Failure Detection Lower Bounds on Registers and Consensus (Preliminary Version)

C. Delporte-Gallet †, H. Fauconnier †, and R. Guerraoui ‡

†Laboratoire d'Informatique Algorithmique: Fondements et Applications
Université Paris VII - Denis Diderot
‡Distributed Programming Laboratory
Swiss Federal Institute of Technology in Lausanne

Abstract

This paper addresses the problem of determining the weakest failure detector to implement consensus in a message passing system when t out of n processes can crash (including when $n/2 \leq t < n-1$), by addressing the problem of determining the weakest failure detector to implement a register. We complement and, in a precise sense, generalise previous results on the implementability of consensus and registers in a message passing model (augmented with the failure detector abstraction).

- **Category:** Regular.
- **Contact Author:** Rachid Guerraoui. LPD EPFL CH1015 Lausanne, Switzerland. Tel: 41 21 693 52 72. Fax: 41 21 693 75 70.

1 Introduction

This paper considers an asynchronous distributed system augmented with the failure detector abstraction [4]. The system is made of n processes that communicate through reliable channels but t among these processes can fail by crashing [9]. The system is asynchronous in the sense that there is no timing assumption on communication delays and process relative speeds. In fact, timing assumptions [6] are encapsulated within the failure detector abstraction: a distributed oracle that provides each process with hints about failures that have occurred in the system.

Several classes of failure detectors were proposed in the literature, each gathering a set of failure detectors that ensure some abstract axiomatic properties. In particular, three interesting classes were identified in [4]: the class $\mathcal{P}$ of *Perfect* failure detectors that eventually suspect all crashed processes and never make false suspicions, the class $\mathcal{S}$ of *Strong* failure detectors that eventually suspect all crashes and never make false suspicions on at least one correct process (if there is such a process), and the class $\Diamond\mathcal{S}$ of *Eventually Strong* failure detectors that eventually suspect all crashes and eventually never make false suspicions on at least one correct process (if there is such a process).

In [4], two algorithms using failure detectors were proposed to implement the consensus problem. In consensus, the processes propose an initial value and correct processes (those that do not crash) need to decide one among these values, such that no two processes decide differently [9]. We consider in this paper the *uniform* variant of consensus, which precludes any disagreement among two processes, even if one of them ends up crashing [13]. We will come back to the impact of this assumption in Section 6. The first algorithm of [4] implements consensus with any failure detector of $\mathcal{S}$ for any t, whereas the second algorithm implements consensus with any failure detector of $\Diamond\mathcal{S}$ if $t < n/2$. It was furthermore shown [5] that there

is an algorithm that transforms any failure detector $\mathcal{D}$ that implements consensus into a failure detector of $\Diamond\mathcal{S}$. In the parlance of [5], $\Diamond\mathcal{S}$ is the *weakest* to implement consensus if $t < n/2$.

When $t \geq n/2$, $\Diamond\mathcal{S}$ is not the weakest for consensus [4]. *What is then the weakest failure detector class for consensus when $t \geq n/2$?* The question remained open for more than a decade now.

In a recent companion paper [7], we addressed this question for the specific case where $t \in \{n, n-1\}$. We first restricted the universe of failure detectors to those, we called *realistic*, that cannot predict the future. Basically, we focused on determining the weakest failure detector class among those that provide information about *past* failures and cannot provide information about failures that *will* occur [14, 7]. Although failure detectors that predict the future are permitted in [4], they cannot be implemented even in the synchronous model of [20], where process relative speeds and communication delays are bounded, and these bounds are known. Note that $\Diamond\mathcal{S}$ is the *weakest* among *realistic* classes for consensus if $t < n/2$.

We then showed that, among *realistic* failure detectors, $\mathcal{P}$ is the weakest for consensus when $t \in \{n, n-1\}$. (We also showed that in this case the classes $\mathcal{P}$ and $\mathcal{S}$ are the same.) The case where $t \in \{n, n-1\}$ is actually very specific. *What happens when $n/2 \leq t < n-1$?* This question turns out to be challenging: none of the failure detector classes defined so far in the literature, including those of [4] (e.g., $\mathcal{S}$), is actually the weakest for consensus if $n/2 \leq t < n-1$.

The motivation of this work was precisely to address this question. While doing so, we faced another interesting problem: determining the weakest failure detector class to implement a register when $n/2 \leq t \leq n$. What we actually mean here is to determine the weakest failure detector class to implement a *wait-free atomic register* in the sense of [3] [1], i.e., to implement a data abstraction accessed through *read()* and *write()* operations such that, despite concurrent accesses and failures of processes, (1) every operation invoked by a correct process eventually returns, (2) although it spans over an interval time, every operation appears to have been executed at a single point in time in this interval, and (3) every read operation returns the last value written. In a purely asynchronous message passing system model (e.g., without failure detectors) a register can be implemented iff $t < n/2$ [3]. To implement a register when $n/2 \leq t \leq n$, we need some information about failures. Several authors considered augmenting an asynchronous model with registers and failure detectors (e.g., [19, 22]) but, to our knowledge, the question of the exact failure detector class that implements a register in a message passing model was never addressed for $n/2 \leq t \leq n$.

This paper defines a new *generic* failure detector class $\mathcal{P}^k$ (*k-Perfect*). Instances of this generic class are determined by the integer k ($0 \leq k \leq n$). Failure detectors of class $\mathcal{P}^k$ eventually suspect all crashed processes and do not falsely suspect more than $max(n-k-1, 0)$ processes at every process. The processes might permanently disagree here on their perception on which processes have crashed, i.e., on the subset of $n-k-1$ processes each process falsely suspects. They might even change their mind about which processes they falsely suspect. For $k < n/2$, failure detectors of class $\mathcal{P}^k$ can be implemented in an asynchronous system if up to $t < n/2$ processes can crash, i.e., they do not encapsulate any timing assumption. For $k \geq n-1$, $\mathcal{P}^k$ is $\mathcal{P}$, i.e., $\mathcal{P}^{n-1} = \mathcal{P}^n = \mathcal{P}$.

We show that, if we assume that t processes can crash, then $\mathcal{P}^t \times \Diamond\mathcal{S}$ is the weakest class to implement consensus. To prove our result, we prove the interesting intermediate result that, if we assume that t processes can crash, then $\mathcal{P}^t$ is the weakest to implement a register. Our results hold among *realistic* failure detectors: the very fact that we exclude failure detectors that can predict the future is meaningful from a practical perspective [7], yet it has however some impact on our theoretical results, as we discuss in Section 6.
To summarize, the contributions of this paper are the following:

1. We introduce a new failure detector class $\mathcal{P}^k$, and assuming that t processes can crash, we show that (among *realistic* failure detectors):
2. $\mathcal{P}^t$ is the weakest to implement a register, and
3. $\mathcal{P}^t \times \Diamond\mathcal{S}$ is the weakest to implement consensus.

For the case where $n/2 \leq t < n-1$, we hence address the open questions of the weakest failure detector classes to implement consensus and atomic register. We also show that, in this case, $\mathcal{S}$ is strictly stronger than $\mathcal{P}^t \times \Diamond\mathcal{S}$, revisiting the first glance intuition that $\mathcal{S}$ might have been the weakest for consensus with

[1] The *wait-free* notion was introduced in [16] and the notion of *atomic register* was introduced in [18].

$n/2 \leq t < n-1$. For $t \in \{n, n-1\}$, and given that in this case $\mathcal{P}^t$ is $\mathcal{P}$, our result comes down to the result of [7] as far as consensus is concerned, and we address the open question of the weakest failure detector to implement an atomic register if any number of processes can crash. For $n/2 > t$, and given that in this case $\mathcal{P}^t$ can be implemented in an asynchronous system, our result simply comes down the known result that, with a majority of correct processes, no timing assumption is needed to implement an atomic register [3], and that $\Diamond\mathcal{S}$ is the weakest for consensus [5].

Interestingly, the decoupled structure of our weakest failure detector class for consensus (*i.e.*, $\mathcal{P}^t \times \Diamond\mathcal{S}$) conveys its double role: $\mathcal{P}^t$ encapsulates the information about failures needed to ensure the safety part of consensus (i.e., to implement a register that will lock the consensus value and prevent disagreement), whereas $\Diamond\mathcal{S}$ encapsulates the information about failures needed to ensure the liveness part of consensus (i.e., to ensure that some correct will eventually stop being suspected and store a decision value in the register). We prove our results using simple algorithm reductions (like in [5] but unlike in [14]). Hence, by determining the weakest failure detector class to implement a register (or consensus), we determine what exact information about failures processes need to know and effectively compute to implement a register (or consensus).

The rest of the paper is organized as follows. In Section 2, we sketch our system model and, in particular, we specify the universe of failure detectors within which we identify the weakest for consensus and atomic register. Section 3 defines our generic failure detector class $\mathcal{P}^k$. We compare instances of this class, and position them with respect to various failure detector classes introduced in the literature. Section 4 shows (sufficient condition) that, if t processes can crash, then $\mathcal{P}^t$ implements a register. We then simply reuse the result of [19] to derive the fact than $\mathcal{P}^t \times \Diamond\mathcal{S}$ implements consensus. Section 5 shows (necessary condition) that if t processes can crash, then any failure detector that implements a register can be transformed into a failure detector of $\mathcal{P}^t$. We then simply reuse the fact that consensus can implement a register and the result of [5] to show that any failure detector that implements consensus can be transformed into a failure detector of $\mathcal{P}^t \times \Diamond\mathcal{S}$. Section 6 concludes the paper by discussing the scope of our results. For space limitation, detailed proofs of our results are given in the appendix.

2 System model

Our model of asynchronous computation with failure detection is the FLP model [9] augmented with the failure detector abstraction [4, 5]. A discrete global clock is assumed, and Φ, the range of the clock's ticks, is the set of natural numbers. The global clock is used for presentation simplicity and is not accessible to the processes. We sketch here the fundamentals of the model. The reader interested in specific details about the model should consult [5, 7].

2.1 Failure patterns and environments

We consider a distributed system composed of a finite set of n processes $\Pi = \{p_1, p_2, \ldots, p_n\}$ ($|\Pi| = n \geq 3$). A process p is said to *crash at time* τ if p does not perform any *action* after time τ (the notion of *action* is recalled below). Failures are permanent, i.e., no process *recovers* after a crash. A *correct* process is a process that does not crash. A *failure pattern* is a function F from Φ to 2^{Π}, where $F(\tau)$ denotes the set of processes that have crashed through time τ. The set of correct processes in a failure pattern F is noted $correct(F)$. We say that a process p is *alive* at time τ if p is not in $F(\tau)$. An *environment* $\mathcal{E}$ is a set of failure patterns. Environments describe the crashes that can occur in a system. In this paper, we consider environments, denoted by $\mathcal{E}_t$, composed of all failure patterns with at most t crashes.

2.2 Failure detectors

Roughly speaking, a failure detector $\mathcal{D}$ is a distributed oracle which gives hints about failure patterns. Each process p has a local failure detector module of $\mathcal{D}$, denoted by $\mathcal{D}_p$. Associated with each failure detector $\mathcal{D}$ is a range $R_{\mathcal{D}}$ (when the context is clear we omit the subscript) of values output by the failure detector. A *failure detector history* H with range R is a function H from $\Pi \times \Phi$ to R. For every process $p \in \Pi$, for every time $\tau \in \Phi$, $H(p, \tau)$ denotes the value of the failure detector module of process p at time τ, i.e., $H(p, \tau)$ denotes the value output by $\mathcal{D}_p$ at time τ. A *failure detector* $\mathcal{D}$ is defined as a function that maps each

failure pattern F to a set of failure detector histories with range $R_{\mathcal{D}}$. $\mathcal{D}(F)$ denotes the set of possible failure detector histories permitted for the failure pattern F, i.e., each history represents a possible behavior of $\mathcal{D}$ for the failure pattern F.

In [4], any function of the failure pattern is a failure detector, including a function that, for a time τ, outputs information about crashes that will occur *after* τ. We restrict ourselves here to (*realistic* [7]) failure detectors $\mathcal{D}$ as functions of the *past*, i.e., we exclude failure detectors that can predict the future (we will come back to the ramifications of this assumption in Section 6). A failure detector cannot distinguish at a time τ two failure patterns based on what will happen after τ. More precisely, $\forall F, F' \in \mathcal{E}_n$, $\forall \tau \in \Phi$ s.t. $\forall \tau_1 \leq \tau, F(\tau_1) = F'(\tau_1)$, we have the following *realism* property: $\forall H \in \mathcal{D}(F), \exists H' \in \mathcal{D}(F')$ s.t. $\forall \tau_1 \leq \tau, \forall p \in \Pi : H(p, \tau_1) = H'(p, \tau_1)$.

Three classes of failure detectors introduced in [4] are of interest in this paper. These classes do all have range $R = 2^{\Pi}$: for any failure detector $\mathcal{D}$ in these classes, any failure pattern F, and any history H in $\mathcal{D}(F)$, $H(p, \tau)$ is the set of processes *suspected* by process p at time τ. (1) The class of *Perfect* failure detectors ($\mathcal{P}$) gathers all those that ensure *strong completeness*, i.e., eventually every process that crashes is permanently suspected by every correct process, and *strong accuracy*, i.e., no process is suspected before it crashes. (2) The class of *Strong* failure detectors ($\mathcal{S}$) gathers those that ensure *strong completeness* and *weak accuracy*, i.e., some correct process is never suspected (if there is such a process). (3) The class of *Eventually Strong* failure detectors ($\Diamond\mathcal{S}$) gathers those that ensure *strong completeness* and *eventual weak accuracy*, i.e., eventually, some correct process is never suspected (if there is such a process). Note that, in the context of this paper, we restrict these classes to failures detectors that are *realistic*.

2.3 Algorithms

An *algorithm* using a failure detector $\mathcal{D}$ is a collection A of n deterministic automata A_p (one per process p). Computation proceeds in steps of the algorithm. In each step of an algorithm A, a process p atomically performs the following three actions: (1) p receives a message from some process q, or a "null" message λ; (2) p queries and receives a value d from its failure detector module $\mathcal{D}_p$ ($d \in R_{\mathcal{D}}$ is said to be *seen* by p); (3) p changes its state and sends a message (possibly null) to some process. This third action is performed according to (a) the automaton A_p, (b) the state of p at the beginning of the step, (c) the message received in action 1, and (d) the value d seen by p in action 2. The message received by a process is chosen non-deterministically among the messages in the message buffer destined to p, and the null message λ. A *configuration* is a pair (I, M) where I is a function mapping each process p to its local state, and M is a set of messages currently in the message buffer. A configuration (I, M) is an initial configuration if $M = \emptyset$ (no message is initially in the buffer): in this case, the states to which I maps the processes are called *initial states*. A *step* of an algorithm A is a tuple $e = (p, m, d, A)$, uniquely defined by the algorithm A, the identity of the process p that takes the step, the message m received by p, and the failure detector value d seen by p during the step. A step $e = (p, m, d, A)$ is *applicable to a configuration* (I, M) if and only if $m \in M \cup \{\lambda\}$. The *unique* configuration that results from applying e to configuration $C = (I, M)$ is noted $e(C)$.

2.4 Schedules and runs

A *schedule* of an algorithm A is a (possibly infinite) sequence $S = S[1]; S[2]; \ldots\ S[k]; \ldots$ of steps of A. A schedule S is applicable to a configuration C if (1) S is the empty schedule, or (2) $S[1]$ is applicable to C, $S[2]$ is applicable to $S[1](C)$ (the configuration obtained from applying $S[1]$ to C), etc.

Let A be any algorithm and $\mathcal{D}$ any failure detector. A *run of* A *using* $\mathcal{D}$ is a tuple $R =< F, H, C, S, T >$ where H is a failure detector history and $H \in \mathcal{D}(F)$, C is an initial configuration of A, S is an infinite schedule of A, T is an infinite sequence of increasing time values, and (1) S is applicable to C, (2) for all $k \leq |S|$ where $S[k] = (p, m, d, A)$, we have $p \notin F(T[k])$ and $d = H(p, T[k])$, (3) every correct process takes an infinite number of steps, and (4) every message sent to a correct process p is eventually received by p.[2]

[2]In fact, our results and proofs hold with a weaker assumption where we only require that (4') every message sent by a correct process to a correct process p is eventually received by p.

2.5 Implementability

An algorithm A *implements* a problem B using a failure detector $\mathcal{D}$ in an environment $\mathcal{E}_t$ if every run of A using $\mathcal{D}$ in $\mathcal{E}_t$ satisfies the specification of B. We say that $\mathcal{D}$ implements B in $\mathcal{E}_t$ if there is an algorithm that implements B using $\mathcal{D}$ in $\mathcal{E}_t$. We say that a failure detector $\mathcal{D}1$ is *stronger* than a failure detector $\mathcal{D}2$ in environment $\mathcal{E}_t$ ($\mathcal{D}2 \preceq_t \mathcal{D}1$) if there is an algorithm (called a *reduction algorithm*) that transforms $\mathcal{D}1$ into $\mathcal{D}2$ in $\mathcal{E}_t$, i.e., that can *emulate* the output $\mathcal{D}2$ using $\mathcal{D}1$ in $\mathcal{E}_t$ [4]. The algorithm does not need to emulate all histories of $\mathcal{D}2$. It is required however that, for every run $R =< F, H, C, S, T >$ where $H \in \mathcal{D}1(F)$, the output of the algorithm with R be a history of $\mathcal{D}2(F)$. We say that $\mathcal{D}1$ is *strictly stronger* than $\mathcal{D}2$ in environment $\mathcal{E}_t$ ($\mathcal{D}2 \prec_t \mathcal{D}1$) if $\mathcal{D}2 \preceq_t \mathcal{D}1$ and $\neg(\mathcal{D}1 \preceq_t \mathcal{D}2)$. We say that $\mathcal{D}1$ is *equivalent to* $\mathcal{D}2$ in environment $\mathcal{E}_t$ ($\mathcal{D}1 \equiv_t \mathcal{D}2$), if $\mathcal{D}2 \preceq_t \mathcal{D}1$ and $\mathcal{D}1 \preceq_t \mathcal{D}2$.

Finally, we say that a failure detector $\mathcal{D}$ is the weakest to implement a problem B in environment $\mathcal{E}_t$ if (a. sufficient condition) $\mathcal{D}$ implements B in $\mathcal{E}_t$ and (b. necessary condition) any failure detector that implements B is stronger than $\mathcal{D}$ in $\mathcal{E}_t$.

3 k-Perfect failure detectors

3.1 Definitions

We define in this section the generic class of *k-Perfect* ($\mathcal{P}^k$) failure detectors, where $0 \leq k \leq n$. This class is generic in the sense that its semantics depend on the value of the integer k. Failure detectors of class $\mathcal{P}^k$ output, at each process p and each time τ, a list of suspected processes $\mathcal{P}^k(p, \tau)$ (i.e., the range of $\mathcal{P}^k$ is 2^Π). These failure detectors ensure *strong completeness* as well as the following *k-accuracy* property: at any time τ, no process suspects more than $n - k - 1$ processes that are alive at time τ. More precisely:

- k- **Accuracy:** $\forall p \in \Pi$, $\forall \tau \in \Phi$ $|\mathcal{P}^k(p, \tau) \setminus F(\tau)| \leq max(n - k - 1, 0)$.

For $k \geq n - 1$, processes do not make false suspicions and $\mathcal{P}^k$ is $\mathcal{P}$. For $k < n - 1$, processes can make false suspicions and can even permanently disagree on the processes they falsely suspect. To better illustrate the behavior of a *k-Perfect* failure detector, consider a system of 5 processes $\{p_1, p_2, p_3, p_4, p_5\}$ and the case $k = 2$. The failure detector should eventually suspect permanently all crashed processes and should not falsely suspect more that 2 processes at every process. Consider a failure pattern where p_1 and p_2 crash. It can be the case that after some time τ, p_3 permanently suspects $\{p_1, p_2, p_4, p_5\}$, p_4 permanently suspects $\{p_1, p_2, p_3, p_5\}$, and p_5 permanently suspects $\{p_1, p_2, p_3, p_4\}$. It can also be the case that after some time τ, p_5 forever alternately suspects $\{p_1, p_2, p_3\}$ and $\{p_1, p_2, p_4\}$.

The idea of limited accuracy is not new. Failure detectors of $\mathcal{S}$ already provide a limited form of accuracy, with respect to those of $\mathcal{P}$, in the sense that they only need to ensure *weak accuracy*: they can falsely suspect correct processes, as long as there is one correct process that is never suspected. Our notion of limited accuracy is different in that processes do not need to agree on a process they never suspect, as conveyed by our example above and specified by our Proposition 3.3 below. In [11, 21], the notion of accuracy was further limited in the sense that only a subset of the processes need to satisfy it: again, and even if we consider the case of *weak accuracy*, the subset of processes should still agree on some process not to suspect. In [4], the authors introduced the notion of *k-Mistaken* failure detectors as those that can make k false suspicions. In our case, except when $k \in \{n-1, n\}$ (*Perfect* failure detection), a failure detector of $\mathcal{P}^k$ can make an infinite number of mistakes.

We also define here the failure detector class $\mathcal{P}^t \times \Diamond\mathcal{S}$. This class gathers failure detectors $\mathcal{D}$ of range $2^\Pi \times 2^\Pi$, such that given H_1 a history of a failure detector of $\mathcal{P}^t$ and H_2 a history of a failure detector of $\Diamond\mathcal{S}$, for each process p and at each time τ, the value of the failure detector module $\mathcal{D}_p$ is a pair $(H_1(p, \tau), H_2(p, \tau))$.

3.2 Relationships

In the following, we give some general properties of $\mathcal{P}^k$ failure detectors. Certain properties follow trivially from the definition. Others are less trivial and we prove them in the appendix.

Proposition 3.1 $\forall t, k, k' \in [0, n],\ k \leq k' \Rightarrow \mathcal{P}^k \preceq_t \mathcal{P}^{k'}$.

Proposition 3.2 $\forall t, t' \in [n/2, n-1],\ t' < t \Rightarrow \mathcal{P}^{t'} \prec_t \mathcal{P}^t$.

We already pointed out the fact that $\mathcal{P}^{n-1} = \mathcal{P}^n = \mathcal{P}$. We state below a relationship between $\mathcal{P}^t$ and $\mathcal{S}$.

Proposition 3.3 $\forall t \in [1, n-2],\ \mathcal{P}^t \times \Diamond\mathcal{S} \prec_t \mathcal{S}$.

We show (later in the paper) that, for any t, $\mathcal{P}^t \times \Diamond\mathcal{S}$ is the weakest failure detector class for consensus in $\mathcal{E}_t$. Hence, a corollary of Proposition 3.3 above is that $\mathcal{S}$ is not the weakest for consensus in $\mathcal{E}_t$ if $t < n-1$. Note that we have shown in [7] that $\mathcal{P}$ (and hence $\mathcal{P}^{n-1}$ and $\mathcal{P}^n$) is equivalent to $\mathcal{S}$ in $\mathcal{E}_{(n-1)}$ and $\mathcal{E}_n$.[3]

We state below some relationships between $\mathcal{P}^k$ and $\perp$, the empty failure detector that never outputs anything: $\perp$ can thus be implemented in an asynchronous system. Through these relationships, we point out some situations in which $\mathcal{P}^k$ can be implemented in an asynchronous system.

Proposition 3.4 $\forall t \in [0, n-1],\ \mathcal{P}^{n-t-1} \equiv_t \perp$.

Proposition 3.5 $\forall t, t' \in [0, \lceil n/2 \rceil - 1],\ \mathcal{P}^{t'} \equiv_t \perp$.

To get an intuition of the implementability of $\mathcal{P}^k$ in an asynchronous system (i.e., the equivalence with $\perp$), consider a system of 5 processes and an environment where a majority of the processes are correct. We can implement $\mathcal{P}^2$ if up to 2 processes can crash as follows: processes periodically exchange messages, and every process waits for 3 messages and suspects the processes from which it did not receive messages (the complete algorithm is given in the appendix).

Given that (as we show in the next section), for any $t \in [0, n]$, $\mathcal{P}^t$ is the weakest for register in $\mathcal{E}_t$, clearly, $\mathcal{P}^k$ cannot be implemented in an asynchronous system in any $\mathcal{E}_t$ where $k \geq t \geq n/2$.

4 The sufficient conditions

In this section, we show that, in any environment $\mathcal{E}_t$, (1) any failure detector of $\mathcal{P}^t$ implements a register, and (2) any failure detector of $\mathcal{P}^t \times \Diamond\mathcal{S}$ implements consensus. To show (2), we reuse the fact that consensus can be implementable with registers and any failure detector of $\Diamond\mathcal{S}$ (in any environment) [19]. To show (1), we first give an algorithm (Figure 1) that implements a 1-writer–1-reader atomic register (for any reader or writer) using a failure detector of $\mathcal{P}^t$. Then, as in [1], we use the fact that a N-writer–M-reader atomic register can be implemented from 1-writer–1-reader atomic registers (see [15] for a tutorial on relationships between registers).

Our register implementation (Figure 1) is an adaptation of [3]. Roughly speaking, whereas [3] uses the assumption of a majority of correct processes to ensure a quorum property for read and write operations, we make use of $\mathcal{P}^t$ to ensure that quorum property. Basically, each process maintains the current value of the register. In order to perform its read (resp write) operation, the reader (resp the writer) sends a message to all and waits until it receives acknowledgments from (1) at least $max(n-t, 1)$ processes, and (2) from every process that is not suspected by its failure detector module.

Proposition 4.1 *With any failure detector of $\mathcal{P}^t$, Algorithm 1 implements a 1-reader–1-writer atomic register (for any reader or writer) in environment $\mathcal{E}_t$.*

Corollary 4.2 *Any failure detector of $\mathcal{P}^t$ implements a (n-writer–n-reader) register in $\mathcal{E}_t$.*

Corollary 4.3 *Any failure detector of $\mathcal{P}^t \times \Diamond\mathcal{S}$ implements consensus in $\mathcal{E}_t$.*

[3] Remember that we consider realistic restrictions.

```
1  Every process p (including p_w and p_r) executes the following code:
2    Initialization:
3      current := ⊥
4      last_write := −1
5      upon receive (WRITE, y, s) from the writer
6      if s > last_write then
7        current := y
8        last_write := s
9        send(ACK_WRITE, s) to the writer
10     upon receive (READ, s) from the reader
11       send(ACK_READ, last_write, current, s) to the reader
12 Code for p_w the (unique) writer:
13   Initialization:
14     seq := 0 /* sequence number */
15   procedure write(x)
16     send(WRITE, x, seq) to all
17     wait until receive (ACK_WRITE, seq)
           from all processes not in P^t_{p_w}
           and from at least max(n − t, 1) processes
18     seq := seq + 1
19   end write
20 Code for p_r the (unique) reader:
21   Initialization:
22     rc := 0 /* reading counter */
23   function read()
24     send(READ, rc) to all
25     wait until receive (ACK_READ, ∗, ∗, rc)
           from all processes not in P^t_{p_r}
           and from at least max(n − t, 1) processes
26     a := max{v | (ACK_READ, v, ∗, rc) is a received message}
27     if a > last_write then
28       current := v such that (ACK_READ, a, v, rc) is a received message
29       last_write := a
30     return(current)
31   end read
```

Figure 1: Implementation of an 1-writer–1-reader atomic register

5 The necessary conditions

In this section, we show that, in any environment $\mathcal{E}_t$, (1) any failure detector class that implements a register is stronger than $\mathcal{P}^t$, and (2) any failure detector class that implements consensus is stronger than $\mathcal{P}^t \times \Diamond\mathcal{S}$.

To state (1) we give an algorithm in Figure 2 that emulates, with any failure detector $\mathcal{D}$ that implements a 1-writer–n-reader register for any writer, a failure detector of $\mathcal{P}^t$. We only describe the algorithm: its proof is given in the appendix. To state (2), we use (1) above plus the very fact that, (2.1) one can implement a register using (uniform) consensus (e.g., through a fault-tolerant state machine replication approach [24]), and (2.2) any failure detector class that implements a consensus is stronger than $\Diamond\mathcal{S}$ [5].

The idea of the algorithm of Figure 2 is the following. We use exactly one 1-writer–n-reader register per process, and we also denote by p the register of process p. Each process p is the writer of its register. Periodically, each process writes in its register alternatively two different values: v_0 and v_1. The very fact that the register has been implemented using a failure detector $\mathcal{D}$ has the nice consequence that, for any write operation that terminates at some time τ, all processes that have not crashed by time τ, except at most $max(n - t - 1, 0)$ processes, have *participated* in the operation (the very fact that $\mathcal{D}$ is *realistic* is important here). A failure detector of $\mathcal{P}^t$ is emulated through a distributed variable where every process basically outputs (*suspects*) the list of processes that did not participate in a write operation. More precisely, each process p maintains, for each register r, a list of participants and the sequence number of the last write

operation seen by the process p on r. Every message exchanged in the context of an operation is tagged with these lists and the sequence numbers of the last write operation on each register. When p ends its write operation, it suspects all processes that are not in its list for p. The details on how processes update the lists, as well as the sequence numbers of the write operations, are given in Figure 2.

```
/* Process p repeatedly writes v_0, v_1 on register p */
/* Output_p is the output of the emulated failure detector */

1   Initialization:
2     ∀i : L[i] = ∅; last_write[i] := 0;

/*L[p] is the list of participants in the write (last_write[i]) operation on the register p */
/*all messages are tagged with (L, last_write)*/

3   When p begins a new write operation on its register (register p)
4     last_write[p] := last_write[p] + 1
5     L[p] := {p}

6   When p ends the current write operation on its register (register p)
7     Output_p = Π − L[p]

8   When p receives a message m with tag M, lw
9     forall i do
10      switch:
11        case lw[i] > last_write[i]:L[i] := M[i] ∪ {p}
12        case lw[i] = last_write[i]:L[i] := L[i] ∪ M[i]
13        case lw[i] < last_write[i]:skip
14      last_write[i] := max(last_write[i], lw[i])
```

Figure 2: $T_{\mathcal{D}\rightarrow\mathcal{P}^t}$ - Emulation at process p of a failure detector in $\mathcal{P}^t$ from a register implementation using a failure detector.

Proposition 5.1 *For any failure detector class $\mathcal{D}$ that implements a register in $\mathcal{E}_t$, we have: $\mathcal{P}^t \preceq_t \mathcal{D}$.*

From Proposition 5.1 and Corollary 4.2, we deduce:

Theorem 5.2 *For any t, $\mathcal{P}^t$ is the weakest failure detector class to implement a register in environment $\mathcal{E}_t$.*

Given that a register cannot be implemented in asynchronous systems in environment $\mathcal{E}_t$ for $n \leq 2t$ [1], we deduce:

Corollary 5.3 *If $t \geq n/2$ then $\mathcal{P}^t$ cannot be implemented in an asynchronous system in $\mathcal{E}_t$.*

Let $\mathcal{D}$ be any failure detector that implements consensus in $\mathcal{E}_t$. From [5], $\mathcal{D}$ can be transformed into a failure detector of $\Diamond\mathcal{S}$ in environment $\mathcal{E}_t$.

Proposition 5.4 *For any failure detector class $\mathcal{D}$ that implements consensus in $\mathcal{E}_t$, we have: $\mathcal{P}^t \times \Diamond\mathcal{S} \preceq_t \mathcal{D}$.*

From Proposition 5.4 and Corollary 4.3, we deduce that:

Theorem 5.5 *For any t, $\mathcal{P}^t \times \Diamond\mathcal{S}$ is the weakest failure detector class to implement consensus in environment $\mathcal{E}_t$.*

6 Concluding remarks

We discuss here the scope of our lower bound failure detection results. We first discuss the universe of failure detectors among which we determine the weakest class, then we consider the environment within which we state and prove our results, and finally we come back to the relationship between registers and consensus and the impact of uniformity.

6.1 The failure detector universe

Our results hold among the original universe of failure detectors defined in [4], with one exception however: we exclude failure detectors that can guess the future [7]. These failure detectors cannot be implemented even in a synchronous system (in the sense of [20]).[4] From a theoretical perspective, excluding such failure detectors has clearly an impact. For instance, there is a class (denoted by $\mathcal{M}$ in [12]) of failure detectors that output exactly the list of faulty processes since time 0, and which implement consensus and registers for any number of possible crashes. This class is incomparable with $\mathcal{P}^t$ (this is not completely trivial and we give the proof in the appendix A-6.1) Hence, our results do not hold within the overall original universe of failure detectors [4]: this might explain why the questions we address in this paper remained open for more than a decade.

6.2 The environments

Strictly speaking, our failure detection lower bound result on consensus does not generalize the result of [5]. Whereas [5] shows that any failure detector that implements consensus in *any* environment can be transformed into a failure detector of $\Diamond\mathcal{S}$, we show that any failure detector that implements consensus in *any* environment where up to t processes can crash ($\mathcal{E}_t$), for *any* t, can be transformed into a failure detector of $\mathcal{P}^t \times \Diamond\mathcal{S}$. Our result generalizes the result of [5] for environments of the form $\mathcal{E}_t$. It is not applicable to arbitrary environments, say where two specific processes p_1 and p_2 cannot crash in the same failure pattern.

6.3 The uniformity of consensus

We considered throughout the paper the uniform variant of consensus. If we consider a *correct-restricted* variant of consensus [23], where two process can decide differently, as long as one of them is faulty, then $\mathcal{P}^t \times \Diamond\mathcal{S}$ is not the weakest for consensus for $t \geq n/2$ in $\mathcal{E}_t$. Indeed, consider the class $\mathcal{P}_<$ of failure detectors (given in [10]) and defined through the strong accuracy property of $\mathcal{P}$ and the following *partial* completeness property: if a process p_i crashes, then eventually every correct process p_j such that $j > i$ permanently suspects p_i. [5] There is an algorithm given in [10] that implements correct-restricted consensus with $\mathcal{P}_<$ for any t. For $t \geq n/2$, $\mathcal{P}_<$ is not stronger than $\mathcal{P}^t$ in $\mathcal{E}_t$ (this is not completely trivial and we give the proof in the appendix A-6.2). For $t \geq n/2$, $\mathcal{P}^t \times \Diamond\mathcal{S}$ is thus not the weakest for consensus in $\mathcal{E}_t$ (we generalize here the observation of [7] for $t \in \{n-1, n\}$). In a sense, the equivalence we state in the paper between consensus in $\mathcal{E}_t$ and $\Diamond\mathcal{S}$ plus a register resilient to t crashes does not hold for *correct-restricted* consensus, i.e., the latter does not always lead to implement a register in a message passing model (augmented with failure detectors).

Acknowledgements

Comments from Partha Dutta, Corine Hari, Idit Keidar, Petr Kouznetsov, and Bastian Pochon helped improve the quality of the presentation of this paper.

References

[1] H. Attiya and J. Welch. *Distributed Computing. Fundamentals, Simulations, and Advanced Topics.* McGraw-Hill, 1998.

[2] K. Birman and R. van Renessee. *Reliable Distributed Computing with the Isis Toolkit.* IEEE Computer Society Press, 1993.

[4] Systems as in [8] that *enforce* perfect failure detection by explicitly crashing processes, make sure that these processes are suspected after they have crashed, i.e, they do not predict the future. It is not clear however how our results apply to systems that somehow predict failures by turning suspicions into exclusions from the group [2].

[5] As with the failure detector class Ω_i, introduced in [22], $\mathcal{P}_<$ has a restricted completeness: some faulty process might never be suspected. However, the two classes are different: with $\mathcal{P}_<$, completeness is restricted *a priori*: the only faulty process that might never be suspected (by any one) is p_n. With Ω_i, except when $i = 1$, any faulty process might never be suspected (by any one).

[3] H. Attiya, A. Bar-Noy, and D. Dolev. *Sharing Memory Robustly in Message Passing Systems.* Journal of the ACM, 42(1), January 1995.

[4] T. Chandra and S. Toueg. *Unreliable Failure Detectors for Reliable Distributed Systems.* Journal of the ACM, 43(2), March 1996.

[5] T. Chandra, V. Hadzilacos and S. Toueg. *The Weakest Failure Detector for Solving Consensus.* Journal of the ACM, 43(4), July 1996.

[6] C. Dwork, N. Lynch and L. Stockmeyer. *Consensus in the presence of partial synchrony.* Journal of the ACM, 35(2), 1988.

[7] C. Delporte-Gallet, H. Fauconnier and R. Guerraoui. *A Realistic Look at Failure Detectors.* Proceedings of the IEEE International Conference on Dependable Systems and Networks, Washington DC, June 2002.

[8] C. Fetzer. *Enforcing Perfect Failure Detection.* Proceedings of the IEEE International conference on Distributed Computing Systems, Phoenix, April 2001.

[9] M. Fischer, N. Lynch and M. Paterson. *Impossibility of Distributed Consensus with One Faulty Process.* Journal of the ACM, 32(2), 1985.

[10] R. Guerraoui. *Revisiting the Relationship Between the Atomic Commitment and Consensus Problems.* Proceedings of the International Workshop on Distributed Algorithms, Springer Verlag (LNCS 972), 1995.

[11] R. Guerraoui and A. Schiper. Γ-*Accurate Failure Detectors.* Proceedings of the International Workshop on Distributed Algorithms, Springer Verlag (LNCS 1151), 1996.

[12] R. Guerraoui. *On the Hardness of Failure Sensitive Agreement Problems.* Information Processing Letters, 79, 2001.

[13] V. Hadzilacos. *On the Relationship Between the Atomic Commitment and Consensus Problems.* Proceedings of the International Workshop on Fault-Tolerant Distributed Computing, Springer Verlag (LNCS 448), 1986.

[14] J. Halpern and A. Ricciardi. *A Knowledge-Theoretic Analysis of Uniform Distributed Coordination and Failure Detectors.* Proceedings of the ACM Symposium on Principles of Distributed Computing, 1999.

[15] P. Jayanti. *Wait-free Computing.* Proceedings of the International Workshop on Distributed Algorithms, Springer Verlag (LNCS 972), 1995.

[16] L. Lamport. *Concurrent Reading and Writing.* Communications of the ACM, 20(11), 1977.

[17] L. Lamport. *Time, clocks and the ordering of events in a distributed system.* Communications of the ACM, 21(7), 1979.

[18] L. Lamport. *On Interprocess Communication (parts I and II).* Distributed Computing, 1, 1986.

[19] W-K. Lo and V. Hadzilacos. *Using Failure Detectors to Solve Consensus in Asynchronous Shared-Memory Systems.* Proceedings of the International Workshop on Distributed Algorithms, Springer Verlag (LNCS 857), 1994.

[20] N. Lynch. *Distributed Algorithms.* Morgan Kaufmann, 1996.

[21] A. Mostéfaoui and M. Raynal. *k-Set Agreement with Limited Accuracy Failure Detectors.* Proceedings of the ACM Symposium on Principles of Distributed Computing, 2000.

[22] G. Neiger. *Failure Detectors and the Wait-Free Hierarchy.* Proceedings of the ACM Symposium on Principles of Distributed Computing, 1995.

[23] G. Neiger and S. Toueg. *Simulating Synchronized Clocks and Common Knowledge in Distributed Systems.* Journal of the ACM, 40(2), April 1993.

[24] F. Schneider. *Replication Management using the State Machine Approach.* Chapter in Distributed Systems, Addison-Wesley, 1993.

Optional Appendix

A-1 Preliminary

Given a run $R =< F, H, C, S, T >$, for presentation simplicity, we assume that the time T is the set of positive integers. Given $R =< F, H, C, S, T >$, the partial run of R up to time τ is $R_\tau =< F, H', C, S', T' >$ such that $T' = \{1, \ldots, \tau\}$, H', the partial history up to time τ, is the restriction to T' of the failure history H, and S' is the initial segment up to τ of S ($S' = S[1]; \ldots; S[\tau]$). [6] Note that, if the failure detector is realistic, then $R'_\tau =< F', H', C, S', T' >$ is also a partial run if F and F' are identical up to time τ.

Most of the following impossibility proofs are based on standard partitioning arguments. In these arguments, we exhibit some indistinguishable runs. Intuitively, two partial runs R and R' are indistinguishable for a process p, that does not crash in R and R', if and only if (1) p takes steps at the same time in R and R', (2) failure detectors histories are the same for p in R and R', and (3) p receives the same messages in R and R'. In this case, at each time, the state of p are the same in R and R'.

Given a partial run up to time τ, $R_\tau =< F, H, C, S, T >$, and a set of processes E, $S\lceil E$ will denote the restriction of S to E.

Let E_0 and E_1 be two sets of processes, H_0 a function from $E_0 \times \{1, \ldots, \tau_0\}$ to 2^Π, and H_1 a function from $E_1 \times \{1, \ldots, \tau_1\}$ to 2^Π. Given a failure detector $\mathcal{D}$ and a failure pattern F, we say that H_0 and H_1 are *compatible* if there exists a history G in $\mathcal{D}(F)$ such that for all $(p, \tau) \in E_0 \times \{1, \ldots, \tau_0\}$ $G(p, \tau) = H_0(p, \tau)$ and for all $(p, \tau) \in E_1 \times \{1, \ldots, \tau_1\}$ $G(p, \tau) = H_1(p, \tau)$. Intuitively, H_0 and H_1 are possible partial histories for processes in E_0 and E_1 and if they are compatible there exists an history that gives these two partial histories for processes in E_0 and E_1.

Then, roughly speaking, the partitioning argument proceeds as follows. Let A and $B = \Pi \setminus A$ be two sets of processes and R and R' two partial runs up to time τ_0. If (1) up to time τ_0, the failure patterns of R and R' are identical, (2) at time $\tau \leq \tau_0$ some process in A receives a message m from B in R, then at time τ, m has been sent in R', (3) at time $\tau \leq \tau_0$ some process in B receives a message m from A in R', then at time τ, m has been sent in R, and (4) the restriction to A of the partial history of R is compatible with the restriction to B of the partial history of R', then there exists a run R'' such that $R''\lceil A = R\lceil A$ and $R''\lceil B = R'\lceil B$.

Note that if the failure detector is not realistic, condition (1) is not sufficient: we have to ensure that the *whole* failure patterns of R and R' are the same.

A-2 Proofs of Section 2

Before proving the fact that the class of realistic failure detectors is closed under reduction, precise some definitions.

Consider environment $\mathcal{E}_t$, recall that algorithm A is a reduction algorithm of failure detector $\mathcal{D}$ into failure detector $\mathcal{D}'$ if and only if, given a failure pattern $F \in \mathcal{E}_t$ and H any failure history in $\mathcal{D}(F)$, $A(H)$, the histories of outputs of A is a failure history in $\mathcal{D}'(F)$. In fact, the reduction algorithm A using the failure detector $\mathcal{D}$ implements the failure detector $A_\mathcal{D}$ defined as follows: for each failure pattern F, $A_\mathcal{D}(F)$ is the set $\{A(H) | H \in \mathcal{D}(F)\}$. Remark that $\mathcal{D}' \preceq \mathcal{D}$ if and only if for every failure patterns F, $A_\mathcal{D}(F) \subseteq \mathcal{D}'(F)$.

The notion of reduction has been defined for the general class of failure detectors. The following propositions prove that this notion can be restricted to the class of realistic failure detectors.

Lemma 1 *If $\mathcal{D}$ is a realistic failure detector and A is a reduction algorithm, then $A_\mathcal{D}$ is a realistic failure detector.*

Proof. Consider F and F' two failure patterns identical up to time τ_0, and $H \in A_\mathcal{D}(F)$. Let $R =< F, H_0, C, S, T >$ be the run of A that output H. As $\mathcal{D}$ is realistic there exists some H_1, an history in $\mathcal{D}(F')$, such that for all p and all $\tau \leq \tau_0$ $H_0(p, \tau) = H_1(p, \tau)$. Then there exists a run $R' =< F', H_1, C, S', T >$ of A such that the partial run of R' up to time τ_0 and the partial run of R up to time τ_0 are indistinguishable

[6] Note that in such a partial run F is a failure pattern for the *whole* run: $F(\tau)$ is defined for all integer τ.

and then output the same history up to time τ_0. Thus, if H' is the output of R', then for all $\tau \leq \tau_0$, for all $H(p,\tau) = H'(p,\tau)$. ■

In the following, and only to the end of this section, we do not assume that the failure detectors are realistic. Let $\mathcal{R}eal$ be the class of all realistic failure detectors.

Proposition 2 *Let $\mathcal{D}_2$ and $\mathcal{D}_1$ be two failure detectors (not necessary realistic). For all t, if $\mathcal{D}_2 \preceq_t \mathcal{D}_1$ then $\mathcal{D}_2 \cap \mathcal{R}eal \preceq_t \mathcal{D}_1 \cap \mathcal{R}eal$*

Proof. Let A a reduction algorithm of $\mathcal{D}_1$ into $\mathcal{D}_2$. As $\mathcal{D}_1 \cap \mathcal{R}eal \subseteq \mathcal{D}_1$, then $A_{\mathcal{D}_1 \cap \mathcal{R}eal} \subseteq \mathcal{D}_2$. From Lemma 1, $A_{\mathcal{D}_1 \cap \mathcal{R}eal}$ is realistic too and then $A_{\mathcal{D}_1 \cap \mathcal{R}eal} \subseteq \mathcal{D}_2 \cap \mathcal{R}eal$ proving that $\mathcal{D}_2 \cap \mathcal{R}eal \preceq_t \mathcal{D}_1 \cap \mathcal{R}eal$. ■

A-3 Proofs of Section 3

Proposition 3.2 is a general property that is not used elsewhere in the paper. For the sake of presentation we prove it here although its proof uses the Corollary 4.2 that will be proved later (Section A-4).

Proposition 3.2 $\forall t, t' \in [n/2, n-1]$, $t' < t \Rightarrow \mathcal{P}^{t'} \prec_t \mathcal{P}^t$.

Proof. We prove first the following lemma:

Lemma 3 *If $t \in [n/2, n-1]$ then there is no algorithm implementing an atomic 1-writer–1-reader atomic register with $\mathcal{P}^{t-1}$ in $\mathcal{E}_t$ environment.*

Proof of lemma. Assume by contradiction Lemma 3 is false. The proof uses a classical partition argument.

Let p_w be the writer and p_r the reader. As $n \leq 2t$, then there is a set W of $n-t$ processes containing p_w and a set R of $n-t$ processes containing p_r such that W and R are disjoint sets. As $t \leq n-1$, these sets are not empty. In the proof, we assume that all processes not in W or in R are initially crashed.

Let v_0 be the initial value of the register.

Consider the following runs:

- In α_0, only processes in W are alive, and p_w writes v_1 ($v_1 \neq v_0$). The failure detectors of all processes in W always suspect all processes in $\Pi \setminus W$. At time τ_0, the write operation completes.
- α_1 is similar to α_0, except that all processes in R are alive, but do not take any step until time τ_0. The failure detectors of all processes in W (respectively in R) suspect always all processes in $\Pi \setminus W$ (respectively in $\Pi \setminus R$). This is possible because, in α_1, we have $2(n-t)$ alive processes and therefore at most $n-t$ alive processes are suspected (remember that the failure detectors are in $\mathcal{P}^{t-1}$). For processes of W, α_1 and α_0 are indistinguishable, and so at time τ_0, the write operation completes.
- α_2 is similar to α_1, except that all processes in W and R do not take any step until time τ_0. The failure detectors of all processes in W (respectively in R) always suspect all processes in $\Pi \setminus W$ (respectively in $\Pi \setminus R$). At time τ_0, all processes of W crash, then p_r begins the read operation at time τ_0 and the read operation completes at time τ_1. Clearly the value returned by the read operation must be the initial value v_0.
- In α_3 all processes in W have the same behavior as in α_1 until time τ_0. All messages sent between W and R are delayed until after time τ_1. Processes in R have the same behavior as in α_2 and the read operation completes at time τ_1. By the definition of an atomic register, this read must return v_1, but for processes in R, α_3 is indistinguishable from α_2 and the read return v_0. Contradiction.

■

Clearly $\mathcal{P}^{t'} \preceq_t \mathcal{P}^t$. In environment $\mathcal{E}_t$, we can implement a 1-writer–1-reader atomic register with $\mathcal{P}_t$ (Proposition 4.1) but by the previous lemma we can not implement this register with $\mathcal{P}^{t-1}$. ■

Now we prove that, in environment $\mathcal{E}_t$, $\mathcal{P}^t \times \Diamond\mathcal{S}$ is strictly weaker than $\mathcal{S}$. Note that the proof of Proposition 3.3 uses the Theorem 5.2 that will be proved later (Section A-5) but, as one can easily verify, Proposition 3.3 will not be used elsewhere in the paper.

Proposition 3.3 $\forall t \in [1, n-2]$, $\mathcal{P}^t \times \Diamond\mathcal{S} \prec_t \mathcal{S}$.

Recall that we consider here only realistic failure detectors.

Proof. First, from Theorem 5.2, as $\mathcal{P}^t \times \Diamond\mathcal{S}$ is the weakest realistic failure detector to solve consensus in $\mathcal{E}_t$, then we have $\mathcal{P}^t \times \Diamond\mathcal{S} \preceq_t \mathcal{S}$.

We prove now by contradiction that $\mathcal{P}^t \times \Diamond\mathcal{S}$ is not stronger to $\mathcal{S}$. Assume that A is a reduction algorithm that constructs $\mathcal{S}$ from $\mathcal{P}^t \times \Diamond\mathcal{S}$ in $\mathcal{E}_t$.

Consider the following two cases:

1. case $n \leq 2t$

   - Consider the two following failure patterns F_1 such that $p_1, p_2, \ldots, p_{n-t-1}$ are the only faulty processes (and initially crash) and FF the failure free pattern. Let H_1 be the history of failure detector of $\mathcal{P}^t$, such that all processes in $\{p_1, \ldots, p_{n-t-1}\}$ are the only suspected processes. Let K_1 be the history of failure detector of $\Diamond\mathcal{S}$, such that all processes except p_n are the only suspected processes. Both H_1 and K_1 are compatible with FF and F_1.

     Note that, as $n-1 > t$, then H_1 contains always at least one process.

     (a) Consider a run R'_1 of the reduction algorithm with failure pattern F_1 and failure detector history (H_1, K_1). By hypothesis, the output of the emulated failure detector is in $\mathcal{S}$. By the completeness property of $\mathcal{S}$, there exists a time τ_1 after which all processes in $\{p_1, p_2, \ldots, p_{n-t-1}\}$ are suspected.[7]

     (b) Consider now a similar run R_1 but with failure pattern FF. For R_1, processes in $\{p_1, p_2, \ldots, p_{n-t-1}\}$ take no steps until time τ_1, and processes in $\{p_{n-t}, \ldots, p_n\}$ take the same steps at exactly the same time as in R'_1. Therefore all processes output the same history for the emulated failure detector until time τ_1 as in R'_1.

   - Consider the following failure pattern F_2 such that $p_{n-t}, \ldots, p_{2(n-t-1)}$ are the only faulty processes and all the processes crash after time τ_1. Let H_2 be the history of failure detector $\mathcal{P}^t$, identical to H_1 until time τ_1 and then all processes in $\{p_{n-t}, \ldots, p_{2(n-t-1)}\}$ are the only suspected processes. Both H_2 and K_1 are compatible with FF and F_2.

     Note that, as $n-1 > t$, then H_2 contains always at least one process.

     (a) Consider the run R'_2 of the reduction algorithm with failure pattern F_2 and the history (H_2, K_1): R'_2 is identical to R_1 until time τ_1. By the completeness property of $\mathcal{S}$, there exists a time τ_2 after which $\{p_{n-t-1+1}, \ldots, p_{2(n-t-1)}\}$ are suspected.

     (b) Consider now a similar run R_2 but with failure pattern FF. For R_2, before time τ_1, processes take steps at exactly the same time as in R'_2. Processes in $\{p_{n-t-1+1}, \ldots, p_{2(n-t-1)}\}$ take no steps between τ_1 and τ_2, and processes in $\{p_1, p_2, \ldots, p_{n-t-1}\}$ take the same steps at exactly the same time as in R'_2. Therefore all processes output the same history for the emulated failure detector until time τ_2 as in R'_2.

   - Consider the following failure pattern F_3 such that $p_{2(n-t-1)+1}, \ldots, p_n$ are the only faulty processes and they all crash after time τ_2. As $t < n-1$, these processes may be faulty. Let H_3 be the history of failure detector $\mathcal{P}^t$, identical to H_2 until time τ_2 and then all processes in $\{p_{2(n-t-1)+1}, \ldots, p_n\}$ are the only suspected processes. Let K_3 be the history of failure detector $\Diamond\mathcal{S}$, identical to K_1 until τ_2, and then all processes except p_1 are the only suspected processes. Both H_3 and K_3 are compatible with FF and F_3.

     Consider the run R_3 of the reduction algorithm with failure pattern F_3 and the history (H_3, K_3): R_3 is identical to R_2 until time τ_2. Therefore all processes output the same history for the emulated failure detector until time τ_2 as in R_2. In F_3, $p_{2(n-t-1)+1}, \ldots, p_n$ are not correct. Before time τ_1, some process suspects $\{p_1, p_2, \ldots, p_{n-t-1}\}$, between τ_1 and τ_2 some process suspects $p_{n-t}, \ldots, p_{2(n-t-1)}$. Thus, R_3 outputs history H in which all correct processes are suspected: contradicting the fact that H is an history of $\mathcal{S}$.

[7] We abuse, here and after, of the language. More precisely, any process p in $\{p_{n-t}, \ldots, p_n\}$ has made at least one step in this run before time τ_1. In this step and the followings the output of the emulated failure detector for p include $\{p_1, p_2, \ldots, p_{n-t-1}\}$ in its list of suspected processes.

```
1  Every process p executes the following code:
2    Initialization:
3      r:=0
4    Task 1:
5      repeat forever
6        send(ARE_YOU_ALIVE, r) to all
7        wait until receive (I_AM_ALIVE, r) from max(n − t, 1) processes
8        Output_p:={q | no message (I_AM_ALIVE, r) from q received by p }
9        /* Output_p is the output for p of the failure detector D */
10       r:=r+1
11   Task 2:
12     upon receive (ARE_YOU_ALIVE, x) from q
13     send(I_AM_ALIVE, x) to q
```

Figure 3: Implementation of $\mathcal{P}^{n-t-1}$ in environment $\mathcal{E}_t$

2. case $2t < n$

   We can implement $\mathcal{P}^t$ in $\mathcal{E}_t$. We have only to show that $\Diamond\mathcal{S}$ is not stronger than $\mathcal{S}$ in $\mathcal{E}_t$.

   The proof is identical to the previous proof except we consider $\lceil n/t \rceil$ sets: $\{p_1, \ldots, p_t\}$, $\{p_{t+1}, \ldots, p_{2t}\}$, $\ldots$, $\{p_{(t-1)\lceil n/t \rceil+1}, \ldots, p_n\}$ instead of $\{p_1, \ldots, p_{n-t+1}\}$, $\{p_{n-t-1+1}, \ldots, p_{2(n-t-1)}\}$, $\{p_{2(n-t-1)+1}, \ldots, p_n\}$. As $t > 0$, each of these sets contains at least one process. Furthermore, each of them has at most t processes and these processes can be faulty.

■

The following propositions are easier to derive:

Proposition 3.4 $\forall t \in [0, n-1]$, $\mathcal{P}^{n-t-1} \equiv_t \perp$.

Proof. Algorithm of Figure 3 implements a failure detector $\mathcal{D}$ in $\mathcal{P}^{n-t-1}$ within $\mathcal{E}_t$.

The completeness of $\mathcal{D}$ is clear. When p waits from $max(n-t, 1)$ processes, it can miss at most t alive processes: $|\mathcal{D}(p, \tau) \setminus F(\tau)| \leq t$. Therefore $\mathcal{D}$ ensures the $(n-t-1)$-accuracy property. ■

Proposition 3.5 $\forall t, t' \in [0, \lceil n/2 \rceil - 1]$, $\mathcal{P}^{t'} \equiv_t \perp$.

Proof. Note that, if $2t < n$, then $t \leq n-t-1$ and by Proposition 3.4 $\mathcal{P}^{n-t-1}$ can be implemented in $\mathcal{E}_t$. Moreover, from the $(n-t-1)$-accuracy property, we have $|\mathcal{P}^{n-t-1}(p, \tau) \setminus F(\tau)| \leq t$ and then $|\mathcal{P}^{n-t-1}(p, \tau) \setminus F(\tau)| \leq n - (\lceil n/2 \rceil - 1) - 1$. Therefore a failure detector in $\mathcal{P}^{n-t-1}$ satisfies $(\lceil n/2 \rceil - 1)$-accuracy property too and is clearly in $\mathcal{P}^{(\lceil n/2 \rceil - 1)}$. By Proposition 3.1, $\mathcal{P}^{t'}$ can be implemented for all t' in $[0, \lceil n/2 \rceil - 1]$ within $\mathcal{E}_t$. ■

A-4 Proofs of Section 4

Proposition 4.1 *With any failure detector of $\mathcal{P}^t$, Algorithm 1 implements a 1-reader–1-writer atomic register (for any reader or writer) in environment $\mathcal{E}_t$.*

Consider e a write or a read operation, e_{begin} denotes the first event (Line 16 for a write operation, Line 24 for a read operation) and e_{end} the last event (Line 18 for a write operation, Line 30 for a read operation).

For the reader and the writer, the end of the operation occurs after the receipt of some acknowledgment messages. If e is a write operation (respectively a read operation), then Q_e is the set of processes from which the writer (resp. the reader) has received an acknowledgment message in Line 17 (resp. in Line 25) before completing e.

We have directly:

Lemma 4 *For each process, the values of its last_write variable are nondecreasing.*

Proof. Following from the algorithm. ■

Lemma 5 *For each read or write operation e (1) At most $max(n-t-1,0)$ alive processes at time $\tau(e_{end})$ are not in Q_e, (2) All processes in Q_e were alive at time $\tau(e_{begin})$, and (3) Q_e contains at least $max(n-t,1)$ processes.*

Proof. Follows from the properties of failure detector $\mathcal{P}^t$ and from the algorithm. ■

Lemma 6 *Let e and e′ be any two write or read operations, such that $\tau(e_{end}) < \tau(e'_{begin})$, then $Q_e \cap Q_{e'}$.*

Proof. Consider Lemma 5: By (2), all processes in $Q_{e'}$ were alive at time $\tau(e'_{begin})$. By (3), at least $max(n-t,1)$ such processes are in $Q_{e'}$. As, by hypothesis, $\tau(e_{end}) < \tau(e'_{begin})$, these processes were also alive at time $\tau(e_{end})$. By (1), at most $max(n-t-1,0)$ processes alive at time $\tau(e_{end})$ are not in Q_e and thus at least one process is in $Q_e \cap Q_{e'}$. ■

Then we can apply the previous lemmas to write and read operations:

Lemma 7 *Let w be a write operation and r a read operation, if $\tau(w_{end})$ is less than $\tau(r_{begin})$, then the sequence number of w is less or equal to the value of the last_write variable of the reader at the end of r.*

Proof. Consider Q_w and Q_r. By Lemma 6, at least one process, say p, belongs to Q_w and Q_r. Process p has written the value of the sequence number of w into its *last_write* variable before time $\tau(w_{end})$.
From Lemma 4, as $\tau(w_{end}) < \tau(r_{begin})$, p has sent during r to the reader an ACK_READ message with a *last_write* value larger or equal to the sequence number of w. ■

Lemma 8 *If r and r′ are two read operations such that $\tau(r_{end})$ is less than $\tau(r'_{begin})$, then the last_write value of the reader at the end of r is less or equal than the one at the end of r′.*

Proof. Direct from Lemma 4. ■
Proof of Proposition. Now we prove Proposition 4.1.

First, note that, as at most t processes have crashed, the writer and the reader are ensured to receive an acknowledgment message from at least $max(n-t,1)$ processes (an alive process can always receive a message from itself) and then, due to the completeness of $\mathcal{P}^t$ the **wait** instructions (Line 17 and Line 25) terminate. Therefore, the write and read operations terminate if respectively the writer and the reader are correct.

As sequence numbers of write operation are strictly increasing and only the writer modifies this sequence number, we deduce from Lemma 7 that each read operation returns the value written by the most recent preceding write operation or a value written by a write operation that is concurrent with this read operation.

Consider r and r' any two read operations such that r ends before the beginning of r'. By Lemma 8, if r and r' returns respectively values from write operations w and w', then either $w = w'$ or w ends before the beginning of w'. ■

We use the fact that a N-writer–M-reader atomic register can be implemented from 1-writer–1-reader atomic registers in order to deduce the following corollary:

Corollary 4.2 *Any failure detector of $\mathcal{P}^t$ implements a (n-writer–n-reader) register in $\mathcal{E}_t$.*

We reuse the result of [19] that consensus can be implementable with registers and any failure detector of $\Diamond\mathcal{S}$ (in any environment) in order to deduce the following corollary:

Corollary 4.3 *Any failure detector of $\mathcal{P}^t \times \Diamond\mathcal{S}$ implements consensus in $\mathcal{E}_t$.*

A-5 Proofs of Section 5

Proposition 5.1 *For any failure detector class $\mathcal{D}$ that implements a register in $\mathcal{E}_t$, we have: $\mathcal{P}^t \preceq_t \mathcal{D}$.*

Proof. We prove that the reduction algorithm of Figure 2 gives $\mathcal{P}^t$.

Consider a write operation performed by writer p and let w_{begin} (resp. w_{end}) the event on p corresponding to the beginning (resp. to the end) of this write operation. As before, $\tau(w_{begin})$ and $\tau(w_{end})$ are the time of occurrence of these events.

Define W as the set of processes that could have participated in the write operation. More precisely, $E_w = \{e : w_{begin} \rightarrow e \rightarrow w_{end}\}$ and $W = \{q \in \Pi \mid \exists e \text{ on process } q \in E_w\}$ where $\rightarrow$ is the causality relation of [17].

Clearly, at the end of a write operation of process p on register p, $L[p]$ is equal to W. Therefore *Output*, the output of the emulated failure detector, is, from the end of this write until the next end of a write operation of p, the set $\Pi \setminus W$.
In order to conclude the proof, we state the following Lemma:

Lemma 9 *(1) At most $max(n-t-1, 0)$ alive processes at time $\tau(w_{end})$ are not in W, and (2) W contains no process that has crashed at time $\tau(w_{begin})$.*

Proof of Lemma.

Part (2) of the lemma is clear by the definition of W.

The proof of (1) uses a classical partitioning argument. Assume by contradiction that (1) is false. This means that there is a run α with a write operation w such that if $t \leq n-1$, at least $n-t$ alive processes at time $\tau(w_{end})$ are not in W, if $t = n$ at least one alive process at time $\tau(w_{end})$ are not in W. In all cases, at least one process is not in W. Assume without loss of generality that the register contains the value v_0 and v_1 is the new value written in the register by w.

Let P_w be the strict causal past of E_w. More precisely, $P_w = \{x \notin E_w \mid \exists a \in E_w : x \rightarrow a\}$. Now consider the following runs:

- α_0: α_0 is similar to α, except that all events causally after $E_w \cup P_w$ are delayed until after time $\tau(w_{end})$.
- α_1: α_1 is similar to α_0, but in α_1 all steps causally after P_w are delayed until after time $\tau(w_{end})$. In particular no steps concerning the write operation are taken. At time $\tau(w_{end})$, all processes in W crash. Let q be any process not belonging to W. In α_1, q begins to read the register at time $\tau(r_{begin})$ just after time $\tau(w_{end})$. As the register is atomic, and $n-t$ processes are still alive, the read operation of q has to return v_0. Let $\tau(r_{end})$ the time of the end of the read operation by q.
- α_2: α_2 is similar to α_0, but all processes of W crash at time $\tau(w_{end})$. In α_2, q begins to read the register at time $\tau(r_{begin})$. All messages between processes of W and of $\Pi \setminus W$ are delayed until after time $\tau(r_{end})$. Due to the realism hypothesis, the output of failure detectors, are the same in α_0 and α_2 until time $\tau(w_{end})$. Therefore all processes have the same behavior in α_0 and α_2 until time $\tau(w_{end})$. Then the write succeeds and by the properties of atomic register, all following reads must return v_1. In α_2, all processes of $\Pi \setminus W$ are scheduled exactly as in α_1 and the failure patterns of α_2 and α_1 are the same, therefore the output of the failure detectors is the same. Then, as in α_1, the read operation of q returns v_0 in α_2, contradicting the properties of the atomic register.

■

The proposition is a direct consequence of the Lemma. ■

Remark: If the failure detector used to implement a register is not realistic, this result does not hold. For example, consider the following failure detector: since the beginning it gives to each process the identity of the same *correct* process. Implementing an atomic register is then trivial: this correct process maintains the register and, to implement a write operation, the writer sends only a write request to this correct process and waits until it receives an acknowledgment. Clearly, only the writer and this process participates in the write operation.

From Proposition 5.1 and Corollary 4.2 we deduce:

Theorem 5.2 *For any t, $\mathcal{P}^t$ is the weakest realistic failure detector to implement an atomic register in environment $\mathcal{E}_t$*

As atomic register can not be implemented in asynchronous systems with at most t crashed processes for $n \leq 2t$, we deduce:

Corollary 5.3 *If $n \leq 2t$ then $\mathcal{P}^t$ can not be implemented in asynchronous systems.*

Proposition 5.4 *For any failure detector class $\mathcal{D}$ that implements consensus in $\mathcal{E}_t$, we have: $\mathcal{P}^t \times \Diamond\mathcal{S} \preceq_t \mathcal{D}$.*

From Proposition 5.4 and Corollary 4.3, we deduce that:

Theorem 5.5 *For any t, $\mathcal{P}^t \times \Diamond\mathcal{S}$ is the weakest failure detector class to implement consensus in environment $\mathcal{E}_t$.*

A-6 Proofs of Section 6

A-6.1 Proofs of Section 6.1

In this section we show first the impact of realism on our results. For this we prove that $\mathcal{M}$ is incomparable with $\mathcal{P}^t$, more precisely:

Lemma 10 *If $n \leq 2t$, $\neg(\mathcal{P}^t \preceq_t \mathcal{M})$ (there is no reduction algorithm from $\mathcal{M}$ to $\mathcal{P}^t$).*

Proof. Recall the definition of failure detector $\mathcal{M}$: for each process $\mathcal{M}$ outputs the exact list of faulty processes since time 0. Clearly this failure detector is not realistic. It can be easily proved that $\mathcal{M}$ enables to implement (uniform) consensus in every environment $\mathcal{E}_t$.

Assume by contradiction Lemma 10 is false. Let F be the set of $n-t$ processes $\{p_1, \ldots, p_{(n-t)}\}$ and consider failure patterns for which the set of faulty processes is F. For these failure patterns, the output of $\mathcal{M}$ is always the same. Note that as $n \leq 2t$ in such failure patterns, less than t processes are faulty.

Consider:

- In α_0, all processes in F are initially crashed. By the completeness property of $\mathcal{P}^t$, there is a time τ after which, in the output of the emulated failure detector all processes in F are suspected.
- In α_1, all processes in F crash after time τ. Moreover, processes in F do nothing until time τ. Each process not in F takes steps at the same time in α_0 and α_1. Until time τ, for processes in $\Pi \setminus F$, α_0 and α_1 are indistinguishable and so the output of the emulated failure detector is the same. In this way, at time τ, $n-t$ alive processes are suspected contradicting t-accuracy.

■

From Lemma 10 and the fact that, there is trivially no reduction algorithm from $\mathcal{P}^t$ to $\mathcal{M}$, we deduce the following:

Proposition 11 *If $n \leq 2t$, $\neg(\mathcal{M} \preceq_t \mathcal{P}^t)$ and $\neg(\mathcal{M} \preceq_t \mathcal{P}^t)$ ($\mathcal{M}$ and $\mathcal{P}^t$ are incomparable in $\mathcal{E}_t$).*

A-6.2 Proof of Section 6.3

Recall the definition of failure detector $\mathcal{P}_<$. This failure detector outputs at each process p_i a list of suspected processes such that (1) no process is suspected before it crashes, (2) if a process p_i crashes, then eventually every correct process p_j such that $j > i$ permanently suspects p_i.

Proposition 12 *If $n \leq 2t$, $\neg(\mathcal{P}^t \preceq_t \mathcal{P}_<)$ (there is no reduction algorithm from $\mathcal{P}_<$ to $\mathcal{P}^t$ in $\mathcal{E}_t$).*

Proof. We prove this by contradiction.

If $t < n$, let A be the set of processes p_i such that $1 \leq i \leq t$. If $t = n$, let A be the set of processes p_i such that $1 \leq i \leq t-1$. Therefore $\mid \Pi \setminus A \mid = max(n-t, 1)$. A $max(n-t, 1) \leq t$, then all processes in $\Pi \setminus A$ can crash.

Let F be a failure pattern where all processes of $\Pi \setminus A$ are initially crash. Let FF be the failure free pattern. Let H be a history of $\mathcal{P}_<$ in which processes of A do not suspect any process, processes of $\Pi \setminus A$ suspect processes of $\Pi \setminus A$. Let H' be a history of $\mathcal{P}_<$ in which processes do not suspect any process. H is a history of $\mathcal{P}_<$ compatible with F and H_1 is a history of $\mathcal{P}_<$ compatible with FF. For processes of A, H and H_1 are identical.

Now consider the following runs:

- Consider first, the run α_0 of the reduction algorithm with failure pattern F, and the history H. In the reduction algorithm, due to the completeness property of $\mathcal{P}^t$ there is a time τ such that the emulated failure detector suspects all processes in $\Pi \setminus A$. By construction, the set $\Pi \setminus A$ is not empty.
- Consider now a similar run α_1 but with failure pattern FF, and the history H_1. For α_1, all processes in $\Pi \setminus A$ do not take any step before time τ. All processes in A have the same history for failure detector $\mathcal{P}_<$ than in α_0. Therefore, for these processes α_0 and α_1 are indistinguishable until time τ and the emulated failure detector gives the processes in A the same output as in α_0, but they suspect $max(n-t, 1)$ alive processes contradicting the t-accuracy property of $\mathcal{P}^t$.

■

Corollary 13 *If $2t \geq n$, an atomic register can not be implemented with failure detector $\mathcal{P}_<$.*