

# 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





**David Jones-Gilardi**  
**Developer Advocate**



**Jack Fryer**  
**Community Manager**



**Aleksandr Volochnev**  
**Developer Advocate**





**Cedrick Lunven**  
**Developer Advocate**



**Jack Fryer**  
**Community Manager**



**Aleksandr Volochnev**  
**Developer Advocate**





**Cedrick Lunven**   
**Director of  
Developer Advocacy**



**Jack Fryer**   
**Community  
Manager**



**David Jones-Gilardi**   
**Developer Advocate**



**Aleksandr  
Volochnev**   
**Developer  
Advocate**



@DataStaxDevs #DataStaxDeveloperDay

<https://community.datastax.com>





**Eric Zietlow**

**Developer Advocate**



**David Gilardi**

**Developer Advocate**



@DataStaxDevs #DataStaxDeveloperDay

<https://community.datastax.com>



[www.datastax.com/keepcalm](http://www.datastax.com/keepcalm)

SRE office hours for Apache Cassandra

Cassandra cluster health checks

@ no charge

Send an email to **keepcalm@dastastax.com**



@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>



# Developer Workshop

1

Bootstrapping

2

Apache Cassandra™ Why, What & When

3

Data Modeling with Apache Cassandra™

4

What's NEXT ?



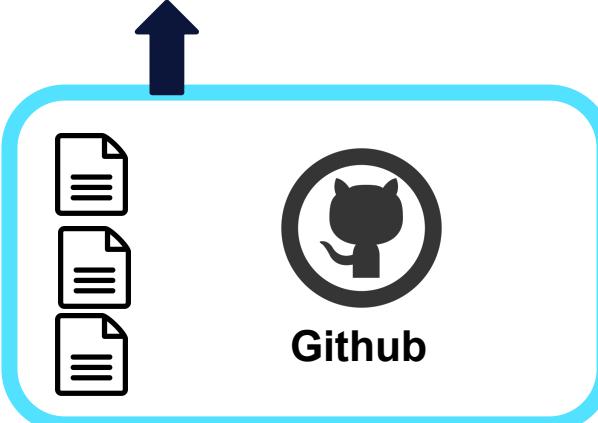
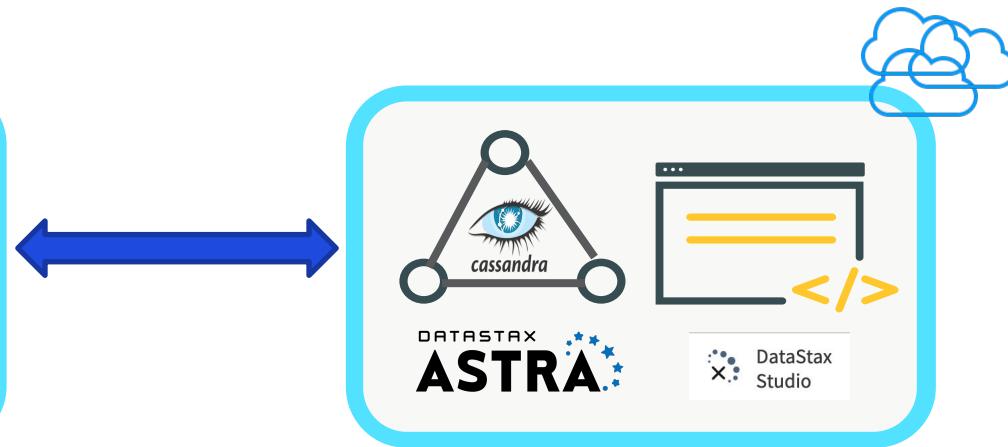
@DataStaxDevs #DataStaxDeveloperDay

<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. Get your FREE certification voucher

<https://bit.ly/cassandra-cert-FREE>

# Introducing Astra



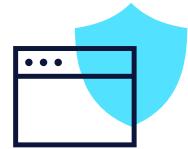
## Eliminate Operations

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

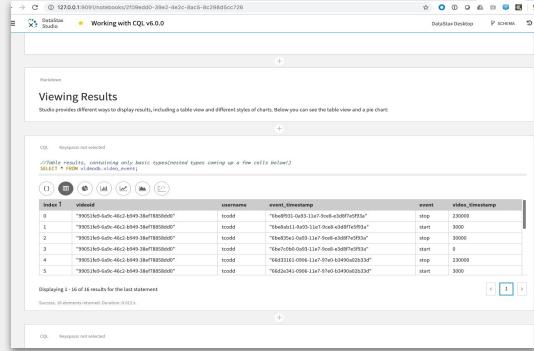


# Simplify Application Development

## 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", "id", "name", "event\_timeStamp", "event", and "value". There are 16 rows of data. The first few rows are:

index	id	name	event_timeStamp	event	value
0	"9995-1d5-40f-4c2-1b9-34f-78785060"	tosdd	"1e4eff01-0a93-11e7-9e0-a53ff0f935a"	stop	33000
1	"9995-1d5-40f-4c2-1b9-34f-78785060"	tosdd	"1e4ebab1-1a93-11e7-9e0-a53ff0f935a"	start	3000
2	"9995-1d5-40f-4c2-1b9-34f-78785060"	tosdd	"1e4ebac1-0a93-11e7-9e0-a53ff0f935a"	stop	30000

## Great Drivers



# Exercise 1



## Create your Astra Instance



The screenshot shows the DataStax Astra interface for creating a new database. The top navigation bar includes "DATASTAX CONSTELLATION", "Your Organization", "Provide Feedback", and a user account. The main content area displays a green banner stating "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." Below this, a "screenshot" card shows details for a database named "okidoki" created by "Cedrick Lunven" on "January 22, 2020". To the right, four cards provide "Size and Location", "Cost", and "Connection Details" information. The "Size and Location" card shows 1 Capacity Unit, 500 GB Total Storage, and the location "us-east-1 (1 capacity unit)". The "Cost" card shows "Spent this month: TBD" and "Estimated Cost" options for "Per Hour" or "Per Month". The "Connection Details" card notes that connection details are only available for active databases.

Size and Location	
1 Capacity Unit	500 GB Total Storage
Storage Used	
Locations: us-east-1 (1 capacity unit)	
Compute Size: Startup	
Replication Factor: 3	

Cost	
Spent this month: TBD	
Estimated Cost	
<input checked="" type="radio"/> Per Hour	<input type="radio"/> Per Month
when running	TBD/hour
when parked	TBD/hour

Connection Details	
Connection details are only available for active databases that you own or have connection permissions for.	

# Developer Workshop

1

Bootstrapping

2

**Apache Cassandra™ Why, What & When**

3

Data Modeling with Apache Cassandra™

4

What's NEXT ?

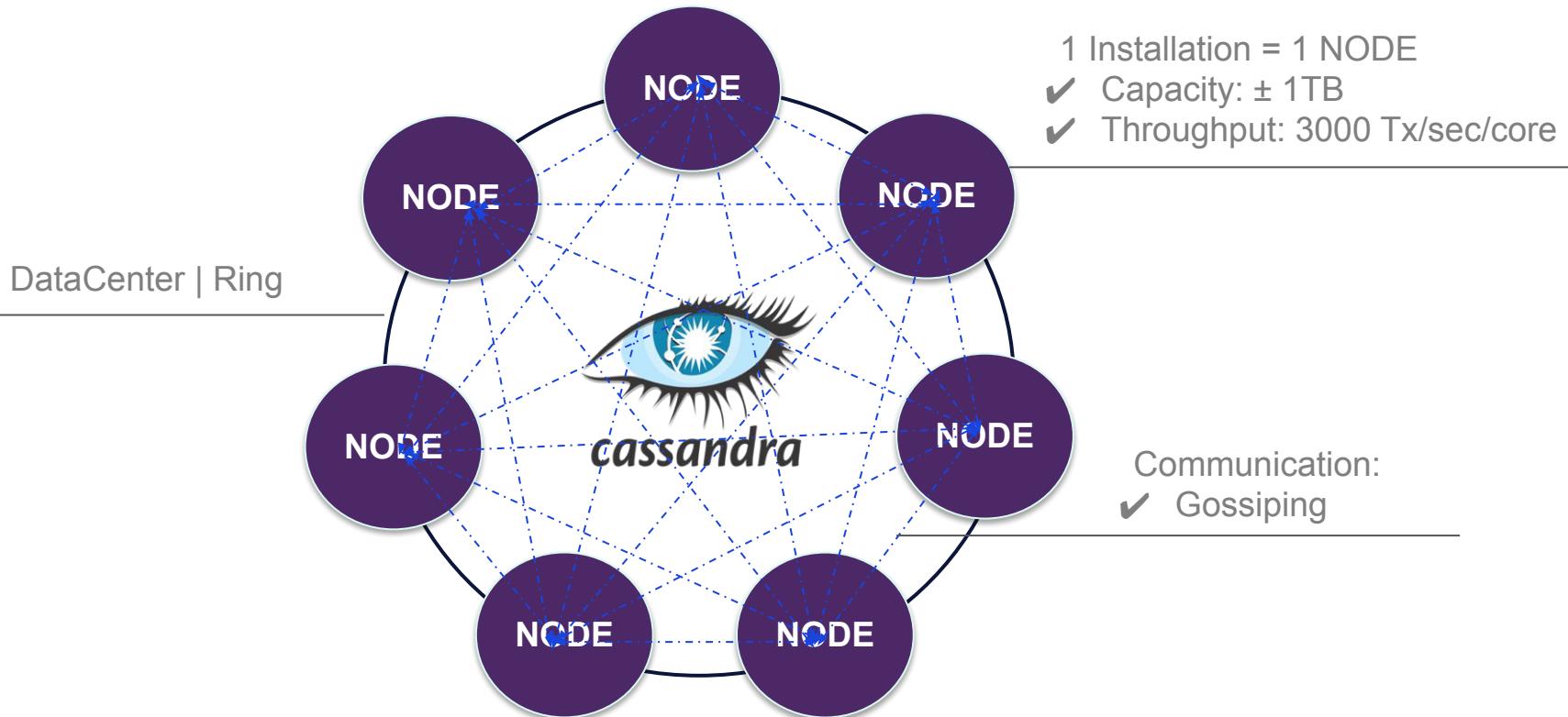


@DataStaxDevs #DataStaxDeveloperDay

<https://community.datastax.com>



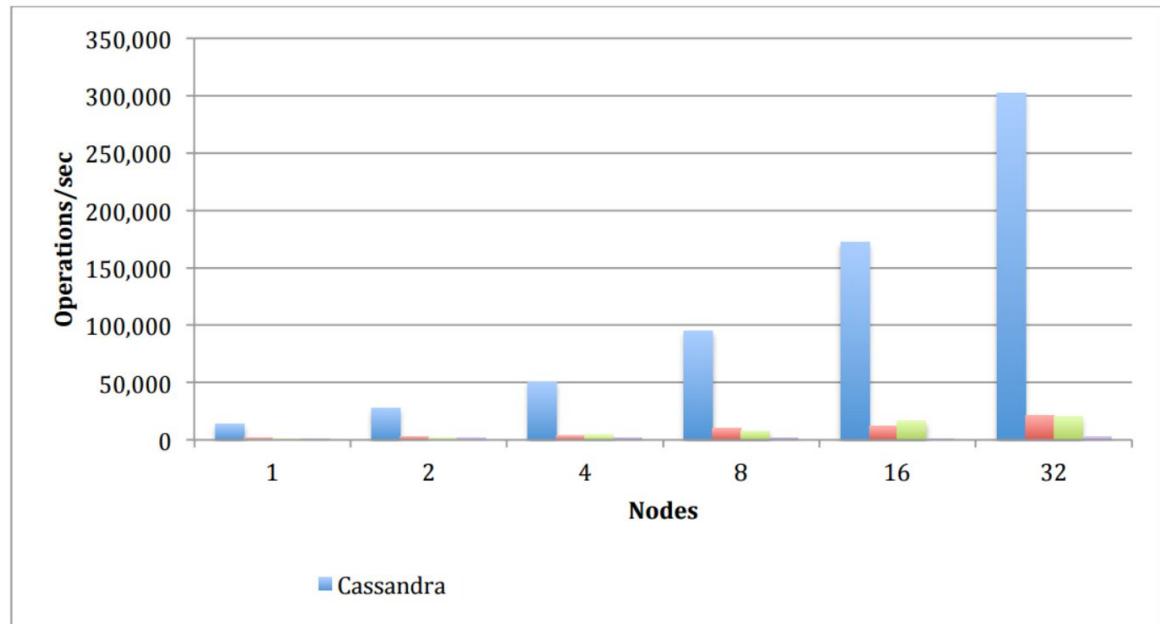
# Apache Cassandra™ = NoSQL Distributed Database



# Scales Linearly

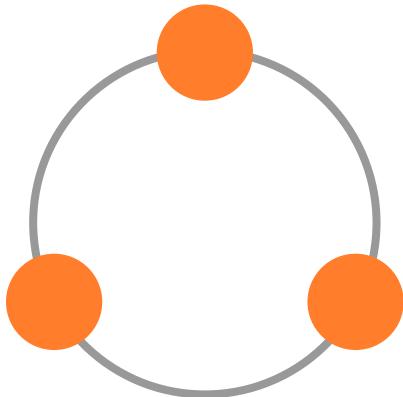
- Need more capacity?
- Need more throughput?
- Add nodes!

Balanced Read/Write Mix

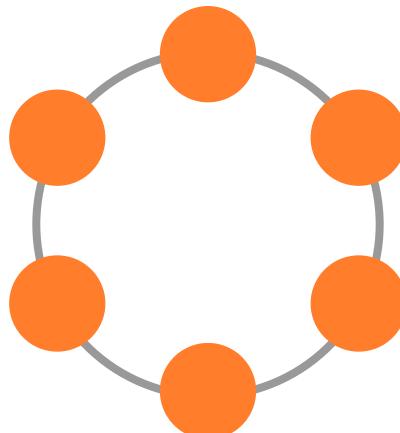


# Horizontal vs. Vertical Scaling

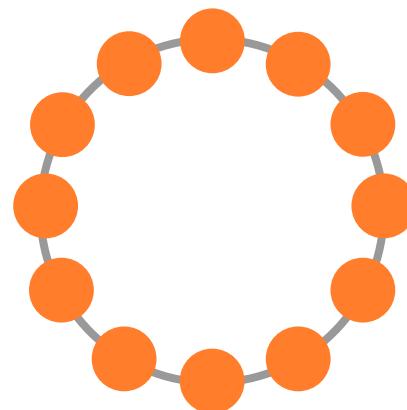
- Vertical scaling requires one large expensive machine
- Horizontal scaling requires multiple less-expensive commodity hardware



100,000 transactions/second



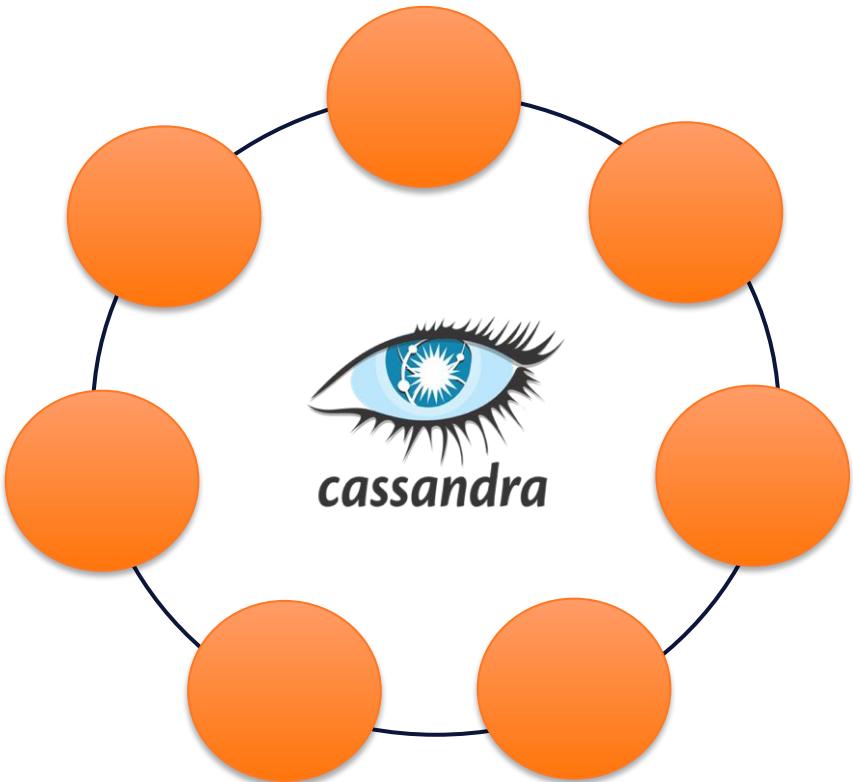
200,000 transactions/second



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

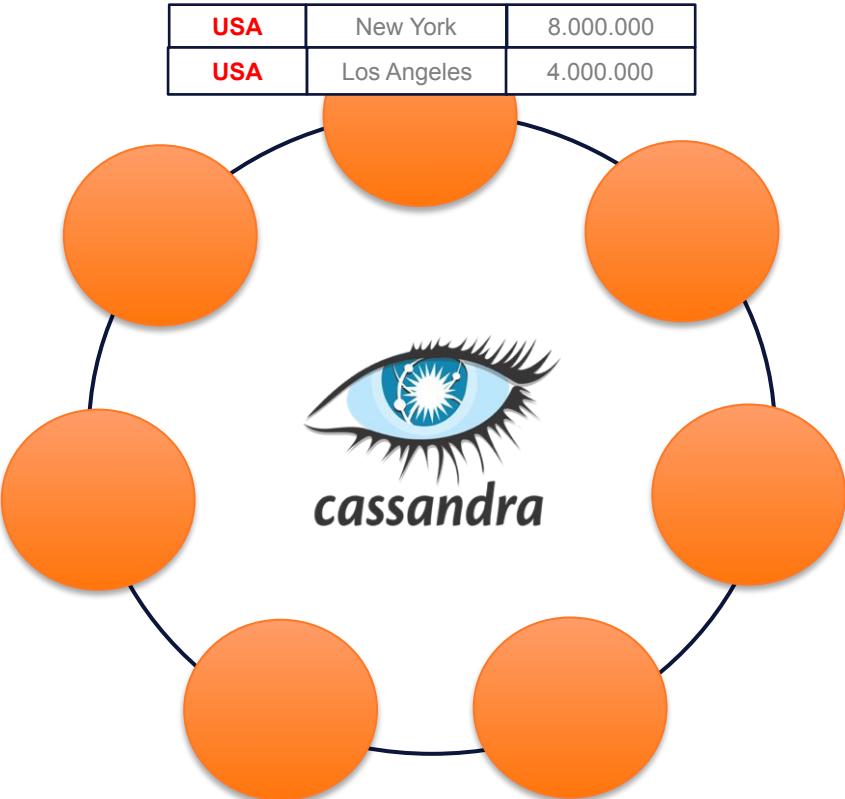


@DataStaxDevs #DataStaxDeveloperDay

<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

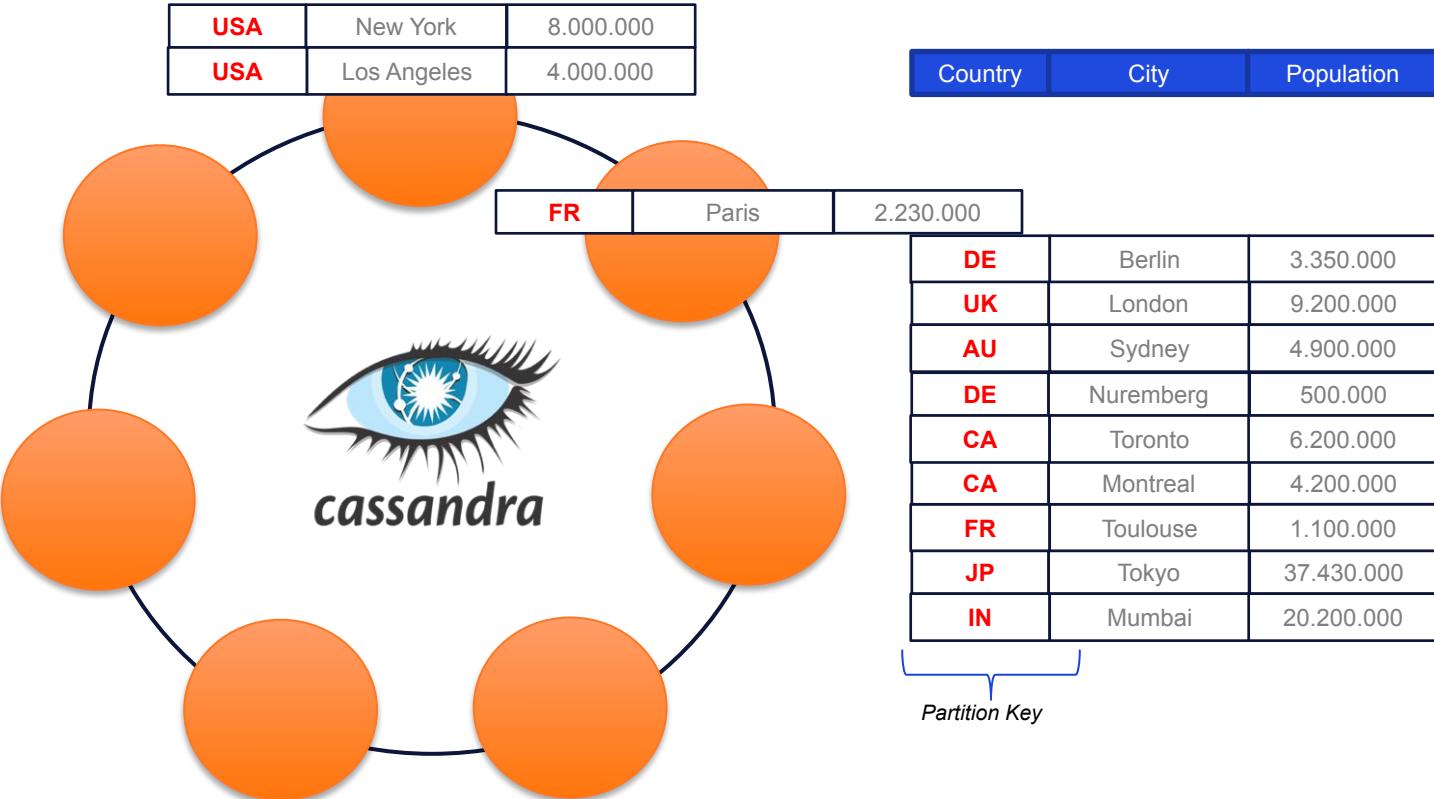


@DataStaxDevs #DataStaxDeveloperDay

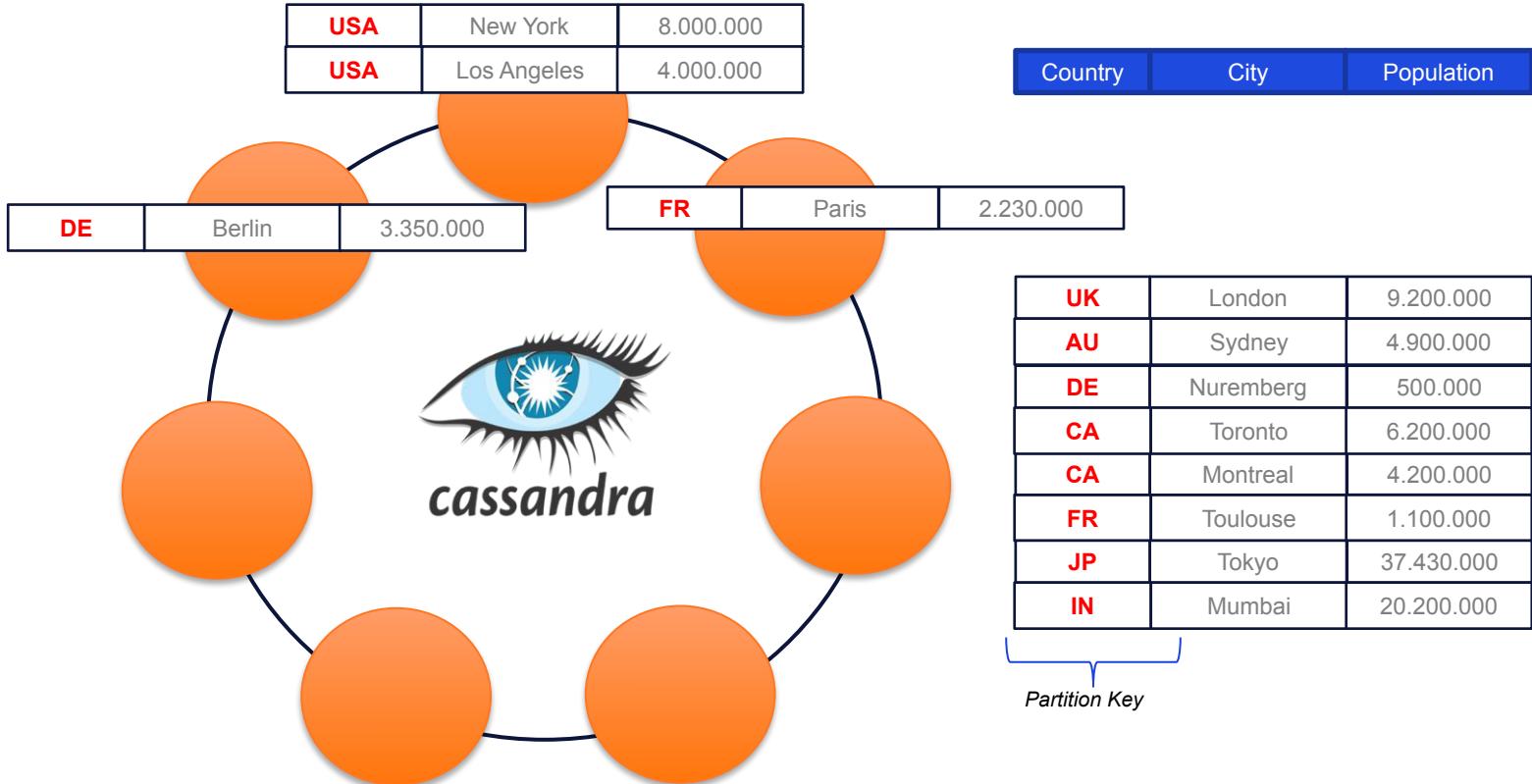
<https://community.datastax.com>



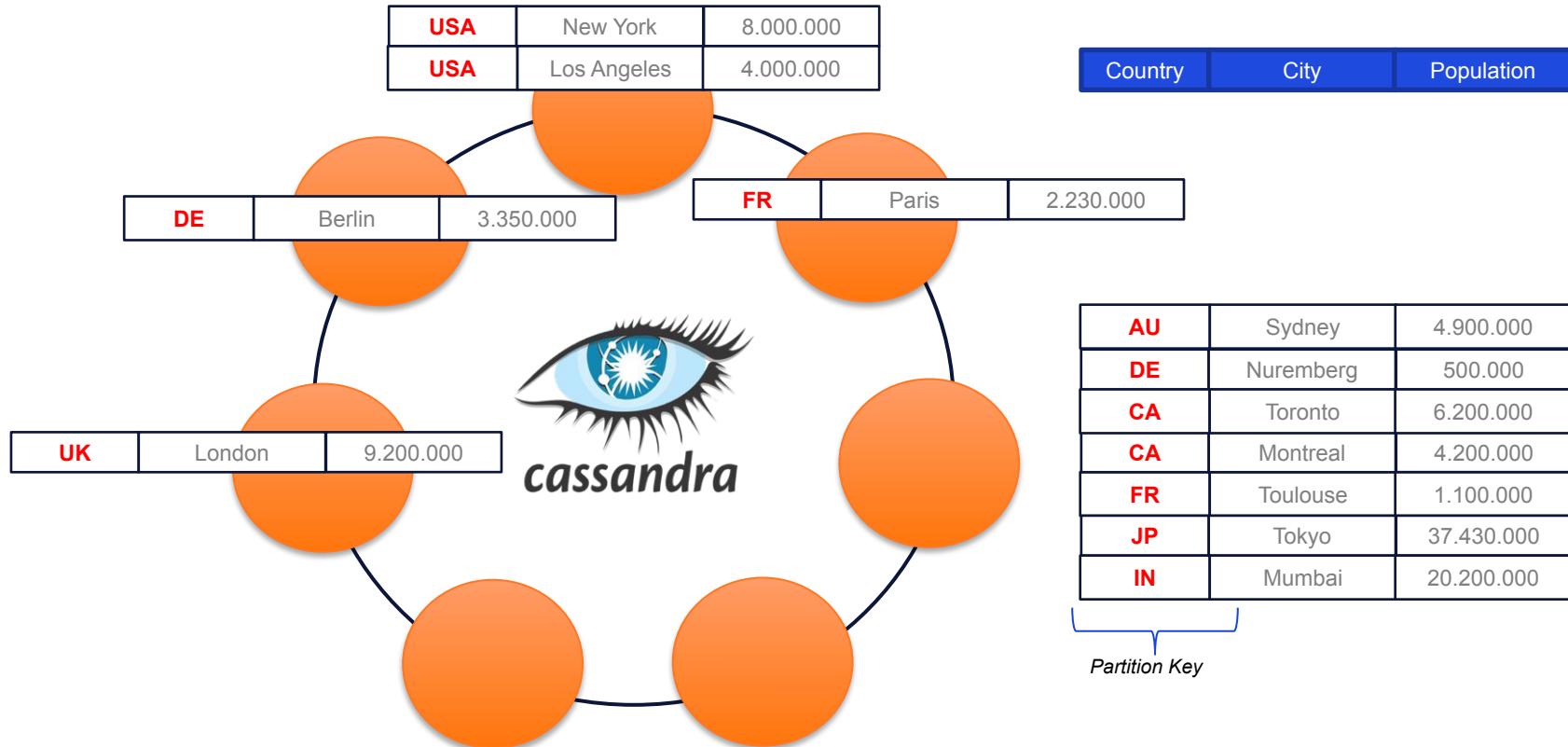
# Data is Distributed



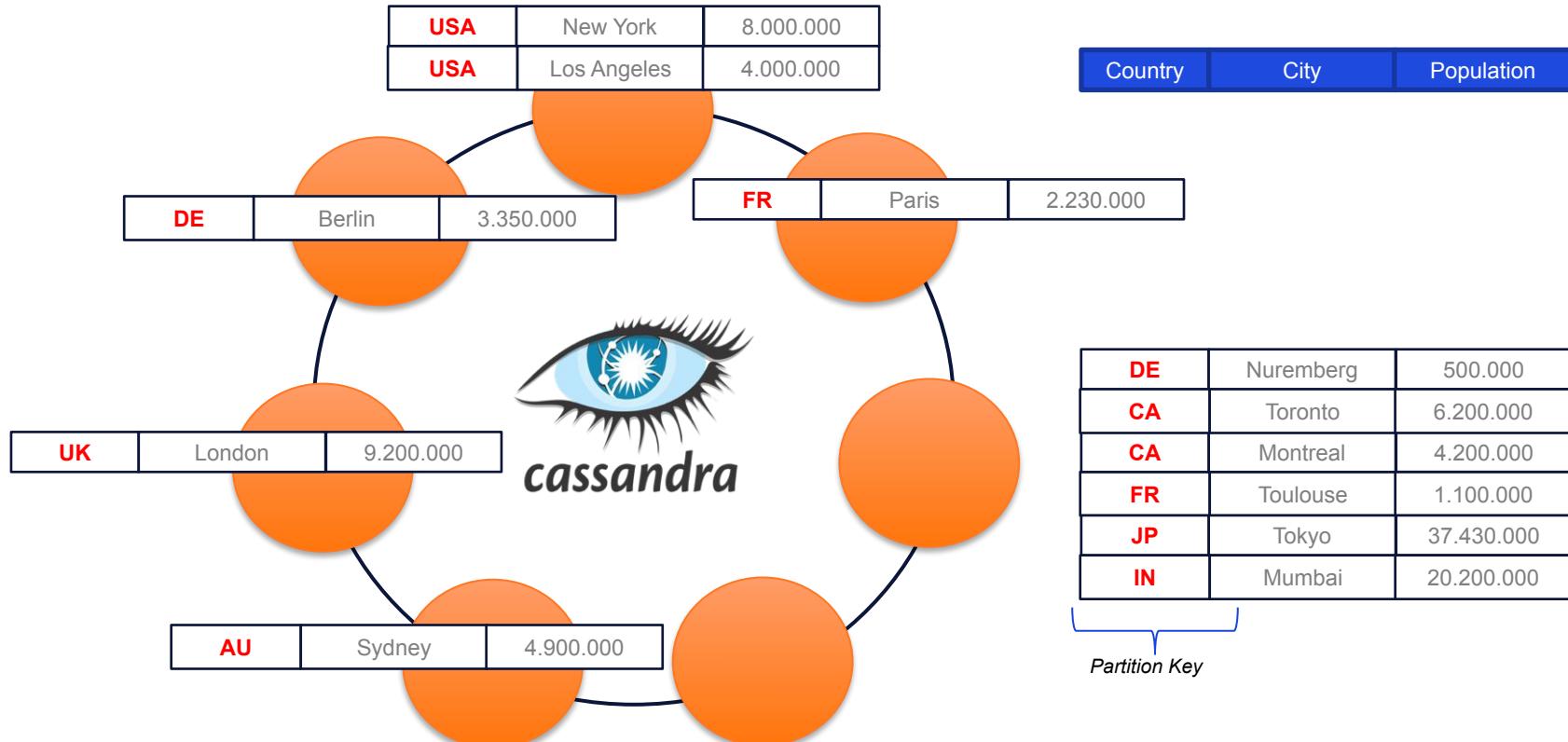
# Data is Distributed



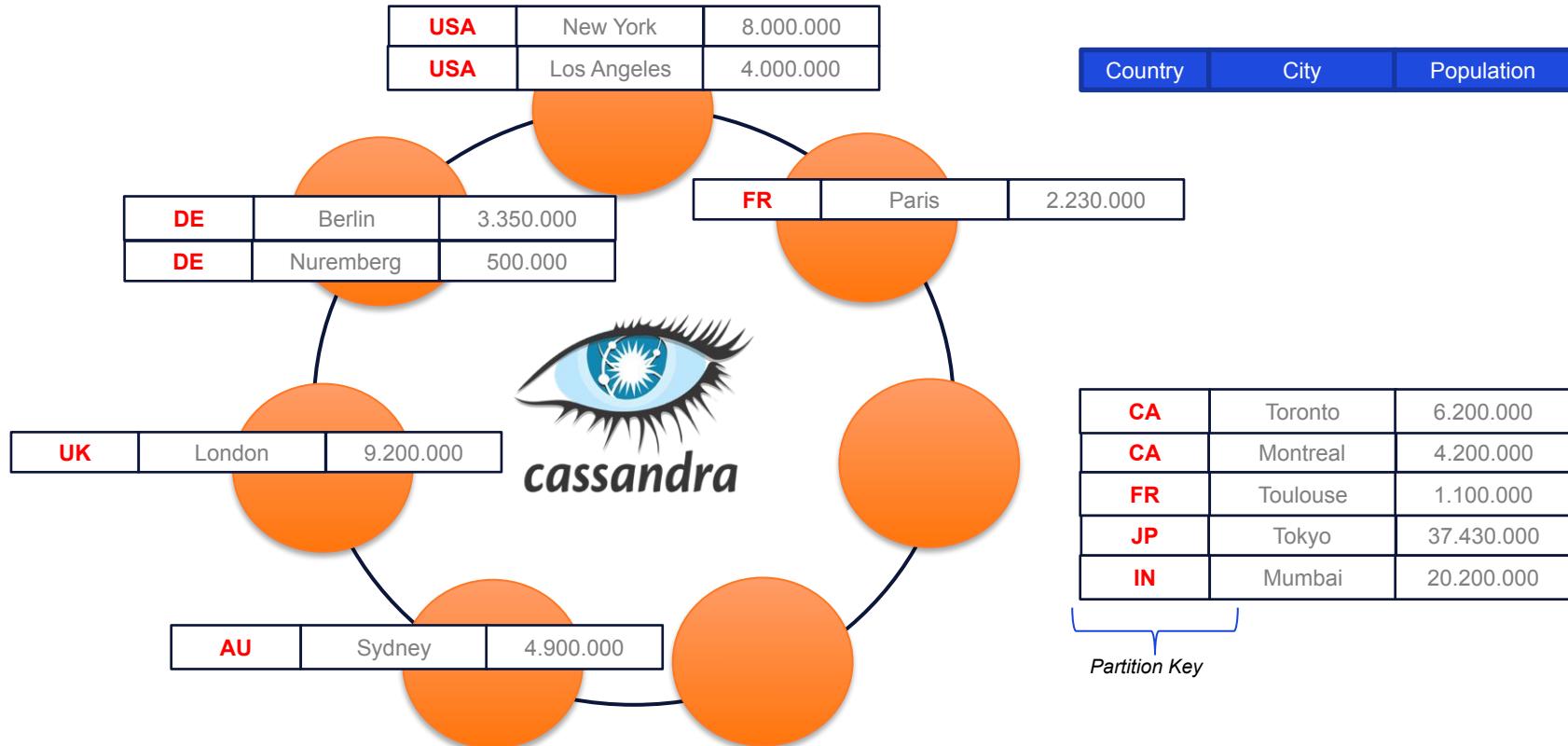
# Data is Distributed



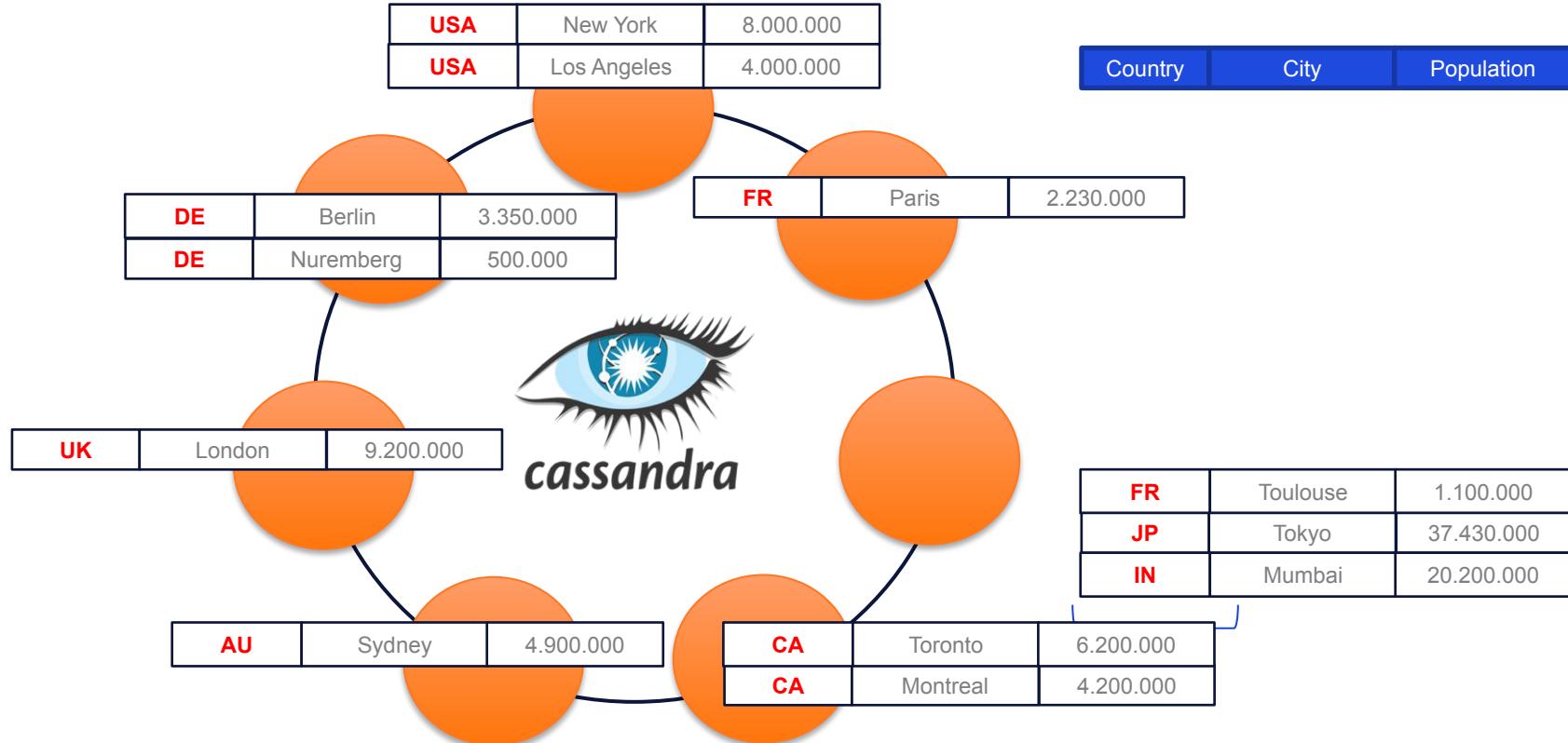
# Data is Distributed



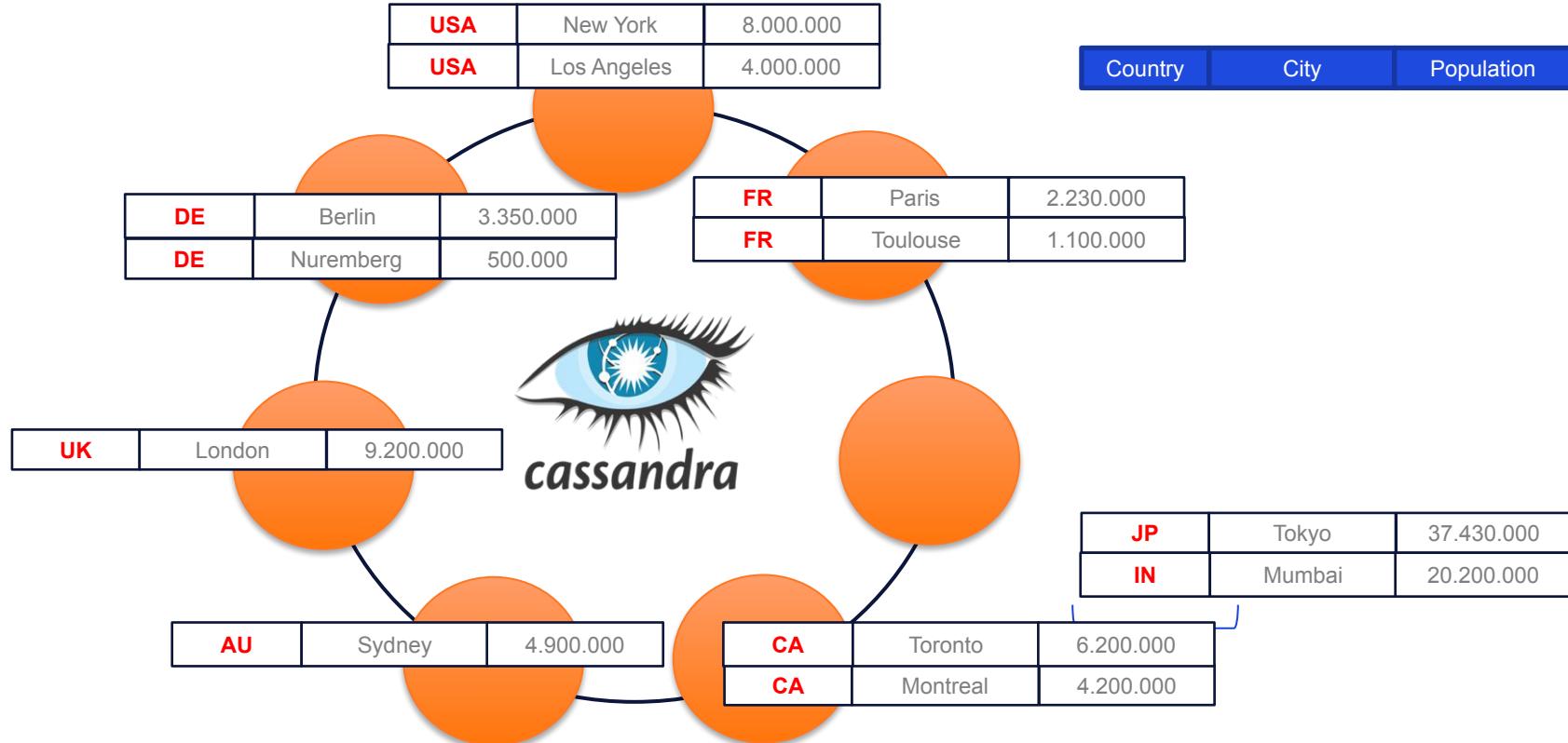
# Data is Distributed



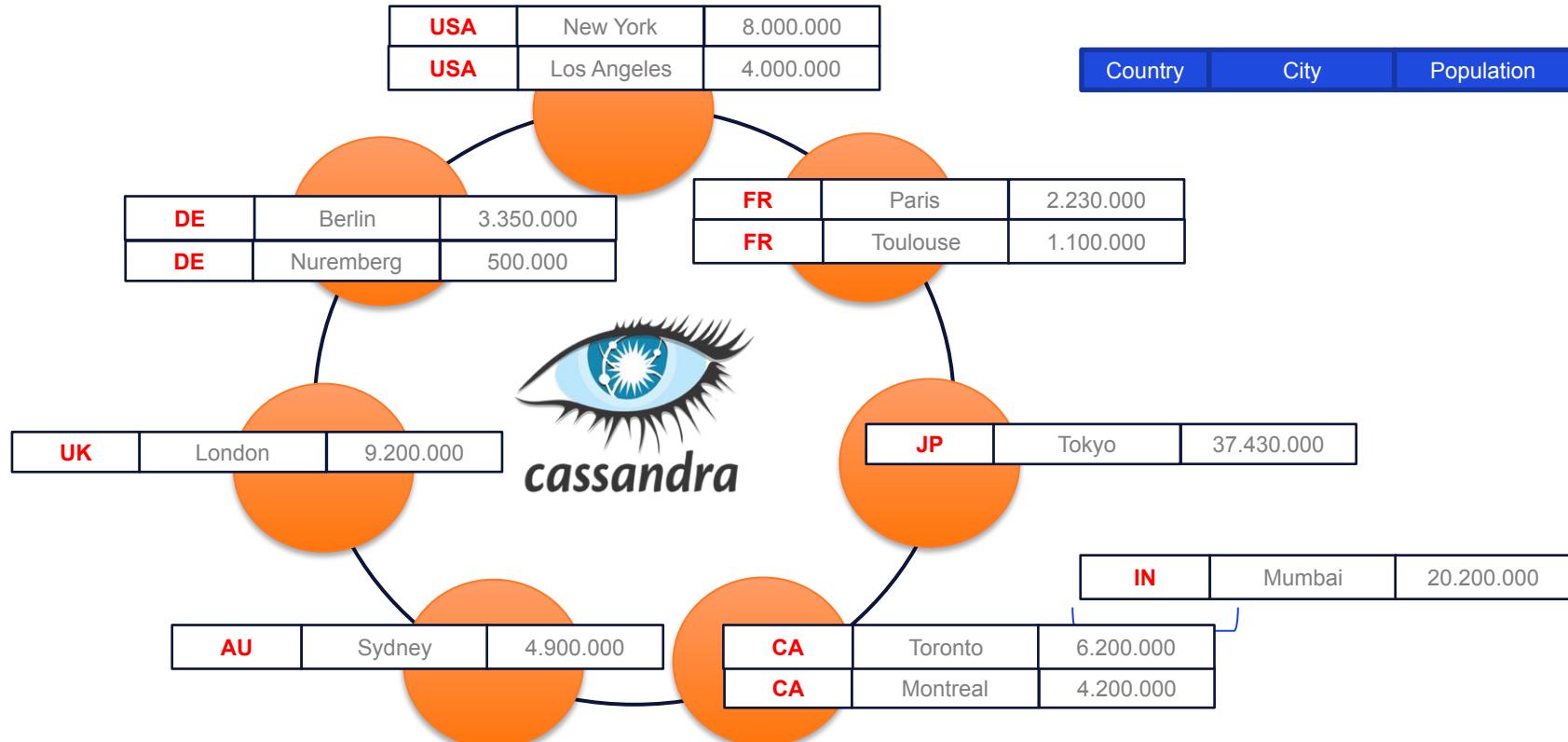
# Data is Distributed



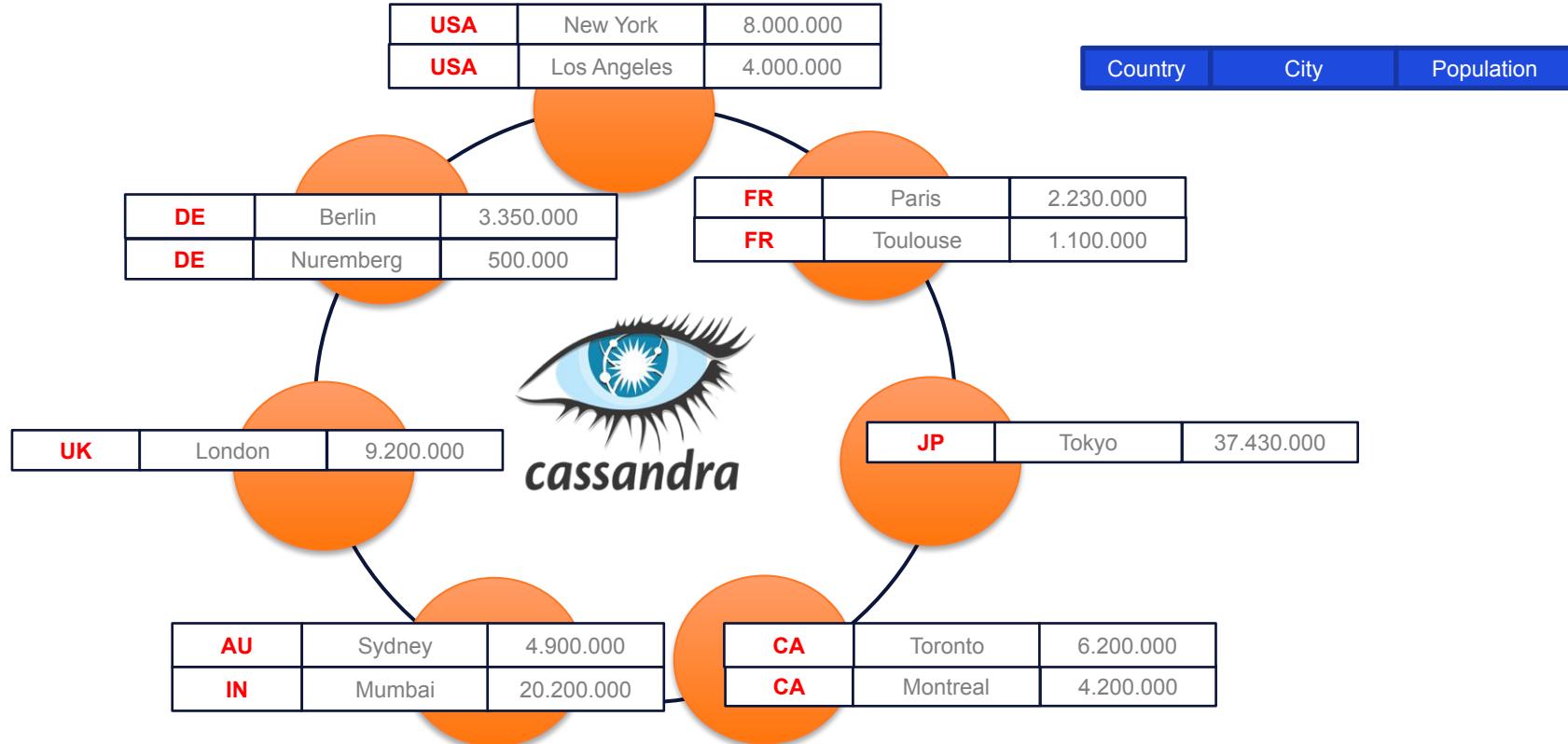
# Data is Distributed



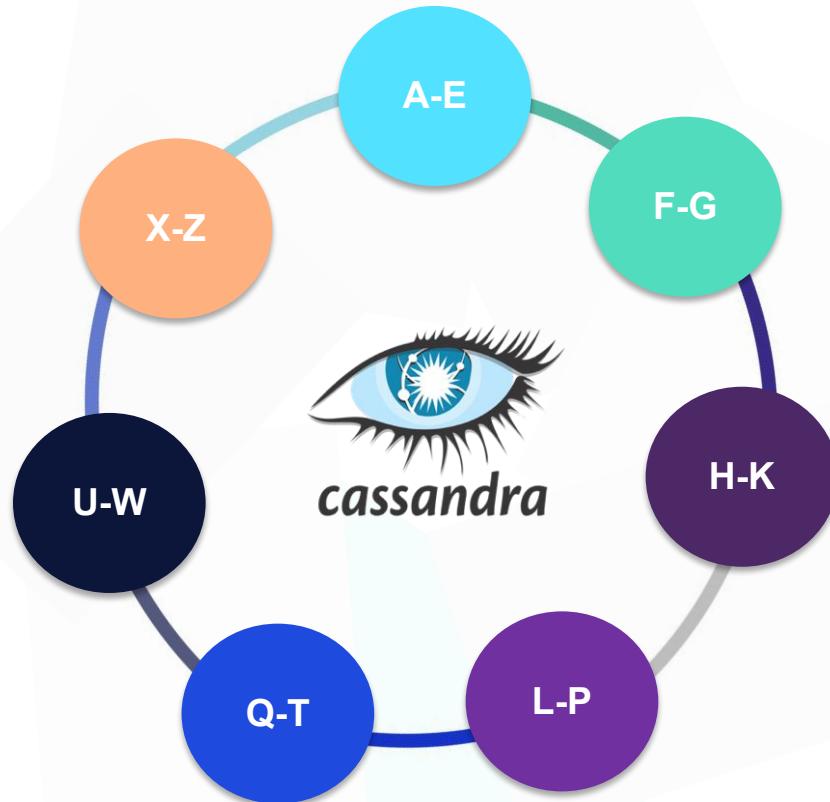
# Data is Distributed



# Data is Distributed



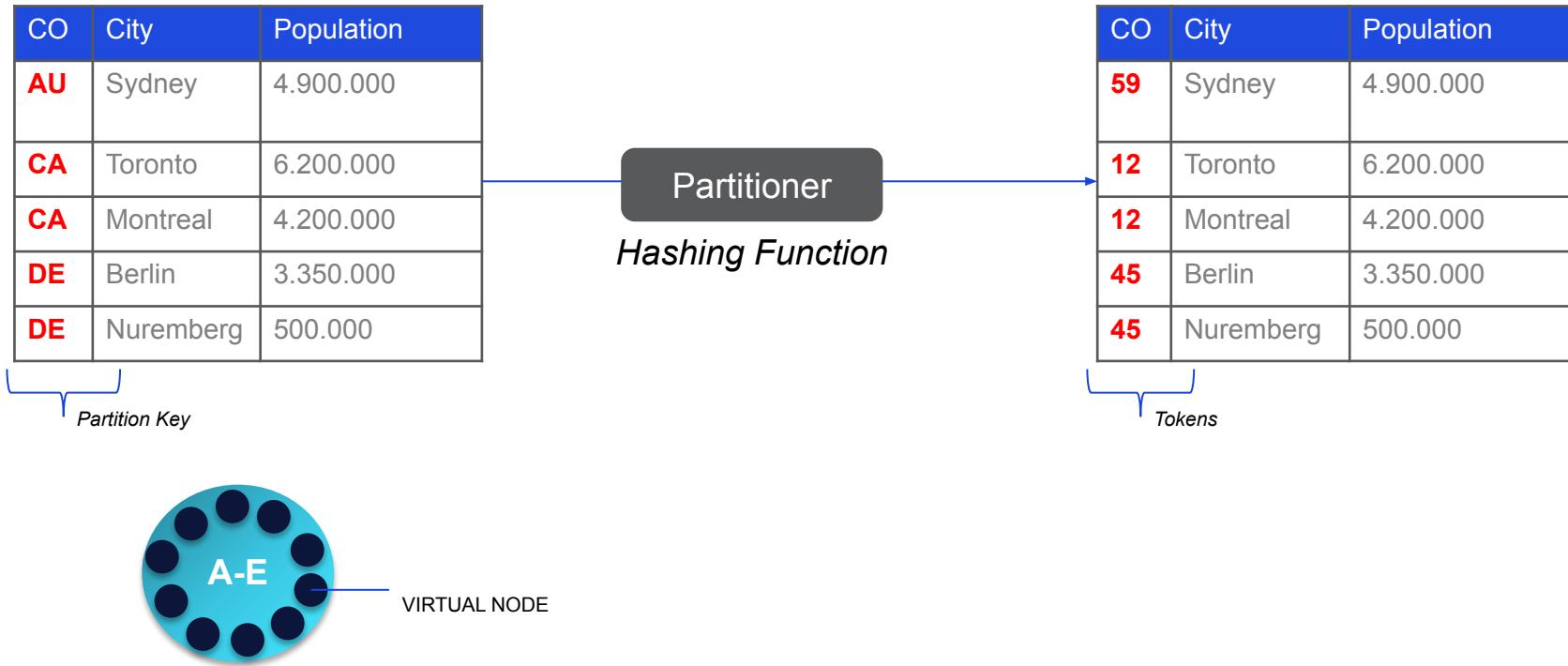
# Data is Distributed



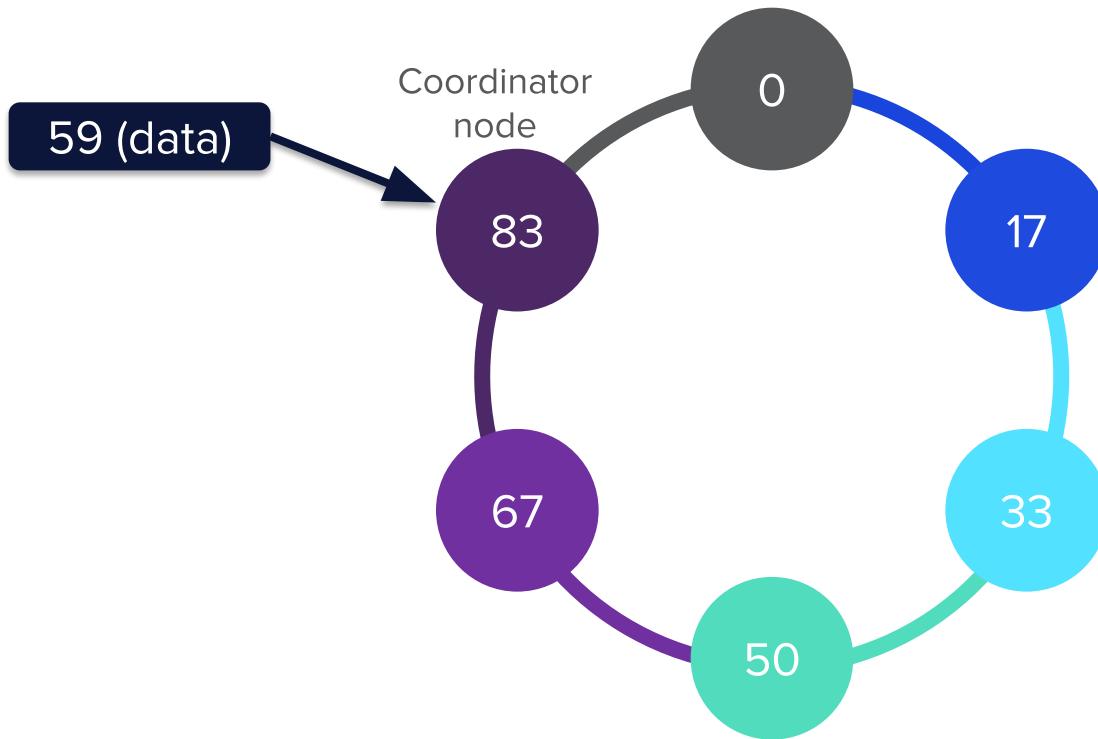
@DataStaxDevs #DataStaxDevelop



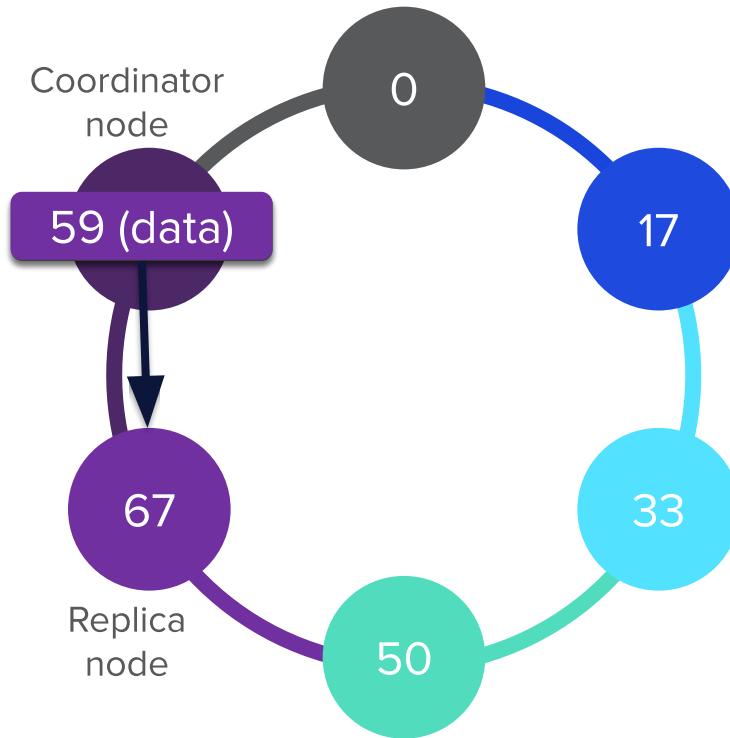
# Data is *Evenly-distributed*



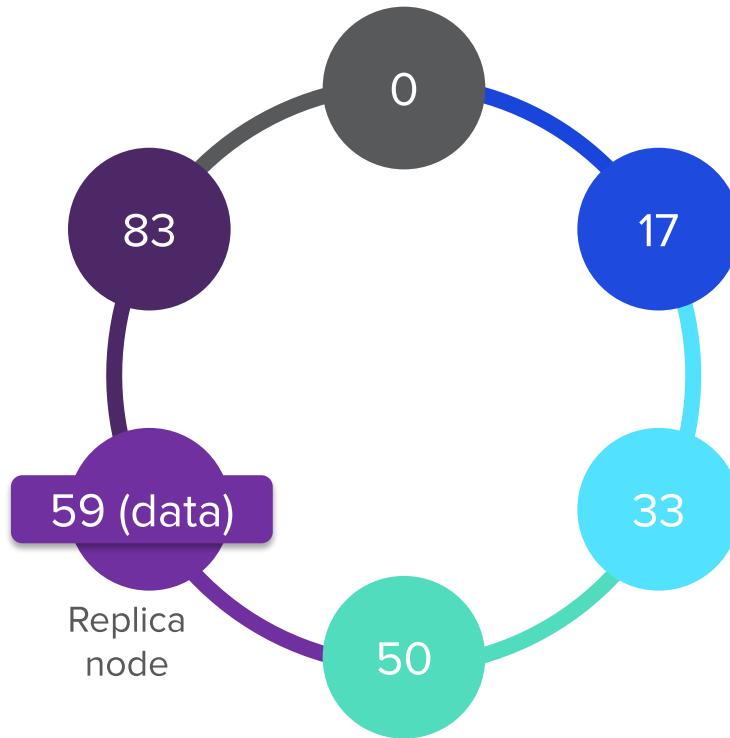
# How the Ring Works



# How the Ring Works

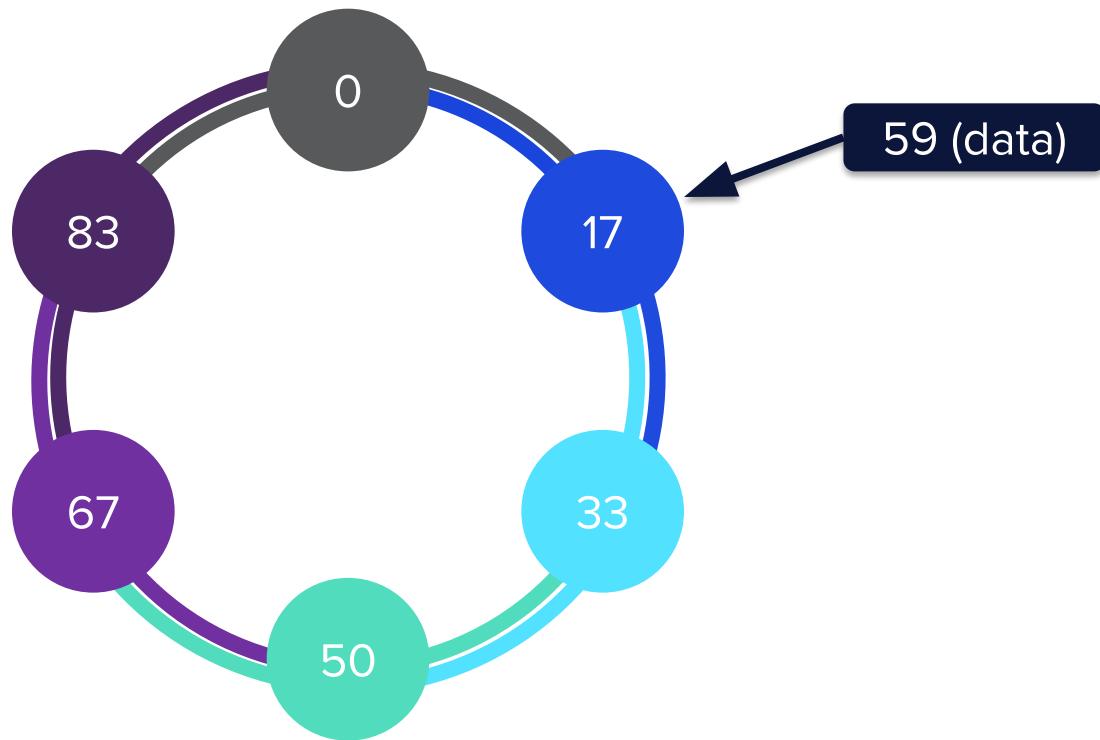


# How the Ring Works



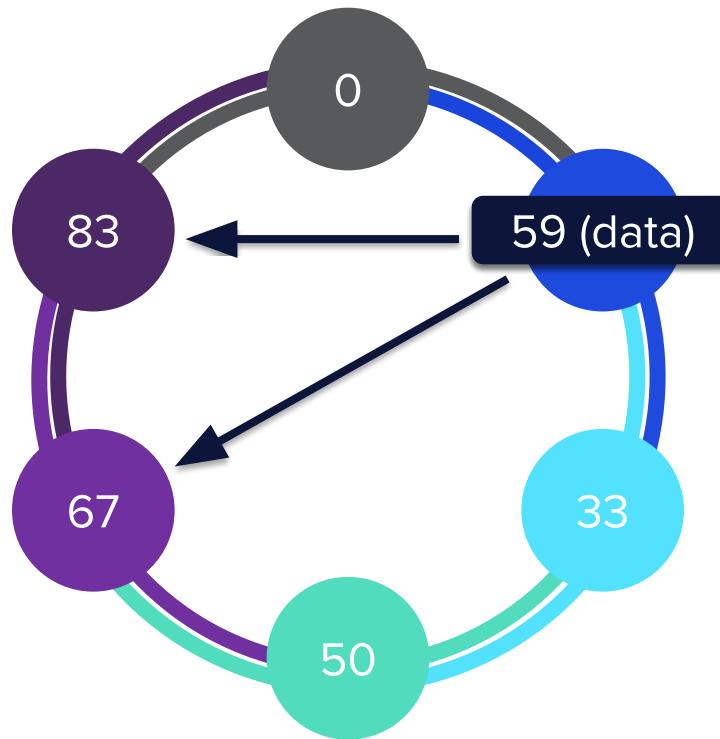
## Replication within the Ring

RF = 2



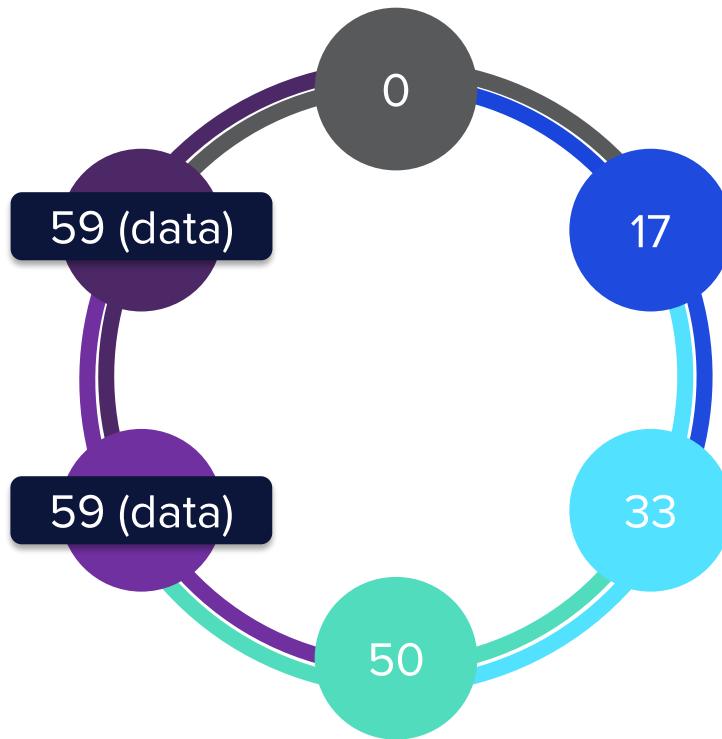
## Replication within the Ring

RF = 2



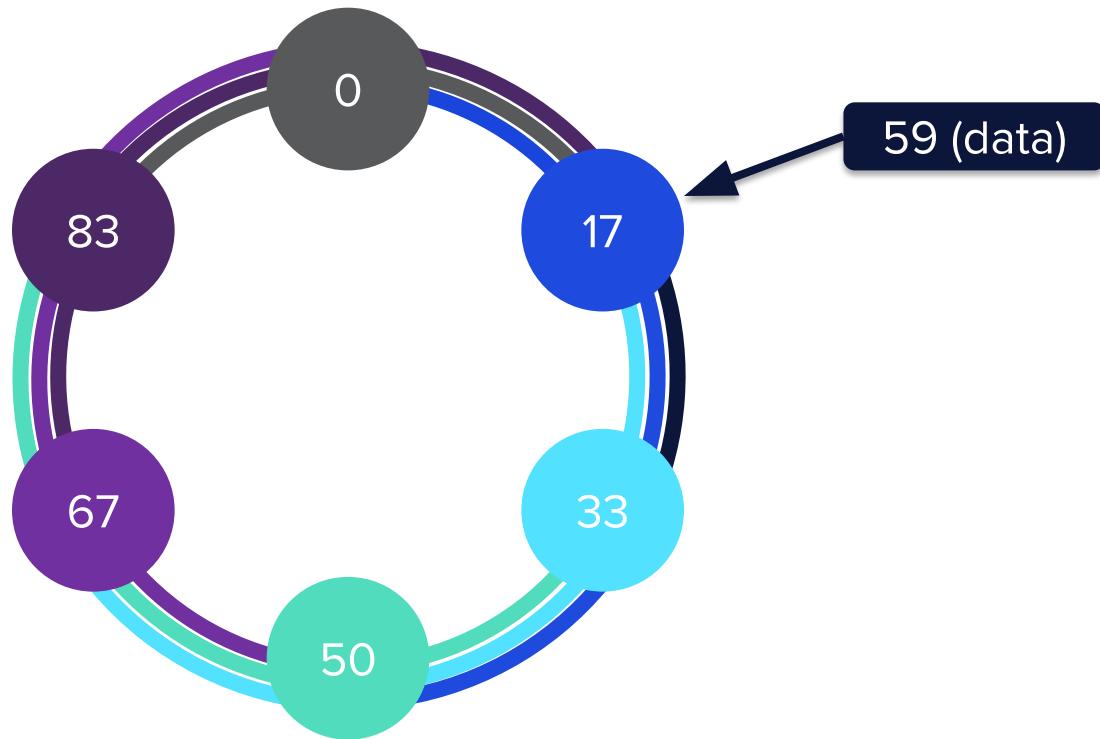
## Replication within the Ring

RF = 2



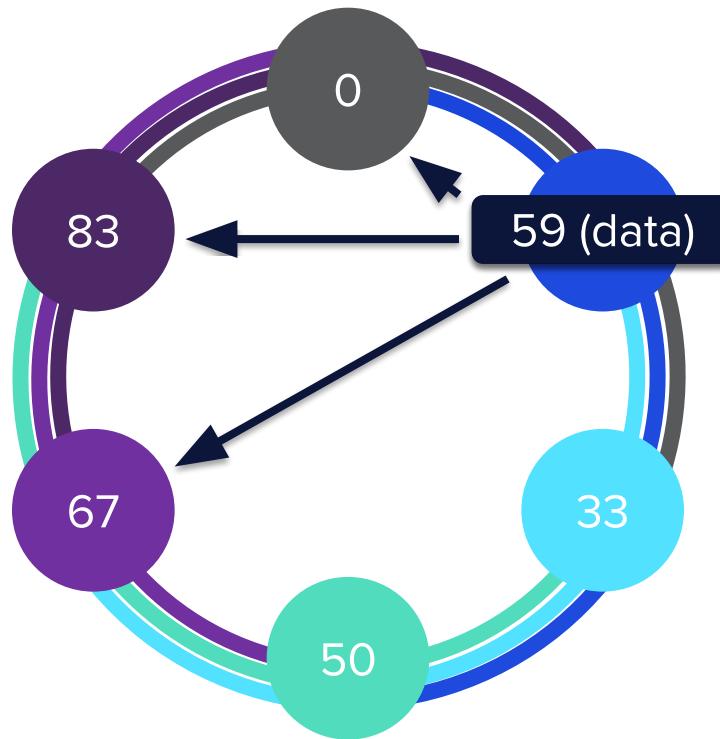
## Replication within the Ring

RF = 3



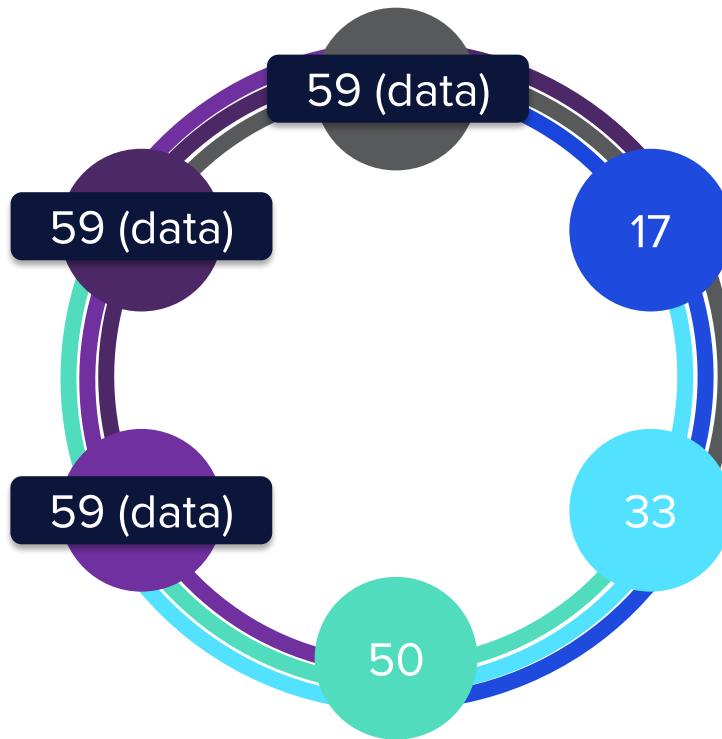
## Replication within the Ring

RF = 3



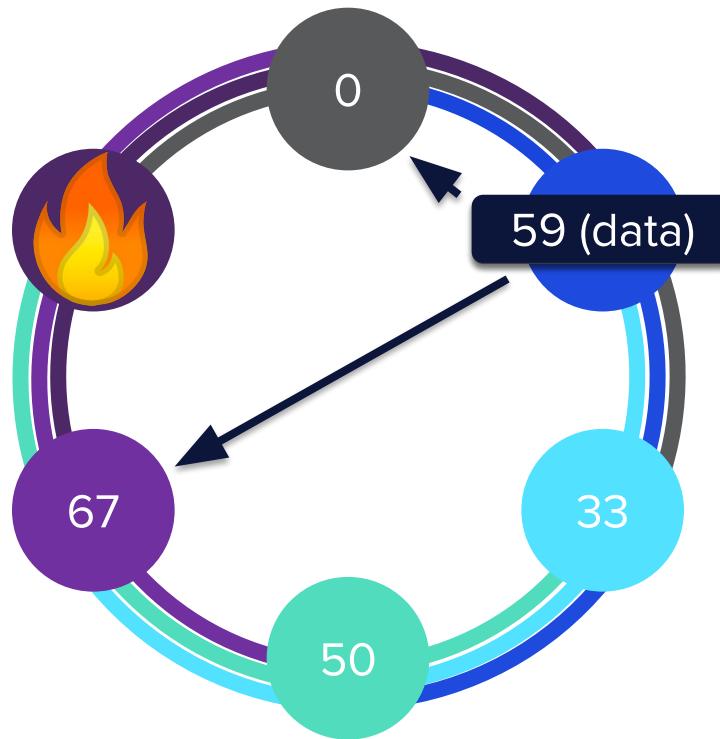
# Replication within the Ring

RF = 3



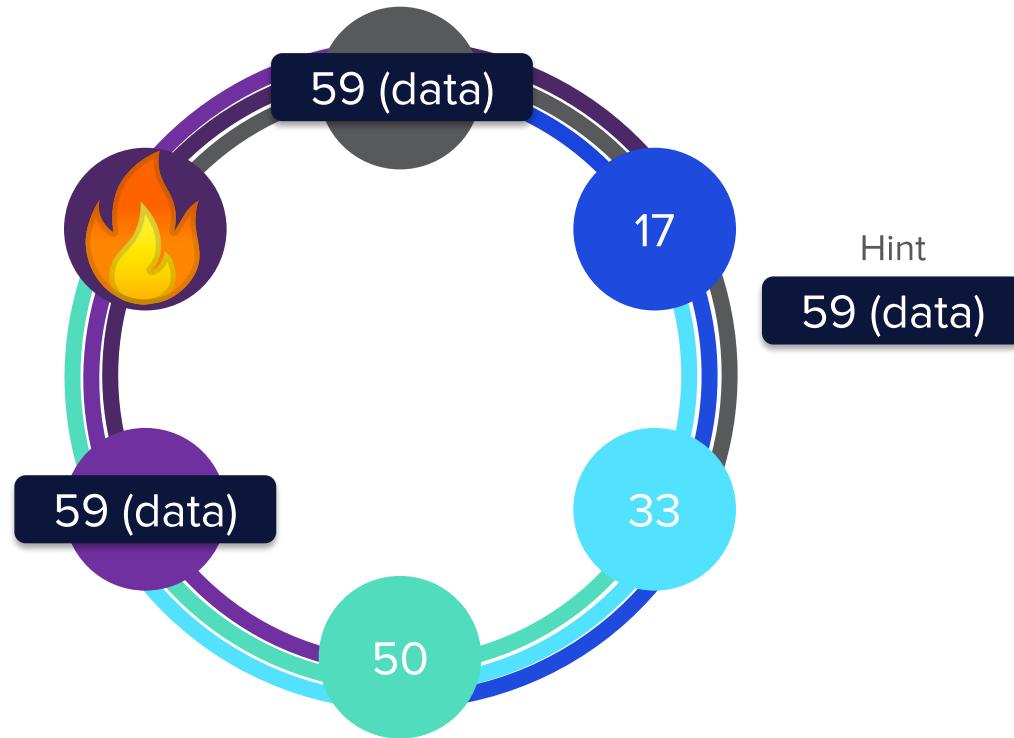
# Node Failure

RF = 3



# Node Failure

RF = 3



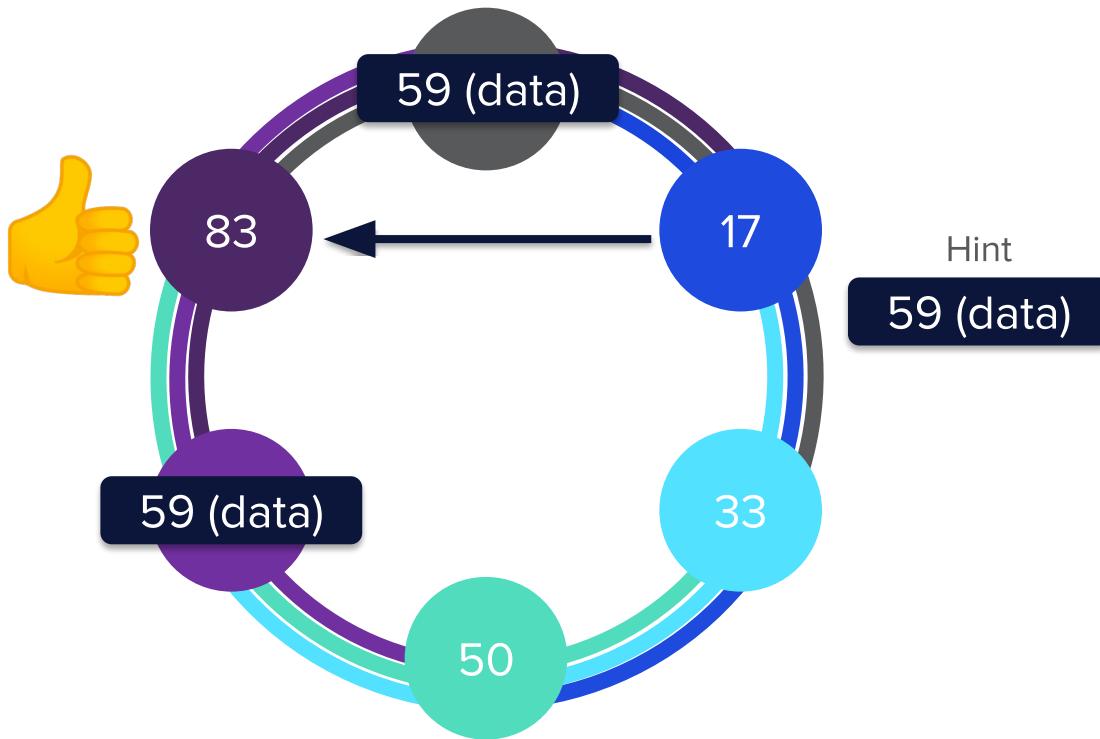
@DataStaxDevs #DataStaxDeveloperDay

<https://community.datastax.com>

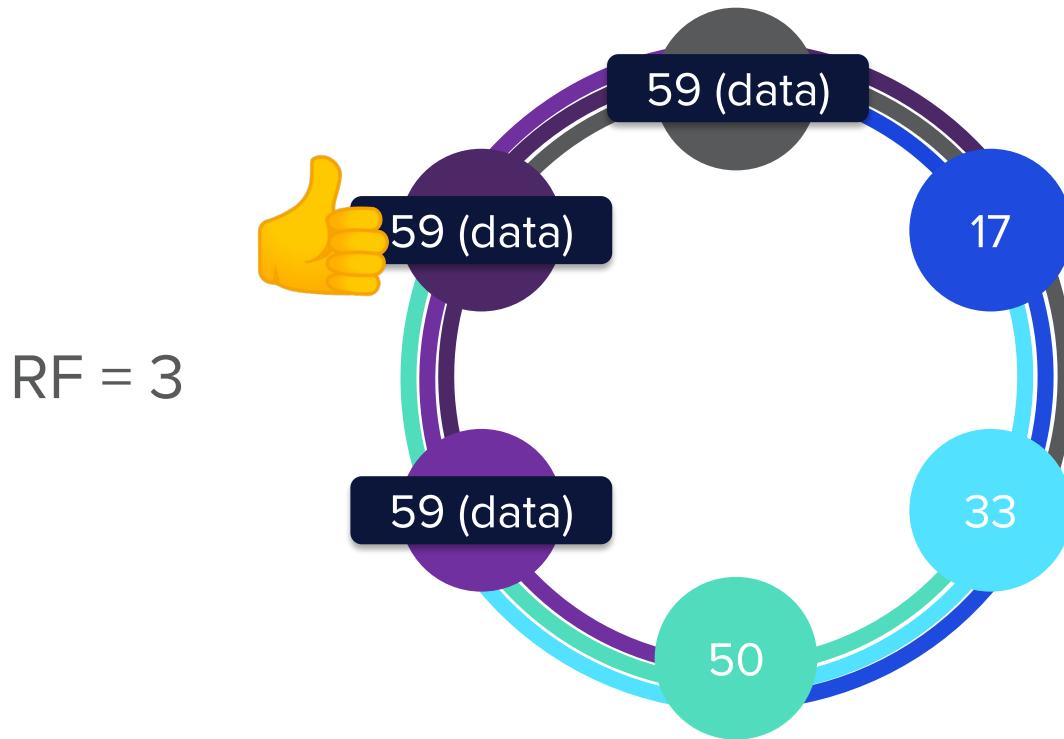


# Node Failure

RF = 3

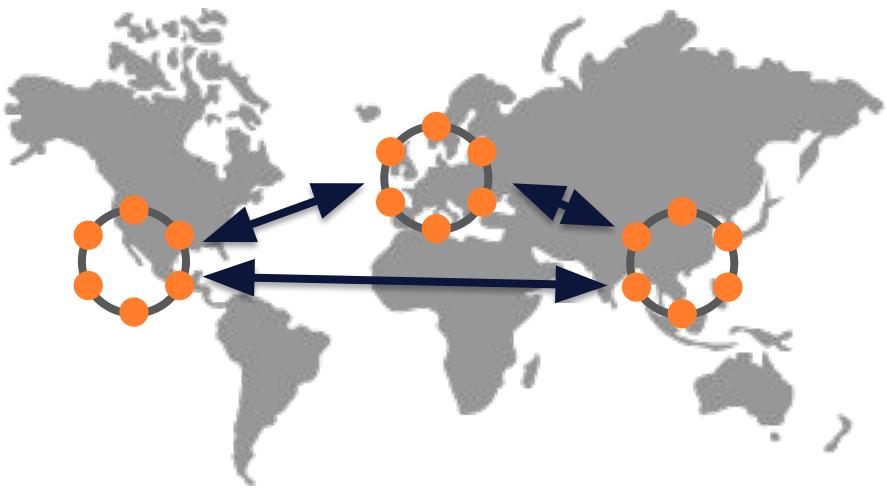


# Node Failure – Recovered!

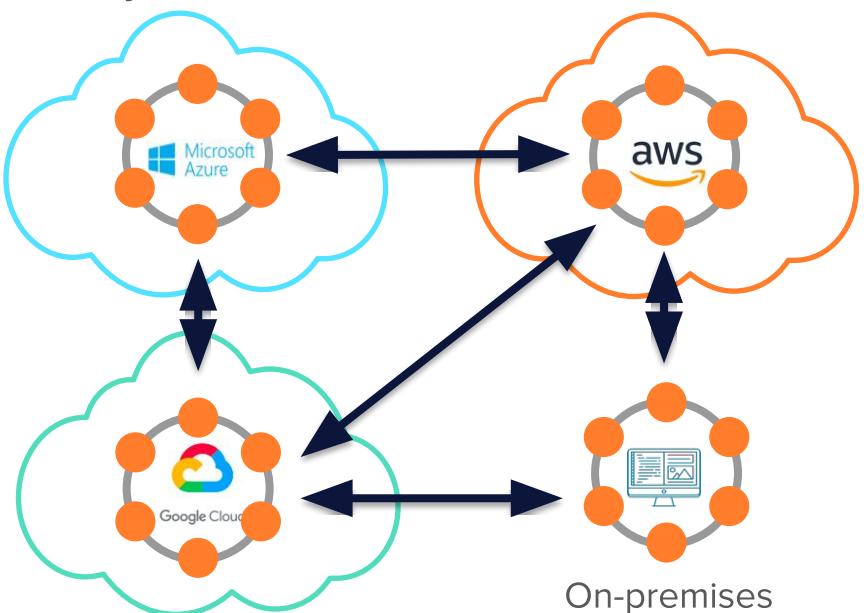


# Data Distributed Everywhere

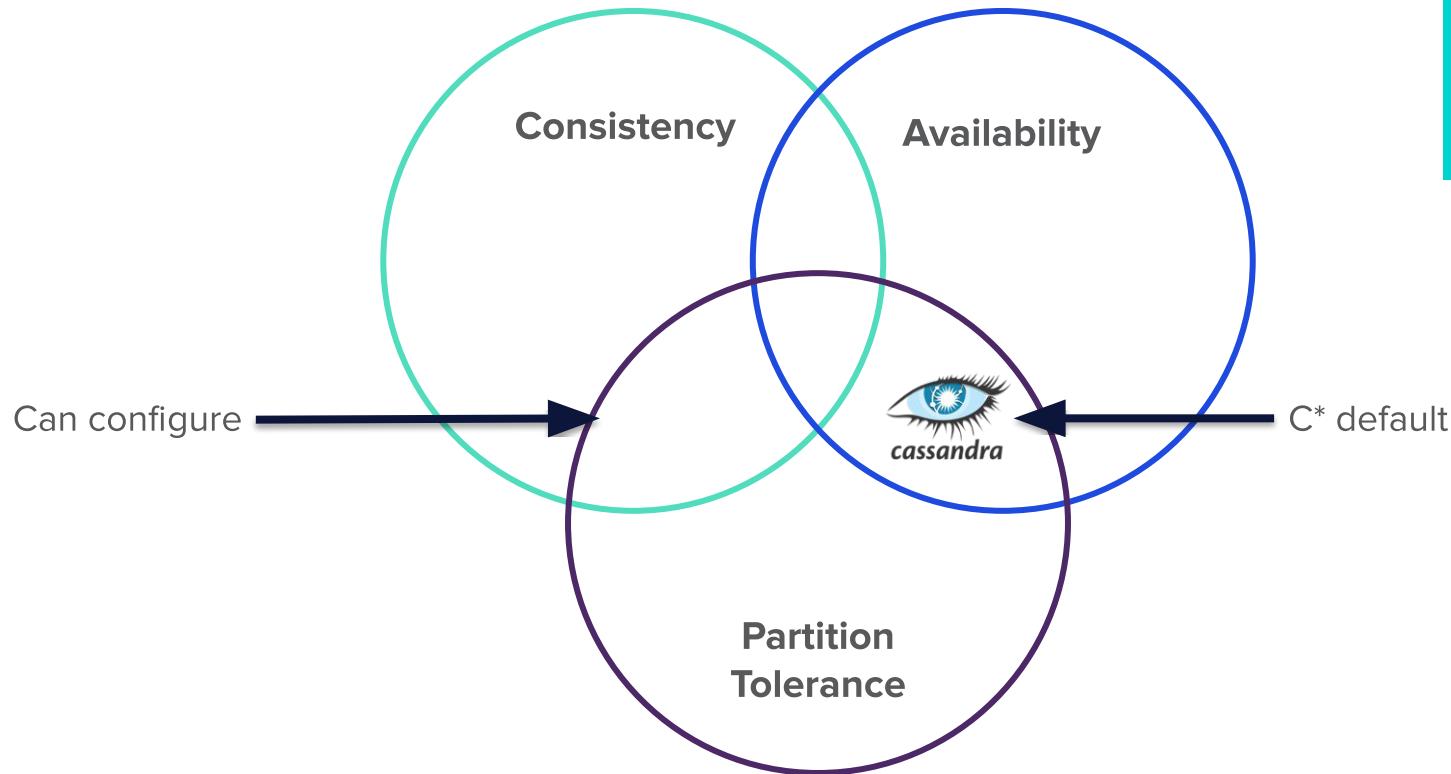
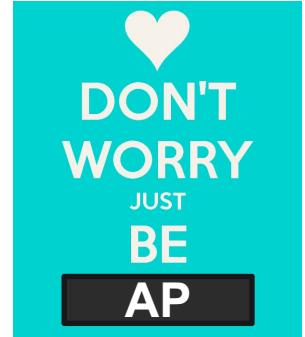
- Geographic Distribution



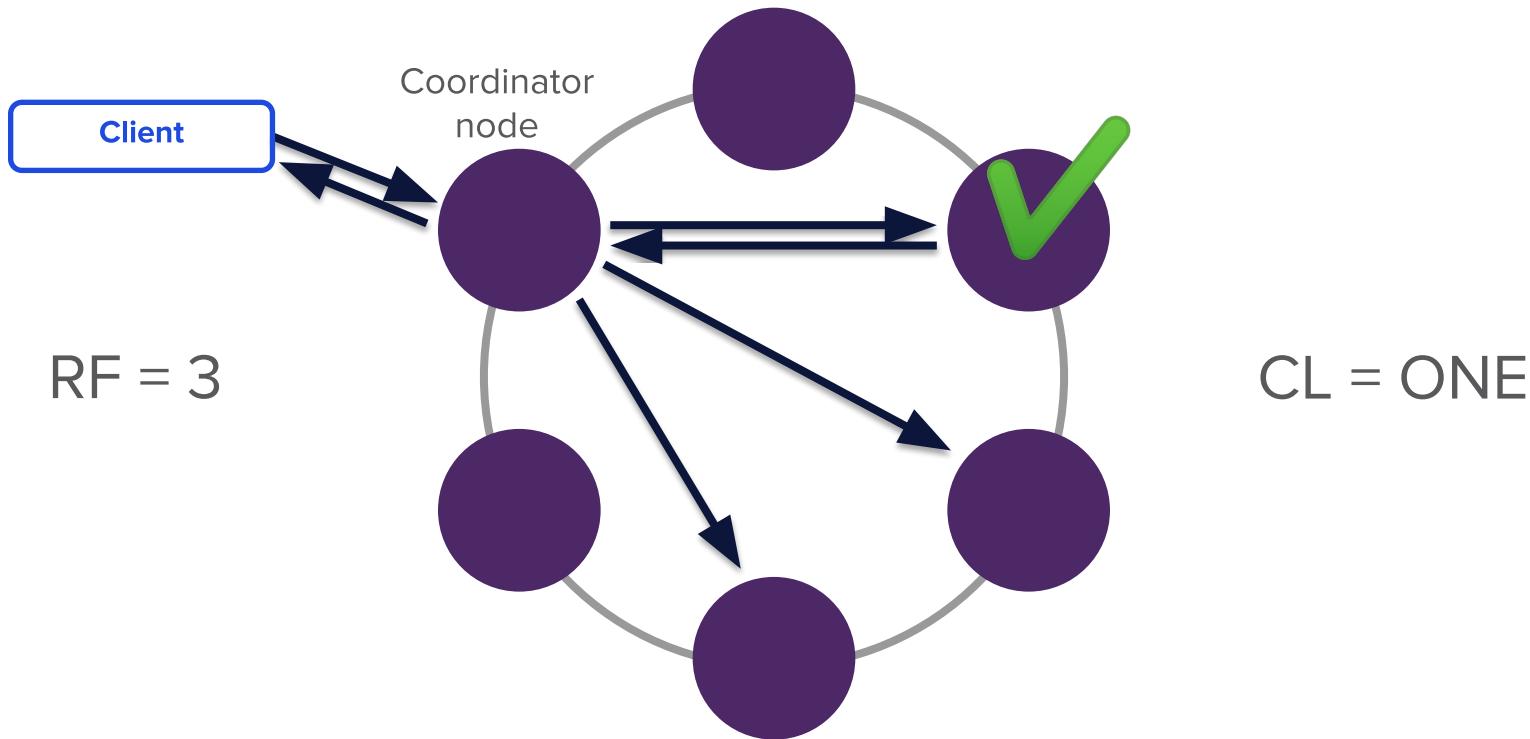
- Hybrid-Cloud and Multi-Cloud



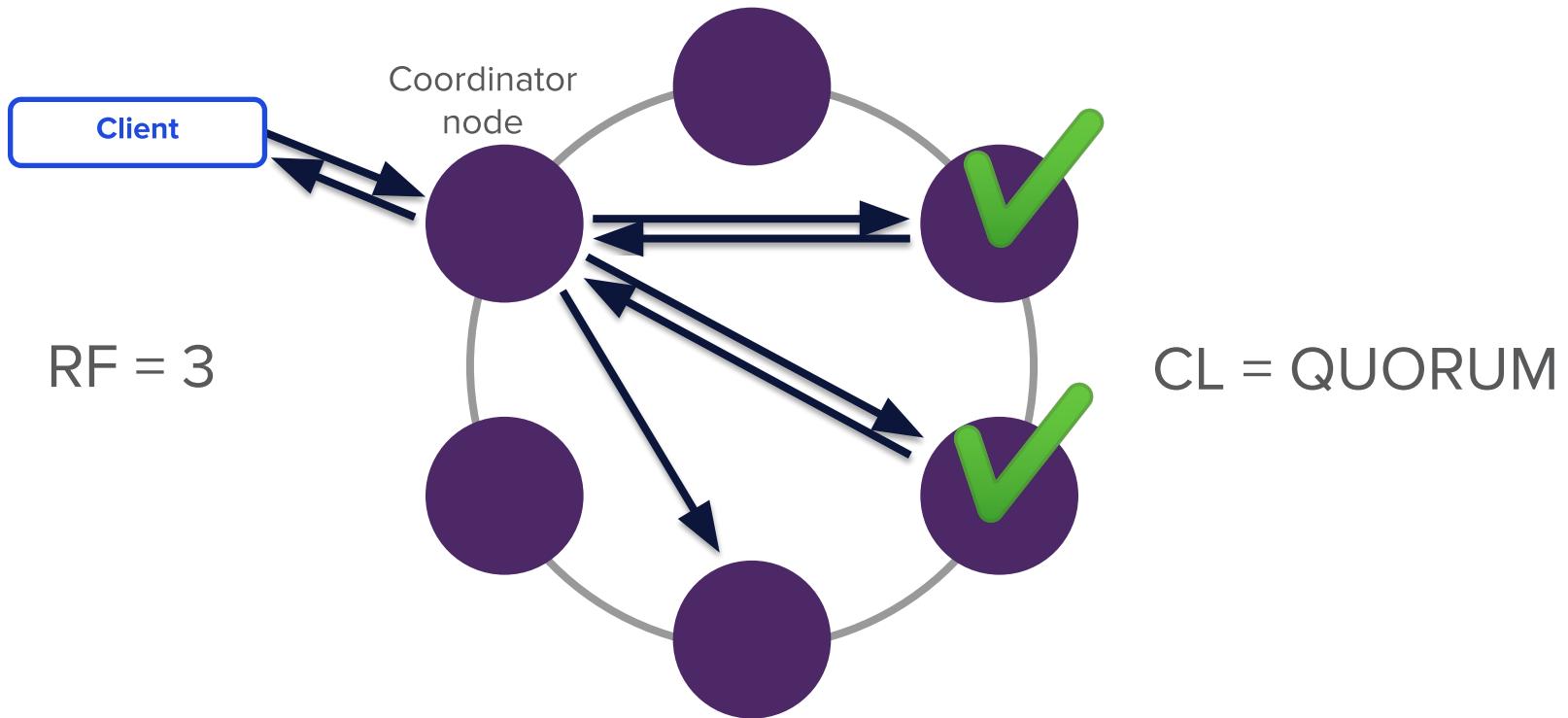
# Cap Theorem



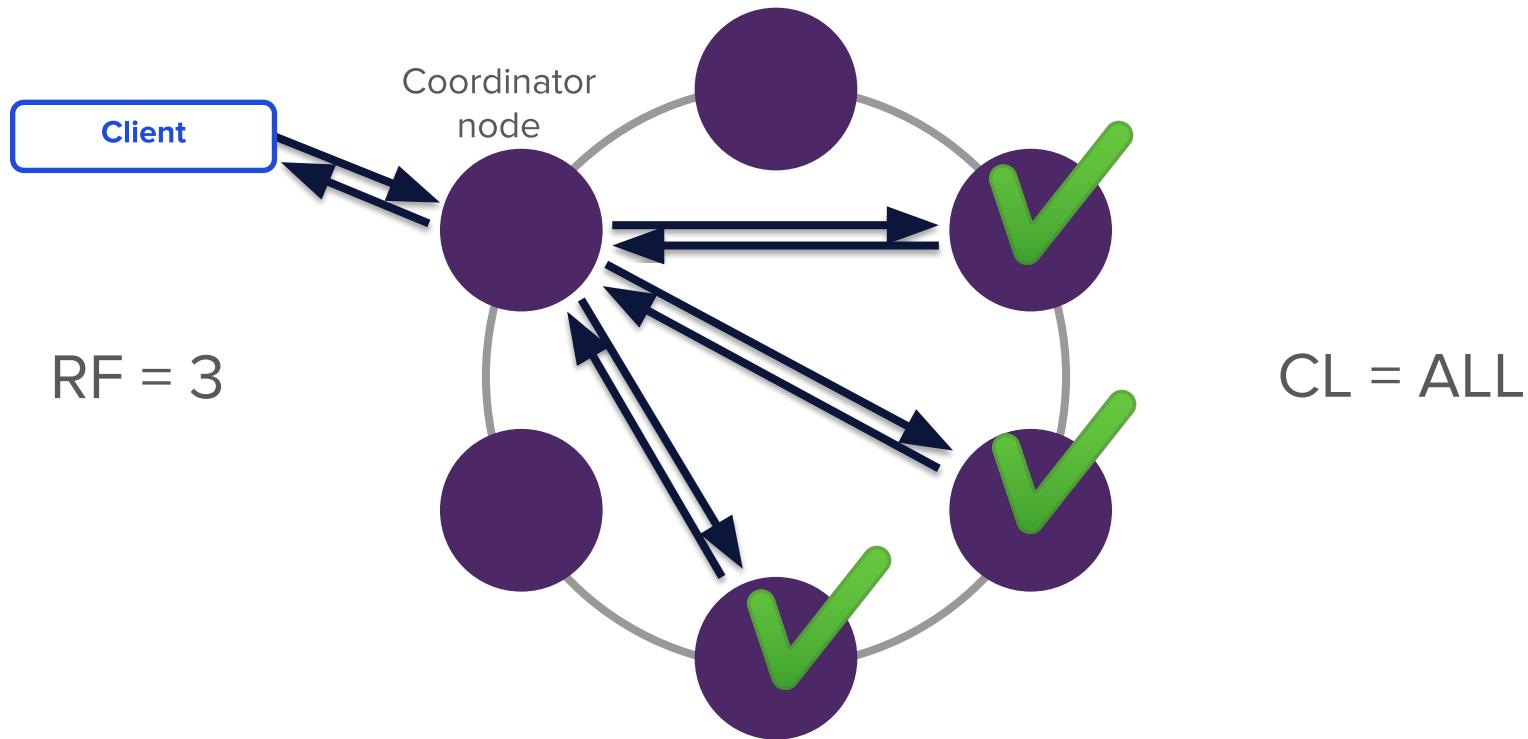
# Consistency Levels



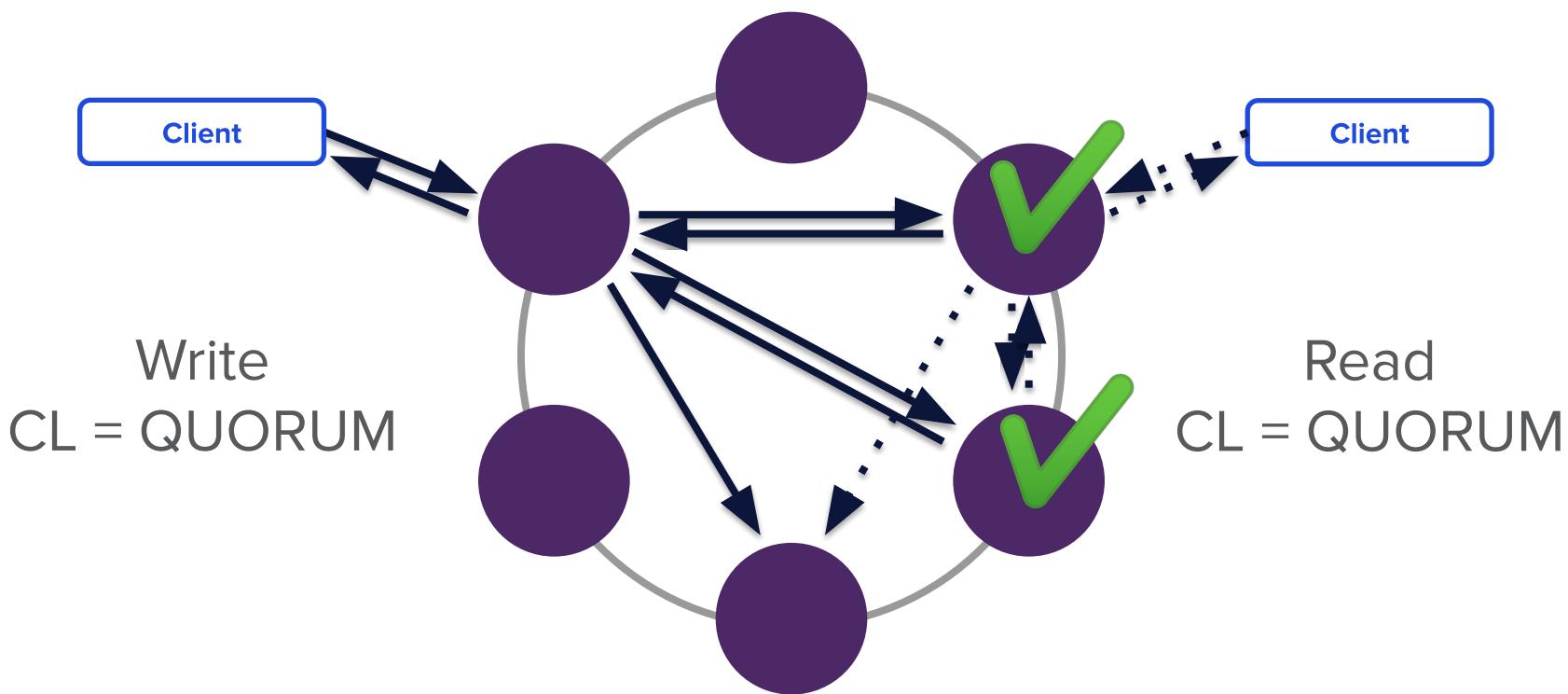
# Consistency Levels



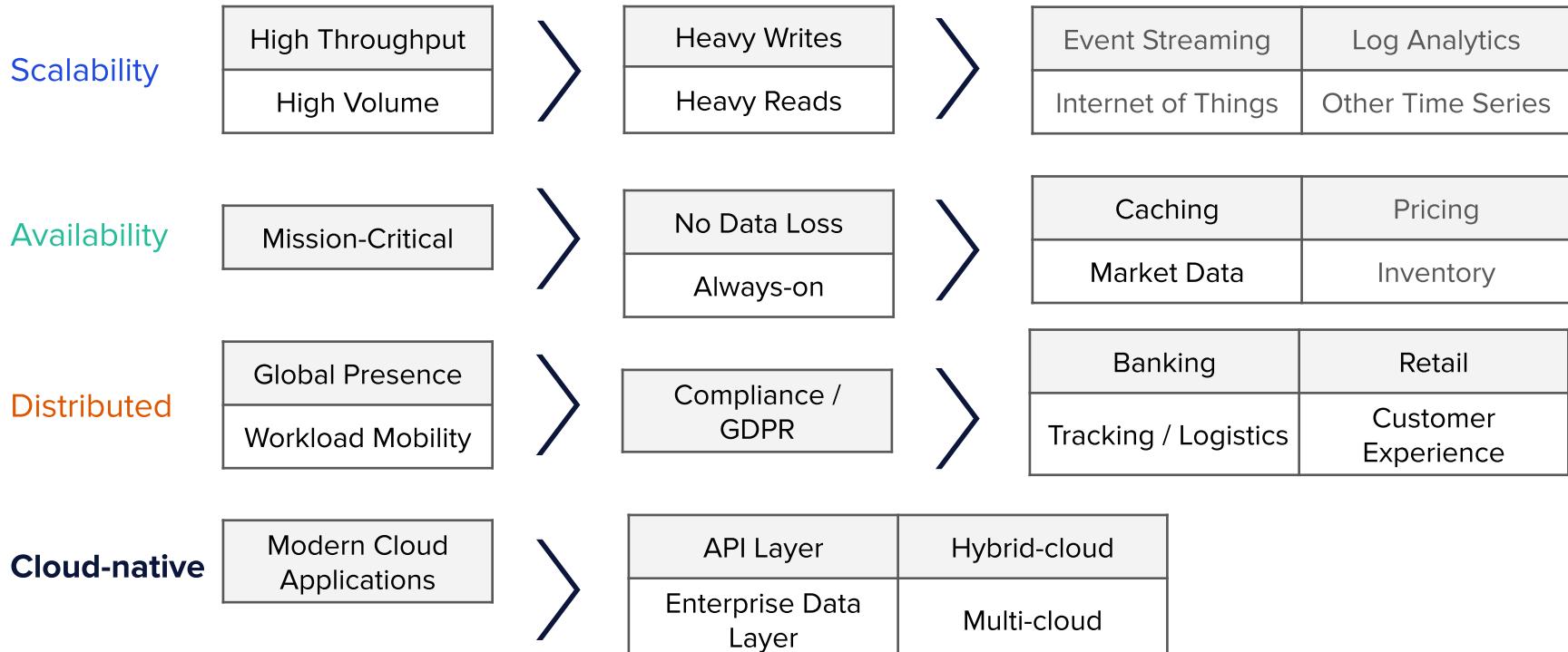
# Consistency Levels



**Immediate Consistency =  $CL_{read} + CL_{write} > RF$**



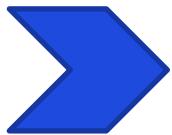
# Understanding Use Cases



# Exercise 2



DataStax Studio



Screenshot of the DataStax Studio interface showing a Markdown editor example:

The interface includes a header bar with the DataStax Studio logo, session information (cedrick.lunven@datastax...), and navigation icons.

## 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.

Language: **Markdown** ▾

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 ?

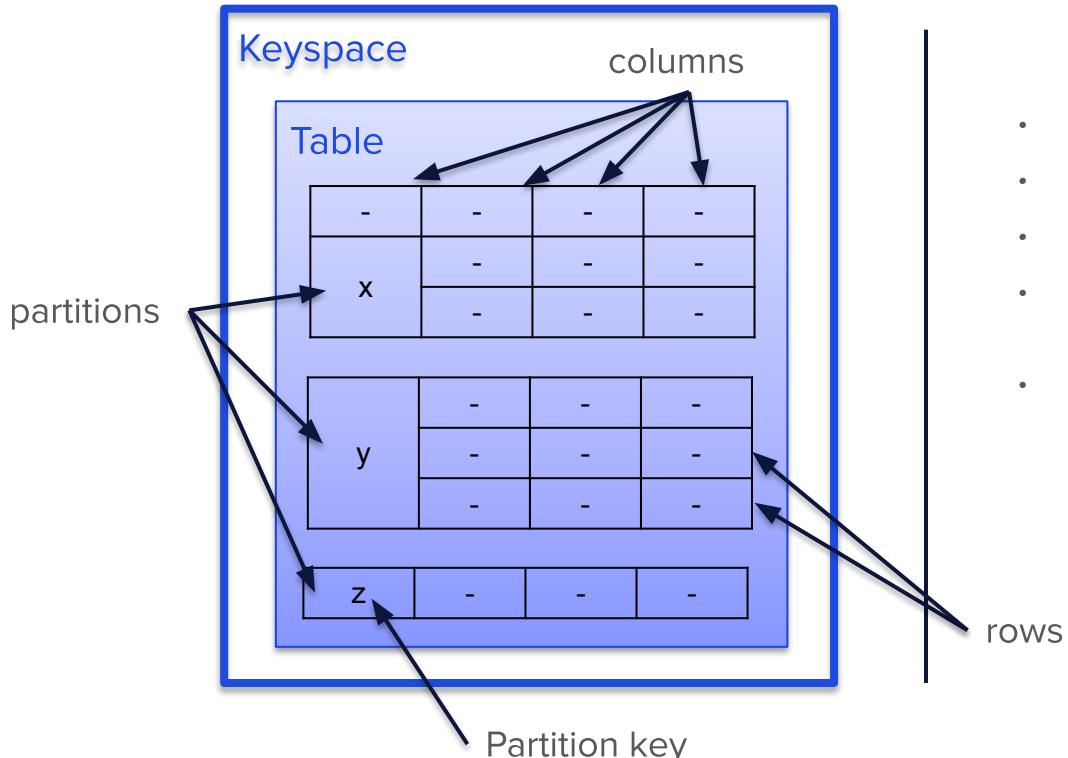


@DataStaxDevs #DataStaxDeveloperDay

<https://community.datastax.com>



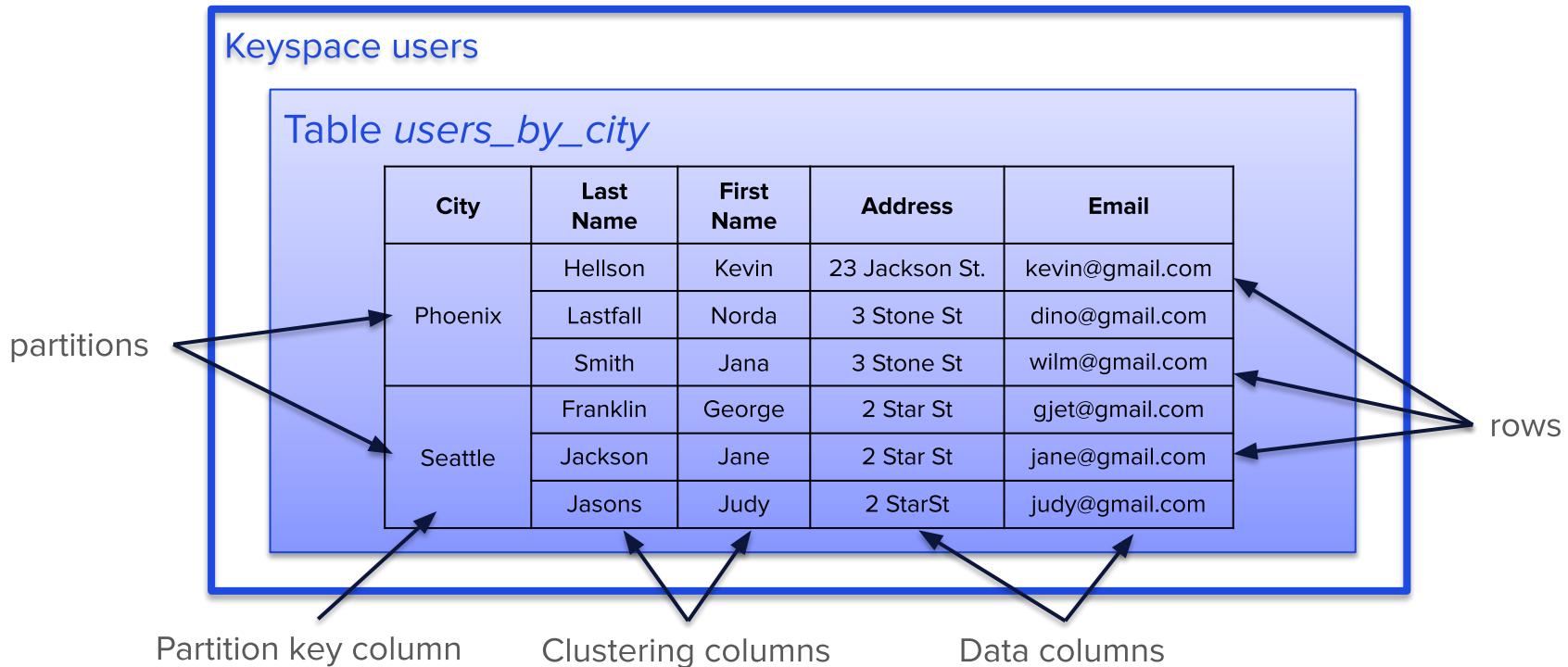
# Cassandra Structure - Partition



- 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

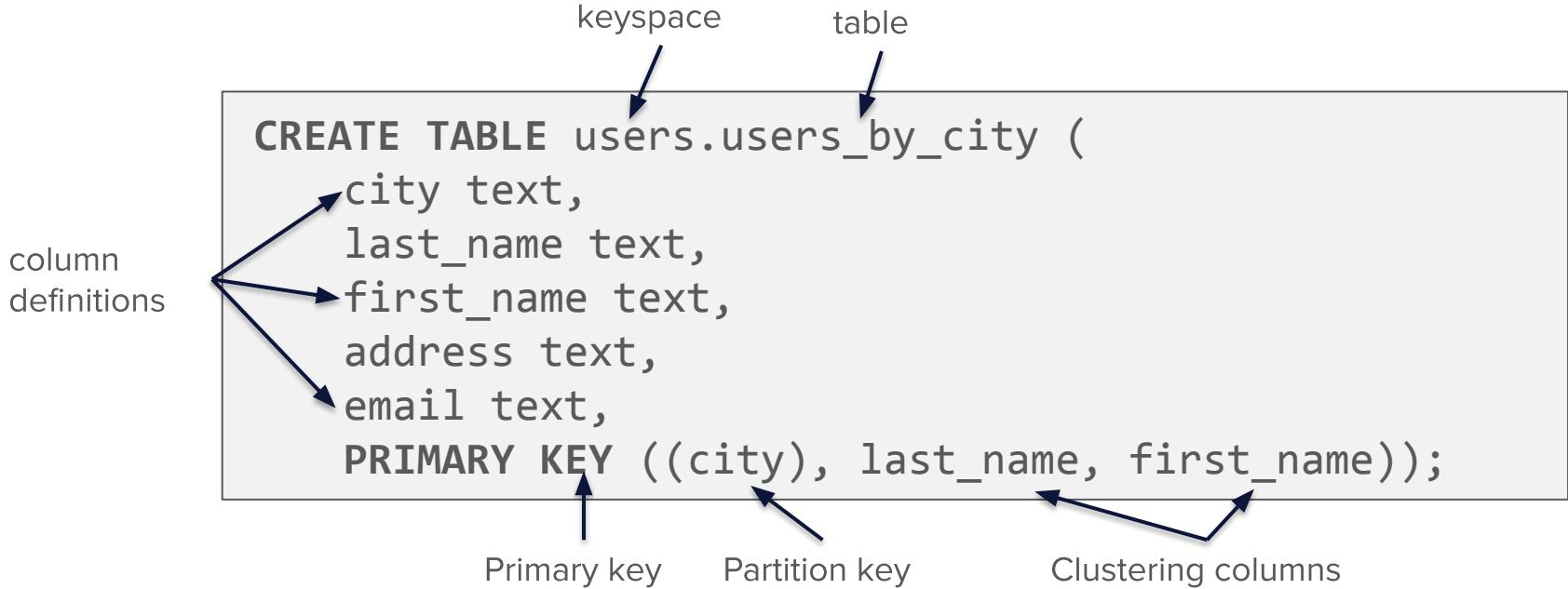
replication strategy

Replication factor by data center

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

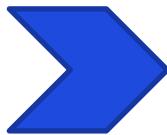


# Creating a Table in CQL



# Exercise 3

## Working with CQL



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

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.

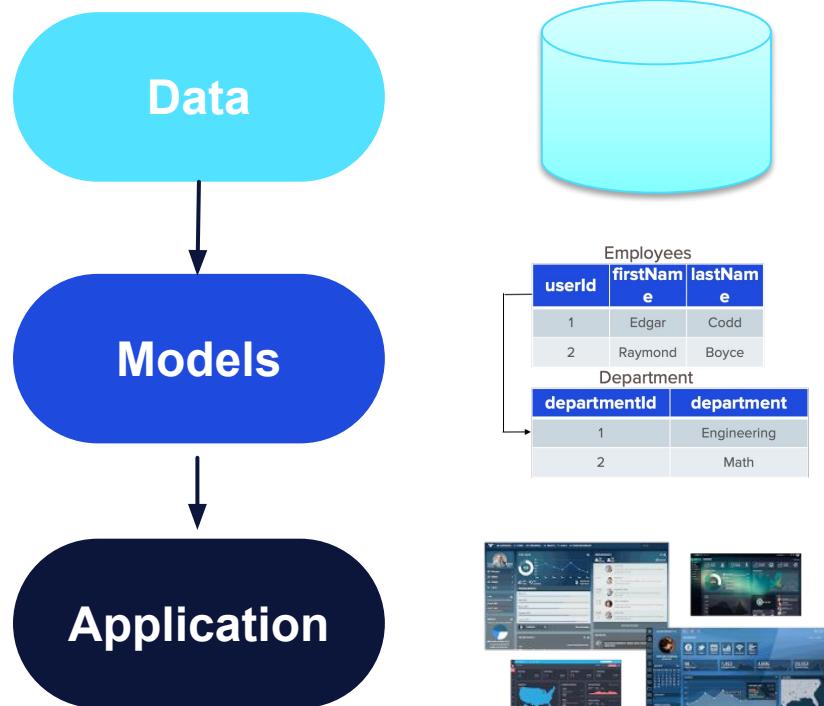
▶ Need a command hint? [Click here.](#)

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

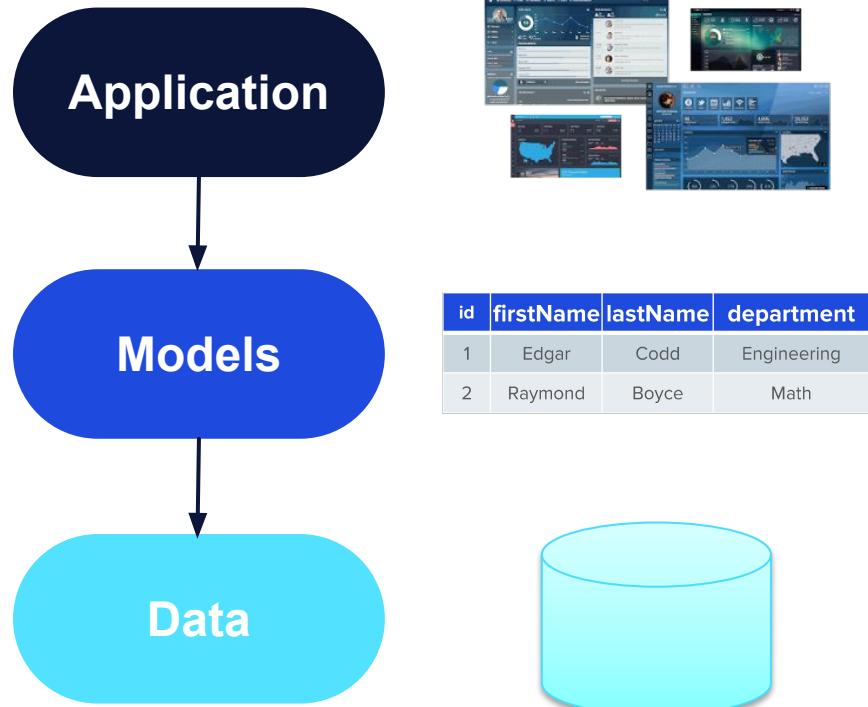
Language: **CQL**    Keyspace: **killrvideo**

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

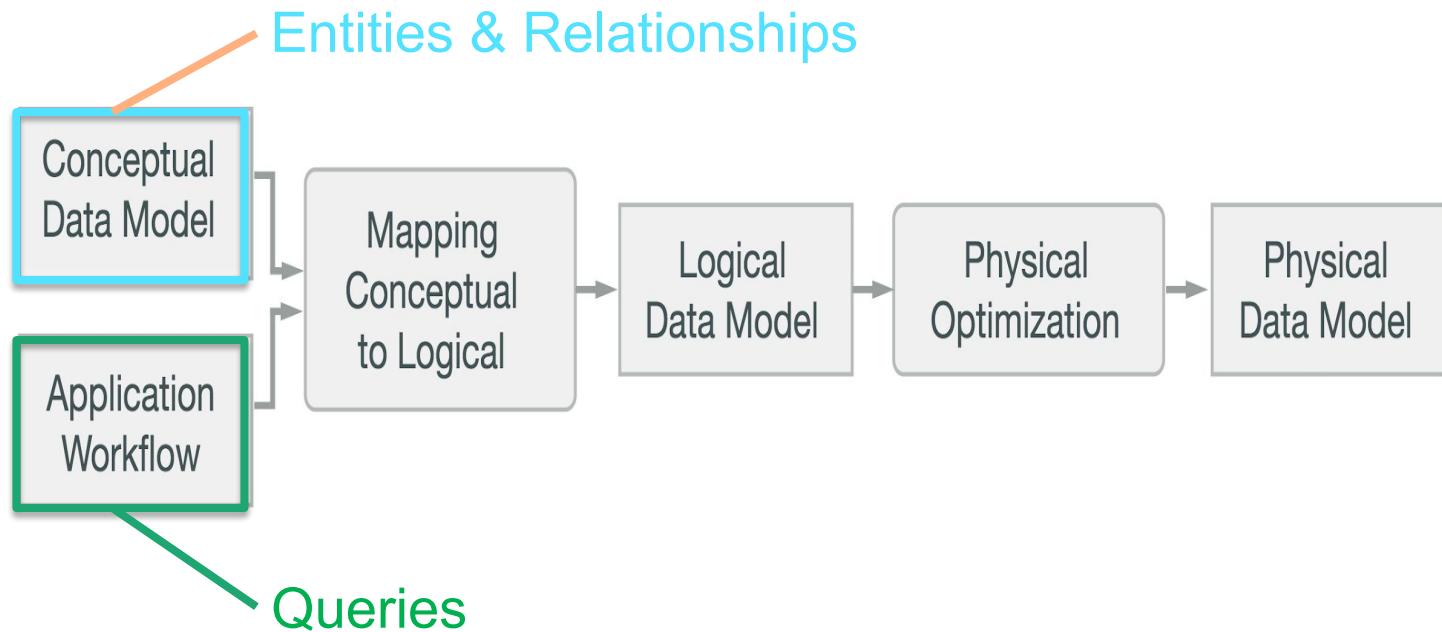
# Relational Modeling



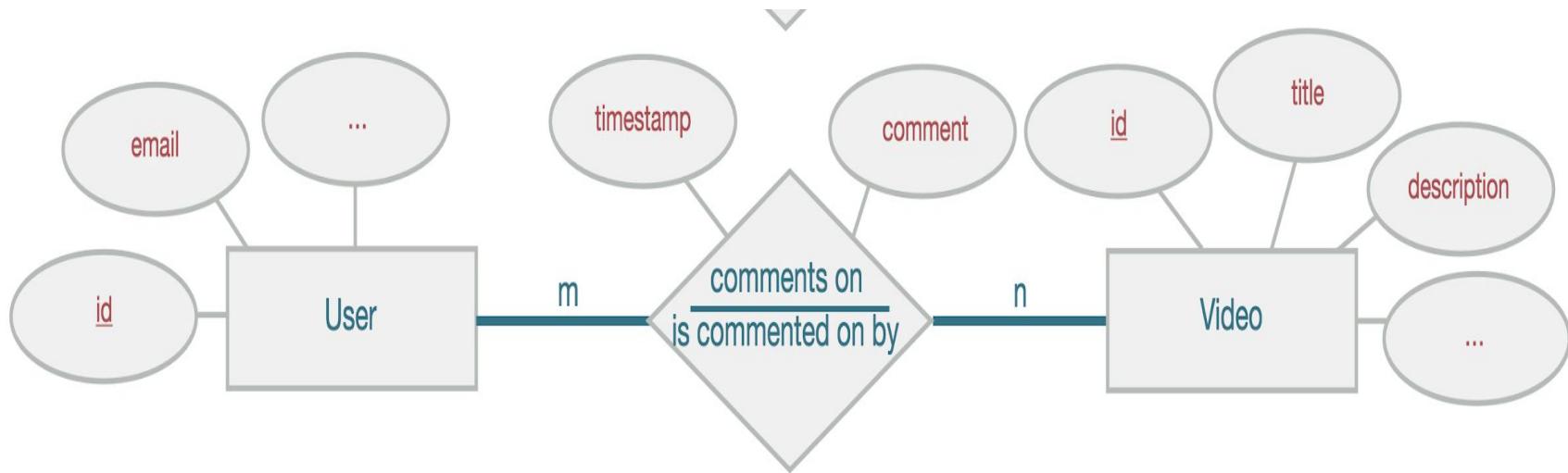
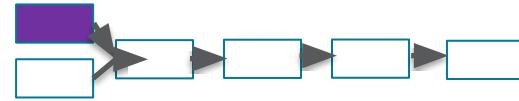
# Cassandra Modeling



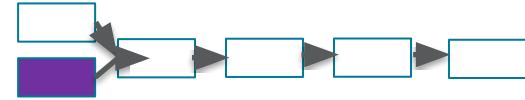
# Designing Data Model



# Conceptual Data Model



# Application Workflow



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

- Get most recent first
- Implement Paging

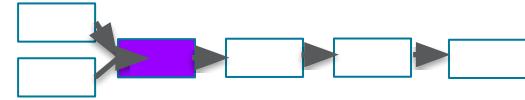
**R2:** Find **comments** related to target **user** using its identifier

- 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`

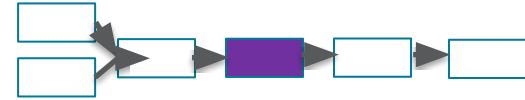
**Q2:** Find comments posted for a user with a known id (show most recent first)

→ `comments_by_user`

**Q3: CRUD Operations**



# Logical Data Model



comments\_by\_user

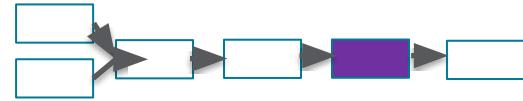
userid	K
creationdate	C ↓
commentid	C ↑
videoid	
comment	

comments\_by\_video

videoid	K
creationdate	C ↓
commentid	C ↑
userid	
comment	



# Physical Data Model

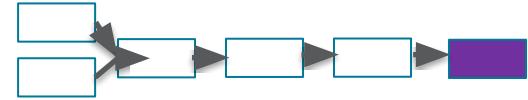


comments_by_user			
userid	UUID	K	
commentid	TIMEUUID	C	↓
videoid	UUID		
comment	TEXT		

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



# Schema DDL

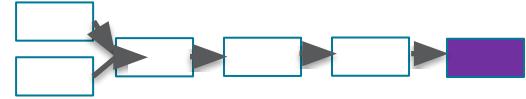


```
CREATE TABLE IF NOT EXISTS comments_by_user (
    userid uuid,
    commentid timeuuid,
    videoid uuid,
    comment text,
    PRIMARY KEY ((userid), commentid)
) WITH CLUSTERING ORDER BY (commentid DESC);
```

```
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



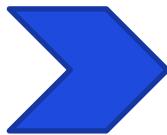
```
SELECT * FROM comments_by_user  
WHERE userid = <some UUID>
```

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



# Exercise 4

## Data Modeling



The screenshot shows the DataStax Studio interface during a Cassandra Developer Workshop. The title bar reads "Cassandra Developer Workshop #3 - Working with CQL". The sidebar on the right is titled "Schema" and shows the "killrvideo" keyspace, which contains 11 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, and videos. There are also sections for User Defined Types, Functions, Aggregates, and Materialized Views. The main content area displays a section titled "DESCRIBE KEYSPACE Command" with instructions for inspecting the keyspace and executing the command. The bottom of the screen shows the CQL language and the "killrvideo" keyspace selected.

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**

# 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**   
**Director of  
Developer Advocacy**



**Jack Fryer**   
**Community  
Manager**



**David Jones-Gilardi**   
**Developer Advocate**



**Aleksandr  
Volochnev**   
**Developer  
Advocate**



@DataStaxDevs #DataStaxDeveloperDay

<https://community.datastax.com>



# Developer Workshop

## PART II

*The RETURN*

1

Advanced Data Types

2

LightWeight Transactions (LWT) and Batches

3

Application Drivers

4

What's NEXT ?



# Leaderboard



# Developer Workshop

## PART II

*The RETURN*

1

Advanced Data Types

2

LightWeight Transactions (LWT) and Batches

3

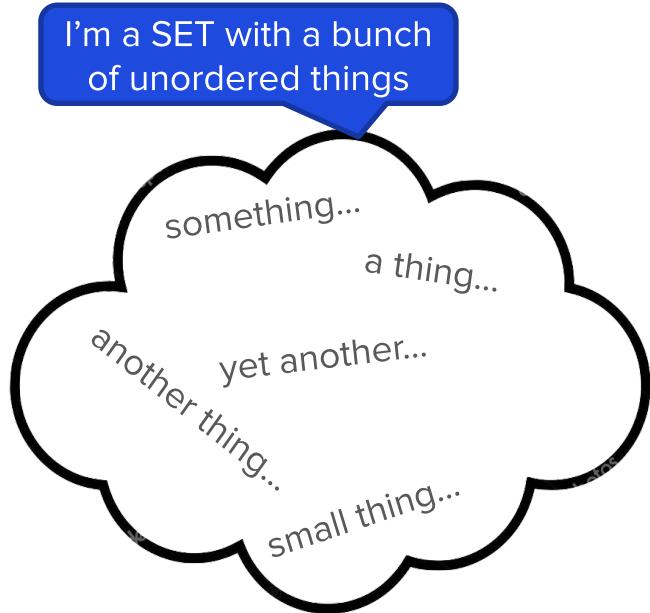
Application Drivers

4

What's NEXT ?



# Collections – Three Types



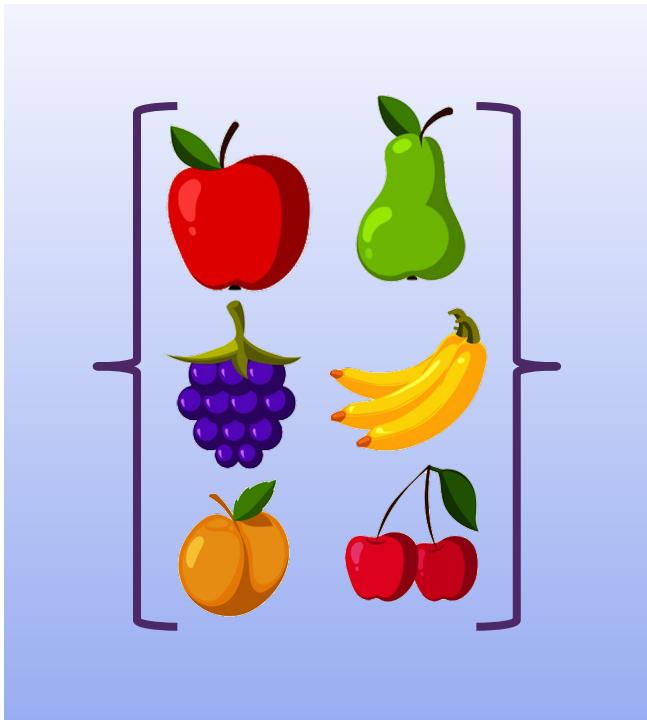
A table with two columns: "Key" and "Value". There are five rows, each consisting of a blue box for the key and a white box for the value. The data is as follows:

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

A blue speech bubble above the table contains the text: "I'm a MAP of key/value pairs".

# 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

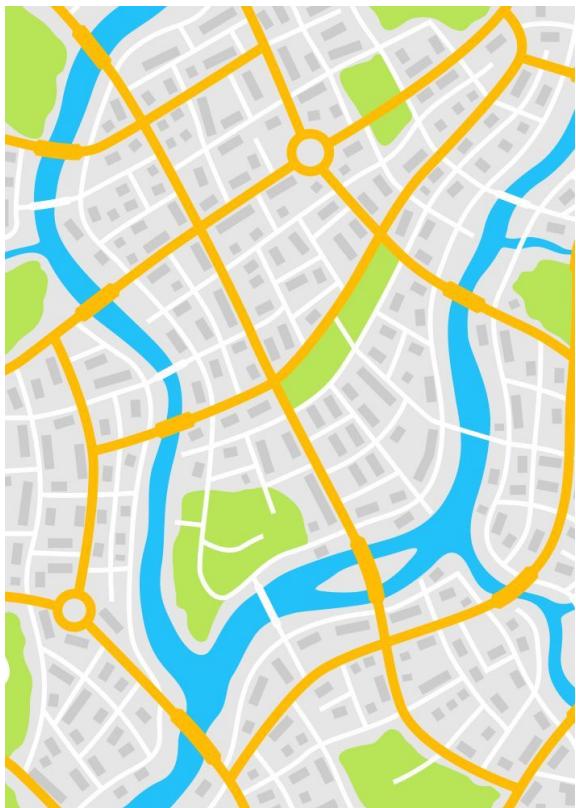
```
SET actors[1] = 'Brown'
```

```
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

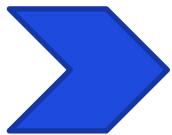
Decrementing a counter:

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

Just change the sign

# Exercise 5

## Advanced Types



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

# Developer Workshop

## PART II

*The RETURN*

1

Advanced Data Types

2

LightWeight Transactions (LWT) and Batches

3

Application Drivers

4

What's NEXT ?



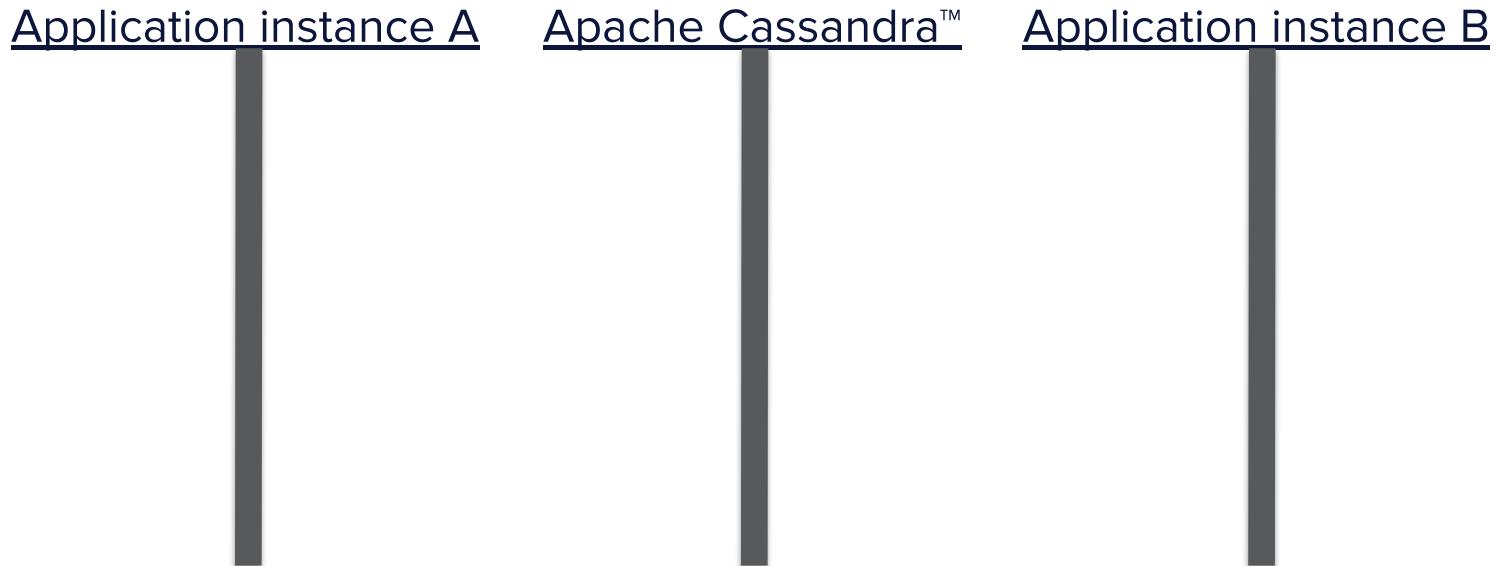
# Cassandra Lightweight Transactions

- 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!**

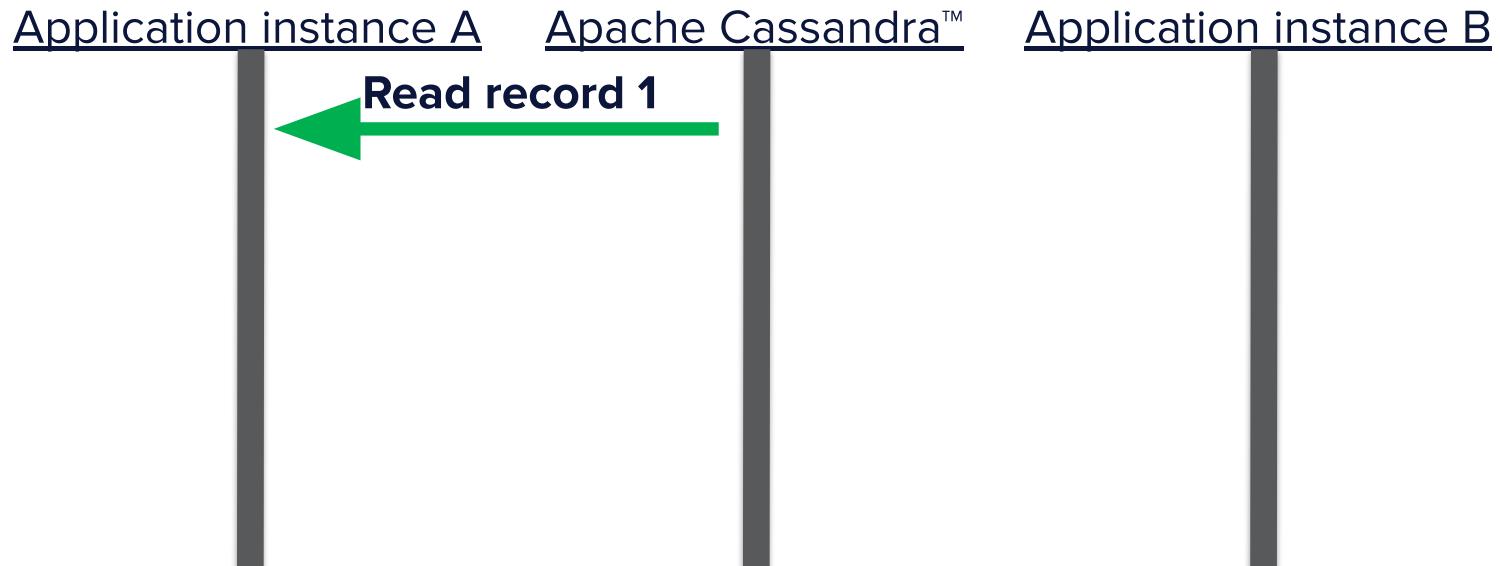
# Cassandra Lightweight Transactions

## The problem



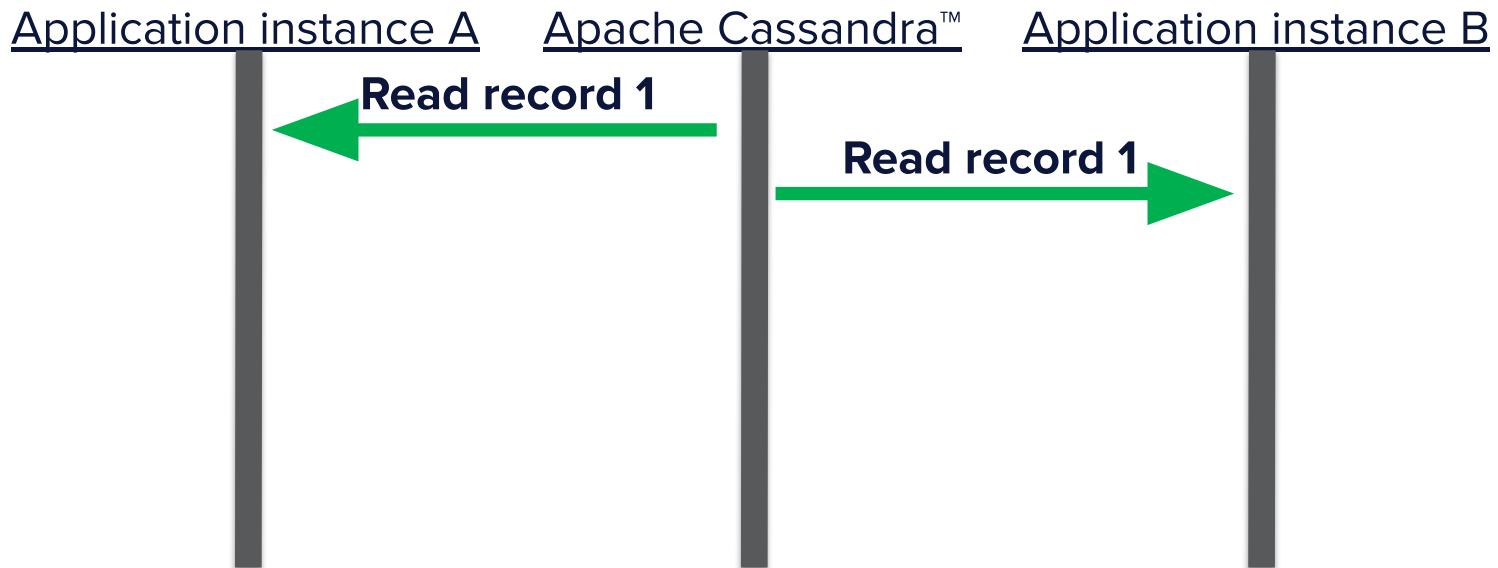
# Cassandra Lightweight Transactions

## The problem



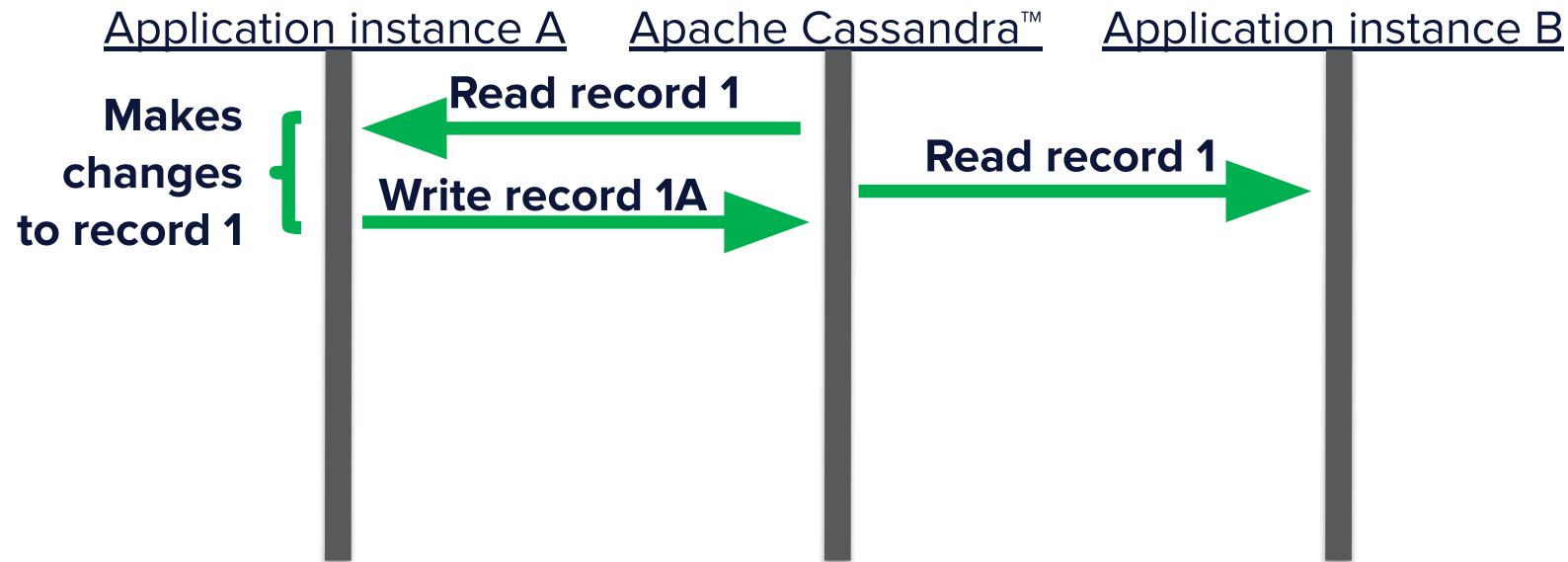
# Cassandra Lightweight Transactions

## The problem



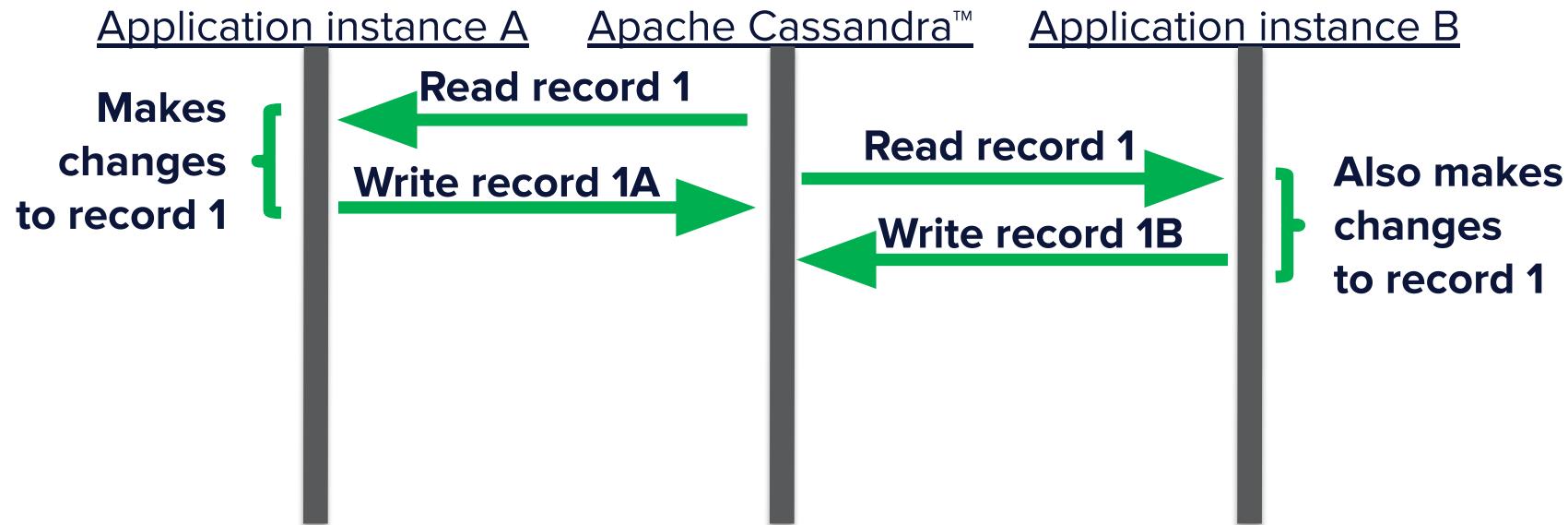
# Cassandra Lightweight Transactions

## The problem



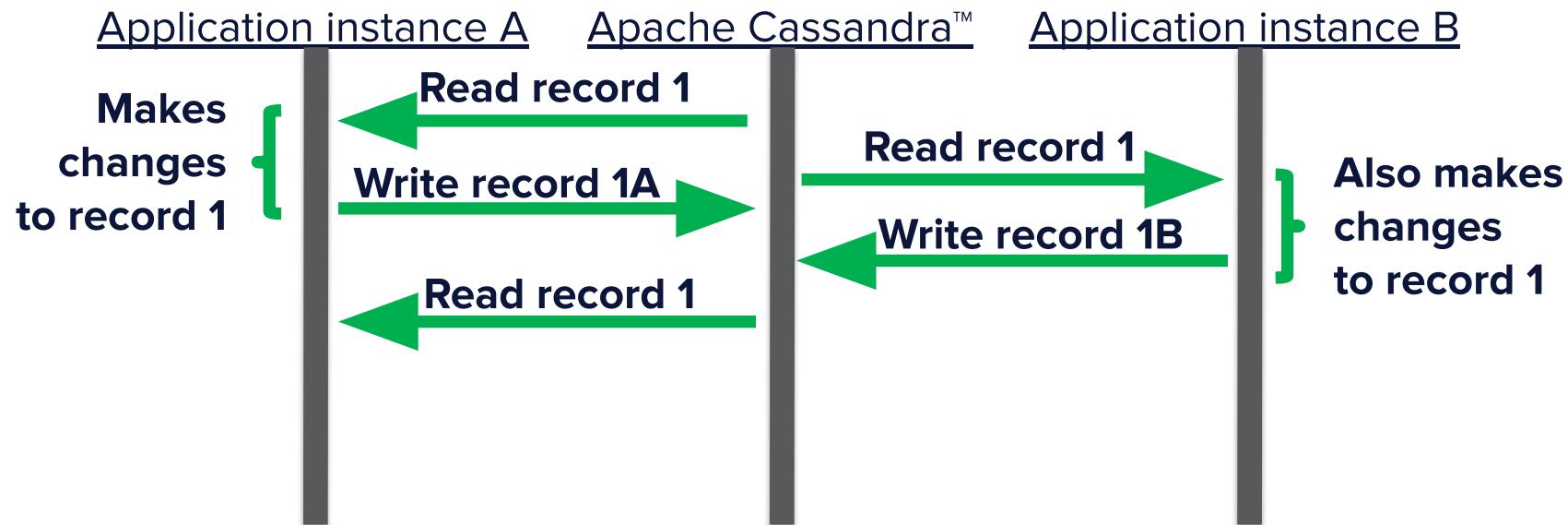
# Cassandra Lightweight Transactions

## The problem



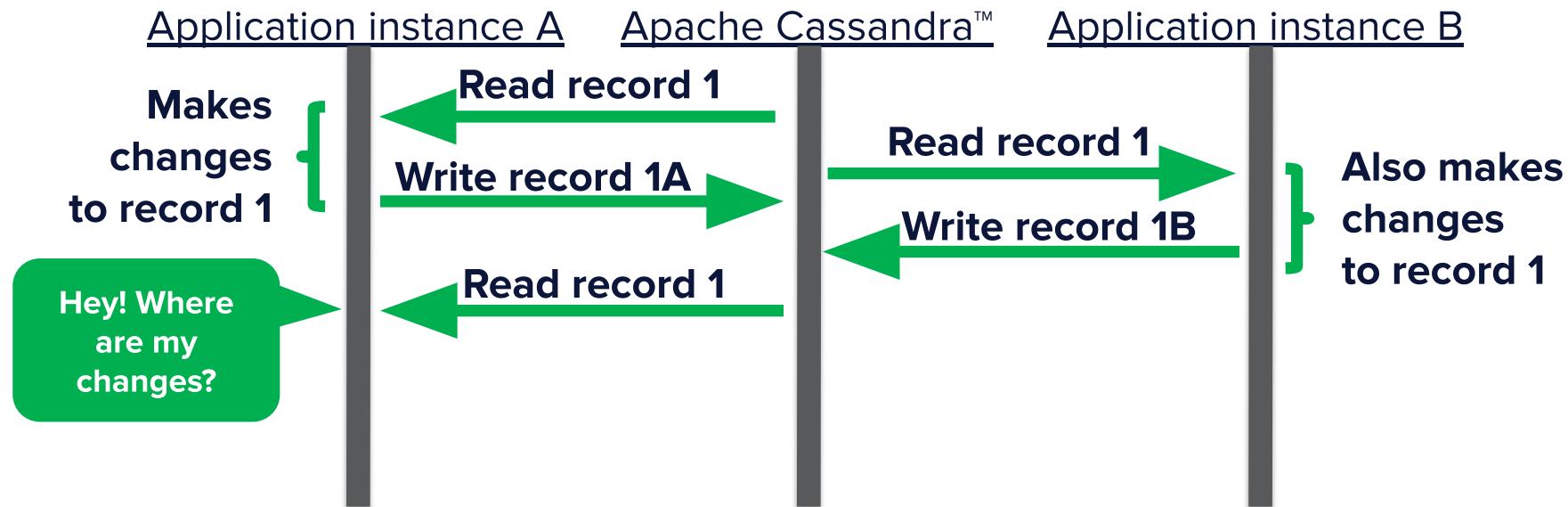
# Cassandra Lightweight Transactions

## 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;**



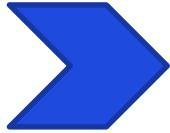
## So Why Use Batch?

- 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 Markdown editor window displays the following content:

```
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

On the right side of the interface, the schema browser shows the `killrvideo` CQL Keyspace, which contains the following tables:

- Tables
  - comments\_by\_user
  - comments\_by\_video
  - latest\_videos
  - tags\_by\_letter
  - user\_credentials
  - user\_videos
  - users
  - video\_playback\_stats
  - video\_ratings
  - video\_recommendations
  - video\_recommendations\_by\_video
  - videos
  - videos\_by\_tag
- User Defined Types
- User Defined Functions
- User Defined Aggregates
- Materialized Views

# Developer Workshop

## PART II

*The RETURN*

1

Advanced Data Types

2

LightWeight Transactions (LWT) and Batches

3

Application Drivers

4

What's NEXT ?



# DataStax Drivers Features



## Connectivity

- ★ Token & Datacenter Aware
- ★ Load Balancing Policies
- ★ Retry Policies
- ★ Reconnection Policies
- ★ Connection Pooling
- ★ Health Checks
- ★ Authentication | Authorization
- ★ SSL

## Query

- ★ CQL Support
- ★ Schema Management
- ★ Sync/Async/Reactive API
- ★ Query Builder
- ★ Compression
- ★ Paging

## Parsing Results

- ★ Lazy Load
- ★ Object Mapper
- ★ Spring Support
- ★ Paging

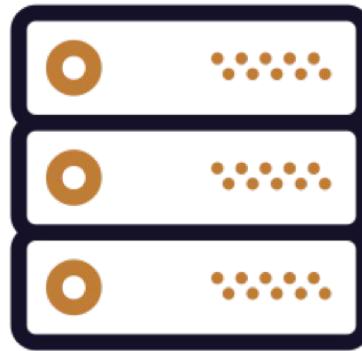


# Drivers

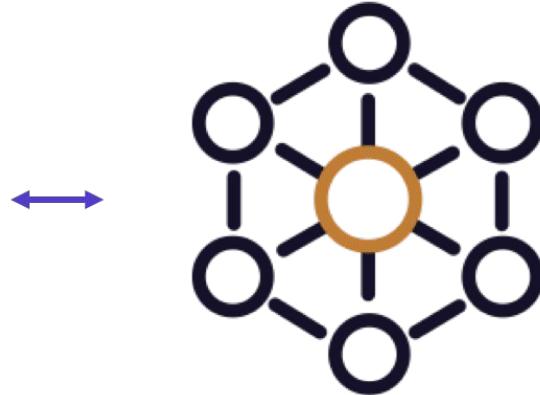
Your Killer App



Our Cool Driver



Cassandra Database



# Dependencies



```
<dependency>
  <groupId>com.datastax.oss</groupId>
  <artifactId>java-driver-core</artifactId>
  <version>${driver.version}</version>
</dependency>
```

Copy



```
"dependencies": {
  "cassandra-driver": "^4.5.1",
  "dse-driver": "^2.3.1"
}
```

## cassandra-driver 3.23.0

pip install cassandra-driver



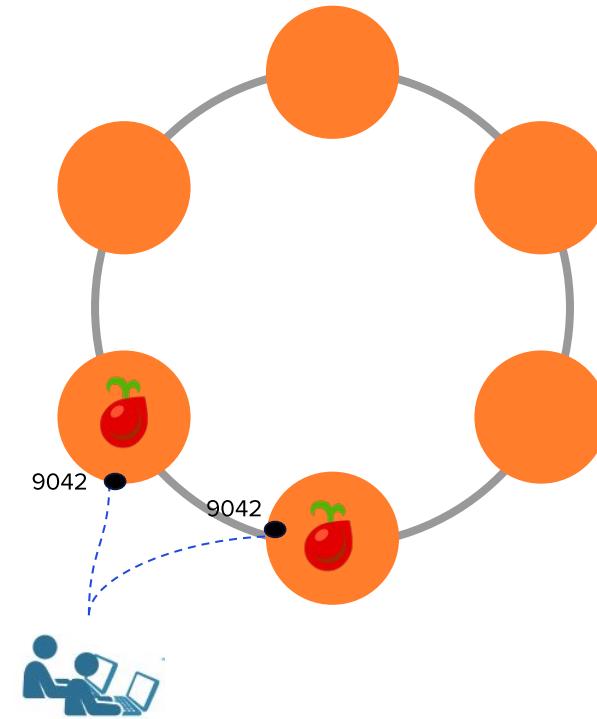
@DataStaxDevs #DataStaxDeveloperDay

<https://community.datastax.com>



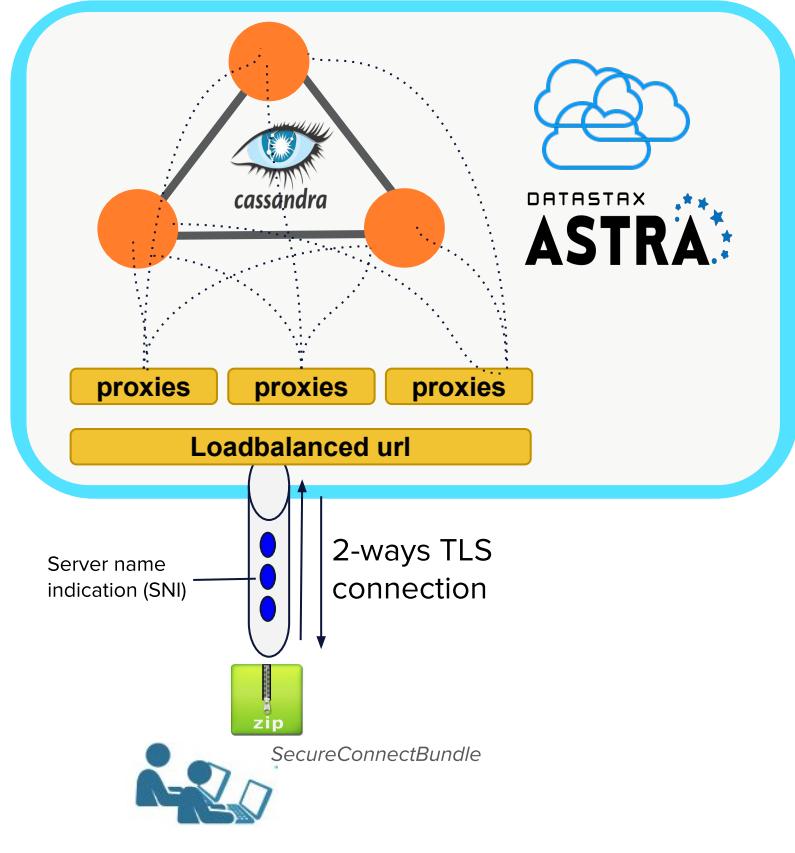
# Contact Points

- Only one necessary
- Unless that node is down
- More are good



# Contact Points with ASTRA

- Download the **SecureConnectBundle** from Astra UI.
- Provide **username, password and keyspace name in the BUILDER.**





# Open a Session

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

// Explicit Settings
CqlSession cqlSession = CqlSession.builder()
    // .addContactPoint(new InetSocketAddress("127.0.0.1", 9042))
    .withCloudSecureConnectBundle(Paths.get("/tmp/apollo.zip"))
    .withContactPoint
    .withKeyspace("killrvideo")
    .withAuthCredentials("KVUser", "KVPassword")
    .build();
```



# Open the Session



```
try (DseSession session = DseSession.builder()
    .withCloudSecureConnectBundle('/path/to/secure-connect-database_name.zip')
    .withAuthCredentials('DBUserName', 'DBPassword')
    .build()) {
```



```
const client = new Client({
  cloud: { secureConnectBundle: 'path/to/secure-connect-database_name.zip' },
  credentials: { username: 'DBUsername', password: 'DBPassword' }
});
```



```
cluster = Cluster(
  cloud={ 'secure_connect_bundle': '/path/to/secure-connect-database_name.zip' },
  auth_provider=PlainTextAuthProvider('DBUsername', 'DBPassword') )
session = cluster.connect()
```

# Important to know about CqlSession

- **CqlSession** is a stateful object handling communications with each node
- **CqlSession** should be unique in the Application (*Singleton*)
- **CqlSession** should be closed at application shutdown (*shutdown hook*) in order to free opened TCP sockets (*stateful*)

```
@PreDestroy  
public void cleanup() {  
    if (null != cqlSession) {  
        cqlSession.close();  
    }  
}
```



# 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);  
  
cqlSession.execute(statement);
```



# Executing the queries



```
session.execute(  
    SimpleStatement.builder("SELECT password FROM killrvideo.user_credentials WHERE email=?")  
    .addPositionalValues("patrick@datastax.com")  
    .build());
```



```
client.execute('SELECT password FROM killrvideo.user_credentials WHERE email=?',  
    ['patrick@datastax.com'])
```



```
session.execute(("SELECT password FROM killrvideo.user_credentials WHERE email=%s",  
    ('patrick@datastax.com')))
```



@DataStaxDevs #DataStaxDeveloperDay

<https://community.datastax.com>



# 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);
```



# 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();
```



# Parsing Rows

```
// Sample row
Row row = resultSet.one();

// Check null before read
Boolean isUserNameNull = row.isNull("username");

// Reading Values from row
String userName1 = row.get("username", String.class);
String userName2 = row.getString("username");
String userName3 = row.getString(CqlIdentifier.fromCql("username"));

// Tons of types available
row.getUuid("userid");
row.getBoolean("register");
row.getCqlDuration("elapsed");
...
```



# Paging

- ResultSet contains up to “**pageSize**” items. When browsing records you may hit this number that will trigger fetching next “pageSize” items.
- To fetch anything else than first page you must provide a **PagingState**.

```
// Enforce few items per page (often = UI requirements)
myStatement = myStatement.setPageSize(10);
ResultSet page1 = cqlSession.execute(myStatement);

// Paging State
ByteBuffer pagingState = page1.getExecutionInfo().getPagingState();
myStatement = myStatement.setPageState(pagingState);

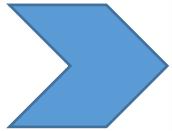
// Very same statement with pagingState provided
ResultSet page2 = cqlSession.execute(myStatement);
```



# Exercise 7



Coding

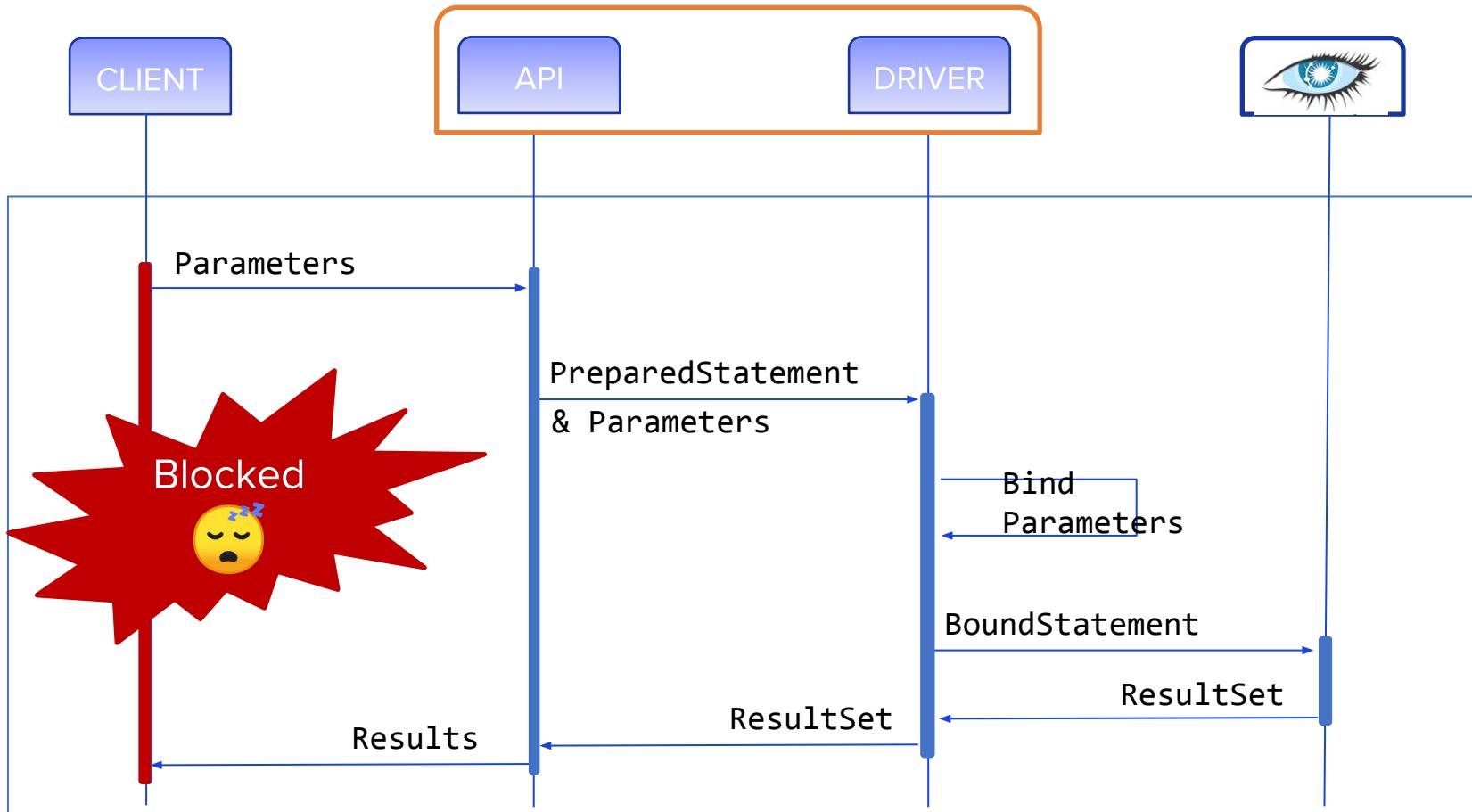


The screenshot shows a code editor interface with a dark theme. On the left is the Explorer sidebar showing project structure:

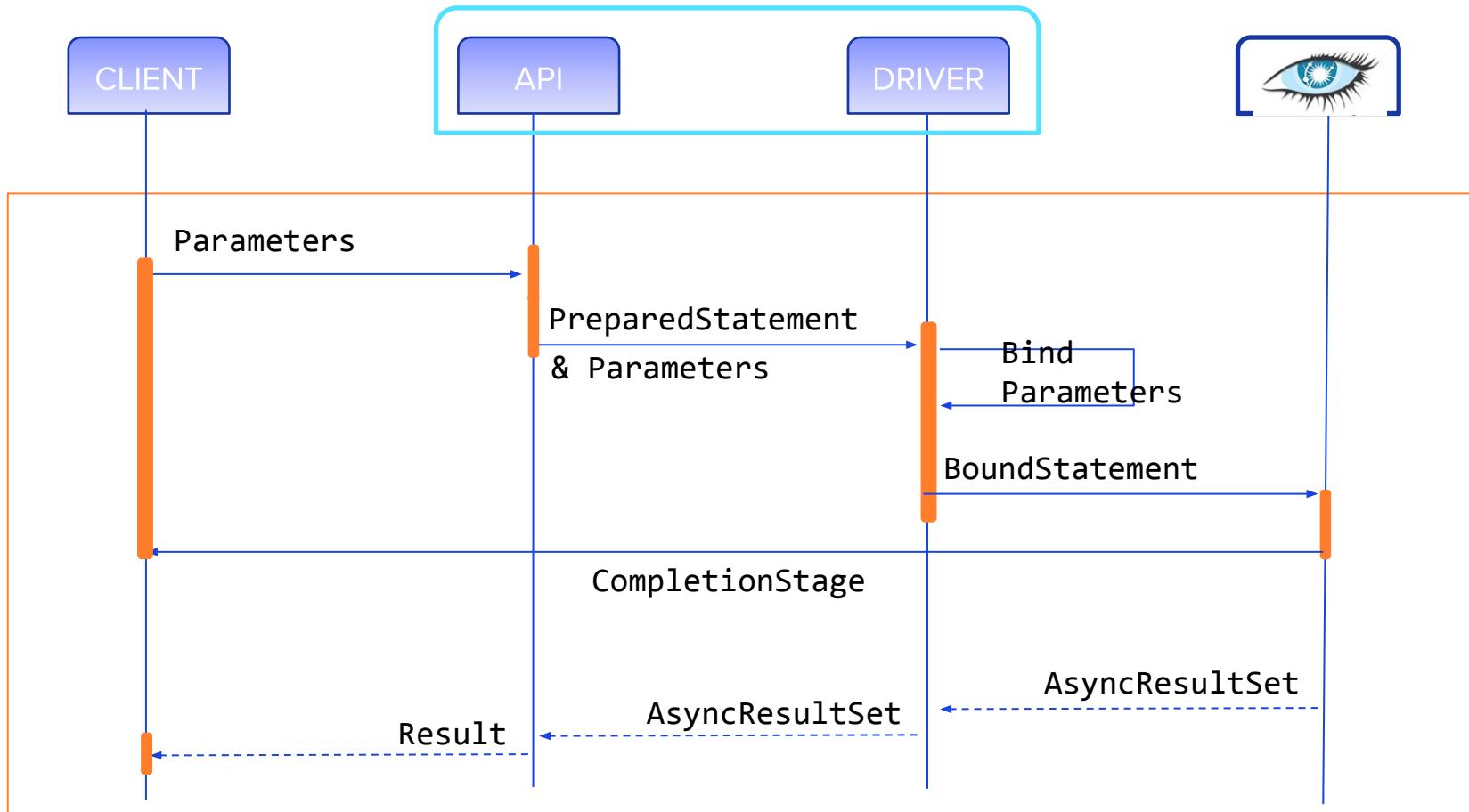
- crud-java (selected)
- src
- main
- java
  - DBConnection.java
  - Delete.java (selected)
  - Insert.java
  - SelectRows.java
  - Update.java
- resources
- target
- pom.xml
- schema.cql
- crud-node.js
- crud-python
- killrvideo-java
- killrvideo-nodejs
- killrvideo-python

The right pane displays the content of the Delete.java file:

```
1 import com.datastax.dse.driver.api.core.DseSession;
2 import com.datastax.oss.driver.api.core.cql.*;
3
4 public class Delete {
5
6     public static void main(String[] args) {
7
8         try (DseSession session = DseSession.builder().withCloudSecureConnectBundle(DBConnection.
9             .withAuthCredentials(DBConnection.getUsername(), DBConnection.getPassword())
10            .build()) {
11
12             session.execute(
13                 SimpleStatement.builder( "DELETE FROM killrvideo.user_credentials WHERE email =
14                     .addPositionalValues("cv@datastax.com")
15                     .build());
16             System.out.println("Successful Delete");
17         }
18     catch(Throwable t) {
19         System.out.println("Failed Delete");
20         t.printStackTrace();
21     }
22 }
23
24 }
```

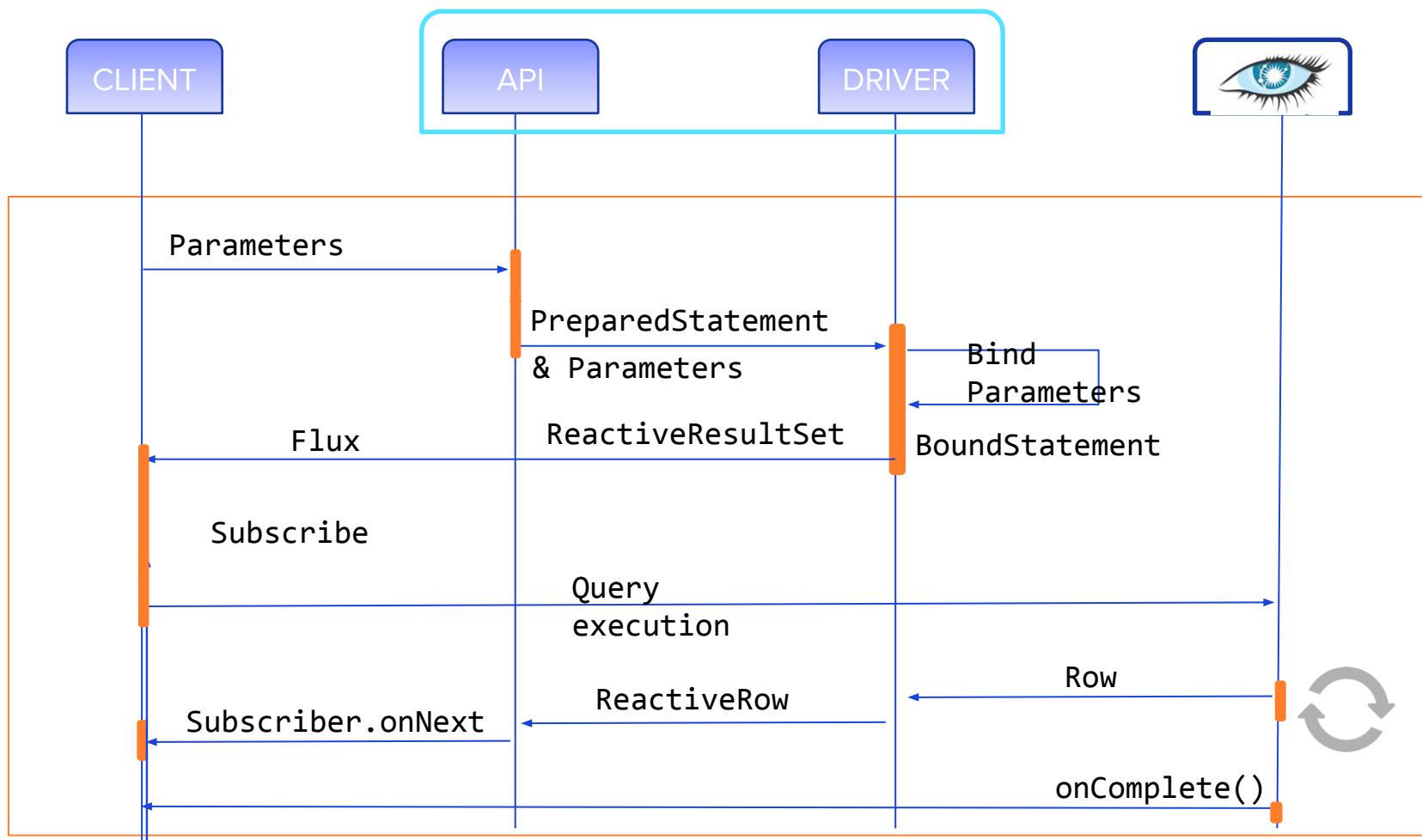


Synchronous Queries



## Asynchronous Queries





# Developer Workshop

## PART II

*The RETURN*

1

Advanced Data Types

2

LightWeight Transactions (LWT) and Batches

3

Application Drivers

4

What's NEXT ?



# Developer Resources

**LEARN**

Join [academy.datastax.com](https://academy.datastax.com)

<https://academy.datastax.com/resources/cassandra-developer-workshop>

Free online courses - Cassandra certifications

**ASK/SHARE**

Join [community.datastax.com](https://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>



[www.datastax.com/keepcalm](http://www.datastax.com/keepcalm)

SRE office hours for Apache Cassandra

Cassandra cluster health checks

@ no charge

Send an email to **keepcalm@datastax.com**



@DataStaxDevs #DataStaxDeveloperDay

<https://community.datastax.com>



# Training Courses at DataStax Academy

- Free self-paced DSE 6 courses
  - DS201: DataStax Enterprise 6 Foundations of Apache Cassandra™
  - DS210: DataStax Enterprise 6 Operations with Apache Cassandra™
  - DS220: DataStax Enterprise 6 Practical Application Data Modeling with Apache Cassandra™
  - DS330: DataStax Enterprise 6 Graph
  - DS332: DataStax Enterprise 6 Graph Analytics (NEW)



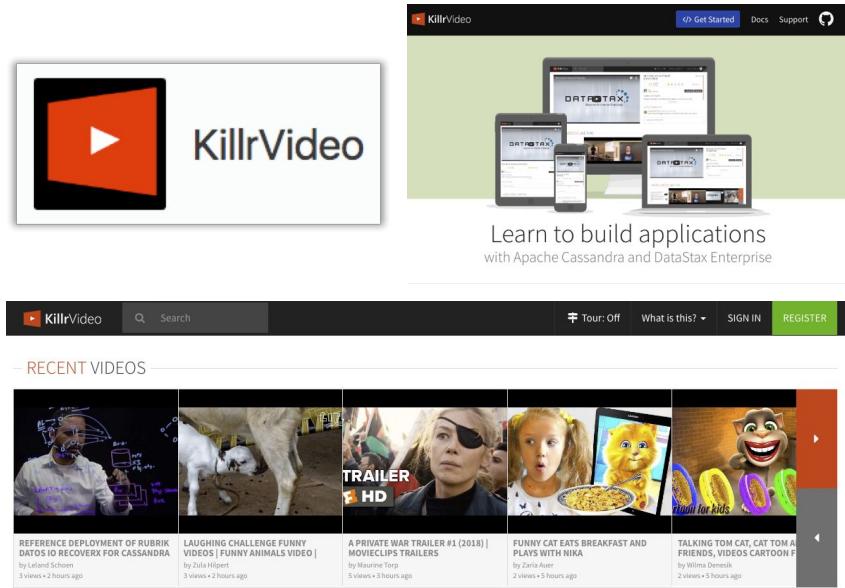
@DataStaxDevs #DataStaxDeveloperDay

<https://community.datastax.com>



# KillrVideo Reference Application

- Reference application for learning how to use Apache Cassandra and DataStax 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>



# Upcoming events

Date	Time	Content	Type
5/13	11am EDT	Cassandra Kubernetes Operator	WORKSHOP = HANDS-ON
5/18	12pm IST	Cassandra Developer Workshop (SAME)	WORKSHOP = HANDS-ON
5/20	12pm PT	Astra BetterBoz	WORKSHOP = HANDS-ON
5/27	12pm EDT	OSS Fallout	WORKSHOP = HANDS-ON
6/3	12pm EDT	NoSQLBench : Benchmark Your Data Models	WORKSHOP = HANDS-ON



@DataStaxDevs #DataStaxDeveloperDay

<https://community.datastax.com>





# Thank You

