

Failure Detectors: definition, algorithms, and applications

Alain Cournier Stéphane Devismes

Université de Picardie Jules Verne

May 18, 2026



Roadmap

- 1 Introduction
- 2 Definition
- 3 Application: a Consensus Algorithm
 - The Problem
 - The Model
 - The Algorithm
 - The Proof
- 4 Implementation of a Failure Detector (an example)
 - $\diamond\mathcal{P}$
 - The Model
 - The Algorithm
 - The Proof
- 5 References

Roadmap

- 1 Introduction
- 2 Definition
- 3 Application: a Consensus Algorithm
 - The Problem
 - The Model
 - The Algorithm
 - The Proof
- 4 Implementation of a Failure Detector (an example)
 - $\diamond P$
 - The Model
 - The Algorithm
 - The Proof
- 5 References

Outline of the FLP's Impossibility [3]

The impossibility of the consensus comes from the fact that correct processes **cannot differentiate slow processes from crashed ones.**

Outline of the FLP's Impossibility [3]

The impossibility of the consensus comes from the fact that correct processes **cannot differentiate slow processes from crashed ones**.

Using synchrony assumptions, processes can obtain **information about crashes**.

Then, this information enables solving problems, including consensus.

The Failure Detector Approach [1]

In a software engineering spirit:

- separate the **necessary knowledge on crashes** to **solve the problem** (the definition of the failure detector)
- from **the way it can be obtained**¹ (the implementation of the failure detector)

¹In particular, the necessary assumptions on the system.

The Failure Detector Approach [1]

In a software engineering spirit:

- separate the **necessary knowledge on crashes** to **solve the problem** (the definition of the failure detector)
- from **the way it can be obtained**¹ (the implementation of the failure detector)

Advantages

- **Separation of concerns**: modularity and simplicity
- Possibility to **compare** and to have a **necessary and sufficient assumption** (the **minimum failure detector** to solve a problem)²

¹ In particular, the necessary assumptions on the system.

² A lesson we be dedicated to this aspect.

Roadmap

- 1 Introduction
- 2 **Definition**
- 3 Application: a Consensus Algorithm
 - The Problem
 - The Model
 - The Algorithm
 - The Proof
- 4 Implementation of a Failure Detector (an example)
 - $\diamond\mathcal{P}$
 - The Model
 - The Algorithm
 - The Proof
- 5 References

(Distributed) Failure Detector: an Oracle

Each process p can access a **local failure detector module** (an oracle function) denoted by \mathcal{D}_p .

(Distributed) Failure Detector: an Oracle

Each process p can access a **local failure detector module** (an oracle function) denoted by \mathcal{D}_p .

Each module watches a subset of system processes (usually the whole set of processes), and returns information about crashed: usually **a set of suspected processes**.³

Precisely, **the identifiers of processes that are suspected of being crashed**.

³*N.b.*, some failure detectors, such as Ω or Σ , do not return a list of suspected processes.

(Distributed) Failure Detector: an Oracle

Each process p can access a **local failure detector module** (an oracle function) denoted by \mathcal{D}_p .

Each module watches a subset of system processes (usually the whole set of processes), and returns information about crashed: usually **a set of suspected processes**.³

Precisely, **the identifiers of processes that are suspected of being crashed**.

Unless otherwise mentioned, we will always assume that

each local failure detector module watches all processes and returns a list of suspected processes.

³*N.b.*, some failure detectors, such as Ω or Σ , do not return a list of suspected processes.

Unreliable Failure Detectors

Each local module **can make mistakes**:

- by **missing** some crashed processes
- by **wrongly suspecting** correct processes

Failure Detector Classes

The **classes of failure detectors** are distinguished by two important properties:

Completeness: restrict the ability of the failure detector module to detect crashes

Accuracy: qualify the possibility of the failure detector module to wrongly suspect correct processes

Completeness: Examples

Strong Completeness: Every faulty process is eventually permanently suspected by **every** correct process.

Completeness: Examples

Strong Completeness: Every faulty process is eventually permanently suspected by **every** correct process.

Weak Completeness: If there is a correct process, then every faulty process is eventually permanently suspected by **some** correct process.

Completeness: Examples

Strong Completeness: Every faulty process is eventually permanently suspected by **every** correct process.

Weak Completeness: If there is a correct process, then every faulty process is eventually permanently suspected by **some** correct process.

Remark: In the original paper [1], it is assumed that at least one process is correct. Under this assumption, the definition of **Weak Completeness** is simplified:

Every faulty process is eventually permanently suspected by at least one correct process.

Accuracy: Examples

Strong Accuracy: No process is suspected (by any alive process⁴) before it crashes.

⁴An alive process is either a correct process or a faulty process that has not crashed yet.

Accuracy: Examples

Strong Accuracy: **No** process is suspected (by any alive process⁴) before it crashes.

Weak Accuracy: **Some** correct process is never suspected (by any alive process).⁵

⁴An alive process is either a correct process or a faulty process that has not crashed yet.

⁵This definition implicitly assumes the existence of at least one correct process.

Accuracy: Examples

Strong Accuracy: **No** process is suspected (by any alive process⁴) before it crashes.

Weak Accuracy: **Some** correct process is never suspected (by any alive process).⁵

Eventual Strong Accuracy: **There is a time after which no** correct process is suspected by any correct process.⁶

⁴An alive process is either a correct process or a faulty process that has not crashed yet.

⁵This definition implicitly assumes the existence of at least one correct process.

⁶As explained in [1], we can use correct instead of alive with loss of generality for eventual strong/weak accuracy.

Accuracy: Examples

Strong Accuracy: **No** process is suspected (by any alive process⁴) before it crashes.

Weak Accuracy: **Some** correct process is never suspected (by any alive process).⁵

Eventual Strong Accuracy: **There is a time after which no** correct process is suspected by any correct process.⁶

Eventual Weak Accuracy: **There is a time after which some** correct process is never suspected by any correct process.

⁴An alive process is either a correct process or a faulty process that has not crashed yet.

⁵This definition implicitly assumes the existence of at least one correct process.

⁶As explained in [1], we can use correct instead of alive with loss of generality for eventual strong/weak accuracy.

Some Classes of Failure Detectors

Completeness	Accuracy			
	Strong	Weak	Eventually Strong	Eventually Weak
Strong	Perfect \mathcal{P}	Strong \mathcal{S}	Eventually Perfect $\diamond\mathcal{P}$	Eventually Strong $\diamond\mathcal{S}$
Weak	Quasi-perfect \mathcal{Q}	Weak \mathcal{W}	Eventually Quasi-perfect $\diamond\mathcal{Q}$	Eventually Weak $\diamond\mathcal{W}$

Implementing Failure Detectors in Partial Synchronous Systems

Some examples

Using Ping or Heartbeat

Implementing Failure Detectors in Partial Synchronous Systems

Some examples

Using Ping or Heartbeat

\mathcal{P} : In a Fully Synchronous System

If no news from p within the timeout, it is crashed!

Implementing Failure Detectors in Partial Synchronous Systems

Some examples

Using Ping or Heartbeat

\mathcal{P} : In a Fully Synchronous System

If no news from p within the timeout, it is crashed!

But, also in a system where: (Correct \Rightarrow Synchronous) + Bi-source

(use the bi-source as a relay to provide news within bounded time)

Implementing Failure Detectors in Partial Synchronous Systems

Some examples

Using Ping or Heartbeat

\mathcal{P} : In a Fully Synchronous System

If no news from p within the timeout, it is crashed!

But, also in a system where: (Correct \Rightarrow Synchronous) + Bi-source
(use the bi-source as a relay to provide news within bounded time)

\mathcal{S} : (Correct \Rightarrow Synchronous) + Source

The heartbeat of the source is always on time!

Implementing Failure Detectors in Partial Synchronous Systems

Some examples

Using Ping or Heartbeat

\mathcal{P} : In a Fully Synchronous System

If no news from p within the timeout, it is crashed!

But, also in a system where: (Correct \Rightarrow Synchronous) + Bi-source

(use the bi-source as a relay to provide news within bounded time)

\mathcal{S} : (Correct \Rightarrow Synchronous) + Source

The heartbeat of the source is always on time!

$\diamond\mathcal{P}$: (Correct \Rightarrow Eventually Synchronous) + Eventual Bi-source

See later

Implementing Failure Detectors in Partial Synchronous Systems

Some examples

Using Ping or Heartbeat

\mathcal{P} : In a Fully Synchronous System

If no news from p within the timeout, it is crashed!

But, also in a system where: (Correct \Rightarrow Synchronous) + Bi-source
(use the bi-source as a relay to provide news within bounded time)

\mathcal{S} : (Correct \Rightarrow Synchronous) + Source

The heartbeat of the source is always on time!

$\diamond\mathcal{P}$: (Correct \Rightarrow Eventually Synchronous) + Eventual Bi-source

See later

$\diamond\mathcal{S}$: (Correct \Rightarrow Eventually Synchronous) + Eventual source

Messages from the eventual source will eventually always arrived in time

Roadmap

- 1 Introduction
- 2 Definition
- 3 Application: a Consensus Algorithm**
 - The Problem
 - The Model
 - The Algorithm
 - The Proof
- 4 Implementation of a Failure Detector (an example)
 - $\diamond P$
 - The Model
 - The Algorithm
 - The Proof
- 5 References

Consensus Specification (recall)

For every process p

Input : $v_p \in \{0, 1\}$

Output : $d_p \in \{\perp, 0, 1\}$ initialized to \perp

Requirements:

Integrity Every process decides, *i.e.*, assigns its d -variable to a non- \perp value, at most once

Termination : Every correct process eventually decides

Validity : Every decided value is an initially proposed value, *i.e.*,
 $\forall p \in V, d_p \neq \perp \Rightarrow d_p \in \{v_q : q \in V\}$

(Uniform) Agreement : If two processes p and q decide, then they decide the same value, *i.e.*, $d_p = d_q$

Failure Detector \mathcal{S}

We now study a **consensus algorithm**⁷ that uses \mathcal{S} :

Strong Completeness: Every faulty process is eventually permanently suspected by **every** correct process.

Weak accuracy: **Some** correct process is never suspected (by any alive process).

⁷This algorithm is inspired for one of Chandra and Toueg [2].

Assumptions

- 1 **Failure detector \mathcal{S}** : local module \mathcal{S}_p for each process p
- 2 Asynchronous processes
- 3 Asynchronous **reliable links** (not necessarily FIFO)
- 4 Any process p can broadcast a message to all processes (p included!)
- 5 **No additional assumption on the number of crashes!**
(Recall that \mathcal{S} implies the existence of at least one correct process.)

Principles

3 Phases:

Phase 1: $n - 1$ asynchronous rounds where **proposed values** are broadcast and relayed.

Phase 2: 1 asynchronous round where all alive processes **agree on the value of a vector Val** based on proposed values.

Phase 3: each alive process **decides according to the vector Val** .

Constants & Variables

- Processes are identified: a process and its identifier are used equivalently
- V : set of processes
- n : number of processes
- v_p : a boolean (read-only) input, the value proposed by process p
- $d_p \in \{\perp, 0, 1\}$: the decision variable
- $r_p \in \mathbb{N}$: the round number at process p
- $\Delta_p[], Val_p[]$: vectors indexed on the process IDs
 $\rightarrow \forall q \in V, Val_p[q] \in \{0, 1, \perp\}$ and $\Delta_p[q] \in \{0, 1, \perp\}$

Algorithm of Chandra and Toueg

```

1:  $d_p \leftarrow \perp$ 
2:  $Val_p \leftarrow [\perp, \dots, \perp]$ 
3:  $Val_p[p] \leftarrow v_p$ 
4:  $\Delta_p \leftarrow Val_p$ 
5: For all  $r_p$  from 1 to  $n-1$  do /* Phase 1 */
6:   broadcast  $\langle r_p, \Delta_p, p \rangle$  to  $V$  ( $p$  included)
7:   wait to receive  $\langle r_p, \Delta_q, q \rangle$  from  $q \in V \setminus S_p$ 
8:   let  $R_p[r_p]$  be the set of received  $\langle r_p, \Delta_q, q \rangle$ 
9:    $\Delta_p \leftarrow [\perp, \dots, \perp]$ 
10:  For all  $k \in V$  do
11:    if  $Val_p[k] = \perp \wedge (\exists (r_p, \Delta_q, q) \in R_p[r_p], \Delta_q[k] \neq \perp)$  then
12:       $Val_p[k] \leftarrow \Delta_q[k]$ 
13:       $\Delta_p[k] \leftarrow \Delta_q[k]$ 
14:    End if
15:  Done
16: Done
17: broadcast  $\langle Val_p \rangle$  to  $V$  ( $p$  included) /* Phase 2 */
18: wait to receive  $\langle Val_q \rangle$  from every process  $q \in V \setminus S_p$ 
19: Let  $lastmsgs_p$  be the set of received  $\langle Val_q \rangle$ 
20: For all  $k \in V$  do
21:   if  $\exists Val_q \in lastmsgs_p, Val_q[k] = \perp$  then  $Val_p[k] \leftarrow \perp$ 
22: Done
23:  $d_p \leftarrow x$  where  $x$  is the first non- $\perp$  value in  $Val_p$  /* Phase 3 */

```

Validity

The following remark can be shown by induction on the number of rounds.

Remark 1

At the end of Round $r \geq 1$, $\Delta_p[q] \neq \perp$ IFF $p \neq q$, $\Delta_p[q]$ is the value proposed by q , and p has received this value for the first time during Round r .

From Remark 1 and owing the fact that $Val_p[p] = v_p$ is the unique non- \perp cell in Val_p right after the initialization (Lines 1-4), we can deduce the following lemma by definition of the algorithm:

Lemma 1

For every two processes p and q , after the initialization (Lines 1-4) and while p is not crashed, $Val_p[q]$ is either v_q or \perp .

```

1:  $d_p \leftarrow \perp$ 
2:  $Val_p \leftarrow [\perp, \dots, \perp]$ 
3:  $Val_p[p] \leftarrow v_p$ 
4:  $\Delta_p \leftarrow Val_p$ 
5: For all  $r_p$  from 1 to  $n-1$  do                                /* Phase 1 */
6:   broadcast  $\langle r_p, \Delta_p, p \rangle$  to  $V$  ( $p$  included)
7:   wait to receive  $\langle r_p, \Delta_q, q \rangle$  from  $q \in V \setminus S_p$ 
8:   let  $R_p[r_p]$  be the set of received  $\langle r_p, \Delta_q, q \rangle$ 
9:    $\Delta_p \leftarrow [\perp, \dots, \perp]$ 
10:  For all  $k \in V$  do
11:    If  $Val_p[k] = \perp \wedge (\exists (r_p, \Delta_q, q) \in R_p[r_p], \Delta_q[k] \neq \perp)$  then
12:       $Val_p[k] \leftarrow \Delta_q[k]$ 
13:       $\Delta_p[k] \leftarrow \Delta_q[k]$ 
14:    End If
15:  Done
16: Done
17: broadcast  $\langle Val_p \rangle$  to  $V$  ( $p$  included)                        /* Phase 2 */
18: wait to receive  $\langle Val_q \rangle$  from  $q \in V \setminus S_p$ 
19: Let  $lastmsgs_p$  be the set of received  $\langle Val_q \rangle$ 
20: For all  $k \in V$  do
21:   If  $\exists Val_q \in lastmsgs_p, Val_q[k] = \perp$  then  $Val_p[k] \leftarrow \perp$ 
22: Done
23:  $d_p \leftarrow x$  where  $x$  is the first non- $\perp$  value in  $Val_p$     /* Phase 3 */

```

Termination

Lemma 2

Every correct process eventually executes Line 23.

Proof.

```

1:  $d_p \leftarrow \perp$ 
2:  $Val_p \leftarrow [\perp, \dots, \perp]$ 
3:  $Val_p[p] \leftarrow v_p$ 
4:  $\Delta_p \leftarrow Val_p$ 
5: For all  $r_p$  from 1 to  $n-1$  do                                /* Phase 1 */
6:   broadcast  $\langle r_p, \Delta_p, p \rangle$  to  $V$  ( $p$  included)
7:   wait to receive  $\langle r_p, \Delta_q, q \rangle$  from  $q \in V \setminus S_p$ 
8:   let  $R_p[r_p]$  be the set of received  $\langle r_p, \Delta_q, q \rangle$ 
9:    $\Delta_p \leftarrow [\perp, \dots, \perp]$ 
10:  For all  $k \in V$  do
11:    If  $Val_p[k] = \perp \wedge (\exists (r_p, \Delta_q, q) \in R_p[r_p], \Delta_q[k] \neq \perp)$  then
12:       $Val_p[k] \leftarrow \Delta_q[k]$ 
13:       $\Delta_p[k] \leftarrow \Delta_q[k]$ 
14:    End If
15:  Done
16: Done
17: broadcast  $\langle Val_p \rangle$  to  $V$  ( $p$  included)                        /* Phase 2 */
18: wait to receive  $\langle Val_q \rangle$  from  $q \in V \setminus S_p$ 
19: Let  $lastmsgs_p$  be the set of received  $\langle Val_q \rangle$ 
20: For all  $k \in V$  do
21:   If  $\exists Val_q \in lastmsgs_p, Val_q[k] = \perp$  then  $Val_p[k] \leftarrow \perp$ 
22: Done
23:  $d_p \leftarrow x$  where  $x$  is the first non- $\perp$  value in  $Val_p$     /* Phase 3 */

```

Termination

Lemma 2

Every correct process eventually executes Line 23.

Proof.

The only way to prevent a correct process p from reaching Line 23 is to block it forever on **Line 7 or 18** in some round r .

```

1:  $d_p \leftarrow \perp$ 
2:  $Val_p \leftarrow [\perp, \dots, \perp]$ 
3:  $Val_p[p] \leftarrow v_p$ 
4:  $\Delta_p \leftarrow Val_p$ 
5: For all  $r_p$  from 1 to  $n-1$  do                                /* Phase 1 */
6:   broadcast  $\langle r_p, \Delta_p, p \rangle$  to  $V$  ( $p$  included)
7:   wait to receive  $\langle r_p, \Delta_q, q \rangle$  from  $q \in V \setminus S_p$ 
8:   let  $R_p[r_p]$  be the set of received  $\langle r_p, \Delta_q, q \rangle$ 
9:    $\Delta_p \leftarrow [\perp, \dots, \perp]$ 
10:  For all  $k \in V$  do
11:    If  $Val_p[k] = \perp \wedge (\exists (r_p, \Delta_q, q) \in R_p[r_p], \Delta_q[k] \neq \perp)$  then
12:       $Val_p[k] \leftarrow \Delta_q[k]$ 
13:       $\Delta_p[k] \leftarrow \Delta_q[k]$ 
14:    End If
15:  Done
16: Done
17: broadcast  $\langle Val_p \rangle$  to  $V$  ( $p$  included)                        /* Phase 2 */
18: wait to receive  $\langle Val_q \rangle$  from  $q \in V \setminus S_p$ 
19: Let  $lastmsgs_p$  be the set of received  $\langle Val_q \rangle$ 
20: For all  $k \in V$  do
21:   If  $\exists Val_q \in lastmsgs_p, Val_q[k] = \perp$  then  $Val_p[k] \leftarrow \perp$ 
22: Done
23:  $d_p \leftarrow x$  where  $x$  is the first non- $\perp$  value in  $Val_p$     /* Phase 3 */

```

Termination

Lemma 2

Every correct process eventually executes Line 23.

Proof.

The only way to prevent a correct process p from reaching Line 23 is to block it forever on **Line 7 or 18** in some round r .

Let $Faulty \subseteq V$ be the set of faulty processes. Let $Correct \subseteq V$ be the set of correct processes.

By definition, $V = Faulty \dot{\cup} Correct$.

By strong completeness, **eventually**
 $V \setminus S_p \subseteq Correct$ forever

```

1:  $d_p \leftarrow \perp$ 
2:  $Val_p \leftarrow [\perp, \dots, \perp]$ 
3:  $Val_p[p] \leftarrow v_p$ 
4:  $\Delta_p \leftarrow Val_p$ 
5: For all  $r_p$  from 1 to  $n-1$  do                                /* Phase 1 */
6:   broadcast  $\langle r_p, \Delta_p, p \rangle$  to  $V$  ( $p$  included)
7:   wait to receive  $\langle r_p, \Delta_q, q \rangle$  from  $q \in V \setminus S_p$ 
8:   let  $R_p[r_p]$  be the set of received  $\langle r_p, \Delta_q, q \rangle$ 
9:    $\Delta_p \leftarrow [\perp, \dots, \perp]$ 
10:  For all  $k \in V$  do
11:    If  $Val_p[k] = \perp \wedge (\exists (r_p, \Delta_q, q) \in R_p[r_p], \Delta_q[k] \neq \perp)$  then
12:       $Val_p[k] \leftarrow \Delta_q[k]$ 
13:       $\Delta_p[k] \leftarrow \Delta_q[k]$ 
14:    End If
15:  Done
16: Done
17: broadcast  $\langle Val_p \rangle$  to  $V$  ( $p$  included)                        /* Phase 2 */
18: wait to receive  $\langle Val_q \rangle$  from  $q \in V \setminus S_p$ 
19: Let  $lastmsgs_p$  be the set of received  $\langle Val_q \rangle$ 
20: For all  $k \in V$  do
21:   If  $\exists Val_q \in lastmsgs_p, Val_q[k] = \perp$  then  $Val_p[k] \leftarrow \perp$ 
22: Done
23:  $d_p \leftarrow x$  where  $x$  is the first non- $\perp$  value in  $Val_p$     /* Phase 3 */

```

Termination

Lemma 2

Every correct process eventually executes Line 23.

Proof.

The only way to prevent a correct process p from reaching Line 23 is to block it forever on **Line 7 or 18** in some round r .

Let $Faulty \subseteq V$ be the set of faulty processes. Let $Correct \subseteq V$ be the set of correct processes.

By definition, $V = Faulty \dot{\cup} Correct$.

By strong completeness, **eventually**
 $V \setminus S_p \subseteq Correct$ forever

We now consider the cases of **Line 7 and 18** separately.

```

1:  $d_p \leftarrow \perp$ 
2:  $Val_p \leftarrow [\perp, \dots, \perp]$ 
3:  $Val_p[p] \leftarrow v_p$ 
4:  $\Delta_p \leftarrow Val_p$ 
5: For all  $r_p$  from 1 to  $n-1$  do                                /* Phase 1 */
6:   broadcast  $\langle r_p, \Delta_p, p \rangle$  to  $V$  ( $p$  included)
7:   wait to receive  $\langle r_p, \Delta_q, q \rangle$  from  $q \in V \setminus S_p$ 
8:   let  $R_p[r_p]$  be the set of received  $\langle r_p, \Delta_q, q \rangle$ 
9:    $\Delta_p \leftarrow [\perp, \dots, \perp]$ 
10:  For all  $k \in V$  do
11:    If  $Val_p[k] = \perp \wedge (\exists (r_p, \Delta_q, q) \in R_p[r_p], \Delta_q[k] \neq \perp)$  then
12:       $Val_q[k] \leftarrow \Delta_q[k]$ 
13:       $\Delta_p[k] \leftarrow \Delta_q[k]$ 
14:    End If
15:  Done
16: Done
17: broadcast  $\langle Val_p \rangle$  to  $V$  ( $p$  included)                        /* Phase 2 */
18: wait to receive  $\langle Val_q \rangle$  from  $q \in V \setminus S_p$ 
19: Let  $lastmsgs_p$  be the set of received  $\langle Val_q \rangle$ 
20: For all  $k \in V$  do
21:   If  $\exists Val_q \in lastmsgs_p, Val_q[k] = \perp$  then  $Val_p[k] \leftarrow \perp$ 
22: Done
23:  $d_p \leftarrow x$  where  $x$  is the first non- $\perp$  value in  $Val_p$     /* Phase 3 */

```

Termination

Case 1: Line 7

By induction on r : $\forall p \in \text{Correct}, p$
completes every Round $r \in [1..n-1]$.

```

1:  $d_p \leftarrow \perp$ 
2:  $Val_p \leftarrow [\perp, \dots, \perp]$ 
3:  $Val_p[p] \leftarrow v_p$ 
4:  $\Delta_p \leftarrow Val_p$ 
5: For all  $r_p$  from 1 to  $n-1$  do /* Phase 1 */
6:   broadcast  $\langle r_p, \Delta_p, p \rangle$  to  $V$  ( $p$  included)
7:   wait to receive  $\langle r_p, \Delta_q, q \rangle$  from  $q \in V \setminus S_p$ 
8:   let  $R_p[r_p]$  be the set of received  $\langle r_p, \Delta_q, q \rangle$ 
9:    $\Delta_p \leftarrow [\perp, \dots, \perp]$ 
10:  For all  $k \in V$  do
11:    If  $Val_p[k] = \perp \wedge (\exists \langle r_p, \Delta_q, q \rangle \in R_p[r_p], \Delta_q[k] \neq \perp)$  then
12:       $Val_p[k] \leftarrow \Delta_q[k]$ 
13:       $\Delta_p[k] \leftarrow \Delta_q[k]$ 
14:    End If
15:  Done
16: Done
17: broadcast  $\langle Val_p \rangle$  to  $V$  ( $p$  included) /* Phase 2 */
18: wait to receive  $\langle Val_q \rangle$  from  $q \in V \setminus S_p$ 
19: Let  $lastmsgs_p$  be the set of received  $\langle Val_q \rangle$ 
20: For all  $k \in V$  do
21:   If  $\exists Val_q \in lastmsgs_p, Val_q[k] = \perp$  then  $Val_p[k] \leftarrow \perp$ 
22: Done
23:  $d_p \leftarrow x$  where  $x$  is the first non- $\perp$  value in  $Val_p$  /* Phase 3 */

```

Termination

Case 1: Line 7

By induction on r : $\forall p \in \text{Correct}$, p completes every Round $r \in [1..n-1]$.

$r = 1$: $\forall q \in \text{Correct}$, q completes Line 6 during Round 1.

By link reliability, $\forall p \in \text{Correct}$, p eventually receives $\langle 1, \Delta_q, q \rangle$ from every $q \in \text{Correct}$ in Round 1.

Since eventually $V \setminus S_p \subseteq \text{Correct}$ forever, p completes Round 1.

```

1:  $d_p \leftarrow \perp$ 
2:  $Val_p \leftarrow [\perp, \dots, \perp]$ 
3:  $Val_p[p] \leftarrow v_p$ 
4:  $\Delta_p \leftarrow Val_p$ 
5: For all  $r_p$  from 1 to  $n-1$  do                                /* Phase 1 */
6:   broadcast  $\langle r_p, \Delta_p, p \rangle$  to  $V$  ( $p$  included)
7:   wait to receive  $\langle r_p, \Delta_q, q \rangle$  from  $q \in V \setminus S_p$ 
8:   let  $R_p[r_p]$  be the set of received  $\langle r_p, \Delta_q, q \rangle$ 
9:    $\Delta_p \leftarrow [\perp, \dots, \perp]$ 
10:  For all  $k \in V$  do
11:    If  $Val_p[k] = \perp \wedge (\exists (r_p, \Delta_q, q) \in R_p[r_p], \Delta_q[k] \neq \perp)$  then
12:       $Val_p[k] \leftarrow \Delta_q[k]$ 
13:       $\Delta_p[k] \leftarrow \Delta_q[k]$ 
14:    End If
15:  Done
16: Done
17: broadcast  $\langle Val_p \rangle$  to  $V$  ( $p$  included)                        /* Phase 2 */
18: wait to receive  $\langle Val_q \rangle$  from  $q \in V \setminus S_p$ 
19: Let  $lastmsgs_p$  be the set of received  $\langle Val_q \rangle$ 
20: For all  $k \in V$  do
21:   If  $\exists Val_q \in lastmsgs_p, Val_q[k] = \perp$  then  $Val_p[k] \leftarrow \perp$ 
22: Done
23:  $d_p \leftarrow x$  where  $x$  is the first non- $\perp$  value in  $Val_p$     /* Phase 3 */

```

Termination

Case 1: Line 7

By induction on r : $\forall p \in \text{Correct}$, p completes every Round $r \in [1..n-1]$.

$r = 1$: $\forall q \in \text{Correct}$, q completes Line 6 during Round 1.

By link reliability, $\forall p \in \text{Correct}$, p eventually receives $\langle 1, \Delta_q, q \rangle$ from every $q \in \text{Correct}$ in Round 1.

Since eventually $V \setminus S_p \subseteq \text{Correct}$ forever, p completes Round 1.

$r > 1$: By induction hypothesis, $\forall q \in \text{Correct}$, q completes Round $r-1$, so q completes Line 6 during Round r .

By link reliability, $\forall p \in \text{Correct}$, p eventually receives $\langle r, \Delta_q, q \rangle$ from every $q \in \text{Correct}$ in Round r .

Since eventually $V \setminus S_p \subseteq \text{Correct}$ forever, $\forall p \in \text{Correct}$, p completes Round r .

```

1:  $d_p \leftarrow \perp$ 
2:  $Val_p \leftarrow [\perp, \dots, \perp]$ 
3:  $Val_p[p] \leftarrow v_p$ 
4:  $\Delta_p \leftarrow Val_p$ 
5: For all  $r_p$  from 1 to  $n-1$  do                                /* Phase 1 */
6:   broadcast  $\langle r_p, \Delta_p, p \rangle$  to  $V$  ( $p$  included)
7:   wait to receive  $\langle r_p, \Delta_q, q \rangle$  from  $q \in V \setminus S_p$ 
8:   let  $R_p[r_p]$  be the set of received  $\langle r_p, \Delta_q, q \rangle$ 
9:    $\Delta_p \leftarrow [\perp, \dots, \perp]$ 
10:  For all  $k \in V$  do
11:    If  $Val_p[k] = \perp \wedge (\exists (r_p, \Delta_q, q) \in R_p[r_p], \Delta_q[k] \neq \perp)$  then
12:       $Val_p[k] \leftarrow \Delta_q[k]$ 
13:       $\Delta_p[k] \leftarrow \Delta_q[k]$ 
14:    End If
15:  Done
16: Done
17: broadcast  $\langle Val_p \rangle$  to  $V$  ( $p$  included)                        /* Phase 2 */
18: wait to receive  $\langle Val_q \rangle$  from  $q \in V \setminus S_p$ 
19: Let  $lastmsgsp$  be the set of received  $\langle Val_q \rangle$ 
20: For all  $k \in V$  do
21:   If  $\exists Val_q \in lastmsgsp, Val_q[k] = \perp$  then  $Val_p[k] \leftarrow \perp$ 
22: Done
23:  $d_p \leftarrow x$  where  $x$  is the first non- $\perp$  value in  $Val_p$     /* Phase 3 */

```

Termination

Case 2: Line 18

From the previous induction,
 $\forall p \in \text{Correct}$, p completes Phase 1.

So, $\forall p \in \text{Correct}$, p completes Line 17.

By link reliability, $\forall p \in \text{Correct}$, p
 eventually receives $\langle \text{Val}_q \rangle$ from every
 $q \in \text{Correct}$.

Since eventually $V \setminus S_p \subseteq \text{Correct}$ forever,
 $\forall p \in \text{Correct}$, p completes Line 18. \square

```

1:  $d_p \leftarrow \perp$ 
2:  $\text{Val}_p \leftarrow [\perp, \dots, \perp]$ 
3:  $\text{Val}_p[p] \leftarrow v_p$ 
4:  $\Delta_p \leftarrow \text{Val}_p$ 
5: For all  $r_p$  from 1 to  $n-1$  do /* Phase 1 */
6:   broadcast  $\langle r_p, \Delta_p, p \rangle$  to  $V$  ( $p$  included)
7:   wait to receive  $\langle r_p, \Delta_q, q \rangle$  from  $q \in V \setminus S_p$ 
8:   let  $R_p[r_p]$  be the set of received  $\langle r_p, \Delta_q, q \rangle$ 
9:    $\Delta_p \leftarrow [\perp, \dots, \perp]$ 
10:  For all  $k \in V$  do
11:    If  $\text{Val}_p[k] = \perp \wedge (\exists (r_p, \Delta_q, q) \in R_p[r_p], \Delta_q[k] \neq \perp)$  then
12:       $\text{Val}_p[k] \leftarrow \Delta_q[k]$ 
13:       $\Delta_p[k] \leftarrow \Delta_q[k]$ 
14:    End If
15:  Done
16: Done
17: broadcast  $\langle \text{Val}_p \rangle$  to  $V$  ( $p$  included) /* Phase 2 */
18: wait to receive  $\langle \text{Val}_q \rangle$  from  $q \in V \setminus S_p$ 
19: Let  $\text{lastmsgs}_p$  be the set of received  $\langle \text{Val}_q \rangle$ 
20: For all  $k \in V$  do
21:   If  $\exists \text{Val}_q \in \text{lastmsgs}_p, \text{Val}_q[k] = \perp$  then  $\text{Val}_p[k] \leftarrow \perp$ 
22: Done
23:  $d_p \leftarrow x$  where  $x$  is the first non- $\perp$  value in  $\text{Val}_p$  /* Phase 3 */

```

Notations

- By **weak accuracy**: there is a correct process c which is never suspected.
- Let Π_1 be the set of processes that terminate the $n - 1$ rounds of Phase 1.
- Let Π_2 be the set of processes that terminate Phase 2.

By definition, $\Pi_2 \subseteq \Pi_1$.

- We note $Val_p \leq Val_q$ IFF $\forall k$, either $Val_p[k] = Val_q[k]$ or $Val_p[k] = \perp$.

Messages from c

Lemma 3

During Round r , with $1 \leq r \leq n-1$, every process of Π_1 receives (r, Δ_c, c) from c , i.e., $(r, \Delta_c, c) \in R_p[c]$.

Proof. Since $p \in \Pi_1$, p terminates all rounds of Phase 1.

At each round, p waits and receives a $\langle r, \Delta_c, c \rangle$ message from c since $c \notin S_p$ forever. \square

```

1:  $d_p \leftarrow \perp$ 
2:  $Val_p \leftarrow [\perp, \dots, \perp]$ 
3:  $Val_p[p] \leftarrow v_p$ 
4:  $\Delta_p \leftarrow Val_p$ 
5: For all  $r_p$  from 1 to  $n-1$  do                                /* Phase 1 */
6:   broadcast  $\langle r_p, \Delta_p, p \rangle$  to  $V$  ( $p$  included)
7:   wait to receive  $\langle r_p, \Delta_q, q \rangle$  from  $q \in V \setminus S_p$ 
8:   let  $R_p[r_p]$  be the set of received  $\langle r_p, \Delta_q, q \rangle$ 
9:    $\Delta_p \leftarrow [\perp, \dots, \perp]$ 
10:  For all  $k \in V$  do
11:    If  $Val_p[k] = \perp \wedge (\exists (r_p, \Delta_q, q) \in R_p[r_p], \Delta_q[k] \neq \perp)$  then
12:       $Val_p[k] \leftarrow \Delta_q[k]$ 
13:       $\Delta_p[k] \leftarrow \Delta_q[k]$ 
14:    End If
15:  Done
16: Done
17: broadcast  $\langle Val_p \rangle$  to  $V$  ( $p$  included)                        /* Phase 2 */
18: wait to receive  $\langle Val_q \rangle$  from  $q \in V \setminus S_p$ 
19: Let  $lastmsgs_p$  be the set of received  $\langle Val_q \rangle$ 
20: For all  $k \in V$  do
21:   If  $\exists Val_q \in lastmsgs_p, Val_q[k] = \perp$  then  $Val_p[k] \leftarrow \perp$ 
22: Done
23:  $d_p \leftarrow x$  where  $x$  is the first non- $\perp$  value in  $Val_p$     /* Phase 3 */

```

Learning Values

Lemma 4

$\forall p \in \Pi_1, Val_c \leq Val_p$ at the end of Phase 1.

Proof.

Assume that $Val_c[q] \neq \perp$ at the end of Phase 1 for some process q .

By Lemma 1, $Val_c[q] = v_q$.

Let $p \in \Pi_1$. We now show that $Val_p[q] = v_q$ at the end of Phase 1.

The case $p = c$ is trivial. So, we now assume that $p \neq c$.

Let r be the first round where c receives v_q (if $c = q$, we let $r = 0$ and assume the end of Round 0 is Line 4)

$\Delta_c[q] = v_q$ at the end of Round r .

Two cases: $r < n-1$ and $r = n-1$

```

1:  $d_p \leftarrow \perp$ 
2:  $Val_p \leftarrow [\perp, \dots, \perp]$ 
3:  $Val_p[p] \leftarrow v_p$ 
4:  $\Delta_p \leftarrow Val_p$ 
5: For all  $r_p$  from 1 to  $n-1$  do                               /* Phase 1 */
6:   broadcast  $\langle r_p, \Delta_p, p \rangle$  to  $V$  ( $p$  included)
7:   wait to receive  $\langle r_p, \Delta_q, q \rangle$  from  $q \in V \setminus S_p$ 
8:   let  $R_p[r_p]$  be the set of received  $\langle r_p, \Delta_q, q \rangle$ 
9:    $\Delta_p \leftarrow [\perp, \dots, \perp]$ 
10:  For all  $k \in V$  do
11:    If  $Val_p[k] = \perp \wedge (\exists (r_p, \Delta_q, q) \in R_p[r_p], \Delta_q[k] \neq \perp)$  then
12:       $Val_p[k] \leftarrow \Delta_q[k]$ 
13:       $\Delta_p[k] \leftarrow \Delta_q[k]$ 
14:    End If
15:  Done
16: Done
17: broadcast  $\langle Val_p \rangle$  to  $V$  ( $p$  included)                       /* Phase 2 */
18: wait to receive  $\langle Val_q \rangle$  from  $q \in V \setminus S_p$ 
19: Let  $lastmsgs_p$  be the set of received  $\langle Val_q \rangle$ 
20: For all  $k \in V$  do
21:   If  $\exists Val_q \in lastmsgs_p, Val_q[k] = \perp$  then  $Val_p[k] \leftarrow \perp$ 
22: Done
23:  $d_p \leftarrow x$  where  $x$  is the first non- $\perp$  value in  $Val_p$    /* Phase 3 */

```

Learning Values

Case 1: $r < n - 1$

During Round $r + 1 \leq n - 1$, c relays v_q by **broadcasting** $\langle r + 1, \Delta_c, c \rangle$ with $\Delta_c[q] = v_q$.

```

1:  $d_p \leftarrow \perp$ 
2:  $Val_p \leftarrow [\perp, \dots, \perp]$ 
3:  $Val_p[p] \leftarrow v_p$ 
4:  $\Delta_p \leftarrow Val_p$ 
5: For all  $r_p$  from 1 to  $n - 1$  do /* Phase 1 */
6:   broadcast  $\langle r_p, \Delta_p, p \rangle$  to  $V$  ( $p$  included)
7:   wait to receive  $\langle r_p, \Delta_q, q \rangle$  from  $q \in V \setminus S_p$ 
8:   let  $R_p[r_p]$  be the set of received  $\langle r_p, \Delta_q, q \rangle$ 
9:    $\Delta_p \leftarrow [\perp, \dots, \perp]$ 
10:  For all  $k \in V$  do
11:    If  $Val_p[k] = \perp \wedge (\exists \langle r_p, \Delta_q, q \rangle \in R_p[r_p], \Delta_q[k] \neq \perp)$  then
12:       $Val_p[k] \leftarrow \Delta_q[k]$ 
13:       $\Delta_p[k] \leftarrow \Delta_q[k]$ 
14:    End If
15:  Done
16: Done
17: broadcast  $\langle Val_p \rangle$  to  $V$  ( $p$  included) /* Phase 2 */
18: wait to receive  $\langle Val_q \rangle$  from  $q \in V \setminus S_p$ 
19: Let  $lastmsgs_p$  be the set of received  $\langle Val_q \rangle$ 
20: For all  $k \in V$  do
21:   If  $\exists Val_q \in lastmsgs_p, Val_q[k] = \perp$  then  $Val_p[k] \leftarrow \perp$ 
22: Done
23:  $d_p \leftarrow x$  where  $x$  is the first non- $\perp$  value in  $Val_p$  /* Phase 3 */

```

Learning Values

Case 1: $r < n - 1$

During Round $r + 1 \leq n - 1$, c relays v_q by **broadcasting** $\langle r + 1, \Delta_c, c \rangle$ with $\Delta_c[q] = v_q$.

By Lemma 3, p receives $\langle r + 1, \Delta_c, c \rangle$ during Round $r + 1$.

```

1:  $d_p \leftarrow \perp$ 
2:  $Val_p \leftarrow [\perp, \dots, \perp]$ 
3:  $Val_p[p] \leftarrow v_p$ 
4:  $\Delta_p \leftarrow Val_p$ 
5: For all  $r_p$  from 1 to  $n - 1$  do /* Phase 1 */
6:   broadcast  $\langle r_p, \Delta_p, p \rangle$  to  $V$  ( $p$  included)
7:   wait to receive  $\langle r_p, \Delta_q, q \rangle$  from  $q \in V \setminus S_p$ 
8:   let  $R_p[r_p]$  be the set of received  $\langle r_p, \Delta_q, q \rangle$ 
9:    $\Delta_p \leftarrow [\perp, \dots, \perp]$ 
10:  For all  $k \in V$  do
11:    If  $Val_p[k] = \perp \wedge (\exists (r_p, \Delta_q, q) \in R_p[r_p], \Delta_q[k] \neq \perp)$  then
12:       $Val_p[k] \leftarrow \Delta_q[k]$ 
13:       $\Delta_p[k] \leftarrow \Delta_q[k]$ 
14:    End If
15:  Done
16: Done
17: broadcast  $\langle Val_p \rangle$  to  $V$  ( $p$  included) /* Phase 2 */
18: wait to receive  $\langle Val_q \rangle$  from  $q \in V \setminus S_p$ 
19: Let  $lastmsgs_p$  be the set of received  $\langle Val_q \rangle$ 
20: For all  $k \in V$  do
21:   If  $\exists Val_q \in lastmsgs_p, Val_q[k] = \perp$  then  $Val_p[k] \leftarrow \perp$ 
22: Done
23:  $d_p \leftarrow x$  where  $x$  is the first non- $\perp$  value in  $Val_p$  /* Phase 3 */

```

Learning Values

Case 1: $r < n - 1$

During Round $r + 1 \leq n - 1$, c relays v_q by **broadcasting** $\langle r + 1, \Delta_c, c \rangle$ with $\Delta_c[q] = v_q$.

By Lemma 3, p receives $\langle r + 1, \Delta_c, c \rangle$ during Round $r + 1$.

From the code of the algorithm, $Val_p[q] = v_q$ at the end of round $r + 1$, and so $Val_p[q] = v_q$ at the end of Phase 1.

```

1:  $d_p \leftarrow \perp$ 
2:  $Val_p \leftarrow [\perp, \dots, \perp]$ 
3:  $Val_p[p] \leftarrow v_p$ 
4:  $\Delta_p \leftarrow Val_p$ 
5: For all  $r_p$  from 1 to  $n - 1$  do /* Phase 1 */
6:   broadcast  $\langle r_p, \Delta_p, p \rangle$  to  $V$  ( $p$  included)
7:   wait to receive  $\langle r_p, \Delta_q, q \rangle$  from  $q \in V \setminus S_p$ 
8:   let  $R_p[r_p]$  be the set of received  $\langle r_p, \Delta_q, q \rangle$ 
9:    $\Delta_p \leftarrow [\perp, \dots, \perp]$ 
10:  For all  $k \in V$  do
11:    If  $Val_p[k] = \perp \wedge (\exists \langle r_p, \Delta_q, q \rangle \in R_p[r_p], \Delta_q[k] \neq \perp)$  then
12:       $Val_p[k] \leftarrow \Delta_q[k]$ 
13:       $\Delta_p[k] \leftarrow \Delta_q[k]$ 
14:    End If
15:  Done
16: Done
17: broadcast  $\langle Val_p \rangle$  to  $V$  ( $p$  included) /* Phase 2 */
18: wait to receive  $\langle Val_q \rangle$  from  $q \in V \setminus S_p$ 
19: Let  $lastmsgs_p$  be the set of received  $\langle Val_q \rangle$ 
20: For all  $k \in V$  do
21:   If  $\exists Val_q \in lastmsgs_p, Val_q[k] = \perp$  then  $Val_p[k] \leftarrow \perp$ 
22: Done
23:  $d_p \leftarrow x$  where  $x$  is the first non- $\perp$  value in  $Val_p$  /* Phase 3 */

```

Learning Values

Case 2: $r = n - 1$

Since c receives v_q for the first time during Round $n - 1$ and every process relays v_q at most once:

v_q has been relayed by $n - 1$ distinct different from c before reaching c .

p necessarily belongs to this set of processes.

□

```

1:  $d_p \leftarrow \perp$ 
2:  $Val_p \leftarrow [\perp, \dots, \perp]$ 
3:  $Val_p[p] \leftarrow v_p$ 
4:  $\Delta_p \leftarrow Val_p$ 
5: For all  $r_p$  from 1 to  $n - 1$  do                                /* Phase 1 */
6:   broadcast  $\langle r_p, \Delta_p, p \rangle$  to  $V$  ( $p$  included)
7:   wait to receive  $\langle r_p, \Delta_q, q \rangle$  from  $q \in V \setminus S_p$ 
8:   let  $R_p[r_p]$  be the set of received  $\langle r_p, \Delta_q, q \rangle$ 
9:    $\Delta_p \leftarrow [\perp, \dots, \perp]$ 
10:  For all  $k \in V$  do
11:    If  $Val_p[k] = \perp \wedge (\exists (r_p, \Delta_q, q) \in R_p[r_p], \Delta_q[k] \neq \perp)$  then
12:       $Val_p[k] \leftarrow \Delta_q[k]$ 
13:       $\Delta_p[k] \leftarrow \Delta_q[k]$ 
14:    End If
15:  Done
16: Done
17: broadcast  $\langle Val_p \rangle$  to  $V$  ( $p$  included)                        /* Phase 2 */
18: wait to receive  $\langle Val_q \rangle$  from  $q \in V \setminus S_p$ 
19: Let  $lastmsgs_p$  be the set of received  $\langle Val_q \rangle$ 
20: For all  $k \in V$  do
21:   If  $\exists Val_q \in lastmsgs_p, Val_q[k] = \perp$  then  $Val_p[k] \leftarrow \perp$ 
22: Done
23:  $d_p \leftarrow x$  where  $x$  is the first non- $\perp$  value in  $Val_p$     /* Phase 3 */
    
```

Learning Values

Case 2: $r = n - 1$

Since c receives v_q for the first time during Round $n - 1$ and every process relays v_q at most once:

v_q has been relayed by $n - 1$ distinct different from c before reaching c .

p necessarily belongs to this set of processes.

Now, $Val_p[q] = v_q$ right before relaying v_q , so $Val_p[q] = v_q$ at the end of Phase 1. □

```

1:  $d_p \leftarrow \perp$ 
2:  $Val_p \leftarrow [\perp, \dots, \perp]$ 
3:  $Val_p[p] \leftarrow v_p$ 
4:  $\Delta_p \leftarrow Val_p$ 
5: For all  $r_p$  from 1 to  $n - 1$  do /* Phase 1 */
6:   broadcast  $\langle r_p, \Delta_p, p \rangle$  to  $V$  ( $p$  included)
7:   wait to receive  $\langle r_p, \Delta_q, q \rangle$  from  $q \in V \setminus S_p$ 
8:   let  $R_p[r_p]$  be the set of received  $\langle r_p, \Delta_q, q \rangle$ 
9:    $\Delta_p \leftarrow [\perp, \dots, \perp]$ 
10:  For all  $k \in V$  do
11:    If  $Val_p[k] = \perp \wedge (\exists (r_p, \Delta_q, q) \in R_p[r_p], \Delta_q[k] \neq \perp)$  then
12:       $Val_p[k] \leftarrow \Delta_q[k]$ 
13:       $\Delta_p[k] \leftarrow \Delta_q[k]$ 
14:    End If
15:  Done
16: Done
17: broadcast  $\langle Val_p \rangle$  to  $V$  ( $p$  included) /* Phase 2 */
18: wait to receive  $\langle Val_q \rangle$  from  $q \in V \setminus S_p$ 
19: Let  $lastmsgs_p$  be the set of received  $\langle Val_q \rangle$ 
20: For all  $k \in V$  do
21:   If  $\exists Val_q \in lastmsgs_p, Val_q[k] = \perp$  then  $Val_p[k] \leftarrow \perp$ 
22: Done
23:  $d_p \leftarrow x$  where  $x$  is the first non- $\perp$  value in  $Val_p$  /* Phase 3 */

```

Decision (1/3)

Lemma 5

$\forall p \in \Pi_2$, $Val_c = Val_p$ at the end of Phase 2.

Proof. Let $p \in \Pi_2$. Let q be a process. We should show that $Val_p[q] = Val_c[q]$ at the end of Phase 2.

By Lemma 1, we have the following two cases:

- $Val_c[q] = v_q$ at the end of Phase 1.

- $Val_c[q] = \perp$ at the end of Phase 1.

```

1:  $d_p \leftarrow \perp$ 
2:  $Val_p \leftarrow [\perp, \dots, \perp]$ 
3:  $Val_p[p] \leftarrow v_p$ 
4:  $\Delta_p \leftarrow Val_p$ 
5: For all  $r_p$  from 1 to  $n-1$  do                                /* Phase 1 */
6:   broadcast  $\langle r_p, \Delta_p, p \rangle$  to  $V$  ( $p$  included)
7:   wait to receive  $\langle r_p, \Delta_q, q \rangle$  from  $q \in V \setminus S_p$ 
8:   let  $R_p[r_p]$  be the set of received  $\langle r_p, \Delta_q, q \rangle$ 
9:    $\Delta_p \leftarrow [\perp, \dots, \perp]$ 
10:  For all  $k \in V$  do
11:    If  $Val_p[k] = \perp \wedge (\exists (r_p, \Delta_q, q) \in R_p[r_p], \Delta_q[k] \neq \perp)$  then
12:       $Val_p[k] \leftarrow \Delta_q[k]$ 
13:       $\Delta_p[k] \leftarrow \Delta_q[k]$ 
14:    End If
15:  Done
16: Done
17: broadcast  $\langle Val_p \rangle$  to  $V$  ( $p$  included)                        /* Phase 2 */
18: wait to receive  $\langle Val_q \rangle$  from  $q \in V \setminus S_p$ 
19: Let  $lastmsgs_p$  be the set of received  $\langle Val_q \rangle$ 
20: For all  $k \in V$  do
21:   If  $\exists Val_q \in lastmsgs_p, Val_q[k] = \perp$  then  $Val_p[k] \leftarrow \perp$ 
22: Done
23:  $d_p \leftarrow x$  where  $x$  is the first non- $\perp$  value in  $Val_p$     /* Phase 3 */

```

□

Decision (1/3)

Lemma 5

$\forall p \in \Pi_2$, $Val_c = Val_p$ at the end of Phase 2.

Proof. Let $p \in \Pi_2$. Let q be a process. We should show that $Val_p[q] = Val_c[q]$ at the end of Phase 2.

By Lemma 1, we have the following two cases:

- $Val_c[q] = v_q$ at the end of Phase 1.

By Lemma 4, $\forall p' \in \Pi_1$ (p and c included), $Val_{p'}[q] = v_q$ at the end of Phase 1.

Thus, for all vectors V sent during Phase 2, we have $V[q] = v_q$.

Hence, $Val_p[q]$ and $Val_c[q]$ remain equal to v_q during Phase 2.

- $Val_c[q] = \perp$ at the end of Phase 1.

```

1:  $d_p \leftarrow \perp$ 
2:  $Val_p \leftarrow [\perp, \dots, \perp]$ 
3:  $Val_p[p] \leftarrow v_p$ 
4:  $\Delta_p \leftarrow Val_p$ 
5: For all  $r_p$  from 1 to  $n-1$  do                               /* Phase 1 */
6:   broadcast  $\langle r_p, \Delta_p, p \rangle$  to  $V$  ( $p$  included)
7:   wait to receive  $\langle r_p, \Delta_q, q \rangle$  from  $q \in V \setminus S_p$ 
8:   let  $R_p[r_p]$  be the set of received  $\langle r_p, \Delta_q, q \rangle$ 
9:    $\Delta_p \leftarrow [\perp, \dots, \perp]$ 
10:  For all  $k \in V$  do
11:    If  $Val_p[k] = \perp \wedge (\exists (r_p, \Delta_q, q) \in R_p[r_p], \Delta_q[k] \neq \perp)$  then
12:       $Val_p[k] \leftarrow \Delta_q[k]$ 
13:       $\Delta_p[k] \leftarrow \Delta_q[k]$ 
14:    End If
15:  Done
16: Done
17: broadcast  $\langle Val_p \rangle$  to  $V$  ( $p$  included)                       /* Phase 2 */
18: wait to receive  $\langle Val_q \rangle$  from  $q \in V \setminus S_p$ 
19: Let  $lastmsgs_p$  be the set of received  $\langle Val_q \rangle$ 
20: For all  $k \in V$  do
21:   If  $\exists Val_q \in lastmsgs_p, Val_q[k] = \perp$  then  $Val_p[k] \leftarrow \perp$ 
22: Done
23:  $d_p \leftarrow x$  where  $x$  is the first non- $\perp$  value in  $Val_p$    /* Phase 3 */

```

□

Decision (1/3)

Lemma 5

$\forall p \in \Pi_2$, $Val_c = Val_p$ at the end of Phase 2.

Proof. Let $p \in \Pi_2$. Let q be a process. We should show that $Val_p[q] = Val_c[q]$ at the end of Phase 2.

By Lemma 1, we have the following two cases:

- $Val_c[q] = v_q$ at the end of Phase 1.

By Lemma 4, $\forall p' \in \Pi_1$ (p and c included), $Val_{p'}[q] = v_q$ at the end of Phase 1.

Thus, for all vectors V sent during Phase 2, we have $V[q] = v_q$.

Hence, $Val_p[q]$ and $Val_c[q]$ remain equal to v_q during Phase 2.

- $Val_c[q] = \perp$ at the end of Phase 1.

$Val_c[q] = \perp$ at the end of Phase 2.

```

1:  $d_p \leftarrow \perp$ 
2:  $Val_p \leftarrow [\perp, \dots, \perp]$ 
3:  $Val_p[p] \leftarrow v_p$ 
4:  $\Delta_p \leftarrow Val_p$ 
5: For all  $r_p$  from 1 to  $n-1$  do                               /* Phase 1 */
6:   broadcast  $\langle r_p, \Delta_p, p \rangle$  to  $V$  ( $p$  included)
7:   wait to receive  $\langle r_p, \Delta_q, q \rangle$  from  $q \in V \setminus S_p$ 
8:   let  $R_p[r_p]$  be the set of received  $\langle r_p, \Delta_q, q \rangle$ 
9:    $\Delta_p \leftarrow [\perp, \dots, \perp]$ 
10:  For all  $k \in V$  do
11:    If  $Val_p[k] = \perp \wedge (\exists (r_p, \Delta_q, q) \in R_p[r_p], \Delta_q[k] \neq \perp)$  then
12:       $Val_p[k] \leftarrow \Delta_q[k]$ 
13:       $\Delta_p[k] \leftarrow \Delta_q[k]$ 
14:    End If
15:  Done
16: Done
17: broadcast  $\langle Val_p \rangle$  to  $V$  ( $p$  included)                       /* Phase 2 */
18: wait to receive  $\langle Val_q \rangle$  from  $q \in V \setminus S_p$ 
19: Let  $lastmsgs_p$  be the set of received  $\langle Val_q \rangle$ 
20: For all  $k \in V$  do
21:   If  $\exists Val_q \in lastmsgs_p, Val_q[k] = \perp$  then  $Val_p[k] \leftarrow \perp$ 
22: Done
23:  $d_p \leftarrow x$  where  $x$  is the first non- $\perp$  value in  $Val_p$    /* Phase 3 */

```

□

Decision (1/3)

Lemma 5

$\forall p \in \Pi_2, Val_c = Val_p$ at the end of Phase 2.

Proof. Let $p \in \Pi_2$. Let q be a process. We should show that $Val_p[q] = Val_c[q]$ at the end of Phase 2.

By Lemma 1, we have the following two cases:

- $Val_c[q] = v_q$ at the end of Phase 1.

By Lemma 4, $\forall p' \in \Pi_1$ (p and c included), $Val_{p'}[q] = v_q$ at the end of Phase 1.

Thus, for all vectors V sent during Phase 2, we have $V[q] = v_q$.

Hence, $Val_p[q]$ and $Val_c[q]$ remain equal to v_q during Phase 2.

- $Val_c[q] = \perp$ at the end of Phase 1.

$Val_c[q] = \perp$ at the end of Phase 2.

Since $c \notin S_p$ forever, p waits and receives Val_c during Phase 2.

Since $Val_c[q] = \perp$, p sets $Val_p[q]$ to \perp during Phase 2: $Val_c[q] = Val_p[q]$ at the end of Phase 2. \square

```

1:  $d_p \leftarrow \perp$ 
2:  $Val_p \leftarrow [\perp, \dots, \perp]$ 
3:  $Val_p[p] \leftarrow v_p$ 
4:  $\Delta_p \leftarrow Val_p$ 
5: For all  $r_p$  from 1 to  $n-1$  do                               /* Phase 1 */
6:   broadcast  $\langle r_p, \Delta_p, p \rangle$  to  $V$  ( $p$  included)
7:   wait to receive  $\langle r_p, \Delta_q, q \rangle$  from  $q \in V \setminus S_p$ 
8:   let  $R_p[r_p]$  be the set of received  $\langle r_p, \Delta_q, q \rangle$ 
9:    $\Delta_p \leftarrow [\perp, \dots, \perp]$ 
10:  For all  $k \in V$  do
11:    if  $Val_p[k] = \perp \wedge (\exists (r_p, \Delta_q, q) \in R_p[r_p], \Delta_q[k] \neq \perp)$  then
12:       $Val_p[k] \leftarrow \Delta_q[k]$ 
13:       $\Delta_p[k] \leftarrow \Delta_q[k]$ 
14:    End if
15:  Done
16: Done
17: broadcast  $\langle Val_p \rangle$  to  $V$  ( $p$  included)                       /* Phase 2 */
18: wait to receive  $\langle Val_q \rangle$  from  $q \in V \setminus S_p$ 
19: Let  $lastmsgs_p$  be the set of received  $\langle Val_q \rangle$ 
20: For all  $k \in V$  do
21:   if  $\exists Val_q \in lastmsgs_p, Val_q[k] = \perp$  then  $Val_p[k] \leftarrow \perp$ 
22: Done
23:  $d_p \leftarrow x$  where  $x$  is the first non- $\perp$  value in  $Val_p$    /* Phase 3 */

```

Decision (2/2)

Corollary 1

No two processes decide differently.

Proof.

By Lemma 5, all processes that execute Phase 3 have the same vector.

So, by Line 23, if they decide, they take the same decision. \square

```

1:  $d_p \leftarrow \perp$ 
2:  $Val_p \leftarrow [\perp, \dots, \perp]$ 
3:  $Val_p[\rho] \leftarrow v_\rho$ 
4:  $\Delta_p \leftarrow Val_p$ 
5: For all  $r_p$  from 1 to  $n-1$  do /* Phase 1 */
6:   broadcast  $\langle r_p, \Delta_p, \rho \rangle$  to  $V$  ( $p$  included)
7:   wait to receive  $\langle r_p, \Delta_q, q \rangle$  from  $q \in V \setminus S_p$ 
8:   let  $R_p[r_p]$  be the set of received  $\langle r_p, \Delta_q, q \rangle$ 
9:    $\Delta_p \leftarrow [\perp, \dots, \perp]$ 
10:  For all  $k \in V$  do
11:    If  $Val_p[k] = \perp \wedge (\exists (r_p, \Delta_q, q) \in R_p[r_p], \Delta_q[k] \neq \perp)$  then
12:       $Val_p[k] \leftarrow \Delta_q[k]$ 
13:       $\Delta_p[k] \leftarrow \Delta_q[k]$ 
14:    End If
15:  Done
16: Done
17: broadcast  $\langle Val_p \rangle$  to  $V$  ( $p$  included) /* Phase 2 */
18: wait to receive  $\langle Val_q \rangle$  from  $q \in V \setminus S_p$ 
19: Let  $lastmsgs_p$  be the set of received  $\langle Val_q \rangle$ 
20: For all  $k \in V$  do
21:   If  $\exists Val_q \in lastmsgs_p, Val_q[k] = \perp$  then  $Val_p[k] \leftarrow \perp$ 
22: Done
23:  $d_p \leftarrow x$  where  $x$  is the first non- $\perp$  value in  $Val_p$  /* Phase 3 */

```

Termination (continued: Line 23 is a decision)

Lemma 6

$\forall p \in \Pi_2, Val_p[c] = v_c$ at the end of Phase 2.

Proof.

□

```

1:  $d_p \leftarrow \perp$ 
2:  $Val_p \leftarrow [\perp, \dots, \perp]$ 
3:  $Val_p[p] \leftarrow v_p$ 
4:  $\Delta_p \leftarrow Val_p$ 
5: For all  $r_p$  from 1 to  $n-1$  do                                /* Phase 1 */
6:   broadcast  $\langle r_p, \Delta_p, p \rangle$  to  $V$  ( $p$  included)
7:   wait to receive  $\langle r_p, \Delta_q, q \rangle$  from  $q \in V \setminus S_p$ 
8:   let  $R_p[r_p]$  be the set of received  $\langle r_p, \Delta_q, q \rangle$ 
9:    $\Delta_p \leftarrow [\perp, \dots, \perp]$ 
10:  For all  $k \in V$  do
11:    If  $Val_p[k] = \perp \wedge (\exists \langle r_p, \Delta_q, q \rangle \in R_p[r_p], \Delta_q[k] \neq \perp)$  then
12:       $Val_p[k] \leftarrow \Delta_q[k]$ 
13:       $\Delta_p[k] \leftarrow \Delta_q[k]$ 
14:    End If
15:  Done
16: Done
17: broadcast  $\langle Val_p \rangle$  to  $V$  ( $p$  included)                    /* Phase 2 */
18: wait to receive  $\langle Val_q \rangle$  from  $q \in V \setminus S_p$ 
19: Let  $lastmsgs_p$  be the set of received  $\langle Val_q \rangle$ 
20: For all  $k \in V$  do
21:   If  $\exists Val_q \in lastmsgs_p, Val_q[k] = \perp$  then  $Val_p[k] \leftarrow \perp$ 
22: Done
23:  $d_p \leftarrow x$  where  $x$  is the first non- $\perp$  value in  $Val_p$   /* Phase 3 */

```

Termination (continued: Line 23 is a decision)

Lemma 6

$\forall p \in \Pi_2, Val_p[c] = v_c$ at the end of Phase 2.

Proof. From the code of the algorithm (Line 3), we know that $Val_c[c] = v_c$ at the end of Phase 1.

□

```

1:  $d_p \leftarrow \perp$ 
2:  $Val_p \leftarrow [\perp, \dots, \perp]$ 
3:  $Val_p[p] \leftarrow v_p$ 
4:  $\Delta_p \leftarrow Val_p$ 
5: For all  $r_p$  from 1 to  $n-1$  do                                /* Phase 1 */
6:   broadcast  $\langle r_p, \Delta_p, p \rangle$  to  $V$  ( $p$  included)
7:   wait to receive  $\langle r_p, \Delta_q, q \rangle$  from  $q \in V \setminus S_p$ 
8:   let  $R_p[r_p]$  be the set of received  $\langle r_p, \Delta_q, q \rangle$ 
9:    $\Delta_p \leftarrow [\perp, \dots, \perp]$ 
10:  For all  $k \in V$  do
11:    If  $Val_p[k] = \perp \wedge (\exists \langle r_p, \Delta_q, q \rangle \in R_p[r_p], \Delta_q[k] \neq \perp)$  then
12:       $Val_p[k] \leftarrow \Delta_q[k]$ 
13:       $\Delta_p[k] \leftarrow \Delta_q[k]$ 
14:    End If
15:  Done
16: Done
17: broadcast  $\langle Val_p \rangle$  to  $V$  ( $p$  included)                        /* Phase 2 */
18: wait to receive  $\langle Val_q \rangle$  from  $q \in V \setminus S_p$ 
19: Let  $lastmsgs_p$  be the set of received  $\langle Val_q \rangle$ 
20: For all  $k \in V$  do
21:   If  $\exists Val_q \in lastmsgs_p, Val_q[k] = \perp$  then  $Val_p[k] \leftarrow \perp$ 
22: Done
23:  $d_p \leftarrow x$  where  $x$  is the first non- $\perp$  value in  $Val_p$     /* Phase 3 */

```

Termination (continued: Line 23 is a decision)

Lemma 6

$\forall p \in \Pi_2, Val_p[c] = v_c$ at the end of Phase 2.

Proof. From the code of the algorithm (Line 3), we know that $Val_c[c] = v_c$ at the end of Phase 1.

By Lemma 4, $\forall q \in \Pi_1, Val_q[c] = v_c$ at the end of Phase 1.

□

```

1:  $d_p \leftarrow \perp$ 
2:  $Val_p \leftarrow [\perp, \dots, \perp]$ 
3:  $Val_p[p] \leftarrow v_p$ 
4:  $\Delta_p \leftarrow Val_p$ 
5: For all  $r_p$  from 1 to  $n-1$  do                                /* Phase 1 */
6:   broadcast  $\langle r_p, \Delta_p, p \rangle$  to  $V$  ( $p$  included)
7:   wait to receive  $\langle r_p, \Delta_q, q \rangle$  from  $q \in V \setminus S_p$ 
8:   let  $R_p[r_p]$  be the set of received  $\langle r_p, \Delta_q, q \rangle$ 
9:    $\Delta_p \leftarrow [\perp, \dots, \perp]$ 
10:  For all  $k \in V$  do
11:    If  $Val_p[k] = \perp \wedge (\exists \langle r_p, \Delta_q, q \rangle \in R_p[r_p], \Delta_q[k] \neq \perp)$  then
12:       $Val_p[k] \leftarrow \Delta_q[k]$ 
13:       $\Delta_p[k] \leftarrow \Delta_q[k]$ 
14:    End If
15:  Done
16: Done
17: broadcast  $\langle Val_p \rangle$  to  $V$  ( $p$  included)                        /* Phase 2 */
18: wait to receive  $\langle Val_q \rangle$  from  $q \in V \setminus S_p$ 
19: Let  $lastmsgs_p$  be the set of received  $\langle Val_q \rangle$ 
20: For all  $k \in V$  do
21:   If  $\exists Val_q \in lastmsgs_p, Val_q[k] = \perp$  then  $Val_p[k] \leftarrow \perp$ 
22: Done
23:  $d_p \leftarrow x$  where  $x$  is the first non- $\perp$  value in  $Val_p$     /* Phase 3 */
    
```

Termination (continued: Line 23 is a decision)

Lemma 6

$\forall p \in \Pi_2, Val_p[c] = v_c$ at the end of Phase 2.

Proof. From the code of the algorithm (Line 3), we know that $Val_c[c] = v_c$ at the end of Phase 1.

By Lemma 4, $\forall q \in \Pi_1, Val_q[c] = v_c$ at the end of Phase 1.

Thus, no process sends a vector V such that $V[c] = \perp$ during Phase 2.

□

```

1:  $d_p \leftarrow \perp$ 
2:  $Val_p \leftarrow [\perp, \dots, \perp]$ 
3:  $Val_p[\rho] \leftarrow v_\rho$ 
4:  $\Delta_p \leftarrow Val_p$ 
5: For all  $r_p$  from 1 to  $n-1$  do                                /* Phase 1 */
6:   broadcast  $\langle r_p, \Delta_p, \rho \rangle$  to  $V$  ( $p$  included)
7:   wait to receive  $\langle r_p, \Delta_q, q \rangle$  from  $q \in V \setminus S_p$ 
8:   let  $R_p[r_p]$  be the set of received  $\langle r_p, \Delta_q, q \rangle$ 
9:    $\Delta_p \leftarrow [\perp, \dots, \perp]$ 
10:  For all  $k \in V$  do
11:    If  $Val_p[k] = \perp \wedge (\exists \langle r_p, \Delta_q, q \rangle \in R_p[r_p], \Delta_q[k] \neq \perp)$  then
12:       $Val_p[k] \leftarrow \Delta_q[k]$ 
13:       $\Delta_p[k] \leftarrow \Delta_q[k]$ 
14:    End If
15:  Done
16: Done
17: broadcast  $\langle Val_p \rangle$  to  $V$  ( $p$  included)                        /* Phase 2 */
18: wait to receive  $\langle Val_q \rangle$  from  $q \in V \setminus S_p$ 
19: Let  $lastmsgs_p$  be the set of received  $\langle Val_q \rangle$ 
20: For all  $k \in V$  do
21:   If  $\exists Val_q \in lastmsgs_p, Val_q[k] = \perp$  then  $Val_p[k] \leftarrow \perp$ 
22: Done
23:  $d_p \leftarrow x$  where  $x$  is the first non- $\perp$  value in  $Val_p$     /* Phase 3 */
    
```

Termination (continued: Line 23 is a decision)

Lemma 6

$\forall p \in \Pi_2$, $Val_p[c] = v_c$ at the end of Phase 2.

Proof. From the code of the algorithm (Line 3), we know that $Val_c[c] = v_c$ at the end of Phase 1.

By Lemma 4, $\forall q \in \Pi_1$, $Val_q[c] = v_c$ at the end of Phase 1.

Thus, no process sends a vector V such that $V[c] = \perp$ during Phase 2.

Hence, from the code of the algorithm, we can deduce that $\forall p \in \Pi_2$,
 $Val_p[c] = v_c$ at the end of Phase 2. \square

```

1:  $d_p \leftarrow \perp$ 
2:  $Val_p \leftarrow [\perp, \dots, \perp]$ 
3:  $Val_p[p] \leftarrow v_p$ 
4:  $\Delta_p \leftarrow Val_p$ 
5: For all  $r_p$  from 1 to  $n-1$  do                                /* Phase 1 */
6:   broadcast  $\langle r_p, \Delta_p, p \rangle$  to  $V$  ( $p$  included)
7:   wait to receive  $\langle r_p, \Delta_q, q \rangle$  from  $q \in V \setminus S_p$ 
8:   let  $R_p[r_p]$  be the set of received  $\langle r_p, \Delta_q, q \rangle$ 
9:    $\Delta_p \leftarrow [\perp, \dots, \perp]$ 
10:  For all  $k \in V$  do
11:    If  $Val_p[k] = \perp \wedge (\exists \langle r_p, \Delta_q, q \rangle \in R_p[r_p], \Delta_q[k] \neq \perp)$  then
12:       $Val_p[k] \leftarrow \Delta_q[k]$ 
13:       $\Delta_p[k] \leftarrow \Delta_q[k]$ 
14:    End If
15:  Done
16: Done
17: broadcast  $\langle Val_p \rangle$  to  $V$  ( $p$  included)                        /* Phase 2 */
18: wait to receive  $\langle Val_q \rangle$  from  $q \in V \setminus S_p$ 
19: Let  $lastmsgs_p$  be the set of received  $\langle Val_q \rangle$ 
20: For all  $k \in V$  do
21:   If  $\exists Val_q \in lastmsgs_p, Val_q[k] = \perp$  then  $Val_p[k] \leftarrow \perp$ 
22: Done
23:  $d_p \leftarrow x$  where  $x$  is the first non- $\perp$  value in  $Val_p$     /* Phase 3 */

```

Result

Theorem 1

The algorithm of Chandra and Toueg solves the consensus in an asynchronous system enriched with a failure detector \mathcal{S} .

Proof.

```

1:  $d_p \leftarrow \perp$ 
2:  $Val_p \leftarrow [\perp, \dots, \perp]$ 
3:  $Val_p[\rho] \leftarrow v_\rho$ 
4:  $\Delta_p \leftarrow Val_p$ 
5: For all  $r_\rho$  from 1 to  $n-1$  do /* Phase 1 */
6:   broadcast  $\langle r_\rho, \Delta_p, \rho \rangle$  to  $V$  ( $p$  included)
7:   wait to receive  $\langle r_\rho, \Delta_q, q \rangle$  from  $q \in V \setminus S_p$ 
8:   let  $R_\rho[r_\rho]$  be the set of received  $\langle r_\rho, \Delta_q, q \rangle$ 
9:    $\Delta_p \leftarrow [\perp, \dots, \perp]$ 
10:  For all  $k \in V$  do
11:    If  $Val_p[k] = \perp \wedge (\exists (r_\rho, \Delta_q, q) \in R_\rho[r_\rho], \Delta_q[k] \neq \perp)$  then
12:       $Val_p[k] \leftarrow \Delta_q[k]$ 
13:       $\Delta_p[k] \leftarrow \Delta_q[k]$ 
14:    End If
15:  Done
16: Done
17: broadcast  $\langle Val_p \rangle$  to  $V$  ( $p$  included) /* Phase 2 */
18: wait to receive  $\langle Val_q \rangle$  from  $q \in V \setminus S_p$ 
19: Let  $lastmsgs_p$  be the set of received  $\langle Val_q \rangle$ 
20: For all  $k \in V$  do
21:   If  $\exists Val_q \in lastmsgs_p, Val_q[k] = \perp$  then  $Val_p[k] \leftarrow \perp$ 
22: Done
23:  $d_p \leftarrow x$  where  $x$  is the first non- $\perp$  value in  $Val_p$  /* Phase 3 */

```

□

Result

Theorem 1

The algorithm of Chandra and Toueg solves the consensus in an asynchronous system enriched with a failure detector \mathcal{S} .

Proof.

- **Integrity:** from the code of the algorithm, Phase 3 is executed only once.

□

```

1:  $d_p \leftarrow \perp$ 
2:  $Val_p \leftarrow [\perp, \dots, \perp]$ 
3:  $Val_p[p] \leftarrow v_p$ 
4:  $\Delta_p \leftarrow Val_p$ 
5: For all  $r_p$  from 1 to  $n-1$  do                                     /* Phase 1 */
6:   broadcast  $\langle r_p, \Delta_p, p \rangle$  to  $V$  ( $p$  included)
7:   wait to receive  $\langle r_p, \Delta_q, q \rangle$  from  $q \in V \setminus S_p$ 
8:   let  $R_p[r_p]$  be the set of received  $\langle r_p, \Delta_q, q \rangle$ 
9:    $\Delta_p \leftarrow [\perp, \dots, \perp]$ 
10:  For all  $k \in V$  do
11:    If  $Val_p[k] = \perp \wedge (\exists (r_p, \Delta_q, q) \in R_p[r_p], \Delta_q[k] \neq \perp)$  then
12:       $Val_p[k] \leftarrow \Delta_q[k]$ 
13:       $\Delta_p[k] \leftarrow \Delta_q[k]$ 
14:    End If
15:  Done
16: Done
17: broadcast  $\langle Val_p \rangle$  to  $V$  ( $p$  included)                             /* Phase 2 */
18: wait to receive  $\langle Val_q \rangle$  from  $q \in V \setminus S_p$ 
19: Let  $lastmsgs_p$  be the set of received  $\langle Val_q \rangle$ 
20: For all  $k \in V$  do
21:   If  $\exists Val_q \in lastmsgs_p, Val_q[k] = \perp$  then  $Val_p[k] \leftarrow \perp$ 
22: Done
23:  $d_p \leftarrow x$  where  $x$  is the first non- $\perp$  value in  $Val_p$        /* Phase 3 */

```

Result

Theorem 1

The algorithm of Chandra and Toueg solves the consensus in an asynchronous system enriched with a failure detector \mathcal{S} .

Proof.

- **Integrity:** from the code of the algorithm, Phase 3 is executed only once.
- **Termination:** Line 23 is a decision (Lemma 6), now every correct eventually executes Line 23 (Lemma 2).
- **Agreement:** Corollary 1.
- **Validity:** Lemma 1.

□

```

1:  $d_p \leftarrow \perp$ 
2:  $Val_p \leftarrow [\perp, \dots, \perp]$ 
3:  $Val_p[\rho] \leftarrow v_p$ 
4:  $\Delta_p \leftarrow Val_p$ 
5: For all  $r_p$  from 1 to  $n-1$  do                                /* Phase 1 */
6:   broadcast  $\langle r_p, \Delta_p, \rho \rangle$  to  $V$  ( $p$  included)
7:   wait to receive  $\langle r_p, \Delta_q, q \rangle$  from  $q \in V \setminus S_p$ 
8:   let  $R_p[r_p]$  be the set of received  $\langle r_p, \Delta_q, q \rangle$ 
9:    $\Delta_p \leftarrow [\perp, \dots, \perp]$ 
10:  For all  $k \in V$  do
11:    If  $Val_p[k] = \perp \wedge (\exists (r_p, \Delta_q, q) \in R_p[r_p], \Delta_q[k] \neq \perp)$  then
12:       $Val_p[k] \leftarrow \Delta_q[k]$ 
13:       $\Delta_p[k] \leftarrow \Delta_q[k]$ 
14:    End If
15:  Done
16: Done
17: broadcast  $\langle Val_p \rangle$  to  $V$  ( $p$  included)                        /* Phase 2 */
18: wait to receive  $\langle Val_q \rangle$  from  $q \in V \setminus S_p$ 
19: Let  $lastmsgs_p$  be the set of received  $\langle Val_q \rangle$ 
20: For all  $k \in V$  do
21:   If  $\exists Val_q \in lastmsgs_p, Val_q[k] = \perp$  then  $Val_p[k] \leftarrow \perp$ 
22: Done
23:  $d_p \leftarrow x$  where  $x$  is the first non- $\perp$  value in  $Val_p$     /* Phase 3 */

```

Roadmap

- 1 Introduction
- 2 Definition
- 3 Application: a Consensus Algorithm
 - The Problem
 - The Model
 - The Algorithm
 - The Proof
- 4 Implementation of a Failure Detector (an example)
 - $\diamond\mathcal{P}$
 - The Model
 - The Algorithm
 - The Proof
- 5 References

A Simple Example: $\diamond\mathcal{P}$

Every failure detector of the class $\diamond\mathcal{P}$ satisfies both **strong completeness** and **eventual strong accuracy**:

Strong Completeness: Every faulty process is eventually permanently suspected by **every** correct process.

Eventual Strong Accuracy: **There is a time after which** no correct process is suspected by any correct process.

System Assumptions

Motivation

Given a failure detector of class $\diamond\mathcal{P}$, consensus can be solved in an asynchronous crash-prone system with reliable links where a majority of processes is correct.⁸

⁸In these settings, consensus can be even solved with a weaker failure detector (Ω). This fact will be established in the next lesson.

System Assumptions

Motivation

Given a failure detector of class $\diamond\mathcal{P}$, consensus can be solved in an asynchronous crash-prone system with reliable links where a majority of processes is correct.⁸

The FLP implies that partial synchrony assumptions are required to implement such a failure detector!

⁸In these settings, consensus can be even solved with a weaker failure detector (Ω). This fact will be established in the next lesson.

System Assumptions

The Partially Synchronous System $S_{\diamond b}$ (1/2)

- Complete Network Topology

System Assumptions

The Partially Synchronous System $S_{\diamond b}$ (1/2)

- Complete Network Topology
- Process failures: only crashes!

System Assumptions

The Partially Synchronous System $S_{\diamond b}$ (1/2)

- Complete Network Topology
- Process failures: only crashes!
- Correct processes are **eventually synchronous**: for every correct process p , there exists a time i_p (**a priori unknown** by **all** processes) from which p executes each of its instructions in a time belonging to $[\alpha_p.. \beta_p]$ with $0 < \alpha_p \leq \beta_p$.

α_p and β_p are **a priori unknown by all processes**, for every process p .

System Assumptions

The Partially Synchronous System $S_{\diamond b}$ (2/2)

- There exists at least one **eventual bi-source** $\diamond b$ (**a priori unknown** by processes): a correct process with all its outgoing and incoming links that are eventually reliable and synchronous, *i.e.*,
 there exists a time $t_{\diamond b}$ from which every message sent to or from $\diamond b$ is delivered within at most $\delta_{\diamond b}$ time units.
 $\diamond b$, $t_{\diamond b}$, and $\delta_{\diamond b}$ are **a priori unknown** (even by $\diamond b$!)

System Assumptions

The Partially Synchronous System $\mathcal{S}_{\diamond b}$ (2/2)

- There exists at least one **eventual bi-source $\diamond b$ (a priori unknown** by processes): a correct process with all its outgoing and incoming links that are eventually reliable and synchronous, *i.e.*,
 there exists a time $t_{\diamond b}$ from which every message sent to or from $\diamond b$ is delivered within at most $\delta_{\diamond b}$ time units.
 $\diamond b$, $t_{\diamond b}$, and $\delta_{\diamond b}$ are **a priori unknown** (even by $\diamond b$!)

Every other link is arbitrary slow and lossy: **only $2n - 2$ unidirectional links are guaranteed to be eventually reliable and synchronous among the $n(n - 1)$ possible ones!**

Recall that every message is delivered or lost within finite time.

Principles

- 1 Each process regularly broadcasts ALIVE messages tagged with its ID

Each message is **relayed once**.

This way, $\diamond b$ acts as a **hub**: eventually messages tagged with IDs of correct processes are delivered within bounded time.

Principles

- 1 Each process regularly broadcasts ALIVE messages tagged with its ID

Each message is **relayed once**.

This way, $\diamond b$ acts as a **hub**: **eventually messages tagged with IDs of correct processes are delivered within bounded time**.

- 2 Each process maintains a **timer** for each other process.

On Time Out: the watched process is **suspected**.

If later, the process receives a message tagged with the ID of some suspected process, **it stops suspecting it and increases the waiting time of the associated timer**.

Constants & Variables

- Processes are identified: a process and its identifier are used equivalently
- *ALIVE*: message type
- *V*: set of processes
- $k \in \mathbb{N}^*$: constant
- *Timer_p*[], *ElapsedTime_p*[]): arrays of integer indexed on the process IDs, except *p*.
- *Alive_p*, *Suspected_p*: sets of identifiers
(*Suspected_p* is the algorithm output)

Algorithm *EP*

Algorithm *EP* for each process p , output: $Suspected_p$

```

1:  $Alive_p \leftarrow V$ ;  $Suspected_p \leftarrow \emptyset$ 
2:  $Timer_p \leftarrow [k, \dots, k]$ ;  $ElapseTime_p \leftarrow [k, \dots, k]$ 
3: While true do
4:   broadcast  $\langle ALIVE, p, p \rangle$  to  $V \setminus \{p\}$ 
5:   For all  $q \in V \setminus \{p\}$  do
6:     If receive  $\langle ALIVE, r, q \rangle$  then
7:       If  $r \neq p$  then
8:         If  $ElapseTime_p[r] \leq 0$  then  $Timer_p[r]++$ 
9:          $ElapseTime_p[r] \leftarrow Timer_p[r]$ 
10:         $Alive_p \leftarrow Alive_p \cup \{r\}$ 
11:        If  $r = q$  then broadcast  $\langle ALIVE, r, p \rangle$  to  $V \setminus \{p\}$ 
12:      End If
13:    End If
14:  Done
15:  For all  $q \in V \setminus \{p\}$  do
16:    If  $ElapseTime_p[q] = 0$  then
17:       $Alive_p \leftarrow Alive_p \setminus \{q\}$ 
18:    else
19:       $ElapseTime_p[q]--$ 
20:    End If
21:  Done
22:   $Suspected_p \leftarrow V \setminus Alive_p$ 
23: Done

```

Eventual Suspicion of Faulty Processes (1/3)

Recall that the wildcard “_” designates any value.

Lemma 1

Each correct process p eventually no more receives $\langle ALIVE, q, _ \rangle$ where q is a faulty process.

Proof.

```

1:  $Alive_p \leftarrow V; Suspected_p \leftarrow \emptyset$ 
2:  $Timer_p \leftarrow [k, \dots, k]; ElapseTime_p \leftarrow [k, \dots, k]$ 
3: While true do
4:   broadcast  $\langle ALIVE, p, p \rangle$  to  $V \setminus \{p\}$ 
5:   For all  $q \in V \setminus \{p\}$  do
6:     If receive  $\langle ALIVE, r, q \rangle$  then
7:       If  $r \neq p$  then
8:         If  $ElapseTime_p[r] \leq 0$  then  $Timer_p[r] ++$ 
9:            $ElapseTime_p[r] \leftarrow Timer_p[r]$ 
10:           $Alive_p \leftarrow Alive_p \cup \{r\}$ 
11:          If  $r = q$  then broadcast  $\langle ALIVE, r, p \rangle$  to  $V \setminus \{p\}$ 
12:        End If
13:      End If
14:    Done
15:    For all  $q \in V \setminus \{p\}$  do
16:      If  $ElapseTime_p[q] = 0$  then
17:         $Alive_p \leftarrow Alive_p \setminus \{q\}$ 
18:      else
19:         $ElapseTime_p[q] --$ 
20:      End If
21:    Done
22:     $Suspected_p \leftarrow V \setminus Alive_p$ 
23: Done

```

□

Eventual Suspicion of Faulty Processes (1/3)

Recall that the wildcard “_” designates any value.

Lemma 1

Each correct process p eventually no more receives $\langle ALIVE, q, _ \rangle$ where q is a faulty process.

Proof. q broadcasts **finitely many** $\langle ALIVE, q, q \rangle$ messages before crashing.

```

1:  $Alive_p \leftarrow V; Suspected_p \leftarrow \emptyset$ 
2:  $Timer_p \leftarrow [k, \dots, k]; ElapseTime_p \leftarrow [k, \dots, k]$ 
3: While true do
4:   broadcast  $\langle ALIVE, p, p \rangle$  to  $V \setminus \{p\}$ 
5:   For all  $q \in V \setminus \{p\}$  do
6:     If receive  $\langle ALIVE, r, q \rangle$  then
7:       If  $r \neq p$  then
8:         If  $ElapseTime_p[r] \leq 0$  then  $Timer_p[r] ++$ 
9:            $ElapseTime_p[r] \leftarrow Timer_p[r]$ 
10:           $Alive_p \leftarrow Alive_p \cup \{r\}$ 
11:          If  $r = q$  then broadcast  $\langle ALIVE, r, p \rangle$  to  $V \setminus \{p\}$ 
12:        End If
13:      End If
14:    Done
15:    For all  $q \in V \setminus \{p\}$  do
16:      If  $ElapseTime_p[q] = 0$  then
17:         $Alive_p \leftarrow Alive_p \setminus \{q\}$ 
18:      else
19:         $ElapseTime_p[q] --$ 
20:      End If
21:    Done
22:     $Suspected_p \leftarrow V \setminus Alive_p$ 
23: Done

```

□

Eventual Suspicion of Faulty Processes (1/3)

Recall that the wildcard “_” designates any value.

Lemma 1

Each correct process p eventually no more receives $\langle ALIVE, q, _ \rangle$ where q is a faulty process.

Proof. q broadcasts **finitely many** $\langle ALIVE, q, q \rangle$ messages before crashing.

Now, each $\langle ALIVE, q, q \rangle$ message is **relayed at most once** by every other processes.

□

```

1:  $Alive_p \leftarrow V; Suspected_p \leftarrow \emptyset$ 
2:  $Timer_p \leftarrow [k, \dots, k]; ElapseTime_p \leftarrow [k, \dots, k]$ 
3: While true do
4:   broadcast  $\langle ALIVE, p, p \rangle$  to  $V \setminus \{p\}$ 
5:   For all  $q \in V \setminus \{p\}$  do
6:     If receive  $\langle ALIVE, r, q \rangle$  then
7:       If  $r \neq p$  then
8:         If  $ElapseTime_p[r] \leq 0$  then  $Timer_p[r] ++$ 
9:            $ElapseTime_p[r] \leftarrow Timer_p[r]$ 
10:           $Alive_p \leftarrow Alive_p \cup \{r\}$ 
11:          If  $r = q$  then broadcast  $\langle ALIVE, r, p \rangle$  to  $V \setminus \{p\}$ 
12:        End If
13:      End If
14:    Done
15:    For all  $q \in V \setminus \{p\}$  do
16:      If  $ElapseTime_p[q] = 0$  then
17:         $Alive_p \leftarrow Alive_p \setminus \{q\}$ 
18:      else
19:         $ElapseTime_p[q] --$ 
20:      End If
21:    Done
22:     $Suspected_p \leftarrow V \setminus Alive_p$ 
23: Done

```

Eventual Suspicion of Faulty Processes (1/3)

Recall that the wildcard “_” designates any value.

Lemma 1

Each correct process p eventually no more receives $\langle ALIVE, q, _ \rangle$ where q is a faulty process.

Proof. q broadcasts **finitely many** $\langle ALIVE, q, q \rangle$ messages before crashing.

Now, each $\langle ALIVE, q, q \rangle$ message is **relayed at most once** by every other processes.

As every sent message is **eventually either received or lost**, the lemma holds. \square

```

1:  $Alive_p \leftarrow V; Suspected_p \leftarrow \emptyset$ 
2:  $Timer_p \leftarrow [k, \dots, k]; ElapseTime_p \leftarrow [k, \dots, k]$ 
3: While true do
4:   broadcast  $\langle ALIVE, p, p \rangle$  to  $V \setminus \{p\}$ 
5:   For all  $q \in V \setminus \{p\}$  do
6:     If receive  $\langle ALIVE, r, q \rangle$  then
7:       If  $r \neq p$  then
8:         If  $ElapseTime_p[r] \leq 0$  then  $Timer_p[r] ++$ 
9:            $ElapseTime_p[r] \leftarrow Timer_p[r]$ 
10:           $Alive_p \leftarrow Alive_p \cup \{r\}$ 
11:          If  $r = q$  then broadcast  $\langle ALIVE, r, p \rangle$  to  $V \setminus \{p\}$ 
12:        End If
13:      End If
14:    Done
15:    For all  $q \in V \setminus \{p\}$  do
16:      If  $ElapseTime_p[q] = 0$  then
17:         $Alive_p \leftarrow Alive_p \setminus \{q\}$ 
18:      else
19:         $ElapseTime_p[q] --$ 
20:      End If
21:    Done
22:     $Suspected_p \leftarrow V \setminus Alive_p$ 
23: Done

```

Eventual Suspicion of Faulty Processes (2/3)

Lemma 2

Let p be a correct process. $\exists t_p, K_p, K'_p \in \mathbb{N}$ with $K_p \geq K'_p$ such that after time t_p , each iteration of the “while” loop by p lasts a time belonging to $[K'_p, K_p]$.

Proof.

By definition, \exists a time t from which each instruction is executed by p within a time that is both lower and upper bounded since p is eventually synchronous.

□

Eventual Suspicion of Faulty Processes (2/3)

Lemma 2

Let p be a correct process. $\exists t_p, K_p, K'_p \in \mathbb{N}$ with $K_p \geq K'_p$ such that after time t_p , each iteration of the “while” loop by p lasts a time belonging to $[K'_p, K_p]$.

Proof.

By definition, \exists a time t from which each instruction is executed by p within a time that is both lower and upper bounded since p is eventually synchronous.

There is a finite number of instructions before the “while” loop. So, in the worst case, the “while” loop begins within bounded time t_p after t .

□

Eventual Suspicion of Faulty Processes (2/3)

Lemma 2

Let p be a correct process. $\exists t_p, K_p, K'_p \in \mathbb{N}$ with $K_p \geq K'_p$ such that after time t_p , each iteration of the “while” loop by p lasts a time belonging to $[K'_p, K_p]$.

Proof.

By definition, \exists a time t from which each instruction is executed by p within a time that is both lower and upper bounded since p is eventually synchronous.

There is a finite number of instructions before the “while” loop. So, in the worst case, the “while” loop begins within bounded time t_p after t .

Similarly, after t_p , $\exists K_p, K'_p \in \mathbb{N}$ with $K'_p \leq K_p$ such that each iteration of the “while” loop by p lasts a time belonging to $[K'_p, K_p]$ since the “while” loop contains a bounded number of instructions and the time to execute any instruction is both lower and upper bounded.

□

Eventual Suspicion of Faulty Processes (3/3)

Corollary 1

Every faulty process is eventually forever suspected by **every** correct process.

Proof. Let q be a faulty process and p be a correct process.

```

1:  $Alive_p \leftarrow V$ ;  $Suspected_p \leftarrow \emptyset$ 
2:  $Timer_p \leftarrow [k, \dots, k]$ ;  $ElapseTime_p \leftarrow [k, \dots, k]$ 
3: While true do
4:   broadcast  $\langle ALIVE, p, p \rangle$  to  $V \setminus \{p\}$ 
5:   For all  $q \in V \setminus \{p\}$  do
6:     If receive  $\langle ALIVE, r, q \rangle$  then
7:       If  $r \neq p$  then
8:         If  $ElapseTime_p[r] \leq 0$  then  $Timer_p[r] ++$ 
9:            $ElapseTime_p[r] \leftarrow Timer_p[r]$ 
10:         $Alive_p \leftarrow Alive_p \cup \{r\}$ 
11:        If  $r = q$  then broadcast  $\langle ALIVE, r, p \rangle$  to  $V \setminus \{p\}$ 
12:      End If
13:    End If
14:  Done
15:  For all  $q \in V \setminus \{p\}$  do
16:    If  $ElapseTime_p[q] = 0$  then
17:       $Alive_p \leftarrow Alive_p \setminus \{q\}$ 
18:    else
19:       $ElapseTime_p[q] --$ 
20:    End If
21:  Done
22:   $Suspected_p \leftarrow V \setminus Alive_p$ 
23: Done
    
```

□

Eventual Suspicion of Faulty Processes (3/3)

Corollary 1

Every faulty process is eventually forever suspected by **every** correct process.

Proof. Let q be a faulty process and p be a correct process.

By Lemma 2, there exists a time t_p from which p executes a “while” loop iteration within at most K_p time units.

```

1:  $Alive_p \leftarrow V$ ;  $Suspected_p \leftarrow \emptyset$ 
2:  $Timer_p \leftarrow [k, \dots, k]$ ;  $ElapseTime_p \leftarrow [k, \dots, k]$ 
3: While true do
4:   broadcast  $\langle ALIVE, p, p \rangle$  to  $V \setminus \{p\}$ 
5:   For all  $q \in V \setminus \{p\}$  do
6:     If receive  $\langle ALIVE, r, q \rangle$  then
7:       If  $r \neq p$  then
8:         If  $ElapseTime_p[r] \leq 0$  then  $Timer_p[r] ++$ 
9:            $ElapseTime_p[r] \leftarrow Timer_p[r]$ 
10:         $Alive_p \leftarrow Alive_p \cup \{r\}$ 
11:        If  $r = q$  then broadcast  $\langle ALIVE, r, p \rangle$  to  $V \setminus \{p\}$ 
12:      End If
13:    End If
14:  Done
15:  For all  $q \in V \setminus \{p\}$  do
16:    If  $ElapseTime_p[q] = 0$  then
17:       $Alive_p \leftarrow Alive_p \setminus \{q\}$ 
18:    else
19:       $ElapseTime_p[q] --$ 
20:    End If
21:  Done
22:   $Suspected_p \leftarrow V \setminus Alive_p$ 
23: Done
    
```

□

Eventual Suspicion of Faulty Processes (3/3)

Corollary 1

Every faulty process is eventually forever suspected by **every** correct process.

Proof. Let q be a faulty process and p be a correct process.

By Lemma 2, there exists a time t_p from which p executes a “while” loop iteration within at most K_p time units.

So, from time t_p , p eventually satisfies $ElapseTime_p[q] = 0$ forever, by Lemma 1.

```

1:  $Alive_p \leftarrow V$ ;  $Suspected_p \leftarrow \emptyset$ 
2:  $Timer_p \leftarrow [k, \dots, k]$ ;  $ElapseTime_p \leftarrow [k, \dots, k]$ 
3: While true do
4:   broadcast  $\langle ALIVE, p, p \rangle$  to  $V \setminus \{p\}$ 
5:   For all  $q \in V \setminus \{p\}$  do
6:     If receive  $\langle ALIVE, r, q \rangle$  then
7:       If  $r \neq p$  then
8:         If  $ElapseTime_p[r] \leq 0$  then  $Timer_p[r] ++$ 
9:          $ElapseTime_p[r] \leftarrow Timer_p[r]$ 
10:         $Alive_p \leftarrow Alive_p \cup \{r\}$ 
11:        If  $r = q$  then broadcast  $\langle ALIVE, r, p \rangle$  to  $V \setminus \{p\}$ 
12:      End If
13:    End If
14:  Done
15:  For all  $q \in V \setminus \{p\}$  do
16:    If  $ElapseTime_p[q] = 0$  then
17:       $Alive_p \leftarrow Alive_p \setminus \{q\}$ 
18:    else
19:       $ElapseTime_p[q] --$ 
20:    End If
21:  Done
22:   $Suspected_p \leftarrow V \setminus Alive_p$ 
23: Done
  
```

□

Eventual Suspicion of Faulty Processes (3/3)

Corollary 1

Every faulty process is eventually forever suspected by **every** correct process.

Proof. Let q be a faulty process and p be a correct process.

By Lemma 2, there exists a time t_p from which p executes a “while” loop iteration within at most K_p time units.

So, from time t_p , p eventually satisfies $ElapseTime_p[q] = 0$ forever, by Lemma 1.

Hence, q is eventually removed from $Alive_p$ forever and so eventually belongs forever to $Suspected_p$. □

```

1:  $Alive_p \leftarrow V$ ;  $Suspected_p \leftarrow \emptyset$ 
2:  $Timer_p \leftarrow [k, \dots, k]$ ;  $ElapseTime_p \leftarrow [k, \dots, k]$ 
3: While true do
4:   broadcast  $\langle ALIVE, p, p \rangle$  to  $V \setminus \{p\}$ 
5:   For all  $q \in V \setminus \{p\}$  do
6:     If receive  $\langle ALIVE, r, q \rangle$  then
7:       If  $r \neq p$  then
8:         If  $ElapseTime_p[r] \leq 0$  then  $Timer_p[r] ++$ 
9:          $ElapseTime_p[r] \leftarrow Timer_p[r]$ 
10:         $Alive_p \leftarrow Alive_p \cup \{r\}$ 
11:        If  $r = q$  then broadcast  $\langle ALIVE, r, p \rangle$  to  $V \setminus \{p\}$ 
12:      End If
13:    End If
14:  Done
15:  For all  $q \in V \setminus \{p\}$  do
16:    If  $ElapseTime_p[q] = 0$  then
17:       $Alive_p \leftarrow Alive_p \setminus \{q\}$ 
18:    else
19:       $ElapseTime_p[q] --$ 
20:    End If
21:  Done
22:   $Suspected_p \leftarrow V \setminus Alive_p$ 
23: Done

```

Unsuspection of Correct Processes (1/4)

Lemma 3

Let p and q be two correct processes such that $p \neq q$. There exists $t_p, \Delta_p \in \mathbb{N}$ such that after time t_p , p receives $\langle ALIVE, q, _ \rangle$ at least every Δ_p time units.

Proof. Let $\diamond b$ be an eventual bi-source.

Unsuspection of Correct Processes (1/4)

Lemma 3

Let p and q be two correct processes such that $p \neq q$. There exists $t_p, \Delta_p \in \mathbb{N}$ such that after time t_p , p receives $\langle ALIVE, q, _ \rangle$ at least every Δ_p time units.

Proof. Let $\diamond b$ be an eventual bi-source.

By Lemma 2 and by definition of eventual bi-source, there exists $t \in \mathbb{N}$ such that after time t ,

- q , $\diamond b$, and p respectively execute each “while” loop iteration within at most K_q , $K_{\diamond b}$, and K_p time units.
- Moreover, every message sent from or to $\diamond b$ is delivered in at most $\delta_{\diamond b}$ time units.

Unsuspection of Correct Processes (1/4)

Lemma 3

Let p and q be two correct processes such that $p \neq q$. There exists $t_p, \Delta_p \in \mathbb{N}$ such that after time t_p , p receives $\langle ALIVE, q, _ \rangle$ at least every Δ_p time units.

Proof. Let $\diamond b$ be an eventual bi-source.

By Lemma 2 and by definition of eventual bi-source, there exists $t \in \mathbb{N}$ such that after time t ,

- q , $\diamond b$, and p respectively execute each “while” loop iteration within at most K_q , $K_{\diamond b}$, and K_p time units.
- Moreover, every message sent from or to $\diamond b$ is delivered in at most $\delta_{\diamond b}$ time units.

Two cases: $q = \diamond b$ or $q \neq \diamond b$.

Unsuspection of Correct Processes (2/4)

Case $q = \diamond b$

$\exists t' \in [t..t + K_q]$ such that q starts a “while” loop iteration at time t' . From t' , q executes a (full) “while” loop iteration at least every K_q time units. So, q broadcasts $\langle ALIVE, q, q \rangle$ at least every K_q time units from t' .

Unsuspection of Correct Processes (2/4)

Case $q = \diamond b$

$\exists t' \in [t..t + K_q]$ such that q starts a “while” loop iteration at time t' . From t' , q executes a (full) “while” loop iteration at least every K_q time units. So, q broadcasts $\langle ALIVE, q, q \rangle$ at least every K_q time units from t' .

These messages are delivered to p within at most $\delta_{\diamond b}$ time units after their sending.

Unsuspection of Correct Processes (2/4)

Case $q = \diamond b$

$\exists t' \in [t..t + K_q]$ such that q starts a “while” loop iteration at time t' . From t' , q executes a (full) “while” loop iteration at least every K_q time units. So, q broadcasts $\langle ALIVE, q, q \rangle$ at least every K_q time units from t' .

These messages are delivered to p within at most $\delta_{\diamond b}$ time units after their sending.

Now, p executes a full “while” loop iteration at least every $2.K_p$ time units.

Hence, with $\Delta_p = K_q + \delta_{\diamond b} + 2.K_p$, the lemma holds in this case.

Unsuspection of Correct Processes (3/4)

Case $q \neq \diamond b$

Similarly to the previous case: $\exists t' \in [t..t + K_q]$ such that q broadcasts $\langle ALIVE, q, q \rangle$ at least every K_q time units from t' .

□

Unsuspection of Correct Processes (3/4)

Case $q \neq \diamond b$

Similarly to the previous case: $\exists t' \in [t..t + K_q]$ such that q broadcasts $\langle ALIVE, q, q \rangle$ at least every K_q time units from t' .

These messages are delivered to $\diamond b$ within at most $\delta_{\diamond b}$ time units after their sending; so at least every $K_q + \delta_{\diamond b}$ time units.

□

Unsuspection of Correct Processes (3/4)

Case $q \neq \diamond b$

Similarly to the previous case: $\exists t' \in [t..t + K_q]$ such that q broadcasts $\langle ALIVE, q, q \rangle$ at least every K_q time units from t' .

These messages are delivered to $\diamond b$ within at most $\delta_{\diamond b}$ time units after their sending; so at least every $K_q + \delta_{\diamond b}$ time units.

$\diamond b$ sends $\langle ALIVE, q, \diamond b \rangle$ to p at least every $K_q + \delta_{\diamond b} + 2.K_{\diamond b}$ time units.

□

Unsuspection of Correct Processes (3/4)

Case $q \neq \diamond b$

Similarly to the previous case: $\exists t' \in [t..t + K_q]$ such that q broadcasts $\langle ALIVE, q, q \rangle$ at least every K_q time units from t' .

These messages are delivered to $\diamond b$ within at most $\delta_{\diamond b}$ time units after their sending; so at least every $K_q + \delta_{\diamond b}$ time units.

$\diamond b$ sends $\langle ALIVE, q, \diamond b \rangle$ to p at least every $K_q + \delta_{\diamond b} + 2.K_{\diamond b}$ time units.

$\langle ALIVE, q, _ \rangle$ messages are delivered to p at least $K_q + 2.\delta_{\diamond b} + 2.K_{\diamond b}$ time units.

□

Unsuspection of Correct Processes (3/4)

Case $q \neq \diamond b$

Similarly to the previous case: $\exists t' \in [t..t + K_q]$ such that q broadcasts $\langle ALIVE, q, q \rangle$ at least every K_q time units from t' .

These messages are delivered to $\diamond b$ within at most $\delta_{\diamond b}$ time units after their sending; so at least every $K_q + \delta_{\diamond b}$ time units.

$\diamond b$ sends $\langle ALIVE, q, \diamond b \rangle$ to p at least every $K_q + \delta_{\diamond b} + 2.K_{\diamond b}$ time units.

$\langle ALIVE, q, _ \rangle$ messages are delivered to p at least $K_q + 2.\delta_{\diamond b} + 2.K_{\diamond b}$ time units.

Since p executes a full “while” loop iteration at least every $2.K_p$ time units, by letting $\Delta_p = K_q + 2.\delta_{\diamond b} + 2.K_{\diamond b} + 2.K_p$, the lemma holds in this case. \square

Unsuspicion of Correct Processes (4/4)

Corollary 2

There exists a time from which **no** correct process is suspected by any correct process.

Proof.

Let p and q be two correct processes.

```

1:  $Alive_p \leftarrow V$ ;  $Suspected_p \leftarrow \emptyset$ 
2:  $Timer_p \leftarrow [k, \dots, k]$ ;  $ElapseTime_p \leftarrow [k, \dots, k]$ 
3: While true do
4:   broadcast  $\langle ALIVE, p, p \rangle$  to  $V \setminus \{p\}$ 
5:   For all  $q \in V \setminus \{p\}$  do
6:     if receive  $\langle ALIVE, r, q \rangle$  then
7:       if  $r \neq p$  then
8:         if  $ElapseTime_p[r] \leq 0$  then  $Timer_p[r] ++$ 
9:            $ElapseTime_p[r] \leftarrow Timer_p[r]$ 
10:         $Alive_p \leftarrow Alive_p \cup \{r\}$ 
11:        if  $r = q$  then broadcast  $\langle ALIVE, r, p \rangle$  to  $V \setminus \{p\}$ 
12:      End if
13:    End if
14:  Done
15:  For all  $q \in V \setminus \{p\}$  do
16:    if  $ElapseTime_p[q] = 0$  then
17:       $Alive_p \leftarrow Alive_p \setminus \{q\}$ 
18:    else
19:       $ElapseTime_p[q] --$ 
20:    End if
21:  Done
22:   $Suspected_p \leftarrow V \setminus Alive_p$ 
23: Done
    
```

□

Unsuspection of Correct Processes (4/4)

Corollary 2

There exists a time from which **no** correct process is suspected by any correct process.

Proof.

Let p and q be two correct processes.

- If $p = q$, q is never removed from $Alive_p$ and so never inserted into $Suspected_p$. Hence, the lemma holds in this case.

```

1:  $Alive_p \leftarrow V; Suspected_p \leftarrow \emptyset$ 
2:  $Timer_p \leftarrow [k, \dots, k]; ElapseTime_p \leftarrow [k, \dots, k]$ 
3: While true do
4:   broadcast( $ALIVE, p, p$ ) to  $V \setminus \{p\}$ 
5:   For all  $q \in V \setminus \{p\}$  do
6:     if receive ( $ALIVE, r, q$ ) then
7:       if  $r \neq p$  then
8:         if  $ElapseTime_p[r] \leq 0$  then  $Timer_p[r] ++$ 
9:            $ElapseTime_p[r] \leftarrow Timer_p[r]$ 
10:         $Alive_p \leftarrow Alive_p \cup \{r\}$ 
11:        if  $r = q$  then broadcast( $ALIVE, r, p$ ) to  $V \setminus \{p\}$ 
12:      End if
13:    End if
14:  Done
15:  For all  $q \in V \setminus \{p\}$  do
16:    if  $ElapseTime_p[q] = 0$  then
17:       $Alive_p \leftarrow Alive_p \setminus \{q\}$ 
18:    else
19:       $ElapseTime_p[q] --$ 
20:    End if
21:  Done
22:   $Suspected_p \leftarrow V \setminus Alive_p$ 
23: Done

```

□

Unsuspicion of Correct Processes (4/4)

Corollary 2

There exists a time from which **no** correct process is suspected by any correct process.

Proof.

Let p and q be two correct processes.

- If $p = q$, q is never removed from $Alive_p$ and so never inserted into $Suspected_p$. Hence, the lemma holds in this case.
- Otherwise, q is regularly inserted into $Alive_p$, by Lemma 3.

```

1:  $Alive_p \leftarrow V; Suspected_p \leftarrow \emptyset$ 
2:  $Timer_p \leftarrow [k, \dots, k]; ElapseTime_p \leftarrow [k, \dots, k]$ 
3: While true do
4:   broadcast( $ALIVE, p, p$ ) to  $V \setminus \{p\}$ 
5:   For all  $q \in V \setminus \{p\}$  do
6:     if receive ( $ALIVE, r, q$ ) then
7:       if  $r \neq p$  then
8:         if  $ElapseTime_p[r] \leq 0$  then  $Timer_p[r] ++$ 
9:            $ElapseTime_p[r] \leftarrow Timer_p[r]$ 
10:         $Alive_p \leftarrow Alive_p \cup \{r\}$ 
11:        if  $r = q$  then broadcast( $ALIVE, r, p$ ) to  $V \setminus \{p\}$ 
12:      End if
13:    End if
14:  Done
15:  For all  $q \in V \setminus \{p\}$  do
16:    if  $ElapseTime_p[q] = 0$  then
17:       $Alive_p \leftarrow Alive_p \setminus \{q\}$ 
18:    else
19:       $ElapseTime_p[q] --$ 
20:    End if
21:  Done
22:   $Suspected_p \leftarrow V \setminus Alive_p$ 
23: Done
  
```

□

Unsuspicion of Correct Processes (4/4)

Corollary 2

There exists a time from which **no** correct process is suspected by any correct process.

Proof.

Let p and q be two correct processes.

- If $p = q$, q is never removed from $Alive_p$ and so never inserted into $Suspected_p$. Hence, the lemma holds in this case.
- Otherwise, q is regularly inserted into $Alive_p$, by Lemma 3.
 Assume, by contradiction, that q is suspected infinitely often by p .

```

1:  $Alive_p \leftarrow V; Suspected_p \leftarrow \emptyset$ 
2:  $Timer_p \leftarrow [k, \dots, k]; ElapseTime_p \leftarrow [k, \dots, k]$ 
3: While true do
4:   broadcast( $ALIVE, p, p$ ) to  $V \setminus \{p\}$ 
5:   For all  $q \in V \setminus \{p\}$  do
6:     if receive ( $ALIVE, r, q$ ) then
7:       if  $r \neq p$  then
8:         if  $ElapseTime_p[r] \leq 0$  then  $Timer_p[r] ++$ 
9:            $ElapseTime_p[r] \leftarrow Timer_p[r]$ 
10:         $Alive_p \leftarrow Alive_p \cup \{r\}$ 
11:        if  $r = q$  then broadcast( $ALIVE, r, p$ ) to  $V \setminus \{p\}$ 
12:      End if
13:    End if
14:  Done
15:  For all  $q \in V \setminus \{p\}$  do
16:    if  $ElapseTime_p[q] = 0$  then
17:       $Alive_p \leftarrow Alive_p \setminus \{q\}$ 
18:    else
19:       $ElapseTime_p[q] --$ 
20:    End if
21:  Done
22:   $Suspected_p \leftarrow V \setminus Alive_p$ 
23: Done
    
```

□

Unsuspection of Correct Processes (4/4)

Corollary 2

There exists a time from which **no** correct process is suspected by any correct process.

Proof.

Let p and q be two correct processes.

- If $p = q$, q is never removed from $Alive_p$ and so never inserted into $Suspected_p$. Hence, the lemma holds in this case.

- Otherwise, q is regularly inserted into $Alive_p$, by Lemma 3.

Assume, by contradiction, that q is suspected infinitely often by p .

Between every removal and insertion of q , $Timer_p[q]$ is incremented.

```

1:  $Alive_p \leftarrow V; Suspected_p \leftarrow \emptyset$ 
2:  $Timer_p \leftarrow [k, \dots, k]; ElapseTime_p \leftarrow [k, \dots, k]$ 
3: While true do
4:   broadcast( $ALIVE, p, p$ ) to  $V \setminus \{p\}$ 
5:   For all  $q \in V \setminus \{p\}$  do
6:     if receive ( $ALIVE, r, q$ ) then
7:       if  $r \neq p$  then
8:         if  $ElapseTime_p[r] \leq 0$  then  $Timer_p[r] ++$ 
9:            $ElapseTime_p[r] \leftarrow Timer_p[r]$ 
10:         $Alive_p \leftarrow Alive_p \cup \{r\}$ 
11:        if  $r = q$  then broadcast( $ALIVE, r, p$ ) to  $V \setminus \{p\}$ 
12:      End if
13:    End if
14:  Done
15:  For all  $q \in V \setminus \{p\}$  do
16:    if  $ElapseTime_p[q] = 0$  then
17:       $Alive_p \leftarrow Alive_p \setminus \{q\}$ 
18:    else
19:       $ElapseTime_p[q] --$ 
20:    End if
21:  Done
22:   $Suspected_p \leftarrow V \setminus Alive_p$ 
23: Done
  
```

□

Unsuspection of Correct Processes (4/4)

Corollary 2

There exists a time from which **no** correct process is suspected by any correct process.

Proof.

Let p and q be two correct processes.

- If $p = q$, q is never removed from $Alive_p$ and so never inserted into $Suspected_p$. Hence, the lemma holds in this case.

- Otherwise, q is regularly inserted into $Alive_p$, by Lemma 3.

Assume, by contradiction, that q is suspected infinitely often by p .

Between every removal and insertion of q , $Timer_p[q]$ is incremented.

So, the time between two consecutive receptions of $\langle ALIVE, q, _ \rangle$ by p regularly increases by Lemma 2 (lower bound), a contradiction to Lemma 3.

□

```

1:  $Alive_p \leftarrow V; Suspected_p \leftarrow \emptyset$ 
2:  $Timer_p \leftarrow [k, \dots, k]; ElapseTime_p \leftarrow [k, \dots, k]$ 
3: While true do
4:   broadcast( $ALIVE, p, p$ ) to  $V \setminus \{p\}$ 
5:   For all  $q \in V \setminus \{p\}$  do
6:     if receive ( $ALIVE, r, q$ ) then
7:       if  $r \neq p$  then
8:         if  $ElapseTime_p[r] \leq 0$  then  $Timer_p[r] ++$ 
9:            $ElapseTime_p[r] \leftarrow Timer_p[r]$ 
10:         $Alive_p \leftarrow Alive_p \cup \{r\}$ 
11:        if  $r = q$  then broadcast( $ALIVE, r, p$ ) to  $V \setminus \{p\}$ 
12:      End if
13:    End if
14:  Done
15:  For all  $q \in V \setminus \{p\}$  do
16:    if  $ElapseTime_p[q] = 0$  then
17:       $Alive_p \leftarrow Alive_p \setminus \{q\}$ 
18:    else
19:       $ElapseTime_p[q] --$ 
20:    End if
21:  Done
22:   $Suspected_p \leftarrow V \setminus Alive_p$ 
23: Done
  
```

Correctness of the Algorithm

By Corollaries 1 and 2, follows.

Theorem 1

Algorithm EP is a failure detector of type $\diamond\mathcal{P}$ in System $\mathcal{S}_{\diamond b}$.

Roadmap

- 1 Introduction
- 2 Definition
- 3 Application: a Consensus Algorithm
 - The Problem
 - The Model
 - The Algorithm
 - The Proof
- 4 Implementation of a Failure Detector (an example)
 - $\diamond\mathcal{P}$
 - The Model
 - The Algorithm
 - The Proof
- 5 References

References

- [1] T. D. Chandra and S. Toueg.
Unreliable failure detectors for asynchronous systems (preliminary version).
In L. Logrippo, editor, *Proceedings of the Tenth Annual ACM Symposium on Principles of Distributed Computing, Montreal, Quebec, Canada, August 19-21, 1991*, pages 325–340. ACM, 1991.
- [2] T. D. Chandra and S. Toueg.
Unreliable failure detectors for reliable distributed systems.
J. ACM, 43(2):225–267, 1996.
- [3] M. J. Fischer, N. A. Lynch, and M. Paterson.
Impossibility of distributed consensus with one faulty process.
J. ACM, 32(2):374–382, 1985.