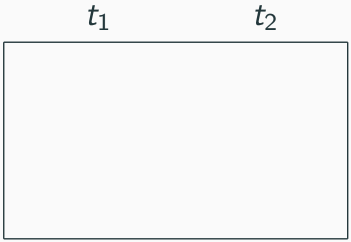
Sound Dynamic Deadlock Prediction in Linear Time

Hünkar Can Tunç
Umang Mathur
Andreas Pavlogiannis
Mahesh Viswanathan

AARHUS UNIVERSITY
NUS National University of Singapore
UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

The Problem



t_1 t_2

The Problem

	t_1	t_2
1	acquire(ℓ_1)	

The Problem

	t_1	t_2
1	acquire(ℓ_1)	
2		acquire(ℓ_2)

The Problem

	t_1	t_2
1	acquire(ℓ_1)	
2		acquire(ℓ_2)
3	request(ℓ_2)	

The Problem

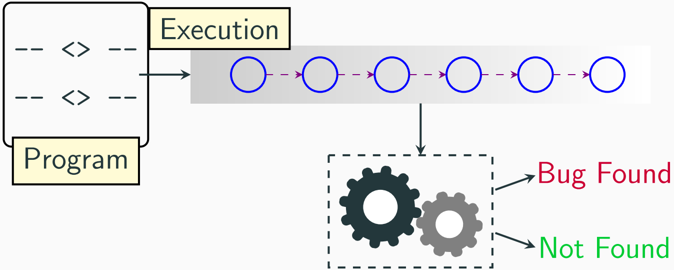
	t_1	t_2
1	acquire(ℓ_1)	
2		acquire(ℓ_2)
3	request(ℓ_2)	
4		request(ℓ_1)

The Problem

	t_1	t_2
1	acquire(ℓ_1)	
2		acquire(ℓ_2)
3	request(ℓ_2)	
4		request(ℓ_1)

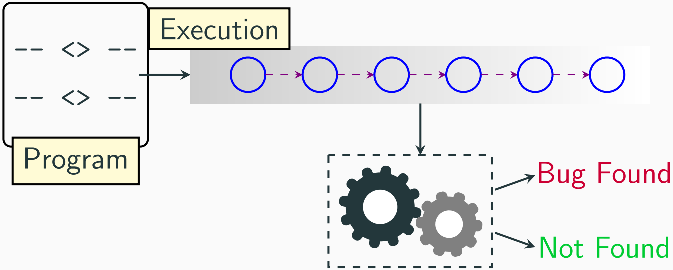
(Resource) Deadlock!

Dynamic Analysis



- Effective method for finding concurrency bugs
- Widely adopted (e.g., ThreadSanitizer, Helgrind)

Dynamic Analysis



- Effective method for finding concurrency bugs
- Widely adopted (e.g., ThreadSanitizer, Helgrind)
- Traditional techniques:
  - ↪ Analyze the current execution
- **Predictive** techniques:
  - ↪ Analyze the current execution + infer alternates

Predictive Analysis

	t_1	t_2
1	acquire(ℓ_1)	
2	acquire(ℓ_2)	
3	release(ℓ_2)	
4	release(ℓ_1)	
5		acquire(ℓ_2)
6		acquire(ℓ_1)
7		release(ℓ_1)
8		release(ℓ_2)

Observed trace

No deadlock

Predictive Analysis

	t_1	t_2
1	acquire(ℓ_1)	
2	acquire(ℓ_2)	
3	release(ℓ_2)	
4	release(ℓ_1)	
5		acquire(ℓ_2)
6		acquire(ℓ_1)
7		release(ℓ_1)
8		release(ℓ_2)

$\xrightarrow{\text{reordering}}$

	t_1	t_2
1		acquire(ℓ_2)
2	acquire(ℓ_1)	
3	acquire(ℓ_2)	acquire(ℓ_1)

Observed trace
No deadlock

Reordered trace
Deadlock!

Contributions

- We study the problem of dynamic deadlock **prediction**
- **Main results:**
  - Complexity characterization
    ↪ Tradeoff between efficiency and precision is unavoidable
  - Novel algorithms
    ↪ Strike a good balance between efficiency and precision
  - Empirical evaluation
    ↪ Outperform state-of-the-art techniques

Dynamic Deadlock Prediction

State-of-the-art

- **SeqCheck**[1]**:**
  - Sound but incomplete
  - High polynomial complexity

    $\hookrightarrow \widetilde{O}(\mathcal{N}^4)$
- **Dirk**[2]**:**
  - Sound and complete
  - Heavyweight SMT solving

---

[1] Yan Cai, Hao Yun, Jinqiu Wang, Lei Qiao, Jens Palsberg. Sound and efficient concurrency bug prediction. ESEC/FSE'21

[2] Christian Gram Kalhauge, Jens Palsberg. Sound deadlock prediction. OOPSLA'18

Dynamic Deadlock Prediction

State-of-the-art

- **SeqCheck**[1]**:**
  - Sound but incomplete
  - High polynomial complexity
    - $\hookrightarrow \widetilde{O}(\mathcal{N}^4)$
- **Dirk**[2]**:**
  - Sound and complete
  - Heavyweight SMT solving

This work

- **Sync-Preserving Deadlocks:**
  - Sound but incomplete
  - (Nearly) Linear time algorithm
    - $\hookrightarrow \widetilde{O}(\mathcal{N})$
    - $\hookrightarrow$ Wrt. number of events

---

[1] Yan Cai, Hao Yun, Jinqiu Wang, Lei Qiao, Jens Palsberg. Sound and efficient concurrency bug prediction. ESEC/FSE'21

[2] Christian Gram Kalhauge, Jens Palsberg. Sound deadlock prediction. OOPSLA'18

Dynamic Deadlock Prediction

State-of-the-art

- **SeqCheck**[1]**:**
  - Sound but incomplete
  - High polynomial complexity
    - ↪ $\widetilde{O}(\mathcal{N}^4)$
- **Dirk**[2]**:**
  - Sound and complete
  - Heavyweight SMT solving

This work

- **Sync-Preserving Deadlocks:**
  - Sound but incomplete
  - (Nearly) Linear time algorithm
    - ↪ $\widetilde{O}(\mathcal{N})$
    - ↪ Wrt. number of events

Focus is on identifying real deadlocks!

---

[1] Yan Cai, Hao Yun, Jinqiu Wang, Lei Qiao, Jens Palsberg. Sound and efficient concurrency bug prediction. ESEC/FSE'21

[2] Christian Gram Kalhauge, Jens Palsberg. Sound deadlock prediction. OOPSLA'18

Predictive Analysis

Two steps of predictive analysis:

① Identify potential buggy events

② Check if the potential bug can be realized

Predictive Analysis

Two steps of predictive analysis:

→ ① Identify potential buggy events

② Check if the potential bug can be realized

Potential Deadlocks

- **Potential deadlocks:**
  - Cyclic lock acquisition patterns

    $\hookrightarrow \ell_1, \ell_2$

	t_1	t_2
1	$\cdots$	
2	acquire(ℓ_1)	
3	acquire(ℓ_2)	
4	$\cdots$	
5	release(ℓ_2)	
6	release(ℓ_1)	
7	$\cdots$	
8		$\cdots$
9		acquire(ℓ_2)
10		acquire(ℓ_1)
11		release(ℓ_1)
12		release(ℓ_2)
13		$\cdots$

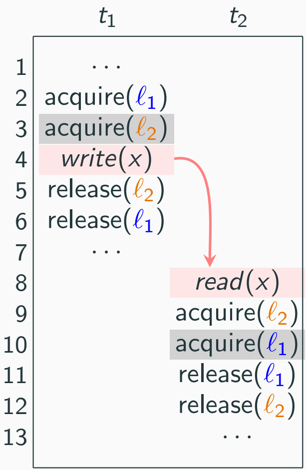
Potential Deadlocks

- **Potential deadlocks:**
  - Cyclic lock acquisition patterns
    $\hookrightarrow$ ℓ_1, ℓ_2
  - Not protected by a common lock
    $\hookrightarrow$ No such ℓ_3

	t_1	t_2
1	acquire(ℓ_3)	
2	acquire(ℓ_1)	
3	acquire(ℓ_2)	
4	$\cdots$	
5	release(ℓ_2)	
6	release(ℓ_1)	
7	release(ℓ_3)	
8		acquire(ℓ_3)
9		acquire(ℓ_2)
10		acquire(ℓ_1)
11		release(ℓ_1)
12		release(ℓ_2)
13		release(ℓ_3)

Potential Deadlocks

- **Potential deadlocks:**
  - Cyclic lock acquisition patterns
    $\hookrightarrow$ ℓ_1, ℓ_2
  - Not protected by a common lock
    $\hookrightarrow$ No such ℓ_3
  - Necessary but insufficient for an actual deadlock
    $\hookrightarrow$ Control flow/data flow dependencies



	t_1	t_2
1	$\cdots$	
2	acquire(ℓ_1)	
3	acquire(ℓ_2)	
4	*write*(x)	
5	release(ℓ_2)	
6	release(ℓ_1)	
7	$\cdots$	
8		*read*(x)
9		acquire(ℓ_2)
10		acquire(ℓ_1)
11		release(ℓ_1)
12		release(ℓ_2)
13		$\cdots$

Potential Deadlocks

- **Our first result:**
  - Identifying potential deadlocks is intractable

    $\hookrightarrow$ NP-hard

Potential Deadlocks

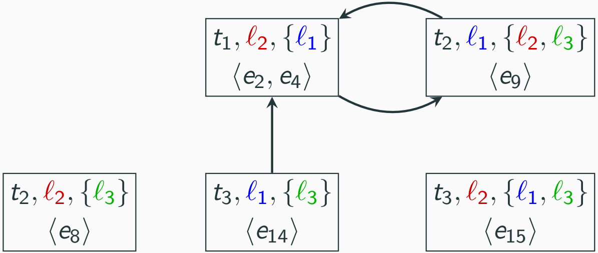
- **Our first result:**
  - Identifying potential deadlocks is intractable
    ↪ NP-hard

- **Our solution:**
  - Abstraction that groups potential deadlocks
    ↪ Abstract lock graph

Abstract Lock Graph

	t_1	t_2	t_3
1	acquire(ℓ_1)		
2	acquire(ℓ_2)		
3	release(ℓ_2)		
4	acquire(ℓ_2)		
5	release(ℓ_2)		
6	release(ℓ_1)		
7		acquire(ℓ_3)	
8		acquire(ℓ_2)	
9		acquire(ℓ_1)	
10		release(ℓ_1)	
11		release(ℓ_2)	
12		release(ℓ_3)	
13			acquire(ℓ_3)
14			acquire(ℓ_1)
15			acquire(ℓ_2)
16			release(ℓ_2)
17			release(ℓ_1)
18			release(ℓ_3)

Observed trace



$t_1, \ell_2, \{\ell_1\}$
$\langle e_2, e_4 \rangle$

$t_2, \ell_1, \{\ell_2, \ell_3\}$
$\langle e_9 \rangle$

$t_2, \ell_2, \{\ell_3\}$
$\langle e_8 \rangle$

$t_3, \ell_1, \{\ell_3\}$
$\langle e_{14} \rangle$

$t_3, \ell_2, \{\ell_1, \ell_3\}$
$\langle e_{15} \rangle$

Thread, Lock, { Locks Held }, ⟨Events⟩

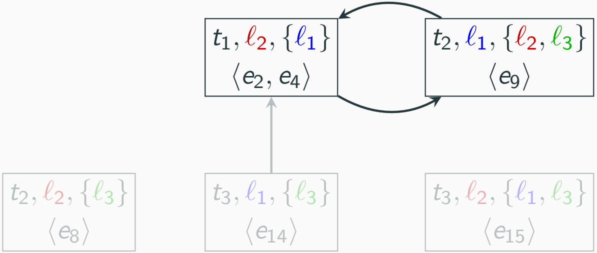
Abstract Lock Graph

Abstract Lock Graph

	t_1	t_2	t_3
1	acquire(ℓ_1)		
2	acquire(ℓ_2)		
3	release(ℓ_2)		
4	acquire(ℓ_2)		
5	release(ℓ_2)		
6	release(ℓ_1)		
7		acquire(ℓ_3)	
8		acquire(ℓ_2)	
9		acquire(ℓ_1)	
10		release(ℓ_1)	
11		release(ℓ_2)	
12		release(ℓ_3)	
13			acquire(ℓ_3)
14			acquire(ℓ_1)
15			acquire(ℓ_2)
16			release(ℓ_2)
17			release(ℓ_1)
18			release(ℓ_3)

Observed trace

Potential deadlocks: $\langle e_2, e_9 \rangle, \langle e_4, e_9 \rangle$



Thread, Lock, $\{$ *Locks Held* $\}$, $\langle$*Events*$\rangle$

Abstract Lock Graph

Predictive Analysis

Two steps of predictive analysis:

① Identify potential buggy events ✓

→ ② Check if the potential bug can be realized

Predicting Deadlocks

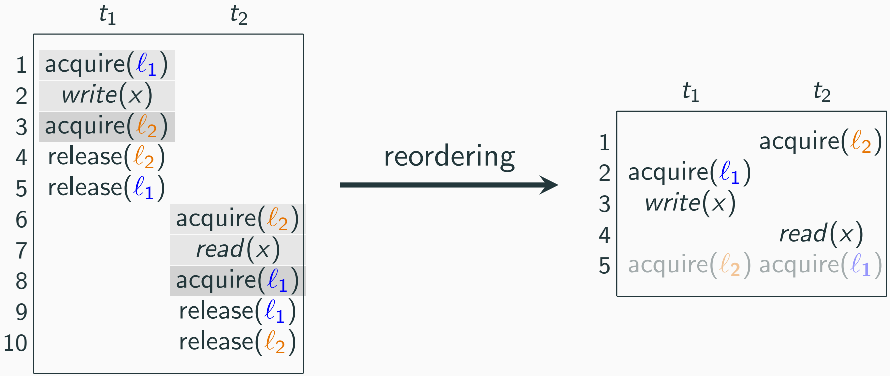
	t_1	t_2
1	acquire(ℓ_1)	
2	$\mathit{write}(x)$	
3	acquire(ℓ_2)	
4	release(ℓ_2)	
5	release(ℓ_1)	
6		acquire(ℓ_2)
7		$\mathit{read}(x)$
8		acquire(ℓ_1)
9		release(ℓ_1)
10		release(ℓ_2)

Observed trace

No deadlock

Potential deadlock → Real deadlock?

Predicting Deadlocks



	t_1	t_2
1	acquire(ℓ_1)	
2	*write*(x)	
3	acquire(ℓ_2)	
4	release(ℓ_2)	
5	release(ℓ_1)	
6		acquire(ℓ_2)
7		*read*(x)
8		acquire(ℓ_1)
9		release(ℓ_1)
10		release(ℓ_2)

Observed trace

No deadlock

	t_1	t_2
1		acquire(ℓ_2)
2	acquire(ℓ_1)	
3	*write*(x)	
4		*read*(x)
5	acquire(ℓ_2)	acquire(ℓ_1)

Witness trace

Deadlock!

Potential deadlock → Real deadlock ✓

Predicting Deadlocks

- **Our second result:** (Given a potential deadlock)
  - Sound and complete deadlock prediction is intractable

    ↪ NP-hard

Predicting Deadlocks

- **Our second result:** (Given a potential deadlock)
  - Sound and complete deadlock prediction is intractable
    $\hookrightarrow$ NP-hard

- **General solution:**
  - Consider a restricted problem setting to gain efficiency
    $\hookrightarrow$ Look for a subset of deadlocks

Predicting Deadlocks

- **Our second result:** (Given a potential deadlock)
  - Sound and complete deadlock prediction is intractable
    - ↪ NP-hard

- **General solution:**
  - Consider a restricted problem setting to gain efficiency
    - ↪ Look for a subset of deadlocks

- **Challenge:**
  - Restrictions should satisfy the following two properties
    - ↪ Enable efficient analysis
    - ↪ Retain high precision

Sync-Preserving Deadlocks

- Adapted from data races[1]
- Subset of deadlocks
  - ↪ More conservative restrictions on the allowed reorderings
- Enables efficient analysis

---

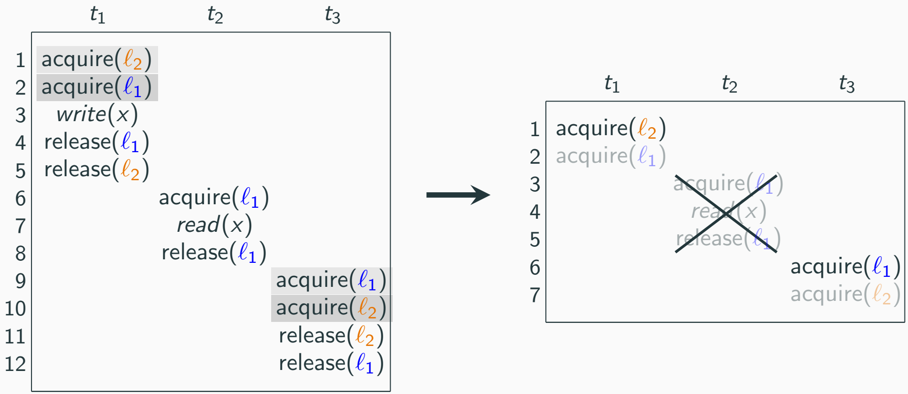
[1] Umang Mathur, Andreas Pavlogiannis, Mahesh Viswanathan. Optimal Prediction of Synchronization-Preserving Races. POPL'21

Sync-Preserving Deadlocks

	t_1	t_2	t_3
1	acquire(ℓ_2)		
2	acquire(ℓ_1)		
3	*write*(x)		
4	release(ℓ_1)		
5	release(ℓ_2)		
6		acquire(ℓ_1)	
7		*read*(x)	
8		release(ℓ_1)	
9			acquire(ℓ_1)
10			acquire(ℓ_2)
11			release(ℓ_2)
12			release(ℓ_1)

Observed trace

Sync-Preserving Deadlocks

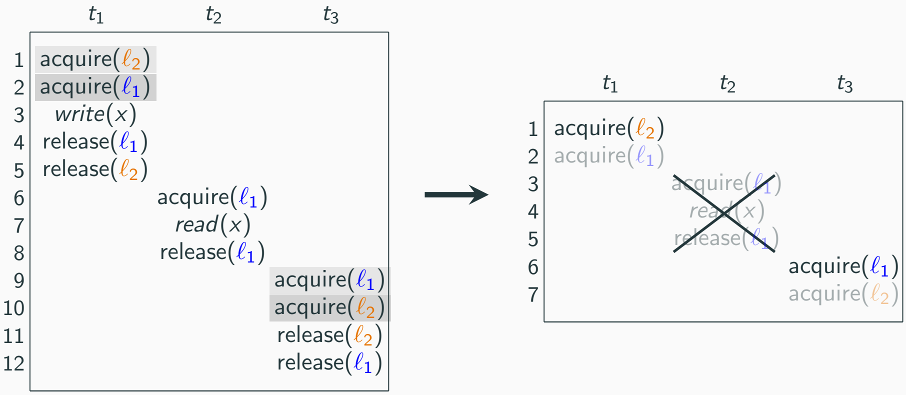


	t_1	t_2	t_3
1	acquire(ℓ_2)		
2	acquire(ℓ_1)		
3	$\mathit{write}(x)$		
4	release(ℓ_1)		
5	release(ℓ_2)		
6		acquire(ℓ_1)	
7		$\mathit{read}(x)$	
8		release(ℓ_1)	
9			acquire(ℓ_1)
10			acquire(ℓ_2)
11			release(ℓ_2)
12			release(ℓ_1)

Observed trace

	t_1	t_2	t_3
1	acquire(ℓ_2)		
2	acquire(ℓ_1)		
3		~~acquire(ℓ_1)~~	
4		~~$\mathit{read}(x)$~~	
5		~~release(ℓ_1)~~	
6			acquire(ℓ_1)
7			acquire(ℓ_2)

Sync-Preserving Deadlocks



	t_1	t_2	t_3
1	acquire(ℓ_2)		
2	acquire(ℓ_1)		
3	*write*(x)		
4	release(ℓ_1)		
5	release(ℓ_2)		
6		acquire(ℓ_1)	
7		*read*(x)	
8		release(ℓ_1)	
9			acquire(ℓ_1)
10			acquire(ℓ_2)
11			release(ℓ_2)
12			release(ℓ_1)

Observed trace

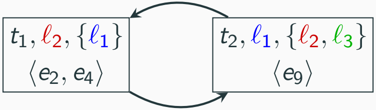
	t_1	t_2	t_3
1	acquire(ℓ_2)		
2	acquire(ℓ_1)		
3		~~acquire(ℓ_1)~~	
4		~~*read*(x)~~	
5		~~release(ℓ_1)~~	
6			acquire(ℓ_1)
7			acquire(ℓ_2)

Sync-Preserving Deadlock

Order of `acquire` events on the same lock that occur in the witness are maintained

Interplay With Abstract Lock Graph

	t_1	t_2	t_3
1	acquire(ℓ_1)		
2	acquire(ℓ_2)		
3	release(ℓ_2)		
4	acquire(ℓ_2)		
5	release(ℓ_2)		
6	release(ℓ_1)		
7		acquire(ℓ_3)	
8		acquire(ℓ_2)	
9		acquire(ℓ_1)	
10		release(ℓ_1)	
11		release(ℓ_2)	
12		release(ℓ_3)	
13			acquire(ℓ_3)
14			acquire(ℓ_1)
15			acquire(ℓ_2)
16			release(ℓ_2)
17			release(ℓ_1)
18			release(ℓ_3)



$t_1, \ell_2, \{\ell_1\}$

$\langle e_2, e_4 \rangle$

$t_2, \ell_1, \{\ell_2, \ell_3\}$

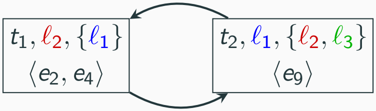
$\langle e_9 \rangle$

Potential deadlocks:

$\langle e_2, e_9 \rangle, \langle e_4, e_9 \rangle$

Interplay With Abstract Lock Graph

	t_1	t_2	t_3
1	acquire(ℓ_1)		
2	acquire(ℓ_2)		
3	release(ℓ_2)		
4	acquire(ℓ_2)		
5	release(ℓ_2)		
6	release(ℓ_1)		
7		acquire(ℓ_3)	
8		acquire(ℓ_2)	
9		acquire(ℓ_1)	
10		release(ℓ_1)	
11		release(ℓ_2)	
12		release(ℓ_3)	
13			acquire(ℓ_3)
14			acquire(ℓ_1)
15			acquire(ℓ_2)
16			release(ℓ_2)
17			release(ℓ_1)
18			release(ℓ_3)



$t_1, \ell_2, \{\ell_1\}$
$\langle e_2, e_4 \rangle$

$t_2, \ell_1, \{\ell_2, \ell_3\}$
$\langle e_9 \rangle$

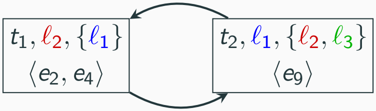
Potential deadlocks:

$\langle e_2, e_9 \rangle, \langle e_4, e_9 \rangle$

- Number of checks per cycle:
  - ↪ **Naive approach:** Exponential
  - ↪ **Sync-preserving deadlocks:** Linear

Interplay With Abstract Lock Graph

	t_1	t_2	t_3
1	acquire(ℓ_1)		
2	acquire(ℓ_2)		
3	release(ℓ_2)		
4	acquire(ℓ_2)		
5	release(ℓ_2)		
6	release(ℓ_1)		
7		acquire(ℓ_3)	
8		acquire(ℓ_2)	
9		acquire(ℓ_1)	
10		release(ℓ_1)	
11		release(ℓ_2)	
12		release(ℓ_3)	
13			acquire(ℓ_3)
14			acquire(ℓ_1)
15			acquire(ℓ_2)
16			release(ℓ_2)
17			release(ℓ_1)
18			release(ℓ_3)



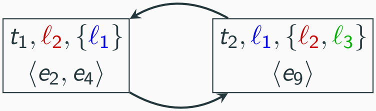
Potential deadlocks:

$\langle e_2, e_9 \rangle, \langle e_4, e_9 \rangle$

- Number of checks per cycle:
  - ↪ **Naive approach:** Exponential
  - ↪ **Sync-preserving deadlocks:** Linear
- Time spent per check:
  - ↪ **All deadlocks:** Exponential
  - ↪ **Sync-preserving deadlocks:** Linear

Interplay With Abstract Lock Graph

	t_1	t_2	t_3
1	acquire(ℓ_1)		
2	acquire(ℓ_2)		
3	release(ℓ_2)		
4	acquire(ℓ_2)		
5	release(ℓ_2)		
6	release(ℓ_1)		
7		acquire(ℓ_3)	
8		acquire(ℓ_2)	
9		acquire(ℓ_1)	
10		release(ℓ_1)	
11		release(ℓ_2)	
12		release(ℓ_3)	
13			acquire(ℓ_3)
14			acquire(ℓ_1)
15			acquire(ℓ_2)
16			release(ℓ_2)
17			release(ℓ_1)
18			release(ℓ_3)



Potential deadlocks:

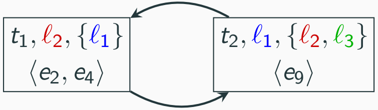
$\langle e_2, e_9 \rangle, \langle e_4, e_9 \rangle$

- Number of checks per cycle:
  - ↪ **Naive approach:** Exponential
  - ↪ **Sync-preserving deadlocks:** Linear
- Time spent per check:
  - ↪ **All deadlocks:** Exponential
  - ↪ **Sync-preserving deadlocks:** Linear

Overall Linear

Interplay With Abstract Lock Graph

	t_1	t_2	t_3
1	acquire(ℓ_1)		
2	acquire(ℓ_2)		
3	release(ℓ_2)		
4	acquire(ℓ_2)		
5	release(ℓ_2)		
6	release(ℓ_1)		
7		acquire(ℓ_3)	
8		acquire(ℓ_2)	
9		acquire(ℓ_1)	
10		release(ℓ_1)	
11		release(ℓ_2)	
12		release(ℓ_3)	
13			acquire(ℓ_3)
14			acquire(ℓ_1)
15			acquire(ℓ_2)
16			release(ℓ_2)
17			release(ℓ_1)
18			release(ℓ_3)



Potential deadlocks:

$\langle e_2, e_9 \rangle, \langle e_4, e_9 \rangle$

- Number of checks per cycle:
  - ↪ **Naive approach:** Exponential
  - ↪ **Sync-preserving deadlocks:** Linear
- Time spent per check:
  - ↪ **All deadlocks:** Exponential
  - ↪ **Sync-preserving deadlocks:** Linear

Overall Linear

How much precision have we lost?

Experimental Results - Offline

acm Artifacts Available V1.1

acm Artifacts Evaluated Reusable V1.1

- Implemented *Sync-preserving Offline*
  - ↪ Postmortem analysis
- Compared with *SeqCheck* and *Dirk*
- 48 benchmark traces
  - ↪ Based on standard Java benchmark suites

	Dirk	*SeqCheck*	**Sync-preserving Offline**
Total Deadlocks	35	40	**40**
Total Time	$>$ 1000 *minutes*	46 *minutes*	**3 minutes**

Experimental Results - Offline

- Implemented *Sync-preserving Offline*
  - ↪ Postmortem analysis
- Compared with *SeqCheck* and *Dirk*
- 48 benchmark traces
  - ↪ Based on standard Java benchmark suites

	Dirk	*SeqCheck*	**Sync-preserving Offline**
Total Deadlocks	35	40	**40**
Total Time	$>$ 1000 *minutes*	46 *minutes*	**3 minutes**

- **False negative analysis:** Only one actual deadlock is missed!
  - ↪ Based on the standard notion of valid reorderings

Online Algorithm

- Online setting → On-the-fly analysis
- No predictive online method
- Non-predictive online techniques:
  ↪ Schedule fuzzing
- **Our work:**
  ↪ Prediction + schedule fuzzing

Experimental Results - Online

- Implemented *Sync-preserving Online*
- Compared with *DeadlockFuzzer*
- 38 benchmarks
  ↪ Based on standard Java benchmark suites

	DeadlockFuzzer	**Sync-preserving Online**
Total Deadlock Hits	2076	**7633**
Total Unique Deadlocks	42	**49**

Conclusion

This work:

- **Complexity characterization:**
  - ↪ Finding potential deadlocks is intractable
  - ↪ Realizing potential deadlocks is intractable

Conclusion

This work:

- **Complexity characterization:**
  - ↪ Finding potential deadlocks is intractable
  - ↪ Realizing potential deadlocks is intractable
- **Sync-preserving deadlocks**:
  - ↪ Achieves efficiency and high precision
  - ↪ Outperforms state-of-the-art

Conclusion

This work:

- **Complexity characterization:**
  - ↪ Finding potential deadlocks is intractable
  - ↪ Realizing potential deadlocks is intractable
- **Sync-preserving deadlocks**:
  - ↪ Achieves efficiency and high precision
  - ↪ Outperforms state-of-the-art

Thank you!