Spanner

2PC Problems

- Availability: what do we do if either some shard or the coordinator fails?

  - 2PC is a blocking protocol, can't make progress until failed node comes back up

- Performance:

  - read-only transactions are slow

  - contention for frequently used locks

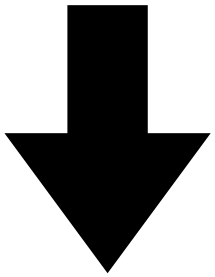
Multi-Key Transactions

- All keys on the same shard
- Keys spread across multiple shards

Leader

Client

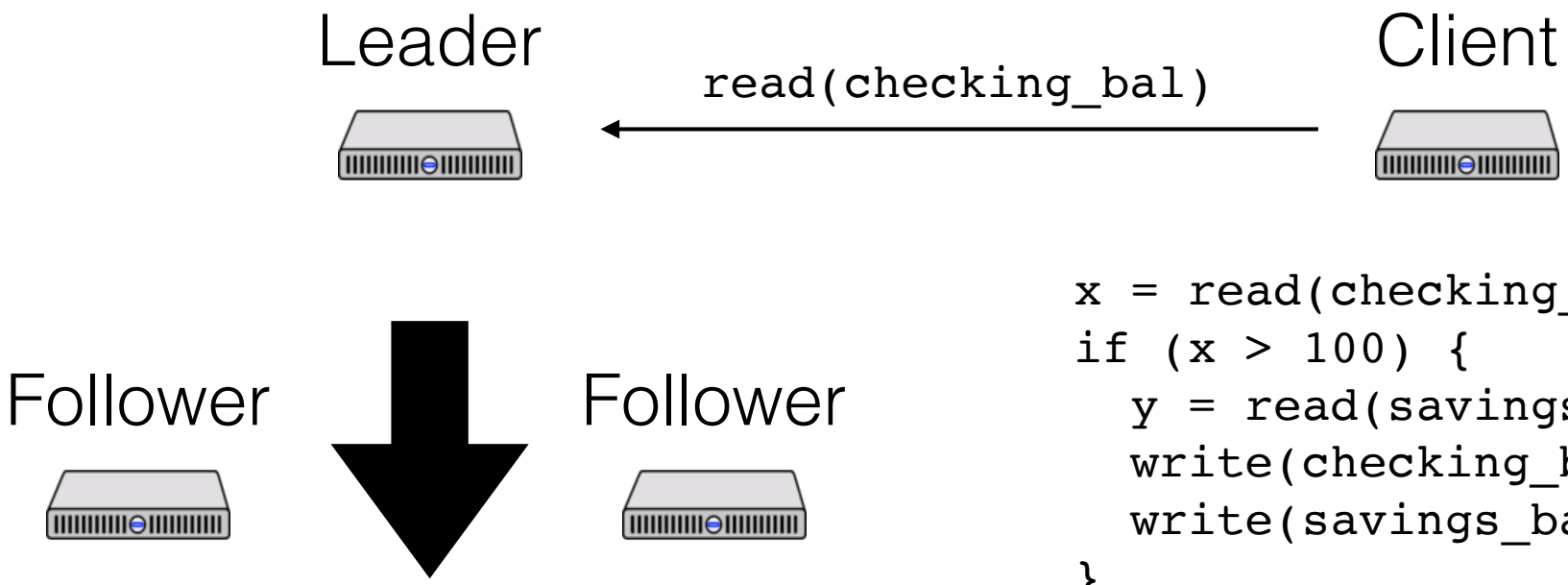
Follower



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.
```



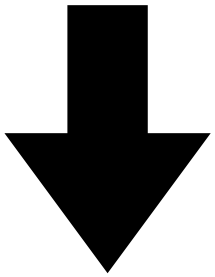

```
x = read(checking_bal)
if (x > 100) {
  y = read(savings_bal)
  write(checking_bal, x - 100)
  write(savings_bal, y + 100)
}
```

```
LOG:
 1.
 2.
 3.
 4.
 5.
 6.
 7.
```

Leader

Client

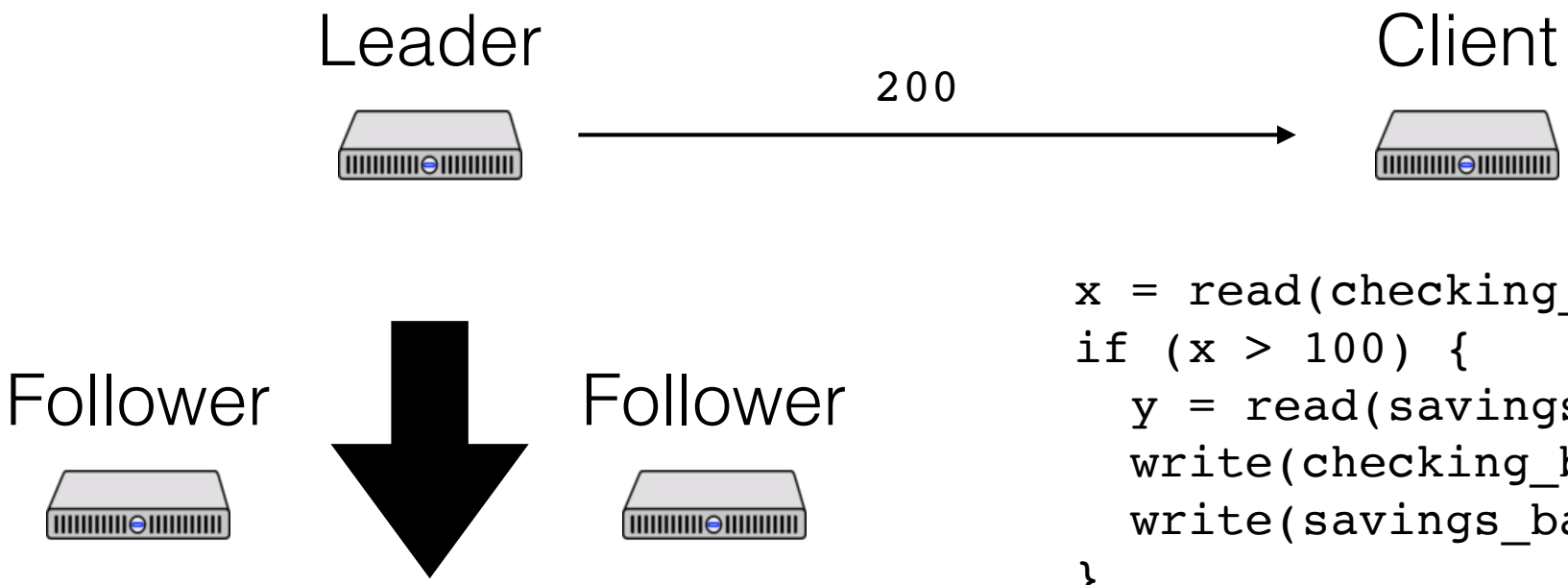
Follower



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.
```



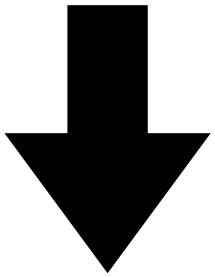

```
x = read(checking_bal)
if (x > 100) {
  y = read(savings_bal)
  write(checking_bal, x - 100)
  write(savings_bal, y + 100)
}
```

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

Leader

Client

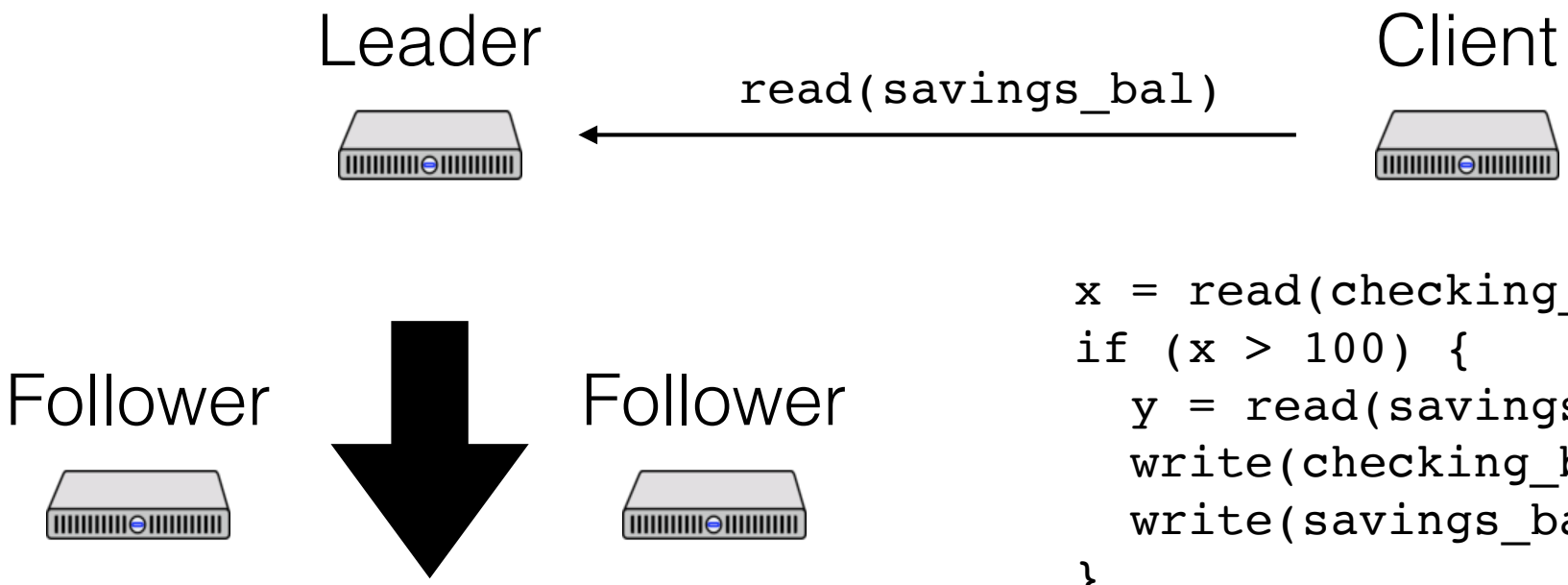
Follower



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.
```



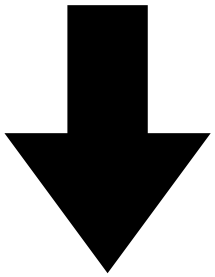

```
x = read(checking_bal)
if (x > 100) {
  y = read(savings_bal)
  write(checking_bal, x - 100)
  write(savings_bal, y + 100)
}
```

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

Leader

Client

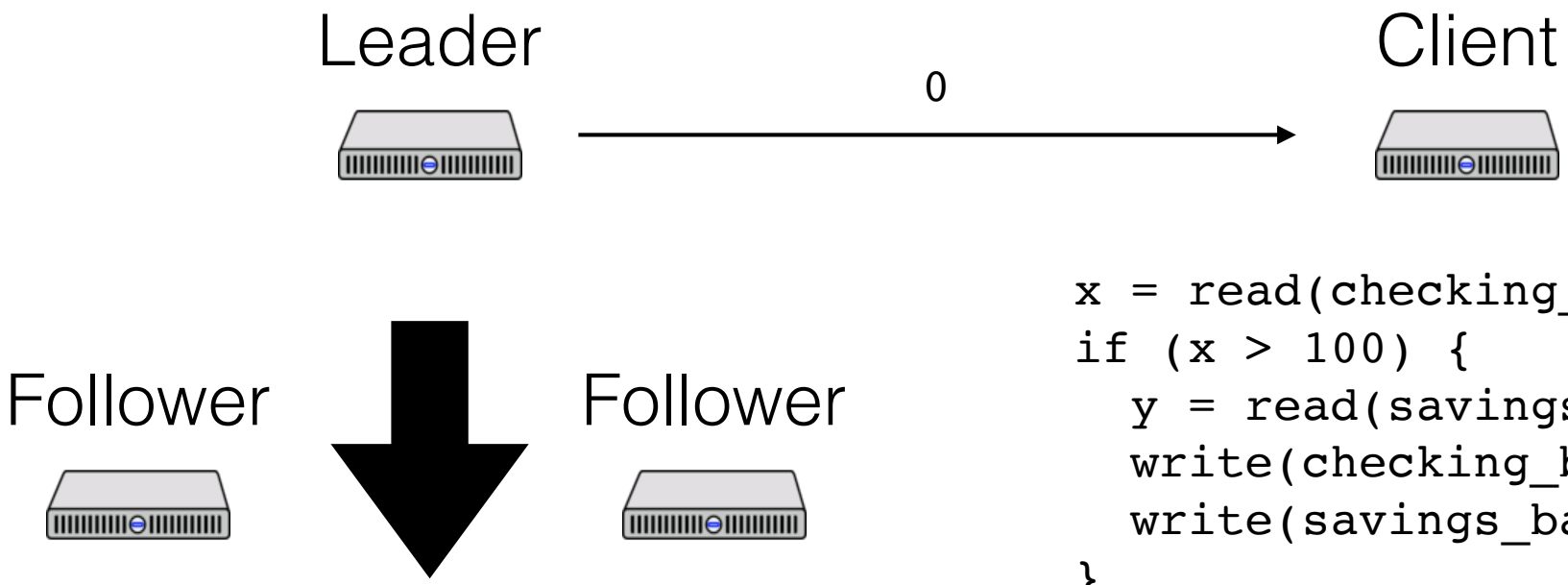
Follower



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.
```



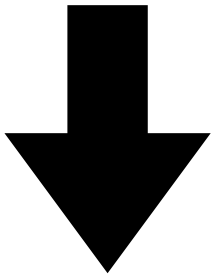

```
x = read(checking_bal)
if (x > 100) {
  y = read(savings_bal)
  write(checking_bal, x - 100)
  write(savings_bal, y + 100)
}
```

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

Leader

Client

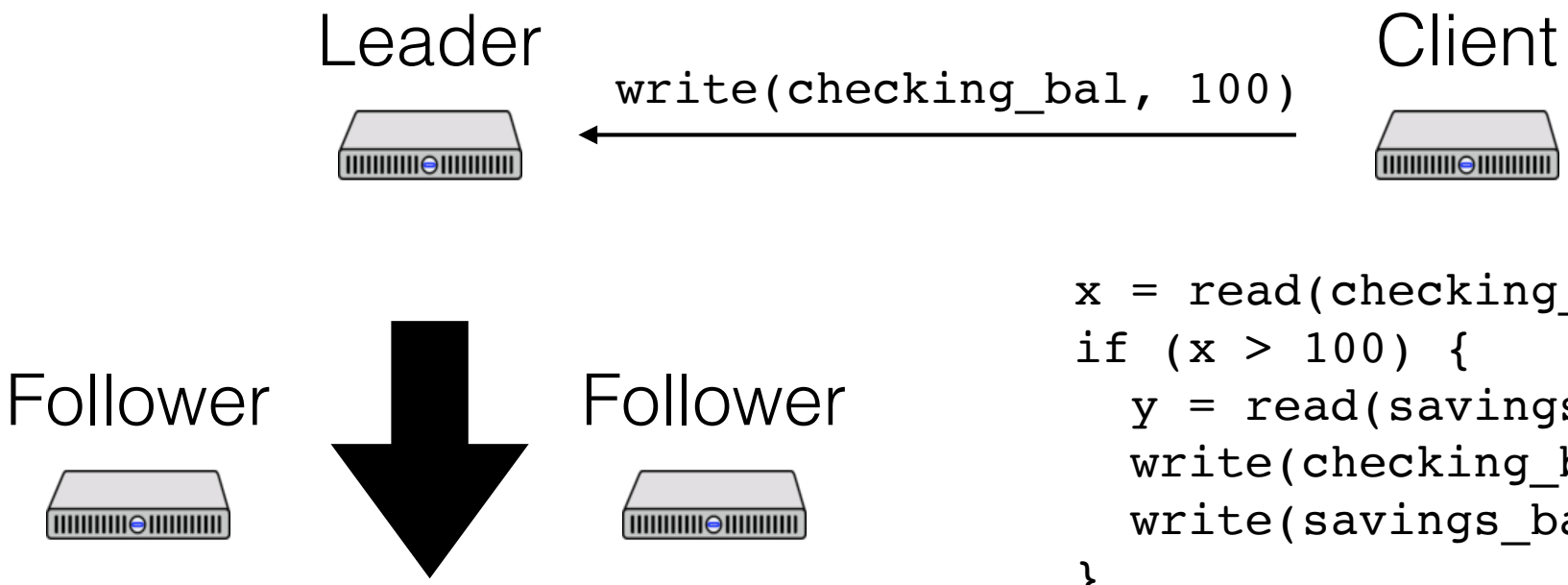
Follower



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.
```



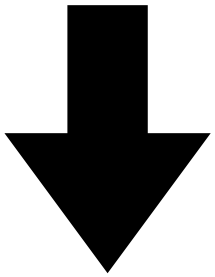

```
x = read(checking_bal)
if (x > 100) {
  y = read(savings_bal)
  write(checking_bal, x - 100)
  write(savings_bal, y + 100)
}
```

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

Leader

Client

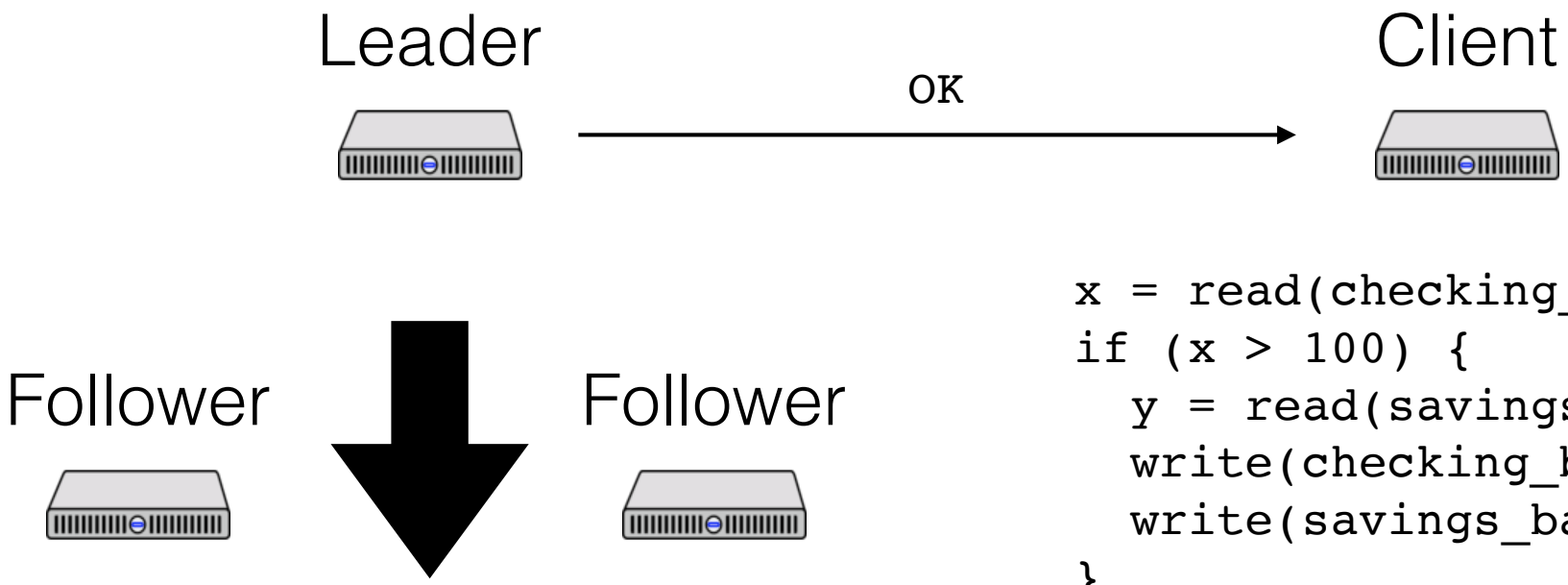
Follower



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.
```



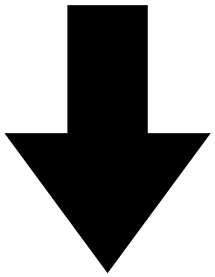

```
x = read(checking_bal)
if (x > 100) {
  y = read(savings_bal)
  write(checking_bal, x - 100)
  write(savings_bal, y + 100)
}
```

```
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

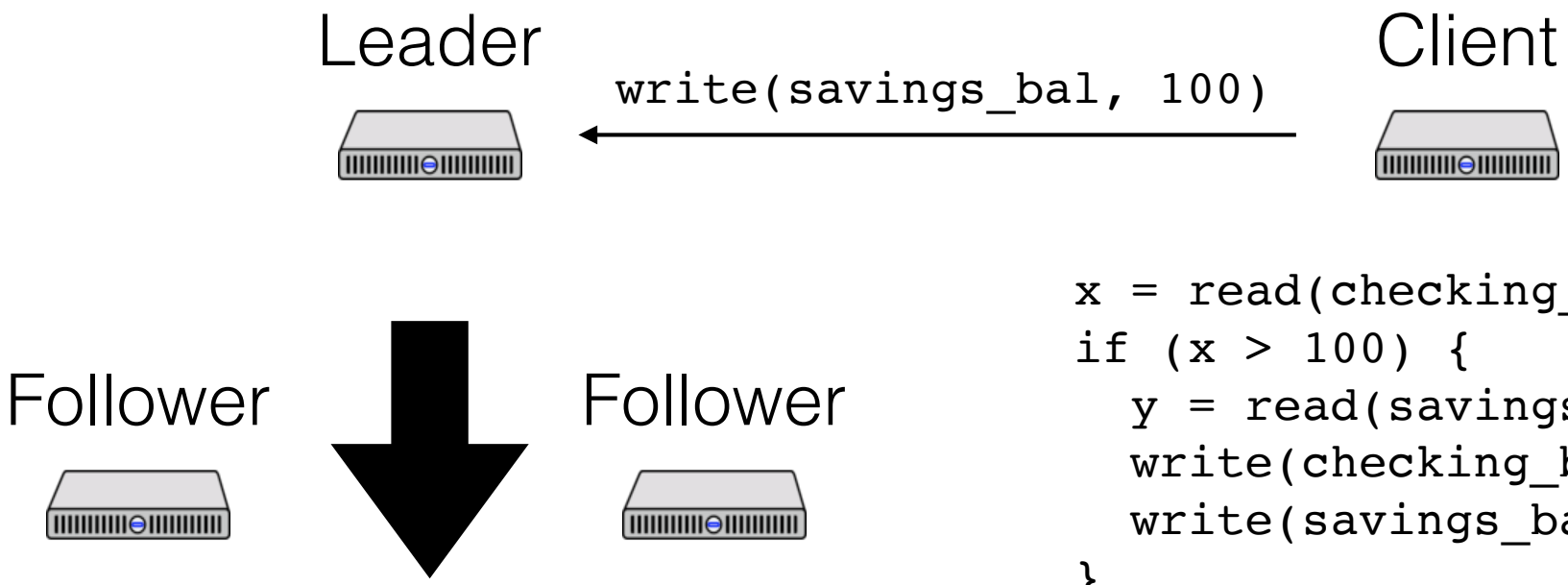
Follower



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.
```



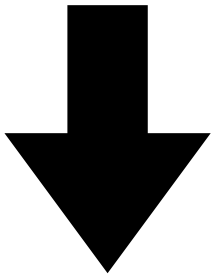

```
x = read(checking_bal)
if (x > 100) {
  y = read(savings_bal)
  write(checking_bal, x - 100)
  write(savings_bal, y + 100)
}
```

```
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

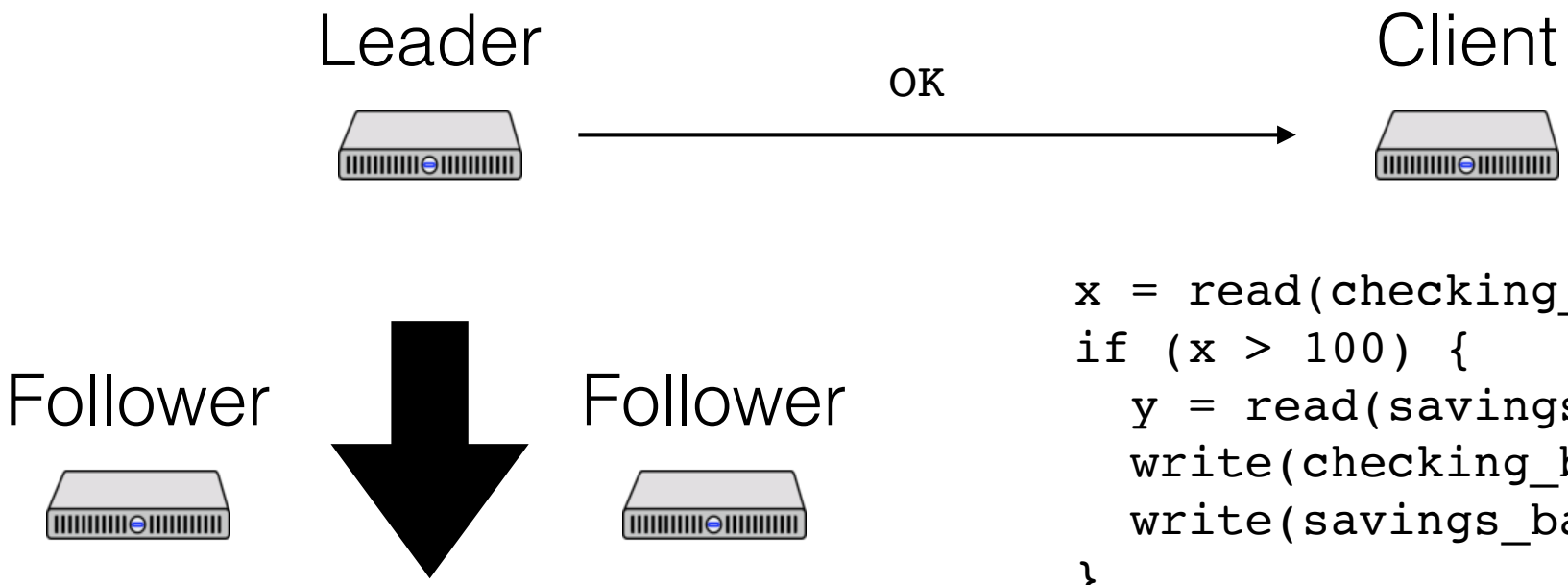
Follower



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.
```



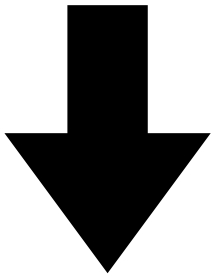

```
x = read(checking_bal)
if (x > 100) {
  y = read(savings_bal)
  write(checking_bal, x - 100)
  write(savings_bal, y + 100)
}
```

```
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

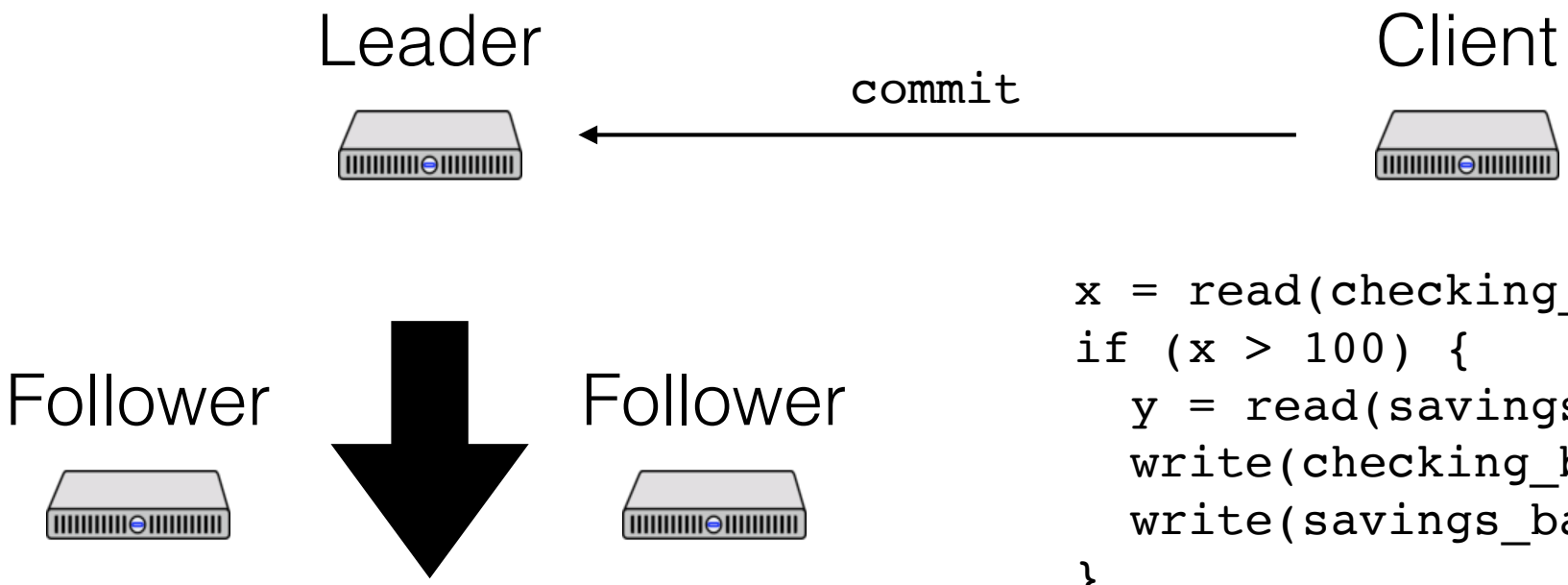
Follower



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.
```



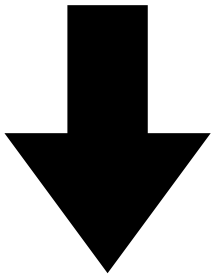

```
x = read(checking_bal)
if (x > 100) {
  y = read(savings_bal)
  write(checking_bal, x - 100)
  write(savings_bal, y + 100)
}
```

```
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

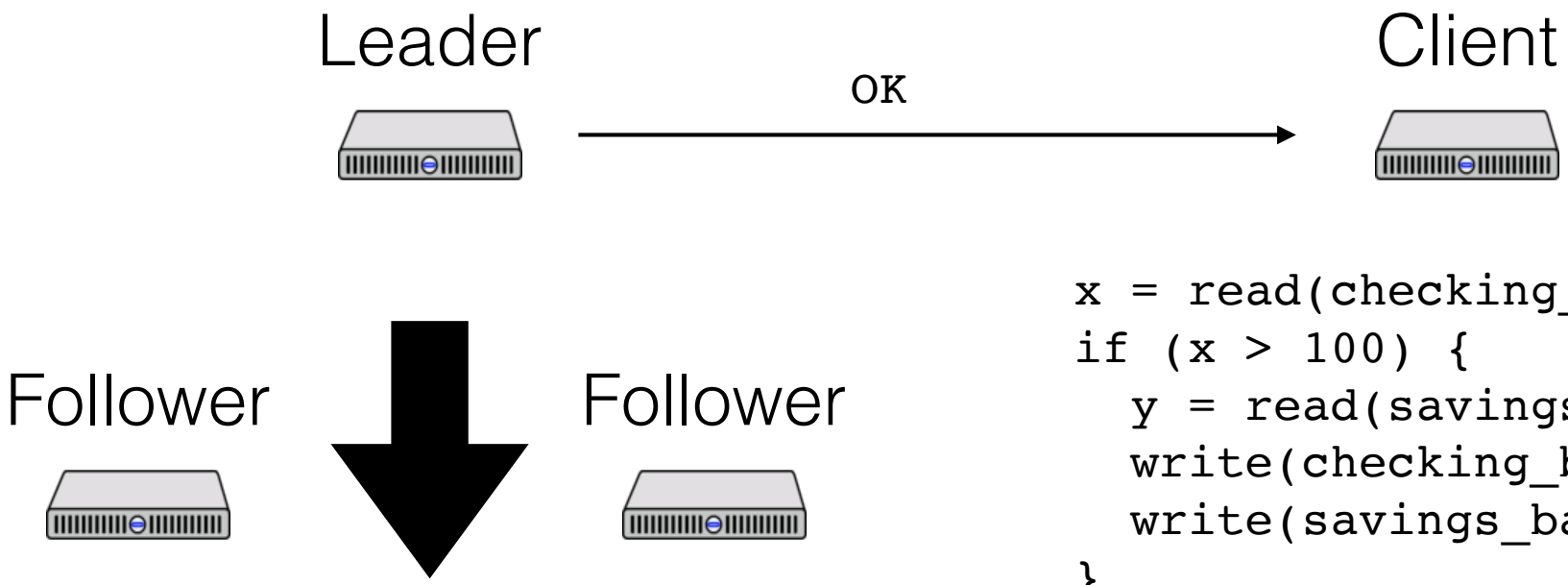
Follower



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
```



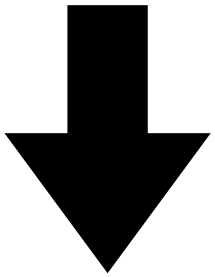

```
x = read(checking_bal)
if (x > 100) {
  y = read(savings_bal)
  write(checking_bal, x - 100)
  write(savings_bal, y + 100)
}
```

```
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

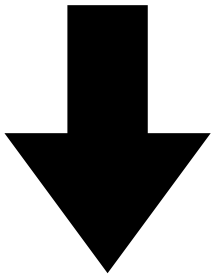
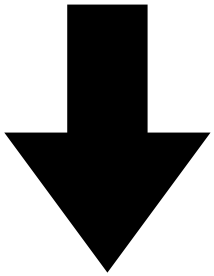
```
x = read(checking_bal)
if (x > 100) {
  y = read(savings_bal)
  write(checking_bal, x - 100)
  write(savings_bal, y + 100)
}
```

```
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

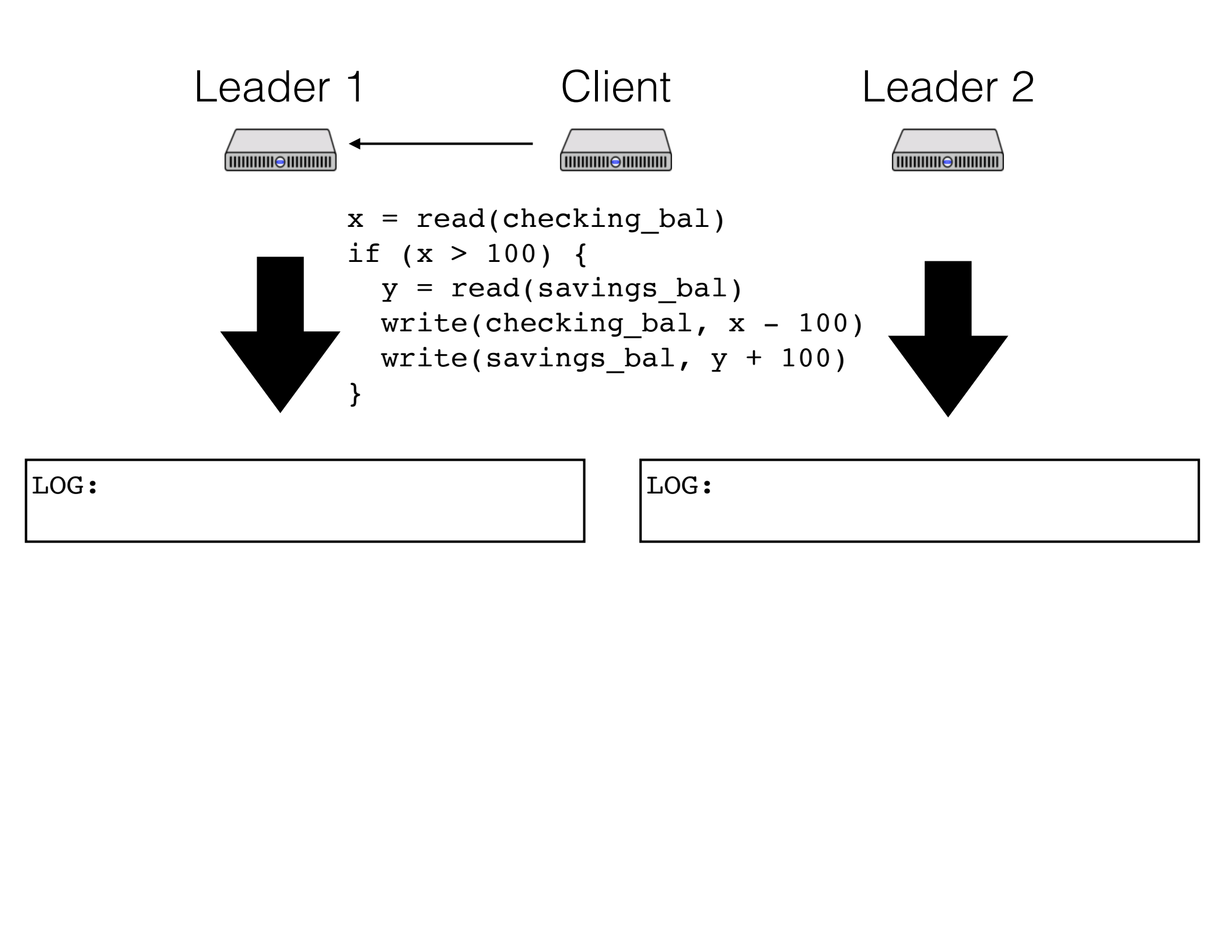


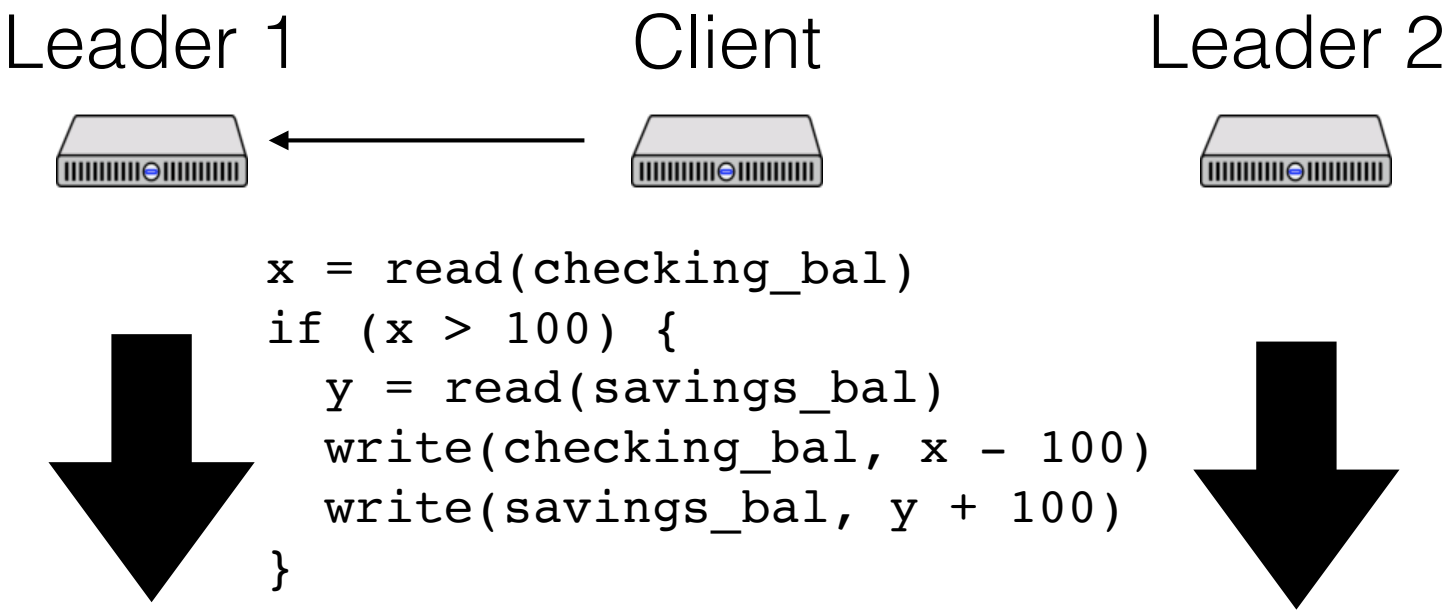
```
x = read(checking_bal)
if (x > 100) {
  y = read(savings_bal)
  write(checking_bal, x - 100)
  write(savings_bal, y + 100)
}
```

LOG:

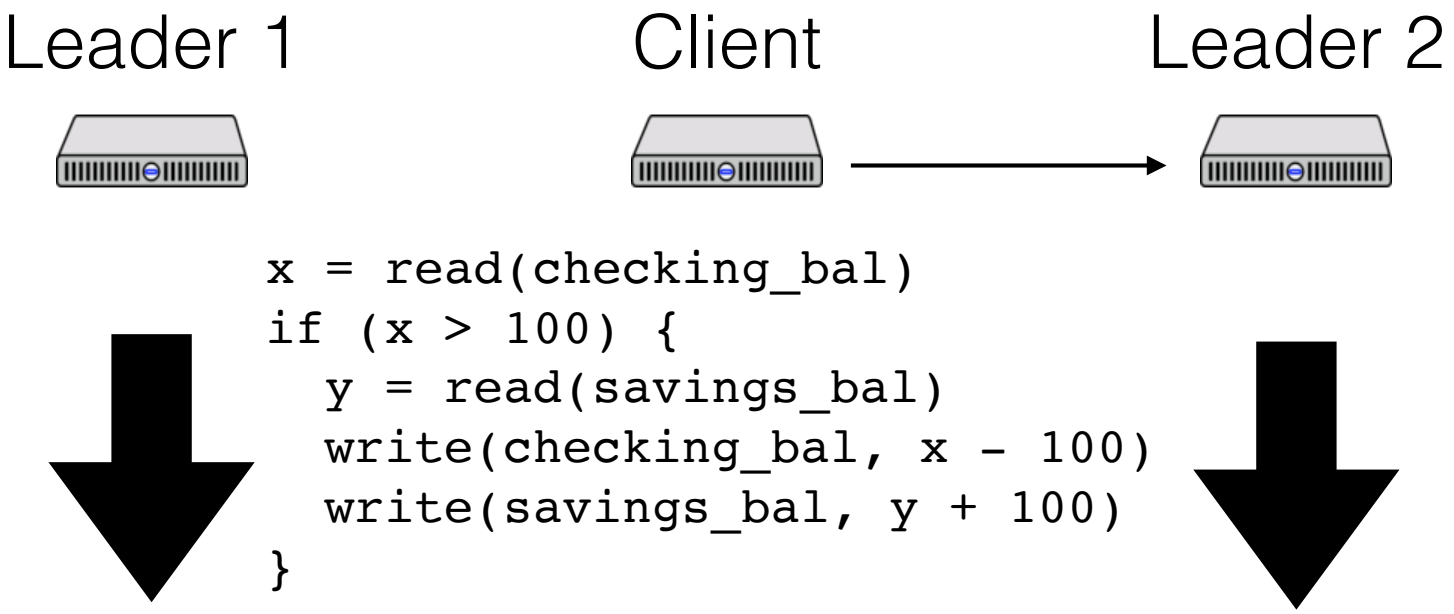
LOG:


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:



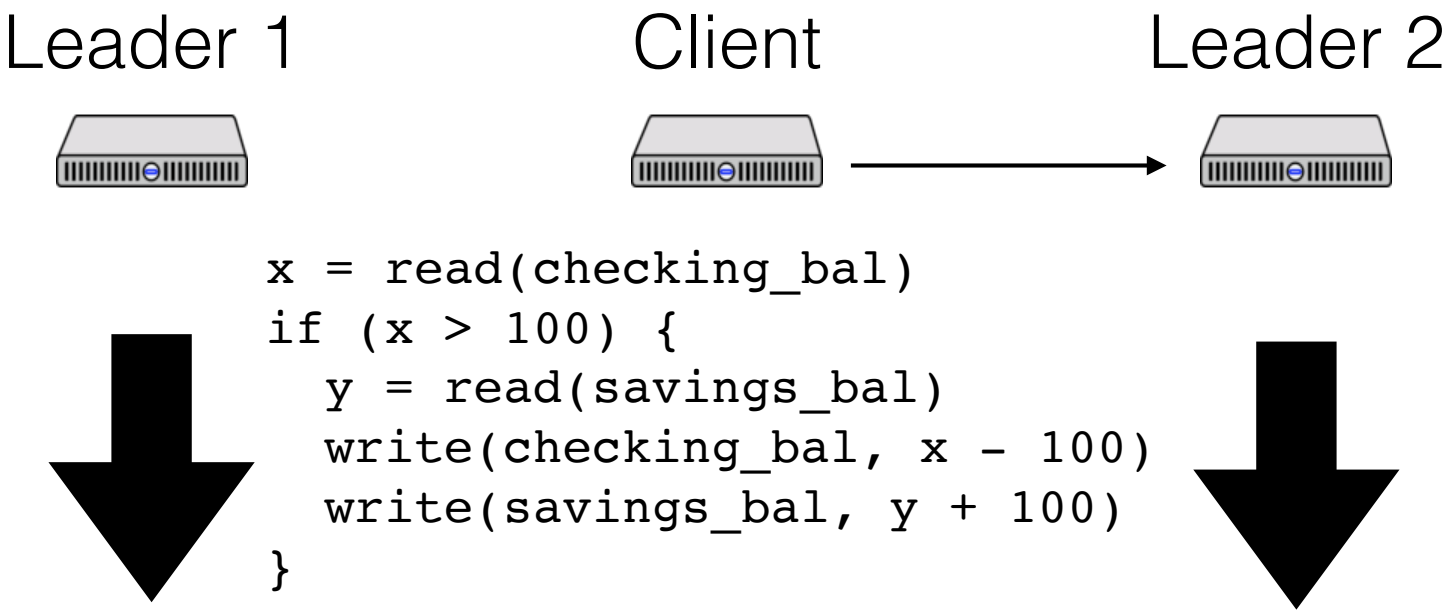

```
LOG:
 1. A: Lock(R, checking_bal)
```

```
LOG:
```



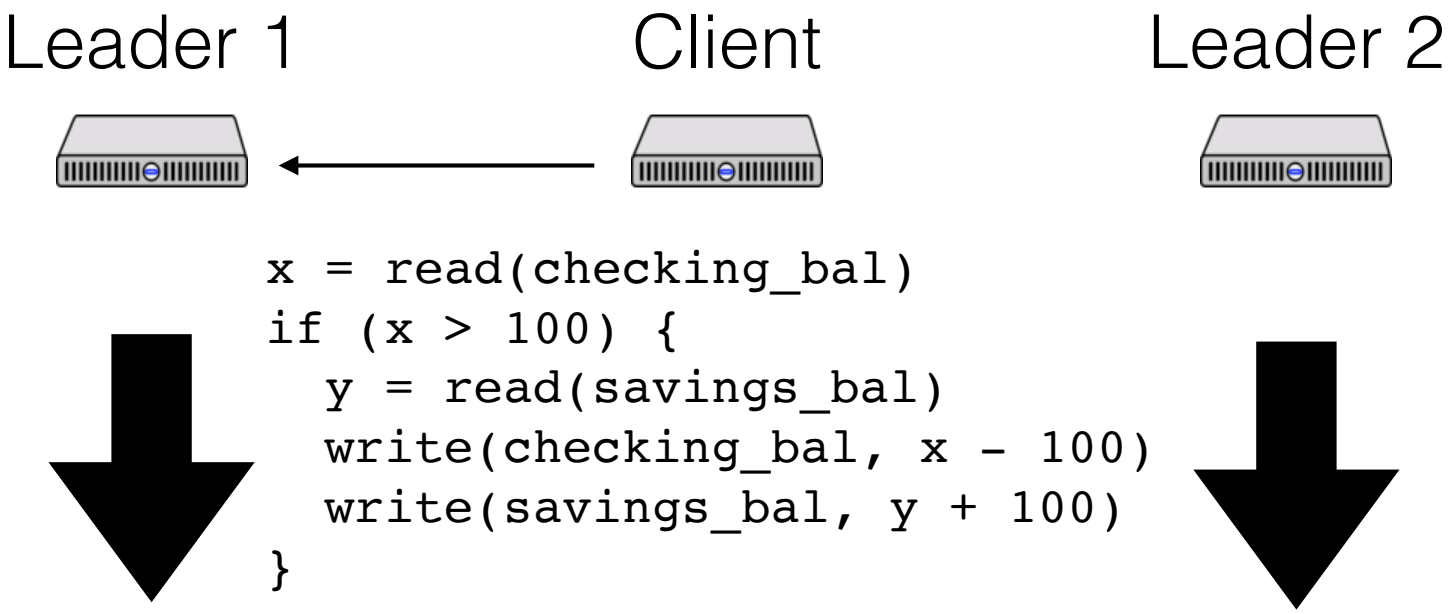

```
LOG:
 1. A: Lock(R, checking_bal)
```

```
LOG:
```



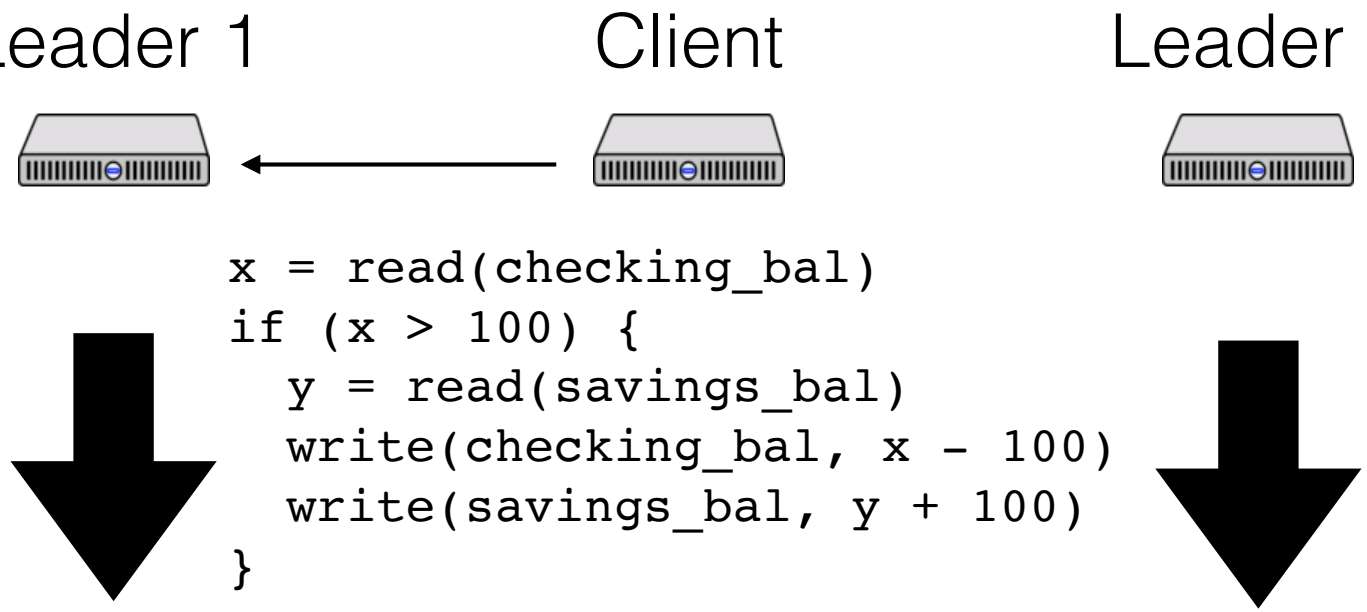

```
LOG:
 1. A: Lock(R, checking_bal)
```

```
LOG:
 1. A: Lock(R, savings_bal)
```



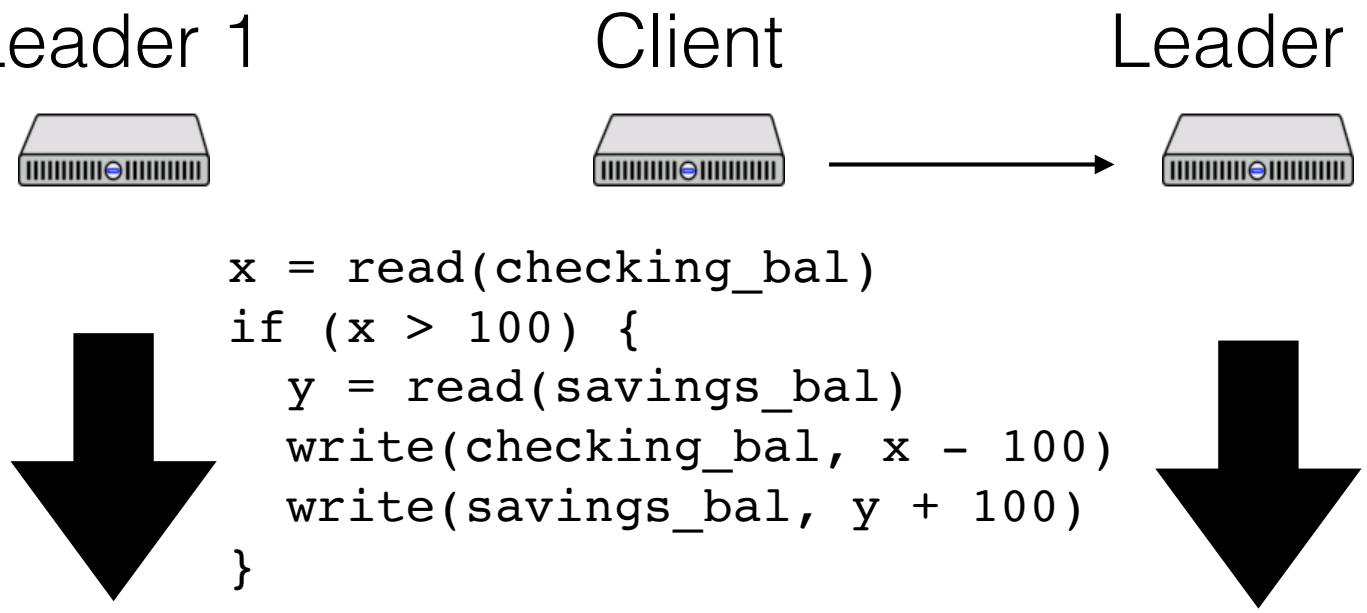

```
LOG:
 1. A: Lock(R, checking_bal)
```

```
LOG:
 1. A: Lock(R, savings_bal)
```



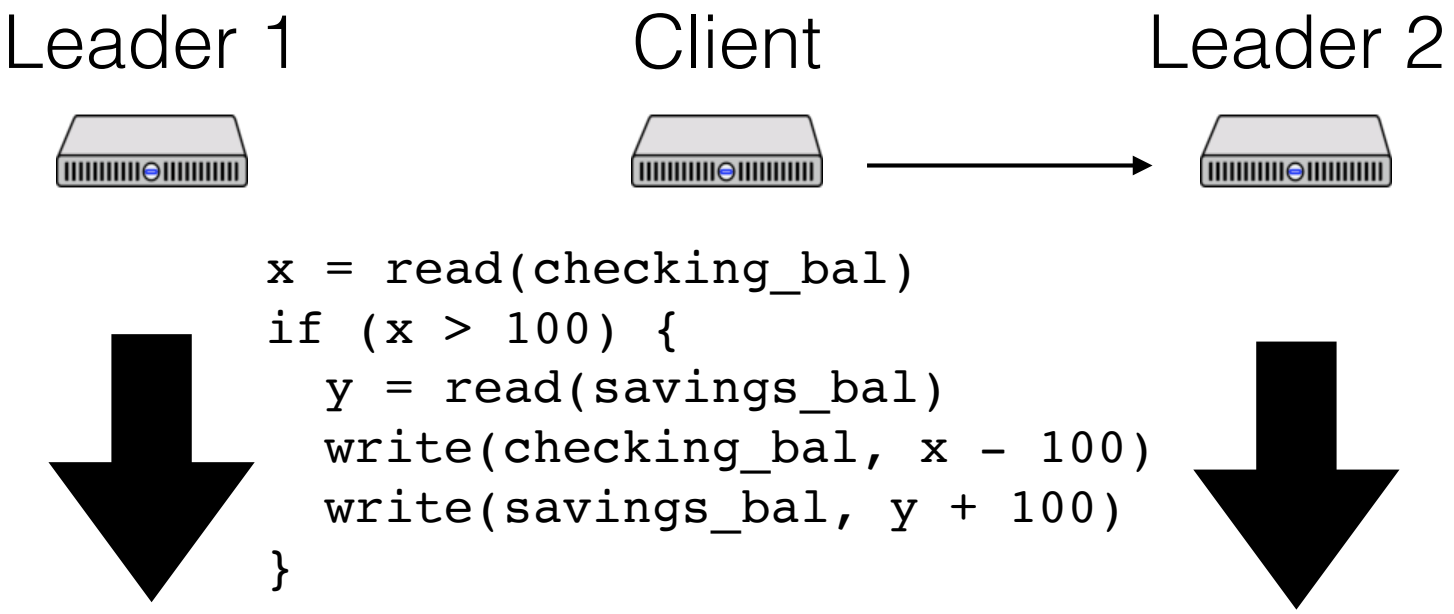

```
LOG:
 1. A: Lock(R, checking_bal)
 2. A: Lock(W, checking_bal)
 3. A: Write(checking_bal, 100)
```

```
LOG:
 1. A: Lock(R, savings_bal)
```




```
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)
```



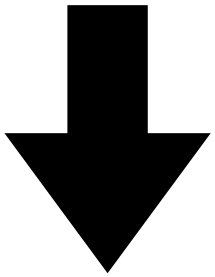
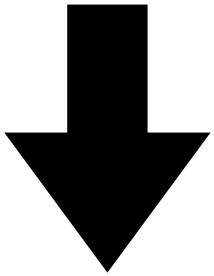

```
LOG:
 1. A: Lock(R, checking_bal)
 2. A: Lock(W, checking_bal)
 3. A: Write(checking_bal, 100)
```

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

Leader 1 Client Leader 2

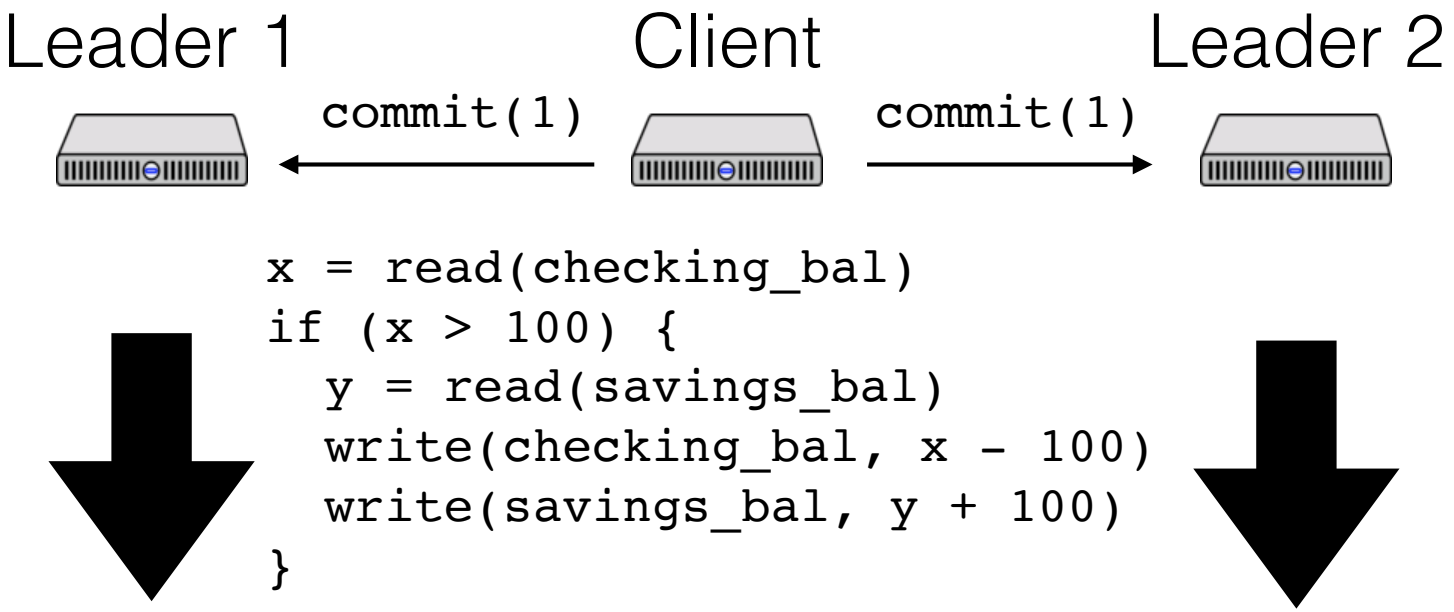


```
x = read(checking_bal)
if (x > 100) {
  y = read(savings_bal)
  write(checking_bal, x - 100)
  write(savings_bal, y + 100)
}
```

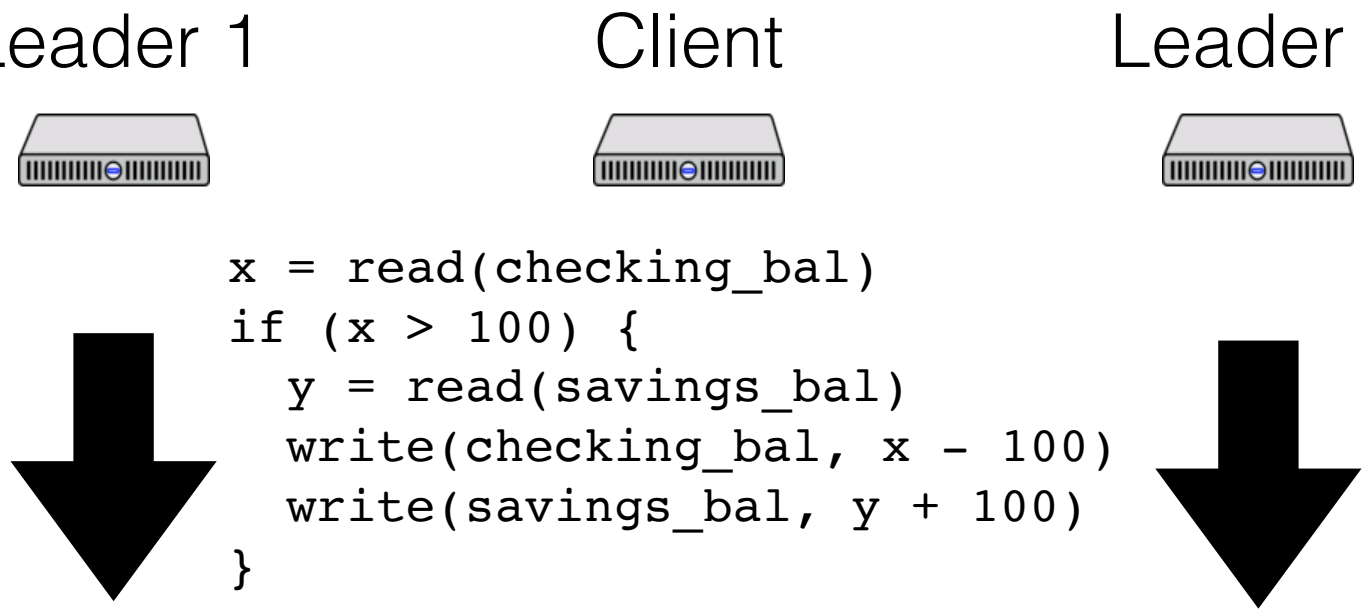
```
LOG:
 1. A: Lock(R, checking_bal)
 2. A: Lock(W, checking_bal)
 3. A: Write(checking_bal, 100)
```

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



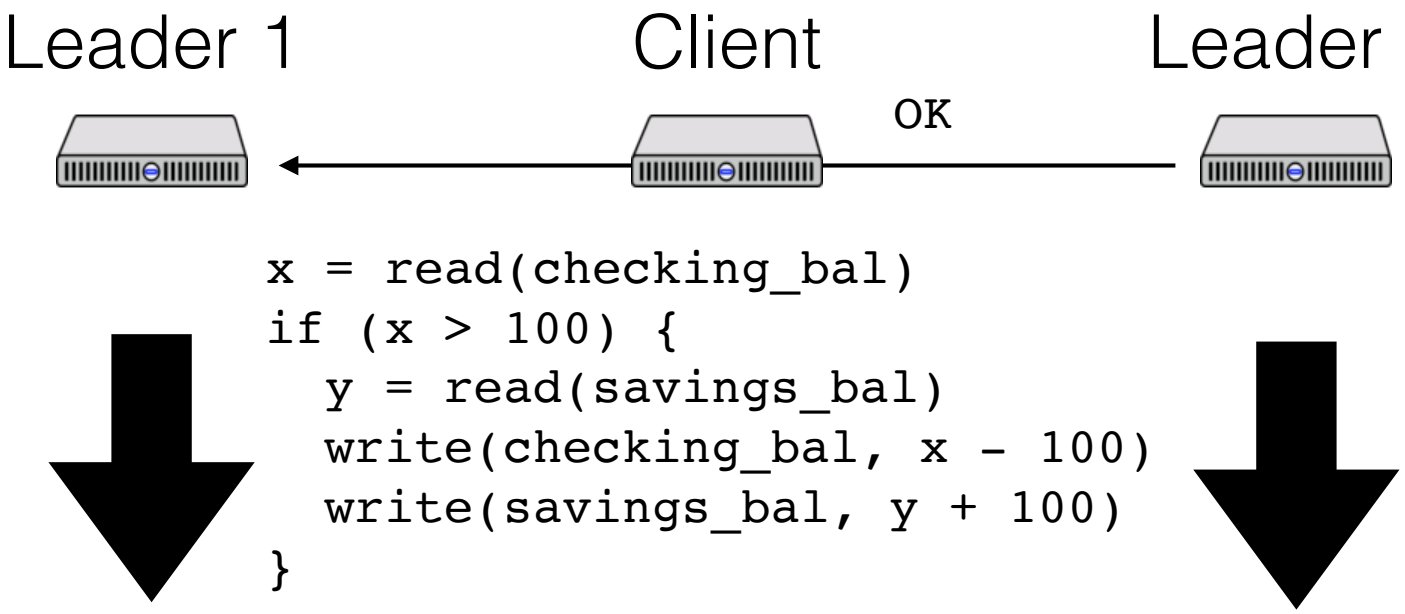

```
LOG:
 1. A: Lock(R, checking_bal)
 2. A: Lock(W, checking_bal)
 3. A: Write(checking_bal, 100)
```

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



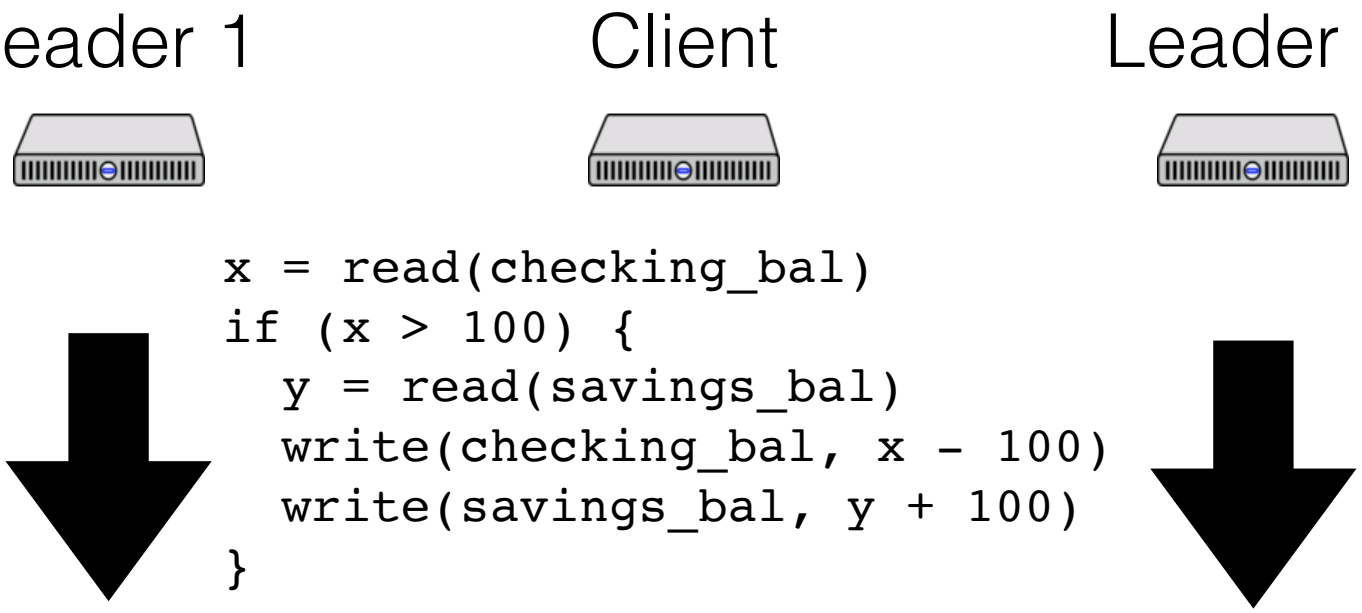

```
LOG:
 1. A: Lock(R, checking_bal)
 2. A: Lock(W, checking_bal)
 3. A: Write(checking_bal, 100)
```

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



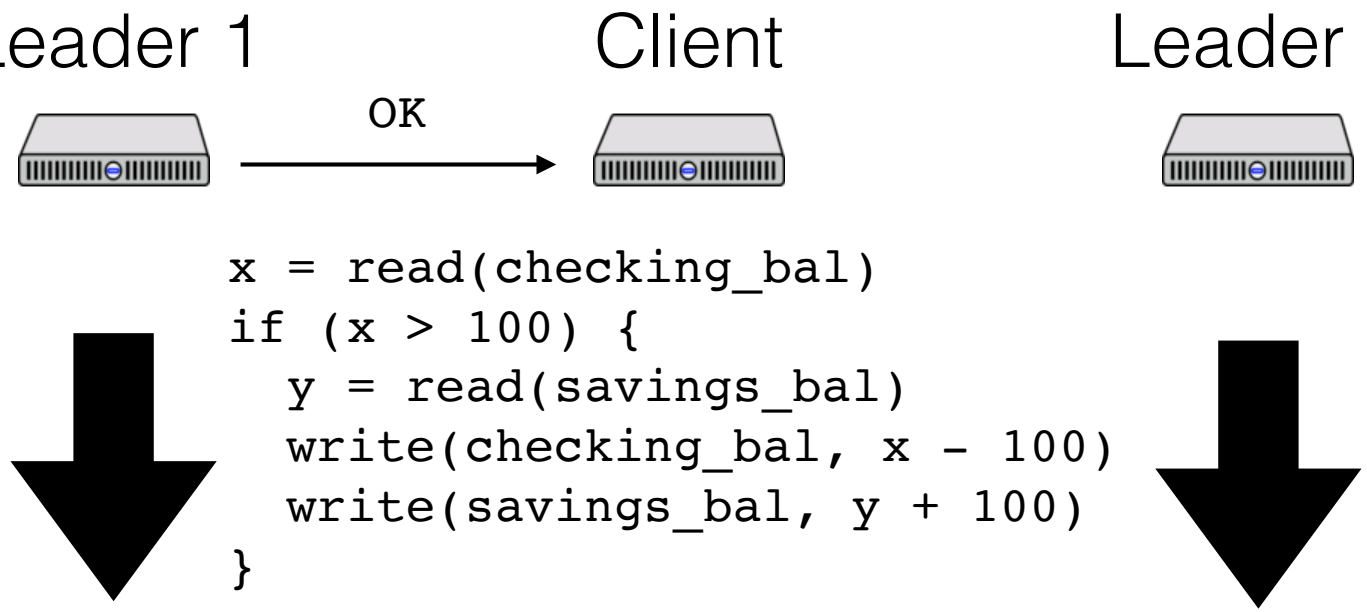

```
LOG:
 1. A: Lock(R, checking_bal)
 2. A: Lock(W, checking_bal)
 3. A: Write(checking_bal, 100)
```

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



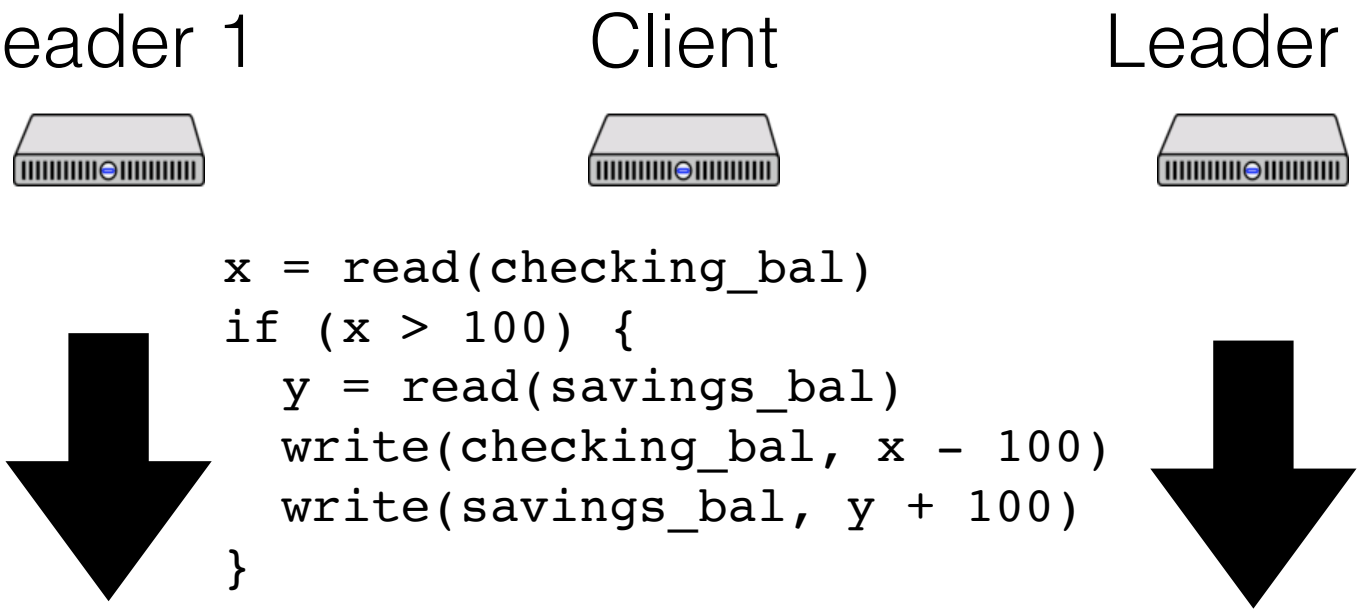

```
LOG:
 1. A: Lock(R, checking_bal)
 2. A: Lock(W, checking_bal)
 3. A: Write(checking_bal, 100)
 4. A: Commit
```

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



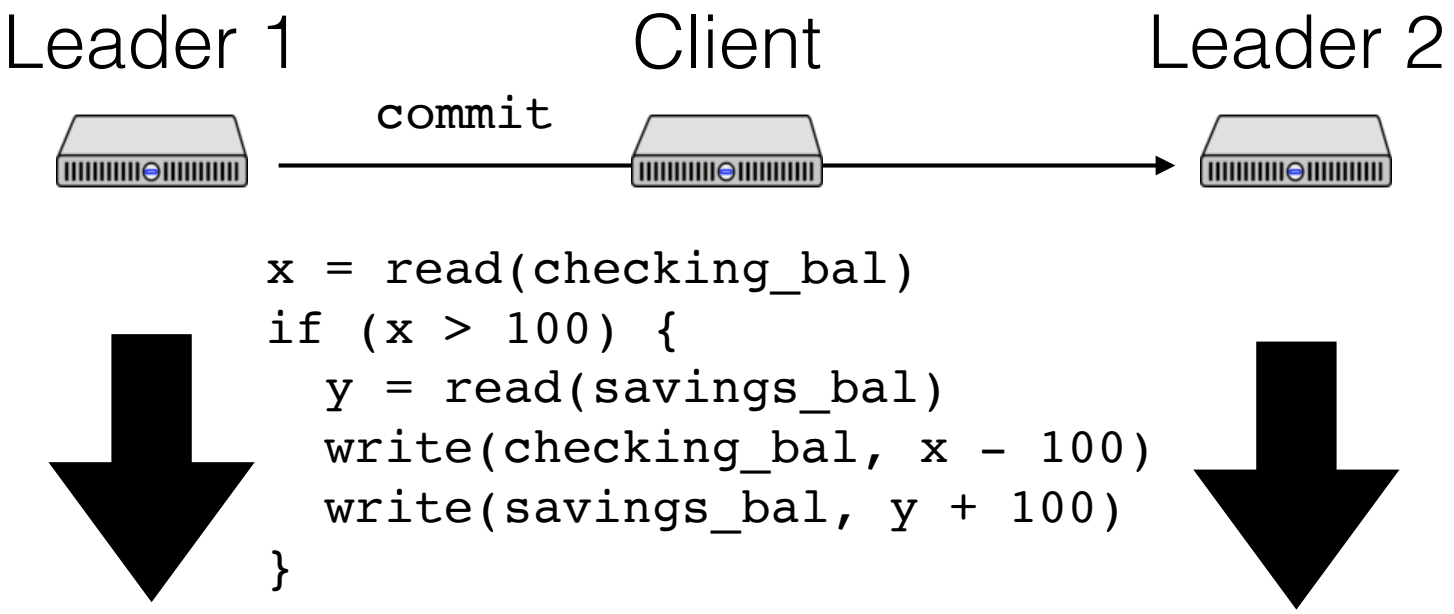

```
LOG:
 1. A: Lock(R, checking_bal)
 2. A: Lock(W, checking_bal)
 3. A: Write(checking_bal, 100)
 4. A: Commit
```

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




```
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
```



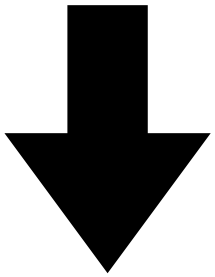
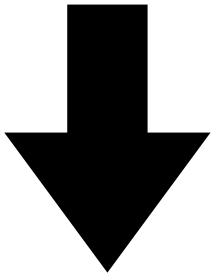

```
LOG:
 1. A: Lock(R, checking_bal)
 2. A: Lock(W, checking_bal)
 3. A: Write(checking_bal, 100)
 4. A: Commit
 5. A: Unlock(checking_bal)
```

```
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
```

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

Bigtable in retrospect

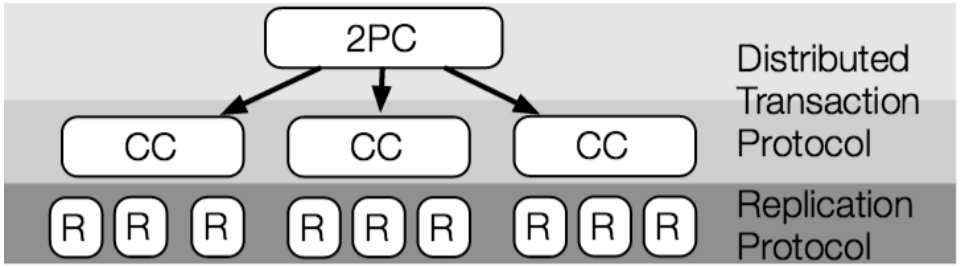
Biggest mistake in design (per Jeff Dean, Google): not supporting distributed transactions!

- became really important w/ incremental updates
- users wanted them, implemented themselves, often incorrectly!
- at least 3 separate attempts within Google to fix this

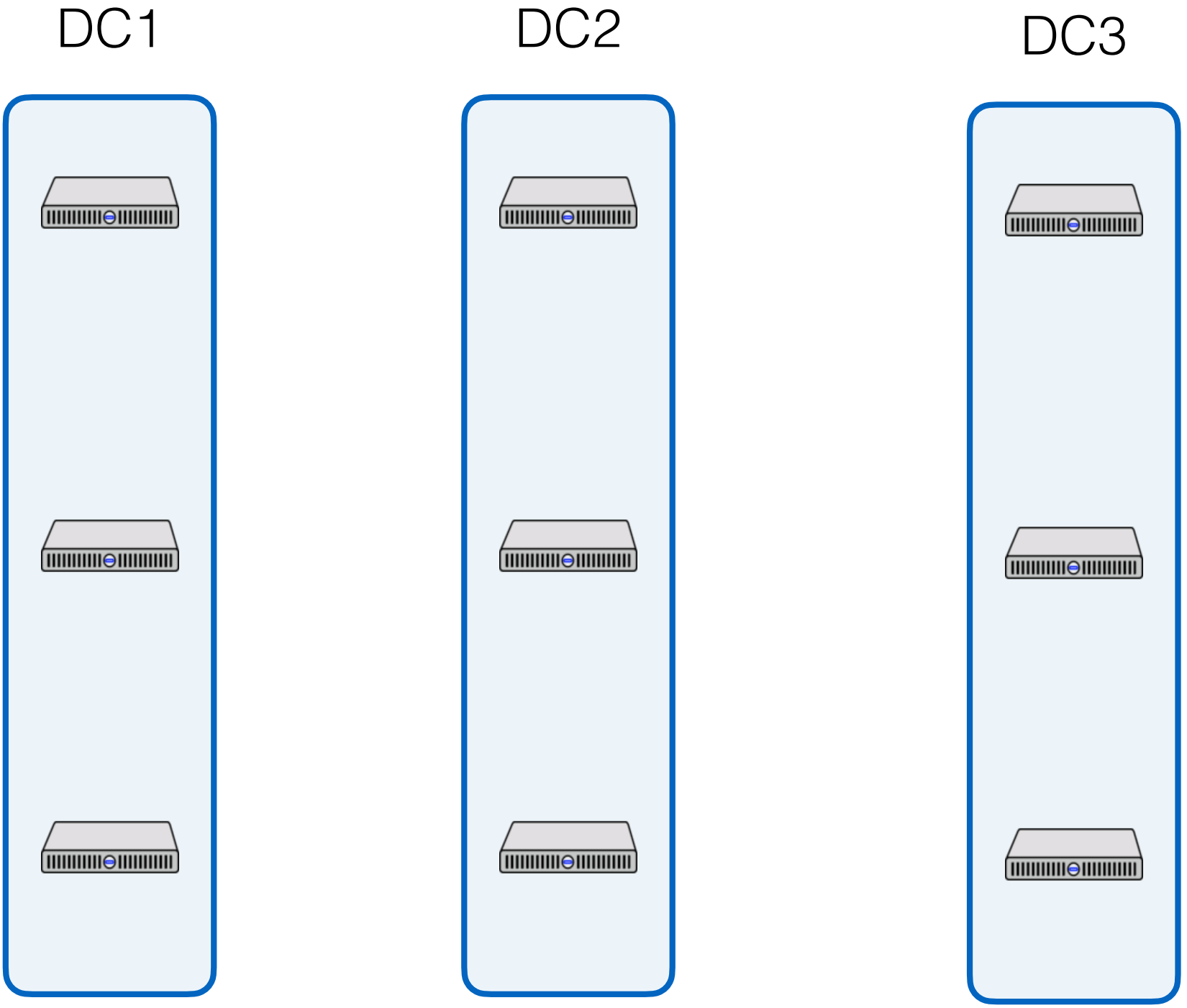
Spanner

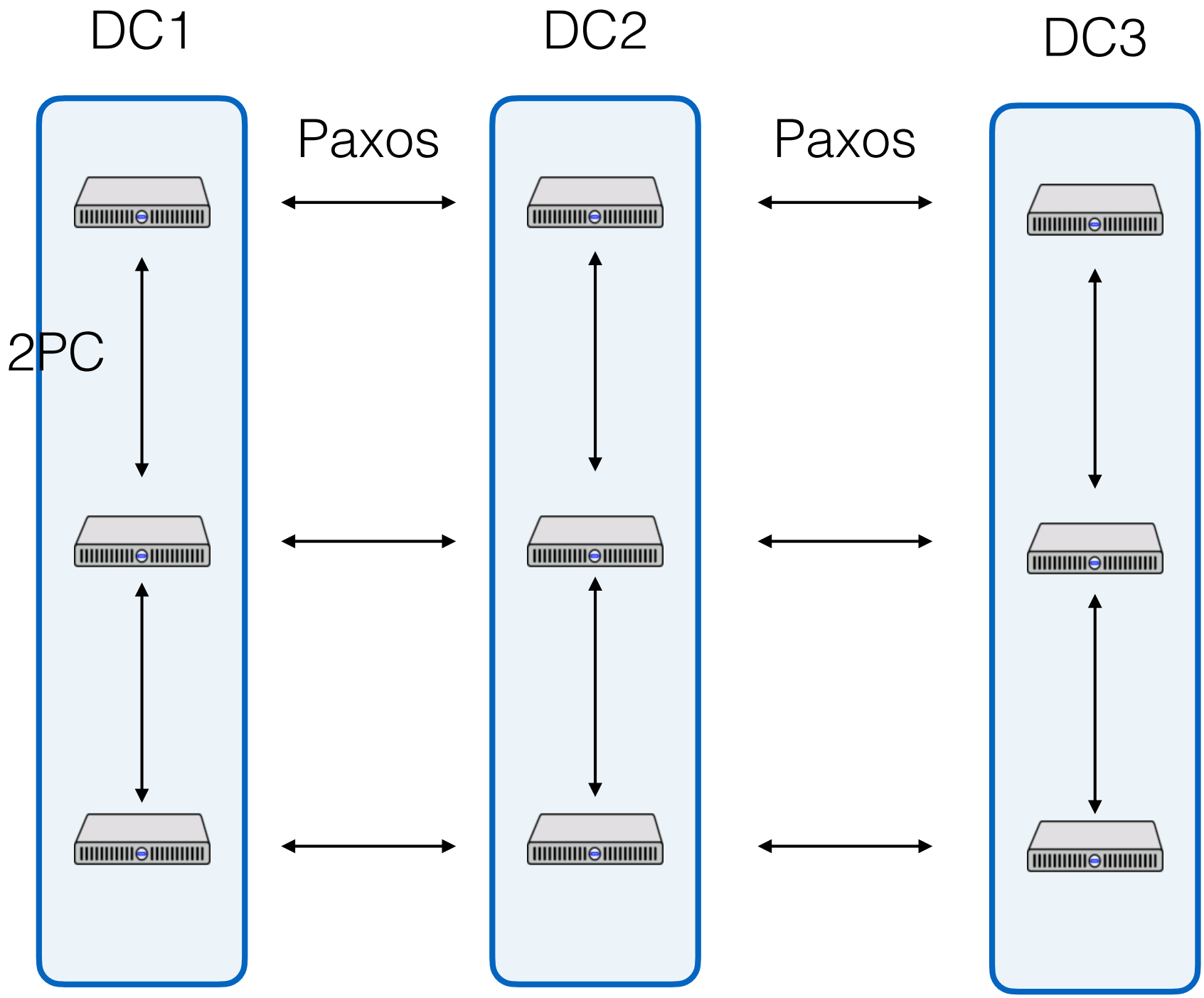
- Backend for the F1 database, which runs the ad system
- Basic model: 2PC over Paxos
- Uses physical clocks for performance

Spanner architecture



- Each shard is stored in a Paxos group
  - replicated across data centers (to survive data center failures)
  - Each group 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


DC1
DC2
DC3


DC1
DC2
DC3
Paxos
Paxos
2PC

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
- Then, ok to say: r/o transaction X reads at timestamp 10

TrueTime

- Common misconception: the magic here is fancy hardware (atomic clocks, GPS receivers)
  - this is actually relatively standard (see NTP)
- Actual key idea: expose the *uncertainty* in the clock value

TrueTime

API that exposes real time, with uncertainty

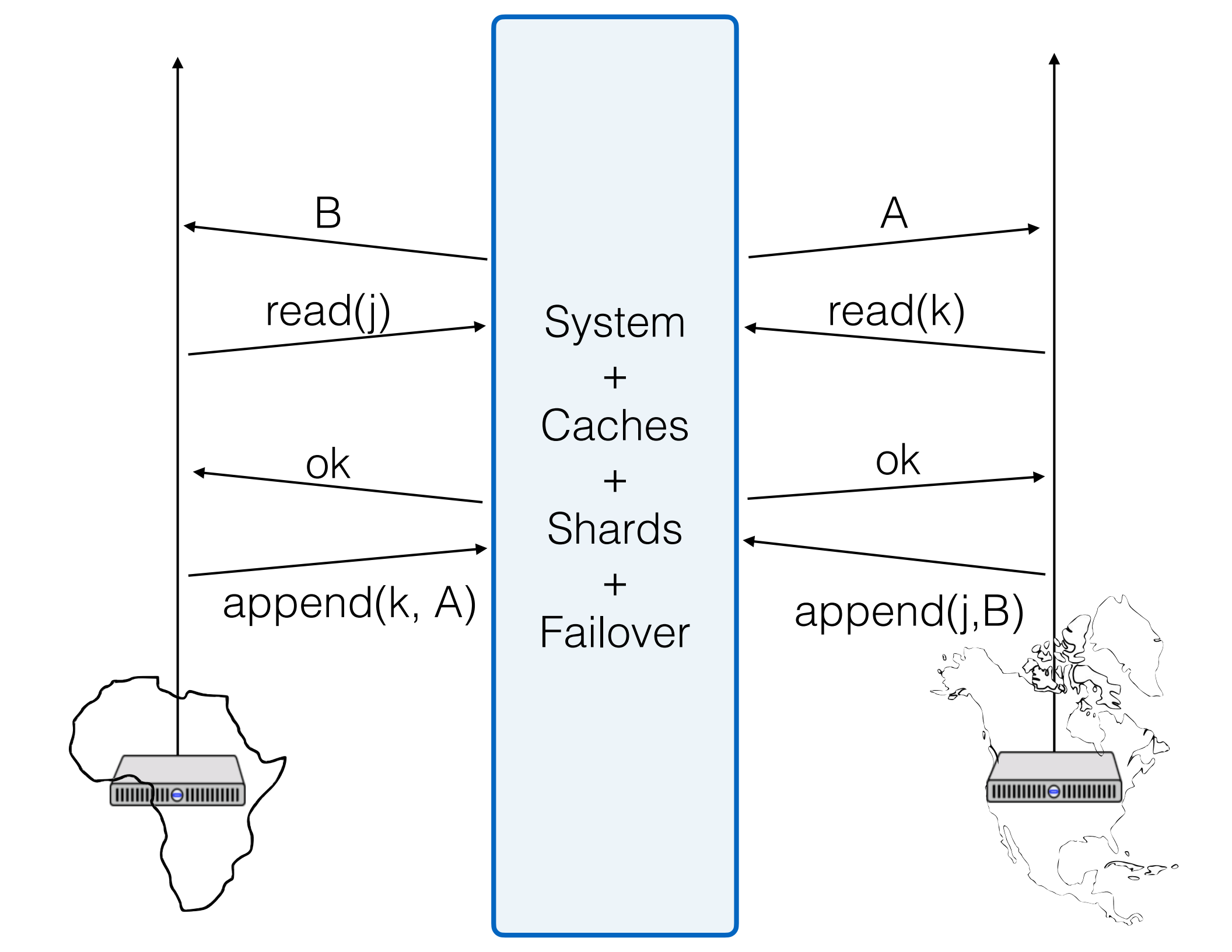
`{earliest: e, latest: l} = TT.now()`

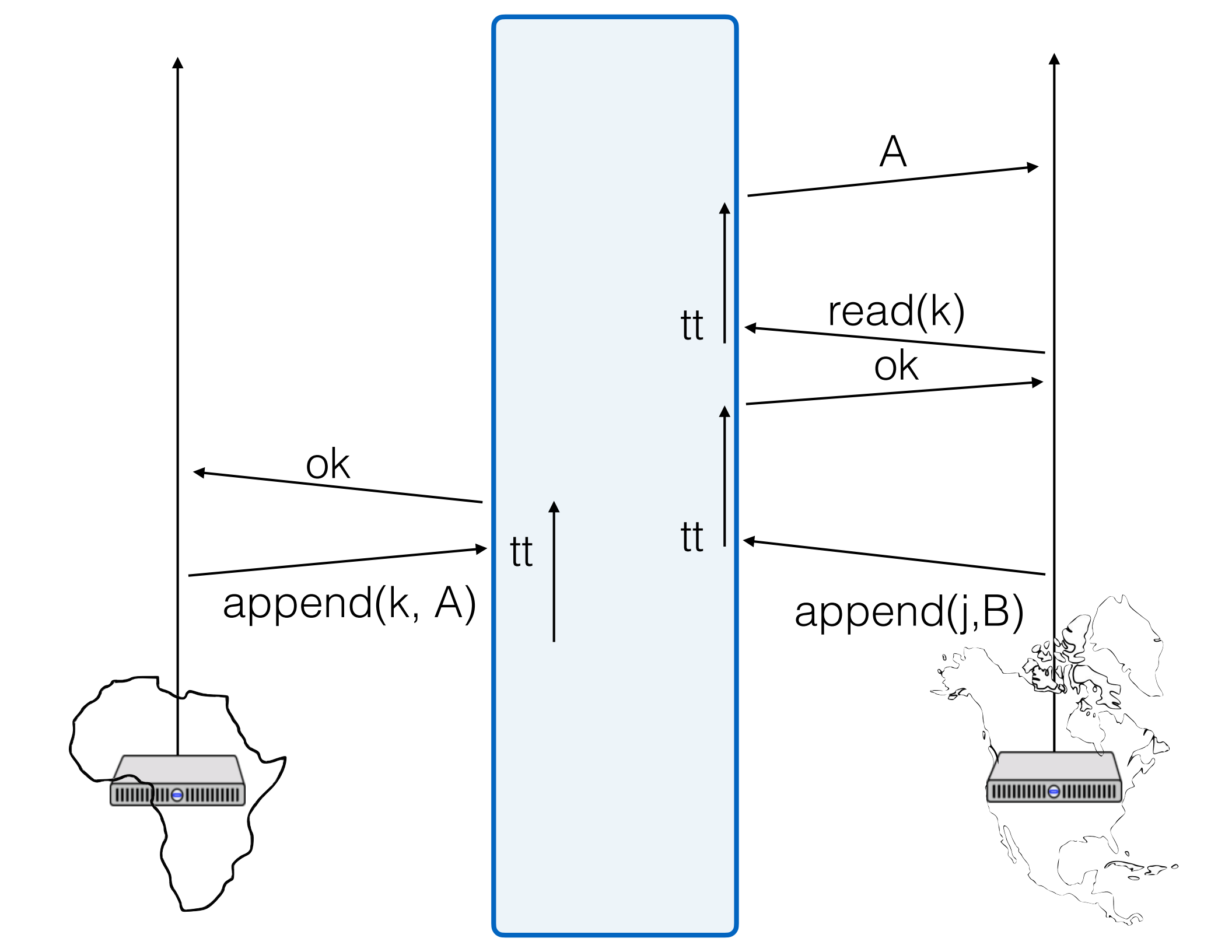
“Real time” is between `earliest` and `latest`

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!


B
read(j)
ok
append(k, A)
System
+
Caches
+
Shards
+
Failover
A
read(k)
ok
append(j,B)


A
tt
read(k)
ok
ok
tt
tt
append(k, A)
append(j,B)

TrueTime usage

Assign a timestamp to each transaction

- At each Paxos group, timestamp increases monotonically

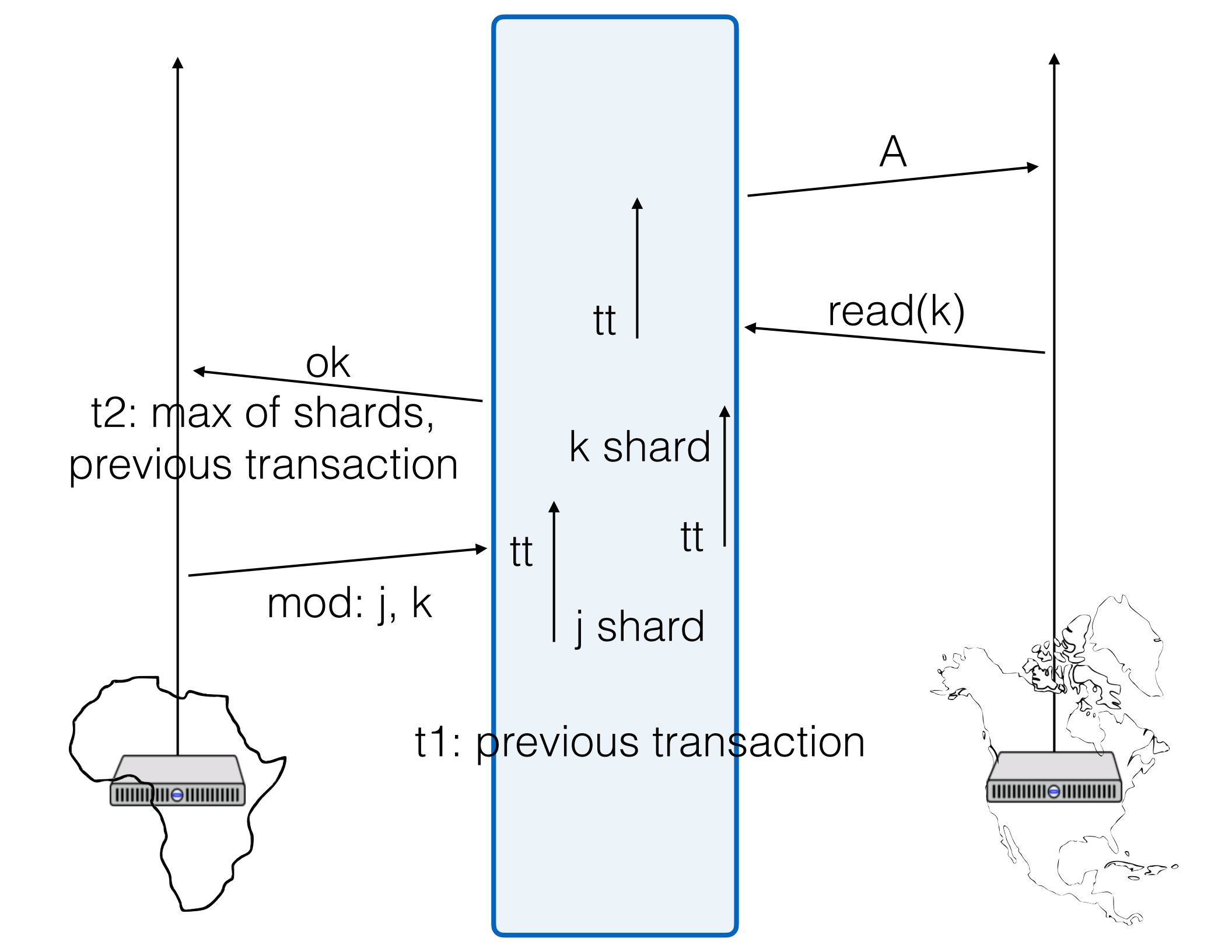
- Globally, if T1 returns before T2 starts,
  timestamp(T1) < timestamp(T2)

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


A
tt
read(k)
ok
t2: max of shards,
previous transaction
k shard
tt
tt
mod: j, k
j shard
t1: previous transaction

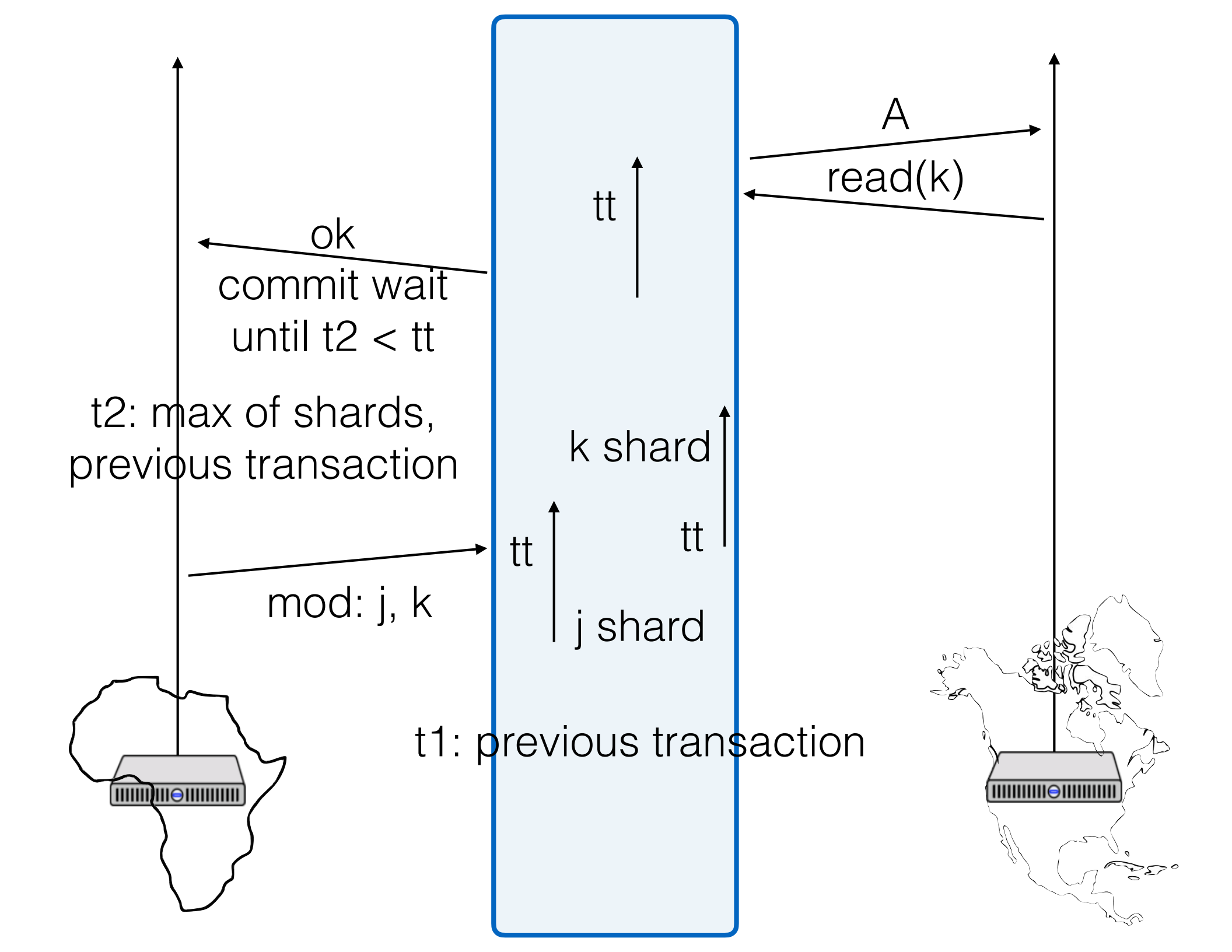
Commit wait

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

Therefore, need to wait until

TT.now() > transaction timestamp

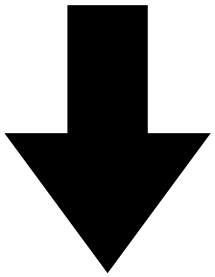
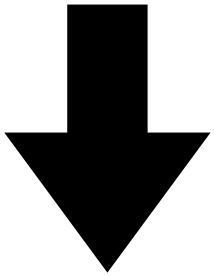
And only then release locks


A
read(k)
tt
ok
commit wait
until t2 < tt
t2: max of shards,
previous transaction
k shard
tt
tt
mod: j, k
j shard
t1: previous transaction

Leader 1 Client Leader 2

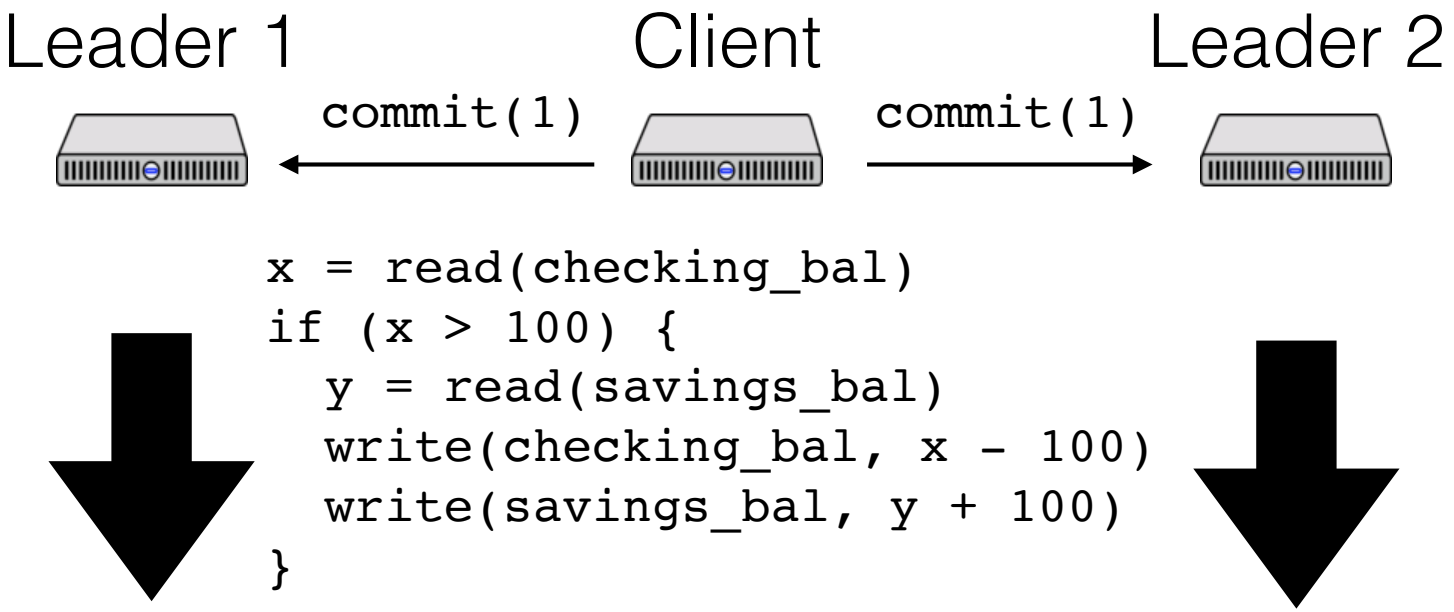


```
x = read(checking_bal)
if (x > 100) {
  y = read(savings_bal)
  write(checking_bal, x - 100)
  write(savings_bal, y + 100)
}
```

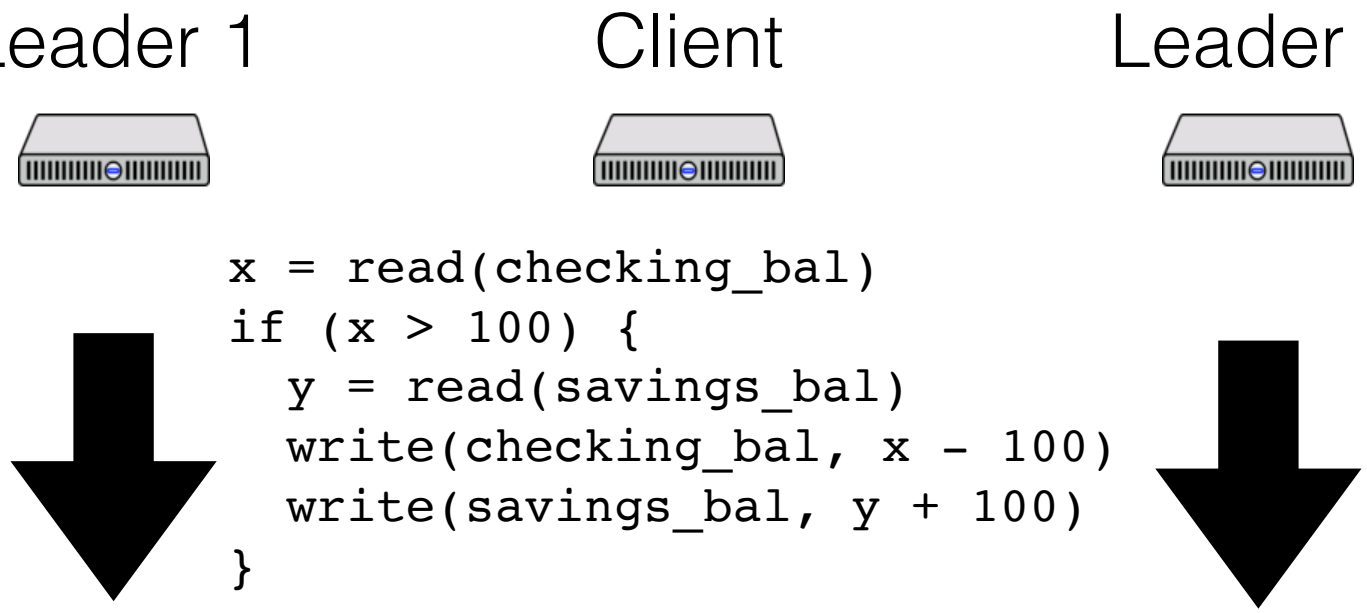
```
LOG:
 1. A: Lock(R, checking_bal)
 2. A: Lock(W, checking_bal)
 3. A: Write(checking_bal, 100)
```

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



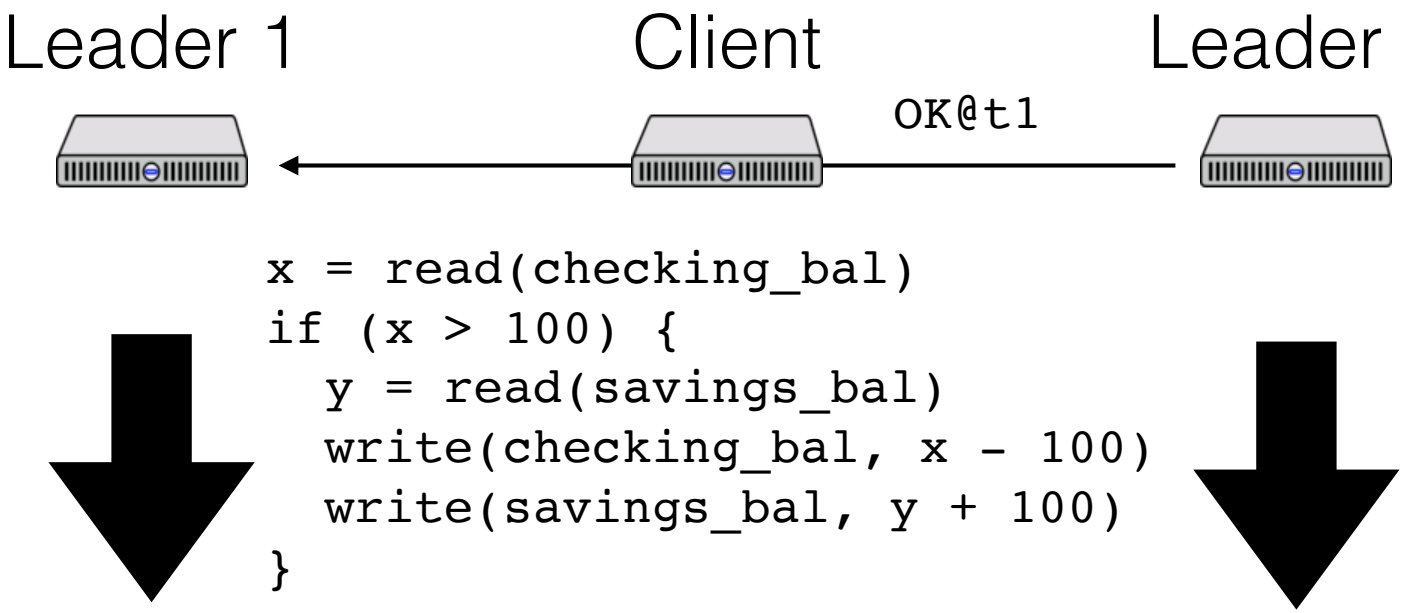

```
LOG:
 1. A: Lock(R, checking_bal)
 2. A: Lock(W, checking_bal)
 3. A: Write(checking_bal, 100)
```

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




```
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
```



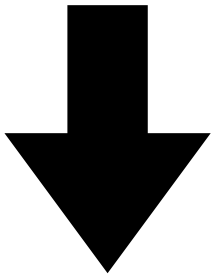
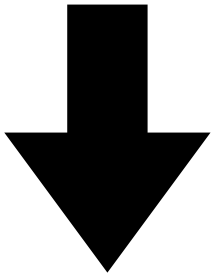

```
LOG:
 1. A: Lock(R, checking_bal)
 2. A: Lock(W, checking_bal)
 3. A: Write(checking_bal, 100)
```

```
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)
}
```

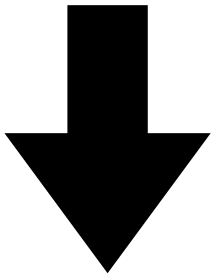
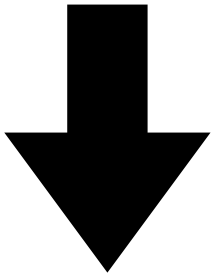
```
LOG:
 1. A: Lock(R, checking_bal)
 2. A: Lock(W, checking_bal)
 3. A: Write(checking_bal, 100)
 4. A: Commit@max(t1, tleader)
```

```
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)
```

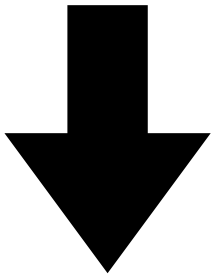
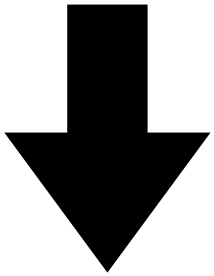
```
LOG:
 1. A: Lock(R, savings_bal)
 2. A: Lock(W, savings_bal)
 3. A: Write(savings_bal, 100)
 4. A: Prepare@t1
```

Waiting…

Leader 1 Client Leader 2

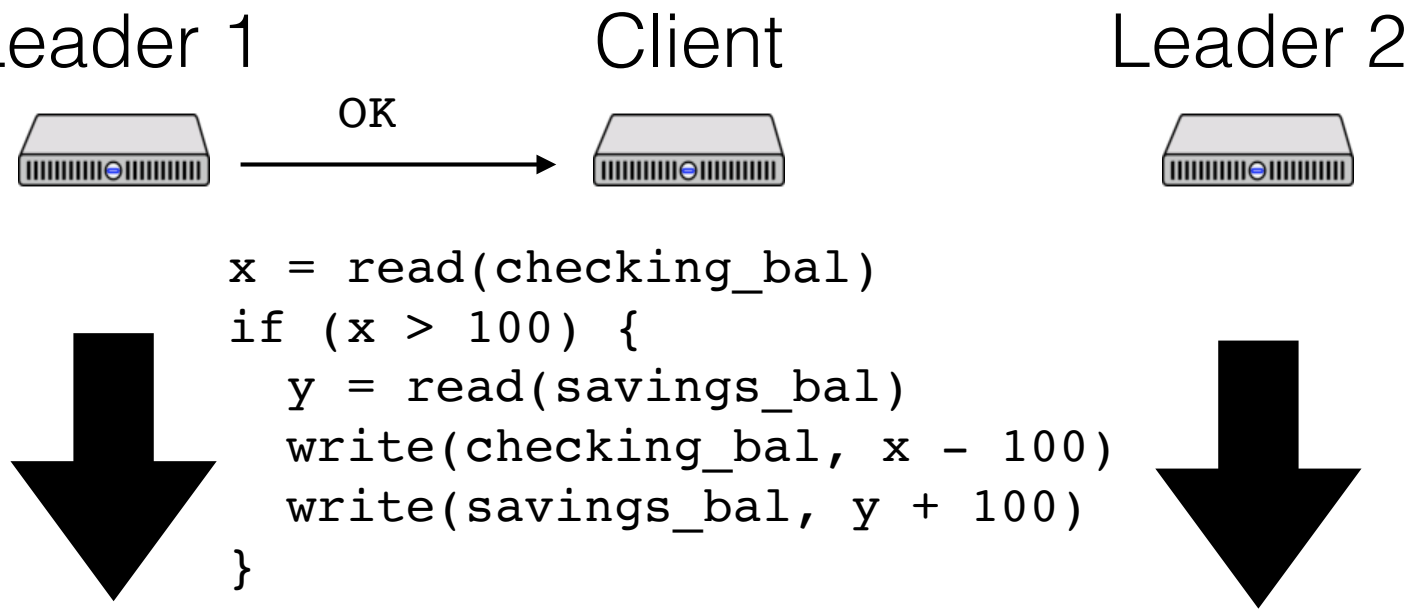


```
x = read(checking_bal)
if (x > 100) {
  y = read(savings_bal)
  write(checking_bal, x - 100)
  write(savings_bal, y + 100)
}
```

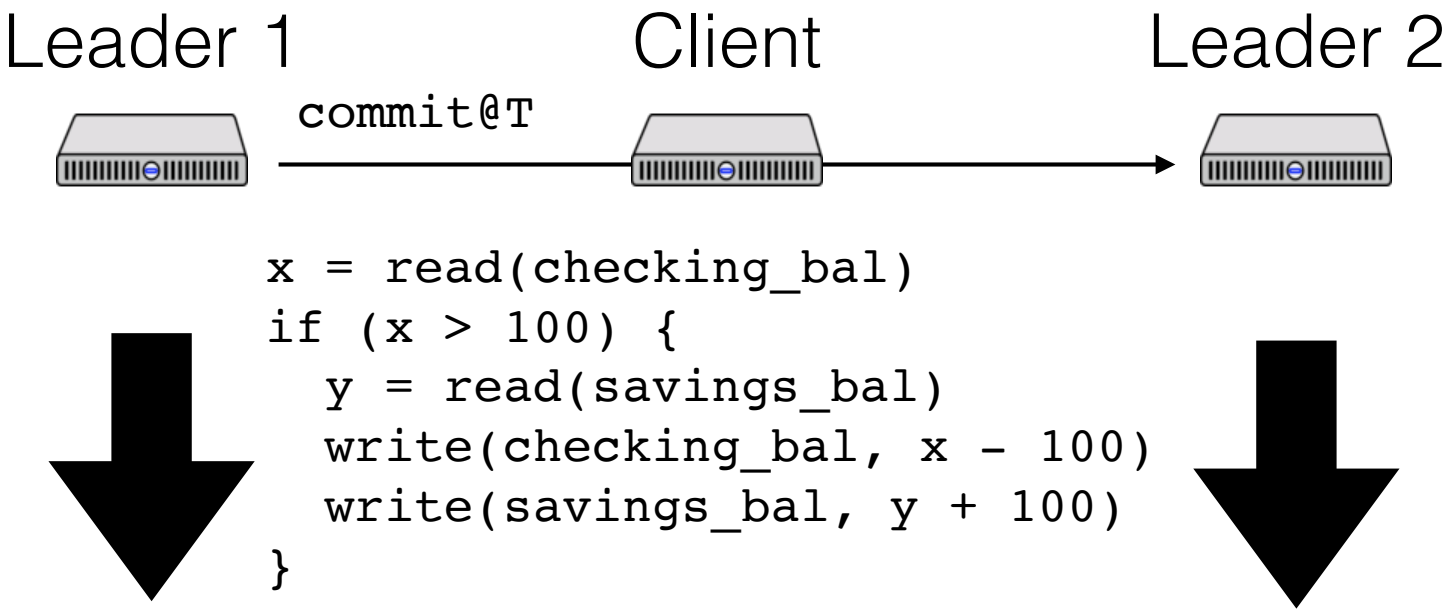
```
LOG:
 1. A: Lock(R, checking_bal)
 2. A: Lock(W, checking_bal)
 3. A: Write(checking_bal, 100)
 4. A: Commit@max(t1, tleader)
 5. A: Unlock(checking_bal)
```

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



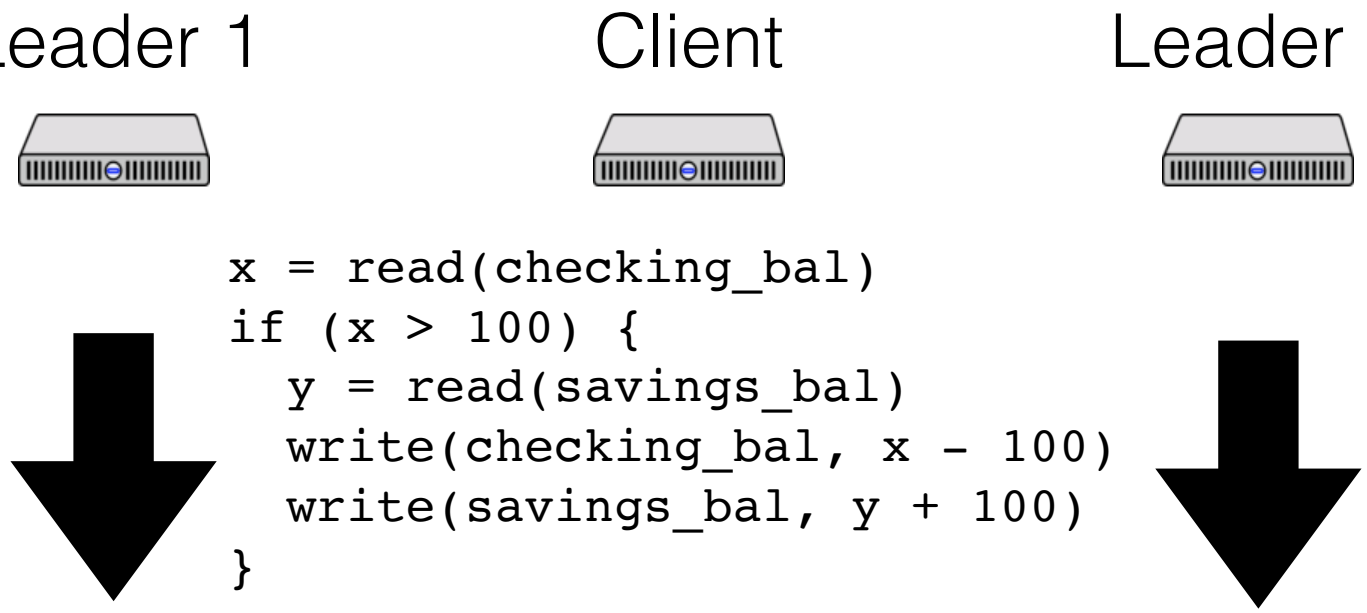

```
LOG:
 1. A: Lock(R, checking_bal)
 2. A: Lock(W, checking_bal)
 3. A: Write(checking_bal, 100)
 4. A: Commit@max(t1, tleader)
 5. A: Unlock(checking_bal)
```

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




```
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
```




```
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!