Pathological Interaction of Locks with Transactional Memory

Haris Volos, Neelam Goyal, Michael M. Swift

Multifacet Project
(www.cs.wisc.edu/multifacet)

Computer Sciences Department
University of Wisconsin—Madison

Brain teaser

- What causes the following to deadlock in LogTM?

```
...
atomic {
 ...
 memcmp(s1, s2, n);
 ...
}
```

Dynamic Linker lock

Brain teaser

- What causes the following to deadlock in LogTM?

Thread 1

```
...
atomic {
 ...
 memcmp();
  elf_bndr()
   enter()
    rt_mutex_lock(&rtldlock)
```

Thread 2

```
...
free();
 elf_bndr()
  enter()
   rt_mutex_lock(&rtldlock)
   ...
   ...
   rt_mutex_unlock(&rtldlock)
```

Not unique to LogTM

Executive Summary

- Problem
  - Interaction of locks with transactional memory
  - Legacy code uses locks
- Interaction Pathologies
  - Identified five pathologies
  - Affect a spectrum of TM systems
- Transaction-safe Lock (TxLock)
  - Execute lock-based critical sections inside transactions
  - Modified OpenSolaris libC

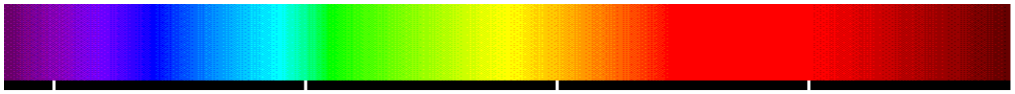
Outline

- **Interaction Pathologies**
- Transaction-Safe Lock (TxLock)
- Conclusion

Interaction Pathologies

- Pathological interaction
  - Correct code making proper use of locks breaks under transactions
- Identified five Interaction Pathologies
  - Blocking
  - Deadlock
  - Livelock
  - Early Release
  - Invisible Locking

Affect a spectrum of TM systems

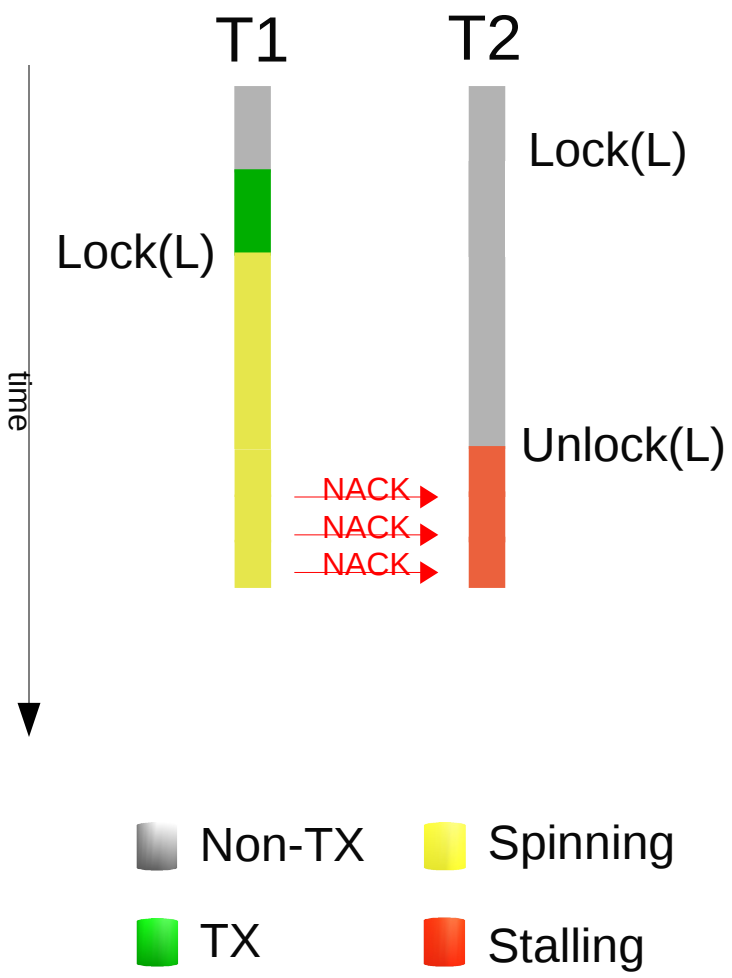


STM HTM

Transactional Memory Design

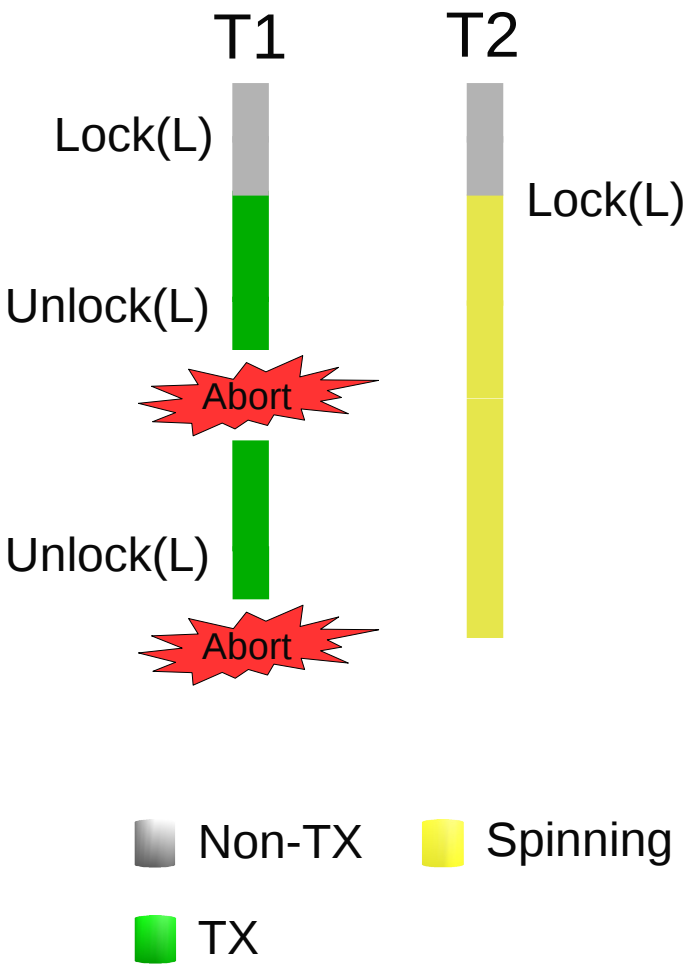
- Version Management
  - Eager, in-place update
  - Lazy, deferred update
- Conflict Detection
  - Eager, early
  - Lazy, late
- Conflict Resolution
  - Requester wins
  - Requester loses
- Atomicity
  - Strong
  - Weak

Deadlock



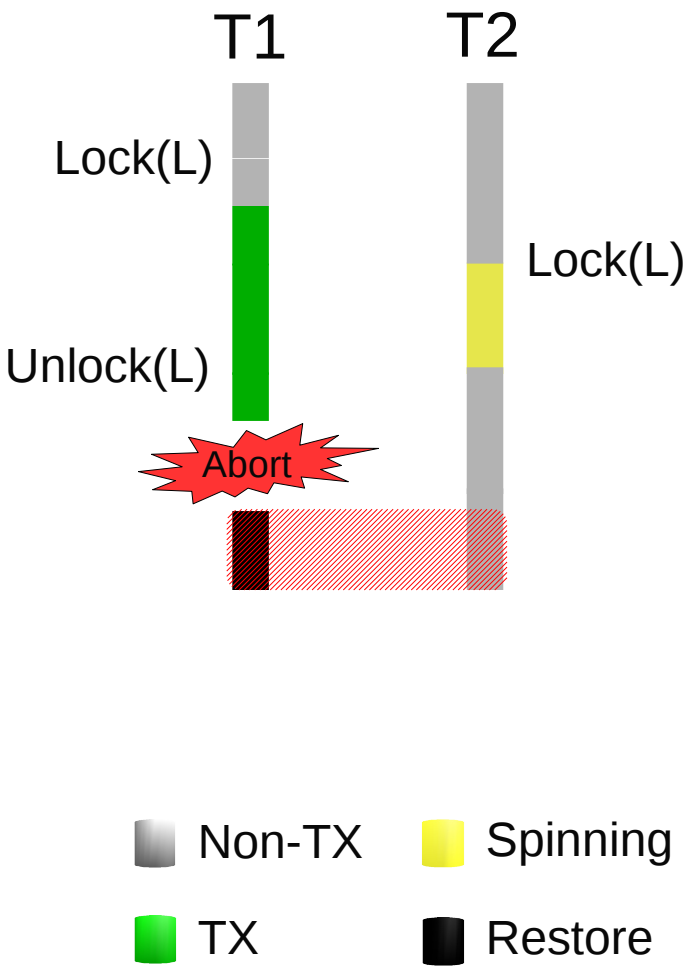
- Impacted Systems
  - Strong atomicity
  - Requester stalls
  - e.g LogTM, OneTM
- Cause
  - Two-level locking
  - No knowledge of lock dependency
- Effect
  - TM system stalls locks' logical owner

Livelock



- Impacted Systems
  - Strong atomicity
  - Requester wins
  - e.g Bulk, LTM
- Cause
  - Two-level locking
  - No knowledge of lock dependency
- Effect
  - TM system aborts locks' logical owner

Early Release



- Impacted Systems
  - Weak atomicity
  - Eager version management
  - e.g McRT-STM
- Cause
  - Restore values without lock held
- Effect
  - Lose lock-based mutual exclusion

Summary

Pathology	Atomicity	Version Management	Conflict Detection	Systems
Blocking	Strong	Any	Any	LogTM, Bulk
Deadlock	Strong	Eager	Eager	LogTM, OneTM
Livelock	Strong	Any	Any	Bulk, LTM
Early Release	Weak	Eager	Any	McRT-STM
Invisible Locking	Weak	Lazy	Any	TL2

Outline

- Pathologies
- **Transaction-Safe Lock (TxLock)**
- Conclusion

TxLock Design

- Goals
  - Execute lock-based critical section code safely
  - Low overhead in non-transactional path
- Design elements
  - Non-transactional lock operations
  - Deferred Unlock
  - Lock-aware Conflict Resolution

Testing & Acquiring a lock

- Escape when Testing & Acquiring
  - Prevents transactional conflicts
  - Allows to observe most recent lock value
- After Acquiring
  - Compensate action to explicitly restore lock value on abort

When waiting on a lock?

- When waiting
  - Inform Contention Manager about dependency

When releasing a lock?

- Coordinate release
  - TM system: Releases transactional locks at commit
  - TxLocks: Commit action to release logical-lock at commit

Implementation

- LogTM-SE
  - Wisconsin GEMS
- OpenSolaris C Library
  - POSIX adaptive mutex locks
  - 80 lines
- Contention Manager and Policy
  - 990 lines

Preliminary Results

- Experience
  - Able to resolve deadlocks in *libc malloc()*
- Quantitative
  - Development of workloads in progress
  - No rigorous performance evaluation
  - Overhead

Lock	Cycles
Solaris adaptive mutex	61
TxLock non-transactional	89
TxLock transactional	185

Low overhead
Higher overhead

Summary

- Problem
  - Legacy code uses locks
  - Interaction of locks with transactional memory
- Interaction Pathologies
  - Identified five pathologies
  - Affect a spectrum of TM systems
- Transaction-safe Lock (TxLock)
  - Execute lock-based critical sections inside transactions
  - Modified OpenSolaris libC

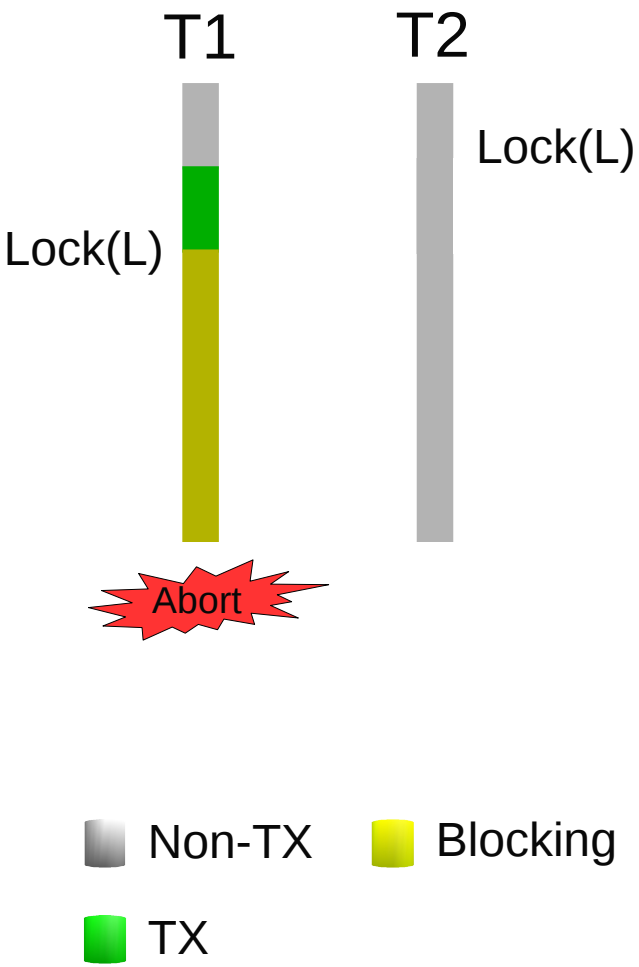
Questions

?

Backup Slides

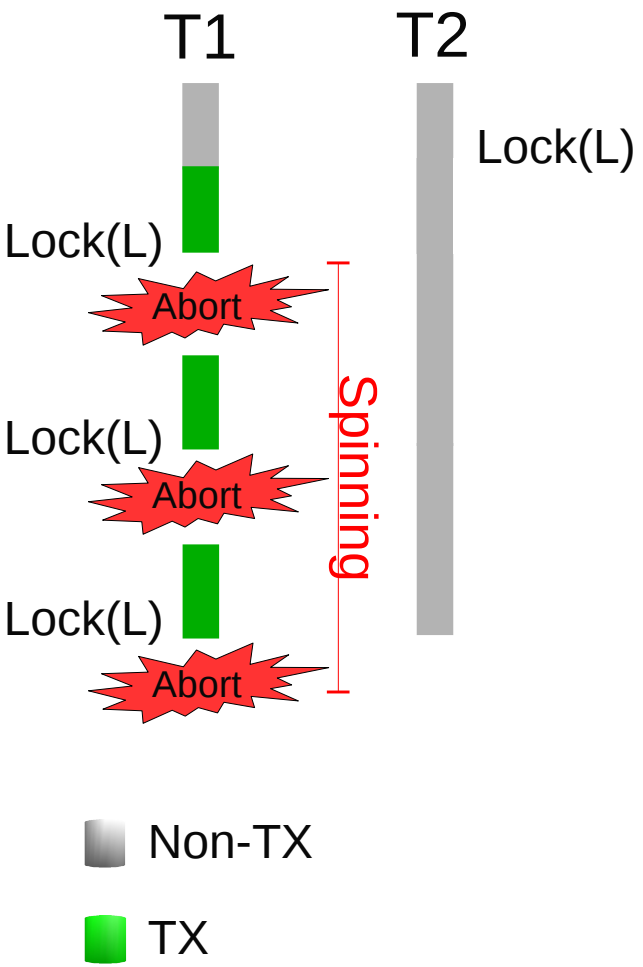
- TxLocks vs SpinLocks
- Rest of Pathologies
  - Blocking
  - Invisible Locking
- Implementation Details – lock
- Implementation Details – unlock
- Design Elements

Blocking



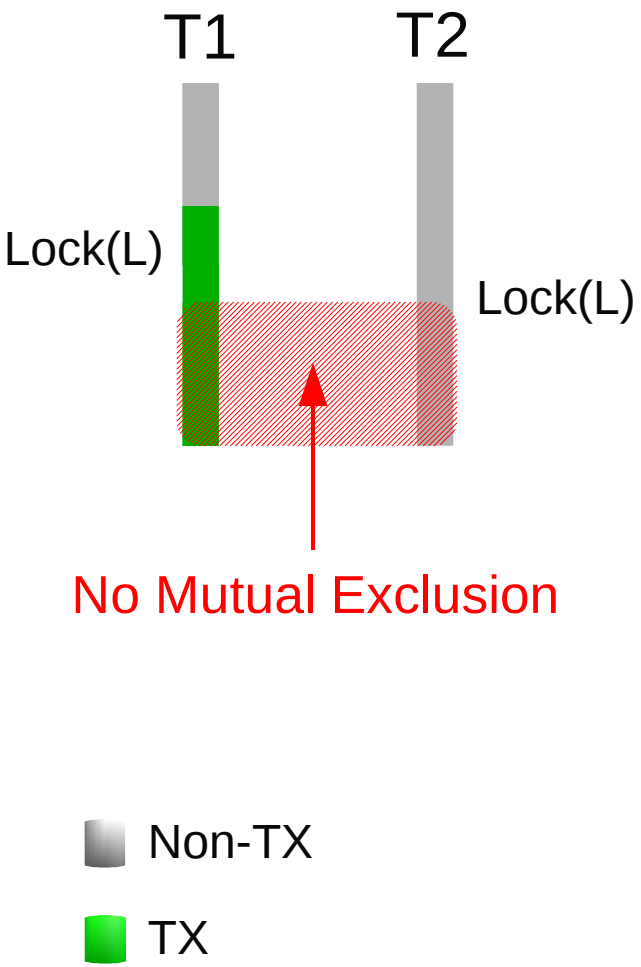
- Cause
  - No support to abort while sleeping
- Effect
  - Cannot abort blocked transaction safely

Blocking



- Cause
  - No support to abort while sleeping
- Effect
  - Cannot abort blocked transaction safely
  - Blocking degenerates into spinning
- Example Systems
  - LogTM
  - Bulk

Invisible Locking



- Cause
  - Weak atomicity
  - Lazy Version Management
- Effect
  - Lose lock-based mutual exclusion
- Example System
  - TL2

TxLocks vs cxspinlocks

- Programming interface
  - cxspinlocks: program using locks
  - TxLocks: program using transactions
- Optimistic concurrency
  - cxspinlocks: execute critical sections with optimistic concurrency
  - TxLocks: execute critical sections w/o optimistic concurrency
- Overhead
  - cxspinlocks: lock always introduces extra coherence traffic
  - TxLocks: pay extra cost when you acquire locks
- Hardware support
  - cxspinlocks: require special hardware: `xcas` and `xtest`
  - TxLocks: no special hardware; applicable to both HTM & STM

Implementation Details - `lock`

```
1.  void lock(txlock_t* mp) {
2.❶   BEGIN_ESCAPE;
3.    if (set_lock_byte(&mp->txlock_lockw) == 0) {
4.      mp->mutex_owner = (uintptr_t)self;
5.      goto lock_acquired;
6.    }
7.    if (mp->mutex_owner == self) {
8.     mp->rcount++;
9.     goto lock_acquired_noTS;
10.   }
11.   if (txlock_trylock_adaptive(mp) != 0) {
12.❹   if (txlock_lock_queue(self, mp) == ABORT) {
13.❶      END_ESCAPE;
14.❹     ABORT_TRANSACTION
15.     }
16.   }
17.   lock_acquired:
18.❸  mp->timestamp = xact_timestamp();
19.   lock_acquired_noTS:
20.   if (in_xact()) {
21.❶   register_compensating_action(
22.❶            txlock_unlock_impl, mp);
23.   }
24.❶ END_ESCAPE;
25. }
```

Implementation Details - unlock

```
26. void unlock(txlock_t* mp) {
27.❶  BEGIN_ESCAPE;
28.     if (mp->mutex_owner == self) {
29.       mp->rcount++;
30.❶      END_ESCAPE;
31.       return;
32.     }
33.     if (in_xact()) {
34.❷      register_commit_action(
35.❷          txlock_unlock_impl, mp);
36.     } else {
37.       txlock_unlock_impl(mp);
38.     }
39.❶  END_ESCAPE;
40. }
```

1. Non-transactional Lock Operations

- Prevents
  - Invisible Locking
  - Livelock
  - Some Deadlock cases
- Method
  - Escape transactional control of lock code
  - Restore lock value through compensating action

2. Deferred Unlock

- Prevents
  - Early Release
- Method
  - Do not release lock in place but at commit
  - Recursive locking to avoid deadlock with itself

3. Lock-aware Conflict Resolution

- Prevents
  - Deadlock
  - Livelock
- Contention Manager
  - Is provided with transactional and lock dependencies
  - Builds full dependence graph
  - Finds correct victim transaction

4. Block/Wake-up Protocols

- Prevents
  - Blocking
- Protocols
  - Locks and contention manager cooperate to wake-up a blocked transaction
  - Use **Abort/Block** protocol when **CANNOT** suspend a TX
  - Use **Block/Abort** protocol when **CAN** suspend a TX