An Almost-Logarithmic Lower Bound for Leader Election with Bounded Value Contention

Dan Alistarh
Institute of Science and Technology Austria (ISTA), Klosterneuburg, Austria

Faith Ellen
University of Toronto, Canada

Alexander Fedorov
Institute of Science and Technology Austria (ISTA), Klosterneuburg, Austria

Abstract

We investigate the step complexity of the Leader Election problem (and implementing the corresponding test-and-set object) in asynchronous shared memory, where processes communicate through registers supporting atomic read and write and must coordinate so that a single process becomes the leader. Determining tight step complexity bounds for solving this problem is one of the key open problems in the theory of shared memory distributed computing. The best known algorithm is a randomized tournament-tree, which has worst-case expected step complexity $O(\log N)$ for N processes. There are provably no deterministic wait-free algorithms, and only restricted lower bounds are known for obstruction-free and randomized wait-free algorithms. We introduce a new lower bound that establishes an $\Omega\left(\frac{\log N}{\log\log N+\log Q}\right)$ step complexity for any obstruction-free Leader Election algorithm, where N is the number of processes, and $2 \leq Q \leq N$ is a bound on the *value contention*, which we define as the maximum number of different values that processes can be simultaneously poised to write to the same register in any execution of the algorithm. Our result is strictly stronger than previous bounds based on write contention. In particular, it implies new lower bounds on step complexity that depend on register size.

2012 ACM Subject Classification Theory of computation → Shared memory algorithms; Theory of computation → Lower bounds and information complexity; Theory of computation → Concurrency

Keywords and phrases Leader Election, Test-and-Set, Shared Memory, Lower Bounds

Digital Object Identifier 10.4230/LIPIcs.DISC.2025.3

Funding The work of Dan Alistarh is supported by grants from ERC, Austrian FWF, and the Google and NVIDIA corporations. Faith Ellen was supported in part by the Natural Science and Engineering Research Council of Canada (NSERC) grant RGPIN-2020-04178.

Acknowledgements The authors would like to thank the DISC 2025 anonymous reviewers for their detailed comments and, in particular, Reviewer C, who inspired our approach for a new and simpler proof of Theorem 5 via the probabilistic method.

1 Introduction

Leader Election is a fundamental coordination problem in distributed computing, in which, as the name suggests, a set of n processes compete to become the leader. Each process should output either `win` or `lose`, indicating whether the process became the leader. At most one process should become the leader and, if all processes that compete take enough steps, then exactly one of these processes becomes the leader. Note that it is not required for other processes to know the identity of the leader, as this would make the problem equivalent to *Consensus*, which is known to be a harder coordination problem.

A closely-related problem is implementing a *test-and-set* object. It differs from Leader Election in that no process may return *lose* before the eventual winner has started the computation. However, any Leader Election algorithm can be deterministically transformed into an implementation of a test-and-set object with just one additional register and at most two additional steps by each process [4].

Leader Election and test-and-set objects have been studied with different formulations and in various models for almost four decades, e.g. [25, 28, 2, 12, 5]. In this paper, we will focus on Leader Election in the *asynchronous shared memory* model [22, 9], where processes can communicate only through *registers*, shared objects that support *read* and *write* operations. Read returns the value stored in the register, while a write atomically update the stored value. In this way, one process can learn about another process if it reads a register that was last written to by that process. A process executes reads and writes on arbitrary registers one by one, but the way in which the operations of a process are interleaved with those of other processes is controlled by an *adversary*. Specifically, it is assumed that the adversary decides the order of the steps, with the goal of making processes execute as many steps as possible. It is known that a test-and-set object has consensus number two. Thus, it has no deterministic wait-free implementation from registers [17].

1.1 Prior Work

The first randomized algorithm for Leader Election among two processes was given by Tromp and Vitányi [26]. The approach was later generalized to N processes by Afek, Gafni, Tromp and Vitányi [1] using a tournament tree [24], in which each process starts at a leaf of the tree. At each node, the process competes in an instance of Leader Election among two processes. If it loses, it immediately returns *lose*. If it wins, it continues to the parent node. The winner at the root can return *win*. This algorithm has $O(\log N)$ worst-case expected step complexity, since each instance of Leader Election among two processes takes an expected constant number of steps and the tournament tree has $O(\log N)$ depth. This can be straightforwardly modified to an obstruction-free variant with logarithmic solo step complexity [5].

The approach described above leads to the best known upper bound for the case of a strong adversary. Whether this upper bound is optimal remains an open problem [15]. Later work extended these results to the adaptive setting, where the step complexity depends on the number of participating processes, k. Specifically, Alistarh, Attiya, Gilbert, Giurgiu, and Guerraoui proposed the `RatRace` algorithm which achieves $O(\log k)$ step complexity in expectation [4, 14]. Golab, Hendler and Woelfel designed an algorithm that solves Leader Election in $O(1)$ remote memory accesses [16]. However, their algorithm is not obstruction-free: if a process crashes, this may cause another process to wait forever for the value in a register to change.

Remarkably, despite decades of interest in the relationship between upper and lower bounds for Leader Election in shared memory, we still do not know whether the logarithmic upper bound is tight, neither for the expected step complexity of randomized algorithms against a strong adversary, nor for the solo step complexity of deterministic obstruction-free algorithms. Intuitively, proving lower bounds for test-and-set is hard because of the apparent simplicity of the object.

The list of relevant lower bounds for Leader Election against a strong adversary is shown in Table 1. In 1989, Styer and Peterson proved that any Leader Election algorithm has to use $\Omega(\log N)$ registers [25]. This space complexity lower bound was later matched by an obstruction-free Leader Election algorithm that uses $O(\log N)$ registers [13]. The remaining lower bounds, including ours, build on ideas from the lower bounds for mutual exclusion

established by James H. Anderson and Yong-Jik Kim [7]. They proved an $\Omega(\frac{\log N}{\log \log N})$ step complexity lower bound for Mutual Exclusion by inductively constructing an execution. Their analysis distinguishes between two scenarios: the high-contention and the low-contention cases. In the low-contention case, they apply a graph theory result to find a large set of processes that do not conflict with one another and use them to extend their execution. In the high-contention case, they run some processes until they enter and leave the critical section, thereby overwriting the contents of the highly contended registers.

The first work to apply Anderson and Kim's ideas to Leader Election was by Alistarh, Gelashvili, and Nadiradze [5]. They proved that any algorithm has worst-case expected step complexity $\Omega\left(\frac{\log N}{\log \kappa}\right)$, where κ is the maximum *write contention*. However, this lower bound becomes trivial when polynomially many processes can write to the same register, that is, when $\kappa = N^{\varepsilon}$, for some constant $\varepsilon > 0$. In other related work, Eghbali and Woefel proved an $\Omega(\frac{\log N}{\log \log N})$ lower bound for implementing an *Abortable* Test-And-Set object [10]. They use a different model, which counts the maximum number of remote memory references (RMRs) performed by a process, rather than the maximum number of steps. However, their proof still follows Anderson and Kim's framework. To handle the high-contention case, their proof heavily relies on the fact that processes can be aborted after they overwrite some register. Thus, their results apply only to Abortable Leader Election, and not to our model.

Table 1 Lower bounds for Leader Election against strong adversary.

Lower Bound	Reference	Comment
$\Omega(\log N)$ *space* complexity	[25]	
$\Omega\left(\frac{\log N}{\log \kappa}\right)$ step complexity	[5]	κ is the write contention
$\Omega\left(\frac{\log N}{\log \log N}\right)$ RMR complexity	[10]	Only for *Abortable* Leader Election
$\Omega\left(\frac{\log N}{\log \log N + \log Q}\right)$ step complexity	**This paper**	Q is the value contention

1.2 Contribution

In this paper, we present a new technique that leads to a lower bound for Leader Election under a more general notion of contention, which we call *value contention*. Specifically, we show a lower bound of $\Omega\left(\frac{\log N}{\log \log N + \log Q}\right)$ on the solo step complexity of any obstruction-free Leader Election algorithm for N processes, where $2 \leq Q \leq N$ is an upper bound on the *value contention*, that is, the *number of different values that processes can be simultaneously poised to write to the same register* in any execution.

This result leads to a new trade-off between the maximum register capacity and the solo step complexity of test-and-set: for instance, if registers can only hold a *poly-logarithmic* number of different values, then test-and-set requires $\Omega(\log N / \log \log N)$ steps in a solo execution, in the worst-case. Similarly, if at most polylog N processes may be simultaneously poised to write to the same register, we obtain the $\Omega(\log N / \log \log N)$ lower bound of Alistarh, Gelashvili and Nadiradze [5], but via new argument. Put differently, our proof shows that constant-time algorithms for test-and-set are only possible if, for some constant $0 < c \leq 1$, N^c processes can concurrently be poised to write different values to the same register. Furthermore, our lower bound shows that no adaptive Leader Election algorithm is possible, unless registers can hold polynomial in N many values, and polynomial in N processes can be poised to write different values to the same register at the same time. We say that an algorithm is *adaptive* if its step complexity depends only on the actual number of participating processes and not on the total number of processes in the system.

Our lower bound is based on a technique that generalizes the knowledge-based approach of Anderson and Kim [7]. At a high level, we inductively construct a sequence of executions in which no active process sees any other active process. Specifically, this leads to an execution by one process in which it takes $\Omega\left(\frac{\log N}{\log\log N+\log Q}\right)$ steps. In each round of the execution, active processes perform one additional read or write. Thus, by round t, each has taken t steps. We split these active processes into groups, called *factions*, depending on the step they are about to perform. We then proceed to schedule the processes adversarially, to minimize the information flow between them. Since information flow may arise from values written to registers in previous rounds, we also erase some processes, or even entire factions, from the execution. To determine which active processes should continue to the next round, we build a conflict graph, where an edge exists between two processes if executing them both would create information flow, or if the two processes are poised to write different values to the same register simultaneously.

So far, this construction is similar to step complexity lower bound proofs for low contention [7, 10, 5]. The key new observation is that we can adjust the execution so that the conflict graph can be represented by a new, special kind of structure that we call a *faction graph*. Roughly, a faction graph can be partitioned into sets of vertices with super-edges from vertices to parts of the partition. Intuitively, this allows us to generate a *compact representation* for any conflict graph. Our main technical lemma is a lower bound of $\Omega\left(\frac{|V|^2}{|E^*|+|V|}\right)$ on the maximum size of an independent set in any conflict graph represented by a faction graph with $|V|$ vertices and $|E^\star|$ multi-edges. This lemma may be of separate interest for other problems that can be described using faction graphs.

Our lower bound argument combines the fact that the conflict graph at each round can be represented by a faction graph, with our technical lemma about such graphs to obtain a new lower bound of $\Omega\left(\frac{N}{Q+t^2}\right)$ on the number of processes that survive until the t^{th} round. These processes do not see one another. We can continue, while keeping at least one process active, for $\Omega\left(\frac{\log N}{\log\log N+\log Q}\right)$ rounds.

The paper by Alistarh, Gelashvili and Nadiradze [5] presents a complex, potential-based covering argument leading to a lower bound of $\Omega(\log N/\log\kappa)$ that depends on step contention κ. In a follow-up online revision [6], the authors provided a simpler argument, which carefully re-works the original Anderson and Kim lower bound to obtain a simpler proof of the same lower bound. Our lower bound proves a slightly weaker result in the bounded *step contention* case. Our focus is on the bounded *value contention* case, which is a generalization of the case when registers have bounded size.

On the upper bound side, we observe that the tournament tree algorithm by Afek, Gafni, Tromp and Vitányi [1] has constant bounded value contention Q. In fact, with slight modifications, it uses only single-bit registers, and thus has value contention $Q = 2$. Therefore, our $\Omega\left(\frac{\log N}{\log\log N}\right)$ lower bound almost matches the $O(\log N)$ upper bound. Moreover, even though there are adaptive Leader Election algorithms with $O(\log k)$ step complexity [4, 14] (where k is the number of participating processes), our main lower bound implies that adaptive Leader Election is impossible unless $Q \in \Omega(N^c)$ for some constant $c > 0$. In other words, we prove that any algorithm for adaptive Leader Election must use registers of logarithmic size and has executions where the number of processes simultaneously poised to write different values to the same register is polynomial in N.

2 Preliminaries

We consider an asynchronous distributed system where a set of N processes, P, communicate through a set of R registers that can be read from and written to by any process. A *step* is a read of a specific register or a write of a specific value to a specific register performed by one process, followed by an update of the state of the process, including any local coin flips it wishes to perform.

An *execution* $\mathcal{E}$ is a sequence of steps of processes in the system, starting from the initial configuration. An execution is P'*-only* if it only contains steps performed by processes in $P' \subseteq P$. If $\mathcal{E}$ is a prefix of execution $\mathcal{E}'$, we say that $\mathcal{E}'$ extends $\mathcal{E}$. A process p is *poised to write* to a register r at the end of an execution $\mathcal{E}$ if, in every execution $\mathcal{E}'$ that extends $\mathcal{E}$ and in which p performs more steps, p's next step after $\mathcal{E}$ is writing to r. An execution is *terminating* if all operations (that have begun) have finished. A *round-by-round execution* is an execution in which, for every positive integer i, all processes that take at least i steps do so before any process takes its $i+1$'st step. The i'th *round* of the execution is a consecutive sequence of steps by different processes, each of which has previously taken $i-1$ steps. Thus, a round-by-round execution is a concatenation of rounds.

The *step complexity* of an execution is the maximum number of steps performed by one process. The *step complexity* of an algorithm is the maximum step complexity of any execution of the algorithm. The *solo step complexity* of an algorithm is the maximum number of steps in any solo execution of the algorithm, i.e., in which only one process takes steps. The *total step complexity* of an algorithm is the maximum number of steps in any execution of the algorithm.

Adversary. This paper assumes the strong (adaptive) adversary model. In this model, the order in which processes take steps is decided adaptively by an adversary. It knows everything about the current state of all processes when it chooses which process takes the next step. By comparison, weaker adversaries have only partial information about process states such as the outcomes of random coin flips [3, 14]. To be specific, our lower bound proofs assume that the adversary knows the next step of every process, but does not need other information about internal process states.

Worst-case expected step complexity. Consider a (possibly randomized) algorithm $\mathcal{A}$ and an adaptive adversary D, which we assume, for simplicity, to be deterministic. Let $T_{p,D}$ be the random variable denoting the number of steps process p performs in the execution of $\mathcal{A}$ when the scheduling is done by the adversary D. The *expected step complexity* of an execution under D is $\mathbb{E}[T] = \max_{p \in P}[T_{p,D}]$, where the expectation is taken over the processors' coin flips. The *worst-case expected step complexity* of algorithm $\mathcal{A}$ is $\sup_D [\mathbb{E}(T_{p,D})]$, i.e. the supremum over the expected step complexity of $\mathcal{A}$, under any adversary.

Leader Election. A *Leader Election* algorithm supports a single operation, `elect`, which each process may perform at most once. This operation returns either `win` or `lose` indicating whether the process won the election. It satisfies the following properties:

- With probability 1, every process that does not crash finishes its operation in a finite number of steps.
- At most one operation returns `win`.
- If all operations that started have finished, then exactly one operation returned `win`.

Splitter. A *Splitter* [21, 23] is an object that supports a single operation, `decide`, which each process may perform at most once. It satisfies the following properties:

- The output of `decide` is either `Stop`, `Left`, or `Right`.
- Every process that calls `decide` finishes it within a finite number of steps.
- At most one process can get `Stop`.
- If only one process calls `decide`, then it is guaranteed to get `Stop`.
- If more than one process calls `decide`, then not all processes get the same output.

Instead of the last property, a *Randomized Splitter* [8] requires that the probability a call of `decide` returns `Left` is the same as the probability it returns `Right`, independent of all other calls. Splitter and Randomized Splitter objects are similar to Leader Election, where getting `Stop` as output is analogous to winning, but they do not guarantee that some process gets output `Stop`, even if all calls have terminated.

We now describe formalize some well-known results used for to proving lower bounds, which were previously applied for this and other distributed problems [19, 14, 5, 25, 10].

Process Visibility. We say that process p *sees* another process if p reads a value from some register that differs from the last value p wrote there or is not the initial value of the register, if p never wrote there. We say that p *sees* process q if q was the last process that wrote to that register before p read it. For any execution $\mathcal{E}$, define $p \leftrightarrow_{\mathcal{E}} q$ if p sees q or q sees p in execution $\mathcal{E}$. Let $p \sim_{\mathcal{E}} q$ be the reflexive transitive closure of $\leftrightarrow_{\mathcal{E}}$. This is an equivalence relation. A set of processes $P' \subseteq P$ is *closed* (with respect to $\sim_{\mathcal{E}}$) if there does not exist $q \in P'$ and $p \notin P'$ such that $q \sim_{\mathcal{E}} p$.

Note that, if P' is a set of processes that is closed with respect to $\sim_{\mathcal{E}}$, then erasing the steps of all processes in P' from $\mathcal{E}$ results in an execution. Since exactly one operation wins in every terminating execution of Leader Election, we get the following result.

▶ **Lemma 1.** *Let $\mathcal{E}$ be an execution of a Leader Election algorithm and let P' be a nonempty set of processes that is closed with respect to $\sim_{\mathcal{E}}$. Let $\mathcal{E}'$ be an execution obtained by erasing the steps of all processes in $P \setminus P'$ from $\mathcal{E}$ and then running all processes in P' until they each finish performing `elect`. Then exactly one process in P' wins in $\mathcal{E}'$.*

Based on this lemma, we formulate the first commonly used idea guaranteeing that processes have to take more steps in an execution $\mathcal{E}$ of Leader Election that contains at least two processes that are not related by $\sim_{\mathcal{E}}$.

▶ **Lemma 2.** *Let $\mathcal{E}$ be an execution of a Leader Election algorithm. Consider a partition of P into sets that are closed with respect to $\sim_{\mathcal{E}}$. Then every set, except possibly one, contains at least one process that has not finished performing `elect`.*

Proof. For every part, P', of the partition, there is an execution obtained by erasing the steps of all processes in $P \setminus P'$ from $\mathcal{E}$ and then running all processes in P' until they each finish performing `elect()`. By Lemma 1, exactly one process in P' wins in this execution. Consider the winning process for each set P' in the partition. Since at most one process wins in $\mathcal{E}$, each of these processes, except possibly one, has not won in $\mathcal{E}$ and, thus, it has not finished performing `elect`. ◀

Intuitively, Lemma 2 shows that, if we execute processes in a way that prevents them from learning about one another, we can ensure that they have to take more steps.

3 Independent Set Lower Bound for Faction Graphs

In this section, we describe our main combinatorial result, which we later use to prove our main lower bound. It is a stronger, specialized version of Turán's Theorem [27]:

▶ **Theorem 3.** *Every graph $G = (V, E)$ contains an independent set of size at least* $\left\lfloor \frac{|V|^2}{2|E|+|V|} \right\rfloor$.

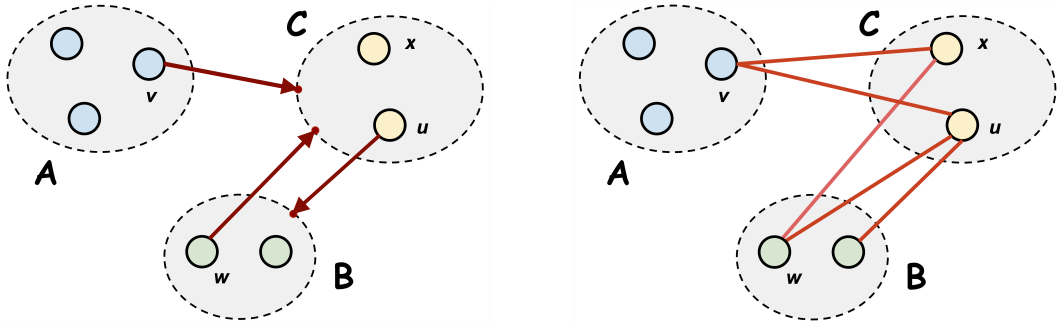
We prove that any graph that can be represented in a special form, which we call a faction graph, has a sufficiently large independent set.

Faction Graphs. A *faction graph* $G^\star = (V, F, E^\star)$ is defined by a partition, F, of its vertex set, V, and a set of *super-edges*, $E^\star$, which connects vertices to parts of the partition, called *factions*. It contains no edges from any vertex to the faction that contains it. The graph $G = (V, E)$ *represented by* $G^\star$ is an undirected graph with the property that, if a vertex v has a super-edge in $E^\star$ to a faction, $A \in F$, then all vertices in A are neighbors of v in G. An example of a faction graph is shown in Figure 1a. The formal definition is given below.

▶ **Definition 4.** *$G^\star = (V, F, E^\star)$ is a* faction graph *if V is a set of vertices, F is a partition of V, and $E^\star \subseteq V \times F$ such that $v \notin A$ for all $(v, A) \in E^\star$. The graph* represented by *this faction graph is $G = (V, E)$, where $E = \{\{v, w\} \mid (v, A) \in E^\star$ and $w \in A$ for some $A \in F\}$.*

Generally, a faction graph is a much more compact representation of the graph it represents.

Figure 1a is a faction graph with 7 vertices, 3 factions, A, B, and C, and 3 super-edges. Figure 1b is the graph it represents. The super-edge (w, C) is expanded into the two edges $\{w, x\}$ and $\{w, u\}$.



(a) A faction graph. **(b)** The graph it represents.

Figure 1 Illustration of faction graphs.

Independent Sets in Faction Graphs. We will show, using the probabilistic method, that the graph $G = (V, E)$ represented by a faction graph $G^\star = (V, F, E^\star)$ has an independent set of size $\Omega\left(\frac{|V|^2}{|E^*|+|V|}\right)$. This is similar to Turán's theorem (Theorem 3). Our key observation is that we can obtain a much bigger lower bound on the size of an independent set, since $|E^*|$ can be asymptotically much smaller than $|E|$.

▶ **Theorem 5.** *The graph $G = (V, E)$ represented by a faction graph $G^\star = (V, F, E^\star)$ contains an independent set of size at least*

$$\frac{1}{e} \cdot \frac{|V|^2}{|V| + |E^\star|} \in \Omega\left(\frac{|V|^2}{|V| + |E^\star|}\right).$$

Proof. Let $N = |V|$ and $M^\star = |E^\star|$. If $E^\star = \emptyset$, then $E = \emptyset$, so V is an independent set of G of size $|V|$. Therefore, assume $M^\star > 0$.

Since there are no edges in E between two nodes in the same faction, every faction is an independent set. For each vertex $v \in V$, let $F(v) \in F$ be the faction that contains v and let

$$\mathrm{OutNhd}^\star(v) = \{\, C \in F \mid (v, C) \in E^\star \,\}.$$

By definition, $F(v) \notin \mathrm{OutNhd}^\star(v)$. Observe that

$$\sum_{v \in V} |\mathrm{OutNhd}^\star(v)| \;=\; |E^\star| \;=\; M^\star,$$

since every super-edge $(v, C) \in E^\star$ contributes exactly once to the sum.

We use the probabilistic method. Select each faction $A \in F$, independently, with some fixed probability $p \in (0, 1)$, obtaining a random subset $F' \subseteq F$. From F' build a random vertex set

$$S = \big\{\, v \in V \mid F(v) \in F' \;\text{ and }\; F' \cap \mathrm{OutNhd}^\star(v) = \emptyset \,\big\}$$

consisting of every vertex whose faction was selected, but has no super-edge to any selected faction.

First, we show that S is an independent set of $G = (V, E)$. Consider any two vertices, $u, v \in S$. If $F(u) = F(v)$, then $\{u, v\} \notin E$, since every faction is an independent set. So suppose $F(u) \neq F(v)$. By definition of S, $F(u), F(v) \in F'$, so $F(v) \notin \mathrm{OutNhd}^\star(u)$ and $F(u) \notin \mathrm{OutNhd}^\star(v)$. Hence $(u, F(v)), (v, F(u)) \notin E^\star$. By Definition 4, it follows the $\{u, v\} \notin E$.

Next, we get a lower bound on the expected size of S. For any fixed $v \in V$, the event that $F(v) \in F'$ is independent of the event that $F' \cap \mathrm{OutNhd}^\star(v) = \emptyset$. Therefore

$$\Pr[v \in S] \;=\; p\,(1-p)^{|\mathrm{OutNhd}^\star(v)|} \text{ and } \mathbb{E}\big[|S|\big] \;=\; \sum_{v \in V} \Pr[v \in S] \;=\; \sum_{v \in V} p\,(1-p)^{|\mathrm{OutNhd}^\star(v)|}.$$

The function $f(x) = (1-p)^x$ is convex for $p \in (0, 1)$. By Jensen's inequality [18],

$$\frac{1}{N} \sum_{v \in V} (1-p)^{|\mathrm{OutNhd}^\star(v)|} \;\geq\; (1-p)^{\frac{1}{N}\sum_{v \in V} |\mathrm{OutNhd}^\star(v)|}.$$

Hence $\mathbb{E}\big[|S|\big] \;\geq\; N\,p\,(1-p)^\alpha$, where $\alpha = M^\star / N$.

We now choose a good value for p. Consider the function $g(p) = p(1-p)^\alpha$ on the interval $[0, 1]$. Since its derivative is $g'(p) = (1-p)^{\alpha-1}\big(1 - (\alpha+1)p\big)$, it follows that g is maximized when $p = 1/(1+\alpha)$. Using this value of p gives

$$\mathbb{E}\big[|S|\big] \;\geq\; N \cdot \frac{1}{1+\alpha} \cdot \Big(1 - \frac{1}{1+\alpha}\Big)^\alpha \;=\; \frac{N}{1+\alpha} \cdot \Big(\frac{\alpha}{1+\alpha}\Big)^\alpha.$$

Since $\ln(1+x) \leq x$ for $x > -1$, we have

$$\Big(\frac{\alpha}{1+\alpha}\Big)^\alpha \;=\; \exp\Big(\alpha \ln \frac{\alpha}{1+\alpha}\Big) \;=\; \exp\Big(-\alpha \ln\big(1 + \tfrac{1}{\alpha}\big)\Big) \;\geq\; \exp(-1) \;=\; \frac{1}{e}.$$

Hence

$$\mathbb{E}\big[|S|\big] \;\geq\; \frac{1}{e} \cdot \frac{N}{1+\alpha} \;=\; \frac{1}{e} \cdot \frac{N^2}{N + M^\star} = \frac{1}{e} \cdot \frac{|V|^2}{|V| + |E^\star|}.$$

Because S is an independent set for every choice of F', it follows that G contains an independent set S of at least this size. ◀

4 The Leader Election Lower Bound

With these preliminaries in place, we now present a new step complexity lower bound for the Leader Election problem (and implementing a test-and-set object) in asynchronous shared memory. To date, no non-trivial lower bound on step complexity for implementing a test-and-set object that depends only on N is known. Our lower bound is no exception: It assumes bounded *value contention* Q, defined as follows.

▶ **Definition 6** (Bounded Value Contention). *An algorithm has* bounded value contention Q *if, at all points during an execution, processes can be poised to write at most Q different values to the same register.*

Discussion. We note that bounded value contention is strictly more general than both write contention and register size. In particular, bounded write contention κ implies value contention at most κ. However, the reverse does not hold because we can allow arbitrary many processes be simultaneously poised to write *the same value* to a register, without exceeding the value contention bound. Likewise, an upper bound, C, on the number of different values that can be written to a register (which is equivalent to a bound on register size) implies value contention at most C. Again, the reverse does not hold, since arbitrarily many different values can be poised to be written to a register in *different* executions, or at *different* configurations during the same execution without exceeding the value contention bound.

To see the intuition why we need this additional restriction on the algorithm to obtain a non-trivial lower bound, consider the following pseudocode:

```
door := {id}
if door != id: // Fast path check
  return Lose // Observed another process, so can lose
// Otherwise perform slow path Leader Election
```

Here, each process first writes its id to a shared register, `door`, and then reads this register. If it reads the id of another process, it immediately loses. Observe that we can add this code at the beginning of any Leader Election algorithm, since the last process to write to `door` will continue to the next part of the algorithm, if it does not crash. The previous step complexity lower bound proofs construct a worst-case execution by repeatedly selecting a set of processes that each perform one more step, one after the other [5, 10, 7]. However, in an execution that begins with a round in which every process takes one step, there is only one process that does not lose when it performs its second step. Consequently, this process will execute the next part of the algorithm by itself. For some algorithms, for example, one that uses a Splitter, this process will also terminate within a constant number of steps. However, in an execution in which no process writes to `door` between the write to `door` and subsequent read of `door` by the same process, all non-faulty processes reach the next part of the algorithm. Our bounded value contention does not allow all processes to be simultaneously poised to write their ids to the same register, so such an algorithm does not contradict our lower bound.

The main result of this section is the following:

▶ **Theorem 7.** *For any obstruction-free Leader Election algorithm for N processes with value contention Q, there exists a solo execution of length $\Omega\left(\frac{\log N}{\log\log N+\log Q}\right)$.*

Proof. We begin by giving an overview of the proof approach. We will construct a round-by-round execution, one round at a time. In each round, we have a set of active processes that each takes one additional step. We maintain the invariant that the processes do not see one another. In other words, for each active process, the execution is indistinguishable from a solo execution. Initially, all processes are active and the execution is empty.

To add an additional round, we consider the next step of all active processes and partition them into groups depending on what register they access, whether they read or write, and what value they read or what value they write. After that, we construct a conflict graph, where the nodes are the active processes and there is an edge between two processes if one of them sees the other in any extension of the current execution in which each active process takes one more step or if they write different values to the same register. We observe that this graph can be represented by a faction graph. Using Theorem 5, we select a sufficiently large independent set of active processes in the conflict graph. All other processes are erased from the execution and then each process in the independent set takes its next step. Similarly to Anderson and Kim [7], this ensures that each process only reads the initial value of a register or the last value that it wrote there. The construction continues until only one process remains.

Let $P_t \subseteq P$ be the set of active processes that participate in the current execution, $\mathcal{E}_t$. In this execution, no process in P_t sees any other process, so, for all processes $p, q \in P_t$, $p \sim_{\mathcal{E}_t} q$ if and only if $p = q$. By Lemma 2, there is at most one process that has terminated at the end of $\mathcal{E}_t$. Each process in P_t that has not terminated has taken exactly t steps in $\mathcal{E}_t$. We also ensure that no two processes write different values to the same register in their i'th step in $\mathcal{E}_t$, for $1 \leq i \leq t$. Initially, $\mathcal{E}_0$ is empty and $P_0 = P$.

To create $\mathcal{E}_{t+1}$, we begin by partitioning the processes in P_t into groups, where two processes are in the same group if, at the end of $\mathcal{E}_t$,

- they are poised to write the same value into the same register,
- they are poised to read the same register and they have never written to that register, or
- they are poised to read the same register and they last wrote the same value to that register.

If there is a process in P_t that is terminated at the end of $\mathcal{E}_t$, it is put in a group by itself. For each register, there are at most Q groups of processes writing different values to the register, there is at most one group of processes that is reading the register, but have never written to it, and there are at most t groups of processes that are reading the register and last wrote to it in one of the previous t rounds.

We need to show that we can choose sufficiently many active processes $P_{t+1} \subseteq P_t$, which each performs one more step, without letting it see any other process. Other active processes will be erased from the execution when creating $\mathcal{E}_{t+1}$. (Note that one of the processes in P_{t+1} may have terminated.) To determine such a set of active processes, we construct a conflict graph. This is a graph with vertex set P_t, where there is an edge between two processes if one of them is poised to read a register from which it could see the other process or both are poised to write different values to the same register. The conflict graph can be represented by a faction graph, where the factions are the groups of processes. We will describe only the set of super-edges, E^*, of this faction graph from which the set of edges of the conflict graph can be deduced. Intuitively, we want to find an independent set in the conflict graph, remove all steps by other processes in $\mathcal{E}_t$, and add a round to the resulting execution to create $\mathcal{E}_{t+1}$.

There are two rules used to construct the super-edges of the faction graph:

`Rule 1.` **Poised to read after a past write.** Consider some process, p, that wrote a value to register r during $\mathcal{E}_t$. Let v be the value that p last wrote to r. Let A be a group of processes that are poised to read register r and last wrote the value $v' \neq v$ to register r or a group of processes that are poised to read register r, but have never written to register r, and $v' \neq v$ is the initial value of register r. In a solo execution, each process in A would read the value v' when it performs this read. Then $(p, A) \in E^*$.

`Rule 2.` **Poised to write different values.** Consider a group of processes, A, poised to write value v to a register r and a group of processes, B, poised to write value $v' \neq v$ to the register r. Then, $(p, B) \in E^*$, for all processes $p \in A$, and $(q, A) \in E^*$, for all processes $q \in B$.

Since all active processes take t steps during $\mathcal{E}_t$, each process p writes to at most t different registers. For each register, r, to which it wrote, $(p, A) \in E^*$ by Rule 1, only if the processes in group A are poised to read r. At most one value is written to r in each round of $\mathcal{E}_t$, so there are at most $t - 1$ groups of processes poised to read r that last wrote a different value to r than p did. There is at most one group of processes poised to read r that have never written to r. Hence, there are at most $t \cdot t = t^2$ groups A for which $(p, A) \in E^*$ by Rule 1.

Suppose $(p, A) \in E^*$ by Rule 2. Then p is poised to write a value v to a register r at the end of $\mathcal{E}_t$ and the processes in A are poised to write a different value to register r at the end of $\mathcal{E}_t$. Since the value contention of the algorithm is at most Q, there are at most $Q - 1$ different groups that are poised to write values other than v to register r at the end of $\mathcal{E}_t$. Hence, there are at most $Q - 1$ groups A for which $(p, A) \in E^*$ by Rule 2.

In total, $|E^*| \leq |P_t| \cdot (t^2 + Q - 1)$. Let P_{t+1} be a maximum independent set in the conflict graph. By Theorem 5,

$$|P_{t+1}| \geq \frac{1}{e} \cdot \frac{|P_t|^2}{|P_t| + |E^*|} \geq \frac{1}{e} \cdot \frac{|P_t|^2}{|P_t| + |P_t| \cdot (t^2 + Q - 1)} = \frac{1}{e} \cdot \frac{|P_t|}{Q + t^2}.$$

Since $P_t \setminus P_{t+1}$ is closed with respect to $\sim_{\mathcal{E}_t}$, removing the steps of all processes in $P_t \setminus P_{t+1}$ from $\mathcal{E}_t$ results in an execution. Let $\mathcal{E}_{t+1}$ be obtained from this execution by appending a round in which all processes in P_{t+1} that are poised to read each take one step and then all processes in P_{t+1} that are poised to write each take one step. Note that, by Rule 2, at most one value is written to each register during the last round of $\mathcal{E}_{t+1}$. Since at most one value is written to each register during each round of $\mathcal{E}_t$, the same is true for $\mathcal{E}_{t+1}$. Furthermore, since each process in P_t that has not terminated takes exactly t steps during $\mathcal{E}_t$, each process in P_{t+1} that has not terminated takes exactly $t + 1$ steps during $\mathcal{E}_{t+1}$.

Suppose there is a process, $q \in P_{t+1}$, in some group, A, that sees another process during $\mathcal{E}_{t+1}$. Since q sees no other processes during $\mathcal{E}_t$, it must be that, during round $t + 1$, q read a value v from register r and, during $\mathcal{E}_t$, either q did not write to r, or q last wrote some value $v' \neq v$ to r. By definition of the groups, the same is true for all processes in group A. Note that all writes to r during round $t + 1$ occur after q read r. Let $p \in P_{t+1}$ be any process that last wrote v to r during $\mathcal{E}_t$. Note that $p \notin A$. By Rule 1, $(p, A) \in E^*$ is an edge in the faction graph. This implies that $\{p, q\}$ is an edge in the conflict graph, which contradicts the fact that P_{t+1} is an independent set. Hence, no process in P_{t+1} sees another process during $\mathcal{E}_{t+1}$.

Thus, the execution $\mathcal{E}_{t+1}$ has all the required properties. We repeatedly construct additional rounds until the remaining set of active processes has size 1. Note that, if $|P_t| > 1$, it is always possible to choose P_{t+1} so that it contains a process that takes $t + 1$ steps during $\mathcal{E}_{t+1}$. Let T be the first number such that $|P_T| = 1$. Then $\mathcal{E}_T$ is a solo execution of length T.

It remains to show that $T \in \Omega\left(\frac{\log N}{\log Q + \log\log N}\right)$. Recall that $|P_{t+1}| \geq \frac{1}{e} \cdot \frac{|P_t|}{Q+t^2}$, so

$$|P_T| \geq \frac{|P_0|}{e^T \prod_{t=0}^{T-1}(Q+t^2)}.$$

Rearranging and using the facts that $|P_0| = N$ and $|P_T| = 1$, we obtain

$$N \leq e^T \prod_{t=0}^{T-1}(Q+t^2) \leq e^T \prod_{t=0}^{T-1}(Q+T^2) = (e(Q+T^2))^T.$$

Hence,

$$T \geq \frac{\log N}{\log(e(Q+T^2))} = \frac{\log N}{\log e + \log(Q+T^2)}.$$

Let $N \geq 4$. Then $\log Q + \log\log N \geq 1$, since $Q \geq 1$. If $T \geq \log N$, then $T \geq \frac{\log N}{\log Q + \log\log N}$. So, assume that $T < \log N$. In this case,

$$T > \frac{\log N}{\log e + \log(Q + (\log N)^2)}.$$

Since $Q, \log N \geq 1$, we have $Q + (\log N)^2) \leq 2Q(\log N)^2$, so $\log e + \log(Q + (\log N)^2) \leq \log e + \log(2Q(\log N)^2) = 1 + \log e + \log Q + 2\log\log N \in O(\log Q + \log\log N)$. Therefore, $T \in \Omega\left(\frac{\log N}{\log Q + \log\log N}\right)$. ◀

This theorem leads to some additional results. First, we observe that our lower bound argument can be applied directly to the more general class of *non-deterministic solo-terminating* algorithms [11], which includes randomized wait-free algorithms for the Leader Election problem.

▶ **Corollary 8.** *Against an adaptive adversary, every randomized wait-free Leader Election algorithm for N processes with value contention Q has $\Omega\left(\frac{\log N}{\log\log N + \log Q}\right)$ worst-case expected step complexity and solo step complexity.*

By specializing the parametrization to focus on particular ranges of interest, we obtain the following two results.

▶ **Corollary 9.** *A Leader Election algorithm with constant solo step complexity is only possible if the number of different values a register can hold is polynomial in N and, hence, only if the number of different processes that can be poised simultaneously poised to access the same register can be polynomial in N.*

▶ **Corollary 10.** *Any (randomized) Leader Election algorithm that has $o\left(\frac{\log N}{\log\log N}\right)$ solo step complexity (with high probability) has value contention that is super-polylogarithmic in N.*

We can also apply our theorem to adaptive Leader Election algorithms, where the step complexity depends on the number of processes that take at least one step, rather than on the total number of processes in the system.

▶ **Corollary 11.** *For value contention $Q = N^{o(1)}$, where $c > 0$ is constant, it is impossible to solve Leader Election with expected solo step complexity (and, hence, with worst-case expected step complexity) $O(f(k))$, where k is the number of participating processes and f is an arbitrary function.*

5 Upper Bounds

We discuss how existing upper bounds relate to our lower bounds. The related algorithms are listed in Table 2.

Table 2 Upper bounds for Leader Election against a strong adversary.

Step Compl.	Total Step Compl.	Q	Reference	Comments
$O(\log N)$	$O(k \log N)$	constant	[1]	with high probability
$O\left(\log_\kappa N\right)$	$O\left(k \log_\kappa N\right)$	κ	[5]	Solves *Weak* Leader Election, κ is the write contention

Leader Election with One-Bit Registers

Afek, Gafni, Tromp and Vitányi presented a Leader Election algorithm for N processes with $O(\log N)$ step complexity with high probability based on a tournament tree [1] using Leader Election for 2 processes at each internal node. Leader Election for 2 processes can be solved by the randomized algorithm of Tromp and Vitányi in expected constant step complexity using 4-valued shared registers [26]. We can then replace each 4-valued register with 4 one-bit registers using the construction of [20]. Thus, the value contention, Q, of the resulting algorithm is 2. The step complexity of this algorithm matches our lower bound to within an $O(\log \log N)$ factor.

Weak Leader Election Based on Splitters

Weak Leader Election differs from Leader Election in that it does not guarantee existence of a winner if there is more than one participating process. Alistarh, Gelashvili and Nadiradze proposed an algorithm for the Weak Leader Election problem with bounded write contention κ [5], which is based on Splitter objects. Their algorithm arranges processes into a complete κ-ary tournament tree, where each node is associated with a Splitter.

Since their tree is κ-ary, the write contention is bounded by κ and, consequently, the value contention is also bounded by κ. The construction in our proof of Theorem 7, ensures that processes do not see other processes, so our lower bound also applies to the Weak Leader Election problem. Therefore, there is an $\Omega\left(\frac{\log N}{\log \log N + \log \kappa}\right)$ lower bound and an almost matching $O\left(\frac{\log N}{\log \kappa}\right)$ upper bound on the step complexity of this problem.

Adaptive RatRace Leader Election

Alistarh, Attiya, Gilbert, Giurgiu, and Guerraoui proposed an algorithm for solving Leader Election, which they call *RatRace* [4], that has $O(\log k)$ step complexity with high probability and $O(N^3)$ space complexity, where k is the number of participating processes.

The algorithm consists of a *primary tree* and a *backup grid*. The primary tree is a complete binary tree of Randomized Splitters of depth $3 \log N$. Every process starts at the root of the primary tree and then either stops or moves to the left or right child depending on the result (`Stop`, `Left` or `Right`) of the corresponding Randomized Splitter. Once a process obtains `Stop`, it tries to move back to the root by participating in an instance of Leader election among 3 processes: the process that received `Stop` from the the Randomized Splitter and one process from each child. The winner of the instance of Leader Election at a node moves

back to its parent, if it has a parent. In the unlikely case when a process reaches a leaf of the tree without obtaining `Stop`, the process accesses the backup grid, which is a $O(N^2)$ grid of deterministic Splitters [4] and instances of Leader Election among 3 processes. The winner of the backup grid and the winner of the instance of Leader Election at the root of the primary tree then participate in a final instance of Leader Election (among 2 processes) to decide the winner of the Leader Election among all N processes. Giakkoupis and Woelfel improved the space complexity of the RatRace algorithm from $O(N^3)$ to $O(N)$ [14] by replacing the backup grid with $\frac{N}{\log N}$ elimination paths of size $4 \log N$ and an elimination path of size N and reducing tree height from $3 \log N$ to $\log N$.

The value contention of a Splitter shared by N processes is N, so the value contention of both these algorithms is N. Corollary 11 proves that adaptive Leader Election is impossible unless Q is polynomial in N. In particular, this means that any adaptive Leader Election algorithm has to use registers of size $\Omega(\log N)$ and it must have configurations in which $N^{\Omega(1)}$ process are poised to write different values to the same register.

6 Discussion

We made a step towards improving Leader Election bounds. Although we did not achieve a matching logarithmic lower bound that depends only on the number of processes, we proved tighter and more general lower bounds than previous work [5] by analyzing a new notion of value contention. Our primary contribution is the establishment of a new lower bound of

$$\Omega\left(\frac{\log N}{\log\log N + \log Q}\right)$$

on the step complexity for any obstruction-free leader election algorithm.

Moreover, we have demonstrated that achieving $o\left(\frac{\log N}{\log\log N}\right)$ step complexity requires value contention to be $\omega(\texttt{polylog}(N))$. On the other hand, we showed that adaptive leader election is impossible unless the value contention is also polynomial in N, i.e. polynomially many processes can be simultaneously poised to write polynomially many different values to the same register. These impossibility results place fundamental limits on the design of efficient leader election algorithms.

Finally, we speculate that our approach stretches the existing approaches to their limit, and that further progress towards a general lower bound will require a more general technique that bounds the information flow achievable in the high-contention scenario where all processors can simultaneously be poised to write different values to the same register.

References

1 Yehuda Afek, Eli Gafni, John Tromp, and Paul M. B. Vitányi. Wait-free test-and-set (extended abstract). In *Proceedings of the 6th International Workshop on Distributed Algorithms (WDAG)*, pages 85–94, 1992. doi:10.1007/3-540-56188-9_6.

2 Marcos Kawazoe Aguilera, Carole Delporte-Gallet, Hugues Fauconnier, and Sam Toueg. Stable leader election (extended abstract). In *Proceedings of the 15th International Symposium on Distributed Computing (DISC)*, volume 2180 of *Lecture Notes in Computer Science*, pages 108–122, 2001.

3 Dan Alistarh and James Aspnes. Sub-logarithmic test-and-set against a weak adversary. In *Proceedings of the 25th International Symposium on Distributed Computing (DISC)*, volume 6950 of *Lecture Notes in Computer Science*, pages 97–109, 2011. doi:10.1007/978-3-642-24100-0_7.

4 Dan Alistarh, Hagit Attiya, Seth Gilbert, Andrei Giurgiu, and Rachid Guerraoui. Fast randomized test-and-set and renaming. In *Proceedings of the 24th international Symposium on Distributed Computing (DISC)*, volume 6343 of *Lecture Notes in Computer Science*, pages 94–108, 2010. doi:10.1007/978-3-642-15763-9_9.

5 Dan Alistarh, Rati Gelashvili, and Giorgi Nadiradze. Lower bounds for shared-memory leader election under bounded write contention. In *Proceedings of the 35th International Symposium on Distributed Computing (DISC)*, volume 209 of *LIPIcs*, pages 4:1–4:17, 2021. doi:10.4230/LIPICS.DISC.2021.4.

6 Dan Alistarh, Rati Gelashvili, and Giorgi Nadiradze. Lower bounds for shared-memory leader election under bounded write contention, 2022. URL: https://arxiv.org/abs/2108.02802, arXiv:2108.02802.

7 James H. Anderson and Yong-Jik Kim. An improved lower bound for the time complexity of mutual exclusion. *Distributed Computing*, 15(4):221–253, 2002. doi:10.1007/S00446-002-0084-2.

8 Hagit Attiya, Fabian Kuhn, C. Greg Plaxton, Mirjam Wattenhofer, and Roger Wattenhofer. Efficient adaptive collect using randomization. *Distributed Computing*, 18(3):179–188, 2006. doi:10.1007/S00446-005-0143-6.

9 Hagit Attiya and Jennifer Welch. *Distributed Computing. Fundamentals, Simulations, and Advanced Topics (second edition)*. Wiley, 2004.

10 Aryaz Eghbali and Philipp Woelfel. An almost tight RMR lower bound for abortable test-and-set. In *Proceedings of the 32nd International Symposium on Distributed Computing (DISC)*, volume 121 of *LIPIcs*, pages 21:1–21:19, 2018. doi:10.4230/LIPICS.DISC.2018.21.

11 Faith Ellen, Rati Gelashvili, and Leqi Zhu. Revisionist simulations: A new approach to proving space lower bounds. In *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing*, PODC '18, pages 61–70, New York, NY, USA, 2018. Association for Computing Machinery. doi:10.1145/3212734.3212749.

12 George Giakkoupis, Maryam Helmi, Lisa Higham, and Philipp Woelfel. An o(sqrt n) space bound for obstruction-free leader election. In *Proceedings of the 27th International Symposium on Distributed Computing (DISC)*, volume 8205 of *Lecture Notes in Computer Science*, pages 46–60, 2013. doi:10.1007/978-3-642-41527-2_4.

13 George Giakkoupis, Maryam Helmi, Lisa Higham, and Philipp Woelfel. Test-and-set in optimal space. In *Proceedings of the 47th Annual ACM Symposium on Theory of Computing (STOC)*, pages 615–623, 2015. doi:10.1145/2746539.2746627.

14 George Giakkoupis and Philipp Woelfel. On the time and space complexity of randomized test-and-set. In *Proceedings of the 31st Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 19–28, 2012. doi:10.1145/2332432.2332436.

15 George Giakkoupis and Philipp Woelfel. Efficient randomized test-and-set implementations. *Distributed Computing*, 32(6):565–586, 2019. doi:10.1007/S00446-019-00349-Z.

16 Wojciech Golab, Danny Hendler, and Philipp Woelfel. An o(1) RMRs leader election algorithm. In *Proceedings of the 25th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 238–247, 2006. doi:10.1145/1146381.1146417.

17 Maurice Herlihy and Mark Tuttle. Wait-free computation in message-passing systems: Preliminary report. In *Proceedings of the 9th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 347–362, 1990.

18 Johan Jensen. Sur les fonctions convexes et les inégalités entre les valeurs moyennes. *Acta Mathematica*, 30(1):175–193, 1906.

19 Yong-Jik Kim and James Anderson. A time complexity bound for adaptive mutual exclusion. *Distributed Computing*, 24(6):271–297, 2012.

20 Leslie Lamport. On interprocess communication. part I: Basic formalism. *Distributed Computing*, 1(2):77–85, 1986. doi:10.1007/BF01786227.

21 Leslie Lamport. A fast mutual exclusion algorithm. *ACM Transactions on Computer Systems (TOCS)*, 5(1):1–11, 1987. doi:10.1145/7351.7352.

22 Nancy A. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.

23 Mark Moir and James H Anderson. Wait-free algorithms for fast, long-lived renaming. *Science of Computer Programming*, 25(1):1–39, 1995. doi:10.1016/0167-6423(95)00009-H.

24 Gary L. Peterson and Michael J. Fischer. Economical solutions for the critical section problem in a distributed system (extended abstract). In *Proceedings of the 9th Annual ACM Symposium on Theory of Computing (STOC)*, pages 91–97, 1977. doi:10.1145/800105.803398.

25 Eugene Styer and Gary L. Peterson. Tight bounds for shared memory symmetric mutual exclusion problems. In *Proceedings of the 8th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 177–191, 1989. doi:10.1145/72981.72993.

26 John Tromp and Paul Vitányi. Randomized two-process wait-free test-and-set. *Distributed Computing*, 15(3):127–135, 2002. doi:10.1007/S004460200071.

27 Paul Turán. On an extremal problem in graph theory. *Matematikai és Fizikai Lapok*, 48:436–452, 1941.

28 Jae-Heon Yang and James H Anderson. Time bounds for mutual exclusion and related problems. In *Proceedings of the 26th Annual ACM symposium on Theory of Computing (STOC)*, pages 224–233, 1994. doi:10.1145/195058.195139.