Integrated Bounds for Disintegrated Storage

Alon Berger
Viterbi Department of Electrical Engineering, Technion, Haifa, Israel

Idit Keidar
Viterbi Department of Electrical Engineering, Technion, Haifa, Israel

Alexander Spiegelman
VMware Research, Israel

Abstract
We point out a somewhat surprising similarity between non-authenticated Byzantine storage, coded storage, and certain emulations of shared registers from smaller ones. A common characteristic in all of these is the inability of reads to safely return a value obtained in a single atomic access to shared storage. We collectively refer to such systems as disintegrated storage, and show integrated space lower bounds for asynchronous regular wait-free emulations in all of them. In a nutshell, if readers are invisible, then the storage cost of such systems is inherently exponential in the size of written values; otherwise, it is at least linear in the number of readers. Our bounds are asymptotically tight to known algorithms, and thus justify their high costs.

2012 ACM Subject Classification Theory of computation → Distributed computing models, Computing methodologies → Distributed algorithms

Keywords and phrases storage, coding, lower bounds, space complexity, register emulations

Digital Object Identifier 10.4230/LIPIcs.DISC.2018.11

Acknowledgements We thank Yuval Cassuto, Gregory Chockler, Rati Gelashvili, and Yuanhao Wei for many insightful discussions on space bounds for coded storage and emulations of large registers from smaller ones.

1 Introduction

1.1 Space bounds for encoded, multi-register, and Byzantine storage
In many data sharing solutions, information needs to be read from multiple sources in order for a single value to be reconstructed. One such example is coded storage where multiple storage blocks need to be obtained in order to recover a single value that can be returned to the application [5, 9, 10, 16–18, 22, 23]. Another example arises in shared memory systems, where the granularity of atomic memory operations (such as load and store) is limited to a single word (e.g., 64 bits) and one wishes to atomically read and write larger values [22]. A third example is replicating data to overcome Byzantine faults (without authentication) or data corruption, where a reader expects to obtain the same block from multiple servers in order to validate it [1, 2, 19].

We refer to such systems collectively as disintegrated storage systems. We show that such a need to read data in multiple storage accesses inherently entails high storage costs: exponential in the data size if reads do not modify the storage, and otherwise linear in the number of concurrent reads. This stands in contrast to systems that use non-Byzantine replication, such as ABD [6], where, although meta-data (e.g., timestamps) is read from several sources, the recovered value need only be read from a single source.
11:2 Integrated Bounds for Disintegrated Storage

1.2 Our results

We consider a standard shared storage model (see Section 2). We refer to shared storage locations (representing memory words, disks, servers, etc.) as objects. To strengthen our lower bounds, we assume that objects are responsive, i.e., do not fail; the results hold a fortiori if objects can also be unresponsive [19]. Objects support general read-modify-write operations by asynchronous processes. We study wait-free emulations of a shared regular register [20].

Section 3 formally defines disintegrated storage. We use a notion of blocks, which are parts of a value kept in storage – code blocks, segments of a longer-than-word value, or full copies of a replicated value. A key assumption we make is that each block in the shared storage pertains to a single write operation; a similar assumption was made in previous studies [11,23]. The disintegration property then stipulates that a reader must obtain some number $\tau > 1$ of blocks pertaining to a value $v$ before returning $v$. For example, $\tau$ blocks are needed in $\tau$-out-of-$n$ coded storage, whereas $\tau = f + 1$ in $f$-tolerant Byzantine replication. To strengthen our results, we allow the storage to hold unbounded meta-data (e.g., timestamps), and count only the storage cost for blocks. Note that the need to obtain $\tau$ blocks implies that meta-data cannot be used instead of actual data.

In Section 4 we give general lower bounds that apply to all types of disintegrated storage – replicated, coded, and multi-register. We first consider invisible reads, which do not modify the shared storage. This is a common paradigm in storage systems and often essential where readers outnumber writers and have different permissions. In this case, even with one reader and one writer, the storage size can be exponential; specifically, if value sizes are $D$ (taken from a domain of size $2^D$), then we show a lower bound of $\tau + (\tau - 1)\left\lceil \frac{2^D - 1}{L} \right\rceil$ blocks, where $L$ is the number of blocks in a reader’s local storage. That is, either the local storage of the reader or the shared storage is exponential.

Section 5 studies a more restrictive flavor of disintegrated storage, called $\tau$-common write, where a reader needs to obtain $\tau$ blocks produced by the same write operation in order to return $v$. In other words, if the reader obtains blocks that originate from two different writes of the same value, then it cannot recognize that they pertain to the same value, as is the case when blocks hold parts of a value or code blocks rather than replicas. In this case, the shared storage cost is high independently of the local memory size. Specifically, we show a bound of $\tau \cdot 2^D$ blocks with invisible readers. In systems that use symmetric coding (i.e., where all blocks are of the same size, namely at least $D/\tau$ bits), this implies a lower bound of $D \cdot 2^D$ bits. For a modest value size of 20 bytes, the bound amounts to $2.66 \cdot 10^{37}$ TB, and for 1KB values it is a whopping $1.02 \cdot 10^{2457}$ TB.

We further consider visible reads, which can modify the objects’ meta-data. Such readers may indicate to the writers that a read is ongoing, and signal to them which blocks to retain. Using such signals, the exponential bound no longer holds – there are emulations that store a constant number of values per reader [2,5,13,22]. We show that such linear growth with the number of readers is inherent. Our results are summarized in Table 1.

These bounds are tight as far as regularity and wait-freedom go: relaxing either requirement allows circumventing our results [1,19]. As for storage cost, our lower bounds are asymptotically tight to known algorithms, whether reads are visible [2,5,22] or not [7,16,18,21].

We note that the study of the inherent storage blowup in asynchronous coded systems has only recently begun [11,23] and is still in its infancy. In this paper, we point out a somewhat surprising similarity between coded storage and other types of shared memory/storage, and show unified lower bounds for all of them. Section 6 concludes the paper and suggests directions for future work.
Table 1 Lower bounds on shared storage space consumption, in units of blocks; $D$ is the value size, $\tau > 1$ is the number of data blocks required in order to recover a value, $L \geq 1$ is the maximal number of blocks stored in a reader’s local data, and $R$ the number of readers.

<table>
<thead>
<tr>
<th></th>
<th>Invisible Reads</th>
<th>Visible Reads</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>General Case</strong></td>
<td>$\tau + (\tau - 1) \left\lceil \frac{2^D - 1}{L} \right\rceil$</td>
<td>$\tau + (\tau - 1) \cdot \min \left( \left\lceil \frac{2^D - 1}{L} \right\rceil, R \right)$</td>
</tr>
<tr>
<td><strong>Common Write</strong></td>
<td>$\tau \cdot 2^D$</td>
<td>$\tau + (\tau - 1) \cdot \min (2^D - 1, R)$</td>
</tr>
<tr>
<td>(e.g., coded storage)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1.3 Related work and applicability of our bounds

Several works have studied the space complexity of register emulations. Two recent works [11, 23] show a dependence between storage cost and the number of writers in crash-tolerant storage, identifying a trade-off between the cost of replication ($f + 1$ copies for tolerating $f$ faults) and that of $\tau$-out-of-$n$ coding (linear in the number of writers). Though they do not explicitly consider disintegrated storage, it is fairly straightforward to adapt the proof from [23] to derive a lower bound of $\tau W$ blocks with $W$ writers. Here we consider the case of single-writer algorithms, where this bound is trivial. Other papers [3, 15] show limitations of multi-writer emulations when objects do not support atomic read-modify-write, whereas we consider single-writer emulations that do use read-modify-write.

Chockler et al. [14] define the notion of amnesia for register emulations with an infinite value domain, which intuitively captures the fact that an algorithm “forgets” all but a finite number of values written to it. They show that a wait-free regular emulation tolerating non-authenticated Byzantine faults with invisible readers cannot be amnesic, but do not show concrete space lower bounds. In this paper we consider a family of disintegrated storage algorithms, with visible and invisible readers, and show concrete bounds for the different cases; if the size of the value domain is unbounded, then our invisible reader bounds imply unbounded shared storage.

Disintegrated storage may also correspond to emulations of large registers from smaller ones, where $\tau$ is the size of the big register divided by the size of the smaller one. Some algorithms in this vein, e.g., [22], indeed have the disintegration property, as the writer writes $\tau$ blocks to a buffer and a reader obtains $\tau$ blocks of the same write. These algorithms are naturally subject to our bounds. Other algorithms, e.g., [12, 13, 20], do not satisfy our assumption that each block in the shared storage pertains to a single write operation, and a reader may return a value based on blocks written by different write operations. Thus, our bounds do not apply to them. It is worth noting that these algorithms nevertheless either have readers signal to the writers and use space linear in the number of readers, or have invisible readers but use space exponential in the value size. Following an earlier publication of our work in [8], Wei [25] showed that these costs – either linear in the number of visible readers or exponential in the value size with invisible ones – are also inherent in emulations of large registers from smaller ones that do share blocks among writes, albeit do not use meta-data at all. Several questions remain open in this context: first, Wei’s bound is not applicable to all types of storage we consider (in particular, Byzantine), and does not apply to algorithms that use timestamps. Second, we are not familiar with any regular register emulations where readers write-back data, and it is unclear whether our bound may be circumvented this way.

Non-authenticated Byzantine storage algorithms that tolerate $f$ faults need to read a value $f + 1$ times in order to return it, and are thus $\tau$-disintegrated for $\tau = f + 1$. Note that while our model assumes objects are responsive, it a fortiori applies to scenarios where objects
may be unresponsive. Some algorithms circumvent our bound either by providing only safe semantics [19], or by forgoing wait-freedom [1]. Others use channels with unbounded capacity to push data to clients [7, 21] or potentially unbounded storage with best-effort garbage collection [18].

As for coded storage, whenever \( \tau \) blocks are required to reconstruct a value, the algorithm is \( \tau \)-disintegrated. And indeed, previous solutions in our model require unbounded storage or channels [9, 10, 16–18], or retain blocks for concurrent visible readers, consuming space linear in the number of readers [5]. Our bounds justify these costs. Our assumption that each block in the shared storage pertains to a single value is satisfied by almost all coded storage algorithms we are aware of, the only exception is [24], which indeed circumvents our lower bound but does not conform to regular register semantics. Other coded storage solutions, e.g., [4], are not subject to our bound because they may recover a value from a single block.

2 Preliminaries

Shared storage model

We consider an asynchronous shared memory system consisting of two types of entities: A finite set \( O = \{o_1, \ldots, o_n\} \) of objects comprising shared storage, and a set \( \Pi \) of processes. Every entity in the system stores data: an object’s data is a single block from some domain \( B \), whereas a process’ data is an array of up to \( L \) blocks from \( B \). We assume a bound \( L \) on the number of blocks in the data array of each process. In addition, each entity stores potentially infinite meta-data, \( meta \). We denote an entity \( e \)'s data as \( e.data \) and likewise for \( e.meta \). A system’s storage cost is the number of objects in the shared storage, \( n \).

Objects support atomic get and update actions by processes. We denote by \( a_p \) an action \( a \) performed by \( p \) and by \( o.a_p \) an \( a_p \) action at \( o \). An \( o.update_p \) is an arbitrary read-modify-write that possibly writes a block from \( B \) to \( o.data \) and modifies \( o.meta, p.meta \), and \( p.data \). An \( o.get_p \) may replace a block in \( p.data \) with \( o.data \) and may modify \( p.meta \).

Algorithms, configurations, and runs

An algorithm defines the behaviors of processes as deterministic state machines, where state transitions are associated with actions. A configuration is a mapping to states (data and meta) from all system components, i.e., processes and objects. In an initial configuration all components are in their initial states.

We study algorithms (executed by processes in \( \Pi \)) that emulate a high-level functionality, exposing high-level operations, and performing low-level gets/updates on objects. We say that high-level operations are invoked and return or respond. Note that, for simplicity, we model gets and updates as instantaneous actions, because the objects are assumed to be atomic, and we do not explicitly deal with object failures in this paper.

A run of algorithm \( A \) is a (finite or infinite) alternating sequence of configurations and actions, beginning with some initial configuration, such that configuration transitions occur according to \( A \). Occurrences of actions in a run are called events. The possible events are high-level operation invocations and responses and get/update occurrences. We use the notion of time \( t \) during a run \( r \) to refer to the configuration reached after the \( i^{th} \) event in \( r \). For a finite run \( r \) consisting of \( t \) events we define \( t_r = t \). Two operations are concurrent in a run \( r \) if both are invoked in \( r \) before either returns. If a process \( p \)'s state transition from state \( S \) is associated with a low-level action \( a_p \in \{get_p, update_p\} \), we say that \( a_p \) is enabled in \( S \). A run \( r' \) is an extension of a (finite) run \( r \) if \( r \) is a prefix of \( r' \); we denote by \( r' \setminus r \) the
suffix of \( r' \) that starts at \( t_r \). If a high-level operation \( op \) has been invoked by process \( p \) but has not returned by time \( t \) in a run \( r \), we say that \( op \)'s invocation is pending at time \( t \) in \( r \). We assume that each process' first action in a run is an invocation, and a process has at most one pending invocation at any time.

For \( e \in \Pi \cup O \), we denote by \( e.data(r, t) \) the set of distinct blocks stored in \( e.data \) at time \( t \) in a run \( r \). Since for an object \( o \), \(|o.data(r, t)| = 1\), we sometimes refer to \( o.data(r, t) \) as the block itself, by slight abuse of notation. We say that \( p \) obtains a block \( b \) at time \( t \) in a run \( r \), if \( b \notin p.data(r, t) \) and \( b \in p.data(r, t + 1) \).

### Register emulations

We study algorithms that emulate a shared register [20], which stores a value \( v \) from some domain \( \mathbb{V} \). We assume that \(|\mathbb{V}| = 2^D > 1\), i.e., values can be represented using \( D > 0 \) bits. For simplicity, we assume that each run begins with a dummy initialization operation that writes the register's initial value and does not overlap any operation. The register exposes high-level \( \text{read}_p \) and \( \text{write}_p(v) \) operations of values \( v \in \mathbb{V} \) to processes \( p \in \Pi \). We consider single-writer (SW) registers where the application at only one process (the writer) invokes \( \text{writes} \), and hence omit the subscript \( p \) from \( \text{write}(v) \). The remaining \( R = |\Pi| - 1 \) processes are limited to performing \( \text{reads} \), and are referred to as readers. For brevity, we refer to the subsequence of a run where a specific invocation of a \( \text{write}(v) / \text{read}_p \) is pending simply as a \( \text{write}(v) / \text{read}_p \) operation.

We assume that whenever a \( \text{read}_p \) operation is invoked at time \( t \) in a run \( r \), \( p.data(r, t) \) is empty. We consider two scenarios: (1) invisible reads, where \( \text{reads} \) do not use updates, and (2) visible reads, where \( \text{reads} \) may perform updates that update meta-data (only) in the shared storage. Note that readers do not write actual data, which is usually the case in regular register emulations, defined below. In a single-reader (SR) register \( R = 1 \), and if \( R > 1 \) the register is multi-reader (MR). If the states of the writer and the objects at the end of a finite run \( r \) are equal to their respective states at the end of a finite run \( r' \), we say that \( t_r \) and \( t_{r'} \) are indistinguishable to the writer and objects, and denote: \( t_r \approx_w t_{r'} \).

Our safety requirement is regularity [20]: a \( \text{read} rd \) must return the value of either the last \( \text{write} \) \( w \) that returns before \( rd \) is invoked, or some write that is concurrent with \( rd \). For liveness, we require wait-freedom, namely that every operation invoked by a process \( p \) returns within a finite number of \( p \)'s actions. In other words, if \( p \) is given infinitely many opportunities to perform actions, it completes its operation regardless of the actions of other processes.

### 3 Disintegrated storage

As noted above, existing wait-free algorithms of coded and/or Byzantine-fault-tolerant storage with invisible readers may store all values ever written [7,9,11,16–18,21]. This is because if old values are erased, it is possible for a slow reader to never find sufficiently many blocks of the same value so as to be able to return it. If readers are visible, then a value per reader is retained. We want to prove that these costs are inherent. The challenge in proving such space lower bounds is that the aforementioned algorithms use unbounded timestamps. How can we show a space lower bound if we want to allow algorithms to use unbounded timestamps? We address this by allowing meta-data to store timestamps, etc., and by not counting the storage cost for meta-data. For example, the above algorithms store timestamps in meta-data alongside data blocks and use them to figure out which data is safe to return, but still need \( \tau \) actual blocks/copies of a value in order to return it. Note that for the sake of the lower
bound, we do not restrict how meta-data is used; all we require is that the algorithm read \( \tau \) data blocks of the same value (or \textit{write}), and we do not specify how the algorithm knows that they pertain to the same value (or \textit{write}). To formalize the property that the algorithm returns \( \tau \) blocks pertaining to the same value or \textit{write}, we need to track, for each block in the shared storage, which \textit{write} produced it. To this end, we define labels. Labels are only an analysis tool, and do not exist anywhere. In particular, they are not timestamps, not meta-data, and not explicitly known to the algorithm. As an external observer, we may add them as abstract state to the blocks, and track how they change.

**Labels**

We associate each block \( b \) in the shared or local storage with a set of labels, \( Labels(b) \), as we now explain. For an algorithm \( A \) and \( v \in V \), denote by \( \mathcal{W}_v^A \) the set of \textit{write}(\( v \)) operations invoked in runs of \( A \). For \( V \subseteq \mathcal{V} \), we denote \( \mathcal{W}_v^A = \bigcup_{v \in V} \mathcal{W}_v^A \), and let \( \mathcal{W}^A = \mathcal{W}_V \). For clarity, we omit \( A \) when obvious from the context, and refer simply to \( \mathcal{W}_v \), \( \mathcal{W}_V \), and \( \mathcal{W} \). We assume that the \( k \textsuperscript{th} \) \textit{update} event occurring in a \textit{write} operation \( w \in \mathcal{W} \) tags the block \( b \) it stores (if any) with a unique label \( \langle w, k \rangle \), so \( Labels(b) \) becomes \( \{\langle w, k \rangle\} \).

Whereas our assumption that each block in the shared storage pertains to a single \textit{write} rules out associating multiple labels with such a block, we do allow the reader’s meta-data to recall multiple accesses encountering the same block. For example, when blocks are copies of a replicated value, the reader can store one instance of the value in local memory and keep a list of the objects where the value was encountered. To this end, a block in a reader’s \textit{data} may be tagged with multiple labels: when a reader \( p \) obtains a block \( b \) from an object \( \omega \) at time \( t \) in a run \( r \), the block \( b \) in \textit{data}(\( \omega \), \( r \), \( t \)) is tagged with \( Labels(\omega, \textit{data}(\( \omega \), \( r \), \( t \))) \); if at time \( t' > t \) \textit{data} still contains \( b \) and \( p \) performs an action on an object \( \omega' \) s.t. \( \omega'.\textit{data}(\( \omega' \), \( r' \)) = b \) and the latter is tagged with label \( \ell \), \( p \) adds \( \ell \) to \( Labels(b) \) (regardless of whether \( b \) is added to \textit{data} once more). When all copies of a block are removed from \textit{data}, all its labels are “forgotten”. We emphasize that labels are not stored anywhere, and are only used for analysis.

We track the labels of a value \( v \in \mathcal{V} \) at time \( t \) in a run \( r \) using the sets \( S\textsuperscript{-}labels(\( v \), \( r \), \( t \)), \textit{L-labels}(\( v \), \( r \), \( t \)), of labels in process \( p \)’s local storage, and \( All\textsuperscript{-}labels(\( v \), \( r \), \( t \)), \textit{com} \) of both. Formally,

- \( S\textsuperscript{-}labels(\( v \), \( r \), \( t \)) \triangleq \bigcup_{\omega \in O} Labels(\omega, \textit{data}(\( \omega \), \( r \), \( t \)))) \cap (\mathcal{W}_v \times \mathbb{N}) \).
- \( L\textsuperscript{-}labels(\( v \), \( r \), \( t \)) \triangleq \bigcup_{b \in \textit{data}(\( \omega \), \( r \), \( t \))} Labels(b) \cap (\mathcal{W}_v \times \mathbb{N}) \).
- \( All\textsuperscript{-}labels(\( v \), \( r \), \( t \)) \triangleq L\textsuperscript{-}labels(\( v \), \( r \), \( t \)) \cup S\textsuperscript{-}labels(\( v \), \( r \), \( t \)).

For a time \( t \) in a run \( r \) and \( p \in \Pi \), we define \textit{values}(\( p \), \( r \), \( t \)) \triangleq \{v \in \mathcal{V} \mid L\textsuperscript{-}labels(\( v \), \( r \), \( t \)) \neq \emptyset \}.

Similarly, we track labels associated with a particular \textit{write} \( w \in \mathcal{W} \) accessible by process \( p \in \Pi \) at time \( t \) in a run \( r 

- \( S\textsuperscript{-}labels(\( w \), \( r \), \( t \)) \triangleq \bigcup_{\omega \in O} Labels(\omega, \textit{data}(\( \omega \), \( r \), \( t \)))) \cap (\{w\} \times \mathbb{N}) \).
- \( L\textsuperscript{-}labels(\( w \), \( r \), \( t \)) \triangleq \bigcup_{b \in \textit{data}(\( \omega \), \( r \), \( t \))} Labels(b) \cap (\{w\} \times \mathbb{N}) \).
- \( All\textsuperscript{-}labels(\( w \), \( r \), \( t \)) \triangleq L\textsuperscript{-}labels(\( w \), \( r \), \( t \)) \cup S\textsuperscript{-}labels(\( w \), \( r \), \( t \)).

We define \textit{writes}(\( p \), \( r \), \( t \)) \triangleq \{w \in \mathcal{W} \mid L\textsuperscript{-}labels(\( w \), \( r \), \( t \)) \neq \emptyset \}. Note that for all \( v \in \mathcal{V} \) and \( w \in \mathcal{W}_v \), (1) \( S\textsuperscript{-}labels(\( w \), \( r \), \( t \)) \subseteq S\textsuperscript{-}labels(\( v \), \( r \), \( t \)), (2) \( L\textsuperscript{-}labels(\( w \), \( r \), \( t \)) \subseteq L\textsuperscript{-}labels(\( v \), \( r \), \( t \)), and (3) \( All\textsuperscript{-}labels(\( v \), \( r \), \( t \)) \subseteq All\textsuperscript{-}labels(\( v \), \( r \), \( t \)).

Since readers do not write-back:

\[\textbf{Observation 1.} \text{If the} \ t \text{th event in a run} \ r \text{is of a reader} \ p \in \Pi \text{, then for all} \ v \in \mathcal{V}, w \in \mathcal{W}: \text{All}\textsuperscript{-}labels(\( v \), \( r \), \( t \)) \subseteq \text{All}\textsuperscript{-}labels(\( v \), \( r \), \( t - 1 \)) \text{and All}\textsuperscript{-}labels(\( w \), \( r \), \( t \)) \subseteq All\textsuperscript{-}labels(\( w \), \( r \), \( t - 1 \)).\]
Disintegrated storage

Intuitively, in disintegrated storage register emulations, for a \texttt{read}_p to return \( v \), \( p \) must encounter \( \tau > 1 \) blocks corresponding to \( v \) that were produced by separate \texttt{update} events. To formalize this, we use labels:

\begin{definition}[\( \tau \)-disintegrated storage]
If a return of \( v \in \mathbb{V} \) by a \texttt{read}_p invocation is enabled at time \( t \) in a run \( r \) then \( |L\text{-labels}_p(v,r,t)| \geq \tau \).
\end{definition}

Thus, a reader can only return \( v \) if it recalls (in its local memory) obtaining blocks of \( v \) with \( \tau \) different labels.

A more restrictive case of \( \tau \)-disintegrated storage occurs when readers cannot identify whether two blocks pertain to a common value unless they are produced by a common write that identifies them, e.g., with the same timestamp. This is the case when value parts or code words are stored in objects rather than full replicas.

To capture this case, for a block \( b \in \bigcup_{e \in \mathbb{O}} e.data \), a value \( v \in \mathbb{V} \), and a \texttt{write} \( w \in \mathbb{W}_v \), if \( \exists k \in \mathbb{N} \text{ s.t. } \langle w, k \rangle \in \text{Labels}(b) \), we say that \( w \) is an origin \texttt{write} of \( b \) and \( v \) is an origin value of \( b \). Common write \( \tau \)-disintegrated storage is then defined:

\begin{definition}[common write \( \tau \)-disintegrated storage]
If a return of \( v \in \mathbb{V} \) by a \texttt{read}_p invocation is enabled at time \( t \) in a run \( r \) then \( \exists w \in \mathbb{W}_v : |L\text{-labels}_p(w,r,t)| \geq \tau \).
\end{definition}

Note that we do not further require \( p.data \) to actually hold \( \tau \) blocks with a common write, because the weaker definition suffices for our lower bounds. For brevity, we henceforth refer to a common write \( \tau \)-disintegrated storage algorithm simply as \( \tau \)-common write.

Permanence

Our lower bounds will all stem, in one way or another, from the observation that in wait-free disintegrated storage, every run must reach a point after which some values (and in the case of common write, also some \texttt{writes}) must permanently have a certain number of blocks in the shared storage. This is captured by the following definition:

\begin{definition}[permanence]
Consider a finite run \( r \), \( k \in \mathbb{N} \), a set \( S \subseteq \mathbb{V} \), and a set of readers \( \Theta \subseteq \Pi \). Let \( z \in \mathbb{V} \cup \mathbb{W} \) be a value or a \texttt{write} operation. We say that \( z \) is \( \langle k, \Theta, S \rangle \)-permanent in \( r \) if in every finite extension \( r' \) of \( r \) s.t. \( r' \setminus r \) readers in \( \Theta \) do not take actions and \texttt{writes} are limited to values from \( S, |S\text{-labels}(z,r',t_r)| \geq k \).
\end{definition}

Intuitively, this means that the shared storage continues to hold \( k \) blocks of \( z \) as long as readers in \( \Theta \) do not signal to the writer and only values from \( S \) are written. For brevity, when the particular sets \( S \) and \( \Theta \) are not important, we refer to the value shortly as \( k \)-permanent.

The observation below follows immediately from the definition of permanence:

\begin{observation}
Let \( v \in \mathbb{V} \), \( w \in \mathbb{W}_v \), \( k \in \mathbb{N} \), \( V_2 \subseteq V_1 \subseteq \mathbb{V} \), \( \Theta_1 \subseteq \Theta_2 \subseteq \Pi \).
\begin{enumerate}
\item If \( w \) is \( \langle k, \Theta_1, V_1 \rangle \)-permanent in a finite run \( r \) then \( v \) is \( \langle k, \Theta_1, V_1 \rangle \)-permanent in \( r \).
\item If \( v \) is \( \langle k, \Theta_1, V_1 \rangle \)-permanent in a finite run \( r \) then \( v \) is \( \langle k, \Theta_2, V_2 \rangle \)-permanent in all finite extensions \( r' \) of \( r \) where in \( r' \setminus r \) \texttt{writes} are limited to values from \( V_1 \) and readers in \( \Theta_1 \) do not take actions.
\end{enumerate}
\end{observation}

Since each object holds a single block associated with a single label:

\begin{observation}
For time \( t \) in a run \( r \), the number of objects is: \( n \geq |\bigcup_{v \in \mathbb{V}} S\text{-labels}(v,r,t)| \).
\end{observation}

Thus, if there are \( m \) different \( k \)-permanent values in a run, then \( n \geq mk \). We observe that with invisible readers, the set \( \Theta \) is immaterial:
Observation 7. Consider $k \in \mathbb{N}$, $V \subseteq \mathbb{V}$, and a finite run $r$ with an invisible reader $p \in \Pi$. If $z \in \mathbb{V} \cup \mathbb{W}$ is $(k \cdot \{p\}, V)$-permanent in $r$ then $z \in \langle k, \emptyset, V \rangle$-permanent in $r$.

The specific lower bounds for the four scenarios we consider differ in the number of permanent values/writes and the number of blocks per value/write ($k = \tau - 1$ or $k = \tau$) we can force the shared storage to retain forever in each case. Interestingly, our notion of permanence resembles the idea that an algorithm is not amnesic introduced in [14] (see Section 1.3), but is more fine-grained in specifying the number of permanent blocks and restricting executions under which they are retained.

4 Lower bounds for disintegrated storage

In this section we provide lower bounds on the number of objects required for $\tau$-disintegrated storage regular wait-free register emulations. Section 4.1 proves two general properties of regular wait-free $\tau$-disintegrated storage algorithms. We show in Section 4.2 that with invisible reads, unless the readers’ local storage size is exponential in $D$, the storage cost of such emulations is at least exponential in $D$. Finally, Section 4.3 shows that if reads are visible, then the storage cost increases linearly with the number of readers.

4.1 General properties

We first show that because readers must make progress even if the writer stops taking steps, at least $2\tau - 1$ blocks are required regardless of the number of readers.

Claim 8. Consider $v_1, v_2 \in \mathbb{V}$ and a run $r$ of a wait-free regular $\tau$-disintegrated storage algorithm with two consecutive responded writes $w_1 \in \mathbb{W}_{v_1}$ followed by $w_2 \in \mathbb{W}_{v_2}$. Let $p \in \Pi$ be a reader s.t. no read$_p$ is pending in $r$. Then there is a time $t$ between the returns of $w_1$ and $w_2$ when $|S$-labels$(v_1, r, t)| \geq \tau$ and $|S$-labels$(v_2, r, t)| \geq \tau - 1$.

Proof. We first argue that at the time $t_i$, $i \in \{1, 2\}$ when $w_i$ returns, $|S$-labels$(v_i, r, t_i)| \geq \tau$. Assume the contrary. We build a run $r'$ identical to $r$ up to $t_i$. In $r'$, only process $p$ performs actions after time $t_i$. Next, invoke a read$_p$ operation $rd$. By regularity and wait-freedom, $rd$ must return $v_i$. Before performing actions on objects, $p$.data$(r', t_i)$ is empty, thus, from $\tau$-disintegrated storage, $p$ must encounter at least $\tau$ blocks with an origin value of $v_i$ in order to return it. Since no process other than $p$ takes actions, $|S$-labels$(v_i, r', t')| < \tau$ for all $t' \geq t_i$ onward, so $rd$ cannot find these blocks and does not return $v_i$, a contradiction. It follows that in $r'$ at $t_i$, and hence also in $r$ at $t_i$, $|S$-labels$(v_i, r, t_i)| \geq \tau$.

Next, if at $t_1$, $|S$-labels$(v_2, r, t_1)| \geq \tau - 1$ then we are done. Otherwise, observe that objects are accessed one-at-a-time. Therefore, and since $|S$-labels$(v_2, r, t_1)| < \tau - 1$, there exists a time $t$ between $t_1$ and $t_2$ when $|S$-labels$(v_2, r, t)| = \tau - 1$.

Finally, assume that $|S$-labels$(v_1, r, t)| < \tau$. Build a run $r''$ identical to $r$ up to $t$, where again only $p$ takes actions after $t$. As above, it follows by regularity, $\tau$-disintegrated storage, and $p$.data$(r'', t) = \emptyset$, that $rd$ never returns, in violation of wait-freedom. It follows that $|S$-labels$(v_1, r'', t)| = |S$-labels$(v_1, r, t)| \geq \tau$.

The following lemma states that every non-empty set $V$ can be split into two disjoint subsets, where one contains a value that is $(\tau - 1)$-permanent with respect to the other subset. The idea is to show that in the absence of such a value, a reader’s accesses to the shared storage may be scheduled in a way that prevents the reader from obtaining $\tau$ labels of the same value. The logic of the proof is the following: we restrict writes to a set of values $V$, and consider the set $S$ of values with blocks in $p$.data $\cap V$. If no value in $S$ is
Assume by contradiction that the lemma does not hold. We construct an extension Proof. no steps in
hypothesis, (1) consider two cases:
\[ r | p \]
\[ L \]
\[ r \]
\[ \Theta \]
\[ \text{ Permanent, then we can bring the shared storage to a state where none of the values in } S \text{ have } \tau \text{ labels, preventing the reader from obtaining the } \tau \text{ labels required to return. By regularity, readers cannot return other values. The formal proof is slightly more subtle, because it needs to consider } L \text{-labels, as well as labels in the shared storage. It shows that the total number of labels of values in } S \text{ (in both the shared and local storage) remains below } \tau \text{ whenever } p \text{ takes a step.} \]

\[ \text{Lemma 9. Consider a non-empty set of values } V \subseteq \mathcal{V}, \text{ a set of readers } \Theta \subseteq \Pi, \text{ a reader } p \in \Pi \setminus \Theta, \text{ and a finite run } r \text{ of a wait-free regular } \tau \text{-disintegrated storage algorithm. Then there is a subset } S \subseteq V \text{ of size } 1 \leq |S| \leq L \text{ and an extension } r' \text{ of } r \text{ where some value } v \in S \text{ is } (\tau - 1, \Theta \cup \{p\}, V \setminus S) \text{-permanent and s.t. in } r' \setminus r \text{ writes are limited to values from } V \text{ and readers in } \Theta \text{ do not take steps.} \]

\[ \text{Proof. Assume by contradiction that the lemma does not hold. We construct an extension } r' \text{ of } r \text{ where a read}_p \text{ operation includes infinitely many actions of } p \text{ yet does not return. To this end, we show that the following property holds at specific times in } r' \setminus r: \]

\[ \varphi(\hat{r}, t) \triangleq \forall v \in \text{values}_p(\hat{r}, t) \cap V : |\text{All-labels}_p(v, \hat{r}, t)| < \tau. \]

First, extend } r \text{ to } r_0 \text{ by returning any pending read}_p \text{ and write, invoking and returning a write(v)}_0\text{ for some } v_0 \in \text{V (the operations eventually return, by wait-freedom), and finally invoking a read}_p \text{ operation rd without allowing it to take actions. We now prove by induction that for all } k \in \mathbb{N}, \text{ there exists an extension } r' \text{ of } r_0 \text{ where } \varphi(\hat{r}', t_r) \text{ holds and in } r' \setminus r: (2) writes are restricted to values from } V, \text{ (3) } p \text{ performs } k \text{ actions on objects following } rd \text{'s invocation, and (4) } rd \text{'s return is not enabled, and (5) processes in } \Theta \text{ do not take steps.} \]

Base: for } k = 0, \text{ consider } r' = r_0. \text{ (3,5) hold trivially. (2) holds since the only write in } r' \setminus r \text{ is of } v \in \text{V. Since } p \text{ performs no actions following the invocation of } rd, \text{ p.data}(r', t_r) \text{ is empty. Therefore, (1) } \varphi(\hat{r}', t_r) \text{ is vacuously true, and } L \text{-labels}_p(v, r, t) \text{ is empty for all } v \in \text{V, thus (4) } rd \text{'s return is not enabled by } \tau \text{-disintegrated storage.} \]

Step: assume inductively such an extension } r_1 \text{ of } r_0 \text{ with } k > 0 \text{ actions performed by } p \text{ following } rd \text{'s invocation. Since } rd \text{ cannot return, by wait-freedom, an action } a_p \text{ is enabled on some object. We construct an extension } r_2 \text{ of } r_1 \text{ by letting } a_p \text{ occur at time } t_{r_2}. \text{ We consider two cases:} \]

1. \text{ } p \text{ does not obtain a block with an origin value in } V \setminus \text{values}_p(r_1, t_{r_1}) \text{ at } a_p, \text{ thus } \text{values}_p(r_2, t_{r_2}) \cap V \subseteq \text{values}_p(r_1, t_{r_1}) \cap V. \text{ Then, by Observation 1 and the inductive hypothesis, (1) } \varphi(r_2, t_{r_2}) \text{ holds and thus, by } \tau \text{-disintegrated storage, } rd \text{ cannot return any value } v \in \text{values}_p(r_2, t_{r_2}) \cap V \text{ at } t_{r_2}. \text{ (4) It cannot return any other value in } \text{values}_p(r_2, t_{r_2}) \text{ by regularity, and } r_2 \text{ satisfies the induction hypothesis for } k + 1, \text{ as (2,3,5) trivially hold.} \]

2. \text{ } p \text{ obtains a block with origin value } u \in V \setminus \text{values}_p(r_1, t_{r_1}) \text{ at time } t_{r_2}. \text{ Then } |L \text{-labels}_p(u, r_2, t_{r_2})| = 1. \text{ By Observation 1 and the inductive hypothesis, for all } v \in \text{values}_p(r_2, t_{r_2}) \setminus \{u\}, |L \text{-labels}_p(v, r_2, t_{r_2})| < \tau, \text{ and thus } rd \text{'s return is not enabled at time } t_{r_2} \text{ by } \tau \text{-disintegrated storage and regularity.} \]

Let } S = \text{values}_p(r_2, t_{r_2}) \cap V, \text{ and note that } |S| \geq 1 \text{ (since } u \in S) \text{ and that } |S| \leq |p.data| \leq L. \text{ By the contradicting assumption, } u \text{ is not } \langle \tau - 1, \Theta \cup \{p\}, V \setminus S \rangle \text{-permanent in } r_2, \text{ thus there exists an extension } r_3 \text{ of } r_2 \text{ s.t. } |S \text{-labels}(u, r_3, t_{r_3})| < \tau - 1 \text{ and in } r_3 \setminus r_2 \text{ writes are limited to values from } V \setminus S \text{ and no readers in } \Theta \cup \{p\} \text{ take steps (3,5 hold). Since } p \text{ takes no steps in } r_3 \setminus r_2, \text{ we have that } L \text{-labels}_p(u, r_3, t_{r_3}) = L \text{-labels}_p(u, r_2, t_{r_2}), \text{ yielding:} \]

\[ |\text{All-labels}_p(u, r_3, t_{r_3})| \leq |L \text{-labels}_p(u, r_2, t_{r_2})| + |S \text{-labels}(u, r_3, t_{r_3})| < 1 + (\tau - 1) = \tau. \text{ (1) } \]

All writes invoked after } t_{r_2} \text{ are from } W_{V \setminus S} \text{ (2 holds), and therefore do not produce new labels associated with values in } S. \text{ Since no values in } S \text{ are written after } t_{r_1} \text{ and}
readers’ actions do not affect the sets \( S \)-labels, by Observation 1, we have that \( \forall v \in S, \)
\( \text{All-labels}_p(v, r_3, t_3) \subseteq \text{All-labels}_p(v, r_1, t_1), \) and since \( \varphi(r_1, t_1) \) holds (inductively) and \( S \setminus \{u\} \subseteq \text{values}_p(r_1, t_1) \cap V, \)
\[ \forall v \in S \setminus \{u\} : |\text{All-labels}_p(v, r_3, t_3)| \leq |\text{All-labels}_p(v, r_1, t_1)| < \tau. \] (2)
From Equations 1 and 2, and since \( \text{values}_p(r_3, t_3) \cap V = \text{values}_p(r_2, t_2) \cap V = S, \) we get \( \varphi(r_3, t_3) \) (1). Since \( r \varphi \)'s return was not enabled at time \( t_2 \) and it took no actions since, its return is still not enabled (4), and we are done.  

\subsection{4.2 Invisible reads}

We now consider a setting of a single reader and single writer where \( \text{reads} \) are invisible. To show the following theorem, we “blow up” the shared storage by repeatedly invoking Lemma 9, each time adding one more \((\tau - 1)\)-permanent value, yielding the following bound:

\begin{theorem}

The storage cost of a regular \( \tau \)-disintegrated storage wait-free SRSW register emulation where \( \text{reads} \) are invisible is at least \( \tau + (\tau - 1) \left\lceil \frac{2^{\phi} - 1}{L} \right\rceil \) blocks.

\end{theorem}

When readers are invisible, the set \( \Theta \) is of no significance, so we consider \( \emptyset \). Given a set of values \( V, \) the value added by Lemma 9 is \( (\tau - 1, \emptyset, V \setminus S) \)-permanent for a smaller set of values \( V \setminus S \) where \( |S| \leq L. \) Therefore, we can invoke Lemma 9 \( m = \left\lceil \frac{2^{\phi} - 1}{L} \right\rceil - 1 \) times before running out of values, showing the following:

\begin{lemma}

Let \( p \in \Pi \) be an invisible reader. There exist finite runs \( r_0, \ldots, r_m \) and sets of values \( V_0 \supseteq V_1 \supseteq \ldots \supseteq V_m \) and \( U_0 \subseteq U_1 \subseteq \ldots \subseteq U_m, \) such that for all \( 0 \leq k \leq m: \)
(1) \( |V_k| \geq 2^D - Lk, \) \( |U_k| = k, \) \( V_k \cap U_k = \emptyset, \) and
(2) all elements of \( U_k \) are \((\tau - 1, \emptyset, V_k)\)-permanent in \( r_k. \)

\end{lemma}

\textbf{Proof.} By induction. Base: \( r_0 \) is the empty run, \( V_0 = \emptyset \) and \( U_0 = \emptyset. \) Assume inductively that the lemma holds for \( k < m. \) Since \( m < \frac{2^{\phi} - 1}{L}, \) we get: \( |V_k| > 2^D - L \left\lceil \frac{2^{\phi} - 1}{L} \right\rceil = 1. \) Since \( V_k \) is non-empty and \( |\emptyset| < R, \) by Lemma 9 there exist an extension \( r_{k+1} \) of \( r_k \) where \( \text{writes} \) in \( r_{k+1} \cap r_k \) are limited to values from \( V_k, \) a set \( S \subseteq V_k, 1 \leq |S| \leq L, \) and a value \( v \in S \) that is \( (\tau - 1, \{p\}, V_k \setminus S)\)-permanent in \( r_{k+1}. \)

Let \( V_{k+1} = V_k \setminus S \) and \( U_{k+1} = U_k \cup \{v\}. \) Note that, because \( V_k \cap U_k = \emptyset \) and \( v \in S \subseteq V_k, \) we get that \( V_{k+1} \cap U_{k+1} = \emptyset \) and \( |U_{k+1}| = |U_k| + 1 = k + 1. \) Since \( 1 \leq |S| \leq L \) we have that \( V_k \supseteq V_{k+1} \) and \( |V_{k+1}| \geq |V_k| - |S| \geq 2^D - L(k + 1). \) By the inductive assumption and Observation 5, all values in \( U_{k+1} \) are \((\tau - 1, \emptyset, V_{k+1})\)-permanent in \( r_{k+1}. \) By Observation 7, \( v \) is also \((\tau - 1, \emptyset, V_{k+1})\)-permanent in \( r_{k+1} \) and we are done.  

Our bound combines the \( 2\tau - 1 \) blocks of Claim 8 with the \((\tau - 1)m\) from Lemma 11:

\textbf{Proof (Theorem 10).} Consider an invisible reader \( p \in \Pi \) and construct \( r_m, V_m, \) and \( U_m \) as in Lemma 11. Note that \( V_m \) contains at least two distinct values that are not in \( U_m, \) since \( V_m \cap U_m = \emptyset \) and \( |V_m| \geq 2^D - Lm > 2^D - L \left\lceil \frac{2^{\phi} - 1}{L} \right\rceil = 1. \) Extend \( r_m \) to \( r_{m+1} \) by invoking and returning \( \text{write}(v) \) and \( \text{write}(v') \) for \( v, v' \in V_m. \)

By Claim 8, there is a time \( t \geq t_{m+1} \) when there are \( 2\tau - 1 \) blocks in the shared storage with origin values of \( v \) or \( v'. \) In addition, by Lemma 11, \( U_m \) consists of \( m \) values that are \((\tau - 1, \emptyset, V_m)\)-permanent in \( r_m, \) and since \( \text{writes} \) in \( r_{m+1} \setminus r_m \) are of values from \( V_m, \) the values in \( U_m \) remain \((\tau - 1, \emptyset, V_m)\)-permanent in \( r_{m+1}. \) By Observation 6:

\[ n \geq 2\tau - 1 + (\tau - 1)m = \tau + (\tau - 1)(m + 1) = \tau + (\tau - 1) \left\lceil \frac{2^{\phi} - 1}{L} \right\rceil. \]  

4.3 Visible reads

We now consider systems where readers may write meta-data in the shared storage. We use a similar technique as in Lemma 11, except that due to readers’ updates, the indistinguishability argument can no longer be used. Instead, we invoke a new reader for each extension, and therefore the number of runs might be limited by the number of readers, R:

► Theorem 12. The storage cost of a regular τ-disintegrated storage wait-free MRSW register emulation with R readers is at least \( \tau + (\tau - 1) \cdot \min \left( \left\lceil \frac{2^D - 1}{L} \right\rceil, R \right) \) blocks.

To achieve this bound, we use Lemma 9 again to construct \( N = \min \left( \left\lceil \frac{2^D - 1}{L} \right\rceil, R \right) - 1 \) extensions of the empty run (note that it does not assume invisible reads).

► Lemma 13. There exist finite runs \( r_0, \ldots, r_N \), sets of values \( V_0 \supset V_1 \supset \ldots \supset V_N \) and \( U_0 \subset U_1 \subset \ldots \subset U_N \), and sets of readers \( \Theta_0 \subset \Theta_1 \subset \ldots \subset \Theta_N \), such that for all \( 0 \leq k \leq N \):

1. \( |V_k| \geq 2^D - Lk \), \( |U_k| = |\Theta_k| = k \), \( V_k \cap U_k = \emptyset \), and \( k \)
2. all elements of \( U_k \) are \( \langle \tau - 1, \Theta_k, V_k \rangle \)-permanent in \( r_k \).

Proof. By induction. Base: \( r_0 \) is the empty run, \( V_0 = V \), \( \Theta_0 = U_0 = \emptyset \). Assume inductively such \( r_k, V_k, U_k, \) and \( \Theta_k \) for \( k < N \), and construct \( r_{k+1} \) as follows: since \( R - |\Theta_k| > 0 \), there is a reader \( p \in \Pi \setminus \Theta_k \). Since \( N < \frac{2^D - 1}{L} \), we get \( |V_k| > 2^D - LN > 1 \). Therefore, by Lemma 9, there exist an extension \( r_{k+1} \) of \( r_k \) where in \( r_{k+1} \setminus r_k \) writes are limited to values from \( V_k \) and readers in \( \Theta_k \) do not take steps, a set \( S \subseteq V_k \), \( 1 \leq |S| \leq L \), and a value \( v \in S \) that is \( \langle \tau - 1, \Theta_k \cup \{ \overline{p} \}, V_k \setminus S \rangle \)-permanent in \( r_{k+1} \).

Let \( V_{k+1} = V_k \setminus S \) and \( U_{k+1} = U_k \cup \{ v \} \). Note that, because \( V_k \cap U_k = \emptyset \) and \( v \in S \subseteq V_k \), it follows that \( V_{k+1} \cap U_{k+1} = \emptyset \) and \( |U_{k+1}| = k + 1 \). Furthermore, since \( 1 \leq |S| \leq L \), we get: \( V_k \supset V_{k+1} \) and \( |V_{k+1}| \geq |V_k| - |S| \geq 2^D - L(k + 1) \). Finally, let \( \Theta_{k+1} = \Theta_k \cup \{ \overline{p} \} \). By the inductive assumption and Observation 5, all values in \( U_k \) are \( \langle \tau - 1, \Theta_{k+1}, V_{k+1} \rangle \)-permanent in \( r_{k+1} \), and so all of \( U_{k+1} \) is \( \langle \tau - 1, \Theta_{k+1}, V_{k+1} \rangle \)-permanent in \( r_{k+1} \), as needed.

From Lemma 13, in \( r_N \) there is a set of \( N \) \( \langle \tau - 1 \rangle \)-permanent values, inducing a cost of \( (\tau - 1)N \). We use Claim 8 to increase the bound by \( 2\tau - 1 \) additional blocks.

Proof (Theorem 12). Construct \( r_N, V_N, U_N, \) and \( \Theta_N \) as in Lemma 13. Note that, since \( R - N \geq 1 \), there exists \( p \in \Pi \Theta_N \). Since \( V_N \cap U_N = \emptyset \) and \( |V_N| \geq 2^D - LN > 2^D - L\frac{2^D - 1}{L} = 1 \), \( V_N \setminus U_N \) contains at least two values. Extend \( r_N \) to \( r_{N+1} \) by invoking and returning \( \text{write}(v) \) and \( \text{write}(v') \) for \( v, v' \in V_N \setminus U_N \).

By Claim 8, there is a time \( t \geq t_N \) in \( r_{N+1} \) when there are \( 2\tau - 1 \) blocks in the shared storage with origin values of \( v \) or \( v' \). \( U_N \) consists of \( N \) additional values that are \( \langle \tau - 1, \Theta_N, V_N \rangle \)-permanent in \( r_N \), and since in \( r_{N+1} \setminus r_N \) writes are of values from \( V_N \) and no reader in \( \Theta_N \) takes steps, the values in \( U_N \) remain \( \langle \tau - 1, \Theta_N, V_N \rangle \)-permanent in \( r_{N+1} \). By Observation 6, the storage cost is:

\[ n \geq 2\tau - 1 + (\tau - 1)N = \tau + (\tau - 1)(N + 1) = \tau + (\tau - 1) \cdot \min \left( \left\lceil \frac{2^D - 1}{L} \right\rceil, R \right). \]

5 Lower bounds for common write disintegrated storage

While the results of the previous section hold a fortiori for \( \tau \)-common write algorithms, in this case we are able to show stronger results, independent of the local storage size. Intuitively, this is because readers can no longer reuse blocks they obtained from previous \text{writes} of
the same value, and so we can prolong the execution that blows up the shared storage by rewriting values. Section 5.1 proves a general attribute of $\tau$-common write algorithms. We show in Section 5.2 that even with a single reader (and a single writer), if reads are invisible, then the required storage cost is at least $\tau \cdot 2^k$. In Section 5.3 we prove a bound for visible reads.

5.1 General observation

In this section we define a property that is a special case of $k$-permanence, which additionally requires that the set of labels associated with a write does not change.

Definition 14 (Constancy). Consider a finite run $r$, $k \in \mathbb{N}$, a set $S \subseteq V$, and a set of readers $\Theta \subseteq \Pi$. We say that a write $w \in W$ is $(k, \Theta, S)$-constant in $r$ if in every finite extension $r'$ of $r$ s.t. in $r' \setminus r$ readers in $\Theta$ do not take actions and writes are limited to values from $S$, $S$-labels $(w, r', t_r) = S$-labels $(w, r, t_r)$ and $|S$-labels $(w, r', t_r)| = k$.

Similarly to Observation 7, it can be shown that:

Observation 15. Consider $V \subseteq \mathbb{V}$, $k \in \mathbb{N}$, and a finite run $r$ with an invisible reader $p \in \Pi$. If $w \in W$ is $(k, \{p\}, V)$-constant in $r$ then $w$ is $(k, \emptyset, V)$-constant in $r$.

We next prove a stronger variant of Lemma 9 that allows us to add a permanent write to the shared storage while some set $C \subseteq W$ of writes are constant. Note that since the number of writes of a value $v$ is infinite and the number of constant writes in a finite run is finite, for any non-empty $V \subseteq \mathbb{V}$, $\mathbb{W}_V \setminus C$ is non-empty.

Lemma 16. Consider a non-empty set of values $V \subseteq \mathbb{V}$, a set of readers $\Theta \subseteq \Pi$, a reader $p \in \Pi \setminus \Theta$, and a finite run $r$ of a wait-free regular $\tau$-common write algorithm. Let $C$ be a set of writes that are $(\tau - 1, \Theta, V)$-constant in $r$. Then there is an extension $r'$ of $r$ where some $w \in \mathbb{W}_V \setminus C$ returns and is $(\tau - 1, \Theta \cup \{p\}, V)$-constant, and s.t. in $r' \setminus r$ writes are limited to $\mathbb{W}_V$ and readers in $\Theta$ do not take actions.

Proof. Assume by contradiction that the lemma does not hold. We build an extension $r'$ of $r$ where a read$_p$ operation includes infinitely many actions of $p$ yet does not return. To this end, we show that the following property holds at specific times in $r' \setminus r$:

$$\psi(r, t) \triangleq \forall w \in \text{writes}_p(r, t) \cap \mathbb{W}_V : |\text{All-labels}_p(w, r, t)| < \tau.$$ 

Note that, by definitions of $\tau$-common write and of All-labels, whenever $\psi(r', t)$ holds, no pending read$_p$ invocation can return a value $v \in \text{values}_p(r', t) \cap V$.

First, extend $r$ to $r_0$ by returning any pending read$_p$ and write, invoking and returning a write$(v_0)$ for some $v_0 \in V$ (the operations eventually return, by wait-freedom), and finally invoking a read$_p$ operation $rd$ without allowing it to take actions. We now prove by induction that for all $k \in \mathbb{N}$, there exists an extension $r'$ of $r_0$ where (1) $\psi(r', t_r)$ holds, (2) no write is pending at $t_r$, and in $r' \setminus r$: (3) writes are restricted to $\mathbb{W}_V$, (4) $p$ performs $k$ actions on objects after invoking $rd$, (5) $rd$’s return is not enabled, and (6) processes in $\Theta$ do not take steps.

Base: for $k = 0$, consider $r' = r_0$. (2,4,6) hold trivially. (3) holds since the only write in $r' \setminus r$ is $w_0 \in \mathbb{W}_V$. Since $p$ performs no actions following the invocation of $rd$, p.data$(r', t_r)$ is empty. Therefore, (1) $\psi(r', t_r)$ is vacuously true, and L-labels$_p(w, r', t_r)$ is empty for all $w \in \mathbb{W}_V$, thus (5) $rd$’s return is not enabled by $\tau$-common write.
Step: assume inductively such an extension \( r_1 \) of \( r_0 \) with \( k \geq 0 \) actions by \( p \) following \( rd \)'s invocation. Since \( rd \) cannot return, by wait-freedom, an action \( a_p \) is enabled on some object. We construct an extension \( r_2 \) of \( r_1 \) by letting \( a_p \) occur at time \( t_{r_1} \). We then consider three cases:

1. \( p \) does not obtain a block with an origin \( \text{write} \) in \( \mathbb{W}_V \setminus \text{writes}_p(r_1, t_{r_1}) \) at \( a_p \), thus \( \text{writes}_p(r_2, t_{r_2}) \cap \mathbb{W}_V \subseteq \text{writes}_p(r_1, t_{r_1}) \cap \mathbb{W}_V \). Then, by Observation 1 and the inductive hypothesis, (1) \( \psi(r_2, t_{r_2}) \) holds and thus, by \( \tau \)-common write, \( rd \) cannot return any value \( v \in \text{values}_p(r_2, t_{r_2}) \cap \mathbb{V} \) at \( t_{r_2} \). (5) It cannot return any other value in \( \text{values}_p(r_2, t_{r_2}) \) by regularity, and \( r_2 \) satisfies the induction hypothesis for \( k + 1 \) as \((2,3,4,6)\) trivially hold.

2. \( p \) obtains a block with origin \( \text{write} \) \( w' \in C \cap \mathbb{W}_V \setminus \text{writes}_p(r_1, t_{r_1}) \) at \( a_p \). Then \(|L\text{-labels}_p(w', r_1, t_{r_1})| = 0\). Since \( w' \) is \((\tau - 1, \Theta, V)\)-constant in \( r \) and in \( r_1 \setminus r \text{writes} \) are restricted to \( \mathbb{W}_V \) and processes in \( \Theta \) do not take steps (inductively), then by definition of constancy, \(|S\text{-labels}(w', r_1, t_{r_1})| = \tau - 1\). By Observation 1, for all \( w \in \text{writes}_p(r_2, t_{r_2}) \cap \mathbb{W}_V: \text{All-labels}_p(w, r_1, t_{r_1}) \subseteq \text{All-labels}_p(w, r_1, t_{r_1})\). Therefore \(|\text{All-labels}_p(w', r_2, t_{r_2})| \leq |\text{All-labels}_p(w', r_1, t_{r_1})| + |S\text{-labels}(w', r_1, t_{r_1})| = \tau - 1\). Together with the inductive hypothesis, \( \forall w \in \text{writes}_p(r_2, t_{r_2}) \cap \mathbb{W}_V \setminus \{w'\}, |\text{All-labels}_p(w, r_2, t_{r_2})| \leq |\text{All-labels}_p(w, r_1, t_{r_1})| < \tau; \psi(r_2, t_{r_2}) \) follows, thus (5) follows, and \((2,3,4,6)\) trivially hold.

3. \( p \) obtains a block with origin \( \text{write} \) \( w' \in \mathbb{W}_V \setminus \text{writes}_p(r_1, t_{r_1}) \cup C \) at \( a_p \). Then \(|L\text{-labels}_p(w', r_2, t_{r_2})| = 1\) and the number of labels of other \( \text{writes} \) in \( \text{writes}_p(r_2, t_{r_2}) \) does not increase following \( a_p \), thus \( rd \)'s return is not enabled at \( t_{r_2} \) by \( \tau \)-common write and regularity.

By the contradicting assumption, \( w' \) is not \((\tau - 1, \Theta \cup \{p\}, V)\)-permanent in \( r_2 \), thus there is an extension \( r_3 \) of \( r_2 \) s.t. \(|S\text{-labels}(w', r_3, t_{r_3})| < \tau - 1\) and in \( r_3 \setminus r_2 \text{writes} \) are limited to \( \mathbb{W}_V \) and no readers in \( \Theta \cup \{p\} \) take steps (3,4,6 hold). \( S \) we further extend \( r_3 \) to \( r_4 \) by letting any pending \( \text{write} \) return (2).

Let \( S = \text{writes}_p(r_2, t_{r_2}) \cap \mathbb{W}_V \). Since every \( w \in S \) returns before \( t_{r_2} \) by the inductive assumption, the \( \text{writes} \) in \( r_4 \setminus r_2 \) do not produce new labels associated with \( w \). Since readers do not affect the sets \( S\text{-labels} \), it follows that \( \forall w \in S: S\text{-labels}(w, r_4, t_{r_4}) \subseteq S\text{-labels}(w, r_3, t_{r_3}) \subseteq S\text{-labels}(w, r_2, t_{r_2}) \). Next, \( p \) takes no steps in \( r_4 \setminus r_2 \) (4 holds), thus \( \forall w \in S: L\text{-labels}_p(w', r_4, t_{r_4}) = L\text{-labels}_p(w', r_2, t_{r_2}) \). It follows that:

\[ |\text{All-labels}_p(w', r_4, t_{r_4})| \leq |L\text{-labels}_p(w', r_2, t_{r_2})| + |S\text{-labels}(w', r_3, t_{r_3})| < 1 + (\tau - 1) = \tau. \] (3)

Moreover, by Observation 1 and the inductive assumption that \( \psi(r_1, t_{r_1}) \) holds,

\[ \forall w \in S \setminus \{w'\}: |\text{All-labels}_p(w, r_4, t_{r_4})| \leq |\text{All-labels}_p(w, r_1, t_{r_1})| < \tau. \] (4)

From Equations 3 and 4, and since \( \text{writes}_p(r_4, t_{r_4}) \cap \mathbb{W}_V = \text{writes}_p(r_2, t_{r_2}) \cap \mathbb{W}_V = S \), we get (1) \( \psi(r_4, t_{r_4}) \). Since \( rd \)'s return is not enabled at \( t_{r_2} \) and (4) it took no actions since, its return is not enabled anywhere in \( r_4 \setminus r_1 \) (5), and we are done.

### 5.2 Invisible reads

We prove the following theorem by constructing a run with an exponential number of \( \tau \)-permanent values. The idea is to show that if there is a value in the domain for which there is no \( \tau \)-permanent \( \text{write} \), then infinitely many \( \text{writes} \) remain \((\tau - 1)\)-constant, which is of course impossible.

\textbf{Theorem 17}. The storage cost of a regular \( \tau \)-common \( \text{write} \) wait-free SRSW register emulation where \( \text{reads} \) are invisible is at least \( \tau \cdot 2^D \) blocks.
Lemma 18. Consider a non-empty set of values $V \subseteq \mathbb{V}$ and a finite run $r$. Let $C$ be a set of \texttt{writes} that are $\langle \tau - 1, \emptyset, V \rangle$-constant in $r$. Then there exists an extension $r'$ of $r$ where \texttt{writes} in $r' \setminus r$ are limited to $\mathbb{V}_V$, and some $w \in \mathbb{V}_V \setminus C$ is either $\langle \tau - 1, \emptyset, V \rangle$-constant or $\langle \tau, \emptyset, V \rangle$-permanent in $r'$.

Proof. Let $p \in \Pi$ be a reader. By Lemma 16, there is an extension $r'$ of $r$ where \texttt{writes} in $r' \setminus r$ are limited to $\mathbb{V}_V$ and some $w \in \mathbb{V}_V \setminus C$ returns and is $\langle \tau - 1, \{p\}, V \rangle$-permanent. By Observation 7, if $w$ is $\langle \tau, \{p\}, V \rangle$-permanent in $r'$, then $w$ is $\langle \tau, \emptyset, V \rangle$-permanent in $r'$ and the lemma follows. Otherwise, there exists an extension $r''$ of $r'$ where in $r'' \setminus r'$ \texttt{writes} are limited to $\mathbb{V}_V$ and $p$ takes no steps, and $|S\text{-}labels(w, r'', t_{r''})| < \tau$. Since $w$ is $\langle \tau - 1, \{p\}, V \rangle$-permanent in $r''$, $|S\text{-}labels(w, r'', t_{r''})| = \tau - 1$.

We show that $w$ is $\langle \tau - 1, \emptyset, V \rangle$-constant in $r''$. Consider an extension $r'''$ of $r''$ where \texttt{writes} are limited to values from $V$ and $p$ takes no steps in $r''' \setminus r''$. Since $w$ has already returned by time $t_{r''}$, no new blocks with an origin \texttt{write} of $w$ can be added to the shared storage in $r'''$ after $t_{r''}$. It follows that $S\text{-}labels(w, r'''', t_{r''}) \subseteq S\text{-}labels(w, r'', t_{r''})$. However, since $w$ is $\langle \tau - 1, \{p\}, V \rangle$-permanent in $r''$, and in $r''' \setminus r'' \texttt{writes}$ are limited $\mathbb{V}_V$ and $p$ takes no steps, then $|S\text{-}labels(w, r'''', t_{r''})| \geq \tau - 1 = |S\text{-}labels(w, r'', t_{r''})|$, yielding that $S\text{-}labels(w, r'''', t_{r''}) = S\text{-}labels(w, r'', t_{r''})$. Thus, $w$ is $\langle \tau - 1, \{p\}, V \rangle$-constant in $r'''$. The lemma follows from Observation 15.

Claim 19. Consider a finite run $r$ and a non-empty $V \subseteq \mathbb{V}$. Then there is an extension $r'$ of $r$ s.t. \texttt{writes} in $r' \setminus r$ are limited to $\mathbb{V}_V$, and some $w \in \mathbb{V}_V$ is $\langle \tau, \emptyset, V \rangle$-permanent in $r'$.

Proof. Consider an algorithm with storage cost $n$, and let $m = \lceil n/(\tau - 1) \rceil + 1$. Assume by contradiction that the claim does not hold. We get a contradiction by constructing $m + 1$ extensions of $r$: $r_0, ..., r_m$ with sets of \texttt{writes} $C_0 \subseteq C_1 \subset \cdots \subset C_m \subseteq \mathbb{V}_V$ s.t. for all $0 \leq k \leq m$:

1. \texttt{writes} in $r_k \setminus r$ are limited to $\mathbb{V}_V$, and
2. $C_k$ is a set of $k$ \texttt{writes} that are $\langle \tau - 1, \emptyset, V \rangle$-constant in $r_k$.

Note that in $r_m$, $\left\lceil \frac{n}{\tau - 1} \right\rceil + 1$ \texttt{writes} are $\langle \tau - 1, \emptyset, V \rangle$-constant, implying a storage cost greater than $n$ by Observation 6, a contradiction.

The construction is by induction. The base case vacuously holds for $r_0 = r$, $C_0 = \emptyset$. Assume inductively such $r_k$ and $C_k$ for $k < m$. By Lemma 18 there exists an extension $r_{k+1}$ of $r_k$ where some $w \in \mathbb{V}_V \setminus C_k$ is either $\langle \tau, \emptyset, V \rangle$-permanent or $\langle \tau - 1, \emptyset, V \rangle$-constant, and \texttt{writes} in $r_{k+1} \setminus r_k$ are limited to $\mathbb{V}_V$. Since all \texttt{writes} in $C_k$ are $\langle \tau - 1, \emptyset, V \rangle$-constant in $r_k$ they are also $\langle \tau - 1, \emptyset, V \rangle$-constant in $r_{k+1}$. By the contracting assumption, $w$ is not $\langle \tau, \emptyset, V \rangle$-permanent in $r_{k+1}$ thus it is $\langle \tau - 1, \emptyset, V \rangle$-constant in the run. Let $C_{k+1} = C_k \cup \{w\}$, therefore $|C_{k+1}| = k + 1$ and all \texttt{writes} in $C_{k+1}$ are $\langle \tau - 1, \emptyset, V \rangle$-constant in $r_{k+1}$, as needed.

We are now ready to prove our lower bound of $\tau \cdot 2^D$ blocks.

Proof (Theorem 17). We show that there exist $2^D + 1$ finite runs $r_0, r_1, ..., r_{2^D}$ and sets of values $V_0 \supseteq V_1 \supseteq ... \supseteq V_{2^D}$ and $U_0 \supseteq U_1 \subset ... \subset U_{2^D}$, such that for all $0 \leq k \leq 2^D$:

1. $|V_k| = 2^D - k, |U_k| = k, V_k \cap U_k = \emptyset$, and
2. all elements of $U_k$ are $\langle \tau, \emptyset, V_k \rangle$-permanent in $r_k$.

By induction. Base: $r_0$ is the empty run, $V_0 = \mathbb{V}, U_0 = \emptyset$. Assume inductively such $r_k, V_k$, and $U_k$ for $k < 2^D$, and construct $r_{k+1}$ as follows: first, because $|V_k| = 2^D - k > 0$, by Claim 19 there is an extension $r_{k+1}$ of $r_k$ where \texttt{writes} in $r_{k+1} \setminus r_k$ are limited to $\mathbb{V}_{V_k}$ and some $w \in \mathbb{V}_{V_k}$ is $\langle \tau, \emptyset, V_k \rangle$-permanent.
Consider the value \( v \in V_k \) written by \( w \). By Observation 5, \( v \) is \((\tau, \emptyset, V_k)\)-permanent in \( r_{k+1} \). Let \( V_{k+1} = V_k \setminus \{ v \} \), then \|V_{k+1}| = |V_k| - 1 = 2^D - (k + 1)\). Further let \( U_{k+1} = U_k \cup \{ v \} \). Note that, because \( V_k \cap U_k = \emptyset \), we get \( v \not\in U_k \) and hence \( V_{k+1} \cap U_{k+1} = \emptyset \) and \|U_{k+1}| = |U_k| + 1 = k + 1\). Since \( V_k \supseteq V_{k+1} \), then \( v \) is \((\tau, \emptyset, V_{k+1})\)-permanent. Additionally, writes in \( r_{k+1} \setminus r_k \) are from \( W_{V_k} \), thus by the inductive assumption and Observation 5, values in \( U_k \) are \((\tau, \emptyset, V_{k+1})\)-permanent in \( r_{k+1} \), and so all of \( U_{k+1} \) are \((\tau, \emptyset, V_{k+1})\)-permanent in \( r_{k+1} \).

Finally, \( U_{2D} \) holds \( 2^D \) values that are \((\tau, \emptyset, \emptyset)\)-permanent in \( r_{2D} \). By Observation 6:
\[
n \geq \tau \cdot 2^D.
\]

### 5.3 Visible reads

To prove a lower bound on the cost of systems with visible reads, we create a similar construction, except that the number of extensions might be limited by the number of readers, \( R \). Instead, the bound depends on \( \min (2^D - 1, R) \):

**Theorem 20.** The storage cost of a regular \( \tau \)-common write wait-free MRSW register emulation is at least \( \tau + (\tau - 1) \cdot \min (2^D - 1, R) \) blocks.

Let \( N = \min (2^D - 1, R) - 1 \). We build a run with \( N \) \((\tau - 1)\)-permanent values:

**Lemma 21.** There exist finite runs \( r_0, r_1, \ldots, r_N \), sets of values \( V_0 \supseteq V_1 \supseteq \ldots \supseteq V_N \) and \( U_0 \subseteq U_1 \subseteq \ldots \subseteq U_N \), and sets of readers \( \Theta_0 \subseteq \Theta_1 \subseteq \ldots \subseteq \Theta_N \), s.t. for all \( 0 \leq k \leq N \):
\[ (1) \quad |V_k| = 2^D - k, \quad |U_k| = |\Theta_k| = k, \quad V_k \cap U_k = \emptyset, \quad \text{and} \]
\[ (2) \quad \text{all elements of } U_k \text{ are } (\tau - 1, \Theta_k, V_k)\text{-permanent in } r_k. \]

**Proof.** By induction. Base: \( r_0 \) is the empty run, \( V_0 = V, \Theta_0 = U_0 = \emptyset \). Assume inductively such \( r_k, V_k, U_k, \) and \( \Theta_k \) for \( k < N \), and construct \( r_{k+1} \) as follows: since \( R - |\Theta_{k+1}| > 0 \), there is a reader \( p \in \Pi \setminus \Theta_k \). Moreover, \( |V_k| > 2^D - N > 0 \). Therefore, by Lemma 16, there is an extension \( r_{k+1} \) of \( r_k \) where writes in \( r_{k+1} \setminus r_k \) are limited to \( W_{V_k} \), readers in \( \Theta_k \) do not take steps in \( r_{k+1} \setminus r_k \), and some \( w \in W_{V_k} \) returns and is \((\tau - 1, \Theta_k \cup \{ p \}, V_k)\)-permanent in \( r_{k+1} \).

Let \( \Theta_{k+1} = \Theta_k \cup \{ p \} \), and consider the value \( v \in V_k \) written by \( w \). By Observation 5, \( v \) is \((\tau - 1, \Theta_{k+1}, V_k)\)-permanent. Let \( V_{k+1} = V_k \setminus \{ v \} \), then \( |V_{k+1}| = 2^D - (k + 1) \). Further let \( U_{k+1} = U_k \cup \{ v \} \). Since \( V_k \cap U_k = \emptyset \), we get that \( V_{k+1} \cap U_{k+1} = \emptyset \) and \|U_{k+1}| = k + 1\).

Since \( V_k \supset V_{k+1} \), \( v \) is \((\tau - 1, \Theta_{k+1}, V_{k+1})\)-permanent. In addition, in \( r_{k+1} \setminus r_k \) writes are limited to \( W_{V_k} \) and readers in \( \Theta_k \) do not take steps, and since \( \Theta_k \subseteq \Theta_{k+1} \), then by the inductive assumption and Observation 5, all values in \( U_k \) are \((\tau - 1, \Theta_{k+1}, V_{k+1})\)-permanent. Therefore all elements of \( U_{k+1} \) are \((\tau - 1, \Theta_{k+1}, V_{k+1})\)-permanent in \( r_{k+1} \), as needed.

From Lemma 21, in \( r_N \) there is a set of \( N \) \((\tau - 1)\)-permanent values, inducing a cost of \((\tau - 1)N \). We use Claim 8 to increase the bound by \( 2\tau - 1 \) additional blocks.

**Proof (Theorem 20).** Construct \( r_N, V_N, U_N, \) and \( \Theta_N \) as in Lemma 21. Note that, since \( R - N \geq 1 \), there is a reader \( p \in \Pi \setminus \Theta_N \). Since \( V_N \cap U_N = \emptyset \) and \|V_N| = 2^D - N = 2^D - (\min (2^D - 1, R) - 1) \geq 2 \), \( V_N \) contains two values, and they are not in \( U_N \). Extend \( r_N \) to \( r_{N+1} \) by invoking and returning \( write(v) \) and \( write(v') \) for \( v, v' \in V_N \).

By Claim 8, there is a time \( t \geq t_{r_N} \) in \( r_{N+1} \) when there are \( 2\tau - 1 \) blocks in the shared storage with origin values of \( v \) or \( v' \). In addition, \( U_N \) consists of \( N \) values that are \((\tau - 1, \Theta_N, V_N)\)-permanent in \( r_N \), and since in \( r_{N+1} \setminus r_N \) writes are of values from \( V_N \) and no reader in \( \Theta_N \) takes steps, the values in \( U_N \) remain \((\tau - 1, \Theta_N, V_N)\)-permanent in \( r_{N+1} \). By Observation 6, the storage cost amounts to at least:
\[
n \geq 2\tau - 1 + (\tau - 1)N = \tau + (\tau - 1)(N + 1) = \tau + (\tau - 1) \cdot \min (2^D - 1, R). \]
6 Discussion

We have shown lower bounds on the space complexity of regular wait-free $\tau$-disintegrated storage algorithms. Although our bounds are stated in terms of blocks, there are scenarios where they entail concrete bounds in terms of bits. In replication, each block stores an entire value, thus the block sizes are $D$ bits. Other applications use symmetric coding where all blocks are of equal size. Using a simple pigeonhole argument, it can be shown that in $\tau$-disintegrated storage emulations that use symmetric coding and that are not $(\tau + 1)$-disintegrated, the size of blocks is at least $D/\tau$ bits, yielding bounds of $D \cdot 2^D$ and $D + D^{\tau-1} \cdot \min(2^D - 1, R)$ with invisible and visible readers, respectively.

Our lower bounds for the common write case explain, for the first time, why previous coded storage algorithms have either had the readers write or consumed exponential (or even unbounded) space. Similarly, they establish why previous emulations of large registers from smaller ones have either had the readers write, had the writer share blocks among different writes, or consumed exponential space.

Our work leaves several open questions. First, when replication is used as a means to overcome Byzantine faults or data corruption, our results suggest that there might be an interesting trade-off between the shared storage cost and the size of local memory at the readers, and a possible advantage to systems that apply replication rather than error correction codes: we have shown that, with invisible readers, the former require $\Omega(2^D/L)$ blocks, rather than the $\Omega(2^D)$ blocks needed by the latter. Whether there are algorithms that achieve this lower cost remains an open question. Second, it is unclear how the bounds would be affected by removing our assumption that each block in the shared storage pertains to a single write. Wei [25] has provided a partial answer to this questions by showing that similar bounds hold without this assumption, but only in the case of emulating large registers from smaller ones without meta-data at all. Similarly, it would be interesting to study whether allowing readers to write data (and not only signals) impacts the storage cost. Finally, future work may consider additional sub-classes of disintegrated storage, e.g., with unresponsive objects, and show that additional costs are incurred in these cases.

References

A. Berger, I. Keidar, and A. Spiegelman


