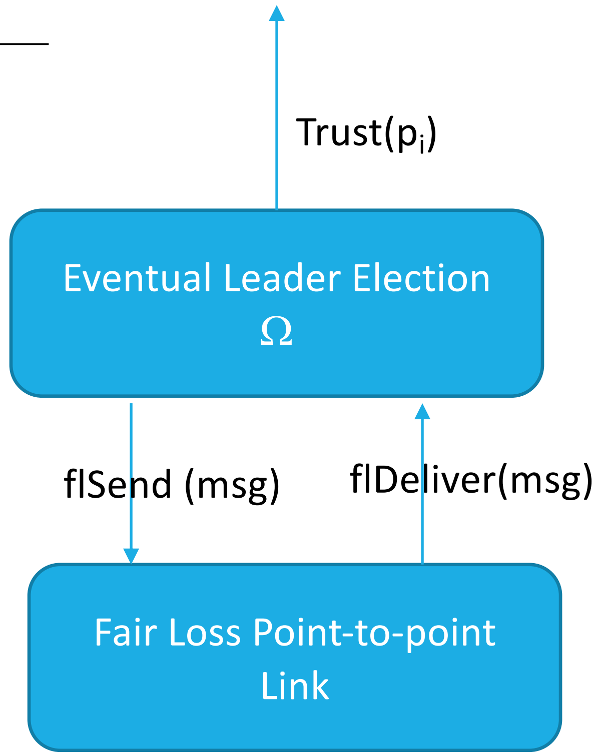
Implements:
EventualLeaderDetector, **instance** Ω.

Uses:
FairLossPointToPointLinks, **instance** *fll*.

upon event $\langle$ *Timeout* $\rangle$ **do**
  newleader := *select*(*candidates*);
  if *newleader* $\neq$ *leader* **then**
    delay := *delay* + Δ;
    leader := *newleader*;
    trigger $\langle$ Ω, *Trust* | *leader* $\rangle$;
  forall $p \in \Pi$ **do**
    trigger $\langle$ *fll*, *Send* | p, [HEARTBEAT, *epoch*] $\rangle$;
  candidates := $\emptyset$;
  starttimer(*delay*);

upon event $\langle$ *fll*, *Deliver* | q, [HEARTBEAT, *ep*] $\rangle$ **do**
  if exists $(s, e) \in$ *candidates* such that $s = q \wedge e < ep$ **then**
    candidates := *candidates* $\setminus \{(q, e)\}$;
  candidates := *candidates* $\cup$ (q, ep);

deterministic function returning one process among all candidates (i.e., process with the lowest epoch number and among the ones with the same epoch number the one with the lowest identifier)



Seminar at 12:55

References

C. Cachin, R. Guerraoui and L. Rodrigues. Introduction to Reliable and Secure Distributed Programming, Springer, 2011

- Chapter 2 – from Section 2.6.1 to Section 2.6.5