# Serializability of Transactions in Software Transactional Memory

**Utku Aydonat**

*uaydonat@eecg.toronto.edu*

*Department of Electrical and Computer Engineering*
*University of Toronto*

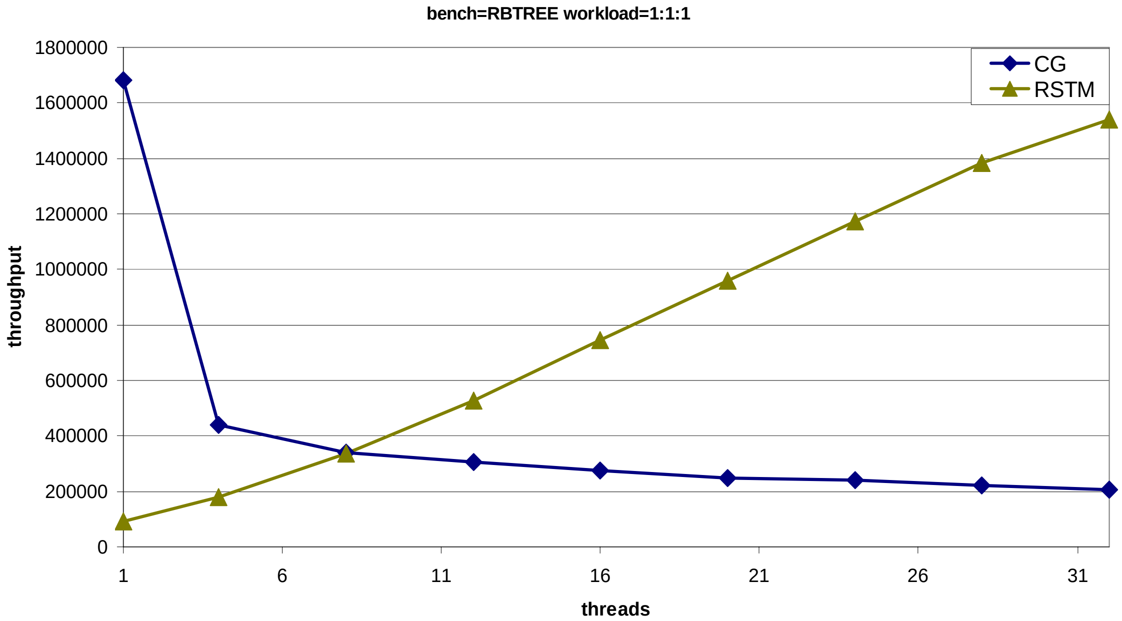
Workshop on Transactional Computing (TRANSACT)
2008

# Outline

- Motivation
  – STM systems perform poorly for some applications.
- 2-Phase Locking
  – Why can it be an overly conservative technique?
- Conflict Serializability
  – What is the advantage of conflict-serializability?
- Ensuring Conflict-Serializability
  – How can we implement conflict-serializability efficiently?
- Multi-Versioning
- Experimental Evaluation
- Conclusions

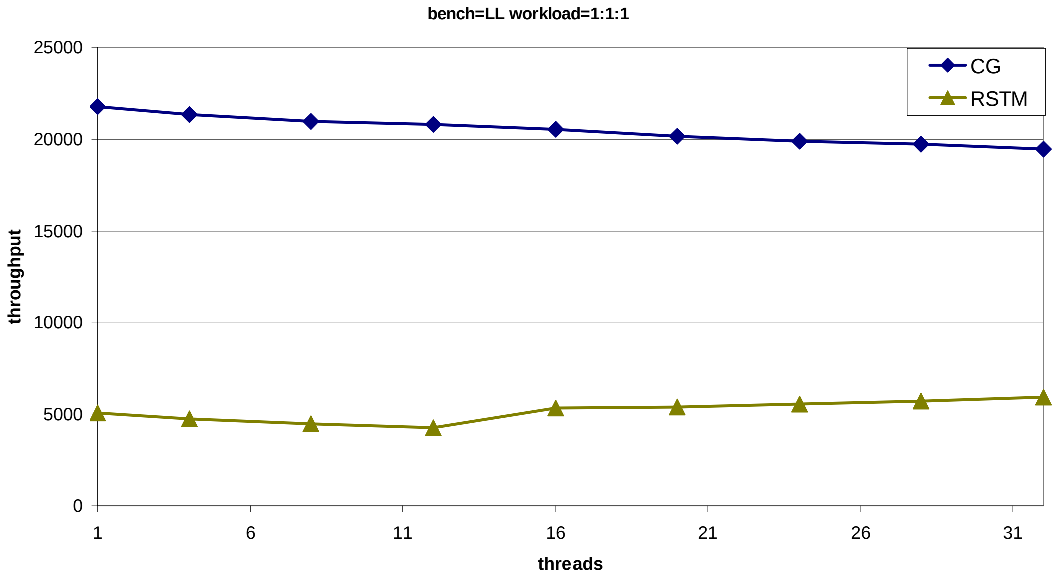
# Motivation

- Short transactions with low data sharing
  - Successful STM implementations (blocking STMs, time-based STMs, etc.)
  - Aborts are insignificant for performance.
    - For RBT, the abort rate is 0.23% at 32 threads, avg. tx. time is 11 μsec.
    - Throughput:



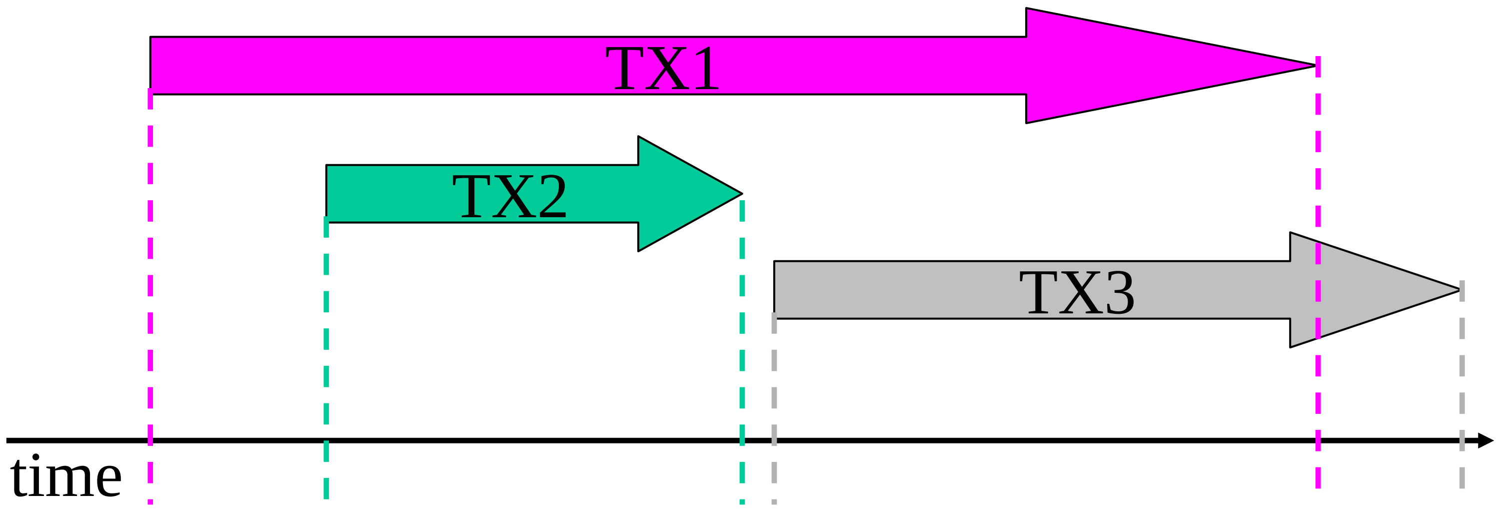
# Motivation

- Long transactions with high data sharing
  - Popular STM implementations suffer in performance.
  - Abort rates impact the performance.
    - For LL, the abort rate is 79% at 32 threads, avg. tx. time is 197 μsec.
    - Throughput:



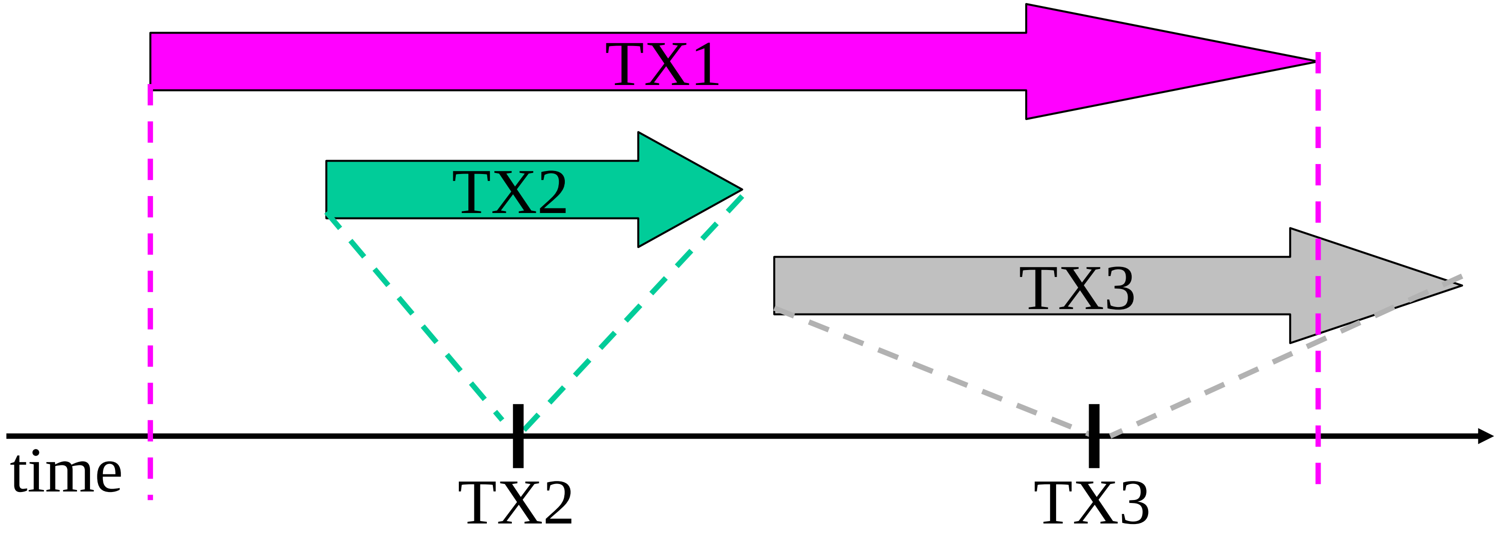
# Consistency in STMs

- Are STMs too eager to abort transactions?
- Linearizability: Correctness criteria for STMs.
  - Transactions appear to execute atomically at some point between their start time and commit time.



# Consistency in STMs

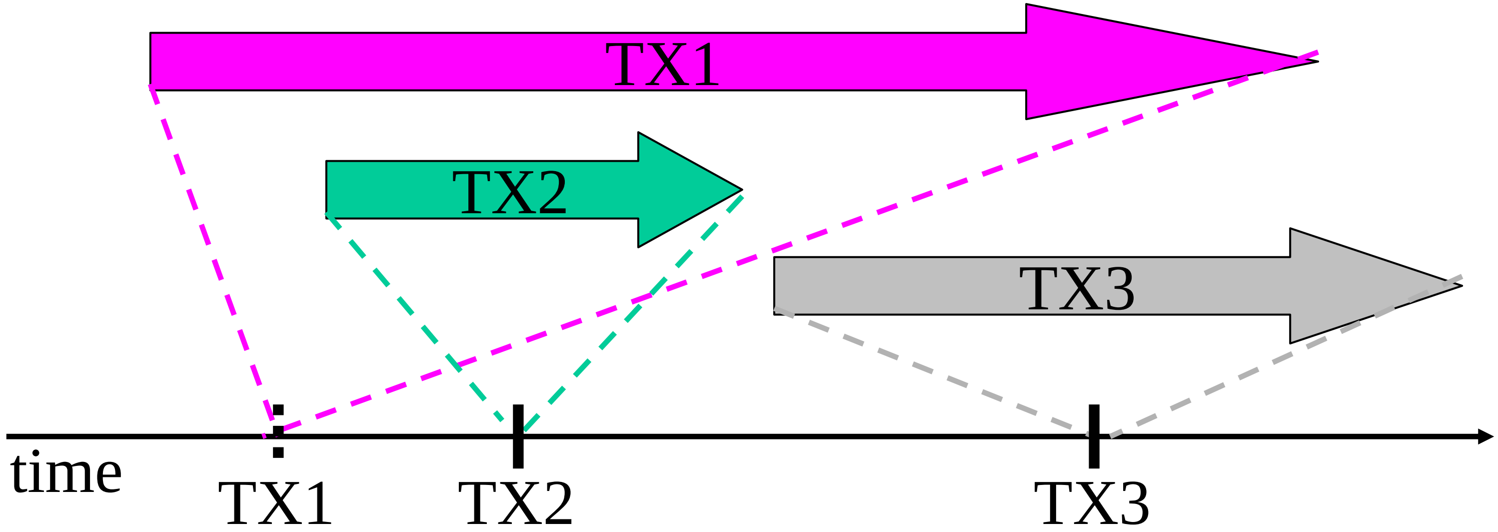
- Are STMs too eager to abort transactions?
- Linearizability: Correctness criteria for STMs.
  - Transactions appear to execute atomically at some point between their start time and commit time.




**Equivalent sequential execution!**

# Consistency in STMs

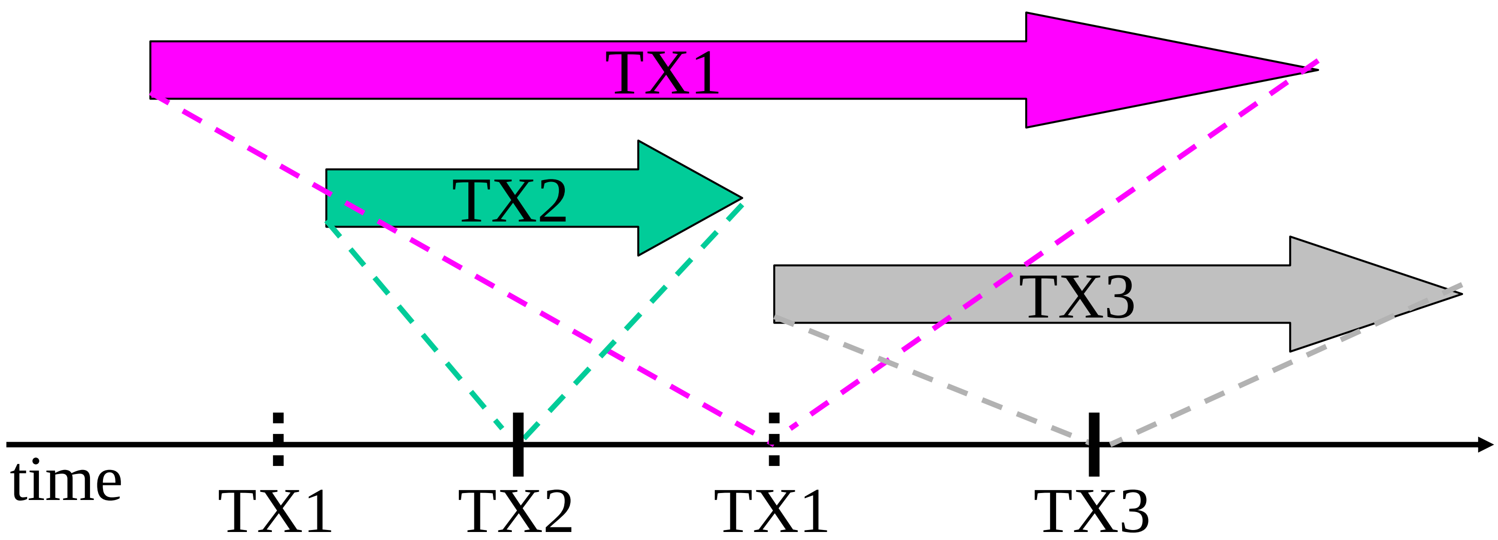
- Are STMs too eager to abort transactions?
- Linearizability: Correctness criteria for STMs.
  - Transactions appear to execute atomically at some point between their start time and commit time.




**Equivalent sequential execution!**

# Consistency in STMs

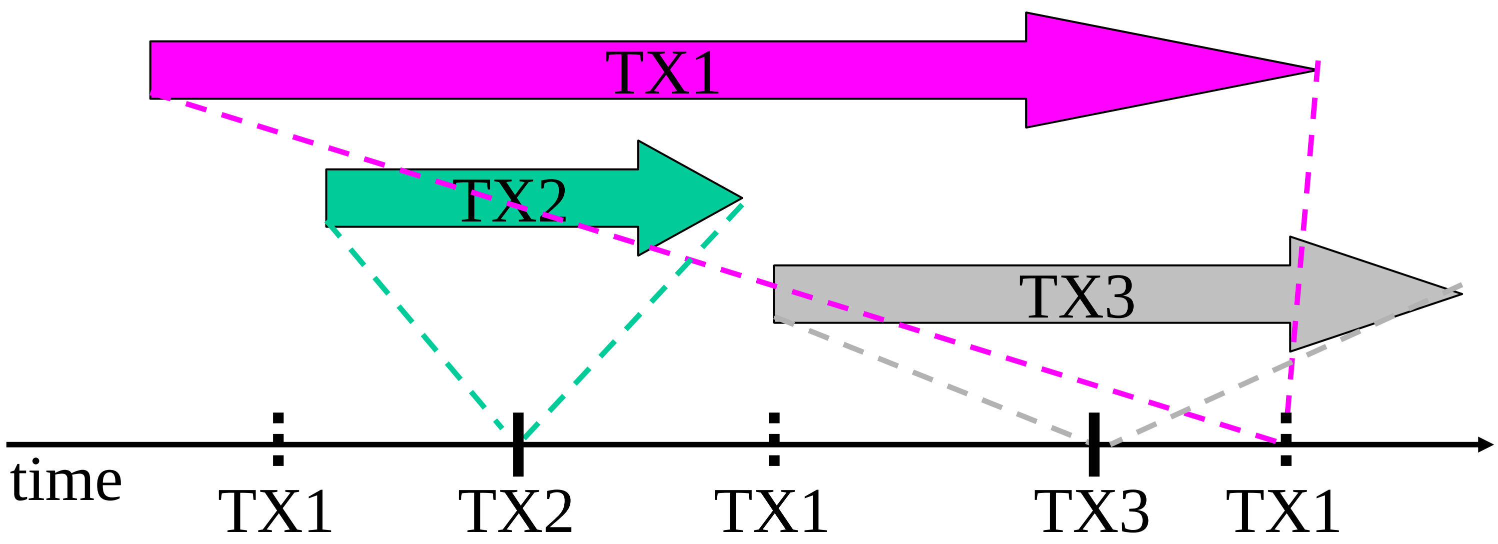
- Are STMs too eager to abort transactions?
- Linearizability: Correctness criteria for STMs.
  - Transactions appear to execute atomically at some point between their start time and commit time.




**Equivalent sequential execution!**

# Consistency in STMs

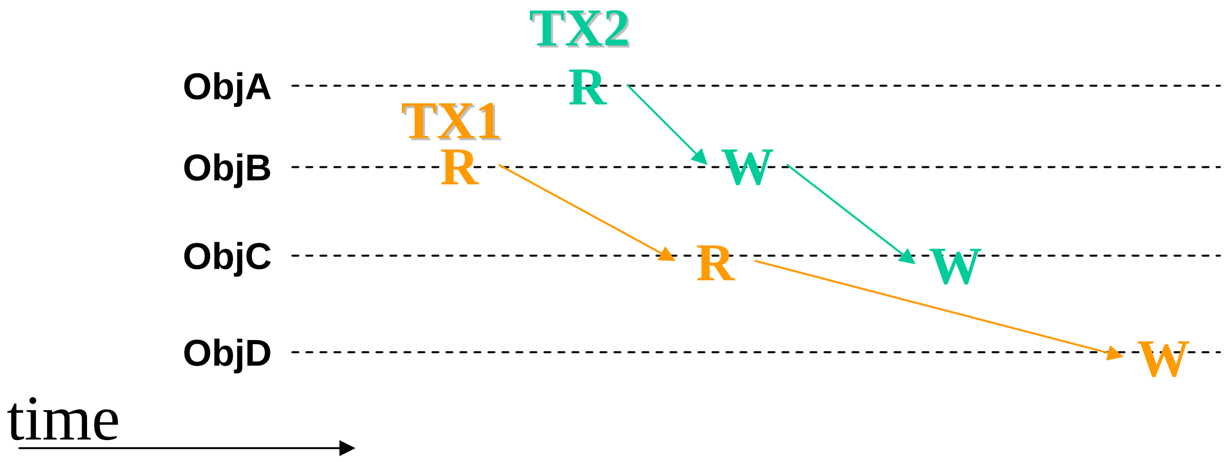
- Are STMs too eager to abort transactions?
- Linearizability: Correctness criteria for STMs.
  - Transactions appear to execute atomically at some point between their start time and commit time.




Equivalent sequential execution!

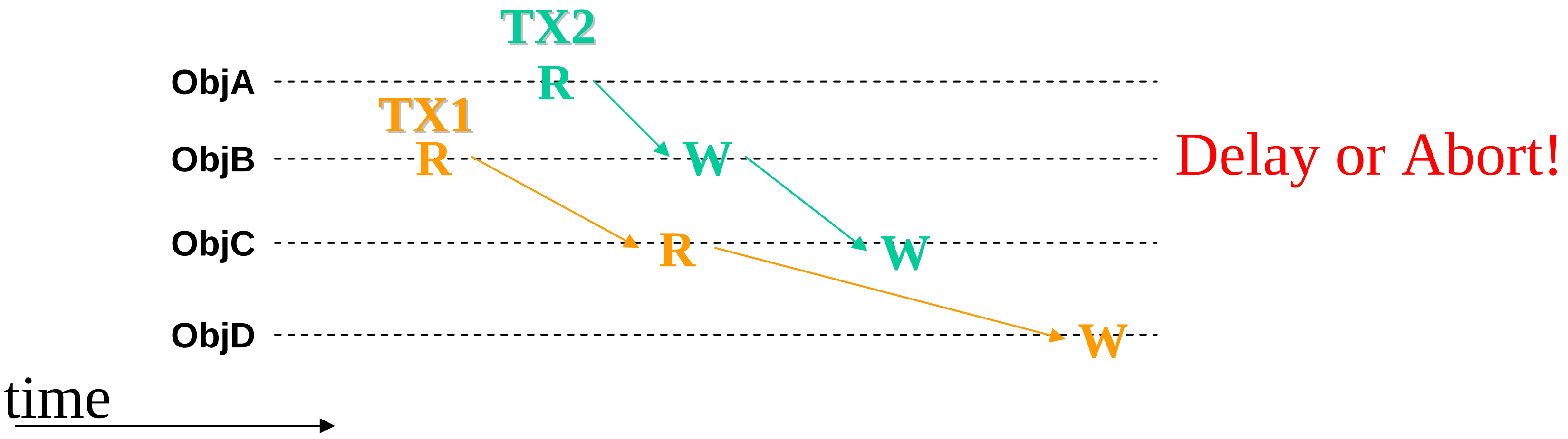
# Two Phase Locking

- STMs emulate two phase locking (2PL) to ensure linearizability.
  - Conflicting accesses are not allowed from the access time till the transaction commits.
  - More conservative than linearizability.
  - Easy to implement: fast transactions but frequent aborts.



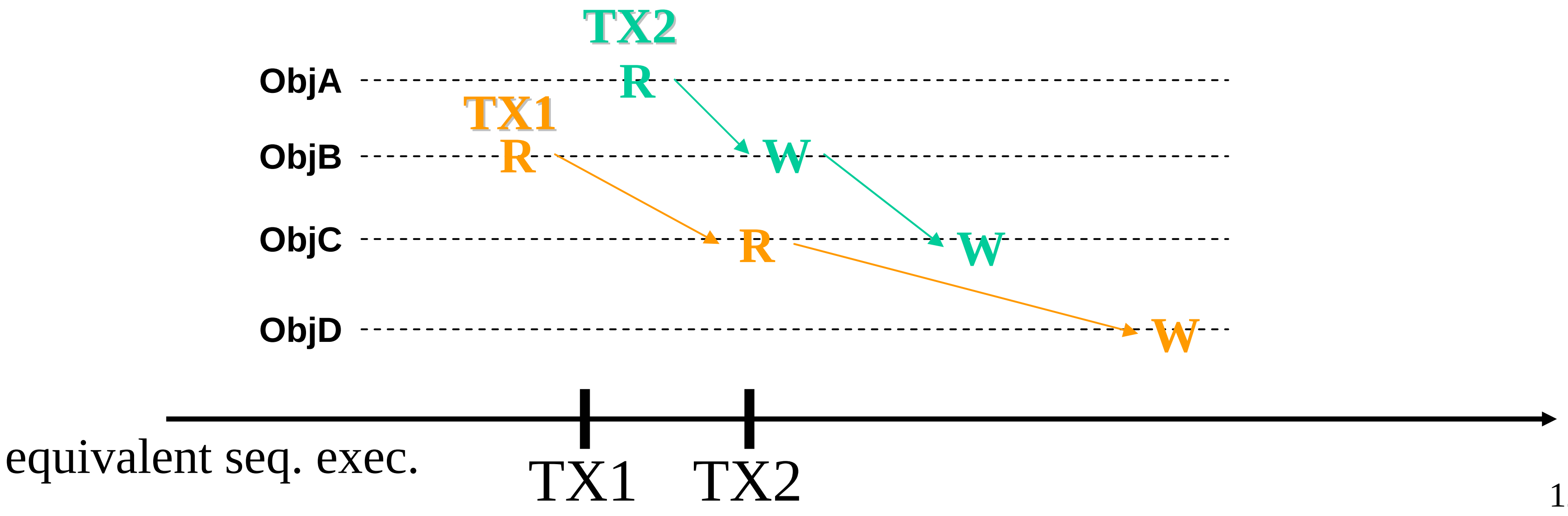
# Two Phase Locking

- STMs emulate two phase locking (2PL) to ensure linearizability.
  - Conflicting accesses are not allowed from the access time till the transaction commits.
  - More conservative than linearizability.
  - Easy to implement: fast transactions but frequent aborts.



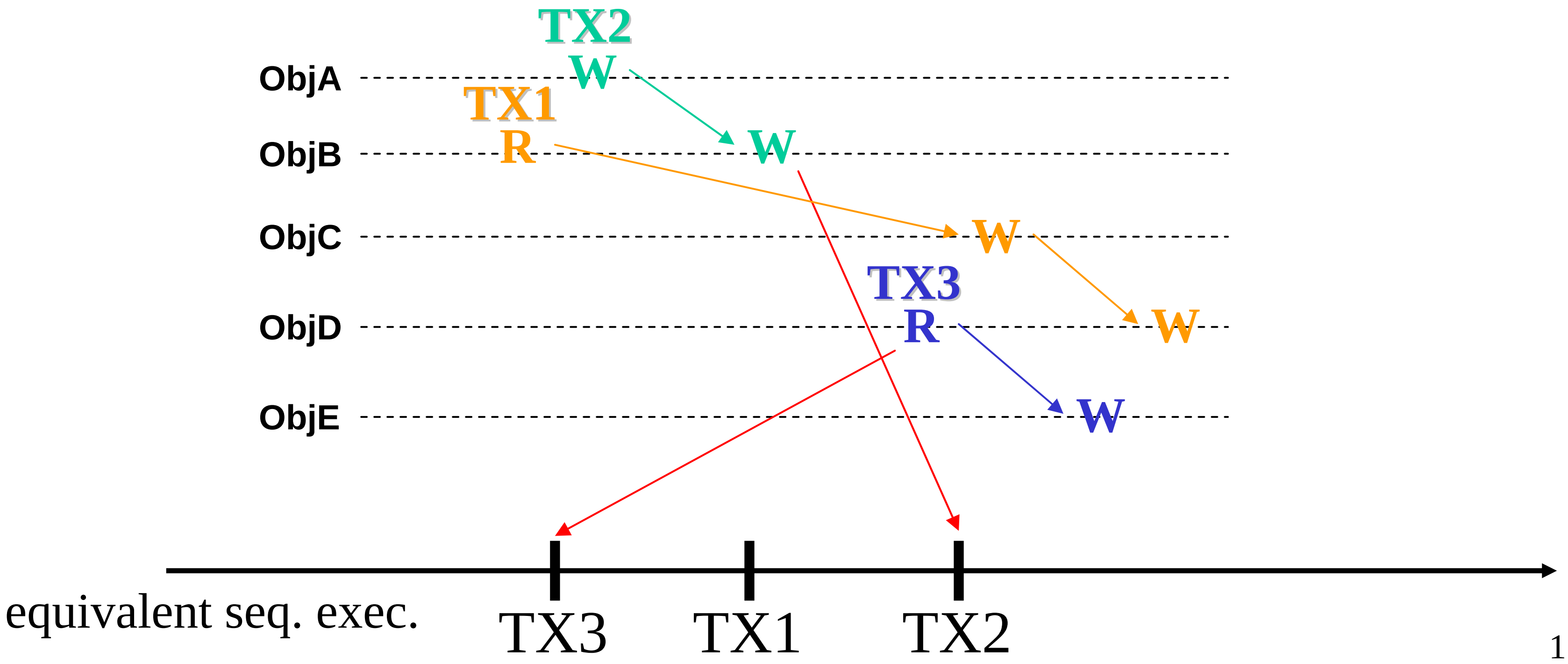
# Two Phase Locking

- STMs emulate two phase locking (2PL) to ensure linearizability.
  - Conflicting accesses are not allowed from the access time till the transaction commits.
  - More conservative than linearizability.
  - Easy to implement: fast transactions but frequent aborts.



# Conflict-Serializability (CS)

- Allows conflicting accesses as long as we can find a valid ordering for transactions; an ordering that matches the order of conflicting accesses.
- More relax than 2PL & linearizability.



# Conflict Serializability (CS)

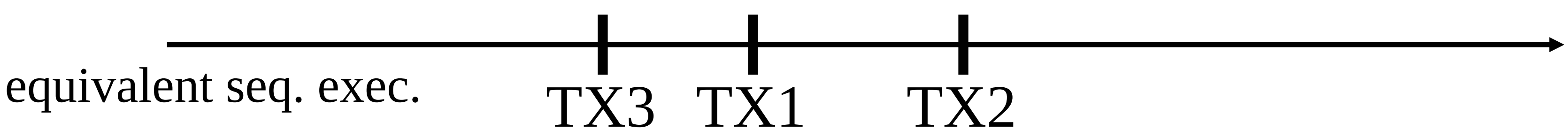
- Conclusion: We can reduce abort rates in STMs by implementing CS rather than 2PL.

- We can build a precedence/ordering/serializability graph.
  - Too slow

- We use Serializability Order Numbers (SONs)

# Serializability Order Numbers (SONs)

- Each transaction gets a SON number when it commits.
- The order of SON numbers matches the order of conflicting accesses for all transactions.
  - Rule1: If TX1 accesses an object committed by TX2
    SON(TX1) > SON(TX2)
  - Rule2: If TX2 commits an object already read by TX1
    SON(TX2) > SON(TX1)
- Keep an SON range for each active transaction [lb, ub]
  - Assign SON number based on lb & ub, such that lb < SON < ub
  - If up < lb at any point, transaction cannot be serialized.



# Serializability Order Numbers (SONs)

*[0:∞]* **Start TX1;** *[0:∞]* **Start TX2;** *[0:∞]* **Start TX3;**

**time**

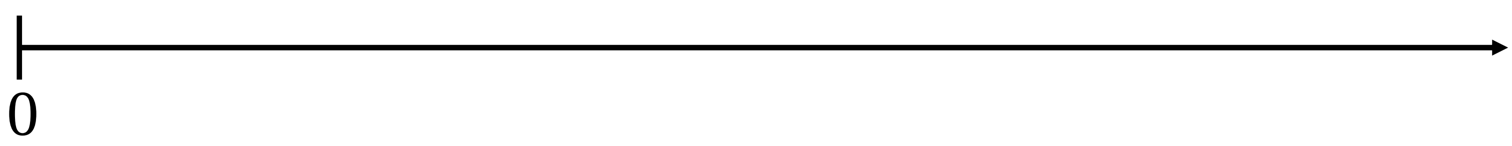
0

**Equivalent sequential execution!**

# Serializability Order Numbers (SONs)

| | | | | | |
|---|---|---|---|---|---|
| *[0:∞]* | **Start TX1;** | *[0:∞]* | **Start TX2;** | *[0:∞]* | **Start TX3;** |
| *[0:∞]* | **Read(A);** | *[0:∞]* | **Read(B);** | | **.** |
| | **.** | | **.** | *[0:∞]* | **Write(B);** |

**time**

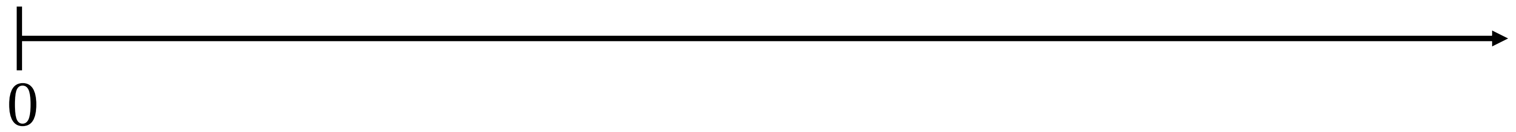



**Equivalent sequential execution!**

# Serializability Order Numbers (SONs)

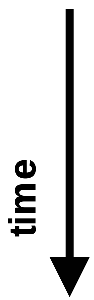
time

| | | | | | |
|---|---|---|---|---|---|
| *[0:∞]* | **Start TX1;** | *[0:∞]* | **Start TX2;** | *[0:∞]* | **Start TX3;** |
| *[0:∞]* | **Read(A);** | *[0:∞]* | **Read(B);** | | **.** |
| | **.** | | **.** | *[0:∞]* | **Write(B);** |
| | | | | | **Commit;** |

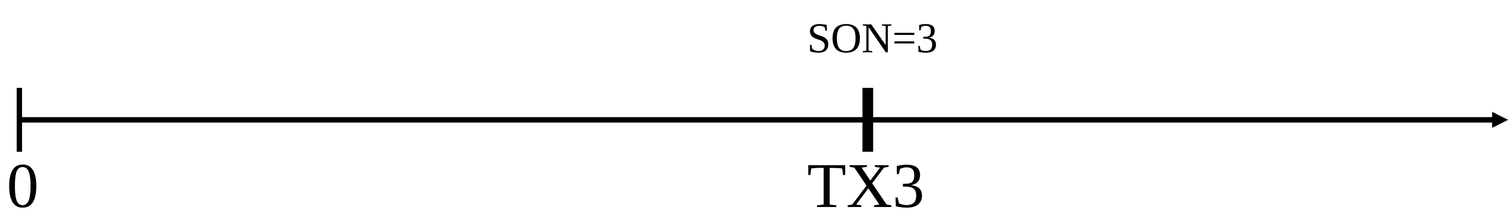



**Equivalent sequential execution!**

# Serializability Order Numbers (SONs)

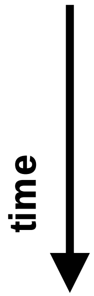



| | | | | | |
|---|---|---|---|---|---|
| *[0:∞]* | **Start TX1;** | *[0:∞]* | **Start TX2;** | *[0:∞]* | **Start TX3;** |
| *[0:∞]* | **Read(A);** | *[0:∞]* | **Read(B);** | | **.** |
| | **.** | | **.** | *[0:∞]* | **Write(B);** |
| | | | | *SON=3* | **Commit;** |

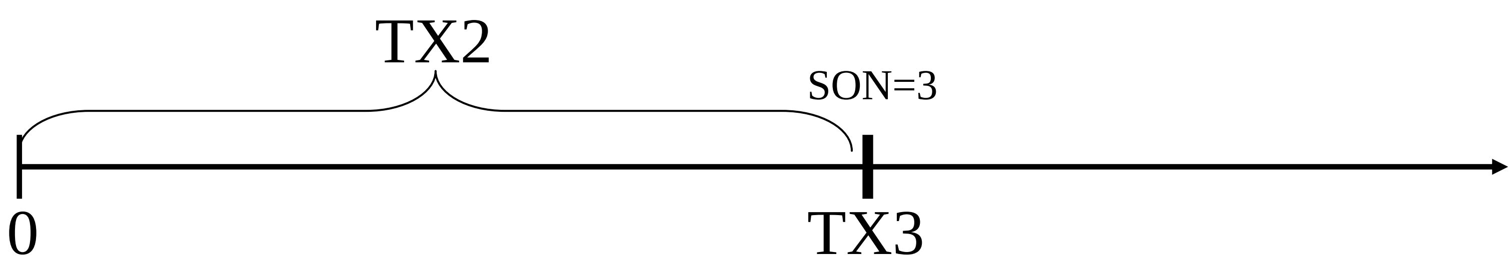



**Equivalent sequential execution!**

# Serializability Order Numbers (SONs)




| | | | | | |
|---|---|---|---|---|---|
| *[0:∞]* | **Start TX1;** | *[0:∞]* | **Start TX2;** | *[0:∞]* | **Start TX3;** |
| *[0:∞]* | **Read(A);** | *[0:∞]* | **Read(B);** | | **.** |
| | **.** | | **.** | *[0:∞]* | **Write(B);** |
| | **.** | *[0:3]* ◄······ | | *SON=3* | **Commit;** |

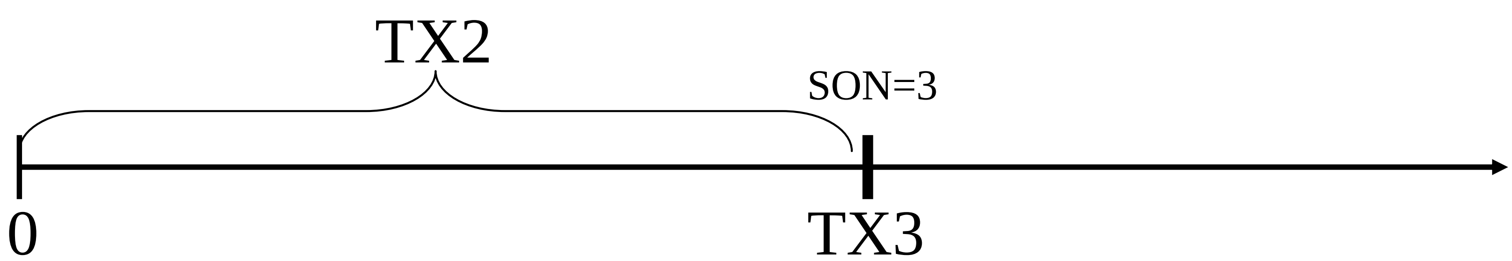



**Equivalent sequential execution!**

# Serializability Order Numbers (SONs)

time ↓

| | | | | | |
|---|---|---|---|---|---|
| *[0:∞]* | **Start TX1;** | *[0:∞]* | **Start TX2;** | *[0:∞]* | **Start TX3;** |
| *[0:∞]* | **Read(A);** | *[0:∞]* | **Read(B);** | | . |
| | . | | . | *[0:∞]* | **Write(B);** |
| | . | *[0:3]* | ← | *SON=3* | **Commit;** |
| | . | *[0:3]* | **Write(A);** | | |
| | | | **Commit;** | | |

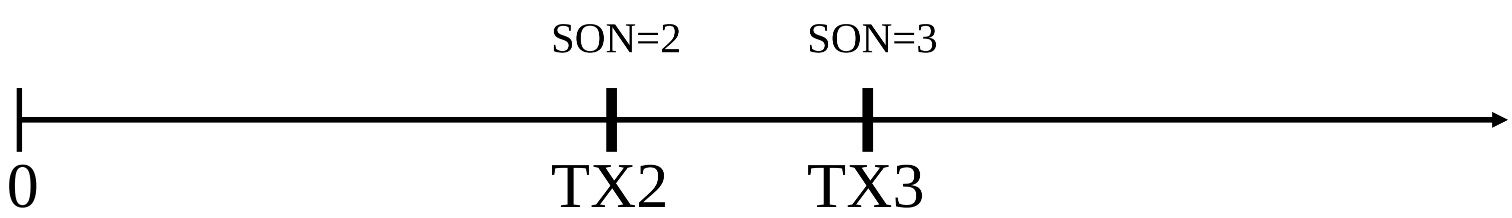



**Equivalent sequential execution!**

# Serializability Order Numbers (SONs)

time ↓

| | TX1 | | TX2 | | TX3 |
|---|---|---|---|---|---|
| *[0:∞]* | **Start TX1;** | *[0:∞]* | **Start TX2;** | *[0:∞]* | **Start TX3;** |
| *[0:∞]* | **Read(A);** | *[0:∞]* | **Read(B);** | | **.** |
| | **.** | | **.** | *[0:∞]* | **Write(B);** |
| | **.** | *[0:3]* | ← | *SON=3* | **Commit;** |
| | **.** | *[0:3]* | **Write(A);** | | |
| | | *SON=2* | **Commit;** | | |

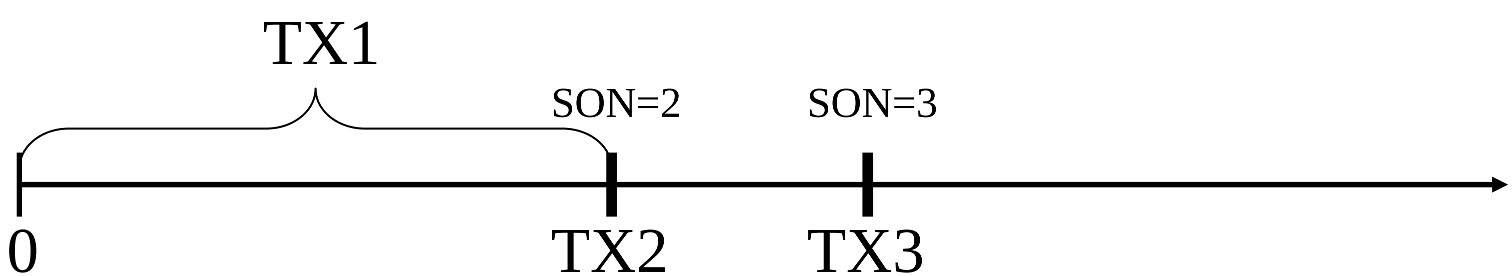



**Equivalent sequential execution!**

# Serializability Order Numbers (SONs)

time ↓

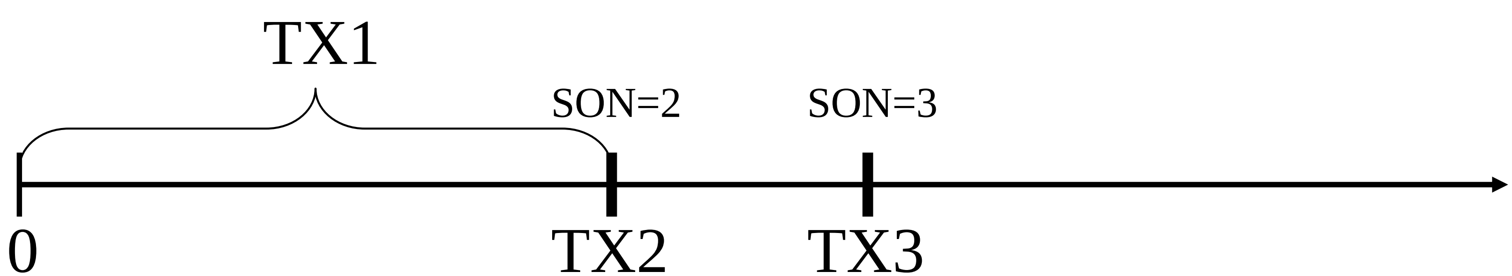
| | | | | | |
|---|---|---|---|---|---|
| *[0:∞]* | **Start TX1;** | *[0:∞]* | **Start TX2;** | *[0:∞]* | **Start TX3;** |
| *[0:∞]* | **Read(A);** | *[0:∞]* | **Read(B);** | | . |
| | . | | . | *[0:∞]* | **Write(B);** |
| | . | *[0:3]* ◄····· | | *SON=3* | **Commit;** |
| | . | *[0:3]* | **Write(A);** | | |
| *[0:2]* ◄····· | | *SON=2* | **Commit;** | | |




**Equivalent sequential execution!**

# Serializability Order Numbers (SONs)

| time | | | | | | |
|---|---|---|---|---|---|---|
| ↓ | *[0:∞]* | **Start TX1;** | *[0:∞]* | **Start TX2;** | *[0:∞]* | **Start TX3;** |
| | *[0:∞]* | **Read(A);** | *[0:∞]* | **Read(B);** | | **.** |
| | | **.** | | **.** | *[0:∞]* | **Write(B);** |
| | | **.** | *[0:3]* | ◀········ | *SON=3* | **Commit;** |
| | | **.** | *[0:3]* | **Write(A);** | | |
| | *[0:2]* | ◀········ | *SON=2* | **Commit;** | | |
| | | **Read(B);** | | | | |

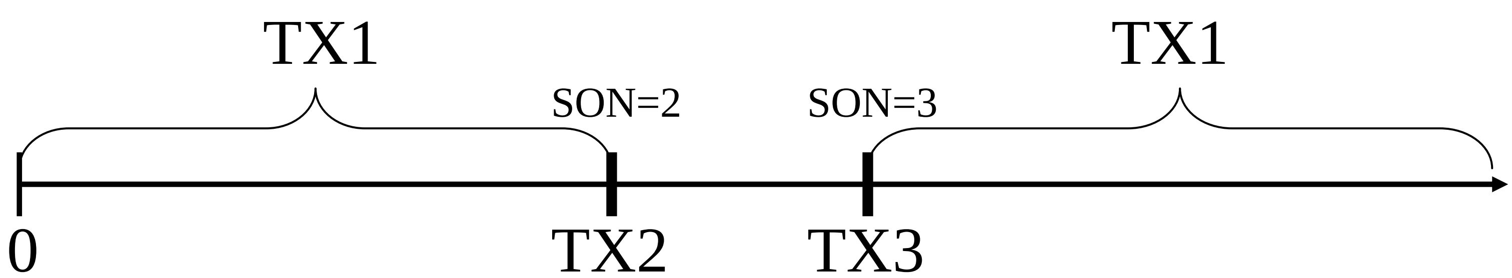



**Equivalent sequential execution!**

# Serializability Order Numbers (SONs)

time ↓

| TX1 | | TX2 | | TX3 | |
|---|---|---|---|---|---|
| *[0:∞]* | **Start TX1;** | *[0:∞]* | **Start TX2;** | *[0:∞]* | **Start TX3;** |
| *[0:∞]* | **Read(A);** | *[0:∞]* | **Read(B);** | | . |
| | . | | . | *[0:∞]* | **Write(B);** |
| | . | *[0:3]* | ◄········ | *SON=3* | **Commit;** |
| | . | *[0:3]* | **Write(A);** | | |
| *[0:2]* | ◄········ | *SON=2* | **Commit;** | | |
| *[3:2]* | **Read(B);** | | | | |

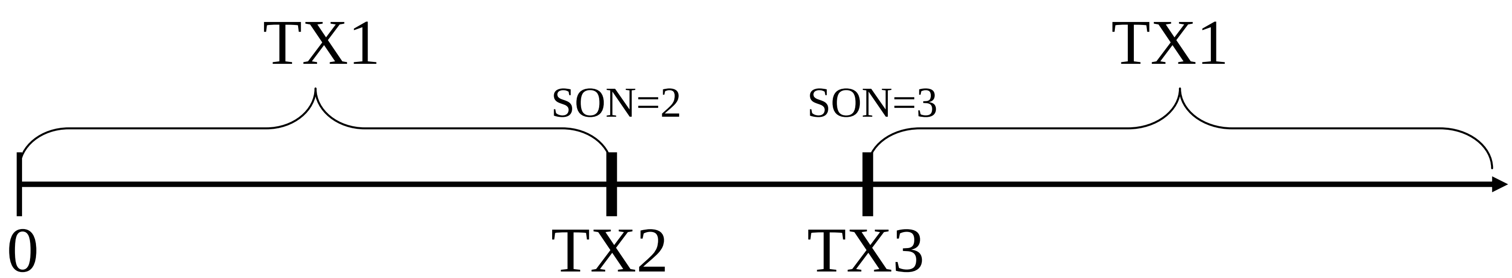



**Equivalent sequential execution!**

# Serializability Order Numbers (SONs)

time ↓

| | | | | | |
|---|---|---|---|---|---|
| *[0:∞]* | **Start TX1;** | *[0:∞]* | **Start TX2;** | *[0:∞]* | **Start TX3;** |
| *[0:∞]* | **Read(A);** | *[0:∞]* | **Read(B);** | | . |
| | . | | . | *[0:∞]* | **Write(B);** |
| | . | *[0:3]* ← | | *SON=3* | **Commit;** |
| | . | *[0:3]* | **Write(A);** | | |
| *[0:2]* ← | | *SON=2* | **Commit;** | | |
| *[3:2]* | **Read(B);** | | | | |
| | **Abort;** | | | | |




**Equivalent sequential execution!**

# Multi-Versioning

- If reading the latest version of data cannot be serialized, perhaps reading an older version can be.
  - Keep older versions for each object.
  - Update SON ranges based on the version read.

- Avoids having to reject operations that arrive to late to be serializable.

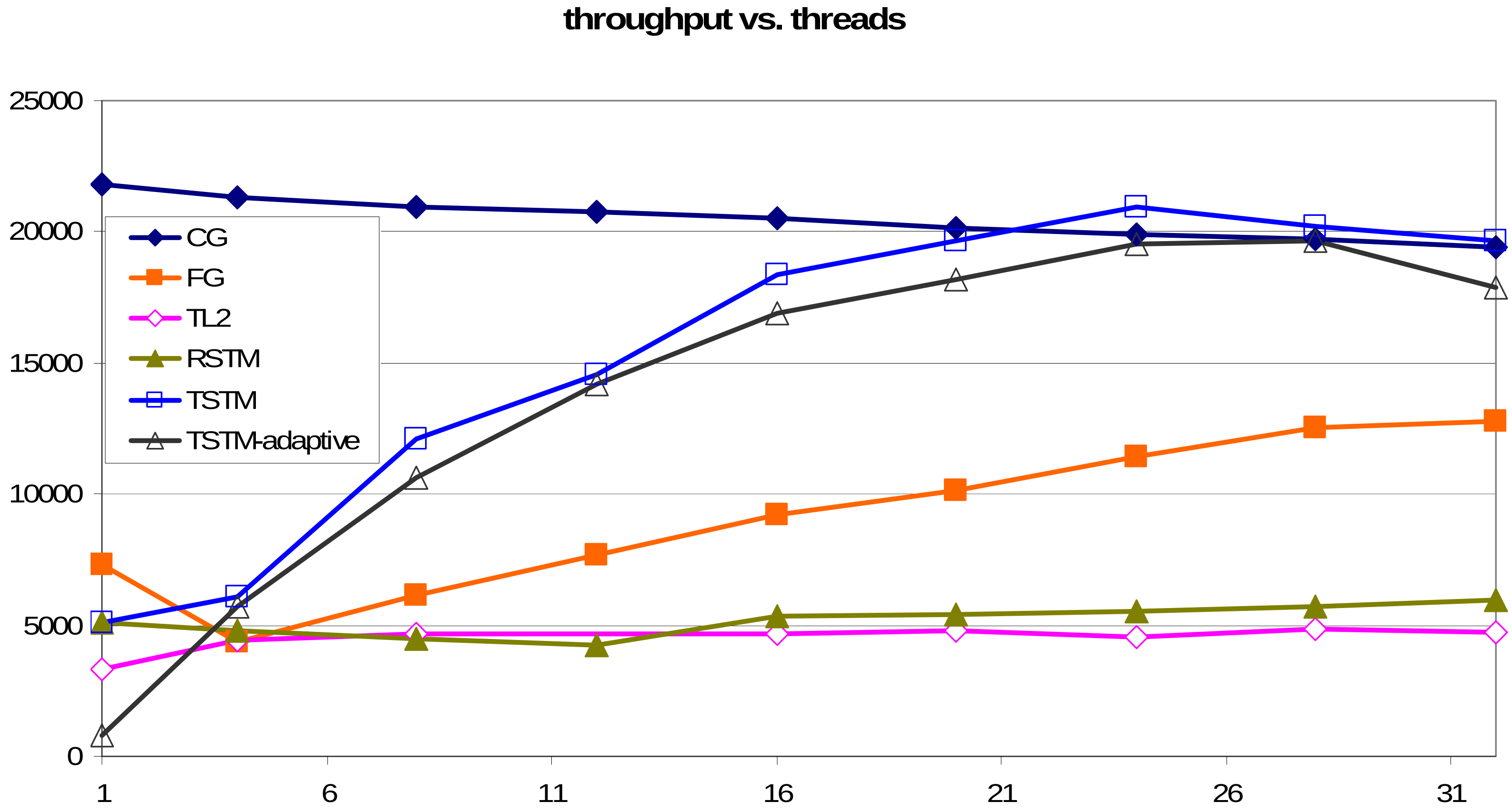
# Toronto STM (TSTM)

- Implements CS with multi-versioning.
- C++, Object-based, indirection with wrappers, obstruction-free, visible readers.
- High overheads
  - Accessing SON ranges.
  - Iterating reader transactions.
  - Pays off when aborts are important.
- Adaptive TSTM
  - Switches consistency model based on abort rates: CS vs. 2PL

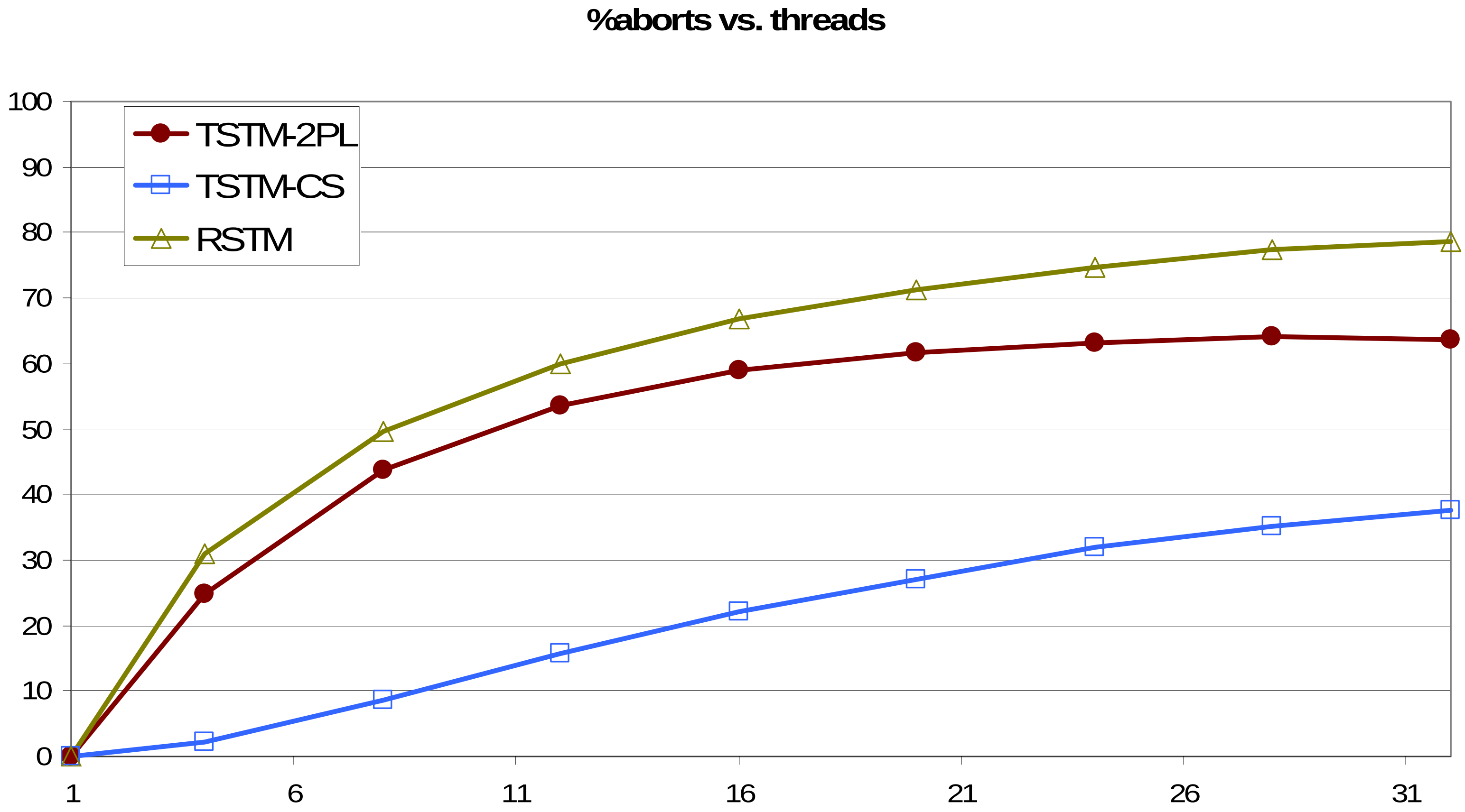
# Experimental Evaluation

- Platform: Sun-Fire 15K, 72 UltraSPARCIII 1.2 GHz proc.
  - Use only 32 processors.
  - Single thread per processor, no nested transactions.
- Benchmarks:
  - LinkedList (LL), RB Tree (RBT), Graph (GR).
  - Workload: 1:1:1 insert/delete/lookup.
  - Value ranges: 16K(LL), 64K(RBT), 4K(GR).
- Evaluated algorithms:
  - Fine-grain locking (FG), Coarse-grain locking (CG).
  - TSTM, TSTM-adaptive.
  - RSTM, TL2.

# LinkedList

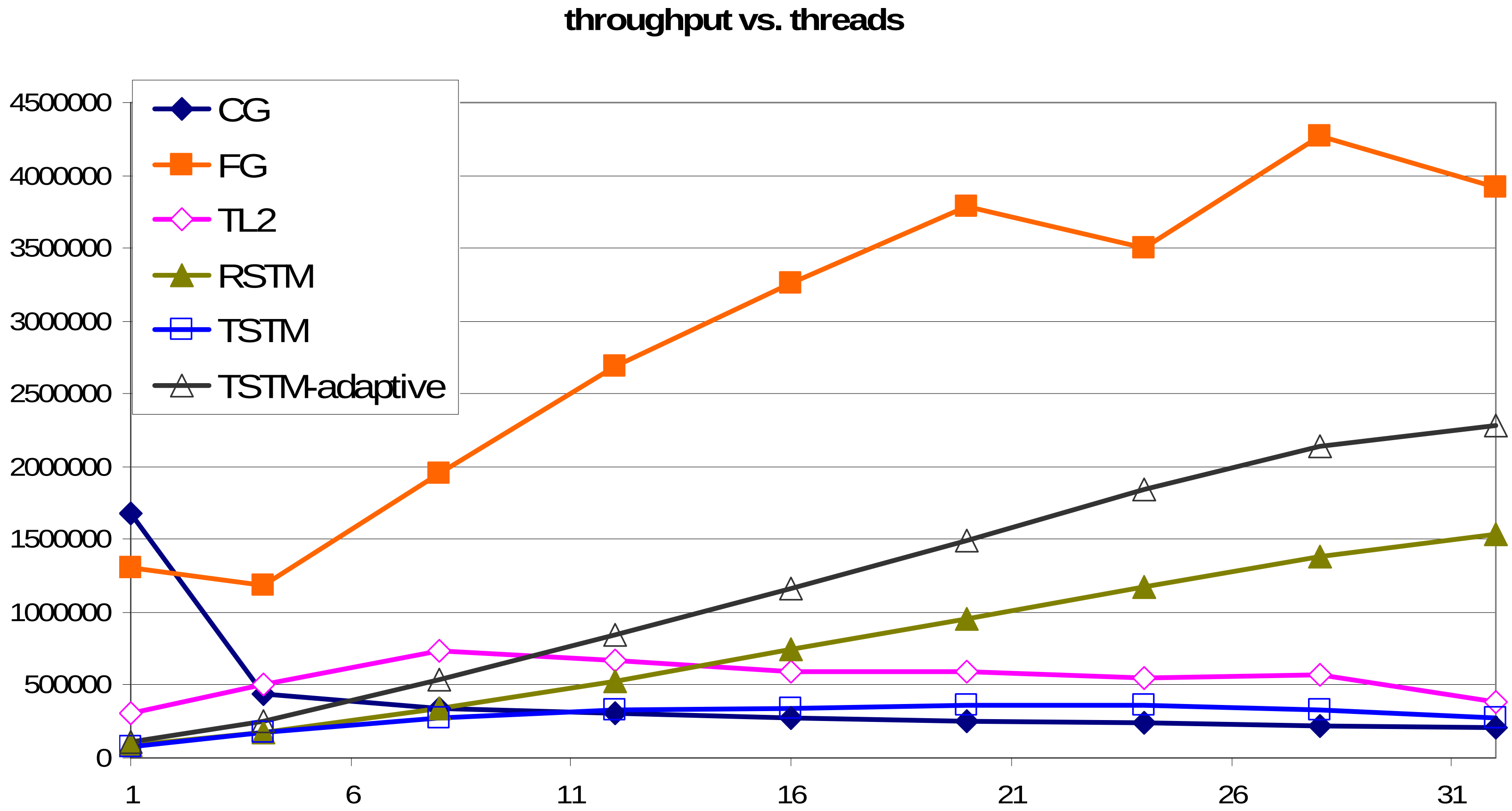

throughput vs. threads
25000
20000
15000
10000
5000
0
1
6
11
16
21
26
31
CG
FG
TL2
RSTM
TSTM
TSTM-adaptive

# LinkedList



%aborts vs. threads
100
90
80
70
60
50
40
30
20
10
0
1
6
11
16
21
26
31
TSTM-2PL
TSTM-CS
RSTM

# RBT


throughput vs. threads
4500000
4000000
3500000
3000000
2500000
2000000
1500000
1000000
500000
0
1
6
11
16
21
26
31
CG
FG
TL2
RSTM
TSTM
TSTM-adaptive

# Related Work

- Early Release & Open Nesting & Collection Classes
  - M. Herlihy, V. Luchangco, M. Moir, and I. William N. Scherer, "Software transactional memory for dynamic-sized data structures,"in *Proc. of PODC*, pp. 92–101, 2003.
  - Y. Ni, V. S. Menon, A.-R. Adl-Tabatabai, A. L. Hosking, R. L. Hudson, J. E. B. Moss, B. Saha, and T. Shpeisman, "Open nesting in software transactional memory," in *Proc. of PPoPP*, pp. 68–78, 2007.
  - B. D. Carlstrom, A. McDonald, M. Carbin, C. Kozyrakis, and K. Olukotun, "Transactional collection classes," in *In Prof. of PPoPP*, pp. 56–67, 2007.
- Snapshot Isolation
  - T. Riegel, P. Felber, and C. Fetzer, "A lazy snapshot algorithm with eager validation," in *Proceedings of the 20th International Symposium on Distributed Computing, DISC 2006*, vol. 4167, pp. 284–298, Sep 2006.
- Z-linearizability
  - T. Riegel, H. Sturzrehm, P. Felber, and C. Fetzer, "From causal to z-linearizable transactional memory," Tech. Rep. RR-I-07-02.1, Universite de Neuchatel, Institut d'Informatique, Feb 2007.
- Databases
  - J. Lindstrom and K. Raatikainen, "Dynamic adjustment of serialization order using timestamp intervals in real-time DBMS," in *Proc. of the Int'l Conf. on Real-Time Computing Systems and Applications*, 1999.

# Conclusions

- 2PL is too restrictive for applications with long transactions and high data sharing.
- CS gives better performance despite its high overheads
  - 2.5 times better throughput for LL.
  - 4 times better throughput for GR.
- Poor performance for applications with short transactions
- Dynamically adapting the model yields the best performance
- We need to look into more models
  - Is adapting between several different models viable?