Applying HTM to an OLTP System: No Free Lunch

David Cervini, *Danica Porobic*, Pınar Tözün, Anastasia Ailamaki

DATA-INTENSIVE APPLICATIONS AND SYSTEMS

Why Hardware Transactional Memory?

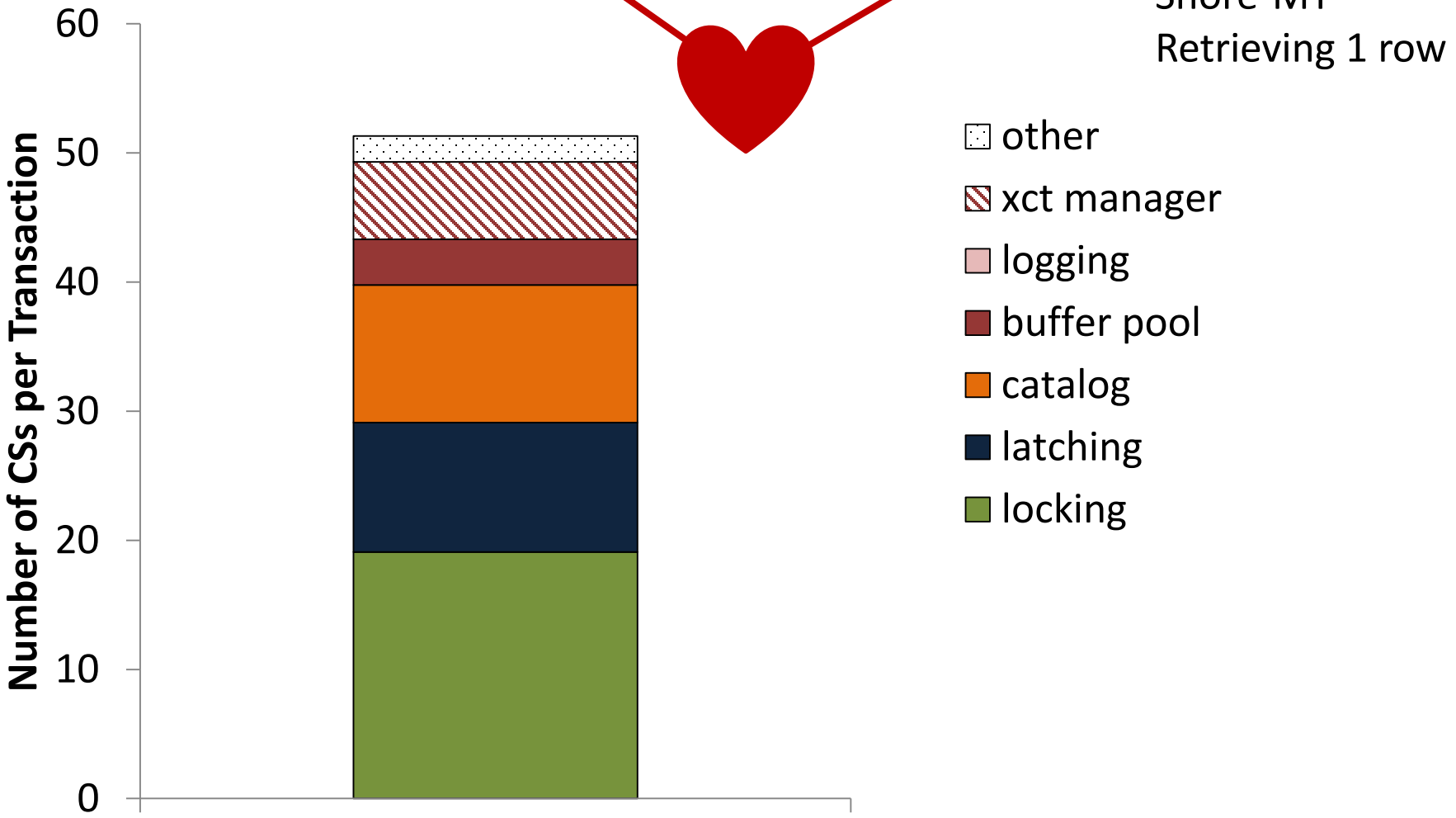
- Multicores are here to stay
- Lock-based and lock-free programming is hard
- Transactional memory should ease programming
- Software transactional memory is not fast enough

transaction {
$a = a - 10;$
$b = b + 10;$
}
Transaction 1

transaction {
$c = c - 20;$
$a = a + 20;$
}
Transaction 2

Very promising for synchronization-heavy software

Why HTM and OLTP?



Many critical sections even for a simple transaction

A match made in heaven?

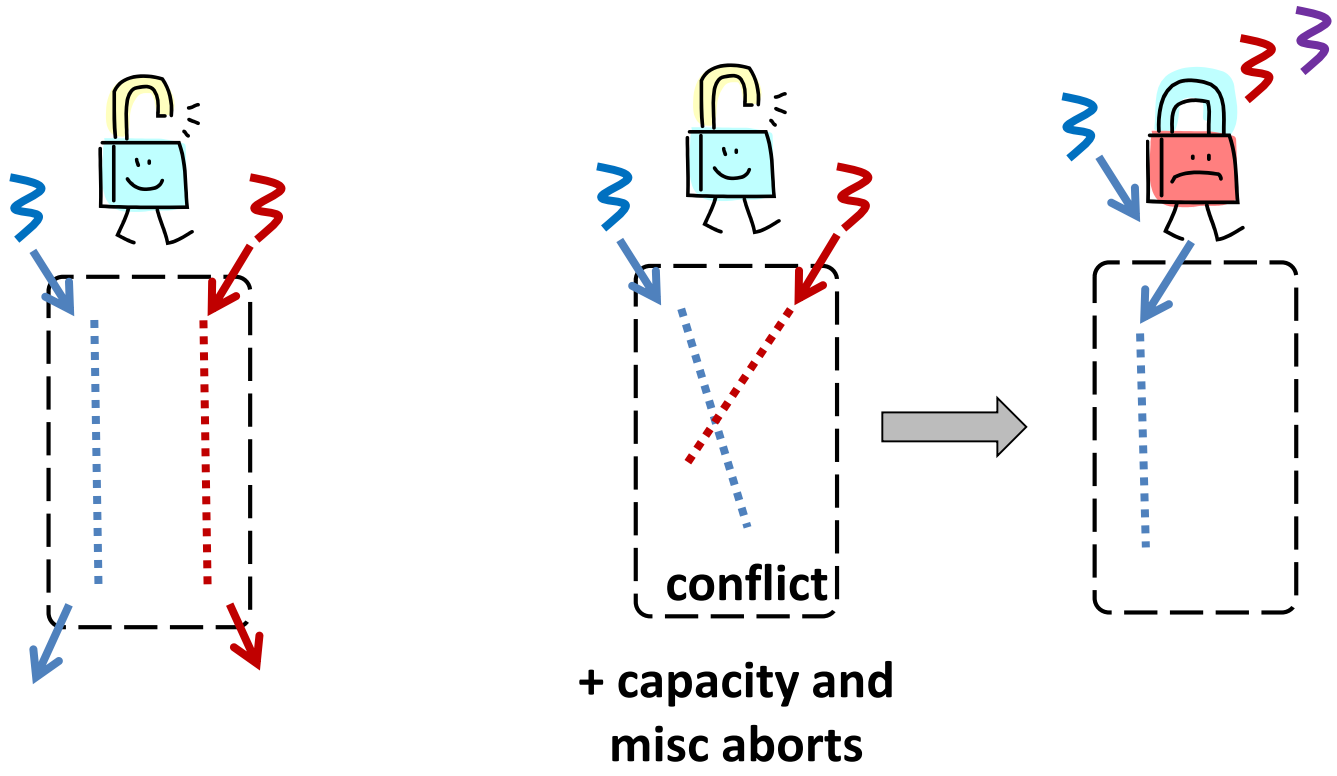
Promise:

- HTM simplifies lock-free programming
- Shore-MT relies on fine-grained locking
- Expect performance improvement

Transactional Synchronization eXtensions

- On Intel's Haswell
- **RTM** (Restricted Transactional Memory)**:**
  - Directly uses TM, more flexible
  - _xbegin, _xabort, _xend, _xtest
  - Requires new implementation
- **HLE** (Hardware Lock Elision):
  - Speculative execution of existing locking code
  - __ATOMIC_HLE_ACQUIRE or __ATOMIC_HLE_RELEASE

TSX in a nutshell

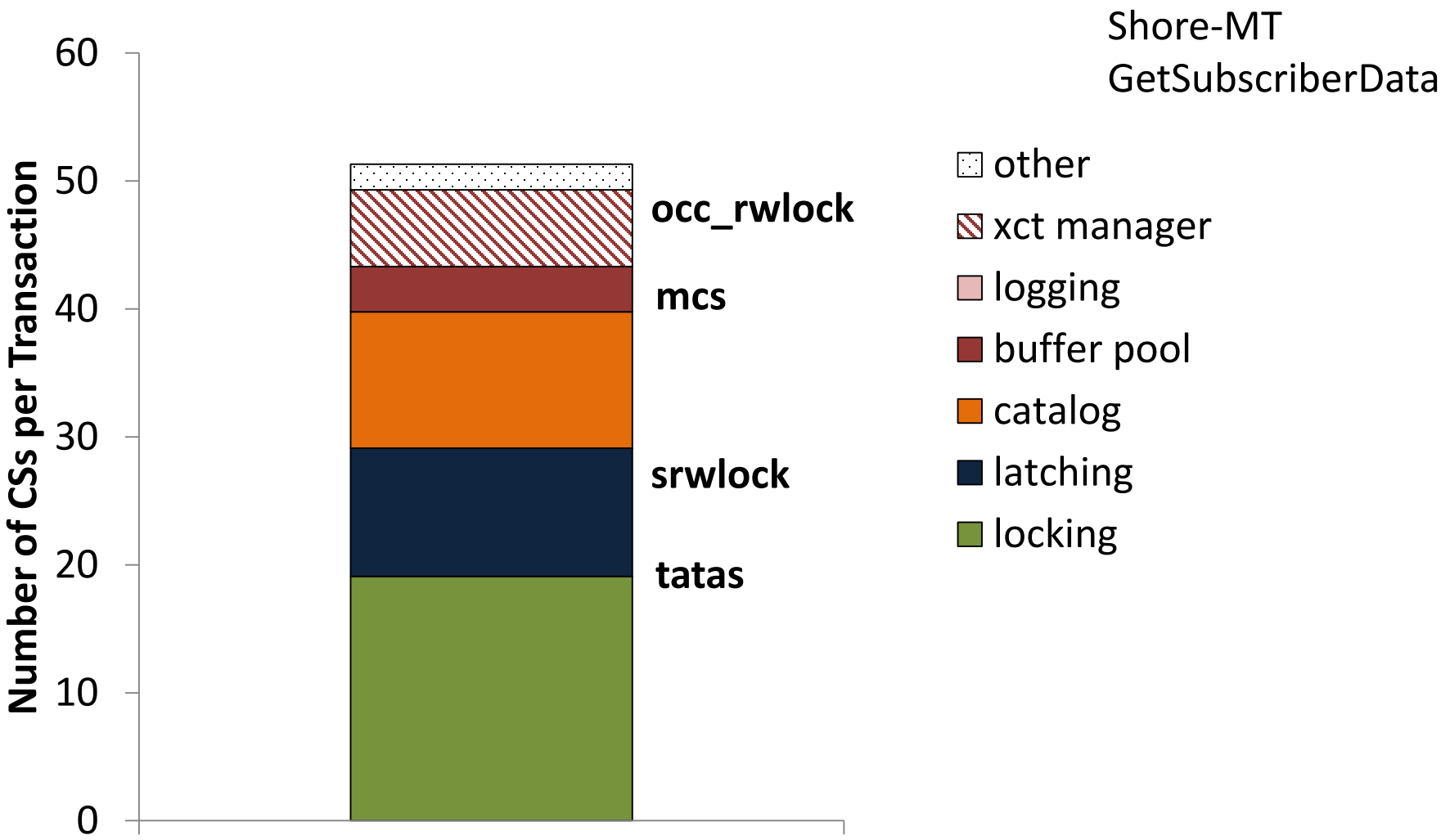


- Uses cache coherency of L1 cache
- Tracks data at cache line granularity

Experimental platform

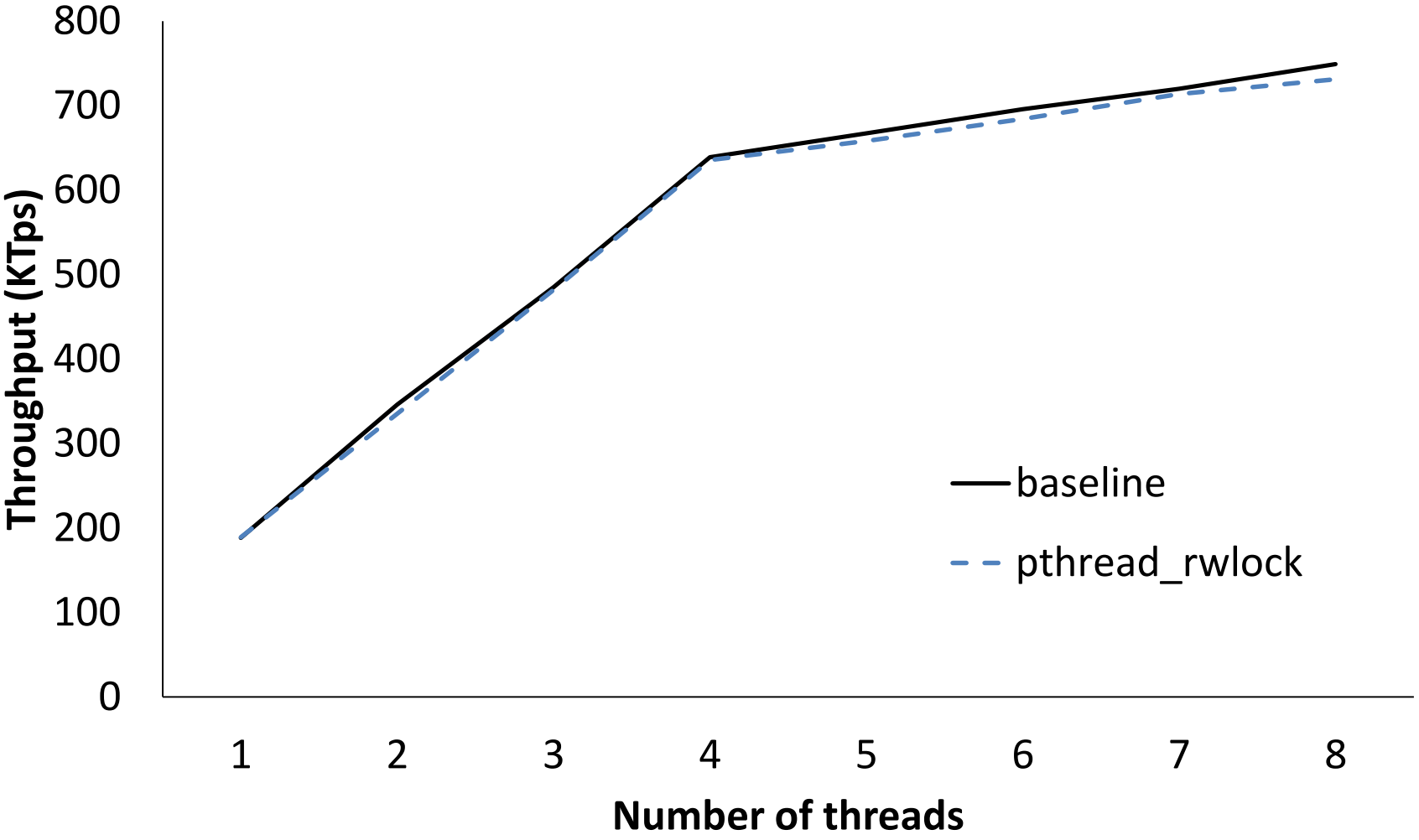
- Software:
  - Shore-MT
  - TM-1 benchmark: GetSubscriberData
  - From 1 to 8 workers
  - SLI enabled
  - 80000 row dataset
- Hardware:
  - Intel i7-4770 3.4Ghz 4-core processor, hyperthreading on
  - 16GB RAM

Which lock types are used?



Many lock types for different use-cases

HLE pthread instead of occ_rwlock



No impact: pthread implementation is limited

RTM-enabled lock example: acquire

```
void occ_rwlock::acquire() {
#ifdef OCC_RWLOCK_RTM_WRAPPER
  unsigned int status;
  for(int i = 0; i < 2; i++) {
    if ((status = _xbegin()) == _XBEGIN_STARTED) {
      if (has_reader()) { _xabort(0xff); }
    return;
    } else if ((status & _XABORT_EXPLICIT) && _XABORT_CODE(status) == 0xff) {
      while (__atomic_load_n(&_active_count, __ATOMIC_ACQUIRE))
        _mm_pause();
    } else if (status & _XABORT_CONFLICT) {
      long int backoff=10*random()/RAND_MAX;
      while (backoff--) _mm_pause();
    } else if (status & _XABORT_RETRY) {
       _mm_pause();
    } else {
      break;
    }
  }
#endif
/**original acquire code here**/
```

Retry transaction multiple times

Tune retry policies

Avoid Lemming Effect

RTM preferable for new code for flexibility

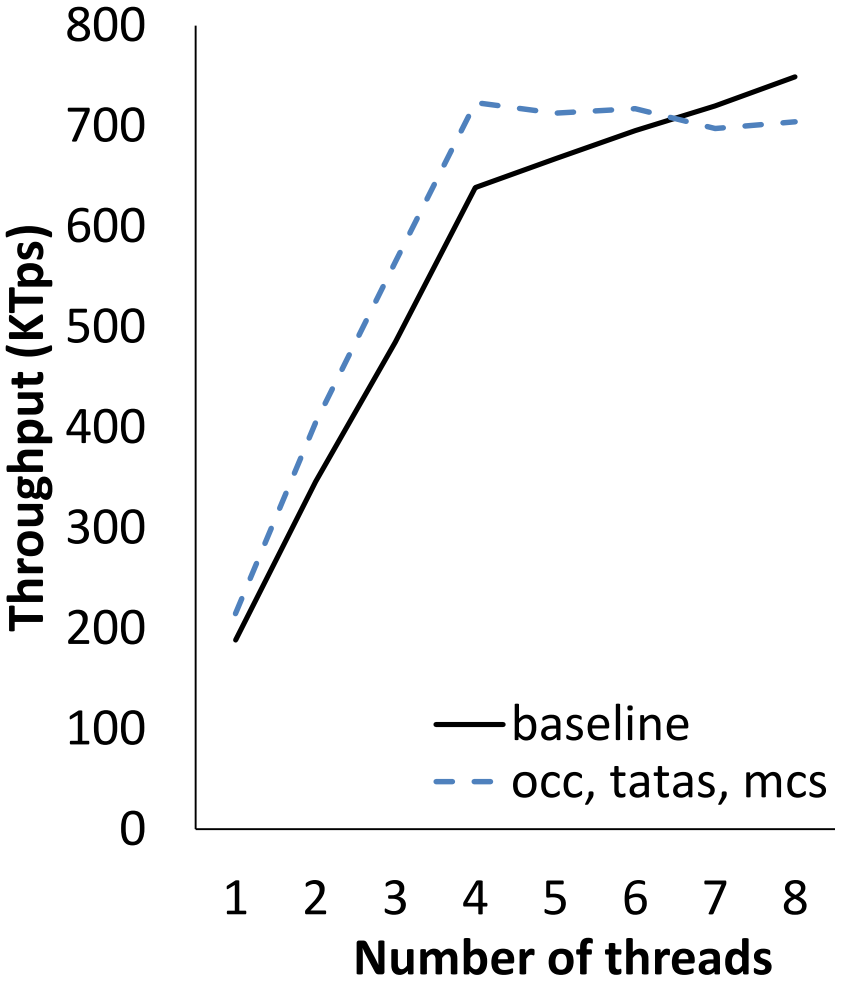
RTM-enabled lock example: release

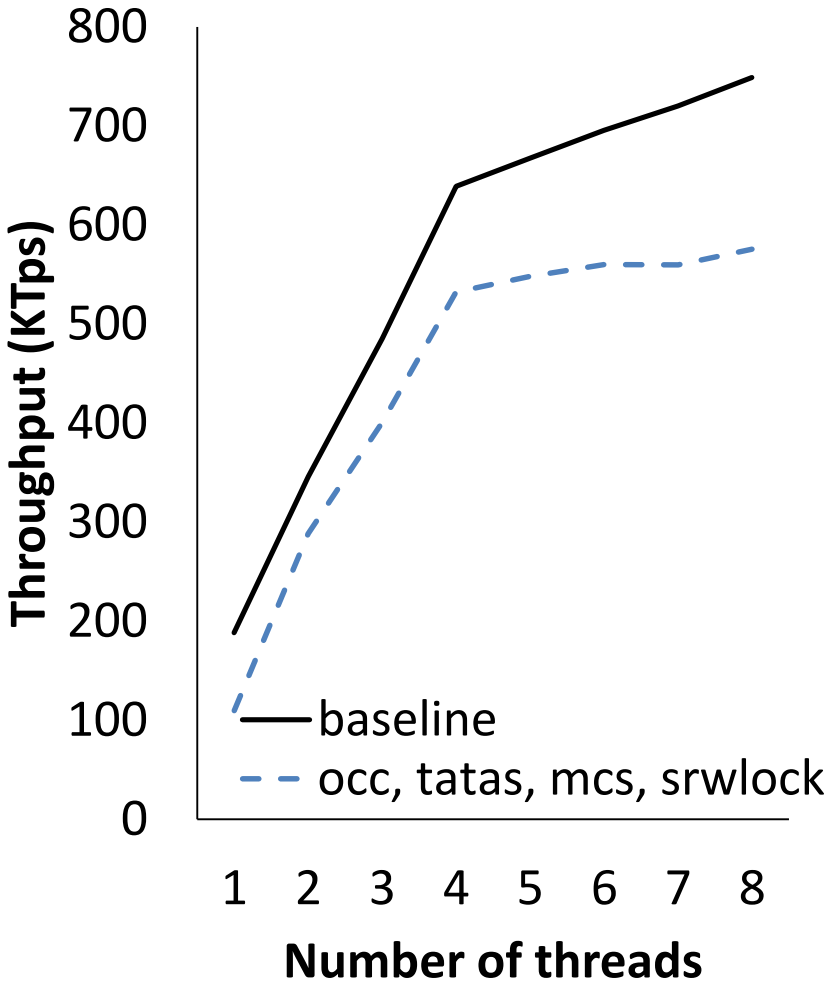
```
void occ_rwlock::release() {
#ifdef OCC_RWLOCK_RTM_WRAPPER
  if (!has_reader() && _xtest()) {
    _xend();
    return;
  }
#endif
/**original release code here**/
}
```

Must be inside a transaction

Ending a transaction requires no tuning

RTM locks: good & bad news

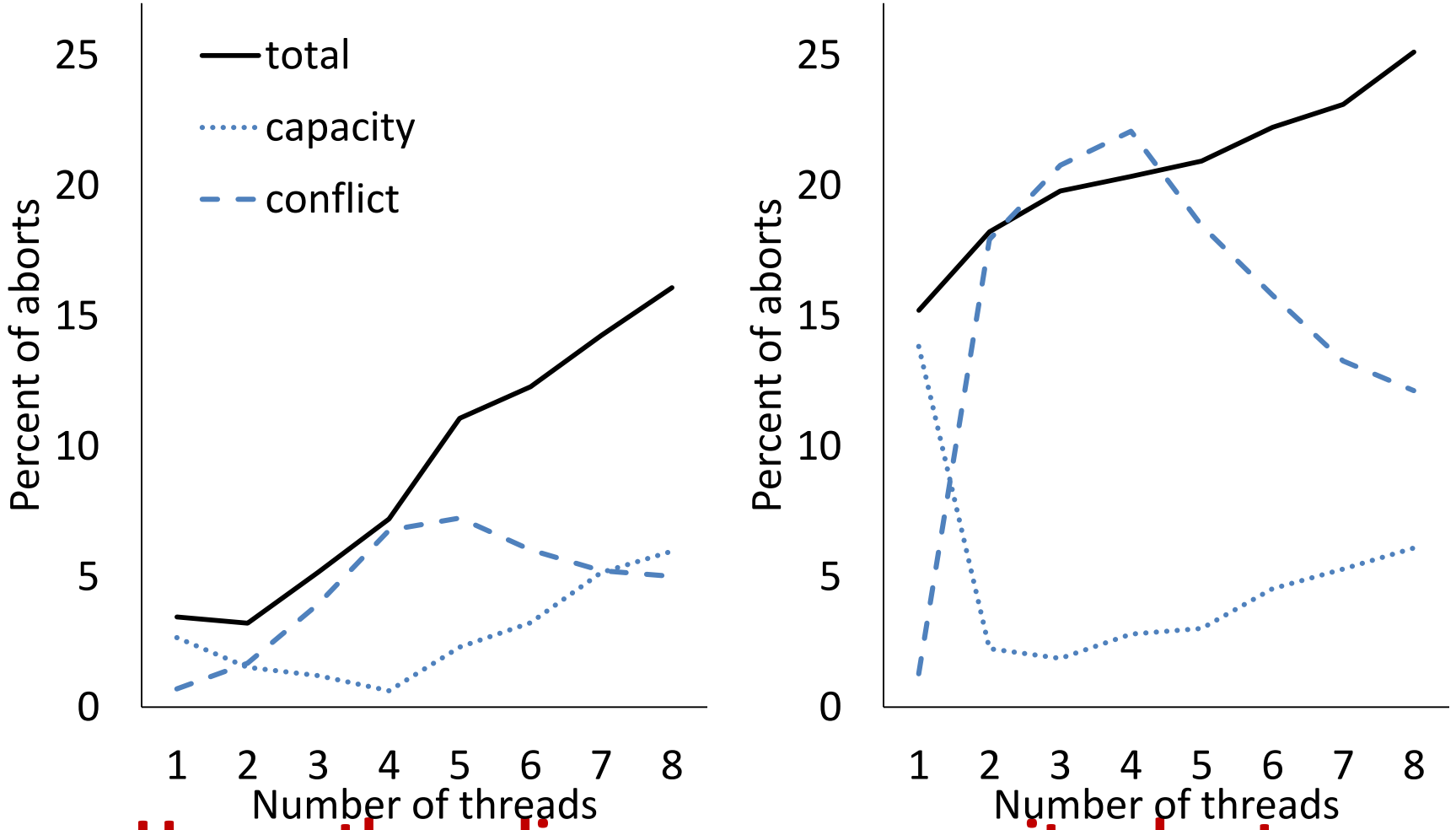




HTM improves throughput 13-18%

Improvement is not guaranteed

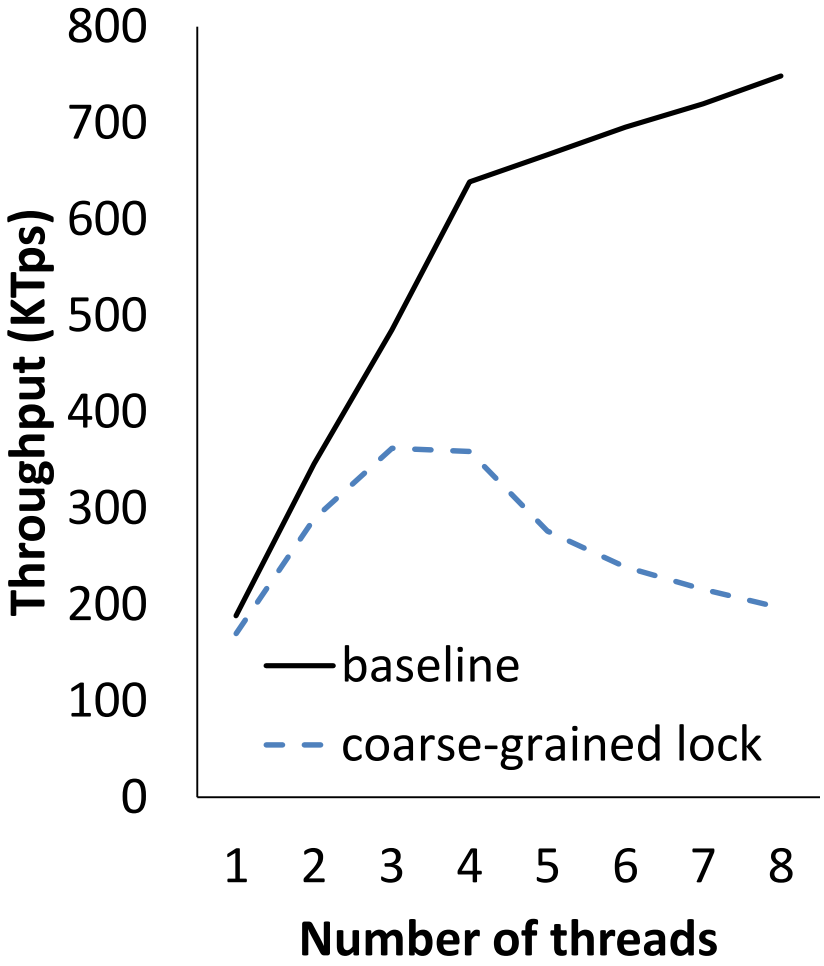
Reason: aborts

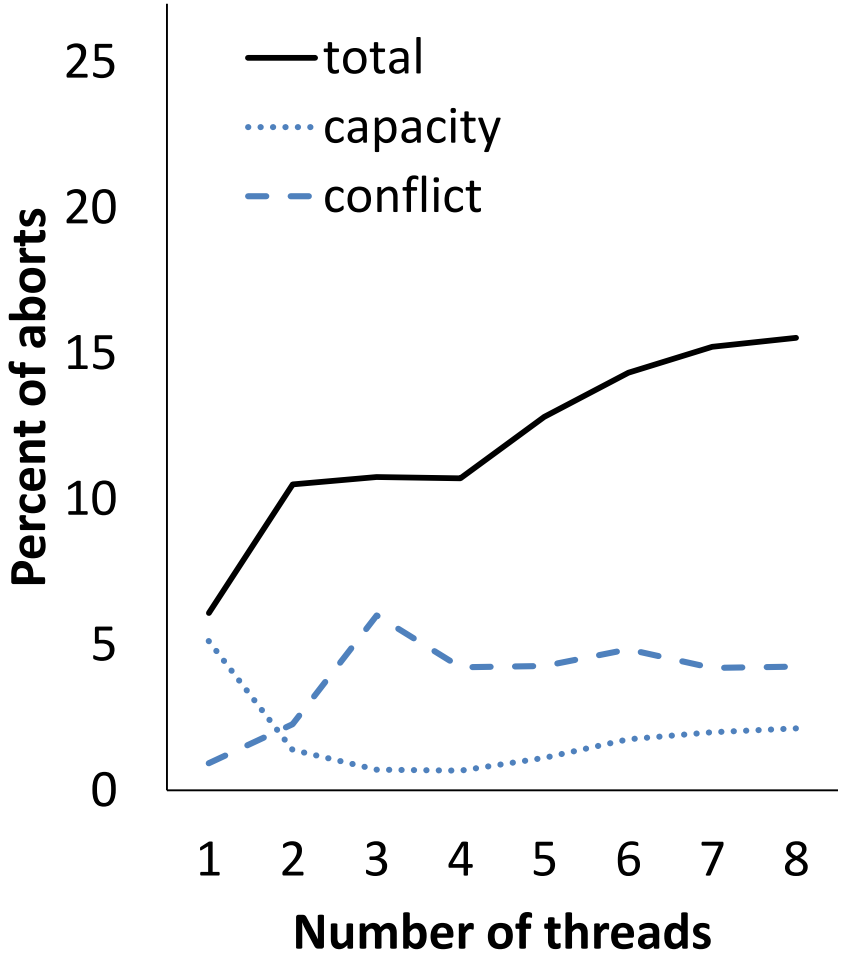


Hyper-threading causes capacity aborts

Real data conflicts cause high abort rates

Coarse grained B-tree lock





Throughput drops by up to 73%

Large critical section make aborts very expensive

Applying HTM to an OLTP system

- Promise
  - TSX democratizes lock-free programming
  - Shore-MT relies on fine-grained locking
  - Possible match made in heaven

- Reality
  - Low hanging fruit: TSX is great for short critical sections
  - Requires tuning – not always beneficial
  - Cannot be used for large code sections
  - Realizing full benefits requires system redesign

Thank you!