Cassandra Developer Workshop

Cedrick Lunven | Director of Developer Advocacy
Eric Zietlow | Developer Advocate
David Gilardi | Developer Advocate
Aleksandr Volochnev | Developer Advocate
Jack Fryer | Community Manager

DATASTAX® + Apache Cassandra™

DATASTAX DEVELOPERS

Cedrick Lunven

Developer Advocate

Jack Fryer

Community Manager

Aleksandr Volochnev

Developer Advocate

Eric Zietlow

Developer Advocate

David Gilardi

Developer Advocate

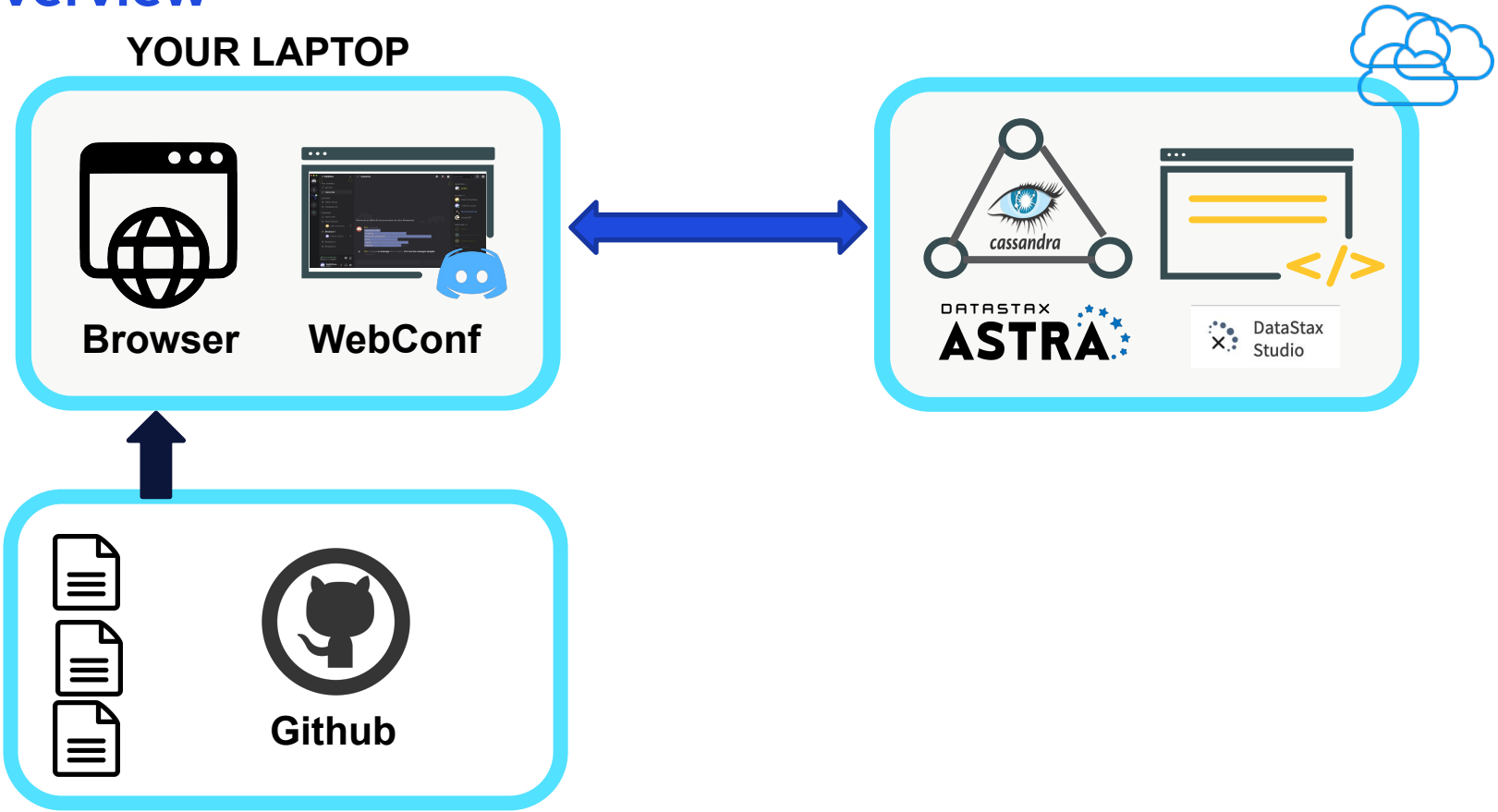
Developer Workshop

1. Bootstrapping
2. Apache Cassandra™ Why, What & When
3. Data Modeling with Apache Cassandra™
4. What's NEXT ?

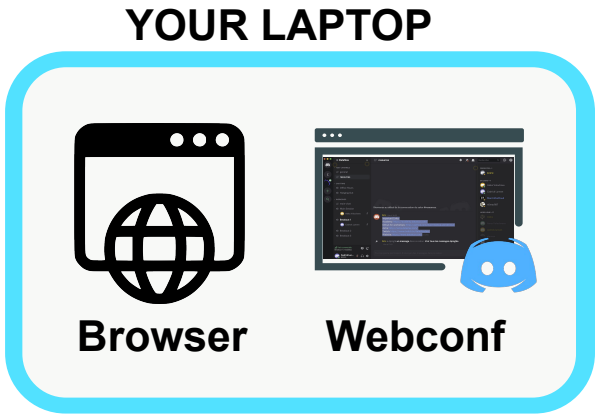
Developer Workshop

1. **Bootstrapping**
2. Apache Cassandra™ Why, What & When
3. Data Modeling with Apache Cassandra™
4. What's NEXT ?

Overview


YOUR LAPTOP
Browser
WebConf
cassandra
DATASTAX
ASTRA
DataStax
Studio
Github

Get your hands on Discord

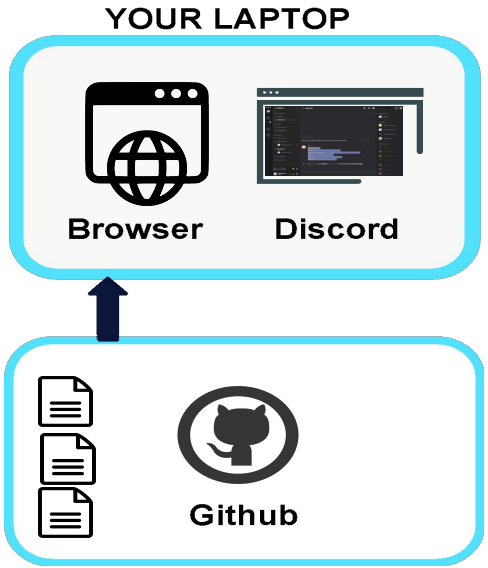


- What you need to know
- List of resources available for you
- How to ask questions
- How to get the exercises
- How to mark the Exercises done

Exercise 0

DATASTAX

Bootstrap your Environment



1. **Clone or download repository material**

```
https://github.com/DataStax-Academy/cassandra-workshop-online
```

2. **Send us an email**

```
jack.fryer@datastax.com
```

Introducing Astra

Eliminate Operations

everything from provisioning to backups is fully automated

Secure Your Data

with the most advanced security available for Cassandra

Simplify App Development

with auto-configured developer tools that deploy with a click

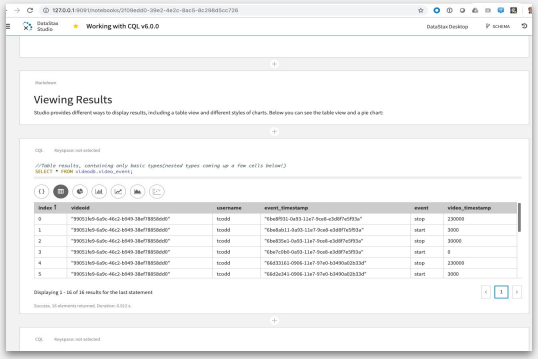
DATASTAX ASTRA

Simplify Application Development

Familiar Language

```
INSERT INTO mytable
(id,name,address) VALUES
(1,'Bob Smith','1 Main
Street')
SELECT * FROM mytable
WHERE id=1
UPDATE mytable SET
name='Tom Smith' WHERE
id=1
DELETE FROM mytable WHERE
id=1
```

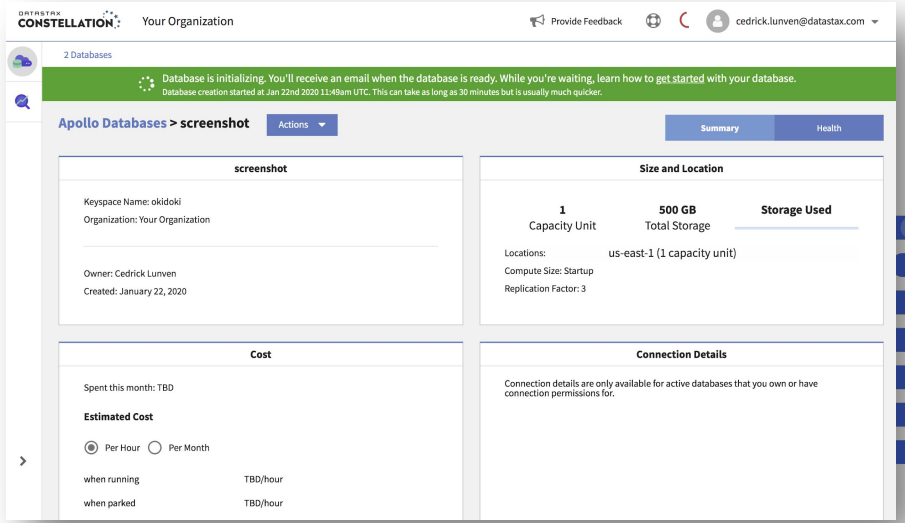
Easy Dev Tools



Great Drivers

Exercise 1

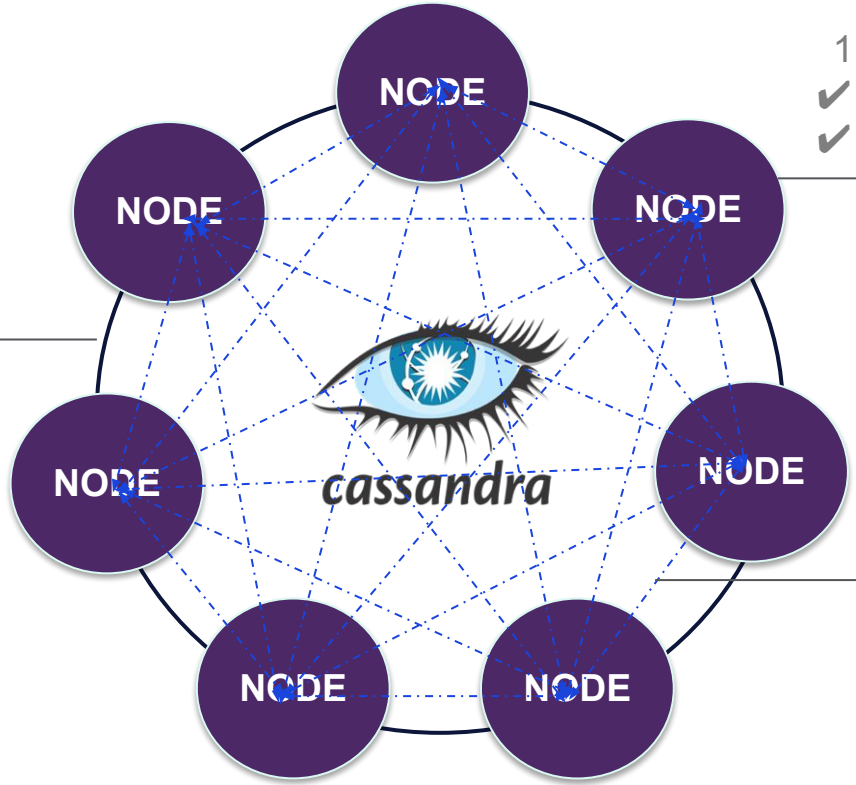
Create your Astra Instance



Developer Workshop

1. Bootstrapping
2. **Apache Cassandra™ Why, What & When**
3. Data Modeling with Apache Cassandra™
4. What's NEXT ?

Apache Cassandra™ = NoSQL Distributed Database



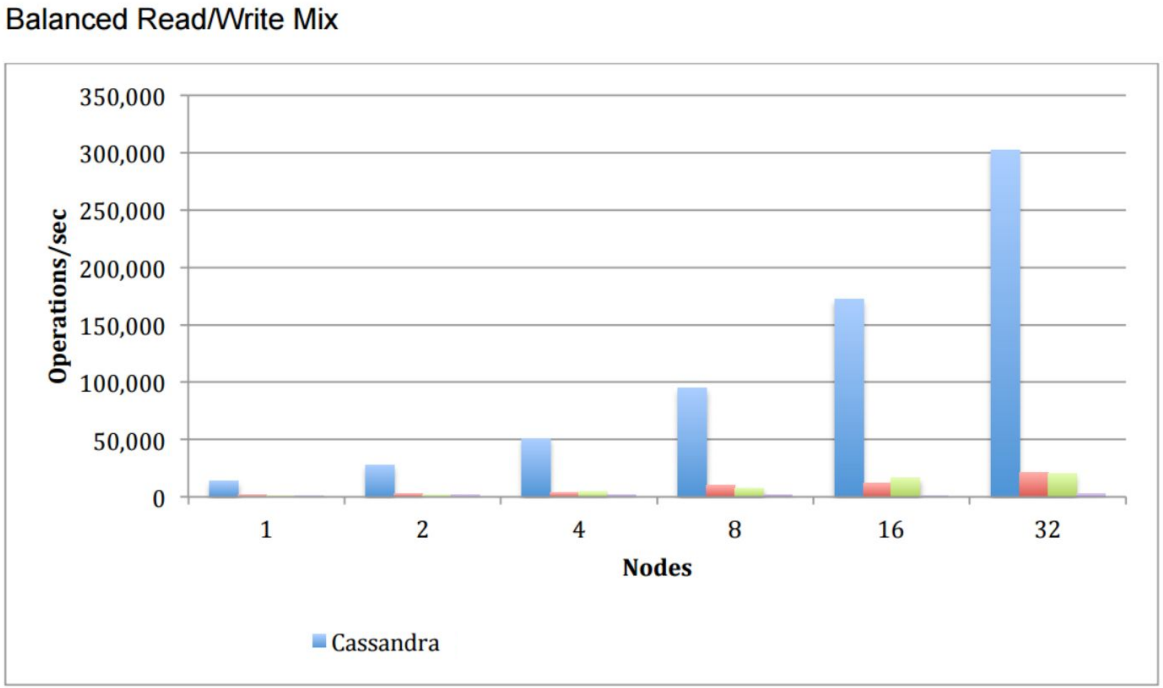
1 Installation = 1 NODE
- ✔ Capacity: ± 1TB
- ✔ Throughput: 3000 Tx/sec/core

DataCenter | Ring

Communication:
- ✔ Gossiping

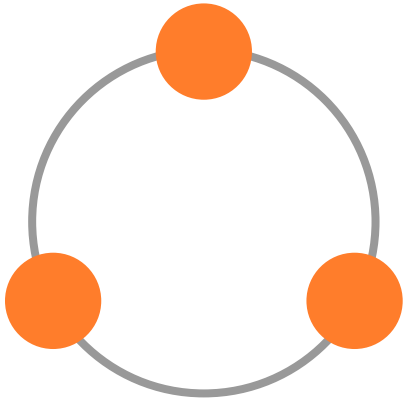
Scales Linearly

- Need more capacity?
- Need more throughput?
- Add nodes!

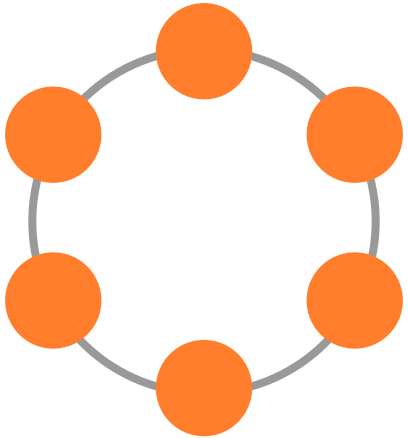


Horizontal vs. Vertical Scaling

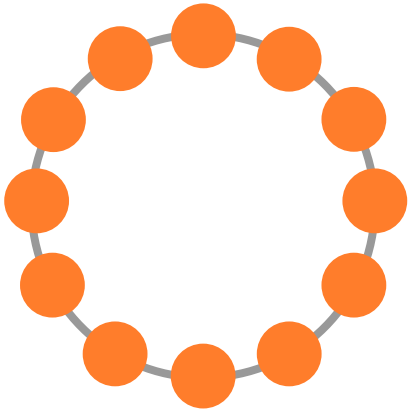
- Vertical scaling requires one large expensive machine
- Horizontal scaling requires multiple less-expensive commodity hardware



100,000 transactions/second

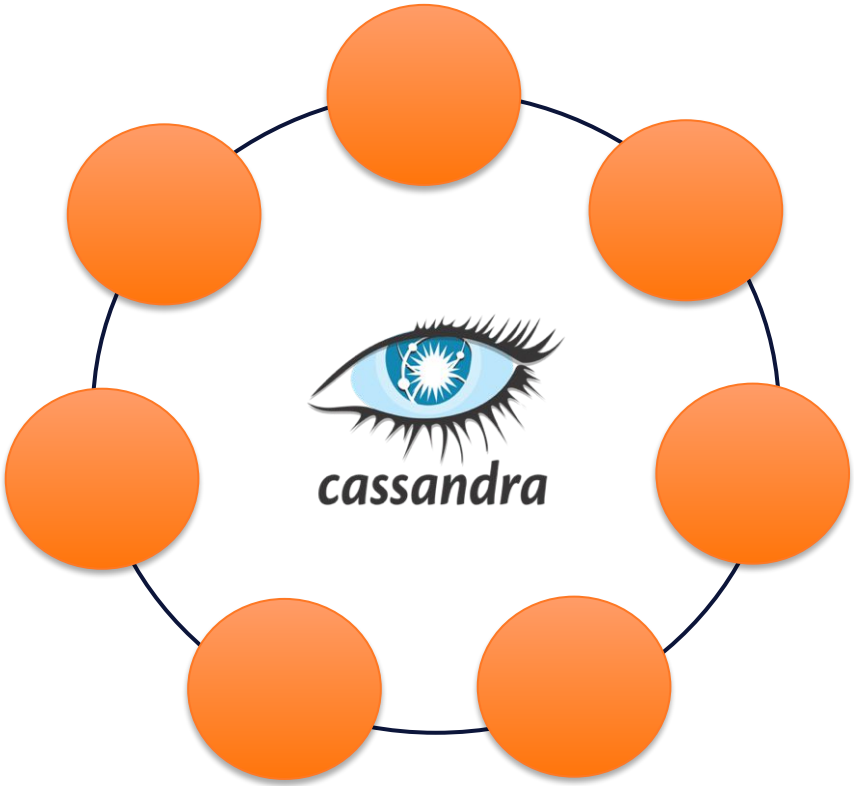


200,000 transactions/second



400,000 transactions/second

Data is Distributed

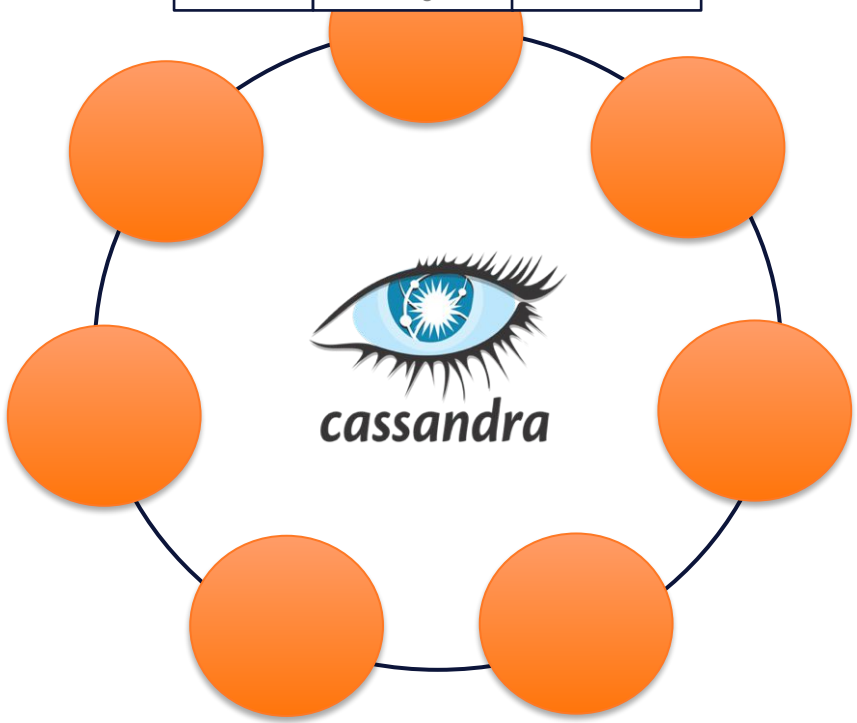


Country	City	Population
USA	New York	8.000.000
USA	Los Angeles	4.000.000
FR	Paris	2.230.000
DE	Berlin	3.350.000
UK	London	9.200.000
AU	Sydney	4.900.000
DE	Nuremberg	500.000
CA	Toronto	6.200.000
CA	Montreal	4.200.000
FR	Toulouse	1.100.000
JP	Tokyo	37.430.000
IN	Mumbai	20.200.000

Partition Key

Data is Distributed

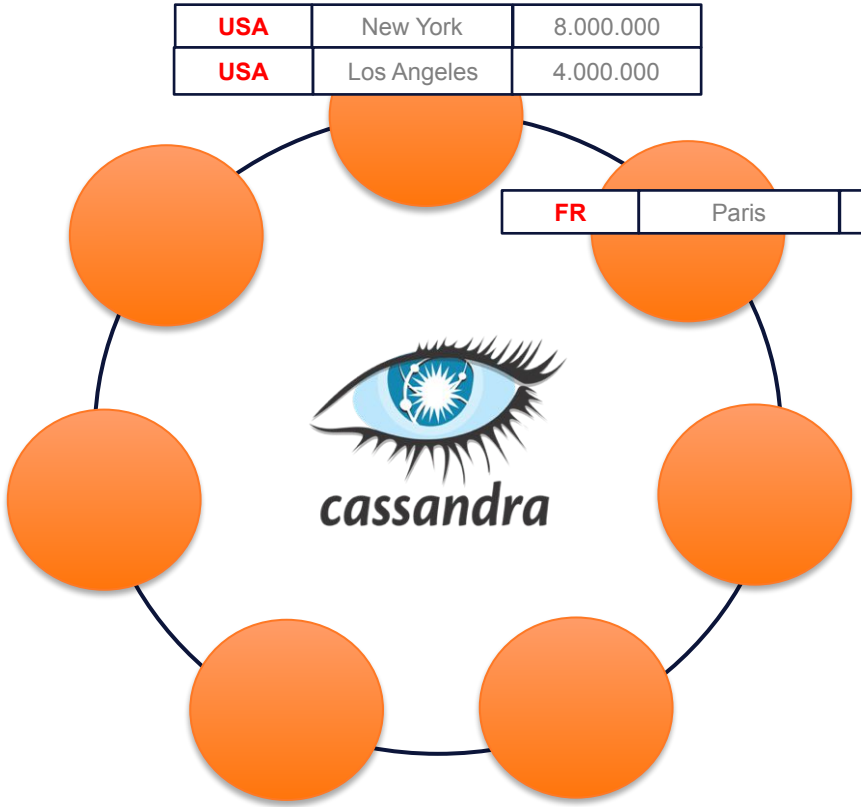
USA	New York	8.000.000
USA	Los Angeles	4.000.000



Country	City	Population
FR	Paris	2.230.000
DE	Berlin	3.350.000
UK	London	9.200.000
AU	Sydney	4.900.000
DE	Nuremberg	500.000
CA	Toronto	6.200.000
CA	Montreal	4.200.000
FR	Toulouse	1.100.000
JP	Tokyo	37.430.000
IN	Mumbai	20.200.000

Partition Key

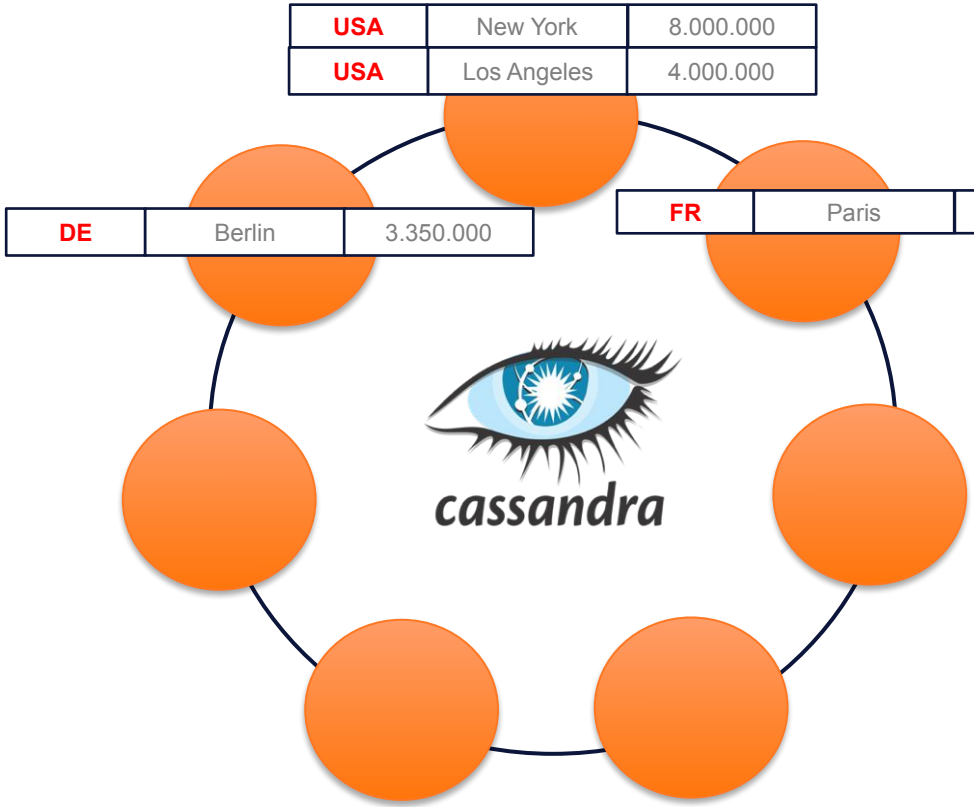
Data is Distributed



Country	City	Population
DE	Berlin	3.350.000
UK	London	9.200.000
AU	Sydney	4.900.000
DE	Nuremberg	500.000
CA	Toronto	6.200.000
CA	Montreal	4.200.000
FR	Toulouse	1.100.000
JP	Tokyo	37.430.000
IN	Mumbai	20.200.000

Partition Key

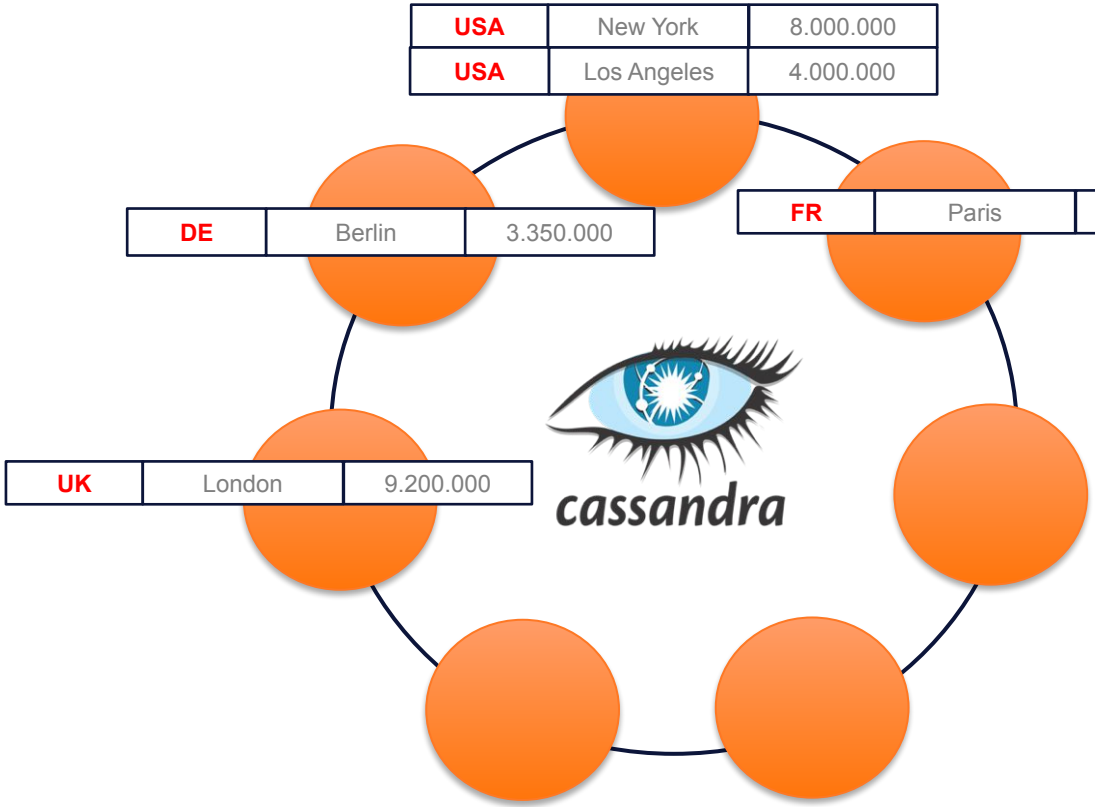
Data is Distributed



Country	City	Population
UK	London	9.200.000
AU	Sydney	4.900.000
DE	Nuremberg	500.000
CA	Toronto	6.200.000
CA	Montreal	4.200.000
FR	Toulouse	1.100.000
JP	Tokyo	37.430.000
IN	Mumbai	20.200.000

Partition Key

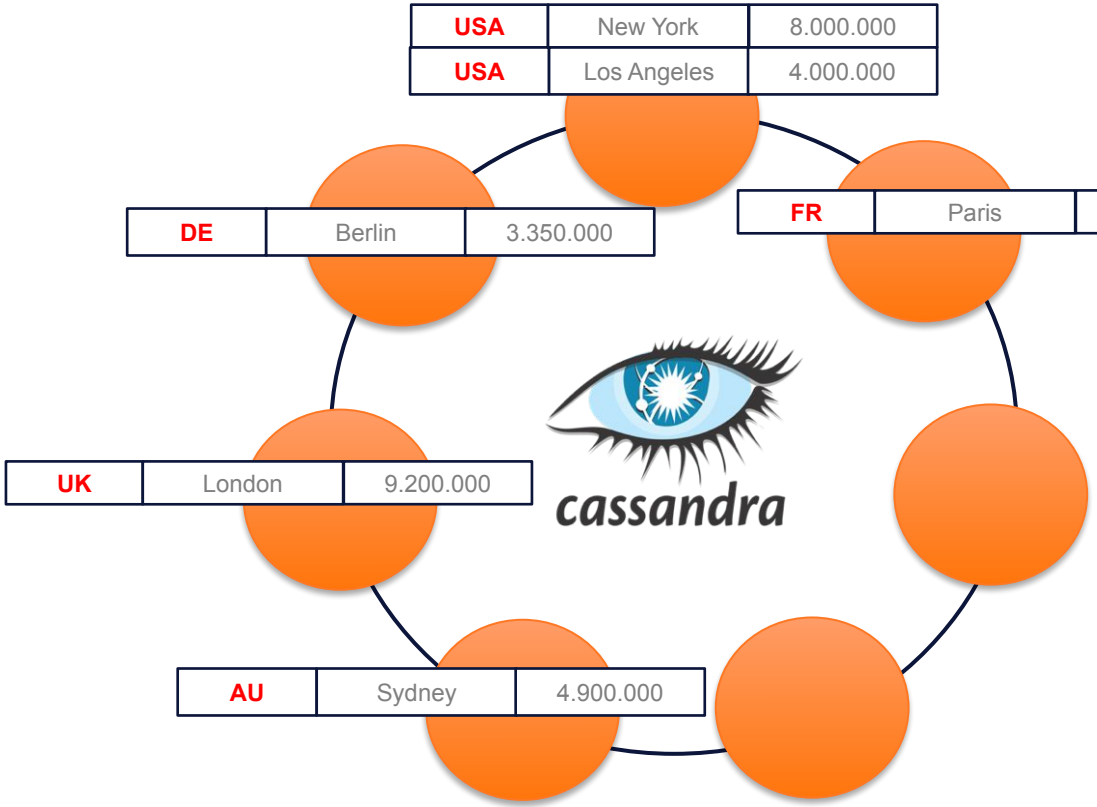
Data is Distributed



Country	City	Population
AU	Sydney	4.900.000
DE	Nuremberg	500.000
CA	Toronto	6.200.000
CA	Montreal	4.200.000
FR	Toulouse	1.100.000
JP	Tokyo	37.430.000
IN	Mumbai	20.200.000

Partition Key

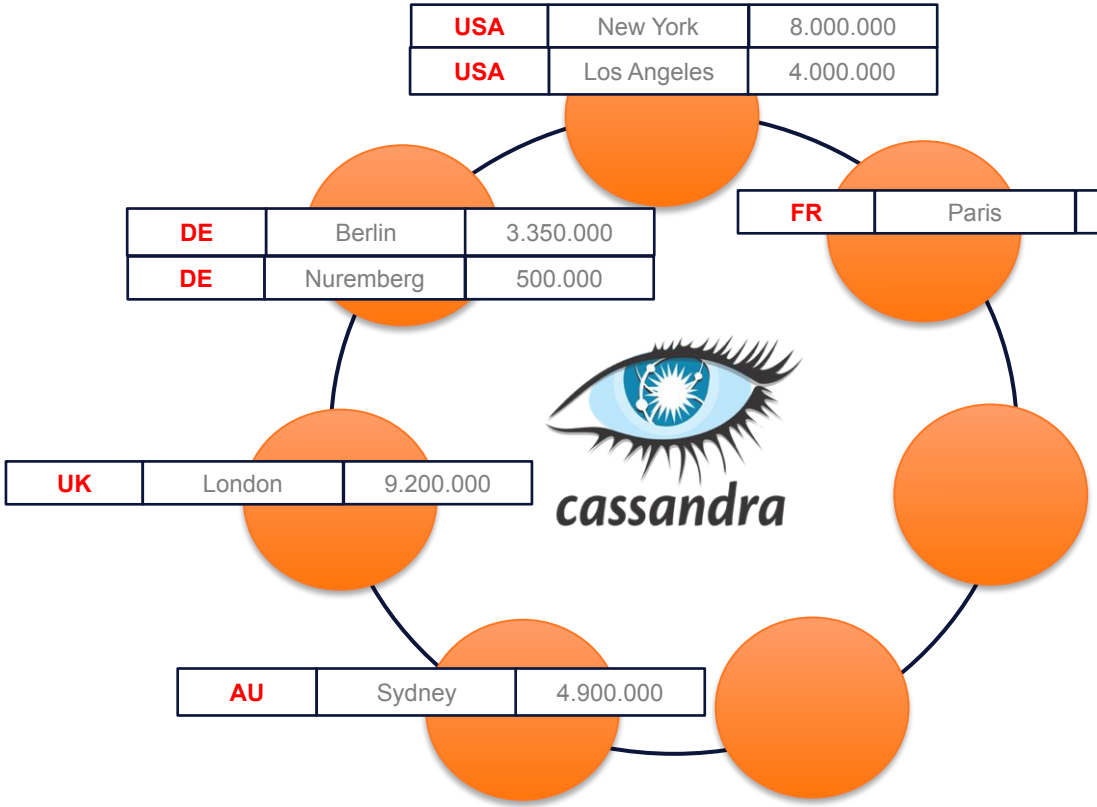
Data is Distributed



Country	City	Population
DE	Nuremberg	500.000
CA	Toronto	6.200.000
CA	Montreal	4.200.000
FR	Toulouse	1.100.000
JP	Tokyo	37.430.000
IN	Mumbai	20.200.000

Partition Key

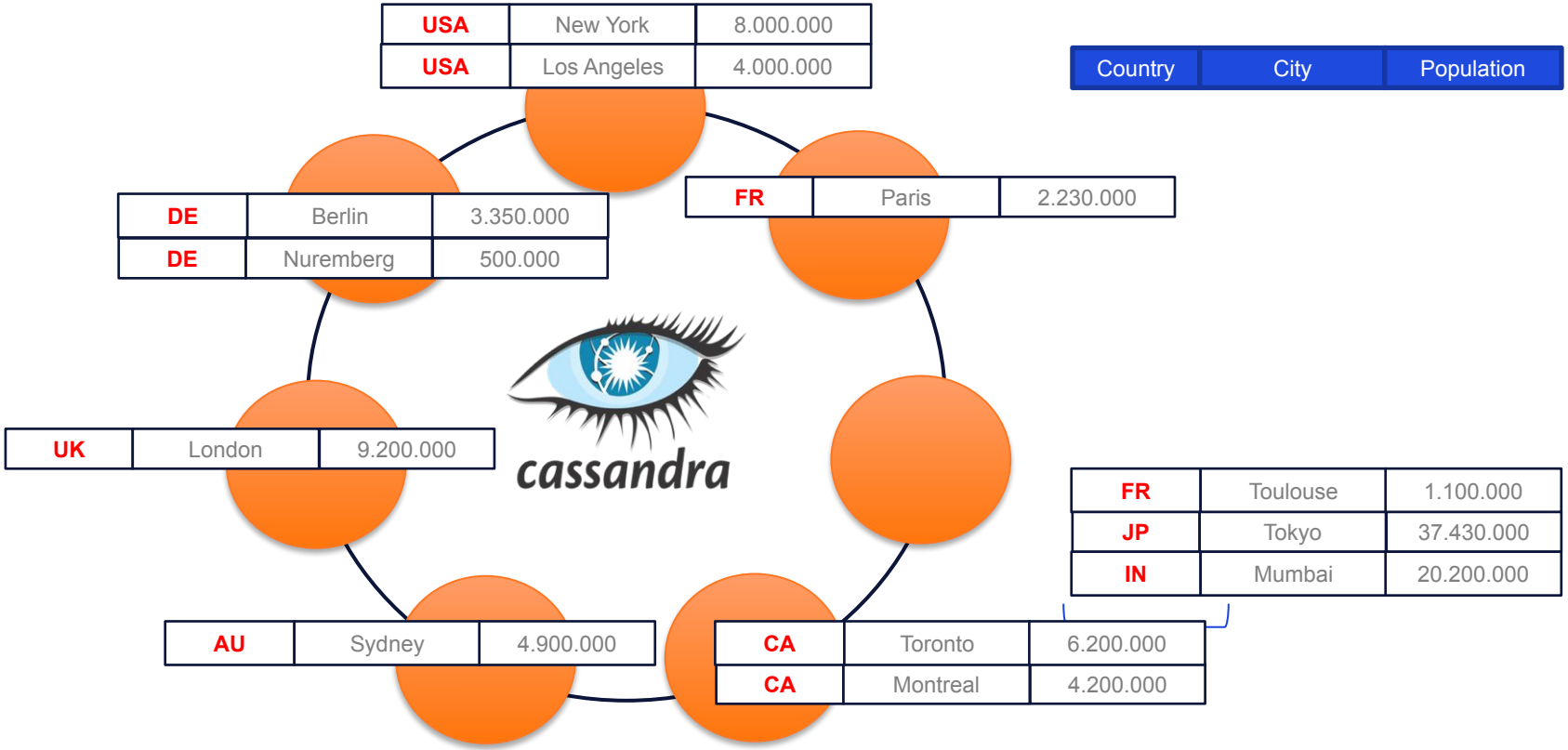
Data is Distributed



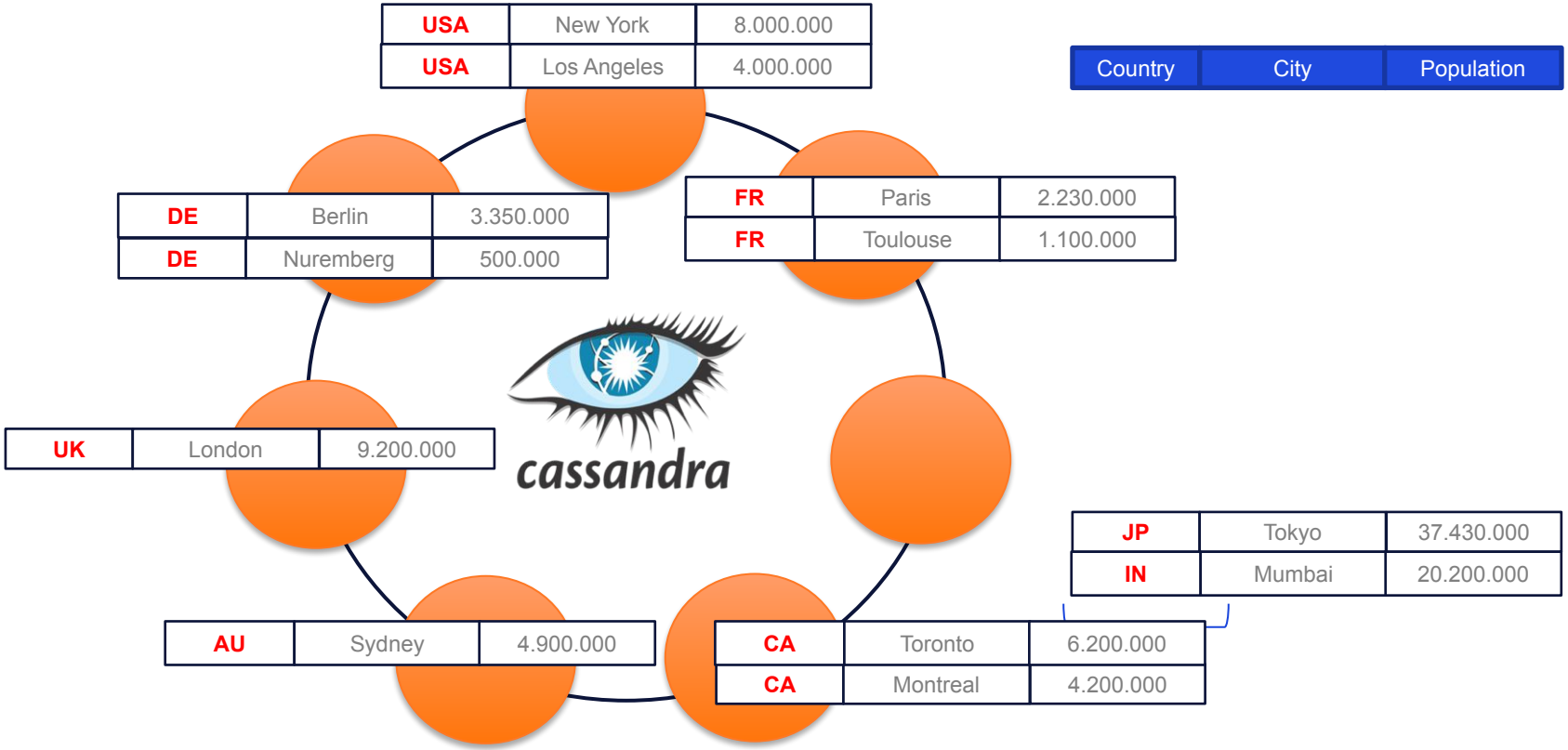
Country	City	Population
CA	Toronto	6.200.000
CA	Montreal	4.200.000
FR	Toulouse	1.100.000
JP	Tokyo	37.430.000
IN	Mumbai	20.200.000

Partition Key

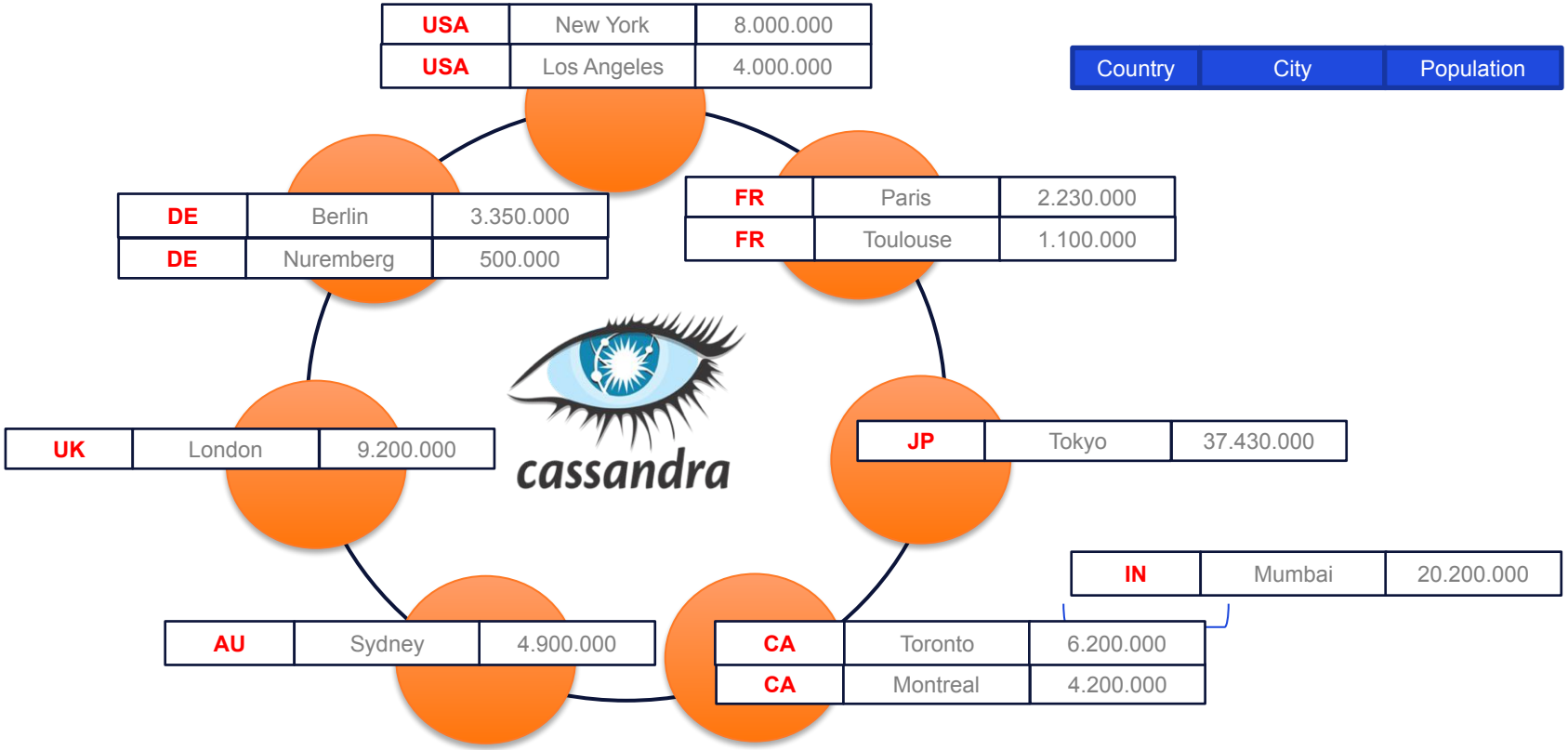
Data is Distributed



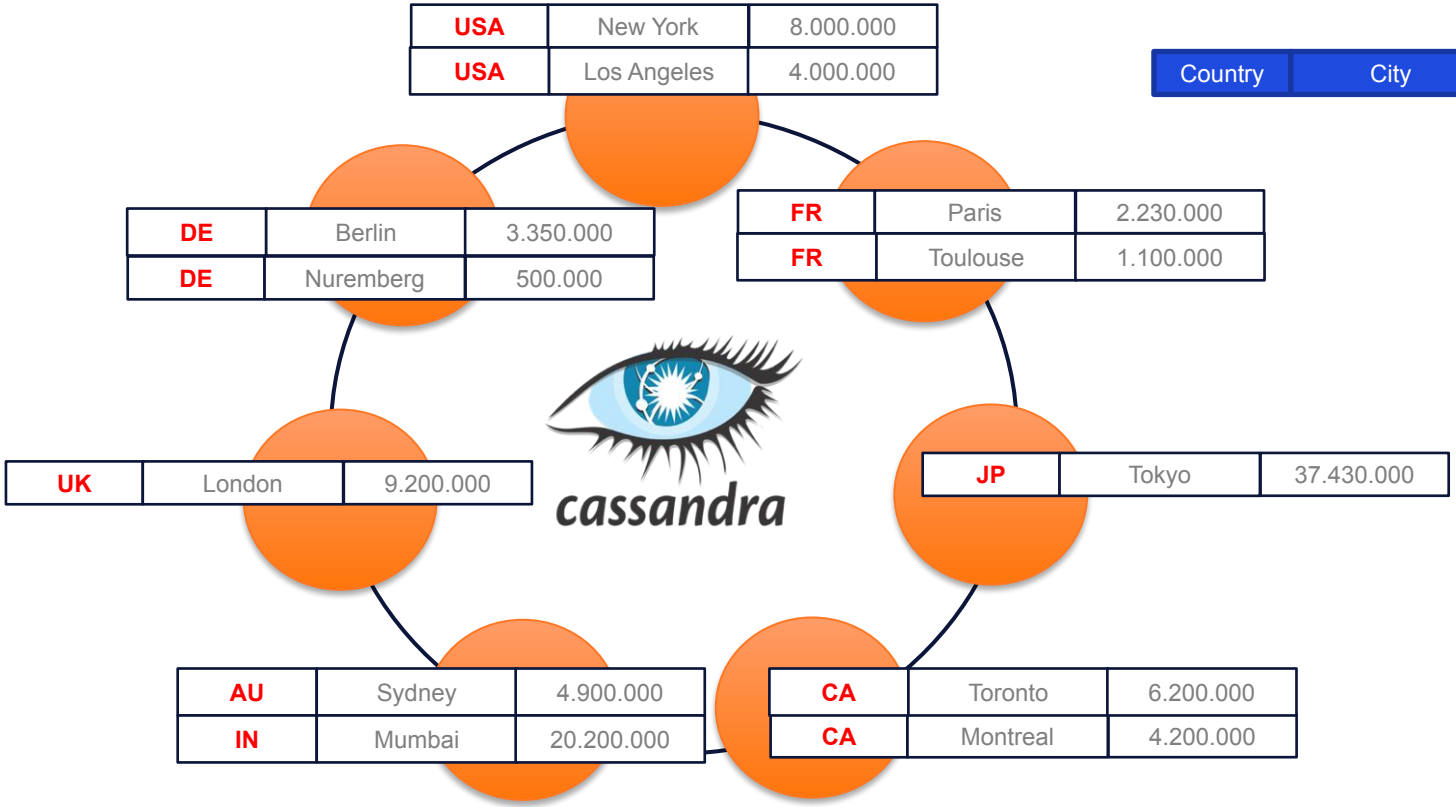
Data is Distributed



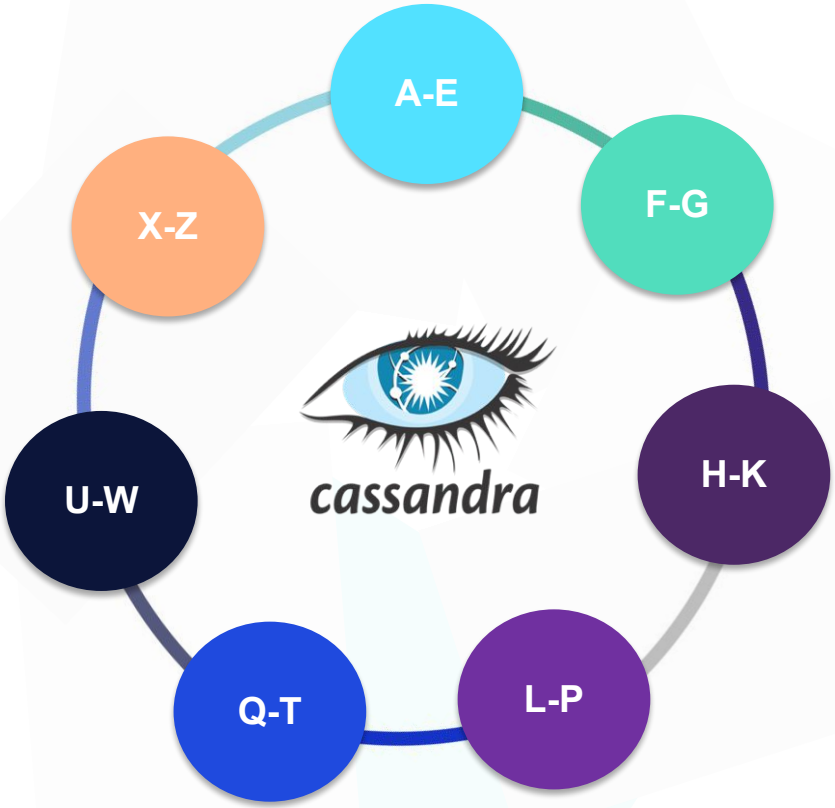
Data is Distributed



Data is Distributed



Data is Distributed



Data is *Evenly*-distributed

CO	City	Population
AU	Sydney	4.900.000
CA	Toronto	6.200.000
CA	Montreal	4.200.000
DE	Berlin	3.350.000
DE	Nuremberg	500.000

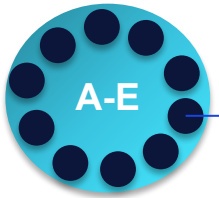
Partition Key

Partitioner

Hashing Function

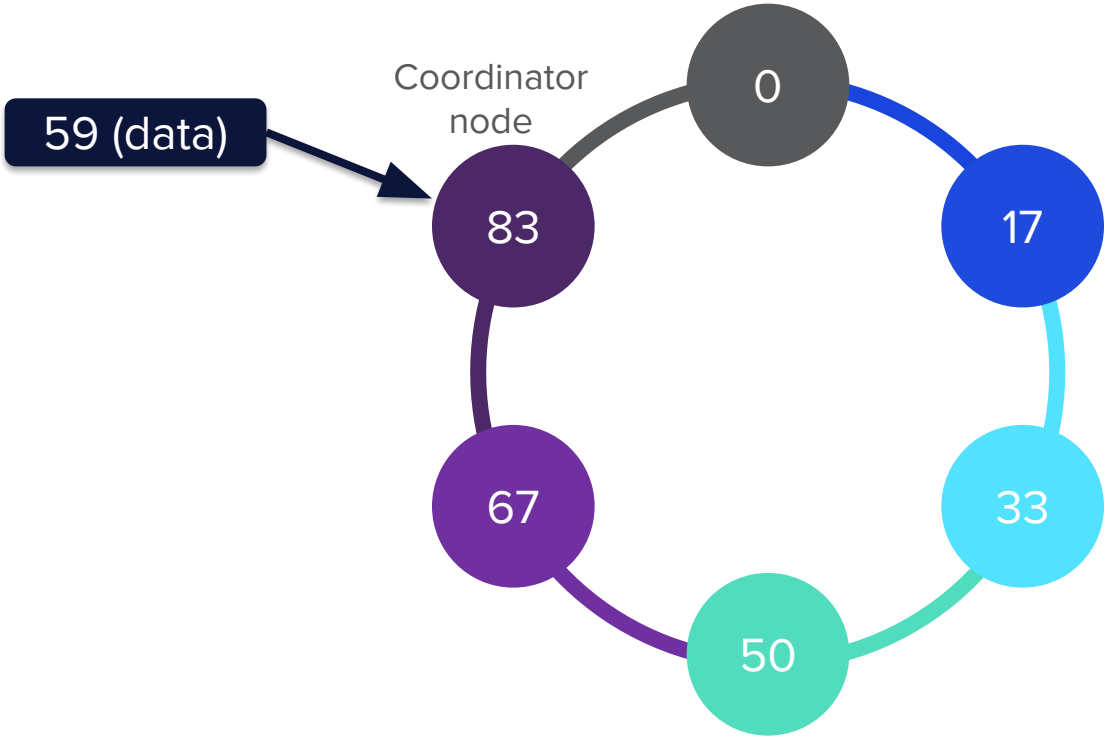
CO	City	Population
59	Sydney	4.900.000
12	Toronto	6.200.000
12	Montreal	4.200.000
45	Berlin	3.350.000
45	Nuremberg	500.000

Tokens

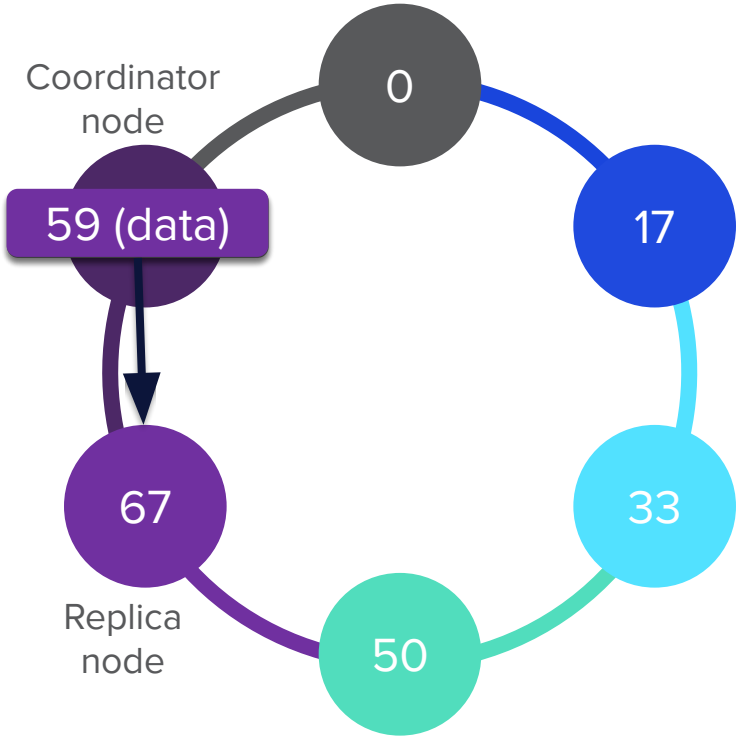


VIRTUAL NODE

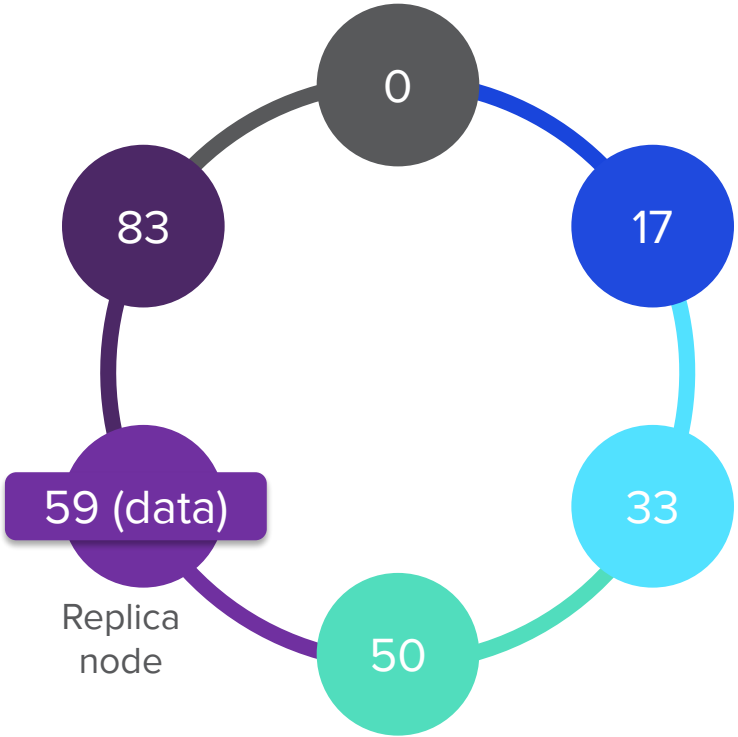
How the Ring Works



How the Ring Works

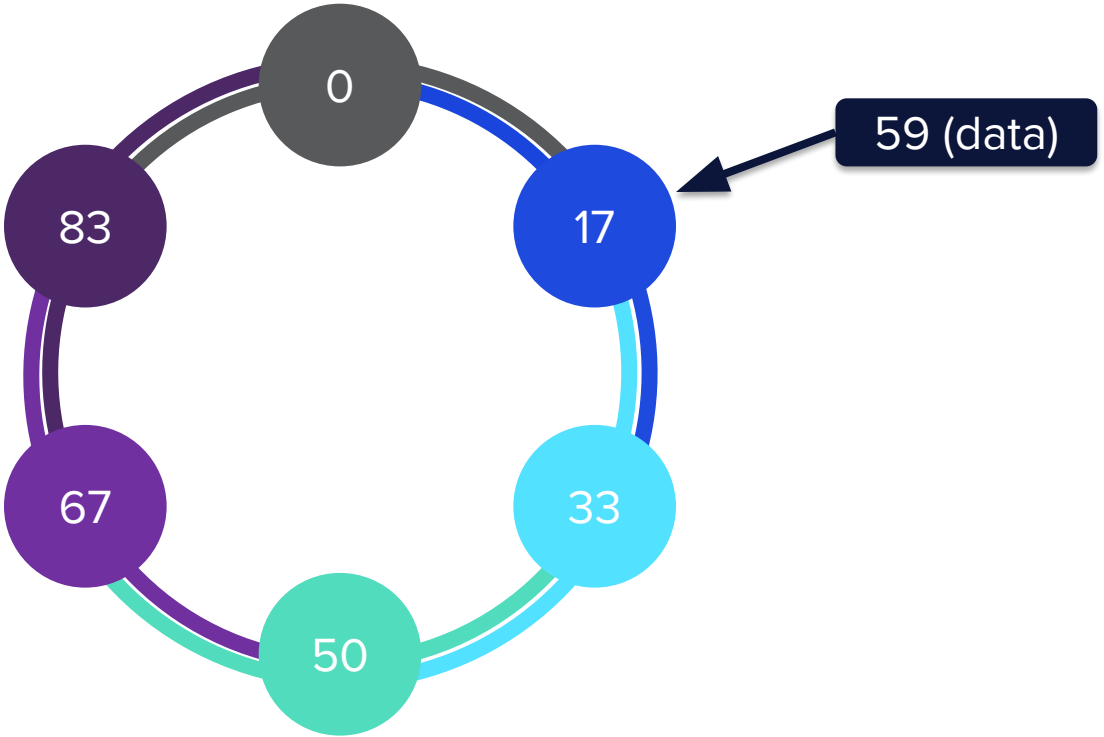


How the Ring Works



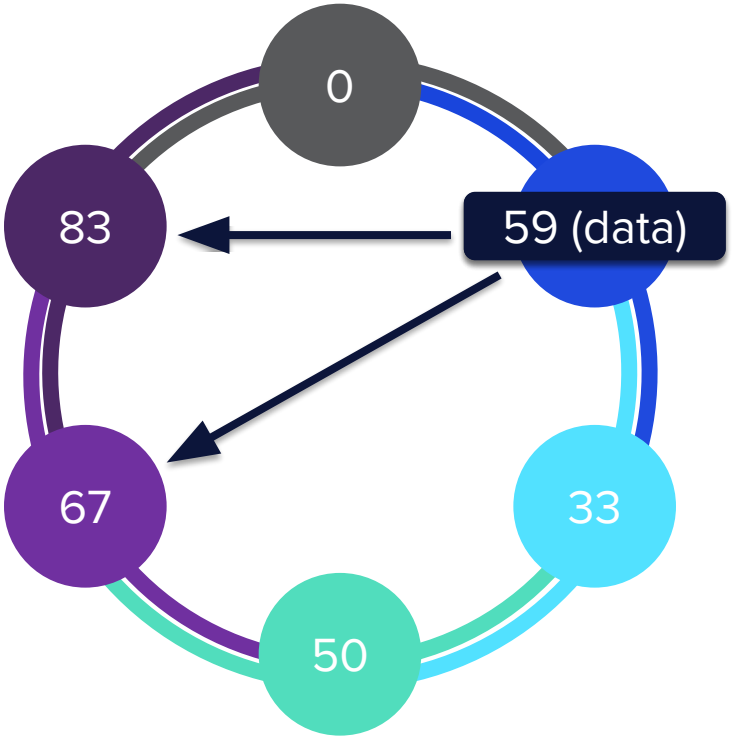
Replication within the Ring

RF = 2



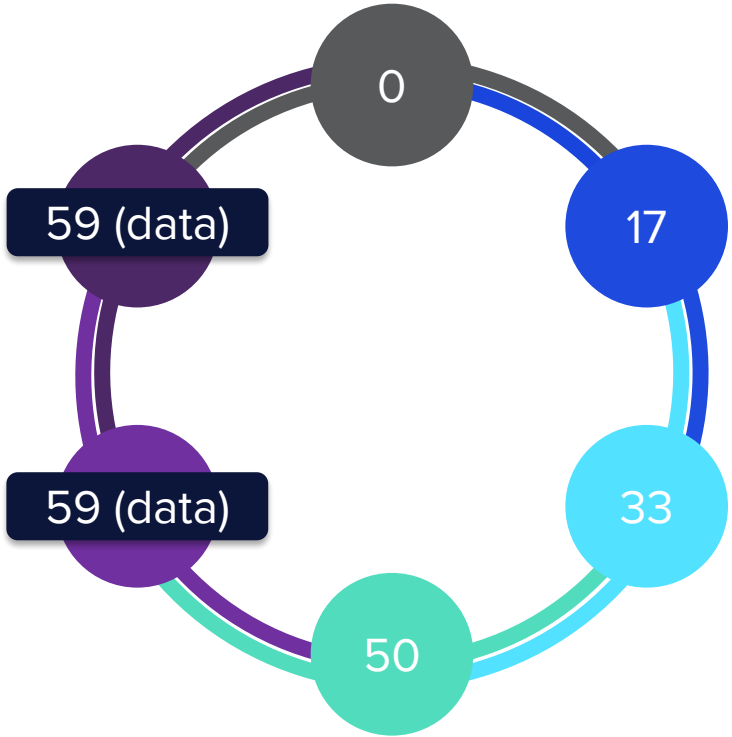
Replication within the Ring

RF = 2



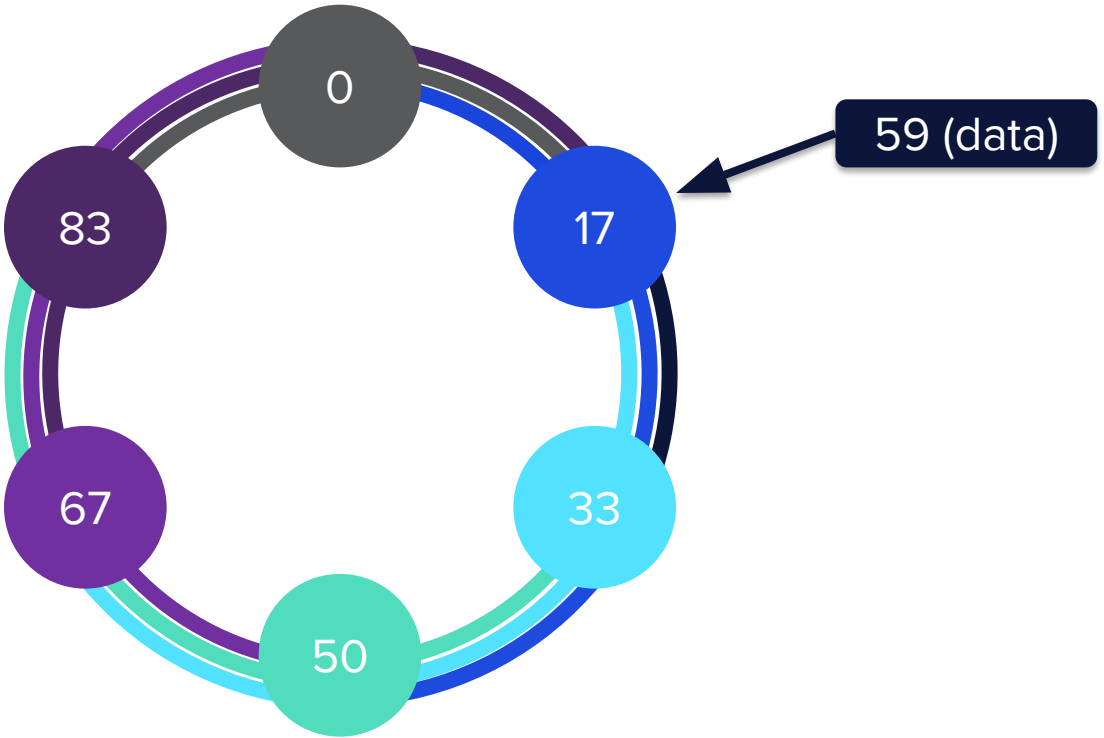
Replication within the Ring

RF = 2



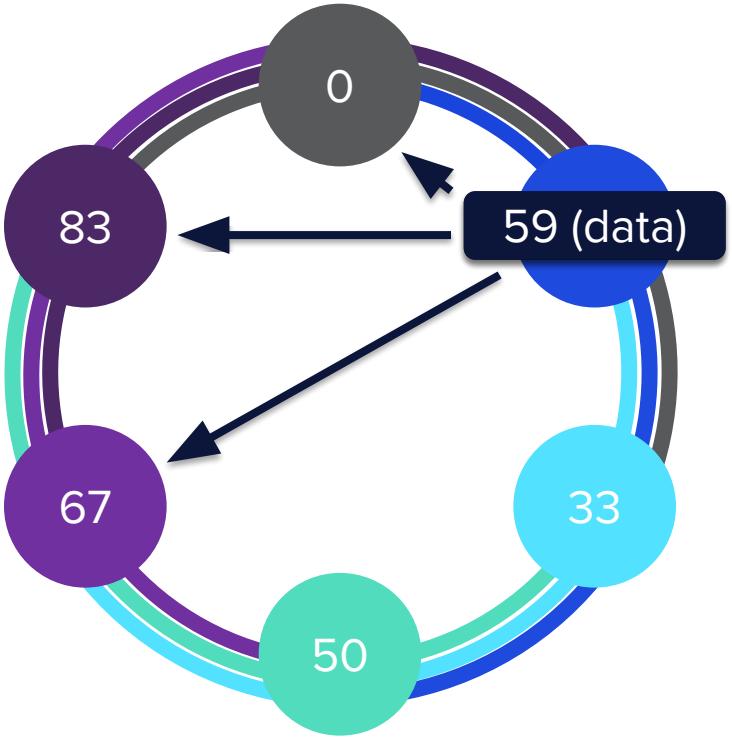
Replication within the Ring

RF = 3



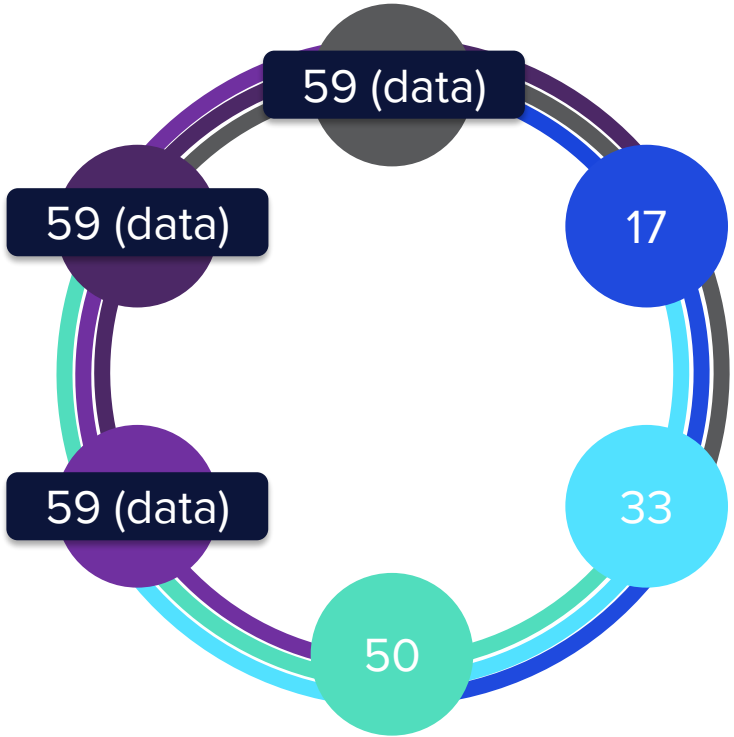
Replication within the Ring

RF = 3



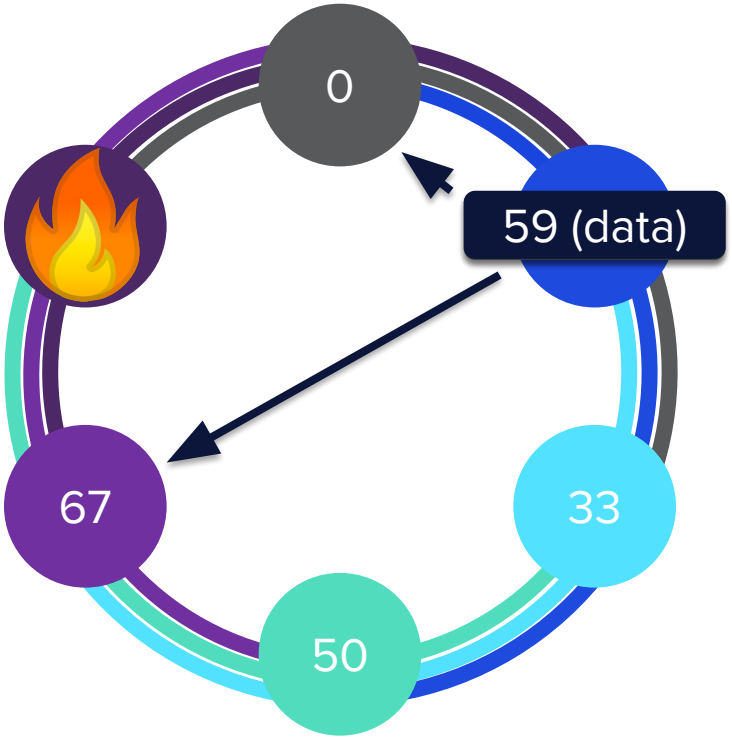
Replication within the Ring

RF = 3



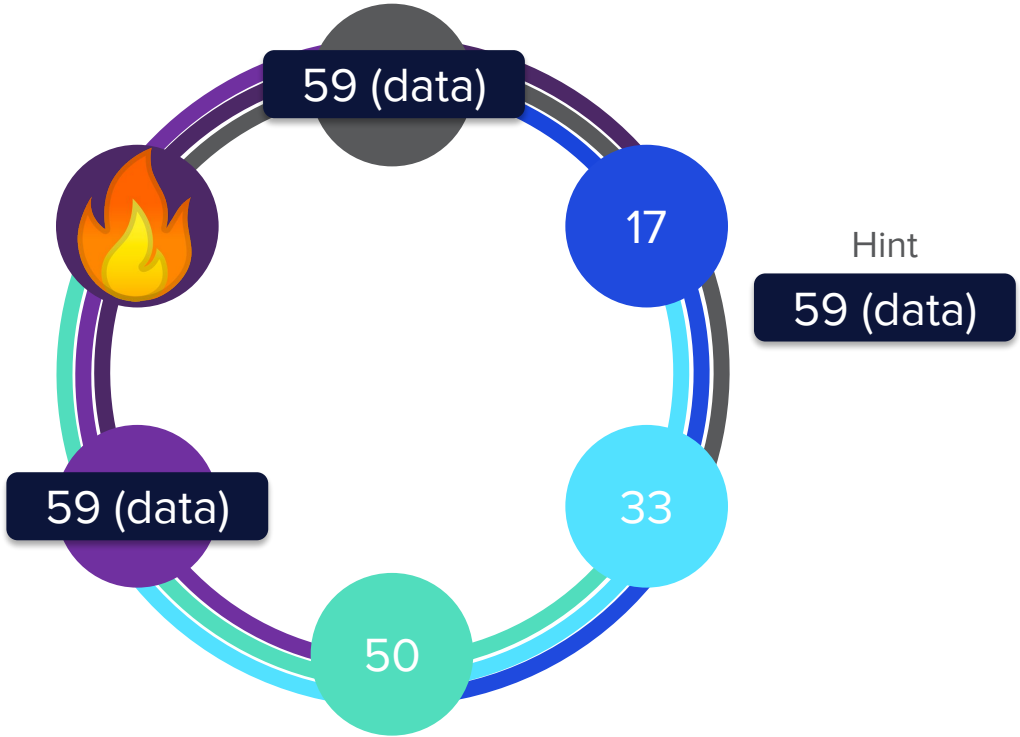
Node Failure

RF = 3

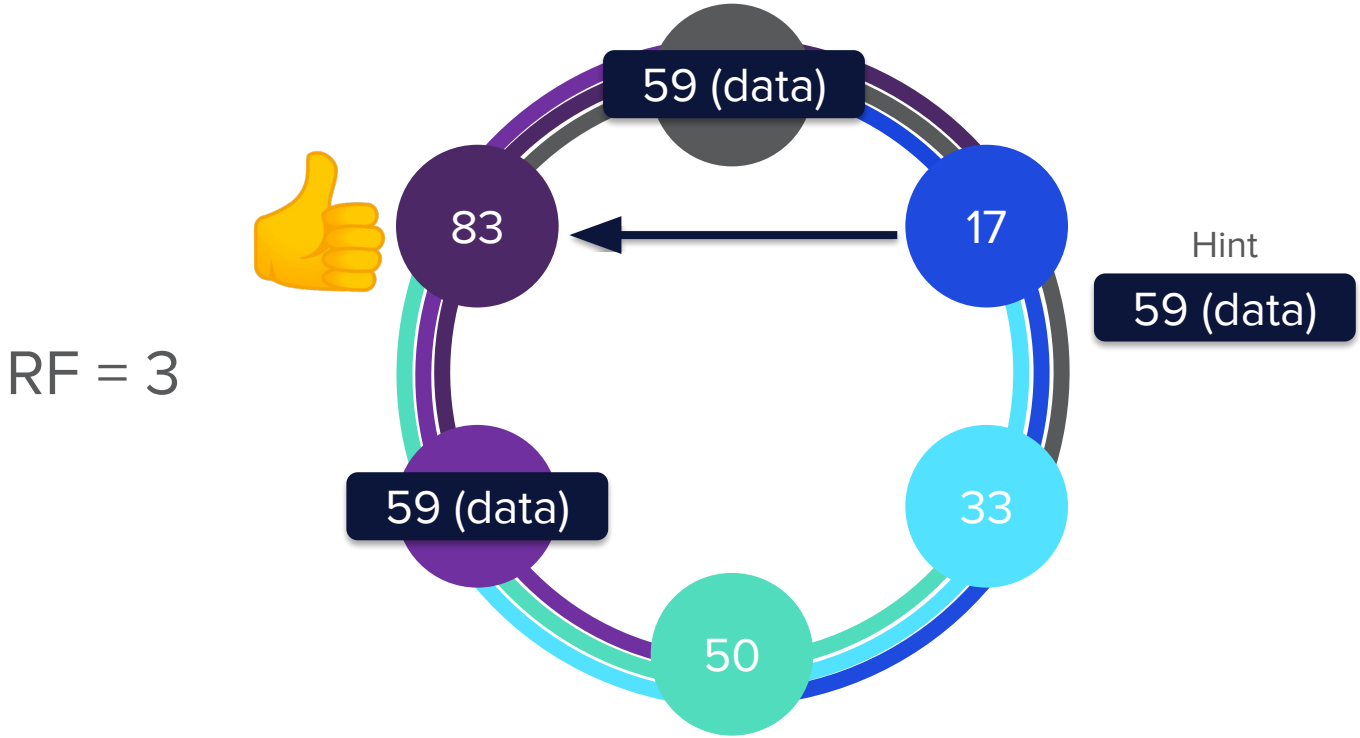


Node Failure

RF = 3

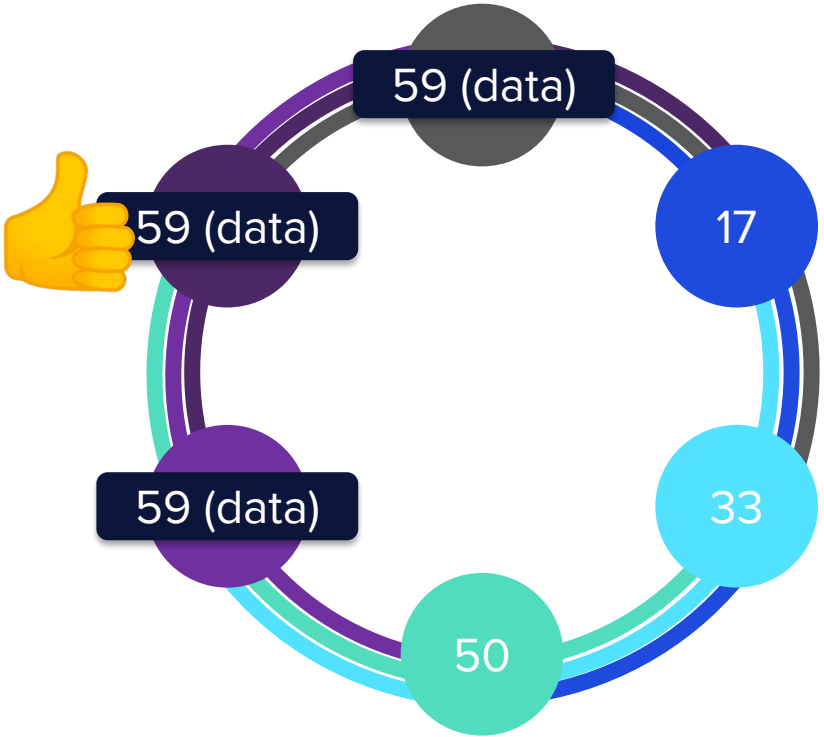


Node Failure



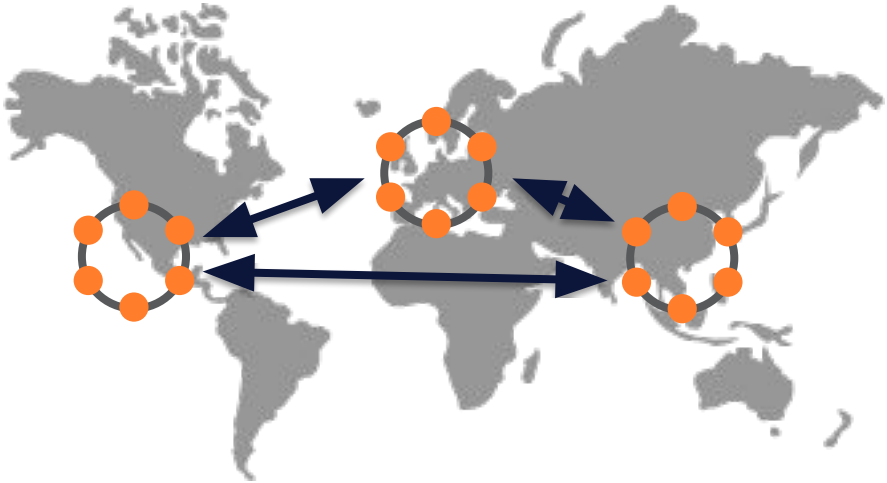
Node Failure – Recovered!

RF = 3

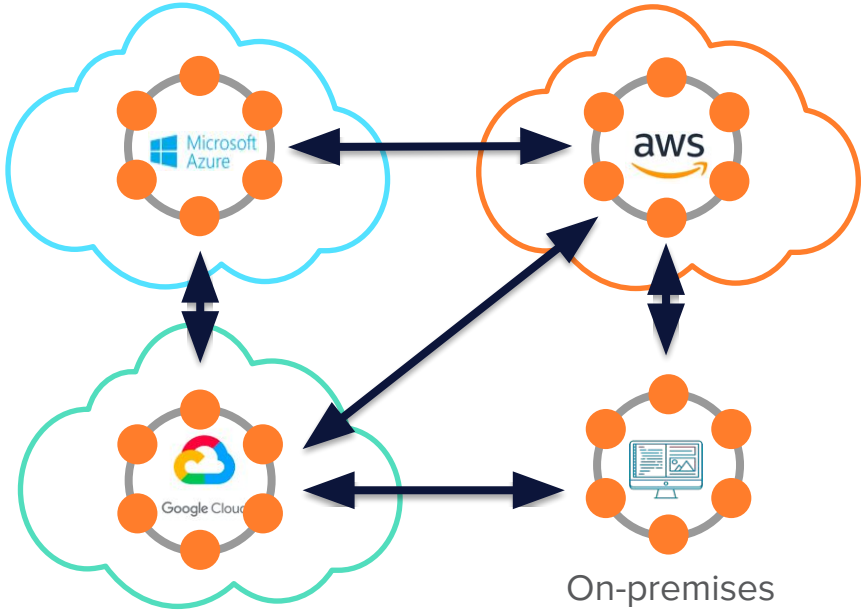


Data Distributed Everywhere

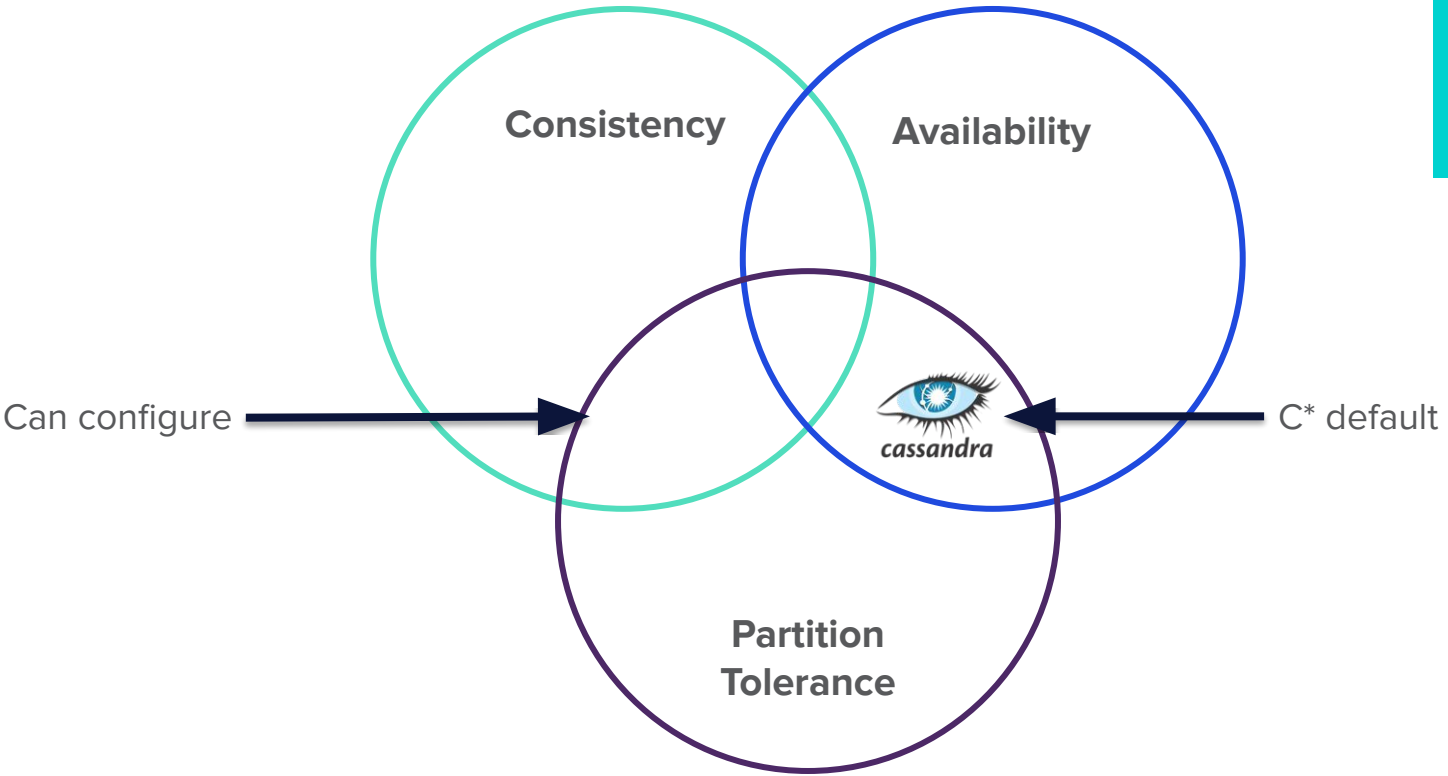
- Geographic Distribution

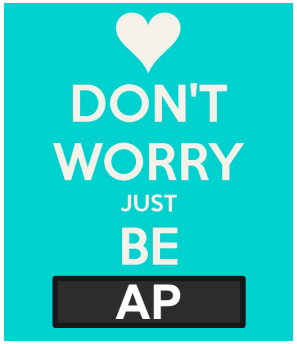


- Hybrid-Cloud and Multi-Cloud

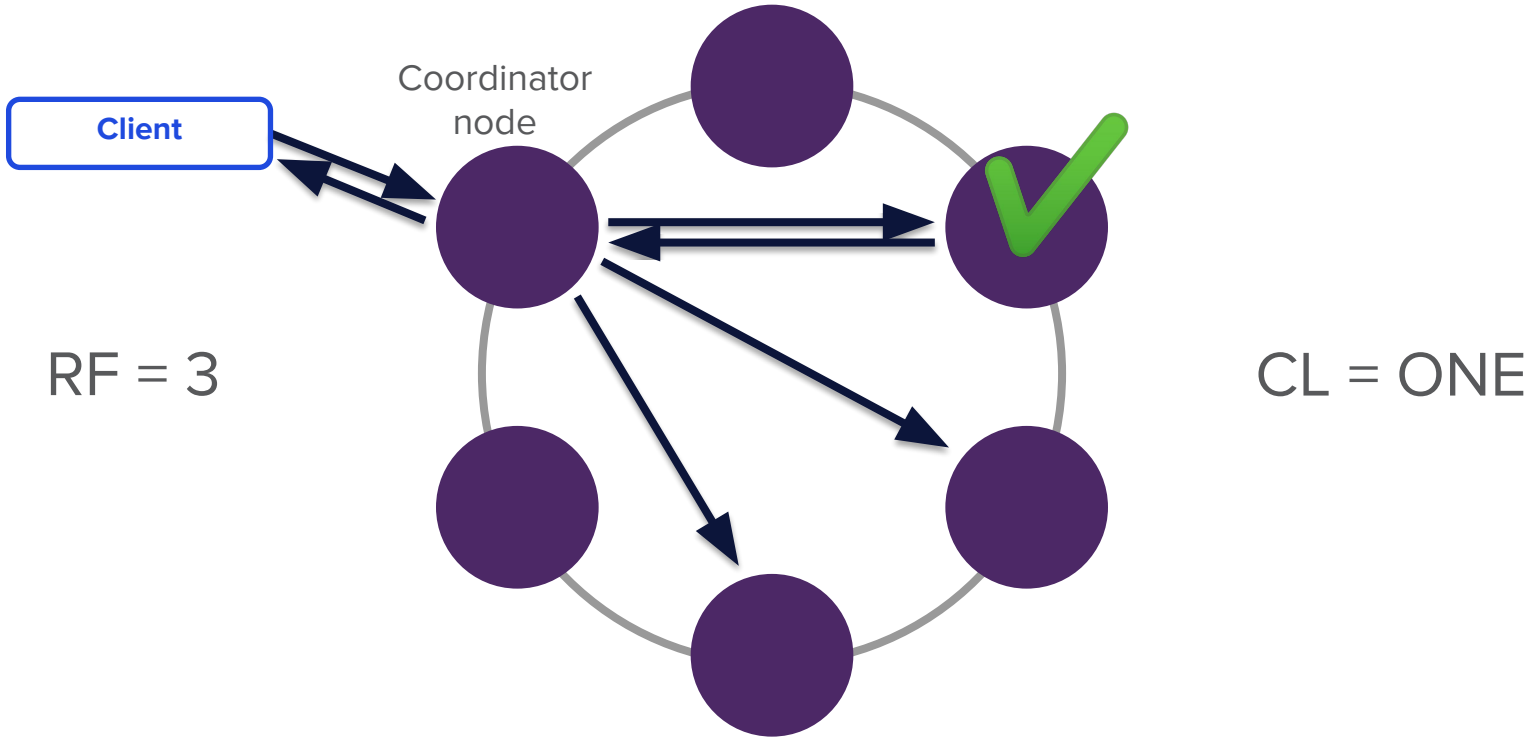


Cap Theorem

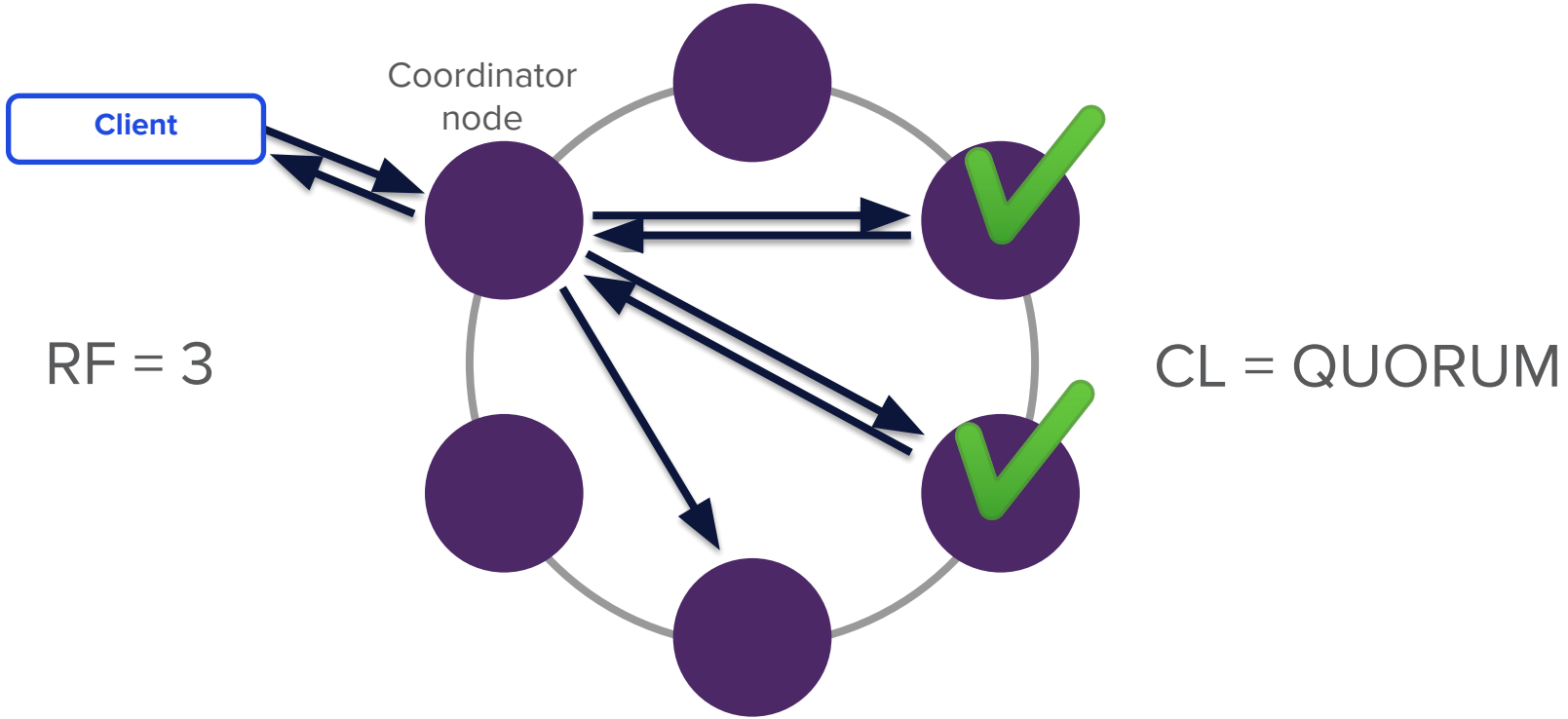

Consistency
Availability
Partition Tolerance
Can configure
C* default
cassandra


DON'T
WORRY
JUST
BE
AP

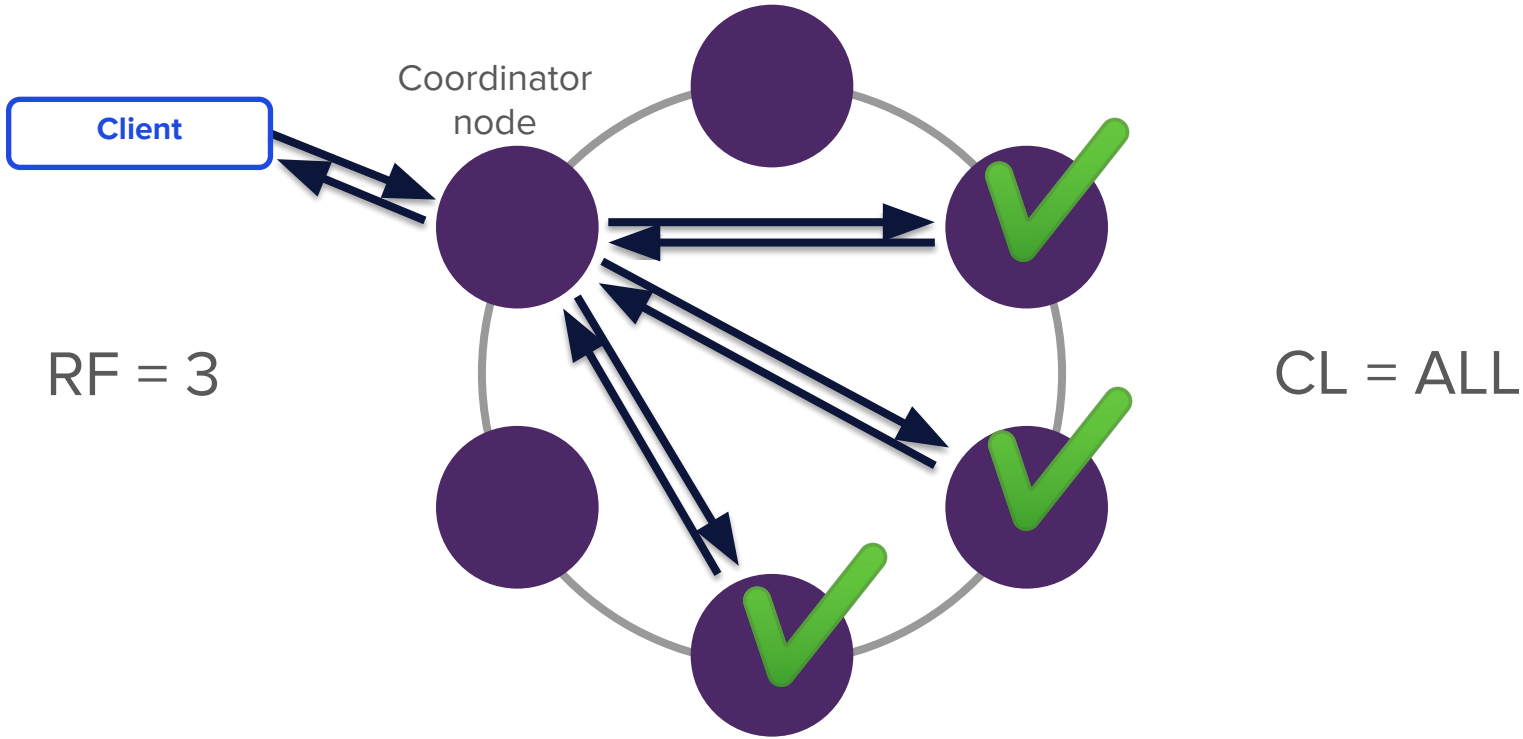
Consistency Levels



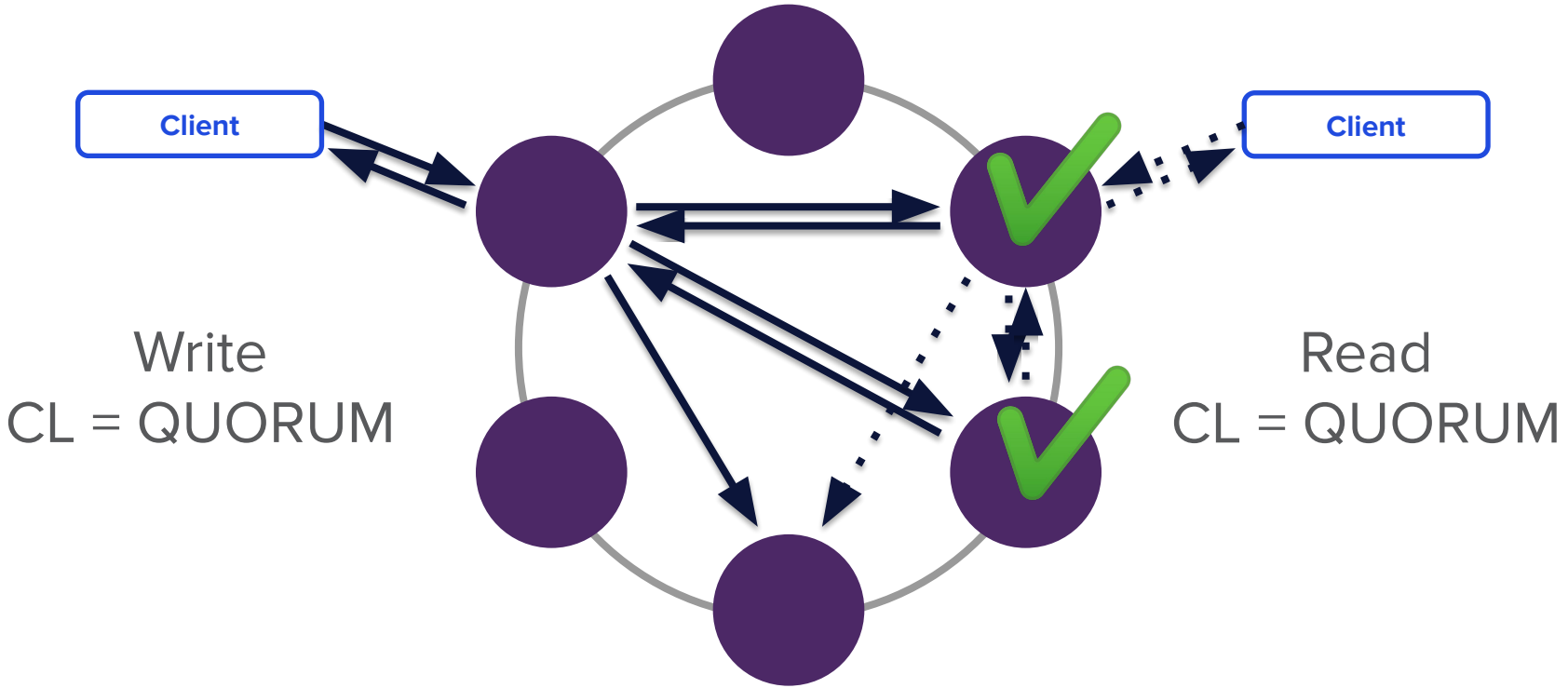
Consistency Levels



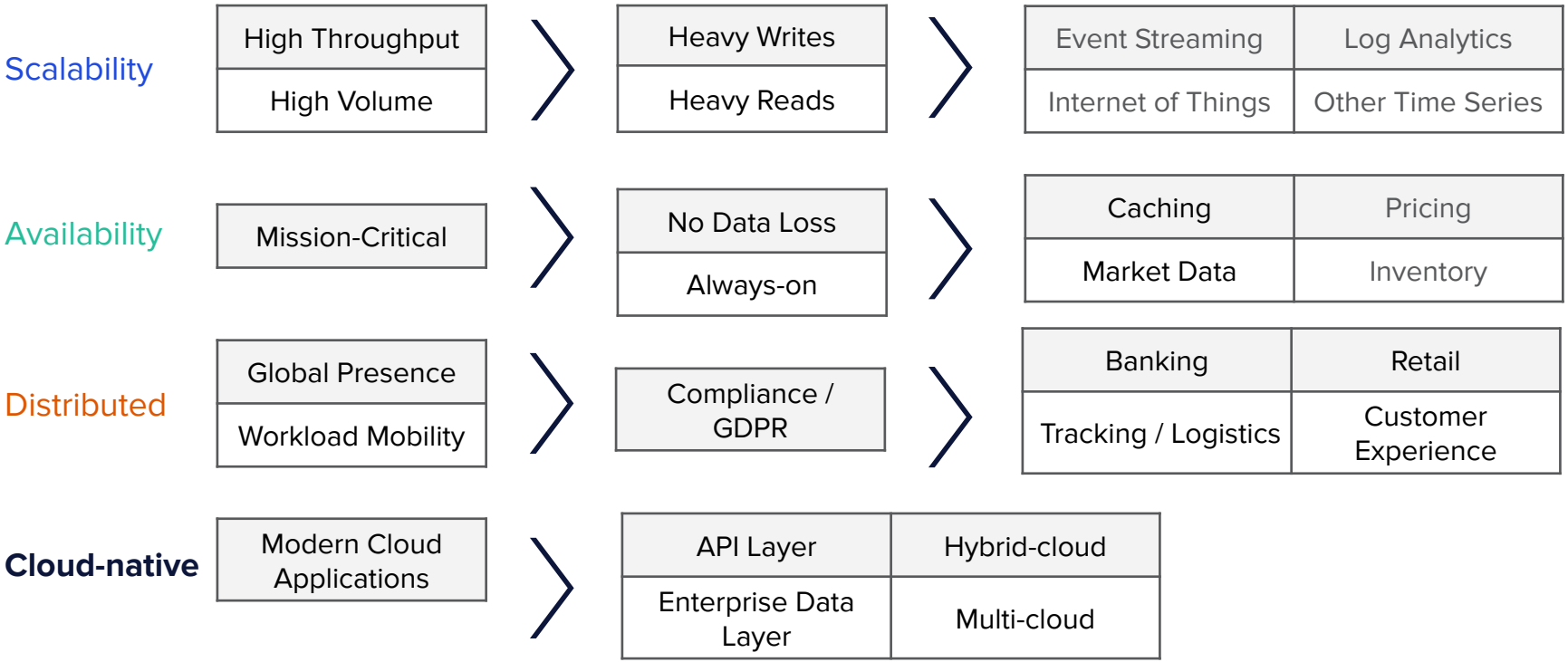
Consistency Levels



Strong Consistency = $CL_{read} + CL_{write} > RF$



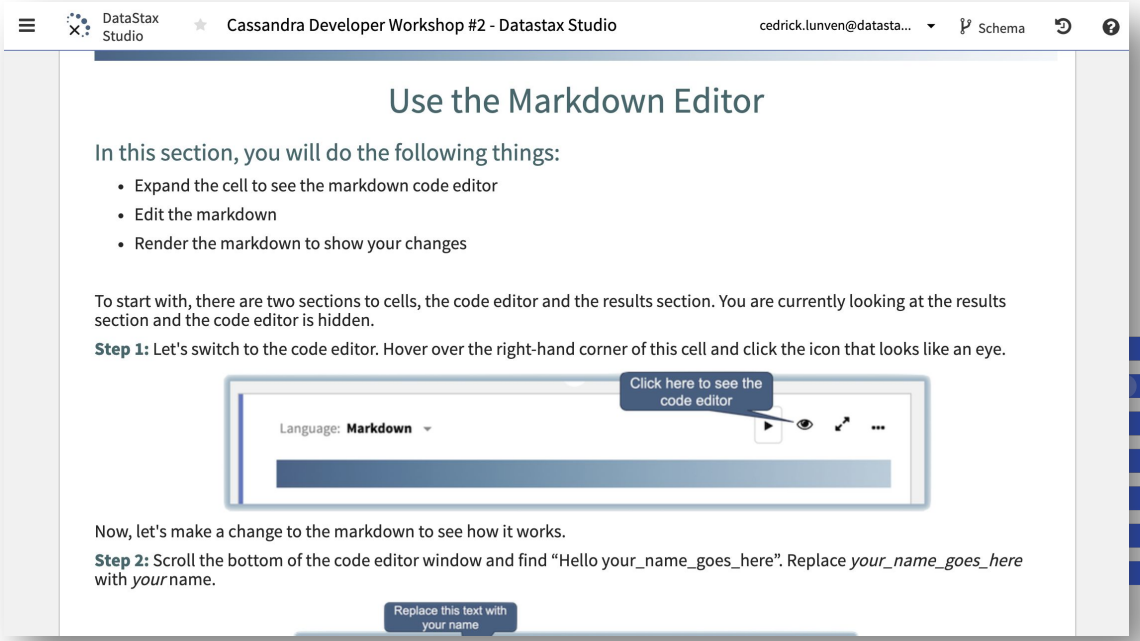
Understanding Use Cases



Exercise 2

DATASTAX

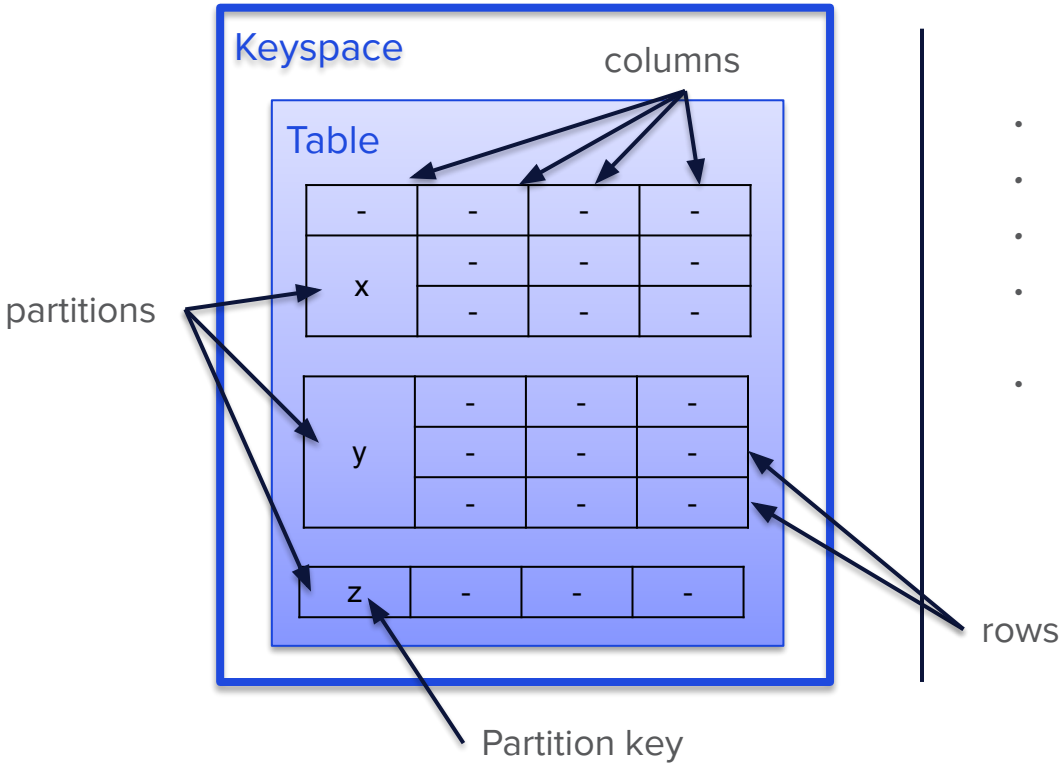
DataStax Studio



Developer Workshop

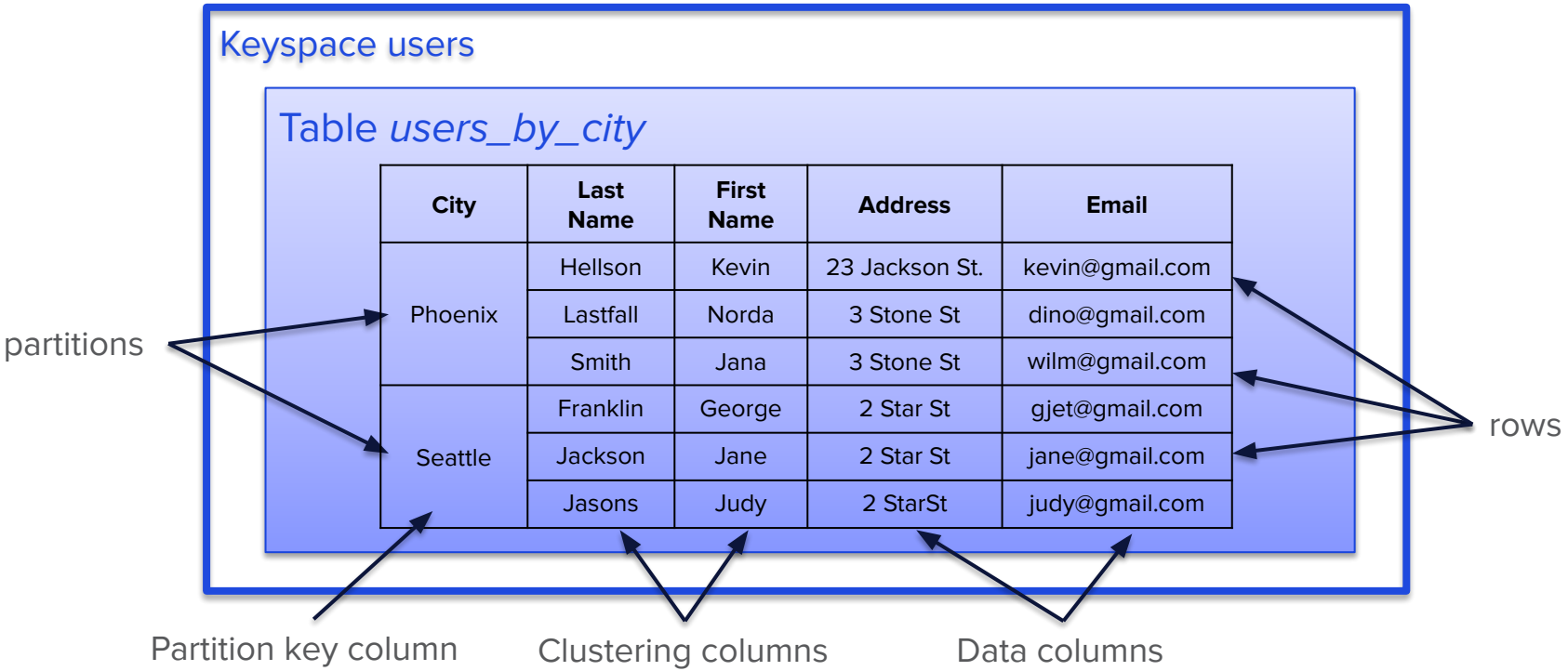
1. Bootstrapping
2. Apache Cassandra™ Why, What & When
3. **Data Modeling with Apache Cassandra™**
4. What's NEXT ?

Cassandra Structure - Partition



- Tabular data model, with one twist
- *Keyspaces* contain *tables*
- *Tables* are organized in *rows* and *columns*
- Groups of related rows called *partitions* are stored together on the same node (or nodes)
- Each row contains a *partition key*
  - One or more columns that are hashed to determine which node(s) store that data

Example Data – Users organized by city



Tables Hold Many Partitions

City	Last Name	First Name	Address	Email
Phoenix	Hellson	Kevin	23 Jackson St.	kevin@gmail.com
	Lastfall	Norda	3 Stone St	dino@gmail.com
	Smith	Jana	3 Stone St	wilm@gmail.com

Table *users_by_city*

Tables Hold Many Partitions

City	Last Name	First Name	Address	Email
Seattle	Franklin	George	2 Star St	gjet@gmail.com
	Jackson	Jane	2 Star St	jane@gmail.com
	Jasons	Judy	2 StarSt	judy@gmail.com

Table *users_by_city*

City	Last Name	First Name	Address	Email
Phoenix	----	----	----	----
	----	----	----	----
	----	----	----	----

Tables Hold Many Partitions

City	Last Name	First Name	Address	Email
Charlotte	Azrael	Chris	5 Blue St	chris@gmail.com
	Stilson	Brainy	7 Azure ln	brain@gmail.com
	Smith	Cristina	4 Teal Cir	clu@gmail.com
	Sage	Grant	9 Royal St	grant@gmail.com
	Seterson	Peter	2 Navy Ct	peter@gmail.com

Table *users_by_city*

City	Last Name	First Name	Address	Email
Phoenix	----	----	----	----
	----	----	----	----
	----	----	----	----

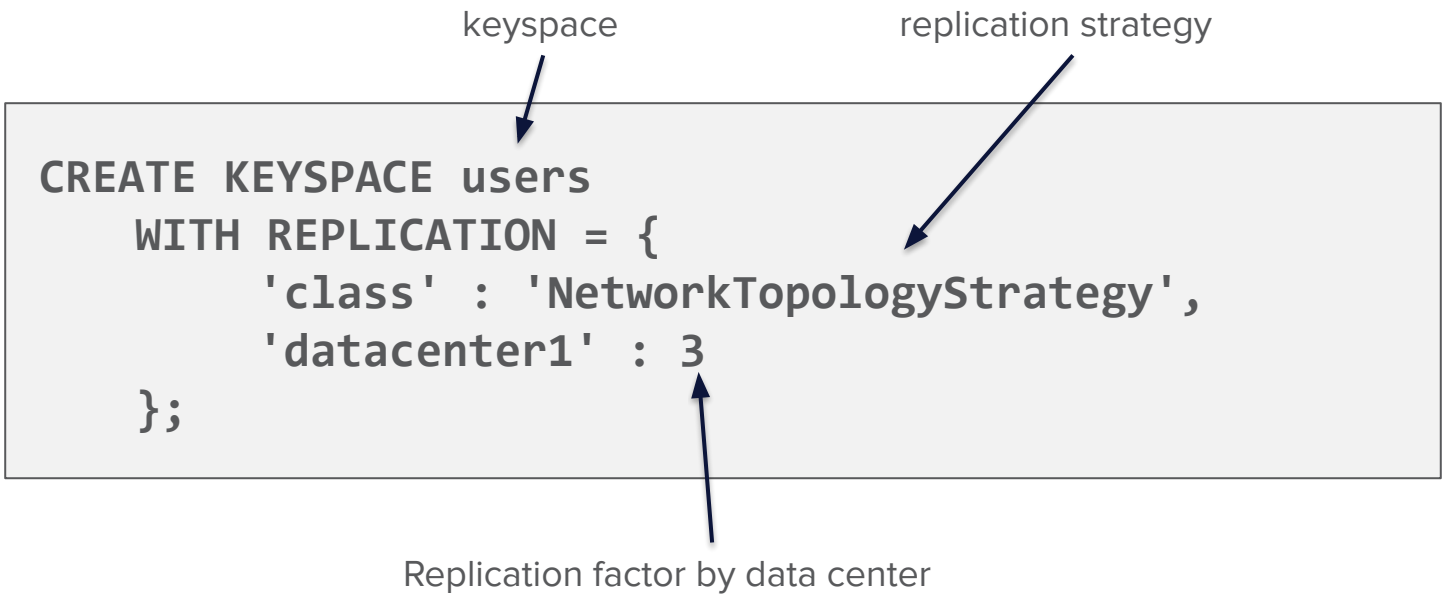
Seattle	----	----	----	----
	----	----	----	----
	----	----	----	----

Tables Hold Many Partitions

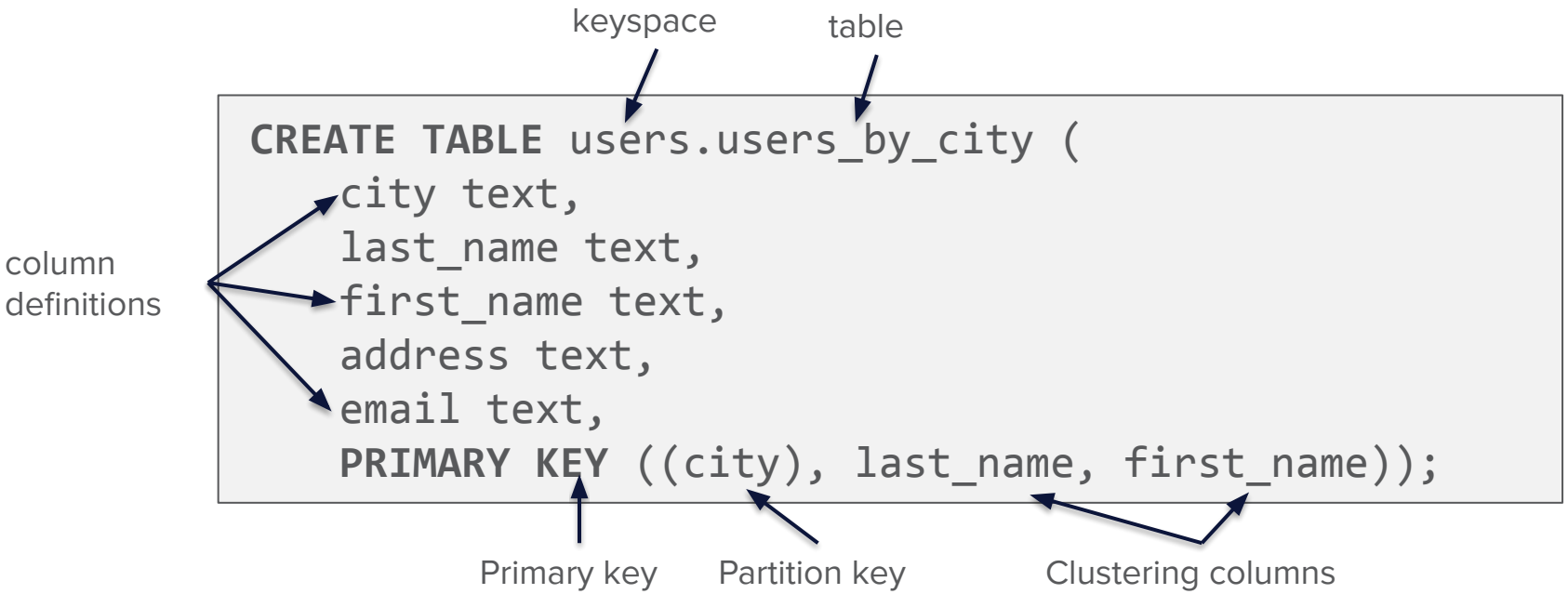
Table *users_by_city*

City	Last Name	First Name	Address	Email
Phoenix	----	----	----	----
	----	----	----	----
	----	----	----	----
Seattle	----	----	----	----
	----	----	----	----
	----	----	----	----
Charlotte	----	----	----	----
	----	----	----	----
	----	----	----	----
	----	----	----	----
	----	----	----	----

Creating a Keyspace in CQL

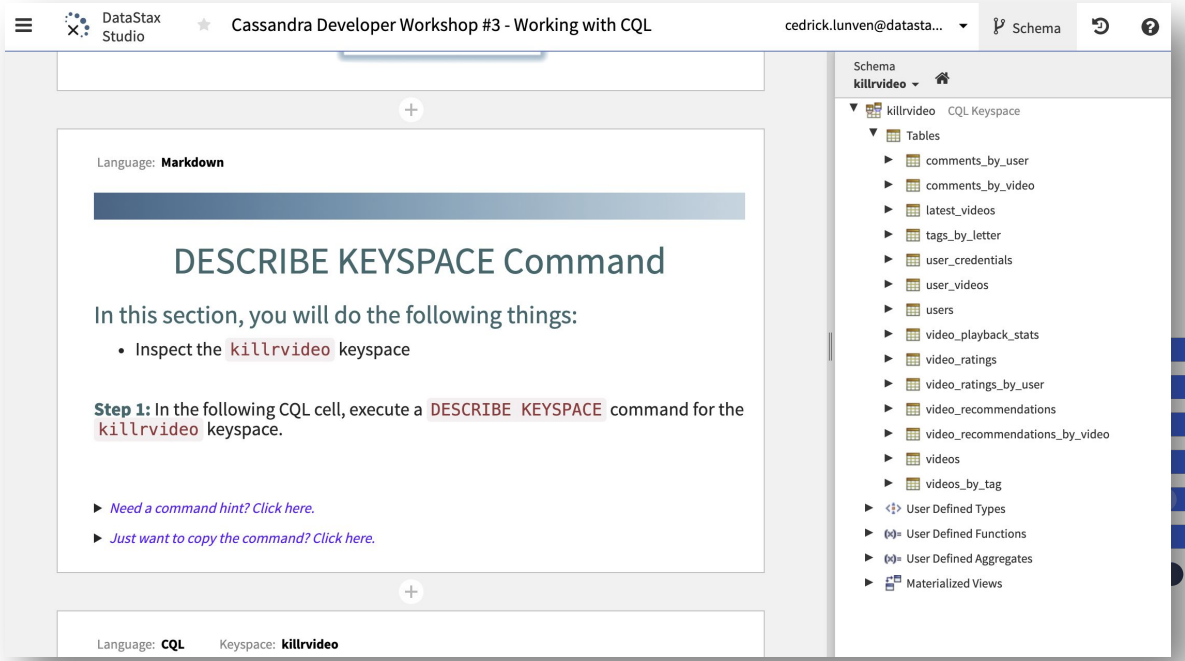


Creating a Table in CQL

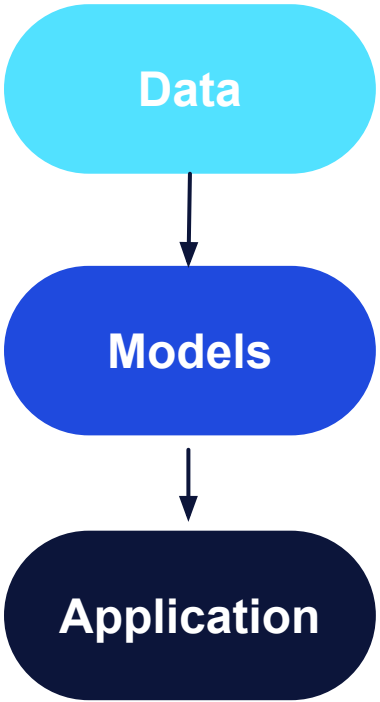


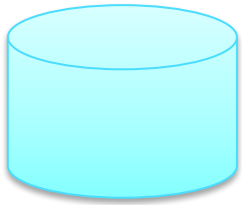
Exercise 3

Working with CQL



Relational Modeling





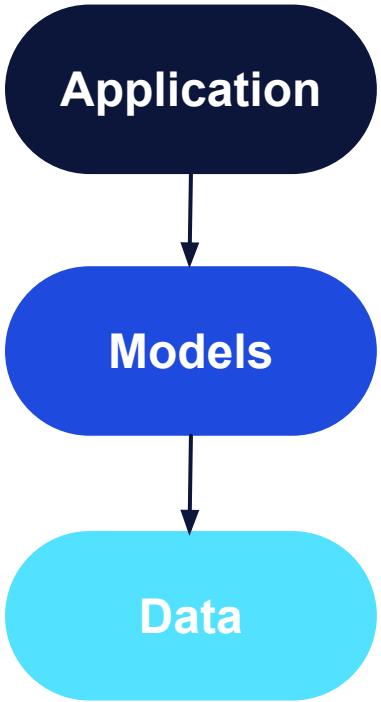
Employees

userId	firstName	lastName
1	Edgar	Codd
2	Raymond	Boyce

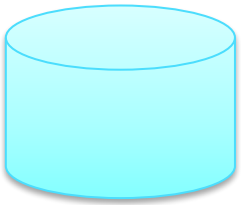
Department

departmentId	department
1	Engineering
2	Math

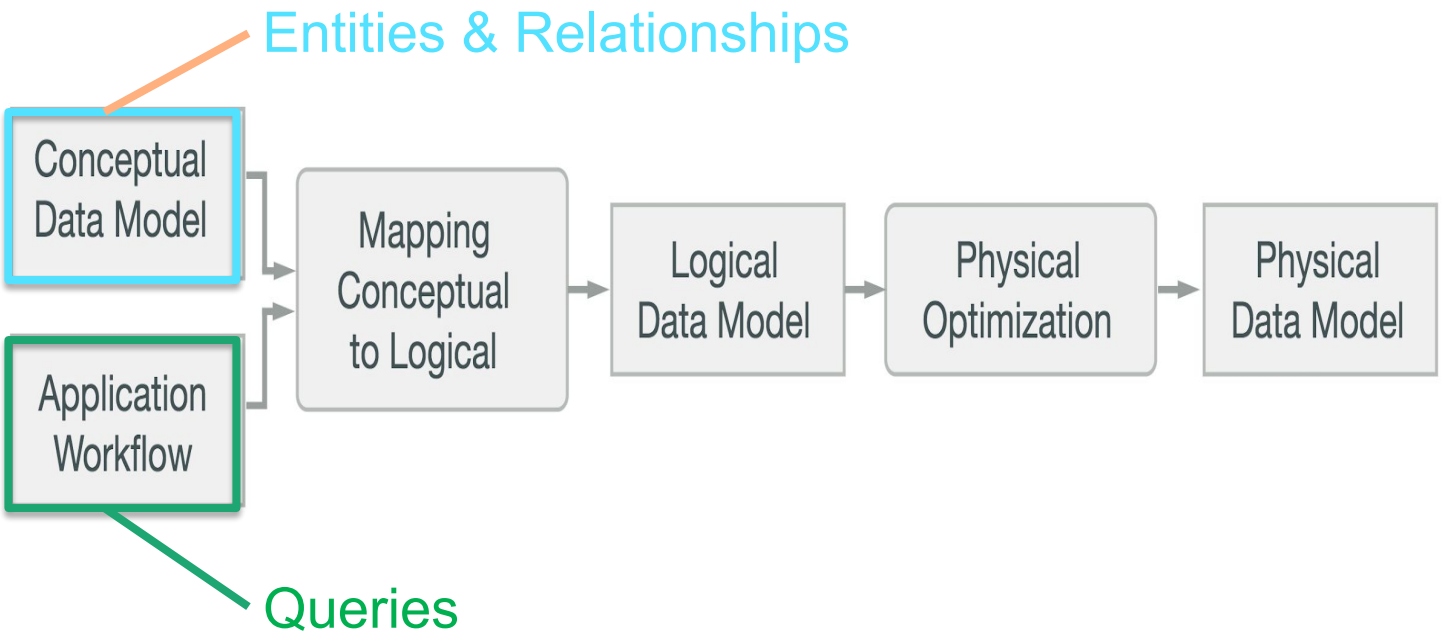
Cassandra Modeling



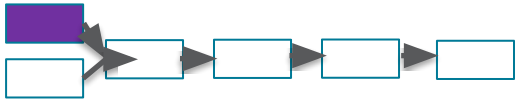
id	firstName	lastName	department
1	Edgar	Codd	Engineering
2	Raymond	Boyce	Math

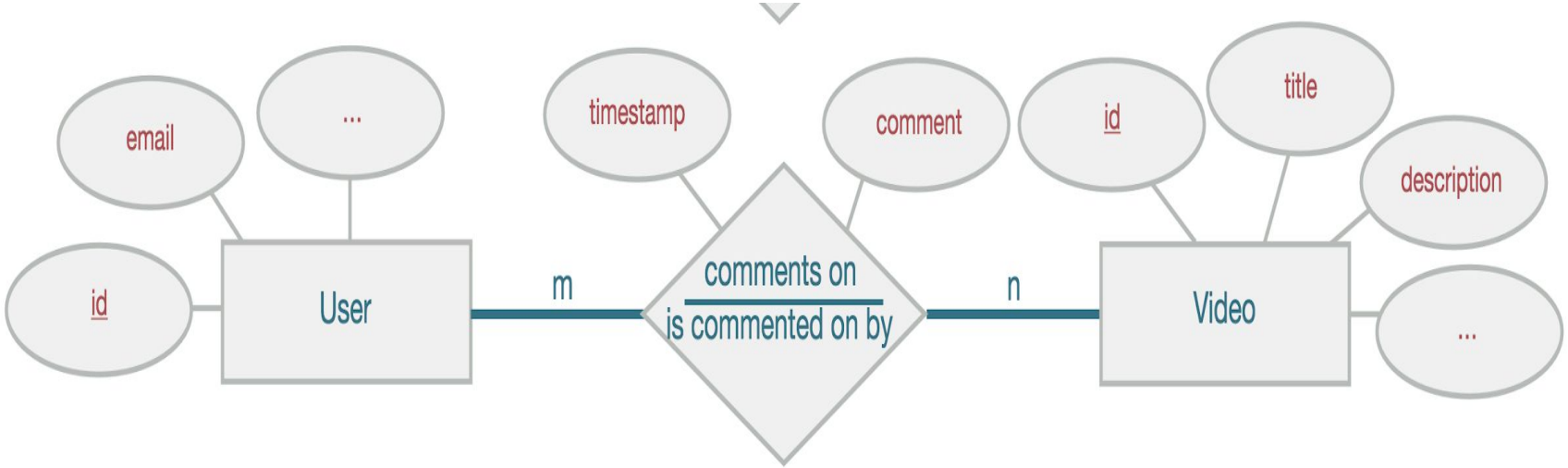


Designing Data Model

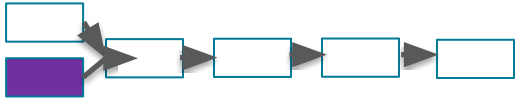


Conceptual Data Model





Application Workflow

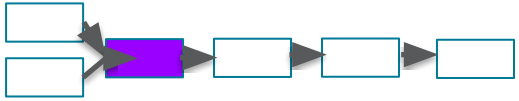


R1: Find comments related to target video using its identifier
- Get most recent first
- Implement Paging

R2: Find comments related to target user using its identifier
- Get most recent first
- Implement Paging

R3: Implement CRUD operations

Mapping



Q1: Find comments for a video with a known id (show most recent first)

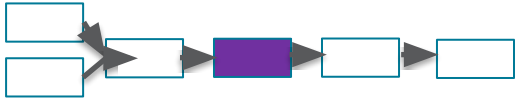
→ comments_by_video

Q2: Find comments posted for a user with a known id (show most recent first)

→ comments_by_user

Q3: CRUD Operations

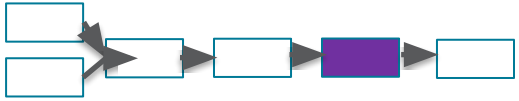
Logical Data Model



comments_by_user	
userid	K
creationdate	C ↓
commentid	C ↑
videoid	
comment	

comments_by_video	
videoid	K
creationdate	C ↓
commentid	C ↑
userid	
comment	

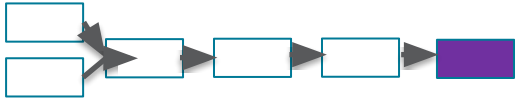
Physical Data Model



comments_by_user		
userid	UUID	K
commentid	TIMEUUID	C ↓
videoid	UUID	
comment	TEXT	

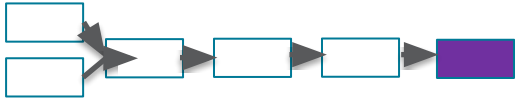
comments_by_video		
videoid	UUID	K
commentid	TIMEUUID	C ↓
userid	UUID	
comment	TEXT	

Schema DDL



```
CREATE TABLE IF NOT EXISTS comments_by_user (
    userid uuid,
    commentid timeuuid,
    videoid uuid,
    comment text,
    PRIMARY KEY ((userid), commentid)
) WITH CLUSTERING ORDER BY (commentid DESC);


CREATE TABLE IF NOT EXISTS comments_by_video (
    videoid   uuid,
    commentid timeuuid,
    userid    uuid,
    comment   text,
    PRIMARY KEY ((videoid), commentid)
) WITH CLUSTERING ORDER BY (commentid DESC);
```

Queries

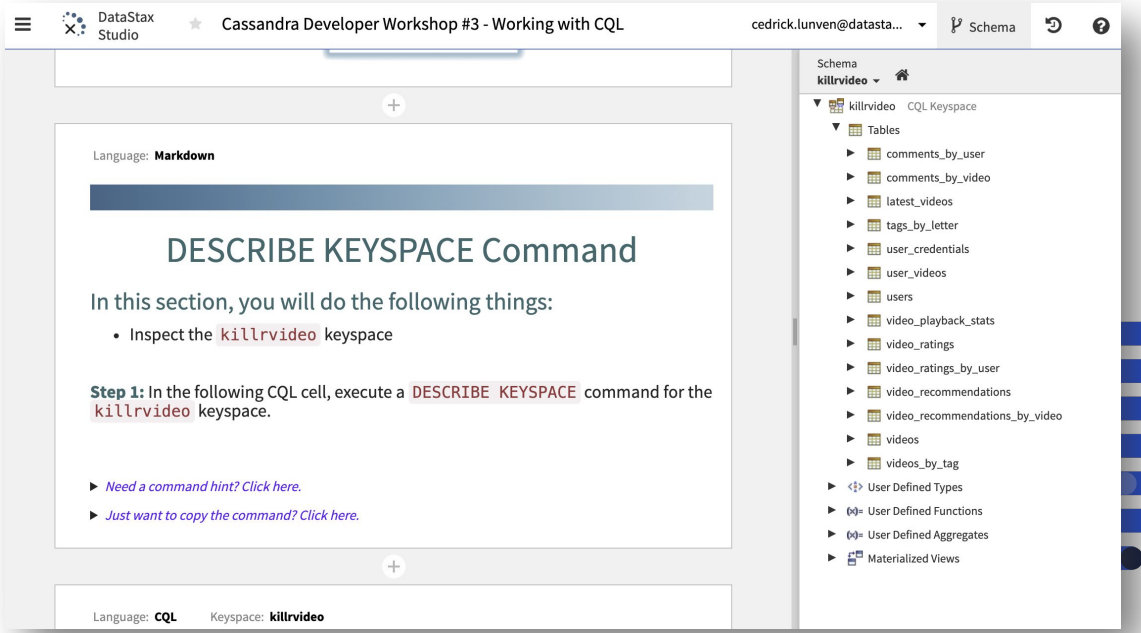


```
SELECT * FROM comments_by_user
WHERE userid = <some UUID>
```

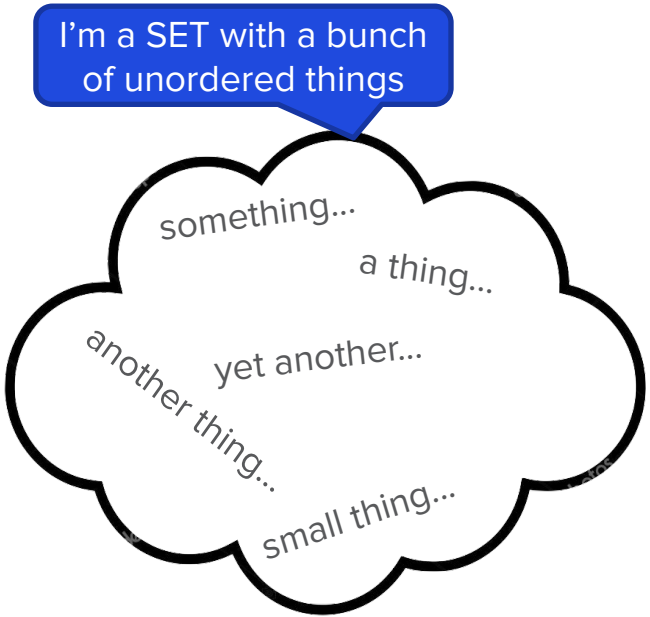
```
SELECT * FROM comments_by_video
WHERE videoid = <some UUID>
```

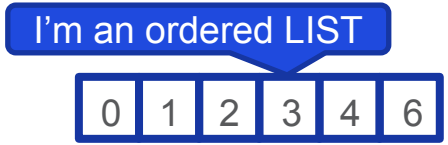
Exercise 4

Data Modelling



Collections – Three Types



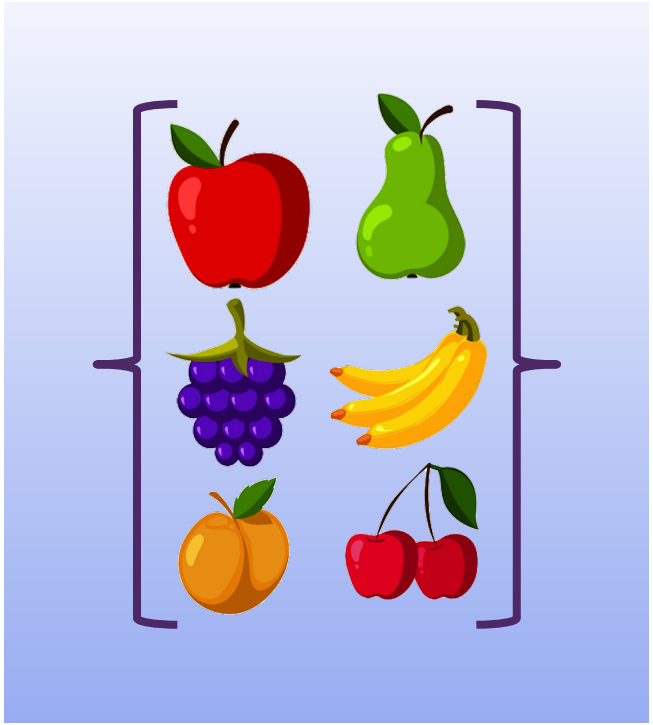


I'm a MAP of key/value pairs

Key	Value
K1	V1
K2	V2
K3	V3
K4	V4
K5	V5

Example Table with Set

```
CREATE killrvideo.videos (
  videoid                 uuid,
  userid                  uuid,
  name                    text,
  description             text,
  location                text,
  location_type           int,
  preview_image_location  text,
  tags                    set<text>,
  added_date              timestamp,
  PRIMARY KEY (videoid)
);
```



Example Set Operations

```
INSERT INTO killrvideo.videos (videoid, tags)
  VALUES(12345678-1234-1234-1234-123456789012,
  {'Side-splitter', 'Short'});
```

Insert

```
UPDATE killrvideo.videos
  SET tags = {'Dark', 'Sad'}
  WHERE videoid = 12345678-1234-1234-1234-123456789012;
```

Replace entire set

```
UPDATE killrvideo.videos
  SET tags = tags + {'Enthralling'}
  WHERE videoid = 12345678-1234-1234-1234-123456789012;
```

Add to set

Example Table with List

```
CREATE killrvideo.actors_by_video (
  videoid  uuid,
  actors   list<text>, // alphabetical list of actors
  PRIMARY KEY (videoid)
);
```

Example List Operations

```
INSERT INTO killrvideo.actors_by_video (videoid, actors)
  VALUES(12345678-1234-1234-1234-123456789012,
  ['Adams', 'Baker', 'Cox']);
```

Insert

```
UPDATE killrvideo.actors_by_video
  SET actors = ['Arthur', 'Beverly']
  WHERE videoid = 12345678-1234-1234-1234-123456789012;
```

Replace entire list

```
UPDATE killrvideo.actors_by_video
  SET actors = actors + ['Crawford']
  WHERE videoid = 12345678-1234-1234-1234-123456789012;
```

Append

Another Example List Operation

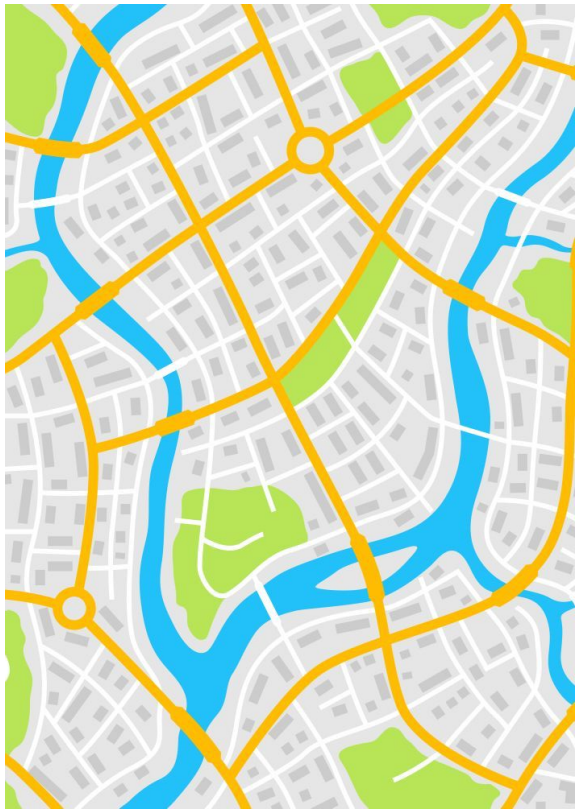
```
UPDATE killrvideo.actors_by_video
  SET actors[1] = 'Brown'
  WHERE videoid = 12345678-1234-1234-1234-123456789012;
```

Replace an element

Note: replacing an element requires a read-before-write, which implies performance penalty.

Example Table with Map

```
CREATE TABLE killrvideo.users(
   userid       uuid,
   phone_nos    map<text, text>,
   PRIMARY KEY (userid)
);
```



Example Map Operations

```
INSERT INTO killrvideo.users (userid, phone_nos)
  VALUES(12345678-1234-1234-1234-123456789012,
  {'cell':'867-5309', 'home':'555-1212',
  'busi':'800-555-1212'});
```

Insert

```
UPDATE killrvideo.users
  SET phone_nos = {'cell':'867-5310', 'office':'555-1212'}
  WHERE userid = 12345678-1234-1234-1234-123456789012;
```

Replace entire map

```
UPDATE killrvideo.users
  SET phone_nos = phone_nos + {'desk': '270-555-1213'}
  WHERE userid = 12345678-1234-1234-1234-123456789012;
```

Add to map

Example User Defined Type (UDT)

```
CREATE TYPE killrvideo.address(
  street text,
  city   text,
  state  text,
);
```

```
CREATE TABLE killrvideo.users(
  userid       uuid,
  location     address,
  PRIMARY KEY (userid)
);
```

Example UDT Operations

Insert

```
INSERT INTO killrvideo.users (userid, location)
  VALUES(12345678-1234-1234-1234-123456789012,
  {street:'123 Main', city:'Metropolis', state:'CA'});
```

Replace entire UDT

```
UPDATE killrvideo.users
  SET location = {street:'234 Elm', city:'NYC', state:'NY'}
  WHERE userid = 12345678-1234-1234-1234-123456789012;
```

Replace one UDT field

```
UPDATE killrvideo.users
  SET location.city = 'Albany'
  WHERE userid = 12345678-1234-1234-1234-123456789012;
```

Another Example UDT Operation

```
SELECT location.city FROM killrvideo.users
  WHERE userid = 12345678-1234-1234-1234-123456789012;
```

Select field

Counters

- 64-bit signed integer
- Use-case:
  - Imprecise values such as likes, views, etc.
- Two operations:
  - Increment
  - Decrement
  - First op assumes the value is zero

Counter Limitations

- Cannot be part of primary key
- Counters not mixed with other types in table
- Value cannot be set
- Rows with counters cannot be inserted
- Updates are not idempotent
  - Counters should *not* be used for precise values

Example Table with Counter

```
CREATE TABLE killrvideo.video_playback_stats (
    videoid uuid,
    views counter,
    PRIMARY KEY (videoid)
);
```

Counter Updates

Incrementing a counter:

This format must be observed

This can be an integer value

```
UPDATE killrvideo.videos SET views = views + 1
  WHERE videoid = 12345678-1234-1234-1234-123456789012;
```

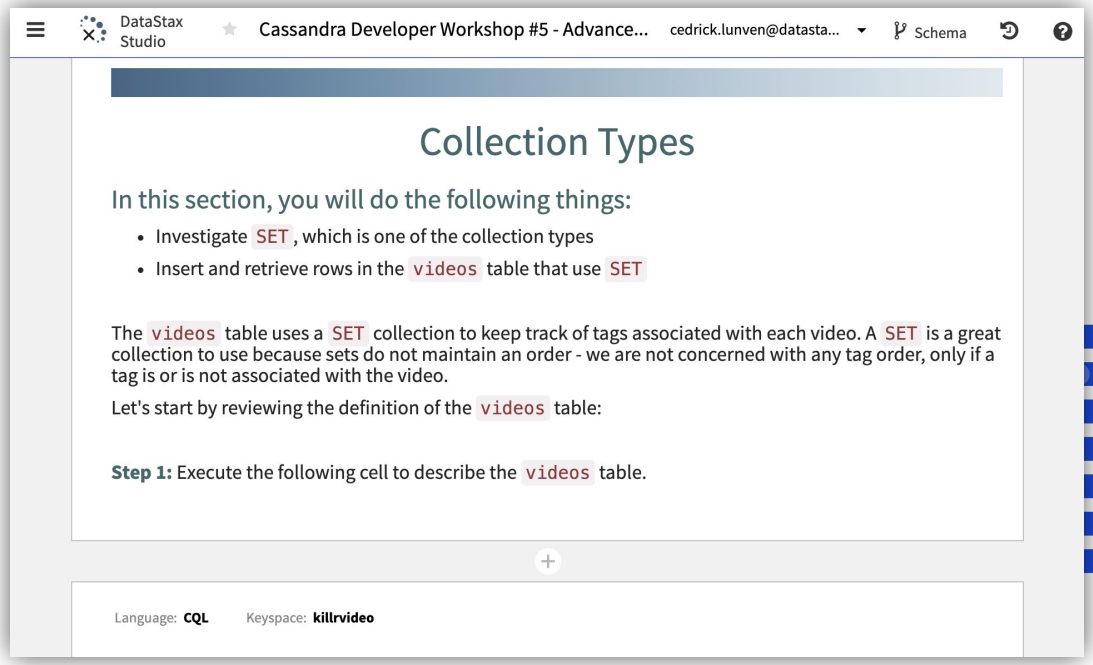
Decrementing a counter:

Just change the sign

```
UPDATE killrvideo.videos SET views = views - 1
  WHERE videoid = 12345678-1234-1234-1234-123456789012;
```

Exercise 5

Advanced Type



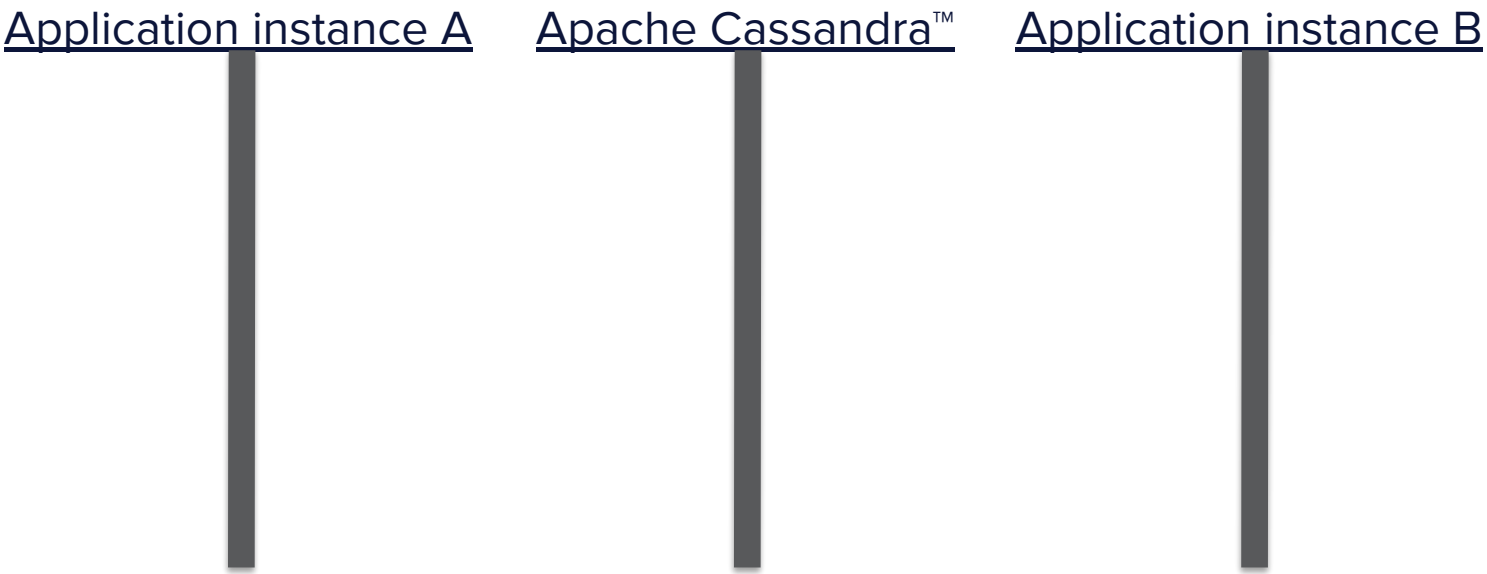
Cassandra Lightweight Transactions

- Sometimes insert or update operations must be unique
  - Requiring a read-before-write
- CQL Lightweight Transactions (LWT) solve this problem
- Uses an IF clause on inserts and updates

WARNING: The read-before-write has performance implications – Use wisely!

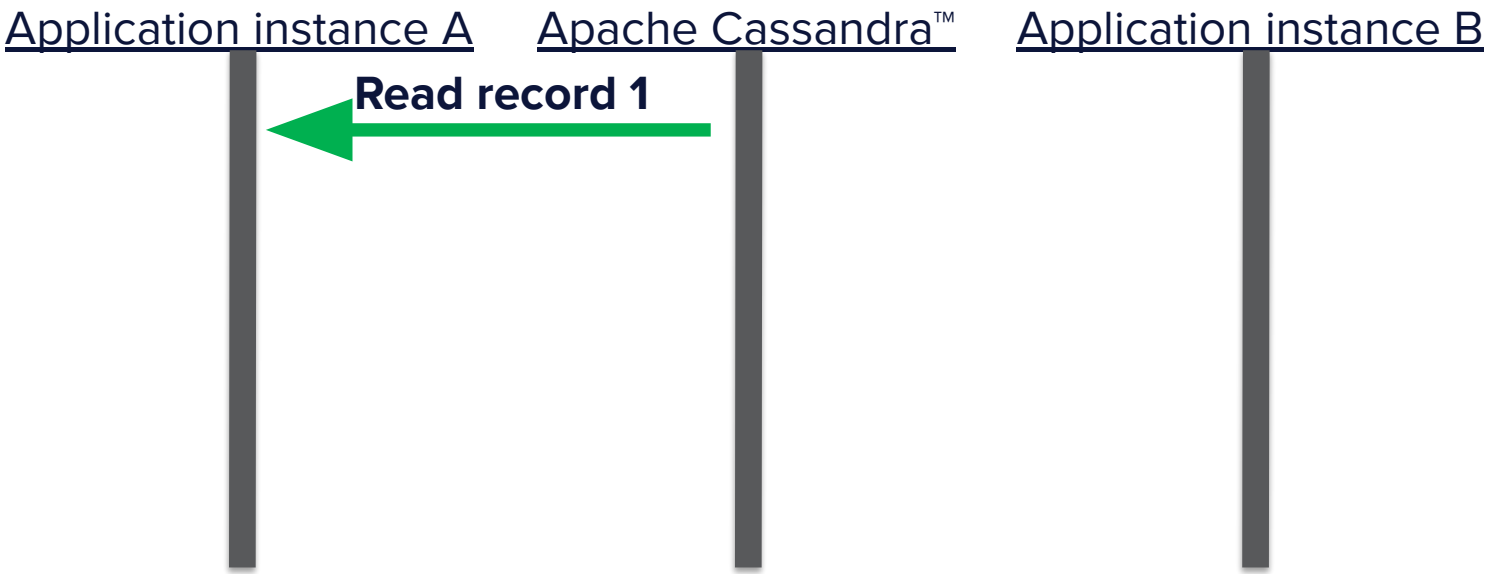
Cassandra Lightweight Transactions

The problem



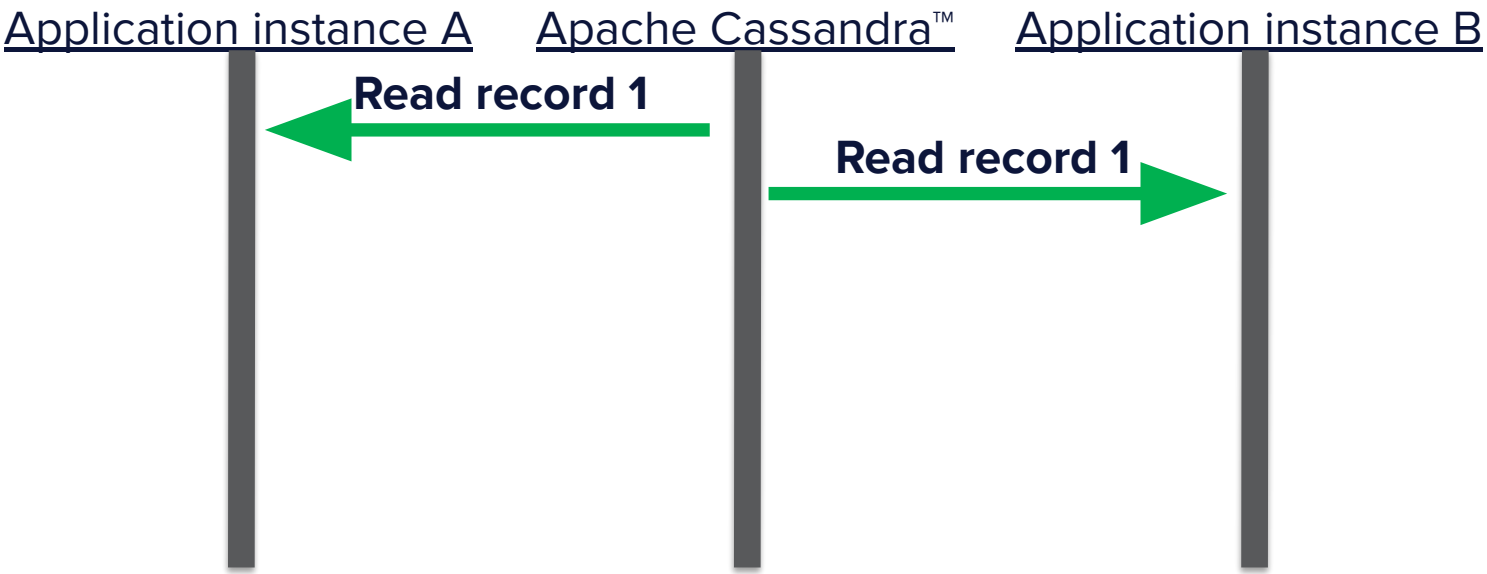
Cassandra Lightweight Transactions

The problem



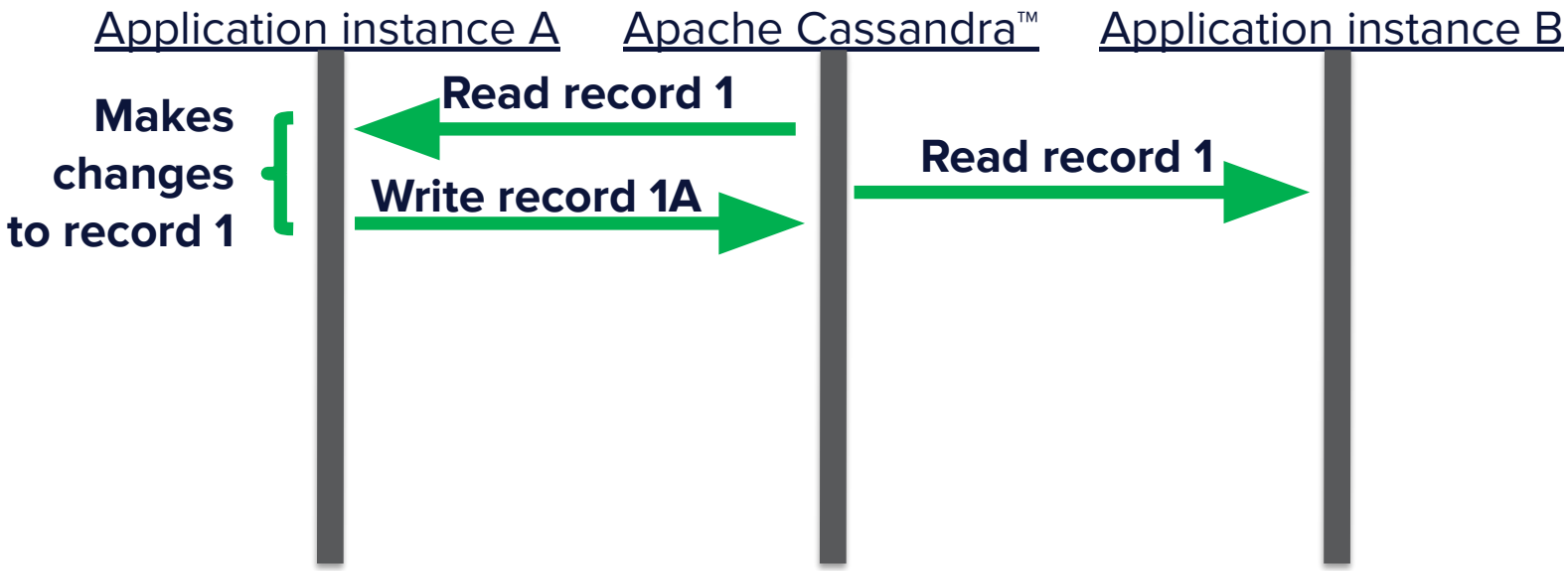
Cassandra Lightweight Transactions

The problem



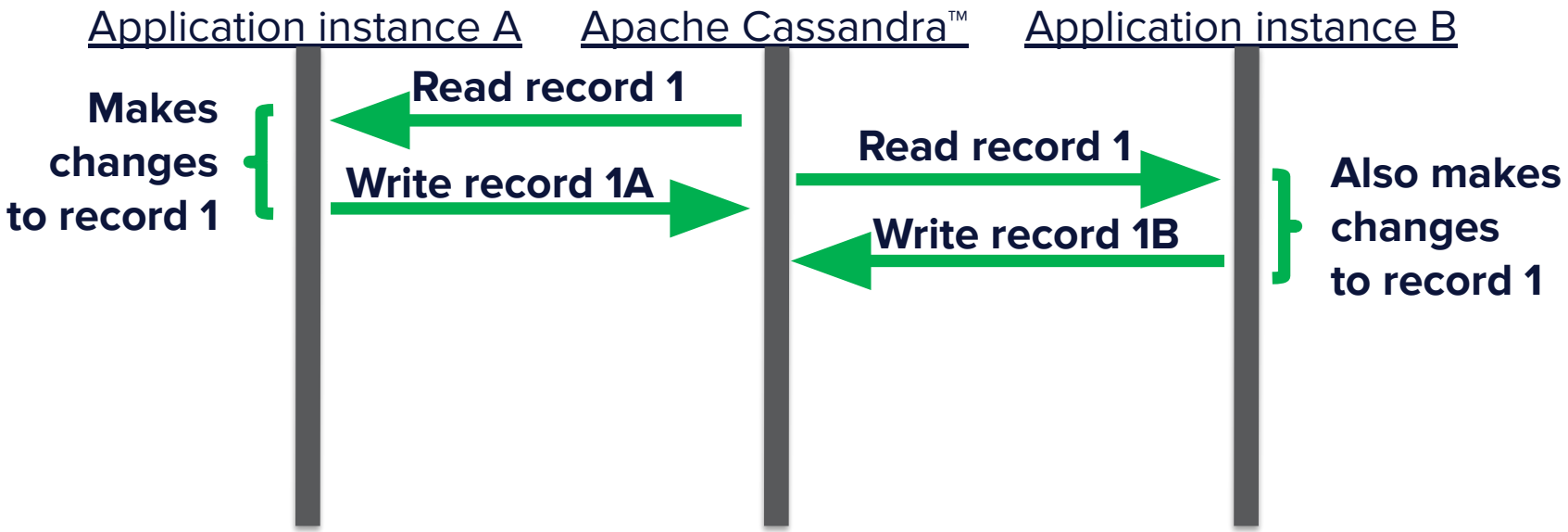
Cassandra Lightweight Transactions

The problem



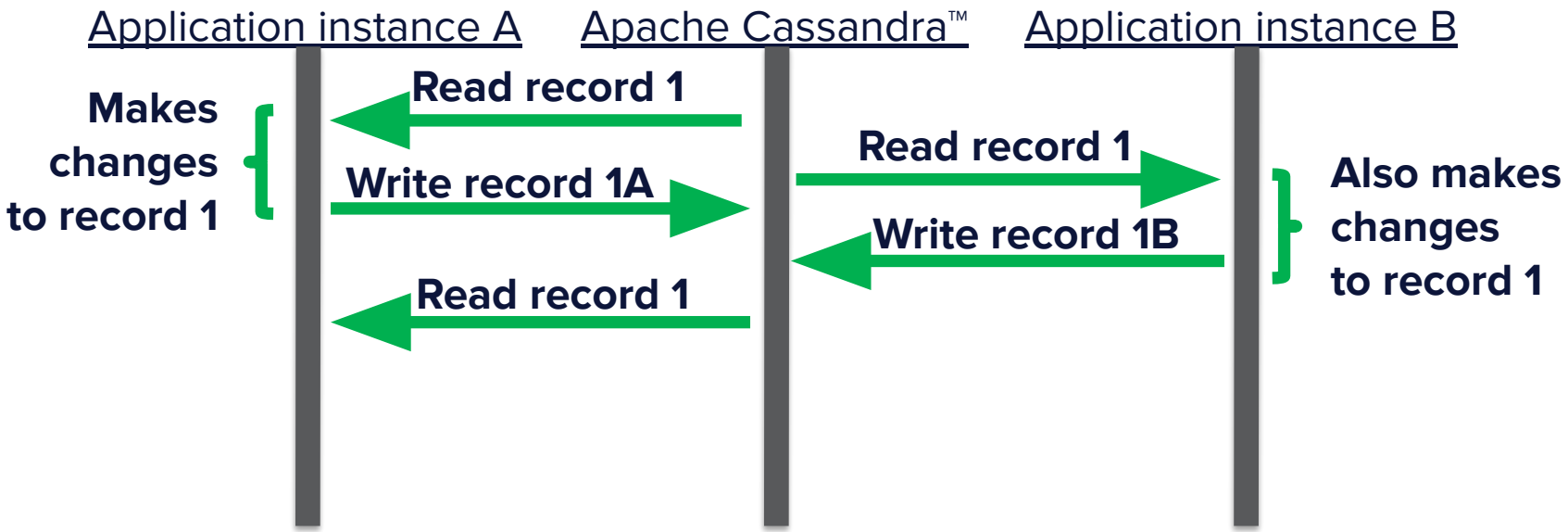
Cassandra Lightweight Transactions

The problem



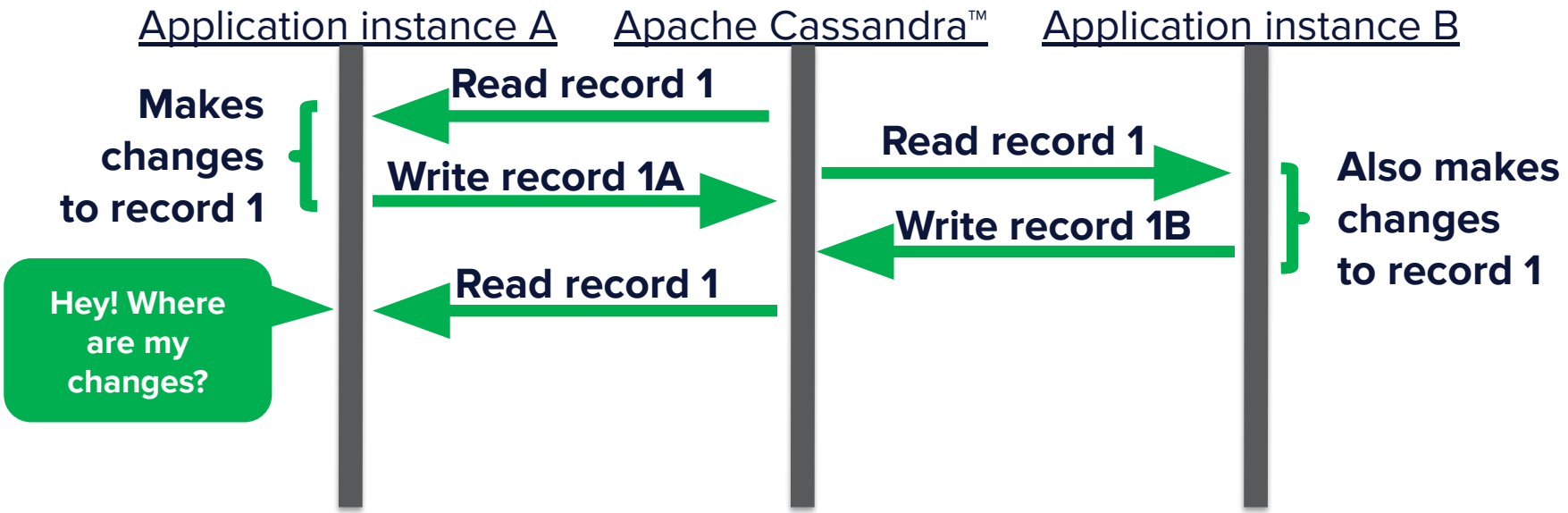
Cassandra Lightweight Transactions

The problem



Cassandra Lightweight Transactions

The problem



Cassandra Lightweight Transactions

```
KVUser@cqlsh> SELECT * FROM killrvideo.user_credentials WHERE email = 'new_user@gmail.com';

 email | password | userid
-------+----------+--------

(0 rows)
```

The table is empty

Cassandra Lightweight Transactions

```
KVUser@cqlsh> SELECT * FROM killrvideo.user_credentials WHERE email = 'new_user@gmail.com';

 email | password | userid
-------+----------+--------

(0 rows)
KVUser@cqlsh> INSERT INTO killrvideo.user_credentials (email, password, userid)
   ...  VALUES('new_user@gmail.com', 'password_A', uuid())
   ...  IF NOT EXISTS;

 [applied]
-----------
      True
```

Here's the LWT

Cassandra Lightweight Transactions

```
KVUser@cqlsh> SELECT * FROM killrvideo.user_credentials WHERE email = 'new_user@gmail.com';

 email | password | userid
-------+----------+--------

(0 rows)
KVUser@cqlsh> INSERT INTO killrvideo.user_credentials (email, password, userid)
   ...  VALUES('new_user@gmail.com', 'password_A', uuid())
   ...  IF NOT EXISTS;

 [applied]
-----------
      True
```

The [applied] column shows the result

Cassandra Lightweight Transactions

```
KVUser@cqlsh> SELECT * FROM killrvideo.user_credentials WHERE email = 'new_user@gmail.com';

 email | password | userid
-------+----------+--------

(0 rows)
KVUser@cqlsh> INSERT INTO killrvideo.user_credentials (email, password, userid)
   ...  VALUES('new_user@gmail.com', 'password_A', uuid())
   ...  IF NOT EXISTS;

 [
```

The LWT Created the row

```
 User@cqlsh> SELECT * FROM killrvideo.user_credentials WHERE email = 'new_user@gmail.com';

 email                | password   | userid
----------------------+------------+--------------------------------------
 new_user@gmail.com | password_A | 6d8d869d-6ff4-4bd8-be66-d4334f3d2e4b

(1 rows)
```

Cassandra Lightweight Transactions

```
KVUser@cqlsh> SELECT * FROM killrvideo.user_credentials WHERE email = 'new_user@gmail.com';

 email                | password   | userid
----------------------+------------+--------------------------------------
 new_user@gmail.com | password_A | 6d8d869d-6ff4-4bd8-be66-d4334f3d2e4b

(1 rows)
```

Now the table has a row

Cassandra Lightweight Transactions

```
KVUser@cqlsh> SELECT * FROM killrvideo.user_credentials WHERE email = 'new_user@gmail.com';

 email                | password   | userid
----------------------+------------+--------------------------------------
 new_user@gmail.com | password_A | 6d8d869d-6ff4-4bd8-be66-d4334f3d2e4b

(1 rows)
```

Notice the password value

Cassandra Lightweight Transactions

```
KVUser@cqlsh> SELECT * FROM killrvideo.user_credentials WHERE email = 'new_user@gmail.com';

 email                | password   | userid
----------------------+------------+--------------------------------------
 new_user@gmail.com | password_A | 6d8d869d-6ff4-4bd8-be66-d4334f3d2e4b

(1 rows)
KVUser@cqlsh> INSERT INTO killrvideo.user_credentials (email, password, userid)
   ...  VALUES('new_user@gmail.com', 'password_XYZ', uuid())
   ...  IF NOT EXISTS;
```

LWT on an existing row

Cassandra Lightweight Transactions

```
KVUser@cqlsh> SELECT * FROM killrvideo.user_credentials WHERE email = 'new_user@gmail.com';

 email                | password   | userid
----------------------+------------+--------------------------------------
 new_user@gmail.com | password_A | 6d8d869d-6ff4-4bd8-be66-d4334f3d2e4b

(1 rows)
KVUser@cqlsh> INSERT INTO killrvideo.user_credentials (email, password, userid)
   ...  VALUES('new_user@gmail.com', 'password XYZ', uuid())
   ...  IF NOT EXISTS;
```

Notice password value

Cassandra Lightweight Transactions

```
KVUser@cqlsh> SELECT * FROM killrvideo.user_credentials WHERE email = 'new_user@gmail.com';

 email                | password   | userid
----------------------+------------+--------------------------------------
 new_user@gmail.com | password_A | 6d8d869d-6ff4-4bd8-be66-d4334f3d2e4b

(1 rows)
KVUser@cqlsh> INSERT INTO killrvideo.user_credentials (email, password, userid)
   ...  VALUES('new_user@gmail.com', 'password_XYZ', uuid())
   ...  IF NOT EXISTS;

 [applied] | email                | password   | userid
-----------+----------------------+------------+--------------------------------------
     False | new_user@gmail.com | password_A | 6d8d869d-6ff4-4bd8-be66-d4334f3d2e4b
```

Notice [applied] value

Cassandra Lightweight Transactions

```
KVUser@cqlsh> SELECT * FROM killrvideo.user_credentials WHERE email = 'new_user@gmail.com';

 email                | password   | userid
----------------------+------------+--------------------------------------
 new_user@gmail.com | password_A | 6d8d869d-6ff4-4bd8-be66-d4334f3d2e4b

(1 rows)
KVUser@cqlsh> INSERT INTO killrvideo.user_credentials (email, password, userid)
   ...  VALUES('new_user@gmail.com', 'password_XYZ', uuid())
   ...  IF NOT EXISTS;

 [applied] | email                | password   | userid
-----------+----------------------+------------+--------------------------------------
     False | new_user@gmail.com | password_A | 6d8d869d-6ff4-4bd8-be66-d4334f3d2e4b
```

The password field did not change

Cassandra Lightweight Transactions

```
KVUser@cqlsh> SELECT * FROM killrvideo.user_credentials WHERE email = 'new_user@gmail.com';

 email                | password   | userid
----------------------+------------+--------------------------------------
 new_user@gmail.com | password_A | 6d8d869d-6ff4-4bd8-be66-d4334f3d2e4b

(1 rows)
KVUser@cqlsh> INSERT INTO killrvideo.user_credentials (email, password, userid)
   ...  VALUES('new_user@gmail.com', 'password_XYZ', uuid())
   ...  IF NOT EXISTS;

 [applied] | email                | password   | userid
-----------+----------------------+------------+--------------------------------------
     False | new_user@gmail.com | password_A | 6d8d869d-6ff4-4bd8-be66-d4334f3d2e4b

KVUser@cqlsh> SELECT * FROM killrvideo.user_credentials WHERE email = 'new_user@gmail.com';

 email                | password   | userid
----------------------+------------+--------------------------------------
 new_user@gmail.com | password_A | 6d8d869d-6ff4-4bd8-be66-d4334f3d2e4b

(1 rows)
```

The row did not change

Cassandra Lightweight Transactions

```
KVUser@cqlsh> SELECT * FROM killrvideo.user_credentials WHERE email = 'new_user@gmail.com';

 email                | password   | userid
----------------------+------------+--------------------------------------
 new_user@gmail.com | password_A | 6d8d869d-6ff4-4bd8-be66-d4334f3d2e4b
```

Again, the table has a row

Operators: =, <, <=, >, >=, != and IN

Cassandra Lightweight Transactions

```
KVUser@cqlsh> SELECT * FROM killrvideo.user_credentials WHERE email = 'new_user@gmail.com';

 email                | password   | userid
----------------------+------------+--------------------------------------
 new_user@gmail.com | password_A | 6d8d869d-6ff4-4bd8-be66-d4334f3d2e4b

(1 rows)
KVUser@cqlsh> UPDATE killrvideo.user_credentials SET password = 'password_XYZ'
   ... WHERE email = 'new_user@gmail.com'
   ... IF password = 'password_A';
```

Here's an UPDATE LWT

Operators: =, <, <=, >, >=, != and IN

Cassandra Lightweight Transactions

```
KVUser@cqlsh> SELECT * FROM killrvideo.user_credentials WHERE email = 'new_user@gmail.com';

 email                | password   | userid
----------------------+------------+--------------------------------------
 new_user@gmail.com | password_A | 6d8d869d-6ff4-4bd8-be66-d4334f3d2e4b

(1 rows)
KVUser@cqlsh> UPDATE killrvideo.user_credentials SET password = 'password_XYZ'
   ... WHERE email = 'new_user@gmail.com'
   ... IF password = 'password_A';

 [applied]
-----------
      True
```

Operators: =, <, <=, >, >=, != and IN

Batch

Main reason to use BATCH

When using denormalization mutating a record requires changes in several tables

Warning: CQL Batch is NOT the same thing as SQL Batch

Any could fail...

```
INSERT INTO one_table …
INSERT INTO another_table...
INSERT INTO yet_another_table...
```

Is Batch Atomic?

```
BEGIN BATCH

INSERT INTO one_table …
INSERT INTO another_table...
INSERT INTO yet_another_table...

APPLY BATCH;
```

Is Batch Atomic?

NO!

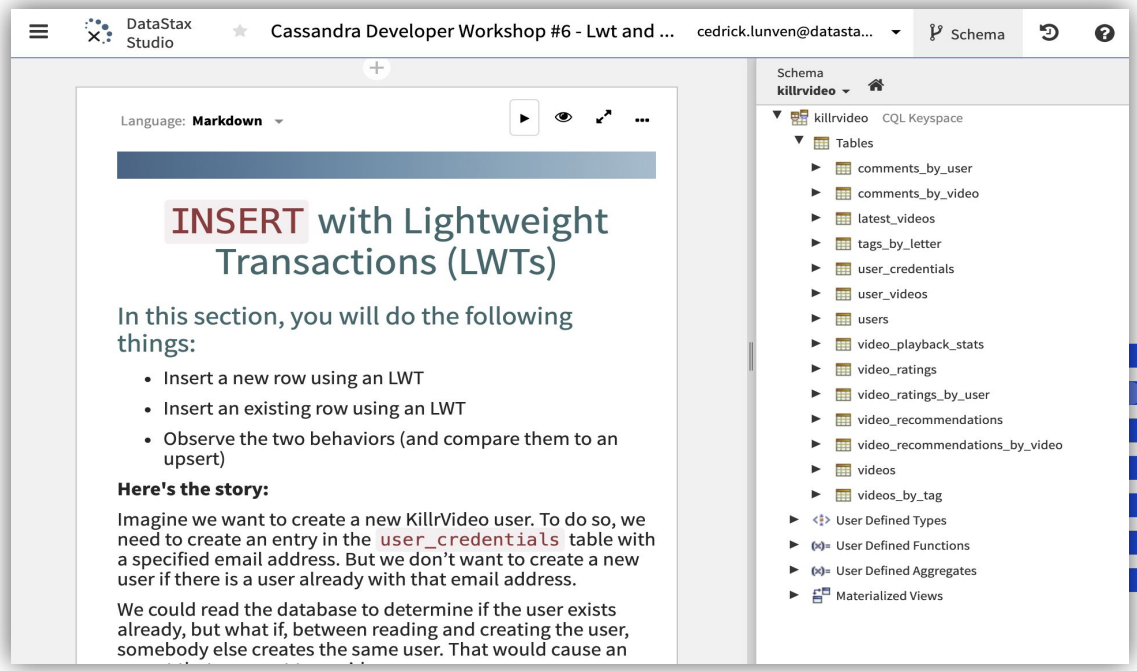
```
BEGIN BATCH

INSERT INTO one_table …
INSERT INTO another_table...
INSERT INTO yet_another_table...

APPLY BATCH;
```

So Why Use Batch?

- Batch uses a much stronger retry method to make sure all operations complete.
- If an operation fails the client will be notified allowing it to handle the error condition.

Exercise 6

Lwt and Batches



DataStax Drivers

- CQL Support
- Sync / Async API
- Address Translation
- Load Balancing Policies
- Retry Policies
- Reconnection Policies

- - Connection Pooling
- - Auto Node Discovery
- - SSL
- - Compression
- - Query Builder
- - Object Mapper

- Synchronous, Asynchronous, Reactive




- ODBC
- JDBC

Apache Maven™

- OSS Driver

```
<dependency>
 <groupId>com.datastax.oss</groupId>
 <artifactId>java-driver-core</artifactId>
</dependency>
```

Builder

```
// Delegate all configuration to file or default
CqlSession cqlSession = CqlSession.builder().build();

// Explicit Settings
CqlSession cqlSession = CqlSession.builder()
 .withCloudSecureConnectBundle(Paths.get("/tmp/apollo.zip"))
 .withKeyspace("killrvideo")
 .withAuthCredentials("myUser", "myPassword")
 .build();
```

How to to execute queries ?

- First job of **CqlSession** is to execute queries using, well, execute method.

```
cqlSession.execute("SELECT * FROM killrvideo.users");
```

SimpleStatement

```
Statement statement = ...

// (1) Explicit SimpleStatement Definition
SimpleStatement.newInstance("select * from t1 where c1 = 5");

// (2) Externalize Parameters (no name)
SimpleStatement.builder("select * from t1 where c1 = ?")
               .addPositionalValue(5);

// (3) Externalize Parameters (name)
SimpleStatement.builder("select * from t1 where c1 = :myVal")
               .addNamedValue("myVal”, 5);

cqlSession.execute(statement);
```

Prepared and Bound Statements

- Compiled once on each node automatically as needed
- Prepare each statement only once per application
- Use one of the many bind variations to create a BoundStatement

```
PreparedStatement ps = cqlSession.prepare("SELECT * from t1 where c1 = ?");

BoundStatement bound = ps.bind(5);

cqlSession.execute(bound);
```

Query Builder

- Fluent API for building CQL string queries programmatically
- Contains methods to build SELECT, UPDATE, INSERT and DELETE statements
- Generates a Statement as per the earlier techniques

OSS Driver (current version 4.2.0)

```
<dependency>
 <groupId>com.datastax.oss</groupId>
 <artifactId>
  java-driver-query-builder
 </artifactId>
</dependency>
```

Query Builder

```
import static com.datastax.oss.driver.api.querybuilder.QueryBuilder.bindMarker;
import static com.datastax.oss.driver.api.querybuilder.QueryBuilder.deleteFrom;
import static com.datastax.oss.driver.api.querybuilder.QueryBuilder.selectFrom;
import static com.datastax.oss.driver.api.querybuilder.relation.Relation.column;

// Simple SELECT using QueryBuilder
Statement stmtSelect = selectFrom(”killrvideo”, ”videos_by_users”)
  .column(”userid”).column(”commentid”)
  .function(”toTimestamp”, Selector.column(”commentid”)).as("comment_timestamp")
  .where(column(”userid”).isEqualTo(bindMarker(”userid”)))
  .build()

// Simple DELETE using QueryBuilder
Statement stmtDelete = deleteFrom(”killrvideo”, ”videos_by_users”)
   .where(column(”userid”).isEqualTo(bindMarker(”userid”)))
   .build()
```

Query Builder

- Can also use **QueryBuilder** to create **PreparedStatements** and later execute at runtime
- Note use of **bindMarker() t**o designate parameters that will be provided later

```
// Prepared QueryBuilder statiements as any statement
PreparedStatement psStmt = cqlSession.prepare(
  deleteFrom(”killrvideo”, ”videos_by_users”)
   .where(column(”userid”).isEqualTo(bindMarker(”userid”)))
   .build()));

// Binding
BoundStatement bsStmt = psStmt.bind(”e7a8ac9f-c12d-415c-a526-4137815df573”);

// Execute
cqlSession.execute(bsStmt);
```

ResultSet

- ResultSet is the object returned for executing query. It contains **ROWS** (data) and **EXECUTION INFO**.
- ResultSet is **iterable** and as such you can navigate from row to row.
- Results are **always paged** for you (avoiding memory and response time issues)

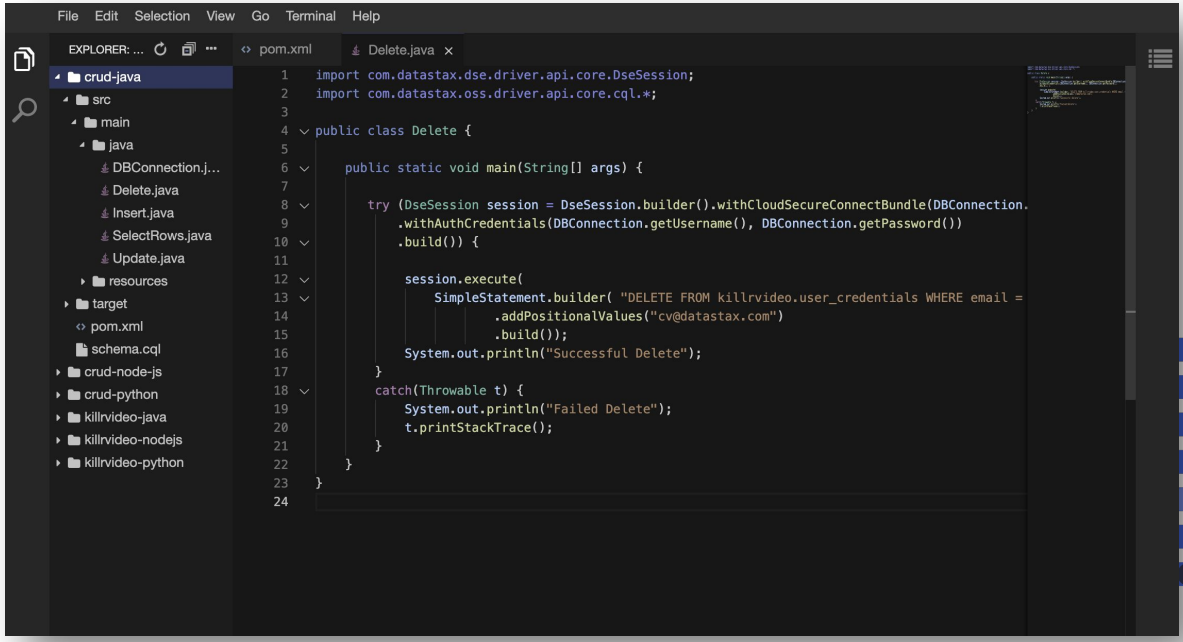
```
ResultSet rs = cqlSession.execute(myStatement);

// Plumbery
ExecutionInfo info = rs.getExecutionInfo();
int executionTime = info.getQueryTrace().getDurationMicros();

// Data: NOT ALL DATA RETRIEVED IMMEDIATELY (only when needed .next())
Iterator<Row> iterRow = rs.iterator();
int itemsFirstCall = rs.getAvailableWithoutFetching();
```

Parsing ResultSet

```
// We know there is a single row (eg: count)
Row singleRow = resultSet.one();

// We know there are not so many results we can get all (fetch all pages)
List<Row> allRows = resultSet.all();

// Browse iterable
for(Row myRow : resultSet.iterator()) {
   // .. Parsing rows
}

// Use Lambda
rs.forEach(row -> { row.getColumnDefinitions(); });

// Use for LWT
boolean isQueryExecuted = rs.wasApplied();
```

Exercise 7

Coding



Developer Workshop

1. Bootstrapping
2. Apache Cassandra™ Why, What & When
3. Data Modeling with Apache Cassandra™
4. **What's NEXT ?**

Developer Resources

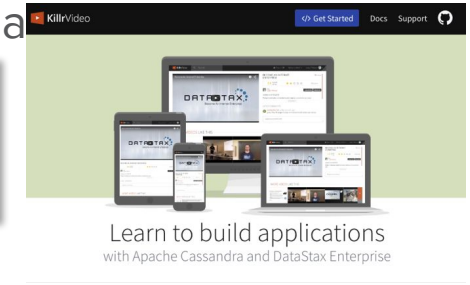
LEARN	Join academy.datastax.com Free online courses - Cassandra certifications
ASK/SHARE	Join community.datastax.com Ask/answer community user questions - share your expertise
CONNECT	Follow us @DataStaxDevs We are on Twitter - Twitch!
REVIEW	Slides and code for this course are available at https://github.com/DataStax-Academy/online-Cassandra-workshop

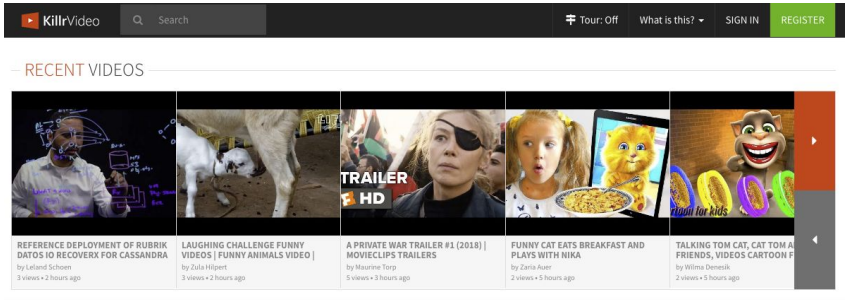
Training Courses at DataStax Academy

- Free self-paced DSE 6 courses
  - DS201: DataStax Enterprise 6 Foundations of Ap
  - DS210: DataStax Enterprise 6 Operations with A
  - DS220: DataStax Enterprise 6 Practical Applicati
  - DS330: DataStax Enterprise 6 Graph
  - DS332: DataStax Enterprise 6 Graph Analytics (N

KillrVideo Reference Application

- Reference application for learning how to use Apache Cassa Enterprise
  - DataStax Drivers
  - Docker images
- Source code freely available
  - https://github.com/killrvideo
- Live version
  - http://killrvideo.com
- Download, test, modify, contribute!





DATASTAX

Thank You