Spanner

November 2025

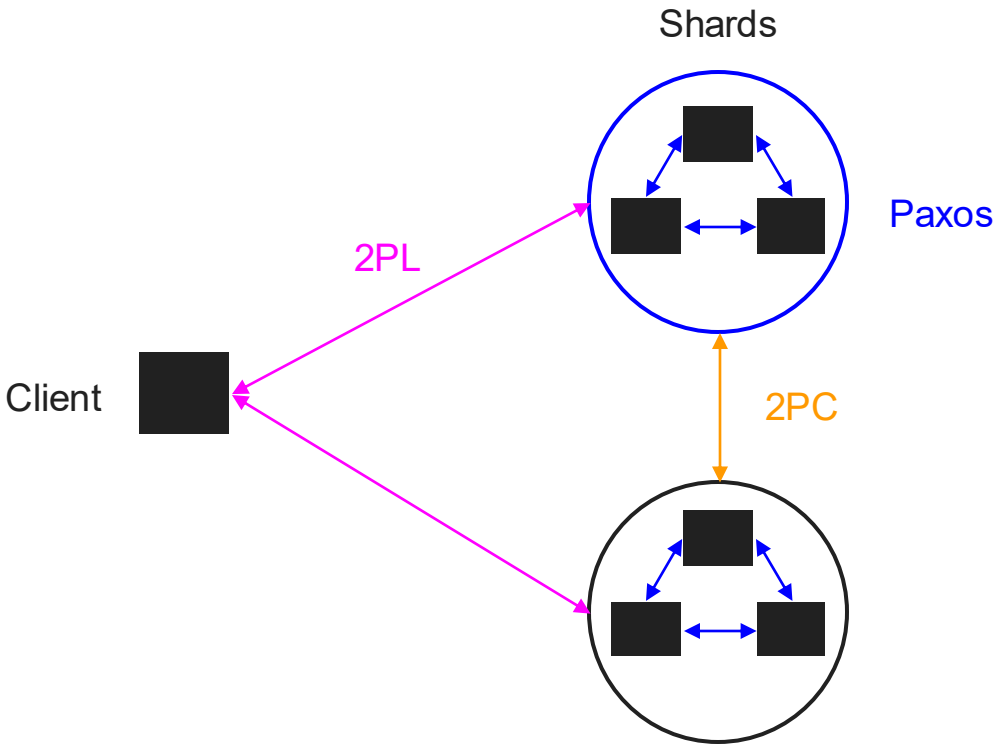
Concurrency Control Recap

- Last precept: 2-phase locking (2PL)
- 2PL:
  - Rule: **Do not acquire a lock once any lock has been released**
  - Growing Phase: acquire shared (read) locks and exclusive (write) locks
  - Shrinking Phase: release locks

How can we achieve strict serializability and scalability?

- **Shard the keyspace**: servers maintain a subset of the keyspace
- Use 2PL to handle concurrent transactions
- Use 2-phase commit (2PC) to achieve atomic commit of transactions
- How does 2PC handle server failures?
  - It doesn't!
- Replicate each shard using Paxos!

Toy example:



One of the shards would be transaction coordinator for 2PC

Putting it together in a real system: Spanner

- Observation: reads are much more frequent than writes
  - Facebook's TAO sees 500 reads per 1 write.
  - Google Ads (F1) on Spanner from 1 DC saw 51.5B reads in a 24 hour period
  - Many reads are across shards
- Takeaway: **Make read-only transactions very efficient**
- Two goals:
  - Lock-free read-only transactions
  - Non-blocking, but stale (not strictly serializable), read-only transactions

Spanner

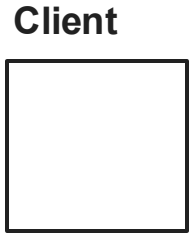
- Main idea: use **real-time** for ordering transactions by finding a maximum clock skew
- TrueTime
  - TrueTime.now()
    - Returns a range [a,b] where a is the earliest possible time, and b is latest
  - TrueTime.after(t)
    - True if the current time is definitely after t
  - TrueTime.before(t)
    - True if the current time is definitely before t

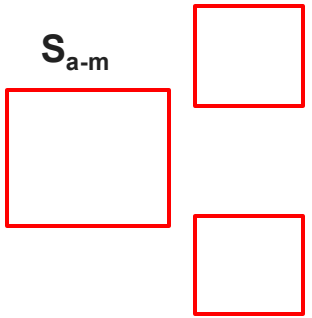
General transactions

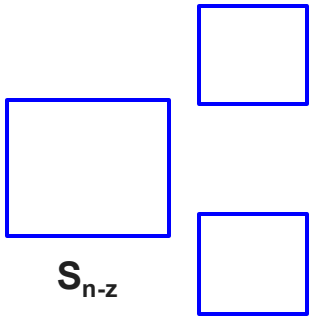
- ***General transactions*** are transactions that can contain reads and writes together, or just read or write
- Similar to 2PL+2PC+Paxos scheme above, but use ***TrueTime*** to determine commit timestamps for transactions
- Each server maintains t_{safe} where all transactions with commit timestamp $s_i < t_{safe}$ are committed and can be read.
    - t_{safe} means: "I guarantee that no future transactions will commit at timestamps ≤ t_{safe}."

Example

txn 1:
x = r(a)
y = r(z)
x = x + y
w(z = x)

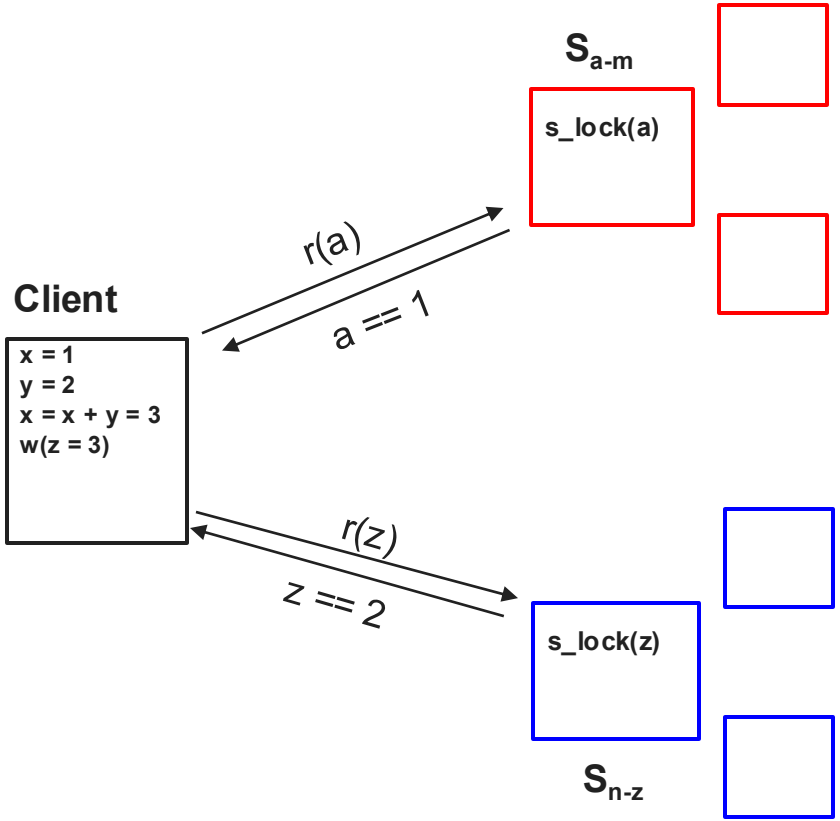






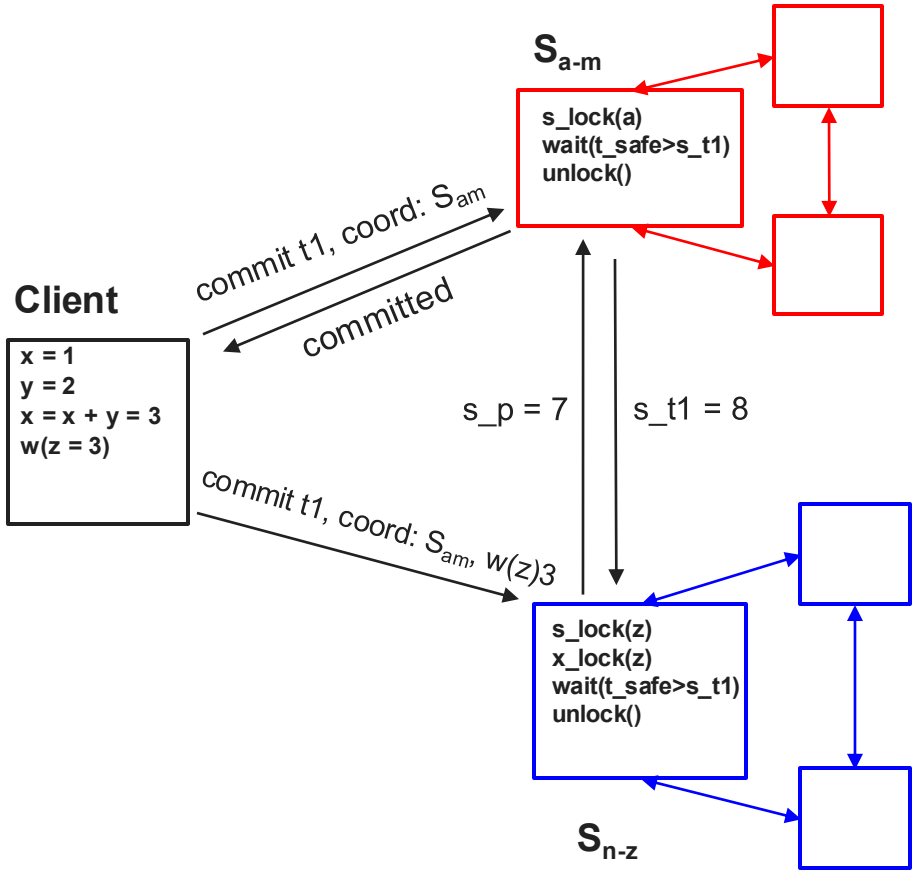
Example

txn 1:
x = r(a)
y = r(z)
x = x + y
w(z = x)



Example

txn 1:
  x = r(a)
  y = r(z)
  x = x + y
  w(z = x)



Now let’s walk through the transaction process step by step

General transactions (steps)

General transactions are driven by the client:

1. Client issues reads to the leader of each shard group
2. Leader acquires read locks and returns the most recent value to the client
3. Client locally performs the writes
4. Client chooses a coordinator from the shard leaders
5. Client initiates the commit protocol by sending a commit message to each leader with the buffered writes and the coordinator ID
6. Leaders execute the commit protocol
7. Client waits for the commit message from the coordinator

General transaction (commit protocol)

1. All shard leaders acquire write locks
2. Non-coordinators
    a. Choose a prepare timestamp > all previous local timestamps
    b. Log the prepare record via Paxos
    c. Notify the coordinator of the prepare timestamp
3. Coordinator
    a. Waits for all prepare timestamps
    b. Chooses a commit timestamp that is
        i. >= prepare timestamps of all other non-coordinators
        ii. > any write timestamps it has applied
        iii. > its current TT.now().latest
    c. Logs commit record via Paxos
    d. Wait until TrueTime.after(commit timestamp)
    e. Sends commit timestamp to replicas, non-coordinators, and the client
4. All apply the transaction at commit timestamp and release the locks

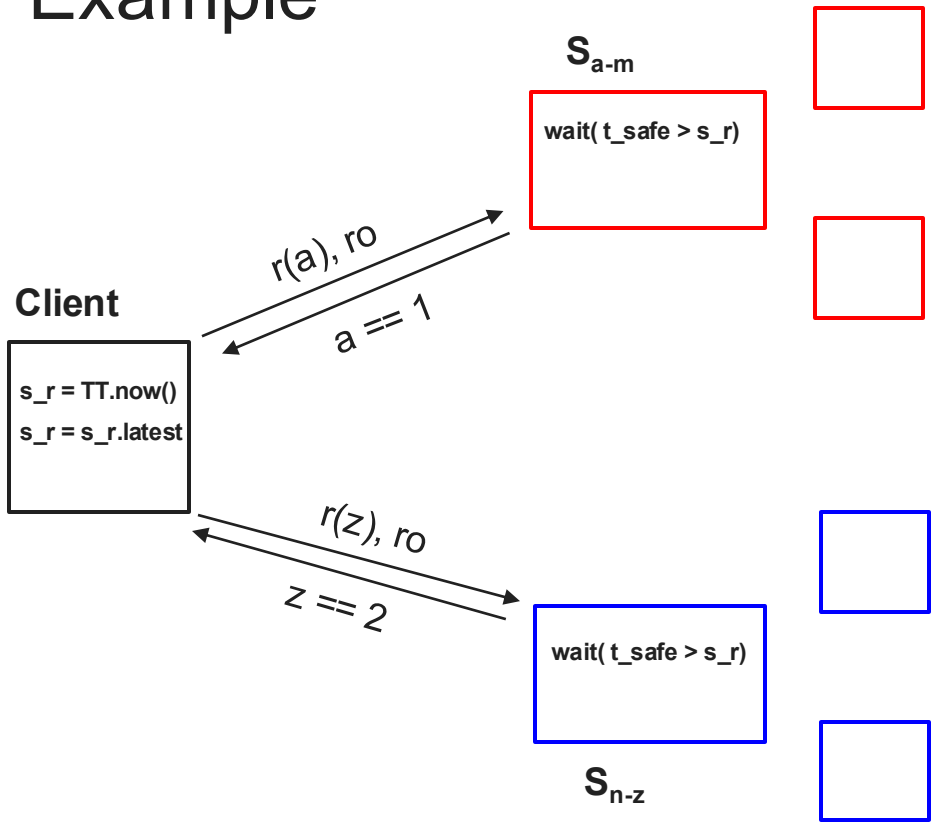
Lock-free read-only transactions

1. Client chooses the commit timestamp (s_{read}) to be TrueTime.now.latest()
2. Shard leaders wait until $s_{read} < t_{safe}$

   The shard *waits* until it's sure that all transactions that could affect s_{read} have already committed.
3. Read data as of the time s_{read}
4. Return data.

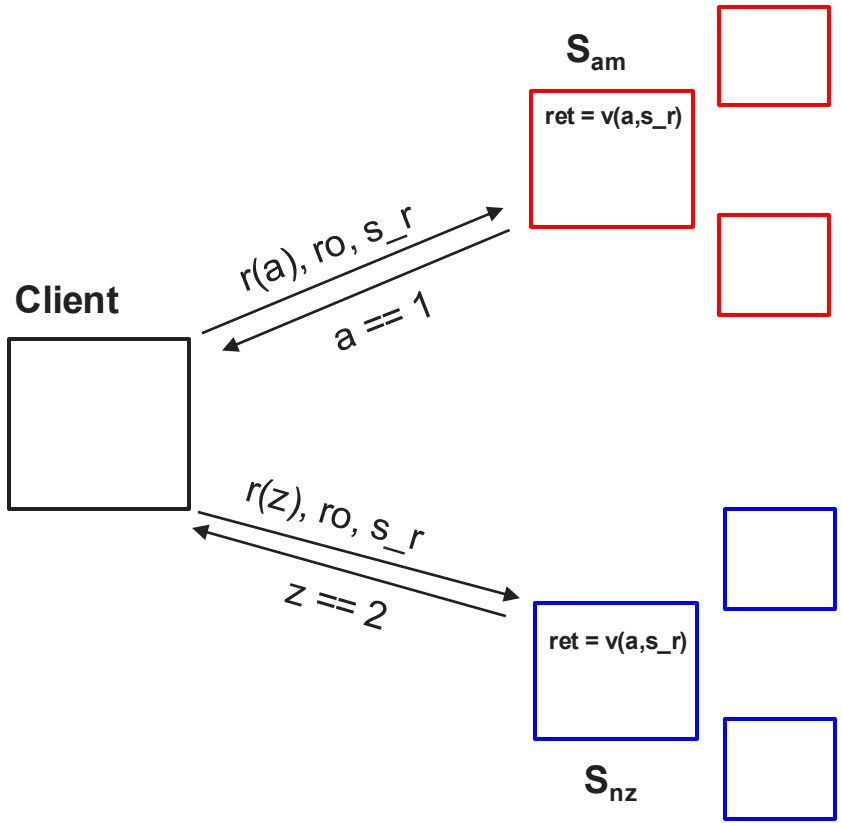
Read-Only Example

txn 1:
x = r(a)
y = r(z)



Non-blocking (BUT STALE) - Read-Only Transactions

txn 1:
x = r(a)
y = r(z)



Summary of Spanner

- Sharded datastore where shards are Paxos groups
- Transactions use Client-driven 2PL
- Commit Wait: 2PC with waiting for the commit time to have passed and be safe to read