Efficient Randomized Test-And-Set Implementations

George Giakkoupis
INRIA, Rennes, France
george.giakkoupis@inria.fr

Philipp Woelfel
University of Calgary
woelfel@ucalgary.ca

February 12, 2019

Abstract

We study randomized test-and-set (TAS) implementations from registers in the asynchronous shared memory model with n processes. We introduce the problem of *group election*, a natural variant of leader election, and propose a framework for the implementation of TAS objects from group election objects. We then present two group election algorithms, each yielding an efficient TAS implementation. The first implementation has expected max-step complexity $O(\log^* k)$ in the location-oblivious adversary model, and the second has expected max-step complexity $O(\log\log k)$ against any read/write-oblivious adversary, where $k \leq n$ is the contention. These algorithms improve the previous upper bound by Alistarh and Aspnes [2] of $O(\log\log n)$ expected max-step complexity in the oblivious adversary model.

We also propose a modification to a TAS algorithm by Alistarh, Attiya, Gilbert, Giurgiu, and Guerraoui [5] for the strong adaptive adversary, which improves its space complexity from super-linear to linear, while maintaining its $O(\log n)$ expected max-step complexity. We then describe how this algorithm can be combined with any randomized TAS algorithm that has expected max-step complexity $T(n)$ in a weaker adversary model, so that the resulting algorithm has $O(\log n)$ expected max-step complexity against any strong adaptive adversary and $O(T(n))$ in the weaker adversary model.

Finally, we prove that for any randomized 2-process TAS algorithm, there exists a schedule determined by an oblivious adversary such that with probability at least $1/4^t$ one of the processes needs at least t steps to finish its TAS operation. This complements a lower bound by Attiya and Censor-Hillel [7] on a similar problem for $n \geq 3$ processes.

1 Introduction

In this paper we study time and space efficient implementations of *test-and-set (TAS)* objects from atomic registers in asynchronous shared memory systems with n processes. The TAS object is a fundamental synchronization primitive, and has been used in algorithms for classical problems such as mutual exclusion and renaming [3–5, 9, 11, 17, 20].

A TAS object stores a bit that is initially 0, and supports the operation `TAS()`, which sets the bit (or leaves it unchanged if it is already set) and returns its previous value; the process whose call returns 0 is the *winner* of the object. TAS objects are among the simplest natural primitives that have no deterministic wait-free linearizable implementations from atomic registers, even in systems with only two processes. In fact, in systems with exactly two processes, a consensus protocol can be implemented deterministically from a TAS object and vice versa.

The TAS problem is very similar to the problem of *leader election*. In a leader election protocol, every process decides for itself whether it becomes the leader (it returns `win`) or whether it loses (it returns `lose`). At most one process can become the leader, and not all participating processes

can lose. I.e., if all participating processes finish the protocol, then exactly one of them returns `win` and all others return `lose`. Obviously, any TAS object immediately yields a leader election protocol: Each process executes a single `TAS()` operation and returns `win` if the `TAS()` call returns 0, or `lose` if `TAS()` returns 1. Similarly, a leader election algorithm, together with one additional register, can be used to implement a linearizable TAS object with just a constant increase in the number of steps [15]. Similar transformations from leader election to linearizable TAS objects are implicit in several TAS algorithms, e.g., [1, 2].

Early randomized TAS implementations assumed a *strong adaptive adversary* model, where the adversary bases its scheduling decisions on the entire past history of events, including the coin flips by processes. Tromp, and Vitányi [22,23] presented a randomized implementation for two processes which has constant expected max-step complexity and constant space complexity against any strong adaptive adversary. (The max-step complexity of an execution is the maximum number of steps any process needs to finish its algorithm in the execution. See Section 2.2 for formal definitions and a discussion.) Afek, Gafni, Tromp, and Vitányi [1] gave a deterministic implementation of a TAS object for n processes, from $O(n)$ 2-process TAS objects. Any execution of this algorithm has max-step complexity $O(\log n)$. Using Tromp and Vitányi's randomized 2-process TAS implementation, one obtains a randomized implementation of a TAS object from registers with $O(\log n)$ expected max-step complexity in the strong adaptive adversary model. Alistarh, Attiya, Gilbert, Giurgiu, and Guerraoui [5] presented an *adaptive* variant of that algorithm, called `RatRace`, in which the expected max-step complexity is logarithmic in the contention k, i.e., the total number of processes accessing the TAS object. The space requirements of `RatRace` are higher, though, as $\Theta(n^3)$ registers are used.

No TAS algorithm with a sub-logarithmic expected max-step complexity against any strong adaptive adversary has been found yet, and no non-trivial time lower bounds are known either. The strong adaptive adversary, however, may be too strong in some settings to model realistic system behavior. Motivated by the fact that consensus algorithms benefit from weaker adversary models, Alistarh and Aspnes [2] devised a simple and elegant TAS algorithm with an expected max-step complexity of $O(\log \log n)$ for the *oblivious adversary* model, where the adversary has to make all scheduling decisions at the beginning of the execution. We will refer to this algorithm as the *AA-algorithm*. Although not explicitly mentioned in [2], the AA-algorithm works even for a slightly stronger adversary, the *read/write-oblivious (r/w-oblivious)* adversary. Such an adversary can take all past operations of processes, including coin flips, into account when making scheduling decisions, but it cannot see whether a process will read or write in its next step, if that decision is made by the process at random. The space complexity of the AA-algorithm is super-linear, as it uses `RatRace` as a component.

Our contribution. In view of their AA-algorithm, Alistarh and Aspnes asked whether any better TAS algorithm exists for the oblivious or even stronger adversary models. We answer this question in the affirmative: We present an adaptive algorithm that has an expected max-step complexity of $O(\log^* k)$ in the oblivious adversary model, where k is the contention. In fact, our result holds for the slightly stronger *location-oblivious* adversary. This adversary makes scheduling decisions based on all past events (including coin flips), but it does not know which register a process will access in its next step, if this decisions is made at random.

This algorithm, however, is not efficient in the r/w-oblivious adversary model. For such adversaries, we devise a different algorithm that has expected max-step complexity $O(\log \log k)$, and uses $O(n)$ registers. It is similar to the AA-algorithm, but introduces a new idea that makes it adaptive.

Adversary	Time	Space	Reference	Comments
strong adaptive	unbounded	$\lceil \log n \rceil + 1$	[21]	deadlock-free only
strong adaptive	$O(1)$	$\Theta(1)$	[22, 23]	2-process implementation
strong adaptive	$O(\log n)$	$\Theta(n)$	[1]	-
strong adaptive	$O(\log k)$	$\Theta(n^3)$	[5]	k is the contention
r/w-oblivious	$O(\log \log n)$	$\Theta(n^3)$	[2]	-
location-oblivious	$O(\log^* k)$	$\Theta(n)$	Theorem 4	-
r/w-oblivious	$O(\log \log k)$	$\Theta(n)$	Theorem 6	-
strong adaptive	$O(\log k)$	$\Theta(n)$	Theorem 7	-
oblivious	$O(\log^* k)$	$\Theta(\log n)$	[13]	uses impl. of Theorem 4

Table 1: Randomized TAS implementations. In the second column we give the expected max-step complexity of the algorithm.

Our two TAS algorithms above are the first ones with sub-logarithmic expected max-step complexity that need only $O(n)$ registers.

Both algorithms rely on a novel framework that uses a variant of the leader election problem, called *group election*, in which more than one process can get elected. We present a TAS implementation based on multiple such group election objects. The performance of the implementation is determined by the *effectiveness* of the group election objects used, which is measured in terms of the expected number of processes that get elected.

The AA-algorithm has the desirable property that its performance degrades gracefully when the adversary is not r/w-oblivious, and against a strong adaptive adversary it still achieves an expected max-step complexity of $O(\log k)$. In their basic form, our algorithms do not exhibit such a behavior—a strong adaptive adversary can find a schedule where processes need $\Omega(k)$ steps to complete their `TAS()` operation. To rectify that, we present a general method to combine any TAS algorithm with `RatRace`, so that if the algorithm has expected max-step complexity $T(k)$ against any r/w-oblivious or location-oblivious adversary, then the combined algorithm has expected max-step complexity $O\big(T(k)\big)$ in the same adversary model, and $O(\log k)$ against any strong adaptive adversary. Further, we propose a modification of `RatRace` that improves its space complexity from $O(n^3)$ to $O(n)$, without increasing its expected max-step complexity. Thus, combining this algorithm with any of our two algorithms for weak adversaries, yields an algorithm with linear space complexity.

Finally, we show for any randomized TAS implementation for two processes, that the oblivious adversary can schedule processes in such a way that for any $t > 0$, with probability at least $1/4^t$ one of the processes needs at least t steps to finish its `TAS()` operation. This result immediately implies the same lower bound on 2-process consensus. Attiya and Censor-Hillel [7] showed that with probability at least $1/c^t$, for some constant c, any randomized f-resilient n-process consensus algorithm does not terminate within a total number of $t(n - f)$ steps. However, the lower bound proof in [7] only works for $n \geq 3$ processes. Thus our result fills in the missing case of $n = 2$.

In the conference version of this paper [14], we also proved a lower bound of $\Omega(\log n)$ for the number of registers needed to implement nondeterministic solo-terminating TAS. After the conference paper was published, Dan Alistarh made us aware that a proof by Styer and Peterson from 1989 [21] implies this result. In particular, Styer and Peterson [21] showed that any implementation of deadlock-free leader election requires at least $\lceil \log n \rceil + 1$ registers. They also described a deadlock-free (deterministic) leader election algorithm that uses $\lceil \log n \rceil + 1$ registers. However, this

algorithm is not wait-free (and thus has unbounded step complexity).

Until recently, it was not unknown whether any randomized wait-free (or obstruction-free) TAS implementation exists that uses fewer than $O(n)$ registers. After completion of the draft of this paper, Giakkoupis, Helmi, Higham, and Woelfel [12, 13] presented deterministic obstruction-free algorithms that use only $O(\sqrt{n})$ and $O(\log n)$ registers, respectively. As the authors observed, these algorithms can be turned into randomize wait-free ones, and can be combined with the first algorithm proposed in this paper to achieve $O(\log^* n)$ expected max-step complexity in the oblivious adversary model, with $O(\sqrt{n})$ and $O(\log n)$ space complexity, respectively.

2 Preliminaries

We consider an asynchronous shared memory model where up to n processes, with IDs $1, \dots, n$, communicate by reading and writing to atomic shared multi-reader multi-writer registers. Registers can store values from an arbitrary countable domain. Algorithms are randomized and use local coin flips to make random decisions. A coin flip is a step that yields a random value from some countable space Ω, using an arbitrary but fixed probability distribution $\mathcal{D}$. Coin flips are private, i.e., only the process that executes the coin flip gets to see the outcome. For the model description we will assume (w.l.o.g.) that processes alternate between coin flip steps and shared memory steps (i.e., reads or writes), and that their first step is always a coin flip. Our algorithm descriptions do not always follow this convention, because in the given programs processes may execute multiple consecutive shared memory steps without any coin flips in-between. Obviously one can simply add "dummy" coin flip steps in order to achieve an alternation.

An *execution* is a possibly infinite sequence, where the i-th element contains all information describing the i-th step. That comprises the ID of the process taking that step, the type of step (read, write, or coin flip), the affected register in case of a read or write, the value returned in case of a read or coin flip, and the value written in case of a write. A *schedule* is a sequence of process IDs in $\{1, \dots, n\}$, and a *coin flip vector* ω is a sequence of coin flip values in Ω; these sequences may be infinite. Every execution $\mathcal{E}$ uniquely defines a schedule $\sigma(\mathcal{E})$ that is obtained from $\mathcal{E}$ by replacing each step with the ID of the process performing that step, and a coin flip sequence $\omega(\mathcal{E})$, which is the sequence of coin flip values defined by $\mathcal{E}$. Similarly, for a given algorithm M, a schedule σ together with an infinite coin flip vector $\omega = (\omega_1, \omega_2, \dots)$ uniquely determine an *execution* $\mathcal{E}_M(\sigma, \omega)$, in which processes execute their shared memory and coin flip steps in the order specified by σ, and the value returned from the i-th coin flip (among all processes) is ω_i. If a process has finished its algorithm, it does not take any more steps, even if it gets scheduled (alternatively, one can think of the process continuing to execute only no-ops).

2.1 Adversary Models

An adversary decides at any point of an execution, which process will take the next step. Formally, an *adversary* A is a function that maps a finite execution $\mathcal{E}$ of some algorithm M to a process ID $A(\mathcal{E})$, which identifies the process to take the next step following $\mathcal{E}$. This way, adversary A and algorithm M, together with an infinite coin flip vector ω, yield a unique infinite schedule $\sigma_M(A, \omega) = (\sigma_1, \sigma_2, \dots)$, where $\sigma_1 = A(\varepsilon)$ for the empty execution ε, and

$$\sigma_{i+1} = A\Big(\mathcal{E}_M\big((\sigma_1, \dots, \sigma_i), \omega\big)\Big).$$

Thus, given algorithm M and adversary A we can obtain a random schedule $\sigma_M(A, \omega)$ and the corresponding random execution $\mathcal{E}_M(\sigma_M(A, \omega), \omega)$ by choosing a coin flip vector ω at random

according to the product distribution $\mathcal{D}^\infty$ over the set Ω^∞ of infinite coin flip vectors. The coin flip vector ω is the only source of randomness, here. We denote the random execution $\mathcal{E}_M(\sigma_M(A,\omega),\omega)$ by $\mathcal{E}_{M,A}$, and call it the *random execution of M scheduled by A*. We are interested in random variables and their expectation defined by $\mathcal{E}_{M,A}$, e.g., the maximum number of shared memory steps any process takes (see Section 2.2).

An *adversary model* $\mathcal{A}$ maps each algorithm M to a family $\mathcal{A}(M)$ of adversaries. We say that an algorithm M has certain properties against any adversary in $\mathcal{A}$ to denote that these properties are satisfied for any adversary $A \in \mathcal{A}(M)$. The *strong adaptive adversary model* is defined for any algorithm as the set of all adversaries. Here, the next process scheduled to take a step is decided based on the entire past execution (including the results of all coin flip steps so far). The *oblivious adversary model* is the weakest standard adversary model, where each adversary A is a function of just the length of the past execution, i.e., $A(\mathcal{E}) = A(\mathcal{E}')$, if $|\mathcal{E}| = |\mathcal{E}'|$. Therefore, an oblivious adversary results in a schedule that is fixed in advance and is independent of the coin flip vector.

Several *weak* adaptive adversary models have been proposed, which are stronger than the oblivious model but weaker than the strong adaptive model. We will consider two such models. An adversary A for algorithm M is *location-oblivious* if for any finite execution $\mathcal{E}$ of M, the next processes $A(\mathcal{E})$ scheduled by A to take a step can depend on the following information:

(i) the complete past schedule $\sigma(\mathcal{E})$;

(ii) the return values of all coin flip steps performed by each process p *preceding* p's latest shared memory step in $\mathcal{E}$; and

(iii) for each process p that does not finish in $\mathcal{E}$ and its next step is a shared memory step, the information whether that step will be a read or a write operation, and, in case of a write, the value that p will write.

In particular, the location-oblivious adversary does not make a scheduling decision based on *which register* each process p will access in its next shared memory step, if that register is determined at random based p's coin flip *after* its latest shared memory step in $\mathcal{E}$.

Similar but incomparable to the location-oblivious adversary model is the *r/w-oblivious* adversary model. An adversary A for algorithm M is r/w-oblivious if for any finite execution $\mathcal{E}$ of M, $A(\mathcal{E})$ can depend on (i) and (ii) above, and also on the following information:

(iii′) for each process p that does not finish in $\mathcal{E}$ and its next step is a shared memory step, the register that p will access in that step.

In particular, the adversary does not make a scheduling decision based on whether a process p's next shared memory step is a read or a write operation, if this decision is made at random based on p's coin flip after its last shared memory step in $\mathcal{E}$.

2.2 Complexity Measures

We use the following standard definitions. The *space complexity* of an implementation is the number of registers it uses. An event occurs *with high probability (w.h.p.)*, if it has probability $1 - 1/m^{\Omega(1)}$ for some parameter m, as $m \to \infty$. In our case, m will be either n, the total number of processes, or k, a notion of congestion defined in Section 2.2.

We are interested in randomized leader election, and a variant of it called group election. These problems are *one-time* in the sense that each process can participate in a leader (or group) election at most once. The following definitions are thus limited to one-time operations *op*.

Let M be an algorithm in which a processes may call some operation *op* (possibly in addition to other operations). For any process p and any execution $\mathcal{E}$ of algorithm M, let $T_{op,p}(\mathcal{E})$ be the number of shared memory steps that p executes in $\mathcal{E}$ during its *op* call, and let $T_{op,p}(\mathcal{E}) = 0$ if p does not call *op*. The *max-step complexity* of *op* in execution $\mathcal{E}$ is defined as

$$\max_p T_{op,p}(\mathcal{E}).$$

The *expected max-step complexity* of *op* in algorithm M against an adversary A is

$$\mathbf{E}\left[\max_p T_{op,p}(\mathcal{E}_{M,A})\right], \tag{1}$$

where $\mathcal{E}_{M,A}$ is a random execution of M scheduled by A (see Section 2.1). The expected max-step complexity of *op* against A is the supremum of the quantity in (1) over all algorithms M. The expected max-step complexity of *op* against an adversary model $\mathcal{A}$ is the supremum of the quantity in (1) over all M and all $A \in \mathcal{A}(M)$.

In previous works [2, 5], the terms "expected individual step complexity" or simply "expected step complexity" have been used to denote what we refer to as "expected max-step complexity." We prefer to use a new and thus unambiguous term to clearly distinguish this measure from other step complexity measures, and in particular, from $\max_p \mathbf{E}[T_{op,p}(\mathcal{E}_{M,A})]$. It follows immediately from the definition of expectation that $\max_p \mathbf{E}[T_{op,p}(\mathcal{E}_{M,A})] \leq \mathbf{E}[\max_p T_{op,p}(\mathcal{E}_{M,A})]$.

Our implementations of group and leader election objects are *adaptive* with respect to contention, i.e., their max-step complexity depends on the number of participating processes rather than n, the number of processes in the system. In fact, the only way in which n is used in the design of our algorithms is to determine the number of registers that must be used. If we allow the implementation to use unbounded space, then n can be unbounded, too.

Expressing the max-step complexity in terms of contention requires some care. We are interested in the conditional expectation of the max-step complexity of an operation *op*, given that the number of processes calling *op* is limited by some value k. A straightforward idea to limit contention would be to consider $\mathbf{E}[\max_p T_{op,p}(\mathcal{E}_{M,A}) \mid K \leq k]$, where K is the actual number of processes that execute *op* in $\mathcal{E}_{M,A}$. But this does not yield satisfying results, as an adaptive adversary may be able to force that conditional expectation to be unreasonably large for any given $k < n$. An adversary might achieve that, e.g., by letting k processes start their operation *op*, and if it sees during the execution that the coin flips are favorable (i.e., will yield a fast execution), it can schedule one more process to invoke *op*, increasing the contention to more than k processes. This would prevent "fast" executions from contributing to $\mathbf{E}[\max_p T_{op,p}(\mathcal{E}_{M,A}) \mid K \leq k]$.

We define a measure of contention, called *max-contention*, that the adversary cannot change once the first process is poised to invoke operation *op*. Let $\mathcal{E}$ be an execution of algorithm M, and let $\mathcal{E}'$ be the prefix of $\mathcal{E}$ ending when the first process becomes poised to invoke *op*; $\mathcal{E}' := \mathcal{E}$ if no such process exists. The *max-contention* of *op* in execution $\mathcal{E}$ of algorithm M, denoted $k_{\max}^{M,op}(\mathcal{E})$, is the maximum number of processes that invoke *op*, in *any execution of M that is an extension of $\mathcal{E}'$*. In other words, $k_{\max}^{M,op}(\mathcal{E})$ is the maximum number of invocations of *op* for any possible way of continuing execution $\mathcal{E}'$ of M.

Let $Exec_{M,A,op}(k)$ be the set of all possible executions $\mathcal{E}'$ of algorithm M that can result for a given adversary A, and have the properties that: (i) $\mathcal{E}'$ ends when the first process becomes poised to invoke *op*; and (ii) $k_{max}^{M,op}(\mathcal{E}') \leq k$. We define the *adaptive expected max-step complexity* of *op* in algorithm M against adversary A to be a function $\tau : \{1, \ldots, n\} \to \mathbb{R}_{\geq 0}$, where

$$\tau(k) := \sup_{\mathcal{E}' \in Exec_{M,A,op}(k)} \mathbf{E}\left[\max_p T_{op,p}(\mathcal{E}_{M,A}) \;\middle|\; \mathcal{E}_{M,A} \text{ is an extension of } \mathcal{E}'\right]. \tag{2}$$

Object `Doorway`
shared: register $B \leftarrow$ `false`

Method `enter()`

```
1 if B.read() = false then
2 |   B.write(true)
3 |   return true
4 end
5 return false
```

Figure 1: A doorway implementation.

The adaptive expected max-step complexity of *op* against adversary A (or against an adversary model $\mathcal{A}$) is defined similarly to $\tau(k)$, except that the supremum is taken also over all algorithms M (respectively, over all M and all $A \in \mathcal{A}(M)$). We say that the *adaptive max-step complexity* of *op* in algorithm M against adversary A is bounded by $b(k)$ with probability $q(k)$, if

$$\Pr\left(\max_p T_{op,p}(\mathcal{E}_{M,A}) \leq b(k) \;\middle|\; \mathcal{E}_{M,A} \text{ is an extension of } \mathcal{E}'\right) \geq q(k), \text{ for all } \mathcal{E}' \in Exec_{M,A,op}(k).$$

We also say that the adaptive max-step complexity of *op* against A (or $\mathcal{A}$) is bounded by $b(k)$ with probability $q(k)$, if the above holds for all algorithms M (respectively, all M and all $A \in \mathcal{A}(M)$). Throughout the remainder of the paper, when we say *(expected) max-step complexity*, we mean *adaptive (expected) max-step complexity*.

In the terminology introduced in this section, we will often replace operation *op* by the object G that supports this operation, if *op* is the only operation that G provides.

2.3 Some Basic Objects

We now describe several simple objects that we use as building blocks for our TAS algorithms.

A *doorway* object supports the operation `enter()` which takes no parameters and returns a boolean value, `true` or `false`. Each process calls `enter()` at most once, and we say that it *enters* the doorway when it invokes `enter()`, and *exits* when the `enter()` method responds. The process *passes through* the doorway if its `enter()` method returns `true`, and is *deflected* if it returns `false`. A doorway object satisfies the following two properties:

(D1) Not all processes entering the doorway are deflected; and

(D2) If a process passes through the doorway, then it entered the doorway before any process exited the doorway.

A simple, wait-free implementation of a doorway object is given in Figure 1. It is straightforward that the implementation satisfies properties (D1) and (D2): The first process that writes to B "closes" the doorway. All processes that read B after that will be deflected, and thus (D2) is true. But the first process that reads B does not get deflected, because at the point of that read, no process has written B. Therefore, (D1) is also true. The implementation uses only one register and each process finishes its `enter()` method in a constant number of steps.

A *randomized 2-process TAS object* can be implemented from a constant number of registers, so that its `TAS()` method has constant expected max-step complexity. More precisely, an implementation by Tromp and Vitányi [23] uses two single-reader single-writer registers, and guarantees for any strong adaptive adversary and any $\ell > 0$, that with probability at least $1 - 1/2^\ell$, both

Object `Splitter`
shared: register X; `Doorway` D

Method `split()`

```
1  X.write(myID)
2  if D.enter() then
3  |  if X.read() = myID  return stop
4  |  return right
5  end
6  return left
```

Object `RSplitter`
shared: register X; `Doorway` D

Method `split()`

```
7  X.write(myID)
8  if D.enter() then
9  |  if X.read() = myID  return stop
10 end
11 Choose dir ∈ {left, right} uniformly at random
12 return dir
```

Figure 2: Deterministic and randomized splitter implementations.

processes finish after $O(\ell)$ steps. In our algorithms, when a process calls the `TAS()` method of a 2-process TAS object it must "simulate" one of two possible IDs, 1 or 2. Thus, we use a 2-process TAS object $\texttt{TAS}_2$ that supports an operation `TAS(`i`)`, where $i \in \{1, 2\}$. If two processes call the method `TAS(`i`)`, they must use different values for i. We will say that a process *wins* (*loses*) if its `TAS()` call returns 0 (respectively 1).

A *splitter object* [8,19] provides a single method `split()`, which takes no parameters and returns a value in $\{\texttt{stop}, \texttt{left}, \texttt{right}\}$. If a process p calls `split()`, we say that p *goes through the splitter*. If the call returns `stop`, we say that p *stops* at the splitter; and if it returns `left` (`right`), we say p *turns left* (respectively *right*).

A *deterministic splitter*, denoted `Splitter`, was proposed by Moir and Anderson [19]. It guarantees that if ℓ processes go through the splitter, then at most $\ell - 1$ turn left, at most $\ell - 1$ turn right, and at most one stops. Thus if only one process goes through the splitter, that process stops.

A *randomized splitter*, denoted `RSplitter`, was proposed by Attiya, Kuhn, Plaxton, Wattenhofer and Wattenhofer [8]. Similarly to the deterministic splitter, it guarantees that if only one process goes through the splitter, then that process must stop. But now, any process that does not stop, turns left or right with equal probability, and independently of other processes. Randomized and deterministic splitters are incomparable in "strength", as for a randomized splitter it is possible that all processes going through it turn to the same direction.

Both splitter implementations, the deterministic one by Moir and Anderson, and the randomized by Attiya et. al., use two shared registers and have max-step complexity $O(1)$ in any execution. For completeness we provide the implementations in Figure 2. The deterministic splitter implementation has the following additional doorway-like property, which is useful for the design of our algorithms:

(S) If a process stops or turns right at the splitter, then its `split()` call was invoked before any other `split()` call on the same object responded.

This follows immediately from the use of doorway D in line 2: Suppose a `split()` operation by process p gets invoked after some other `split()` call by process q responded. Then q has already exited the doorway, when p enters it, so by doorway-property (D2) process p gets deflected, and its `split()` call returns `left`.

3 Fast TAS for Weak Adversaries

We present implementations of TAS objects for weak adversary models. In Section 3.1, we introduce the problem of group election, which is a natural variant of leader election, and in Section 3.2, we give a TAS implementation from group election objects. Then, in Sections 3.3 and 3.4, we provide efficient randomized implementations of group election from registers, for the location-oblivious and the r/w-oblivious adversary models, respectively.

3.1 Group Election

In the *group election problem* processes must elect a non-empty subset of themselves, but unlike in leader election, it is not required that exactly one process gets elected. Still it is desirable that the *expected* number of processes elected should be bounded by a small function in the number of participating processes.

Formally, a group election object, denoted `GroupElect`, provides the method `elect()`, which takes no parameters and returns either `win` or `lose`. We say a process *participates* in a group election when it calls `elect()`. The processes whose `elect()` calls return `win` get *elected*. A group election object must satisfy the following property:

(GR) Not all participating processes' `elect()` calls return `lose`.

That is, if at least one process participates and all participating processes finish their `elect()` calls, then at least one process gets elected.

We are interested in group election objects for which the expected number of elected processes is bounded by a (small) function of the max-contention. This function is called the *effectiveness* of the group election object and is formally defined next.

Consider an n-process group election object G. Let M be an algorithm in which processes invoke the `elect()` operation of G, and let A be an adversary. For an execution $\mathcal{E}$ of M, let $win(\mathcal{E})$ denote the number of processes that get elected on G. Similarly to definition (2) for max-step complexity, let $Exec_{M,A,G}(k)$ be the set of all possible executions $\mathcal{E}'$ of M that can result for adversary A, and have the properties that: (i) $\mathcal{E}'$ ends when the first process is poised to invoke G.`elect()`; and (ii) $k_{max}^{M,G}(\mathcal{E}') \leq k$. The *effectiveness* of group election object G in algorithm M against adversary A is a function $\varphi : \{1, \ldots, n\} \to [1, n]$, where

$$\varphi(k) := \sup_{\mathcal{E}' \in Exec_{M,A,G}(k)} \mathbf{E}\left[win(\mathcal{E}_{M,A}) \;\middle|\; \mathcal{E}_{M,A} \text{ is an extension of } \mathcal{E}' \right].$$

The effectiveness of G against adversary A (or against an adversary model $\mathcal{A}$) is defined similarly to $\phi(k)$, except that the supremum is taken also over all algorithms M (respectively, over all M and $A \in \mathcal{A}(M)$).

3.2 TAS from Group Election

We now present an implementation of a TAS object from n group election objects. The algorithm uses also n deterministic splitters, n 2-process TAS objects, and one doorway object. All these objects can be implemented from a total number of $O(n)$ registers, as we saw in Section 2.3.

Object `TAS`
shared: `GroupElect` $G[1 \dots n]$; `Splitter` $S[1 \dots n]$; `TAS`$_2$ $T[1 \dots n]$; `Doorway` D

Method `TAS()`

```
1  if D.enter() = false return 1
2  i ← 0
3  repeat
4  |  i ← i + 1
5  |  if G[i].elect() = lose return 1
6  |  s ← S[i].split()
7  |  if s = left return 1
8  until s = stop
9  if T[i].TAS(1) = 1 return 1
10 while i > 1 do
11 |  i ← i − 1
12 |  if T[i].TAS(2) = 1 return 1
13 end
14 return 0
```

Figure 3: An implementation of TAS from group election objects.

The implementation is given in Figure 3. First, each process enters a doorway, and if deflected, its `TAS()` call immediately returns 1. Any process that passes through the doorway participates in a series of group elections, on objects $G[1], \dots, G[n]$. If the process is not elected on $G[i]$, then its `TAS()` returns 1. Otherwise, it goes through splitter $S[i]$ next. If the process turns left at the splitter, then `TAS()` returns 1; if it turns right, it participates in the next group election, on $G[i+1]$. Finally, if the process stops at $S[i]$, then it does not participate in any further group elections. Instead, it tries to win a series of 2-process TAS, on $T[i], \dots, T[1]$, until it either loses in one of them and returns 1, or wins in all of them and returns 0.

The idea is that fewer and fewer processes participate in each group election, as only processes that get elected in $G[i]$ may participate in $G[i+1]$. The rate at which the number of processes drops depends on the effectiveness of the group election objects. The purpose of the doorway at the beginning is to achieve linearizability (without the doorway, we would obtain a leader election object instead). The splitter objects serve two purposes. First, they ensure that as soon as only one process remains, that process will not participate in other group elections, and will switch to the list of 2-process TAS objects. Second, they guarantee that the number of processes participating in each $G[i]$ *strictly* decreases with i. This ensures that no more than n group election objects are needed. Finally, the 2-process TAS objects ensure that (at most) one process returns 0.

Next we prove the correctness of the implementation, and analyze its max-step complexity in terms of the max-step step complexity and effectiveness of the group election objects used.

We use the following standard notation. For any function $f\colon X \to Y$, where $Y \subseteq X$, and for $i \geq 0$, we denote by $f^{(i)}$ the i-fold composition of f, defined recursively by $f^{(0)}(x) = x$ and $f^{(i+1)}(x) = f(f^{(i)}(x))$. Further, if f is a real function, we define

$$f^*(x) = \inf\{i\colon f^{(i)}(x) \leq 1\}.$$

Theorem 1. *Figure 3 gives an implementation of a TAS object from a set of group election objects. Suppose that for each group election object $G[i]$ used in this implementation, the expected max-step complexity of $G[i]$ against a given adversary A is bounded by a function $t(k)$ of the max-contention k of $G[i]$, and the effectiveness of $G[i]$ against A is bounded by a function $f(k)$. Suppose also that functions f and t are non-decreasing, and f is concave. Then the expected max-step complexity of the TAS implementation against A is $O\big(t(k)\cdot g^*(k)\big)$, where $g(k) := \min\{f(k), k-1\}$. Moreover, the same bound on the expected max-step complexity applies even if the assumption that the effectiveness of $G[i]$ is bounded by $f(k)$ holds only for $1 \le i \le g^*(n)$.*

The assumption that functions t and f are non-decreasing is not restrictive, as the expected max-step complexity and the effectiveness are by definition non-decreasing functions of the max-contention. The requirement that f is concave is also reasonable, as it suggests that the larger the max-contention, the smaller the increase in the expected number of elected processes, for the same increase in max-contention. This assumption is needed in the analysis for the following reason: We inductively obtain upper bounds for the expectation $E[k_j]$ of the number k_j of processes participating in the group election on $G[j]$. Concavity of f allows us to bound the expected number of processes getting elected on $G[j]$ by Jensen's inequality: $E[f(k_j)] \le f(E[k_j])$.

3.2.1 Proof of Theorem 1

We first show that the implementation is correct, and then analyze its max-step complexity.

Correctness. Consider an arbitrary execution. For $j \ge 1$, let m_j be the number of processes that begin j iterations of the repeat-until loop (lines 3–8), and for $1 \le j \le n$ let e_j be the number of processes that get elected on group election object $G[j]$ (in line 5). Clearly, $e_j \le m_j$, and at most e_j processes go through splitter $S[j]$ (in line 6). Moreover, by the splitter semantics, at most $e_j - 1$ of them turn right, provided $e_j \ge 1$. Thus, if $e_j \ge 1$, at most $e_j - 1 \le m_j - 1$ processes execute a $(j+1)$-th iteration of the repeat-until loop. Hence, $m_{j+1} < m_j$, and in particular $m_{n+1} = 0$. Thus, we have

$$j^* := \max_j\{m_j \ge 1\} \le n. \tag{3}$$

Next we observe that each 2-process TAS object $T[j]$, $1 \le j \le n$, is accessed by at most two processes: possibly a single process which stops at $S[j]$ and then calls $T[j]$.`TAS(1)` (in line 9), and possibly the winner of $T[j+1]$, if $j < n$, which calls $T[j]$.`TAS(2)` (in line 12). At most one of them can win $T[j]$. Since a process needs to win $T[1]$ (either in line 9 or in line 12) in order to win the implemented `TAS()` method, it follows that at most one process wins.

We now argue that *at least* one process wins, provided that at least one process calls the implemented `TAS()` method, and that all processes that do so finish their call. Recall that by (3), $j^* \le n$ is the largest index such that at least one process starts its j^*-th iteration of the while-loop. By (GR), at least one process gets elected on $G[j^*]$, and subsequently goes through splitter $S[j^*]$. I.e., $e_{j^*} \ge 1$. Since $m_{j^*+1} = 0$, none of these e_{j^*} processes executes another iteration of the repeat-until loop, and thus they all turn left or stop at $S[j^*]$. As not all of them can turn left either, at least one (and thus by the splitter semantics exactly one) process must stop at $S[j^*]$. It follows that at least one process calls $T[j^*]$.`TAS(1)`, and so at least one process wins $T[j^*]$. Since for $1 < j \le n$ the winner of $T[j]$ continues to $T[j-1]$, some process must win $T[1]$. Thus some process wins the implemented `TAS()` method.

It remains to show that the TAS implementation is linearizable. If process p's `TAS()` call z returns 0, then by property (D2) of the doorway, p must have entered doorway D during z before any other process exited it. In particular, no `TAS()` call *happens before* z (i.e., responds before z

gets invoked). Thus, we can obtain a linearization of the execution by putting z first, and adding all other `TAS()` operations after z in the order of their invocation. The resulting sequential history is valid (the first `TAS()` returns 0, and all other `TAS()` return 1), and preserves the happens-before order, because no `TAS()` happens before z.

Step Complexity. Consider an algorithm M that uses the implemented TAS object. Let $\mathcal{E} := \mathcal{E}_{M,A}$ be a random execution of M scheduled by adversary A.

For each $1 \leq j \leq n$, let $\mathcal{E}_j$ be the prefix of execution $\mathcal{E}$, until the first process is poised to invoke $G[j]$.`elect()`; or $\mathcal{E}_j := \mathcal{E}$ if no such process exists. Observe that if $\mathcal{E}_1 \neq \mathcal{E}$, then the last step of $\mathcal{E}_1$ is the step at which the first process passes through doorway D. Similarly for $i > 1$, if $\mathcal{E}_i \neq \mathcal{E}$ then the last step of $\mathcal{E}_i$ is the step in which the first process turns right at splitter $S[j-1]$.

For $1 \leq j \leq n$, let $k_j := k_{max}^{M,G[j]}(\mathcal{E})$ be the max-contention of $G[j]$ in $\mathcal{E}$. By definition, it is also

$$k_j = k_{max}^{M,G[j]}(\mathcal{E}_j).$$

Let $\mathcal{E}_0$ be the prefix of $\mathcal{E}$ until the first process is poised to invoke the implemented `TAS()` operation, i.e., it is poised to enter doorway D. Let $k_0 := k_{max}^{M,D}(\mathcal{E})$ be the max-contention of D in $\mathcal{E}$, and thus $k_0 = k_{max}^{M,D}(\mathcal{E}_0)$ as well.

Observe that, for any $1 \leq j \leq n$, execution $\mathcal{E}_{j-1}$ is a prefix of $\mathcal{E}_j$, and $k_{j-1} \geq k_j$.

Let $T(\mathcal{E})$ denote the max-step complexity of the implemented TAS in execution $\mathcal{E}$. To prove the expected max-step complexity bound claimed in the theorem we must show that for any given $k \geq 0$, if $k_0 = k$ then

$$\mathbf{E}[T(\mathcal{E}) \mid \mathcal{E}_0] = O\big(t(k) \cdot g^*(k)\big).$$

We will assume $k_0 \geq 1$, otherwise $T(\mathcal{E}) = 0$ as no process invokes the implemented `TAS()`.

First we bound the expected number of group election objects accessed by at least one process in the execution.

For $1 \leq j \leq n$, let e_j be the number of processes elected in the group election on $G[j]$. From the theorem's assumption that the effectiveness of $G[j]$ is bounded by function f of the max-contention of $G[j]$, it follows

$$\mathbf{E}[e_j \mid \mathcal{E}_j] \leq f(k_j).$$

We take the conditional expectation given $\mathcal{E}_0$ to obtain

$$\mathbf{E}[\mathbf{E}[e_j \mid \mathcal{E}_j] \mid \mathcal{E}_0] \leq \mathbf{E}[f(k_j) \mid \mathcal{E}_0].$$

The expression on the left equals $\mathbf{E}[e_j \mid \mathcal{E}_0]$ by the tower rule, since $\mathcal{E}_0$ is a prefix of $\mathcal{E}_j$. For the right side we have $\mathbf{E}[f(k_j) \mid \mathcal{E}_0] \leq f(\mathbf{E}[k_j \mid \mathcal{E}_0])$, by Jensen's inequality and the assumption that f is concave. Therefore,

$$\mathbf{E}[e_j \mid \mathcal{E}_0] \leq f(\mathbf{E}[k_j \mid \mathcal{E}_0]). \tag{4}$$

For $j = 1$, (4) yields

$$\mathbf{E}[e_1 \mid \mathcal{E}_0] \leq f(\mathbf{E}[k_1 \mid \mathcal{E}_0]) \leq f(\mathbf{E}[k_0 \mid \mathcal{E}_0]) = f(k_0),$$

where the second inequality holds because $k_1 \leq k_0$ and f is non-decreasing, and the last equation holds because k_0 is completely determined given $\mathcal{E}_0$.

For $j > 1$, we have $k_j \leq e_{j-1}$: This is trivial if $k_j = 0$. If $k_j \geq 1$ then the last step of $\mathcal{E}_j$ is when the first process p turns right at splitter $S[j-1]$. Property (S) then implies that any other process q that may participate at the group election in $G[j]$ must have already invoked $S[j-1]$.`split()`, and thus must have already been elected at $G[j-1]$.

Using the inequality $k_j \leq e_{j-1}$ we have just shown, and the assumption f is non-decreasing, we obtain from (4) that for $j > 1$,

$$\mathbf{E}[e_j \mid \mathcal{E}_0] \leq f(\mathbf{E}[e_{j-1} \mid \mathcal{E}_0]).$$

Combining the above inequalities for $j = 1, 2, \ldots$, and using that f is non-decreasing we get

$$\mathbf{E}[e_j \mid \mathcal{E}_0] \leq f^{(j)}(k_0).$$

The e_j processes elected on $G[j]$ will participate in an additional number of at most e_j group election objects beyond the first j ones, as each splitter $S[i]$ ensures $e_{i+1} \leq e_i - 1$, if $e_i > 0$. Therefore, if $j^* := \max\{j \colon k_j > 0\}$ is the total number of group election objects accessed by at least one process, then for any $0 \leq j \leq k_0$,

$$\mathbf{E}[j^* \mid \mathcal{E}_0] \leq j + f^{(j)}(k_0). \tag{5}$$

Let

$$x := \max\{y \leq k_0 \colon f(y) \geq y - 1\}, \quad \lambda := \min\{i \colon f^{(i)}(k_0) \leq x\}.$$

Note that $x \geq 1$, as $f(y) \geq 0$ for $y \geq 0$. Also, since f is concave and non-negative, it follows

$$f(y) \geq y - 1, \text{ for } 0 \leq y \leq x.$$

Setting $j := \lambda$ in (5) we obtain $\mathbf{E}[j^* \mid \mathcal{E}_0] \leq \lambda + f^{(\lambda)}(k_0)$. Since $f(y) \geq y - 1$ for $0 \leq y \leq x$, and by definition, $f(y) < y - 1$ for $x < y \leq k_0$, it follows that $\lambda + f^{(\lambda)}(k_0) \leq g^*(k_0) + 1$, where $g(k) := \min\{f(k), k - 1\}$. Therefore,

$$\mathbf{E}[j^* \mid \mathcal{E}_0] \leq g^*(k_0) + 1. \tag{6}$$

In the following we will assume that $\mathcal{E}_0$ is fixed, thus so is k_0.

Next we will bound the expectation of the maximum number of steps any single process takes on the group election objects. This number is bounded by $\sum_{1 \leq j \leq j^*} t_j$, where t_j is the max-step complexity of $G[j]$ in $\mathcal{E}$. We will bound the expectation of this sum using a version of Wald's Theorem (note that the number j^* of terms in the sum as well as the terms t_j are random variables.) From the assumption that the max-step complexity of $G[j]$ is bounded by a function t of the max-contention on $G[j]$, we have that

$$\mathbf{E}[t_j \mid \mathcal{E}_j] \leq t(k_j).$$

Since $k_j \leq k_0$ and t is a non-decreasing function, it follows $\mathbf{E}[t_j \mid \mathcal{E}_j] \leq t(k_0)$. This implies

$$\mathbf{E}[t_j \mid j^* \geq j] \leq t(k_0),$$

as the execution prefix $\mathcal{E}_j$ is sufficient to determine whether or not $j^* \geq j$ holds. We will use the above inequality to apply the following variant of Wald's Theorem, for random variables that are not independent. A proof of this theorem can be found, e.g., in [16].

Theorem 2 (Wald's Theorem)**.** *Let $X_1, X_2, \ldots$ be a sequence of non-negative random variables and let Y be a non-negative integer random variable such that the expectations of Y and of each X_j exist. If for all j, $\mathbf{E}[X_j \mid j \leq Y] \leq \mu$ for some $\mu \geq 0$, then $\mathbf{E}[X_1 + \cdots + X_Y] \leq \mu \cdot \mathbf{E}[Y]$.*

We apply the theorem for $X_j = t_j$, $Y = j^*$, and $\mu = t(k_0)$ to obtain

$$\mathbf{E}\left[\sum_{1 \leq j \leq j^*} t_j\right] \leq t(k_0) \cdot \mathbf{E}[j^*].$$

Object `GroupElect`
/* $\ell := \lceil \log n \rceil$ */
shared: register $R[1 \dots \ell+1] \leftarrow [0 \dots 0]$

Method `elect()`

1 Choose $x \in \{1, \dots, \ell\}$ at random such that $\Pr(x = i) = 2^{-i}$ for $1 \le i < \ell$, and $\Pr(x = \ell) = 2^{-\ell+1}$
2 $R[x]$.`write(1)`
3 **if** $R[x+1]$.`read()` $= 0$ **return** `win`
4 **return** `lose`

Figure 4: A group election implementation for the location-oblivious adversary model.

Using the same argument we can also bound the expectation of $\sum_{1 \le j \le j^*} t'_j$, where t'_j is the max-step complexity of the $\texttt{TAS}_2$ object $T[j]$ in $\mathcal{E}$. For $T[j]$ we have that its expected max-step complexity is constant (against any adversary), i.e., $\mathbf{E}[t'_j \mid \mathcal{E}'_j] = O(1)$, for the prefix $\mathcal{E}'_j$ of $\mathcal{E}$ until some process is poised to invoke $T[j]$.`TAS()`. Then the same reasoning as above yields

$$\mathbf{E}\left[\sum_{1 \le j \le j^*} t'_j\right] = O(1) \cdot \mathbf{E}[j^*].$$

Finally, the number of remaining steps of a process in $\mathcal{E}$, that are not steps on one of the objects $G[j]$ or $T[j]$, is bounded by $O(j^*)$.

Therefore the expected max-step complexity of the TAS implementation is bounded by

$$t(k_0) \cdot \mathbf{E}[j^*] + O(1) \cdot \mathbf{E}[j^*] + O(\mathbf{E}[j^*]) \overset{(6)}{=} O\big(t(k_0) \cdot g^*(k_0)\big).$$

Finally, note that for the above analysis we do not need any assumptions on the effectiveness of objects $G[j]$ for $j > g^*(n)$, as (5) is used only for $j := \lambda \le g^*(k_0)$. This completes the proof of Theorem 1. □

3.3 Group Election for Location-Oblivious Adversaries

We present a simple randomized group election implementation from registers, which has effectiveness $O(\log k)$ in the location-oblivious adversary model, and constant max-step complexity. This can be used to implement a TAS object with expected max-step complexity $O(\log^* k)$ against location-oblivious adversaries.

The group election implementation is given in Figure 4. Each process first writes to a random register among the $\ell := \lceil \log n \rceil$ registers $R[1], \dots, R[\ell]$, where $R[i]$ is chosen with probability $1/2^i$ if $1 \le i < \ell$, and with probability $1/2^{\ell-1}$ if $i = \ell$. Then the process reads the next register, $R[i+1]$, and gets elected if and only if no process has previously written to that register.

We have that at least one process gets elected, namely a process that writes to the rightmost register that gets written. The idea for the $O(\log k)$ bound on the effectiveness is as follows. Since the probability that a process chooses index $i+1$ equals half the probability it chooses i, at most a constant expected number of processes write to $R[i]$ before some process writes to $R[i+1]$. After a process has written to $R[i+1]$, no process that writes to $R[i]$ can still get elected. Therefore, for every index i there will only be a constant expected number of processes that choose that index and get elected. Moreover, if at most k processes participate in the group election, then with sufficiently high probability (in k) only the first $O(\log k)$ registers get written at all. A simple calculation then shows that only an expected number of $O(\log k)$ processes get elected.

Lemma 3. *Figure 4 gives a randomized implementation of a group election object with effectiveness at most* $2\log k + 4$ *and constant max-step complexity against any location-oblivious adversary.*

Proof. Let M be an algorithm that uses the implemented group election object, and consider any execution of M in which all processes participating in the group election finish their `elect()` call. Let i^* be the largest index such that some process p writes to $R[i^*]$ (in line 2). Then p reads the value 0 from $R[i^*+1]$ in the next line and returns `win`. Hence, at least one process gets elected. Further, each process does exactly two shared memory operations, thus the max-step complexity is constant. It remains to bound the effectiveness of the group election object.

Let A be a location-oblivious adversary, and let $\mathcal{E} := \mathcal{E}_{M,A}$ be a random execution of M scheduled by A. Fix the prefix $\mathcal{E}'$ of $\mathcal{E}$ until the first process is poised to invoke `elect()`, and let $k := k_{max}^{M,\texttt{elect()}}(\mathcal{E}') = k_{max}^{M,\texttt{elect()}}(\mathcal{E})$ be the max-contention of `elect()` in $\mathcal{E}$. Let $k' \le k$ be the actual number of processes that execute the write operation in line 2 during $\mathcal{E}$, and for $1 \le i \le k'$, let p_i be the i-th process to execute the write operation.

Since adversary A is location-oblivious, it does not know the index of the register on which p_i will write, before p_i finishes that operation. We can thus assume that a list $x_1, \ldots, x_k$ of indices is chosen in advance, right after the last step of $\mathcal{E}'$, such that each index x_i is drawn independently at random according to the distribution in line 1, and then for each $1 \le i \le k'$, process p_i writes to register $R[x_i]$ in line 2. Note that although just the first k' of the values x_i are actually used, we draw $k \ge k'$ values initially as k' may not be known in advance.

For each $1 \le i \le k'$, let X_i be the 0/1 random variable that is 1 if and only if p_i gets elected, i.e., p_i reads the value 0 on register $R[x_i+1]$ in line 3, and returns `win`. Further, for each $1 \le i \le k$, let Y_i be the 0/1 random variable that is 1 if and only if $x_j \ne x_i + 1$ for all $j < i$. Clearly, $X_i \le Y_i$ for $i \le k'$, as p_i reads the value 0 only if none of the processes $p_1, \ldots, p_{i-1}$ writes to register $R[x_i+1]$. The expected number of processes that get elected is then

$$\mathbf{E}\left[\sum_{1\le i\le k'} X_i\right] \le \mathbf{E}\left[\sum_{1\le i\le k} Y_i\right] = \sum_{1\le i\le k} \mathbf{E}[Y_i]. \tag{7}$$

Using that $x_1, \ldots, x_i$ are chosen independently $(*)$, and that $x_i = \ell$ implies $Y_i = 1$ $(\dagger)$, we obtain

$$\begin{aligned}
\mathbf{E}[Y_i] &= \Pr\left(\bigwedge_{1\le j<i} (x_j \ne x_i + 1)\right) \\
&= \sum_{1\le x\le \ell} \Pr\left((x_i = x) \wedge \bigwedge_{1\le j<i} (x_j \ne x+1)\right) \\
&\overset{(\dagger)}{=} \sum_{1\le x<\ell} \Pr\left((x_i = x) \wedge \bigwedge_{1\le j<i} (x_j \ne x+1)\right) + \Pr(x_i = \ell) \\
&\overset{(*)}{=} \sum_{1\le x<\ell} \Pr(x_i = x) \prod_{j=1}^{i-1} \Pr(x_j \ne x+1) + \frac{1}{2^{\ell-1}} \\
&= \sum_{1\le x<\ell} \frac{1}{2^x}\left(1 - \frac{1}{2^{x+1}}\right)^{i-1} + \frac{1}{2^{\ell-1}}.
\end{aligned}$$

Substituting that to (7) yields

$$\begin{aligned}
\mathbf{E}\left[\sum_{1\le i\le k'} X_i\right] &\le \sum_{1\le j\le k}\left(\sum_{1\le i<\ell}\frac{1}{2^i}\left(1-\frac{1}{2^{i+1}}\right)^{j-1}+\frac{1}{2^{\ell-1}}\right)\\
&= \sum_{1\le i<\ell}\frac{1}{2^i}\sum_{1\le j\le k}\left(1-\frac{1}{2^{i+1}}\right)^{j-1}+\sum_{1\le j\le k}\frac{1}{2^{\ell-1}}\\
&= \sum_{1\le i<\ell}\frac{1}{2^i}\cdot\frac{1-\left(1-\frac{1}{2^{i+1}}\right)^k}{1/2^{i+1}}+\frac{k}{2^{\ell-1}}\\
&= 2\sum_{1\le i<\ell}\left(1-\left(1-\frac{1}{2^{i+1}}\right)^k\right)+\frac{k}{2^{\ell-1}}.
\end{aligned}$$

We bound the sum in the last line by bounding with 1 each of the first $\log k$ terms, and using for the remaining terms that $1-\left(1-\frac{1}{2^{i+1}}\right)^k \le 1-\left(1-\frac{k}{2^{i+1}}\right)=\frac{k}{2^{i+1}}$. We get

$$\begin{aligned}
\mathbf{E}\left[\sum_{1\le i\le k'} X_i\right] &\le 2\sum_{1\le i<\log k} 1+2\sum_{\log k\le i<\ell}\frac{k}{2^{i+1}}+\frac{k}{2^{\ell-1}}\\
&\le 2\log k+2\frac{k}{2^{\log k}}+\frac{k}{2^{\ell-1}}\\
&\le 2\log k+4,
\end{aligned}$$

as $\ell=\lceil\log n\rceil\ge\log k$. This completes the proof of Lemma 3. □

We can now apply Theorem 1 to obtain the following result.

Theorem 4. *There is a randomized implementation of a TAS object from $\Theta(n)$ registers with expected max-step complexity $O(\log^* k)$ against any location-oblivious adversary.*

Proof. We consider the TAS implementation of Figure 3, and use the algorithm in Figure 4 to implement the group election objets $G[j]$, for $1\le j\le 2\log^* n$. For $2\log^* n<j\le n$, we just let $G[j]$ be a trivial group election object, where all participating processes get elected and the max-step complexity is zero. From Lemma 3, the group election objects $G[j]$, for $1\le j\le 2\log^* n$, have constant max-step complexity, and effectiveness bounded by $f(k)=2\log k+4$ against any location-oblivious adversary. For $g(k):=\min\{2\log k+4,k-1\}$, we have $g^*(k)=\log^* k+O(1)<2\log^* n$. Theorem 1 then implies that the resulting TAS object has expected max-step complexity $O(\log^* k)$ against any location-oblivious adversary. Moreover the algorithm uses $\Theta(n)$ registers, as each of the first $2\log^* n$ group election objection requires $\log n+O(1)$ registers, while the remaining trivial group election objects do not use any registers. □

3.4 Group Election for R/W-Oblivious Adversaries

We present a randomized group election implementation from registers, which has constant effectiveness and expected max-step complexity $O(\log\log k)$ in the r/w-oblivious adversary model. This can be used to implement a TAS object with expected max-step complexity $O(\log\log k)$ against r/w-oblivious adversaries.

The group election implementation is given in Figure 5. The algorithm consists of two phases, the *backward sifting phase* and the *forward sifting phase*. The latter phase is similar to a sifting

Object `GroupElect`
/* $b := \frac{3}{2}$ `and` $\ell := \lceil \log_b \log n \rceil$ */
shared: register $Up[1 \dots \ell] \leftarrow [0 \dots 0]$, $Down[1 \dots \ell - 1] \leftarrow [0 \dots 0]$

Method `elect()`

```
1  i ← 0
2  repeat
3  |  i ← i + 1
4  |  Choose c_i ∈ {heads, tails} at random such that Pr(c_i = heads) = q_i := 1/2^(b^(i−1))
5  |  if c_i = heads then
6  |  |  Up[i].write(1)
7  |  else
8  |  |  if Up[i].read() = 1 return lose
9  |  end
10 until c_i = tails or i = ℓ
11 while i > 1 do
12 |  i ← i − 1
13 |  Choose c'_i ∈ {heads, tails} at random such that Pr(c'_i = heads) = q_i
14 |  if c'_i = heads then
15 |  |  Down[i].write(1)
16 |  else
17 |  |  if Down[i].read() = 1 return lose
18 |  end
19 end
20 return win
```

Figure 5: A group election implementation for the r/w-oblivious adversary model.

procedure used to eliminate processes in the TAS algorithm by Alistarh and Aspnes [2]. Their algorithm, however, is not adaptive. To achieve that, the backward sifting phase runs essentially the same sifting procedure but in the opposite direction.

Two shared arrays of registers are used, one in each phase, namely, $Up[1 \dots \ell]$ and $Down[1 \dots \ell - 1]$, where $\ell = \lceil \log_b \log n \rceil$ and $b = \frac{3}{2}$. All entries in both arrays are initially 0.

In the backward sifting phase, for each $i = 1, 2, \dots$, each process p decides at random to either read register $Up[i]$ or to write the value 1 to it. The probability of writing decreases with i, more precisely, it is $q_i = 1/2^{b^{i-1}}$. The phase ends for p as soon as it has executed a read operation or has written to all registers of Up. If p reads the value 1 on $Up[i]$, it means that some other process has written to $Up[i]$ before, and p returns `lose` immediately. If p reads 0 on $Up[i]$, then it moves on to the forward sifting phase. If p writes to $Up[i]$ instead, then it continues to the next element of Up if $i < \ell$, or if p has already reached the end of array Up, it moves on to the forward sifting phase.

Suppose that process p reaches the forward sifting phase after reading the value 0 on register $Up[i_p]$ for some index $i_p \in \{1, \dots, \ell\}$, or after writing the value 1 on register $Up[i_p]$ for $i_p = \ell$. Then, for each $i = i_p - 1, i_p - 2, \dots, 1$, processes p either reads register $Down[i]$ or writes the value 1 to it. As before, the decision is made at random and the probability of writing is q_i. If p reads the value 1, it returns `lose`. If p writes to $Down[i]$ or reads 0 from it, then p continues to $Down[i-1]$ if $i > 1$, or p returns `win` if $i = 1$.

Let k be the maximum number of processes participating in the group election. Then with high probability no process accesses a register of array Up beyond the first $O(\log \log k)$ registers, because for larger indices i the probability q_i of writing to $Up[i]$ is polynomially small in k. This implies the $O(\log \log k)$ bound on the expected max-step complexity. The bound on the effectiveness is

obtained as follows. We have that the number r_i of processes that move from the backward to the forward sifting phase after reading register $Up[i]$ is in expectation bounded by $1/q_i$: Each of those r_i processes must read register $Up[i]$ before any process has written to $Up[i]$, and the probability of writing to that register is q_i. We show by an inductive argument that the number s_i of processes that access $Down[i]$ and do not return `lose` right after the operation is $O(1/q_i)$ in expectation, thus the number $s_1 + r_1$ of processes that get elected is $O(1)$ in expectation. The inductive argument goes as follows: The number of processes that access $Down[i]$ is $s_{i+1} + r_{i+1}$ (where s_ℓ is defined as the number of processes that write to $Up[\ell]$). The expectation of $s_{i+1} + r_{i+1}$ is $O(1/q_{i+1})$, by the induction hypothesis and the earlier observation that r_i is bounded by $1/q_i$. The first write operation on $Down[i]$ occurs in expectation after $1/q_i$ accesses, and after that only processes that write to $Down[i]$ do not return `lose`. So in total the expected number of processes that do not return `lose` after accessing $Down[i]$ is at most $1/q_i$ plus the fraction $q_i(r_{i+1} + s_{i+1})$ of processes that write to $Down[i]$. A simple calculation bounds that by $O(1/q_i)$.

Lemma 5. *Figure 5 gives a randomized implementation of a group election object with effectiveness at most* 16 *and expected max-step complexity* $O(\log\log k)$ *against any r/w-oblivious adversary.*

Proof. Let M be an algorithm that uses the implemented group election object, and consider any execution of M. First we argue that not all `elect()` calls return `lose` in the execution. Suppose, towards a contradiction, that they all do. Then each process reads the value 1 on some register $Up[i]$ or some register $Down[i]$, and the process returns `lose` immediately after that. This implies that at least one process writes the value 1 to some register. We argue that no process writes to any of the registers $Down[i]$: Otherwise, let $i_{\min}$ be the smallest index such that some process $p_{\min}$ writes the value 1 to $Down[i_{\min}]$. But then $p_{\min}$ does not return `lose` at any point, because after writing to $Down[i_{\min}]$, $p_{\min}$ may only read registers $Down[j]$ for $j < i_{\min}$. Thus, some process must write to a register $Up[i]$. Let $i_{\max}$ be the largest index such that some process $p_{\max}$ writes the value 1 to $Up[i_{\max}]$. But then $p_{\max}$ does not return `lose` at any point, as after writing to $Up[i_{\max}]$, $p_{\max}$ may only read registers $Up[i]$ for $i > i_{\max}$, and registers $Down[i]$, for $1 \le i \le \ell - 1$, none of which has value 1.

Next we bound the effectiveness of the implementation. Let A be some r/w-oblivious adversary, and let $\mathcal{E} := \mathcal{E}_{M,A}$ be a random execution of algorithm M scheduled by A. Fix the prefix $\mathcal{E}'$ of $\mathcal{E}$ until the first process is poised to invoke `elect()`, and let $k := k_{max}^{M,\texttt{elect()}}(\mathcal{E}')$ be the max-contention of `elect()` in $\mathcal{E}$. For $1 \le i \le \ell$, let r_i be the number of processes in $\mathcal{E}$ that read register $Up[i]$ before any process writes to it. For $1 \le i \le \ell - 1$, let s_i be the number of processes that either read register $Down[i]$ before any process writes on it, or write on register $Down[i]$. We also define s_ℓ to be the number of processes that write on $Up[\ell]$. The total number of processes that access register $Down[i]$, for $1 \le i \le \ell - 1$, is then at most[1] $r_{i+1} + s_{i+1}$, and the number of processes that get elected in the group election is $r_1 + s_1$. We will show that $\mathbf{E}[r_1 + s_1] \le 16$.

Since adversary A is r/w-oblivious, it does not know whether a process poised to access a shared register will read or write to that register. We can thus assume that right after the last step of $\mathcal{E}'$, we perform for each register $Up[i]$ and each register $Down[i]$ a series of k independent coin flips with heads probability q_i, and that the j-th process to subsequently accesses that register uses the j-th coin flip value in the series to decide whether it should read or write on the register. We observe that once these series of coin flips have been fixed, the values of all random variables r_i and s_i are completely determined by the number $k' \le k$ of processes that invoke `elect()`, provided that all these k' processes finish their `elect()` call. (In particular, r_i and s_i do not depend on the order in which the k' processes are scheduled to take steps.) Moreover, if not all k' `elect()`

[1]We say 'at most' instead of 'exactly' because we do not require that all processes finish their `elect()` call in $\mathcal{E}$.

calls are executed to completion, then for each $1 \leq i \leq \ell$, r_i and s_i are smaller or equal than the corresponding values if all k' calls were executed to completion.

It follows that instead of the schedule determined by adversary A, we can consider a schedule with the following convenient properties: Exactly k of processes call the implemented `elect()` method and all processes finish their call; for each $1 \leq i < \ell$, all operations on register $Up[i]$ are scheduled before any operation on $Up[i+1]$; for each $1 < i \leq \ell - 1$, all operations on $Down[i]$ are scheduled before any operation on $Down[i-1]$; and all operations on array Up are scheduled before any operation on array $Down$. Let R_i and S_i denote the same quantities as r_i and s_i but for a schedule as described above. Then $R_i \geq r_i$ and $S_i \geq s_i$, if the same series of coin flips are used for each register under both schedules, and thus

$$\mathbf{E}[R_1 + S_1] \geq \mathbf{E}[r_1 + s_1].$$

We now bound $\mathbf{E}[R_1 + S_1]$. For that we no longer assume that coin flips are fixed in advance.

For each $1 \leq i \leq \ell - 1$, the values of R_{i+1} and S_{i+1} are determined before the first process accesses register $Down[i]$. It follows that

$$\mathbf{E}[S_i \mid R_{i+1}, S_{i+1}] \leq 1/q_i + q_i(R_{i+1} + S_{i+1}),$$

where the term $1/q_i$ accounts for the processes that read $Down[i]$ before any process writes on $Down[i]$, and the term $q_i(R_{i+1} + S_{i+1})$ accounts for the processes that write on $Down[i]$. Taking the unconditional expectation yields

$$\mathbf{E}[S_i] \leq 1/q_i + q_i(\mathbf{E}[R_{i+1}] + \mathbf{E}[S_{i+1}]) \leq 1/q_i + q_i(1/q_{i+1} + \mathbf{E}[S_{i+1}]). \tag{8}$$

We now show by induction on $i = \ell, \ell - 1, \ldots, 1$ that

$$\mathbf{E}[S_i] \leq 7/q_i. \tag{IH}$$

Recall that $\ell = \lceil \log_b \log n \rceil \geq \log_b \log n$, and $b = 3/2$. We also have $q_i = 1/2^{b^{i-1}}$ and thus $q_{i+1} = q_i^b$. For the base case of $i = \ell$, we have that S_ℓ is the number of processes that write to $Up[\ell]$, thus

$$\mathbf{E}[S_\ell] \leq kq_\ell = \frac{kq_\ell^2}{q_\ell} \leq \frac{k\big(1/2^{b^{\log_b \log n - 1}}\big)^2}{q_\ell} = \frac{k/n^{4/3}}{q_\ell} \leq \frac{n^{-1/3}}{q_\ell} < \frac{7}{q_\ell}.$$

For $i < \ell$, we obtain from (8) that

$$\begin{aligned}
\mathbf{E}[S_i] &\leq 1/q_i + q_i(1/q_{i+1} + \mathbf{E}[S_{i+1}]) \\
&\leq 1/q_i + q_i(1/q_{i+1} + 7/q_{i+1}), \quad \text{by (IH)} \\
&= 1/q_i + 8q_i/q_{i+1} \\
&= 1/q_i + 8q_i/q_i^b \\
&= (1/q_i)(1 + 8q_i^{2-b}) \\
&\leq (1/q_i)(1 + 8q_1^{2-b}) \\
&< (1/q_i) \cdot 7.
\end{aligned}$$

This completes the inductive proof that $\mathbf{E}[S_i] \leq 7/q_i$. Applying this inequality, for $i = 1$, we obtain

$$\mathbf{E}[R_1 + S_1] \leq 1/q_1 + 7/q_1 = 16.$$

Therefore, the effectiveness of the implemented group election is $\mathbf{E}[r_1 + s_1] \leq \mathbf{E}[R_1 + S_1] \leq 16$.

It remains to bound the expected max-step complexity of the implementation. Let i^* be the maximum index i such that some process accesses register $Up[i]$ in execution $\mathcal{E}$. Then the maximum number of shared memory operations by any process is at most $2i^* - 1$. We have that $\Pr(i^* \geq i)$ is bounded by the expected number of processes that access $Up[i]$, and this is bounded by kq_i. Thus, for $\lambda := \lceil \log_b \log k \rceil$, we have

$$\begin{aligned}
\mathbf{E}[i^*] &= \sum_{i \geq 1} \Pr(i^* \geq i) \\
&\leq \lambda + \sum_{i \geq \lambda+1} \Pr(i^* \geq i) \\
&\leq \lambda + \sum_{i \geq \lambda+1} kq_i \\
&\leq \lambda + kq_{\lambda+1} \sum_{i \geq 0} q_{\lambda+1}^{b^i - 1}.
\end{aligned}$$

Since $q_{\lambda+1} = 1/2^{b^\lambda} \leq 1/k$ and $\sum_{i \geq 0} q_{\lambda+1}^{b^i-1} \leq \sum_{i \geq 0} q_1^{b^i-1} < 3$, it follows that $\mathbf{E}[i^*] \leq \lambda + 4$. Hence, the expected max-step complexity is at most $2\,\mathbf{E}[i^*] - 1 \leq 2(\lambda + 4) - 1 = 2\lceil \log_b \log k \rceil + 7$. This completes the proof of Lemma 5. □

Theorem 6. *There is a randomized implementation of a TAS object from $\Theta(n)$ registers with expected max-step complexity $O(\log \log k)$ against any r/w-oblivious adversary.*

Proof. We consider the TAS implementation of Figure 3, and use the algorithm in Figure 5 to implement the group election objets $G[j]$, for $1 \leq j \leq 16$. For $16 < j \leq n$, we let $G[j]$ by a trivial group election object, where all participating processes get elected and the max-step complexity is zero. From Lemma 5, the group election objects $G[j]$, for $1 \leq j \leq 16$, have effectiveness at most 16 and expected max-step complexity $O(\log \log k)$ against any r/w-oblivious adversary. Theorem 1 then implies that the resulting TAS algorithm has expected max-step complexity $O(16 \cdot \log \log k)$ against any location-oblivious adversary. Moreover the algorithm uses $\Theta(n)$ registers, as each of the first 16 group election objection uses $O(\log \log n)$ registers, and the remaining trivial group election objects do not use any registers. □

4 Linear-Space TAS for Strong Adaptive Adversaries

We present a TAS implementation from $\Theta(n)$ registers that has max-step complexity $O(\log k)$ both in expectation and w.h.p. (i.e., with probability $1 - 1/k^{\Omega(1)}$), against any strong adaptive adversary. Our implementation is a variant of the `RatRace` algorithm proposed by Alistarh et al. [5], which has the same max-step complexity but uses $\Theta(n^3)$ registers.

Theorem 7. *There is a randomized implementation of a TAS object from $\Theta(n)$ registers with max-step complexity $O(\log k)$, both in expectation and w.h.p., against any strong adaptive adversary.*

Before we prove Theorem 7, we give an overview of the original `RatRace` algorithm. To simplify exposition, throughout this section we treat $\log n$, $n/\log n$, and $\log \log n$ as integers. It is easy to accommodate the calculations for the case that this is not true, by rounding appropriately.

Overview of RatRace. `RatRace` [5] uses two shared memory data structures, a *primary tree* and a *backup grid*. The primary tree is a perfect binary tree of height $3 \log n$, where each node v stores a randomized splitter object S_v, and a randomized 3-process TAS object T_v. The latter can be implemented from two 2-process TAS objects.

Each process p starts at the root of the primary tree and moves downwards towards the leaves. The process goes through the splitters at the nodes it visits along the way, until it stops at a splitter, or "falls off" the bottom of the tree (which happens only with low probability). If p turns left or right at a splitter S_v, then it moves respectively to the left or right child of v, provided v is not a leaf. If v is a leaf, p moves to the backup grid as explained below. If p stops at S_v then it stops moving downwards, and starts to move upwards towards the root, along the same path. At each node u in the path to the root, p tries to win the TAS on object T_u. If p loses that TAS, it immediately loses the implemented TAS. Otherwise, it moves to the parent of u in the tree. The process that wins the TAS at the root competes against the winner at the backup grid.

The backup grid is an $n \times n$ square grid, where each node $v = (i, j) \in \{1, \ldots, n\}^2$ stores a *deterministic* splitter object, and also a randomized 3-process TAS object as before. We define the left and right children of node (i, j) at the grid to be nodes $(i + 1, j)$ and $(i, j + 1)$, respectively. Each process that falls off the primary tree starts at node $(1, 1)$, and proceeds in a similar way as in the primary tree: At each node the process goes through the splitter, moving to the child as indicated by the direction to which the process turns at the splitter, until it stops at some splitter. Then, the process tries to move back to node $(1, 1)$ along the same path, by winning all the TAS in the nodes along the way. The properties of deterministic splitters guarantee that the process wins a splitter before it falls off the grid.

The winner of the TAS at node $(1, 1)$ of the backup grid, and the winner of the TAS at the root of the primary tree participate in a randomized 2-process TAS, which determines the winner of `RatRace`.

To ensure linearizability, a doorway object is used such that only processes that pass through the doorway participate in the above algorithm, whereas processes that are deflected lose immediately.

Reducing the Space Complexity (Proof of Theorem 7). `RatRace` requires $\Theta(2^{3 \log n}) = \Theta(n^3)$ registers for the primary tree of height $3 \log n$, and $\Theta(n^2)$ registers for the backup $n \times n$ grid. Next we show how to reduce this space complexity, without increasing the max-step complexity.

We use a data structure, which we call an *elimination path*, that is similar to the backup grid but uses fewer registers. An *elimination path of length* ℓ is an ℓ-node path where each node $i \in \{1, \ldots, \ell\}$ stores a deterministic splitter S_i, and a randomized 2-process TAS object T_i. The possible outcomes for a process accessing an elimination path is to *win*, *lose*, or *fall off* the path. A process p enters the elimination path at node $i = 1$, and moves towards node ℓ, going through splitter S_i at each node i it visits. If p turns left at S_i, then it *loses* and takes no more steps. If it turns right, then it moves to the next node, $i+1$, if $i < \ell$, whereas if $i = \ell$, p *falls off* the path and takes no more steps in the path. Last, if p stops at S_i, then it starts moving back towards node 1. From node $i > 1$, it moves to $i - 1$ if it wins the TAS on T_i, otherwise, it loses and stops. The *winner* of the elimination path is the winner of T_1.

With some slight modifications, the TAS algorithm in Figure 3 implements an elimination path of length n. More precisely, we remove line 1, where the process accesses the doorway, and replace line 5, where the process participates in a group election, with the statement: **if** $i > n$ **return** `fall-off`. The process wins (loses) if the return value is 0 (respectively, 1).

The next lemma summarizes the main properties of an elimination path.

Lemma 8. *At most one process wins in an elimination path, and not all processes that access the elimination path lose. If $k \leq \ell$ processes access an elimination path of length ℓ, then no process visits a node with index $j > k$, and no process falls off.*

Proof. The properties that at most one process wins and not all processes lose follow from the same properties of the TAS implementation in Figure 3. For the second part of the lemma, we have that at each splitter, not all processes can turn right. Hence, if at most k processes enter the elimination path, then at most $k - i$ processes turn right at splitter S_i, for $i \leq k$. This implies that no process visits a node with index $j > k$, and that no process falls off the end of the path. □

To reduce the space complexity of the `RatRace` algorithm, the first modification we make is to replace the backup grid by a *backup elimination path* B of length n. Lemma 8 implies that B has the same properties as the backup grid against a strong adaptive adversary. Unlike the backup grid however, B requires only $\Theta(n)$ registers.

A second modification is that we replace the primary tree of height $3 \log n$, by a data structure consisting of a smaller primary tree, of height $\log n - \log \log n$, and $n/\log n$ elimination paths P_i of length $4 \log n$, where $1 \leq i \leq n/\log n$. Note that we have as many elimination paths as the leaves of the primary tree. The total number of registers required is $\Theta(2^{\log n - \log \log n} + (4 \log n) \cdot n/\log n) = \Theta(n)$. The primary tree is used in the same way as before, but now any process that falls off moves to one of the elimination paths, instead of the backup grid. More precisely, a process that falls off the i-th leaf moves to elimination path P_i. The winner at each P_i (if there is one) moves back to the primary tree, at leaf i, and from there it tries to reach the root as in the original `RatRace` algorithm. Any process that falls off a path P_i moves to the backup elimination path B. Finally, as before, the winner of B and the winner of the primary tree participate in a 2-process TAS to determine the winner of the implemented TAS.

Consider a random execution of an algorithm that uses the above TAS implementation, scheduled by a strong adaptive adversary. Fix the prefix of this execution until the first process is poised to invoke the implemented TAS, and suppose the max-contention is k.

If $\log k \leq (\log n - \log \log n)/3$, then a bound of $O(\log k)$ on the expected max-step complexity, and also on the max-step complexity w.h.p., follows from the analysis of the original `RatRace` [5].

In the following we assume that $\log k > (\log n - \log \log n)/3$. We use the next simple lemma, which implies that w.h.p. the number of processes that enter each elimination path P_i is not greater than its length.

Lemma 9. *With probability at least $1 - 1/n$, each leaf node in the primary tree is visited by at most $4 \log n$ processes.*

Proof. The number of processes that visit a given leaf node is stochastically dominated by the number of balls that fall in a given bin in the standard bins-and-balls model, with n balls and $n/\log n$ bins. In this model each ball is placed in a bin chosen independently and uniformly at random. The domination follows because we can assume each process p comes with an independent and uniform random bit string of length $\log n - \log \log n$. If p goes through a randomized splitter in a node at distance $i - 1$ from the root, and does not stop at that splitter, then the i-th bit in the bit string determines whether p will turn left or right at the splitter. Hence, the random bit string uniquely determines the leaf that p will reach, if it does not stop at any splitter along the way.

For $1 \leq i \leq n$, let X_i be the 0/1 random variable that is 1 if and only if the i-th ball falls in some fixed bin b. Let $X = X_1 + \cdots + X_n$ be the total number of balls that fall in b. Then $\mathbf{E}[X] = \log n$, and by a standard Chernoff bound, stated as Theorem 10 below, we obtain

$$\Pr(X > 4 \log n) \leq e^{-\frac{3^2 \log n}{2(1+1)}} < n^{-2}.$$

Therefore, the same n^{-2} upper bound applies to the probability that more than $4\log n$ processes visit a given leaf node. Then by a union bound, the probability that the maximum number of visits at any of the $n/\log n$ leaves exceeds $4\log n$ is at most $n^{-1}/\log n$. □

The following Chernoff Bound, used in the proof above, can be found in [18, Theorem 2.3(b)].

Theorem 10 (Chernoff Bound)**.** *Let $X_1, X_2, \ldots, X_n$ be independent random variables with $0 \leq X_i \leq 1$, for each $i \in \{1, \ldots, n\}$, and let $X = X_1 + \cdots + X_m$ and $\mu = \mathbf{E}[X]$. Then for any $\delta > 0$,*

$$\Pr\big(X \geq (1+\delta)\mu\big) \leq e^{-\frac{\delta^2 \mu}{2(1+\delta/3)}}.$$

From Lemma 9, we have that w.h.p. no more than $4\log n$ processes enter any single elimination path P_i, and thus w.h.p. no process enters the backup elimination path B, by Lemma 8. If no process enters B, then each process traverses at most a path of length $\log n - \log\log n$ in the primary tree (from the root to a leaf), and at most one of the elimination paths P_i of length $4\log n$. Therefore, each process goes through at most $O(\log n)$ splitters, and participates in at most $O(\log n)$ 3-process TAS objects. It follows that the max-step complexity is bounded by $O(\log n) = O(\log k)$ w.h.p. Since w.h.p. no process reaches B, and the maximum number of steps a process takes at B is $O(n)$ w.h.p., it follows that the expected max-step complexity is bounded by $O(\log k)$, as well. This completes the proof of Theorem 7. □

5 Combining TAS Algorithms for Different Adversaries

`RatRace` and its linear-space variant presented in Section 4 achieve logarithmic max-step complexity in the strong adaptive adversary model. These algorithms do not benefit from weaker adversaries, as their expected max-step complexity is still logarithmic even in the oblivious adversary model. On the other hand, the TAS implementations in Section 3, which are more efficient against weaker adversaries, exhibit poor performance in the strong adaptive adversary model, having linear expected max-step complexity. In this section we describe how one can combine any of the implementations in Section 3 with `RatRace`, to obtain a TAS object that has the expected max-step complexity of `RatRace` against any strong adaptive adversary, and the expected max-step complexity of the corresponding algorithm in Section 3 in the weaker adversary model.

Theorem 11. *For any randomized TAS implementation* `Imp`*, there is a randomized TAS implementation* `Comb` *that has the following properties:*

(a) If f is a non-decreasing function such that the expected max-step complexity of `Imp` *is at most $f(k)$ against any location-oblivious (or r/w-oblivious) adversary, then* `Comb` *has expected max-step complexity $O\big(f(k)\big)$ against any location-oblivious (respectively r/w-oblivious) adversary;*

(b) `Comb` *has expected max-step complexity $O(\log k)$ against any strong adaptive adversary; and*

(c) The space complexity of `Comb` *is $\Theta(n)$ plus the space complexity of* `Imp`*.*

Combining Theorem 11 with Theorems 4 and 6, yields the following result.

Corollary 12. *There are randomized implementations of TAS objects from $\Theta(n)$ registers with expected max-step complexity $O(\log^* k)$ or $O(\log\log k)$ against any location-oblivious adversary or any r/w-oblivious adversary, respectively, and with expected max-step complexity $O(\log k)$ against any strong adaptive adversary.*

5.1 Proof of Theorem 11

5.1.1 Implementation

We present a TAS implementation, `Comb`, which achieves the step and space complexities stated in Theorem 11. Each process first enters a doorway D, and the processes that get deflected lose immediately. A process that passes through D, then runs both `Imp` and a variant of `RatRace`, in parallel. The only difference of the `RatRace` variant used from the original `RatRace` is that its initial doorway is removed. More precisely, after passing through D, each process executes a step of `Imp` in every odd step, and a step of `RatRace` (without doorway) in every even step.

A natural way to combine the two interleaved executions would be that each process takes steps until it either wins or loses in one of the two algorithms; if it loses it also loses in the combined implementation, and if it wins in one of the two algorithms it competes against the winner of the other algorithm. This approach, however, could yield an execution in which no process wins. For instance, suppose that `Imp` is also an instance of `RatRace`. In an execution in which only two processes, p and q, participate, process p might loses against q on one of the 2- or 3-process TAS objects in the first instance of `RatRace`, and at the same time q may lose against p on a TAS object in the second instance of `RatRace`; thus all processes lose.

To solve this problem we impose the rule that if a process loses in `Imp` at a point when it has already stopped at some splitter object in `RatRace`, then the process continues to execute `RatRace`. More precisely, we use the rules below to combine the two executions, with the help of an auxiliary 2-process TAS object T_{top}.

(C1) If a process wins either `RatRace` or `Imp`, then it stops taking steps in the other algorithm, and tries to win T_{top}; if it wins T_{top} then it wins the implemented TAS object, otherwise it loses.

(C2) If a process loses `RatRace` then it stops taking steps in `Imp`, and it loses the implemented TAS object.

(C3) If a process loses `Imp` while it has a pending `split()` call on a (randomized or deterministic) splitter of `RatRace`, then it keeps taking steps in `RatRace`, until its pending `split()` operation completes. Once it has no more pending `split()` operation it does one of the following:

- (C3a) If it has not yet stopped at any of the splitter objects in `RatRace`, then it stops taking steps in `RatRace`, and it loses the implemented TAS object.
- (C3b) If it has already stopped at one of the splitter objects in `RatRace`, then it continues taking steps in `RatRace` until `RatRace` finishes, and it either wins or loses `RatRace`. If it wins `RatRace`, then it proceeds as in (C1); otherwise it loses the implemented TAS.

We now prove that `Comb` is a correct (linearizable) TAS implementation, and then we show that it satisfies properties (a)–(c) of Theorem 11.

5.1.2 Correctness

A process accesses T_{top} if and only if it wins either `RatRace` or `Imp`. It follows that at most two processes can execute the `TAS()` operation on T_{top}, one that won `RatRace` and one that won `Imp`, and thus at most one process can win `Comb`. In the following we show that in any execution in which all processes complete their `TAS()` call, at least one process wins (therefore exactly one process wins). Due to the initial doorway D, linearizability follows from exactly the same arguments as for the algorithm in Section 3.2 (see the correctness proof of Theorem 1).

For the purpose of a contradiction, consider an execution $\mathcal{E}$ in which all participating processes take sufficiently many steps to finish `Comb`, and they all lose `Comb`. Let Q be the set of processes that stop at some `RatRace` splitter in $\mathcal{E}$.

First suppose that Q is empty. Then no process wins or loses `RatRace`, as otherwise it would have first stopped at some splitter, and thus it would be in Q. Hence, by (C1)–(C3) all processes execute `Imp` to completion, and either win or lose `Imp` eventually. Then, by the assumption that `Imp` is a correct TAS algorithm, exactly one process wins `Imp`, and this process also wins T_{top}, since no process wins `RatRace`, and hence no other process participates in a `TAS()` operation on T_{top}. This contradicts the assumption that all processes lose `Comb`.

Now suppose that Q is not empty, that is, in execution $\mathcal{E}$ at least one process stops at a splitter of `RatRace`. By the assumption that all processes lose `Comb` in $\mathcal{E}$, there is no process that wins either `Imp` or `RatRace` (otherwise that process would execute a `TAS()` operation on T_{top} and some process would win T_{top}, and thus `Comb`). In particular, no process in Q wins `RatRace`, and since by (C3b), each $q \in Q$ does not stop taking steps in `RatRace` even after losing `Imp`, it must lose `RatRace` at some point. Hence, each process in Q loses a `TAS()` operation on some 2- or 3-process TAS object of `RatRace`. Recall that the TAS objects used by `RatRace` are arranged in a rooted tree (where we consider the elimination paths as part of the tree). Whenever a process wins a non-root TAS object T of that tree, it continues to the parent of T. Among all TAS objects on which processes in Q lose, let T^* be one that is closest to the root. Then there must be a process $q \in Q$ that wins T^*, so q ascends to the parent of T^*. Then q must lose on some other TAS object closer to the root than T^*, which contradicts the definition of T^*.

5.1.3 Complexity

The linear space complexity of `Comb` claimed in part (c) of Theorem 11 follows immediately from the construction and our `RatRace` implementation given in Section 4, which uses $\Theta(n)$ registers (Theorem 7). We now analyze the expected max-step complexity of `Comb`.

High Level Idea. We first describe the general idea for bounding the expected max-step complexity of `Comb`, ignoring some of the subtleties that arise in the detailed analysis to follow. We relate the expected max-step complexity of `Comb` to the expected max-step complexity of `RatRace` and `Imp`, respectively, depending on what adversary model is used. Note that the 2-process TAS object T_{top} has constant expected max-step complexity even against a strong adaptive adversary, so it does not affect the asymptotic max-step complexity of `Comb`. Recall that during `Comb` processes alternate between steps of `RatRace` and `Imp` until one of those two algorithms terminate, and if `RatRace` terminates first, then the calling process also terminates its `Imp` call (but not necessarily the other way around). Therefore, the asymptotic max-step complexity of `Comb` is dominated by that of `RatRace`. Hence, if a random execution of k processes calling `Comb` is scheduled by a strong adaptive adversary, then the maximum number of steps any process takes is $O(\log k)$, which is the upper bound for `RatRace` as stated in Theorem 7.

Now suppose such a random execution is scheduled by a location-oblivious or r/w-oblivious adversary. It suffices to show that the expected maximum number of steps any process devotes to `RatRace` during `Comb` is bounded asymptotically by the expected maximum number of steps any process devotes to `Imp`. A process can devote more steps to `RatRace` than to `Imp` only if, by the time it finishes `Imp`, it has either already stopped at a splitter in `RatRace`, or it has a pending `split()` call that will return `stop`. Hence, it suffices to consider processes that stop at `RatRace` splitters. Suppose a process stops at a `RatRace` splitter in its i-th `split()` operation. Since the process alternates between `RatRace` and `Imp` steps prior to its last `split()` operation, and each

`split()` operation takes a constant number of steps, until finishing its i-th `split()` operation the process devotes $\Theta(i)$ steps to `RatRace` and $\Theta(i)$ steps to `Imp`. In the remainder of its `RatRace` execution, the process executes at most $i+1$ `TAS()` calls on 2- or 3-process TAS objects (one for each splitter it went through previously, in addition to T_{top}). The number of steps for each such `TAS()` call is bounded by a geometrically distributed random variable. Using Chernoff Bounds, we show that with probability $1 - 1/4^i$ the process needs only $O(i)$ steps for its at most $i+1$ `TAS()` calls to finish `RatRace` after stopping at the i-th splitter. Due to the arrangements of splitters in a primary tree and elimination paths, at most 2^i processes can stop after their i-th `split()` operation. Thus, by a union bound applied to all processes stopping at the i-th splitter they go through, with probability exponentially close to 1, all these processes need only $O(i)$ steps to finish `RatRace`. To summarize: all processes that stop at their i-th splitter devote $\Omega(i)$ steps of `Comb` to `Imp`, $\Theta(i)$ steps to `split()` operations during `RatRace`, and with high probability $O(i)$ steps to the remainder of `RatRace`. Hence, by the union bound applied to all $i > 0$, the expected maximum number of steps any process needs for `RatRace` is asymptotically bounded by the number of steps it devotes to `Imp`.

Detailed Analysis. First, we modify `Comb` such that there is no initial doorway D, and processes do not access the 2-process TAS object T_{top} after winning `RatRace` or `Imp`. Instead, a process simply terminates if it wins `RatRace` or `Imp`. Since the expected max-step complexity of T_{top} is constant, removing T_{top} does not affect the asymptotic expected max-step complexity. We will refer to this modified algorithm as `Comb`$'$.

Consider an execution prefix $\mathcal{E}$ of an algorithm M that uses `Comb`, where $\mathcal{E}$ ends when the first process exits doorway D, and suppose P is the set of processes that enter D during $\mathcal{E}$. Then the max-contention $k_{\max}^{M,\texttt{Comb}}(\mathcal{E})$ is at least $|P|$. Hence, it suffices to show for any set P, that a random execution of `Comb`$'$ by the processes in P has expected max-step complexity $O\big(f(|P|)\big)$ if scheduled by a location-oblivious (or r/w-oblivious) adversary, and $O(\log |P|)$ if scheduled by a strong adaptive adversary.

To that end, let M_C be the algorithm in which the processes in P (and only them) call `Comb`$'$, and let A_C be some adversary. A scheduling of M_C by A_C yields a random execution in which a subset of the processes in P take steps (the max-congestion in that execution is $|P|$). For two random coin flip vectors $\omega_I, \omega_R \in \Omega^\infty$, let $\mathcal{E}_C$ denote the random execution of M_C scheduled by A_C, where the i-th coin flip result obtained during the execution of `Imp` and `RatRace` within `Comb`$'$ is the i-th element of ω_I and ω_R, respectively. For a process $p \in P$, let T_C^p denote the number of steps p executes in $\mathcal{E}_C$, and let T_I^p and T_R^p denote the number of those steps that are devoted to `Imp` and `RatRace`, respectively. Let $T_C = \max_{p\in P} T_C^p$, $T_I = \max_{p\in P} T_I^p$, and $T_R = \max_{p\in P} T_R^p$. Then $\mathbf{E}[T_C]$ is the expected max-step complexity of `Comb`$'$ in M_C against A_C.

Lemma 13. *There are constants $d_I, d_R > 0$ such that*

$$\mathbf{E}[T_C] \le d_I \cdot (\mathbf{E}[T_I] + 1), \text{ and} \tag{9}$$

$$\mathbf{E}[T_C] \le d_R \cdot (\mathbf{E}[T_R] + 1). \tag{10}$$

Before we prove Lemma 13, we argue that it implies parts (a) and (b) of Theorem 11.

To prove part (a), we assume that adversary A_C is location-oblivious (or r/w-oblivious). Let M_I be the algorithm in which the process in P call `Imp`.

We construct a location-oblivious (or r/w-oblivious) adversary A_I that schedules M_I by simulating adversary A_C as follows. Let $\omega_R^* \in \Omega^\infty$ be a coin flip sequence such that $E[T_I \mid \omega_R = \omega_R^*]$ is maximized. To schedule an execution of algorithm M_I, adversary A_I simulates adversary A_C on

algorithm M_C, using the i-th element of ω_R^* for the i-th coin flip used in `RatRace`. By the structure of `Comb'`, in which processes alternate steps of `RatRace` and `Imp`, it is uniquely determined when a process p executes its i-th step of `Imp`. Therefore, even the location-oblivious (or r/w-oblivious) adversary can simulate all steps of `RatRace` in $\mathcal{E}_C$, and schedule processes to take steps in M_I exactly in the same order as they take steps in the `Imp` portion of $\mathcal{E}_C$.

Let τ_I denote the expected max-step complexity of `Imp` against adversary A_I. Then we have $E[T_I \mid \omega_R = \omega_R^*] \leq \tau_I(|P|)$. Since ω_R^* is chosen to maximize the conditional expectation on the left side, it follows that $E[T_I] \leq \tau_I(|P|)$. Moreover, by the theorem's assumption that the expected max-step complexity of `Imp` is bounded by f, we have $\tau_I(|P|) \leq f(|P|)$. From the last two inequalities and (9), we obtain $E[T_C] \leq d_I \cdot (f(|P|) + 1) = O(f(P))$. As this is true for all sets P, and any location-oblivious (or r/w-oblivious) adversary A_C, it proves part (a) of Theorem 11.

The proof of part (b) is almost identical: We now assume A_C is a strong adaptive adversary. We construct a strong adaptive adversary A_R which schedules processes in P to execute `RatRace` by simulating adversary A_C on algorithm M_C, assuming the worst-case vector ω_I. As before, we argue that $E[T_R] \leq \tau_R(|P|)$, where τ_R is the expected max-step complexity of `RatRace` against A_R. Since by Theorem 7 the expected max-step complexity of `RatRace` is $O(\log k)$ against any strong adaptive adversary, $\tau_R(|P|) = O(\log |P|)$. Then from (10) it follows $E[T_C] = O(\log |P|)$. This completes the proof of Theorem 11. It remains to prove Lemma 13.

5.2 Proof of Lemma 13

We first prove (10). Consider a process $p \in P$ that invokes `Comb'` in $\mathcal{E}_C$. Process p alternates devoting steps to `Imp` and `RatRace` (starting with a step of `Imp`), until either its `Comb'` call ends, because p won `RatRace` or lost `RatRace` or won `Imp` (see (C1) and (C2)), or until it stops executing steps of `Imp` (see (C3)). Hence, in either case at least $\lfloor T_C^p/2 \rfloor$ of p's steps in $\mathcal{E}_C$ are devoted to `RatRace`, and thus $T_R \geq (T_C - 1)/2$. This implies (10).

Next we prove (9). We will use the next statement which follows easily from Chernoff Bounds.

Lemma 14. *For every constant $0 < q < 1$, there exists a constant $c > 0$ such that the following is true for all $\Delta \geq 0$, and all integers $m \geq 1$. If $X_1, \ldots, X_m$ are random variables satisfying $\Pr(X_i > \ell \mid X_1, \ldots, X_{i-1}) \leq q^\ell$ for every integer $\ell \geq 0$, then*

$$\Pr\left(\sum_{1 \leq i \leq m} X_i > c \cdot (m + \Delta)\right) \leq 4^{-\Delta}.$$

Proof. If we choose $c \geq \lceil \log_4(1/q) \rceil$, then the statement is true for $m = 1$. Therefore, in the rest of the proof it suffices to consider $m \geq 2$. Conditionally on $X_1, \ldots, X_{i-1}$, random variable X_i is dominated by a geometric random variable Y_i with parameter q. It follows that $\sum_{1 \leq i \leq m} X_i$ is dominated by $\sum_{1 \leq i \leq m} Y_i$, where the random variables Y_i are mutually independent. Let $Y = \sum_{1 \leq i \leq m} Y_i$, so $\mathbf{E}[Y] = m/q$. We can then apply a Chernoff Bound for independent geometric random variables (e.g., [10, Theorem 1.14]), which states that for any $\delta > 0$,

$$\Pr\left(Y \geq (1 + \delta)\,\mathbf{E}[Y]\right) \leq \exp\left(-\frac{\delta^2(m-1)}{2(1+\delta)}\right).$$

Setting $\delta = cq + cq\Delta/m - 1$, for a c large enough that $\delta > 0$, we obtain

$$\Pr\left(\sum_{1\le i\le m} X_i \ge c\cdot(m+\Delta)\right) \le \Pr\big(Y \ge c\cdot(m+\Delta)\big) = \Pr\big(Y \ge E[Y]\cdot(q/m)\cdot c\cdot(m+\Delta)\big)$$
$$\le \Pr\big(Y \ge \mathbf{E}[Y](1+cq+cq\Delta/m-1)\big) \le \exp\left(-\frac{(cq+cq\Delta/m-1)^2(m-1)}{2(cq+cq\Delta/m)}\right). \quad (11)$$

For $c \ge 4/q$, we have $(cq+cq\Delta/m-1)^2 \ge (cq+cq\Delta/m)^2/2$, and thus

$$\frac{(cq+cq\Delta/m-1)^2(m-1)}{2(cq+cq\Delta/m)} \ge \frac{(cq+cq\Delta/m)^2(m-1)}{4(cq+cq\Delta/m)} = (cq+cq\Delta/m)(m-1)/4$$
$$\ge cq(m-1)/4 + cq\Delta/8 > cq\Delta/8.$$

(For the second to last inequality we used $m \ge 2$.) For large enough c, this is at least $\Delta \ln 4$, and then the claim follows from (11). □

The next lemma bounds the probability a process devotes more steps to `RatRace` than to `Imp`. Let Q be the set of processes $p \in P$ that stop at some `RatRace` splitter when executing `Comb′` in $\mathcal{E}_C$.

Lemma 15. *There is a constant $c > 0$ such that for all $\Delta \ge 0$ and any process $p \in P$,*

$$\Pr\big(T_R^p > c(T_I^p + \Delta) \mid T_I^p,\, p \in Q\big) \le 4^{-\Delta}.$$

Proof. In the `RatRace` portion of `Comb′`, a process first executes only `split()` operations until it either loses `RatRace` (and thus `Comb′`), or stops at a splitter. After stopping at a splitter, p's remaining execution of `RatRace` comprises only `TAS()` operations on 2-process and 3-process TAS objects. In particular, p executes at most one such `TAS()` operation for each splitter it went through until it stopped at one. Recall also that once p has finished the `Imp` portion of `Comb′`, it finishes at most one more `split()` call in `RatRace` (if it has a pending such call). Hence, p executes at most T_I^p `split()` calls in the `RatRace` portion of `Comb′`, and thus also at most T_I^p `TAS()` operations. Thus, defining Z as the number of steps p takes during those `TAS()` operations, we have

$$T_R^p \le Z + T_I^p + O(1). \quad (12)$$

For $i \in \{1, \ldots, T_I^p\}$ let Z_i denote the number of steps process p executes in order to finish its i-th `TAS()` operation on a 2- or 3-process TAS object of the `RatRace` portion of `Comb′`; if p executes fewer than i such `TAS()` operations, then $Z_i = 0$. As discussed in Section 2.3, for $\ell \ge 0$ a process finishes a `TAS()` on a 2-process TAS object in $O(\ell)$ steps with probability at least $1 - 1/2^\ell$. We can implement each 3-process TAS object from two 2-process TAS objects in such a way that for each `TAS()` operation on the 3-process TAS, a process needs only to complete one or two `TAS()` operations on the 2-process TAS objects. This way, we get the same asymptotic bound as for 2-process TAS objects, i.e., for $\ell \ge 0$, with probability at least $1 - 1/2^\ell$ a process finishes a `TAS()` operation on a 3-process TAS object in $O(\ell)$ steps. Therefore, there is a constant $s > 0$ such that $\Pr(Z_i > s\ell \mid T_I^p, Z_1, \ldots, Z_{i-1},\, p \in Q) \le 2^{-\ell}$ for all $\ell \ge 0$. Then by Lemma 14, applied to $X_i = Z_i/s$, there is a constant $c' > 0$, so that for all $\Delta \ge 0$ and all $m \ge 1$,

$$\Pr\big(Z > c'(T_I^p + \Delta) \mid T_I^p,\, p \in Q\big) = \Pr\big(X_1 + \cdots + X_{T_I^p} > (c'/s)\cdot(T_I^p + \Delta) \mid T_I^p,\, p \in Q\big) \le 4^{-\Delta}.$$

Applying (12) yields the claim for a sufficiently large constant $c > 0$. □

By Lemma 15 (used for the inequality labeled $(*)$ below), there is a constant $c > 0$ such that for any $\Delta \geq 0$,

$$\Pr(T_R^p > 2c\Delta \mid T_I^p \leq \Delta, p \in Q) \leq \Pr\big(T_R^p > c(T_I^p + \Delta) \mid T_I^p \leq \Delta, p \in Q\big) \overset{(*)}{\leq} 4^{-\Delta}. \tag{13}$$

Recall that in `RatRace` a process can stop either at a randomized splitter on the primary tree, or at a deterministic splitter on an elimination path. Moreover, at most one process can stop at each splitter, so at most 2^i processes can stop at the i-th splitter they go through. Since a process p executes fewer than T_I^p `split()` calls in `RatRace` before stopping at a splitter or terminating `Comb'`, the number of processes $p \in Q$ satisfying $T_I^p \leq \Delta$ is at most 2^Δ. Hence,

$$\sum_{p \in P} \Pr(T_I^p \leq \Delta \wedge p \in Q) = \mathbf{E}[|\{p \in Q : T_I^p \leq \Delta\}|] \leq 2^\Delta. \tag{14}$$

For any process $p \in P$ that does not stop at any `RatRace` splitter, i.e., $p \in P \setminus Q$, we have $T_R^p \leq T_I^p + O(1)$, because once p has finished the `Imp` portion of `Comb'`, it finishes at most one `split()` call in `RatRace` before finishing `Comb'`. It follows that for any $p \in P$, $T_R^p > 2cT_i^p$ implies $p \in Q$, if the constant c is sufficiently large. Using this observation we obtain

$$\begin{aligned}
\sum_{p \in P} \Pr(T_R^p > 2c\Delta \wedge T_I^p \leq \Delta) &= \sum_{p \in P} \Pr(T_R^p > 2c\Delta \wedge T_I^p \leq \Delta \wedge p \in Q) \\
&= \sum_{p \in P} \Pr(T_R^p > 2c\Delta \mid T_I^p \leq \Delta, p \in Q) \cdot \Pr(T_I^p \leq \Delta \wedge p \in Q) \\
&\overset{(13)}{\leq} \sum_{p \in P} 4^{-\Delta} \cdot \Pr(T_I^p \leq \Delta \wedge p \in Q) \overset{(14)}{\leq} 4^{-\Delta} \cdot 2^\Delta = 2^{-\Delta}.
\end{aligned} \tag{15}$$

It follows that

$$\begin{aligned}
\Pr(T_R > 2c\Delta) &= \Pr(T_R > 2c\Delta \wedge T_I > \Delta) + \Pr(T_R > 2c\Delta \wedge T_I \leq \Delta) \\
&\leq \Pr(T_I > \Delta) + \sum_{p \in P} \Pr(T_R^p > 2c\Delta \wedge T_I^p \leq \Delta) \\
&\overset{15}{\leq} \Pr(T_I > \Delta) + 2^{-\Delta}.
\end{aligned} \tag{16}$$

Then

$$\begin{aligned}
\mathbf{E}[T_R] = \sum_{t \geq 0} \Pr(T_R > t) &\overset{(16)}{\leq} \sum_{t \geq 0} \Big(\Pr\big(T_I > t/(2c)\big) + 2^{-t/(2c)} \Big) \\
&\leq \sum_{t \geq 0} \Pr\big(T_I > \lfloor t/(2c) \rfloor\big) + \sum_{t \geq 0} 2^{-t/(2c)} \leq \sum_{j \geq 0} 2c \cdot \Pr(T_I > j) + O(1) = O(\mathbf{E}(T_I)).
\end{aligned}$$

Finally, combining that with the fact that $T_C = T_I + T_R$, implies (9). This completes the proof of Lemma 13. □

6 A 2-Process Time Lower Bound for Oblivious Adversaries

We show a lower bound on the max-step complexity of any 2-process TAS implementation, against the worst possible oblivious adversary.

Theorem 16. *For any randomized 2-process TAS implementation and any integer $t \geq 0$, there is an oblivious adversary A such that with probability at least $1/4^t$ the max-step complexity of the implemented* `TAS()` *operation against A is at least t.*

Proof. The proof employs Yao's minimax principle [24].

Let M be a randomized implementation of a 2-processes TAS object. For any execution $\mathcal{E}$ of this implementation, let $c_t(\mathcal{E}) = 1$ if some process executes at least t shared memory steps in $\mathcal{E}$; let $c_t(\mathcal{E}) = 0$ otherwise.

Let Σ_t be the set of all possible schedules $\sigma = (\sigma_1, \sigma_2, \ldots)$, where $\sigma_i \in \{0, 1\}$, and σ has the following properties: (i) $|\sigma| = 2k$, for some $k \in \{t, \ldots, 2t-1\}$; (ii) $\sigma_{2i-1} = \sigma_{2i}$, for all $i \in \{1, \ldots, k\}$; and (iii) some process $p \in \{0, 1\}$ appears exactly $2t$ times at σ (so, the other process, $1-p$, appears $2(k-t) < 2t$ times). We have

$$|\Sigma_t| \leq \sum_{k=t}^{2t-1} 2^k \leq 2^{2t} = 4^t. \tag{17}$$

Consider the coin flip sequences $\omega_p = (\omega_{p,1}, \ldots, \omega_{p,t}) \in \Omega^t$, for $p \in \{0, 1\}$. For any schedule $\sigma \in \Sigma_t$, let $\mathcal{E}_M(\sigma, \omega_0, \omega_1)$ denote the execution of algorithm M where processes are scheduled according to σ, and the i-th coin flip of process p returns the value $\omega_{p,i}$. Recall our model assumption that (w.l.o.g.) each process alternates between coin flip steps and shared memory steps. Since each process appears at most $2t$ times in $\sigma \in \Sigma_t$, each process executes at most t coin flips in the resulting execution. We will now show that

$$\forall\, \omega_0, \omega_1 \in \Omega^t\, \exists\, \sigma \in \Sigma_t \colon c_t\big(\mathcal{E}_M(\sigma, \omega_0, \omega_1)\big) = 1. \tag{18}$$

To prove (18), let λ_p, for $p \in \{0, 1\}$, be an arbitrary but fixed infinite extension of ω_p, e.g., we can choose $\lambda_p = (\omega_{p,1}, \ldots, \omega_{p,t}, 0, 0, \ldots)$, assuming that 0 is an element of Ω. Let M_λ be the TAS algorithm where each process p executes the same program as in M, but ignores its coin flips, and instead acts as if its i-th coin flip is the i-th element of vector λ_p. Then M_λ behaves as a deterministic 2-process TAS algorithm. Since there is no *wait-free* deterministic 2-process TAS algorithm, there exists an execution of M_λ in which at least one process executes at least t shared memory steps without finishing its `TAS()` call. Moreover, there is such an execution $\mathcal{E}'$ which has the additional property that each coin flip step by process $p \in \{0, 1\}$ (whose result is replaced in the algorithm by an element of λ_p) is immediately followed in $\mathcal{E}'$ by the next shared memory step of the same process p. Let $\mathcal{E}$ be the prefix of $\mathcal{E}'$ that ends when the first process has executed its t-th shared memory step, and let σ be the schedule corresponding to $\mathcal{E}$. The prefix $\mathcal{E}$ exists, because we argued above that some process executes at least t shared memory steps without finishing its `TAS()` call. It follows that $c_t(\mathcal{E}) = 1$ and $\sigma \in \Sigma_t$, and also $\mathcal{E} = \mathcal{E}_M(\sigma, \omega_0, \omega_1)$. This proves (18).

Now let (ω_0^*, ω_1^*) be chosen according to any product distribution over $\Omega^t \times \Omega^t$, and σ^* according to any probability distribution over Σ_t. By Yao's minimax principle [24],

$$\max_{\sigma \in \Sigma_t} \mathbf{E}\left[c_t\big(\mathcal{E}_M(\sigma, \omega_0^*, \omega_1^*)\big)\right] \geq \min_{\omega_0, \omega_1 \in \Omega^t} \mathbf{E}\left[c_t\big(\mathcal{E}_M(\sigma^*, \omega_0, \omega_1)\big)\right]. \tag{19}$$

Let ε denote the left side in this inequality, and recall that $c_t(\mathcal{E}_M(\sigma, \omega_0^*, \omega_1^*))$ is a 0–1 random variable indicating whether some process executes at least t steps in $\mathcal{E}_M(\sigma, \omega_0^*, \omega_1^*)$. Hence, ε is a lower bound for the probability that some process needs at least t steps to finish its `TAS()` call in a random execution of M, for the worst possible schedule σ. Thus, it suffices to prove that $\varepsilon \geq 1/4^t$. To do so we choose σ^* uniformly in Σ_t to obtain

$$\varepsilon \overset{(19)}{\geq} \min_{\omega_0, \omega_1 \in \Omega^t} \mathbf{E}\left[c_t\big(\mathcal{E}_M(\sigma^*, \omega_0, \omega_1)\big)\right] = \min_{\omega_0, \omega_1 \in \Omega^t} \Pr\Big(c_t\big(\mathcal{E}_M(\sigma^*, \omega_0, \omega_1)\big) = 1\Big) \overset{(18)}{\geq} \frac{1}{|\Sigma_t|} \overset{(17)}{\geq} \frac{1}{4^t}.$$

This completes the proof of Theorem 16. ∎

Conclusion

In this paper we devised several efficient randomized TAS algorithms. Most importantly, we presented an algorithm with an expected max-step complexity of $O(\log^* k)$ against the oblivious and some slightly stronger adversary models, where k is a measure of contention.

The progress in improving randomized TAS algorithms is mirrored by recent progress on randomized consensus algorithms. Aspnes [6] has devised a randomized consensus algorithm that has $O(\log \log n)$ expected max-step complexity in the oblivious adversary model. This algorithm is based on the sifting technique from [2]. It would be interesting to investigate whether techniques similar to those presented here can be used to achieve even faster consensus algorithms. In particular, we believe that our group election implementation for r/w-oblivious adversaries proposed in Section 3.4 could be used in the framework of [6] to obtain an *adaptive* binary consensus algorithm with expected max-step complexity $O(\log \log k)$.

Several other important problems remain open. For the oblivious adversary, no TAS implementations with constant expected max-step complexity are known, and no super-constant lower bounds are known even in the strong adaptive adversary model.

Acknowledgements

We thank Dan Alistarh for pointing out Styer and Peterson's $\Omega(\log n)$ space lower bound for deadlock-free leader election [21]. We also thank the anonymous reviewers for their helpful feedback.

References

[1] Yehuda Afek, Eli Gafni, John Tromp, and Paul M. B. Vitányi. Wait-free test-and-set. In *Proceedings of the 6th International Workshop on Distributed Algorithms (WDAG)*, pages 85–94, 1992.

[2] 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)*, pages 97–109, 2011.

[3] Dan Alistarh, James Aspnes, Keren Censor-Hillel, Seth Gilbert, and Morteza Zadimoghaddam. Optimal-time adaptive strong renaming, with applications to counting. In *Proceedings of the 30th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 239–248, 2011.

[4] Dan Alistarh, James Aspnes, Seth Gilbert, and Rachid Guerraoui. The complexity of renaming. In *Proceedings of the 52nd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 718–727, 2011.

[5] 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)*, pages 94–108, 2010.

[6] James Aspnes. Faster randomized consensus with an oblivious adversary. In *Proceedings of the 31st ACM Symposium on Principles of Distributed Computing (PODC)*, pages 1–8, 2012.

[7] Hagit Attiya and Keren Censor-Hillel. Lower bounds for randomized consensus under a weak adversary. *SIAM Journal on Computing*, 39(8):3885–3904, 2010.

[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.

[9] Harry Buhrman, Alessandro Panconesi, Riccardo Silvestri, and Paul M. B. Vitányi. On the importance of having an identity or, is consensus really universal? *Distributed Computing*, 18(3):167–176, 2006.

[10] Benjamin Doerr. Analyzing randomized search heuristics: Tools from probability theory. In *Theory of Randomized Search Heuristics: Foundations and Recent Developments*, pages 1–20. World Scientific, 2011.

[11] Wayne Eberly, Lisa Higham, and Jolanta Warpechowska-Gruca. Long-lived, fast, waitfree renaming with optimal name space and high throughput. In *Proceedings of the 12th International Symposium on Distributed Computing (DISC)*, pages 149–160, 1998.

[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)*, pages 46–60, 2013.

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

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

[15] Wojciech Golab, Danny Hendler, and Philipp Woelfel. An $O(1)$ RMRs leader election algorithm. *SIAM Journal on Computing*, 39:2726–2760, 2010.

[16] Jens Jägersküpper. Algorithmic analysis of a basic evolutionary algorithm for continuous optimization. *Theoretical Computer Science*, 279(3):329–347, 2007.

[17] Clyde P. Kruskal, Larry Rudolph, and Marc Snir. Efficient synchronization on multiprocessors with shared memory. *ACM Transactions on Programming Languages and Systems*, 10(4):579–601, 1988.

[18] Colin 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.

[19] Mark Moir and James H. Anderson. Fast, long-lived renaming. In *Proceedings of the 8th International Workshop on Distributed Algorithms (WDAG)*, pages 141–155, 1994.

[20] Alessandro Panconesi, Marina Papatriantafilou, Philippas Tsigas, and Paul M. B. Vitányi. Randomized naming using wait-free shared variables. *Distributed Computing*, 11(3):113–124, 1998.

[21] Eugene Styer and Gary L. Peterson. Tight bounds for shared memory symmetric mutual exclusion problems. In *Proceedings of the 8th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 177–192, 1989.

[22] John Tromp and Paul M. B. Vitányi. Randomized wait-free test-and-set. Manuscript, 1990.

[23] John Tromp and Paul M. B. Vitányi. Randomized two-process wait-free test-and-set. *Distributed Computing*, 15(3):127–135, 2002.

[24] Andrew Chi-Chih Yao. Probabilistic computations: Towards a unified measure of complexity. In *Proceedings of the 17th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 222–227, 1977.