Unreliable Failure Detectors for Reliable Distributed Systems

Chandra & Toueg (1996)
Journal of the ACM
Vol. 43, No. 2, pp. 225-267

The Consensus Problem

A set of processes must reach a common decision, which depends on their initial inputs, despite of failures

All correct processes propose a value and must reach a unanimous and irrevocable decision on some value that is related to the proposed values

Impossibility Result

FLP: Consensus cannot be solved deterministically in asynchronous systems that are subject to even a single process crash (unannounced death)

Idea

The FLP result assumes that, in an asynchronous environment, crashes of processes cannot be detected reliably (crash or very slow?)

- To stop waiting or not to stop waiting, that is the question

- Do the same as ancient Greeks did: ask an oracle

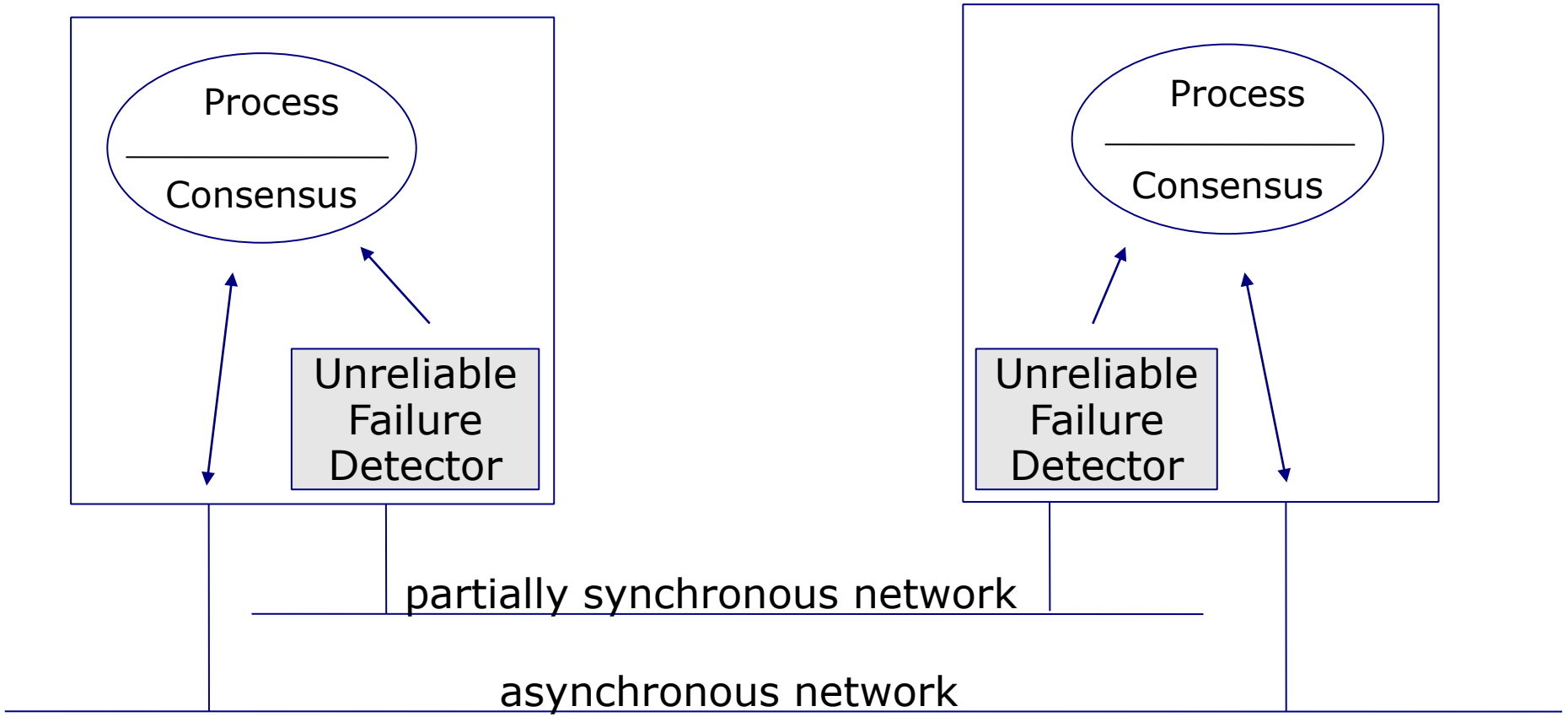
Idea

Separate the consensus problem into

- The consensus algorithm itself
- A “failure detector” that announces suspected failures but that can make mistakes

Unreliable Failure Detector

A distributed oracle that outputs (possibly incorrect) information about the operational status of other processes



Unreliable Failure Detector

Outputs (hints) may be incorrect:

- FD may give different hints to different processes
- FD may change its mind (over and over) about the operational status of a process

Unreliable Failure Detector

When the underlying system behaves well (synchronously) during a “sufficiently long” enough period, the outputs of the failure detector are no longer approximate but become correct and the consensus algorithm can terminate

Failure detectors are required in distributed systems to maintain liveness in spite of process crashes

Possibility Result

Chandra & Toueg*:* Consensus can be solved in an asynchronous system subject to crash failures with an *unreliable failure detector*

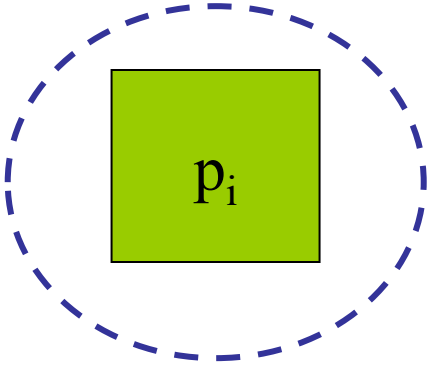
Obviously, a failure detector cannot be implemented in an asynchronous system!

But it can be implemented in a partially synchronous system (using timeouts in some way)

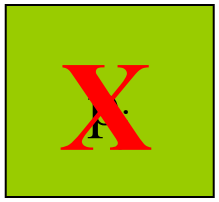
Partial synchrony assumptions can be encapsulated in the unreliability of failure detectors

What's a Failure Detector?

needs to know about p_j's failure

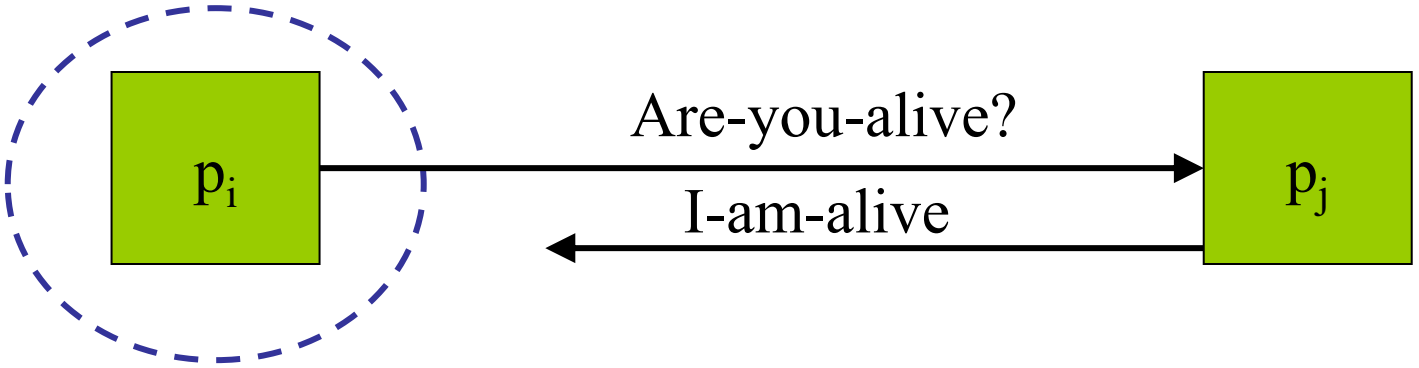


Crash failure



Ping-Ack Protocol

needs to know about p_j's failure



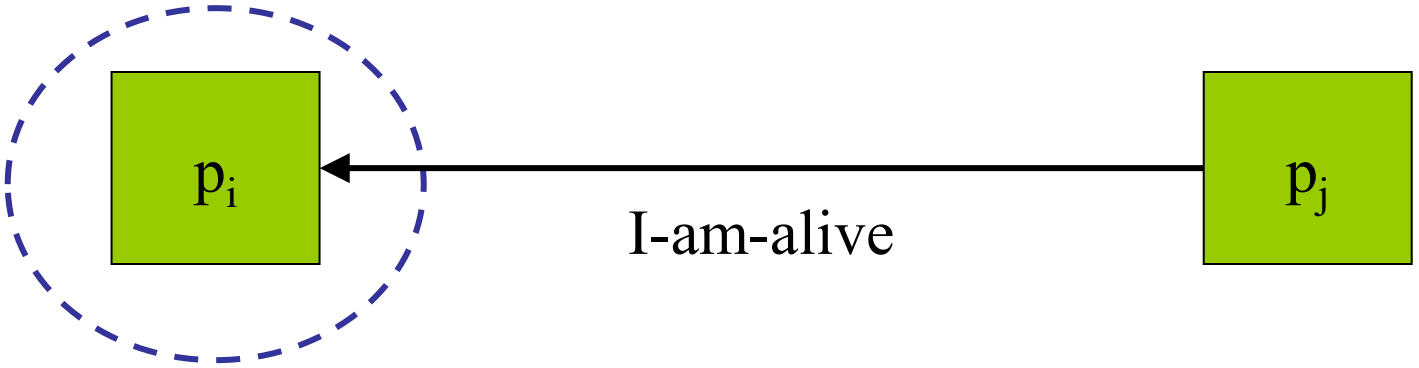
p_i queries p_j once every T time units

p_j replies

if p_j does not respond within T time units,
p_i adds p_j to its list of suspects

Heartbeating Protocol

needs to know about p_j's failure



p_j sends p_i a heartbeat with incremented sequence number after every T time units

if p_i has not received a new heartbeat for the past T time units, p_i adds p_j to its list of suspects

Implementation of an Adaptive Failure Detector

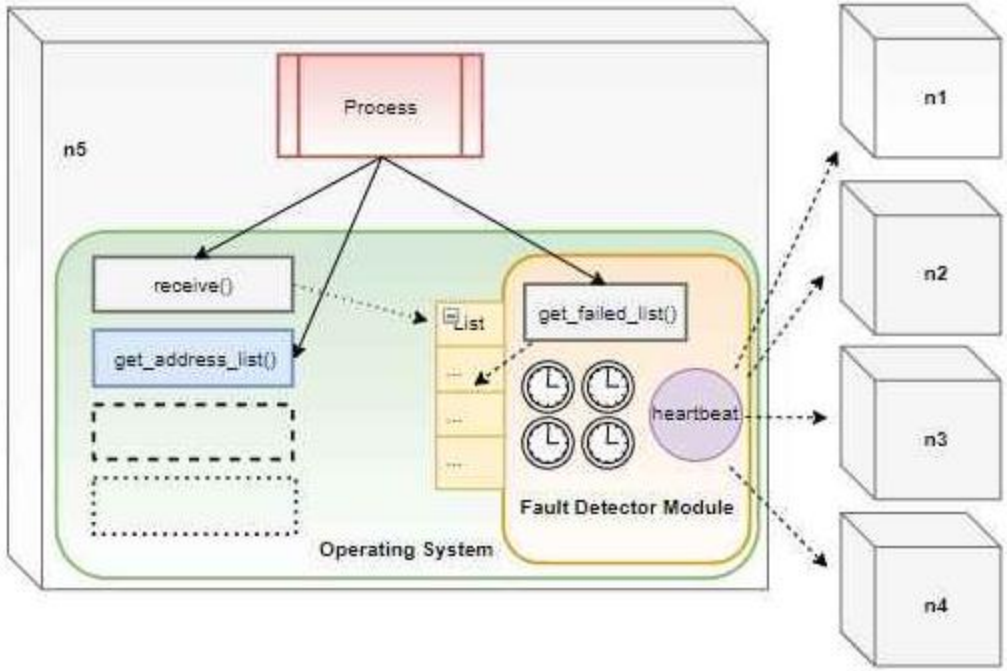
Every process q periodically sends a “q-is-alive” message to all the processes

If a process p **times-out** on some process q, it adds q to its list of suspects

If p later receives a "q-is-alive" message, p recognizes that it made a mistake by prematurely timing out on q: p removes q from its list of suspects, and increases the length of its timeout period in an attempt to prevent a similar mistake in the future

Eventually accurate

Implementation of an Adaptive Failure Detector



Augmenting the system with an oracle

Unreliable Failure Detectors

Characterized in terms of two abstract properties:

- **Completeness**: degree to which failed processes are suspected
- **Accuracy**: degree to which correct processes are not suspected

Independent of specific implementations

Completeness

Eventually every process failure is suspected (no misses)

- Strong completeness: Eventually, every process that crashes is permanently suspected by ***every*** correct process
- Weak completeness: Eventually, every process that crashes is permanently suspected by ***some*** correct process

Accuracy

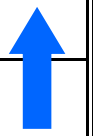
Every suspected failure corresponds to a crashed process (no mistakes)

- Strong accuracy: ***Every*** correct process is never suspected
- Weak accuracy: ***Some*** correct process is never suspected

- Eventual strong accuracy: there is a time after which ***every*** correct process is never suspected
- Eventual weak accuracy: there is a time after which ***some*** correct process is never suspected

Classes of Failure Detectors

Completeness	Accuracy			
	Strong	Weak	Eventual strong	Eventual weak
Strong	*Perfect* P	*Strong* S	*Eventually Perfect* $\Diamond P$	*Eventually Strong* $\Diamond S$
Weak	*Quasi-Perfect* Q	*Weak* W	*Eventually Quasi-Perfect* $\Diamond Q$	*Eventually Weak* $\Diamond W$



× **Weak Completeness Reduction (equivalence)**

Synchrony assumptions are stronger

Weak Completeness Reduction

Simple idea for boost completeness:

- Every node q broadcast suspicions set **Susp** periodically
  // also works like a heartbeat
- Upon event receive <S,q> **Susp** := (**Susp** ∪ S) - {q}

Every crash is eventually detected by every correct p

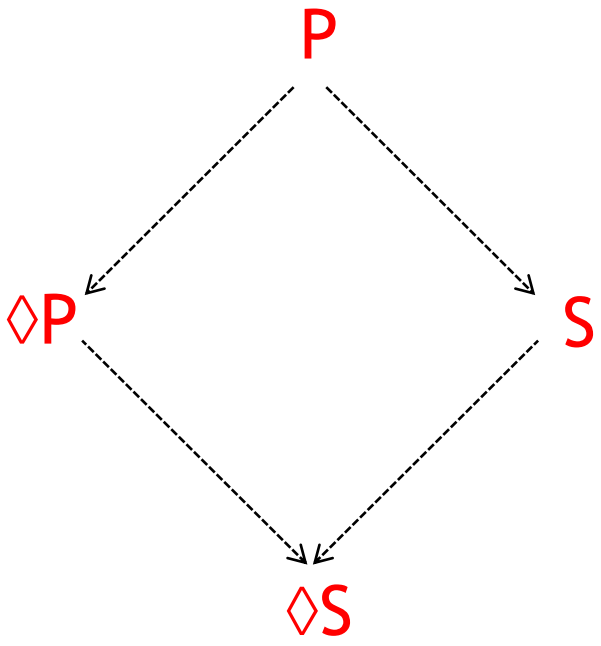
Comparing Failure Detectors by Reducibility

◊P≼P
P is always strongly accurate,
thus, also eventually strongly accurate

◊S≼S
S is always weakly accurate,
thus, also eventually weakly accurate

S≼P
P is always strongly accurate,
thus, also always weakly accurate

◊S≼◊P
◊P is always eventually strongly accurate,
thus, also always eventually weakly accurate



C → C':
C' is strictly weaker than C

Consensus Problem

The problem is specified as follows:

- *Agreement*
  - no two correct processes decide differently
- *Uniform validity*
  - if a process decides v, then v was proposed by some process
- *Termination*
  - every correct process eventually decides some value

Solving Consensus

In synchronous systems:

- Possible

In asynchronous systems:

- Impossible, even if all links are reliable and at most one process may crash

Why this difference

In synchronous systems:

- Use timeouts to determine with certainty whether a process has crashed
- Perfect failure detector

In asynchronous systems:

- Cannot determine with certainty whether a process has crashed or not (it may be slow, or its messages are delayed)
- No failure detector

System Model

- Partially synchronous distributed system
- Finite set of processes $\Pi = \{p_1, p_2, \ldots, p_n\}$
- Reliable channels connecting every pair of processes (fully connected system)
- Crash failure model (halt, no recovery)
- A process is correct if it never crashes

Failure Detectors and Consensus Algorithms

Is perfect failure detection necessary for consensus?

No

The consensus algorithm that uses *S* tolerates any number of failures

The consensus algorithm that uses ◊*S* requires a majority of correct processes (n > 2t)

Eventually Strong Failure Detector

The detector ◊S: “Eventually strong”

Strong completeness and eventual weak accuracy

Defined by two properties:

- Eventually every process that crashes is permanently suspected by **every** correct process
- There is a time after which **some** correct process is never suspected by any correct process

Eventually Weak Failure Detector

The detector $\Diamond\mathcal{W}$: “Eventually weak”

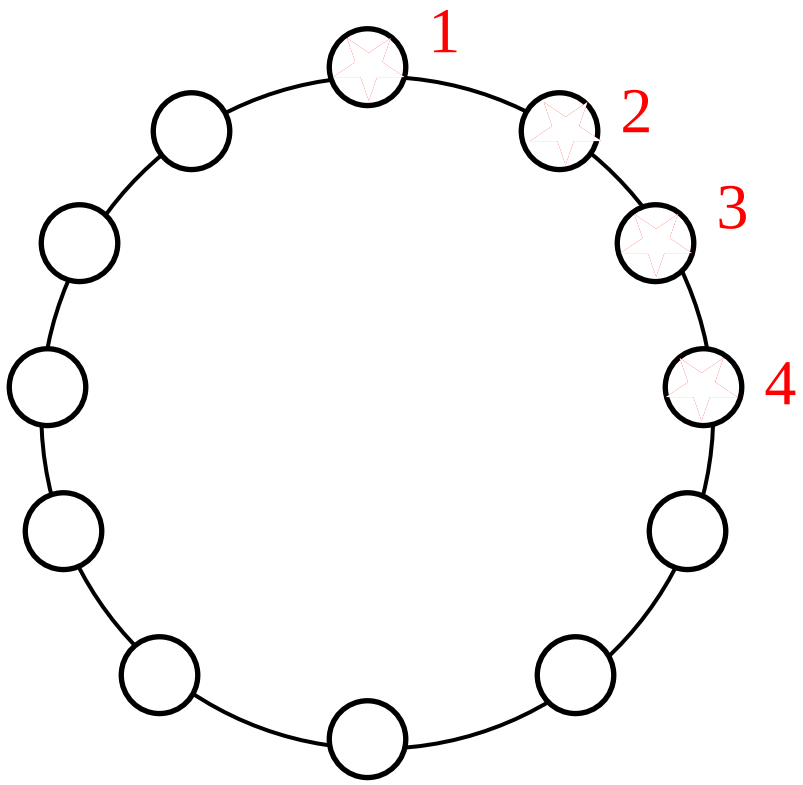
Weak completeness and eventual weak accuracy

Defined by two properties:

- Eventually every process that crashes is permanently suspected by **some** correct process
- There is a time after which **some** correct process is never suspected by any correct process

Consensus Algorithm using ◊S

The protocol proceeds in asynchronous rounds with a *rotating coordinator* p_i in each round r, until a decision is reached



i = (r mod n) + 1

Consensus Algorithm

In round r, the coordinator:

- tries to impose its estimate as the consensus value
- succeeds if it does not crash and it is not suspected by the failure detector

Consensus Algorithm – Variables

Each process keeps track of

- its current preferred decision value (initially equal to the input of the process) and
- the last round where it changed its preferred decision value (the value's timestamp)

Consensus Algorithm – Epochs

1. The coordinator determines a consistent decision value from several proposed values

2. A value gets locked: no other decision value is possible
3. The processes decide the locked value and consensus is reached

Consensus Algorithm – Actions in each round

1.a) All processes send *vote*(r, preference, timestamp) to the current coordinator.

1.b) The coordinator waits to receive messages from at least half of the processes. It then chooses as its preference a value with the most recent timestamp among those sent. The coordinator then broadcasts *value*(r, preference).

Consensus Algorithm – Actions in each round

1.c) Each process waits to receive *value*(r, preference) from the coordinator **or** for its failure detector to identify the coordinator as crashed.

- In the first case, it sets its own preference to the coordinator's preference and responds with *ack*(r).
- In the second case, it responds with *nack*(r).

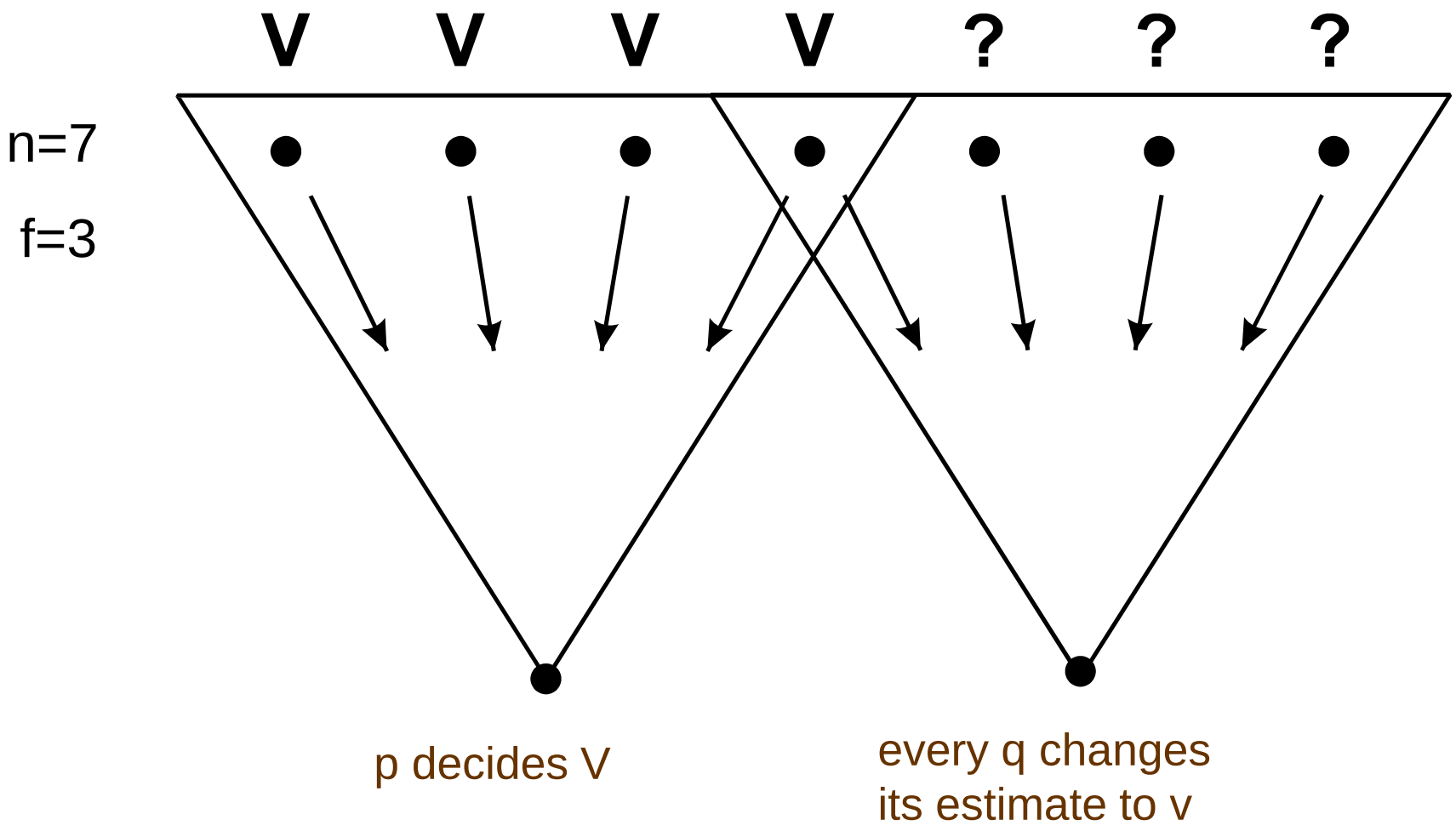
1.d) The coordinator waits to receive replies from at least half of the processes. If the coordinator does not receive *ack*(r) from a majority of processes **goto 1a (of next round)**.

Consensus Algorithm – Actions in each round

2. The coordinator decides for preference, broadcasts *decide*(preference) and terminates

3. Any process that receives *decide*(preference) for the first time decides for preference and terminates

Consensus Algorithm – Agreement



Consensus Algorithm – Termination

- No process blocks forever waiting for a message from a dead coordinator

  (strong completeness)

- Eventually some correct process c is not falsely suspected. When c becomes the coordinator, every process receives its estimate and decides

  (eventually weak accuracy)

Failure detector is required for liveness

Eventually Weak Accuracy

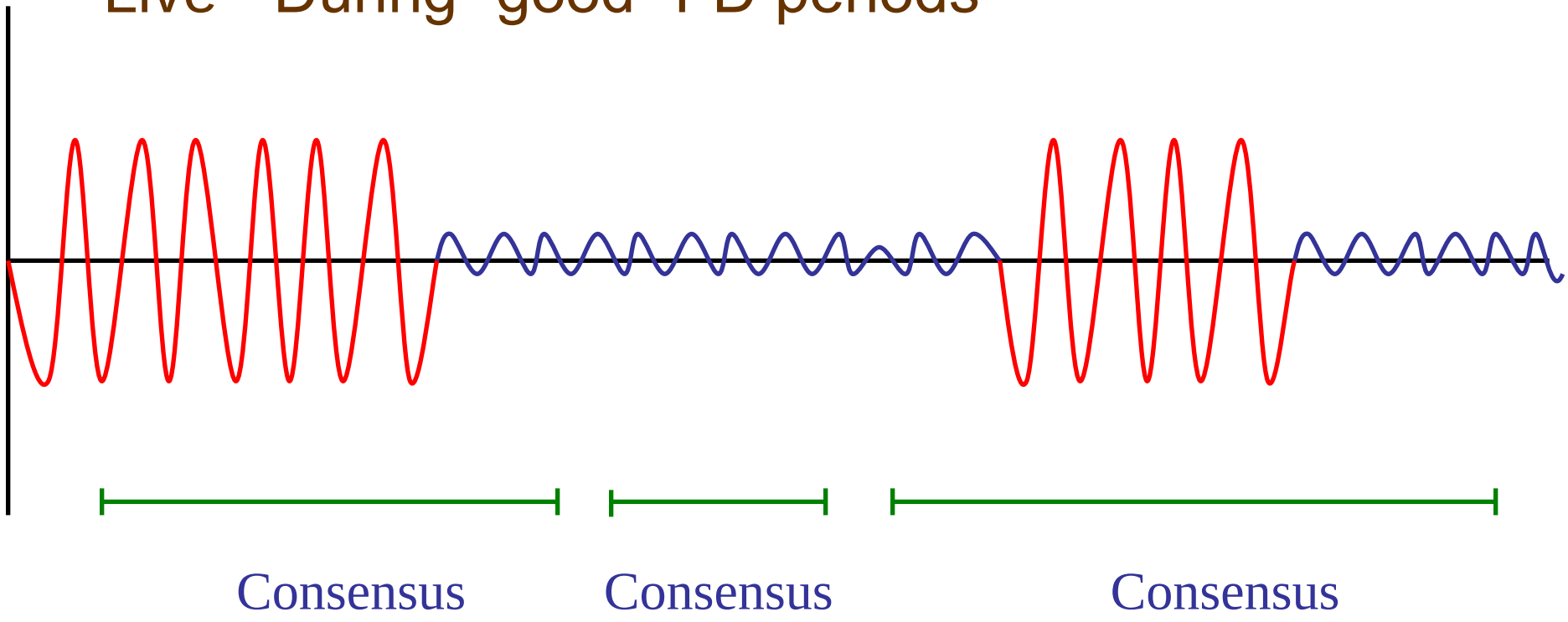
In practice, it is not really required that the eventual weak accuracy property holds forever

When solving a problem that "terminates", such as consensus, it is enough that this property (there are no premature timeouts on at least one correct process) holds for a "sufficiently long" period of time for the algorithm to achieve its goal (e.g., for correct processes to decide)

Eventually Weak Accuracy

Consensus algorithm is:

- Safe - Always!
- Live - During “good” FD periods



Another Consensus Algorithm using ◊S

every process p sets estimate to its initial value

for rounds r := 0, 1, 2 ... do {round r msgs are tagged with r}

the coordinator c of round r sends its estimate v to all

every p waits until (a) it receives v from c or

(b) it suspects c (according to ◊ S)

if (a) then send v to all

if (b) then send ? to all

every p waits until it receives a msg v or ? from n-f processes

if it received at least (n+1)/2 msgs v then decide v

if it received at least one msg v then estimate := v

if it received only ? msgs then do nothing

Leader Election

Formally, leader election is an FD

Always suspects all nodes except one (leader)

Module:

Name: EventualLeaderDetector (Ω).

Events:

Indication: $\langle$ *trust* | p_i $\rangle$: Used to notify that process p_i 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.

Leader Election Algorithm Ω

A simple implementation of Ω consists of implementing ◊S first and outputting the smallest process currently not suspected by the failure detector

Most practical consensus protocols today use such a service as an important component of their system - the leader is often referred to as the “master” or the “primary”

Failure Detection Abstraction

Some advantages

- Determines the minimal information necessary about failures to solve consensus
- Suggests why consensus is not so difficult in practice
- Increases the modularity and portability of algorithms

Timing assumptions are hidden in the failure detector

Problem Solvability in Distributed Computing Models