



# TRANSACTIONAL MEMORY AT SUN

**Mark Moir**

Scalable Synchronization Research Group  
Sun Labs



**2007  
Sun Labs  
Open House**



# TM research is hot!

- Distributed computing:
  - > PODC '94, '03-'07, DISC '05-'06
- Systems:
  - > ISCA '93, '04-'07, ASPLOS '05-'06, HPCA '05-'06, PPOPP '06
- Programming languages:
  - > PLDI '05-'06, OOPSLA '06
- Compilers:
  - > CGO '07
- New workshops dedicated to TM:
  - > CSJP '04, SCOOL '05, Transact '06-'07, ...

# Why Now? The Multi-Core Revolution

- Industry shift: more cores, not faster processors
- More throughput, less power/heat
- Great for consolidation, but what about individual applications?
- Increasingly, everyday programmers must become concurrent programmers

# Concurrent Programming is Too Hard!

- Today's concurrent programming: locks and conditions
- Tradeoffs between performance, scalability, software engineering
- Only “experts” can get it right (and they don't always...)
- Transactional Memory can help!

# Example: queue transfer

```
xferValue(Q1,Q2) {  
    Q1.lock.acquire();  
    Q2.lock.acquire();  
    v = Q1.dequeue();  
    Q2.enqueue(v);  
    Q2.lock.release();  
    Q1.lock.release();  
}
```

- Locks exposed for composition
- Programmer must know and obey locking convention
- This solution broken: deadlock

# Transactional programming

- Express **what** to do atomically, not **how** to do it.

```
xferValue(Q1,Q2) {  
  Q1.lock.acquire();  
  Q2.lock.acquire();  
  v = Q1.dequeue();  
  Q2.enqueue(v);  
  Q2.lock.release();  
  Q1.lock.release();  
}
```



```
xferValue(Q1,Q2) {  
  atomic {  
    v = Q1.dequeue();  
    Q2.enqueue(v);  
  }  
}
```

# Scalable Synchronization Research Group

- Dave Dice, Victor Luchangco, Mark Moir, Dan Nussbaum, Nir Shavit
- Visiting professor: Maurice Herlihy (Brown)
- Eternal intern: Yossi Lev (Brown)

# Interns, Academic Collaborations

- Previous interns:
  - > Simon Doherty (Victoria, NZ)
  - > Sasha Fedorova (Harvard, Simon Fraser)
  - > Virendra Marathe (Rochester)
  - > Bill Scherer (Rochester, Rice)
  - > Ori Shalev (Tel Aviv)
- Other academic collaborators including University of Auckland (NZ), University of Wisconsin

# Some papers by SSRG members

- Transactional Memory: Architectural Support...
  - > Herlihy and Moss *ISCA 1993*
- Software Transactional Memory
  - > Shavit and Touitou *PODC 1994*
- Hybrid Transactional Memory
  - > Damron, Fedorova, Lev, Luchangco, Moir, Nussbaum, *ASPLOS 2006*
- Transactional Locking II (TL2)
  - > Dice, Shalev, Shavit *DISC 2006*
- A Flexible Framework for Implementing STM (DSTM2)
  - > Herlihy, Luchangco, Moir *OOPSLA 2006*

# Implementing Transactional Memory

- Bounded Hardware TM
  - > Herlihy & Moss, fixed size transactional cache
- Best-effort Hardware TM
  - > Leverage existing caches, store buffers, ...
- Unbounded Hardware TM
  - > Complex, risky
- Software TM
  - > Flexible, but heavy overhead

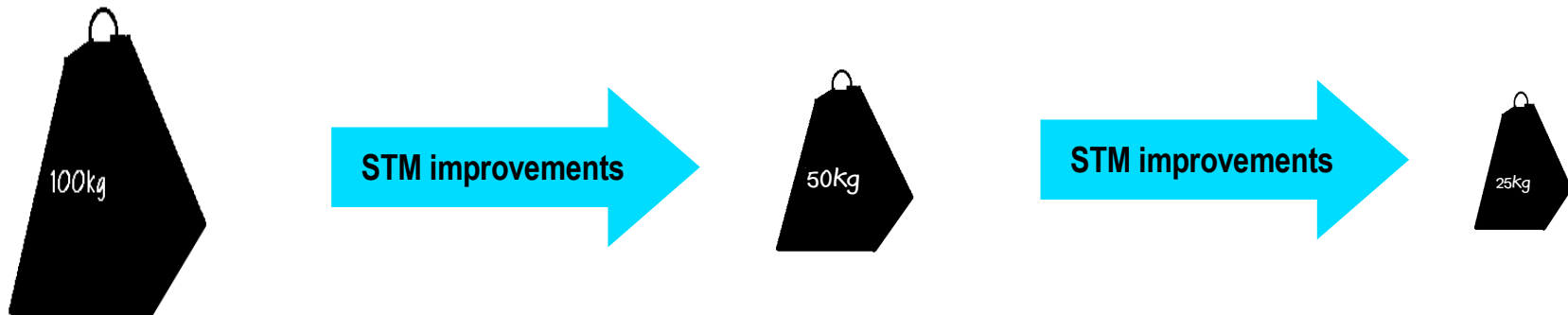


# Hybrid Transactional Memory

- Fully functional Software TM integrated into compiler
- Can develop and test transactional programs today, experiment with programming model, etc.
- Use best effort Hardware TM to boost performance
- Simplifies hardware design, gives more flexibility: HTM can “give up when the going gets tough”, omit features that are too complicated or too uncommon

# Extending the Free Lunch

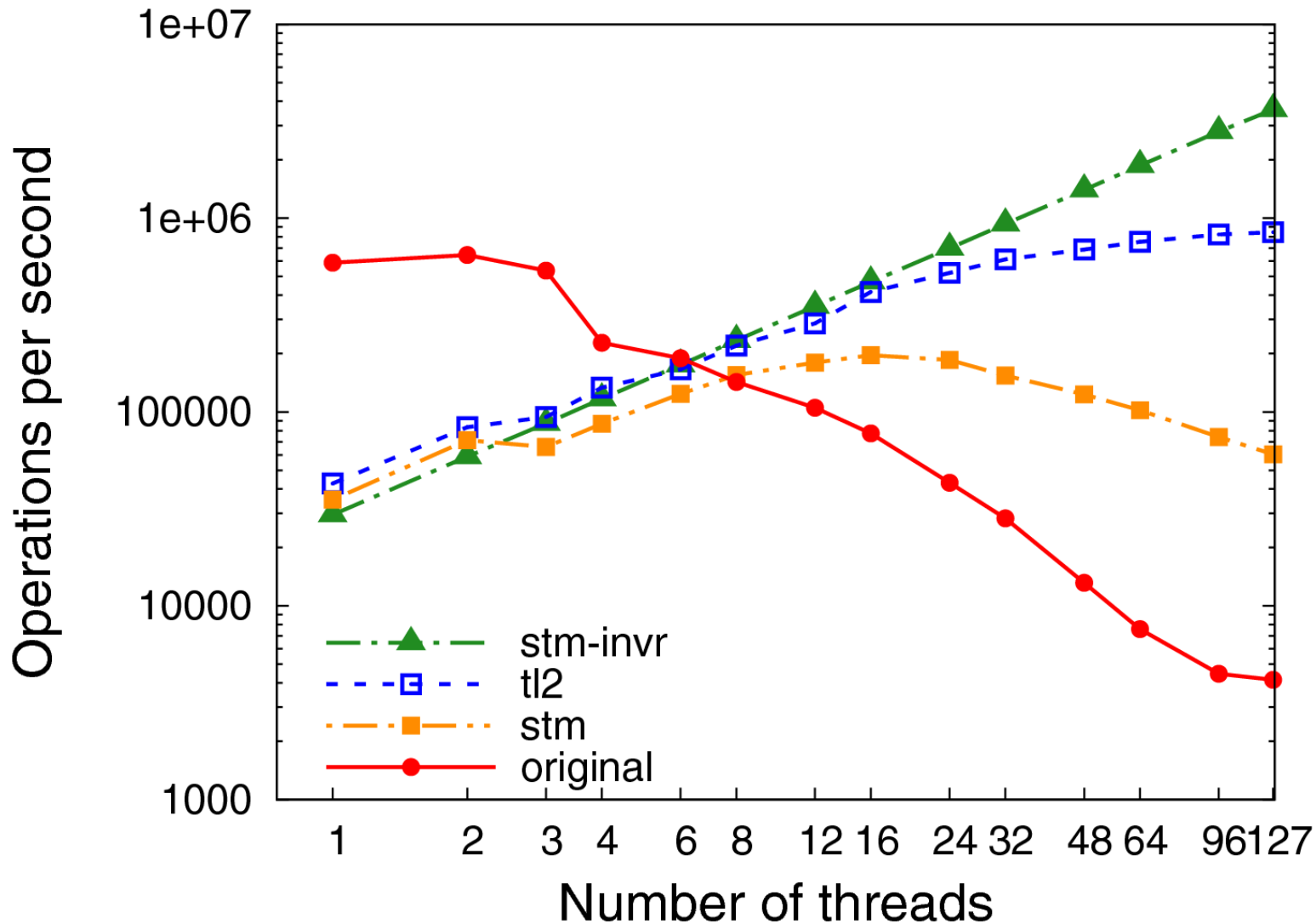
- Develop transactional code now, use future best-effort hardware support to boost performance, future “better effort” can continue to improve performance



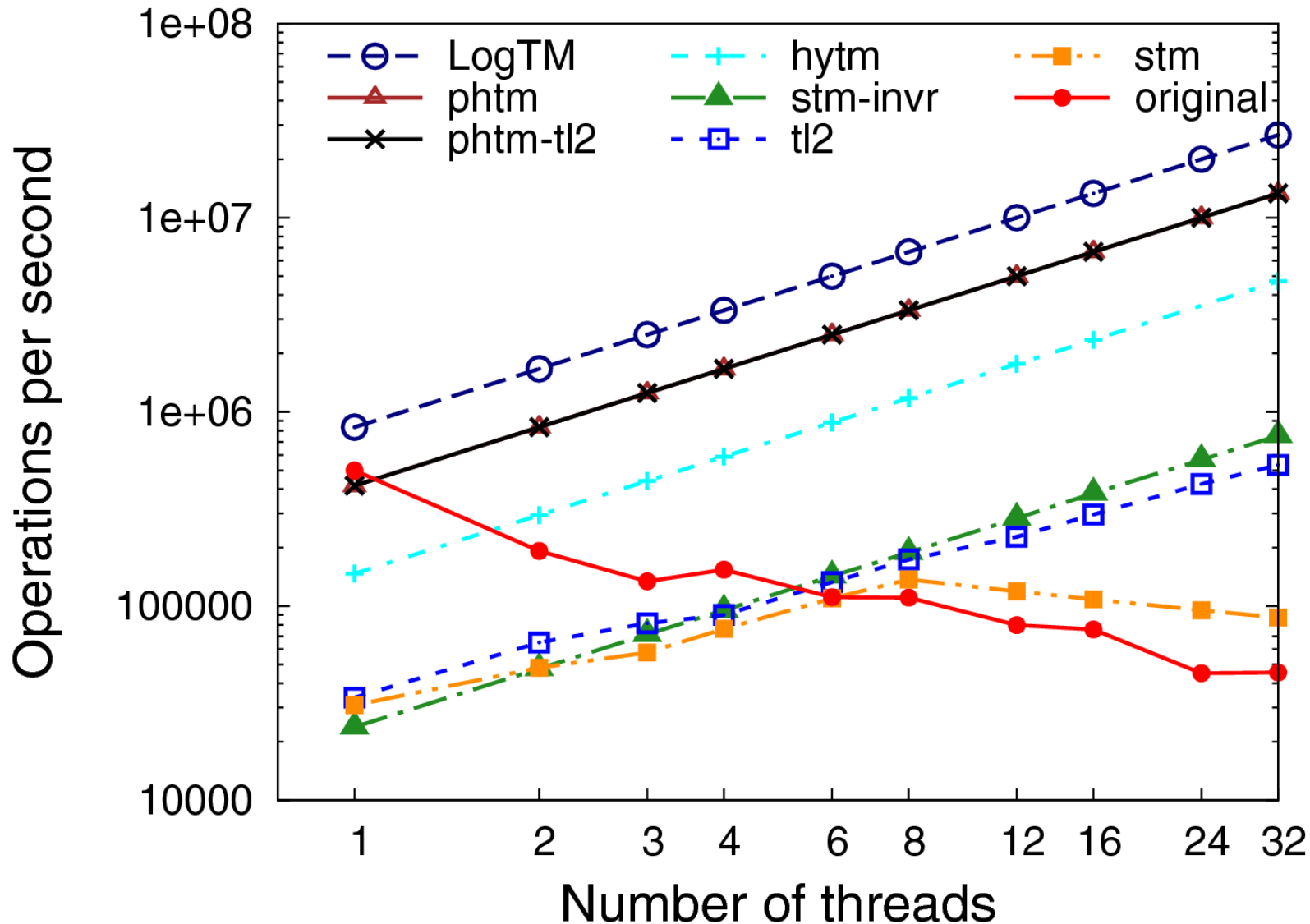
# PhTM: Phased TM

- Hybrid TM (ASPLOS 2006) imposes significant overhead on hardware transactions, constrains STM design
- PhTM: different “modes” support different transaction implementations, seamlessly switch between them
- Adapt to best mode for workload, execution environment

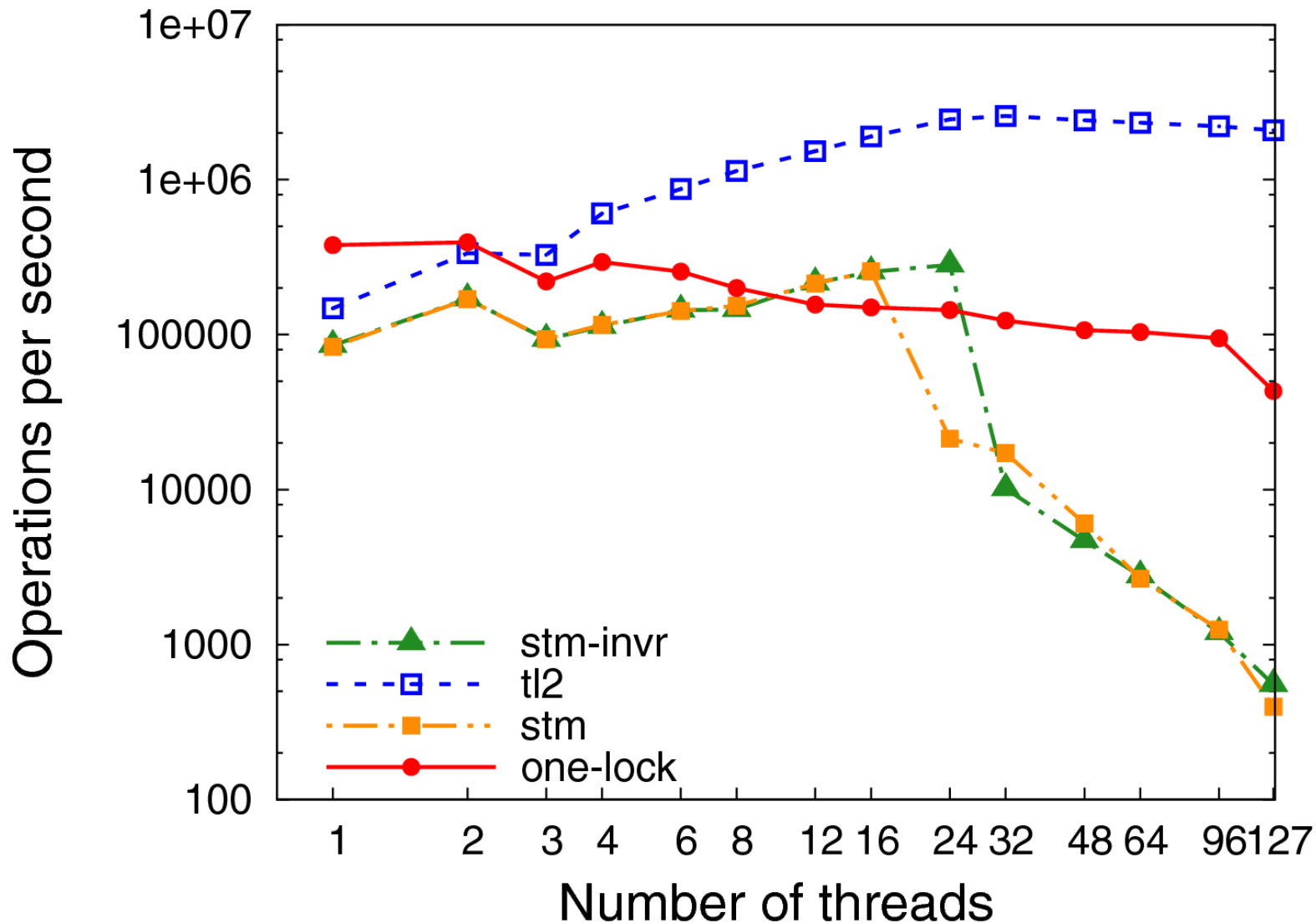
# Berkeley DB, Sun Fire™ E25K



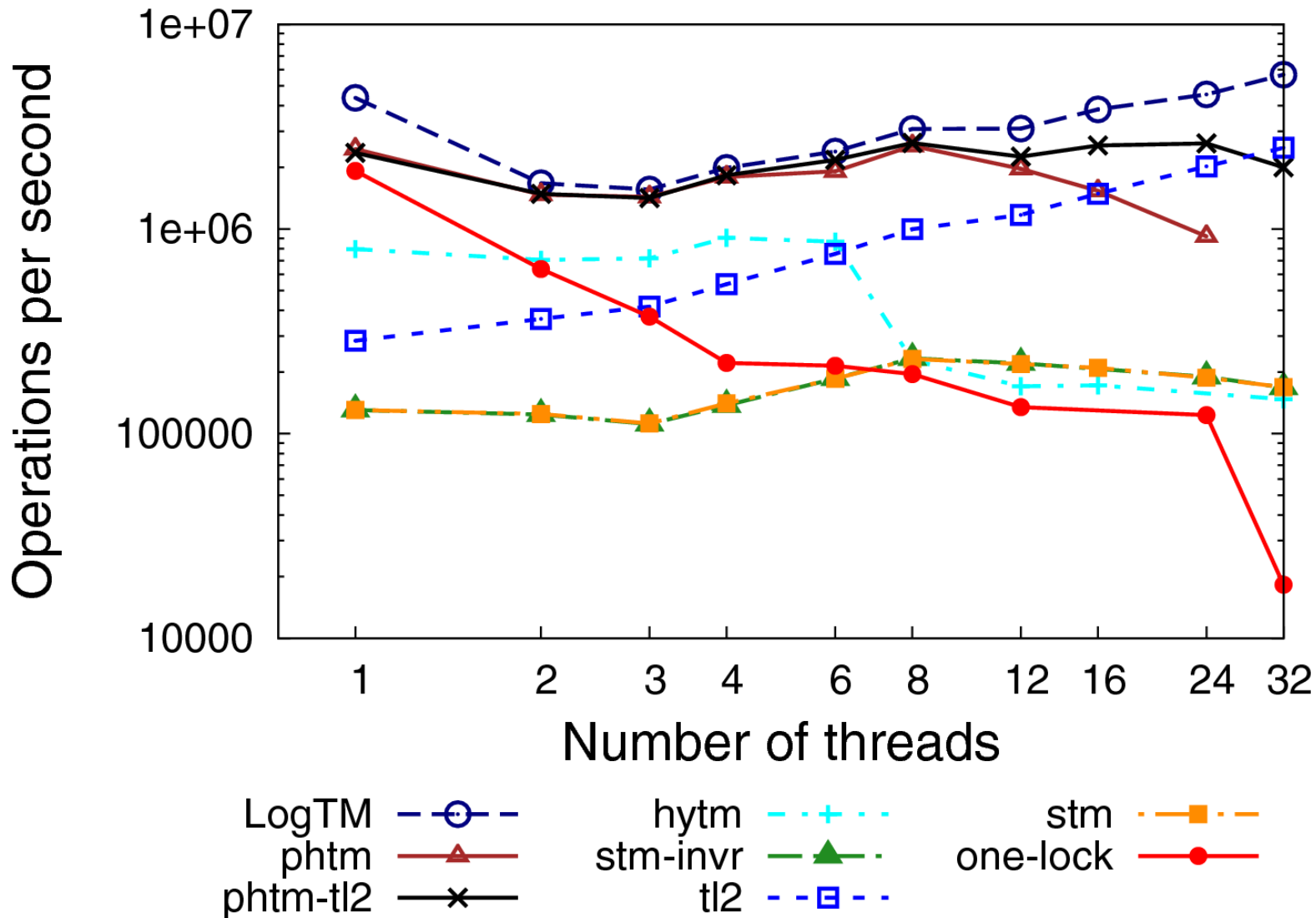
# Berkeley DB, with simulated HTM



# Red-Black Tree, Sun Fire™ E25K



# Red-black Tree, with simulated HTM



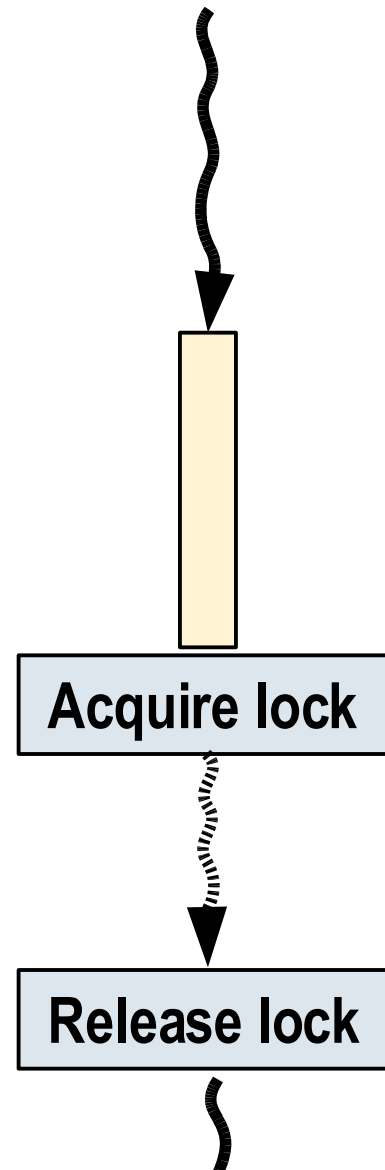
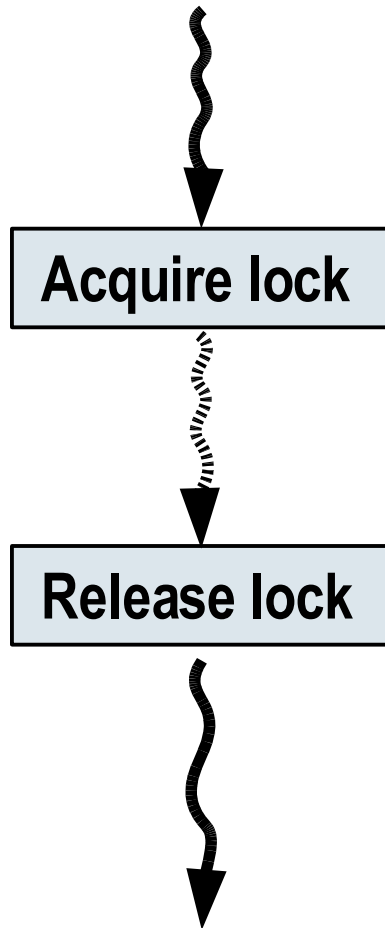
# Language and Tool Integration

- Built into production-quality C/C++ compiler, but supports only rudimentary programming interface
- Working on incremental approach to integrating a more complete programming model
- Address key issues, such as:
  - > Weak vs. strong atomicity
  - > Exception handling
- Debugger integration (Lev & Moir, Transact '06)

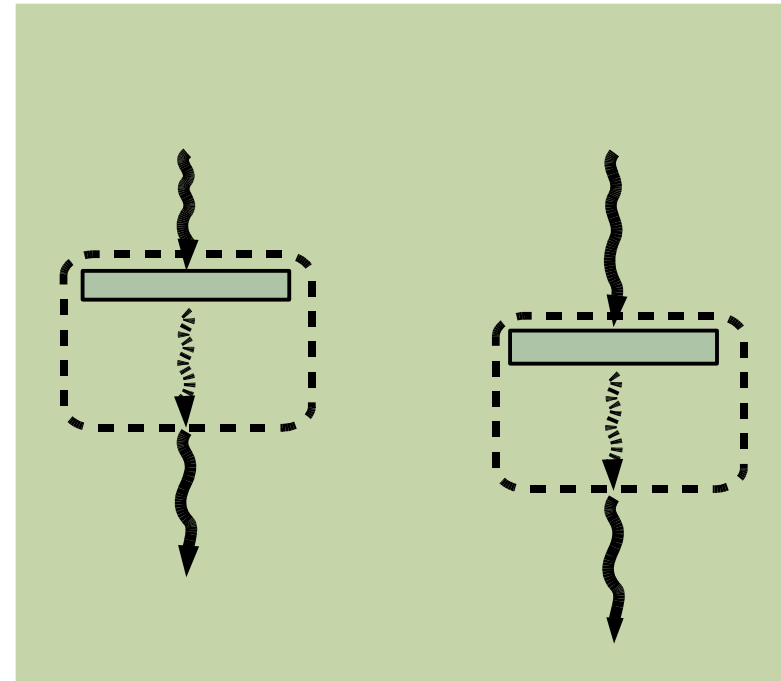
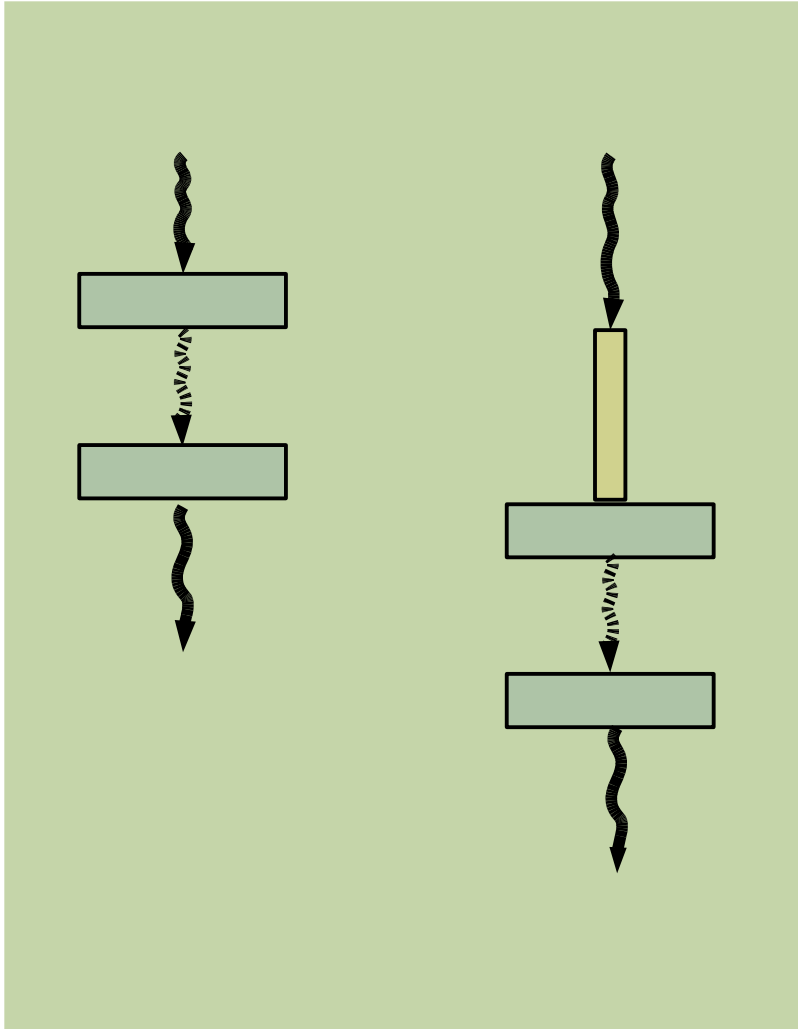
# Boosting Performance of *Existing* Applications

- Good progress demonstrating viability of Hybrid TM approach, closing gap to unbounded HTM
- But transactional programming models will take time to mature, be adopted
- More compelling case can be made by boosting existing applications and systems

# Lock bottlenecks



# Eliminating lock bottlenecks with HTM



- Use HTM to execute critical section *without acquiring lock*

# Eliminating lock bottlenecks with HTM

- If no conflicts and don't exceed HTM limitations, critical sections execute in parallel
- Otherwise, retry, eventually get lock
- Don't want to use it in the wrong places
- Solaris™: apply selectively
- Java™: can apply optimistically with JIT, back out when not profitable



# Concluding Remarks

- Hybrid TM supports transactional programs today, can use best-effort HTM to boost performance later
- We're improving STM and Hybrid TM, approaching performance of unbounded HTM
- Best-effort HTM supports other purposes too, including performance improvements for existing applications and systems
- We therefore argue for building best-effort HTM if unbounded is not feasible



# QUESTIONS?

**Mark Moir**

mark.moir@sun.com

<http://research.sun.com/scalable>



**2007  
Sun Labs  
Open House**

