• Efficient read-only transactions in strictly serializable systems
  • Strict serializability is desirable but costly!
  • Reads are prevalent! (340x more than write txns)
  • Efficient rotxns → good system overall performance

Recap: Ideas Behind Read-Only Txns

• Tag writes with physical timestamps upon commit
  • Write txns are strictly serializable, e.g., 2PL

• Read-only txns return the writes, whose commit timestamps precede the reads’ current time
  • Rotxns are one-round, lock-free, and never abort

Recap: TrueTime

• Timestamping writes must enforce the invariant
  • If T2 starts after T1 commits (finishes), then T2 must have a larger timestamp

• TrueTime: partially-synchronized clock abstraction
  • Bounded clock skew (uncertainty)
  • TT.now() → [earliest, latest]; earliest <= T_{abs} <= latest
  • Uncertainty (ε) is kept short

• TrueTime enforces the invariant by
  • Use at least TT.now().latest for timestamps
  • Commit wait
Enforcing the Invariant with TT

If T2 starts after T1 commits (finishes), then T2 must have a larger timestamp

Let T1 write $S_B$ and T2 write $S_A$

- $\text{T.abs} = 3$
- $\text{T.abs} = 8$
- $\text{T.abs} = 15$
- $\text{T.abs} = 16$
- $\text{T.abs} = 20$

- $\text{T1.now()} = [3, 15]$
- $\text{T1.commit} (ts = 15)$
- $\text{T2.now()} = [18, 22]$
- $\text{T2.commit} (ts = 22)$

- $\text{T1.ts} > \text{T2.ts}$

### This Lecture

- How write transactions are done
  - 2PL + 2PC (sometimes 2PL for short)
  - How they are timestamped

- How read-only transactions are done
  - How read timestamps are chosen
  - How reads are executed

### Read-Write Transactions (2PL)

- Three phases
  - Execute → Prepare → Commit
  - 2PC: atomicity
**Read-Write Transactions (2PL)**

**Execute:**
- Does reads: grab read locks and return the most recent data, e.g., R(A=a)
- Client computes and buffers writes locally, e.g., A = a+1, B = a+1, C = a+1

**Prepare:**
- Choose a coordinator, e.g., A, others are participants
- Send buffered writes and the identity of the coordinator; grab write locks
- Each participant prepares T by logging a prepare record via Paxos with its replicas. Coord skips prepare (Paxos Logging)
- Participants send OK to coord if lock grabbed and after Paxos logging is done

**Commit:**
- After hearing from all participants, coord commits T if all OK; o/w, abort T
- Coord logs commit/abort record via Paxos, applies writes if commit, release locks
- Coord sends commit/abort messages to participants
- Participants log commit/abort via Paxos, apply writes if commit, release locks
- Coord sends result to client either after its "log commit" or after ack

---

**Timestamping Read-Write Transactions**

**Commit:**
- Coordinator: choose a timestamp, e.g., ts_A, larger than
  - Any writes it has applied
  - Any timestamps proposed by the participants, e.g., ts_B and ts_C
  - Its current TT.now().latest
- Coord commit-waits: TT.after(ts_A) = true. Commit-wait overlaps w Paxos logging
  - ts_A is T's commit timestamp
Read-Only Transactions (shards part)

Client chooses a read timestamp \( ts = \text{TS.now().latest} \).

- If no prepared write, return the preceding write, e.g., on A.
- If write prepared with \( ts' > ts \), no need to wait, proceed with read, e.g., on B.
- If write prepared with \( ts' < ts \), wait until write commits, e.g., on C.

\[ \text{Txn } T' = \text{R}(A=\_, B=\_, C=\_\_\_) \]

- Client specifies a read timestamp \( ts = \text{TS.now().latest} \).
- If no prepared write, return the preceding write, e.g., on A.
- If write prepared with \( ts' > ts \), no need to wait, proceed with read, e.g., on B.
- If write prepared with \( ts' < ts \), wait until write commits, e.g., on C.

Serializable Snapshot Reads

- Client specifies a read timestamp way in the past
  - E.g., one hour ago
- Read shards at the stale timestamp
- Serializable
  - Old timestamp cannot ensure real-time order
- Better performance
  - No waiting in any cases
  - E.g., non-blocking, not just lock-free
- Can have performance but still strictly serializable?
  - E.g., one-round, non-blocking, and strictly serializable
  - Coming in next lecture!

Takeaway

- Strictly serializable (externally consistent)
  - Make it easy for developers to build apps!
- Reads dominant, make them efficient
  - One-round, lock-free
- TrueTime exposes clock uncertainty
  - Commit wait and at least \( \text{TT.now().latest()} \) for timestamps ensure real-time ordering
- Globally-distributed database
  - 2PL w/ 2PC over Paxos!