

MIT Open Access Articles

*On the optimal space complexity of
consensus for anonymous processes*

The MIT Faculty has made this article openly available. **Please share**
how this access benefits you. Your story matters.

As Published: <https://doi.org/10.1007/s00446-018-0331-9>

Publisher: Springer Berlin Heidelberg

Persistent URL: <https://hdl.handle.net/1721.1/131297>

Version: Author's final manuscript: final author's manuscript post peer review, without publisher's formatting or copy editing

Terms of Use: Article is made available in accordance with the publisher's policy and may be subject to US copyright law. Please refer to the publisher's site for terms of use.



On the Optimal Space Complexity of Consensus for Anonymous Processes

Rati Gelashvili

the date of receipt and acceptance should be inserted later

Abstract The optimal space complexity of consensus in asynchronous shared memory was an open problem for two decades. For a system of n processes, no algorithm using a sublinear number of registers is known. Up until very recently, the best known lower bound due to Fich, Herlihy, and Shavit was $\Omega(\sqrt{n})$ registers.

Fich, Herlihy, and Shavit first proved their lower bound for the special case of the problem where processes are anonymous (i.e. they run the same algorithm) and then extended it to the general case.

In this paper we close the gap for the anonymous case of the problem. We show that any consensus algorithm from read-write registers for anonymous processes that satisfies nondeterministic solo termination has to use $\Omega(n)$ registers in some execution. This implies an $\Omega(n)$ lower bound on the space complexity of deterministic obstruction-free and randomized wait-free consensus, matching the upper bound.

We introduce new techniques for marshalling anonymous processes and their executions, in particular, the concepts of *leader-follower pairs* and *reserving executions*, that play a critical role in the lower bound argument and will hopefully be more generally applicable.

Keywords Consensus · Anonymous Processes · Space Complexity · Registers

Support is gratefully acknowledged from the National Science Foundation under grants CCF-1217921, CCF-1301926, and IIS-1447786, the Department of Energy under grant ER26116/DE-SC0008923, and the Oracle and Intel corporations.

Rati Gelashvili
MIT Computer Science and Artificial Intelligence Laboratory
32 Vassar Street, 32-G630, Cambridge, MA 02139, USA
Tel.: +1-617-715-2459
E-mail: gelash@mit.edu

1 Introduction

The celebrated Fischer, Lynch and Paterson (FLP) [9] result proved that fundamental synchronization tasks including consensus and test-and-set are not solvable in a wait-free manner using read-write registers. However, the work of Ben-Or [4] shows that it is possible to circumvent FLP and obtain efficient distributed algorithms, if we allow probabilistic termination. It is also possible to solve these tasks deterministically, but in an obstruction-free manner; it is known how to convert any deterministic obstruction-free algorithm into a randomized wait-free algorithm against an oblivious adversary. (See [10].)

In this paper, we consider algorithms in which processes communicate using only read-write registers. The space complexity of such an algorithm is defined as the maximum number of registers used in any execution. A lot of research has been dedicated to improving space complexity upper and lower bounds for canonical tasks. For test-and-set, an $\Omega(\log n)$ lower bound was shown in [14]. On the other hand, an $O(\sqrt{n})$ deterministic obstruction-free upper bound was given in [10]. The final breakthrough was the recent obstruction-free algorithm designed by Giakkoupis et al. [11], with $O(\log n)$ space complexity.

For consensus, randomized wait-free algorithms that work against a strong adversary and use $O(n)$ read-write registers are long known [1, 2, 13]. Algorithms that solve consensus in a deterministic obstruction-free manner using $O(n)$ registers are also known [5, 6, 12, 15]. A lower bound of $\Omega(\sqrt{n})$ by Fich et al. [8] first appeared in 1993. The proof is notorious for its technicality and utilizes a neat inductive combination of covering and valency arguments. Another version of the proof appeared in a textbook [3]. The authors of [8] conjectured a tight

lower bound of $\Omega(n)$, but such a bound or a sublinear space algorithm remained elusive up until very recently, when Zhu showed a lower bound of $n - 1$ registers [16]. This has been improved to n in [7].

For the intervening two decades, however, the linear lower bound had not been proven even in the restricted case, in a system where all processes are anonymous [8]. In such a system, processes have no identifiers and can be thought of as running the same code: all processes with the same input start in the same initial state, behave identically and remain in the same states until they read different values or observe different coin flip outcomes. A more precise definition is given later.

The authors of [8] introduced the notion of *clones* of processes, which has since become a standard tool for proving anonymous space lower bounds. They first showed the $\Omega(\sqrt{n})$ lower bound for anonymous processes, and then extended it to a much more complex argument for the general case. These lower bounds hold for consensus algorithms satisfying the *nondeterministic solo termination* property, which was also introduced in [8]. Any lower bound for algorithms satisfying the nondeterministic solo termination implies a lower bound for deterministic obstruction-free and randomized wait-free algorithms.

The linear upper bound holds for anonymous processes, as a number of deterministic obstruction-free consensus algorithms that use $O(n)$ registers are anonymous [5, 12, 15]. In the algorithms of [15], the processes are memoryless (do not use local memory), in addition to being anonymous, and exactly n registers are used. In this further restricted model, [15] also showed that n registers are actually necessary.

Contribution A conference version of this paper preceded the result of [16] and showed an $\Omega(n)$ lower bound in the anonymous case for consensus algorithms that satisfy nondeterministic solo termination. As in [3, 8], the bound is for the worst-case space complexity of the algorithm, i.e. for the number of registers used in some execution, regardless of its actual probability.

Our argument relies heavily on the anonymity of the processes, and introduces specific techniques that we hope will be useful for future work. We design a class of executions, which we call *reserving*, and define a notion of valency which corresponds to the possible return values for these executions. We also extend the role of the *clones* of processes, by considering pairs of processes that can be split and reunited. This enables proving larger lower bounds by reusing the clones.

2 Definitions and Notation

We consider the standard asynchronous shared-memory model with anonymous processes and atomic read-write registers. Processes take steps, where each step is either a shared-memory step or an internal step. There are no guarantees on when a process takes its next step. In fact, any process is allowed stop taking steps altogether.

A shared-memory step of a process is either a read or a write to some register. With an internal step, a process can make local nondeterministic choices, or return a value, after which the process takes no more steps. Naturally, the outcomes of nondeterministic choices influence the state of the process and its next steps.

A *configuration* is a collection of all process states and the contents of all registers, and describes the global state of the system. An *execution* is a sequence of steps by processes and a *solo execution* is an execution where all steps are taken by a single process. An execution starts in a configuration and leads to a configuration. We will use capital latin letters to denote configurations and lower case greek letters to denote executions, and we will refer to the configuration reached by an execution α that started in configuration C as the configuration $C\alpha$. Finally, an execution $\alpha\beta$ simply stands for the execution α followed by the execution β .

Notice that, if a process p in state s makes a certain nondeterministic choice and ends up in state s' , then, at any time, any process q that is in the same state s might make the same nondeterministic choice and also end up in state s' . In this work, we will restrict our attention to executions where all processes in the same state make the same nondeterministic choices.

We will also only consider executions where any process, after a shared-memory step, immediately performs all subsequent internal steps, leading to a shared-memory step, unless it returns a value. Therefore, from now on, the term *step* will refer exclusively to a shared-memory step, and we will consider only process states in which the next step is a shared-memory step.

In a system of anonymous processes, all processes with the same input start in the same state. If in some configuration, a process p in state s writes v to some register r and changes state to s' , then in any configuration, any process q in the same state s will also write v to register r with its next step and change state to s' . If in some configuration, a process p in state s reads v from register r and changes state to s' , then in any configuration, any process q in state s will also read from the register r with its next step. Notice that the reads by p and q are in different configurations and might return different results. However, if q happens to read the value v , then it will also change its state to s' . The

above statements are true since, by our assumption, p and q make the same nondeterministic choices.

We say that a process p is *covering* a register r , if the next step of p is a write to r . A *block write* of a set of processes P to a set of covered registers R is a sequence of write steps by processes in P , where each step writes to a different register in R and all registers in R get written to.

A *clone* of a process p , exactly as in [3, 8], is defined as another process with the same input as p , that shadows p by performing the same steps as p in lockstep, reading and writing the same values immediately after p , and remaining in the same state, until just before some write of p . Because the system consists of anonymous processes, in any execution with sufficiently many processes, for any write step of p , there always exists an alternative execution with a clone q that shadowed p until just before the write. In particular, in the alternative execution, process q covers the register and is about to write the value that p last wrote there. Moreover, the two executions with or without the clone covering the register are completely indistinguishable to all processes other than the clone itself.

In the binary consensus problem, each process starts with a binary input, 0 or 1, and is supposed to return a binary output. The correctness requirement is that all outputs must be the same and equal to the input of some process. We say that an execution *decides* 0 (or 1) if some process returns 0 (or 1, respectively) during this execution.

The wait-free termination requirement means that each process is supposed to eventually return an output within a finite number of its own steps, regardless of how the other processes are scheduled. The obstruction-free termination requirement means that any process that runs solo is supposed to eventually return an output within a finite number of steps, starting from each reachable configuration. In randomized algorithms, processes are allowed to flip random coins and decide their next steps accordingly. In this setting, the randomized wait-free termination requirement means that each process is supposed to eventually return an output within a finite number of its own steps with probability 1.

The FLP result shows that, in the asynchronous shared memory model with read-write registers, no deterministic algorithm can solve binary consensus in a wait-free way. However, it is possible to solve consensus both in a deterministic obstruction-free and randomized wait-free way. The *nondeterministic solo termination* property means that from each reachable configuration, for each process, there exists a finite solo execution by the process where it terminates and returns an output. We prove our lower bounds for binary

consensus algorithms that satisfy the nondeterministic solo termination property, because both deterministic obstruction-free algorithms and randomized wait-free algorithms fall into this category.

The following two standard lemmas are key ingredients in our lower bound arguments.

Lemma 1 (Indistinguishability Lemma) *If a process p has the same state in two configurations C and D , and the contents of all registers are the same in both of these configurations, then every solo execution by p starting from C can have the same results starting from D .*

Lemma 2 *Let C be a reachable configuration by a consensus algorithm, such that some execution α returns 0 starting from C and some execution β returns 1 starting from C . Then, no process has returned in C .*

Proof Suppose for contradiction that some process p has already returned a value $v \in \{0, 1\}$ in C . If $v = 0$, consider the execution β starting from C that returns 1. If $v = 1$, consider the execution α starting from C that returns 0. In the resulting execution from the initial configuration both values are returned, violating consensus as the returned values must all be the same.

3 A Square-Root Lower Bound

To demonstrate our approach, we start by presenting a proof of the $\Omega(\sqrt{n})$ space lower bound in the anonymous setting. This is the same as the best known lower bound from [8], but the inductive argument and the valency definition used are considerably different.

If there is a solo execution of some process that returns $v \in \{0, 1\}$ starting from a configuration, then we call this configuration *v -solo-deciding*. Nondeterministic solo termination implies that every configuration is 0-solo-deciding or 1-solo-deciding. Note that the same configuration can be simultaneously 0-solo-deciding and 1-solo-deciding. We call such configurations *solo-bivalent*. Configurations which are not solo-bivalent are called *solo-univalent*. If a configuration is 0-solo-deciding, but not 1-solo-deciding (i.e. no solo execution from this configuration decides 1), then we call it *0-solo-valent*, meaning that the configuration is solo-univalent with valency 0. Analogously, a configuration is *1-solo-valent* if it is 1-solo-deciding but not 0-solo-deciding.

Lemma 3 *Consider a system of at least two processes. Then, in every solo-bivalent configuration, we can always find two distinct processes p and q , such that, from this configuration, there is a solo execution of p returning 0 and a solo execution of q returning 1.*

Proof Either the configuration is solo-bivalent because of solo executions of distinct processes, in which case we are done, or two solo executions of some process p return different values. In this case it suffices to consider any terminating solo execution of another process q .

Lemma 4 *Consider a system of $\frac{(r-1)r}{2} + 2$ anonymous processes for any $r \geq 0$. Then, for any consensus algorithm that uses only atomic read-write registers and satisfies the nondeterministic solo termination property, there exists a configuration C_r reachable by an execution ρ_r with the following properties:*

- *There is a set R of r registers, each of which has been written to during ρ_r , and*
- *the configuration C_r is solo-bivalent.*

Proof The proof is by induction, with the base case $r = 0$. Our system consists of two processes p and q , p starts with input 0, q starts with input 1, and C_0 is the initial configuration. The solo-bivalency of C_0 follows from Lemma 1 as process p cannot distinguish between C_0 and a configuration where both processes start with input 0. By nondeterministic solo termination, p has a terminating solo execution starting from this configuration. This execution is required to return 0. Thus, some solo execution of process p starting from C_0 returns 0. Analogously, some solo execution of process q starting from C_0 returns 1.

Let us assume the induction hypothesis for some r and prove it for $r + 1$. We can reach a solo-bivalent configuration C_r using $\frac{(r-1)r}{2} + 2$ processes by an execution ρ_r that writes to a set R of r registers. The goal is to use another set of r processes in order to write to a new register and extend C_r to C_{r+1} .

From configuration C_r , by Lemma 3, there exists a solo execution α by process p that returns 0 and a solo execution β of process q that returns 1. For each register in R , let us add a new clone of the process that last wrote to it. Let ρ'_r be the same execution as ρ_r , except with r new clones, and let C'_r be the configuration reached by ρ'_r . Configuration C'_r is the same as C_r , except for the new clones. In C'_r , each register in R is covered by a clone, poised to write the same value as present in the register in configuration C_r .

Let us now apply a covering argument utilizing the clones. We know that process p returns 0 after $\rho_r \alpha$. During its solo execution α , process p has to write to a register outside of R . Otherwise, configurations C_r and $C'_r \alpha \gamma$ are indistinguishable to process q . This is because the values in all registers are the same, and q is still in the same state as in C_r . Hence, by Lemma 1, q can return 1 during $\rho'_r \alpha \gamma \beta$ as it would during $\rho_r \beta$, contradicting the correctness of the consensus algorithm. Analogously, process q has to write outside of R during

β . Let $\alpha = \alpha' w_p \alpha''$, where w_p is the first write of p outside the set of registers R , and let $\beta = \beta' w_q \beta''$, with w_q being the first write outside of R . Let ℓ be the length of $\gamma \beta' w_q$ and, for $0 \leq i \leq \ell$, let π_i be the length i prefix of $\gamma \beta' w_q$.

Next, we use a valency argument to reach C_{r+1} . We show that either the configuration $\rho'_r \alpha' \gamma \beta' w_q$ or $\rho'_r \alpha' \pi_i w_p$, for some $i \in \{1, \dots, \ell\}$ satisfies the properties necessary to be ρ_{r+1} . The number of processes used in ρ_{r+1} is $\frac{(r-1)r}{2} + 2$ from ρ_{r+1} , plus the r clones that perform γ , which gives $\frac{r(r+1)}{2} + 2$ as required. Moreover, we can find $r + 1$ registers that have been written to during ρ_{r+1} : the r registers in R and one more register written to by either w_p or w_q . Thus, we only need to show that one of the configurations $C'_r \alpha' \gamma \beta' w_q$, $C'_r \alpha' \pi_0 w_p, \dots, C'_r \alpha' \pi_\ell w_p$ is solo-bivalent.

Assume the contrary. Then the configuration after $\rho'_r \alpha' \pi_0 w_p$ is solo-univalent. Moreover, since C_r is the configuration reached by ρ_r , π_0 is the empty execution, and p returns 0 in $\rho_r \alpha' w_p \alpha''$, the configuration $C'_r \alpha' \pi_0 w_p$ is actually 0-solo-valent. On the other hand, the configuration reached by $\rho'_r \alpha' \gamma \beta' w_q = \rho'_r \alpha' \pi_\ell$ must be 1-solo-valent. It is solo-univalent by the contradiction assumption and 1-solo-deciding as q cannot distinguish between configurations C_r and $C'_r \alpha' \gamma$, and thus by Lemma 1, the solo execution β'' of q can return 1 starting from $C'_r \alpha' \gamma \beta' w_q$ as it can from $C_r \beta' w_q$.

Because the configuration $C'_r \alpha' \pi_\ell$ is 1-solo-valent, any terminating solo execution of process p from that configuration must also return 1. In particular, every terminating solo execution that starts by p performing its next step w_p returns 1. So the configuration $C'_r \alpha' \pi_\ell w_p$ must be 1-solo-valent: a terminating solo execution of p returns 1 and it is solo-univalent by the contradiction assumption for $i = \ell$. Therefore, configuration $C'_r \alpha' \pi_i w_p$ is 0-solo-valent for $i = 0$ and 1-solo-valent for $i = \ell$. Hence, there exists an i , such that $X = C'_r \alpha' \pi_i w_p$ is 0-solo-valent, and $Y = C'_r \alpha' \pi_{i+1} w_p$ is 1-solo-valent. Let o be the last step in π_{i+1} .

o is either performed by process q or by the new clones as a part of block write γ . It cannot be a read or a write to the same register as w_p writes to, since configurations X and Y would be indistinguishable to p . Hence, by Lemma 1, p could return the same output from both configurations. This would contradict the different solo-univalencies of X and Y . Therefore, steps w_p and o commute. Let σ be a terminating solo execution from Y by the process f that performed o . Since Y is 1-solo-valent, f returns 1 after σ . Since w_p and o commute, the configurations Xo and Y are indistinguishable to f . By Lemma 1, f returns 1 after the solo execution σ from configuration Xo . However, $o\sigma$ is also a solo execution by f from X that returns

1, contradicting the 0-valency of X . The contradiction proves the induction step, completing our argument.

Notice that for n processes, Lemma 4 directly implies the existence of an execution where $\Omega(\sqrt{n})$ registers are written to, proving the desired lower bound.

4 Linear Lower Bound

Consider a system with n anonymous processes and an arbitrary correct consensus algorithm satisfying the nondeterministic solo termination property. We will assume that no execution of the algorithm uses $\lfloor n/14 - 1 \rfloor$ registers and derive a contradiction. For notational convenience, let us define m to be $\lfloor n/14 \rfloor - 2$.

The argument in Lemma 4 in the previous section relies on a new set of clones in each iteration to overwrite the changes to the contents of the registers made during the inductive step. This is the primary reason why we only get an $\Omega(\sqrt{n})$ lower bound. As the authors of [8] also mention, to get a stronger lower bound we would instead have to reuse processes to overwrite the registers. However, after the overwriting, we cannot guarantee that processes would still cover various registers. Moreover, it is insufficient to simply cover registers with processes without any knowledge of what they are about to write. We start by introducing concepts that will be used to overcome these challenges.

4.1 Process Pairs: The Leader and The Follower

To prove Lemma 4, we used clones that covered registers to overwrite these registers using a block-write, to reset their contents to be the same as in a previous configuration with known valency. In order to do something similar, but without using new clones, we will consider pairs of processes, consisting of a leader process and a follower process. The follower is a clone of the leader process and the pair remains in the same states and performs the same steps during the whole execution. Every process in the system will be either a leader or a follower in some pair.

Usually, when we schedule a leader to perform a step, its follower performs the same step immediately after the leader. In this case, we say that *the pair performed the step*. However, sometimes we will *split* the pair by having only the leader perform a write step and let the follower cover the register. We will explicitly say when this is the case. After we split the pair in such a way, we will delay scheduling the leader and the follower will remain poised to write to the covered register. Later, we will schedule the follower to write, effectively

resetting the register to the value it had after the write by the leader. As the leader did not take any steps in the meanwhile, after the write the follower will again be in the same state as its leader. Hence, the pair of the leader and the follower will no longer be split, and will continue taking steps in lock-step as a pair.

This is very different from the way clones were used in the proof of Lemma 4, because after the pair of the leader and its follower is united, it can be split again. Therefore, the same follower can reset the contents of registers written by its leader multiple times.

We call a split pair of a leader and a follower *fresh* as long as the register that the leader wrote to, and its follower is covering, has not been overwritten. After the register is overwritten, we call the split pair *stale*. In any configuration, there is at most one fresh split pair whose follower covers a particular register.

In addition, we also use cloning in a way similar to the proof of Lemma 4, except that we do this only a constant number of times, as opposed to r times, to reach the next configuration C_{r+1} . Moreover, each time we do this, we actually clone a pair, i.e. we create duplicates of both a leader and its follower. The new leader-follower pair is in the same state as the original pair, and from there on behaves as any other pair.

By definition, both the leader and the follower in unsplit pairs are in the same state. Therefore, we say that an unsplit pair covers a register when both the leader and the follower cover it and we say that an unsplit pair returns when both the leader and the follower return in two successive steps. A solo execution by an unsplit pair p is an execution containing an even number of steps, where the leader in pair p takes a step, immediately followed by exactly the same step of the follower in p . Thus, nondeterministic solo termination for the leader process implies nondeterministic solo termination for the executions of the unsplit pair.

4.2 Reserving Executions

Intuitively, *reserving executions* ensure that, for each register written to during an execution, some pair is reserved to cover it. We can use these pairs for covering in subsequent inductive configurations.

Definition 1 Let C be some configuration reachable by the algorithm, and let P be a set of at least $m + 1$ unsplit pairs. We call an execution γ that starts from configuration C *reserving* from C by P if:

- γ is a sequence of steps by pairs in P (first by the leader of the pair, immediately followed by the follower).

- For each prefix γ' of γ and for each register written to during γ' , there is an unsplit pair $p \in P$ that covers it in configuration $C\gamma'$.
- If a pair $p \in P$ returns during γ , then this happens during the last two steps of γ .

Notice that any reserving execution contains an even number of steps and any even-length prefix of a reserving execution is also a reserving execution. Let $\text{Res}(C, P)$ be the set of all reserving executions from C by P that end with a pair $p \in P$ returning. We first show that, given sufficiently many pairs, such an execution exists. This will be essential for defining valency based on reserving executions. Recall that we assumed a strict upper bound of m on the number of registers that can ever be written.

Lemma 5 *For any reachable configuration C and any set P of at least $m + 1$ unsplit pairs that have not returned, $\text{Res}(C, P) \neq \emptyset$.*

Proof For a given C and P , we will prove the lemma by constructing a particular reserving execution γ that ends when some pair $p \in P$ returns. We start with an empty γ and repeatedly extend it.

In the first stage, consider each pair $p \in P$, one at a time. By nondeterministic solo termination, there exists a solo execution by pair p where p returns. If this solo execution contains write steps, extend γ by the prefix of the execution before the first write, and consider the next pair in P . Otherwise, complete γ by extending it with the whole solo execution of p .

The first stage consists of extending the execution at most $|P|$ times. Each time, we extend γ by a prefix of a finite solo execution of some pair $p \in P$. These steps are reads by leaders and followers of pairs in P , and therefore the constructed prefix of γ is reserving.

If some pair returns in the first stage, the construction of γ is complete. Otherwise, each pair in P is covering a register at the end of the first stage and we move on to the second stage.

In the second stage, the execution γ is extended by repeatedly doing the following two phases. We maintain the invariant that, at the beginning of the first phase each of the at least $m + 1$ pairs in P is covering a register. Let R be the set of registers covered by pairs in P . Since $|R| \leq m < |P|$, we can find two pairs $p, q \in P$ covering the same register in R . By nondeterministic solo termination, there exists a solo execution of pair p where p returns. If this solo execution contains a write to a register outside of R , extend γ by the prefix of the execution before this write. Note that p still covers a register, satisfying the invariant. Add this register to R . Then continue performing the first phase. Oth-

erwise, complete γ by extending it with the whole solo execution of pair p .

In the second stage, each iteration extends γ by a finite number of steps. After each iteration, if the construction is not complete, the size of R increases by one. Because only m registers can ever be written, R always has size at most m . Thus, after at most m finite extensions, we will complete the construction of γ when some pair returns.

To see that γ is reserving, notice that all registers that were written to are in R . By construction, each register in R remains covered from the beginning of the second stage or when it is first added to R during the second stage.

The next lemma follows immediately from the definition of reserving executions.

Lemma 6 *Consider a reachable configuration C , a set of at least $m + 1$ unsplit pairs P' none of which have returned in C , and some unsplit pair $p \notin P'$ poised to perform a write w_p in C . Let C' be the configuration reached from C after the pair p performs w_p (first the leader, then the follower). Moreover, assume that another unsplit pair $q \neq p$ with $q \notin P'$ is covering the same register that w_p writes to. If $\gamma \in \text{Res}(C', P')$, then $w_p w_p \gamma$ is in $\text{Res}(C, P' \cup \{p\} \cup \{q\})$.*

4.3 New Definition of Valency

Here we define valency based on reserving executions. For a set of process pairs U , we say that a configuration C is *0-reserving-deciding $_U$* , if there exists a subset of $m + 1$ unsplit pairs $P \subseteq U$, and a reserving execution in $\text{Res}(C, P)$ returning 0. We define *1-reserving-deciding $_U$* analogously. The next result immediately follows using Lemma 5.

Lemma 7 *Let U contain at least $m + 1$ unsplit pairs that have not returned in configuration C . Then C is *v-reserving-deciding $_U$* for at least one $v \in \{0, 1\}$.*

A configuration that is both 0-reserving-deciding $_U$ and 1-reserving-deciding $_U$ is called *reserving-bivalent $_U$* . Otherwise, the configuration is called *reserving-univalent $_U$* .

If a configuration is 0-reserving-deciding $_U$, but not 1-reserving-deciding $_U$ (i.e. no reserving execution by $m + 1$ unsplit pairs in U starting from this configuration decides 1), then we call it *0-reserving-valent $_U$* . Analogously, a configuration is called *1-reserving-valent $_U$* if it is 1-reserving-deciding $_U$, but not 0-reserving-deciding $_U$.

The next lemma says that, from a reserving-bivalent configuration, there are reserving executions by disjoint sets of processes that decide different values.

Lemma 8 *Let C be a reserving-bivalent _{U} configuration for U a set of at least $3m + 2$ unsplit pairs. Then there are disjoint sets of $m + 1$ unsplit pairs $P' \subseteq U$ and $Q' \subseteq U$, such that an execution in $\text{Res}(C, P')$ returns 0 and an execution in $\text{Res}(C, Q')$ returns 1.*

Proof None of the pairs in U have already returned in configuration C , as that would contradict the existence of a reserving execution returning the other output. As C is reserving-bivalent _{U} , there exist sets of $m + 1$ unsplit pairs P and Q , such that an execution in $\text{Res}(C, P)$ returns 0 and an execution in $\text{Res}(C, Q)$ returns 1. If P and Q do not intersect then we are done by setting $P' = P$ and $Q' = Q$.

Otherwise, consider an arbitrary set $H \subseteq U - P - Q$ of $m + 1$ unsplit pairs. If C is 0-reserving-deciding _{H} , we set $P' = H$ and $Q' = Q$, and if C is 1-reserving-deciding _{H} , then we set $P' = P$ and $Q' = H$. One of these cases holds due to Lemma 7, completing the proof.

4.4 The Main Proof

Consider any correct consensus algorithm satisfying non-deterministic solo termination in a system of anonymous processes, with the property that every execution uses at most m registers. We will restrict attention to executions in which the processes are partitioned into leader-follower pairs. Let $0 \leq r \leq m$. Suppose there exists a set U containing $5m + 6 + 2r$ leader-follower pairs and a configuration C_r that is reachable by an execution by pairs in U .

Consider configuration C_r . Let R_s denote the set of registers that are covered by the follower of a fresh split pair. Suppose that there are no stale split pairs. Suppose there is a set R_c of $r - |R_s|$ other registers that are each covered by at least one unsplit pair. Let V consist of r leader-follower pairs covering the r registers in $R_c \cup R_s$. In particular, all split pairs are in V . Finally, suppose that there are two disjoint sets of unsplit pairs $P, Q \subseteq U - V$, such that some execution $\alpha \in \text{Res}(C_r, P)$ returns 0, some execution $\beta \in \text{Res}(C_r, Q)$ returns 1, and $|P| + |Q| \leq 2m + 4$. Recall that by definition of reserving executions, in both α and β , all steps are taken as pairs - first the leader, then its follower, and the last two steps are some leader-follower returning the output. Notice that, by the existence of α and β , Lemma 2 implies that no process in U has returned in C_r .

Let C_0 be an initial configuration that contains a set U of $5m + 6$ leader-follower pairs, half of which have input 0 and half of which have input 1. Let R_s, R_c , and V be empty. Let P be a set of $m + 1$ pairs in U with input 0 and let Q be a set of $m + 1$ pairs in U

with input 1. There are no split pairs in any initial configuration. By Lemma 5, $\text{Res}(C_0, P), \text{Res}(C_0, Q) \neq \emptyset$. Since all steps of executions in $\text{Res}(C_0, P)$ are by processes with input 0, all executions in $\text{Res}(C_0, P)$ return 0. Similarly, all executions in $\text{Res}(C_0, Q)$ return 1. Thus, it is possible to satisfy all the assumptions when $r = 0$.

We will construct a set U' of $5m + 6 + 2(r + 1)$ leader-follower pairs, a configuration C_{r+1} that is reachable by an execution by pairs in U' , disjoint sets of registers R'_s and R'_c and disjoint sets of leader-follower pairs $V', P', Q' \subseteq U'$ such that, in C_{r+1} ,

1. R'_s is the set of registers that are covered by the follower of a fresh split pair (all of which are in V'),
2. one unsplit pair in V' covers each register in R'_c ,
3. $|V'| = |R'_s| + |R'_c| = r + 1$,
4. some execution in $\text{Res}(C_{r+1}, P')$ returns 0,
5. some execution in $\text{Res}(C_{r+1}, Q')$ returns 1,
6. $|P'| + |Q'| \leq 2m + 4$, and
7. there are no stale split pairs.

The fourth and fifth properties and Lemma 2 implies that no process in U' has returned in C_{r+1} .

We will construct an execution that starts in C_r and reaches C_{r+1} . In $U' - U$ we have two more pairs available that have not taken steps and can be used to clone a leader-follower pair. Let T denote $U - V - P - Q$. Since $|V| = r$ and $|P| + |Q| \leq 2m + 4$, we have $|T| \geq 3m + 2 + r \geq 3m + 2$.

In C_r , each register in R_c is covered by some pair (both a leader and its follower) in V . Let γ_c be a block write to all registers in R_c by only the leaders but not the followers of the respective covering pairs, i.e. after each write we get a new fresh split pair. T contains at least $3m + 2 \geq m + 1$ pairs that have not returned in $C_r \gamma_c$, as no pair in T has returned in C_r or taken a step in γ_c . Thus, by Lemma 7, $C_r \gamma_c$ is v -reserving-deciding _{T} for at least one $v \in \{0, 1\}$. Without loss of generality assume that $C_r \gamma_c$ is 1-reserving-deciding _{T} .

For any execution δ by processes not in V , we denote by $W(\delta)$ the set of registers written to during δ . In C_r , each register in R_s is covered by a follower of a split pair in V whose leader has already performed the write and is stopped. For any execution δ' in which processes in V do not take steps, we define $\gamma_s(\delta')$ to be the block write to all registers in $R_s \cap W(\delta')$ (registers written to during δ' that are in R_s) by the respective followers of split pairs in V . After each write, another follower catches up with its leader and a previously split pair is united. So, if we run an execution δ' from C_r that changes the contents of some registers in R_s , we can clean these changes up by executing $\gamma_s(\delta')$, which leads to all registers in R_s having the same contents as in C_r .

Using a crude covering argument we can show that

Lemma 9 *The execution α must contain a write step outside $R_c \cup R_s$.*

Proof Assume the contrary. We know that the execution α starting from C_r returns 0. Only processes in P take steps during α , P is disjoint from V and only processes in V take steps during γ_c and $\gamma_s(\alpha)$. T is disjoint from both P and V , and thus the configurations $C_r\alpha\gamma_s(\alpha)\gamma_c$ and $C_r\gamma_c$ are indistinguishable to all processes in T . This is because no process in T has taken steps, the registers in $R_c \cup R_s$ contain the same values, and no other registers have been written to during α , $\gamma_s(\alpha)$ or γ_c . Configuration $C_r\gamma_c$ is 1-reserving-deciding $_T$, hence the same execution from $\text{Res}(C_r\gamma_c, T)$ that returns 1, also returns 1 when executed from $C_r\alpha\gamma_s(\alpha)\gamma_c$. This contradicts the correctness of the consensus algorithm.

Let us write $\alpha = \alpha'w_pw_p\alpha''$, where w_p is the first write step to a register $\text{reg} \notin (R_c \cup R_s)$, performed by some leader-follower pair $p \in P$. Next, we prove the following technical lemma using case analysis:

Lemma 10 *There are disjoint sets of registers R'_s, R'_c , disjoint sets of pairs $V', P', Q' \in U'$ and an execution ρ starting from C_r such that, in configuration $C_r\rho$, the first six properties are satisfied. Furthermore, in $C_r\rho$, each split fresh pair belongs to V' and, for each split stale pair $p \in U' - V' - P' - Q'$, p was a fresh split pair in C_r , neither the follower or the leader of p takes steps in ρ , and ρ includes a write to the register covered by the follower of p .*

Proof In C_r , all split pairs are in V , which is disjoint from T , so T does not contain any split pairs. T contains at least $3m+2 \geq m+1$ pairs that have not returned in $C_r\alpha'$, as no pair in T has returned in C_r or taken a step in α' . Thus, by Lemma 7, $C_r\alpha'$ is v -reserving-deciding $_T$ for at least one $v \in \{0,1\}$. This implies that $C_r\alpha'$ is either 1-reserving-deciding $_T$, or 0-reserving-valent $_T$. We consider these two cases.

Case 1: the configuration $C_r\alpha'$ is 1-reserving-deciding $_T$. Let ℓ be the length of $w_pw_p\alpha''$ and let π_j be the prefix of $w_pw_p\alpha''$ of length $2j$ for $0 \leq j \leq \ell/2$. Hence, the difference between π_j and π_{j+1} is the same step performed twice by a leader and a follower of a pair $p \in P$, as illustrated in Figure 1. Pairs in P are not split in $C_r\alpha'$, since $V \cap P = \emptyset$, α' is a prefix of $\alpha \in \text{Res}(C_r, P)$ with even length, and, by assumption, all split pairs in C_r are in V . We consider two further subcases.

Case 1.1: for some $0 \leq j \leq \ell/2$, the configuration $C_r\alpha'\pi_j$ is reserving-bivalent $_T$. In this case, we let ρ be $\alpha'\pi_j$. Since $C_r\rho$ is reserving-bivalent $_T$ and T contains at

least $3m+2$ unsplit pairs, by Lemma 8, there are $P' \subseteq T$ and $Q' \subseteq T$, with $P' \cap Q' = \emptyset$ and $|P'| = |Q'| = m+1$, such that an execution in $\text{Res}(C_r\rho, P')$ returns 0 and an execution in $\text{Res}(C_r\rho, Q')$ returns 1.

Case 1.2: for every $0 \leq i \leq \ell/2$, the configuration $C_r\alpha'\pi_i$ is reserving-univalent $_T$. By assumption, $C_r\alpha'\pi_0$ is 1-reserving-deciding $_T$, hence, it must be 1-reserving-valent $_T$. On the other hand, α returns 0, so the configuration $C_r\alpha'\pi_{\ell/2}$ must be 0-reserving-valent $_T$. No intermediate configuration is bivalent, so we can find a configuration $C_r\alpha'\pi_j$ that is 1-reserving-valent $_T$, while $C_r\alpha'\pi_{j+1}$ is 0-reserving-valent $_T$, for some $0 \leq j < \ell/2$. We let ρ be $\alpha'\pi_j$.

Let oo be the steps by a leader-follower pair in P separating π_j and π_{j+1} . o may not be a read, as no process in T could distinguish between the 1-reserving-valent $_T$ configuration $C_r\alpha'\pi_j$ and the 0-reserving-valent $_T$ configuration $C_r\alpha'\pi_{j+1}$.

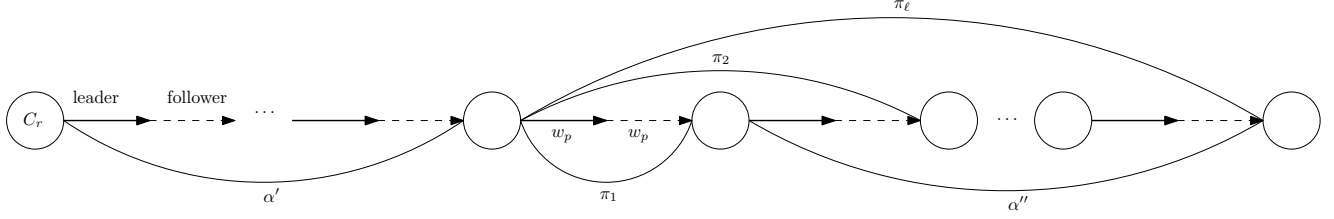
Let Q' be a set of any $m+1$ pairs in T . By Lemma 5, $\text{Res}(C_r\rho, Q')$ is non-empty and since $C_r\rho$ is 1-reserving-valent $_T$, all executions in $\text{Res}(C_r\rho, Q')$ return 1. Recall that U' contains the pairs in U and an additional two leader-follower pairs that have not taken any steps. Let us use these pairs p' and p'' to clone the leader-follower pair performing the write step o . Both cloned leaders and followers will be in the same state as the leader and follower in the original pair performing o . The leader and follower in p' are thus poised to perform write steps o' identical to the step o at configuration $C_r\rho$, and the leader and the follower in p'' are poised to perform o'' identical to o .

Let F be a set of $m+1$ pairs from $T-Q'$. Recall that $|T| \geq 3m+2$. Let P' be $F \cup \{p', p''\}$, so $|P'| = m+3$. By Lemma 5 and the fact that configuration $C_r\alpha\pi_{j+1}$ is 0-reserving-valent $_T$, there is a reserving execution $\xi \in \text{Res}(C_r\alpha\pi_{j+1}, F)$ that returns 0.

The configuration $C_r\rho o' o'$ is indistinguishable from $C_r\alpha\pi_{j+1}$ to any pair in F , because they have not taken steps and the contents of all registers are the same. Thus, execution ξ from $C_r\rho o' o'$ also returns 0 and $o' o' \xi$ from $C_r\rho$ returns 0. Since p'' is poised to perform a write to the register $o' o'$ wrote to, Lemma 6 implies $o' o' \xi \in \text{Res}(C_r\alpha\pi_j, P') = \text{Res}(C_r\rho, P')$. By construction $|P'| + |Q'| = 2m+4$ and $P' \cap Q' = \emptyset$. The reason why we cloned two pairs instead of cloning one pair and using a pair from P that was poised to perform o is that we later require $P', Q' \subseteq T \cup (U' - U)$, in order to ensure that P' and Q' are disjoint from V' , constructed below.

In both subcases, the set R'_s is $R_s - W(\rho)$, i.e. the registers from R_s that have not been written to during the execution $\rho = \alpha'\pi_j$ from C_r . For each of these registers we still have the same pair from V split on it,

Fig. 1 Proof of Lemma 10, Case 1



and since this pair was fresh in C_r and the register has not been written to during ρ , it is still fresh in $C_r\rho$ as required. There are no other fresh split pairs in $C_r\rho$: no new split pairs were introduced during ρ , and the rest of the fresh pairs in C_r were split on $R_s \cap W(\rho)$. These pairs are no longer fresh in $C_r\rho$, as the registers their followers cover were written to during ρ .

The set R'_c is simply $(R_c \cup R_s \cup \{\text{reg}\}) - R'_s = R_c \cup (R_s \cap W(\rho)) \cup \{\text{reg}\}$. We must show that there is a leader-follower pair covering each of these registers in configuration $C_r\rho$. For each register in R_c , we take the same pair from V that was covering it in C_r . For each register in $(R_s \cap W(\rho)) \cup \{\text{reg}\}$, we find a pair from P covering it in $C_r\rho$. Since α is a reserving execution from C_r , all its prefixes of an even length including $\alpha'\pi_j$ are also reserving. Thus, in $C_r\alpha'\pi_j = C_r\rho$, for each register that has been written to during ρ , in particular for registers in $(R_s \cap W(\rho)) \cup \{\text{reg}\}$, there is an unsplit pair in P that covers it in configuration $C_r\rho$. Technically, if $j = 0$, register reg is not yet written, but the next step in α is w_p by a pair covering reg .

The set $V' \subseteq V \cup P$ contains all $r + 1$ pairs we used to cover registers in $R'_c \cup R'_s = R_c \cup R_s \cup \{\text{reg}\}$. Recall that, by our construction, $P', Q' \subseteq T \cup (U' - U)$. Also, T was disjoint from V , P and Q , and so are the two pairs in $U' - U$ that had never taken steps before. Therefore, we have $(P' \cup Q') \cap V' = \emptyset$ as required.

Case 2: the configuration $C_r\alpha'$ is 0-reserving-valent $_T$: No processes in V take steps in α' , and therefore $\gamma_s(\alpha')$ is well defined. In $\gamma_s(\alpha')$, each register in $R_s \cap W(\alpha')$ is overwritten so that it has the value the register had in C_r by the follower of the split pair in V that covered it. Hence, for all processes in T , configuration $C_r\alpha'\gamma_s(\alpha')\gamma_c$ is indistinguishable from the 1-reserving-deciding $_T$ configuration $C_r\gamma_c$. This is because the processes in T have not taken steps (since T is disjoint from $P \cup V$) and the contents of all registers are the same in these configurations (since α' contains writes only to registers in $R_c \cup R_s$).

Let us denote by ℓ the length of execution $\gamma_s(\alpha')\gamma_c$ and let σ_j be the prefix of this execution of length j for $0 \leq j \leq \ell$. Notice that, unlike the previous case,

where the difference between π_j and π_{j+1} was a pair of identical steps by a leader and the follower, the difference between σ_j and σ_{j+1} is exactly one step, by either a leader or a follower of some pair. By definition, each step in $\gamma_s(\alpha')$ is performed by a follower (uniting a previously split pair), while each step in γ_c is performed by a leader (creating a new fresh split pair). This is illustrated in Figure 2.

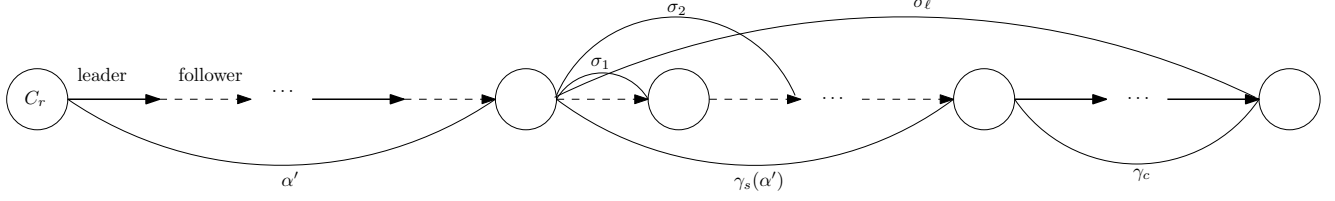
Case 2.1: for some $0 \leq j \leq \ell$, the configuration $C_r\alpha'\sigma_j$ is reserving-bivalent $_T$: We let ρ be $\alpha'\sigma_j$. As in **Case 1.1**, $C_r\rho$ is reserving-bivalent $_T$ and $|T| \geq 3m+2$. Therefore, by Lemma 8, there are $P' \subseteq T$ and $Q' \subseteq T$, with $P' \cap Q' = \emptyset$ and $|P'| = |Q'| = m + 1$, such that an execution in $\text{Res}(C_r\rho, P')$ returns 0 and an execution in $\text{Res}(C_r\rho, Q')$ returns 1.

Case 2.2: for every $0 \leq i \leq \ell$, the configuration $C_r\alpha'\sigma_i$ is reserving-univalent $_T$: The configuration $C_r\alpha'\sigma_0$ is 0-reserving-valent $_T$ by the assumption. Since all writes in α' are to registers in $R_c \cup R_s$, $\gamma_s(\alpha')$ overwrites each register in $W(\alpha')$ with the value it had in C_r , and $\sigma_\ell = \gamma_s(\alpha')\gamma_c$, the contents of all registers are the same in $C_r\alpha'\sigma_\ell$ and $C_r\gamma_c$. Moreover, the processes in T have not taken steps in $\alpha'\sigma_\ell$ or γ_c . Therefore, configuration $C_r\alpha'\sigma_\ell$ is indistinguishable from $C_r\gamma_c$ to these processes. Since $C_r\gamma_c$ is 1-reserving-deciding $_T$, so is $C_r\alpha'\sigma_\ell$.

No intermediate configuration is bivalent, so there is a configuration $C_r\alpha'\sigma_j$ that is 0-reserving-valent $_T$, while $C_r\alpha'\sigma_{j+1}$ is 1-reserving-valent $_T$, for some $0 \leq j < \ell$. We let ρ be $\alpha'\sigma_j$.

We can construct P' and Q' in a very similar way to **Case 1.2**. If o is the step separating σ_j and σ_{j+1} , o cannot be a read as no process in T could distinguish between the 0-reserving-valent $_T$ configuration $C_r\alpha'\sigma_j$ and the 1-reserving-valent $_T$ configuration $C_r\alpha'\sigma_j o = C_r\alpha'\sigma_{j+1}$. We use two pairs in $U' - U$ to clone leader-follower pairs p' and p'' , both about to perform identical write steps o' and o'' . We let P' be a set of any $m + 1$ pairs in T and Q' be a set of $m + 3$ pairs, with $m + 1$ pairs from $T - P'$ and the two cloned pairs. Then, by the same argument as in **Case 1.2**, P' and Q' satisfy all required properties.

Fig. 2 Proof of Lemma 10, Case 2



In both subcases, the set R'_s is $(R_s - W(\alpha')) \cup (R_c \cap W(\sigma_j))$. It consists of registers from R_s that were not written to during α' and registers from R_c that were written to during σ_j (during the prefix of block write γ_c that is a suffix of σ_j).

For each register in $R_s - W(\alpha')$, we still have the same pair from V split on it in configuration $C_r\rho$ as in C_r . This split pair was fresh in C_r and covered a register in R_s that has not been written to during $\rho = \alpha'\sigma_j$. Therefore, it is still fresh as required in $C_r\rho$. Execution σ_j may contain a prefix of γ_c , during which only leaders but not the followers take steps and new fresh split pairs are created. These pairs are split on registers in $R_c \cap W(\sigma_j)$. There is one newly split fresh pair for each of these registers. Fresh pairs that were split on $R_s \cap W(\alpha')$ in C_r are not fresh in $C_r\rho$, as the registers covered by their followers were written to during α' . Thus no other split pairs are fresh.

The set R'_c is $(R_c - W(\sigma_j)) \cup (R_s \cap W(\alpha')) \cup \{\text{reg}\}$. Recall that α' is a prefix of the reserving execution $\alpha \in \text{Res}(C_r, P)$, so for each register in $R_s \cap W(\alpha')$ there is a covering pair from P in $C_r\rho$. The register **reg** is covered by the leader-follower pair in P with the pending write w_p . For each register in $R_c - W(\sigma_j)$, we take the pair from V that was covering it in C_r . Neither leader nor follower in this pair have taken steps during $\rho = \alpha'\sigma_j$ and still cover the same register in $C_r\rho$.

As in **Case 1**, the set $V' \subseteq V \cup P$ contains all $r + 1$ pairs used to cover registers in $R'_c \cup R'_s = R_c \cup R_s \cup \{\text{reg}\}$. Also, $P', Q' \subseteq T \cup (U' - U)$ and it follows that $(P' \cup Q') \cap (V') = \emptyset$.

In order to finish the proof, we need to get rid of any stale split pairs that exist in configuration $C_r\rho$. There are no stale split pairs in C_r , so any that exist in $C_r\rho$ are created during ρ . For each register in $R_s \cap W(\rho)$, there is a split pair $p \in V$ that was fresh in C_r , but is stale in $C_r\rho$, because the register covered by the follower of p was overwritten during ρ . Let Y be the set of these pairs. By Lemma 10, Y is disjoint from $V' \cup P' \cup Q'$.

We now modify the execution ρ . For each pair p in Y let s be the first write step during ρ to the register the follower of p covers. We add a write step by the follower of p in execution ρ immediately before step s . This way,

no process other than the follower may observe a difference, since the changes are promptly overwritten by s . The follower, meanwhile, catches up with the leader, hence, the pair is re-united.

Let ρ' be the resulting execution and let $C_{r+1} = C_r\rho'$. Then C_{r+1} does not contain any stale split pairs, satisfying the seventh property. R'_s, R'_c, V', P', Q' are defined according to Lemma 10. Since Y is disjoint from $V' \cup P' \cup Q'$, C_{r+1} is indistinguishable from $C_r\rho$ for all processes in $V' \cup P' \cup Q'$. Thus, the first six properties are also satisfied due to Lemma 10.

Theorem 1 *In a system of n anonymous processes, any consensus algorithm satisfying non-deterministic solo termination must use $\lfloor n/14 \rfloor - 1$ registers.*

Proof Suppose for the sake of contradiction that all executions use at most $m = \lfloor n/14 \rfloor - 2$ registers, where n is the number of anonymous processes. Then, by applying the construction described above $m + 1$ times, we can reach C_{m+1} starting from C_0 using $10m + 12 + 4m + 4 < n$ processes. In configuration C_{m+1} , there are $m + 1$ registers in $R_c \cup R_s$. Each register in R_s has already been written to and each register in R_c is covered by an un-split pair of processes.

5 Acknowledgments

The author would like to thank Nir Shavit, Michael Coulombe, Dan Alistarh, Faith Ellen and Leqi Zhu for helpful conversations and feedback, and anonymous reviewers for their excellent comments.

References

1. Abrahamson, K.: On achieving consensus using a shared memory. In: Proceedings of the 7th annual ACM Symposium on Principles of Distributed Computing, pp. 291–302. ACM (1988)
2. Aspnes, J., Herlihy, M.: Fast randomized consensus using shared memory. *Journal of Algorithms* **11**(3), 441–461 (1990)
3. Attiya, H., Ellen, F.: Impossibility Results for Distributed Computing, vol. 5. Morgan & Claypool Publishers (2014)

4. Ben-Or, M.: Another advantage of free choice (extended abstract): Completely asynchronous agreement protocols. In: Proceedings of the 2nd Annual ACM Symposium on Principles of Distributed Computing, pp. 27–30. ACM (1983)
5. Bouzid, Z., Raynal, M., Sutra, P.: Brief announcement: Anonymous obstruction-free (n, k) -set agreement with $n-k+1$ atomic read/write registers. In: Proceedings of the 29th International Symposium on Distributed Computing, p. 669. Springer (2015)
6. Bowman, J.: Obstruction-free snapshot, obstruction-free consensus, and fetch-and-add modulo k . Tech. Rep. TR2011-681, Dartmouth College, Computer Science, Hanover, NH (2011)
7. Ellen, F., Gelashvili, R., Zhu, L.: Revisionist simulations: A new approach to proving space lower bounds. arXiv preprint arXiv:1711.02455 (2018)
8. Fich, F., Herlihy, M., Shavit, N.: On the space complexity of randomized synchronization. *Journal of the ACM (JACM)* **45**(5), 843–862 (1998)
9. Fischer, M.J., Lynch, N.A., Paterson, M.S.: Impossibility of distributed consensus with one faulty process. *Journal of the ACM (JACM)* **32**(2), 374–382 (1985)
10. Giakkoupis, G., Helmi, M., Higham, L., Woelfel, P.: An $\mathcal{O}(\sqrt{n})$ space bound for obstruction-free leader election. In: Proceedings of the 27th International Symposium on Distributed Computing, pp. 46–60. Springer (2013)
11. Giakkoupis, G., Helmi, M., Higham, L., Woelfel, P.: Test-and-set in optimal space. In: Proceedings of the 47th Annual ACM on Symposium on Theory of Computing, pp. 615–623. ACM (2015)
12. Guerraoui, R., Ruppert, E.: What can be implemented anonymously? In: Proceedings of the 19th International Symposium on Distributed Computing, pp. 244–259. Springer (2005)
13. Saks, M., Shavit, N., Woll, H.: Optimal time randomized consensus – making resilient algorithms fast in practice. In: Proceedings of the 2nd Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 351–362. Society for Industrial and Applied Mathematics (1991)
14. Styer, E., Peterson, G.L.: Tight bounds for shared memory symmetric mutual exclusion problems. In: Proceedings of the 8th Annual ACM Symposium on Principles of Distributed Computing, pp. 177–191. ACM (1989)
15. Zhu, L.: Brief announcement: Tight space bounds for memoryless anonymous consensus. In: Proceedings of the 29th International Symposium on Distributed Computing, p. 665. Springer (2015)
16. Zhu, L.: A tight space bound for consensus. In: Proceedings of the 48th Annual ACM Symposium on Theory of Computing, pp. 345–350. ACM (2016)