



Lecture 14: Timestamp Ordering

CREATING THE NEXT®

Administrivia

- Mid-term Exam on Mar 07 (topics covered until Feb 23)
- Project Proposal on Mar 02
- Project (extra credit: 10%)

Deliverables

- Proposal: 1-page report
- Checkpoint: 2-page report
- Final Presentation: 2-page report + 5 min presentation (subset of teams)

Project - Proposal

- Each proposal must discuss:
 - ▶ What is the problem being addressed by the project?
 - ▶ Why is this problem important?
 - ▶ How will the team solve this problem?

Project - Presentations

- Five minute presentation on the final status of your project.
- You'll want to include any performance measurements or benchmarking numbers for your implementation.
- Demos are always hot too.
- Prizes for top teams picked by the class.

Today's Agenda

Recap

Basic Timestamp Ordering

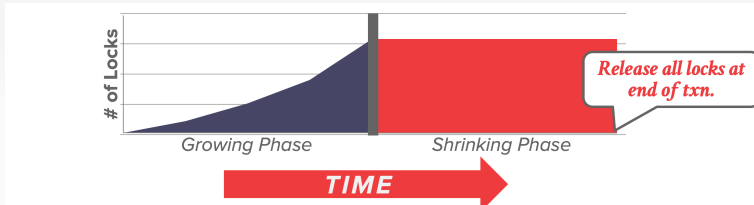
Partition-based Timestamp Ordering

Conclusion

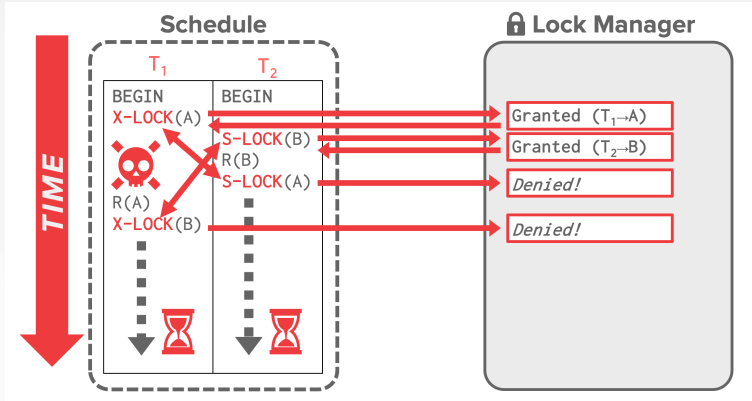
Recap

Strong Strict Two-Phase Locking

- The txn is not allowed to acquire/upgrade locks after the growing phase finishes.
- Allows only conflict serializable schedules, but it is often stronger than needed for some apps.



Deadlocks



2PL Deadlocks

- A **deadlock** is a cycle of transactions waiting for locks to be released by each other.
- Two ways of dealing with deadlocks:
 - ▶ **Approach 1: Deadlock Detection**
 - ▶ **Approach 2: Deadlock Prevention**

2PL: Summary

- 2PL is used in almost all DBMSs.
- Automatically generates correct interleaving:
 - ▶ Locks + protocol (2PL, SS2PL ...)
 - ▶ Deadlock detection + handling
 - ▶ Deadlock prevention

Concurrency Control Approaches

- Two-Phase Locking (2PL)
 - ▶ **Pessimistic approach**
 - ▶ Assumption that collisions are commonplace.
 - ▶ Determine serializability order of conflicting operations at runtime while txns execute.
- Timestamp Ordering (T/O)
 - ▶ **Optimistic approach**
 - ▶ Assumption that collisions between transactions will rarely occur.
 - ▶ Determine serializability order of txns before they execute.

Today's Agenda

- Basic Timestamp Ordering
- Partition-based Timestamp Ordering

Basic Timestamp Ordering

T/O Concurrency Control

- Use timestamps to determine the serializability order of txns.
- If $TS(T_i) < TS(T_j)$, then the DBMS must ensure that the execution schedule is equivalent to a serial schedule where T_i appears before T_j .

Timestamp Allocation

- Each txn T_i is assigned a unique fixed timestamp that is monotonically increasing.
 - ▶ Let $TS(T_i)$ be the timestamp allocated to txn T_i .
 - ▶ Different schemes assign timestamps at different times during the txn.
- Multiple implementation strategies:
 - ▶ Physical system clock (*e.g.*, timezones)
 - ▶ Logical counter (*e.g.*, overflow)
 - ▶ Hybrid

Basic T/O

- Txns read and write objects without locks.
- Every object X is tagged with timestamp of the last txn 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 txn tries to access an object from the future, it aborts and restarts.

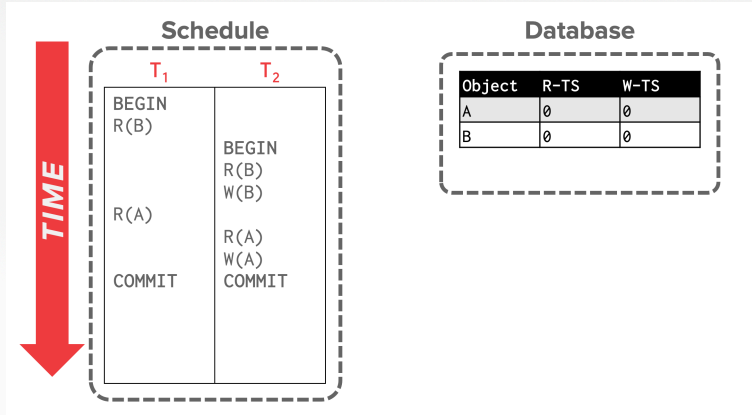
Basic T/O – Reads

- If $TS(T_i) < W - TS(X)$, this violates timestamp order of T_i with regard to the writer of X .
 - ▶ Abort T_i and restart it with a newer TS (so that is later than the writer of X).
- Else:
 - ▶ Allow T_i to read X .
 - ▶ Update $R - TS(X)$ to $\max(R - TS(X), TS(T_i))$
 - ▶ Have to make a local copy of X to ensure repeatable reads for T_i .

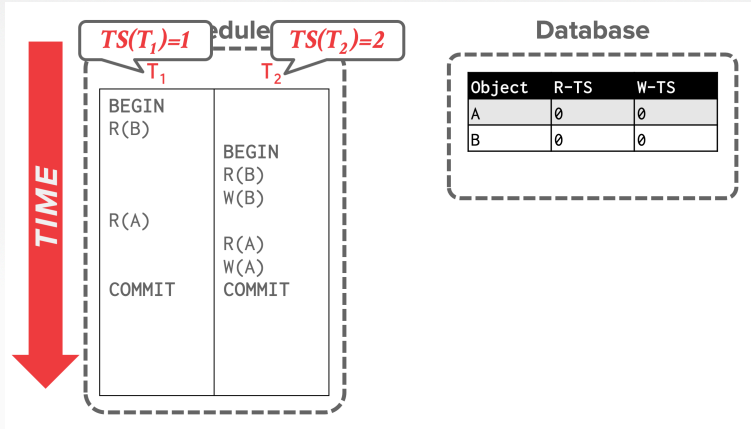
Basic T/O – Writes

- If $TS(T_i) < R - TS(X)$ or $TS(T_i) < W - TS(X)$
 - ▶ Abort and restart T_i .
- Else:
 - ▶ Allow T_i to write X and update $W - TS(X)$
 - ▶ Also have to make a local copy of X to ensure repeatable reads for T_i .

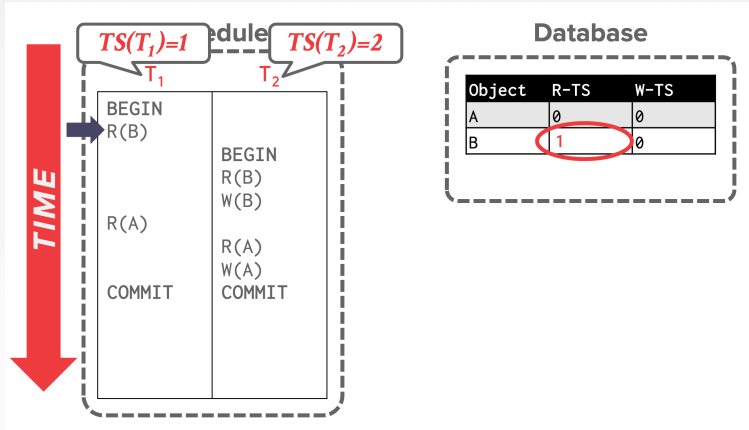
Basic T/O – Example 1



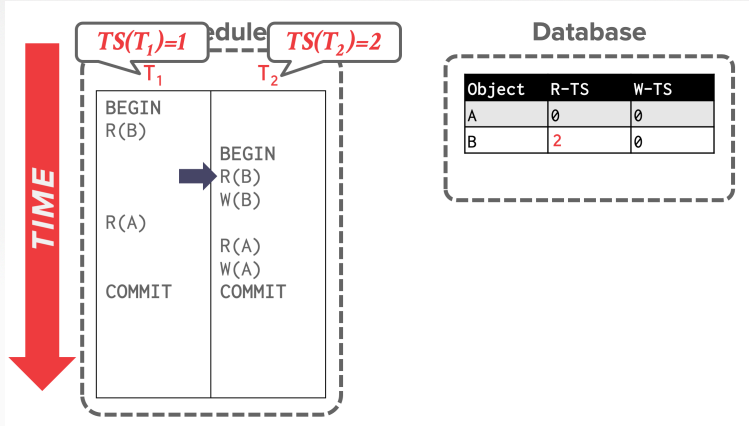
Basic T/O – Example 1



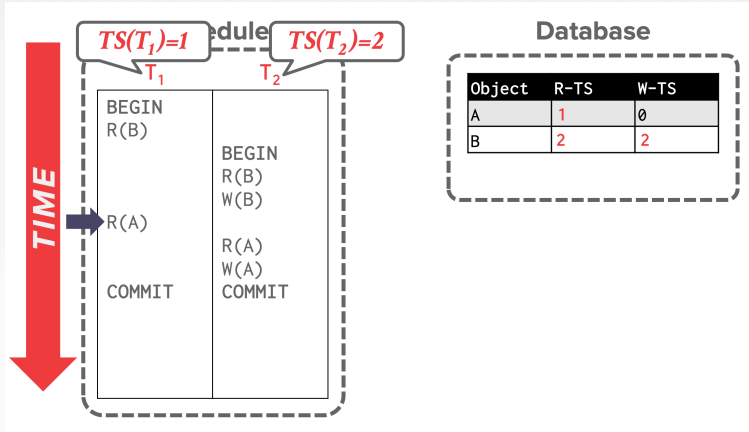
Basic T/O – Example 1



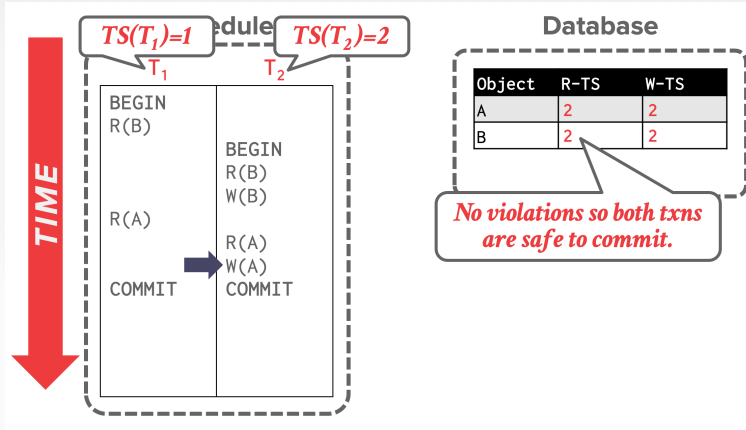
Basic T/O – Example 1



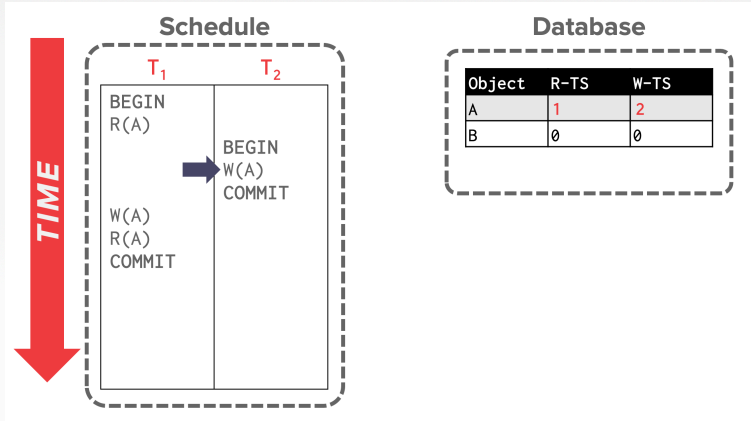
Basic T/O – Example 1



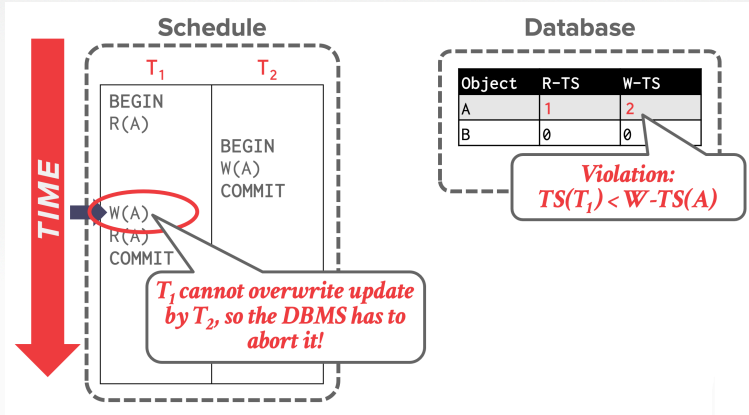
Basic T/O – Example 1



Basic T/O – Example 2



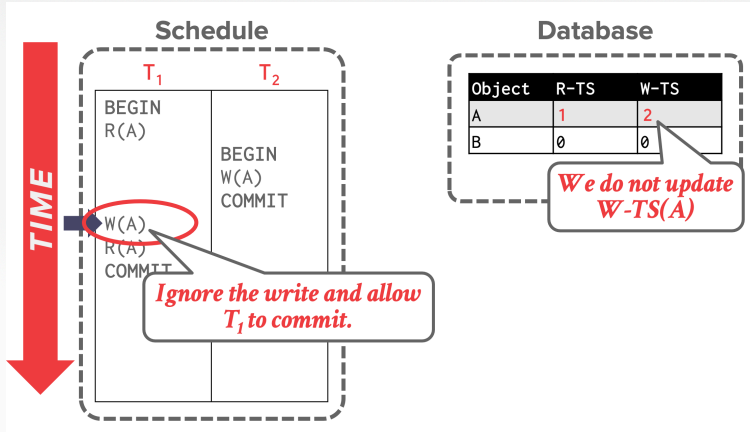
Basic T/O – Example 2



Thomas Write Rule

- If $TS(T_i) < R - TS(X)$:
 - ▶ Abort and restart T_i .
- If $TS(T_i) < W - TS(X)$:
 - ▶ **Thomas Write Rule**: Ignore the write, make a local copy, and allow the txn to continue.
 - ▶ This violates timestamp order of T_i .
- Else:
 - ▶ Allow T_i to write X and update $W - TS(X)$

Basic T/O – Example 2



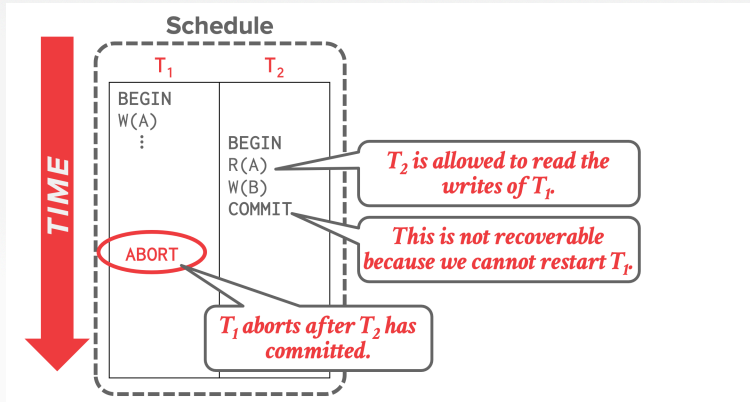
Basic T/O

- Generates a schedule that is conflict serializable if you do **not** use the Thomas Write Rule.
 - ▶ No deadlocks because no txn ever waits.
 - ▶ Possibility of starvation for long txns if short txns keep causing conflicts.
- Permits schedules that are **not recoverable**.

Recoverable Schedules

- A schedule is **recoverable** if txns commit only after all txns whose changes they read, commit.
- Otherwise, the DBMS cannot guarantee that txns read data that will be restored after recovering from a crash.

Recoverable Schedules



Basic T/O – Performance Issues

- High overhead from copying data to txn's local workspace and from updating timestamps.
- Long running txns can get starved.
 - ▶ The likelihood that a txn will read something from a newer txn increases.

Observation

- When a txn commits, the T/O protocol checks to see whether there is a conflict with concurrent txns.
 - ▶ This requires latches.
- If you have a lot of concurrent txns, then this is slow even if the conflict rate is low.

Partition-based Timestamp Ordering

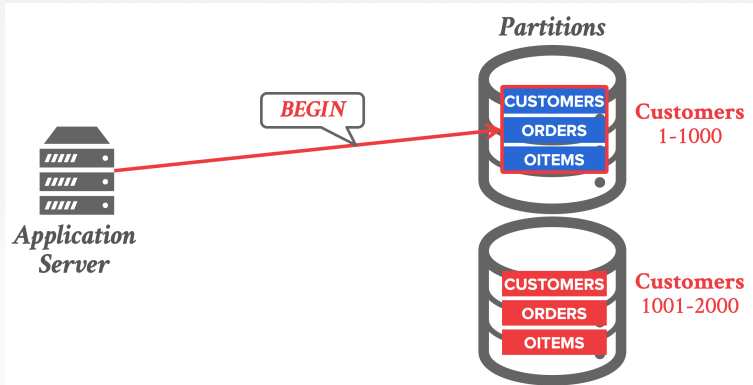
Partition-based T/O

- Split the database up in disjoint subsets called horizontal partitions (aka shards).
- Use timestamps to order txns for serial execution at each partition.
 - ▶ Only check for conflicts between txns that are running in the same partition.

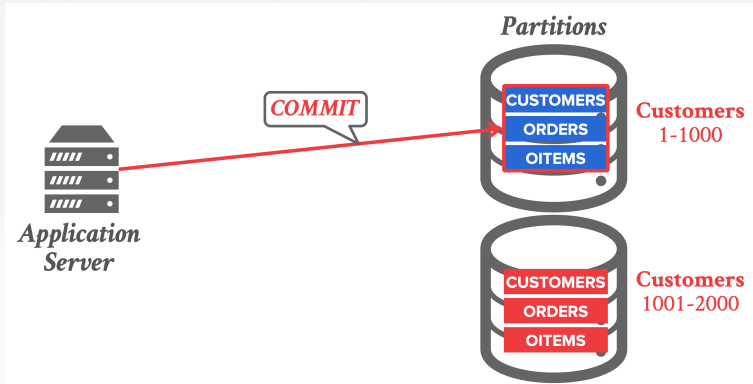
Database Partitioning

```
CREATE TABLE customer (  
  c_id INT PRIMARY KEY,  
  c_email VARCHAR UNIQUE,  
);  
  
CREATE TABLE orders (  
  o_id INT PRIMARY KEY,  
  o_c_id INT REFERENCES customer (c_id) --- Foreign key  
);  
  
CREATE TABLE oitems (  
  oi_id INT PRIMARY KEY,  
  oi_o_id INT REFERENCES orders (o_id),  
  o_c_id INT REFERENCES orders (o_c_id) --- Foreign key  
);
```

Horizontal Partitioning



Horizontal Partitioning



Partition-based T/O

- Txns are assigned timestamps based on when they arrive at the DBMS.
- Partitions are protected by a **single lock**:
 - ▶ Each txn is queued at the partitions it needs.
 - ▶ The txn acquires a partition's lock if it has the lowest timestamp in that partition's queue.
 - ▶ The txn starts when it has all of the locks for all the partitions that it will read/write.
- **Examples:** VoltDB, FaunaDB

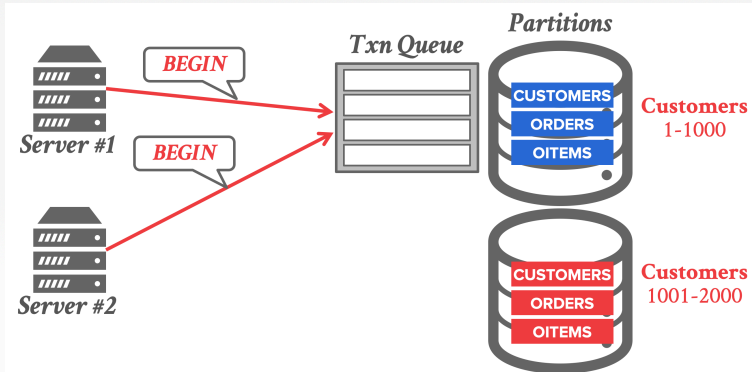
Partition-based T/O – Reads

- Txns can read anything that they want at the partitions that they have locked.
- If a txn tries to access a partition that it does not have the lock, it is **aborted + restarted**.

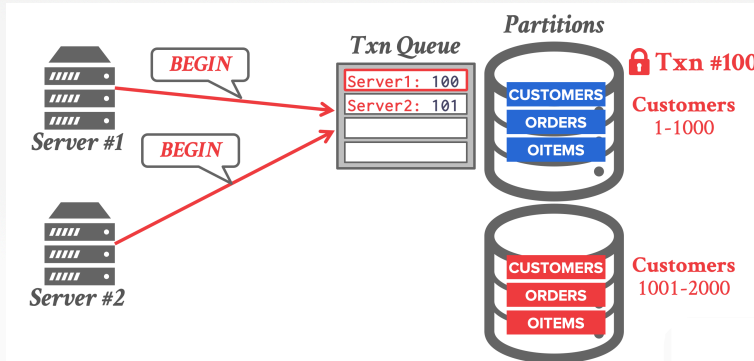
Partition-based T/O – Writes

- All updates occur in place (*i.e.*, no private workspace).
 - ▶ Maintain a separate in-memory buffer to undo changes if the txn aborts.
- If a txn tries to write to a partition that it does not have the lock, it is aborted + restarted.

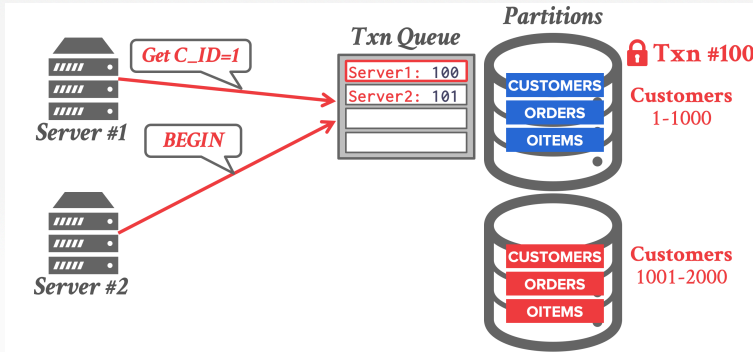
Partition-based T/O



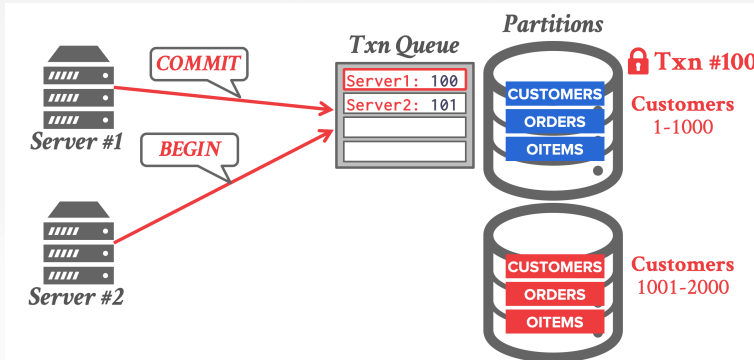
Partition-based T/O



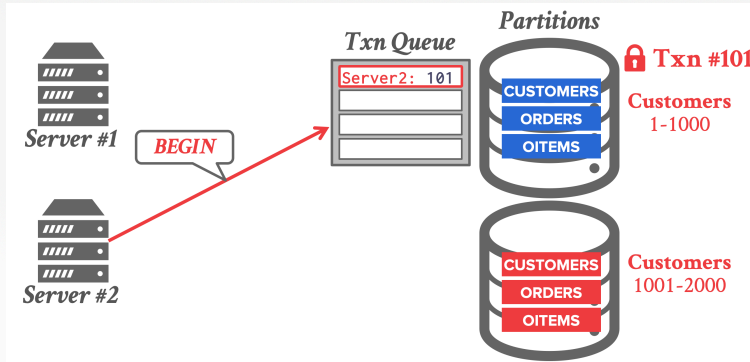
Partition-based T/O



Partition-based T/O



Partition-based T/O



Partition-based T/O – Performance Issues

- Partition-based T/O protocol is fast if:
 - ▶ The DBMS knows what partitions the txn needs before it starts.
 - ▶ Most (if not all) txns only need to access a single partition.
- Multi-partition txns causes partitions to be **idle** while txn executes.
 - ▶ Stored procedures
 - ▶ Reconnaissance mode

Conclusion

Parting Thoughts

- Every concurrency control can be broken down into the basic concepts that I have described in the last two lectures.
 - ▶ Two-Phase Locking (2PL): Assumption that collisions are commonplace
 - ▶ Timestamp Ordering (T/O): Assumption that collisions are rare.
- I am not showing benchmark results because I don't want you to get the wrong idea.

Next Class

- Optimistic Concurrency Control
- Isolation Levels