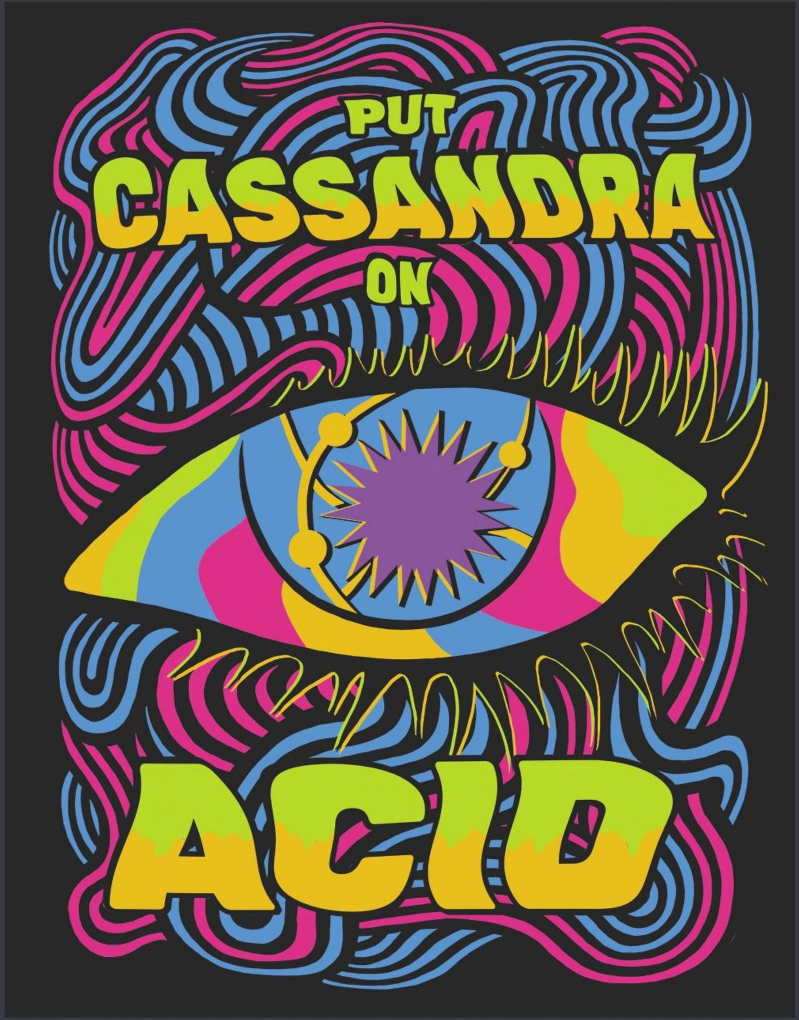
DataStax

ACID transactions in Apache Cassandra®

Cassandra Day - Version 0.1.0 BETA


PUT
CASSANDRA
ON
ACID

Background

If you haven’t been on the dev list for a while

Cassandra is great, but…

I need transactions

Lightweight Transactions

Cassandra 2.0 - 2013

Pros

- Paxos - Established protocol
- CAS functionality
- Guarantees exclusive operation

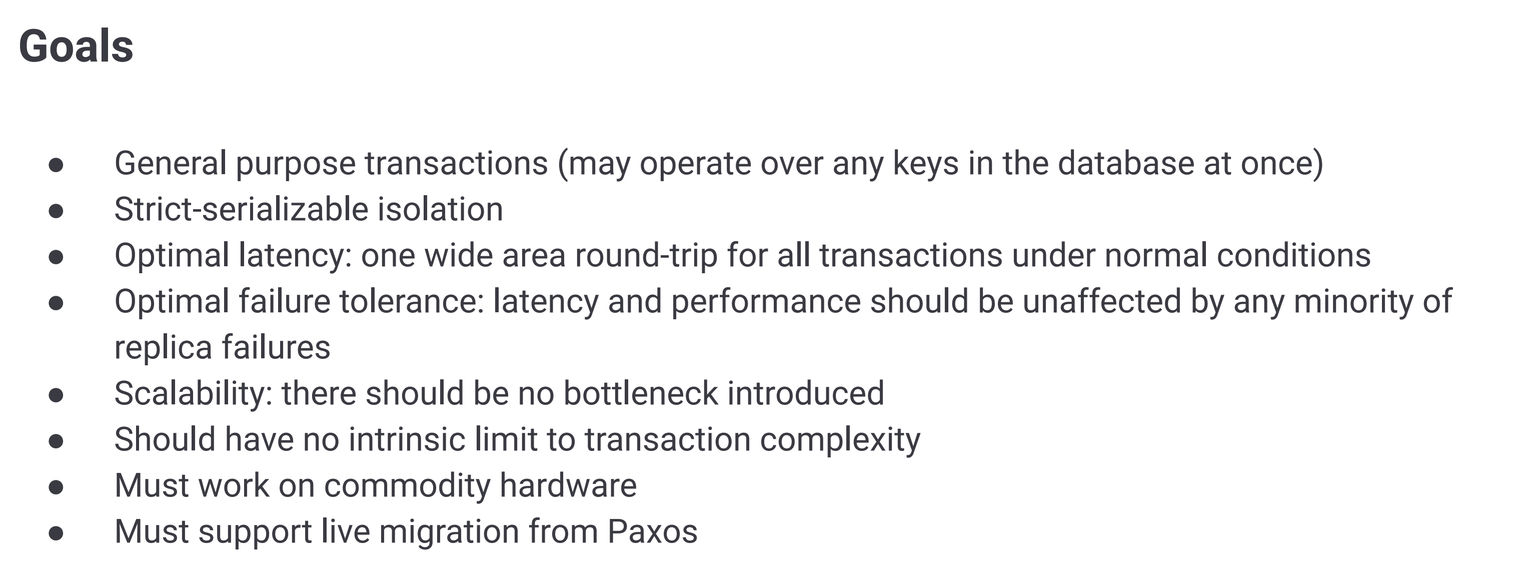
Cons

- Only one partition
- Serialized operations - nope
- Was slower (Fixed in V2)

```
INSERT INTO cycling.cyclist_name (id, lastname, firstname)
VALUES (4647f6d3-7bd2-4085-8d6c-1229351b5498, 'KNETEMANN', 'Roxxane')
IF NOT EXISTS;
```

```
UPDATE cycling.cyclist_name
SET firstname = ‘Roxane’
WHERE id = 4647f6d3-7bd2-4085-8d6c-1229351b5498
IF firstname = ‘Roxxane’;
```

CEP-15: General Purpose Transactions



Goals

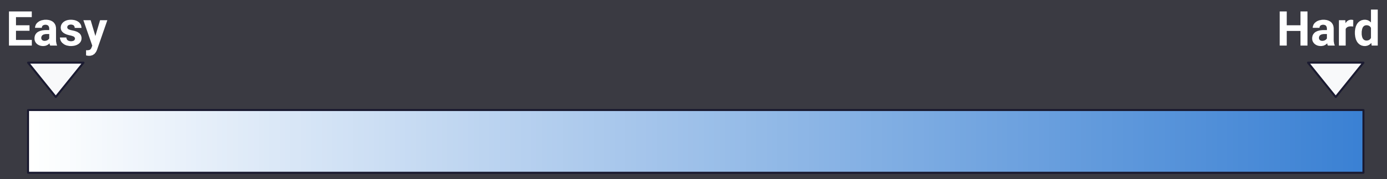
- General purpose transactions (may operate over any keys in the database at once)
- Strict-serializable isolation
- Optimal latency: one wide area round-trip for all transactions under normal conditions
- Optimal failure tolerance: latency and performance should be unaffected by any minority of replica failures
- Scalability: there should be no bottleneck introduced
- Should have no intrinsic limit to transaction complexity
- Must work on commodity hardware
- Must support live migration from Paxos

TL;DR ACID Transactions in Cassandra

Developer Impact

The people who count

Cassandra Relationship With Developers



Easy

Hard

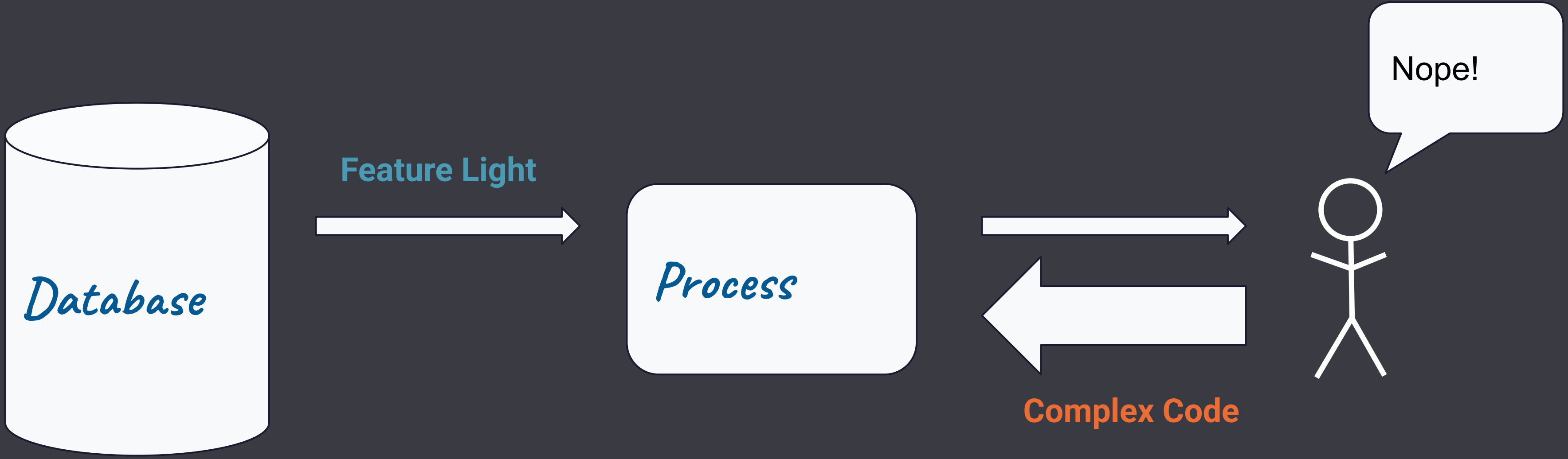
- Scale
- Resilience
- Distribution

Built-in

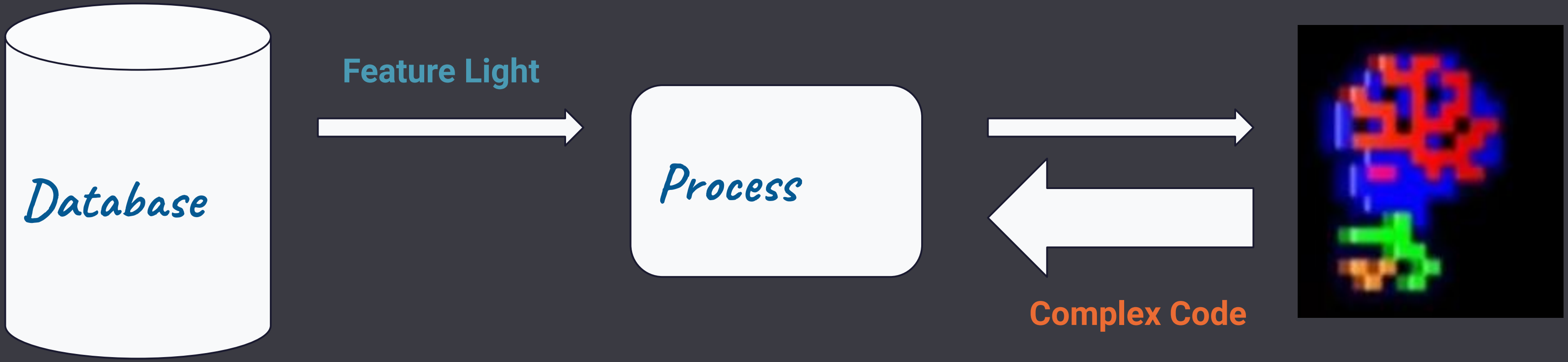
- Atomicity
- Serialized changes
- Complicated State

Data Modeling & Code

Developer Experience



Developer Experience



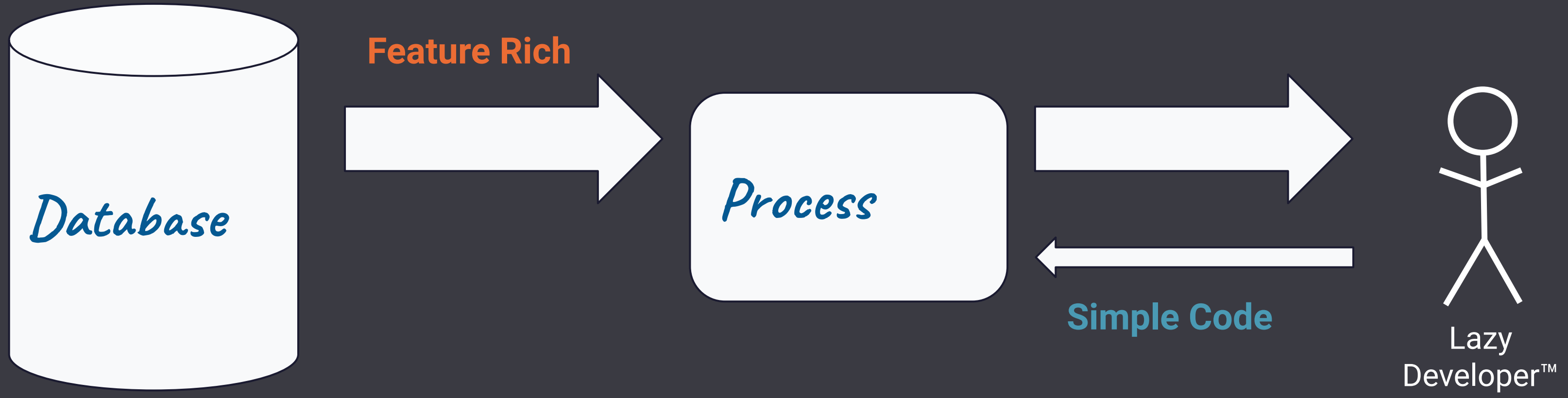
Database

Feature Light

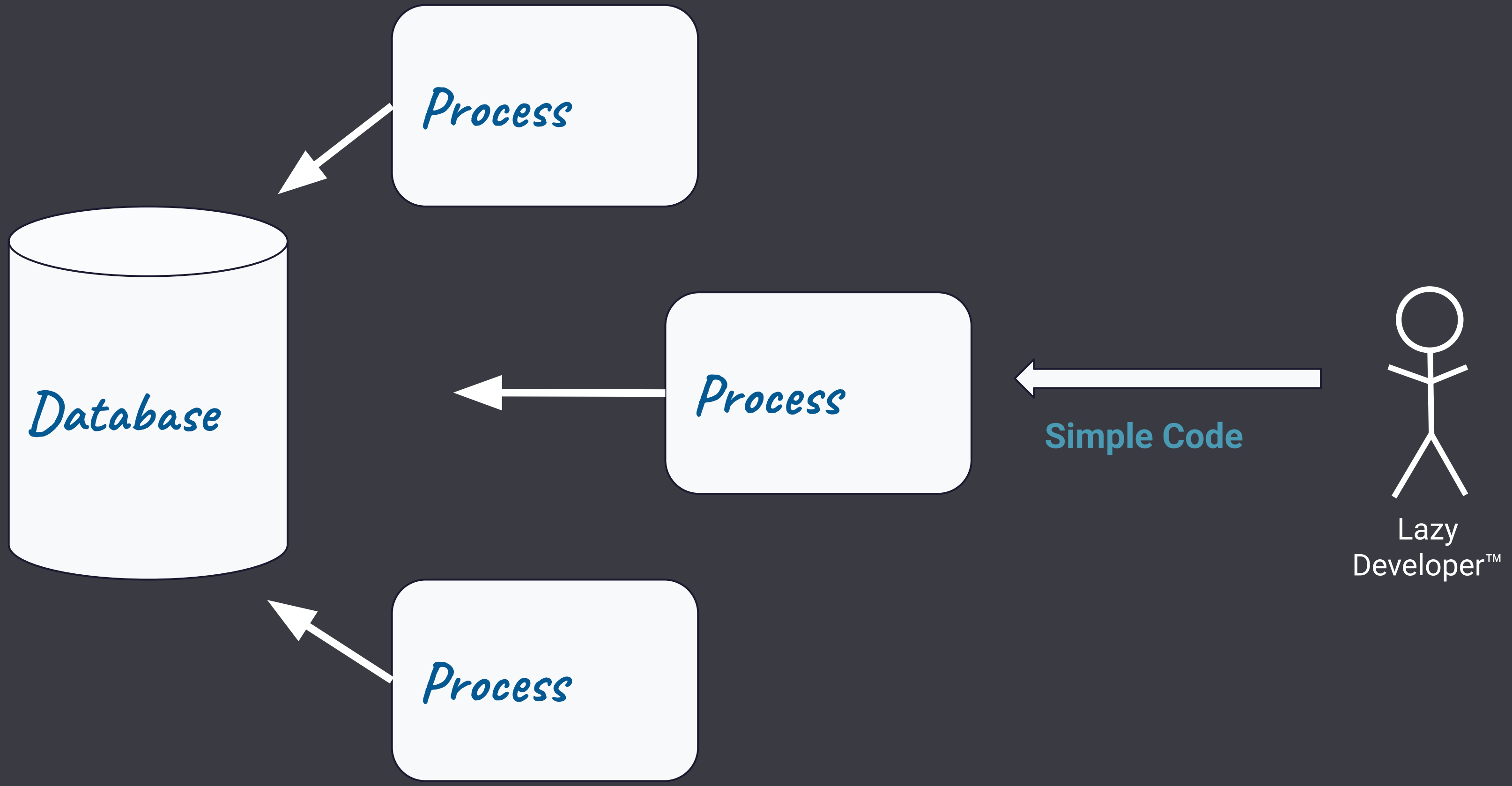
Process

Complex Code

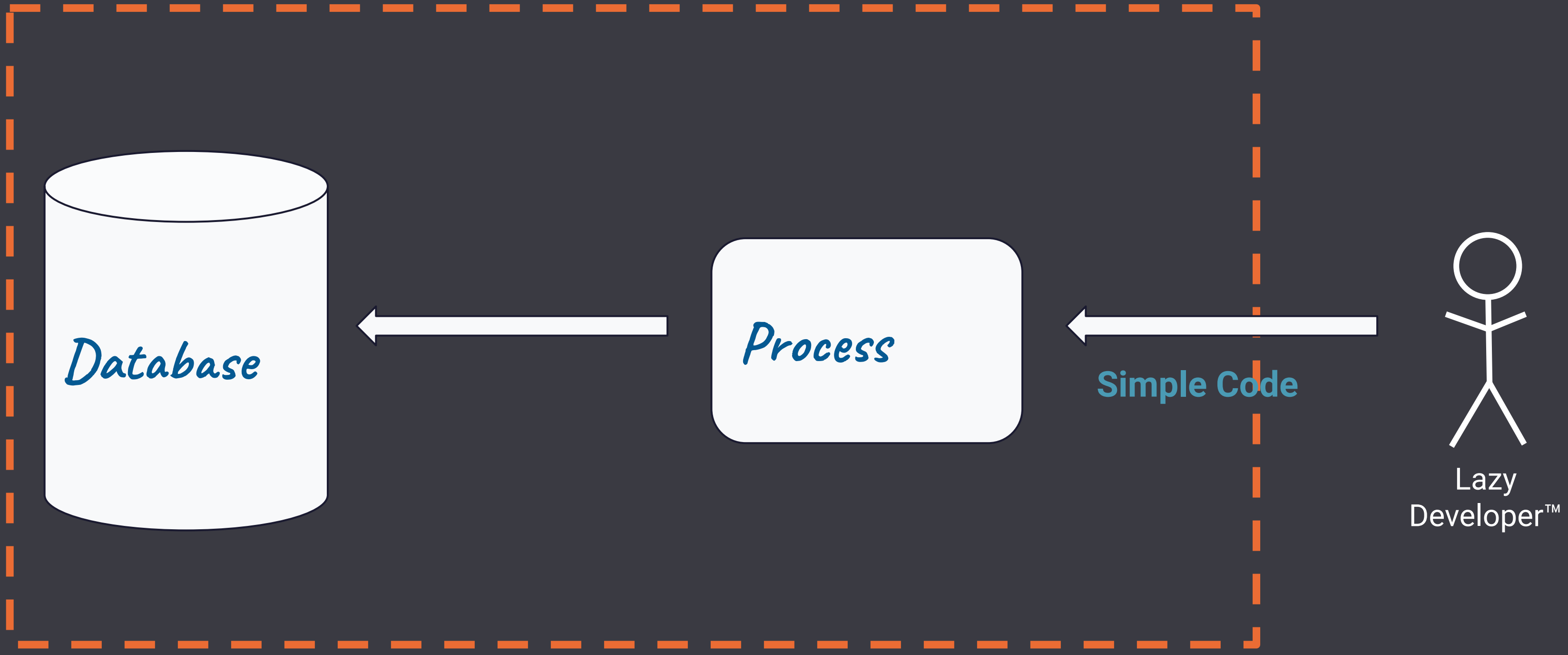
Developer Experience



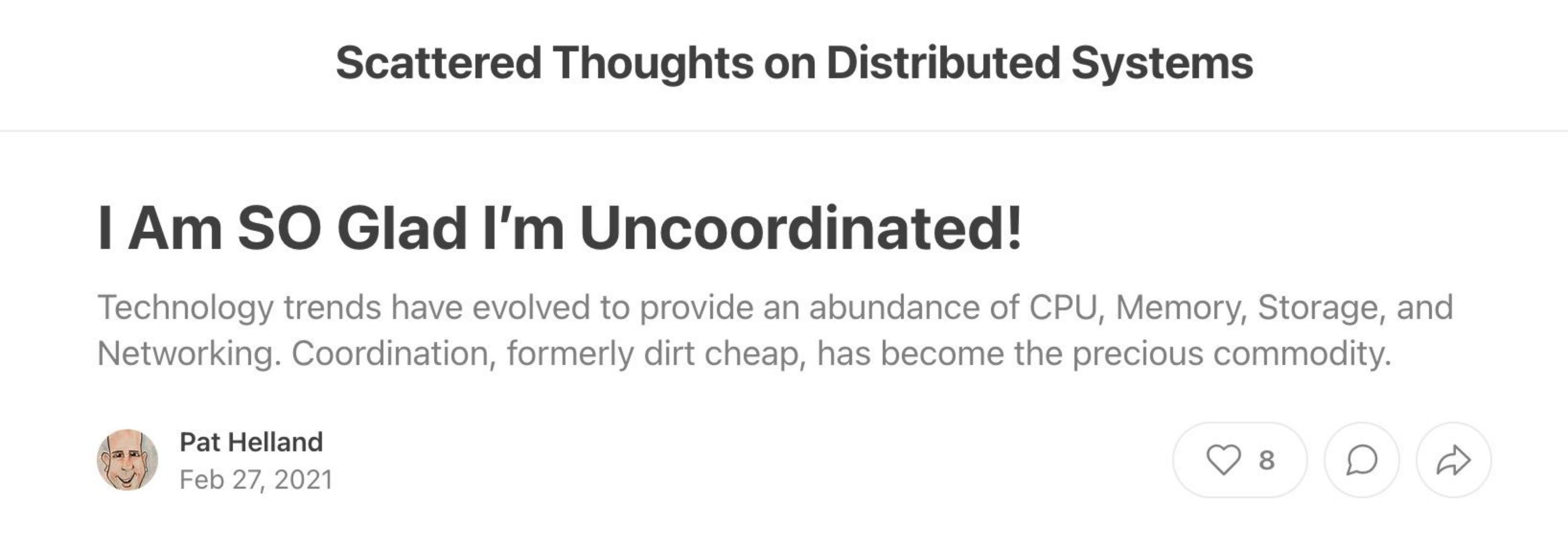
Observer Reference Frame – Non Exclusive



Observer Reference Frame – Exclusive



https://pathelland.substack.com/p/i-am-so-glad-im-uncoordinated



Scattered Thoughts on Distributed Systems

I Am SO Glad I'm Uncoordinated!

Technology trends have evolved to provide an abundance of CPU, Memory, Storage, and Networking. Coordination, formerly dirt cheap, has become the precious commodity.

Pat Helland
Feb 27, 2021

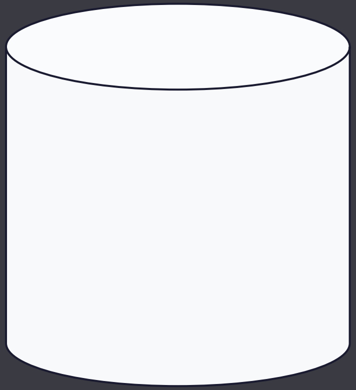
ACM Keynote Video: https://www.youtube.com/watch?v=E6JUA_XH_tE

Cost of Latency

CPU	Cheap
Network	Cheap
Memory	Cheap
Storage	Cheap
Coordination	Expensive(last frontier)

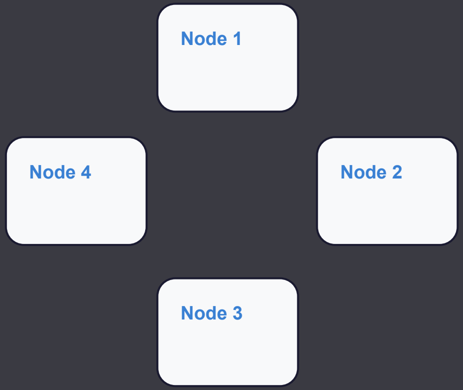
Cost of Transaction

Single System



Cheap

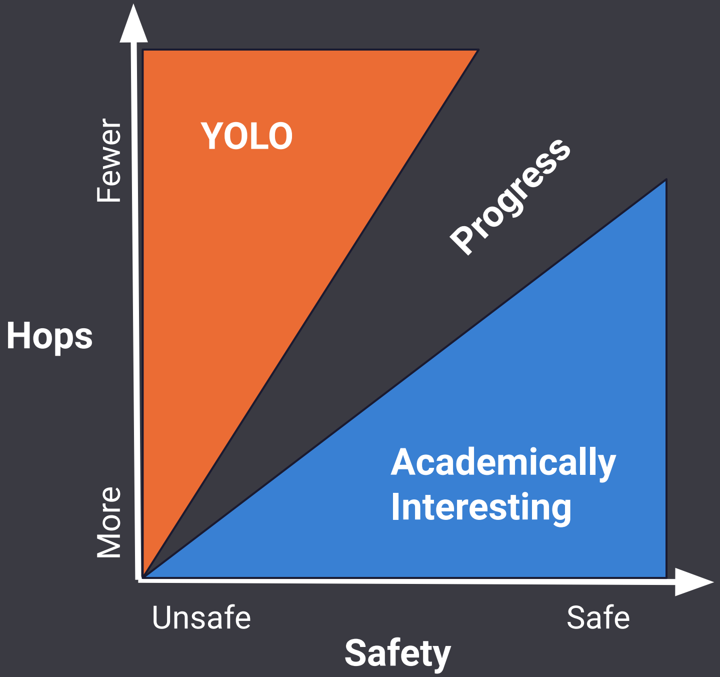
Distributed System



Expensive

Coordination
Round-Trips

aka **Hops**

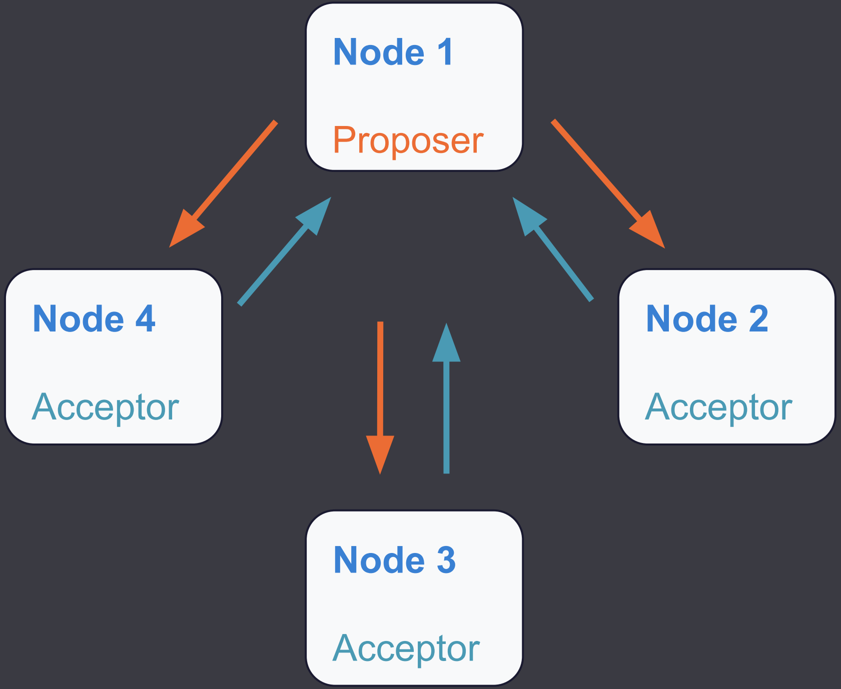


What’s different

Discussion about Consensus Protocols

Nerds may spontaneously erupt in debate, conjecture and questionable behavior.

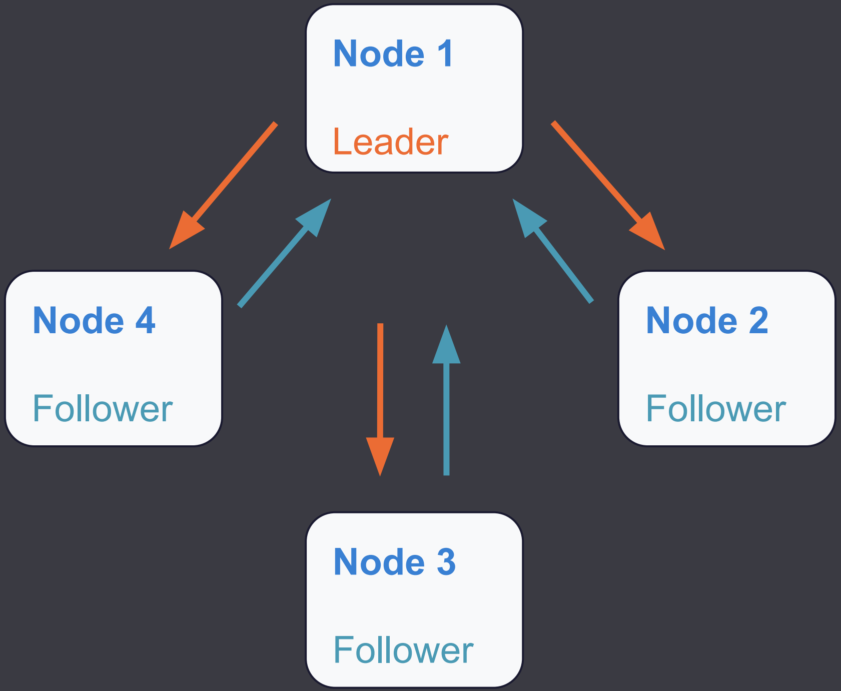
Paxos



- Origin of most consensus protocols
- **Proposer**: I want to do something
- **Acceptor**: Ok or Nope
- Do that until you have majority
- Network round trips add up
- Used in LWT(multi-paxos)

1989

Raft

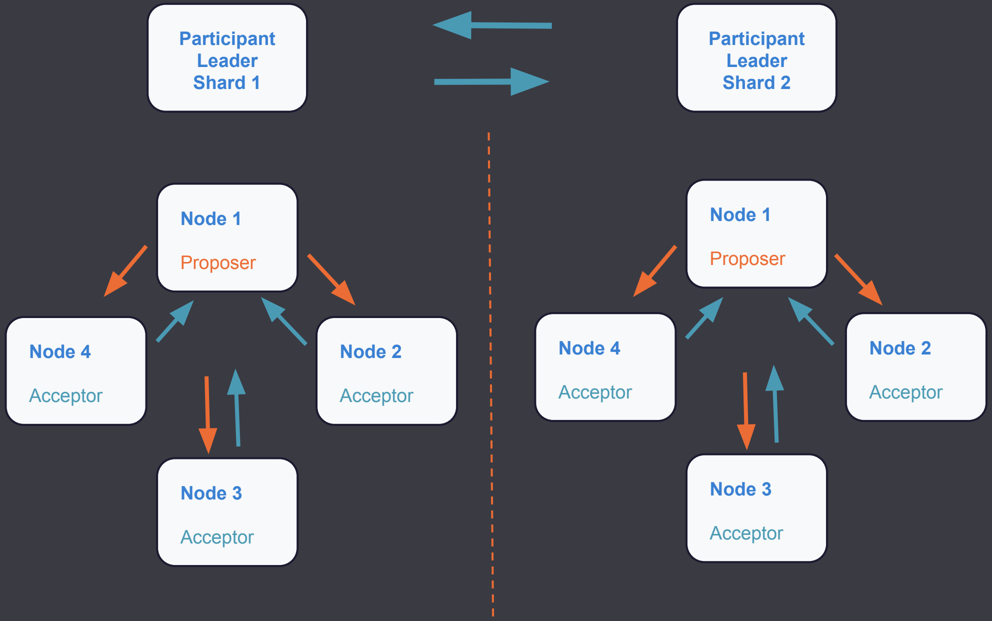


- Leader election to eliminate round trips
- **Leader**: All changes through me
- **Follower**: I trust dear leader

Bad for Cassandra

- Failure Modes lead to latency
- Multi-datacenter? Nope

Spanner

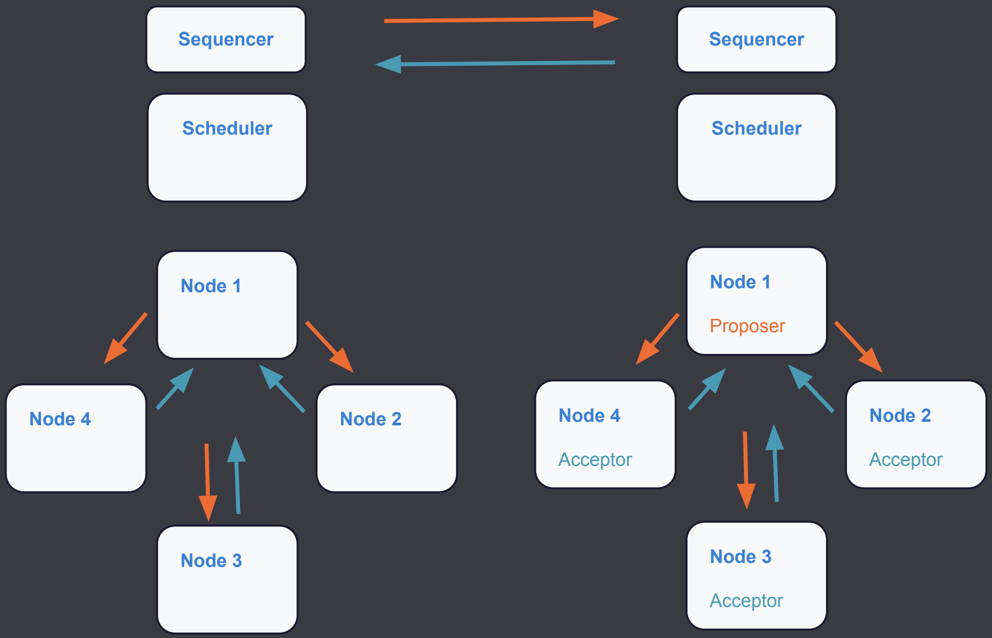


- One paxos group per shard
- Single row and single shard: Paxos
- Multi-row: Leaders coordinate for 2pc
- All depends on TrueTime™

Bad for Cassandra

- Depends on TrueTime™
- Many hops for one insert

Calvin

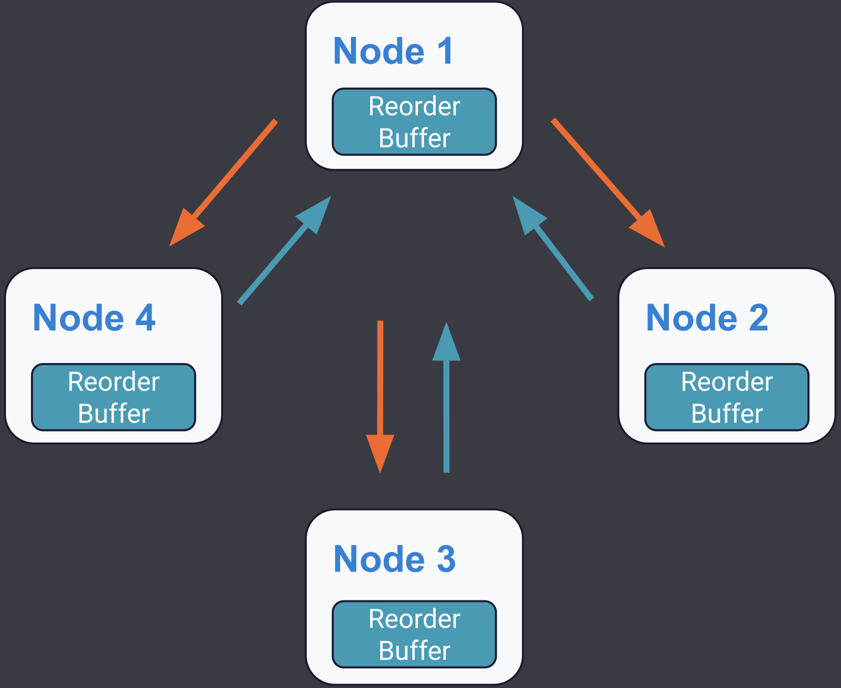


- Improvement in hops from Spanner
- Global Consensus vs Sharded
- Sequencer addresses clock skew
- Scheduler eliminates the TPC

Bad for Cassandra

- Complex failure modes
- Latency on multi-datacenter

Accord



- Every node has a Reorder Buffer
- Clock skew is cool
- Leaderless timestamp protocol
- Fast Path Electorates: Fault tolerance
- TL;DR One Round Trip - ish

Good for Cassandra

- Leaderless
- Scales like Cassandra
- Failure modes match

DS

Usage

Transaction Syntax - Boundaries

```
BEGIN TRANSACTION
   LET <tuple> = ( SELECT <column1>,<column2>..  FROM <table> WHERE <condition);


   SELECT <return_column>  FROM <table> WHERE <condition>;


   IF <tuple_condition> THEN
       UPDATE|INSERT|DELETE..
   END IF
COMMIT TRANSACTION;
```

- Everything inside happens or it doesn’t
- All statements are isolated
- All mutations are atomic
- No rollbacks (yet)

Transaction Syntax – State collection

```
BEGIN TRANSACTION
   LET <tuple> = (SELECT <column1>,<column2>.. FROM <table> WHERE <condition);


   SELECT <return_column>  FROM <table> WHERE <condition>;


   IF <tuple_condition> THEN
       UPDATE|INSERT|DELETE..
   END IF
COMMIT TRANSACTION;
```

- Gather state for use in the conditional below
- Tuple stores one or more columns of data

Transaction Syntax – Return value

```
BEGIN TRANSACTION
   LET <tuple> = ( SELECT <column1>,<column2>.. FROM <table> WHERE <condition);

   SELECT <return_column> FROM <table> WHERE <condition>;

   IF <tuple_condition> THEN
       UPDATE|INSERT|DELETE..
   END IF
COMMIT TRANSACTION;
```

- Return state from before transaction

Transaction Syntax – Conditional mutation

```
BEGIN TRANSACTION
   LET <tuple> = ( SELECT <column1>,<column2>.. FROM <table> WHERE <condition);


   SELECT <return_column> FROM <table> WHERE <condition>;


   IF <tuple_condition> THEN
       UPDATE|INSERT|DELETE..
   END IF
COMMIT TRANSACTION;
```

- Condition test (=, !=, >, <, <=, >=)
- Introduction of NULL, NOT NULL test
- False condition avoids updates

New Use Cases

DS

Bank Transaction

Multi-Partition Exclusive Operation

Bank Transaction – Setup

Table

```
CREATE TABLE ks.accounts (
   account_holder text,
   account_balance decimal,
   PRIMARY KEY (account_holder)
);
```

Data

```
INSERT INTO ks.accounts(account_holder, account_balance) VALUES ('bob', 100);
INSERT INTO ks.accounts(account_holder, account_balance) VALUES ('alice', 100);
```

Bank Transaction – Begin

```
BEGIN TRANSACTION
   // Get the balance from Alice’s account and store as a Tuple
   LET fromBalance = (SELECT account_balance FROM ks.accounts WHERE account_holder='alice');

   // Return the balance before update after transaction complete
   SELECT account_balance FROM ks.accounts WHERE account_holder='alice';

   // If Alice’s account balance is greater than $20, move $ 20 to Bob
   IF fromBalance.account_balance >= 20 THEN
       UPDATE ks.accounts SET account_balance -= 20 WHERE account_holder='alice';
       UPDATE ks.accounts SET account_balance +=20 WHERE account_holder='bob';
   END IF
COMMIT TRANSACTION;
```

Bank Transaction - Pre-condition

```
BEGIN TRANSACTION
   // Get the balance  from Alice’s account  and store as a Tuple
   LET fromBalance = (SELECT account_balance FROM ks.accounts WHERE account_holder='alice');

   // Return the balance before  update after transaction complete
   SELECT account_balance  FROM ks.accounts  WHERE account_holder='alice';

   // If Alice’s account balance is greater than zero, move $  20 to Bob
   IF fromBalance.account_balance >=  20 THEN
       UPDATE ks.accounts  SET account_balance -=  20 WHERE account_holder='alice';
       UPDATE ks.accounts  SET account_balance +=20 WHERE account_holder='bob';
   END IF
COMMIT TRANSACTION;
```

Bank Transaction – Output previous state

```
BEGIN TRANSACTION
   // Get the balance  from Alice’s account  and store as a Tuple
   LET fromBalance = ( SELECT account_balance  FROM ks.accounts  WHERE account_holder='alice');

   // Return the balance before  update after transaction complete
   SELECT account_balance FROM ks.accounts WHERE account_holder='alice';

   // If Alice’s account balance is greater than zero, move $  20 to Bob
   IF fromBalance.account_balance >=  20 THEN
       UPDATE ks.accounts  SET account_balance -=  20 WHERE account_holder='alice';
       UPDATE ks.accounts  SET account_balance += 20 WHERE account_holder='bob';
   END IF
COMMIT TRANSACTION;
```

Bank Transaction – Conditional Statement

```
BEGIN TRANSACTION
   // Get the balance  from Alice’s account  and store as a Tuple
   LET fromBalance = ( SELECT account_balance  FROM ks.accounts  WHERE account_holder='alice');

   // Return the balance before  update after transaction complete
   SELECT account_balance  FROM ks.accounts WHERE account_holder='alice';

   // If Alice’s account balance is greater than zero, move $  20 to Bob
   IF fromBalance.account_balance >= 20 THEN
       UPDATE ks.accounts SET account_balance -= 20 WHERE account_holder='alice';
       UPDATE ks.accounts SET account_balance +=20 WHERE account_holder='bob';
   END IF
COMMIT TRANSACTION;
```

Bank Transaction – Commit

```
BEGIN TRANSACTION
   // Get the balance  from Alice’s account  and store as a Tuple
   LET fromBalance = ( SELECT account_balance  FROM ks.accounts  WHERE account_holder='alice');

   // Return the balance before  update after transaction complete
   SELECT account_balance  FROM ks.accounts  WHERE account_holder='alice';

   // If Alice’s account balance is greater than zero, move $  20 to Bob
   IF fromBalance.account_balance >=  20 THEN
       UPDATE ks.accounts  SET account_balance -=  20 WHERE account_holder='alice';
       UPDATE ks.accounts  SET account_balance +=20 WHERE account_holder='bob';
   END IF
COMMIT TRANSACTION;
```

Inventory Management

Multi-Table/Multi-Partition Exclusive Update

Inventory Management – Setup

Tables

```
CREATE TABLE ks.products (
   item text,
   inventory_count int,
   PRIMARY KEY (item)
);
```

```
CREATE TABLE ks.shopping_cart (
   user_name text,
   item text,
   item_count int,
   PRIMARY KEY (user_name, item)
);
```

Data

```
INSERT INTO ks.products(item, inventory_count) VALUES ('PlayStation 5', 100);
```

Inventory Management – Pre-Condition

```
BEGIN TRANSACTION
   // Find out how many PlayStations are left
   LET inventory = (SELECT inventory_count FROM ks.products WHERE item='PlayStation 5');

   // Return the inventory count before deducting
   SELECT item, inventory_count FROM ks.products WHERE item='PlayStation 5';

   // Take a PlayStation out of inventoryand put in users shopping cart
   IF inventory.inventory_count >0 THEN
       UPDATE ks.products SET inventory_count -= 1 WHERE item='PlayStation 5';
       INSERT INTO ks.shopping_cart(user_name, item, item_count)VALUES ('patrick', 'PlayStation 5', 1);
   END IF
COMMIT TRANSACTION;
```

Inventory Management – Output Previous State

```
BEGIN TRANSACTION
    // Find out how many PlayStations are left
    LET inventory = (SELECT inventory_count FROM ks.products WHERE item='PlayStation 5');

    // Return the inventory count before deducting
    SELECT item, inventory_count FROM ks.products WHERE item='PlayStation 5';

    // Take a PlayStation out of inventoryand put in users shopping cart
    IF inventory.inventory_count >0 THEN
        UPDATE ks.products SET inventory_count -= 1 WHERE item='PlayStation 5';
        INSERT INTO ks.shopping_cart(user_name, item, item_count)VALUES ('patrick', 'PlayStation 5', 1);
    END IF
COMMIT TRANSACTION;
```

Inventory Management – Conditional Statement

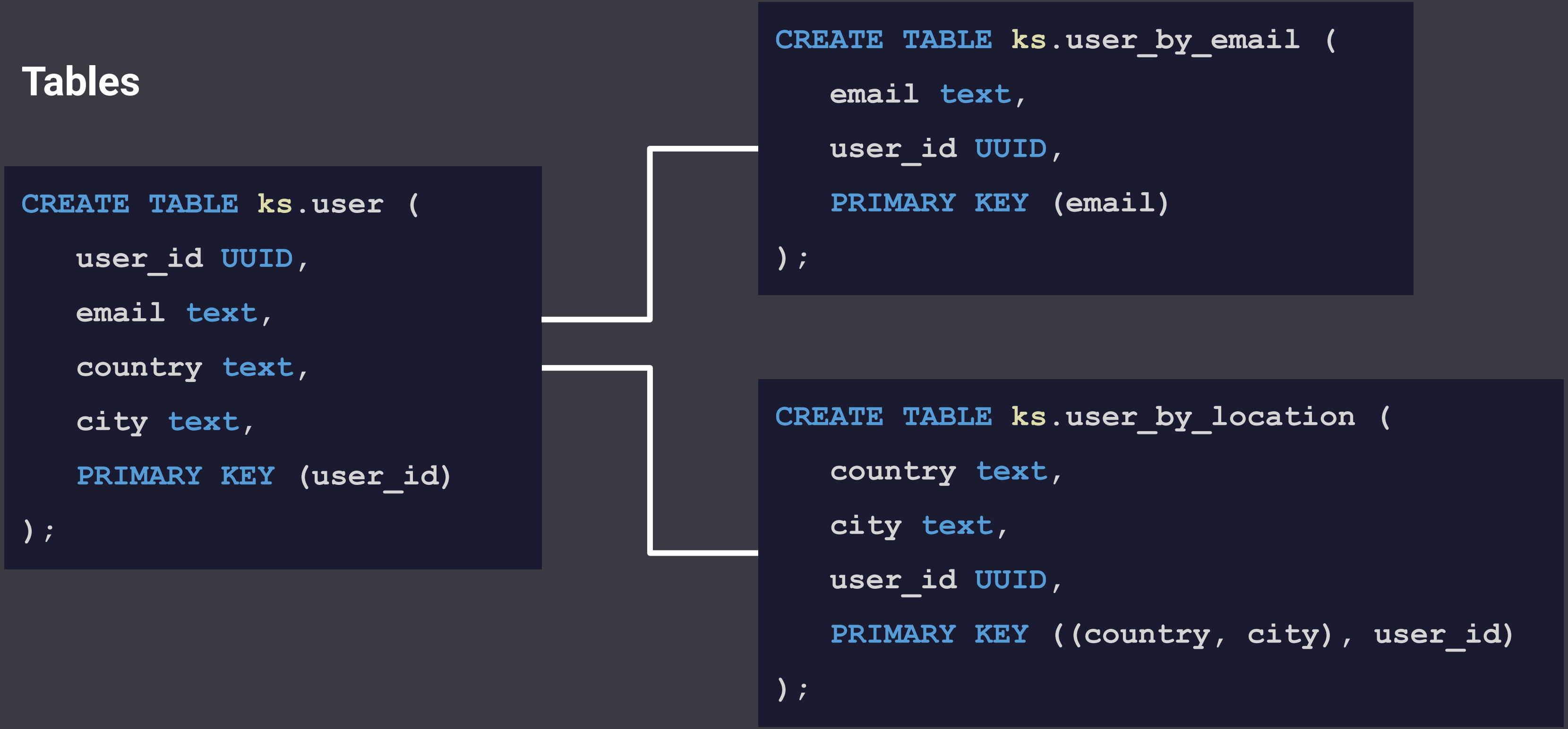
```
BEGIN TRANSACTION
   // Find out how many PlayStations are left
   LET inventory = (SELECT inventory_count FROM ks.products WHERE item='PlayStation 5');

   // Return the inventory count before deducting
   SELECT item, inventory_count FROM ks.products WHERE item='PlayStation 5';

   // Take a PlayStation out of inventory and put in users shopping cart
   IF inventory.inventory_count > 0 THEN
       UPDATE ks.products SET inventory_count -= 1 WHERE item='PlayStation 5';
       INSERT INTO ks.shopping_cart(user_name, item, item_count) VALUES ('patrick', 'PlayStation 5', 1);
   END IF
COMMIT TRANSACTION;
```

Real Atomic Batch

Foreign Key Management

Real Atomic Batch – Setup

Tables



```
CREATE TABLE ks.user (
   user_id UUID,
   email text,
   country text,
   city text,
   PRIMARY KEY (user_id)
);
```

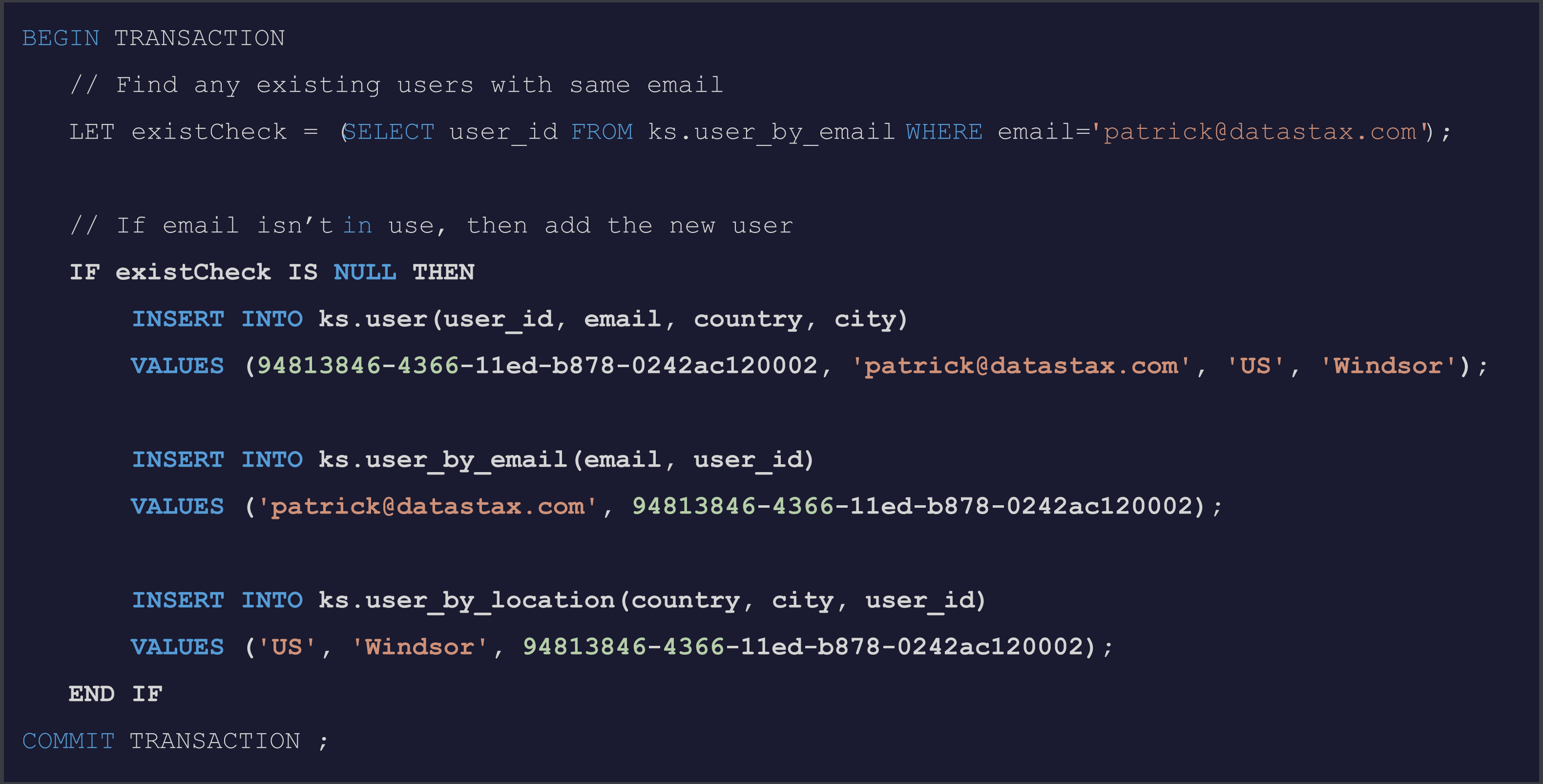
```
CREATE TABLE ks.user_by_email (
   email text,
   user_id UUID,
   PRIMARY KEY (email)
);
```

```
CREATE TABLE ks.user_by_location (
   country text,
   city text,
   user_id UUID,
   PRIMARY KEY ((country, city), user_id)
);
```

Real Atomic Batch – Existence Check

```
BEGIN TRANSACTION
   // Find any existing users with same email
   LET existCheck = (SELECT user_id FROM ks.user_by_email WHERE email='patrick@datastax.com');

   // If email isn’t in use, then add the new user
   IF existCheck IS NULL THEN
       INSERT INTO ks.user(user_id, email, country, city)
       VALUES (94813846-4366-11ed-b878-0242ac120002, 'patrick@datastax.com', 'US', 'Windsor');

       INSERT INTO ks.user_by_email(email, user_id)
       VALUES ('patrick@datastax.com', 94813846-4366-11ed-b878-0242ac120002);

       INSERT INTO ks.user_by_location(country, city, user_id)
       VALUES ('US', 'Windsor', 94813846-4366-11ed-b878-0242ac120002);
   END IF
COMMIT TRANSACTION ;
```

Real Atomic Batch - Execution



```
BEGIN TRANSACTION
   // Find any existing users with same email
   LET existCheck = (SELECT user_id FROM ks.user_by_email WHERE email='patrick@datastax.com');

   // If email isn’t in use, then add the new user
   IF existCheck IS NULL THEN
       INSERT INTO ks.user(user_id, email, country, city)
       VALUES (94813846-4366-11ed-b878-0242ac120002, 'patrick@datastax.com', 'US', 'Windsor');

       INSERT INTO ks.user_by_email(email, user_id)
       VALUES ('patrick@datastax.com', 94813846-4366-11ed-b878-0242ac120002);

       INSERT INTO ks.user_by_location(country, city, user_id)
       VALUES ('US', 'Windsor', 94813846-4366-11ed-b878-0242ac120002);
   END IF
COMMIT TRANSACTION ;
```

Usable Counters

Imagine a world without counter columns

Usable Counters – Setup

Table

```
CREATE TABLE ks.products (
   item text,
   inventory_count decimal,
   PRIMARY KEY (item)
);
```

Usable Counters – Operations

Set

```
BEGIN TRANSACTION
      UPDATE ks.products SET inventory_count = 100 WHERE item='PlayStation 5';
COMMIT TRANSACTION;
```

Increment

```
BEGIN TRANSACTION
      UPDATE ks.products SET inventory_count += 1 WHERE item='PlayStation 5';
COMMIT TRANSACTION;
```

Decrement

```
BEGIN TRANSACTION
      UPDATE ks.products SET inventory_count -= 1 WHERE item='PlayStation 5';
COMMIT TRANSACTION;
```

Other Things (Haven’t Worked Them Out Yet)

Document Locking

Lease Management

Match Making (gaming)

Final Thoughts

What’s next?

- Performance testing. TPC-C?
- Gather user feedback
- We need to find limits and communicate early

This Will Change Cassandra in Profound Ways

Save the Date!

CASSANDRA SUMMIT

MARCH 13-14, 2023 • SAN JOSE, CA

- Training day March 12
- In-person + Virtual
- CFP Open
- Reg coming soon

http://cassandrasummit.org

Thank You!