Distributed Transactions Without Atomic Clocks

Sometimes, it's all just about *good timing*

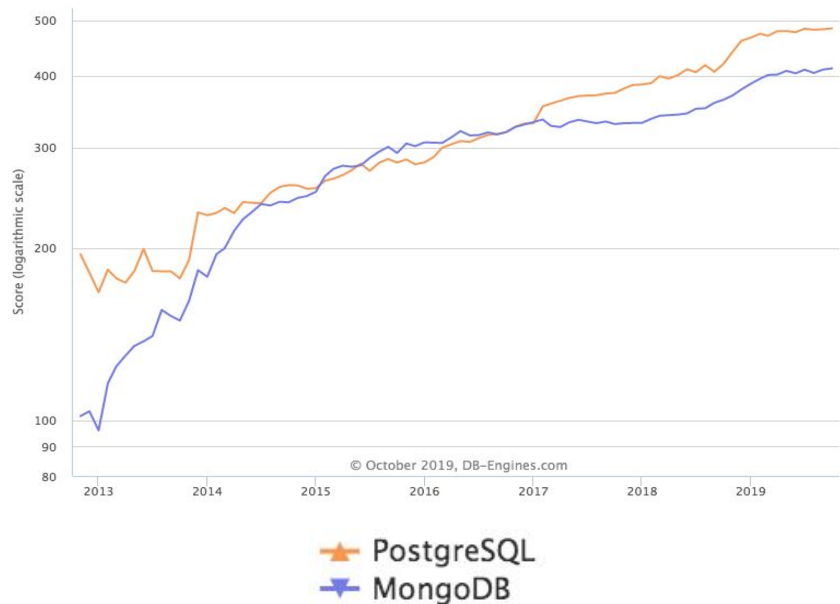
Karthik Ranganathan, co-founder & CTO

Introduction

© 2019 All rights reserved.

Designing the Perfect Distributed SQL Database

Skyrocketing adoption of PostgreSQL for cloud-native applications



Google Spanner

The first horizontally scalable, strongly consistent, relational database service

PostgreSQL is not highly available or horizontally scalable

Spanner does not have the RDBMS feature set

© 2019 All rights reserved.

Design Goals for YugabyteDB

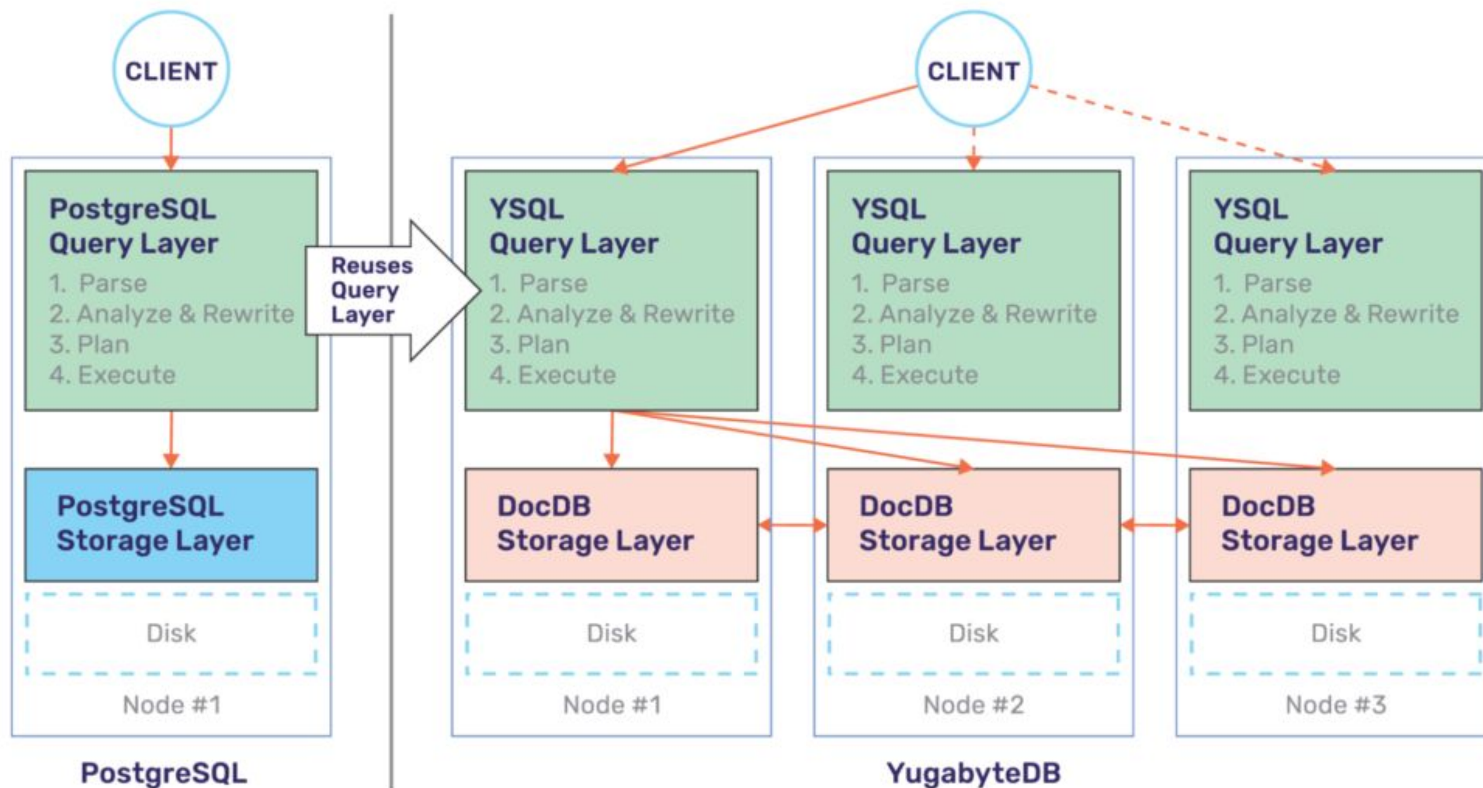
Transactional, distributed SQL database designed for resilience and scale

- 100% open source
- PostgreSQL compatible
- Enterprise-grade RDBMS
  - Day 2 operational simplicity
  - Secure deployments
- Public, private, hybrid clouds
- High performance

	PostgreSQL	Google Spanner	YugabyteDB
SQL Ecosystem	✓ Massively adopted	✘ New SQL flavor	✓ **Reuse PostgreSQL**
RDBMS Features	✓ Advanced *Complex*	✘ Basic *cloud-native*	✓ **Advanced** *Complex and cloud-native*
Highly Available	✘	✓	✓
Horizontal Scale	✘	✓	✓
Distributed Txns	✘	✓	✓
Data Replication	Async	Sync	Sync + Async

© 2019 All rights reserved.

YugabyteDB Reuses PostgreSQL Query Layer



© 2019 All rights reserved.

Transactions are fundamental to SQL...

But they require time synchronization between nodes.

Why?

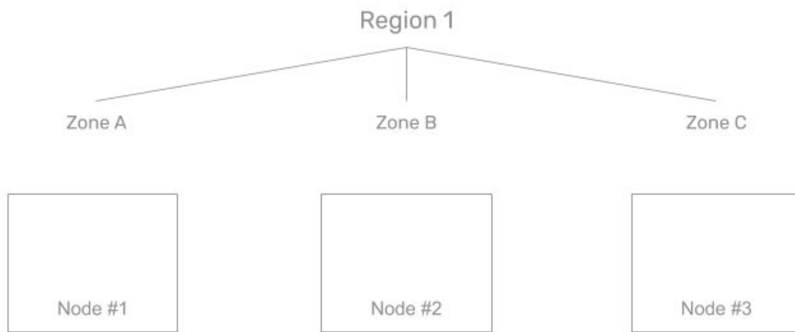
Let's look at single-row transactions before answering this

© 2019 All rights reserved.

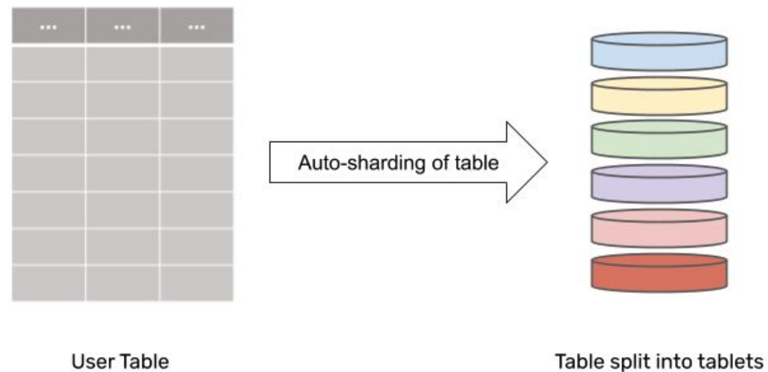
Single-Row Transactions: Raft Consensus

© 2019 All rights reserved.

Distributing Data For Horizontal Scalability



- Assume 3-nodes across zones
- How to distribute data across nodes?



- User tables sharded into tablets
- Tablet = group of rows
- Sharding is transparent to user

© 2019 All rights reserved.

Tablets Use Raft-Based Replication

Tablet data

Raft Algorithm for replicating data: per-row linearizability

1. Start leader election

A: Tablet Peer

B: Tablet Peer

C: Tablet Peer

2. Leader serves queries

User queries

A: Tablet Leader

B: Tablet Peer

C: Tablet Peer

© 2019 All rights reserved.

Replication in a 3 Node Cluster

- Assume rf = 3
- Survives 1 node or zone failure
- Tablets replicated across 3 nodes
- Follower (replica) tablets balanced across nodes in cluster

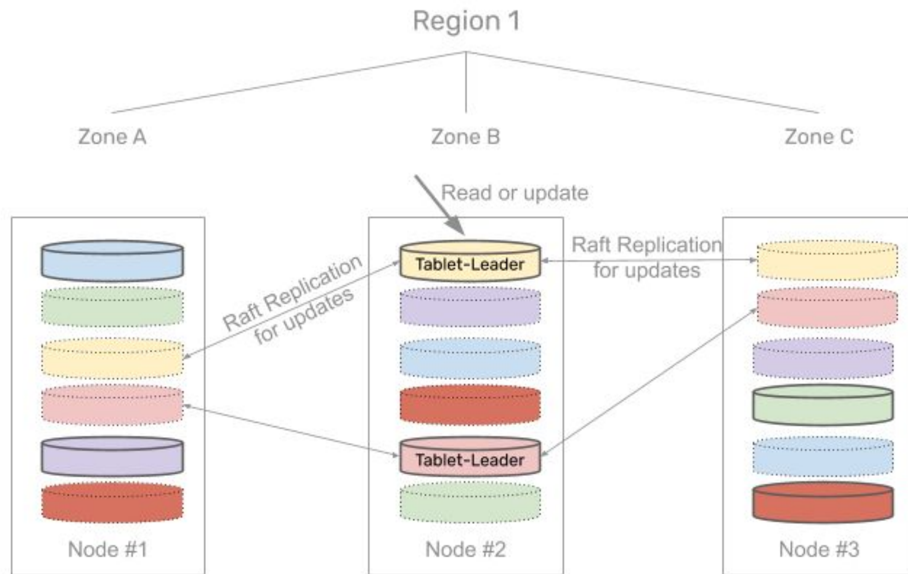


Diagram with replication factor = 3

© 2019 All rights reserved.

No Time Synchronization Needed So Far

YugabyteDB Raft replication based on:

`leader leases` + `time intervals` + `CLOCK_MONOTONIC`

From Jepsen Testing Report:

Whatever the case, this is a good thing for operators: nobody wants to worry about clock safety unless they have to, and YugaByte DB appears to be mostly robust to clock skew. Keep in mind that we cannot (rigorously) test YugaByte DB's use of CLOCK_MONOTONIC_RAW for leader leases, but we suspect skew there is less of an issue than CLOCK_REALTIME synchronization.

© 2019 All rights reserved.

Need for Time Synchronization: Distributed Transactions

© 2019 All rights reserved.

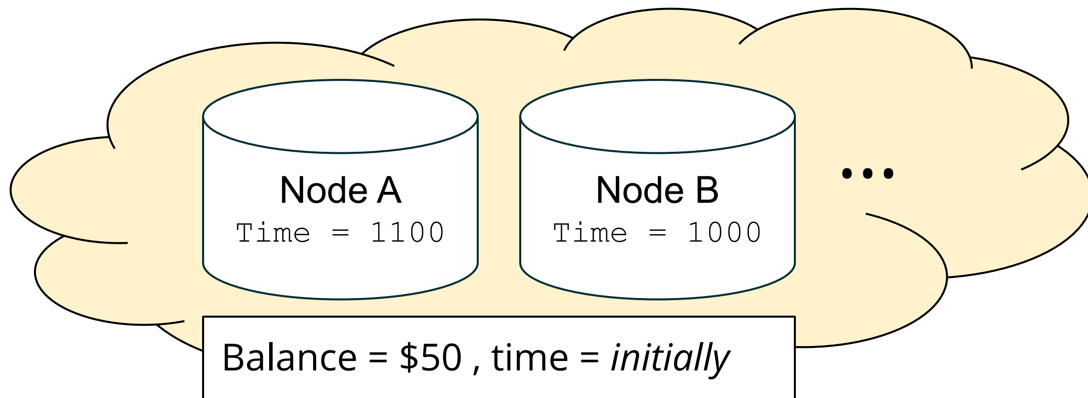
Let’s Take a Simple Example Scenario

User:

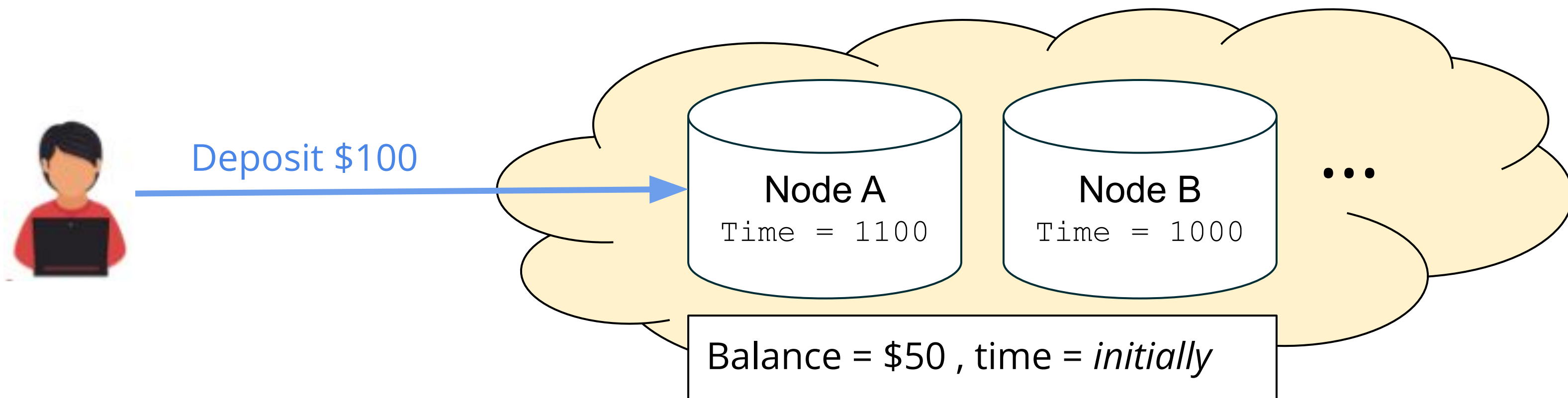
- *Deposit followed by withdrawal*
- A user has $50 in bank account
- Deposit $100, new balance = $150
- Withdraw $70, should be ok always

Cluster:

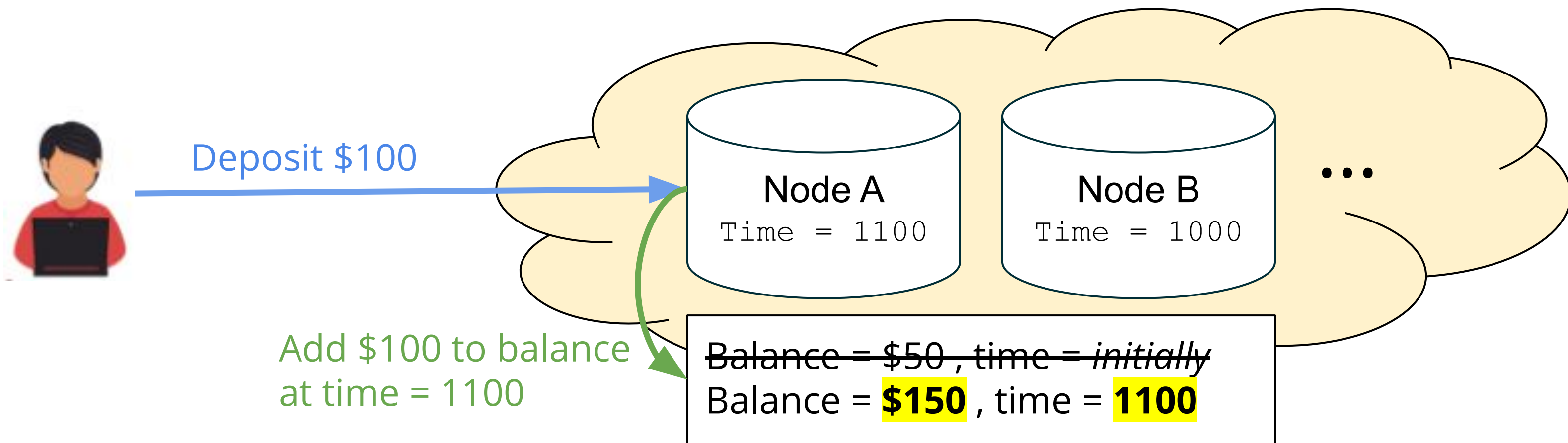
- *Nodes have clock skew*
- Node A is 100ms ahead of node B
- Deposit on Node A
- Withdraw from B, 50ms after deposit



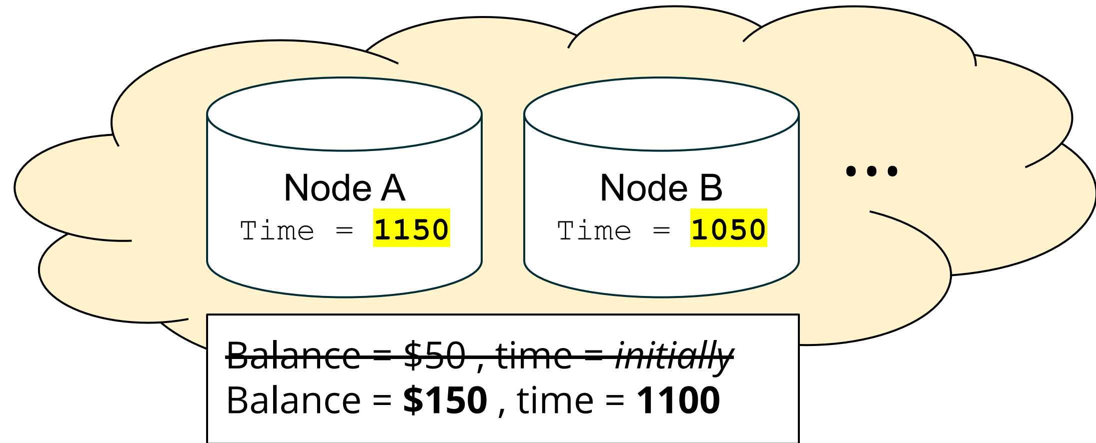
© 2019 All rights reserved.



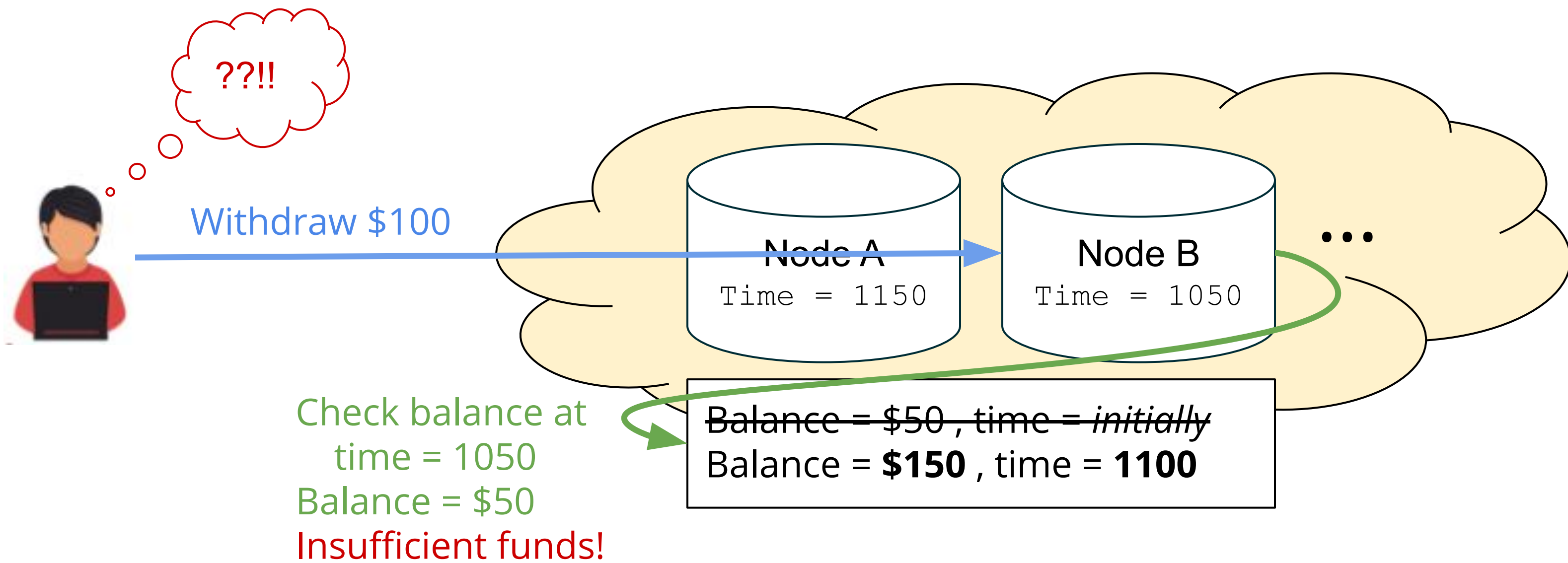
© 2019 All rights reserved.



© 2019 All rights reserved.



© 2019 All rights reserved.



© 2019 All rights reserved.

Time Sync Needed For Distributed Txns Also

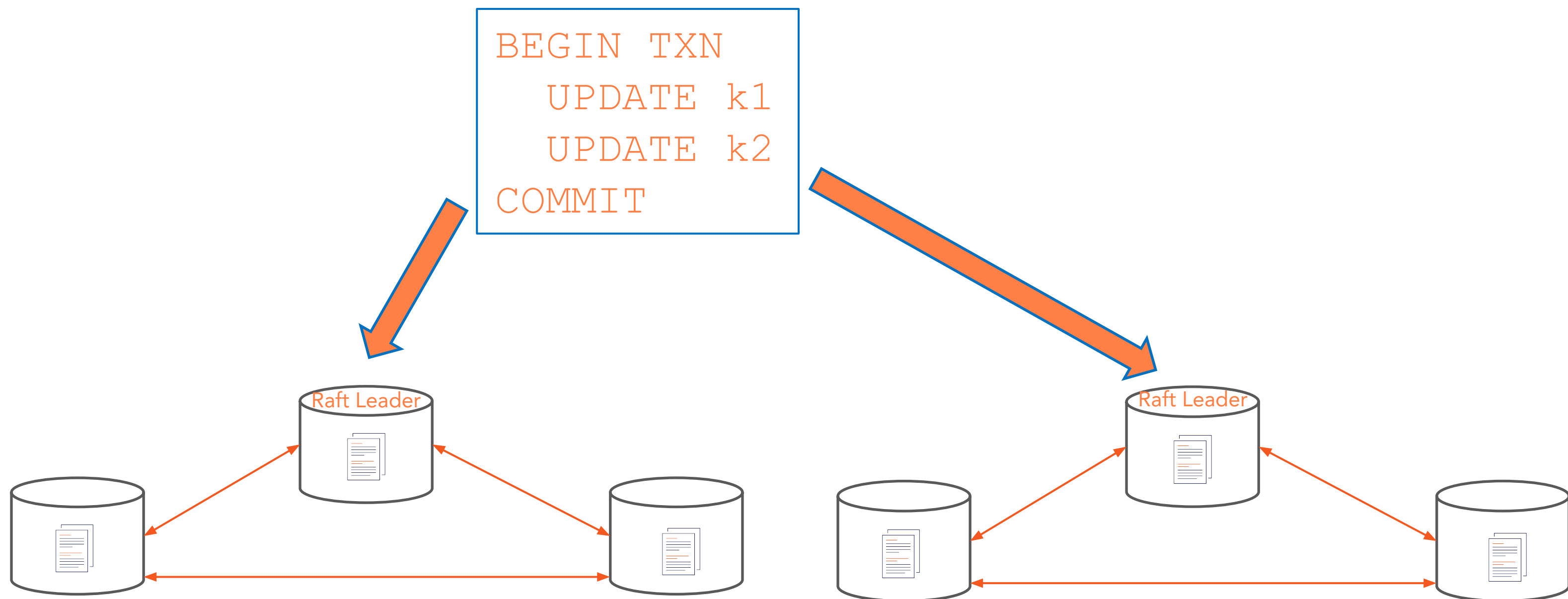
```
BEGIN TXN
  UPDATE k1
  UPDATE k2
COMMIT
```

k1 and k2 may belong to **different shards**

Belong to **different Raft groups** on completely **different nodes**

© 2019 All rights reserved.

What Do Distributed Transactions Need?



Updates should get written at the same time

But how will nodes agree on time?

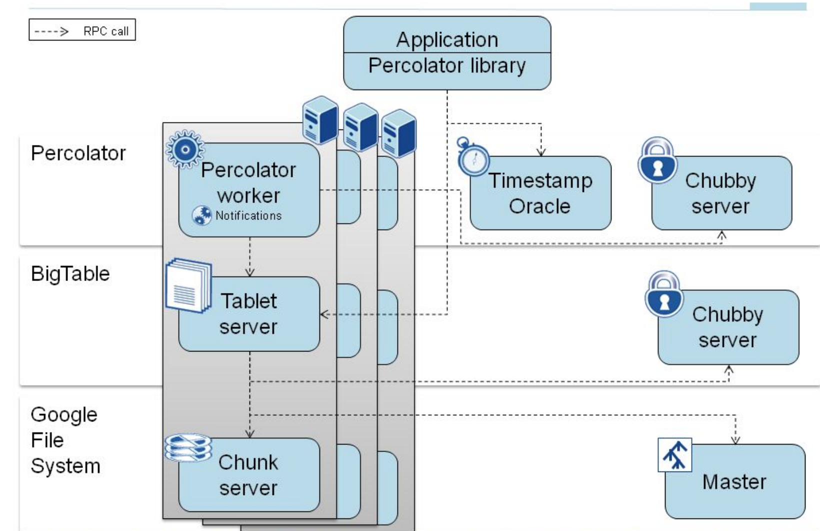
© 2019 All rights reserved.

Time Synchronization:
Timestamp Oracle vs Distributed Time Sync

© 2019 All rights reserved.

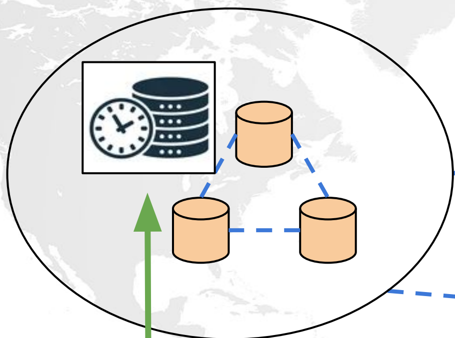
Timestamp Oracle - Google Percolator, Apache Omid

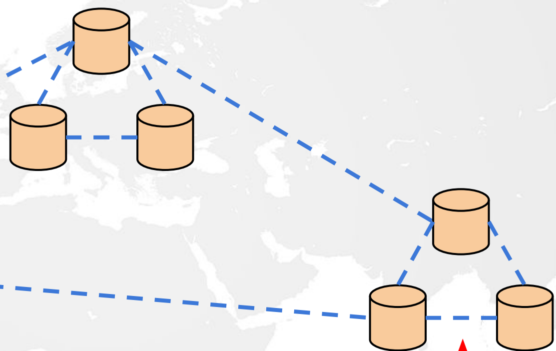
- Not scalable
  - Bottleneck in system
- Poor multi-region deployments
  - High latency
  - Low availability
- Prediction
  - Clock sync will improve esp in public clouds over time



Sources: http://research.google.com/pubs/pub36726.html, http://labs.google.com/papers/bigtable.html, http://labs.google.com/papers/gfs.html

© 2019 All rights reserved.





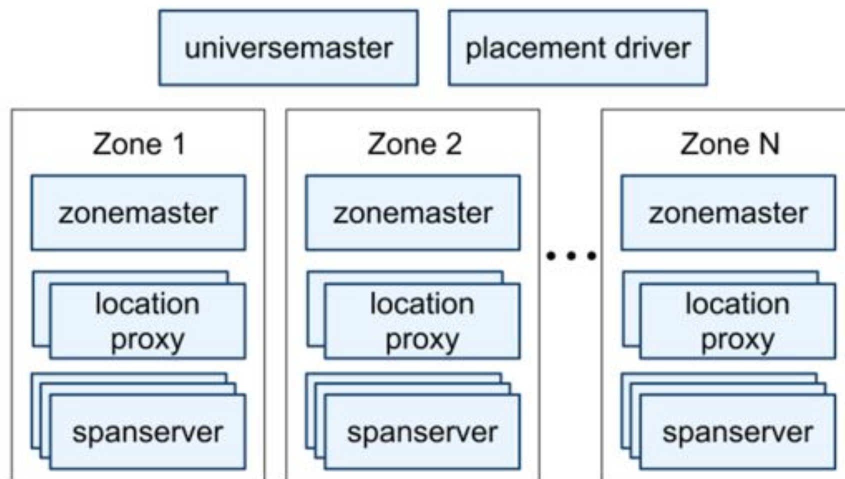
1. Timestamp Oracle gets partitioned away from rest of the cluster

2. Remote transactions fail without connectivity to the Timestamp Oracle

© 2019 All rights reserved.

Distributed Time Sync - Google Spanner

- Scalable
  - Only nodes involved in a txn need to coordinate
- Multi-region deployments
  - Distributed, region-local time synchronization
- Based on 2-phase commit
- Uses hybrid logical clocks



© 2019 All rights reserved.

Distributed Time Sync
Using GPS/Atomic Clock Service

© 2019 All rights reserved.

Atomic Clock Service

Atomic Clock Based Time Service:
highly available, globally synchronized clocks, tight error bounds

Not a commodity service

Most of physical clocks are very poorly synchronized:
`ntp` has clock skew of `100ms - 250ms`

© 2019 All rights reserved.

How Does an Atomic Clock Service Help?

Guarantees an upper bound on clock skew between nodes.

TrueTime service used by Google Spanner: 7ms max skew

Let’s try that scenario again with an atomic clock service

© 2019 All rights reserved.

Let’s Take an Example Scenario

GPS/Atomic clock service

Node A
Time = 1107

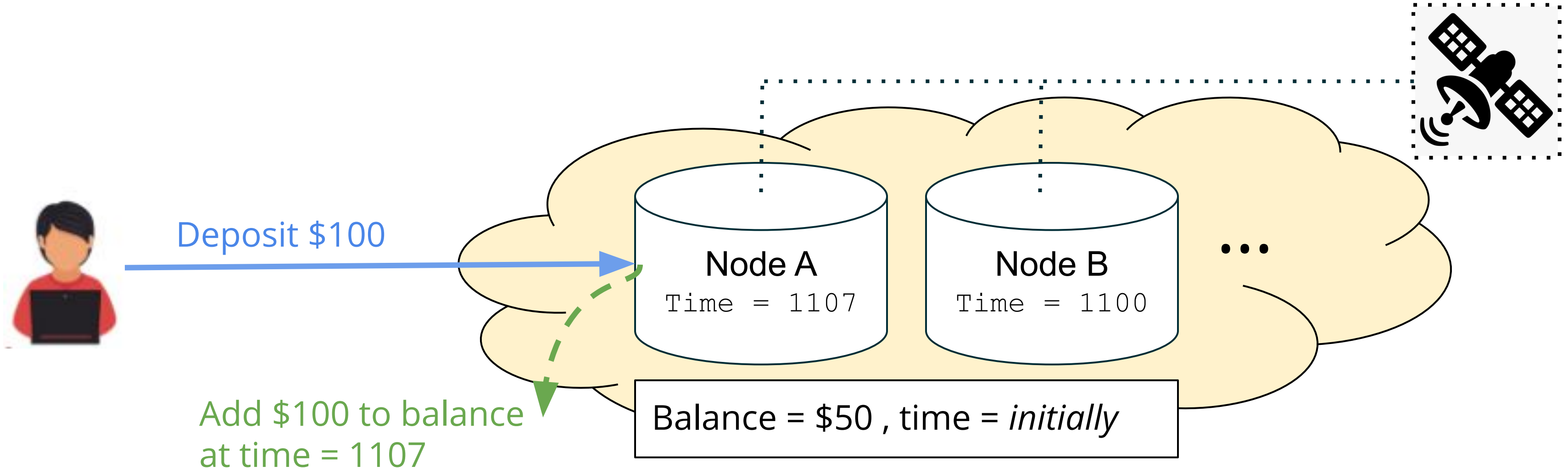
Node B
Time = 1100

...

Balance = $50 , time = *initially*

7ms max clock skew between the nodes

© 2019 All rights reserved.



© 2019 All rights reserved.

Deposit $100

Node A
Time = 1107

Node B
Time = 1100

...

Add $100 to balance
at time = 1107

Balance = $50 , time = *initially*

Introduce > 7ms **`commit_wait`** delay - allows all clocks to catch up

© 2019 All rights reserved.

Transaction conflicts detected and handled appropriately

Withdraw $70: CONFLICT

Deposit $100

Node A
Time = 1108

Node B
Time = 1101

...

Add $100 to balance
at time = 1100

Balance = $50 , time = *initially*

© 2019 All rights reserved.

Deposit $100

Node A
Time = **1117**

Node B
Time = **1110**

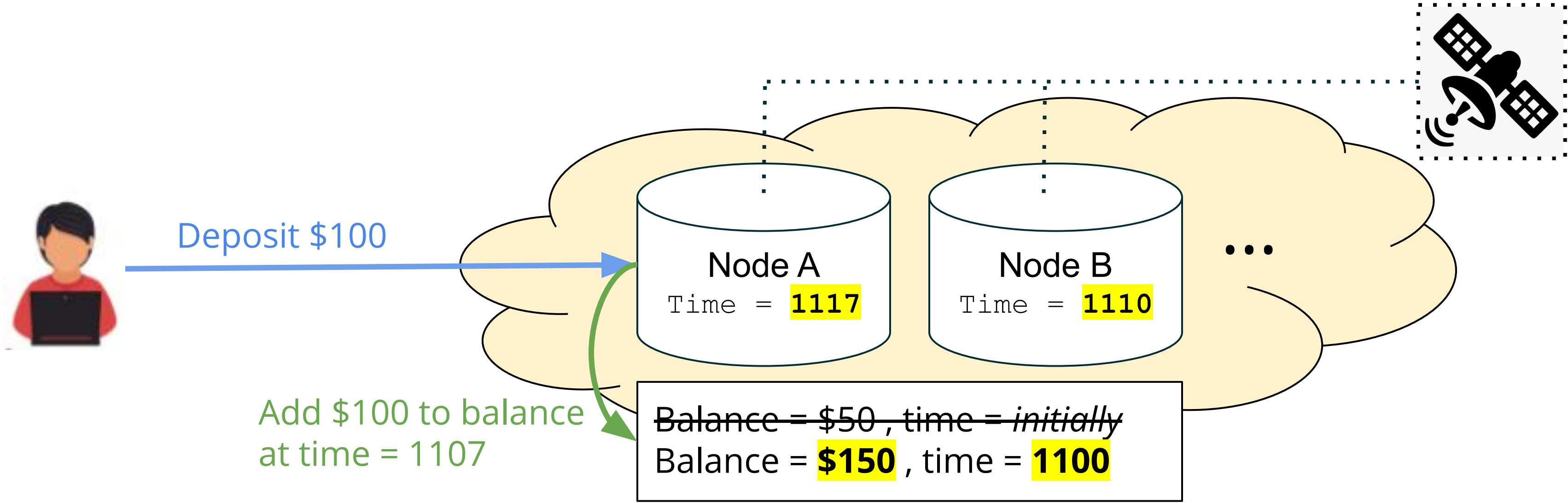
...

Add $100 to balance
at time = 1107

Balance = $50 , time = *initially*

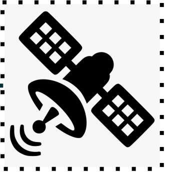
All clocks are guaranteed to have caught up to commit time 1107

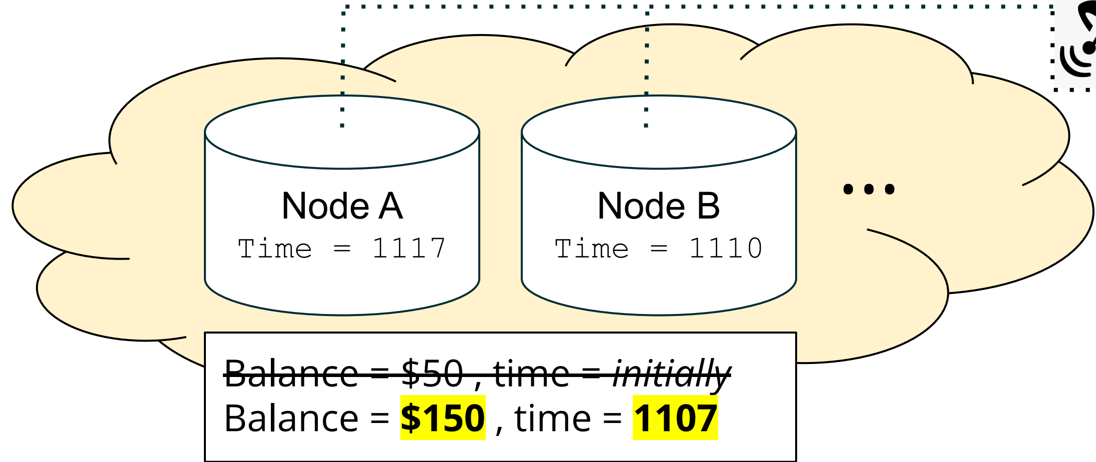
© 2019 All rights reserved.



© 2019 All rights reserved.

All transactions will now occur after time **1110** and hence are safe





© 2019 All rights reserved.

Doing This Without Atomic Clocks: Hybrid Logical Clocks

© 2019 All rights reserved.

Hybrid Logical Clock (HLC)

Combine coarsely-synchronized physical clocks with Lamport Clocks to track causal relationships

Hybrid Logical Clock = (physical component, logical component)

synchronized using NTP — a monotonic counter

Nodes update HLC on each Raft exchange for things like heartbeats, leader election and data replication

© 2019 All rights reserved.

64-bit HLC value

(`physical component` , `logical component`)

Physical timestamp

- 52-bit value
- Microsecond precision
- Synchronized using `ntp`

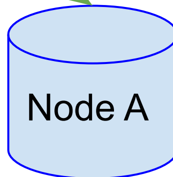
Logical timestamp

- 12-bit value
- 4096 values max
- Vector clock component

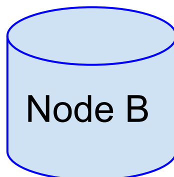
© 2019 All rights reserved.

UTC Time: T0

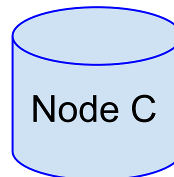
1. Update request



Wall clock: 1200
HLC time : (1200, 10)



Wall clock: 1000
HLC time : (1000, 30)
Skew : 200ms

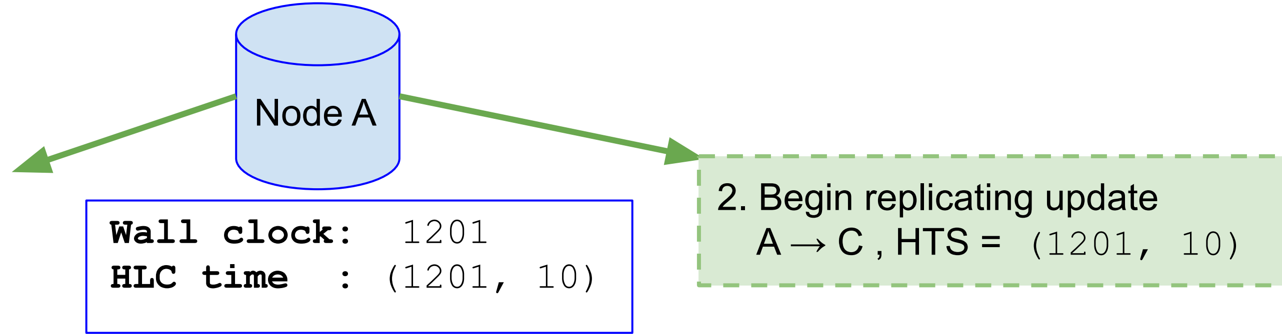


Wall clock: 1100
HLC time : (1100, 20)
Skew : 100ms

© 2019 All rights reserved.

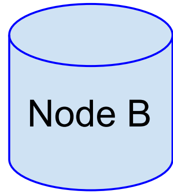
UTC Time: T0 + 1

Time has progressed on all nodes by 1 second

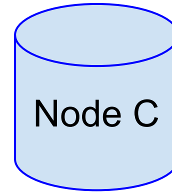


2. Begin replicating update A → B , HTS = (1201, 10)

2. Begin replicating update A → C , HTS = (1201, 10)



Wall clock: 1001
HLC time : (1001, 30)
Skew : 200ms

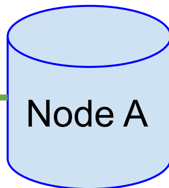


Wall clock: 1101
HLC time : (1101, 20)
Skew : 100ms

© 2019 All rights reserved.

UTC Time: T0 + 2

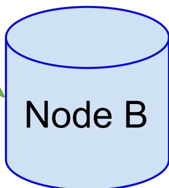
Time has progressed on all nodes by 2 second



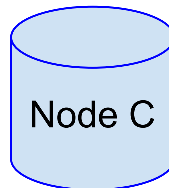
Wall clock: 1202
HLC time : (1202, 10)

3. Finish replication A → B after 1ms, HTS = (1201, 10)

3. Replication still in flight A → C after 1ms, HTS = (1201, 10)



Wall clock: 1002
HLC time : ~~(1002, 30)~~ (1201, 31)
Skew : ~~200ms~~ 1ms



Wall clock: 1102
HLC time : (1101, 20)
Skew : 100ms

© 2019 All rights reserved.

UTC Time: T0 + 3

Time has progressed
on all nodes by 3ms

Node A

Wall clock: 1203
HLC time : (1203, 10)

4. Finish replication
A → C after 2ms
HTS = (1201, 10)

Node B

Wall clock: 1003
HLC time : (1201, 31)
Skew : 2ms

Node C

Wall clock: 1103
HLC time : ~~(1103, 20)~~ (1201,
Skew : ~~100ms~~ 2ms

yugabyteDB

© 2019 All rights reserved.

Uncertainty Time-Windows and Max Clock Skew

© 2019 All rights reserved.

Typically, **frequent RPCs between nodes** forces quick time synchronization in a cluster

Sometimes, this may not be the case. In such cases, DB **transparently detects conflicting transactions** and **retries them if possible** to do so.

© 2019 All rights reserved.

Recall This Scenario We Discussed Before

Deposit $100

Node A
Time = **1107**

Node B
Time = **1100**

...

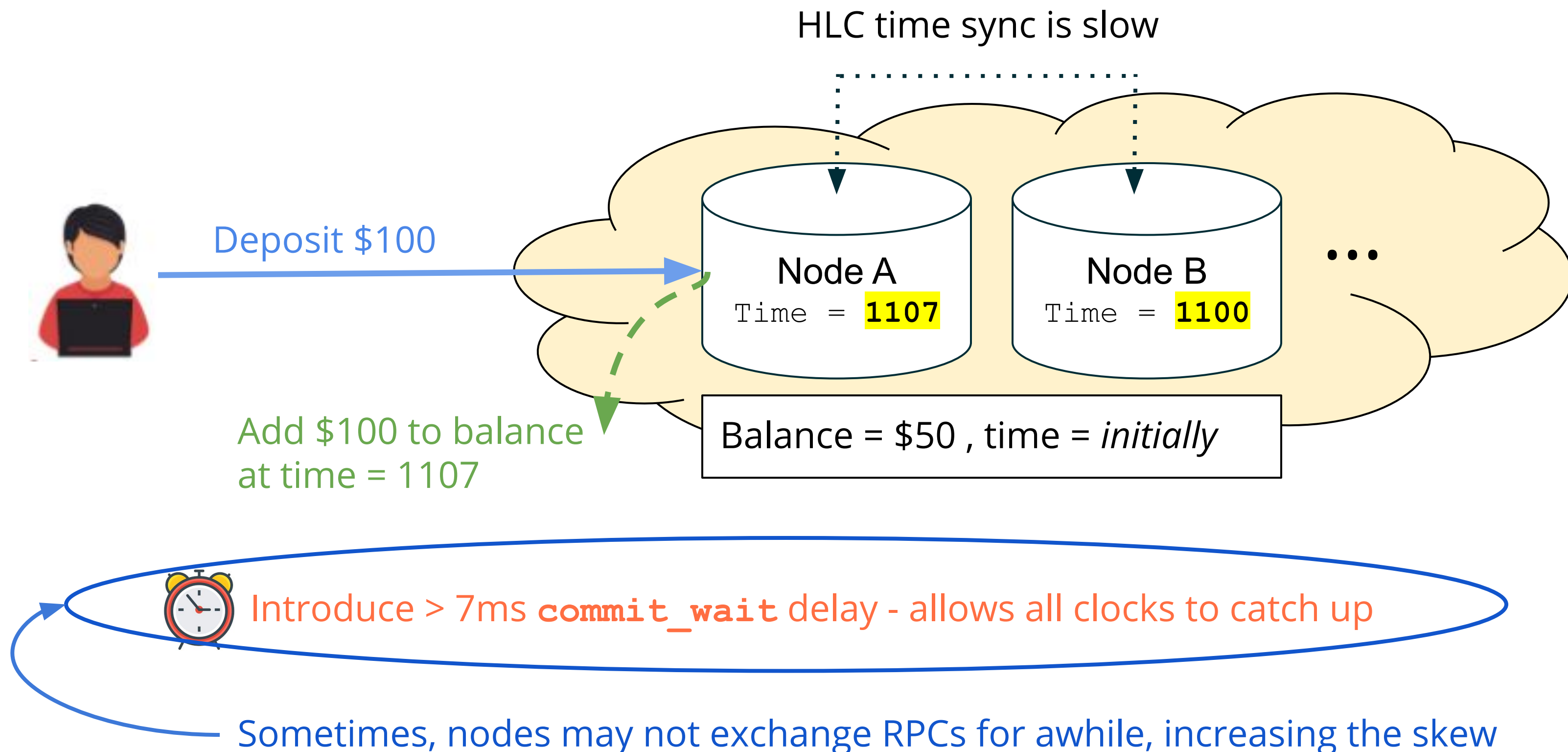
Balance = $50 , time = *initially*

Add $100 to balance
at time = 1107

Introduce > 7ms `commit_wait` delay - allows all clocks to catch up

© 2019 All rights reserved.

Knowing the Max Skew Is Important with HLCs



© 2019 All rights reserved.

Rare scenarios, but safety first!

Recommendation: set `max_skew` to large enough value

```
max_skew = 500ms # suggested default
```


© 2019 All rights reserved.

Integrating Raft with Hybrid Logical Clocks

© 2019 All rights reserved.

Raft vs Hybrid Logical Clock (HLC)

Raft Consensus:

- Per tablet
- Issues Raft Sequence ID, which is a purely logical sequence number
- Single row transactions
- Raft sequence id is monotonically increasing

Cluster:

- Per node
- Issues HLC timestamp, which can be compared across nodes
- Distributed transactions
- HLC timestamp is monotonically increasing

© 2019 All rights reserved.

Raft vs Hybrid Logical Clock (HLC)

Raft Consensus:

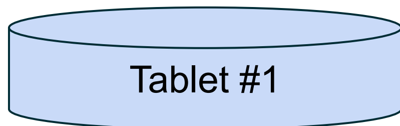
- Per tablet
- Issues Raft Sequence ID, which is a purely logical sequence number
- Single row transactions
- Raft sequence id is monotonically increasing

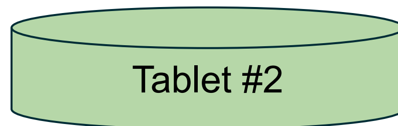
Cluster:

- Per node
- Issues HLC timestamp, which can be compared across nodes
- Distributed transactions
- HLC timestamp is monotonically increasing

Motonicity allows these two ids to be correlated

© 2019 All rights reserved.





Raft Log for tablet #1

```
Txn1: Raft-Id=100,
      HLC=(1000, 32),
      <txn 1 details>

Txn5: Raft-Id=101,
      HLC=(1005, 21),
      <txn 5 details>

Txn7: Raft-Id=102,
      HLC=(1100, 75),
      <txn 7 details>

…
```

Raft Log for tablet #2

```
Txn2: Raft-Id=1200,
      HLC=(1004, 14),
      <txn 2 details>

Txn7: Raft-Id=1201,
      HLC=(1100, 75),
      <txn 7 details>

Txn8: Raft-Id=1202,
      HLC=(1105, 91),
      <txn 8 details>

…
```


© 2019 All rights reserved.

Tablet #1

Tablet #2

Raft Ids are issued per-tablet and are not correlated across tablets.

HLCs are correlated across tablets because of distributed time sync

Raft Log for tablet #

Txn1: Raft-Id=100,
HLC=(1000, 32),
<txn 1 details>

Txn5: Raft-Id=101,
HLC=(1005, 21),
<txn 5 details>

Txn7: **Raft-Id=102,**
HLC=(1100, 75),
<txn 7 details>

…

t Log for tablet #2

: Raft-Id=1200,
HLC=(1004, 14),
<txn 2 details>

Txn7: **Raft-Id=1201,**
HLC=(1100, 75),
<txn 7 details>

Txn8: Raft-Id=1202,
HLC=(1100, 32),
<txn 8 details>

…

© 2019 All rights reserved.

DocDB:
Distributed Transactions

© 2019 All rights reserved.

Fully Decentralized Architecture

- **No single point of failure or bottleneck**
  - Any node can act as a Transaction Manager
- **Transaction status table distributed across multiple nodes**
  - Tracks state of active transactions
- **Transactions have 3 states**
  - Pending
  - Committed
  - Aborted
- **Reads served only for Committed Transactions**
  - Clients never see inconsistent data

© 2019 All rights reserved.

Isolation Levels

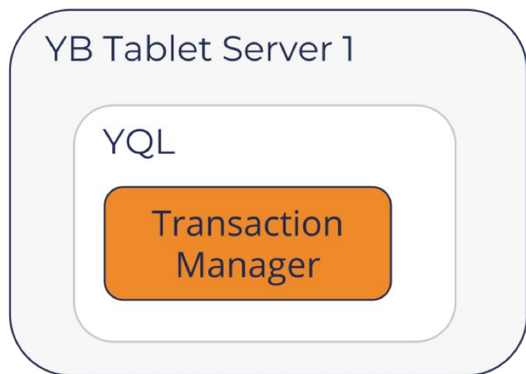
- **Serializable Isolation**
  - Read-write conflicts get auto-detected
  - Both reads and writes in read-write txns need provisional records
  - Maps to SERIALIZABLE in PostgreSQL

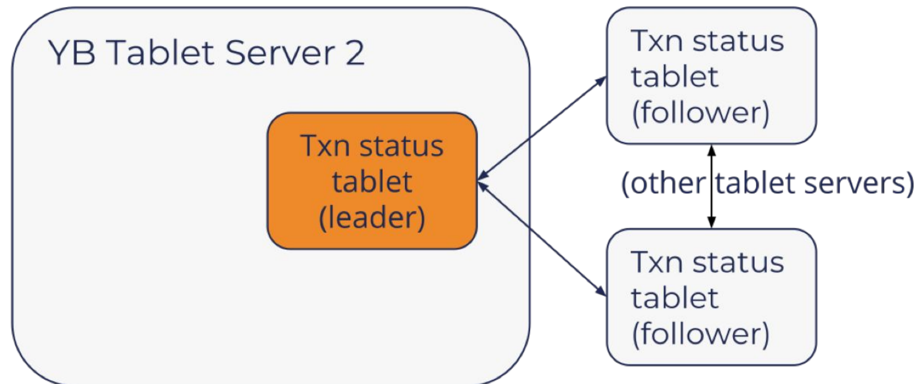
- **Snapshot Isolation**
  - Write-write conflicts get auto-detected
  - Only writes in read-write txns need provisional records
  - Maps to REPEATABLE READ, READ COMMITTED & READ UNCOMMITTED in PostgreSQL

- **Read-only Transactions**
  - Lock free

© 2019 All rights reserved.

Distributed Transactions - Write Path



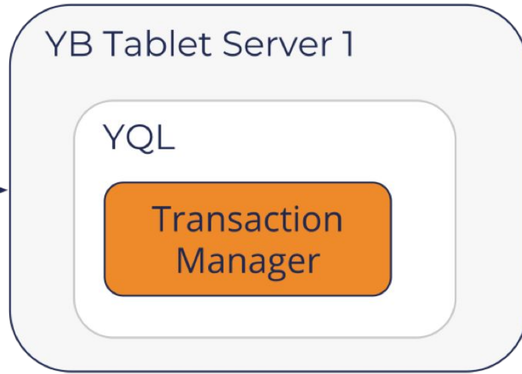


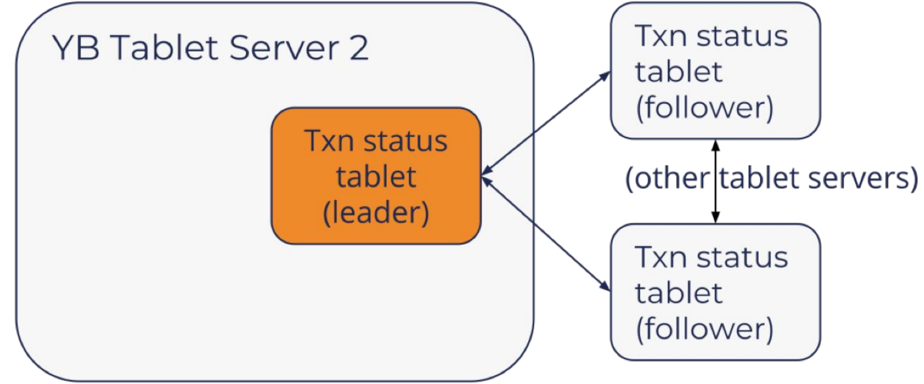
© 2019 All rights reserved.

Distributed Transactions - Write Path

1

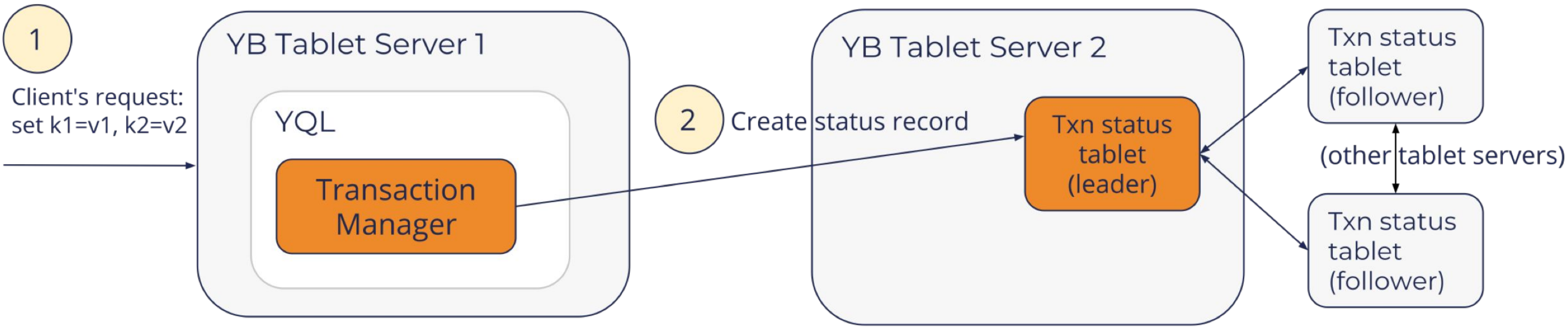
Client's request:
set k1=v1, k2=v2





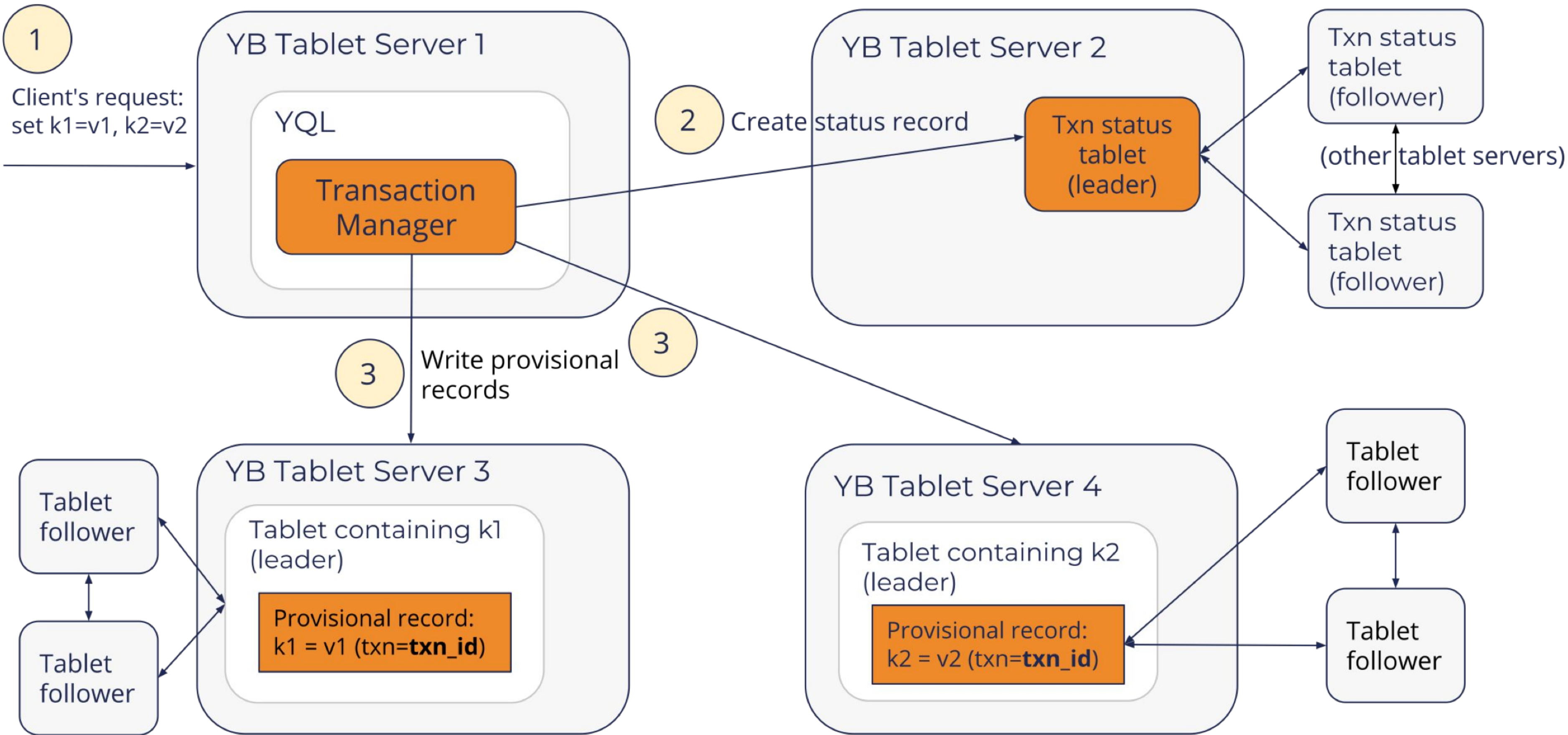
© 2019 All rights reserved.

Distributed Transactions - Write Path



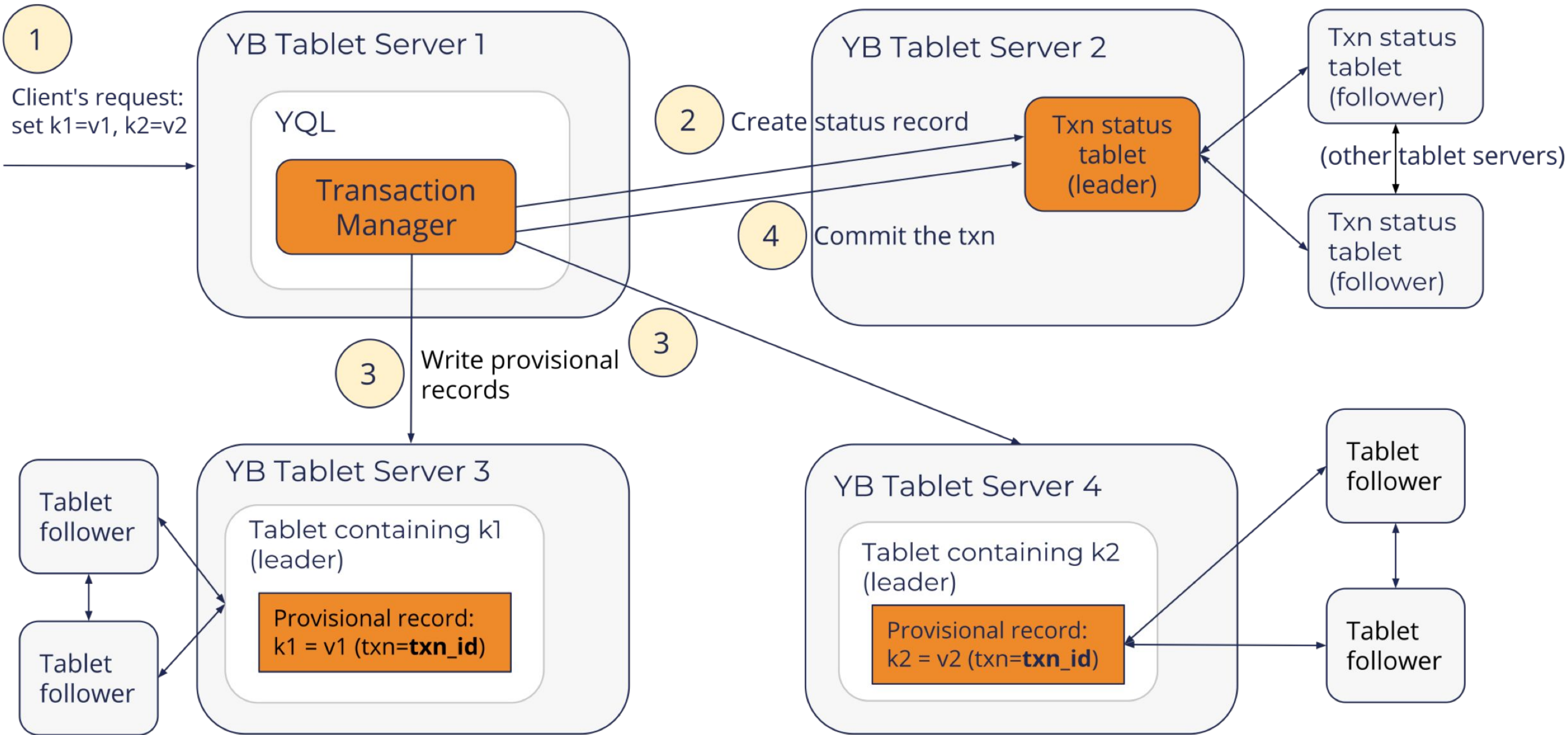
© 2019 All rights reserved.

Distributed Transactions - Write Path



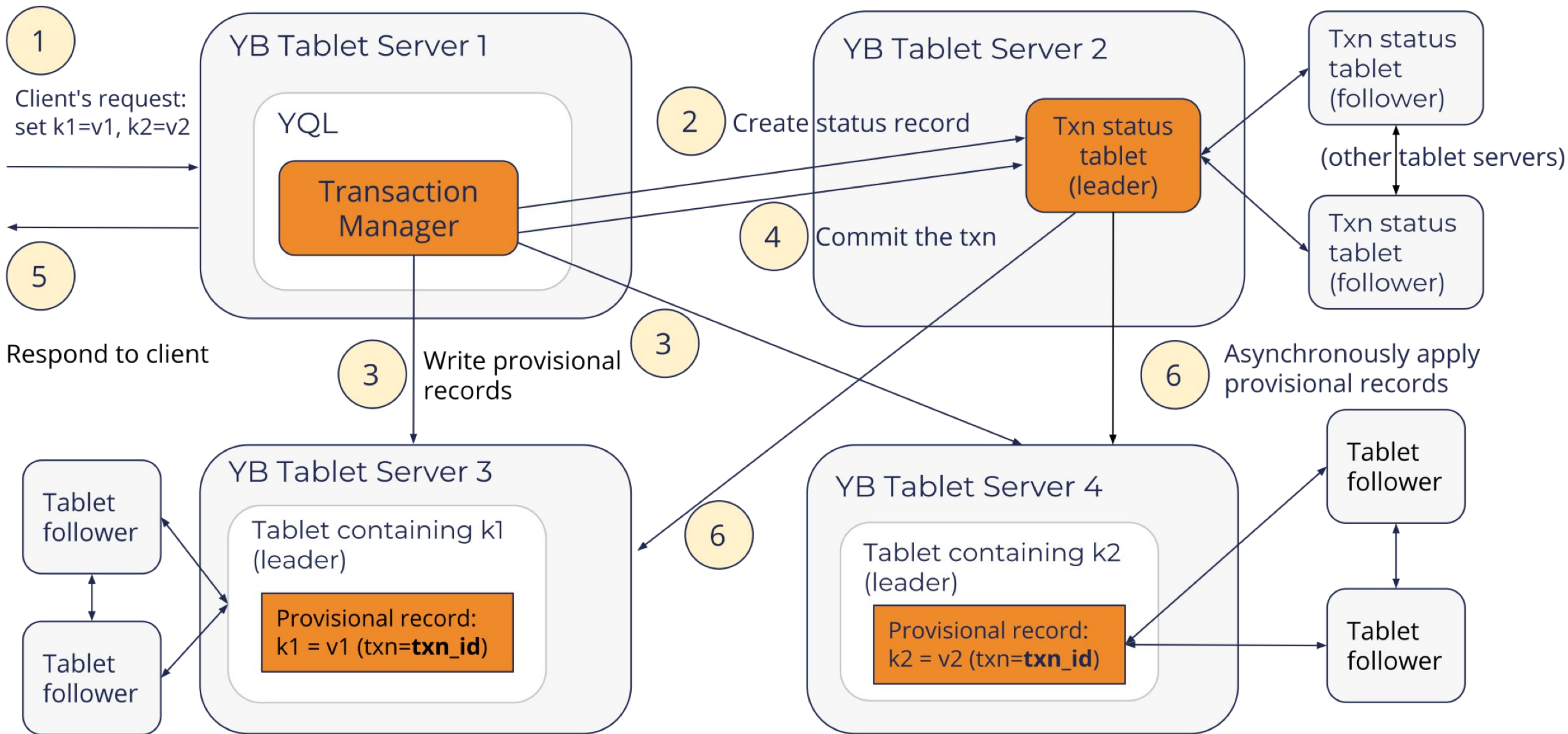
© 2019 All rights reserved.

Distributed Transactions - Write Path



© 2019 All rights reserved.

Distributed Transactions - Write Path



© 2019 All rights reserved.

Was that interesting?

If so, join us on Slack for more:

yugabyte.com/slack

© 2019 All rights reserved.

Thanks!

© 2019 All rights reserved.