# Brief Announcement: Revisiting Lower Bounds for Two-Step Consensus

Fedor Ryabinin
IMDEA Software Institute
Universidad Politécnica de Madrid
Madrid, Spain

Alexey Gotsman
IMDEA Software Institute
Madrid, Spain

Pierre Sutra
Télécom SudParis, Inria Saclay
Institut Polytechnique de Paris
Palaiseau, France

## Abstract

A seminal result by Lamport shows that at least $\max\{2e + f + 1, 2f+1\}$ processes are required to implement partially synchronous consensus that tolerates $f$ process failures and can furthermore decide in two message delays under $e$ failures. This lower bound is matched by the classical Fast Paxos protocol. However, more recent practical protocols, such as Egalitarian Paxos, provide two-step decisions with fewer processes, seemingly contradicting the lower bound. We show that this discrepancy arises because the classical bound requires two-step decisions under a wide range of scenarios, not all of which are relevant in practice. We propose a more pragmatic condition for which we establish tight bounds on the number of processes required. Interestingly, these bounds depend on whether consensus is implemented as an atomic object or a decision task. For consensus as an object, $\max\{2e+f-1, 2f+1\}$ processes are necessary and sufficient for two-step decisions, while for a task the tight bound is $\max\{2e + f, 2f + 1\}$.

## CCS Concepts

• **Theory of computation → Distributed algorithms**.

## Keywords

Distributed algorithms, lower bounds, consensus

**ACM Reference Format:**
Fedor Ryabinin, Alexey Gotsman, and Pierre Sutra. 2025. Brief Announcement: Revisiting Lower Bounds for Two-Step Consensus. In *ACM Symposium on Principles of Distributed Computing (PODC '25), June 16–20, 2025, Huatulco, Mexico.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3732772.3733541

## 1 Introduction

*Context.* Consensus is a fundamental abstraction in distributed computing, widely used in practice for state-machine replication. It is well-known that partially synchronous consensus tolerating $f$ process crashes requires at least $2f + 1$ processes [5]. Protocols that match this bound, such as Paxos, are usually leader-driven. If the system is synchronous and the initial leader process is correct, these protocols can decide within two message delays; but if the initial leader is faulty, the latency increases. This issue is addressed by *fast* consensus protocols [8, 11] that can decide in two message delays under *any* process failures up to a given threshold $e \leq f$. This is usually done via a *fast path*, which avoids the leader. Lamport showed that fast consensus requires at least $\max\{2e + f + 1, 2f + 1\}$ processes [9], a bound matched by the Fast Paxos protocol [8]. But surprisingly, a more recent protocol called Egalitarian Paxos [10] decides within two message delays under $e = \lceil (f + 1)/2 \rceil$ failures while using only $2f + 1 = 2e + f - 1$ processes. Lamport's bound in this case requires $2f + 3 = 2e + f + 1$ processes. What's going on?

*Contribution.* To resolve this conundrum, we revisit Lamport's definition of fast protocols. Roughly speaking, the existing definition considers a consensus protocol *fast* if for every proposer $p$ and *every* correct process $q$, there exists a run in which $p$ is the only process that sends its proposal, and $q$ decides in two message delays. However, in practical protocols a client typically submits its proposal to one of the processes participating in consensus – a *proxy* – which replies to the client with the decision [13]. In this setting, we would like the proxy to decide fast, but the speed of the decision at other processes is irrelevant. We thus propose a more pragmatic definition that reflects this consideration. We consider a protocol fast if: *(i)* for all initial configurations, *some* process can decide within two message delays; and *(ii)* starting from configurations where all correct processes propose the same value, *every* correct process can decide in two message delays. We thus require fast decisions at all processes only when the proposals are the same; otherwise, only a single process is required to decide fast. We show that depending on formulation of consensus, satisfying our definition requires up to two fewer processes than Lamport's. This difference is practically significant for wide-area deployments, where contacting an additional process may incur a cost of hundreds of milliseconds per command.

The standard definition of consensus is that of a *decision task*: each process has an initial value, and processes try to agree on one of these. We first show that implementing a fast consensus task is possible if and only if the number of processes is at least $\max\{2e + f, 2f + 1\}$. An alternative definition of consensus is that of an *atomic object* [1], where processes explicitly specify their proposal $v$ by calling an operation `propose(v)`; in particular, this allows a process not to make any proposal at all. The consensus is required to be linearizable and wait-free. We show that this version of fast consensus can be solved if and only if the number of processes is at least $\max\{2e + f - 1, 2f + 1\}$.

*Related work.* In synchronous systems, bounds on consensus latency were investigated by Charron-Bost and Schiper [3]. Other work considered bounds on consensus in asynchronous systems

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
*PODC '25, Huatulco, Mexico*
© 2025 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-1885-4/2025/06
https://doi.org/10.1145/3732772.3733541

with failure detectors [2], showing that two-step decisions are possible if the failure detector selects a correct initial leader [4]. In this paper we do not rely on this assumption and require fast decisions without any additional information. The work in [6] shows that $2e + f - 1$ is a lower bound for a limited class of consensus object protocols constructed similarly to Egalitarian Paxos. In comparison to that work, our lower bounds apply to arbitrary consensus protocols, both task and object, and we provide matching upper bounds. Kuznetsov et al. [7] studied the relation between the thresholds $f$ and $e$ under Byzantine failures, and proved that $3f+2e-1$ processes are necessary and sufficient for a protocol to be fast according to a definition close to Lamport's. An interesting direction of future work is to combine their techniques with ours to see if this can lower the number of processes required in the Byzantine case.

## 2 Results

We assume a system $\Pi = \{p_1, \dots, p_n\}$ of $n \geq 3$ crash-prone processes communicating via reliable links. The system is partially synchronous [5]: after some global stabilization time (GST) messages take at most $\Delta$ units of time to reach their destinations. The bound $\Delta$ is known to the processes, while GST is unknown. We say that events that happen during the time interval $[0, \Delta)$ form *the first round*, events that happen during the time interval $[\Delta, 2\Delta)$ *the second round*, and so on.

In the *consensus* decision task, each process has an input value (a *proposal*) and may output a value (a *decision*). It is required that: every decision is the proposal of some process *(Validity)*; no two decisions are different *(Agreement)*; and every correct process eventually decides *(Termination)*. Consensus can alternatively be modeled as a shared object, where processes explicitly propose a value $v$ by calling `propose`($v$), which eventually returns the decision. This allows a process not to propose anything at all.

Definition 1. *A protocol is **f-resilient** if it achieves consensus under at most $f$ processes failures.*

While an $f$-resilient protocol can guarantee termination with up to $f$ process failures, these failures can nevertheless make the protocol run slow even when the system is synchronous. We thus also consider another threshold, $e$, which determines the maximal number of failures that a protocol can tolerate while still providing fast, two-step decisions in synchronous runs (we assume $e \leq f$). We next define this kind of runs formally.

Definition 2. *Given $E \subseteq \Pi$, a run is **E-faulty synchronous**, if:*
*(1) All processes in $\Pi \setminus E$ are correct, and all processes in $E$ are faulty.*
*(2) Processes in $E$ crash at the beginning of the first round.*
*(3) All messages sent during a round are delivered precisely at the beginning of the next round.*
*(4) All local computations are instantaneous.*

We next define protocols that can provide two-step decisions in synchronous runs with up to $e$ failures. We formulate our definitions for tasks and defer their analogs for objects to [12, §A].

Definition 3. *A run is **two-step** for a process $p$ if in this run $p$ decides a value by time $2\Delta$.*

Definition 4. *A protocol is **e-two-step** if for all $E \subseteq \Pi$ of size $e$:*
*(1) For every initial configuration $I$, there exists an $E$-faulty synchronous run starting from $I$ which is two-step for some process.*
*(2) For every initial configuration $I$ in which processes in $\Pi \setminus E$ propose the same value, for each process $p \in \Pi \setminus E$, there exists an $E$-faulty synchronous run starting from $I$ which is two-step for $p$.*

Differently from fast consensus protocols introduced by Lamport [9], we require only a single process to decide fast (item 1), unless all the proposals are the same (item 2). Paxos is not $e$-two-step for any $e > 0$, while Fast Paxos [8] is $e$-two-step if $n \geq \max\{2e + f + 1, 2f + 1\}$. We are interested in the following problem: *what is the minimal number of processes to solve $f$-resilient $e$-two-step consensus?* Surprisingly, this bound is lower than for Fast Paxos and furthermore depends on the problem definition.

Theorem 1. *An $f$-resilient $e$-two-step consensus task is implementable iff $n \geq \max\{2e + f, 2f + 1\}$.*

Theorem 2. *An $f$-resilient $e$-two-step consensus object is implementable iff $n \geq \max\{2e + f - 1, 2f + 1\}$.*

We defer the proofs of the lower bounds to [12, §B], and present matching upper bounds next.

## 3 Upper Bounds

Figure 1 presents a consensus protocol that matches our lower bounds. With the red lines ignored, the protocol implements a consensus task for $\max\{2e + f, 2f + 1\}$ processes (Theorem 1); with the red lines included, it implements a consensus object for $\max\{2e + f - 1, 2f + 1\}$ processes (Theorem 2). We first describe the task version, and then briefly highlight the differences for the object version. Our protocol improves the classical Fast Paxos [8] to reduce the number of processes required. It operates in a series of *ballots*, with each process storing the current ballot in a variable bal. The initial ballot 0 is called the *fast ballot*, while all others are *slow ballots*. During a ballot processes attempt to reach an agreement on one of the initial proposals by exchanging their votes, tracked in a variable val. Processes maintain a variable vbal to track the last ballot in which they cast a vote.

*Fast ballot.* Each process starts by sending its proposal (stored in a variable `initial_val`) to all other processes in a `Propose` message (line 4). A process accepts the message from a process $p$ only if it is in ballot 0, has not yet voted, and the value received is greater than or equal to its own proposal (line 6). In this case it updates val and replies to $p$ with a `2B` message, analogous to the eponymous message of Paxos. A process that gathers support from $n - e$ processes including itself (the first disjunct at line 10) considers its value decided (line 12) and communicates this to the other processes via a `Decide` message (line 13).

This flow guarantees that the protocol is $e$-two-step. Indeed, consider an initial configuration $I$, a set $E \subseteq \Pi$ of size $e$, and a process $p \notin E$ that proposes the highest value $v$ in $I$ among all processes in $\Pi \setminus E$. Then there exists an $E$-faulty synchronous run in which the `Propose` message sent by $p$ is the first one accepted by all other correct processes. Since $p$ does not accept any `Propose` message for value lower than $v$ (line 6), it will be able to collect the `2B` messages from $n - e - 1$ other processes. It will then satisfy the condition at line 10 and decide by $2\Delta$.

```
1  at startup / upon an invocation of propose($v$)
2      if val = $\bot$ then
3          initial_val $\leftarrow v$
4          send Propose(initial_val) to $\Pi \setminus \{p_i\}$
5  when received Propose($v$) from $q$
6      pre: bal = 0 $\wedge$ val = $\bot \wedge v \geq$ initial_val
         $\wedge$ (initial_val $\neq \bot \implies v$ = initial_val)
7      (val, proposer) $\leftarrow (v, q)$
8      send 2B$(0, v)$ to $q$
9  when received 2B(bal, $v$) from all $q \in P$
10     pre: (bal = 0 $\wedge |P \cup \{p_i\}| \geq n - e \wedge$ val $\in \{\bot, v\}) \vee$
              (bal $\neq 0 \wedge |P| \geq n - f$)
11     (val, decided) $\leftarrow (v, v)$
12     decide $v$
13     send Decide($v$) to $\Pi \setminus \{p_i\}$
14 when received Decide($v$)
15     (val, decided) $\leftarrow (v, v)$
16     decide $v$
17 when received 1A($b$) from $q$
18     pre: $b >$ bal
19     bal $\leftarrow b$
20     send 1B($b$, vbal, val, proposer, decided) to $q$
21 on timeout
22     let $b$ = (a ballot > bal such that $i \equiv b \pmod{n}$)
23     send 1A($b$) to $\Pi$
24 when received 1B($b, vbal_q, v_q, proposer_q, decided_q$) from all $q \in Q$
25     pre: $|Q| = n - f$
26     let $val = \bot$
27     let $b_{\max} = \max\{vbal_q \mid q \in Q\}$
28     let $R = \{q \in Q \mid proposer_q \notin Q\}$
29     if $\exists q \in Q.\ decided_q \neq \bot$ then
30         $val \leftarrow decided_q$
31     else if $b_{\max} > 0$ then
32         $val \leftarrow v_q$ such that $vbal_q = b_{\max}$
33     else if $\exists v \neq \bot.\ \exists S \subseteq R.\ |S| > n - f - e \wedge \forall q \in Q_f.\ v_q = v$ then
34         $val \leftarrow v$
35     else if $\exists v \neq \bot.\ \exists S \subseteq R.\ |S| = n - f - e \wedge \forall q \in Q_f.\ v_q = v$ then
36         $val \leftarrow$ the maximal value $v$ satisfying the condition at line 35
37     else if initial_val $\neq \bot$ then
38         $val \leftarrow$ initial_val
39     if $val \neq \bot$ then send 2A($b, val$) to $\Pi$
40 when received 2A($b, v$) from $q$
41     pre: bal $\leq b$
42     (val, bal, vbal) $\leftarrow (v, b, b)$
43     send 2B($b, v$) to $q$
```

**Figure 1: Consensus task at a process $p_i$. Red lines highlight the changes needed to implement a consensus object.**

*Slow ballots.* If processes do not reach an agreement at ballot 0 within $2\Delta$, the protocol nominates a process $p_i$ to initiate a new slow ballot (line 22); this nomination is done using standard techniques [12, §C]. Process $p_i$ broadcasts a 1A message asking the others to join the new ballot (line 23), and processes respond with a 1B message, carrying information about their state (line 17). Once $p_i$ receives $n - f$ replies from a set $Q$ (line 24), it computes a proposal for its ballot as we describe in the following. It then sends this proposal in a 2A message (line 39), to which processes reply with 2B messages (line 43). Process $p_i$ decides after collecting $n-f$ matching votes (the second disjunct at line 10).

*Computing the proposal.* Process $p_i$ computes its proposal based on the states received in the 1B messages. If some process has already decided a value, then $p_i$ selects that value (line 29). Otherwise, $p_i$ considers the highest ballot $b_{\max}$ where a vote was cast (line 27): the votes in $b_{\max}$ supersede those in lower ballots. If $b_{\max} > 0$, then $p_i$ selects the associated value, just like in the usual Paxos (line 31).

If $b_{\max} = 0$, then some value may have been decided on the fast path. To handle this case, process $p_i$ first excludes the values whose proposers belong to $Q$ (line 28): these proposers have not taken the fast path, because line 29 did not execute; they will not take it in the future either, because they have moved to a slow ballot when replying to $p_i$ (line 19). If after this there is a value $v$ with more than $n - f - e$ votes, then $p_i$ proposes $v$ (line 33); we prove below that $v$ is unique. If no such $v$ exists, $p_i$ considers the values with exactly $n - f - e$ votes (line 35). There may be multiple such values, so $p_i$ selects the greatest one (line 36). Finally, in all other cases $p_i$ selects its own initial value (line 38). The following lemma shows that this algorithm correctly recovers fast path decisions.

Lemma 1. *Assume $n \geq 2e + f$. If a value $v$ is decided via the fast path at ballot 0, then lines 33–38 always select $v$ as val.*

Proof. Since $v$ takes the fast path, there are at least $n-e$ processes that vote for $v$ at ballot 0 (implicitly including the proposer of $v$). The set of these processes intersects with $Q$ (line 24) in $\geq n - e - f$ processes, thus enabling the condition at either line 33 or line 35.

Consider first the case of line 33, so that $> n - e - f$ processes in $Q$ voted for $v$. Then the number of processes in $Q$ that could have voted for another value is $< (n - f) - (n - e - f) = e = (2e + f) - e - f \leq n - e - f$. Hence, $v$ is the only value that can satisfy the condition at line 33, and *val* must be assigned to $v$ at line 34.

Assume now that the condition at line 35 holds, so that $n - e - f$ processes in $Q$ voted for $v$. Since $v$ went onto the fast path, $\geq n - e$ processes voted for $v$ overall. Thus, all $f$ processes outside $Q$ must have voted for $v$. Since the proposer of any other value $v'$ that also satisfies the condition at line 35 does not belong to $Q$ (line 28), it must have voted for $v$. But then $v' < v$ by line 6, which proves that *val* is assigned to $v$ at line 36. □

*Consensus object.* When implementing a consensus object, the variable initial_val (initially $\bot$, lower than any other value) is explicitly updated upon an invocation of propose (line 3), provided the process has not yet voted for someone else's proposal. The only other difference is the additional condition at line 6: a process responds to the Propose($v$) message only if it has not proposed yet (initial_val $= \bot$), or if $v$ matches its own proposal (initial_val $= v$).

In [12, §C] we prove that the protocols we have just presented indeed satisfy the conditions of Theorems 1 and 2.

## Acknowledgments

This work was supported by projects BYZANTIUM, DECO and PRODIGY funded by MCIN/AEI, and REDONDA funded by CHIST-ERA. We thank Petr Kuznetsov for helpful discussions. We also thank Benedict Elliot Smith, whose work on fast consensus in Cassandra served as a practical motivation for this paper.

## References

[1] Armando Castañeda, Sergio Rajsbaum, and Michel Raynal. 2018. Unifying Concurrent Objects and Distributed Tasks: Interval-Linearizability. *J. ACM* 65, 6 (2018).

[2] Tushar Deepak Chandra and Sam Toueg. 1996. Unreliable Failure Detectors for Reliable Distributed Systems. *J. ACM* 43, 2 (1996).

[3] Bernadette Charron-Bost and André Schiper. 2004. Uniform consensus is harder than consensus. *J. Algorithms* 51, 1 (2004).

[4] Partha Dutta and Rachid Guerraoui. 2002. Fast Indulgent Consensus with Zero Degradation. In *European Dependable Computing Conference (EDCC)*.

[5] Cynthia Dwork, Nancy A. Lynch, and Larry J. Stockmeyer. 1988. Consensus in the Presence of Partial Synchrony. *J. ACM* 35, 2 (1988).

[6] Tuanir França Rezende and Pierre Sutra. 2020. Leaderless State-Machine Replication: Specification, Properties, Limits. In *International Symposium on Distributed Computing (DISC)*.

[7] Petr Kuznetsov, Andrei Tonkikh, and Yan X Zhang. 2021. Revisiting Optimal Resilience of Fast Byzantine Consensus. In *Symposium on Principles of Distributed Computing (PODC)*.

[8] Leslie Lamport. 2006. Fast Paxos. *Distributed Computing* 19 (2006).

[9] Leslie Lamport. 2006. Lower Bounds for Asynchronous Consensus. *Distributed Computing* 19 (2006).

[10] Iulian Moraru, David G. Andersen, and Michael Kaminsky. 2013. There Is More Consensus in Egalitarian Parliaments. In *Symposium on Operating Systems Principles (SOSP)*.

[11] Fernando Pedone and André Schiper. 1999. Generic Broadcast. In *International Symposium on Distributed Computing (DISC)*.

[12] Fedor Ryabinin, Alexey Gotsman, and Pierre Sutra. 2025. Revisiting Lower Bounds for Two-Step Consensus. arXiv:2505.03627 [cs.DC] http://arxiv.org/abs/2505.03627

[13] Fred B. Schneider. 1990. Implementing Fault-Tolerant Services Using the State Machine Approach: A Tutorial. *Comput. Surveys* 22 (1990).