



Return of the Read-Write Lock

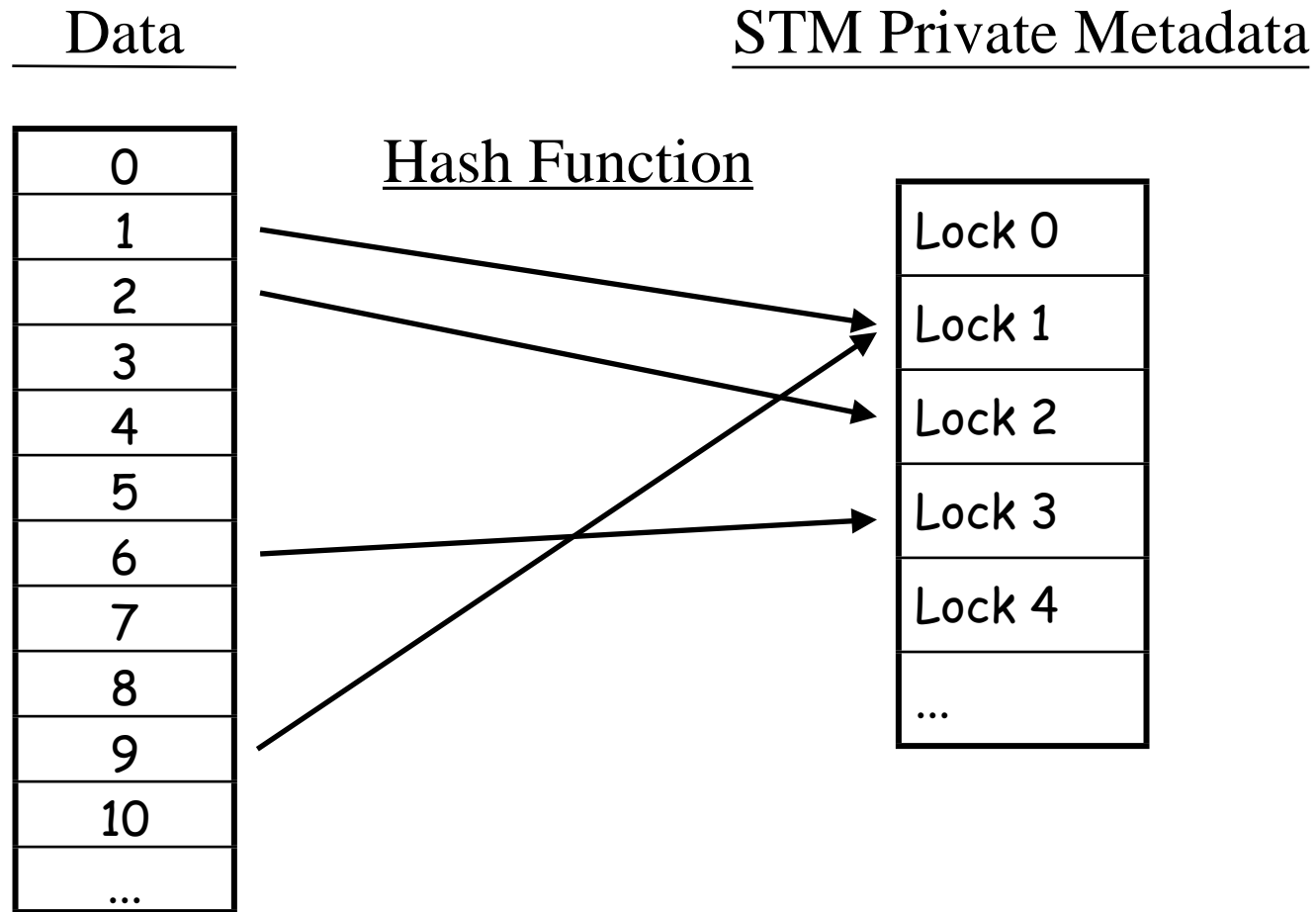
Dave Dice and Nir Shavit

» SunLabs SSRG

Transact 2009



Background: Stripe-based STMs



Pyrrhic Scaling Victories

- STMs achieve near ideal scaling ...
- But overhead is so high ...
- Only profitable in a thin wedge of the performance envelope
 - exotic and contrived microbenchmarks
- Long path lengths + atomics = latency
- Balance
 - scaling : still need to beat crude locking
 - drive down latency

Naive TLRW

- Lock record contains a read-write lock
- Acquire R access on 1st txl read to a stripe
- Acquire W access on 1st txl write to a stripe
- Update in-place with undo log
- Release RW locks at commit-time
- Provides implicit privatization - unlike TL2
 - useful for simple lock replacement
- Sole source of abort is deadlock avoidance
- Low abort rate compared to TL2
- Progress properties

Heresy !

- Common folk wisdom (myth)
 - Readers shouldn't write (shared metadata)
 - You'll burn in coherence traffic hell
 - Key idea behind TL2, Seqlocks, RCU
 - Avoid CAS + avoid ST to shared lines
- doesn't hold on all platforms
 - Nehalem and single-die Niagara **can** tolerate high degrees of write sharing
- Really 2 issues: CAS and write coherency

CAS Performance Impediments

- Invalidates line from L1\$ on Niagara
 - To be fixed in future processors
- Retries from concurrent interference
- Longer path because of retry loop
- High local latency
 - X-bar interface outside core
- Aside: Nehalem has fast CAS; fast MFENCE; and low-latency coherence.

TLRW-ByteLock : Lock Record

- Writer ID
- Visible readers :
 - Reader count for **unslotted** threads
 - CAS to increment and decrement
 - Reader array for **slotted** threads
 - Array of atomically addressable bytes
 - 48 or 112 slots
 - At most 1 thread per slot at any one time
 - Arriving readers just ST instead of CAS

Traditional

TLRW-ByteLock : read path

- Already writer ... fetch & done
- Slotted:
 - already reader ... fetch & done
 - Fast read-after-read test : fetch slot byte
 - Exploit locality in application
 - ST into slot; MEMBAR
- Unslotted
 - Check local read-set - already reader?
 - CAS to increment Reader counter
- Check writer : RW conflict detection
- Add to local read-set
- Fetch Location

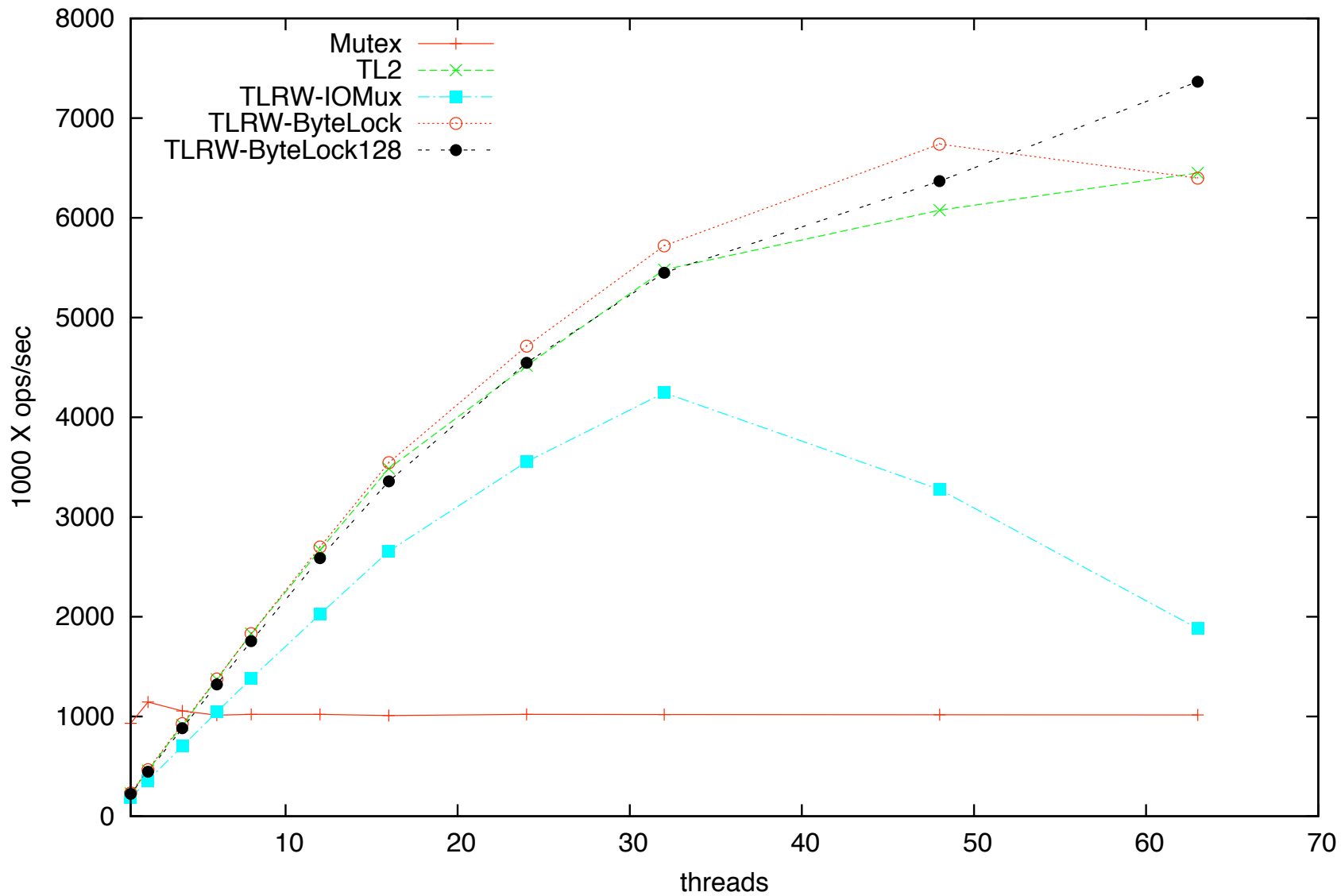
TLRW-ByteLock : write path

- Already writer ... save contents; store; done
- Acquire exclusive access with CAS
 - Resolve WW conflicts
- Wait for readers to drain : broadword loads
- Write in place : undo log write-set
 - No need for lookaside in read path
- Scalability trade off:
 - write lock hold-time
 - reader path length

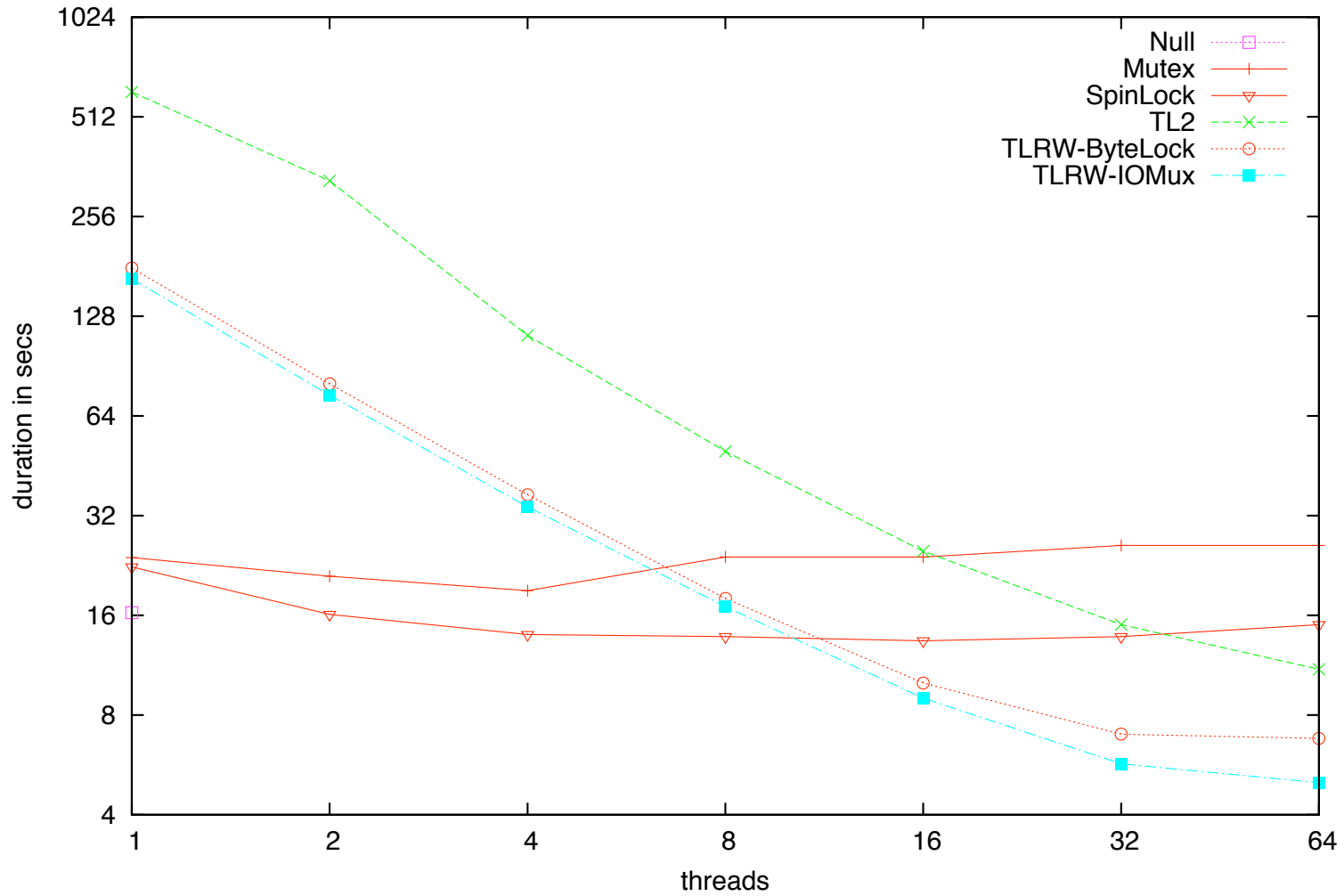
TLRW-ByteLock

- Polite writers - pessimistic
- Visible readers
- Avoid CAS for many reader threads
- Very short common-case paths
- Trade off: path length v. ideal scaling
- Requires low-latency write sharing

Niagara-2 D25U25 20000



Niagara-2 MSF



Irrevocable Transactions

- Nearly free by-product of TLRW
- At any one time at most one thread can be in irrevocable mode
- Deadlock resolution: irrevocable thread always wins
- In practice: unbounded spin for R or W access
- Allows IO in txns
- Automatically switch to irrevocable after N aborts

[Editorial Statement]

- Encourage programmers to minimize mutable shared state
- Clojure, for example
 - Viable STM : txns are small and short
 - Only access txl variables within txns
 - Enforced by type system
 - No privatization problem
- Scala ...

Thank You

Backup Slides

Miscellany

- CAS Invalidates
[http://blogs.sun.com/dave/entry/cas and cache trivia invalidate](http://blogs.sun.com/dave/entry/cas_and_cache_trivia_invalidate)
- Readers shouldn't write
[http://blogs.sun.com/dave/entry/seqlocks in java](http://blogs.sun.com/dave/entry/seqlocks_in_java)

Memory Model

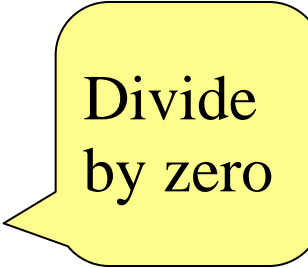
- Single global lock Semantics in a Weakly Atomic STM
 - Vijay Menon et al. (Intel now Google)
 - Transact 2008
 - Useful taxonomy
- TLRW* provides ALA or ELA or DLA

Privatization Problem

- Privatization
 - Access location txly
 - Mark as no longer txl - Escape
 - Access location non-txly
- Problem (in many STMs like TL2)
 - Mixed accesses → unexpected behavior
 - Admit outcomes not possible under locks
 - Challenge for simple lock replacement
 - Explicit and Implicit
- Not a concern for ROCK Hardware TM

TL2 Privatization Pathology

Initially: X=10 and XShared=True

Thread #1	Thread #2
<pre>// Privatize X // Fly-by writer atomic { XShared=False;} X=0;</pre>	<pre>atomic { if XShared : tmp=tmp/X; }</pre> 

Economics

- *CAS*
 - Local latency
 - *MEMBAR* equivalent
 - Implemented at X-bar interface off core
 - Niagara: L1\$ eviction and subsequent miss
 - Retries
 - Loop path length
- *MEMBAR/MFence*
- Relative latency of *MEMBAR* vs *CAS*
- Memory coherency traffic
 - Latency and bandwidth
 - Read-Read sharing is only fast form
 - RW, WW require write-invalidation
- Path length on *CMT* is important

TLRW

- Deadlock avoidance and contention management conflated
- Accurate deadlock detection not profitable
- Implementation admits false+ aborts
- Bounded spins to acquire R or W access
- Otherwise abort
- Post abort randomized back-off : TBEB
- Irrevocable thread uses unbounded spins
- Preordained winner

TLRW : path length

- No lookaside into write-set by reader
- But longer write-lock hold times
- Trade scaling for 1T latency
- Fast test for read-after-read
 - Simply fetch slot byte
 - Leverages spatial + temporal locality

TLRW

- Read-after-read
 - Assumes temporal and spatial locality
 - Fast membership test in TLRW
 - Check the slot
- Predicated on low-latency write coherency
- 4:1 Read:Write ratio is typical
- More pessimistic than TL2
- Polite writers; visible readers

BREAK