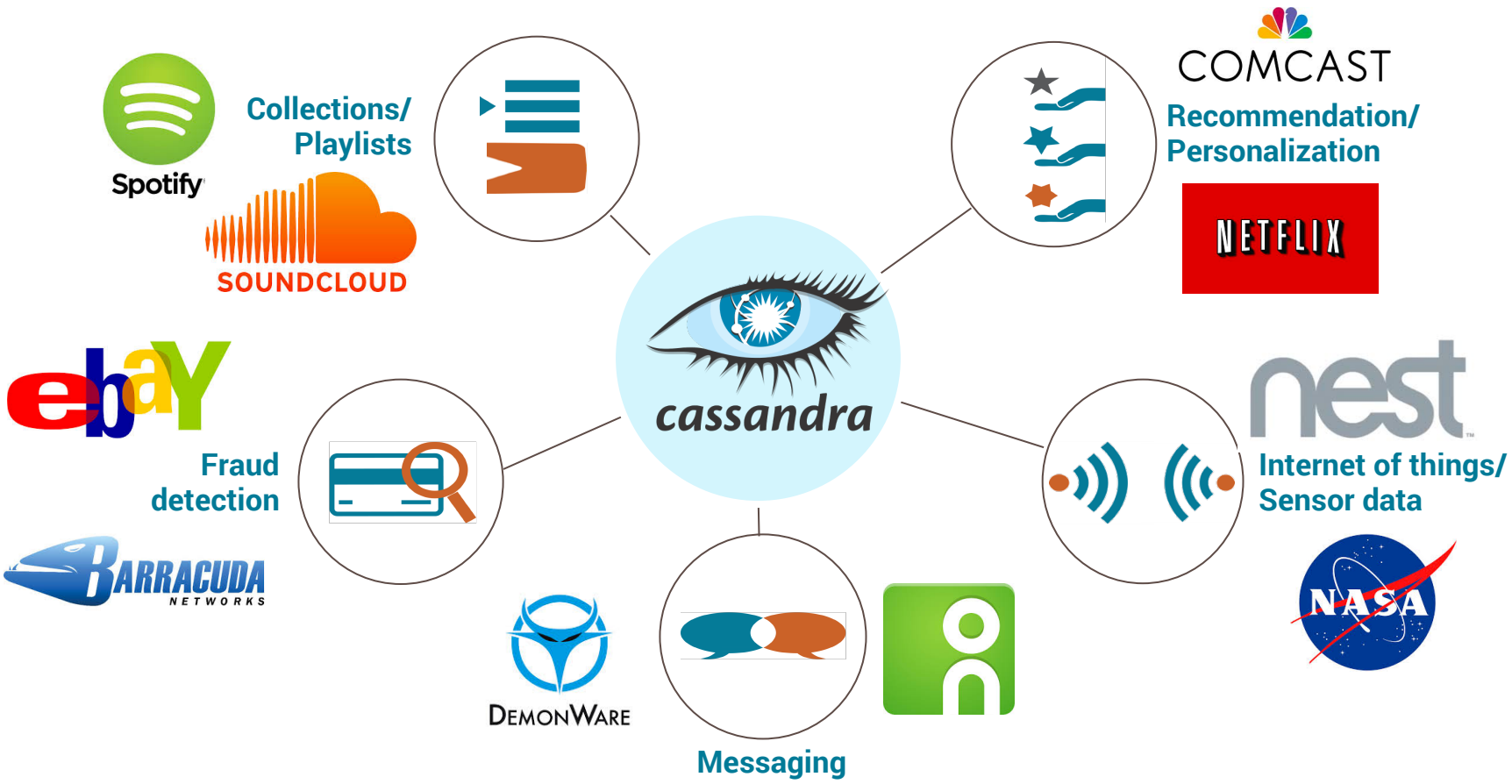
Getting started with Apache Cassandra™

DuyHai DOAN
Datastax Technical Advocate
Apache Zeppelin™ Committer

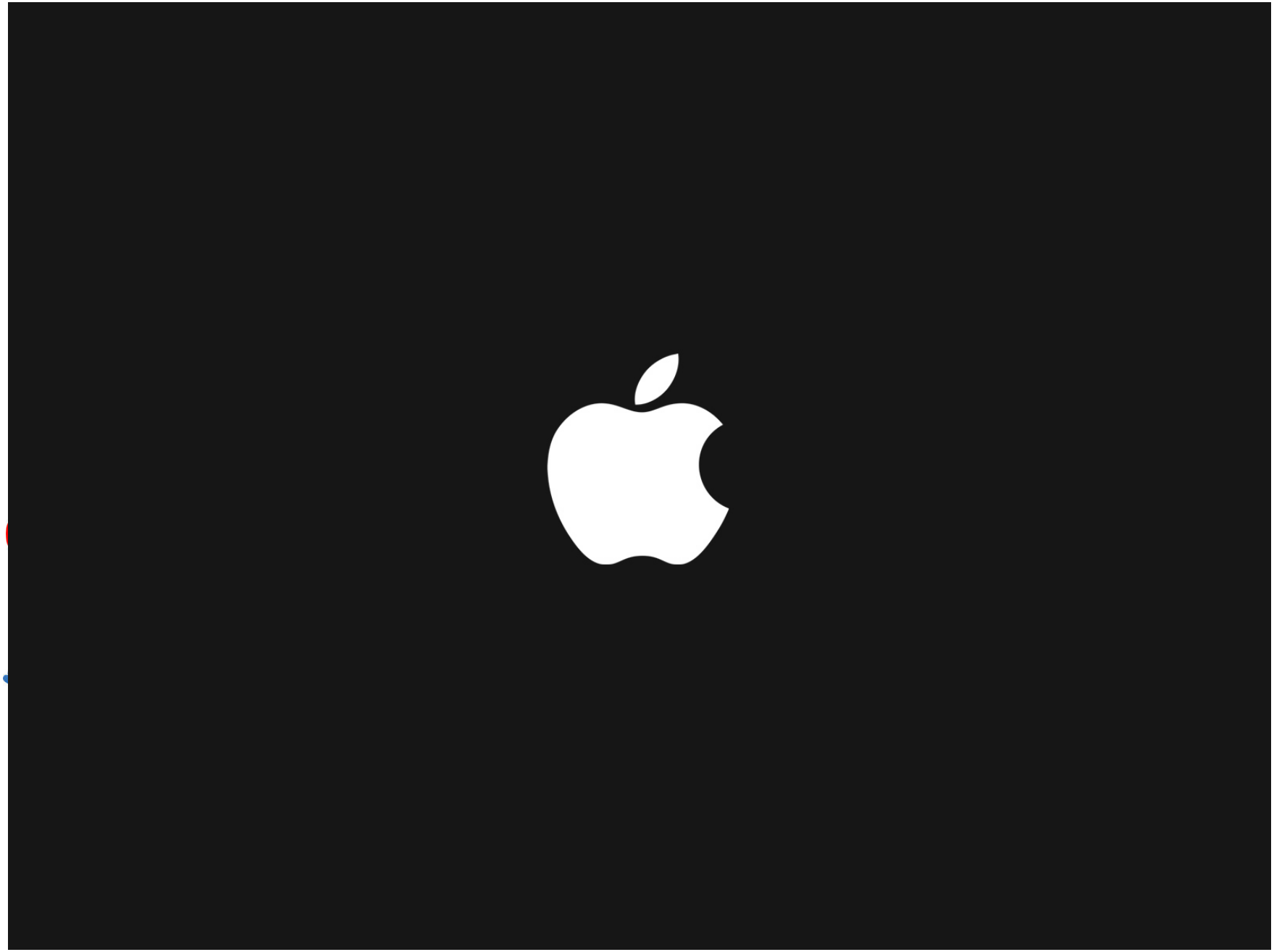
1	Apache Cassandra™ use-cases
2	Why do I need Apache Cassandra™ ?
3	Distribution, replication & consistency model
4	Features Summary

Apache Cassandra™ use-cases

Before (< 2016)



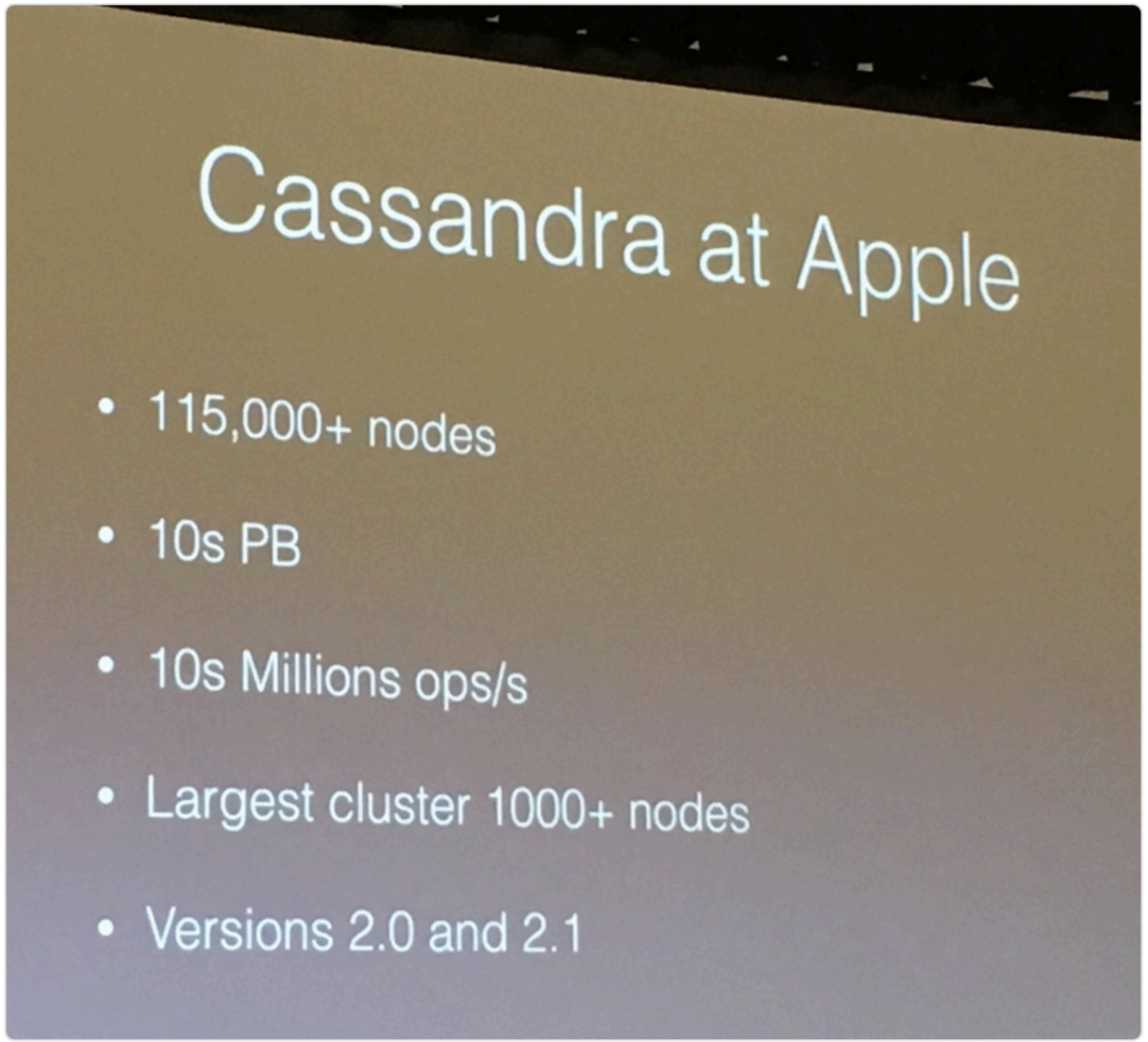
DATASTAX



Michaël Figuière
@mfiguiere

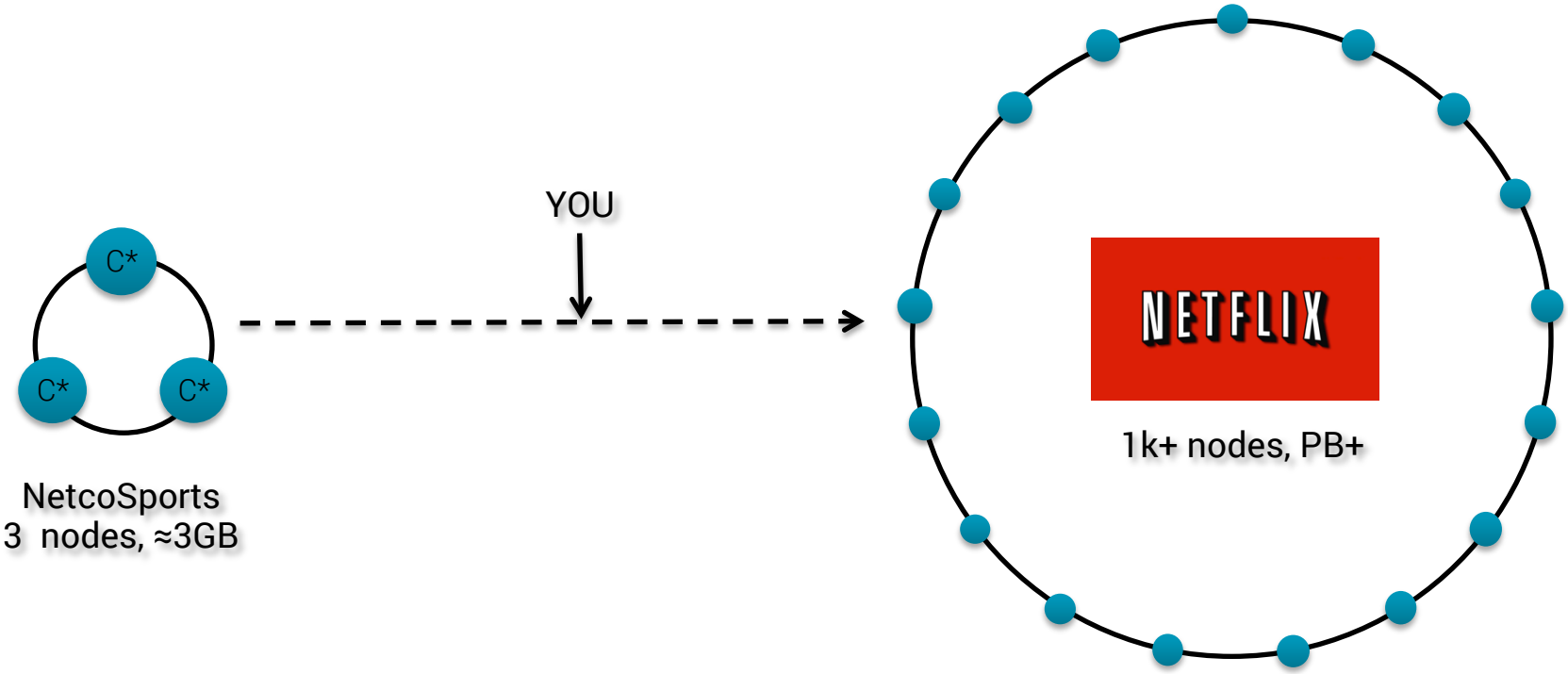
Following

Cassandra keeps growing at Apple, now 115,000+ nodes in production.



Why do I need Apache Cassandra™ ?

Linear Scalability



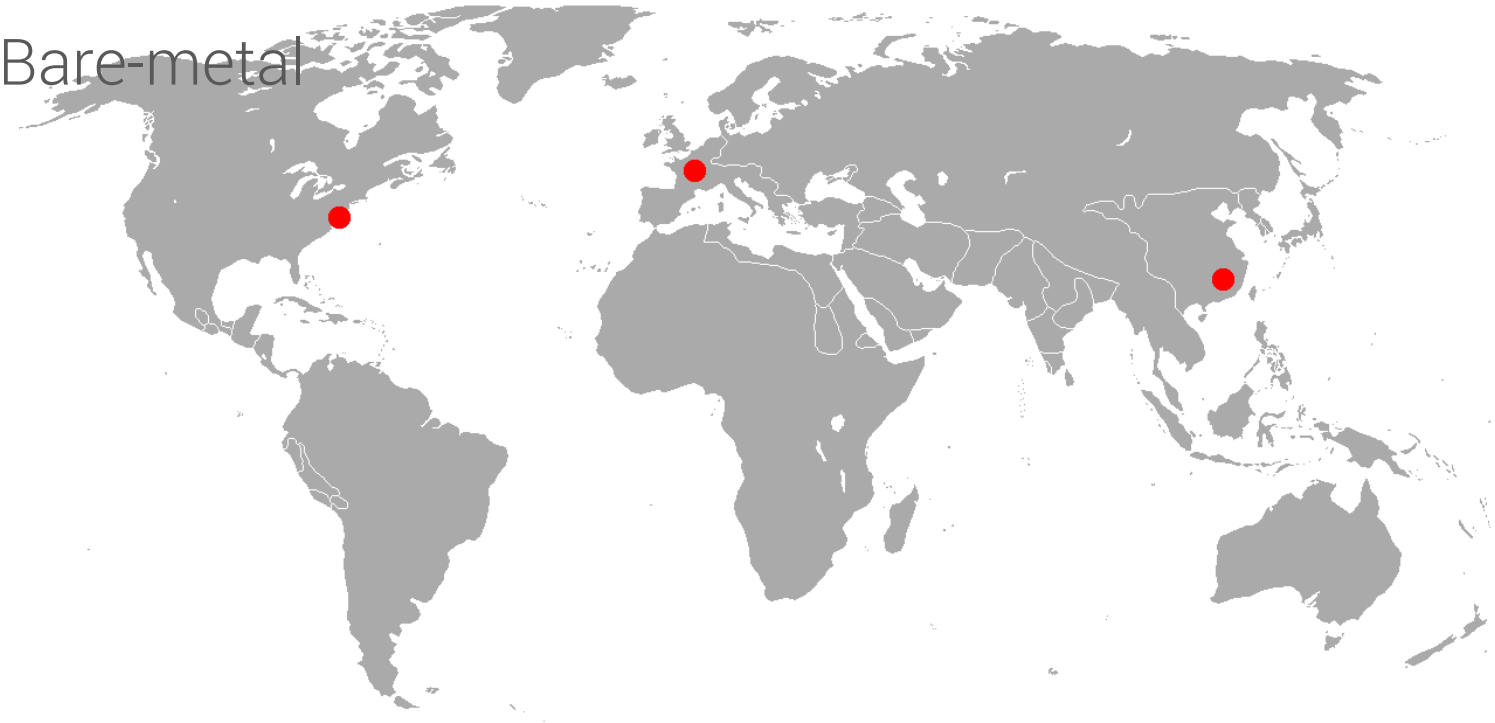
Continuous availability

- thanks to the **Dynamo** architecture

DATASTAX

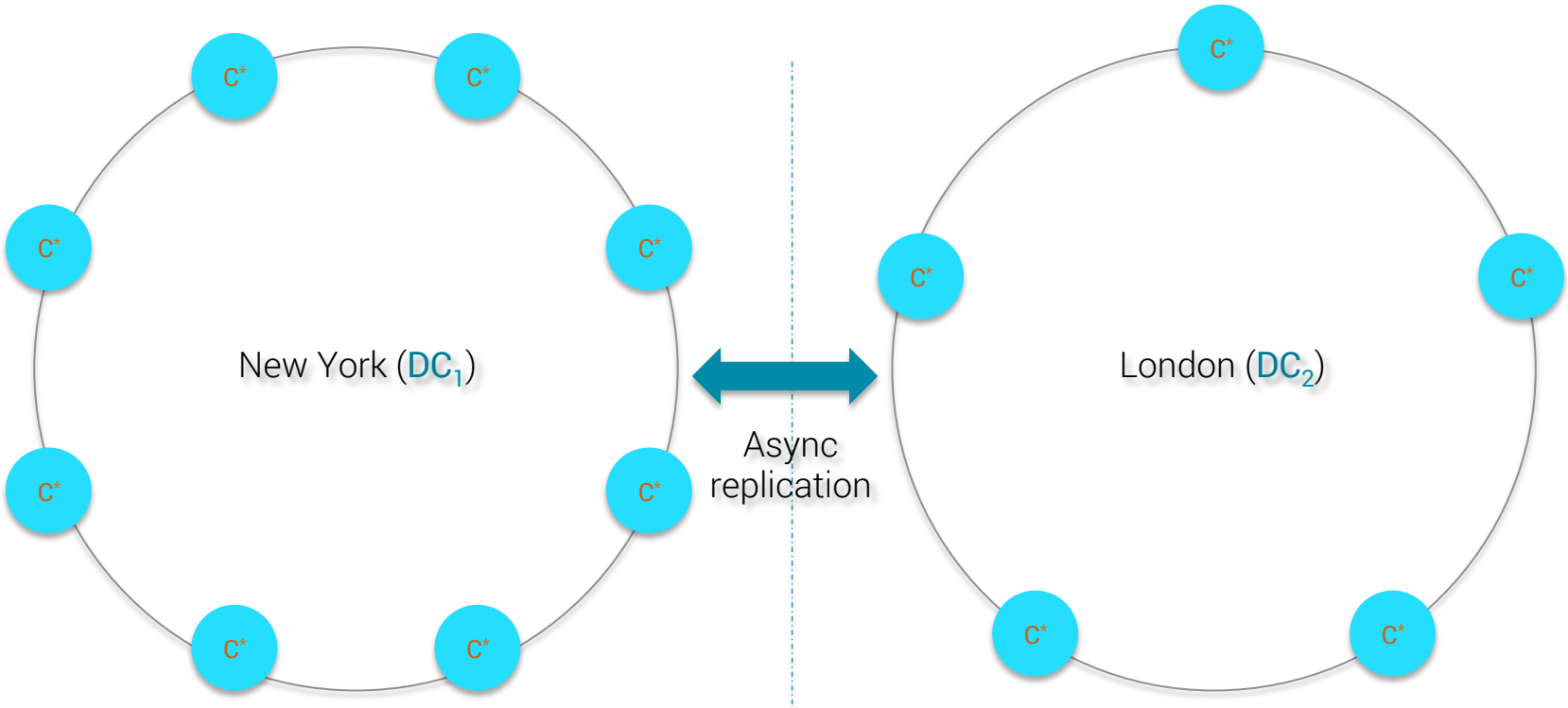
Multi data-centers/cloud native

- out-of-the-box (config only)
- AWS/GCE/Azure/CloudStack support
- Cloud/Bare-metal



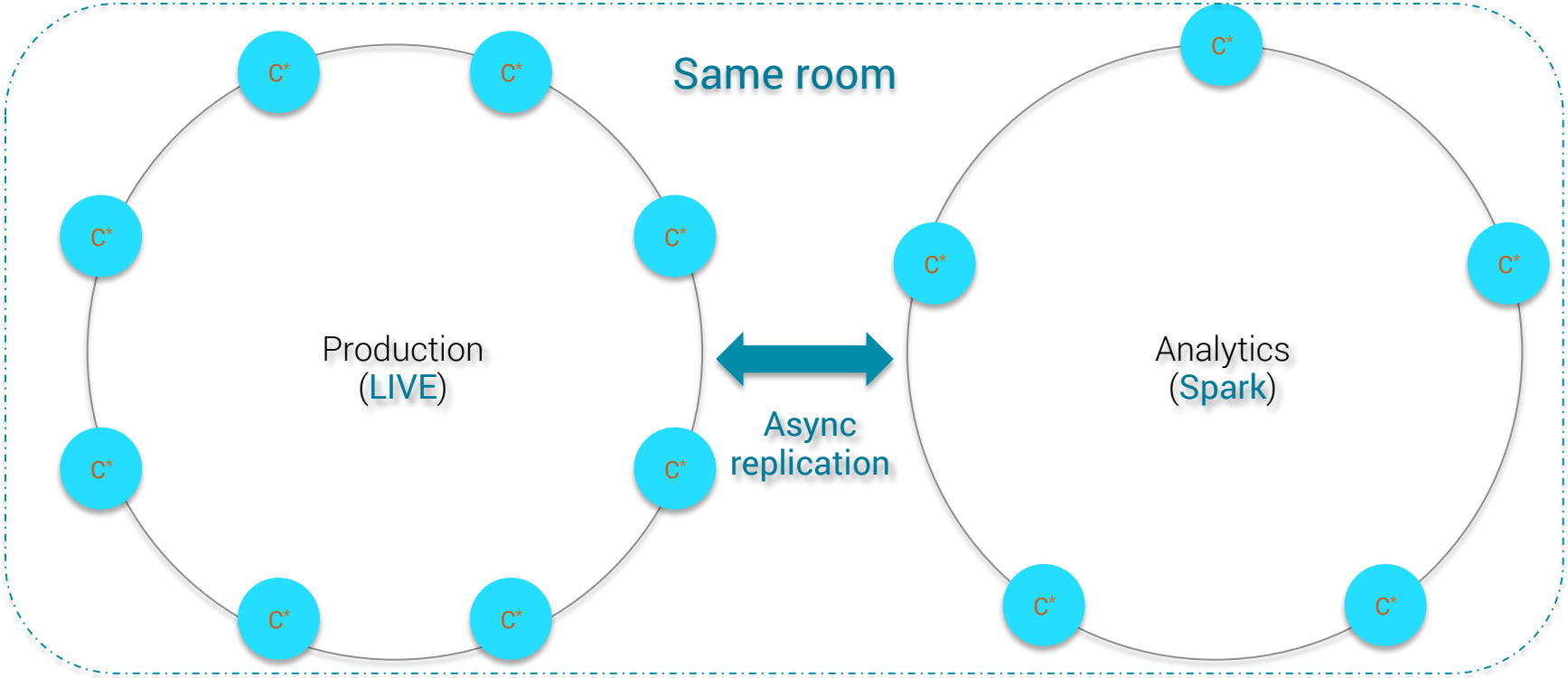
Multi-DC usages

Data locality, disaster recovery



Multi-DC usages

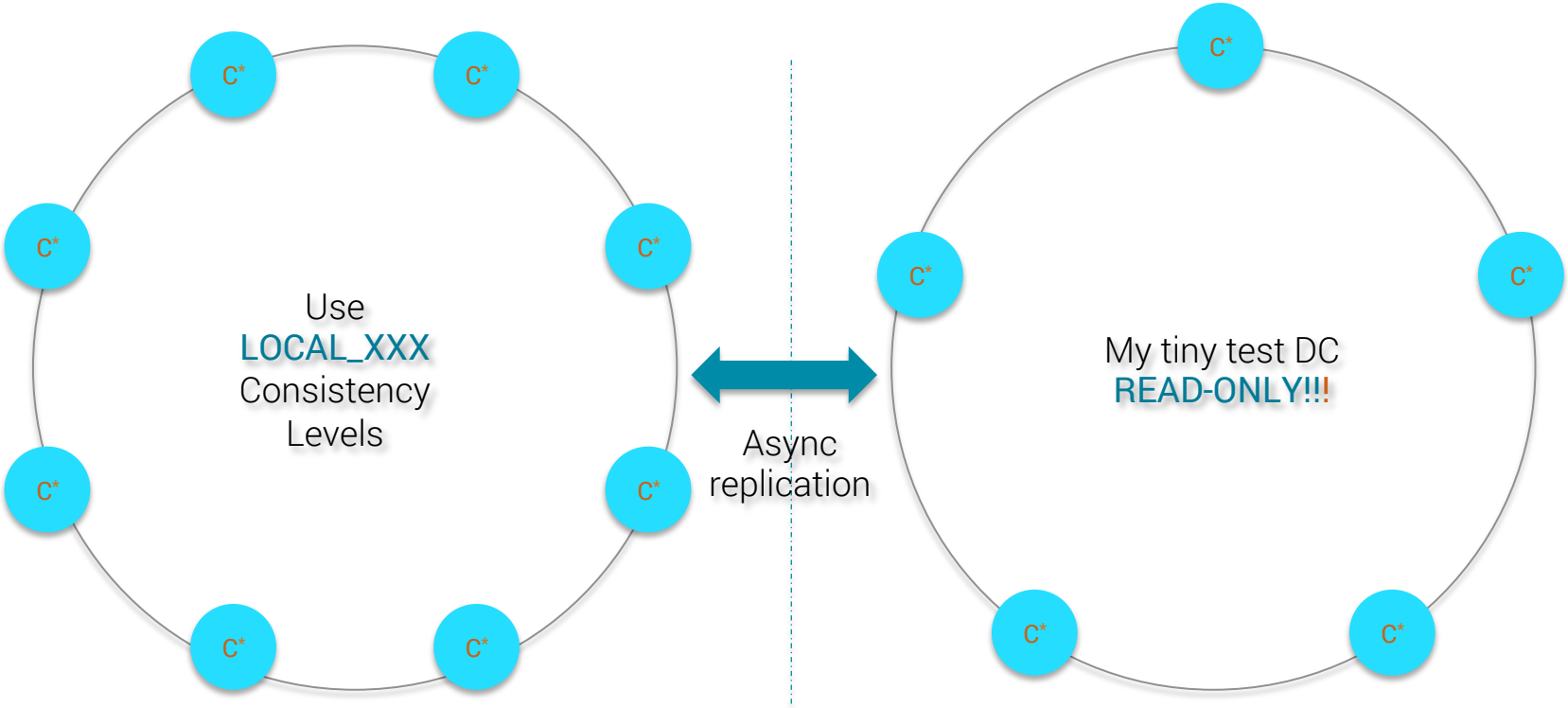
Virtual DC for workload segregation



DATASTAX

Multi-DC usages

Prod data copy for back-up/benchmark



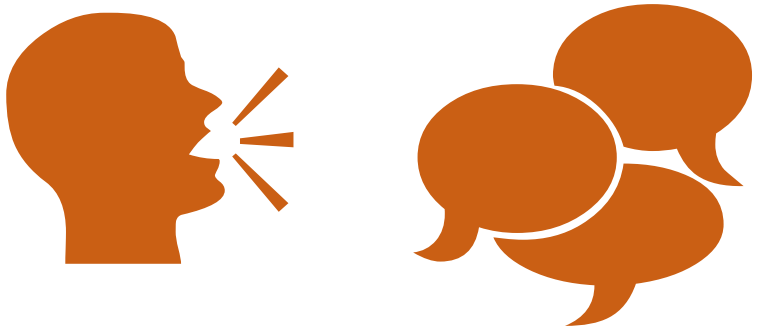
DATASTAX

Operational simplicity

- 1 node = 1 process + 2 config files (*cassandra.yaml* + *cassandra-rackdc.properties*)
- deployment automation (Ansible …)
- No role between nodes, perfect symmetry

Eco System

- Apache Spark – Apache Cassandra integration
  - analytics
  - joins, aggregation
  - SparkSQL/Dataframe integration with CQL (predicates push down)
- Apache Zeppelin – Apache Cassandra integration
  - web-based notebook
  - tabular/graph display



Q & A

Apache Cassandra™ Architecture

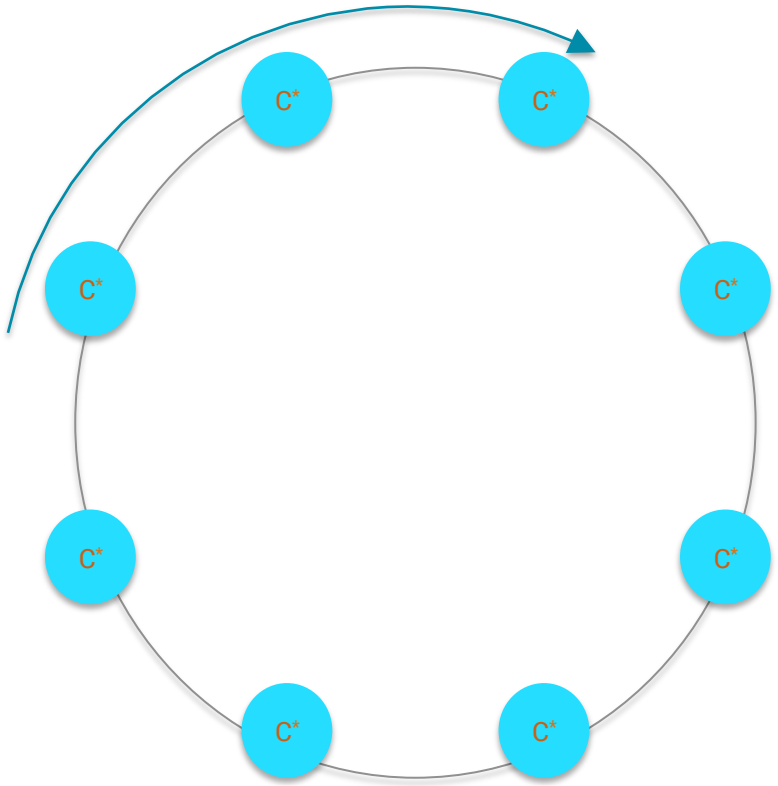
The Tokens

Random hash of **#partition** → **token** = hash(**#p**)

Hash:] –x, x]

hash range: 2^{64} values

x = $2^{64}/2$



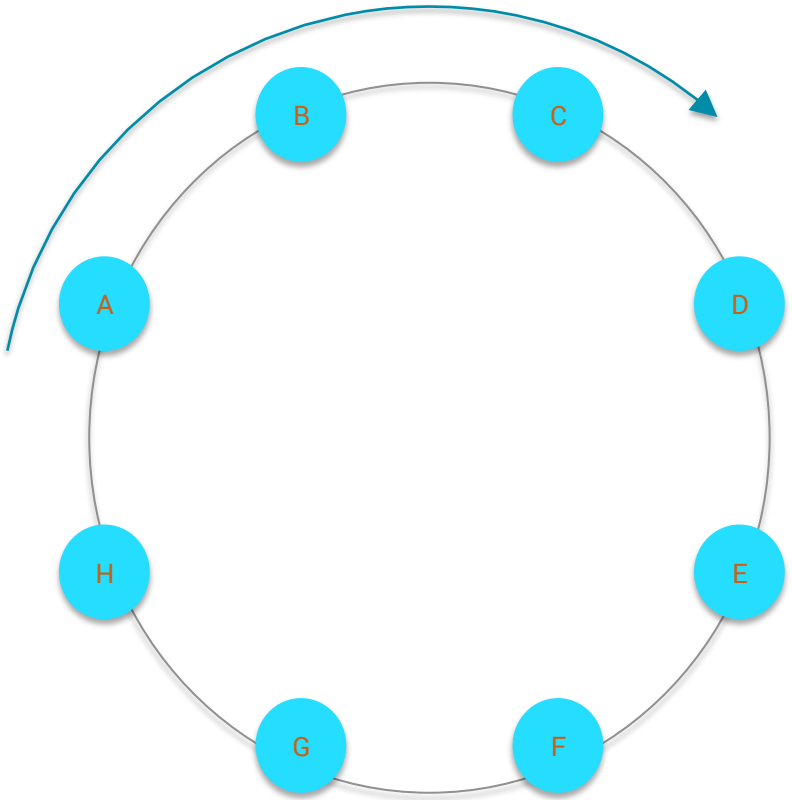
Token Ranges

$$A:\left]-x,-\frac{3x}{4}\right] \qquad E:\left]0,\frac{x}{4}\right]$$

$$B:\left]-\frac{3x}{4},-\frac{2x}{4}\right] \qquad F:\left]\frac{x}{4},\frac{2x}{4}\right]$$

$$C:\left]-\frac{2x}{4},-\frac{x}{4}\right] \qquad G:\left]\frac{2x}{4},\frac{3x}{4}\right]$$

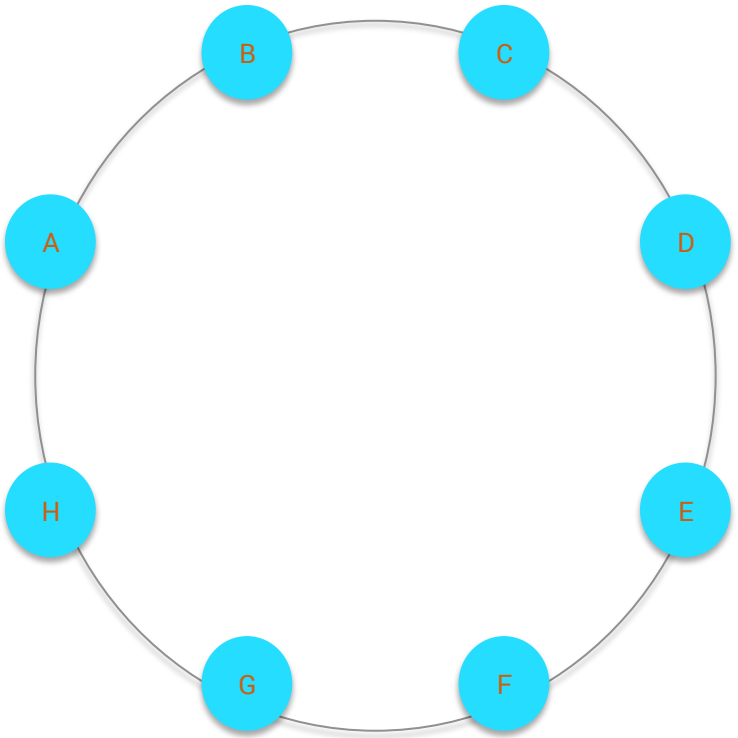
$$D:\left]-\frac{x}{4},0\right] \qquad H:\left]\frac{3x}{4},x\right]$$



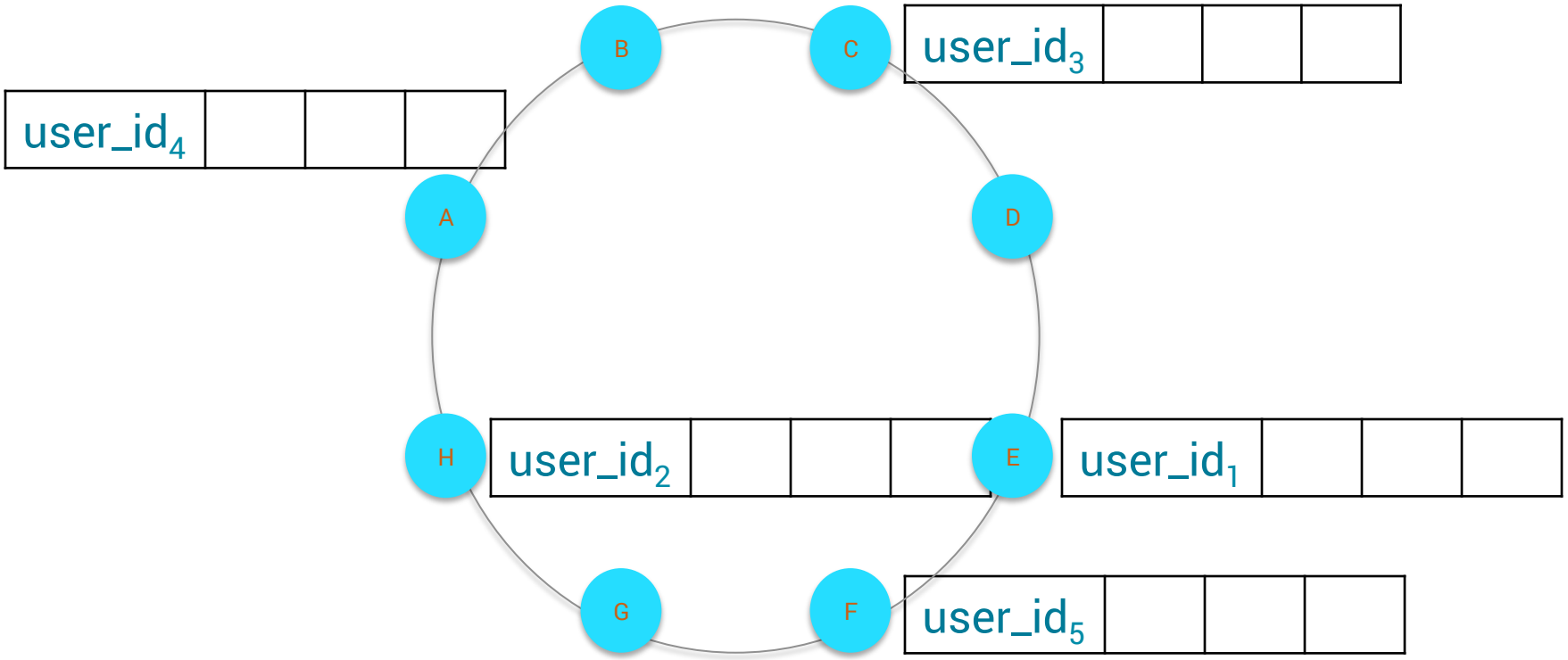
Distributed Tables

```
CREATE TABLE users(
   user_id int,
   ...,
   PRIMARY KEY(user_id)
);
```

$user\_id_1$			
$user\_id_2$			
$user\_id_3$			
$user\_id_4$			
$user\_id_5$			



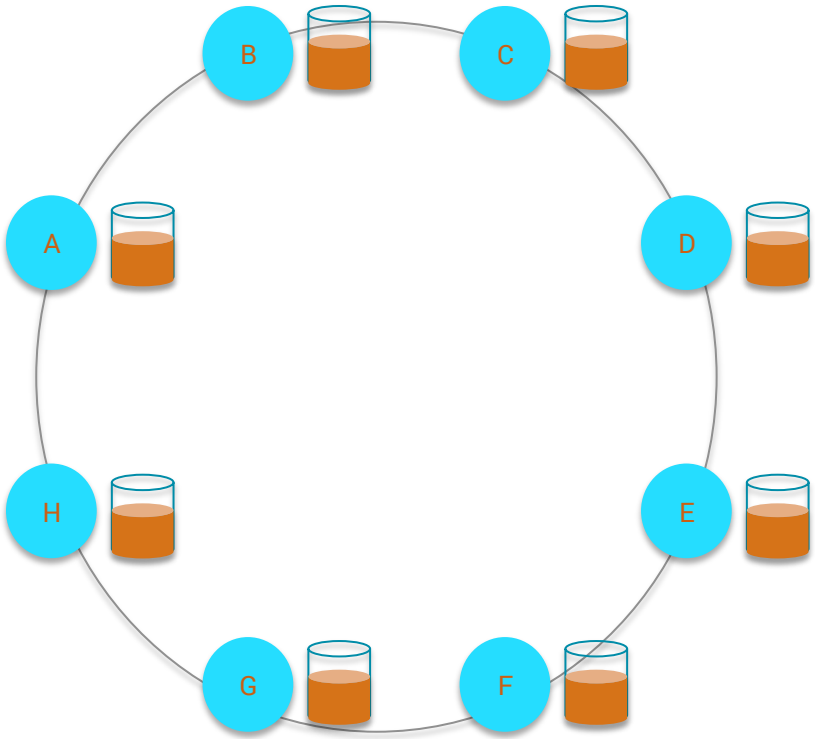
Distributed Tables



Linear Scalability

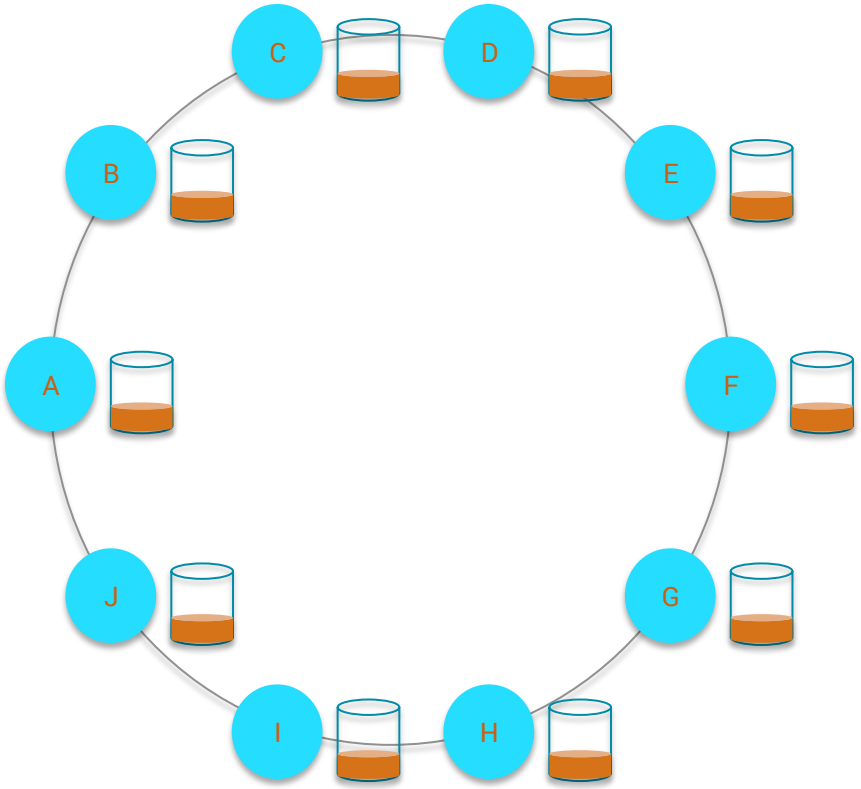
Today = high load, production

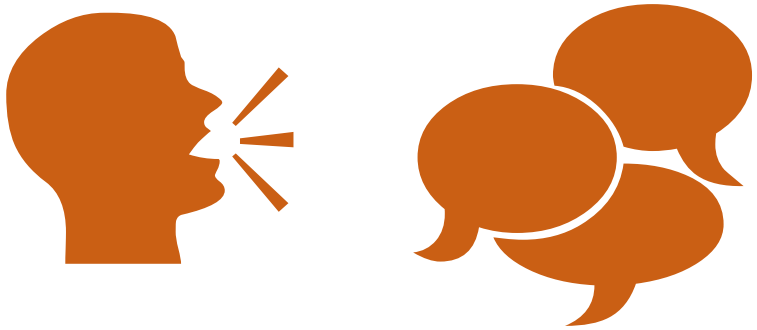
In danger



Scaling Out

+2 nodes to lower the pressure



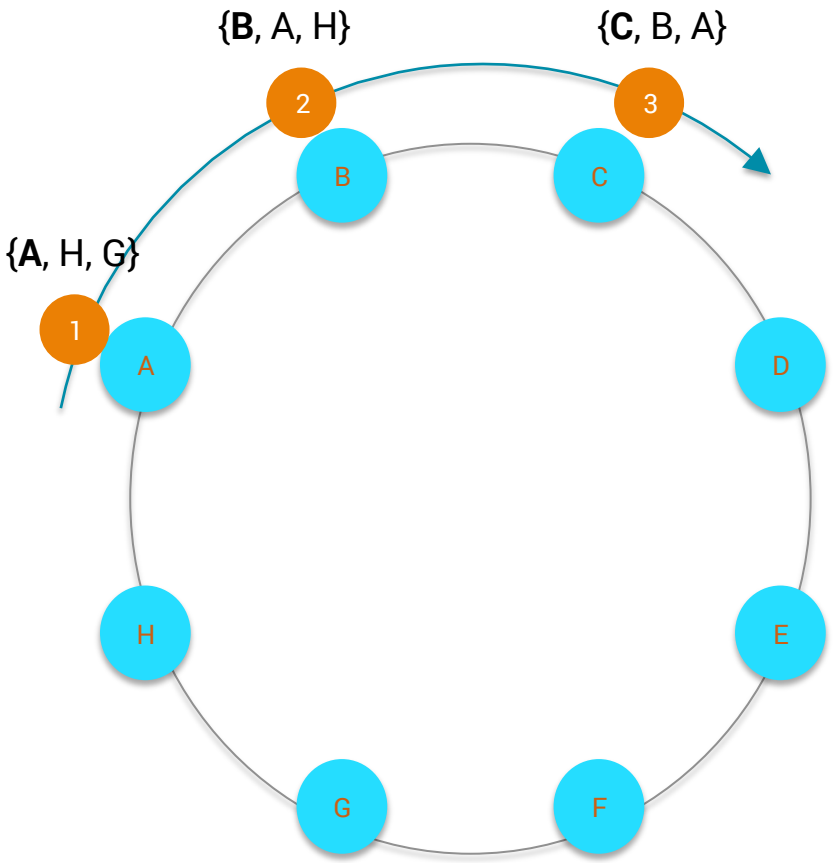


Q & A

Data Replication

Failure Tolerance

Replication factor (RF) = 3



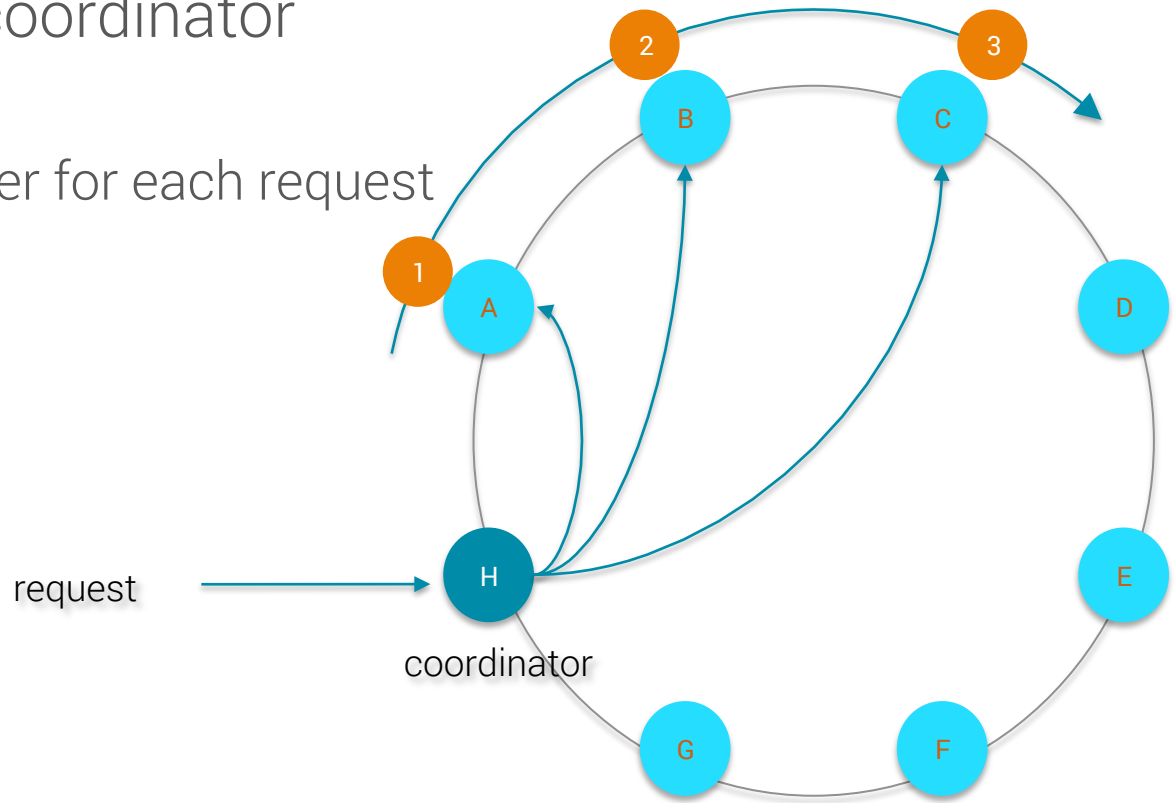
DATASTAX

Coordinator Node

Responsible for handling requests (read/write)

Every node can be coordinator

- **masterless**
- **round robin** master for each request
- no **SPOF**
- **proxy** role



Consistency Model

Consistency Level

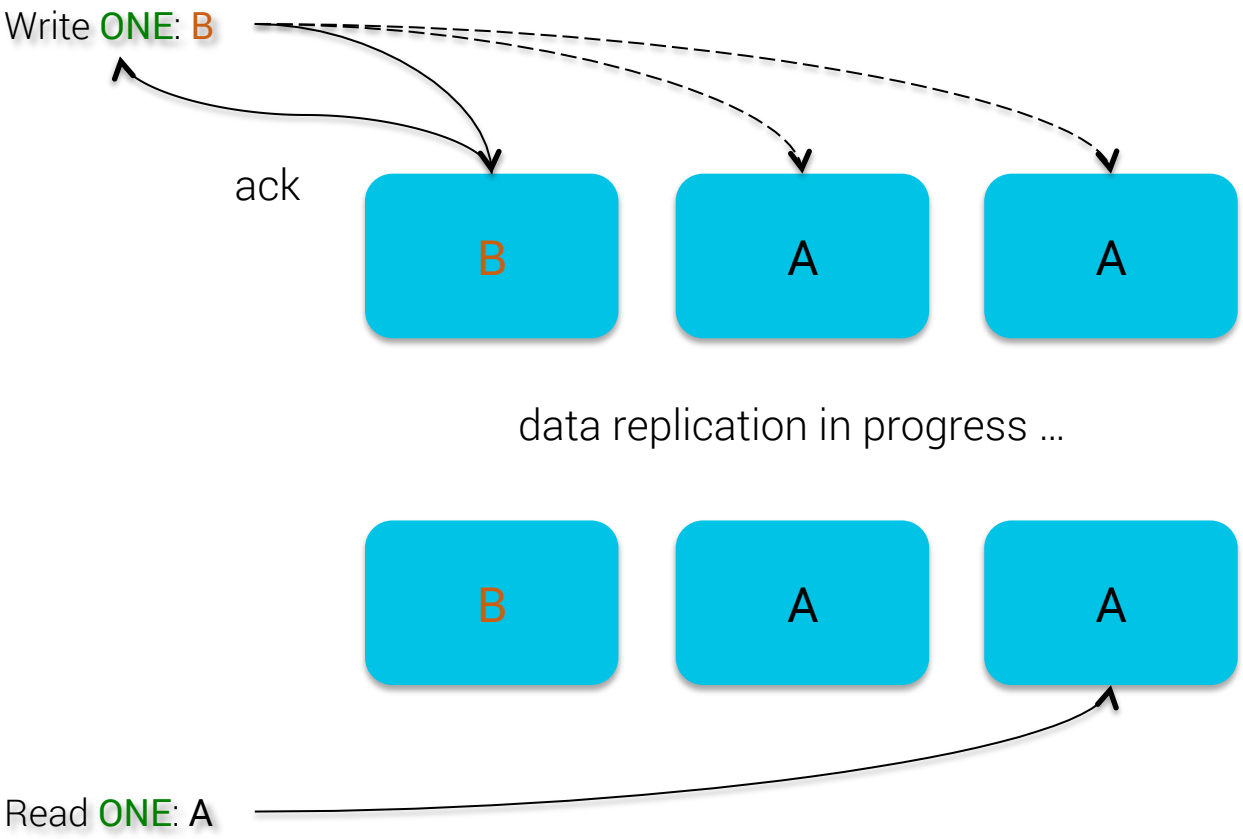
Tunable at **runtime**

- ONE
- QUORUM (strict majority w.r.t RF)
- ALL

Applicable to any request (**read**/**write**)

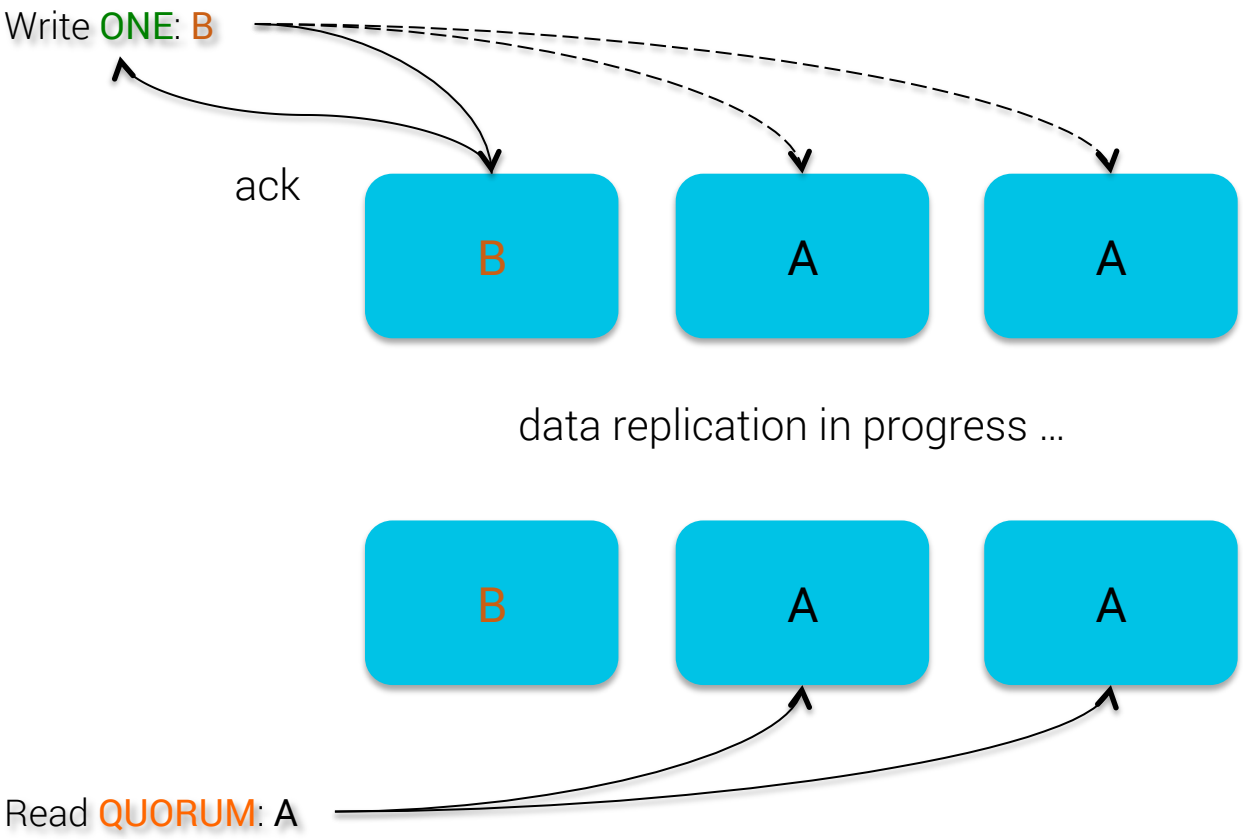
Consistency In Action

RF = 3, Write ONE, Read ONE



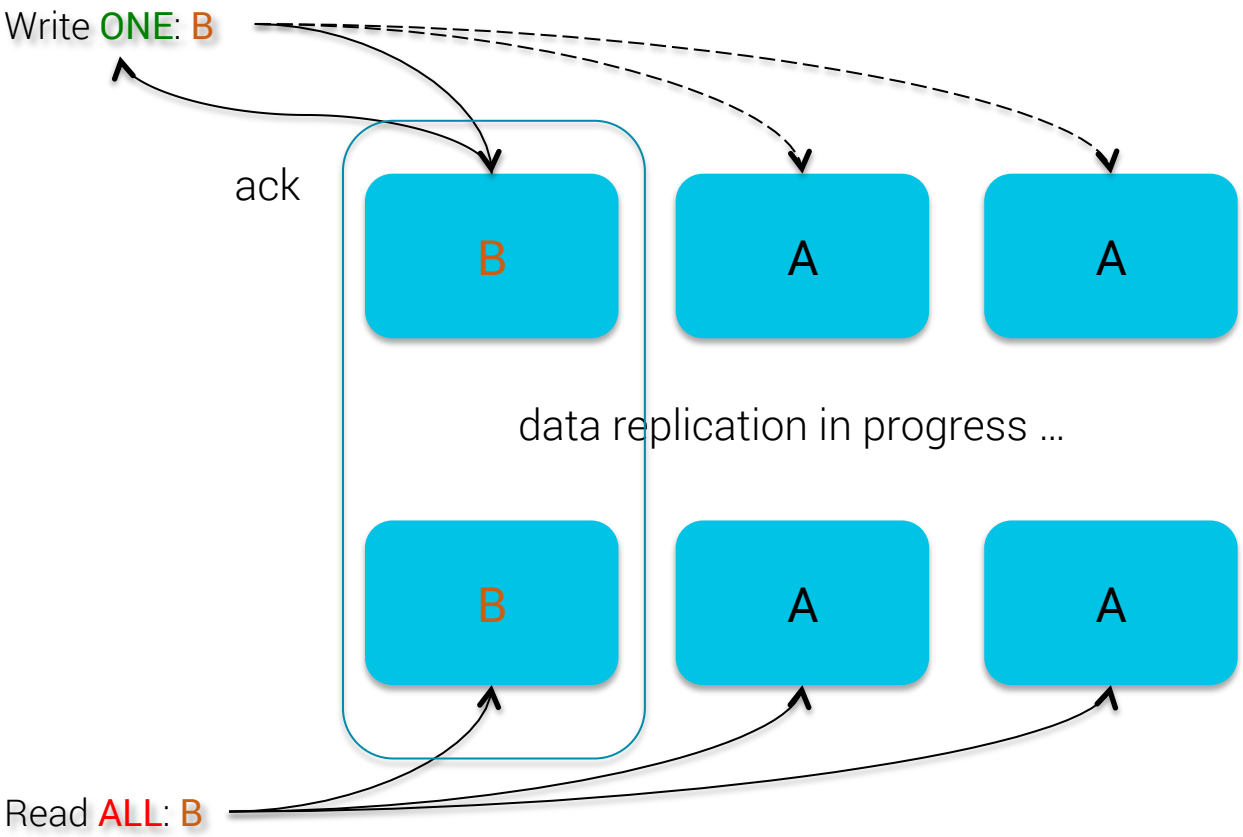
Consistency In Action

RF = 3, Write ONE, Read QUORUM

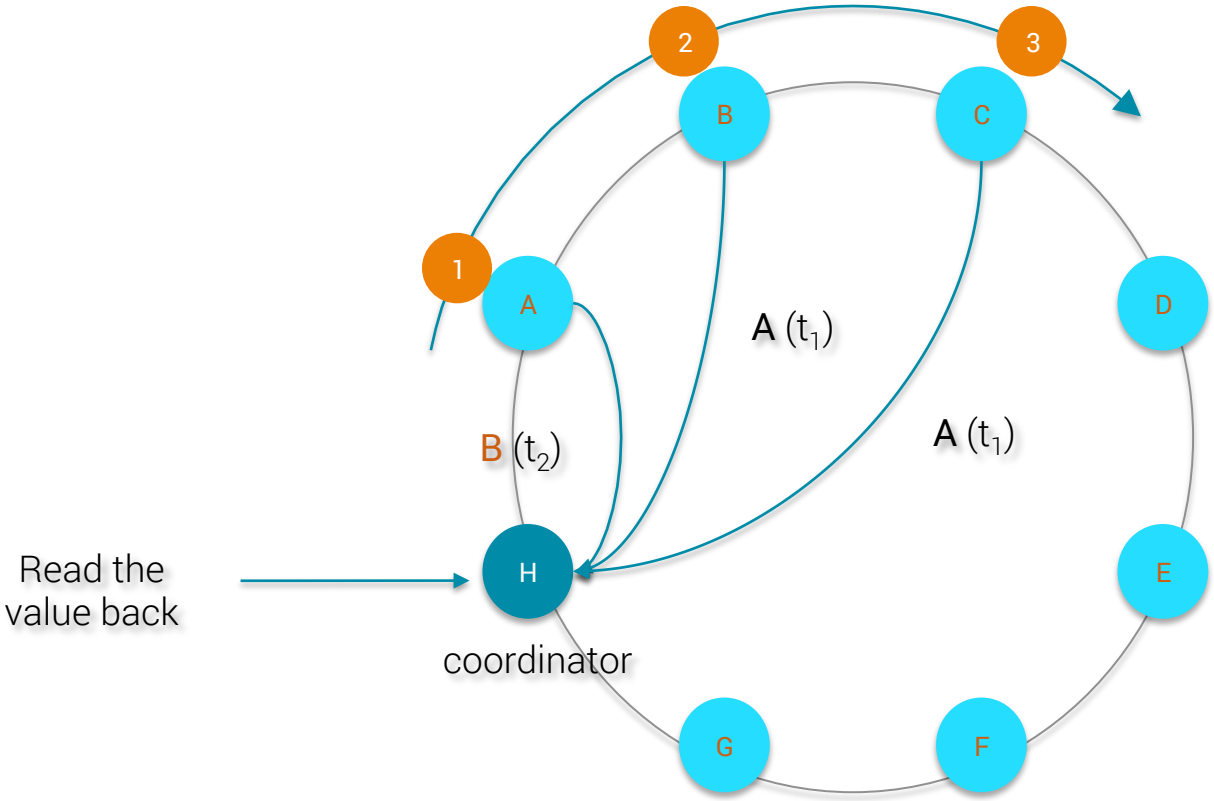


Consistency In Action

RF = 3, Write ONE, Read ALL

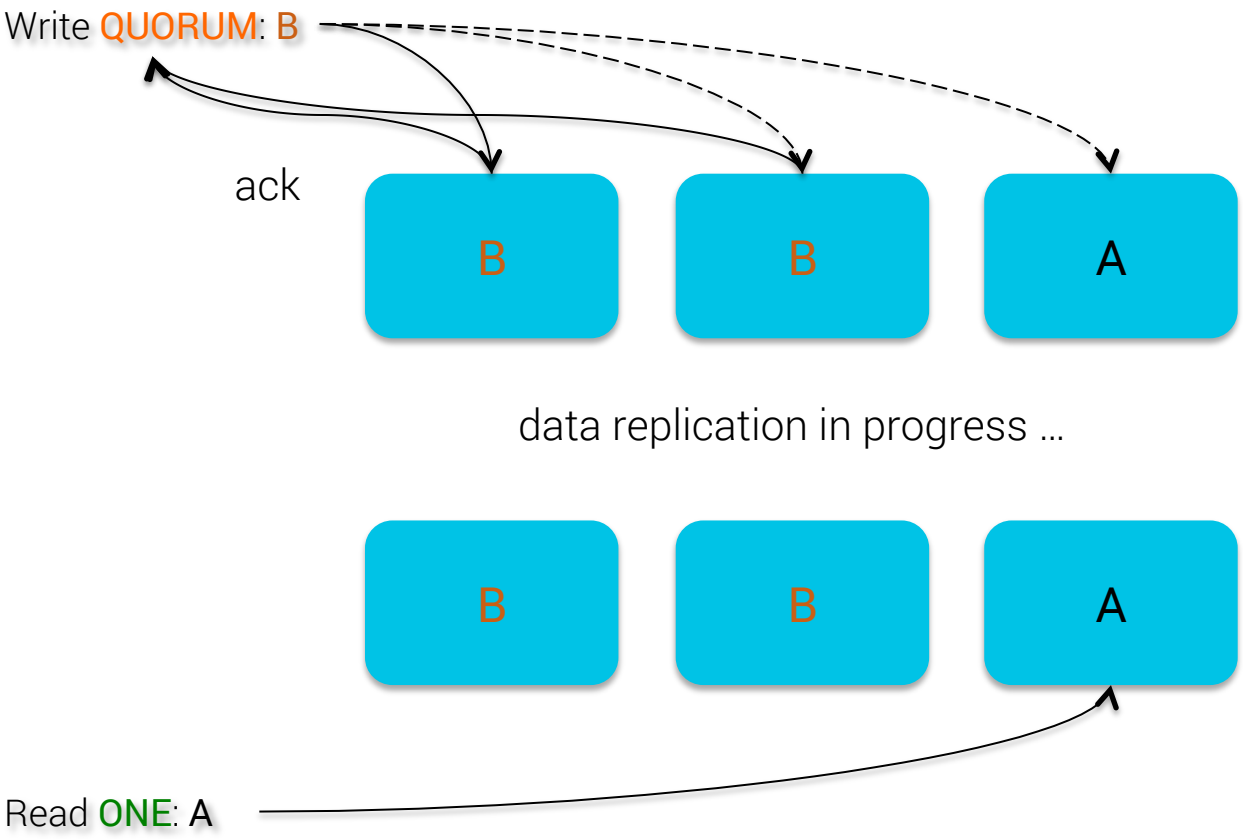


Last Write Win



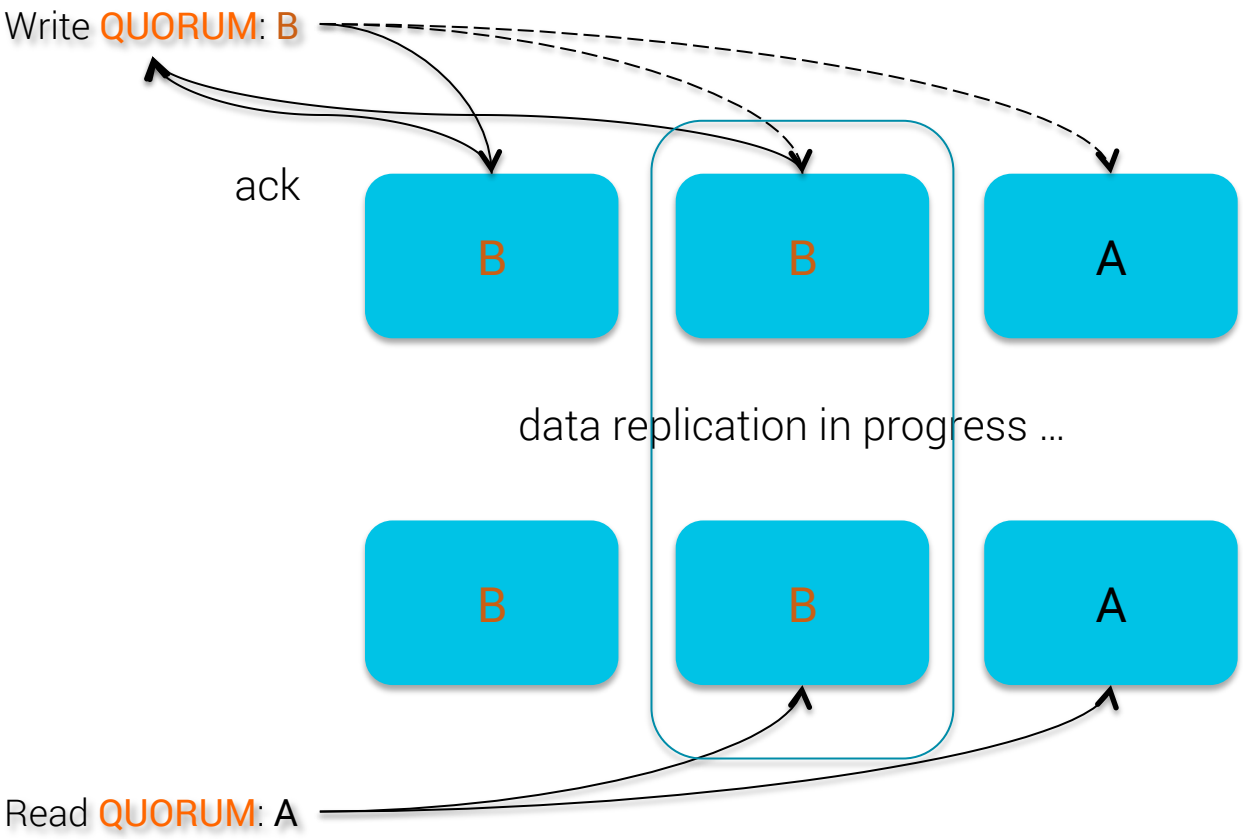
Consistency In Action

RF = 3, Write QUORUM, Read ONE



Consistency In Action

RF = 3, Write QUORUM, Read QUORUM



Consistency Level = Trade-off

Consistency Level

ONE

Fast, may not read latest written value

Consistency Level

QUORUM

Strict majority w.r.t. **R**eplication **F**actor

Good balance

Consistency Level

ALL

Paranoid

Slow, lost of high availability

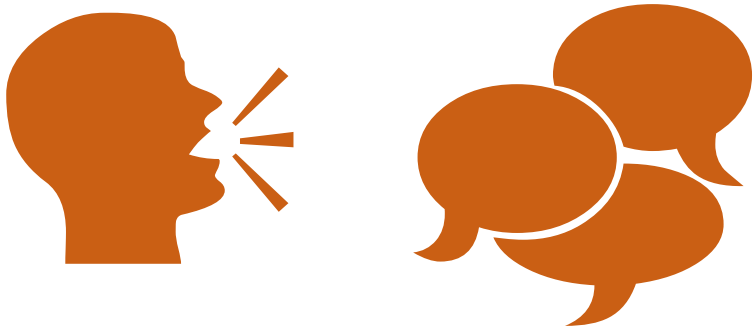
Consistency Level Common Patterns

$ONE_{Read} + ONE_{Write}$

☞ **available** for read/write even (**N-1**) replicas down

$QUORUM_{Read} + QUORUM_{Write}$

☞ **available** for read/write even if (**RF - 1**) replica (s) down



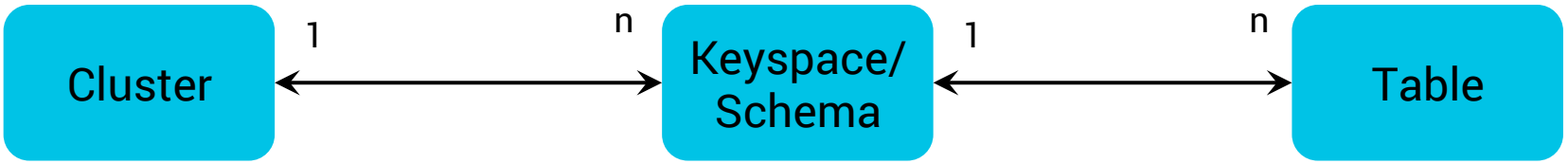
Q & A

Features Overview

What is a keyspace/schema ?

Simple table container

Defines how data are replicated in the cluster (RF)



Keyspace/Schema Creation

Single Data-center

```
CREATE KEYSPACE single_dc WITH REPLICATION =
{ 'class': 'SimpleStrategy', 'replication_factor': 3}
```

Multi Data-center

```
CREATE KEYSPACE multi_dc WITH REPLICATION =
{ 'class': 'NetworkTopologyStrategy', 'DC1': 3, 'DC2': 3, 'DC3': 1, …}
```

DDL Syntax

```
CREATE TABLE users (
        login text,
        name text,
        age int,
        ...
        PRIMARY KEY(login));
```

```
ALTER TABLE users ADD address text;
ALTER TABLE users DROP address;
...
DROP TABLE
```

DML Syntax

```
INSERT INTO users(login, name, age) VALUES('jdoe', 'John DOE', 33);
```

```
UPDATE users SET age = 34  WHERE login = 'jdoe';
```

```
DELETE age FROM users WHERE login = 'jdoe';
```

```
SELECT age FROM users WHERE login = 'jdoe';
```

Built-In Security Features

Role

- CREATE ROLE **x** WITH PASSWORD **y** *(NOSUPERUSER | SUPERUSER)*
- ALTER ROLE **x** WITH PASSWORD **y** *(NOSUPERUSER | SUPERUSER)*
- DROP ROLE **x**

Permissions

- **GRANT** <xxx> PERMISSION ON <resource> TO <role_name>
- **REVOKE** <xxx> PERMISSION ON <resource> FROM <role_name>

Collections

```
CREATE TABLE xxx(
   ...,
   li list<text>,
   se set<text>,
   ma map<int, text>,
   ...
);

UPDATE xxx SET li = li + [append] ...
UPDATE xxx SET se = se + {append}
UPDATE xxx SET ma[key] = value ...
```

User Defined Types

```
CREATE TYPE address (
   number int,
   street text,
   zipcode text,
   city text,
   country text
);

CREATE TABLE user (
    ...
    full_address address,
    ...
);
```

LightWeight Transactions

```
INSERT INTO users(...) VALUES(...) IF NOT EXISTS;

DELETE users WHERE ... IF EXISTS;

UPDATE users SET age = xxx WHERE ... IF age = 30;
```

Linearizable writes on a single partition

Time To Live

```
INSERT INTO users(...) VALUES(...) USING TTL = 3600;

UPDATE users USING TTL = 3600 SET age = xxx
WHERE ...;
```

Materialized Views

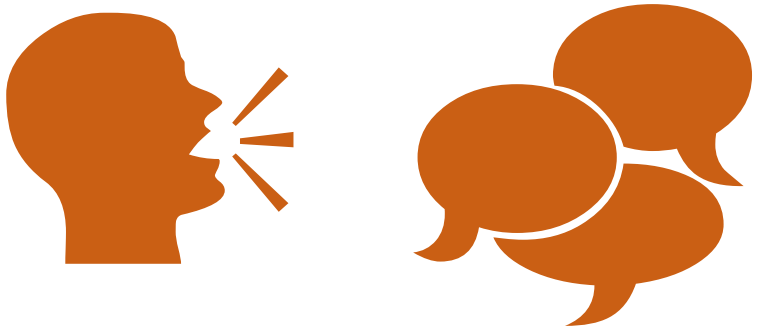
```
CREATE MATERIALIZED VIEW user_by_email
AS SELECT * FROM users
WHERE user_id IS NOT NULL AND email IS NOT NULL
PRIMARY KEY ((email), user_id);
```

JSON Syntax for INSERT/UPDATE/DELETE

```
CREATE TABLE users (
        id text PRIMARY KEY,
        age int,
        state text );

INSERT INTO users JSON '{"id": "user123", "age": 42, "state": "TX"}';

INSERT INTO users(id, age, state) VALUES('me', fromJson('20'), 'CA');

UPDATE users  SET age = fromJson('25') WHERE id = fromJson('"me"');

DELETE FROM users WHERE id = fromJson('"me"');
```

JSON Syntax for SELECT

```
SELECT JSON * FROM users WHERE id = 'me';
[json]
-------------------------------------
 {"id": "me", "age": 25, "state": "CA"}

SELECT JSON age,state FROM  users WHERE id = 'me';
[json]
-------------------------------------
 {"age": 25, "state": "CA"}

SELECT age, toJson(state) FROM users WHERE id = 'me';
age | system.tojson(state)
-----+--------------------
25 |                "CA"
```



Q & A