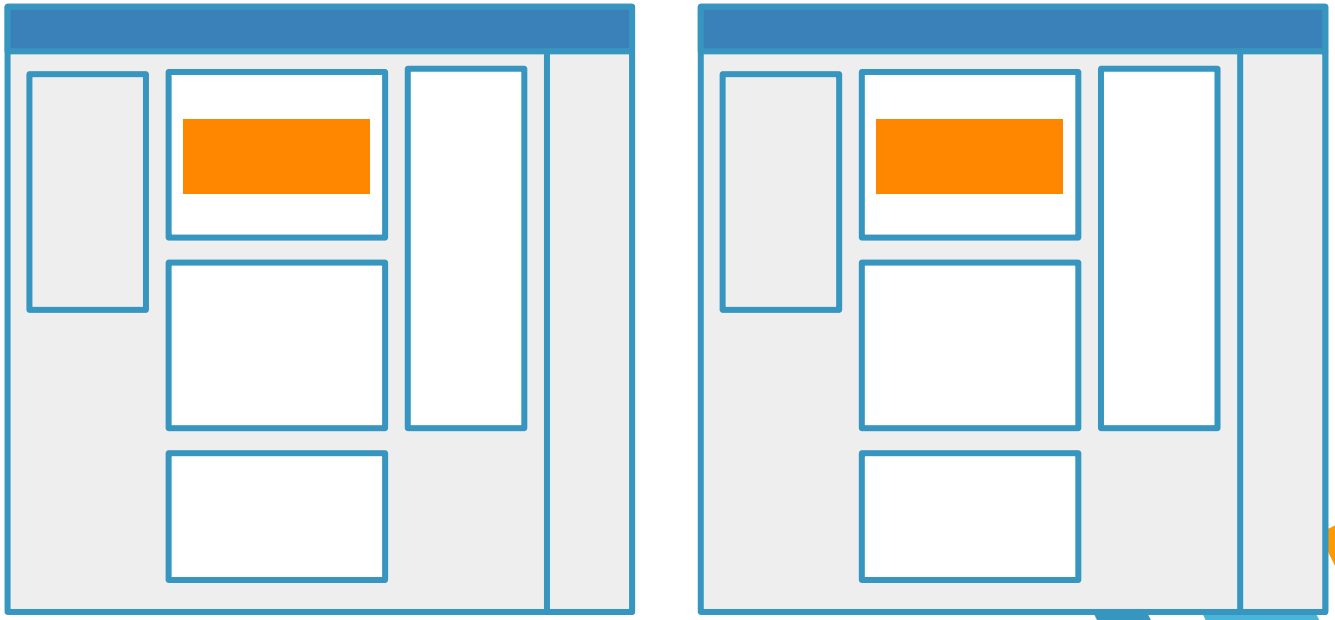
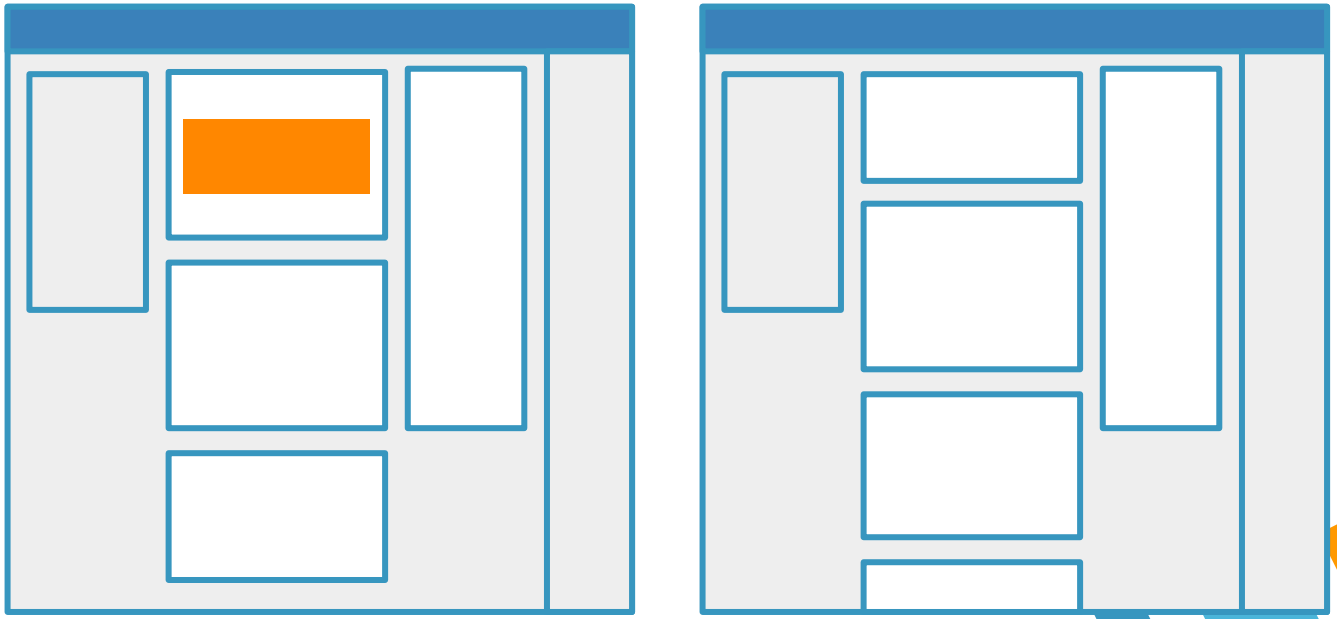
Consistent Storage or Scalable Storage – Why Not Both?

CONSISTENCY

Strong Consistency



Eventual Consistency



"Consistency in database systems refers to the requirement that **any given database transaction must change affected data only in allowed ways**."

Wikipedia

Consistency (database systems)

PickTix Concert Tickets schema

- User
  - id
  - name
- TicketOrder
  - id
  - user_id
  - concert_id
  - num_tickets
- Concert
  - id
  - name
  - tickets_left

Transactional Consistency

Begin transaction

- Read concert.tickets_left
- Create new invoice for 10 tickets
- Write tickets_left minus 10 from previous value

End transaction

Strong Consistency

If I write X, then read X (from anywhere), it'll include that write.

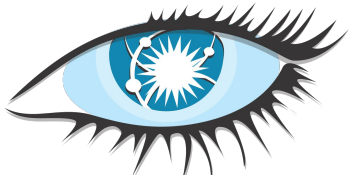
Eventual Consistency

If I write X, then read X, it might not have the update now, but eventually it'll have it.

Transactional Consistency

Write and read-write transactions across the database are atomic and isolated.

CASE STUDIES



Apache Cassandra

Built by Facebook

Open sourced 2008

Arguably 2nd most popular

Has schemas and SQL-like query language

Spanner

Built by Google

Paper published 2012

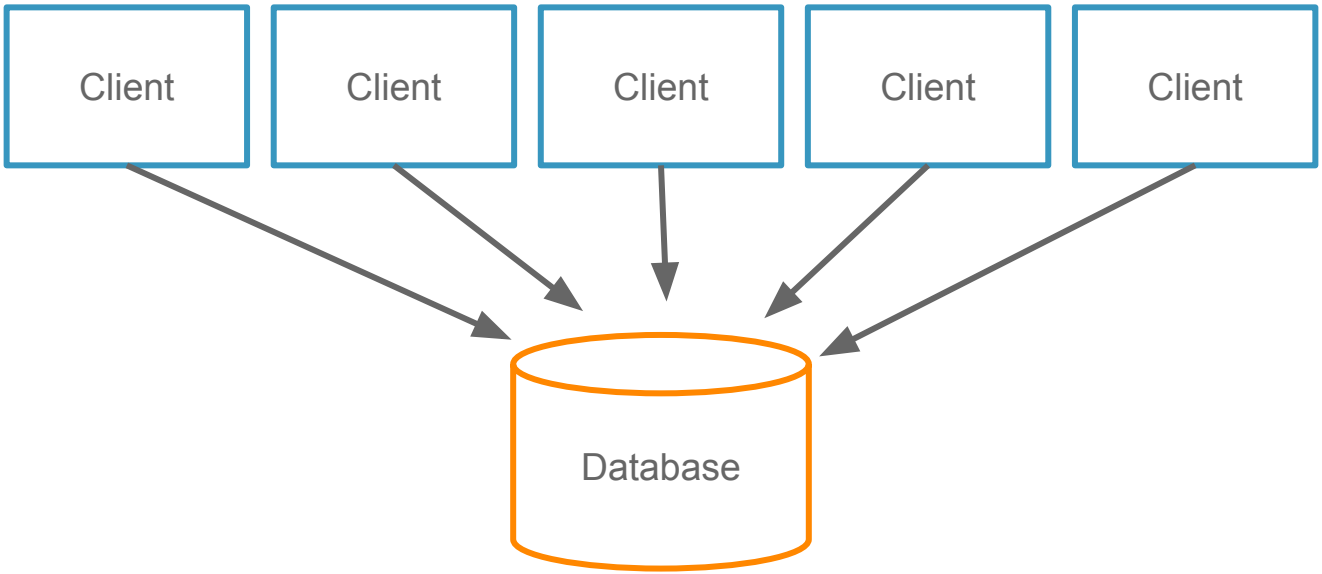
Recently released as beta

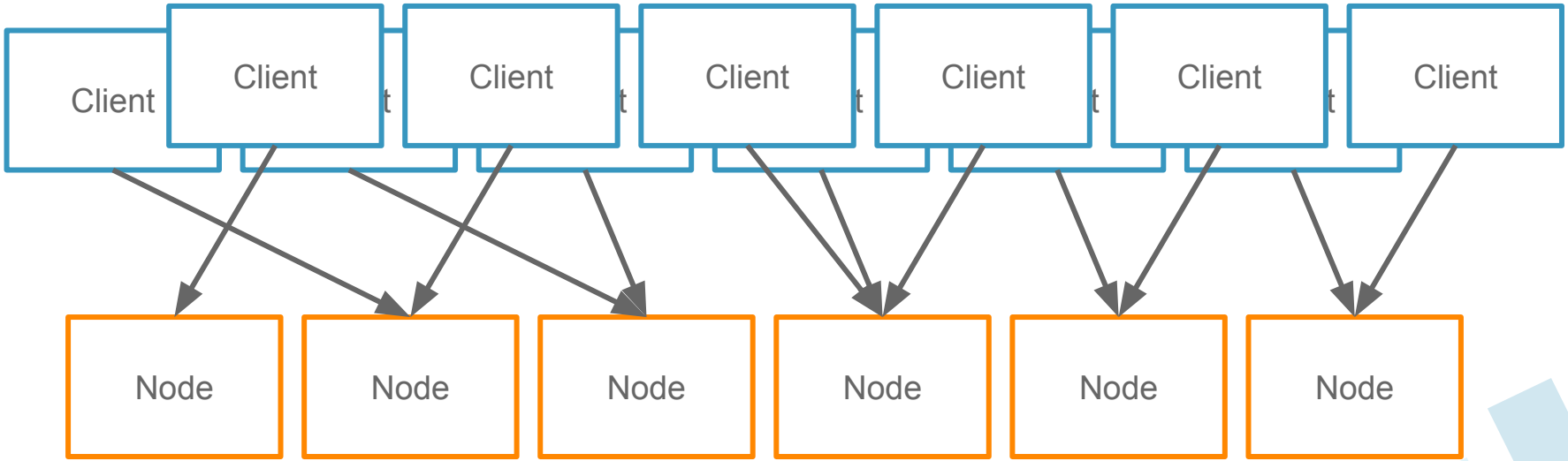
Schemas and SQL-like query language

1.

Cassandra

Representative non-relational storage system


Client
Client
Client
Client
Client
Database


Client
Client
Client
Client
Client
Client
Client
Node
Node
Node
Node
Node
Node

PickTix Concert Tickets schema

- User
  - id
  - name

- Concert
  - id
  - name
  - tickets_left

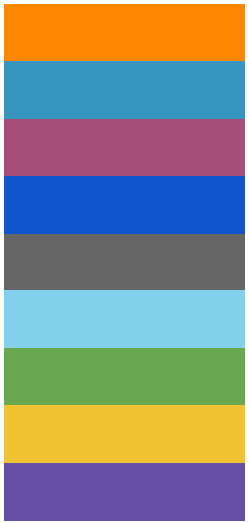
- TicketOrder
  - id
  - user_id
  - concert_id
  - num_tickets

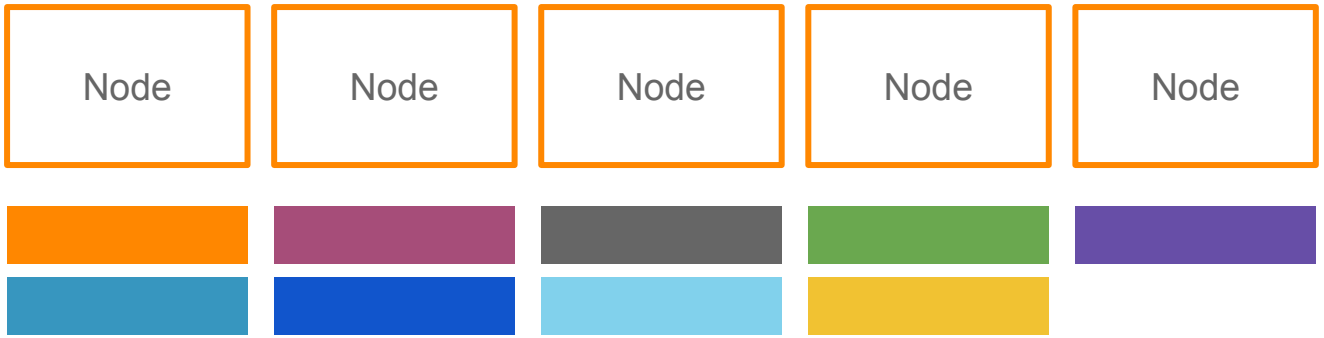
Partition: Ticket Orders

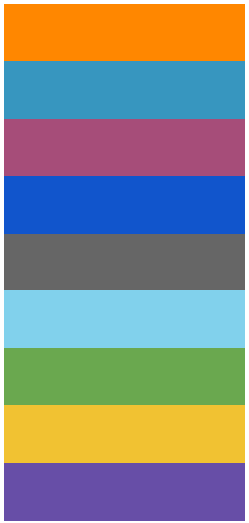
Primary key			
concert_ id (Partition key)	ticket_order_id	user_id	num_tickets
'adele'	1	'alice'	4
'adele'	2	'bob'	5
'gaga'	3	'alice'	1
'gaga'	4	'fred'	43

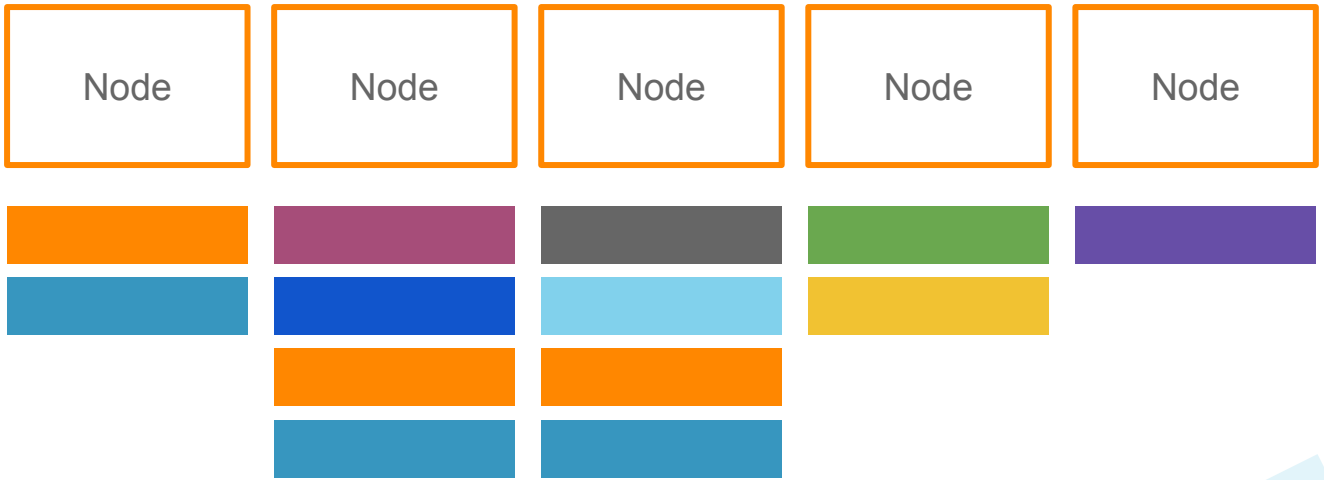
Partition: Ticket Orders

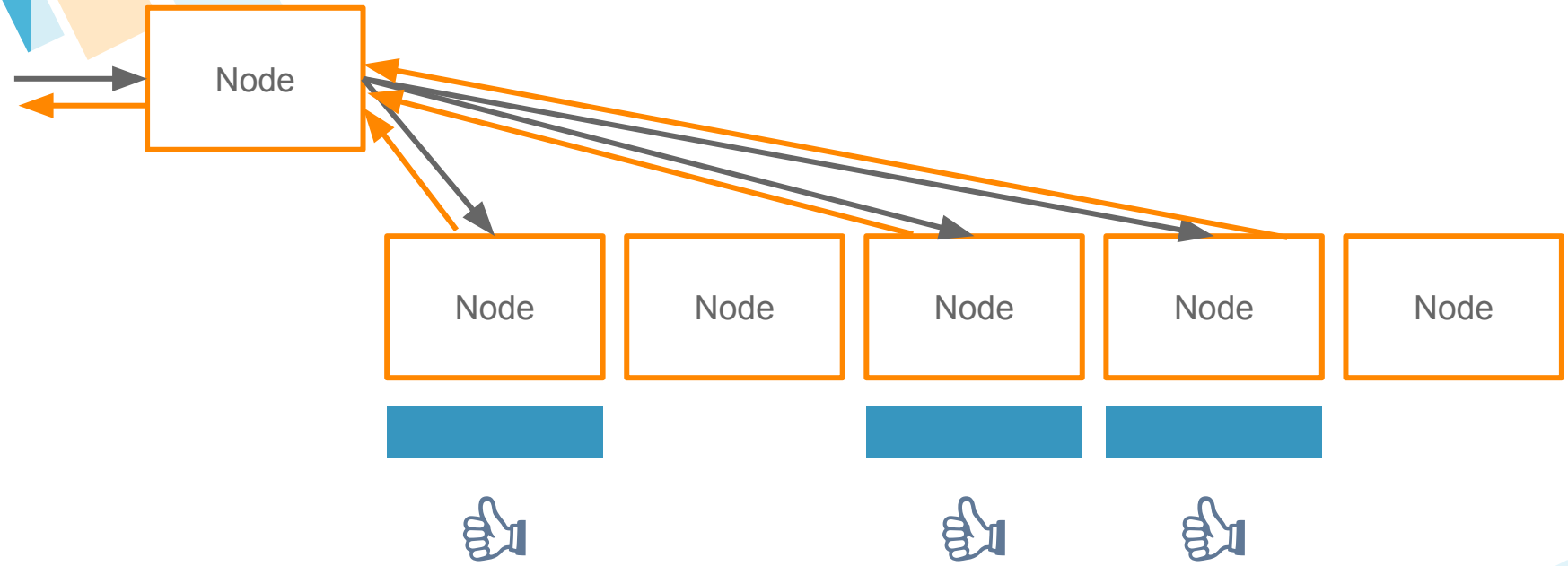
Primary key			
concert_ id (Partition key)	ticket_order_id	user_id	num_tickets
'adele'	1	'alice'	4
'adele'	2	'bob'	5
'gaga'	3	'alice'	1
'gaga'	4	'fred'	43



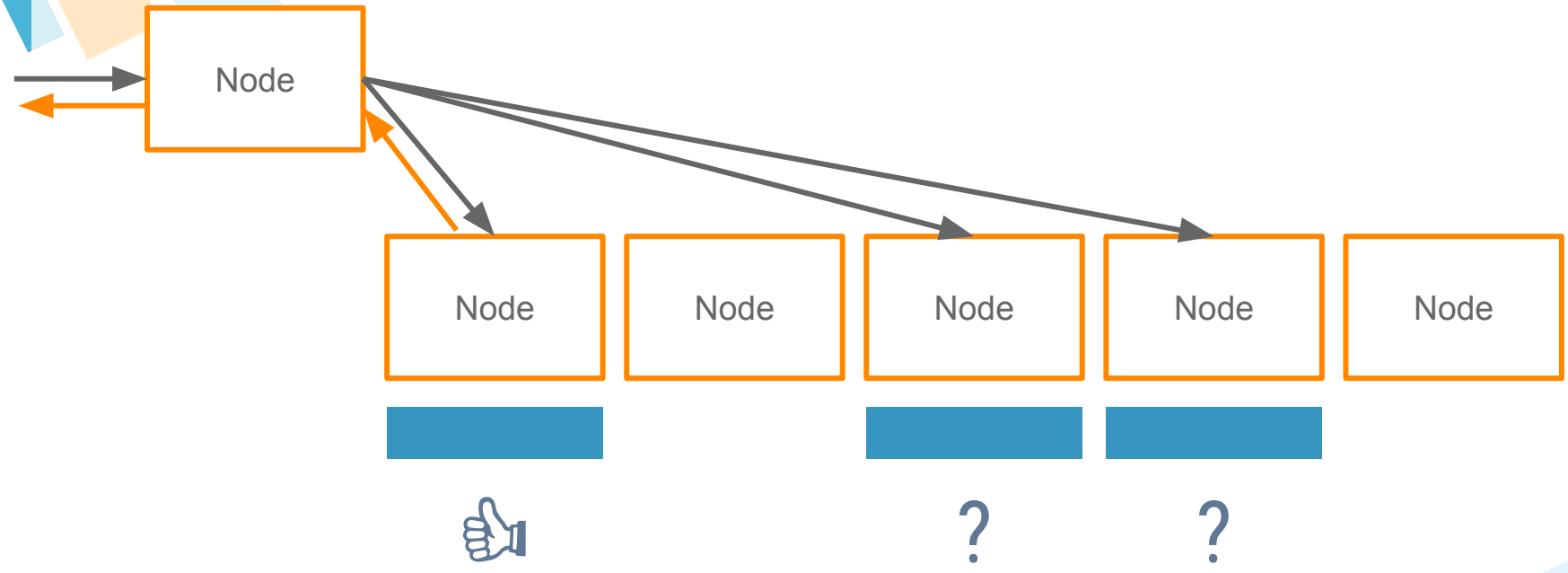

Node
Node
Node
Node
Node



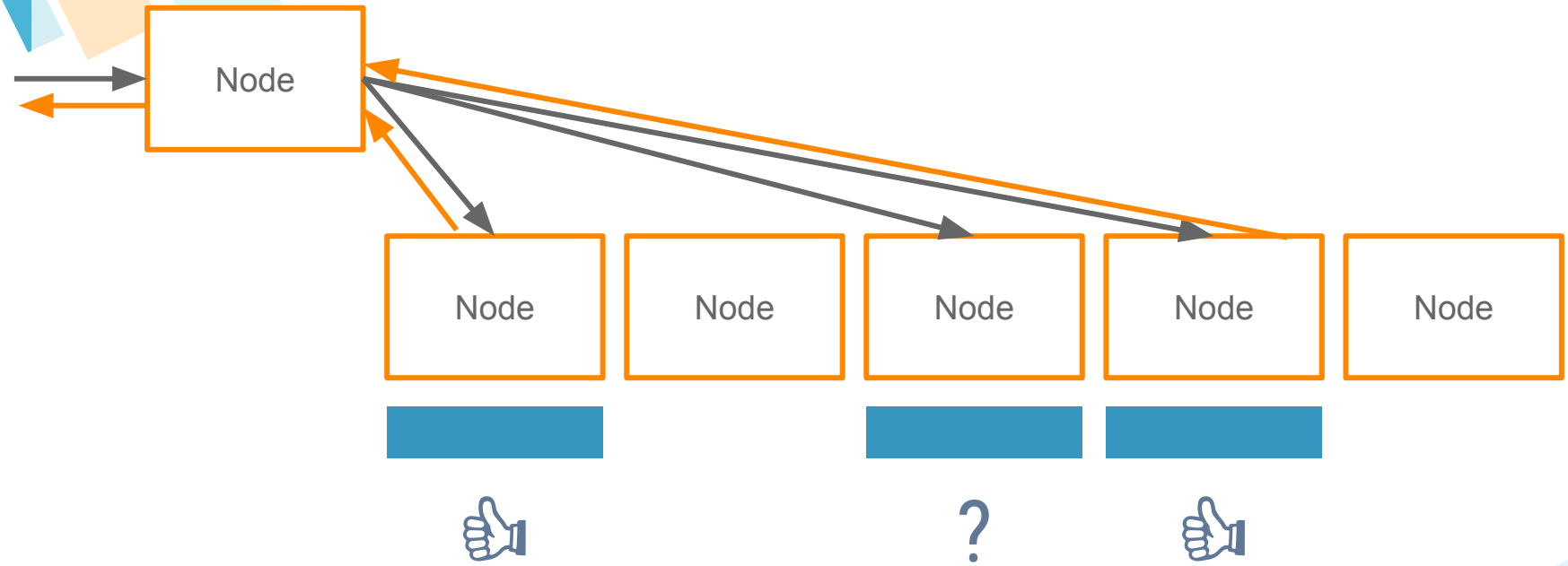

Node
Node
Node
Node
Node



Write Consistency Level: All



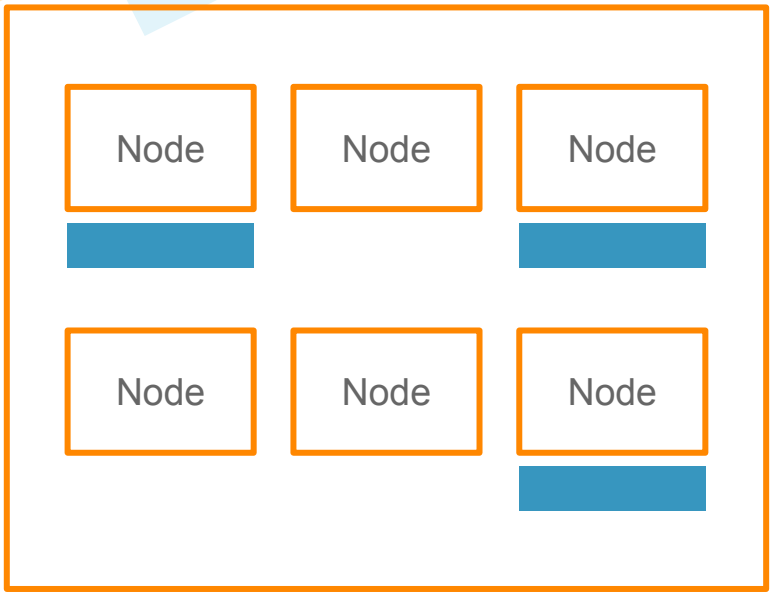
Write Consistency Level: One

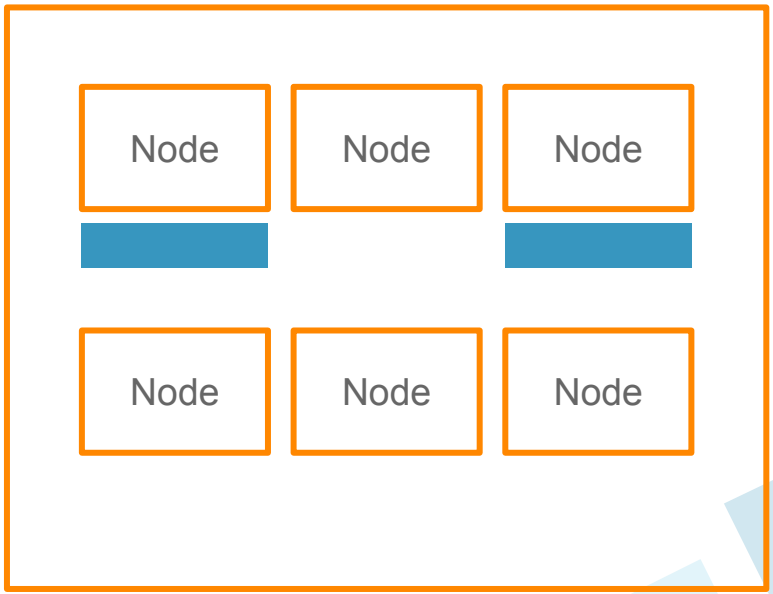


Write Consistency Level: Quorum

$$N_W + N_R > N$$

Global Replication





Global replication

Eventual Consistency

Yes.

Strong Consistency

Yes, iff (W + R > N) is satisfied.

Transactional Consistency

Limited operations within partitions.

Bob	Alice
Add pending friend request to Alice	Add pending friend request from Bob
Check pending friend request Add Bob to friends Delete pending friend request	Add Alice to friends Delete pending friend request

Bob	Alice
Add pending friend request to Alice	Add pending friend request from Bob
Check pending friend request Cancel pending friend request Add Bob to friends Delete pending friend request	Add Alice to friends Delete pending friend request

Development Costs

- Choose partition keys wisely
  - Include any data which must be kept consistent with it
  - Don't let it get too big
- Duplicate (denormalise) data
- Background cleanup tasks

2.

Spanner

Representative scalable relational storage system

Can I use Spanner now?

It works! It Scales! It's battle-tested! It's ready to be used, except:

- Google Cloud Platform only
- Beta (no SLA)
- Single region only
- Expensive


@jlawrence124 /r/wallpaper/

Read-write/write-write consistency

Alice wants to accept a pending friend request from Bob

1. Check that the friend request is still valid
2. Add Alice as a friend to Bob
3. Add Bob as a friend to Alice

No risk of Bob cancelling the friend reqest between step 1 and 2/3

The consistency guarantees we want

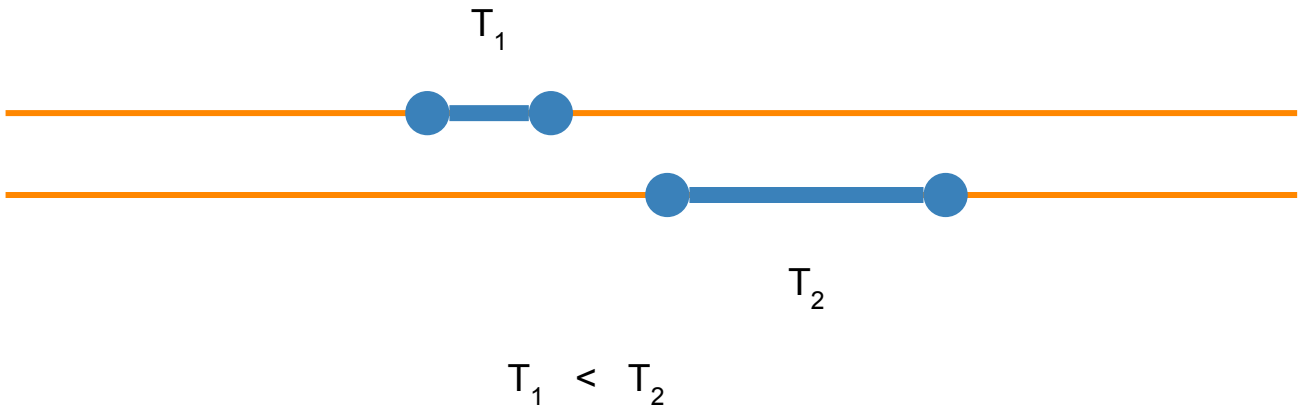
Write, write-write and read-write transactions

- Atomic
- Isolated

Read and Read-read transactions

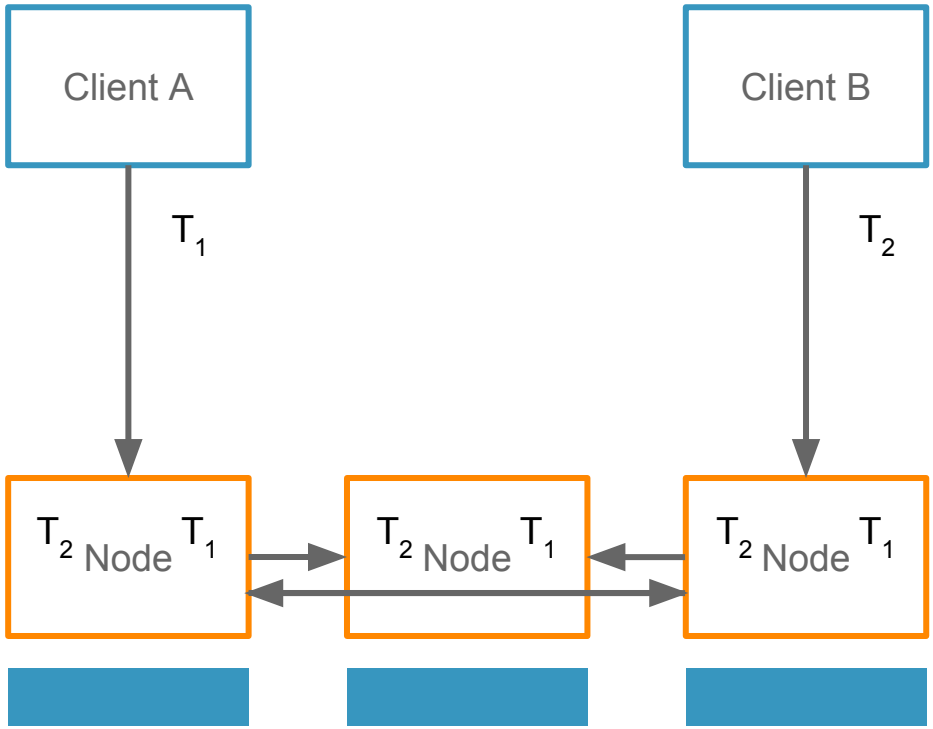
- Never see partial writes
- If writes depend on each other, never see them out of order

Linearizabilty



$T_1 < T_2$

Cassandra Write Timestamps



Clock drift

A's clock is slightly ahead

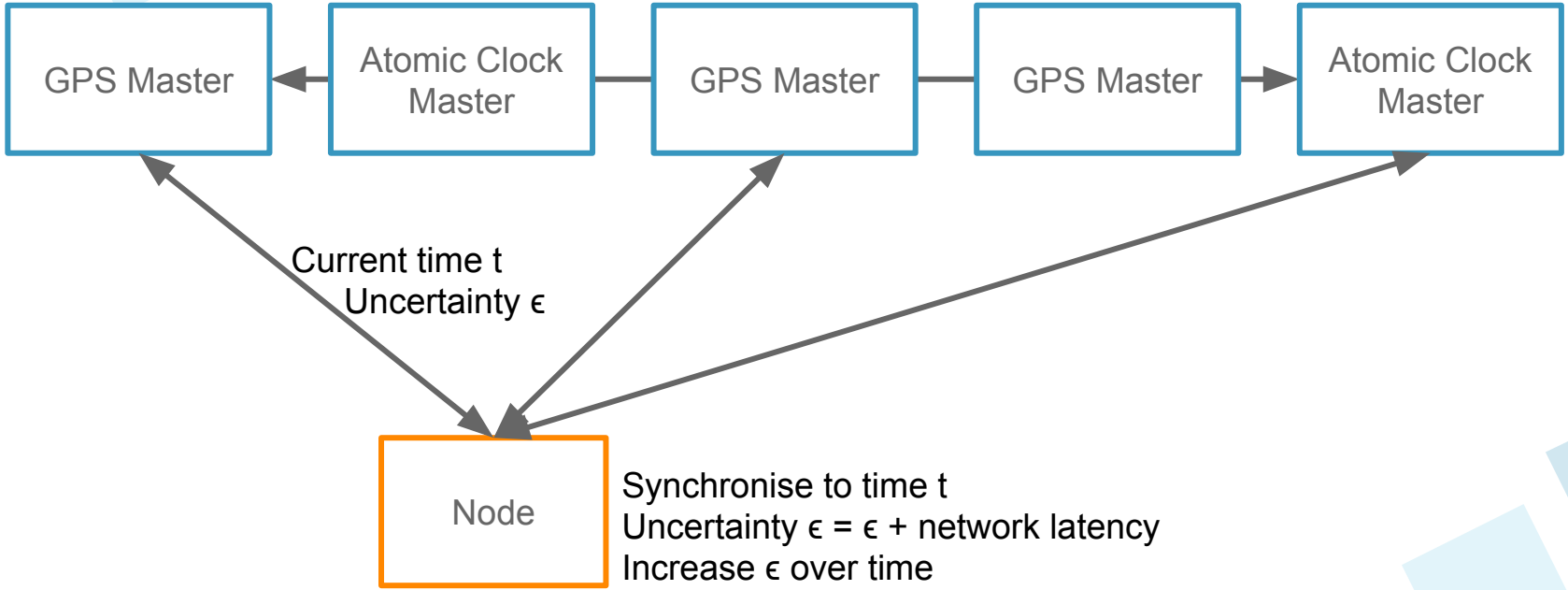
B's clock is slightly behind

A writes with timestamp T_1 (Client A generated timestamp)

B reads at timestamp T_1

B writes at timestamp T_2 (Client B generated timestamp)

$T_2 < T_1$ so B's write is *before* A's

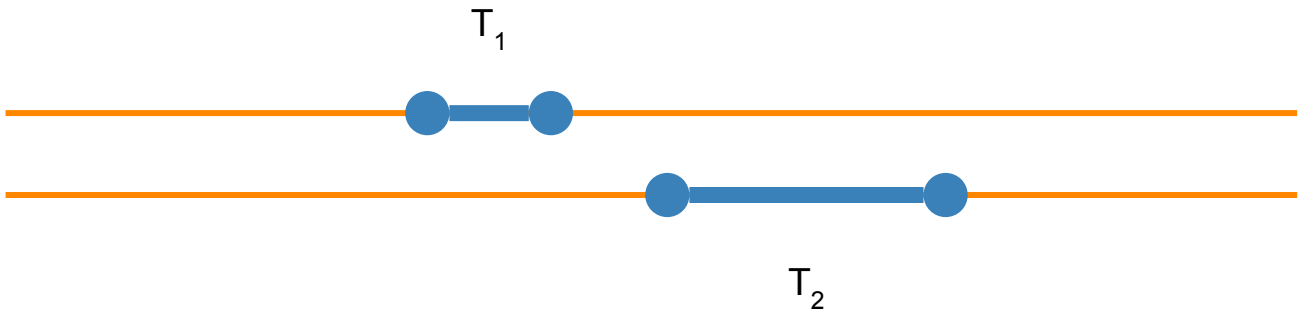

GPS Master
Atomic Clock Master
GPS Master
GPS Master
Atomic Clock Master
Current time t
Uncertainty ϵ
Node
Synchronise to time t
Uncertainty ϵ = ϵ + network latency
Increase ϵ over time

TrueTime

TrueTime

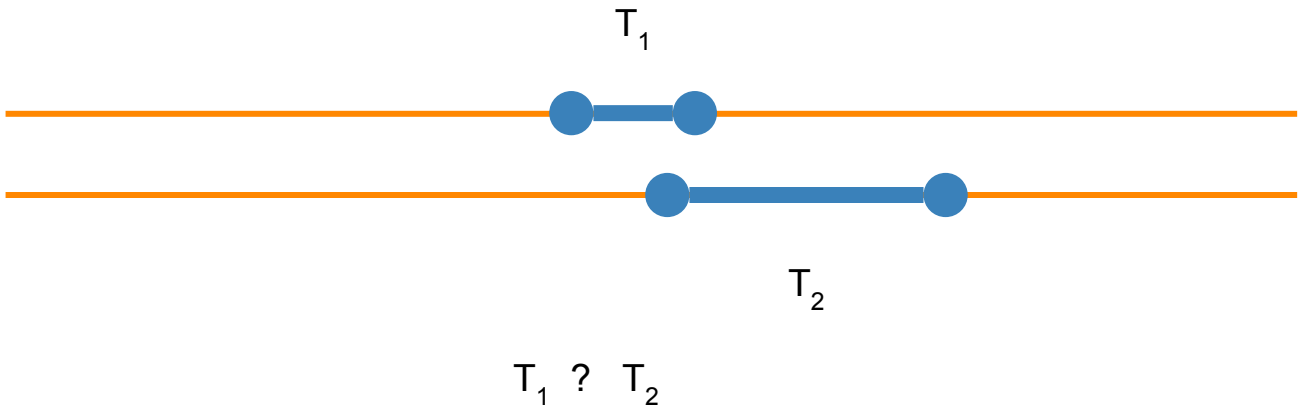
TT.now() = [earliest, latest]

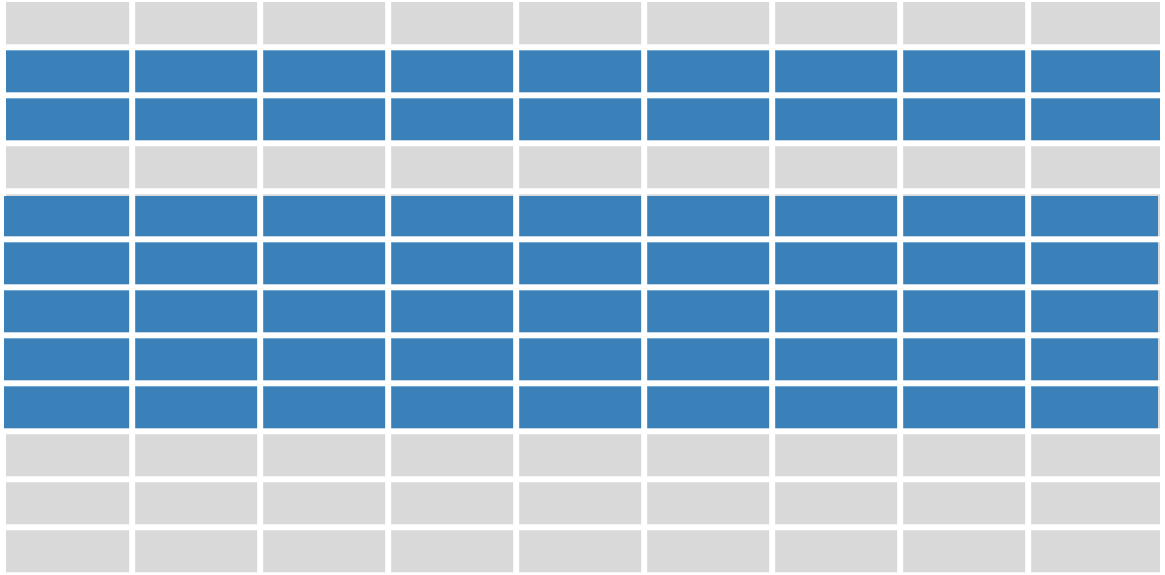
Linearizabilty with TrueTime



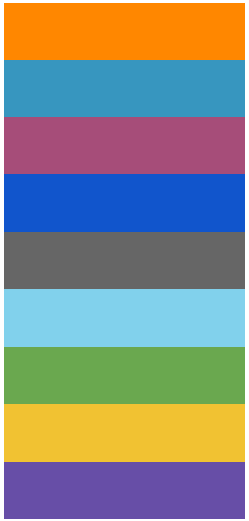
1. Transaction starts
2. Assign transaction timestamp T_1 to be TT.latest
3. Prepare transaction
4. Wait for T_1 to be earlier than TT.earliest
5. Commit transaction
6. Return success

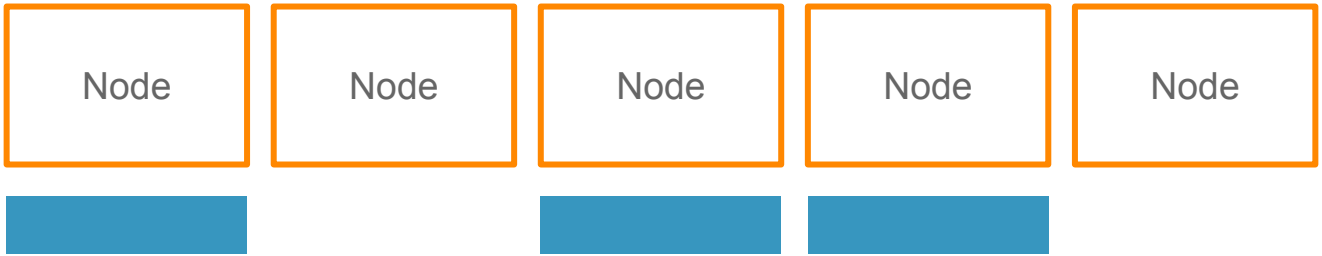
Linearizabilty



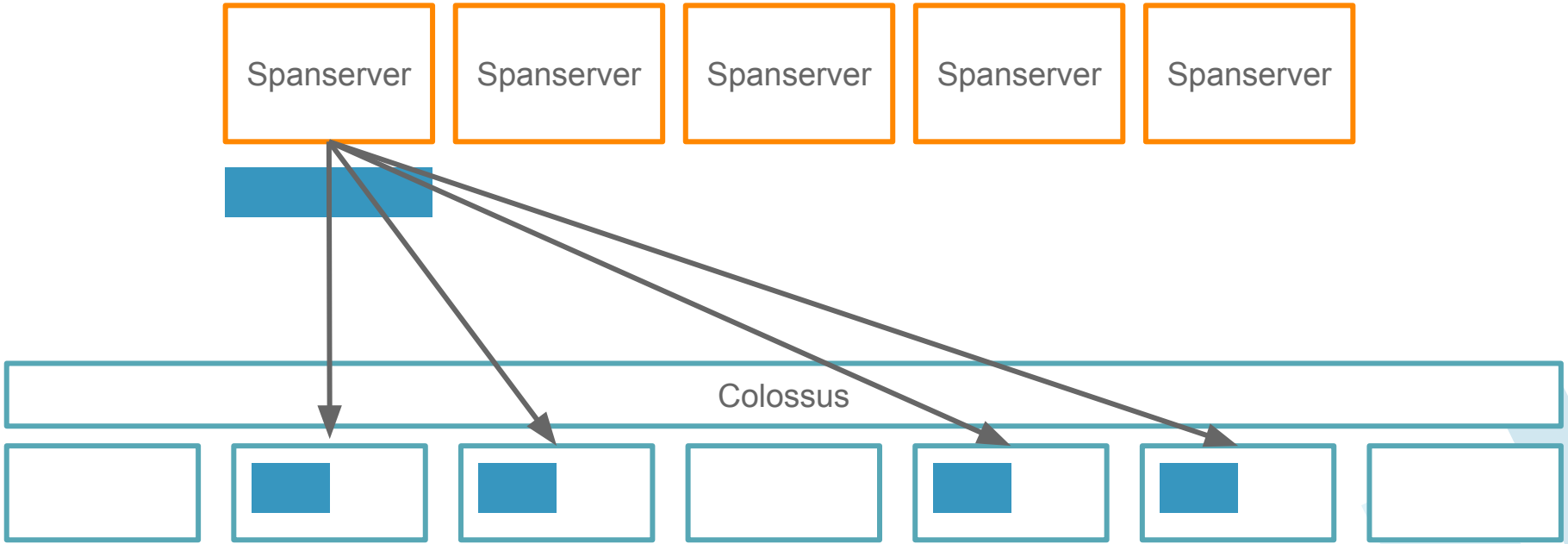


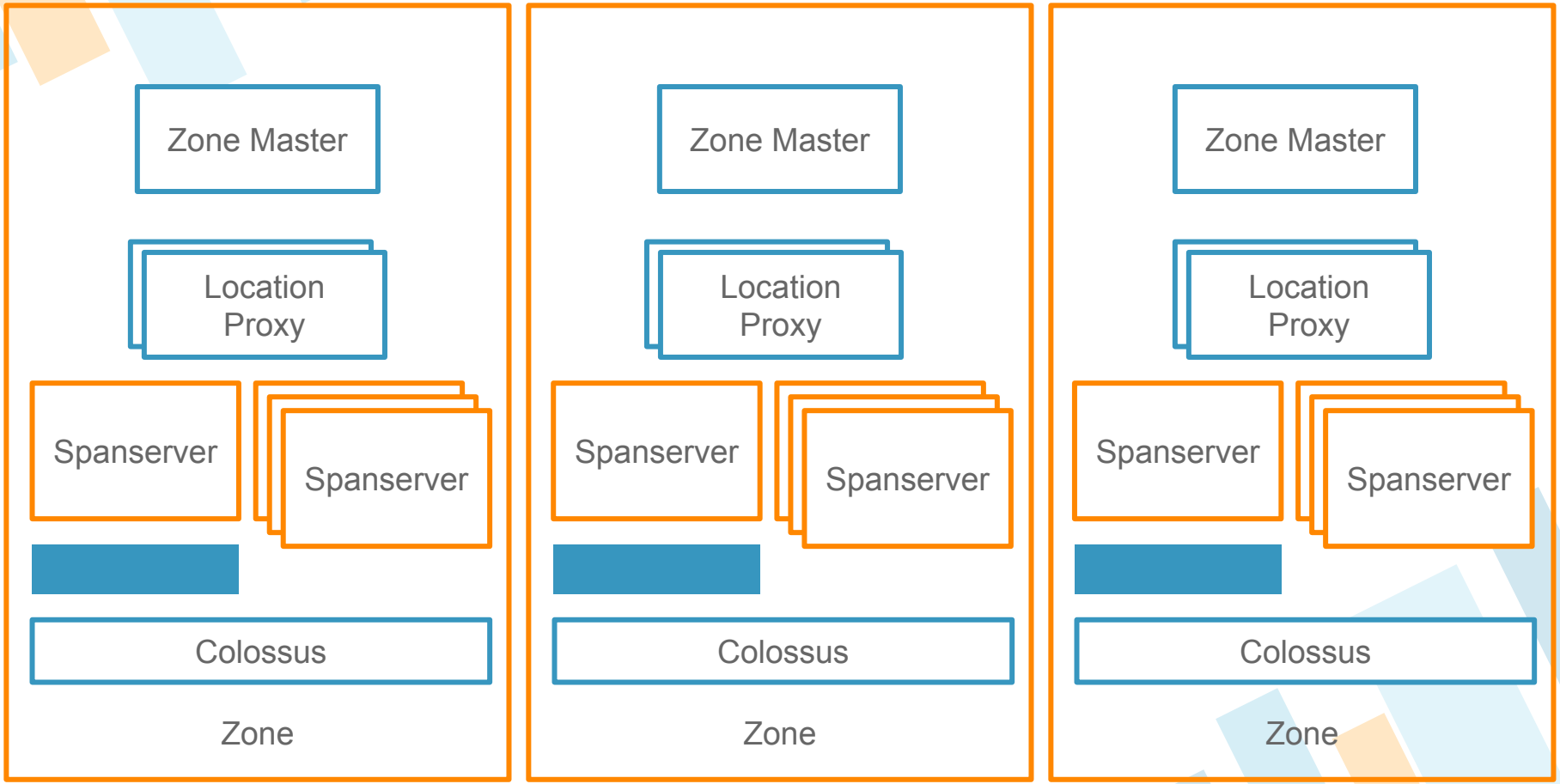
Spanner Partitions (tablets)

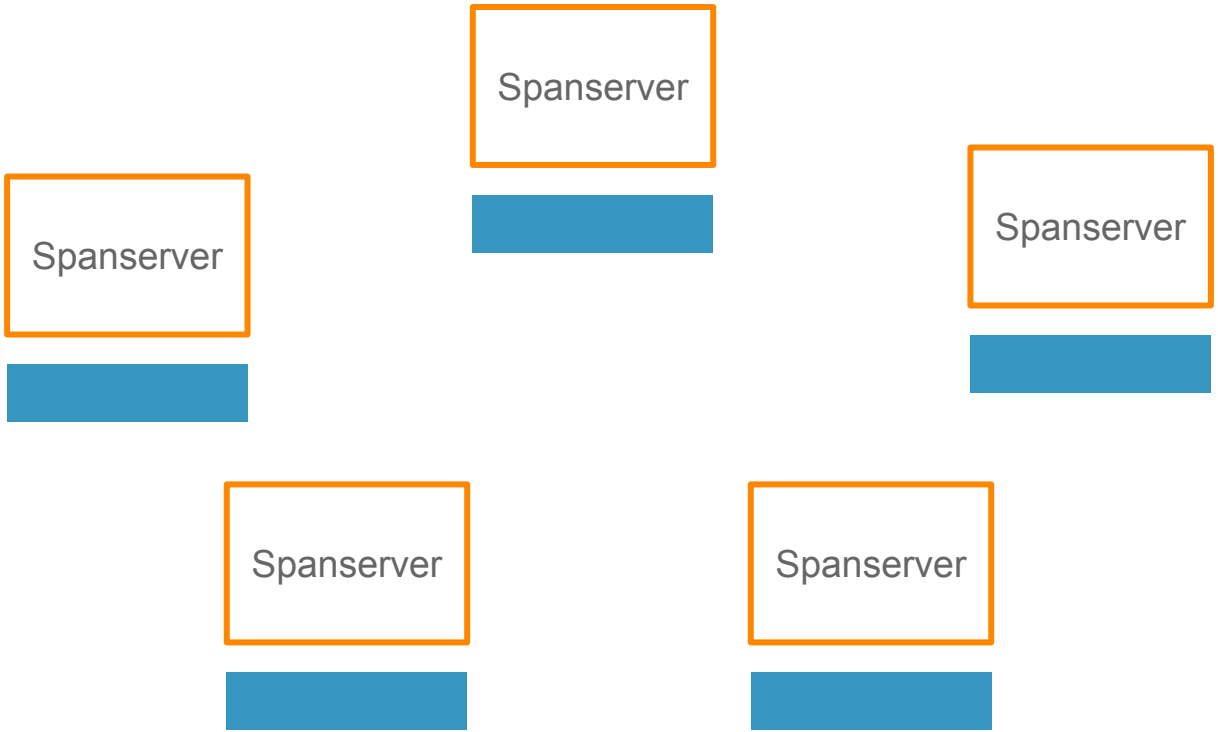




Cassandra partition replication


Spanserver
Spanserver
Spanserver
Spanserver
Spanserver
Colossus


Zone Master
Location Proxy
Spanserver
Spanserver
Colossus
Zone
Zone Master
Location Proxy
Spanserver
Spanserver
Colossus
Zone
Zone Master
Location Proxy
Spanserver
Spanserver
Colossus
Zone


Spanserver
Spanserver
Spanserver
Spanserver
Spanserver

The consistency guarantees we want

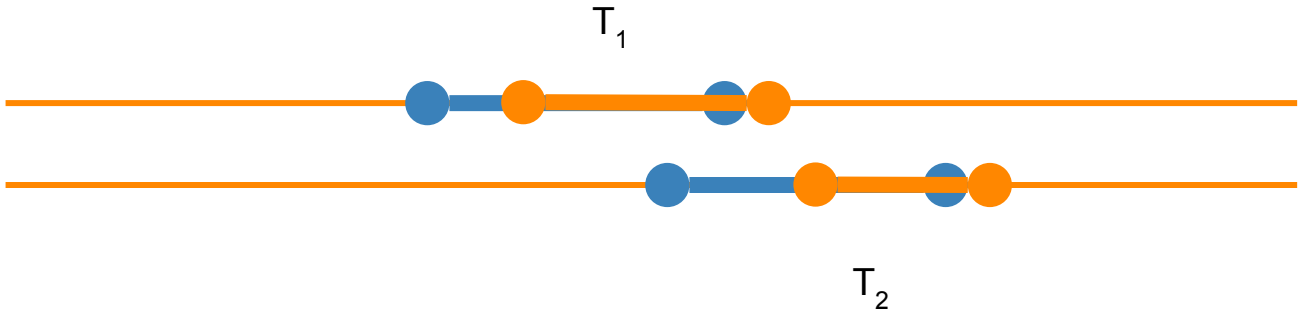
Write and read-write transactions

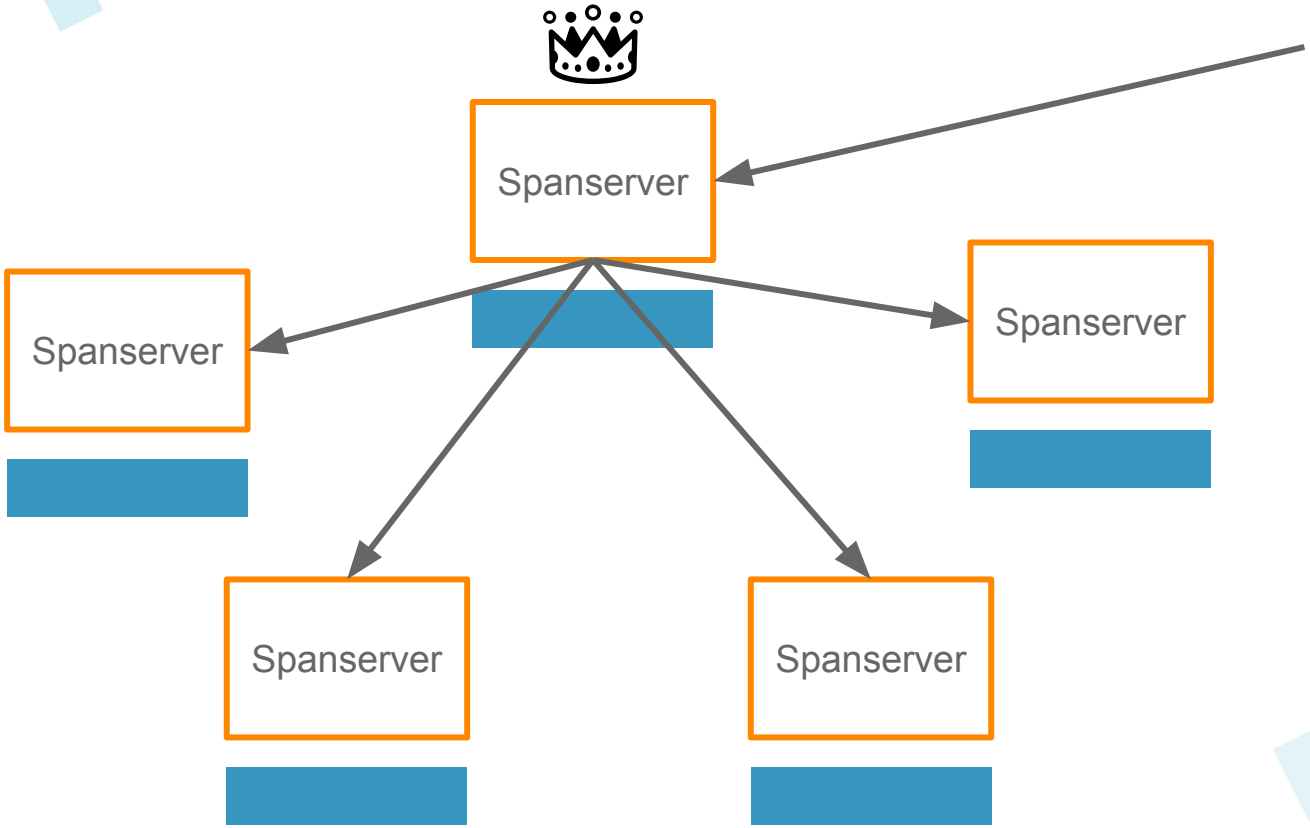
- Atomic
- Isolated

Read-read transactions

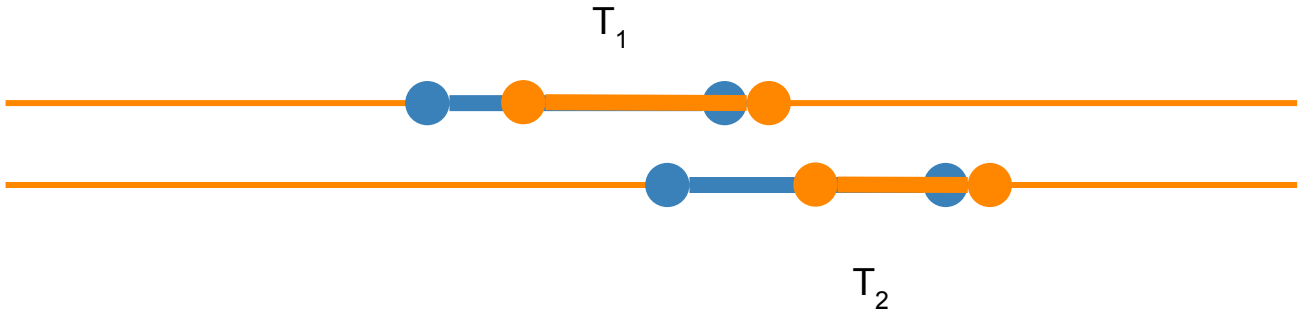
- Never see partial writes
- If writes depend on each other, never see them out of order

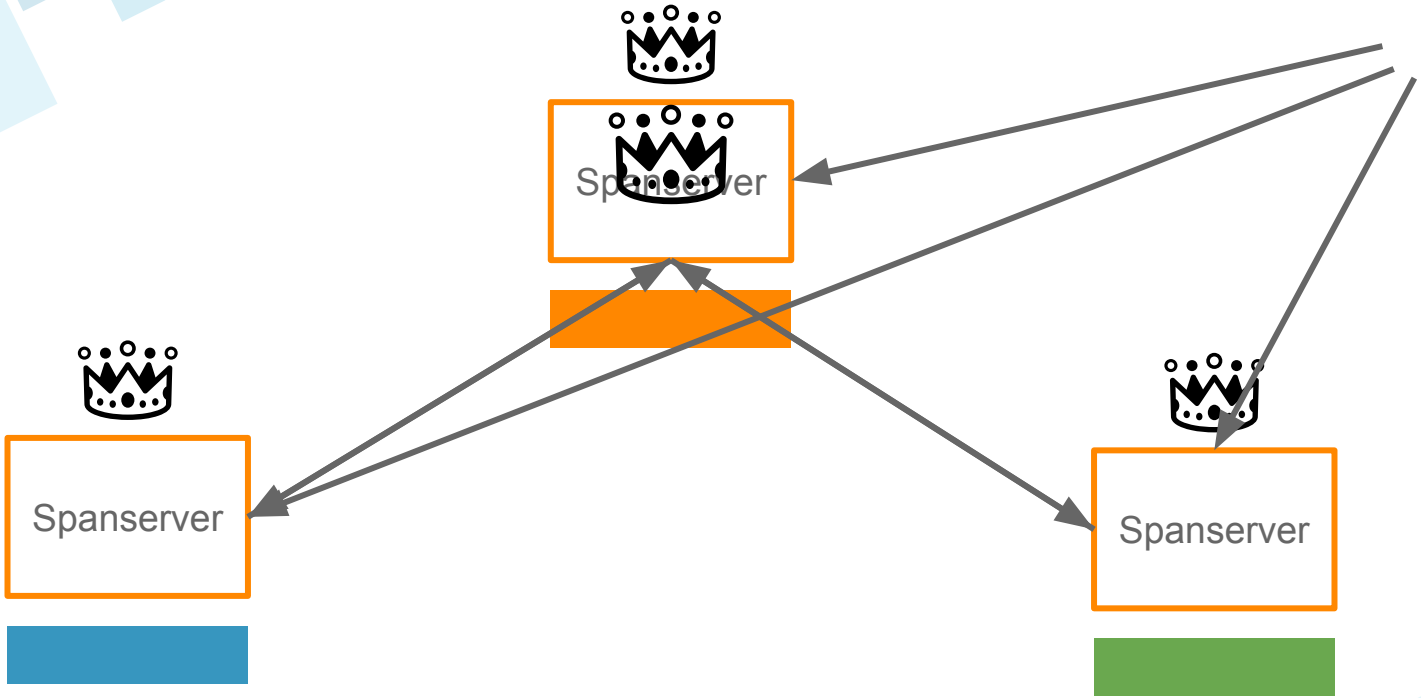
Transactions that conflict




Spanserver
Spanserver
Spanserver
Spanserver
Spanserver

Transactions that conflict





Transactions across paxos groups (tablets)

The consistency guarantees we want

Write and read-write transactions

- Atomic
- Isolated

Read and Read-read transactions

- Never see partial writes
- If writes depend on each other, never see them out of order

Reads

- Consistent reads at a timestamp
- Strongly consistent reads
- Time-bounded staleness reads

Conclusions

- Consistency guarantees make happy developers.
- Transactional consistency at scale is feasible.
- Perfect for high-read, low-write, consistency-sensitive data.
- Consider using Spanner, if it works for you.
- Keep a look out for the next generation! Or build it!

THANKS!

Any questions?

I have copies of the Spanner, Cassandra and related papers here.

You can find me at

- katiebell.net
- @notsolonecoder