An O(sqrt(n)) space bound for obstruction-free leader election

George Giakkoupis, Maryam Helmi, Lisa Higham, Philipp Woelfel

An $O(\sqrt{n})$ Space Bound for Obstruction-Free Leader Election

George Giakkoupis[1]*, Maryam Helmi[2], Lisa Higham[2], and Philipp Woelfel[2]**

[1] INRIA Rennes – Bretagne Atlantic, France
george.giakkoupis@inria.fr
[2] Department of Computer Science, University of Calgary, Canada
{mhelmikh,higham,woelfel}@ucalgary.ca

Abstract. We present a deterministic obstruction-free implementation of leader election from $O(\sqrt{n})$ atomic $O(\log n)$-bit registers in the standard asynchronous shared memory system with n processes. We provide also a technique to transform any deterministic obstruction-free algorithm, in which any process can finish if it runs for b steps without interference, into a randomized wait-free algorithm for the oblivious adversary, in which the expected step complexity is polynomial in n and b. This transformation allows us to combine our obstruction-free algorithm with the leader election algorithm by Giakkoupis and Woelfel [21], to obtain a fast randomized leader election (and thus test-and-set) implementation from $O(\sqrt{n})$ $O(\log n)$-bit registers, that has expected step complexity $O(\log^* n)$ against the oblivious adversary.
Our algorithm provides the first sub-linear space upper bound for obstruction-free leader election. A lower bound of $\Omega(\log n)$ has been known since 1989 [29]. Our research is also motivated by the long-standing open problem whether there is an obstruction-free consensus algorithm which uses fewer than n registers.

Keywords: leader election, test-and-set, shared memory model, randomized algorithms, obstruction-free algorithms

1 Introduction

One of the fundamental theoretical questions in shared memory research is whether certain standard primitives can be simulated from other ones (given certain progress conditions), and if yes, how much resources (usually time and space) are necessary for such simulations. Perhaps the best studied problem in this context is that of *consensus*, where each process receives an input and processes have to agree on one of their inputs. Consensus cannot be solved deterministically with wait-free progress in shared memory systems that provide

* This work was funded in part by INRIA Associate Team RADCON and ERC Starting Grant GOSSPLE 204742.
** This research was undertaken, in part, thanks to funding from the Canada Research Chairs program and the HP Labs Innovation Research Program.

only shared atomic registers [20]. The study of which primitives can be used to solve consensus deterministically in systems with a certain number of processes has led to Herlihy's famous wait-free hierarchy [24]. Randomized algorithms can solve consensus and guarantee randomized wait-freedom even if only registers are available. The randomized step complexity of the consensus problem has been studied thoroughly and is well understood for most of the common adversary models [8–13].

On the other hand, it is still open how many registers are needed in a system with n processes to have a randomized wait-free implementation of consensus, or even an obstruction-free one. Fich, Herlihy and Shavit [18] showed that at least $\Omega(\sqrt{n})$ registers are necessary, but no obstruction-free algorithm that uses fewer than n registers is known. (The lower bound holds in fact even for the weaker progress condition of nondeterministic solo termination, and for implementations from any historyless base objects.) The space complexity of other fundamental primitives has also been investigated, e.g., the implementation of timestamp objects from registers and historyless objects [17, 23], or that of a wide class of strong primitives called perturbable objects such as counters, fetch-and-add and compare-and-swap from historyless objects [25].

In this paper we consider *leader election*, another fundamental and well-studied problem, which is related to consensus but is seemingly much simpler. In a leader election protocol for n processes, each process has to decide on one value, `win` or `lose`, such that exactly one process (the leader) wins. The problem is related to *name consensus*, where processes have to agree on the ID of a leader—whereas in leader election each process only has to decide whether it is the leader or not. Leader election is also closely related to, and in most models equally powerful as, the *test-and-set (TAS)* synchronization primitive. TAS is perhaps the simplest standard shared memory primitive that has no wait-free deterministic implementation from registers. A TAS object stores one bit, which is initially 0, and supports a `TAS()` operation which sets the bit's value to 1 and returns its previous value. It has consensus number two, so it can be used together with registers to solve deterministic wait-free consensus only in systems with two processes. TAS objects have been used to solve many classical problems such as mutual exclusion and renaming [4–6, 14, 15, 26, 28]. Processes can solve leader election using one TAS object by simply calling `TAS()` once, and returning `win` if the `TAS()` call returned 0, or `lose` otherwise. On the other hand a very simple algorithm using a leader election protocol and one additional binary register can be used to implement a linearizable TAS object, where for a `TAS()` operation each process needs to execute only a constant number of operations in addition to the leader election protocol [22].

Significant progress has been made in understanding the step complexity of randomized leader election [2, 3, 6, 21, 30]. In particular, in the oblivious adversary model (where the order in which processes take steps is independent of random decisions made by processes), the most efficient algorithm guarantees that the expected step complexity (i.e., the expected maximum number of steps executed by any process) is $O(\log^* k)$, where k is the contention [21].

Little is known, however, about the space complexity of randomized wait-free or obstruction-free leader election. For the much weaker progress condition of deadlock freedom, it is known that the space complexity of leader election is $\Theta(\log n)$ [29]. Clearly, this implies also a space lower bound of $\Omega(\log n)$ for randomized wait-free and obstruction-free leader election. Still, prior to our work presented here, no obstruction-free (or even nondeterministic solo terminating) algorithm was known for solving leader election with fewer than n registers.

We devise the first deterministic obstruction-free algorithm for leader election (and thus for TAS) which uses only $O(\sqrt{n})$ registers. The algorithm is simple but elegant.

Theorem 1. *There is an obstruction-free implementation of a leader election object for n processes from $\sqrt{2n} + o(\sqrt{n})$ atomic $O(\log n)$-bit registers.*

This result raises the question whether it is also possible to obtain a fast randomized wait-free algorithm for leader election. The relation between wait-freedom and obstruction-freedom has been investigated before: Fich, Luchangco, Moir, and Shavit [19] showed that obstruction-free algorithms can be transformed to wait-free ones in the unknown-bound semi-synchronous model.

In this paper we follow a different approach, as we use randomization, but stay in the fully asynchronous model. It is easy to see that any deterministic obstruction-free algorithm can be transformed into an algorithm which is randomized wait-free against the oblivious adversary: Whenever a process is about to perform a shared memory step in the algorithm, it can flip a coin, and with probability 1/2 it performs the step of the algorithm (called "actual" step), while with the remaining probability it executes a "dummy" step, e.g., reads an arbitrary registers. Suppose a process is guaranteed to finish the obstruction-free algorithm if it performs b unobstructed steps. Any execution of length bn (i.e., where exactly bn shared memory steps are performed) must contain a process that executes at least b steps, and with probability at least $1/2^{bn}$ that process executes b actual steps while all other processes execute just dummy steps. Then during an execution of length $bn \cdot 2^{bn} \cdot (\log n + c)$ some process runs unobstructed for at least b actual steps with probability $1 - O(1/2^c)$. Hence, the algorithm is randomized wait-free.

This naive transformation yields exponential expected step complexity. We provide a slightly different but also simple transformation (which requires a more sophisticated analysis) to show the following result.

Theorem 2. *Suppose there is a deterministic obstruction-free algorithm which guarantees that any process finishes after it has executed at most b steps without interference from other processes. Then the algorithm can be transformed into a randomized one that has the same space complexity, and for any fixed schedule (determined by an oblivious adversary) each process returns after at most $O\big(b(n+b)\log(n/\delta)\big)$ of its own steps with probability at least $1-\delta$, for any $\delta > 0$ that can be a function of n.*

As mentioned above, Giakkoupis and Woelfel [21] have recently presented a randomized leader election algorithm which has $O(\log^* k)$ expected step com-

plexity against the oblivious adversary, where k is the contention. The algorithm requires $\Theta(n)$ registers, but with high probability (*w.h.p.*) processes access only the first poly-logarithmic number of them. The idea is now to reduce the space requirements of this algorithm by removing the registers which are not needed in most executions, and then in the unlikely event that processes run out of registers, they switch to the algorithm obtained by applying the transformation of Theorem 2 to the algorithm from Theorem 1.

Theorem 3. *There is a randomized implementation of leader election from $\sqrt{2n} + o(\sqrt{n})$ atomic $O(\log n)$-bit registers which guarantees that for any fixed schedule (determined by an oblivious adversary), the maximum number of steps executed by any process is $O(\log^* k)$ in expectation and $O(\log n)$ w.h.p., where k is the contention.*

Model and Preliminaries

We consider the standard asynchronous shared memory model where up to n processes communicate by reading and writing to shared atomic multi-reader multi-writer $O(\log n)$-bit registers. Processes may fail by crashing at any time.

An algorithm may be deterministic or randomized. If it is randomized, then processes can use local coin-flips to make random decisions. For randomized algorithms, the scheduling and process crashes are controlled by an adversary, which at any point of an execution decides which process will take the next step. In this paper we only deal with the *oblivious adversary*, which determines the entire (infinite) schedule ahead of time, i.e., before the first process takes a step.

A deterministic algorithm is *wait-free* if every process finishes in a finite number of its own steps. It is *obstruction-free*, if it guarantees that any process will finish if it performs enough steps alone (i.e, without interference from other processes). If the algorithm is randomized, and every process finishes in an *expected* finite number of steps, then the algorithm is *randomized wait-free* [24].

Our algorithms use an obstruction-free and linearizable `scan()` operation, which returns a view of an M-element array R (a view is the vector of all array entry values). Implementations of M-component *snapshot* objects which provide M-element arrays supporting linearizable `scan()` operations are well-known [1, 7, 16]. But in order to achieve our space upper bounds we need a snapshot implementation that uses only $O(M)$ bounded registers. The wait-free implementation by Fich, Fatourou, and Ruppert [16] has space-complexity $O(M)$ but uses unbounded registers. In Appendix A we present a linearizable obstruction-free implementation of a linearizable snapshot object from $M + 1$ $O(\log n)$-bit registers, where each `scan()` operation finishes after $O(M)$ unobstructed steps.

2 Obstruction-Free Leader Election

We present an obstruction-free implementation of leader election from $O(\sqrt{n})$ registers. The algorithm proceeds in phases, during which processes have access

Algorithm 1: Pseudocode for process p.

```
   /* Let m = √(2n) + c·∜n, where c > 0 is a suitable constant          */
   shared: array R[0...m] of pairs (process_ID, phase_number) initialized to (0,0)
 1 φ ← 1                                          /* p's current phase number */
 2 while φ ≤ m do
 3 |   r[0...m] ← R.scan()
 4 |   if ∃ i, p', φ' > φ: r[i] = (p', φ')
 5 |         or ( |{j: r[j] = (p, φ)}| ≤ 1 and ∃ q |{j: r[j] = (q, φ)}| ≥ 2 ) then
 6 |   |   return lose
 7 |   else if r[0...φ] = [(p, φ)...(p, φ)] then
 8 |   |   φ ← φ + 1                              /* proceed to the next phase */
 9 |   else
10 |   |   Let i be the smallest index such that r[i] ≠ (p, φ).          /* i ≤ φ */
11 |   |   R[i] ← (p, φ)
12 |   end
13 end
14 return win
```

to a shared array $R[0 \dots m]$ of registers, where $m = \sqrt{2n} + o(\sqrt{n})$. Each register of R stores a pair (process ID, phase number). In phase $1 \leq \phi \leq m$, process p tries to write value (p, ϕ) on all registers $R[0 \dots \phi]$. After each write, p obtains a view r of array R using a `scan()`.

Process p loses if one of the two happens: (i) some entry of view r contains a phase number larger than p's phase ϕ; or (ii) two (or more) entries of r have the same value (q, ϕ) for some $q \neq p$, while at most one entry has value (p, ϕ). If neither (i) nor (ii) happens, then p picks the smallest i such that $i \leq \phi$ for which $r[i] \neq (p, \phi)$ and writes (p, ϕ) to $R[i]$. If no such i exists, i.e., all entries of $r[0 \dots \phi]$ are equal to (p, ϕ), then p enters the next phase, $\phi + 1$. A process wins when it reaches phase $m + 1$. Pseudocode is given in Algorithm 1.

The above algorithm is not wait-free: First of all, our `scan()` operation is only obstruction-free. But even if we used wait-free snapshot objects, no process may finish the algorithm for certain schedules. E.g., suppose two processes alternate in executing the while-loop of Algorithm 1 (and each of them executes the entire loop without obstruction). Then whenever one of them scans R in line 3, $R[0]$ does not contain that process' ID, so the process remains in phase 1 and writes to $R[0]$ in line 11. We show below that our algorithm is obstruction-free.

The proof of Theorem 1 unfolds in a series of lemmas. First we show that not all processes lose; thus at least one process wins or does not finish. Then we argue that a process finishes if it runs without interruption for long enough. Last, we show that no two processes win.

To distinguish between the local variables of different processes, we may explicitly state the process as a subscript, e.g., ϕ_p.

Lemma 1. *There is no execution in which all participating processes lose.*

Proof. Suppose, towards a contradiction, that there is some non-empty execution in which all participating processes lose. Let $\phi_{\max}$ be the largest phase in which any process participates in this execution. Clearly $\phi_{\max} \le m$, because if for some process p we have $\phi_p = m+1$ then ϕ_p must have increased from m to $m+1$ in line 8, and after that p cannot lose as it does not do another iteration of the while-loop. Among all processes participating in phase $\phi_{\max}$, consider the last process p that executes a `scan()` in line 3, i.e., the linearization point of the `scan()` by p is after the corresponding linearization points of the `scan()` operations by any other process participating in phase $\phi_{\max}$. After p has executed line 3 for the last time, r_p must satisfy the condition of the if-statement in the next line (otherwise p does not lose), i.e., either (i) $\exists\, i, p', \phi' > \phi_{\max} \colon r[i] = (p', \phi')$, or (ii) we have

$$|\{j \colon r_p[j] = (p, \phi_{\max})\}| \le 1 \ \wedge \ \exists\, q \ |\{j \colon r_p[j] = (q, \phi_{\max})\}| \ge 2. \tag{1}$$

By $\phi_{\max}$'s definition, condition (i) does not hold; hence, condition (ii) holds. Consider now a process q that realizes this condition, and consider the last `scan()` by that process. Then by the same argument as for p, after this `scan()` we have that r_q satisfies

$$|\{j \colon r_q[j] = (q, \phi_{\max})\}| \le 1 \ \wedge \ \exists\, w \ |\{j \colon r_q[j] = (w, \phi_{\max})\}| \ge 2. \tag{2}$$

Since q does not execute any write to R after its `scan()`, and since we have assumed that the last `scan()` by p linearizes after the `scan()` by q, it follows that $\{j \colon r_p[j] = (q, \phi_{\max})\} \subseteq \{j \colon r_q[j] = (q, \phi_{\max})\}$. However, the cardinality of the set to the left is at least 2 by (1), and the cardinality of the set to the right is at most 1 by (2). We have thus reached the desired contradiction. □

Lemma 2. *For any reachable configuration C, an execution started at C in which just a single process p takes steps finishes after at most $O(n^{3/2})$ steps.*

Proof. The step complexity of the execution is dominated by the step complexity of the `scan()` operations by p, in line 3. Each of these operations is completed in $O(m)$ steps, as p runs solo. Further, for each phase ϕ in which p participates it performs (at most) $\phi + 1$ iterations of the while-loop, until it overwrites all entries of $R[0 \dots \phi]$ by (p, ϕ), in line 11. It follows that p finishes after a number of steps bounded by $O\big(\sum_{1 \le \phi \le m} \phi m\big) = O(m^3) = O(n^{3/2})$. □

Lemma 3. *There is a constant $c > 0$ such that if $m \ge \sqrt{2n} + c\sqrt[4]{n}$, then in any execution at most one process wins.*

Proof. For each $1 \le \phi \le m+1$, let N_ϕ be the set of processes that participate in phase ϕ and let $n_\phi = |N_\phi|$. To simplify notation, we assume that there is also phase 0, in which all n processes participate by default, and phases $\phi > m+1$ in which no process participates; we extend the definitions of N_ϕ and n_ϕ to those dummy phases as well. Clearly, the sequence of n_ϕ, $\phi \ge 0$, is non-increasing. Below we analyze how fast this sequence decreases.

We show that the number k of phases after phase ϕ, until at most k processes are left is bounded by $\sqrt{2n_\phi} + O\left(\sqrt[4]{n_\phi}\right)$, and at most $n_\phi + 1$ phases are needed after ϕ to be left with a single process. Formally, for any $0 \le \phi \le m$ we show

(a) $\min\{k\colon n_{\phi+k} \leq k\} \leq \sqrt{2n_\phi} + O\left(\sqrt[4]{n_\phi}\right)$; and
(b) $\min\{k\colon n_{\phi+k} \leq 1\} \leq n_\phi + 1$.

From this claim, Lemma 3 follows easily: For $\phi = 0$ it follows from (a) that $n_{k_1} \leq k_1$ for some $k_1 \leq \sqrt{2n_0} + O(\sqrt[4]{n_0}) = \sqrt{2n} + O(\sqrt[4]{n})$. Applying (a) again, for $\phi = k_1$, yields $n_{k_1+k_2} \leq k_2$ for some $k_2 \leq \sqrt{2n_{k_1}} + O(\sqrt[4]{n_{k_1}}) \leq \sqrt{2k_1} + O(\sqrt[4]{k_1}) = O(\sqrt[4]{n})$. Finally, for $\phi = k_1 + k_2$, we obtain from (b) that $n_{k_1+k_2+k_2} \leq 1$ for some $k_3 \leq k_2 + 1$. Therefore, $n_\phi \leq 1$ for $\phi = k_1 + k_2 + k_3 \leq \sqrt{2n} + O(\sqrt[4]{n})$.

It remains to prove (a) and (b). We start with the proof of (b), which is more basic. Suppose that $n_\phi = \ell > 1$. We must show that $n_{\phi+\ell+1} \leq 1$. Assume, for the sake of contradiction, that $n_{\phi+\ell+1} \geq 2$, and let p be the first process to enter phase $\phi + \ell + 1$, i.e., p's last `scan()` operation in phase $\phi + \ell$ precedes the corresponding operations of other processes from $N_{\phi+\ell+1}$. This `scan()` returns a view r of R in which all entries of $r[0 \ldots \phi+\ell]$ have value $(p, \phi+\ell)$. We claim that after this happens no other process can enter phase $\phi + \ell + 1$, thus contradicting the assumption that $n_{\phi+\ell+1} \geq 2$. Observe that each process writes to R at most once before it executes a `scan()` on R. Further at most $n_\phi - 1 = \ell - 1$ processes $q \neq p$ can write to $R[\phi \ldots \phi + \ell]$. Thus, if any such process q executes a `scan()` on R, it will find at least two entries with values (p, k), for $k \geq \phi + \ell$, and at most one entry $(q, \phi + \ell)$, and thus q will lose.

Next we prove (a). We proceed as follows. For a phase $i = \phi+k$, if $n_i - n_{i+1} < k$ (i.e., fewer than k of the processes participating in phase i fail to enter the next phase, $i + 1$), we argue that during the time interval in which the last $d_i = k - (n_i - n_{i+1})$ processes enter phase $i+1$, at least some minimum number of processes from N_ϕ perform their "final" write operation to R. We show that this minimum number of processes is at least $d_i(d_i - 1)/2$ if $n_i \geq k$, and observe that the total number of such processes for all $i \geq \phi$ is bounded by n_ϕ. Further, we have that the sum of the differences $k - d_i$ is also bounded by n_ϕ. Combining these two inequalities yields the claim.

We give now the detailed proof. Consider a phase $i \geq \phi$, let $k = i - \phi$, and suppose that $n_i \geq k$. Let $d_i = \max\{0, k - (n_i - n_{i+1})\}$. Suppose that $d_i \geq 2$, and consider the last d_i processes from N_i to enter phase $i + 1$. Let t_i be the time when the first of these d_i processes enters phase $i + 1$, and t'_i be the time when the last one does. We argue now that at least $\sum_{1 \leq j < d_i} j = d_i(d_i - 1)/2$ processes from N_ϕ perform their last write operation between times t_i and t'_i. First note that no process enters a phase other than $i+1$ between times t_i and t'_i. Suppose now that the j-th of the d_i processes has just entered phase $i+1$, where $1 \leq j < d_i$, and let p be that process. Then, right after p's `scan()` we have that all entries in $R[0 \ldots i]$ are equal to (p, i). Unless all but at most one of the entries $R[\phi \ldots i]$ are subsequently overwritten by processes in phase i or smaller, no other process can enter phase $i+1$. Since p is not the last process to enter phase $i+1$, and the number of processes left in phase i is $(d_i - j) + (n_i - n_{i+1}) = k - j$, it follows that to overwrite $i - \phi = k$ of the $k + 1$ entries of $R[\phi \ldots i]$ at least $k - (k - j) = j$ of them must be overwritten by processes that are in phases smaller than i; this will be the final write for those processes. It follows that at least $\sum_{1 \leq j < d_i} j = d_i(d_i - 1)/2$ processes from N_ϕ perform their last write

operation between t_i and t'_i, as desired. Note that this result holds also when $d_i < 2$, as in this case the above sum is 0. Observe now that for two distinct i with $d_i \geq 2$ the intervals $[t_i, t'_i]$ do not overlap, and thus the sets of processes that do their final write to each of these intervals are distinct. It follows that the total number of processes from N_ϕ that do a final write in the execution is at least $\sum_{\phi \leq i < \kappa} d_i(d_i - 1)/2$, where $\kappa = \min\{k \colon n_{\phi+k} \leq k\}$. Since this number cannot exceed the size of N_ϕ, we have

$$n_\phi \geq \sum_{\phi \leq i < \phi+\kappa} d_i(d_i - 1)/2. \tag{3}$$

In addition to the above inequality, we have that

$$\begin{aligned} n_\phi = \sum_{\phi \leq i < \phi+\kappa} (n_i - n_{i+1}) + n_{\phi+\kappa} &\geq \sum_{\phi \leq i < \phi+\kappa} (n_i - n_{i+1}) \\ &\geq \sum_{\phi \leq i < \phi+\kappa} (i - \phi - d_i) = \kappa(\kappa-1)/2 - \sum_{\phi \leq i < \phi+\kappa} d_i. \end{aligned} \tag{4}$$

(The second inequality follows from the definition of d_i.)

We now combine the two inequalities above to bound κ. Let $\lambda = \sum_{\phi \leq i < \phi+\kappa} d_i$. Then (3) gives

$$n_\phi \geq \sum_{\phi \leq i < \phi+\kappa} d_i^2/2 - \lambda/2 \geq \lambda^2/(2\kappa) - \lambda/2 \quad \text{(by Cauchy-Schwarz Inequality)}.$$

Solving for λ gives $\lambda \leq (\kappa + \sqrt{\kappa^2 + 8\kappa n_\phi})/2$. Applying this bound of $\lambda = \sum_{\phi \leq i < \phi+\kappa} d_i$ to (4) and rearranging gives $\kappa^2 \leq 2n_\phi + 2\kappa + \sqrt{\kappa^2 + 8\kappa n_\phi}$. Solving for κ yields $\kappa \leq \sqrt{2n_\phi} + O\big(\sqrt[4]{n_\phi}\big)$. This completes the proof of (a) and the proof of Lemma 3. □

Lemmas 1-3 imply that Algorithm 1 is a correct obstruction-free leader election algorithm using $2\sqrt{n} + o(\sqrt{n})$ registers, proving Theorem 1.

Remark 1. We can use an early termination criterion, in which p exits the while-loop (and wins) if the condition of line 7 is satisfied and, in addition, p has seen no process other than itself during phases ϕ and $\phi - 1$: Since p does not see another process during phase $\phi - 1$, it follows that no process finishes phase $\phi - 1$ before p. And since p does not see any process during phase ϕ either, it follows that no process finishes phase $\phi - 1$ before p finishes phase ϕ. Thus, no process other than p ever completes phase ϕ. The detailed argument is straightforward and is omitted due to space constraints. Applying this early termination criterion achieves that each process p finishes after $O(n)$ instead of $O(n^{3/2})$ solo steps.

3 Obstruction Freedom vs. Randomized Wait Freedom

We present now a simple technique that transforms any deterministic obstruction-free algorithm into a randomized one that has the same space complexity and is randomized wait-free against the oblivious adversary. In particular, if the deterministic implementation guarantees that any process finishes

after executing at most b steps without interference, then the randomized implementation guarantees that any process finishes w.h.p. after a number of steps that is bounded by a polynomial function of n and b, namely, $O(b(n+b)\log n)$.

We apply the above transformation to Algorithm 1 presented in Section 2, to obtained a randomized implementation for leader election that has the same $O(\sqrt{n})$ space complexity, and polynomial step complexity against the adaptive adversary. Then we explain how this randomized implementation can be combined with known faster randomized leader election implementation to achieve simultaneously both space- and time-efficiency.

Next we describe the simple transformation technique. Suppose we are given a deterministic obstruction-free algorithm which guarantees that any participating processes p finishes its execution after it takes a sequence of at most b steps during which no other process takes steps. (E.g., from Lemma 2, we have that $b = O(n^{3/2})$ for Algorithm 1.) The randomized implementation we propose is as follows. Every process p flips a biased coin before its first step, and also again every b steps. Each coin flip returns heads with probability $1/n$ and tails with probability $1 - 1/n$, independently of other coin flips. If the outcome of a coin flip by p is heads, then in the next b steps following the coin flip, p executes the next b steps of the given deterministic algorithm; if the outcome is tails then the next b steps of p are *dummy* steps, e.g., p repeatedly reads some shared register.

Analysis. We show that the requirements of Theorem 2 are met by the randomized implementation described above, i.e., a process flips a coin every b steps and with probability $1/n$ it executes the next b steps of the deterministic algorithm, while with probability $1 - 1/n$ it takes b dummy steps instead.

Let $\sigma = (\pi_1, \pi_2, \ldots)$ be an arbitrary schedule determining an order in which processes take steps. We assume that σ is fixed before the execution of the algorithm, and in particular before processes flip their coins. For technical reasons we assume that after a process finishes it does not stop, but it takes *no-op* steps whenever it is its turn to take a step according to σ. Also the process continues to flip a coin every b steps; the outcome of this coin flip has no effect on the execution, but is used for the analysis.

We start with a rough sketch of the proof. We sort processes participating in schedule σ in increasing order in which they are scheduled to take their (λb)-th step, where $\lambda = \Theta\big((n+b)\log(n/\delta)\big)$. Let p_i denote the i-th process in this order. We will argue about p_1 first. We define λ disjoint *blocks* of σ, where the ℓ-th block starts with the first step of p_1 after its ℓ-th coin flip, and finishes after the last step of p_1 before its $(\ell+1)$-th coin flip. Let m_ℓ denote the number of steps contained in block ℓ; then $\sum_\ell m_\ell \le n\lambda b$ by p_1's definition. Further, the number of coin flips that occur in block ℓ is at most $O(m_\ell/b+n)$. These coin flips, plus at most n additional coin flips preceding the block (one by each process), determine which of the steps in the block are actual steps and which ones are dummy. If all these coin flips by processes other than p_1 return tails, we say that the block is *unobstructed*. Such a block does not contain any actual steps by processes other than p_1. It follows that the probability of block ℓ to be unobstructed is at least $(1-1/n)^{O(m_\ell/b+n)}$. The expected number of unobstructed blocks is then

$\sum_\ell (1-1/n)^{O(m_\ell/b+n)}$, and we show that this is $\Omega(\lambda)$ using that $\sum_\ell m_\ell \le n\lambda b$. We then show that this $\Omega(\lambda)$ bound on the number of unobstructed blocks holds also w.h.p. This would follow easily if for different blocks the events that the blocks are unobstructed were independent; but they are not, as they may depend on the outcome of the same coin flip. Nevertheless we observe that the dependence is limited, as each coin flip affects steps in at most b different blocks and each block is affected by at most $O(n)$ coin flips on average. To obtain the desired bound we use a concentration inequality from [27], which is a refinement of the standard method of bounded differences. Once we have established that $\Omega(\lambda)$ blocks are unobstructed, it follows that the probability that process p_1 flips heads at the beginning of some unobstructed block is $1-(1-1/n)^{\Omega(\lambda)} = 1-e^{-\Omega(\lambda/n)} \ge 1-\delta/n$ for the right choice of constants. Hence with at least this probability, p_1 finishes after at most λb steps.

Similar bounds are obtained also for the remaining processes: We use the same approach as above for each p_i, except that in place of σ we use the schedule σ_i obtained from σ by removing all instances of p_j except for the first λb ones, for all $1 \le j < i$. We conclude that with probability $1-\delta/n$, p_i finishes after taking at most λb steps, assuming that each of $p_1, \ldots, p_{i-1}$ also finishes after at most λb steps. The theorem then follows by combining the results for all processes.

We give now the detailed proof. Let $\lambda = \beta(n+b)\ln(n/\delta)$, for a constant $\beta > 0$ to be quantified later. Let $p_1, \ldots, p_k$ be the processes participating in schedule σ, listed in the order in which they are scheduled to take their (λb)-th step; processes that take fewer steps than λb are not listed. Let σ_i, for $1 \le i \le k$, be the schedule obtained from σ after removing all instances of p_j except for the first λb ones, for all $1 \le j < i$. For each $1 \le i \le k$, we identify λ disjoint blocks of σ_i, where the ℓ-th block, denoted $\sigma_{i,\ell}$, starts with p_i's step following its ℓ-th coin flip, and finishes after the last step of p_i before its $(\ell+1)$-th coin flip. By $|\sigma_{i,\ell}|$ we denote the number of steps contained in $\sigma_{i,\ell}$. We have $\sum_\ell |\sigma_{i,\ell}| \le n\lambda b$, i.e., blocks $\sigma_{i,1}, \ldots, \sigma_{i,\lambda}$ contain at most $n\lambda b$ steps in total, namely, λb steps by each of processes $p_1, \ldots, p_i$, and fewer than λb steps by each of the remaining processes.

Observe that if p_i has not finished before block $\sigma_{i,\ell}$ begins, and if p_i's coin flip before block $\sigma_{i,\ell}$ returns heads, then p_i is guaranteed to finish during $\sigma_{i,\ell}$ if all other steps by non-finished processes during $\sigma_{i,\ell}$ are dummy steps.

We say that a coin flip *potentially obstructs* $\sigma_{i,\ell}$ if it is performed by a process $p \ne p_i$, and at least one of the b steps by p following that coin flip takes place during $\sigma_{i,\ell}$. This step will be an actual step only if the coin flip is heads (this is not 'if and only if' because p may have finished, in which case it does a no-op). We say that block $\sigma_{i,\ell}$ is *unobstructed* if all coin flips that potentially obstruct this block are tails. The number of coin flips that potentially obstruct $\sigma_{i,\ell}$ is bounded by $|\sigma_{i,\ell}|/b + 2n$, because if process $p \ne p_i$ takes $s > 0$ steps in $\sigma_{i,\ell}$, then the coin-flips by p that potentially obstruct $\sigma_{i,\ell}$ are the at most $\lceil s/b \rceil$ ones during $\sigma_{i,\ell}$, plus at most one before $\sigma_{i,\ell}$.

From the above, the probability that $\sigma_{i,\ell}$ is unobstructed is at least $(1-1/n)^{|\sigma_{i,\ell}|/b+2n}$. Thus the expected number of unobstructed blocks among

$\sigma_{i,1}, \ldots, \sigma_{i,\lambda}$ is at least $\sum_\ell (1-1/n)^{|\sigma_{i,\ell}|/b+2n}$. Using now that $\sum_\ell |\sigma_{i,\ell}| \leq n\lambda b$, and that $(1-1/n)^{x+2n}$ is a convex function of x, we obtain that the above sum is minimized when all λ blocks have the same size, equal to nb. Thus, the expected number of unobstructed blocks is at least

$$\sum_\ell (1-1/n)^{|\sigma_{i,\ell}|/b+2n} \geq \lambda(1-1/n)^{(nb)/b+2n} \geq \lambda(1-1/n)^{3n} > \lambda/4^3 = \lambda/64.$$

Next we use Theorem 4, from Appendix B, to lower bound the number of unobstructed blocks w.h.p. Let the binary random variables $X_1, X_2, \ldots$ denote the outcome of the coin flips that potentially obstruct at least one block $\sigma_{i,1}, \ldots, \sigma_{i,\lambda}$ ($X_j = 1$ if and only if the j-th of those coin flips is heads). Then, $\Pr[X_j = 1] = 1/n$. Let $f(X_1, X_2, \ldots)$ denote the number of unobstructed blocks. We showed above that $\mathbf{E}[f(X_1, X_2, \ldots)] \geq \lambda/64$. Further, we observe that changing the value of X_j can change the value of f by at most the number of blocks that X_j potentially obstructs; let c_j denote that number. Then, $\max_j c_j \leq b$. Finally, since each block $m_{i,\ell}$ is potentially obstructed by at most $|\sigma_{i,\ell}|/b + 2n$ coin flips, $\sum_j c_j \leq \sum_\ell (|\sigma_{i,\ell}|/b + 2n) \leq 3n\lambda$, as $\sum_\ell |\sigma_{i,\ell}| \leq n\lambda b$, and thus $\sum_j c_j^2 \leq (3n\lambda/b) \cdot b^2 = 3nb\lambda$. Applying now Theorem 4 for $t = \lambda/128 \leq \mathbf{E}[f(X_1, X_2, \ldots)]/2$ gives $\Pr\big(f(X_1, \ldots, X_n) \leq t\big) \leq 2\exp\big(-\frac{t^2}{6b\lambda + 2tb/3}\big)$. Substituting $t = \lambda/128 = (\beta/128)(n+b)\ln(n/\delta)$ and letting $\beta = 2(6 \cdot 128^2 + 2 \cdot 128/3)$ yields $\Pr\big(f(X_1, \ldots, X_n) \leq \lambda/128\big) \leq 2e^{-2\ln(n/\delta)} < \delta/(2n)$. Thus, with probability at least $1 - \delta/(2n)$ at least $\lambda/128$ of the blocks $\sigma_{i,1}, \ldots, \sigma_{i,\lambda}$ are unobstructed. The probability that process p_i flips heads in at least one unobstructed block is then at least $\big(1 - \delta/(2n)\big) \cdot \big(1 - (1-1/n)^{\lambda/128}\big)$. Since $1 - (1-1/n)^{\lambda/128} \geq 1 - e^{\lambda/(128n)} > 1 - \delta/(2n)$, the above probability is at least $\big(1 - \delta/(2n)\big)^2 \geq 1 - \delta/n$.

We have just shown that for any $1 \leq i \leq k$, with probability at least $1 - \delta/n$ process p_i finishes after at most λb steps *assuming schedule* σ_i. However, schedules σ and σ_i yield identical executions if processes $p_1, \ldots, p_{i-1}$ all finish after executing at most λb steps (the executions are identical assuming that the same coin flips are used in both). Then, by the union bound, the probability that *all* processes p_i finish after executing no more than λb steps each is at least $1 - n \cdot \delta/n = 1 - \delta$. This concludes the proof of Theorem 2.

Randomized Leader Election. From Theorem 2 and Lemma 2 it follows that Algorithm 1 can be transformed into a randomized leader election implementation with step complexity $O(n^3 \log n)$.

Corollary 1. *There is a randomized variant of Algorithm 1 that has the same space complexity, and for any fixed schedule (determined by an oblivious adversary), w.h.p. every process finishes after at most $O(n^3 \log n)$ steps.*

If we use a variant of Algorithm 1 which employs the early termination criterion described in Remark 1 on page 8, then $b = O(n)$ and thus the step complexity of the randomized algorithm obtained is $O(n^2 \log n)$ w.h.p.

Giakkoupis and Woelfel [21] proposed a randomized implementation for leader election from $\Omega(n)$ registers with expected step complexity $O(\log^* n)$ against the oblivious adversary. Next we give an overview of this algorithm, and explain how to combine it with the randomized variant of Algorithm 1 to reduce space complexity to $O(\sqrt{n})$, without increasing the asymptotic step complexity.

The algorithm in [21] uses a chain of n group-election objects $G_1, \ldots, G_n$ alternating with n deterministic splitters $S_1, \ldots, S_n$, and a chain of n 2-process leader election objects $L_1, \ldots, L_n$. Each group-election object G_i guarantees that at least one of the processes accessing G_i gets elected, and if k processes access G_i then $O(\log k)$ get elected in expectation. Each splitter S_i returns one of the three outcomes: `win`, `lose`, or `cont` (for continue). It guarantees that if k processes access it, then at most one wins, at most $k-1$ lose, at most $k-1$ continue; thus, if only one process accesses the splitter, that process wins.

A process p proceeds by accesses the group-election objects in increasing index order. If p accesses G_i and fails to get elected, it loses immediately; if it does get elected, it then tries to win splitter S_i. If it loses S_i, it loses also the implemented leader election; if it returns `cont` it continues to the next group-election object, G_{i+1}; and if it wins S_i, it switches to the chain of 2-process leader election objects. In the last case it subsequently tries to win $L_i, L_{i-1}, \ldots, L_1$ (in this order). If it succeeds, it wins the implemented leader election, else it loses.

The analysis of the above algorithm given in [21] shows that in expectation only the first $O(\log^* n)$ group-election objects are used. Further, for any $i = \omega(\log^* n)$, the probability that S_i is used is bounded by $2^{-\Omega(i)}$.

We propose the following simple modification to this algorithm: For an index $K = \Theta(\log^2 n)$, we replace group-election object G_K with the randomized variant of Algorithm 1, and then remove all objects G_i, S_i, and L_i, for $i > K$. Clearly, the first modification does not affect the correctness of the algorithm, since any leader election algorithm is also a group-election algorithm. This modification guarantees that at most one process will ever access S_K. It follows that objects G_i, S_i, and L_i for $i > K$ will never be used, and thus are no longer needed. Hence, the space complexity of the new implementation is equal to that of Algorithm 1 plus $O(\log^3 n)$ registers, as each group-election object can be implemented from $O(\log n)$ registers. Further, the step complexity of the algorithm is the same as that of the original algorithm from [21], because a process reaches G_K with probability at most $2^{-\Omega(\log^2 n)} = n^{-\Omega(\log n)}$, and when this happens at most $O(n^3 \log n)$ additional steps are needed w.h.p., by Corollary 1. Thus, we have proved Theorem 3.

Conclusion

We provided a randomized wait-free algorithm for leader election (and thus test-and-set) from $O(\sqrt{n})$ registers and with $O(\log^* n)$ expected step complexity against the oblivious adversary. To obtain our result we first developed an obstruction-free algorithm with $O(\sqrt{n})$ space complexity. Then we devised and

applied a general construction that shows how any deterministic obstruction-free algorithm can be transformed to a randomized wait-free one, such that the expected step complexity is polynomial in n and in the maximum number of unobstructed steps a process needs to finish the obstruction-free algorithm.

We are not aware of any other obstruction-free implementation of an object with consensus number two or higher from $o(n)$ registers. Perhaps the most interesting open questions remains to be whether there is a consensus algorithm that needs only sub-linear many registers. While it is not clear whether our techniques can help developing such an algorithm, we believe that it yields interesting insights. Finding an $o(n)$-space algorithm for consensus would seem hopeless if it weren't even possible for the seemingly simpler problem of leader election.

References

1. Y. Afek, H. Attiya, D. Dolev, E. Gafni, M. Merritt, and N. Shavit. Atomic snapshots of shared memory. *J. of the ACM*, 40(4):873–890, 1993.
2. Y. Afek, E. Gafni, J. Tromp, and P. Vitányi. Wait-free test-and-set. In *Proc. of 6th WDAG*, pages 85–94, 1992.
3. D. Alistarh and J. Aspnes. Sub-logarithmic test-and-set against a weak adversary. In *Proc. of 25th DISC*, pages 97–109, 2011.
4. D. Alistarh, J. Aspnes, K. Censor-Hillel, S. Gilbert, and M. Zadimoghaddam. Optimal-time adaptive strong renaming, with applications to counting. In *Proc. of 30th PODC*, pages 239–248, 2011.
5. D. Alistarh, J. Aspnes, S. Gilbert, and R. Guerraoui. The complexity of renaming. In *Proc. of 52nd FOCS*, pages 718–727, 2011.
6. D. Alistarh, H. Attiya, S. Gilbert, A. Giurgiu, and R. Guerraoui. Fast randomized test-and-set and renaming. In *Proc. of 24th DISC*, pages 94–108, 2010.
7. J. H. Anderson. Composite registers. *Distr. Comp.*, 6(3):141–154, Apr. 1993.
8. J. Aspnes. Randomized consensus in expected $O(n^2)$ total work using single-writer registers. In *Proc. of 25th DISC*, pages 363–373, 2011.
9. J. Aspnes. Faster randomized consensus with an oblivious adversary. In *Proc. of 31st PODC*, pages 1–8, 2012.
10. J. Aspnes. A modular approach to shared-memory consensus, with applications to the probabilistic-write model. *Distr. Comp.*, 25(2):179–188, 2012.
11. H. Attiya and K. Censor. Tight bounds for asynchronous randomized consensus. *J. of the ACM*, 55(5), 2008.
12. H. Attiya and K. Censor-Hillel. Lower bounds for randomized consensus under a weak adversary. *SIAM J. on Comp.*, 39(8):3885–3904, 2010.
13. Y. Aumann. Efficient asynchronous consensus with the weak adversary scheduler. In *Proc. of 16th PODC*, pages 209–218, 1997.
14. H. Buhrman, A. Panconesi, R. Silvestri, and P. Vitányi. On the importance of having an identity or, is consensus really universal? *Distr. Comp.*, 18(3):167–176, 2006.
15. W. Eberly, L. Higham, and J. Warpechowska-Gruca. Long-lived, fast, waitfree renaming with optimal name space and high throughput. In *Proc. of 12th DISC*, pages 149–160, 1998.
16. F. Ellen, P. Fatourou, and E. Ruppert. Time lower bounds for implementations of multi-writer snapshots. *J. of the ACM*, 54(6), 2007.

17. F. Ellen, P. Fatourou, and E. Ruppert. The space complexity of unbounded timestamps. *Distr. Comp.*, 21(2):103–115, 2008.
18. F. Fich, M. Herlihy, and N. Shavit. On the space complexity of randomized synchronization. *J. of the ACM*, 45(5):843–862, 1998.
19. F. Fich, V. Luchangco, M. Moir, and N. Shavit. Obstruction-free algorithms can be practically wait-free. In *Proc. of 19th DISC*, pages 78–92, 2005.
20. M. Fischer, N. Lynch, and M. Paterson. Impossibility of distributed consensus with one faulty process. *J. of the ACM*, 32(2):374–382, 1985.
21. G. Giakkoupis and P. Woelfel. On the time and space complexity of randomized test-and-set. In *Proc. of 31st PODC*, pages 19–28, 2012.
22. W. Golab, D. Hendler, and P. Woelfel. An $O(1)$ RMRs leader election algorithm. *SIAM J. on Comp.*, 39:2726–2760, 2010.
23. M. Helmi, L. Higham, E. Pacheco, and P. Woelfel. The space complexity of long-lived and one-shot timestamp implementations. In *Proc. of 30th PODC*, pages 139–148, 2011.
24. M. Herlihy. Wait-free synchronization. *ACM Trans. Program. Lang. Syst.*, 13(1):124–149, 1991.
25. P. Jayanti, K. Tan, and S. Toueg. Time and space lower bounds for nonblocking implementations. *SIAM J. on Comp.*, 30(2):438–456, 2000.
26. C. Kruskal, L. Rudolph, and M. Snir. Efficient synchronization on multiprocessors with shared memory. *ACM Trans. Program. Lang. Syst.*, 10(4):579–601, 1988.
27. C. McDiarmid. Concentration. In M. Habib, C. McDiarmid, J. Ramirez-Alfonsin, and B. Reed, editors, *Probabilistic Methods for Algorithmic Discrete Mathematics*, pages 195–248. Springer-Verlag, 1998.
28. A. Panconesi, M. Papatriantafilou, P. Tsigas, and P. Vitányi. Randomized naming using wait-free shared variables. *Distr. Comp.*, 11(3):113–124, 1998.
29. E. Styer and G. Peterson. Tight bounds for shared memory symmetric mutual exclusion problems. In *Proc. of 8th PODC*, pages 177–191, 1989.
30. J. Tromp and P. Vitányi. Randomized two-process wait-free test-and-set. *Distr. Comp.*, 15(3):127–135, 2002.

Appendix A: Obstruction-Free M-Component Snapshots

We present an obstruction-free implementation of an M-component snapshot object from $M+1$ bounded registers. Formally, an M-component snapshot stores a vector $V = (V_1, \ldots, V_M)$ of M values from some domain D. It supports two operations; `scan()` takes no parameter and returns the value of V, and `update(`i`,`x`)`, $i \in \{1, \ldots, M\}$, $x \in D$, writes x to the i-th component of V and returns nothing.

Our implementation uses an array $A[1 \ldots M]$ of shared registers and a register S. Each array entry $A[i]$ stores a triple (w_i, p_i, b_i), where $w_i \in D$ represents the i-th entry in the vector V of the snapshot object, p_i is a process ID or $\perp$ which identifies the last process that wrote to $A[i]$, and $b_i \in \{0, 1\}$ is a bounded (modulo 2) sequence number. Initially, $S = \perp$ and each array entry $A[i]$ has the value $(w_i, \perp, 0)$ for some fixed $w_i \in D$.

Now suppose process p calls `update(`i, x`)`, and this is p's j-th update of the i-th component of V. To perform the update, p first writes its ID to S and then it writes the triple $(x, p, j \bmod 2)$ to $A[i]$.

To execute a `scan()`, process p first writes its ID to S. Then it performs a collect (i.e., it reads all entries of A) to obtain a *view* $a[1 \dots M]$, and another collect to obtain a second view $a'[1 \dots M]$. Finally, the process reads S. If S does not contain p's ID or if the views a and a' obtained in the two collects differ, then p starts its `scan()` over; otherwise it returns view a.

Obviously `scan()` is obstruction-free, and `update()` is even wait-free. Note also that a process which runs without obstruction can finish each operation in $O(M)$ steps.

To prove linearizability, we use the following linearization points: Each `update`(i, x) operation linearizes at the point when the calling process writes to $A[i]$, and each `scan()` operation that terminates linearizes at the point just before the calling process performs its last collect during its `scan()`. (We don't linearize pending `scan()` operations.)

Consider a `scan()` operation by process p which returns the view $a = a[1 \dots M]$. Let t be the point when that `scan()` linearizes, i.e., just before p starts its last collect. To prove linearizability it suffices to show that $A = a$ at point t.

For the purpose of a contradiction assume that this is not the case, i.e., there is an index $i \in \{1, \dots, M\}$ such that at time t the triple stored in $A[i]$ is not equal to $a[i]$. Let t_1 and t_2 be the points in time when p reads the value $(w, q, b) = a[i]$ from $A[i]$ during its penultimate and ultimate collect, respectively. Then $t_1 < t < t_2$. Since $A[i] \neq (w, q, b)$ at time t but $A[i] = (w, q, b)$ at times t_1 and t_2, process q writes (w, q, b) to $A[i]$ at some point in the interval $(t, t_2) \subseteq (t_1, t_2)$. Since p does not write to A during its `scan()`, this implies $q \neq p$.

First suppose q writes to $A[i]$ at least twice during (t_1, t_2). Each such write must happen during an `update()` operation by q. Since each `update()` operation starts with a write to S, q writes its ID to S at least once in (t_1, t_2). But since the penultimate collect of p's `scan()` starts before t_1 and the ultimate collect finishes after t_2, S cannot change in the interval (t_1, t_2), which is a contradiction.

Hence, suppose q writes to $A[i]$ exactly once in (t_1, t_2); in particular it writes the triple (w, q, b) to $A[i]$ at some point $t^* \in (t_1, t_2)$. Recall that each time q writes to $A[i]$ it alternates the bit it writes to the third component. Hence, at no point in $[t_1, t^*]$ the second and third component of $A[i]$ can have value q and b. In particular, $A[i] \neq (w, q, b)$ at point t_1, which is a contradiction.

Appendix B: A Concentration Inequality

The next result follows from [27, Theorem 3.9], which is an extension to the standard method of bounded differences.

Theorem 4. *Let $X_1, \dots, X_n$ be independent 0/1 random variables with $\Pr(X_i = 1) = p$. Let f be a bounded real-valued function defined on $\{0,1\}^n$, such that $|f(x) - f(x')| \leq c_i$, whenever vectors $x, x' \in \{0,1\}^n$ differ only in the i-the coordinate. Then for any $t > 0$,*

$$\Pr\left(|f(X_1, \dots, X_n) - \mathbf{E}[f(X_1, \dots, X_n)]| \geq t\right) \leq 2 \exp\left(-\frac{t^2}{2p\sum_i c_i^2 + 2t \max_i\{c_i\}/3}\right).$$