10/10/23

Dependable Distributed Systems Master of Science in Engineering in Computer Science

AA 2023/2024

LECTURE 7: LEADER ELECTION

Recap on Timing Assumptions

Synchronous

- timing assumptions are explicit 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

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

monitor the process and elect an alive process

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

as master / slave approach

Leader Election Specification

Module 2.7: Interface and properties of leader election

Module:

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

Events:

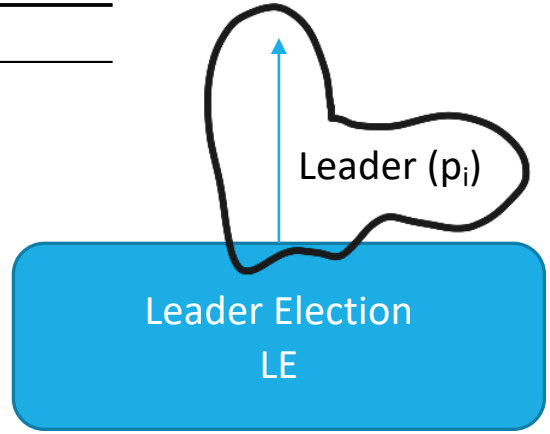
Indication: ⟨ *le*, *Leader* | *p* ⟩: Indicates that process *p* is elected as leader.

Properties:

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

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

↳ he will remain the leader, until he will crash



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 := $\perp$; → no one is the leader

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

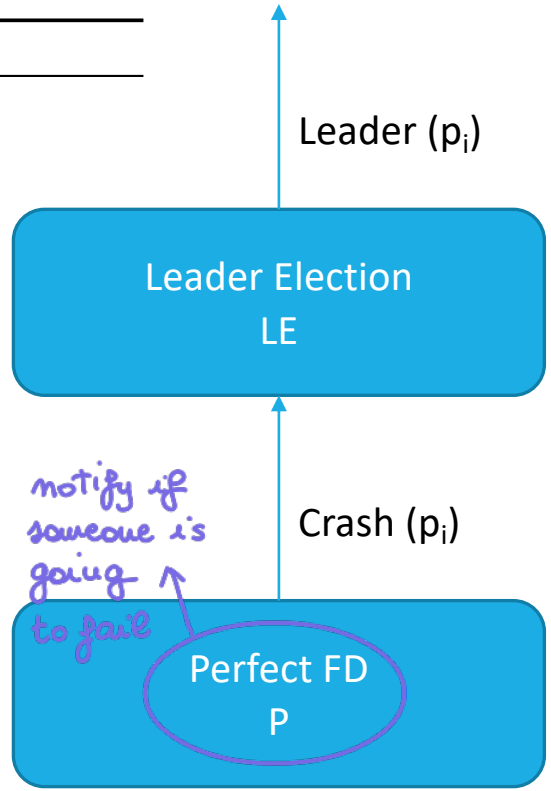
order ID in the system — SORT TAKE THE MAX

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

leader event

p_{10} CRASH

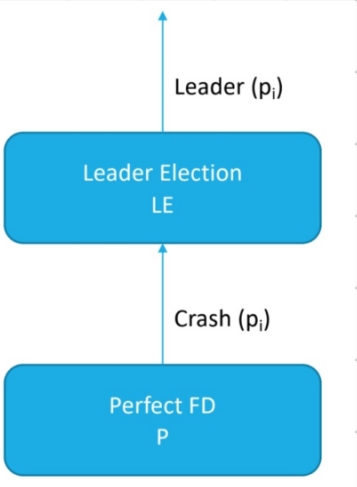
p_{10} is the leader



7: LEADER ELECTION

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.



Algorithm 2.6: Monarchical Leader Election

Implements:

LeaderElection, **instance** *le*.

Uses:

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

upon event $\langle$ *le*, *Init* $\rangle$ **do**

① *suspected* := $\emptyset$;
leader := $\perp$; no one is the leader

upon event $\langle$ $\mathcal{P}$, *Crash* | *p* $\rangle$ **do**

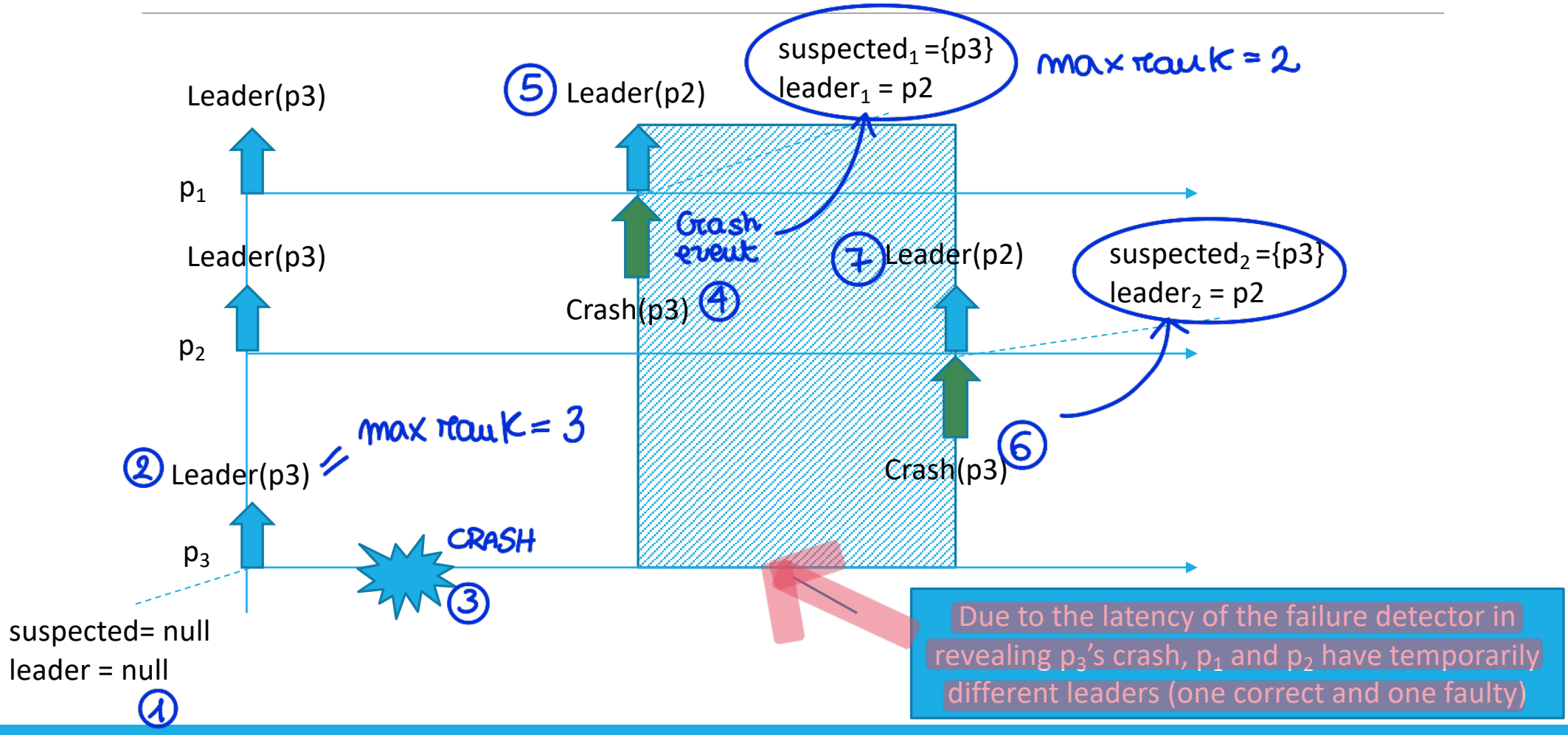
suspected := *suspected* $\cup$ $\{p\}$;

sort the list — all the process — the crashed processes

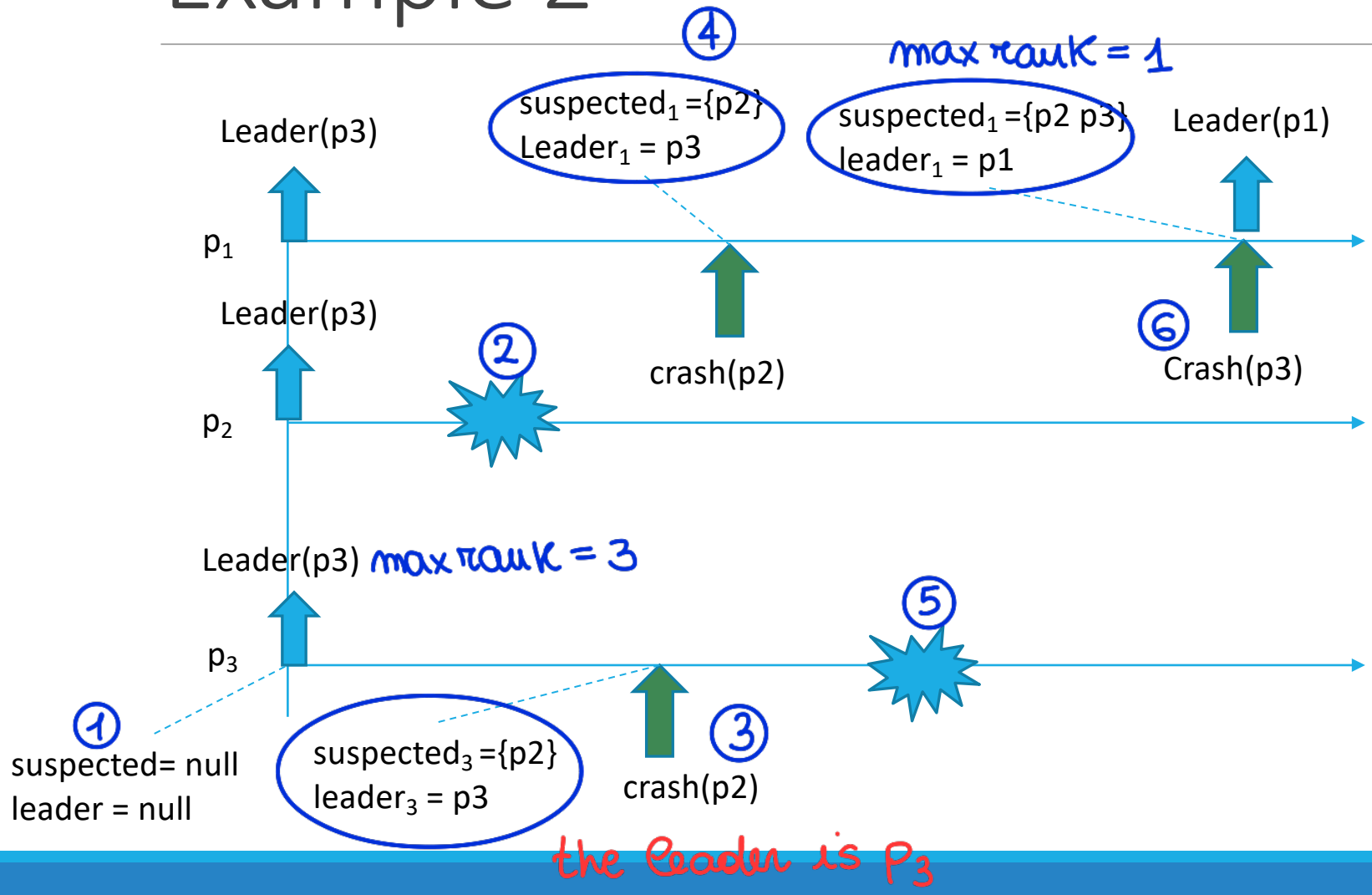
upon *leader* $\neq$ maxrank($\Pi \setminus$ *suspected*) **do** → WHO IS THE 1° IN THE LIST?

② *leader* := maxrank($\Pi \setminus$ *suspected*);
trigger $\langle$ *le*, *Leader* | *leader* $\rangle$;

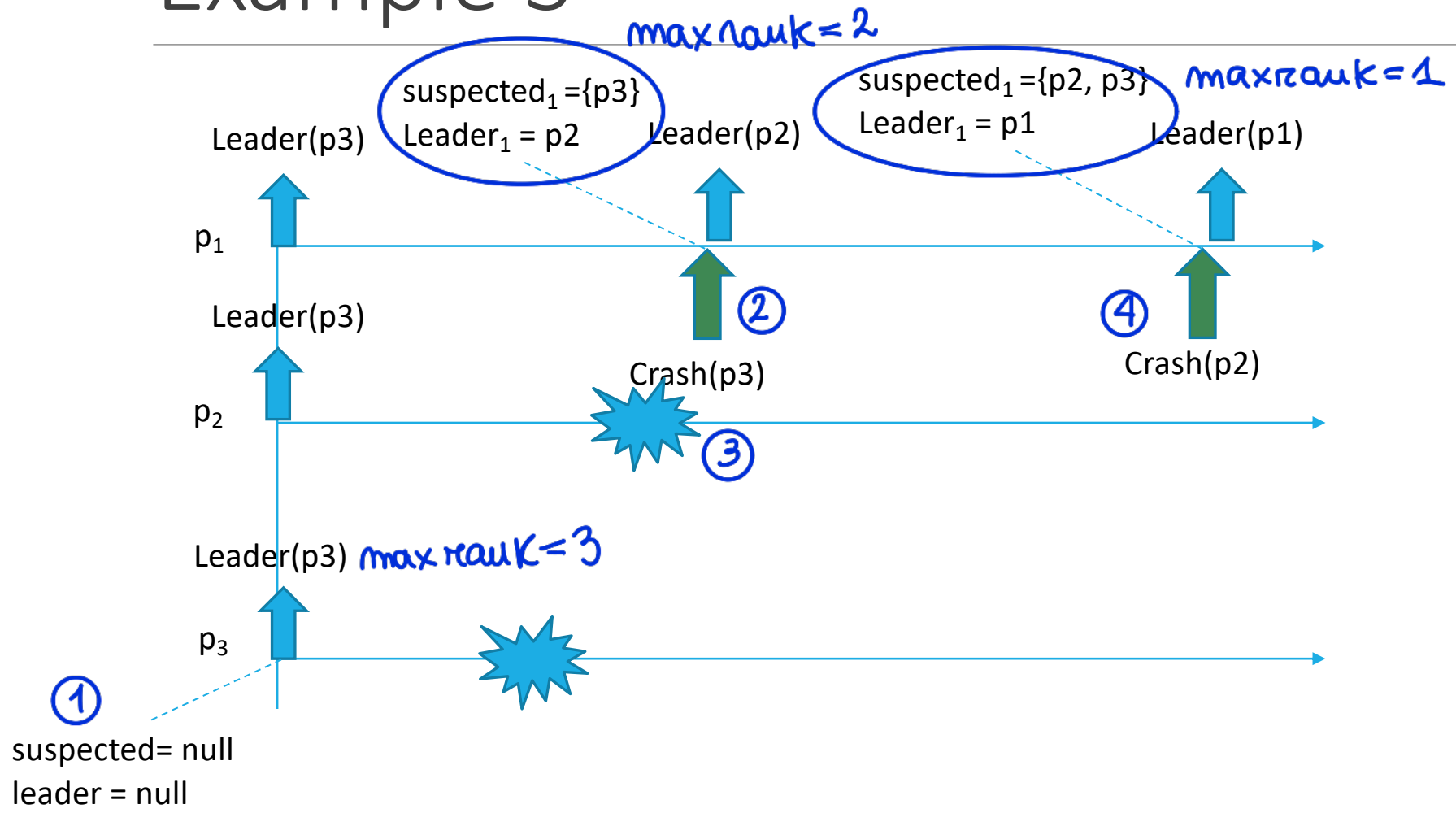
Example


suspected$_1$ ={p3}
leader$_1$ = p2
max rank = 2
Leader(p3)
5 Leader(p2)
p$_1$
Leader(p3)
Crash event
7 Leader(p2)
suspected$_2$ ={p3}
leader$_2$ = p2
Crash(p3) 4
p$_2$
max rank = 3
2 Leader(p3)
Crash(p3)
6
CRASH
p$_3$
3
suspected= null
leader = null
1
Due to the latency of the failure detector in revealing p$_3$'s crash, p$_1$ and p$_2$ have temporarily different leaders (one correct and one faulty)

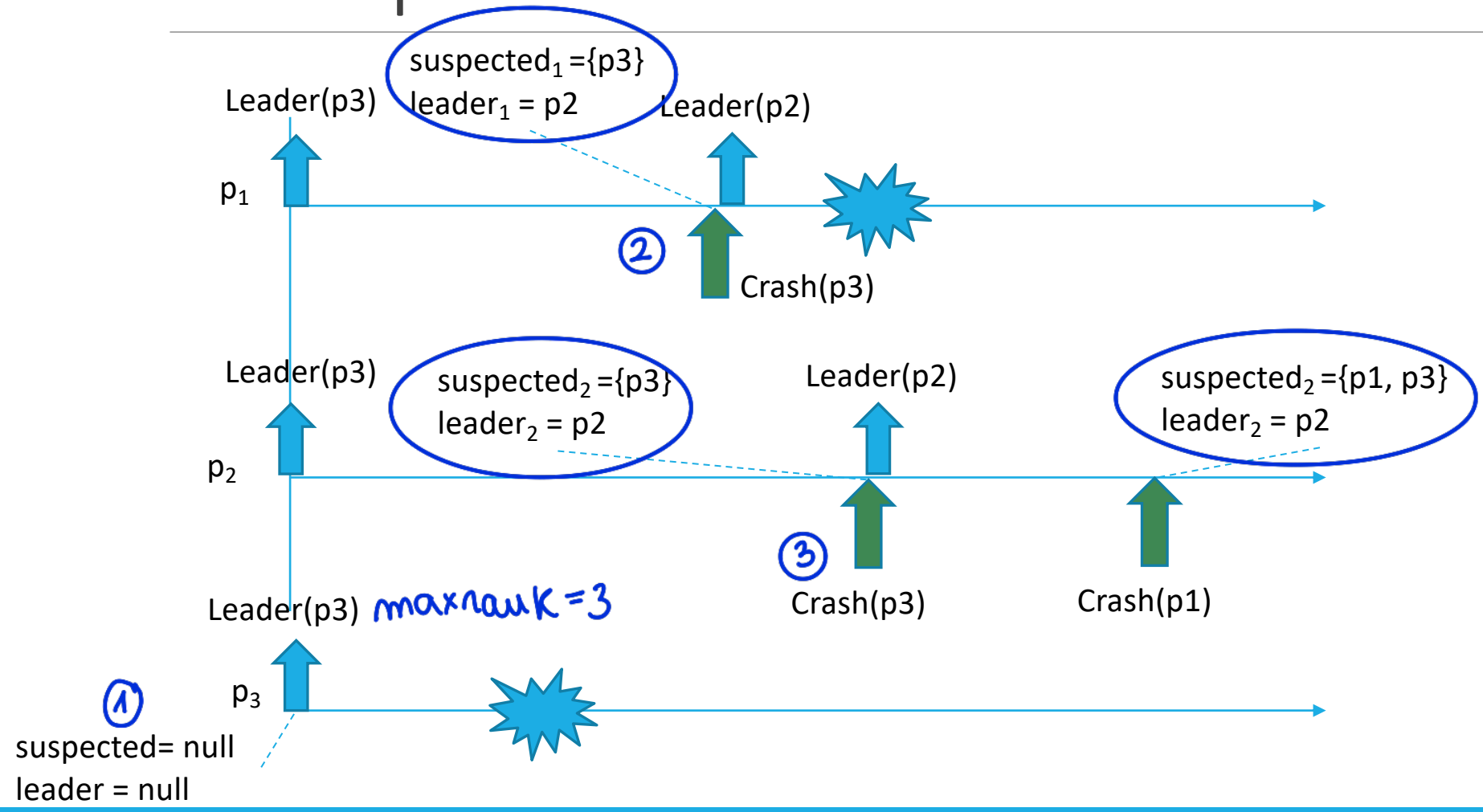
Example 2


4
max rank = 1
suspected$_1$ ={p2}
Leader$_1$ = p3
suspected$_1$ ={p2 p3}
leader$_1$ = p1
Leader(p3)
Leader(p1)
p$_1$
Leader(p3)
crash(p2)
6
Crash(p3)
2
p$_2$
Leader(p3) max rank = 3
5
p$_3$
1
suspected= null
leader = null
suspected$_3$ ={p2}
leader$_3$ = p3
crash(p2)
3
the leader is p$_3$

Example 3


maxrank=2
suspected$_1$ ={p3}
Leader$_1$ = p2
suspected$_1$ ={p2, p3}
Leader$_1$ = p1
maxrank=1
Leader(p3)
Leader(p2)
Leader(p1)
p_1
Leader(p3)
②
④
Crash(p3)
Crash(p2)
p_2
③
Leader(p3) maxrank=3
p_3
①
suspected= null
leader = null

Example 4


suspected$_1$ ={p3}
leader$_1$ = p2
Leader(p3)
Leader(p2)
p$_1$
②
Crash(p3)
Leader(p3)
suspected$_2$ ={p3}
leader$_2$ = p2
Leader(p2)
suspected$_2$ ={p1, p3}
leader$_2$ = p2
p$_2$
③
Crash(p3)
Crash(p1)
Leader(p3)
maxrank=3
p$_3$
①
suspected= null
leader = null

Correctness

What if the Failure detector is not perfect?

Eventual leader election (Ω)

Module 2.9: Interface and properties of the eventual leader detector

Module:

Name: EventualLeaderDetector, **instance** Ω.

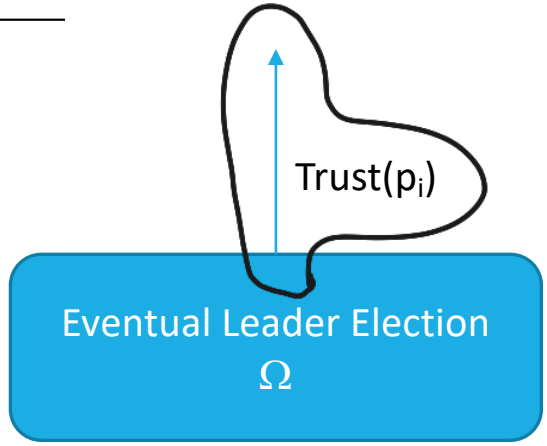
Events:

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.

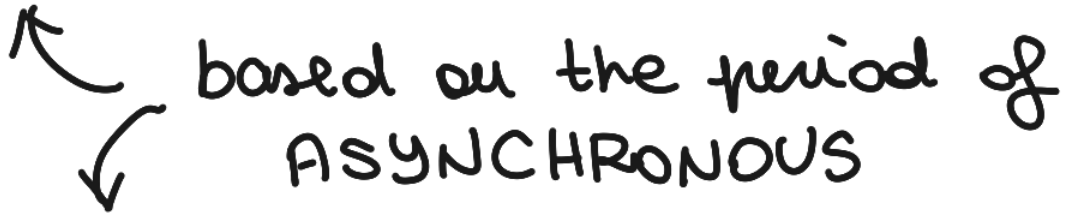
ELD2: *Eventual agreement:* There is a time after which no two correct processes trust different correct processes.



lost the ACCURACY → if a p is a leader, the previous leaders are crashed

Observation on Ω

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

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

IMPORTANT ASSUMPTION

- There always exists at least one correct process (otherwise Ω cannot be built)

Ω Implementation

Algorithm 2.8: Monarchical Eventual Leader Detection

Implements:
EventualLeaderDetector, **instance** Ω.

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

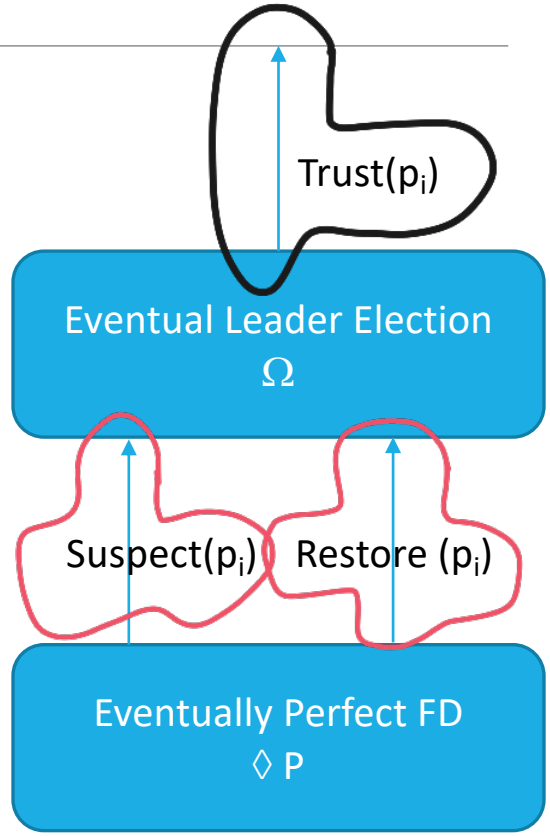
upon event $\langle\ \Omega, Init\ \rangle$ **do**
$suspected := \emptyset$;
$leader := \perp$;

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

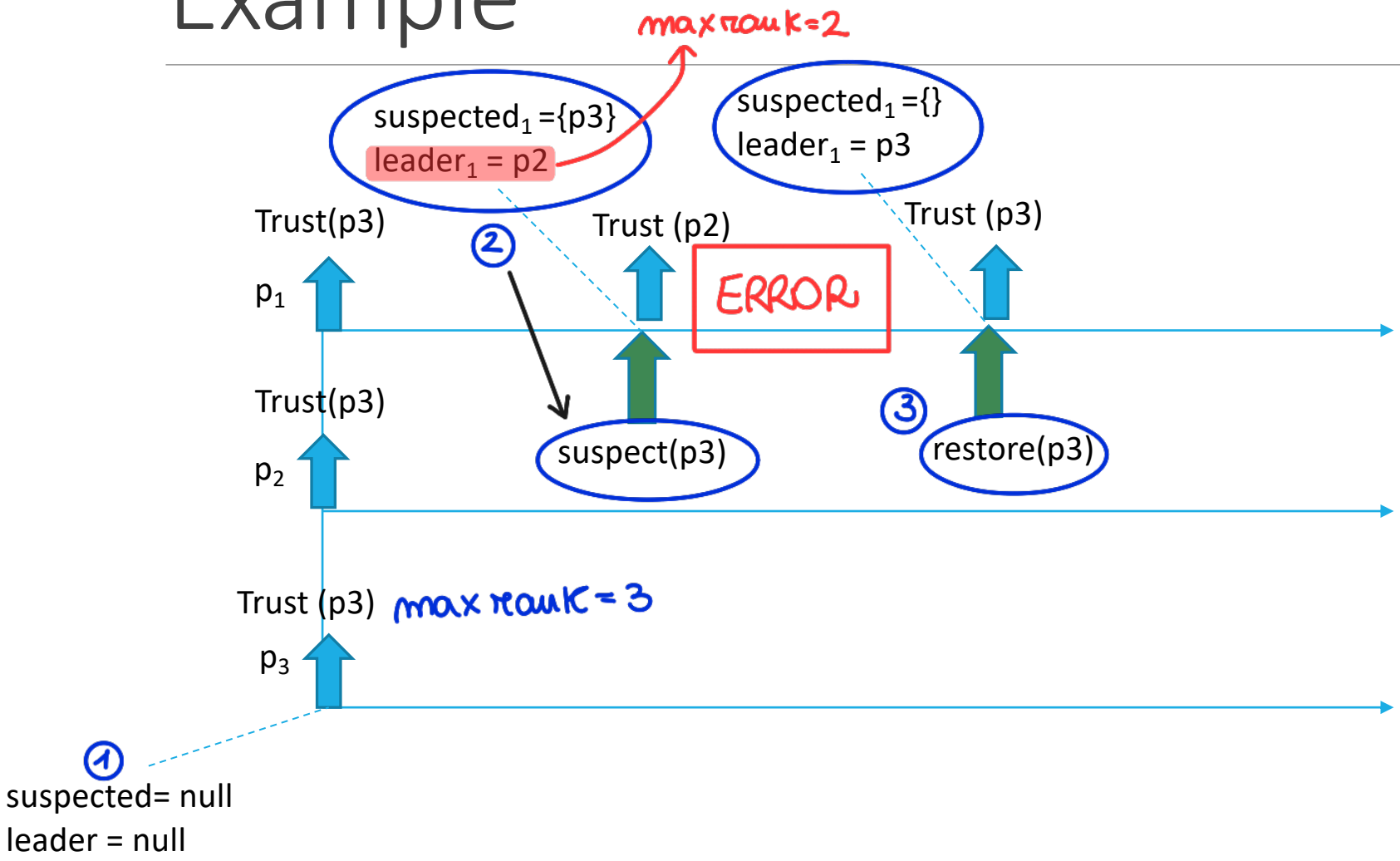
upon event $\langle\ \diamond\mathcal{P}, Restore \mid p\ \rangle$ **do**
$suspected := suspected \setminus \{p\}$;

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

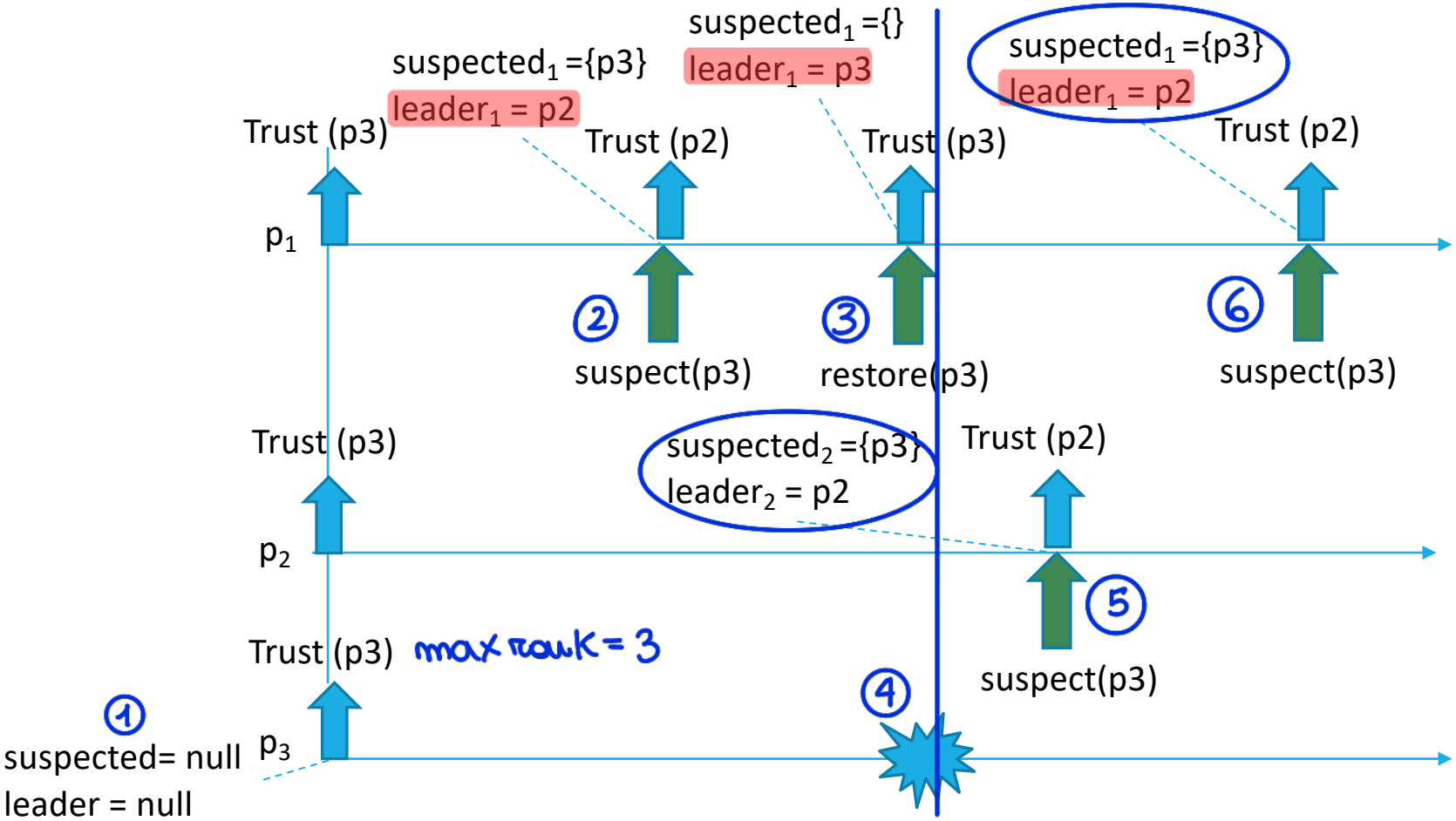
} deterministic rule



Example


max rank=2
suspected$_1$ ={p3}
leader$_1$ = p2
suspected$_1$ ={}
leader$_1$ = p3
Trust(p3)
p$_1$
Trust (p2)
Trust (p3)
ERROR
2
Trust(p3)
p$_2$
suspect(p3)
3
restore(p3)
Trust (p3)
max rank = 3
p$_3$
1
suspected= null
leader = null

Example 2


suspected$_1$ ={}
leader$_1$ = p3
suspected$_1$ ={p3}
leader$_1$ = p2
suspected$_1$ ={p3}
leader$_1$ = p2
Trust (p3)
Trust (p2)
Trust (p3)
Trust (p2)
p$_1$
2
suspect(p3)
3
restore(p3)
6
suspect(p3)
Trust (p3)
suspected$_2$ ={p3}
leader$_2$ = p2
Trust (p2)
p$_2$
5
Trust (p3)
max rank = 3
suspect(p3)
1
4
suspected= null
leader = null
p$_3$

Eventual leader election (Ω)

System model

- Crash-Recovery
- Partial synchrony

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

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

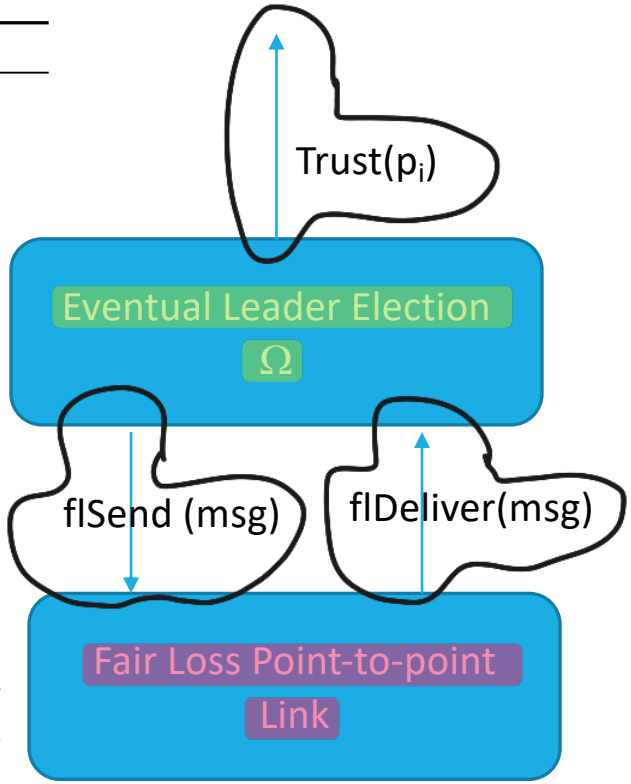
keeps track of how many times the process crashed and recovered

period of time in which you are suspected crash and then recover

count how many time you have CRASH-RECOVER

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

every p know the C-R of a process



Ω With crash-recovery, fair lossy links and timeouts

Algorithm 2.9: Elect Lower Epoch

Implements:

EventualLeaderDetector, **instance** Ω.

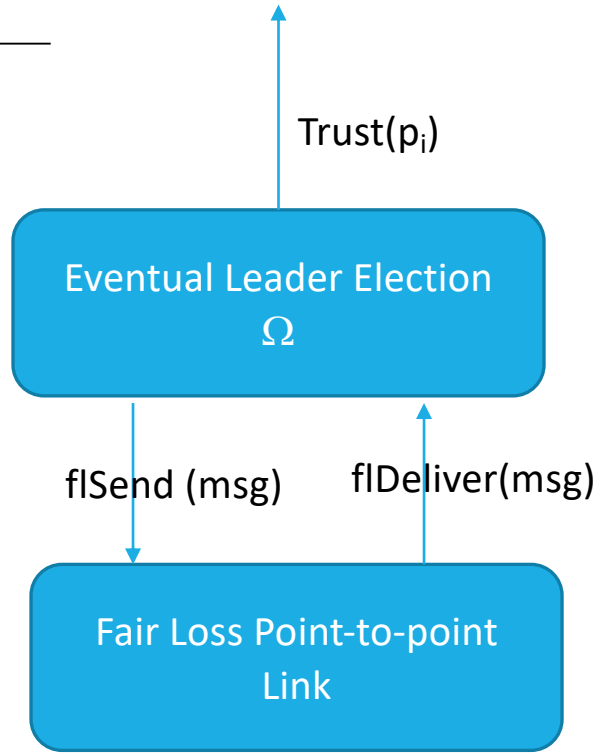
Uses:

FairLossPointToPointLinks, **instance** *fll*.

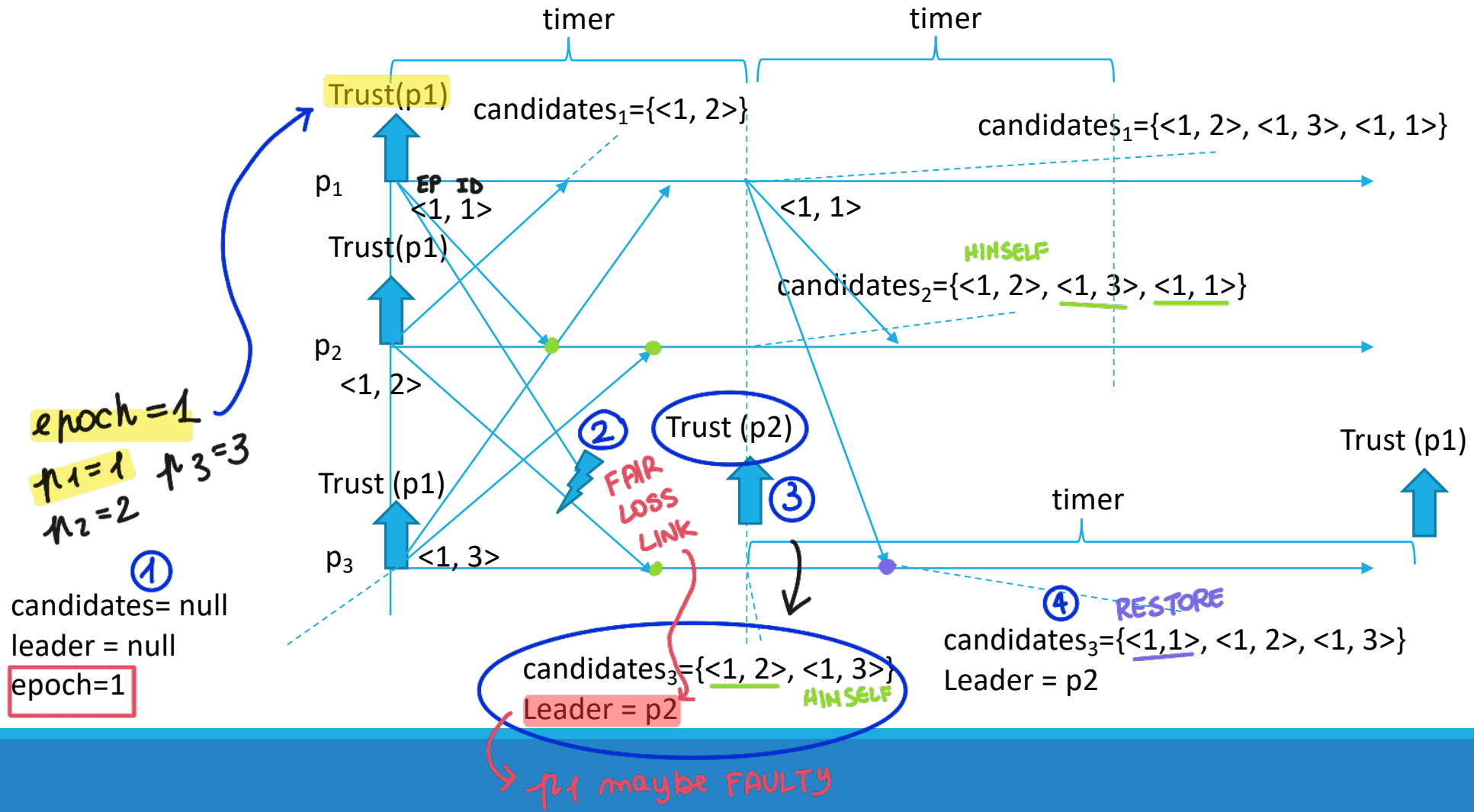
upon event $\langle$ *Timeout* $\rangle$ **do**
- *newleader* := *select*(*candidates*);
- **if** *newleader* $\neq$ *leader* **then**
  - *delay* := *delay* + Δ;
  - *leader* := *newleader*;
  - **trigger** $\langle$ Ω, *Trust* | *leader* $\rangle$;
- **forall** $p \in \Pi$ **do**
  - **trigger** $\langle$ *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* $\cup (q, ep)$;

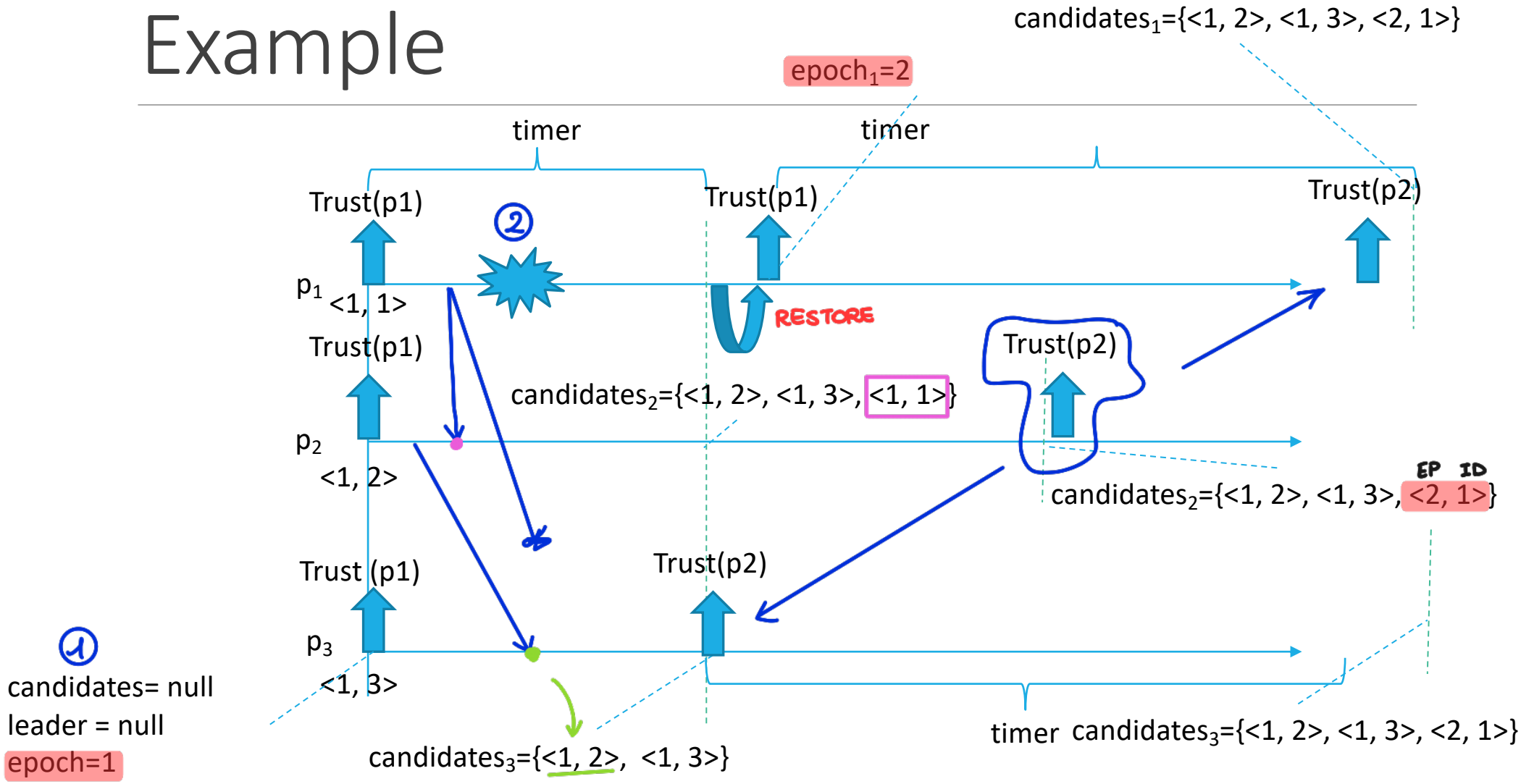
deterministic function returning one process among all candidates (i.e., process with the lowest epoch number and among the ones with the same epoch number the one with the lowest identifier)



Example


timer
timer
Trust(p1)
candidates$_1$={<1, 2>}
candidates$_1$={<1, 2>, <1, 3>, <1, 1>}
p_1
EP ID
<1, 1>
<1, 1>
Trust(p1)
HIMSELF
candidates$_2$={<1, 2>, <1, 3>, <1, 1>}
p_2
<1, 2>
epoch=1
p_1=1
p_3=3
p_2=2
2
Trust (p2)
Trust (p1)
FAIR LOSS LINK
3
timer
Trust (p1)
p_3
<1, 3>
1
candidates= null
leader = null
epoch=1
4
RESTORE
candidates$_3$={<1,1>, <1, 2>, <1, 3>}
Leader = p2
candidates$_3$={<1, 2>, <1, 3>}
HIMSELF
Leader = p2
p_1 maybe FAULTY

Example


candidates$_1$={<1, 2>, <1, 3>, <2, 1>}
epoch$_1$=2
timer
timer
Trust(p1)
Trust(p1)
Trust(p2)
②
p$_1$ <1, 1>
RESTORE
Trust(p1)
candidates$_2$={<1, 2>, <1, 3>, <1, 1>}
Trust(p2)
p$_2$
<1, 2>
EP ID
candidates$_2$={<1, 2>, <1, 3>, <2, 1>}
Trust (p1)
Trust(p2)
①
p$_3$
<1, 3>
candidates= null
leader = null
epoch=1
candidates$_3$={<1, 2>, <1, 3>}
timer
candidates$_3$={<1, 2>, <1, 3>, <2, 1>}

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