CS244b - Spanner

Learning goals:

- Putting it all together
  (Paxos, 2PC, linearizability, ...)
- The power of real-time clocks
  (or change your assumptions
  when a problem is too hard)

operation	latency (ms)		count
	mean	std dev	
all reads	8.7	376.4	21.5B
single-site commit	72.3	112.8	31.2M
multi-site commit	103.0	52.2	32.1M

Table 6: F1-perceived operation latencies measured over the course of 24 hours.

```
CREATE TABLE Users {
  uid INT64 NOT NULL, email STRING
} PRIMARY KEY (uid), DIRECTORY;

CREATE TABLE Albums {
  uid INT64 NOT NULL, aid INT64 NOT NULL,
  name STRING
} PRIMARY KEY (uid, aid),
  INTERLEAVE IN PARENT Users ON DELETE CASCADE;
```

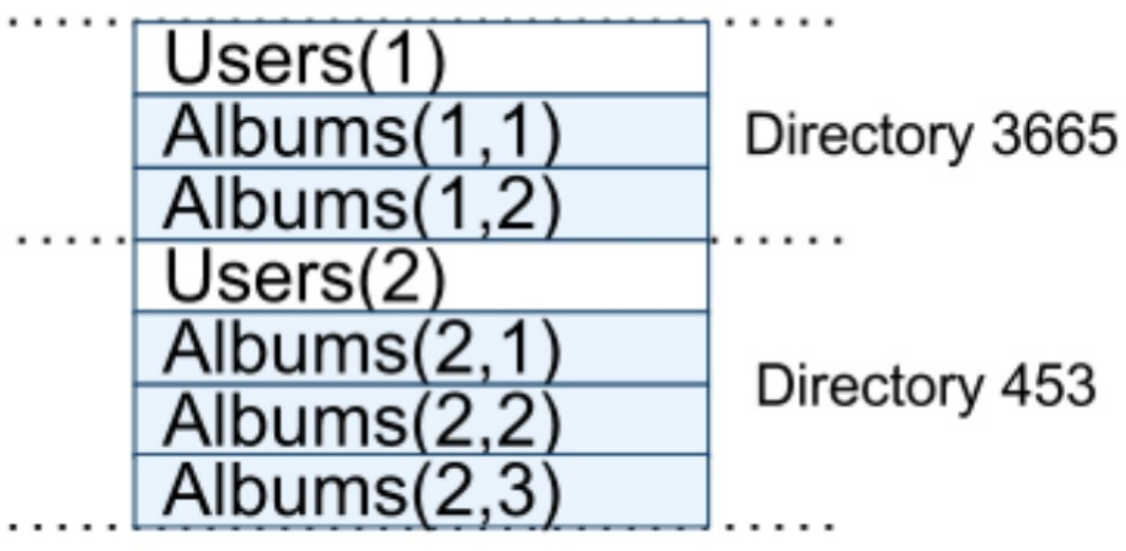


Figure 4: Example Spanner schema for photo metadata, and the interleaving implied by `INTERLEAVE IN`.

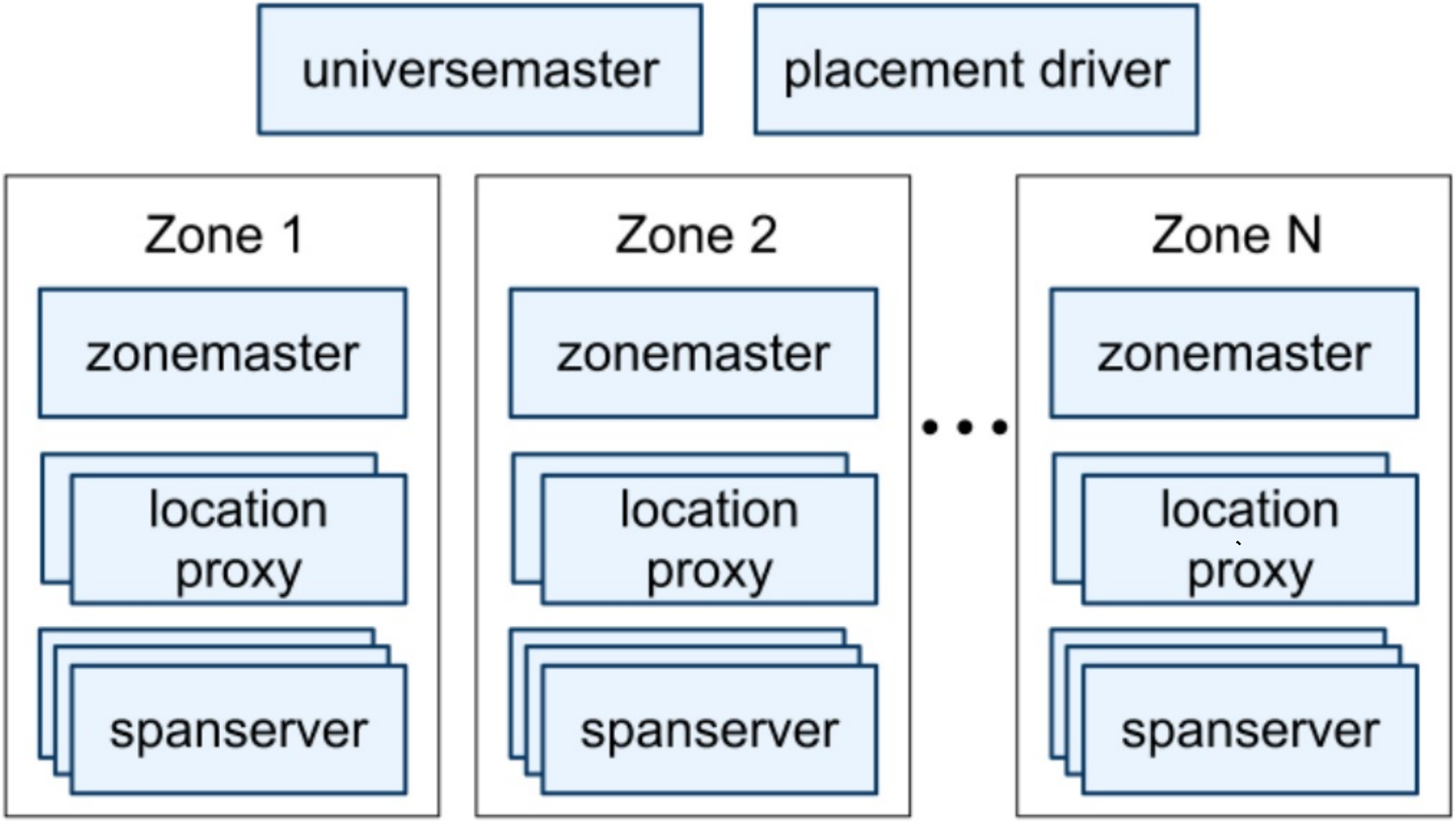
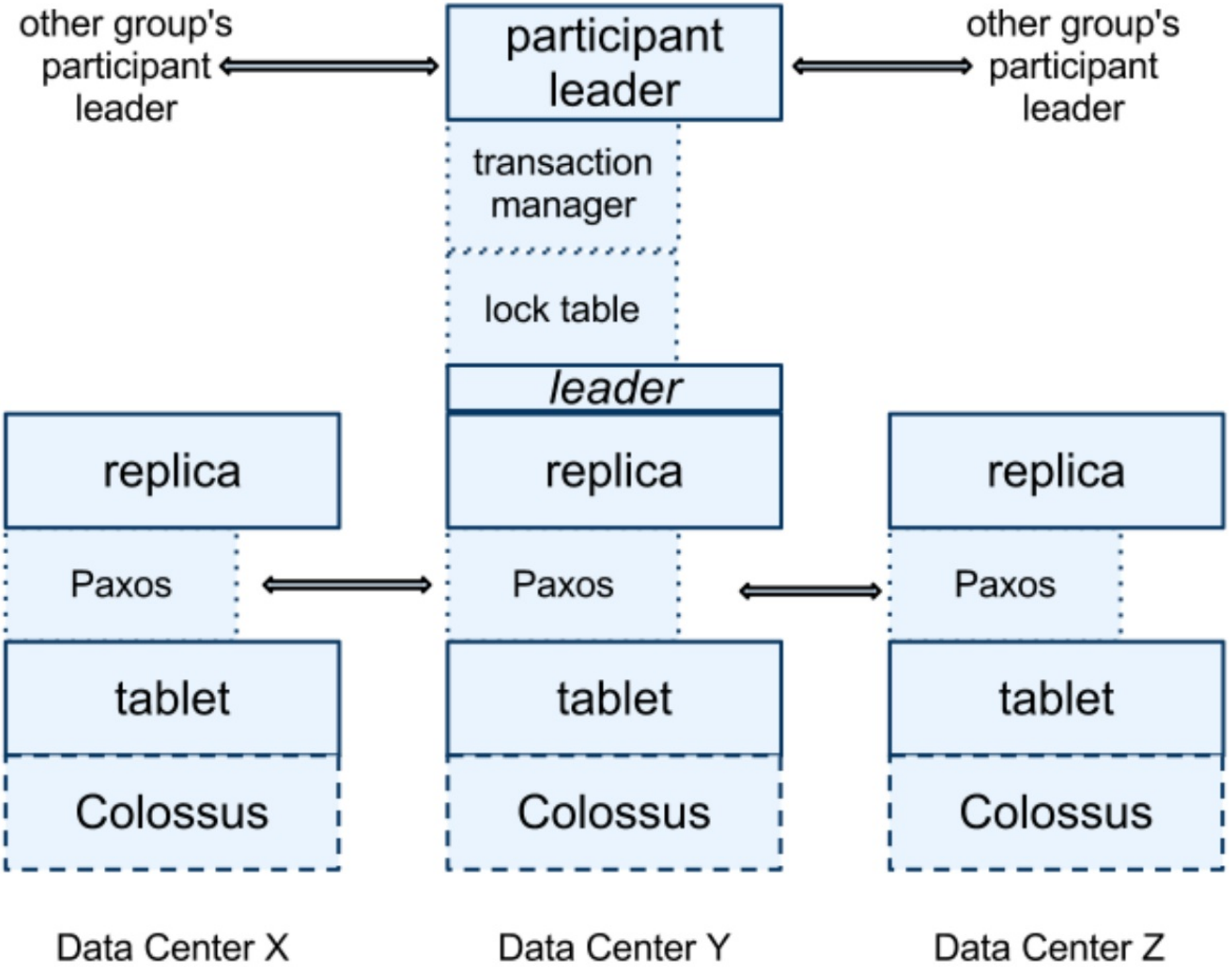


Figure 1: Spanner server organization.

tablet : map (key, timestamp) → value

Fig. 2


other group's participant leader
participant leader
other group's participant leader
transaction manager
lock table
leader
replica
replica
replica
Paxos
Paxos
Paxos
tablet
tablet
tablet
Colossus
Colossus
Colossus
Data Center X
Data Center Y
Data Center Z

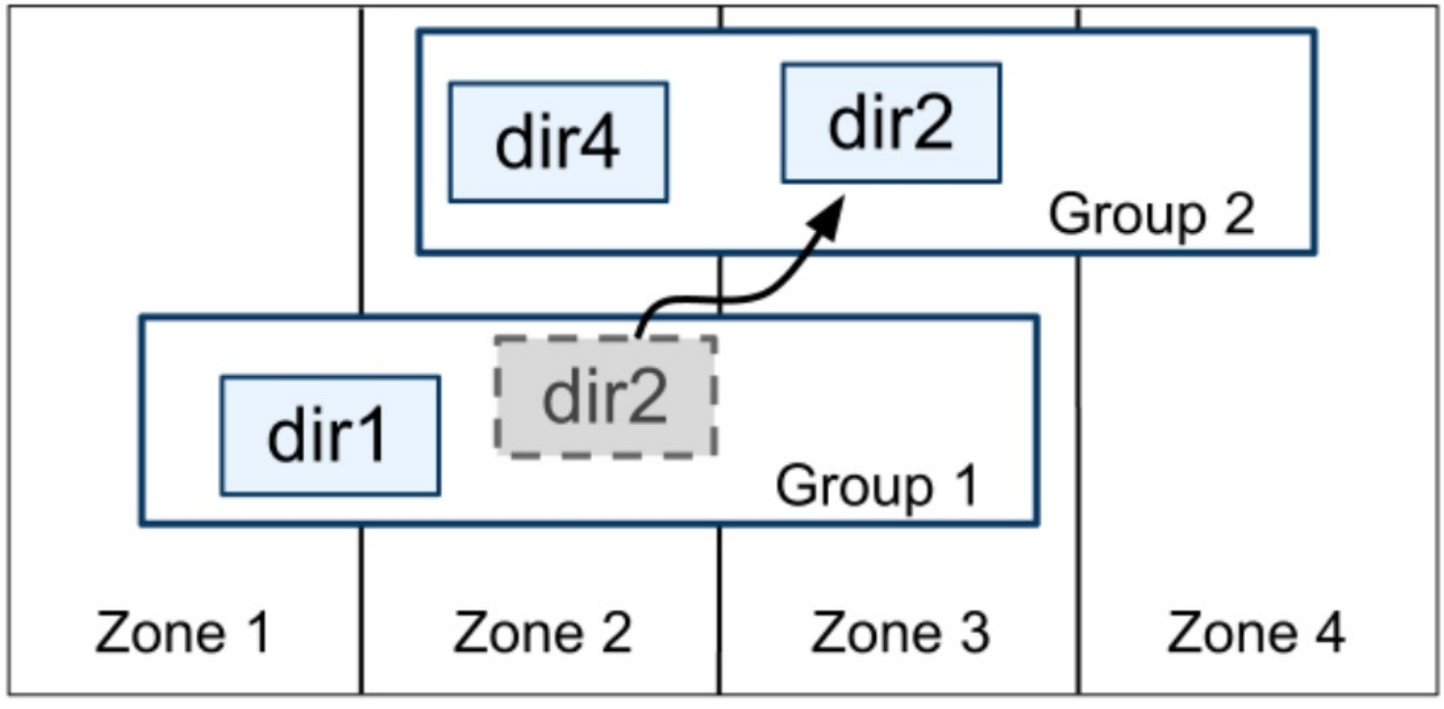


Figure 3: Directories are the unit of data movement between Paxos groups.

# fragments	# directories
1	>100M
2–4	341
5–9	5336
10–14	232
15–99	34
100–500	7

Table 5: Distribution of directory-fragment counts in F1.

p. 254

A directory is also the smallest unit whose geographic-replication properties (or *placement*, for short) can be specified by an application. The design of our placement-specification language separates responsibilities for managing replication configurations. Administrators control two dimensions: the number and types of replicas, and the geographic placement of those replicas. They create a menu of named options in these two dimensions (e.g., *North America, replicated 5 ways with 1 witness*). An application controls how data is replicated, by tagging each database and/or individual directories with a combination of those options. For example, an application might store each end-user's data in its own directory, which would enable user *A*'s data to have three replicas in Europe, and user *B*'s data to have five replicas in North America.

Straw man #1

Use 2-phase locking on Paxos
Maintain lock table at leader

Problem:

User 1: ⓐ Post A to Group 1
ⓑ Post B to Group 2

User 2: Reads A
Reads B
> concurrently

Might see B but not A

Straw man #2

Cross-Paxos-group transactions w. locking

User 2: Read A
Read B
> concurrently

would require holding locks on A & B

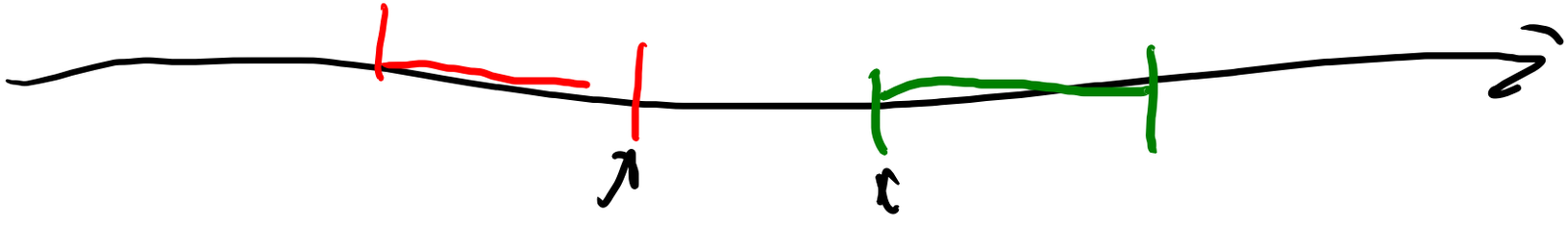
Problems

Read locks bad for scalability

All the load goes to leaders even for reads

TrueTime API

TT.now() → TTinterval [earliest, latest]



Assume drift ≤ 200 μsec/sec 0.02%

RO: Assign read Tx timestamp $\leq$
that preserves strict serializability

RW Tx:

- Client reads data - acquires read locks
- Writes buffered locally
- Commit: propagate writes to Paxos Groups
  Pick timestamp, Commit atomically

RW detail

1 Paxos group only

Client send writes to leader

Leader picks timestamp s

s must be ≥ any previously committed Tx in Paxos log

Two-phase locking → point where all locks held

s > TT.now().latest when commit request received

Commit wait: Don't reply until

s < TT.now().earliest

R/W multiple Paxos groups

- Client picks 1 paxos group to be coordinator
- Sends commit req. to coordinator leader
- Sends writes to all participant leaders
  informs them of coordinator
- Participant leaders acquire locks
  Pick <u>prepare timestamp</u> (in leader's lease)
  must be > all committed txs
- When all participants vote commit, pick s
  - $s > TT.now().latest$ when commit recv'd from client
  - $s \geq max(\text{prepare timestamps})$
  - s - must be in lease terms of all leaders
- Commit wait

Method	Returns
$TT.now()$	$TTinterval$: $[earliest, latest]$
$TT.after(t)$	true if t has definitely passed
$TT.before(t)$	true if t has definitely not arrived

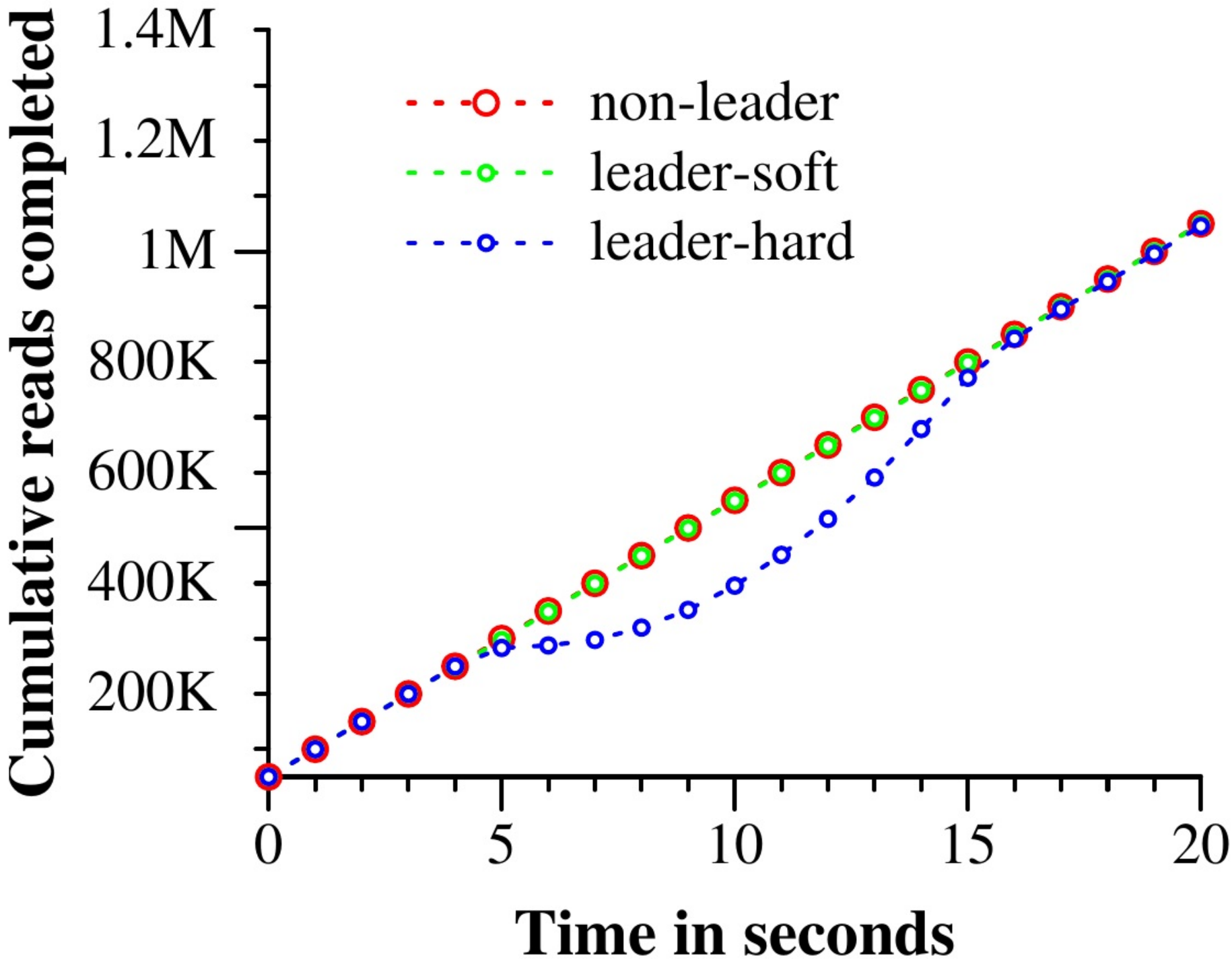
Table 1: TrueTime API. The argument t is of type *TTstamp*.

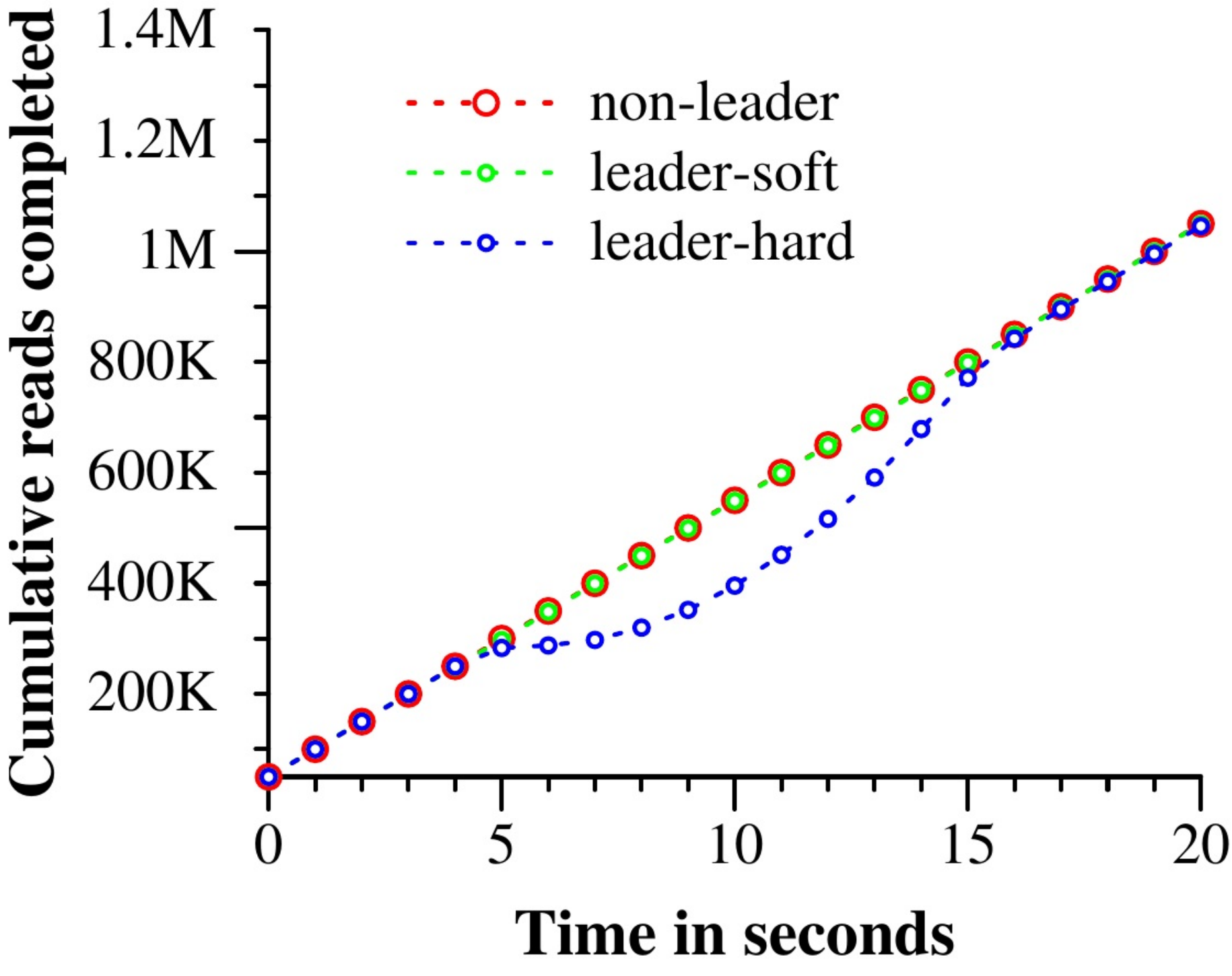
replicas	latency (ms)			throughput (Kops/sec)		
	write	read-only transaction	snapshot read	write	read-only transaction	snapshot read
1D	9.4±.6	—	—	4.0±.3	—	—
1	14.4±1.0	1.4±.1	1.3±.1	4.1±.05	10.9±.4	13.5±.1
3	13.9±.6	1.3±.1	1.2±.1	2.2±.5	13.8±3.2	38.5±.3
5	14.4±.4	1.4±.05	1.3±.04	2.8±.3	25.3±5.2	50.0±1.1

Table 3: Operation microbenchmarks. Mean and standard deviation over 10 runs. 1D means one replica with commit wait disabled.

operation	latency (ms)		count
	mean	std dev	
all reads	8.7	376.4	21.5B
single-site commit	72.3	112.8	31.2M
multi-site commit	103.0	52.2	32.1M

Table 6: F1-perceived operation latencies measured over the course of 24 hours.


Cumulative reads completed
1.4M
1.2M
1M
800K
600K
400K
200K
non-leader
leader-soft
leader-hard
0
5
10
15
20
Time in seconds


Cumulative reads completed
1.4M
1.2M
1M
800K
600K
400K
200K
non-leader
leader-soft
leader-hard
0
5
10
15
20
Time in seconds