Time-Space Tradeoffs for Implementations of Snapshots

Panagiota Fatourou
University of Ioannina
Greece

Faith Ellen Fich
University of Toronto
Canada

Eric Ruppert
York University
Canada

ABSTRACT

A snapshot object is an abstraction of the fundamental problem of obtaining a consistent view of the contents of the shared memory in a distributed system while other processes may concurrently update those contents. A snapshot object stores an array of m components and can be accessed by two operations: an UPDATE that changes the value of an individual component and a powerful SCAN that returns the contents of the entire array.

This paper proves time-space tradeoffs for fault-tolerant implementations of a snapshot object from registers that support only **read** and **write** operations. For anonymous implementations (where all processes are programmed identically), we prove that a SCAN requires $\Omega(n/r)$ time, where n is the number of processes in the system and r is the number of registers used by the implementation. For the general non-anonymous case, we prove that, for any fixed r, the time required to do a SCAN grows without bound as n increases. These tradeoffs hold even in the case where the snapshot object has just two components.

This is the first time a lower bound on the tradeoff between time complexity and the number of registers has been proved for any problem in asynchronous shared-memory systems. We introduce a new tool for proving distributed lower bounds: the notion of a *shrinkable* execution, from which an adversary can remove portions as necessary.

Categories and Subject Descriptors

F.2.3 [**Analysis of Algorithms and Problem Complexity**]: Tradeoffs between Complexity Measures; E.1 [**Data Structures**]: Distributed data structures

General Terms

Theory

Keywords

distributed computing, shared memory, snapshot, wait-free, lower bound, time-space tradeoff, anonymous processes

1. INTRODUCTION

In asynchronous systems where processes can communicate only by writing to and reading from shared registers, there are many basic problems that are unsolvable or have only very complicated algorithms. Because processes can run at arbitrarily varying speeds and can crash, it is difficult for different processes to get consistent global views of the shared memory while other processes may be concurrently changing the values of some registers.

The snapshot object [1, 4, 8] is an important abstraction that has been widely used to address this difficulty. A snapshot provides an array of m components and a powerful SCAN operation that returns a consistent view of all components. Processes can also perform UPDATE operations on individual components. In general, every process can update every component. The special case of a single-writer snapshot, where only one process can update each component, has also been studied. Snapshot objects have been used to construct fault-tolerant algorithms for numerous distributed problems, including timestamps [20], approximate agreement [10] and randomized consensus [6, 7]. Unlike most other synchronization primitives, a snapshot object can be implemented from registers. However, a SCAN cannot be performed by simply reading the value in each component, because the value of components that are read first may be out of date by the time later components are read.

Despite its wide use and numerous implementations, the time complexity of implementing snapshot objects is not well understood. Some snapshot implementations use small space, but are slow, and others that are faster use more space. In this paper, we give the first lower bound on the tradeoff between time and space for snapshot implementations in asynchronous shared-memory systems.

The classic snapshot implementation [1, 16] uses an array of m registers, to store an m-component snapshot object shared by n processes, but both SCAN and UPDATE take $\Theta(mn)$ steps, in the worst case. There is a snapshot object implemented using single-writer registers (*i.e.* only one process can write to each register) [11, 5], where SCAN takes $\Theta(n)$ steps and UPDATE takes $\Theta(n \log n)$ steps. For any number of components, m, this implementation uses n registers. In the fastest known implementation of a snapshot object [24, 5], both SCAN and UPDATE take $\Theta(n)$ steps, but an unbounded number of registers are used.

Although the optimal space complexity of implementing a snapshot object is known, there are only very limited lower bounds on time complexity. Jayanti, Tan, and Toueg [26] proved that an m-component snapshot object shared by n processes requires at least $\min\{m, n\} - 1$ registers and `SCANS` take at least $\min\{m, n\} - 1$ steps in the worst case. We improved the lower bound on the number of registers to $\min\{m, n\}$ [15], matching the implementations mentioned above. We also proved that, when $m < n$, any space-optimal implementation takes $\Omega(mn)$ steps to perform a `SCAN` [16]. If single-writer registers are available in addition to the m registers and $m < 3n/4$, we proved an $\Omega(\sqrt{mn})$ lower bound on the number of steps a process must do to complete an operation. Attiya, Fatourou and Fich [9] extended a result by Israeli and Shirazi [25] to obtain an $\Omega(m)$ lower bound on the number of steps to perform an `UPDATE` for space-optimal implementations. When even one additional register is available, only the $\min\{m, n\} - 1$ lower bound on time complexity applies. Thus, when m is constant, there are no nonconstant time lower bounds for implementations using $m + 1$ or more registers.

In this paper, we prove the first general lower bound on the tradeoff between the time to perform a `SCAN` and the number of registers used for implementations of snapshot objects. The tradeoffs apply to a snapshot object with $m = 2$ components and, hence, for all values of m). There are only a small number of lower bounds on tradeoffs between time and space for shared-memory asynchronous systems [2, 3, 30], but, unlike this paper, they are concerned with the number of bits per register rather than the number of registers. There is also a tradeoff between write-contention and space [22].

It is known that snapshots can be implemented from registers even when processes are anonymous (*i.e.* they execute identical code) [21]. For this special case, we prove that r, the number of registers used by an implementation, and k, the worst-case number of steps needed to perform a `SCAN` in that implementation, must satisfy $rk \in \Omega(n)$. We then extend the proof to get our main contribution: a general tradeoff between k and r that applies even for non-anonymous systems. This tradeoff implies that, in any implementation of a 2-component snapshot object shared by n processes that uses a constant number of registers, the number of steps required to do a `SCAN` grows without bound as n increases, even if each register can be arbitrarily large. It follows that the same result holds for implementations of an m-component snapshot, for any $m \geq 2$.

The proofs in this paper use covering arguments [14, 18], in which processes are poised to write to specific registers, so that they can be later used to obliterate information stored in those registers. (A process is *poised* to perform an operation if it will do that operation when it next takes a step.) In our proofs, an adversary carefully constructs an execution where some process t performing a `SCAN` must take many steps, because crucial information is hidden from t by other processes that write useless information to registers just before t reads from them. Although the proof of the lower bound for space-optimal implementations [16] also uses a covering argument, it begins by proving a number of structural properties of these implementations that simplify the rest of the proof considerably. Specifically, the properties allow the adversary to poise a process at any particular register just by specifying the component it updates. Moreover, all these processes can be poised at the beginning of the execution without performing any `writes`, so that they do not know the crucial information and other processes do not know about their existence.

Unfortunately, these structural properties disappear when even $m + 1$ registers are available. This makes the proof of our general time-space tradeoff considerably more difficult. In particular, the adversary must hide the poised processes so that other processes do not know about them.

In anonymous systems, it is easy to poise a process at any register that has been written to. The idea is to have the process act as a "clone" of a process p that wrote there, performing the same steps as p until it is just about to write to the desired register. However, if a process writes useful information into a register that has not previously been written to, the adversary has no way to hide the information. Overcoming this difficulty is an essential part of the lower bound proof in the anonymous case.

In the general, non-anonymous case, poising a particular process at a particular register may be impossible, since that process may just never write to that register. However, there is *some* register at which the adversary can poise many processes. Then, the adversary repeatedly increases the number of registers at which many processes are poised. To do so, it may use some of the previously poised processes. As the number of such registers increases, the number of processes poised at each decreases, but not too quickly.

When the adversary poises processes at registers, it is important that other processes do not know of their existence. In the anonymous case, this is achieved because other processes cannot detect the presence of a clone. In the general, non-anonymous case, it is quite difficult to hide the existence of a poised process from other processes. To deal with this problem, we introduce the concept of a shrinkable execution, which allows any subset of `UPDATE` operations that are in progress to be entirely deleted together with some final steps of some `SCAN` operations so that no other processes are aware of the difference.

The main reason we consider the anonymous case is to provide intuition for the general, non-anonymous case. In fact, it was essential for our development of the general lower bound. In the words of Obi-Wan Kenobi, "without the clones, it would not have been a victory" [27]. However, anonymous systems are also interesting in their own right. One motivation for such systems is privacy protection: users may wish to access information stored in shared memory without divulging their own identities to the server that stores the information [13, 29].

Fich, Herlihy, and Shavit [17] proved an $\Omega(\sqrt{n})$ lower bound on the number of registers needed for n-process consensus. Their paper also begins by considering the special case of anonymous processes, using clones to poise processes. For the consensus problem, there are only two possible outputs, 0 and 1, and an execution is incorrect if both 0 and 1 are output, so a simple valency argument [19] suffices to prove that additional registers must be written to. However, lots of different outputs can occur in a single legal execution of a snapshot implementation. Therefore, our lower bound for anonymous systems requires different techniques that make our lower bound more difficult. To poise processes in the general, non-anonymous case, Fich, Herlihy, and Shavit introduced the concept of an interruptible execution, which can be broken into pieces between which executions involving other processes can be inserted. Shrinkable executions, which we use instead, are simpler and more natural.

The lower bound proofs we give in this paper can easily be modified so that the adversary does not need to have a process perform multiple operations. Thus, our tradeoffs also apply (with slightly different constant factors) to *one-shot* implementations of snapshot objects, which can be accessed by each process at most once.

After briefly presenting the computational model in Section 2, we prove the tradeoff for anonymous systems in Section 3. The structure of the proof is similar to the proof for the general case and it provides important intuition. In Section 4, we give the definition of a shrinkable execution, followed by the tradeoff for general, non-anonymous systems.

2. MODEL

This paper considers an asynchronous shared-memory model in which a set of n deterministic processes communicate by accessing r shared registers. A register supports two atomic operations, `read` and `write`. A `read`() operation returns the value of the register and leaves the value unchanged, while a `write`(v) operation writes the value v in the register and returns ack. Every process can perform `read` and `write` operations on any register. The order in which operations performed by processes are interleaved is assumed to be controlled by an adversarial scheduler. Algorithms must work correctly regardless of the schedule the adversary chooses. An algorithm is called *wait-free* if nonfaulty processes terminate in a finite number of steps, even when any number of processes fail. Formal definitions of this model can be found in [12, 28].

An *execution* is a sequence of `read` and `write` operations by processes starting from some configuration. The process performing each operation, the register on which it is performed, and the response of each `read` operation are also specified. A *legal* execution is an execution in which the subsequence of the operations performed by each process follows its algorithm and the response to each `read` from a register is the last value written to it. A *solo* execution is performed by a single process. An execution α is *indistinguishable* from an execution β by a process p, if p has taken the same sequence of steps during α and β and each of the shared registers has the same value at the end of α and β.

A *configuration* describes a system at some point in time. It is comprised of a state for each process and a value for each register. An *initial configuration* is a configuration in which each process is in an initial state and each register has its initial value. A process is *poised* at a register R in some configuration if it will write to R when next allocated a step by the scheduler. A *block* `write` is a sequence of consecutive `write` operations to different registers by different processes. A block `write` can be performed starting in a configuration only if all of those processes are poised at distinct registers.

A *snapshot* object A [1, 4, 8] is an object with m components $A_0, \ldots, A_{m-1}$. It supports two atomic operations, `SCAN` and `UPDATE`. The operation `UPDATE`(i, v) sets the value of component A_i to v. The `SCAN` operation returns a vector consisting of the values of the m components of A. In this paper, we consider the problem of implementing a snapshot object. An *implementation* provides, for each process, an algorithm for performing `UPDATE` and an algorithm for performing `SCAN`. Before an execution of the latter algorithm terminates, it must produce a response, namely a vector of m values. A `SCAN` or `UPDATE` operation is *pending* at some configuration of an execution if the process performing it has already performed the first step of this invocation of its algorithm, but has not yet finished performing the algorithm. A process is *idle* if it does not have a pending operation. An *idle* configuration is one where all processes are idle. The *time complexity* of an implementation is the maximum number of steps taken by a process to perform an `UPDATE` or `SCAN` operation.

We consider only *linearizable* implementations [23]. This means that in any execution, every simulated operation appears to take effect at some instant during the period of time it is pending. Specifically, there exists a linear order of the simulated `UPDATE` and `SCAN` operations so that the correct responses for this ordering of the operations are the same as those given in the actual execution. Furthermore, if the algorithm for one invocation of an operation finishes executing before the algorithm for another invocation of an operation has begun, the latter operation must come later in the linear order. Thus, the vector of values returned by a `SCAN` are the values of the components in the configuration at the point it appears to take effect.

We use $\bar{i}$ to denote the complement of the bit i, so that $\bar{i} = 1 - i$.

3. THE ANONYMOUS TIME-SPACE TRADEOFF

Consider any anonymous implementation of a 2-component snapshot object from r registers shared by n processes such that `UPDATES` eventually terminate if given sufficiently many *consecutive* steps and `SCANS` take at most k steps. Any anonymous wait-free implementation in which k is the worst-case number of steps needed to perform a `SCAN` is such an algorithm. In this section, we prove that $kr \in \Omega(n)$. We begin with an informal description of the proof.

Since we are using a covering argument, our first task is to ensure that processes can be poised at many registers. The system is anonymous, so it suffices to construct an execution in which many registers are written to; then we can insert clones of processes into this execution, leaving them poised to write to those registers. We construct such executions inductively. Suppose there is an execution in which a set $\mathcal{R}$ of registers have been written to. Then, for any sequence of k (not necessarily distinct) registers in $\mathcal{R}$, there is another execution using k additional clones, after which the k clones are poised to write to this sequence of registers. An adversary can construct an extension of this execution using two idle processes q and t. Process q performs a series of `SCANS` and `UPDATES` with a hidden value h that has not previously appeared in the snapshot object. Process t performs a troublesome `SCAN`. The operations performed by q are designed to ensure that the snapshot object contains the value h in at least one of its components at all times during t's `SCAN`. If processes never write outside of $\mathcal{R}$, then t can never discover that a component contains h, because each time t reads a register in $\mathcal{R}$, the adversary ensures that the register's contents have been overwritten by a clone. Since t does at most k `reads`, the clones can be poised to do this. Hence a process must write outside of $\mathcal{R}$ in the constructed extension. This argument allows us to inductively construct an execution where every register is written to, provided there are enough processes to use for the construction. (We call this the level-1 induction. There is also a nested induction, called the level-2 induction, for the construction of the troublesome `SCAN`.) If it is possible for the adversary to poise a clone at every register, it is easy to argue that the implementation is incorrect, because an `UPDATE` can be hidden by a block `write` to all registers by those r clones.

One technical requirement complicates the construction: the adversary may need to use a clone that is poised to `write` to a particular register, but is not performing an `UPDATE` to a particular component. If all clones poised at this particular register are performing `UPDATES` to this particular component, then the construction will not work. Thus, the construction ensures that each register is written to either by a process performing a `SCAN` or by both a process performing an `UPDATE` to component 0 and a process performing an `UPDATE` to component 1. Instead of keeping track of a single set, $\mathcal{R}$, we have two sets, $\mathcal{R}_0$ and $\mathcal{R}_1$, where $\mathcal{R}_i$ keeps track of registers written by `UPDATES` to component A_i or `SCANS`. As a result, the level-1 inductive argument is iterated $2r-1$ times. Each iteration uses up to k clones. If $n \geq k(2r-1)+2+r$, then the adversary can construct an incorrect execution. We now formalize these ideas in the following lemma.

Lemma 1. *For $0 \leq d \leq 2r-1$, there exist sets of registers $\mathcal{R}_0$ and $\mathcal{R}_1$, with $|\mathcal{R}_0|+|\mathcal{R}_1| = d$, and an execution β involving at most $kd+2$ processes that ends in an idle configuration such that, for all $i \in \{0,1\}$ and all $R \in \mathcal{R}_i$, some process writes to R in β while doing an* `UPDATE` *to component A_i or a* `SCAN`.

The time-space tradeoff follows easily from the lemma with $d = 2r-1$. Note that the termination property for `UPDATE` operations is weaker than wait-freedom.

Theorem 2. *In any n-process anonymous implementation of a 2-component snapshot object from r registers, such that each process will complete any* `UPDATE` *operation given sufficiently many consecutive steps, the worst-case number of steps needed to perform a* `SCAN` *operation is in $\Omega(n/r)$.*

Proof. To derive a contradiction, assume that k, the worst-case number of steps needed to perform a `SCAN` operation, is at most $(n-2-r)/(2r-1)$. Let $\mathcal{R}_0, \mathcal{R}_1$ and β satisfy the lemma for $d = 2r-1$. Since $|\mathcal{R}_0|+|\mathcal{R}_1| = 2r-1$, there is some $i \in \{0,1\}$ such that $\mathcal{R}_i$ contains all registers. Let β' be formed from β by adding r clones so that, at the end of β', each register has a clone poised at it and that clone has a pending `UPDATE` to component A_i or a pending `SCAN`. Let h be a value that does not appear as the parameter of any `UPDATE` operation during β. Let $\gamma = \beta' \cdot Z \cdot U \cdot W \cdot S$, where Z is a sequence of steps that completes all pending `UPDATES` to $A_{\bar{i}}$, U is a solo `UPDATE` to $A_{\bar{i}}$ with value h by some idle process, W is a block `write` to all registers by the r clones we poised, and S is a solo `SCAN` by some other idle process. Then S must return h as the value for component $A_{\bar{i}}$, since all other `UPDATES` to $A_{\bar{i}}$ are completed before U begins. But $\gamma' = \beta \cdot Z \cdot W \cdot S$ is also a legal execution, since W overwrites all evidence of U. In γ', the `SCAN` S also returns the value h. However, h does not appear as the parameter of any `UPDATE` operation during γ'. Thus, the implementation is incorrect. Hence $k > (n-2-r)/(2r-1) \geq \frac{n-2}{2r-1} - 1 \in \Omega(n/r)$. □

Proof of Lemma 1 (sketch). We proceed by induction on d. We call this the level-1 induction.

Level-1 Base Case ($d = 0$). Take $\mathcal{R}_0 = \mathcal{R}_1 = \emptyset$ and let β be the empty execution.

Level-1 Induction Step. Let $0 \leq d < 2r-1$ and suppose there exist $\mathcal{R}_0$, $\mathcal{R}_1$ and β that satisfy the claim for d. To derive a contradiction, assume the claim is false for $d+1$. Let $\mathcal{R} = \mathcal{R}_0 \cup \mathcal{R}_1$. Let h be a value that does not appear in the snapshot object during β. We pick two processes: q to do `UPDATES` with the value h, and t to do the troublesome `SCAN`. We call h the hidden value and `UPDATES` by q hidden `UPDATES`. (If possible, we choose processes that participated in β.) We inductively construct executions in which t performs a `SCAN` that takes increasingly many steps. Eventually, we arrive at a contradiction, since a `SCAN` can take at most k steps. Specifically, we have:

Level-2 Induction Claim. There exist indices $x_0, \ldots, x_{k+1} \in \{0,1\}$, registers $R_1, \ldots, R_{k+1}$, processes $p_1, \ldots, p_k$, and executions $\alpha_0, \ldots, \alpha_k$, where, for each $0 \leq l \leq k$,

- $\alpha_l = \beta_l \cdot \gamma_l$,
- β_l is the same as β, except for the addition of the l clones $p_1, \ldots, p_l$, which are poised to perform `writes` $w_1, \ldots, w_l$ at the end of β_l,
- $\gamma_0 = U_0 \cdot L_0$; if $l \geq 1$, $\gamma_l = \gamma_{l-1} \cdot (w_l \cdot L_l \cdot Z_l \cdot U_l \cdot S_l)$,
- if $l \geq 1$, w_l is a `write` by p_l to register $R_l \in \mathcal{R}_{x_l}$ during its execution of a `SCAN` or an `UPDATE` to component A_{x_l},
- U_l is a complete solo execution of an `UPDATE` to component A_{x_l} by q with the hidden value h,
- $L_0 \cdot L_1 \cdots L_l$ is a prefix of the sequence of steps performed by process t to do a single `SCAN`,
- if $l \geq 1$, Z_l is a solo execution by clone p_l that completes its pending operation, and
- if $l \geq 1$, S_l is a complete solo execution of a `SCAN` by process q,

such that

(1) L_0 contains no `reads` of registers in $\mathcal{R}$; if $l \geq 1$, L_l starts with a `read` of register R_l and contains no `reads` of registers in $\mathcal{R} - \{R_l\}$; at the end of α_l, t is poised to read $R_{l+1} \in \mathcal{R}_{x_{l+1}}$,

(2) during α_l, processes performing `UPDATES` to component A_i write only to registers in $\mathcal{R}_i$ (for each $i \in \{0,1\}$), and processes other than t performing `SCANS` write only to registers in $\mathcal{R}$,

(3) if $l \geq 1$, S_l returns the hidden value h for both components, and

(4) at most $kd+2+l$ processes take steps in α_l.

The executions $\gamma_0, \ldots, \gamma_l$ are illustrated in Figure 1.

Level-2 Base Case ($l = 0$). For $i \in \{0,1\}$, let $U(i,h)$ be the sequence of steps that q takes when doing a solo `UPDATE` to component A_i with the hidden value h, starting from the final configuration of β. The `UPDATE` $U(i,h)$ never writes to any register outside $\mathcal{R}_i$; otherwise we could add such a register to $\mathcal{R}_i$ and show the lemma holds for $d+1$. This contradicts our assumption that the lemma is false for $d+1$.

Let L be the sequence of steps for t to perform a solo `SCAN` starting from the final configuration of β. If t does not read a register in $\mathcal{R}_i$ during L, then $\beta \cdot U(i,h) \cdot L$ is legal and linearizability guarantees that t returns value h for component A_i. Since $\beta \cdot U(i,h) \cdot L$ is indistinguishable

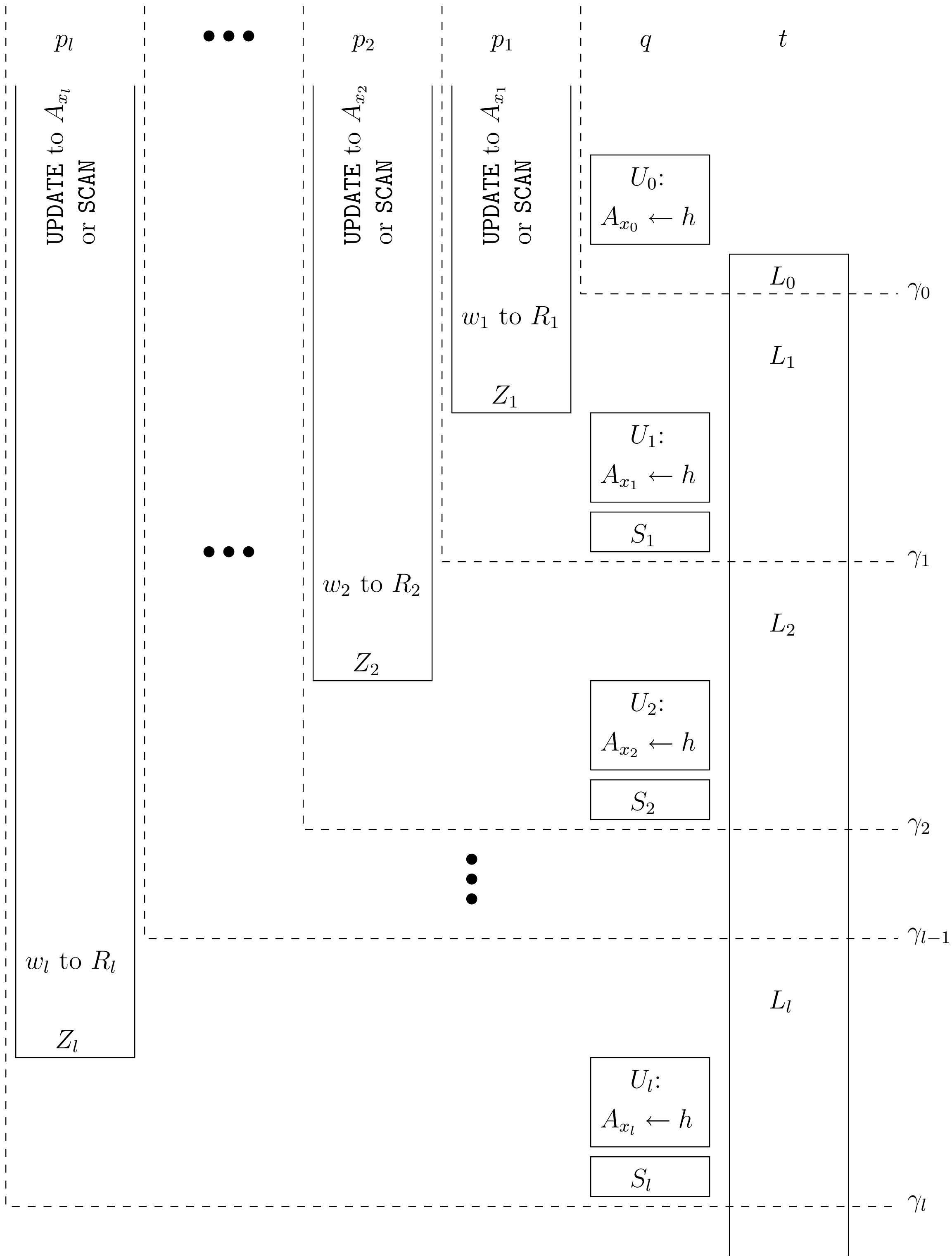

Figure 1: Illustration of $\gamma_0, \ldots, \gamma_l$. The vertical axis represents time.

from $\beta \cdot L$ by t, t must also return h for A_i in $\beta \cdot L$. But this is incorrect, since no `UPDATE` with parameter h appears in $\beta \cdot L$.

Let R_1 be the first register in $\mathcal{R}$ that t reads during L and let L_0 be the prefix of L that precedes that first `read`. Choose x_1 so that $R_1 \in \mathcal{R}_{x_1}$. Let $x_0 = \overline{x_1}$. Let $U_0 = U(x_0, h)$ be the solo `UPDATE` to component A_{x_0} by q with the hidden value h, starting from the final configuration of β.

Let $\gamma_0 = U_0 \cdot L_0$, $\beta_0 = \beta$, and $\alpha_0 = \beta_0 \cdot \gamma_0$. It is now easy to verify the level-2 claim.

LEVEL-2 INDUCTION STEP. Let $0 < l \le k$. Suppose there are indices $x_0, \dots, x_l$, registers $R_1, \dots, R_l$ and executions $\alpha_0, \dots, \alpha_{l-1}$ that satisfy the claim. We show how to choose x_{l+1}, R_{l+1} and α_l.

By the level-2 induction hypothesis, $R_l \in \mathcal{R}_{x_l}$. So, some process doing an `UPDATE` to A_{x_l} or a `SCAN` has written to R_l during β. We form β_l from β_{l-1} by adding a clone p_l of this process, leaving it poised to write to R_l. Let $\phi = \beta_l \cdot \gamma_{l-1} \cdot w_l \cdot L$ where w_l is the `write` p_l is poised to perform on R_l and L is the sequence of steps that t takes to complete its `SCAN`.

CLAIM 1.1. *The `SCAN` by t returns the hidden value h for some component.*

PROOF. Consider any linearization of the `SCANS` and `UPDATES` that occur in ϕ. We show that the snapshot object contains at least one component with value h at all points after U_0 is linearized. All operations completed in β_{l-1} are linearized before U_0. The operations by q in γ_{l-1} are linearized in the order $U_0, U_1, S_1, \dots, U_{l-1}, S_{l-1}$. The only other `UPDATES` are those performed by some of the clones $p_1, \dots, p_l$. Note that all such `UPDATES` use values different from h.

First suppose $l = 1$. Then q performs `UPDATE` U_0 on component A_{x_0} and p_1 performs a `SCAN` or an `UPDATE` to the other component, A_{x_1}. Thus, A_{x_0} contains the hidden value at all points after U_0 is linearized.

Now suppose that $l > 1$. We divide the interval during which t does its `SCAN` into subintervals and prove that some snapshot component contains h throughout each subinterval. By part (3) of the induction hypothesis, S_1 returns (h, h). No hidden `UPDATE` to A_{x_0} is linearized between U_0 and S_1. So, A_{x_0} must contain value h from the time when U_0 is linearized to the time when S_1 is linearized. For $2 \le i \le l-1$, S_{i-1} and S_i both return (h, h). The only hidden `UPDATE` linearized between them is the `UPDATE` U_i to A_{x_i}, so $A_{\overline{x_i}}$ must contain value h at all points between the times when S_{i-1} and S_i are linearized. Finally, S_{l-1} returns (h, h) and the operations by $p_1, \dots, p_{l-1}$ are completed before S_{l-1}, so at most one `UPDATE` can be linearized after S_{l-1} (namely, an `UPDATE` by p_l to A_{x_l}). Thus, $A_{\overline{x_l}}$ contains h at all points after S_{l-1} is linearized. Since t's `SCAN` starts after U_0 is finished, it must return h for some component, proving Claim 1.1.

CLAIM 1.2. *Process t reads some register in $\mathcal{R} - \{R_l\}$ during L.*

PROOF. Suppose not. By part (1) of the induction hypothesis, L_0 contains no `reads` of registers in $\mathcal{R}$ and, for $1 \le j \le l-1$, the only register in $\mathcal{R}$ that process t reads during L_j reads is R_j, which is overwritten by w_j just before L_j. Similarly, the register R_l is overwritten by w_l just before L. So t never reads a value written by q. Furthermore, the values written by w_j cannot give t information about q's `UPDATES` because the clone p_j has not seen any values written by q before it performs w_j. Only q's `UPDATES` use the value h. Since t has seen no information about q's `UPDATES`, t cannot return h, contradicting Claim 1.1. This proves Claim 1.2.

Now we define x_{l+1}, R_{l+1} and α_l. Let R_{l+1} be the first register in $\mathcal{R} - \{R_l\}$ that t reads during L. Let L_l be the prefix of L up to (but not including) that `read`. Pick x_{l+1} so that $R_{l+1} \in \mathcal{R}_{x_{l+1}}$.

Let $\gamma_l = \gamma_{l-1} \cdot w_l \cdot L_l \cdot Z_l \cdot U_l \cdot S_l$, where Z_l is a solo execution by clone p_l that completes its pending operation, U_l is a complete solo `UPDATE` by q to component A_{x_l} with the hidden value h, and S_l is a complete solo `SCAN` operation performed by q. Let $\alpha_l = \beta_l \cdot \gamma_l$. It is easy to prove that this is a legal execution and satisfies parts (1) and (4) of the level-2 claim.

We now prove part (2). By part (2) of the induction hypothesis, it follows that, during $\alpha_{l-1} = \beta_{l-1} \cdot \gamma_{l-1}$, processes performing `UPDATES` to component A_i write only to registers in $\mathcal{R}_i$ (for each $i \in \{0, 1\}$), and processes other than t performing `SCANS` write only to registers in $\mathcal{R}$. Because p_l is a clone of a process taking steps during β, this is also true during β_l. Now, suppose that part (2) is violated during $w_l \cdot L_l \cdot Z_l \cdot U_l \cdot S_l$. Then some process doing an `UPDATE` to component A_i or a `SCAN` writes to a register $R \notin \mathcal{R}_i$. Run pending operations to completion to get an execution β' that ends in an idle configuration and uses at most $k(d+1)+2$ processes. Add R to $\mathcal{R}_i$. Then the lemma holds for $d+1$. This contradicts our assumption at the beginning of the level-1 induction step. Hence, part (2) is not violated during $w_l \cdot L_l \cdot Z_l \cdot U_l \cdot S_l$.

Next we prove part (3). Consider any linearization of the `SCANS` and `UPDATES` that occur in α_l. The operations by q that take place after β_l are linearized in the order $U_0, U_1, S_1, \dots, U_l, S_l$. The operations by $p_1, \dots, p_l$ are all completed prior to the beginning of U_l, so they are all linearized before U_l. Thus, in α_l, S_l returns the hidden value h for component A_{x_l}. Let $j \le l$ be the smallest value such that $x_j = x_{j+1} = \dots = x_l$. Note that $j > 0$ since $x_0 \ne x_1$. So $x_{j-1} = \overline{x_l}$. The `UPDATE` U_{j-1} puts the hidden value h into component $A_{x_{j-1}}$. The only operations by $p_1, \dots, p_l$ that can be linearized after U_{j-1} are the last operations by each of the processes $p_j, \dots, p_l$ since all other operations are completed before U_{j-1} begins. None of them are `UPDATES` to component $A_{x_{j-1}}$. So, in α_l, S_l returns the hidden value h for component $A_{x_{j-1}}$ too.

This completes the level-2 induction proof. When we take $l = k$, there is an execution α_k where some `SCAN` takes at least k steps and is poised to do one more. This contradicts the fact that the worst-case time complexity of a `SCAN` is k, completing the proof of the level-1 induction hypothesis and the lemma. □

4. THE GENERAL TIME-SPACE TRADEOFF

Consider any (non-anonymous) implementation of a 2-component snapshot object from r registers shared by n processes such that `UPDATES` eventually terminate if given sufficiently many *consecutive* steps and `SCANS` take at most k steps. Our goal is to construct an execution similar to the anonymous case, where a troublesome `SCAN` takes many steps. In the non-anonymous case, the adversary must have lots of processes poised at registers, just in case they will be needed later. These processes cannot simply be added whenever they are needed, as the adversary could do with anonymous clones. To ensure that lots of processes are poised at each register is considerably more difficult, and involves an additional level of induction in the proof. (This additional level is now called level 2, and the level-2 induction of the anonymous case becomes level 3.)

Furthermore, having many processes with pending UPDATES causes difficulties. To ensure that a hidden value is in the snapshot object at all times, we need to carefully argue about the linearization of those UPDATES used by the adversary to overwrite information, as in the anonymous case. This requires removing from the execution all pending UPDATES of processes that were poised but not used by the adversary. The removal of an UPDATE could, in general, result in an illegal execution, since some other processes may have seen evidence of this UPDATE. The executions are constructed so that only processes within a particular set Ψ can have seen any evidence of the pending UPDATES: the removal of any set of pending UPDATES can be concealed by also removing some suffix of the sequence of steps of each process in Ψ. This motivates our definition of a Ψ-shrinkable execution.

Consider any finite execution β and let C be its final configuration. The steps of β can be partitioned into *chunks*, where each chunk is a contiguous sequence of steps performed by a single process, and consecutive chunks are performed by different processes. Let Υ be a set of processes with pending UPDATES at C and let Σ be a set of processes with pending SCANS at C. Denote by $\beta(\Upsilon, \Sigma)$ a sequence of steps similar to β except for the removal of (1) all steps of the UPDATES by the processes in Υ that are pending at C and (2) the last chunk of β by each process in Σ.

DEFINITION 3. *Consider any finite execution β and let C be its final configuration. Let Ψ be a set of processes with pending* SCANS *at C. We say that β is* Ψ-shrinkable*, if, for each set Υ of processes that have pending* UPDATES *at C, there exists a set $\Sigma \subseteq \Psi$ such that $\beta(\Upsilon, \Sigma)$ is a legal execution that cannot be distinguished from β by any process not in $\Upsilon \cup \Sigma$.*

Next, we provide a high-level description of the inductive procedure to poise processes. Let β be a Ψ-shrinkable execution for some set Ψ of processes. Suppose $\mathcal{R}_0$ and $\mathcal{R}_1$ are sets of registers such that, at the end of β, there are sufficiently many processes poised to write to each register in $\mathcal{R}_i$ that have either pending UPDATES to component A_i or pending SCANS. Let $d = |\mathcal{R}_0| + |\mathcal{R}_1|$. At each register in $\mathcal{R}_i$, the adversary poises at least $T(d) = (2r-d)!(k+4)^{2r-d}$ processes outside Ψ that have either pending UPDATES to component A_i or pending SCANS. Before adding a register to $\mathcal{R}_i$, the adversary ensures that the number of such processes poised there is at least $T(d+1)$.

The adversary attempts to extend β as in Section 3. Eventually, some process performing an UPDATE to component A_i or a SCAN is forced to write to some register R outside of $\mathcal{R}_i$; otherwise an incorrect execution could be constructed. The adversary pauses that process just before it writes to R and adds some additional steps to make the execution shrinkable. Once the adversary has extended the execution in this way enough times, some register can be added to $\mathcal{R}_0$ or $\mathcal{R}_1$. Each extension uses up some of the processes that were already poised at registers in $\mathcal{R}_0$ and $\mathcal{R}_1$ and some idle processes. For this construction to work, at least $I(d) = 4r-2d+2+\sum_{i=d}^{2r-1}(4r-2i)(T(i+1)-1)$ idle processes must be available at the end of β.

For each $i \in \{0,1\}$ and for each register R, let $\Pi^{i,R}(\beta)$ be the set of processes p such that, at the end of β, p is poised to write to register R and has a pending UPDATE on component A_i or a pending SCAN.

LEMMA 4. *Assume $n \geq I(0)$. For $0 \leq d \leq 2r$, there is a set Ψ of processes, sets $\mathcal{R}_0$ and $\mathcal{R}_1$ of processes, with $|\mathcal{R}_0| + |\mathcal{R}_1| = d$, and a Ψ-shrinkable execution β, such that there are at least $I(d)$ idle processes at the end of β and $|\Pi^{i,R}(\beta) - \Psi| \geq T(d)$ for all $i \in \{0,1\}$ and all $R \in \mathcal{R}_i$.*

We first show how this lemma can be used to prove the following time-space tradeoff.

THEOREM 5. *In any n-process implementation of a 2-component snapshot object from r registers in which processes eventually complete* UPDATE *operations given sufficiently many consecutive steps, the worst-case number of steps needed to perform a* SCAN *operation is greater than $\frac{1}{2r}\left(\frac{n-4r-2}{4}\right)^{\frac{1}{2r-1}} - 4$.*

PROOF. Suppose $n \geq I(0)$. Then we can apply Lemma 4 with $d = 2r$ to see that there is an execution β such that $|\Pi^{0,R}(\beta)| \geq |\Pi^{0,R}(\beta) - \Psi| \geq T(2r) = 1$ for every register R and there are $I(2r) = 2$ idle processes at the end of β. An argument very similar to the proof of Theorem 2 shows that this is impossible, because an UPDATE to component A_1 can be hidden from a subsequent SCAN by having one process in each of the sets $\Pi^{0,R}(\beta)$ do a block `write`. So, $n < I(0)$. But $I(0) \leq 4r+2+4(2r)!(k+4)^{2r-1} \leq 4r+2+4(2r(k+4))^{2r-1}$. Solving for k gives the required bound. $\square$

Due to lack of space we only briefly sketch the proof of Lemma 4 here.

PROOF OF LEMMA 4 (SKETCH). The proof is by induction on d. (Because there will be three nested levels of induction in this proof, we call this induction the level-1 induction.)

LEVEL-1 BASE CASE. When $d = 0$, let $\Psi = \mathcal{R}_0 = \mathcal{R}_1 = \emptyset$ and let β be the empty execution.

LEVEL-1 INDUCTION STEP. Assume the lemma holds for d. We use a second-level induction argument on f, the number of times the execution is extended until some register can be added to $\mathcal{R}_0$ or $\mathcal{R}_1$ (with $T(d+1)$ processes poised there). The set $\Phi_f^{i,R}$ is used to keep track of the processes performing SCANS or UPDATES to component A_i that became poised at register $R \notin \mathcal{R}_i$ during the first f extensions. Since there are $(2r-d)$ pairs (i, R) with $R \notin \mathcal{R}_i$ and each extension poises one more process, the pigeonhole principle says that $(2r-d)(T(d+1)-1)+1$ extensions are sufficient.

LEVEL-2 INDUCTION CLAIM. For $0 \leq f \leq (2r-d)(T(d+1)-1)+1$, there exist a set of processes Ψ_f, an execution δ_f, and a set of processes $\Phi_f^{i,R}$, for each $i \in \{0,1\}$ and each $R \notin \mathcal{R}_i$, such that

$$\sum_{\substack{j\in\{0,1\}\\ R\notin\mathcal{R}_j}} |\Phi_f^{j,R}| = f, \tag{2.1}$$

(2.2) $\Phi_f^{i,R} \subseteq \Pi^{i,R}(\delta_f) - \Psi_f$, for all $i \in \{0,1\}$ and $R \notin \mathcal{R}_i$,

(2.3) $|\Pi^{i,R}(\delta_f) - \Psi_f| \geq T(d) - f \cdot (k+2)$, for all $i \in \{0,1\}$ and all $R \in \mathcal{R}_i$,

(2.4) $|\Psi_f| \leq |\Psi| + f$,

(2.5) the number of idle processes at the end of δ_f is at least $I(d) - 2f$, and

(2.6) δ_f is Ψ_f-shrinkable.

LEVEL-2 BASE CASE. When $f = 0$, let $\delta_0 = \beta$, let $\Phi_0^{i,R} = \emptyset$ for all $i \in \{0,1\}$ and all $R \notin \mathcal{R}_i$, and let $\Psi_0 = \Psi$.

LEVEL-2 INDUCTION STEP. Let $0 < f \leq (2r-d)(T(d+1)-1)+1$ and assume the claim holds for $f-1$. Let V be the set of values that appear as the parameters of UPDATES in δ_{f-1}. Let q and t be any two processes that are idle at the end of δ_{f-1}. They play the same role as in Section 3. Part (2.5) of the induction hypothesis can be used to prove that they exist. To construct δ_f, we attempt to inductively construct executions $\alpha_0, \ldots, \alpha_k$, as in Section 3. This is the third level of induction.

LEVEL-3 INDUCTION CLAIM. Either we can construct Ψ_f, δ_f, and $\Phi_f^{i,R}$ (for all $i \in \{0,1\}$ and all $R \notin \mathcal{R}_i$) that satisfy the level-2 claim or there exist indices $x_0, \ldots, x_{k+1} \in \{0,1\}$, registers $R_1, \ldots, R_{k+1}$, processes $p_1, \ldots, p_k$, and executions $\alpha_0, \ldots, \alpha_k$, where, for each $0 \leq l \leq k$,

- $\alpha_l = \delta_{f-1} \cdot \gamma_l$
- $\gamma_0 = U_0 \cdot L_0$; if $l \geq 1$, $\gamma_l = \gamma_{l-1} \cdot (w_l \cdot L_l \cdot Z_l \cdot U_l \cdot S_l)$,
- if $l \geq 1$, w_l is a `write` by p_l to register $R_l \in \mathcal{R}_{x_l}$ during its execution of a SCAN or an UPDATE to component A_{x_l},
- U_l is a complete solo execution of an UPDATE to component A_{x_l} by q with the hidden value h,
- $L_0 \cdot L_1 \cdots L_l$ is a prefix of the sequence of steps performed by process t to do a single SCAN,
- if $l \geq 1$, Z_l is a solo execution by clone p_l that completes its pending operation, and
- if $l \geq 1$, S_l is a complete solo execution of a SCAN by process q,

such that

(3.1) L_0 contains no `reads` of registers in $\mathcal{R}$; if $l \geq 1$, L_l starts with a `read` of register R_l and contains no `reads` of registers in $\mathcal{R} - \{R_l\}$; at the end of α_l, t is poised to read R_{l+1},

(3.2) during γ_l, processes performing UPDATES to component A_i write only to registers in $\mathcal{R}_i$ (for each $i \in \{0,1\}$), and processes other than t performing SCANS write only to registers in $\mathcal{R}$,

(3.3) if $l \geq 1$, S_l returns the hidden value h for both components, and

(3.4) if $l \geq 1$, $p_l \notin \Psi_{f-1} \cup \bigcup_{\substack{j\in\{0,1\}\\ R\notin\mathcal{R}_j}} \Phi_{f-1}^{j,R}$.

The proof of this claim has the same overall structure as the level-2 induction in the proof of Lemma 1. Here we only sketch the new ideas.

LEVEL-3 BASE CASE($l = 0$). Let $i \in \{0,1\}$ and let $U(i,h)$ be the sequence of steps that q takes when doing a solo UPDATE to component A_i with a hidden value $h \notin V$, starting from the final configuration of δ_{f-1}. If $U(i,h)$ does not write outside of $\mathcal{R}_i$, the proof is the same as in the anonymous case. Suppose $U(i,h)$ does write to a register outside $\mathcal{R}_i$. Let $\hat{R}$ be the first such register and let $\hat{U}$ be the prefix of $U(i,h)$ up to, but not including, this `write`. We define Ψ_f, δ_f, and $\Phi_f^{j,R}$, for all $j \in \{0,1\}$ and all $R \notin \mathcal{R}_j$, to satisfy the level-2 induction claim.

Part (2.3) of the induction hypothesis can be used to show that there is a set of processes Λ disjoint from $\Psi_{f-1} \cup \bigcup_{\substack{j\in\{0,1\}\\ R\notin\mathcal{R}_j}} \Phi_{f-1}^{j,R}$ that contains one process poised to write to each register in $\mathcal{R}_i$ at the end of δ_{f-1}. Then a block `write` W by the processes in Λ hides all evidence of q's pending UPDATE $U(i,h)$. Let $\delta_f = \delta_{f-1} \cdot \hat{U} \cdot W \cdot Z$, where Z is a sequence of steps where the processes in Λ all complete their pending operations (to ensure shrinkability). Let $\Psi_f = \Psi_{f-1}$. Let $\Phi_f^{i,\hat{R}} = \Phi_{f-1}^{i,\hat{R}} \cup \{q\}$ and let $\Phi_f^{j,R} = \Phi_{f-1}^{j,R}$ for all other pairs (j,R) with $j \in \{0,1\}$ and $R \notin \mathcal{R}_j$. Part (2.1) of the level-2 claim follows from these definitions and part (2.1) of the induction hypothesis. Part (2.4) follows from the level-2 induction hypothesis and the fact that $\Psi_f = \Psi_{f-1}$. Part (2.3) follows from part (2.3) of the induction hypothesis and the fact that at most one process poised at each register at the end of δ_{f-1} takes steps during $\hat{U} \cdot W \cdot Z$. Part (2.5) is implied by part (2.5) of the induction hypothesis and the fact that q is the only process that is idle at the end of δ_{f-1} and takes steps during $\hat{U} \cdot W \cdot Z$. One process was added to $\Phi_f^{i,\hat{R}}$, and it was chosen to satisfy part (2.2). All other processes in the sets $\Phi_f^{j,R}$ take no steps in $\hat{U} \cdot W \cdot Z$, so they are still poised at R at the end of δ_f. Part (2.2) follows.

We now prove part (2.6). Let Υ be any set of processes with pending UPDATES at the end of δ_f. Then $\Upsilon \cap \Lambda = \emptyset$, since all processes in Λ run their operations to completion in δ_f. We show that there is a set $\Sigma \subseteq \Psi_f$ such that $\delta_f(\Upsilon, \Sigma)$ is a legal execution that cannot be distinguished from δ_f by any process not in $\Upsilon \cup \Sigma$.

CASE 1: $q \notin \Upsilon$. Since δ_{f-1} is Ψ_{f-1}-shrinkable, there exists a set $\Sigma \subseteq \Psi_{f-1} = \Psi_f$ such that $\delta_{f-1}(\Upsilon, \Sigma)$ is legal and indistinguishable from δ_{f-1} by any process not in $\Upsilon \cup \Sigma$. Furthermore, $\Sigma \subseteq \Psi_{f-1}$ implies that $\Sigma \cap \Lambda = \emptyset$. As remarked above, $\Upsilon \cap \Lambda = \emptyset$. We also have that $q \notin \Sigma$, since q is idle at the end of δ_{f-1}. So, no process in $\Upsilon \cup \Sigma$ takes steps during $\hat{U} \cdot W \cdot Z$ and $\delta_f(\Upsilon, \Sigma) = \delta_{f-1}(\Upsilon, \Sigma) \cdot \hat{U} \cdot W \cdot Z$. This execution is legal and indistinguishable from δ_f by all processes outside $\Upsilon \cup \Sigma$.

CASE 2: $q \in \Upsilon$. Let $\Upsilon' = \Upsilon - \{q\}$. All processes in Υ' have pending operations at the end of δ_{f-1}, which is Ψ_{f-1}-shrinkable by the level-2 induction hypothesis. So there is a set $\Sigma \subseteq \Psi_{f-1} = \Psi_f$ such that $\delta_{f-1}(\Upsilon', \Sigma)$ is legal and indistinguishable from δ_{f-1} by any process outside $\Upsilon' \cup \Sigma$. Furthermore, $\Sigma \subseteq \Psi_{f-1}$ implies that $\Lambda \cap \Sigma = \emptyset$. So no processes in $\Upsilon' \cup \Sigma$ take steps in $W \cdot Z$. Then, $\delta_f(\Upsilon, \Sigma) = \delta_{f-1}(\Upsilon', \Sigma) \cdot W \cdot Z$. This execution is legal and indistinguishable from $\delta_{f-1} \cdot W \cdot Z$ by any process outside $\Upsilon' \cup \Sigma$. This latter execution is indistinguishable from δ_f by any process other than q, since W overwrites every register in $\mathcal{R}_i$. So $\delta_f(\Upsilon, \Sigma)$ is legal and indistinguishable from δ_f by any process outside $\Upsilon \cup \Sigma$.

LEVEL-3 INDUCTION STEP.: Let $0 < l \leq k$. Suppose there are $x_0, \ldots, x_l \in \{0,1\}$, registers $R_1, \ldots, R_l$, processes

$p_1, \ldots, p_l$ and executions $\alpha_0, \ldots, \alpha_{l-1}$ that satisfy the claim. We show how to choose x_{l+1}, R_{l+1} and α_l.

We define an execution ϕ as in the anonymous case. In the anonymous case, the only pending UPDATES at the end of β_l are those performed by $p_1, \ldots, p_l$. Here, we have many more pending UPDATES at the end of δ_{f-1} to satisfy part (2.3) of the induction hypothesis. In order to argue about the linearization of the operations by processes $p_1, \ldots, p_l$, we remove all other UPDATES pending at the end of δ_{f-1}. This is one place where we use the shrinkability property guaranteed by the level-2 induction hypothesis. Let Υ be the set of processes other than $p_1, \ldots, p_l$ that have pending UPDATES at the end of δ_{f-1}. Since δ_{f-1} is Ψ_{f-1}-shrinkable, there is a set $\Sigma \subseteq \Psi_{f-1}$ such that $\delta_{f-1}(\Upsilon, \Sigma)$ is legal and indistinguishable from δ_{f-1} by any process outside of $\Upsilon \cup \Sigma$. Let $\phi = \delta_{f-1}(\Upsilon, \Sigma) \cdot \gamma_{l-1} \cdot w_l \cdot L$, where w_l is a write to R_l by some process $p_l \in \Pi^{x_l, R_l}(\delta_{f-1}) - \Psi_{f-1} - \{p_1, \ldots, p_{l-1}\} - \bigcup_{\substack{j \in \{0,1\} \\ R \notin \mathcal{R}_j}} \Phi_{f-1}^{j,R}$ and L is the sequence of steps t takes after ϕ to complete its SCAN. Using part (2.3) of the induction hypothesis, it can be proved that such a process p_l exists. Claims 1.1 and 1.2 can be proved for ϕ in a way similar to the anonymous case.

Let R_{l+1} be the first register in $\mathcal{R} - \{R_l\}$ that t reads during L. Let L_l be the prefix of L up to (but not including) that read. Pick x_{l+1} so that $R_{l+1} \in \mathcal{R}_{x_{l+1}}$. Let $\alpha_l = \delta_{f-1} \cdot \gamma_{l-1} \cdot w_l \cdot L_l \cdot Z_l \cdot U_l \cdot S_l$ where Z_l, U_l, and S_l are defined as in the anonymous case.

If, during $w_l \cdot L_l \cdot Z_l \cdot U_l \cdot S_l$, no process performing an UPDATE to A_i or a SCAN writes outside $\mathcal{R}_i$, then we can prove α_l satisfies the level-3 induction claim using an argument similar to the one in the anonymous model. Otherwise, let $\hat{p}$ be the process that does the first such write. We consider the prefix of α_l up to, but not including, this write. To define δ_f, we extend this prefix as follows. A set of $|\mathcal{R}|$ processes not in $\Psi_{f-1} \cup \{p_1, \ldots, p_l\} \cup \bigcup_{\substack{j \in \{0,1\} \\ R \notin \mathcal{R}_j}} \Phi_{f-1}^{j,R}$ do a block write, which obliterates all evidence of $\hat{p}$'s pending operation, and then these processes run to completion. (The existence of these processes poised at the desired locations can be proved using parts (2.1) and (2.3) of the induction hypothesis.) We add $\hat{p}$ to one of the sets $\Phi_{f-1}^{i,R}$ to form $\Phi_f^{i,R}$. If $\hat{p} = p_l$ and its pending operation is an UPDATE, let $\Psi_f = \Psi_{f-1} \cup \{t\}$ because the first register t reads during L_l is the one where $\hat{p}$ does its write w_l. In the other cases, let $\Psi_f = \Psi_{f-1}$. Then we can prove that δ_f is Ψ_f-shrinkable using a lengthy case analysis and techniques similar to those used in the level-3 base case. This completes the level-3 induction.

When $l = k$, t's SCAN in α_k has taken k steps and is poised to take one more, contradicting the definition of k. So the level-3 induction claim for $l = k$ can only be satisfied if there exists a δ_f that satisfies the level-2 claim. This completes the level-2 induction proof.

Part (2.1) of the level-2 induction claim states that $\sum_{\substack{j \in \{0,1\} \\ R \notin \mathcal{R}_j}} |\Phi_f^{j,R}| = f$. The number of terms in the sum is $2r - |\mathcal{R}_0| - |\mathcal{R}_1| = 2r - d$. When $f = (2r-d)(T(d+1)-1)+1$, there must be at least one pair (i', R') with $i' \in \{0,1\}$ and $R' \notin \mathcal{R}_{i'}$ such that $|\Phi_f^{i',R'} - \Psi_f| = |\Phi_f^{i',R'}| \geq T(d+1)$. We use claim (2.3) to prove that, for all $i \in \{0,1\}$ and all $R \in \mathcal{R}_i$, $|\Pi^{i,R}(\delta_f) - \Psi_f| \geq T(d+1)$. Let $\beta' = \delta_f$. Let $\Psi' = \Psi_f$. Then β' is a Ψ'-shrinkable execution, by part (2.6) of the claim. We use part (2.5) of the claim to prove there are at least $I(d) - 2f \geq I(d+1)$ idle processes at the end of β'. Then we can add R' to $\mathcal{R}_0$ or $\mathcal{R}_1$ to complete the level-1 induction. □

Acknowledgements

This research was supported by the Natural Sciences and Engineering Research Council of Canada and the Scalable Synchronization Research Group of Sun Microsystems.

5. REFERENCES

[1] Yehuda Afek, Hagit Attiya, Danny Dolev, Eli Gafni, Michael Merritt, and Nir Shavit. Atomic snapshots of shared memory. *Journal of the ACM*, 40(4), pages 873–890, September 1993.

[2] Yehuda Afek and Gideon Stupp. Optimal time-space tradeoff for shared memory leader election. *Journal of Algorithms*, 25(1), pages 95–117, October 1997.

[3] Rajeev Alur and Gadi Taubenfeld. Contention-free complexity of shared memory algorithms. *Information and Computation*, 126(1), pages 62–73, April 1996.

[4] James H. Anderson. Composite registers. *Distributed Computing*, 6(3), pages 141–154, April 1993.

[5] James H. Anderson. Multi-writer composite registers. *Distributed Computing*, 7(4), pages 175–195, 1994.

[6] James Aspnes. Time- and space-efficient randomized consensus. *Journal of Algorithms*, 14(3), pages 414–431, May 1993.

[7] James Aspnes and Maurice Herlihy. Fast, randomized consensus using shared memory. *Journal of Algorithms*, 11(2), pages 441–461, September 1990.

[8] James Aspnes and Maurice Herlihy. Wait-free data structures in the asynchronous PRAM model. In *Proc. 2nd ACM Symposium on Parallel Algorithms and Architectures*, pages 340–349, 1990.

[9] Hagit Attiya, Panagiota Fatourou, and Faith Ellen Fich. A lower bound on update time for multi-writer snapshot objects. Manuscript, 2004.

[10] Hagit Attiya, Nancy Lynch, and Nir Shavit. Are wait-free algorithms fast? *Journal of the ACM*, 41(4), pages 725–763, July 1994.

[11] Hagit Attiya and Ophir Rachman. Atomic snapshots in $O(n \log n)$ operations. *SIAM Journal on Computing*, 27(2), pages 319–340, April 1998.

[12] Hagit Attiya and Jennifer Welch. *Distributed Computing: Fundamentals, Simulations and Advanced Topics, 2nd edition.* Wiley, 2004.

[13] Oliver Berthold, Hannes Federrath, and Marit Köhntopp. Project "anonymity and unobservability in the internet". In *Proc. 10th Conference on Computers, Freedom and Privacy*, pages 57–65, 2000.

[14] James Burns and Nancy Lynch. Bounds on shared memory for mutual exclusion. *Information and Computation*, 107(2), pages 171–184, December 1993.

[15] Panagiota Fatourou, Faith Fich, and Eric Ruppert. Space-optimal multi-writer snapshot

objects are slow. In *Proceedings of the 21st Annual ACM Symposium on Principles of Distributed Computing*, pages 13–20, 2002.

[16] Panagiota Fatourou, Faith Fich, and Eric Ruppert. A tight time lower bound for space-optimal implementations of multi-writer snapshots. In *Proceedings of the 35th ACM Symposium on Theory of Computing*, pages 259–268, 2003.

[17] Faith Fich, Maurice Herlihy, and Nir Shavit. On the space complexity of randomized synchronization. *Journal of the ACM*, 45(5), pages 843–862, September 1998.

[18] Faith Fich and Eric Ruppert. Hundreds of impossibility results for distributed computing. *Distributed Computing*, 16(2-3), pages 121–163, 2003.

[19] Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2), pages 374–382, April 1985.

[20] Rainer Gawlick, Nancy Lynch, and Nir Shavit. Concurrent timestamping made simple. In *Proceedings of the Israel Symposium on the Theory of Computing and Systems*, volume 601 of *LNCS*, pages 171–183, 1992.

[21] Rachid Guerraoui and Eric Ruppert. What can be implemented anonymously? In *Proc. 19th International Symposium on Distributed Computing*, volume 3724 of *LNCS*, pages 244–259, 2005.

[22] Danny Hendler and Nir Shavit. Operation-valency and the cost of coordination. In *Proceedings of the 22nd Annual ACM Symposium on Principles of Distributed Computing*, pages 84–91, 2003.

[23] Maurice P. Herlihy and Jeannette M. Wing. Linearizability: A correctness condition for concurrent objects. *ACM Transactions on Programming Languages and Systems*, 12(3), pages 463–492, July 1990.

[24] Michiko Inoue, Wei Chen, Toshimitsu Masuzawa, and Nobuki Tokura. Linear time snapshots using multi-writer multi-reader registers. In *Distributed Algorithms, 8th International Workshop*, volume 857 of *LNCS*, pages 130–140, 1994.

[25] Amos Israeli and Asaf Shirazi. The time complexity of updating snapshot memories. *Information Processing Letters*, 65(1), pages 33–40, January 1998.

[26] Prasad Jayanti, King Tan, and Sam Toueg. Time and space lower bounds for nonblocking implementations. *SIAM Journal on Computing*, 30(2), pages 438–456, 2000.

[27] George Lucas and Jonathan Hales. *Star Wars: Episode II–Attack of the Clones* [motion picture]. Twentieth Century Fox, 2002.

[28] Nancy A. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.

[29] Michael K. Reiter and Aviel D. Rubin. Crowds: Anonymity for web transactions. *ACM Transactions on Information and System Security*, 1(1), pages 66–92, November 1998.

[30] Eric Schenk. Faster approximate agreement with multi-writer registers. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science*, pages 714–723, 1995.