The Complexity of Robust Atomic Storage

Dan Dobre¹, Rachid Guerraoui², Matthias Majuntke¹, Neeraj Suri¹ and Marko Vukolić³

¹ TU Darmstadt, Hochschulstr. 10, 64289 Darmstadt, Germany, phone: +496151165657
{dan,majuntke,suri}@cs.tu-darmstadt.de
² IC, EPFL, CH-1015 Lausanne, Switzerland, phone: +41216935272
rachid.guerraoui@epfl.ch
³ IBM Research - Zurich, Switzerland, phone: +41447248715
mvu@zurich.ibm.com

Abstract. We study the time-complexity of robust atomic read/write storage from fault-prone storage components in asynchronous message-passing systems. Robustness here means wait-free tolerating the largest possible number t of Byzantine storage component failures (optimal resilience) without relying on data authentication. We show that no single-writer multiple-reader (SWMR) robust atomic storage implementation exists if (a) read operations complete in less than four communication round-trips (rounds), and (b) the time-complexity of write operations is constant. More precisely, we present two lower bounds. The first is a read lower bound stating that three rounds of communication are necessary to read from a SWMR robust atomic storage. The second is a write lower bound, showing that $\Omega(\log(t))$ write rounds are necessary to read in three rounds from such a storage. Applied to known results, our lower bounds close a fundamental gap: we show that time-optimal robust atomic storage can be obtained using well-known transformations from regular to atomic storage and existing time-optimal regular storage implementations.

Regular Paper (eligible for best student paper award).
1 Introduction

Background. Variable sharing is critical to modern distributed and concurrent computing. The atomic read/write register abstraction [17] is essential to sharing information in distributed systems; it abstracts away the complexity incurred by concurrent access to shared data by providing processes an illusion of sequential access to data. This abstraction is also referred to as atomic storage, for its importance as a building-block in practical distributed storage and file systems (see e.g., [23, 24]). Besides, its read/write API, despite being very simple, is today the heart of modern “cloud” key-value storage APIs (e.g., [5]).

In this paper, we study atomic storage implementations in asynchronous message-passing systems in which a set of reader and writer processes (clients) share data leveraging a set of storage object processes. We consider fault-tolerant, robust [3] storage implementations characterized by: a) wait-freedom [16], i.e., the fact that read/write operations invoked by correct clients always eventually return, and b) ensuring correctness despite the largest possible number \( t \) of object failures (optimal resilience). We allow for the most general type of failures, arbitrary, also called Byzantine [18] failures\(^4\) without assuming authenticated (also called self-verifying [22]) data to limit the adversary (by relying on e.g., digital signatures).

In this model, we ask a fundamental question: what is the optimal worst-case complexity of robust atomic storage implementations? Our complexity metric is an important one: time-complexity, or latency, measured in number of communication round-trips (or simply rounds) between a client and objects. The relevance of the question we ask extends beyond theoretical. Namely, with the growth in storage outsourcing driven by the advent of cloud computing, the arbitrary failure model becomes increasingly relevant in absence of the full trust in the cloud [6]. In addition, the number of interactions with the remote cloud storage needed to access the data, maps to our latency metric and is often directly associated with the monetary cost; this obviously increases further the practical relevance of the question we ask.

Perhaps surprisingly and despite the wealth of literature exploring latency-optimal storage implementations, this question has not been answered. It is known that the worst-case latency of writing into robust storage is at least 2 rounds [1]. In this paper, we show that the optimal worst-case latency of reading from scalable robust atomic storage is 4 (four) rounds. Here, the notion of scalability captures two basic criteria: a) support for any number of readers, and b) constant write-latency. Our results close a fundamental gap, showing that latency-optimal scalable and robust atomic storage, combining 2-round writes and 4-round reads, can be achieved (in the case of single-writer multi-reader (SWMR) storage) using standard transformations from weaker, regular [17] registers to the atomic ones [4, 19].

Our contribution goes through proving two lower bounds. However, to help fully appreciate our contributions, we first discuss how the scope of this paper fits into related work.

Related work. Several papers have explored the time-complexity metric in the context of a read/write register abstraction. A seminal crash-tolerant robust atomic SWMR register implementation assuming a majority of correct processes was presented in [3]. In [3], all write operations complete in a single round; on the other hand, read operations always take two rounds between a client and objects.

The problem of modifying [3] to enable single round reads was explored in [8], which showed that such fast atomic implementations are possible albeit they come with the price of limited number of readers and suboptimal resilience. Moreover, the reader in [8] needs to write (i.e., modify the objects’ state) as dictated by the lower bound of [11] which showed that every atomic read must write into at least \( t \) objects. [9] extends the result of [8] to the Byzantine failure model assuming authenticated (i.e., digitally signed) data and established the impossibility of fast crash-tolerant multi-writer multi-reader (MWMR) atomic reg-

\(^{4}\)In the Byzantine failure model, optimal resilience corresponds to using \( 3t + 1 \) objects to tolerate \( t \) failures [22].
ister implementations. This result is in line with classical MWMR implementations such as [21] that have read/write latency of at least 2 rounds. The limitation on the number of readers of [8], was relaxed in [12], where a crash-tolerant robust SWMR atomic register implementation was presented, in which most of the reads complete in a single round, yet a fraction of reads is permitted to be slow and complete in 2 rounds.

In the Byzantine context, optimizing latency is particularly interesting when data is assumed to be unauthenticated, which we also assume here. [1] showed that any Byzantine-tolerant storage employing at most $4t$ storage objects has at least some write operation complete in 2 rounds. Moreover, [1] showed a tight lower bound of $t + 1$ rounds from reading from robust SWMR safe [17] storage, with the constraint that readers are precluded from writing. However, allowing readers to write helps improve latency as shown in [14], through a 2-round tight lower bound on reading from robust SWMR regular [17] storage. This bound was circumvented in [7], assuming storage of secret values, used to detect concurrent operations, where reads are expedited to complete in a single round. However, none of these papers dealt with optimal worst-case latency of reading from robust atomic storage, which is precisely the scope of our paper.

On the other hand, few papers have explored the best-case complexity of Byzantine-tolerant optimally resilient atomic storage. Here, “best-case” encompasses synchrony, no or few object failures and the absence of read/write concurrency. In this context, [13] presented the first robust atomic storage implementation in which both reads and writes are fast in the best-case (i.e., complete in a single round-trip). Furthermore, [15] considered robust atomic storage implementations with the possibility of having fast reads and writes gracefully degrade to 2 or 3 rounds, depending on the size of the available quorum of correct objects. Unlike these papers, we are interested here with the unconditional, worst-case latency of atomic storage.

Finally, the worst-case read latency in existing Byzantine-tolerant robust atomic storage implementations for unauthenticated data (e.g., [2, 13, 15, 22]) is either unbounded or $\Omega(t)$ rounds at best [2].

**Contributions.** We present two lower bounds (impossibility results) on time-complexity of reading from robust atomic storage for unauthenticated data, implemented from storage objects prone to Byzantine faults. Together, our lower bounds imply that there is no scalable robust atomic storage implementation in the Byzantine unauthenticated model in which all reads complete in less than 4 rounds.

- The first lower bound, referred to as the read lower bound, demonstrates the impossibility of reading from robust SWMR atomic storage in two rounds. More precisely, we show that if the number of storage objects $S$ is at most $4t$ and if the number of readers $R$ is greater than 3, then no SWMR atomic implementation may have all reads complete in two rounds.

  Our proof scheme resembles that of [8] and relies on sequentially appending reads on a write operation, while progressively deleting the steps of a write and preceding read operations, exploiting asynchrony and possible failures. This deletion ultimately allows reusing readers and reaching an impossibility with as few as $R = 4$ readers. As none of these appended operations are concurrent under step contention, the impossibility also holds in the stronger data model of [7], in which the adversary is unable to simulate step contention among operations, making use of secret values.

- Our second lower bound, referred to as the write lower bound, shows that if read operations are required to complete in three communication rounds, then the number of write rounds $k$ is $\Omega(\log(t))$. More precisely, we show that if the number of storage objects is at most $3t + \lfloor t/k \rfloor$ and $R \geq k$, then no implementation of a SWMR atomic storage may have all reads complete in three rounds and all writes in $k \leq \lfloor \log(\lceil \frac{3t+1}{2} \rceil) \rfloor$ rounds. In a sense, our lower bound generalizes the write lower bound of [1], which proves our result for the special case of $k = 1$.

  While using a similar approach, the write lower bound proof is much more involved and differs from our read lower bound proof in several key aspects. Due to the additional third read round, read steps
cannot be entirely deleted, which prohibits the reuse of readers. Consequently, the number of supported readers \( R \) and the number of write rounds \( k \) are related \( (R \geq k) \). Furthermore, the proof relies on a set of malicious objects that forges critical steps of the write and of prior reads with respect to subsequent reads. This set grows with the number of appended reads, relating the number of faulty objects \( t \) and the number of readers (which is at least \( k \)). At the heart of the proof we use a recurrent formula that relates \( t \) and \( k \), similar to a Fibonacci sequence, which describes the exact relation between the two parameters. In its closed form, the formula transforms to the \( \log \) function \( (k = \Omega(\log(t))) \).

The rest of the paper is organized as follows. In Section 2 we give our model and definitions. Sections 3 and 4 gives the proof of our read lower bound. Section 5 gives the proof of our write lower bound. Section 5 concludes the paper by discussing modular implementations that match our lower bounds.

2 Model

Basics. The distributed system we consider consists of three disjoint sets of processes: a set objects of size \( S \) containing processes \( \{s_1, \ldots, s_S\} \) and representing the base register elements; a singleton writer containing a single process \( \{w\} \); and a set readers of size \( R \) containing processes \( r_1, \ldots, r_R \). The set clients is the union of the sets writer and readers. We assume that every client may communicate with any process by message passing using point-to-point reliable communication channels. However, objects cannot communicate among each other, nor send messages to clients other than in reply to clients’ messages.

Here we define only the notions we use in our proofs; model details can be found in [19]. A distributed algorithm \( A \) is a collection of automata [20], where automaton \( A_p \) is assigned to process \( p \). Computation proceeds in steps of \( A \); each step is denoted by a pair of process id and a set of messages received in that step \( \langle p, M \rangle \) (\( M \) might be \( \emptyset \)). A run is an infinite sequence of steps of \( A \). A partial run is a finite prefix of some run. A (partial) run \( r \) extends some partial run \( pr \) if \( pr \) is a prefix of \( r \). At the end of a partial run, all messages that are sent but not yet received are said to be in transit. In any run, any client can fail by crashing and up to \( t \) objects may be malicious. An algorithm that assumes \( S = 3t + 1 \) is said to be optimally resilient.

Atomic Storage. A register abstraction is a read/write data structure. It provides two operations: write(v), which stores \( v \) in the register, and read(), which returns the value from the register. We assume that each client invokes at most one operation at a time (i.e., does not invoke the next operation until it receives the response for the current operation). Only readers invoke read operations and only the writer invokes write operations. We further assume that the initial value of a register is a special value \( \perp \), which is not a valid input value for a write operation. We say that an operation \( op \) is complete in a (partial) run if the run contains a response step for \( op \). In any run, we say that a complete operation \( op_1 \) precedes operation \( op_2 \) (or \( op_2 \) succeeds \( op_1 \)) if the response step of \( op_1 \) precedes the invocation step of \( op_2 \) in that run. If neither \( op_1 \) nor \( op_2 \) precedes the other, then the operations are said to be concurrent.

An algorithm implements a register if every run of the algorithm satisfies wait-freedom and atomicity properties. Wait-freedom states that if a process invokes an operation, then eventually, unless that process crashes, the operation completes (even if all other client processes have crashed). Here we give a definition of atomicity for the single-writer registers. In the single-writer setting, the writes in a run have a natural ordering which corresponds to their physical order. Denote by \( wr_k \) the \( k \)th write in a run \( (k \geq 1) \), and by \( val_k \) the value written by the \( k \)th write. Let \( val_0 = \perp \). We say that a partial run satisfies atomicity if the following properties hold: (1) if a read returns \( x \) then there is \( k \) such that \( val_k = x \), (2) if a read \( rd \) is complete and it succeeds some write \( wr_k \) \( (k \geq 1) \), then \( rd \) returns \( val_l \) such that \( l \geq k \), (3) if a read \( rd \)

---

5 An extension to the model of [25] using distinct thresholds for malicious crash objects’ faults is included in Appendix A.1.
returns $val_k \ (k \geq 1)$, then $wr_k$ either precedes $rd$ or is concurrent with $rd$, and (4) if some read $rd_1$ returns $val_k \ (k \geq 0)$ and a read $rd_2$ that succeeds $rd_1$ returns $val_l$, then $l \geq k$.

**Time-complexity.** We measure the time-complexity of an atomic register implementation in terms of communication round-trips (or simply rounds). A round is defined as follows [8, 10, 12, 21]:

**Definition 1** Client $c$ performs a communication round during operation $op$ if the following conditions hold:
1. The client $c$ sends messages to a subset of the objects.
2. Objects, on receiving such a message, reply to the reader (resp. the writer) before receiving any other messages (as dictated by our model).
3. When the invoking client receives enough replies, the round ($rnd$) terminates. (either completing the operation $op$ or starting a new round).

Note that, since any number of clients can crash, we can construct partial runs in which no client receives any message from any other client. In our proofs in Section 3 and 4 we focus, without loss of generality, on such partial runs. Moreover, since up to $t$ objects might be faulty in our model, ideally, in every round $rnd$ the invoking client can only wait for reply messages from at most $S - t$ correct objects.

## 3 The Read Lower Bound

In this section we prove the following proposition.

**Proposition 1**: If $S \leq 4t$ and $R > 3$, then no read implementation $I$ of a multi-reader (SWMR) atomic register exists that completes in two rounds.

**Preliminaries.** In the proof $w$ denotes the writer, $r_i$ for $1 \leq i \leq 4$ denote the readers, and $s_i$ for $1 \leq i \leq S$ denote objects. Suppose by contradiction that $R = 4$ and there is an atomic register implementation $I$ that uses at most $4t$ objects, such that in every partial run of $I$ every read operation completes in two rounds.

We partition the set objects into four disjoint subsets (which we call blocks), denoted $B_i$ for $1 \leq i \leq 4$ each of size exactly $t \geq 1$. We refer to the initial state of every correct block $B_i$ as $\sigma_i^0$. For simplicity we simply write $\sigma_0$, where the block name is implicit.

We say that a round $rnd$ of an operation $op$ skips a set of blocks $BS$ in a partial run, (where $BS \subseteq \{B_1, \ldots, B_4\}$), if (1) no object in any block $BL \in BS$ receives any message in round $md$ from $op$ in that partial run; (2) all other objects receive all messages in round $md$ from $op$ and reply to the messages, and (3) in case round $md$ is terminated, the invoking client has received all these reply messages or, in case $rnd$ is not terminated, all these reply messages are in transit. We say that an operation $op$ skips a set of blocks $BS$ in a partial run if every round of $op$ skips $BS$.

To show a contradiction, we construct a partial run of the implementation $I$ that violates atomicity: a partial run of $I$ in which no value is ever written and some read returns 1.

**Partial writes.** Throughout the proof there is only one write operation write(1) by $w$ that writes value 1. Consider a partial run $wr$ in which $w$ completes write(1) on the register and let $k$ be the number of rounds invoked by $w$ in $wr$. We denote the state of every correct object $B_i$ after $B_i$ has replied to the messages of the write during round 1 to $i$ where $1 \leq i \leq k$, as $\sigma_i^1$. For simplicity we refer to $\sigma_i^1$ as $\sigma_i$. The write operation skips blocks $B_3$. We define a series of partial runs containing an incomplete write(1) invocation, each being a prefix of $wr$. For $1 \leq i \leq k$ and $1 \leq j \leq 4$, we define $wr_i^j$ as the partial run in which (1) rounds 1 to $i - 1$ are terminated and skip $B_3$; (2) round $i$ is not terminated and skips blocks $\{B_1, \ldots, B_4\} \setminus \{B_j \mid j \leq l \leq 3\}$, and (3) all objects are correct. We make two observations: (1) partial run $wr_i^k$ differs from $wr$ only at $w$ and (2) partial run $wr_i^1$ differs from a run in which write(1) is never invoked only at $w$. 

4
Block diagrams. We illustrate the proof in Figure 1 (a)-(g). We depict a round $\text{rnd}$ of an operation $\text{op}$ through a set of rectangles arranged in a single column. In the column corresponding to some round $\text{rnd}$ of $\text{op}$ we draw a rectangle in a given row, if all objects in the corresponding block $\text{BL}$ have received the message from the client in round $\text{rnd}$ of $\text{op}$ and have sent reply messages, i.e., if round $\text{rnd}$ of $\text{op}$ does not skip $\text{BL}$. We write “@” in the row corresponding to $\text{BL}$ iff $\text{BL}$ is malicious.

Appending reads. Partial run $\text{pr}_1$ extends $\text{wr}$ by appending a complete read $\text{rd}_1$ by $\text{r}_1$ that skips $\text{B}_2$ in round one and $\text{B}_1$ in round two (see Figure 1 (a)). In $\text{pr}_1$, all objects in block $\text{B}_1$ are malicious, and forge their state to $\sigma_{k-1}$ before replying to $\text{rd}_1$. By atomicity $\text{rd}_1$ returns 1. Observe that $\text{r}_1$ cannot distinguish $\text{pr}_1$ from some partial $\Delta\text{pr}_1$ that extends $\text{wr}_k^1$ by appending $\text{rd}_1$, and where all objects are correct.

Partial run $\text{pr}_2$ extends $\Delta\text{pr}_1$ by appending a complete read $\text{rd}_2$ by $\text{r}_2$ that skips $\text{B}_3$ in round one and $\text{B}_2$ in round two (see Figure 1 (a) after deleting the crossed steps). In $\text{pr}_2$, all objects in block $\text{B}_2$ are malicious, and forge their state to $\sigma_{k-1}$ before replying to $\text{rd}_2$. By atomicity $\text{rd}_2$ returns 1. Observe that $\text{r}_2$ cannot distinguish $\text{pr}_2$ from some partial run $\Delta\text{pr}_2$, that extends $\text{wr}_k^2$ by appending an incomplete $\text{rd}_1$ and a complete $\text{rd}_2$ and where all objects are correct (Figure 1 (b) after deleting the crossed steps).

Partial run $\text{pr}_3$ extends $\Delta\text{pr}_2$ by appending a complete read $\text{rd}_3$ by $\text{r}_3$ that skips $\text{B}_3$ in round one and $\text{B}_3$ in round two. In $\text{pr}_3$, all objects in block $\text{B}_3$ are malicious, and forge their state to $\sigma_{k-1}$ before replying to $\text{rd}_3$. By atomicity $\text{rd}_3$ returns 1. Let $\sigma_1^k$ denote the state of the objects in block $\text{B}_4$ in run $\text{pr}_3$ before replying to $\text{rd}_2$. Observe that $\text{r}_3$ cannot distinguish $\text{pr}_3$ from some partial run $\Delta\text{pr}_3$, that extends $\text{wr}_k^3$ by appending incomplete reads $\text{rd}_1$ and $\text{rd}_2$ and a complete read $\text{rd}_3$ and in which (1) all objects in $\text{B}_4$ are malicious and (2) they forge their state to $\sigma_1^k$ before replying to $\text{rd}_2$ (Figure 1 (c) after deleting the crossed steps). Note that since $\text{r}_3$ cannot distinguish $\text{pr}_3$ and $\Delta\text{pr}_3$, it cannot wait for more replies in order to complete. More
precisely, rd₃ completes in Δpr₃ (resp. pr₃) the first (resp. second) round based on replies only from correct objects (at least S − t).

Partial run pr₄ (illustrated in Figure 1(d)) extends Δpr₃ by appending a complete read rd₄ by r₄ that skips B₁ in round one and B₄ in round two. In pr₄, all objects in block B₄ are malicious and forge their state (1) to σ₂ before replying to rd₂ and (2) to σ₀ before replying to rd₄. By atomicity rd₄ returns 1. Let σ₂ denote the state of the objects in block B₁ before replying to rd₃. Observe that r₄ cannot distinguish pr₄ from some partial run Δpr₄, that extends wrₜ−¹ by appending incomplete reads rd₂, rd₃ and a complete read rd₄, and in which (1) all objects in B₁ are malicious and (2) they forge their state to σ₂ before replying to rd₃ (Figure 1(d) after deleting the crossed steps). Since r₄ cannot distinguish pr₄ and Δpr₄, rd₄ cannot wait for additional replies in order to complete, without violating termination. To see why, notice that in run Δpr₄ (resp pr₄), rd₄ receives first (resp. second) round replies from S − t correct objects.

After appending rd₄ and constructing Δpr₄ by deleting all steps from pr₄ which are not “visible” to rd₄, we notice that we have erased all steps in column k of write(1) as well as, deleted all steps of rd₁. Thus, we can recycle r₁ by appending rd₁ again and start deleting the steps in column k − 1.

Starting from Δpr₄ we iteratively define the following partial runs for 1 ≤ i ≤ k − 1 and 1 ≤ j ≤ 4 (see Figure 1(d)-(g)). Partial run pr₄ᵢ+ⱼ extends Δpr₄ᵢ+ⱼ−₁ by appending rdⱼ. In pr₄ᵢ+ⱼ, all objects in block Bⱼ are malicious and they forge their state (1) to σᵢ₊ⱼ before replying to rdⱼ−ᵢ₊ⱼ and (2) to σᵢ₊ⱼ−₁ before replying to rdⱼ. Let σᵢ₊ⱼ denote the state of the objects in block Bⱼ before replying to rdⱼ−₁. Observe that rⱼ cannot distinguish pr₄ᵢ+ⱼ from some partial run Δpr₄ᵢ+ⱼ, that extends wrᵢ₊ⱼ−₁ by appending incomplete reads rdⱼ−₁ and a complete read rdⱼ, and in which (1) all objects in Bⱼ are malicious, and (2) they forge their state to σᵢ₊ⱼ−₁ before replying to rdⱼ−₁ (Figure 1(d)-(g) after deleting the crossed steps). Since rⱼ cannot distinguish Δpr₄ᵢ+ⱼ and pr₄ᵢ+ⱼ, rdⱼ cannot wait for additional replies, because in run Δpr₄ᵢ+ⱼ and pr₄ᵢ+ⱼ, rdⱼ receives first and second round replies respectively from S − t correct objects.

Read rd₄ in Δpr₄ returns 1. Since pr₅ extends Δpr₄ by appending rd₁, by atomicity, rd₁ in pr₅ returns 1. However, as r₁ cannot distinguish pr₅ from Δpr₅, rd₁ in Δpr₅ returns 1. In general, since pr₄ᵢ+ⱼ extends Δpr₄ᵢ+ⱼ−₁ by appending rdⱼ (for 1 ≤ i ≤ k − 1 and 1 ≤ j ≤ 4), and rⱼ cannot distinguish pr₄ᵢ+ⱼ from Δpr₄ᵢ+ⱼ, it follows by induction that rdⱼ in Δpr₄ᵢ+ⱼ returns 1. In particular, rd₃ reads 1 in Δprₔₖ−₁. By our construction, Δprₔₖ−₁ extends wrₜ₊₁ and wrₜ₊₂ is indistinguishable from a run in which write(1) is never invoked. Hence, rd₃ returns 1 even if no write is invoked, violating atomicity.

4 The Write Lower Bound

In this section we prove the following proposition.

Proposition 2: If S ≤ 3t + ⌊t/k⌋ and every read completes in three rounds then no write implementation I of a multi-reader atomic register exists that completes in min{R, ⌊log((3t + 1)/2)⌋} rounds.

We first prove the following key lemma. In the effort of making its involved proof easier to follow we first proceed through a careful proof setup that we found worthwhile. To further help follow the proof, we also visualize runs we use in the proof in Figure 2.

Lemma 1. Let k ≥ 1, t−₁ = t₀ = 0 and tₖ = tₖ−₁ + 2tₖ−₂ + 1. There is no implementation I of a k-reader atomic storage with 3tₖ + 1 objects and tₖ faults such that the write completes in k rounds and the read completes in three rounds.

6 Please note that when we write rdⱼ−₁, we always mean rdⱼ−₁(−(c−j) mod 4).
Preliminaries. Recall that $w$ denotes the writer, $r_i$ for $1 \leq i \leq k$ denote the readers, and $s_i$ for $1 \leq i \leq S$ denote the objects. The initial value of the register is $\perp$. In the proof, there is only one write operation $\text{write}(1)$ by $w$ that writes value 1. We know from [1] that the lemma is true for $k = 1$; hence, we assume $k \geq 2$. Suppose by contradiction that there is an implementation $I$ that uses at most $3t_k + 1$ objects, such that in every partial run of $I$ every write (resp., read) completes in $k$ (resp. 3) rounds.

We partition the set objects into $2k$ distinct blocks, $B_0, \ldots, B_{k+1}$ and $C_1, \ldots, C_k$ such that $|\bigcup_{j=0}^{k+1} B_j| = 2t_k + 1$ and $|\bigcup_{j=1}^{k} C_j| = t_k$. Block $B_0$ contains a single object. For $1 \leq l \leq k$, the size of $B_l$ is $t_l - t_{l-2}$ and the size of $B_{k+1}$ is $2t_k + 1 - |\bigcup_{j=0}^{k} B_j| = t_k - t_{k-1}$. For $1 \leq l \leq k - 1$, the size of $C_l$ is $t_{l-1} - t_{l-2}$ and the size of $C_k$ is $t_k - |\bigcup_{j=1}^{k-1} C_j| = t_k - t_{k-2}$. Note here that $C_1$ is empty. Moreover, we use the abbreviation $BL_{i,j}$ to denote the set $\{BL_i, BL_j\}$, for some $BL \in \{B,C\}$.

We also define three sets of blocks called superblocks: the “malicious” superblock $\mathcal{M}_l$, the “parity” superblock $\mathcal{P}_l$ and the “correct” superblock $\mathcal{C}_l$. Superblock $\mathcal{M}_l$ contains all blocks with index at most $l$, i.e., $\mathcal{M}_l = \{B_0, B_1, \ldots, B_l, C_1, \ldots, C_l\}$ for $0 \leq l \leq k - 1$, and $\mathcal{M}_k = \emptyset$. Superblock $\mathcal{P}_l$ contains all blocks $B_j$ with index $j \geq l + 1$ such that $j$ and $l$ have the same parity. More formally, for $1 \leq l \leq k$, we define $\mathcal{P}_l = \{B_j | l \leq j \leq k - 1 \land j \equiv (l \mod 2)\}$. For instance, if $k$ is even then $\mathcal{P}_1 = \{B_1, B_3, \ldots, B_{k-1}, B_{k+1}\}$ and $\mathcal{P}_2 = \{B_2, B_4, \ldots, B_{k-2}, B_k\}$. Finally, superblock $\mathcal{C}_l = \{C_1, \ldots, C_k\}$.

Given the size of the individual blocks, we can determine the cardinality of the union of all elements of a superblock. Namely, if $S \in \{\mathcal{M}_l, \mathcal{P}_l, \mathcal{C}_l\}$, then we define the union of its elements as $\bigcup S = \{s \in BL | BL \in S\}$. Having in mind that $t_k = t_{k-1} + 2t_{k-2} + 1$ (Def.) and $t_{-1} = t_0 = 0$, we have:

$|\bigcup \mathcal{M}_l| = t_l + 2t_{l-1} + 1 \ (\text{Def.}) \quad \text{for } 0 \leq l \leq k - 1$ \quad (1)
$|\bigcup \mathcal{P}_l| = t_k - t_{l-2} \quad \text{for } 1 \leq l \leq k + 1$ \quad $|\bigcup \mathcal{C}_l| = t_k - t_{k-2} \quad \text{for } 1 \leq l \leq k$ \quad (2,3)

Block diagrams. Figure 2 illustrates the proof for $R = k = 4$. Reader $r_i$ invokes read $rd_i$, $1 \leq l \leq k$. In the column corresponding to some round $rd$ of $op$ we draw a rectangle in a given row, iff round $rd$ of $op$ does not skip the corresponding block $BL$. We write “@” in the row of $BL$ iff $BL$ is malicious.

Read patterns. We first characterize a complete read $rd_l$ for $1 \leq l \leq k - 1$. A complete $rd_l$ skips (1) $\mathcal{M}_{l-2} \cup \mathcal{P}_{l+1}$ in round one and two, and (2) $\mathcal{M}_{l-1} \cup \mathcal{C}_{l+1}$ in round three. Read $rd_k$ skips $\mathcal{M}_{k-2} \cup \mathcal{P}_{k+1}$. Observe that by equations (1), (2) and (3), a read skips exactly $t_k$ objects in each round.

Consider the example in Figure 2. Complete reads $rd_1$, $rd_2$ and $rd_3$ skip (respectively): (1) $\{B_2, 4\}$, $\{B_0\} \cup \{B_3, 5\}$ and $\{B_0\} \cup \{B_1\}$ in rounds one and two, and (2) $\{C_{2,3,4}\}$, $\{B_0\} \cup \{C_{3,4}\}$ and $\{B_{0,1}\} \cup \{C_4\}$ in round three. Read $rd_4$ skips $\{B_{0,1,2}, C_2\} \cup \{B_5\}$.

We further define three types of incomplete reads $inc1$, $inc2$ and $inc3$, depending on the read’s progress. For $1 \leq l \leq k$, read $rd_l$ is of type $inc1$ if the first round is not terminated and skips all blocks except $\mathcal{P}_l$. For $1 \leq l \leq k - 1$, read $rd_l$ is of type (1) $inc2$ if the first round is terminated, and the second round is not terminated and skips all blocks except $\mathcal{C}_l$, and (2) $inc3$ if the second round is terminated and the third round is not terminated and skips $\mathcal{M}_{l-2} \cup \mathcal{C}_{l+1} \cup \mathcal{P}_{l+1}$.

Consider our example in Figure 2 (c) that illustrates partial run $\Delta pr_2$ (after deleting the crossed out steps). Observe that (1) $rd_2$ is incomplete of type $inc3$ (its third round skips $\{B_0\} \cup \{C_{3,4}\} \cup \{B_{3,5}\}$), (2) $rd_3$ is incomplete of type $inc2$ (its second round skips all blocks except $\{C_{2,3,4}\}$) and (3) $rd_3$ (resp., $rd_4$) is incomplete of type $inc1$; its first round skips all blocks except $\{B_{3,5}\}$ (resp., $\{B_4\}$).

Towards a contradiction, we construct a partial run of the atomic register implementation $I$ that violates atomicity. More specifically, we exhibit a partial run in which some read returns a value that was never written.

\footnote{The definition of skipping extends here from Sec. B}
Initialization. Consider a partial run $pr_{init}$ in which (1) all blocks are correct and (2) $pr_{init}$ extends the empty run by appending incomplete reads $rd_l$ by $r_l$ of type $inc_l$, for $1 \leq l \leq k$, one after the other. In $pr_{init}$, there is no write operation. We refer to the state of each correct block $BL \in P_l$ after replying to $rd_l$ as $\sigma^l_0$. Thus, the state of $B_l$ at the end of $pr_{init}$ corresponds to $\sigma^l_0$ for $1 \leq l \leq k$. Further, $B_{k+1}$ is in state $\sigma^{k+1}_0$. To see why, note that $B_{k-1}$ and $B_{k+1}$ have the same parity and there are only $k$ reads.

Consider our example Figure 2(a). At the end of $pr_{init}$, block $B_1$ (resp., $B_2; B_3; B_4$) replied to $rd_1$ (resp., $rd_2; rd_3; rd_2; rd_4$); thus, at the end of the run its state is $\sigma^1_0$ (resp., $\sigma^2_0; \sigma^3_0; \sigma^4_0$).

\[ Fig. 2. \] Instance of the proof with $k = 4. \]
Partial writes. We extend $pr_{init}$ to a partial run $wr^k$ by appending a complete write(1) that completes in $k$ rounds and skips superblock $C_l$. Moreover, we define a series of partial runs each being a prefix of $wr^k$. For $1 \leq i \leq l$, let $wr^{k-i}$ be the partial run which extends $pr_{init}$ by appending an incomplete write(1) such that (i) round 1 to $k-i$ are terminated and (ii) round $k-i+1$ is not terminated and skips $C_l$ and all $B_j$'s such that $j > 0$ and $i$ and $j$ have the same parity, i.e., $C_l \cup P_{2-(i \mod 2)}$. We refer to the state of the blocks $B_l \in P_{2-(i \mod 2)}$ at the end of $wr^{k-i}$ as $\sigma^*_{k-i}$ for $1 \leq l \leq k$. If $B_{k+1} \in P_{2-(i \mod 2)}$, then we refer to its state at the end of $wr^{k-i}$ as $\sigma^*_{k-i}$. Note here that $\sigma^*_{k-i}$ results from $\sigma^*_0$ by appending $k-i$ rounds of the write. When the context is clear, for simplicity we refer to these states using the implicit notation $\sigma^*_{k-i}$. Finally, we refer to the state of $B_0$ at the end of runs $wr^k$ and $wr^{k-1}$ as $\sigma_k$.

We refer to our example in Figure 2(a), (c), (e) and (g) for illustrations of the runs $wr^3$ to $wr^0$ and the corresponding states. For instance Figure 2(a), illustrates $wr^3$ as an extension of $pr_{init}$. The states of the blocks $B_0$, $B_1$ and $B_{3,5}$ at the end of $wr^3$ are $\sigma_4$ (4 rounds of write), $\sigma_3^1$ and $\sigma_3^3$ (3 rounds of write).

Appending Reads. Partial run $pr_1$ extends $wr^{k-1}$ by appending the missing steps of a complete read $rd_1$. In $pr_1$ all objects are correct and thus $rd_1$ receives replies from $S - t_k$ correct objects. After receiving the third round replies, $rd_1$ completes and returns value $x$. We now show that $x = 1$. We define a partial run $pr_1$, (Fig. 2(b)) which is identical to $wr^k$ except that in $pr_0$ (1) no read by $r_1$ occurs and (2) superblock $P_1$ is malicious and mimics the occurrence of $rd_1$ by forging its initial state to $\sigma^*_0$. By equation (1), the malicious objects in $pr_0$ amount to $t_k$. Partial run $pr_1'$ (Fig. 2(b)) is defined as an extension of $pr_0$ by appending a complete read $rd_1$. Read $rd_1$ cannot distinguish $pr_1'$ from $pr_1$ because $P_1$, which is malicious in $pr_1'$, mimics $pr_1$. Specifically, $P_1$ forges its state to $\sigma_0$ before replying to $rd_1$’s first round, and then to $\sigma^*_k$ before replying to $rd_1$’s second round. In $pr_1'$, by atomicity $rd_1$ returns $1$. Since $pr_1'$ and $pr_1$ are indistinguishable to reader $r_1$, $x = 1$.

Next, we define partial run $\Delta pr_1$ obtained from $pr_1'$ by deleting the steps of the read and the write as illustrated in Figure 2(a). More specifically, $\Delta pr_1$ extends $wr^{k-2}$ by appending the missing steps of an incomplete read $rd_1$ of type inc3, after which $rd_1$ crashes. In $\Delta pr_1$, $M_0 = \{B_0\}$ is malicious and forges its state to $\sigma_0$ before replying to $rd_1$. Observe that at the end of $pr_1$ and $\Delta pr_1$, every correct block is in the same state, except $P_2$. We refer to the state of $B_l$ at the end of $\Delta pr_1$ as $\sigma^*_l$.

Starting from $\Delta pr_1$ we iteratively define the following partial runs for $2 \leq l \leq k$ (see Fig. 2). Partial run $pr_l$ extends $\Delta pr_{l-1}$ by appending the missing steps of a complete read $rd_l$. In $pr_l$, superblock $M_{l-2}$ is malicious and all other blocks are correct. Since $rd_l$ does not receive any messages from $M_{l-2}$, it completes only on the basis of replies from correct objects (at least $S - t_k$ by equation (1)). At the end of $pr_l$, $rd_l$ completes and returns value $x$. To show that $x = 1$, we define a partial run $@pr_{l-1}$ which is identical to $pr_{l-1}$ except that in $@pr_{l-1}$ (1) there is no read by $r_l$ and (2) and (in addition to $M_{l-3}$), superblock $P_l$ is malicious and forges its state to $\sigma_0$, simulating the occurrence of $rd_l$ as in $pr_{l-1}$. The count of malicious objects in $@pr_{l-1}$ is exactly $t_l$. To see why, notice that by equation (1) and (2) the malicious objects in $@pr_{l-1}$ amount to $|P_l| + |M_{l-3}| = t_k - t_{l-2} + t_{l-1} = t_k$.

Then, partial run $pr_l'$ extends $@pr_{l-1}$ by appending $rd_l$. Note that $rd_l$ cannot distinguish $pr_l'$ from $pr_l$ because superblock $P_l$, which is malicious in $pr_l'$, mimics $pr_l$. In particular, $P_l$ forges its state to $\sigma_0$ before replying to $rd_l$’s first round and then to $\sigma^*_k$ before replying to $rd_l$’s second round. By atomicity, $rd_l$ returns $1$ in $pr_l'$. Since $pr_l'$ and $pr_l$ are indistinguishable to reader $r_l$, $x = 1$.

Next, we define partial run $\Delta pr_l$. For $2 \leq l < k$, $\Delta pr_l$ is obtained from $pr_l$ by deleting steps of $rd_l$, $rd_{l-1}$ and the write (see Fig. 2(c) and (e)). In $\Delta pr_l$, superblock $M_{l-1}$ is malicious, all other block are correct, and blocks $\{B_{l-1}, C_{l-1}\} \in M_{l-1}$ forge their state to $\sigma^*_l$ before replying to $rd_l$.

In more detail, $\Delta pr_l$ extends

---

The states are different and are indexed by the object's id, which for simplicity of presentation is made implicit.
carefully choosing $c$ forge their state to multi-writer atomic storage can be implemented by applying the standard transformations further \[4, 19\].

can be replaced in the above transformation with the corresponding time-optimal regular implementation.

ular – SWMR atomic transformation technique \[4,19\].

reads and writes, and b) transforming it to the SWMR atomic implementations using a standard SWMR regular – SWMR atomic transformation technique \[4,19\].

time-complexity of write operations is constant.

5 Conclusion

Proof. Without loss of generality we can assume that $t \geq t_k$ because every implementation is subject to the resilience lower bound of $S \geq 3t + 1$. The observation is that if we multiply each of the blocks in the proof of Lemma 1 with a constant $c$, then the result holds for $S' = cS = 3ct_k + c$ objects and $ct_k$ faults. By carefully choosing $c = t/t_k$, we obtain a lower bound proof for $S' = 3t + \lfloor t/t_k \rfloor$ and $t$ faults.

Finally, we generalize our result to a resilience of $3t + \lfloor t/t_k \rfloor$ for $t \geq t_k$, proving Proposition 2.

Proof. Without loss of generality we can assume that $t \geq t_k$ because every implementation is subject to the resilience lower bound of $S \geq 3t + 1$. The observation is that if we multiply each of the blocks in the proof of Lemma 1 with a constant $c$, then the result holds for $S' = cS = 3ct_k + c$ objects and $ct_k$ faults. By carefully choosing $c = t/t_k$, we obtain a lower bound proof for $S' = 3t + \lfloor t/t_k \rfloor$ and $t$ faults.

In this paper, we show that no single-writer multiple-reader (SWMR) robust atomic storage implementation exists if (a) read operations complete in less than four communication round-trips (rounds), and (b) the time-complexity of write operations is constant.

However, we observe that a matching implementation can simply be obtained by a) reusing the SWMR regular storage implementation of \[14\] which features the worst-case time complexity of 2 rounds for both reads and writes, and b) transforming it to the SWMR atomic implementations using a standard SWMR regular – SWMR atomic transformation technique \[4,19\]. This yields a sought SWMR atomic implementation in which write operations complete in 2 rounds whereas reads complete in 4 rounds.

Furthermore, in the stronger authentication model that allows for secret values \[7\], regular storage of \[14\] can be replaced in the above transformation with the corresponding time-optimal regular implementation \[7\], yielding a 2-round write 3-round read atomic storage, which is optimal in this model. In both models, multi-writer atomic storage can be implemented by applying the standard transformations further \[4,19\].

In summary, we present two lower bounds. The first is a read lower bound stating that three rounds of communication are necessary to read from a SWMR robust atomic storage. The second is a write lower bound, showing that $\Omega(\log(t))$ write rounds are necessary to read in three rounds from such a storage. Our results close a fundamental gap: we show that time-optimal, 2-round write 4-round read (resp. 3-round read in the secret value model) robust atomic storage can be obtained using well-known transformations from regular to atomic storage and existing time-optimal regular storage implementations.

\[9\] In short, this transformation employs $R + 1$ regular registers, one dedicated to the writer and $R$ additional ones, one per reader, in which a given reader writes back the read value.
References


A Appendix

A.1 Extension of the Read Lower Bound of Section 3

In this section we prove the result of Section 3 correct in the model of [25], where at most b out of t objects may be malicious and the rest of t − b are simple crash failures. Here we assume b > 0.

**Proposition 3**: If \( S \leq 2t + 2b \) and \( R > 3 \), then no read implementation \( I \) of a multi-reader (SWMR) atomic register exists that completes in two rounds.

**Preliminaries.** In the proof \( w \) denotes the writer, \( r_i \) for \( 1 \leq i \leq 4 \) denote the readers, and \( s_i \) for \( 1 \leq i \leq S \) denote the objects. Suppose by contradiction that \( R = 4 \) and there is

an atomic register implementation \( I \) that uses at most \( 2t + 2b \) objects, such that in every partial run of \( I \) every read operation completes in two communication rounds.

We partition the set objects into six disjoint subsets (which we call blocks), denoted \( B_i \) for \( 1 \leq i \leq 4 \) each of size at most \( b \geq 1 \), and \( C_j \) for \( 1 \leq j \leq 2 \) each of size \( t - b \). Since, \( |B_i \cup C_j| \leq t \) all objects in \( B_i \cup C_j \) may fail together. Throughout the proof, the blocks \( B_i \) fail Byzantine and \( C_j \) fail by crashing.

Without loss of generality we can assume that \( S \geq 2t + 2 \) because every implementation \( I \) must conform to the resilience lower bound of \( S \geq 2t + b + 1 \) [22] (recall that we assume \( b \geq 1 \)). Hence, we can assume that the blocks \( B_i \) contain at least one object. If \( b = t \), then the blocks \( C_1 \) and \( C_2 \) are empty. We refer to the initial state of every correct block \( B_j \) as \( \sigma_0 \). For simplicity we simply write \( \sigma_0 \), where the block name is implicit.

We say that a round \( r_{nd} \) of an operation \( op \) skips a set of blocks \( BS \) in a partial run, (where \( BS \subseteq \{B_1, \ldots, B_4, C_1, C_2\} \)), if (1) no object in any block \( BL \in BS \) receives any message in round \( r_{nd} \) from \( op \) in that partial run, (2) all other objects receive all messages in round \( r_{nd} \) from \( op \) and reply to the messages (3) in case round \( r_{nd} \) is terminated, the invoking client has received all these reply messages or, in case \( r_{nd} \) is not terminated, all these reply messages are in transit. We say that an operation \( op \) skips a set of blocks \( BS \) in a partial run if every round of \( op \) skips \( BS \).

To show a contradiction, we construct a partial run of the implementation \( I \) that violates atomicity: a partial run of \( I \) in which no value is ever written and some read returns 1.

**Partial writes.** The initial value of the register is \( \perp \). Throughout the proof there is only one write operation \( \text{write}(1) \) by \( w \) that writes value 1. Consider a partial run \( wr \) in which \( w \) completes \( \text{write}(1) \) on the register. Let \( k \) be the number of rounds invoked by \( w \) in \( wr \). We denote the state of every correct object \( B_j \) after \( B_j \) has replied to the messages of the write during round 1 to \( i \) where \( 1 \leq i \leq k \), as \( \sigma_i^j \). For simplicity we refer to \( \sigma_i^j \) as \( \sigma_i \). The write operation skips blocks \( \{B_4, C_2\} \). We define a series of partial runs containing an incomplete write(1) invocation, each being a prefix of \( wr \). For \( 1 \leq i \leq k \), we define \( wr_i \) as the partial run in which (1) rounds 1 to \( i \) skip \( \{B_4, C_2\} \), (2) rounds 1 to \( i - 1 \) are terminated and round \( i \) is not terminated, and (3) all objects are correct. For \( 1 \leq i \leq k \) and \( 2 \leq j \leq 4 \), we define \( wr_i^j \) as the partial run in which (1) rounds 1 to \( i - 1 \) are terminated and skip \( \{B_4, C_2\} \), (2) round \( i \) is not terminated and skips blocks \( \{B_1, \ldots, B_4, C_1, C_2\} \setminus \{B_l | j \leq l \leq 3\} \), and (3) all objects are correct. We make two simple observations: (1) partial run \( wr_i^k \) differs from \( wr \) only at \( w \) and (2) partial run \( wr_i^1 \) differs from a run in which \( \text{write}(1) \) is never invoked only at \( w \).
We illustrate the proof in Figure 3 (a)-(k). We depict a round \( \text{rnd} \) through a set of rectangles arranged in a single column. In the column corresponding to some round \( \text{rnd} \) we draw a rectangle in a given row, if all objects in the corresponding block \( BL \) have received the message from the client in round \( \text{rnd} \) of \( op \) and have sent reply messages, i.e., if round \( \text{rnd} \) of \( op \) does not skip \( BL \). We write “@” in the row corresponding to \( BL \) iff \( BL \) is malicious.

**Appending reads.** Partial run \( pr_1 \) extends \( wr \) by appending a complete read \( rd_1 \) by \( r_1 \) that skips \{ \( B_2, C_1 \) \} in round one and \{ \( B_1, C_1 \) \} in round two (see Figure 3 (a)). In \( pr_1 \), all objects in block \( B_1 \) are malicious, and forge their state to \( \sigma_{k-1} \) before replying to \( rd_1 \). By atomicity \( rd_1 \) returns 1. Observe that \( r_1 \) cannot

---

**Fig. 3.** Illustration of the runs used in the proof of Proposition 3
distinguish \( pr_1 \) from some partial \( \Delta pr_1 \) that extends \( wr^k_2 \) by appending \( rd_1 \), and where all objects are correct.

Partial run \( pr_2 \) extends \( \Delta pr_1 \) by appending a complete read \( rd_2 \) by \( r_2 \) that skips \( \{B_3, C_2\} \) in round one and \( \{B_2, C_1\} \) in round two (see Figure 3(b)). In \( pr_2 \), all objects in block \( B_2 \) are malicious, and forge their state to \( \sigma_{k-1} \) before replying to \( rd_2 \). By atomicity \( rd_2 \) returns 1. Observe that \( r_2 \) cannot distinguish \( pr_2 \) from some partial run \( \Delta pr_2 \), that extends \( wr^k_3 \) by appending an incomplete \( rd_1 \) and a complete \( rd_2 \) and where all objects are correct (as illustrated in Figure 3(b) after deleting the crossed steps).

Partial run \( pr_3 \) extends \( \Delta pr_2 \) by appending a complete read \( rd_3 \) by \( r_3 \) that skips \( \{B_3, C_2\} \) in round one and \( \{B_3, C_2\} \) in round two. In \( pr_3 \), all objects in block \( B_3 \) are malicious, and forge their state to \( \sigma_{k-1} \) before replying to \( rd_3 \). By atomicity \( rd_3 \) returns 1. Let \( \sigma^*_r \) denote the state of the objects in block \( B_4 \) in run \( pr_3 \) before replying to \( rd_2 \). Observe that \( r_3 \) cannot distinguish \( pr_3 \) from some partial run \( \Delta pr_3 \), that extends \( wr^k_4 \) by appending incomplete reads \( rd_1 \) and \( rd_2 \) and a complete read \( rd_3 \) and in which (1) all objects in \( B_4 \) are malicious and (2) they forge their state to \( \sigma^*_r \) before replying to \( rd_2 \) (as shown in Figure 3(c) after deleting the crossed steps). Note that since \( r_3 \) cannot distinguish \( pr_3 \) and \( \Delta pr_3 \), it cannot wait for more replies in order to complete. More precisely, \( rd_3 \) completes in \( \Delta pr_3 \) (resp. \( pr_3 \)) the first (resp. second) round based on replies only from correct objects (at least \( S - t \)).

Partial run \( pr_4 \) (illustrated in Figure 3(a)) extends \( \Delta pr_3 \) by appending a complete read \( rd_4 \) by \( r_4 \) that skips \( \{B_1, C_1\} \) in round one and \( \{B_4, C_2\} \) in round two. In \( pr_4 \), all objects in block \( B_4 \) are malicious and forge their state (1) to \( \sigma^*_r \) before replying to \( rd_2 \) and (2) to \( \sigma_0 \) before replying to \( rd_4 \). By atomicity \( rd_4 \) returns 1. Let \( \sigma^*_j \) denote the state of the objects in block \( B_1 \) before replying to \( rd_3 \). Observe that \( r_4 \) cannot distinguish \( pr_4 \) from some partial run \( \Delta pr_4 \), that extends \( wr^k_{r-1} \) by appending incomplete reads \( rd_2 \), \( rd_3 \) and a complete read \( rd_4 \), and in which (1) all objects in \( B_4 \) are malicious and (2) they forge their state to \( \sigma^*_j \) before replying to \( rd_3 \) (see Figure 3(e)). Since \( r_4 \) cannot distinguish \( pr_4 \) and \( \Delta pr_4 \), \( rd_4 \) cannot wait for additional replies in order to complete, without violating termination. To see why, notice that in run \( \Delta pr_4 \) (resp \( pr_4 \), \( rd_4 \) receives first (resp. second) round replies from \( S - t \) correct objects.

After appending \( rd_4 \) and constructing \( \Delta pr_4 \) by deleting all steps from \( pr_4 \) which are not “visible” to \( rd_4 \), we notice that we have erased all steps in column \( k \) of \( write(1) \) as well as, deleted all steps of \( rd_1 \). Thus, we can recyle \( r_1 \) by appending \( rd_1 \) again and start deleting the steps in column \( k-1 \).

Starting from \( \Delta pr_4 \) we iteratively define the following partial runs for \( 1 \leq i \leq k-1 \) and \( 1 \leq j \leq 4 \) (depicted in Figure 3(d)-(k)). Partial run \( pr_{4i+j} \) extends \( \Delta pr_{4i+j-1} \) by appending \( rd_j \). In \( pr_{4i+j} \), all objects in block \( B_j \) are malicious and they forge their state (1) to \( \sigma^*_{4i+(j-3)} \) before replying to \( rd_{j-2} \) and (2) to \( \sigma_{(j \mod 4) + (j-1)} \) before replying to \( rd_j \). Let \( \sigma^*_{4i+(j-2)} \) denote the state of the objects in block \( B_{(j \mod 4) + 1} \) before replying to \( rd_{j-1} \). Observe that \( r_j \) cannot distinguish \( pr_{4i+j} \) from some partial run \( \Delta pr_{4i+j} \), that extends \( wr_{(j \mod 4) + 1} \) by appending incomplete reads \( rd_{j-2} \) and \( rd_{j-1} \) and a complete read \( rd_j \), and in which (1) all objects in \( B_{(j \mod 4) + 1} \) are malicious, and (2) they forge their state to \( \sigma^*_{4i+(j-2)} \) before replying to \( rd_{j-1} \) (see Figure 3(e),(g),(i),(k)). Since \( r_j \) cannot distinguish \( \Delta pr_{4i+j} \) and \( pr_{4i+j} \), \( rd_j \) cannot wait for additional replies, because in run \( \Delta pr_{4i+j} \) and \( pr_{4i+j} \), \( rd_{j} \) receives first and second round replies respectively from \( S - t \) correct objects.

Read \( rd_4 \) in \( \Delta pr_4 \) returns 1. Since \( pr_5 \) extends \( \Delta pr_4 \) by appending \( rd_1 \), by atomicity, \( rd_3 \) in \( pr_5 \) returns 1. However, as \( r_1 \) cannot distinguish \( pr_5 \) from \( \Delta pr_5 \), \( rd_1 \) in \( pr_5 \) returns 1. In general, since \( pr_{4i+j} \) extends \( \Delta pr_{4i+j-1} \) by appending \( rd_j \) (for \( 1 \leq i \leq k-1 \) and \( 1 \leq j \leq 4 \)), and \( r_j \) cannot distinguish \( pr_{4i+j} \) from \( \Delta pr_{4i+j} \), it follows by induction that \( rd_j \) in \( \Delta pr_{4i+j} \) returns 1. In particular, \( rd_3 \) reads 1 in \( \Delta pr_{4k-1} \). By our construction, \( \Delta pr_{4k-1} \) extends \( wr^1_3 \) and \( wr^1_4 \) is indistinguishable from a run in which \( write(1) \) is never invoked. Hence, \( rd_3 \) returns 1 even if no write is invoked, violating atomicity. \( \square \)

\(^{10}\) Please note that when we write \( rd_{j-c} \), we always mean \( rd_{4i+(c-j) \mod 4} \).

14