

HTM in the wild

Konrad Lai

June 2015

Industrial Considerations for HTM

- Provide a clear benefit to customers
 - Improve performance & scalability
 - Ease programmability going forward
- Improve something common and fundamental
 - Widely used critical section/lock-based primitives
- In an easy to use and deploy manner
 - Minimal eco-system impact and effort
 - Clean architectural boundaries
- While managing HW design and validation complexity

HTM [Mechanism]

- 1993 HTM paper, Herlihy & Moss
- 2001 Lock elision, Rajwar & Goodman
- 2003 STM, TM [programming model], ...
- 2006 1st TRANSACT

- Commercial Implementations
 - 2011 IBM Blue Gene/Q
 - 2012 IBM zEC12 mainframe
 - 2013 Intel 4th generation Core (Haswell)
 - 2014 IBM POWER8
 - 2015 Intel Xeon E7 v3, 4-way and 8-way SMP

- 1993 idea plus 2001 usage model
 - Lock Elision
 - Probabilistic lock free
- 2003 onward is still work in progress

HTM Features Convergence

- Convergence over basic functionalities...
 - Best effort HTM
 - Leverage cache coherency protocol/cache(s)
 - Strong Isolation
 - Hardware buffering
 - Reasonable buffer size
 - No instruction count limit
 - Checkpoint of Registers
 - Implicitly Transactional
- Some differences...
 - IBM BGQ supports thread speculation
 - IBM zEC supports constrained transactions
 - IBM POWER8 supports suspend/resume
 - IBM zEC/POWER8 supports non-txn stores (but differently)
 - IBM POWER8 supports Recovery Only Transactions
 - TX capacity varies medium to large

Lemming Effect

XA : xbegin; test; xabort; (retry loop when lock is busy)

L-U: Lock; critical section; Unlock (non-transactional execution)

T1 --AL-----UXAXAXAXAXA sssssssssssssssssL-----UXAXA

T2 ---AXAXAXAXAXA sssL-----UXAXAXAXAXAXA sssssssssssssL---

T3 ---AXAXAXAXAXA sssssssssssssssssL-----UXAXAXAXAXA ssssss

Persistent convoy of non-transactional execution

Elision is effectively disabled until all threads have serially released the lock

- Disabled forever if at least 1 thread is holding the lock

Fix is simple

- Don't retry until the lock is free
- Use well-known test-and-test-&-set pattern

T1 --AL-----UX-----

T2 ---A sssssssssssssssX-----

T3 ---A sssssssssssssssX-----

Appear in far too many refereed papers

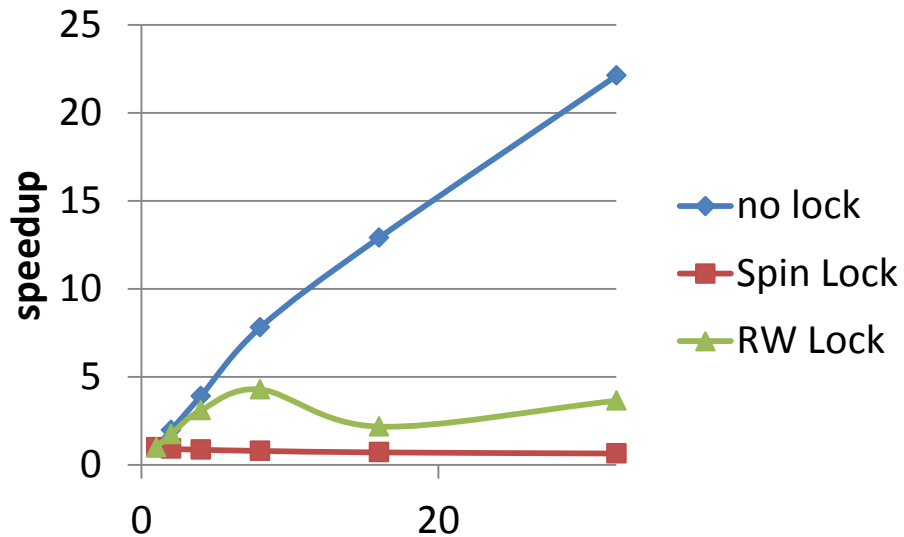
Intel TSX Case Studies: Databases

- HPCA 2014
Improving In-Memory Database Index Performance with Intel® Transactional Synchronization Extensions
- Tomas Karnagel, Roman Dementiev, Ravi Rajwar, Konrad Lai, Thomas Legler, Benjamin Schlegel, Wolfgang Lehner (Intel, SAP AG and TU Dresden)
- EuroSys 2014
Using Restricted Transactional Memory to Build a Scalable In-Memory Database.
- Zhaoguo Wang, Hao Qian, Jinyang Li, Haibo Chen (Fudan University, Shanghai Jiao Tong University, New York University)
- TDKE 2015
Scaling HTM-Supported Database Transactions to Many Cores
- Viktor Leis, Alfons Kemper, Thomas Neumann (TU Munchen)

A Case Study: Two Index Implementations

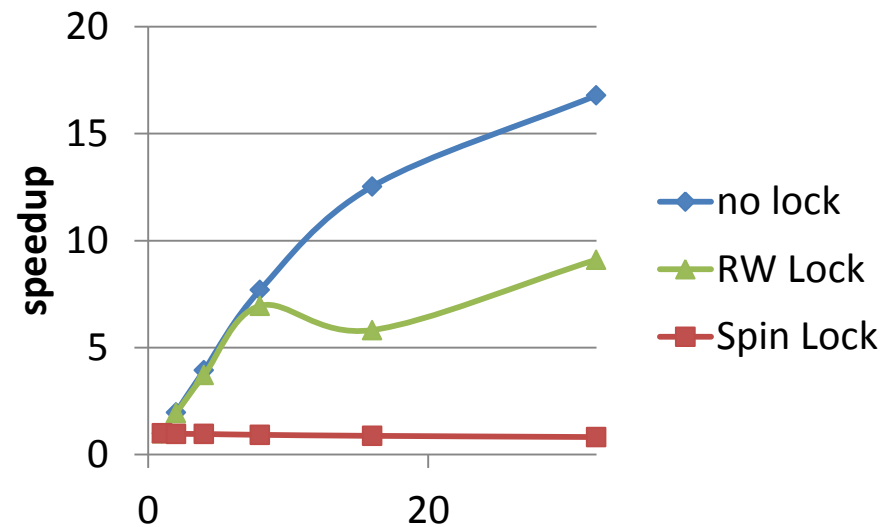
B+Tree Index

(a common index implementation)



Delta Storage Index

(from the SAP HANA® database)

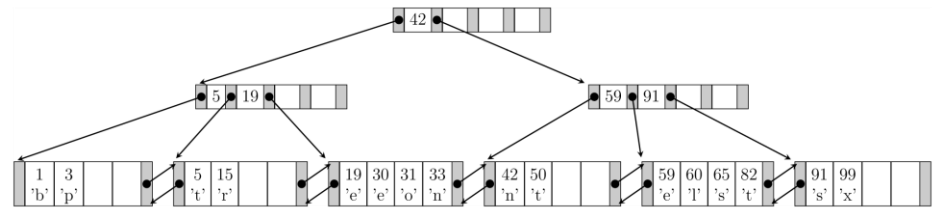


- **Read-Only** Queries on Dual Socket Intel® Xeon® E5-2680 Server

Hidden Scalability Impact of Atomic Read-Modify-Write Operations

Case Study: Index Tree Implementations

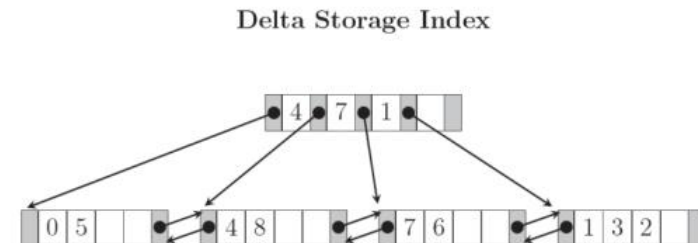
- SAP HANA Database
 - Read optimized column store database system
- Two index implementations
 - B+Tree [Data Structure]
 - Common implementation
 - Smaller foot print
 - Delta Storage Index (B+Tree with a Dictionary)
 - Complex data structure with additional structures
 - Large foot print



Lock protect access

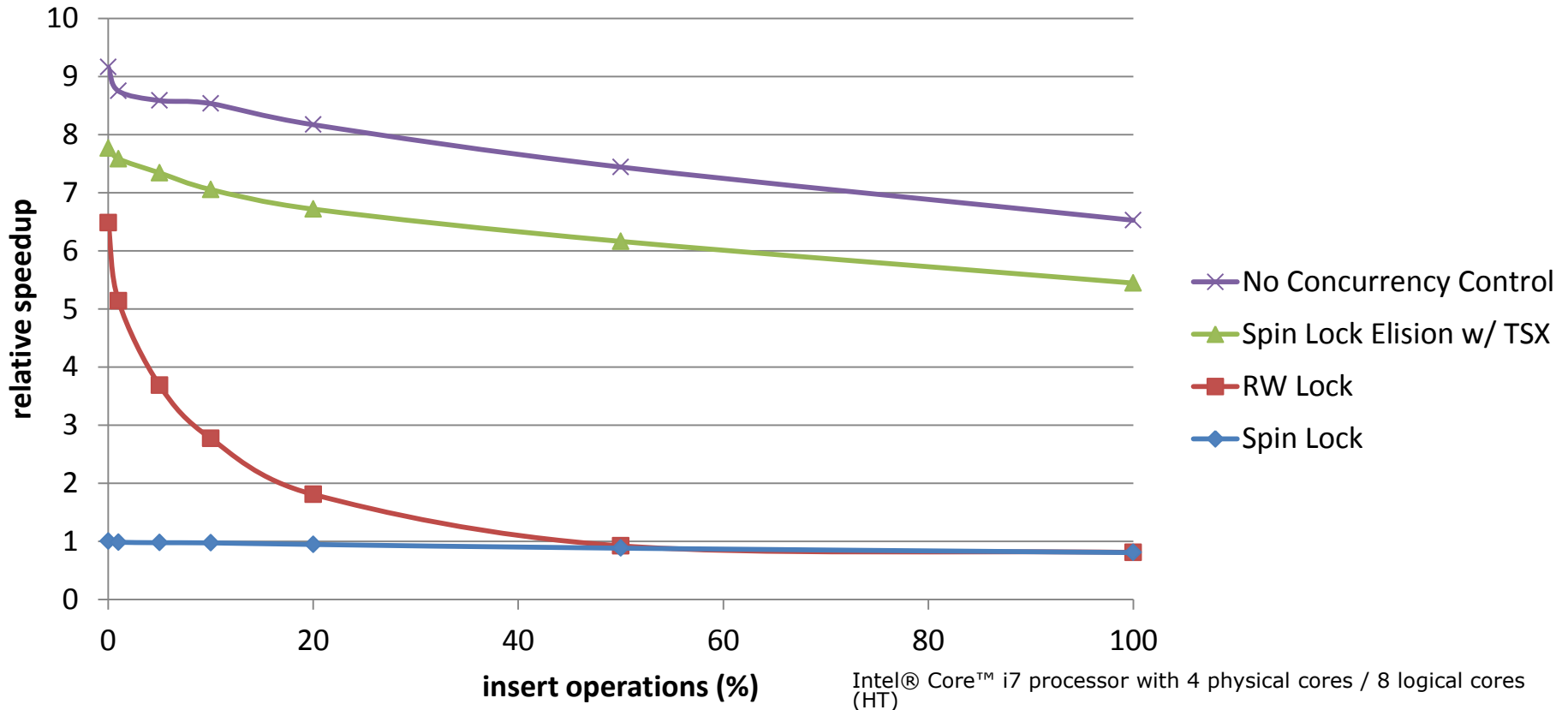
- Reader-Writer
- Spin Lock

Column Data	Dictionary
	ID Entry
1	0 Bakersfield
7	1 San Francisco
2	2 Santa Clara
3	3 San Jose
1	4 Fresno
0	5 Berkeley
8	6 Sacramento
2	7 Palo Alto
...	8 Oakland



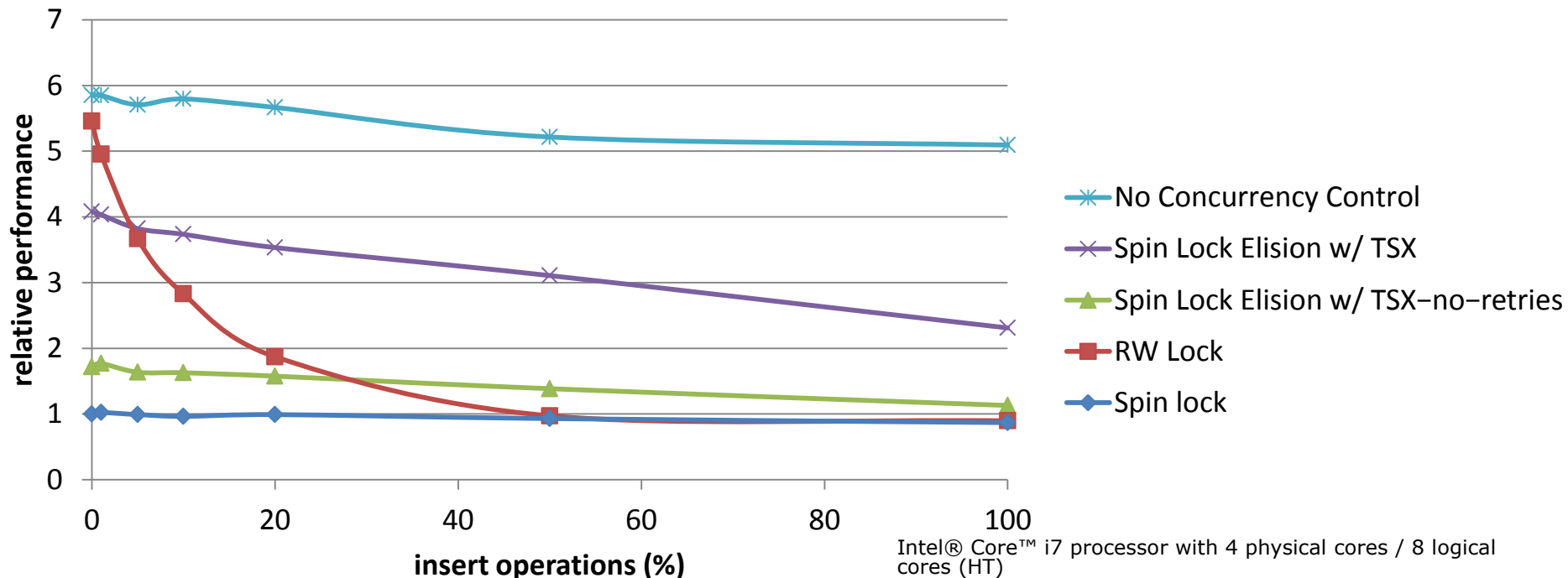
Initial Results: B+Tree

- Intel TSX provides significant gains with no application changes
 - Outperforms RW lock on read-only queries
 - Significant gains with increasing inserts (6x for 50%)



Initial Results: Delta Storage Index

- Intel TSX provides gains with no application changes
 - Different profile as compared to B+Tree
 - Spin lock w/ Intel TSX better than RW Lock when $> 5\%$ insert
 - Significant gap as compared to no concurrency control
- Baseline should implement good retry policy on aborts



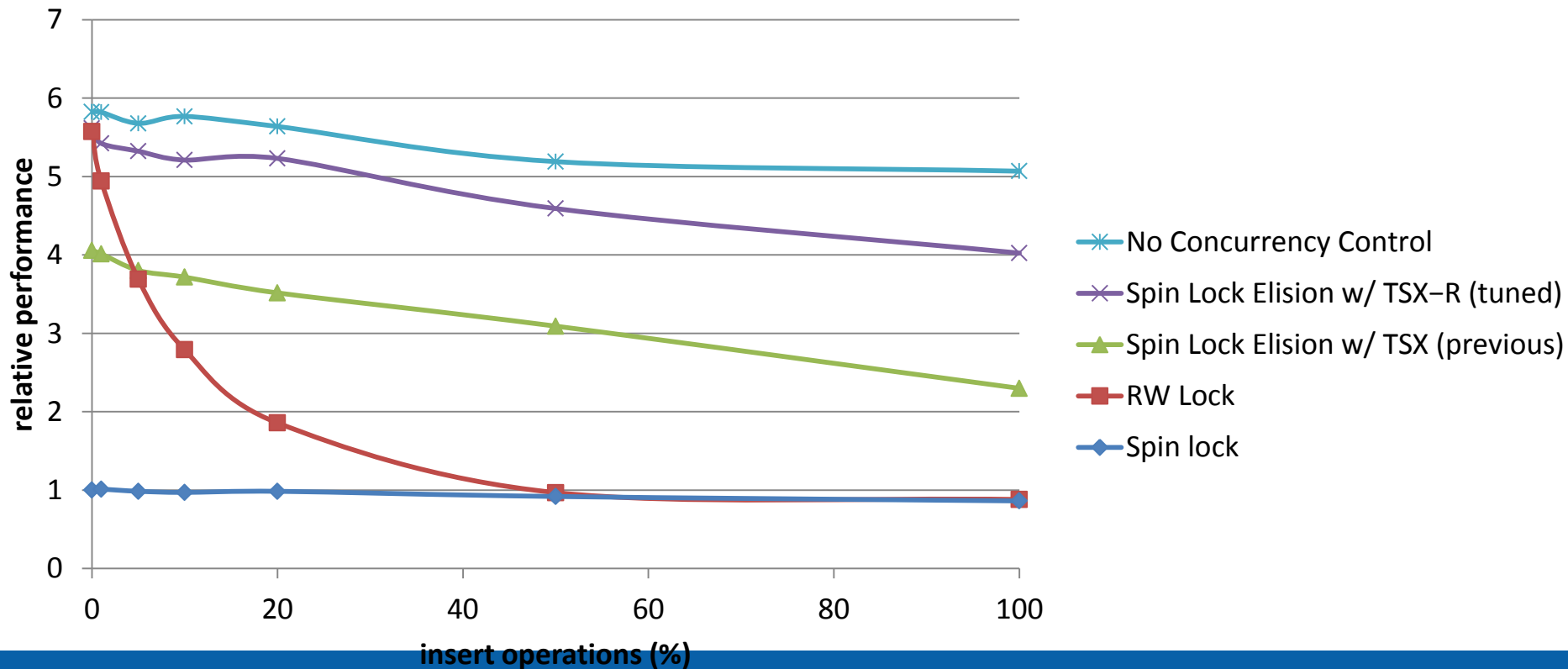
Software Transformations

- Capacity Aborts
 - Node/Leaf Search Scan
 - Causes $O(n)$ random lookups
 - Transformation – Binary Search
 - Causes $O(\log(n))$ random lookups
- Data Conflicts
 - Single dictionary
 - Global memory allocator
 - Transformation – Multiple Dictionaries, per-thread/core allocators

Well Known Transformations

Tuned Results: Delta Storage Index

- Intel TSX provides significant gains with transformations
 - Restores read-only query performance
 - Spin lock w/ Intel TSX significantly outperforms RW lock (5x for 50% inserts)
 - Close to 'No Concurrency Control'



4 way Intel Xeon E7 v3 w/wo TSX

Incremental performance gains in transactional processing when running SAP HANA on the Intel Xeon processor E7 v3 family with Intel® TSX enabled

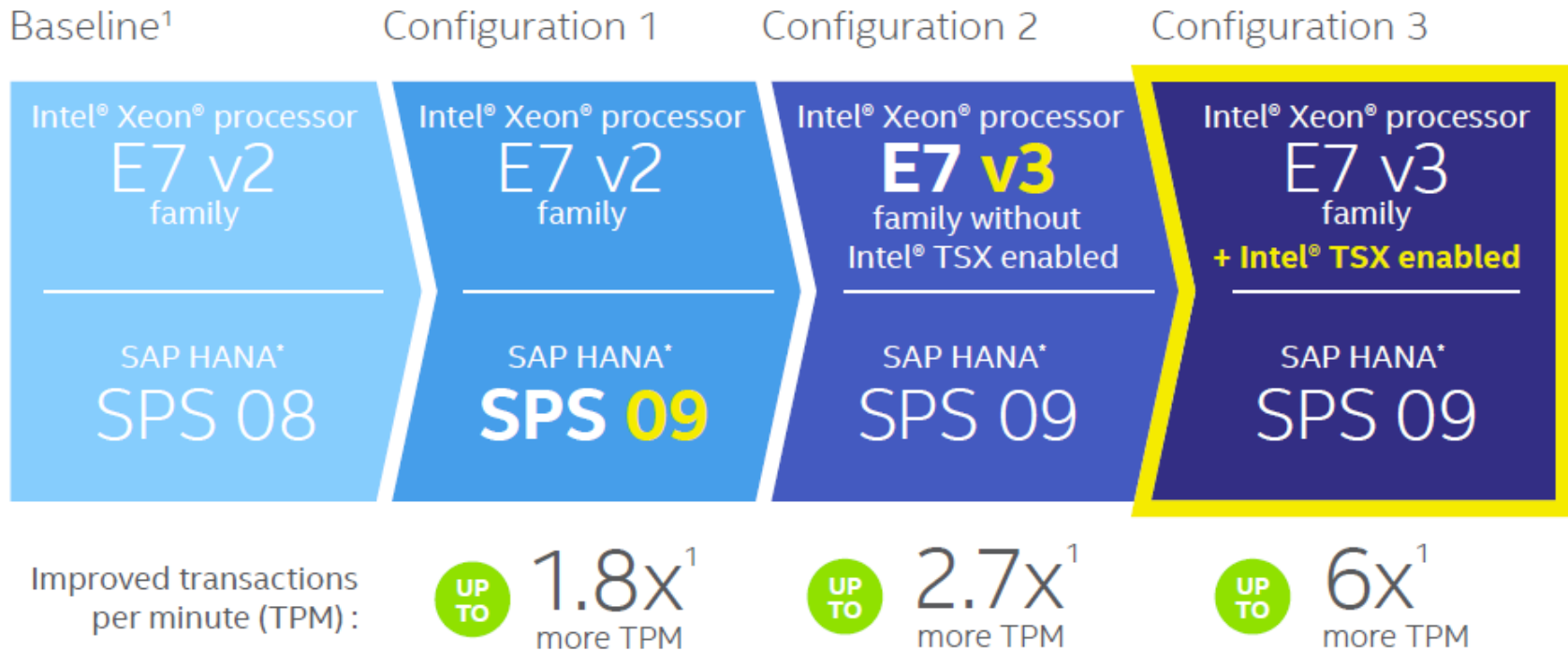


Figure 1. Upgrading to the Intel® Xeon® processor E7 v3 family and SAP HANA® SPS 09 (S-OLTP stress test lab results) provides incremental performance gains.¹

TUM HyPer

- Breakup DB Txn
 - Small HTM txn
- HTM Txn
 - Sync access to DS
- Use timestamp to “commit” DB Txn

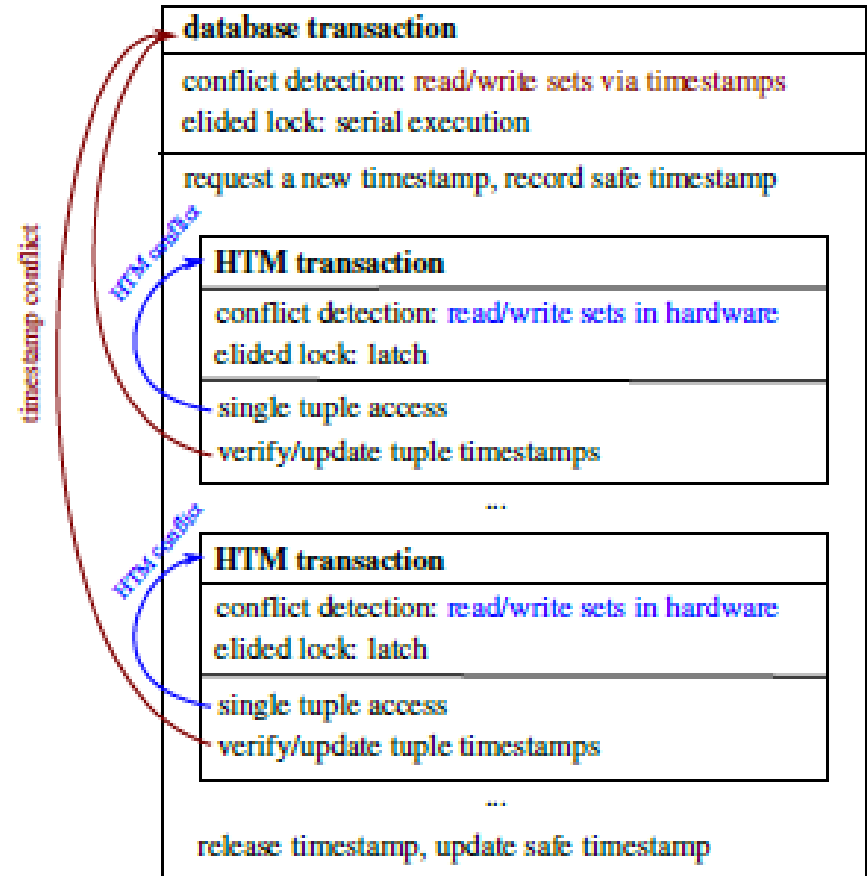


Figure 7. Transforming database transactions into HTM transactions

TUM HyPer Result – 2 way Xeon EP

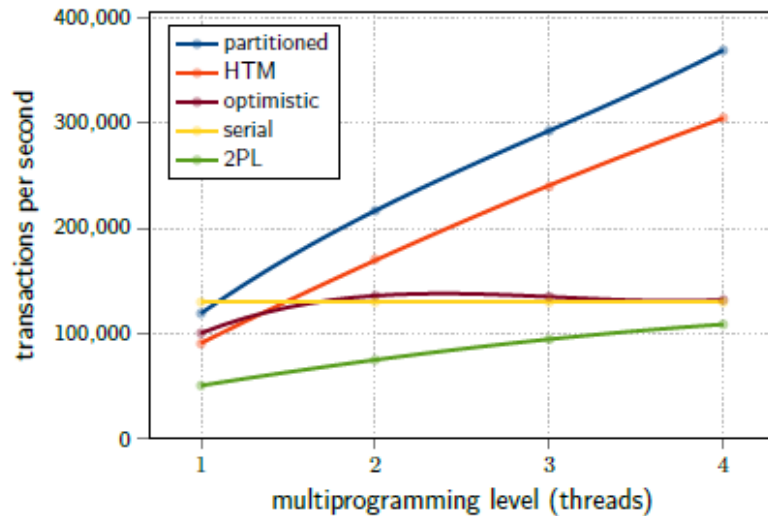


Figure 18. Scalability of TPC-C on desktop system

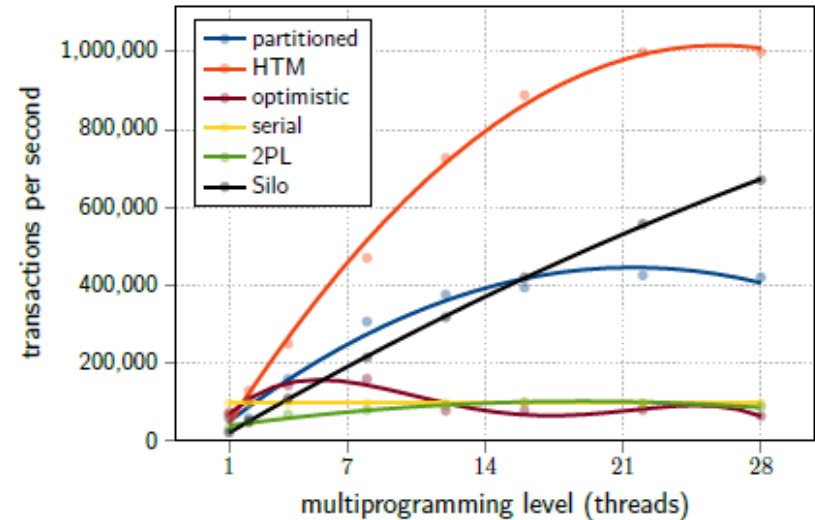


Figure 20. Scalability of TPC-C on server system

TM Programming Model (C++TM)

- Is this a research toy?
 - No – not even a toy as few play with it
 - Take this out of the glass cage, and play with it
 - Should we ban or boycott STAMP as workload ;-)
- Did not address issues raised in 2005
 - Conditional synchronization
 - Open and/or closed nesting
 - Escape actions
 - Inter-operate with other paradigms, e.g. locks
- Is the current set sufficient?
 - Need broad usage experience
 - Does this limit holistic performance?
- New issue – TM and persistent memory

Better support for critical section?

- Even C++'11 is not good enough
- Tight definition of critical section (or sync block)
 - Not just a coding convention
 - Enable efficient application of lock elision
 - Enable other transformations, like Hybrid Lock Elision
- How about adding lock declaration to C++TM synchronization block?
 - Semi-automatic code refactoring needed
 - Could be stepping stone to transactions
- Do we need cleaner threading library?
 - Pthread has high overhead