

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





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 ?



@DataStaxDevs #DataStaxDeveloperDay

<https://community.datastax.com>



Developer Workshop

1

Bootstrapping

2

Apache Cassandra™ Why, What & When

3

Data Modeling with Apache Cassandra™

4

What's NEXT ?



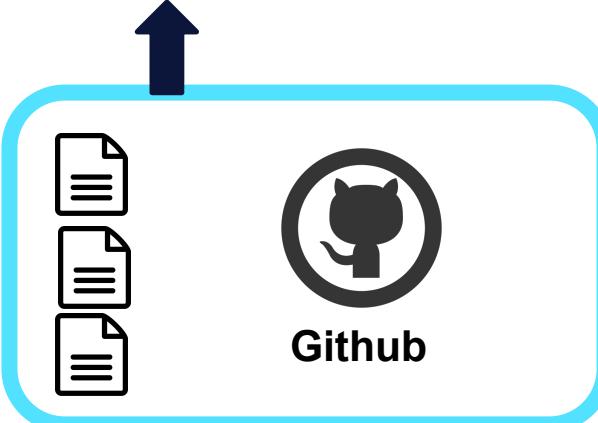
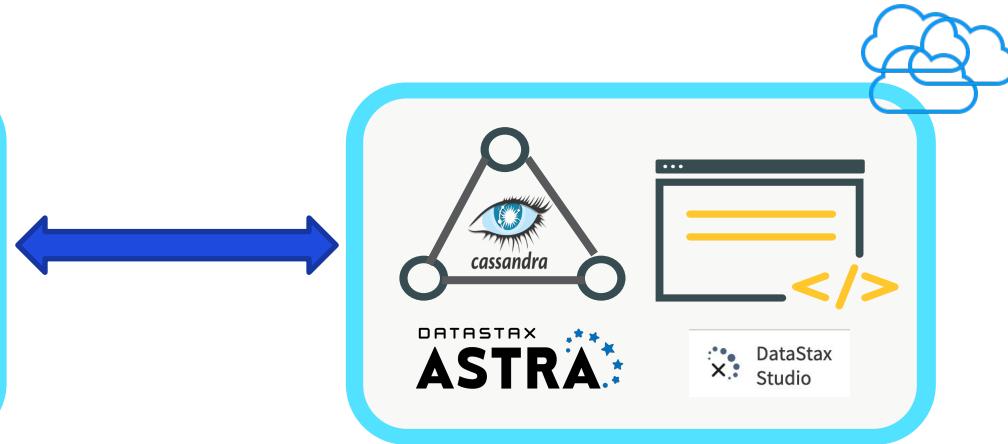
@DataStaxDevs #DataStaxDeveloperDay

<https://community.datastax.com>



Overview

YOUR LAPTOP



@DataStaxDevs #DataStaxDeveloperDay

<https://community.datastax.com>



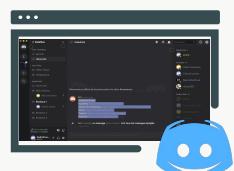
Get your hands on Discord



YOUR LAPTOP



Browser



Webconf

- 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



@DataStaxDevs #DataStaxDeveloperDay

<https://community.datastax.com>



Exercise 0



Bootstrap your Environment

YOUR LAPTOP



Browser



Discord

1. Clone or download repository material

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



Github

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

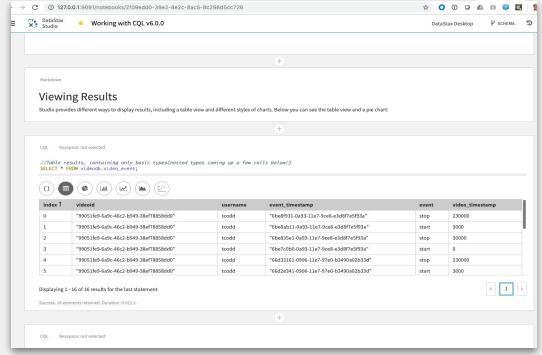


Simplify Application Development

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



The screenshot shows the DataStax Studio interface. At the top, it says "Working with CQL v6.0.0". Below that is a "Viewing Results" section with a table header:

index	event_id	username	event_timestamp	event	value	Timestamp
0	"9005-1d5-40f-4c2-1b9-34f-78750607"	tosidd	"1e4bf551-0a93-11e7-9e0-a5287ff7052a"	stop	33000	
1	"9005-1d5-40f-4c2-1b9-34f-78750607"	tosidd	"1e4bfab1-1a03-11e7-9e0-a5287ff7052a"	start	3000	
2	"9005-1d5-40f-4c2-1b9-34f-78750607"	tosidd	"1e4bfac1-0a93-11e7-9e0-a5287ff7052a"	stop	30000	
3	"9005-1d5-40f-4c2-1b9-34f-78750607"	tosidd	"1e4bfad1-0a93-11e7-9e0-a5287ff7052a"	start	3	
4	"9005-1d5-40f-4c2-1b9-34f-78750607"	tosidd	"1e4bfad1-0a93-11e7-9e0-a5287ff7052a"	stop	330000	
5	"9005-1d5-40f-4c2-1b9-34f-78750607"	tosidd	"1e4bfad1-0a93-11e7-9e0-a5287ff7052a"	start	3000	

At the bottom of the interface, there is a CQL input field.

Great Drivers



Exercise 1



Create your Astra Instance

A screenshot of the DataStax Astra interface. At the top, it says "Your Organization". Below that, a green banner states: "Database is initializing. You'll receive an email when the database is ready. While you're waiting, learn how to get started with your database. Database creation started at Jan 22nd 2020 11:49am UTC. This can take as long as 30 minutes but is usually much quicker." The main area shows a table for the "Apollo Databases > screenshot" database. The table has four columns: "Screenshot" (with details like Keyspace Name: okidoki, Organization: Your Organization, Owner: Cedrick Lunven, Created: January 22, 2020), "Size and Location" (Capacity Unit: 1, 500 GB Total Storage, Storage Used: TBD, Locations: us-east-1 (1 capacity unit), Compute Size: Startup, Replication Factor: 3), "Cost" (Spent this month: TBD, Estimated Cost: Per Hour or Per Month), and "Connection Details" (Connection details are only available for active databases).

Screenshot		Size and Location	
Keyspace Name: okidoki Organization: Your Organization	1 Capacity Unit 500 GB Total Storage Storage Used: TBD	Locations: us-east-1 (1 capacity unit) Compute Size: Startup Replication Factor: 3	
Owner: Cedrick Lunven Created: January 22, 2020			

Cost		Connection Details	
Spent this month: TBD	Estimated Cost: <input checked="" type="radio"/> Per Hour <input type="radio"/> Per Month		Connection details are only available for active databases that you own or have connection permissions for.
when running	TBD/hour		
when parked	TBD/hour		

Developer Workshop

1

Bootstrapping

2

Apache Cassandra™ Why, What & When

3

Data Modeling with Apache Cassandra™

4

What's NEXT ?

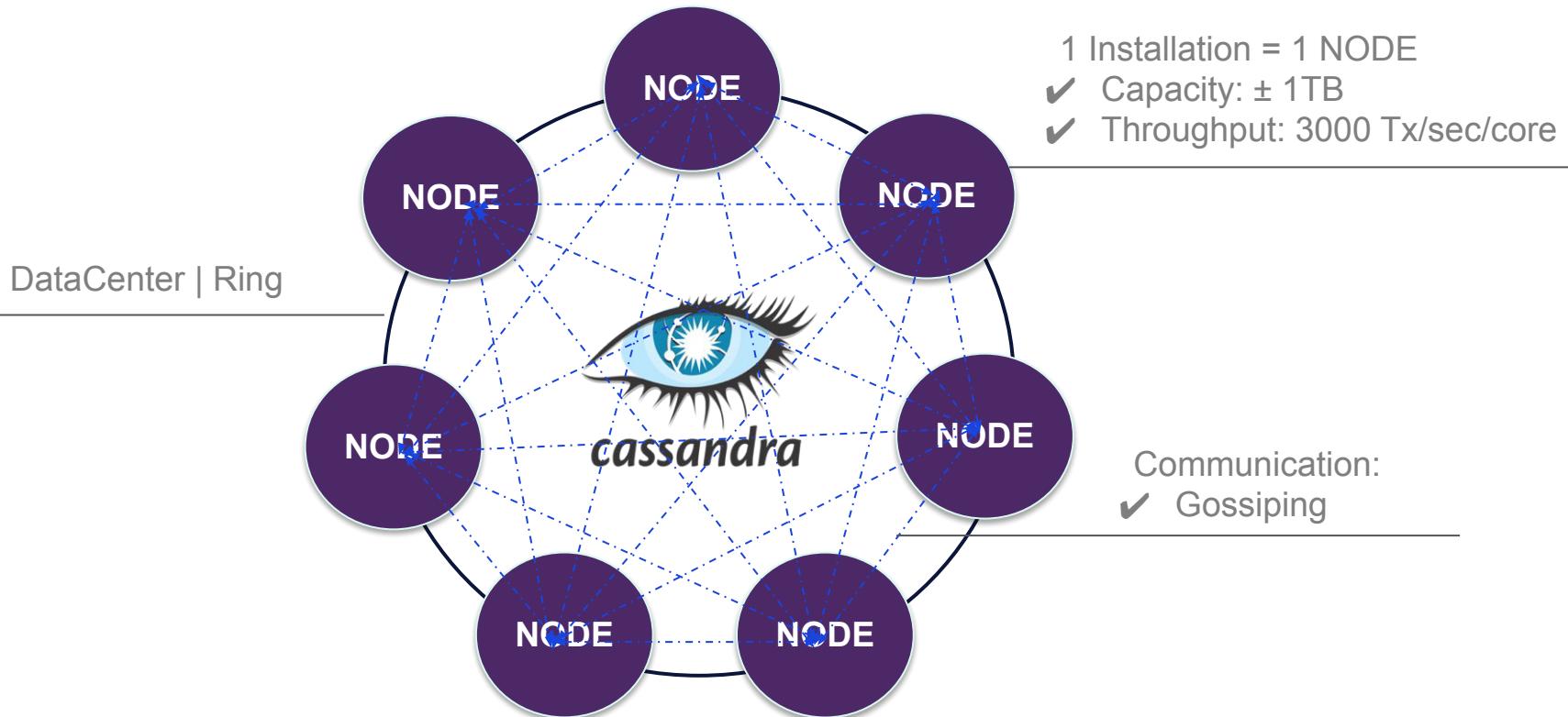


@DataStaxDevs #DataStaxDeveloperDay

<https://community.datastax.com>



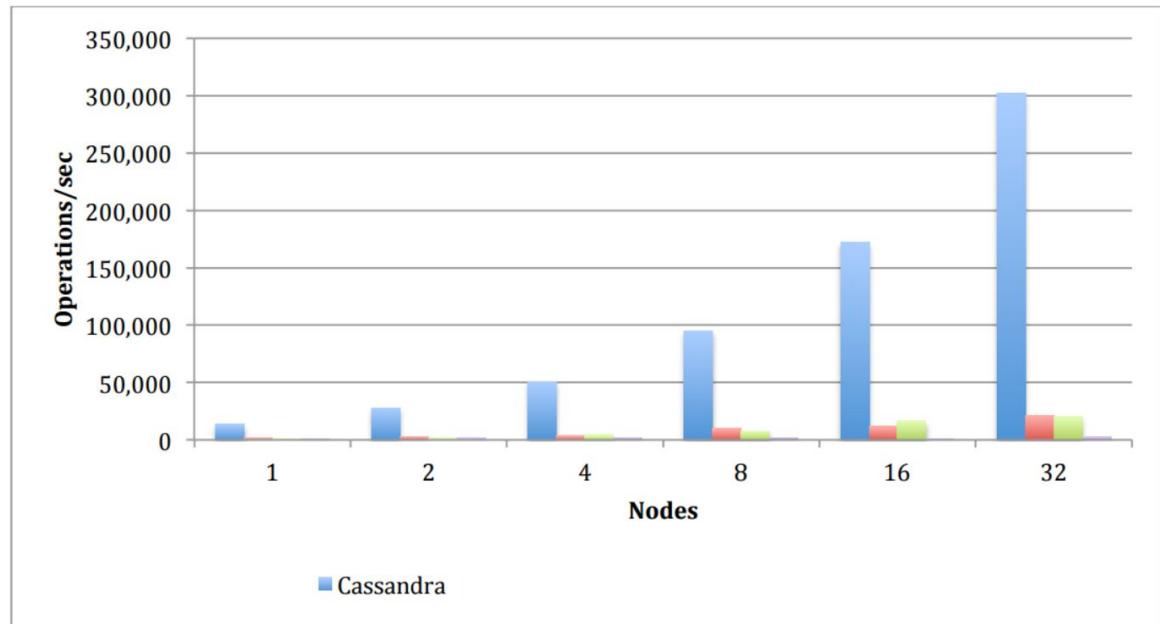
Apache Cassandra™ = NoSQL Distributed Database



Scales Linearly

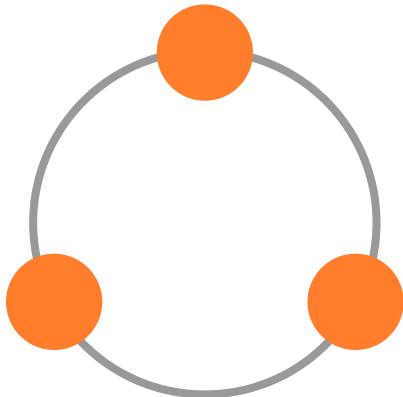
- Need more capacity?
- Need more throughput?
- Add nodes!

Balanced Read/Write Mix

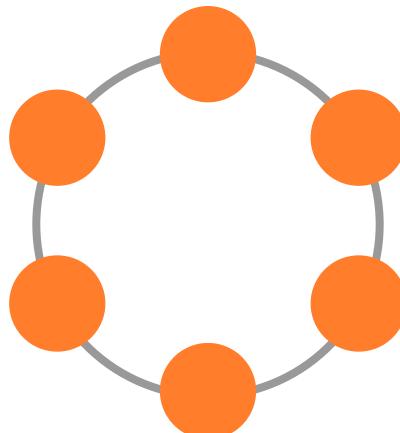


Horizontal vs. Vertical Scaling

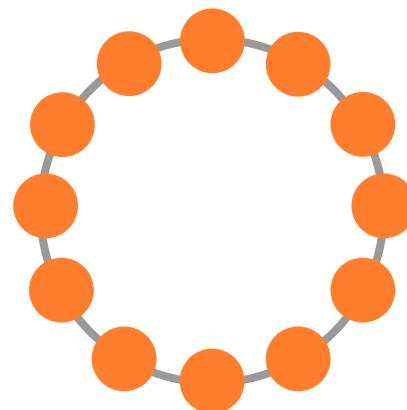
- Vertical scaling requires one large expensive machine
- Horizontal scaling requires multiple less-expensive commodity hardware



100,000 transactions/second



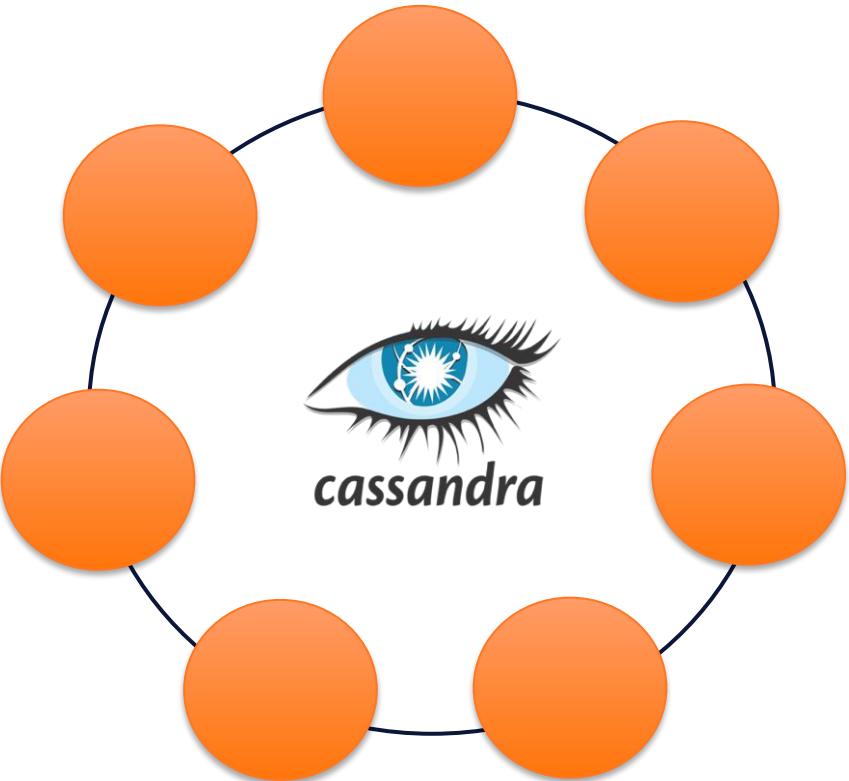
200,000 transactions/second



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

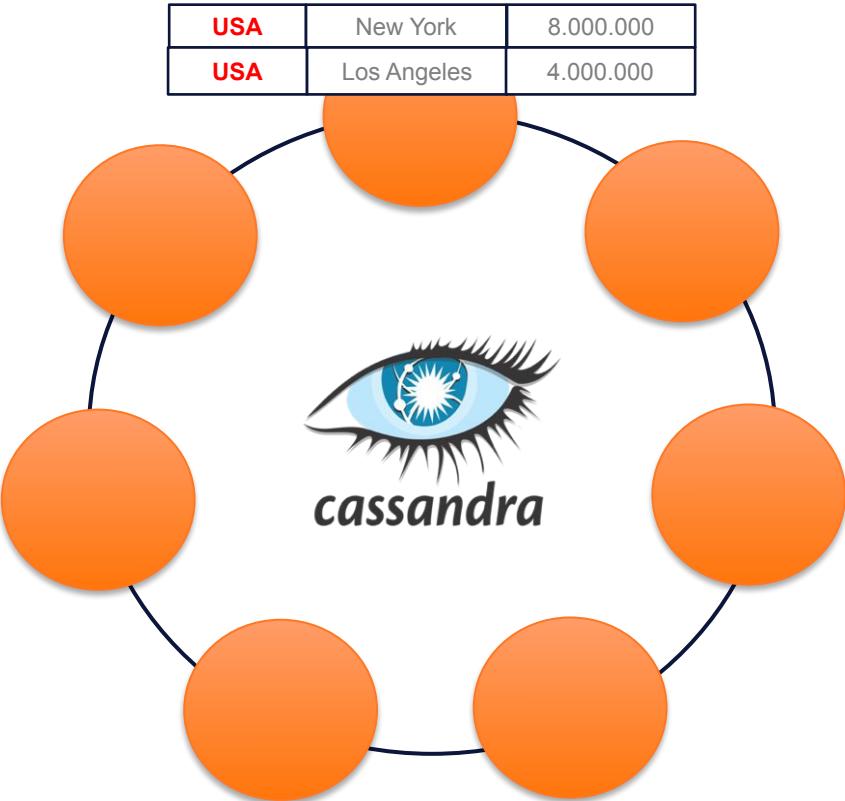


@DataStaxDevs #DataStaxDeveloperDay

<https://community.datastax.com>



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

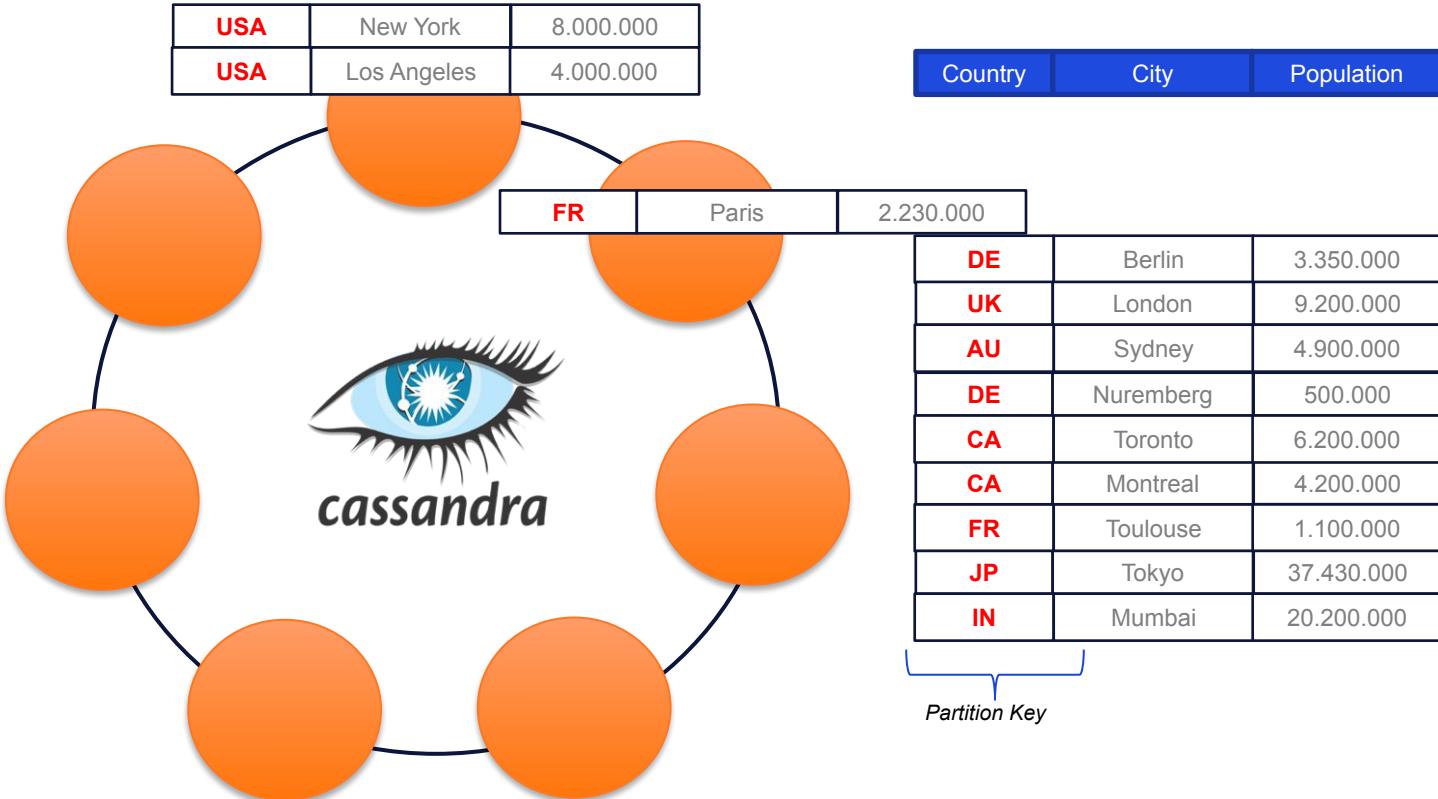


@DataStaxDevs #DataStaxDeveloperDay

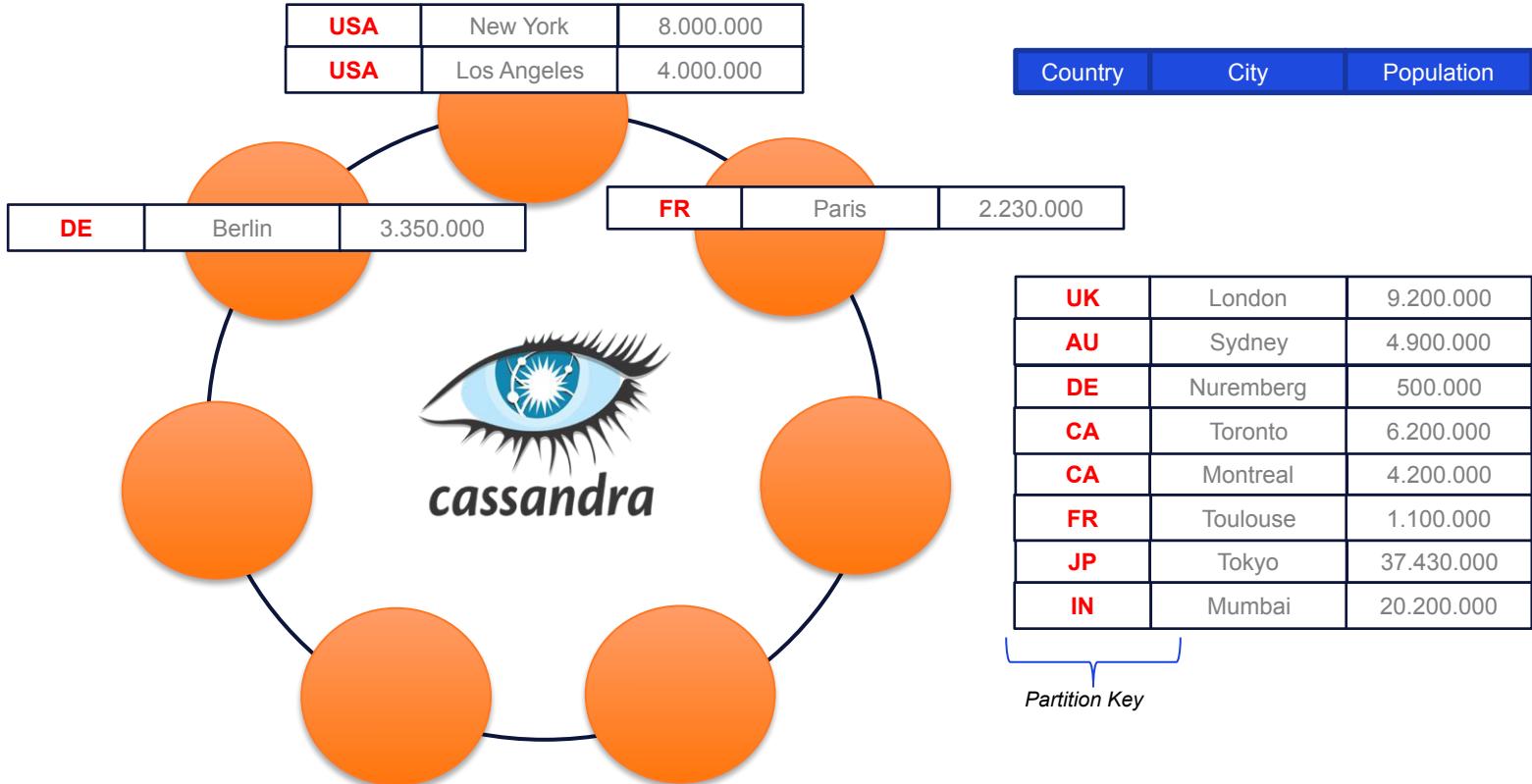
<https://community.datastax.com>



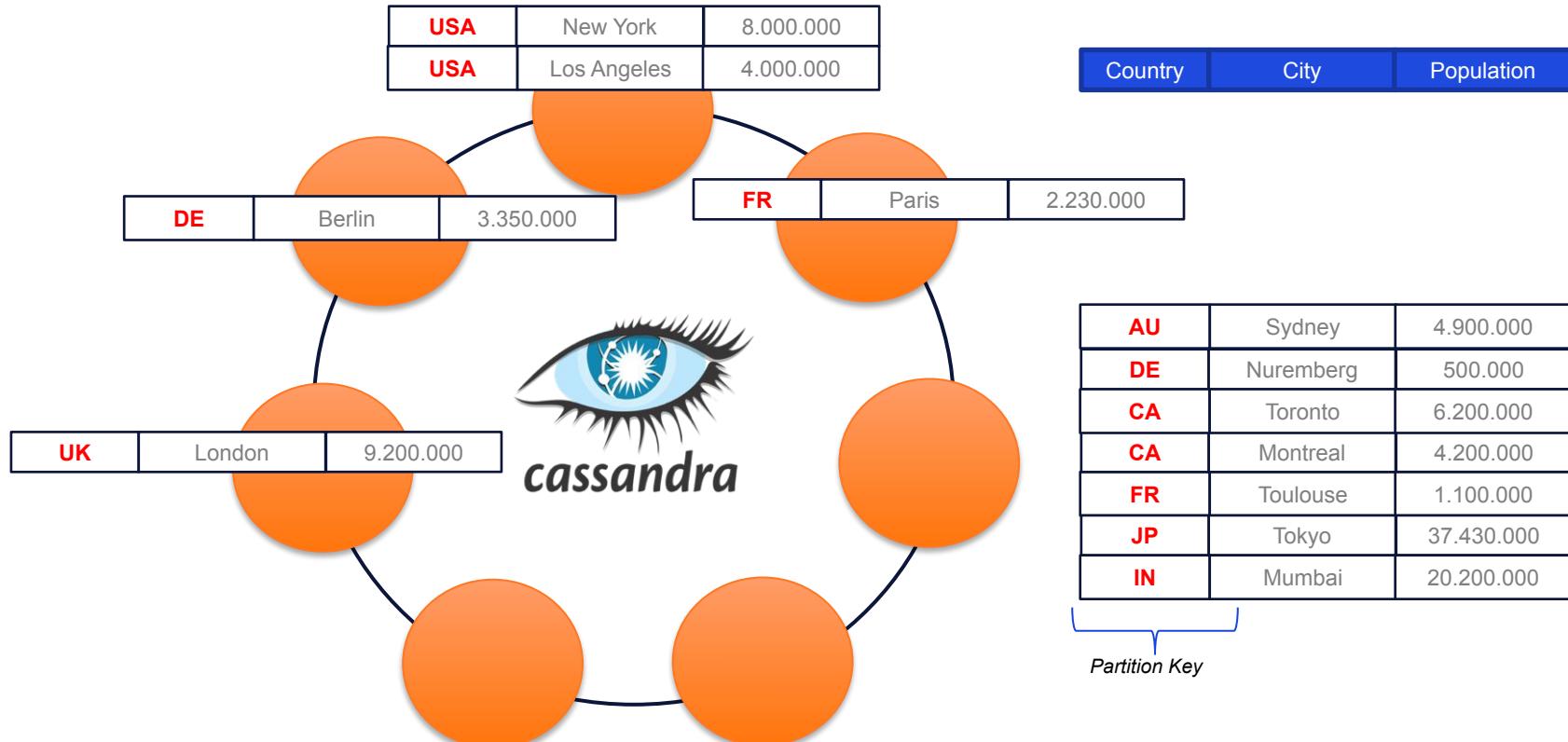
Data is Distributed



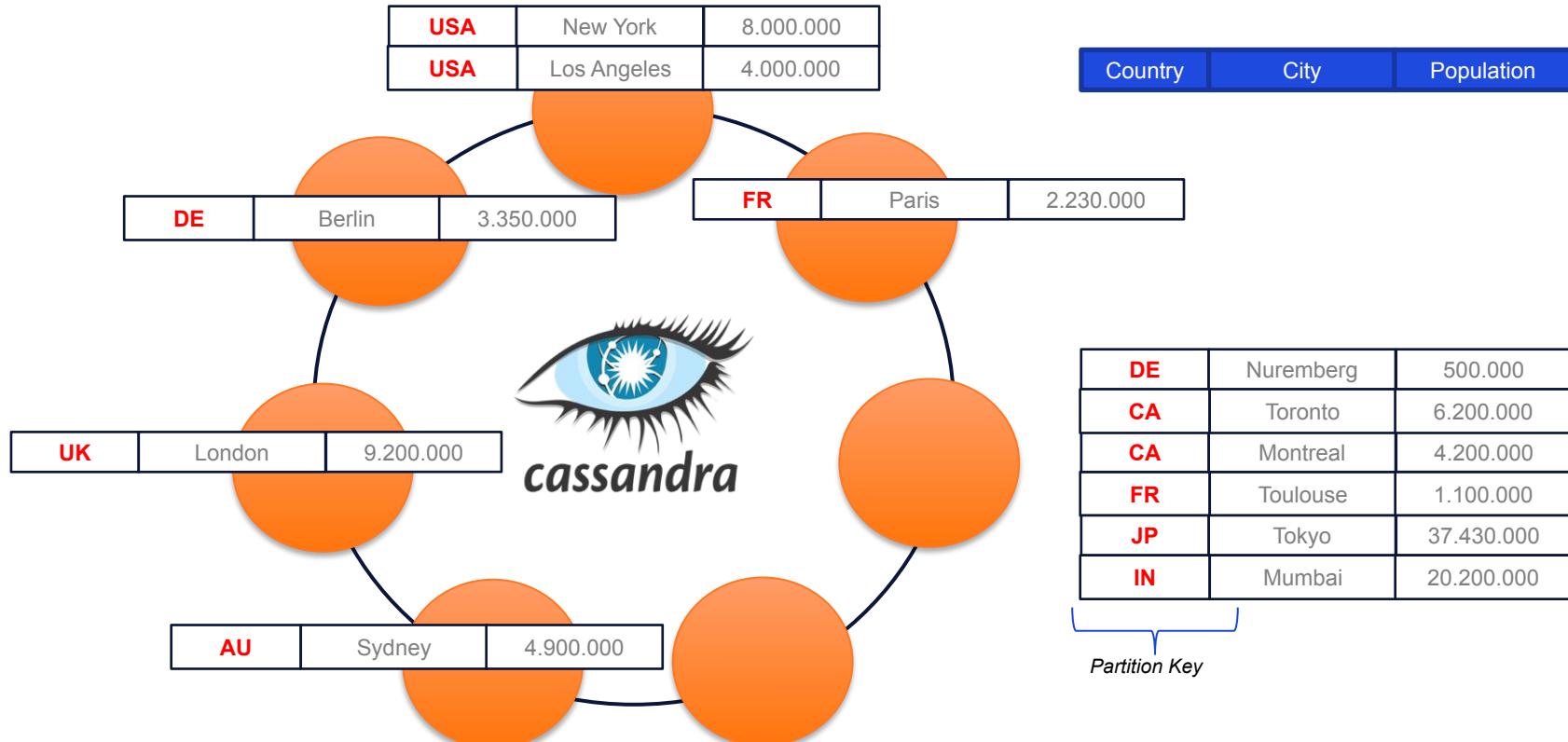
Data is Distributed



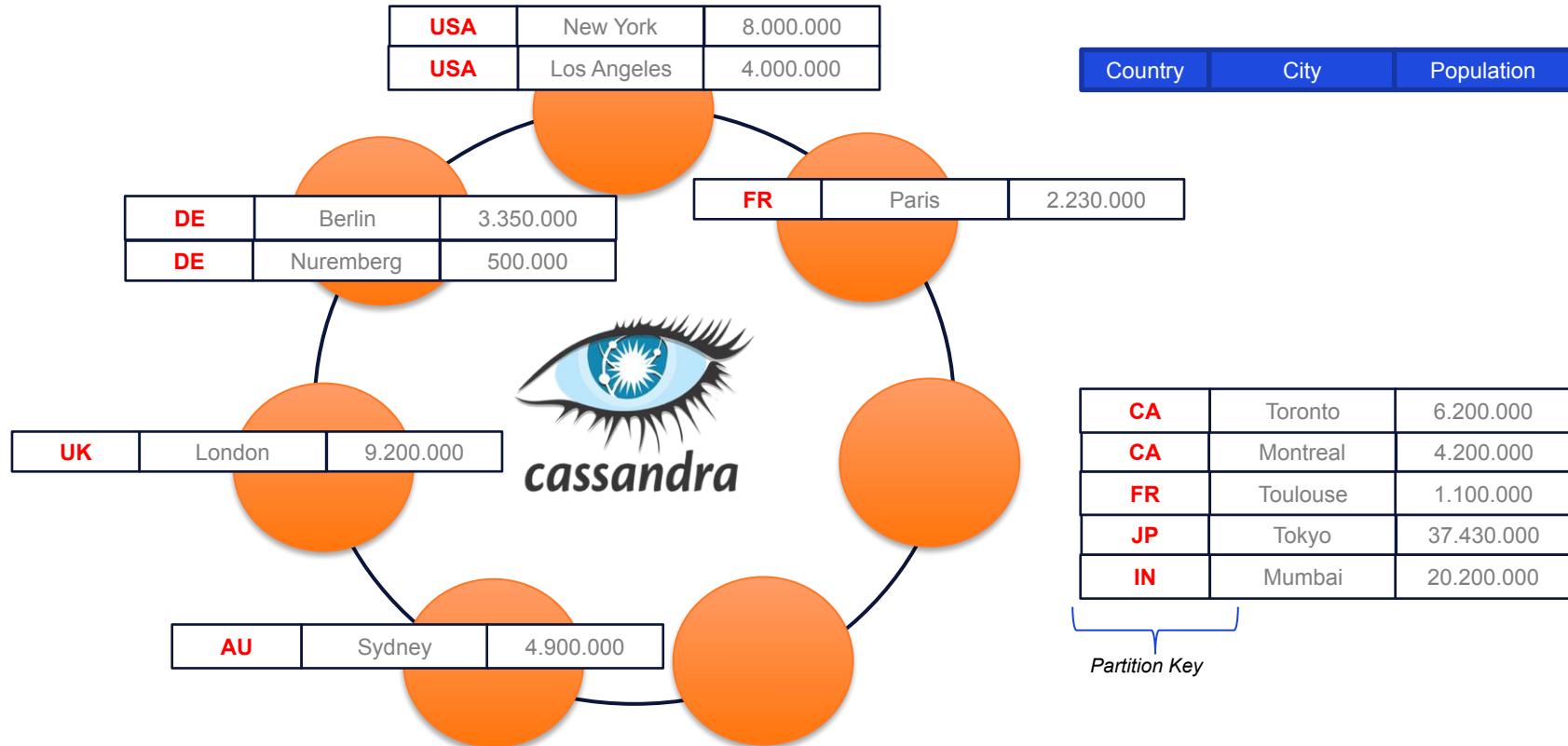
Data is Distributed



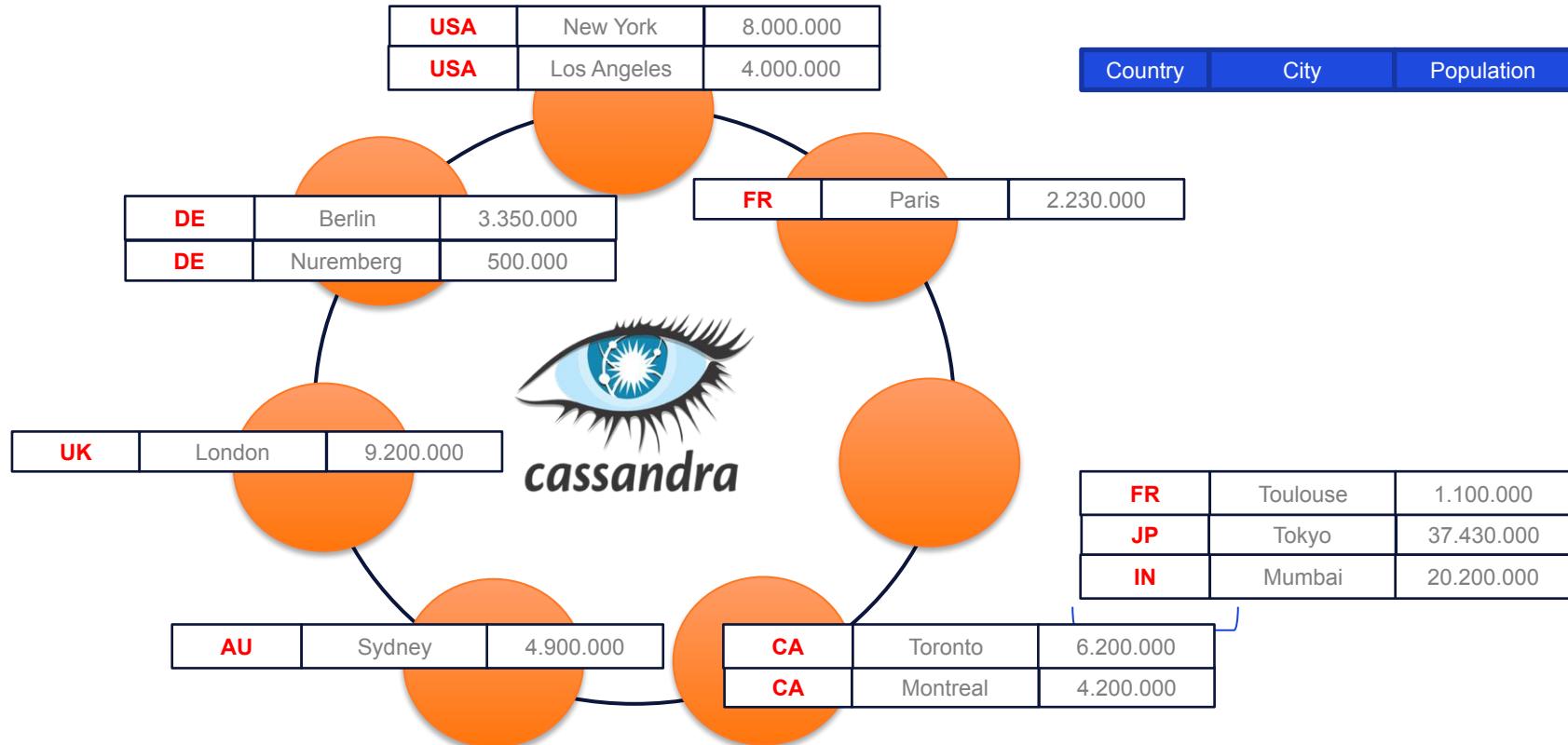
Data is Distributed



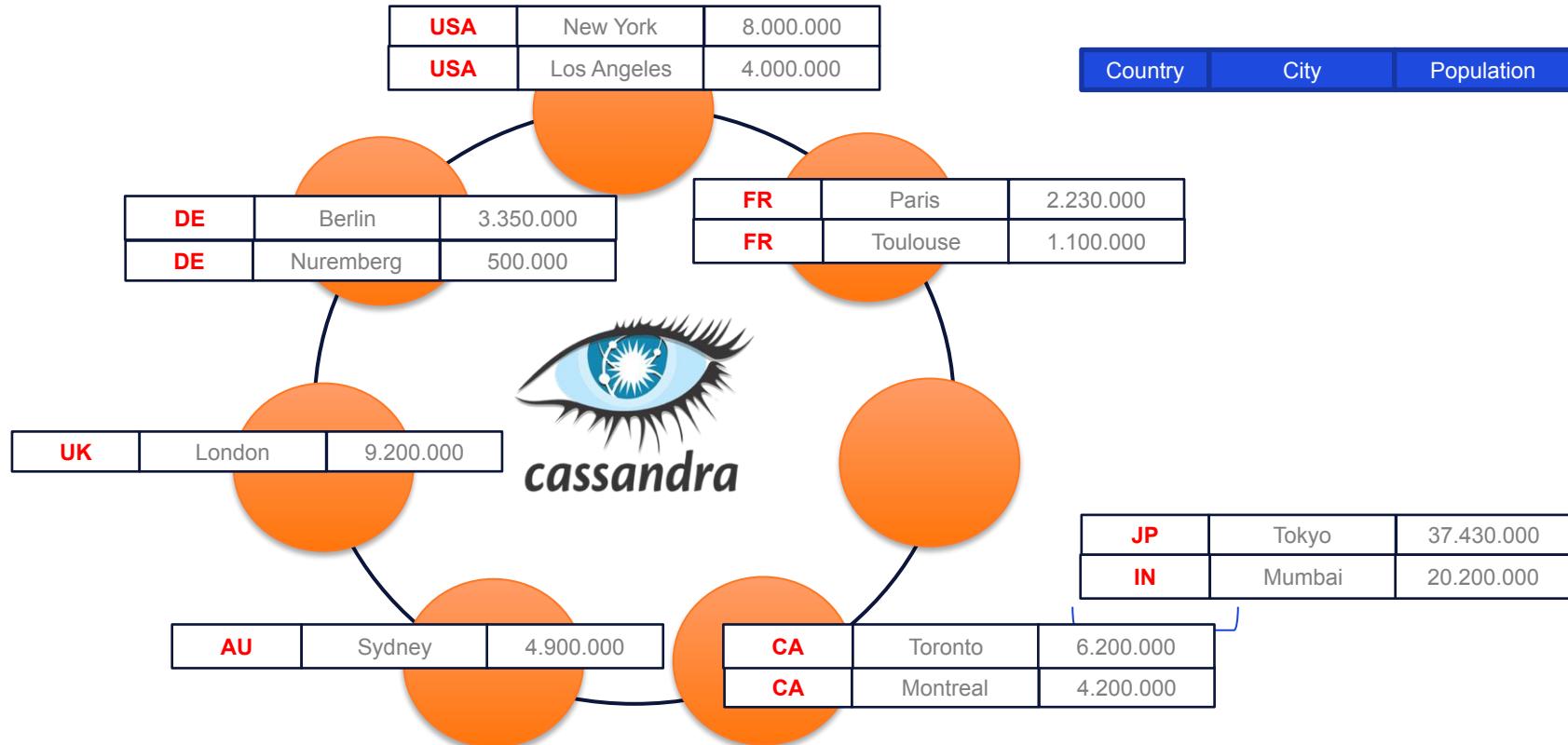
Data is Distributed



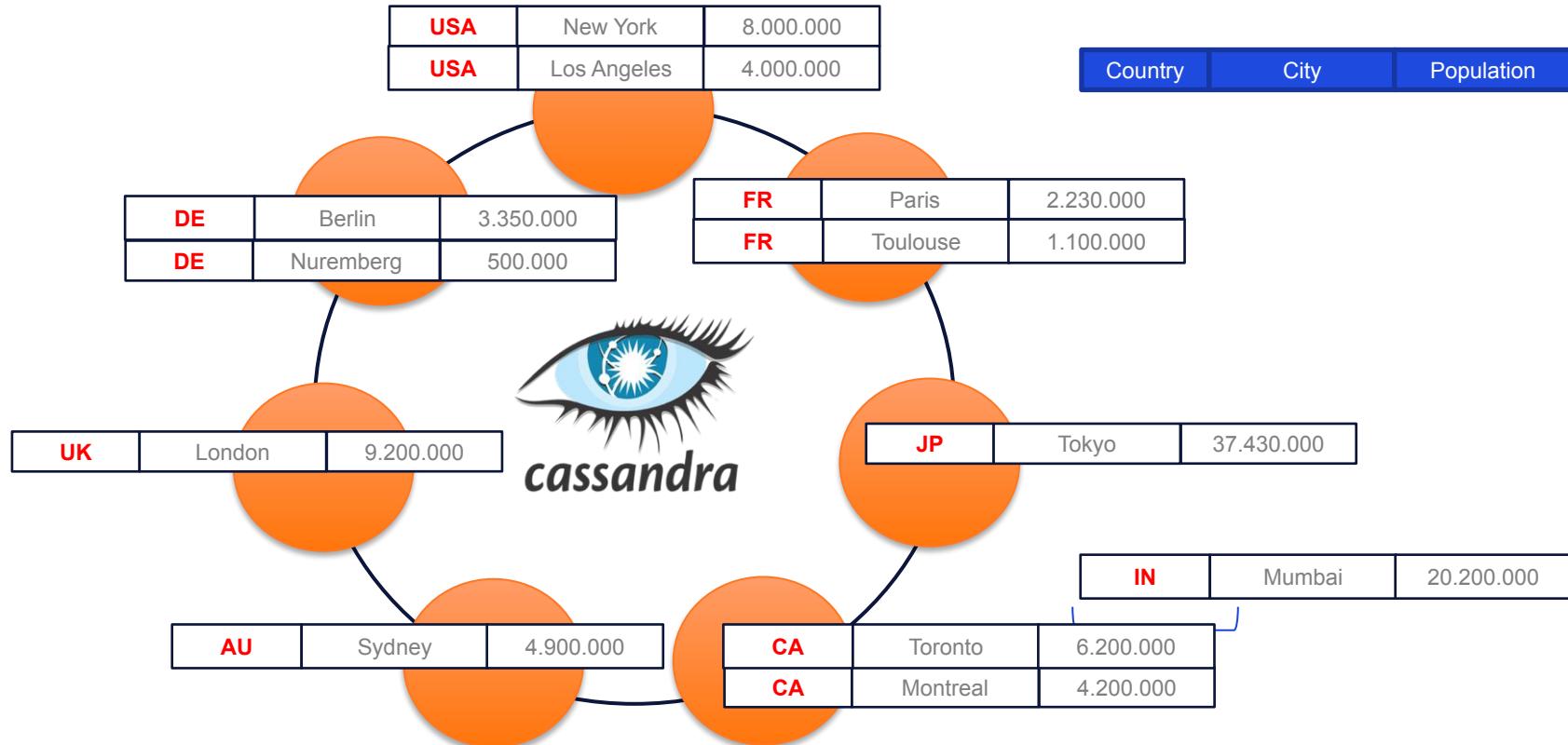
Data is Distributed



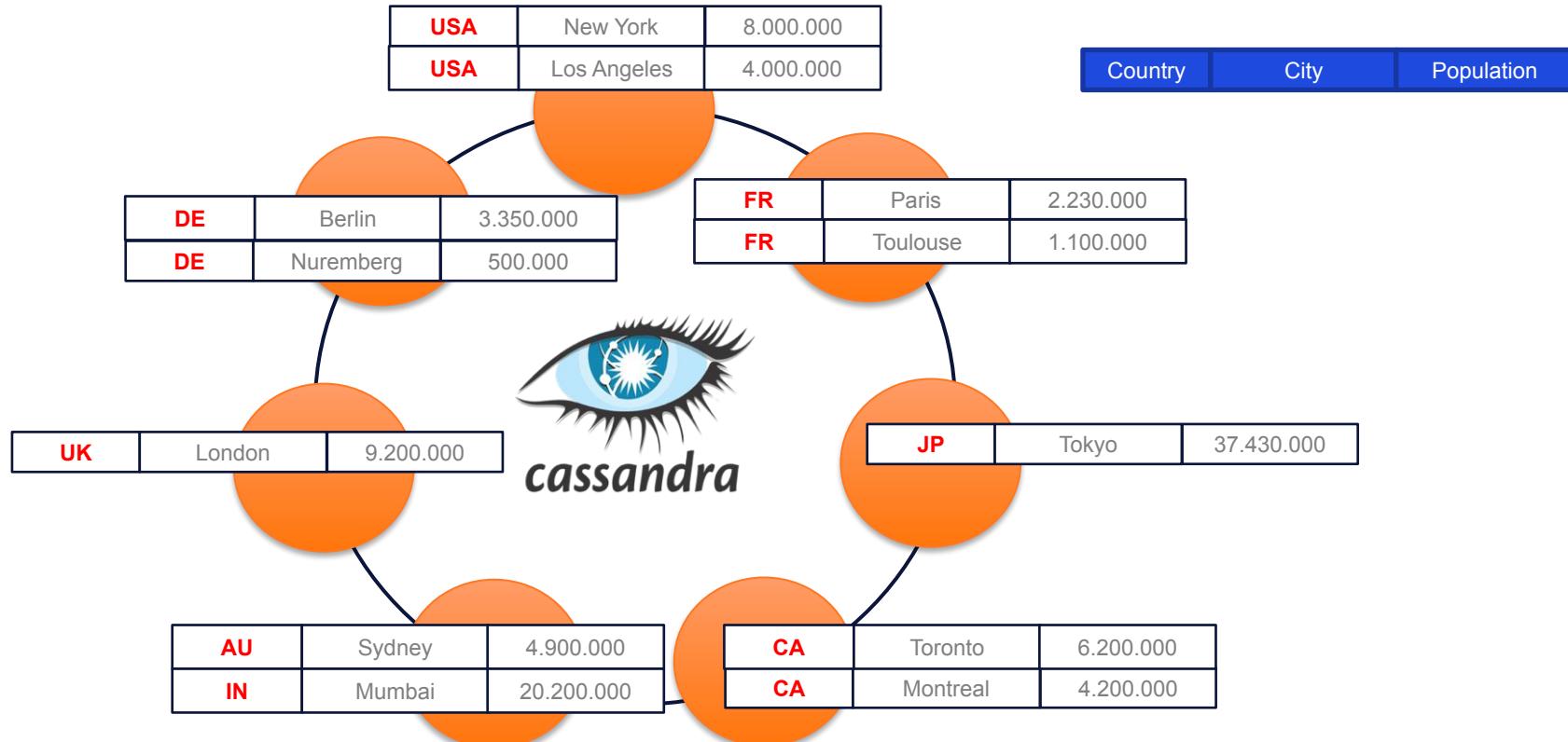
Data is Distributed



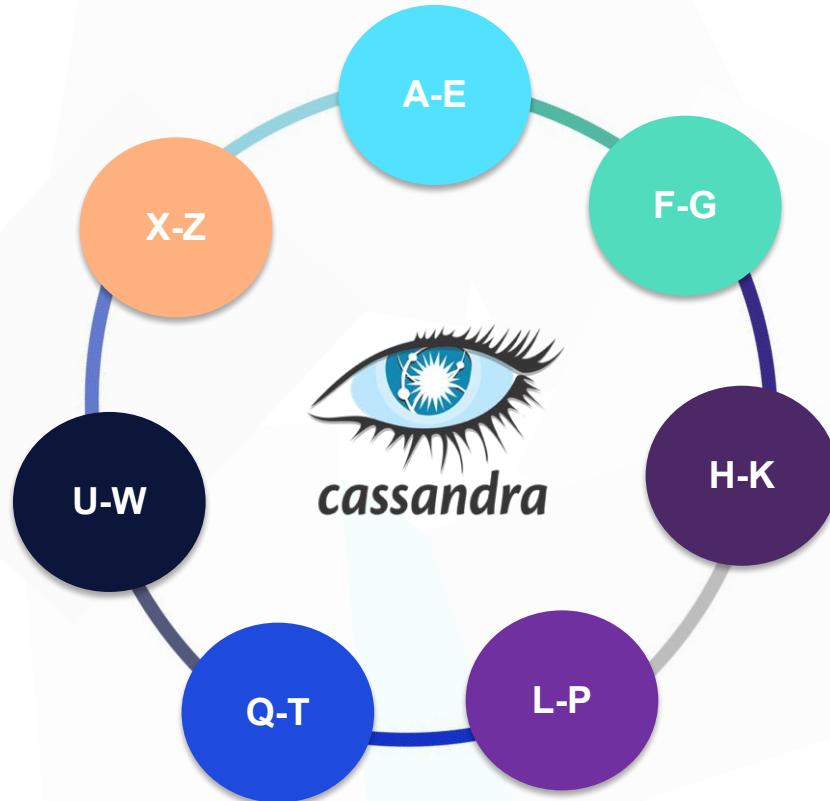
Data is Distributed



Data is Distributed



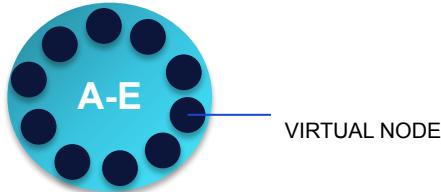
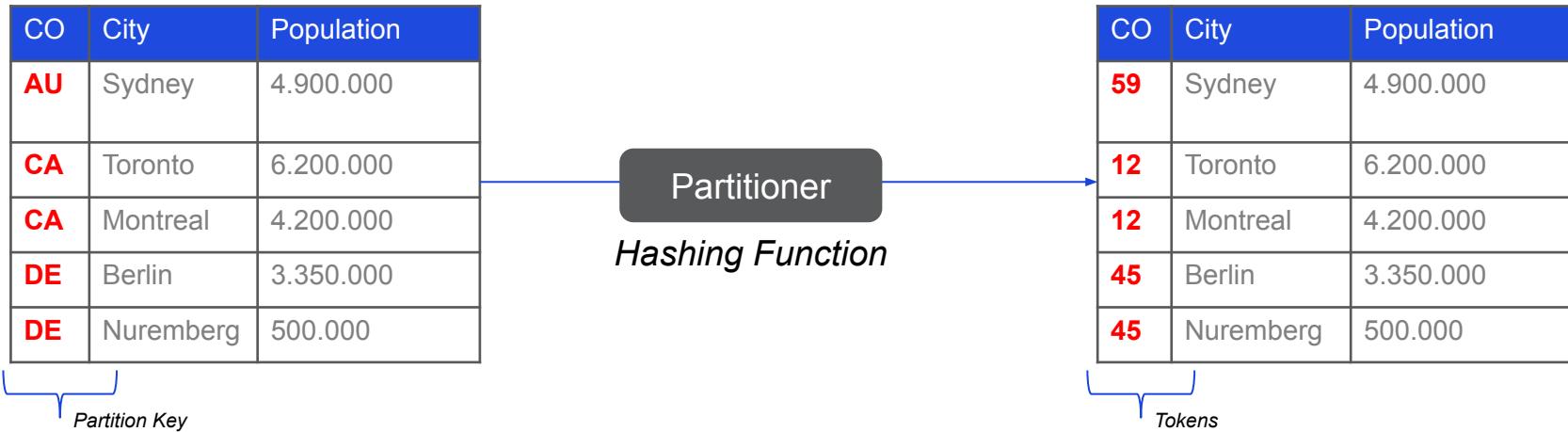
Data is Distributed



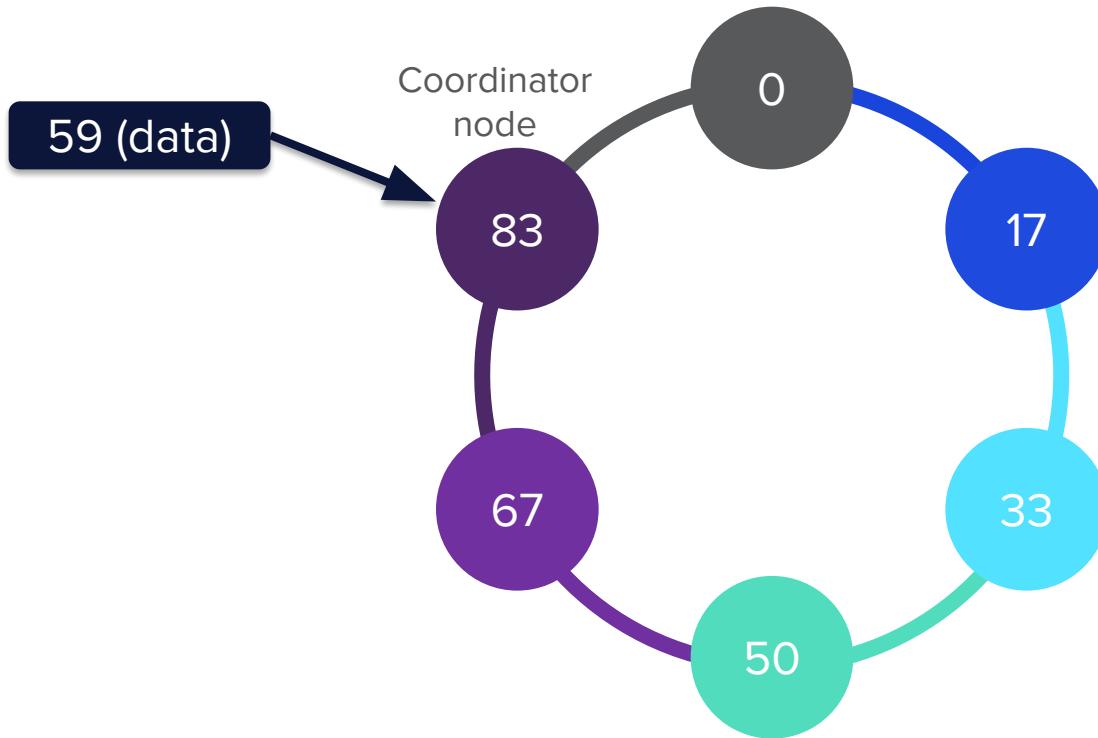
@DataStaxDevs #DataStaxDevelop



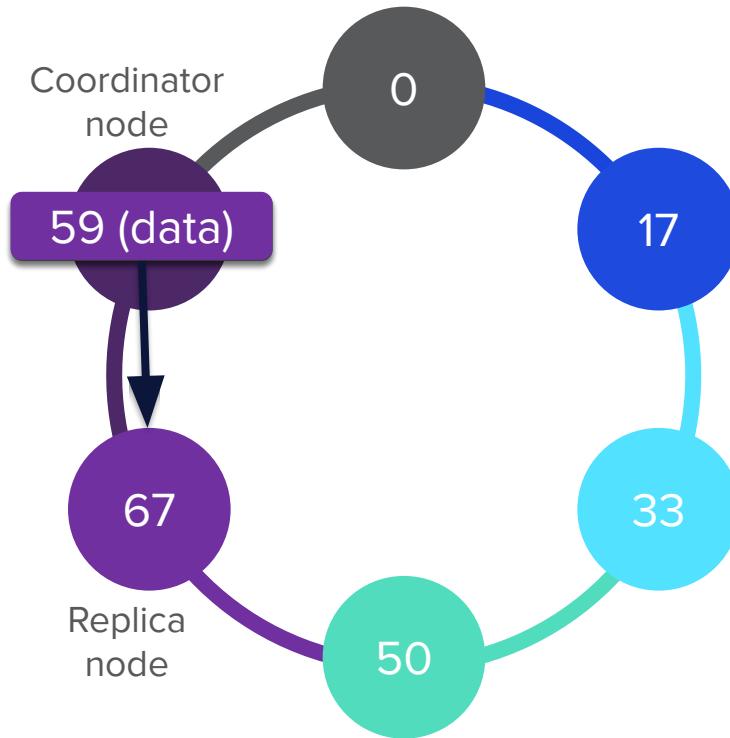
Data is *Evenly-distributed*



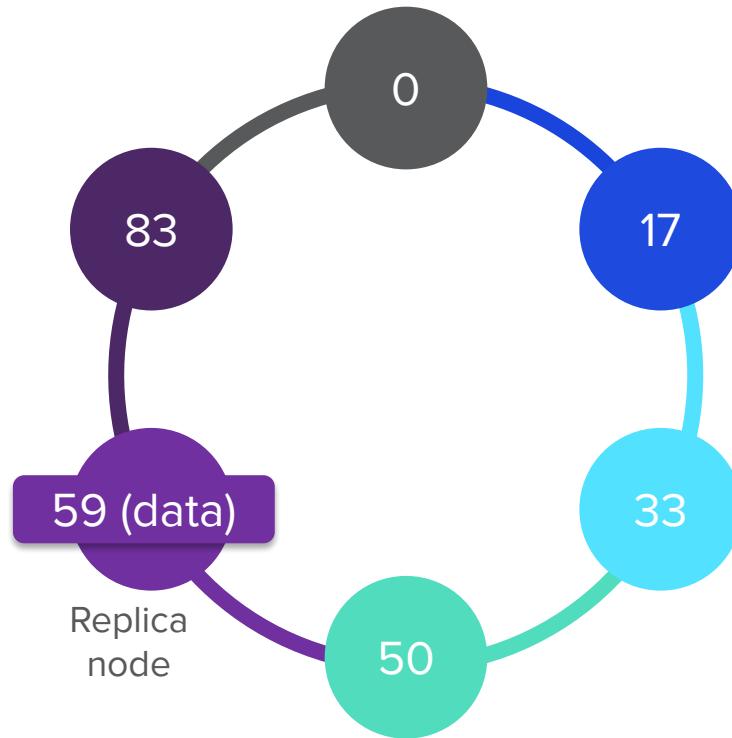
How the Ring Works



How the Ring Works

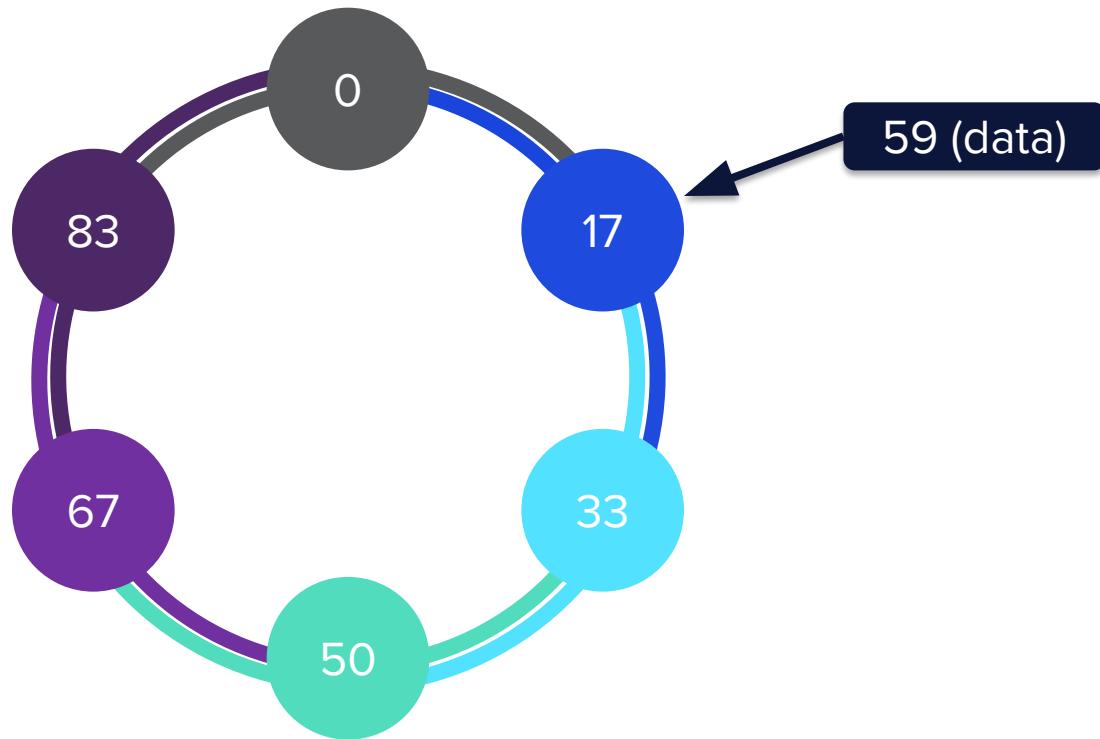


How the Ring Works



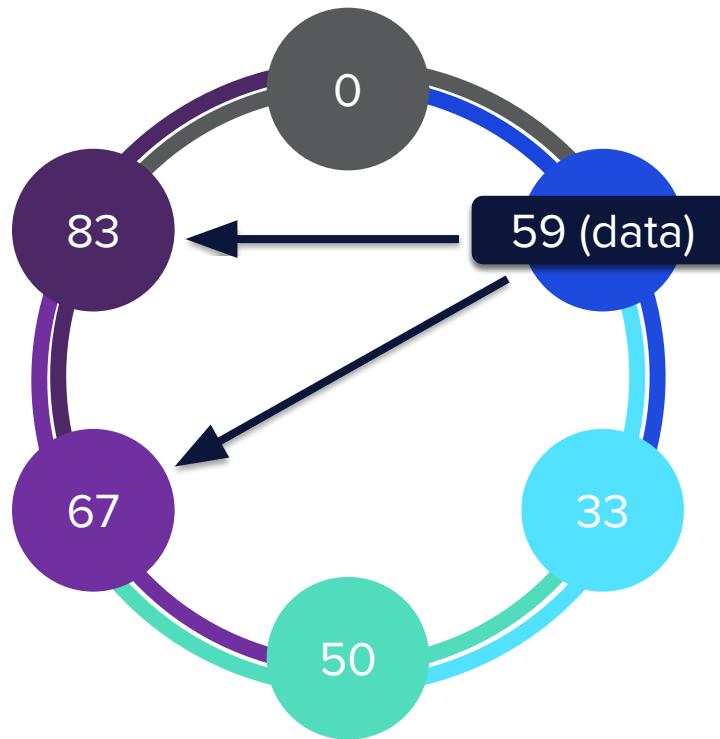
Replication within the Ring

RF = 2



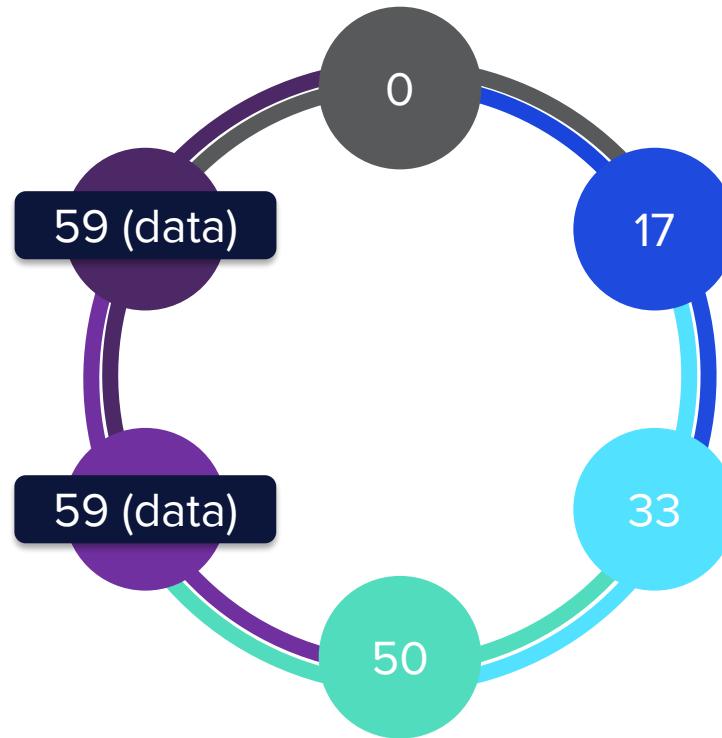
Replication within the Ring

RF = 2



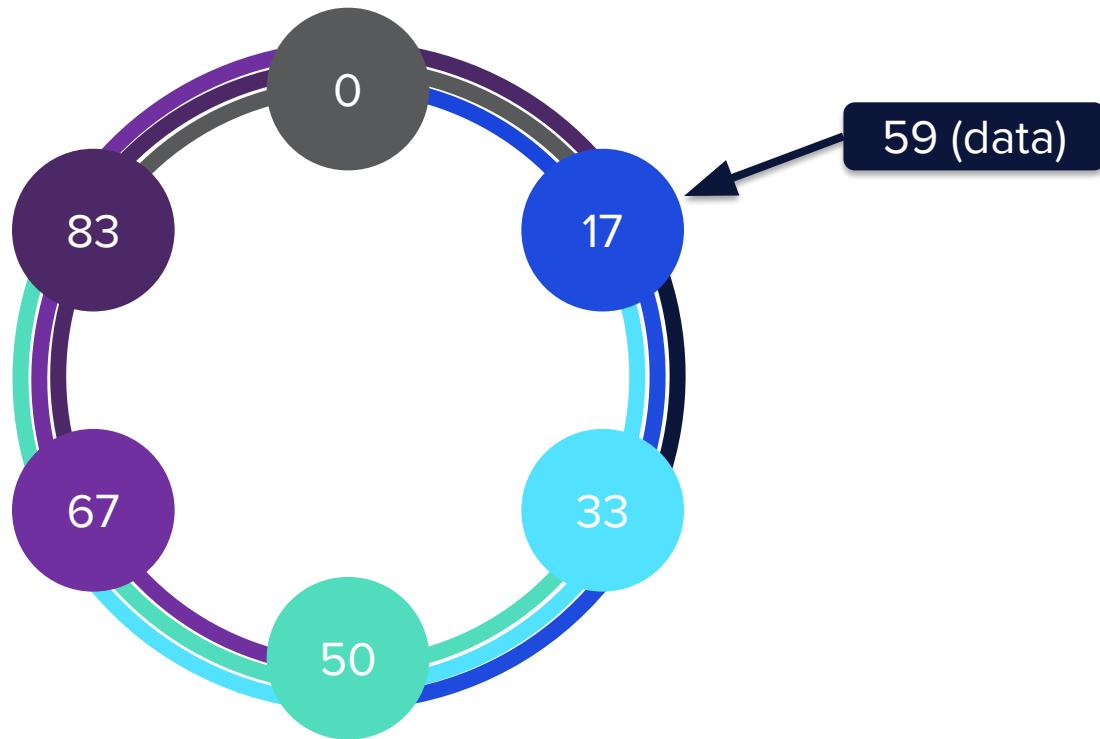
Replication within the Ring

RF = 2



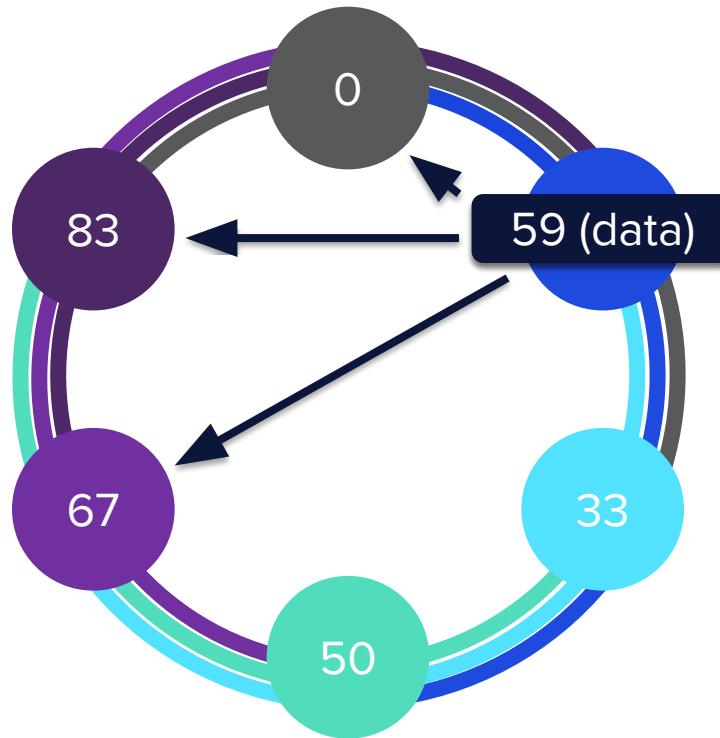
Replication within the Ring

RF = 3



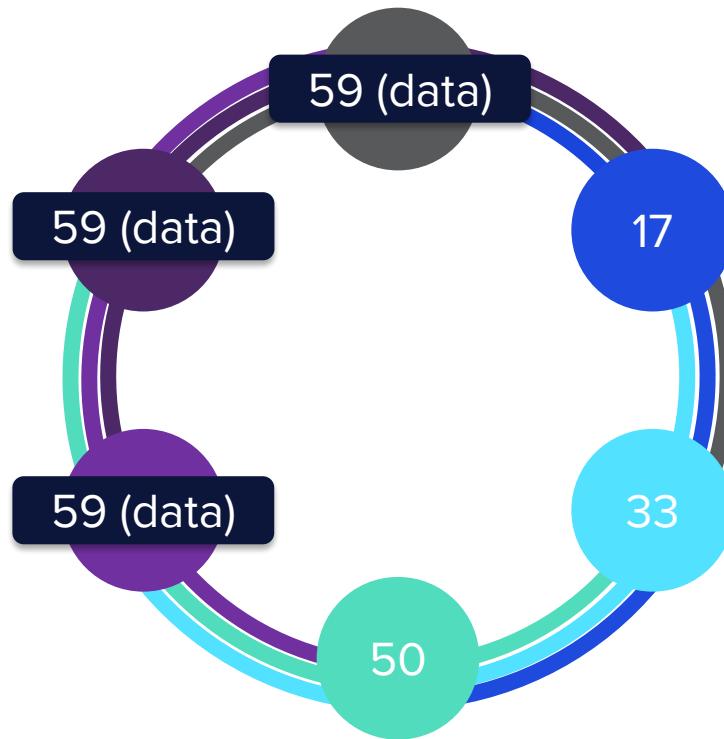
Replication within the Ring

RF = 3



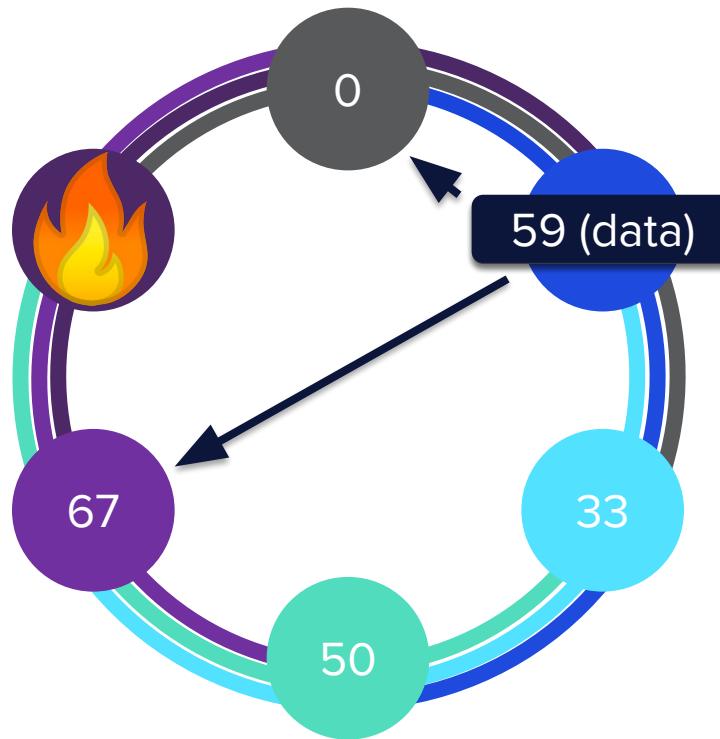
Replication within the Ring

RF = 3



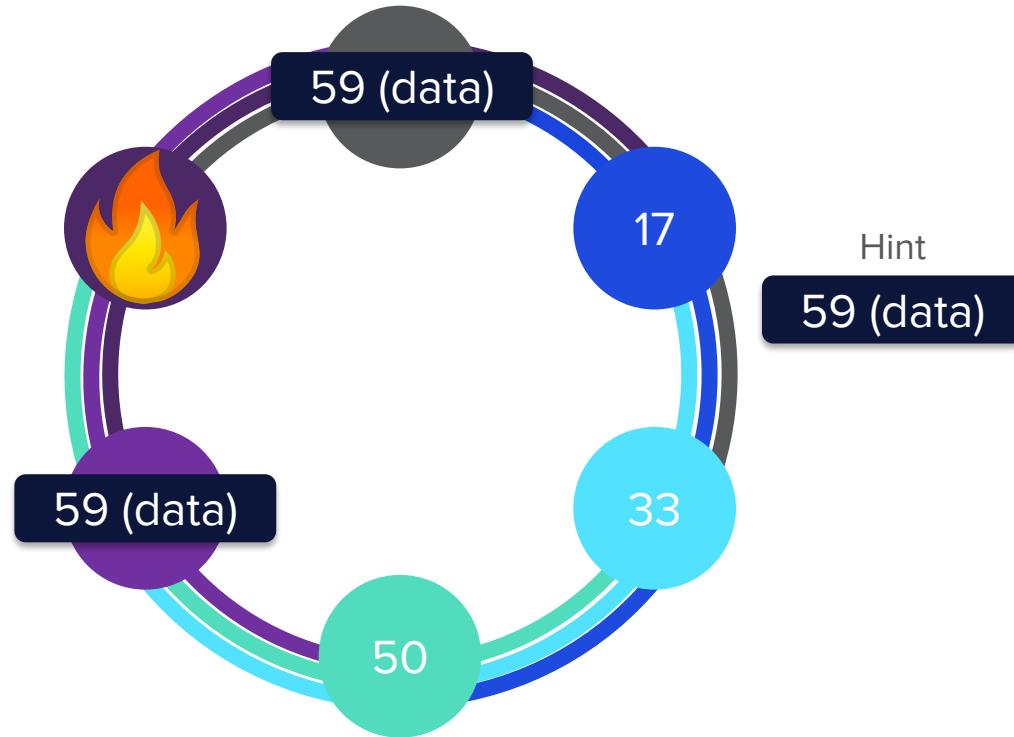
Node Failure

RF = 3



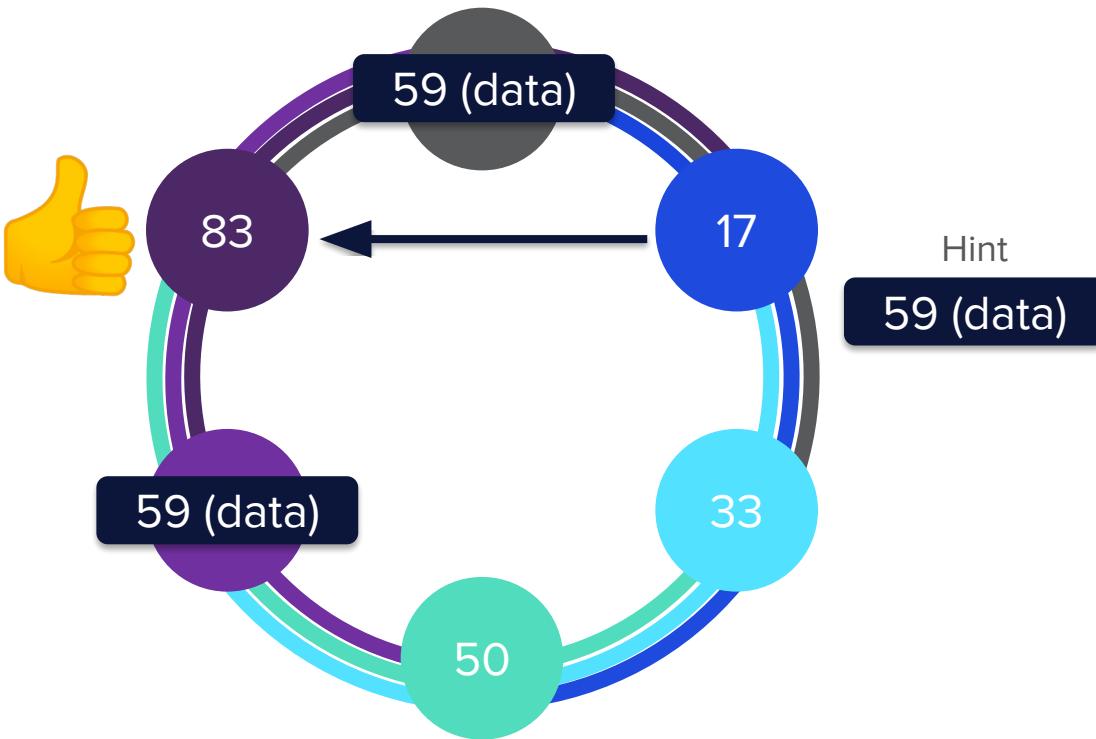
Node Failure

RF = 3



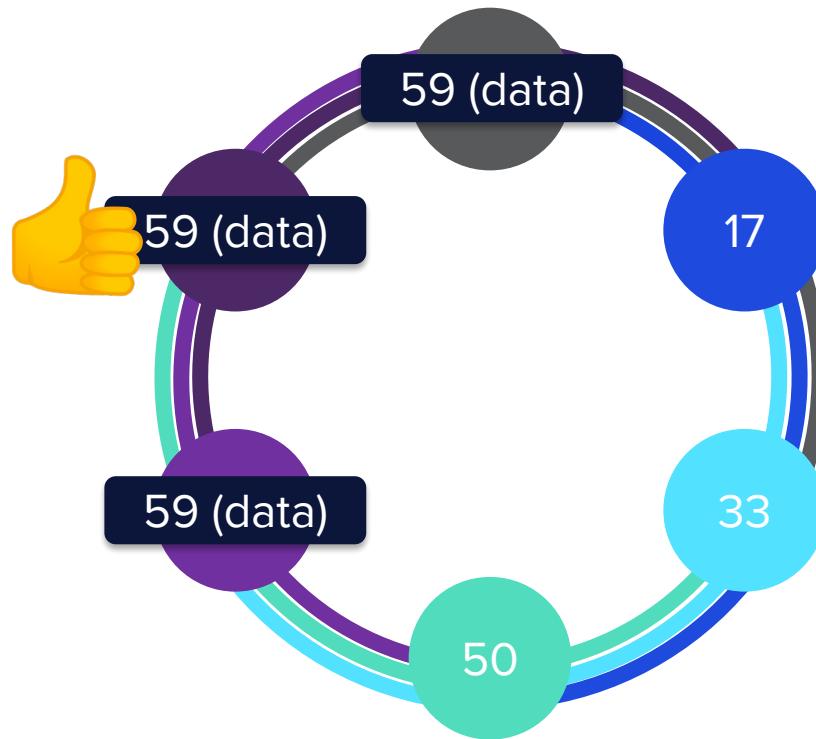
Node Failure

RF = 3



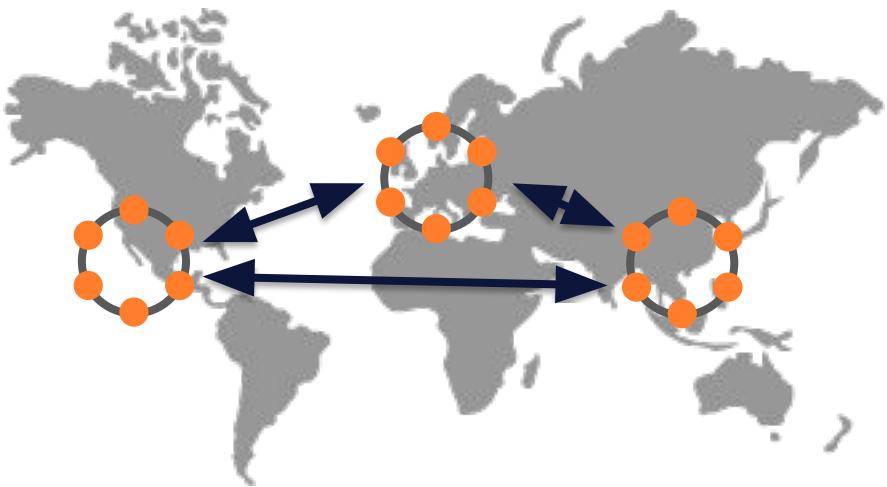
Node Failure – Recovered!

RF = 3

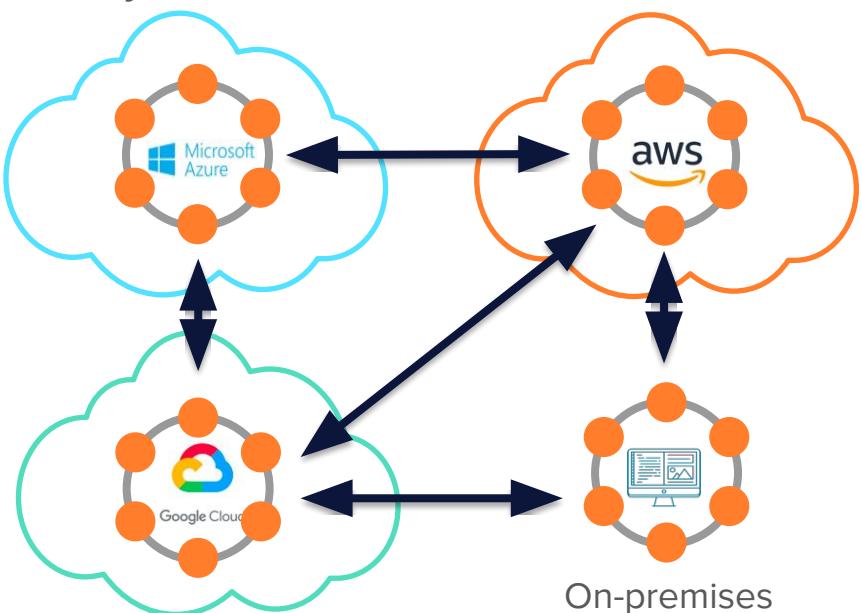


Data Distributed Everywhere

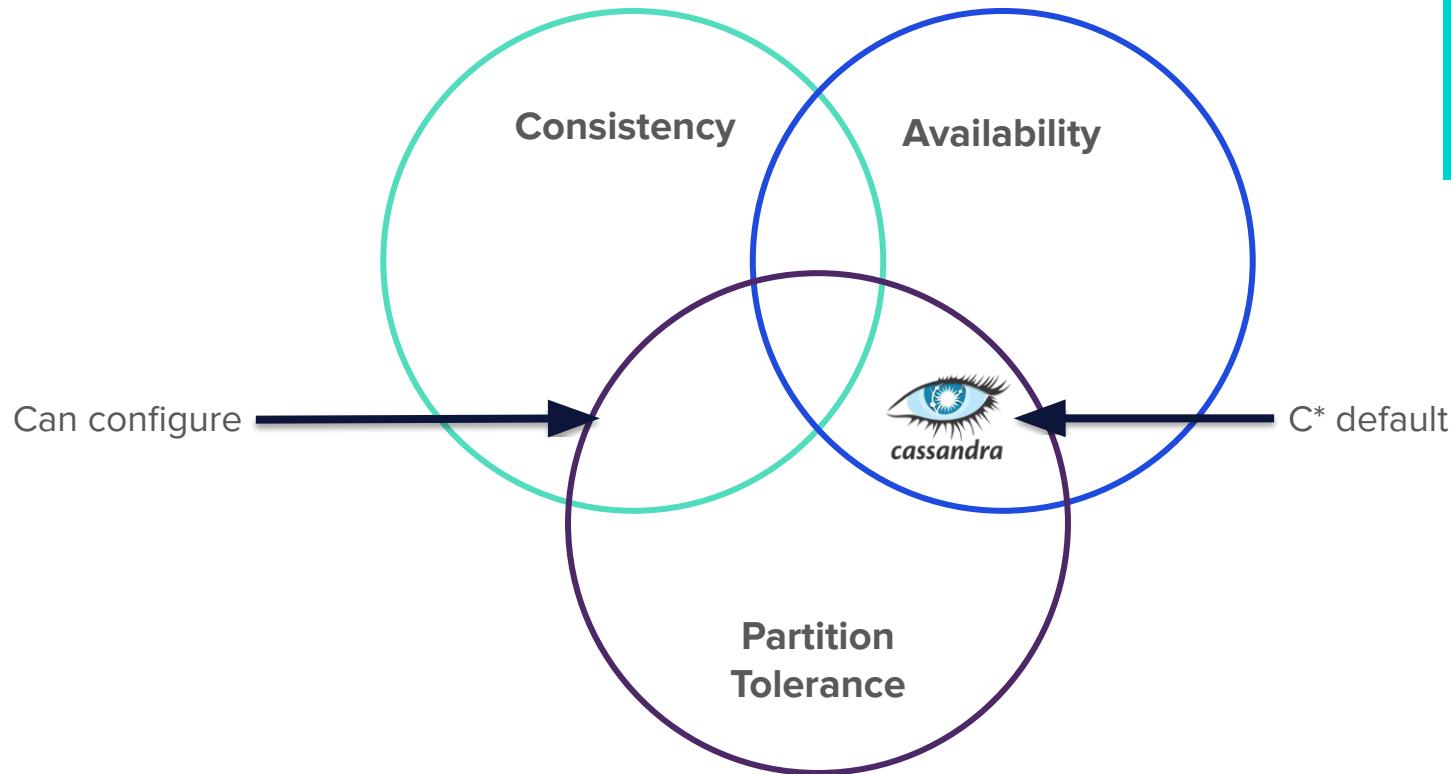
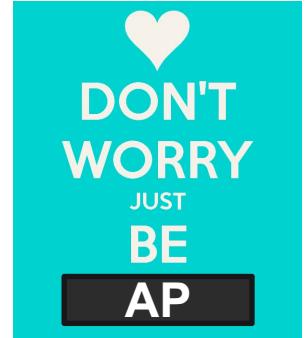
- Geographic Distribution



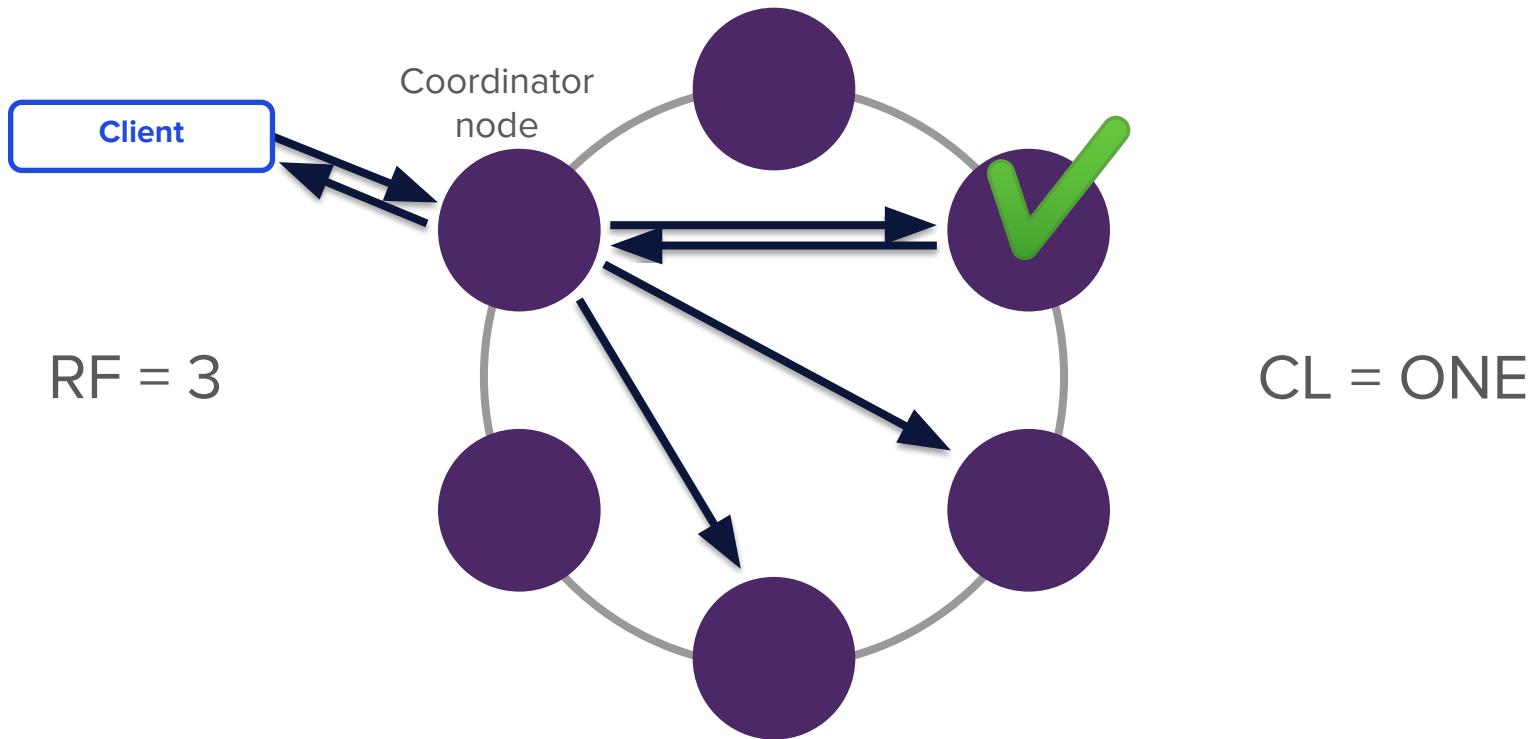
- Hybrid-Cloud and Multi-Cloud



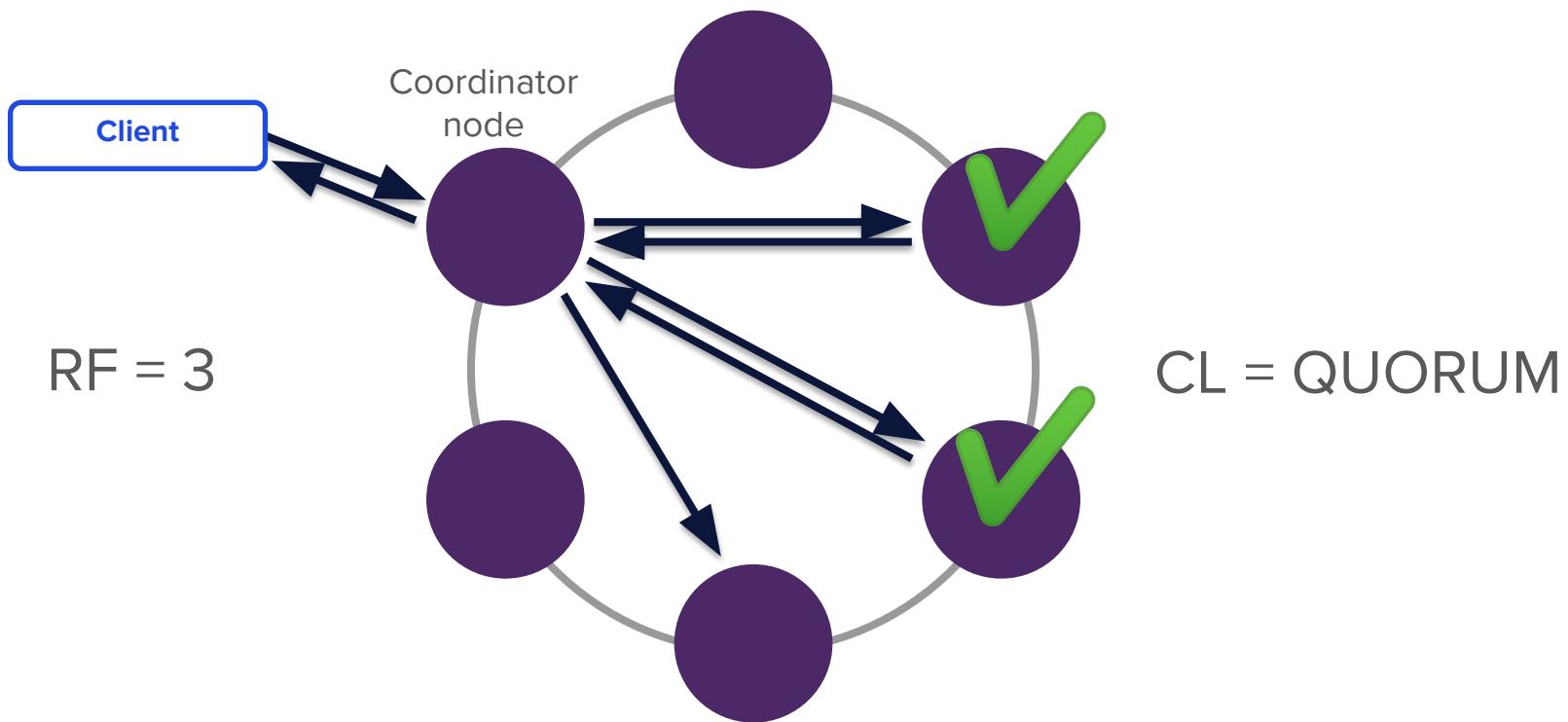
Cap Theorem



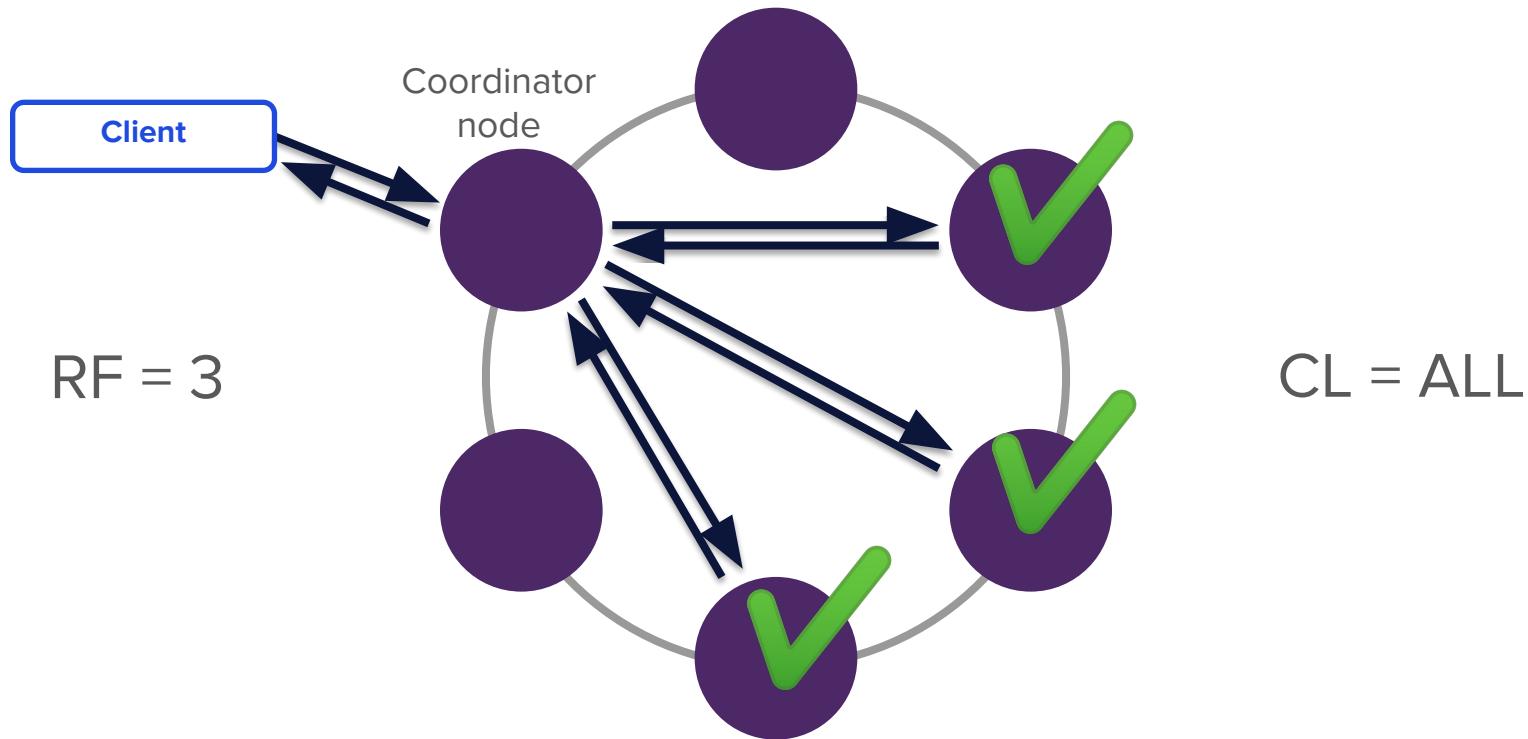
Consistency Levels



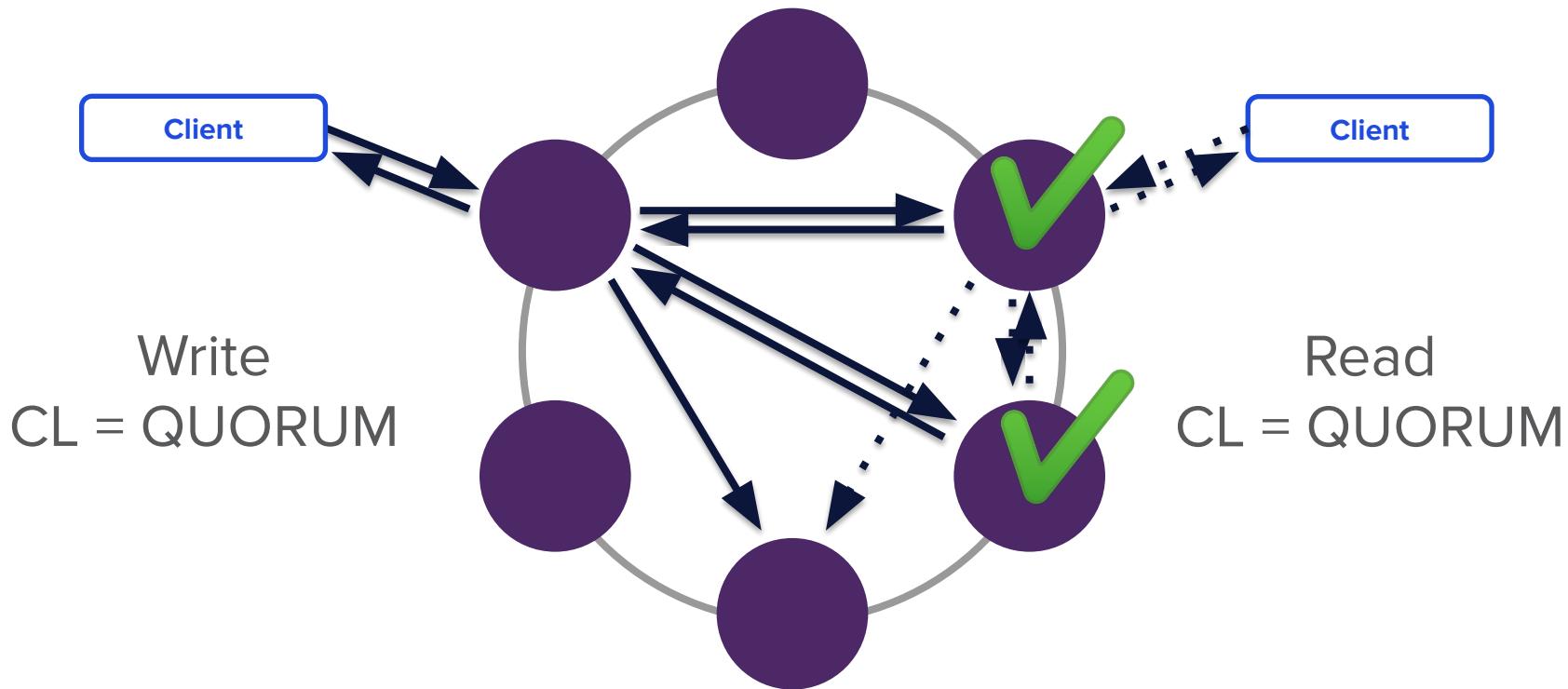
Consistency Levels



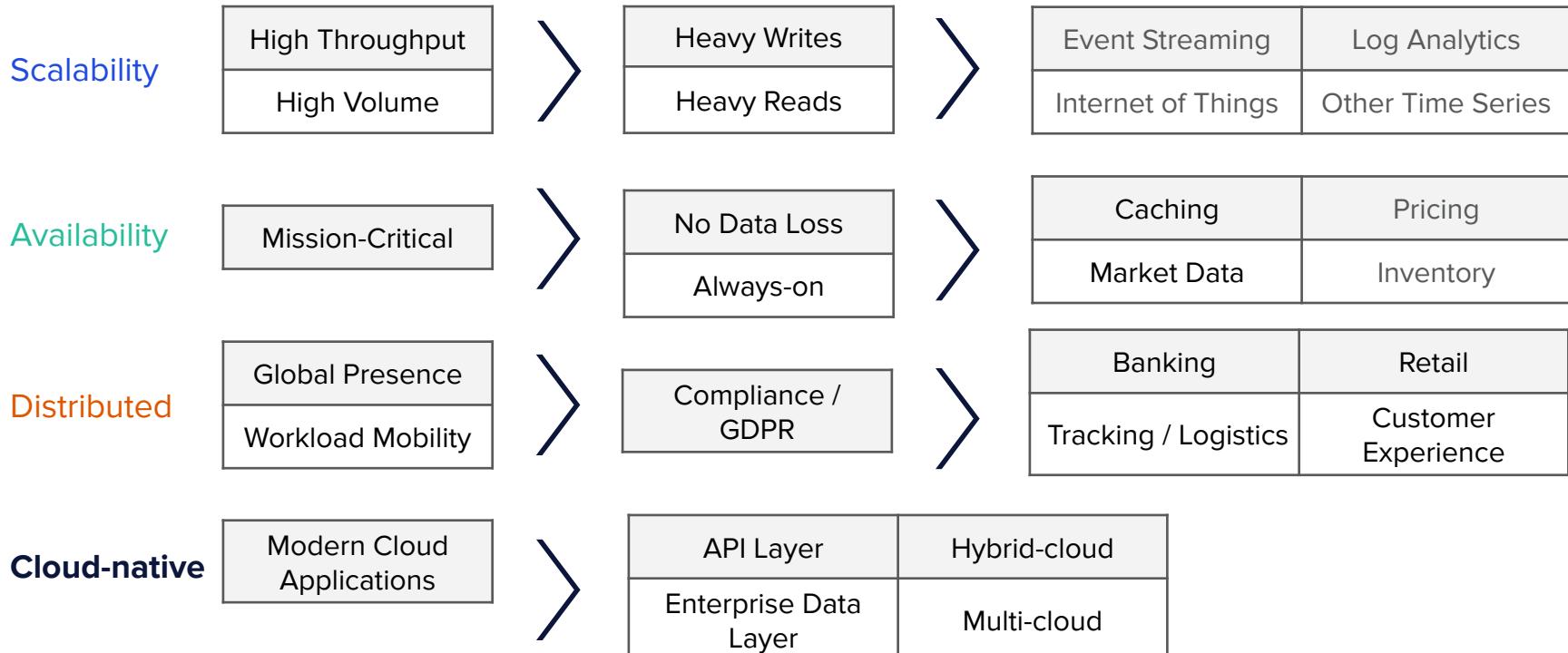
Consistency Levels



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



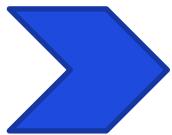
Understanding Use Cases



Exercise 2



DataStax Studio



Cassandra Developer Workshop #2 - Datastax Studio cedrick.lunven@datasta... Schema ?

Use the Markdown Editor

In this section, you will do the following things:

- Expand the cell to see the markdown code editor
- Edit the markdown
- Render the markdown to show your changes

To start with, there are two sections to cells, the code editor and the results section. You are currently looking at the results section and the code editor is hidden.

Step 1: Let's switch to the code editor. Hover over the right-hand corner of this cell and click the icon that looks like an eye.

The screenshot shows the DataStax Studio interface with a title bar. The main content area has a heading "Use the Markdown Editor". Below it, a text block says: "In this section, you will do the following things:" followed by a bulleted list. A callout bubble points to the top-right corner of a code editor window with the text "Click here to see the code editor". The code editor window has a "Language: Markdown" dropdown and some icons. At the bottom of the editor, a text input field contains placeholder text "Replace this text with your name".

Now, let's make a change to the markdown to see how it works.

Step 2: Scroll the bottom of the code editor window and find "Hello your_name_goes_here". Replace *your_name_goes_here* with your name.

Developer Workshop

1

Bootstrapping

2

Apache Cassandra™ Why, What & When

3

Data Modeling with Apache Cassandra™

4

What's NEXT ?

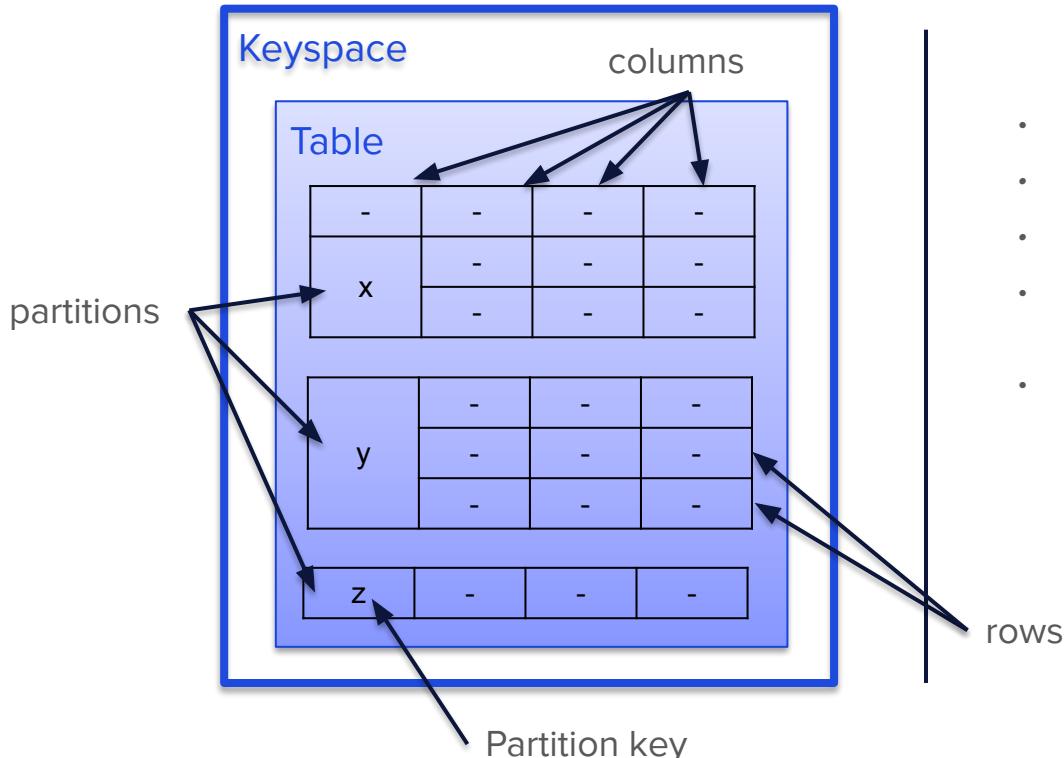


@DataStaxDevs #DataStaxDeveloperDay

<https://community.datastax.com>



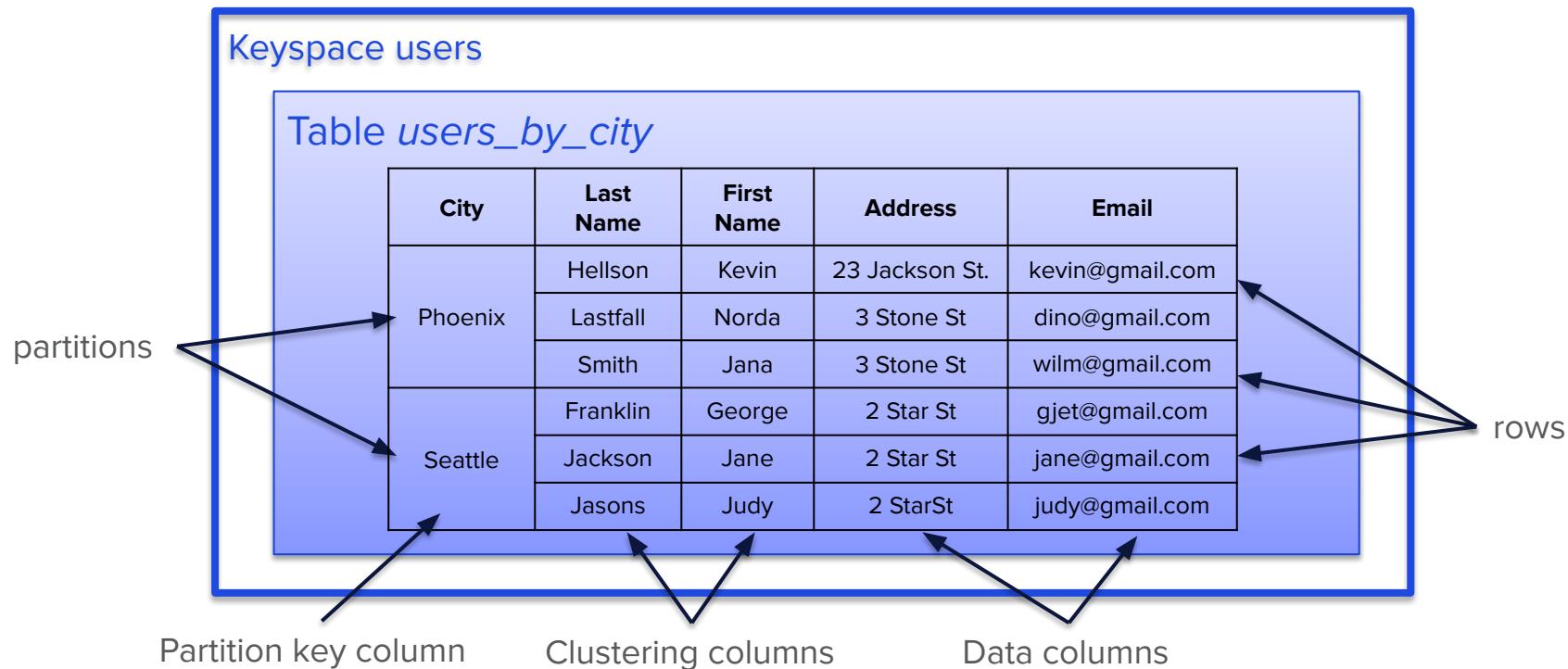
Cassandra Structure - Partition



- 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	---	---	---	---
	---	---	---	---
	---	---	---	---

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



Tables Hold Many Partitions

Table *users_by_city*

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

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

Charlotte	---	---	---	---
	---	---	---	---
	---	---	---	---
	---	---	---	---
	---	---	---	---



Creating a Keyspace in CQL

```
CREATE KEYSPACE users
  WITH REPLICATION = {
    'class' : 'NetworkTopologyStrategy',
    'datacenter1' : 3
};
```

keyspace

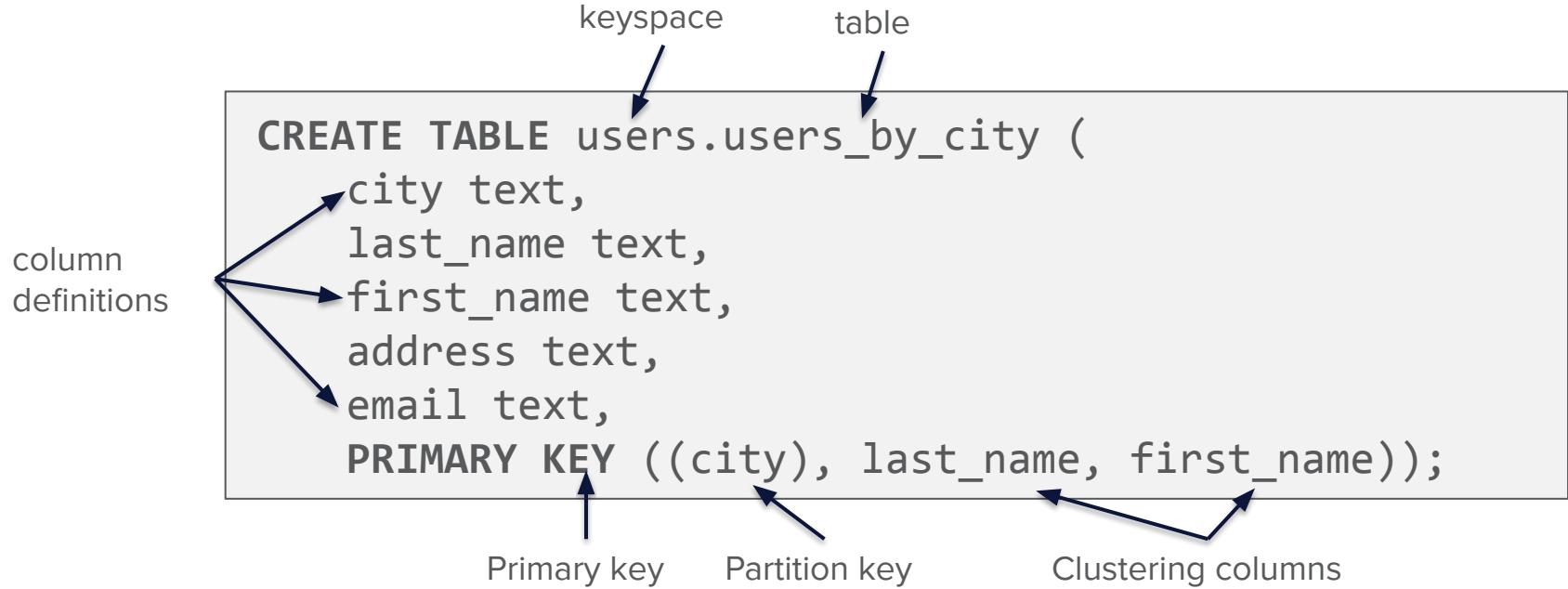
replication strategy

Replication factor by data center

```
graph TD; A[CREATE] --> B[KEYSPACE]; C[REPLICATION] --> D["WITH REPLICATION"]; E[Replication Factor] --> F["datacenter1"];
```

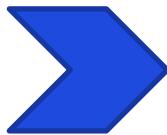


Creating a Table in CQL



Exercise 3

Working with CQL



DataStax Studio Cassandra Developer Workshop #3 - Working with CQL cedrick.lunven@datastax... Schema

Language: **Markdown**

DESCRIBE KEYSPACE Command

In this section, you will do the following things:

- Inspect the `killrvideo` keyspace

Step 1: In the following CQL cell, execute a `DESCRIBE KEYSPACE` command for the `killrvideo` keyspace.

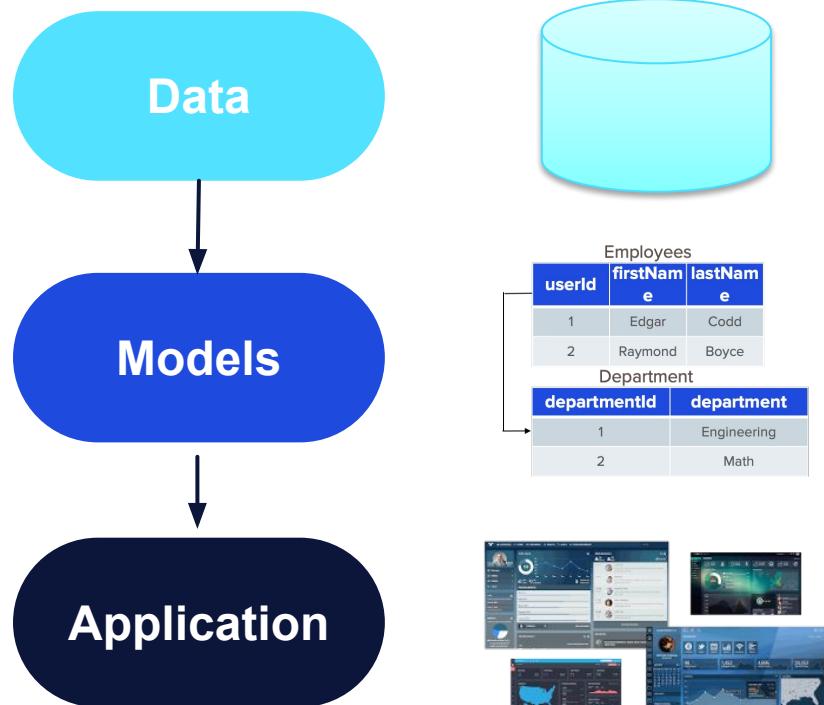
▶ Need a command hint? [Click here.](#)

▶ Just want to copy the command? [Click here.](#)

Language: **CQL** Keyspace: **killrvideo**

Schema
killrvideo ▾
 Tables
 comments_by_user
 comments_by_video
 latest_videos
 tags_by_letter
 user_credentials
 user_videos
 users
 video_playback_stats
 video_ratings
 video_ratings_by_user
 video_recommendations
 video_recommendations_by_video
 videos
 videos_by_tag
 User Defined Types
 User Defined Functions
 User Defined Aggregates
 Materialized Views

Relational Modeling



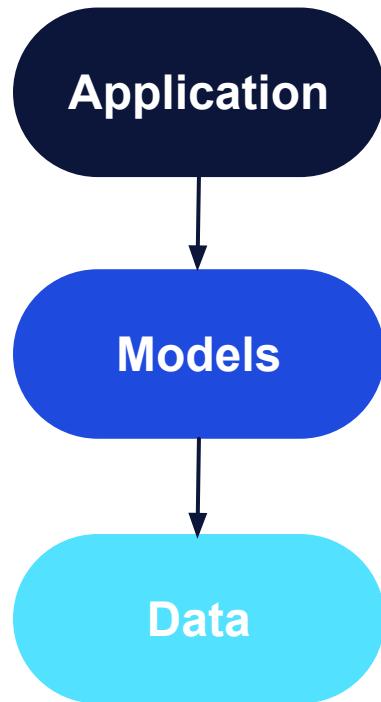
Employees		
userId	firstName	lastName
1	Edgar	Codd
2	Raymond	Boyce

Department

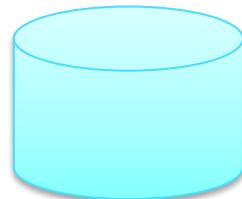
departmentId	department
1	Engineering
2	Math



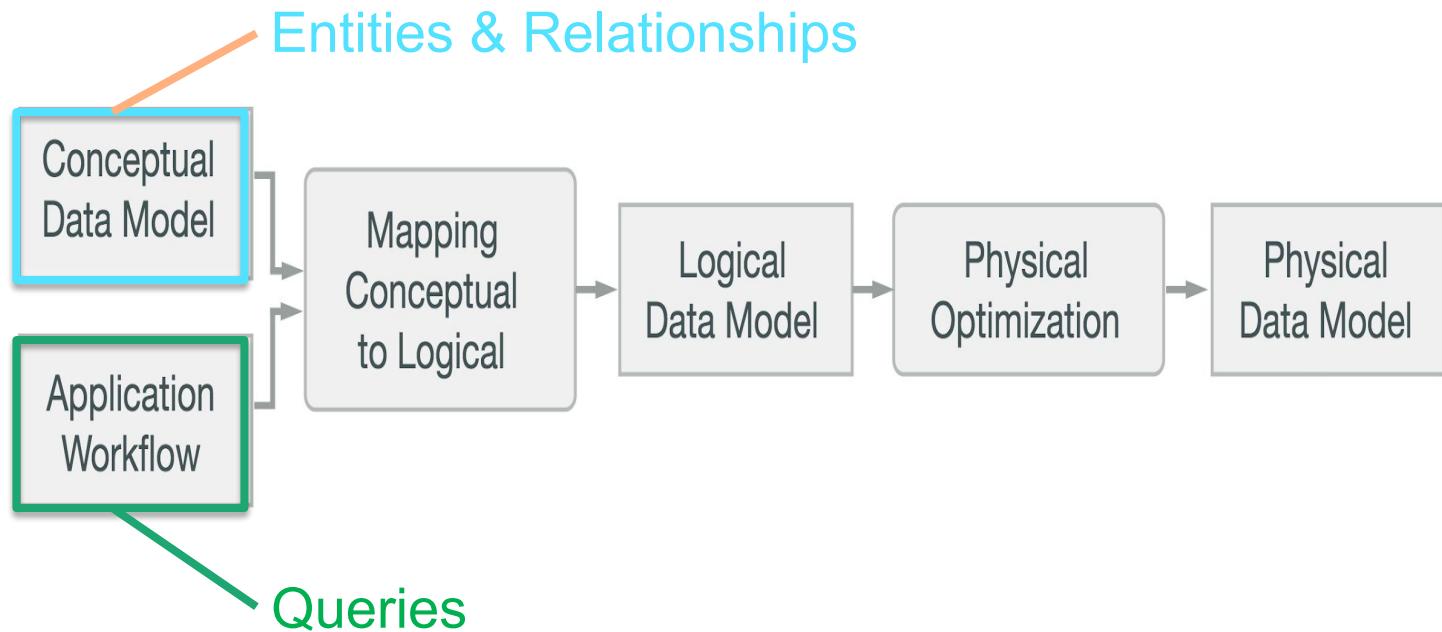
Cassandra Modeling



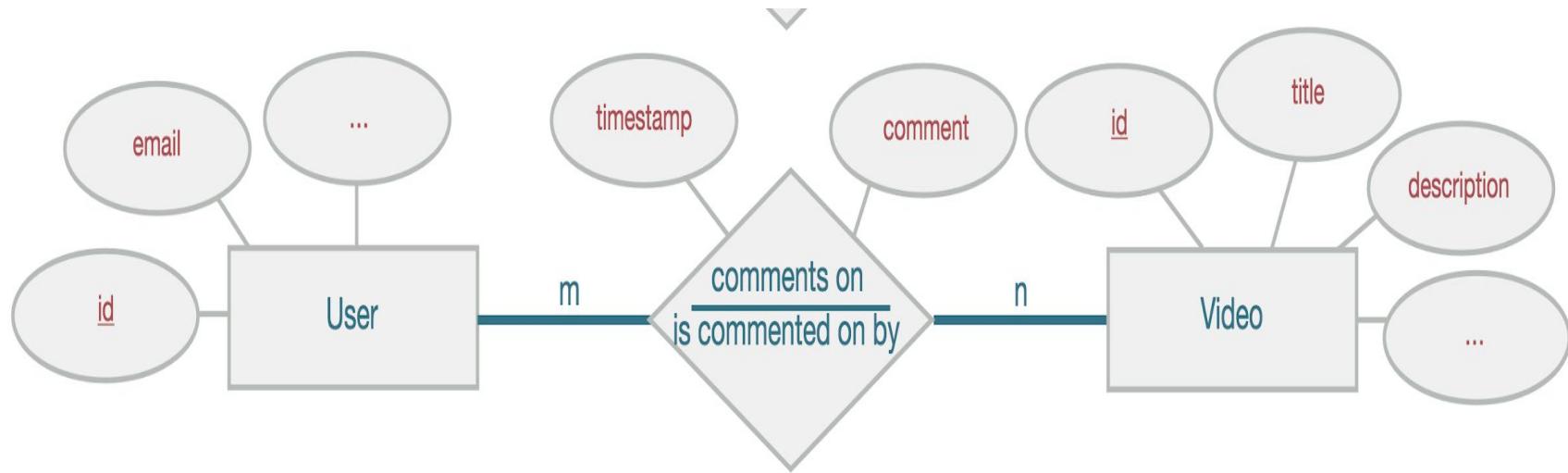
id	firstName	lastName	department
1	Edgar	Codd	Engineering
2	Raymond	Boyce	Math



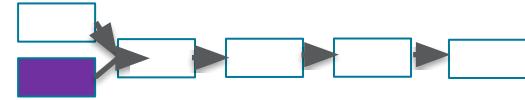
Designing Data Model



Conceptual Data Model



Application Workflow



R1: Find **comments** related to target **video** using its identifier

- Get most recent first
- Implement Paging

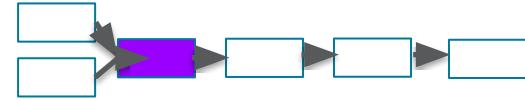
R2: Find **comments** related to target **user** using its identifier

- 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`

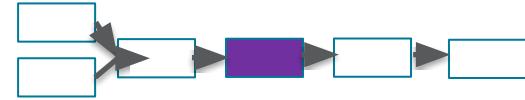
Q2: Find comments posted for a user with a known id (show most recent first)

→ `comments_by_user`

Q3: CRUD Operations



Logical Data Model



comments_by_user

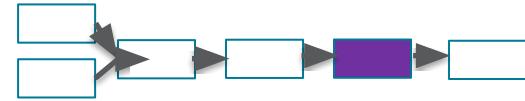
userid	K
creationdate	C ↓
commentid	C ↑
videoid	
comment	

comments_by_video

videoid	K
creationdate	C ↓
commentid	C ↑
userid	
comment	



Physical Data Model



comments_by_user

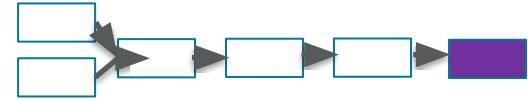
userid	UUID	K
commentid	TIMEUUID	C ↓
videoid	UUID	
comment	TEXT	

comments_by_video

videoid	UUID	K
commentid	TIMEUUID	C ↓
userid	UUID	
comment	TEXT	



Schema DDL

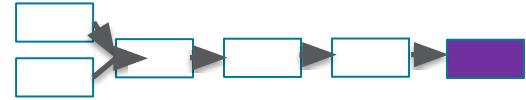


```
CREATE TABLE IF NOT EXISTS comments_by_user (
    userid uuid,
    commentid timeuuid,
    videoid uuid,
    comment text,
    PRIMARY KEY ((userid), commentid)
) WITH CLUSTERING ORDER BY (commentid DESC);
```

```
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



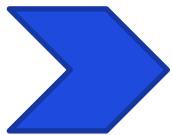
```
SELECT * FROM comments_by_user  
WHERE userid = <some UUID>
```

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



Exercise 4

Data Modelling



The screenshot shows the DataStax Studio interface. The title bar reads "Cassandra Developer Workshop #3 - Working with CQL". The left sidebar shows the schema for the "killrvideo" keyspace, listing various tables and other database objects. The main pane displays a Markdown document titled "DESCRIBE KEYSPACE Command". It contains instructions for inspecting the keyspace and provides links for command hints and copying the command. At the bottom, it shows the CQL language and the "killrvideo" keyspace.

Language: **Markdown**

DESCRIBE KEYSPACE Command

In this section, you will do the following things:

- Inspect the `killrvideo` keyspace

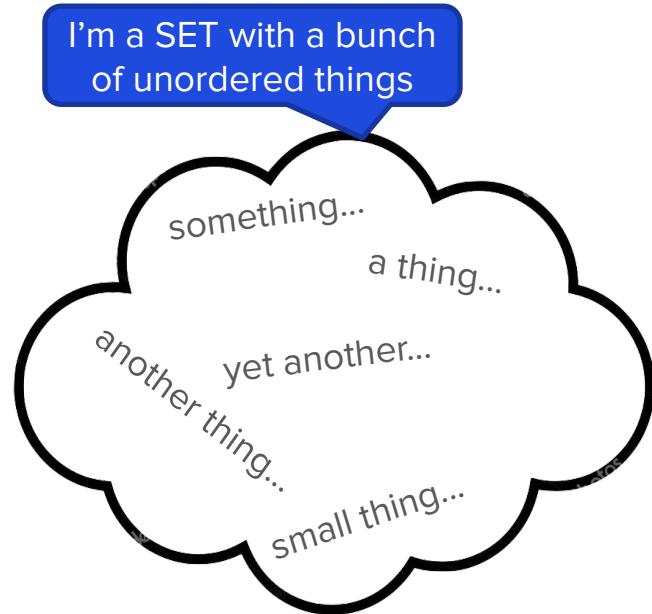
Step 1: In the following CQL cell, execute a `DESCRIBE KEYSPACE` command for the `killrvideo` keyspace.

► *Need a command hint? Click here.*

► *Just want to copy the command? Click here.*

Language: **CQL** Keyspace: **killrvideo**

Collections – Three Types

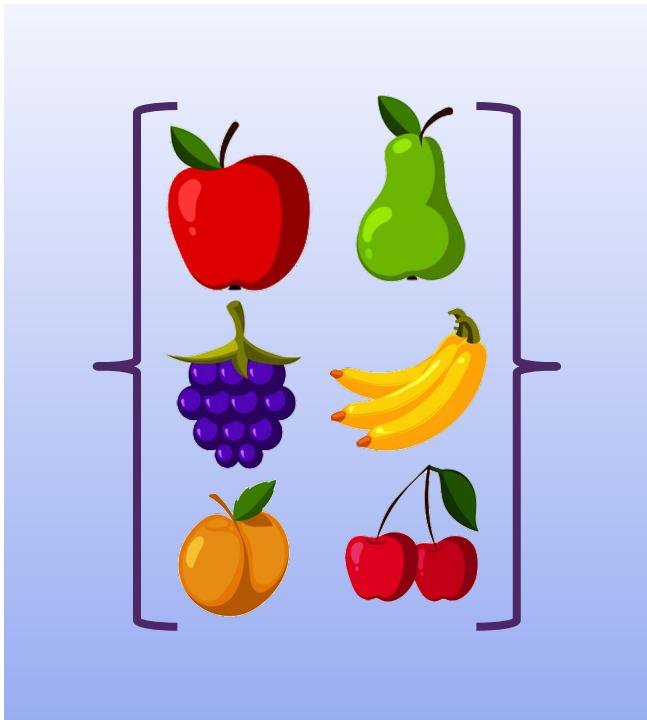


I'm a MAP of
key/value pairs

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

Example Table with Set

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



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
```

Replace an element

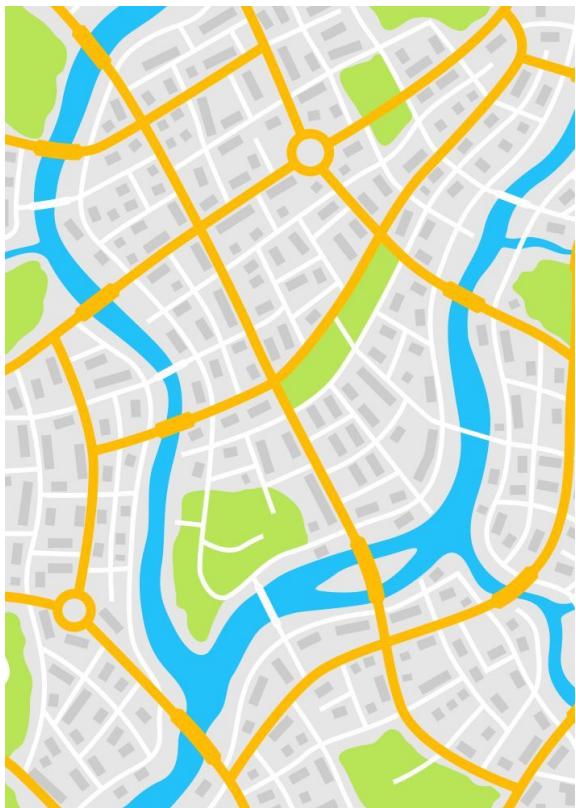
```
SET actors[1] = 'Brown'
```

```
WHERE videoid = 12345678-1234-1234-1234-123456789012;
```

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

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

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

Another Example UDT Operation

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

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:

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

This format must be observed

This can be an integer value

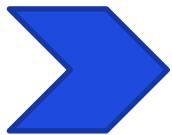
Decrementing a counter:

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

Just change the sign

Exercise 5

Advanced Type



DataStax Studio Cassandra Developer Workshop #5 - Advance... cedrick.lunven@datastax.com Schema ?

Collection Types

In this section, you will do the following things:

- Investigate `SET`, which is one of the collection types
- Insert and retrieve rows in the `videos` table that use `SET`

The `videos` table uses a `SET` collection to keep track of tags associated with each video. A `SET` is a great collection to use because sets do not maintain an order - we are not concerned with any tag order, only if a tag is or is not associated with the video.

Let's start by reviewing the definition of the `videos` table:

Step 1: Execute the following cell to describe the `videos` table.

+

Language: CQL Keyspace: killrvideo

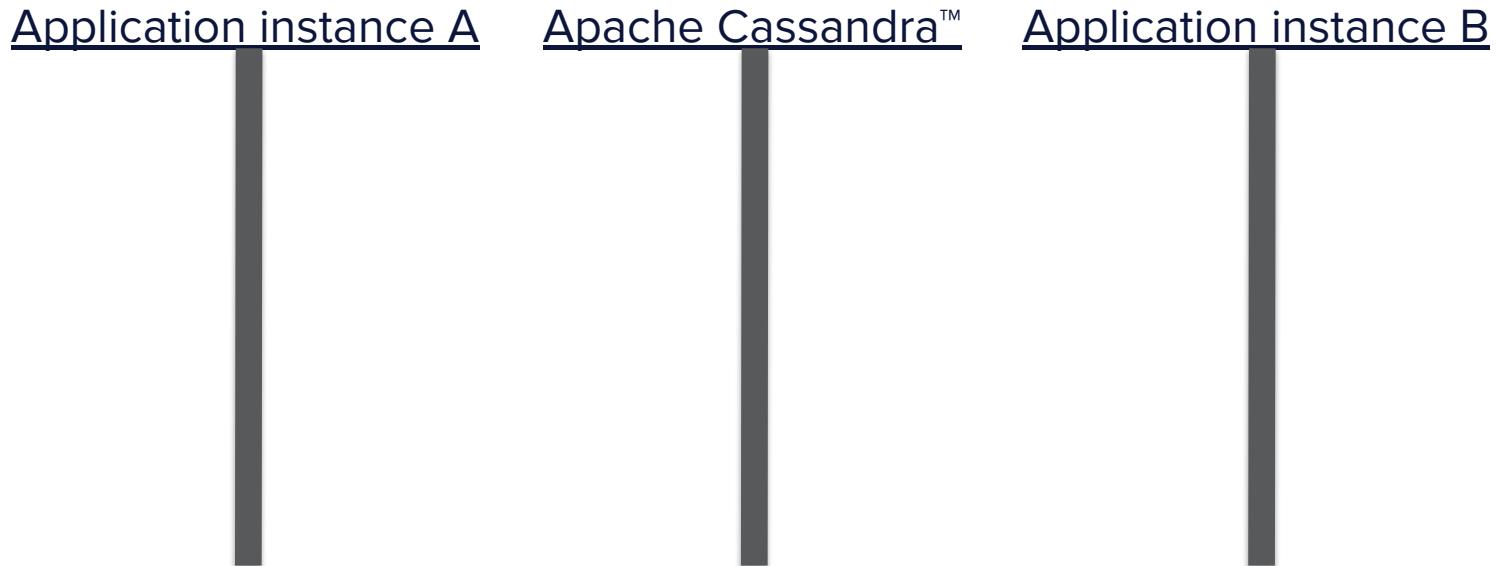
Cassandra Lightweight Transactions

- 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!

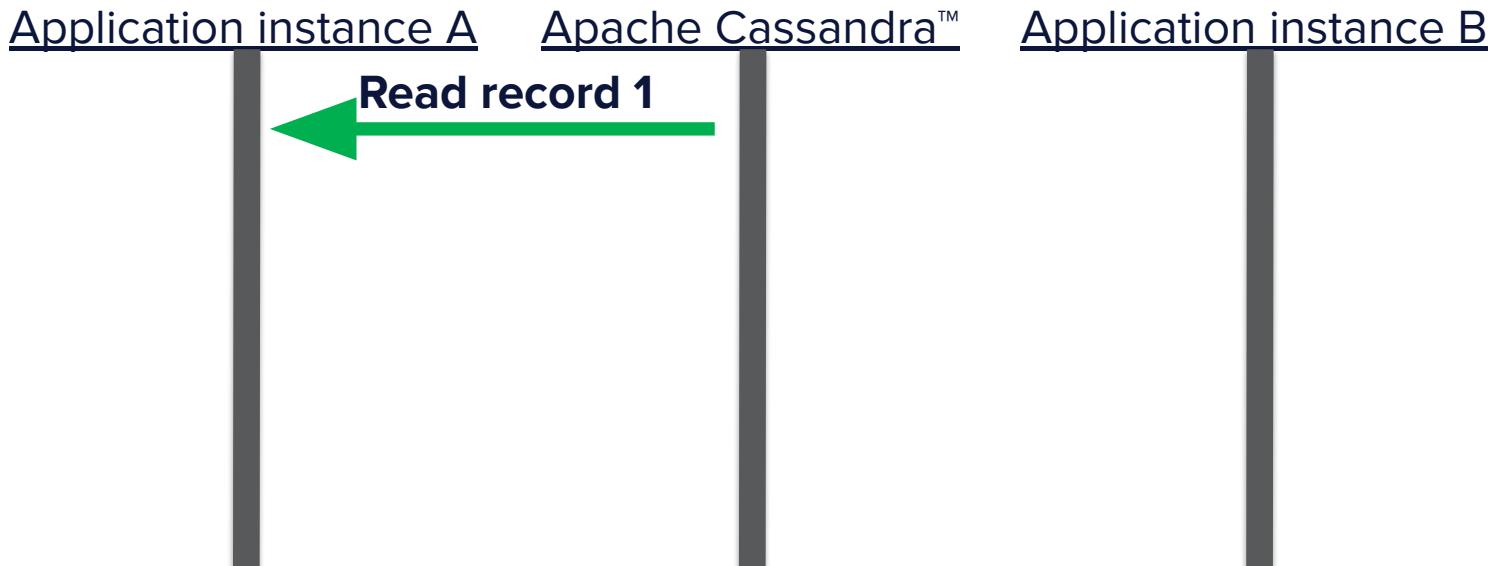
Cassandra Lightweight Transactions

The problem



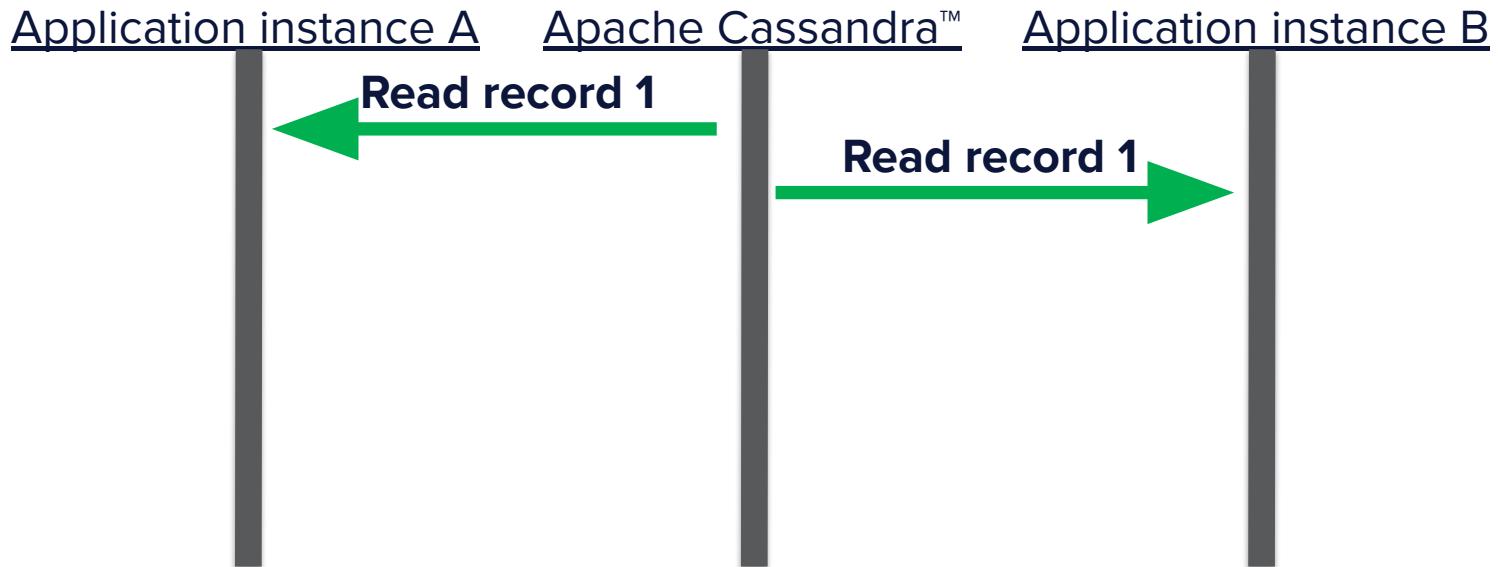
Cassandra Lightweight Transactions

The problem



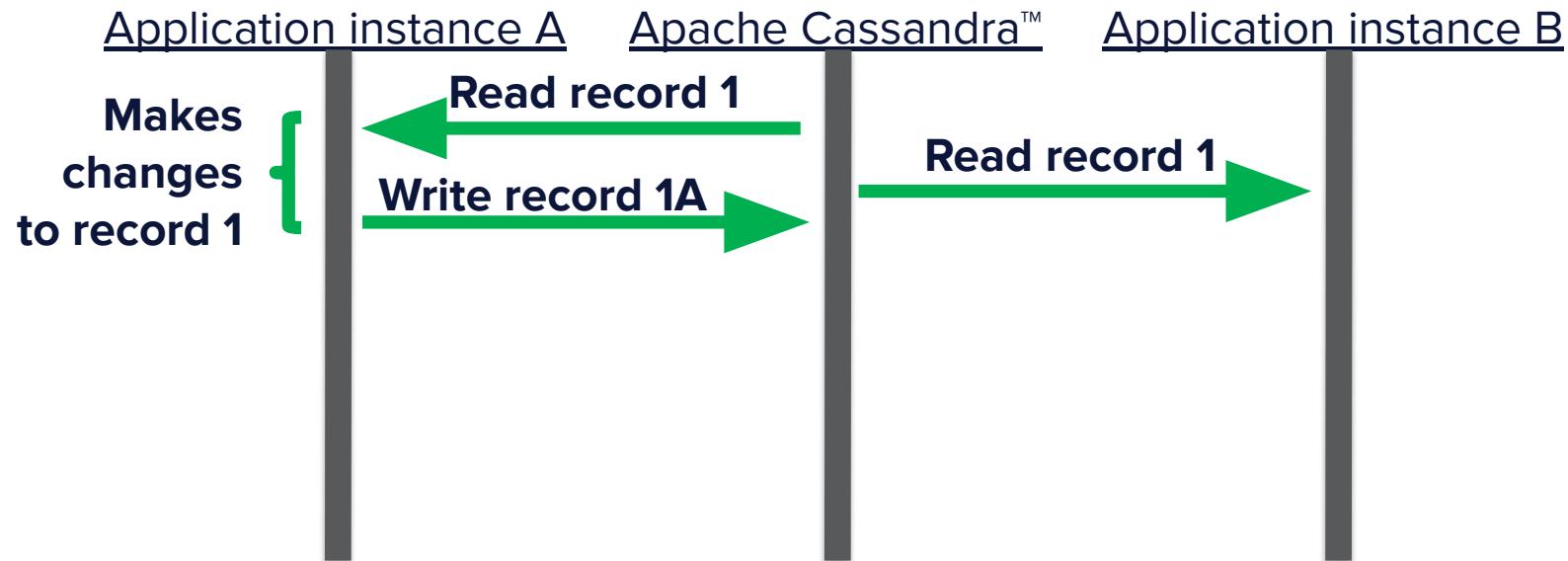
Cassandra Lightweight Transactions

The problem



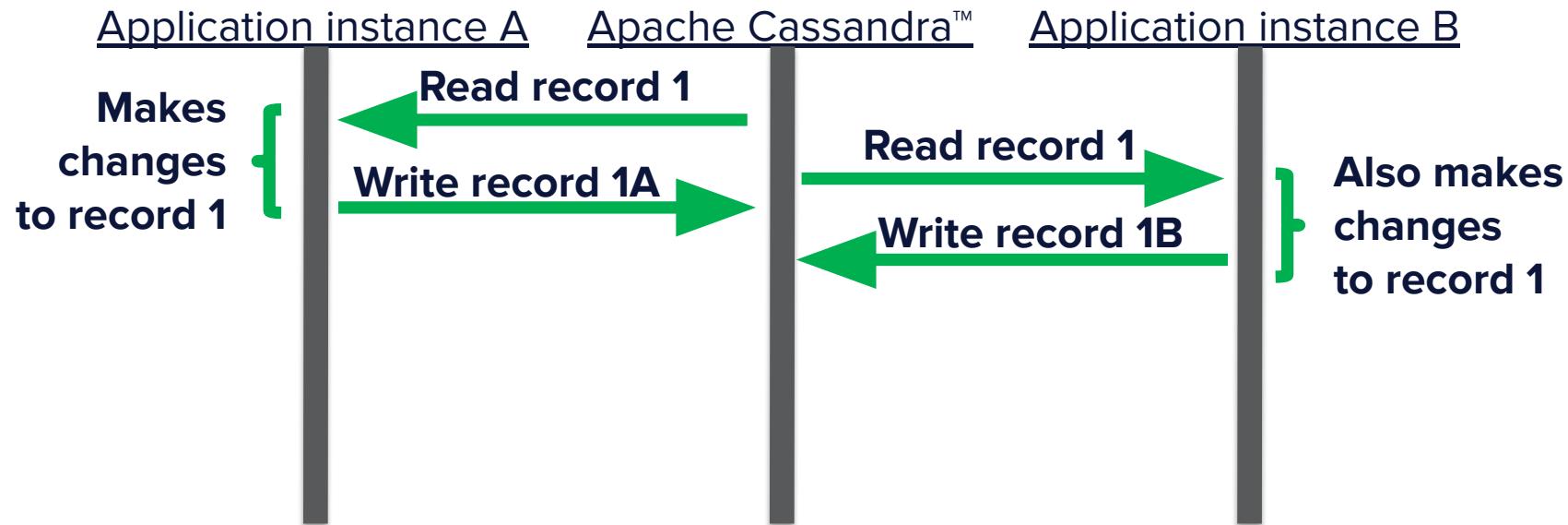
Cassandra Lightweight Transactions

The problem



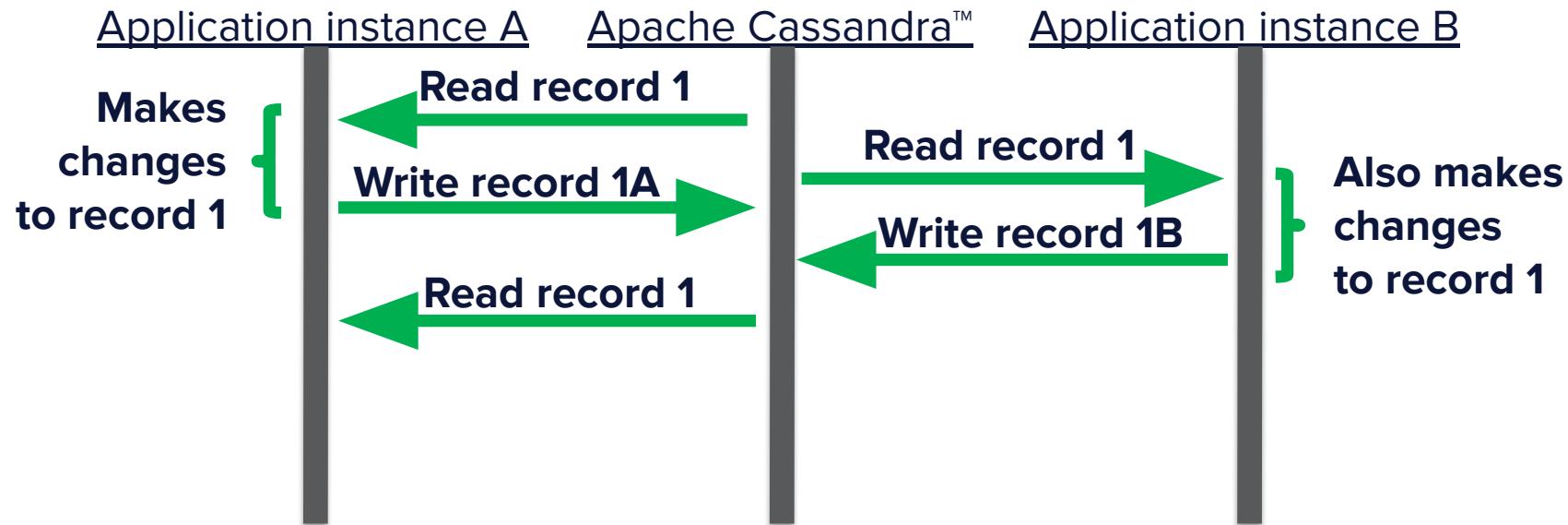
Cassandra Lightweight Transactions

The problem



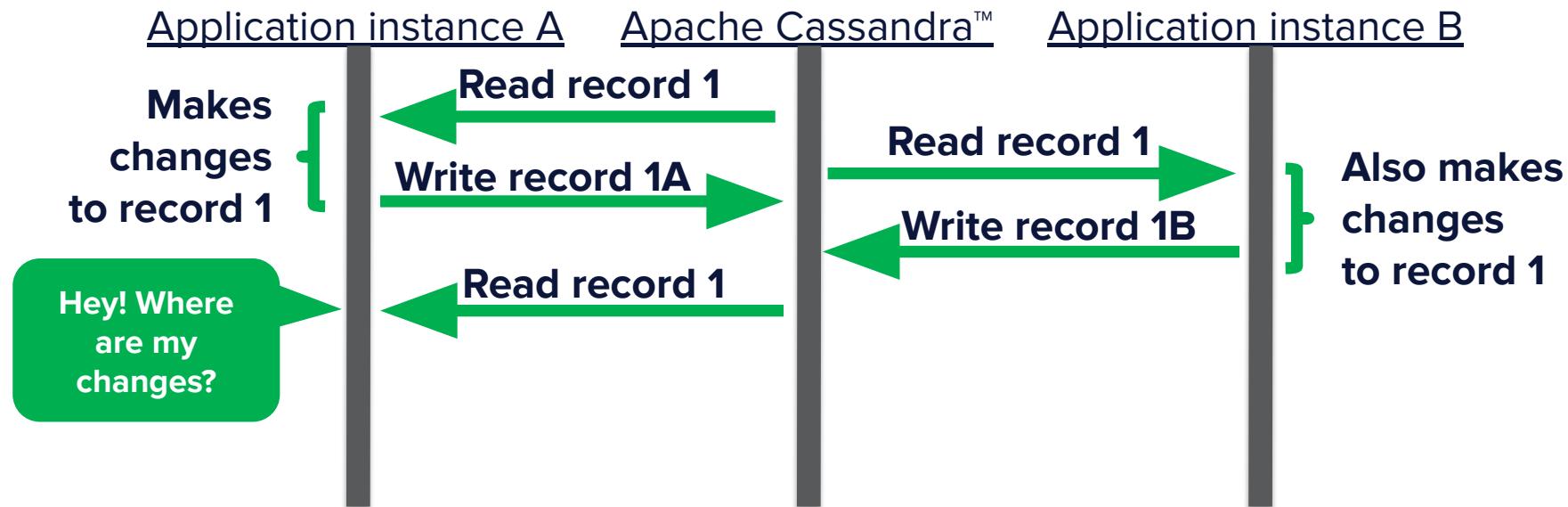
Cassandra Lightweight Transactions

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

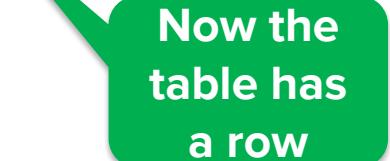
```
User@cqlsh> SELECT * FROM killrvideo.user_credentials WHERE email = 'new_user@gmail.com';  
  
email | password | userid  
-----+-----+-----  
new_user@gmail.com | password_A | 6d8d869d-6ff4-4bd8-be66-d4334f3d2e4b  
  
(1 rows)
```

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;



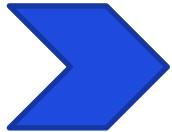
So Why Use Batch?

- 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



The screenshot shows the DataStax Studio interface. On the left, a central window displays a Markdown document with the title "INSERT with Lightweight Transactions (LWTs)". The text describes the goal of the section: "In this section, you will do the following things:" followed by a bulleted list. Below this, there is a section titled "Here's the story:" with a paragraph explaining the scenario of creating a new user while checking for existing users. On the right side of the interface, a "Schema" panel is open, showing the structure of the "killrvideo" CQL Keyspace, which includes various tables like "comments_by_user", "latest_videos", and "user_credentials".

Language: **Markdown**

INSERT with Lightweight Transactions (LWTs)

In this section, you will do the following things:

- Insert a new row using an LWT
- Insert an existing row using an LWT
- Observe the two behaviors (and compare them to an upsert)

Here's the story:

Imagine we want to create a new KillrVideo user. To do so, we need to create an entry in the `user_credentials` table with a specified email address. But we don't want to create a new user if there is a user already with that email address.

We could read the database to determine if the user exists already, but what if, between reading and creating the user, somebody else creates the same user. That would cause an

Schema
killrvideo

Tables

- ▶ comments_by_user
- ▶ comments_by_video
- ▶ latest_videos
- ▶ tags_by_letter
- ▶ user_credentials
- ▶ user_videos
- ▶ users
- ▶ video_playback_stats
- ▶ video_ratings
- ▶ video_ratings_by_user
- ▶ video_recommendations
- ▶ video_recommendations_by_video
- ▶ videos
- ▶ videos_by_tag

▶ User Defined Types

▶ User Defined Functions

▶ User Defined Aggregates

▶ Materialized Views

DataStax Drivers

- CQL Support
 - - Connection Pooling
 - - Auto Node Discovery
 - - SSL
 - - Compression
 - - Query Builder
 - - Object Mapper
- Sync / Async API
- Address Translation
- Load Balancing Policies
- Retry Policies
- Reconnection Policies
- Synchronous, Asynchronous, Reactive



• ODBC
• JDBC

- 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 execute queries ?

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

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

Statement

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);  
  
cqSession.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()** to designate parameters that will be provided later

```
// Prepared QueryBuilder statements 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
}

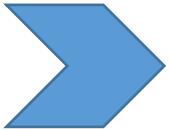
// Use Lambda
rs.forEach(row -> { row.getColumnDefinitions(); });

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

Exercise 7



Coding

A screenshot of a Java IDE (IntelliJ IDEA) showing the code for a Java application named "Delete". The code uses the DataStax Java Driver to delete a row from a database table. The code is as follows:

```
import com.datastax.dse.driver.api.core.DseSession;
import com.datastax.oss.driver.api.core.cql.*;

public class Delete {

    public static void main(String[] args) {
        try (DseSession session = DseSession.builder().withCloudSecureConnectBundle(DBConnection.
                .withAuthCredentials(DBConnection.getUsername(), DBConnection.getPassword()))
                .build()) {
            session.execute(
                    SimpleStatement.builder( "DELETE FROM killrvideo.user_credentials WHERE email =
                        .addPositionalValues("cv@datastax.com")
                        .build());
            System.out.println("Successful Delete");
        }
        catch(Throwable t) {
            System.out.println("Failed Delete");
            t.printStackTrace();
        }
    }
}
```

Developer Workshop

1

Bootstrapping

2

Apache Cassandra™ Why, What & When

3

Data Modeling with Apache Cassandra™

4

What's NEXT ?



@DataStaxDevs #DataStaxDeveloperDay

<https://community.datastax.com>



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](https://twitter.com/DataStaxDevs)

We are on Twitter - Twitch!

REVIEW

Slides and code for this course are available at

<https://github.com/DataStax-Academy/online-Cassandra-workshop>



@DataStaxDevs #DataStaxDeveloperDay

<https://community.datastax.com>



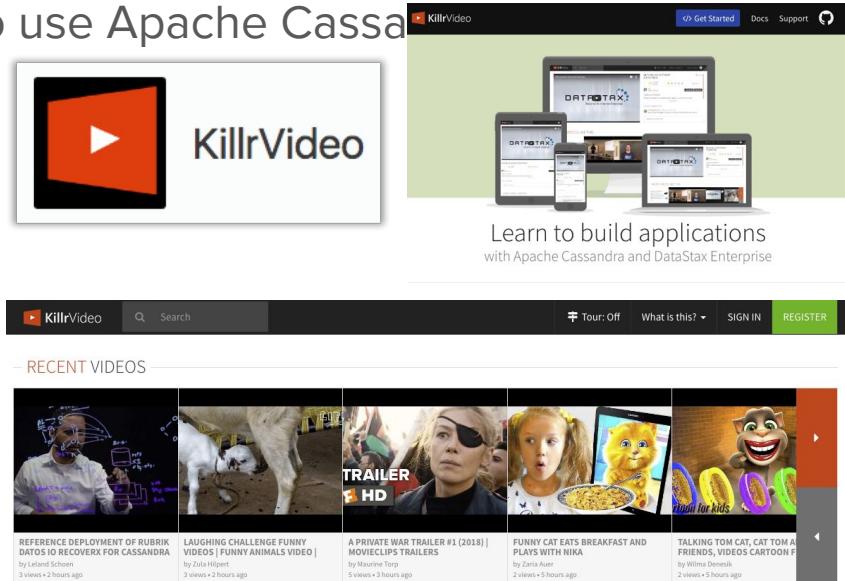
Training Courses at DataStax Academy

- 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 Applicat
 - DS330: DataStax Enterprise 6 Graph
 - DS332: DataStax Enterprise 6 Graph Analytics (I



KillrVideo Reference Application

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



@DataStaxDevs #DataStaxDeveloperDay

<https://community.datastax.com>





Thank You

