

Spanner

Bigtable in retrospect

- Definitely a useful, scalable system!
- Still in use at Google, motivated lots of NoSQL DBs
- Biggest mistake in design (per Jeff Dean, Google): not supporting distributed transactions!
 - became really important w/ incremental updates
 - users wanted them, implemented themselves, often incorrectly!
 - at least 3 papers later fixed this

Two-phase commit Recap

- keys partitioned over different hosts; one coordinator per transaction
- acquire locks on all data read/written; release after commit
- to commit, coordinator first sends prepare message to all shards; they respond prepare_ok or abort
 - if prepare_ok, they *must* be able to commit transaction
- if all prepare_ok, coordinator sends commit to all; they write commit record and release locks

2PC Problems

- Availability: what do we do if either some shard or the coordinator fails?
 - generally: 2PC is a blocking protocol, can't make progress until it comes back up
- Performance: can we really afford to take locks and hold them for the entire commit process?

Spanner

- Backend for the F1 database, which runs the ad system
- Basic model: 2PC over Paxos
- Uses physical clocks for performance

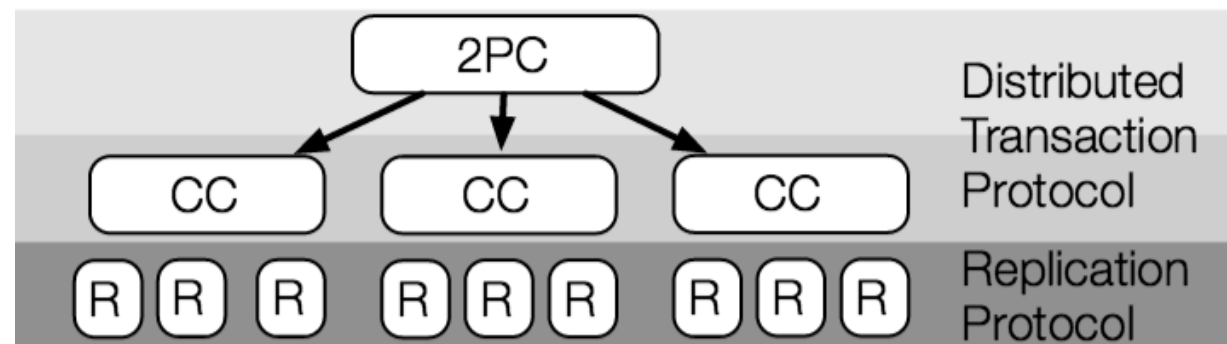
Example: social network

- simple schema: user posts and friends lists
- but sharded across thousands of machines
- each replicated across multiple continents

Example: social network

- example: generate page of friends' recent posts
- what if I remove friend X, post mean comment?
 - maybe he sees old version of friends list, new version of my posts?
- How can we solve this with locking?
 - acquire read locks on friends list, and on each friend's posts
 - prevents them from being modified concurrently
 - but potentially really slow?

Spanner architecture

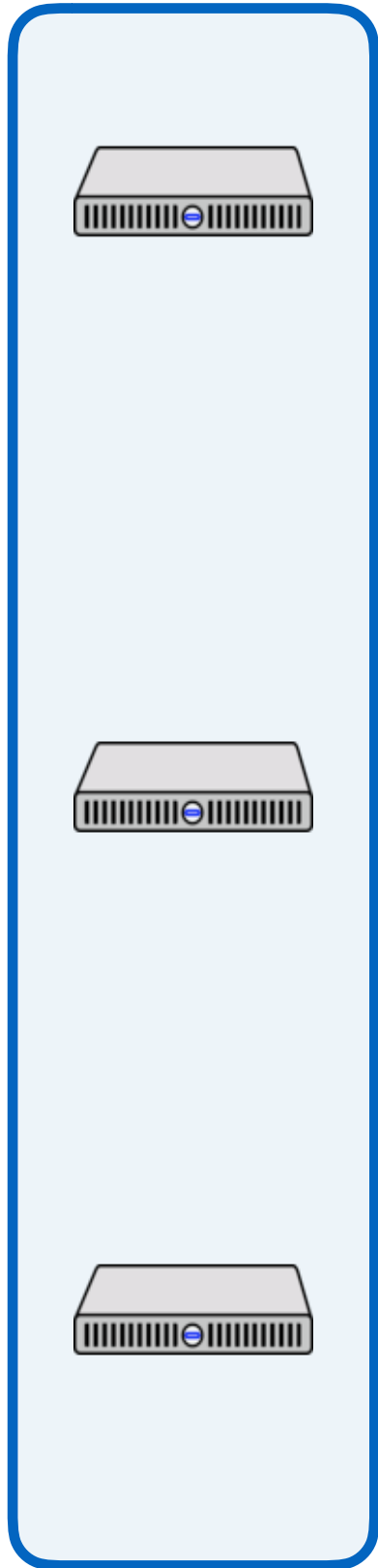


- Each shard is stored in a Paxos group
 - replicated across data centers
 - has a (relatively long-lived) leader
- Transactions span Paxos groups using 2PC
 - use 2PC for transactions
 - leader of each Paxos group tracks locks
 - one group leader becomes the 2PC coordinator, others participants

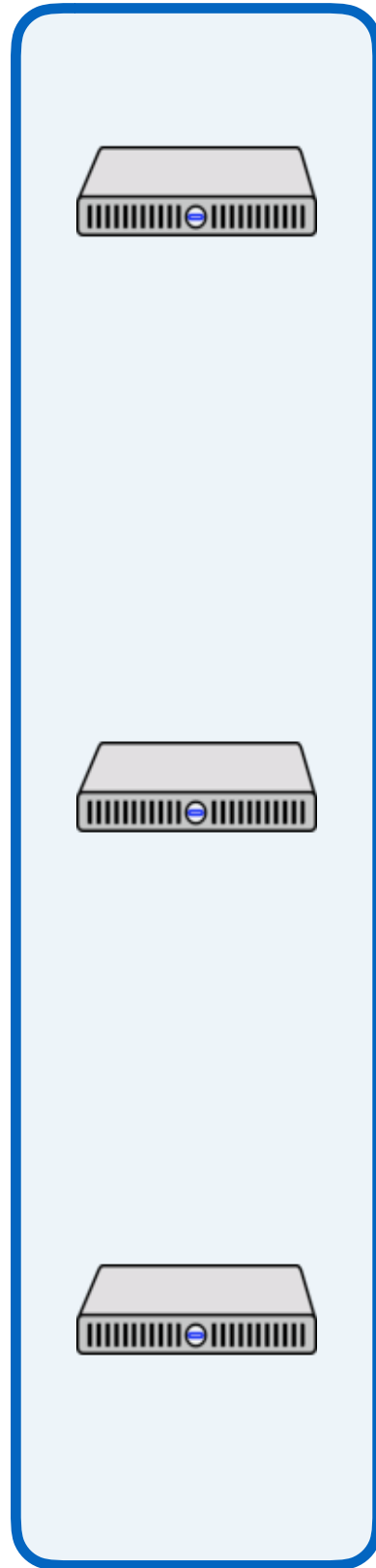
Basic 2PC/Paxos approach

- during execution, read and write objects
 - contact the appropriate Paxos group leader, acquire locks
- client decides to commit, notifies the coordinator
 - coordinator contacts all shards, sends PREPARE message
 - they Paxos-replicate a prepare log entry (including locks),
 - vote either ok or abort
- if all shards vote OK, coordinator sends commit message
 - each shard Paxos-replicates commit entry
 - leader releases locks

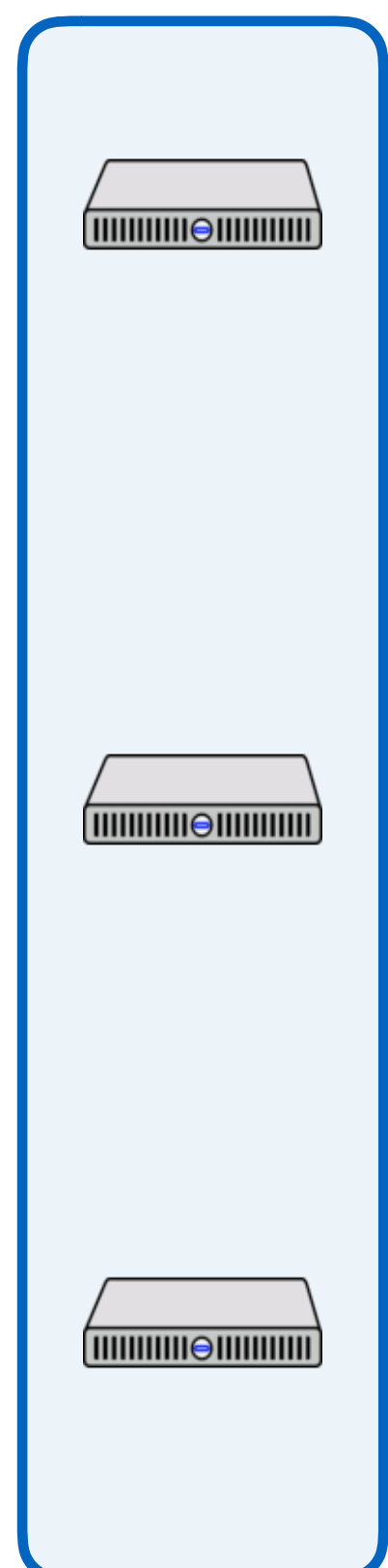
DC1



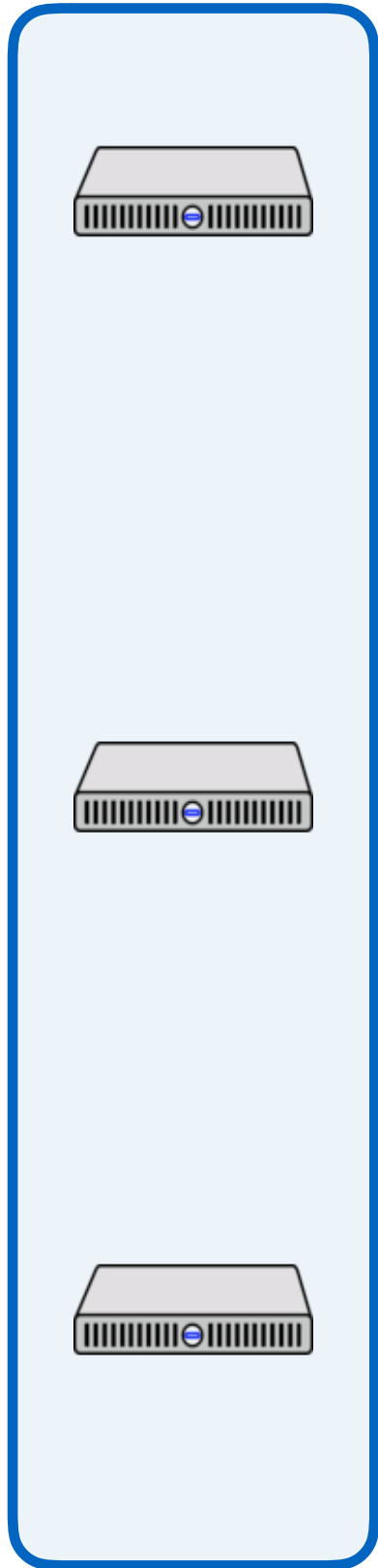
DC2



DC3



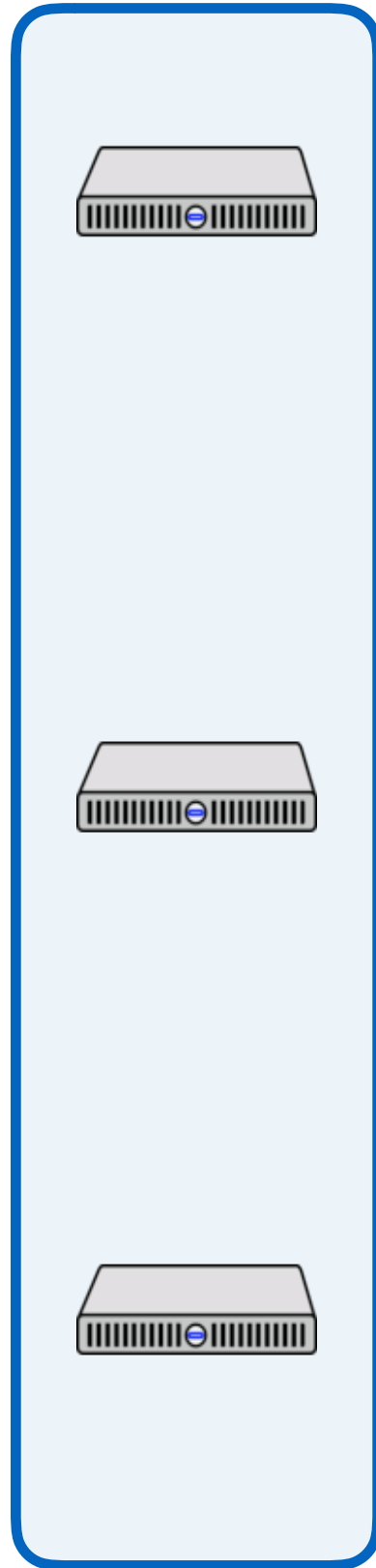
DC1



Paxos



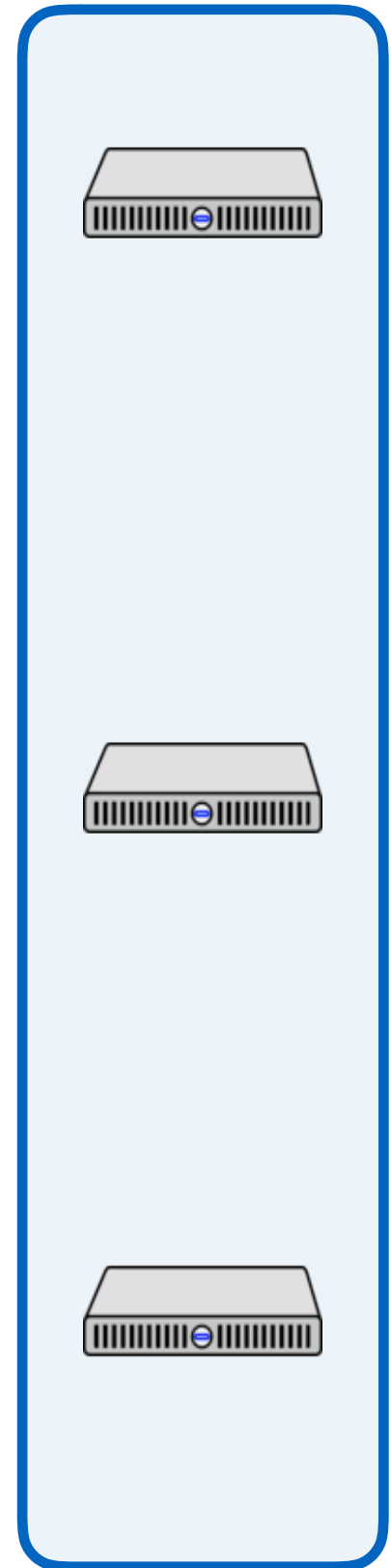
DC2

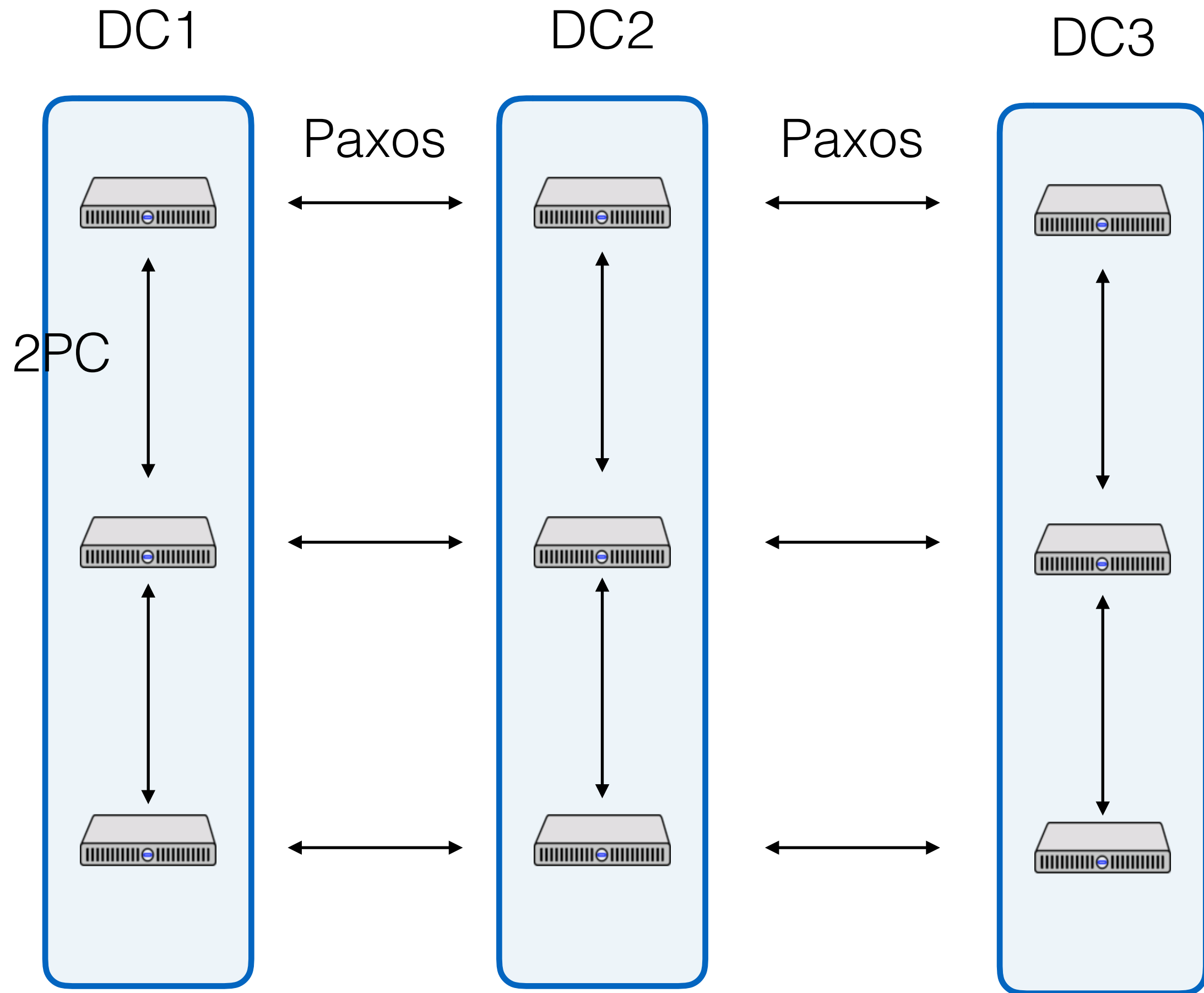


Paxos



DC3





Basic 2PC/Paxos approach

- Note that this is really the same as basic 2PC from before
- Just replaced writes to a log on disk with writes to a Paxos replicated log!
- It is linearizable (= strict serializable = externally consistent)

- So what's left?
 - Lock-free read-only transactions

Lock-free r/o transactions

- Key idea: assign meaningful timestamp to transaction
 - such that timestamps are enough to order transactions meaningfully
- Keep a history of versions around on each node
- Then, reasonable to say:
r/o transaction X reads at timestamp 10

Spanner pt 2

1. Distributed transactions, in detail
 - On one Paxos group
 - Between Paxos groups
2. Fast read-only transactions with TrueTime
3. Discussion

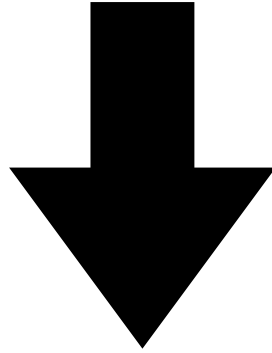
Leader



Client



Follower



Follower



```
x = read(checking_bal)
if (x > 100) {
    y = read(savings_bal)
    write(checking_bal, x - 100)
    write(savings_bal, y + 100)
}
```

LOG:

- 1.
- 2.
- 3.
- 4.
- 5.
- 6.
- 7.

Leader



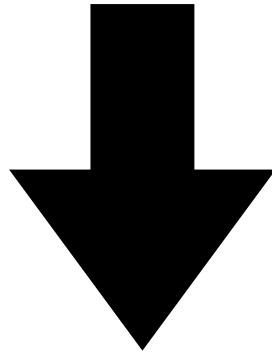
Client



read(checking_bal)



Follower



Follower



```
x = read(checking_bal)
if (x > 100) {
    y = read(savings_bal)
    write(checking_bal, x - 100)
    write(savings_bal, y + 100)
}
```

LOG:

- 1.
- 2.
- 3.
- 4.
- 5.
- 6.
- 7.

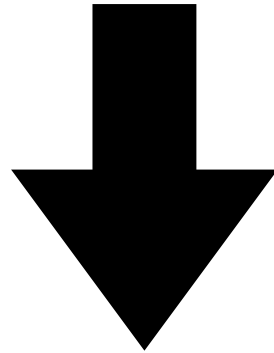
Leader



Client



Follower



Follower



```
x = read(checking_bal)
if (x > 100) {
    y = read(savings_bal)
    write(checking_bal, x - 100)
    write(savings_bal, y + 100)
}
```

LOG:

1. A: Lock(R, checking_bal)
- 2.
- 3.
- 4.
- 5.
- 6.
- 7.

Leader



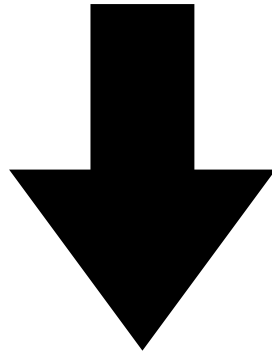
200



Client



Follower



Follower



```
x = read(checking_bal)
if (x > 100) {
    y = read(savings_bal)
    write(checking_bal, x - 100)
    write(savings_bal, y + 100)
}
```

LOG:

1. A: Lock(R, checking_bal)
- 2.
- 3.
- 4.
- 5.
- 6.
- 7.

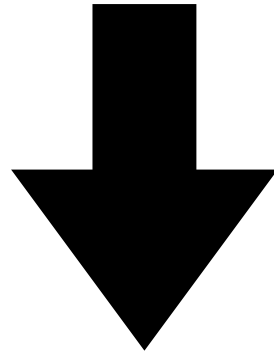
Leader



Client



Follower



Follower



```
x = read(checking_bal)
if (x > 100) {
    y = read(savings_bal)
    write(checking_bal, x - 100)
    write(savings_bal, y + 100)
}
```

LOG:

1. A: Lock(R, checking_bal)
- 2.
- 3.
- 4.
- 5.
- 6.
- 7.

Leader



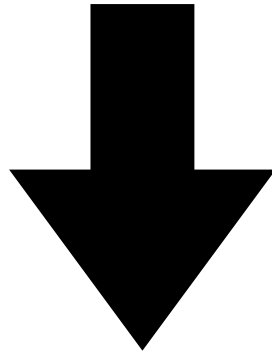
Client



`read(savings_bal)`



Follower



Follower



```
x = read(checking_bal)
if (x > 100) {
    y = read(savings_bal)
    write(checking_bal, x - 100)
    write(savings_bal, y + 100)
}
```

LOG:

1. A: Lock(R, checking_bal)
- 2.
- 3.
- 4.
- 5.
- 6.
- 7.

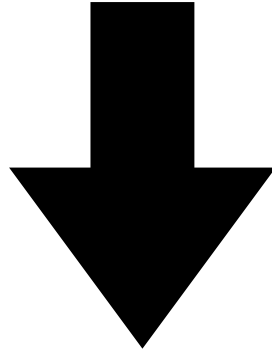
Leader



Client



Follower



Follower



```
x = read(checking_bal)
if (x > 100) {
    y = read(savings_bal)
    write(checking_bal, x - 100)
    write(savings_bal, y + 100)
}
```

LOG:

1. A: Lock(R, checking_bal)
2. A: Lock(R, savings_bal)
- 3.
- 4.
- 5.
- 6.
- 7.

Leader

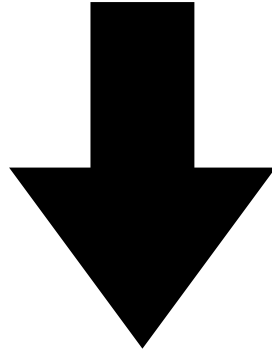


0

Client



Follower



Follower



```
x = read(checking_bal)
if (x > 100) {
    y = read(savings_bal)
    write(checking_bal, x - 100)
    write(savings_bal, y + 100)
}
```

LOG:

1. A: Lock(R, checking_bal)
2. A: Lock(R, savings_bal)
- 3.
- 4.
- 5.
- 6.
- 7.

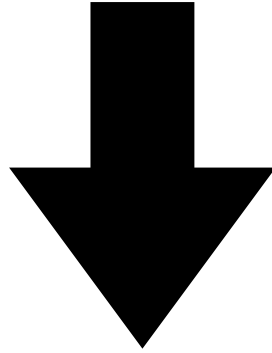
Leader



Client



Follower



Follower



```
x = read(checking_bal)
if (x > 100) {
    y = read(savings_bal)
    write(checking_bal, x - 100)
    write(savings_bal, y + 100)
}
```

LOG:

1. A: Lock(R, checking_bal)
2. A: Lock(R, savings_bal)
- 3.
- 4.
- 5.
- 6.
- 7.

Leader



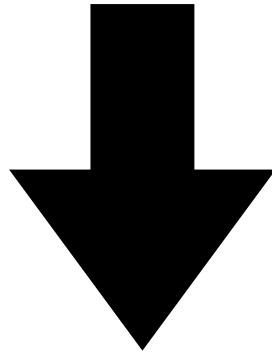
Client



`write(checking_bal, 100)`



Follower



Follower



```
x = read(checking_bal)
if (x > 100) {
    y = read(savings_bal)
    write(checking_bal, x - 100)
    write(savings_bal, y + 100)
}
```

LOG:

1. A: Lock(R, checking_bal)
2. A: Lock(R, savings_bal)
- 3.
- 4.
- 5.
- 6.
- 7.

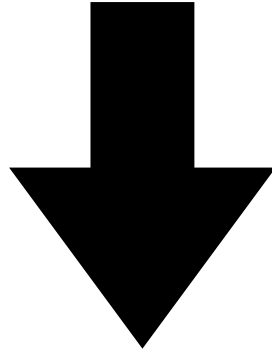
Leader



Client



Follower



Follower



```
x = read(checking_bal)
if (x > 100) {
    y = read(savings_bal)
    write(checking_bal, x - 100)
    write(savings_bal, y + 100)
}
```

LOG:

1. A: Lock(R, checking_bal)
2. A: Lock(R, savings_bal)
3. A: Lock(W, checking_bal)
4. A: Write(checking_bal, 100)
- 5.
- 6.
- 7.

Leader

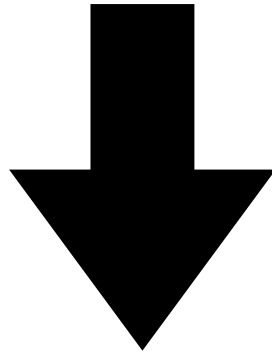


OK

Client



Follower



Follower



```
x = read(checking_bal)
if (x > 100) {
    y = read(savings_bal)
    write(checking_bal, x - 100)
    write(savings_bal, y + 100)
}
```

LOG:

1. A: Lock(R, checking_bal)
2. A: Lock(R, savings_bal)
3. A: Lock(W, checking_bal)
4. A: Write(checking_bal, 100)
- 5.
- 6.
- 7.

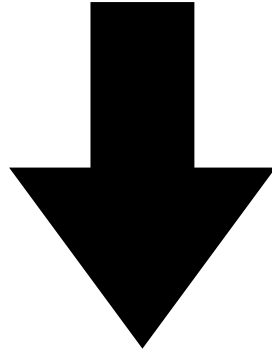
Leader



Client



Follower



Follower



```
x = read(checking_bal)
if (x > 100) {
    y = read(savings_bal)
    write(checking_bal, x - 100)
    write(savings_bal, y + 100)
}
```

LOG:

1. A: Lock(R, checking_bal)
2. A: Lock(R, savings_bal)
3. A: Lock(W, checking_bal)
4. A: Write(checking_bal, 100)
- 5.
- 6.
- 7.

Leader



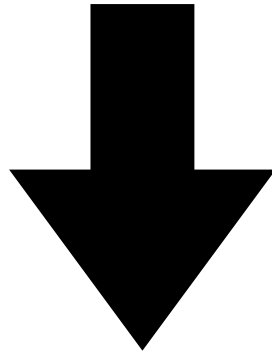
Client



`write(savings_bal, 100)`



Follower



Follower



```
x = read(checking_bal)
if (x > 100) {
    y = read(savings_bal)
    write(checking_bal, x - 100)
    write(savings_bal, y + 100)
}
```

LOG:

1. A: Lock(R, checking_bal)
2. A: Lock(R, savings_bal)
3. A: Lock(W, checking_bal)
4. A: Write(checking_bal, 100)
- 5.
- 6.
- 7.

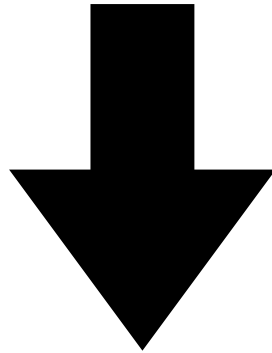
Leader



Client



Follower



Follower



```
x = read(checking_bal)
if (x > 100) {
    y = read(savings_bal)
    write(checking_bal, x - 100)
    write(savings_bal, y + 100)
}
```

LOG:

1. A: Lock(R, checking_bal)
2. A: Lock(R, savings_bal)
3. A: Lock(W, checking_bal)
4. A: Write(checking_bal, 100)
5. A: Lock(W, savings_bal)
6. A: Write(savings_bal, 100)
- 7.

Leader



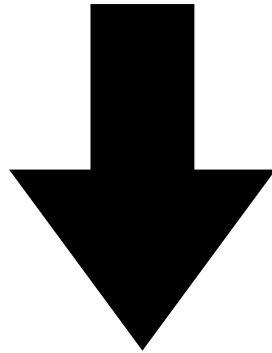
OK



Client



Follower



Follower



```
x = read(checking_bal)
if (x > 100) {
    y = read(savings_bal)
    write(checking_bal, x - 100)
    write(savings_bal, y + 100)
}
```

LOG:

1. A: Lock(R, checking_bal)
2. A: Lock(R, savings_bal)
3. A: Lock(W, checking_bal)
4. A: Write(checking_bal, 100)
5. A: Lock(W, savings_bal)
6. A: Write(savings_bal, 100)
- 7.

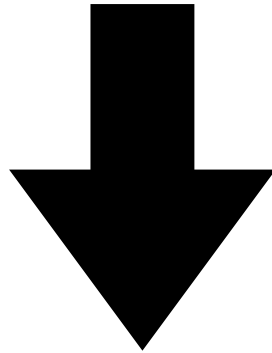
Leader



Client



Follower



Follower



```
x = read(checking_bal)
if (x > 100) {
    y = read(savings_bal)
    write(checking_bal, x - 100)
    write(savings_bal, y + 100)
}
```

LOG:

1. A: Lock(R, checking_bal)
2. A: Lock(R, savings_bal)
3. A: Lock(W, checking_bal)
4. A: Write(checking_bal, 100)
5. A: Lock(W, savings_bal)
6. A: Write(savings_bal, 100)
- 7.

Leader



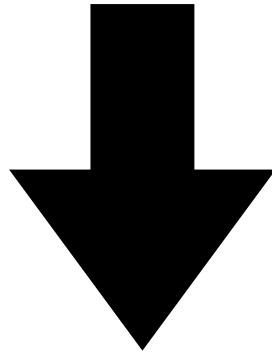
Client



commit



Follower



Follower



```
x = read(checking_bal)
if (x > 100) {
    y = read(savings_bal)
    write(checking_bal, x - 100)
    write(savings_bal, y + 100)
}
```

LOG:

1. A: Lock(R, checking_bal)
2. A: Lock(R, savings_bal)
3. A: Lock(W, checking_bal)
4. A: Write(checking_bal, 100)
5. A: Lock(W, savings_bal)
6. A: Write(savings_bal, 100)
- 7.

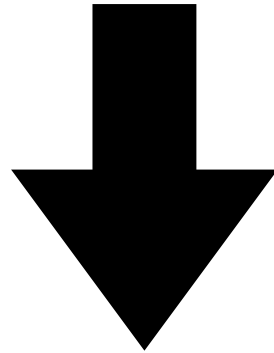
Leader



Client



Follower



Follower



```
x = read(checking_bal)
if (x > 100) {
    y = read(savings_bal)
    write(checking_bal, x - 100)
    write(savings_bal, y + 100)
}
```

LOG:

1. A: Lock(R, checking_bal)
2. A: Lock(R, savings_bal)
3. A: Lock(W, checking_bal)
4. A: Write(checking_bal, 100)
5. A: Lock(W, savings_bal)
6. A: Write(savings_bal, 100)
7. A: Commit

Leader



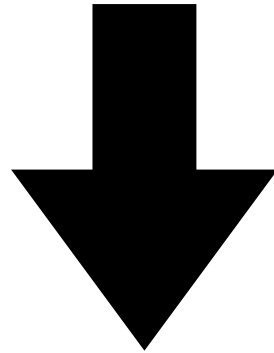
OK



Client



Follower



Follower



```
x = read(checking_bal)
if (x > 100) {
    y = read(savings_bal)
    write(checking_bal, x - 100)
    write(savings_bal, y + 100)
}
```

LOG:

1. A: Lock(R, checking_bal)
2. A: Lock(R, savings_bal)
3. A: Lock(W, checking_bal)
4. A: Write(checking_bal, 100)
5. A: Lock(W, savings_bal)
6. A: Write(savings_bal, 100)
7. A: Commit

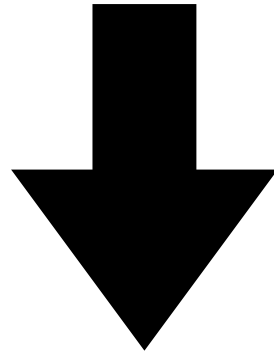
Leader



Client



Follower



Follower



```
x = read(checking_bal)
if (x > 100) {
    y = read(savings_bal)
    write(checking_bal, x - 100)
    write(savings_bal, y + 100)
}
```

LOG:

1. A: Lock(R, checking_bal)
2. A: Lock(R, savings_bal)
3. A: Lock(W, checking_bal)
4. A: Write(checking_bal, 100)
5. A: Lock(W, savings_bal)
6. A: Write(savings_bal, 100)
7. A: Commit
8. A: Unlock(checking_bal)
9. A: Unlock(savings_bal)

Leader 1



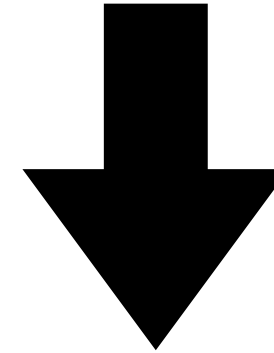
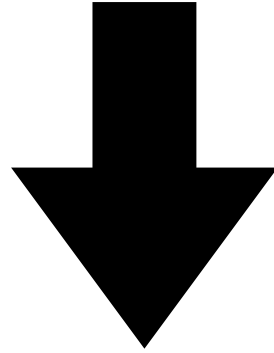
Client



Leader 2



```
x = read(checking_bal)
if (x > 100) {
  y = read(savings_bal)
  write(checking_bal, x - 100)
  write(savings_bal, y + 100)
}
```



LOG:

LOG:

Leader 1



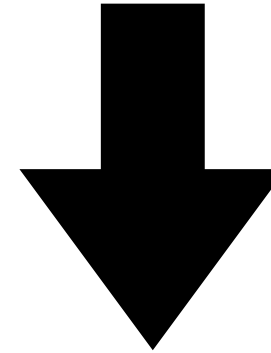
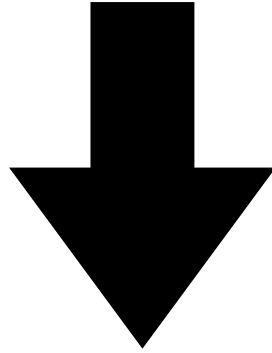
Client



Leader 2



```
x = read(checking_bal)
if (x > 100) {
  y = read(savings_bal)
  write(checking_bal, x - 100)
  write(savings_bal, y + 100)
}
```



LOG:

LOG:

Leader 1



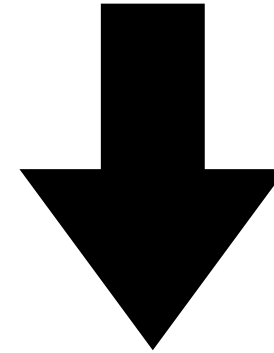
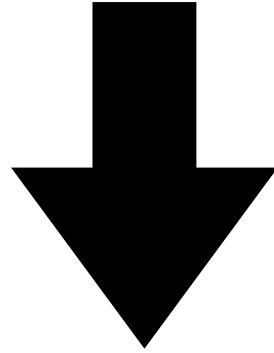
Client



Leader 2



```
x = read(checking_bal)
if (x > 100) {
  y = read(savings_bal)
  write(checking_bal, x - 100)
  write(savings_bal, y + 100)
}
```



LOG:

1. A: Lock(R, checking_bal)

LOG:

Leader 1



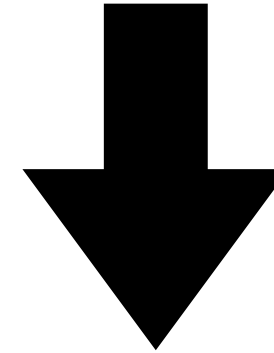
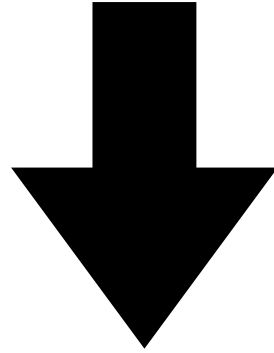
Client



Leader 2



```
x = read(checking_bal)
if (x > 100) {
  y = read(savings_bal)
  write(checking_bal, x - 100)
  write(savings_bal, y + 100)
}
```



LOG:

1. A: Lock(R, checking_bal)

LOG:

Leader 1



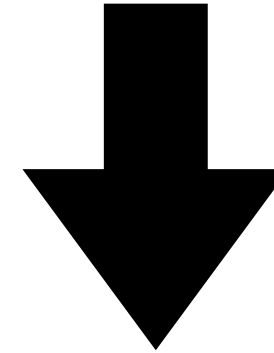
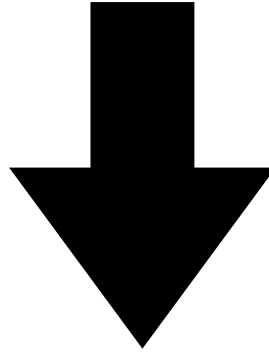
Client



Leader 2



```
x = read(checking_bal)
if (x > 100) {
  y = read(savings_bal)
  write(checking_bal, x - 100)
  write(savings_bal, y + 100)
}
```



LOG:

1. A: Lock(R, checking_bal)

LOG:

1. A: Lock(R, savings_bal)

Leader 1



Client

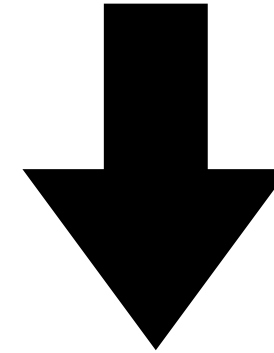
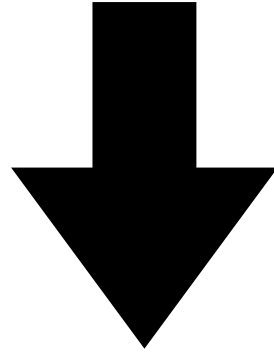


Leader 2



←

```
x = read(checking_bal)
if (x > 100) {
  y = read(savings_bal)
  write(checking_bal, x - 100)
  write(savings_bal, y + 100)
}
```



LOG:

1. A: Lock(R, checking_bal)

LOG:

1. A: Lock(R, savings_bal)

Leader 1



Client

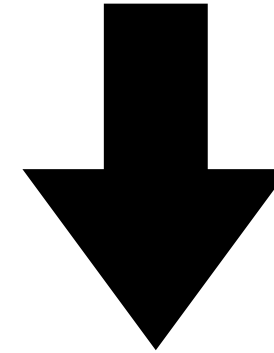
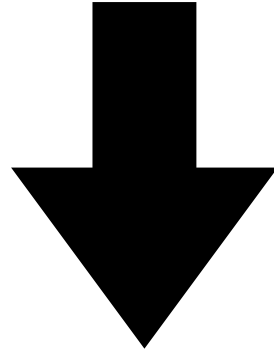


Leader 2



←

```
x = read(checking_bal)
if (x > 100) {
  y = read(savings_bal)
  write(checking_bal, x - 100)
  write(savings_bal, y + 100)
}
```



LOG:

1. A: Lock(R, checking_bal)
2. A: Lock(W, checking_bal)
3. A: Write(checking_bal, 100)

LOG:

1. A: Lock(R, savings_bal)

Leader 1



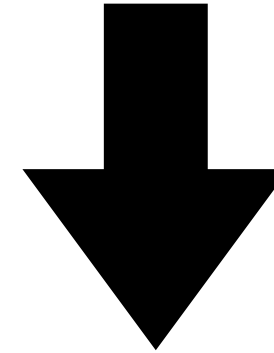
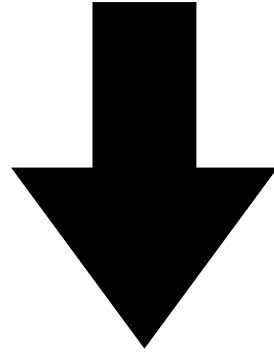
Client



Leader 2



```
x = read(checking_bal)
if (x > 100) {
  y = read(savings_bal)
  write(checking_bal, x - 100)
  write(savings_bal, y + 100)
}
```



LOG:

1. A: Lock(R, checking_bal)
2. A: Lock(W, checking_bal)
3. A: Write(checking_bal, 100)

LOG:

1. A: Lock(R, savings_bal)

Leader 1



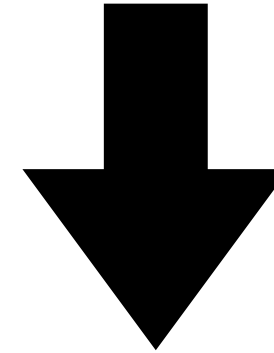
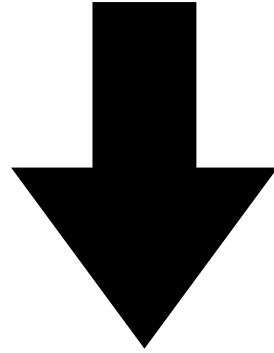
Client



Leader 2



```
x = read(checking_bal)
if (x > 100) {
  y = read(savings_bal)
  write(checking_bal, x - 100)
  write(savings_bal, y + 100)
}
```



LOG:

1. A: Lock(R, checking_bal)
2. A: Lock(W, checking_bal)
3. A: Write(checking_bal, 100)

LOG:

1. A: Lock(R, savings_bal)
2. A: Lock(W, savings_bal)
3. A: Write(savings_bal, 100)

Leader 1



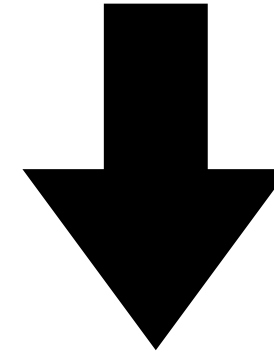
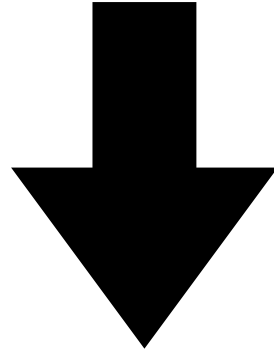
Client



Leader 2



```
x = read(checking_bal)
if (x > 100) {
  y = read(savings_bal)
  write(checking_bal, x - 100)
  write(savings_bal, y + 100)
}
```



LOG:

1. A: Lock(R, checking_bal)
2. A: Lock(W, checking_bal)
3. A: Write(checking_bal, 100)

LOG:

1. A: Lock(R, savings_bal)
2. A: Lock(W, savings_bal)
3. A: Write(savings_bal, 100)

Leader 1



commit(1)



Client



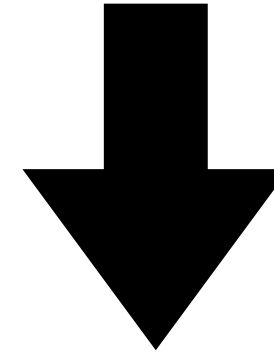
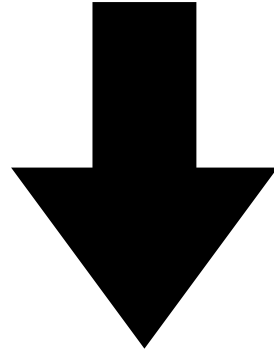
commit(1)



Leader 2



```
x = read(checking_bal)
if (x > 100) {
  y = read(savings_bal)
  write(checking_bal, x - 100)
  write(savings_bal, y + 100)
}
```



LOG:

1. A: Lock(R, checking_bal)
2. A: Lock(W, checking_bal)
3. A: Write(checking_bal, 100)

LOG:

1. A: Lock(R, savings_bal)
2. A: Lock(W, savings_bal)
3. A: Write(savings_bal, 100)

Leader 1



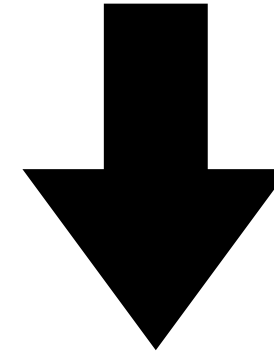
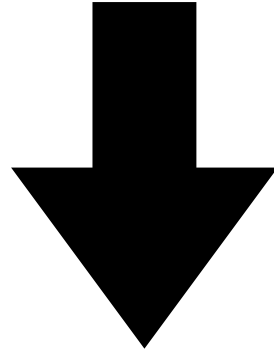
Client



Leader 2



```
x = read(checking_bal)
if (x > 100) {
  y = read(savings_bal)
  write(checking_bal, x - 100)
  write(savings_bal, y + 100)
}
```



LOG:

1. A: Lock(R, checking_bal)
2. A: Lock(W, checking_bal)
3. A: Write(checking_bal, 100)

LOG:

1. A: Lock(R, savings_bal)
2. A: Lock(W, savings_bal)
3. A: Write(savings_bal, 100)
4. A: Prepare

Leader 1



Client

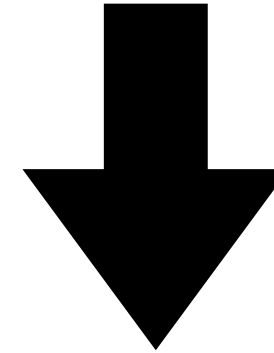
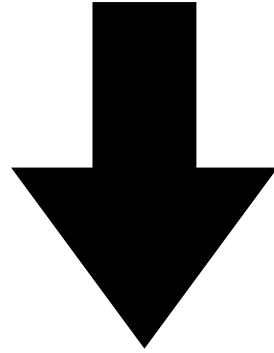


OK

Leader 2



```
x = read(checking_bal)
if (x > 100) {
  y = read(savings_bal)
  write(checking_bal, x - 100)
  write(savings_bal, y + 100)
}
```



LOG:

1. A: Lock(R, checking_bal)
2. A: Lock(W, checking_bal)
3. A: Write(checking_bal, 100)

LOG:

1. A: Lock(R, savings_bal)
2. A: Lock(W, savings_bal)
3. A: Write(savings_bal, 100)
4. A: Prepare

Leader 1



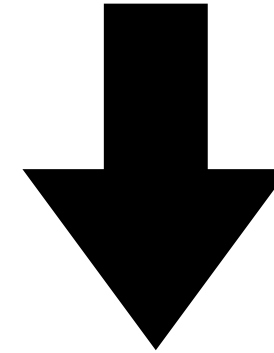
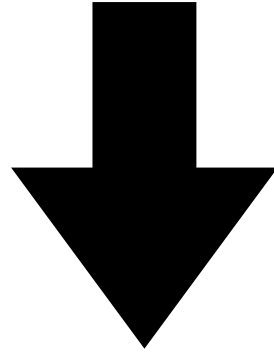
Client



Leader 2



```
x = read(checking_bal)
if (x > 100) {
  y = read(savings_bal)
  write(checking_bal, x - 100)
  write(savings_bal, y + 100)
}
```



LOG:

1. A: Lock(R, checking_bal)
2. A: Lock(W, checking_bal)
3. A: Write(checking_bal, 100)
4. A: Commit

LOG:

1. A: Lock(R, savings_bal)
2. A: Lock(W, savings_bal)
3. A: Write(savings_bal, 100)
4. A: Prepare

Leader 1



OK



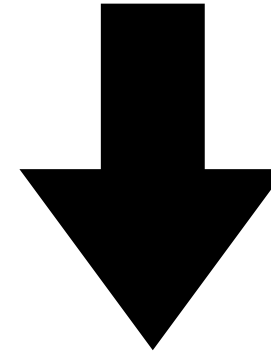
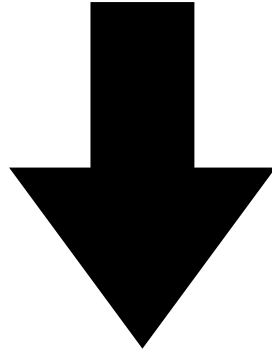
Client



Leader 2



```
x = read(checking_bal)
if (x > 100) {
  y = read(savings_bal)
  write(checking_bal, x - 100)
  write(savings_bal, y + 100)
}
```



LOG:

1. A: Lock(R, checking_bal)
2. A: Lock(W, checking_bal)
3. A: Write(checking_bal, 100)
4. A: Commit

LOG:

1. A: Lock(R, savings_bal)
2. A: Lock(W, savings_bal)
3. A: Write(savings_bal, 100)
4. A: Prepare

Leader 1



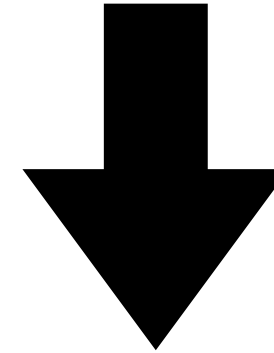
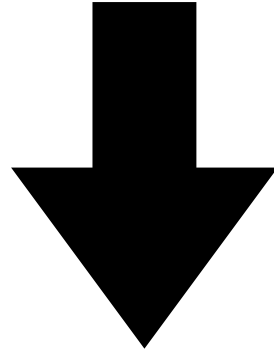
Client



Leader 2



```
x = read(checking_bal)
if (x > 100) {
  y = read(savings_bal)
  write(checking_bal, x - 100)
  write(savings_bal, y + 100)
}
```



LOG:

1. A: Lock(R, checking_bal)
2. A: Lock(W, checking_bal)
3. A: Write(checking_bal, 100)
4. A: Commit
5. A: Unlock(checking_bal)

LOG:

1. A: Lock(R, savings_bal)
2. A: Lock(W, savings_bal)
3. A: Write(savings_bal, 100)
4. A: Prepare

Leader 1



commit

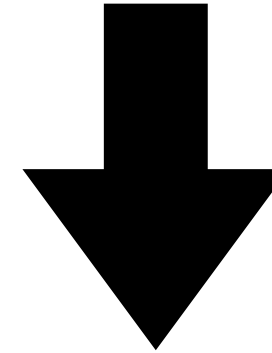
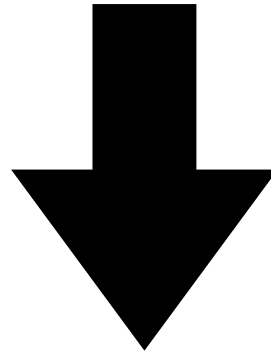
Client



Leader 2



```
x = read(checking_bal)
if (x > 100) {
  y = read(savings_bal)
  write(checking_bal, x - 100)
  write(savings_bal, y + 100)
}
```



LOG:

1. A: Lock(R, checking_bal)
2. A: Lock(W, checking_bal)
3. A: Write(checking_bal, 100)
4. A: Commit
5. A: Unlock(checking_bal)

LOG:

1. A: Lock(R, savings_bal)
2. A: Lock(W, savings_bal)
3. A: Write(savings_bal, 100)
4. A: Prepare

Leader 1



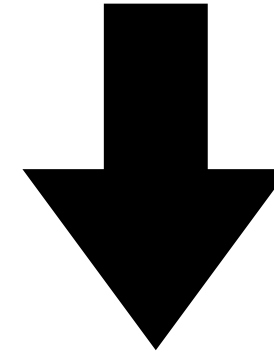
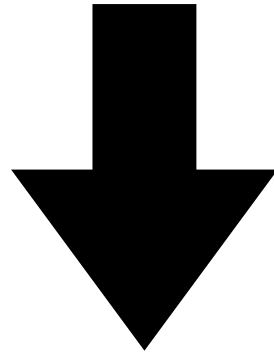
Client



Leader 2



```
x = read(checking_bal)
if (x > 100) {
  y = read(savings_bal)
  write(checking_bal, x - 100)
  write(savings_bal, y + 100)
}
```



LOG:

1. A: Lock(R, checking_bal)
2. A: Lock(W, checking_bal)
3. A: Write(checking_bal, 100)
4. A: Commit
4. A: Unlock(checking_bal)

LOG:

1. A: Lock(R, savings_bal)
2. A: Lock(W, savings_bal)
3. A: Write(savings_bal, 100)
4. A: Prepare
5. A: Commit

How can we get fast reads?

R/W transactions are complicated!

- And slow

Can we do fast, lock-free reads?

- Real time to the rescue

TrueTime

API that exposes real time, with uncertainty

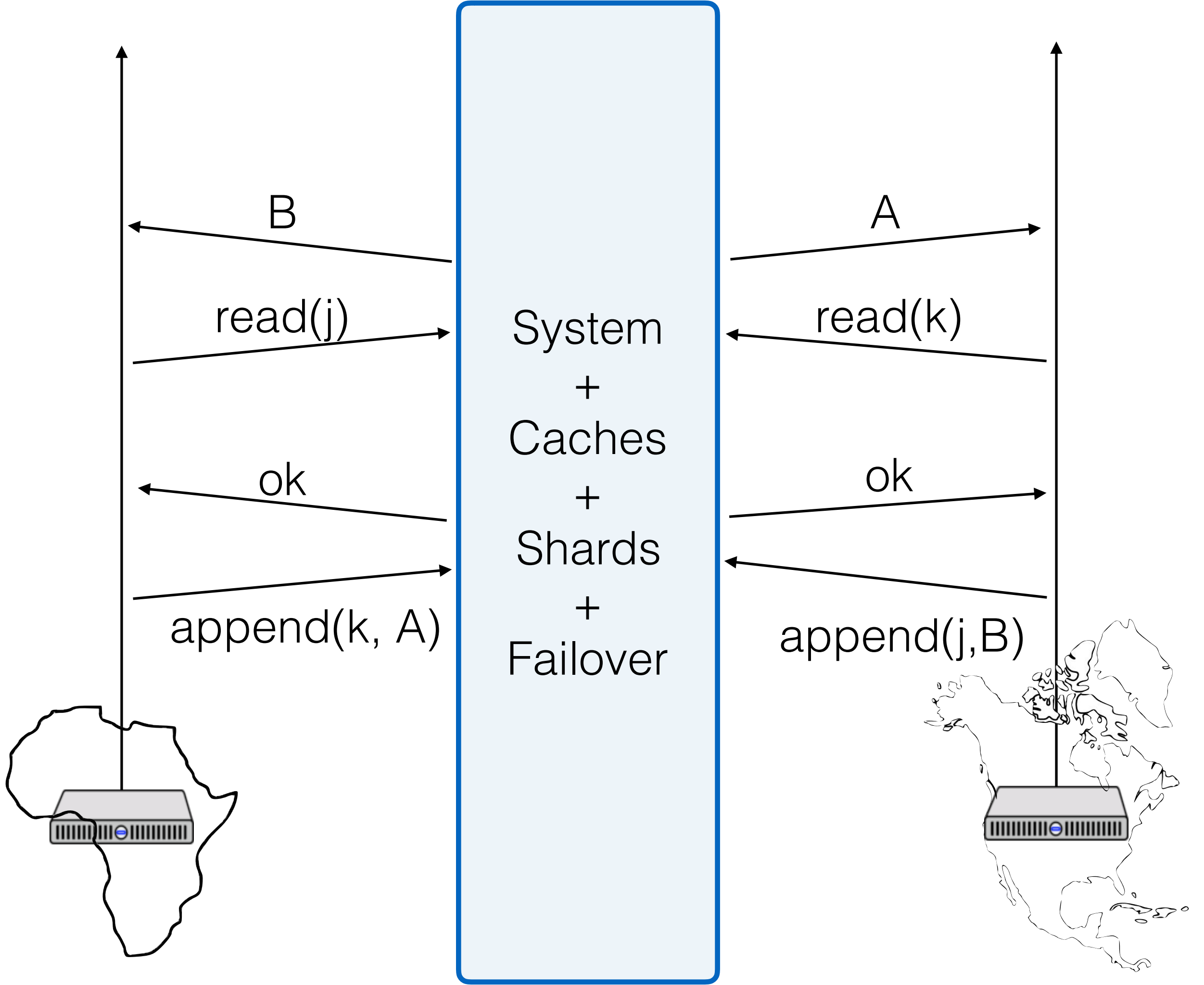
```
{earliest: e, latest: l} = TT.now()
```

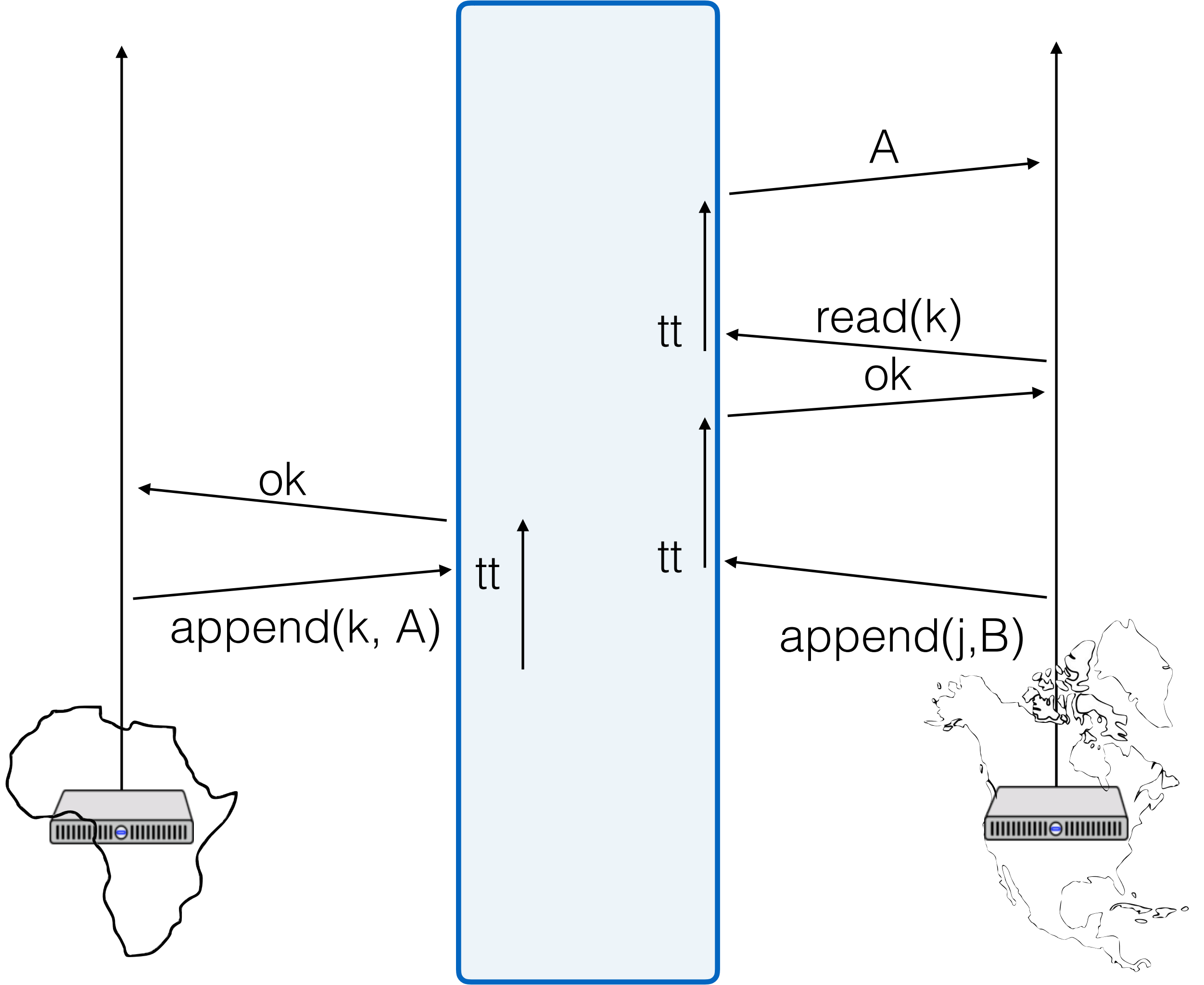
“Real time” is between `earliest` and `latest`

Time is an illusion!

If I call `TT.now()` on two nodes simultaneously, intervals *guaranteed* to overlap!

If intervals don't overlap, the later one happened later!





TrueTime usage

Assign a timestamp to each transaction

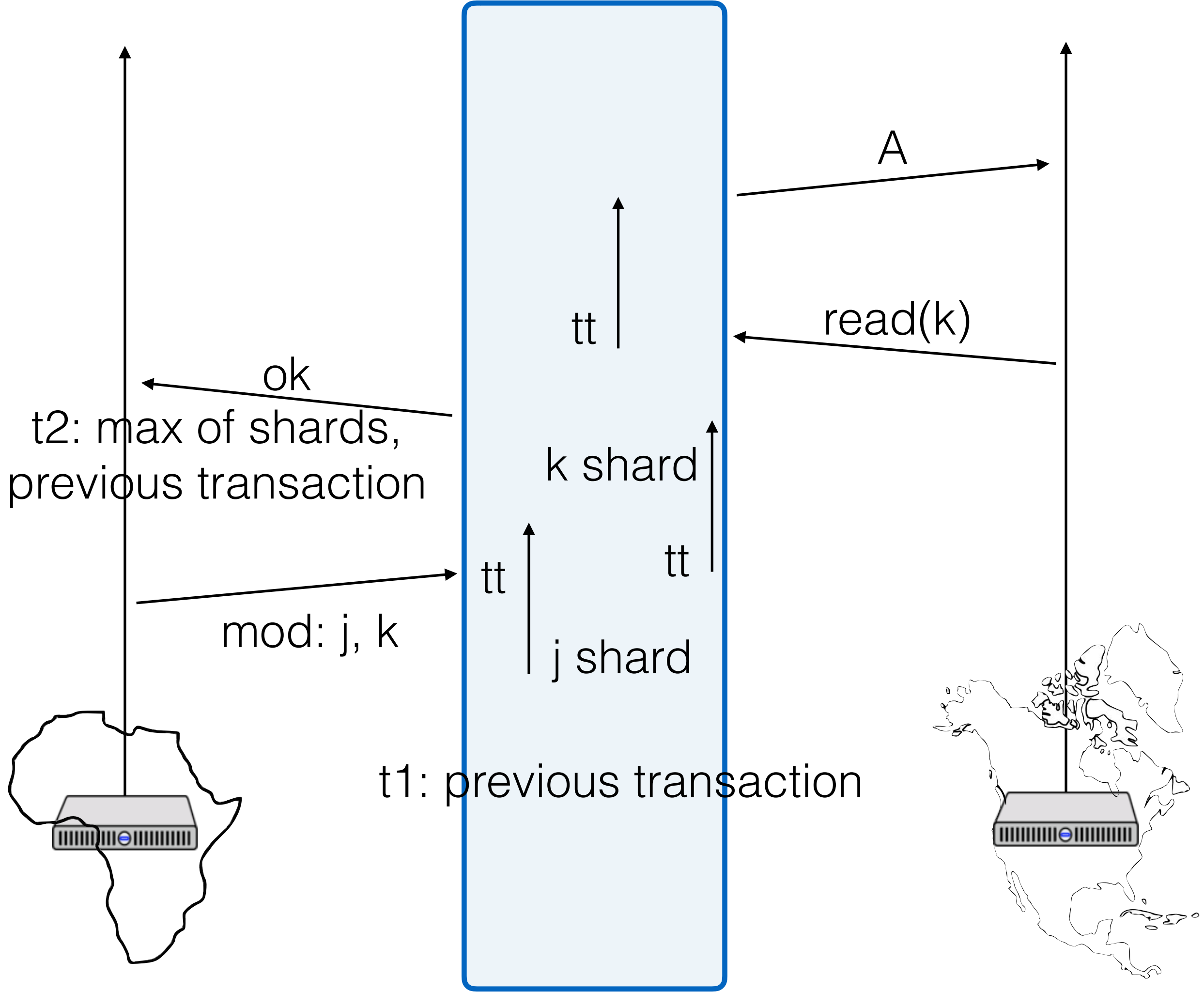
- At each Paxos group, timestamp increases monotonically
- Globally, if T1 returns before T2 starts,
timestamp(T1) < timestamp(T2)

TrueTime usage

Timestamp for an RW transaction chosen by coordinator leader

Timestamps for R/W transactions is max of:

- Local time (when client request reached coord.)
- Prepare timestamps at every participant
- Timestamp of any previous local transaction



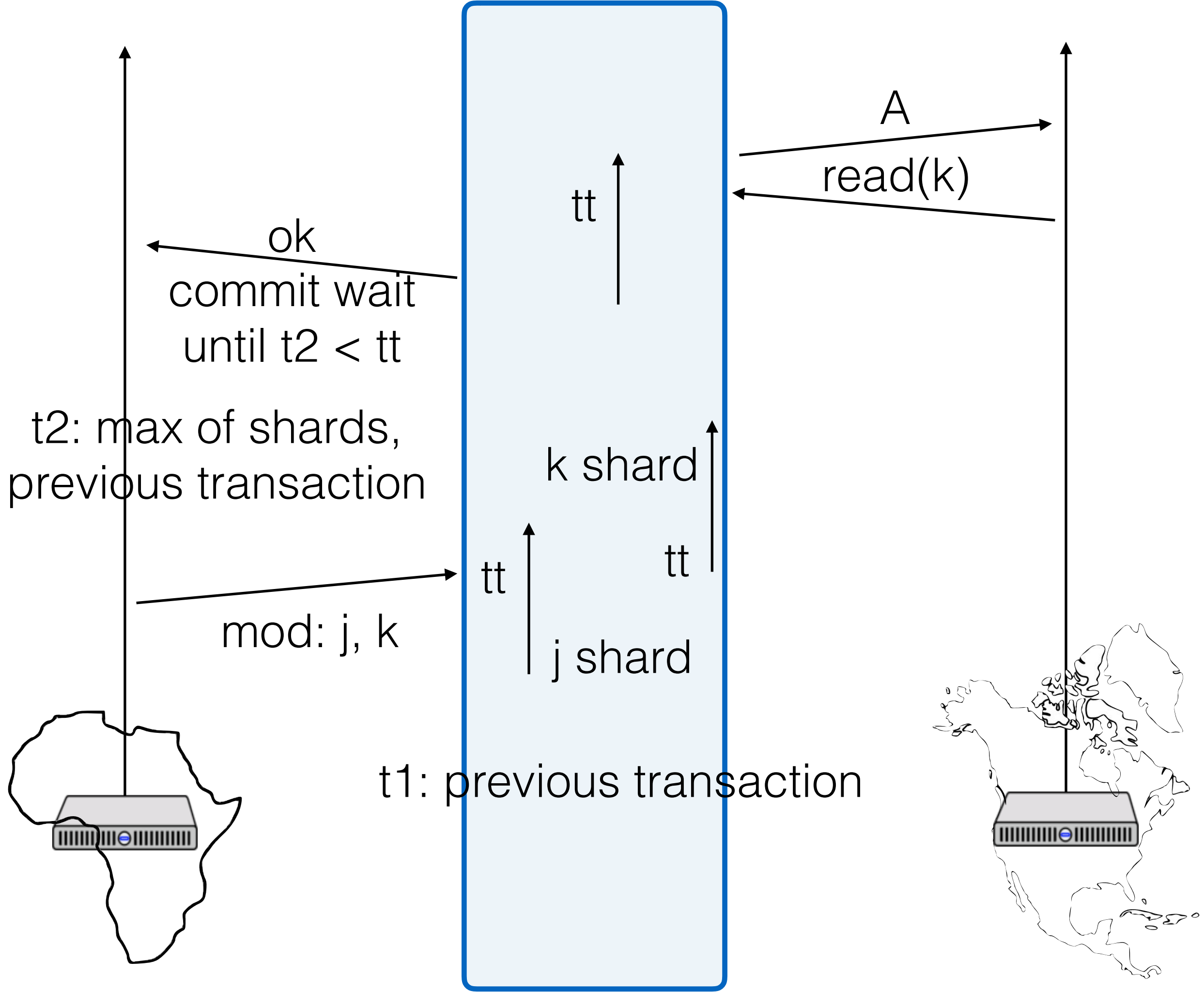
Commit wait

Need to ensure that all future transactions will get a higher timestamp

Therefore, need to wait until

$TT.now() > \text{transaction timestamp}$

And only then release locks



Leader 1



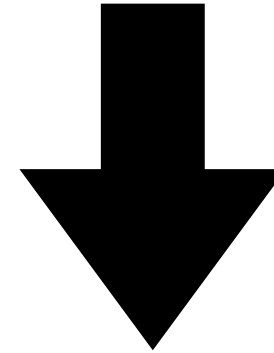
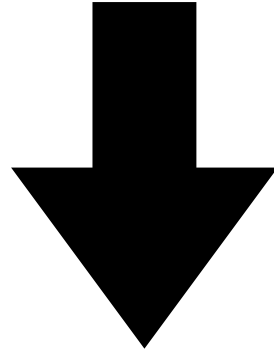
Client



Leader 2



```
x = read(checking_bal)
if (x > 100) {
  y = read(savings_bal)
  write(checking_bal, x - 100)
  write(savings_bal, y + 100)
}
```



LOG:

1. A: Lock(R, checking_bal)
2. A: Lock(W, checking_bal)
3. A: Write(checking_bal, 100)

LOG:

1. A: Lock(R, savings_bal)
2. A: Lock(W, savings_bal)
3. A: Write(savings_bal, 100)

Leader 1



commit(1)



Client



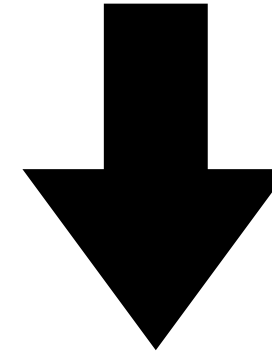
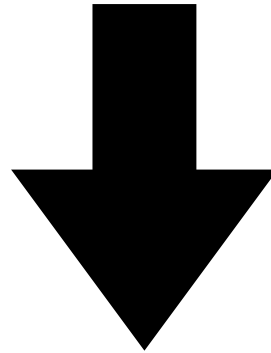
commit(1)



Leader 2



```
x = read(checking_bal)
if (x > 100) {
  y = read(savings_bal)
  write(checking_bal, x - 100)
  write(savings_bal, y + 100)
}
```



LOG:

1. A: Lock(R, checking_bal)
2. A: Lock(W, checking_bal)
3. A: Write(checking_bal, 100)

LOG:

1. A: Lock(R, savings_bal)
2. A: Lock(W, savings_bal)
3. A: Write(savings_bal, 100)

Leader 1



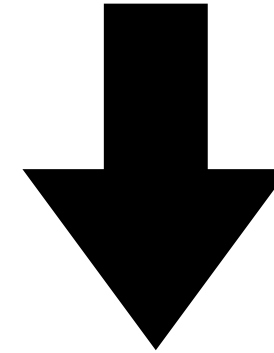
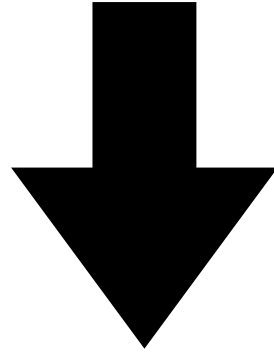
Client



Leader 2



```
x = read(checking_bal)
if (x > 100) {
  y = read(savings_bal)
  write(checking_bal, x - 100)
  write(savings_bal, y + 100)
}
```



LOG:

1. A: Lock(R, checking_bal)
2. A: Lock(W, checking_bal)
3. A: Write(checking_bal, 100)

LOG:

1. A: Lock(R, savings_bal)
2. A: Lock(W, savings_bal)
3. A: Write(savings_bal, 100)
4. A: Prepare@t1

Leader 1



Client

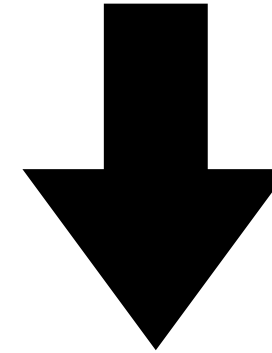
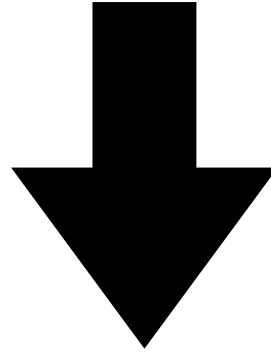


Leader 2



OK@t1

```
x = read(checking_bal)
if (x > 100) {
  y = read(savings_bal)
  write(checking_bal, x - 100)
  write(savings_bal, y + 100)
}
```



LOG:

1. A: Lock(R, checking_bal)
2. A: Lock(W, checking_bal)
3. A: Write(checking_bal, 100)

LOG:

1. A: Lock(R, savings_bal)
2. A: Lock(W, savings_bal)
3. A: Write(savings_bal, 100)
4. A: Prepare@t1

Leader 1



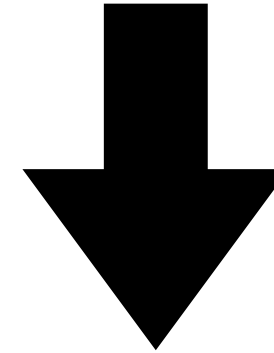
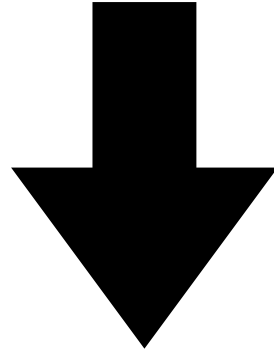
Client



Leader 2



```
x = read(checking_bal)
if (x > 100) {
  y = read(savings_bal)
  write(checking_bal, x - 100)
  write(savings_bal, y + 100)
}
```



LOG:

1. A: Lock(R, checking_bal)
2. A: Lock(W, checking_bal)
3. A: Write(checking_bal, 100)
4. A: Commit@max(t1, tleader)

LOG:

1. A: Lock(R, savings_bal)
2. A: Lock(W, savings_bal)
3. A: Write(savings_bal, 100)
4. A: Prepare@t1

Leader 1



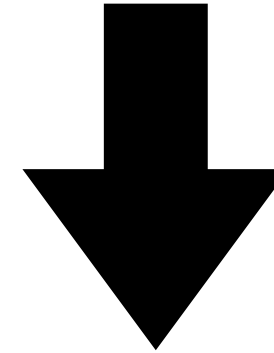
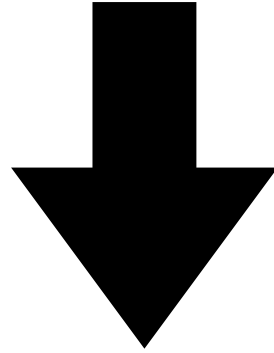
Client



Leader 2



```
x = read(checking_bal)
if (x > 100) {
  y = read(savings_bal)
  write(checking_bal, x - 100)
  write(savings_bal, y + 100)
}
```



LOG:

1. A: Lock(R, checking_bal)
2. A: Lock(W, checking_bal)
3. A: Write(checking_bal, 100)
4. A: Commit@max(t1, tleader)

LOG:

1. A: Lock(R, savings_bal)
2. A: Lock(W, savings_bal)
3. A: Write(savings_bal, 100)
4. A: Prepare@t1

Waiting...

Leader 1



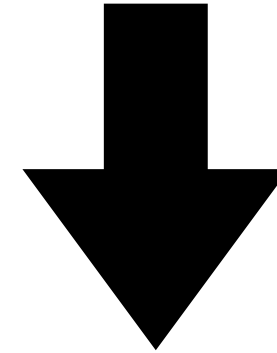
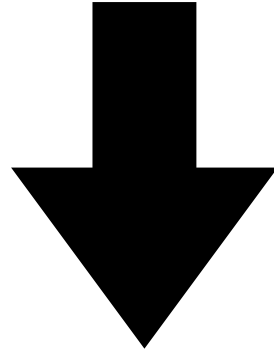
Client



Leader 2



```
x = read(checking_bal)
if (x > 100) {
  y = read(savings_bal)
  write(checking_bal, x - 100)
  write(savings_bal, y + 100)
}
```



LOG:

1. A: Lock(R, checking_bal)
2. A: Lock(W, checking_bal)
3. A: Write(checking_bal, 100)
4. A: Commit@max(t1, tleader)
5. A: Unlock(checking_bal)

LOG:

1. A: Lock(R, savings_bal)
2. A: Lock(W, savings_bal)
3. A: Write(savings_bal, 100)
4. A: Prepare@t1

Leader 1



OK



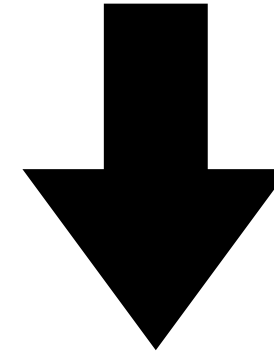
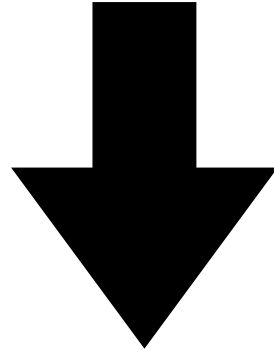
Client



Leader 2



```
x = read(checking_bal)
if (x > 100) {
  y = read(savings_bal)
  write(checking_bal, x - 100)
  write(savings_bal, y + 100)
}
```



LOG:

1. A: Lock(R, checking_bal)
2. A: Lock(W, checking_bal)
3. A: Write(checking_bal, 100)
4. A: Commit@max(t1, tleader)
5. A: Unlock(checking_bal)

LOG:

1. A: Lock(R, savings_bal)
2. A: Lock(W, savings_bal)
3. A: Write(savings_bal, 100)
4. A: Prepare@t1

Leader 1



commit@T

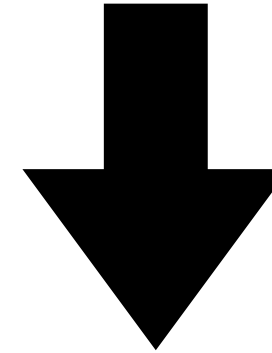
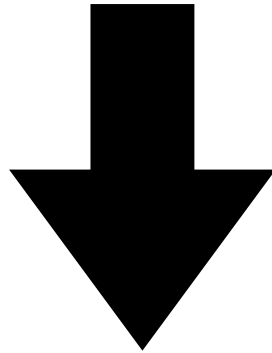
Client



Leader 2



```
x = read(checking_bal)
if (x > 100) {
  y = read(savings_bal)
  write(checking_bal, x - 100)
  write(savings_bal, y + 100)
}
```



LOG:

1. A: Lock(R, checking_bal)
2. A: Lock(W, checking_bal)
3. A: Write(checking_bal, 100)
4. A: Commit@max(t1, tleader)
5. A: Unlock(checking_bal)

LOG:

1. A: Lock(R, savings_bal)
2. A: Lock(W, savings_bal)
3. A: Write(savings_bal, 100)
4. A: Prepare@t1

Leader 1



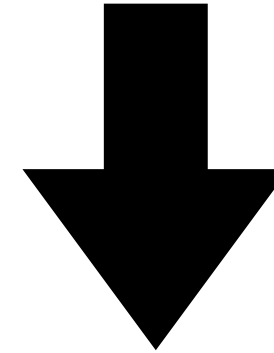
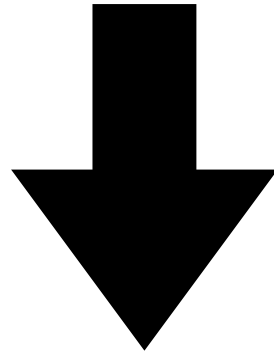
Client



Leader 2



```
x = read(checking_bal)
if (x > 100) {
  y = read(savings_bal)
  write(checking_bal, x - 100)
  write(savings_bal, y + 100)
}
```



LOG:

1. A: Lock(R, checking_bal)
2. A: Lock(W, checking_bal)
3. A: Write(checking_bal, 100)
4. A: Commit@max(t1, tleader)
5. A: Unlock(checking_bal)

LOG:

1. A: Lock(R, savings_bal)
2. A: Lock(W, savings_bal)
3. A: Write(savings_bal, 100)
4. A: Prepare@t1
5. A: Commit@max(t1, tleader)
6. A: Unlock(savings_bal)

Commit wait

- What does this mean for performance?
- Larger TrueTime uncertainty bound
=> longer commit wait
- Longer commit wait => locks held longer
=> can't process conflicting transactions
=> lower throughput
- i.e., if time is less certain, Spanner is slower!

What does this buy us?

- Can now do a read-only transaction at a particular timestamp, have it be meaningful
- Example: pick a timestamp T in the past, read version w/ timestamp T from all shards
 - since T is in the past, they will never accept a transaction with timestamp $< T$
 - don't need locks while we do this!
- What if we want the current time?

TrueTime implementation

GPS, atomic clocks

All local clocks synced with masters, and expose uncertainty to local apps

Assumptions made about local clock drift

What if TrueTime fails?

- Google argument: picked using engineering considerations, less likely than a total CPU failure
- But what if it went wrong anyway?
 - can cause very long commit wait periods
 - can break ordering guarantees, no longer externally consistent
 - but system will always be serializable: gathering many timestamps and taking the max is a Lamport clock

Conclusions

What's cool about Spanner?

- Distributed transactions with decent performance
- What makes that possible?
- Read-only transactions with great performance
- What makes that possible?

Clocks are a form of communication!