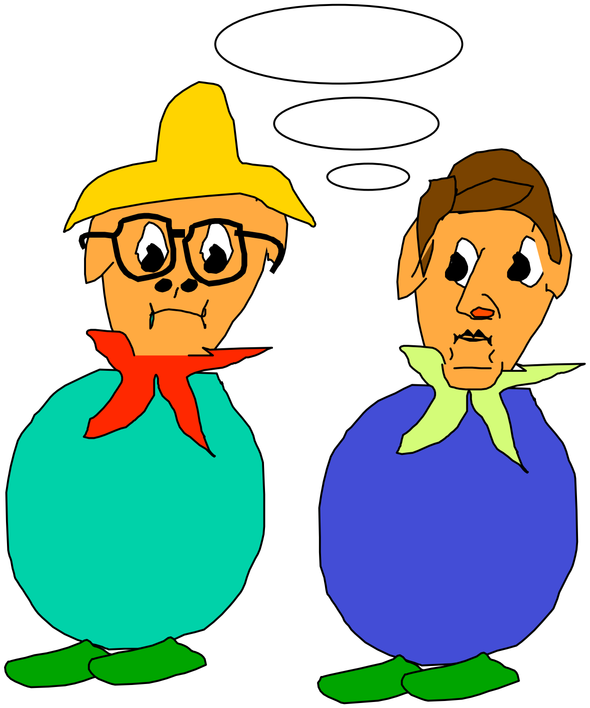
Transactional Locking II

Nir Shavit, Dave Dice and Ori Shalev

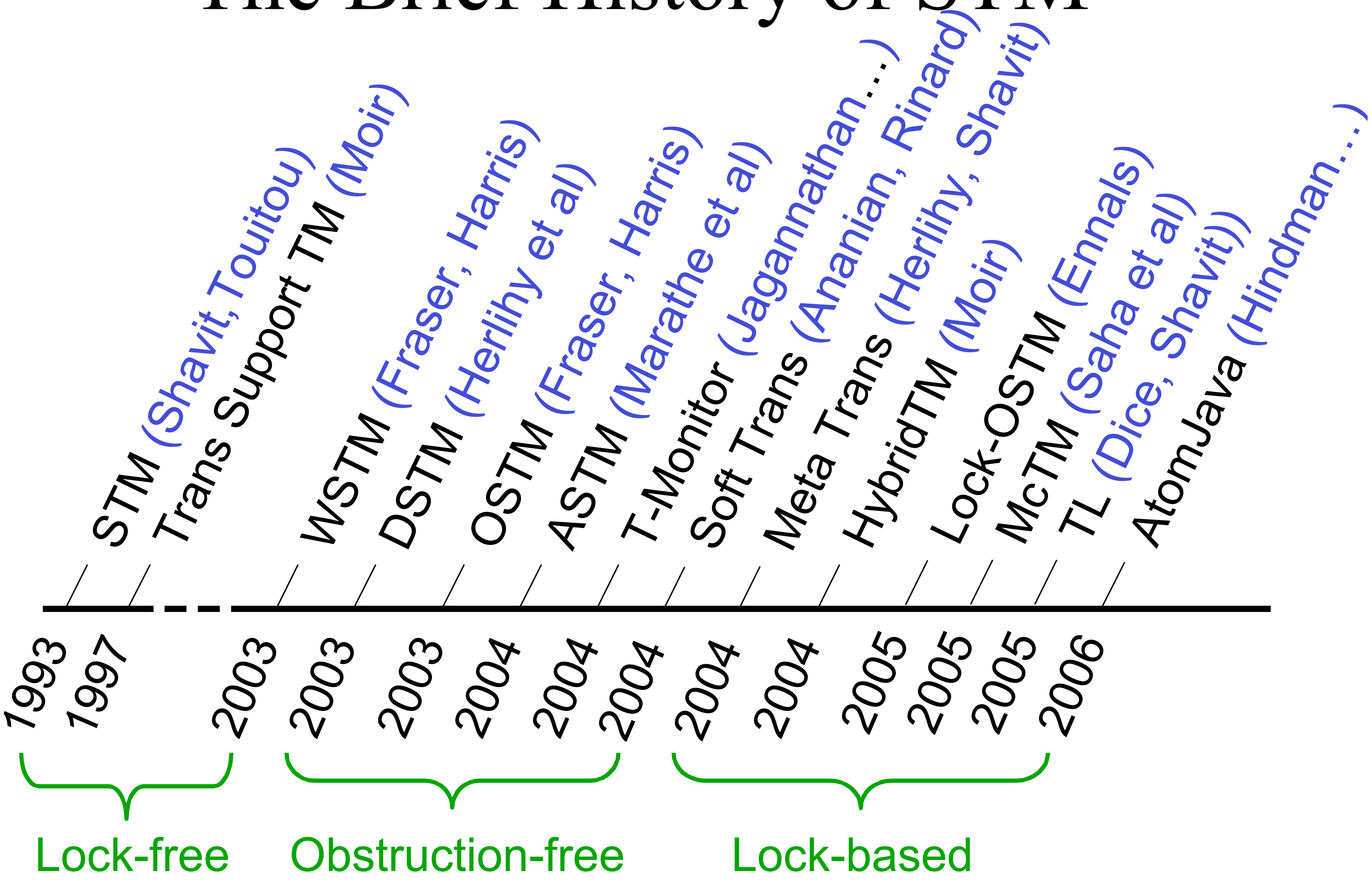
Scalable Synchronization Group
Sun Labs

Transactional Memory

[HerlihyMoss93]



The Brief History of STM



STM (Shavit,Touitou) 1993

Trans Support TM (Moir) 1997

WSTM (Fraser, Harris 2003

DSTM (Herlihy et al) 2003

OSTM (Fraser, Harris 2003

ASTM (Marathe et al) 2004

T-Monitor (Jagannathan…) 2004

Soft Trans (Ananian, Rinard) 2004

Meta Trans (Herlihy, Shavit) 2004

HybridTM (Moir) 2004

Lock-OSTM (Ennals) 2005

McTM (Saha et al) 2005

TL (Dice, Shavit)) 2005

AtomJava (Hindman…) 2006

Lock-free

Obstruction-free

Lock-based

As Good As Fine Grained

Postulate (i.e. take it or leave it):

If we could implement fine-grained locking with the same simplicity of course grained, we would never think of building a transactional memory.

Implication:

Lets try to provide TMs that get as close as possible to hand-crafted fine-grained locking.

Premise of Lock-based STMs

1. Performance: ballpark fine grained
2. Memory Lifecycle: work with GC or any malloc/free
3. Hardware➔Software: support voluptuous transactions

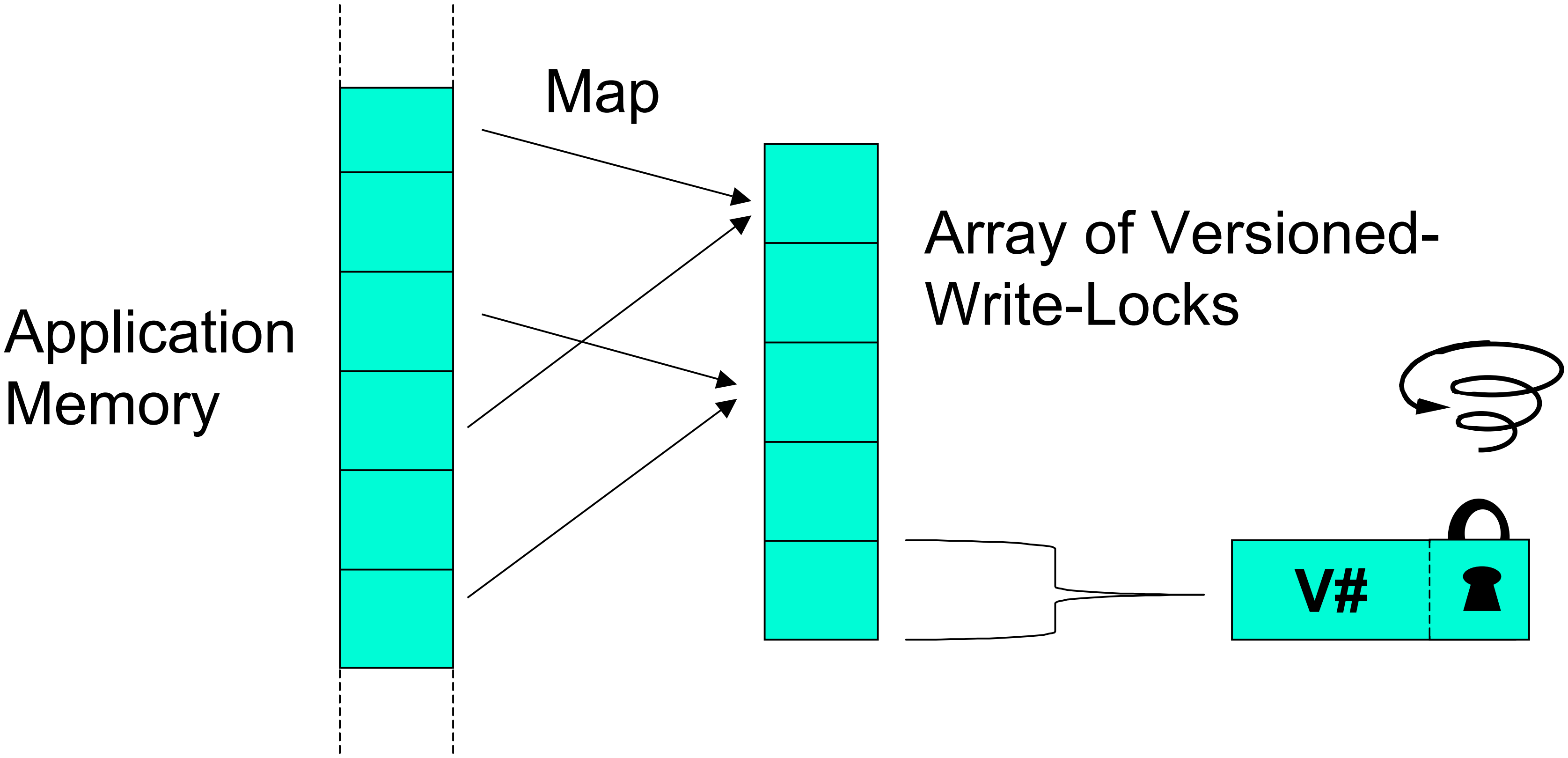
4. Safety: need to work on coherent state

   Unfortunately: OSTM, HyTM, Ennals, Saha, AtomJava deliver only 1 and 3 (in some cases)…

Transactional Locking

- TL2 Delivers all four properties
- How? use what we learned…
  - Unlike all prior algs: use Commit time locking instead of Encounter order locking
  - Introduce a Global Version Clock mechanism for validation

Locking STM Design Choices



Map

Application Memory

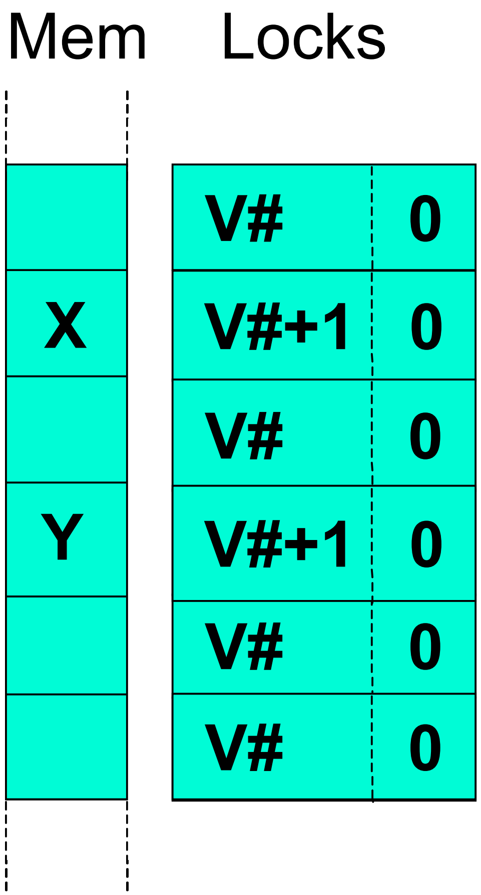
Array of Versioned-Write-Locks

PS = Lock per Stripe
(separate array of locks)

PO = Lock per Object
(embedded in object)

Encounter Order Locking (Undo Log)

[Ennals,Saha,Harris,…]



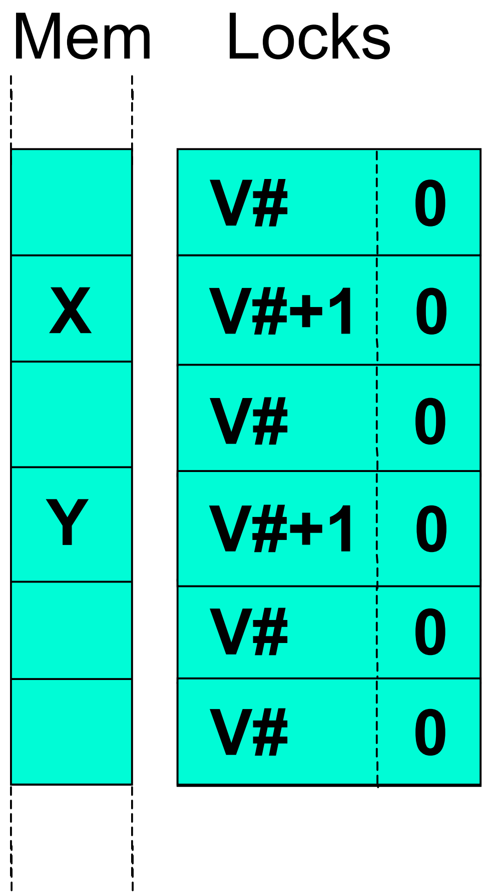
Mem	Locks	
	V#	0
X	V#+1	0
	V#	0
Y	V#+1	0
	V#	0
	V#	0

1. To Read: load lock + location
2. Check unlocked add to Read-Set
3. To Write: lock location, store value
4. Add old value to undo-set
5. Validate read-set v#'s unchanged
6. Release each lock with v#+1

Quick read of values freshly written by the reading transaction

Commit Time Locking (Write Buff)

[TL,TL2]



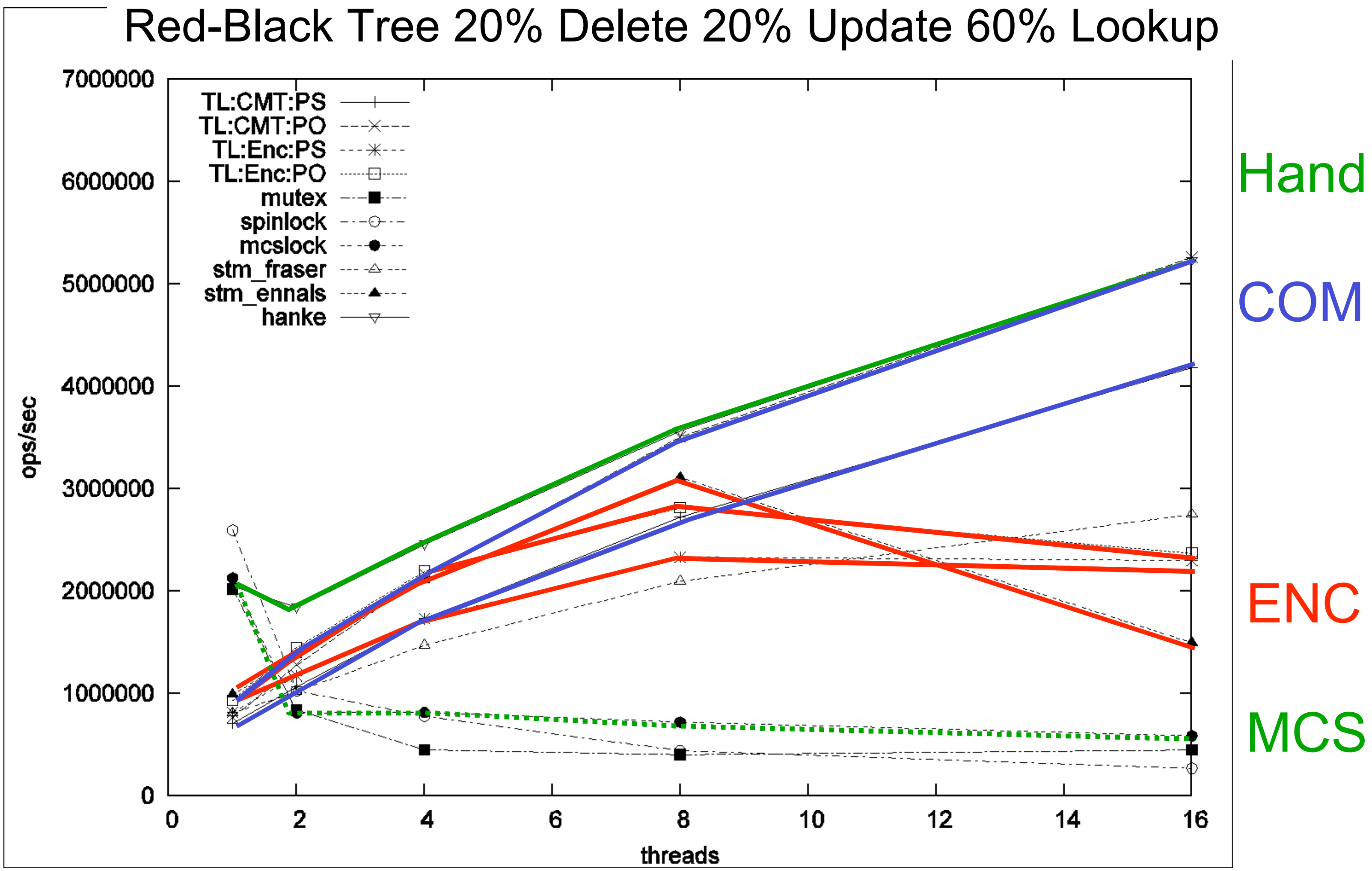
1. To Read: load lock + location
2. Location in write-set? (Bloom Filter)
3. Check unlocked add to Read-Set
4. To Write: add value to write set
5. Acquire Locks
6. Validate read/write v#'s unchanged
7. Release each lock with v#+1

Hold locks for very short duration

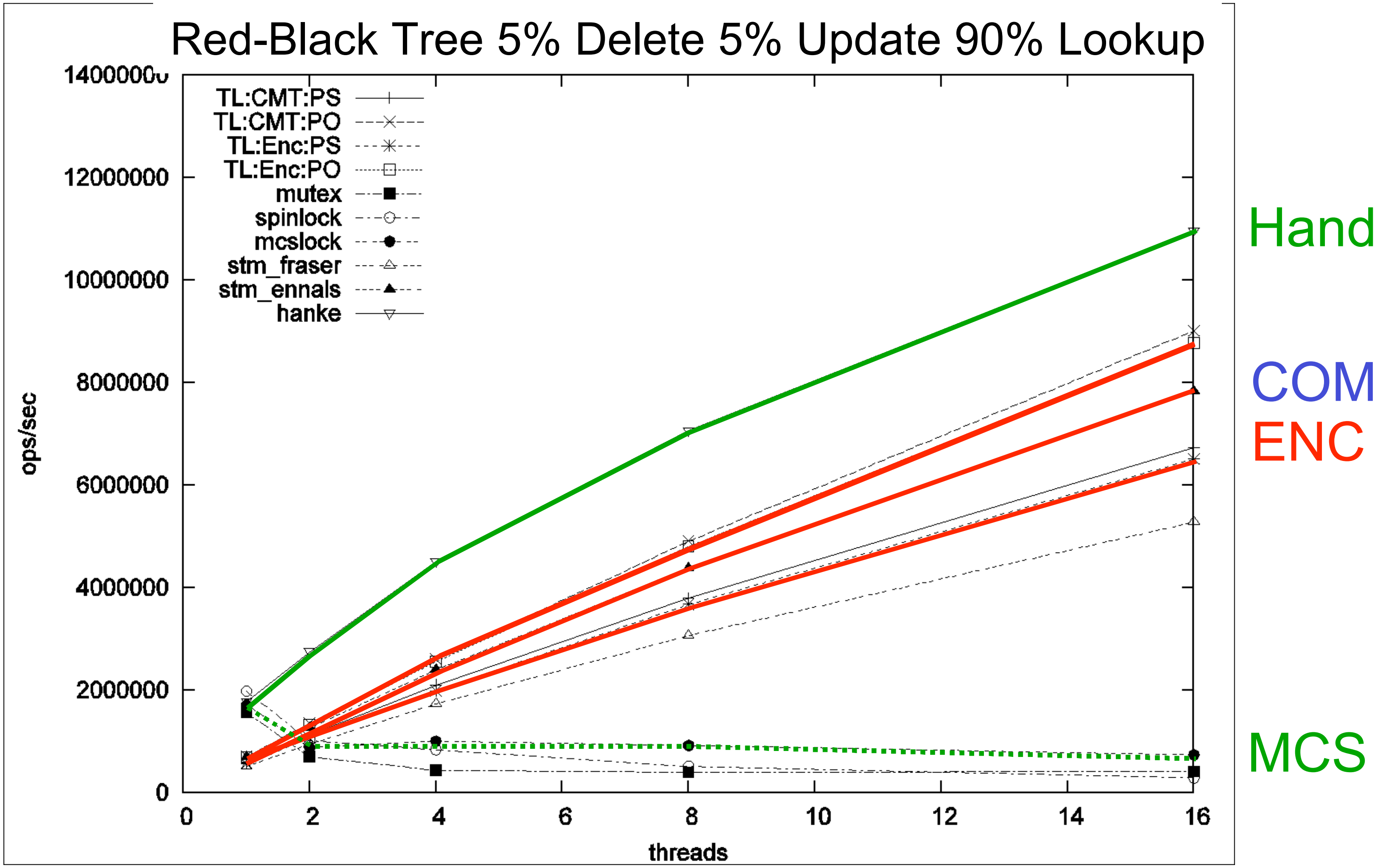
Why COM and not ENC?

1. Under low load they perform pretty much the same.
2. COM withstands high loads (small structures or high write %). ENC does not withstand high loads.
3. COM works seamlessly with Malloc/Free. ENC does not work with Malloc/Free.

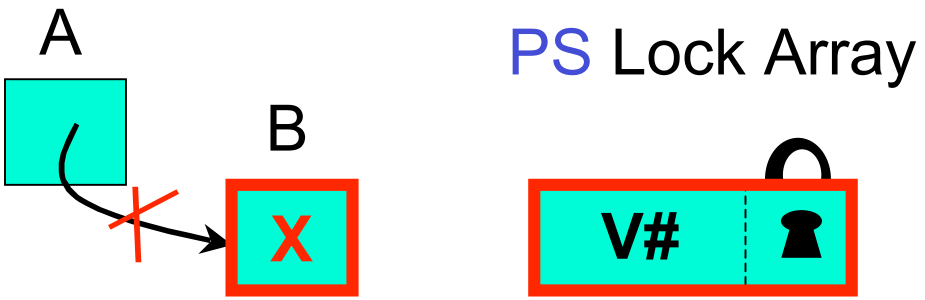
COM vs. ENC High Load



COM vs. ENC Low Load



COM: Works with Malloc/Free



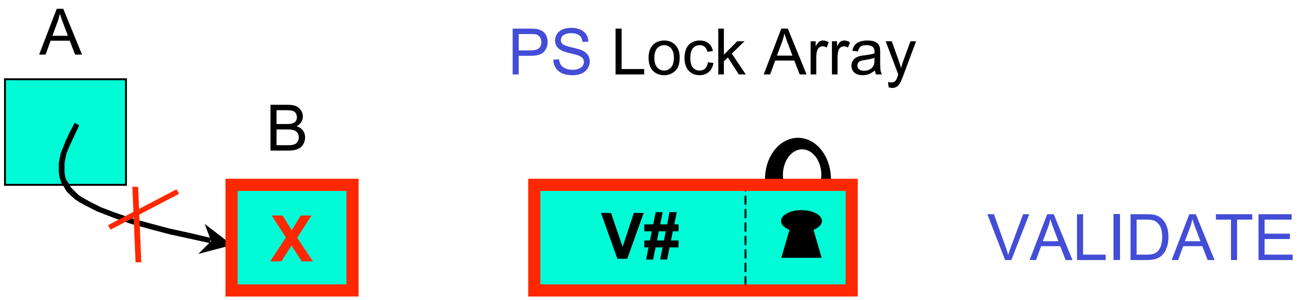
VALIDATE FAILS
IF INCONSISTENT

To free B from transactional space:

1. Wait till its lock is free.
2. Free(B)

B is never written inconsistently
because any write is preceded by
a validation while holding lock

ENC: Fails with Malloc/Free



Cannot free B from transactional space because undo-log means locations are written after every lock acquisition and before validation.

Possible solution: validate after every lock acquisition (yuck)

Problem: Application Safety

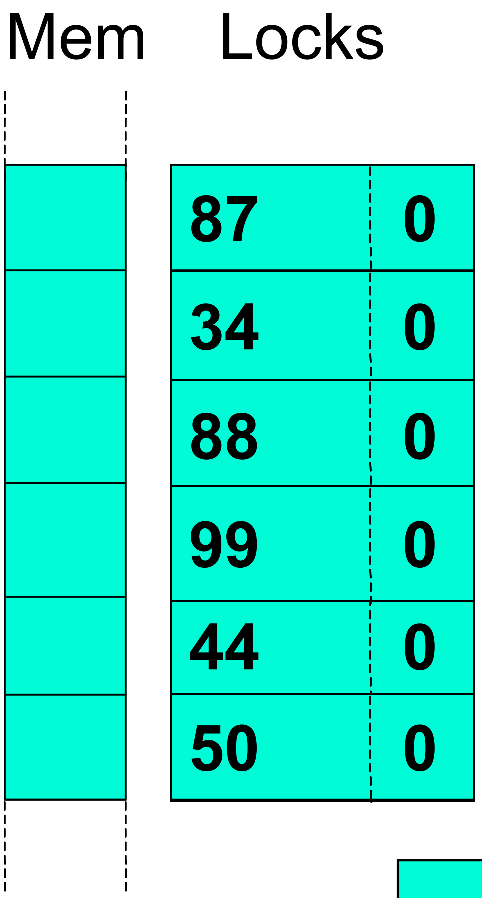
1. All current lock based STMs work on inconsistent states.
2. They must introduce validation into user code at fixed intervals or loops, use traps, OS support,…
3. And still there are cases, however rare, where an error could occur in user code…

Solution: TL2's "Version Clock"

- Have one shared global version clock
- Incremented by (small subset of) writing transactions
- Read by all transactions
- Used to validate that state worked on is always consistent

Later: how we learned not to worry about contention and love the clock

Version Clock: Read-Only COM Trans



Mem	Locks	
	87	0
	34	0
	88	0
	99	0
	44	0
	50	0

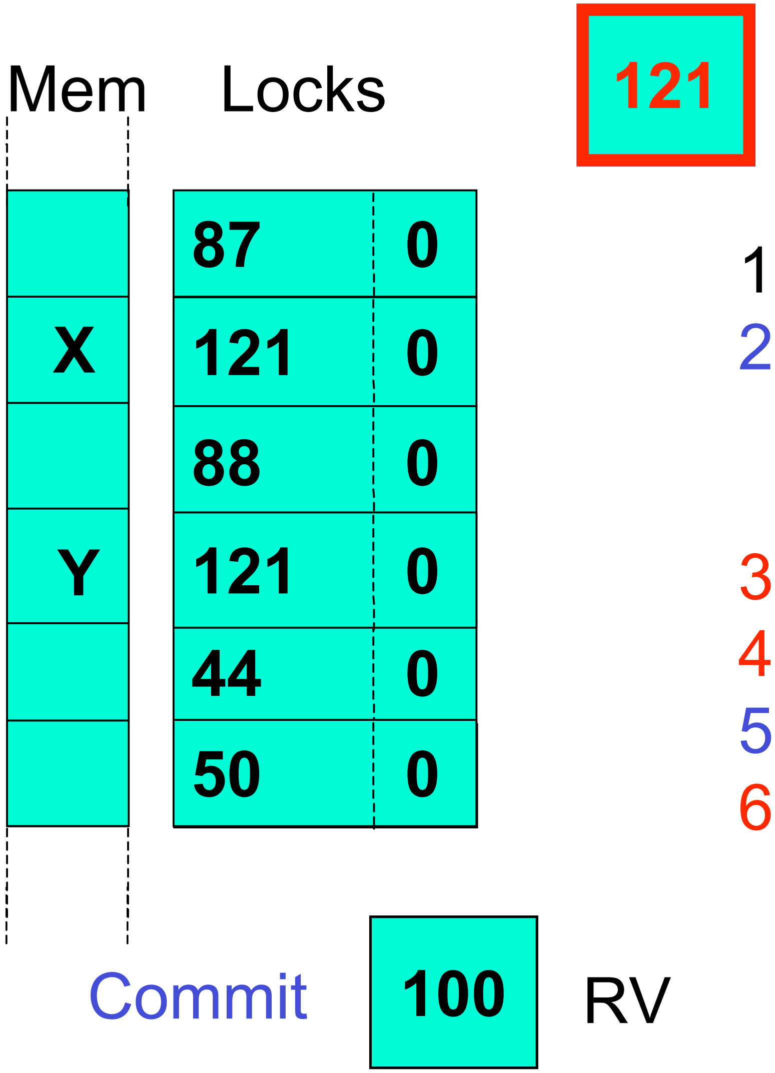
100 VClock

1. RV ← VClock
2. On Read: read lock, read mem, read lock: check unlocked, unchanged, and v# <= RV
3. Commit.

Reads form a snapshot of memory.
No read set!

100 RV

Version Clock: Writing COM Trans



Mem	Locks	
	87	0
X	121	0
	88	0
Y	121	0
	44	0
	50	0

121 VClock

Commit 100 RV

1. RV ← VClock
2. On Read/Write: check unlocked and v# <= RV then add to Read/Write-Set
3. Acquire Locks
4. WV = F&I(VClock)
5. Validate each v# <= RV
6. Release locks with v# ← WV

Reads+Inc+Writes =Linearizable

Version Clock Implementation

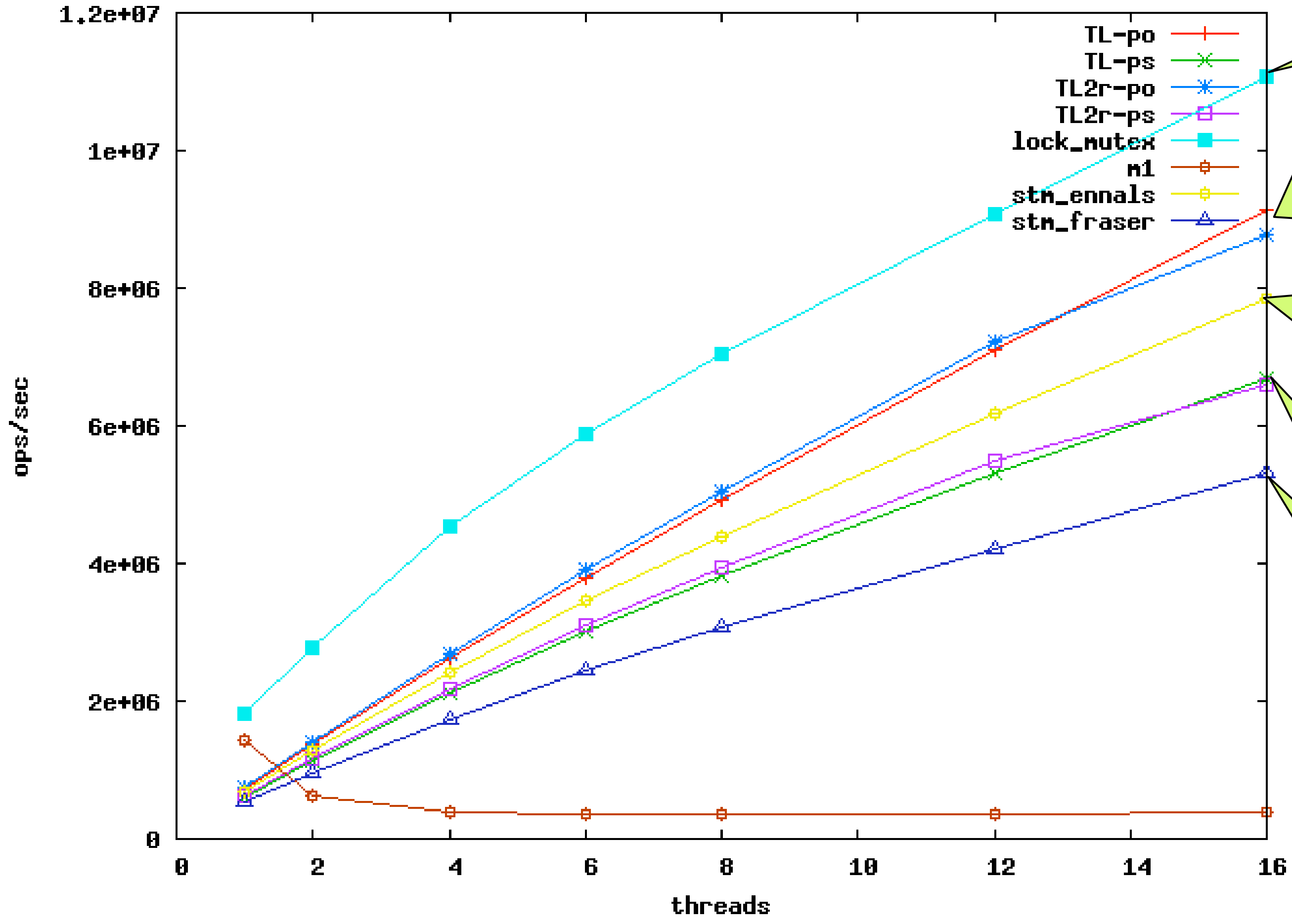
- On sys-on-chip like Sun T2000™ Niagara: almost no contention, just CAS and be happy
- On others: add TID to VClock, if VClock has changed since last write can use new value +TID. Reduces contention by a factor of N.
- Future: Coherent Hardware VClock that guarantees unique tick per access.

Performance Benchmarks

- Mechanically Transformed Sequential Red-Black Tree using TL2
- Compare to STMs and hand-crafted fine-grained Red-Black implementation
- On a 16–way Sun Fire™ running Solaris™ 10

Uncontended Large Red-Black Tree

5% Delete 5% Update 90% Lookup



Hand-crafted

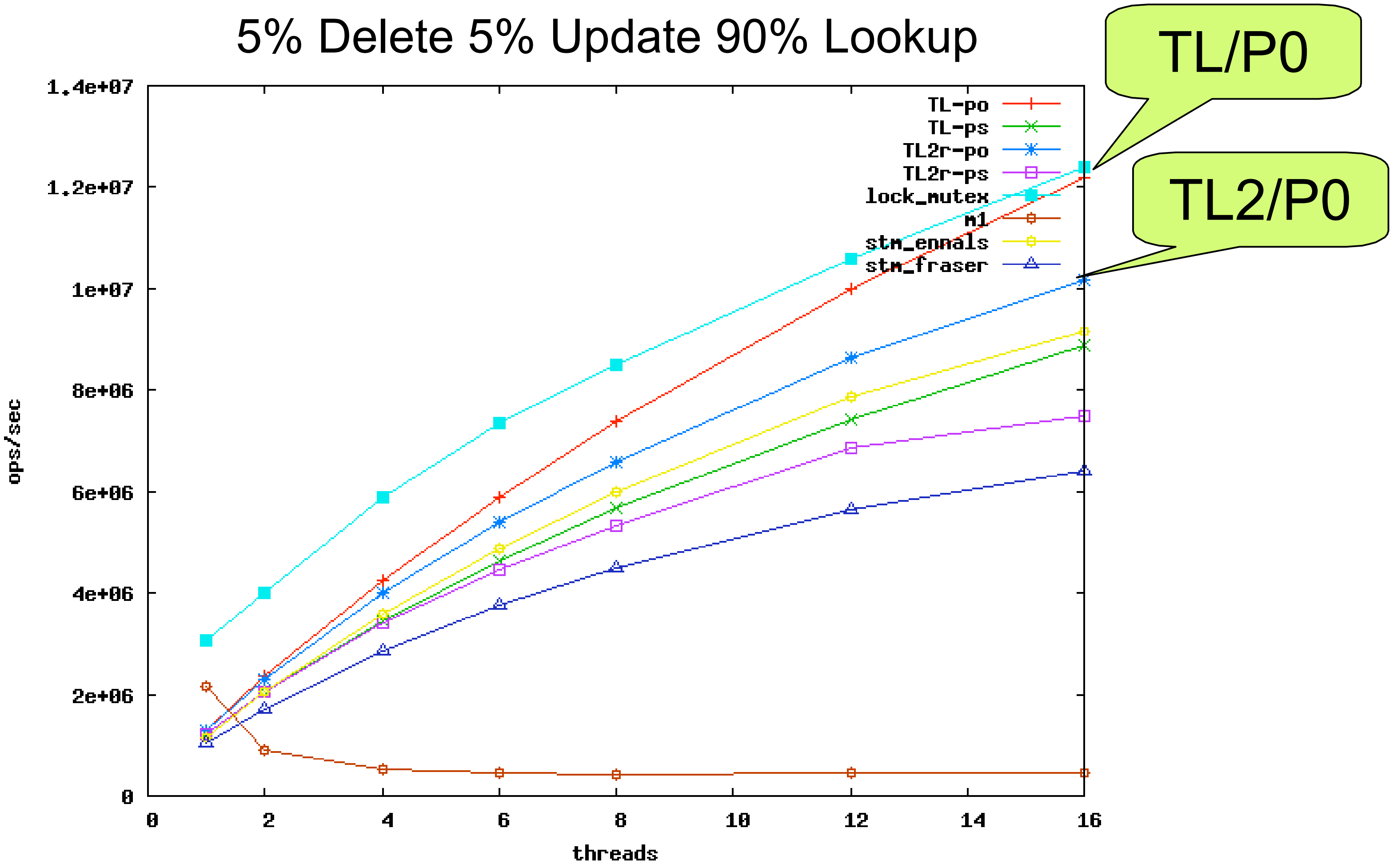
TL/PO
TL2/P0

Ennals

TL/PS
TL2/PS

Farser
Harris
Lock-free

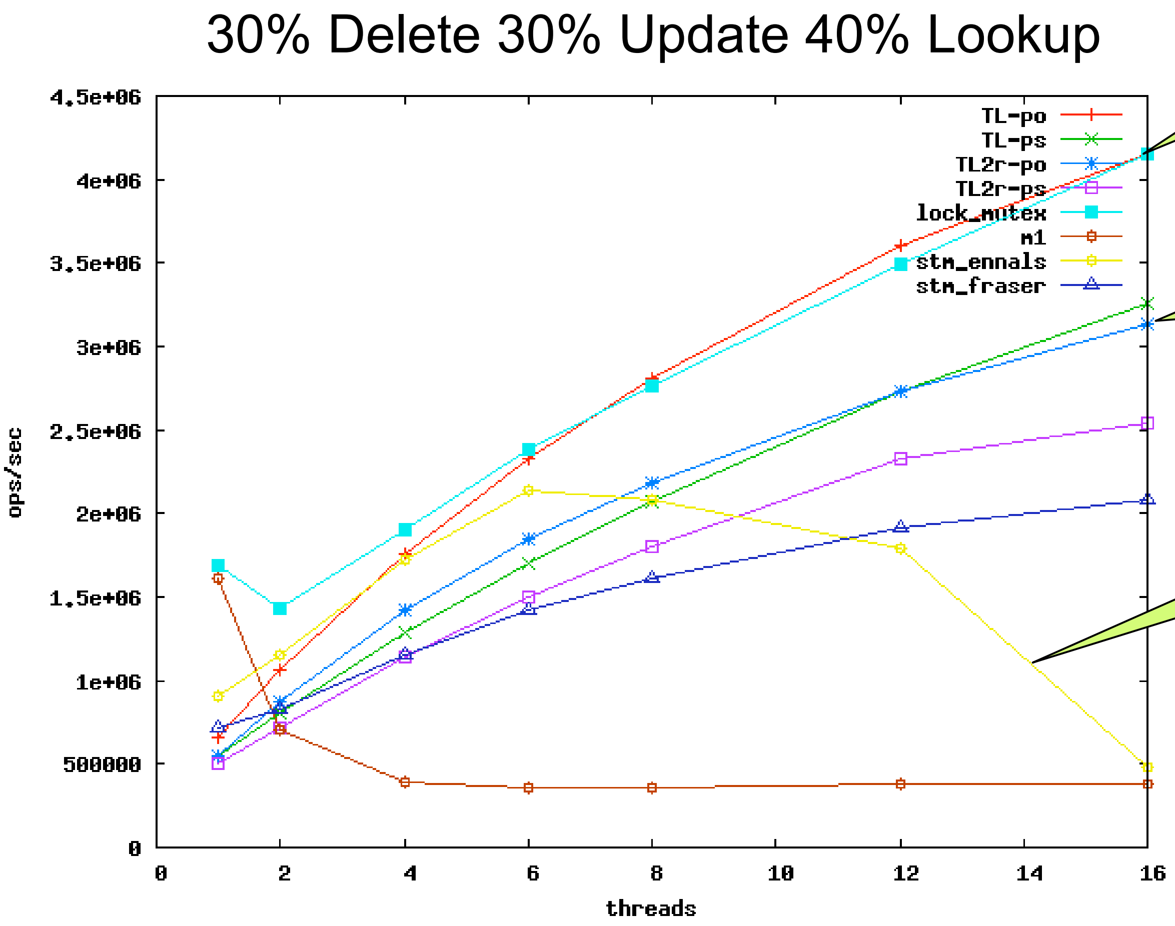
Uncontended Small RB-Tree



TL/P0

TL2/P0

Contended Small RB-Tree

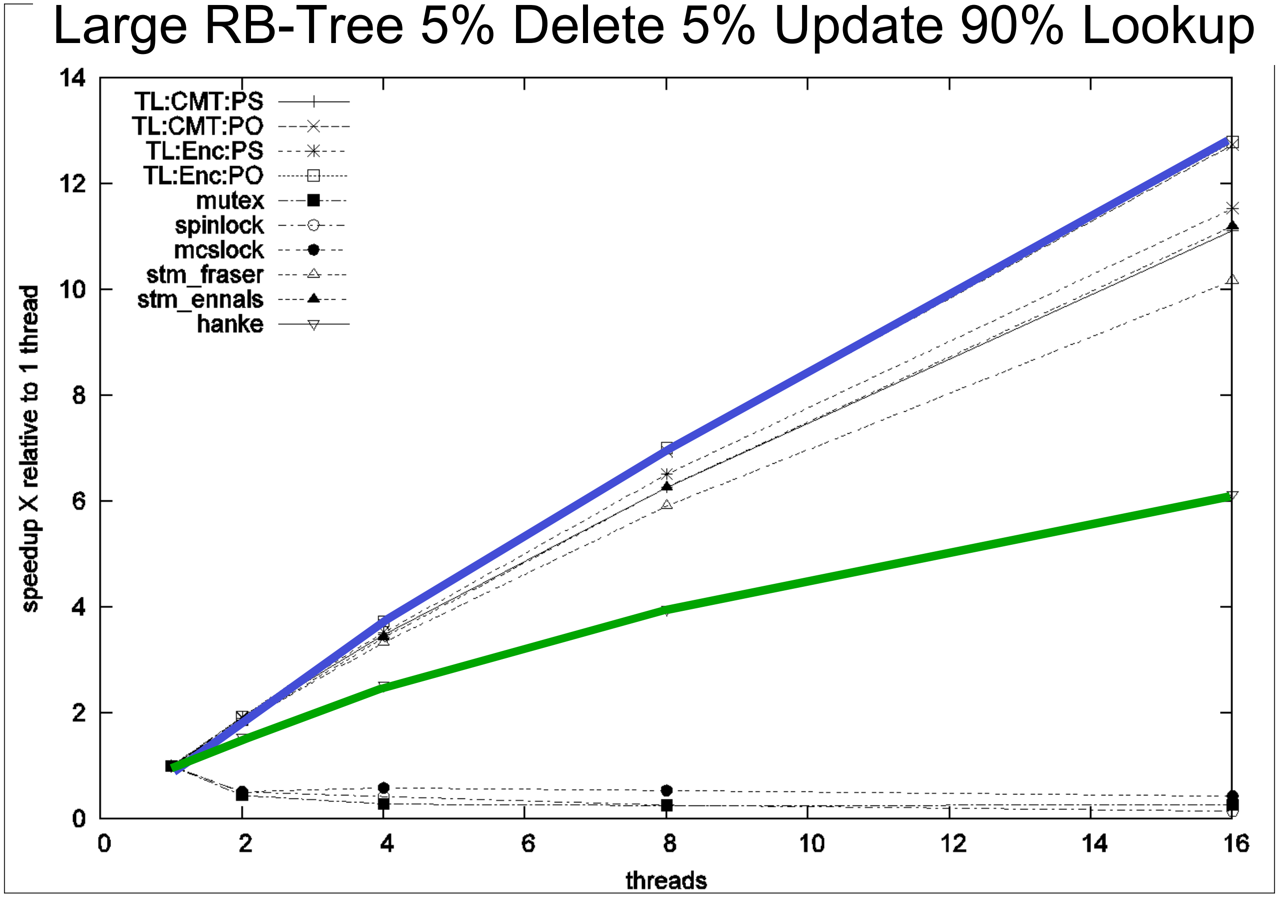


TL/P0

TL2/P0

Ennals

Speedup: Normalized Throughput



Large RB-Tree 5% Delete 5% Update 90% Lookup

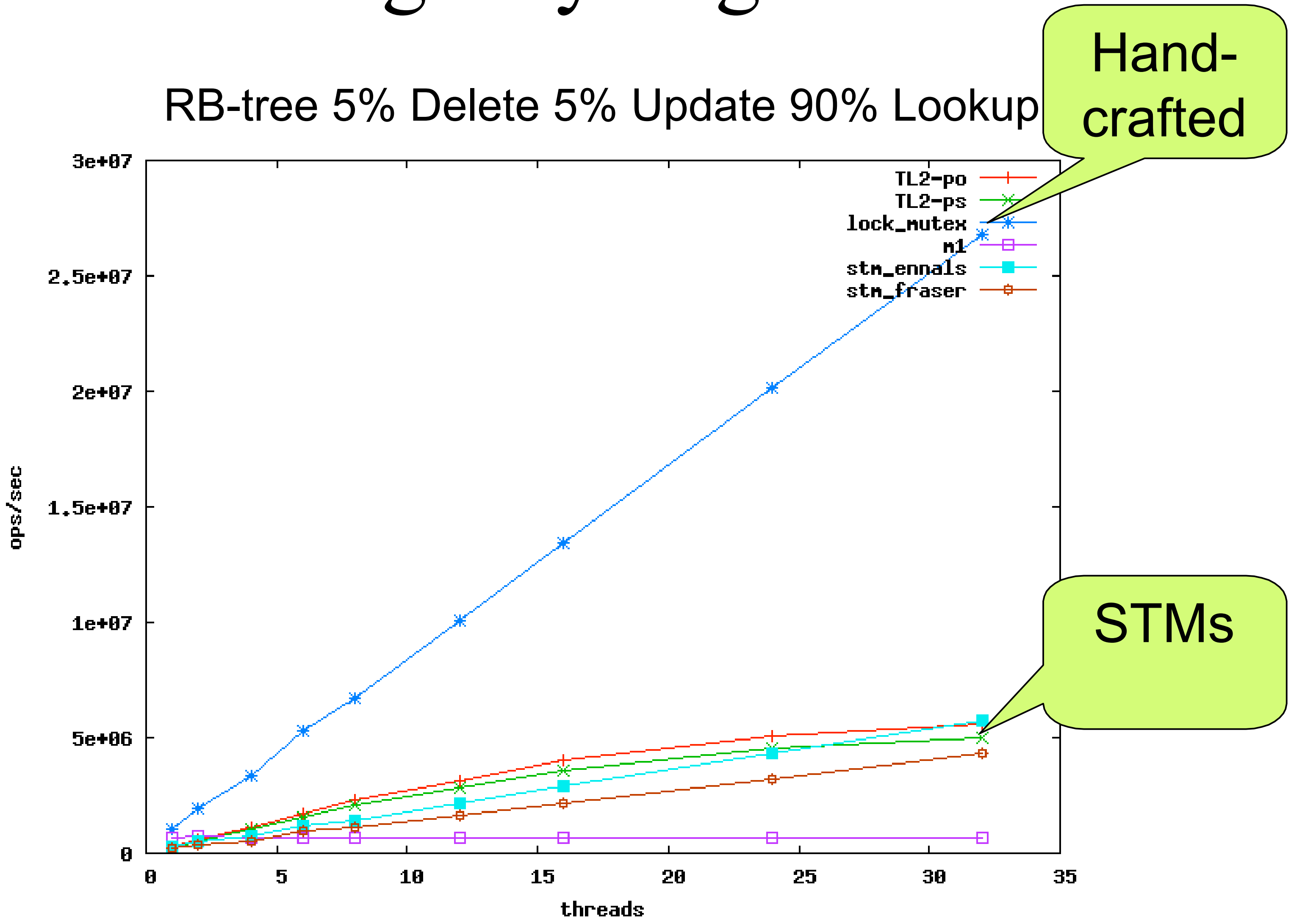
TL/PO

Hand-Crafted

Overhead Overhead Overhead

- STM scalability is as good if not better than hand-crafted, but overheads are much higher
- Overhead is the dominant performance factor – bodes well for HTM
- Read set and validation cost (not locking cost) dominates performance

On Sun T2000™ (Niagara): maybe a long way to go…



Hand-crafted

STMs

Conclusions

- COM time locking, implemented efficiently, has clear advantages over ENC order locking:
  - No meltdown under contention
  - Seamless operation with malloc/free
- VCounter can guarantee safety so we
  - don’t need to embed repeated validation in user code

What Next?

- Further improve performance
- TL2 library available shortly
- Mechanical code transformation tool…
- Cut read-set and validation overhead, maybe with hardware support?
- Add hardware VClock to Sys-on-chip.

Thank You