Maintaining Multiple Versions in Software Transactional Memory

IDIT KEIDAR
DMITRI PERELMAN
RUI FAN

Aborts in STM

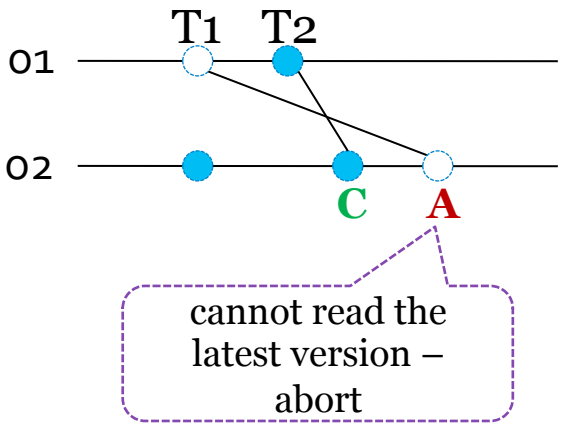
- Forceful aborts – an algorithm suspects correctness violation

- Aborting transactions is bad
  - work is lost
  - resources are wasted
  - overall throughput decreases
  - danger of livelock

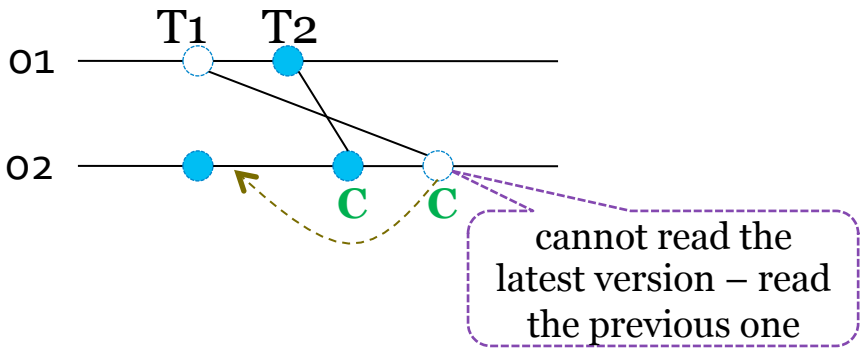
Multi-versioning in STM

- Keeping multiple versions can prevent aborts

Single-versioned STM

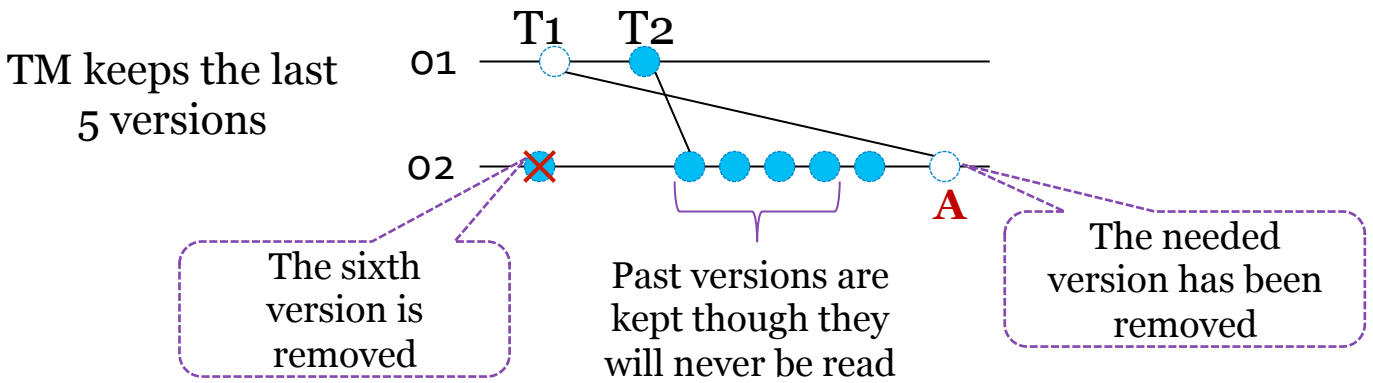


Multi-versioned STM



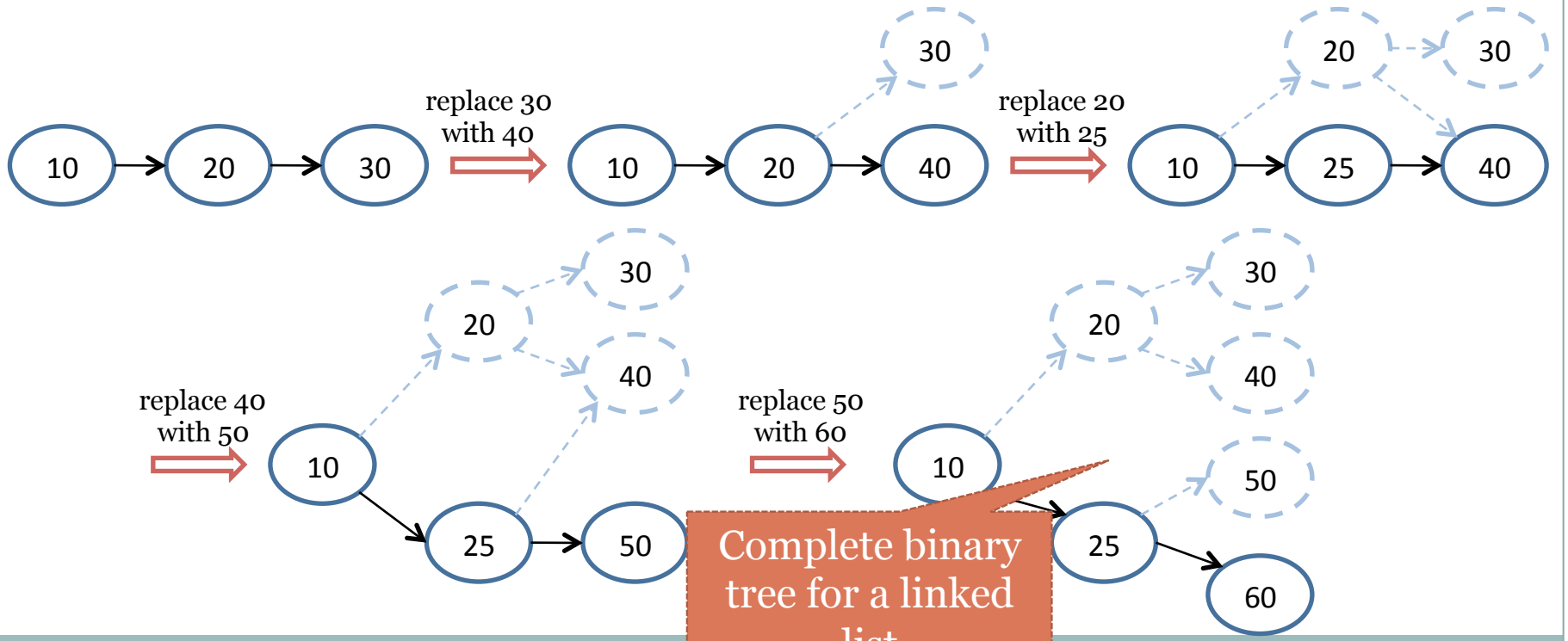
GC challenge

- Must clean up the old versions
- Some existing TMs keep a list of n past versions
  - some kept versions are useless
  - some potentially useful versions are removed



Memory growth

- k versions => k times more memory?
  - No! It can be much worse…
  - Consider pointer-based data structures



Permissiveness in multi-versioned STM

- Multi-Versioned (MV)-Permissiveness
  - each read-only transaction commits
  - an update transaction aborts only if it conflicts with another update transaction
- Practical – satisfied by Vboxes, SMV
  - would have been achieved by most multi-versioned algorithms
  - if they had kept all the needed object versions
- Responsive STM
  - a txn operation does not wait for other transactions to invoke new txn operations
  - to avoid trivial "global lock" solutions (no aborts and no concurrency)

Space optimality for MV-permissiveness

- Space optimality
  - An MV-permissive STM1 is space optimal if for any MV-permissive STM2 at any point of time:
    - #versions in STM1 ≤ #versions in STM2
- No responsive MV-permissive STM can be space optimal
- Sometimes, it's impossible to know whether to remove an old version
  - could save the need to keep other versions in future

MV permissiveness vs DAP

- Disjoint Access Parallelism (DAP) property: txns with disjoint data sets do not contend (no "common bottleneck")

- A responsive MV-permissive STM *cannot be DAP*
- Intuitively, contention point is "responsible" for real-time order guarantee
  - can forfeit RTO and satisfy DAP

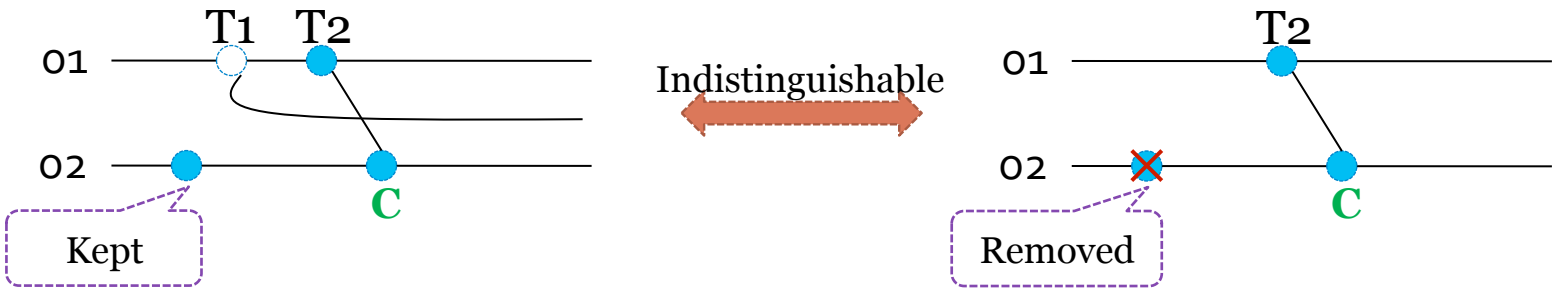
SMV: Selective Multi-Versioning STM

- Responsive and MV-Permissive
  - each read-only transaction commits
  - cannot be space-optimal
  - cannot be DAP
- Versions are kept as long as they might be needed
- Read-only transactions are invisible to other transactions
  - do not change data that can be read by others
  - avoids cache thrashing

GC challenge

- A version is removed when it has no potential readers
- Readers are invisible

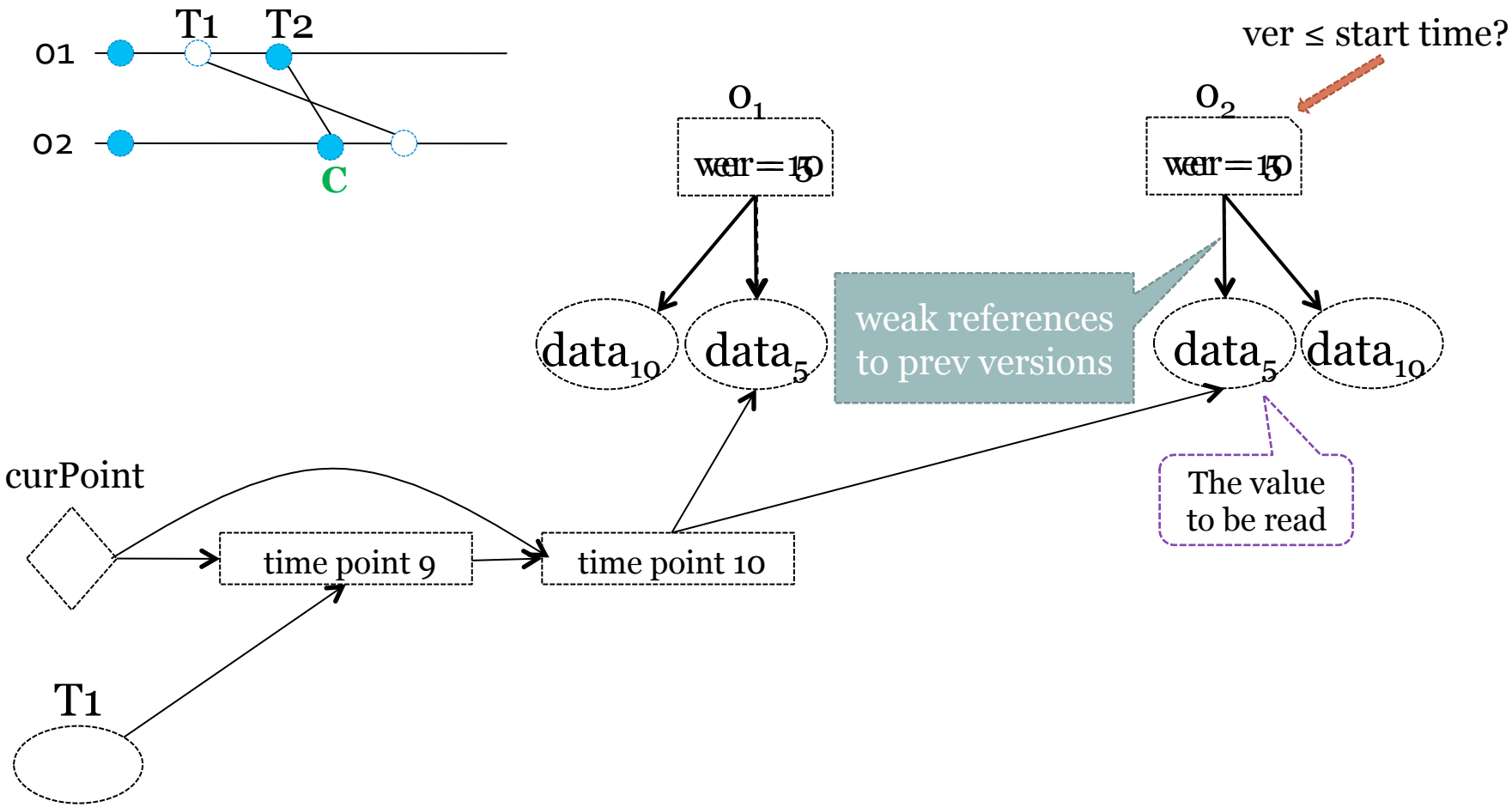
- No transaction can know whether a given version can be removed
  - explicit GC is not possible



Automated GC in SMV

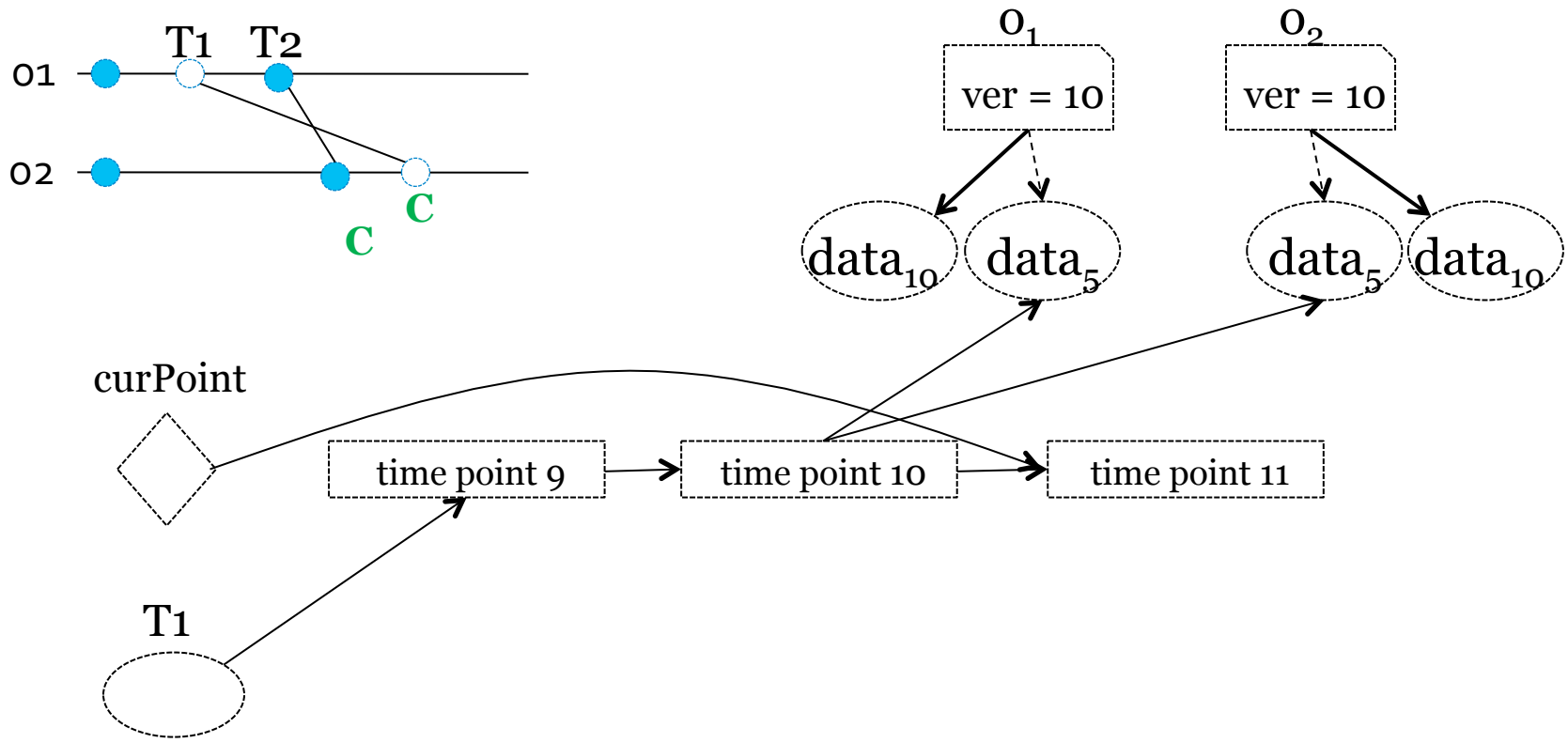
- Solution: use auxiliary GC threads provided by managed memory systems
  - remove unreachable object versions
- Read-only transactions are invisible to other transactions, but visible to the “see-all” GC threads
  - theoretically visible
  - practically invisible
    - GC threads run infrequently
    - does not add cache-coherency overhead

SMV overview



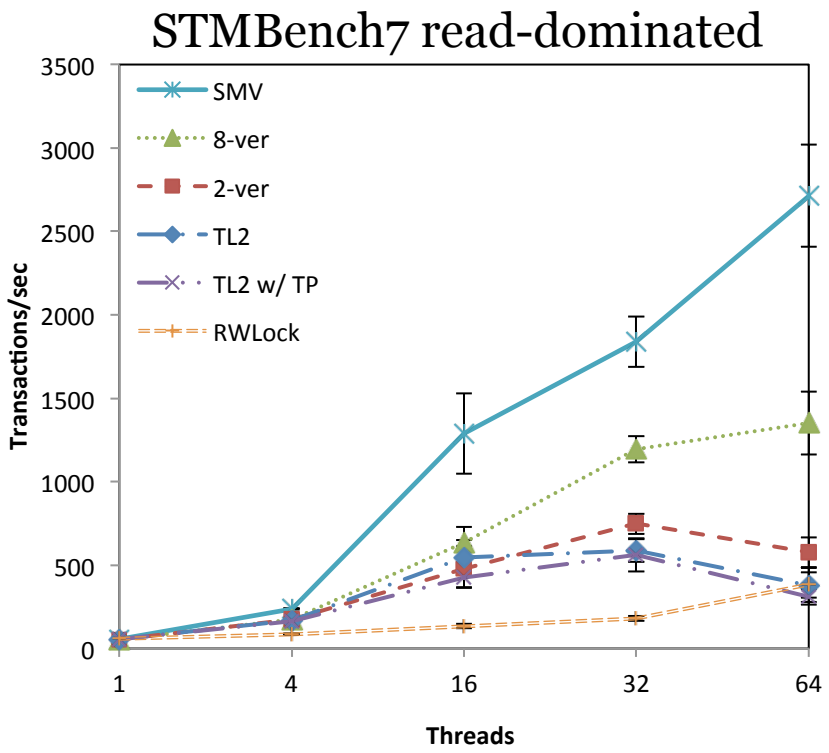
SMV GC

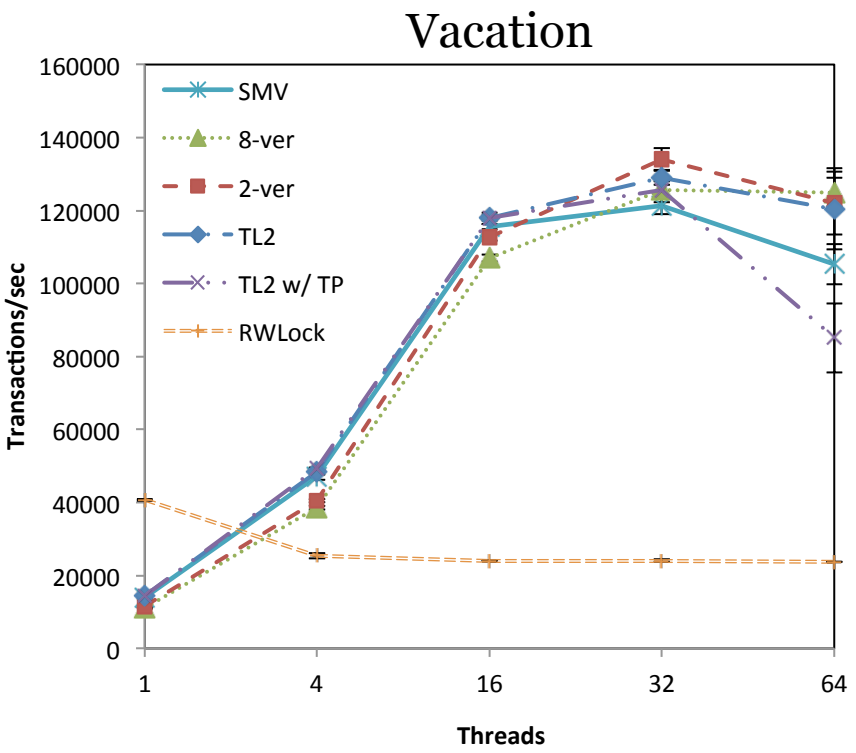
- Unneeded versiones are GCed automatically



SMV performance

- Great for read-dominated workloads
  - especially with long read operations (snapshot, aggregation, etc.)
- Low overhead if there are no read-only transactions at all





Conclusions

- Multi-versioning can improve STM performance
  - especially useful for long read-only transactions
- Keeping a constant number of versions is not efficient
  - not every needed version can be found
  - exponential memory growth
- MV-permissiveness imposes overheads of its own
  - cannot be space efficient
  - cannot be DAP
- SMV uses automatic GC capabilities for deleting old versions
  - the readers stay invisible

References

- On Maintaining Multiple Versions in STM
  - Perelman, Keidar, Fan, PODC'10
- SMV: Selective Multi-Versioning STM
  - Perelman, Byshevsky, Litmanovich, Keidar, submitted