



# From Zero to Cloud with Apache Cassandra™

NOVEMBER 25th 2019

Cedrick Lunven - Developer Advocate DataStax



# About me



@clunven



meetup

BIGDATA  
LDN. MEETUP



<https://fr.linkedin.com/in/clunven>

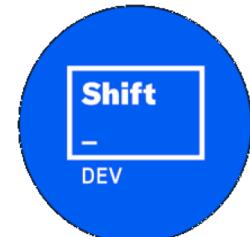
DevFest du  
Bout du Monde

VOXXED DAYS  
MICROSERVICES

TIN #19

Breizh C@mp  
La conférence à l'Ouest

VOXXED DAYS  
LUXEMBOURG



KILLRVIDEO



# Agenda

0

Bootstraping

1

Understanding Apache Cassandra™ Use Cases

2

Data Modelling with Apache Cassandra™

3

Application Development

4

What's NEXT ?

# Meetup @Stockholm

---

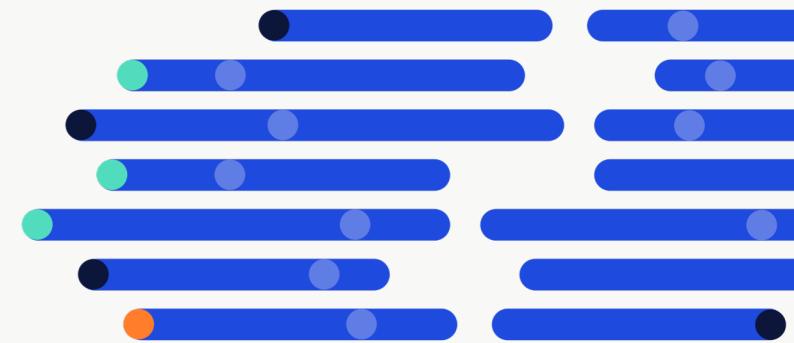
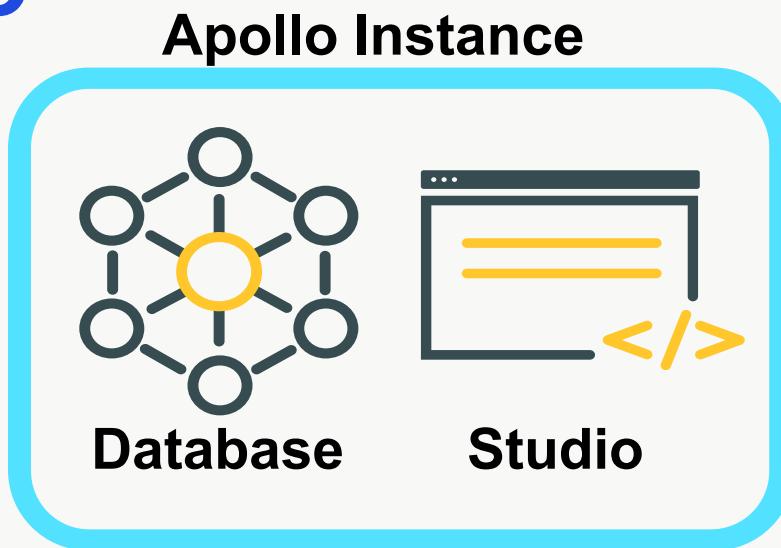


## Bootstrapping

# Exercise



## Set-up Apollo



# Apollo.datastax.com

← Databases > Create New Database

Location

AWS

us-east-1

Database Details

Database Name \*

MyDB

Keyspace Name \*

demo\_apollo

Enter alphanumeric characters only. Use quotes to preserve case.

Database User name \*

cedrick

Database User password \*

.....

Confirm Password \*

.....

Launch Database

Cancel

Compute Size

Choose the compute power of your database. Higher tiers provide higher throughput at lower latency.

- Developer: low-cost development option with 25 GB storage.
- Startup: starter configuration for development and light production workloads.
- Standard: higher throughput, lower latency, 99.9% uptime SLA, plus support.
- Enterprise: highest throughput, lowest latency, 99.99% uptime SLA, and best support tier.

Estimated Cost

TBD

# Let's things initializing...

 Database is initializing. Look for an email once it is ready to use. Feel free to close this page and come back later.

Databases > MyDB Actions ▾

Summary Health

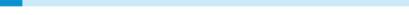
MyDB

Keyspace Name: demo\_apollo  
Organization: Your Organization

---

Owner: Cedrick Lunven  
Created: October 11, 2019

Size and Location

<b>1</b> Capacity Unit	<b>500 GB</b> Total Storage	<b>Storage Used</b> 
Locations:	us-east-1 (1 capacity unit)	
Compute Size:	Developer	
Replication Factor:	3	

# Simplify application development

DATASTAX  
**APOLLO**



Local Desktop



Public IP  
Endpoint



DATASTAX  
**APOLLO**



CLIENT

Client VPC

# Simplify application development

DATASTAX  
**APOLLO**

1-BUTTON CLICK FROM CLOUD CONSOLE TO WEB-HOSTED INTEGRATED DEVELOPER ENVIRONMENT (DATASTAX STUDIO)

DATASTAX CONSTELLATION Your organization

matt.kennedy@datastax.com

Spent this month: TBD    Estimated bill this month: TBD    Credit remaining: TBD

Databases > Create New Database

Location

AWS    GCP    Azure

Select a region.

Database Details

Database Name \*    Keyspace Name \*

Unquoted names are forced to lowercase.

username \*    Confirm Password \*

Launch Database    Cancel

DataStax Studio    transactions

CQL    Keyspace: example\_keyspace

select \* from transactions where account\_number='1234123412341240' and transaction\_time > '2017-09-06';

index | account\_number | transaction\_time | amount | items | location | merchant | notes | status | tags

0	1234123412341240	2017-09-06T16:31:46.959+0000	1458.74	{} []	Tampa	PizzaHut	HouseHold		[{"HouseHold","Home"}]
1	1234123412341240	2017-09-06T14:54:43.125+0000	696.57	{} []	Atlanta	Amazon	Shopping	SUCCESS	[{"Shopping","Home"}]
2	1234123412341240	2017-09-06T04:45:51.837+0000	240.46	{} []	West Palm Beach	Amazon	Shopping		[{"Shopping","Home"}]
3	1234123412341240	2017-09-06T00:06:44.733+0000	528.49	{} []	Dallas	Target	Shopping	CANCELLED	[{"Shopping","Home"}]

Displaying 1 - 4 of 4 results for the last statement

Success. 4 elements returned. Duration: 0.03 s.

default    Schema example\_keyspace   

example\_keyspace CQL Keyspace

Tables

Columns

account\_number (text)  
amount (double)  
items (map<text, double>)  
location (text)  
merchant (text)  
notes (text)  
status (text)  
tags (set<text>)  
transaction\_id (text)  
transaction\_time (timestamp)  
user\_id (text)

Partitioning Key

Clustering Keys

Secondary Indexes

Search Indexes

Materialized Views

transactions\_by\_location

transactions\_by\_merchant

User Defined Types

User Defined Functions

# Agenda

0

Bootstraping

1

Understanding Apache Cassandra™ Use Cases

2

Data Modelling with Apache Cassandra™

3

Application Development

4

What's NEXT ?

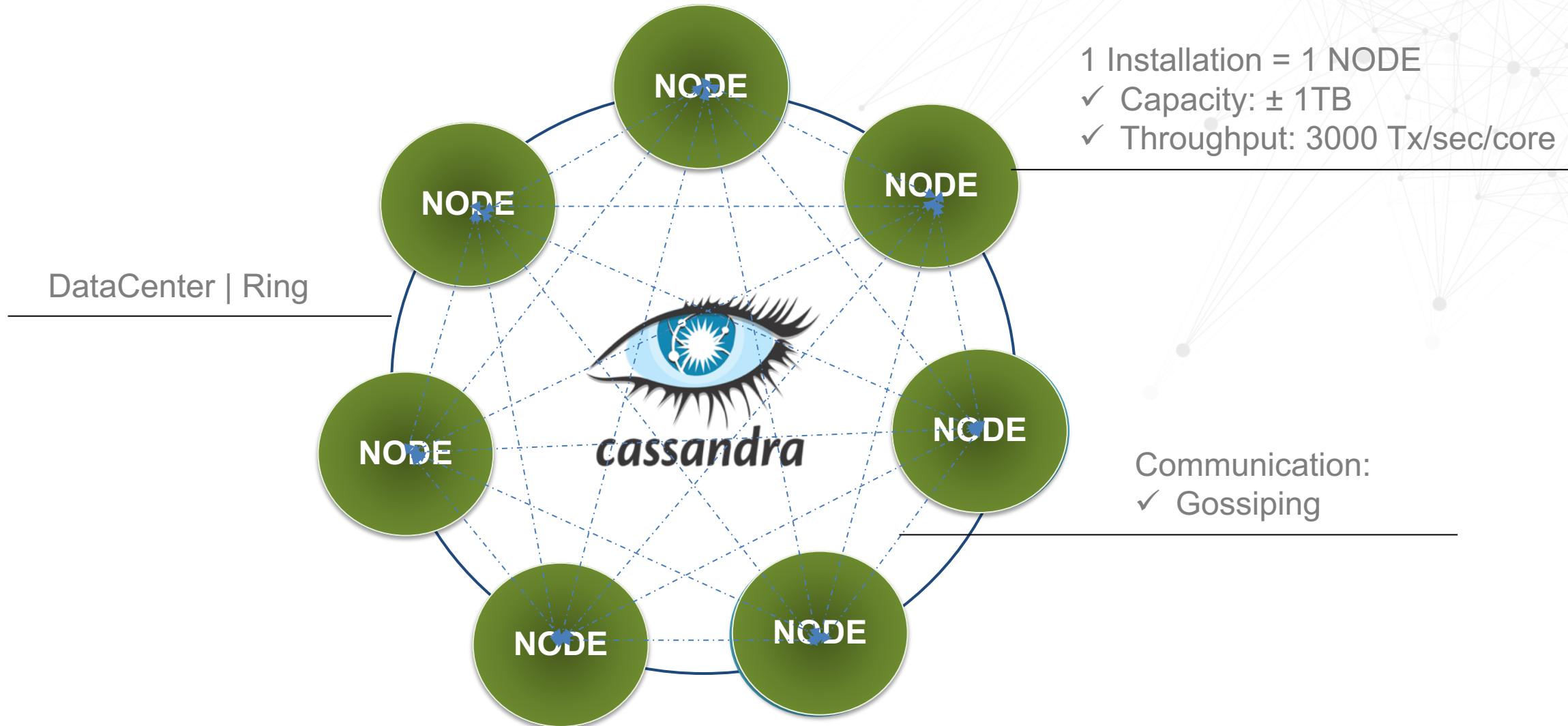
# Meetup @Stockholm

---



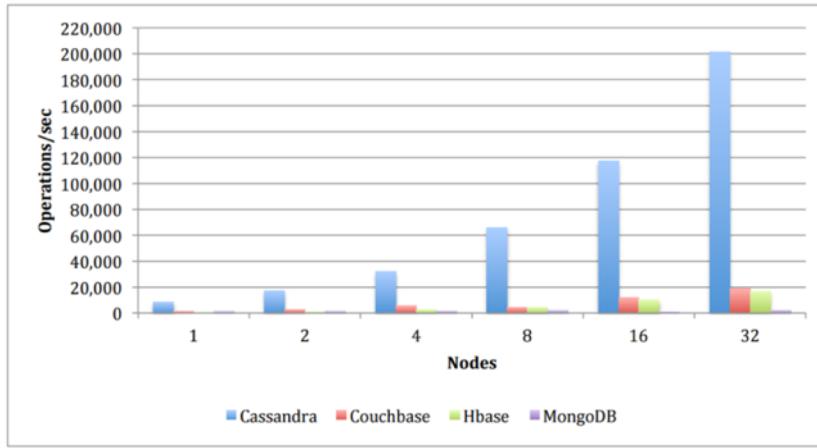
# Understanding Apache Cassandra™ Use Cases

# Apache Cassandra™ = Distributed NoSQL Database

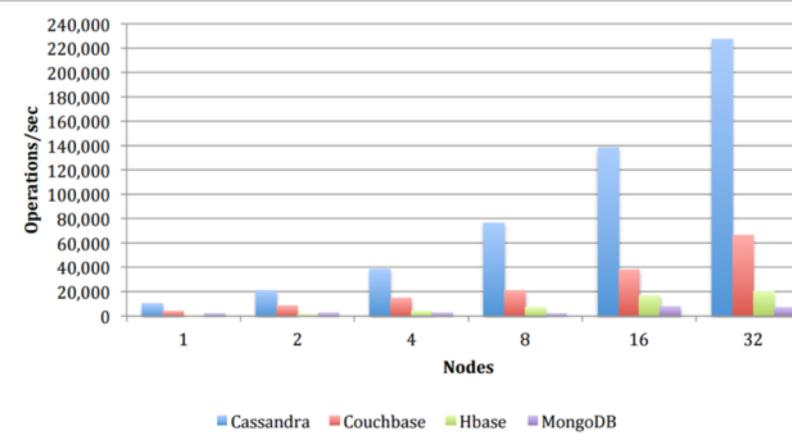


# Linear Scalability

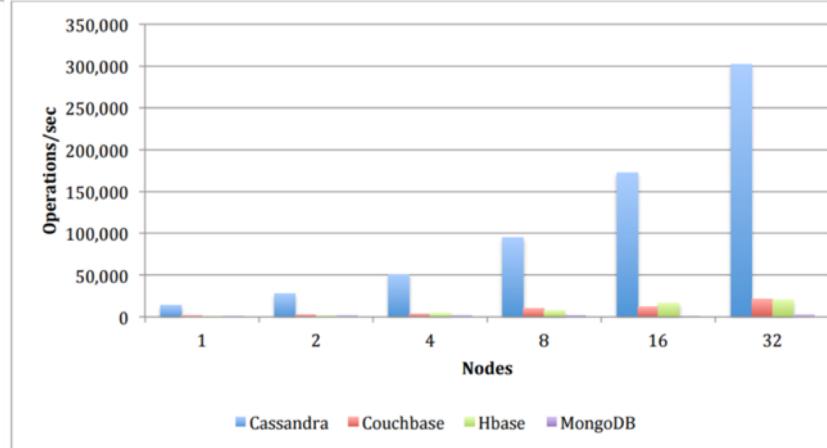
Read-Modify-Write Workload



Read-mostly Workload



Balanced Read/Write Mix



- Need More Capacity ? → Add new nodes
- Need more Throughput ? → Add new nodes

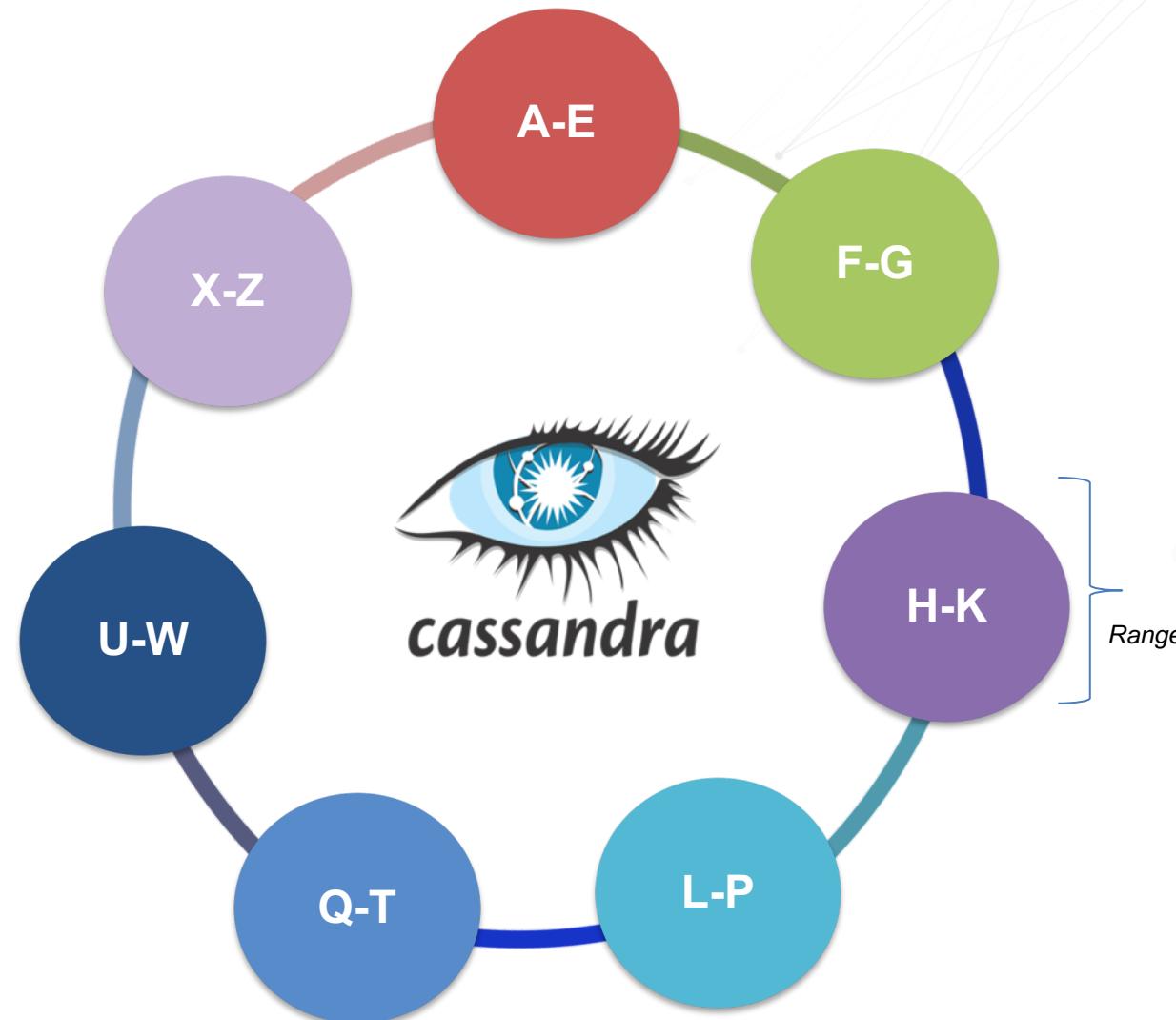
# Data is Distributed



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

Partition Key

# Data is Distributed



# Data is *Evenly-distributed*

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

Partitionner  
*Hashing Function*

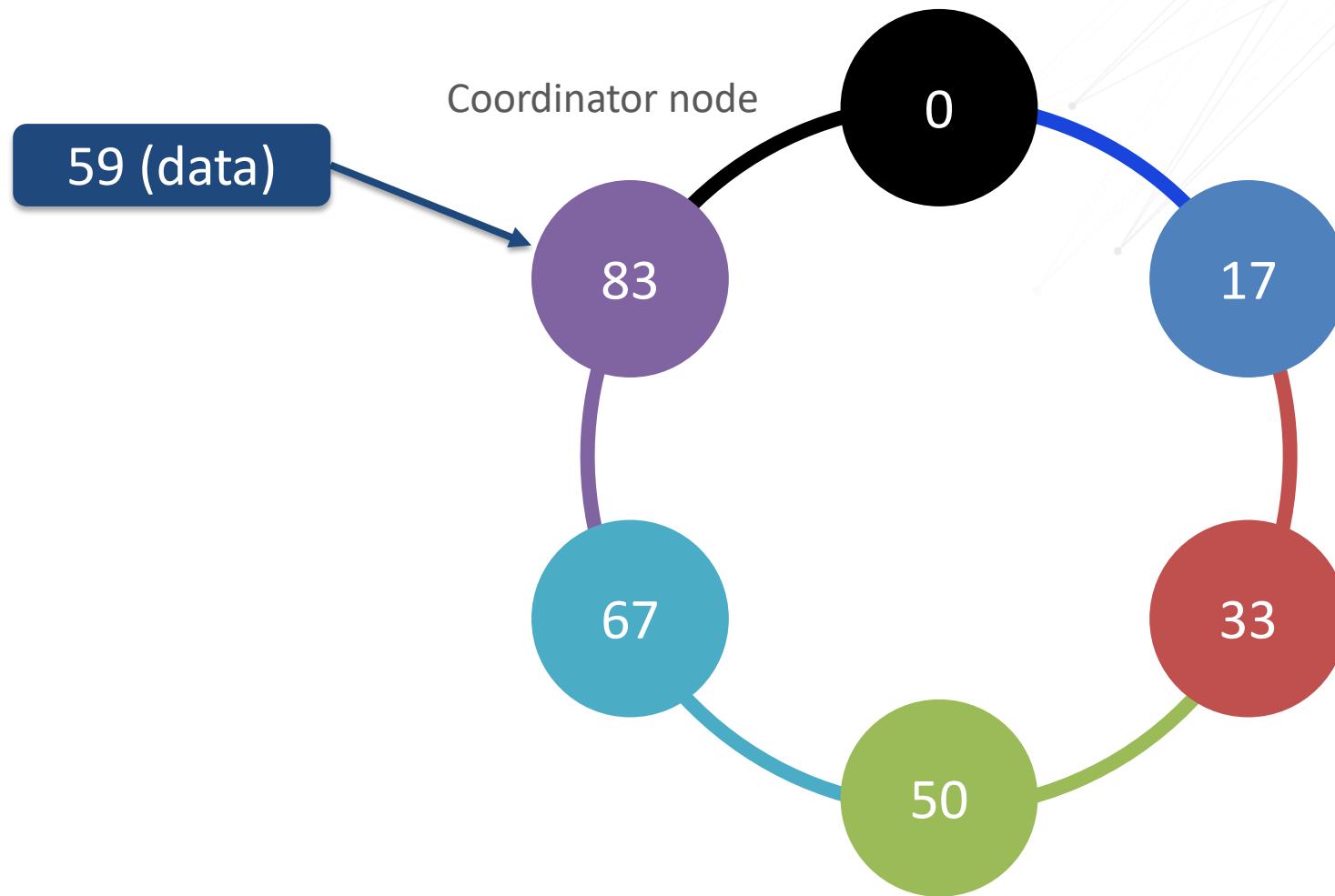
CO	City	Habitant
59	Sydney	4.900.000
12	Toronto	6.200.000
12	Montreal	4.200.000
45	Berlin	3.350.000
45	Nuremberg	500.000

Partition Key

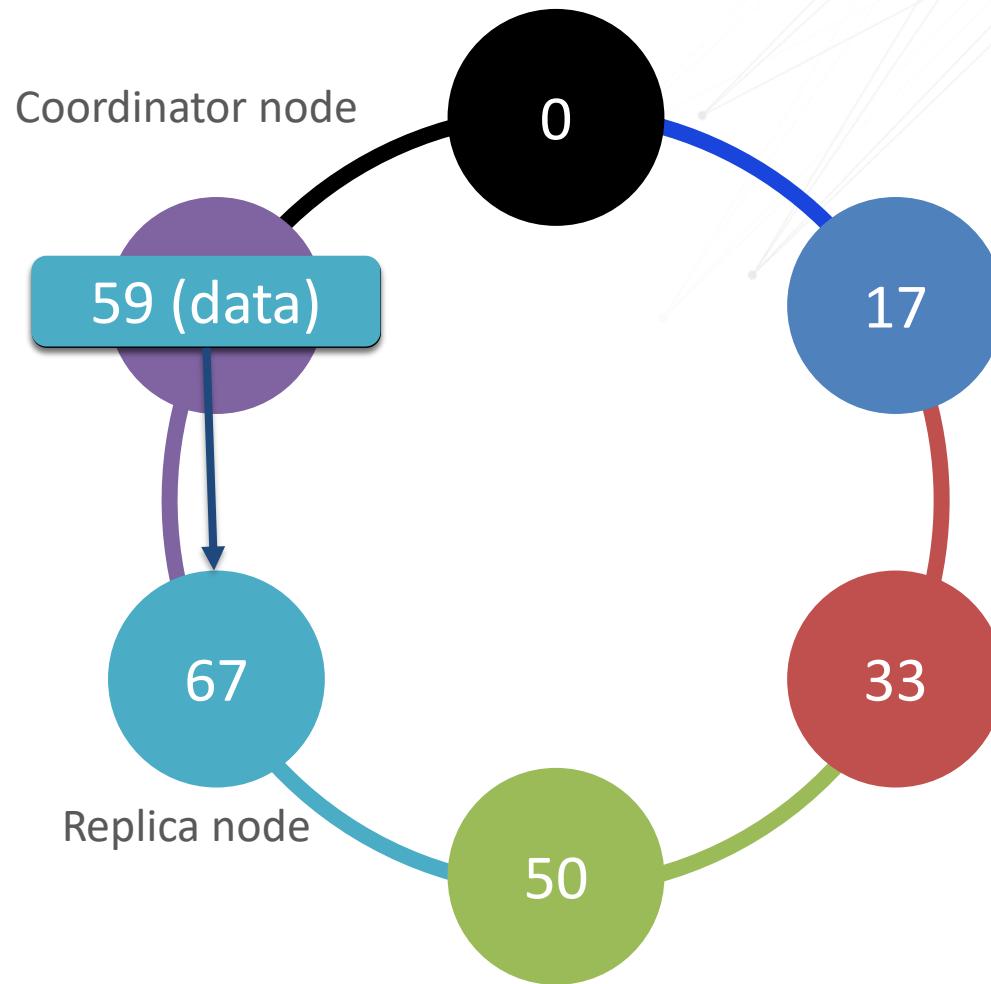
Tokens

A-E

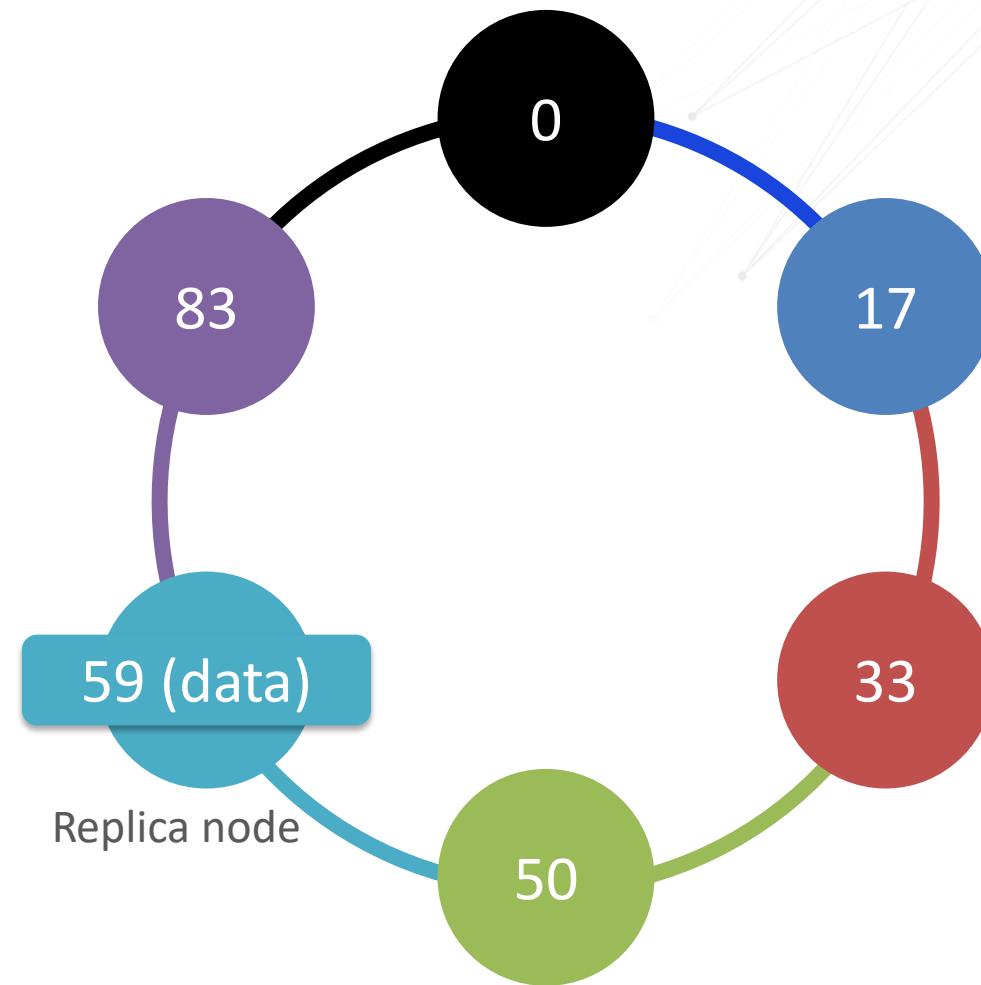
# How the Ring Works



# How the Ring Works

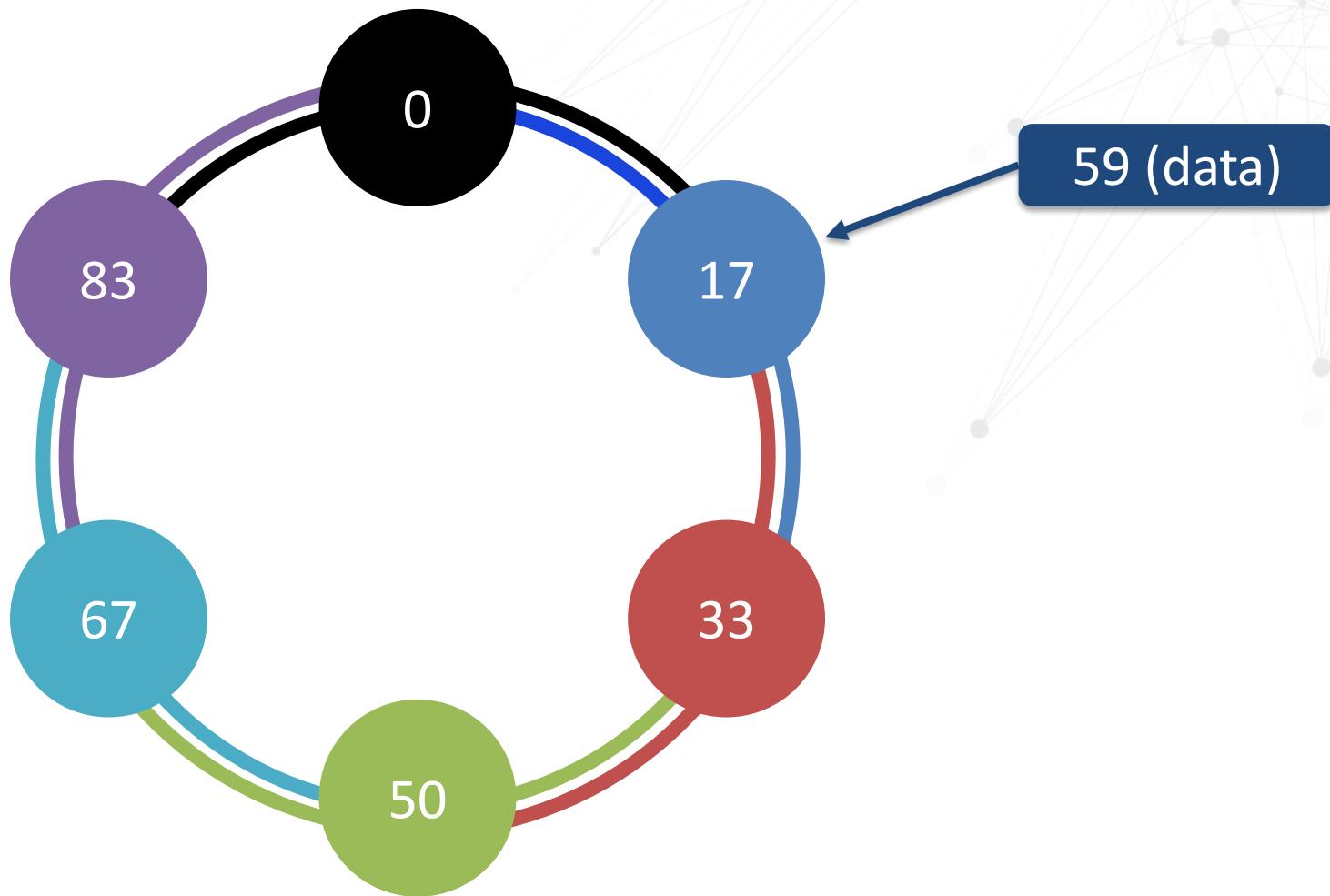


# How the Ring Works



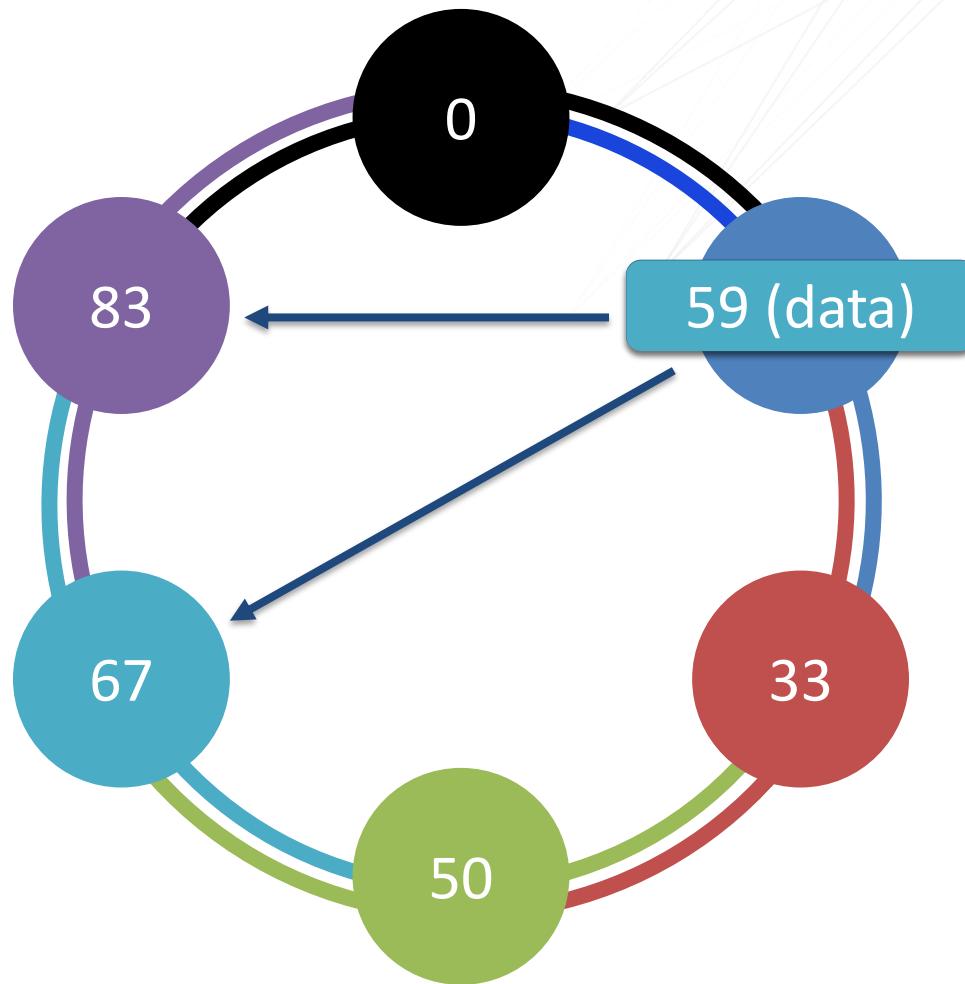
## Replication within the Ring

RF = 2



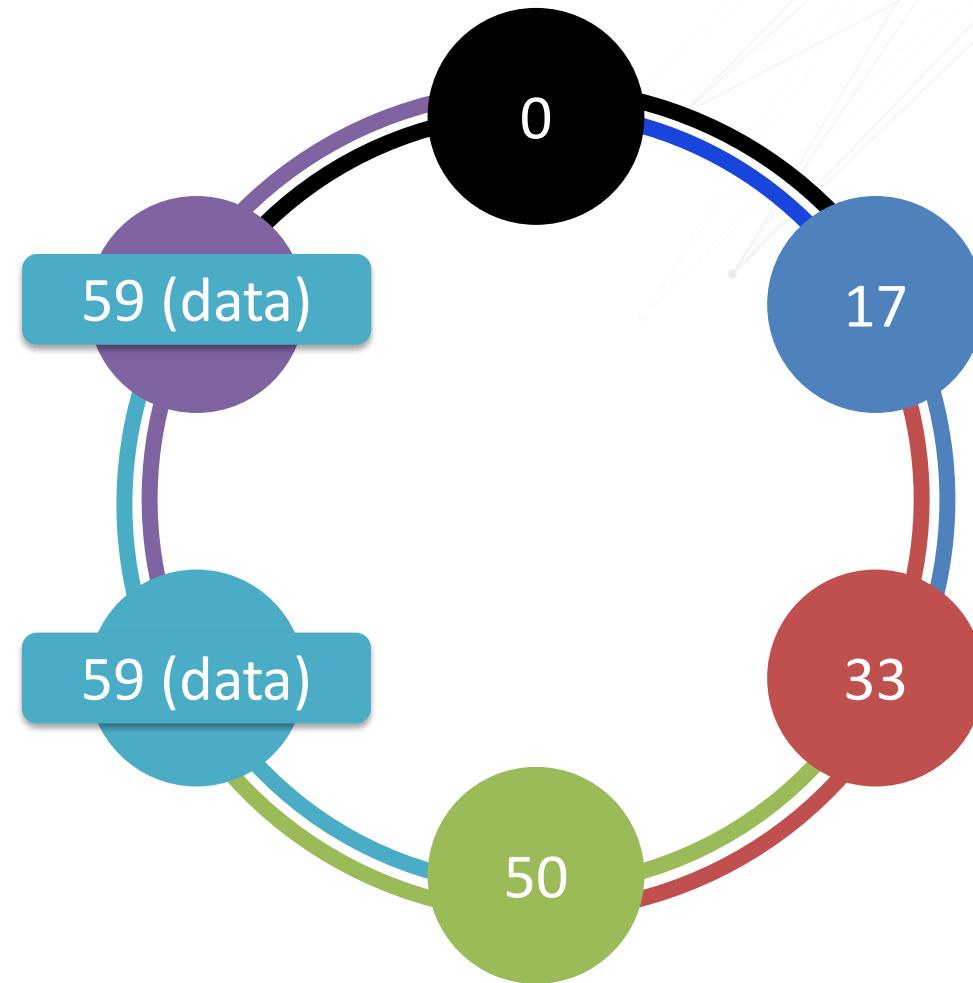
# Replication within the Ring

RF = 2



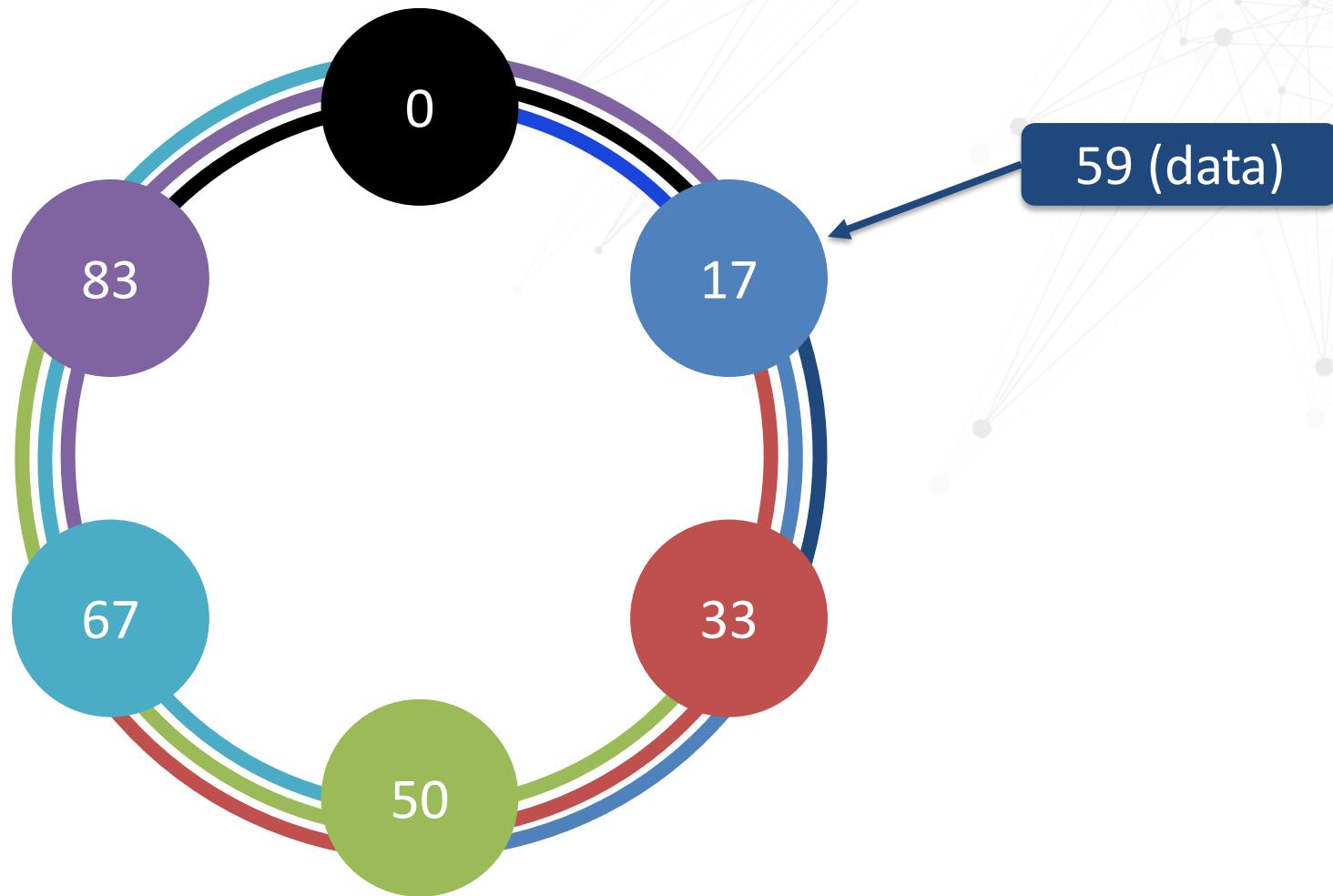
# Replication within the Ring

RF = 2



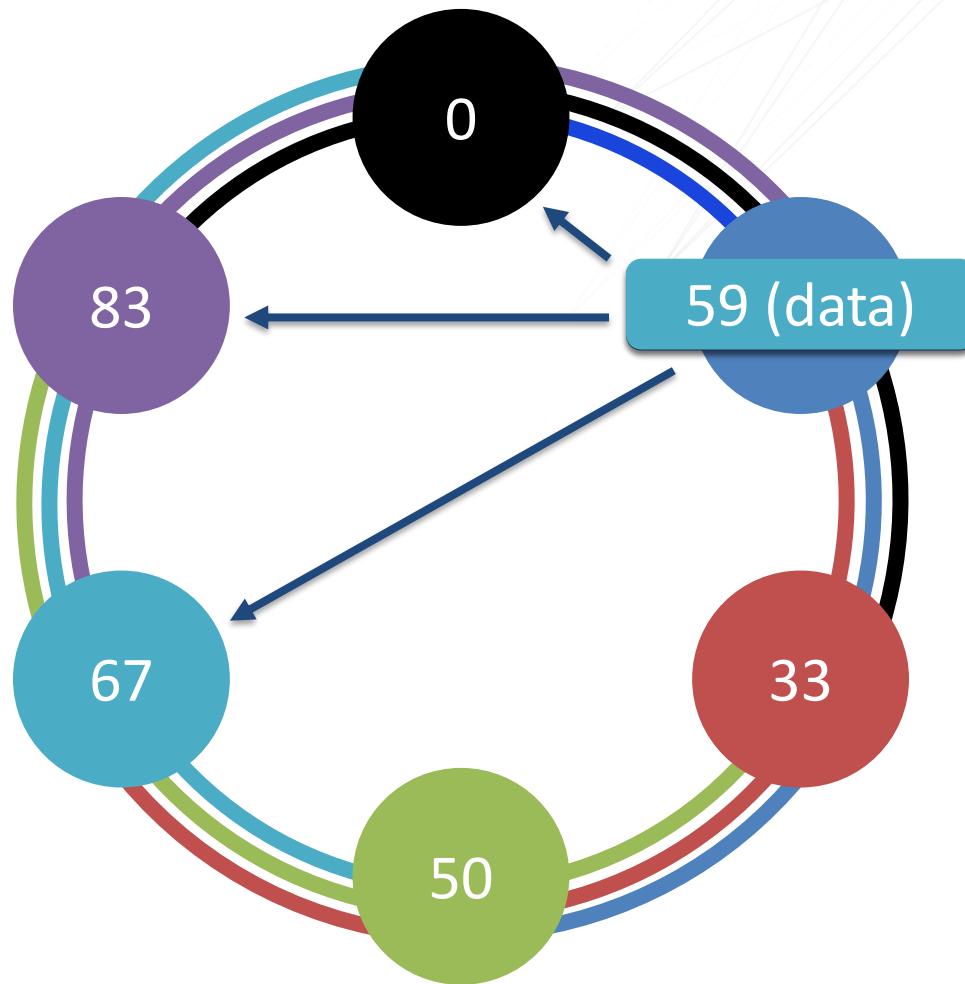
## Replication within the Ring

RF = 3



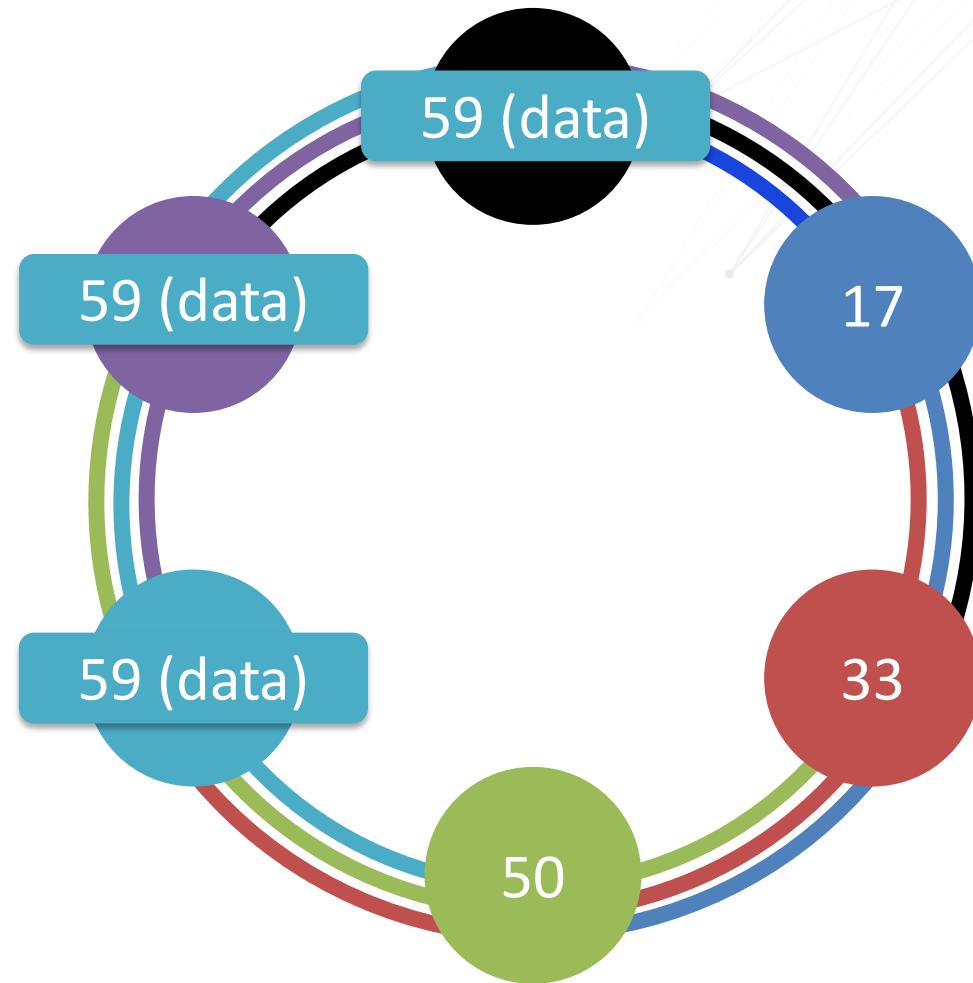
# Replication within the Ring

RF = 3



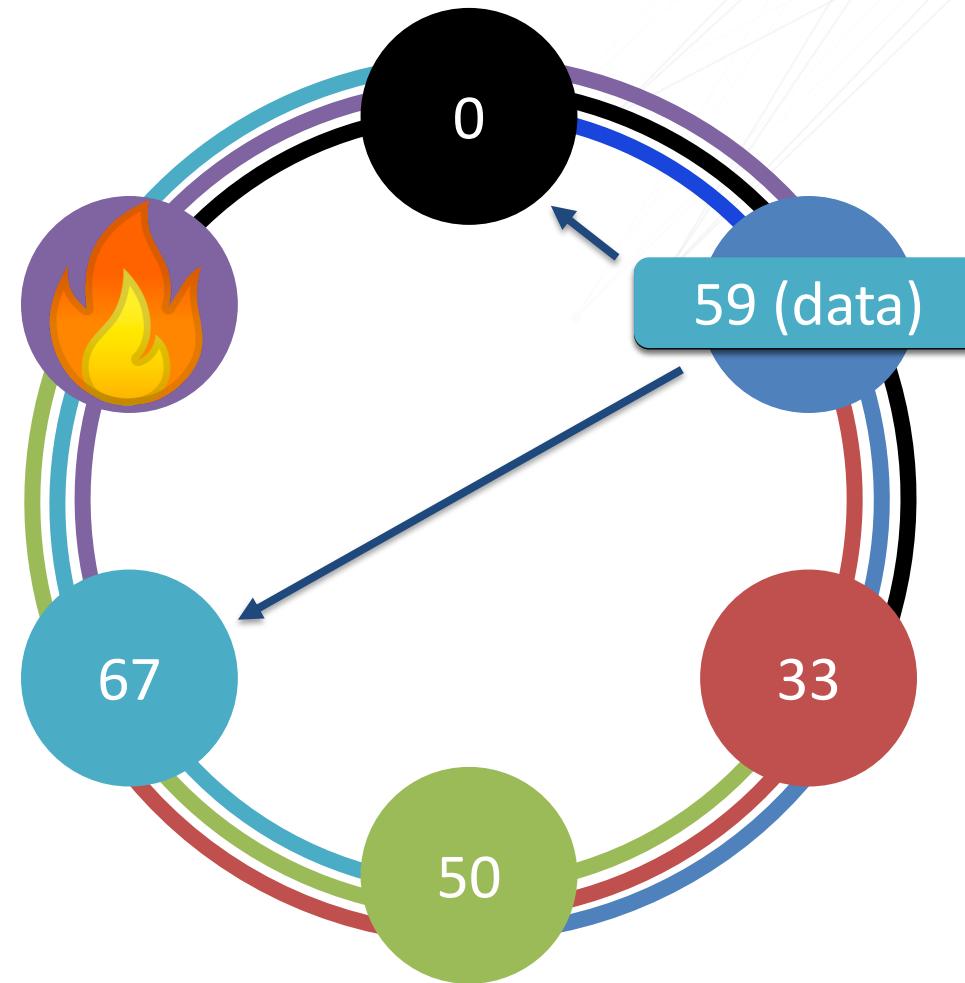
# Replication within the Ring

RF = 3



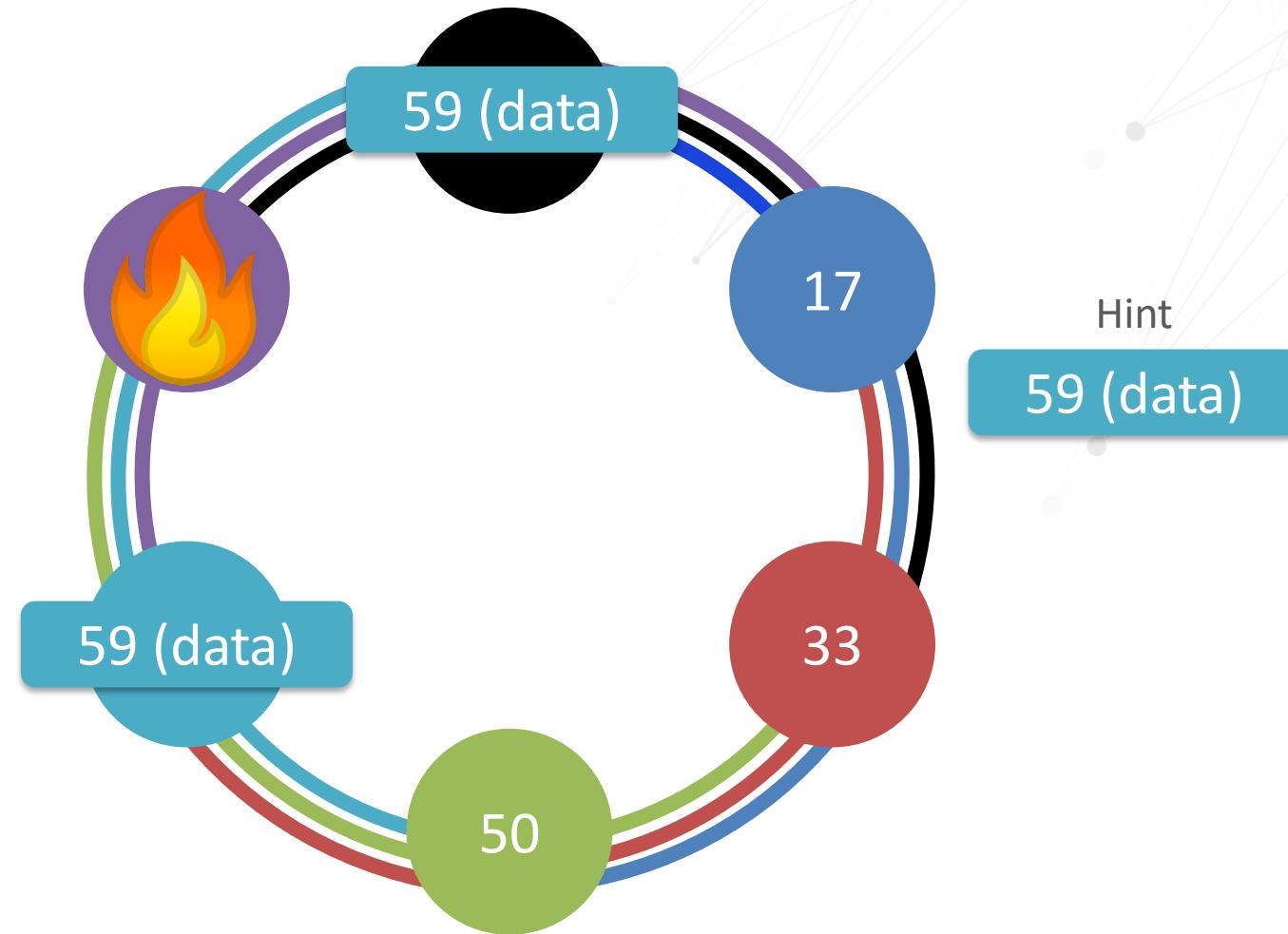
# Node Failure

RF = 3



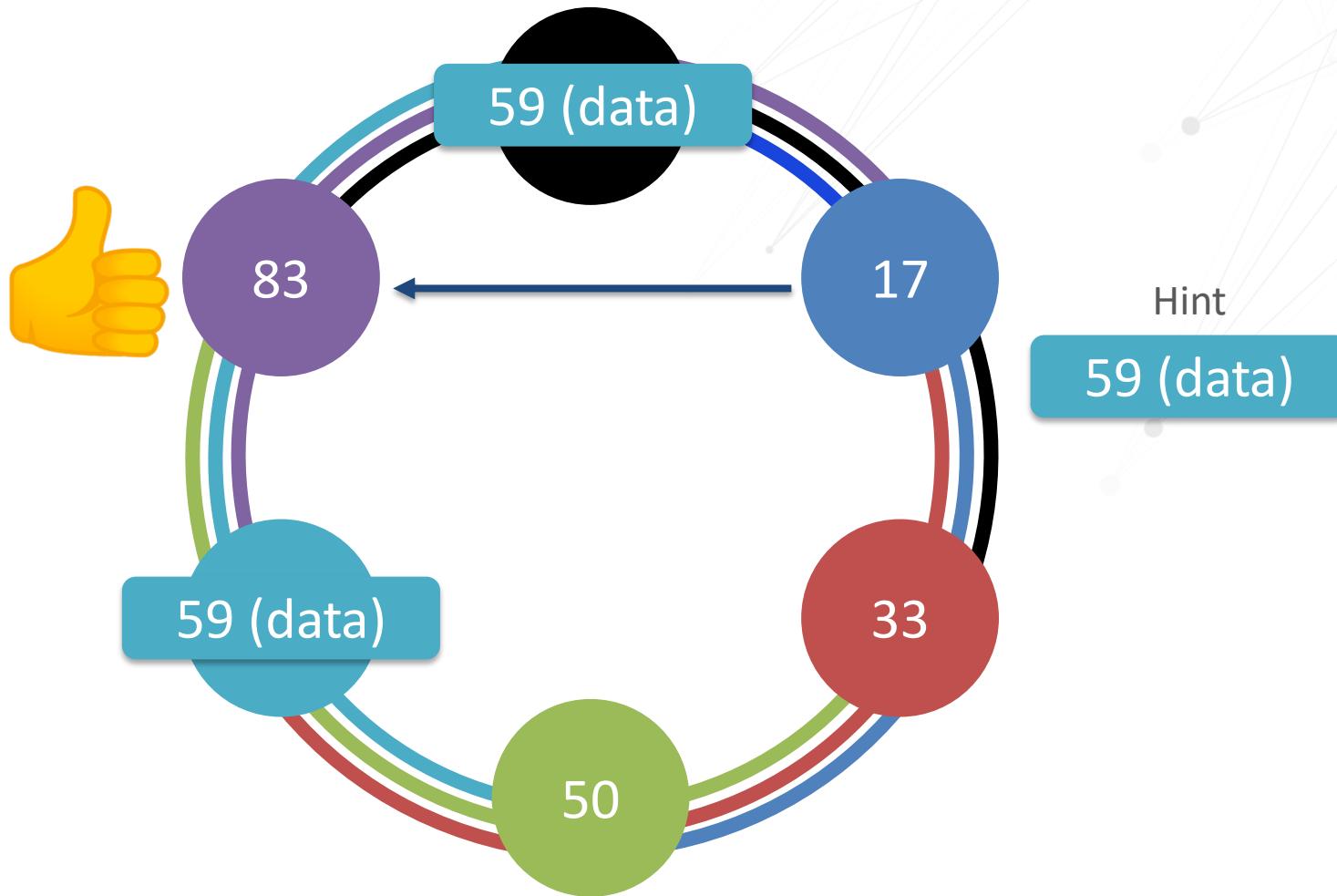
# Node Failure

RF = 3



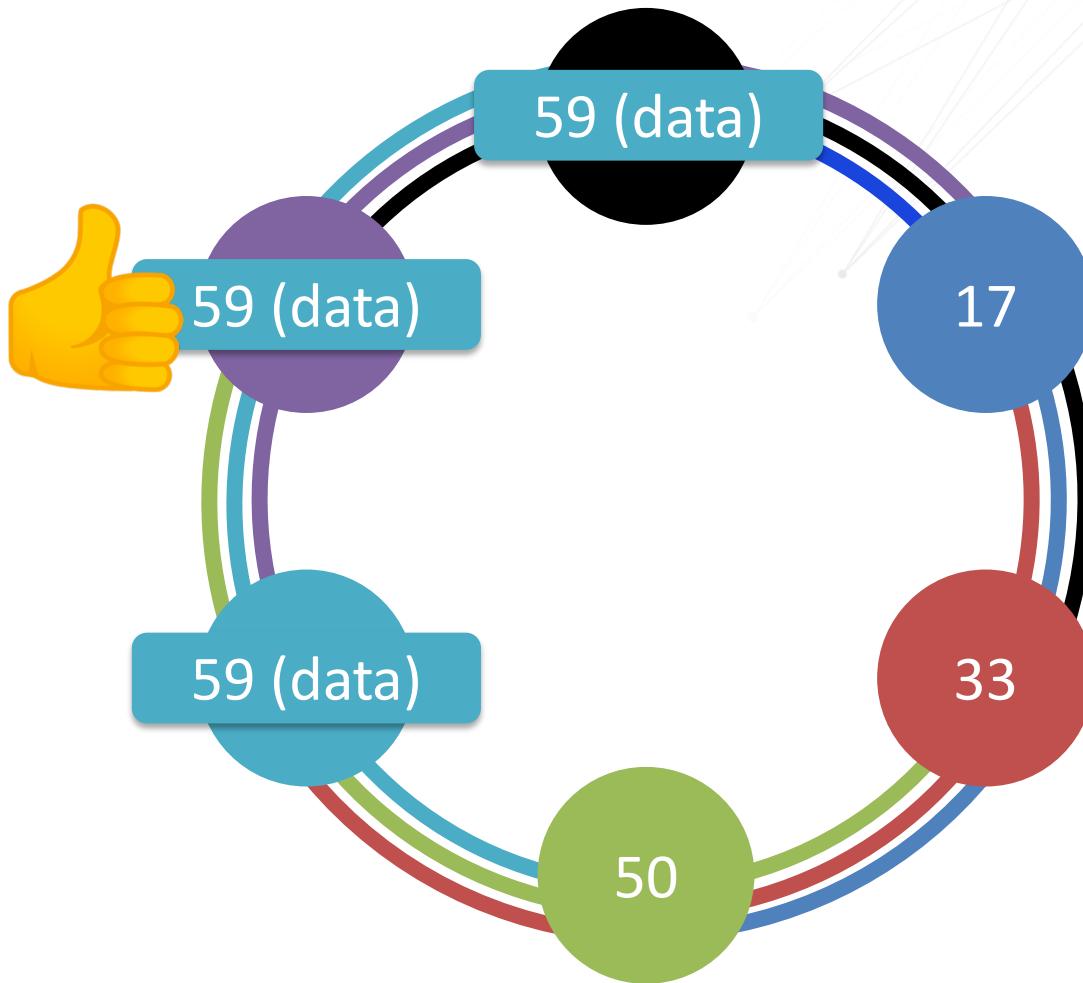
# Node Failure

RF = 3

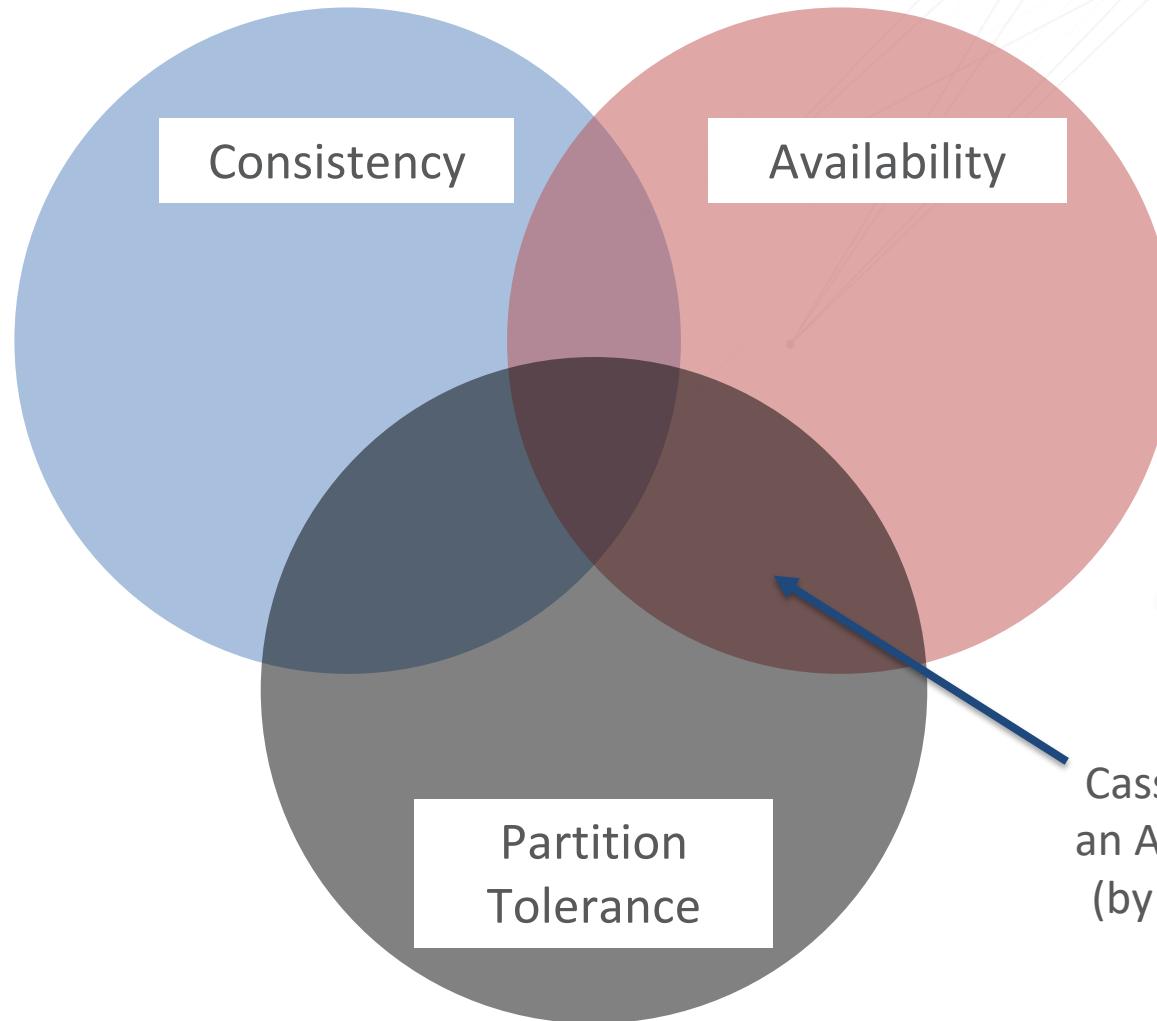


# Node Failure – Recovered!

RF = 3

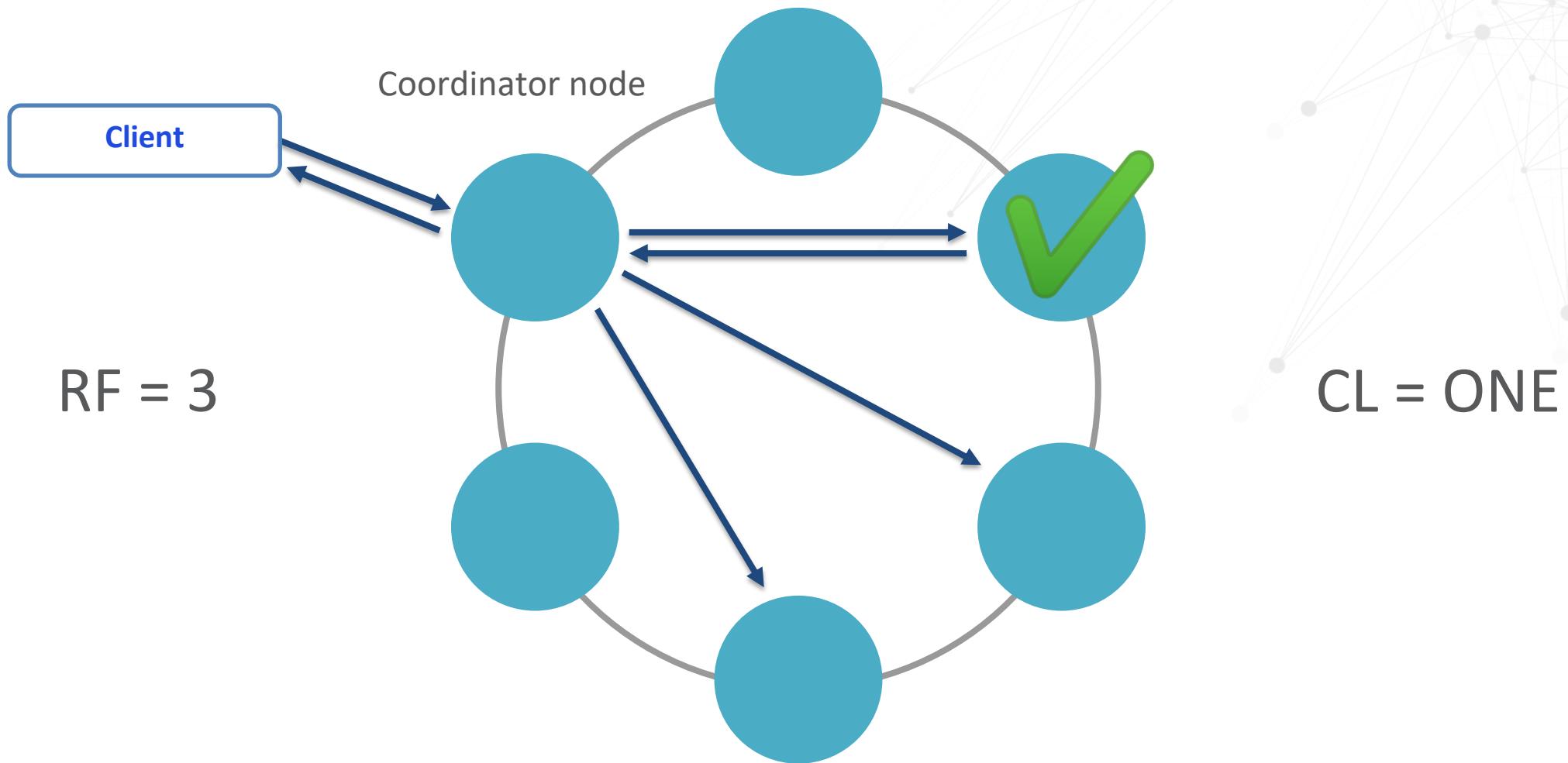


# CAP Theorem

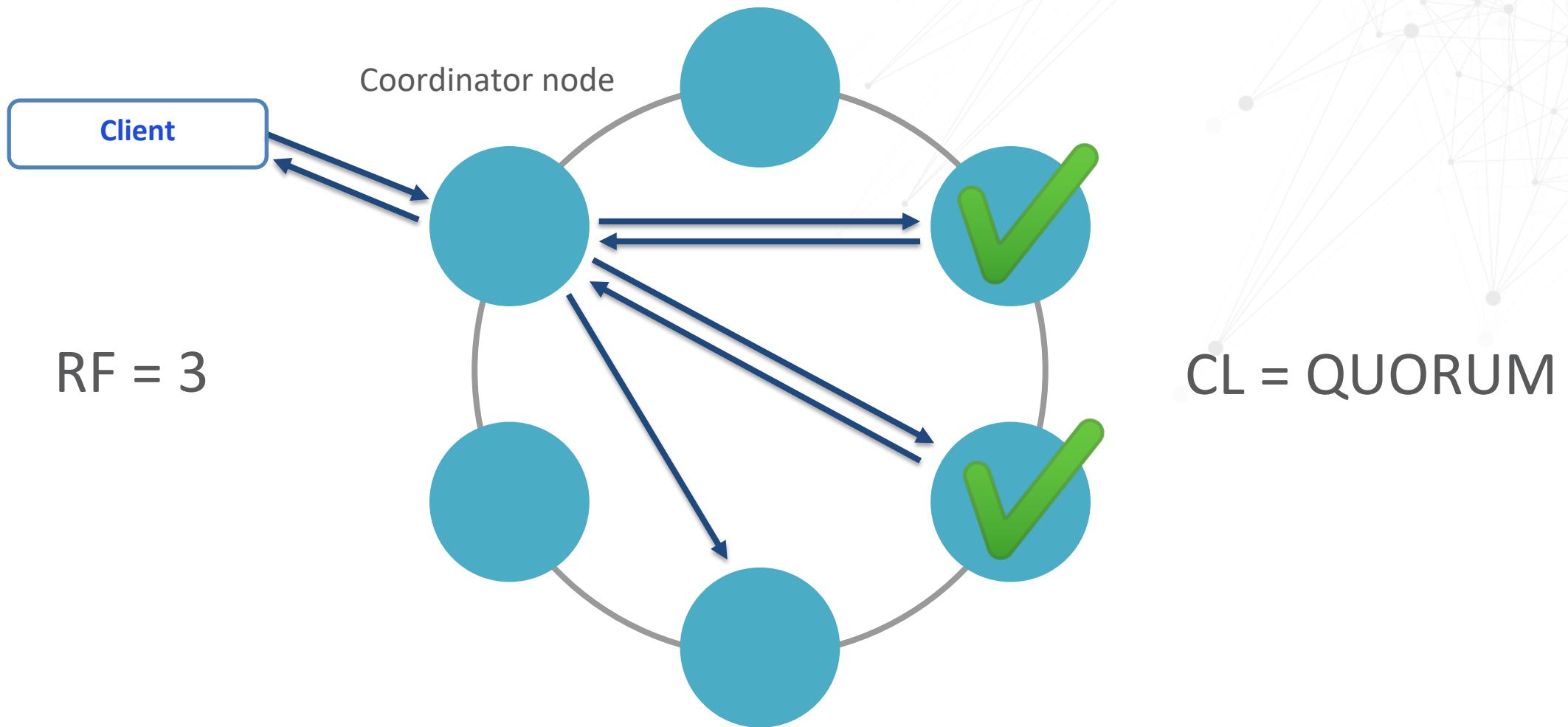


Cassandra is  
an AP system  
(by default)

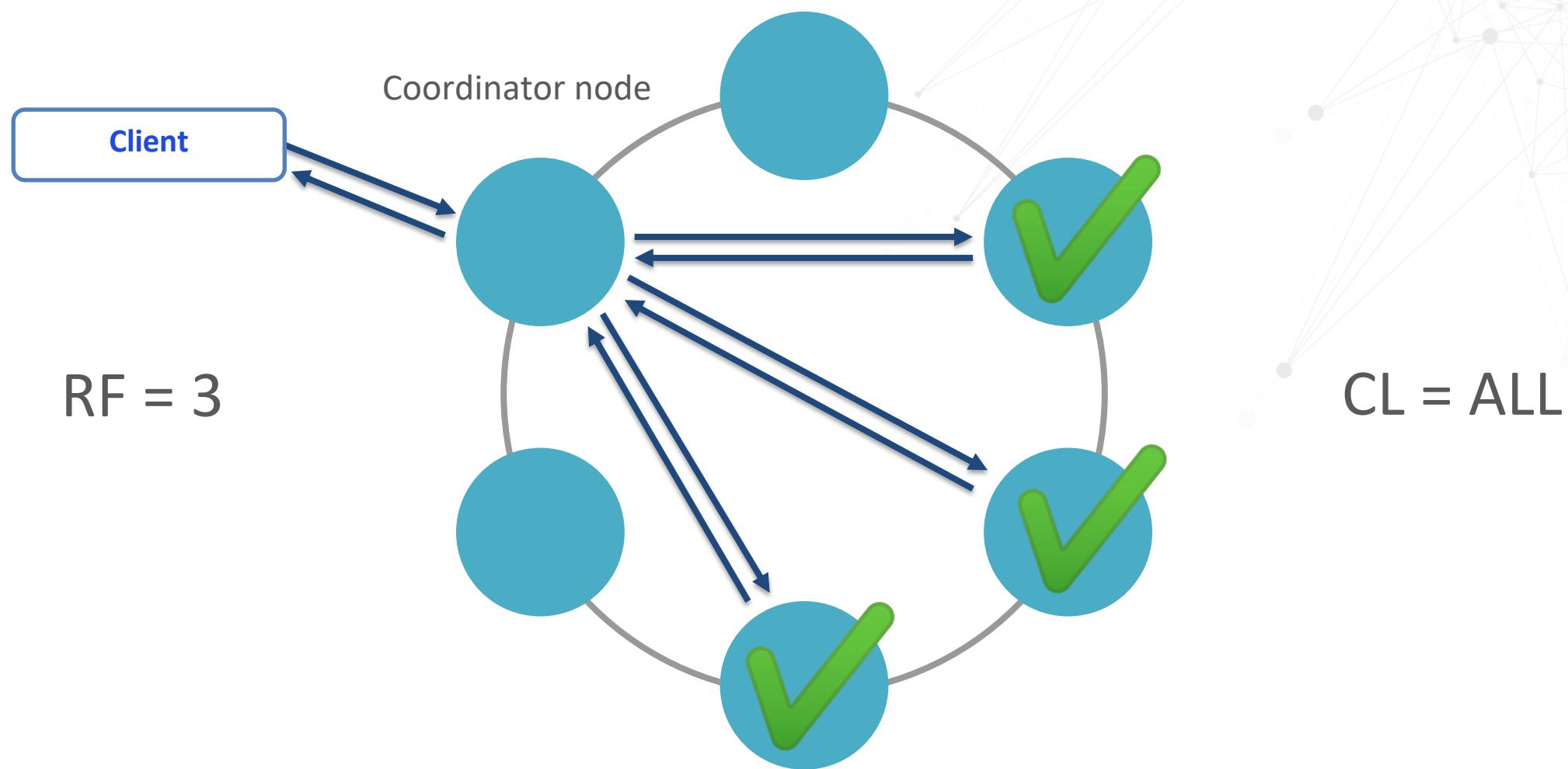
# Consistency Levels



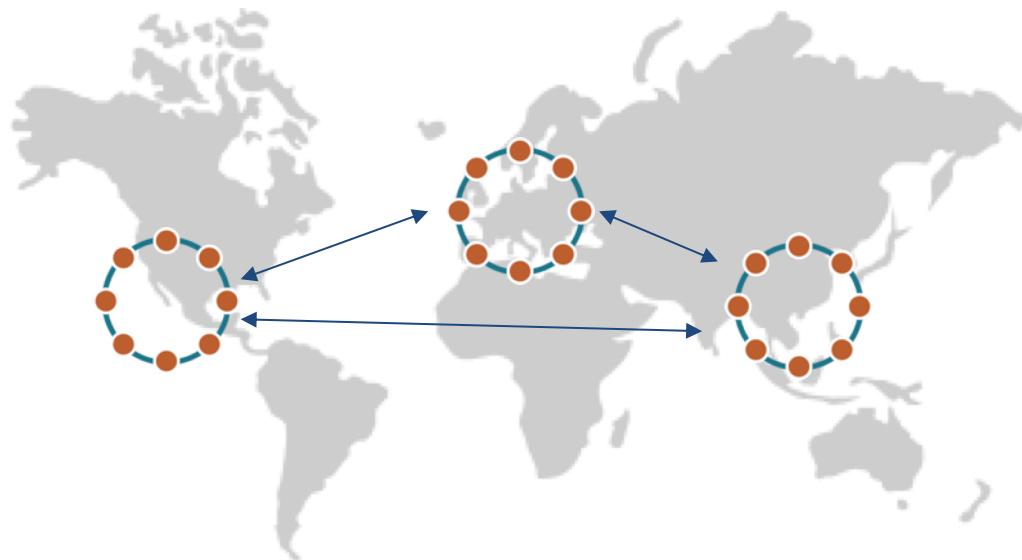
# Consistency Levels



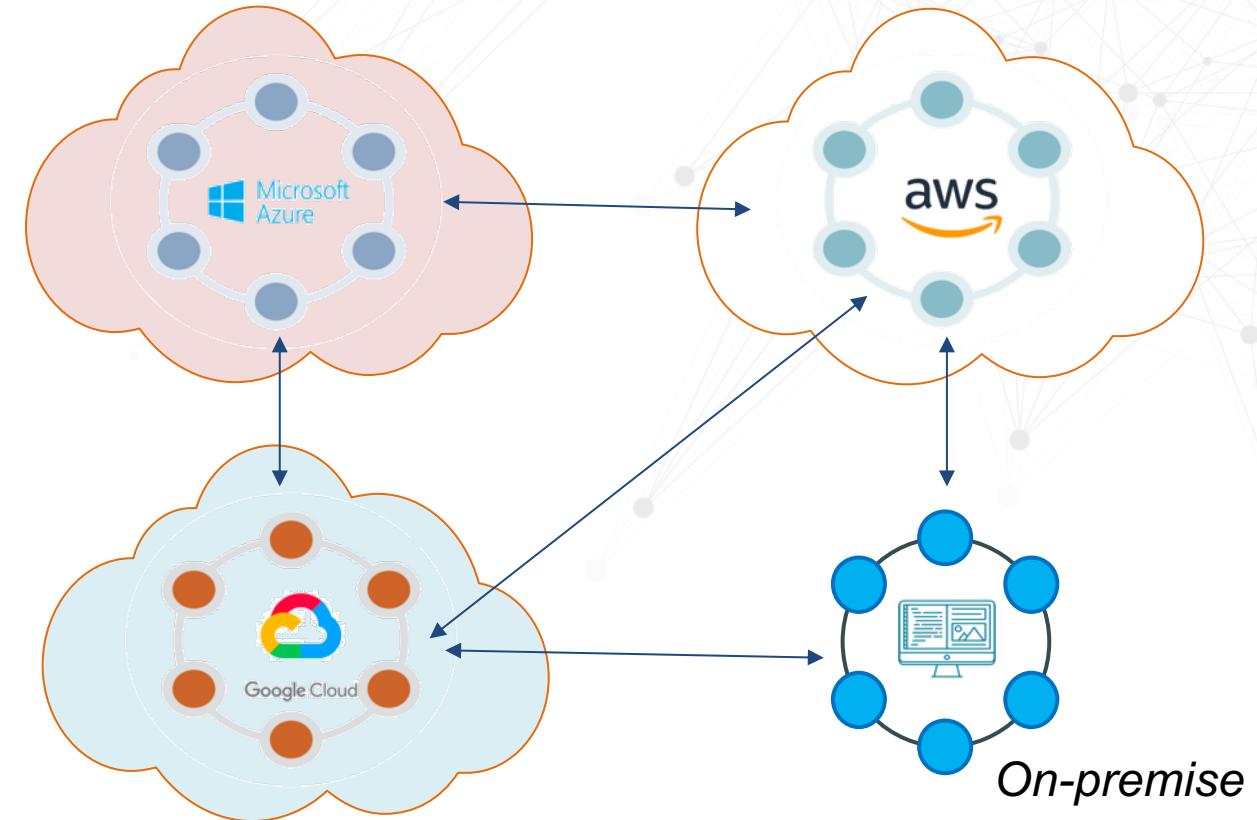
# Consistency Levels



# Data Distributed Everywhere



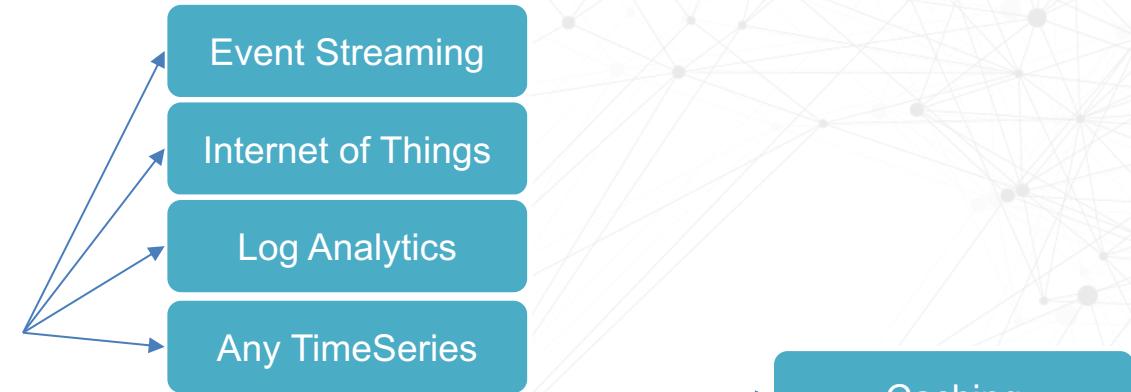
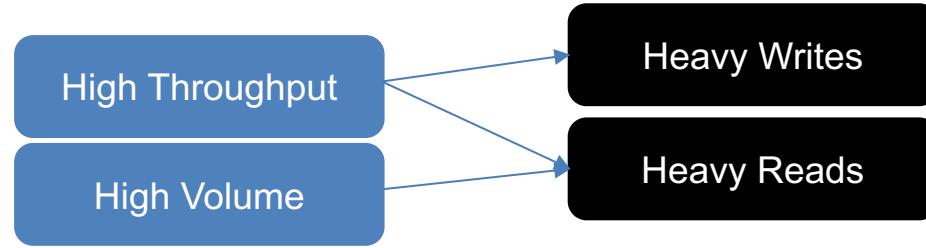
GEOGRAPHICALLY



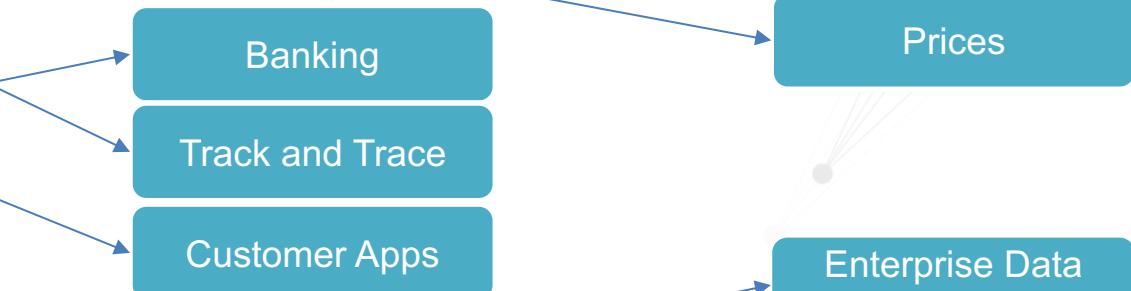
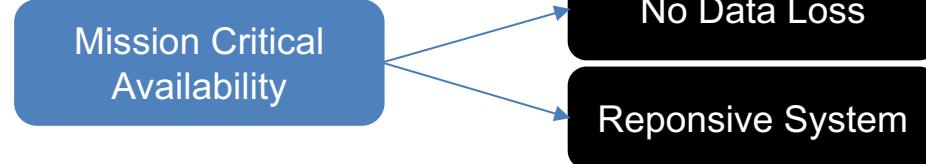
HYBRID- MULTI CLOUD

# Understanding Use Cases

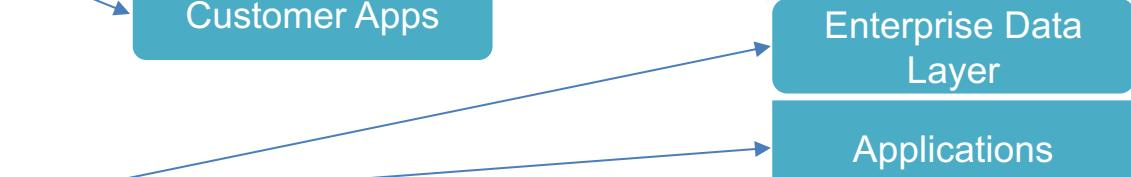
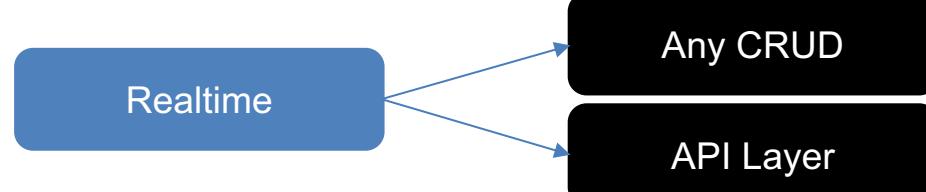
**S**



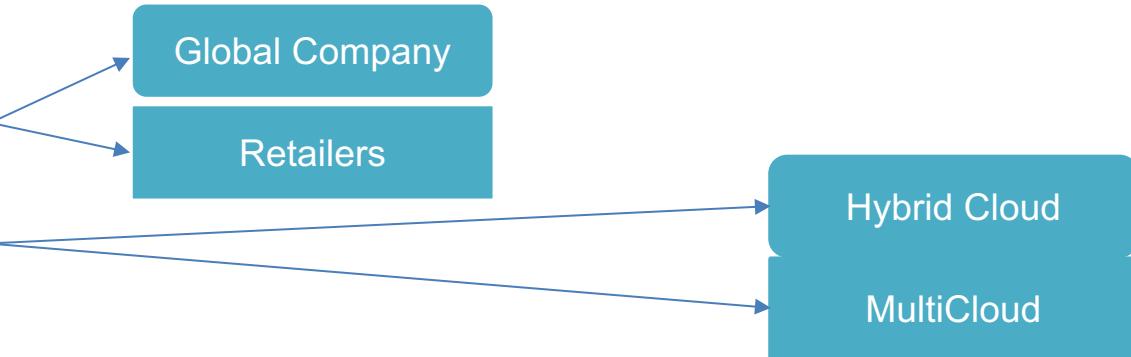
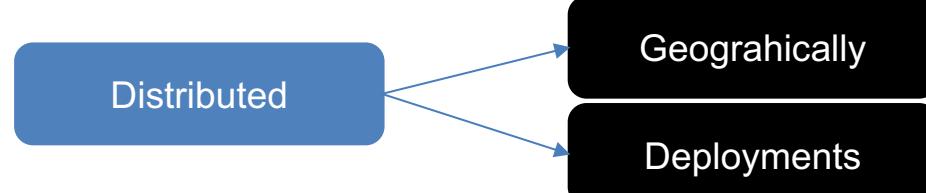
**A**



**R**



**D**



# Agenda

0

Bootstraping

1

Understanding Apache Cassandra™ Use Cases

2

Data Modelling with Apache Cassandra™

3

Application Development

4

What's NEXT ?

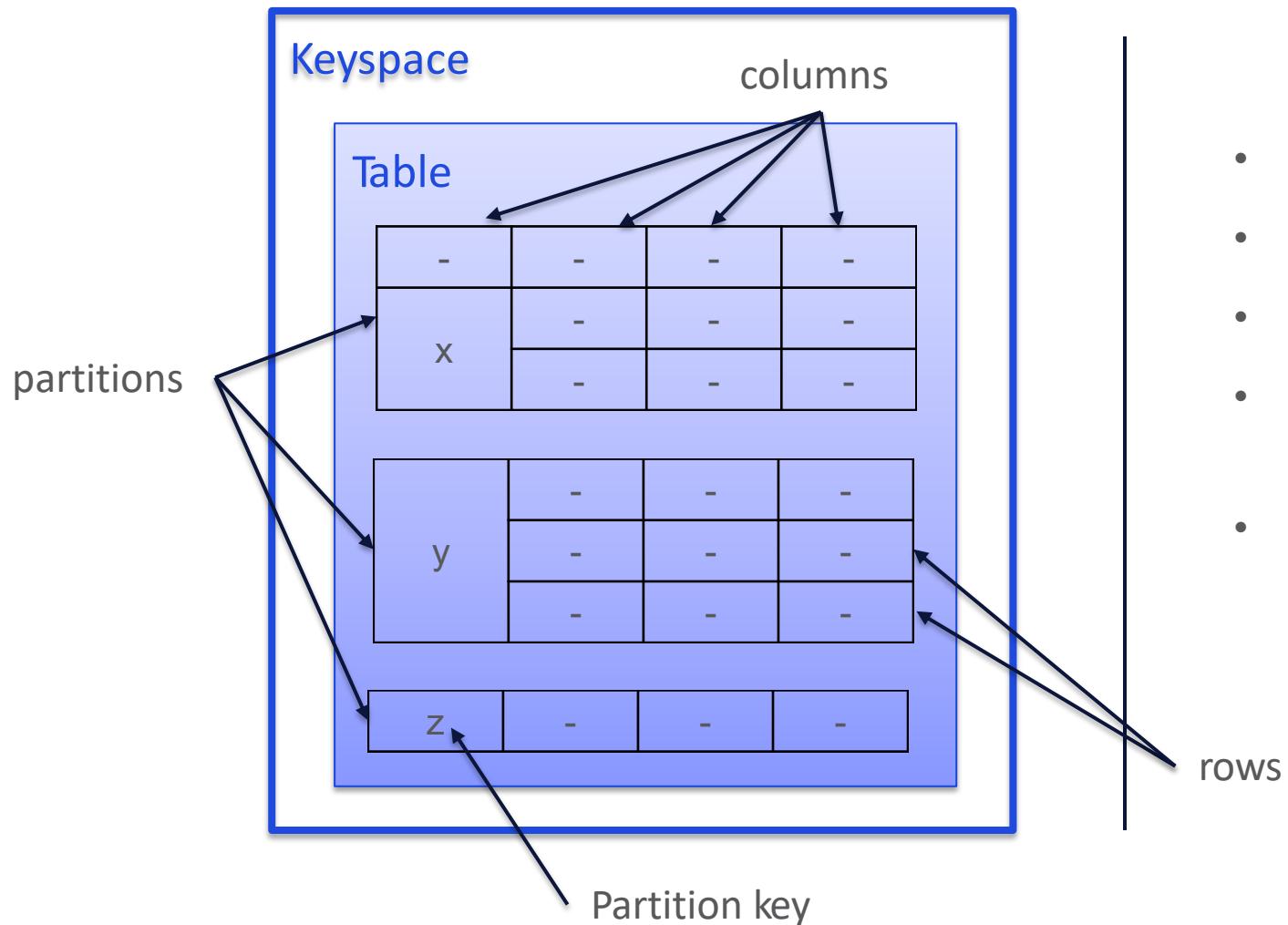
# Meetup @Stockholm

---



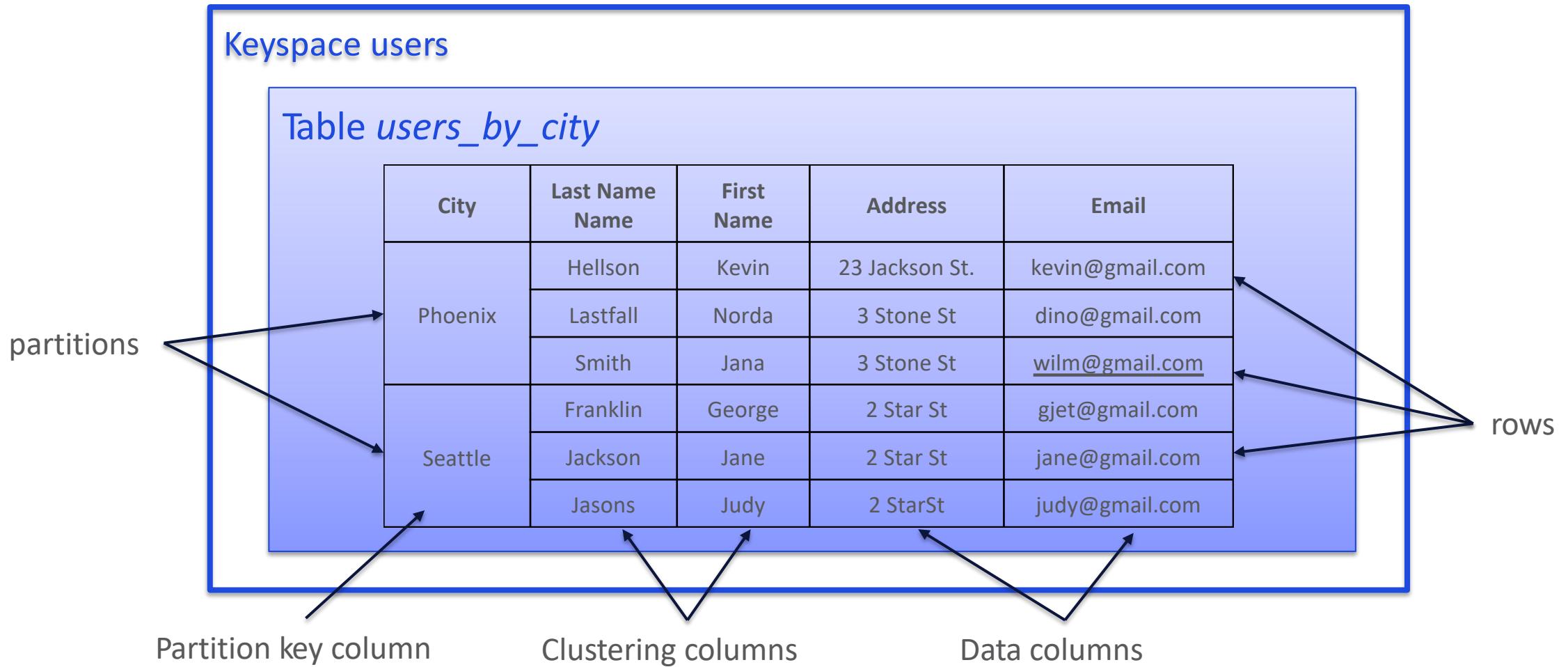
# Data Modelling with Apache Cassandra™

# 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 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 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 Name	First Name	Address	Email
Charlotte	Azrael	Chris	5 Blue St	chris@gmail.com
	Stilson	Brainy	7 Azure In	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	---	---	---	---
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 KEYSPACE users] --> B[keyspace]; A --> C[replication strategy]; A --> D[Replication factor by data center];
```

# Creating a Table in CQL

```
CREATE TABLE users.users_by_city (
    city text,
    last_name text,
    first_name text,
    address text,
    email text,
    PRIMARY KEY ((city), last_name, first_name));
```

The diagram illustrates the structure of a CQL CREATE TABLE statement with the following annotations:

- keyspace**: Points to the schema identifier "users".
- table**: Points to the table identifier "users\_by\_city".
- column definitions**: Points to the list of columns: "city", "last\_name", "first\_name", "address", and "email".
- Primary key**: Points to the start of the PRIMARY KEY clause.
- Partition key**: Points to the column "city" which is part of the composite primary key.
- Clustering columns**: Points to the columns "last\_name" and "first\_name" which are part of the composite primary key.

# Time for exercises

---



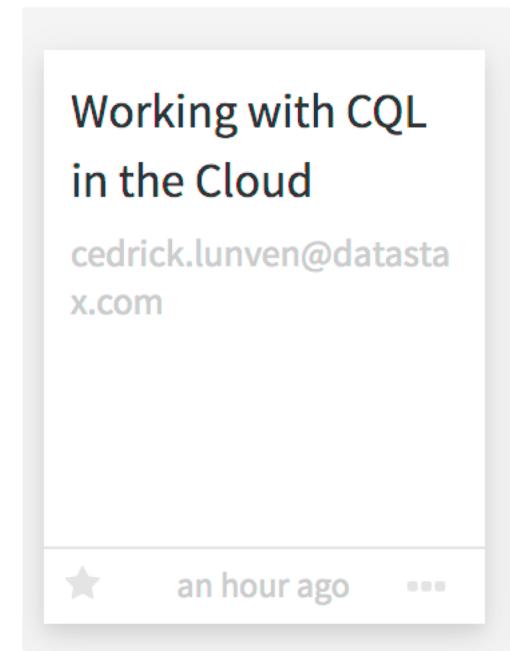
## DataStax Studio



# #1 – Working with CQL in the Cloud

---

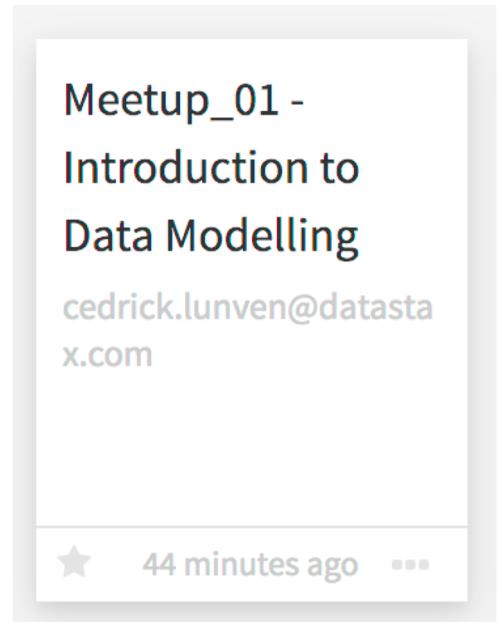
- Check that your Database is UP in Apollo
- Open DataStax Studio
- Navigate to « **Working with CQL in the Cloud** »
- Play with the Notebook



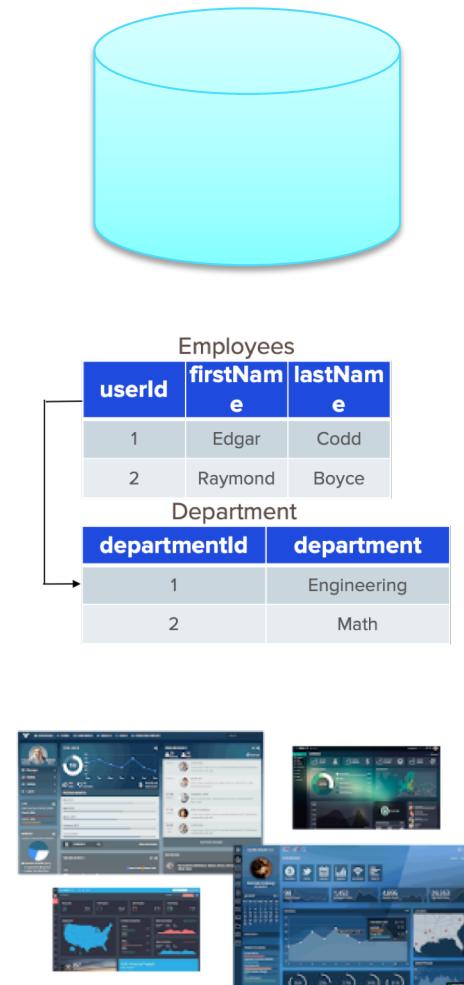
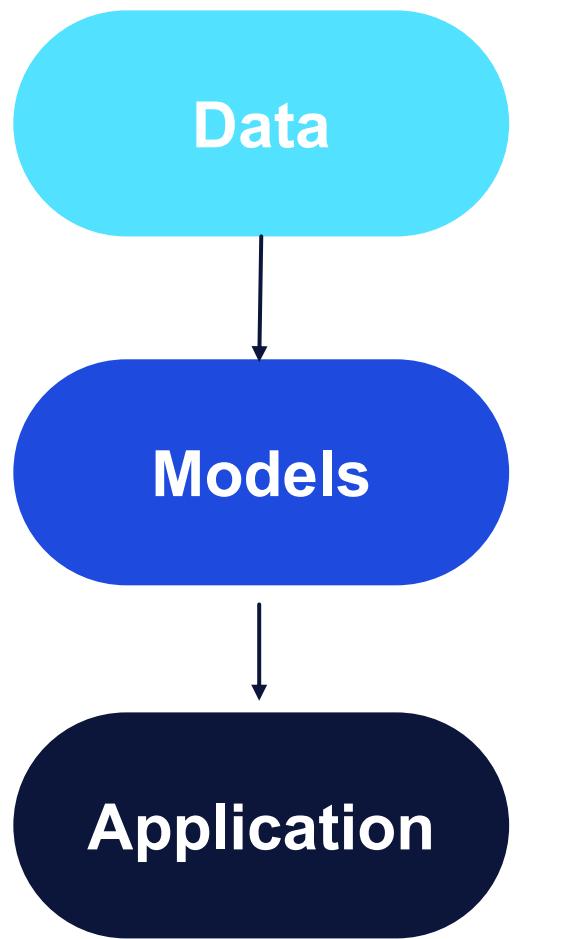
## #2 – Introduction to Data Modelling

---

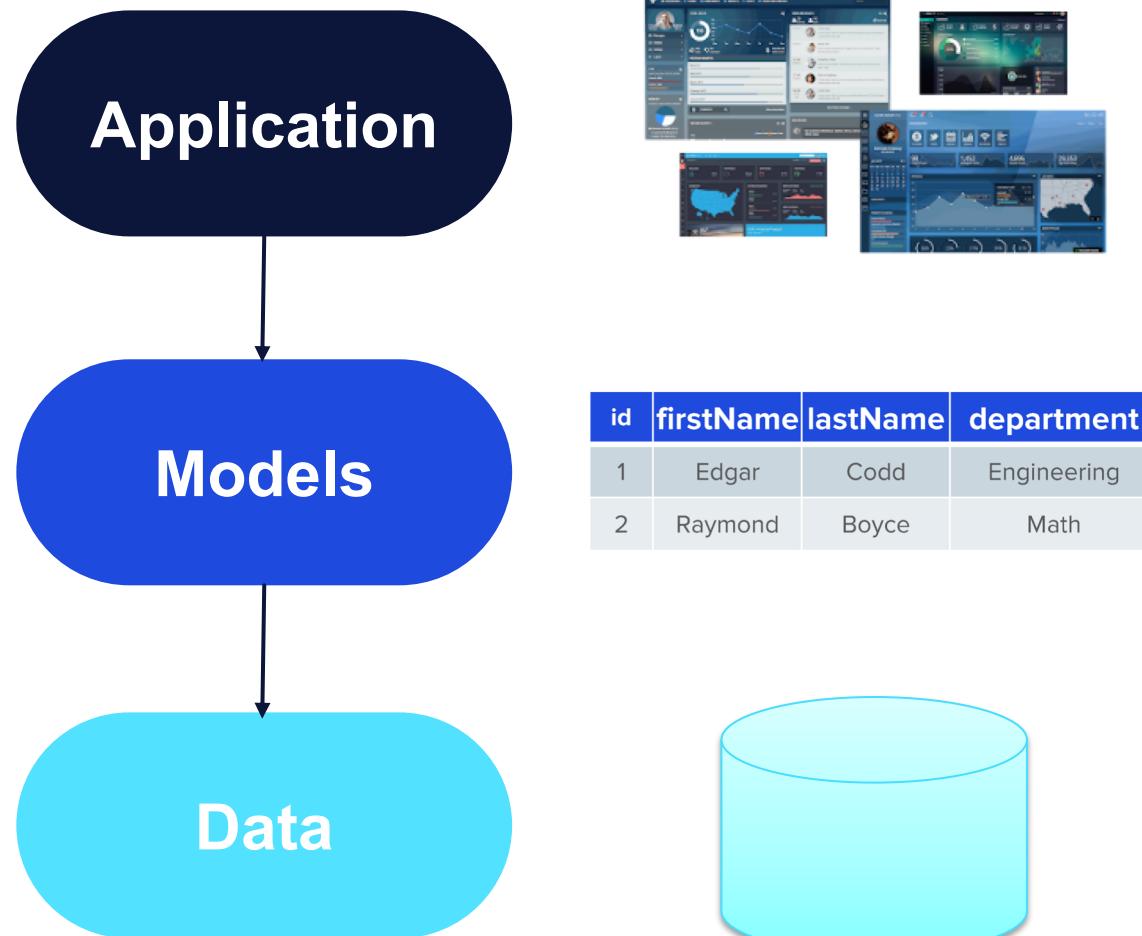
- `git clone https://github.com/clun/meetup_zero2Cloud.git`
- Locate the "notebooks" the folder
- Drag and drop « **Meetup\_01 - Introduction to Data Modelling** » in the Studio
- Execute the exercises in the Notebook



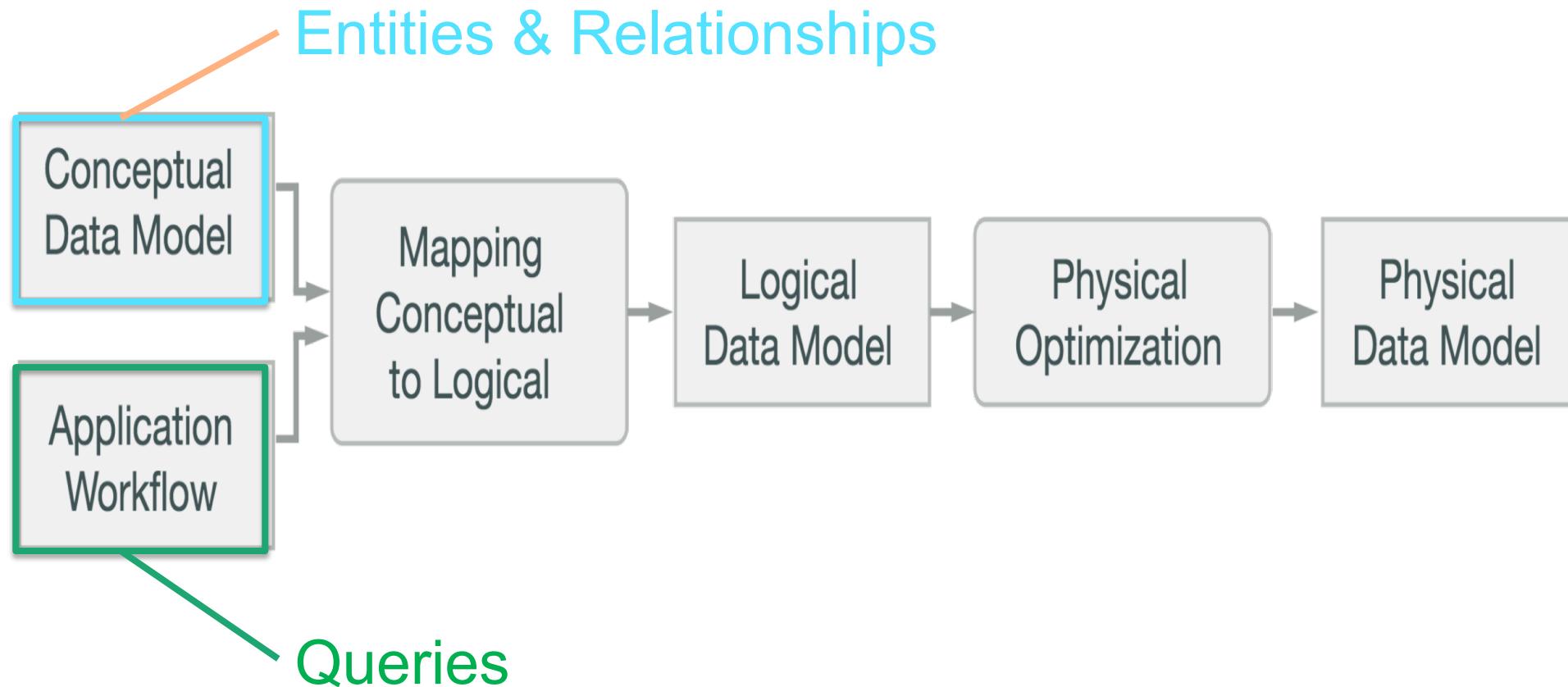
# Relational Modeling



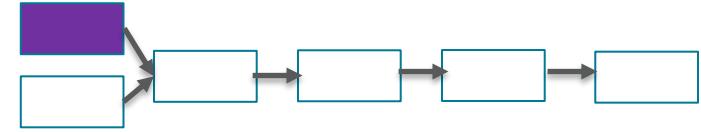
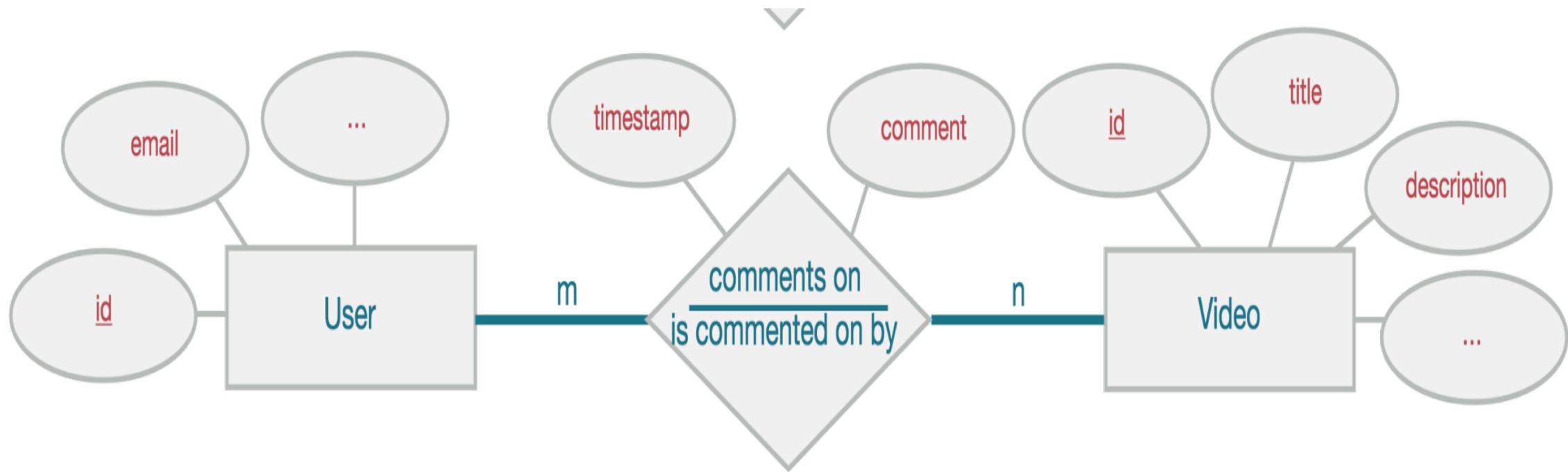
# Cassandra Modeling



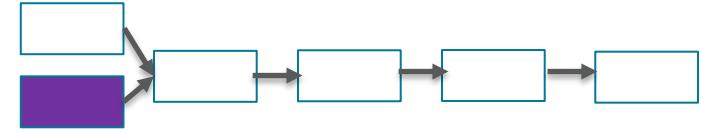
# Designing Data Model



# Conceptual Data Model



# Application Workflow



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

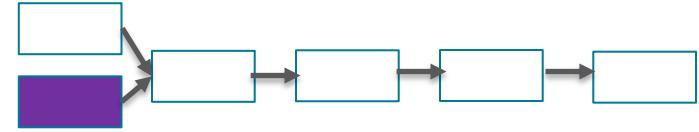
- Get most recent first
- Implement Paging

**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)

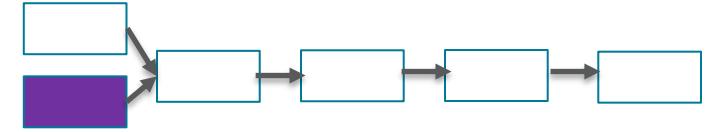


**Q2:** Find comments posted for a user with a known id (show most recent first)



**Q3:** CRUD Operations

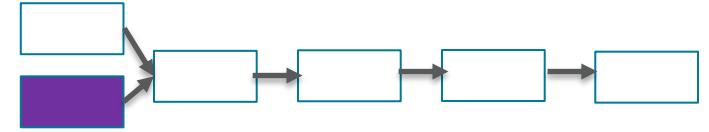
# Logical Data Model



comments_by_user	
userid	K
creationdate	C↓
commentid	C↑
videoid	
comment	

comments_by_video	
videoid	K
creationdate	C↓
commentid	C↑
userid	
comment	

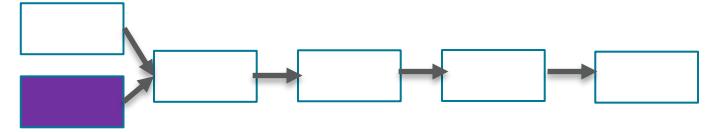
# Physical Data Model



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,
    videooid uuid,
    comment text,
    PRIMARY KEY ((userid), commentid)
) WITH CLUSTERING ORDER BY (commentid DESC);
```

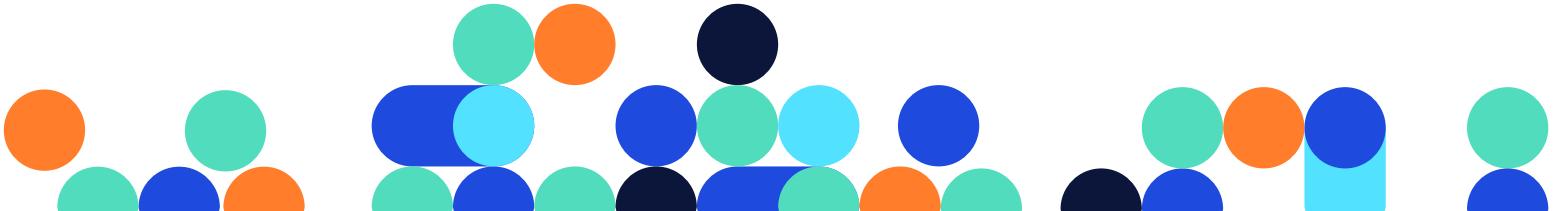
```
CREATE TABLE IF NOT EXISTS comments_by_video (
    videooid uuid,
    commentid timeuuid,
    userid uuid,
    comment text,
    PRIMARY KEY ((videooid), commentid)
) WITH CLUSTERING ORDER BY (commentid DESC);
```

# Time for an exercise!

---



## “Designing Queries” Notebook



## #3 – Cassandra Land

---

- Drag and drop « **Meetup\_02 – Cassandra-Land PART 1** » in the Studio
- Execute the exercises in the Notebook

Meetup\_02 -  
Cassandra-Land  
Project PART 1  
cedrick.lunven@datasta  
x.com

★ a few seconds ago ...

- Drag and drop « **Meetup\_03 – Cassandra-Land PART 3** » in the Studio
- Execute the exercises in the Notebook

Meetup\_03 -  
Cassandra-Land  
Project PART 2  
cedrick.lunven@datasta  
x.com

★ a few seconds ago ...

# Agenda

0

Bootstraping

1

Understanding Apache Cassandra™ Use Cases

2

Data Modelling with Apache Cassandra™

3

Application Development

4

What's NEXT ?

# Meetup @Stockholm

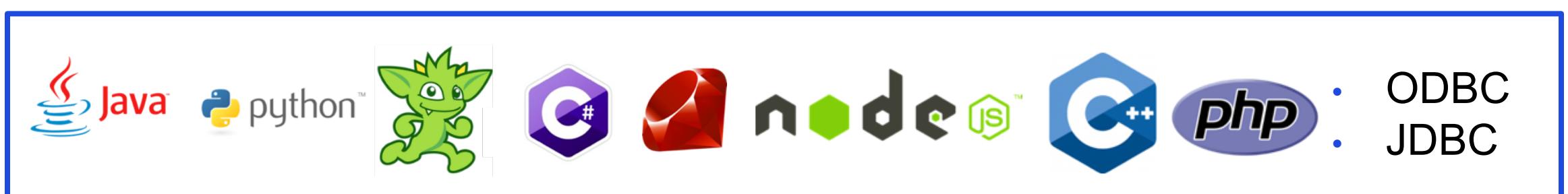
---



# Application Development

# DataStax Drivers

- DataStax Cassandra Drivers (OSS)
  - CQL Support
  - Sync / Async API
  - Address Translation
  - Load Balancing Policies
  - Retry Policies
  - Reconnection Policies
  - Connection Pooling
  - Auto Node Discovery
  - SSL
  - Compression
  - Query Builder
  - Object Mapper
- DataStax Enterprise Drivers
  - OSS Drivers capabilities plus Enterprise improvements for
    - Performance, Usability, Scalability, Ecosystem
    - DSE Advanced Security, Unified Authentication
    - DSE Graph Fluent API
    - DSE Geometric Types



# Apache *Maven*<sup>™</sup>

- OSS Driver

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

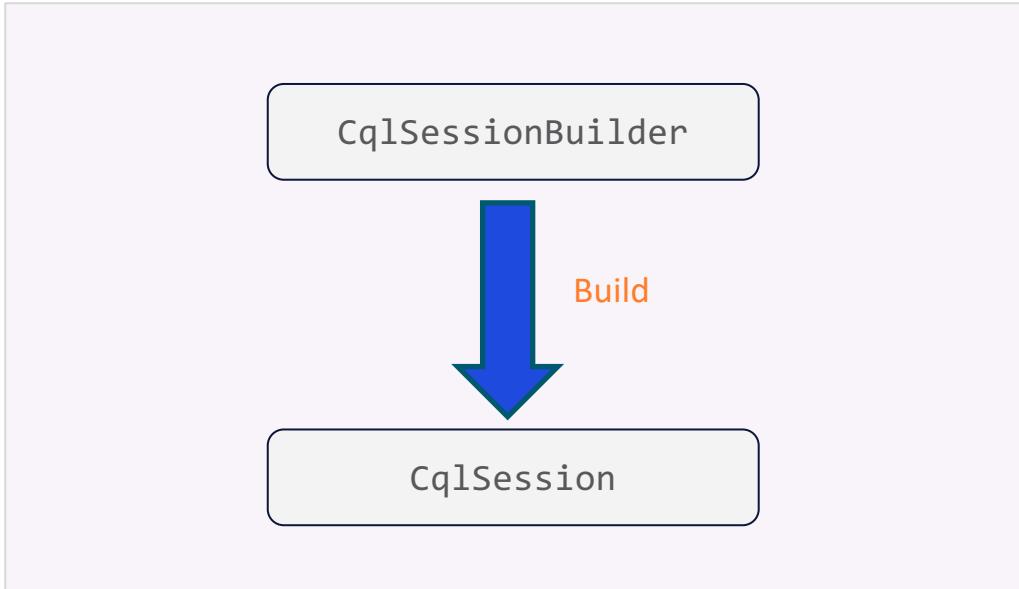
- DSE Driver

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

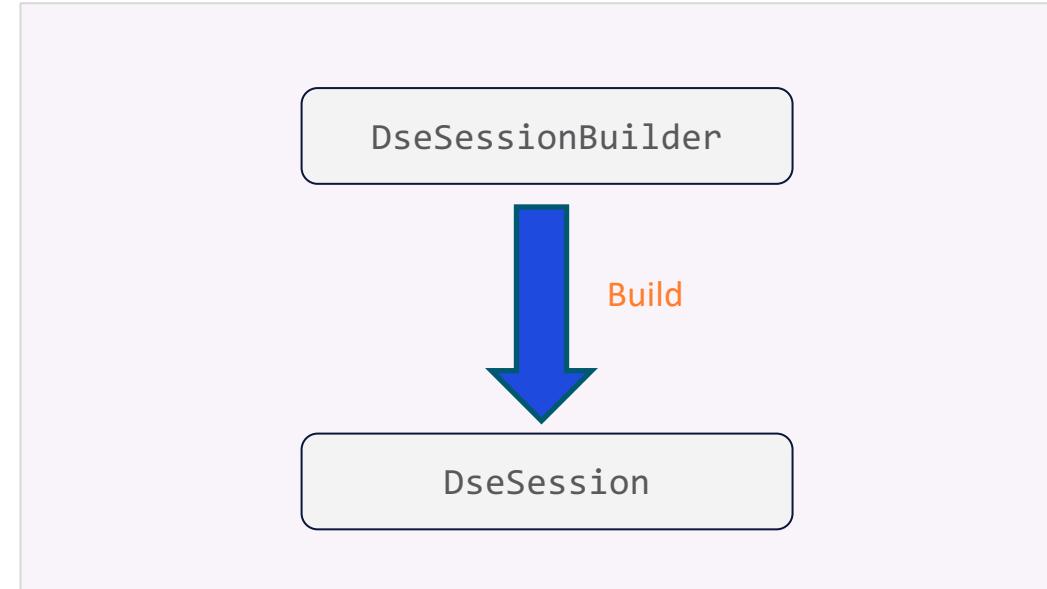


# Connectivity

- OSS Driver



- DSE Driver



*NB : “Cluster” concept from previous driver versions has been collapsed into “Session”*

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

# Prepared and Bound Statements

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

```
PreparedStatement ps = cqlSession.prepare("SELECT * from t1 where c1 = ?");  
  
BoundStatement bound = ps.bind(5);  
  
cqlSession.execute(bound);
```

# Query Builder



# 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>
```

## DSE Driver (current version 2.2.0)

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



# QueryBuilder

```
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()
```

# QueryBuilder

- 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();
```

# #4 – Application Development

---

- Drag and drop « **Meetup\_04 – Application Development** » in the Studio
  - Execute the exercises in the Notebook
- 
- Drag and drop « **Meetup\_05 – Start the Application** » in the Studio
  - Execute the exercises in the Notebook

# Agenda

0

Bootstraping

1

Understanding Apache Cassandra™ Use Cases

2

Data Modelling with Apache Cassandra™

3

Application Development

4

What's NEXT ?

# Developers and Architect Resources



# DataStax Academy

Tell more about this section here

[BACK TO AGENDA](#)

# 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)



<https://academy.datastax.com>

# Learning Paths on DataStax Academy

- Unsure where to start?
- Follow a learning path to learn about topics related to your role.
  - Administrator
  - Analytics Specialist
  - Architect
  - Developer
  - Graph Specialist
  - Search Specialist

<https://academy.datastax.com/paths>

## Developers and Architect Resources

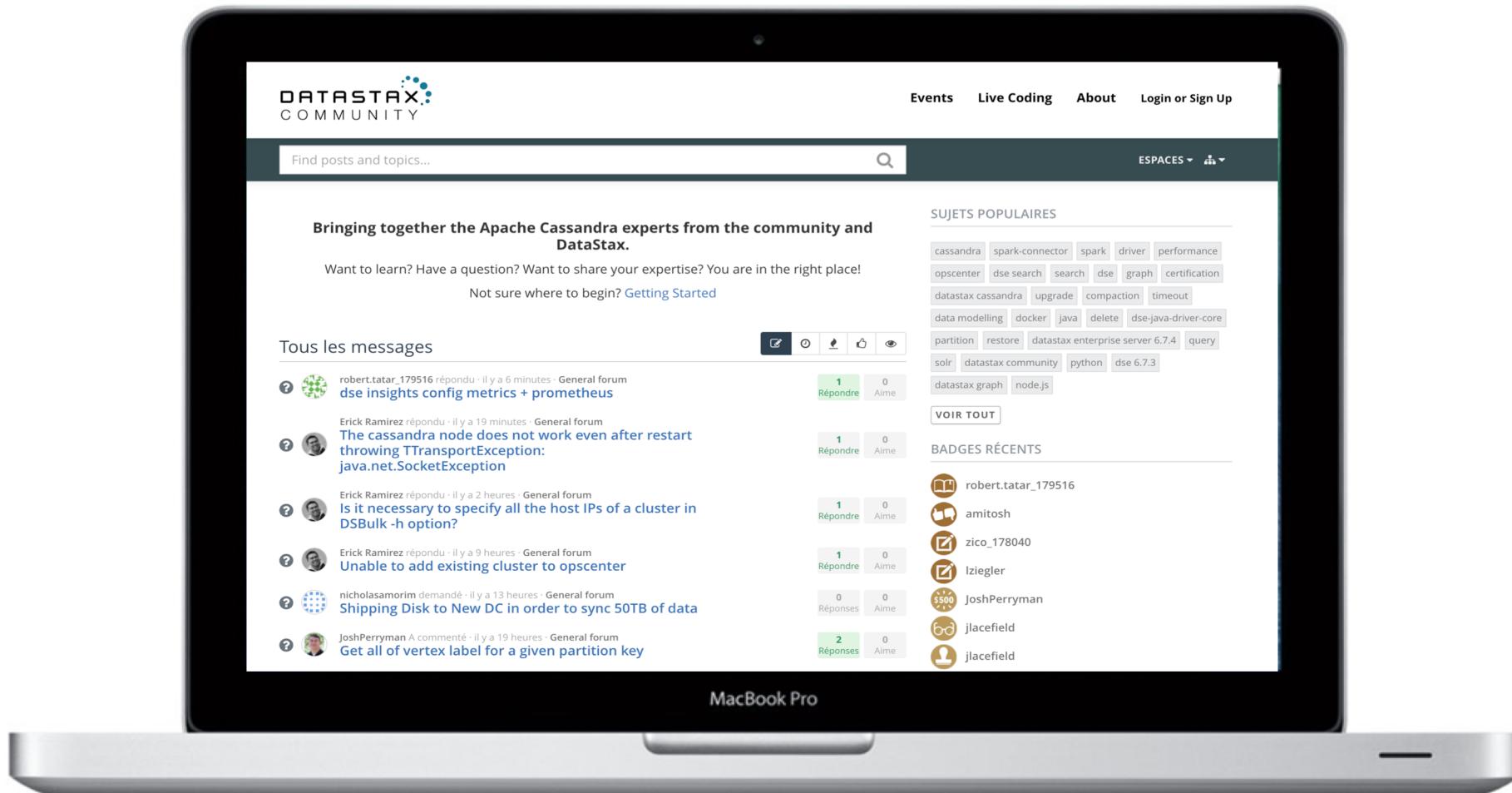


# DataStax Community

Tell more about this section here

[BACK TO AGENDA](#)

# community.datastax.com



## Developers and Architect Resources



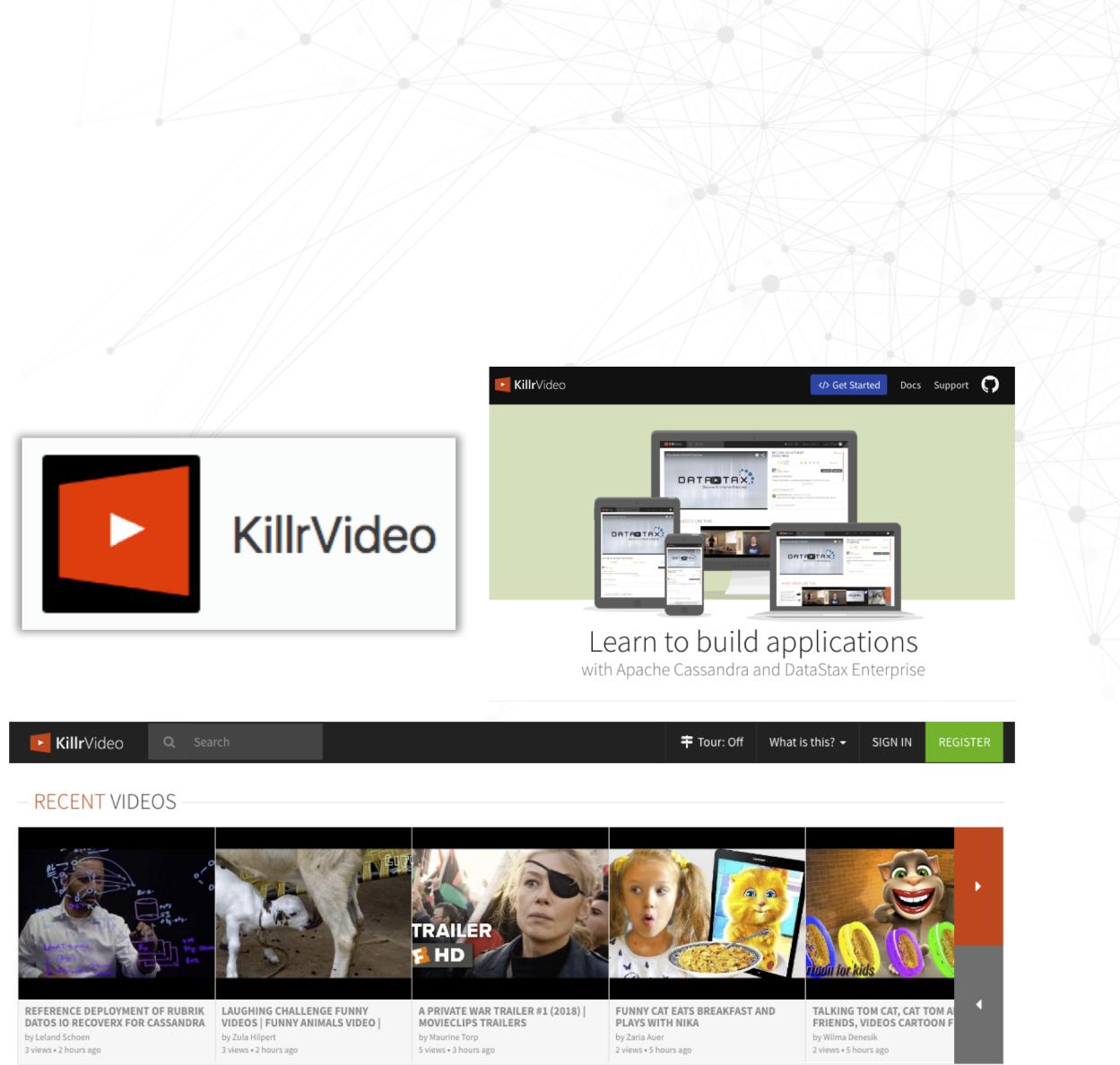
# Sample Code and Reference Applications

Tell more about this section here

[BACK TO AGENDA](#)

# 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!



## Developers and Architect Resources



# Podcast, LiveStreams

Tell more about this section here

[BACK TO AGENDA](#)

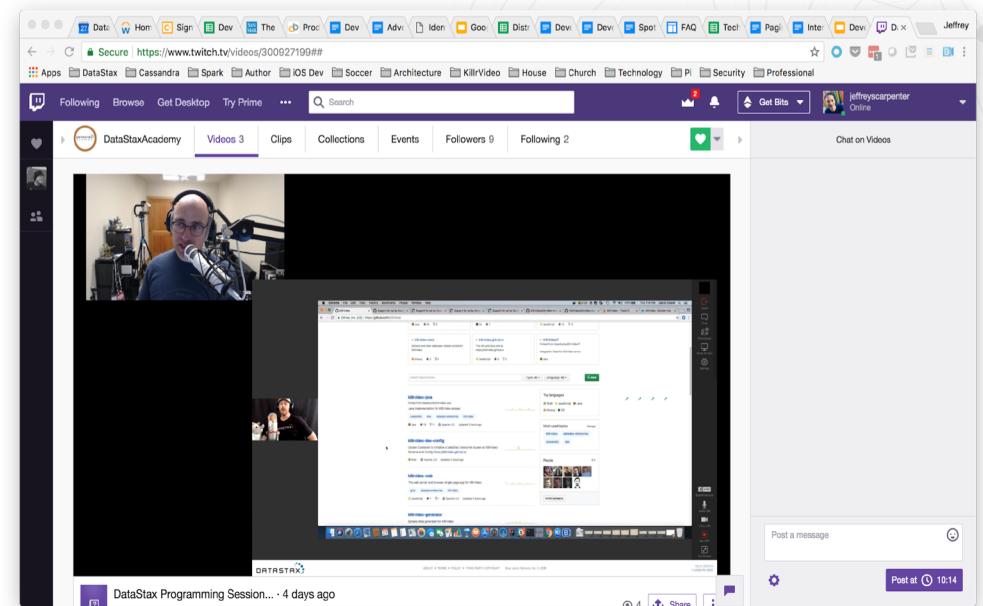
# Distributed Data Show

- Interview-style show featuring a mix of DataStax and industry guests
- We go in-depth on the technology and challenges of data in large-scale distributed systems
- Released weekly on DataStax Academy [YouTube channel](#) and as a podcast
- Send us your suggestions for topics and guests – we love customer use cases



# Live Coding on Twitch

- Live coding sessions with advocates and guests each Thursday
  - <https://www.twitch.tv/datastaxacademy>
- Working through the challenges of building distributed systems
- Join the conversation and ask questions
- Some advocates also do streaming on personal channels





# Thank You

