MC 302 – DBMS: Concurrency Control 2

Goonjan Jain

Dept. of Applied Mathematics

Delhi Technological University

Timestamp Allocation

- Each transaction T_i is assigned a unique fixed timestamp that is monotonically increasing.
  - Let **TS(**Ti**)** be the timestamp allocated to transaction Ti
  - Different schemes assign timestamps at different times during the transaction.
- Multiple implementation strategies:
  - System Clock.
  - Logical Counter.
  - Hybrid.

Timestamp Ordering Concurrency Control

- Use these timestamps to determine the serializability order.
- If **TS(**Ti**)** < **TS(**Tj**)**, then the DBMS must ensure that the execution schedule is equivalent to a serial schedule where Ti appears before Tj.

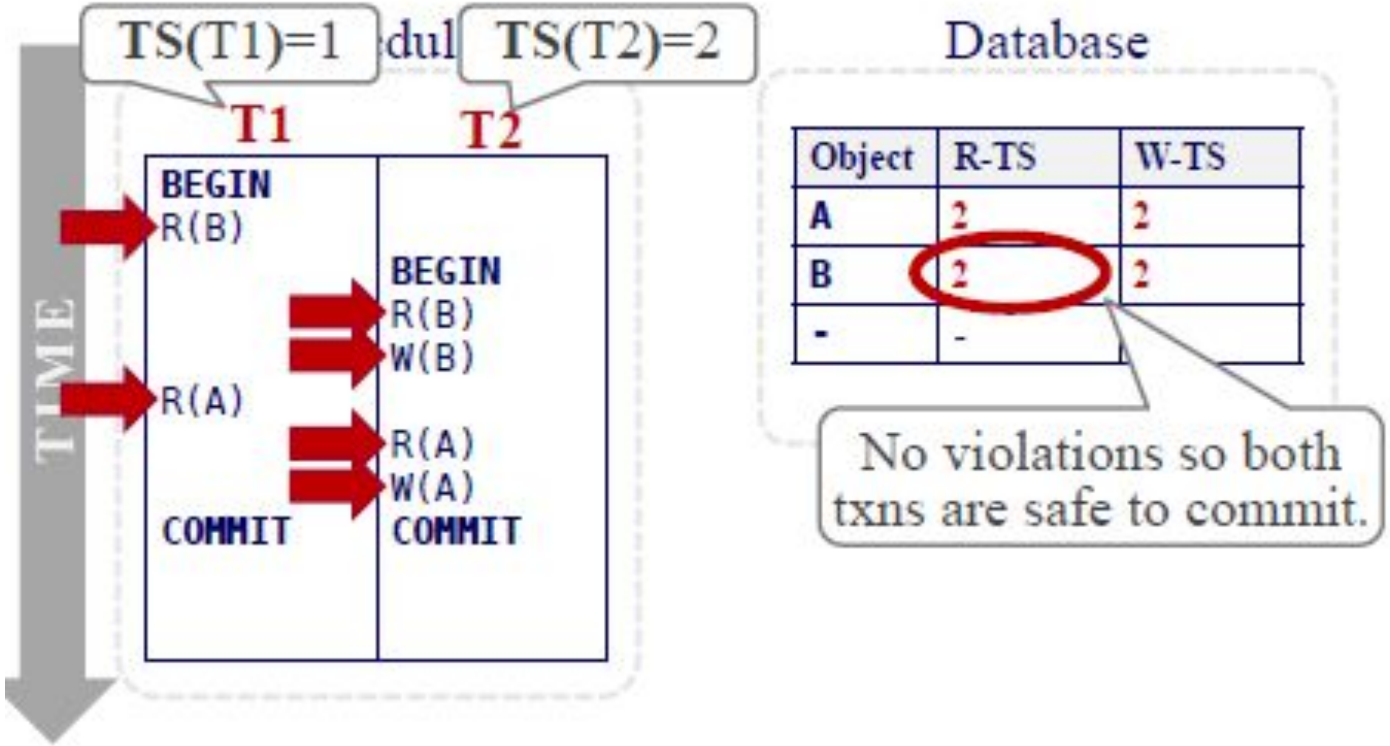
Basic T/O

- Transactions read and write objects without locks.
- Every object X is tagged with timestamp of the last transaction that successfully did read/write:
  - **W-TS(**X**)** – Write timestamp on X
  - **R-TS(**X**)** – Read timestamp on X
- Check timestamps for every operation:
  - If transaction tries to access an object “from the future”, it aborts and restarts.

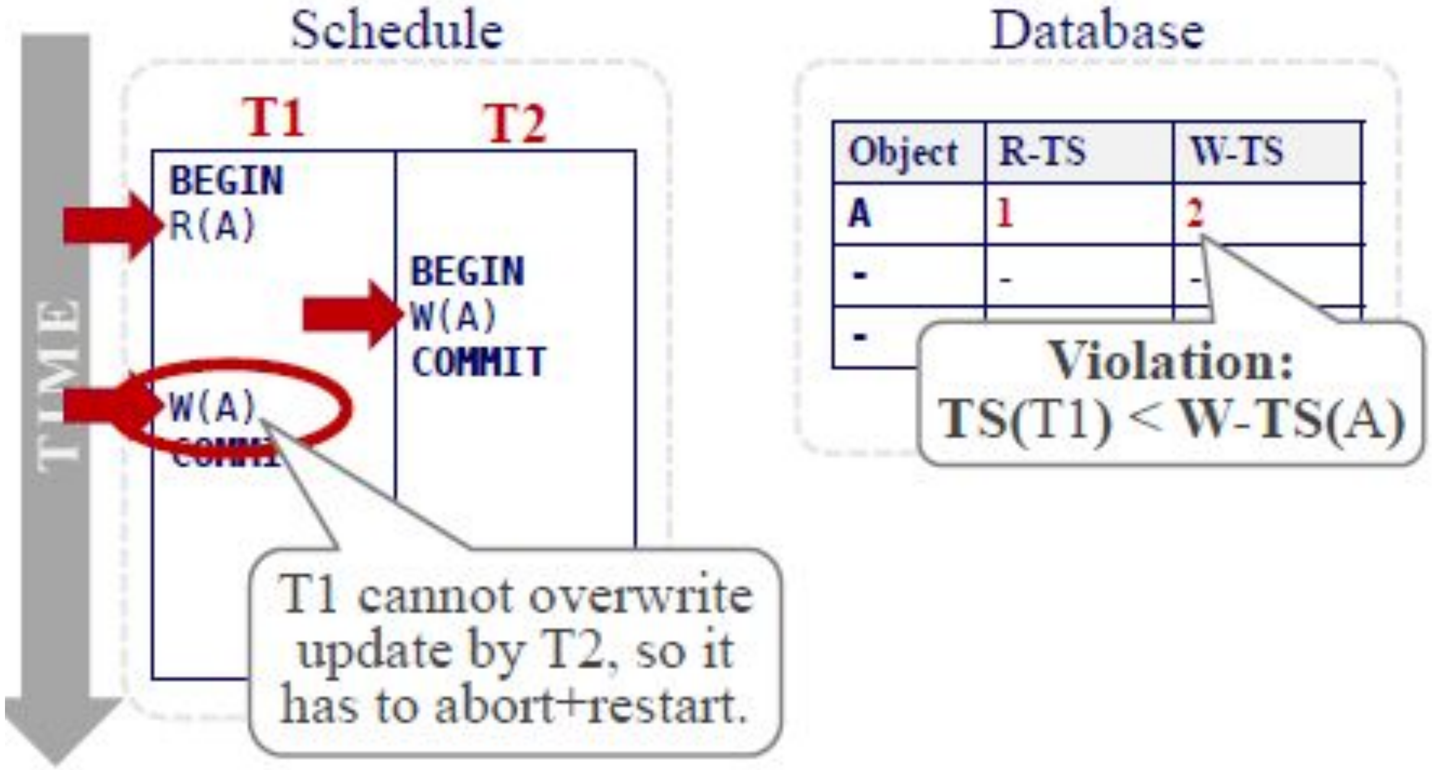
Basic T/O – Reads and Writes

- Reads
- If **TS(**Ti**)** < **W-TS(**X**)**, this violates timestamp order of Ti w.r.t. writer of X.
  - Abort Ti and restart it (with same TS? why?)
- Else:
  - Allow Ti to read X.
  - Update **R-TS(**X**)** to **max(R-TS(**X**)**, **TS(**Ti**))**
  - Have to make a local copy of X to ensure repeatable reads for Ti.
- Writes
- If **TS(**Ti**)** < **R-TS(**X**)** or **TS(**Ti**)** < **W-TS(**X**)**
  - Abort and restart Ti.
- Else:
  - Allow Ti to write X and update **W-TS(**X**)**
  - Also have to make a local copy of X to ensure repeatable reads for Ti.

Basic T/O Example



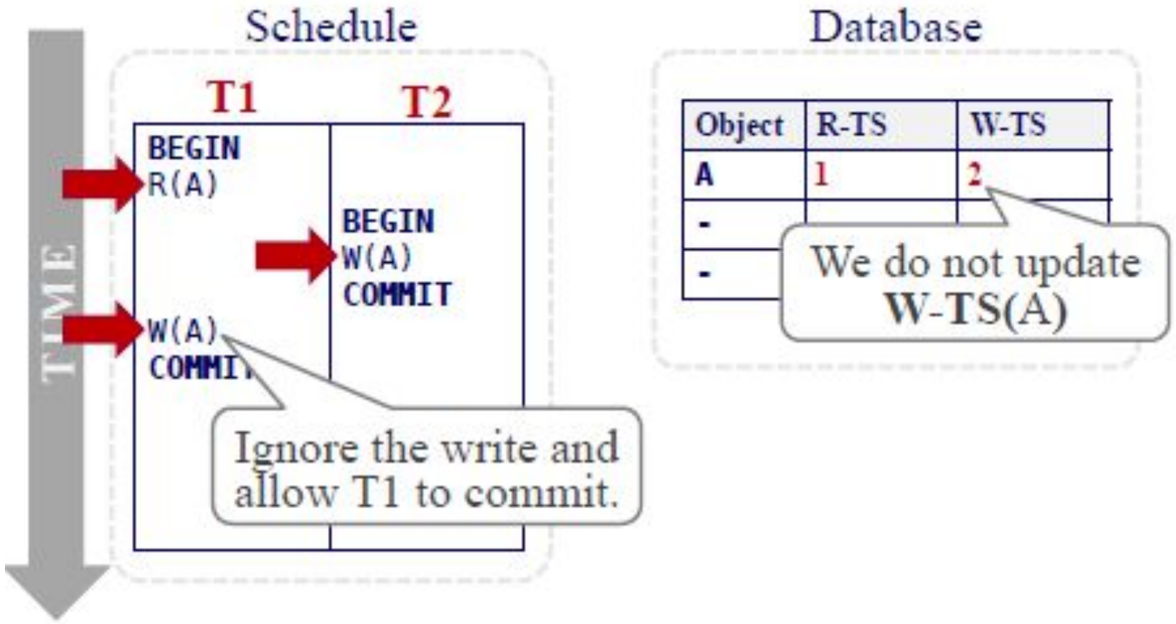
Basic T/O Example 2



Basic T/O: Thomas Write Rule

- If **TS(**Ti**)** < **R-TS(**X**)**:
  - Abort and restart Ti.
- If **TS(**Ti**)** < **W-TS(**X**)**:
  - **Thomas Write Rule:** Ignore the write and allow the txn to continue.
  - This violates timestamp order of Ti
- Else:
  - Allow Ti to write X and update **W-TS(**X**)**

Basic T/O: Thomas Write Rule

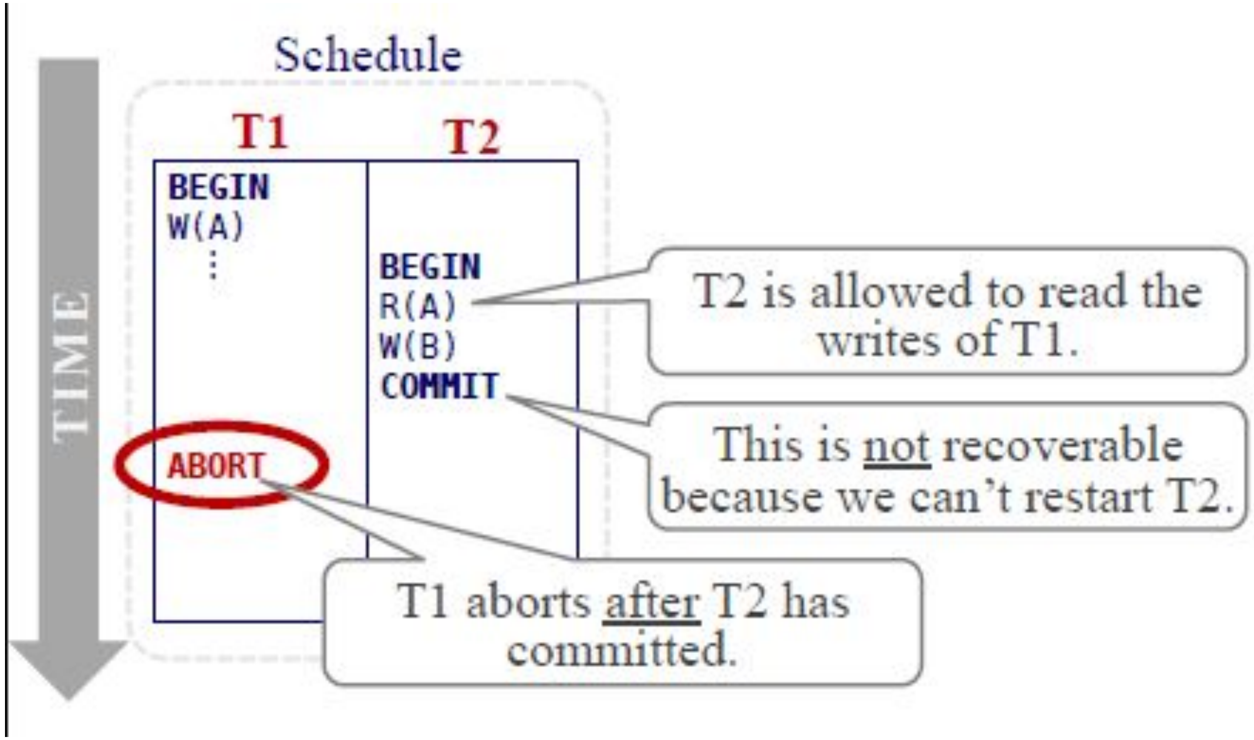


Basic T/O

- Ensures conflict serializability if you don’t use the Thomas Write Rule.
- No deadlocks because no transaction ever waits.
- Possibility of starvation for long transactions if short transactions keep causing conflicts.
- Permits schedules that are not ***recoverable****.*

Recoverable Schedules

- Transactions commit only after all transactions whose changes they read, commit.



Basic T/O: Performance Issues

- High overhead from copying data to transaction's workspace and from updating timestamps.
- Long running transactions can get starved.
- Suffers from timestamp bottleneck.

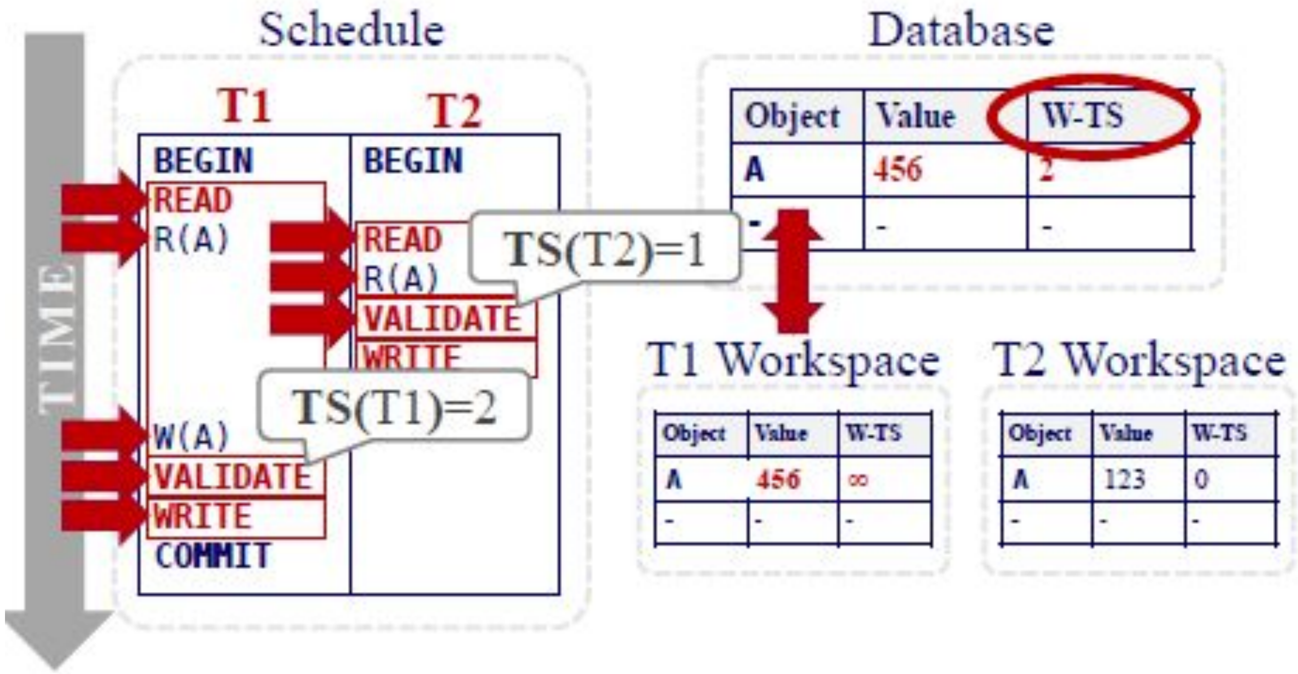
Optimistic Concurrency Control

- Assumption: Conflicts are rare
- Forcing transactions to wait to acquire locks adds a lot of overhead.
- Optimize for the no-conflict case.

OCC Phases:

- **Read:** Track the read/write sets of transactions and store their writes in a private workspace.
- **Validation:** When a transaction commits, check whether it conflicts with other transactions.
- **Write:** If validation succeeds, apply private changes to database. Otherwise abort and restart the transaction.

OCC Example



Schedule
TIME
T1
T2
BEGIN
READ
R(A)
W(A)
VALIDATE
WRITE
COMMIT
BEGIN
READ
R(A)
VALIDATE
WRITE
TS(T2)=1
TS(T1)=2
Database
Object
Value
W-TS
A
456
2
T1 Workspace
Object
Value
W-TS
A
456
∞
T2 Workspace
Object
Value
W-TS
A
123
0

OCC: Validation Phase

- Need to guarantee only serializable schedules are permitted.
- At validation, Ti checks other txns for RW and WW conflicts and makes sure that all conflicts go one way (from older txns to younger txns).

OCC: Serial Validation

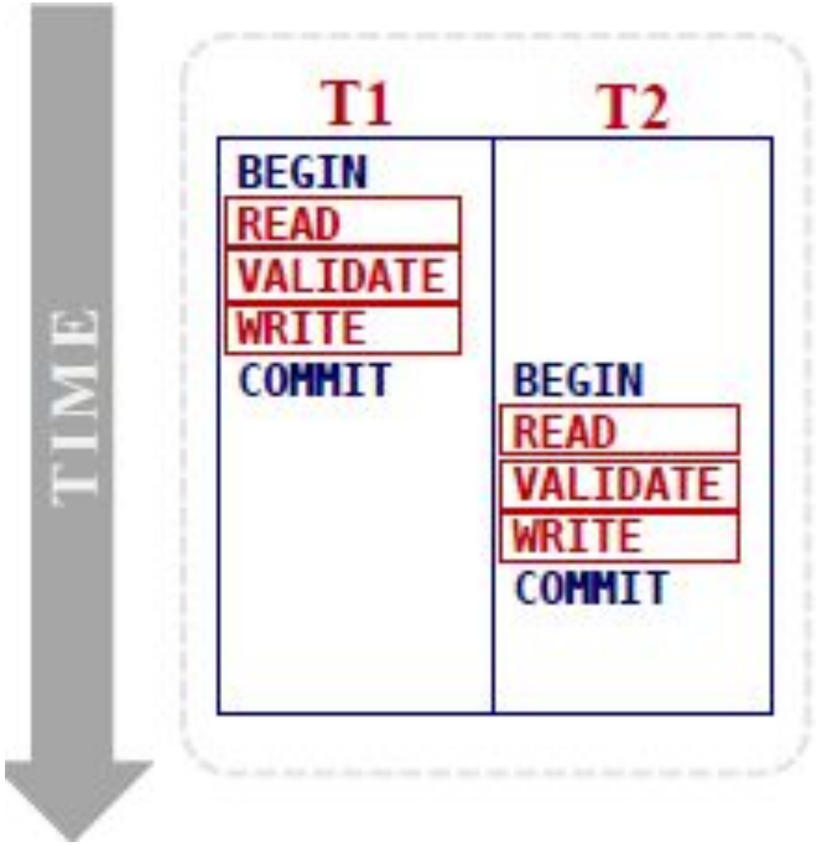
- Maintain global view of all active transactions.
- Record read set and write set while transactions are running and write into private workspace.
- Execute **Validation** and **Write** phase inside a protected critical section.

OCC: Validation Phase

- Each transaction's timestamp is assigned at the beginning of the validation phase.
- Check the timestamp ordering of the committing transaction with all other running transactions.
- If **TS(**Ti**)** < **TS(**Tj**)**, then one of the following three conditions must hold…

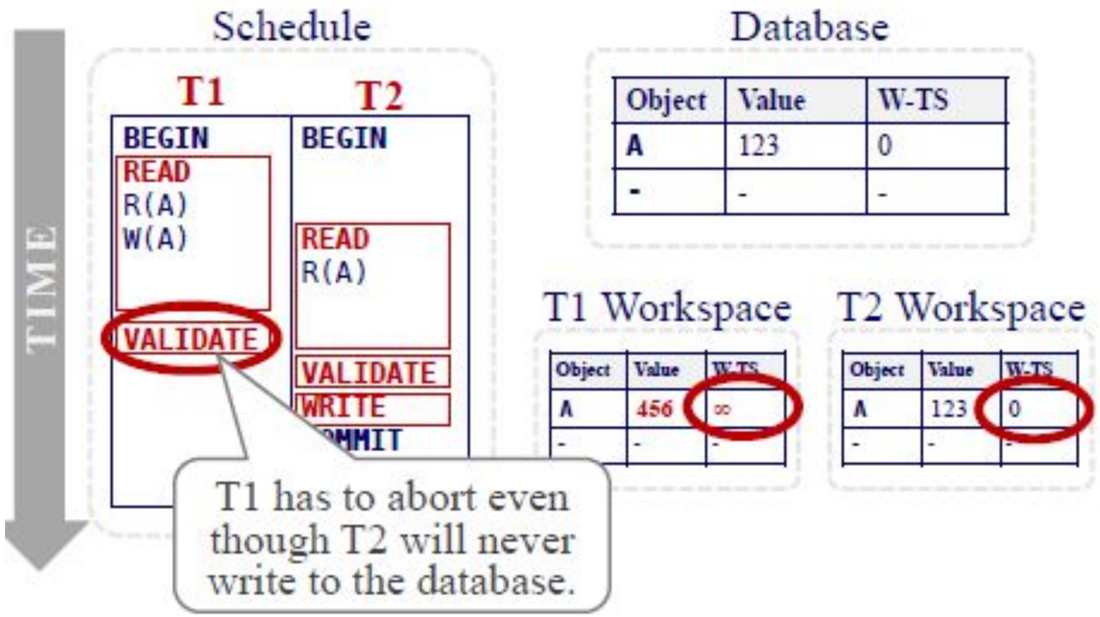
OCC Validation #1

- Ti completes all three phases before Tj begins.

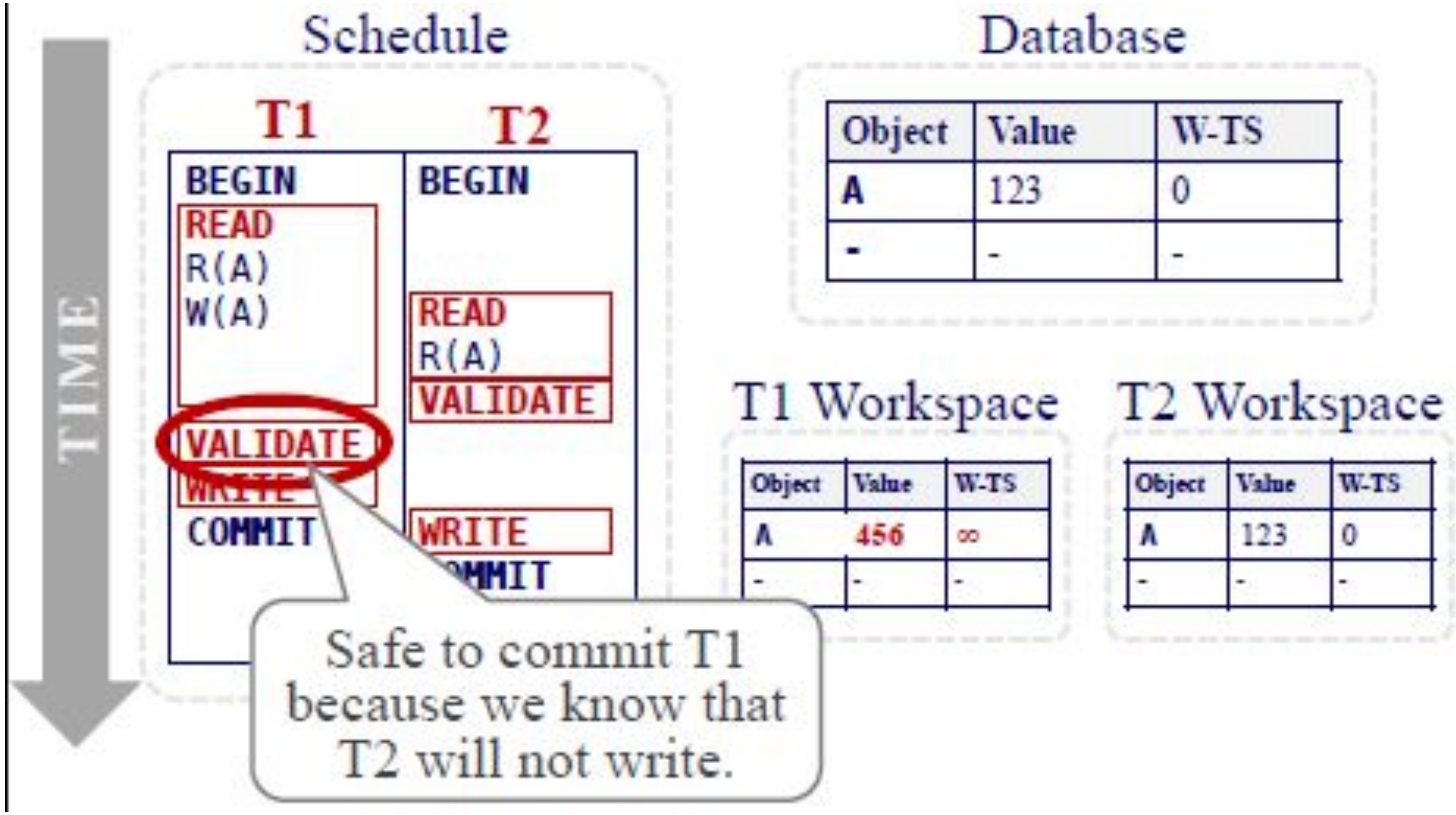


OCC – Validation#2

- Ti completes before Tj starts its **Write** phase, and Ti does not write to any object read by Tj.
  - WriteSet(Ti) ∩ ReadSet(Tj) = Ø

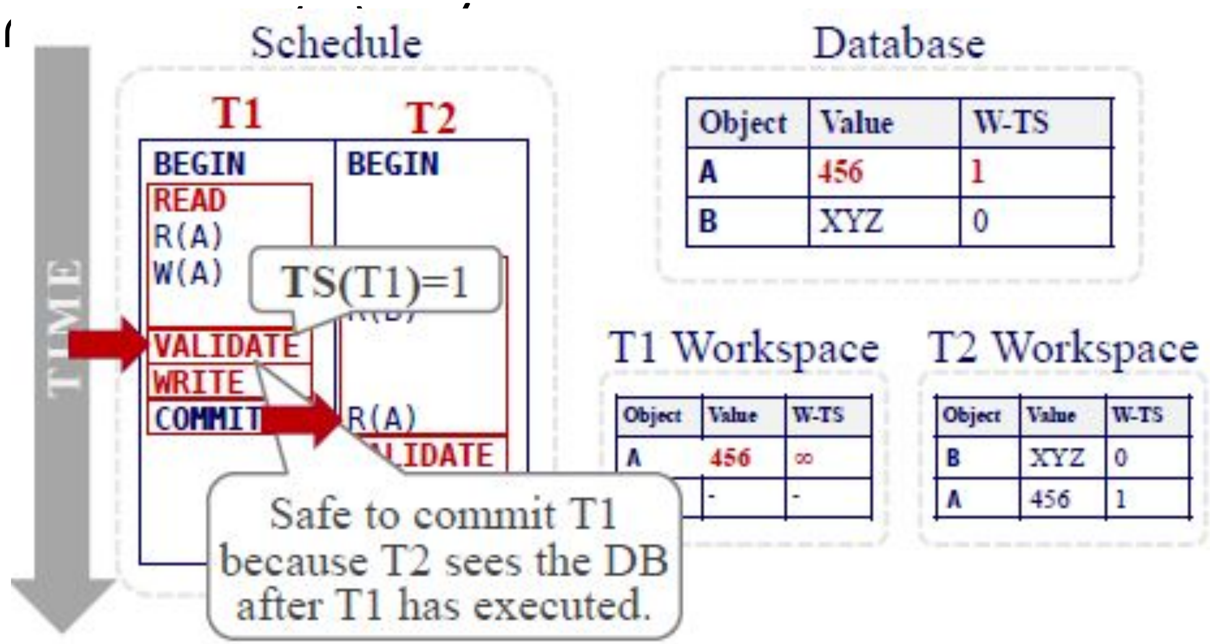


OCC-Validation #2



OCC-Validation#3

- Ti completes its **Read** phase before Tj completes its **Read** phase
- And Ti does not write to any object that is either read or written by Tj:
  - WriteSet(Ti) ∩ ReadSet(Tj) = Ø
  - WriteSet(Ti)



OCC-Observations

- **Q:** When does OCC work well?
- **A:** When # of conflicts is low:
  - All transactions are read-only (ideal).
  - Transactions access disjoint subsets of data.
- If the database is large and the workload is not skewed, then there is a low probability of conflict, so again locking is wasteful.

OCC Performance Issues

- High overhead for copying data locally.
- **Validation/Write** phase bottlenecks.
- Aborts are more wasteful because they only occur *after* a txn has already executed.
- Suffers from timestamp allocation bottleneck.

Multi-Version Concurrency Control

- Writes create new versions of objects instead of in-place updates:
  - Each successful write results in the creation of a new version of the data item written.
- Use write timestamps to label versions.
  - Let X_k denote the version of X where for a given trnasaction T_i: **W-TS(**X_k**)** $\leq$ **TS(**T_i**)**

MVCC – Reads and Writes

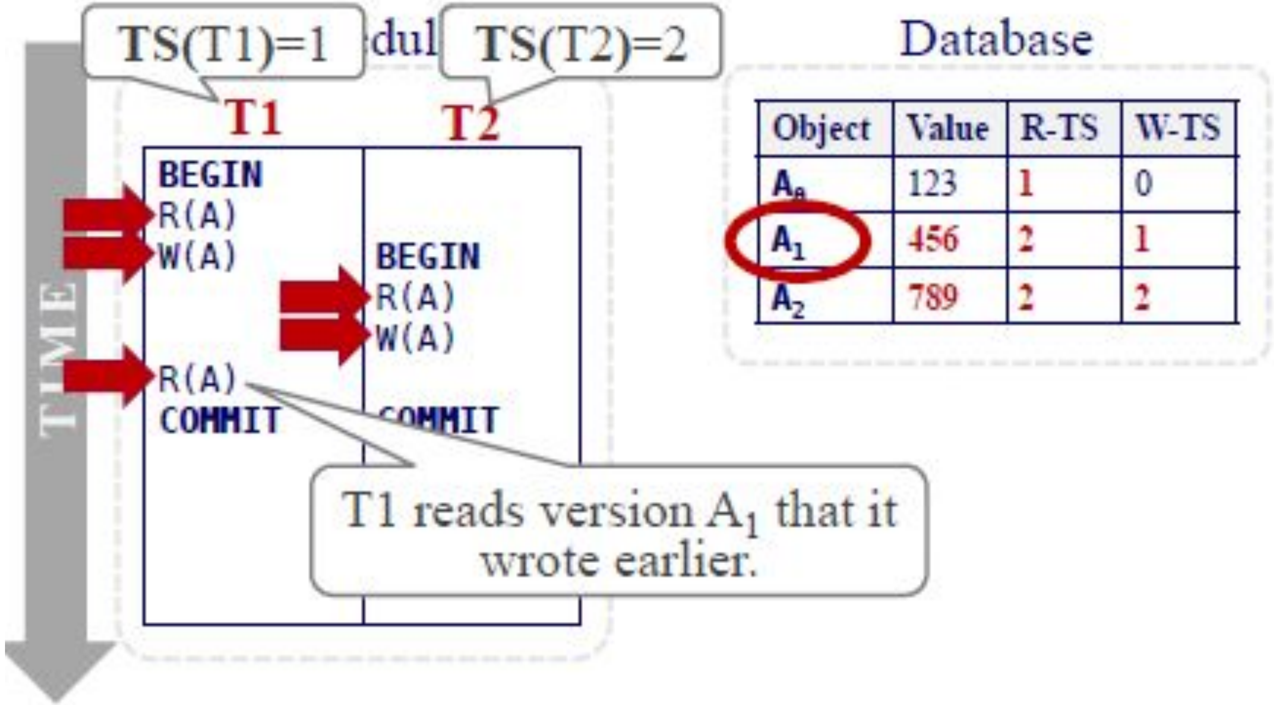
Reads

- Any read operation sees the latest version of an object from right before that transaction started.
- Every read request can be satisfied without blocking the transaction.
- If **TS(**T_i**)** > **R-TS(**X_k**)**:
  - Set **R-TS(**X_k**) = TS(**T_i**)**

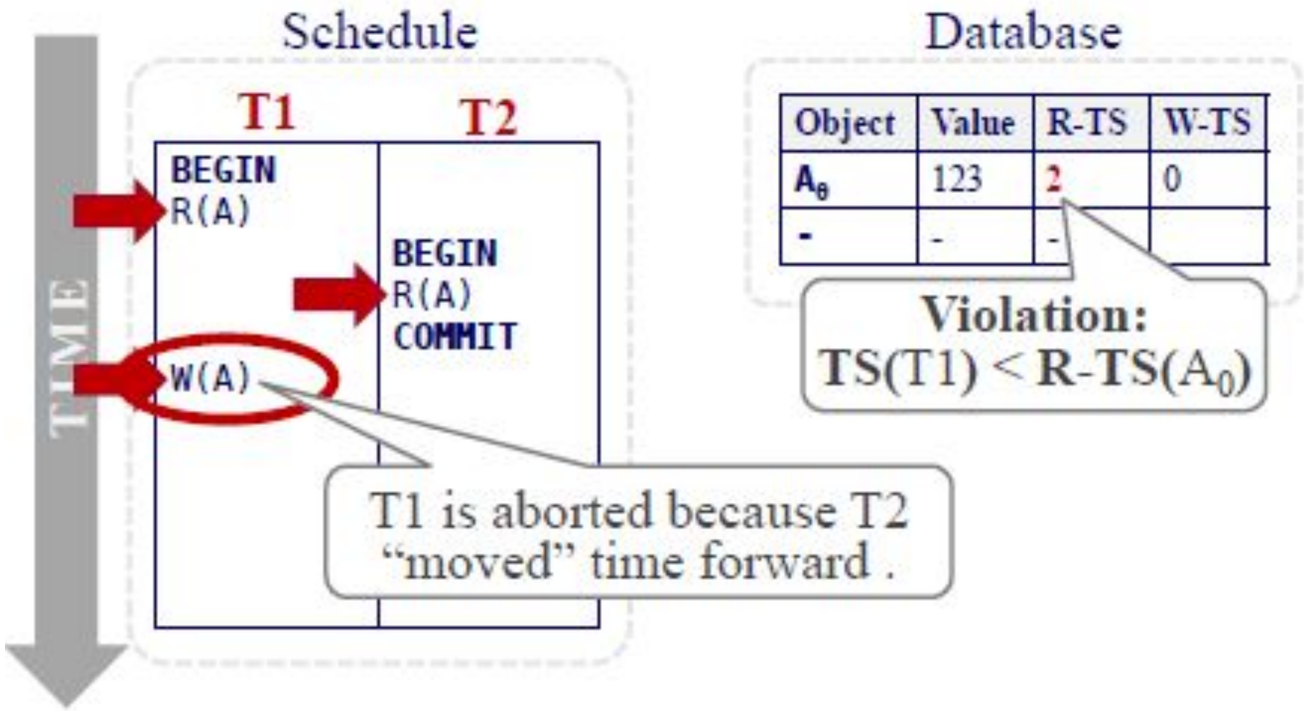
Writes

- If **TS(**T_i**)** < **R-TS(**X_k**)**:
  - Abort and restart T_i.
- If **TS(**T_i**)** = **W-TS(**X_k**)**:
  - Overwrite the contents of X_k.
- Else:
  - Create a new version of X_{k+1} and set its write timestamp to **TS(**T_i**)**.

MVCC – Example #1



MVCC – Example #2



MVCC

- Can still incur cascading aborts because a transaction sees uncommitted versions from transactions that started before it did.
- Old versions of tuples accumulate.
- The DBMS needs a way to remove old versions to reclaim storage space.