Recap: Spanner is Strictly Serializable

- 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 ($\epsilon$) 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 $S_a$

- $T_{abs}$
  - $T1.now() = [3, 15]$
  - $T1.commit (ts = 15)$

- $T_{abs}$
  - $T2.now() = [18, 22]$
  - $T2.commit (ts = 22)$

- $S_a$
  - $T2.ts > T1.ts$

- $S_b$
  - $T2.ts > T1.ts$

Enforcing the Invariant with TT

What if T1.commit delayed, such that T2 happens after T1.now() but before T1.commit? Tricky as T1.commit.ts = T1.now().latest

- Answer: T2 delayed until after T1 commits. Discussed later.

- $T_{abs}$
  - $T2.now() = [18, 22]$
  - $T2.commit (ts = 22)$

- $S_a$
  - $T1.now() = [3, 15]$
  - $T1.commit (ts = 15)$

- $S_b$
  - $T1.now() = [3, 15]$
  - $T1.commit (ts = 15)$

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

---

Transaction: $T = \{R(A=?) \land W(A=\text{a}+1), W(B=\text{a}+1), W(C=\text{a}+1)\}$

**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

- **Participant:** Choose timestamp (e.g., $t_{B}$ and $t_{C}$) larger than any writes it has applied.
- **Coordinator:** Choose a timestamp, e.g., $t_{A}$, larger than:
  - Any writes it has applied
  - Any timestamps proposed by the participants, e.g., $t_{B}$ and $t_{C}$
  - Its current $\text{TT.now}()$.latest

---

Read-Only Transactions (shards part)

- **Txn $T'$** = $R(A=\,?\,)\,B=\,?\,,\,C=\,?\,)\,$
  - **Client** chooses a read timestamp $ts = \text{TT.now}().\text{latest}$
  - 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

---

Read-Only Transactions (Paxos part)

- **A** shard can process a read at $ts$ if $ts \leq ts_{safe}$
- $ts_{safe} = \min(t_{Paxos}, t_{TM}, t_{safe})$:
  - before $ts_{safe}$, all system states (writes) have finalized
  - $t_{Paxos}$ = Timestamp of highest-applied Paxos write
  - $t_{TM}$ = infinity if zero prepared (but not committed) transactions
  - Else, min of all prepare timestamps of any prepared txns
### 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 TT.now.latest() for timestamps ensure real-time ordering
- Globally-distributed database
  - 2PL w/ 2PC over Paxos!