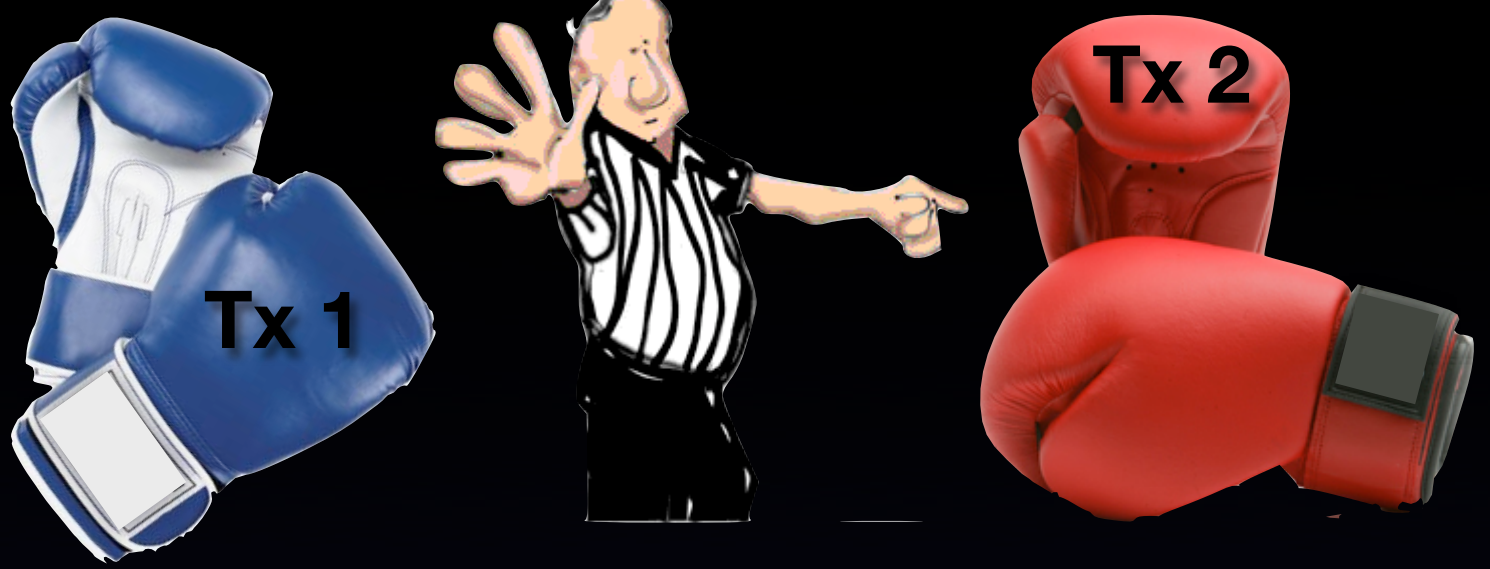


Refereeing Conflicts in Hardware Transactional Memory

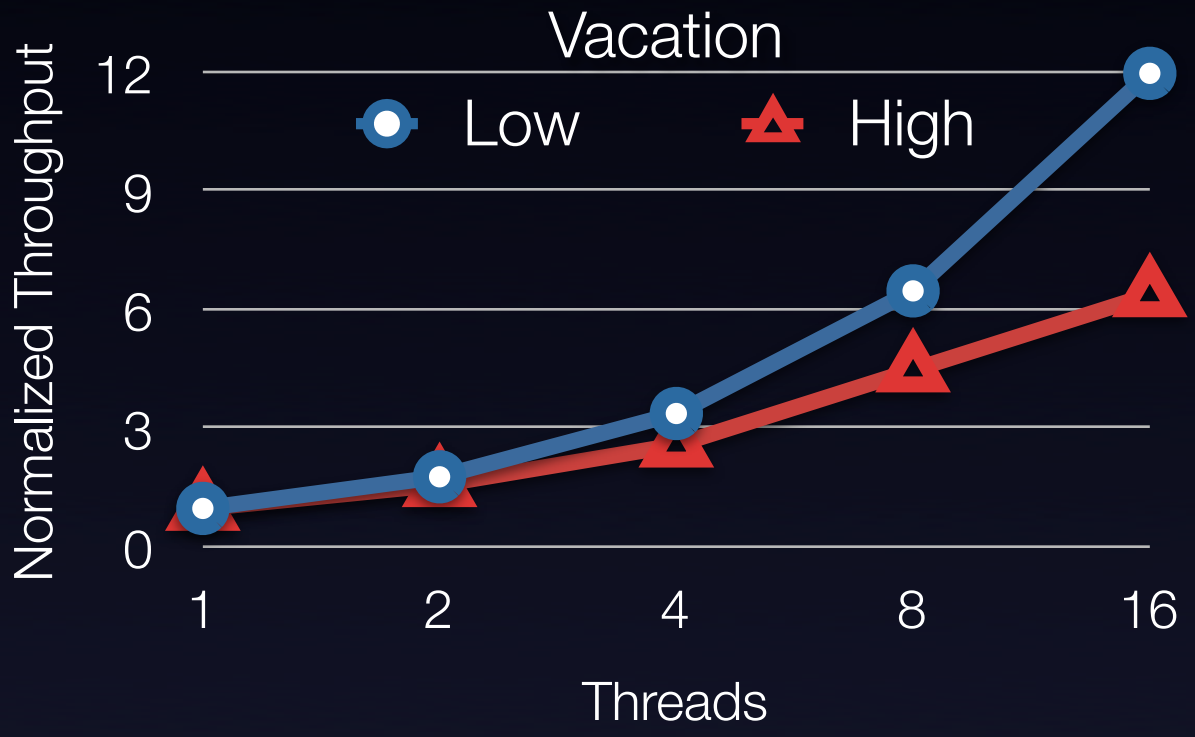
Arrvindh Shriraman
Sandhya Dwarkadas
Department of Computer Science

UNIVERSITY of ROCHESTER

Conflicts affect performance

Conflict: concurrent accesses to the same location from two different transactions where at least one is a write

- In the absence of conflicts, Hardware TM provides
  - low latency and high scalability
- With conflicts, performance can degrade significantly



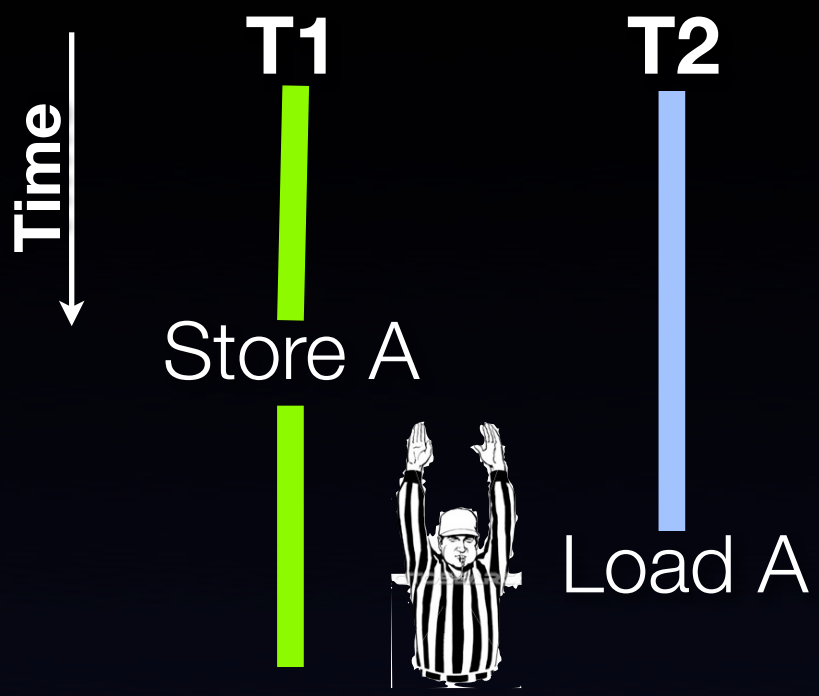
Conflicts can be common

Application	txs w/ conflicts
Bayes	85%
Delaunay	85%
Intruder	90%
Kmeans	15%
Vacation	73%
STMBench7	68%

- We anticipate that as TM becomes popular
  - large and long transactions will become common
  - new intricate sharing patterns will introduce conflicts

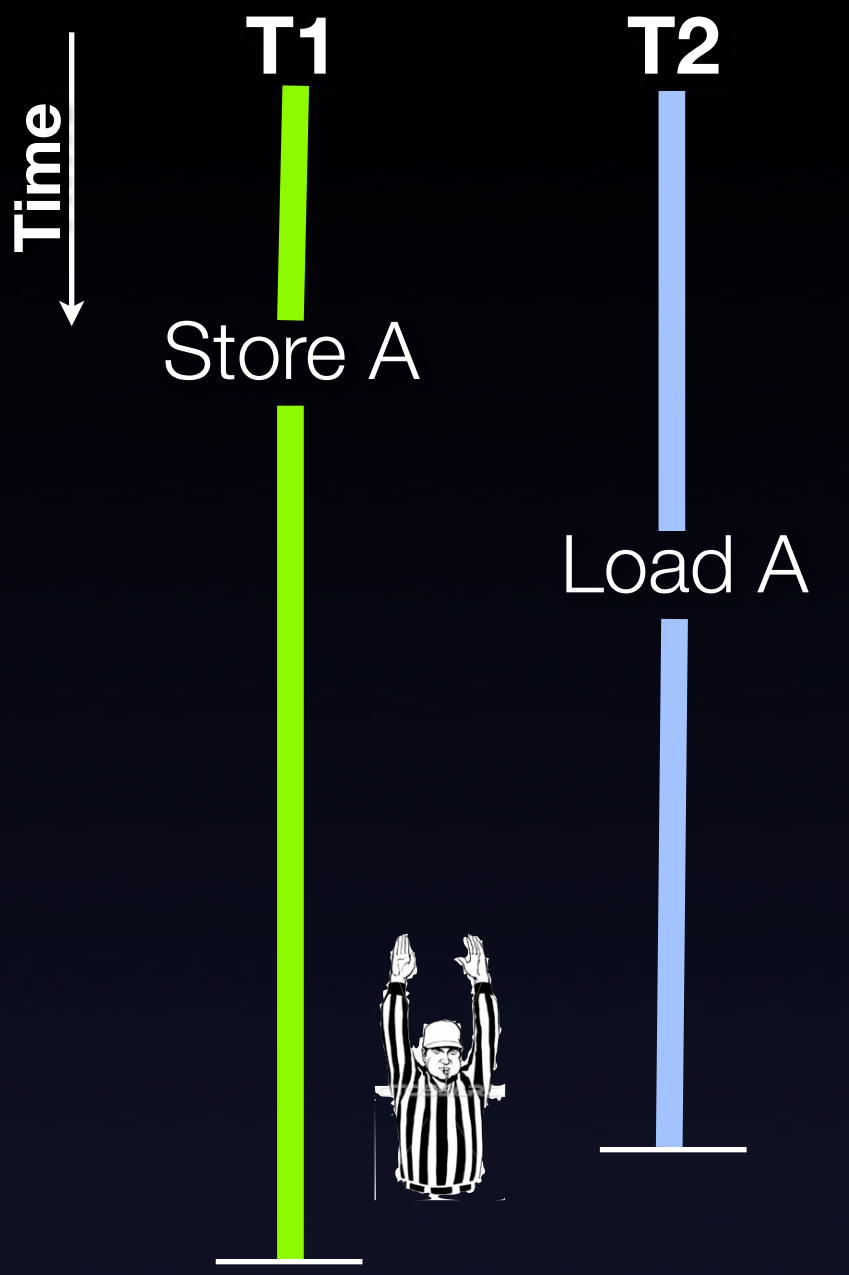
Conflict Management Primer

Conflict Management Primer



- Conflict Type
  - what type of accesses ?
  - read-write, write-read, write-write
- Conflict Detection
  - when to resolve ?
  - Eager (at access), Lazy (at commit)

Conflict Management Primer



- Conflict Type
  - what type of accesses ?
  - read-write, write-read, write-write
- Conflict Detection
  - when to resolve ?
  - Eager (at access), Lazy (at commit)
- Contention Management
  - How to choose loser ?
  - priority, timestamp, etc.
- Action
  - What action to take ?
  - stall, abort self, abort other etc.

Our Contributions

- Comprehensive study of policy in HTMs
  - conflict detection and conflict management interplay
  - quantify effect on application performance
- Is Lazy better than Eager ?
  - can we do better ?
- How does the contention manager help ?
  - is it important ?

Experimental Platform

- FlexTM [ISCA'08]
  - allows conflict detection to be controlled in software
  - permits pluggable software contention managers
- TM Hardware: 16 core CMP, Private L1s, Shared L2
  - signatures for conflict detection
  - private L1s for speculative buffering
    - overflow handled by hardware controller
  - transaction commit protocol
    - allows parallel transaction commits
    - no centralized arbiter

Workloads

- TM Workloads
  - STAMP (Stanford)
  - STMBench7 database (EPFL)
  - Web-cache and Graph stress tests (U.Rochester)
- TM Policies
  - Conflict detection: Lazy, Eager, and Mixed
  - Contention Management: w/ and w/o stalling, timestamps, access sets, aborts

Is Lazy better than Eager ?

Can we do better ? : Mixed

Is the contention manager important ?

Conflict Detection

- Eager (manages conflict at access time)

Goal: If transactions can't commit together, save work

  - to progress, transactions abort enemies
  - transactions can stall and try to elide the conflict

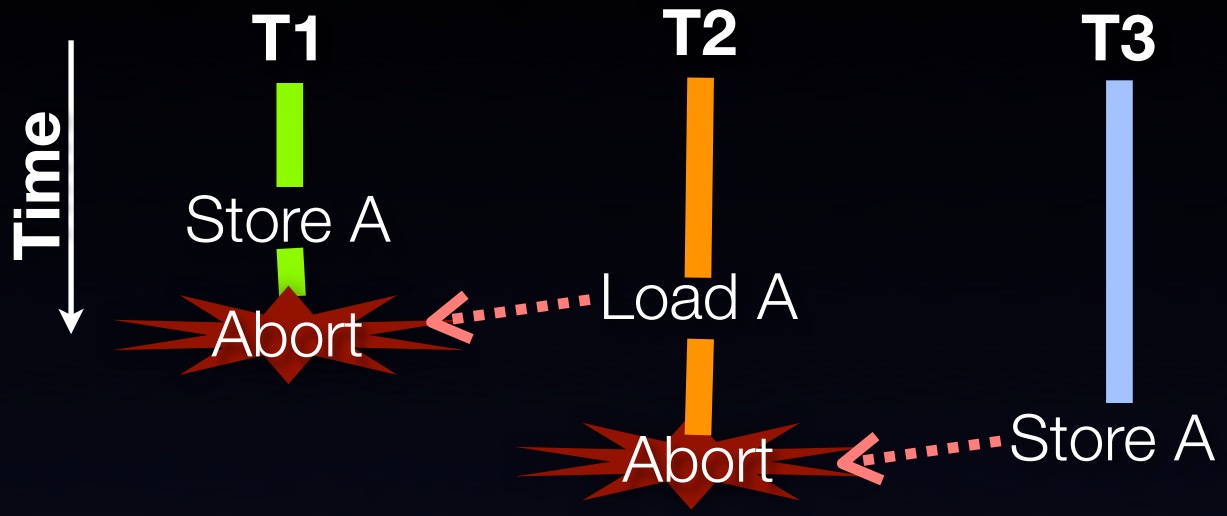
- Lazy (manages conflict at commit)

Goal: postpone detection hoping conflict disappears

  - to commit, writers abort enemies
  - writers can stall to elide reader conflicts

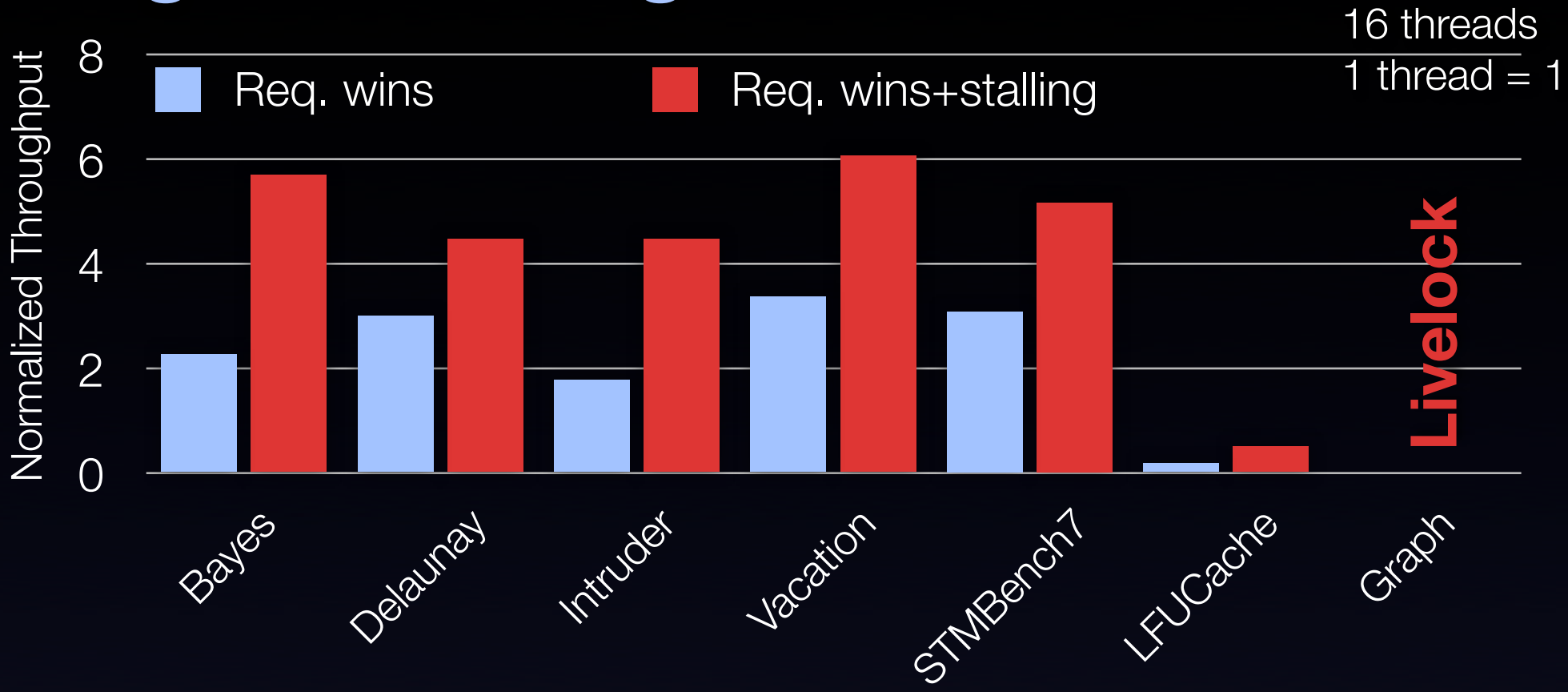
Eager's Performance Limitations

- Futile aborts waste work and hinder progress
  - stalling access may help avoid the conflict



- Inability to overlap conflicting transactions

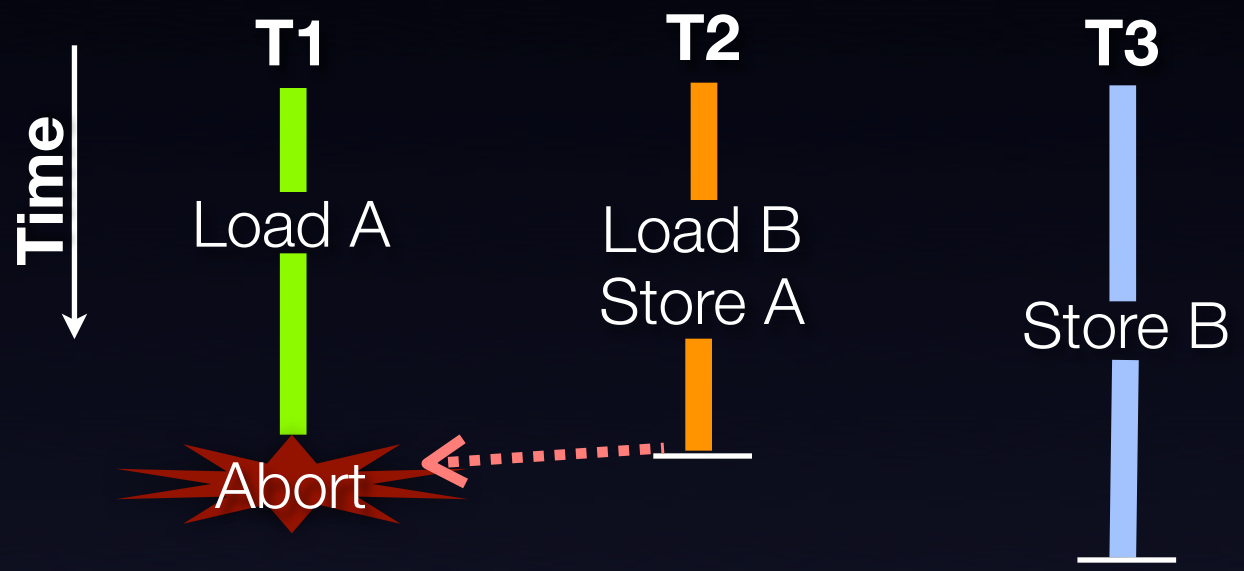
Eager w/ Stalling



- Can reduce occurrence of futile aborts (livelock ?)
  - reduces wasted work due to aborts
- Is it good enough ?
  - cannot exploit concurrency in application

Lazy's Benefits (1/2)
Small Contention Window

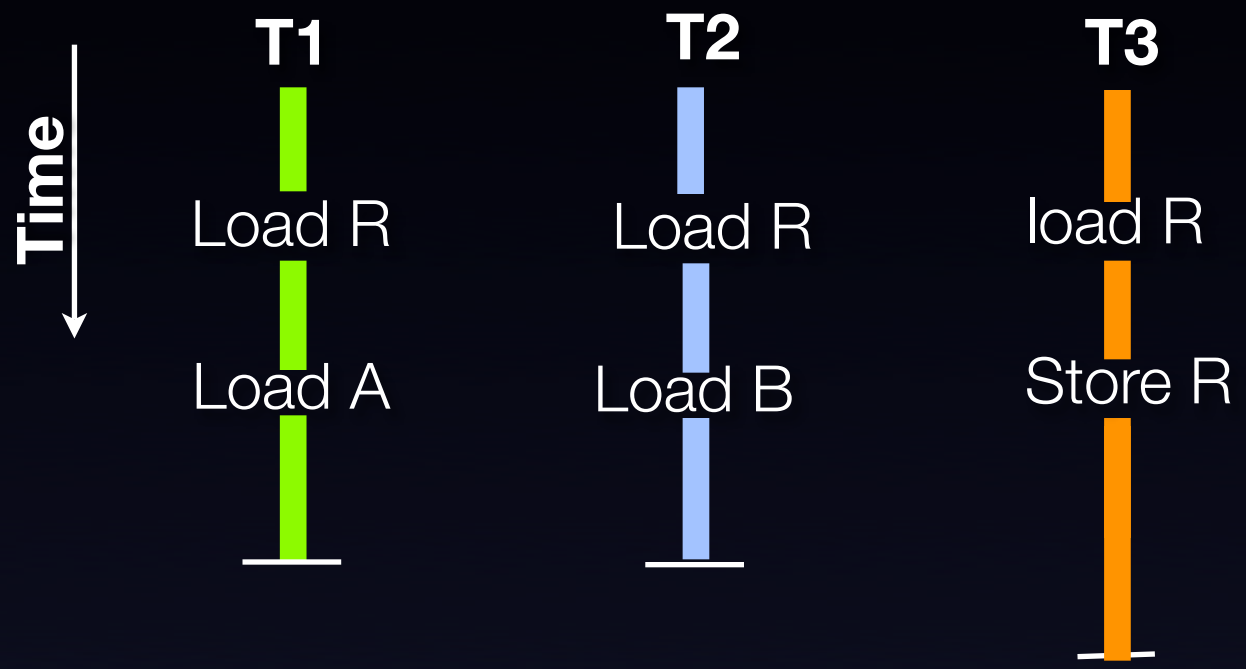
- Conflicts checked only at commit
  - reduces likelihood of conflict winner being aborted
  - can reduce the occurrence of futile aborts
  - prioritizing the commiter avoids livelock in practice



Also observed in Software TMs by Spear et al. [PPOPP'09]

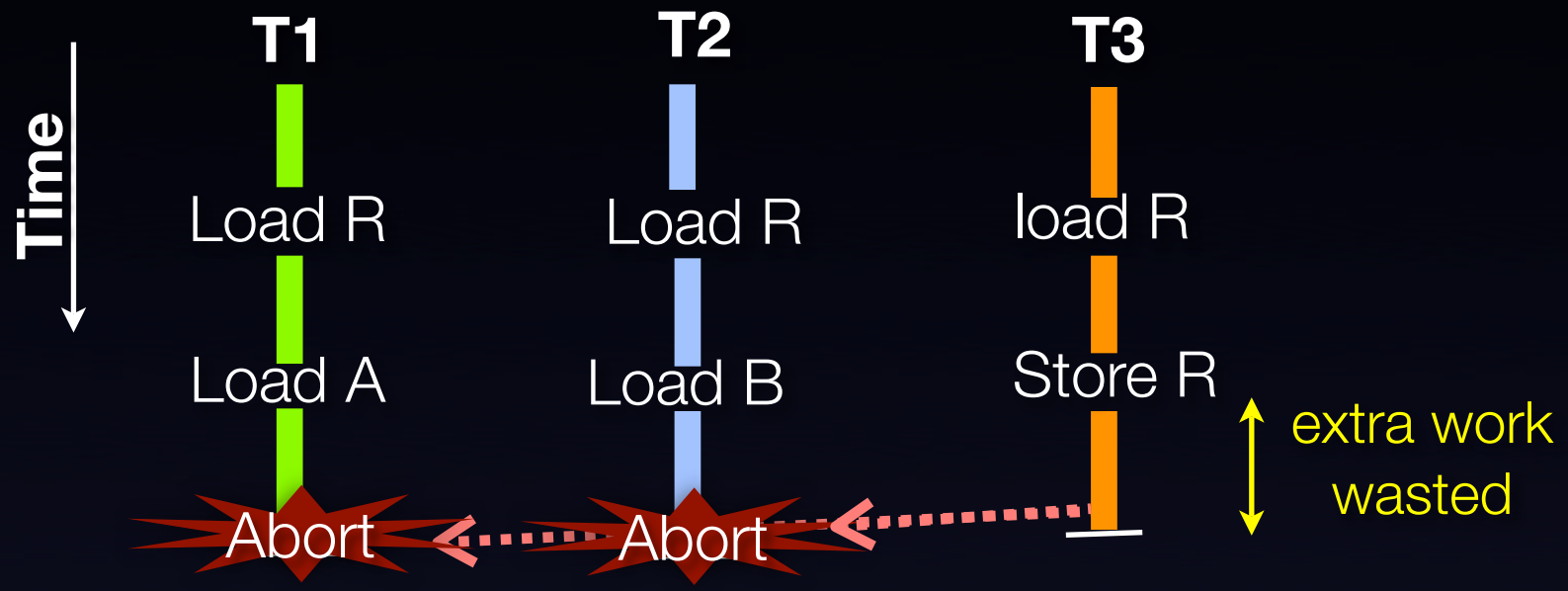
Lazy's Benefits (2/2): More commits

- Even transactions with overlapping accesses
  - can execute concurrently
  - can commit concurrently



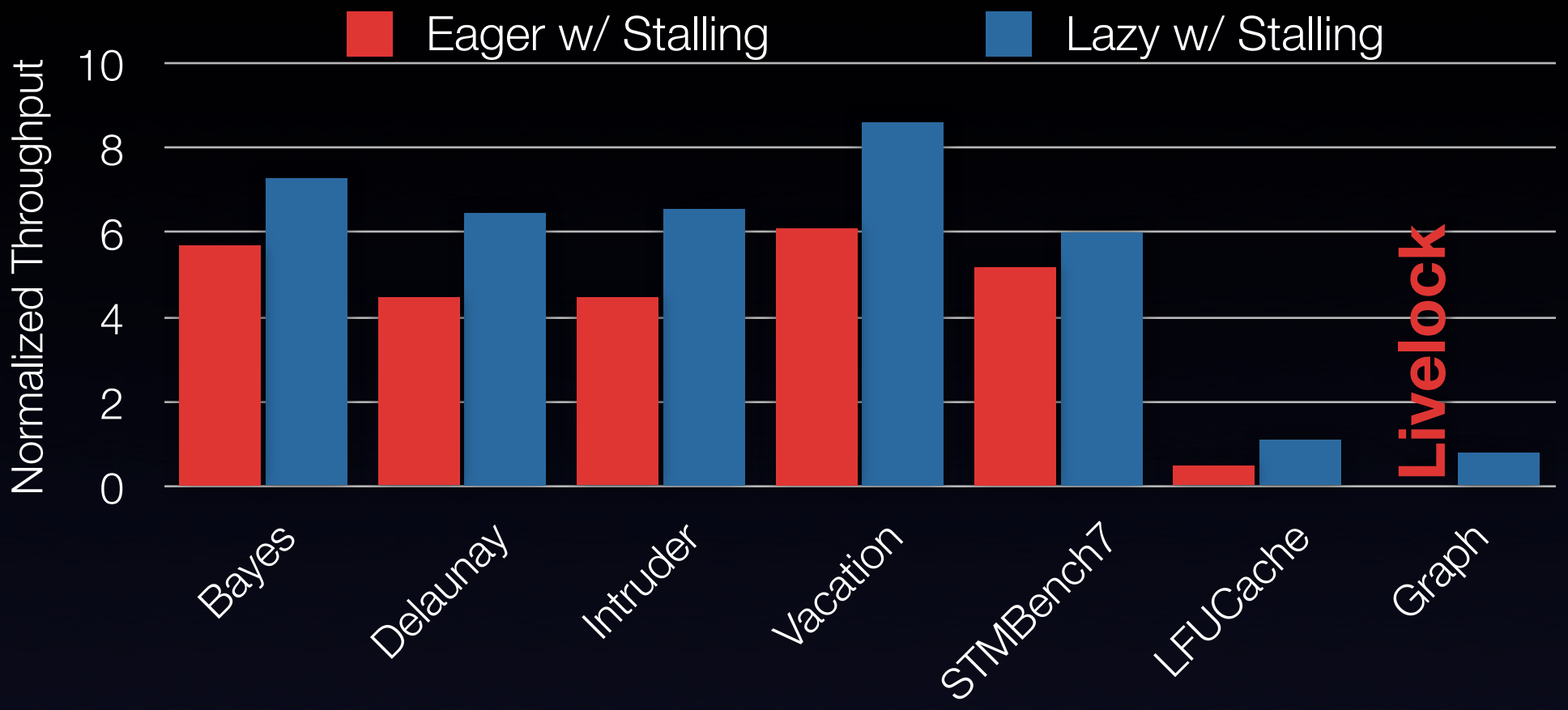
Lazy's Benefits (2/2): More commits

- Even transactions with overlapping accesses
  - can execute concurrently
  - can commit concurrently



- Caveat: Can waste more work than Eager
  - postponing conflict detection was futile (T2 commits first)
  - may be solved by stalling commit

Lazy performs better than Eager



- Lazy improves performance over Eager (Avg. 40% , Max. 2x)
- Ensures progress in non-scalable workloads
- Lazy may lose performance due to wasted work (STMBench7)
  - postponing dueling read-write conflicts is futile

Is Lazy better than Eager ?

Can we do better ? : Mixed

Is the contention manager important ?

Mixed Conflict Detection

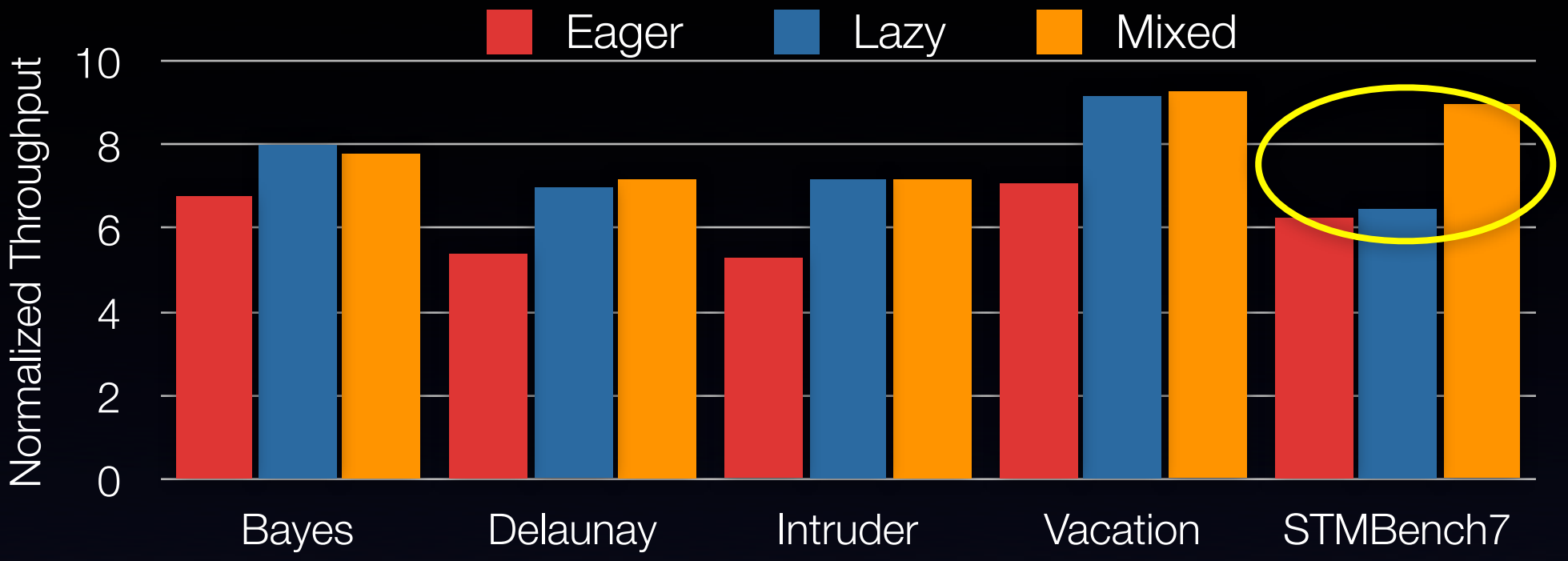
Tunes detection based on conflict type

- Detects Write-Write conflicts eagerly
  - may save wasted work, if winner commits
- Detects Read-Write and Write-Read conflicts lazily
  - allows useful concurrency

Added Benefit: complexity-effective implementation

- needs to support only single-writer and/or multiple-readers
- at most two versions of data, speculative and non-speculative

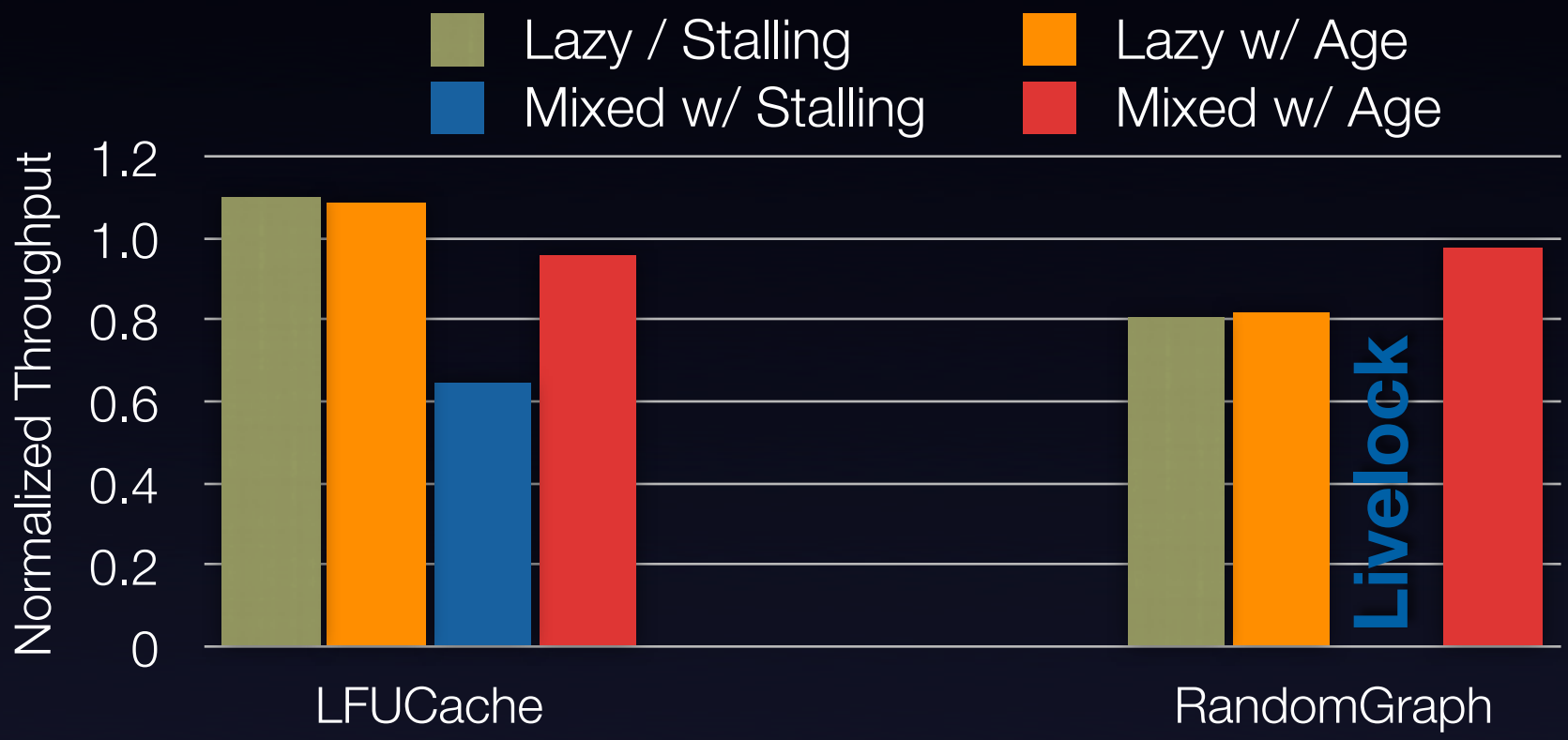
Mixed



- Mixed improves performance by ~40% in STMBench7
  - saves wasted work on conflicts between long and short writers
  - exploits reader-writer concurrency like Lazy

Mixed's Problem

- Mixed can suffer from weaker progress conditions than Lazy
  - inherited from Eager write-write detection
  - can be solved with appropriate contention managers



Is Lazy better than Eager ?

Can we do better ? : Mixed

Is the contention manager important ?

Contention Management

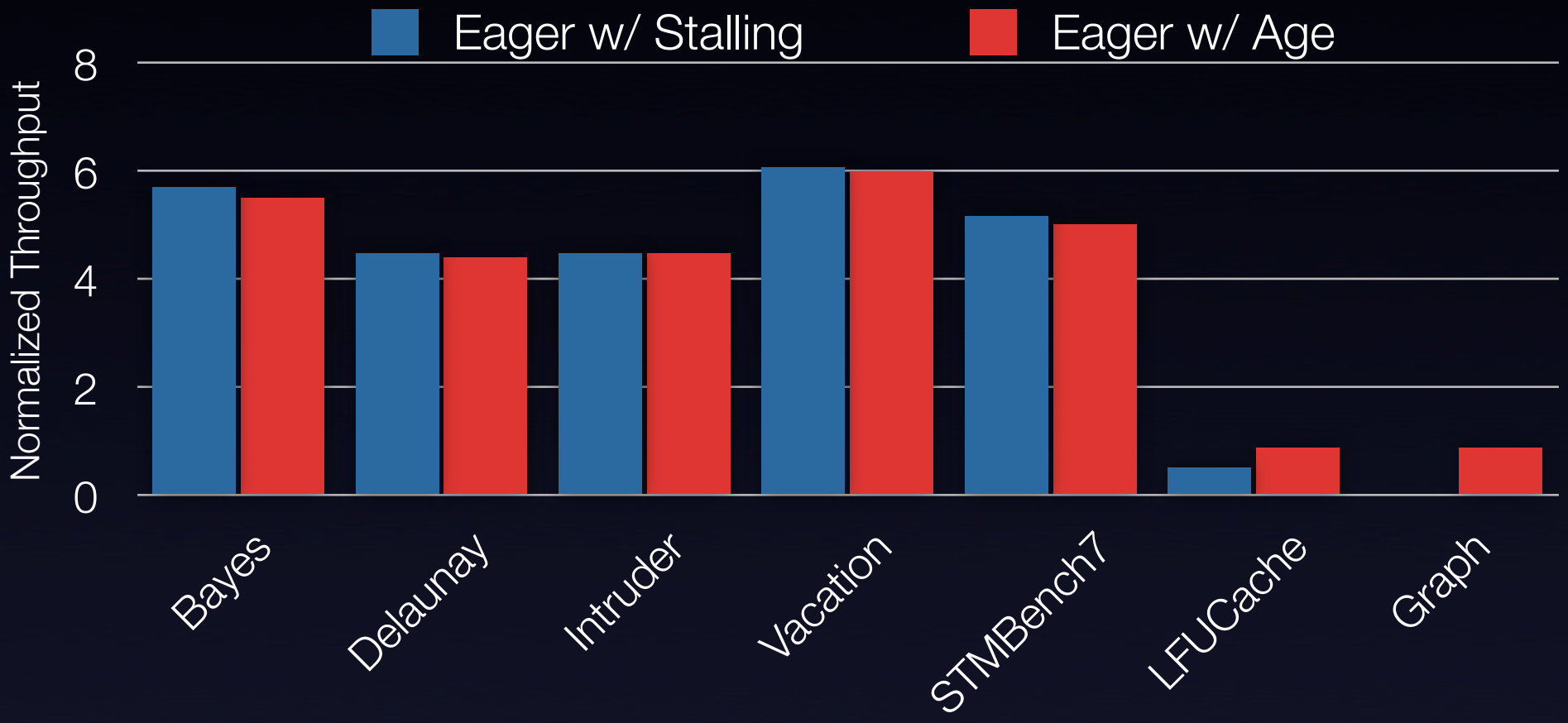
- Is a priority scheme that chooses winner in a conflict
  - can help progress by prioritizing starving transactions
  - simplified in Hardware TMs since transactions are visible
- Priority arbitration (our implementation)
  - always stall before making a decision
  - higher priority transaction always make's progress
  - lower priority transaction can stall or abort itself
  - priority changed on various dynamic events, hardware performance counters to reduce overheads

Priority Schemes

- Age (similar to Greedy [Guerraroui, PODC'05])
  - global timestamp acquired by transaction at begin, retained on aborts, discarded on commits
  - ensures progress of the oldest transaction
- Aborts
  - local abort counter
  - tries to ensure progress of starving transaction
  - theoretically, transaction could always get beaten
- Size (similar to Polka [Scherer, PODC'05])
  - local read set counter, retained on aborts (like Karma)
  - prioritizes transactions which have made progress

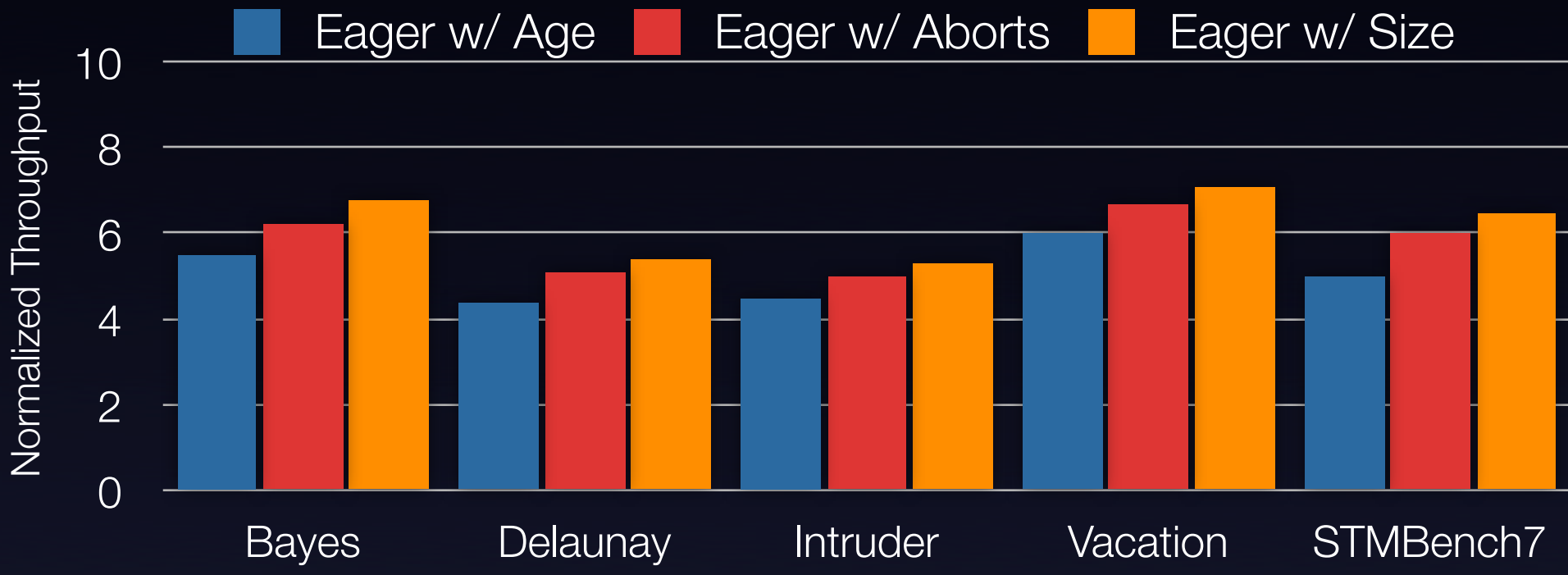
Centralized Priority (Age)

- Implemented in software
- Timestamp suffers from scalability issues
- Hinders concurrency in Eager by convoying readers behind a writer (performance drops ~10%)



Distributed Priority (Size and Aborts)

- Cheaper to implement, no centralized mechanisms
- Weaker progress guarantees
  - no provable starvation or livelock freedom
- Size is highest performing manager
  - maximizes parallelism ensuring reader sharers make progress
  - ensures writers don't starve in practice



Summary

Policy important in HTMs, tradeoffs similar to STMs

- Lazy performs better than Eager (Avg. 40% increase)
  - narrows contention window and ensures progress
  - exploits reader-writer parallelism to attain more throughput
- Mixed is a good tradeoff between desire to exploit concurrency and implementation complexity
- Contention manager
  - less important in Lazy
  - can help with progress in Eager and Mixed

Summary

Look at paper for details on
1) conflict patterns in our TM workloads
2) implementation tradeoff discussion

Acknowledgments
Multifacet Research group, Wisconsin
STAMP group, Stanford
Transaction Benchmark group, EPFL

http://www.cs.rochester.edu/research/synchronization