Google Spanner - A Globally Distributed, Synchronously-Replicated Database System

James C. Corbett, et. al.

Feb 14, 2013. Presented By Alexander Chow For CS 742

Motivation

- “Eventually-consistent” sometimes isn’t good enough.
  - General Purpose Transactions (ACID)
- Application desires complex, evolving schemas
  - Schematized Tables
- SQL-like query language

The Problem

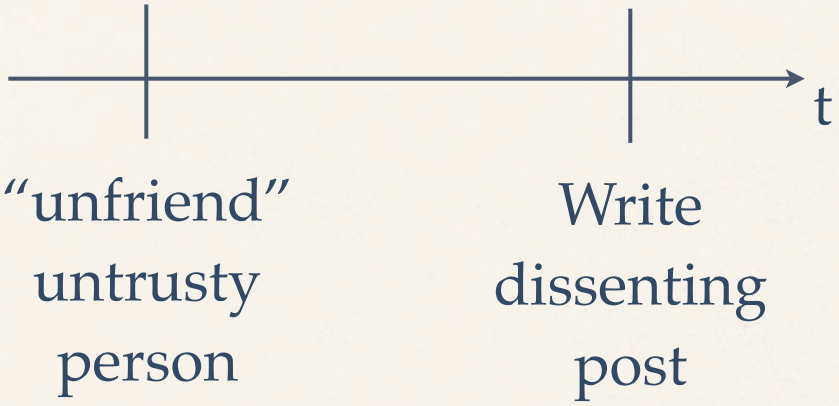
- Store data across thousands of machines, hundreds of data centres
- Replication across data centres, even continents

Spanner Features

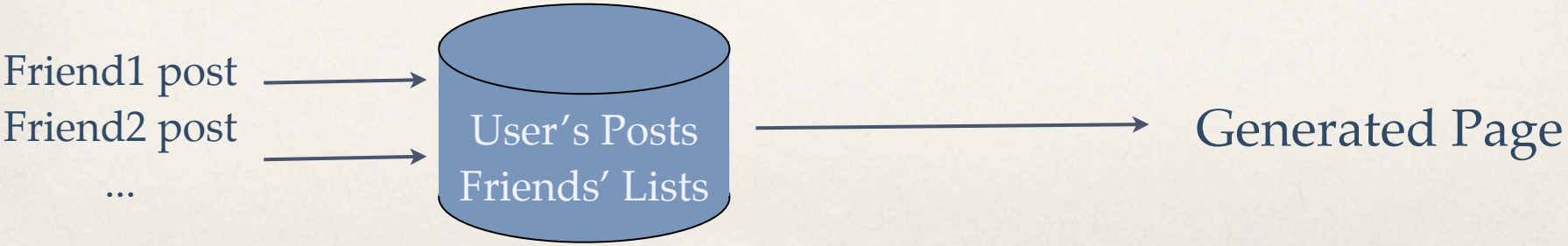
- Lock-free distributed read transactions from any sufficiently-up-to-date replica
- External consistency
  - Commit order == Timestamp Order == Global Wall Clock Time
    - The "TrueTime" API

Lock-free Reads

- Example

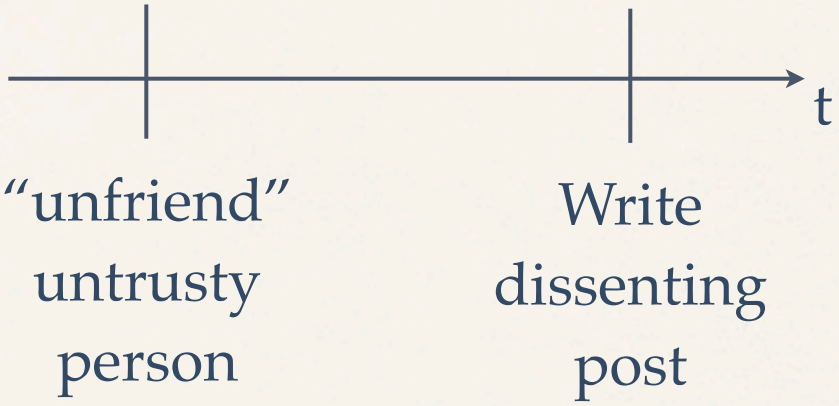


- Single Machine Read

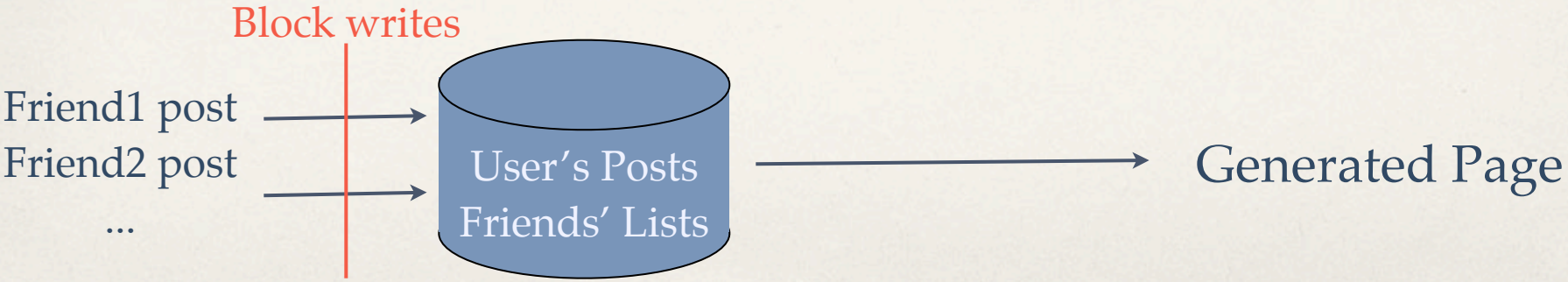


Lock-free Reads

- Example

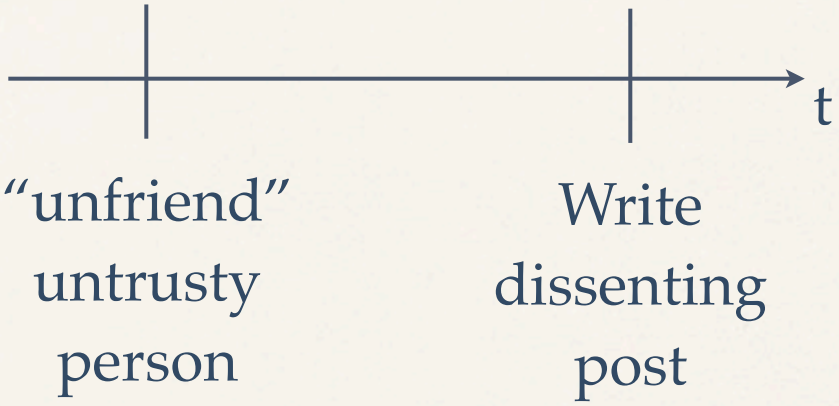


- Single Machine Read

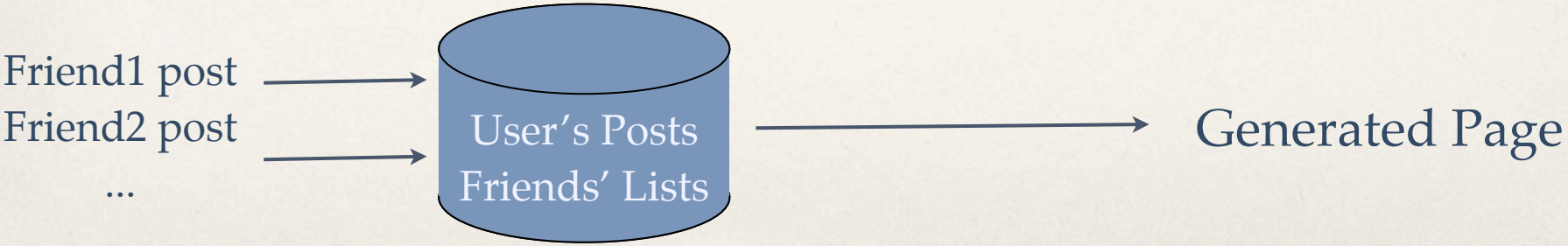


Lock-free Reads

- Example

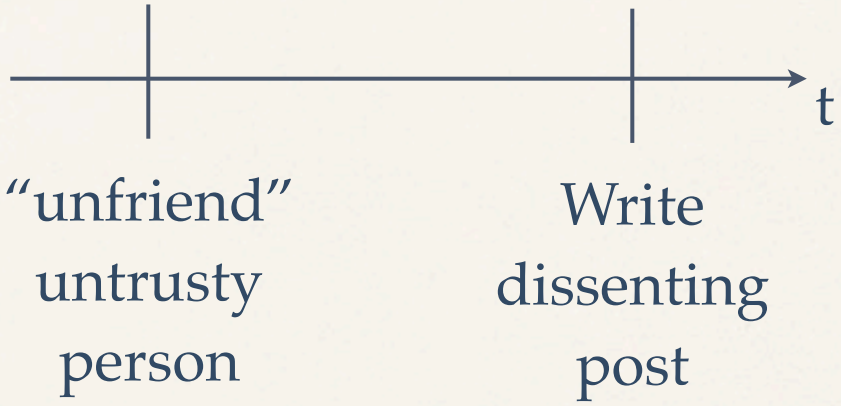


- Single Machine Read

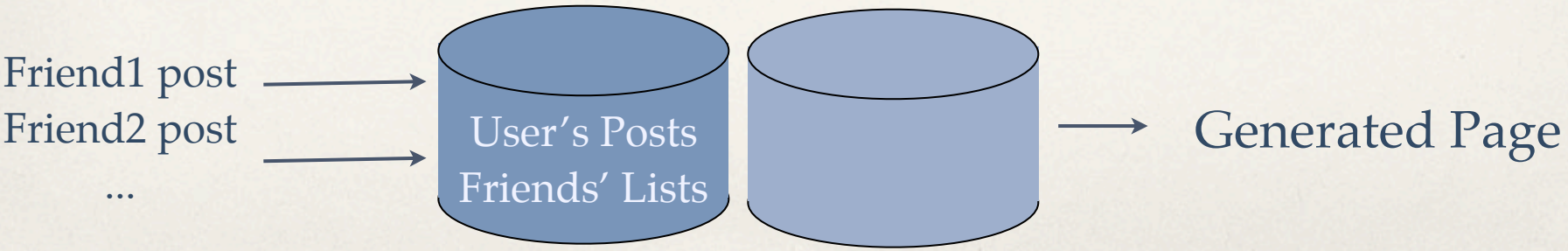


Lock-free Reads

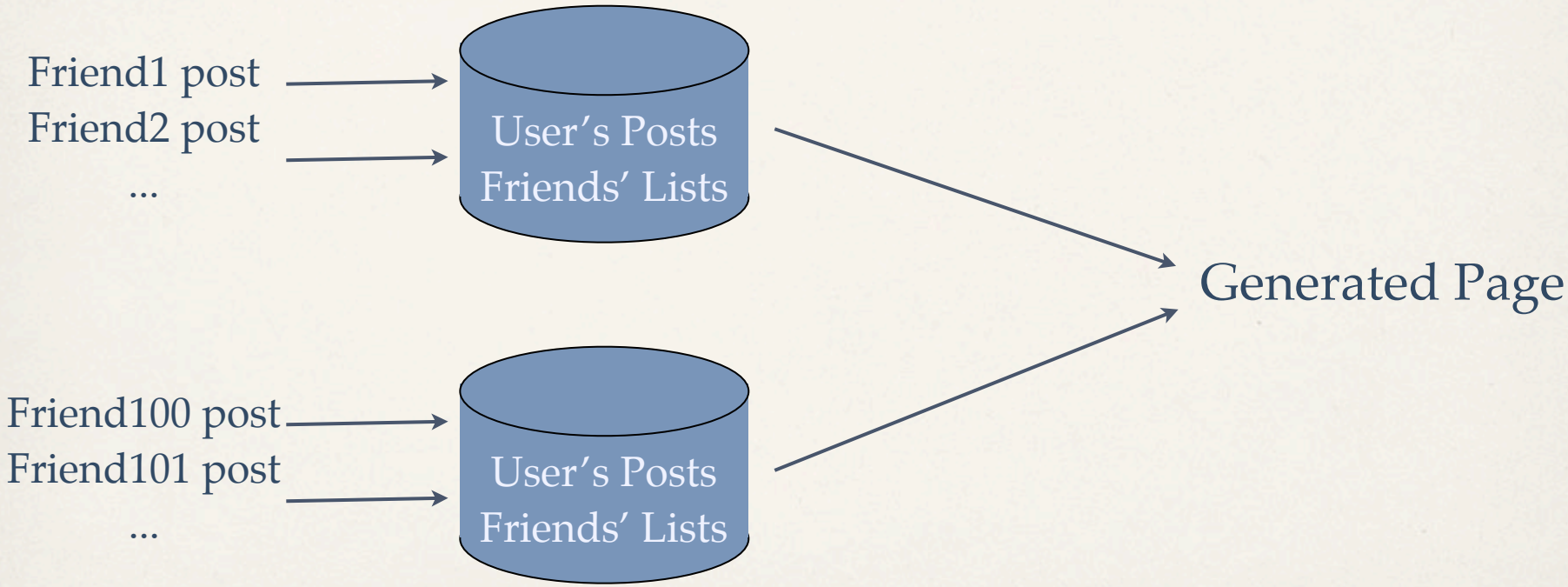
- Example



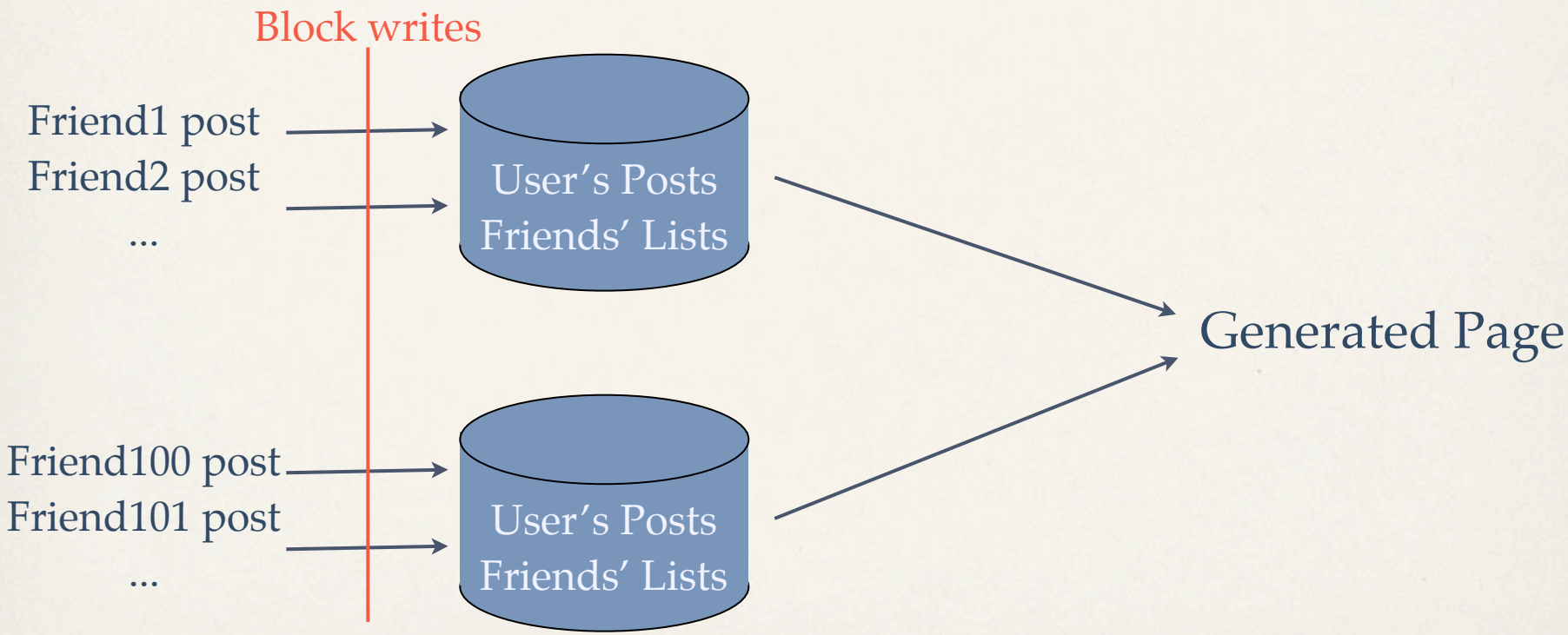
- Single Machine Read



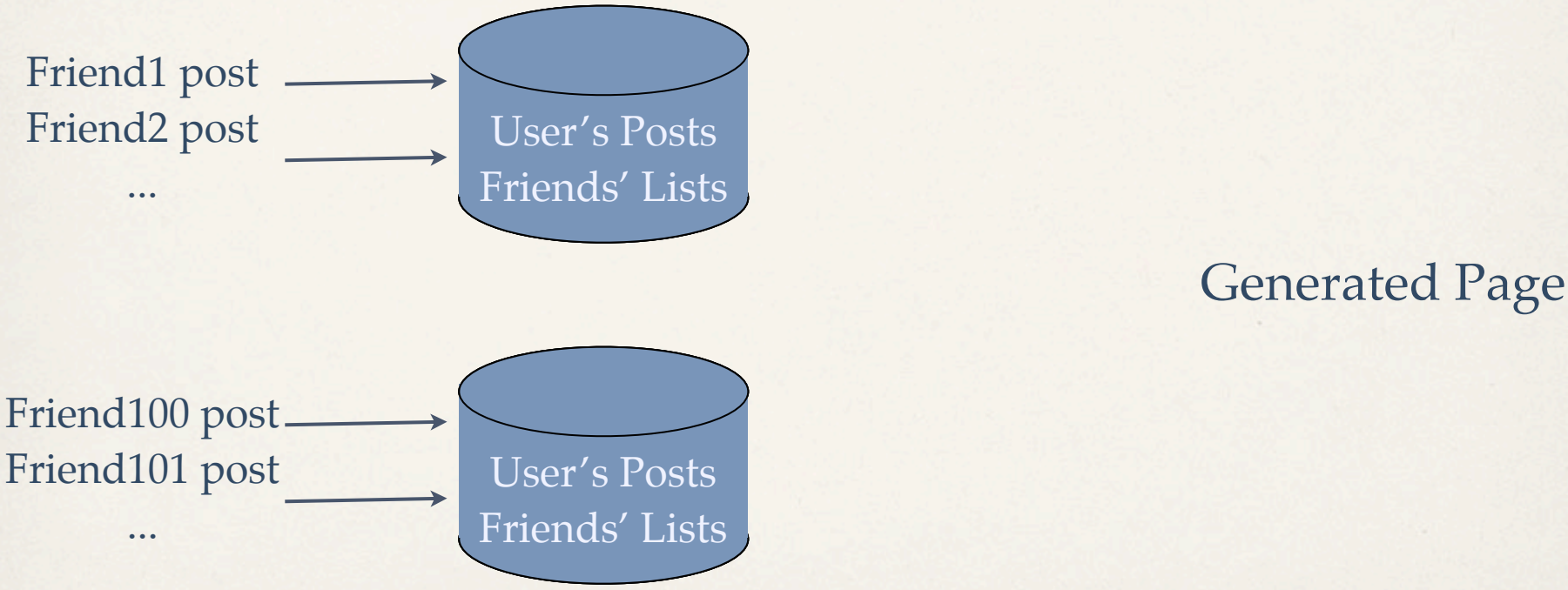
Lock-free Reads



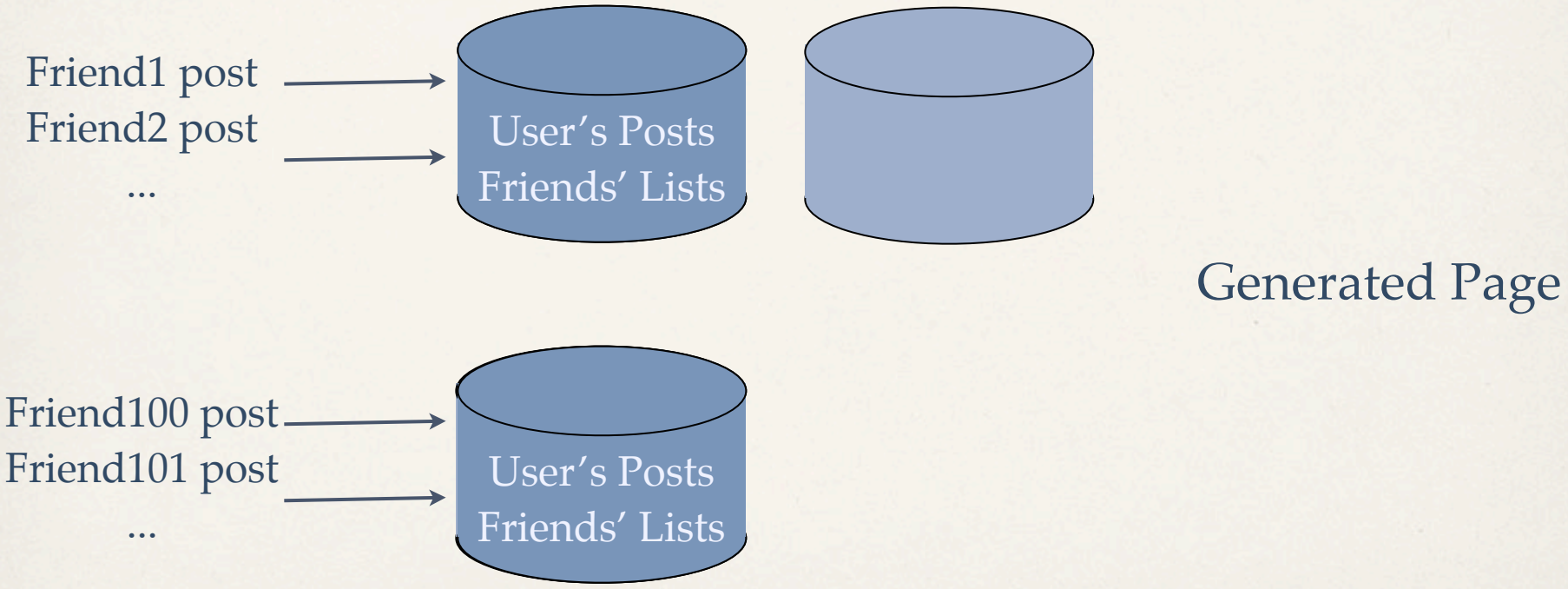
Lock-free Reads



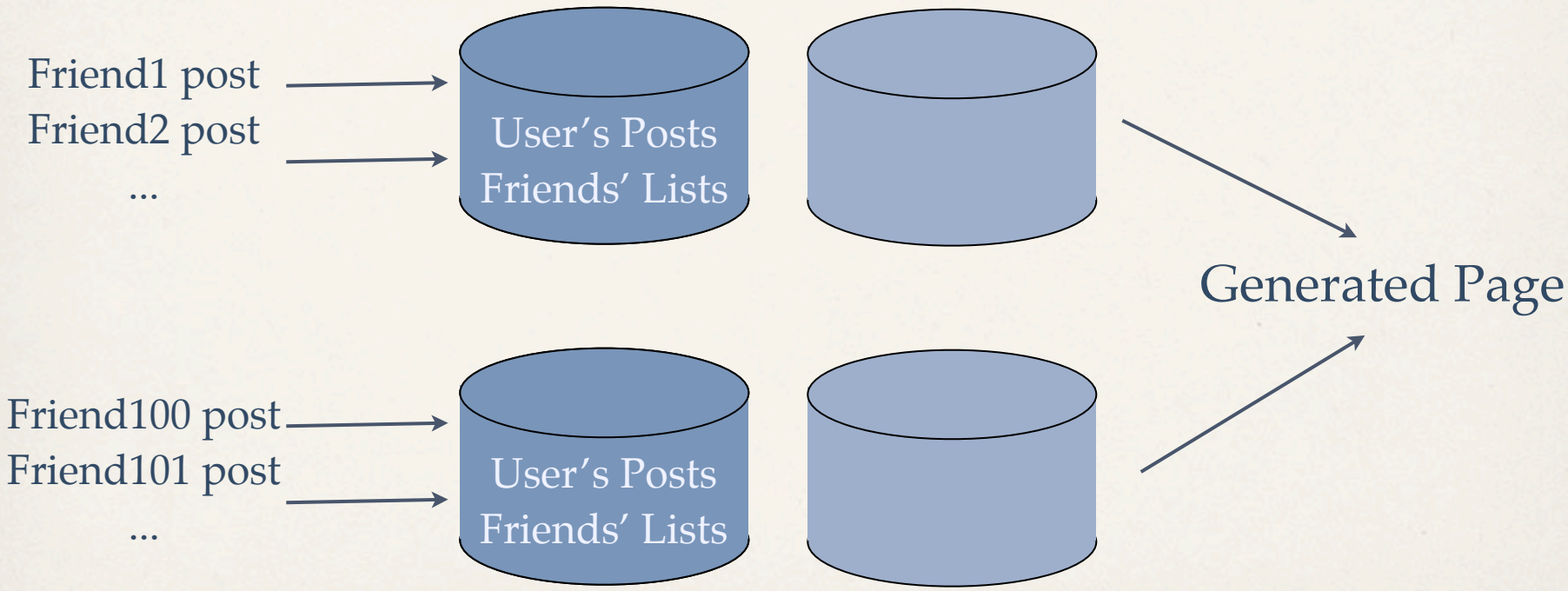
Lock-free Reads



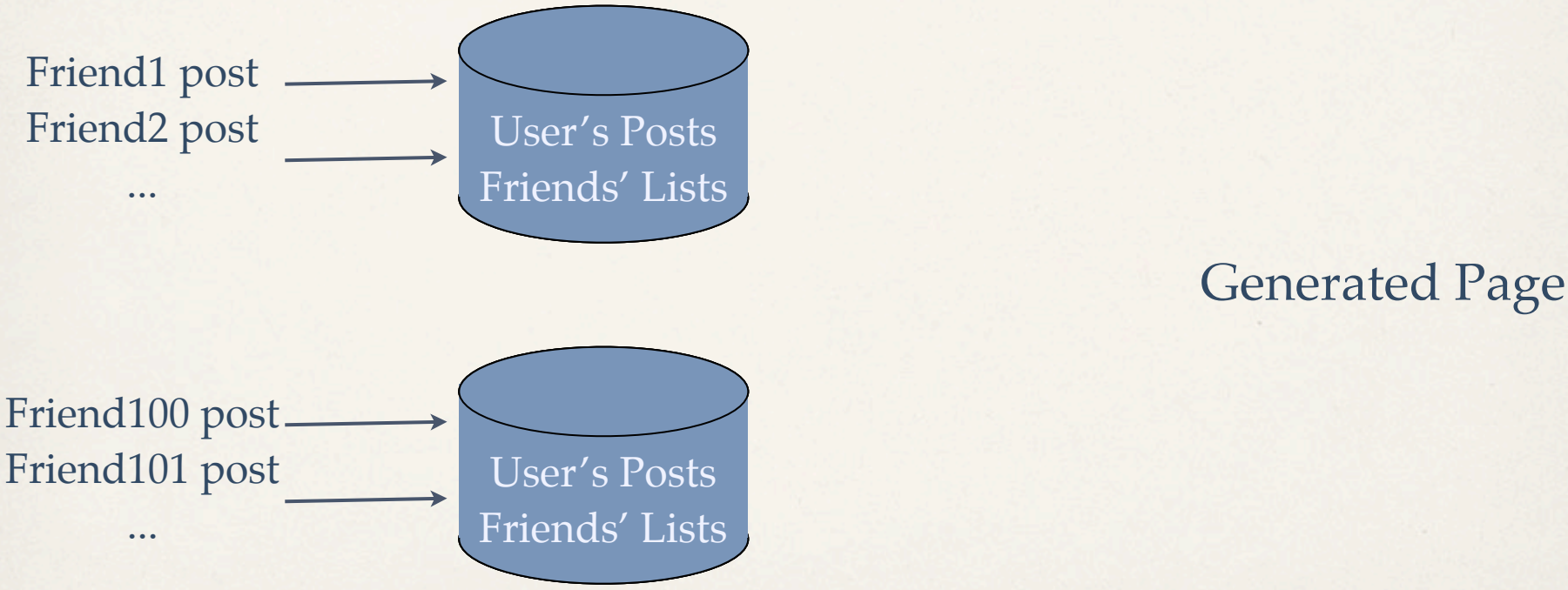
Lock-free Reads



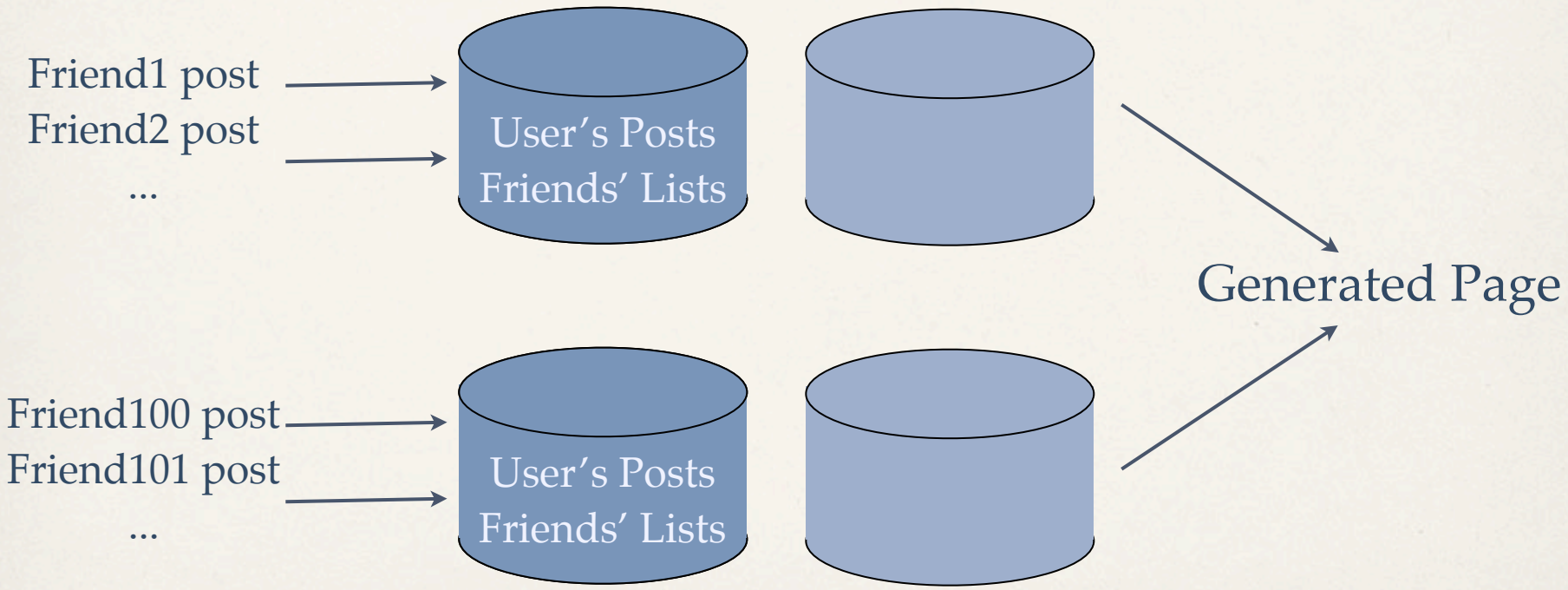
Lock-free Reads



Lock-free Reads

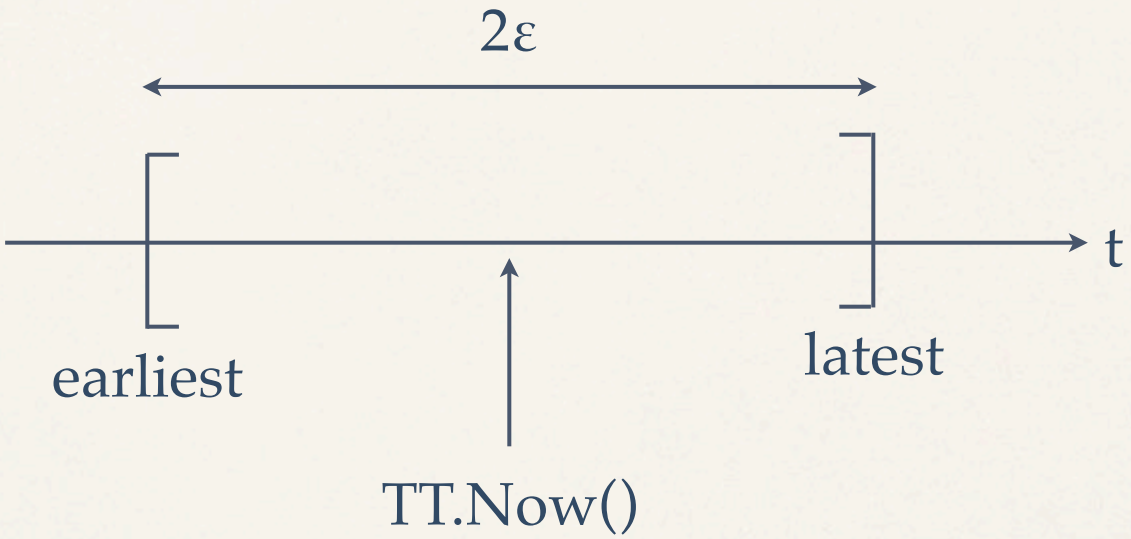


Lock-free Reads



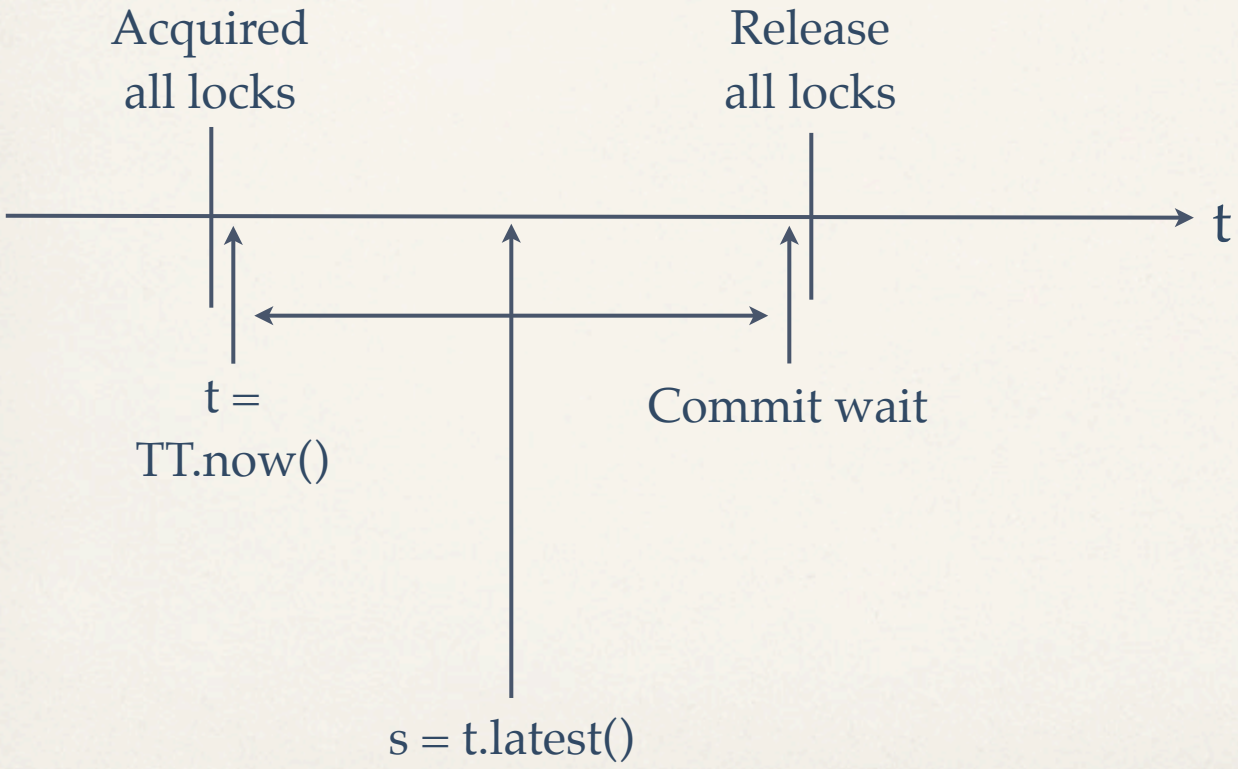
TrueTime API

- TT.Now()



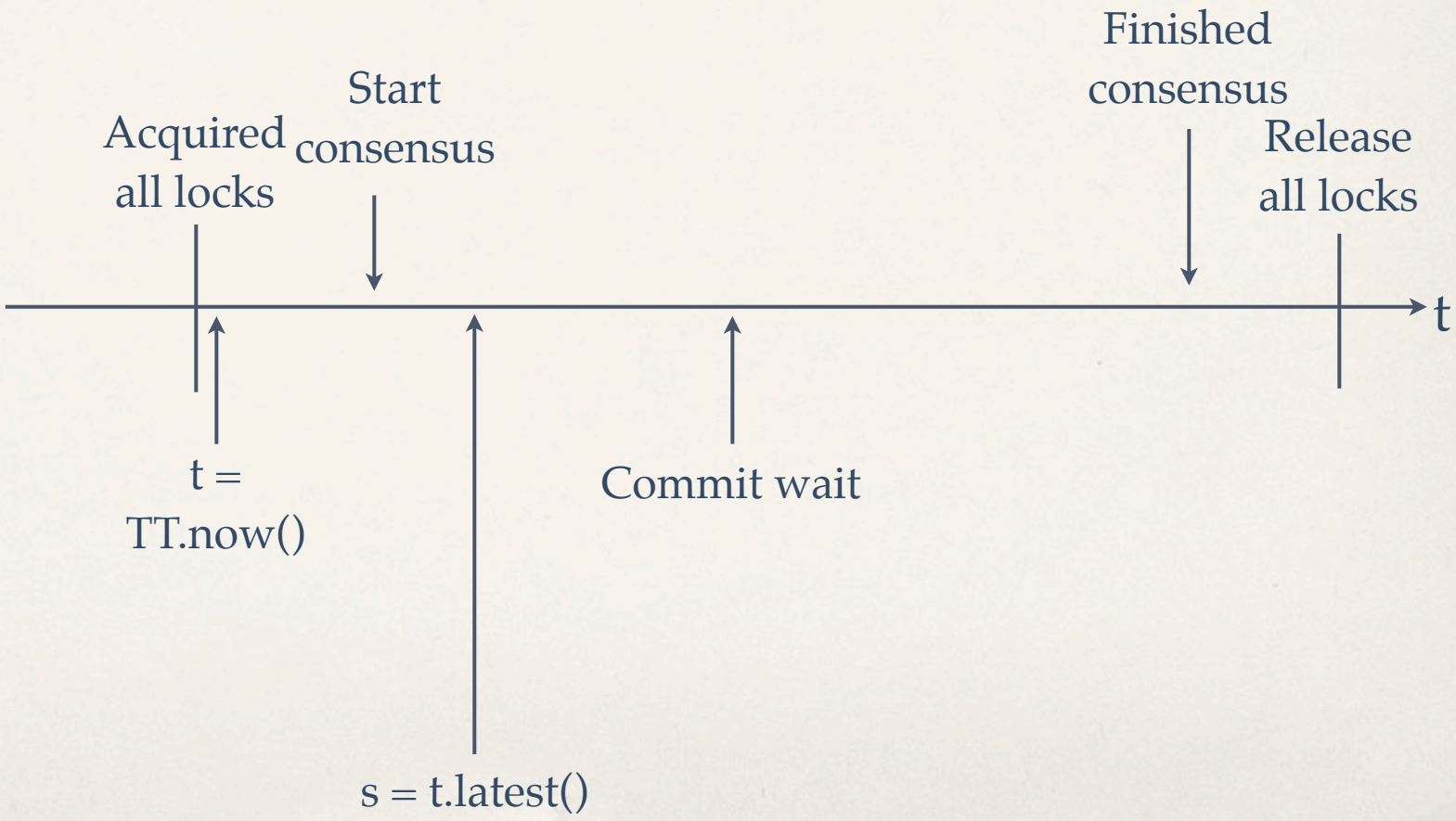
Read-Write Transaction

- 2 Phase Locking

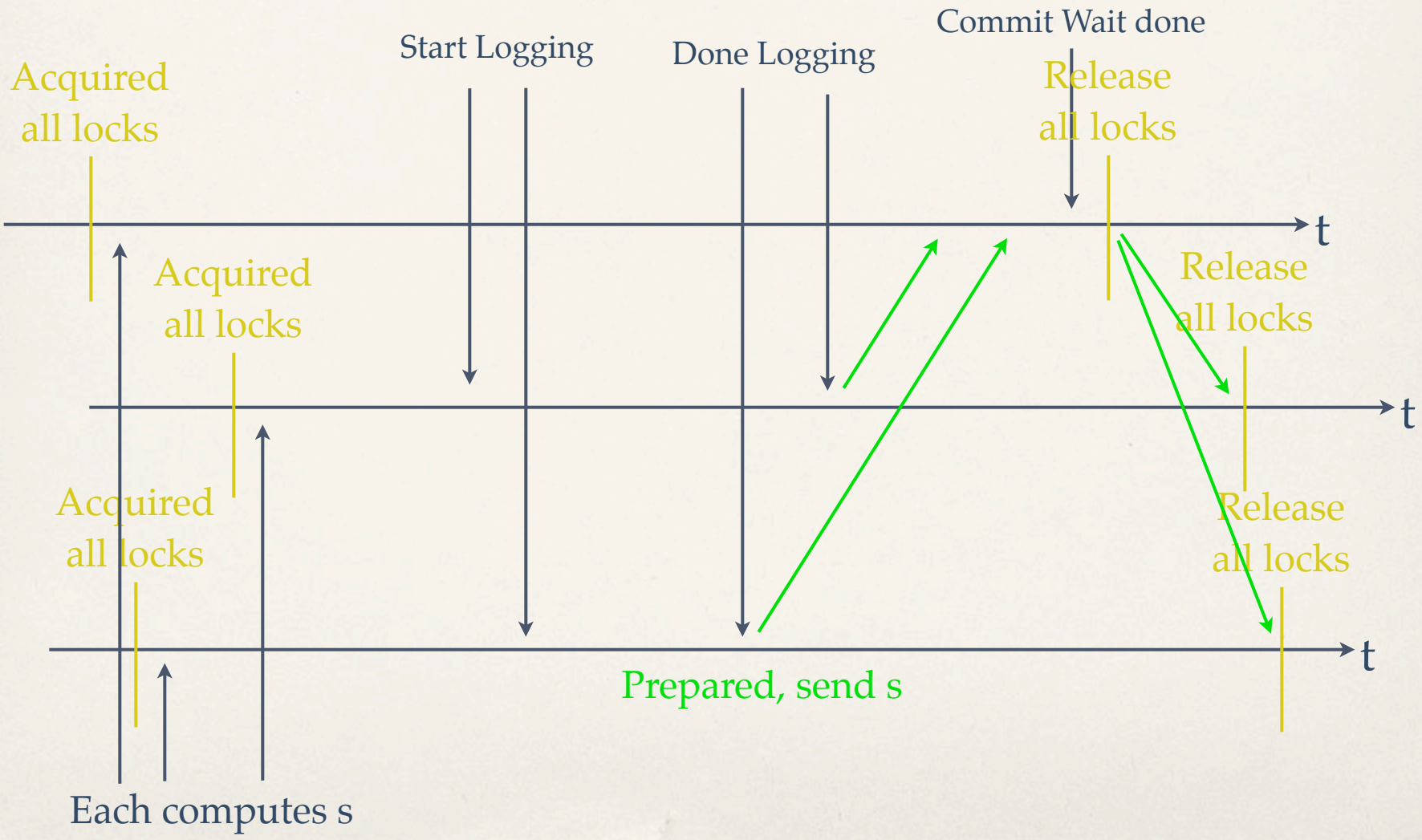


Overlapping with Commit Wait

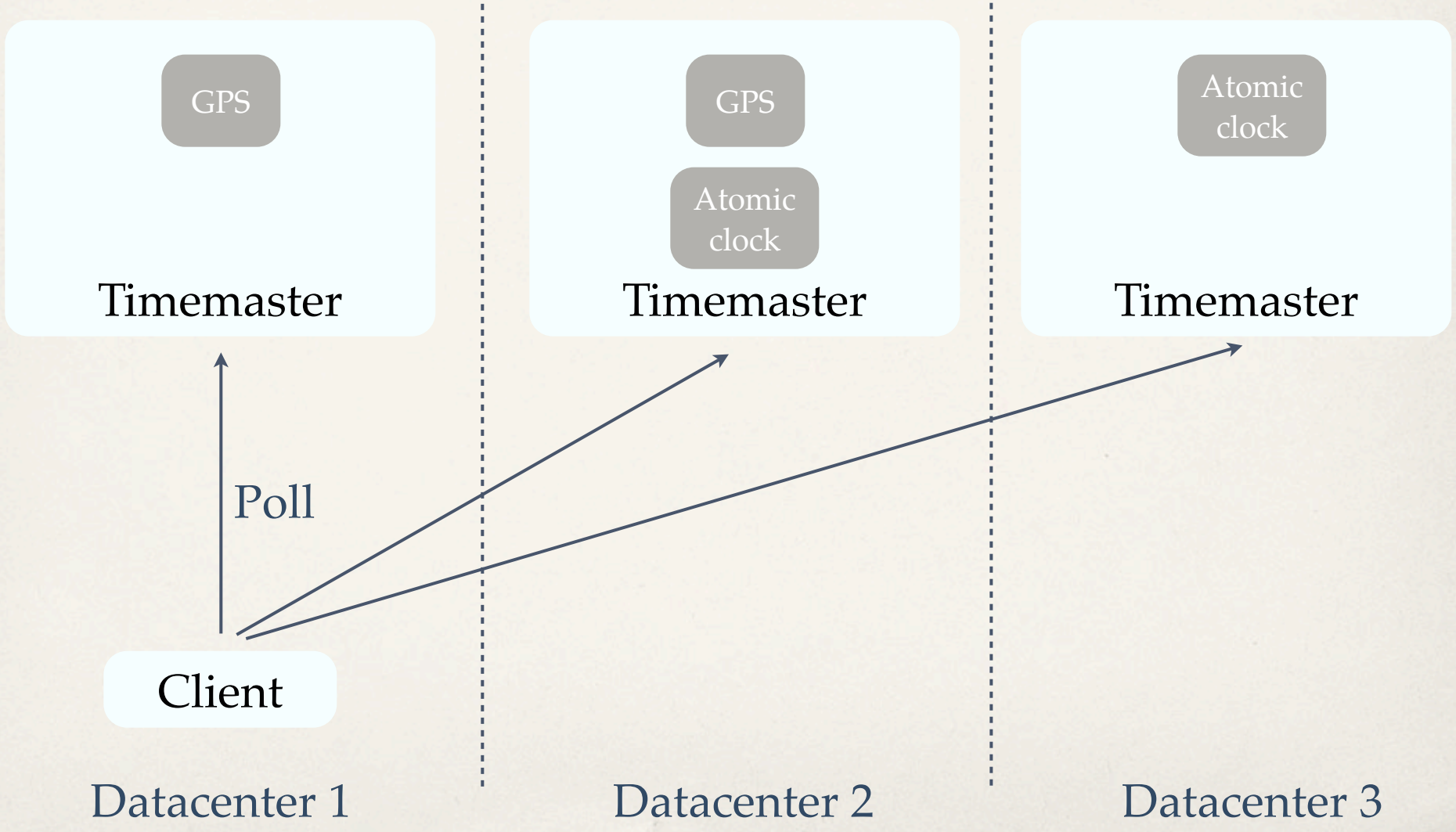
- Network cost to achieve consensus far dominates time for commit wait; no need to wait



Integrating 2PC and TrueTime



Implementing TrueTime



Implementing TrueTime

- Time at synchronization (polling of timemasters, every 30 seconds)
  - Time is from nearest available timemaster
  - Poll nearby datacenter's timemasters for redundancy, detect rogue timemasters. Use variation on Marzullo's Algorithm to detect liars, compute time of non-liars.
  - ε resets to ε broadcast by Timemaster plus communication time (1ms) plus
- Between synchronizations:
  - Increase ε by local drift (200us/s)

Time availability by design

- Commit time uses variable ε
- If local timemaster not available, can use remote timemaster from other data center (100+ ms delay)
- Spanner slows down automatically

Easy Schema Change

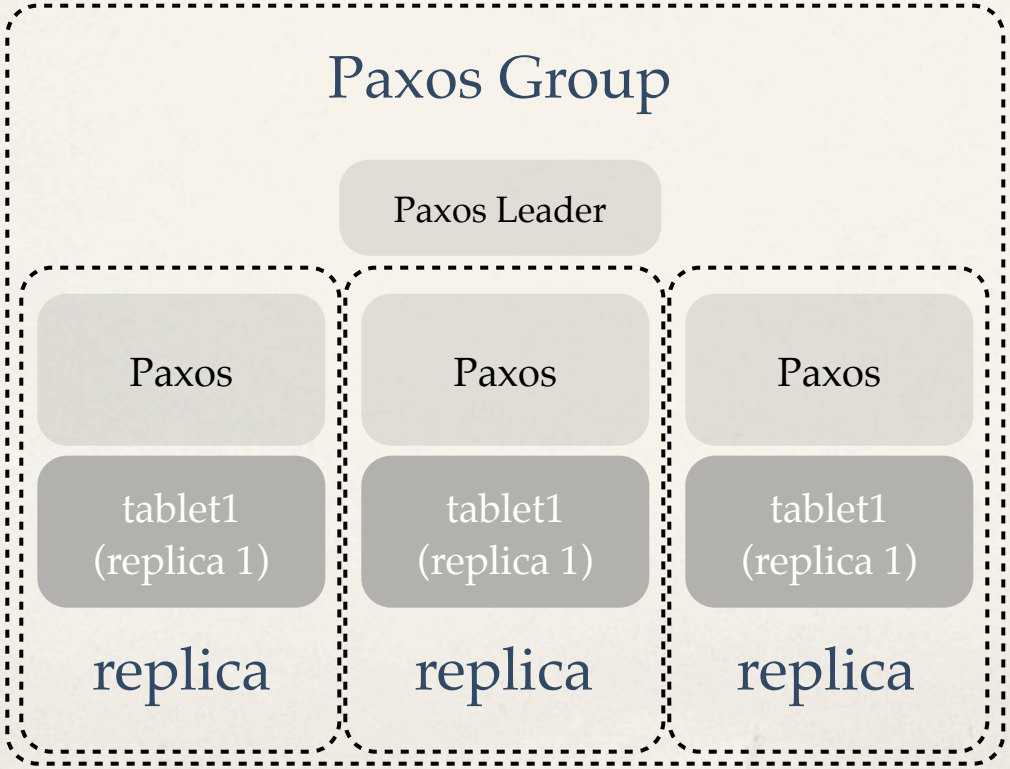
- Non-blocking variant of regular transaction
- At prepare stage, choose a timestamp t in the future
- Reads and writes which implicitly depend on schema:
  - If their time is before t, proceed
  - If their time is after t, block
- Without TrueTime, defining a schema change to happen at "time t" would be meaningless.

Spanner Implementation Details

- Tablet: Similar to Bigtable's *tablet*. A bag of mappings of:
  - (key:string, timestamp:int64) -> string
  - More like multi-version database
  - Stored on Colossus (distributed file system)

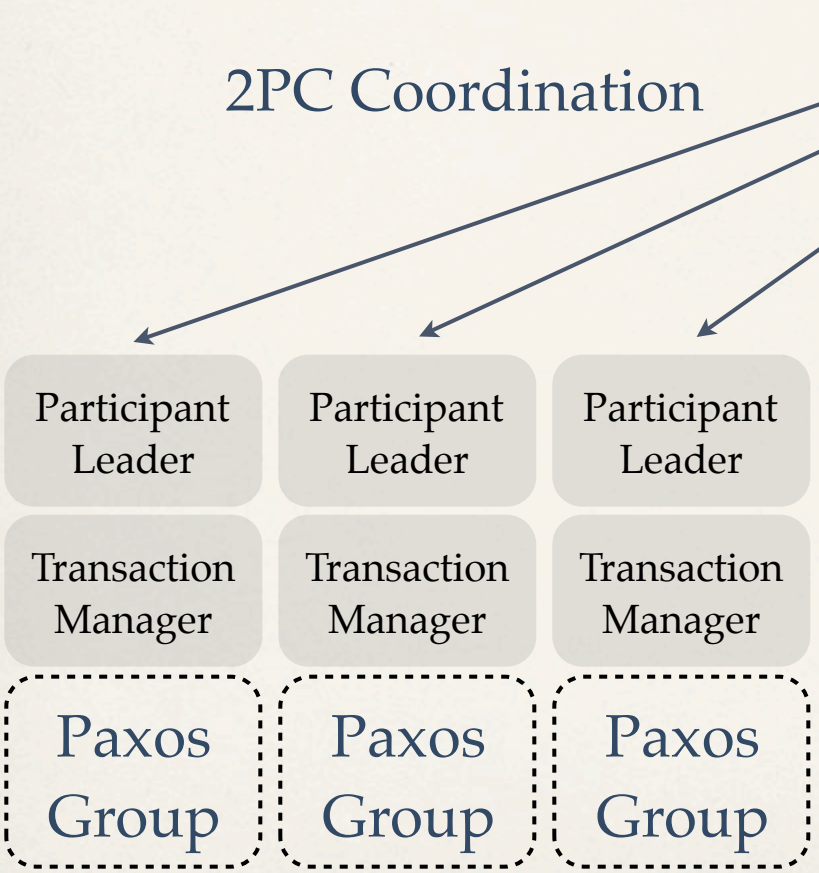
Spanner Implementation Details

- Tablets are replicated (between datacenters, possibly inter-continental), concurrency coordination by Paxos
- A transaction needs consistency across its replicas; coordinated by Paxos



Paxos Group: A tablet and its replicas as well as the concurrency machinery across the replicas

Spanner Implementation Details



If transaction involves multiple Paxos Groups, use transaction management machinery atop of Paxos groups to coordinate 2PC

- If a transaction involves a single Paxos Group, can bypass Transaction Manager and Participant Leader machinery.
- Thus, system involves 2 stages of concurrency control, 2PC and Paxos, where one stage can be skipped.

Lock-free Reads at a Timestamp

- Each replica maintains t_{safe}
- $t_{safe} = \min(t^{paxos}_{safe}, t^{TM}_{safe})$
  - t^{paxos}_{safe} is timestamp of highest-applied Paxos write
  - t^{TM}_{safe} is much harder:
    - $= \infty$ if no pending 2PC transaction
    - $= \min_i (s^{prepare}_{i,g})$ over i prepared transactions in group g.
- Thus, t_{safe} is maximum timestamp at which reads are safe

Data Locality

- Application-level controllable data locality
- Prefix of key used to define the *bucket*
  - *Key: 0PZX2N47HL5N4MAE3Q...*
  - *Key: 0PZX2N47HL5N7U9OY2...*
  - *Key: 0PZX2N47HL5NQBDP73...*
- Entries in the same bucket are always in the same Paxos group.
- Can balance load between Paxos groups by moving buckets.

Benchmarks

- 50 Paxos groups, 2500 buckets, 4KB reads or writes, datacenters 1ms apart
- Latency remains mostly constant as number of replicas increases because Paxos executes in parallel at a group's replicas
- Less sensitivity to a slow replica as number of replicas increases (easy to achieve quorum).

participants	latency (ms)	
	mean	99th percentile
1	17.0 ±1.4	75.0 ±34.9
2	24.5 ±2.5	87.6 ±35.9
5	31.5 ±6.2	104.5 ±52.2
10	30.0 ±3.7	95.6 ±25.4
25	35.5 ±5.6	100.4 ±42.7
50	42.7 ±4.1	93.7 ±22.9
100	71.4 ±7.6	131.2 ±17.6
200	150.5 ±11.0	320.3 ±35.1

Benchmarks

- All leaders explicitly placed in zone Z1.
- Killing all servers in a zone at 5 seconds. For Z1 test, completion rate drops to almost 0.
- Recovers quickly after reelection of new leader

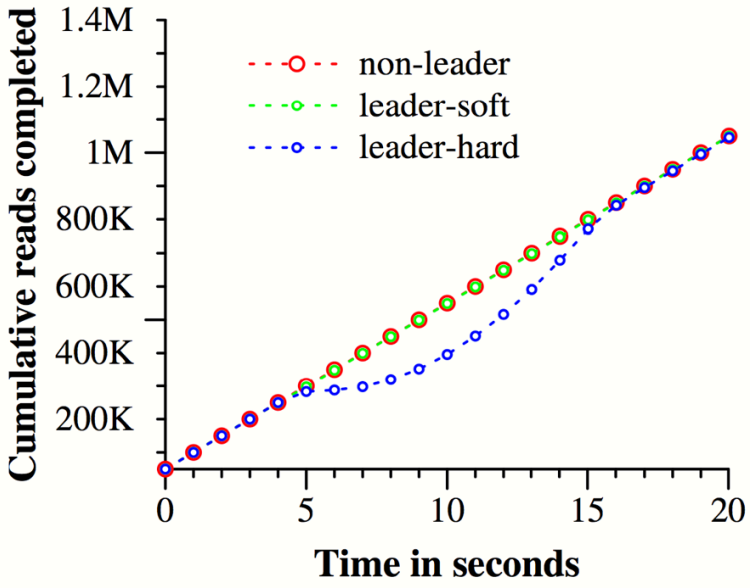


Figure 5: Effect of killing servers on throughput.

Critique

- No background on current global time synchronization techniques
- Lack of proofs of absolute error bounds in their TrueTime implementation
- External consistency? Guess at implied meaning (referenced PhD dissertation not available online)
- Pipelined Paxos? Not described. Is each replica governed by a replica-wide lock so one replica cannot undergo Paxos concurrently on disjoint rows?