

# Time-Based Transactional Memory with Scalable Time Bases

**Torvald Riegel** (Dresden University of Technology, Germany)

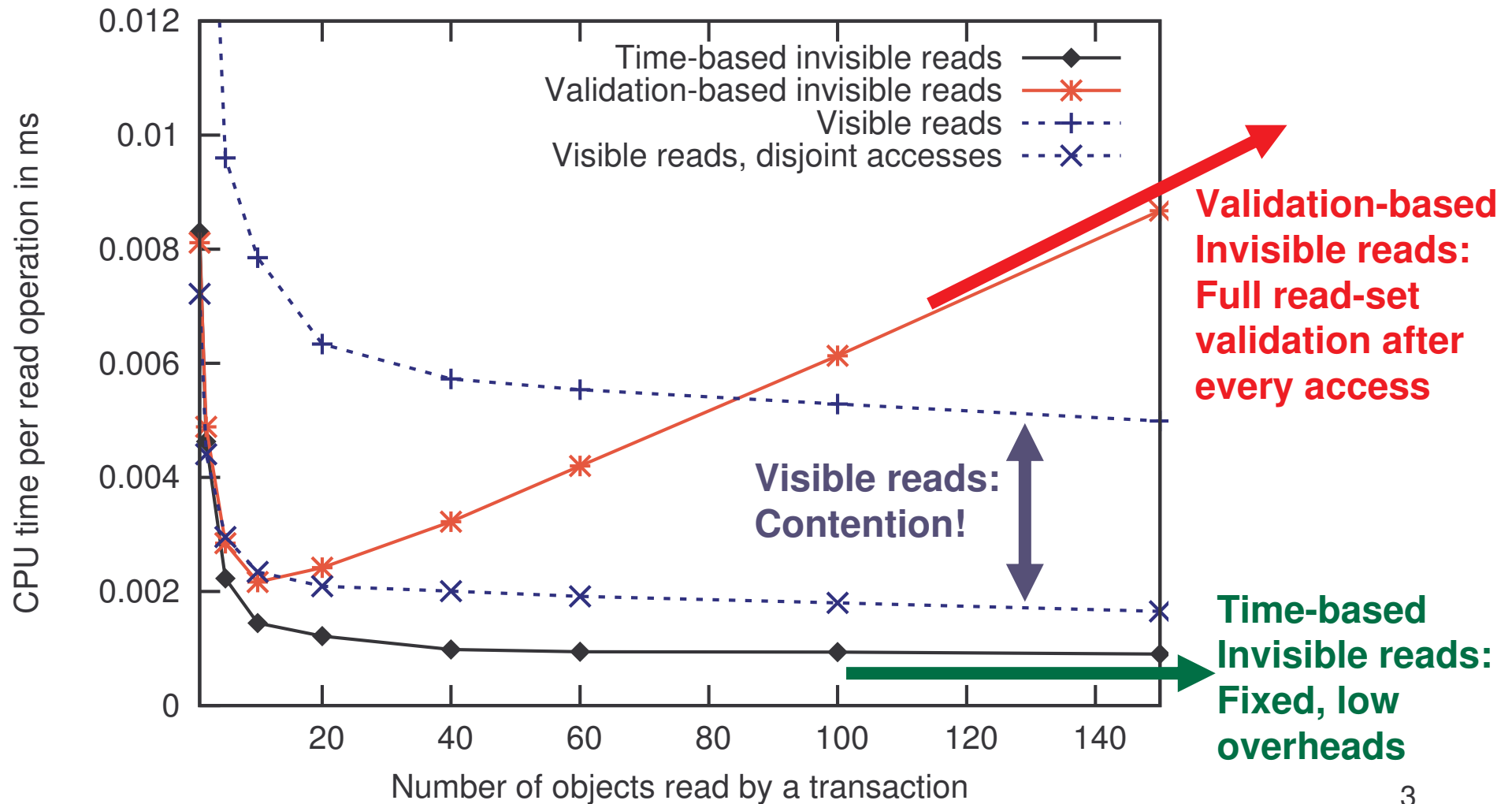
**Christof Fetzer** (Dresden University of Technology, Germany)

**Pascal Felber** (University of Neuchatel, Switzerland)

# Problem Statement and Agenda

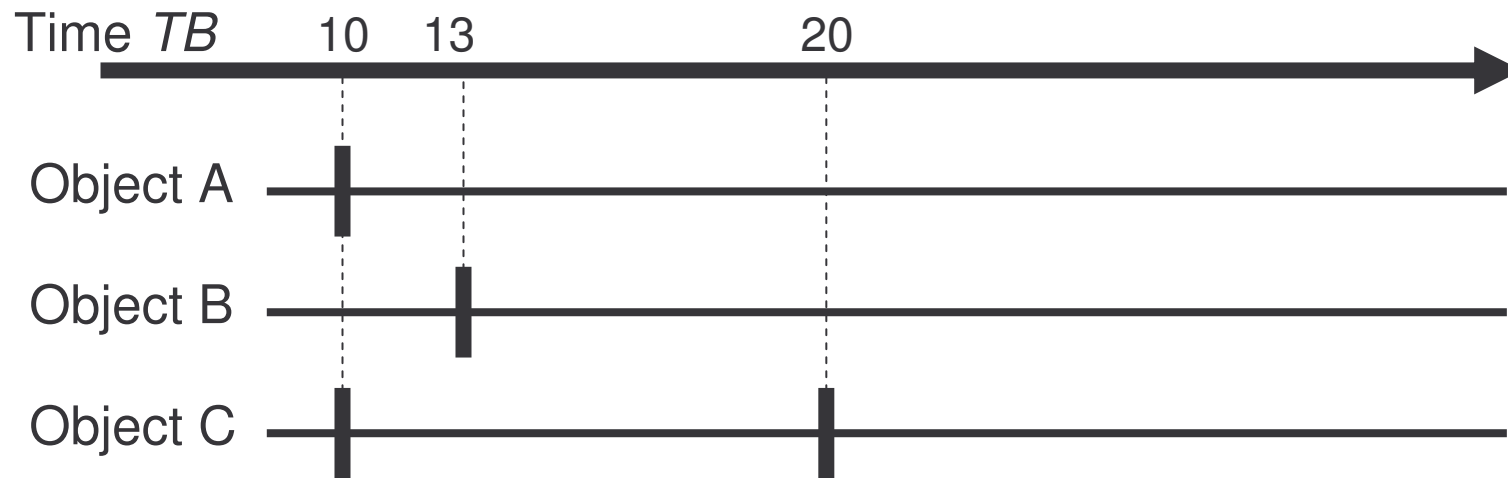
- Overhead of read operations has high impact on Transactional Memory (TM) performance
- Time-based Transactional Memories:
  - Avoid overheads of both visible reads and validation-based invisible reads
  - Global time base required: potential sequential bottleneck
- Contribution: How to use scalable time bases: Synchronized clocks
- Performance: Case study

# Visible vs. Invisible vs. Time-Based Reads



# Time-based Transactional Memory

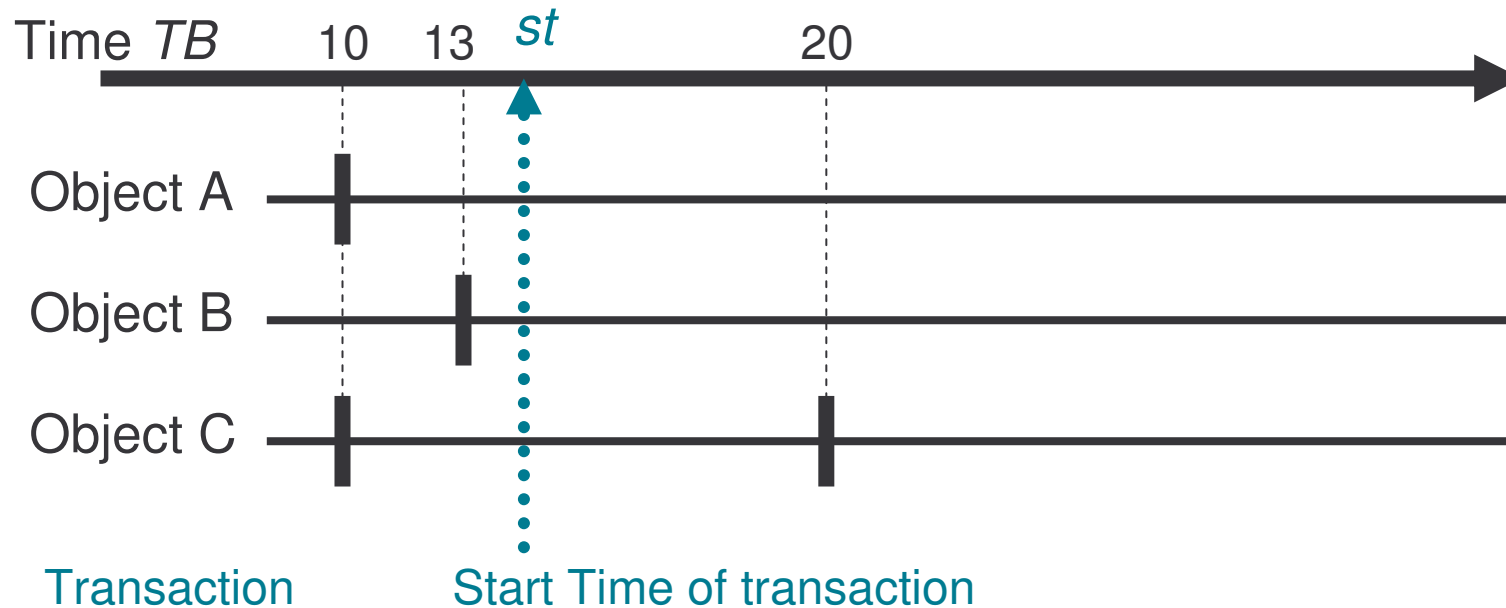
- Global time base  $TB$
- Time-stamped object versions



Transaction

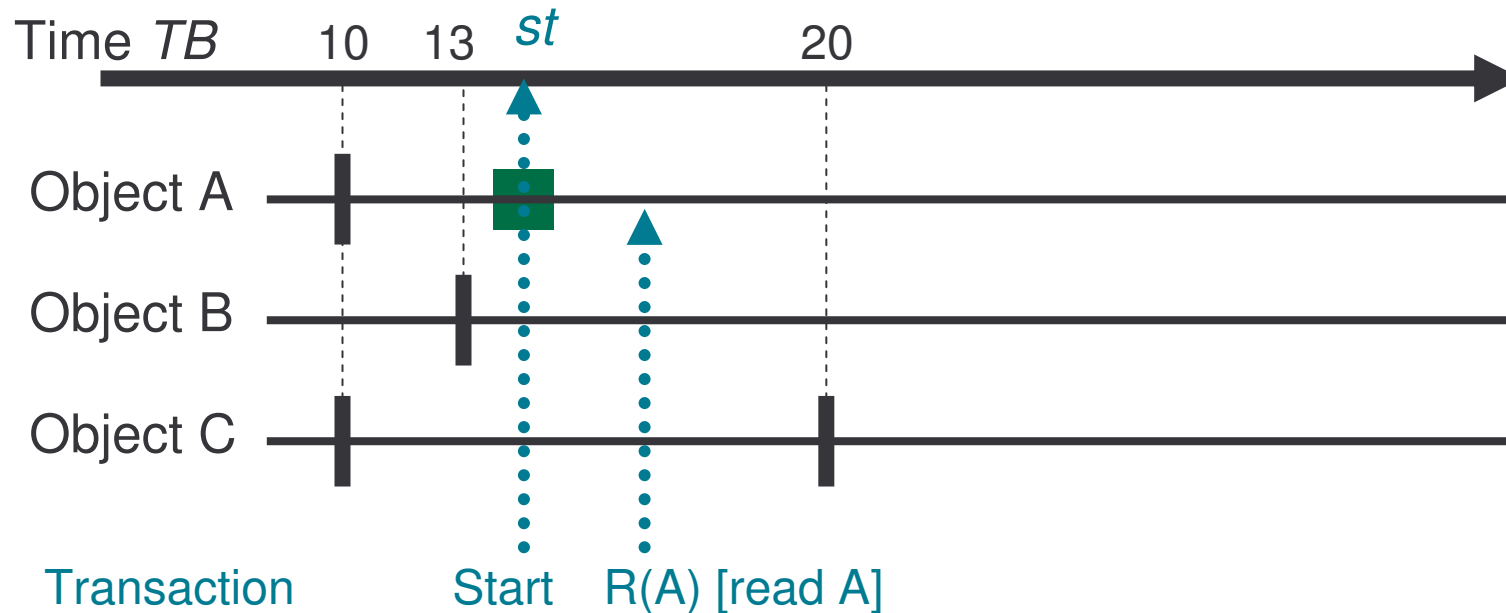
# Time-based Transactional Memory

- Global time base  $TB$
- Time-stamped object versions
- Consistent **snapshots** at time  $st$ : only read versions most recent at  $st$



# Time-based Transactional Memory

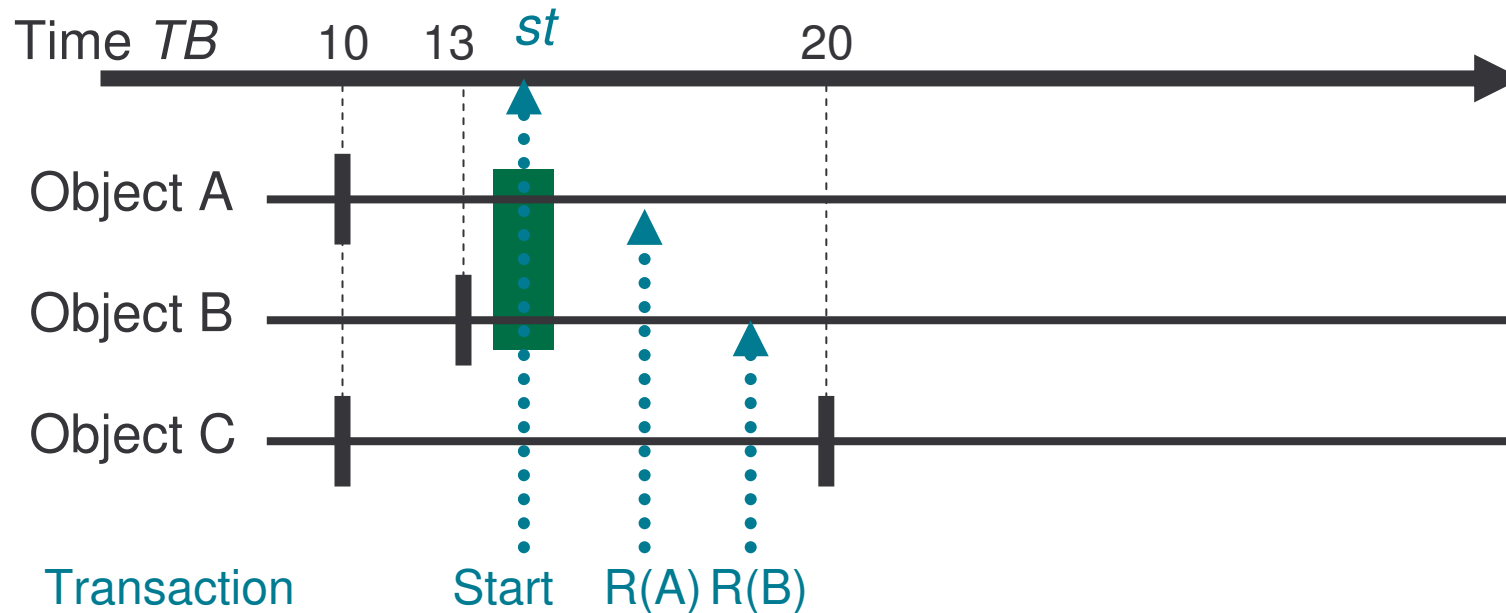
- Global time base  $TB$
- Time-stamped object versions
- Consistent **snapshots** at time  $st$ : only read versions most recent at  $st$



(c) Torvald Riegel

# Time-based Transactional Memory

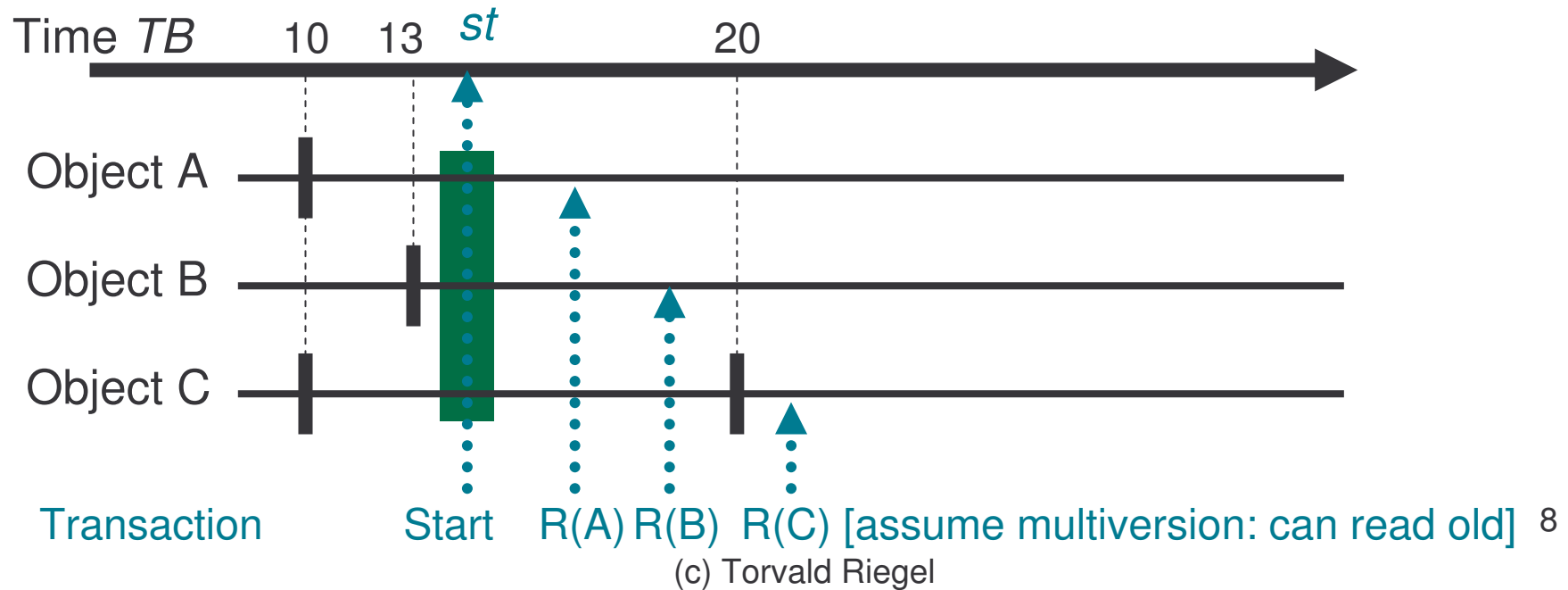
- Global time base  $TB$
- Time-stamped object versions
- Consistent **snapshots** at time  $st$ : only read versions most recent at  $st$



(c) Torvald Riegel

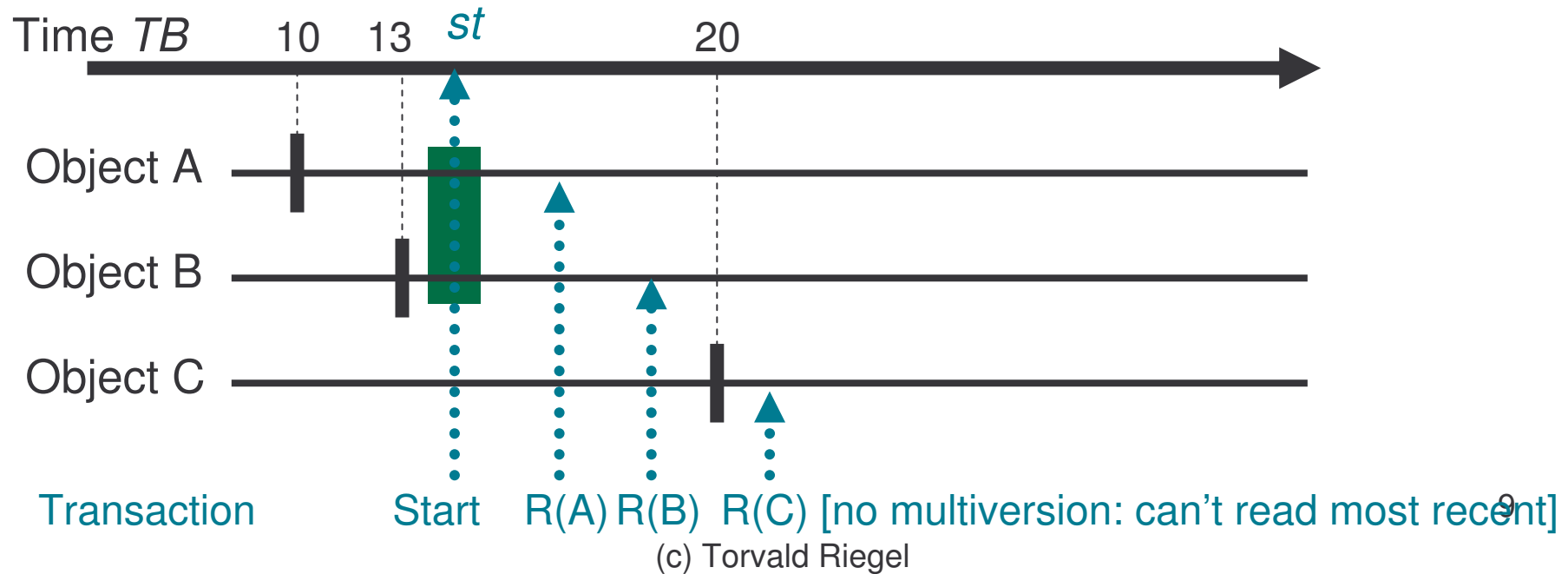
# Time-based Transactional Memory

- Global time base  $TB$
- Time-stamped object versions
- Consistent **snapshots** at time  $st$ : only read versions most recent at  $st$



# Time-based Transactional Memory

- Global time base  $TB$
- Time-stamped object versions
- Consistent **snapshots** at time  $st$ : only read versions most recent at  $st$



# Time-based Transactional Memory

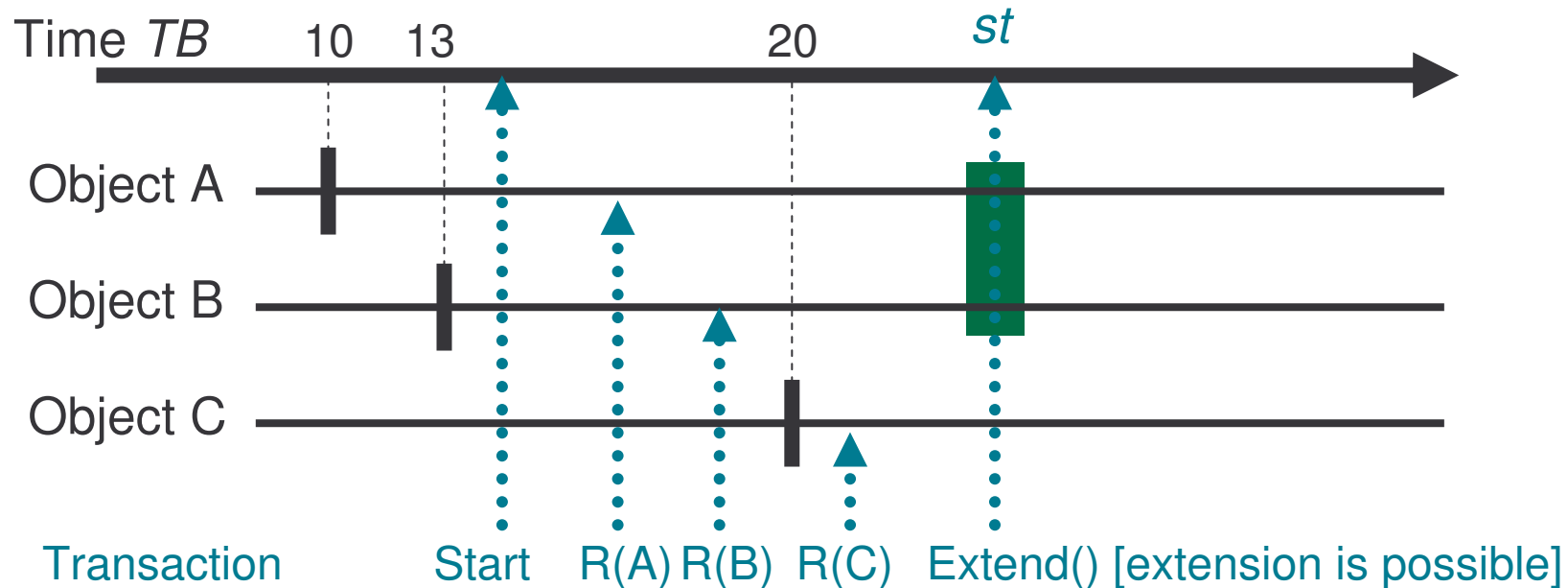
- Global time base  $TB$
- Time-stamped object versions
- Consistent **snapshots** at time  $st$ : only read versions most recent at  $st$
- Extension: check if snapshot is consistent at more recent time



(c) Torvald Riegel

# Time-based Transactional Memory

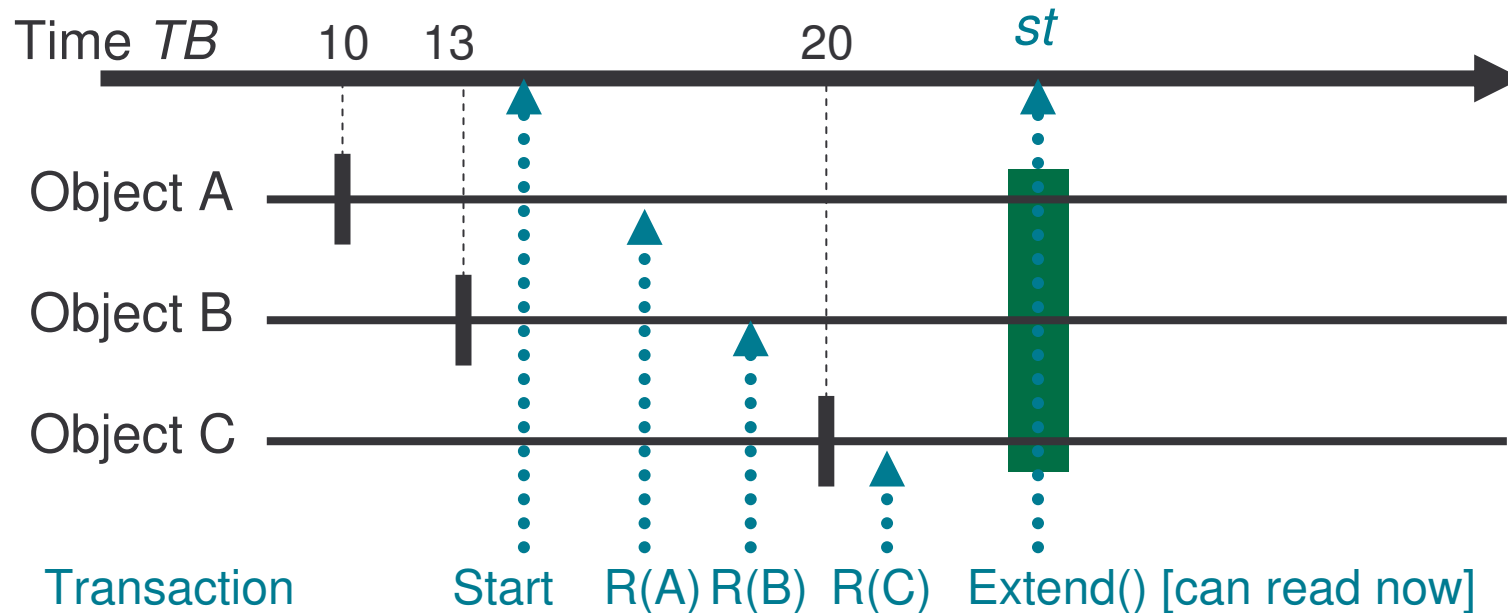
- Global time base  $TB$
- Time-stamped object versions
- Consistent **snapshots** at time  $st$ : only read versions most recent at  $st$
- Extension: check if snapshot is consistent at more recent time



(c) Torvald Riegel

# Time-based Transactional Memory

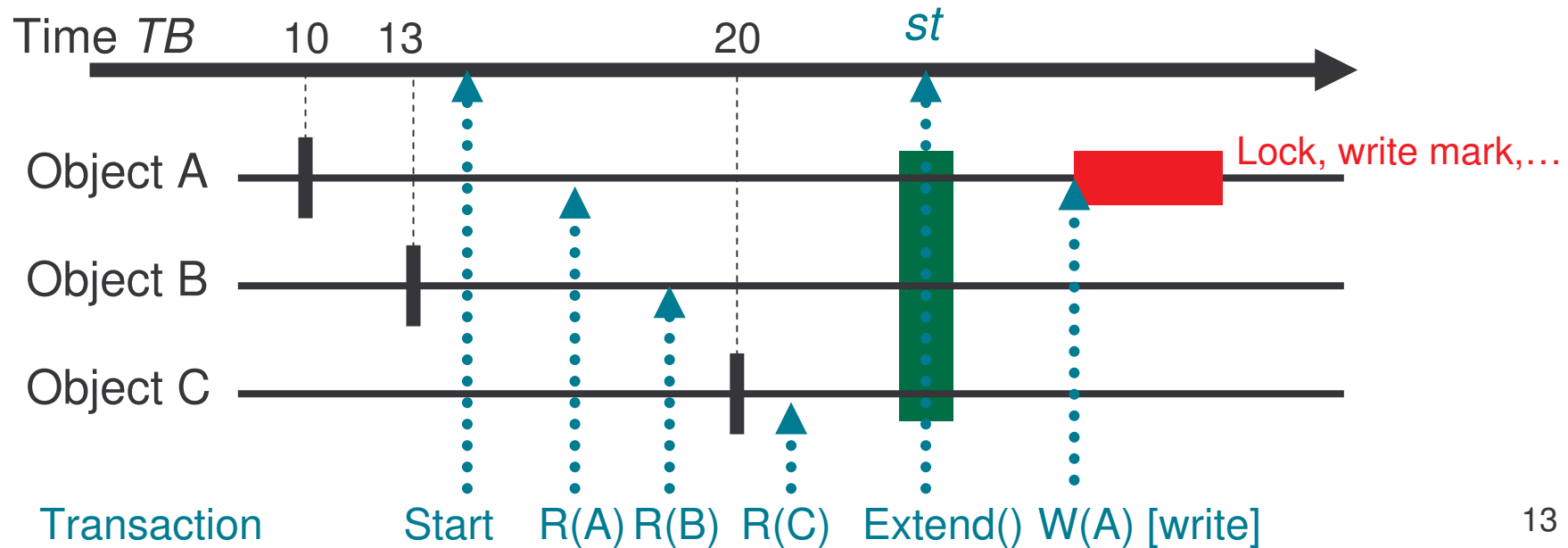
- Global time base  $TB$
- Time-stamped object versions
- Consistent **snapshots** at time  $st$ : only read versions most recent at  $st$
- Extension: check if snapshot is consistent at more recent time



(c) Torvald Riegel

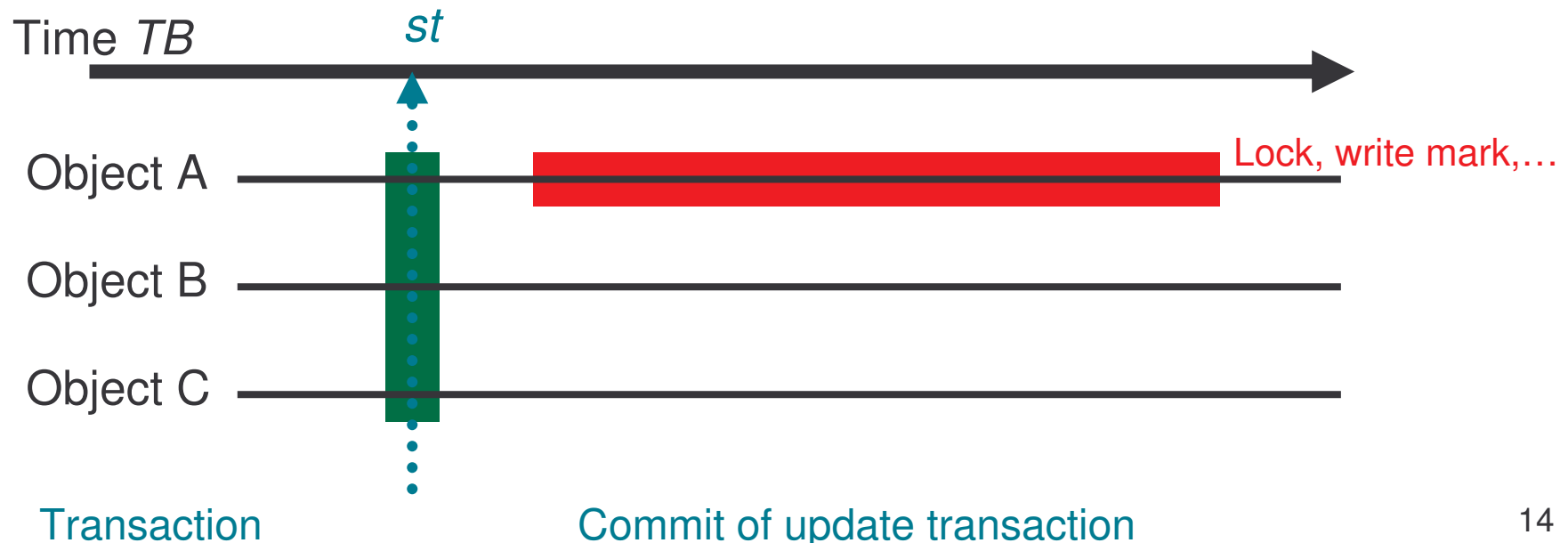
# Time-based Transactional Memory

- Global time base  $TB$
- Time-stamped object versions
- Consistent **snapshots** at time  $st$ : only read versions most recent at  $st$
- Extension: check if snapshot is consistent at more recent time



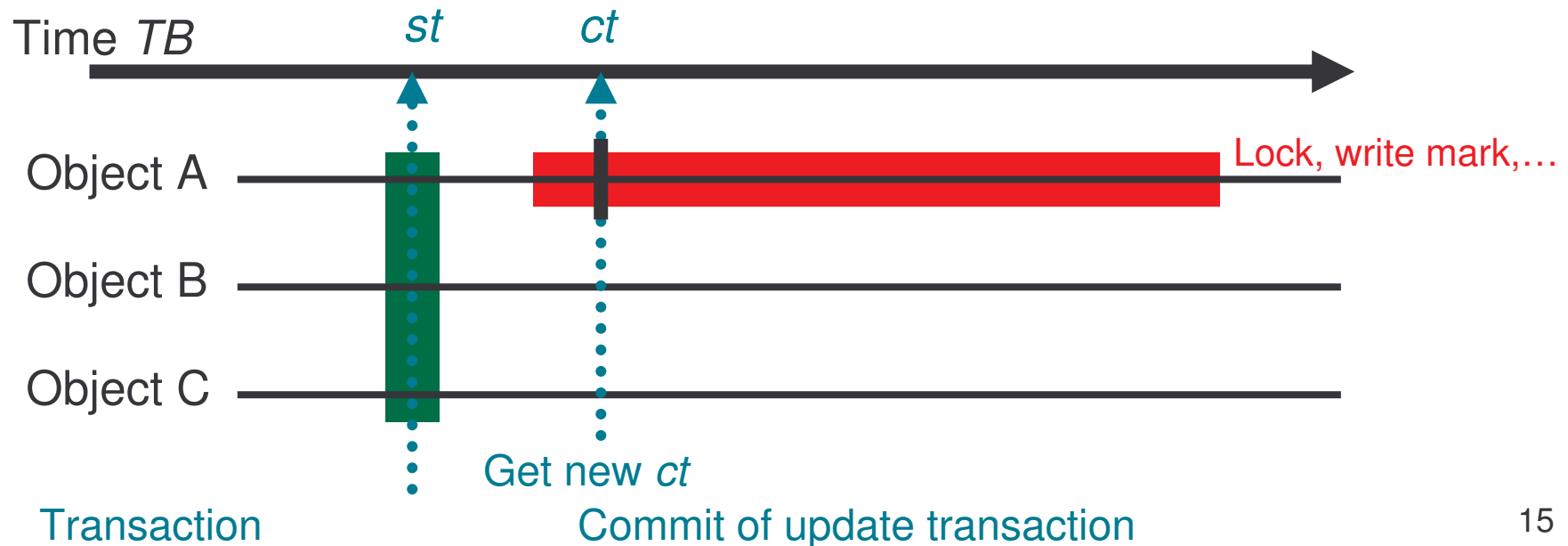
# Time-based Transactional Memory

- Global time base  $TB$
- Time-stamped object versions
- Consistent **snapshots** at time  $st$ : only read versions most recent at  $st$
- Extension: check if snapshot is consistent at more recent time
- Commit of update txn: get new commit time  $ct$  from  $TB$ , extend to  $ct$



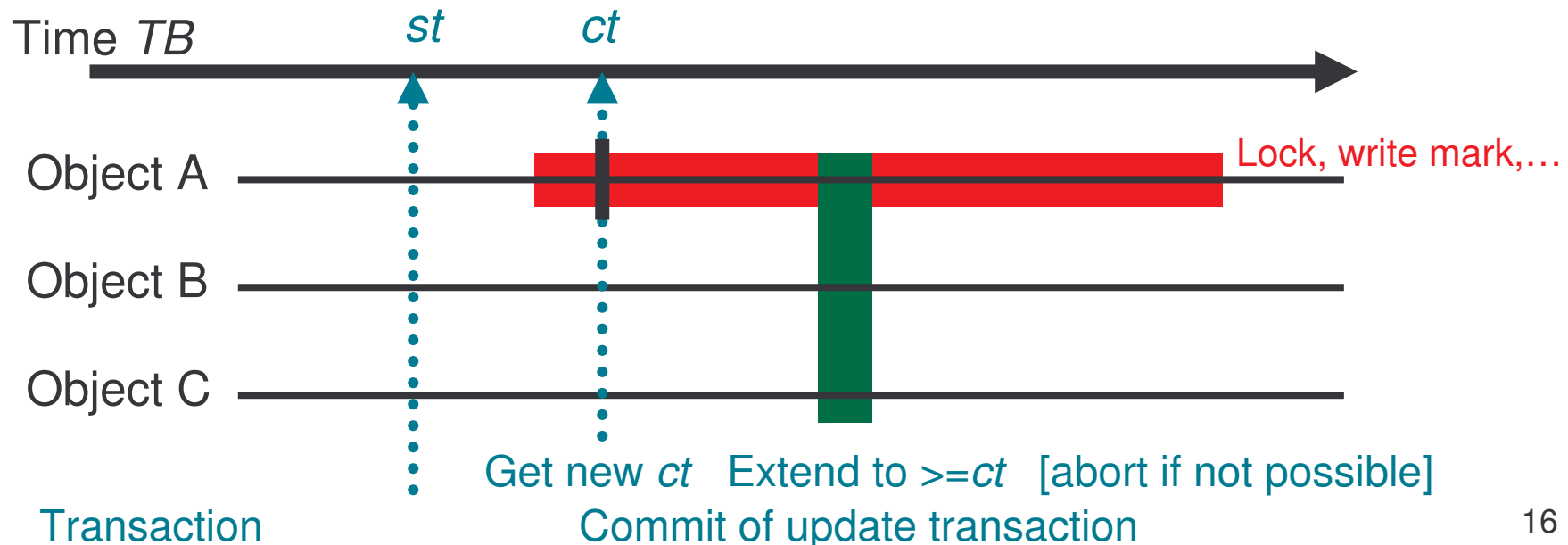
# Time-based Transactional Memory

- Global time base  $TB$
- Time-stamped object versions
- Consistent **snapshots** at time  $st$ : only read versions most recent at  $st$
- Extension: check if snapshot is consistent at more recent time
- Commit of update txn: get new commit time  $ct$  from  $TB$ , extend to  $ct$



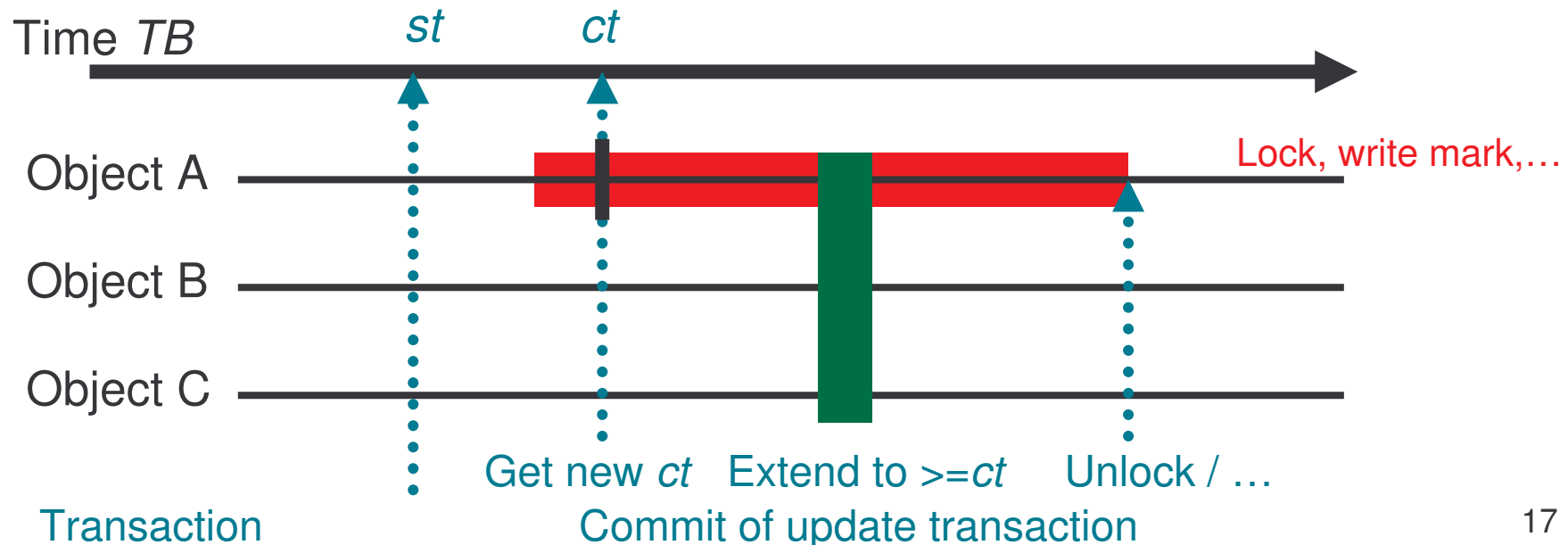
# Time-based Transactional Memory

- Global time base  $TB$
- Time-stamped object versions
- Consistent **snapshots** at time  $st$ : only read versions most recent at  $st$
- Extension: check if snapshot is consistent at more recent time
- Commit of update txn: get new commit time  $ct$  from  $TB$ , extend to  $ct$



# Time-based Transactional Memory

- Global time base  $TB$
- Time-stamped object versions
- Consistent **snapshots** at time  $st$ : only read versions most recent at  $st$
- Extension: check if snapshot is consistent at more recent time
- Commit of update txn: get new commit time  $ct$  from  $TB$ , extend to  $ct$



# Simple Time Base: Shared Counter

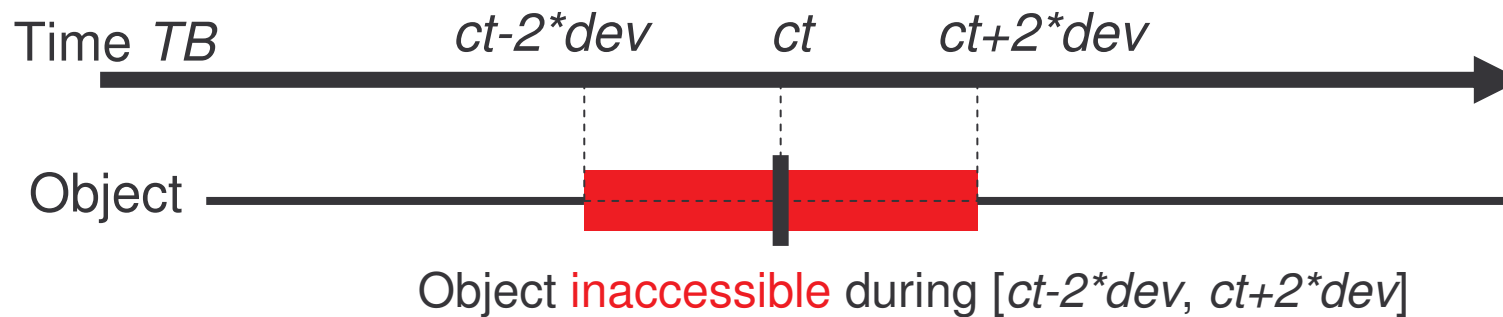
- Ticks on demand (commit attempts)
- Small systems: works well
  - Validation/Extension fast-path:  
No tick -> no changes
- Large systems: Contention!
- Optimizations that reduce contention:
  - Sharing timestamps (counter still shared)
  - Several counters (several problems ...)

# Perfectly synchronized physical clocks

- Observation: in large systems, counters tick steadily, like a clock
- Only difference: must wait for new clock tick
  - Must not surprise readers
  - Can't force clock to tick
- No contention (no or one-way communication)
- Synchronization with wall clock not required
- Perfect synchronization difficult to implement

# Synchronized Clocks with Bounded Deviation

- External clock synchronization (deviation bound:  $dev$ )
  - If linearizable global clock returns  $t$
  - Synchronized clock returns  $[t-dev, t+dev]$
- Possible deviation introduces **uncertainty**
- Solution: **mask** potential deviation
  - $2*dev$ : deviation at updater and reader
- Synchronized clocks less difficult to implement



# Performance

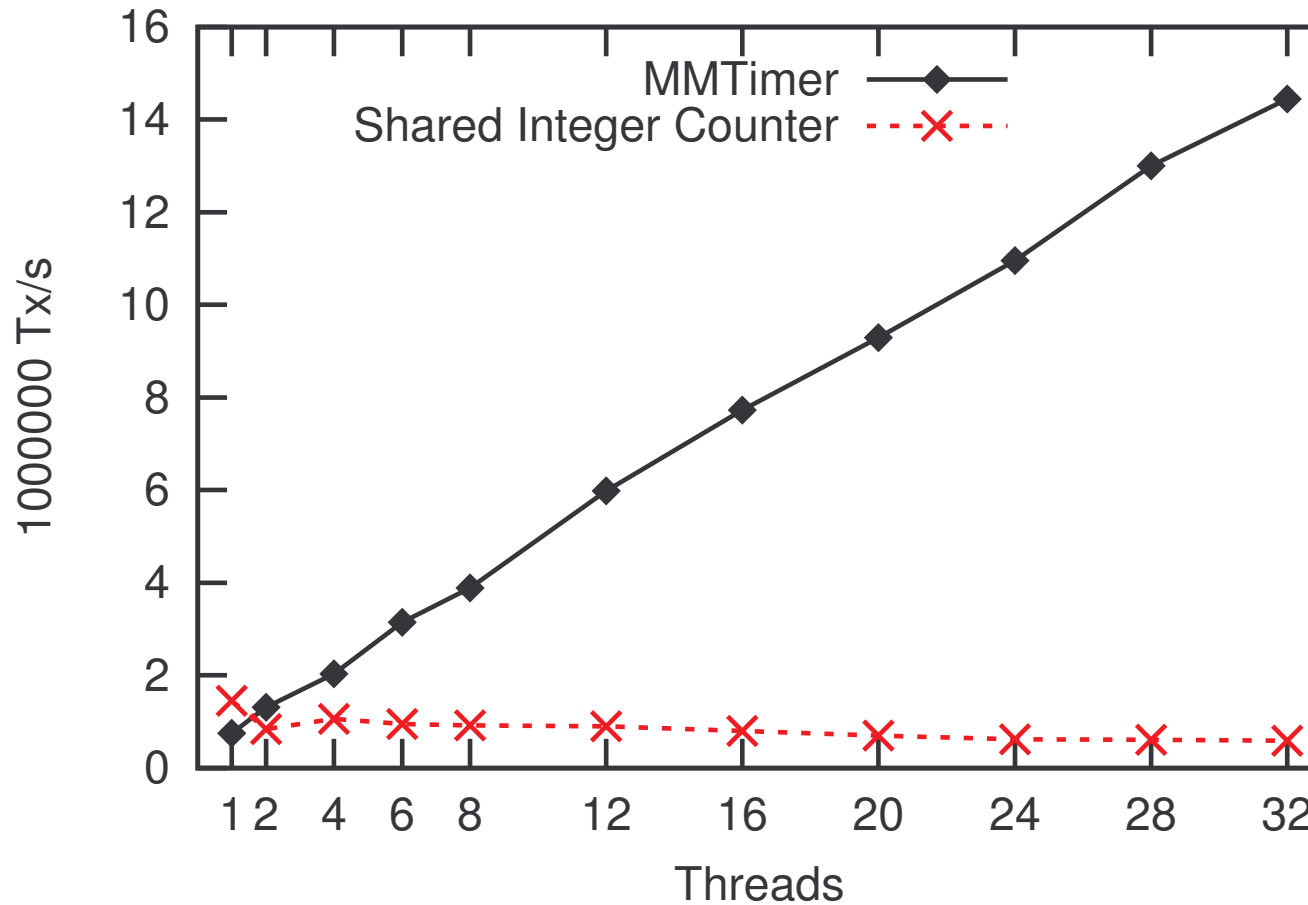
- Costs for time base accesses:
  - Time needed to read time base (at least 1 read per transaction)
    - Counter: cache miss (assume large system)
    - Synchronized clocks: depends on implementation
  - Waiting for / acquiring new CT:
    - Counter: fetch-and-add contention
    - Synchronized clocks: depends on clock granularity, read costs (e.g., no waiting with fast clocks)
- Attention: all costs relative to TM costs and workload
  - No general statements possible

# Case Study:

## MMTimer on an SGI Altix 4700

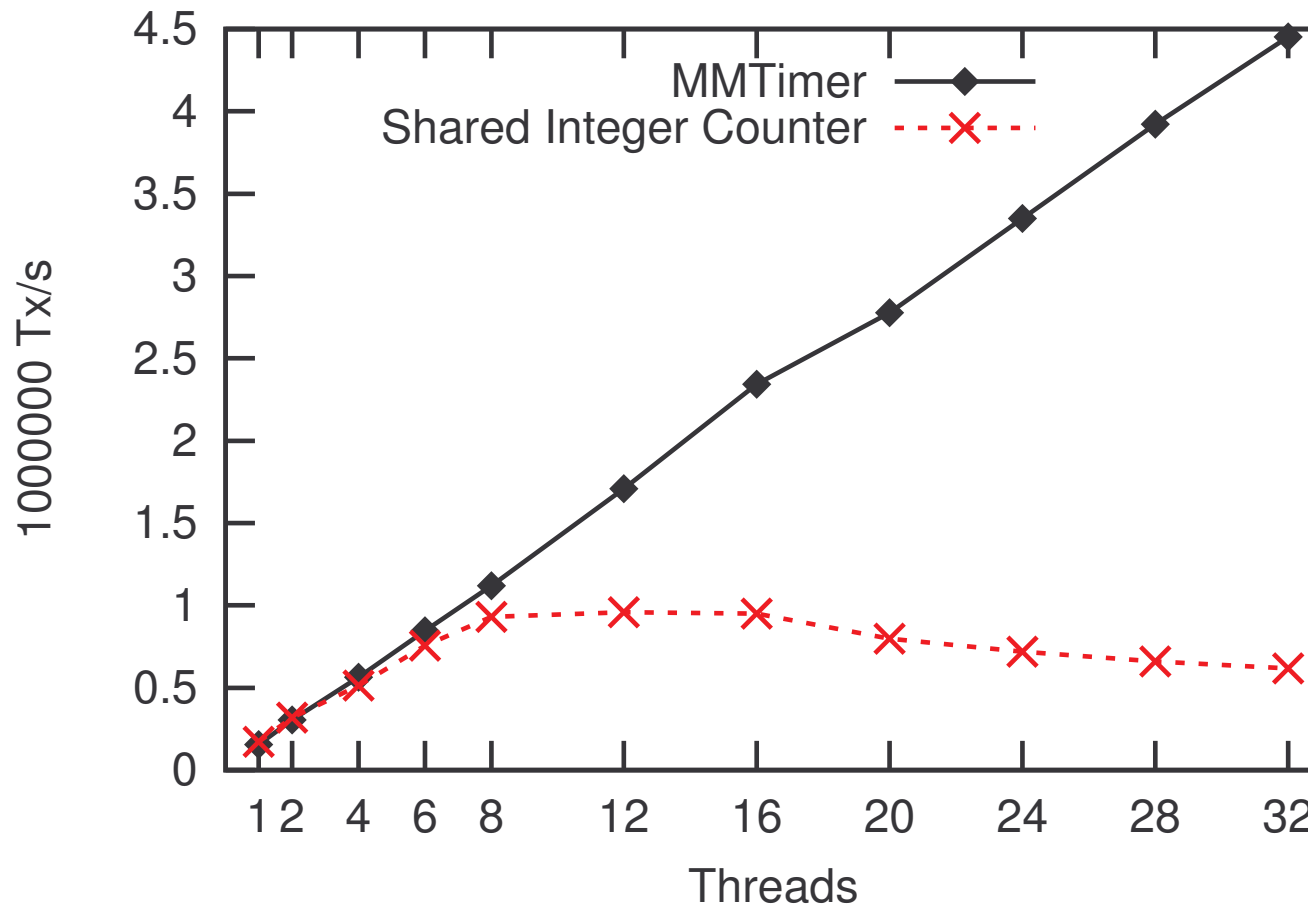
- SGI Altix 4700
  - ccNUMA machine, partitions of 512 Itanium cores (our benchmarks: 32 cores)
- MMTimer:
  - Synchronized 20MHz hardware clock
  - Reading takes 8 ticks / 400ns / 600cycles
  - Single node broadcasts clock signal
  - Dedicated wires for clock signal

# Transactions with disjoint accesses



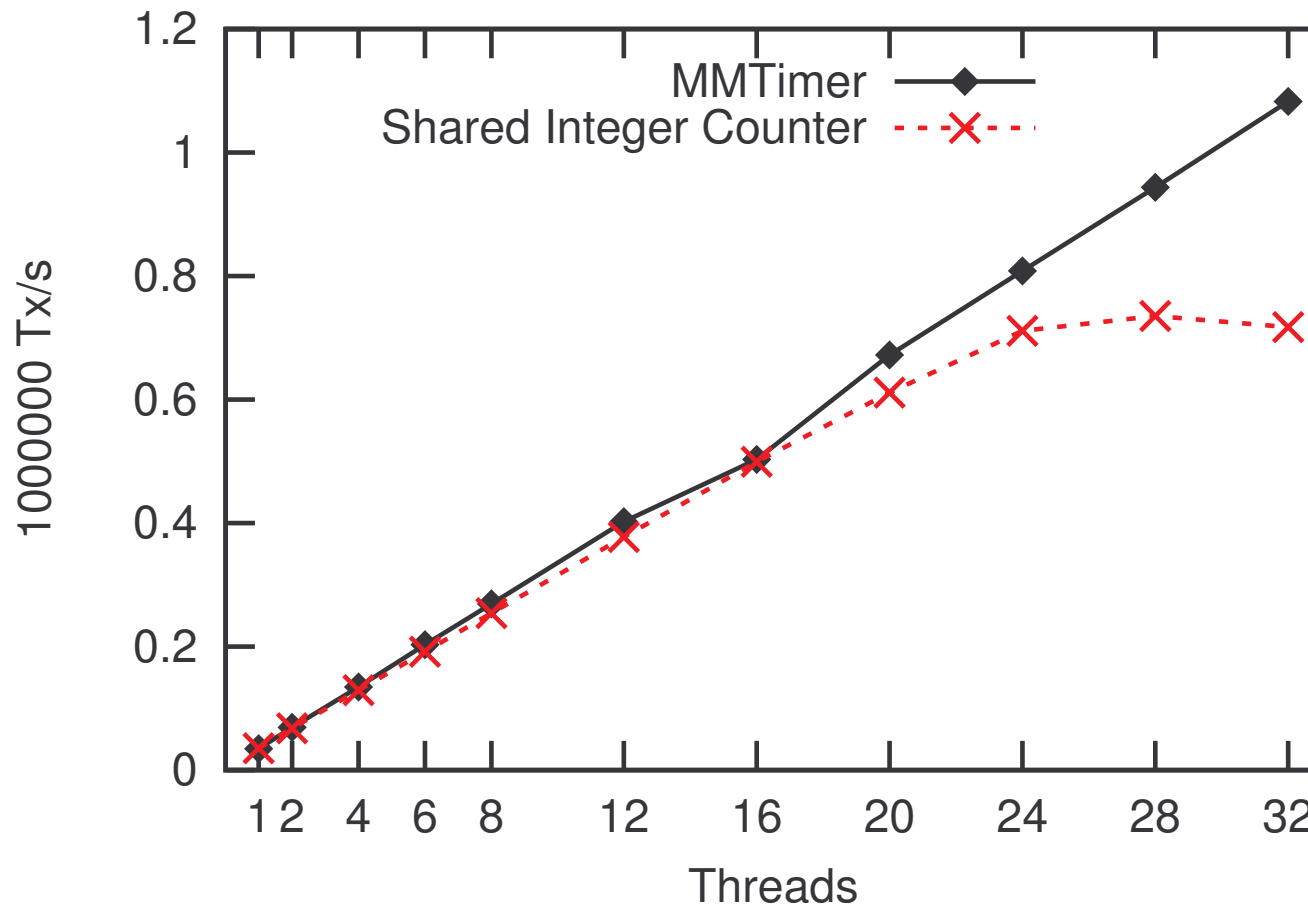
- **10 accesses** per transaction
- 90% reads
- Single thread:
  - MMTimer: **0.7** MTx/s
  - Counter: 1.4 MTx/s
- Two threads:
  - **1.3** vs. 0.8

# Transactions with disjoint accesses



- **100 accesses** per transaction
- 90% reads
- Single thread:
  - MMTimer: **0.15** MTx/s
  - Counter: 0.17 MTx/s

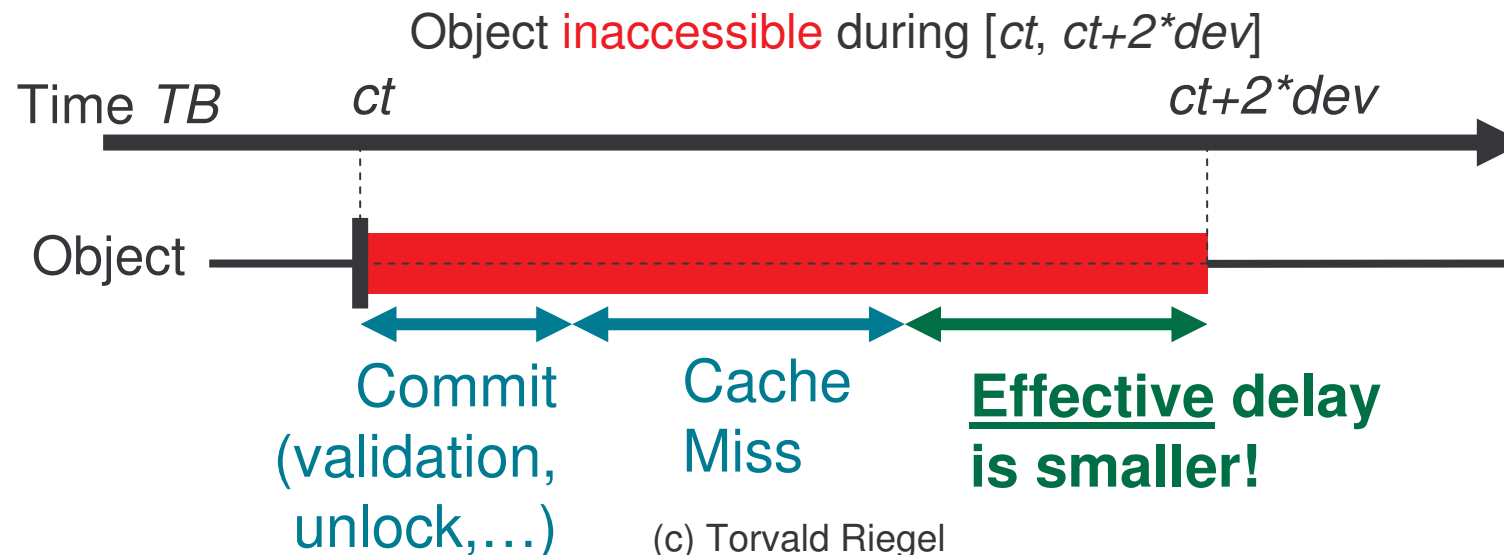
# Transactions with disjoint accesses



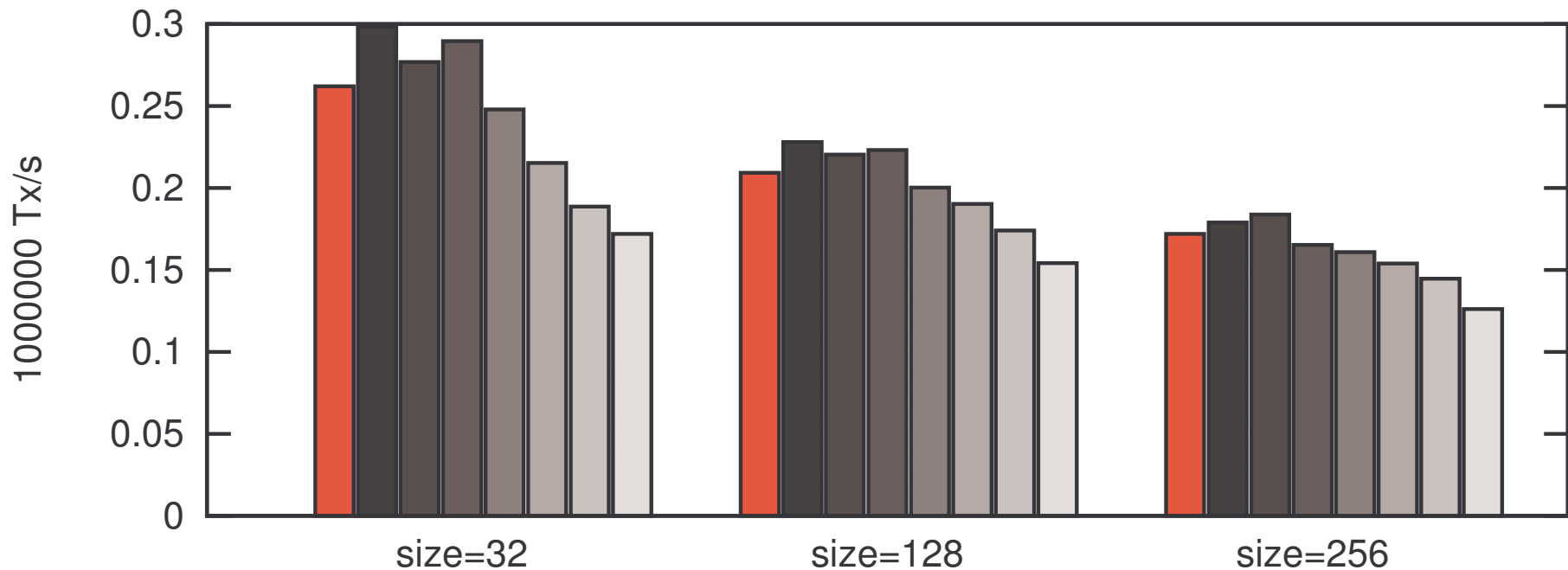
- **500 accesses** per transaction
- 90% reads
- Single thread:
  - MMTimer: 34.4 kTx/s
  - Counter: 35.2 kTx/s

# Influence of Clock Deviation Bounds

- Possible clock deviation prevents reading
- Problem: Read after write
  - (Multiversion TMs: write after read, too)
- Can delay transaction only once by max.  $2*dev$
- Only in workloads with handover / contention
  - Clock ID for each version: no delay when reading own updates



# Influence of Deviation Bounds: Linked-List Set



insert()/remove()/contains() transactions, 32 Threads, 20% update txns

# Conclusion

- Global time bases useful for concurrent algorithms
  - Timing information allows you to see history, not just the current situation
  - Snapshots are made easier and faster
- Current hardware clocks make global time bases feasible - even on large systems
- Give us better synchronized hardware clocks!

More about our work:  
<http://se.inf.tu-dresden.de>