Hardware Acceleration of Software Transactional Memory

Arrvindh Shriraman, Virendra J. Marathe
Sandhya Dwarkadas, Michael L. Scott
David Eisenstat, Christopher Heriot, William N. Scherer III, Michael F. Spear

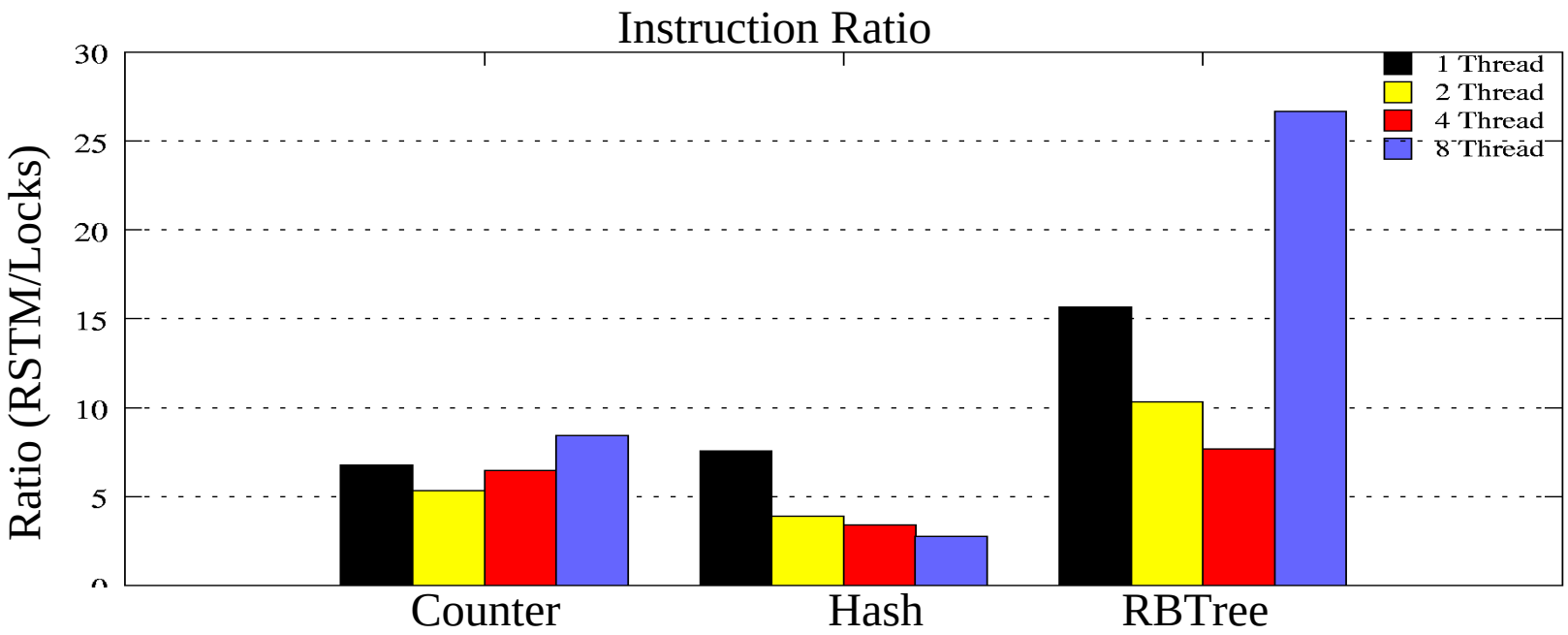
Department of Computer Science

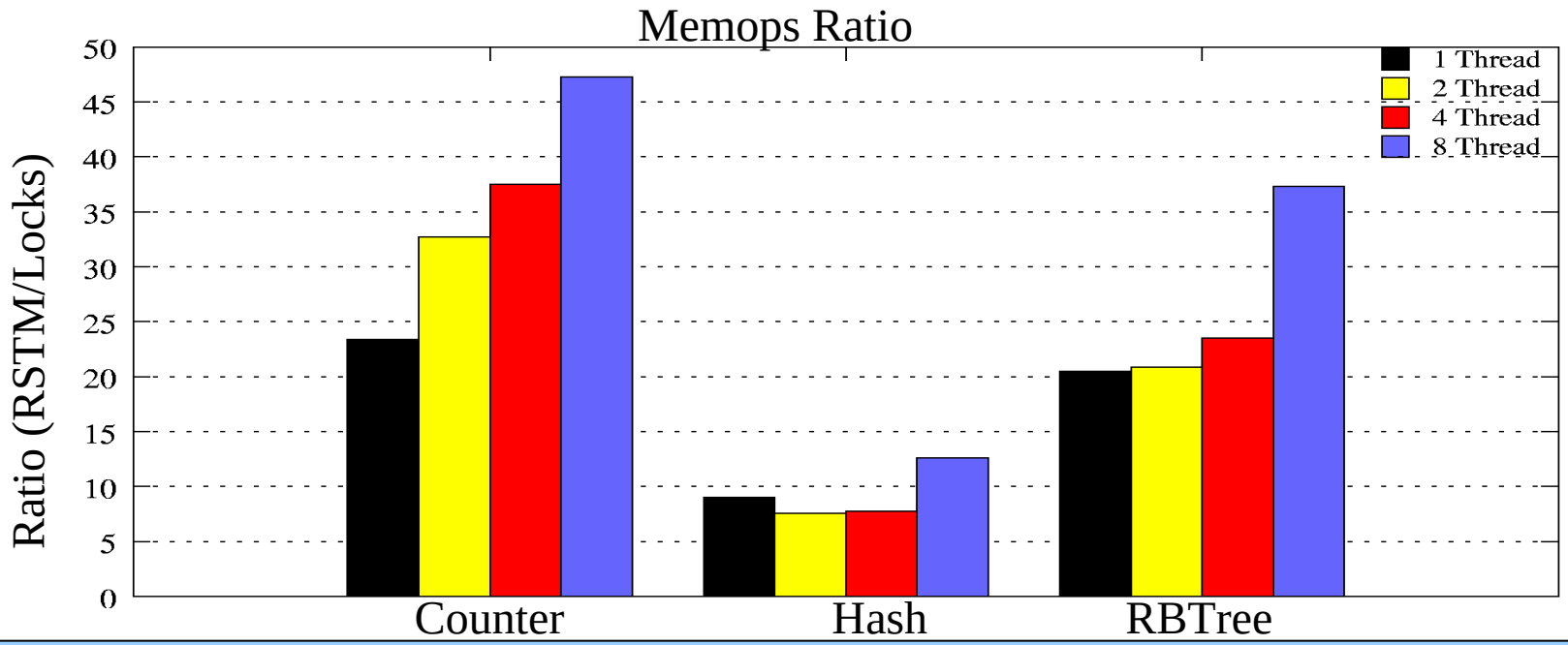
University of Rochester

Hardware and Software TM

- Software
  - – High runtime overhead
  - + Policy flexibility
    - conflict detection
    - contention management
    - non-transactional accesses
- Hardware
  - + Speed
  - – Premature embedding of policy in silicon

RSTM Overhead w.r.t. Locks per Txn





STM Performance Issues

- Memory management overhead
  - Garbage collection, object cloning/buffering of writes
  - Multiple pointer chasing required to access object data

- Validation overhead
  - Visible readers require 2N CASs to read N objects
  - Invisible readers need to perform bookkeeping and validation – $O(N^2)$ operation with N objects

RTM: HW-Assisted STM

- Leave (almost) all policy in SW
  - don't constrain conflict detection, contention mgmt, non-transactional accesses, irreversible ops
- HW for in-place mutation
  - eliminate copying, memory mgmt
- HW for fast invalidation
  - eliminate validation overhead

Outline

- RTM API and Software Metadata
- Support for isolation
  - TMESI coherence protocol with concurrent readers and writers
- Abort-on-invalidate
- Policy flexibility
- Preliminary results

RTM API and Object Metadata

Threads define a set of objects as shared with associated metadata headers

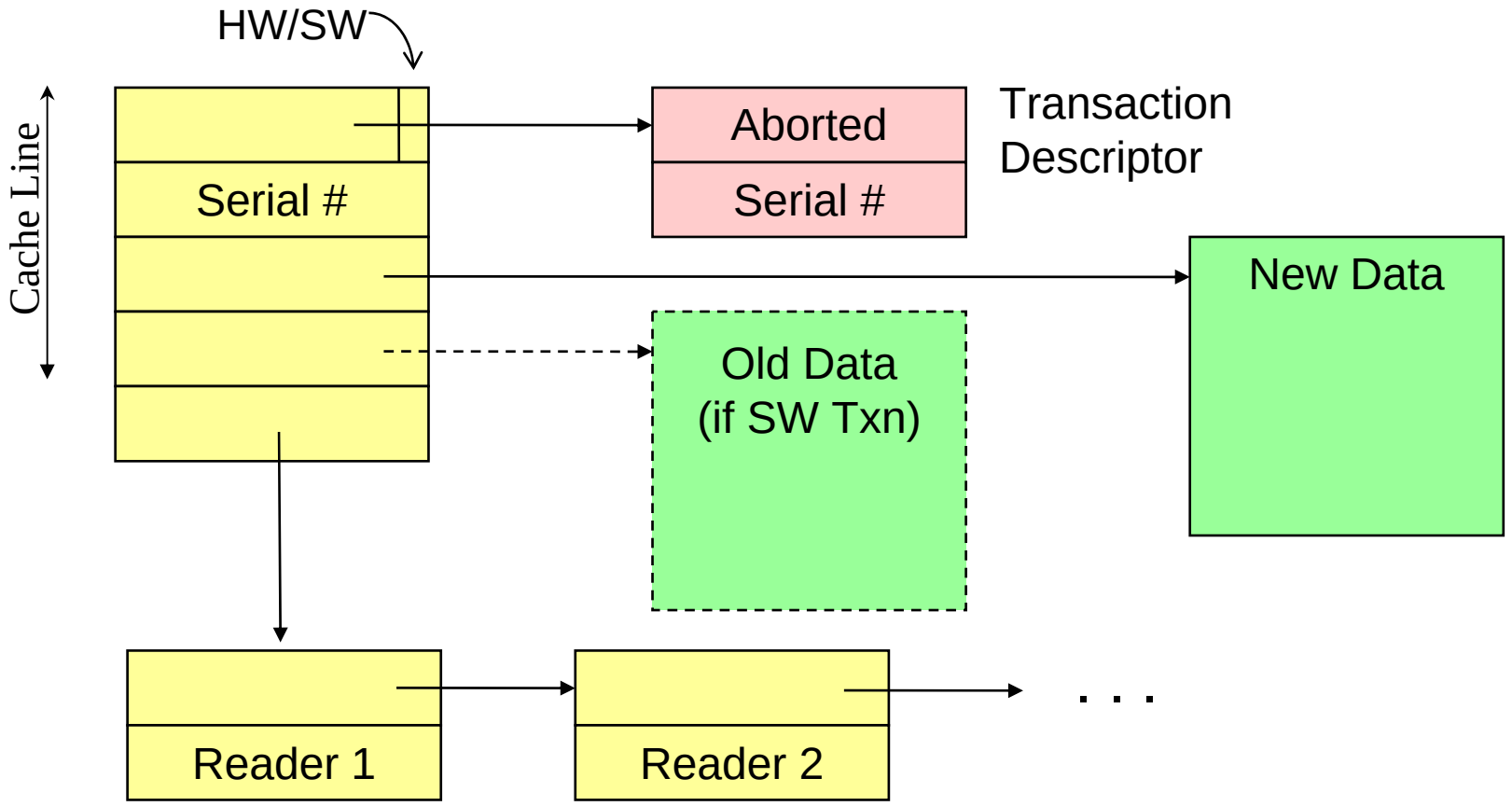
Transactions involve

(1) Indicating start of transaction and registering abort-handler PC

(2) Opening object metadata before reading/writing object data

(3) Acquiring ownership of all objects that are written

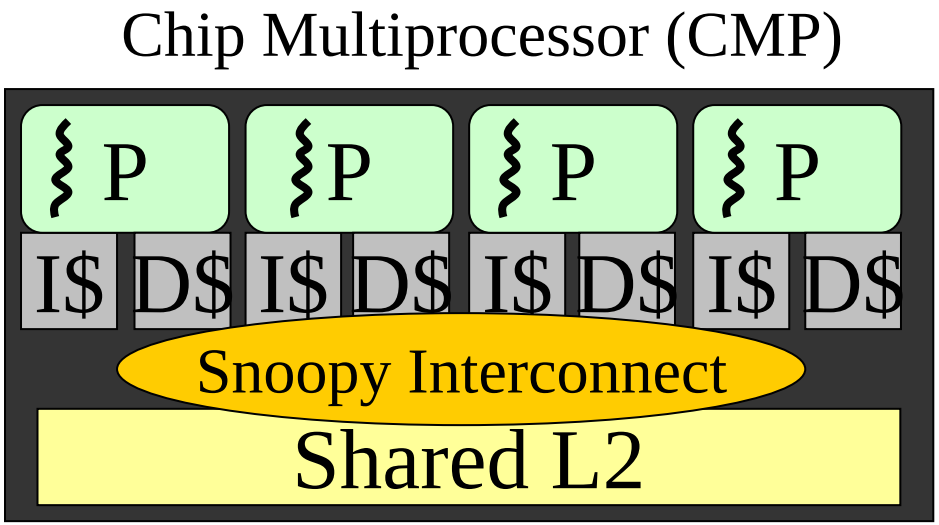
(4) Switching status atomically to committed, if still active.



RTM Highlights

- Leave policy decisions in software
  - A multiple writer hardware coherence protocol (TMESI) to achieve isolation, along with lightweight commit and abort
  - Hardware conflict detection support and contention management controlled by software
- Eliminate the copying overhead
  - Employ caches as thread local buffers
- Minimize the validation overhead
  - Provide synchronous remote thread aborts
- Fall back to SW on context switch or overflow of cache resources

Prototype RTM TMESI



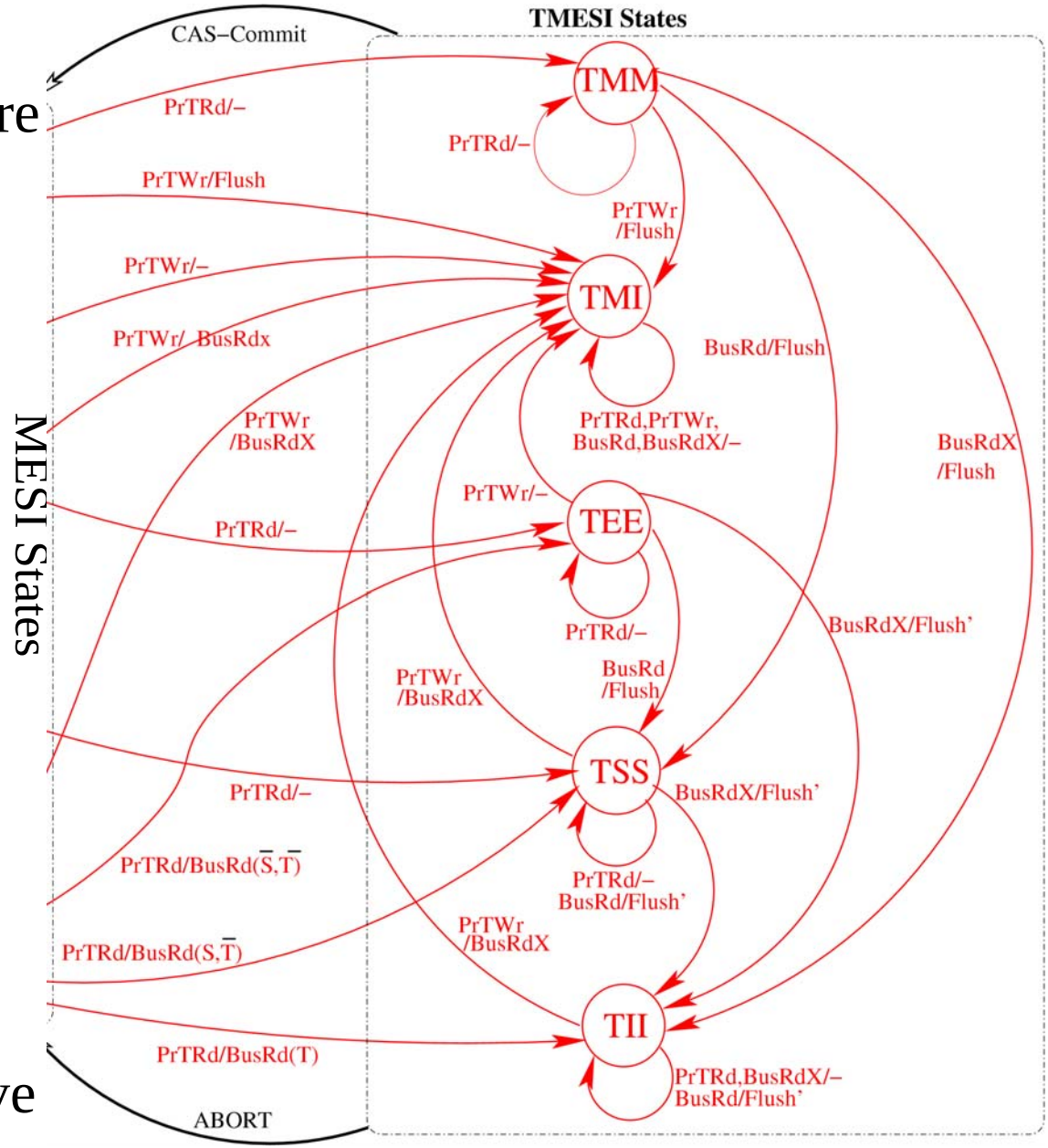
- Prototype system
  - CMP, 1 Thread/core
  - Private L1 caches
  - Shared L2 cache
  - Totally-Ordered network
- Additions to the base MESI coherence protocol
  - Transactional and abort-on-invalidate states

Transactional States

- T-MM/EE/SS analogous to M/E/S
  - -Writes from other transactions are isolated; BusRdX results in dropping to TII

- TMI buffers/isolates transactional stores
  - - supports concurrent writers; BusRdX ignored
  - - supports concurrent readers; BusRd threatened and data response suppressed

- TII isolates concurrent readers from transactional writers
  - -Threatened cache line reads move to TII

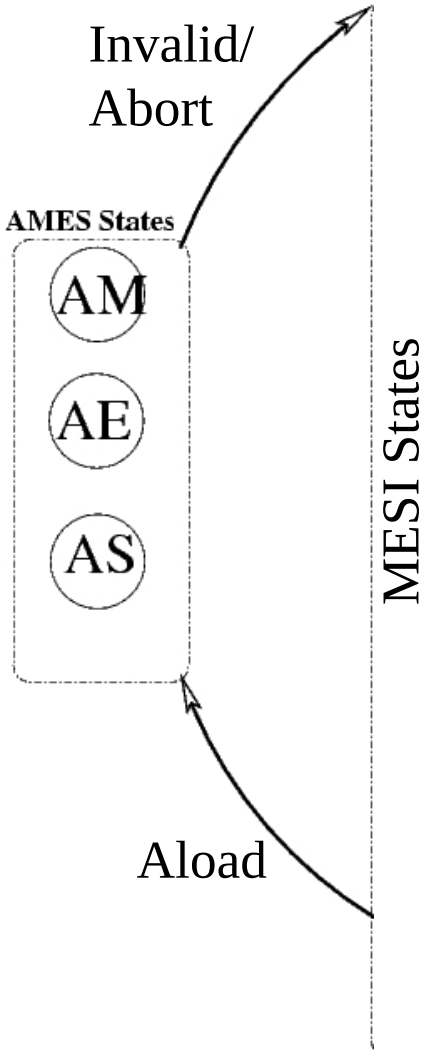


All cache lines in TMESI return to MESI on commit/abort.

Transactional (Speculative) Lines

- TLoaded lines
  - can be dropped without aborting
- TStored lines
  - must remain in cache (or txn falls back to SW)
  - revert to M on commit, I on abort
- Support R-W and W-W speculation (if SW wants)
- No extra transactional traffic; no global consensus in HW
  - commit is entirely local; SW responsible for correctness

Abort on Invalidate



•A-tagged line invalidation aborts a transaction and jumps to a software handler

•Invalidation can be due to

- Capacity: Abort since cache cannot track conflicts for object
- Coherence: Remote potential writer/reader of the object cache line has acquired object ownership

• Transactional object headers when ALoaded in open() eliminate the need for

-incremental validation

-explicitly visible hardware readers

• Transaction descriptors are ALoaded by all transactions, allowing synchronous aborts

ISA Additions

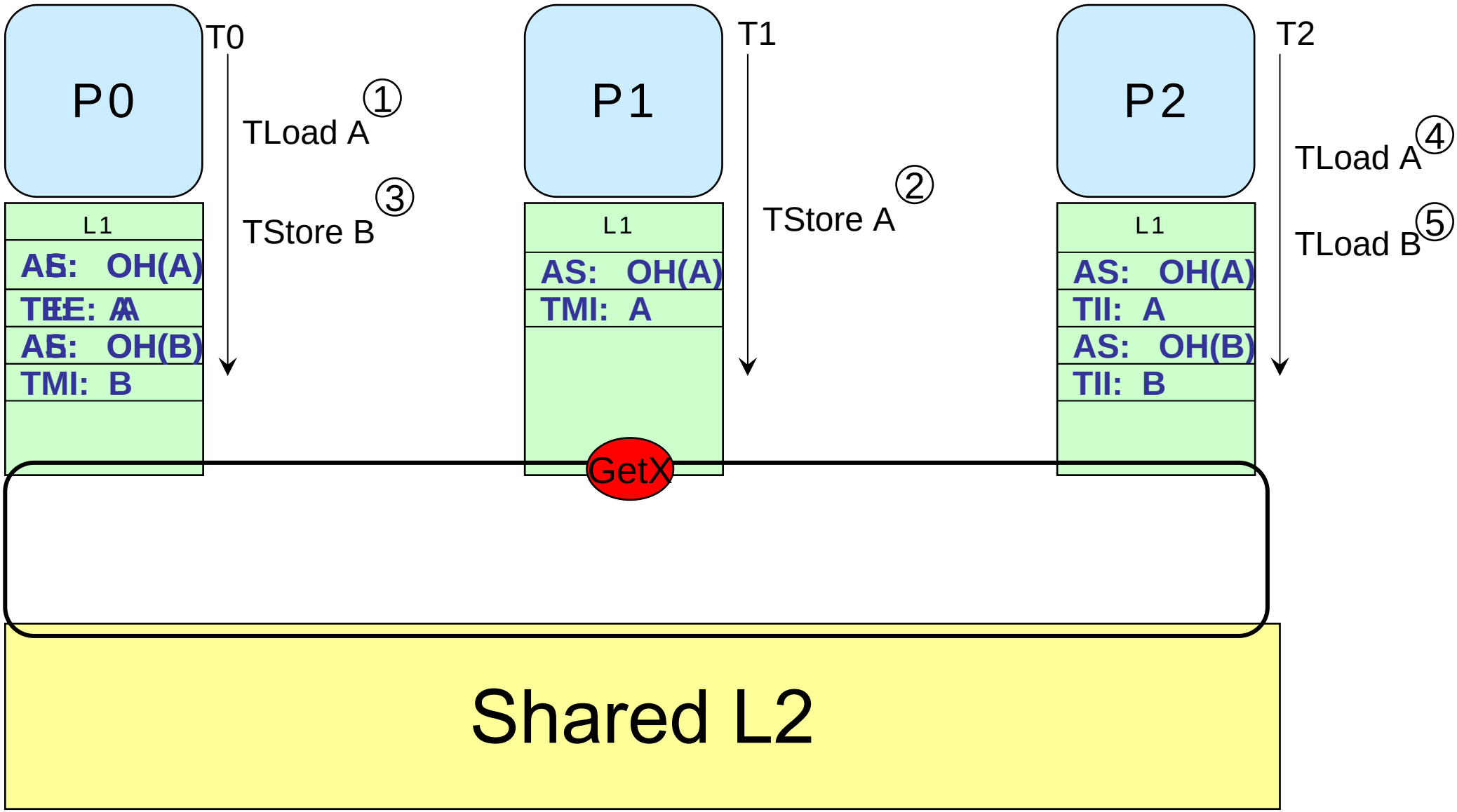
- TLoad, TStore — transactional (speculative) load, store
- ALoad, ARelease — abort if line is invalidated
- SetHandler — where to jump on abort
- CAS-Commit — if successful, revert T&A lines
- Abort — self-induced, for condition synchronization
- 2-C-4-S — if compare succeeds, swap 4 words (example, IA-64)

RTM Policy Flexibility

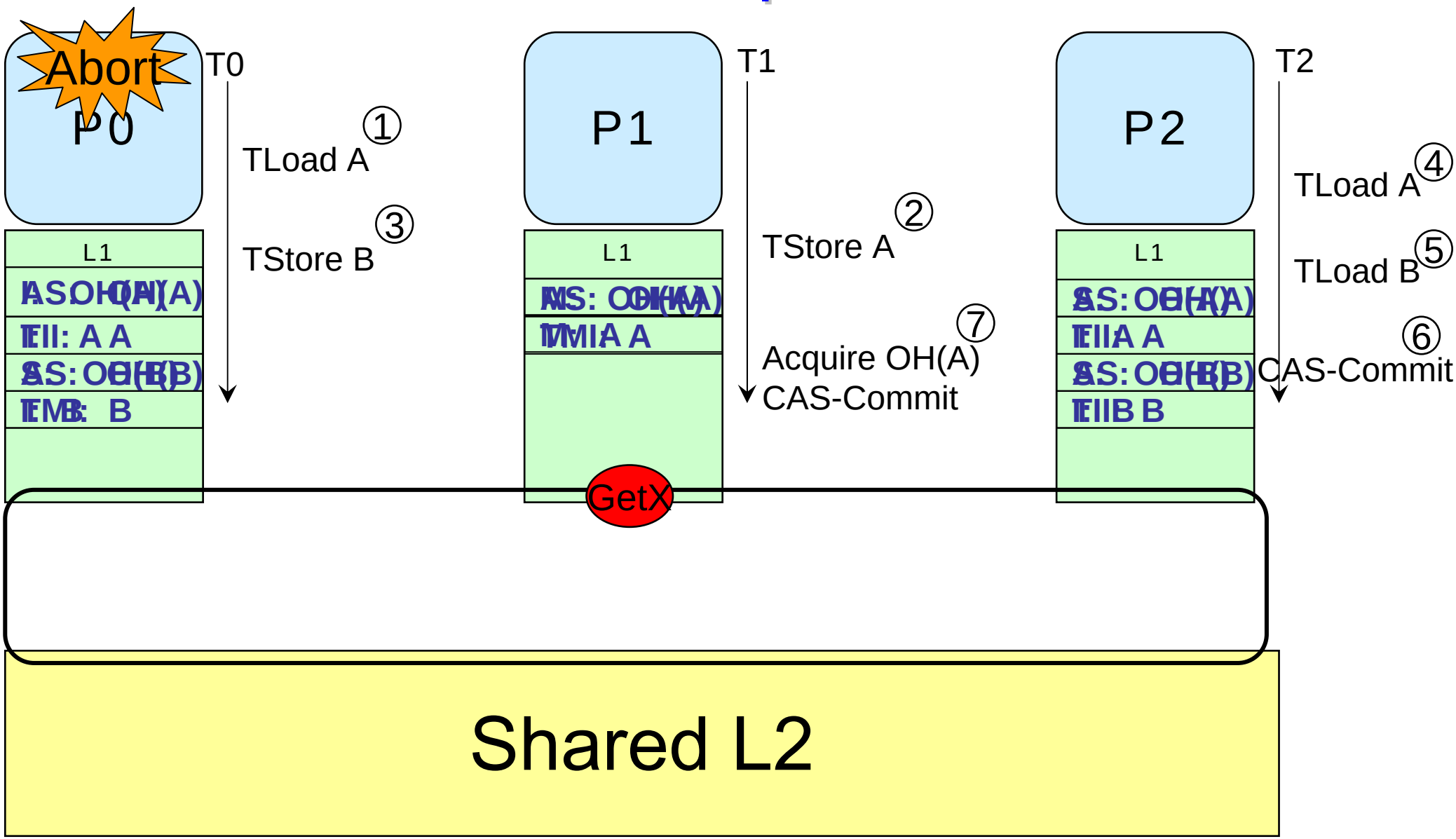
- Conflict detection
  - Eager (i.e., at open())
  - Lazy (i.e., at commit)
  - Mixed (i.e., eager write-write detection and lazy read-write detection)

- Flexible software contention managers
  - Contention managers arbitrate among conflicting transactions to decide the winner

Example


P0
T0
TLoad A
①
TStore B
③
L1
AS: OH(A)
TMI: A
AS: OH(B)
TMI: B
P1
T1
TStore A
②
L1
AS: OH(A)
TMI: A
GetX
P2
T2
TLoad A
④
TLoad B
⑤
L1
AS: OH(A)
TMI: A
AS: OH(B)
TMI: B
Shared L2

Example

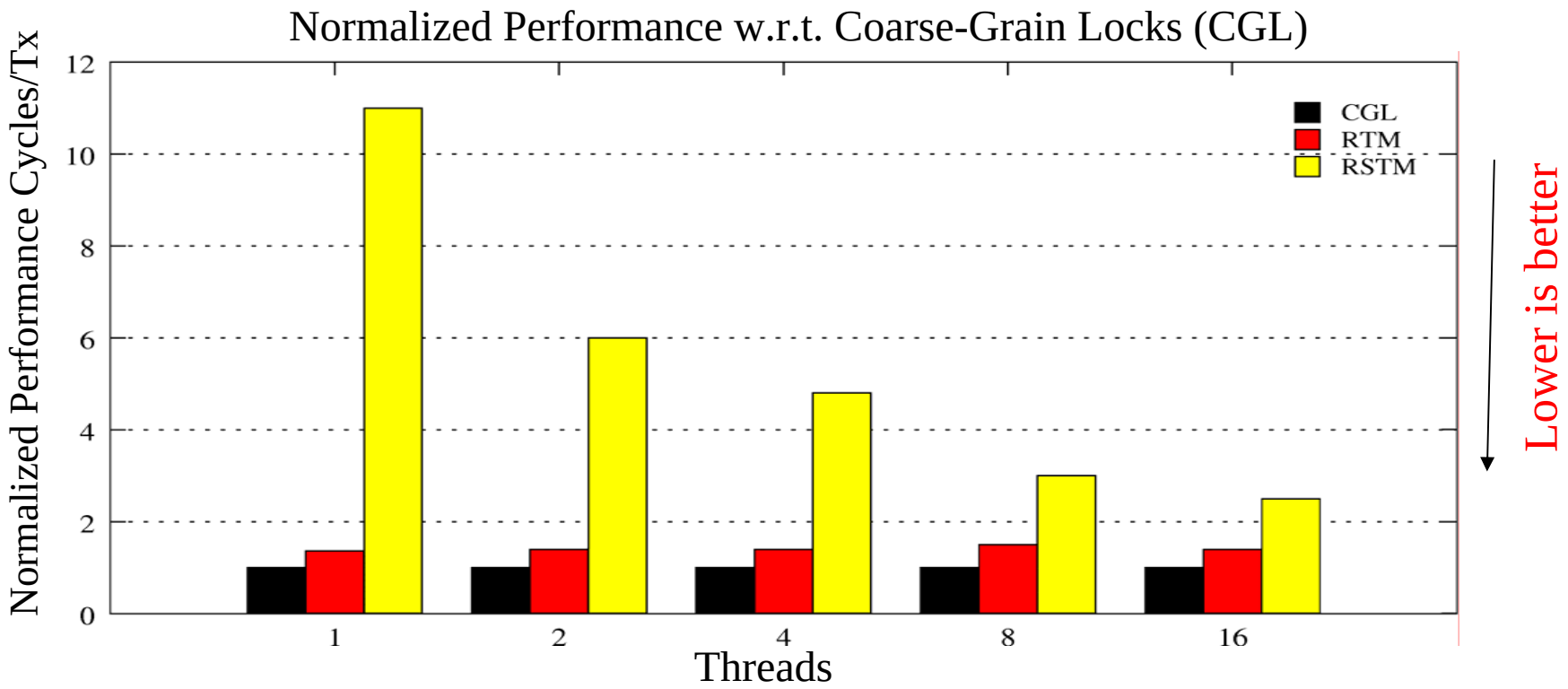

Abort
P0
T0
TLoad A
①
TStore B
③
L1
P1
T1
TStore A
②
Acquire OH(A)
⑦
CAS-Commit
L1
GetX
P2
T2
TLoad A
④
TLoad B
⑤
⑥
CAS-Commit
L1
Shared L2

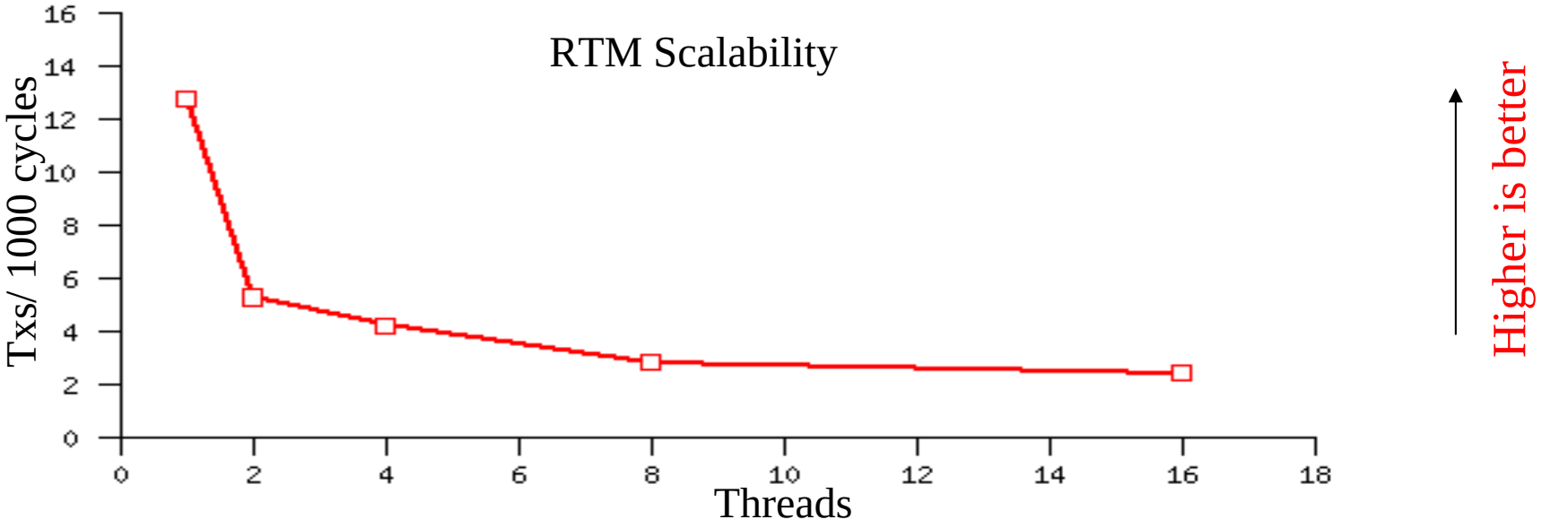
Simulation Framework

- Target Machine:16 way CMP running Solaris 9
  - 16 SPARC V9 processors
  - 1.2GHz in-order processors with ideal IPC=1
  - 64KB 4-way split L1 cache, latency=1 cycle
  - 8MB 12way L2 with 16 banks, latency=20cycle
  - 4-ary hierarchical tree
    - Broadcast address network and point-point data network
    - On-Chip link-latency=1cycle
  - 4GB main memory , 80 cycle access latency
  - Snoopy broadcast protocol

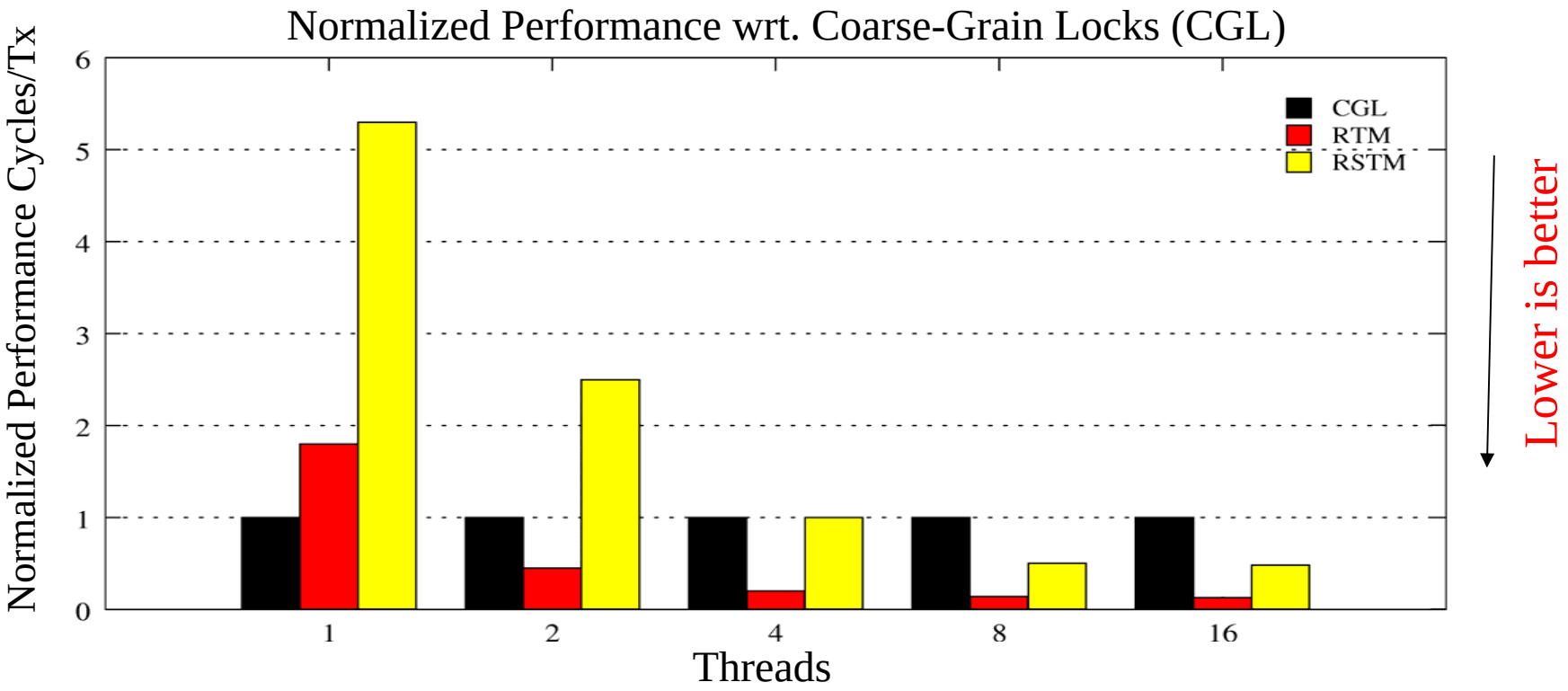
- Infrastructure
  - Virtutech Simics for full-system function
  - Multifacet GEMS [Martin et. al, CAN 2005] Ruby framework for memory system timing
  - Processor ISA extensions implemented using SIMICS magic no-ops
  - Coherence protocol developed using SLICC [Sorin et. al, TPDS 2002]

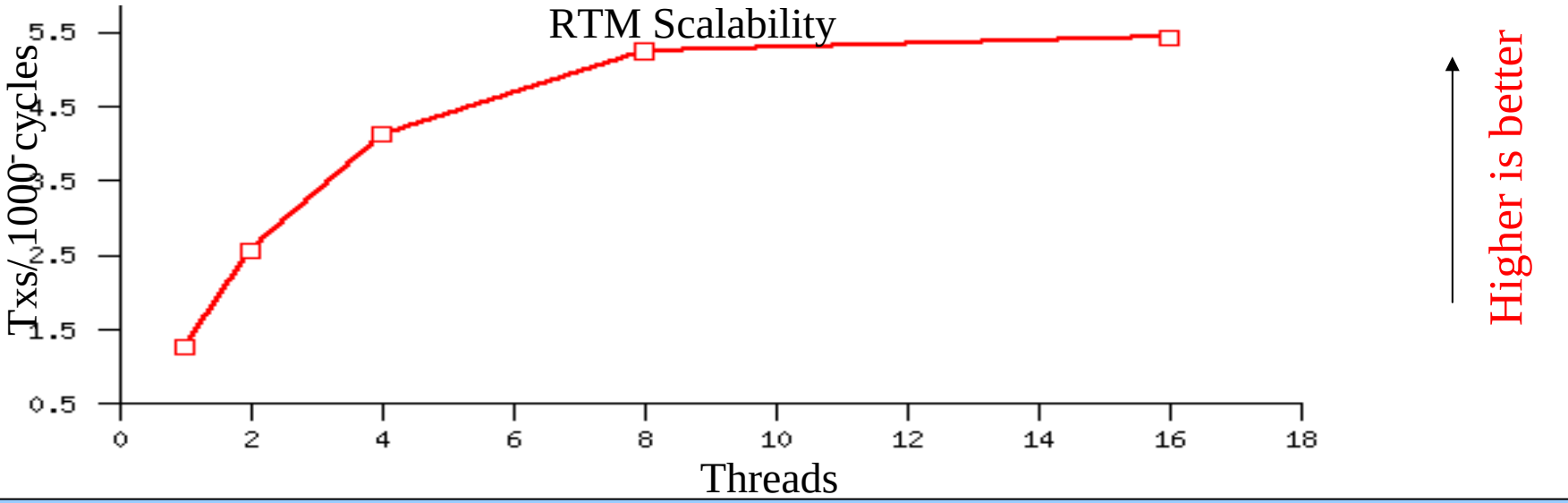
Shared Counter





Hash Table


Normalized Performance wrt. Coarse-Grain Locks (CGL)
Normalized Performance Cycles/Tx
CGL
RTM
RSTM
Threads
Lower is better


RTM Scalability
Txs/1000 cycles
Threads
Higher is better

Conclusions

- Coherence protocol additions at the L1 cache allow
  - Transactional overhead reductions in copying and conflict detection in order to enforce isolation
  - Flexible policy decisions implemented in software that improve the scalability of the system
- Allowing software fallback permits transactions unbounded in space and time
- Additional features
  - Deferred aborts

Future Work

- A more thorough evaluation of the proposed architecture including
  - Effects of policy flexibility
- Extensions to multiple levels of sharing and to directory-based coherence protocols
- Incremental fallback to software for only those cache lines that don’t fit in the cache