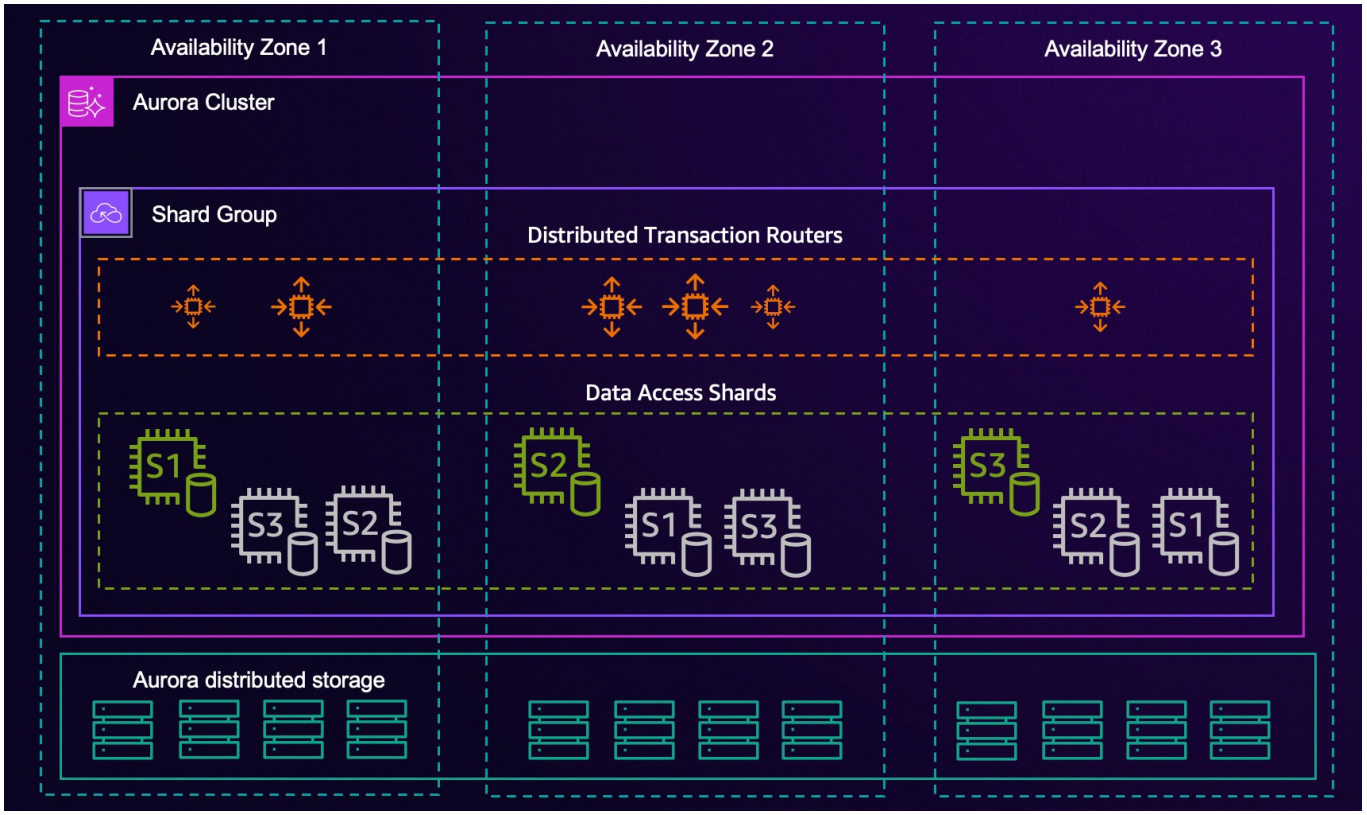
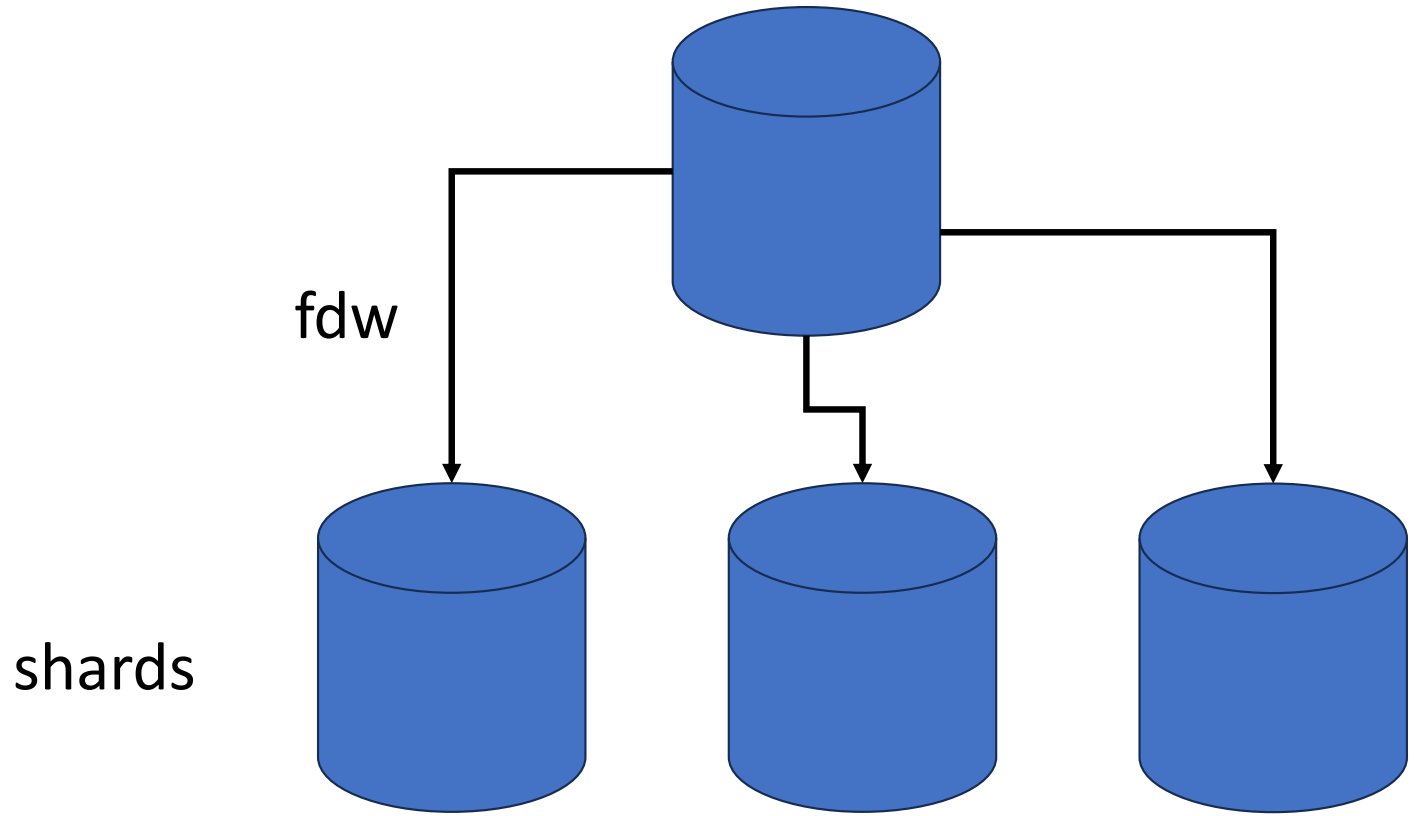
Teaching Elephants To Tell Time

Adventures in Distributing Snapshot Isolation

A complicated view…

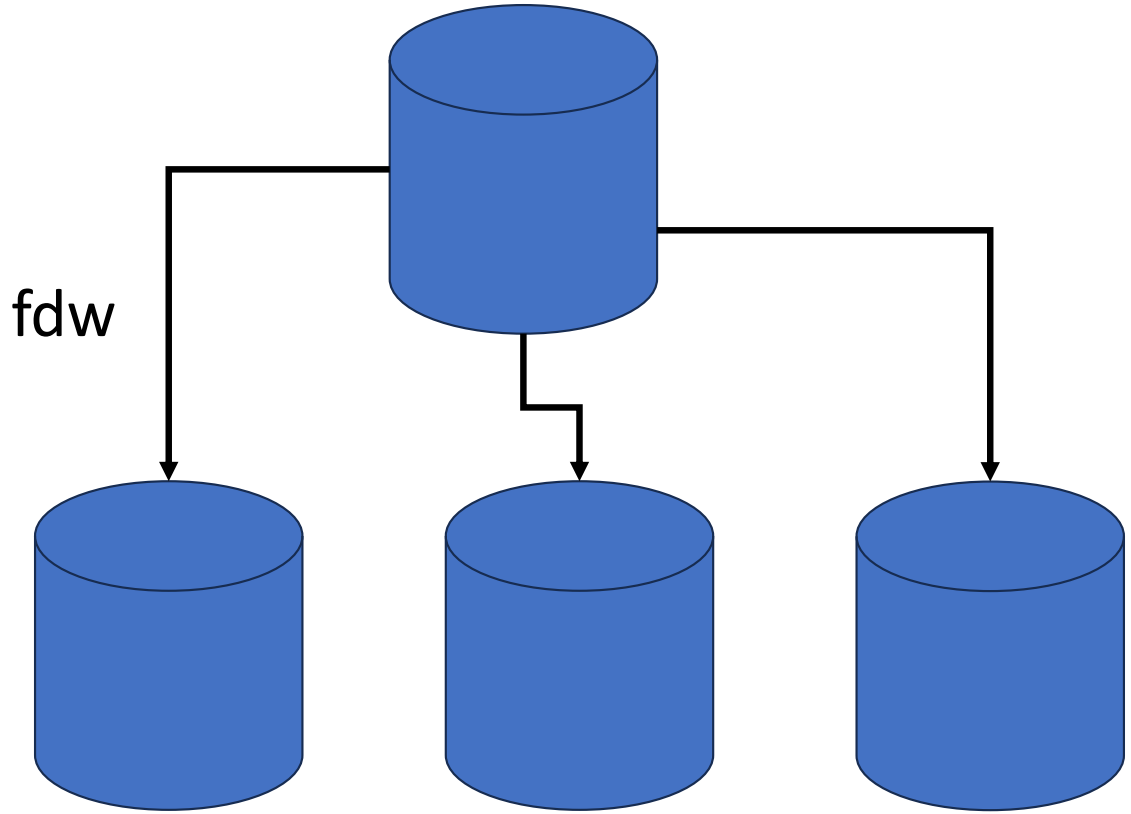


…of a familiar pattern



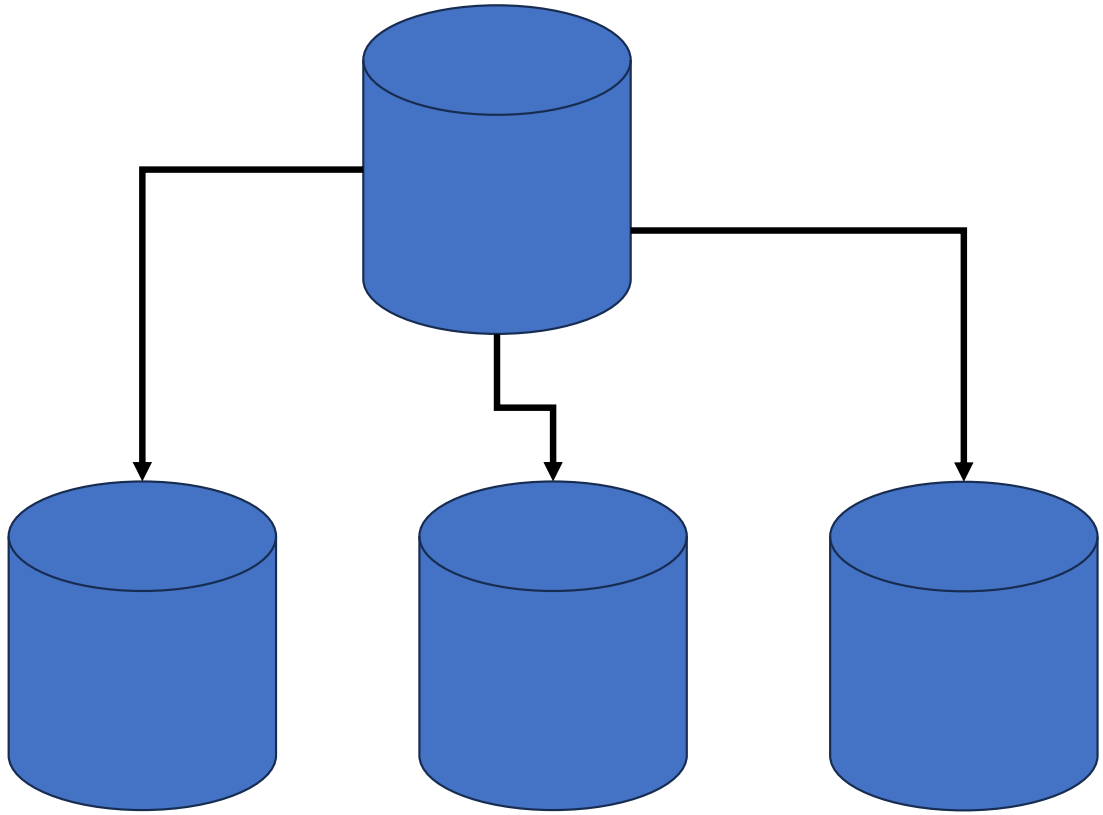
…of a familiar pattern

PostgreSQL and FDW can’t provide snapshot isolation



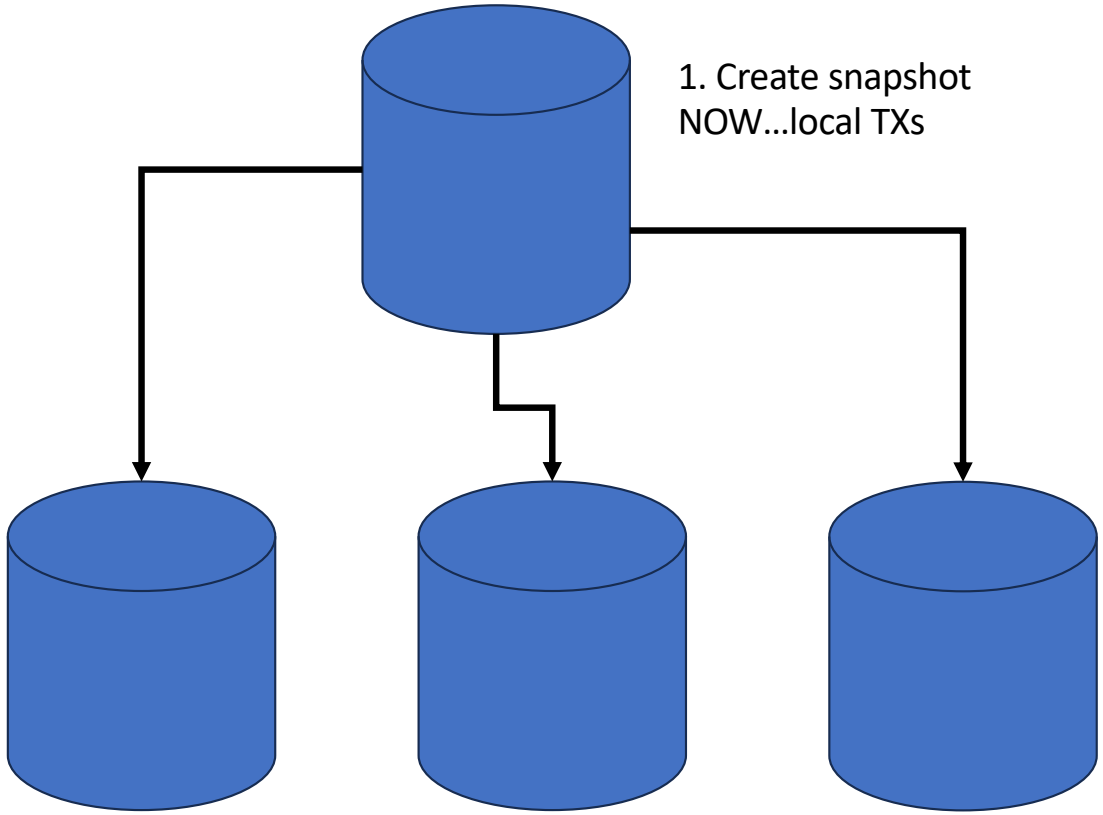
PostgreSQL creates snapshots as of NOW
with serialized local state

Now doesn’t work in distributed systems



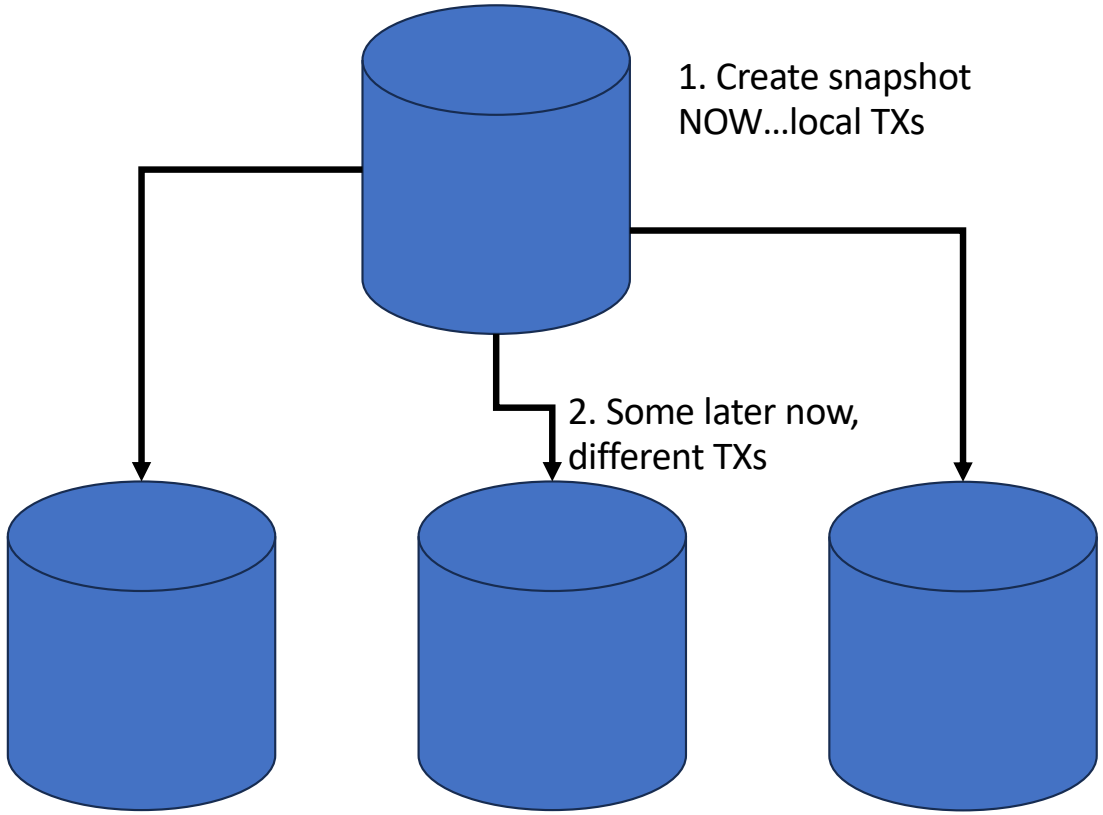
Now doesn’t work in distributed systems

```
select
sum(abalance)
from
pgbench_accounts;
```



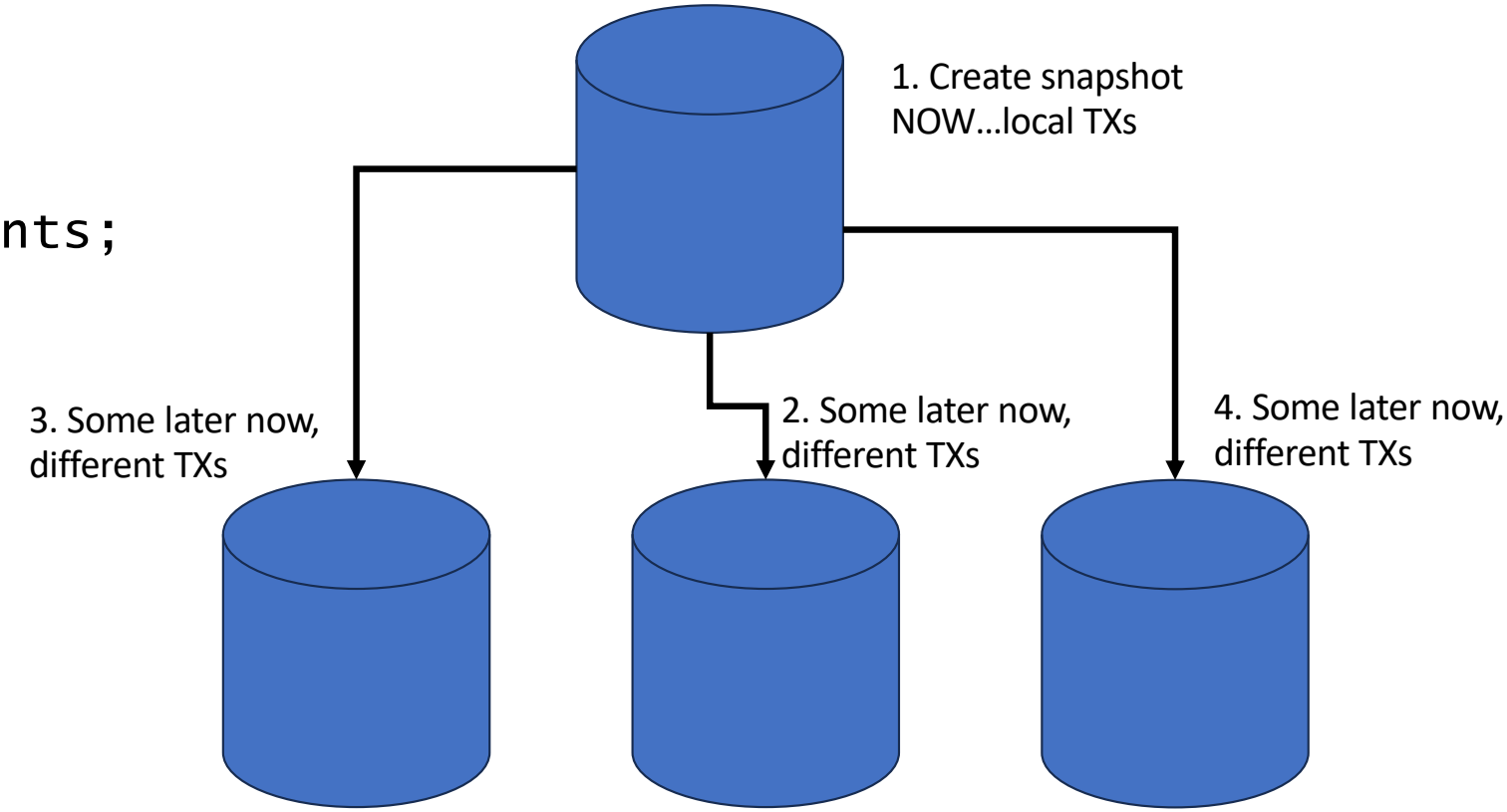
Now doesn’t work in distributed systems

```
select
sum(abalance)
from
pgbench_accounts;
```



NOW doesn't work in distributed systems

```
select
sum(abalance)
from
pgbench_accounts;
```



No consistent snapshot

😕

PostgreSQL creates snapshots as of ~~NOW~~ THEN

with s~~erialized~~ NO local state

PostgreSQL creates snapshots as of ~~NOW~~ THEN
with s~~erialized~~ NO local state

Time-based MVCC

Simple building blocks

1. Establish snapshot on current time…`GetSnapshotData()` irrelevant

Simple building blocks

1. Establish snapshot on current time…`GetSnapshotData()` irrelevant

2. Pass snapshot time along with query fragment to shards.

Simple building blocks

1. Establish snapshot on current time…`GetSnapshotData()` irrelevant

2. Pass snapshot time along with query fragment to shards.

3. Record commit time in clog

Simple building blocks

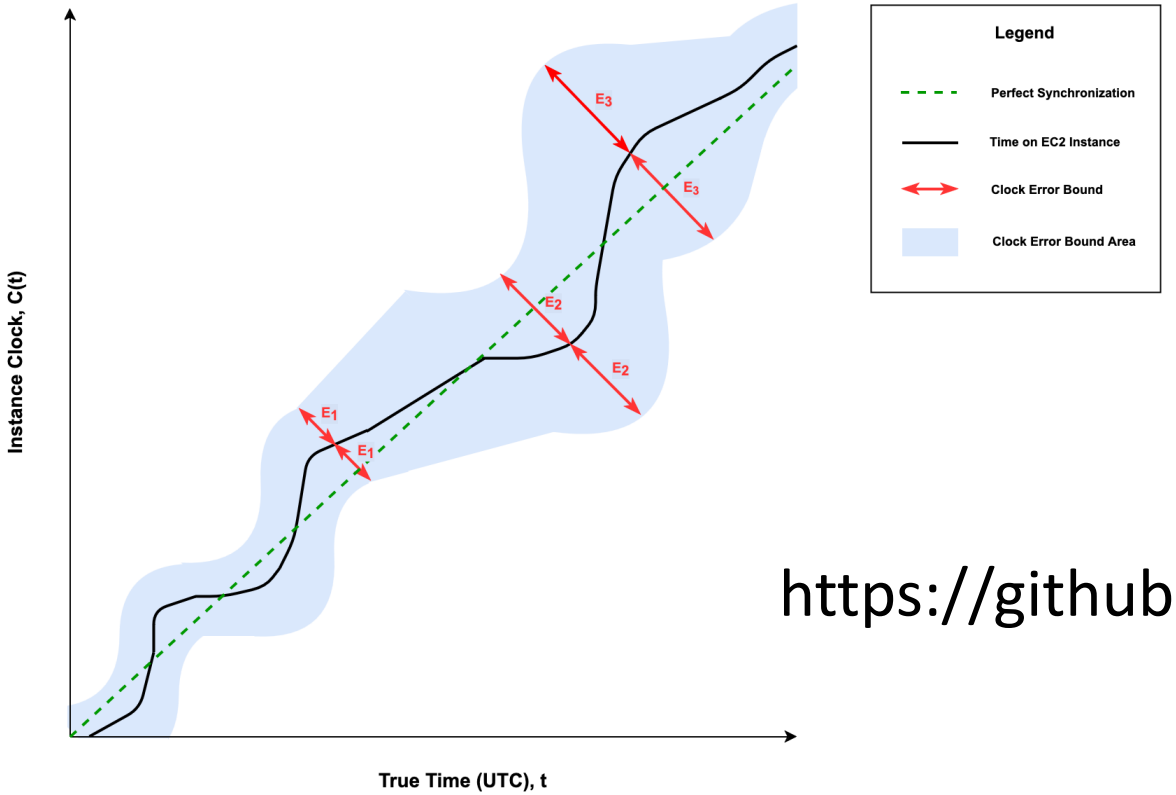
1. Establish snapshot on current time…`GetSnapshotData()` irrelevant

2. Pass snapshot time along with query fragment to shards.

3. Record commit time in clog

4. Visibility is commit time vs snapshot time

Distributed clocks????

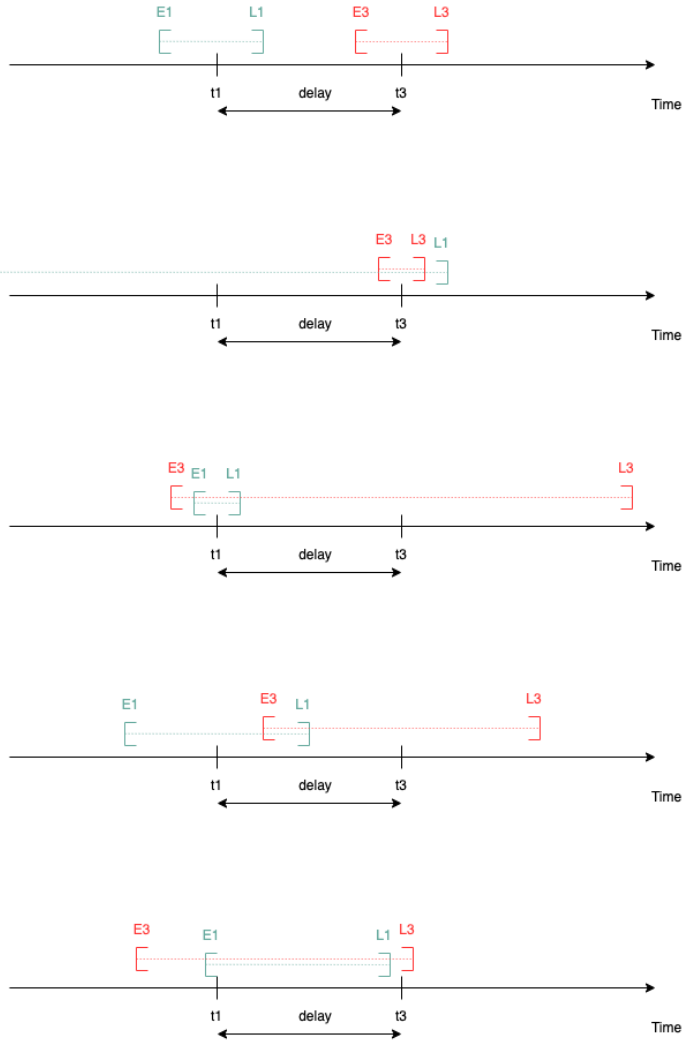


https://github.com/aws/clock-bound

Some cool clock invariants

Events: $\{E_1 \ldots T_1 \ldots L_1\}$ and $\{E_3 \ldots T_3 \ldots L_3\}$

T_1 occurs before T_3 → $E_1 < L_3$



Pseudo commit algorithm

```
doCommit()
{
       // returns immediately with lsn and HLC time of commit log record
      pair{lsn,TimeOfCommit} = flushLogToStorageAsync();

      // returns when the 3AZ durability point >= the commit record
      // Typically returns 1.5ms after commit time @ p50
      waitForLsnDurable(lsn);

      // returns when ClockBound.earliest > TimeOfCommit
      // Typically returns immediately as (commit time – earliest) < 1ms @ p90
      waitForEarliest(TimeOfCommit);
}
```

Pseudo commit algorithm

```
doCommit()
{
        // returns immediately with lsn and HLC time of commit log record
       pair{lsn,TimeOfCommit} = flushLogToStorageAsync();

       // returns when the 3AZ durability point >= the commit record
       // Typically returns 1.5ms after commit time @ p50
       waitForLsnDurable(lsn);

       // returns when ClockBound.earliest > TimeOfCommit
       // Typically returns immediately as (commit time – earliest) < 1ms @ p90
       waitForEarliest(TimeOfCommit);
}
```

No blocking in waitForEarliest observed in cluster sustaining 5MM NOPM (HammerDb)

No blocking in 60 shard cluster performing 2MM commits / sec

That’s just the intro

Clog compaction

Snapshot horizon gossip for vacuum control

Distributed commit coordinator (2PC)

Hooks that make this an extension

Non-blocking 2PC algorithm

Committing status for long-fork anomaly prevention