Performance Pathologies in Hardware Transactional Memory

David A. Wood

Multifacet Project, Univ. of Wisconsin—Madison

December, 2007 @ IIT-Kanpur

"The distant threat has come to pass… parallel computers are the inexorable next step in the evolution of computers."

— James Larus, Microsoft, & Ravi Rajwar, Intel
In *Transactional Memory*, Morgan/Claypool, 2007

Why Locks are Hard

- Coarse-grain locks
  - Simple
  - No deadlock
  - Few data races
  - Limited concurrency

- Fine-grain locks
  - Greater concurrency
  - Greater code complexity
  - Potential deadlocks
    - Not composable
  - Potential data races
    - Which lock to lock?

```
// WITH FINE-GRAIN LOCKS
void move(T s, T d, Obj key){
  LOCK(s);
  LOCK(d);
  tmp = s.remove(key);
  d.insert(key, tmp);
  UNLOCK(d);
  UNLOCK(s);
}
```

Thread 0

```
move(a, b, key1);
```

Thread 1

```
move(b, a, key2);
```

DEADLOCK!

Transactional Memory (TM)

- Database systems are the parallel programming success story
  - Declarative programming model (e.g., SQL)
  - Transactions enforce ACID properties
    - Serializability facilitates sequential reasoning
  - Even naïve programmers (often) get correct parallel execution

- TM makes shared-memory synchronization declarative
  - Programmer says
    - "I want this atomic"
  - TM system
    - "Makes it so"

```
void move(T s, T d, Obj key){
  atomic {
     tmp = s.remove(key);
     d.insert(key, tmp);
  }
}
```

Some Transaction Terminology

Transaction: State transformation that is:
(1) Atomic (all or nothing)
~~(2) Consistent~~
(3) Isolated (serializable)
~~(4) Durable (permanent)~~

Commit: Transaction successfully completes

Abort: Transaction fails & must restore initial state

Read (Write) Set: Items read (written) by a transaction

Conflict: Two concurrent transactions conflict if either's write set overlaps with the other's read or write set

Nested Transactions for Software Composition

- Modules expose interfaces, NOT implementations
- Example
  - Insert() calls getID() from within a transaction
  - The getID() transaction is nested inside the insert() transaction

```
void insert(object o){
   // parent TX
   atomic {
      t.insert(getID(), o);
   }
}
```

```
int getID() {
   // child TX
   atomic {
       id = global_id++;
   }
   return id;
}
```

Closed Nesting

Child transactions remain isolated until parent commits

- On Commit child transaction is merged with its parent
- Flat
  - Nested transactions “flattened” into a single transaction
  - Only outermost begins/commits are meaningful
  - Any conflict aborts to outermost transaction
- Partial rollback
  - Child transaction can be aborted independently
  - Can avoid costly re-execution of parent transaction

Implementing TM: Software, Hardware, or Hybrid

- Software TM (STM) Implementations
  - Exist today
  - Currently slower than locks
  - Always slower than hardware?
- Hybrid TM (HyTM) Implementations
  - Software TM with best-effort hardware acceleration
  - Next step, supported by Sun in the Rock processor
- Hardware TM (HTM) Implementations
  - Fast, leverages cache coherence & speculative execution
  - Likely long-run winner for performance reasons
  - But finite hardware presents OS virtualization challenges

Implementing Transactional Memory

- (Data) Version Management
  - Keep **old** values for abort **AND new** values for commit
  - **Eager**: record old values "elsewhere"; update "in place" ← Fast commit
  - **Lazy**: update "elsewhere"; keep old values "in place"
- (Data) Conflict Detection
  - Find read-write, write-read or write-write conflicts among concurrent transactions
  - **Eager**: detect conflict on every read/write ← Less wasted work
  - **Lazy**: detect conflict at end (commit/abort)

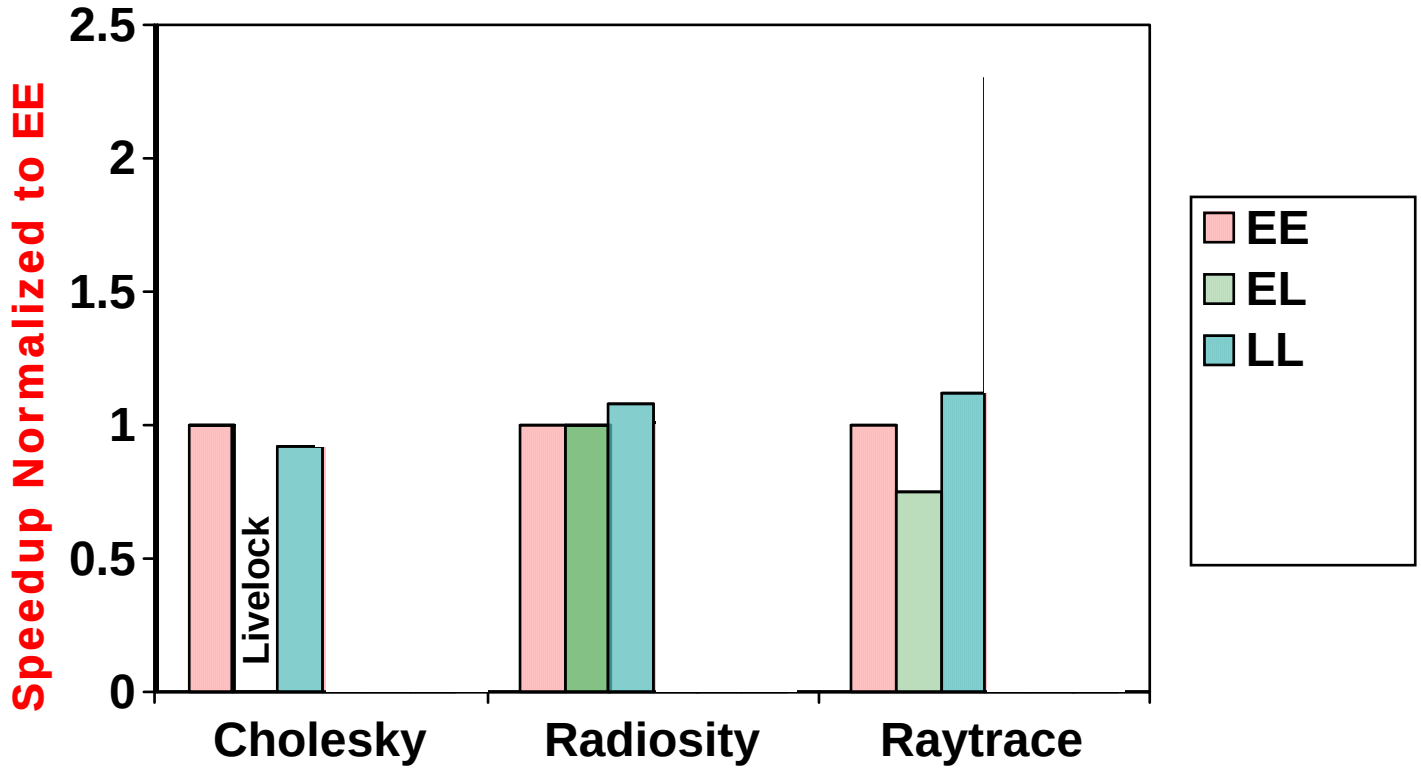
How Do Hardware TM Systems Differ?

	Version Management	
Conflict Detection	Lazy: buffer updates & move on commit	Eager: update “in place” after saving old values
Lazy: check on commit	Like Databases with Optimistic Conc. Ctrl. Stanford TCC Illinois Bulk LL	No HTMs (yet)
Eager: check before read/write	Herlihy/Moss TM MIT LTM Intel/Brown VTM EL	Like Databases with Conservative C. Ctrl. MIT UTM Wisconsin LogTM EE

Conflict Resolution: Ad Hoc

Executive Summary (1/2)

- Multiple Hardware Transactional Memory Designs
  - Best Performance ?

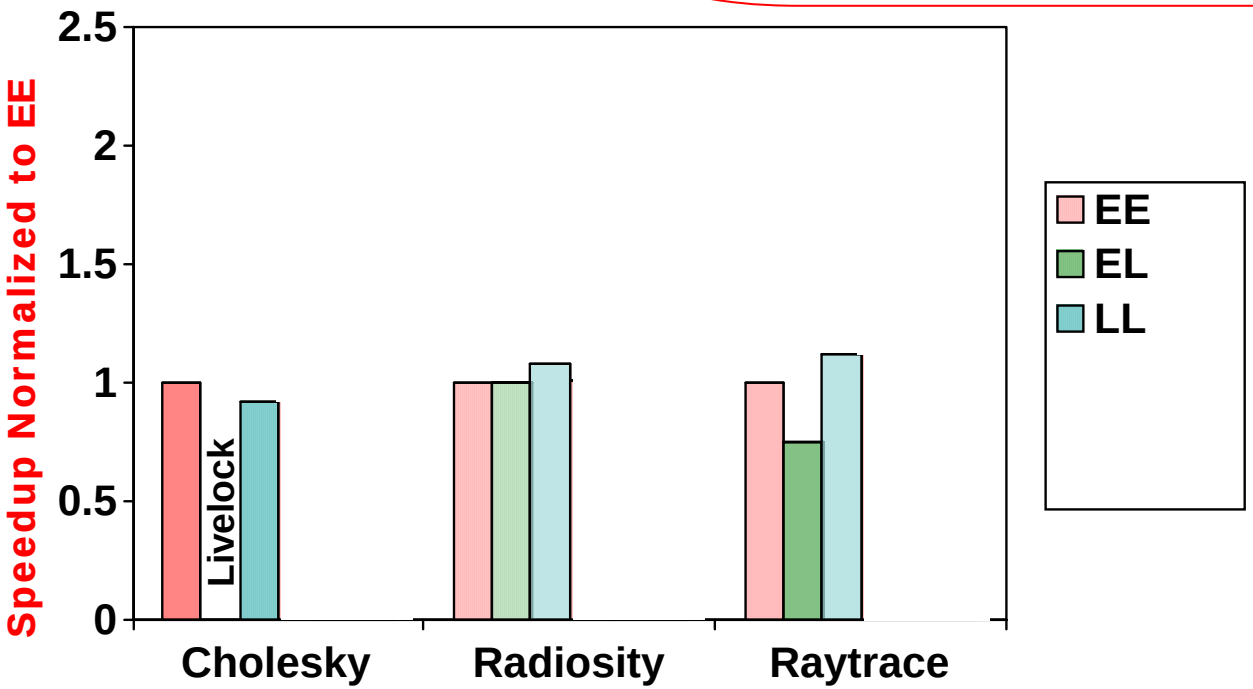


- Depends !

Executive Summary (2/2)

- Why?
  - Performance Pathologies
- What to do?
  - Better Conflict Resolution

pa●thol●o●gy: any deviation from a normal, healthy or efficient condition



Performance Pathologies

Performance

Talk Outline

- **Motivation**
- **Base HTM Results**

- Performance Pathologies
- Enhanced HTM Systems

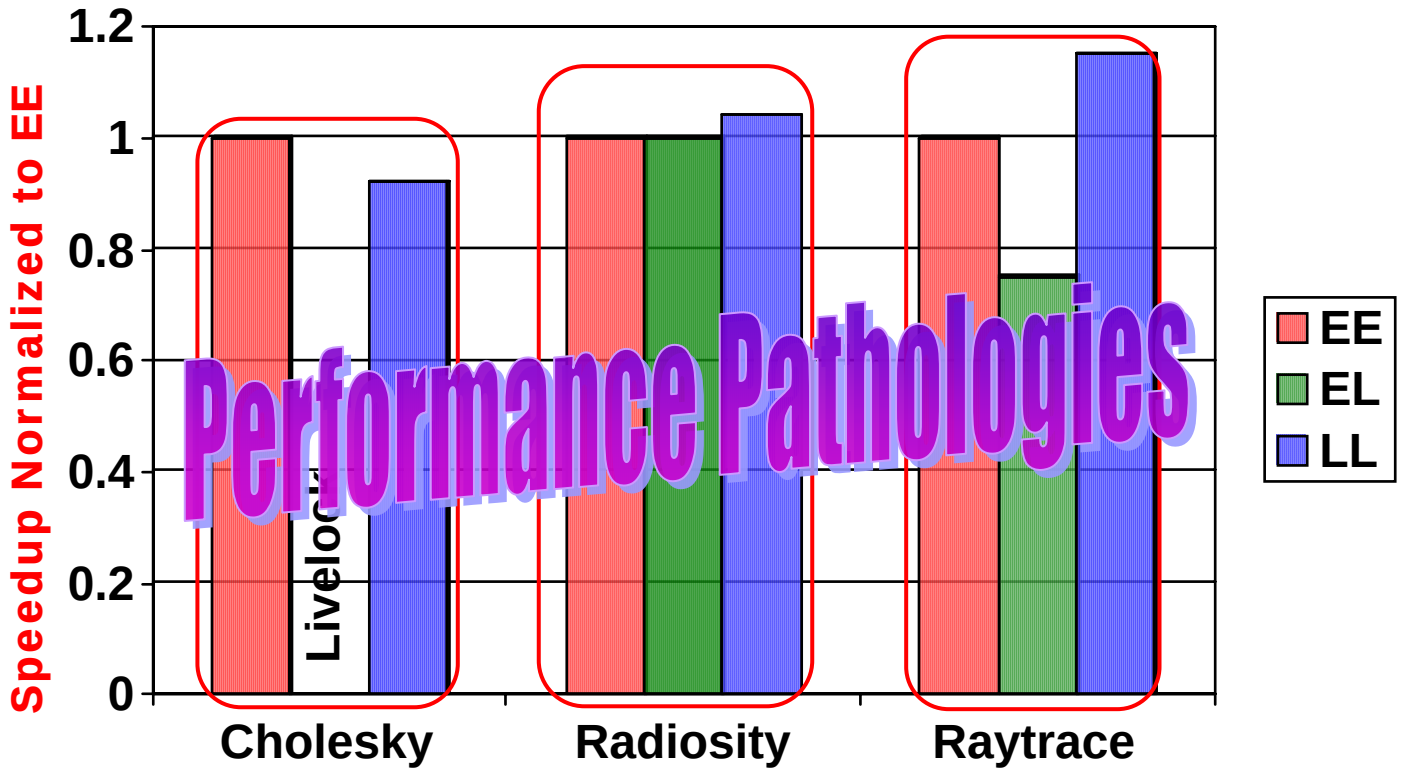
Motivation

- Which HTM system performs best?
- Different Assumptions
  - Hardware
    - Broadcast vs Directory Based
  - Software:
    - Continuous Transactions vs Critical Section Transactions
- How to compare?

Experimental Setup

- Base HTM Systems
  - LL
  - EL
  - EE
- Common Hardware Platform
  - 32 core CMP
  - Directory-Based Coherence
- Common TM Programming Model
  - Critical Section Based Transactions
- Common Workloads
  - 5 SPLASH + 2 Microbenchmarks (Btree, LFUCache)

And the winner is…



- Low Contention: Similar
- High Contention: Depends

Talk Outline

- Motivation
- Base HTM Results

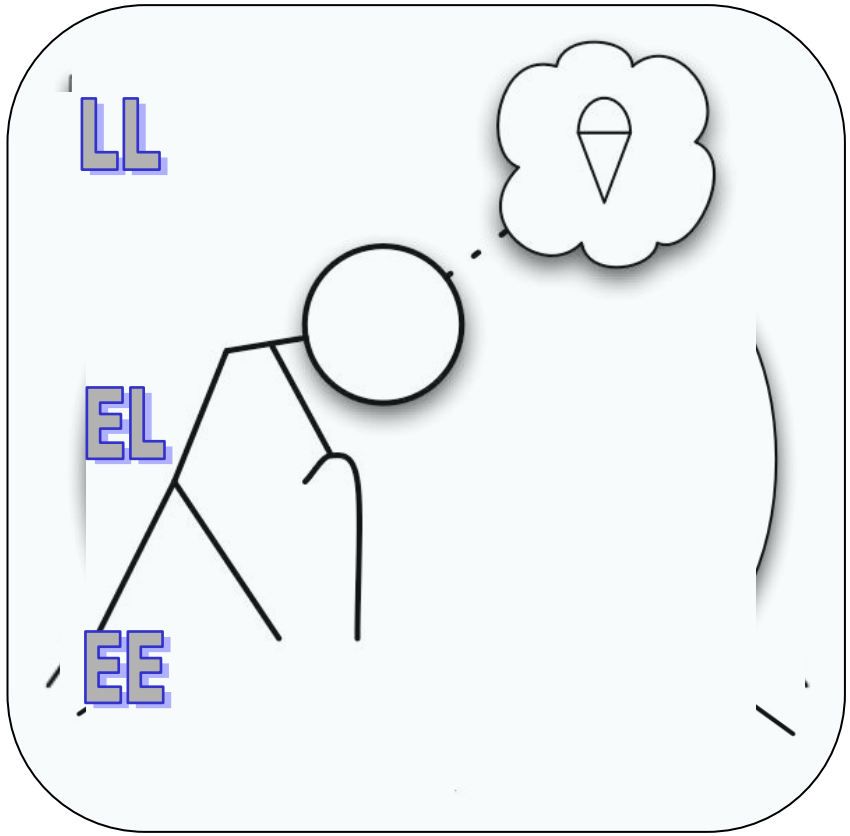
- **Performance Pathologies**
- **Enhanced HTM Systems**

Performance Pathologies

- StarvingElder
- SerializedCommit
- RestartConvoy

- FriendlyFire

- DuelingUpgrades
- FutileStall
- StarvingWriter



LL System

- **Conflict Detection**: Lazy

  Detect conflicts at commit time (Validation)

- **Version Management**: Lazy

  New value elsewhere, Update on commit

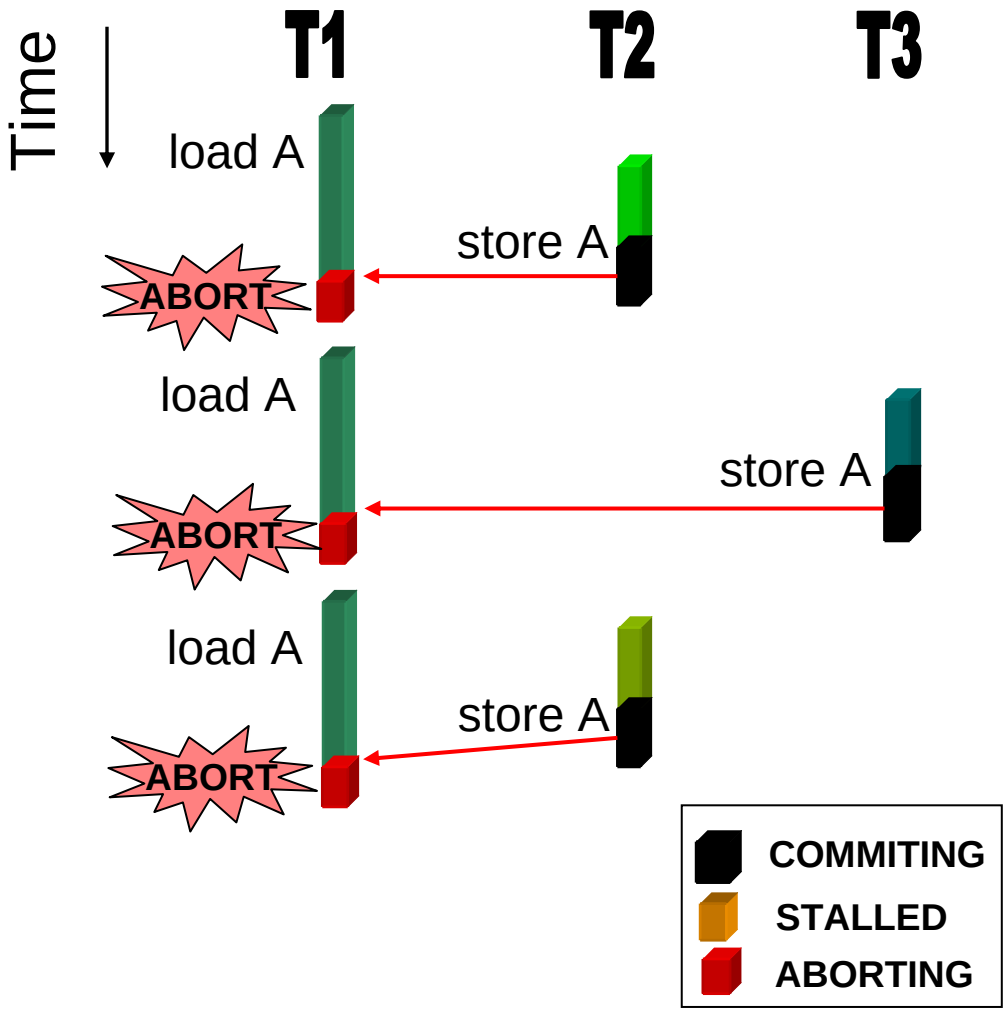
  \+ Abort

  \- Commit

- **Conflict Resolution**: Committer Wins

compare with TCC, Bulk

StarvingElder



CAUSE

Committer Wins

EFFECT(S)

- Starvation
- Load Imbalance

FIX

Elder gets Priority ?

LL Pathologies

- Pathology (% ExecutionTime)

	LL		
	Starving Elder	Serialized Commit	Restart Convoy
Raytrace	45	27	5.2

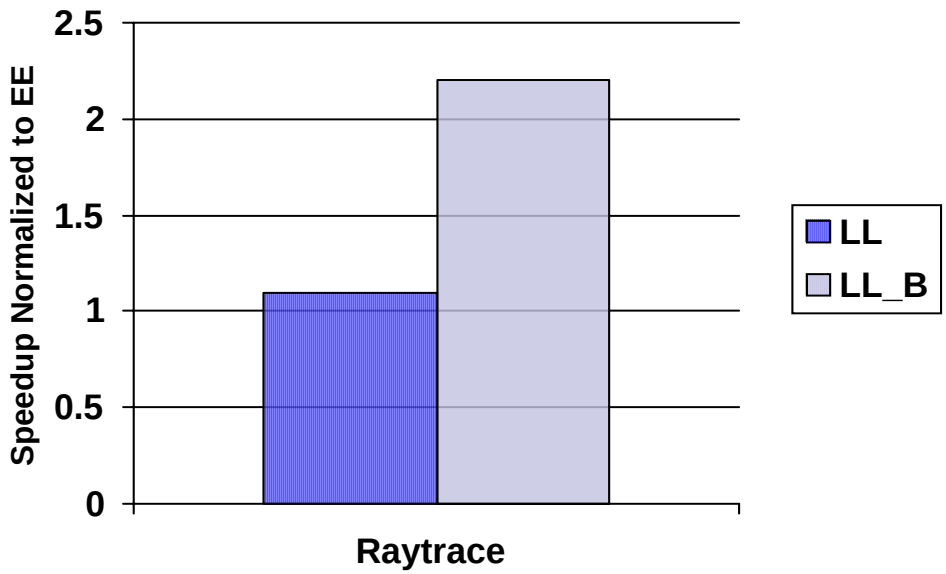
- Enhancement
  - Linear Backoff on Abort
  - Eliminate RestartConvoys

Enhanced LL (LL_B)

- Pathology (% ExecutionTime)

	LL		
	Starving Elder	Serialized Commit	Restart Convoy
Raytrace	45	27	5.2

- Performance



Performance Pathologies

- StarvingElder
- SerializedCommit
- RestartConvoy

LL

- **FriendlyFire**

EL

- DuelingUpgrades
- FutileStall
- StarvingWriter

EE

EL System

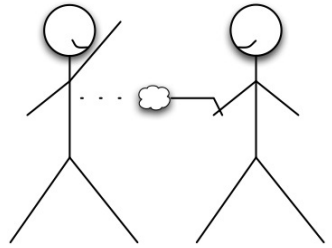
- **Conflict Detection**: Eager
  - Detect conflicts immediately
- **Version Management**: Lazy
  - New Value elsewhere, Update on commit

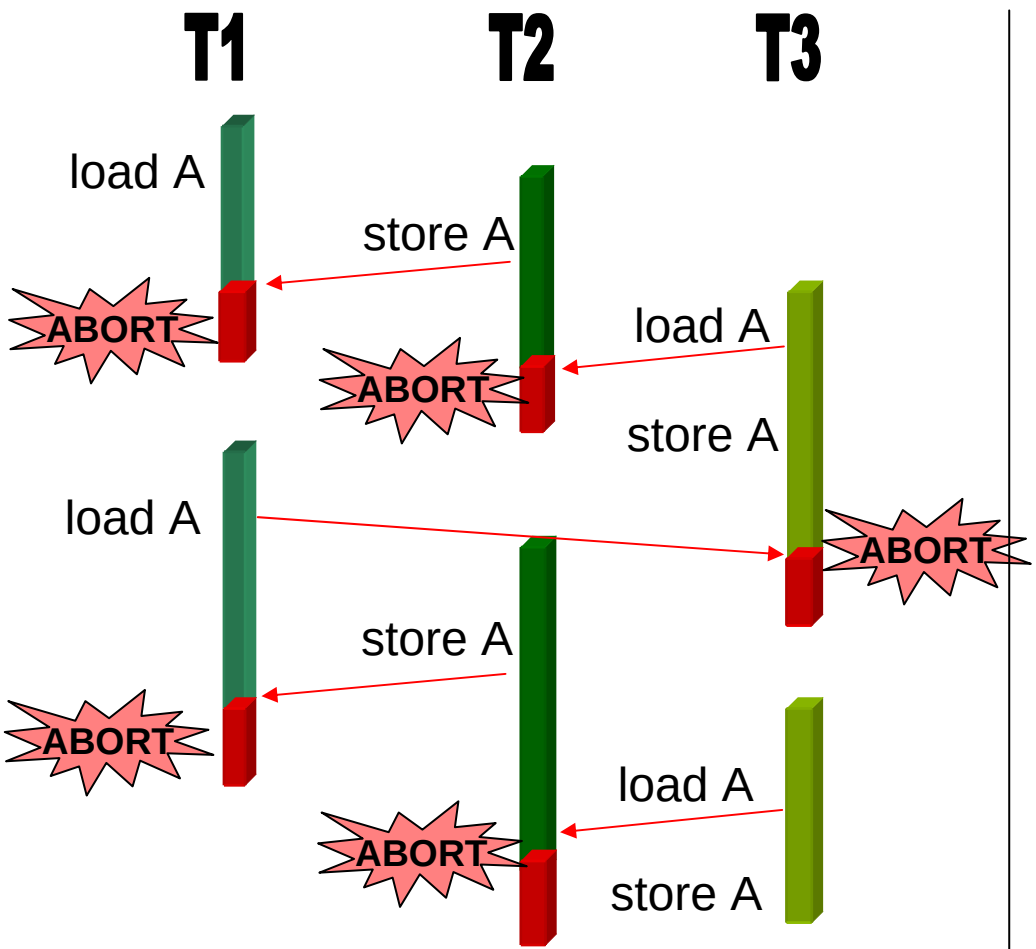
  \+ Abort

  \- Commit
- **Conflict Resolution**:
  - Requester Wins
  - Exponential Backoff on Abort

compare with HMTM, LTM

FriendlyFire





CAUSE

Requester Wins

EFFECT(S)

- Unnecessary Aborts
- Livelock

FIX

Priority Based Conflict Resolution ?

EL Pathologies

- Pathology (% ExecutionTime)

	EL
	Friendly Fire
Raytrace	73

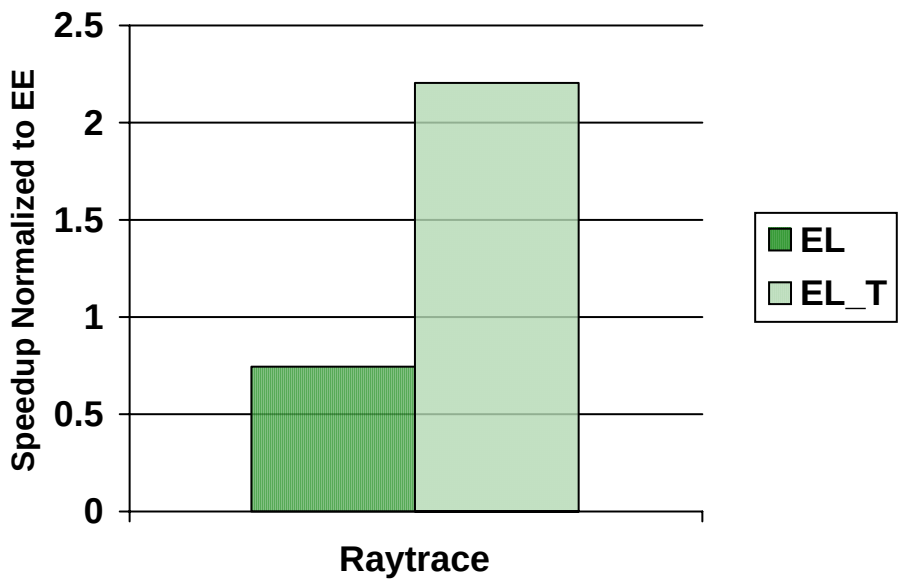
- Enhancement
  - Timestamp Based Conflict Resolution
  - Eliminate FriendlyFire

Enhanced EL (EL_T)

- Pathology (% ExecutionTime)

	EL
	Friendly Fire
Raytrace	73

- Performance



Performance Pathologies

- StarvingElder
- ~~SerializedCommit~~
- RestartConvoy

LL

- FriendlyFire

EL

- **DuelingUpgrades**
- ~~FutileStall~~
- StarvingWriter

EE

EE System

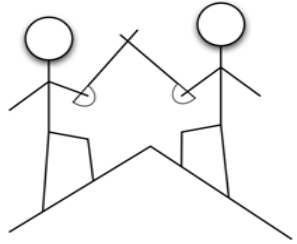
- **Conflict Detection**: Eager
  - Detect conflicts immediately
- **Version Management**: Eager
  - New Value in-place, Restore on Abort

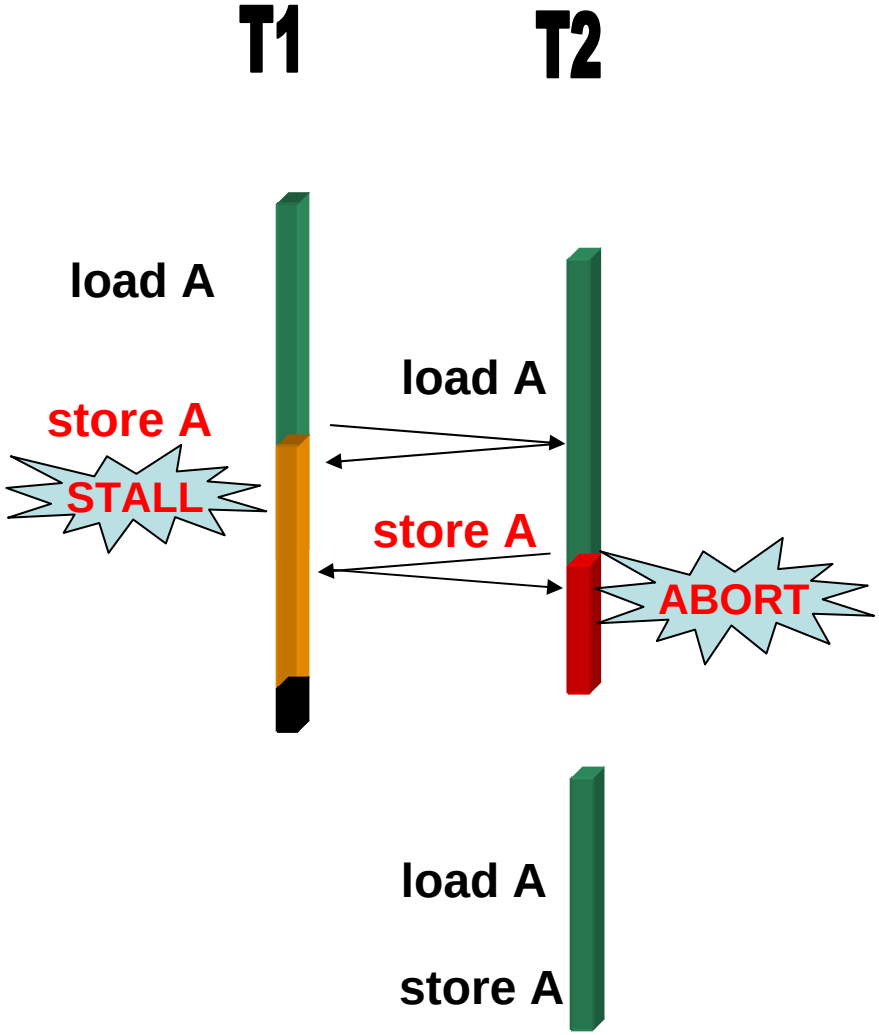
  \- Abort

  \+ Commit
- **Conflict Resolution**
  - Stall Requester
  - Abort Requester on possible deadlock (Conservative Deadlock Avoidance)

compare with LogTM

DuelingUpgrades





CAUSE

Conservative Deadlock Avoidance

EFFECT(S)

- Unnecessary Stall
- Unnecessary Abort

FIX

Acquire Store Permissions Early ?

EE Pathologies

- Pathology (% ExecutionTime)

	EE		
	% Execution Time		Aborts /Transaction
	FutileStall	Starving Writer	Dueling Upgrades
Raytrace	1.0	4.6	4.2

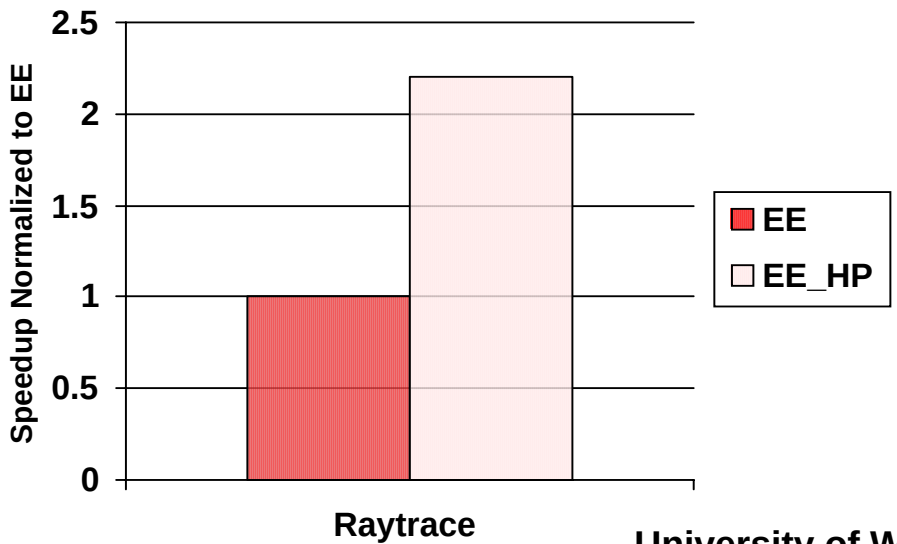
- Enhancements
  - DuelingUpgrades: Store-Set Predictor
  - StarvingWriter: Writer Aborts Readers immediately

Enhanced EE (EE_HP)

- Pathology (% ExecutionTime)

	EE		
	% Execution Time		Aborts /Transaction
	FutileStall	Starving Writer	Dueling Upgrades
Raytrace	1.0	4.6	4.2

- Performance



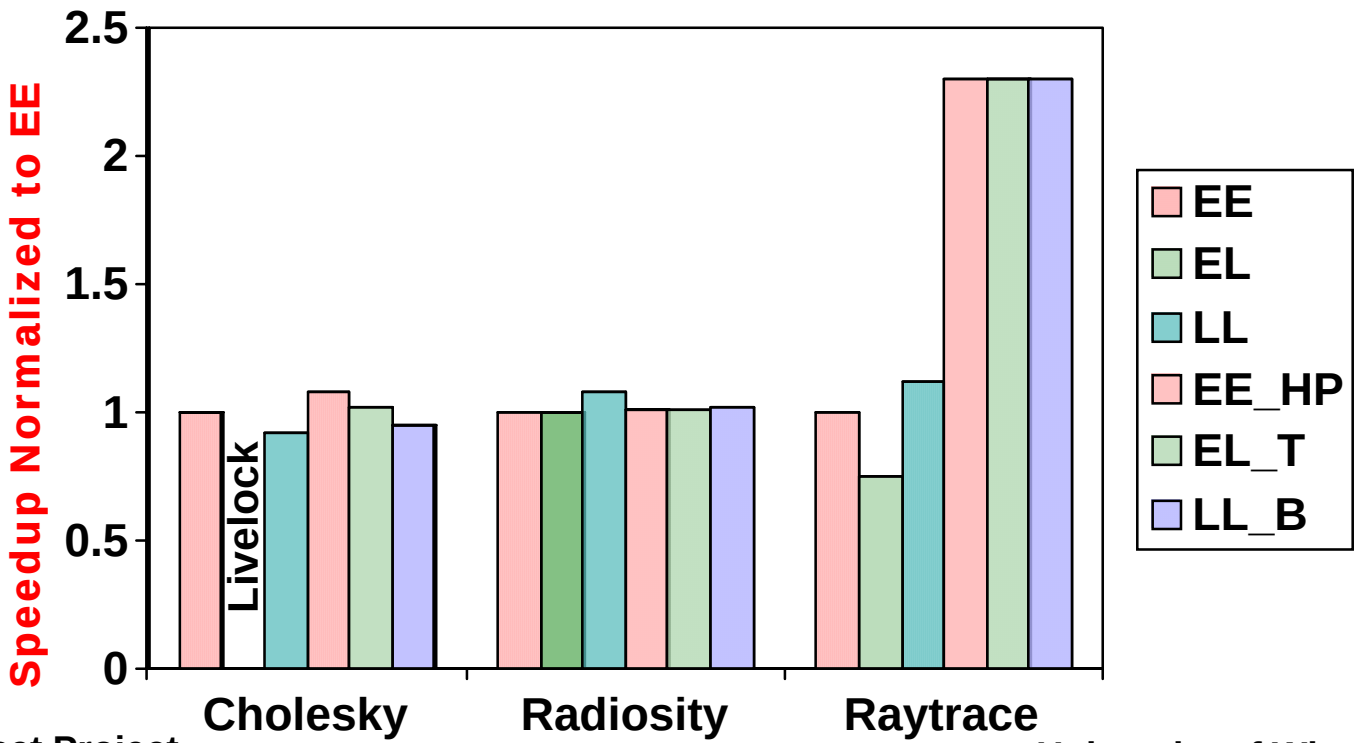
Caveats / FutureWork

	LL			LL_B		
	Restart Convoy	Starving Elder	Serialized Commit	Restart Convoy	Starving Elder	Serialized Commit
Mp3d	21	36	30	9.0	28	25

- High Contention Scenarios
  - Better Conflict Resolution
  - Unidentified Pathologies
- Better TM Workloads

Executive Summary

- TM promises to simplify parallel programming
- But, Performance Pathologies exist 
- Enhanced HTM systems help



Future Work

- Pathologies exist in emerging TM systems
  - Represent performance bugs on given platform
- TM converts correctness bugs to performance bugs
  - Large transactions eliminate data races
  - But serialize execution

- Need automatic performance debugger tools
  - Hard for programmers to analyze
  - Leverage machine learning techniques to find bugs
  - Work for emerging systems (Sun's Rock)
  - Work better for future systems (Wisconsin's LogTM)

Questions?

- For more information:
  - http://www.cs.wisc.edu/multifacet
  - Email to david@cs.wisc.edu