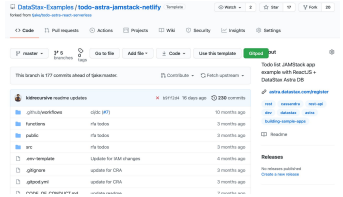

Cassandra Day
2022

Hands-on Workshop

# Cassandra Fundamentals

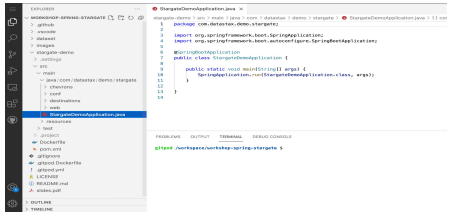
Sponsored by DataStax

Source code + exercises + slides



GitHub

IDE



Gitpod

Nothing to install !

Database + Api + Streaming


DataStax Astra DB

Hands-On Housekeeping

572 and counting


# Intro to Cassandra Workshop

Awarded to **YOUR NAME** • attendee@workshop.com

Issued on **Jun 16, 2021**

Offered By
DataStax Developers

Upgrade Complete! This badge is to certify successful completion of the DataStax Cassandra Workshop: "Fundamentals of Apache Cassandra".

**Verified**

Last verified by Badgr on **Jun 17, 2021**

Re-verify Badge

**EARNING CRITERIA**

Recipients must complete the earning criteria to earn this Badge

To earn this badge, individuals must complete the follow
during the **Intro to Cassandra Workshop**:

- Attend the lecture
- Complete the practical steps by doing all requi


Achievement Unlocked !

01

Why Cassandra ?

02

Introduction to
Apache Cassandra™

03

Distributed Architecture

04

CAP Theorem

05

Tables
and Partitions

06

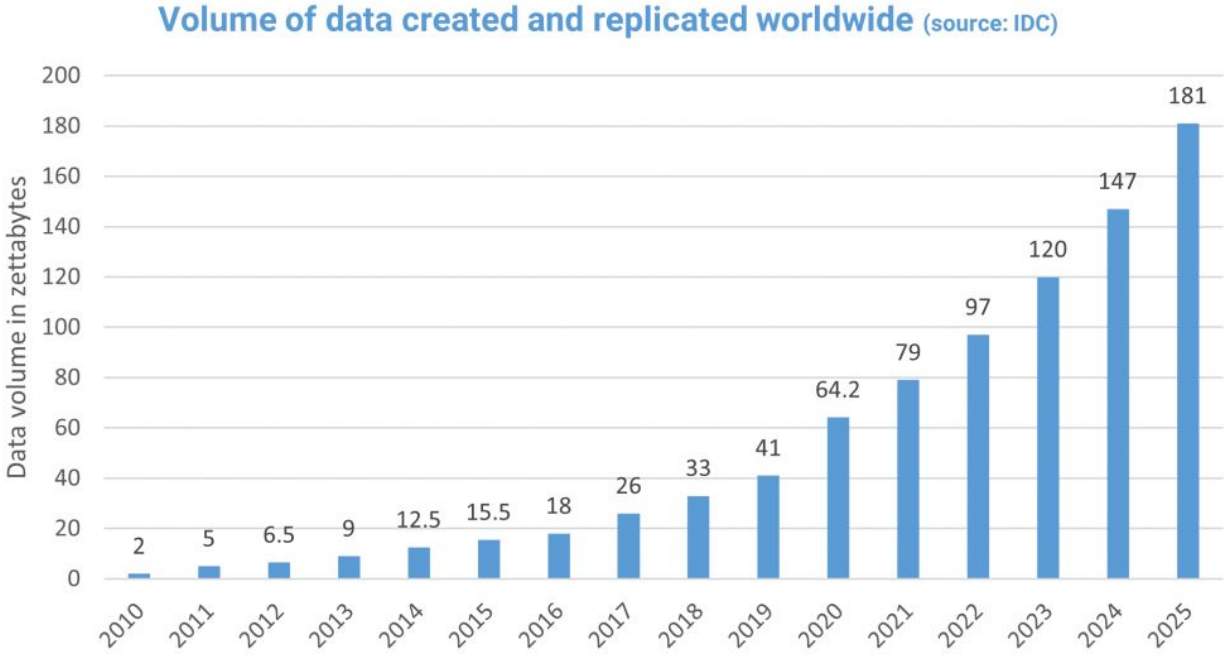
What's next?
Homework, Next Session

# Agenda

# RDBMS were designed to run on a single machine

| WHAT'S A ZETTABYTE? | |
|---|---|
| 1 kilobyte | 1,000 |
| 1 megabyte | 1,000,000 |
| 1 gigabyte | 1,000,000,000 |
| 1 terabyte | 1,000,000,000,000 |
| 1 petabyte | 1,000,000,000,000,000 |
| 1 exabyte | 1,000,000,000,000,000,000 |
| 1 zettabyte | 1,000,000,000,000,000,000,000 |

SOURCES: CISCO



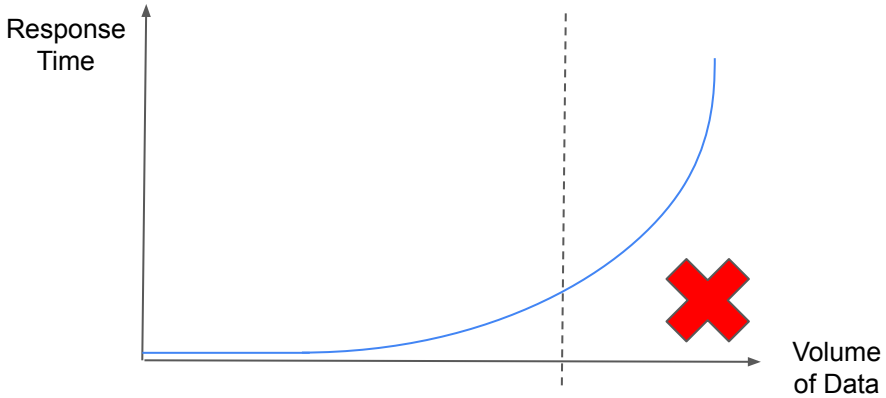

Relation Databases have limited scalability

**OLTP**: Online Transaction Processing

Fast Queries
"Customer-facing"
High number of transactions
Usually Hot / Live Data
High SLA Requirements
(Response Time / Availability)

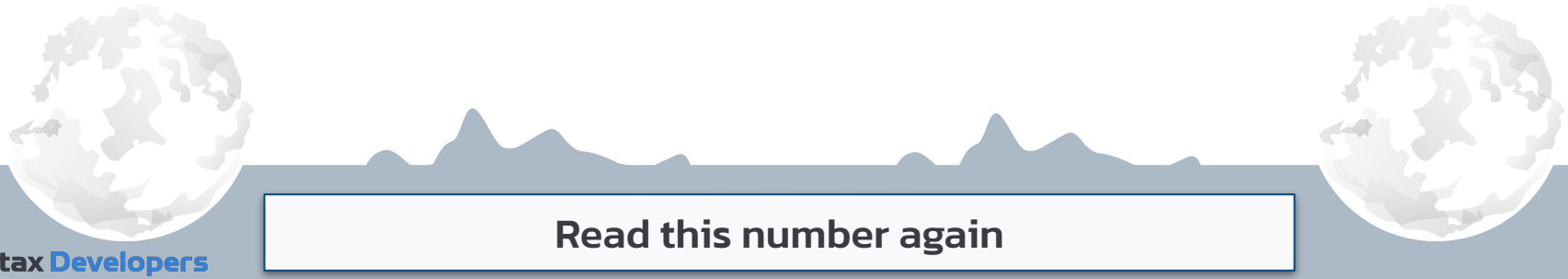
**OLAP**: Online Analytical Processing

Complex Queries
Historical
High volume of data
Often "Cold Data"
Used by / for Analytics



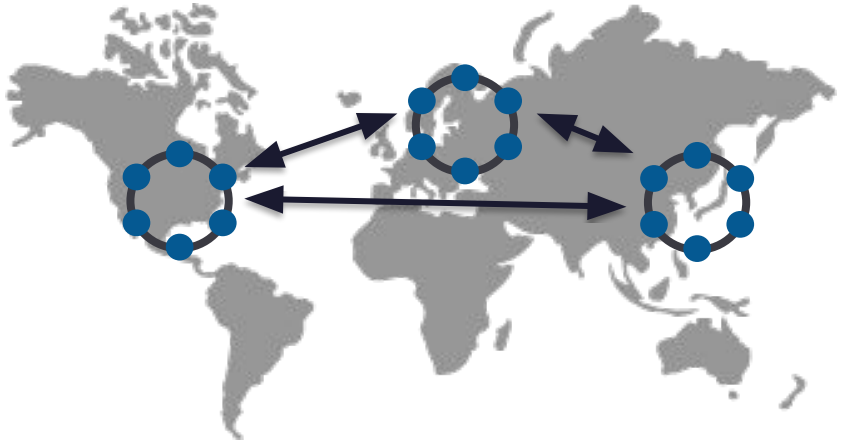

Relational Databases are versatile

In the year 2008 Facebbook reached their first 100,000,000 users.



Read this number again

- Users around all the world
- Fast Internet Connection
- A LOT of data
- High Availability requirements



Distributed Decentralised OLTP Database

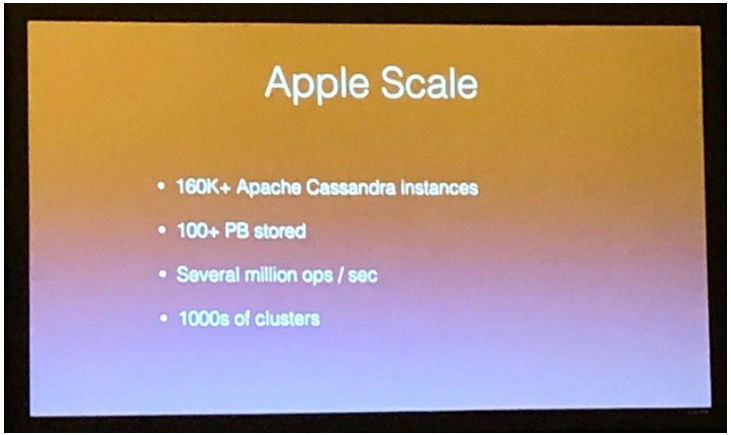
## Apache Cassandra @ Netflix

- 98% of streaming data is stored in Apache Cassandra
- Data ranges from customer details to viewing history to billing and payments
- Foundational datastore for serving millions of operations per second

- 30 million ops/sec on most active single cluster
- 500 TB most dense single cluster
- 9216 CPUs in biggest cluster

| | |
|---|---|
| O(100) | Clusters |
| O(10000) | Instances |
| O(10,000,000) | Replications per second |
| O(100,000,000) | Operations per second |
| O(1,000,000,000,000,000) | Petabytes of data |

dtsx.io/cassandra-at-netflix




And many others...

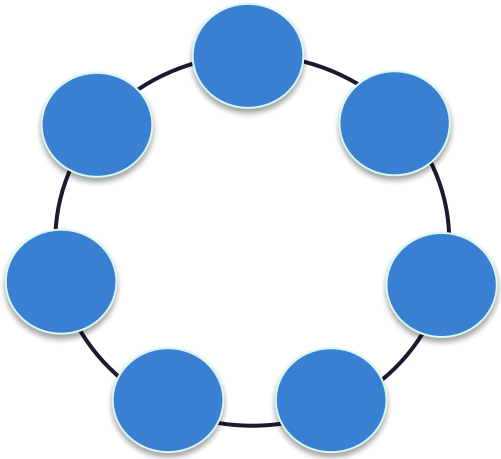
## Cassandra Biggest Users (and Developers)

# Apache Cassandra

**NoSQL Distributed Decentralised Database Management System**


- Big Data Ready
- Read / Write Performance
- Linear Scalability
- Highest Availability
- Self-Healing and Automation
- Geographical Distribution
- Platform Agnostic
- Vendor Independent

**Partitioning** over distributed architecture makes the database capable to handle data of any size: we mean petabytes scale. Need more volume? Add more nodes.

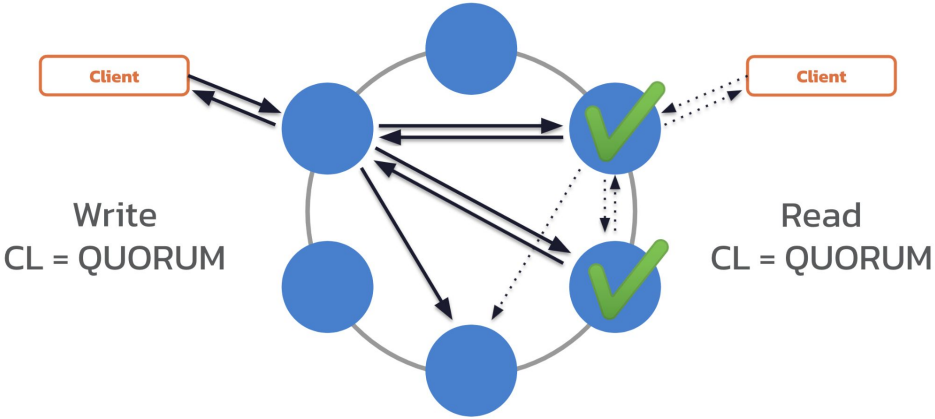




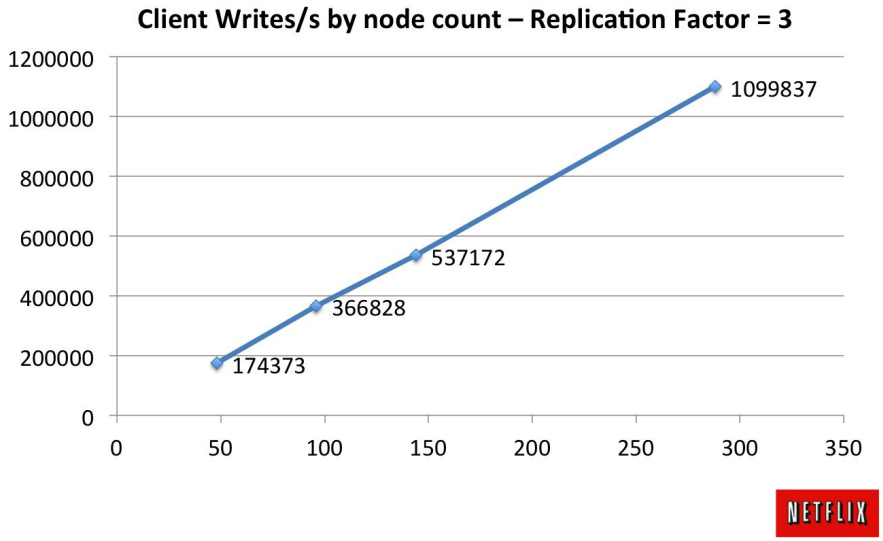
Big Data Ready

Even a single Cassandra node is very performant but a cluster consisting of multiple nodes and data centers brings throughput to the next level.

Decentralisation (**masterless architecture**) means that every node is able to deal with any request, read or write.



- For volume or velocity, there are no limitations
- **Linear** - No overhead on new nodes, scales with your needs*

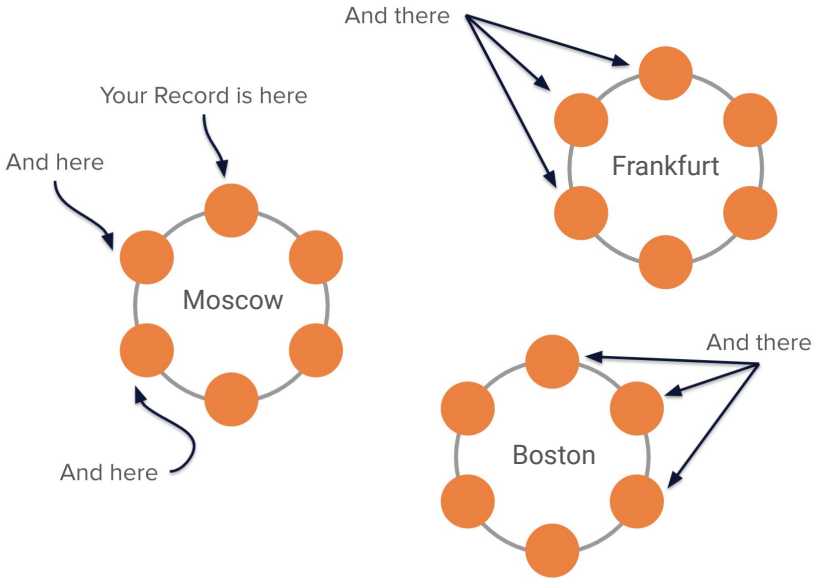



NETFLIX

## Linear Scalability

Replication, Decentralisation, and Topology-Aware Placement Strategy take care of possible downtimes:

- Multiple Live Replicas
- No Single Point of Failure
- Network topology-aware data placement
- Client-side Smart Reconnection and Strong Retry Mechanism



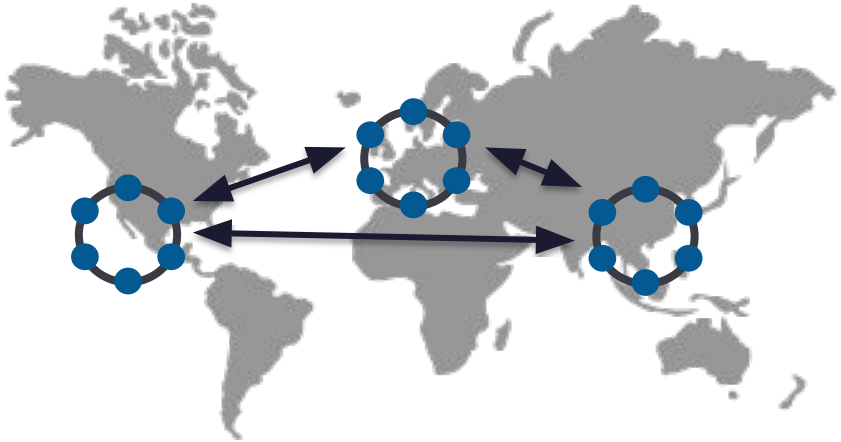



**Highest Availability**

Operations for a huge cluster can be exhausting so Apache Cassandra clusters are smart and able to scale, change data placement and recover automagically.

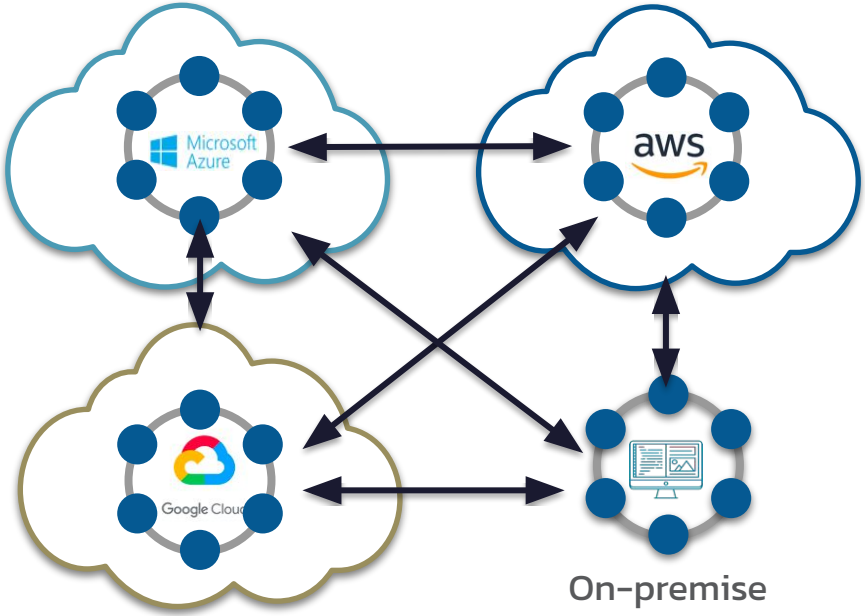
Cassandra's trademark is multi-datacenter deployments, granting you an exceptional capability for disaster tolerance while keeping your data close to your clients - worldwide.

All DCs are active (available for both writes and reads)!



## Geographical Distribution

Apache Cassandra is **not bound to any platform** or service provider, helping you build hybrid-cloud and multi-cloud solutions with ease.




## Platform Agnostic

Cassandra doesn't belong to any of commercial vendors but controlled by a non-profit Open Source **Apache Software Foundation**, already familiar to you by *Hadoop*, *Spark*, *Kafka*, *Zookeeper, Maven* and many other projects.


**Vendor Independent**

01

Why Cassandra ?

02

Introduction to
Apache Cassandra™

03

Distributed Architecture

04

CAP Theorem

05

Tables
and Partitions

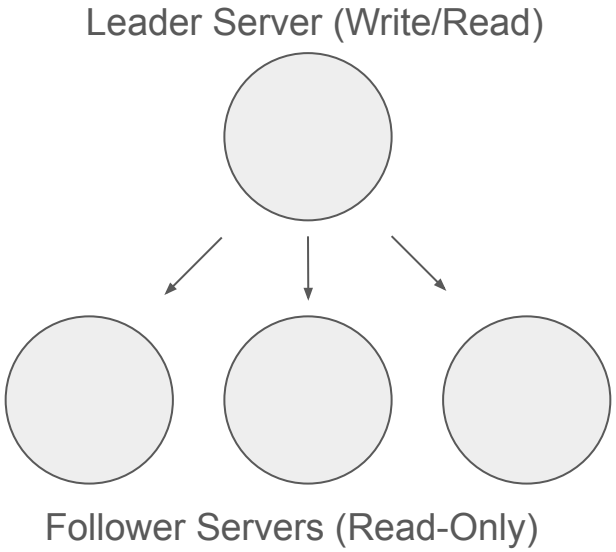
06

What's next?
Homework, Next Session

# Agenda

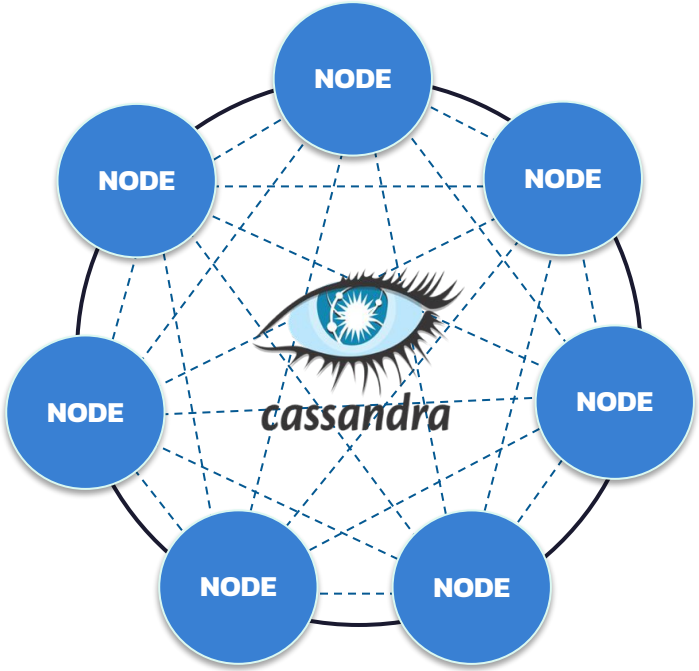
# All servers are created equal

*Cassandra Holy Book, P.I Paragraph I*




1. Single Point of Failure
2. Hard to scale for writes
3. Application needs to know where to write

# Traditional Architecture




1. **NO** Single Point of Failure
2. Scales for writes and reads
3. Application can contact any node
   (in case of failure - just contact next one)

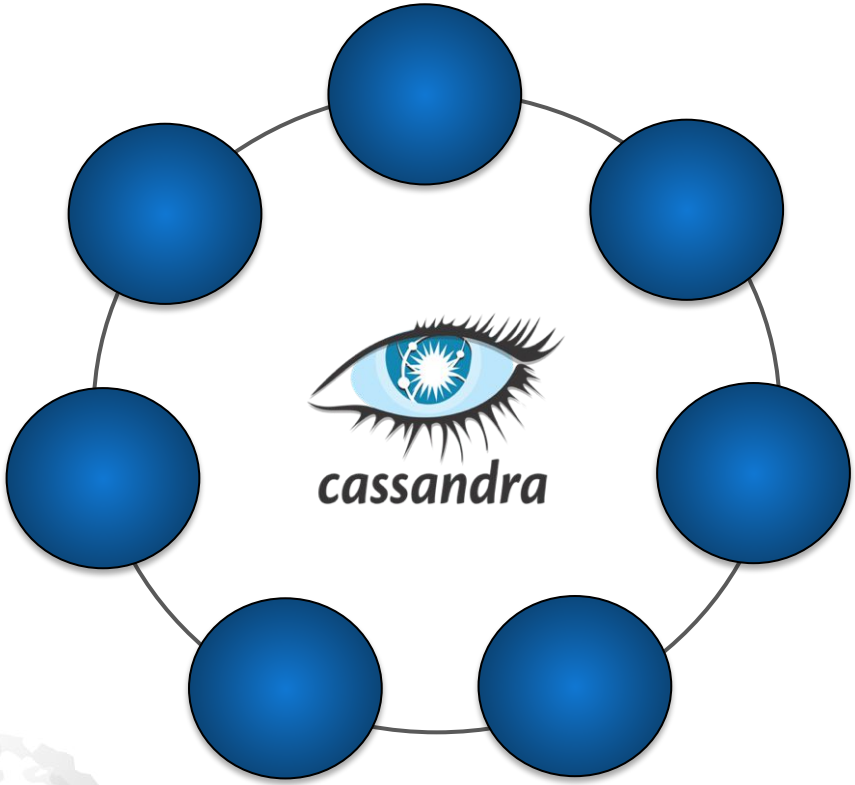
# Master-less (Peer-to-Peer) Architecture

# Data is Partitioned

```
CREATE TABLE sensors_by_network (
    network     text,
    sensor      text,
    temperature integer,
    ...
);
```

| sensors_by_network | | |
|---|---|---|
| network | sensor | temperature |
| alabama | a001 | 92 |
| alabama | a002 | 88 |
| california | c001 | 210 |
| delaware | d001 | 45 |
| delaware | d002 | 50 |
| florida | f001 | 72 |
| georgia | g001 | 69 |
| georgia | g002 | 70 |
| hawaii | h001 | 40 |
| idaho | i001 | 105 |
| idaho | i002 | 110 |
| kansas | k001 | 35 |

Let's speak data

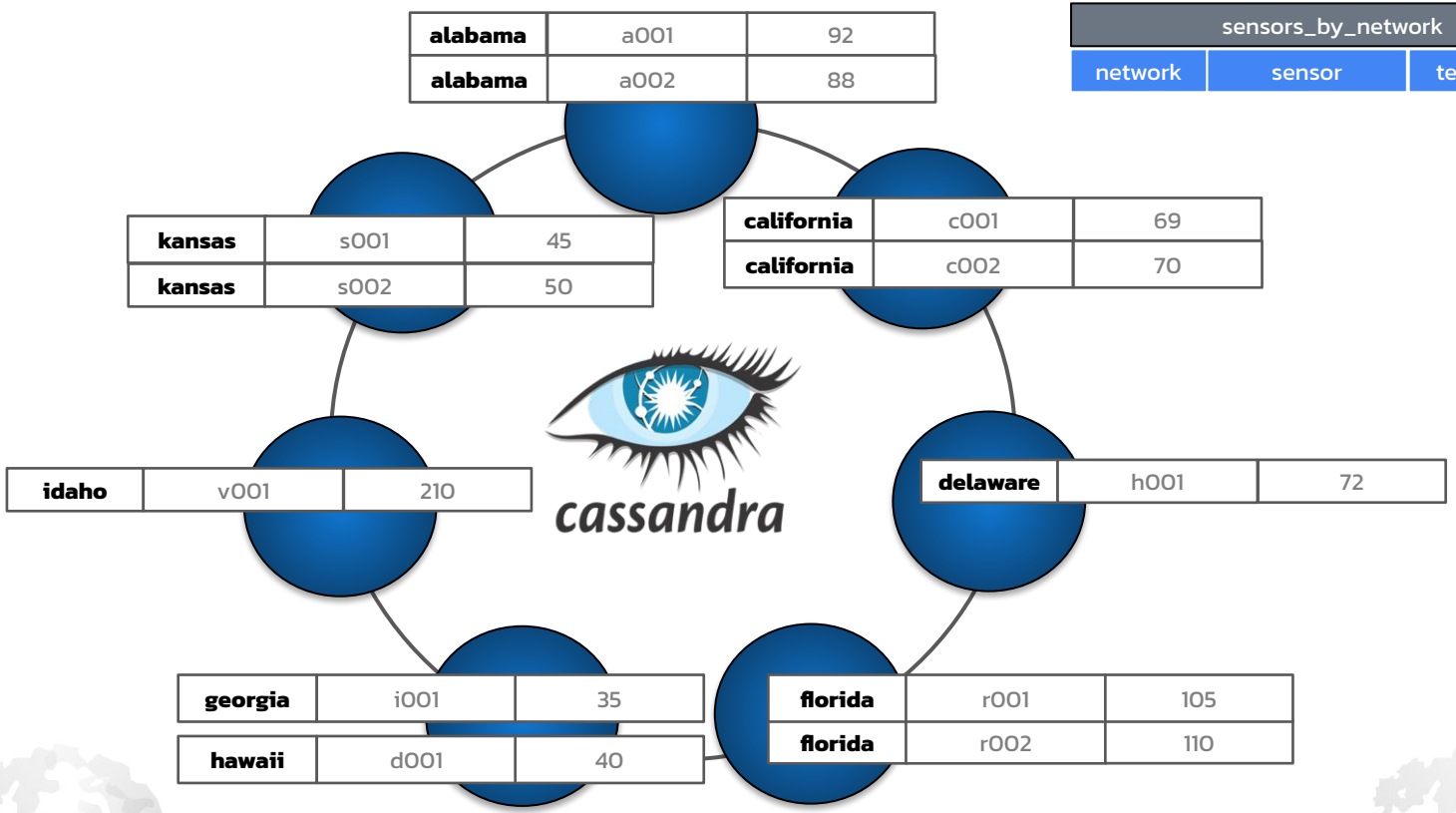



| sensors_by_network | | |
|---|---|---|
| network | sensor | temperature |
| alabama | a001 | 92 |
| alabama | a002 | 88 |
| california | c001 | 210 |
| delaware | d001 | 45 |
| delaware | d002 | 50 |
| florida | f001 | 72 |
| georgia | g001 | 69 |
| georgia | g002 | 70 |
| hawaii | h001 | 40 |
| idaho | i001 | 105 |
| idaho | i002 | 110 |
| kansas | k001 | 35 |

*Partition Key*

Data is organized as Tables

| sensors_by_network | | |
|---|---|---|
| network | sensor | temperature |




Data is organized as Distributed tables

Table

```
CREATE TABLE sensor_data.sensors_by_network (
    network     text,
    sensor      text,
    temperature integer,
    PRIMARY KEY ((network), sensor)
);
```

Partition key



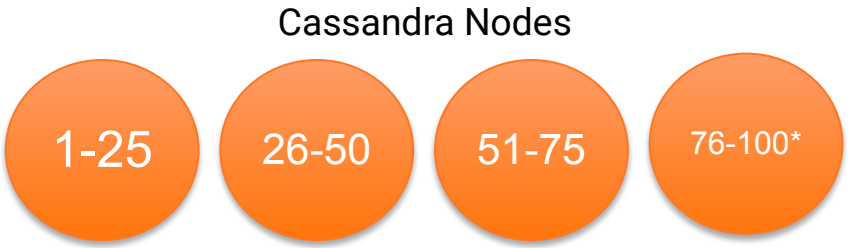
Key-based Partitioning

| network | sensor | Value |
|---|---|---|
| **alabama** | f001 | 92 |
| **alabama** | f002 | 88 |
| **idaho** | s001 | 45 |
| **idaho** | s002 | 50 |
| **hawaii** | r001 | 105 |
| **hawaii** | r002 | 110 |

***Partition Keys***


| Network | Sensor | Value |
|---|---|---|
| **59** | f001 | 92 |
| **59** | f002 | 88 |
| **12** | s001 | 45 |
| **12** | s002 | 50 |
| **45** | r001 | 105 |
| **45** | r002 | 110 |

***Tokens***




## Partitioning and Token Ranges

# Why partitioning?
# Because scaling doesn't have to be [s]hard!

Big Data doesn't fit to a single server, splitting it into chunks we can easily spread them over dozens, hundreds or even thousands of servers, adding more if needed.

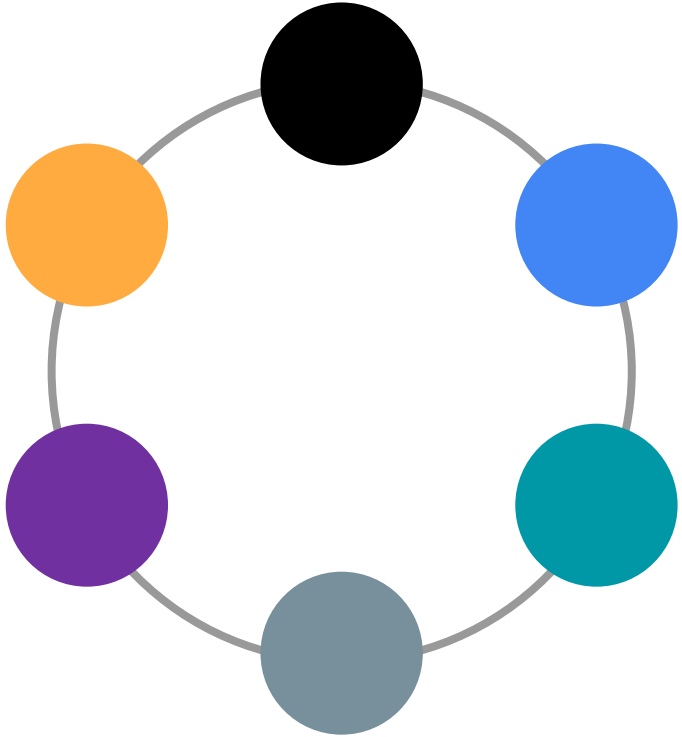

NOTICE: No downtime! All scaling operations are live!

## Token Ranges Recalculation

# Data is Replicated

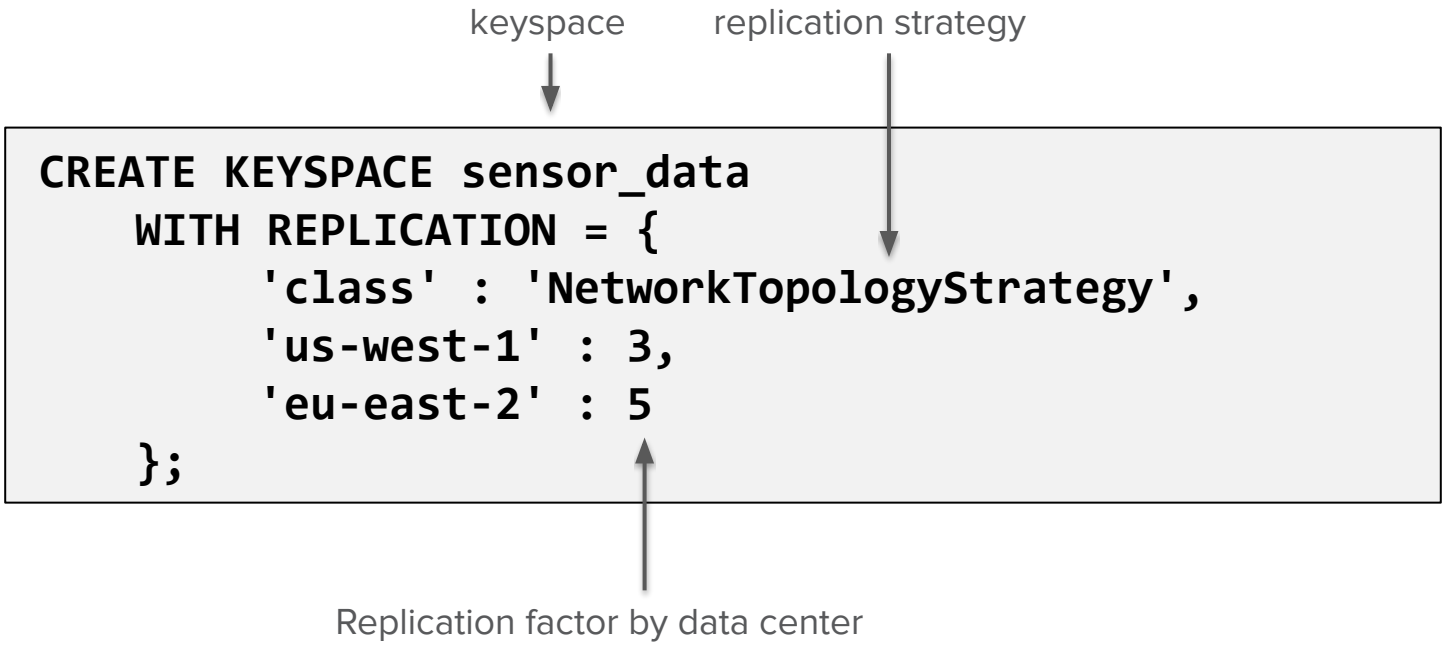
RF = ?

Replication Factor means the number of nodes used to store each partition





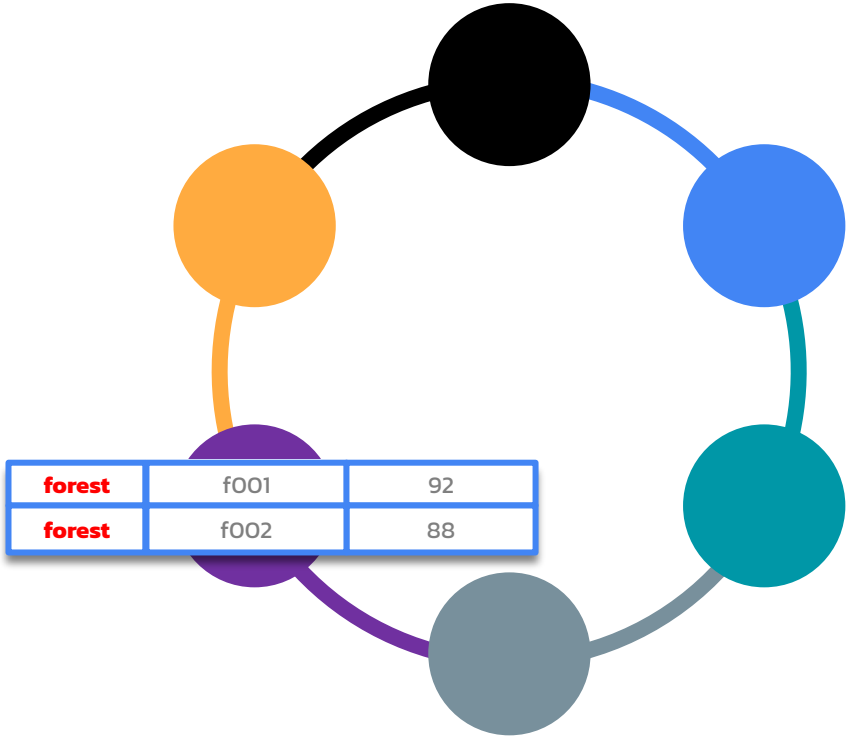
**Replication Factor**




## Data is Replicated

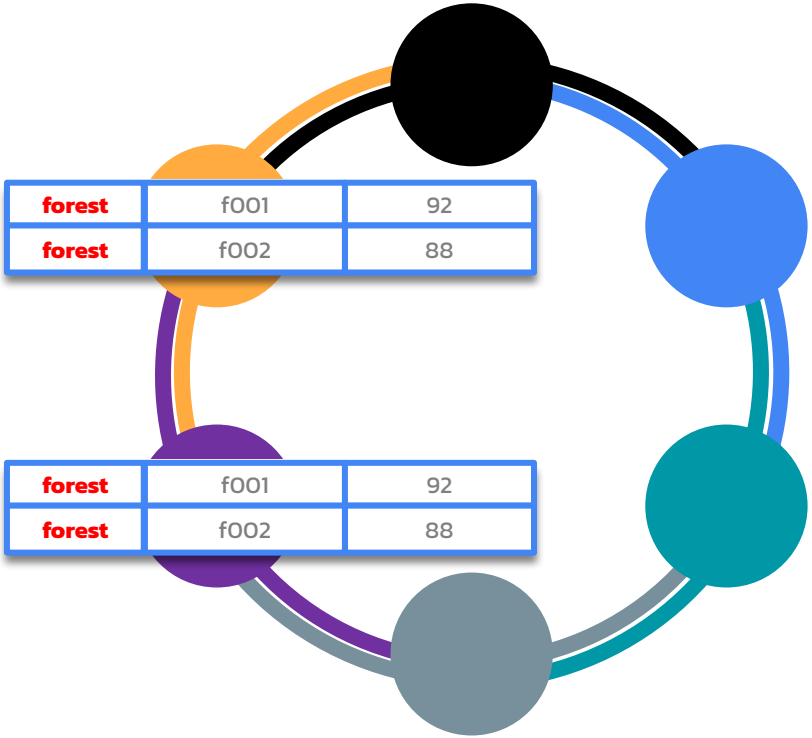
# RF = 1

Replication Factor 1 means that every partition is stored on 1 node



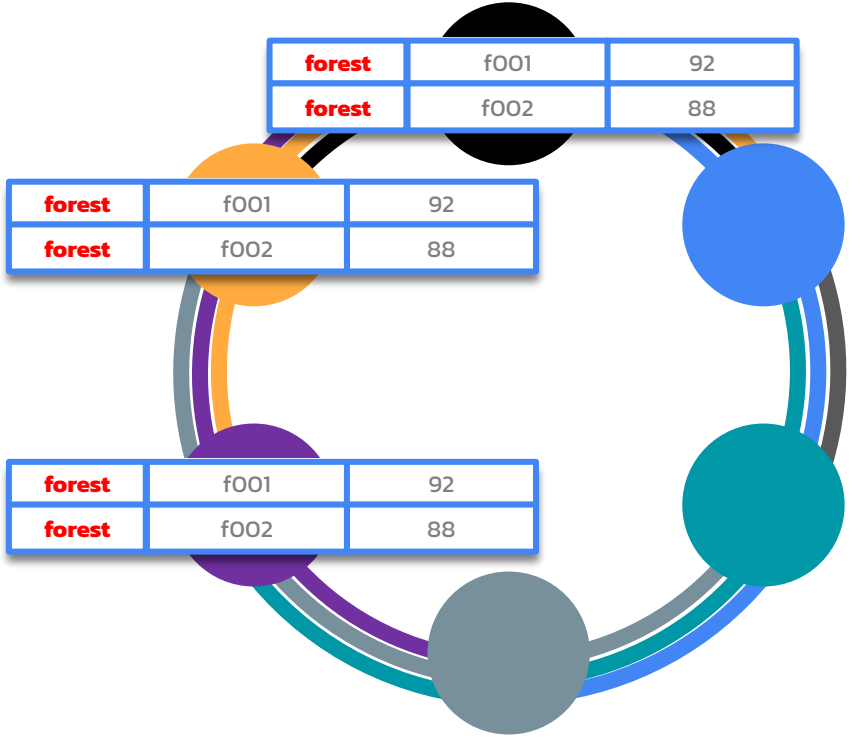
# RF = 2

Replication Factor 2 means that every partition is stored on 2 nodes



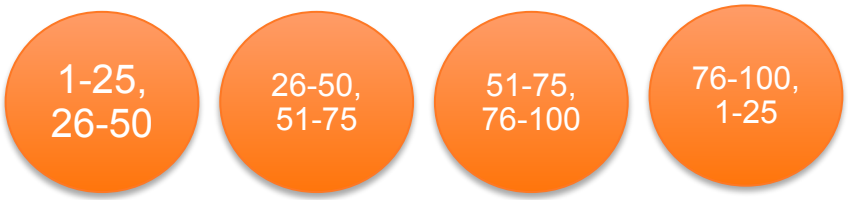
# RF = 3

Replication Factor 3 means that every partition is stored on 3 nodes



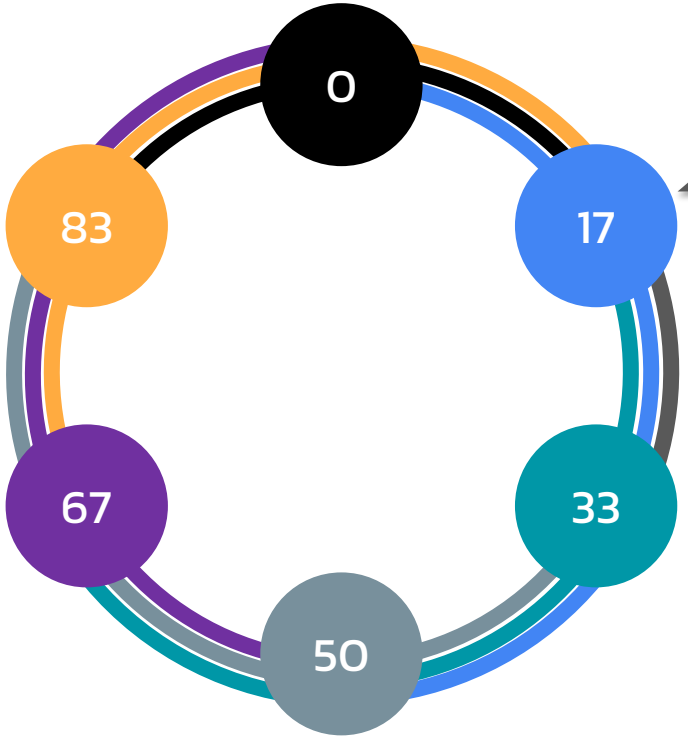
Replication Factor = 1


Replication Factor = 2




And so on…

RF = 3

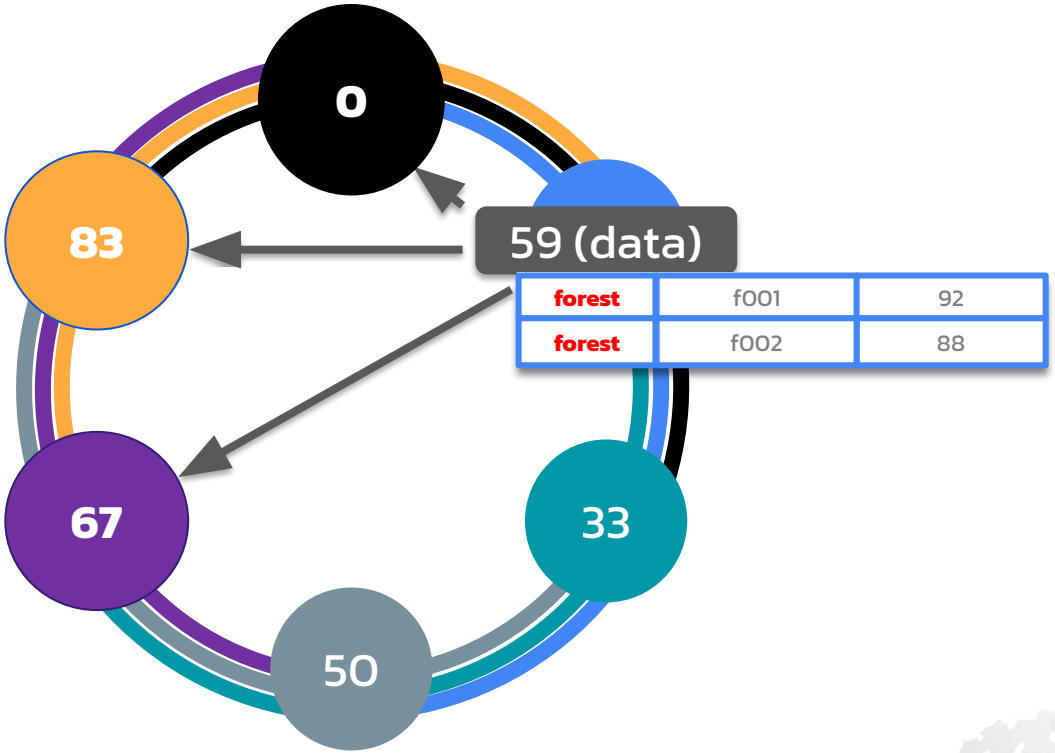



| | | |
|---|---|---|
| forest | f001 | 92 |
| forest | f002 | 88 |

59 (data)

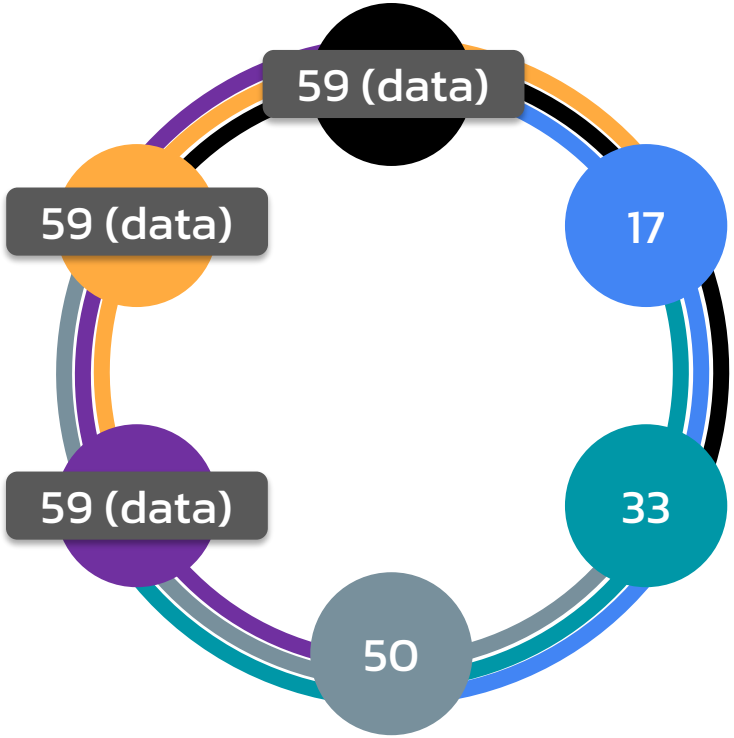
Partitioning + Replication

RF = 3




Partitioning + Replication

RF = 3



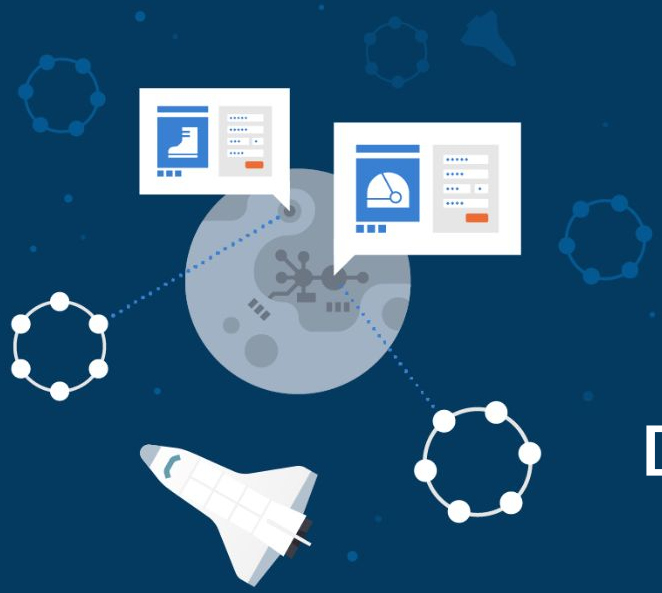



Partitioning + Replication

# IMPORTANT

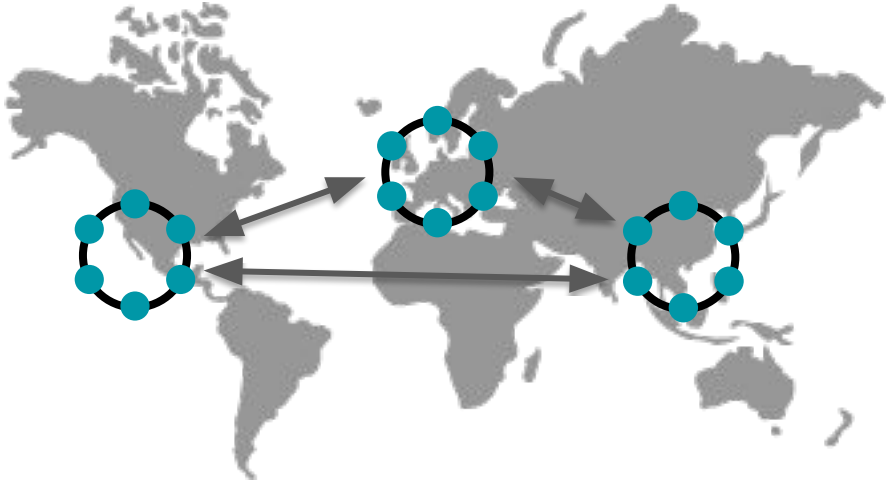
Each Cassandra node and even each Cassandra driver knows Data Allocation in a cluster (it's called Token-Aware), so your application can contact literally ANY server and still get the answer.

(but in normal circumstances a driver will contact the replica node)

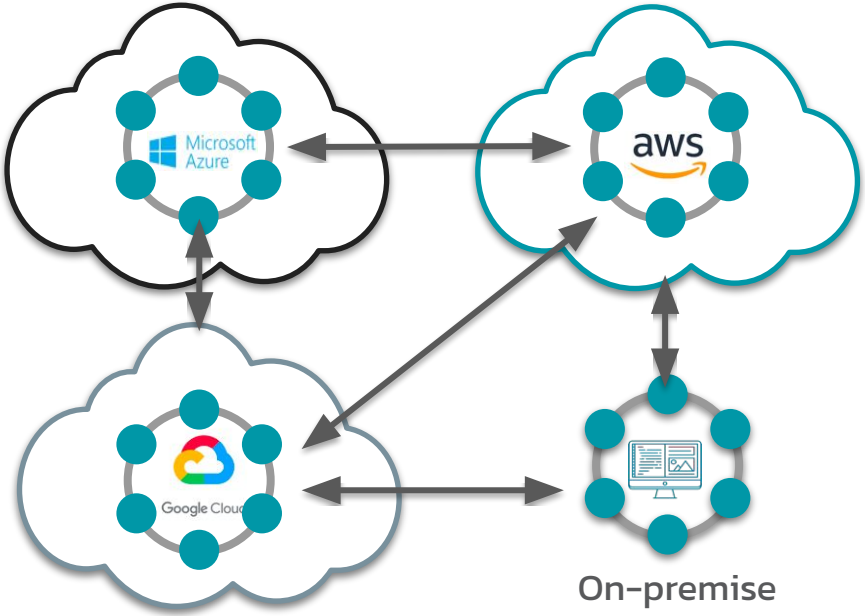


# Data is GLOBALLY distributed

## Geographical Distribution



## Hybrid-Cloud and Multi-Cloud




**Data is globally distributed**

# Lab 1
# Database Setup

github.com/datastaxdevs/workshop-cassandra-fundamentals

- ✅ Create a new database
- ✅ Wake up an existing hibernated database

### Free to Use

Up to 80GB storage and/or 20 million operations monthly.

### Serverless

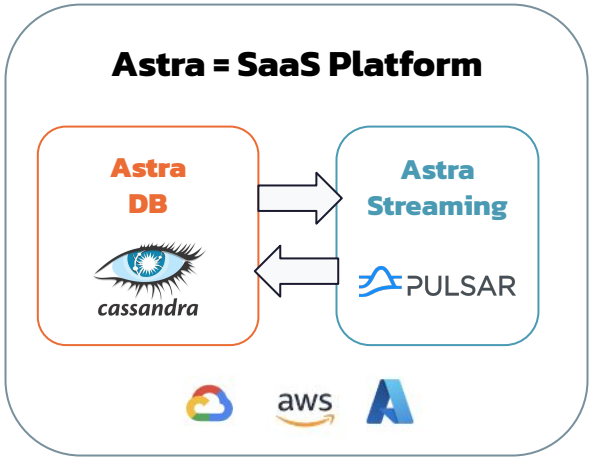
Lower your costs by running Cassandra clusters only when needed.

### No Operations

Eliminate the overhead to install, operate, and scale Cassandra.

### Data APIs

Work natively with Document (JSON), REST, GraphQL and gRPC APIs.




### Global Scale

Put your data where you need it without compromising performance, availability or accessibility.

### End-to-End Security

Secure connect with VPC peering and Private Link. Bring your own encryption key management. SAML SSO secure account access.

### Zero Lock-in

Deploy on AWS, GCP or Azure and keep compatibility with open-source Cassandra.

### Relational Indexes

Storage Attached Indexing (SAI) lets you query tables using any columns.

**Astra = Cassandra As a Service++**

01

Why Cassandra ?

02

Introduction to
Apache Cassandra™

03

Distributed Architecture

04

CAP Theorem

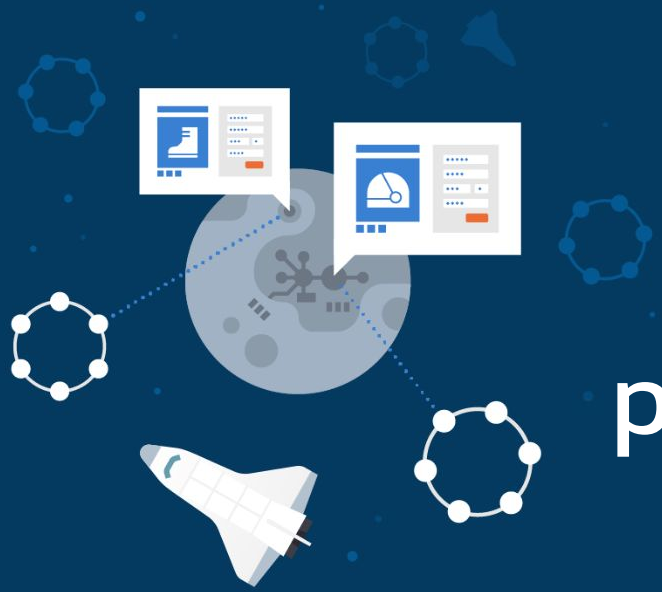
05

Tables
and Partitions

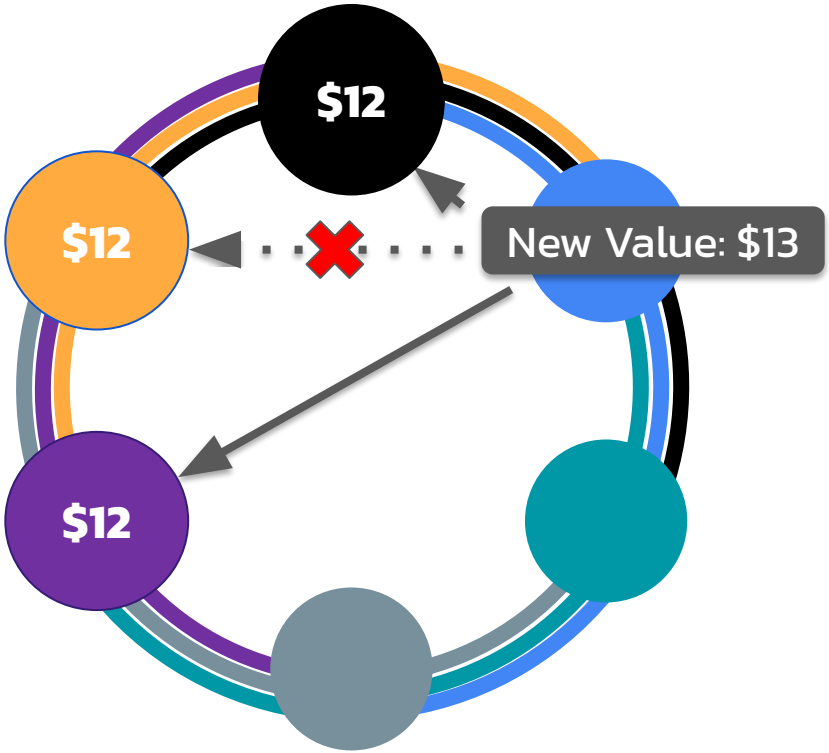
06

What's next?
Homework, Next Session

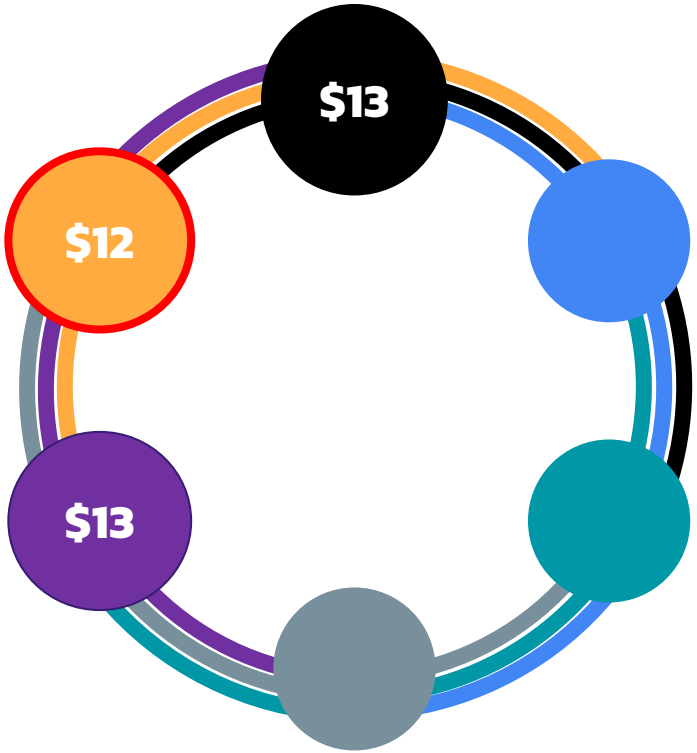
# Agenda



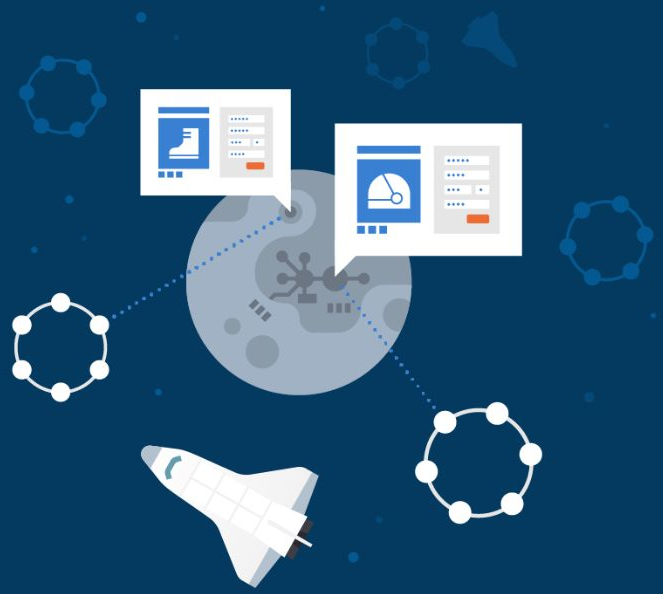
# What is the biggest problem of replication?




**Potential Inconsistency**

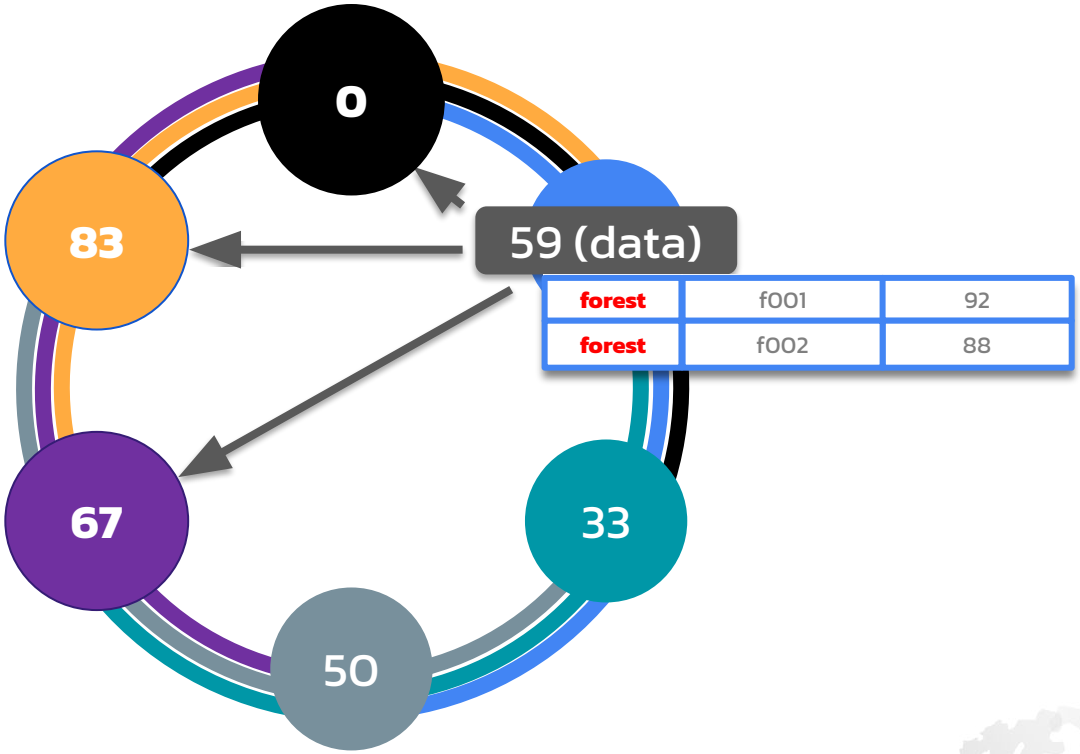



## Potential Inconsistency



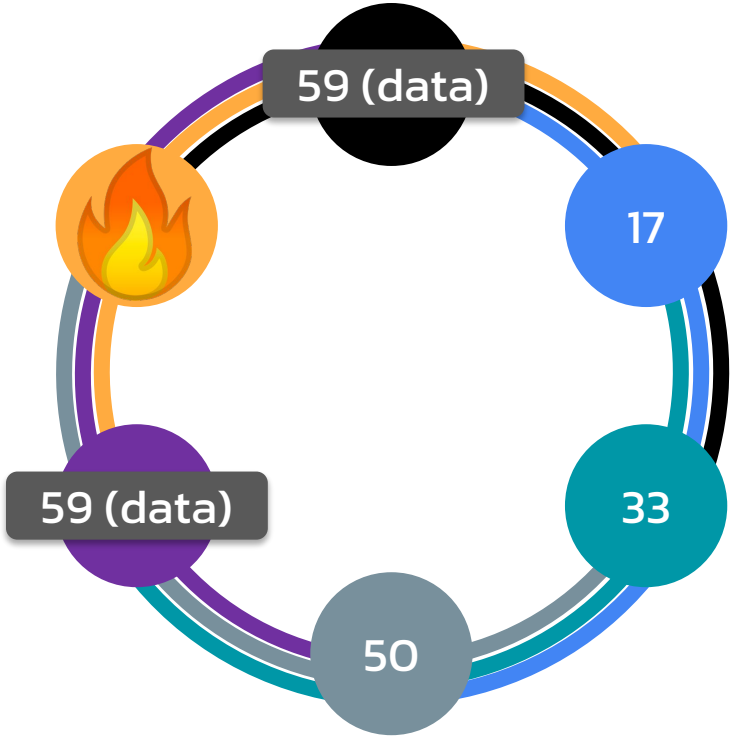
# Cassandra Layered Self-Defence

RF = 3




## Hinted Hand-offs

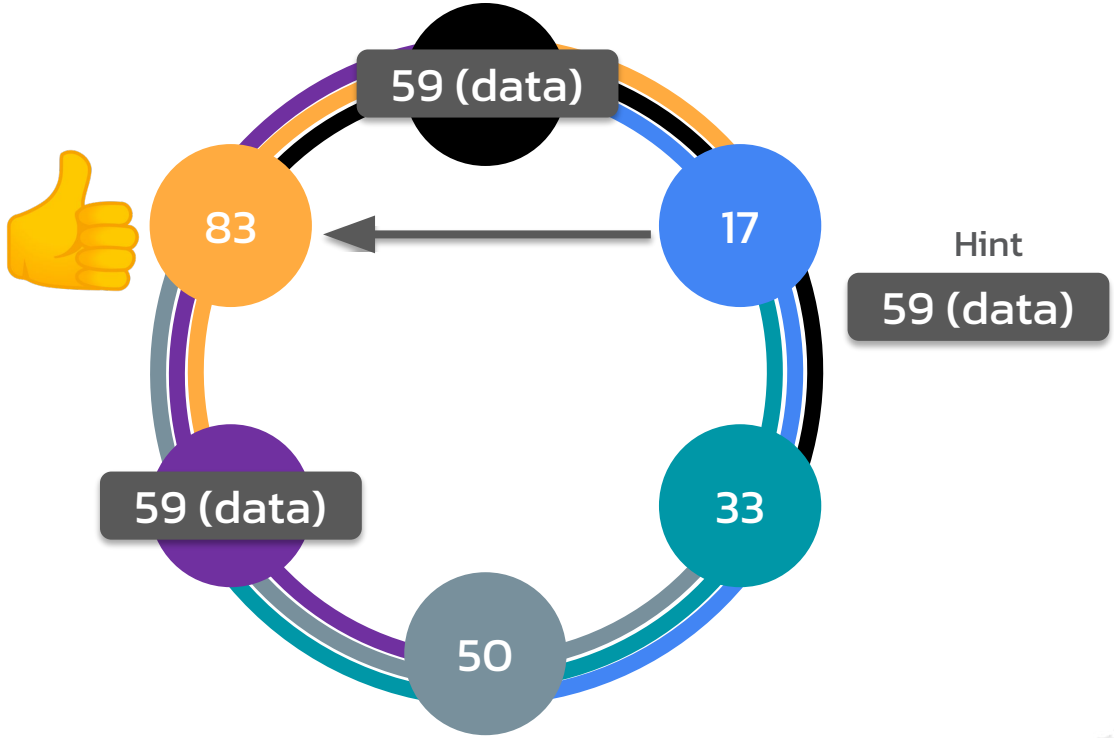
RF = 3




Hint

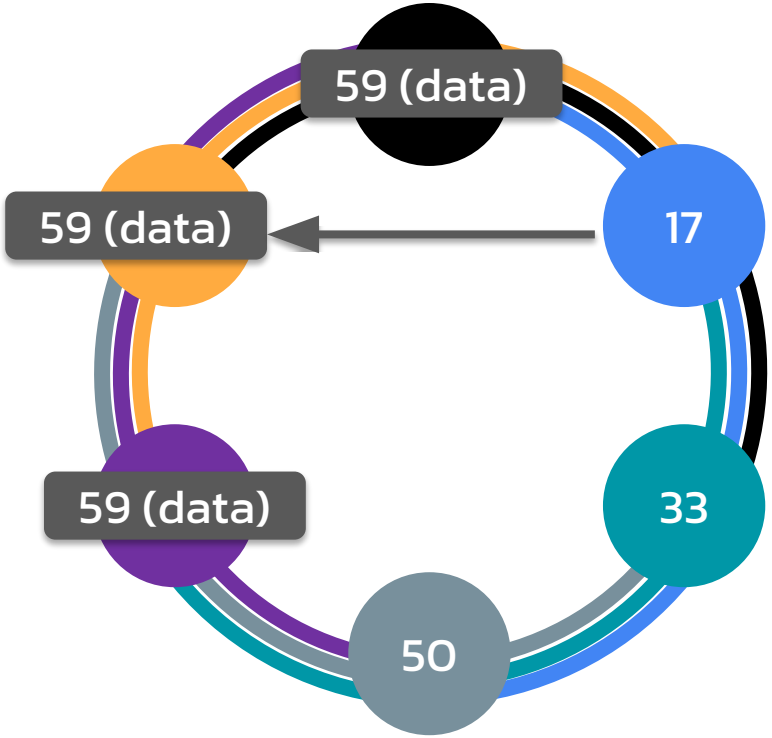
59 (data)

## Hinted Hand-offs




# Hinted Hand-offs

RF = 3




## Hinted Hand-offs

Besides **hinted hand-offs**, very are few more tools to keep consistency under control:

- Consistency Levels — *Developer job*
- Repair on Read — *Cassandra job*
- Repairs — *Admin job*

01

Why Cassandra ?

02

Introduction to
Apache Cassandra™

03

Distributed Architecture

04

CAP Theorem

05

Tables
and Partitions

06

What's next?
Homework, Next Session

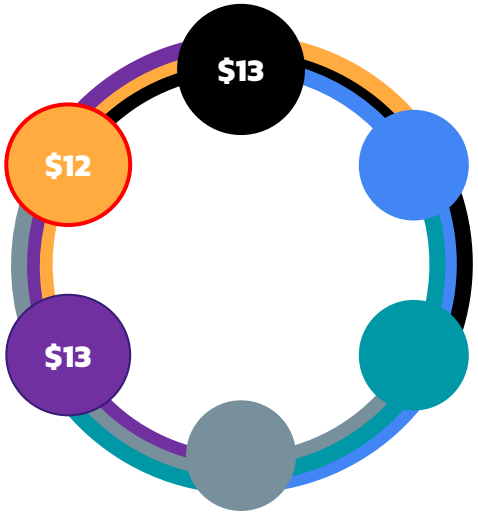
# Agenda

CAP Theorem operates three features:

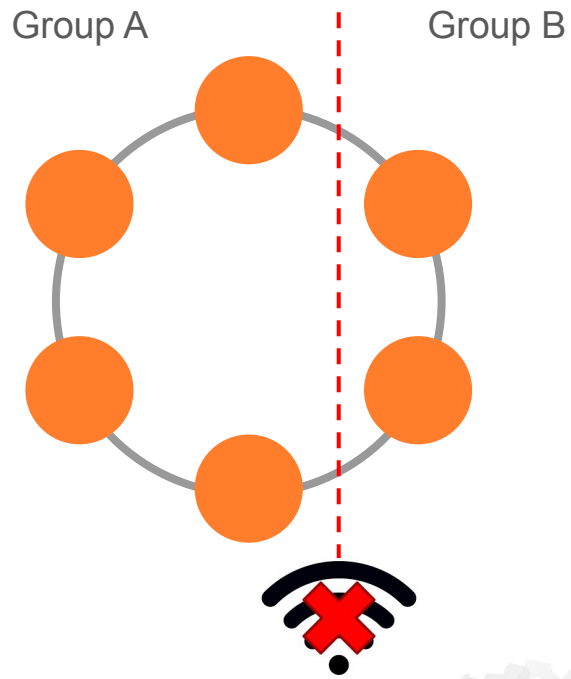
1. Availability
2. Consistency
3. Partition Tolerance

**Availability** basically means “Uptime”. You ask the question, you get the answer. If failure of a single or even multiple servers doesn’t lead to no response (no downtime), your system is **available**.

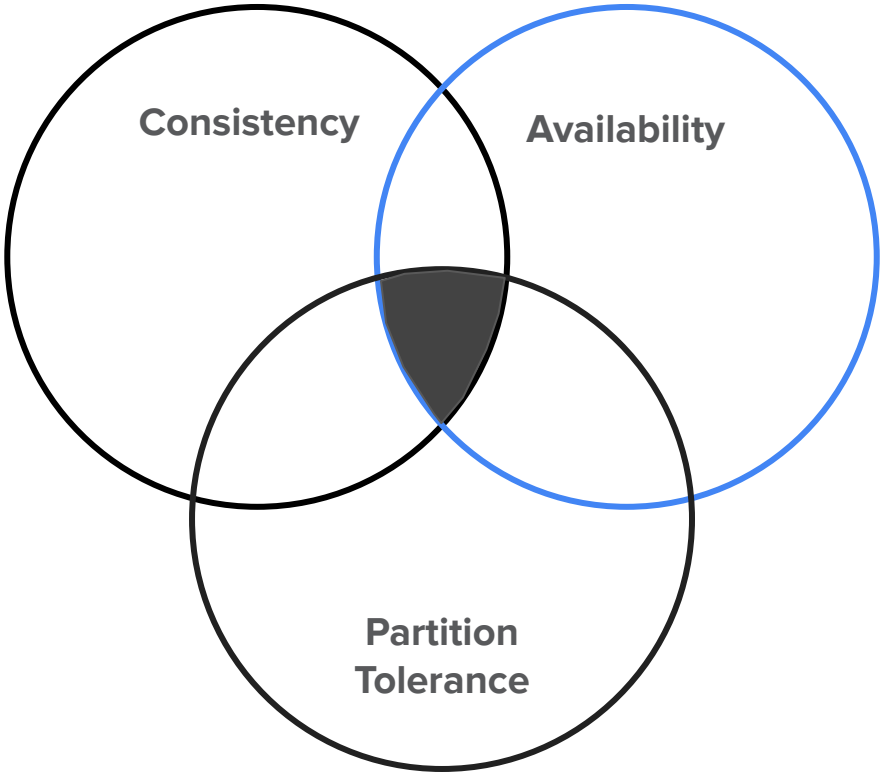
**Consistency** means “no stale data”. You ask for something, you get the most recent value. If you get outdated information, your system is **inconsistent**.



**Partition Tolerance** is the ability of a distributed system to survive “network partitioning”. Network partitioning means that the part of the servers can not reach the second part.



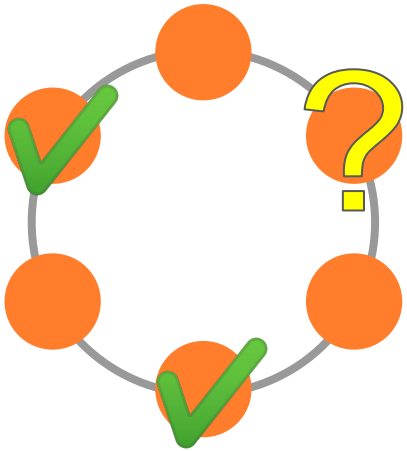
In the distributed environment **in case of emergency** you can have only two guaranteed qualities out of three :(



Cassandra is [configurably consistent.](#) In any moment of the time, for any particular query you can set the Consistency Level you require to have. It defines how many **CONFIRMATIONS** you'll wait before the response is dispatched;

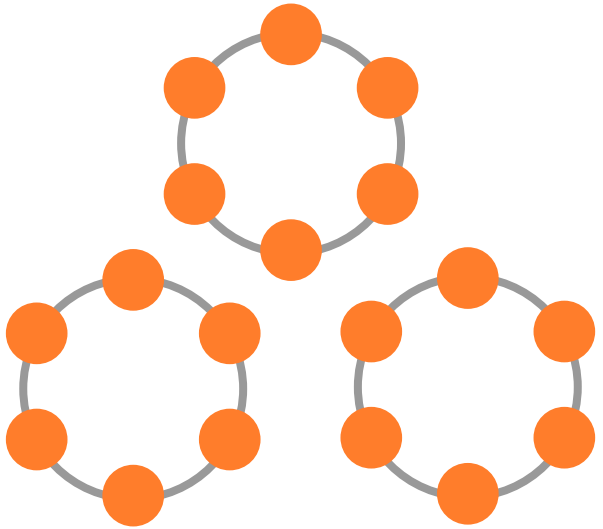
```
PreparedStatement pstmt = session.prepare(
    "INSERT INTO product (sku, description) VALUES (?, ?)"
);
pstmt.setConsistencyLevel(ConsistencyLevel.ONE);
```

```
cqlsh> CONSISTENCY

Current consistency level is QUORUM.

cqlsh> CONSISTENCY ALL

Consistency level set to ALL.
```

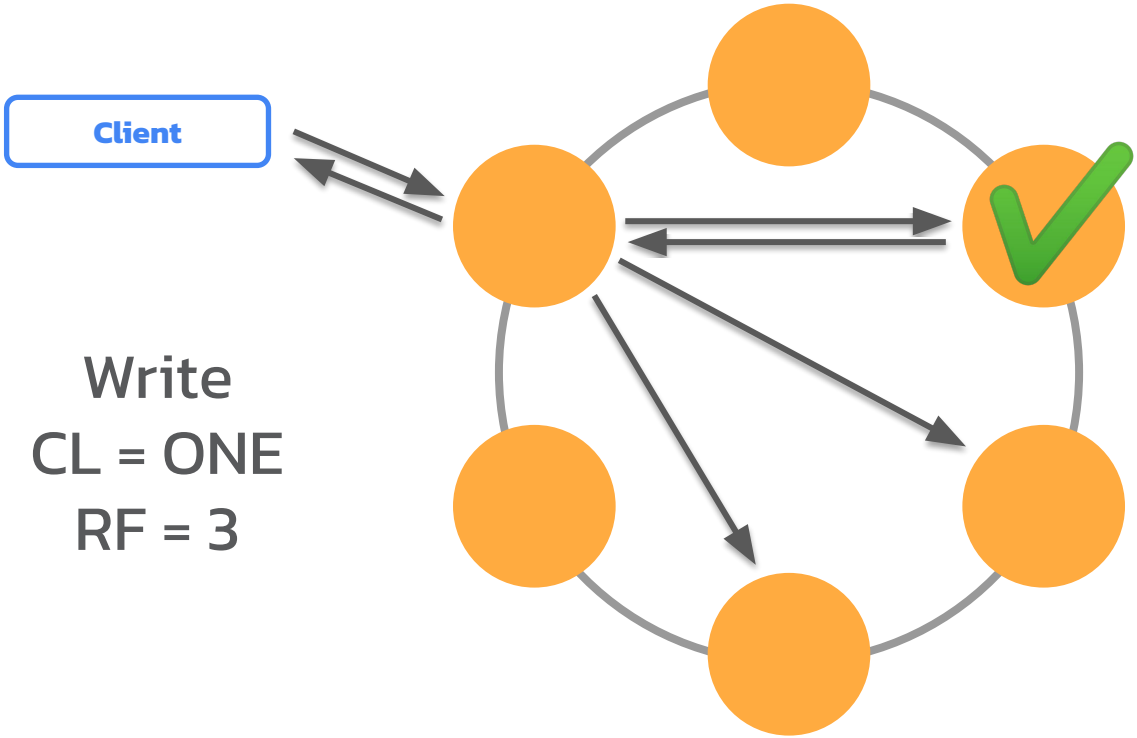


## Is Cassandra AP or CP?

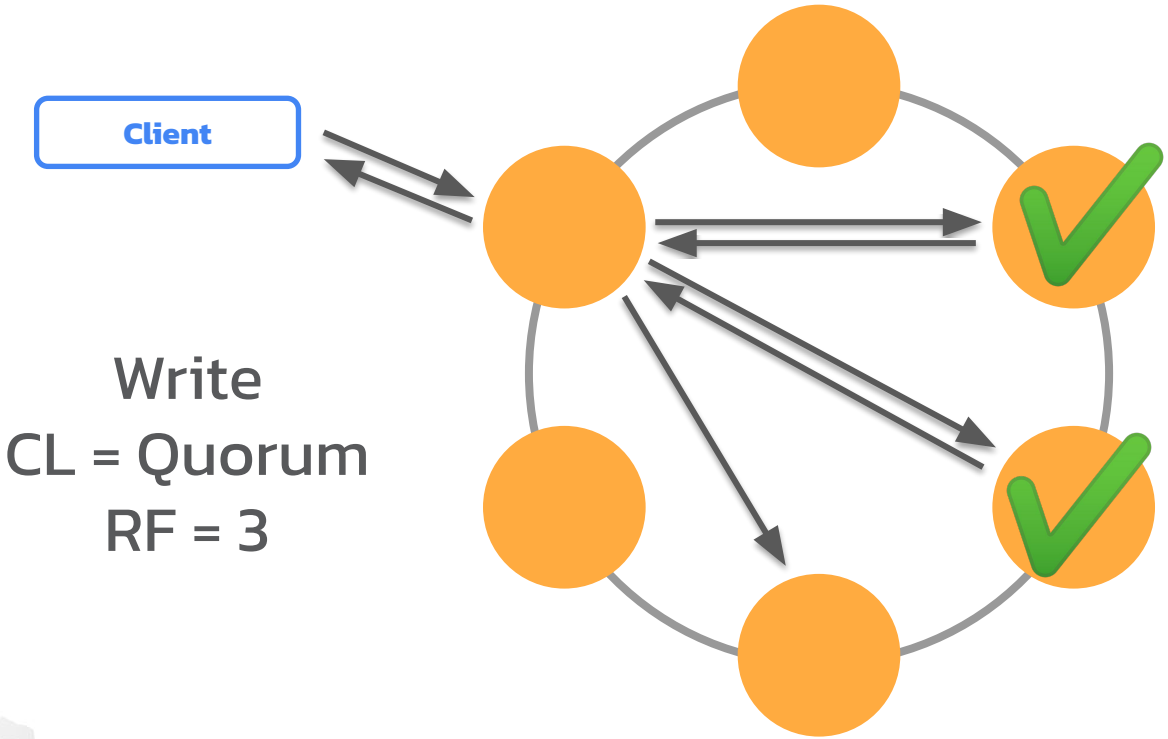
- ANY
- ONE
- LOCAL_ONE
- TWO, THREE
- QUORUM
- LOCAL_QUORUM
- EACH_QUORUM
- ALL



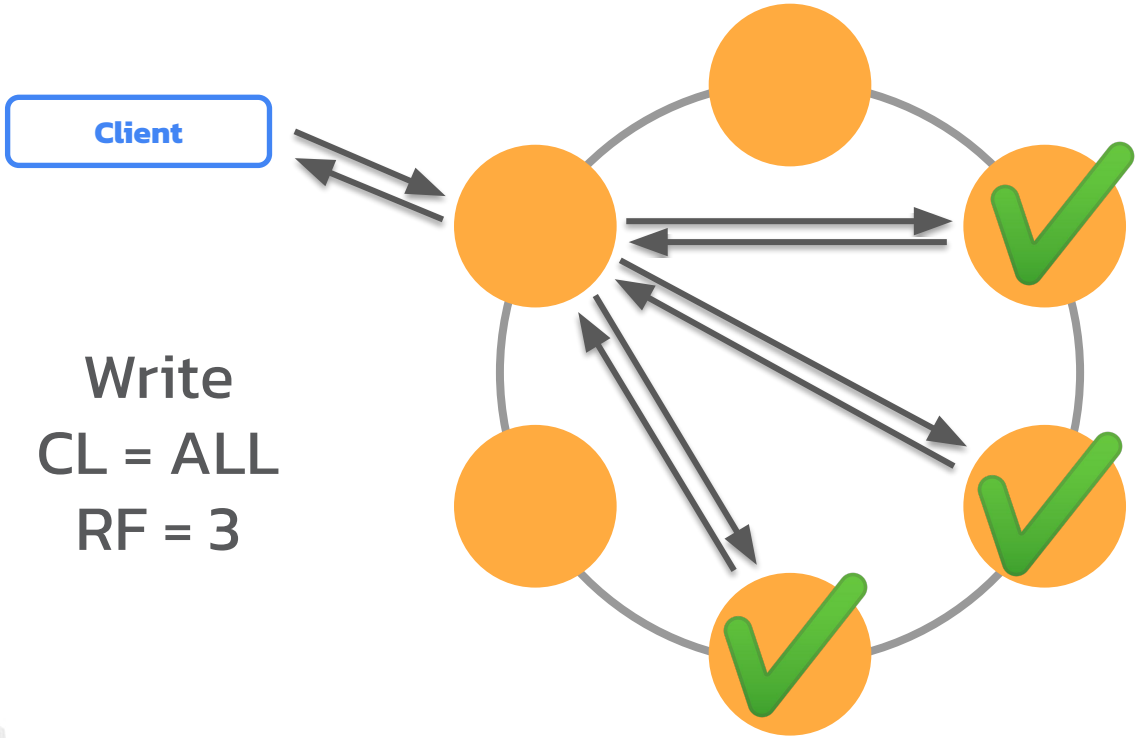
## Query Consistency Levels




## Consistency Level One



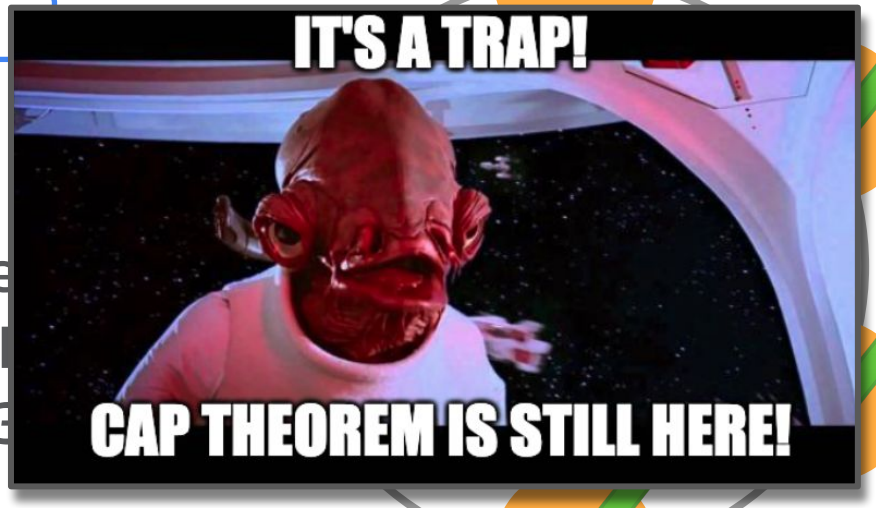

## Consistency Level Quorum



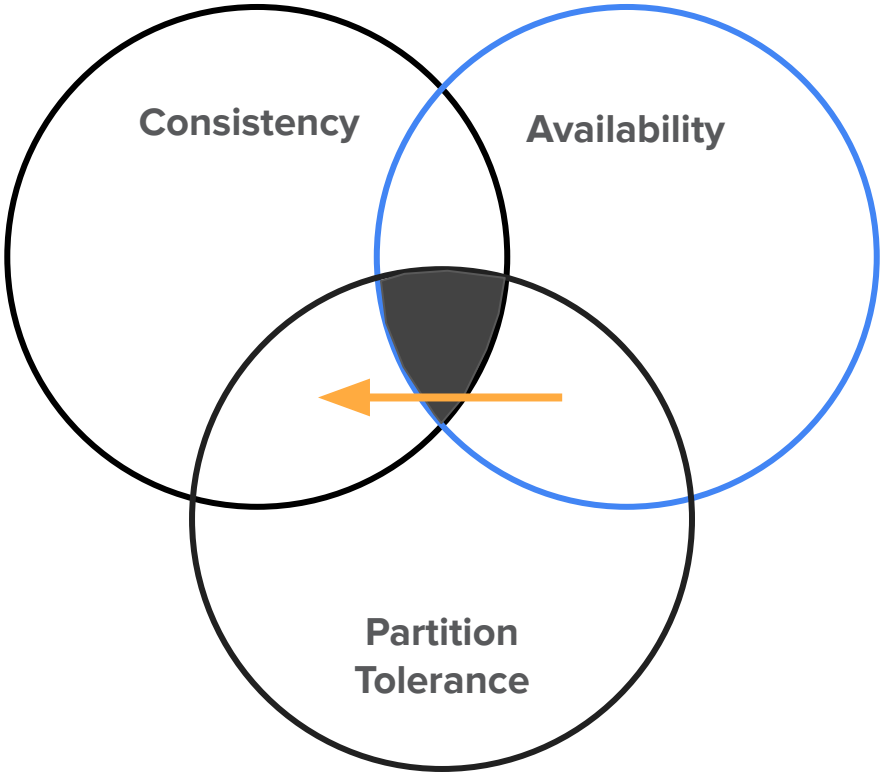

Consistency Level ALL

Client

Write
CL = A
RF = 3

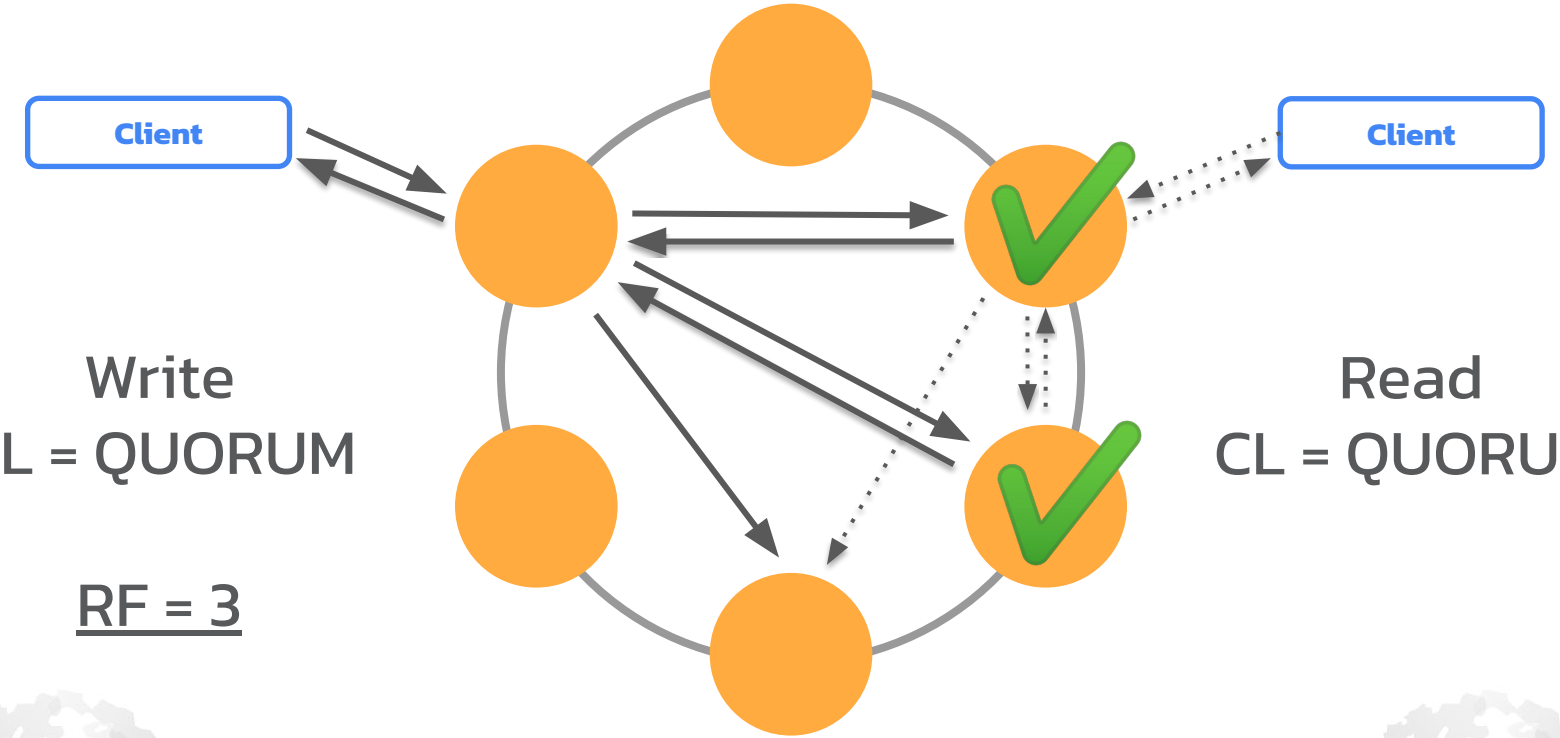



Consistency Level ALL




CL = ALL

CL Write + CL Read > RF → Immediate Consistency



# Lab 2
# Create Tables

**github.com/datastaxdevs/**
**workshop-cassandra-fundamentals**

01

Why Cassandra ?

02

Introduction to
Apache Cassandra™

03

Distributed Architecture

04

CAP Theorem

05

Tables
and Partitions

06

What's next?
Homework, Next Session

# Agenda


An intersection of a row
and a column, stores data.

| foo1 |
| --- |

## Data Structure: a Cell


A single, structured data item in a table.

| | | |
|---|---|---|
| **forest** | **f001** | **92** |

## Data Structure: a Row


A group of rows having the same partition token, a base unit of access in Cassandra.

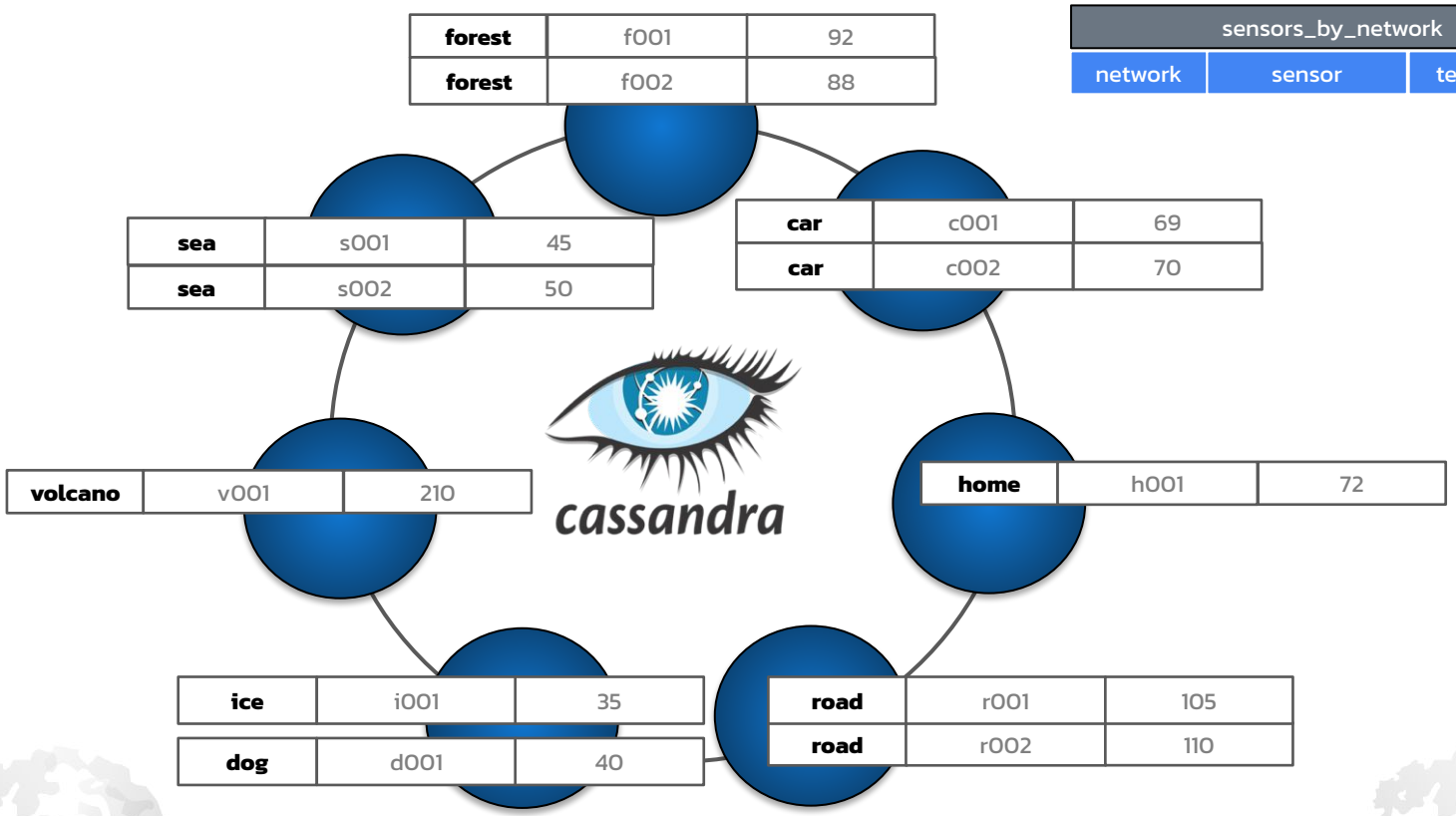
IMPORTANT: stored together, all the rows are guaranteed to be neighbors.

| | | |
|---|---|---|
| **forest** | **f001** | **92** |
| **forest** | **f002** | **88** |
| **forest** | **f003** | **90** |
| **sea** | **s001** | **45** |


A group of columns and rows storing partitions.

| network | sensor | temperature |
|---|---|---|
| **forest** | f001 | 92 |
| **forest** | f002 | 88 |
| **volcano** | v001 | 210 |
| **sea** | s001 | 45 |
| **sea** | s002 | 50 |
| **home** | h001 | 72 |
| **car** | c001 | 69 |
| **car** | c002 | 70 |
| **dog** | d001 | 40 |
| **road** | r001 | 105 |
| **road** | r002 | 110 |
| **ice** | i001 | 35 |

## Data Structure: a Table




Data is organized as Distributed tables

An identifier for a partition.
Consists of at least one column, may have more if needed.

**PARTITIONS TABLE.**
**Defines data distribution strategy**

```
CREATE TABLE sensor_data.temperatures_by_sensor (
  sensor     TEXT,
  date       DATE,
  timestamp  TIMESTAMP,
  value      FLOAT,
  PRIMARY KEY ((sensor, date), timestamp)
);
```

Partition key

Examples:

```
PRIMARY KEY ((sensor), timestamp);
```

```
PRIMARY KEY ((sensor), date, timestamp);
```

```
PRIMARY KEY ((sensor, timestamp));
```

Partition Key

Used to **ensure uniqueness** and **sorting order.** Optional.

**Defines physical ordering on disk**

```
CREATE TABLE sensor_data.temperatures_by_sensor (
  sensor      TEXT,
  date        DATE,
  timestamp   TIMESTAMP,
  value       FLOAT,
  PRIMARY KEY ((sensor, date), timestamp)
) WITH CLUSTERING ORDER BY (timestamp DESC);
```

Clustering columns

```
PRIMARY KEY ((sensor));
```
Not Unique 

```
PRIMARY KEY ((sensor), timestamp);
```
Not filter on dates 

```
PRIMARY KEY ((sensor), value, timestamp);
```
Not sorted 

```
PRIMARY KEY ((sensor), date, timestamp);
```


## Clustering Columns

An identifier for a row. Consists of at least one Partition Key and zero or more Clustering Columns.

**UNIQUELY IDENTIFIES A ROW**

```
CREATE TABLE sensor_data.temperatures_by_sensor (
  sensor     TEXT,
  date       DATE,
  timestamp  TIMESTAMP,
  value      FLOAT,
  PRIMARY KEY ((sensor, date), timestamp)
) WITH CLUSTERING ORDER BY (timestamp DESC);
```

Primary key

Examples:

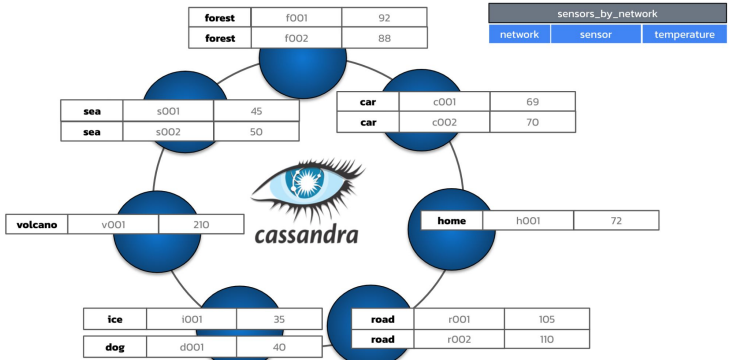
```
PRIMARY KEY ((sensor), timestamp);
```

```
PRIMARY KEY ((sensor, date), timestamp);
```

**Primary Key**

## Partition Keys

define data distribution over the cluster




## Clustering Columns

define how data is physically stored (written on disk)

```
{
  "partition" : {
    "key" : [ "y58dbfa0-b87f-11ea-ab9b-f55dd84a27b8" ],
    "position" : 131
  },
  "rows" : [
    {
      "type" : "row",
      "position" : 261,
      "liveness_info" : { "tstamp" : "2021-02-26T14:59:18.322385Z" },
      "cells" : [
        { "name" : "createdt", "value" : "2020-06-28 11:00:00.000" },
        { "name" : "publish_date", "value" : "2020-06-28 11:00:00.000Z" },
        { "name" : "title", "value" : "Title02" },
        { "name" : "user_email", "value" : "placeholder@email.com" },
        { "name" : "user_name", "value" : "name01" }
      ]
    }
  ]
},
{
  "partition" : {
    "key" : [ "c58dbfa0-b87f-11ea-ab9b-f55dd84a27b8" ],
    "position" : 262
  },
  "rows" : [
    {
      "type" : "row",
      "position" : 390,
      "liveness_info" : { "tstamp" : "2021-02-26T14:59:17.893212Z" },
      "cells" : [
        { "name" : "createdt", "value" : "2020-06-28 11:00:00.000" },
        { "name" : "publish_date", "value" : "2020-06-28 11:00:00.000Z" },
        { "name" : "title", "value" : "Title01" },
        { "name" : "user_email", "value" : "placeholder@email.com" },
        { "name" : "user_name", "value" : "name01" }
      ]
    }
  ]
}
```

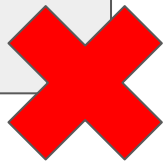
**Primary Key = Ensure Uniqueness = PK + CC**

**Important:**

Once created, the data model cannot be changed! You will need new tables and migration. Stay lazy, design it right in advance!

```
ALTER TABLE temperatures_by_sensor
ADD season TEXT;
```

```
ALTER TABLE temperatures_by_sensor
DROP PRIMARY KEY
ADD PRIMARY KEY (sensor, timestamp)
```



# Lab 3
# CRUD Operations

**github.com/datastaxdevs/workshop-cassandra-fundamentals**

That’s step I of your homework!

01

Why Cassandra ?

02

Introduction to
Apache Cassandra™

03

Distributed Architecture

04

CAP Theorem

05

Tables
and Partitions

06

What's next?
Homework, Next Session

# Agenda

# Homework


## Intro to Cassandra Homework

Welcome and thank you! Here you can submit your homework for the datastax developers "Intro to Cassandra" workshop. In case of any questions please contact organisers at https://dtsx.io/aleks or just send an email to aleksandr.volochnev@datastax.com

- Workshop materials: https://github.com/datastaxdevs/workshop-intro-to-cassandra
- Discord chat: https://dtsx.io/discord

cedrick.lunven@datastax.com Switch account

The name and photo associated with your Google account will be recorded when you upload files and submit this form. Only the email you enter is part of your response.

* Required

Apache CASSANDRA


# Thank you!

**Sponsored by DataStax**