Is Any of Us Faster than All of Us?

Early Local Decisions in Distributed Agreement *

Partha Dutta Rachid Guerraoui Bastian Pochon

Distributed Programming Laboratory
Swiss Federal Institute of Technology in Lausanne (EPFL)

Abstract

When devising a distributed agreement algorithm, it is common to minimize the time complexity of *global* decisions, which is typically measured as the number of communication rounds needed for *all* correct processes to decide. In practice, what we might want to minimize is the time complexity of *local* decisions, which we define as the number of communication rounds needed for *at least one* correct process to decide. The motivation of this paper is to figure out whether there is any difference between local and global decision bounds, and if there is, whether the same algorithm can match *both* bounds.

We address these questions for several models and various agreement problems. We show that, in a synchronous model with crash failures, the local decision bound is generally *strictly smaller* than the global decision bound, and rather surprisingly, depending on the number of failures that occur, these bounds cannot both be achieved by a single algorithm. In *synchronous* runs of the eventually synchronous model however, we show that the local decision bound is the same as the global bound. All our bounds are tight, i.e., we give optimal algorithms to match each of our bounds.

1 Introduction

Local vs Global Agreement Decisions. Determining how long it takes to reach agreement among a set of processes is an important question in distributed computing. For instance, the performance of a replicated system is impacted by the performance of the underlying consensus service used to ensure that the replica processes agree on the same order to deliver client requests [20]. Similarly, the performance of a distributed transactional system is impacted by the performance of the underlying atomic commit service used to ensure that the database servers agree on a transaction outcome [15].

Traditionally, lower bounds on the time complexity of distributed agreement have been stated in terms of the number of communication rounds (also called steps) needed for *all* correct processes to decide [22] (i.e., *global decision*), or even *halt* [10], possibly as a function of the number of failures f that actually occur, out of the total number t of failures that are tolerated.

From a practical perspective, what we might sometimes want to measure and optimize, is the number of rounds needed for *at least one* correct process to decide, i.e., for a *local decision*. Indeed, a replicated service can respond to its clients as soon as a single replica decides on a reply and knows that other replicas will reach the same decision. Similarly, the client of an atomic commit service might be happy to know the outcome of a transaction once the outcome has been determined, even if some database servers have yet to be informed of the outcome.

*Technical Report ID: IC/2003/24, School of Computer and Communication Sciences, Swiss Federal Institute of Technology (EPFL).

Motivations. Surprisingly, despite the large body of work on the performance of agreement, so far, no study on local decision lower bounds has appeared in the literature. To get an intuition of some of the ramifications underlying such a study, consider the (non-uniform) consensus problem [25, 23] in the synchronous model where a set of processes, p_1, p_2, ..., p_n, proceed by exchanging message in a round by round manner [19]. Any subset of the processes may fail, but only by crashing.

The problem consists for the processes to decide on a common final value, out of values they initially proposed, such that all correct processes eventually decide and agree on a common decision. The following algorithm (from [13]) conveys the fact that there can indeed be a difference between local and global decision lower bounds. Process p_1 decides at round 0 (i.e., at the very beginning) its own initial value, and then broadcasts this value to all. Process p_2 decides at round 1, p_1's value if p_2 receives it, or otherwise its own value. At round 2, process p_3 decides p_1's value if p_3 receives it, otherwise p_2's value if p_3 receives it, otherwise its own value, and so forth. In runs of this algorithm with at most f failures, at least one correct process decides by round f. Hence, if we denote by l_f the tight local decision lower bound for consensus in runs of the synchronous model with f failures, the very existence of the algorithm means that $l_f \leq f$. In fact, l_f is exactly f. However, if we denote by g_f the tight global decision lower bound, we know from [22] that g_f is exactly $f+1$. This observation opens several questions.

- Can we match both lower bounds with the *same* algorithm? The synchronous consensus algorithm we just sketched matches the lower bound $l_f = f$ but does obviously not match the lower bound $g_f = f + 1$. Is there any other algorithm that does so? Otherwise, we would be highlighting a rather interesting trade-off in the design of consensus algorithms.
- What is the impact of the very nature of the agreement?
  - Consider for instance the uniform variant of consensus [17], where no process disagrees with any other process, even one that crashed. Clearly, the algorithm sketched above needs to be revisited. We can relatively easily show for instance that $l_0 = 1$, and we know from [18] that $g_0 = 2$. But, is $g_f = l_f + 1$ in general, i.e., for any f?
  - Similarly, consider the non-blocking atomic commit problem [26, 17], where the processes have to decide 0 if some process proposes 0, and have to decide 1 if no process proposes 0 or crashes. In this case, it is easy to see that no algorithm can have any correct process decide at round 1, even in a run where no process crashes, i.e., $l_0 > 1$. In fact, $l_0 = 2$, which is also g_0 [9]. Does it mean that the local decision bound for non-blocking atomic commit is generally strictly larger than for uniform consensus? This would draw an interesting sharp line between the inherent time complexity of consensus and atomic commit.
- What is the impact of the model? What if we leave the purely synchronous model and consider consensus for instance in the eventually synchronous model [8], where we would compare (a) the number of rounds g_f needed for all correct processes to decide in *synchronous* runs with f process crashes, and (b) the number of rounds l_f needed for at least one correct process to decide in such runs? We can show that *no* process can decide by round 1 (given $n > 3$), and there are several algorithms where *all* correct processes decide by round 2 [18]. In other words, $l_0 = g_0 = 2$. But what about other values of f?

Contributions. This paper shows that, in the synchronous model, except for some specific values of f (which we precise in the paper), $l_f = g_f - 1$ for consensus, uniform consensus, non-blocking atomic commit, and interactive consistency. Interestingly, (non-uniform) consensus introduces a surprising trade-off: for any value of $f < t$, it is impossible for a consensus algorithm to match both l_{f+1} and g_f. These bounds are summarised in Table 1.

This paper also considers uniform consensus in the eventually synchronous model.[1] We show that, for synchronous runs of this model, the tight lower bound for local decision is the same as that for global decision. In fact, we show this result in the asynchronous round based model augmented with a *perfect* failure detector [5, 12, 4]; the result immediately extends to synchronous runs of the eventually synchronous model as well as to the synchronous model with send-omission [16]. Furthermore, in the eventually synchronous model, the only fact we knew about consensus lower bound in synchronous runs was that $g_f \geq f + 2$ [3, 18, 7]. The general question of the tight lower bound was left open [7]. We address this question by giving an optimal algorithm where $g_f = l_f = f + 2$.

Roadmap. Section 2 states and discusses our results in the synchronous model. We give our results for the eventually synchronous model in Section 3. Section 4 concludes the paper with some final remarks. We give our proofs in the optional appendices A and B, except that of Proposition 2, which we include in Section 2 as a representative case. Our lower bound proofs are devised following the layering technique of [24], also used in [18]. We slightly extend the layering scheme of [18] to deal with local decisions. (An alternate valency-based technique for proving lower bounds in the synchronous model is presented in [1].) To strengthen our results, we provide our lower bound proofs for the binary version of the agreement problems (proposal values are restricted to 0 and 1), and propose matching algorithms for the multivalued case.

2 Local Decision in the Synchronous Model

We recall below the assumptions underlying the synchronous model, and we give an overview of our results in this model.

Assumptions. We assume a distributed system model composed of $n \geq 3$ processes, $\Pi = \{p_1, p_2, \ldots, p_n\}$. Processes communicate by message passing and every pair of processes is connected by a bi-directional communication channel. Processes may fail by crashing and do not recover from a crash. Any process that does not crash in a run is said to be correct in that run; otherwise the process is faulty. In any given run, at most $t < n$ processes can fail. Processes proceed in rounds [19]. Each round consists of two phases: (a) in the *send phase*, processes are supposed to send messages to all processes; (b) in the *receive phase*, processes receive messages sent in the send phase, update local states, and (possibly) decide.

If some process p_i completes the send phase of the round, every process that completes the receive phase of the round, receives the message sent by p_i in the send phase. If p_i crashes during the send phase, then any subset of the messages p_i is supposed to send in that round may be lost.

Agreement Problems. In the *consensus* problem [25, 23] processes start with a proposal value and eventually decide on a final value such that the following properties are satisfied: (validity) if a process decides v, then some process has proposed v; (agreement) no two correct processes decide differently; and (termination) every correct process eventually decides. Binary consensus is a variant of consensus in which the proposal values are restricted to 0 and 1. *Uniform consensus* [17] is a stronger variant of consensus in which the agreement property is replaced by the following *uniform agreement* property: no two processes decide differently.

In the *non-blocking atomic commit* problem [15], each process is supposed to cast a vote of whether to abort or commit a transaction, and eventually decide. The termination and the agreement properties are the same as that for uniform consensus. Validity is defined as follows: (abort validity) abort can be

[1] In this model, neither non-blocking atomic commit nor interactive consistency is possible when $t \geq 1$, and consensus is similar to uniform consensus [13].

decided only if some process proposes to abort or fails, and (commit validity) commit can be decided only if all processes propose to commit. For presentation uniformity, we make the following changes in notations: (1) we say that a process proposes 0 (resp. 1) iff the process votes abort (resp. commit), and (2) we say that a process decides 0 (resp. 1) iff the process decides to abort (resp. to commit).

In the Interactive Consistency (IC) problem [25], each process is supposed to propose an initial value and eventually decide on a vector of values. Termination and agreement are the same as that for uniform consensus. Validity is defined as follows: for any decision vector V, the j^{th} component of V is either the value proposed by p_j or $\perp$, and may be $\perp$ only if p_j fails.

Time Complexity Metrics. Let AP be any of the agreement problems mentioned above. We say that a process decides in round $k \geq 1$ iff it decides in the receive phase of round k. For convenience of presentation, we say that a process decides at round 0 if it decides before sending any message in round 1 (i.e., apart from deciding in the receive phase of any round, a process may decide before sending any message in round 1). A run of an AP algorithm *globally decides* in round k if all correct processes decide in round k or in a *lower* round, and some correct process decides in round k. For every $0 \leq f \leq t$, we define the *global decision tight lower bound* g_f for AP, as the round number such that, every AP algorithm has a run with at most f failures, which globally decides in round g_f, or in a higher round, and there is an AP algorithm, which globally decides by round g_f in every run with at most f failures. A run of an AP algorithm *locally decides* in round k if all correct processes decide in round k or in a *higher* round and some correct process decides in round k. For every $0 \leq f \leq t$, we define the *local decision tight lower bound* l_f for AP as the round number such that, every AP algorithm has a run with at most f failures, which locally decides in round l_f or in a higher round, and there is an AP algorithm, which locally decides by round l_f in every run with at most f failures.

Overview of the Results. Our results on local decision lower bound in the synchronous model are summarized in Table 1, where they are put in perspectives with corresponding tight global decisions bounds from the literature [22, 19, 3, 18, 6]. In addition, we show that no consensus algorithm can match both g_f and l_{f+1} (for $0 \leq f \leq t-1$). Since g_{t-1} and l_t is the same for consensus and uniform consensus, no uniform consensus algorithm can match both g_{t-1} and l_t (marked $\otimes$ in Table 1).

Problem	l_f	g_f	Scope
C	f	f+1	$0 \leq f \leq t,\ 1 \leq t \leq n-1$
UC	f+1	f+2	$0 \leq f \leq t-2,\ 2 \leq t \leq n-1$
	f+1	f+1$^{\otimes}$	$f = t-1,\ 1 \leq t \leq n-1$
	f$^{\otimes}$	f+1	$f = t,\ 1 \leq t \leq n-2$
	f	f	$f = t,\ t = n-1$
NBAC/IC	2	2	$f = 0,\ 2 \leq t \leq n-1$
	f+1	f+2	$0 \leq f \leq t-2,\ 2 \leq t \leq n-1$
	f+1	f+1	$f = t-1,\ 1 \leq t \leq n-1$
	f	f+1	$f = t,\ 1 \leq t \leq n-2$
	f	f	$f = t,\ t = n-1$

Table 1: Tight global vs. local decision bounds in the synchronous model

Consensus. The algorithm (informally) presented in Section 1 implies that $l_f \leq f$; i.e., there is a consensus algorithm such that, for every $0 \leq f \leq t$, and for every run with at most f failures, some correct process decides in round f. We know from [22] that $g_f = f + 1$. Intuitively, one can easily see

that $l_f \geq g_f - 1$ (and hence, $l_f \geq f$): since some correct process p_i always decides by l_f, p_i can send its decision value to all in round $l_f + 1$, and thus, force a global decision at round $l_f + 1$.

Proposition 1 *(a) Let $1 \leq t \leq n-1$. For every consensus algorithm and for every $0 \leq f \leq t$, there is a run with at most f failures in which no correct process decides before round f. (b) Let $1 \leq t \leq n-2$. For every consensus algorithm, and for every $0 \leq f \leq t$, there is a run with at most f failures in which at most one correct process decides before round $f+1$.*

Proposition 1(a) states that $l_f \geq f$. Proposition 1(b) gives a lower bound on the number of correct processes that can decide earlier than g_f. Furthermore, Proposition 1(b) implies that, there is a run with at most f failures in which at least $n-t-1$ correct processes decide in round $f+1$ or in a higher round. When there are at least two correct processes ($t \leq n-2$), it immediately follows that some correct process decides in round $f+1$ or in a higher round. In this sense, Proposition 1(b) can also be seen as a generalization of the well-known $f+1$ global decision lower bound of [22, 19, 1].

We have already seen an algorithm that matches local decision lower bound, and we know from [22] that there is a matching algorithm for global decision lower bound. One wonders whether a single algorithm can match both local and global decision bounds. Indeed, for a given value of f ($0 \leq f \leq t$), it is easy to design an algorithm which locally decides by $l_f = f$ and globally decides by $g_f = f+1$. But in general, while designing an early deciding algorithm, we are interested in algorithms that are optimal for a large range of f, preferably for every value of f from 0 to t. Surprisingly, for any value of f, no algorithm can match both g_f and l_{f+1}. Thus, there does not exist a consensus algorithm which is optimal for both local and global decision, even for two consecutive values of f.

Proposition 2 *Let $1 \leq t \leq n-2$. For any f such that $0 \leq f \leq t-1$, no consensus algorithm can achieve both the following bounds: (a) in every run with at most f failures, every correct process decides by round $f+1$, and (b) in every run with at most $f+1$ failures, some correct process decides by round $f+1$.*[2]

Proof: *(Sketch)* A configuration at (the end of) round $k \geq 1$ in a run, is the collection of all process states at the end of round k. For any configuration C at the end of round k, $r(C)$ is a run in which the round k configuration is C, and no process crashes after round k; $val(C)$ is the decision value of correct processes in $r(C)$. We say a process is *alive* in C if it has not crashed in C. It is easy to see that a process p_i is alive in C iff p_i is correct in $r(C)$. Due to lack of space, we assume the following elements (whose explanations are based on the layering scheme, and provided in the Optional Appendix). We consider only those runs in the synchronous model in which at most one process crashes in a round. Furthermore, we assume the following (slightly modified) lemma from [18]. *For every binary consensus algorithm and $0 \leq k \leq t$, there are two k-round configurations y and y' such that (1) at most k processes have crashed in each configuration, (2) the configurations differ at exactly one process, and (3) $val(y) = 0$ and $val(y') = 1$.*

Suppose by contradiction that there is a binary consensus algorithm A and an integer f such that (a) at round $f+1$ of every run with at most f failures, every correct process decides, (b) at round $f+1$ of every run with at most $f+1$ failures, some correct process decides, and (c) $0 \leq f \leq t-1$.

From the lemma assumed above, at the end of round f there are two configurations y_0 and y_1 such that (a) at most f processes have crashed in each configuration, (b) the configurations differ at exactly one process, say p_i, and (c) $val(y_0) = 0$ and $val(y_1) = 1$.

Consider the run $r(y_0)$. Obviously, $r(y_0)$ is a run with at most f failures, and from our initial assumption, every correct process decides $val(y_0) = 0$ at the end of round $f+1$. Similarly, we construct the run $r(y_1)$ which is a failure-free extension of y_1, and every correct process decides $val(y_1) = 1$ at

[2]Note that, if $f = t$, then bound (a) implies bound (b), because there are no runs with $t+1$ failures.

the end of round $f+1$. There are two cases to consider.

Case 1. Process p_i is alive in y_0 and y_1. Consider the extension of y_0 to a run $r'(y_0)$ such that p_i crashes before sending any message in round $f+1$, and no process crashes thereafter. (Recall that $f \leq t-1$.) Notice that $r'(y_0)$ is a run with at most $f+1$ failures and p_i is a faulty process in $r'(y_0)$. Thus, from our initial assumption about A, it follows that there is a correct process $p_j (\neq p_i)$ in $r'(y_0)$ which decides some value $v \in \{0,1\}$ at round $f+1$. (Notice that, since $p_j \neq p_i$, p_j cannot decide before round $f+1$.)

Similarly, consider the extension of y_1 to a run $r'(y_1)$ such that p_i crashes before sending any message in round $f+1$, and no process crashes thereafter. Notice that, at the end of round $f+1$, p_j cannot distinguish $r'(y_1)$ from $r'(y_0)$ because p_j does not receive any message from p_i in round $f+1$ of both runs. Therefore, p_j decides v at the end of round $f+1$ in $r'(y_1)$.

Consider runs $r'(y_{1-v})$ and $r(y_{1-v})$. Since $t \leq n-2$, in run $r'(y_{1-v})$ there is correct process p_l which is distinct from p_i and p_j. Obviously, p_l is also correct in $r(y_{1-v})$, and hence, p_l decides $val(y_{1-v}) = 1-v$ at the end of round $f+1$ in $r(y_{1-v})$.

Now we construct a run r'' by extending configuration y_{1-v}: process p_i crashes at the beginning of round $f+1$ such that, in round $f+1$, p_l receives a message from p_i but p_j does not receive any message from p_i. No process distinct from p_i crashes in round $f+1$ or a higher round. Obviously, p_j and p_l are correct in r''. At the end of round $f+1$ of run r'', p_j cannot distinguish r'' from $r'(y_{1-v})$ because p_j does not receive a message from p_i in round $f+1$ in both runs. Therefore, p_j decides v at the end of round $f+1$ in r''. However, since p_l receives a message from p_i in round $f+1$, at the end of round $f+1$, p_l cannot distinguish r'' from $r(y_{1-v})$, and therefore, decides $1-v$ at the end of round $f+1$; a contradiction to consensus agreement.

Case 2. Process p_i has crashed in either y_0 or y_1. Without loss of generality, we can assume that p_i has crashed in y_0, and hence, p_i is alive in y_1. (Recall that p_i has different states in the two configurations.) Since $t \leq n-2$ and at most $f-1$ processes have crashed in y_1, there are at least two correct process p_j and p_l (distinct from p_i) in $r(y_1)$. Consider the run r' which extends y_1 such that process p_i crashes in round $f+1$ and only process p_l *does not* receive round $f+1$ message from p_i, and no process crashes after round $f+1$. Obviously p_j and p_l are correct in r'. At the end of round $f+1$, p_l cannot distinguish $r(y_0)$ from r' because p_l does not receive the round $f+1$ message from p_i in both runs. Thus, p_l decides 0 at the end of round $f+1$ in r'. At the end of round $f+1$, p_j cannot distinguish $r(y_1)$ from r' because p_j receives round $f+1$ message from p_i in both runs. Thus, p_j decides 1 at the end of round $f+1$ in r'; a contradiction. □

Uniform Consensus. The following proposition gives a local decision lower bound on uniform consensus, and generalizes the well-known $f+2$ round lower bound on uniform consensus global decision [3, 18].

Proposition 3 *(a) Let $1 \leq t \leq n-1$. For every uniform consensus algorithm and every $0 \leq f \leq t-1$, there is a run with f failures in which no correct process decides before round $f+1$. (b) Let $3 \leq t \leq n-1$. For every uniform consensus algorithm and every $0 \leq f \leq t-3$, there is a run with f failures in which at most one correct process decides before round $f+2$.*

We give (in Appendix A.2.2) two new uniform consensus algorithms that match both the local and global decision bounds (in Table 1), and thus, show that our lower bounds are tight. Since the tight lower bounds of uniform consensus are in general larger than those of consensus, Proposition 2 does not impact our uniform consensus algorithms, except for the following special case. Notice in Table 1 that g_{t-1} and l_t are the same for both consensus and uniform consensus. By Proposition 2, there is no uniform consensus algorithm that can achieve both g_{t-1} and l_t. We thus give two algorithms. The first algorithm (Figure 1, Appendix A.2.2) achieves all bounds in Table 1, except l_t: in some runs

with t failures the algorithm locally decides in $t+1$ rounds (instead of $l_t = t$). The second algorithm (Figure 2, Appendix A.2.2) achieves all bounds in Table 1, except g_{t-1}: in some runs with $t-1$ failures the algorithm globally decides in $t+1$ rounds (instead of $g_{t-1} = t$).

The l_f and g_f values for uniform consensus (in Table 1) hold even if the validity condition is replaced by the following *weak validity*: for every value $v \in \{0,1\}$, there is a failure-free run in which some correct process decides v [18]. Binary uniform consensus with weak validity is a weaker problem than non-blocking atomic commit and interactive consistency. Hence, *the lower bounds on uniform consensus immediately extends to the other two problems.*

Non-Blocking Atomic Commit (NBAC) and Interactive Consistency (IC). For the failure-free case ($f = 0$), the local decision tight lower bound for non-blocking atomic commit is strictly larger than that of uniform consensus.

Proposition 4 *Let $2 \leq t < n$. For every NBAC algorithm, there is a failure-free run in which no process decides at round 1.*

The above proposition highlights a fundamental difference between the time complexity of non-blocking atomic commit and that of uniform consensus in the synchronous model. The proposition extends to interactive consistency. In fact, any interactive consistency algorithm can be transformed to a non-blocking atomic commit algorithm (without any additional rounds) as follows. Let $V1$ denote the vector of n components in which every component is 1. Suppose we have an IC algorithm with IC-propose() primitive. We implement the AC-propose() primitive of the NBAC specification in the following way. When a process AC-proposes $v \in \{0,1\}$, it IC-proposes v. If a process IC-decides $V1$, then is AC-decides 1; otherwise it AC-decides 0. Note that the transformation by itself does not require any additional communication, and hence can be performed in an asynchronous model.

Appendix A.3 gives an interactive consistency algorithm which matches the lower bounds for local and global decision given in Table 1 (which can easily modified to a matching algorithm for non-blocking atomic commit as well). The algorithm is modular: it composes instances of our optimal uniform consensus algorithm to build the optimal algorithm (instead of giving one ad-hoc algorithm for interactive consistency).

3 Local Decision in the Eventually Synchronous Model

Intuitively, the eventually synchronous model *ES* [8], is a model that is guaranteed to become synchronous, but only after an unbounded period of time. In *ES*, computation proceeds in rounds. We consider "communication-open" rounds: messages sent to correct processes are eventually received.[3]

In a round of *ES*, every process sends a message to all processes and waits for other messages sent in the round. The model notifies the processes when to stop waiting for the messages in each round. However, unlike the synchronous model, messages may be *delayed* (not received in the same round in which they were sent) by an arbitrary number of rounds, provided that the following conditions are met in every run: (1) messages sent to a correct process are eventually received, (2) in every round k, if some process p_i completes the round, then p_i has received at least $n-t$ messages of that round, and (3) there is an unknown round number GST [8], such that, in every round $k \geq GST$, for any process p_i, if some process completes round k without receiving round k message from p_i, then p_i has crashed before completing round k. We say that a run in *ES* is *synchronous* if in every round $k \geq 1$, for any process p_i, if some process completes round k without receiving round k message from p_i, then p_i has crashed before completing round k.

[3] *ES* can be emulated in an asynchronous system augmented with an *eventually perfect* failure detector $\diamond P$ [5]. In [7] we specify the eventually synchronous model based on round-by-round fault detector framework [12] and denote it by $RF_{\diamond P}$.

It is easy to see that atomic commit and interactive consistency are impossible to solve in *ES* when $t \geq 1$. Furthermore, every algorithm which solves consensus also solves uniform consensus [13]. Hence, we consider only the uniform consensus problem in this model. From [11] we know that for every uniform consensus algorithm, and for every $0 \leq f \leq t$ ($t \geq 1$), there is a run of the algorithm with at most f failures which takes arbitrary number of rounds for local decision. Hence, we define l_f and g_f in this model as bounds on the synchronous runs of the algorithm with at most f failures.

Since we consider only synchronous runs, the lower bounds for local and global decision for the synchronous model, immediately extends to *ES*. Furthermore, we know from [7] that, when $t-1 \leq f \leq t$, the global decision lower bound in *ES* is strictly higher than for the synchronous model. We show that, for most values of f, the local decision lower bound is the same as the global decision lower bound (i.e., $f+2$ rounds).

Proposition 5 *Let* $1 \leq t \leq n-1$. *For every uniform consensus algorithm in* ES *and every* $0 \leq f \leq t-3$, *there is a synchronous run with at most* f *failures where no correct process decides before round* $f+2$.

In fact, (in Appendix B) we prove Proposition 5 for the synchronous runs of an asynchronous round based model enriched with the perfect failure detector [5, 12, 4]. The result immediately extends to synchronous runs of *ES* because every run of the round based model with perfect failure detector is a run of *ES*. The result also extends to the synchronous send-omission model [16] as well, with a slight modification in the proof.

We also give (in Appendix B.2) an algorithm in *ES* that globally decides (and hence, locally decides) within $f+2$ rounds in every synchronous run with at most f failures. Thus, the $f+2$ round global decision bound is tight for $0 \leq f \leq t$ and the $f+2$ round local decision bound is tight for $0 \leq f \leq t-3$.

4 Concluding Remarks

Beliefs. In [21], Lamport pointed out *"two conflicting beliefs about the cost of making a decision"* in agreement algorithms.

- *"One is that the leader can simply decide and then inform the other processors of its decision with a single message delay [2]."*
- *"The other belief is that a three-phase commit a la Skeen [27] is needed."*

Lamport furthermore concluded that *"If making a decision is interpreted as solving the consensus problem, then neither belief is correct"*.

Interpretations. By studying the local decision lower bound in various models and for various agreement problems, we give in this paper precise interpretations of those beliefs that make them complementary rather than conflicting.

- We interpret the first belief as: the tight local decision lower bound for failure-free runs of (non-uniform) consensus in a synchronous model is 0, i.e., $l_0 = 0$ for consensus in the synchronous model.[4]

[4]Recall that, the consensus algorithm that we sketch in the introduction matches this lower bound.

- We interpret the second belief as: the tight local decision lower bound for failure-free runs of non-blocking atomic commit is 2 (even in the synchronous model), i.e., $l_0 = 2$ for non-blocking atomic commit in the synchronous model.[5]

Through our interpretation of the second belief, we show here that, in a rigorous sense, non-blocking atomic commit (and hence interactive consistency) is inherently slower than uniform consensus. To our knowledge, this is the first time a sharp line is drawn between non-blocking atomic commit and uniform consensus in terms of time complexity, and this was possible precisely because we considered local decisions. In terms of global decisions, there is no difference between these problems in the synchronous model.

Initial Configurations. It is also important to notice that when determining local decision lower bounds, we actually seek to determine the minimum number of rounds needed for some correct process to decide, considering the worst case possible initial configuration. Devising algorithms that are optimal for some specific initial configurations might however lead to interesting observations. For instance, considering runs of non-blocking atomic commit in the synchronous model, the case (a) where all correct processes decide 0 by round 1, in *every* run where some process proposed 0, is incompatible with the case (b) where all correct processes decide 1 by round 2, in the failure-free run in which every process propose 1. In particular, a closer look at the centralized three-phase commit algorithm [26] reveals that the algorithm is indeed optimal for case (b) but not for case (a). (We prove in the Appendix A.3 that, although each of the two bounds can be individually achieved, no algorithm can be optimal in both cases.) In other words, if the goal is indeed to optimize the *commit* case where all processes propose 1, then one needs to give up on the efficiency of the *abort* case where all correct processes decide at round 1 whenever some process proposes 0.

References

[1] Aguilera M. K. and Toueg S., A Simple Bivalency Proof that t-Resilient Consensus Requires $t + 1$ Rounds. *Information Processing Letters (IPL)*, 71(3-4):155-158, 1999.

[2] Birman K., Schiper A. and Stephenson P., Lightweight Causal and Atomic Group Multicast. *ACM Transactions on Computer Systems (TOCS)*, 9(3):272-314, 1991.

[3] Charron-Bost B. and Schiper A., Uniform Consensus Harder than Consensus. Technical Report DSC/2000/028, Swiss Federal Institute of Technology in Lausanne, 2000.

[4] Charron-Bost B, Guerraoui R. and Schiper A., Synchronous System and Perfect Failure Detector: solvability and efficiency issues. *Proc. of the IEEE International Conference on Dependable Systems and Networks (DSN)*, pp. 523-532, New York, 2000.

[5] Chandra T. D. and Toueg S., Unreliable Failure Detectors for Reliable Distributed Systems. *Journal of the ACM (JACM)*, 43(2):225-267, 1996.

[6] Delporte-Gallet C., Fauconnier H., Helary J. M. and Raynal M., Early stopping in global data computation. *Proc. 21st ACM Symposium on Principles of Distributed Computing, (PODC'02)*, ACM Press, pp. 258, Monterey (CA), 2002.

[7] Dutta P. and Guerraoui R., The Inherent Price of Indulgence. *Proc. 21st ACM Symposium on Principles of Distributed Computing, (PODC'02)*, ACM Press, pp. 88-97, Monterey (CA), 2002.

[5] It is important to note, with respect to the second belief mentioned by Lamport, that the notion of phase in the parlance of Skeen [26] is different from the notion of round. The first actually captures the status of the processes (*undecided, pre-committed,* and *decided*). The decentralized three-phase commit algorithm of Skeen requires two rounds in failure-free runs [26].

[8] Dwork C., Lynch N. and Stockmeyer L., Consensus in the Presence of Partial Synchrony. *Journal of the ACM (JACM)*, 35(2):288-323, 1988.

[9] Dwork C. and Skeen D., The inherent cost of nonblocking commitment. *Proc. 2nd ACM Symposium on Principles of Distributed Computing, (PODC'83)*, ACM Press, pp. 1-11, 1983.

[10] Dolev D., Reischuk R and Strong R., Early stopping in byzantine agreement. *Journal of the ACM (JACM)*, 37(4):720-741, 1990.

[11] Fischer M.J., Lynch N. and Paterson M.S., Impossibility of Distributed Consensus with One Faulty Process. *Journal of the ACM (JACM)*, 32(2):374-382, 1985.

[12] Gafni E., A Round-by-Round Failure Detector: Unifying Synchrony and Asynchrony. *Proc. 17th ACM Symposium on Principles of Distributed Computing, (PODC'98)*, ACM Press, pp. 143-152, Puerto Vallarta (Mexico), 1998.

[13] Guerraoui R., Revisiting the relationship between Non-Blocking Atomic Commitment and Consensus problems. *Proc. 9th International Workshop on Distributed Algorithms, (WDAG'95)*, Springer Verlag (LNCS 972), pp. 87-100, 1995.

[14] Guerraoui R., Indulgent Algorithms. *Proc. 19th ACM Symposium on Principles of Distributed Computing, (PODC'00)*, ACM Press, pp. 289-298, Portland (OR), 2000.

[15] Gray J., Notes on database operating systems. *Operating systems; an advanced course*, Springer Verlag (LNCS 60), 1978.

[16] Hadzilacos V., Byzantine agreement under restricted types of failures (not telling the truth is different from telling lies). Tech. Rep. 19-83, Aiken Computation Laboratory, Harvard University, 1983.

[17] Hadzilacos V., On the relationship between the Atomic Commitment and Consensus problems. *Proc. 9th International Workshop on Fault-Tolerant Computing*, Springer Verlag (LNCS 448), pp. 201-208, 1987.

[18] Keidar I. and Rajsbaum S., A Simple Proof of the Uniform Consensus Synchronous Lower Bound. *Information Processing Letters (IPL)*, 85(1):47-52, 2002.

[19] Lynch N., Distributed Algorithms. *Morgan Kaufmann Pub.*, San Francisco (CA), 872 pages, 1996.

[20] Lamport L., The Part-Time Parliament. *ACM Transactions on Computer Systems (TOCS)*, 16(2):133-169, 1998.

[21] Lamport L., Lower bounds on consensus. *Unpublished Note*, 2000.

[22] Lamport L. and Fischer M., Byzantine generals and transaction commit protocols. Technical Report 62, SRI International, 1982.

[23] Lamport L., Shostak R. and Pease M., The Byzantine Generals Problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 4(3):382-401, 1982.

[24] Moses Y. and Rajsbaum S., A Layered Analysis of Consensus. *SIAM Journal on Computing*, 31(4):989-1021, 2002.

[25] Pease M., Shostak R. and Lamport L., Reaching Agreement in the Presence of Faults. *Journal of the ACM (JACM)*, 27(2):228-234, 1980.

[26] Skeen D., Nonblocking commit protocols. *ACM SIGMOD International Symposium on Management of Data*, 133-142, 1981.

[27] Skeen D., Crash Recovery in a Distributed Database System. PhD thesis, University of California, Berkeley, May 1982.

A Synchronous Model

Proof Technique. We follow the layering scheme of [24], also used in [18]. Similar to [18, 24], we consider only a subset of runs in the model for showing lower bounds. A *subsystem* is a subset of the set of all possible runs in the model. We denote the synchronous crash-stop system by SCS. We consider the subsystem sub_{scs} (of SCS) where at most one process may crash in every round. If p_i crashes at round k, any subset of the messages p_i is supposed to send in that round may not be received.[6] Unless otherwise mentioned, the remaining discussion in Appendix A is in the context of sub_{scs}.

A configuration at (the end of) round $k \geq 1$ in a run is the collection of the state of all processes at the end of round k. The state of a process which has crashed in a configuration is a special symbol denoting that the process has crashed. We say that a process p_i is *alive* in a given configuration if p_i has not crashed in that configuration. A initial configuration (or round 0 configuration) in a run is the collection of initial state of all processes in that run. We denote the set of all initial configurations as $Init$. A run of an algorithm is completely defined by its initial configuration, its initial state and its failure pattern. (The failure pattern for a run, states for each round k, the process which crashes in round k, and the set of processes which did not receive round k message from the crashed process.) Therefore, for any configuration C at round k (of an agreement algorithm), we can define $r(C)$ as the run in which (1) round k configuration is C, and (2) no processes crashes after round k. We denote by $val(C)$ the decision value of the correct processes in $r(C)$. Note that a process p_i is alive in C iff p_i is correct in $r(C)$.

We denote a one round extension of a round k configuration C as follows: for $1 \leq i \leq n$ and $S \subseteq \Pi$, $C.(i, S)$ denotes the configuration reached by crashing p_i in round $k+1$ such that any process p_j does not receive a round $k+1$ message from p_i if any of the following holds: (1) $p_j = p_i$, (2) p_j is crashed in C, or (3) $p_j \in S$. $C.(0, \emptyset)$ denotes the one round extension of C in which no process crashes. Obviously, (i, S) for $i > 0$ and $S \subseteq \Pi$ is an *applicable* extension to C if at most $t-1$ processes have crashed in C and p_i is alive in C.

A layer $L(C)$ is defined as $\{C.(i, S) | i \in \Pi, S \subseteq \Pi, (i, S) \text{ is applicable to } C\}$. For a set of configurations SC at the same round, $L(SC)$ is another set of configurations defined as $\cup_{C \in SC} L(C)$. $L^k(SC)$ is recursively defined as follows: $L^0(SC) = SC$ and for $k > 0$, $L^k(SC) = L(L^{k-1}(SC))$.

Two configurations C and D at the same round are similar, denoted $C \sim D$, if they are identical or there exists a process p_j such that (1) C and D are identical except at p_j, and (2) there exists a process $p_i \neq p_j$ that is alive in both C and D.

A set of configurations SC is similarly connected if, for every $C, D \in SC$ there are states $C = C_0, \ldots, C_m = D$ such that $C_i \sim C_{i+1}$ for all $0 \leq i < m$. We now revisit Lemma 2.3 of [18] in sub_{scs}.

Lemma 6 *Let $SC = L^0(SC)$ be a similarly connected set of configurations in which no process has crashed, then for all $k \leq t$, $L^k(SC)$ is a similarly connected set of states in which no more than k processes are crashed in any configuration.*

Proof: (A simple modification of the proof of [18].[7]) The proof is by induction on round number k. The base case $k = 0$ is immediate. For the inductive step, assume that $L^{k-1}(SC)$ is similarly connected and in every configuration at most $k-1$ processes have crashed. Notice that in every extension which is applicable to any configuration in $L^{k-1}(SC)$, at most one new process can crash. Therefore, in every configuration in $L^k(SC)$ at most k processes have crashed.

[6] Notice that the subsystem in [18] contains runs with the additional restriction that, if a process p_i crashes in the send phase of round k, the round k messages may not be received by a *prefix* of Π. Thus, the subsystem in [18] is a subset of sub_{scs}.

[7] The statement of the Lemma is the same, however, the proof is different because we consider a subsystem different from that of [18].

We now show that for any configuration $C \in L^{k-1}(SC)$, $L(C)$ is similarly connected. Consider any two configurations in $L(C)$, $C1 = C.(i, S1)$ and $C2 = C.(j, S2)$, where $S1, S2 \subseteq \Pi$, and p_i and p_j are alive in C. We will show that $C1$ and $C.(0, \emptyset)$ are similarly connected. Using the same procedure, we can show that $C2$ and $C.(0, \emptyset)$ are similarly connected, thus showing that $C1$ and $C2$ are similarly connected.

$C.(i, \emptyset) \sim C(0, \emptyset)$ since the configurations only differ at p_i. If $S1 = \emptyset$ then we are done. Hence, let $S1 = \{q_1, q_2, \ldots, q_m\}$. For $1 \leq l \leq m$, let $S1_l = \{q_1, \ldots, q_l\}$, and $S1_0 = \emptyset$. For $0 \leq l < m$, $C.(i, S1_l) \sim C.(i, S1_{l+1})$ because the two configurations differ only at q_{l+1}. Thus, $C(i, \emptyset) = C.(i, S1_0)$ and $C1 = C.(i, S1_m)$ are similarly connected.

It remains to be shown that if $C \sim D$ and $C, D \in L^{k-1}(SC)$ then there are configurations $C' \in L(C)$ and $D' \in L(D)$ which are similar. Let p_i be the process such that C and D are different only at p_i. Then, configurations $C.(i, \Pi)$ and $D.(i, \Pi)$ are identical because no process receives message from p_i in round $k + 1$. □

Lemma 7 *(a) In a similarly connected set SC of states, if there are states C and D such that $val(C) \neq val(D)$, then there are two states $C1, D1 \in SC$ such that (1) $C1 \sim D1$ and (2) $val(C1) \neq val(D1)$.*
(b) For every binary consensus algorithm and $0 \leq k \leq t$, there are two configurations $y, y' \in L^k(Init)$ such that (1) at most k processes have crashed in each configuration, (2) the configurations differ at exactly one process, and (3) $val(y) = 0$ and $val(y') = 1$.

Proof: (a) Suppose, by contradiction, in a similarly connected set SC of states there are two states C and D such that $val(C) \neq val(D)$ and for every pair of similar states $C1, D1 \in SC$, $val(C1) = val(D1)$. Since C and D are similarly connected, there exist a set of sets, $C = C_0, C_1, \ldots, C_m = D$, such that, for $0 \leq l < m$, $C_l \sim C_{l+1}$. From our initial assumption, and a simple induction, it follows that $val(C_0) = val(C_1) = \ldots = val(C_m)$; a contradiction.
(b) We use the well-known lemma that $Init$ is similarly connected [11, 18]. Thus, from Lemma 1, $L^k(Init)$ is similarly connected. Consider the configuration C at round k of the failure-free run in which all processes propose 1. Obviously, $C \in L^k(Init)$, and from consensus validity, $val(C) = 1$. Similarly, consider the configuration D at round k in the failure-free run in which all processes propose 0. We have, $D \in L^k(Init)$ and $val(D) = 0$. Thus from Lemma 1 and Lemma 2(a), at the end of round k there exists two configurations y and y' such that (1) at most k processes have crashed in each configuration, (2) the configurations are similar, and (c) $val(y) = 0$ and $val(y') = 1$. Since, $val(y) \neq val(y')$, the configurations cannot be identical. Thus, they differ at exactly one process. □

A.1 Consensus

Proposition 1 (a) Let $1 \leq t \leq n - 1$. For every consensus algorithm and for every $0 \leq f \leq t$, there is a run with at most f failures in which no correct process decides before round f.
(b) Let $1 \leq t \leq n - 2$. For every consensus algorithm and for every $0 \leq f \leq t$, there is a run with at most f failures in which at most one correct process decides before round $f + 1$.

Proof. (a) Suppose by contradiction that there exists a binary consensus algorithm A and a round number f such that $0 \leq f \leq t$ and in every run with f failures, some correct process decides in round $f - 1$ or in a lower round. Notice that, the contradiction is immediate for the case $f = 0$: no process can decide in round -1 or in a lower round. So we consider the case $1 \leq f \leq t$.

Consider the set of configurations at round $f - 1$: $L^{f-1}(Init)$. From our assumption it follows that in every configuration $x \in L^{f-1}(Init)$, there is an alive process p_j which has decided. (Otherwise, since every correct process in $r(x)$ is an alive process in x, $r(x)$ is a run with less than f crashes in which no

correct process decides in round $f-1$ or in a lower round.) Furthermore, p_j decides $val(x)$ in x because p_j is a correct process in $r(x)$.

From Lemma 7(b) we know that there are two configurations $y, y' \in L^{f-1}(Init)$ such that (1) at most $f-1$ processes have crashed in each configuration, (2) the configurations differ at exactly one process (say p_i), and (3) $val(y) = 0$ and $val(y') = 1$. From our earlier discussion it follows that, in y, there is an alive process q_1 which has decided $val(y) = 0$.

We now show that, in y, no alive process distinct from p_i can decide. Suppose there is an alive process $p_j (\neq p_i)$ which has decided in y. As p_j is alive in y, so p_j is a correct process in $r(y)$, and hence, p_j decides $val(y) = 0$ in y. Furthermore, as $p_j \neq p_i$, p_j has identical states in y and y', so p_j is alive and has decided 0 in y'. Thus, in $r(y')$, p_j is a correct process and decides 0. However, every correct process in $r(y')$ decides $val(y') = 1$; a contradiction.

It immediately follows that $p_i = q_1$; i.e., p_i is alive and has decided 0 in y. Consider any run r' such that, r' is an extension of y and only process p_i crashes after round $f-1$, e.g., $r' = r(y.(i, \Pi))$. At most f processes crash in r'. At the end of round $f-1$ in r', the only alive process which has decided is p_i, and p_i is a faulty process in r'. Thus, r' is a run with f failures in which no correct process decides in round $f-1$ or in a lower round, a contradiction.

(b) Suppose by contradiction that there is a binary consensus algorithm A and a round number f such that, $0 \leq f \leq t$ and in every run with at most f failures, two correct processes decide in round f or in a lower round. Consider the configurations of A, at the end of round f: $L^f(Init)$. From our assumption, in every configuration $x \in L^f(Init)$, there are two alive processes which have decided. (Otherwise, since every correct process in $r(x)$ is an alive process in x, $r(x)$ is a run with f failures in which less than two correct processes have decided in round f or in a lower round.) Furthermore, these two processes have decided $val(x)$ because they are correct in $r(x)$.

From Lemma 7(b) we know that there are two configurations $y, y' \in L^f(Init)$ such that (1) y and y' differ at exactly one process, and (2) $val(y) = 0$ and $val(y') = 1$. However, from our earlier discussion it follows that (1) in y, there are two alive processes which have decided 0, and (2) in y', there are two alive processes which have decided 1. Since y and y' differ at exactly one process, in y, there is an alive process, say p_j, which has decided 1. Thus, p_j is correct and decides 1 in $r(y)$. However, every correct process in $r(y)$ decides $val(y) = 0$; a contradiction. □

A.2 Uniform Consensus

A.2.1 Lower Bound

Proposition 3 (a) Let $1 \leq t \leq n-1$. For every uniform consensus algorithm and every $0 \leq f \leq t-1$, there is a run with f failures in which no correct process decides before round $f+1$.
(b) Let $3 \leq t \leq n-1$. For every uniform consensus algorithm and every $0 \leq f \leq t-3$ there is a run with f failures in which at most one correct process decides before round $f+2$.

Proof: (a) Suppose by contradiction that there is a binary uniform consensus algorithm A and a round number f such that $0 \leq f \leq t-1$, and in every run with at most f failures, some correct process decides before round $f+1$. Consider the set of configurations of A at the end of round f: $L^f(Init)$. From our assumption it follows that in every configuration $x \in L^f(Init)$, there is an alive process p_j which has already decided. (Otherwise, since every correct process in $r(x)$ is an alive process in x, $r(x)$ is a run with f crashes in which no correct process decides before round $f+1$.) Furthermore, p_j decides $val(x)$ in x because p_j is a correct process in $r(x)$.

From Lemma 7(b) we know that there are two configurations $y, y' \in L^f(Init)$ such that (1) at most f processes have crashed in each configuration, (2) the configurations differ at exactly one process, say p_i, and (3) $val(y) = 0$ and $val(y') = 1$. From our assumption it follows that, in y, there is an alive

process q_1 which has decided 0, and, in y', there is an alive process q_2 which has decided 1. There are following two cases.

(1) $q_1 \neq p_i$: As y and y' are identical at all processes different from p_i, in y', q_1 is alive and has decided 0. Thus in $r(y')$, q_1 is a correct process and decides 0. However, in $r(y')$ every correct process decides $val(y') = 1$; a contradiction.

(2) $q_1 = p_i$: We distinguish two subcases:

- $q_2 = p_i$: Thus, $p_i = q_1 = q_2$, and hence, p_i is alive in y and y'. Consider a run $r1$ which extends y and in which p_i crashes before sending any message in round $f + 1$; i.e., $r1 = r(y.(i, \Pi))$. (Recall that $f \leq t - 1$). As p_i has decided 0 in y, from uniform agreement, it follows that every correct process decides 0 in $r1$. Since $t < n$, there is at least one correct process, say p_l in $r1$. Now consider a run $r2$ which extends y' and in which p_i crashes before sending any message in round $f + 1$; i.e., $r2 = r(y'.(i, \Pi))$. Notice that no correct process can distinguish between $r1$ and $r2$: no alive process which is distinct from p_i can distinguish y from y', and p_i crashes before sending any message in round $f + 1$. Thus, every correct process decides the same value in $r1$ and $r2$, in particular p_l decides 0 in $r2$. However, $p_i = q_2$ decides 1 in $r2$; a contradiction of uniform agreement.

- $q_2 \neq p_i$: Then, q_2 has the same state in y and y'. Thus, in y, q_2 is alive and has decided 1. In any extension of y, $p_i = q_1$ has decided 0 and q_2 has decided 1; a contradiction of uniform agreement.

(b) For this part we consider a subsystem sub_{scs1} which is the set of all runs in the synchronous crash-stop model, i.e., any number of processes can crash in a round. (We revisit few definitions which were presented in the context of the subsystem sub_{scs} in which at most one process can crash in a round.) We define the following in sub_{scs1}. For any configuration C at round k of a uniform consensus algorithm A, we define $R(C)$ as the run in which (1) the configuration at round k is C and (2) no process crashes after round k. We denote by $Val(C)$ the decision value of correct processes in $R(C)$. *Serial runs* are those runs of A which are in sub_{scs} (i.e., runs in which at most one process crashes in every round). Similarly, *Serial configurations* are the configurations of serial runs of A. As every run in sub_{scs} is a serial run sub_{scs1}, from Lemma 7(b) we immediately have

Claim 3(c) *For every binary consensus algorithm and $0 \leq k \leq t$, there are two (serial) configurations $y, y' \in L^k(Init)$ such that (1) at most k processes have crashed in each configuration, (2) the configurations differ at exactly one process, and (3) $Val(y) = 0$ and $Val(y') = 1$.*

Proof of Proposition 3(b) continued. Suppose by contradiction that there is a binary uniform consensus algorithm A and a round number $f + 1$ such that $0 \leq f \leq t - 3$, and in every run of A with at most f failures, there are two correct processes which decide before round $f + 2$.

We know from Claim 3(c) that at the end of round f there are two serial configurations of A, y and y', such that, (1) at most f processes have crashed in each configuration, (2) the configuration differ at exactly one process, say p_i, and (3) $Val(y) = 0$ and $Val(y') = 1$. Let z and z' denote the configuration at the end of round $f + 1$ of $R(y)$ and $R(y')$, respectively. From our initial assumption about A, in z, there are two alive processes q_1 and q_2 which have decided 0. Similarly, in z', there are two alive processes q_3 and q_4 which have decided 1. Since q_1 and q_2 are distinct, at least one of them is distinct from p_i, say q_1. Similarly, without loss of generality we can assume that q_3 is distinct from p_i. (Processes q_1 and q_3 may or may not be distinct.) There are following two cases.

Case 1. Process p_i is alive in y and y'. Consider the following two non-serial runs:[8]
R1 is a run such that (1) the configuration at the end of round f is y, (2) p_i crashes in the send phase of round $f+1$ such that only q_1 receives the message from p_i, (3) q_1 and q_3 crash before sending any message in round $f+2$, and (3) no process distinct from p_i, q_1, and q_3 crashes after round f. Notice that, q_1 cannot distinguish the configuration at end of round $f+1$ in $R1$ from z, and therefore, decides 0 at the end of round $f+1$ in $R1$. By uniform agreement, every correct process decides 0. Since $t \leq n-1$, there is at least one correct process in $R1$, say p_l.
R2 is a run such that (1) configuration at the end of round f is y', (2) p_i crashes in the send phase of round $f+1$ such that only q_3 receives the message from p_i, (3) q_1 and q_3 crash before sending any message in round $f+2$, and (3) no process distinct from p_i, q_1, and q_3 crashes after round f. Notice that, q_3 cannot distinguish the configuration at end of round $f+1$ in $R2$ from z', and therefore, decides 1 at the end of round $f+1$ in $R2$. However, p_l can never distinguish $R1$ from $R2$: at the end of round $f+1$, the two runs are different only at p_i, q_1, and q_3, and none of the three processes send messages after round $f+1$ in both runs. Thus, (as in $R1$) p_l decides 0 in $R2$; a contradiction of uniform agreement.

Case 2. Process p_i has crashed in either y or y'. Without loss of generality, we can assume that p_i has crashed in y, and hence, p_i is alive in y'. (Recall that p_i has different states in two configurations.) Consider the following two non-serial run:
R12 is a run such that (1) configuration at the end of round f is y (and hence, p_i has crashed before round $f+1$), (2) no process crashes in round $f+1$, and (3) q_1 and q_3 crash before sending any message in round $f+2$. No process distinct from p_i, q_1 and q_3 crashes after round f. Notice that, q_1 cannot distinguish the configuration at the end of round $f+1$ in R12 from z because q_1 does not receive the round $f+1$ message from p_i in both runs. Thus, (as in z) q_1 decides 0 at the end of round $f+1$ in R12. Due to uniform agreement, every correct process decides 0 in R12. Since $f \leq t-3 \leq n-4$, there is at least one correct process in R12, say p_l.
R21 is a run such that (1) configuration at the end of round f is y', (2) p_i crashes in the send phase of round $f+1$ such that only q_3 receives the message from p_i, and (3) q_1 and q_3 crash before sending any message in round $f+2$. No process distinct from p_i, q_1 and q_3 crashes after round f. Notice that, q_3 cannot distinguish the configuration at the end of round $f+1$ in R21 from z' because it receives the message from p_i in both runs. Thus, (as in z') q_3 decides 1 at the end of round $f+1$ in R21. However, p_l can never distinguish R12 from R21: at the end of round $f+1$, the two configurations are different only at p_i, q_1 and q_3, and none of them send messages after round $f+1$ in both runs. Thus, (as in R12), p_l decides 0 in R21; a contradiction of uniform agreement. □

Notice that in the proof of Proposition 3, runs $R1$ and $R2$ are not in sub_{scs}: two processes crash at round $f+2$. In fact, Lemma 5(b) does not hold in sub_{scs}. For example, there is a binary consensus algorithm in sub_{scs}, in every failure-free run of which two processes decide at the end of round 1.

A.2.2 Matching Algorithms

Proposition 8 *The algorithm in Fig. 1 solves Uniform Consensus.*

Proof: Termination is ensured as the algorithm executes for at most $t+1$ rounds. Validity follows from the fact that processes initially start with their own initial value as estimate, exchange their estimate thereafter, and eventually decide on their estimate value. Thus the estimate value of any process always contains some initial value. To prove agreement, we proceed through a series of lemmas.

Lemma 9 *If any process p_i does not crash before the end of round i ($i \leq t-1$), all processes adopt the same estimate by round i.*

[8]The runs are not serial because in round $f+2$, two processes crash in each run.

```
1:  At process p_i:
2:  procedure propose(v_i)
3:     est_i := v_i
4:     decision_i := ⊥

5:     for round r = 1, ..., t − 1 do                                  {Rounds 1 to (t − 1)}
6:        if r = i then                                                {Send Phase}
7:           send ESTIMATE(est_i,i) to all
8:        if r = i + 1 then
9:           send ALIVE(i) to all

10:       if receive ESTIMATE(est_r,r) from p_r then                   {Receive Phase}
11:          est_i := est_r
12:       if r = i then
13:          decision_i := est_i
14:       if receive ALIVE(r − 1) then
15:          decision_i := est_i

16:    if i = t − 1 then                                               {Round t, Send Phase}
17:       send ALIVE(t − 1) to all
18:    send ESTIMATE(est_i,t) to all

19:    if receive ALIVE(t − 1) then                                    {Receive Phase}
20:       decision_i := est_i
21:    receive ESTIMATE(est_*,t) from processes in S_t
22:    if |S_t| ≥ n − t + 1 then
23:       decision_i := est_{min{j | p_j ∈ S_t}}

24:    if decision_i ≠ ⊥ then          {Round (t + 1), Send Phase. Not needed if t = n − 1}
25:       send DECISION(decision_i,t + 1) to all                       {Not needed if t = n − 1}
26:    else
27:       send ESTIMATE(est_i,t + 1) to all                            {Not needed if t = n − 1}

28:    if received DECISION(decision_*,t + 1) from any process then    {Receive Phase. Not needed if t = n − 1}
29:       decision_i := decision_*                                     {Not needed if t = n − 1}
30:    else
31:       receive ESTIMATE(est_*,t + 1) from processes in S_{t+1}      {Not needed if t = n − 1}
32:       est_i := est_{min{j | p_j ∈ S_{t+1}}}                        {Not needed if t = n − 1}

33:    decision_i := est_i
```

Figure 1: A synchronous uniform consensus algorithm

Proof: By the algorithm, process p_i ($i \leq t-1$) sends its estimate to all processes in round i. As p_i does not crash before the end of the round, its message reaches all processes. By the algorithm, all processes adopt the received value as their estimate. □

Lemma 10 *At the end of round* $(t-1)$*, either all alive processes adopt the same estimate or* $(t-1)$ *processes have crashed.*

Proof: By Lemma 9, if any process p_i ($i \leq t-1$) does not crash before the end of round i, then all processes adopt the same estimate by round i. If two or more processes have distinct estimates by round $(t-1)$, processes $p_1, \ldots, p_{t-1}$ crashed. □

We derive an easy corollary of Lemma 10, as at most t processes can crash.

Corollary 11 *At the end of round $(t-1)$, if two or more processes have distinct estimates, at most one process can still crash.*

Lemma 12 *If any process receives an* alive *message in round $(i+1)$ from any process p_i $(i \leq t-1)$, p_i has decided on v and all processes have adopted v as estimate.*

Proof: In the algorithm, process p_i $(i \leq t-1)$ sends its estimate to all in round i, and sends an *alive* message in round $(i+1)$. If any process receives an *alive* message from process p_i, p_i necessarily reaches the end of round i. By Lemma 9, all processes adopt the same estimate at the end of round i. In the algorithm, p_i decides on its estimate at the end of round i. □

Lemma 13 *If any process decides on value v by round $(t-1)$, each alive process either adopts or decides on v.*

Proof: In the algorithm, a process can decide at two different locations by round $(t-1)$, either (1) at the end of the round in which it sends its estimate (round i, $i \leq t-1$), or (2) after receiving an *alive message*. In case (1), by Lemma 9, all processes adopt v as estimate. In case (2), by Lemma 12, each process has either decided or adopted v as estimate. □

Lemma 14 *If two or more processes decide in round t, they all decide on the same value.*

Proof: In round t, all alive processes send their estimate to each other. A process decides on the minimum of theses estimates whenever it receives at least $(n-t+1)$ estimates.

According to Lemma 10, processes may or may not have the same estimate at the end of round $(t-1)$. If they do, all processes that decide in round t, decide on the same value, as they compute the minimum function on the same set of identical values.

Consider now the case where two or more processes have distinct estimates at the end of round $(t-1)$. Assume by contradiction that two distinct processes p_i and p_j respectively decide on two different values v and v' in round t. Assume without loss of generality that $v < v'$. We denote by S_i (resp. S_j) the set of estimate values received by p_i (resp. p_j) in round t. We have $v \notin S_j$ otherwise p_j decides on v. Because p_i and p_j reach the end of round t, and p_j does not receive v in round t, p_i and p_j do not have v as estimate at the beginning of round t. Denote by p_v the process that has v as its estimate at the beginning of round t. Process p_v necessarily crashes in round t, as p_j does not receive v. Nevertheless, p_j receives $(n-t+1)$ estimates in round t. By Corollary 11, there are $(t-1)$ crashed processes at the beginning of round $(t-1)$. Moreover, p_v is alive at the beginning of round t but crashes in round t, without reaching p_j; and there are $(n-t+1)$ other processes whose messages are received by p_j. This contradicts the fact that we have n processes in the system. □

Lemma 15 *If two or more processes have distinct estimates at the beginning of round t, and if any process p_i decides on value v in round t, either all processes decide on v in round t, or p_i is correct.*

Proof: If two or more processes have distinct estimates at the beginning of round t, then by Corollary 11, at most one process can still crash at the beginning of round t. By Lemma 14, all processes that decide in round t decide on the same value. Consider now that process p_i decides on v in round t, and any process p_j distinct from p_i does not decide in round t. To not decide, p_j receives less than $(n-t+1)$ estimates in round t. Thus some process distinct from p_i and p_j crashes in round t. By Lemma 10, there now are t faulty process. Hence p_i is necessarily correct. □

Lemma 16 *If no process has decided by round t, no process can crash any more.*

Proof: Assume that no process has decided by round t, in particular distinct processes p_i and p_j. At the end of round t, p_i and p_j receive less than $(n-t+1)$ estimates, otherwise they decide. This means there are at most $(n-t)$ alive processes, or equivalently, at least t crashed processes. □

(End of proof of Proposition 8) Consider the first process to decide in the algorithm. By Lemma 13, if it decides before or at round $(t-1)$, all other processes adopt the same estimate, and agreement is ensured. In rounds t and $t+1$, agreement is trivially ensured if all processes have adopted the same estimate. Otherwise, by Lemma 14 all processes that decide at round t decide on the same estimate. If any process p_i decides in round t and some process does not, because p_i sends its decision in round $(t+1)$, and, by Lemma 15, p_i is correct, all processes that do not decide in round t decide in round $(t+1)$ on the same value as processes in round t. If no process decides in round t, then all processes that decide decide in round $(t+1)$, and, by Lemma 16, decide on the same estimate. □

Proposition 17 *In the algorithm in Fig. 1, for $2 \leq t \leq n-1$, at least one process decides by round $f+1$ $(0 \leq f \leq t)$, in all runs with at most f failures, and all processes decide by round $g_f = f+2$ $(0 \leq f \leq t-2)$, $g_f = f+1$ $(t-1 \leq f \leq t,\ t \leq n-2)$ or $g_f = f$ $(f = t,\ t = n-1)$, in all runs with at most f failures.*[9]

Proof: We first consider the case $0 \leq f \leq t-2$. For l_f, consider any run r, such that there are at most f faulty processes in r. Assume by contradiction that no process decides by round $f+1$. In the algorithm, process p_i decides on its estimate after sending it to all other processes, at the end of round i. If no process decides after $(f+1)$ rounds, there are necessarily $(f+1)$ faulty processes; a contradiction. For g_f, assume by contradiction that there exists at least one process that does not decide by round $(f+2)$. In the algorithm, process p_i $(1 \leq i \leq t-1)$ sends its estimate in round i, and sends an *alive* message in round $(i+1)$. Any process that receives an *alive* message decides on its estimate. If some process does not decide by round $(f+2)$, there must be $f+1$ faulty processes; a contradiction.

For $f = t-1$, g_f implies l_f. Consider any run r, such that there are at most f failures in r. If any of the $(t-1)$ processes to send its estimate in the $(t-1)$ first rounds is correct, the algorithm reaches a global decision by round t. So consider that the first $(t-1)$ processes that send their estimate are faulty. As there are at most $(t-1)$ faulty processes, there are at least $(n-t+1)$ other correct processes. In round t, each alive process receives at least $(n-t+1)$ estimates, and decides.

For $f = t$, we show global decision (and hence, local decision) by round $t+1$. In the case $t \leq n-2$, the algorithm obviously reaches a global decision by round $(t+1)$. For $t = n-1$, consider any run r with at most t failures. If there are less that t failures, all processes decide by round t. Otherwise, if any process p_i does not decide and is still alive at the end of round t, p_i is necessarily the only alive process remaining in the system, and it can safely decides on its own estimate value. □

Proposition 18 *The algorithm in Fig. 2 solves Uniform Consensus.*

Proof: Termination and validity are similar to Algorithm 1. To prove uniform agreement, we first revisit Lemma 9 and extends it to rounds t and $t+1$.

Lemma 19 *If any process p_i does not crash before the end of round i $(i \leq t+1)$, all processes adopt the same estimate by round i.*

Proof: By the algorithm, process p_i $(i \leq t+1)$ sends its estimate to all processes in round i. As p_i does not crash before the end of the round, its message reaches all processes. By the algorithm, all processes adopt the received value as their estimate. □

[9] This matches all l_f and g_f except l_t.

```
1: At process p_i:
2: procedure propose(v_i)
3:    est_i := v_i
4:    decision_i := ⊥

5:    for round r = 1, ..., t do                                   {Rounds 1 to t}
6:       if r = i then                                               {Send Phase}
7:          send ESTIMATE(est_i,i) to all
8:       if r = i + 1 then
9:          send ALIVE(i) to all

10:      if receive ESTIMATE(est_r,r) from p_r then                  {Receive Phase}
11:         est_i := est_r
12:      if r = i then
13:         decision_i := est_i
14:      if receive ALIVE(r − 1) then
15:         decision_i := est_i

16:   if i = t + 1 then          {Round (t + 1), Send Phase. Not needed if t = n − 1}
17:      decision_i := est_i                          {Not needed if t = n − 1}
18:      send ESTIMATE(est_i,i) to all                {Not needed if t = n − 1}

19:   if receive ESTIMATE(est_{t+1},t + 1) from p_{t+1} then   {Receive Phase. Not needed if t = n − 1}
20:      est_i := est_{t+1}                           {Not needed if t = n − 1}

21:   decision_i := est_i
```

Figure 2: A second synchronous uniform consensus algorithm

Lemma 20 *At the end of round t, either all alive processes adopt the same estimate, or no process can crash any more.*

Proof: By Lemma 19, if any process p_i ($i \leq t$) does not crash before the end of round i, then all processes adopt the same estimate by round i. If two or more processes have distinct estimates at the end of round t, processes $p_1, \ldots, p_t$ crashed, and no process can crash any more. □

(End of proof of Proposition 18) Consider the first process to decide in the algorithm. We consider two cases. In case (1), process p_i ($i \leq t$) decides after sending its estimate to all. By Lemma 19, all other processes adopt the same estimate, ensuring agreement. In case (2), process p_{t+1} decides at the beginning of round $(t+1)$. By Lemma 20, either all processes (including p_{t+1}) already have the same estimate at the beginning of round $(t+1)$, and eventually decide on that estimate, or no process can crash. In the latter case, all processes receive the message sent by p_{t+1} in round $(t+1)$, and decide on that value, thus ensuring agreement. □

Proposition 21 *In the algorithm in Fig. 2, for $2 \leq t \leq n-1$, at least one process decides by round $l_f = f+1$ $(0 \leq f \leq t-1)$ or by $l_f = f$ $(f = t)$, in all runs with at most f failures, and all processes decide by round $f+2$ for $0 \leq f \leq t-1$, by $g_f = f+1$ $(f = t,\ t \leq n-2)$ or by $g_f = f$ $(f = t,\ t = n-1)$, in all runs with at most f failures.*[10]

Proof: For the case $0 \leq f \leq t-2$, the algorithm is actually similar as the algorithm in Fig. 1, and the proof is identical to Proposition 17, for both l_f and g_f.

[10]This matches all l_f and g_f except g_{t-1}.

Consider now the case $f = t - 1$. In any run r with at most f failures, by Lemma 19, some correct process decides by round $l_f = f + 1$. It then sends an *alive* message to all processes, which decide by round $g_f = f + 2$.

For the case $f = t$, consider any run r with at most f failures. By Lemma 19, if one of the processes to send its estimate value in the first t rounds is correct in r, then it decides by round t. If all the processes to send their estimate value in the first t rounds are faulty in r, then by Lemma 20, p_{t+1} is correct, and it decides on its estimate at the beginning of round $t+1$. Hence $l_f = f$. For g_f, we trivially have $g_f = f + 1$ in this case, as the algorithm executes for at most $(t+1)$ rounds. Note that in the case $t = n - 1$, if any process $p_i \neq p_{t+1}$ is correct, all processes that decide, decide before round f. If p_{t+1} is the only process in the system at the end of round t, it does not need to send its estimate value. Hence $g_f = f$. □

A.3 Non-blocking Atomic Commit (NBAC) and Interactive Consistency (IC)

Proposition 4 Let $2 \leq t < n$. For every NBAC algorithm, there is a failure-free run in which no process decides at round 1.

Proof: Suppose by contradiction that $2 \leq t < n$ and there is a NBAC algorithm A such that, in every failure-free run, there is a process which decides at round 1. Let C be the initial configuration in which all processes propose 1. Consider the failure-free run $R1$ starting from C: $R1 = r(C)$. Suppose some process p_i decides at the end of round 1. From AC-abort validity, we know that p_i decides 1 in $R1$.

Consider another run $R2$ such that every process proposes 1. Some process p_j ($\neq p_i$) crashes in round 1 and only p_i received round 1 message from p_j. Process p_i crashes in round 2, before sending round 2 message to any process, and no process crashes thereafter. That is, $R2 = r(C.(j, \Pi - \{p_i\}).(i, \Pi))$. At the end of round 1, p_i cannot distinguish $R1$ from $R2$. Thus, p_i decide 1 in $R2$. From AC-agreement we know that every process distinct from p_i and p_j decides 1. There exist at least one such process, say p_l, because $2 \leq t < n$.

Let $C0$ be the initial configuration in which p_j propose 0 and all other processes propose 1. Consider a run $R3$ starting from $C0$ with the same failure pattern as $R2$: $R3 = r_3(C0.(j, \Pi - \{p_i\}).(i, \Pi))$. No process distinct from p_i and p_j can distinguish $R2$ from $R3$: at the end of round 1, only p_i receives message from p_j, but p_i crashes before sending any message in round 2. Therefore, every process distinct from p_i and p_j, decides 1 (as in $R2$), in particular p_l; a contradiction of AC-commit validity. □

Proposition 22 Early-Abort NBAC: *There is an algorithm that globally decides at round* 1 *in every run starting from an initial configuration in which some process proposes 0.*

Proof: For presentation simplicity, we assume an underlying NBAC algorithm NBAC1. (We make no assumption on the time complexity of NBAC1.) The required NBAC algorithm, denoted by NBAC2, is as follows. In the first round, every process sends its proposal value to all. At the end of the first round, a process decides 0 if it receives at least one 0, or if it receives less than n messages. At the beginning of the second round, a process starts NBAC1 with proposal value 1 if it has not decided in round 1, otherwise it does not invoke NBAC1.

AC-termination of NBAC2 follows from AC-termination of NBAC1. For showing AC-agreement of NBAC2, we consider the following three exhaustive cases:

1. If no process invokes NBAC1, then any process which decides, does so at round 1, and therefore, decides 0.

2. Consider any run r of NBAC2 when some process p_i decides at round 1 (and hence does not invoke NBAC1) and there is another process which invokes NBAC1. Obviously, every process

which decides at round 1 (including p_i) decides 0. Let us denote by S, the set of processes which crash in round 1 or decide at round 1 of r. We have $p_i \in S$. From our algorithm, no process in S invokes NBAC1. Therefore, the run of NBAC1 within r is indistinguishable from an independent run of NBAC1 in which every process in S proposes 0 to NBAC1 and then crashes before sending any message. Therefore, from AC-commit validity of NBAC1, every process which decides in r after invoking NBAC1, decides 0.

3. If all processes invoke NBAC1, then no process has decided in round 1, and therefore, AC-agreement of NBAC2 follows from the same property of NBAC1.

To see the AC-commit validity of NBAC2, notice that a process cannot decide 1 at round 1, i.e., can decide 1 only after invoking NBAC1. From AC-commit validity of the NBAC1, the decision is 1 only if all processes propose 1 to NBAC1. A process proposes 1 to NBAC1 only if it receives 1 from all processes in round 1 of NBAC2. Therefore, if a process decides 1 in a run of NBAC2, then all processes have proposed 1 in that run.

If all processes propose 1 to NBAC2 and no process crashes, then every process receives 1 from all processes, and hence, propose 1 to NBAC1 and do not crash. By the AC-abort validity property of NBAC1, every processes decides 1. Thus, we have the AC-abort validity property of NBAC2.

It is easy to see that, for every run of NBAC2 starting from an initial configuration in which some process proposes 0, all processes that decide, decide 0 at the end of round 1. □

Now we show that no early-aborting NBAC algorithm can have an efficient global decision in failure-free run.

Proposition 23 *For $3 \leq t \leq n-1$, there does not exist a NBAC algorithm that achieves the following two bounds.*

1. *In every run starting from an initial configuration in which some process proposes 0, global decision is reached by round* 1.

2. *In the failure-free run starting from the initial configuration in which all processes propose 1, there is a global decision at round* 2.[11]

Proof: Suppose by contradiction that there is a NBAC algorithm $\mathcal{A}$ which achieves the two bounds mentioned above. We exploit indistinguishability between five different runs of $\mathcal{A}$, and derive a contradiction.

Consider run $R1$, in which process p_1 proposes 0, and all other processes propose 1. Process p_1 crashes before sending any message in round 1. By AC-commit validity, the only possible decision in this run is 0. From property 1, every process distinct from p_1 decides 0 at the end of round 1, in particular p_2.

Consider now run $R2$, starting from the initial configuration in which all processes propose 1 (including p_1). Process p_1 crashes in round 1 after sending a message to all processes but p_2. Clearly, p_2 cannot distinguish between $R1$ and $R2$. Thus it decides 0 at the end of round 1 in $R2$.

Consider now run $R3$, which is identical to $R2$, except that p_2 now crashes at the beginning of round 2, before sending any message in round 2, and p_3 crashes at the beginning of round 3. All remaining processes are correct. Clearly, at the end of round 1, $R2$ and $R3$ are indistinguishable for p_2, and hence, p_2 decides 0 at the end of round 1 in $R3$, and then crashes.

Now consider run $R4$, which is failure-free, starting from the initial configuration in which all processes propose 1. From property 2, and the AC-abort validity property of $\mathcal{A}$, all processes decide 1 at the end of round 2 in $R4$, in particular p_3.

[11] Notice that the second property is even weaker than "global decision in 2 rounds in every failure free run".

Finally consider run $R5$, similar to $R4$, but in which processes p_1 and p_2 crash at the beginning of round 2, each process sends a message to only p_3 in round 2, and process p_3 crashes at the beginning of round 3. Clearly, $R4$ and $R5$ are indistinguishable for p_3 at the end of round 2. Thus it decides 1 at the end of round 2, and then crashes.

In $R3$, process p_2 decides 0 and crashes. In $R5$, process p_3 decides 1 and crashes. Runs $R3$ and $R5$ are however indistinguishable for all remaining processes. This contradicts our initial assumption that algorithm $\mathcal{A}$ satisfies AC-agreement. □

The following Lemma generalizes Proposition 4 for IC.

Lemma 24 *For any IC algorithm and any run in which f processes crash, for any f and t such that $0 \leq f \leq t-2 \leq n-3$, no process decides at round 1.*[12]

Proof: Suppose by contradiction that there is an binary IC algorithm A (i.e., valid proposal values are 0 and 1) such that A has a run with f failures ($0 \leq f \leq t-2 \leq n-3$) such that some process decides at the end of round 1. Let r and p_i be the corresponding run and the deciding process, respectively. Let p_i (and all correct processes) decide on the decision vector V in r, where $V[j]$ denotes the j^{th} component.

Consider another process p_j distinct from p_i and correct in r. (Recall that $f \leq n-3$.) Let $x \in \{0,1\}$ be the proposal value of p_j in r. By the IC-validity property, $V[j] = x$.

Consider a one round configuration C which is generated by an execution identical to the first round of r, except that p_j crashes in round 1, and any message sent by p_j to processes distinct from p_i is lost. Obviously, p_i cannot distinguish the first round of r from the round which generated C. Therefore, p_i has decided V in C. We construct a run r' of A by extending C such that p_i crashes before sending any message in round 2, and no processes crashes thereafter. (Recall that $f \leq t-2$, and hence, two new processes can crash in r' as compared to r.) In r', p_i decides V at round 1, and from IC-agreement it follows that, every correct process decides V. In particular, there exists a correct process in r', say p_k, which is distinct from p_i and p_j, and which decides V. (Recall that, $t-2 \leq n-3$, i.e., there exists a correct process in every run.)

Consider another run r'' whose initial configuration and failure pattern are identical to r' except that p_j proposes $1-x$. Notice that, no process distinct from p_i and p_j can distinguish r' from r''. Therefore, p_k decides V in r''. Thus, p_j proposes $1-x$ in r'' but p_k decides V with $V[j] = x$; a contradiction of IC-validity. □

A.3.1 Matching Algorithm

We give now an optimal algorithm for Interactive Consistency that matches the lower bounds in Table 1 and Lemma 24.

The algorithm is given in Figure 3, and is constructed out of n instances of our uniform consensus algorithm (either that of Figure 1 or Figure 2) modified as follows. We denote by $unifCons_j$ the j^{th} instance of modified uniform consensus algorithm, and by v_i the value proposed by some process p_i to the interactive consistency algorithm. *In $unifCons_j$, p_j plays the part of p_1, p_{j+1} that of p_2, ..., p_n that of $p_{n-(j-1)}$, p_1 that of p_{n-j} and so on (a cyclic shift by $j-1$).*

Notice that in our original uniform consensus algorithm, if processes decide the proposal value of p_i then every p_y such that $y < i$ is faulty. Thus, in $unifCons_j$, if processes decide the proposal value of p_z then (1) if $j \leq z$ then all processes from $p_j, \ldots, p_z$ are faulty, and (2) if $j > z$ then all processes from $p_j, \ldots, p_n, p_1, \ldots, p_z$ are faulty (*Ordering property*).

In the interactive consistency algorithm, n instances of $unifCons$ are run concurrently. Process p_j proposes v_j to $unifCons_i$ if $i = j$, else p_j proposes $\perp$. A process IC-decides when it has decided in every $unifCons$.

[12]Note that, this result implies that, in *every* failure-free run, *no* process decides at round 1.

Proposition 25 *The algorithm in Figure 3 is correct.*

Proof: IC-agreement and IC-termination properties trivially follow from the uniform agreement and termination properties of the underlying $unifCons$ algorithm. We now prove that the algorithm satisfies IC-validity. Consider instance $unifCons_i$, whose purpose is to agree on the component corresponding to process p_i:

1. By the algorithm, only two values, p_i's input value and $\perp$, are ever proposed to this instance. By the validity property of $unifCons$, the decided value is one of them.

2. If p_i is correct, then by the ordering property of $unifCons_i$, processes that decide, decide on p_i's value in $unifCons_i$.

From (1) and (2), IC-validity is ensured. □

Proposition 26 *The algorithm in Fig. 3 is optimal: it achieves the bounds in Table 1 and of Lemma 24.*

Proof: *(Sketch)* In the following, we show that our interactive consistency algorithm achieves the g_f and l_f of the underlying uniform consensus algorithm, except for l_0. Thus showing the bounds to be tight. We also match $l_0 = 2$ for interactive consistency.

In the failure-free run, every $unifCons$ algorithm globally decides in round 2, and hence the interactive consistency algorithm globally (and locally) decides in round 2. Consider the case when $f \neq 0$. In the following l_f and g_f denote the tight lower bounds for uniform consensus.

For g_f, consider any run r with at most $f > 0$ failures. In r, every instance of algorithm $unifCons$ globally decides by round g_f. Thus, the algorithm in Fig. 3 reaches a IC global decision by round g_f.

For l_f, consider any run r with at most $f > 0$ failures. In each $unifCons$ instance in Fig. 3, there exists a process that locally decides by round l_f, and all processes that decide, decide by round g_f. To show that there exists a process that IC locally decides by round l_f, we proceed by contradiction.

Assume that no process IC locally decided by round l_f in r. We know however that each individual UC instance locally decides by round l_f. To prevent an IC local decision by l_f, there must exist at least two instances which globally decide at $g_f (\leq l_f + 1)$. (If only one $unifCons$ instance, say $unifCons_x$, globally decides after l_f, and all other $unifCons$ instances globally decide by l_f, then the correct process which has decided by l_f in $unifCons_x$ has also decided in all other $unifCons$ instances (and hence, has IC-decided).

For any instance of our $unifCons$ to globally decides after l_f, the f processes that send their estimate during the first f rounds must be faulty (otherwise, if at least one process is correct among them, a global decision is reached by round l_f). Since processes play different parts each of the n instances of $unifCons$, for any two distinct $unifCons$ instances, there exists at least one process which (1) is among the first f processes to send its estimate in one $unifCons$ instance, and (2) is not among the first f processes to send its estimate in the other one.

Thus, for two $unifCons$ instances to globally decide after round l_f, there must exist at least $f + 1$ faulty processes, which contradicts our initial assumption that r has f faulty processes. □

B Eventually Synchronous Model

B.1 The Uniform Consensus Lower Bound

Before giving our lower bound result, we briefly recall two other models which are weaker than synchronous model.

```
1: At process p_i:
2: procedure propose(v_i)
3:    V_i := [⊤, ..., ⊤]
4:    for j from 1 to n do
5:       if j = i then
6:          unifCons_i.propose(v_i)          {unifCons_i is such that v_i is decided if p_i is correct}
7:       else
8:          unifCons_j.propose(⊥)

9: upon unifCons_k.decide(v), for some 1 ≤ k ≤ n do
10:   V_i[k] := v
11:   if (∀j)(V_i[j] ≠ ⊤) then
12:      decide(V_i)
```

Figure 3: An interactive consistency algorithm based on uniform consensus instances

- *Synchronous Send-Omission:* In addition to failure by crash-stop, a process may fail by send-omission. If a process p_i fails by send omission at a round k, then any message it sends in round k or in a higher round, may be lost. However a process that fails by send omission does not stop and keeps on executing subsequent rounds, unless it fails by crash-stop. The model is synchronous in the following sense: if some process p_i completes the send phase of the round and p_i has not yet failed by send omission, every process which completes the receive phase of the round, receives the message sent by p_i in the send phase.

- *Round based model with Perfect Failure Detection (WS):* Computation proceeds in rounds based on a Perfect failure detector [5].[13] Each round consists of a send and a receive phase. Processes can crash in any phase. In send phase of any round k, each process p_i is supposed to send messages to all processes, and in the receive phase, for every process p_j, p_i waits until it receives round k messages from p_j, or the local failure detector module suspects p_j. Every message sent to a correct process is eventually received (maybe after a delay of arbitrary number of rounds). In [4] it is shown that this model satisfies the following *Weak Synchrony* property: *if any process completes round k without receiving round k message from a process p_i, then p_i has crashed before completing round $k+1$.*

In each of the three models weaker than the synchronous model (*ES*, synchronous send-omission, *WS*) a run is synchronous if the following holds: *in every round $k \geq 1$, if some process completes round k without receiving round k message from p_i then p_i has crashed before completing round k.*

We prove our lower bound result on *WS* and then extend it to the other two models. Let A be any uniform consensus algorithm in *WS*. *Synchronous configurations* are the configurations of synchronous runs of A. For any synchronous configuration C at round k of A, we define $R(C)$ as the synchronous run in which (1) the configuration at round k is C and (2) no process crashes after round k. We denote by $Val(C)$ the decision value of correct processes in $R(C)$. As every run in sub_{scs1} (Appendix A, proof of Proposition 3(b)) is a synchronous run in *WS*, from Claim 3(c) we immediately have in *WS*:

Claim WS1. There are two synchronous configurations at the end of round f ($0 \leq f \leq t$) of any uniform consensus algorithm in *WS*, y and y', such that, (1) at most f processes have crashed in each

[13] A Perfect failure detector is a distributed oracle, which outputs a list of suspected processes at every process such that: (a) (Strong Completeness) eventually every process that crashes is permanently suspected by every correct process, and (b) (Strong accuracy) no process is suspected before it crashes.

configuration, (2) the configurations differ at exactly one process, and (3) $Val(y) = 0$ and $Val(y') = 1$.

Lemma 27 *Let $1 \leq t \leq n-1$. For every uniform consensus algorithm in* WS *and every $0 \leq f \leq t-3$, there is a synchronous run with at most f failures where no correct process decides before round $f+2$.*

Proof: Suppose by contradiction that there is a binary uniform consensus algorithm A in *WS* and an integer f such that $0 \leq f \leq t-3$ and in every synchronous run of A with at most f failures, some correct process decides in round $f+1$ or in a lower round. From Claim *WS*, we know that at the end of round f there are two synchronous configurations of A, y and y', such that, (1) at most f processes have crashed in each configuration, (2) the configuration differ at exactly one process, say p_i, and (3) $Val(y) = 0$ and $Val(y') = 1$. Let z and z' denote the configuration at the end of round $f+1$ in synchronous runs $R(y)$ and $R(y')$, respectively.

From our initial assumption on A, in z, there is at least one alive process, say q_1, which has decided 0. Similarly, in z', there is at least one alive process, say q_2, which has decided 1. There are following four cases:

Case 1. $p_i \notin \{q_1, q_2\}$ and p_i is alive in y and y'. Consider the following two synchronous runs of A.
R1 is a run such that (1) configuration at the end of round f is y, (2) p_i crashes in the send phase of round $f+1$ such that only q_1 receives the message from p_i, (3) q_1 and q_2 crashes before sending any message in round $f+2$, and (4) no process distinct from p_i, q_1, and q_2 crashes after round f. Notice that, q_1 cannot distinguish the configuration at the end of round $f+1$ in $R1$ from z, and therefore, decides 0 at the end of round $f+1$ in $R1$. By uniform agreement, every correct process decides 0 in $R1$. Since $t \leq n-1$, there is at least one correct process in $R1$, say p_l.
R2 is a run such that (1) configuration at the end of round f is y', (2) p_i crashes in the send phase of round $f+1$ such that only q_2 receives the message from p_i, (3) q_1 and q_2 crashes before sending any message in round $f+2$, and (4) no process distinct from p_i, q_1, and q_2 crashes after round f. Notice that, q_2 cannot distinguish the configuration at the end of round $f+1$ in $R2$ from z', and therefore, decides 1 at the end of round $f+1$ in $R2$. However, p_l can never distinguish $R1$ from $R2$: at the end of round $f+1$, the two runs are different only at p_i, q_1, and q_2, and none of the three processes send messages after round $f+1$ in both runs. Thus, (as in $R1$) p_l decides 0 in $R2$; a contradiction of uniform agreement.

Case 2. $p_i \notin \{q_1, q_2\}$ and p_i has crashed in either y or y'. Without loss of generality we can assume that p_i has crashed in y, and hence, p_i is alive in y'. Consider the following two synchronous runs.
R12 is a run such that (1) configuration at the end of round f is y (and hence, p_i has crashed before round $f+1$), (2) no process crashes in round $f+1$, and (3) q_1 and q_2 crash before sending any message in round $f+2$. No process distinct from p_i, q_1 and q_2 crashes after round f. Notice that, q_1 cannot distinguish the configuration at the end of round $f+1$ in R12 from z because q_1 does not receive the round $f+1$ message from p_i in both runs. Thus, (as in z) q_1 decides 0 at the end of round $f+1$ in R12. Due to uniform agreement, every correct process decides 0 in R12. Since $t \leq n-1$, there is at least one correct process in R12, say p_l.
R21 is a run such that (1) configuration at the end of round f is y', (2) p_i crashes in the send phase of round $f+1$ such that only q_2 receives the message from p_i, and (3) q_1 and q_2 crash before sending any message in round $f+2$. No process distinct from p_i, q_1 and q_2 crashes after round f. Notice that, q_2 cannot distinguish the configuration at the end of round $f+1$ in R21 from z' because q_2 receives the message from p_i in both runs. Thus, (as in z') q_2 decides 1 at the end of round $f+1$ in R21. However, p_l can never distinguish R12 from R21: at the end of round $f+1$, the two configurations are different only at p_i, q_1 and q_2, and none of them send messages after round $f+1$ in both runs. Thus, (as in R12), p_l decides 0 in R21; a contradiction of uniform agreement.

Case 3. $p_i \in \{q_1, q_2\}$ and p_i is alive in both y and y'. Notice that if $p_i = q_1$ then $R1$ is not in WS; p_i cannot crash in the send phase of round $f+1$, and decide at the end of round $f+1$. (Similarly, if $p_i = q_2$ then $R2$ is not in WS.) Thus, we construct non-synchronous runs of A to show the contradiction. Without loss of generality we can assume that $p_i = q_1$. (Note that the proof holds even if $p_i = q_1 = q_2$.) Consider the following synchronous run $R3$ and two *non-synchronous* runs, $R4$ and $R5$.

R3 is a run such that (1) configuration at the end of round f is y, (2)p_i crashes in round $f+1$ before sending any message, (3) if $q_2 \neq p_i$ then q_2 crashes before sending any message in round $f+2$ and every message send by q_2 in round $f+1$ is received in the same round, and (4) no process distinct from p_i and q_2 crashes after round f. Since $t \leq n-1$, there is at least one correct process in $R3$, say p_l. Suppose p_l decides $v \in \{0,1\}$ in some round $K' \geq f+1$.[14]

R4 is a run such that (1) configuration at the end of round f is y, (2) p_i crashes before sending any message in round $f+2$, such that, in round $f+1$, every message from p_i to any process distinct from p_i and q_2 is delayed till round $K'+1$, (3) if $q_2 \neq p_i$ then q_2 crashes before sending any message in round $f+2$ and every message send by q_2 in round $f+1$ is received in the same round, and (4) no process distinct from p_i and q_2 crashes after round f. Notice that, p_i cannot distinguish the configuration at the end of round $f+1$ in $R4$ from z (because p_i receives its own message in round $f+1$), and thus, p_i decides 0 at the end of round $f+1$ in $R4$. However, p_l cannot distinguish the configuration at the end of round K' in $R4$ from that in $R3$ because (1) at the end of round f the two runs are different only at p_i, and all round $f+1$ messages from p_i to processes distinct from p_i and q_2 are delayed till round $K'+1$, and (2) p_i and q_2 does not send messages after round $f+1$. Thus, (as in $R3$) p_l decides v at the end of round K'.

R5 is a run such that (1) configuration at the end of round f is y', (2) p_i crashes before sending any message in round $f+2$, such that, in round $f+1$, every message from p_i to any process distinct from p_i and q_2 is delayed till round $K'+1$, (3) if $q_2 \neq p_i$ then q_2 crashes before sending any message in round $f+2$ and every message send by q_2 in round $f+1$ is received in the same round, and (4) no process distinct from p_i and q_2 crashes after round f. Notice that, q_2 cannot distinguish the configuration at the end of round $f+1$ in $R5$ from z' ((because q_2 receives the message from p_i in round $f+1$), and thus, q_2 decides 1 at the end of round $f+1$ in $R5$. However, p_l cannot distinguish the configuration at the end of round K' in $R5$ from that in $R3$ because, (1) at the end of round f the two runs are different only at p_i, and all round $f+1$ message from p_i to processes distinct from p_i and q_2 are delayed till round $K'+1$, and (2) p_i and q_2 does not send messages after round $f+1$. Thus, (as in $R3$) p_l decides v at the end of round K'.

It is easy to see that either $R4$ or $R5$ violates agreement: p_l decides v in both runs, however, p_i decides 0 in $R4$ and q_2 decides 1 in $R5$.

Case 4. $p_i \in \{q_1, q_2\}$ and p_i has crashed in either y or y'. Notice that the case $p_i = q_1 = q_2$ is not possible because, in that case, p_i is alive in z and z', and hence in y and y'. We show the contradiction for the case when $p_i = q_1 \neq q_2$. (The contradiction for $p_i = q_2 \neq q_1$ is symmetric.)

Since, $p_i = q_1$, p_i is alive in z, and hence, alive in y. Thus, p_i has crashed in y'. Consider the following non-synchronous runs.

R6 is a run such that (1) configuration at the end of round f is y, (2) p_i crashes before sending any message in round $f+2$, such that, in round $f+1$, every message from p_i to a process distinct from p_i, is delayed till round $f+2$, and (3) no process distinct from p_i crashes after round f. At the end of round $f+1$ in $R6$, $p_i = q_1$ cannot distinguish the configuration from z (because p_i receives its own message in round $f+1$), and therefore, decides 0 at the end of round $f+1$ in $R6$. However, q_2 does not receive

[14]To see that p_l cannot decide before round $f+1$ in $R3$ notice that the state of p_l at the end of round f is the same in runs $R(y)$, $R(y')$ and $R3$. If p_l decides v before round $f+1$ in $R3$ then it also decides v in $R(y)$ and $R(y')$. However, $Val(y) \neq Val(y')$.

the round $f+1$ message from p_i in $R6$ (the message is delayed till the next round), and furthermore, even in z', q_2 does not received the round $f+1$ from p_i (because p_i has crashes in y'). Thus, q_2 cannot distinguish the configuration at the end of round $f+1$ in $R6$ from z', and hence, decides 1 in $R6$; a contradiction of uniform agreement. □

Since every run in WS is a run is ES, and every synchronous run in WS is a synchronous run in ES, Lemma 27 immediately extends to ES.

Proposition 5 Let $1 \leq t \leq n-1$. For every uniform consensus algorithm in ES and every $0 \leq f \leq t-3$, there is a synchronous run with at most f failures where no correct process decides before round $f+2$.

However, some non-synchronous run in WS are not valid runs in the synchronous send-omission model: a delayed message to a correct process in WS is eventually received, but in the synchronous send-omission model, every delayed message is lost. In particular, $R4$, $R5$, and $R6$ (in the proof of Lemma 27) are not possible in the synchronous send-omission model. However, it is easy to see that the proof of Lemma 27 holds in synchronous send-omission model if we simply modify $R4$, $R5$, and $R6$ such that every delayed message is lost. (This actually simplifies the proof.)

B.2 A Matching Algorithm (Figure 4)

Figure 4 gives a uniform consensus algorithm A_{f+2} in the eventually synchronous model which matches the $f+2$ global decision lower bound (and hence, matches the local decision bound) in synchronous runs. Namely, the algorithm satisfies the following property:

Fast Early Decision: Let $0 \leq t < n/2$. In every synchronous run of A_{f+2} with at most f failures ($0 \leq f \leq t$), every process which decides, decides in round $f+2$ or in a lower round.

A_{f+2} assumes an independent uniform consensus algorithm C,[15] accessed by procedure propose$_C(*)$. The fast decision property is achieved by A_{f+2} regardless of the time complexity of C. More precisely, our algorithm assumes:

(1) the model ES with $0 \leq t < n/2$;

(2) messages send by a process to itself is received in the same round in which it is send;

(3) an independent uniform consensus algorithm C in ES;

(4) the set of proposal values in a run is a totally ordered set; e.g., every process p_i can tag its proposal value with its index i and then the values can be ordered based on this tag.

B.2.1 Description

The processes invoke propose$(*)$ with their respective proposal values, and the procedure progresses in round. Every process p_i maintains three primary variables:

- STATE$_i$ at the end of a round denotes the fact that p_i considers (a) the run to be non-synchronous (STATE = NSYNC), (b) the run to be synchronous but p_i cannot decide at the next round (STATE = SYNC1), (c) the run to be synchronous with a possibility of deciding at the next round (STATE = SYNC2).

- est_i is the estimate of the possible decision value, and roughly speaking, the minimum value seen by p_i.

[15] This algorithm can be any $\diamond P$-based or $\diamond S$-based uniform consensus algorithm (e.g., the one based on $\diamond S$ in [5]) transposed to ES.

- $Halt_i$ is a set of processes. At the end of a round, $Halt_i$ contains p_j if any of the following holds in the current round or in a lower round: (1) p_i did not receive a message from p_j, (2) p_i receives a messages from p_j with STATE $=$ NSYNC, or (3) p_i receives a messages from p_j with $p_i \in Halt_j$.

In the first $t+2$ rounds, the processes exchange these three variables and then updates their variable depending on the messages received. We say that a message is a state S$'$ message, if it is sent with STATE $=$ S$'$. Figure 5 shows the rules for updating STATE in each round k. At the end of round $t+2$, if a process has not yet decided, then it invokes the underlying uniform consensus C with its est as the proposal value. The algorithm ensures the following *elimination property*: if a process completes some round $k < t+2$ with STATE $=$ SYNC2 and $est = est'$ and no process decides in round k or in a lower round, then every process which completes round k with STATE $=$ SYNC1 has $est \geq est'$, and every process which completes round k with STATE $=$ SYNC2 has $est = est'$. (Processes which complete round k with STATE $=$ NSYNC may have $est < est'$.)

We now briefly discuss the uniform agreement property of our algorithm assuming the elimination property. If every process which decides, decides at a round higher than $t+2$ then uniform agreement follows from the corresponding property of algorithm C. Consider the lowest round $k' \leq t+2$ in which some process p_i decides, say d. From line 14, at least $n-t$ processes (a majority) completes round $k'-1$ with STATE $=$ SYNC2, and hence, every process which completes round $k-1$ receives a message with STATE $=$ NSYNC and $est = d$. From the elimination property, processes which complete round $k'-1$ with $est < d$ has STATE $=$ NSYNC. Notice that, while updating est for the next round, processes with STATE $=$ SYNC1 orSTATE $=$ SYNC2, ignore messages from processes with STATE $=$ NSYNC (line 11, line 12). Therefore, every process with STATE $=$ SYNC1 or STATE $=$ SYNC2, updates est to d (line 13). Since a majority of processes sends round k' messages with $est = d$ and STATE $=$ SYNC2, every process which completes round k' with STATE $=$ NSYNC updates est to d (line 22). Consequently, every process which completes round k', does so with $est = d$, and no value distinct from d can be decided at round k' or at a higher round.

B.2.2 Correctness

The validity, termination, and integrity properties of A_{f+2} easily follow from the corresponding properties of the underlying uniform consensus algorithm C. We focus here on the uniform agreement and the fast early decision properties. For presentation simplicity, we introduce the following notation. Given a variable val_i at process p_i, we denote by $val_i[k]$ $(1 \leq k \leq t+2)$ the value of the variable val_i immediately after the completion of round k; $val_i[0]$ denotes the value of val_i immediately after completing line 4 (i.e., before sending any message in round 1). We assume that there is a symbol *undefined* which is distinct from any possible value of the variables in the algorithm A_{f+2}. If p_i crashes before completing round k, then $val_i[k] = undefined$; if p_i crashes before completing line 4, then $val_i[0] = undefined$. In other words, if for any variable val, $val_i[k] \neq undefined$ then p_i completed round k.

Lemma 29: *Consider a process p_l and a round $1 \leq k \leq t+2$, such that* STATE$_l[k] \in$ {SYNC*1*, SYNC*2*} *(p_l completes round k with* STATE $=$ SYNC*1* *or* STATE $=$ SYNC*2). Let $senderMS_l[k]$ be the set of processes which have sent the messages in $msgSet_l[k]$. Then, $senderMS_l[k] = \Pi - Halt_l[k]$.*

Proof: Process p_l completes round k with STATE $=$ SYNC1 or STATE $=$ SYNC2, and hence, updates $Halt$ and $msgSet$ at line 11 and line 12 of round k, respectively. Consider any process $p_m \in \Pi$. There are two exhaustive and mutually exclusive cases regarding the message from p_m to p_l in round k $(1 \leq k \leq t+2)$:

- If p_l does not receive the messages from p_m in round k, then from the third condition in line 11, $p_m \in Halt_l[k]$, and from line 12, $p_m \in senderMS_l[k]$.

at process p_i

1: **procedure** propose(v_i)
2: **start Task1; start Task 2**

3: **Task 1**
4: STATE$_i \leftarrow$ SYNC1 ; $est_i \leftarrow v_i$; $Halt_i \leftarrow \emptyset$
5: **for** $1 \leq k_i \leq t+2$
6: send(k_i, est_i, STATE$_i$, $Halt_i$) to all
7: **wait until** received messages in this round
8: **if** received(k_i, est', DECIDE, $*$) **then**
9: send(k_i+1, est', DECIDE, $\emptyset$) to $\Pi \backslash p_i$, return(est') {*decision*}
10: **if** STATE$_i \in$ {SYNC1, SYNC2} **then**
11: $Halt_i \leftarrow Halt_i \cup \{p_j \mid (p_i$ received(k_i, $*$, NSYNC, $*$) from $p_j)$ **or**
(p_i received(k_i, $*$, $*$, $Halt_j$) from p_j s.t. $p_i \in Halt_j$) **or** (p_i did not receive round k_i message from p_j)}
12: $msgSet_i \leftarrow \{ m \mid m$ is a round k_i message received from $p_j \notin Halt_i\}$
13: $est_i \leftarrow$ **Min**$\{est \mid (k_i, est, *, *) \in msgSet_i\}$
14: **if** (STATE$_i$ = SYNC2) **and** ($|Halt_i| \leq t$) **and** (STATE = SYNC2 for every message in $msgSet_i$) **then**
15: send(k_i+1, est_i, DECIDE, $\emptyset$) to $\Pi \backslash p_i$, return(est_i) {*decision*}
16: **if** $|Halt_i| \leq k_i - 1$ **then**
17: STATE$_i \leftarrow$ SYNC2
18: **if** $k_i \leq |Halt_i| \leq t$ **then**
19: STATE$_i \leftarrow$ SYNC1
20: **if** $|Halt_i| > t$ **then**
21: STATE$_i \leftarrow$ NSYNC
22: **if** (STATE = NSYNC) **and** (received(k_i, est', SYNC2, $*$)) **then**
23: $est_i \leftarrow est'$
24: return(propose$_C(est_i)$)

25: **Task 2**
26: **upon** receiving (k', est', DECIDE, $*$) **do**
27: when $k_i = k' + 1$: send(k_i, est', DECIDE, $\emptyset$) to $\Pi \backslash p_i$, return(est') {*decision*}

Figure 4: A Uniform Consensus algorithm A_{f+2} in *ES*

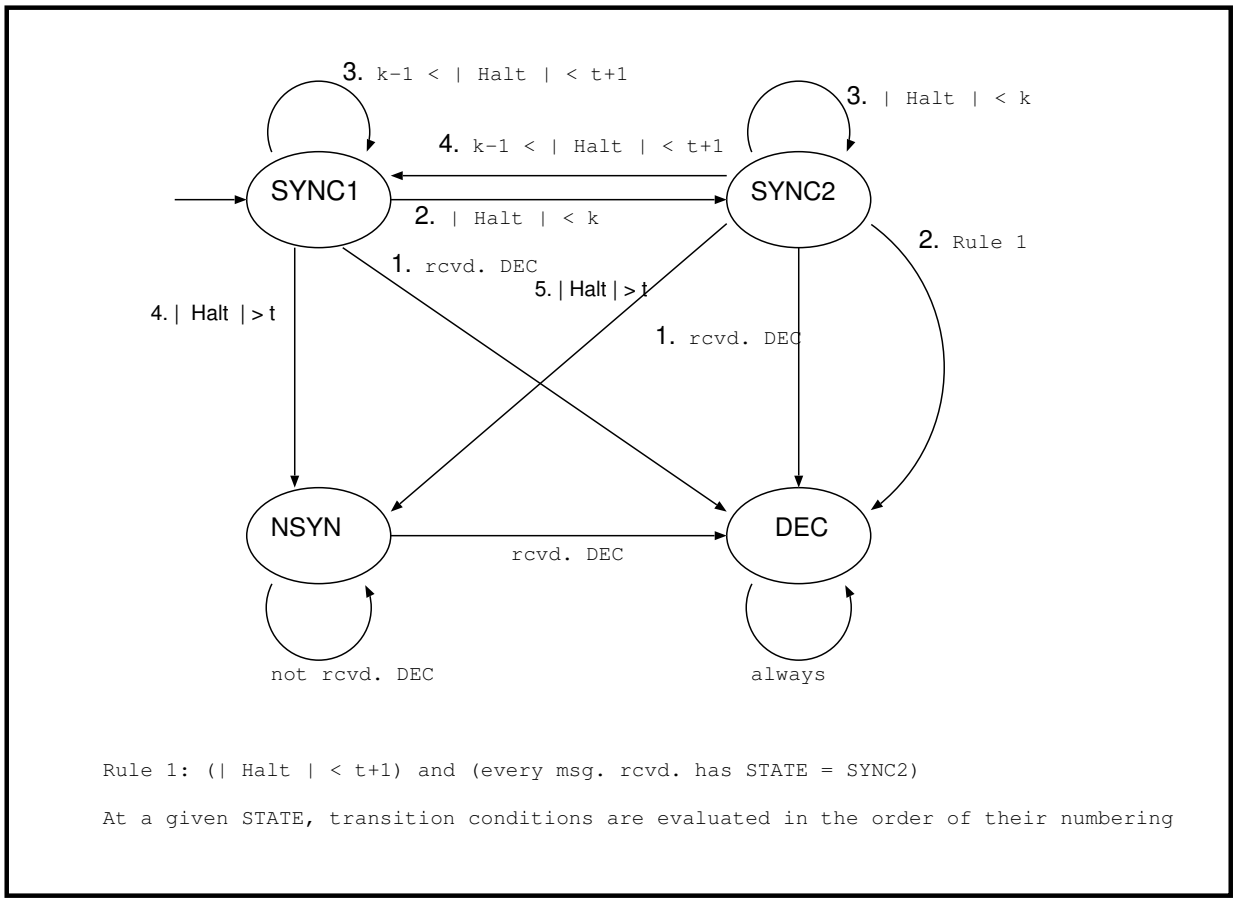

Figure 5: Rules for updating STATE at round k for process p_i (algorithm A_{f+2})

- If p_l receives the round the message from p_m in round k, then from line 12, $p_m \in senderMS_l[k]$ iff $p_m \notin Halt_l[k]$. □

Lemma 30. *(Uniform Agreement) No two processes decide differently.*
Proof. If no process ever decides then the lemma is trivially true. If every process which decides, decide in algorithm C, then the lemma follows from the uniform agreement property of C. Thus, we consider the case where some process decides within the first $t+2$ rounds. Consider the lowest round number in which some process decides, say round $k'+1 (\leq t+2)$. It is easy to see that, if some process decides v in line 9 or line 27, then some other process has decided v in a lower round. Thus, some process decides at line 15 of round $k'+1$. We claim the following:

Claim 30.1: *If there are two processes p_x and p_y such that* $\text{STATE}_x[k'] \in \{\text{SYNC}1, \text{SYNC}2\}$ *and* $\text{STATE}_y[k'] = \text{SYNC}2$ *then* $est_x[k'] \geq est_y[k']$.

[Proof of Lemma 30 cont.] We now complete the proof of uniform agreement assuming Claim 30.1. We later give the proof of Claim 30.1. Suppose that some process p_w decides d at line 15 of round $k'+1$. From line 14 it follows that p_w has completed round k' with STATE = SYNC2 and $est = d$. Consider another process p_u which completes round k' with STATE = SYNC2 and $est = d'$. In Claim 30.1, if we substitute p_x by p_w and p_y by p_u then, $d \geq d'$. Similarly, if we substitute p_x by p_u and p_y by p_w then, $d' \geq d$. Thus, $d' = d$, and any process which completes round k' with STATE = SYNC2, does so with $est = d$. Notice that, every process which decides at line 15 in round $k'+1$, completes round k' with STATE = SYNC2 and decides on its own est (line 14, line 15). Thus, every process which decides in round $k+1$ decides d. It remains to be shown that no process decides a different value in a higher round.

From line 14 we have $|Halt_w[k'+1]| \leq t$, and hence, Lemma 29 implies that $msgSet_w[k'+1]$ contains at least $n-t$ messages, i.e., messages from a majority of processes. Furthermore, the last condition in line 14 requires that all messages in $msgSet_w[k'+1]$ has STATE = SYNC2. Applying Claim 30.1, we have, in round $k'+1$, messages from a majority of processes have STATE = SYNC2 and $est = d$, and every message with STATE = SYNC1 has $est \geq d$.

Now consider the est value of any process p_i at the end of round $k'+1$. If $\text{STATE}_i[k'+1] = \text{NSYNC}$, then p_i has received at least one message with STATE = SYNC2 and $est = d$ (because a majority of processes send such messages, and in every round p_i receives messages from a majority of processes), and therefore, updates its est to d (line 22). If $\text{STATE}_i[k'+1] \neq \text{NSYNC}$ then $Halt_i[k'+1] \leq t$ (line 20). Therefore, $msgSet_i[k'+1]$ contains at least $n-t$ messages (Lemma 29). Furthermore, $msgSet_i[k'+1]$ contains no message with $\text{STATE}_i[k'+1] = \text{NSYNC}$ (line 11, line 12). Therefore, from Claim 30.1, every message in $msgSet_i[k'+1]$ has $est \geq d$ and at least one message with STATE = SYNC2 and $est = d$ (because a majority of processes sent messages with STATE = SYNC2 and $est = d$ in round $k'+1$). Therefore, at line 13, p_i updates est to d.

Thus, every process which completes round $k'+1$ updates its est to d, and every process which decides at line 15 of round $k'+1$, decides d. Now notice that the est value of a process at the end of some round k is est value of some process at the end of round $k-1$ $(1 < k \leq t+2)$. Therefore, for round k such that $k'+1 \leq k \leq t+2$, no process completes round k with est different from d (**D1**). Notice that, if a process decides d' at line 15 of round k such that $k'+1 \leq k \leq t+2$, then its est is d' at the end of round $k-1$. Therefore, from D1, $d' = d$. Furthermore, proposal value for the underlying consensus algorithm C at a given process p_i is the est value of p_i at the end of round $t+2$. Hence, from D1, every proposal value for algorithm C is d, and from validity property of C, every process which decides in algorithm C, decides d. □

Claim 30.1: *If $k'+1 \leq t+2$ is the lowest round in which some process decides then: if there*

are two processes p_x and p_y such that $\text{STATE}_x[k'] \in \{\text{SYNC}1,\text{SYNC}2\}$ *and* $\text{STATE}_y[k'] = \text{SYNC}2$ *then* $est_x[k'] \geq est_y[k']$.
Proof: Suppose by contradiction that there are two processes p_x and p_y such that

Assumption **A1**: $\text{STATE}_x[k'] \in \{\text{SYNC1},\text{SYNC2}\}$, $\text{STATE}_y[k'] = \text{SYNC2}$, $est_x[k'] = c$, $est_y[k'] = d$, and $c < d$.
We show Claims 30.1.1 to 30.1.7 based on the definition of k' and the assumption A1. Claim 30.1.4 contradicts Claim 30.1.7, which completes the proof of Claim 30.1 by contradiction.

Let us define the following sets for $1 \leq k \leq k' + 1$:

- $C[k] = \{p_i | est_i[k] \leq c\}$ (Set of processes which complete round k with $est \leq c$.)
- $crashed[k]$= set of processes which crashes before completing round k.
- $NSYN[k] = \{p_i | \text{STATE}_i[k] = \text{NSYNC}\}$.
- $Z[k] = C[k] \cup crashed[k] \cup NSYN[k]$.

Additionally, let us define, $C[0]$ to be the set of processes whose proposal value is less than or equal to c, $crashed[0]$ to be the set of processes which crash before sending any message in round 1, $NSYN[0] = \emptyset$, and $Z[0] = C[0] \cup crashed[0] \cup NSYN[0]$. We make the following observation:
Observation **A2**: $|C[0]| \geq 1$, and hence, $|Z[0]| \geq 1$. Otherwise, if every process proposed a value greater than c, then $est_x[k'] > c$ (contradicts A1).

Claim 30.1.1: (a) For $0 \leq k \leq k' - 1$, $(crashed[k] \cup NSYN[k]) \subseteq (crashed[k+1] \cup NSYN[k+1])$. (b) For $0 \leq k \leq k' - 1$, if $p_i \notin (NSYN[k] \cup crashed[k])$ then p_i sends messages with $\text{STATE} \in \{\text{SYNC1}, \text{SYNC2}\}$ in round k and in the lower rounds.
Proof: (a) Suppose by contradiction that there is process p_i such that $p_i \in crashed[k] \cup NSYN[k]$ and $p_i \notin crashed[k+1] \cup NSYN[k+1]$. Obviously, $crashed[k] \subseteq crashed[k+1]$, and hence,$p_i \notin crashed[k+1] \cup NSYN[k+1]$ implies $p_i \notin crashed[k]$. Then,$p_i \in crashed[k] \cup NSYN[k]$ implies $p_i \in NSYN[k]$, i.e., p_i completes round k with $\text{STATE} = \text{NSYNC}$. Notice that, by the definition of k' (i.e., $k'+1$ is the lowest round in which some process decides), p_i does not decide in round $k+1$. Thus, the STATE of p_i remains NSYNC at the end of round $k+1$, i.e., $p_i \in NSYN[k+1]$; a contradiction.
(b) If $p_i \notin (NSYN[k] \cup crashed[k])$, then from 30.1.1.a, it follows that, $p_i \notin (NSYN[k_1] \cup crashed[k_1])$ for $0 \leq k_1 \leq k$; i.e., p_i completes every round lower than round k with $\text{STATE} \neq \text{NSYNC}$. Thus, p_i cannot send message with $\text{STATE} \neq \text{NSYNC}$in round k or in a lower round. □

Claim 30.1.2: $Z[k] \subseteq Z[k+1]$ $(0 \leq k \leq k' - 1)$.
Proof: Suppose by contradiction that there is a process p_i and a round number k such that $p_i \in Z[k]$ and $p_i \notin Z[k+1]$. Since $p_i \notin Z[k+1]$, then $p_i \notin crashed[k+1] \cup NSYN[k+1]$. Applying Claim 30.1.1.a, we get $p_i \notin crashed[k] \cup NSYN[k]$. However, $p_i \in Z[k] = C[k] \cup crashed[k] \cup NSYN[k]$, and hence, $p_i \in C[k]$.

Since $p_i \notin crashed[k]$, $p_i \notin NSYN[k]$, and $p_i \in C[k]$, p_i sends round $k+1$ message m' with $est \leq c$ and $\text{STATE} \neq \text{NSYNC}$. As $p_i \notin crashed[k+1] \cup NSYN[k+1]$, so p_i evaluates est in line 13 of round $k'+1$. From Claim 30.1.1.b and $p_i \notin NSYN[k]$, it follows that p_i never sends a message with $\text{STATE} = \text{NSYNC}$ at round k or at a lower round. Since a process always receives the message sent to itself without a delay and p_i never sends a message with $\text{STATE} = \text{NSYNC}$ at round k or at a lower round, $p_i \notin Halt_i[k+1]$. Applying Lemma 29 we have, $p_i \in senderMS_i[k+1]$, and therefore, $m' \in msgSet_i[k+1]$. Thus, when p_i evaluate est in round $k+1$, it consider message m' with $est \leq c$,

and adopts a values less than equal to c as the new est. Thus, $p_i \in C[k+1] \subseteq Z[k+1]$; a contradiction. □

Claim 30.1.3: $0 \leq k \leq k'-1, \forall p_i \notin Z[k+1], Z[k] \subseteq Halt_i[k+1]$.
Proof: Consider a process $p_j \in Z[k]$ and a process $p_i \notin Z[k+1]$. In round $k+1$, $msgSet_i[k+1]$ either contains a message from p_j or does not contain any message from p_j. In the second case, Lemma 29 implies that $p_j \in Halt_i[k+1]$. Consider the case when $msgSet_i[k+1]$ contains a message m from p_j. From line 11 and line 12, it follows that, m has STATE $\neq$ NSYNC, and hence, $p_j \notin NSYN[k]$. Furthermore, p_j sent a message in round $k+1$, and so, $p_j \notin crashed[k]$. Thus, $p_j \in Z[k]$ but $p_j \notin crashed[k] \cup NSYN[k]$. So, $p_j \in C[k]$. Thus, m has $est \leq c$, and hence, $est_i[k+1] \leq c$. Thus, $p_i \in C[k+1] \subseteq Z[k+1]$; a contradiction. Thus, $msgSet_i[k+1]$ does not contains a message m from p_j. □

Claim 30.1.4: $|Z[k'-1]| \leq k'-1$.
Proof: Suppose by contradiction $|Z[k'-1]| > k'-1$. From A1, it follows that $p_y \notin Z[k']$. Therefore, from claim 30.1.3, $Z[k'-1] \subseteq Halt_y[k']$. Hence, $|Halt_y[k']| > k'-1$. However, $\text{STATE}_y[k'] = \text{SYNC2}$ implies that $|Halt_y[k']| \leq k'-1$ (line 16, line 17), a contradiction. □

Claim 30.1.5: $p_x \in Z[k']$ *and* $p_x \notin Z[k'-2]$.
Proof: As $est_x[k'] = c$, so $p_x \in C[k'] \subseteq Z[k']$.

For the second part of the claim, suppose by contradiction that $p_x \in Z[k'-2]$. From Claim 30.1.3, for every process $p_i \notin Z[k'-1]$, $p_x \in Halt_i[k'-1]$. Therefore, in round k', if any process in $\Pi - Z[k'-1]$ sends a message m, then $p_x \in m.Halt$ (where, $m.Halt$ denotes the $Halt$ field of m). If p_x receives m then it includes the sender of m in $Halt_x$ (condition 2, line 11), and even if p_i does not receive m then it includes the sender of m in $Halt_x$ (condition 3, line 11). Thus, $\Pi - Z[k'-1] \subseteq Halt_x[k']$. Using, Claim 30.1.4, $|Halt_x[k']| \geq |\Pi - Z[k'-1]| \geq n - (k'-1)$. Since $k'+1 \leq t+2$ and $t < n/2$, we have $|Halt_x[k']| \geq n - t > t$. However, $|Halt_x[k']| > t$ implies that $\text{STATE}_x[k'] = \text{NSYNC}$ (line 20, line 21); a contradiction. □

Claim 30.1.6: *(1) For every* k *such that* $0 \leq k \leq k'-3$*:* $Z[k] \subset Z[k+1]$. *(*$Z[k]$ *is a proper subset of* $Z[k+1]$*). (2)* $|Z[k'-2]| \geq k'-1$.
Proof: (1) Recall from Claim 30.1.2 that $Z[k] \subseteq Z[k+1]$ $(0 \leq k \leq k'-1)$. Suppose by contradiction that there is a round number s $(0 \leq s \leq k'-3)$, such that $Z[s] = Z[s+1]$.

We first show by induction on the round number k that, for $s+1 \leq k \leq k'-1$, $C[k] - (NSYN[k] \cup crashed[k]) \supseteq C[k+1] - (NSYN[k+1] \cup crashed[k+1])$.

Base Case ($k = s+1$)*:* $C[s+1] - (NSYN[s+1] \cup crashed[s+1]) \supseteq C[s+2] - (NSYN[s+2] \cup crashed[s+2])$. Suppose by contradiction that there is a process p_i such that $p_i \in C[s+2] - (NSYN[s+2] \cup crashed[s+2])$ (**A4**) and $p_i \notin C[s+1] - (NSYN[s+1] \cup crashed[s+1])$ (**A5**).

A4 implies that $p_i \notin NSYN[s+2] \cup crashed[s+2]$. Applying Claim 30.1.1, we have $p_i \notin NSYN[s+1] \cup crashed[s+1]$, and therefore, from A5 it follows that $p_i \notin C[s+1]$. Thus, p_i completes round $s+1$ with $est > c$. Furthermore, A4 implies that $p_i \in C[s+2]$, and hence, p_i completes round $s+2$ with $est \leq c$. So, $msgSet_i[s+2]$ contains a message with $est \leq c$ from some process p_j (i.e., $p_j \in senderMS_i[s+2]$). From the definition of $Z[s+1]$, it follows that $p_j \in C[s+1] \subseteq Z[s+1]$.

As $p_i \notin NSYN[s+1] \cup crashed[s+1]$ and $p_i \notin C[s+1]$, so from the definition of $Z[s+1]$ we have $p_i \notin Z[s+1]$. Claim 30.1.3 implies that $Z[s] \subseteq Halt_i[s+1]$. Recall that we assumed $Z[s] = Z[s+1]$ and, from line 11, $Halt_i[s+1] \subseteq Halt_i[s+2]$. Therefore, $Z[s+1] \subseteq Halt_i[s+2]$. Thus, $p_j \in C[s+1] \subseteq Z[s+1]$ implies that $p_j \in Halt_i[s+2]$. Therefore, $p_j \in senderMS_i[s+2] \cap Halt_i[s+2]$.

As $p_i \notin NSYN[s+2] \cup crashed[s+2]$, then p_i completed round $s+2$ with STATE = SYNC1 or STATE = SYNC2. From Lemma 29 it follows that $senderMS_i[s+2] \cap Halt_i[s+2] = \emptyset$. However, $p_j \in senderMS_i[s+2] \cap Halt_i[s+2]$; a contradiction.

Induction Hypothesis ($s+1 \leq k \leq r < k'-1$): $C[k]-(NSYN[k]\cup crashed[k]) \supseteq C[k+1]-(NSYN[k+1]\cup crashed[k+1])$.

Induction Step ($k = r+1$): $C[r+1]-(NSYN[r+1]\cup crashed[r+1]) \supseteq C[r+2]-(NSYN[r+2]\cup crashed[r+2])$. Suppose by contradiction that there is a process p_i such that $p_i \in C[r+2]-(NSYN[r+2]\cup crashed[r+2])$ (**A6**) and $p_i \notin C[r+1]-(NSYN[r+1]\cup crashed[r+1])$ (**A7**).

As in the base case, using A6, A7, and Claim 30.1.1, we can show $p_i \notin NSYN[r+2]\cup crashed[r+2]$, $p_i \notin NSYN[r+1]\cup crashed[r+1]$, and $p_i \notin C[r+1]$. Thus, $p_i \notin Z[r+1]$. Since $s+1 < r+1$, from Claim 30.1.2, we have $Z[s+1] \subseteq Z[r+1]$, and therefore, $p_i \notin Z[s+1]$.

Applying Claim 30.1.3 on $p_i \notin Z[s+1]$ implies that $Z[s] \subseteq Halt_i[s+1]$. Recall that we assumed $Z[s] = Z[s+1]$, and from line 11, $Halt_i[s+1] \subseteq Halt_i[r+2]$. Therefore, $Z[s+1] \subseteq Halt_i[r+2]$ (**A8**).

From induction hypothesis, we have $(C[s+1]-(NSYN[s+1]\cup crashed[s+1])) \supseteq (C[r+1]-(NSYN[r+1]\cup crashed[r+1]))$. From the definition of $Z[s+1]$, $C[s+1]-(NSYN[s+1]\cup crashed[s+1]) \subseteq C[s+1] \subseteq Z[s+1]$, and therefore, $C[r+1]-(NSYN[r+1]\cup crashed[r+1]) \subseteq Z[s+1]$. Applying A8, we have $(C[r+1]-(NSYN[r+1]\cup crashed[r+1])) \subseteq Halt_i[r+2]$ (**A9**).

As $p_i \notin Z[r+1]$, p_i completes round $r+1$ with $est > c$. Furthermore, A6 implies that $p_i \in C[r+2]$, and hence, p_i completes round $r+2$ with $est \leq c$. Therefore, $msgSet_i[r+2]$ contains a message with $est \leq c$ from some process p_j (i.e., $p_j \in senderMS_i[r+2]$). From the definition of $Z[r+1]$, it follows that $p_j \in C[r+1] \subseteq Z[r+1]$.

As the round $r+2$ message of p_j is in $msgSet_i[r+2]$, so from line 11 it follows that the message sent by p_j had STATE $\neq$ NSYNC. Therefore, $p_j \notin NSYN[r+1]$ and $p_j \notin crashed[r+1]$. Therefore, $p_j \in C[r+1]-(NSYN[r+1]\cup crashed[r+1])$. From A9 it follows that $p_j \in Halt_i[r+2]$.

As $p_i \notin NSYN[r+2]\cup crashed[r+2]$ (from A6), so p_i completed round $r+2$ with STATE = SYNC1 or STATE = SYNC2. Lemma 29 implies that $senderMS_i[r+2] \cap Halt_i[r+2] = \emptyset$. However, $p_j \in senderMS_i[r+2] \cap Halt_i[r+2]$; a contradiction.

From the above result, we have $(C[k'-2]-(NSYN[k'-2]\cup crashed[k'-2])) \supseteq C[k']-(NSYN[k']\cup crashed[k']))$. From A1, $p_x \in C[k']-(NSYN[k']\cup crashed[k']))$. From Claim 30.1.5, we have $p_x \notin Z[k'-2] \supseteq (C[k'-2]-(NSYN[k'-2]\cup crashed[k'-2]))$. Therefore, there is process in $C[k']-(NSYN[k']\cup crashed[k'])$ which is not in $C[k'-2]-(NSYN[k'-2]\cup crashed[k'-2])$; a contradiction.

(2) Part (1) of this lemma implies that for every k such that $0 \leq k \leq k'-3$, $|Z[k+1]|-|Z[k]| \geq 1$. We know from A4 that $|Z[0]| \geq 1$. Therefore, $|Z[k'-2]| \geq k'-1$. □

Claim 30.1.7: $|Z[k'-1]| > k'-1$.

Proof: Suppose by contradiction that $|Z[k'-1]| \leq k'-1$. Since $Z[k'-2] \subseteq Z[k'-1]$ (Claim 30.1.2) and $|Z[k'-2]| \geq k'-1$ (Claim 30.1.6.b), we have $Z[k'-2] = Z[k'-1]$ and $|Z[k'-2]| = |Z[k'-1]| = k'-1$ (**A10**).

We know from Claim 30.1.5 that $p_x \notin Z[k'-2] = Z[k'-1]$. Applying Claim 30.1.3, we have $Z[k'-2] \subseteq Halt_x[k'-1]$. As $Z[k'-2] = Z[k'-1]$ (from A10), it follows that $Z[k'-1] \subseteq Halt_x[k'-1]$.

Since, $p_x \notin Z[k'-1]$, p_x completes round $k'-1$ with $est > c$ and STATE $\neq$ NSYNC. From A1, we also know that p_x completes round k' with $est \leq c$ and STATE $\neq$ NSYNC. Therefore, $msgSet_x[k']$ contains a message, say from process p_j, with $est \leq c$ (i.e., $p_j \in senderMS_x[k']$). From the definition of $C[k'-1]$, $p_j \in C[k'-1] \subseteq Z[k'-1]$. However, we showed earlier that $Z[k'-1] \subseteq Halt_x[k'-1]$, and from line 11, it follows that $Halt_x[k'-1] \subseteq Halt_x[k']$. Thus, $Z[k'-1] \subseteq Halt_x[k']$ and $p_j \in Halt_x[k']$.

From A1, we know that p_x completed round k' with STATE = SYNC1 or STATE = SYNC2. Therefore, Lemma 29 implies that $senderMS_x[k'] \cap Halt_x[k'] = \emptyset$. However, $p_j \in senderMS_x[k'] \cap Halt_x[k']$; a

contradiction. □

Lemma 31. *In a synchronous run, consider any process p_i which completes round $k \leq t+2$. Every process in $Halt_i[k]$ crashes before completing round k.*

Proof. Let $H[l]$ $(0 \leq l \leq t+2)$ be the union of $Halt_j[l]$ such that $Halt_j[l] \neq$ *undefined*. We claim the following which immediately implies the lemma: *Every process in $H[l]$ $(0 \leq l \leq t+2)$ crashes before completing round l.*

We prove the claim by induction on round l. For $l = 0$, the lemma is trivially true, because $H[l] = \emptyset$ (base case). Suppose that the claim is true for $0 \leq l \leq l1-1 \leq t+1$: every process in $H[l]$ crashes before completing round l (induction hypothesis). Consider $H[l1]$ (induction step). If $H[l1] - H[l1-1] = \emptyset$ then the induction step is trivial. Suppose by contradiction that there is process $p_j \in H[l1] - H[l1-1]$ such that p_j completes round $l1$. Thus, there is a process p_a such that $p_j \notin H_a[l1-1]$ and $p_j \in H_a[l1]$.

Since p_j completes round $l1$ and the run is synchronous, in that round, p_a must have received the round $l1$ message m of p_j. Since, $p_j \in H_a[m]$, m contains either (a) STATE $=$ NSYNC or (b) $Halt_j$ such that $p_a \in Halt_j$. Now, we show both the cases to be impossible and thus prove the induction step by contradiction.

From our assumption, for every round lower than $l1$, every process in $Halt_j$ has crashed. Since more than t processes can never crash in a run, in rounds lower than $l1$, $|Halt_j|$ is never more than t. Thus, p_j can not update its STATE to NSYNC in rounds lower than $l1$ (line 20). Thus, the round $l1$ message from p_j does nor contain STATE $=$ NSYNC.

If the round $l1$ message from p_j contains $Halt_j$ such that $p_a \in Halt_j$ then $p_a \in Halt_j[l1-1] \subseteq H[l1-1]$. However, from our assumption, every process in $H[l1-1]$ crashes before completing round $l1-1$, which implies that p_a crashes before completing round $l1-1$; a contradiction. □

Lemma 32. *(Fast Early Decision) In every synchronous run of A_{f+2} with at most f failures $(0 \leq f \leq t < n/2)$, every process which decides, decides at round $f+2$ or at a lower round.*

Proof. Consider a synchronous run in which at most f processes fail. From Lemma 31, $|Halt|$ at every process is less than or equal to f in the first $t+2$ rounds (**A11**). Suppose by contradiction that some process p_i completes round $f+2$ but does not decides in that round. Then, either (1) $\text{STATE}_i[f+2] =$ NSYNC, or (2) some process p_j sent a message in round $f+2$ with STATE $=$ SYNC1. For case 1 to hold, $|Halt_i| > t$ in round $f+2$ or a lower round (line 20), which clearly violates Observation A11. For case 2 to be true, $\text{STATE}_j[f+1] =$ SYNC1, and therefore, $|Halt_j[f+1]| \geq f+1$ (line 18), which contradicts A11 as well. □