Coping With Context Switches in Lock-Based STMs

Yoav Cohen

Joint with

Yehuda Afek and Adam Morrison

Tel Aviv University

Agenda

- **Background and motivation**
- The Lock Stealing Algorithm
- TL2 Implementation
- Empirical Evaluation

Software Transactional Memory

- Programmers define blocks of code as *transactions*:

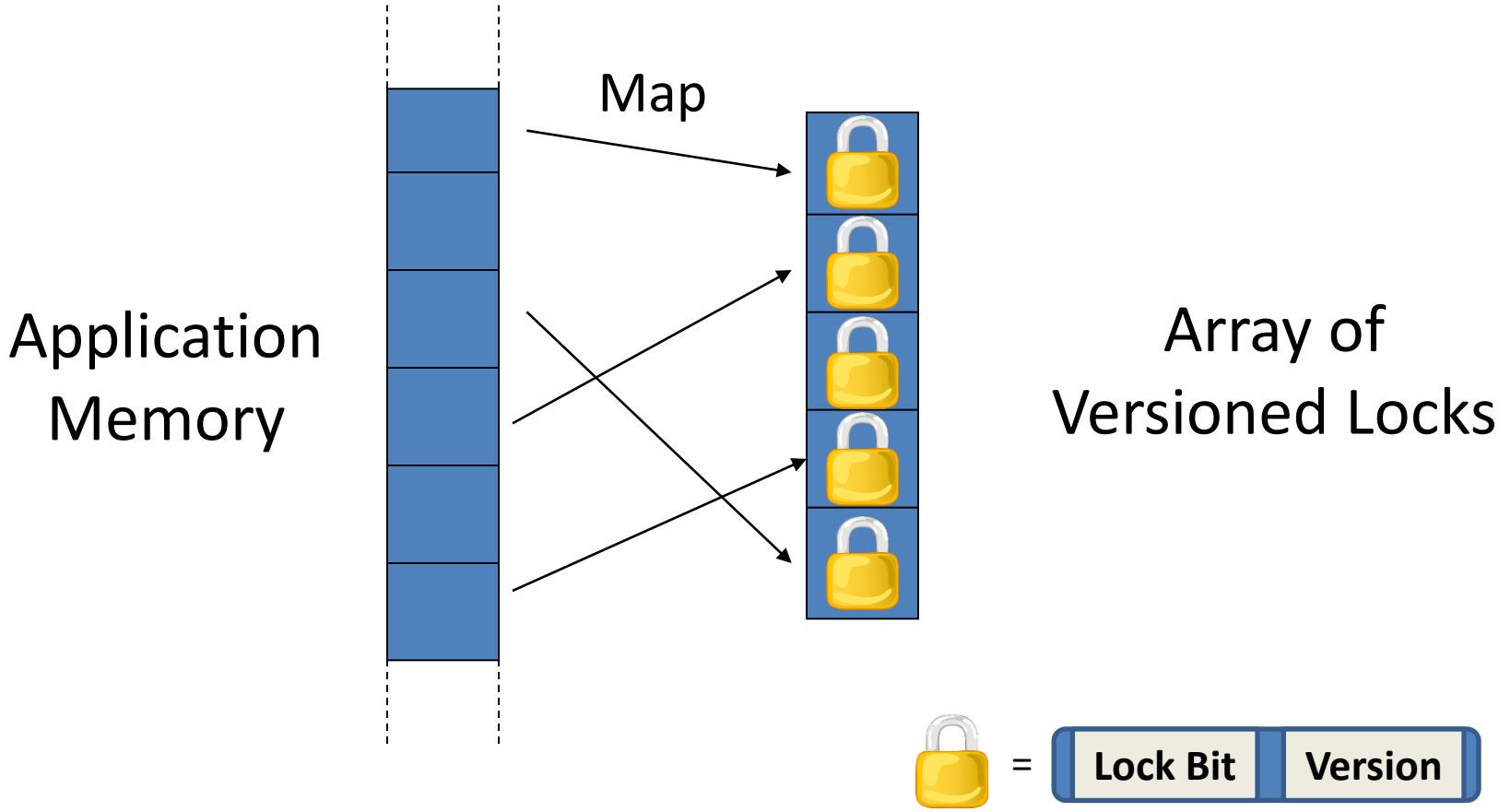
```
atomic {
   <code block>
}
```

- Transactions take effect atomically

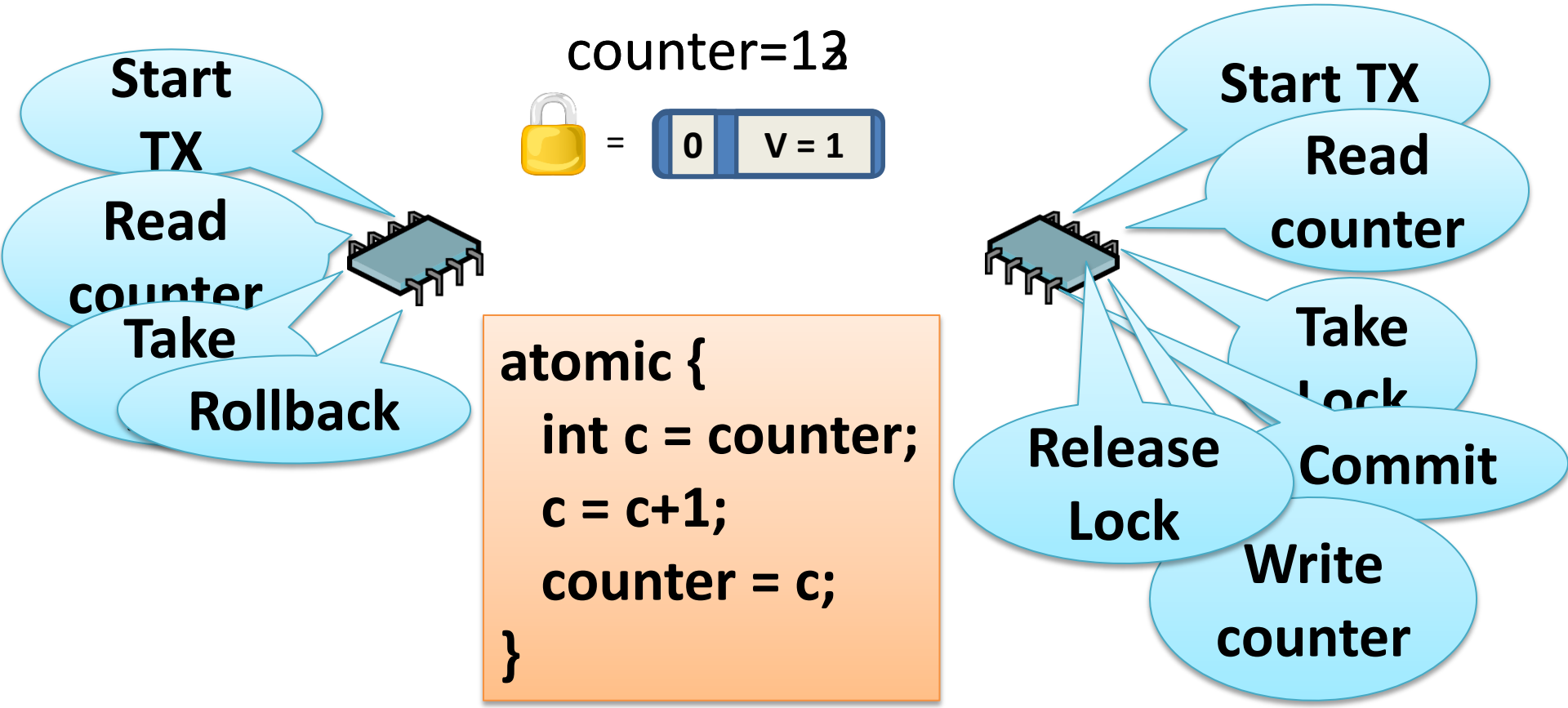
Simplicity of Global Clock with performance of Fine-Grained Locking

Lock-Based STMs

Basic design:



Incrementing a Shared Counter



Context Switches

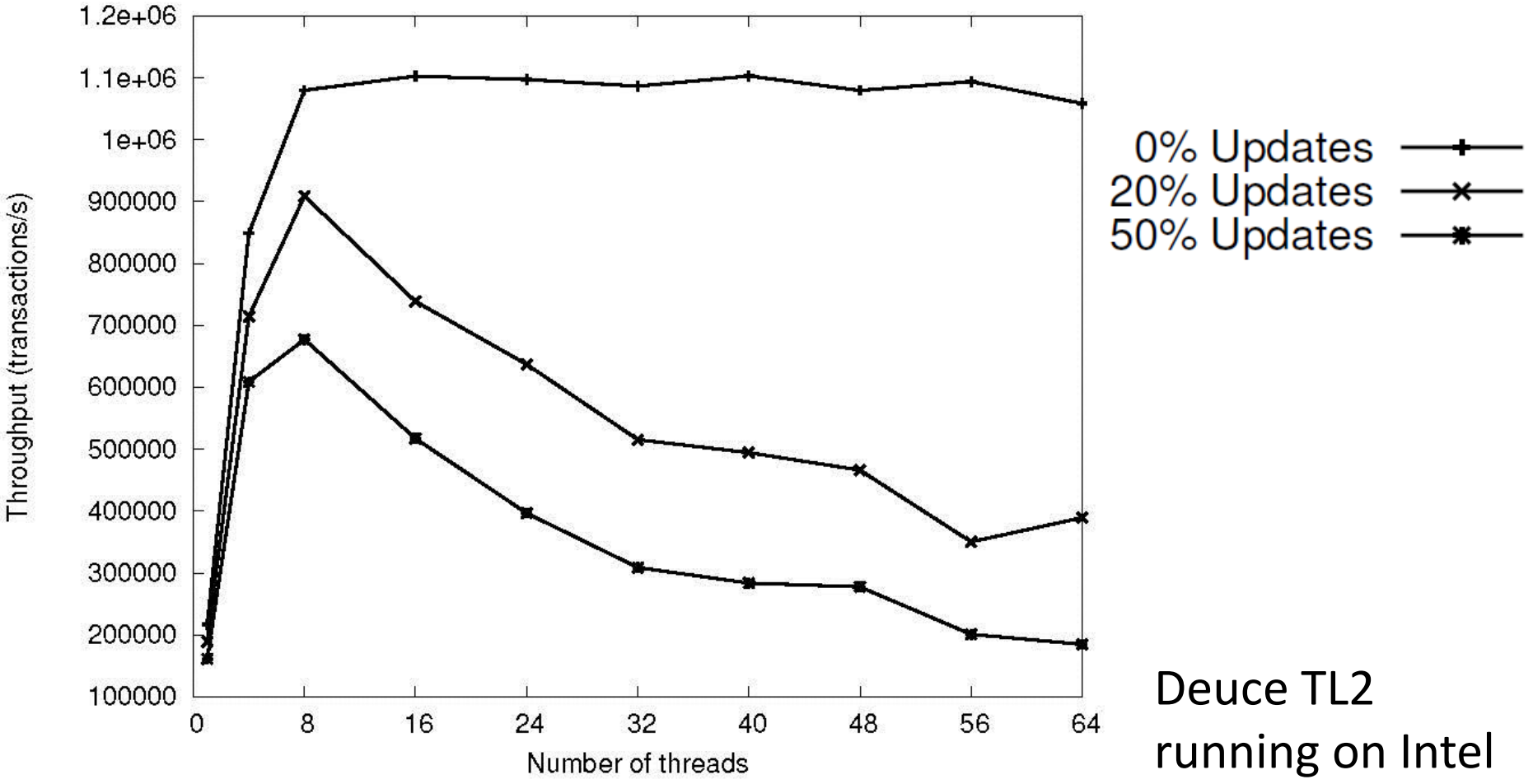
Threads may be switched-out when:

- # S/W threads > #H/W threads
- Interrupts
- Page faults

Q: A thread with a lock is switched out. What happens?

A: Transactions that need this lock abort or wait

The Result: Throughput Degradation



More context switches →

Deuce TL2 running on Intel i7 with 8 hyper threads

Agenda

- Background and motivation
- **The Lock Stealing Algorithm**
- TL2 Implementation
- Empirical Evaluation

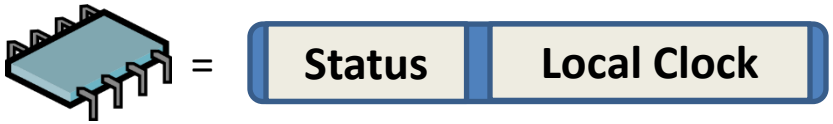
The Solution: Lock Stealing

Instead of waiting for a switched-out lock, **steal it**:

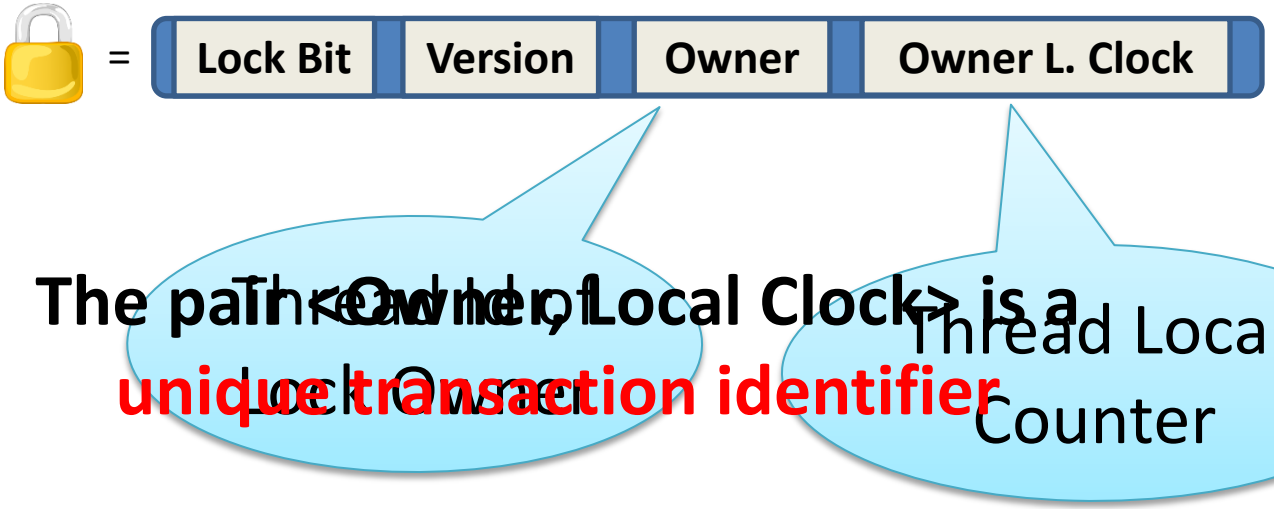
- Abort the switched-out transaction
- Take the lock

Lock Stealing

- Status field per thread:
  - RUNNING, COMMITTED or ABORTED



- Enhanced locks:



Lock Stealing

- <T1,24> aborts <T2,10>:
  - CAS(T2, <**RUNNING**,10>, <**ABORTED**,10>)

- <T1,24> steals L from <T2,10>:
  - CAS(Lock,

    <l=1, v=2, **owner=T2, local_clock=10**>,

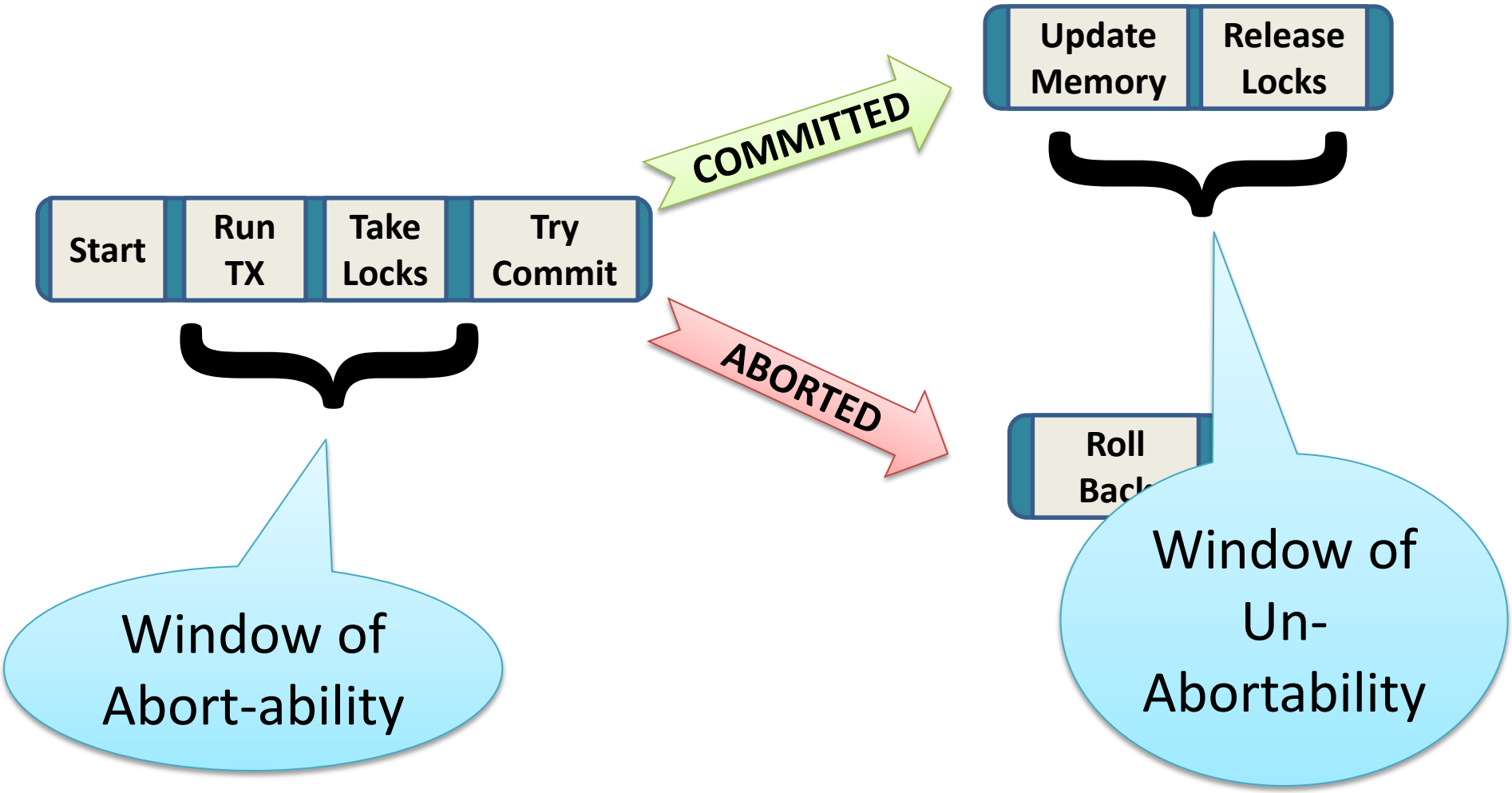
    <l=1, v=2, **owner=T1, local_clock=24**>)

Does It Always Work?

Q: Can we always do this trick?

A: Nope. When a transaction is COMMITTED, it can’t be aborted.

Transaction Lifecycle



Brief Summary

- Context switches cause throughput degradation
- Because switched out locks result in lots of aborts
- New approach: instead of waiting for locks, **abort other and steal the lock**

Agenda

- Background and motivation
- The Lock Stealing Algorithm
- **TL2 Implementation**
- Empirical Evaluation

Lock Stealing for TL2

- Based on Deuce
  - An open-source Java STM framework
- Added Contention Management support:
  - Upon conflict contention manager invoked
  - Decides what to do:
    - Restart current transaction
    - Wait for lock
    - Abort other transaction and steal lock

Lock Stealing for TL2

- Lock-Waiting Contention Managers:
  - Suicide, Aggressive, Karma and Polka
- Lock-Stealing Contention Managers:
  - AggressiveLS, KarmaLS and KillPrioLS

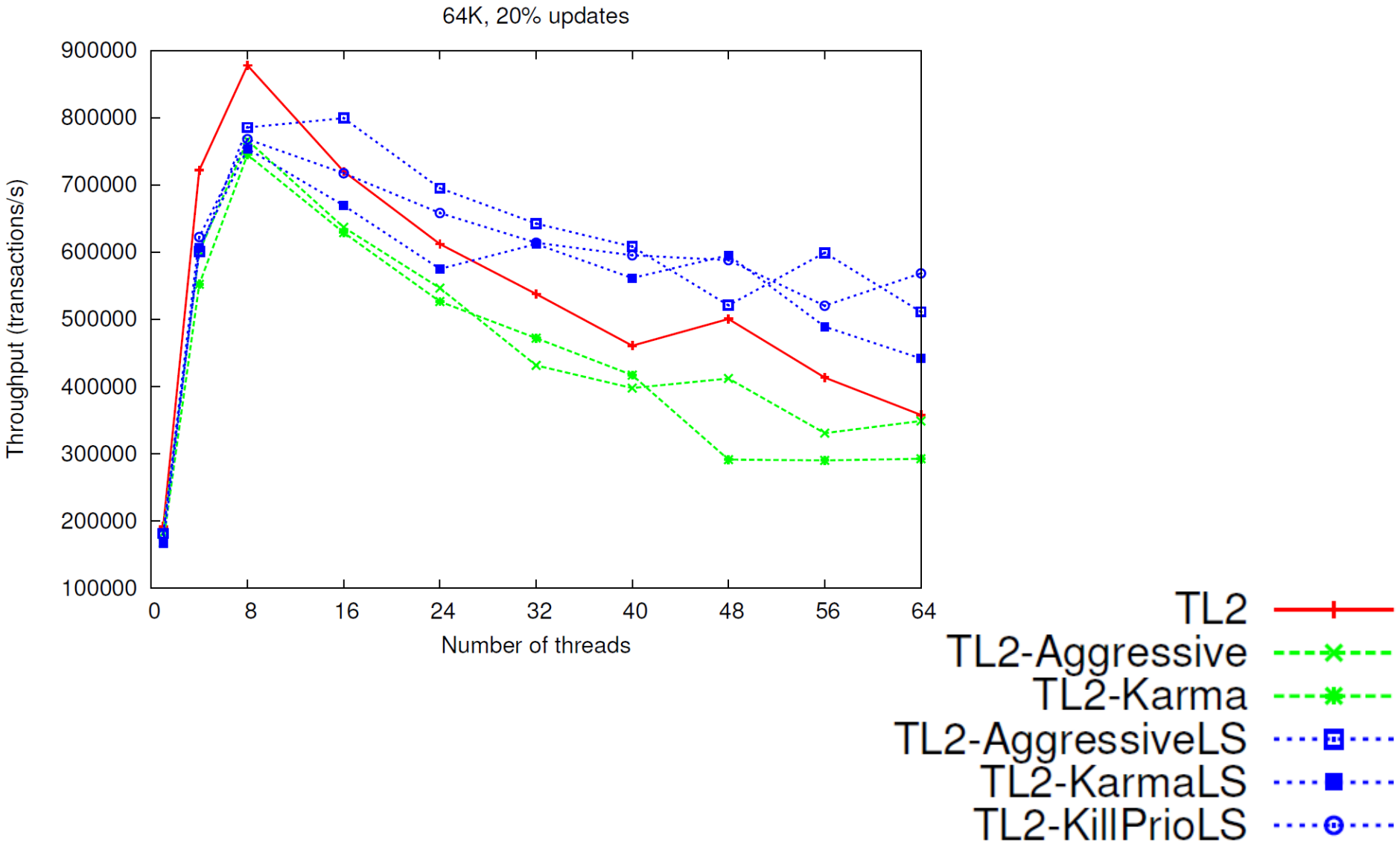
Agenda

- Background and motivation
- The Lock Stealing Algorithm
- TL2 Implementation
- **Empirical Evaluation**

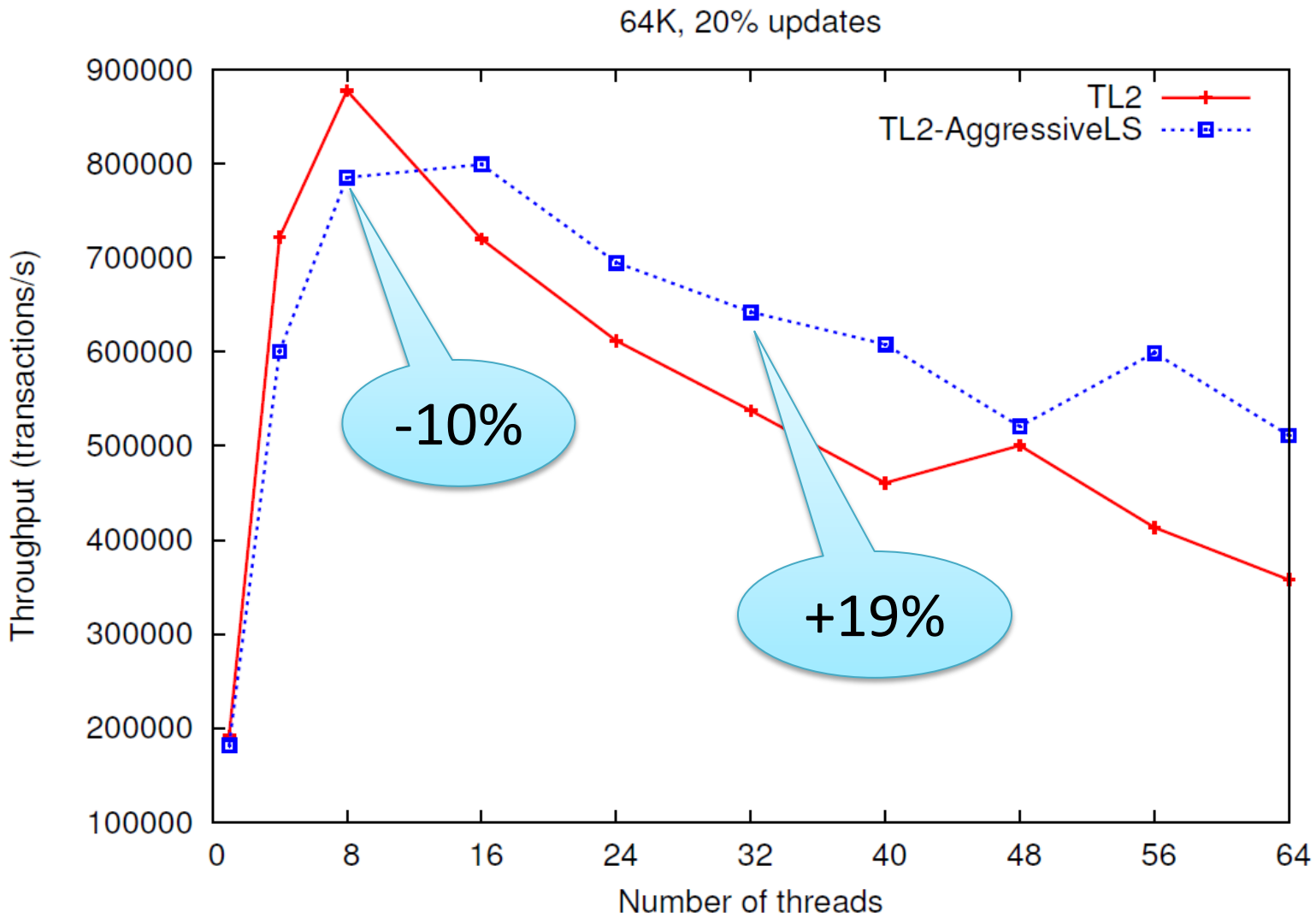
Empirical Evaluation

- Benchmarks:
  - Integer-Set microbenchmarks
  - STAMP – simulates real applications
- Hardware:
  - Intel i7 920 Extreme Edition (Nehalem) 2.67 GHz
  - 4 cores, each running 2 hardware threads

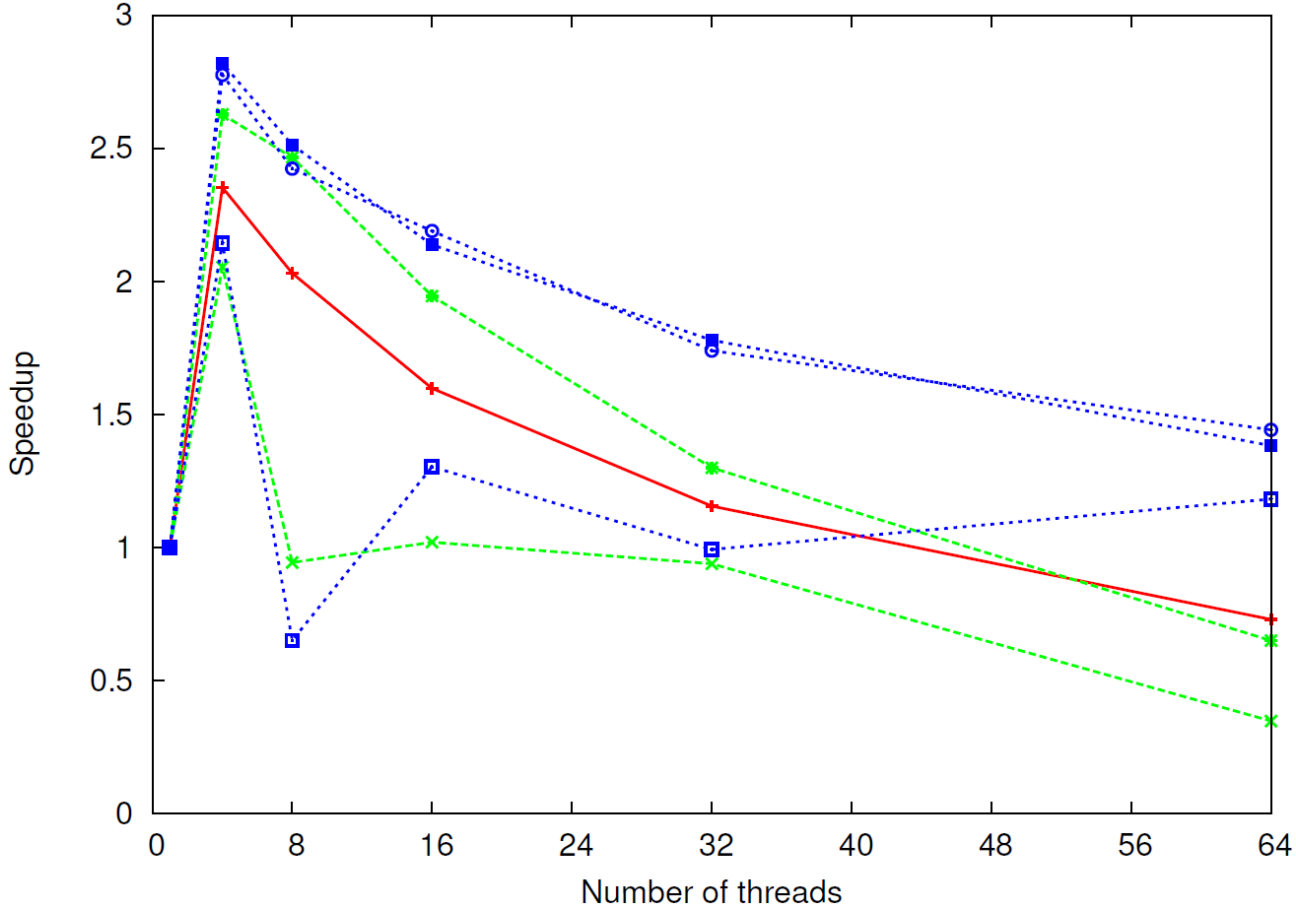
Red-Black Tree Integer Set



Red-Black Tree Integer Set

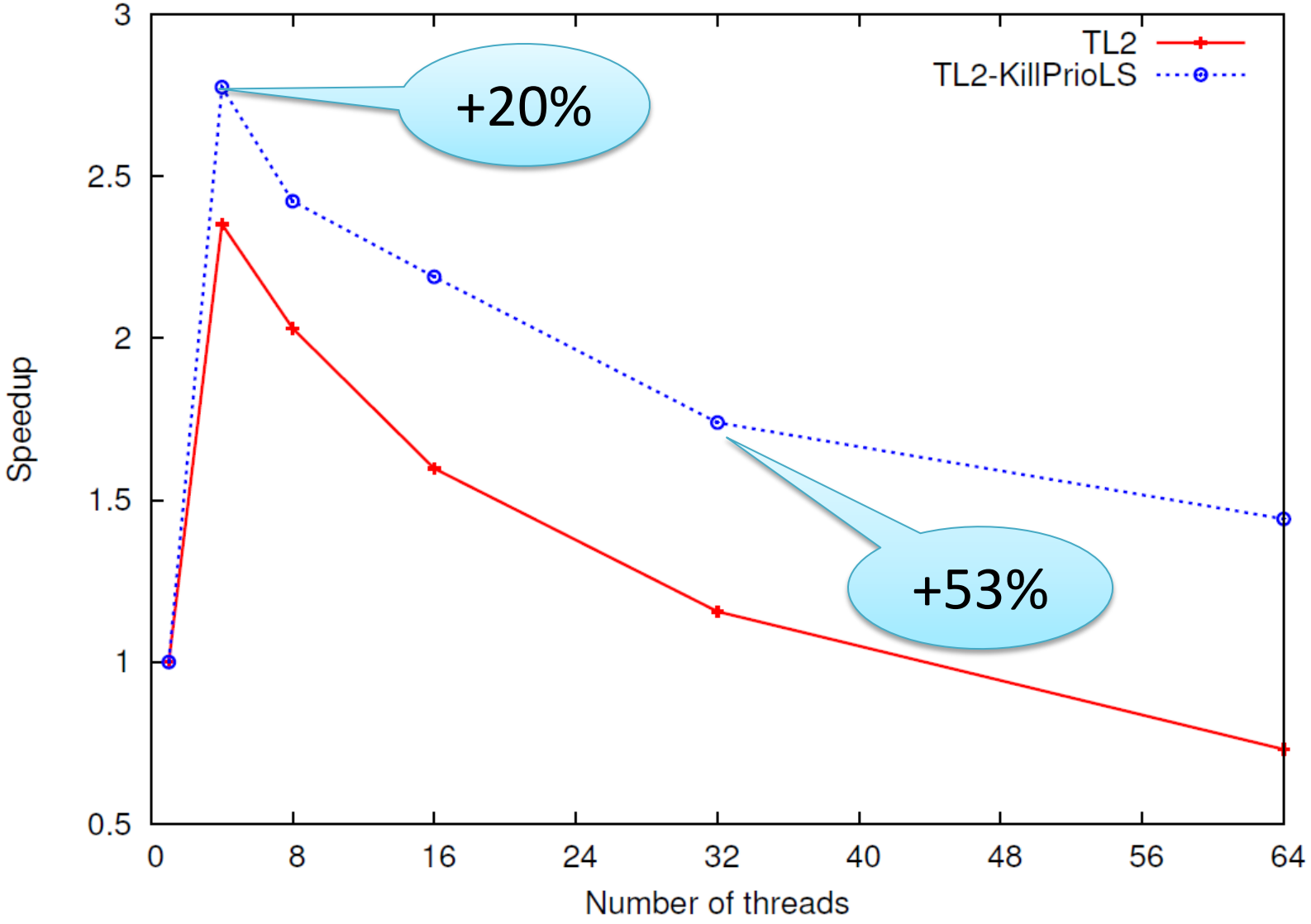


STAMP Intruder



TL2
TL2-Aggressive
TL2-Karma
TL2-AggressiveLS
TL2-KarmaLS
TL2-KillPrioLS

STAMP Intruder



Thank You

Links

- Deuce STM project
  - http://sites.google.com/site/deucestm/
  - org.deuce.transaction.tl2cm package