04|10|23

Dependable Distributed Systems Master of Science in Engineering in Computer Science

AA 2023/2024

LECTURE 6: FAILURE DETECTION

Recap on Timing Assumptions

Synchronous

- timing assumptions are explicit and known
  - Upper bounds on process execution time
  - Upper bounds on communication time
  - Existence of a common global clock (apart from synchronization error)

abstraction of physical time

Asynchronous

- there are no timing assumptions — no physical clock
  - the only ordering is given by the happen-before relationship
  - it is possible to build scalar or vector clocks → causal relat.

Partial synchrony

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

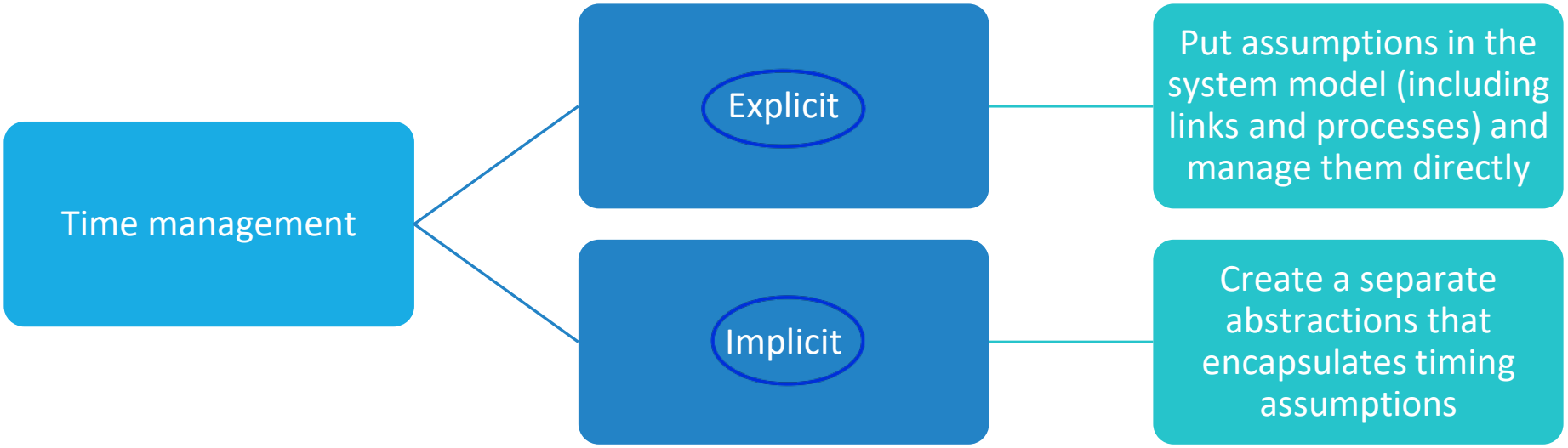
→ asynchronous most of the time

then synchronous

Recap on Timing Assumptions

NOTE

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



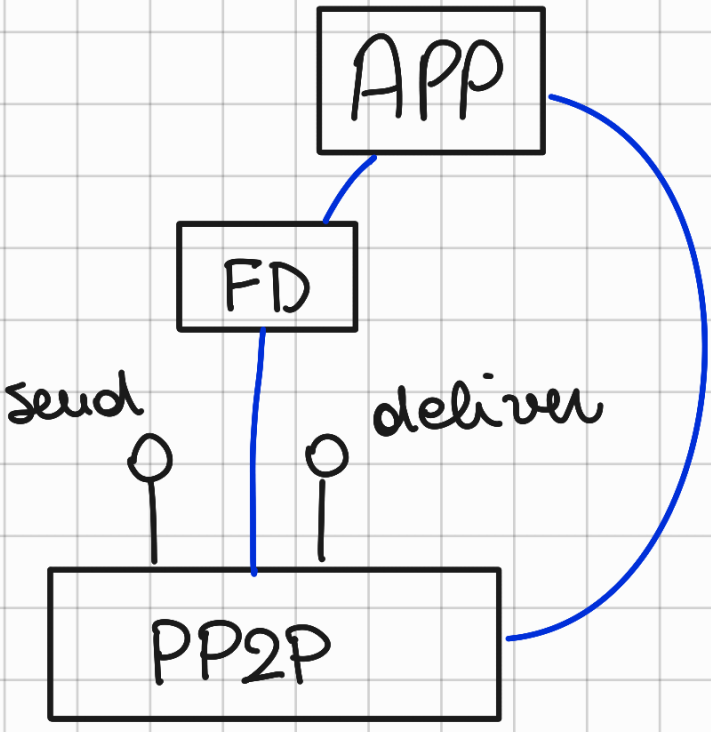
Failure Detector Abstraction

A Failure Detector is **an oracle providing information about the failure state of a process** in the system

- it is a software module to be used together with process and link abstractions
- It encapsulates timing assumptions of either partially synchronous or fully synchronous system
- *Stronger are the timing assumption, more accurate the information provided by a failure detector will be*

A failure detector is generally described by two properties:

- **Accuracy** (informally, it is the ability to avoid mistakes in the detection)
- **Completeness** (informally, it is ability to detect all failures)



ACCURACY : avoid mistakes in FD

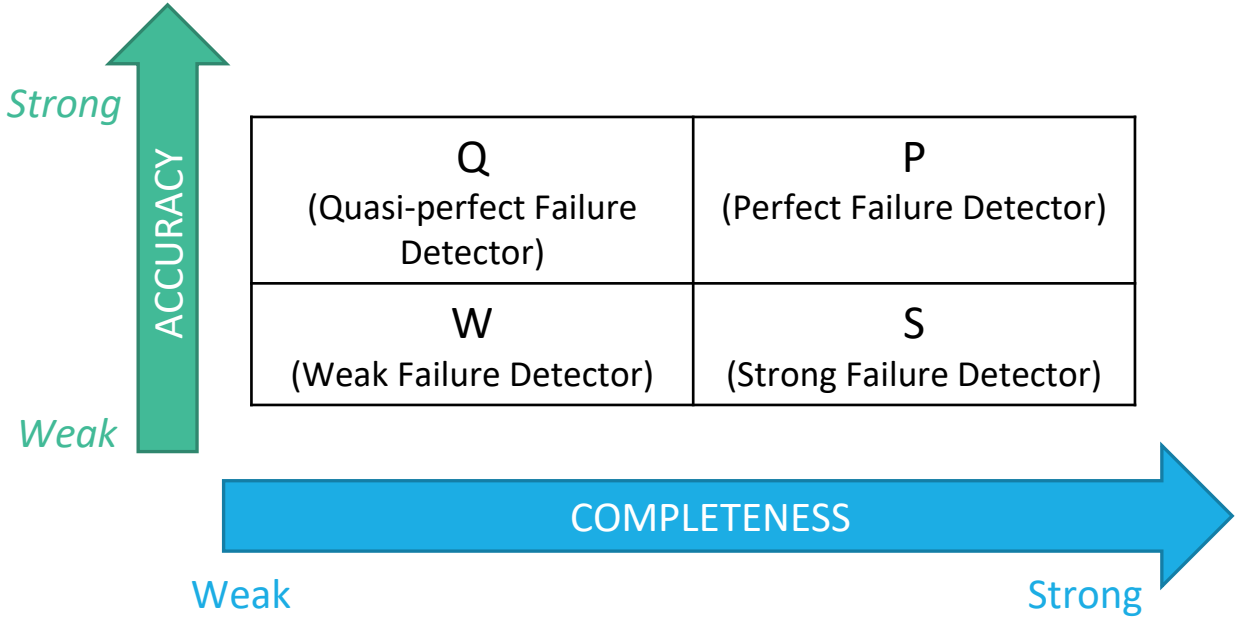
COMPLETENESS : detect all failures

Failure Detectors Classification

Correct processes are never suspected

Some correct process is never suspected

there is a process that never crash



Eventually every process that crashes is permanently suspected by **some** correct process

notice by some processes → not all to view the failure

CRASH FAULT → stop all action

Eventually every process that crashes is permanently suspected by **every** correct process

Failure Detectors Classification

OBSERVATION

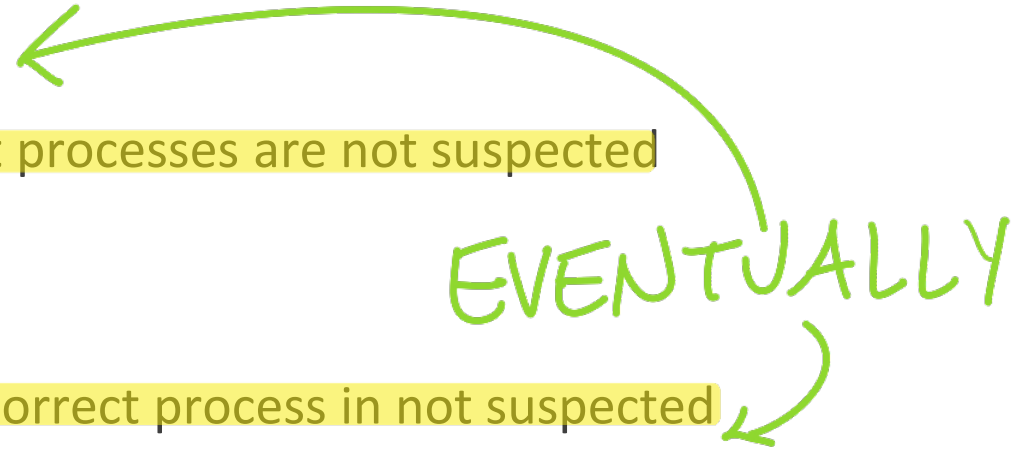
- W guarantees that there is at least one correct process that is never suspected.
- Practically speaking, this could be difficult to achieve and thus it is worth to consider accuracy properties to hold *eventually*

EVENTUAL STRONG ACCURACY

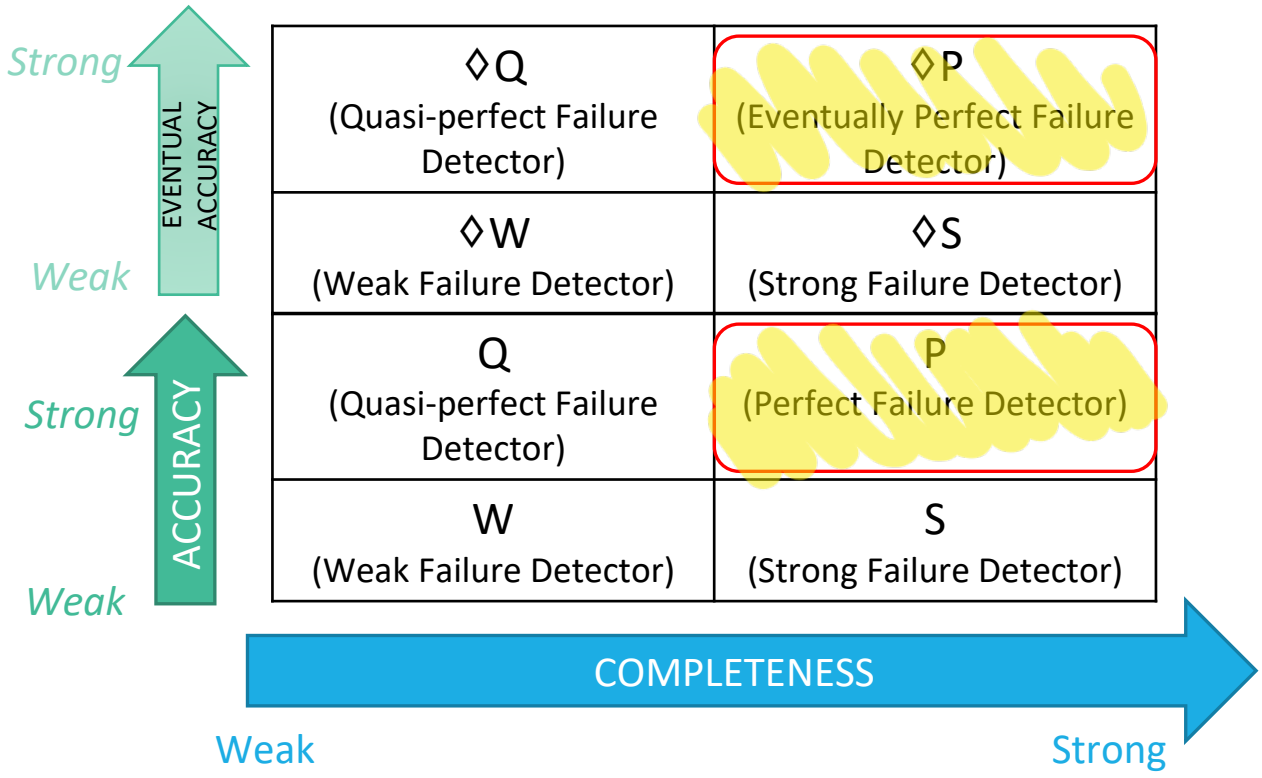
- There is a time after which correct processes are not suspected

EVENTUAL WEAK ACCURACY

- There is a time after which some correct process in not suspected



Failure Detectors Classification



Perfect Failure detectors (P)

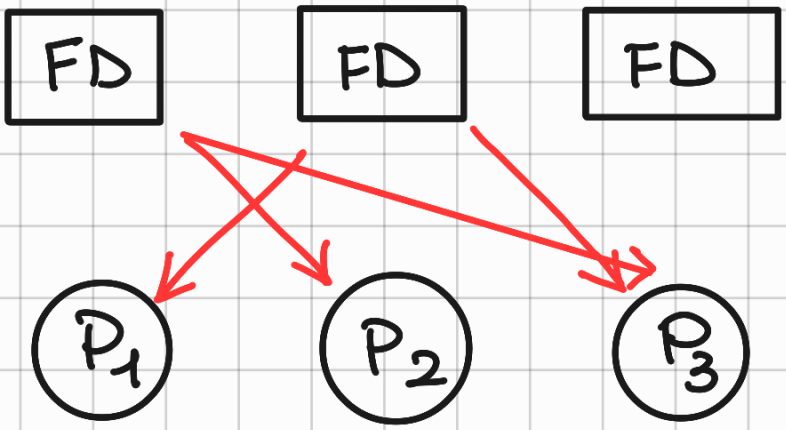
System model

- synchronous system
- crash failures

based on behaviour: WE KNOW THE TOTAL EXECUTION

some processes can stop their execution

Using its own clock and the bounds of the synchrony model, a process can infer if another process has crashed



Detect a crash: send messages to all processes and check the respouse if it anive or not

Perfect failure detectors (P) Specification

Module 2.6: Interface and properties of the perfect failure detector

Module:

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

Events:

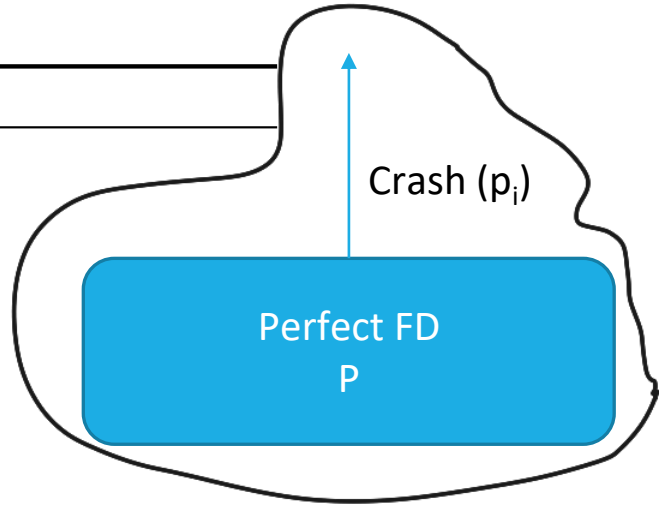
Indication: $\langle\ \mathcal{P}, Crash \mid p\ \rangle$: Detects that process p has crashed.

Properties:

not in a specific t

PFD1: *Strong completeness:* Eventually, every process that crashes is permanently detected by every correct process.

PFD2: *Strong accuracy:* If a process p is detected by any process, then p has crashed.



Perfect failure detectors (P) Implementation

Algorithm 2.5: Exclude on Timeout

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

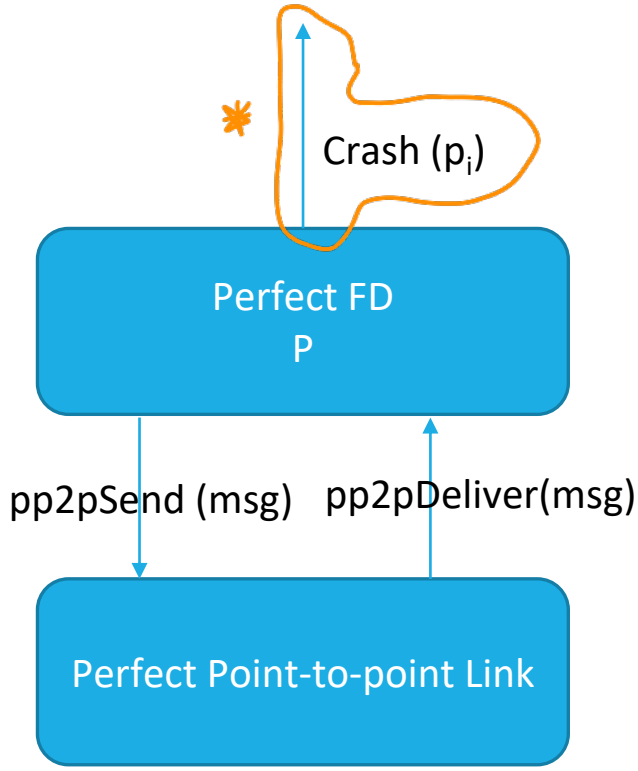
Uses:
PerfectPointToPointLinks, **instance** pl.

upon event $\langle\ \mathcal{P}, Init\ \rangle$ **do**
$alive := \Pi$; all processes of system
$detected := \emptyset$;
$starttimer(\Delta)$;

upon event $\langle\ Timeout\ \rangle$ **do**
forall $p \in \Pi$ **do**
if $(p \notin alive) \wedge (p \notin detected)$ **then**
$detected := detected \cup \{p\}$;
trigger $\langle\ \mathcal{P}, Crash \mid p\ \rangle$; ✱
trigger $\langle\ pl, Send \mid p, [\text{HEARTBEATREQUEST}]\ \rangle$;
$alive := \emptyset$;
$starttimer(\Delta)$;

upon event $\langle\ pl, Deliver \mid q, [\text{HEARTBEATREQUEST}]\ \rangle$ **do**
trigger $\langle\ pl, Send \mid q, [\text{HEARTBEATREPLY}]\ \rangle$;

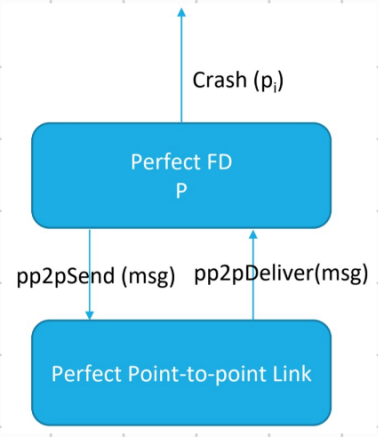
upon event $\langle\ pl, Deliver \mid p, [\text{HEARTBEATREPLY}]\ \rangle$ **do**
$alive := alive \cup \{p\}$;



6: PERFECT FAILURE DETECTOR

PFD1: *Strong completeness:* Eventually, every process that crashes is permanently detected by every correct process.

PFD2: *Strong accuracy:* If a process p is detected by any process, then p has crashed.



Algorithm 2.5: Exclude on Timeout

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

Uses:
PerfectPointToPointLinks, **instance** *pl*.

upon event $\langle\ \mathcal{P}, Init\ \rangle$ **do**
alive := Π; all processes of system
detected := $\emptyset$; is empty
starttimer(Δ);

upon event $\langle$ *Timeout* $\rangle$ **do**
forall $p \in \Pi$ **do**
if $(p \notin alive) \wedge (p \notin detected)$ **then** CRASH
detected := *detected* $\cup \{p\}$;
trigger $\langle\ \mathcal{P}, Crash \mid p\ \rangle$;
trigger $\langle\ pl, Send \mid p, [\text{HEARTBEATREQUEST}]\ \rangle$; PING : p_1 are you alive? ...
alive := $\emptyset$; every p is stopped
starttimer(Δ);

upon event $\langle\ pl, Deliver \mid q, [\text{HEARTBEATREQUEST}]\ \rangle$ **do**
trigger $\langle\ pl, Send \mid q, [\text{HEARTBEATREPLY}]\ \rangle$; : p_1 says: I'm alive

upon event $\langle\ pl, Deliver \mid p, [\text{HEARTBEATREPLY}]\ \rangle$ **do**
alive := *alive* $\cup \{p\}$; update "alive"

if p_m CRASH this part doesn't exist

TIMEOUT: is equal to RTT + time processing
(generate request, ...)

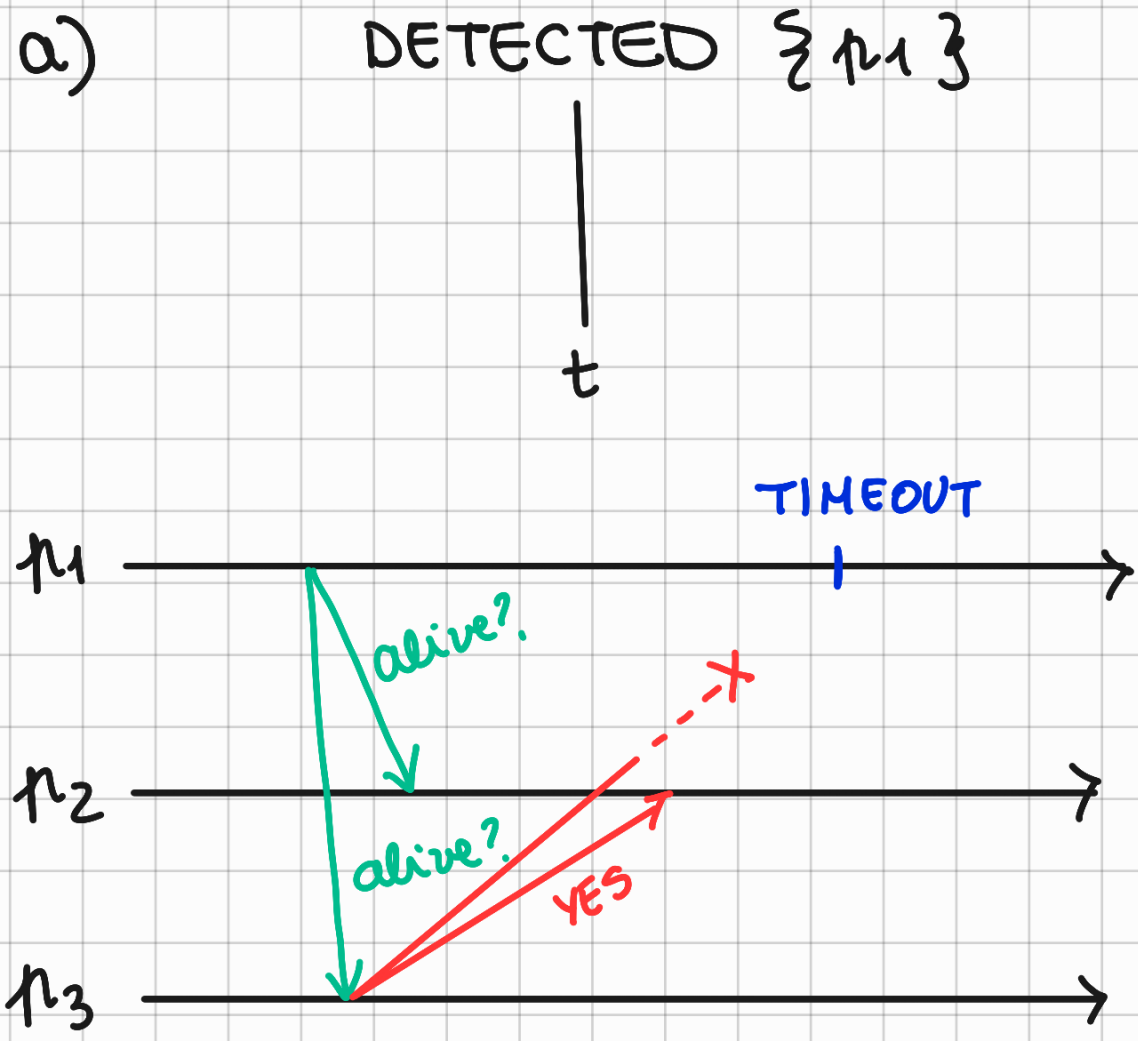
Correctness

- To prove the correctness, we must prove that both *Strong Completeness* and *Strong Accuracy* are satisfied
- What if we select a timeout too long?
- What if we select a timeout too short?
- What if links are fair loss? m can be lost — you can invalidate the accuracy: a process can reply but the m can be lost → TARGET AS CRASHED

a) If a process p_c is in *detected* of a process p_i at time t , is p_c alive at time t?

b) If a process p_c is in *detected* of a process p_i at time t, is p_c is in *detected* of each process p_j at time t?

a)



After 2 TIMEOUT all the processes know the failure of a specific process

Eventually perfect failure detectors (◊P)

System model

- partial synchrony
- crash failures
- perfect point-to-point links

Crashes can be accurately detected only after a (unknown) time *t*

- Before time *t* the systems behaves as an asynchronous one
- The failure detector may make mistake before time *t* considering correct processes as crashed.
- The notion of detection becomes *suspicious*

Eventually perfect failure detectors (◊P) Specification

Module 2.8: Interface and properties of the eventually perfect failure detector

Module:

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

Events:

Indication: $\langle\ \diamond\mathcal{P}, \mathit{Suspect} \mid p\ \rangle$: Notifies that process p is suspected to have crashed.

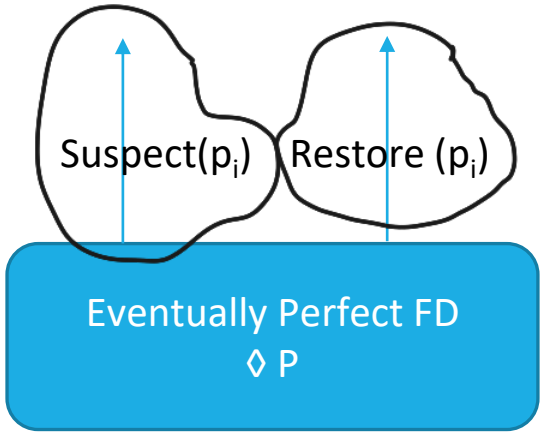
Indication: $\langle\ \diamond\mathcal{P}, \mathit{Restore} \mid p\ \rangle$: Notifies that process p is not suspected anymore.

Properties:

we don't know when

EPFD1: *Strong completeness:* Eventually, every process that crashes is permanently suspected by every correct process.

EPFD2: *Eventual strong accuracy:* Eventually, no correct process is suspected by any correct process.



Basic constructions rules of an eventually perfect FD

Use timeouts to suspect processes that did not sent expected messages

A suspect may be wrong

- A process p_i may suspect another one p_j as *the current timeout is too short*

◊P is ready to change its judgment as soon as it receives a message from p_j

- In this case, *the timeout value is updated*

If p_j has actually crashed, p_i does not change its judgment anymore

Eventually perfect failure detectors (◊P) Implementation

Algorithm 2.7: Increasing Timeout

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

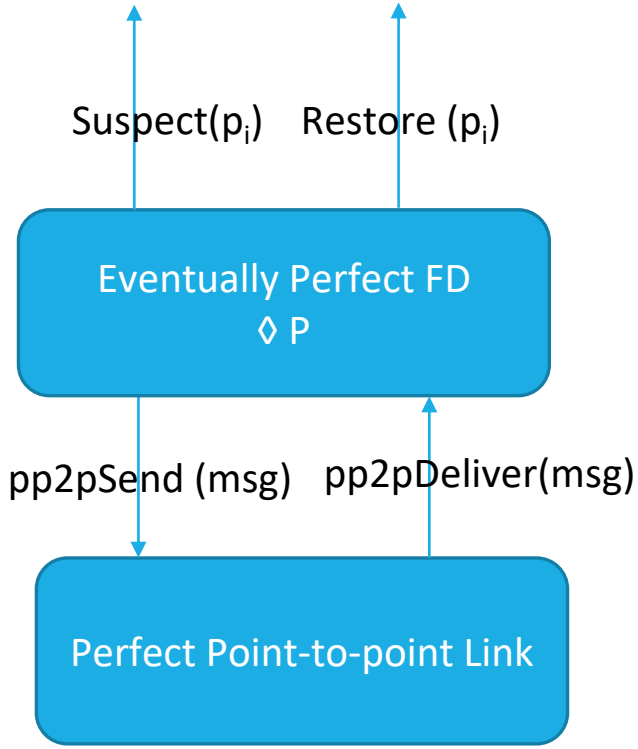
Uses:
PerfectPointToPointLinks, **instance** *pl*.

upon event $\langle\ \diamond\mathcal{P}, \mathit{Init}\ \rangle$ **do**
  $\mathit{alive} := \Pi$;
  $\mathit{suspected} := \emptyset$;
  $\mathit{delay} := \Delta$;
  starttimer(*delay*);

upon event $\langle\ \mathit{Timeout}\ \rangle$ **do**
  if $\mathit{alive} \cap \mathit{suspected} \neq \emptyset$ **then**
    $\mathit{delay} := \mathit{delay} + \Delta$;
  forall $p \in \Pi$ **do**
    if $(p \notin \mathit{alive}) \wedge (p \notin \mathit{suspected})$ **then**
      $\mathit{suspected} := \mathit{suspected} \cup \{p\}$;
      trigger $\langle\ \diamond\mathcal{P}, \mathit{Suspect} \mid p\ \rangle$;
    else if $(p \in \mathit{alive}) \wedge (p \in \mathit{suspected})$ **then**
      $\mathit{suspected} := \mathit{suspected} \setminus \{p\}$;
      trigger $\langle\ \diamond\mathcal{P}, \mathit{Restore} \mid p\ \rangle$;
    trigger $\langle\ \mathit{pl}, \mathit{Send} \mid p, [\text{HEARTBEATREQUEST}]\ \rangle$;
  alive := ∅;
  starttimer(*delay*);

upon event $\langle\ \mathit{pl}, \mathit{Deliver} \mid q, [\text{HEARTBEATREQUEST}]\ \rangle$ **do**
  trigger $\langle\ \mathit{pl}, \mathit{Send} \mid q, [\text{HEARTBEATREPLY}]\ \rangle$;

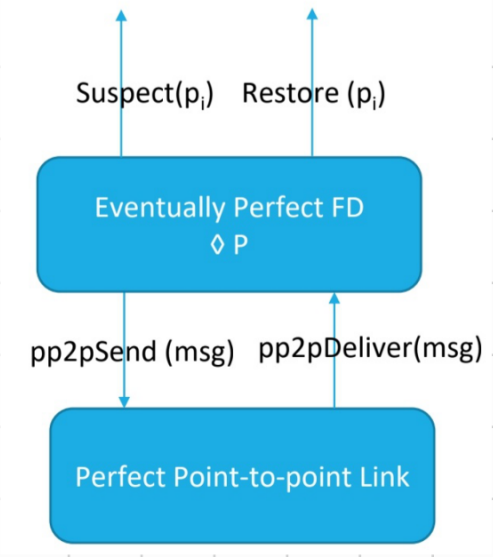
upon event $\langle\ \mathit{pl}, \mathit{Deliver} \mid p, [\text{HEARTBEATREPLY}]\ \rangle$ **do**
  $\mathit{alive} := \mathit{alive} \cup \{p\}$;



6: EVENTUALLY PFD

EPFD1: *Strong completeness:* Eventually, every process that crashes is permanently suspected by every correct process.

EPFD2: *Eventual strong accuracy:* Eventually, no correct process is suspected by any correct process.



Algorithm 2.7: Increasing Timeout

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

Uses:
PerfectPointToPointLinks, **instance** *pl*.

upon event $\langle\ \Diamond\mathcal{P}, Init\ \rangle$ **do**
  $alive := \Pi$; (all processes)
  $suspected := \emptyset$;
  $delay := \Delta$;
  starttimer($delay$);

upon event $\langle\ Timeout\ \rangle$ **do**
  if $alive \cap suspected \neq \emptyset$ **then** → to avoid the multiple SUSPECT and RESTORE ∴ make the TIMEOUT longer
    $delay := delay + \Delta$;
  forall $p \in \Pi$ **do**
    if $(p \notin alive) \wedge (p \notin suspected)$ **then** (when TIMEOUT expires)
      $suspected := suspected \cup \{p\}$;
      trigger $\langle\ \Diamond\mathcal{P}, Suspect \mid p\ \rangle$; → delay or crash?
    else if $(p \in alive) \wedge (p \in suspected)$ **then** → WHEN a PROCESS is in SUSPECTED but then REPLAY so it is ALIVE
      $suspected := suspected \setminus \{p\}$; (remove)
      trigger $\langle\ \Diamond\mathcal{P}, Restore \mid p\ \rangle$;
    trigger $\langle\ pl, Send \mid p, [\text{HEARTBEATREQUEST}]\ \rangle$; (ARE U ALIVE?)
  $alive := \emptyset$;
  starttimer($delay$);

upon event $\langle\ pl, Deliver \mid q, [\text{HEARTBEATREQUEST}]\ \rangle$ **do**
  trigger $\langle\ pl, Send \mid q, [\text{HEARTBEATREPLY}]\ \rangle$;

upon event $\langle\ pl, Deliver \mid p, [\text{HEARTBEATREPLY}]\ \rangle$ **do**
  $alive := alive \cup \{p\}$; (I'M ALIVE)

Correctness

Strong completeness

- If a process crashes, it will stop to send messages. Therefore the process will be suspected by any correct process and no process will revise the judgement.

Eventual strong accuracy

- After time T the system becomes synchronous. i.e., after that time a message sent by a correct process p to another one q will be delivered within a bounded time. If p was wrongly suspected by q, then q will revise its suspicious.

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

T. Chandra, S. Toueg *Unreliable Failure Detectors for Reliable Distributed Systems*

- https://dl.acm.org/doi/pdf/10.1145/226643.226647