Conflict Detection and Validation Strategies for Software Transactional Memory

Michael F. Spear, Virendra J. Marathe,
William N. Scherer III, and
Michael L. Scott

University of Rochester
www.cs.rochester.edu/research/synchronization/

DISC 2006

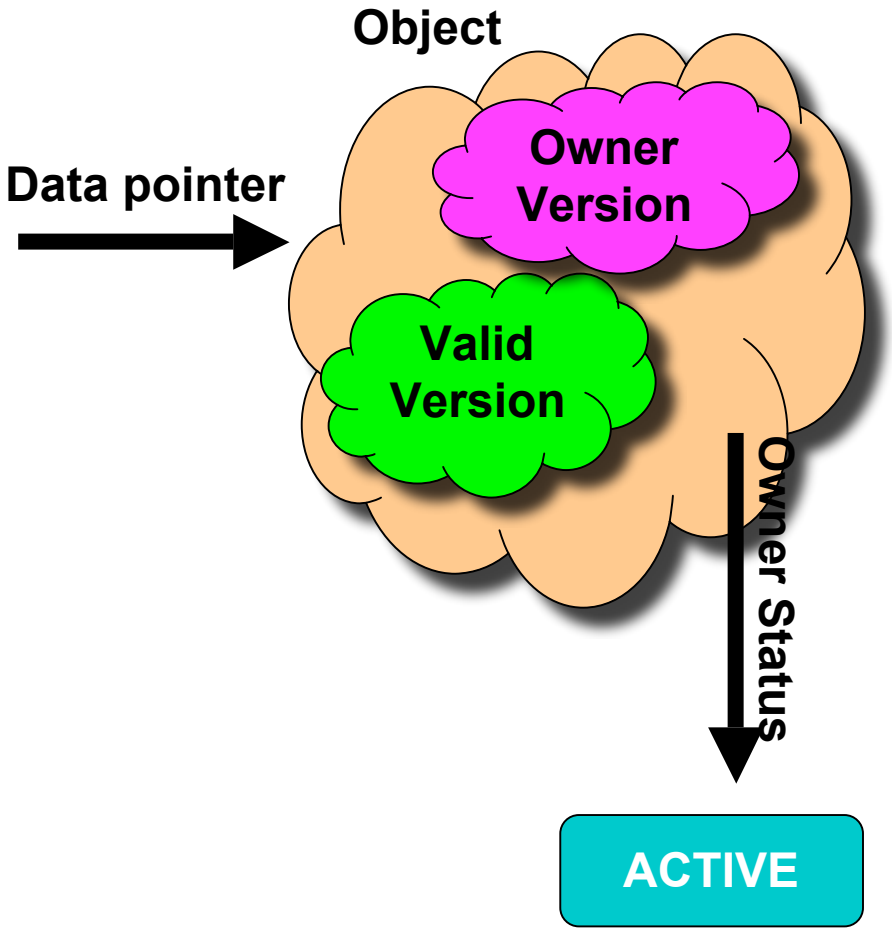
Software Transactional Memory

- Atomicity rather than mutual exclusion; avoids deadlock, priority inversion, and "wrong lock" bugs
  - » aims for the clarity of coarse grain locks with the performance of fine(r)-grain locks
- *Nonblocking* STM additionally tolerates preemption, page faults, and thread failure
  - » these may be more important on "general purpose" systems than they have been at the high end
    - more multiprogramming
    - less predictable job mix
- ✶ Focus on NB TM—how fast can we make it go?
  - » some results applicable to blocking TM as well

RSTM Overview

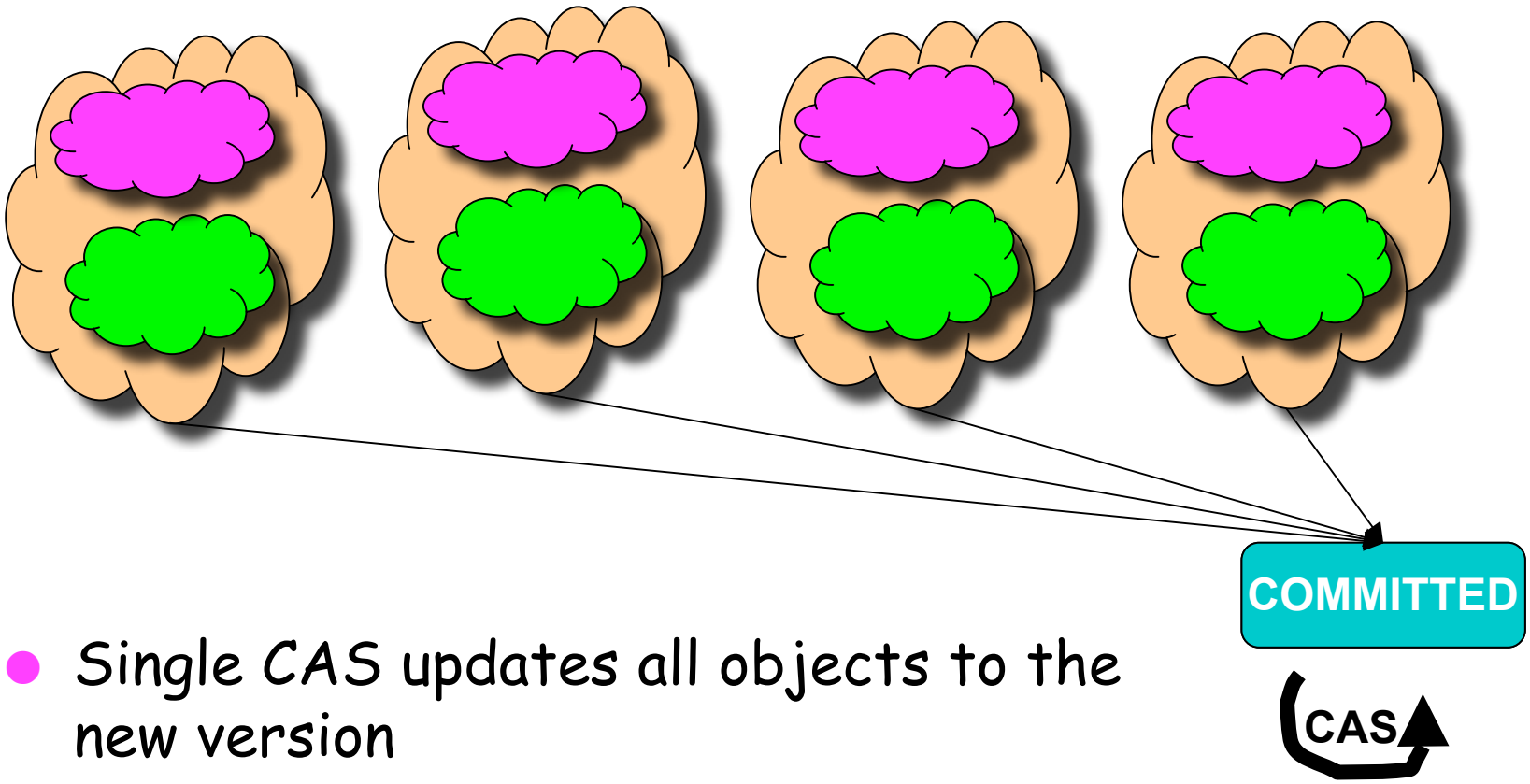
- Nonblocking (obstruction free) TM library for C++
- Per-object metadata, explicit atomic operations
- Roll back only `Shared` (transactional) objects on failure
- `Open` operation (as opposed to per-access bookkeeping)
- Described at TRANSACT '06
- Optimized for low heap churn and limited indirection

Object-based NB STM



- Only owner can change the object
- Only one transaction can own the object at a time
- Until the owner commits, everyone sees Valid Version

Atomic Commit



- Single CAS updates all objects to the new version

Conflict Detection

- Identifies threats to serializability
  - » If A and B use X concurrently and at least one writes, they cannot both commit
  - » Choosing between them is the subject of other papers
- Writers become visible by *acquiring* an object
  - » can do so eagerly (at *open* time) or lazily (at commit)
  - » eager avoids useless work; lazy avoids spurious aborts
- Readers may or may not be visible
  - » if visible, they suffer misses for metadata update
  - » otherwise, only readers can detect RW conflicts
- *Mixed* invalidation delays action on RW conflicts
  - » both threads may commit if reader does so first

Validation

- Assures isolation of doomed transactions
- Newly opened objects must be consistent with previous reads

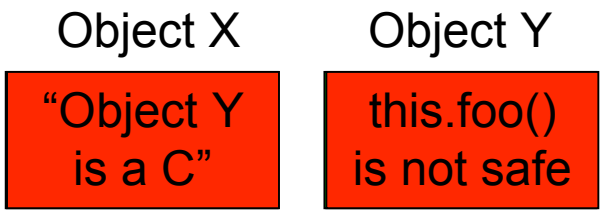
- Tolerate errors in doomed transactions?
- Rely on a typesafe language?

- No! These do not protect us.

Validation matters!

- Suppose class A has subclasses B and C
- B.foo() is tx-safe; C.foo() is not

Transaction 1	Transaction 2
Read X	
	Acquire Y
	Acquire X
	Make Y a B
	Update X
	COMMIT
Read Y — Validation could detect inconsistency here	
Call Y.foo() — Error!	



★ ***Periodic checks are not enough***

» must validate prior to every dangerous op

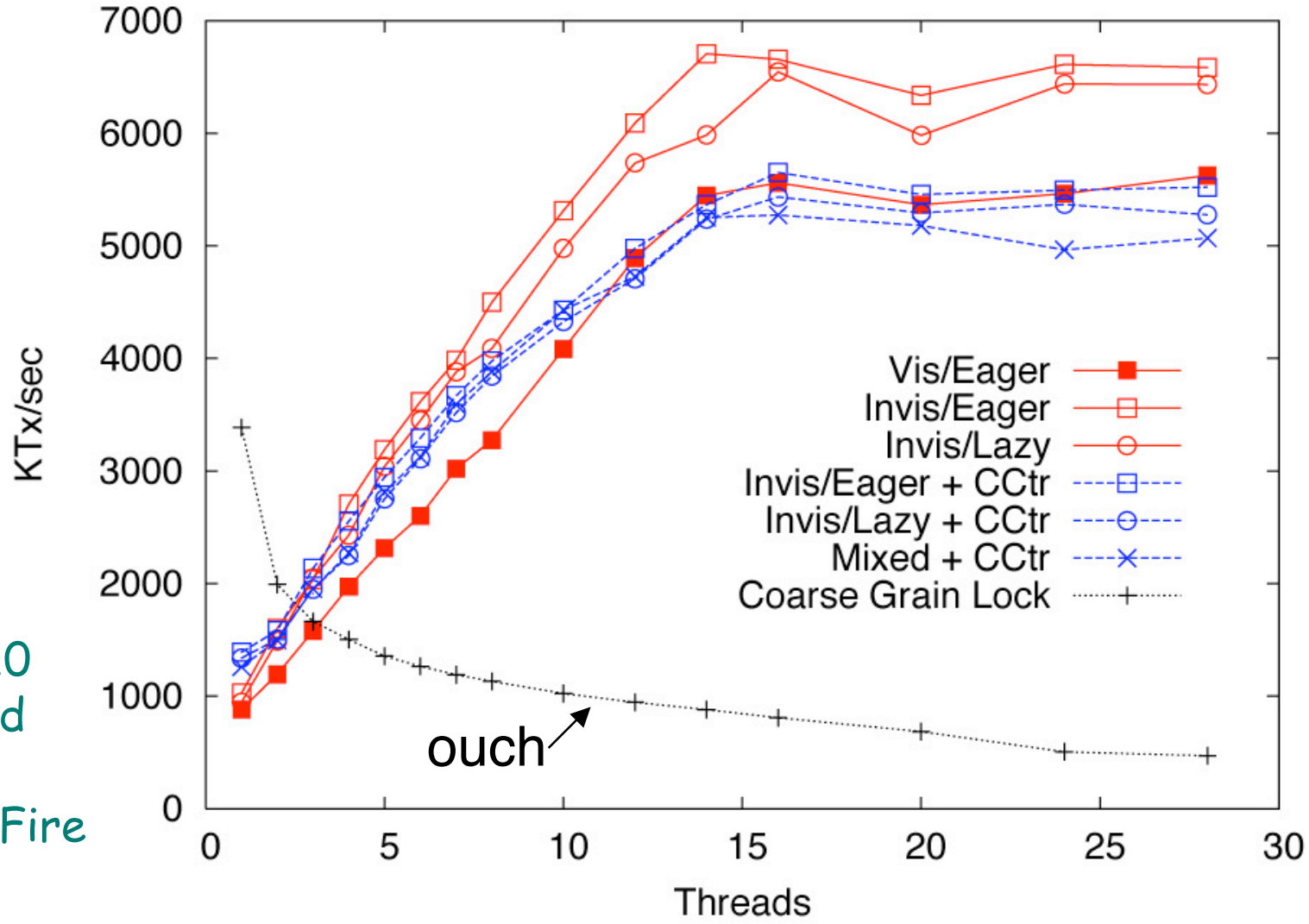
What to do?

- Complete sandboxing (high constant OH)
- Visible readers (heavy cache penalty)
- Incremental validation ($O(n^2)$ total cost)
- HW assist (not in this talk :-)
- Dump it on the programmer
- Incremental + heuristics
  - » Lev & Moir: per-object reader count, global conflict count; skip validation if the latter does not change
    - 2N extra atomic ops for an N-object reader, most of which will miss in the cache

The Global Commit Counter

- Shared count of transactions that have tried to commit
- Check on every new open, and prior to commit
- If value has changed, do incremental validation; otherwise transaction is still safe
  - » approximates mixed invalidation
  - » (we also built true mixed; see paper for details)
- If value never seems to change, can optimistically elide bookkeeping and contention management
  - » single-thread optimization
  - » (must still update and check the global counter)

Hash Table

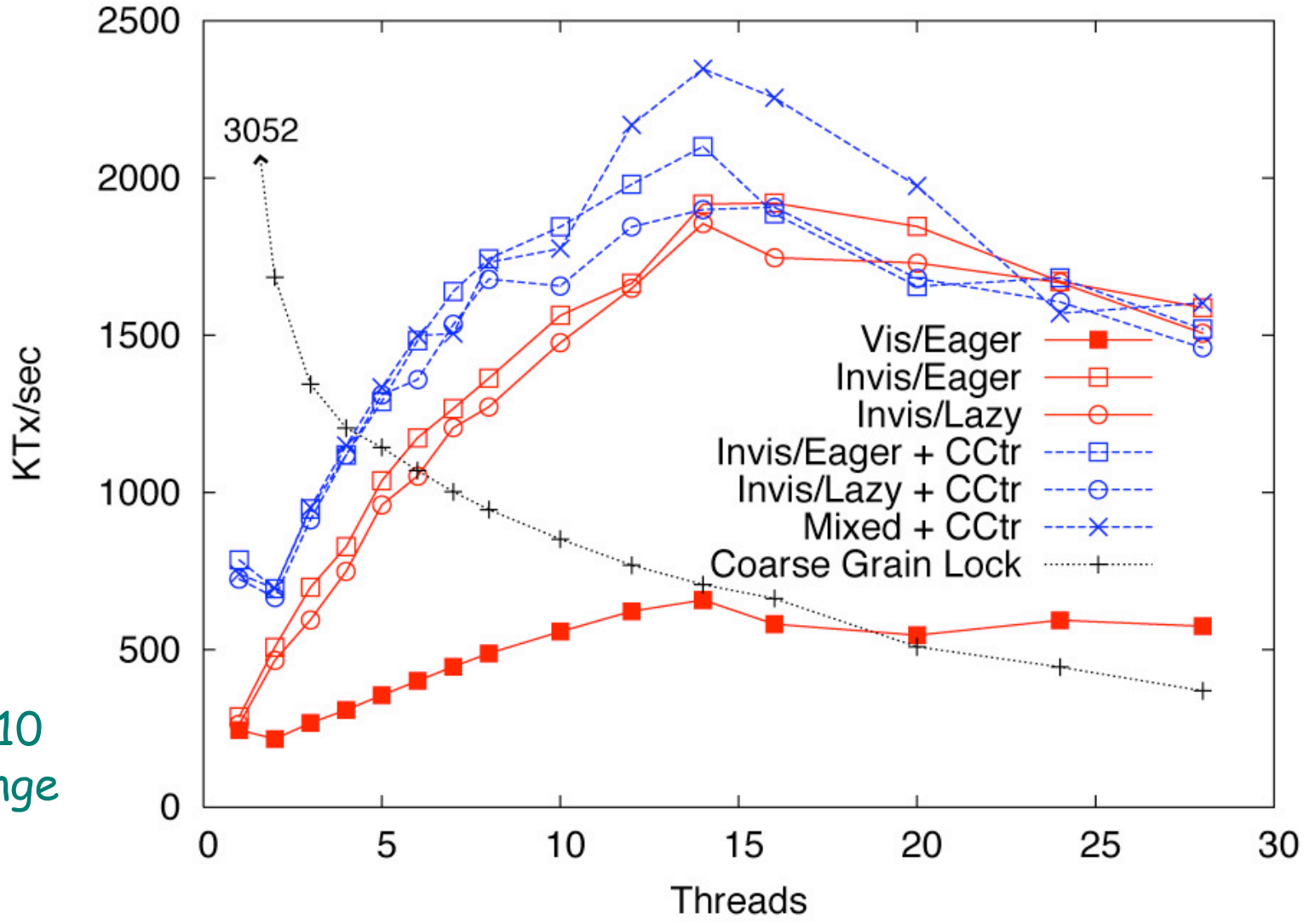


80/10/10
50% load

16p SunFire

- Very few objects read; invisible readers work great
- Very little contention; global commit counter doesn't help

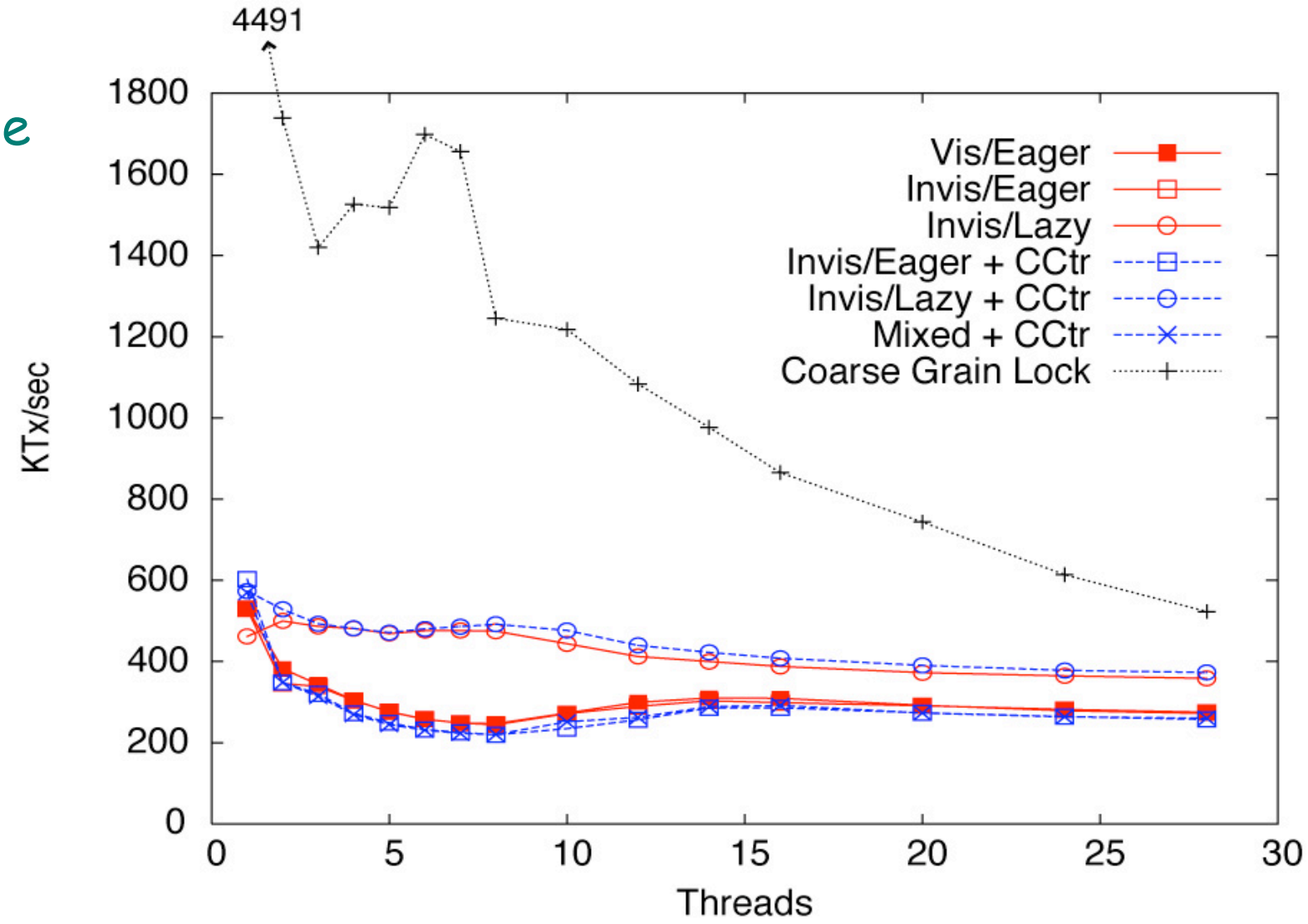
Red-Black Tree



80/10/10
64K range

- Modest contention; visible reads are too expensive
- ~16 objects read —> sig. validation OH; CCtr, Mixed work well

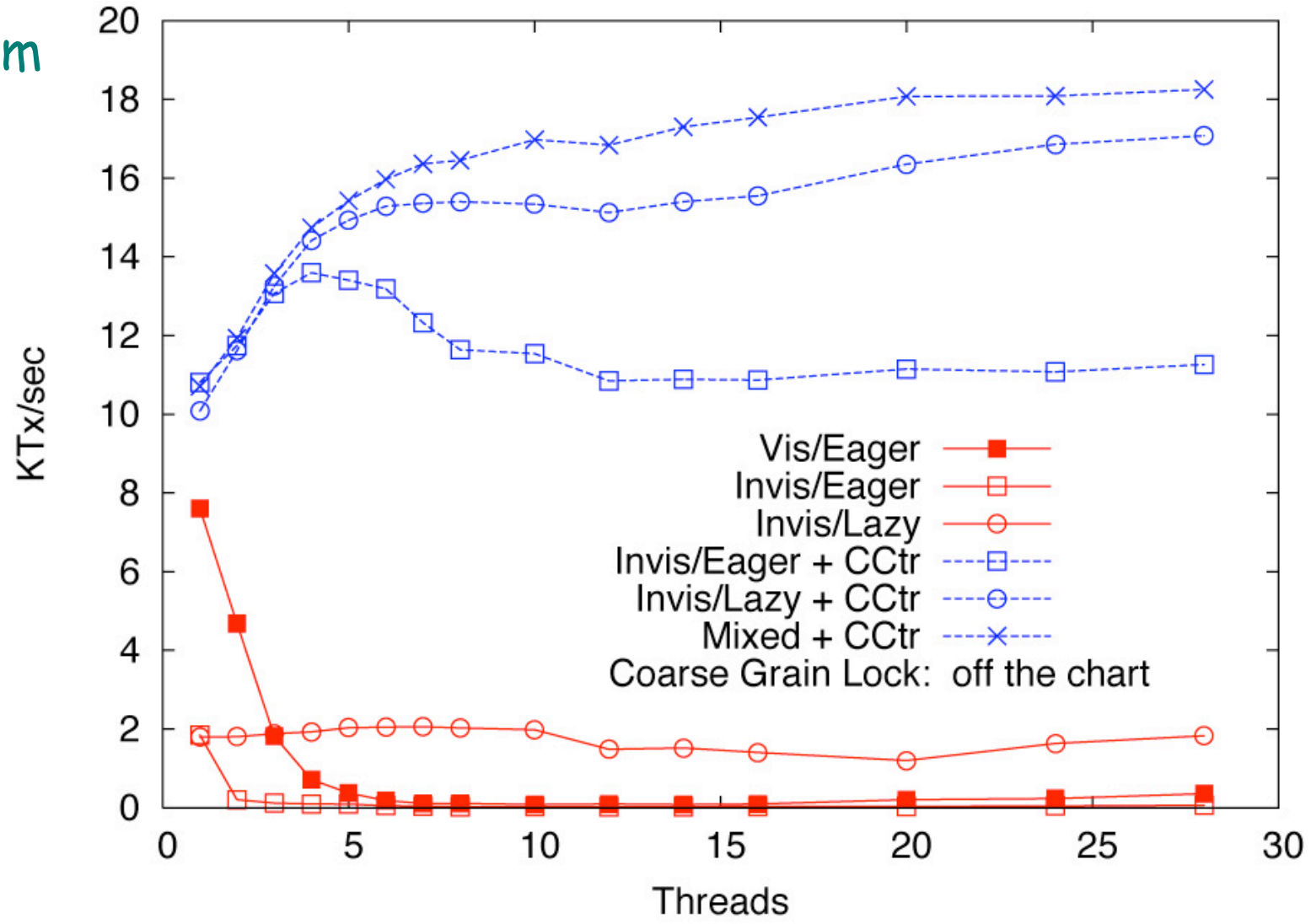
LFU Cache



index + PQ; Zipf

- Extreme case: short transactions, high contention —> Invis/Lazy
- Validation not significant —> CCtr doesn't help much

Random Graph



~128 nodes; degree avg. ~4

- Torture test: long transactions, high contention
- CCtr, Mixed a huge win, but coarse grain lock much better

Conclusions

- Conflict detection and validation matter
  - » dramatic effect on performance
- No overall winning policy—one size does *not* fit all
  - » adaptation a promising direction for future work
- Visible readers are rarely worthwhile
  - » too many cache misses
- Global commit counter a valuable heuristic
  - » can dramatically reduce the cost of validation
  - » approximates mixed invalidation
  - » 20% – 5X gain when running single-threaded
  - » may hurt a little in low-contention SMP workloads

Status and Plans

- RSTM 2 available for download
  - » www.cs.rochester.edu/research/synchronization/rstm/
  - » for gcc C++ on SPARC v8-plus and x86
  - » Java version planned for later this year (collaboration w/Sun)
- Formalization of STM semantics } TRANSACT'06
- RTM HW/SW hybrid
- Planning compiler support
- Numerous semantic/notational issues (ask me later!)
- Benchmarks and applications
- Interoperation with lock-based and nonblocking code

UNIVERSITY OF
ROCHESTER
COMPUTER SCIENCE

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