Failure Detectors: Hierarchy and Minimality

Alain Cournier   Stéphane Devismes

Université de Picardie Jules Verne

April 26, 2023

Roadmap

1. Introduction
2. Reduction
   - General Algorithm for Boosting Completeness
   - First Example of Reduction: $\mathcal{P} \cong \mathcal{Q}$
   - Second Example of Reduction: $\diamond\mathcal{S} \cong \diamond\mathcal{W}$
   - Taxonomy
3. Minimality
   - The Weakest Failure Detector
   - With a majority of correct process
   - Without a majority of correct process
4. References

Roadmap

(Distributed) Failure Detector: an Oracle

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

[1] *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.[1]

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

[1] *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.[1]

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.

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

The Failure Detector Approach [2]

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[2] (the implementation of the failure detector)

[2] In particular, the necessary assumptions on the system.

The Failure Detector Approach [2]

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[2] (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] In particular, the necessary assumptions on the system.

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

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 $\diamond\mathcal{W}$

Roadmap

1. Introduction
2. Reduction
   - General Algorithm for Boosting Completeness
   - First Example of Reduction: $\mathcal{P} \cong \mathcal{Q}$
   - Second Example of Reduction: $\diamond\mathcal{S} \cong \diamond\mathcal{W}$
   - Taxonomy
3. Minimality
   - The Weakest Failure Detector
   - With a majority of correct process
   - Without a majority of correct process
4. References

Motivation

In presence of arbitrary process crashes, consensus requires partial synchrony assumptions to be solved [6].

However, the expressive power of two different partially synchronous systems may be difficult to compare.

Motivation

In presence of arbitrary process crashes, consensus requires partial synchrony assumptions to be solved [6].

However, the expressive power of two different partially synchronous systems may be difficult to compare.

For example:

- A system where all processes are synchronous and where there is at least one source.

  A source is a (synchronous) correct process with reliable and synchronous outgoing links.

- A system where all processes are eventually synchronous and all links are eventually reliable and synchronous.

Motivation

In presence of arbitrary process crashes, consensus requires partial synchrony assumptions to be solved [6].

However, the expressive power of two different partially synchronous systems may be difficult to compare.

For example:

- A system where all processes are synchronous and where there is at least one source.

  A source is a (synchronous) correct process with reliable and synchronous outgoing links.

- A system where all processes are eventually synchronous and all links are eventually reliable and synchronous.

Failure detectors can be compared by **reduction**!

Definition [2]

Similar to reductions in NP-Completeness

Let $\mathcal{D}$ and $\mathcal{D}'$ be two failure detectors.

Definition [2]

Similar to reductions in NP-Completeness

Let $\mathcal{D}$ and $\mathcal{D}'$ be two failure detectors.

$T_{\mathcal{D}\to\mathcal{D}'}$ is a reduction algorithm from $\mathcal{D}$ to $\mathcal{D}'$ if it emulates the output of $\mathcal{D}'$ using only $\mathcal{D}$.

Definition [2]

Similar to reductions in NP-Completeness

Let $\mathcal{D}$ and $\mathcal{D}'$ be two failure detectors.

$T_{\mathcal{D}\rightarrow\mathcal{D}'}$ is a reduction algorithm from $\mathcal{D}$ to $\mathcal{D}'$ if it emulates the output of $\mathcal{D}'$ using only $\mathcal{D}$.

In this case, $\mathcal{D}'$ is reductible to $\mathcal{D}$ and $\mathcal{D}'$ is weaker than $\mathcal{D}$ ($\mathcal{D}' \preceq \mathcal{D}$).

Definition [2]

Similar to reductions in NP-Completeness

Let $\mathcal{D}$ and $\mathcal{D}'$ be two failure detectors.

$T_{\mathcal{D}\rightarrow\mathcal{D}'}$ is a reduction algorithm from $\mathcal{D}$ to $\mathcal{D}'$ if it emulates the output of $\mathcal{D}'$ using only $\mathcal{D}$.

In this case, $\mathcal{D}'$ is reductible to $\mathcal{D}$ and $\mathcal{D}'$ is weaker than $\mathcal{D}$ ($\mathcal{D}' \preceq \mathcal{D}$).

In this case, every problem solvable with $\mathcal{D}'$ can be also solved with $\mathcal{D}$.

$\prec$

If there exists a reduction algorithm from $\mathcal{D}$ to $\mathcal{D}'$, but not vice versa, then $\mathcal{D}'$ strictly weaker than $\mathcal{D}$, denoted by

$$\mathcal{D}' \prec \mathcal{D}.$$

$\cong$

If there are both reduction algorithms from $\mathcal{D}$ to $\mathcal{D}'$ and from $\mathcal{D}'$ to $\mathcal{D}$, then $\mathcal{D}$ and $\mathcal{D}'$ are said to be equivalent, denoted by

$$\mathcal{D}' \cong \mathcal{D}.$$

Example of Reduction: Boosting Completeness

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

Weak Completeness: Every faulty process is eventually permanently suspected by some correct process.

Example of Reduction: Boosting Completeness

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

Weak Completeness: Every faulty process is eventually permanently suspected by some correct process.

Idea: Spread suspicions using broadcast. However, to not break accuracy, premature rumor should be undone.

Boosting Completeness

Assumptions

1. Complete Network Topology
2. Asynchronous identified processes: a process and its identifier are used equivalently (V is the set of processes)
3. Asynchronous reliable links (not necessarily FIFO)
4. Process failures: only crashes!
5. Any process p can broadcast a message to all processes (p included!)
6. $\mathcal{D}$: a failure detector

Boosting Completeness

Algorithm for every process p, output: $\mathit{Suspected}_p$

$\mathit{Suspected}_p \leftarrow \emptyset$
While *true* **do**
    `broadcast` $\langle \mathcal{D}_p, p \rangle$ `to` V
    For all $q \in V$ **do**
        If `receive` $\langle S, q \rangle$ **then**
            $\mathit{Suspected}_p \leftarrow (\mathit{Suspected}_p \cup S) \setminus \{q\}$
        End If
    Done
Done

Example of Reduction

$\mathcal{P} \cong \mathcal{Q}$

$\mathcal{P}$: Strong Completeness $+$ Strong Accuracy

$\mathcal{Q}$: Weak Completeness $+$ Strong Accuracy

Example of Reduction

$\mathcal{P} \cong \mathcal{Q}$

$\mathcal{P}$: Strong Completeness + Strong Accuracy

$\mathcal{Q}$: Weak Completeness + Strong Accuracy

By definition, $\mathcal{Q} \preceq \mathcal{P}$.

Example of Reduction

$\mathcal{P} \cong \mathcal{Q}$

$\mathcal{P}$: Strong Completeness + Strong Accuracy

$\mathcal{Q}$: Weak Completeness + Strong Accuracy

By definition, $\mathcal{Q} \preceq \mathcal{P}$.

We now let $\mathcal{D} = \mathcal{Q}$ and show that the previous algorithm is a reduction algorithm from $\mathcal{Q}$ to $\mathcal{P}$, *i.e.*, $\mathcal{P} \preceq \mathcal{Q}$.

Strong Accuracy

Strong accuracy: no process is suspected before it crashes

Since $\mathcal{Q}$ satisfies strong accuracy, broadcast messages only contain IDs of crashed processes.

Every received ID in S is the ID of some crashed process.

Every ID inserted into $Suspected_p$ is an identifier of some crashed process.

$\square$

$Suspected_p \leftarrow \emptyset$
While *true* **do**
    `broadcast` $\langle \mathcal{Q}_p, p \rangle$ `to` V
    For all $q \in V$ **do**
        If `receive` $\langle S, q \rangle$ **then**
            $Suspected_p \leftarrow (Suspected_p \cup S) \setminus \{q\}$
        End If
    Done
Done

Strong Completeness

Crashed processes only sent finitely many messages and every sent message is eventually received (reliable links): IDs of crashed processes are eventually no more removed from *Suspected*$_p$.

$Suspected_p \leftarrow \emptyset$
While *true* **do**
    `broadcast` $\langle Q_p, p \rangle$ `to` V
    For all $q \in V$ **do**
        If `receive` $\langle S, q \rangle$ **then**
            $Suspected_p \leftarrow (Suspected_p \cup S) \setminus \{q\}$
        End If
    Done
Done

□

Strong Completeness

Crashed processes only sent finitely many messages and every sent message is eventually received (reliable links): IDs of crashed processes are eventually no more removed from $\mathit{Suspected}_p$.

Let q be a faulty process.

Since $\mathcal{Q}$ satisfies weak completeness, q is eventually permanently suspected by some correct process p: eventually $q \in \mathcal{Q}_p$ forever.

p **correct** $+$ **Link Reliability:** Every correct process received infinitely many messages with $q \in S$ (from p).

Eventually $q \in \mathit{Suspected}_c$ forever, for every correct process c. □

$\mathit{Suspected}_p \leftarrow \emptyset$
While *true* **do**
    `broadcast` $\langle \mathcal{Q}_p, p \rangle$ `to` V
    For all $q \in V$ **do**
        If `receive` $\langle S, q \rangle$ **then**
            $\mathit{Suspected}_p \leftarrow (\mathit{Suspected}_p \cup S) \setminus \{q\}$
        End If
    Done
Done

From Weak to Strong Completeness

Since we have a reduction algorithm from $\mathcal{Q}$ to $\mathcal{P}$, we have $\mathcal{P} \preceq \mathcal{Q}$.

Now, by definition, $\mathcal{Q} \preceq \mathcal{P}$.

Hence, $\mathcal{P} \cong \mathcal{Q}$.

From Weak to Strong Completeness

Since we have a reduction algorithm from $\mathcal{Q}$ to $\mathcal{P}$, we have $\mathcal{P} \preceq \mathcal{Q}$.

Now, by definition, $\mathcal{Q} \preceq \mathcal{P}$.

Hence, $\mathcal{P} \cong \mathcal{Q}$.

Using, the same reduction algorithm, we can also show that $\mathcal{S} \cong \mathcal{W}$, $\diamond\mathcal{P} \cong \diamond\mathcal{Q}$, and $\diamond\mathcal{S} \cong \diamond\mathcal{W}$.

Example of Reduction

$\diamond\mathcal{S} \cong \diamond\mathcal{W}$

$$\begin{aligned} \diamond\mathcal{S} &: \text{ Strong Completeness} + \text{Eventually Weak Accuracy} \\ \diamond\mathcal{W} &: \text{ Weak Completeness} + \text{Eventually Weak Accuracy} \end{aligned}$$

Example of Reduction

$\diamond\mathcal{S} \cong \diamond\mathcal{W}$

$$\begin{aligned} \diamond\mathcal{S} &: \text{ Strong Completeness} + \text{Eventually Weak Accuracy} \\ \diamond\mathcal{W} &: \text{ Weak Completeness} + \text{Eventually Weak Accuracy} \end{aligned}$$

By definition, $\diamond\mathcal{W} \preceq \diamond\mathcal{S}$.

Example of Reduction

$\diamond\mathcal{S} \cong \diamond\mathcal{W}$

$$\begin{aligned} \diamond\mathcal{S} &: \text{ Strong Completeness} + \text{Eventually Weak Accuracy} \\ \diamond\mathcal{W} &: \text{ Weak Completeness} + \text{Eventually Weak Accuracy} \end{aligned}$$

By definition, $\diamond\mathcal{W} \preceq \diamond\mathcal{S}$.

We now let $\mathcal{D} = \diamond\mathcal{W}$ and show that the previous algorithm is a reduction algorithm from $\diamond\mathcal{W}$ to $\diamond\mathcal{S}$, *i.e.*, $\diamond\mathcal{S} \preceq \diamond\mathcal{W}$.

Eventually Weak Accuracy

Eventually Weak Accuracy: there is a time after which some correct process is never suspected by any correct process.

Since $\diamond\mathcal{W}$ satisfies eventually weak accuracy, there is some correct process c that is eventually no more suspected by all correct processes.

Hence, eventually no more broadcast message contains c.

Eventually no received ID in S is the ID of c.

Eventually c is no more inserted into $Suspected_p$.

c is removed from $Suspected_p$ infinitely often since links are reliable and c is correct.

Eventually $c \notin Suspected_p$ forever. $\square$

$Suspected_p \leftarrow \emptyset$
While *true* **do**
    `broadcast` $\langle \diamond\mathcal{W}_p, p \rangle$ `to` V
    For all $q \in V$ **do**
        If `receive` $\langle S, q \rangle$ **then**
            $Suspected_p \leftarrow (Suspected_p \cup S) \setminus \{q\}$
        End If
    Done
Done

Strong Completeness

As previously ...

$Suspected_p \leftarrow \emptyset$
While *true* **do**
    `broadcast` $\langle \diamond\mathcal{W}_p, p \rangle$ `to` V
    For all $q \in V$ **do**
        If `receive` $\langle S, q \rangle$ **then**
            $Suspected_p \leftarrow (Suspected_p \cup S) \setminus \{q\}$
        End If
    Done
Done

From Weak to Strong Completeness

Since we have a reduction algorithm from $\diamond\mathcal{W}$ to $\diamond\mathcal{S}$, we have $\diamond\mathcal{S} \preceq \diamond\mathcal{W}$.

Now, by definition, $\diamond\mathcal{W} \preceq \diamond\mathcal{S}$.

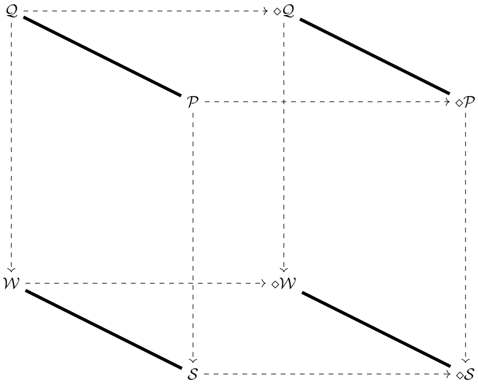
Hence, $\diamond\mathcal{S} \cong \diamond\mathcal{W}$.

Taxonomy [3] (1/2)

Theorem 1

- $\mathcal{P} \cong \mathcal{Q}$,
- $\mathcal{S} \cong \mathcal{W}$,
- $\diamond\mathcal{P} \cong \diamond\mathcal{Q}$,
- $\diamond\mathcal{S} \cong \diamond\mathcal{W}$,
- $\mathcal{S} \prec \mathcal{P}$,
- $\diamond\mathcal{S} \prec \diamond\mathcal{P}$,
- $\diamond\mathcal{P} \prec \mathcal{P}$,
- $\diamond\mathcal{S} \prec \mathcal{S}$,
- $\diamond\mathcal{S} \prec \mathcal{P}$, and
- $\mathcal{S}$ and $\diamond\mathcal{P}$ are **incomparable**.

Taxonomy [3] (2/2)



$\mathcal{D} \dashrightarrow \mathcal{D}'$: $\mathcal{D} \succ \mathcal{D}'$

$\mathcal{D}$ —— $\mathcal{D}'$: $\mathcal{D} \cong \mathcal{D}'$

Roadmap

1. Introduction
2. Reduction
   - General Algorithm for Boosting Completeness
   - First Example of Reduction: $\mathcal{P} \cong \mathcal{Q}$
   - Second Example of Reduction: $\diamond\mathcal{S} \cong \diamond\mathcal{W}$
   - Taxonomy
3. Minimality
   - The Weakest Failure Detector
   - With a majority of correct process
   - Without a majority of correct process
4. References

The Weakest Failure Detector

The weakest failure detector to solve a problem P is the failure detector $\mathcal{D}$ that is both necessary and sufficient to solve P, *i.e.*, it is weaker than any other failure detector that can solve P.

The Weakest Failure Detector

The weakest failure detector to solve a problem P is the failure detector $\mathcal{D}$ that is both necessary and sufficient to solve P, *i.e.*, it is weaker than any other failure detector that can solve P.

To that goal, it is sufficient to show the following two claims:

- There exists an algorithm that solves P using $\mathcal{D}$.
- It is possible to emulate $\mathcal{D}$ with any failure detector $\mathcal{D}'$ that is sufficient to solve P.

  (In particular, we can use every algorithm that solves P using $\mathcal{D}'$ in the reduction.)

Ω [1]

Eventual Leader Election:

There is a correct process c such that eventually $\Omega_p = c$ forever for every correct process p.

$\Omega \cong \diamond\mathcal{W}$ ($\cong \diamond\mathcal{S}$)

$T_{\Omega\to\diamond\mathcal{W}}$: return $V \setminus \{\Omega\}$.

Eventually $\Omega_p = c$ forever for each correct process p, where c is a correct process $\Rightarrow$ Eventual Weak Accuracy $+$ Weak Completeness.

So, $\Omega \preceq \diamond\mathcal{W}$.

$T_{\diamond\mathcal{W}\to\Omega}$:

- Regularly evaluate $\diamond\mathcal{W}_p$.
- Local count the number of times each process is suspected.
- Broadcast local counters $+$ keep the max for each process.
- Elect the less suspected (use IDs to break tees).

By eventually weak accuracy, at least one correct process has a bounded counter. Counters of faulty processes are unbounded.

So, $\diamond\mathcal{W} \preceq \Omega$.

Assumptions

1. Complete Network Topology
2. A majority of processes is correct: the maximal number of crashes f satisfies $n > 2f$ where n is the number of processes
3. Asynchronous identified processes: a process and its identifier are used equivalently (V is the set of processes)
4. Asynchronous reliable links (not necessarily FIFO)
5. Any process p can broadcast a message to all processes (p included!)
6. Failure Detector: Ω

Ω is necessary and sufficient

Under these assumptions, Ω is the weakest failure detector to solve the consensus [1].

We admit the proof of necessity (which is quite complex ...)

Let see now the sufficient part of the proof, *i.e.*, a consensus algorithm!

The Ben-Or Algorithm (Recall)

$d_p \leftarrow \perp$
$r \leftarrow 0$
While *true* **do**
    $r{+}{+}$
    `broadcast` (R, r, v_p) `to all processes` (p `included)`
    `wait to receive` $n - f$ messages $(R, r, \_)$ where "_" is 0 or 1
    If more than $\frac{n}{2}$ received messages (R, r, x) with the same value x **then**
        `broadcast` (P, r, x) `to all processes` (p `included)`
    else
        `broadcast` $(P, r, ?)$ `to all processes` (p `included)`
    End If
    `wait to receive` $n - f$ messages $(P, r, \_)$ where "_" is 0, 1, or ?
    If at least $f + 1$ received messages (P, r, x) with $x \neq ?$ **then**
        If $d_p = \perp$ **then** $d_p \leftarrow x$
    End If
    If at least 1 received message (P, r, x) with $x \neq ?$ **then**
        $v_p \leftarrow x$
    else
        $v_p \leftarrow \mathtt{Random}(0, 1)$
    End If
Done

Derandomization of The Ben-Or Algorithm using Ω

$d_p \leftarrow \perp$
$r \leftarrow 0$
While *true* **do**
$r{+}{+}$
broadcast (V, r, v_p) to all processes (p included)
wait to receive (V, r, y) from Ω_p
$v_p \leftarrow y$
broadcast (R, r, v_p) to all processes (p included)
wait to receive $n - f$ messages $(R, r, \_)$ where "_" is 0 or 1
If more than $\frac{n}{2}$ received messages (R, r, x) with the same value x **then**
broadcast (P, r, x) to all processes (p included)
else
broadcast $(P, r, ?)$ to all processes (p included)
End If
wait to receive $n - f$ messages $(P, r, \_)$ where "_" is 0, 1, or ?
If at least $f + 1$ received messages (P, r, x) with $x \neq ?$ **then**
If $d_p = \perp$ **then** $d_p \leftarrow x$
End If
If at least 1 received message (P, r, x) with $x \neq ?$ **then**
$v_p \leftarrow x$
else
$\cancel{v_p \leftarrow \text{Random}(0,1)}$
End If
Done

Sketch of Proof

Agreement, Integrity, and Validity: like in the Ben-Or's proof

Termination:

- Eventually (at some round r) all alive processes are correct and agree on the same correct process.
- They all wait for the same value from Ω.
- They all report the same value.
- They all decide the same value (as for Ben-Or)!

Σ: the quorum failure detector [4]

Σ = list of trusted processes.

Eventual Strong Completeness + Quorum

Eventual Strong Completeness: for every correct process p, eventually Σ_p forever outputs lists only containing correct processes.

Quorum:

$\forall p, q \in V,$

$\forall t, t',$

if p is alive at time t and q is alive at time t', then

$\Sigma_p^t \cap \Sigma_q^{t'} \neq \emptyset.$

Assumptions

1. Complete Network Topology
2. Asynchronous identified processes: a process and its identifier are used equivalently (V is the set of processes)
3. Asynchronous reliable links (not necessarily FIFO)
4. Any process p can broadcast a message to all processes (p included!)
5. Failure Detector: $\Sigma \times \Omega$

Under these assumptions, $\Sigma \times \Omega$ is the weakest failure detector to solve the consensus [5].

We admit the proof of necessity (which is quite complex ...)

Let's see now the sufficient part of the proof, *i.e.*, a consensus algorithm!

Derandomization of the Ben-Or Algorithm using $\Sigma \times \Omega$

$d_p \leftarrow \perp$
$r \leftarrow 0$
While *true* **do**
    $r{+}{+}$
    `broadcast` (V, r, v_p) `to all processes` (p `included)`
    `wait to receive` (V, r, y) from Ω_p
    $v_p \leftarrow y$
    `broadcast` (R, r, v_p) `to all processes` (p `included)`
    `wait to receive` messages $(R, r, \_)$ (where "_" is 0 or 1) from all processes in Σ_p
    If all received messages (R, r, x) with the same value x **then**
        `broadcast` (P, r, x) `to all processes` (p `included)`
    else
        `broadcast` $(P, r, ?)$ `to all processes` (p `included)`
    End If
    `wait to receive` messages $(P, r, \_)$ (where "_" is 0, 1, or ?) from all processes in Σ_p
    If all received messages (P, r, x) with the same value $x \neq ?$ **then**
        If $d_p = \perp$ **then** $d_p \leftarrow x$
    End If
    If at least 1 received message (P, r, x) with $x \neq ?$ **then**
        $v_p \leftarrow x$
    End If
Done

Sketch of Proof

The Quorum property guarantees that if a process decides x in Line 17 at Round r then

1. No process can decide differently during Round r; and
2. all processes that will not decide on Round r will set their v-variable to x on Lines 19-21 during Round r!

Hence, the proof arguments are similar to the Ben-Or's proof!

Roadmap

1. Introduction
2. Reduction
   - General Algorithm for Boosting Completeness
   - First Example of Reduction: $\mathcal{P} \cong \mathcal{Q}$
   - Second Example of Reduction: $\diamond\mathcal{S} \cong \diamond\mathcal{W}$
   - Taxonomy
3. Minimality
   - The Weakest Failure Detector
   - With a majority of correct process
   - Without a majority of correct process
4. References

References I

[1] T. D. Chandra, V. Hadzilacos, and S. Toueg.
The weakest failure detector for solving consensus.
J. ACM, 43(4):685–722, jul 1996.

[2] 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.

[3] T. D. Chandra and S. Toueg.
Unreliable failure detectors for reliable distributed systems.
J. ACM, 43(2):225–267, 1996.

[4] C. Delporte-Gallet, H. Fauconnier, and R. Guerraoui.
Shared memory vs message passing.
Technical report, EPFL, 2003.
https://infoscience.epfl.ch/record/52584?ln=fr.

References II

[5] C. Delporte-Gallet, H. Fauconnier, R. Guerraoui, V. Hadzilacos, P. Kouznetsov, and S. Toueg.
The weakest failure detectors to solve certain fundamental problems in distributed computing.
In S. Chaudhuri and S. Kutten, editors, *Proceedings of the Twenty-Third Annual ACM Symposium on Principles of Distributed Computing, PODC 2004, St. John's, Newfoundland, Canada, July 25-28, 2004*, pages 338–346. ACM, 2004.

[6] M. J. Fischer, N. A. Lynch, and M. Paterson.
Impossibility of distributed consensus with one faulty process.
J. ACM, 32(2):374–382, 1985.