ICISN 2023

DATASTAX DEVELOPERS

Cassandra Training Workshop Student Akathon

Build Cloud Native applications

SWINBURNE UNIVERSITY OF TECHNOLOGY

Alliance with FPT Education

❯ Cédrick Lunven

Director of Developer Relations

clunven

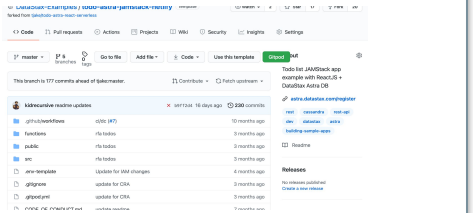
clunven

clun

- Trainer
- Public Speaker
- Developers Support
- Developer Applications
- Developer Tooling
- Creator of ff4j (ff4j.org)
- Maintainer for 8 years+
- Implementing APIs for 8 years

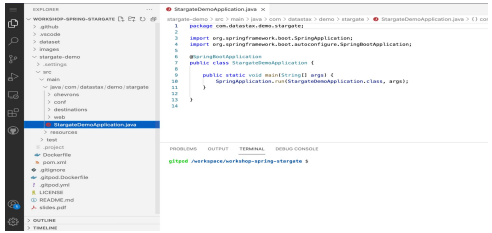
Hands-on housekeeping

Source code + exercises + slides



GitHub !github

IDE

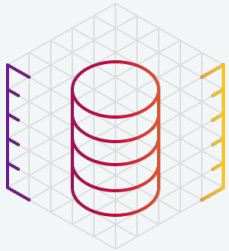


Gitpod !gitpod

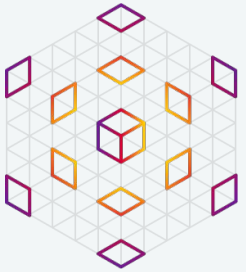
Database + Api + Streaming

DATASTAX ASTRA DB !astra

Nothing to install !



Agenda



01
Introduction to NoSQL
Why, what, how

02
Getting Started with
Apache Cassandra

03
Data Modeling
Application design

04
Application Developmen
Python and Java

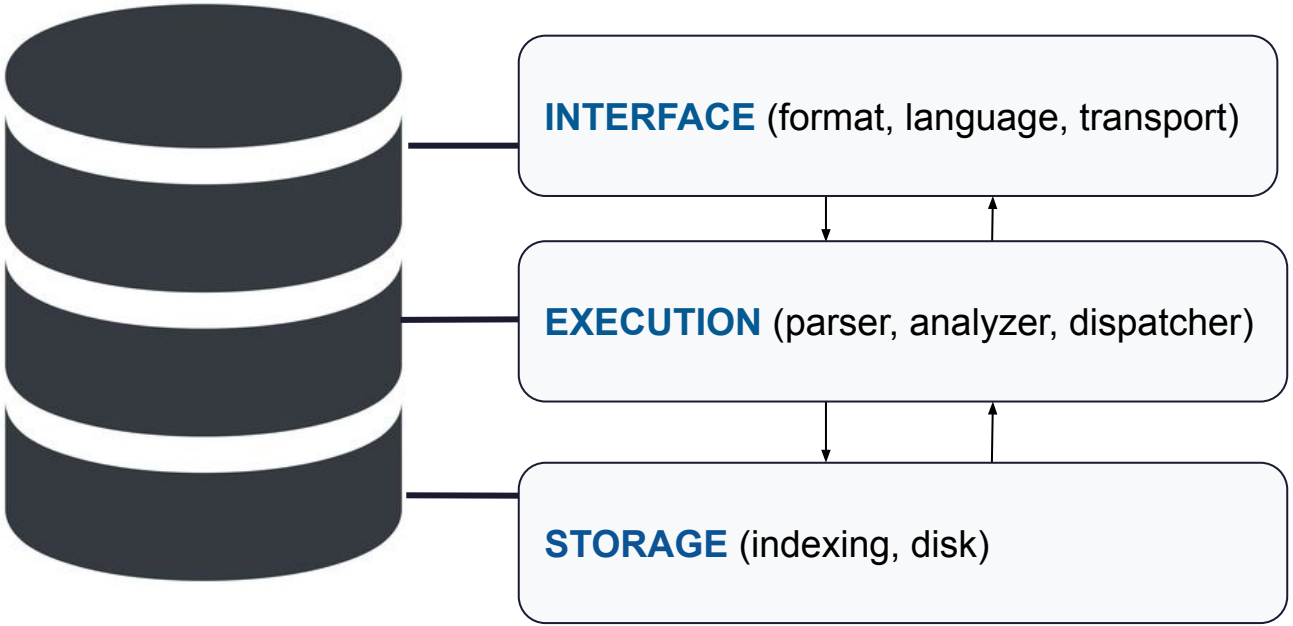
05
Stargate Apis
Cloud Native

06
What's next?
Homework, next sessions

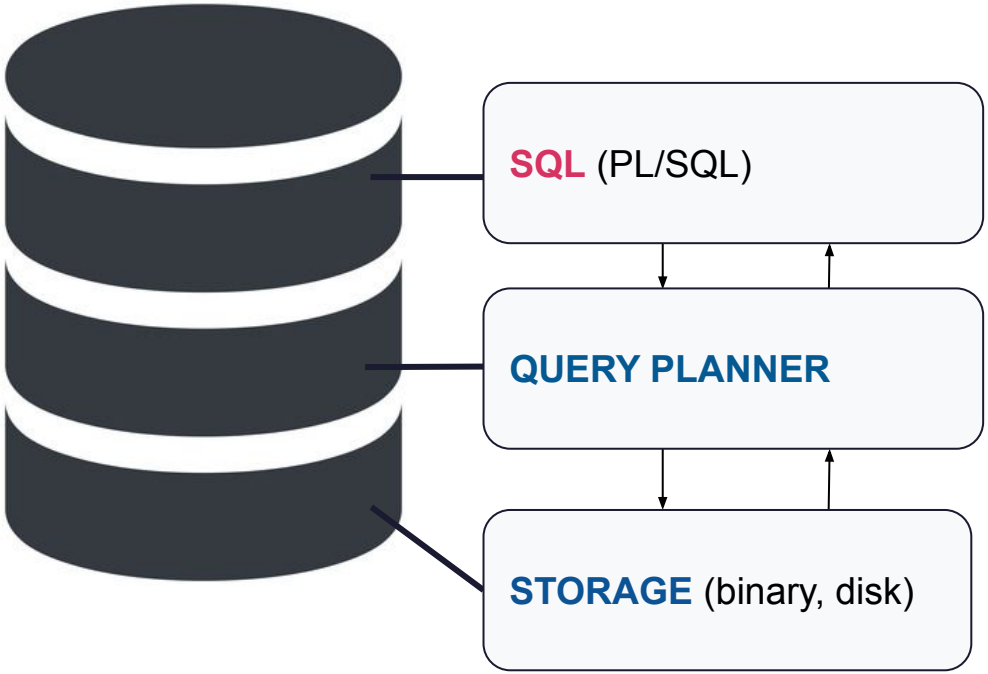
What is a Database?

- **An organized collection of related data items**
- **Database Management System (DBMS)**

Introducing to Databases



Introducing to No SQL Databases



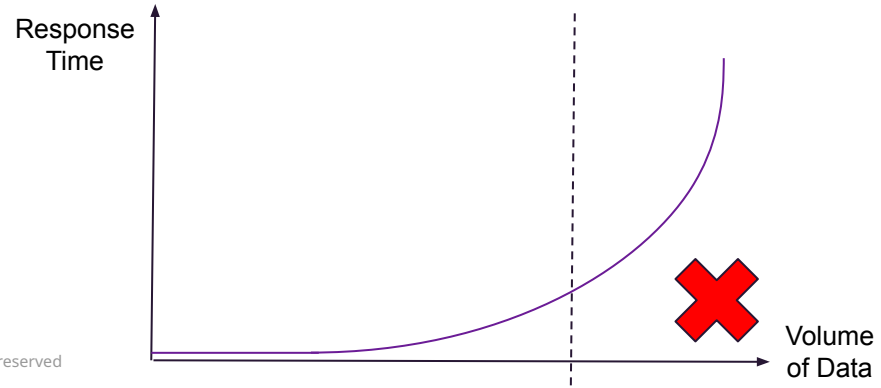
Relational Databases are versatile

OLTP: Online Transaction Processing

Fast Processing
Transactional
High number of transactions
Hot / Live Data
“Row oriented”
Normalized Data (3NF)

OLAP: Online Analytical Processing

Slow Queries
Historical
High volume of data
Cold Data
“Column oriented”
Denormalized Data

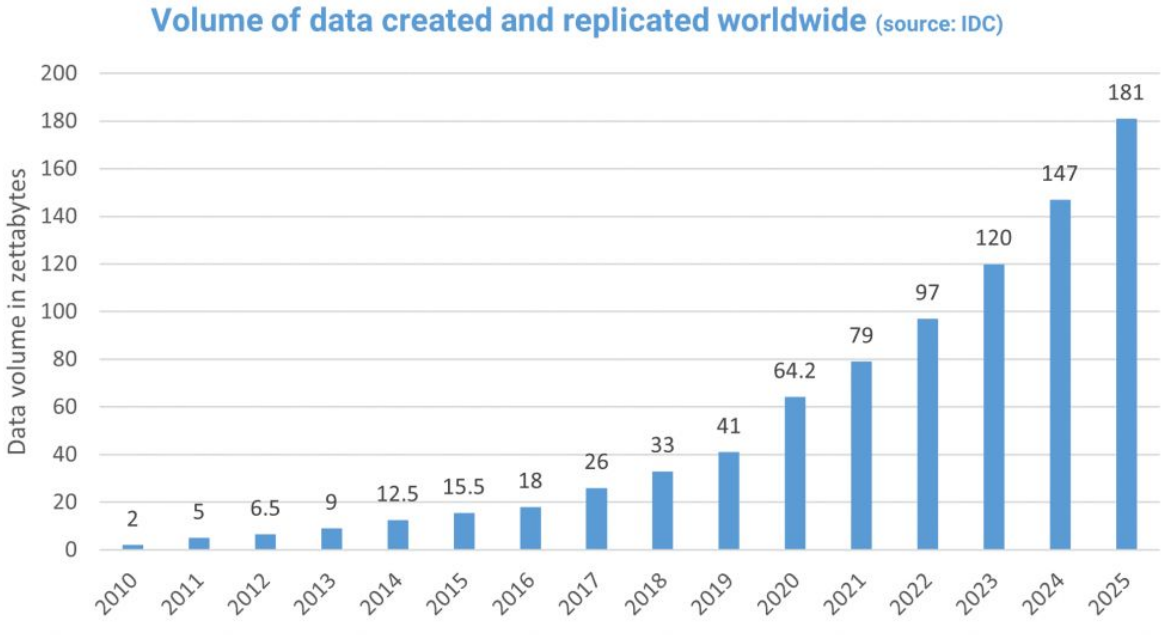


› Relation Databases have limited scalability

- They have been designed to run on a single machine

- New requirements
  - V: VOLUME
  - V: VELOCITY
  - V: VARIETY

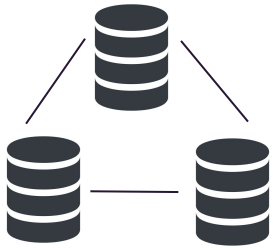
WHAT'S A ZETTABYTE?	
1 kilobyte	1,000
1 megabyte	1,000,000
1 gigabyte	1,000,000,000
1 terabyte	1,000,000,000,000
1 petabyte	1,000,000,000,000,000
1 exabyte	1,000,000,000,000,000,000
1 zettabyte	1,000,000,000,000,000,000,000



Introducing NoSQL

- Meetup names on June 11, 2009 in San Francisco

- Web Giants to needed more scalable systems
  - Distributed by design (horizontal scalability)
  - Data is replicated
  - Cope with the variety of data

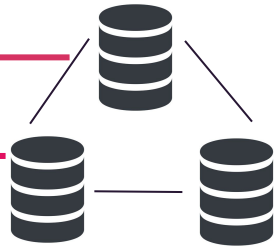


Not Only SQL

NoSQL Databases

AP vs CP (CA = not distributed)

Server vs **VM** vs **Container** vs **Cloud**



SQL or **JSON** or **CQL** or **N1QL** or **Cypher...**

Query Parser and execution engine

Text, Binary, Graph

Ledger Databases *awsqldb, ksql_db*	Time-Series Databases *Influx, OpenTSDB, Prometheus*	Tabular Databases ***Cassandra**, Hbase, Bigtable*	Object Databases *S3, Minio, Ceph*
Document Databases *MongoDB, Elastic, DocDb*	Graph Databases *Neo4j, Datastax Graph, Titan*	Key/Value Databases *Redis, Dynamo, Memcache*	

4 Types of NoSQL Databases

Tabular or wide-column database

Document database

Key-value database

Graph database

Astra DB

Couchbase

mongoDB.

redis

DynamoDB

DataStax Graph

JanusGraph

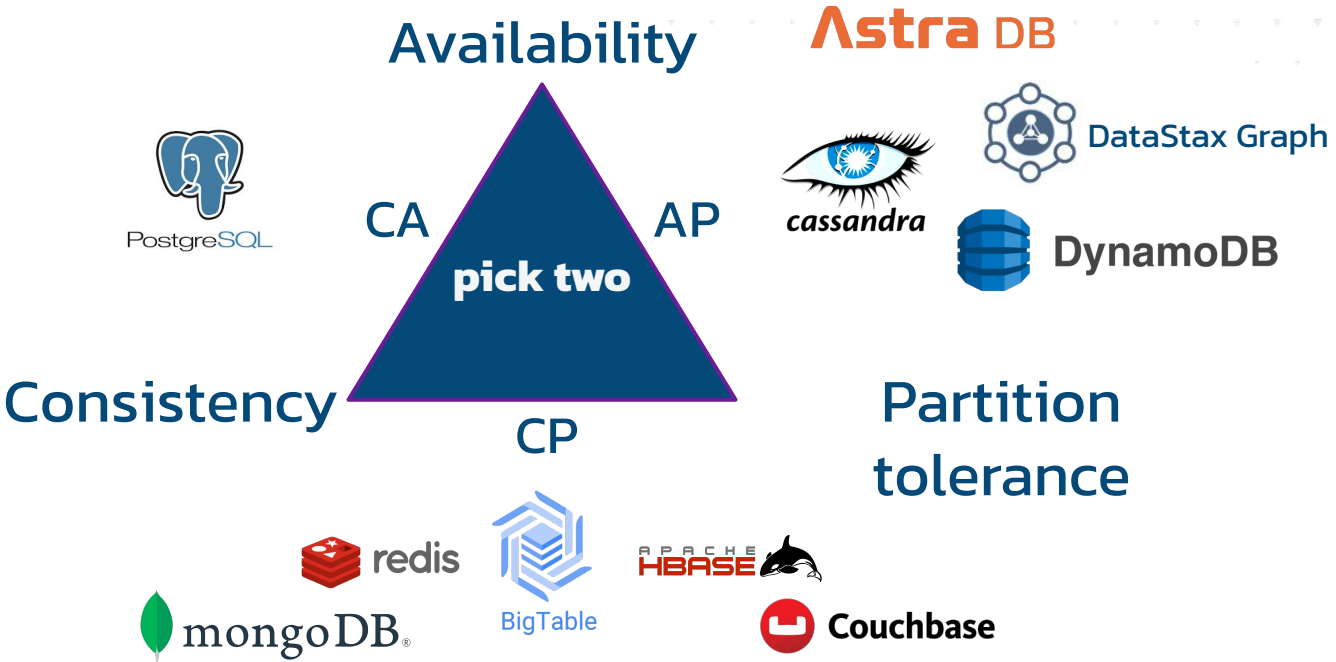
The CAP Theorem

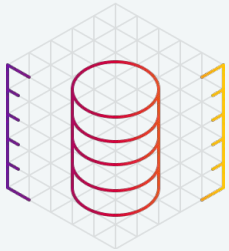
Always responds,
may not always return
the most recent write

Every read receives
the most recent write
or an error

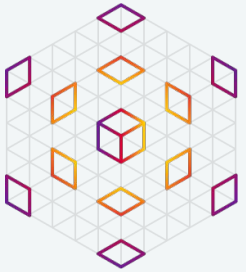
Operates in the presence
of network partition
failures

The CAP Theorem





Agenda



01

Introduction to NoSQL
Why, what, how

02

Getting Started with Apache Cassandra

03

Data Modeling
Application design

04

Application Developmen
Python and Java

05

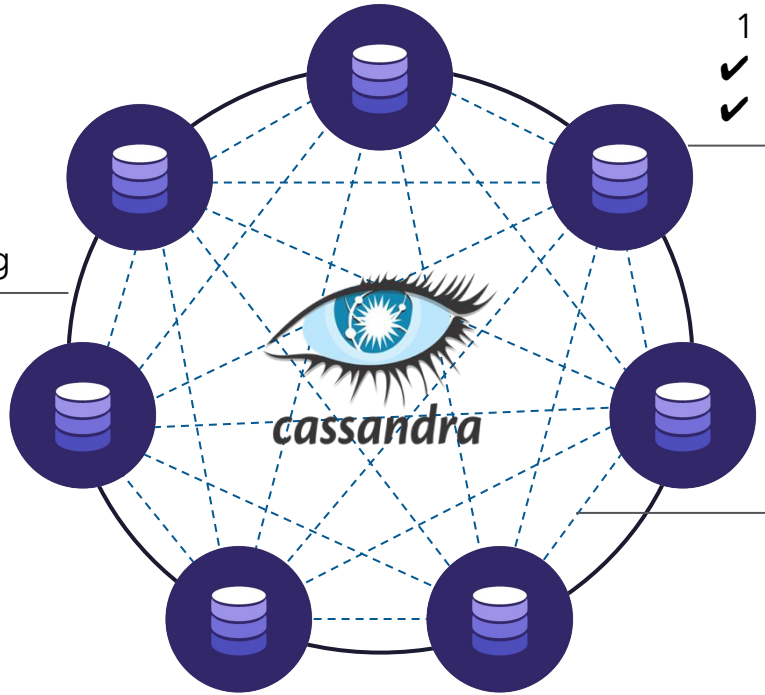
Stargate Apis
Cloud Native

06

What's next?
Homework, next sessions

› **Nosql** Distributed database

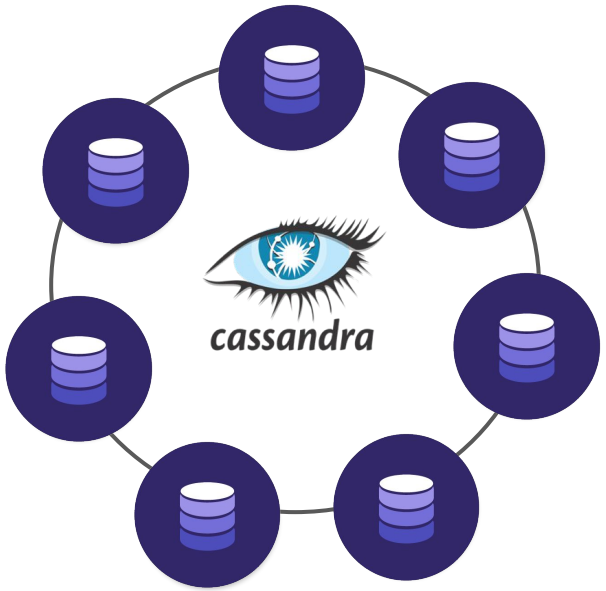
DataCenter (DC) | Ring



1 Installation = 1 NODE
- ✔ Capacity = ~ 2-4TB
- ✔ Throughput = LOTS Tx/sec/core

Communication:
- ✔ Gossiping
- ✔ No Master (peer-to-peer)

› **Nosql** Distributed database

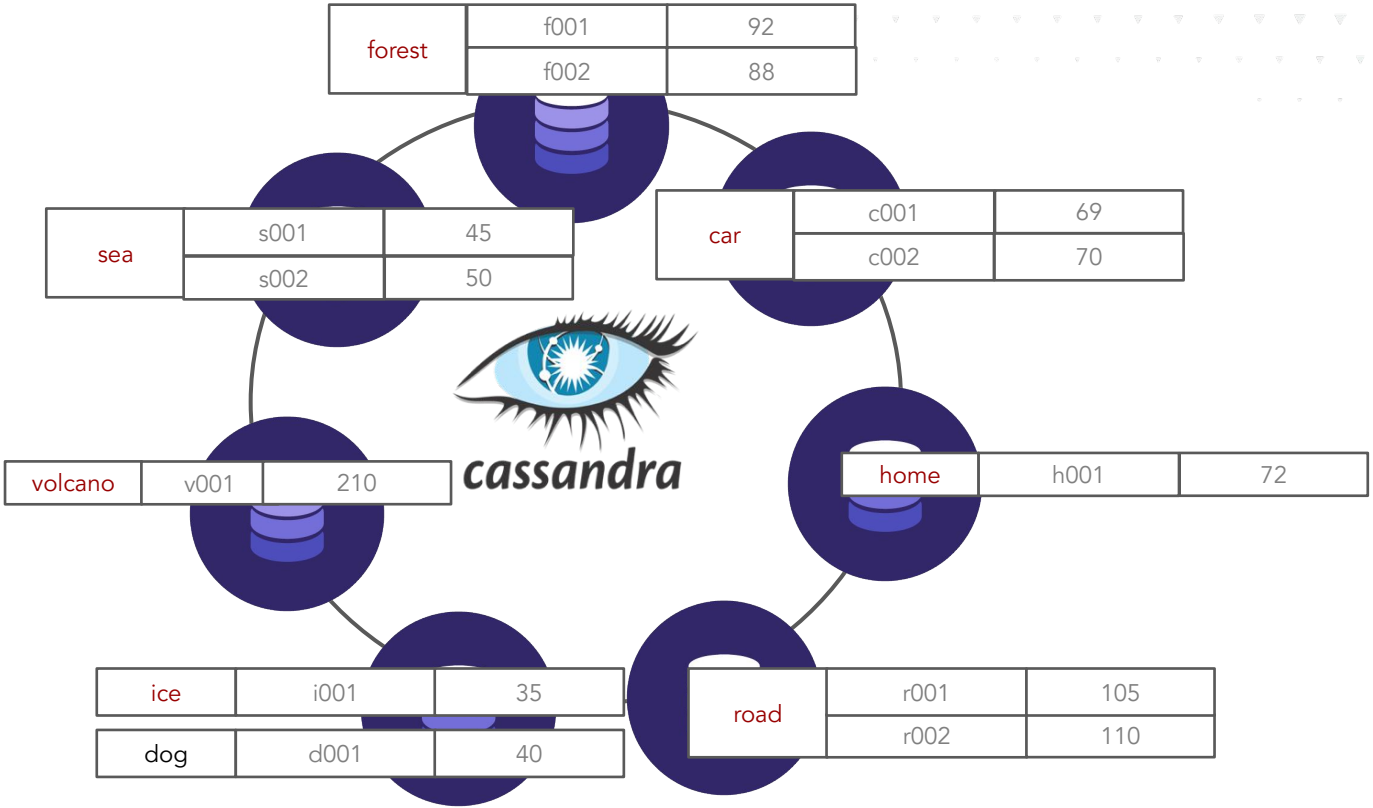


sensors_by_network		
network	sensor	temperature
forest	f001	92
forest	f002	88
volcano	v001	210
sea	s001	45
sea	s002	50
home	h001	72
car	c001	69
car	c002	70
dog	d001	40
road	r001	105
road	r002	110
ice	i001	35

Partition Key

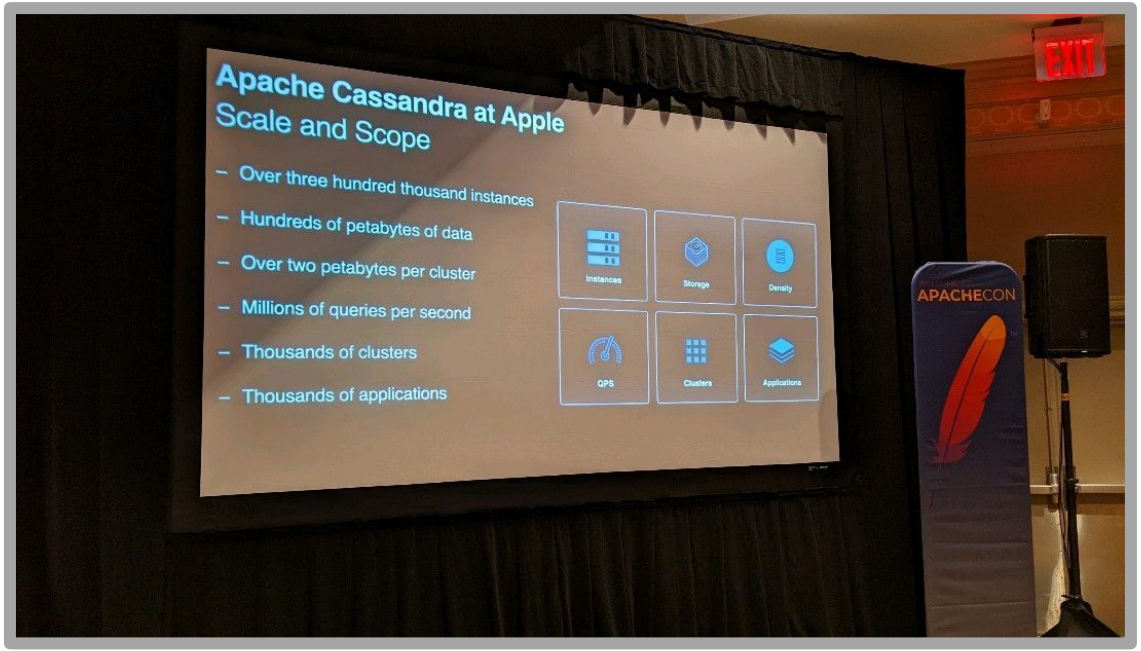
Primary Key

Distributed



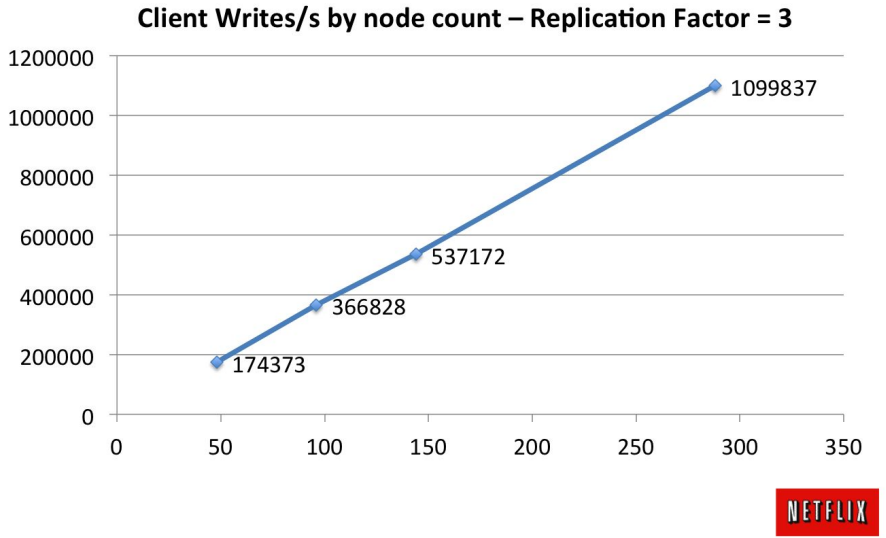
Than can scalability…a LOT

300 000+ nodes



Linear Scalability

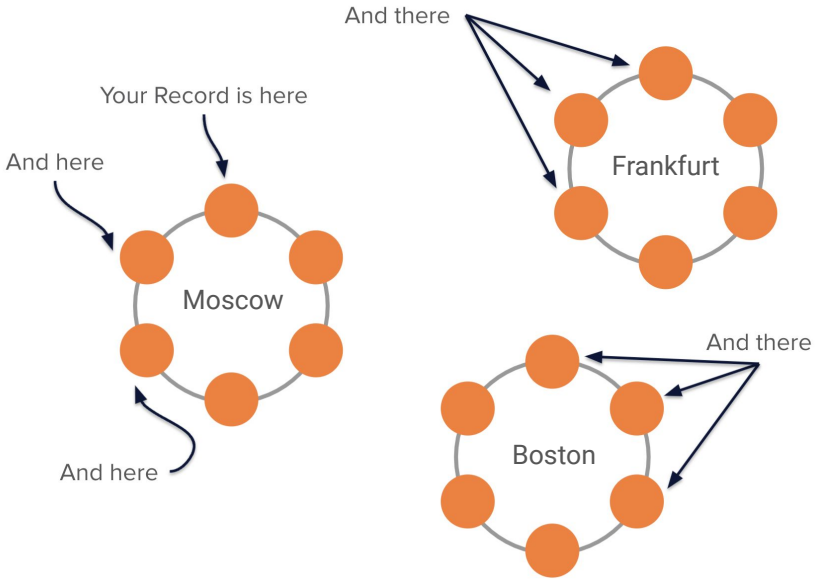
- For volume or velocity, there are no limitations
- **Linear** - No overhead on new nodes, scales with your needs*



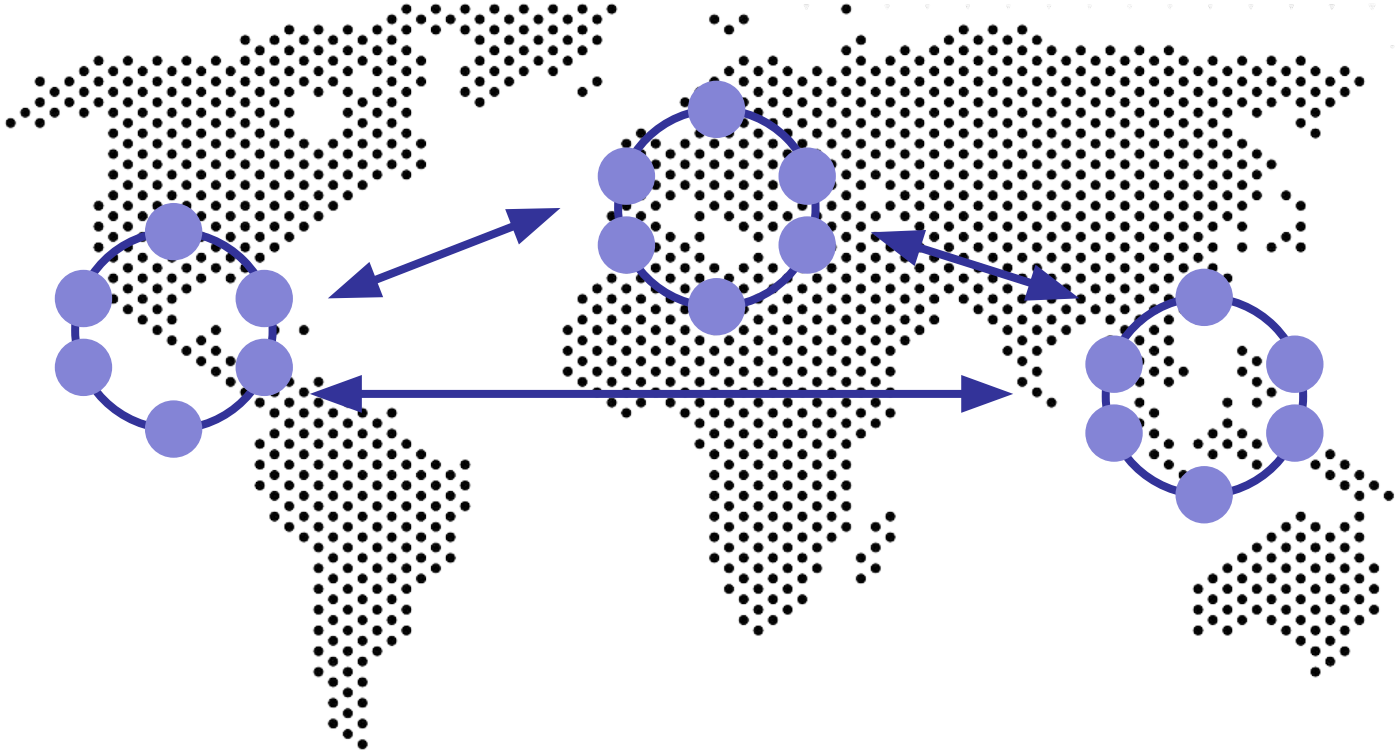
Highest Availability

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

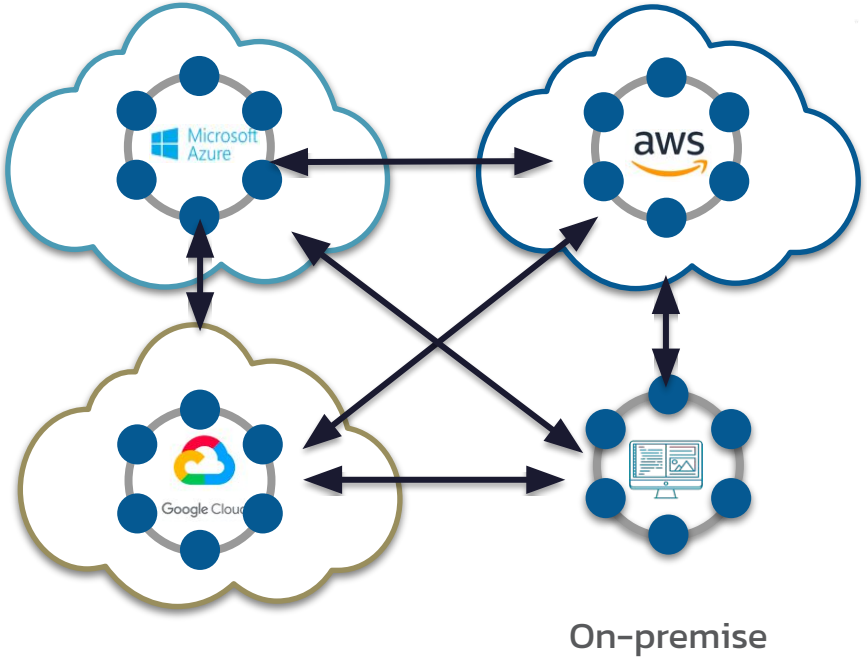


› **Geographically** distributed



Platform Agnostic

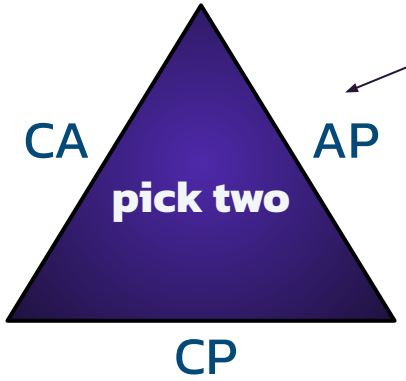
Apache Cassandra is **not bound to any platform** or service provider, helping you build hybrid-cloud and multi-cloud solutions with ease.



The CAP Theorem

Always responds, may not always return the most recent write

Availability



Consistency

Every read receives the most recent write or an error

Partition tolerance

Operates in the presence of network partition failures

Apache Cassandra

High Availability

Always On

Every second of downtime translates into lost revenue

Linear Scalability

Hyper Scalability

Millions of operations per day, hour, or second

Low Latency

Faster Pace

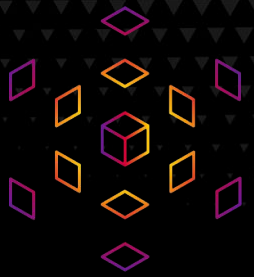
Every millisecond of latency has consequence

Global Distribution

Data Everywhere

On-premises, hybrid, multi-cloud, centralized, or edge

Cassandra..in the CLOUD



DATASTAX

ASTRA DB

Astra DB design goals

Deliver the power of Apache Cassandra
Extend and Improve Cassandra Functionality

Cloud-native, multi-cloud and as-a-service

Fine-grained auto scaling, Low latency

Developer flexibility & productivity

Zero Ops. HA with Zero Downtime, Cost Effective

Multi-cloud
Database-as-a-Service (DBaaS)
built on Apache Cassandra

Fast Time-to-Market
Cloud delivered, easy provisioning, no capacity planning/sizing. Multi-tenancy built-in.

Zero Lock-In
Deploy on any cloud: AWS, Azure & GCP

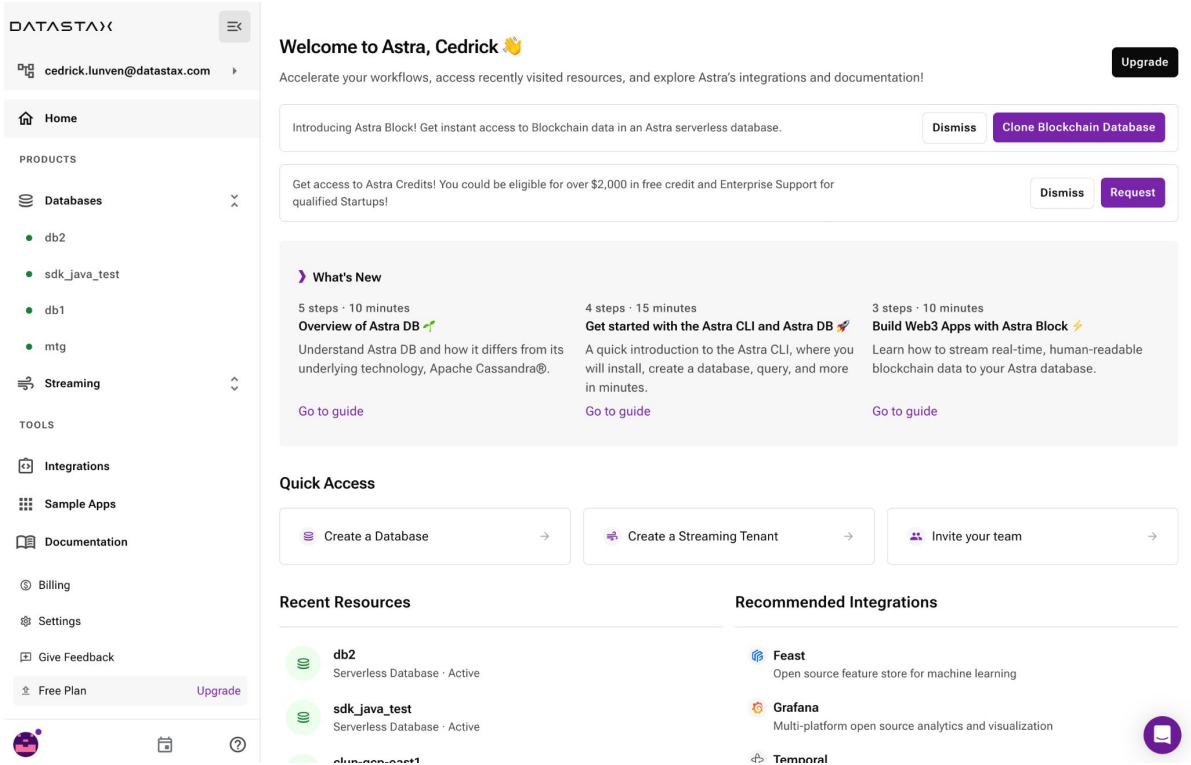
Dynamic Elasticity, High Availability
No over- or under-provisioning;
automatically scales to accommodate peaks & troughs.
Highly Available, Resilient and fault-tolerant

Developer Velocity
Accelerate modern application development via commonly used APIs & a flexible data model

Zero Operational Overhead
Focus on driving business value;
no managing infrastructure or operations

True Consumption-Based Billing
i.e., reads, writes, data transfer and storage based billing

Astra DB



DATASTAX | ASTRA

CQL API
Compatibility
It is Apache Cassandra !

PULSAR CLIENT
Compatibility
It is Apache Pulsar !

GRAPHQL API
Discoverability, Flexibility
Read only what you need

JAVA MESSAGING SYSTEM
Interoperability
Plug any Existing Java Application

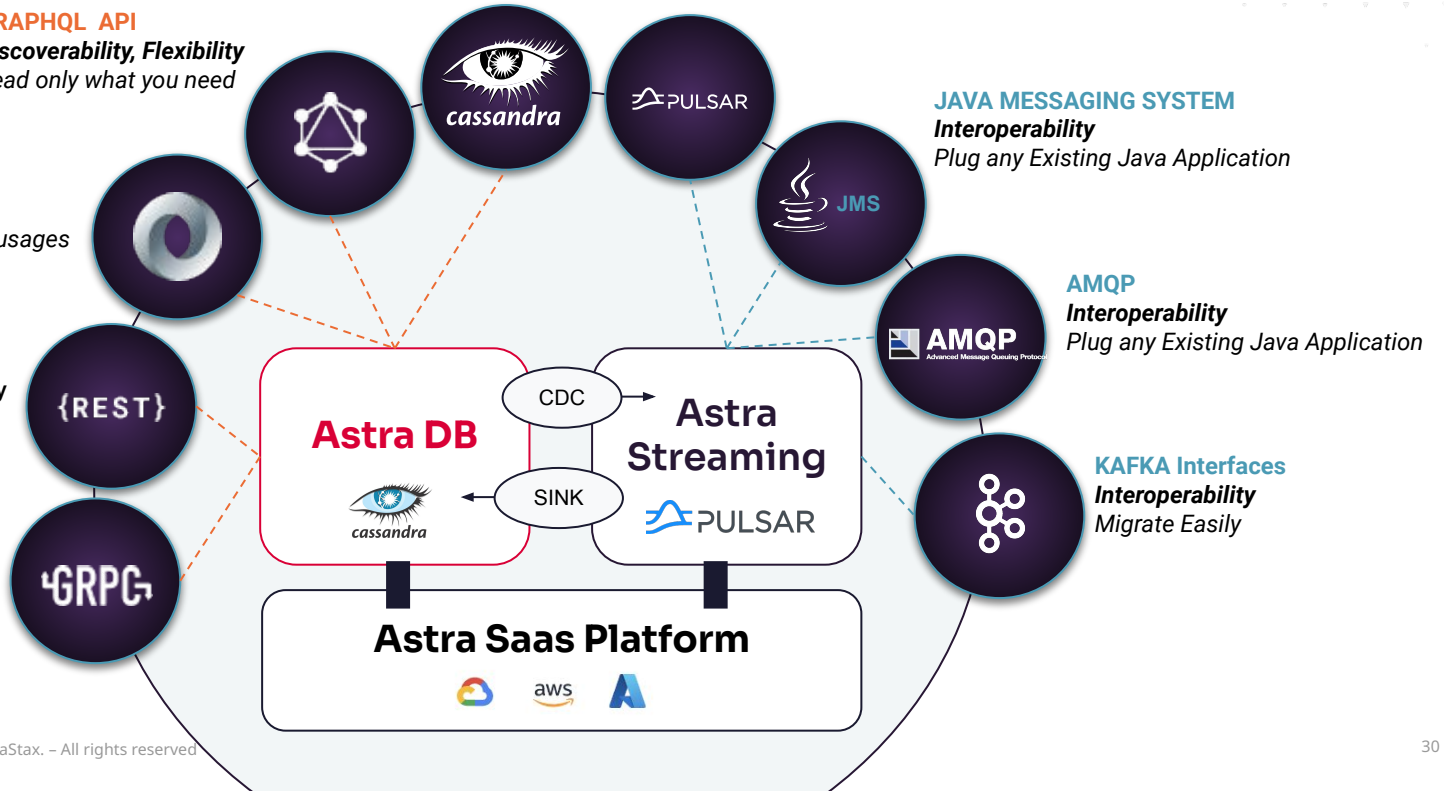
DOCUMENT API
Flexible Schema
Document-Oriented usages

AMQP
Interoperability
Plug any Existing Java Application

REST API
Stateless, Interoperability
CRUD for Cassandra

KAFKA Interfaces
Interoperability
Migrate Easily

GRPC API
Performance, Interoperability
Execute cql in the fastest way



Lab 1

Environment Initialization

1.1 Create Astra Instance

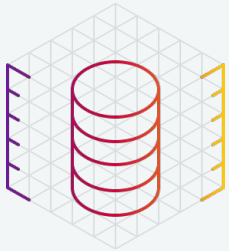
1.2 Create Astra Token

1.3 Open environment

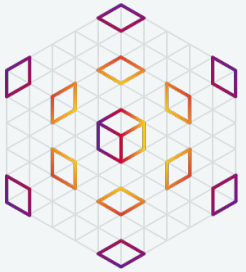
1.4 Setup Astra CLI

1.5 Create Database

github.com/datastaxdevs/workshop-swinburne



Agenda



01

Introduction to NoSQL
Why, what, how

02

Getting Started with Apache Cassandra

03

Data Modeling
Application design

04

Application Developmen
Python and Java

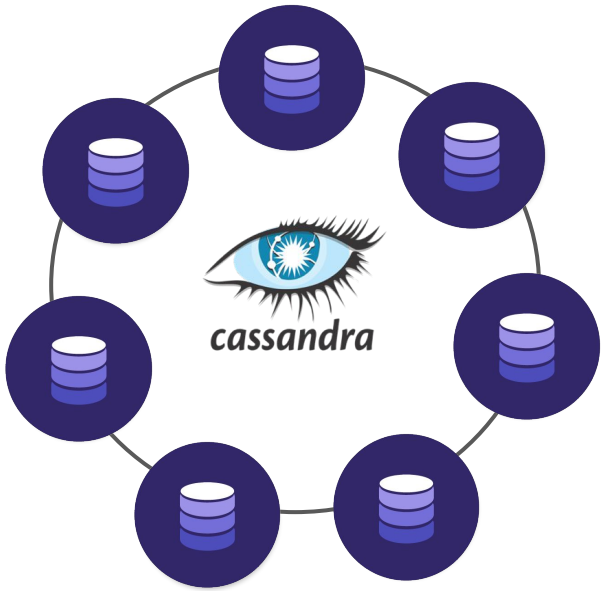
05

Stargate Apis
Cloud Native

06

What's next?
Homework, next sessions

› **Nosql** Distributed database

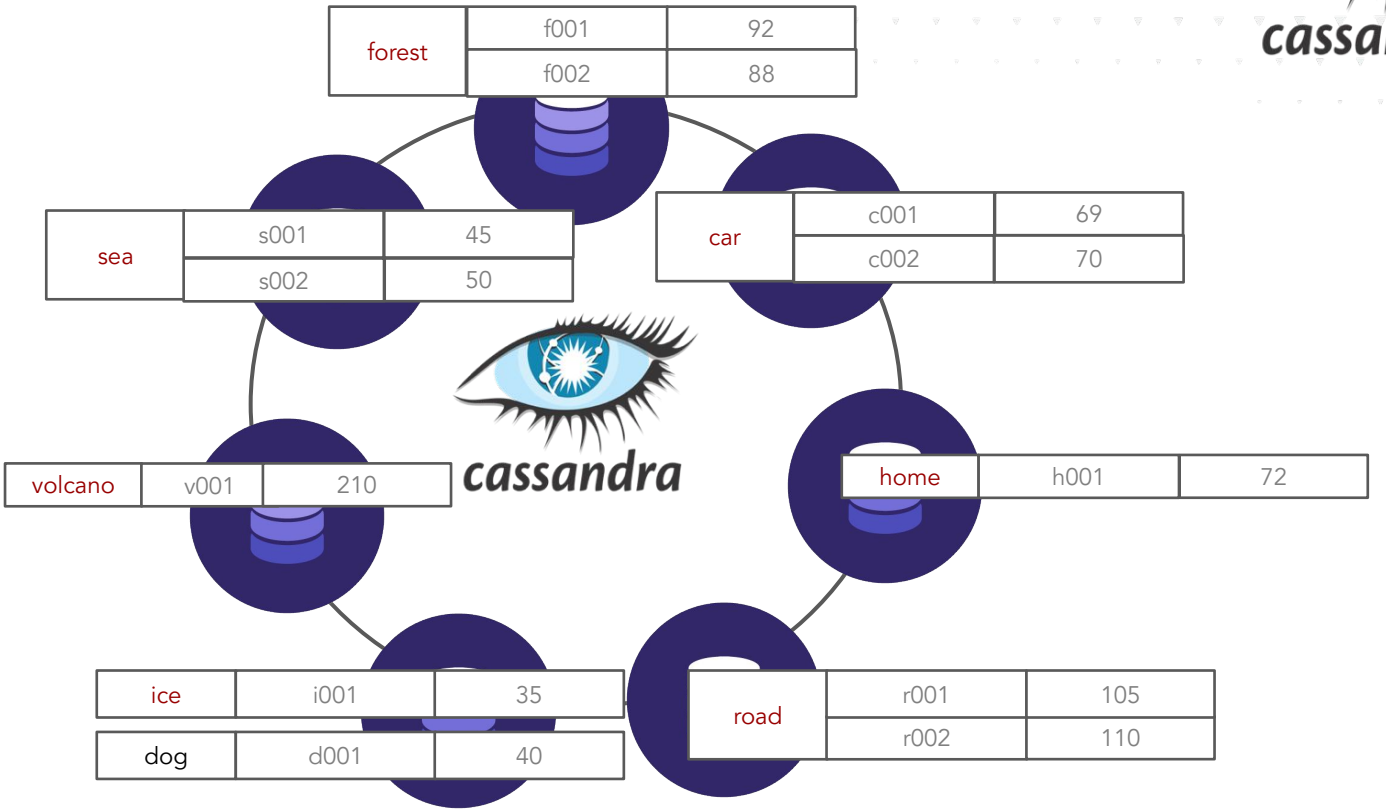


sensors_by_network		
network	sensor	temperature
forest	f001	92
forest	f002	88
volcano	v001	210
sea	s001	45
sea	s002	50
home	h001	72
car	c001	69
car	c002	70
dog	d001	40
road	r001	105
road	r002	110
ice	i001	35

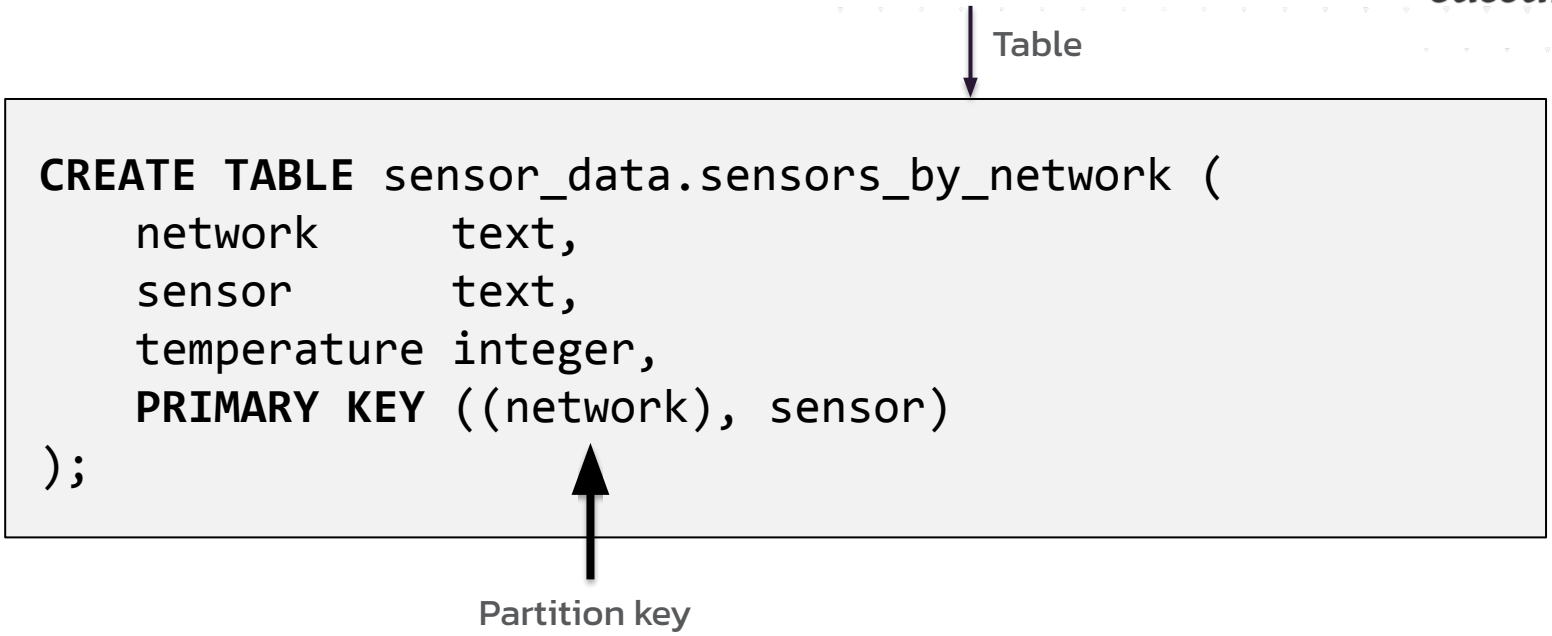
Partition Key

Primary Key

Distributed



Partition key definition in CQL



Internals: Partitioning and Token Ranges

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

Partition Keys

Partitioner

Murmur3 Hashing

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

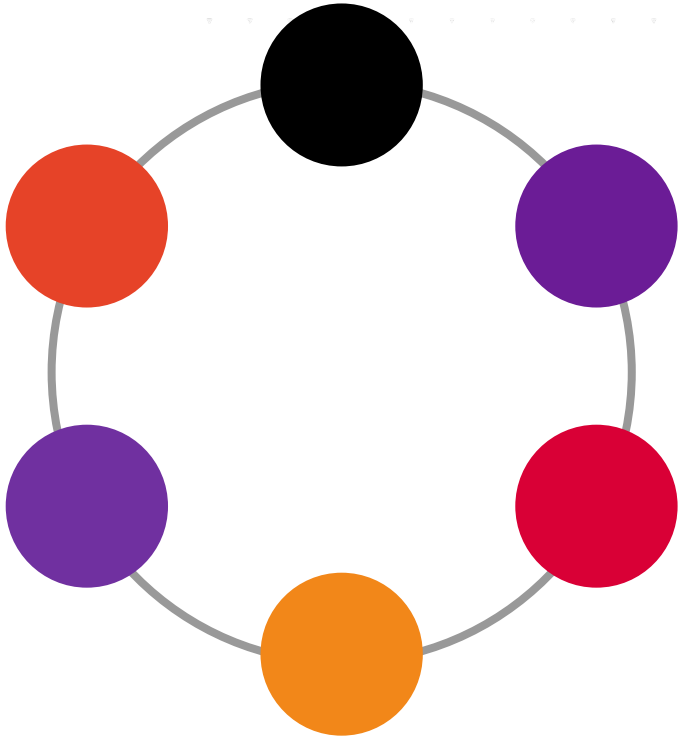
Tokens

Cassandra Nodes

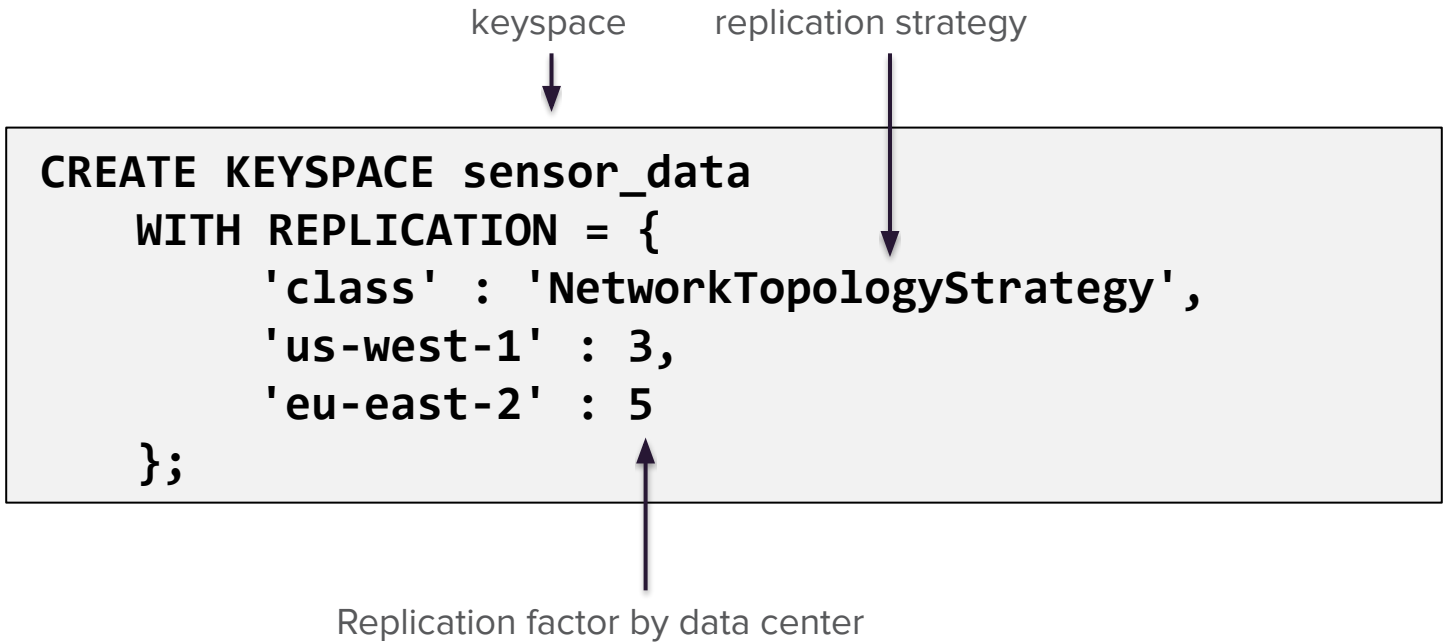
❯ Replication Factor

RF = ?

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



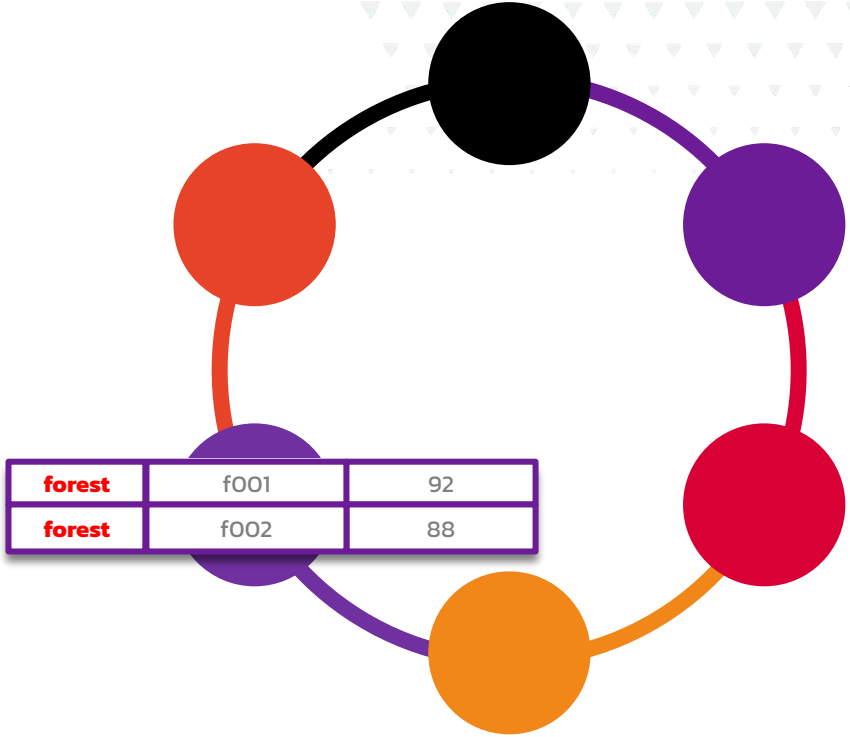
Replication is defined per keyspace



RF = 1

RF = 1

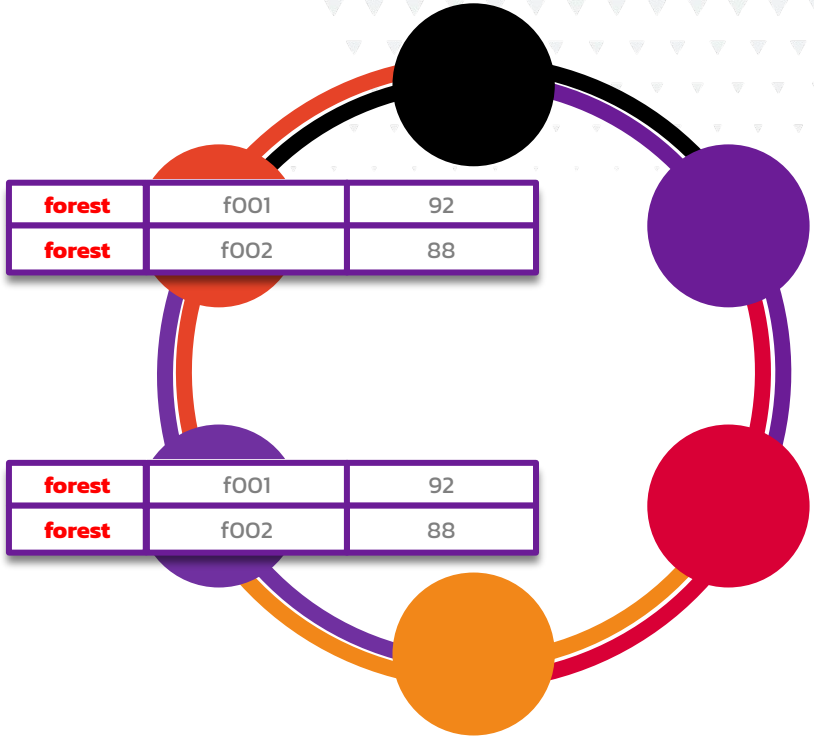
Replication Factor 1 means that every partition is stored on 1 node



RF = 2

RF = 2

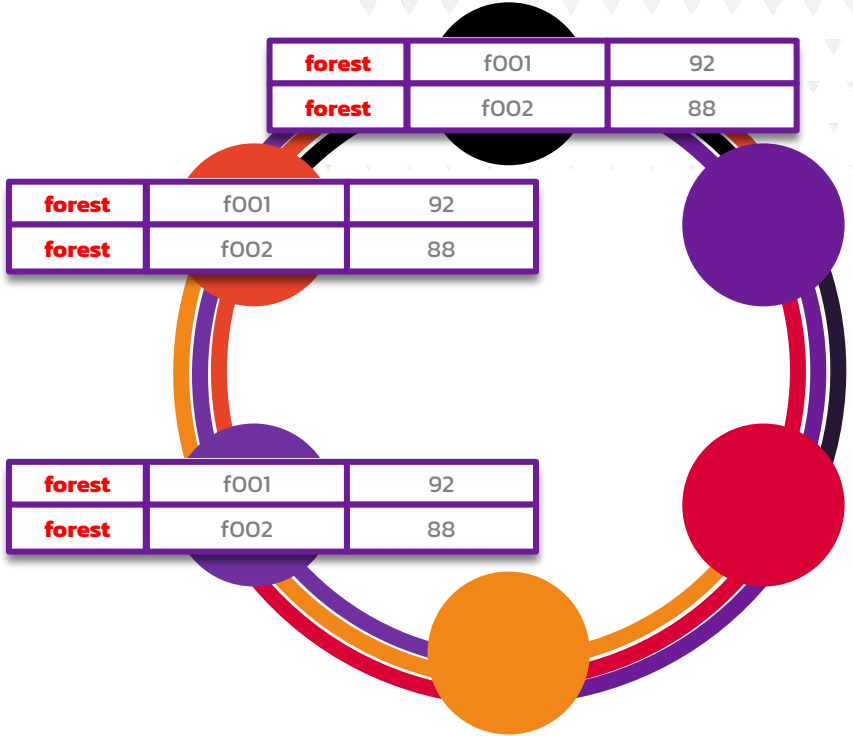
Replication Factor 2 means that every partition is stored on 2 nodes



RF = 3

RF = 3

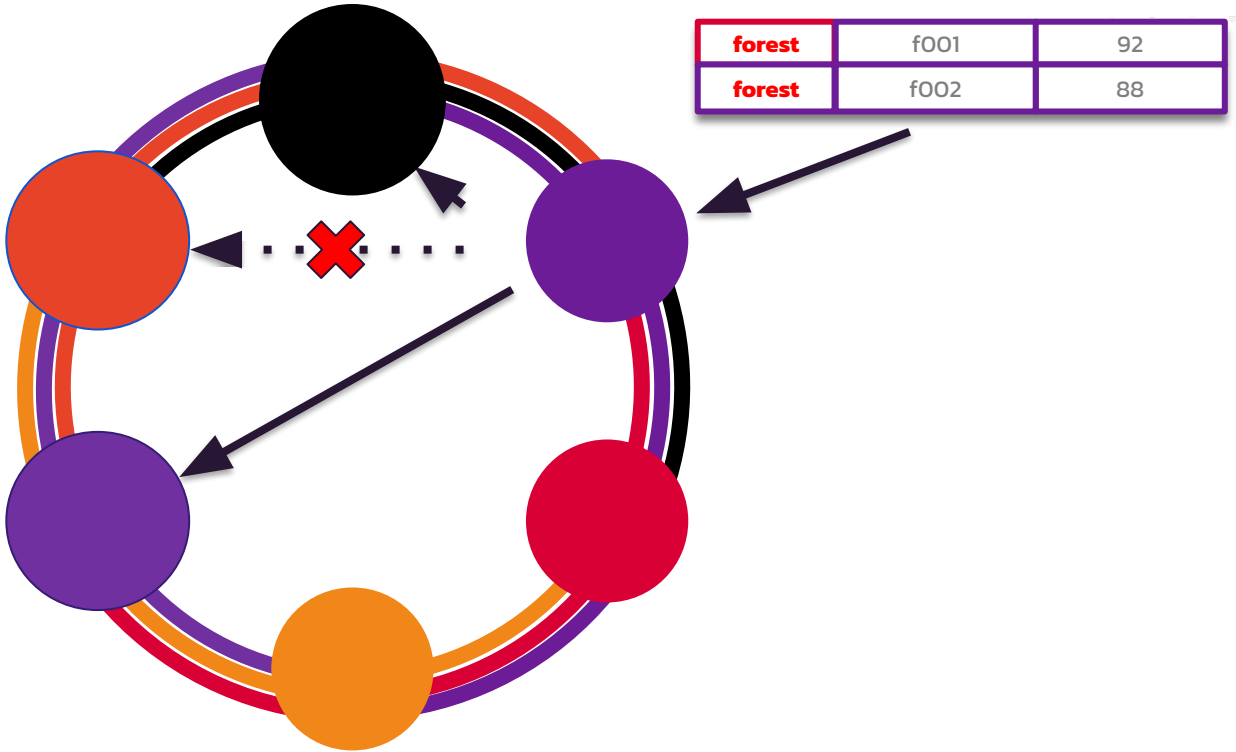
Replication Factor 3 means that every partition is stored on 3 nodes



Replication, consistency and availability

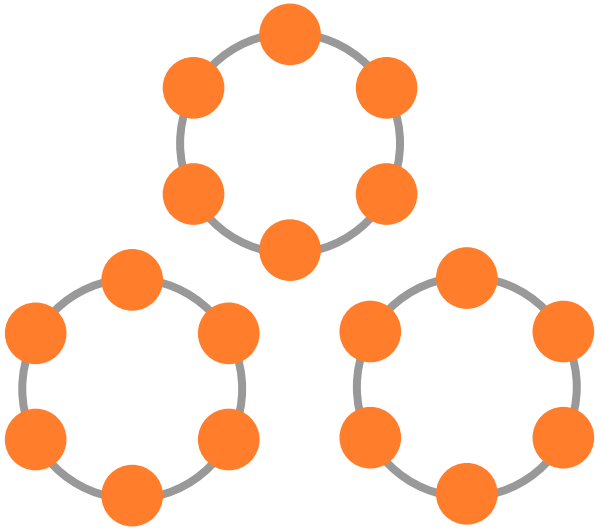
But what if ...

RF = 3

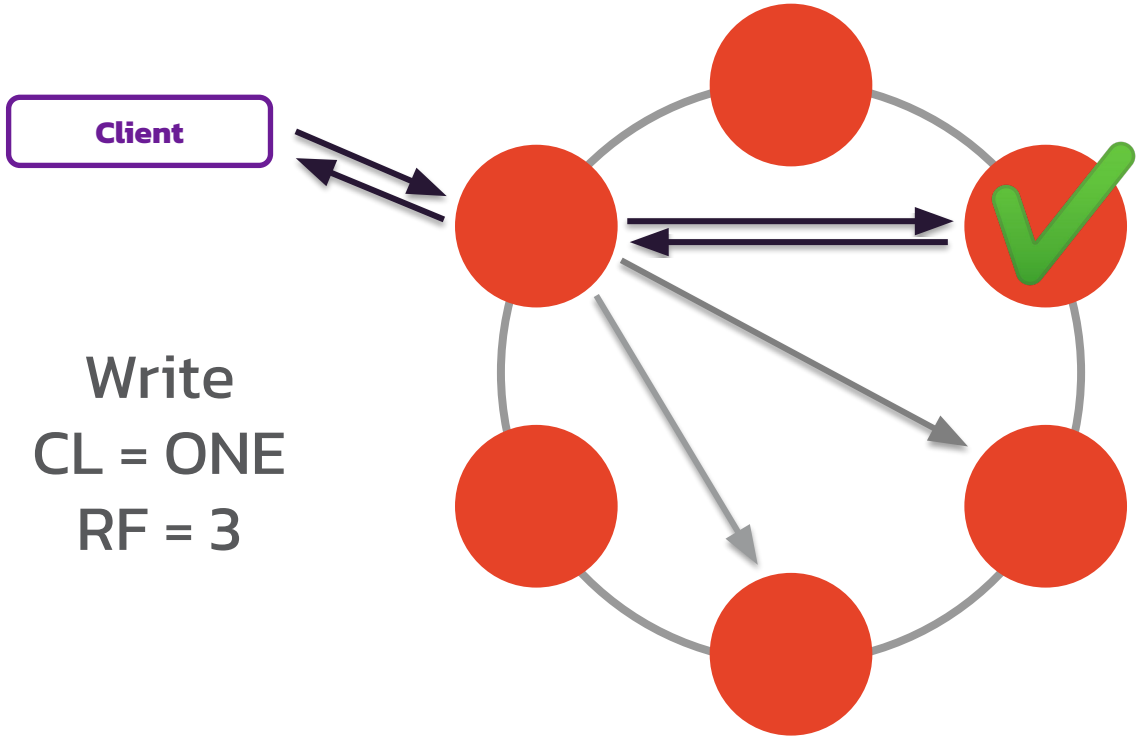


Tunable Consistency and Consistency Levels

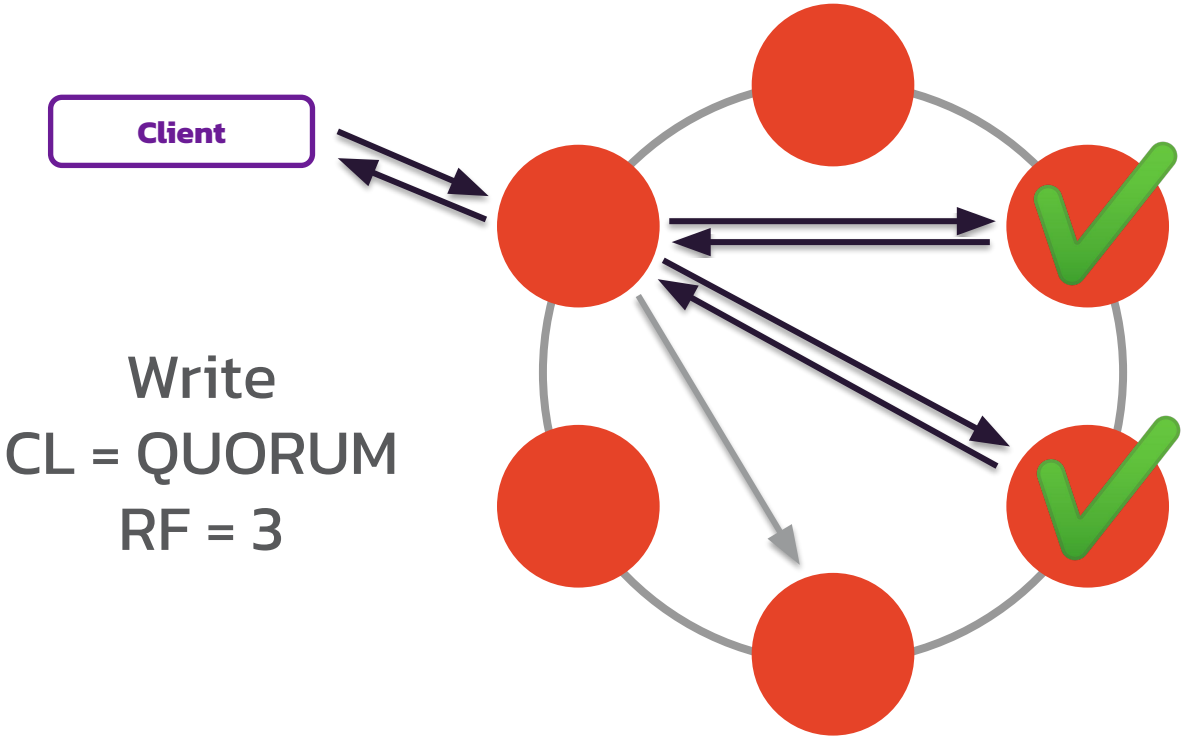
- ANY
- ONE
- TWO
- THREE
- QUORUM
- LOCAL_ONE
- LOCAL_QUORUM
- EACH_QUORUM
- ALL



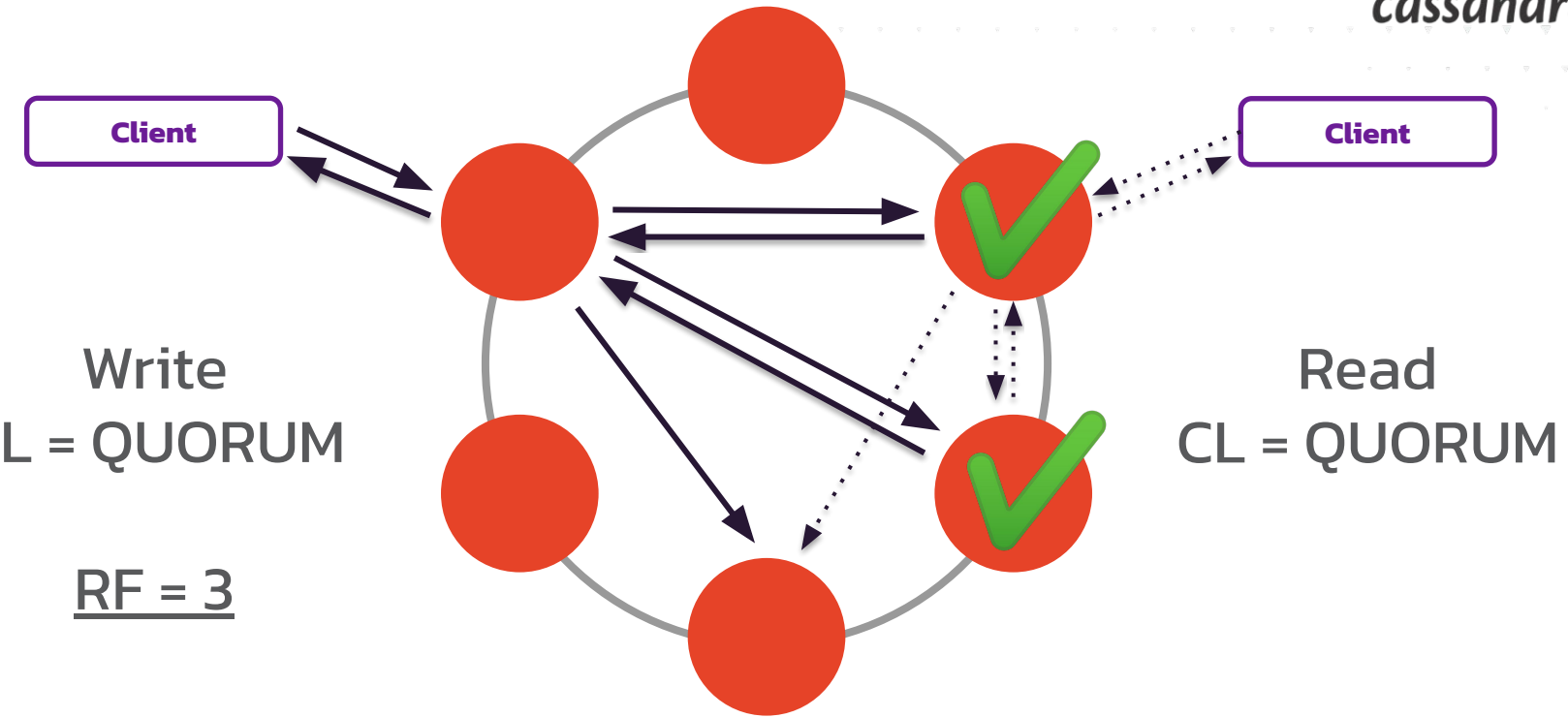
Consistency Level ONE



Consistency Level QUORUM



Immediate Consistency



CL Write + CL Read > RF → Immediate Consistency

Data Structure: a Cell

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

Data Structure: a Row

A single, structured data item in a table.

1	John	Doe	Wizardry

Data Structure: a Partition

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.

ID	First Name	Last Name	Department
1	John	Doe	Wizardry
399	Marisha	Chapez	Wizardry
415	Maximus	Flavius	Wizardry

Data Structure: a Table

A group of columns and rows storing partitions.

ID	First Name	Last Name	Department
1	John	Doe	Wizardry
2	Mary	Smith	Dark Magic
3	Patrick	McFadin	DevRel

Lab 2

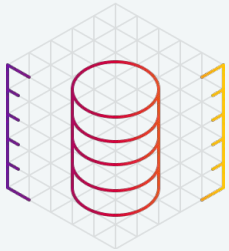
Discovering CQL

2.1 Create SCHEMA

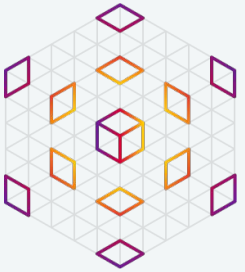
2.2 Insert Data

2.3 Data Exploration

github.com/datastaxdevs/workshop-swinburne



Agenda



01

Introduction to NoSQL
Why, what, how

02

Getting Started with
Apache Cassandra

03

Data Modeling
Application design

04

App Development
Python and Java

05

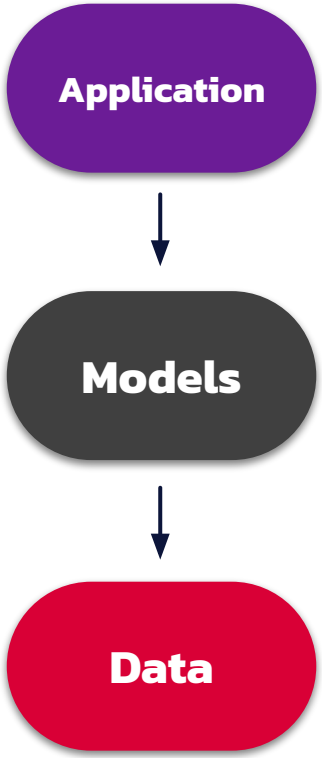
Stargate Apis
Cloud Native

06

What's next?
Homework, next sessions

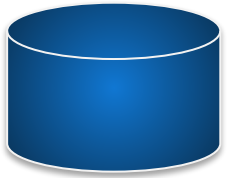
NoSQL Data Modelling

1. Analyze user behaviour (customer first!)
2. Identify workflows, their dependencies and needs
3. Define Queries to fulfill these workflows
4. Knowing the queries, design tables, using **denormalization**.
5. Use BATCH when inserting or updating denormalized data of multiple tables

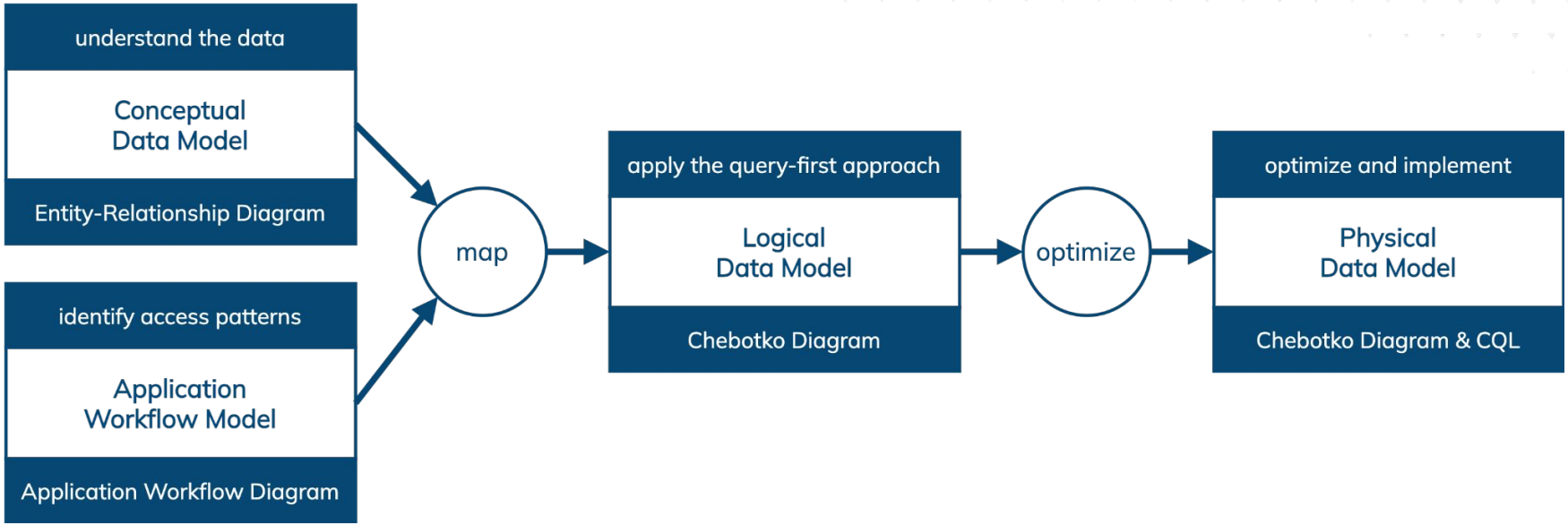


Employees

userId	firstName	lastName	department
1	Edgar	Codd	Engineering
2	Raymond	Boyce	Math
3	Sage	Lahja	Math
4	Juniper	Jones	Botany

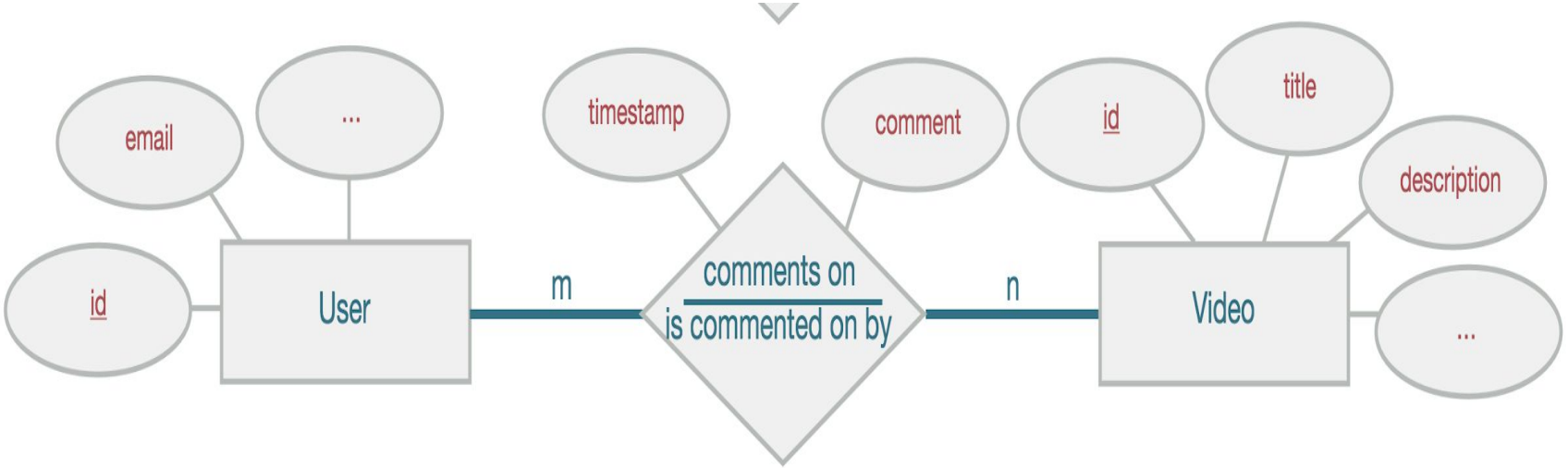


Data Modelling Methodology



very often, 1 query = 1 table

Conceptual Data Model



Application Workflow

Use-Case I:

- A User opens a Profile

WF2: Find comments related to target user using its identifier, get most recent first

Use-Case II:

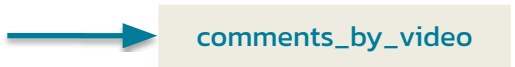
- A User opens a Video Page

WF1: Find comments related to target video using its identifier, most recent first

Mapping (1 / 2)

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

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



Mapping (2 / 2)

comments_by_user	
userid	K
creationdate	C ↓
commentid	C ↑
videoid	
comment	

comments_by_video	
videoid	K
creationdate	C ↓
commentid	C ↑
userid	
comment	

Optimize

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	

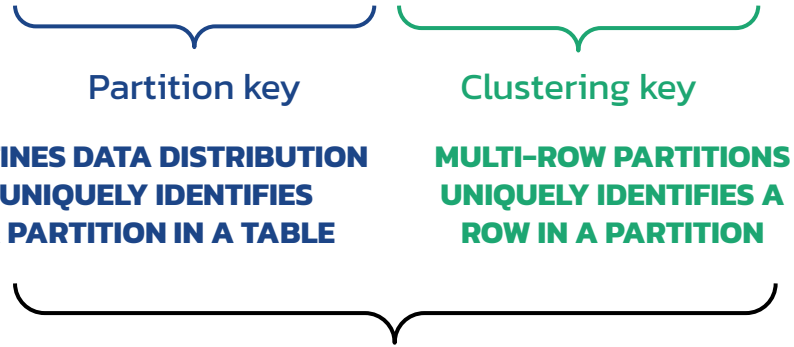
Cassandra Query Language

```
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);
```

Primary Key, Partition Key, Clustering Key

```
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);
```



Structure => Valid Queries

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

```
SELECT * FROM temperatures_by_sensor ...

    WHERE sensor = ?;

    WHERE sensor > ?;

    WHERE sensor = ? AND date > ?;

    WHERE sensor = ? AND date = ?;

    WHERE sensor = ? AND date = ? AND timestamp > ?;
```

Good partition rules

Q: Show temperature evolution over time for sensor X On Nov 10, 2022

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

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

- **Store together what you retrieve together**
- Avoid big partitions
- Avoid hot partitions

Good partition rules

BUCKETING

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

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

- Up to 2 billion cells per partition
- Up to ~100k values in a partition
- Up to ~100MB in a Partition

- Store together what you retrieve together
- **Avoid big partitions**
- Avoid hot partitions

Good partition rules

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

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

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

- Store together what you retrieve together
- Avoid big partitions
- **Avoid hot partitions**

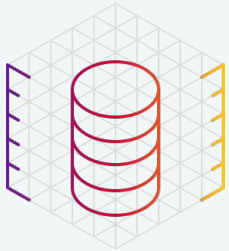
Lab 3

Data Model

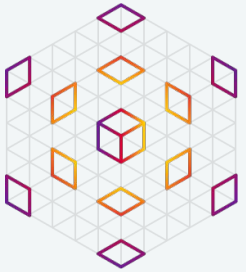
2.1 Create SCHEMA

2.2 Insert Data

github.com/datastaxdevs/workshop-swinburne



Agenda



01

Introduction to NoSQL
Why, what, how

02

Getting Started with
Apache Cassandra

03

Data Modeling
Application design

04

Application Developmen
Python and Java

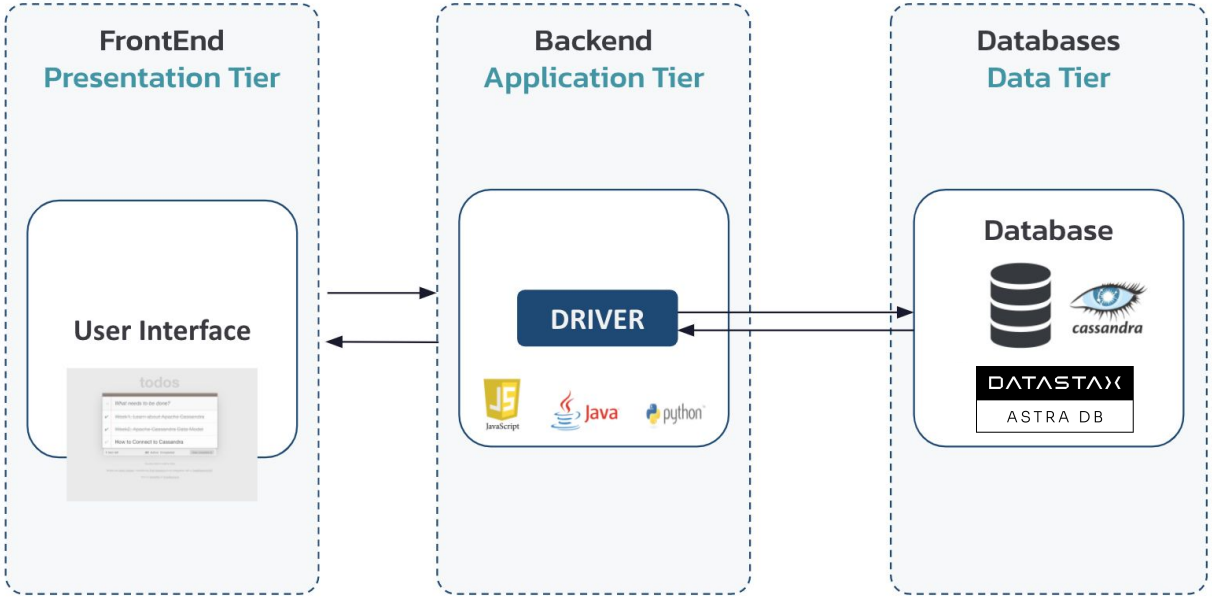
05

Stargate Apis
Cloud Native

06

What's next?
Homework, next sessions

Application Development with Cassandra



Drivers





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

Installing the Drivers

```xml
<dependency>
    <groupId>com.datastax.oss</groupId>
    <artifactId>java-driver-core</artifactId>
    <version>4.13.1</version>
</dependency>
```

```
npm install cassandra-driver
```

4.6.3

```
{
  "dependencies": {
      "cassandra-driver": "^4.6.3"
  }
}
```

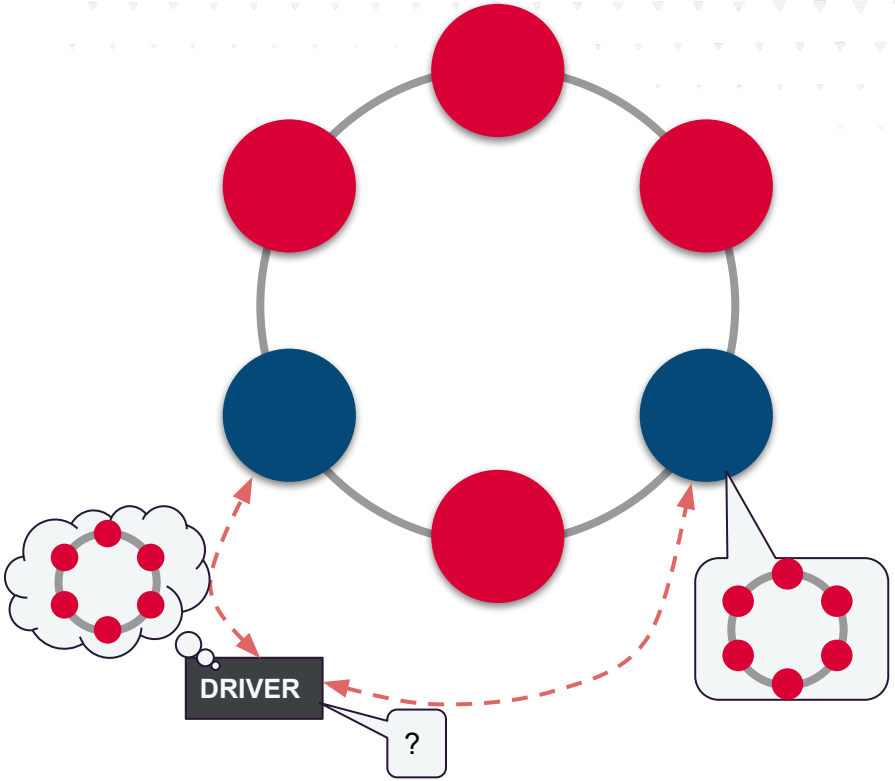
```
pip install cassandra-driver==3.25.0
```

nuget v3.15.0

```
Install-Package CassandraCSharpDriver -Version 3.15.0
```

Contact Points (Cassandra)

- One contact point *would be enough*
  … unless that node is down
- ~3 nodes per DC for resilience
- From there, drivers discover whole cluster
- Local Datacenter



Create Session Client (Cassandra)

```java
CqlSession cqlSession = CqlSession.builder()
 .addContactPoint(new InetSocketAddress("127.0.0.1", 9042))
 .withKeyspace("sensor_data")
 .withLocalDatacenter("dc1")
 .withAuthCredentials("U","P")
 .build();
```


```nodejs
const client = new cassandra.Client({
  contactPoints: ['127.0.0.1'],
  localDataCenter: 'dc1',
  keyspace: 'sensor_data',
  credentials: { username: 'U', password: 'P' }
});
await client.connect();
```


```python
auth_provider = PlainTextAuthProvider(
   username='U', password='P')
cluster = Cluster(['127.0.0.1'],
  auth_provider=auth_provider, protocol_version=5)
session = cluster.connect('sensor_data')
```


```dotnet
Cluster cluster = Cluster.Builder()
.AddContactPoint("127.0.0.1")
.WithCredentials("U", "P")
.Build();
session = cluster.Connect("sensor_data");
```

Create Session Client (Astra)

```java
CqlSession cqlSession = CqlSession.builder()
 .withCloudSecureConnectBundle(Paths.get("secure.zip"))
 .withAuthCredentials("U","P")
 .withKeyspace("sensor_data")
 .build();
```

```nodejs
const client = new cassandra.Client({
  cloud: { secureConnectBundle: 'secure.zip' },
  credentials: { username: 'u', password: 'p' },
  keyspace: 'sensor_data'
});
await client.connect();
```

```python
auth_provider = PlainTextAuthProvider(
   username='U', password='P')
cluster = Cluster(
   cloud ={'secure_connect_bundle': 'secure.zip'},
   auth_provider=auth_provider, protocol_version=4)
session= cluster.connect('sensor_data')
```

```dotnet
var cluster = Cluster.Builder()
 .WithCloudSecureConnectionBundle("secure.zip")
 .WithCredentials("u", "p")
 .Build();
var session = cluster.Connect("sensor_data");
```

There should onlybe one session

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

```
Java:       cqlSession.close();
Python:     session.shutdown();
Node:       client.shutdown();
CSharp:     IDisposable
```

Executing Cql Queries

```
session.execute(
    "SELECT * FROM sensors_by_network WHERE network = %s;",
    (network,)
)
```

```
cqlSession.execute(
    "SELECT * FROM sensors_by_network WHERE network = '" + network + "'"
);
```

Prepare Cql Statements

```
q3_statement = session.prepare(
    "SELECT * FROM sensors_by_network WHERE network = ?;"
)
rows = session.execute(q3_statement, (network,) )
```

```
PreparedStatement q3Prepared = session.prepare(
    "SELECT * FROM sensors_by_network WHERE network = ?");
BoundStatement q3Bound = q3Prepared.bind(network);
ResultSet rs = session.execute(q3Bound);
```

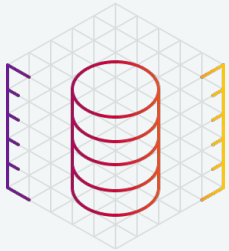
Lab 4

Coding Java and Python

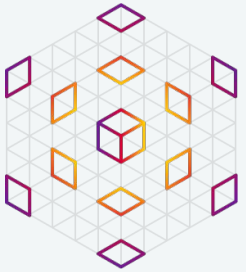
4.1 Sensor Application

4.2 Sensor APi

github.com/datastaxdevs/workshop-swinburne



Agenda



01

Introduction to NoSQL
Why, what, how

02

Getting Started with
Apache Cassandra

03

Data Modeling
Application design

04

Application Developmen
Python and Java

05

Stargate Apis
Cloud Native

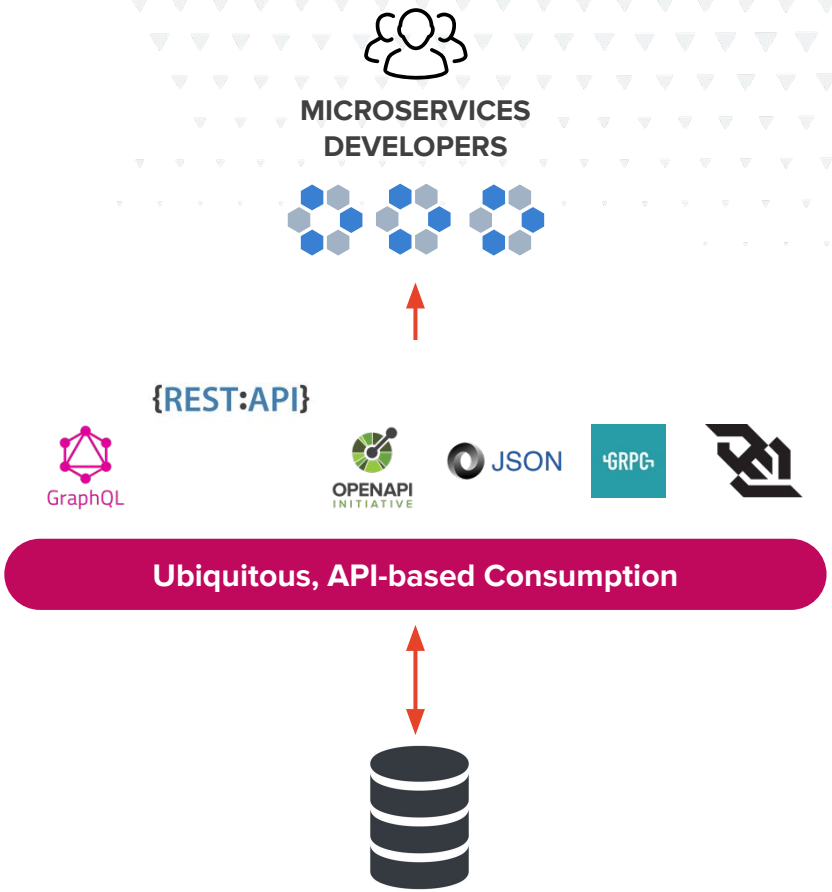
06

What's next?
Homework, next sessions

Data Gateway

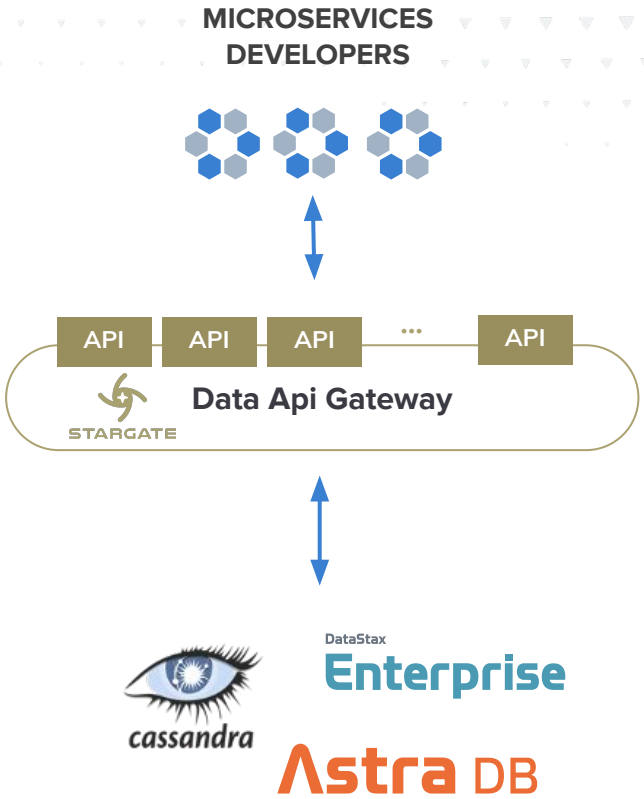
> Developers want the option to use modern APIs and development gateways.

Apache Cassandra is a distributed database designed for the new standard and the associated APIs that facilitate the first choice of developers.

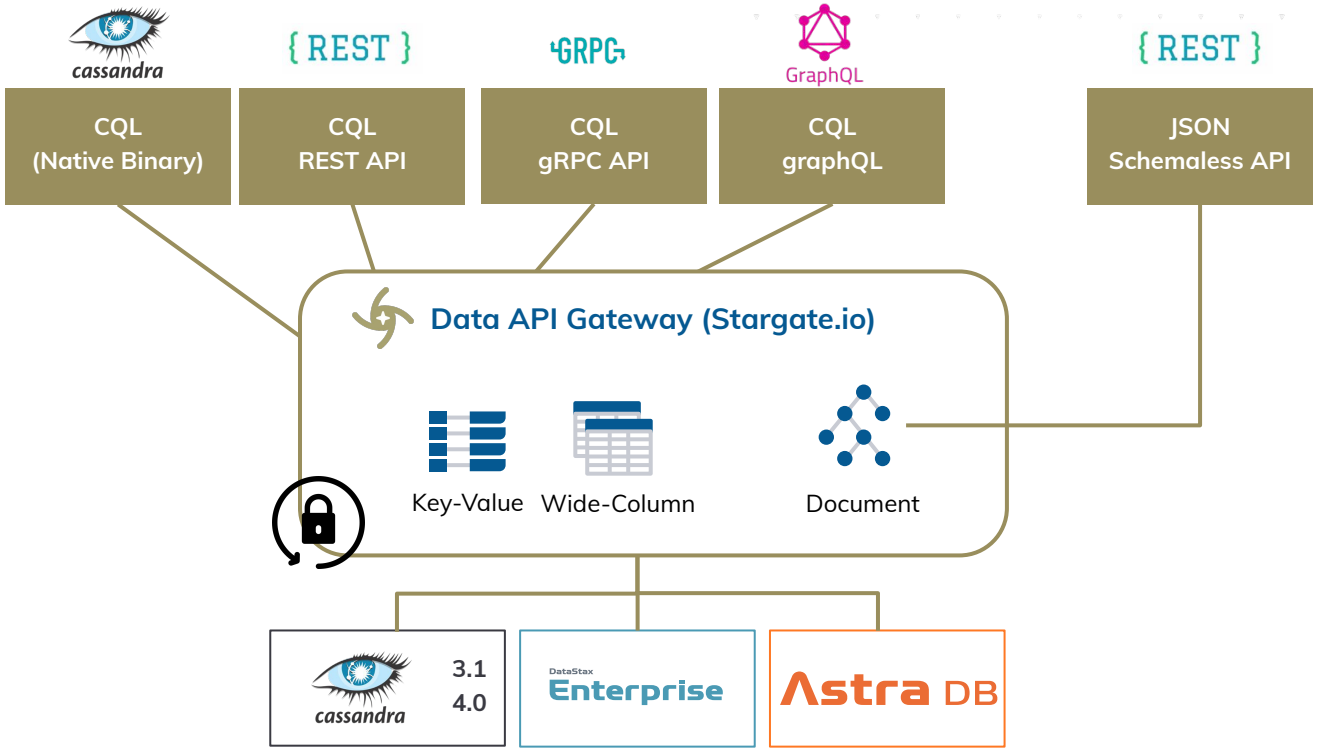


What is Stargate ?

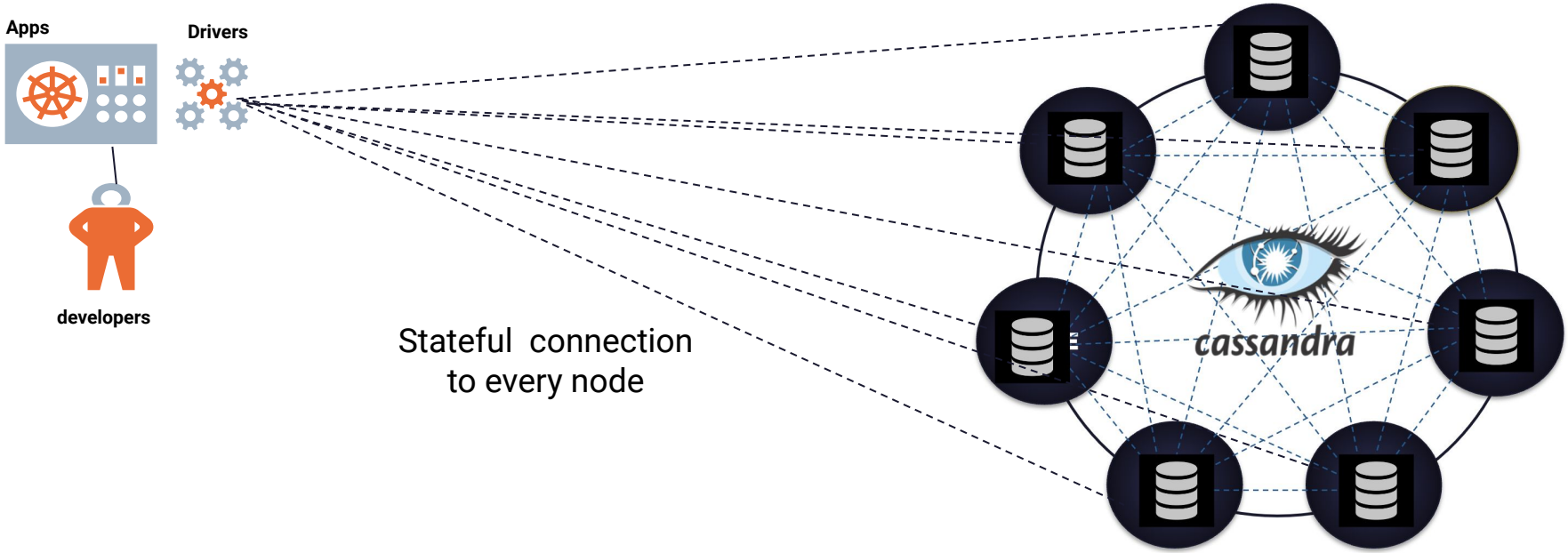
- Data API gateway
- Sits between applications and Cassandra
- Translates API calls into CQL
- Extensible, open source project on GitHub
- Apps/microservices can use more than one API



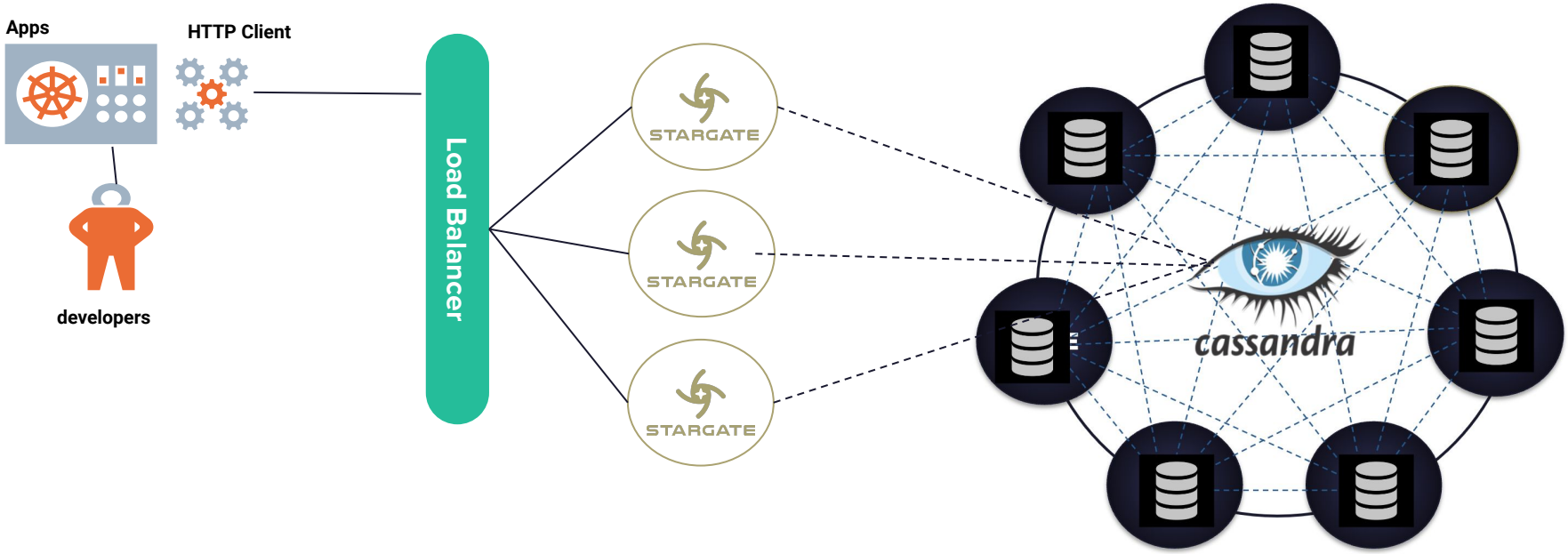
Stargate Data Gateway



Architecture



Architecture with Stargate



Stargate Api Overview

Drivers | **Open API**

Cassandra Query Language

SQL like Table Model

Structured Data

Key-Value Data

Strong Types

Minimal query overhead

gRPC

gRPC

Structured Data (CQL)

Lighter weight

Native driver alternative

Low query overhead

GraphQL

Hierarchy of types and fields

Structured Data

Key-Value Data

Low query overhead

{REST}

REST

Row based

Structured Data

Key-Value Data

Weaker Types

High query overhead

Document

JSON Documents

Semi-Structured Data

Weaker Types

High query overhead

More Performant ⟷ **More Flexible**

Stargate Api Overview

Cassandra Query Language | **Document !**

Cassandra Query Language

SQL like Table Model

Structured Data

Key-Value Data

Strong Types

Minimal query overhead

gRPC

gRPC

Structured Data (CQL)

Lighter weight

Native driver alternative

Low query overhead

GraphQL

GraphQL

Hierarchy of types and fields

Structured Data

Key-Value Data

Low query overhead

{REST}

REST

Row based

Structured Data

Key-Value Data

Weaker Types

High query overhead

JSON

Document

JSON Documents

Semi-Structured Data

Weaker Types

High query overhead

More Performant ⟷ **More Flexible**

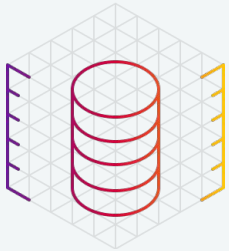
Lab 5

Stargate API

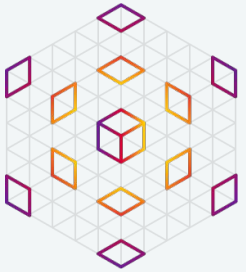
5.1 Rest API

5.2 GraphQL Api

github.com/datastaxdevs/workshop-swinburne



Agenda



01

Introduction to NoSQL
Why, what, how

02

Getting Started with
Apache Cassandra

03

Data Modeling
Application design

04

Application Developmen
Python and Java

05

Stargate Apis
Cloud Native

06

What's next?

Stay in touch!

Discord: **dtsx.io/discord**

Academy: **academy.datastax.com**

Workshops: **datastax.com/workshops**

YouTube: **@DataStax Developers**

DATASTAX

Thank You