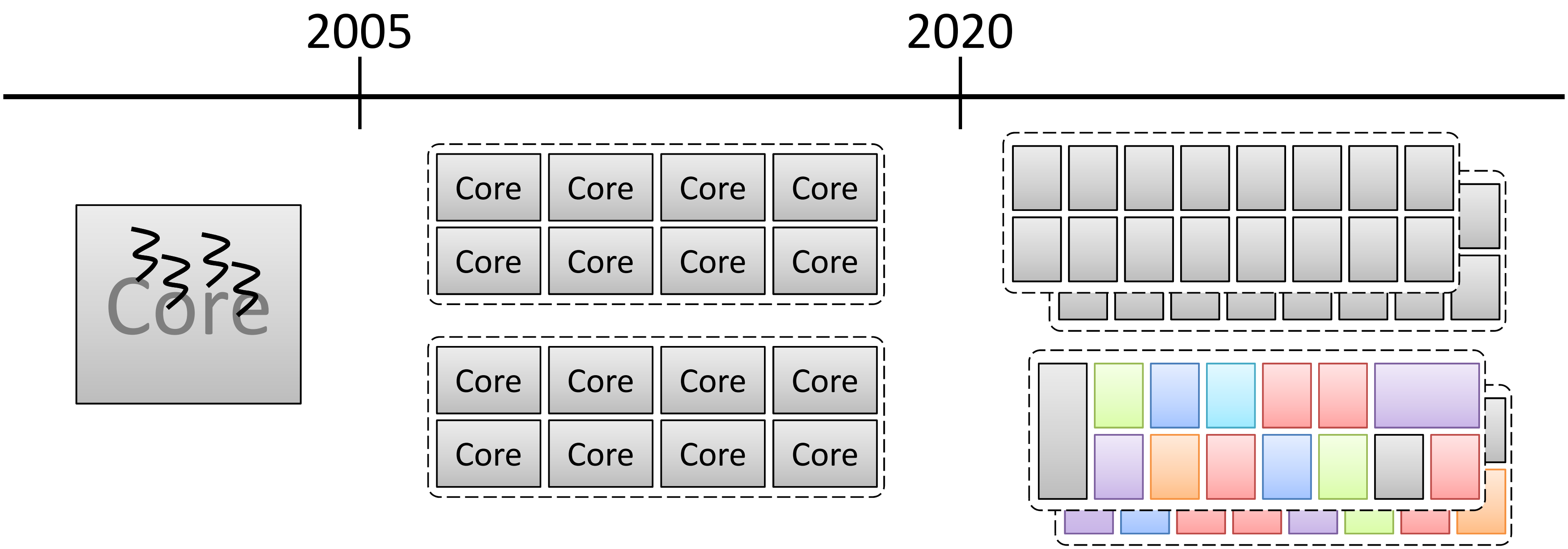
Toward Scalable Transaction Processing

Evolution of Shore-MT

Anastasia Ailamaki (EPFL)
Ryan Johnson (University of Toronto)
Ippokratis Pandis (IBM Research – Almaden)
Pınar Tözün (EPFL)

hardware parallelism: a fact of life



pipelining
ILP
multithreading

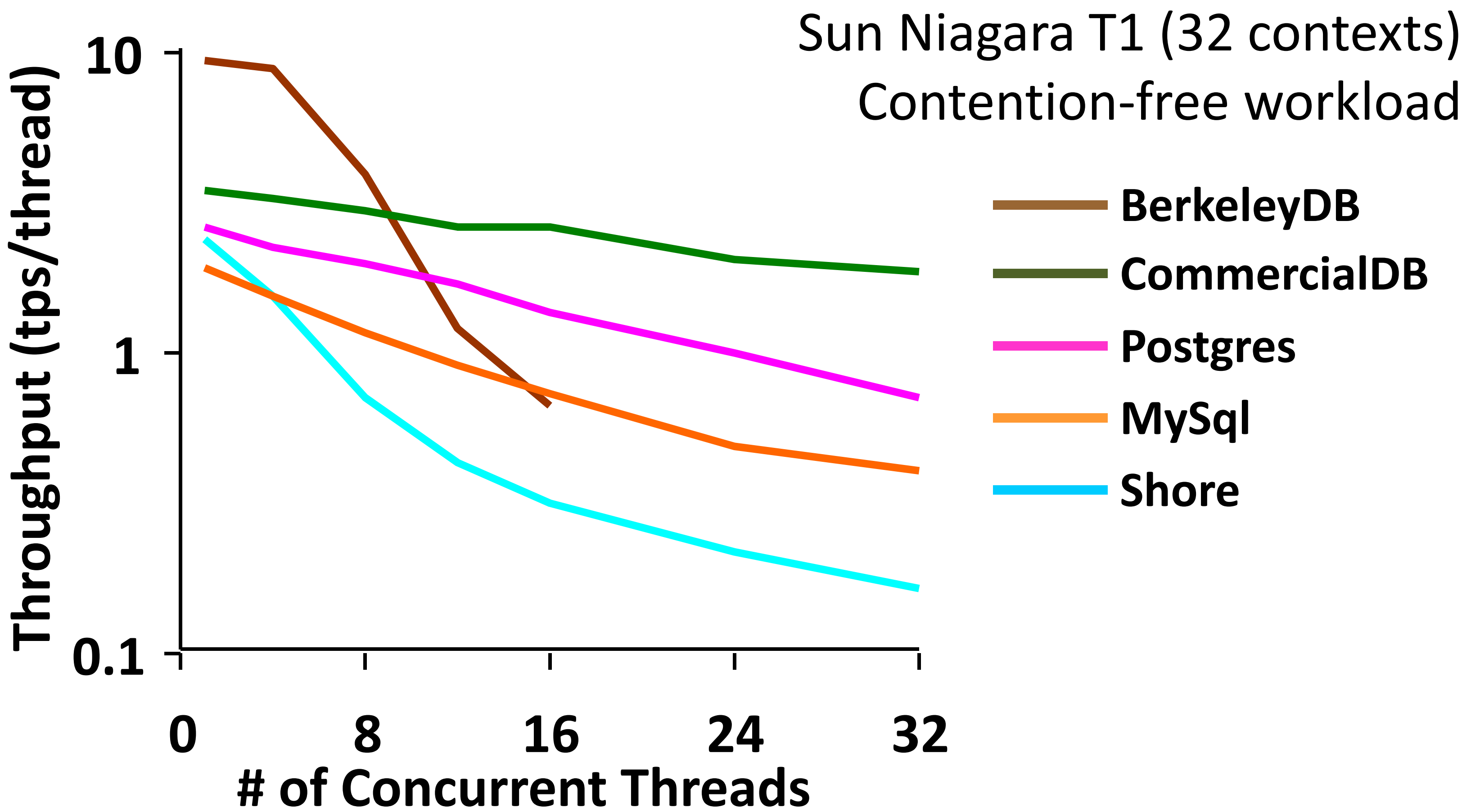
multisocket multicores
(CMP)

heterogeneous CMP

“performance” = scalability

software parallelism doesn't just "happen"

[EDBT2009]

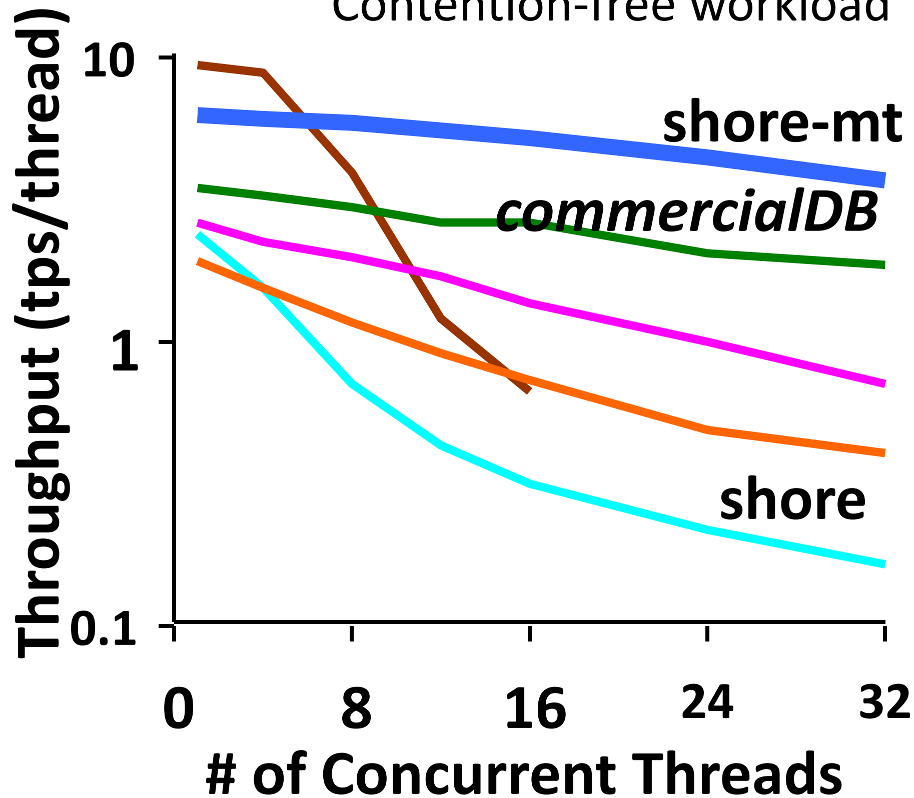


best scalability 30% of ideal

Shore-MT: an answer to multicore

- Multithreaded version of SHORE
- State-of-the-art DBMS features
- Two-phase row-level locking
- ARIES-style logging/recovery
  - ARIES-KVL [VLDB1990]
  - ARIES-IM [SIGMOD1992]
- Similar at instruction-level with commercial DBMSs

Sun Niagara T1
Contention-free workload



test-bed for database research

infrastructure for micro-architectural analysis

Shore-MT in the wild

- Goetz Graefe (HP Labs)
  - Foster B+Trees [TODS2012]
  - Controlled lock violation [SIGMOD2013a]

- Alan Fekete (U. Sydney)
  - A Scalable Lock Manager for Multicores [SIGMOD2013b]
- Tom Wenisch (U. Michigan)
  - phase-change memory [PVLDB2014]

- Steven Swanson (UCSD)
  - non-volatile memories

- Andreas Moshovos (U. Toronto)
  - storage systems

- … many more

Shore-MT 7.0

- Improved portability

OS	*CPU*	*Compiler*

- Reduced complexity in adding new workloads

- Bug fixes

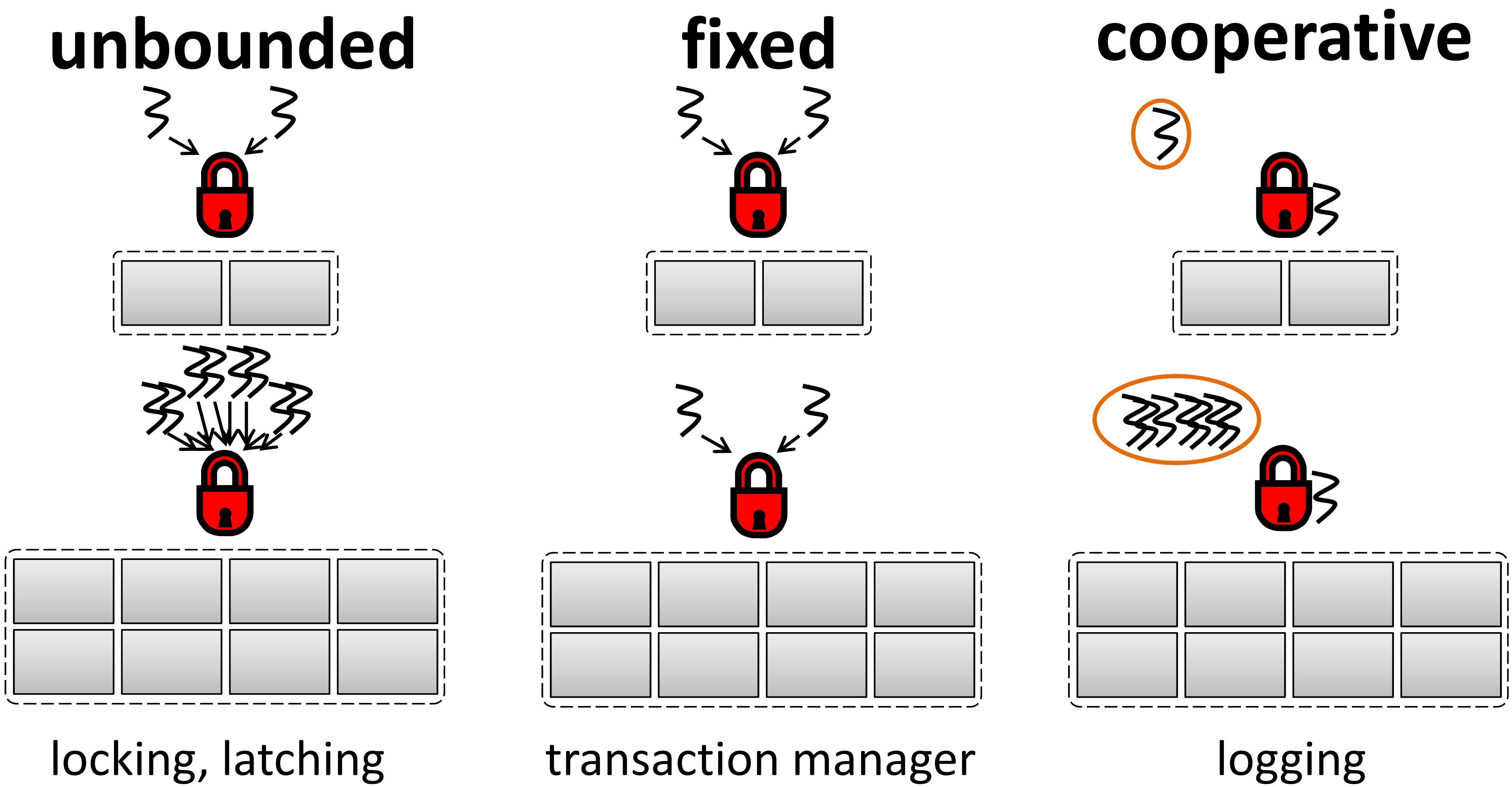
http://diaswww.epfl.ch/shore-mt

scaling-up OLTP on multicores

- Extreme physical partitioning
  - *H-Store/VoltDB* [VLDB2007]
  - *HyPer* [ICDE2011]
- Logical & Physiological partitioning
  - *Oracle RAC* [VLDB2001]
  - *DORA/PLP on Shore-MT* [PVLDB2010b,PVLDB2011]
- Lock-free algorithms & MVCC
  - *TokuDB* [SPAA2005a]
  - *MemSQL*
  - *Hekaton* [SIGMOD2013]

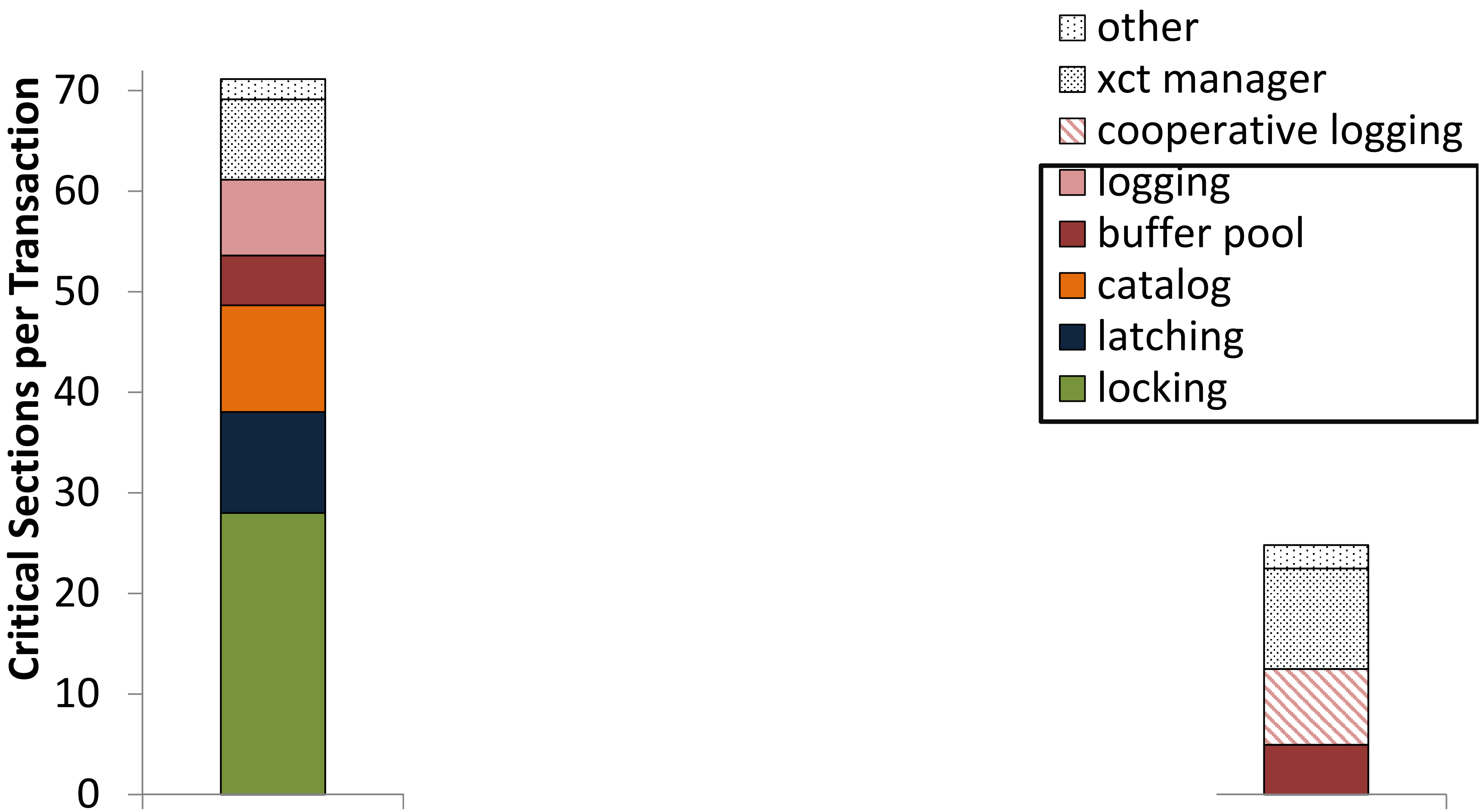
not all interference is bad

[VLDBJ2013]



unbounded → fixed / cooperative

communication in Shore-MT



outline

- introduction *~ 20 min*
- part I: achieving scalability in Shore-MT *~ 1 h*
- part II: behind the scenes *~ 20 min*
- part III: hands-on *~ 20 min*

outline

- introduction *~ 20 min*
- part I: achieving scalability in Shore-MT *~ 1 h*
  - taking global communication out of locking
  - extracting parallelism in spite of a serial log
  - designing for better communication patterns
- part II: behind the scenes *~ 20 min*
- part III: hands-on *~ 20 min*

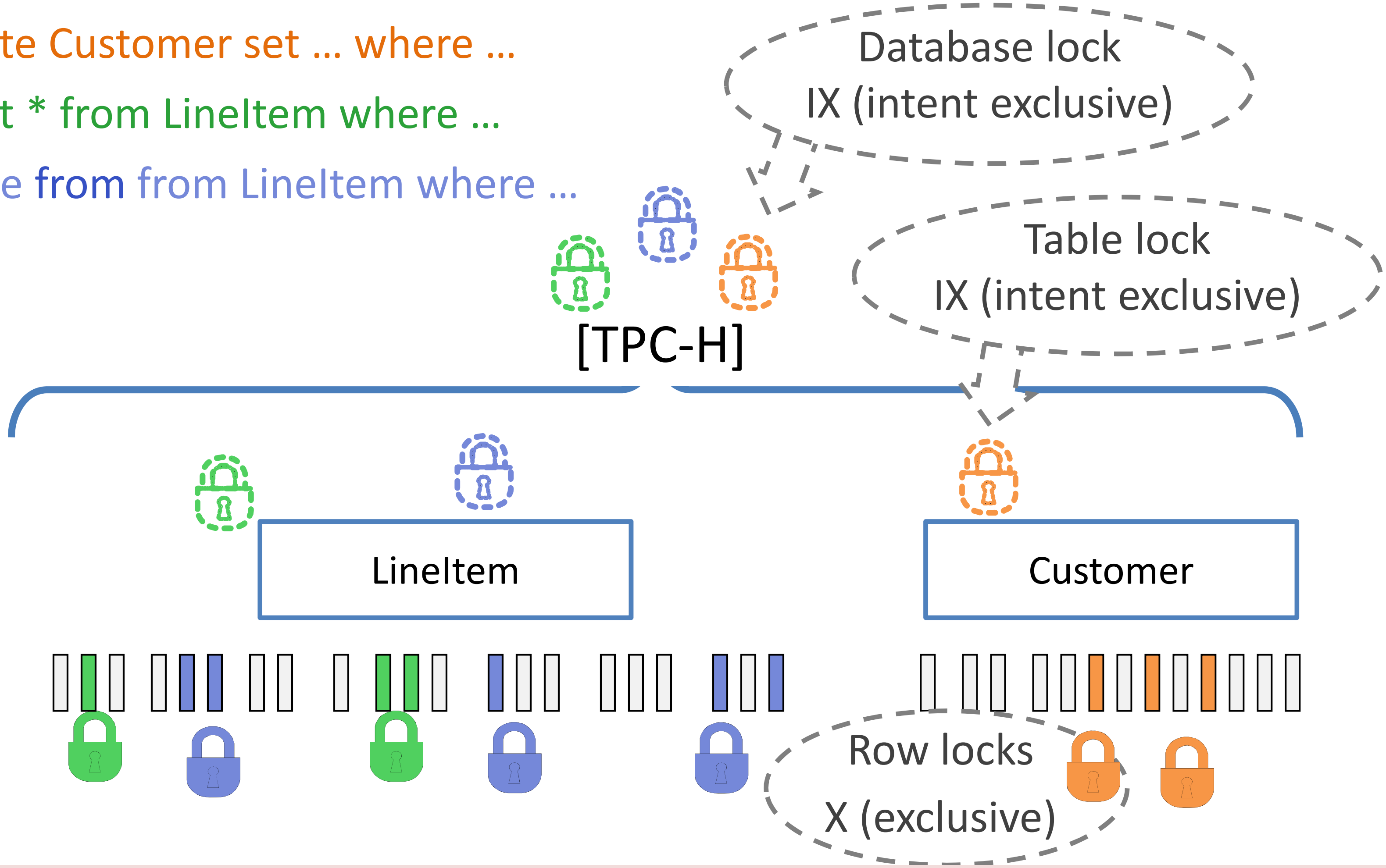
hierarchical locking is good… and bad

Good: concurrent access to distinct tuples

update Customer set … where …

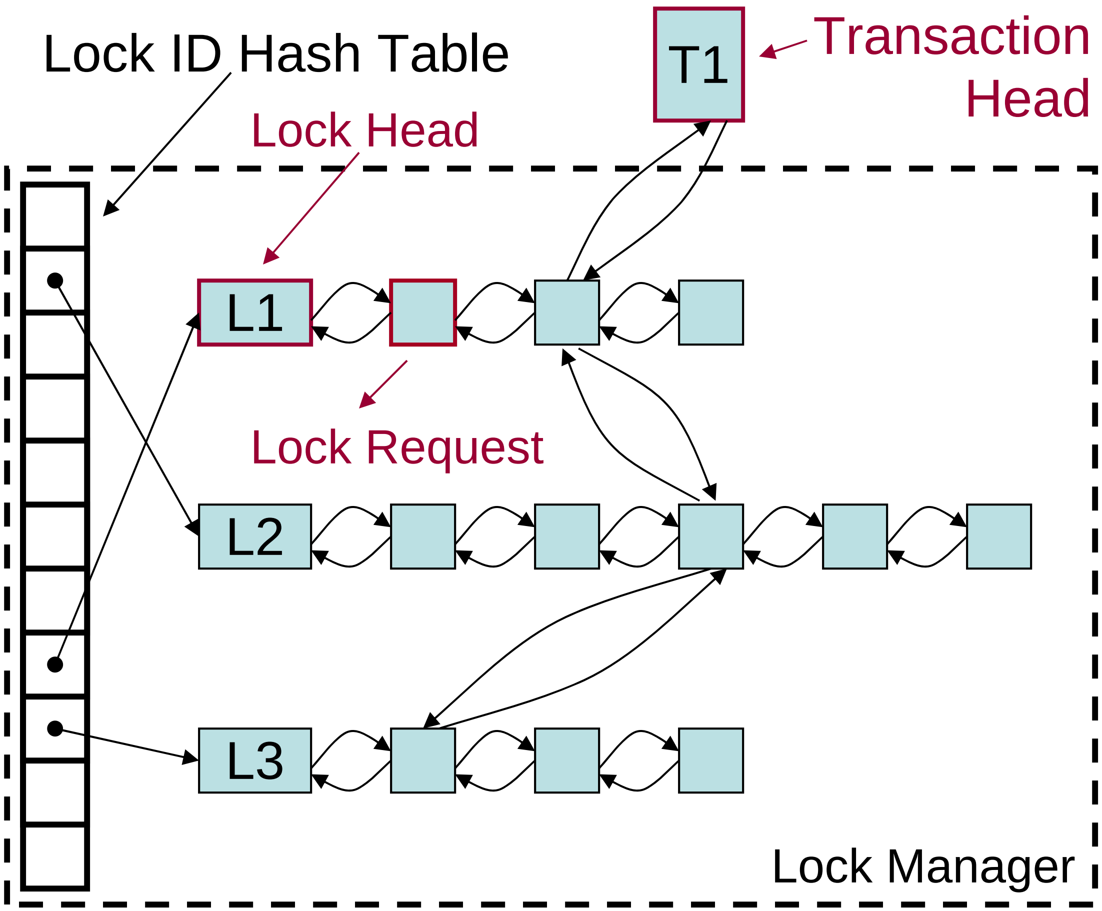
select * from LineItem where …

delete from from LineItem where …



bad: lock state update is complex and serial

inside the lock manager - acquire



Requirements

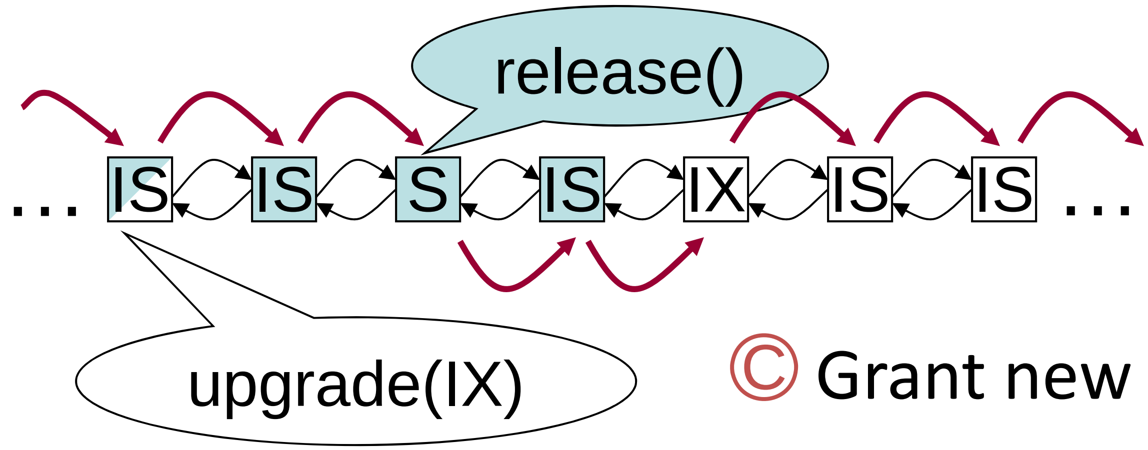
⇒ Find/create many locks in parallel

⇒ Each lock tracks many requests

⇒ Each transaction tracks many locks

inside the lock manager - release

Ⓐ Compute new lock mode (supremum)



Ⓒ Grant new requests

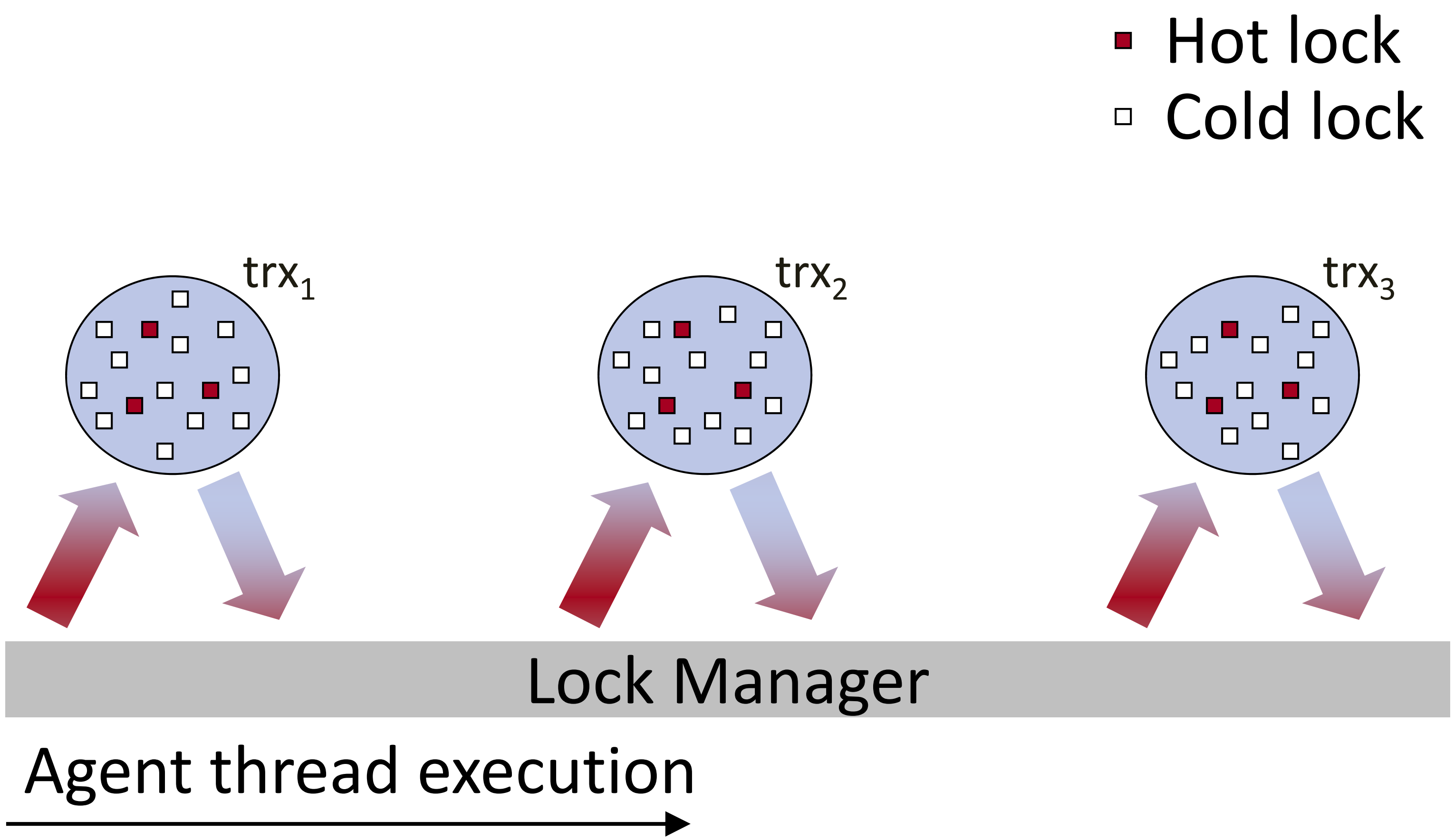
Ⓑ Process upgrades

Lock strengths

IS < IX < S

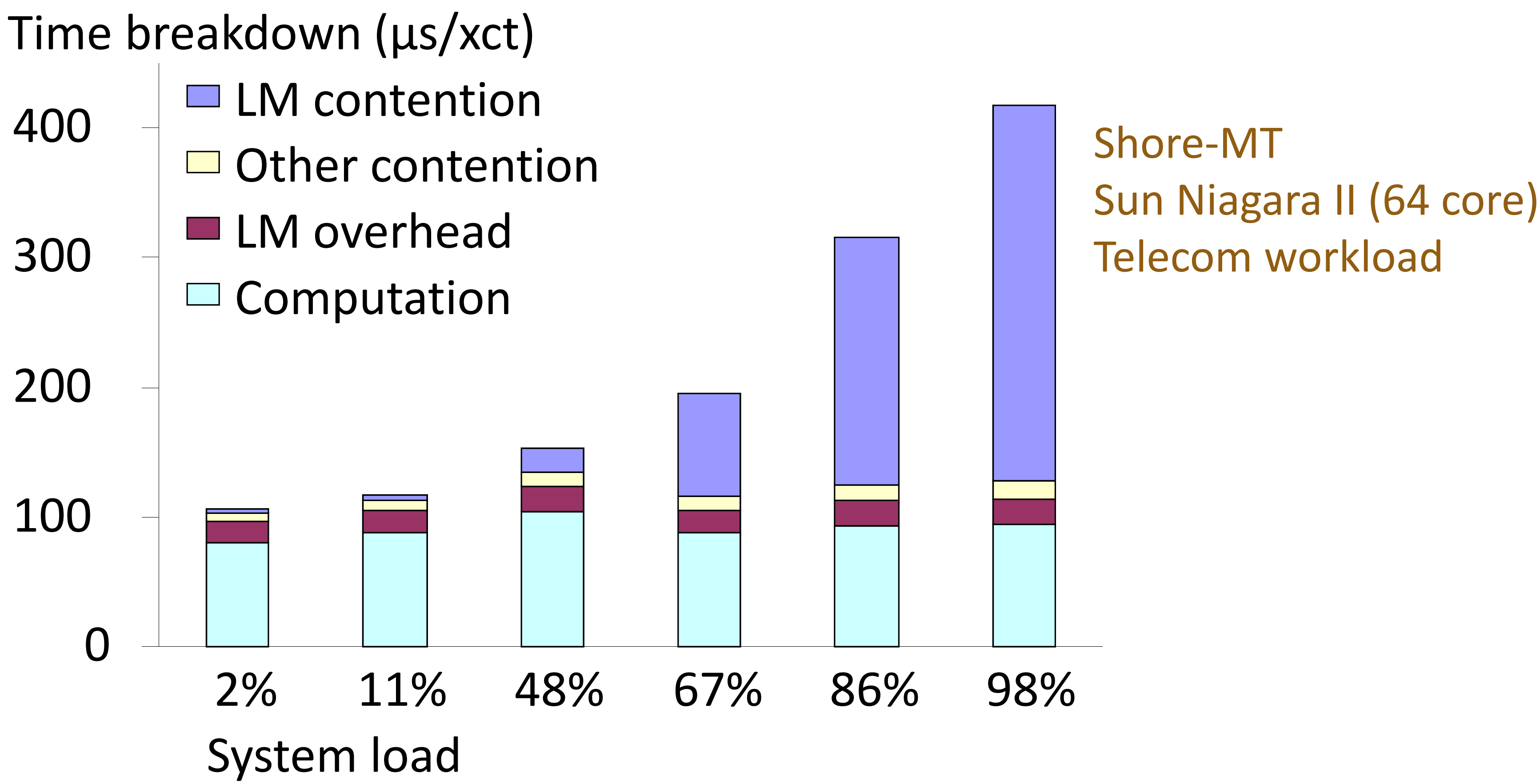
intent locks => long request chains

hot shared locks cause contention



release and request the same locks repeatedly

How much do hot locks hurt?

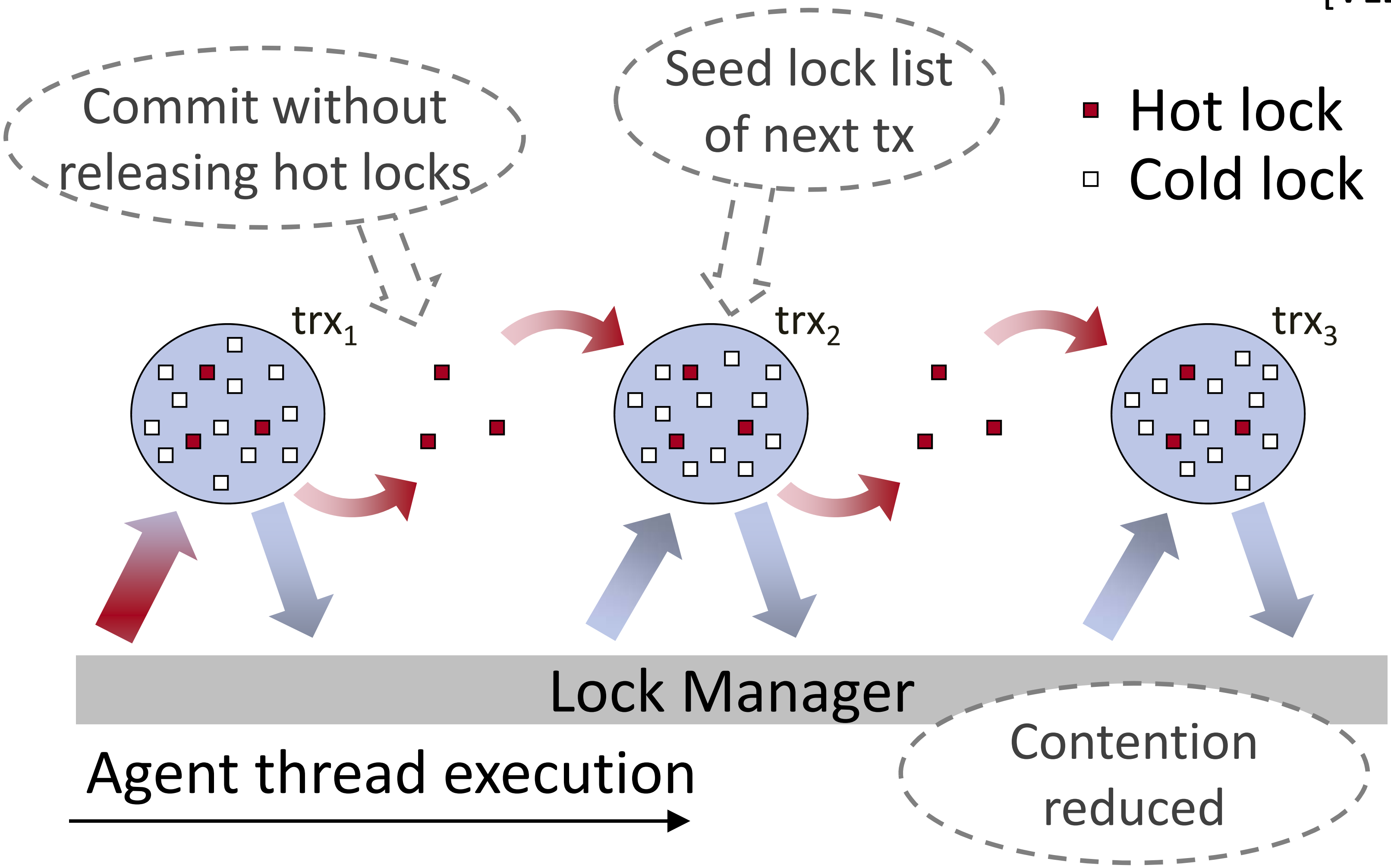


Answer: pretty bad (especially for short transactions)

even worse: these are share-mode locks!

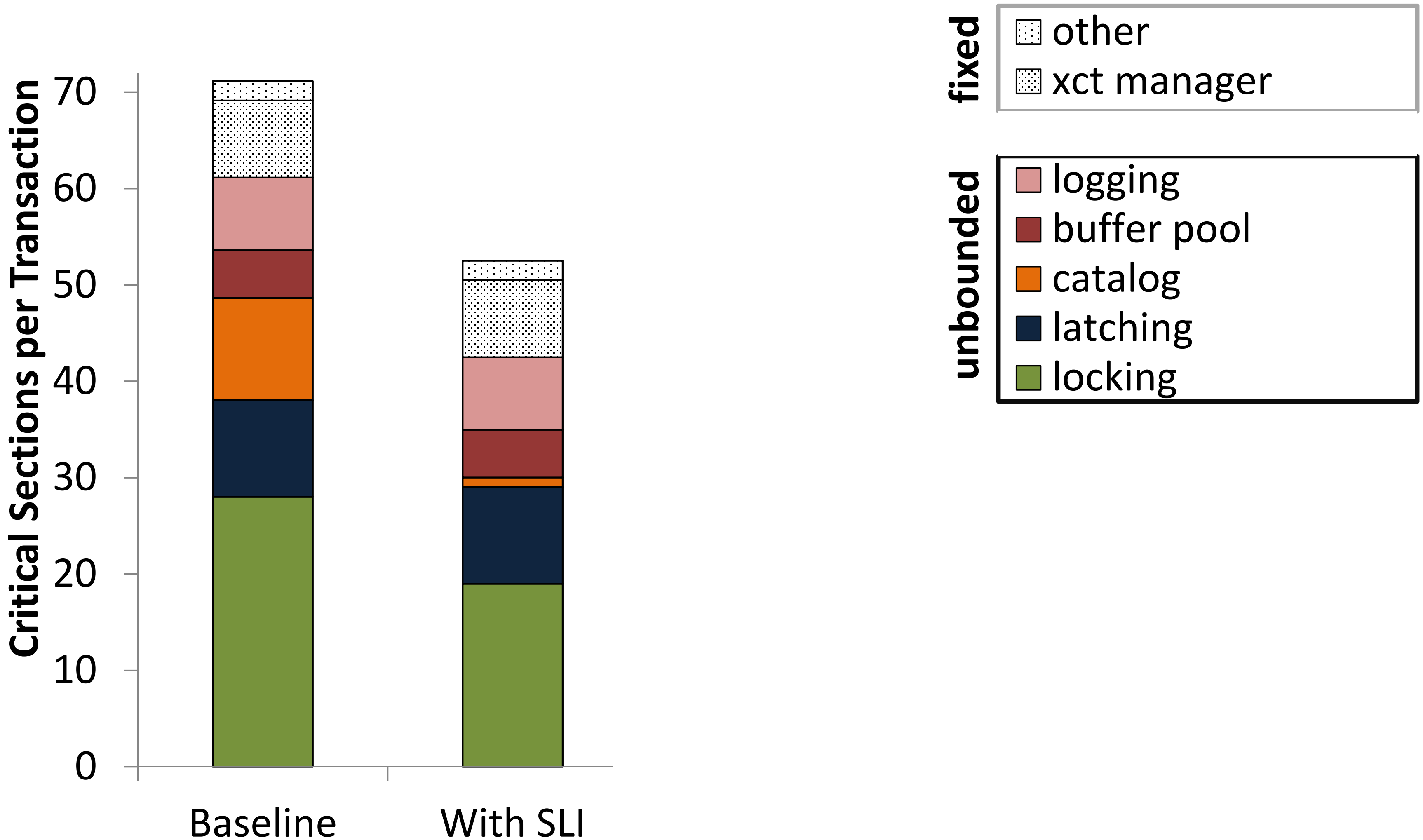
speculative lock inheritance

[VLDB2009]



small change; big performance impact

impact of SLI on communication



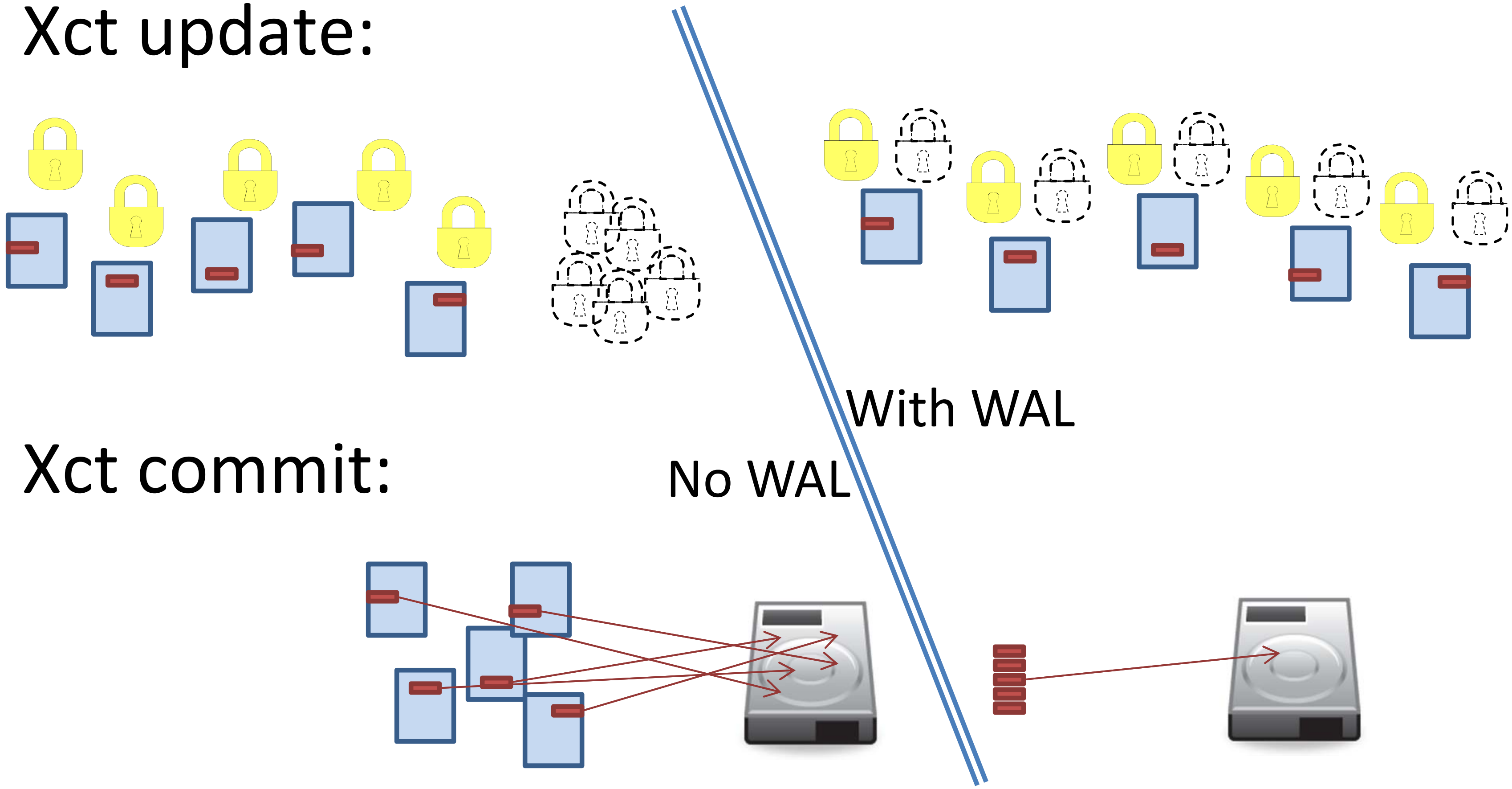
avoiding the unbounded communication

outline

- introduction *~ 20 min*
- part I: achieving scalability in Shore-MT *~ 1 h*
  - taking global communication out of locking
  - extracting parallelism in spite of a serial log
  - designing for better communication patterns
- part II: behind the scenes *~ 20 min*
- part III: hands-on *~ 20 min*

WAL: gatekeeper of the DBMS

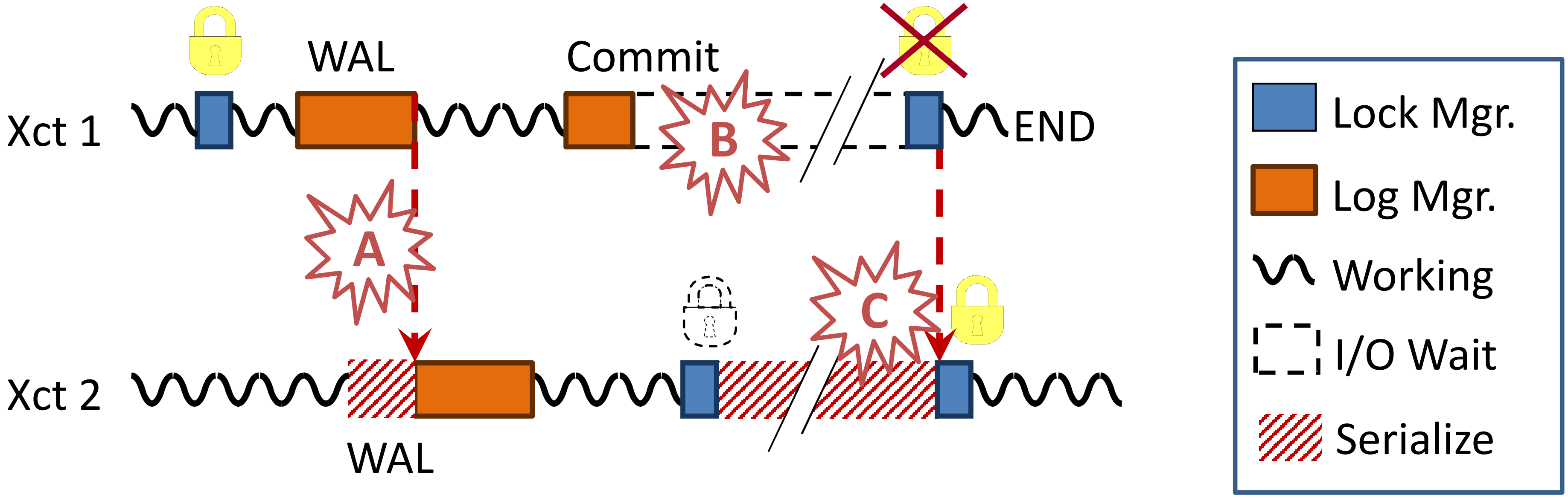
- Write ahead logging is a performance enabler
- Xct update:
- Xct commit:



but… logging is completely serial (by design!)

a day in the life of a serial log

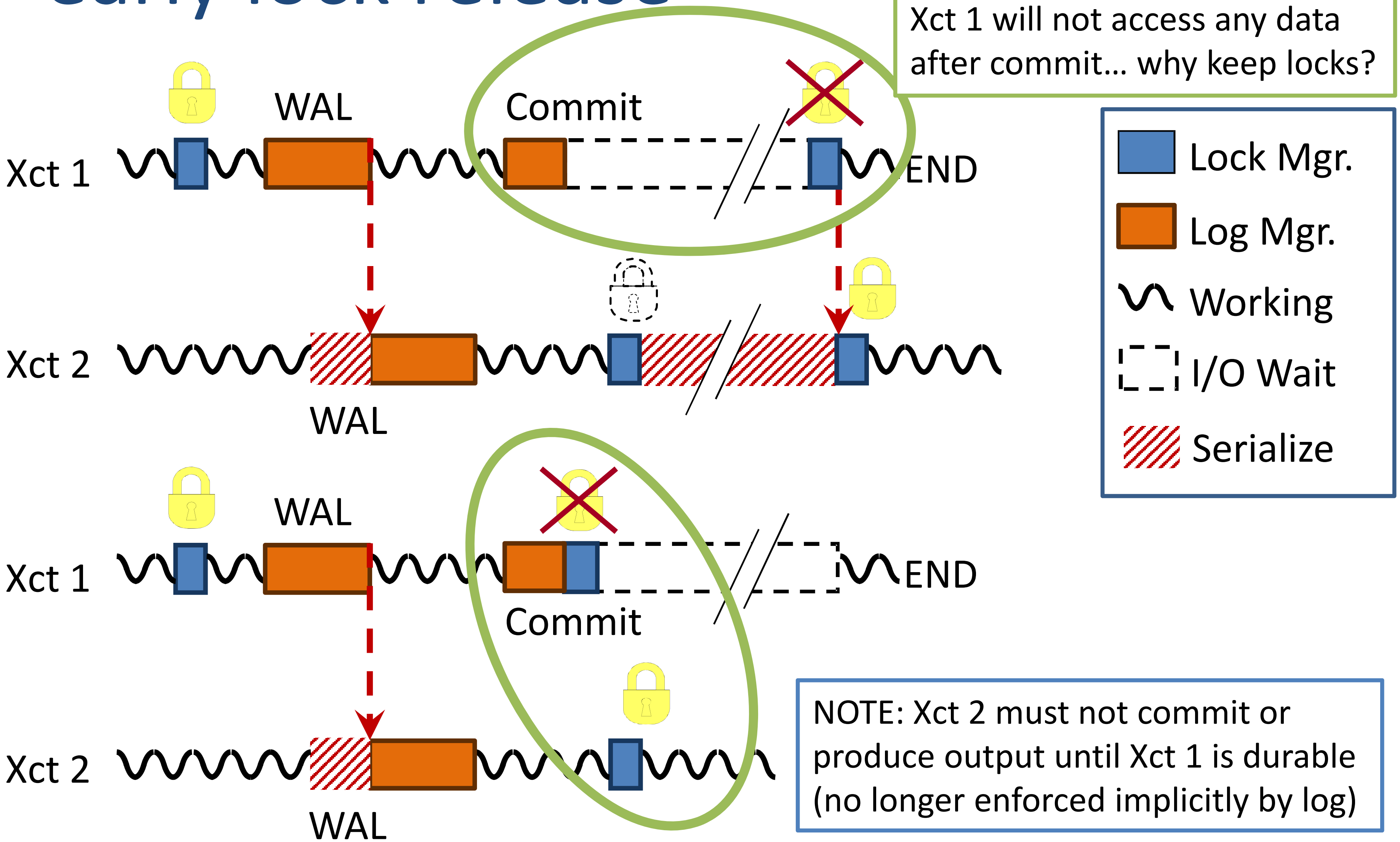
[PVLDB2010a]



- A: Serialize at the log head
- B: I/O delay to harden the commit record
- C: Serialize on incompatible lock

early lock release

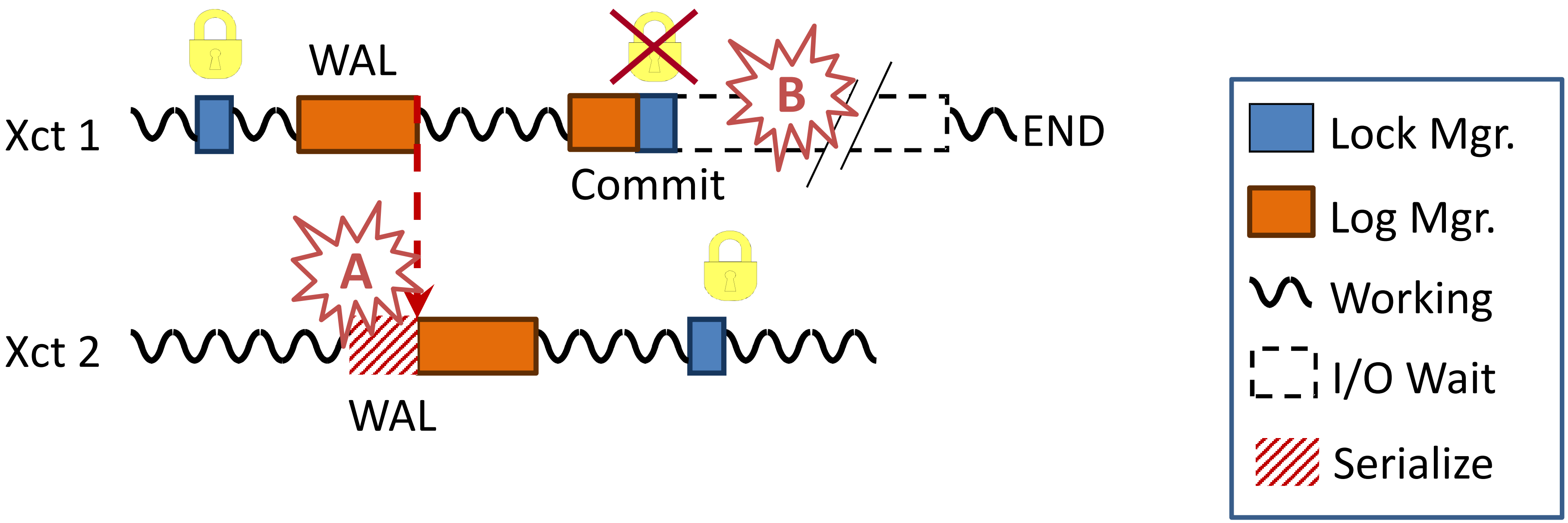
Xct 1 will not access any data after commit… why keep locks?



NOTE: Xct 2 must not commit or produce output until Xct 1 is durable (no longer enforced implicitly by log)

no overhead, eliminates lock amplification

a day in the life of a serial log

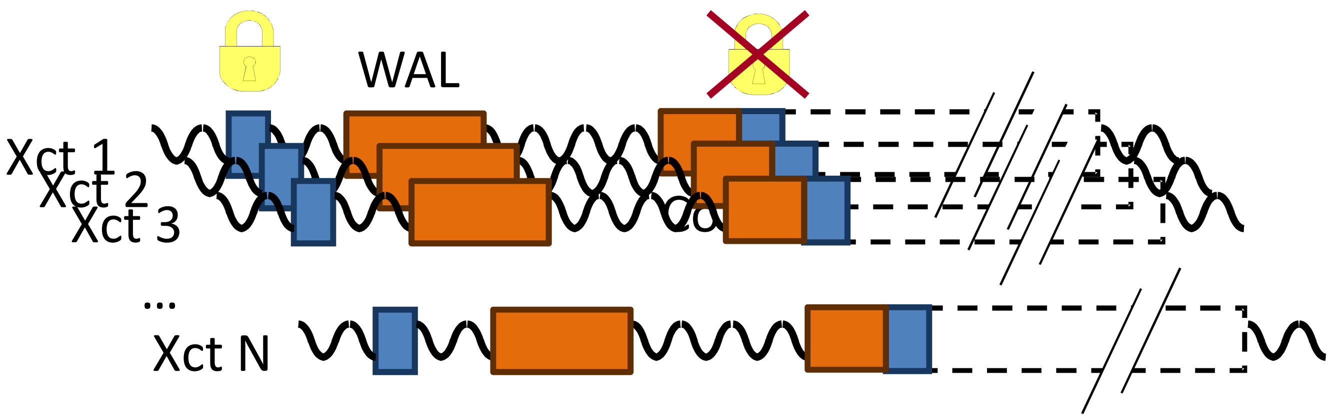


A Serialize at the log head

B I/O delay to harden the commit record

~~Serialize on incompatible lock~~

a day in the life of a serial log

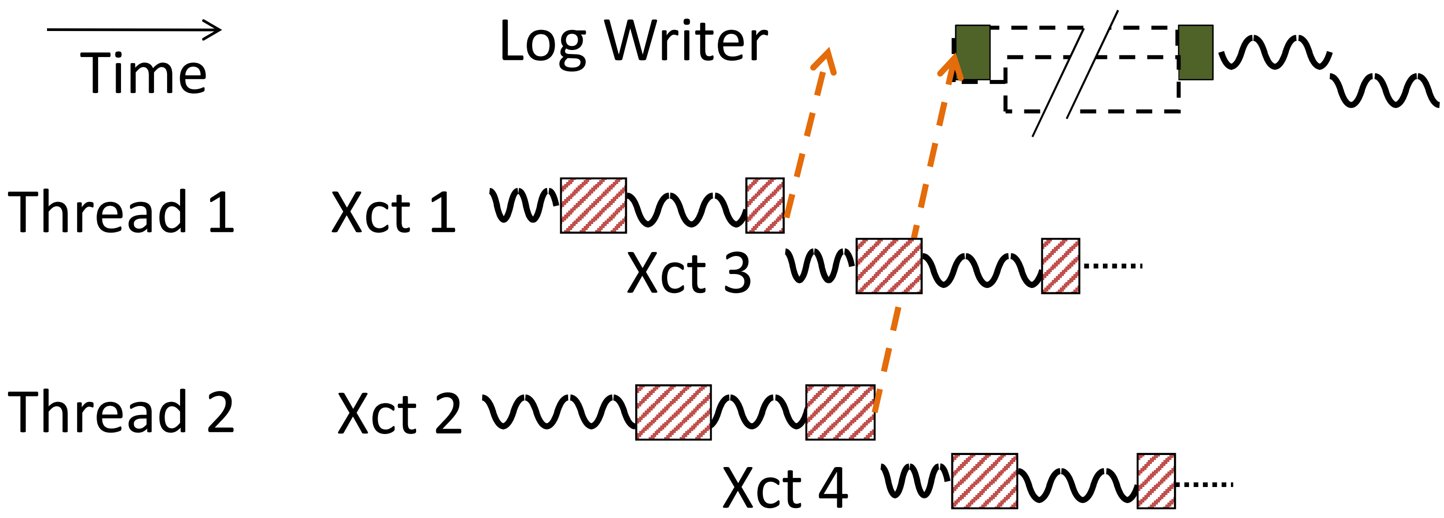


- Log commit => 1+ context switches per xct
  - Bad: each context switch wastes 8-16µs CPU time
  - Worse: OS can “only” handle ~100k switches/second
- Group commit doesn’t help
  - Block pending on completion signal (instead of on I/O)

let someone else process the completion!

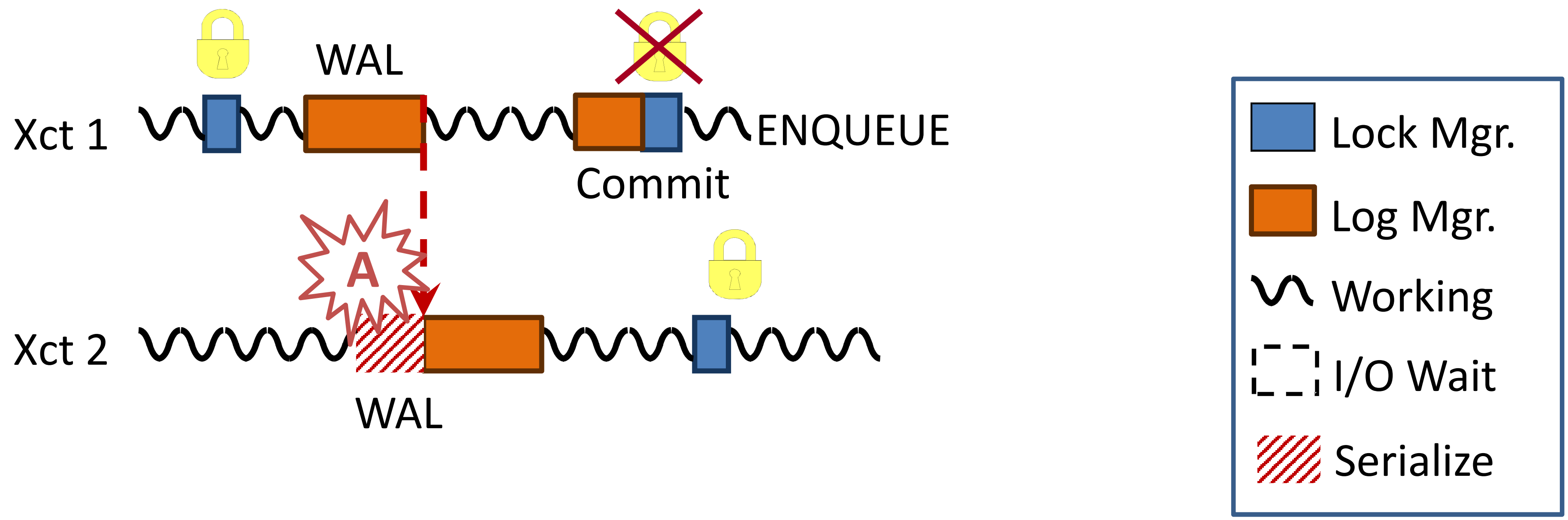
commit pipelining

- Request log sync but do not wait
- Detach transaction state and enqueue it somewhere (xct nearly stateless at commit)
- Dedicate 1+ workers to commit processing



commit rate no longer tied to OS & I/O

a day in the life of a serial log

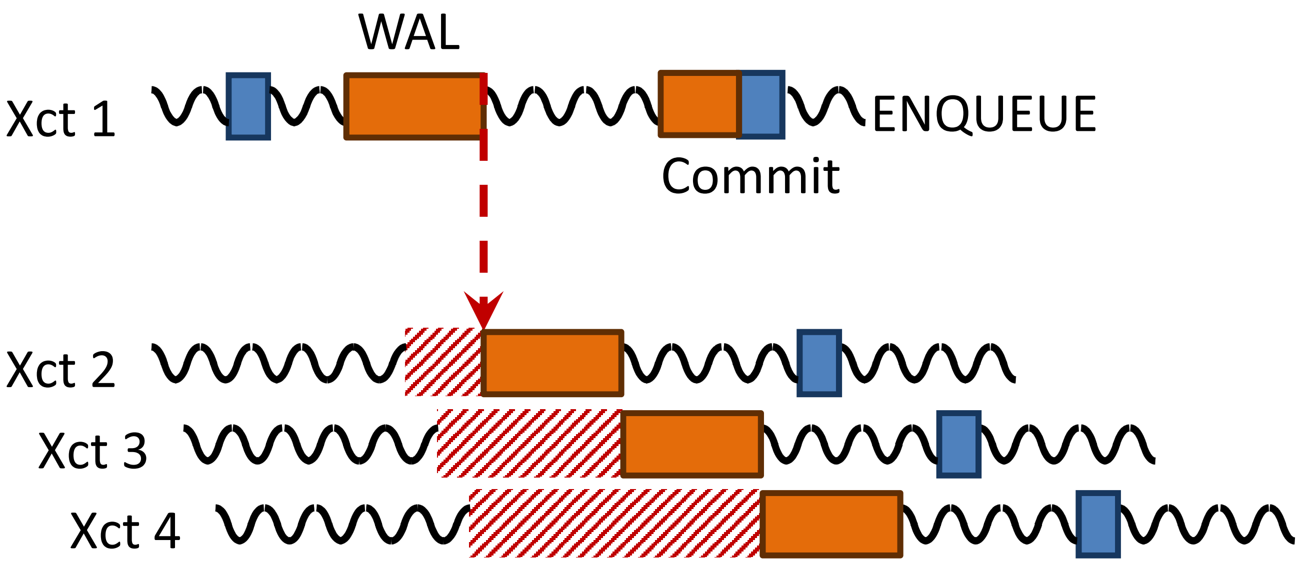


A Serialize at the log head

~~I/O delay to harden the commit record~~

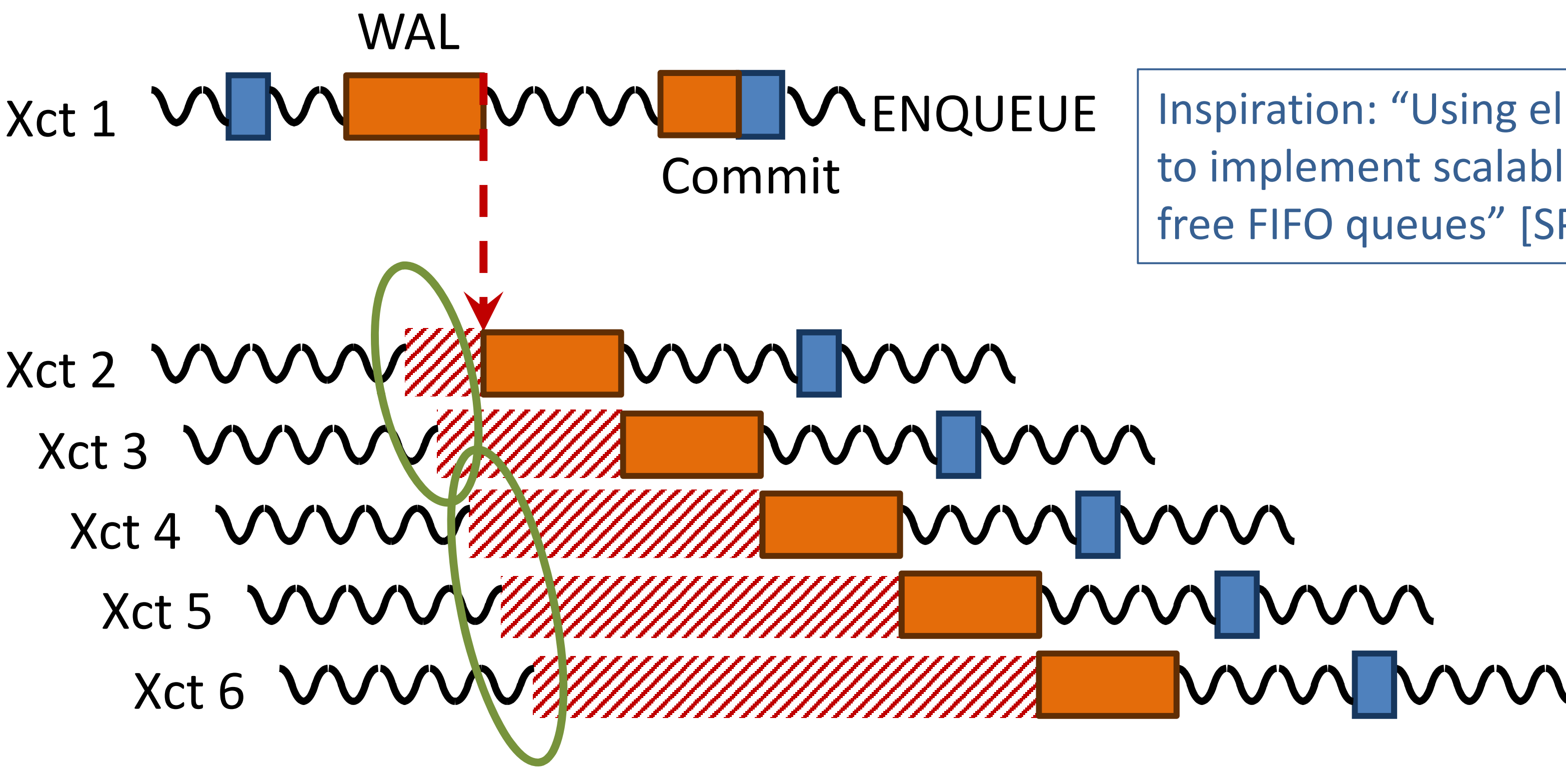
~~Serialize on incompatible lock~~

a day in the life of a serial log



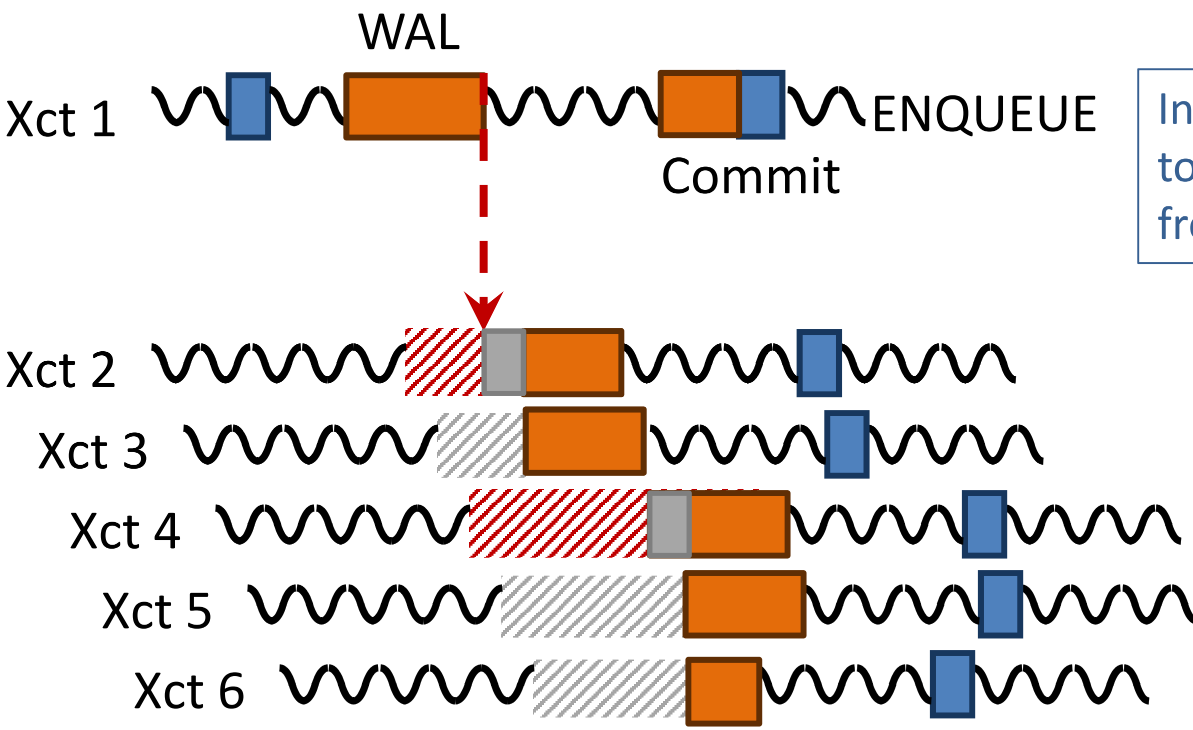
Log insertion becomes a bottleneck for large numbers of threads on modern machines

insight: aggregate waiting requests



Inspiration: "Using elimination to implement scalable and lock-free FIFO queues" [SPAA2005b]

insight: aggregate waiting requests



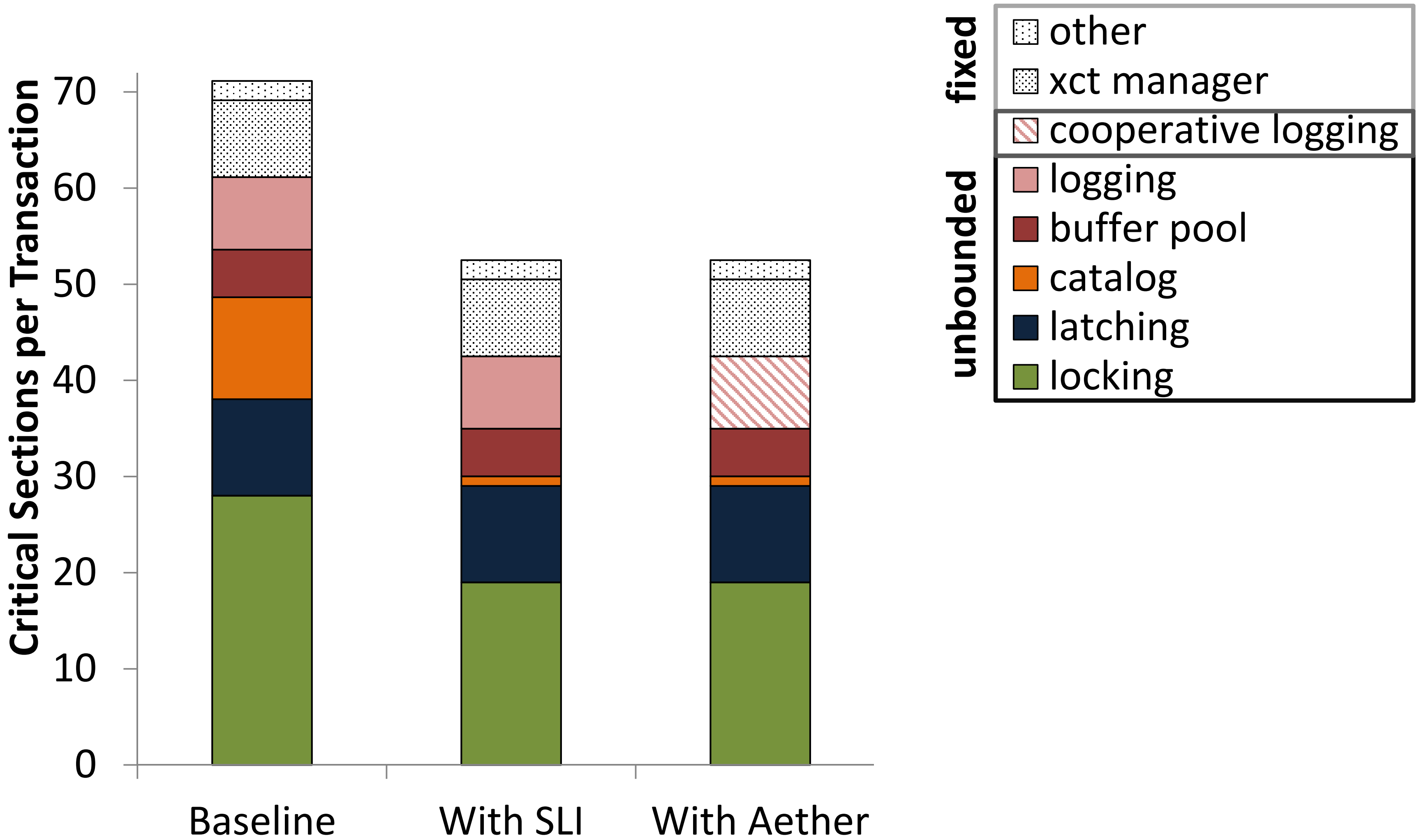
Inspiration: "Using elimination to implement scalable and lock-free FIFO queues" [SPAA2005b]

Self-regulating:

longer queue -> larger groups -> shorter queue

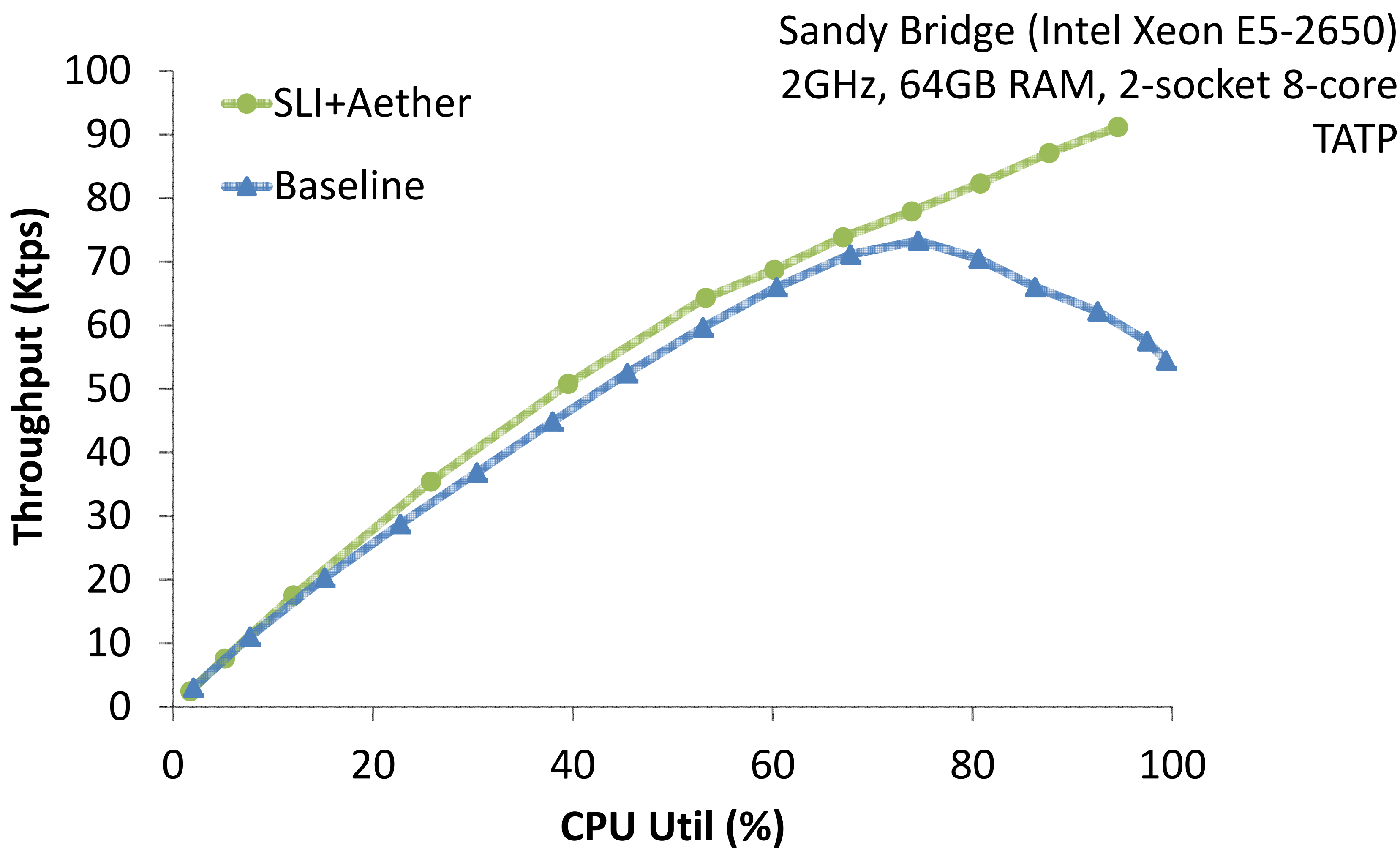
decouple contention from #threads & log entry size

impact of logging improvements



same amount of communication, but well-behaved

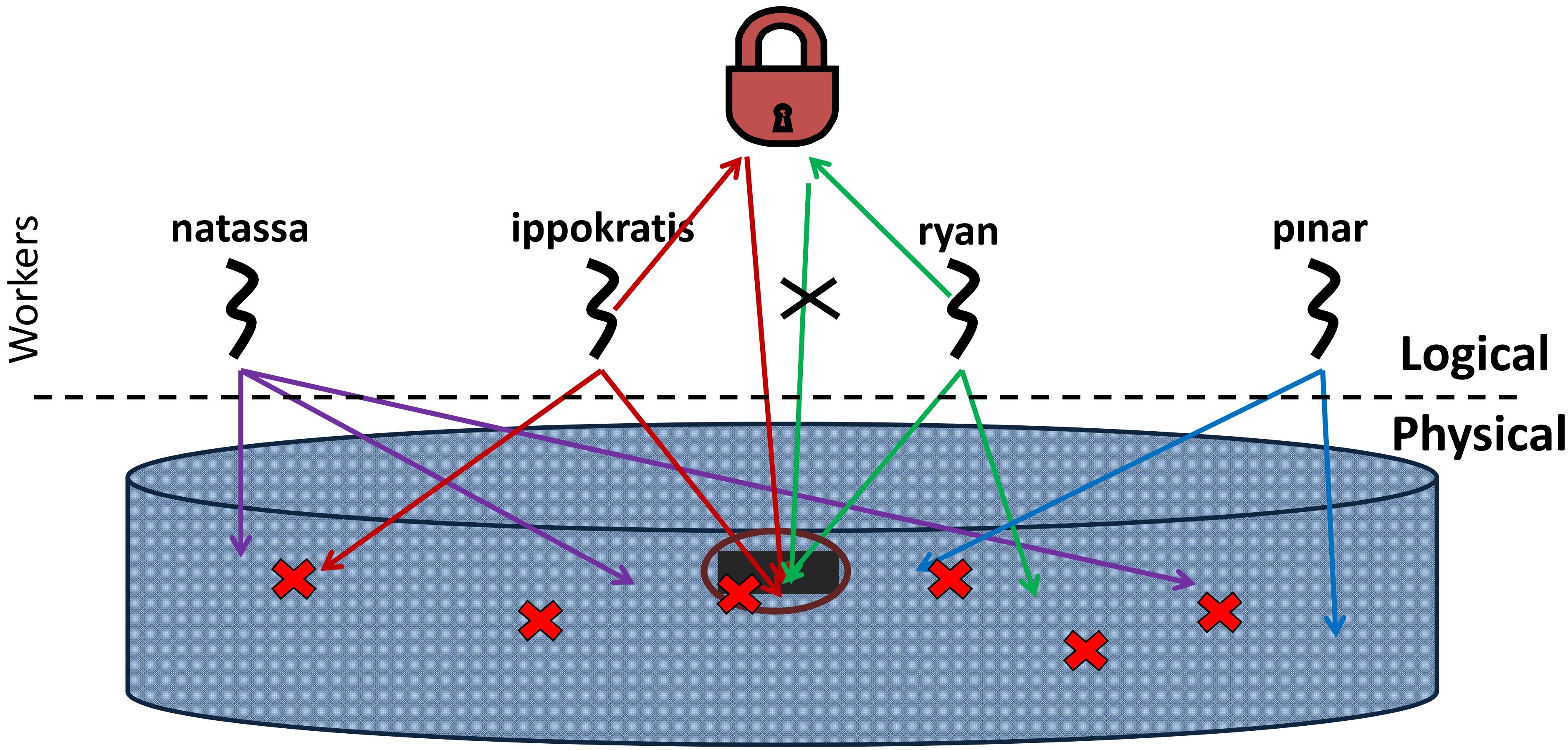
performance impact of SLI&Aether



outline

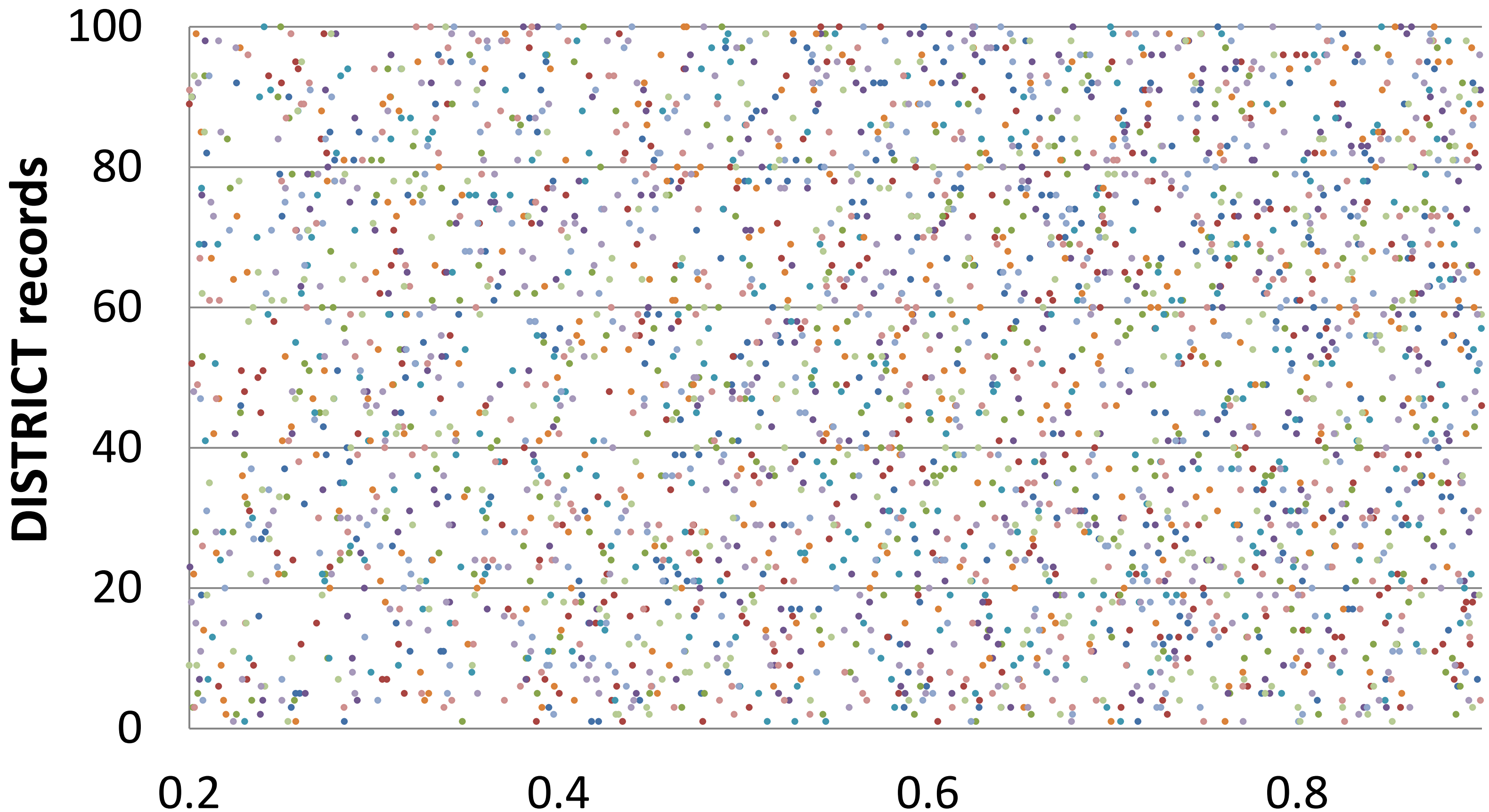
- introduction ~ *20 min*
- part I: achieving scalability in Shore-MT ~ *1 h*
  - taking global communication out of locking
  - extracting parallelism in spite of a serial log
  - designing for better communication patterns
- part II: behind the scenes ~ *20 min*
- part III: hands-on ~ *20 min*

shared-everything



contention due to unpredictable data accesses

thread-to-transaction – access pattern



unpredictable data accesses

clutter code with critical sections -> contention

data-oriented transaction execution

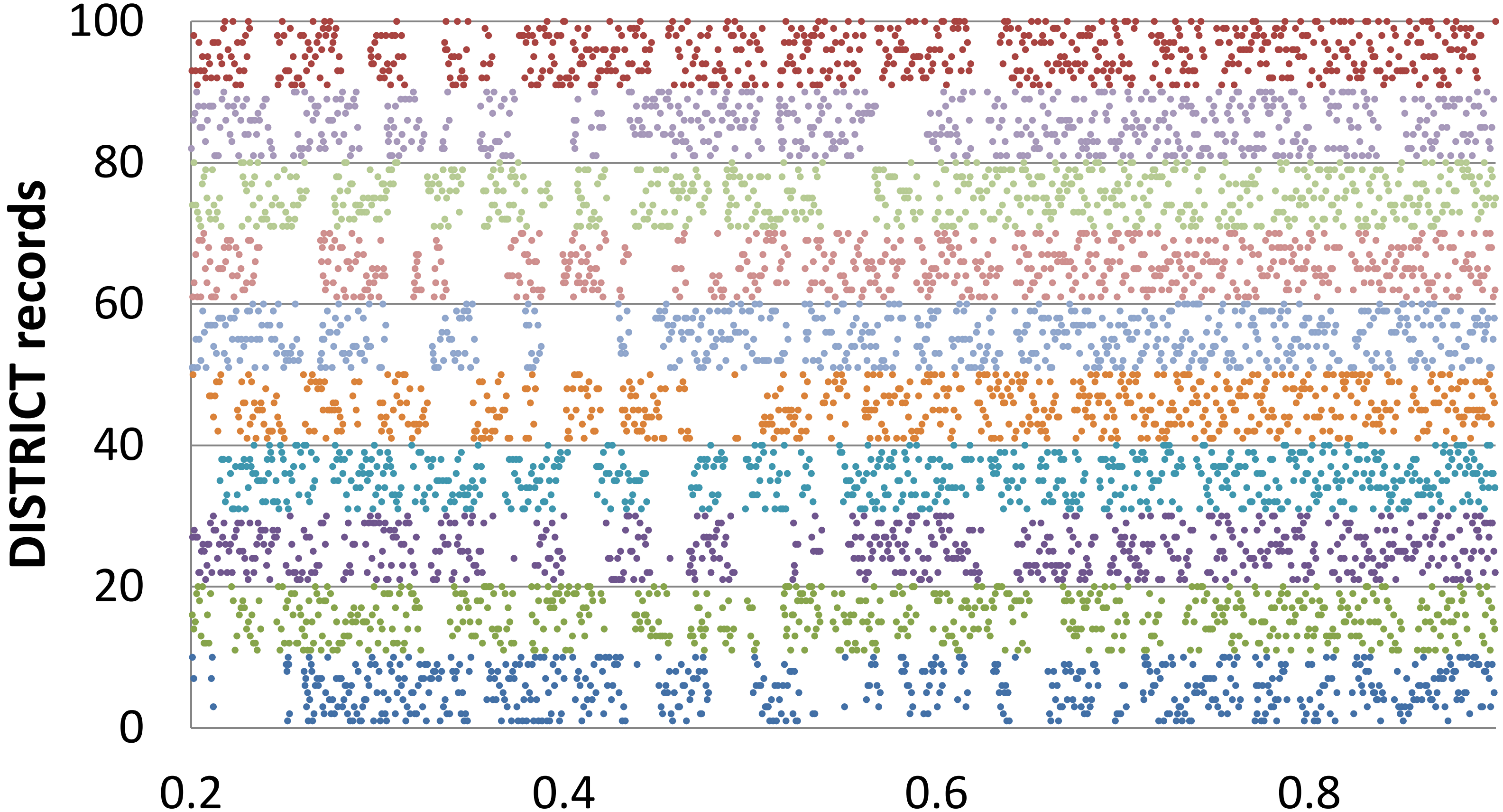
[PVLDB2010b]

- Transaction does not dictate the data accessed by worker threads
- Break each transaction into smaller actions
  - Depending on the data they touch
- Execute actions by "data-owning" threads
- Distribute and privatize locking across threads

new transaction execution model

convert centralized locking to thread-local

thread-to-data – access pattern

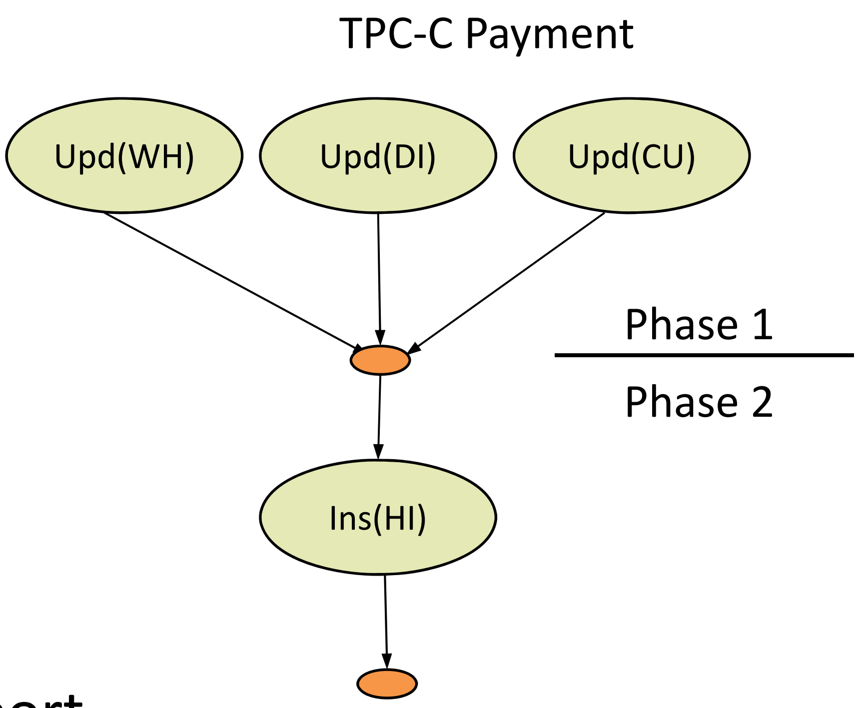


predictable data access pattern

opens the door to many optimizations

input: transaction flow graph

- Graph of Actions & Rendezvous Points
- Actions
  - Operation on specific database
  - Table/Index it is accessing
  - Subset of routing fields
- Rendezvous Points
  - Decision points (commit/abort)
  - Separate different phases
  - Counter of the # of actions to report
  - Last to report initiates next phase



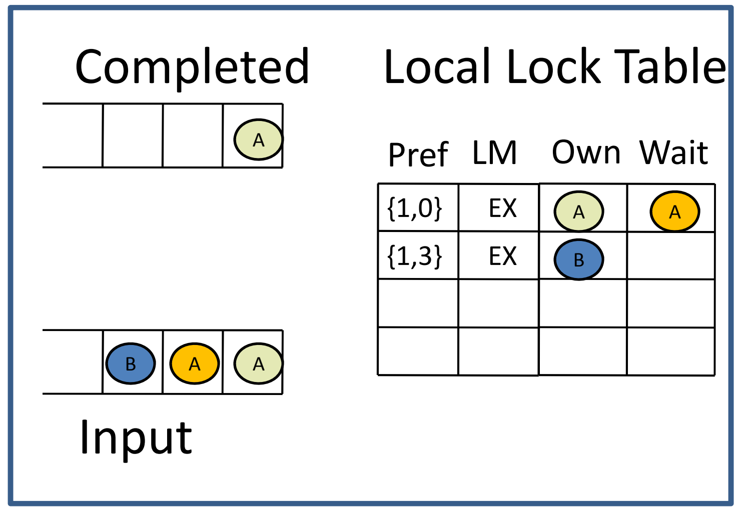
partitions & executors

- Routing table at each table
  - {Routing fields → executor}

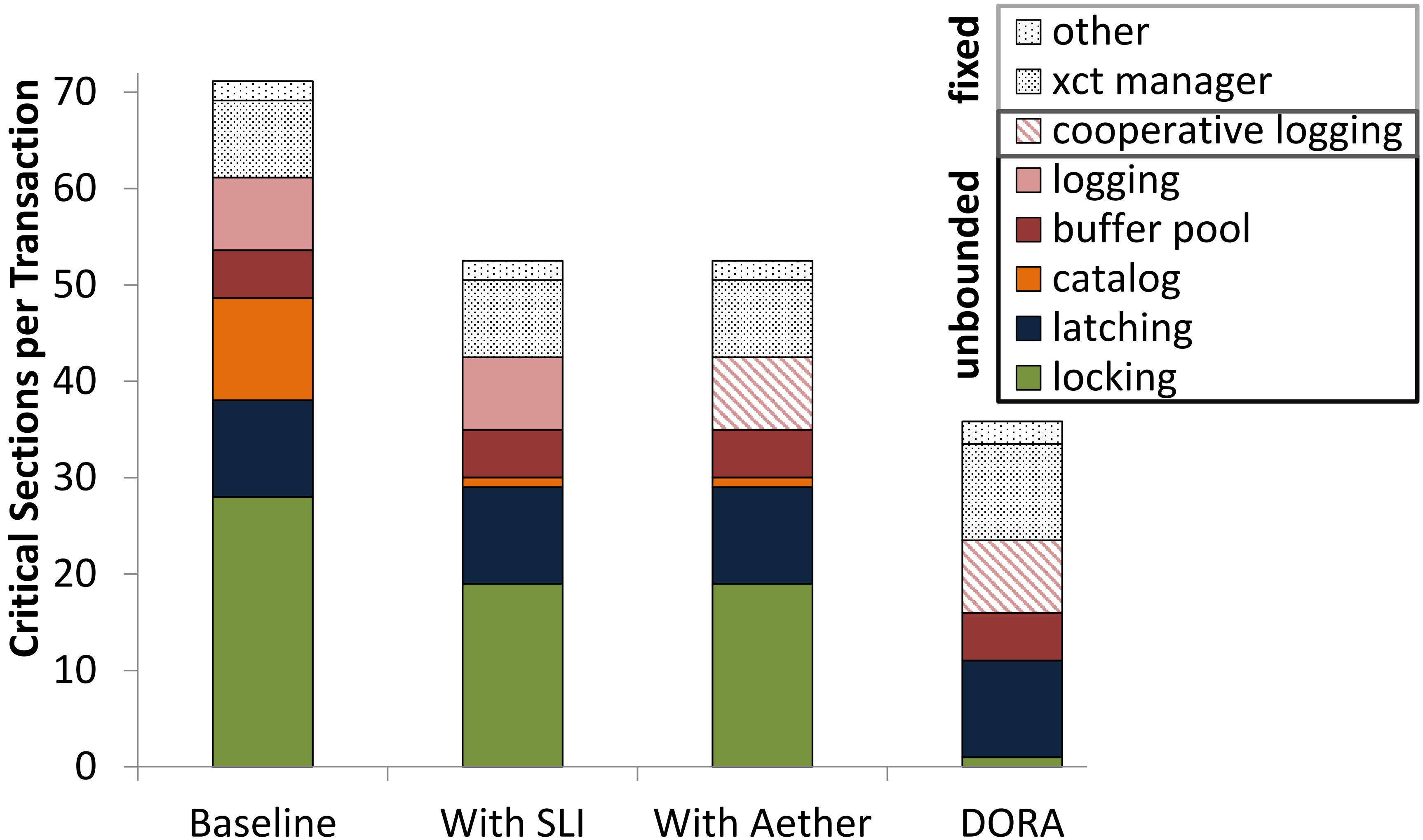
Routing fields: {WH_ID, D_ID}

Range	Executor
A-H	1
I-N	2

- Executor thread
  - Local lock table
    - {RoutingFields + partof(PK), LockMode}
    - List of blocked actions
  - Input queue
    - New actions
  - Completed queue
    - On xct commit/abort
    - Remove from local lock table
  - Loop completed/input queue
  - Execute requests in serial order



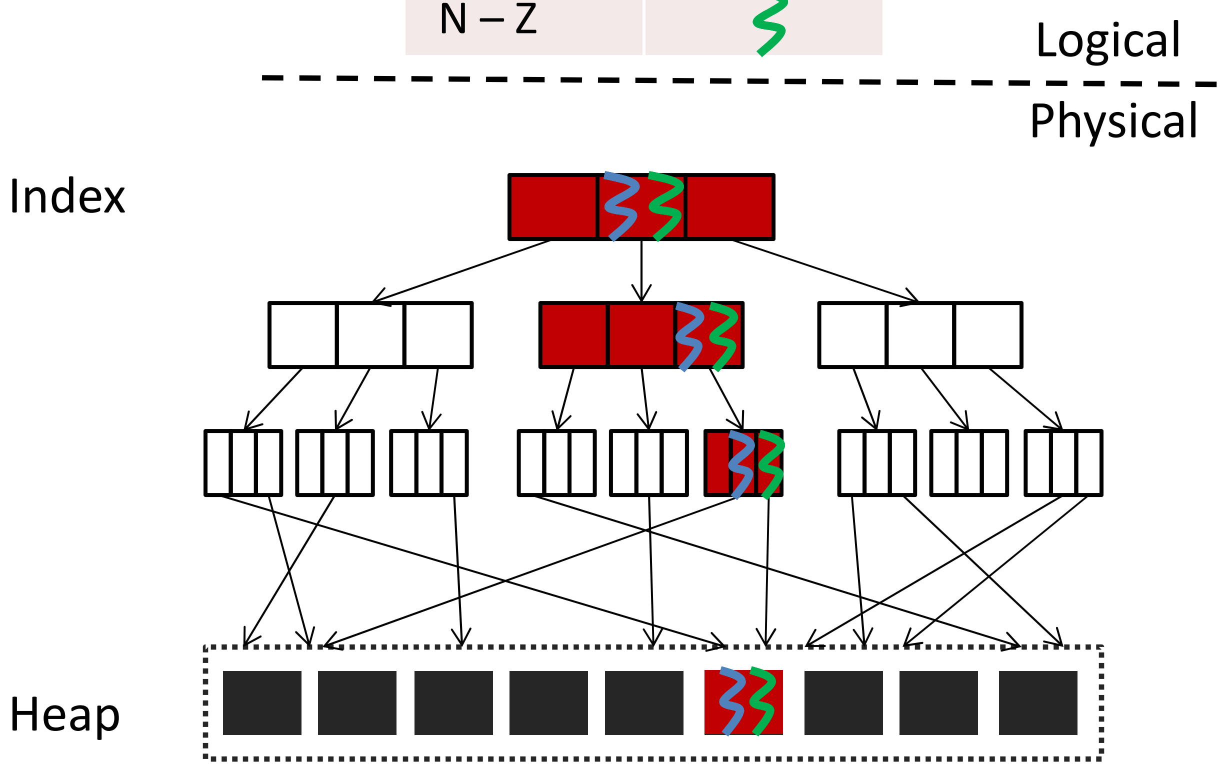
DORA's impact on communication



re-architected the unbounded communication

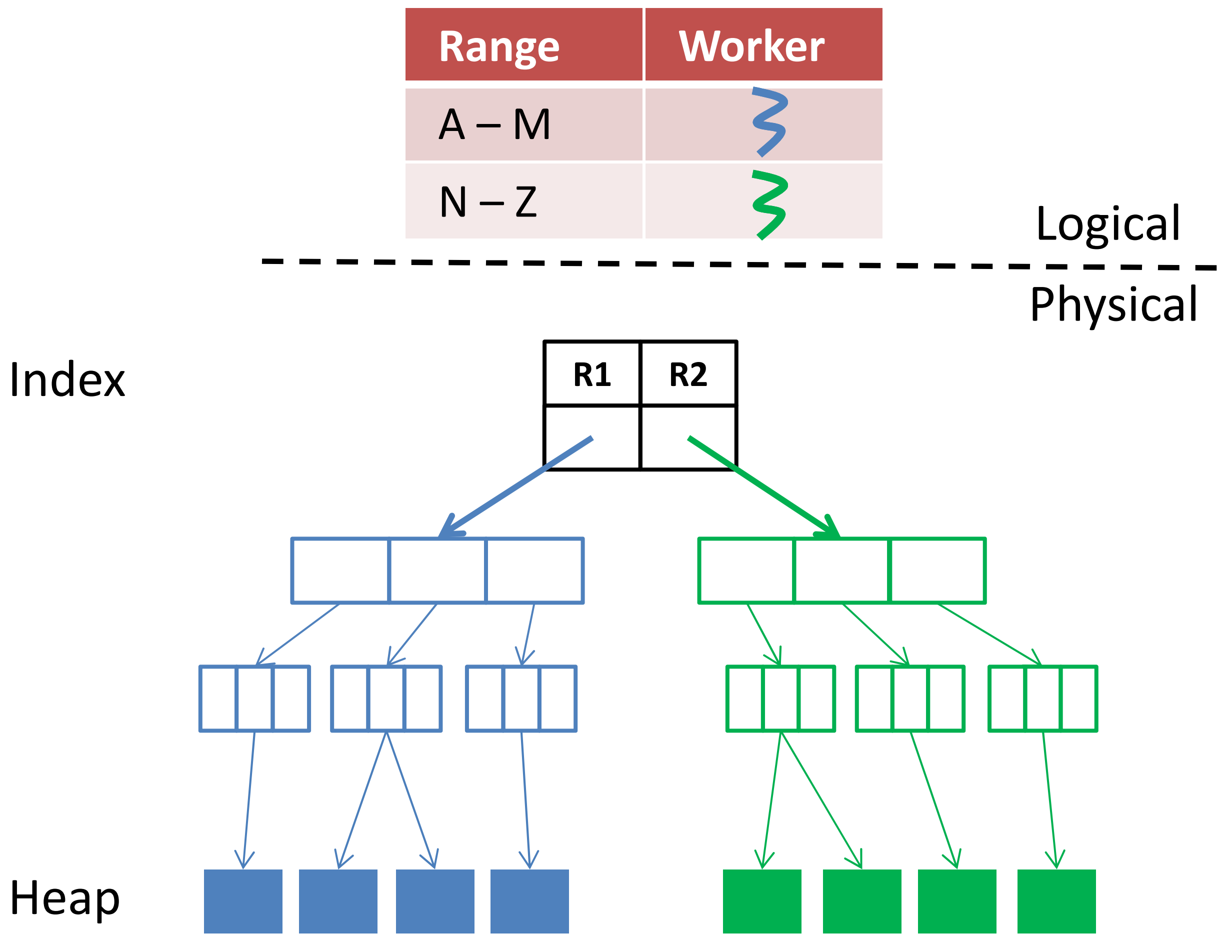
physical conflicts

Range	Worker
A – M	
N – Z	



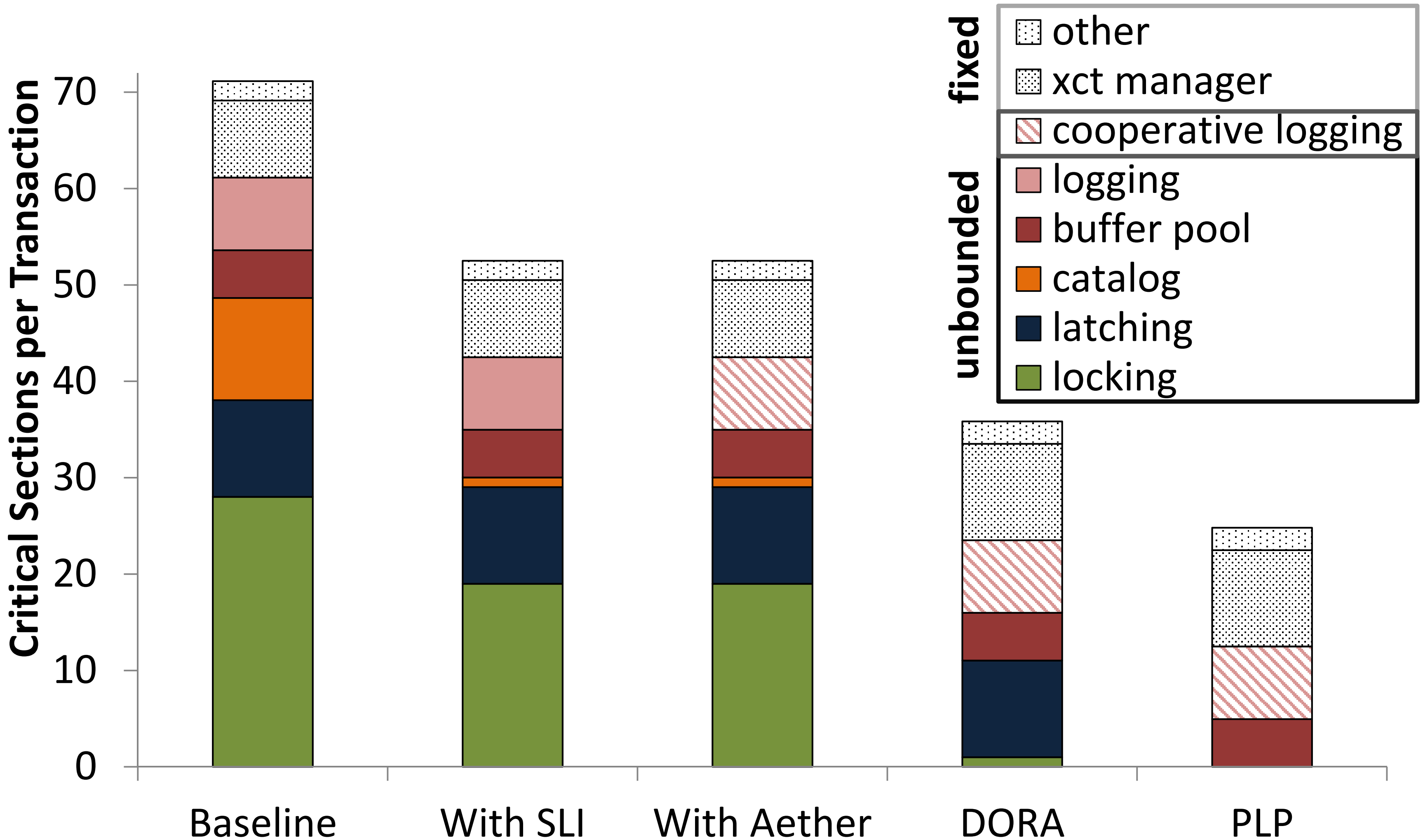
conflicts on both index & heap pages

physiological partitioning (PLP) [PVLDB2011]



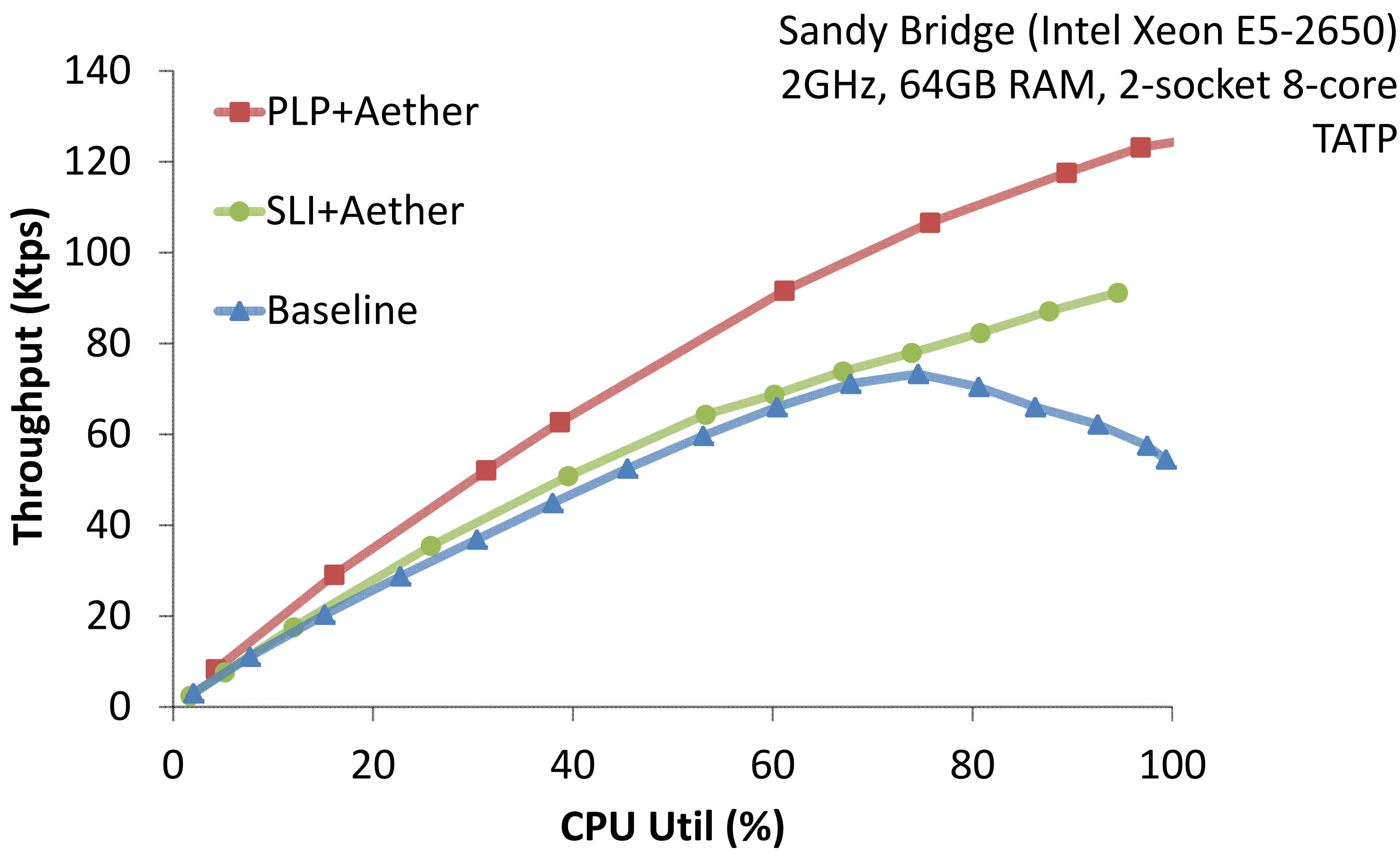
latch-free physical accesses

road to scalable OLTP



eliminated 90% of unbounded communication

performance impact of DORA&PLP



outline

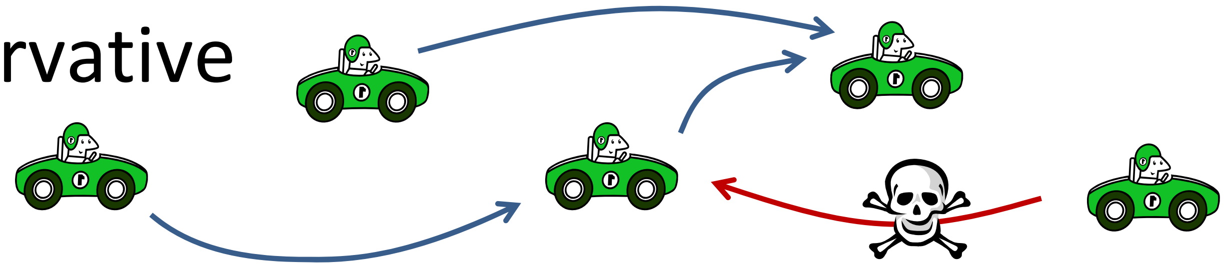
- introduction *~ 20 min*
- part I: achieving scalability in Shore-MT *~ 1 h*
- part II: behind the scenes *~ 20 min*
  - Characterizing synchronization primitives
  - Scalable deadlock detection
- part III: hands-on *~ 20 min*

lots of little touches in Shore-MT

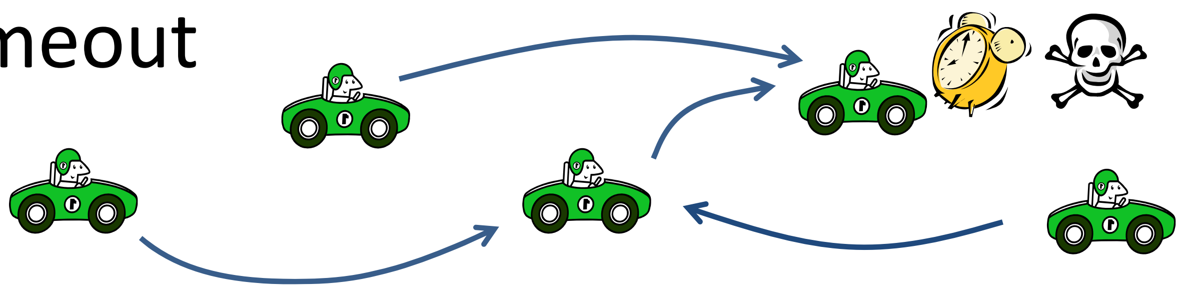
- "Dreadlocks" deadlock detection since 2009
- Variety of efficient synchronization primitives
- Scalable hashing since 2009
  - Lock table: fine-grained (per-bucket) latching
  - Buffer pool: cuckoo hashing
- Multiple memory management schemes
  - Trash stacks, region allocators
  - Thread-safe slab allocators, RCU-like "lazy deletion"
- Scalable page replacement/cleaning

Deadlock detection is hard!

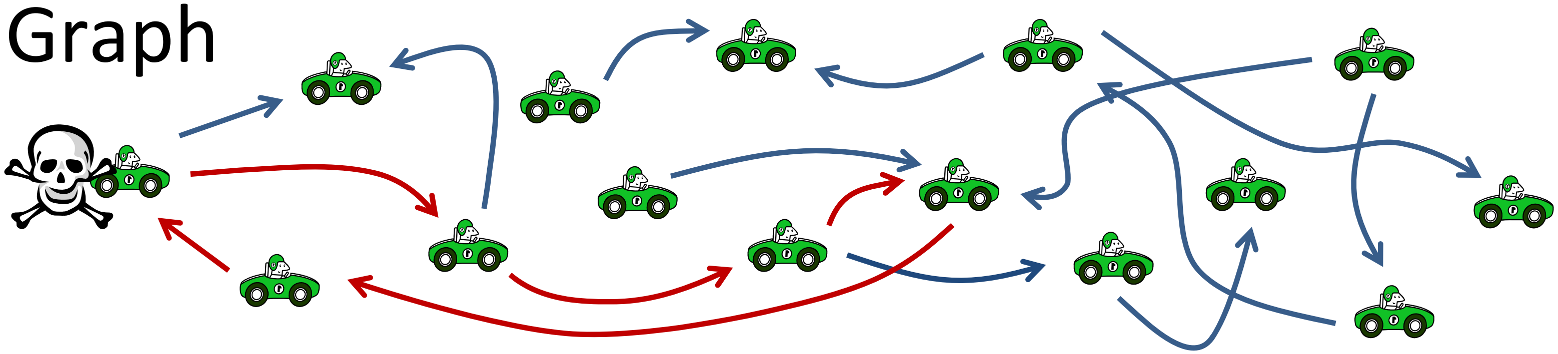
- Conservative



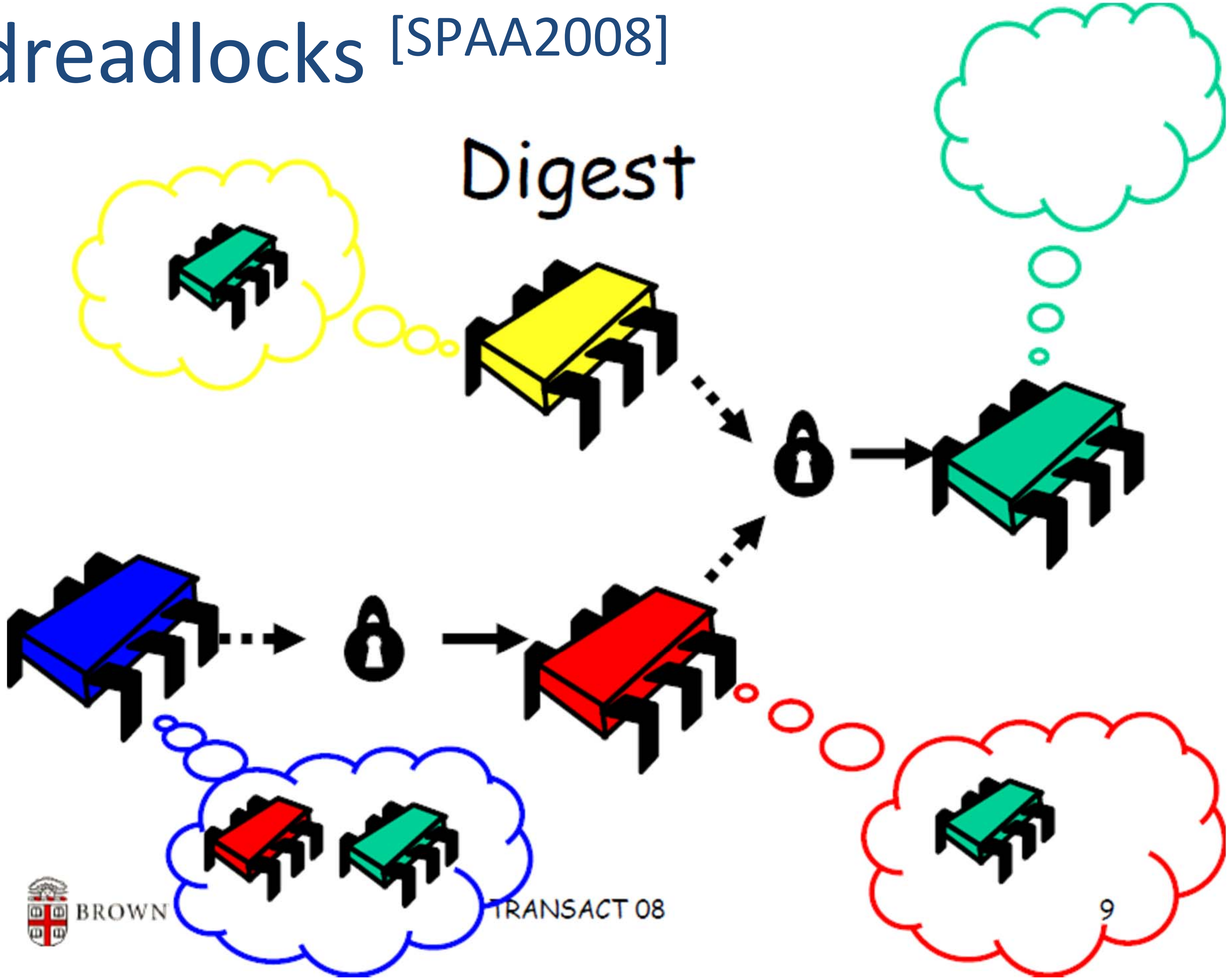
- Timeout



- Graph

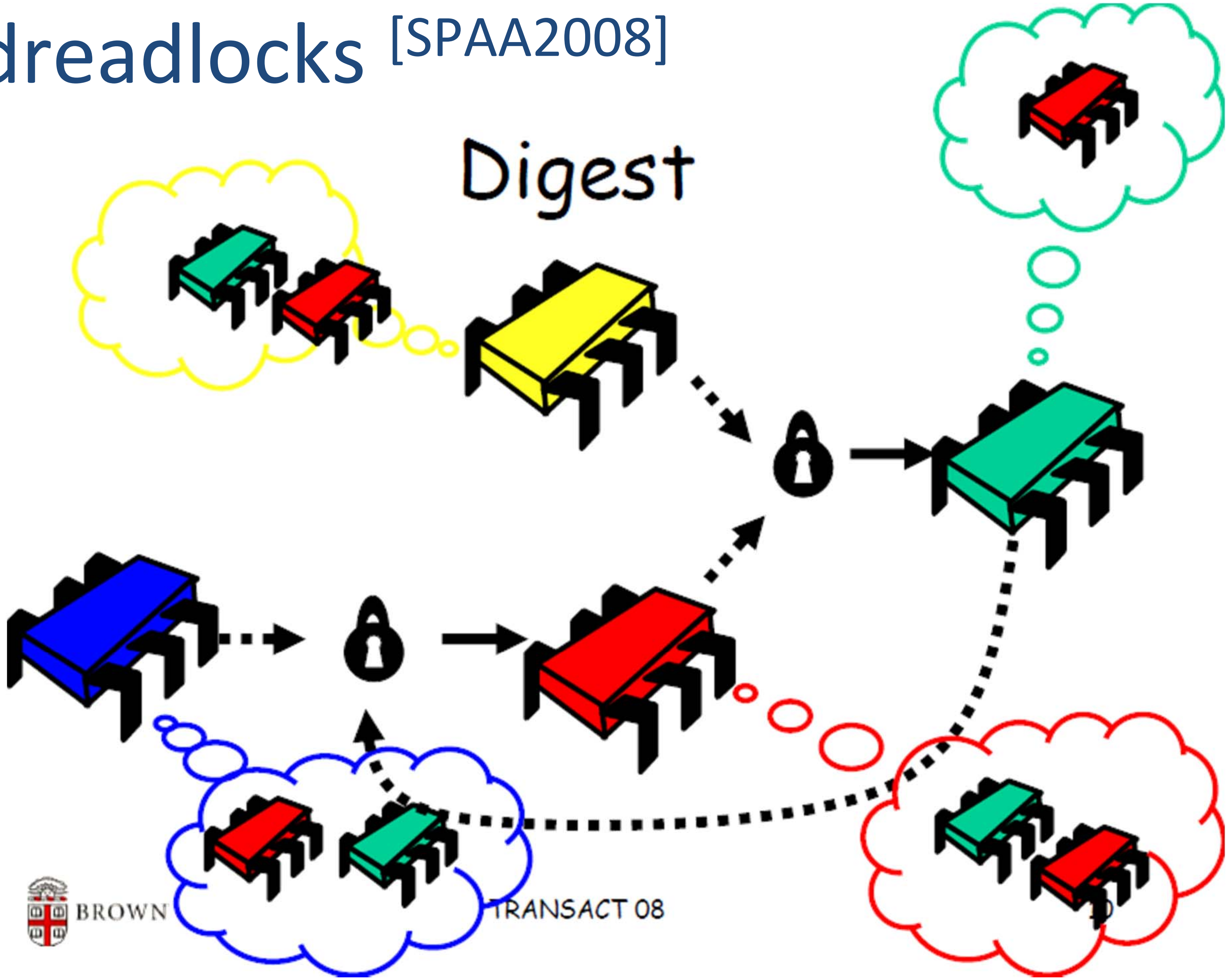


dreadlocks [SPAA2008]

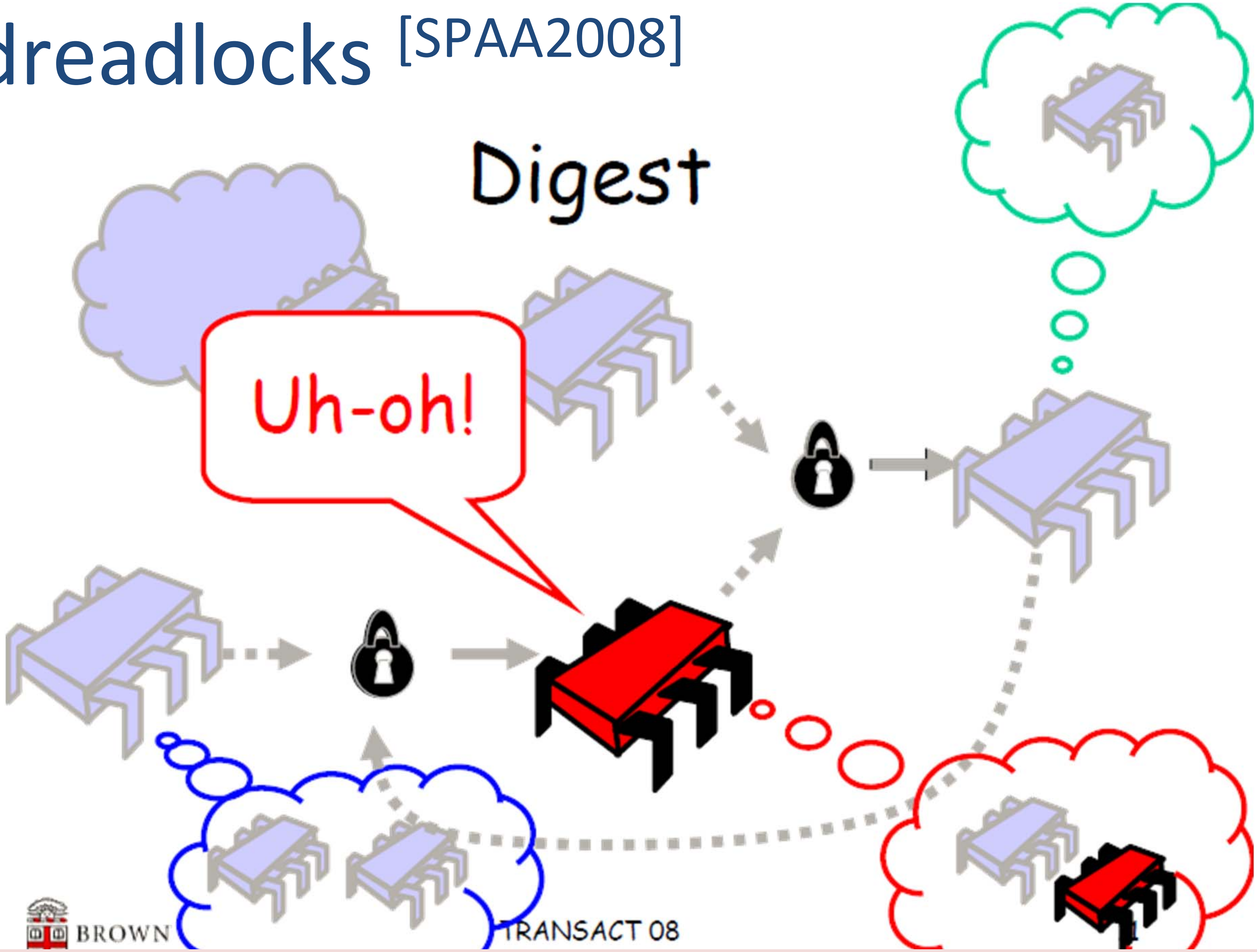


Source: http://wwwa.unine.ch/transact08/slides/Herlihy-Dreadlocks.pdf

dreadlocks [SPAA2008]



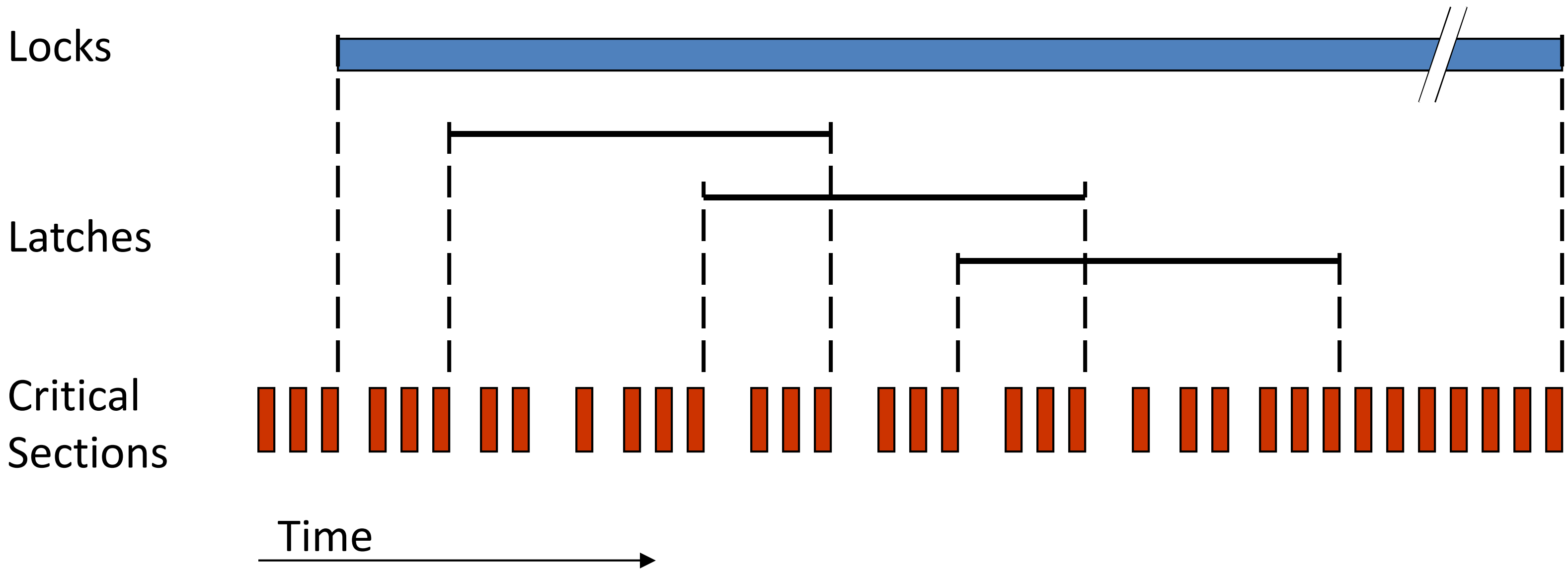
dreadlocks [SPAA2008]



simple, scalable, & efficient! choose any three

locks and latches aren't everything

Synchronization required for one index probe (non-PLP)



- Critical sections protect log buffer, stats, lock and latch internal state, thread coordination…

diverse use cases, selecting the best primitive?

lock-based approaches

Blocking OS mutex

- ✓ Simple to use
- × Overhead, unscalable

Test and set spinlock (TAS)

- ✓ Efficient
- × Unscalable

Queue-based spinlock (“MCS”)

- ✓ Scalable
- × Mem. management

Reader-writer lock

- ✓ Concurrent readers
- × Overhead

lock-free approaches

Atomic updates

- ✓ Efficient
- × Limited applicability

Lock-free algorithms

- ✓ Scalable
- × Special-purpose algs

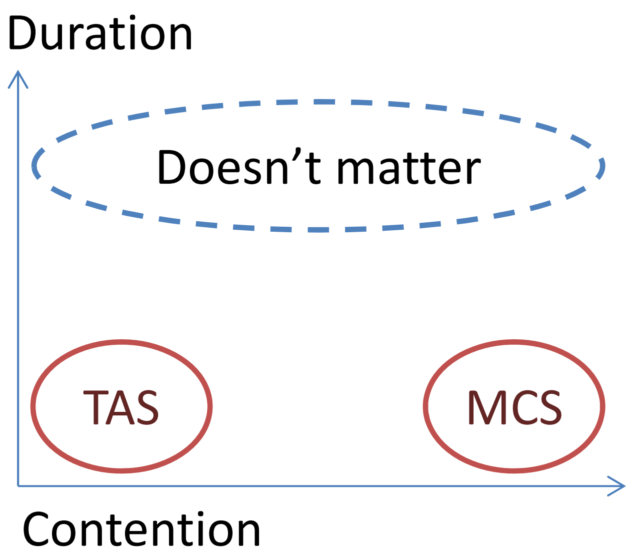
Optimistic concurrency control (OCC)

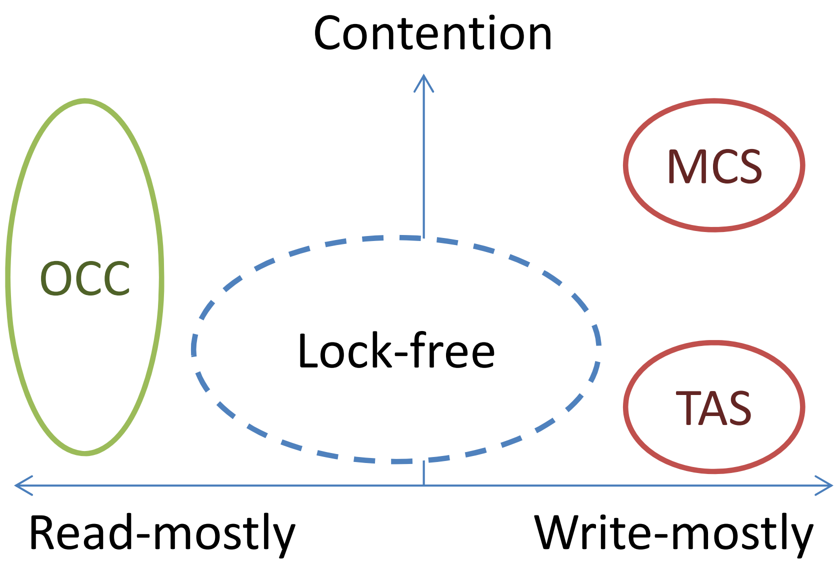
- ✓ Low read overhead
- × Writes cause livelock

Hardware approaches (e.g. transactional memory)

- ✓ Efficient, scalable
- × Not widely available

synchronization “cheat sheet”





- OS blocking mutex: only for scheduling
- Reader-writer lock: dominated by OCC/MCS
- Lock-free: sometimes (but be very, very careful)

outline

- introduction *~ 20 min*
- part I: achieving scalability in Shore-MT *~ 1 h*
- part II: behind the scenes *~ 20 min*
- part III: hands-on *~ 20 min*

Shore-MT: first steps

- Download

  $ hg clone https://bitbucket.org/shoremt/shore-mt

- Build

  $ ./bootstrap

  $ ./configure --enable-dbgsymbols(optional)
  [in SPARC/Solaris: CXX=CC ./configure ...]

  $ make –j

- Storage manager (sm)
  - Quick tests, experiments: src/sm/tests

Shore-MT API

- src/sm/sm.h
  - API function declarations and documentation
- src/sm/smindex.cpp
  - Implementation of the index related API functions
- src/sm/smfile.cpp
  - Implementation of the record file related API functions

concurrency control in Shore-MT

Concurrency Control	
t_cc_none	t_cc_record
t_cc_page	t_cc_file
t_cc_vol	
t_cc_kvl (default)	
t_cc_im (default in kits)	

Locks
Volume
Store (Files, Indexes)
Key-Value
Page
Record
Extent

Key-Value
t_cc_kvl: if index is unique <key> else <key, value>
t_cc_im: <value> (actually, record-id)

Shore-Kits

- Application layer for Shore-MT
- Available benchmarks:
  - OLTP: TATP, TPC-B, TPC-C, TPC-E
  - OLAP: TPC-H, Star schema benchark (SSB)
  - Hybrid: TPC-CH (coming-up)

download & build Shore-Kits

- Download

  $ hg clone https://bitbucket.org/shoremt/shore-kits

- Build

  $ ln –s <shore-storage-manager-dir>/m4

  $ ./autogen.sh

  $ ./configure --with-shore=<shore-storage-manager-dir>
  --with-glibtop(for reporting throughput periodically)
  --enable-debug(optional)
  [in SPARC/Solaris: CXX=CC ./configure ...]

  $ make -j

Shore-Kits: directory structure

- src|include/sm
  - Interaction with Shore-MT API
- src|include/workloads
  - Workload implementations for baseline Shore-MT
- src|include/dora
  - DORA/PLP logic and workload implementations
- shore.conf
  - Where you specify workload parameters

how to run Shore-Kits?

- $ ln –s log log-tpcb-10
- $ rm log/*; rm databases/*
- $ ./shore_kits –c tpcb-10 –s baseline –d normal –r
  - $ help
  - $ trxs
  - $ elr
  - $ log cd
  - $ measure 10 1 10 10 0 1
- <restart>

some advice for benchmarking

- For in-memory runs
  - Your laptop might suffer (mine does ☺)
  - Unless you want convoys, make sure
    - #loaders, #clients, #workers used < #available hardware contexts
- If you want high utilization
  - Do not have synchronous clients
    (e.g. asynch option in VoltDB)
  - Or make your clients send requests in large batches
    (e.g. shore-kits, db-cl-batchsz parameter in shore.conf)
  - Group commit and commit pipelining won’t improve throughput if all outstanding requests are in the group!

more advice for benchmarking

- Use fixed-duration measurement runs (e.g. “measure” command in shore-kits)
  - Start workers, snapshot stats, wait, snapshot stats again, stop workers; result is delta between snapshots.
  - Avoids start/stop effects
  - Duration of runs more predictable (even if throughput is unexpectedly low or high)
- Run long enough to catch log checkpointing
  - Checkpoints do impact performance, unfair to ignore them
  - Gives page cleaning time to ramp up as well

why shore-kits isn't enough?

- Shore-Kits is great, but …
  - Implementation overhead for simple queries
  - Does not keep metadata persistently
  - Does not allow ad-hoc requests
  - Cannot switch databases on-the-fly

coming soon: Shore-Kits++

Closing Remarks

- Hardware keeps giving more parallelism
- But achieving scalability is hard
- Any unbounded communication eventually becomes a bottleneck

- Shore-MT and Shore-Kits
  - Good test-bed for research
  - New release: 7.0
  - Check http://diaswww.epfl.ch/shore-mt/

Thank you!

References – I

- [VLDB1990] C. Mohan: ARIES/KVL: a key-value locking method for concurrency control of multiaction transactions operating on B-tree indexes.
- [SIGMOD1992] C. Mohan, F. Levine: ARIES/IM: an efficient and high concurrency index management method using write-ahead logging.
- [VLDB2001] T. Lahiri, V. Srihari, W. Chan, N. MacNaughton, S. Chandrasekaran: Cache Fusion: Extending Shared-Disk Clusters with Shared Caches.
- [SPAA2005a] M.A. Bender, J.T. Fineman, S. Gilbert, B.C. Kuszmaul: Concurrent cache-oblivious B-trees.
- [SPAA2005b] M. Moir, D. Nussbaum, O. Shalev, N. Shavit: Using elimination to implement scalable and lock-free FIFO queues.
- [VLDB2007] M. Stonebraker, S. Madden, D. J. Abadi, S. Harizopoulos, N. Hachem, P. Helland: The End of an Architectural Era (It's Time for a Complete Rewrite).
- [SPAA2008] E. Koskinen, M. Herlihy: Dreadlocks: Efficient Deadlock Detection.
- [DaMoN2008] R. Johnson, I. Pandis, A. Ailamaki: Critical Sections: Re-Emerging Scalability Concerns for Database Storage Engines.

References – II

- [EDBT2009] R. Johnson, I. Pandis, N. Hardavellas, A. Ailamaki, B. Falsafi: Shore-MT: a scalable storage manager for the multicore era.
- [VLDB2009] R. Johnson, I. Pandis, A. Ailamaki: Improving OLTP Scalability using Speculative Lock Inheritance.
- [PVLDB2010a] R. Johnson, I. Pandis, R. Stoica, M. Athanassoulis, A. Ailamaki: Aether: A Scalable Approach to Logging.
- [PVLDB2010b] I. Pandis, R. Johnson, N. Hardavellas, A. Ailamaki: Data-Oriented Transaction Execution.
- [PVLDB2011] I. Pandis, P. Tözün, R. Johnson, A. Ailamaki: PLP: Page Latch-free Shared-everything OLTP.
- [ICDE2011] A. Kemper, T. Neumann: HyPer – A hybrid OLTP & OLAP main memory database system based on virtual memory snapshots.
- [TODS2012] G. Graefe, H. Kimura, H. Kuno: Foster B-trees.
- [VLDBJ2013] R. Johnson, I. Pandis, A. Ailamaki: Eliminating unscalable communication in transaction processing.

References – III

- [SIGMOD2013] C. Diaconu, C. Freedman, E. Ismert, P. Larson, P. Mittal, R. Stonecipher, N. Verma, M. Zwilling: Hekaton: SQL Server's Memory-Optimized OLTP Engine.
- [SIGMOD2013a] G. Graefe, M. Lillibridge, H. Kuno, J. Tucek, A. Veitch: Controlled Lock Violation.
- [SIGMOD2013b] H. Jung, H. Han, A. Fekete, G. Heiser, H. Yeom: A Scalable Lock Manager for Multicores.
- [PVLDB2014] Pelley et al