The Complexity of Dynamic Data Race Prediction

Umang Mathur  
University of Illinois, Urbana Champaign  
USA  
umathur3@illinois.edu

Andreas Pavlogiannis  
Aarhus University  
Denmark  
pavlogiannis@cs.au.dk

Mahesh Viswanathan  
University of Illinois, Urbana Champaign  
USA  
vmahesh@illinois.edu

Abstract

Writing concurrent programs is notoriously hard due to scheduling non-determinism. The most common concurrency bugs are data races, which are accesses to a shared resource that can be executed concurrently. Dynamic data-race prediction is the most standard technique for detecting data races: given an observed, data-race-free trace \( t \), the task is to determine whether \( t \) can be reordered to a trace \( t' \) that exposes a data-race. Although the problem has received significant practical attention for over three decades, its complexity has remained elusive. In this work, we address this lacuna, identifying sources of intractability and conditions under which the problem is efficiently solvable. Given a trace \( t \) of size \( n \) over \( k \) threads, our main results are as follows.

First, we establish a general \( O(k \cdot n^{2(k-1)}) \) upper-bound, as well as an \( O(n^6) \) upper-bound when certain parameters of \( t \) are constant. In addition, we show that the problem is NP-hard and even \( \text{W}[1] \)-hard when parameterized by \( k \), and thus unlikely to be fixed-parameter tractable. Second, we study the problem over acyclic communication topologies, such as pipelines and server-clients hierarchies. We establish an \( O(k^2 \cdot d \cdot n^2 \cdot \log n) \) upper-bound, where \( d \) is the number of shared variables accessed in \( t \). In addition, we show that even for traces from programs with \( k = 2 \) threads, there is no \( O(n^{7-\epsilon}) \) algorithm for the problem under the Orthogonal Vectors conjecture. Since any trace with 2-threads defines an acyclic topology, our upper-bound for acyclic topologies is (conditionally) optimal wrt polynomial improvements for up to moderate values of \( k \) and \( d \). Finally, motivated by existing heuristics, we study a distance-bounded version of the problem, where the task is to expose a data race by a witness trace that is similar to \( t \). We develop an algorithm that works in \( O(n) \) time when certain parameters of \( t \) are constant.

1 Introduction

A concurrent program is said to have a data race if it can exhibit an execution in which two accesses to the same memory location are “concurrent”. Data races in concurrent programs are often symptomatic of bugs in software like data corruption [6, 17, 24], pose challenges in defining the semantics of programming languages, and have led to serious problems in the past [39]; it is no surprise that data races have been deemed pure evil [7]. Automatically finding data races in programs remains a widely studied problem because of its critical importance in building correct concurrent software. Data race detection techniques can broadly be classified into static and dynamic. Given that the race-detection problem in programs is undecidable, static race detection approaches [23, 30] are typically conservative, produce false alarms, and do not scale to large software. On the other hand, since dynamic approaches [12, 21, 29, 33] have the more modest goal of discovering data races by analyzing a single trace, they are lightweight, and can often scale to production-level software. Moreover, many dynamic approaches are sound, i.e., do not raise false race reports. The effectiveness and scalability of dynamic approaches has lead to many practical advances on the topic. Despite a wide-spread interest on the problem, characterizing its complexity has remained elusive. Informally, the dynamic race prediction problem is the following: given an observed trace \( t \) of a multi-threaded program, determine if \( t \) demonstrates the presence of a data race in the program that generates \( t \). This means that either \( t \) has two conflicting data accesses \(^1\) that are concurrent, or a different trace resulting from scheduling the threads of \( t \) in a different order, witnesses such a race. Additional traces that result from alternate thread schedules are captured by the notion of a correct reordering of \( t \), that characterizes the set of all traces that can be exhibited by any program that can generate \( t \); a precise definition of correct reordering is given in Section 2.1. So formally, the data race prediction problem is, given a trace \( t \), determine if there is a correct reordering of \( t \) in which a pair of conflicting data accesses are concurrent.

While the data race prediction problem is clearly in \( \text{NP} \) – guess a correct reordering and check if it demonstrates a data race — its precise complexity has not been identified. Evidence based on prior work, suggests a belief that the problem might be \( \text{NP-complete} \). First, related problems, like data race detection for programs with strong synchronization primitives [25–27], or verifying sequential consistency [14], are known to be \( \text{NP-hard} \). Second, all known “complete” algorithms run in worst-case exponential time. These approaches either rely on an explicit enumeration of all correct reorderings [9, 34], or they are symbolic approaches that reduce the race predication problem to a constraint satisfaction problem [16, 32, 37]. On the other hand, a slew of

1\(^{\dagger}\)Two events are conflicting if they access the same memory location, with one of them being a write access.
"partial order"-based methods have been proposed whose goal is to predict data races in polynomial time, but at the cost of being incomplete and failing to detect data races in some traces. These include algorithms based on the classical happens-before partial order [12, 19–21, 35], and those based on newer partial orders that improve the prediction of data races over happens-before [13, 18, 28, 31, 36].

In this paper, we study the problem of data race prediction from a complexity-theoretic perspective. Our goal is to understand whether the problem is intractable, the causes for intractability, and conditions under which the problem can be solved efficiently. This paper provides partial answers to all these questions, and in some cases characterizes the tractability/intractability landscape precisely in the form of optimality results.

**Contributions.** Consider an input trace $t$ of size $n$ over $k$ threads. Our main contributions are as follows. We refer to Section 2.3 for a formal summary.

Our first result shows that the data-race prediction problem is solvable in $O(k \cdot n^{2(k-1)})$ time, and can be improved to $O(n^k)$ when certain additional parameters of $t$ are constant. We note that most benchmarks used to exhaustively evaluate practical techniques for race prediction have a constant number of threads [12, 13, 18, 20, 28, 31, 36], and in such cases our upper-bound is polynomial.

The observation that data race prediction is in polynomial time for constantly many threads naturally leads to two follow-up questions. Does the problem remain tractable for any $k$? And if not, is it fixed parameter tractable (FPT) wrt $k$, i.e., is there an algorithm with running time of the form $O(f(k) \cdot n^{O(1)})$? Our second result answers both these questions in the negative, by showing that the problem is $W[1]$-hard. This formally establishes the NP-hardness of the problem, and excludes efficient algorithms when $k$ is even moderately large (e.g., $k = \Omega(\log n)$).

We then investigate whether there are practically relevant contexts where data-race prediction is more efficiently solvable, i.e., the degree of the polynomial is fixed and independent of $k$. We consider the case of traces over acyclic communication topologies [2], such as pipelines and server-clients hierarchies. Our third result shows that, perhaps surprisingly, over such topologies data-race prediction can be solved in $O(k^2 \cdot d \cdot n^2 \cdot \log n)$ time, where $d$ is the total number of synchronization variables (locks) and global memory locations.

In practice, the size $n$ of the trace is by far the dominating parameter, while $k$ and $d$ are many orders of magnitude smaller. Hence, given the above upper-bound, the relevant question is whether the complexity on $n$ can be improved further. Our fourth result shows that this is unlikely: we show that, under the famous Orthogonal Vectors conjecture, there is no $O(n^{2-\epsilon})$ algorithm even for traces restricted to only 2 threads. As any trace with 2 threads induces an acyclic topology, our upper-bound is (conditionally) optimal wrt polynomial improvements.

Finally, the majority of practical data-race prediction heuristics search for a data race witness among correct reorderings that are very similar to the observed trace $t$, i.e., by only attempting a few event reorderings on $t$. Motivated by these approaches, we investigate the complexity of a distance-bounded version of data-race prediction, where the goal is to expose a data race by only looking at correct reorderings of $t$ that are a small distance away. Here, distance between traces is measured by the number of critical sections and write events whose order is reversed from $t$. Our fifth result is a linear-time (and thus, optimal) algorithm for this problem, when certain parameters of the trace $t$ are constant. This result gives a solid basis for the principled development of fast heuristics for dynamic data-race prediction.

**Technical contributions.** Towards our main results, we make several technical contributions that might be of independent interest. We summarize some of them below.

1. We improve the lower-bound of the well-known problem on verifying sequential consistency (VSC) [14] from the long-lasting NP-hardness to $W[1]$-hard.
2. We show that VSC can be solved efficiently on tree communication topologies of any number of threads, which improves a recent result of [28] for VSC for only 2 threads.
3. The first challenge that data-race prediction techniques face given a trace $t$, is to choose the set $X$ of events of $t$ over which to attempt to construct a correct reordering. Identifying such choices for $X$ has been proven a significant challenge in recent works [13, 16, 31]. For the first time, we establish non-trivial upper-bounds on the number of choices for $X$, and show that they are constantly many when certain parameters of $t$ are constant.
4. Particularly for tree communication topologies, we show that a single choice for such $X$ suffices.

**Related Work.** Antoni Mazurkiewicz [1, 22] used the notion of traces to mathematically model executions of concurrent programs and their allowable reorderings. Bertoni et. al. [4] studied various language-theoretic questions about Mazurkiewicz traces. The simplistic framework of Mazurkiewicz traces, however, cannot model the more general notion of correct reorderings we work with. The folklore results about the NP-hardness of race detection are often attributed to Netzer and Miller [25–27]. However, the problem considered in their work differs in significant ways from the problem of data-race prediction. First, the notion of feasible executions in [25] (the counterpart of the notion of correct reorderings) requires that any two conflicting events be ordered in the same way as the originally observed execution, and hence, are less permissive. Next, the NP-hardness arises from the use of complex synchronization primitives like wait.
and signal, which are more powerful than the primitives we study here (release/acquire of locks and read/write of registers). The results due to Netzer and Miller, thus, do not apply to the problem of data-race prediction and extending the result to the setting of our paper is non-trivial. Gibbons and Korach [14] establish NP-hardness for a closely related problem in distributed computing — verifying sequential consistency (VSC), which asks, given the projections of a trace onto individual threads, determine if there is a consistent total order that interleaves these projections in a consistent manner. Yet again, the problem is different (harder) than the problem of race prediction. In fact, when the input trace \( t \) comprises only of memory access events (for which VSC is shown to be NP-hard), then, \( t \) exhibits a predictable data race if it has a pair of conflicting events unordered by the HB [19] partial order, making the problem tractable in linear time. Complexity theoretic investigations have also been undertaken for other problems in distributed computing like linearizability [11, 15] or transactional consistency [5]. Hence, although there have been many theoretical results on related problems in concurrency, none of them addresses dynamic data-race prediction. Our work fills this gap.

Due to space restrictions, some proofs are relegated to the appendix.

## 2 Preliminaries

### 2.1 Model

**General notation.** Given a natural number \( k \), let \([k]\) denote the set \( \{1, \ldots, k\} \). Given a function \( f : X \to Y \), we let \( \text{dom}(f) = X \) and \( \text{img}(f) = Y \). Given two functions \( f, g \), we write \( f \subseteq g \) to denote that \( \text{dom}(f) \subseteq \text{dom}(g) \) and for every \( x \in \text{dom}(f) \) we have \( f(x) = g(x) \). Given a set \( X' \subseteq \text{dom}(f) \), we denote by \( f|X' \) the function with \( \text{dom}(f|X') = X' \) and \( f|X' \subseteq f \).

**Concurrent program.** We consider a shared-memory concurrent program \( P \) that consists of \( k \) threads \( \{p_i\}_{i \in [k]} \), under sequential consistency semantics. For simplicity of presentation we assume no thread is created dynamically and the set \( \{p_i\}_{i \in [k]} \) is known a-priori. All results presented here can be extended to a setting with dynamic thread creation. Communication between threads occurs over a set of global variables \( G \), and synchronization over a set of locks \( L \) such that \( G \cap L = \emptyset \). We let \( \mathcal{V} = G \cup L \) be the set of all variables of \( P \). Each thread is deterministic, and performs a sequence of operations on execution. We are only interested in the operations that access a global variable or a lock, which are called events. In particular, the allowed events are the following.

1. Given a global variable \( x \in G \), a thread can either write to \( x \) via an event \( w(x) \) or read from \( x \) via an event \( r(x) \).
2. Given a lock \( \ell \in L \), a thread can either acquire \( \ell \) via an event \( \text{acq}(\ell) \) or release \( \ell \) via an event \( \text{rel}(\ell) \).

Each such event is atomic. Given an event \( e \), we let \( \text{loc}(e) \) denote the global variable or lock that \( e \) accesses. We denote by \( \mathcal{W}_p \) (resp. \( R_p \), \( L^A_p \), \( L^R_p \)) the set of all write (resp. read, lock-acquire, lock-release) events that can be performed by thread \( p \). We let \( E_p = \mathcal{W}_p \cup R_p \cup L^A_p \cup L^R_p \) and assume that \( E_p \cap E_{p'} = \emptyset \) for every \( p \neq p' \). We denote by \( E = \bigcup_p E_p \), \( \mathcal{W} = \bigcup_p \mathcal{W}_p \), \( R = \bigcup_p R_p \), \( L^A = \bigcup_p L^A_p \), \( L^R = \bigcup_p L^R_p \) the events, write, read, lock-acquire and lock-release events of the program \( P \), respectively. Given an event \( e \in E \), we denote by \( p(e) \) the thread that \( e \) belongs to. Finally, given a set of events \( X \subseteq E \), we denote by \( \mathcal{R}(X) \) (resp., \( \mathcal{W}(X) \), \( L^A(X) \), \( L^R(X) \)) the set of read (resp., write, lock-acquire, lock-release) events of \( X \). For succinctness, we let \( \mathcal{W}(X) = \mathcal{W}(X \cup \mathcal{R}(X)) \), \( R_L(X) = \mathcal{R}(X) \cup L^R(X) \) and \( \mathcal{W}L(X) = \mathcal{W}(X) \cup L^A(X) \). The semantics of \( P \) are the standard for sequential consistency, and are omitted here for brevity.

**Conflicting events.** Given two distinct events \( e_1, e_2 \in E \), we say that \( e_1 \) and \( e_2 \) are conflicting, denoted by \( e_1 \bowtie e_2 \), if (i) \( \text{loc}(e_1) = \text{loc}(e_2) \) (i.e., both events access the same global variable or the same lock) and (ii) \( \exists e \in E \) such that \( e \bowtie e_1 \) and \( e \bowtie e_2 \). Yet again, the problem is different (harder) than the lock-acquire event. We extend the notion of conflict to sets events in the natural way: two sets of events \( X_1, X_2 \subseteq E \) are called conflicting, denoted by \( X_1 \bowtie X_2 \) if \( \exists (e_1, e_2) \in (X_1 \times X_2) \) such that \( e_1 \bowtie e_2 \).

**Event sequences.** Let \( t \) be a sequence of events. We denote by \( \mathcal{E}(t) \) the set of events, by \( \mathcal{L}(t) \) the set of locks, and by \( \mathcal{G}(t) \) the set of global variables in \( t \). We let \( \mathcal{W}(t) \) (resp., \( \mathcal{R}(t), L^A(t), L^R(t) \)) denote the set \( \mathcal{W}(\mathcal{E}(t)) \) (resp., \( \mathcal{R}(\mathcal{E}(t)), L^A(\mathcal{E}(t)), L^R(\mathcal{E}(t)) \)), i.e., it is the set of read (resp., write, lock-acquire, lock-release) events of \( t \). Given two distinct events \( e_1, e_2 \in \mathcal{E}(t) \), we say that \( e_1 \) is earlier than \( e_2 \) in \( t \), denoted by \( e_1 <_t e_2 \), iff \( e_1 \) appears before \( e_2 \) in \( t \). We say that \( e_1 \) is thread-ordered earlier than \( e_2 \), denoted \( e_1 <_{\text{TO}(t)} e_2 \), when \( e_1 <_t e_2 \) and \( p(e_1) = p(e_2) \). For events \( e_1, e_2 \in \mathcal{E}(t) \), we say \( e_1 \leq_t e_2 \) (resp. \( e_1 \leq_{\text{TO}(t)} e_2 \)) if either \( e_1 = e_2 \) or \( e_1 <_t e_2 \) (resp. \( e_1 <_{\text{TO}(t)} e_2 \)). We will often use \( <_{\text{TO}(t)} \) (resp. \( \leq_{\text{TO}(t)} \)) in place of \( <_{\text{TO}(t)} \) (resp. \( \leq_{\text{TO}(t)} \)) when the trace \( t \) is clear from context. Given a set of events \( X \subseteq E \), we denote by \( t|X \) the projection of \( t \) onto \( X \). Given a thread \( p_i \), we let \( t|p_i = t \cap \mathcal{E}(p_i) \). Finally, given two sequences \( t_1, t_2 \), we denote by \( t_1 \bowtie t_2 \) their concatenation.

**Lock events.** Given a sequence of events \( t \) and a lock-acquire event \( a \in L^A(t) \), we denote by \( \text{match}_t(a) \) the earliest lock-release event in \( L^R(t) \) such that \( rel \bowtie a \) and \( a <_t rel \), and let \( \text{match}_t(a) = \bot \) if no such lock-release event exists. If \( \text{match}_t(a) \neq \bot \), we require that \( p(a) = p(\text{match}_t(a)) \), i.e., the two lock events belong to the same thread. Similarly, given a lock-release event \( rel \in L^R(t) \), we denote by \( \text{match}_t(rel) \) the acquire event \( a \in L^A(t) \) such that \( \text{match}_t(a) = rel \) and require that such a lock-acquire event always exists. Given a lock-acquire event \( a \), the critical section \( CS_t(a) \) is the set of events \( e \) such that (i) \( a <_{\text{TO}} e \) and (ii) if \( \text{match}_t(a) \neq \bot \),
then \( e <_{\text{TO}} \text{match}_t(\text{acq}). \) For simplicity of presentation, we assume that locks are not reentrant. That is, for any two lock-acquire events \( \text{acq}_1, \text{acq}_2 \) with \( \text{acq}_1 \neq \text{acq}_2 \) and \( \text{acq}_1 <_{\text{TO}} \text{acq}_2, \) we must have \( \text{match}_t(\text{acq}_1) <_{\text{TO}} \text{acq}_2. \)
The lock-nesting depth of \( t \) is the maximum number \( \ell \) such that there exist distinct lock-acquire events \( \{\text{acq}_i\}_{i=1}^\ell \) with (i) \( \text{acq}_1 <_{\text{TO}} \text{acq}_2 <_{\text{TO}} \cdots <_{\text{TO}} \text{acq}_t \) and (ii) for all \( i \in [\ell], \) if \( \text{match}_t(\text{acq}_i) \in E(t) \) then \( \text{acq}_t <_{\text{TO}} \text{match}_t(\text{acq}_i). \)

**Traces and observation functions.** An event sequence \( t \) is called a trace if for any two lock-acquire events \( \text{acq}_1, \text{acq}_2 \in L^A(t), \) if \( \text{loc}(\text{acq}_1) = \text{loc}(\text{acq}_2) \) and \( \text{acq}_1 <_t \text{acq}_2, \) then \( \text{rel}_1 = \text{match}_t(\text{acq}_1) \in L^R(t) \) and \( \text{rel}_1 <_t \text{acq}_2. \) A trace therefore ensures that locks obey mutual exclusion, i.e., critical sections over the same lock cannot overlap. Given a trace \( t, \) we define its observation function \( O_t : \mathcal{R}(t) \to \mathcal{W}(t) \) as follows: \( O_t(t) = w \) if \( w <_t r \) and \( \forall w' \in \mathcal{W}(t) \) with \( w \equiv w', \) we have \( w' <_t r \Rightarrow r' <_t w. \) That is, \( O_t \) maps every read event \( r \) to the write event \( w \) that \( r \) observes in \( t. \) For simplicity, we assume that \( t \) starts with a write event to every location, hence \( O_t \) is well-defined. For notational convenience, we extend the observation function \( O_t \) to lock-release events, such that, for any lock-release event \( w \in L^R(t), \) we have \( O_t(w) = \text{match}_t(w), \) i.e., \( w \) observes its matching lock-acquire event.

**Correct reordering, enabled events and predictable data races.** A trace \( t' \) is a correct reordering of trace \( t \) if (i) \( E(t') \subseteq E(t), \) (ii) for every thread \( p_i, \) we have that \( t'_i|p_i \) is a prefix of \( t|p_i, \) and (iii) \( O_{t'} \subseteq O_t, \) i.e., the observation functions of \( t' \) and \( t \) agree on their common read and lock-release events. Given a trace \( t, \) event \( e \in E(t) \) and a correct reordering \( t' \) of \( t, \) we say that \( e \) is enabled in \( t' \) if \( e \not\in E(t'), \) and for every \( e' \in E(t) \) such that \( e' <_{\text{TO}} t \) \( e, \) we have that \( e' \in E(t'). \) Given two conflicting events \( e_1, e_2 \in E(t) \) with \( \text{loc}(e_1) = \text{loc}(e_2) \in \mathcal{G}, \) we say the pair \( (e_1, e_2) \) is a predictable data race of trace \( t \) if there is a correct reordering \( t' \) of \( t \) such that both \( e_1 \) and \( e_2 \) are enabled in \( t' \); in this case, we say that \( t' \) witnesses the data race \( (e_1, e_2). \) Finally, we say \( t \) has a predictable data race if there is a pair \( (e_1, e_2) \) which is a predictable data race of \( t. \)

**The communication topology.** The trace \( t \) naturally induces a communication topology graph \( G = (V,E), \) such that (i) \( V = \{p_i\}_{i=1}^n \) and (ii) \( E = \{(p_i,p_j) | i \neq j \in \mathbb{N} \land \text{acq}(t|p_i) \equiv \text{acq}(t|p_j)\}. \) In words, we have one node in \( G \) per program thread, and there is an edge between two distinct nodes if the corresponding threads execute events that are conflicting (note that \( G \) is undirected). For simplicity of presentation, we assume that \( G \) is connected. In later sections, we will make a distinction between tree topologies (i.e., that do not contain cycles) and general topologies (that might contain cycles). Common examples of tree topologies include stars (e.g., server-clients), pipelines, and the special case of two threads.

### 2.2 Problem Statement

In the dynamic data race prediction problem, we are given an observed trace \( t \), and the task is to identify whether \( t \) has a predictable data race. In this work we focus on the decision problem — given a trace \( t \) and two conflicting events \( e_1, e_2 \in E(t), \) and the task is to decide whether \( (e_1, e_2) \) is a predictable data race of \( t. \) Clearly, having established the complexity of the decision problem, the general problem can be solved by answering the decision problem for all \( O(n^2) \) pairs of conflicting variable access events of \( t. \) In the other direction, as the following lemma observes, detecting whether \( t \) has some predictable data race is no easier than detecting whether a given event pair of \( t \) constitutes a predictable data race. We refer to Appendix A for the proof.

**Lemma 2.1.** Given a trace \( t \) of length \( n \) and two events \( e_1, e_2 \in E(t), \) we can construct a trace \( t' \) in \( O(n) \) time so that \( t' \) has a predictable data race if \( (e_1, e_2) \) is a predictable data race of \( t. \)

To make the presentation simpler, we assume w.l.o.g that there are no open critical sections in \( t, \) i.e., every lock-acquire event \( \text{acq} \) is followed by a matching lock-release event \( \text{match}_t(\text{acq}). \) Motivated by practical applications, we also study the complexity of dynamic data race prediction parameterized by a notion of distance between the input trace \( t \) and the witness \( t' \) that reveals the data race.

**Trace distances.** Consider a trace \( t \) and a correct reordering \( t' \) of \( t, \) and let \( X = \mathcal{W}(t') \) be the set of write and lock-acquire events of \( t'. \) We define the set of reversals between \( t \) and \( t' \) as

\[
\text{Rv}(t, t') = \{(w_1, w_2) \in X \times X | w_1 \equiv w_2 \land w_1 <_t w_2 \land w_2 <_t w_1\}.
\]

In words, \( \text{Rv}(t, t') \) contains the pairs of write and lock-acquire events, the order of which has been reversed in \( t' \) when compared to \( t. \) The **distance of \( t' \) from \( t \) is defined as** \( \delta(t, t') = |\text{Rv}(t, t')|\). Our notion of distance, thus, only counts the number of reversals of conflicting write or lock-acquire events instead of counting reversals over all events (or even conflicting write-read events).

**Distance-bounded dynamic data race prediction.** Consider a trace \( t \) and two events \( e_1, e_2 \) of \( t. \) Given an integer \( \ell \geq 0, \) the \( \ell \)-distance-bounded dynamic data race prediction problem asks for an algorithm with the following properties.

1. If the algorithm outputs True, then \( (e_1, e_2) \) is a predictable data race of \( t. \)
2. If the algorithm outputs False, then either \( (e_1, e_2) \) is not a predictable data race of \( t, \) or any correct reordering \( t' \) that witnesses the data race is such that \( \delta(t, t') > \ell. \)

### 2.3 Summary of Main Results

Here we state the main results of this work, and present the technical details in the later parts of the paper.
2.3.1 The General Case

First, we study the algorithmic complexity of the problem as a function of various parameters of the input trace. These parameters are the number of threads, the number of variables, the lock-nesting depth, as well as the lock-dependence factor, which, intuitively, measures the amount of data flow between critical sections.

The lock-dependence factor. The lock-dependence graph of a trace $t$ is the graph $G_t = (V_t, E_t)$ defined as follows.

1. The set of vertices is $V_t = L^A(t)$, i.e., it is the set of lock-acquire events of $t$.
2. The set of edges is such that $(\text{acq}_1, \text{acq}_2) \in E_t$ if (i) $\text{acq}_1 \not<_{\text{TOO}} \text{acq}_2$, (ii) $\text{acq}_1 <_{\text{TOO}} \text{match}_t(\text{acq}_2)$, and (iii) $\text{match}_t(\text{acq}_1) \not<_{\text{TOO}} \text{match}_t(\text{acq}_2)$.

Here, $<_{\text{TOO}}$ (defined formally in Section 3) is the smallest partial order that contains $<_T$, and also orders read events after their corresponding observed write event (i.e., $O(r) <_{\text{TOO}} r$ for every $r \in R(L(E(t)))$). Given a lock-acquire event $\text{acq} \in V_t$, let $A_{\text{acq}}$ be the set of lock-acquire events that can reach $\text{acq}$ in $G_t$. We define the lock dependence factor of $t$ as $\max_{\text{acq} \in V_t} |A_{\text{acq}}|$. We show the following theorem.

**Theorem 2.2.** Consider a trace $t$ of length $n$, $k$ threads, $d$ variables, lock-nesting depth $\gamma$, and lock-dependence factor $\zeta$. The dynamic data race prediction problem on $t$ can be solved in $O(\alpha \cdot \beta)$ time, where $\alpha = \min(n, k \cdot \gamma \cdot \zeta)^{k-2}$ and $\beta = k \cdot \min(n^k, n^{d+1})$.

In particular, the problem is polynomial-time solvable for a fixed number of threads $k$. In practice, the parameters $k$, $\gamma$ and $\zeta$ typically behave as constants, and in such cases our upper-bound becomes $O(n^k)$. Theorem 2.2 naturally leads to two questions, namely (i) whether there is a polynomial-time algorithm for any $k$, and (ii) if not, whether the problem is FPT wrt $k$, i.e., having complexity $O(f(k) \cdot n^{O(1)})$, for some computable function $f$. Note that question (ii) is very relevant, as typically $k$ is several orders of magnitude smaller than $n$. We complement Theorem 2.3 with the following lower-bound, which answers both questions in negative.

**Theorem 2.3.** The dynamic data race prediction problem is $W[1]$-hard parameterized by the number of threads.

2.3.2 Tree Communication Topologies

Next, we study the problem for tree communication topologies, such as pipelines and server-clients architectures. We show the following theorem.

**Theorem 2.4.** Let $t$ be a trace over a tree communication topology with $n$ events, $k$ threads and $d$ variables. The dynamic data race prediction problem for $t$ can be solved in $O(k^2 \cdot d \cdot n^2 \cdot \log n)$ time.

Perhaps surprisingly, in sharp contrast to Theorem 2.3, for tree topologies there exists an efficient algorithm where the degree of the polynomial is fixed and does not depend on any input parameter (e.g., number of threads). Note that the dominating factor in this complexity is $n^2$, while $k$ and $d$ are typically much smaller. Hence, the relevant theoretical question is whether the dependency on $n$ can be improved further. We show that this is unlikely, by complementing Theorem 2.4 with the following conditional lower-bound, based on the Orthogonal Vectors conjecture [8].

**Theorem 2.5.** Let $t$ be a trace with $n$ events, $k \geq 2$ threads and $d \geq 8$ shared variables with at least one lock. There is no algorithm that solves the decision problem of dynamic data race prediction for $t$ in time $O(n^{2-\epsilon})$, for any $\epsilon > 0$, unless the Orthogonal Vectors conjecture fails.

Since $k = 2$ implies a tree communication topology, the result of Theorem 2.4 is conditionally optimal, up-to poly-logarithmic factors, for a reasonable number of threads and variables (e.g., when $k, d = \log^{O(1)}(n)$).

2.3.3 Witnesses in Small Distance

Finally, we study the problem under more practical scenarios, namely, when (i) the number of threads, lock-nesting depth, lock-dependence factor of $t$ are bounded, and (ii) we are searching for a witness at a small distance from $t$. We show the following theorem.

**Theorem 2.6.** Fix a reversal bound $\ell \geq 0$. Consider a trace $t$ of length $n$ and constant number of threads, lock-nesting depth and lock-dependence factor. The $\ell$-distance-bounded dynamic data race prediction problem for $t$ can be solved in $O(n) \cdot time$.

3 Trace Ideals

3.1 Partial Orders

**Partially ordered sets.** A partially ordered set (or poset) is a pair $(X, P)$ where $X$ is a set of (write, read, lock-acquire, lock-release) events and $P$ is a reflexive, antisymmetric and transitive relation over $X$. We often use the notation $\leq$ in place of $P$ and write $e_1 \leq e_2$ to denote $(e_1, e_2) \in P$. Given two events $e_1, e_2 \in X$ we write $e_1 < e_2$ to denote that $e_1 \leq e_2$ and $e_1 \neq e_2$, and we write $e_1 \ll e_2$ to denote that $e_1 < e_2$ and there exists no event $e$ such that $e_1 < e < e_2$. Given two distinct events $e_1, e_2 \in X$, we say that $e_1$ and $e_2$ are unordered by $P$, denoted by $e_1 \parallel e_2$, if neither $e_1 < e_2$ nor $e_2 < e_1$. We call an event $e$ in $X$ maximal (resp., minimal) if there exists no other event $e' \in X$ such that $e < e'$ (resp., $e' < e$). Given a set $Y \subseteq X$, we denote by $P[Y]$ the projection of $P$ on $Y$, i.e., we have $\leq_{P[Y]} \subseteq Y \times Y$, and for all $e_1, e_2 \in Y$, we have $e_1 \leq_{P[Y]} e_2$ iff $e_1 \leq e_2$. Given two posets $(X, P)$ and $(Y, Q)$, we say that the partial order $Q$ refines $P$, denoted by $Q \sqsubseteq P$, if for every pair of events $e_1, e_2 \in X$, if $e_1 \leq e_2$ then $e_1 \leq Q e_2$. If $Q$ refines $P$, we say that $P$ is weaker than $Q$. We denote by $Q \sqsubseteq P$ the fact that $Q \subseteq P$ and $P \not\sqsubseteq Q$. A linearization of $(X, P)$ is a total order over $X$ that refines $P$. An order ideal (or simply ideal) of a poset $(X, P)$ is a subset $Y \subseteq X$ such that for every two events $e_1 \in Y$ and $e_2 \in X$ with $e_2 \leq P e_1$, we have $e_2 \in Y$. An event $e$ is executable in $Y$ if $Y \cup \{e\}$ is also an ideal of $(X, P)$.
Partially ordered sets with observations. A poset with observations (or o-poset) is a tuple \((X, P, O)\) where
1. \(O : RL(X) \rightarrow W(\mathcal{L}(X))\) is an observation function such that for all \(e \in RL(X)\), we have \(O(e) \in W(\mathcal{L}(X))\), and
2. \((X, P)\) is a poset such that for all \(e \in RL(X)\) we have \(O(e) \subseteq P\). Notation from posets is naturally lifted to o-posets, e.g., an ideal of \(P\) is an ideal of \((X, P)\).

Thread-observation order and trace ideals. Given a trace \(t\), the thread-observation order \(TO(t)\) (or simply \(TO\) when \(t\) is clear from context) is the weakest partial order over the set \(E(t)\) such that (i) \(\subseteq_{TO} \subseteq\) \(TO\), and (ii) \((E(t), TO, O_t)\) is an o-poset. In particular, we have \(TO = \{\top \cup \{O_t(e) < e' | e \in WR(t)\}\}^*\). A trace ideal of \(t\) is an ideal \(X\) of the poset \((E(t), TO)\). We say an event \(e \in E(t) \setminus X\) is enabled in \(X\) if for every \(e' \in X\), we have \(e' < e\). We call \(X\) lock-feasible if for every pair of lock-acquire events \(acq_1, acq_2 \in \mathcal{L}^A(X)\) with \(acq_1 \triangleright eq acq_2\), we have \(matchi(acq_1) \subseteq X\) for some \(i \in \{2\}\). We call \(X\) feasible if it is lock-feasible, and there exists a partial order \(P\) over \(X\) such that (i) \(P \subseteq TO(X)\) and (ii) for every pair of lock-acquire events \(acq_1, acq_2 \in \mathcal{L}^A(X)\) with \(acq_1 \triangleright eq acq_2\) and \(matchi(acq_2) \neq X\), we have \(rel_2 \prec acq_1\), where \(rel_2 \triangleright eq acq_1\). If \(X\) is feasible, we define the canonical o-poset of \(X\) as \((X, Q, O_t)\), where \(Q\) is the weakest among all such partial orders \(P\). It is easy to see that \(Q\) is well-defined, i.e., there exists at most one weakest partial order among all such partial orders \(P\).

The realizability problem of feasible trace ideals. The realizability problem for an o-poset \(P = (X, P, O)\) asks whether there exists a linearization \(t\) of \(P\) such that \(O = O_t\). Given a trace \(t\) and a feasible trace ideal \(X\) of \(t\), the realizability problem for \(X\) is the realizability problem of the canonical o-poset \((X, P, O)\) of \(X\). The following remark relates the decision problem of dynamic race prediction in \(t\) with the realizability of trace ideals of \(t\).

**Remark 1.** If \(t^*\) is a witness of the realizability of \(X\) then \(t^*\) is a correct reordering of \(t\). Two conflicting events \(e_1, e_2 \in E(t)\) are a predictable data race of \(t\) iff there exists a realizable trace ideal \(X\) of \(t\) such that \(e_1, e_2\) are enabled in \(X\).

Read pairs and triplets. For notational convenience, we introduce the notion of read pairs and read triplets. Given an o-poset \(P = (X, P, O)\), a read pair (or pair for short) of \(P\) is a pair \((w, w')\) such that \(r \in RL(X)\) and \(w = O(r)\) (note that \(w \in X\)). A read triplet (or triplet for short) is a triplet \((w, r, w')\) such that (i) \((w, r)\) is a pair of \(X\), (ii) \(w' \prec w\), and (iii) \(w' \neq w\). We denote by \(Pairs(P)\) and \(Triplets(P)\) the set of pairs and triplets of \(P\), respectively.

Closed o-posets. We call an o-poset \(P = (X, P, O)\) closed if for every triplet \((w, r, w') \in \text{Triplets}(P)\), we have (i) if \(w' \prec w\) then \(w' \subseteq P\) \(w\), and (ii) if \(w \prec w'\) then \(r \sqsubseteq P\).

Given an o-poset \(P = (X, P, O)\), the closure of \(P\) is an o-poset \(Q = (X, Q, O)\) where \(Q\) is the weakest partial order over \(X\) such that \(Q \subseteq P\) and \(Q\) is closed. If no such \(Q\) exists, we let the closure of \(P\) be \(\bot\). The closure is well-defined [28]. The associated Closure problem is, given an o-poset \(P\), to decide whether the closure of \(P\) is \(\bot\).

**Remark 2.** An o-poset is realizable only if its closure exists and is realizable.

### 3.2 Bounds on the Number of Feasible Trace Ideals

As Remark 1 states, the dynamic data race prediction problem for a trace \(t\) is reducible to deciding whether \(t\) has some realizable trace ideal. In general, if \(t\) has length \(n\) and \(k\) threads, there exist \(n^k\) possible trace ideals to test for realizability. Here we derive another upper-bound on the number of such ideals that are sufficient to test, based on the number of threads of \(t\), its lock-nesting depth, the lock-dependence factor. These parameters typically behave as constants in practice, and thus understanding the complexity of dynamic data race prediction in terms of these parameters is crucial.

Causal cones. Given an event \(e \in E(t)\), the causal cone \(Cone_e(S)\) of \(e\) is the smallest trace ideal \(X\) of \(t\) so that \(e\) is enabled in \(X\). Given a set of events \(S \subseteq E(t)\), we define the casual cone of \(S\) as \(Cone_e(S) = \bigcup_{e \in S} Cone_e(e)\).

Candidate ideal set. Given a set of events \(X\), we denote by \(OpenAcqs(X)\) the set of lock-acquire events \(acq\) such that \(match_r(acq) \not\subseteq X\). Given two events \(e_1, e_2 \in E(t)\), the candidate ideal set \(CIS_t(e_1, e_2)\) of \(e_1, e_2\) is the smallest set of trace ideals of \(t\) such that the following hold.

1. \(Cone_e((e_1, e_2)) \in CIS_t(e_1, e_2)\).
2. Let \(Y \in CIS_t(e_1, e_2)\), \(acq \in \text{OpenAcqs}(Y)\), \(rel = match_r(acq)\), and \(Y' = Cone_e(Y \cup \{rel\}) \cup \{rel\}\). If \(e_1, e_2 \not\subseteq Y'\), then \(Y' \in CIS_t(e_1, e_2)\).

As the following lemma shows, in order to decide whether \((e_1, e_2)\) is a predictable data race of \(t\), it suffices to test for realizability of all the ideals in \(CIS_t(e_1, e_2)\).

**Lemma 3.1.** \((e_1, e_2)\) is a predictable data race of \(t\) iff there exists a realizable ideal \(X \in CIS_t(e_1, e_2)\) such that \(e_1, e_2 \not\subseteq X\).

The following lemma gives an upper-bound on \(|CIS_t(e_1, e_2)|\).

**Lemma 3.2.** We have \(|CIS_t(e_1, e_2)| \leq \min(n, \alpha)^{k-2}\), where \(\alpha = k \cdot \gamma \cdot \zeta\), and \(k\) is the number of threads, \(\gamma\) is the lock-nesting depth, and \(\zeta\) is the lock-dependence factor of \(t\).

### 4 The General Case

In this section we address the general case of dynamic data-race prediction. The section is organized in two parts, which present the formal details of Theorem 2.2 and Theorem 2.3.

#### 4.1 Upper Bound

In this section we establish Theorem 2.2. Recall that, by Lemma 3.1, the problem is reducible to detecting a realizable o-poset in the candidate ideal set of the two events that are tested for a data-race. O-poset realizability is known to be NP-complete [14], and solvable in polynomial time when the
number of threads is bounded [2]. Here we establish more precise upper-bounds, based on the number of threads and the number of variables. In particular, we show the following.

**Lemma 4.1.** O-poset realizability can be solved in $O(\beta)$ time, where $\beta = k \cdot \min(n^k, n^{d+1})$, for an o-poset of size $n$, $k$ threads and $d$ variables.

**Frontiers and $\sigma$-extensions.** Let $P = (X, P, O)$ be an o-poset, and consider an ideal $Y$ of $P$. The frontier of $Y$, denoted $\text{Frontier}_P(Y)$, is the set of pairs $(w, r) \in \text{Pairs}(P)$ such that $w \in Y$ and $r \notin Y$. In the following, we fix an arbitrary total order $\sigma$ on $X$ such that every read or lock-release event appears before every write or lock-acquire in $X$. An event $e$ executable in $Y$ is said to extend $Y$ if for every triplet $(w, r, e) \in \text{Triplets}(P)$, we have $(w, t) \notin \text{Frontier}_P(Y)$. We say that $e$ $\sigma$-extends $Y$ if, in addition, the following hold.

1. If $e \in \mathcal{R}_L(X)$ then $t \leq_{\sigma} e'$ for any $e'$ that extends $Y$.
2. If $e \in \mathcal{W}_L(X)$ then for any event $e'$ that extends $Y$, $e' \in \mathcal{W}_L(X)$.

In this case, we say that $Y \cup \{e\}$ is a $\sigma$-extension of $Y$ via $e$.

**Ideal graphs and canonical traces.** Let $P = (X, P, O)$ be an o-poset. The ideal graph of $P$, denoted $G_P = (\mathcal{V}_P, \mathcal{E}_P)$, is a directed graph defined as follows.

1. $\mathcal{V}_P$ is the set of ideals of $P$.
2. We have $(Y_1, Y_2) \in \mathcal{E}_P$ iff $Y_2$ is a $\sigma$-extension of $Y_1$.

The ideal tree of $P$, denoted $T_P = (I_P, \mathcal{R}_P)$, is an arbitrary spanning tree of $G_P$ restricted to the nodes of $G_P$ reachable by $\emptyset$. Let $\emptyset$ be the root of $T_P$. Given an ideal $Y \in I_P$, we define the canonical trace $t_Y$ of $Y$ inductively, as follows. If $Y = \emptyset$ then $t_Y = \epsilon$. Otherwise, $Y$ has a parent $Y'$ in $T_P$ such that $Y = Y' \cup \{e\}$ for some event $e \in X$. We define $t_Y = t_{Y'} \circ e$. Lemma 4.1 relies on the following lemmas. We refer to Appendix C.1 for the proofs.

**Lemma 4.2.** We have $X \in I_P$ iff $P$ is realizable.

**Lemma 4.3.** The ideal graph $G_P$ has $O(\min(n^k, n^{d+1}))$ nodes.

Using Lemma 4.1, we can prove Theorem 2.2.

**Proof of Theorem 2.2.** Consider the problem on $t$ for two conflicting events $e_1, e_2 \in \mathcal{W}_R(t)$. By Lemma 3.1, to decide whether $(e_1, e_2)$ is a predictable data race of $t$, it suffices to iterate over all feasible trace ideals $X$ in the candidate ideal set $\text{CIS}_I(e_1, e_2)$, and test whether any such $X$ is realizable. By Lemma 3.2, we have $|\text{CIS}_I(e_1, e_2)| = O(a)$, where $a = \min(n, k \cdot \gamma)$ $k^{-2}$. Finally, due to Lemma 4.1, the realizability of such an ideal can be performed in $O(\beta)$ time. □

### 4.2 Hardness of Data Race Prediction

Here we establish that the problem of dynamic data race prediction is $\mathcal{W}[1]$-hard when parameterized by the number of threads $k$. Our proof is established in two steps. In the first step, we show the following lemma.

**Lemma 4.4.** O-poset realizability parameterized by the number of threads $k$ is $\mathcal{W}[1]$-hard.

O-poset realizability is known to be NP-hard [14], and Lemma 4.4 strengthens the result of [14] by showing that the problem is even unlikely to be FPT. In the second step, we show how the class of $\mathcal{W}[1]$-hard instances constructed in Lemma 4.4 can be reduced to dynamic data-race prediction.

**Hardness of o-poset realizability.** Our reduction is from the INDEPENDENT-SET($c$) problem, which takes as input an undirected graph $G = (V, E)$ and asks whether $G$ has an independent set of size $c$. INDEPENDENT-SET($c$) parameterized by $c$ is one of the canonical $\mathcal{W}[1]$-hard problems [10].

Given an input $G = (V, E)$ of the INDEPENDENT-SET($c$) problem, we construct an o-poset $P_G = (X, P, O)$ of size $O(c \cdot n)$ and $O(c)$ threads such that $P_G$ is realizable iff $G$ has an independent set of size $c$. We assume wlog that every node in $G$ has at least one neighbor, otherwise, we can remove all such nodes and solve the problem for parameter $c' = c - s$. The o-poset $P_G$ consists of $k = 2 \cdot c + 2$ total orders $(X_i, \tau_i)$. Figure 1 provides an illustration. In high level, for each $i \in [c]$, $\tau_i$ and $\tau_{i+1}$ are used to encode the $i$-th copy of $G$, whereas the last two total orders are auxiliary. Superscripts on the events and/or their variables refer to the node of $G$ that is encoded by those events. Below we describe the events and certain orderings between them. The partial order $P$ is taken to be the transitive closure of these orderings.

1. For $i = 2 \cdot c + 1$, $\tau_i$ consists of a single event $\tau_i = w(x)$.
2. For $i = 2 \cdot c + 2$, we have $\tau_i = \sigma \circ \tau_i$, where
   $$\sigma = r(s_1), \ldots, r(s_n), acq(\ell_1), \ldots, acq(\ell_c) \text{ and } \tau_i \circ \tau_i = r(x), rel(\ell_c), \ldots, rel(\ell_1).$$
3. For each $i \in [c]$, we have $\tau_i = t_i = o \tau_i \circ \tau_i \circ \ldots \tau_i$, where each $\tau_i$ encodes node $j$ of $G$ and is defined as follows. Let
   $$\tau_i = \sigma_i \circ \tau_i,$$
   where
   $$\sigma_i = acq(\ell_j, l_1), \ldots, acq(\ell_j, l_m) \text{ and } \tau_i = rel(\ell_j, l_1), \ldots, rel(\ell_j, l_m).$$
   where $l_1, \ldots, l_m$ are the neighbors of $j$ in $G$. For each $j \in [n] \setminus \{1, n\}$, the sequence $\tau_i$ is identical to $\tau_i$, with the addition that the innermost critical section (i.e., between $acq(\ell_j, l_m)$ and $rel(\ell_j, l_1)$) contains the sequence $w(y_i^j), r(x_i^j)$. The sequence $\tau_i$ is defined similarly, except that the innermost critical section contains the sequence $w(s_i), r(x_i^j)$. Finally, the sequence $\tau_i$ is defined similarly, except that the innermost critical section contains the sequence $w(y_i^j), r(x_i^j)$.
4. For each $i \in [c]$, we have $\tau_i = \tau_i = t_i \circ \tau_i \circ \ldots \tau_i$, where $\tau_i$ is defined similarly, hence the observation function $O$ is defined implicitly. In addition, for every read event $r$, we have $O(r) <_P r_i(x)$ for each $i \in [c]$.
Finally, we turn our attention to the hardness of dynamic data-race prediction. Consider the INDEPENDENT-SET(\ell) problem on a graph \( G \) and the associated o-poset \( \mathcal{P}_G = (X, P, O) \) defined above. We construct a trace \( t \) with \( E(t) = X \) and \( O_t = O \) such that \((w(x), r(x))\) is a predictable data-race of \( t \) if \( \mathcal{P}_G \) is realizable. In particular, \( t \) consists of \( 2 \cdot c + 2 \) threads \( p_i \), one for each total order \( \tau_i \) of \( \mathcal{P}_G \). We obtain \( t \) as

\[
T = \tau_{2,c+1} \circ \tau_1 \circ \ldots \circ \tau_c \circ \tau_{2,c+2},
\]

where each \( \tau_i \) is an appropriate interleaving of the total orders \( \tau_i \) and \( \tau_{c+i} \) that respects the observation function \( O \). It is straightforward to verify that \( t \) exists, and \( t \) is a valid trace. In Appendix C.2, we conclude the proof of Theorem 2.3 by showing that \( G \) has an independent set of size \( c \) if \( (w(x), r(x)) \) is a predictable data-race of \( t \).

5 Tree Communication Topologies

In this section we focus on the case where the input trace \( t \) constitutes a tree communication topology. The section is organized in two parts, which present the formal details of Theorem 2.4 and Theorem 2.5.

5.1 An Efficient Algorithm for Tree Topologies

In this section we present the formal details of Theorem 2.4. Recall that the Theorem 2.2 states an \( O(\alpha \cdot \beta) \) upper-bound of dynamic data-race prediction, where \( \beta \) is the complexity of deciding o-poset realizability, and \( \alpha \) is an upper-bound on the number of candidate ideal sets on which we need to decide realizability. For tree communication topologies, we obtain Theorem 2.4: (i) we show an improved upper-bound \( \beta \) on the complexity of the realizability of trace ideals over tree topologies, and (ii) we show that a single trace ideal needs to be tested for realizability (hence \( \alpha = 1 \)). We start with point (i), and then proceed with (ii).

Tree-inducible o-posets. Let \((X, P)\) be a poset where \( X \subseteq E(t). \) We call \((X, P)\) tree-inducible if \( X \) can be partitioned into \( k \) sets \( \{X_i\}_{1 \leq i \leq k} \) such that the following conditions hold.

1. The graph \( T = ([k], \{(i, j) | X_i \cap X_j \neq \emptyset\}) \) is a tree.
2. \( P|X_i \) is a total order for each \( i \in [k] \).
3. For every node \( \ell \in [k] \) such that \( \ell \) is an internal node in \( T \), for every two connected components \( C_1, C_2 \) of \( T \) that are created after removing \( \ell \) from \( T \), consider any two nodes \( i \in C_1 \) and \( j \in C_2 \). For every two events \( e_1 \in X_i \) and \( e_2 \in X_j \) such that \( e_1 < P e_2 \), there exists some event \( e' \in X'_\ell \) such that \( e_1 < P \ell e < P \ell e' \).

We call an o-poset \((X, P, O)\) tree-inducible if the poset \((X, P)\) is tree-inducible. Our motivation behind tree-inducible posets comes from the following lemma.
Lemma 5.1. O-poset realizability of tree-inducible o-poses can be solved in $O(k^2 \cdot d \cdot n^2 \cdot \log n)$ time, for an o-poset of size $n$, $k$ threads and $d$ variables.

The proof of Lemma 5.1 is in two steps. Recall the definition of closed o-poses from Section 3.1. First, we show that a tree-inducible, closed o-poset is realizable (Lemma 5.2). Second, we show that the closure of a tree-inducible o-poset is also tree-inducible (Lemma 5.3).

**Lemma 5.2.** Every closed, tree-inducible o-poset is realizable.

Indeed, consider a tree-inducible, closed o-poset $\mathcal{P} = (X, P, O)$. The witness $t$ realizing $\mathcal{P}$ is obtained in two steps.

1. We construct a poset $(X, Q)$ with $Q \subseteq P$ as follows. Initially, we let $Q$ be identical to $P$. Let $T = ([k], \{(i, j) \mid X_i \neq X_j\})$ be a rooted tree such that $P$ is tree-inducible to $T$. We traverse $T$ top-down, and for every node $i$ and child $j$ of $i$, for every two events $e_1 \in X_i$ and $e_2 \in X_j$ such that $e_1 \neq e_2$ and $e_2 \neq P e_1$, we order $e_1 <_Q e_2$. Finally, we transitively close $Q$.

2. We construct $t$ by linearizing $(X, Q)$ arbitrarily.

In Appendix D.1 we show that $t$ is well-defined and realizes $\mathcal{P}$. The next lemma shows that tree-inducibility is preserved under taking closures (if the closure exists).

**Lemma 5.3.** Consider an o-poset $\mathcal{P} = (X, P, O)$ and let $Q = (X, Q, O)$ be the closure of $\mathcal{P}$. If $\mathcal{P}$ is tree-inducible then $Q$ is also tree-inducible.

Since, by Remark 2, an o-poset is realizable only if its closure exists and is realizable, Lemma 5.2 and Lemma 5.3 allow to establish realizability of o-poses of size $n$, $k$ threads and $d$ variables.

**Lock causal cones.** Consider a trace $t$ that defines a tree communication topology $G_t = (V_t, E_t)$. Given an event $e \in E(t)$ the lock causal cone $LCone_t(e)$ of $e$ is the set $X$ defined by the following process. Consider that $G_t$ is rooted in $p(e)$.

1. Initially $X$ contains all predecessors of $e$ in $(E(t), TO)$.

2. We perform a top-down traversal of $G_t$, and consider a current thread $p_t$ visited by the traversal.

3. While there exists some lock-acquire event $acq_1 \in X[p_t]$ and there exists another lock-acquire event $acq_2 \in \text{OpenAcqqs}(X)$ with $acq_1 \neq acq_2$ and $p(acq_1) = p_t$, we insert in $X$ all predecessors of $rel_1$ in $(E(t), TO)$ (including $rel_1$), where $rel_1 = \text{match}_t(acq_1)$.

Observe that, by construction, $LCone_t(e)$ is a lock-feasible trace ideal of $t$. In addition, for any two events $e_1, e_2 \in E(t)$, the set $LCone_t(e_1) \cup LCone_t(e_2)$ is an ideal of $t$, though not necessarily lock-feasible. Our motivation behind lock causal cones comes from the following lemma.

**Lemma 5.4.** Let $X = LCone_t(e_1) \cup LCone_t(e_2)$. We have that $(e_1, e_2) \Rightarrow (i) (e_1, e_2) \cap X = \emptyset$, and (ii) $X$ is a realizable trace ideal of $t$.

The $(\Rightarrow)$ direction of the lemma is straightforward. We refer to Appendix D.1 for the $(\Leftarrow)$ direction. Finally, Theorem 2.4 follows immediately from Lemma 5.1 and Lemma 5.4.

**Proof of Theorem 2.4.** By Lemma 5.4, we have that $(e_1, e_2)$ is a predictable data race of $t$ if and only if $(e_1, e_2) \cap X = \emptyset$ and $X$ is realizable. By Lemma 5.1, deciding the realizability of $X$ is done in $O(k^2 \cdot n^2 \cdot \log n)$ time. The desired result follows.

### 5.2 A Lower Bound for Two Threads

In this section we prove a conditional quadratic lower bound on the complexity of dynamic data race prediction for two threads. Our proof is by a reduction from the Orthogonal Vectors problem, which has a long-standing quadratic worst-case upper-bound. To make it conceptually simpler, we present our reduction in two steps. First, we show a fine-grained reduction from the problem of Orthogonal Vectors to the problem of computing whether an o-poset with 2 threads and 7 variables is realizable. Afterwards, we show how the realizability problem for the o-poses constructed in the first step can be reduced to the decision problem of dynamic data race prediction for 2 threads, 8 variables and 1 lock.

**The Orthogonal Vectors problem (OV).** An instance of Orthogonal Vectors consists of two sets $A, B$, where each set contains $n/2$ binary vectors in $D$ dimensions. The task is to determine whether there exists a pair of vectors $(a, b) \in A \times B$ that is orthogonal, i.e., for all $i \in [D]$ we have $a[i] \cdot b[i] = 0$. There exist simple algorithms that solve the problem in $O(n^2 \cdot D)$ and $O(2^D \cdot n)$ time, simply by computing the inner product of each pair $(a, b) \in A \times B$ and following a classic Four-Russians technique, respectively. It is conjectured that there is no truly sub-quadratic algorithm for OV [8].

**Conjecture 5.5 (Orthogonal Vectors).** There is no algorithm for OV that operates in $O(n^{2-\epsilon} \cdot D^{O(1)})$ time, for any $\epsilon > 0$.

It is also known that the Strong Exponential Time Hypothesis (SETH) implies the OV conjecture [38]. We first show the following lemma, which relates OV with o-poset realizability.

**Lemma 5.6.** O-poset realizability for and o-poset with 2 threads and at least 7 variables has no $O(n^{2-\epsilon})$-time algorithm for any $\epsilon > 0$, under the Orthogonal Vectors conjecture.

**Reduction from OV to o-poset realizability.** For a fine-grained reduction from OV to o-poset realizability, consider an OV instance $(A, B)$, where $A = (a_j)_{1 \leq j \leq n/2}$, $B =$
Formal construction. We now present the formal details of the construction, which is also illustrated in Figure 2. The construction creates various events which have the form \( e^a_i \) and \( e^b_i \) when they are used at the vector level, and have the form \( e^{a_i}_i \) and \( e^{b_i}_i \), where \( i \in [D] \) is some coordinate, when they are used at the coordinate level. As a general rule, for each \( j, l \in [n/2 - 1] \), we have \( e^{a_i}_j <_{\tau_A} e^{a_i}_{j+1} \) and \( e^{b_i}_j <_{\tau_B} e^{b_i}_{j+1} \), both for events at the vector and at the coordinate level. To make the presentation more succint, we often write \( e_1, e_2 < e_3 \) to denote \( e_1 < e_2 < e_3 \). We next describe the events, and various orderings between them. The partial order \( P \) consists of the transitive closure of these orderings.

**Events on \( x_1 \) and \( x_2 \).** For every vector \( a \in A \) and coordinate \( i \in [D] \), we create three events \( w^a_i(x_1) \), \( w^a_i(x_2) \) and \( \tau^a_i(x_2) \). We make \( O(\tau^a_i(x_2)) = w^a_i(x_2) \), and order

\[
\tau^a_i(x_2) <_{\tau_A} w^a_i(x_1), w^a_i(x_2).
\]

For every vector \( b \in B \) and coordinate \( i \in [D] \), we create three events \( w^b_i(x_1) \), \( w^b_i(x_2) \) and \( \tau^b_i(x_2) \). We make \( O(\tau^b_i(x_2)) = w^b_i(x_2) \), and order

\[
w^b_i(x_1), w^b_i(x_2) <_{\tau_B} \tau^b_i(x_1).
\]

In addition, we order

\[
w^a_i(x_2) <_{\tau_A} w^a_i(x_1) \iff a_j[i] = 1 \quad \text{and} \quad w^b_i(x_2) <_{\tau_B} w^b_i(x_1) \iff b_j[i] = 1.
\]

Observe that if \( a_j[i] \cdot b_j[i] = 1 \) and the closure orders \( w^a_i(x_1) < w^a_i(x_2) \), then we also have \( w^b_i(x_1) < w^b_i(x_2) \) and hence by closure \( \tau^a_i(x_2) < \tau^a_i(x_1) \).

**Events on \( x_3 \).** Let \( i \in [D - 1] \) be a coordinate. For every vector \( a_j \in A \), we create an event \( w^a_{i+1}(x_3) \), and order

\[
w^a_{i+1}(x_3) <_{\tau_A} w^a_{i+1}(x_3) <_{\tau_A} a_j(x_1), w^a_{i+1}(x_2).
\]

For every vector \( b_j \in B \), we create two events \( w^b_{i+1}(x_3) \) and \( \tau^b_{i+1}(x_3) \), and make \( O(\tau^b_{i+1}(x_3)) = w^b_{i+1}(x_3) \). In addition, we order

\[
w^b_{i+1}(x_3), w^b_{i+1}(x_3) <_{\tau_B} w^b_{i+1}(x_3) <_{\tau_B} b_j(x_1), b_j(x_2).
\]

Observe that if the closure orders \( w^a_i(x_1) < w^b_i(x_1) \) then we also have \( w^a_{i+1}(x_3) < \tau^b_{i+1}(x_3) \), hence by closure \( w^a_{i+1}(x_3) < w^b_{i+1}(x_3) \) and thus \( w^a_{i+1}(x_3) < \tau^b_{i+1}(x_3) \).

**Events on \( x_6 \).** For every coordinate \( i \in [D - 1] \), we do as follows. For every vector \( a_j \in A \), we create two events \( w^a_{i+1}(x_6) \)

---

![Figure 2. Illustration of the reduction of an OV instance \((A = \{a_1, a_2\}, B = \{b_1, b_2\})\) to the Closure problem of an \( \mathcal{P} \).](image-url)
and \( r_i^{a_j}(x_6) \), and make \( O(r_i^{a_j}(x_6)) = w_i^{a_j}(x_6) \). We order
\[
\begin{align*}
\text{and } w_i^{a_j}(x_6) < r_{x_7} w_{i+1}^{a_j}(x_6) < r_{x_7} r_i^{a_j}(x_2) \quad \text{and}
\end{align*}
\]
for every vector \( b_j \in B \), we create one event \( w_i^{b_j}(x_6) \), and order
\[
\begin{align*}
\text{for every vector } b_j \in B, \text{ we create one event } w_i^{b_j}(x_6), \text{ and order }
\end{align*}
\]
and make two orderings across \( \tau_A \) and \( \tau_B \), namely
\[
\begin{align*}
\text{and make two orderings across } \tau_A \text{ and } \tau_B, \text{ namely}
\end{align*}
\]
Correctness. Observe that we have used 7 variables, while \( |X_A| + |X_B| = O(n \cdot D) \). We refer to Appendix D.2 for the full proofs of the correctness of the above construction. This concludes Lemma 5.6, as any algorithm for o-poset realizability on the above instances that runs in \( O(n \cdot D)^{2-\varepsilon} \) time also solves \( OV \) in \( O(n^{2-\varepsilon} \cdot D^{O(1)}) \) time. Although the full proof is rather technical, the correctness is conceptually straightforward. Consider a closure computation on \( P \), that starts with processing the ordering \( w_i^{a_j}(x_1) < r_i^{b_j}(x_1) \). The closure will strengthen \( P \) with an ordering \( w_i^{a_j}(x_1) < w_i^{b_j}(x_1) \), which transitively leads to \( w_i^{a_j}(x_1) < r_i^{b_j}(x_1) \). Again, the closure will strengthen \( P \) with an ordering \( w_i^{a_j}(x_1) < w_i^{b_j}(x_1) \), which transitively leads to \( w_i^{a_j}(x_1) < r_i^{b_j}(x_1) \). Performing this process inductively, we end up with \( w_i^{a_j}(x_1) < w_i^{b_j}(x_1) \), for each \( i \in [D] \). If there is some \( i \) with \( a_i[i] \cdot b_i = 1 \), we will have \( w_i^{a_j}(x_1) < w_i^{b_j}(x_1) \), the closure will strengthen \( P \) with an ordering \( r_i^{a_j}(x_1) < w_i^{b_j}(x_1) \). A similar induction on the events on variables \( x_2 \) and \( x_3 \) propagates these orderings “downwards” and eventually leads to \( r_i^{a_j}(x_1) < w_i^{b_j}(x_1) \), which signifies that \( a_i \) is not orthogonal to \( b_i \). (And only then), we have ordered \( w_i^{a_j}(x_1) < b_i(x_1) \), and the closure will strengthen \( P \) with an ordering \( w_i^{a_j}(x_1) < w_i^{b_j}(x_1) \). However, this leads transitively to \( w_i^{a_j}(x_1) < r_i^{b_j}(x_1) \), which signifies the start of testing for orthogonality \( a_i \) against \( b_i \), by repeating the previous process. If (and only if) the process reaches \( b_i n/2 \) and \( a_i \) is found not orthogonal to \( b_i n/2 \), a similar reasoning on events on variables \( x_2 \) and \( x_3 \) leads to an ordering \( w_i^{a_j}(x_1) < r_i^{b_j}(x_1) \), which signifies the start of testing for orthogonality \( a_i \) against \( b_i \), by repeating the above process. Finally, if (and only if) the above process reaches vectors \( a_i n/2 \) and \( b_i n/2 \), and they are found not orthogonal, the closure will end up with an ordering \( w_i^{a_j}(x_1) < w_i^{b_j}(x_1) \). However, notice that \( w_i^{a_j}(x_1) < w_i^{b_j}(x_1) \), and hence a cycle is formed, thus the closure of \( P \) does not exist. On the other hand, if the closure process finds an orthogonal pair along the way, the cross ordering \( w_i^{a_j}(x_1) < w_i^{b_j}(x_1) \) is not contradicted, and the closure of \( P \) has been computed.

Reduction to dynamic data-race prediction. Consider an instance of the o-poset \( P = (X, P, O) \) constructed in the above reduction, and we construct a trace \( t \) and two events \( e_t, e_2 \in E(t) \) such that \( P \) is realizable iff \( (e_t, e_2) \) is a predictable data race of \( t \). The trace \( t \) consists of two threads \( P_A, P_B \) and two local traces \( \tau_A \) and \( \tau_B \) such that \( \tau_A \) and \( \tau_B \) contain the events of \( P_A \) and \( P_B \), respectively. Each of \( \tau_A \) and \( \tau_B \) is identical to \( r_{x_7} \) and \( r_{x_7} \), respectively, with some additional events inserted in it. In particular, besides the variables \( x_i \), \( i \in [7] \) that appear in the events of \( X \), we introduce one
variable $y$ and one lock $\ell$. For the event set, we have
\[
E(t) = \{X \cup \{w_1(y), r_1(y)\} \cup \{acq_A(\ell), rel_A(\ell)\} \cup \{acq_B(\ell), rel_B(\ell)\} \cup \{w_2(y), r_2(y)\}\}.
\]
The local traces $t'_A$ and $t'_B$ are constructed as follows.
1. For $t'_A$, we insert an empty critical section $acq_A(\ell), rel_A(\ell)$ right after $w^{n/2}_1(x_1)$. Additionally, we insert the read event $r_1(y)$ right before $t'_1^{n/2}(x_2)$, and the read event $r_2(y)$ as the last event of $t'_B$.
2. For $t'_B$, we insert the write event $w_1(y)$ right after $w^{n/2}_1(x_2)$. Additionally, we insert $w_2(y)$ right after $t'_1^{n/2}(x_1)$, and surround these two events with $acq_B(\ell), rel_B(\ell)$.

Finally, we obtain $t$ as $t = t'_B \circ t'_A$, i.e., the two local traces are executed sequentially and there is no context switching. The task is to decide whether $(w_2(y), r_2(y))$ is a predictable data race of $t$. We refer to Appendix D.2 for the correctness of the construction, which concludes Theorem 2.5.

6 Witnesses in Small Distance

The results in the previous sections completely neglect information provided by the input trace $t$ about constructing a correct reordering that witnesses the data race. Indeed, our hardness results show that, in the worst case, the orderings in $t$ provide no help. However, practical observations show that typically when a data race exists, a witness trace $t^*$ can be constructed that is similar to $t$. In fact, virtually all practical techniques predict data races by constructing $t^*$ to be very similar to $t$ (e.g., [12, 18, 28, 31, 36]).

The distance-bounded realizability problem of feasible trace ideals. Given a natural number $\ell$, a trace $t$ and a feasible trace ideal $X$ of $t$, the $\ell$-distance-bounded realizability problem for $X$ asks for an algorithm that outputs True/False and has the following properties.
1. If the algorithm outputs True, then $X$ is realizable.
2. If the algorithm outputs False, then either $X$ is not realizable, or any witness $t^*$ that realizes $X$ is such that $\delta(t, t^*) > \ell$.

As before, we are interested in the case where $\ell = O(1)$. There exists a straightforward algorithm that operates in $O(|X|^2 \ell)$ time. The algorithm iterates over all possible subsets of pairs of conflicting write and lock-acquire events that have size at most $\ell$, and tries all possible combinations of conflicting-write reversals in that set. Theorem 2.6 is based on the following lemma, which states that the problem can be solved much faster when $k$ is constant.

Lemma 6.1. Consider given a natural number $\ell$, a trace $t$ over $n$ events and $k$ threads, and a feasible trace ideal $X$ of $t$. The $\ell$-distance-bounded realizability problem for $X$ can be solved in $O(k^{\ell+O(1)} \cdot n)$ time.

Using Lemma 6.1, we can prove Theorem 2.6.

Proof of Theorem 2.6. By Lemma 6.1, given a trace ideal $X$ of $t$, we can solve the $\ell$-distance-bounded realizability problem for $X$ in $O(n)$ time. The proof then follows easily by Lemma 3.1 and Lemma 3.2, as to decide whether $(e_1, e_2)$ is a predictable data race of $t$, it suffices to examine $O(1)$ many trace ideals of $t$. □

In the remaining of this section we prove Lemma 6.1. We first define the notion of read extensions of graphs. Afterwards, we present the algorithm for the lemma, and show its correctness and complexity.

Read extensions. Consider a digraph $G = (X, E)$. Given two events $e_1, e_2 \in E$, we write $e_1 \rightarrow e_2$ to denote that $e_2$ is reachable from $e_1$. We call $G$ write-ordered if for every two distinct conflicting write or lock-acquire events $w_1, w_2 \in W^L(X)$, we have $w_1 \rightarrow w_2$ or $w_2 \rightarrow w_1$ in $G$. Given an acyclic write-ordered graph $G_1 = (X, E_1)$, the read extension of $G_1$ is the digraph $G_2 = (X, E_2)$ where $E_2 = E_1 \cup A \cup B$, where the sets $A$ and $B$ are defined as follows.

\[A = \{(r, w) \in RL \times W^L | r \bowtie w \text{ and } (O_t(r), w) \in E\},\]
\[B = \{(w, r) \in W^L \times RL | r \bowtie w \text{ and } (w, O_t(r)) \in E\} \]

A fast algorithm for distance-bounded o-poset realizability. Let $P = (X, P, O)$ be the canonical o-poset of $X$, and the task is to decide the realizability of $P$ with $\ell$ reversals. We describe a recursive algorithm for solving the problem for some o-poset $Q = (X, Q, O)$ with $\ell'$ reversals, for some $\ell' \leq \ell$, where initially $Q = P$ and $\ell' = \ell$.

Algorithm and correctness. Consider the set
\[C = \{(w_1, w_2) \in W^L \times W^L | w_1 \bowtie w_2 \text{ and } w_1 \parallel_{Q} w_2 \leq w_2\} \]

We construct a graph $G_1 = (X, E_1)$, where $E_1 = (TOO(X)) \cup C$. Note that $G_1$ is write-ordered. If it is acyclic, we construct the read extension $G_2$ of $G_1$. Observe that if $G_1$ is acyclic then any linearization $t^*$ of $G$ realizes $Q$, hence we are done. Now consider that either $G_2$ or $G_3$ is not acyclic, and let $G = G_1$ if $G_1$ is not acyclic, otherwise $G = G_2$. Given a cycle $C$ of $G$, represented as a collection of edges, define the set of cross-edges of $C$ as $C \setminus Q$. Note that, since there are $k$ threads, $G$ has a cycle with $\leq k$ cross edges. In addition, any trace $t^*$ that realizes $Q$ must linearize an o-poset $(X, Q, O)$ where $a = (e_1, e_2)$ ranges over the cross-edges of $C$. In particular, we take $Q_a = Q \cup \{b\}$, where
\[b = \{(e_2, e_1), \text{ if } a \in W^L \times W^L, (O_2(e_2), e_1), \text{ if } a \in W^L \times RL, (e_2, O_1(e_1)), \text{ if } a \in RL \times W^L\}.\]

Observe that any such choice of $b$ reverses the order of two conflicting write events or lock-acquire events in $t$. Since there are $\leq k$ cross edges in $C$, there are $\leq k$ such choices for $Q_a$. Repeating the same process recursively for the o-poset $(X, Q_a, O)$ for $\ell' - 1$ levels solves the $\ell'$-distance-bounded
realizability problem for $Q$. Since initially $t' = t$ and $Q = P$, this process solves the same problem for $P$ and thus for $X$.

**Complexity.** The recursion tree above has branching $\leq k$ and depth $\leq t'$, hence there will be at most $k^{t'}$ recursive instances. In Appendix E, we provide some lower-level algorithmic details which show that each instance can be solved in $O(k^{O(1)} \cdot n)$ time. The main idea is that each of the graphs $G_1$ and $G_2$ have a sparse transitive reduction [3] of size $O(k \cdot n)$, and thus each graph can be analyzed in $O(k \cdot n)$ time.

7 Conclusion

In this work, we have studied the complexity of dynamic data-race prediction, and have drawn a rich complexity landscape depending on various parameters of the input trace. Our main results indicate that the problem is in polynomial time when the number of threads is bounded, however, it is unlikely to be FPT wrt this parameter. On the other hand, we have shown that the problem can be solved in, essentially, quadratic time, when the communication topology is acyclic.

We have also proved a conditional quadratic lower bound for this case, which shows that our algorithm for tree communication topologies is optimal. Finally, motivated by practical techniques, we have shown that a distance-bounded version of data-race prediction can be solved in linear time under mild assumptions on the input parameters.

References


A Details of Section 2

In this section we provide the proof of Lemma 2.1.

Lemma 2.1. Given a trace t of length n and two events e1, e2 ∈ E(t), we can construct a trace t′ in O(n) time so that t′ has a predictable data race iff (e1, e2) is a predictable data race of t.

Proof. We outline the construction of t′. We introduce two locks ℓ1, ℓ2, and for each i ∈ [2], we surround e1 with the lock ℓ, i.e., we replace e1 with acq(ℓ1), e1, rel(ℓ1). For every other event e ∈ E(t) \ {e1, e2}, we replace e with acq(ℓ1), acq(ℓ2), e, rel(ℓ2), rel(ℓ1).

It is easy to see that (e1, e2) can be the only predictable data race of t′, and any correct reordering of t that witnesses the data race (e1, e2) can be transformed to a correct reordering of t′ that witnesses the same data race, and vice versa.

The desired result follows. □

B Details of Section 3

Here we prove Lemma 3.1 and Lemma 3.2.

Lemma 3.1. (e1, e2) is a predictable data race of t iff there exists a realizable ideal X ∈ CISi(e1, e2) such that e1, e2 ∉ X.

Proof. The (⇐) direction of the statement is straightforward, and here we focus on the (⇒) direction. Let t′ be a correct reordering that witnesses (e1, e2), and X′ = E(t′). We show that there exists an ideal X ∈ CISi(e1, e2) such that (i) X ⊆ X′, and (ii) OpenAcqs(X) ⊆ OpenAcqs(X′). Observe that the two conditions imply the lemma: (i) since X ⊆ X′, we have that e1, e2 ∉ X, while both events are enabled in X, and (ii) since X ⊆ X′ and OpenAcqs(X) ⊆ OpenAcqs(X′), we have that t′ | X is a correct reordering of t, and hence X is realizable.

Consider any ideal Y ∈ CISi(e1, e2) such that Y ⊆ X′. Clearly, at least one such Y exists, by taking Y = Conei((e1, e2)) and noticing that Y ⊆ X′. If OpenAcqs(Y) ⊈ OpenAcqs(X′), there exists some lock-acquire event acq ∈ OpenAcqs(Y) such that rel ∈ X′, where rel = matchi(acq). But then Y′ ∈ CISi(e1, e2), where Y′ = Y ∪ Conei{[rel]} ∪ {rel}. Note that Y′ ⊆ X′, and repeat the process for Y′ = Y′'. Since Y ⊆ Y′, this process can be repeated at most n times, thus at some point we have chosen an ideal Y ∈ CISi(e1, e2) with the desired properties of X.

The desired result follows. □

Lemma 3.2. We have |CISi(e1, e2)| ≤ min(n, ℓ)^k-2, where α = k · γ · ζ, and k is the number of threads, γ is the lock-nesting depth, and ζ is the lock-dependence factor of t.

Proof. Let Z = CISi(e1, e2) and X = Conei((e1, e2)). For an ideal Y ∈ Z \ {X}, let relY be the lock-release event that lead to Y according to item 2 of the definition of CISi(e1, e2), and acqY = matchi(rel). In addition, we call the ideal Y′ ∈ Z with acqY ∈ OpenAcqs(Y′) that lead to adding Y ∈ Z the parent of Y. We define inductively AX = ∅, and Ay = Ay′ ∪ {acqY}, where Y′ is the parent of Y. Note that every ideal Y ∈ Z is uniquely characterized by Ay.

Let Gt be the lock-dependence graph of t. We show by induction that for every Y ∈ Z \ {X} there exists a lock-acquire event acq ∈ OpenAcqs(X) such that acq is reachable from acqY in Gt. Let Yi be the smallest (wrt set inclusion) ancestor of Y such that acqY ∈ OpenAcqs(Yi). The statement holds if Y1 = X, by taking acq = acqY. Otherwise, let Y2 be the parent of Yi, and we have acqY1 ∈ OpenAcqs(Y2). Note that (i) acqY1 < TOO acqY2 (since acqY1 \ Y2), (ii) acqY2 < TOO relYi (since acqY2 ∈ Conei(relYi)), and (iii) relY1 < TOO relYi (since relY1 \ Y1). It follows that (acqY1, acqY2) is an edge in Gi. By the induction hypothesis, we have that there exists some lock-acquire event acq ∈ OpenAcqs(X) that is reachable from acqY1. Hence acq is reachable by acqY in Gi, as desired.

Now, let A = ∪y∈Z Ay, and by the previous paragraph, for every lock-acquire event acq′ ∈ A, there exists a lock-acquire event acq ∈ OpenAcqs(X) that is reachable from acq′ in Gt. Hence, |A| ≤ |OpenAcqs(X) · ζ|. In addition, since there are k threads and the lock-nesting depth is γ, we have |OpenAcqs(X)| ≤ k · γ · ζ = min(n, α)k-2 such ideals.

The desired result follows. □

C Details of Section 4

C.1 Details of Section 4.1

Here we provide formal proofs to Lemma 4.2, Lemma 4.3 and, using these, Lemma 4.4.

Lemma 4.2. We have X ∈ IP iff P is realizable.

Proof. We prove each direction separately.

(⇒). We prove by induction that every canonical trace tY of TIP realizes Y. The statement clearly holds if Y = e. Otherwise, let tY = tY · r, where Y′ is the σ-extension of Y by the event r, and by the induction hypothesis we have that tY′ realizes Y. The statement clearly holds if e ∈ W ′L(t), so we focus on the case where e ∈ R ′L(t).

Consider the pair (w, e) ∈ Pairs(X), and note that w < TOO e and thus w ∈ Y′. It remains to argue that for every triplet (w, e, w′) ∈ Triplets(P), if w′ ∈ Y′ then w′ < tyr w. Assume towards contradiction otherwise, thus there exist two ancestors Y1, Y2 ∈ Y′ such that (i) Y2 = Y1 \ Y′ and (ii) w ∈ Y1. In that case we have (w, r) ∈ FrontierP(Y1), hence w′ could not have been executable in Y1, a contradiction.

(⇐). Let t be a witness trace that realizes P. We argue that for every prefix t′ of t, there exists a node X′ ∈ IP such
that (i) $\mathcal{RL}(t') \subseteq \mathcal{RL}(X')$ and (ii) $\mathcal{WL}(t') = \mathcal{WL}(X')$. The proof is by induction on the prefixes $t'$. The statement clearly holds if $t' = e$, by taking $X'$ to be the root of $t_P$. Otherwise, let $t' = t''e$, for some event $e \in X$, and by the induction hypothesis, there exists an ideal $X'' \in \mathit{IP}$ such that $\mathcal{RL}(t'') \subseteq \mathcal{RL}(X'')$ and $\mathcal{WL}(t'') = \mathcal{WL}(X'')$. If $e \in X'$ we are done. Otherwise, we proceed as follows.

First, we argue that $e$ extends $X''$. The statement is trivial if $e \in \mathcal{RL}(X)$, hence we focus on the case where $e \notin \mathcal{WL}(X)$. It suffices to argue that for every pair $(w, r) \in \mathit{Frontier}_P(Y'')$ we have $(w, r, e) \notin \text{Triplets}(P)$. Consider any such triplet, and since $t'$ is a witness of the realizability of $X$, we have $r \in \mathcal{E}(t')$. By the induction hypothesis, we have $w \in \mathcal{E}(t'')$ and $r \notin \mathcal{E}(t'')$. But then $O_{t'}(r) \neq w$, a contradiction.

Second, we argue that $X''$ has a sequence of successors $X_1, \ldots, X_t$, with $X_1 = X''$, such that, for each $i \in [t]$, if $e$ extends $X_i$, but does not $\sigma$-extend $X_i$, then (a) $e$ extends $X_{i+1}$, and (b) $X_{i+1} \setminus X'' \subseteq \mathcal{RL}(X)$. Since we have already argued that $e$ extends $X_t$ and there are many $X_i$ there exists some $\ell \in [t]$ such that $e$ extends $X_{\ell}$, and thus we can take $X' = X_{\ell} \cup \{e\}$, which yields that $\mathcal{RL}(t') \subseteq \mathcal{RL}(X')$ and $\mathcal{WL}(t') = \mathcal{WL}(X')$, as desired. Indeed, if $e$ does not $\sigma$-extend $X_t$, then there exists a read event or lock-release event $e' \in \mathcal{RL}(X)$ that $\sigma$-extends $X_t$. Thus $X_t$ has a successor $X_{t+1}$ such that (a) $\mathit{Frontier}_P(X_{t+1}) \subseteq \mathit{Frontier}_P(X_t)$ and thus $e$ also extends $X_{t+1}$, while (b) $X_{t+1} \setminus X'' \subseteq \mathcal{RL}(X)$ since $e' \in \mathcal{RL}(X)$ while by the induction hypothesis $X_t \setminus X'' \subseteq \mathcal{RL}(X)$.

The desired result follows.

\begin{lemma}
The ideal graph $G_P$ has $O(\min(n^k, n^{d+1}))$ nodes.
\end{lemma}

\begin{proof}
Since there are $k$ threads, the bound $n^k$ follows straightforwardly. For the bound $n^{d+1}$, associate each ideal $Y$ a function $f_Y : [d] \rightarrow \mathcal{WL}(X) \cup \{\bot\}$ that maps each of the $d$ memory locations to the last write or lock-acquire event of the canonical trace $t_P$ that modifies that location (where $\bot$ encodes no such event). Since there are $\leq n$ read and lock-release events, each function $[d] \rightarrow \mathcal{WL}(X)$ is associated with $\leq n$ ideals $Y$. Hence there are $\leq n \cdot (n + 1)^d = O(n^{d+1})$ nodes in $G_P$.

The desired result follows.
\end{proof}

We are now ready to prove Lemma 4.1.

\begin{lemma}
O-poset realizability can be solved in $O(\beta)$ time, where $\beta = k \cdot \min(n^k, n^{d+1})$, for an o-poset of size $n$, $k$ threads and $d$ variables.
\end{lemma}

\begin{proof}
Consider an o-poset $P$ of size $n$, $k$ threads and $d$ variables. By Lemma 4.2, to decide the realizability of $P$ it suffices to construct the ideal graph $G_P$ and test whether $X \in \mathcal{VP}$. By Lemma 4.3, $G_P$ has $O(\min(n^k, n^{d+1}))$ nodes, and since $P$ has $k$ threads, there each node of $G_X$ has $\leq k$ outgoing edges. Hence $G_X$ can be constructed in $O(\beta)$ time.

The desired result follows.
\end{proof}

\subsection{C.2 Details of Section 4.2}

Here we prove formally Lemma 4.4 and, using this, Theorem 2.3.

\begin{lemma}
O-poset realizability parameterized by the number of threads $k$ is $W[1]$-hard.
\end{lemma}

\begin{proof}
We show that $P_G$ is realizable if and only if $G$ has an independent set of size $c$. We prove each direction separately.

($\Rightarrow$) Let $t$ be a trace that realizes $P_G$. For each $i \in [c]$, let $m_i$ be the maximum integer $j$ such that the event $r(z_i^j)$ has a predecessor $r_i$ in $t$, with $e_i < r(x)$. Note that for each such $i$ we have $w(s_i) < r(x)$, and since $w(s_i)$ is the predecessor of $r(z_i^j)$ in $t$, the index $m_i$ is well-defined. We argue that $A = \{l_i\}_{i \in [c]}$ is an independent set of $G$, where $l_i = m_i$ if $r(x) < r(z_i^{m_i})$ and $l_i = m_i + 1$ otherwise. We note in the second case, $l_i = n$, as, otherwise, we would also have $\text{acq}_i(t_i) < r(x)$, and since $r(x)$ is protected by lock $l_i$, we would also have that $\text{rel}_i(t_i) < r(x)$. But then, $w(y_i^{l_i+1}) < r(x)$, which would contradict our choice of $m_i$.

Indeed, consider any distinct $i_1, i_2 \in [c]$ and any neighbor $v_1$ and $v_2$ of $m_i$ and $m_i$, respectively. Note that our choice of $A = \{l_i\}_{i \in [c]}$ implies that both $\text{acq}_i(t_i) < r(x)$ and $\text{acq}_j(t_j) < r(x)$. It suffices to argue that both $r(x) < r(z_i^{m_i})$ and $r(x) < r(z_j^{m_j})$. As then we have that $(l_i, l_j) \notin E$, which concludes that $A$ is an independent set of $G$.

Assume towards contradiction that $\text{rel}_i(t_i) < r(x)$. Clearly $l_i < n$, as $r(x) < r(z_i^{m_i})$. But then, $\text{acq}_i(t_i) < r(z_i^{m_i})$ and thus $\text{acq}_i(t_i) < r(x)$. Since $r(x)$ appears in a critical section on lock $l_i$, we have $\text{rel}_i(t_i) < r(x)$. But then $e_i < r(x)$, where $e_i$ is the predecessor of $r(z_i^{m_i+1})$ in $t_i$, which contradicts our definition of $l_i$. Hence, $r(x) < \text{rel}_i(t_i)$, $r(x) < \text{rel}_j(t_j)$, and thus $A$ is an independent set of $G$.

($\Leftarrow$) Let $A$ be an independent set of $G$ of size $c$, and $l_1, \ldots, l_c$ some arbitrary ordering of $A$. For each $i \in [c]$ let $Y_i$ be the following events of threads $i$ and $c + i$.

1. All strict predecessors of $e_i$ in $t_i$, where $e_i = r(x_i^{l_i})$ if $l_i < n$ and $e_i = r_i(x)$ otherwise.

2. If $l_i > 1$, all predecessors of $\text{rel}_i^{l-1}(t_i)$ in $t_{c+i}$ (including $\text{rel}_i^{l-1}(t_i)$).

For each $i \in [c]$, let $Z_i = \{e \in Y_i : \exists \text{acq}_i \in \text{OpenAcqs}(Y_i) \text{ s.t. } \text{acq}_i \leq e\}$

i.e., $Z_i$ contains all events of $t_i$ and $t_{c+i}$ that succeed some lock-acquire event that is open in $Y_i$. We argue that for any two distinct $i_1, i_2 \in [c]$, any two lock-acquire events $\text{acq}_i \in Z_i$ and $\text{acq}_j \in Z_j$, access a different lock. Note that the statement follows easily if one $\text{acq}_i$ or $\text{acq}_j$ belongs to $t_{c+i}$, for some $i \in [c]$, as the locks accessed by each such total
ordered are only also accessed by $τ_{2c+2}$. Hence, we focus on the case where each of $acq_i$ and $acq_{i+1}$ belong to $τ_i$ for some $i \in [c]$. Assume towards contradiction otherwise, hence there exist distinct $i_1, i_2 \in [c]$ such that (i) $acq_{i_1}(τ_{i_1}, t_{i_1}) \in Z_{i_1}$ and $acq_{i_2}(τ_{i_2}, t_{i_2}) \in Z_{i_2}$, while (ii) $rel_{i_1}(τ_{i_1}, t_{i_1}) \notin Z_{i_1}$ and $rel_{i_2}(τ_{i_2}, t_{i_2}) \notin Z_{i_2}$. It follows that $(i_1, i_2) \in E$, which contradicts the fact that $A$ is an independent set of $G$.

We now construct a trace $t$ that realizes $P_G$ in five phases, where initially, we have $t = ε$.

1. **Phase 1:** For each $i \in [c]$, we linearize the partial order $P|\left(Y_i \setminus Z_i\right)$ arbitrarily, and append it to $t$.
2. **Phase 2:** For each $i \in [c]$, we linearize the partial order $P|Z_i$ arbitrarily, and append it to $t$.
3. **Phase 3:** We append to $t$ the sequence $t_1 \circ w(x), r(x) \circ t_2$, where $t_1$ and $t_2$ are sequences over the events of $τ_{2c+2}$, as follows.

   \[ t_1 = r(s_1), \ldots, r(s_c), acq(ℓ_1), \ldots, acq(ℓ_c) \]  

   \[ t_2 = rel(ℓ_1), \ldots, rel(ℓ_c) \]  

4. **Phase 4:** For each $i \in [c]$, we linearize the partial order $P|S_i \setminus Y_i$, where $S_i$ is the smallest ideal of $(X, P)$ that contains the matching lock-release events of all lock-acquire events that are open in $Y_i$.
5. **Phase 5:** For each $i \in [c]$ we linearize $P$ over the remaining events of $τ_i$ and $τ_{c+i}$ arbitrarily, and append them to $t$.

Finally, we argue that $t$ is a valid witness trace. It is straightforward to verify that $t$ is a linearization of $P$. Moreover, since every memory location is written exactly once in $X$, for every read event $r \in R(X)$ we have $O(r) = O(t)$. It remains to argue that $t$ respects the critical sections of $P_G$. Observe that the only phase in which we interleave open critical sections between total orders $τ_i$ that access the same lock is in Phase 2. However, as we have shown, for every two distinct $i_1, i_2 \in [c]$, any two lock-acquire events $acq_{i_1} \in Z_{i_1}$ and $acq_{i_2} \in Z_{i_2}$ access a different lock. It follows that $t$ respects the critical sections of $P_G$.

The desired result follows. \qed

We are now ready to prove Theorem 2.3.

**Theorem 2.3.** The dynamic data race prediction problem is $W[1]$-hard parameterized by the number of threads.

**Proof.** We show that $(w(x), r(x))$ is a predictable data-race of $t$ iff $P_G$ is realizable. If $P_G$ is realizable, then $(w(x), r(x))$ is a predictable data-race of $t$ witnessed by the witness $t^*$ of the realizability of $P_G$, as constructed in the proof of Lemma 4.4 (direction $\Leftarrow$).

For the inverse direction, let $t^*$ be a correct reordering of $t$ that witnesses the data race $(w(x), r(x))$. We construct a trace $t^*$ that realizes $P_G$ as

\[ t^* = t^* \circ w(x), r(x), rel(ℓ_1), \ldots, rel(ℓ_c) \circ t_4 \circ t_5, \]

where $t_4$ and $t_5$ are analogous to the linearizations in phase 4 and Phase 5, respectively, of the construction in the proof of Lemma 4.4. In particular, let $t''$ be the prefix of $t'$ until the event $rel(t_i)$.

1. We construct $t_1$ as follows. For each $i \in [c]$, we linearize the set $S_i \setminus E(t'')$ and append it to $t_1$, where $S_i$ is the smallest ideal of $X$ that contains the matching lock-release events of all lock-acquire events that are open in $t''$.
2. We construct $t_2$ as follows. For each $i \in [c]$ we linearize the remaining events of $τ_i$ and $τ_{c+i}$ arbitrarily, and append them to $t_2$.

The correctness is established similarly to the proof of Lemma 4.4.

**D Details of Section 5**

**D.1 Details of Section 5.1**

Here we provide details of Section 5.1. We start with the proof of Lemma 5.2. First, consider the construction of witness trace $t$, by linearizing the poset $(X, Q)$. We have the following lemma, which states that $(X, Q)$ is well-defined.

**Lemma D.1.** $(X, Q)$ is a poset.

**Proof.** Assume towards contradiction otherwise. Consider the process in which we insert the orderings in $Q$ in sequence, and examine the first pair of conflicting events $(e_1, e_2)$ such that we try to order $e_1 \prec_Q e_2$ whereas it already holds that $e_2 \prec_Q e_1$. For $j \in \{2\}$, let $i_j$ be such that $e_j \in X_{i_j}$, and it follows that $i_j$ is the parent of $i_{j-1}$ in $T$. Observe that, by the construction of $Q$, we have $e_1 \parallel_P e_2$. Consider the sequence of orderings

\[ e_2 = \overline{e}_1 \prec_Q \overline{e}_2 \prec_Q \ldots \prec_Q \overline{e}_l = e_1 \]

that witnesses (transitively) that $e_2 \prec_Q e_1$. Since $P$ is tree-inducible, there exists some $j \in \{l\}$ such that $\overline{e}_j \in X_{i_j}$ and $e_2 \prec_P \overline{e}_j$. Since $X_{i_j}$ is totally ordered in $P$, the events $\overline{e}_j$ and $e_j$ are ordered in $P$. Clearly $e_1 \not\prec_P \overline{e}_j$, otherwise we would have $e_2 \not\prec_P e_1$. But then we already have $\overline{e}_j \prec_Q e_1$, and hence a cycle already exists in $Q$, contradicting our assumption about $(e_1, e_2)$ being the first pair where a cycle is encountered. The desired result follows. \qed

As the above lemma establishes that $t$ is well-defined, we can prove Lemma 5.2 by showing that $t$ is a witness of the realizability of $P$.

**Lemma 5.2.** Every closed, tree-inducible o-poset is realizable.

**Proof.** Consider any triplet $(w, r, w') \in Triplets(P)$ and we argue that (i) $w <_T r$ and (ii) $w <_T w'$ then $r <_T w'$. For (i), by the definition of o-posets, we have $w <_P r$ and thus $w <_T r$. We now turn our attention to (ii). If $w' \not\parallel_P r$ or $w' \not\parallel_P w$, since $P$ is closed, we have either $w' \not<_P w$ or $w \not<_P w'$, and hence $w' \not<_Q w$ or $r \not<_Q w'$. Since $t$ is a linearization of $Q$, the first case leads to a contradiction, whereas the second case leads to $r <_T w'$, as desired. Now assume that $w' \parallel_P r$ and $w' \parallel_P w$, and let $i \in [k]$ be such that $r \in X_i$. Since $P$ is tree-inducible,
it follows that \( w \in X_i \) and \( w' \in X_j \) for some \( i \neq j \), and such that \((i, j)\) is an edge of \( T \). Since \( w <_t w' \), we have \( w <_Q w' \), and thus, by construction, \( t \) is a child of \( i \) in \( T \). Again, by construction, we have \( r <_Q w \). Since \( t \) is a linearization of \( Q \), we have that \( O_t(r) \neq w' \).

The desired result follows. \( \square \)

Next, we prove Lemma 5.3, which states that tree-inducibility is preserved under taking closures.

**Lemma 5.3.** Consider an o-poset \( \mathcal{P} = (X, P, O) \) and let \( Q = (X, Q, O) \) be the closure of \( \mathcal{P} \). If \( \mathcal{P} \) is tree-inducible then \( Q \) is also tree-inducible.

**Proof.** Let \( T = ([k], \{(i, j) \mid X_i \Rightarrow X_j\}) \) be a rooted tree such that \( \mathcal{P} \) is tree-inducible to \( T \), and we argue that \( Q \) is also tree-inducible to \( T \). Since \( Q \subseteq P \), clearly the condition 1 and condition 2 of tree-inducibility are met. Condition 3 follows from the fact that (i) \( Q \) is the transitive closure of \( P \cup Q' \), where \( Q' \) is a relation between conflicting events, and (ii) since \( T \) is a tree, for any two conflicting events \( e_1, e_2 \) we have either \( e_1, e_2 \in X_i \) for some \( i \in [k] \), or \( e_1 \in X_i \) and \( e_2 \in X_j \) for some \( i, j \in [k] \) such that \((i, j)\) is an edge of \( T \).

The desired result follows. \( \square \)

Now we can conclude the proof of Lemma 5.1, which is step (i) towards the proof of Theorem 2.4.

**Lemma 5.1.** O-poset realizability of tree-inducible o-posets can be solved in \( O(k^2 \cdot d \cdot n^2 \cdot \log n) \) time, for an o-poset of size \( n \), \( k \) threads and \( d \) variables.

**Proof.** Consider a tree-inducible o-poset \( \mathcal{P} \). In \( O(k^2 \cdot d \cdot n^2 \cdot \log n) \) time, we can decide whether the closure of \( \mathcal{P} = (X, P, O) \) exists [28]. By Remark 2, if the closure does not exist, \( \mathcal{P} \) is not realizable. On the other hand, if the closure exists denote it by \( Q = (X, Q, O) \). By Lemma 5.3, \( Q \) is tree-inducible, and by Lemma 5.2, it is realizable by a witness \( t^* \).

Since \( Q \subseteq P \), we have that \( t^* \subseteq P \), and thus \( t^* \) also realizes \( \mathcal{P} \).

The desired result follows. \( \square \)

Now we turn our attention to Lemma 5.4 which forms step (ii) towards Theorem 2.4. We start with the following lemma, which, in high level, guarantees that if \( t^* \) is a trace in which \( e \) is enabled, then \( t^* \mid Lcone_e(e) \) also has this property.

**Lemma D.2.** Let \( X = Lcone_e(e) \), and \( t^* \) be any trace in which \( e \) is enabled. The following assertions hold.

1. \( X \subseteq E(t^*) \).
2. Consider any thread \( p_1 \neq p(e) \), and thread \( p_2 \) such that (i) \( X[p_1] \neq \emptyset \), and (ii) \( p_2 \) is a parent of \( p_1 \) in \( G_t \) rooted at \( p(e) \). Let \( e_2 \) be the unique maximal event of \( X[p_2] \) in \( t^* \), and \( e_1 \) any event in \( X[p_1] \). We have that \( e_1 <_{t^*} e_2 \).

**Proof.** The proof is by induction on the steps of the process that constructs \( Lcone_e(e) \). Clearly, both statements hold after Item 1 of the process has been executed. Similarly, both statements hold easily after each time Item 2 of the process has been executed. We now proceed with Item 3 of the process. For each \( i \in [2] \), consider the lock-acquire events \( acq_i \in X \) as identified in this step, and \( rel_i = match_i(acq_i) \). Observe that \( e_2 \) appears in the critical section of \( acq \), while by the induction hypothesis, we have \( acq_1, acq_2 \in E(t^*) \). We distinguish the step that led to \( acq_i \in X \).

Step 2. Then \( acq_i \prec_{TO} e_2 \), and thus (i) all predecessors of \( rel_1 \) (including \( rel_1 \)) appear in \( t^* \), and (ii) \( rel_1 \prec_{rel} acq_2 \), hence \( rel_1 \prec_{rel} e_2 \), and thus \( e_1 <_{t^*} e_2 \) for all predecessors of \( rel_1 \) inserted to \( X \).

Step 3. For each \( i \in [2] \), consider the lock-acquire events \( acq_i' \) as identified in that step, and \( rel_i' = match_i(acq_i') \). Note that \( acq_i \prec_{TO} rel_i' \), and since, by the induction hypothesis, we have \( rel_i' \prec_{rel} e_2 \), we obtain \( acq_i \prec_{rel} e_2 \). Thus, again, (i) all predecessors of \( rel_1 \) (including \( rel_1 \)) appear in \( t^* \), and (ii) \( rel_1 \prec_{rel} acq_2 \), hence \( rel_1 \prec_{rel} e_2 \), and thus \( e_1 <_{t^*} e_2 \) for all predecessors of \( rel_1 \) inserted to \( X \).

The desired result follows. \( \square \)

Next we have a technical lemma, which will allow us to conclude that if \( (e_1, e_2) \) is a predictable data race of \( t \), the ideal \( Lcone_e(e_1) \cup Lcone_e(e_2) \) is lock-feasible.

**Lemma D.3.** For any two conflicting events \( e_1, e_2 \), consider the set \( X = Lcone_e(e_1) \cup Lcone_e(e_2) \). For any two conflicting events \( e_1', e_2' \in X \) such that \( p(e_1') \neq p(e_2) \) for each \( i \in [2] \), we have that \( e_1', e_2' \in Lcone_e(e_i) \), for some \( i \in [2] \).

**Proof.** We assume that \( p(e_1) \neq p(e_2) \), as the statement clearly holds otherwise. Assume w.l.o.g. that \( e_1' \in Lcone_e(e_i) \) for each \( i \in [2] \), and consider the tree \( T = ([k], \{(i, j) \mid E(t)[p_1] \Rightarrow E(t)[p_2]\}) \). Let \( i, i_2, i_1, i_2' \in \{k\} \) be such that \( p(e_1) = p_{i_1} \), \( p(e_2) = p_{i_2} \), \( p(e_1') = p_{i_1'} \) and \( p(e_2') = p_{i_2'} \), and since \( e_1 \sim e_2 \), we have that \((i_1, i_2)\) is an edge of \( T \). Consider the two components \( C_1, C_2 \) that are created in \( T \) by removing the edge \((i_1, i_2)\), such that \( i_1 \in C_1 \) and \( i_2 \in C_2 \). Since \( i_1' \sim e_1 \) and \( i_1', i_2' \neq i_1 \), we have that \((i_1', i_2') \in C_1 \) or \((i_1', i_2') \in C_2 \). We only consider \((i_1', i_2') \in C_2 \), as the other case is similar.

Since \( e_1' \in Lcone_e(e_1) \), there exists event \( e \in Lcone_e(e_1) \) such that \( p(e) = p_{i_1'} \) and either \( e' = e \) or \( e_1' \in Lcone_e(e) \). If \( e \sim_{TO} e_2 \), then \( Lcone_e(e) \subseteq Lcone_e(e_2) \) and thus \( e_1' \in Lcone_e(e_2) \). Otherwise, \( e_2 \in Lcone_e(e_1) \), hence \( Lcone_e(e_2) \subseteq Lcone_e(e_1) \) and thus \( e_2' \in Lcone_e(e_1) \).

The desired result follows. \( \square \)

We are now ready to prove Lemma 5.4.

**Lemma 5.4.** Let \( X = Lcone_e(e_1) \cup Lcone_e(e_2) \). We have that \((e_1, e_2)\) is a predictable data race of \( t \) if (i) \( (e_1, e_2) \cap X = \emptyset \), and (ii) \( X \) is a realizable trace ideal of \( t \).

**Proof.** For the \((\Rightarrow)\) direction, notice that \( e_1 \) and \( e_2 \) are enabled in \( X \), and since \( X \) is realizable, we have that \((e_1, e_2)\) is a predictable data race of \( t \). We now focus on the \((\Leftarrow)\) direction.
Let $t'$ be a witness of the data race $(e_1, e_2)$. As $e_1$ and $e_2$ are enabled in $t'$, we have $(e_1, e_2) \cap E(t') = \emptyset$. By Lemma D.2, we have $X \subseteq E(t')$, and thus $(e_1, e_2) \cap X = \emptyset$.

We now argue that $t' = t^* | X$ realizes $X$. Since $X \subseteq E(t')$ and $X$ is a trace ideal, it only remains to show that $t'$ respects the critical sections. Consider any two lock-acquire events $acq_1, acq_2 \in X$ such that $acq_1 \preceq acq_2$ and $acq_1 \prec_X acq_2$, and we argue that $rel_1 \in X$, where $rel_1 = match(acq_1)$. Let $Y = E(t)|\{p(e_1), p(e_2)\}$, and observe that $X|\{p(e_1), p(e_2)\} = Y$, thus the statement is true if $\{p(acq_1), p(acq_2)\} \subseteq \{p(e_1), p(e_2)\}$. Otherwise, we have $p(acq_1) \neq p(e_1)$ for some $i \in [2]$ and each $j \in [2]$. By Lemma D.3, we have that $acq_1, acq_2 \in Lcone_{j}(e_i)$ for some $l \in [2]$. Assume towards contradiction that $rel_1 \notin X$. By the definition of $Lcone_{j}(e_i)$, we have that $p(acq_1)$ is a parent of $p(acq_2)$ in the tree $G_t$ rooted at $p(e_i)$. By Lemma D.2, we have $acq_2 \prec_X e$, where $e$ is the last event of $X|p(acq_1)$ in $t'$. Note that $e$ belongs to the critical section of $acq_1$, hence we must have $acq_2 \prec_X acq_1$, a contradiction.

The desired result follows.

\[ \square \]

D.2 Details of Section 5.2

Here we provide the proof of Lemma 5.6. We first develop some helpful notation.

Given integers $j, l \in [n/2]$, we denote by $X^j_l$ the events of $X_A$ and $X_B$ that correspond to vectors $a_j$ and $b_l$, that is, events that have superscript $a_j$ or $b_l$. If the observation of a read event $r$ is not in $X^j_l$, then $r$ is also not in $X^j_l$. The notation carries over to partial orders $Q^j_l$ and observation functions $O^j_l$. We also use inequalities for the subscripts and superscripts of $X$ (e.g., $X_{j, l/2}^j$) to denote events that corresponds to vectors $a_j^l$ and $b_j^l$ for $j$ and $l'$ that satisfy the inequalities.

Consider a partial order $Q'$ over a subset of $Y \subseteq X$ such that $Q' \subseteq \tau_A|Y$ and $Q' \subseteq \tau_B|Y$. Given two events $e_1, e_2 \in Y$ such that $\{e_1, e_2\} \not\subseteq X_A$ and $\{e_1, e_2\} \not\subseteq X_B$ (i.e., the events belong to different sets among $X_A$ and $X_B$), we say that $Q'$ has a cross edge $e_1 <_{Q'} e_2$ to mean that it has the ordering $e_1 <_{Q'} e_2$ and the orderings that are introduced transitively through it.

Before the final proof of Lemma 5.6, we present some technical, but conceptually simple, lemmas. Consider the two total orders $\tau_A$ and $\tau_B$ without any cross edges. The first lemma reasons at the coordinate level of two vectors $a_j$ and $b_l$. It states that if we start with a cross edge $w_{1}^{a_j}(x_1) \prec_{Q} r_{1}^{b_l}(x_1)$, the closure rules eventually lead to an ordering $w_{1}^{a_j}(x_1) <_{Q} \{b_{l}^{1}(x_1) \iff a_j$ and $b_l$ are not orthogonal.

\textbf{Lemma D.4.} For any $j, l \in [n/2]$, consider the o- prefix $Q^j_l = (X^j_l, Q^j_l, O^j_l)$, where $Q^j_l$ has a single cross edge $w_{1}^{a_j}(x_1) <_{Q^j_l} r_{1}^{b_l}(x_1)$. The closure of the partial order $Q^j_l$ is $Q^j_l = (X^j_l, S^j_l, O^j_l)$ be the closure of $Q^j_l$. We have $w_{1}^{a_j}(x_2) <_{S^j_l} b_{l}^{1}(x_2)$ iff $a_j$ and $b_l$ are not orthogonal.

\textbf{Proof.} First, observe that since $w_{1}^{a_j}(x_1) <_{Q^j_l} r_{1}^{b_l}(x_1)$, by closure, we will have $w_{1}^{a_j}(x_1) <_{Q^j_l} r_{1}^{b_l}(x_1)$. By a simple induction on the events on variables $x_1$ and $x_2$, we have that $w_{1}^{a_j}(x_1) <_{S^j_l} w_{1}^{b_l}(x_1)$ for each $i \in [D]$. In turn, one of these orderings leads to $w_{1}^{a_j}(x_2) <_{S^j_l} w_{1}^{b_l}(x_2)$ and thus $r_{1}^{a_j}(x_2) <_{S^j_l} w_{1}^{b_l}(x_2)$ iff $a_j \cup b_l = 1$. By a simple induction on the events on variables $x_2$ and $x_6$, we have that $w_{1}^{a_j}(x_2) <_{S^j_l} w_{1}^{b_l}(x_2)$ if $a_j$ and $b_l$ are not orthogonal. \[ \square \]

The next lemma concerns the vector $B$ level. Consider any fixed $i \in [n/2]$. The lemma states that if we order $w_{1}^{a_i}(x_1) <_{Q^j_l} r_{1}^{b_i}(x_1)$, if $a_i$ is not orthogonal to $b_i$ for any $l' \preceq n/2 - 1$, then the closure rules eventually lead to $w_{1}^{a_i}(x_1) <_{Q^j_l} r_{1}^{b_i}(x_1)$. On the other hand, if there exists a smallest $l_i$ such that $a_i$ is orthogonal to $b_{l_i}$, the closure rules will stop inserting cross edges between events of vectors $a_i$ and all $b_{l_i}$ for all $l' \geq l_i + 1$.

\textbf{Lemma D.5.} For any $i \in [n/2]$ and $l \preceq n/2 - 1$, consider the o- prefix $Q^j_l = (X^j_{i, l}, Q^j_{i, l}, O^j_{i, l})$, where $Q^j_{i, l}$ has a single cross edge $w_{1}^{a_i}(x_1) <_{Q^j_{i, l}} r_{1}^{b_i}(x_1)$. Let $S^j_{i, l} = (X^j_{i, l}, S^j_{i, l}, O^j_{i, l})$ be the closure of $Q^j_{i, l}$. The following assertions hold.

1. If $a_i$ is not orthogonal to $b_i$ for any $l' \in [l]$, then $w_{1}^{a_i}(x_1) <_{Q^j_{i, l}} r_{1}^{b_i}(x_1)$.
2. If $a_i$ is orthogonal to $b_i$, for some $l' \in [l]$, then there are no cross edges in $(X^j_{i, l'}, S^j_{i, l'}, O^j_{i, l'})$, where $l_i$ is the smallest $l'$ such that $a_i$ is orthogonal to $b_{l_i}$.

\textbf{Proof.} The proof is by induction on $l$. For the base case ($l = 1$), consider the o- prefix $Q^j_1$, and by Lemma D.4, we have $w_{1}^{a_i}(x_2) <_{S^j_1} b_{1}^{1}(x_2)$ iff $a_j$ and $b_1$ are not orthogonal. Observe that if $w_{1}^{a_i}(x_2) <_{S^j_1} b_{1}^{1}(x_2)$, we also have $w_{1}^{a_i}(x_2) <_{S^j_{i, l'}} b_{l'}^{1}(x_2)$, and thus $w_{1}^{a_i}(x_2) <_{S^j_{i, l'}} b_{l'}^{1}(x_2)$. Then, by closure we have $w_{1}^{a_i}(x_1) <_{S^j_{i, l'}} w_{1}^{b_i}(x_2)$ and thus transitively $w_{1}^{a_i}(x_1) <_{S^j_{i, l'}} b_{l'}^{1}(x_1)$. On the other hand, it can be easily seen that if $w_{1}^{a_i}(x_2) <_{S^j_1} b_{1}^{1}(x_1)$ then the cross edges in $(X^j_{i, l'}, S^j_{i, l'})$ are precisely the cross edges in $(X^j_{i, l'}, S^j_{i, l'})$, and thus there are no cross edges in $(X^j_{i, l'}, S^j_{i, l'})$.

Now assume that the claim holds for $l$, and we argue that it holds for $l + 1$. By the induction hypothesis, we have that if $a_i$ is not orthogonal to $b_i$ for any $l' \in [l]$, then $w_{1}^{a_i}(x_1) <_{Q^j_{i, l}} r_{1}^{b_i}(x_1)$ and thus $w_{1}^{a_i}(x_1) <_{Q^j_{i, l + 1}} r_{l + 1}^{b_i}(x_1)$. A similar analysis to the base case shows that if $a_i$ is not orthogonal to $b_{l + 1}$, then $w_{1}^{a_i}(x_1) <_{Q^j_{i, l + 1}} r_{l + 1}^{b_i}(x_1)$. On the other hand, if $a_i$ is orthogonal to $b_{l + 1}$, it can be easily seen that the cross edges in $(X^j_{i, l + 2}, S^j_{i, l + 2})$ are precisely the cross edges in $(X^j_{i, l + 1}, S^j_{i, l + 1})$, and thus there are no cross edges in $(X^j_{i, l + 1}, S^j_{i, l + 1})$, where
\( l_i = l + 1 \). Finally, if \( a_j \) is orthogonal to \( b_l \) for some \( l' \in [l] \), the statement follows easily by the induction hypothesis.

The desired result follows. \( \square \)

The next lemma reasons about \( \mathcal{A} \) level. The lemma states that if we order \( w_{\pi}(x_1) < r_{\pi}(x_1) \), if \( a_j \) is not orthogonal to \( b_l \) for any \( j' \leq l - 2 \) and \( l' \leq l \), the closure rules eventually lead to \( w_{\pi'}(x_1) < r_{\pi'}(x_1) \). On the other hand, if there exists a smallest \( j_1 \) such that \( a_{j_1} \) is orthogonal to \( b_{l'} \), for some \( l' \in [l] \), the closure rules will stop inserting crossing edges between events of all vectors \( a_j \) and all vectors \( b_l \), for \( j' \geq j_1 + 1 \) and \( l' \in [n] \).

**Lemma D.6.** For any \( j \in [n/2 - 1] \), consider the o-pose \( Q_{\leq j + 1}^{\leq n/2} = (X_j^{\leq n/2},Q_{j+1}^{\leq n/2},O_{j+1}^{\leq n/2}) \), where \( Q_{\leq j + 1}^{\leq n/2} \) has a single cross edge \( w_{\pi_1}(x_1) \). Let \( S_{\leq j + 1}^{\leq n/2} = (X_j^{\leq n/2},Q_{j+1}^{\leq n/2},O_{j+1}^{\leq n/2}) \) be the closure of \( Q_{\leq j + 1}^{\leq n/2} \). The following assertions hold.

1. If \( a_j \) is not orthogonal to \( b_j \) for any \( j' \in [j] \), then \( w_{\pi_{j+1}}(x_1) \).

2. If \( a_j \) is orthogonal to \( b_j \) for some \( j' \in [j] \) and \( j' \in [n/2] \), then there is no cross edges in \( (X_j^{\leq n/2},Q_{j+1}^{\leq n/2},O_{j+1}^{\leq n/2}) \) for some \( j' \in [n] \).

**Proof:** The proof is by induction on \( j \). For the base case \( (j = 1) \), consider the o-pose \( Q_1^{\leq n/2} = (X_1^{\leq n/2},Q_1^{\leq n/2},O_1^{\leq n/2}) \), and by Lemma D.5, we have that \( a_1 \) is orthogonal to \( b_1 \) for any \( l' \in [n/2 - 1] \), then \( w_{\pi_1}(x_1) < w_{\pi_{j+1}}(x_1) \). Consider the o-pose \( Q_1^{\leq n/2} \), and by Lemma D.4, we have \( w_{\pi_2}(x_2) < w_{\pi_{j+1}}(x_2) < w_{\pi_{j+2}}(x_2) \) if \( a_1 \) and \( b_1 \) are not orthogonal. Observe that if \( w_{\pi_2}(x_2) < w_{\pi_{j+1}}(x_2) \), we also have \( w_{\pi_2}(x_2) < w_{\pi_{j+1}}(x_2) \), and thus \( w_{\pi_2}(x_2) < w_{\pi_{j+1}}(x_2) \), hence, transitively, \( w_{\pi_2}(x_2) < w_{\pi_{j+1}}(x_2) \). Thus, by closure, we have \( r_{\pi_2}(x_2) < w_{\pi_{j+1}}(x_2) \) and thus transitively \( w_{\pi_2}(x_2) < w_{\pi_{j+1}}(x_2) \). On the other hand, it can be easily seen that if \( w_{\pi_2}(x_2) < w_{\pi_{j+1}}(x_2) \), then the cross edges in \( (X_j^{\leq n/2},S_{\leq j+1}^{\leq n/2}) \) are precisely the cross edges in \( (X_j^{\leq n/2},S_{\leq j+1}^{\leq n/2}) \) and there are no cross edges in \( (X_j^{\leq n/2},S_{\leq j+1}^{\leq n/2}) \). Now assume that the claim holds for \( j \), and we argue that it holds for \( j + 1 \). By the induction hypothesis, we have that if \( a_j \) is orthogonal to \( b_l \) for any \( l' \in [l] \) and \( l' \in [n] \), then \( w_{\pi_{j+1}}(x_1) < w_{\pi_{j+1}}(x_2) \) and thus \( w_{\pi_{j+1}}(x_1) < w_{\pi_{j+1}}(x_2) \). Similar analysis to the base case shows that if \( a_{j+1} \) is orthogonal to \( b_{l'} \) for any \( l' \in [l'] \), then \( w_{\pi_{j+1}}(x_2) < w_{\pi_{j+1}}(x_2) \). On the other hand, if \( a_{j+1} \) is orthogonal to some \( b_{l'} \), it can be easily seen that the cross edges in \( (X_j^{\leq n/2},S_{\leq j+1}^{\leq n/2}) \) are precisely the cross edges in \( (X_j^{\leq n/2},S_{\leq j+1}^{\leq n/2}) \). Thus, there are no cross edges in \( (X_j^{\leq n/2},S_{\leq j+1}^{\leq n/2}) \), where \( j = j' + 1 \). Finally, if \( a_j \) is orthogonal to \( b_j \) for some \( j' \in [j] \) and \( l' \in [n/2] \), the statement follows easily.

The desired result follows. \( \square \)

We next have three lemmas that each is symmetric to Lemma D.4, Item 2 of Lemma D.5, and Item 2 of Lemma D.6, respectively. The proof of each lemma is analogous to its symmetric lemma, and is omitted here.

**Lemma D.7.** For any \( j,l \in [n/2] \), consider the o-pose \( Q_j^{l-1} = (X_j^{l-1},Q_j^{l-1},O_j^{l-1}) \), where \( Q_j^{l-1} \) has a single cross edge \( w_{\pi_1}(x_1) < w_{\pi_1}(x_2) \). Let \( S_j^{l-1} = (X_j^{l-1},Q_j^{l-1},O_j^{l-1}) \) be the closure of \( Q_j^{l-1} \). We have \( r_{\pi_1}(x_1) < w_{\pi_1}(x_1) \) iff \( a_j \) and \( b_l \) are not orthogonal.

**Lemma D.8.** For any \( j \in [n/2] \) and \( l \geq 2 \), consider the o-pose \( Q_j^{l-1} = (X_j^{l-1},Q_j^{l-1},O_j^{l-1}) \), where \( Q_j^{l-1} \) has a single cross edge \( w_{\pi_1}(x_1) < w_{\pi_1}(x_2) \). Let \( S_j^{l-1} = (X_j^{l-1},Q_j^{l-1},O_j^{l-1}) \) be the closure of \( Q_j^{l-1} \). If \( a_j \) is orthogonal to \( b_l \), for some \( l' \in [l] \), then there are no cross edges in \( (X_j^{l-2},Q_j^{l-2},O_j^{l-2}) \), where \( l' \) is the largest \( l' \) such that \( a_j \) is orthogonal to \( b_l \).

**Lemma D.9.** For any \( j \geq 2 \) with \( j > 1 \), consider the o-pose \( Q_j^{l-1} = (X_j^{l-1},Q_j^{l-1},O_j^{l-1}) \), where \( Q_j^{l-1} \) has a single cross edge \( w_{\pi_1}(x_1) < w_{\pi_1}(x_2) \). Let \( S_j^{l-1} = (X_j^{l-1},Q_j^{l-1},O_j^{l-1}) \) be the closure of \( Q_j^{l-1} \). If \( a_j \) is orthogonal to \( b_l \), for some \( l' \in [l] \), then there are no cross edges in \( (X_j^{l-2},Q_j^{l-2},O_j^{l-2}) \), where \( l' \) is the largest \( l' \) such that \( a_j \) is orthogonal to \( b_l \), for some \( l' \in [n/2] \).

Using Lemma D.4, Lemma D.5, Lemma D.6 we can now prove Lemma D.6.

**Lemma 5.6.** O-pose realizability for and o-pose with 2 threads and at least 7 variables has no \( O(n^\epsilon) \)-time algorithm for any \( \epsilon > 0 \), under the Orthogonal Vectors conjecture.

**Proof:** We show that \( \mathcal{P} \) is realizable iff there exist \( j,l \in [n/2] \) such that \( a_j \) is orthogonal to \( b_l \). We prove each direction separately.

\( (\Rightarrow) \). Since \( \mathcal{P} \) is realizable, by Remark 2, we have that the closure of \( \mathcal{P} \) exists. Let \( \mathcal{K} = (X,K,O) \) be the closure of \( \mathcal{P} \). Let \( Q = (X,Q,O) \) where \( Q \) has a single cross edge \( w_{\pi_1}(x_1) < w_{\pi_1}(x_2) \), and \( S = (X,S,O) \) be the closure of \( Q \). Note that \( P \subseteq Q \) and thus \( K \subseteq S \), and we have that \( w_{\pi_1}(x_1) \).
Lemma D.4, we have that \( a_{n/2} \) is orthogonal to \( b_{n/2} \). Otherwise, if \( w_{1}^{a_{n/2}}(x_{1}) \leq S_{1}^{b_{n/2}}(x_{1}) \), by Item 1 of Lemma D.5, we have that \( a_{n/2} \) is orthogonal to \( b_{l} \), for some \( l \in [n/2] \). Finally, if \( w_{1}^{a_{n/2}}(x_{1}) \not\leq S_{1}^{b_{n/2}}(x_{1}) \), by Item 1 of Lemma D.6, we have that \( a_{n/2} \) is orthogonal to \( b_{j} \), for some \( j \leq n/2-1 \) and \( l \in [n/2] \).

(\( \Rightarrow \)). Assume that \( \mathcal{P} \) is realizable and \( t' \) is a witness trace. We construct the correct reordering \( t'' \) as follows. We have \( \mathcal{E}(t') = \mathcal{E}(t) \setminus \{rel_2(t), w_2(y), r_2(y)\} \). We make \( t' \) identical to \( t'' \), i.e., \( t''|X = t' \). For the events \( \mathcal{E}(t'') \setminus X \), we make (i) \( w(y) \) appear right after \( w_{1}^{a_{n/2}}(x_{2}) \), (ii) \( r(y) \) appear right before \( r_{1}^{a_{n/2}}(x_{2}) \), (iii) \( \text{acq}_{q}(t) \), \( \text{rel}_{k}(t) \) appear right after \( w_{1}^{a_{n/2}}(x_{1}) \), and (iv) \( \text{acq}_{q}(t) \) appear last in \( t'' \). Observe that \( t' \) is a correct reordering in which \( w_2(y) \) and \( r_2(y) \) are enabled, hence \( t'' \) is a witness of the data race \( (w_2(y), r_2(y)) \).

\( \Leftarrow \). Assume that \( (w_2(y), r_2(y)) \) is a predictable data race of \( t \), and let \( t'' \) be a correct reordering of \( t \) witnessing the data race. We construct a trace \( t' \) as \( t''|X = t' \), and argue that \( t' \) realizes \( \mathcal{P} \). Clearly \( O_t = O \). To see that \( t'' \) is a linearization of \( (X, P) \), it suffices to argue that

\[
\begin{align*}
\mathcal{E}(t') &= \mathcal{E}(t) \setminus \{rel_2(t), w_2(y), r_2(y)\}. \\
\text{For the first ordering, observe that } \mathcal{E}(t'') &= \mathcal{E}(t) \setminus \{rel_2(t), w_2(y), r_2(y)\}. \\
\text{Thus, } w_{1}^{a_{n/2}}(x_1) < r_{1}^{a_{n/2}}(x_2) \quad \text{and} \quad w_{1}^{a_{n/2}}(x_1) < r_{1}^{b_{n/2}}(x_1)
\end{align*}
\]

The desired result follows.

## E Details of Section 6

In this section we present the full proof of Lemma 6.1.

**Lemma 6.1.** Consider given a natural number \( \ell \), a trace \( t \) over \( n \) events and \( k \) threads, and a feasible trace ideal \( X \) of \( t \). The \( \ell \)-distance-bounded realizability problem for \( X \) can be solved in \( O(k^{\ell+O(1)} \cdot n) \) time.

**Proof.** Let \( \mathcal{P} = (X, P, O) \) be the canonical o-poset of \( X \), and the task is to decide the realizability of \( \mathcal{P} \) with \( \ell \) reversals. We describe a recursive algorithm for solving the problem for some o-poset \( Q = (X, Q, O) \) with \( \ell' \) reversals, for some \( \ell' \leq \ell \), where initially \( Q = P \) and \( \ell' = \ell \). We first give a high-level description and argue about its correctness. Afterwards, we describe some low-level details that allow us to reason about the complexity.

**Algorithm and correctness.** Consider the set

\[
\begin{align*}
C = \{(w_1, w_2) \in \mathcal{W}(X) \times \mathcal{W}(X) : w_1 &\sim w_2 \quad \text{and} \quad w_1 \parallel_Q w_2 \text{ and } w_1 \ll_Q w_2 \}
\end{align*}
\]

We construct a graph \( G_1 = (X, E_1) \), where \( E_1 = (\text{TOO}|X) \cup C \). Note that \( G_1 \) is write-ordered. If it is acyclic, we construct the read extension \( G_2 \) of \( G_1 \). Observe that if \( G_2 \) is acyclic then any linearization \( t' \) of \( G \) realizes \( Q \), hence we are done. Now consider that either \( G_1 \) or \( G_2 \) is not acyclic, and let
We now argue that all events $t$ can be retrieved from $G$ as follows. We simply traverse the total order $\tau_i^*$ for each thread $i$ backwards until we find either (i) an event $w_k'$ with the desired properties, or (ii) an event $w_k''$ that has been examined before when inserting edges to thread $p_i$. In the first case, we simply add the edge $(w_k' \rightarrow w)$ as described above. In the second case we do nothing, as the desired ordering is already present due to transitivity through a write event $w'$ of thread $p_i$ that has been examined before (thus $w' \leq_{\text{TO}} w$). Hence, every event in every total order $\tau_i^*$ is examined $O(1)$ times, thus the total time for identifying all $w_k'$ events is $O(n)$.

Finally, the above process creates graphs $G'_1$ and $G_2'$ that have $O(k^{O(1)} \cdot n)$ edges. Detecting a cycle $\mathcal{C}'$ in either $G'_1$ or $G_2'$ is done by a simple DFS, which takes linear time in the number of edges. Converting $\mathcal{C}'$ to a cycle $\mathcal{C}$ such that $\mathcal{C}$ has at most $k$ cross edges can be done in $O(|\mathcal{C}'|) = O(k^{O(1)} \cdot n)$ time, by removing multiple edges whose endpoints are $\text{TO}$-ordered. The desired result follows.  

1. Events $w_1$: as we traverse $i$ top-down, we simply remember for each memory location $x$, the last write event or lock-acquire event on location $x$ for each thread. Hence, the total time for identifying all such events is $O(n)$.

2. Events $w_2$: Given the event $w_1$ on memory location $x$, we simply traverse the total order $\tau_i^*$ from $w_1' \rightarrow w_1$ backwards until we find either (i) an event $w_2'$ with the desired properties, or (ii) an event $w_2''$ that has been examined before when inserting edges to thread $p_i$. In the first case, we simply add the edge $(w_2' \rightarrow w_2')$ as described above. In the second case we do nothing, as the desired ordering is already present due to transitivity through a write event $w_2'$ of thread $p_i$ that has been examined before (thus $w_2' \leq_{\text{TO}} w_2$). Hence, every event in every total order $\tau_i^*$ is examined $O(1)$ times, thus the total time for identifying all such events is $O(n)$.

3. Events $w_3$: after we have constructed $G'_1$, each pair of such events can be retrieved from $G'_1$ in $O(1)$ time. Hence the total time for identifying all such events is $O(n)$.

For the graph $G_1$, we construct a sparse graph $G'_1$ that preserves the reachability relationships of $G_1$ as follows. We traverse the trace $t$ top-down. For the current event $w$ such that $w \in W_L(X)$, for every $i \in [k]$, let $w_i$ be the last event of thread $i$ which precedes $w$ in $t$, and such that $w_i \in W_L(X)$ and $w \approx w_i$. Let $w_i'$ be the last event of $t$ such that $w_i' \leq_{\text{TO}} w_i$, and $w \leq_\mathcal{Q} w_i'$ (note that possibly $w_i' = w_i$). If such a $w_i'$ exists, since $X$ is an ideal, we have $w_i' \in X$. We introduce the edge $w_i' \rightarrow w$ in $G'_1$.

Similarly, for the graph $G_2$, we construct a sparse graph $G'_2$ that preserves the reachability relationships of $G_2$ as follows. We iterate over all read and lock-release events $r \in X$. For each such $r$ and thread $i \in [k]$, we insert two edges $(w_2, r)$ and $(r, w_3)$ in $G'_2$, where $w_2$ (resp., $w_3$) is the latest predecessor (resp., earliest successor) of $r$ in thread $i$ that conflicts with $r$.

We now argue that all events $w_1, w_1', w_2$ and $w_3$ above can be identified in $O(n)$ total time. For every $i \in [k]$ and memory location $x$, we construct a total order $\tau_i^*$ of all write events or lock-acquire events on location $x$ of thread $p_i$. Clearly all such total orders can be constructed in $O(n)$ time.