CASSANDRA SUMMIT2014 | EUROPE

London, U.K.
December 3 - 4

#CassandraSummit
Park Plaza Riverbank Hotel

Training Day | *December 3rd*

Beginner Track
- *Introduction to Cassandra*
- *Introduction to Spark, Shark, Scala and Cassandra*

Advanced Track
- *Data Modeling*
- *Performance Tuning*

Conference Day | *December 4th*

Cassandra Summit Europe 2014 will be the single largest gathering of Cassandra users in Europe. Learn how the world's most successful companies are transforming their businesses and growing faster than ever using Apache Cassandra.

http://bit.ly/cassandrasummit2014

Introduction to Cassandra for Java Developers

Why, What and How.

Johnny Miller, Solutions Architect
@CyanMiller
www.linkedin.com/in/johnnymiller

CASSANDRA SUMMIT2014 | EUROPE

London, U.K.
December 3 - 4

#CassandraSummit
Park Plaza Riverbank Hotel

Training Day | *December 3rd*

Beginner Track
- *Introduction to Cassandra*
- *Introduction to Spark, Shark, Scala and Cassandra*

Advanced Track
- *Data Modeling*
- *Performance Tuning*

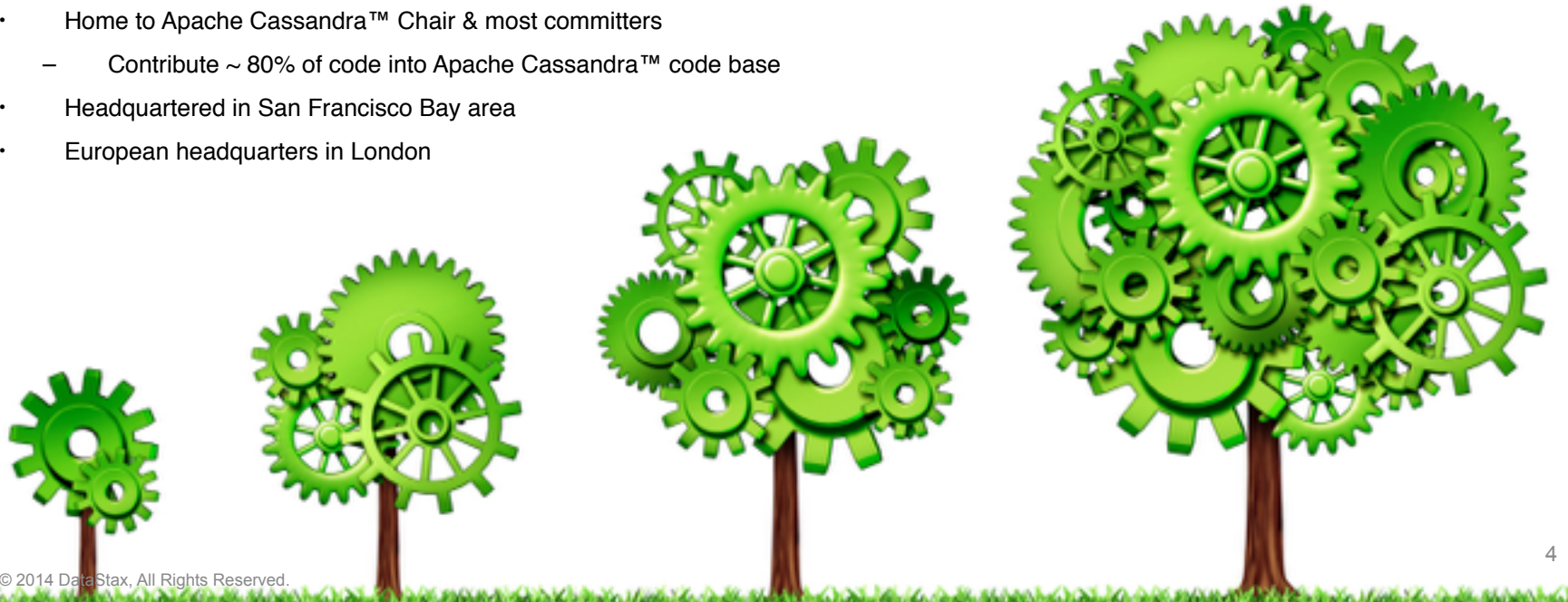
Conference Day | *December 4th*

Cassandra Summit Europe 2014 will be the single largest gathering of Cassandra users in Europe. Learn how the world's most successful companies are transforming their businesses and growing faster than ever using Apache Cassandra.

http://bit.ly/cassandrasummit2014

DataStax

- Founded in April 2010
- We drive Apache Cassandra™
- 300+ employees
- Home to Apache Cassandra™ Chair & most committers
  - Contribute ~ 80% of code into Apache Cassandra™ code base
- Headquartered in San Francisco Bay area
- European headquarters in London



DataStax Enterprise

www.datastax.com

DataStax Enterprise is free for startups!!!

- **Unlimited, free use of the software in DataStax Enterprise.**
- No limit on number of nodes or other hidden restrictions.
- If you're a startup, it's free!

www.datastax.com/startups

Training

- Checkout the DataStax Academy for **free** online training
  - **http://academy.datastax.com**
- Public courses
- On-site training

www.datastax.com/training

We are hiring!

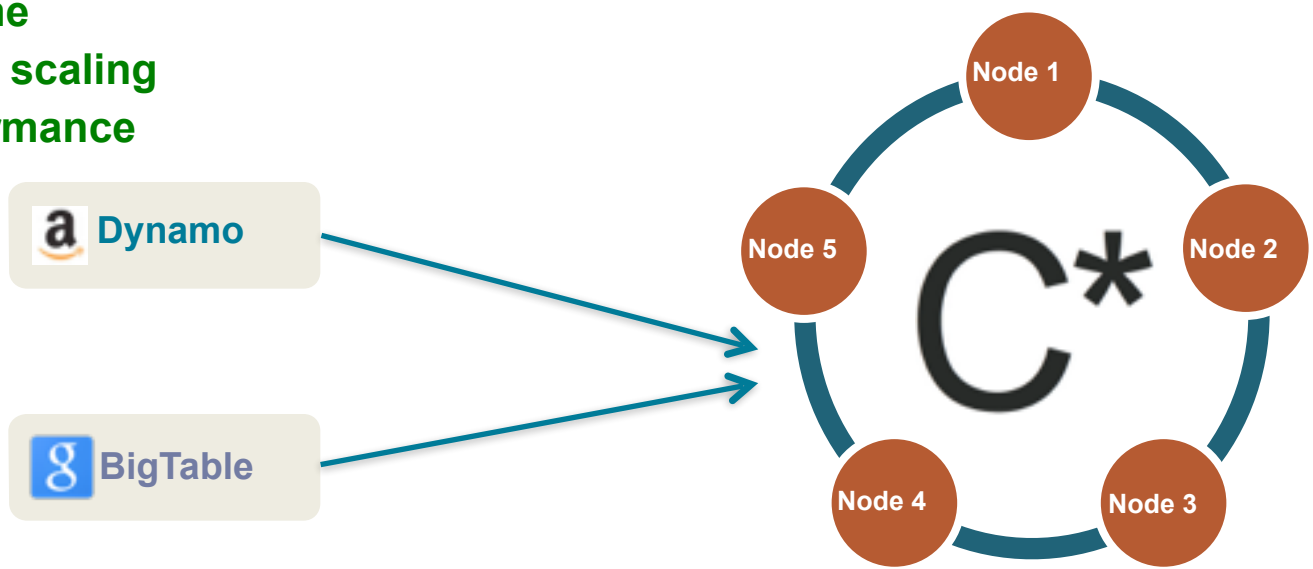
www.datastax.com/careers

@DataStaxCareers

Why Cassandra?

Apache Cassandra

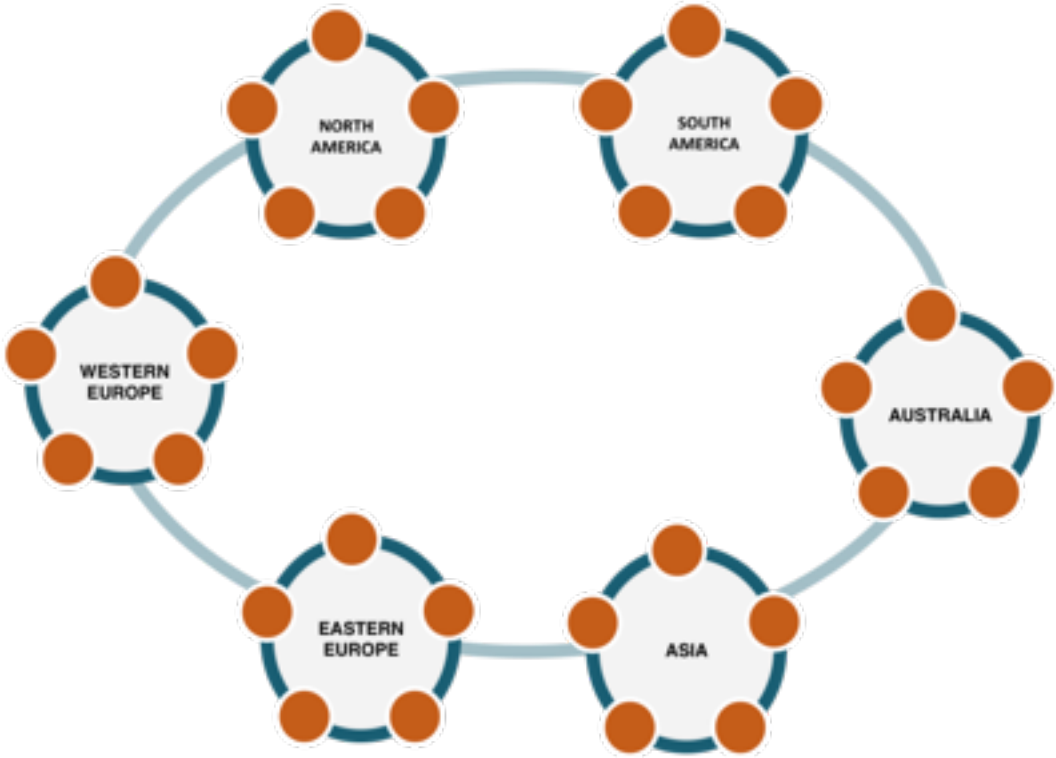
- Cassandra is a massively scaleable **open source** NoSQL **database**
- **100% uptime**
- **Predictable scaling**
- **High Performance**



BigTable: http://research.google.com/archive/bigtable-osdi06.pdf
Dynamo: http://www.allthingsdistributed.com/files/amazon-dynamo-sosp2007.pdf

Cassandra Core Values

- Ease of use
- Massive scalability
- High performance
- Always Available



Availability and Speed Matters for online apps!

- UK retailers **lost 8.5 billion last year to slow web sites**, which is 1 million for every 10 million in online sales
- Over half of all web users expect a **response time of 2 seconds or less**
- A 1 second delay causes a nearly 10% reduction in customer interactions
- A 1 second decrease in Amazon page load time costs the company $1.6 billion in sales

Adoption

Rank	Last Month	DBMS	Database Model	Score	Changes
1.	1.	Oracle	Relational DBMS	1466.91	-3.95
2.	2.	MySQL	Relational DBMS	1297.14	+15.92
3.	3.	Microsoft SQL Server	Relational DBMS	1208.87	-33.62
4.	4.	PostgreSQL	Relational DBMS	255.79	+5.94
5.	5.	MongoDB	Document store	240.98	+3.63
6.	6.	DB2	Relational DBMS	197.03	-9.39
7.	7.	Microsoft Access	Relational DBMS	140.48	+0.86
8.	8.	SQLite	Relational DBMS	92.61	+3.74
9.	↑ 10.	Cassandra	Wide column store	87.86	+5.96
10.	↓ 9.	Sybase ASE	Relational DBMS	85.42	-0.75
11.	↑ 12.	Solr	Search engine	75.77	+6.74
12.	↓ 11.	Redis	Key-value store	74.60	+3.80
13.	13.	Teradata	Relational DBMS	66.15	+0.77
14.	14.	FileMaker	Relational DBMS	52.63	+0.57
15.	15.	HBase	Wide column store	45.02	+3.11
16.	16.	Elasticsearch	Search engine	41.52	+2.72
17.	17.	Informix	Relational DBMS	33.45	+0.29
18.	18.	Memcached	Key-value store	31.59	+0.60
19.	19.	Hive	Relational DBMS	31.40	+0.59
20.	20.	Splunk	Search engine	29.76	+2.15
21.	21.	CouchDB	Document store	25.97	+1.84
22.	22.	Neo4j	Graph DBMS	24.22	+1.31
23.	23.	SAP HANA	Relational DBMS	23.18	+0.86
24.	24.	Couchbase	Document store	19.67	+2.13

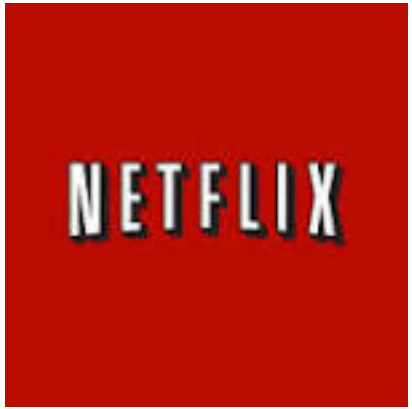
http://db-engines.com/en/ranking
September 2014

.com Cemetry 8digits academicworks ACTUATE AddThis adgear AGENTIS
airpim Algar ALLIED MYND ALTOROS Answers.com Aol. apereo apigee APIXIO applifier
Apptegic Arise ARKIVUM Atigeo avanade AZUL backupify base be-mobile BOSS
BlackBerry booshaka boundary BOXEVER BrightGauge BROCADE Burt. bwin.party Energy CABLEVISION CABOT canary
caplore cardspring careerbuilder CARTERA caserta CharityUSA Chegg Chill Cinnober CISCO CloudHealth CloudPassage
cloudsoft cloze comverge COMCAST conestone correlor coursera Credit Suisse CrowdRiff CSR
CURALATE DATADOG Demandforce DeNA DesireToLearn digby Digital River Disney DISQUS
docomo DRILLSTER druva Dyn bros ebay eNanoPay ENGINE enkata epsilon EUROTECH
xecom EXOSITE FanBridge FedEx feedzai fold3 FullContact
GAMEFLY Gazzang gemalto GIGASPACES GitHub qlispa Globant GOGRID GoSquared GOTIME!
GRAINGER GRIDPOINT grubHub GX Healthline Healthx HOBSONS HubSpot hulu IBM
iland IMPETUS infor ING inkling instaclustr integral ID intuit ISIDOREY
iSIGHTPARTNERS ITHAKA ixonos JASPERSOFT Jetlore Jini juniper KASPERSKY Keen IO kik
KISSmetrics KNEWTON krux kwarter kweo lacuna LOR LeaseEagle Libon librato. LIFESTREET Lithium LiveOffice
livefyre LIVEPERSON loop logic lucid LYTICS Mahalo MARKETO martini media mate1.com medidata medio Melt
Microsoft midokura mnubo Mollom monitis moodyo MORNINGSTAR MOVIO nanigans napshare nativeX
NAVTEQ NBCUniversal nearbuy NetSeer netvibes neustar ngmoco NREL SCALED Data Security
NUTANIX ONEsite onesto OOYALA OBI OpenAdExchange OpenMarket opentracker OpenX opsmatic orange
Other outbrain paddypower pagerduty Palantir paper.li parseix pentaho PeopleBrowsr
PERFECT WORLD PETER-SERVICE PING4 PitneyBowes PLAYGEN POLYVORE PRESS priceline.com proofpoint PROS. ProtectWise PULSAR
Qualtera quandl quantcast QUOVA Rakuten RAZORlabs realo Recruiting.com RED BRICK MEDIA referentia
RelateIQ Relay42 Reltio retailigence Rhapsody RIGHTSCALE riptide rockyou ruckus Sabre SAFEWAY SanDisk SBR.net
Scandit Scoop.it screen9 SeaChange SEEMOTO SENSIRION sessionm SEVEN ShareThis SHAZAM SHIFT
SHODAN shopsavvy shore.li SIGNAL SimpleGeo simplereach singlewire sirca SITA sitevision СКБ КОНТУР smeet SMX
SocialArtisan SOCIALFLOW softonic Software Projects Sonico spidertracks splunk Spotify spotXchange Spredfast springcm Springer
springfuse springpad SPROUTSOCIAL Startapp Stormpath stratio Streamer struq sumologic SwiftKey Symantec
SYNAPSE Taboola talentica talkbits Teads teamfocus Telefonica TENDRIL echonest The New York Times
thinglink THOMSON REUTERS TILDE tink TinyCo TOPTARIF TRADEMOB TRADING TECHNOLOGIES trapit Twilio TRICAST trivadis truccin
Twenga TWIMPACT twitpic UMBEL UNBXD Vast VEEZIO viafoura ViaSat VigLink
VisualDNA vmware VTEX WyD wajam Walmart wayin we heart it Webcollage here whisper wibidata WIDESPACE
WILDFIRE WILLIAMS-SONOMA WooCommerce WORKWARESYSTEMS WSO2 Xebia xobni XOOM YAHOO!
zangbezang zeebox Zemanta ZIPZAPPLAY zoemob ZOHAR zoominfo [24]7

Performance & Scale

Cassandra works for small to huge deployments.

- Cassandra footprint @ Netflix
- 80+ Clusters
- 2500+ nodes
- 4 Data Centres (Amazon Regions)
- > 1 Trillion transactions per day



See: *http://www.datastax.com/resources/casestudies/netflix*

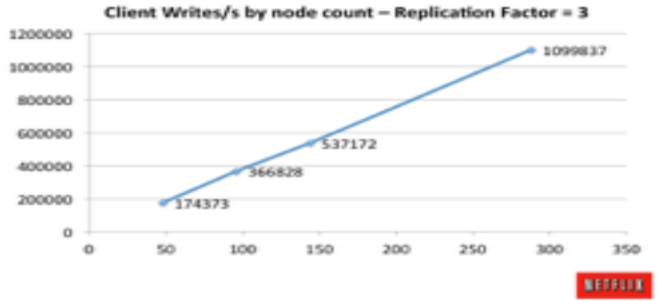
Performance & Scale

"***In terms of scalability, there is a clear winner throughout our experiments. Cassandra achieves the highest throughput for the maximum number of nodes in all experiments with a linear increasing throughput.***"

Solving Big Data Challenges for Enterprise Application Performance Management, Tilman Rable, et al., August 2012. Benchmark paper presented at the Very Large Database Conference, 2012.
http://vldb.org/pvldb/vol5/p1724_tilmannrabl_vldb2012.pdf

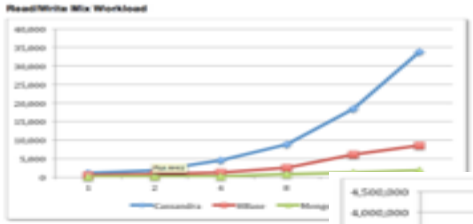
Netflix Cloud Benchmark…

Scale-Up Linearity

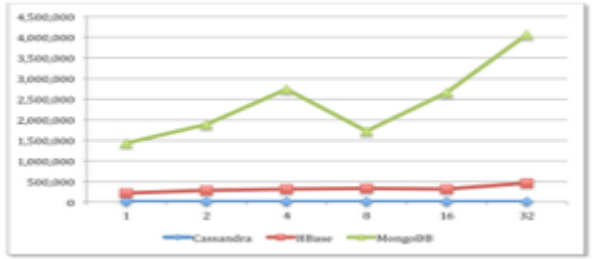


End Point Independent NoSQL Benchmark

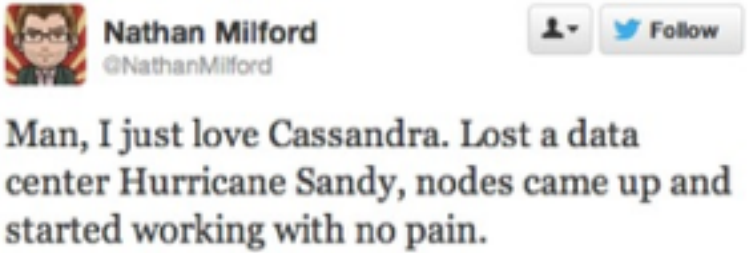
Highest in throughput…

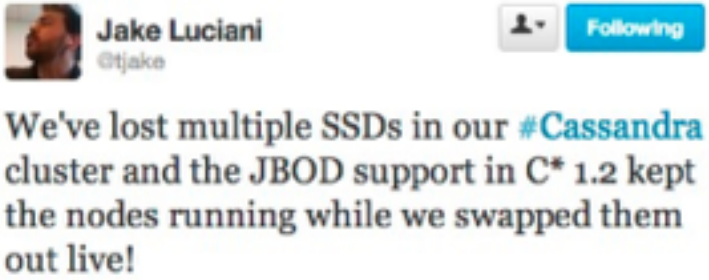


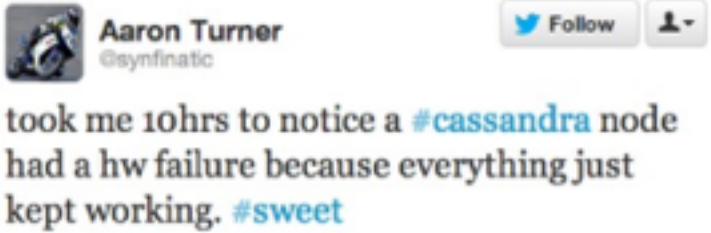
Lowest in latency…

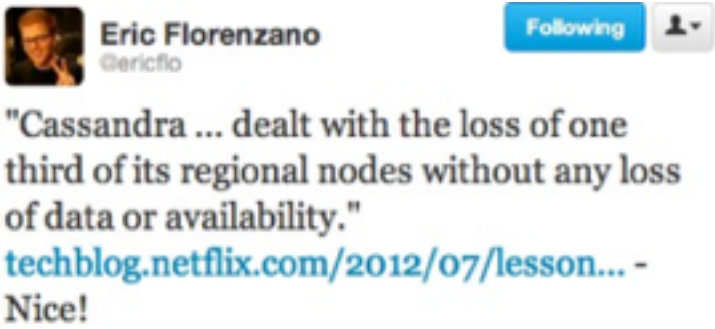


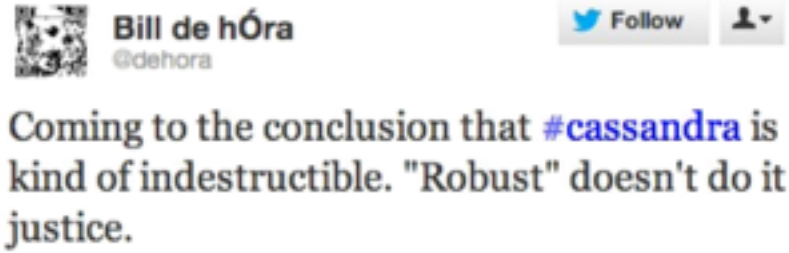
Availability











Ideal Cassandra Use Cases

Product catalogs and playlists
- Collection of items

Internet of things
- Sensor data

Messaging
- Emails, instant messages, alerts, comments

Recommendation and personalization
- Trend information

Fraud detection
- Data patterns

Time-series and time-ordered data
- Data with time dimension

http://planetcassandra.org

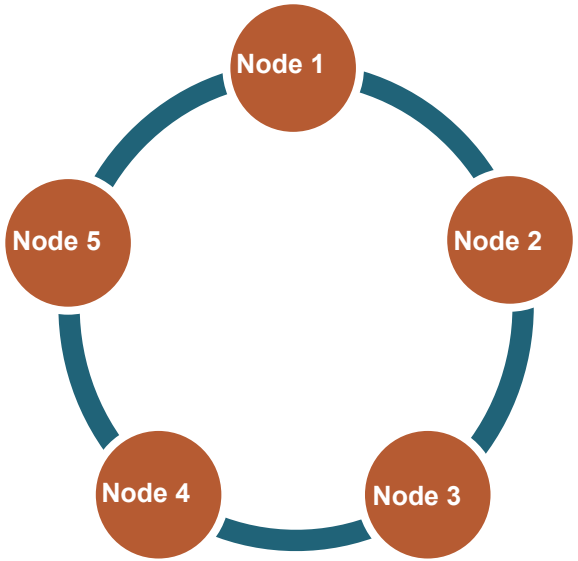
DATASTAX

comSysto

What is Apache Cassandra?

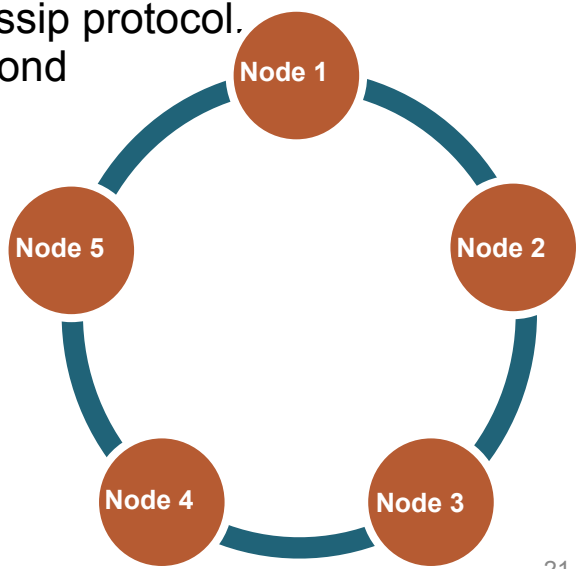
Overview

- Cassandra was designed with the understanding that system/hardware failures can and do occur
- Peer-to-peer, distributed system
- **All nodes the same**
- **Data partitioned among all nodes** in the cluster
- Custom data replication to ensure fault tolerance
- Read/Write-anywhere and across data centres



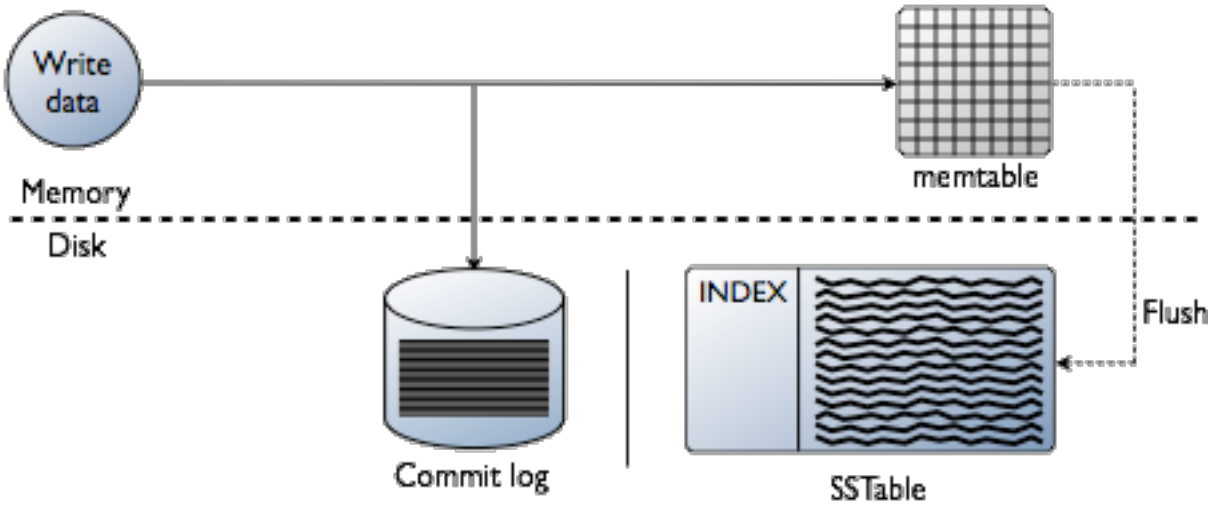
Cassandra > 1 Server

- All nodes participate in a cluster
- Add or remove as needed
- All nodes the same – **masterless** with no single point of failure
- Each node communicates with each other through the Gossip protocol, which exchanges information across the cluster every second
- **Data partitioned among all nodes in the cluster**
- **More capacity? Add a server!**
- **More throughput? Add a server!**



Write Path on individual server

- **A commit log** is used on each node to capture write activity. Data durability is assured
- **Data also written to an in-memory structure** (memtable) and then to disk once the memory structure is full (an SStable)

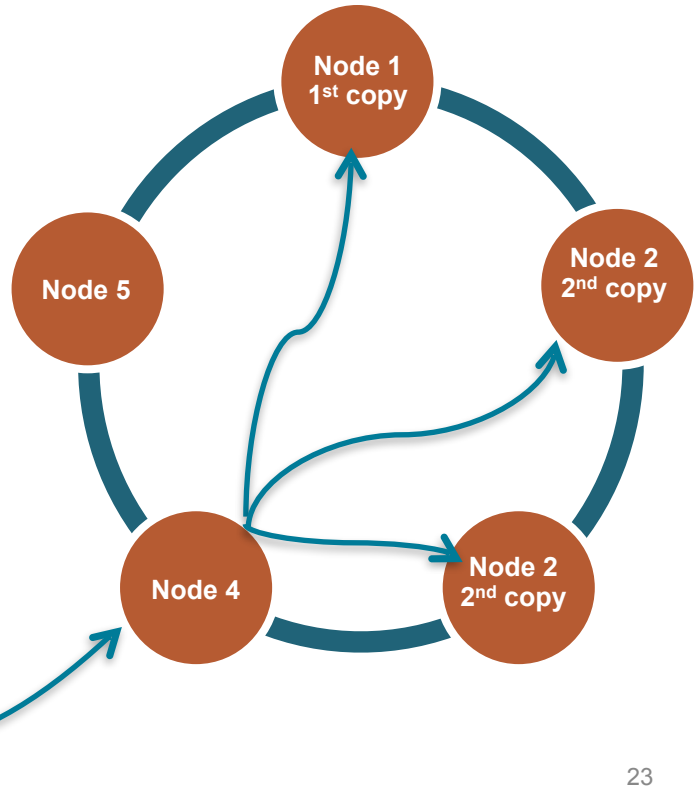


Distributed

- Client reads or writes **to any node**
- Node **coordinates** with others
- Data read or replicated in **parallel**

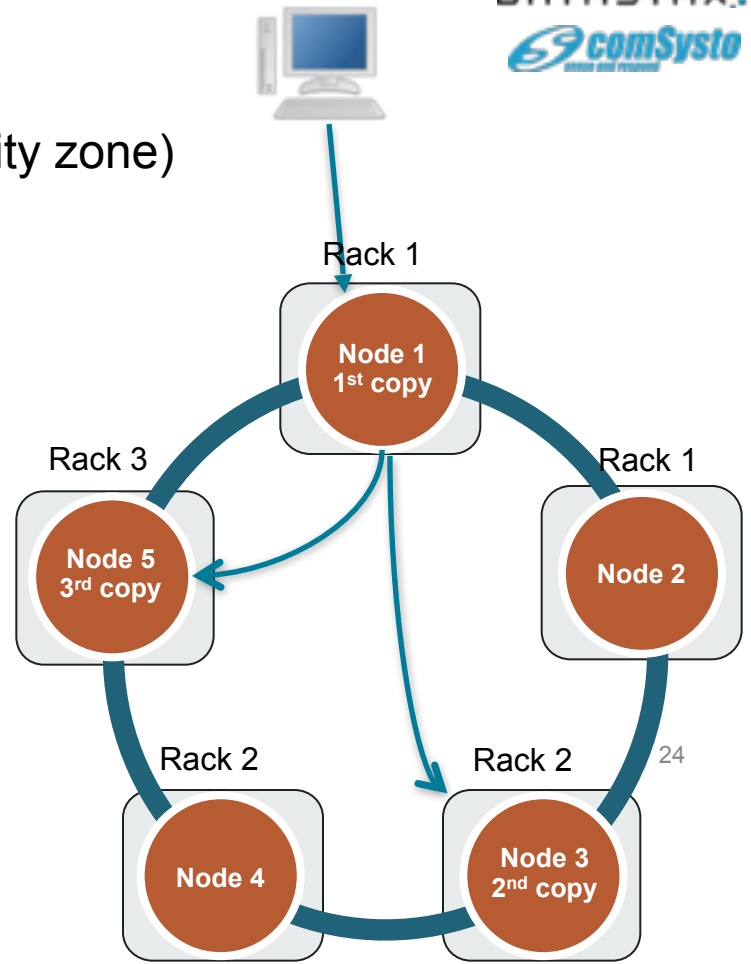
- Replication factor (RF): How many copies of your data?
- RF = 3 in this example
- Each node is storing 60% of the clusters total data i.e. 3/5

Handy Calculator: http://www.ecyrd.com/cassandracalculator/



Rack Aware

- Cassandra is aware of which rack (or availability zone) each node resides in.
- **It will attempt to place each data copy in a different rack.**
- RF = 3 in this example



Data Distribution and Replication

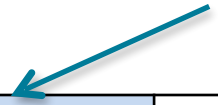
- Cassandra is **designed as a peer-to-peer system that makes copies of the data** and distributes the copies among a group of nodes
- **Data is organized by table and identified by a primary key**. The primary key determines which node the data is stored on.
- **Each node is assigned data based on the hash value of the primary key** and the range of hash values associated with that node.

Partitioners

- **A partitioner determines how data is distributed across the nodes in the cluster.**
- Cassandra has three:
  - Murmur3Partitioner (default)
  - RandomPartitioner
  - ByteOrderedPartitioner
    (don't use this unless you know what your doing and have a very good reason to do this)

Murmer3 Example

Primary Key

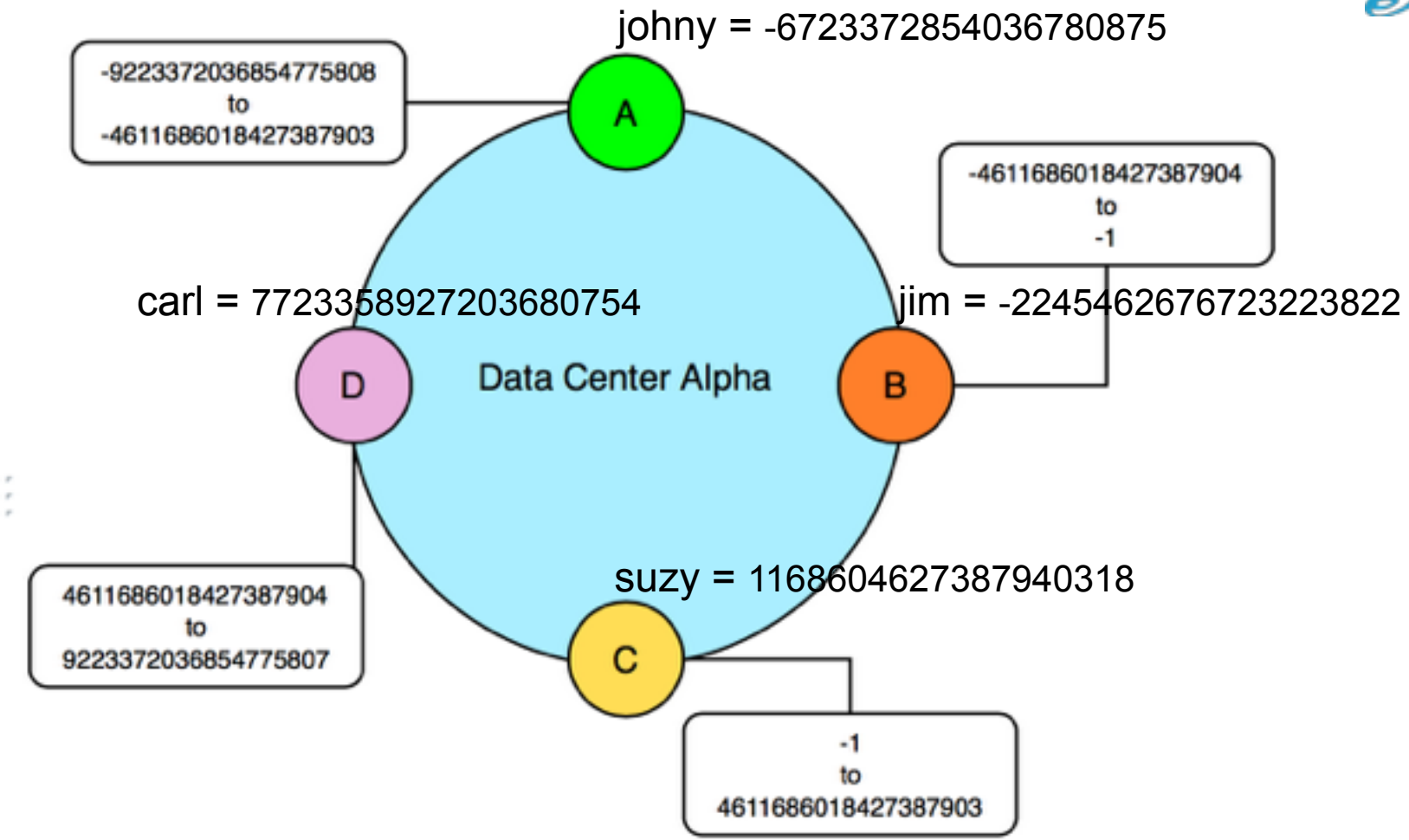


- **Data:**

jim	age: 36	car: ford	gender: M
carol	age: 37	car: bmw	gender: F
johnny	age: 12	gender: M	
suzy:	age: 10	gender: F	

- **Murmer3 Hash Values:**

Primary Key	Murmur3 hash value
jim	-2245462676723223822
carol	7723358927203680754
johnny	-6723372854036780875
suzy	1168604627387940318


johny = -6723372854036780875
-9223372036854775808
to
-4611686018427387903
A
-4611686018427387904
to
-1
carl = 7723358927203680754
jim = -2245462676723223822
D
Data Center Alpha
B
4611686018427387904
to
9223372036854775807
suzy = 1168604627387940318
C
-1
to
4611686018427387903

Murmer3 Example

Data is distributed as:

Node	Start range	End range	Primary key	Hash value
A	-9223372036854775808	-4611686018427387903	johnny	-6723372854036780875
B	-4611686018427387904	-1	jim	-2245462676723223822
C	0	4611686018427387903	suzy	1168604627387940318
D	4611686018427387904	9223372036854775807	carol	7723358927203680754

Replicas

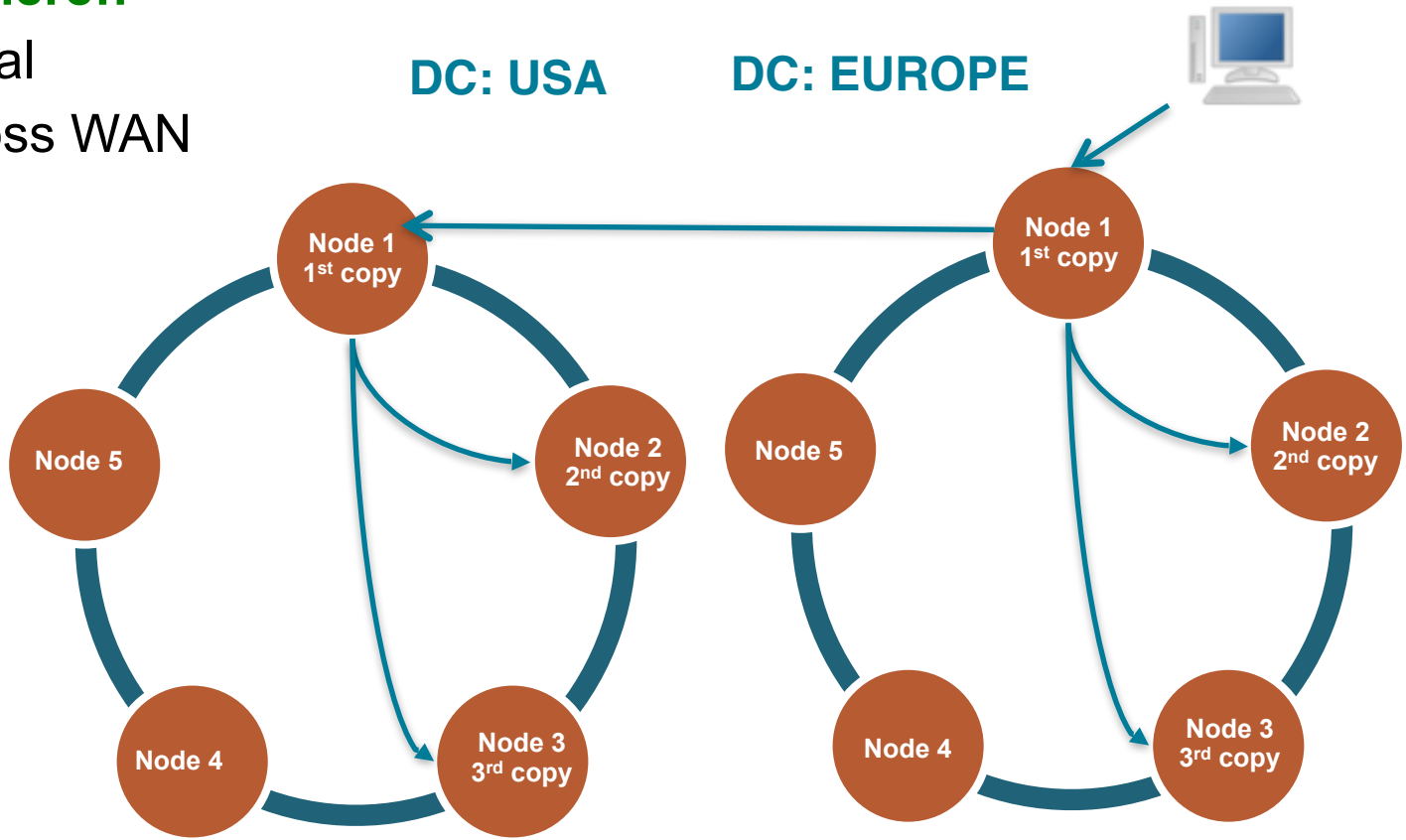
- A replication strategy determines the nodes where replicas are placed.
- All replicas are equally important; **there is no primary or master replica**.
- **Two replication strategies available**:
  - ***SimpleStrategy*** **(prototype only, don't use in production**)
  - ***NetworkTopologyStrategy***
- Replication strategy is set at the keyspace/schema level.

NetworkTopologyStrategy

- Specified how many replicas you want in each data center (physical or virtual).
- **NetworkTopologyStrategy places replicas in the same data center by walking the ring clockwise until reaching the first node in another rack.**

Data Center Aware

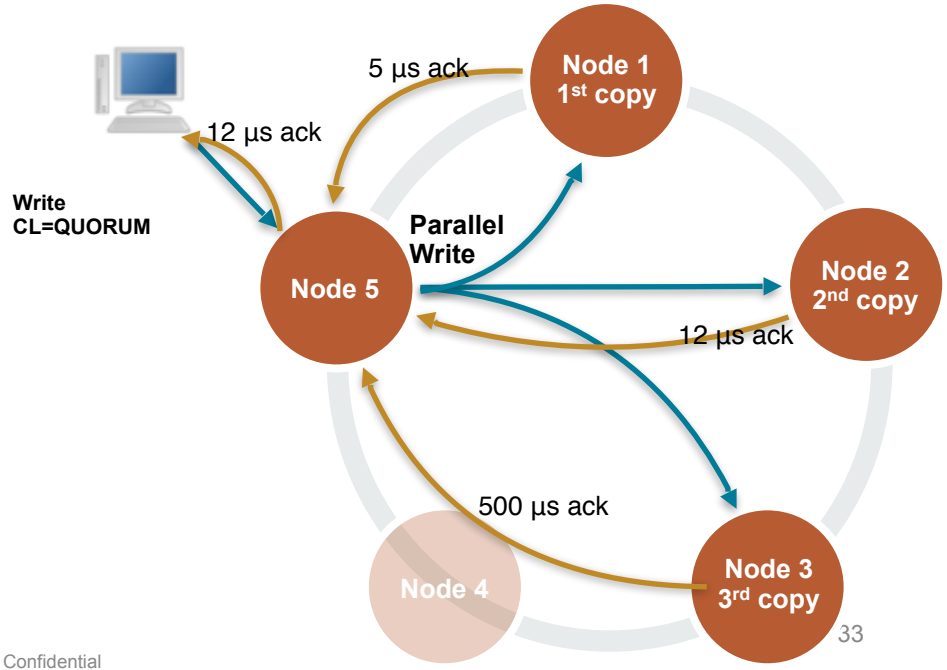
- **Active Everywhere!!**
- Client writes local
- Data syncs across WAN



Tunable Consistency

- Consistency Level (CL)
- **Client specifies per operation**
- Handles multi-data center operations

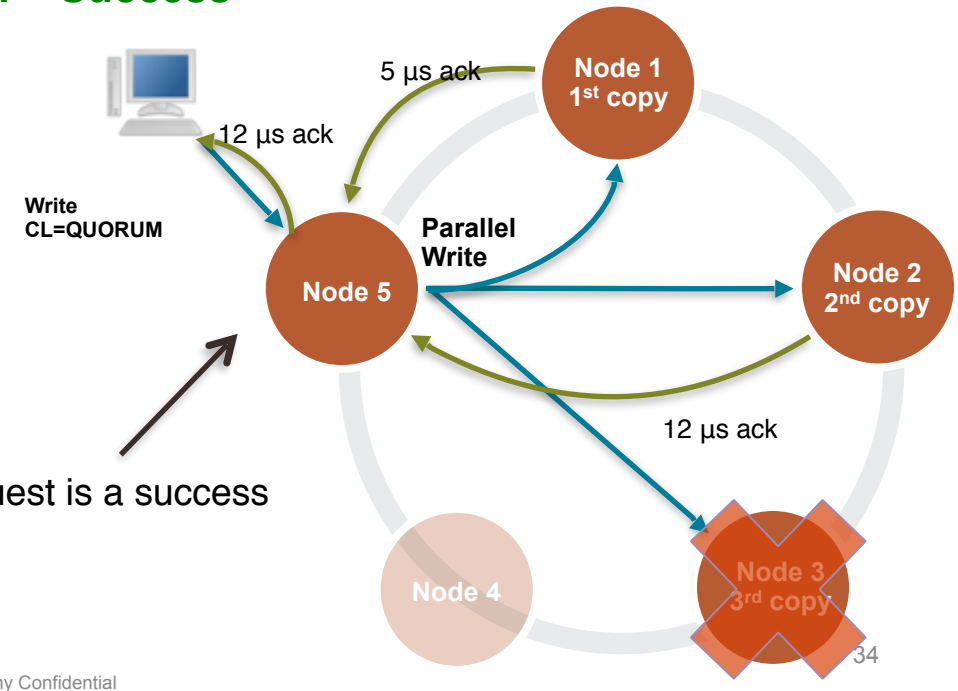
ALL = All replicas ack
QUORUM = > 51% of replicas ack
LOCAL_QUORUM = > 51% in local DC ack
ONE = Only one replica acks
Plus more…. (see docs)



Node Failure

- **A single node failure shouldn't bring failure.**
- **Replication Factor + Consistency Level = Success**

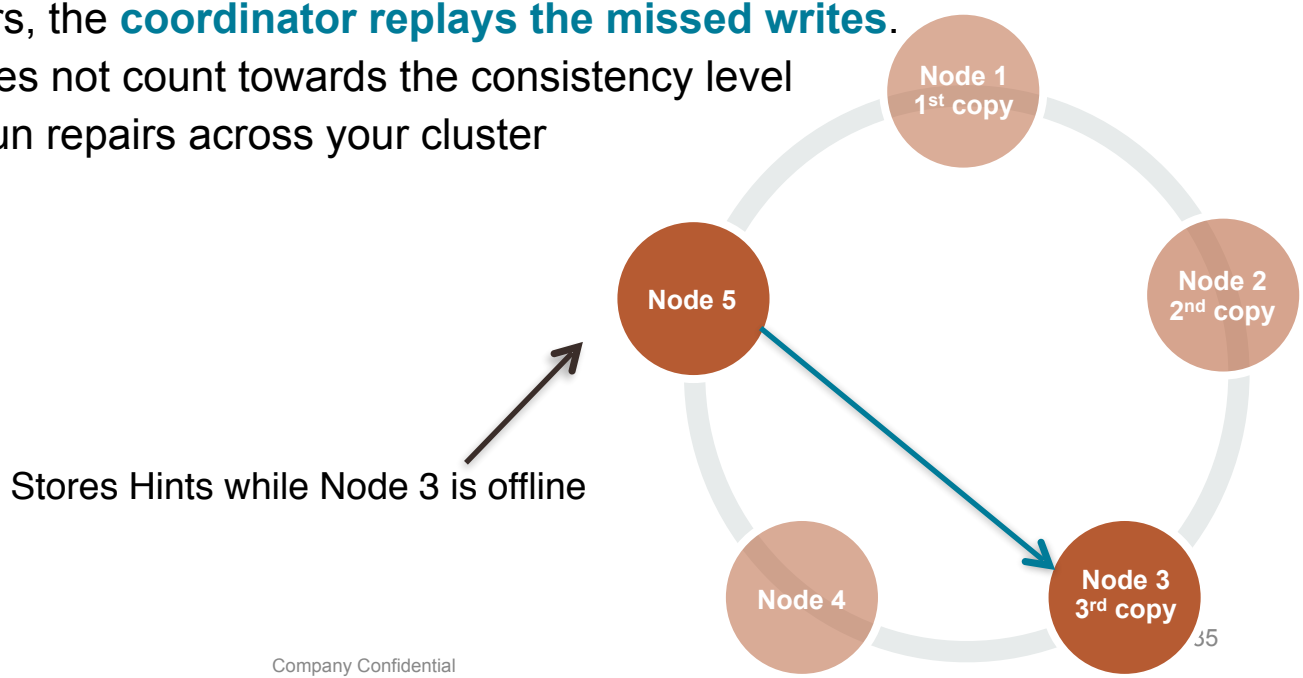
- This example:
  - RF = 3
  - CL = QUORUM



>51% ack – so request is a success

Node Recovery

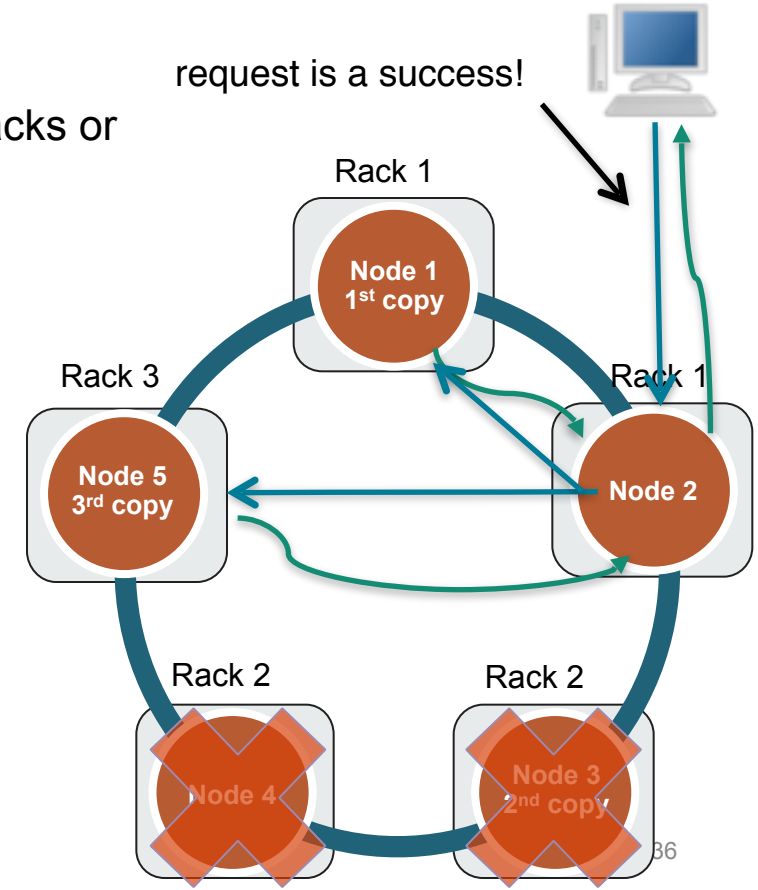
- When a write is performed and a replica node for the row is unavailable the coordinator **will store a hint locally.**
- When the node recovers, the **coordinator replays the missed writes**.
- **Note:** a hinted write does not count towards the consistency level
- **Note:** you should still run repairs across your cluster



Rack Failure

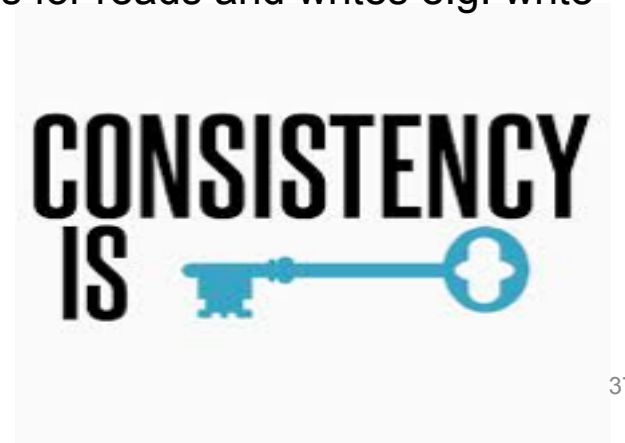
- Cassandra will place the data in as many different racks or availability zones as it can.

- This example:
  - RF = 3
  - CL = QUORUM
  - Rack 2 fails
- **Data copies still available in Node 1 and Node 5**
- **Quorum can be honored i.e. > 51% ack**



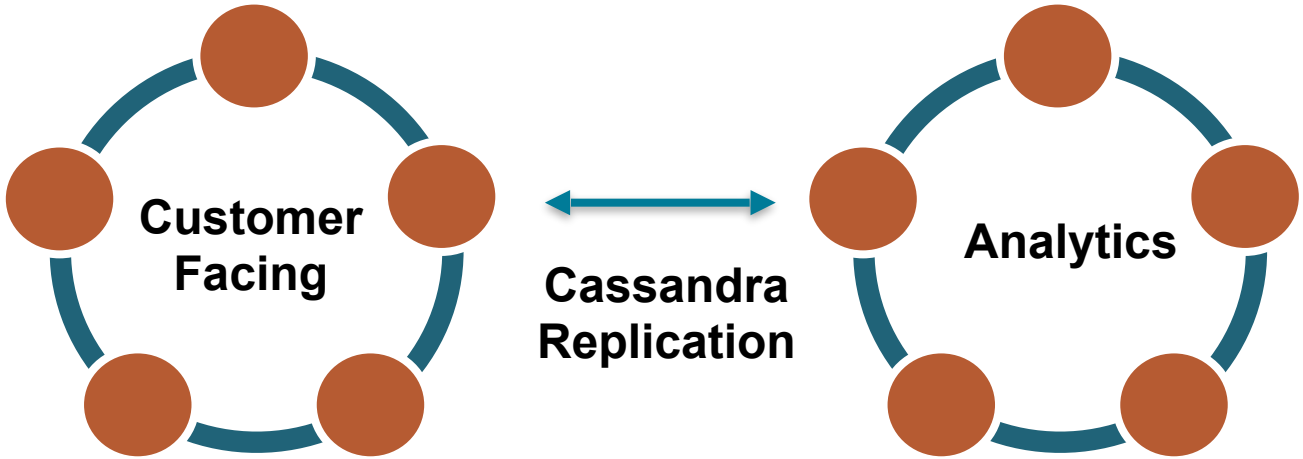
Don’t be afraid of weak consistency

- **More tolerant to failure**
- Consistency Level of 1 is the most popular (I think)
- If you want stronger consistency go for LOCAL_QUORUM i.e. quorum is honored in the local data centre.
- If you go stronger than LOCAL_QUORUM – really understand what this means and why you are doing it.
- Remember – you can have different consistency levels for reads and writes e.g. write with CL:1, read with CL:LOCAL_QUORUM



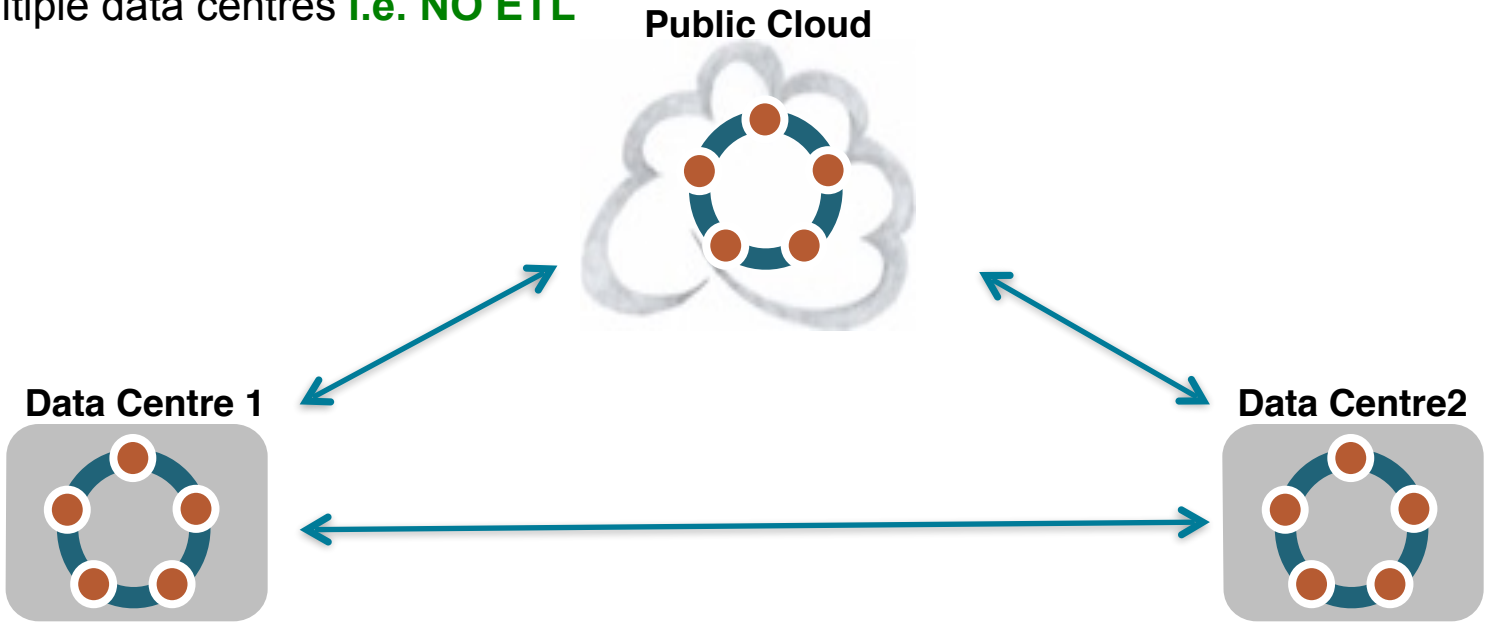
Ring fenced resources

- If you need to isolate resources for different uses, Cassandra is a great fit.
- You can create separate virtual data centres optimised as required – different workloads, hardware, availability etc..
- Cassandra will replicate the data for you automatically – **no ETL is necessary**



Hybrid Cloud

- **Cassandra is fmulti-data centre and cloud capable**
- Data written to any node is automatically and transparently written to all other nodes in multiple data centres **i.e. NO ETL**



Security

FEATURES

Internal Authentication
Manages login IDs and passwords inside the database

Object Permission Management
controls who has access to what and who can do what in the database

Client to Node Encryption
protects data in flight to and from a database cluster

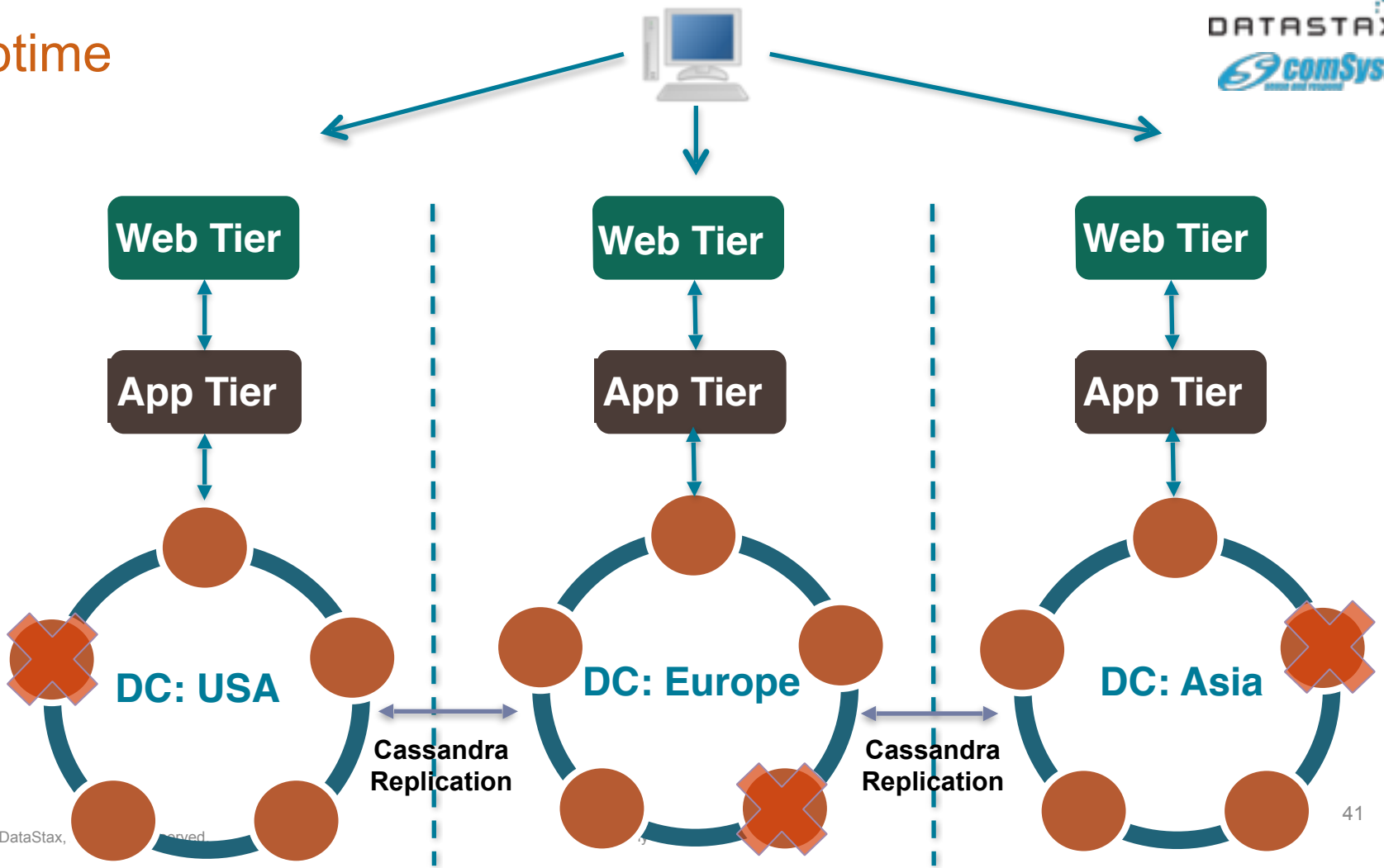
BENEFITS

- Ensures only authorized users can access a database system using internal validation
- Simple to implement and easy to understand
- No learning curve from the relational world

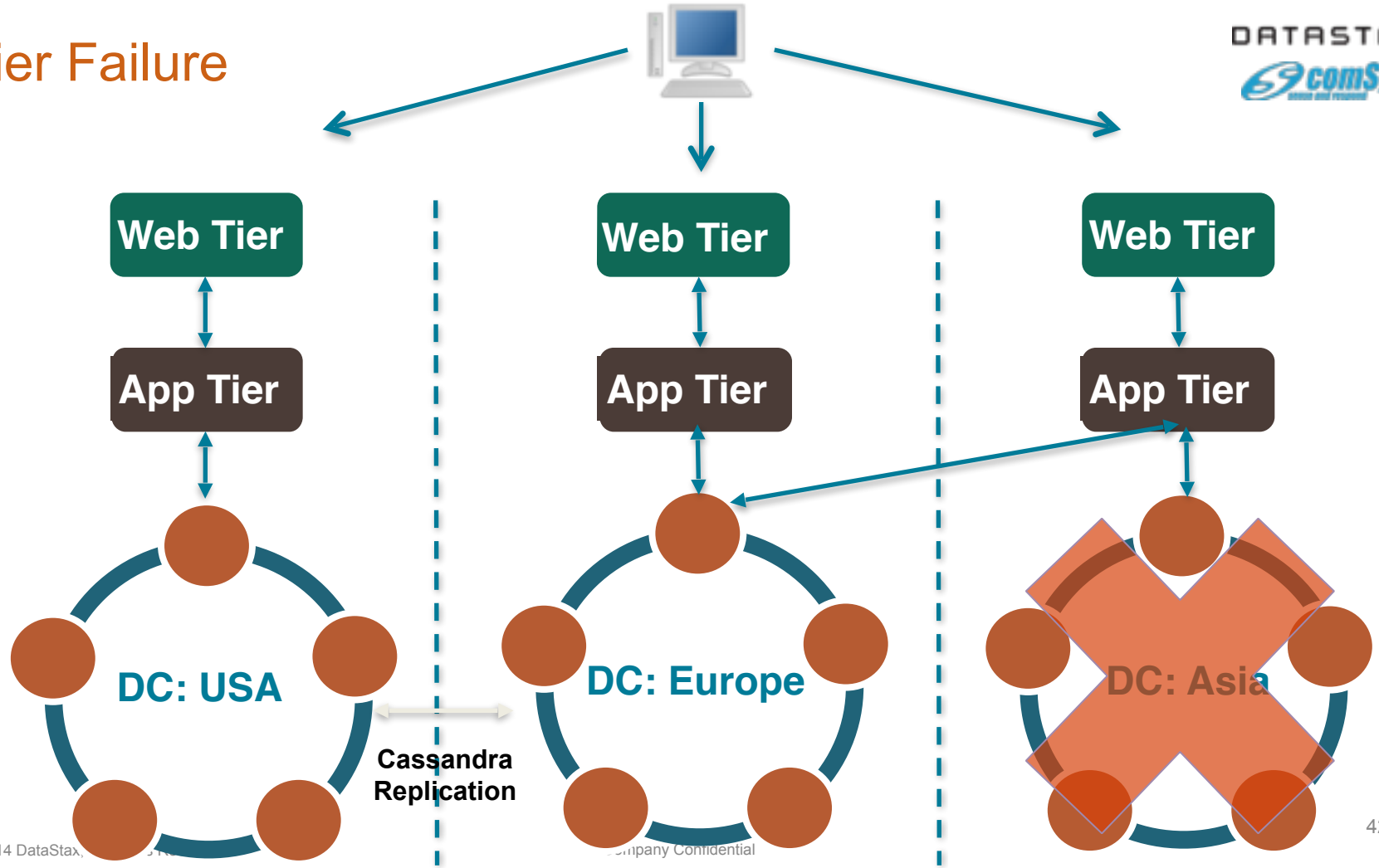
- Provides granular based control over who can add/ change/delete/read data
- Uses familiar GRANT/ REVOKE from relational systems
- No learning curve

- Ensures data cannot be captured/stolen in route to a server
- Data is safe both in flight from/to a database and on the database; complete coverage is ensured

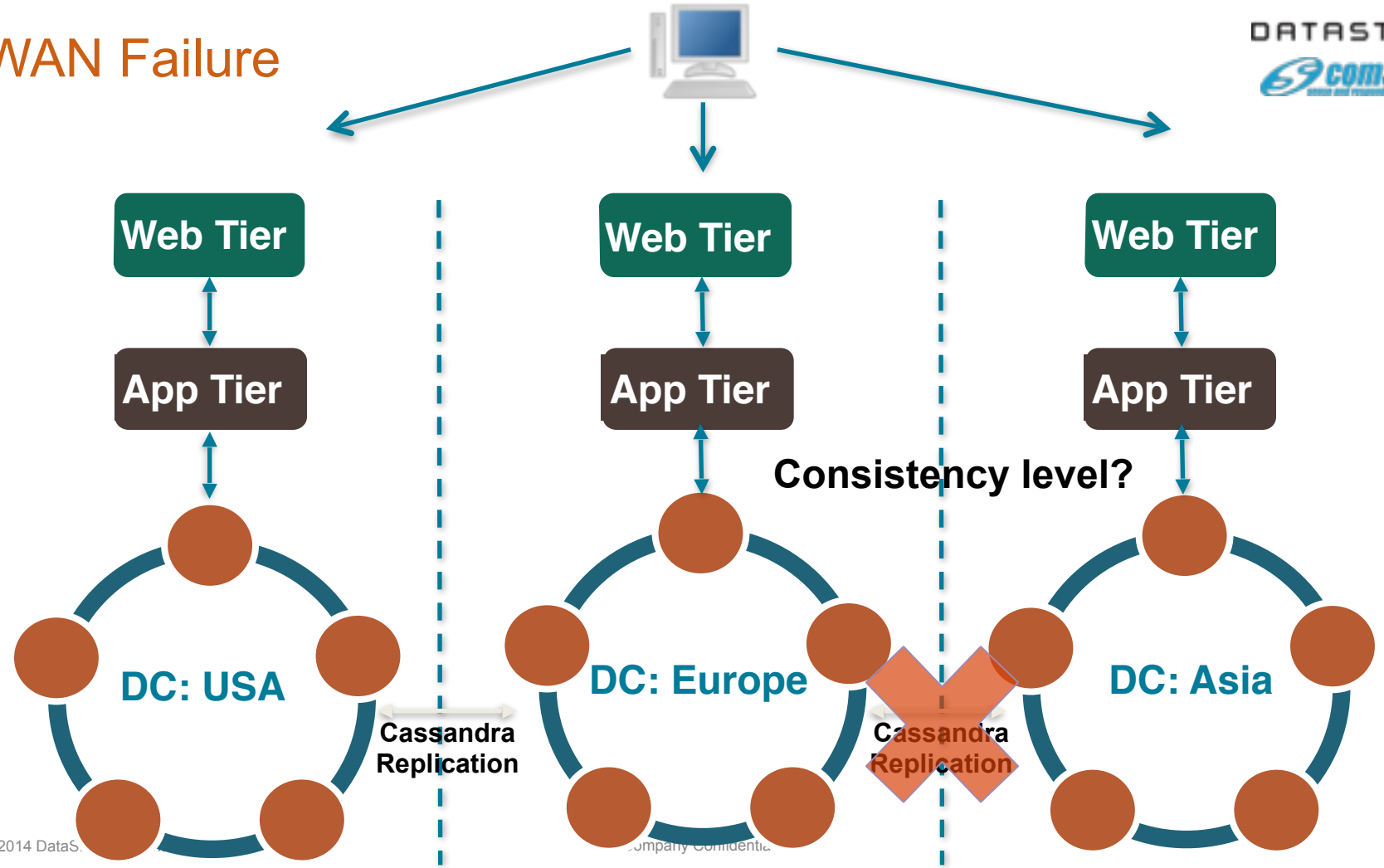
Uptime



Tier Failure



WAN Failure



How?

DATASTAX

comSysto

Data Modeling

Why Good Data Modeling is Important?

Cassandra is a highly available, highly scalable, & highly distributed database - with no single point of failure

To achieve this, **Cassandra is optimized for non-relational data models.**

- **Joins do not function well on distributed databases.**
- **Locking and transactions jam up distributed nodes**

So what do I do?

- Lots of different approaches to this…
- Basically, create multiple tables optimized for your queries.

Query Based Modeling

- The main point is:

DON'T BE AFRAID OF WRITES

DENORMALISE AND MODEL YOUR DATA AROUND YOUR QUERIES

For more on data modeling…

- **Data modeling video series by Patrick McFadin**
- Part 1: The Data Model is Dead, Long Live the Data Model
  http://www.youtube.com/watch?v=px6U2n74q3g
- Part 2: Become a Super Modeler
  http://www.youtube.com/watch?v=qphhxujn5Es
- Part 3: The World's Next Top Data Model
  http://www.youtube.com/watch?v=HdJIsOZVGwM

DATASTAX

comSysto

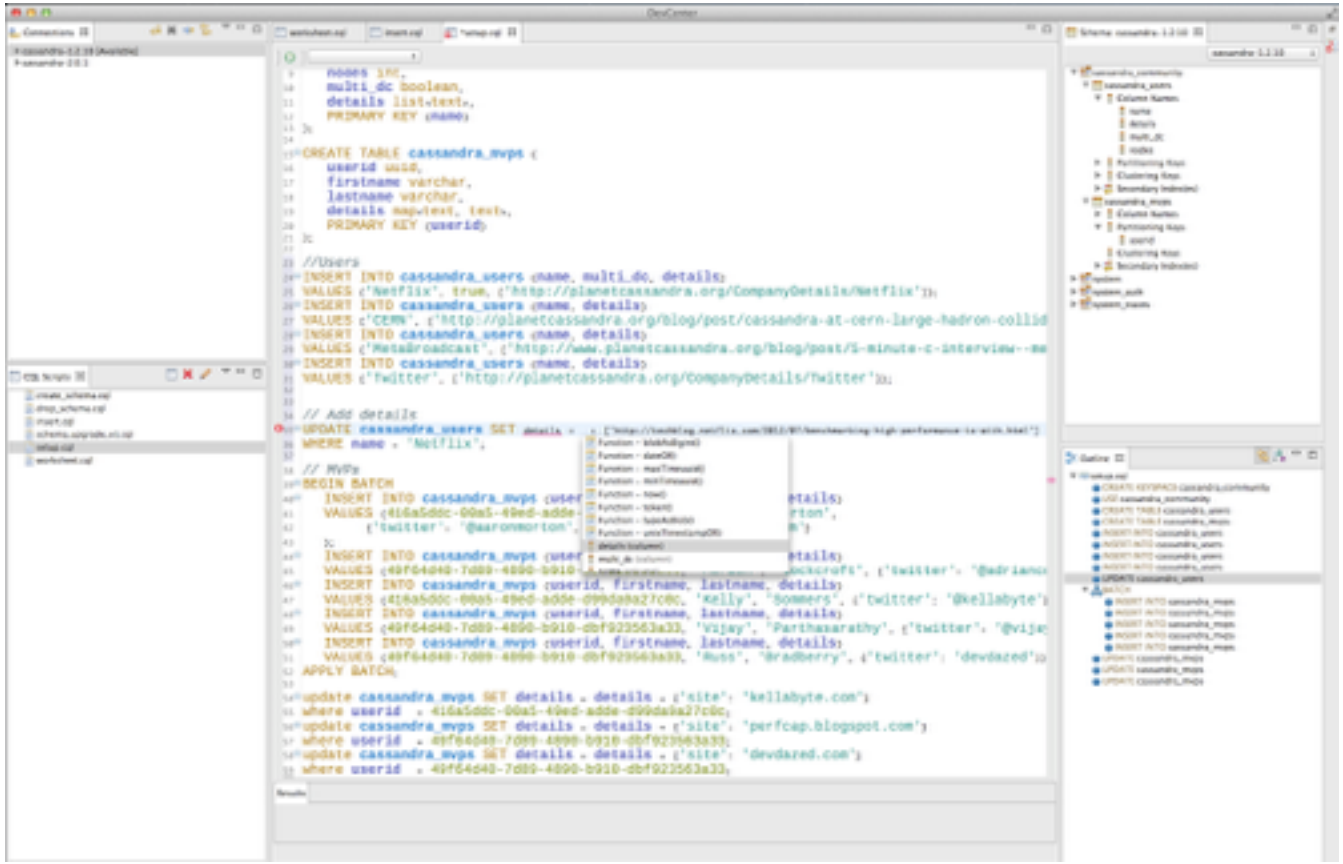
CQL

Cassandra Query Language

CQL

- **Cassandra Query Language**
- SQL–like language to query Cassandra
- Limited predicates. Attempts to prevent bad queries
  - but, you can still get into trouble!
- **Keyspace** – analogous to a schema.
  - Has various storage attributes.
  - The keyspace determines the RF (replication factor).
- **Table** – looks like a SQL Table.
  - A table must have a Primary Key.
  - We can fully qualify a table as <keyspace>.<table>

DevCenter



CQLSH

Command line interface comes with Cassandra

Launching on Linux

- `$ cqlsh [options] [host [port]]`

Launching on Windows

- `python cqlsh [options] [host [port]]`

Example

- `$ cqlsh`
- `$ cqlsh -u student -p cassandra 127.0.0.1 9160`

CQLSH

```
johnny@JPM-MacBook-Pro:/apps/dse/4.0/dse-4.0.2/bin$ ./cqlsh
Connected to Test Cluster at localhost:9160.
[cqlsh 4.1.1 | Cassandra 2.0.6.28 | CQL spec 3.1.1 | Thrift protocol 19.39.0]
Use HELP for help.
cqlsh> HELP

Documented shell commands:
==========================
CAPTURE      COPY  DESCRIBE  EXPAND  SHOW    TRACING
CONSISTENCY  DESC  EXIT      HELP    SOURCE

CQL help topics:
================
ALTER                          CREATE_TABLE_OPTIONS  SELECT
ALTER_ADD                      CREATE_TABLE_TYPES    SELECT_COLUMNFAMILY
ALTER_ALTER                    CREATE_USER           SELECT_EXPR
ALTER_DROP                     DELETE                SELECT_LIMIT
ALTER_RENAME                   DELETE_COLUMNS        SELECT_TABLE
ALTER_USER                     DELETE_USING          SELECT_WHERE
ALTER_WITH                     DELETE_WHERE          TEXT_OUTPUT
APPLY                          DROP                  TIMESTAMP_INPUT
ASCII_OUTPUT                   DROP_COLUMNFAMILY     TIMESTAMP_OUTPUT
BEGIN                          DROP_INDEX            TRUNCATE
BLOB_INPUT                     DROP_KEYSPACE         TYPES
BOOLEAN_INPUT                  DROP_TABLE            UPDATE
COMPOUND_PRIMARY_KEYS          DROP_USER             UPDATE_COUNTERS
CREATE                         GRANT                 UPDATE_SET
CREATE_COLUMNFAMILY            INSERT                UPDATE_USING
CREATE_COLUMNFAMILY_OPTIONS    LIST                  UPDATE_WHERE
CREATE_COLUMNFAMILY_TYPES      LIST_PERMISSIONS      USE
CREATE_INDEX                   LIST_USERS            UUID_INPUT
CREATE_KEYSPACE                PERMISSIONS
CREATE_TABLE                   REVOKE

cqlsh>
```

CQL Basics

- Usual statements
- CREATE / DROP / ALTER TABLE
- SELECT / INSERT / UPDATE
- **Plus many, many more!**

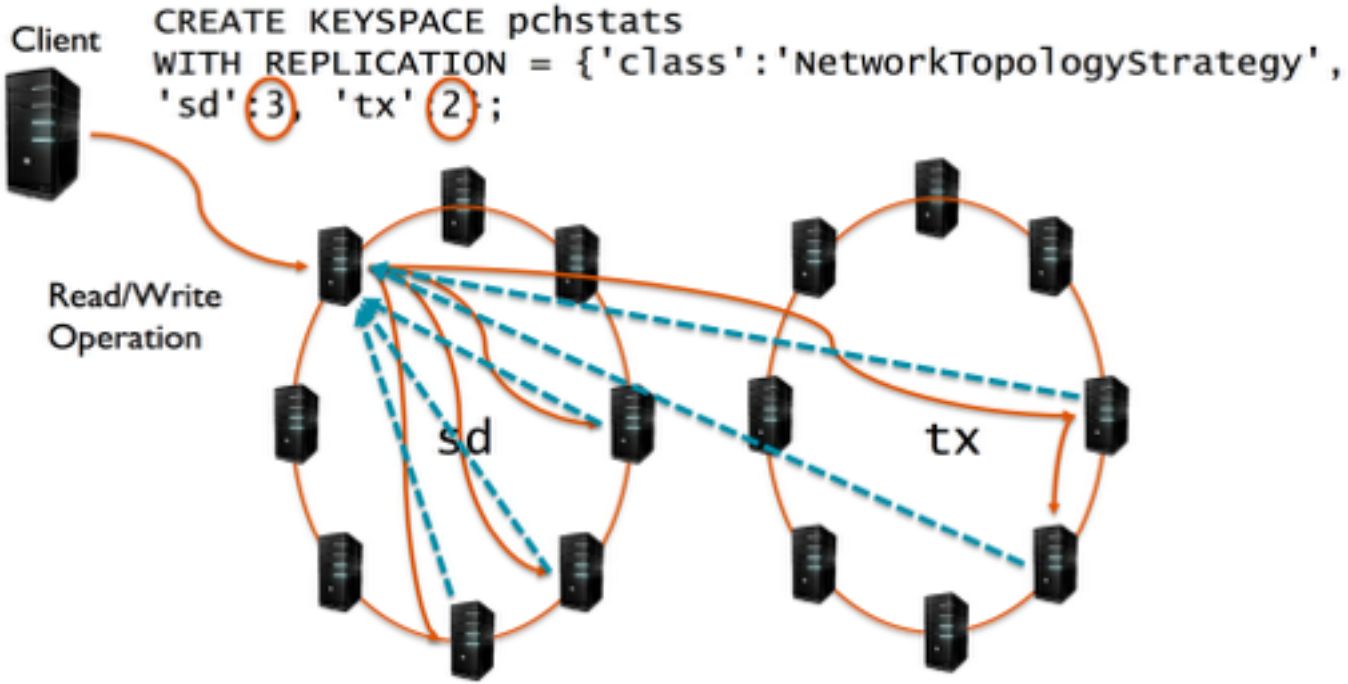
www.datastax.com/docs

What is a keyspace?

- **Keyspace or schema is a top-level namespace**
- All data objects (e.g., tables) must belong to some keyspace
- Defines how data is replicated on nodes
- Keyspace per application is a good idea

- **Replica placement strategy**
- **SimpleStrategy (prototyping)**
- **NetworkTopologyStrategy (production)**

Creating a keyspace - Multiple Data Centre



Use and Drop a keyspace

- To work with data objects (e.g., tables) in a keyspace:
- `USE pchstats;`
- To delete a keyspace and all internal data objects
- `DROP KEYSPACE pchstats;`

If you drop a keyspace or table, Cassandra will back it up!
You can disable this via auto_snapshot in the cassandra.yaml

Creating a table

```
CREATE TABLE cities (
    city_name   varchar,
    elevation   int,
    population  int,
    latitude    float,
    longitude   float,
    PRIMARY KEY (city_name)
);
```

- We can visualize it this way:

city_name	elevation	population	latitude	longitude
Springfield	978	60608	39°55'37"	83°48'15" W

- **Here, city_name is the partition key**
- In this example, the partition key = primary key

Compound Primary Key

The Primary Key

- The key uniquely identifies a row.
- A composite primary key consists of:
  - A **partition key**
  - One or more **clustering columns**
- e.g. `PRIMARY KEY (partition key, cluster columns, ...)`

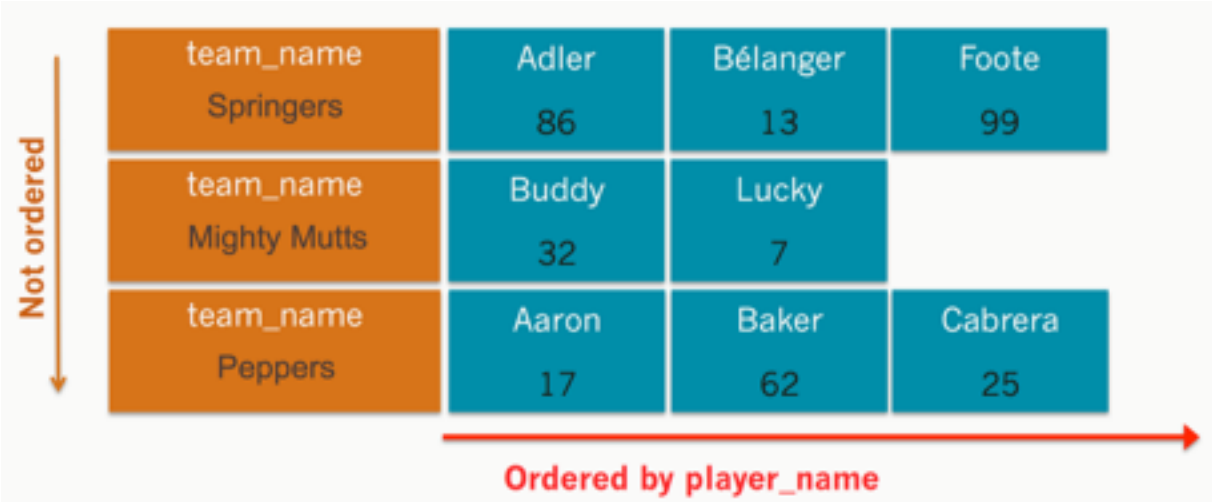
- The **partition key** determines on which node the partition resides
- Data is **ordered** in **cluster column** order within the partition

Compound Primary Key

```
CREATE TABLE sporty_league (
    team_name   varchar,
    player_name varchar,
    jersey      int,
    PRIMARY KEY (team_name, player_name)
);
```

Clustering Column

Partition Key



Simple Select

This is a bad query

```
SELECT * FROM sporty_league;
```

Partition Key

team_name	player_name	jersey
Peppers	Aaron	17
Peppers	Baker	62
Peppers	Cabrera	25
Springers	Adler	86
Springers	Bélanger	13
Springers	Foote	99
Mighty Mutts	Buddy	32
Mighty Mutts	Lucky	7

Clustering Column
ordered

- More that a few rows can be slow. (Limited to 10,000 rows by default)
- Use **LIMIT** keyword to choose fewer or more rows

Select on Partition Key and Cluster Columns

```
SELECT * FROM sporty_league WHERE team_name = 'Mighty Mutts';
```

```
 team_name     | player_name | jersey
---------------+-------------+--------
 Mighty Mutts |       Buddy |     32
 Mighty Mutts |       Lucky |      7
```

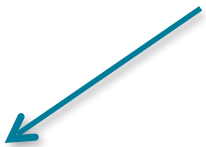
```
SELECT * FROM sporty_league WHERE team_name = 'Mighty Mutts'
                and player_name = 'Lucky';
```

```
 team_name     | player_name | jersey
---------------+-------------+--------
 Mighty Mutts |       Lucky |      7
```

Composite Partition Key

```
CREATE TABLE cities (
    city_name   varchar,
    state       varchar
    PRIMARY KEY ((city_name, state))
);
```

Partition Key



(city_name, state)	elevation	latitude	longitude	population
Springfield, OH	978	39.926	-83.804	60608
(city_name, state)	elevation	latitude	longitude	population
Springfield, IL	39.783	39.783	-89.650	116250

CQL Predicates

- On the **partition key**: = and IN
- On the **cluster columns**: <, <=, =, >=, >, IN

Performance Considerations

- The best queries are in a single partition.
  i.e. **WHERE partition key = <something>**
- Each new partition requires a new disk seek.
- Queries that span multiple partitions are **s-l-o-w**
- Queries that span multiple cluster columns are **fast**

Ordering

- Partition keys are **not** ordered, but the cluster columns are.
- However, **you can only order by a column if it's a cluster column**.
- Data will returned by default in the order of the clustering column.
- You can also use the ORDER BY keyword **– but only on the clustering column**!

```
SELECT * FROM sporty_league
WHERE team_name = 'Mighty Mutts'
ORDER BY player_name DESC;
```

Secondary Indexes

If we want to do a query on a column that is not part of your PK, you can create an index:

```
CREATE INDEX ON <table>(<column>);
```

Than you can do a select:

```
SELECT * FROM product WHERE type= 'PC';
```

Suitable for low cardinality data

Much more efficient to model your data around the query

Watch out for global indexes in Cassandra 3.0 (CASSANDRA-6477)

Insert/Update

```
INSERT INTO sporty_league (team_name, player_name, jersey)
VALUES ('Mighty Mutts','Felix',90);

UPDATE sporty_league SET jersey = 77
WHERE team_name = 'Mighty Mutts' AND player_name = 'Felix';
```

Inserts a row into a table

- Must specify columns to insert values into
- **Primary key columns are mandatory (identify the row)**
- Other columns do not have to have values and **Non-existent 'values' do not take up space**

Inserts are atomic

- All values of a row are inserted or none

Inserts are isolated

- Two inserts with the same values in primary key columns will not interfere – executed one after another

What is an upsert?

- **UPdate + inSERT**
- Both UPDATE and INSERT are write operations
- No reading before writing

- **Term “upsert” denotes the following behavior**
- INSERT updates or overwrites an existing row
  - When inserting a row in a table that already has another row with the same values in primary key columns
- UPDATE inserts a new row
  - When a to-be-updated row, identified by values in primary key columns, does not exist
- **Upserts are legal and do not result in error or warning messages**

How to avoid UPSERTS

Guarantee that your primary keys are unique from one another

Use an appropriate natural key based on your data

Use a surrogate key for partition key

Risks with natural keys

Depending on the type of natural key that is used, there may still be an increased risk of UPSERTs

Changing the datum used for a Natural Key requires a lot of overhead.

So why not use a sequence to generate a surrogate key?

You cant – **Cassandra doesn't provide sequences!**

What? No sequences!

- **Sequences are a handy feature in RDMBS for auto-creation of IDs for you data.**
  - Guaranteed unique
  - E.g. `INSERT INTO user (id, firstName, LastName) VALUES (seq.nextVal(), 'Ted', 'Codd')`

- **Cassandra has no sequences!**
  - Extremely difficult in a masterless distributed system
  - **Requires a lock (perf killer)**

- What to do?
  - **Use part of the data to create a unique key**
  - **Use a UUID or TIMEUUID**

UUID

- Universal Unique ID
- 128 bit number represented in character form e.g. 99051fe9-6a9c-46c2-b949-38ef78858dd0
- Easily generated on the client
- Version 1 has a timestamp component (TIMEUUID)
- Version 4 has no timestamp component
  - Faster to generate

TIMEUUID

TIMEUUID data type supports Version 1 UUIDs

Generated using time (60 bits), a clock sequence number (14 bits), and MAC address (48 bits)

- **CQL function ‘now()’ generates a new TIMEUUID**

Time can be extracted from TIMEUUID

- **CQL function dateOf() extracts the timestamp as a date**

TIMEUUID values in clustering columns or in column names are ordered based on time

- **DESC order on TIMEUUID lists most recent data first**

Time-to-Live (TTL)

DATASTAX
comSysto

TTL a row:

```
INSERT INTO users (id, first, last) VALUES ('abc123', 'catherine', 'cachart')
USING TTL 3600; // Expires data in one hour
```

TTL a column:

```
UPDATE users USING TTL 30 SET last = 'miller' WHERE id = 'abc123'
```

- TTL in seconds
- **Can also set default TTL at a table level**
- Expired columns/values automatically deleted
- With no TTL specified, columns/values never expire
- TTL is useful for automatic deletion
- Re-inserting the same row before it expires will overwrite TTL

Collection Data Type

```
CREATE TABLE collections_example (
    id int PRIMARY KEY,
    set_example set<text>,
    list_example list<text>,
    map_example map<int, text>
);
```

- CQL supports having columns that contain collections of data.
- The collection types include:
  - **Set, List and Map.**

- These data types are intended to support the type of one-to-many relationships that can be modeled in a relational DB e.g. a user has many email addresses.

- **Some performance considerations around collections.**
  - Often more efficient to denormalise further rather than use collections if intending to store lots of data.
  - **Favor sets over list**

You now have collection indexing in Cassandra 2.1

Lightweight Transactions (LWT)

Why?

- Solve a class of race conditions in Cassandra that you would otherwise need to install an external locking manager to solve.

Syntax:

```
INSERT INTO customer_account (customerID, customer_email)
VALUES ('Johnny', 'jmiller@datastax.com')
IF NOT EXISTS;

UPDATE  customer_account
SET customer_email='jmiller@datastax.com'
IF customer_email='jmiller@datastax.com';
```

Example Use Case:

- Registering a user

Not Will Ferrell @itsWillyFerrell · Apr 5

In about 20 years, the hardest thing our kids will have to do is find a username that isn't taken.

Race Condition

```
SELECT name
FROM users
WHERE username = 'johnny';

(0 rows)
```

```
INSERT INTO users
  (username, name, email,
   password, created_date)
VALUES ('johnny',
        'Johnny Miller',
        ['jmiller@datastax.com'],
        'ba27e03fd9...',
        '2011-06-20 13:50:00');
```

```
SELECT name
FROM users
WHERE username = 'johnny';
(0 rows)
```

```
INSERT INTO users
  (username, name, email,
   password, created_date)
VALUES ('johnny',
        'Johnny Miller',
        ['jmiller@datastax.com'],
        'ea24e13ad9...',
        '2011-06-20 13:50:01');
```

This one wins!

Lightweight Transactions (LWT)

```
INSERT INTO users
  (username, name, email,
   password, created_date)
VALUES ('johnny',
        'Johnny Miller',
        ['jmiller@datastax.com'],
        'ba27e03fd9...',
        '2011-06-20 13:50:00')
IF NOT EXISTS;
```

```
 [applied]
-----------
      True
```

```
INSERT INTO users
  (username, name, email,
   password, created_date)
VALUES ('johnny',
        'Johnny Miller',
        ['jmiller@datastax.com'],
        'ea24e13ad9...',
        '2011-06-20 13:50:01')
IF NOT EXISTS;
```

```
 [applied] | username | created_date    | name
-----------+----------+-----------------+----------------
     False | johnny   | 2011-06-20 ... | Johnny Miller
```

Lightweight Transactions (LWT)

Consequences of Lightweight Transactions

- **4 round trips vs. 1 for normal updates (uses Paxos algorithm)**
- Operations are done on a per-partition basis
- **Will be going across data centres to obtain consensus** (unless you use LOCAL_SERIAL consistency)
- Cassandra user will need read and write access i.e. you get back the row!

Great for 1% your app, but eventual consistency is still your friend!

Batch Statements

```
BEGIN BATCH
  INSERT INTO users (userID, password, name) VALUES ('user2', 'ch@ngem3b', 'second user')
  UPDATE users SET password = 'ps22dhds' WHERE userID = 'user2'
  INSERT INTO users (userID, password) VALUES ('user3', 'ch@ngem3c')
  DELETE name FROM users WHERE userID = 'user2'
APPLY BATCH;
```

BATCH statement combines multiple INSERT, UPDATE, and DELETE statements into a single logical operation

Atomic operation

If any statement in the batch succeeds, all will

No batch isolation

Other “transactions” can read and write data being affected by a partially executed batch

User Defined Types (UDT's)

```
CREATE TYPE address (
  street text,
  city text,
  zip_code int,
  phones set<text>
)

CREATE TABLE users (
  id uuid PRIMARY KEY,
  name text,
  addresses map<text, frozen <address>>
)

SELECT id, name, addresses.city, addresses.phones FROM users;

       id |     name | addresses.city |         addresses.phones
----------+----------+----------------+--------------------------
 63bf691f | johnny   |         London | {'0201234567', '0796622222'}
```

Lets store some JSON

```
{
  "productId": 2,
  "name": "Kitchen Table",
  "price": 249.99,
  "description" : "Rectangular table with oak finish",
  "dimensions": {
    "units": "inches",
    "length": 50.0,
    "width": 66.0,
    "height": 32
  },
  "categories": {
    {
      "category" : "Home Furnishings" {
        "catalogPage": 45,
        "url": "/home/furnishings"
      },
      {
      "category" : "Kitchen Furnishings" {
        "catalogPage": 108,
        "url": "/kitchen/furnishings"
    }
  }
}
```

Lets store some JSON

```
{
  "productId": 2,
  "name": "Kitchen Table",
  "price": 249.99,
  "description" : "Rectangular table with oak finish",
  "dimensions": {
    "units": "inches",
    "length": 50.0,
    "width": 66.0,
    "height": 32
  },
  "categories": {
    {
      "category" : "Home Furnishings" {
        "catalogPage": 45,
        "url": "/home/furnishings"
      },
      {
      "category" : "Kitchen Furnishings" {
        "catalogPage": 108,
        "url": "/kitchen/furnishings"
    }
  }
}
```

```
CREATE TYPE dimensions (
   units text,
   length float,
   width float,
   height float
)
```

Lets store some JSON

```
{
  "productId": 2,
  "name": "Kitchen Table",
  "price": 249.99,
  "description" : "Rectangular table with oak finish",
  "dimensions": {
    "units": "inches",
    "length": 50.0,
    "width": 66.0,
    "height": 32
  },
  "categories": {
    {
      "category" : "Home Furnishings" {
        "catalogPage": 45,
        "url": "/home/furnishings"
      },
      {
      "category" : "Kitchen Furnishings" {
        "catalogPage": 108,
        "url": "/kitchen/furnishings"
    }
  }
}
```

```
CREATE TYPE dimensions (
   units text,
   length float,
   width float,
   height float
)
```

```
CREATE TYPE category (
   catalogPage int,
   url text
);
```

Lets store some JSON

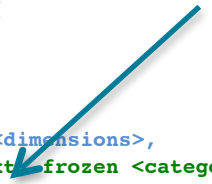
```
{
  "productId": 2,
  "name": "Kitchen Table",
  "price": 249.99,
  "description" : "Rectangular table with oak finish",
  "dimensions": {
    "units": "inches",
    "length": 50.0,
    "width": 66.0,
    "height": 32
  },
  "categories": {
    {
      "category" : "Home Furnishings" {
        "catalogPage": 45,
        "url": "/home/furnishings"
      },
      {
      "category" : "Kitchen Furnishings" {
        "catalogPage": 108,
        "url": "/kitchen/furnishings"
    }
  }
}
```

```
CREATE TYPE dimensions (
   units text,
   length float,
   width float,
   height float
)
```

```
CREATE TYPE category (
   catalogPage int,
   url text
);
```

```
CREATE TABLE product (
   productId int,
   name text,
   price float,
   description text,
   dimensions frozen <dimensions>,
   categories map <text, frozen <category>>,
   PRIMARY KEY (productId)
);
```

What's frozen mean?



Frozen

- This applies to User Defined Types, Tuples and nested collections.
- Frozen types **are serialized as a single value in Cassandra's storage engine**, whereas **non-frozen types are stored in a form that allows updates to individual subfields.**
- When using the frozen keyword, you cannot update parts of a user-defined type value. **The entire value must be overwritten.**
- Cassandra treats the value of a frozen type like a blob.
- **Cassandra 3.0 will support non-frozen**. As of 2.1, we only support frozen UDTs.
- Helps us avoid tech debt!

Lets store some JSON

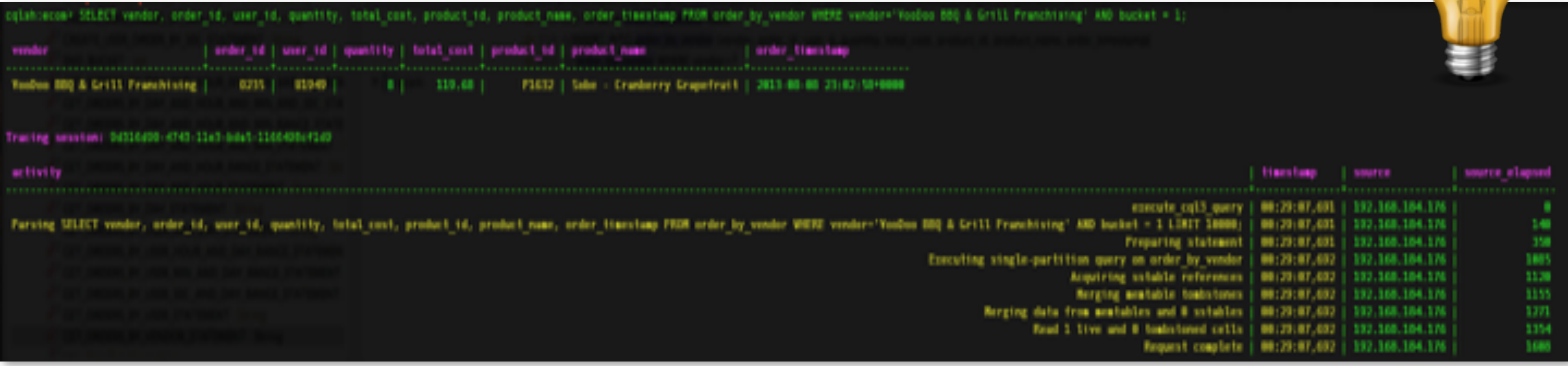
```
INSERT INTO product (productId, name, price,
description, dimensions, categories)
VALUES (2, 'Kitchen Table', 249.99, 'Rectangular
table with oak finish',
   {
     units: 'inches',
     length: 50.0,
     width: 66.0,
     height: 32
   },
   {
     'Home Furnishings': {
       catalogPage: 45,
       url: '/home/furnishings'
     },
     'Kitchen Furnishings': {
       catalogPage: 108,
       url: '/kitchen/furnishings'
     }

  }
);
```

dimensions frozen <dimensions>

categories map <text, frozen <category>>

Query Tracing

- You can turn on tracing on or off for queries with **the TRACING ON | OFF** command.
- Helps you understand what Cassandra is doing and identify any performance problems.



Authentication and Authorisation

CQL supports creating users and granting them access to tables etc..

You need to enable authentication in the cassandra.yaml config file.

You can create, alter, drop and list users

You can then GRANT permissions to users accordingly – ALTER, AUTHORIZE, DROP, MODIFY, SELECT.

Plus lots, lots more you can do with CQL ….

- Have a look at our documentation:
  http://www.datastax.com/documentation

- Technical Blog
  http://www.datastax.com/dev/blog

DataStax Java Driver

How to get it?

- https://github.com/datastax/java-driver

```
<dependency>
  <groupId>com.datastax.cassandra</groupId>
  <artifactId>cassandra-driver-core</artifactId>
  <version>2.1.0</version>
</dependency>
```

- The Java client driver 2.1 (branch 2.1) is **compatible** with Apache Cassandra 1.2, 2.0 and 2.1.
- If you try to use a feature that's not in the version of Cassandra you are connecting to, you will get an **UnsupportedFeatureException**

Not Thrift

- Traditionally, Cassandra clients (Hector, Astynax[1] etc..) were developed using Thrift
- With Cassandra 1.2 (Jan 2013) and the introduction of CQL3 and **the CQL native protocol** a new easier way of using Cassandra was introduced.
- Why?
  - Easier to develop and model
  - Best practices for building modern distributed applications
  - Integrated tools and experience
  - Enable Cassandra to evolve easier and support new features

[1]Astynax is being updated to include the native driver: https://github.com/Netflix/astyanax/wiki/Astyanax-over-Java-Driver

Thrift post-Cassandra 2.1

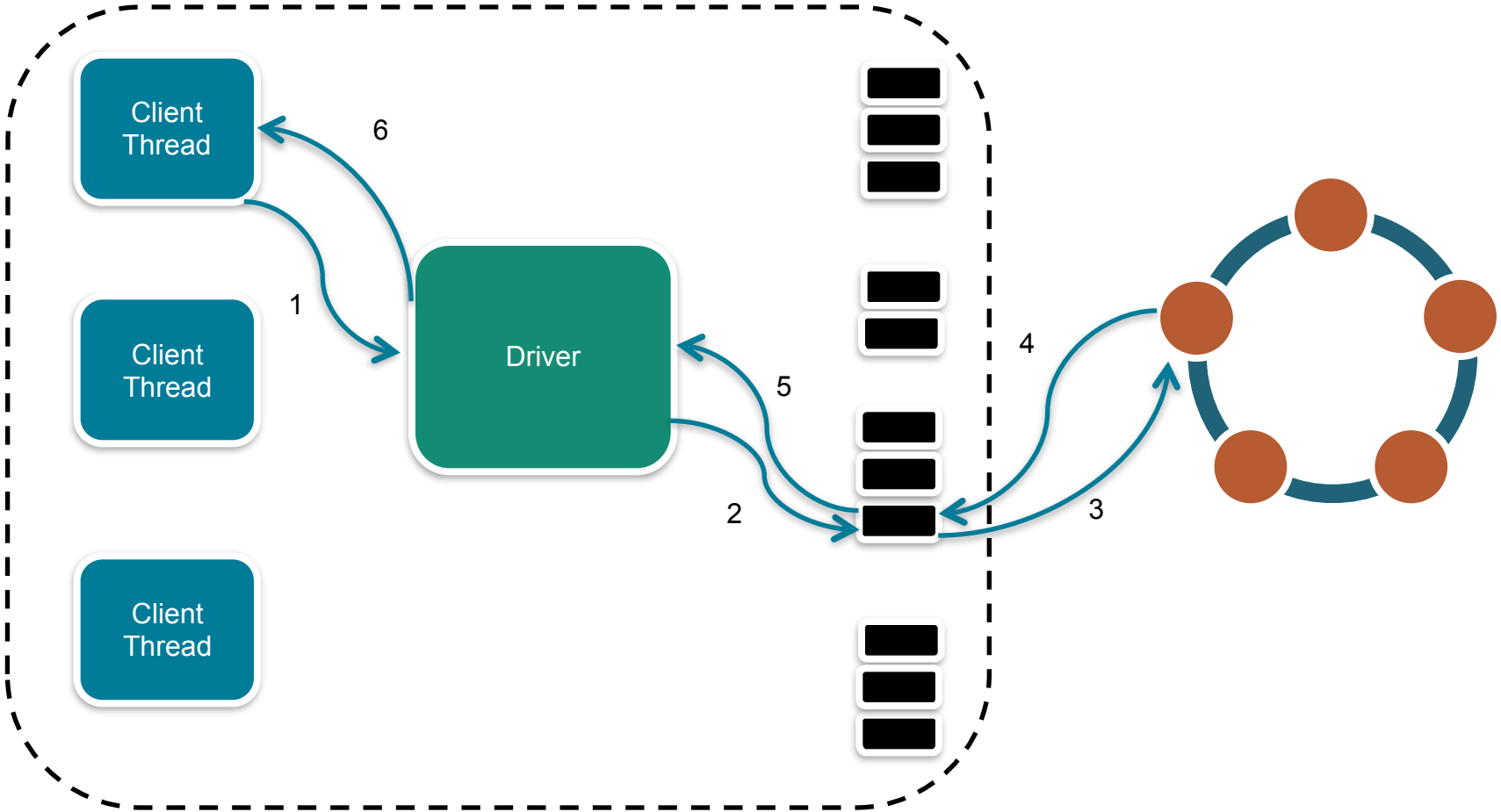
There is no more work on Thrift post 2.1.0

- http://bit.ly/freezethrift

Will retain it for backwards compatibility, but no new features or changes to the Thrift API after 2.1.0

- *"**CQL3 is almost two years old** now and has proved to be the **better API** that Cassandra needed. CQL drivers have caught up with and passed the Thrift ones in terms of **features, performance, and usability**. CQL is **easier to learn and more productive** than Thrift."*
- - Jonathan Ellis, Apache Chair, Cassandra

Asynchronous Architecture



Connect and Write

```
Cluster cluster = Cluster.builder()
                  .addContactPoints("10.158.02.40", "10.158.02.44")
                  .build();

Session session = cluster.connect("akeyspace");

session.execute(
        "INSERT INTO user (username, password) ”
        + "VALUES(‘johnny’, ‘password1234’)"
);
```

Note: Clusters and Sessions should be long-lived and re-used.

Read from a table

```
ResultSet rs = session.execute("SELECT * FROM user");

List<Row> rows = rs.all();

for (Row row : rows) {
        String userName = row.getString("username");
        String password = row.getString("password");
}
```

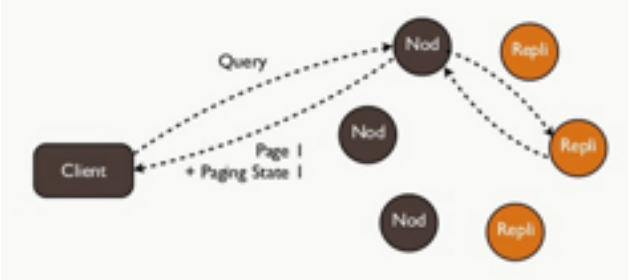
Paging

This is great!

Historically difficult to get huge result sets out of Cassandra. It has generally been necessary to explicitly enumerate your row keys in reasonably small batches (1000 rows or so per batch would be common).

This feature now allows you to get huge result sets (including “select * from table), and have the server automatically page the results, while the client is just able to trivially iterate over the entire result set.

Makes data exploration much easier.



Asynchronous Requests

```
ResultSetFuture future = session.executeAsync("SELECT * FROM user");

for (Row row : future.get()) {
        String userName = row.getString("username");
        String password = row.getString("password");
}
```

Note: The future returned implements Guava's ListenableFuture interface. This means you can use all Guava's Futures[1] methods!

[1]http://docs.guava-libraries.googlecode.com/git/javadoc/com/google/common/util/concurrent/Futures.html

Read with Callbacks

```
final ResultSetFuture future = session.executeAsync("SELECT * FROM user");
Futures.addCallback(future, new FutureCallback<ResultSet>() {
            @Override
            public void onFailure(Throwable t) {
                log.error("Problem encountered“, t);
                 //Do something...
            }

            @Override
            public void onSuccess(ResultSet r) {
                log.debug("Success!!);
                 //Do something...
            }

        }
);
```

Parallelize Calls

```
int queryCount = 99;

List<ResultSetFuture> futures = new ArrayList<ResultSetFuture>();

for (int i=0; i<queryCount; i++) {
        futures.add(
                session.executeAsync("SELECT * FROM user "
                        +"WHERE username = '"+i+"'"));
}

for(ResultSetFuture future : futures) {
        for (Row row : future.getUninterruptibly()) {
                //do something
        }
}
```

Tip

- If you need to do a lot of work, **it's often better to make many small queries concurrently than to make one big query.**
  - **executeAsync and Futures – makes this really easy!**
  - Big queries can put a high load on one coordinator
  - Big queries can skew your 99th percentile latencies for other queries
  - If one small query fails you can easily retry, if a big query than you have to retry the whole thing

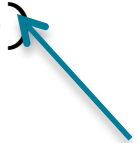
Query Builder

```
Query query = QueryBuilder
        .select()
        .all()
        .from("akeyspace", "user")
        .where(eq("username", "johnny"));

    query.setConsistencyLevel(ConsistencyLevel.ONE);

ResultSet rs = session.execute(query);
```

Multi Data Center Load Balancing

Local nodes are queried first, if none are available the request will be sent to a remote data center

```
Cluster cluster = Cluster.builder()
        .addContactPoints("10.158.02.40", "10.158.02.44")
        .withLoadBalancingPolicy(
            new DCAwareRoundRobinPolicy("DC1"))
        .build();
```



Name of the local DC

Token Aware Load Balancing

Nodes that own a replica of the data being read or written by the query will be contacted first

```
Cluster cluster = Cluster.builder()
        .addContactPoints("10.158.02.40", "10.158.02.44")
        .withLoadBalancingPolicy(
            new TokenAwarePolicy(
                new DCAwareRoundRobinPolicy("DC1")))
        .build();
```

Reconnection Policies

Policy that decides how often the reconnection to a dead node is attempted.

```
Cluster cluster = Cluster.builder()
        .addContactPoints("10.158.02.40",
"10.158.02.44")   .withRetryPolicy(DowngradingConsistencyRetryPolicy.INSTANCE)
        .withReconnectionPolicy(new ConstantReconnectionPolicy(1000))
        .withLoadBalancingPolicy(new TokenAwarePolicy(new DCAwareRoundRobinPolicy("DC1")))
        .build();
```

- ConstantReconnectionPolicy
- **ExponentialReconnectionPolicy (Default)**

Query Tracing

Tracing is enabled on a per-query basis.

```
Query query = QueryBuilder
            .select()
            .all()
            .from("akeyspace", "user")
            .where(eq("username", "johnny"))
            .enableTracing();

ResultSet rs = session.execute(query);
ExecutionInfo executionInfo = rs.getExecutionInfo();
QueryTrace queryTrace = executionInfo.getQueryTrace();
```

Query Tracing

```
Connected to cluster: xerxes
Simplex keyspace and schema created.
Host (queried): /127.0.0.1
Host (tried): /127.0.0.1
Trace id: 96ac9400-a3a5-11e2-96a9-4db56cdc5fe7

activity                                  | timestamp    | source      | source_elapsed
------------------------------------------+--------------+-------------+---------------
                      Parsing statement | 12:17:16.736 | /127.0.0.1 |            28
                    Peparing statement | 12:17:16.736 | /127.0.0.1 |           199
     Determining replicas for mutation | 12:17:16.736 | /127.0.0.1 |           348
         Sending message to /127.0.0.3 | 12:17:16.736 | /127.0.0.1 |           788
         Sending message to /127.0.0.2 | 12:17:16.736 | /127.0.0.1 |           805
        Acquiring switchLock read lock | 12:17:16.736 | /127.0.0.1 |           828
                Appending to commitlog | 12:17:16.736 | /127.0.0.1 |           848
             Adding to songs memtable | 12:17:16.736 | /127.0.0.1 |           900
      Message received from /127.0.0.1 | 12:17:16.737 | /127.0.0.2 |            34
      Message received from /127.0.0.1 | 12:17:16.737 | /127.0.0.3 |            25
        Acquiring switchLock read lock | 12:17:16.737 | /127.0.0.2 |           672
        Acquiring switchLock read lock | 12:17:16.737 | /127.0.0.3 |           525
                Appending to commitlog | 12:17:16.737 | /127.0.0.2 |           692
                Appending to commitlog | 12:17:16.737 | /127.0.0.3 |           541
             Adding to songs memtable | 12:17:16.737 | /127.0.0.2 |           741
             Adding to songs memtable | 12:17:16.737 | /127.0.0.3 |           583
      Enqueuing response to /127.0.0.1 | 12:17:16.737 | /127.0.0.3 |           751
      Enqueuing response to /127.0.0.1 | 12:17:16.738 | /127.0.0.2 |           950
      Message received from /127.0.0.3 | 12:17:16.738 | /127.0.0.1 |           178
         Sending message to /127.0.0.1 | 12:17:16.738 | /127.0.0.2 |          1189
      Message received from /127.0.0.2 | 12:17:16.738 | /127.0.0.1 |           249
   Processing response from /127.0.0.3 | 12:17:16.738 | /127.0.0.1 |           345
   Processing response from /127.0.0.2 | 12:17:16.738 | /127.0.0.1 |           377
```

User Defined Types - Retrieval

```
USE ks;

CREATE TYPE address (
    street text,
    city text,
    zip int
);

CREATE TABLE user_profiles (
    email text PRIMARY KEY,
    address address
);
```

```
Row row = session.execute(
  "SELECT * FROM user_profiles").one();
UDTValue address =
row.getUDTValue("address");

// You get a map-like object with the
usual
// getters and setters (by name, by
index):

String street =
address.getString("street");
int zip       = address.getInt(2);
```

User Defined Types - Creation

```
USE ks;

CREATE TYPE address (
    street text,
    city text,
    zip int
);

CREATE TABLE user_profiles (
    email text PRIMARY KEY,
    address address
);
```

```
// Must first get hold of the datatype:
// From an existing value:
UserType addressType = address.getType();

// OR...

// From the cluster metadata:
UserType addressType =
    cluster.getMetadata()
           .getKeyspace("ks")
           .getUserType("address");
```

User Defined Types - Creation

```
USE ks;

CREATE TYPE address (
    street text,
    city text,
    zip int
);

CREATE TABLE user_profiles (
    email text PRIMARY KEY,
    address address
);
```

```
UDTValue address =
    addressType.newValue()
                .setString("street", "...")
                .setString("city",
"Washington")
                .setInt("zip", 20500);

// Now use it in a query like any other
type:
session.execute(
    "INSERT INTO user_profiles (email,
address) VALUES (?, ?)",
    "xyz@example.com", address);
```

Tuples - retrieval

```
USE ks;

CREATE TABLE
points_of_interest (
    id int PRIMARY KEY,
    name text,
    coordinates
        tuple<float, float>
);
```

```
Row row = session.execute(
    "SELECT * FROM
points_of_interest").one();
TupleValue coordinates =
    row.getTupleValue("coordinates");


float latitude  = coordinates.getFloat(0);
float longitude = coordinates.getFloat(1);
```

Tuples - creation

```
USE ks;

CREATE TABLE
points_of_interest (
    id int PRIMARY KEY,
    name text,
    coordinates
        tuple<float, float>
);
```

```
TupleType coordinatesType =
    TupleType.of(DataType.cfloat(),
                 DataType.cfloat());

TupleValue newCoordinates =
    coordinatesType.newValue(48.858222F,
                 2.2945F);
```

Simple Object Mapper

Query results ↔ Java objects

- Avoid boilerplate for common use cases (CRUD, etc.)
- Keep it **simple** and close to the bare metal
  - **Do not hide Cassandra from the developer**
  - Avoid "clever tricks" à la Hibernate

It comes as an additional Maven artifact:

```
<dependency>
    <groupId>com.datastax.cassandra</groupId>
    <artifactId>cassandra-driver-mapping</artifactId>
    <version>2.1.0</version>
</dependency>
```

Object Mapper Basic CRUD

```
USE ks;

CREATE TYPE address (
    street text,
    city text,
    zip int
);

CREATE TABLE user_profiles (
    email text PRIMARY KEY,
    address address
);
```

```
@UDT(keyspace = "ks", name = "address")
public class Address {
    private String street;
    private String city;
    private int zip;

    // getters and setters omitted...
}

@Table(keyspace = "ks", name =
"user_profiles")
public class UserProfile {
    @PartitionKey
    private String email;
    private Address address;

    // getters and setters omitted...
}
```

Object Mapper Basic CRUD

```
USE ks;

CREATE TYPE address (
    street text,
    city text,
    zip int
);

CREATE TABLE user_profiles (
    email text PRIMARY KEY,
    address address
);
```

```
MappingManager manager =
    new MappingManager(session);

Mapper mapper =
manager.mapper(UserProfile.class);

UserProfile myProfile =
    mapper.get("xyz@example.com");

ListenableFuture saveFuture =
    mapper.saveAsync(anotherProfile);

mapper.delete("xyz@example.com");
```

Accessor custom queries

```
USE ks;

CREATE TYPE address (
    street text,
    city text,
    zip int
);

CREATE TABLE user_profiles (
    email text PRIMARY KEY,
    address address
);
```

```
@Accessor
interface ProfileAccessor {
    @Query("SELECT * FROM user_profiles
LIMIT :max")
    Result firstN(@Param("max") int limit);
}

ProfileAccessor accessor =
  manager.createAccessor(ProfileAccessor.class);

Result profiles = accessor.firstN(10);

// Result is like ResultSet, but specialized for a
// mapped class:
for (UserProfile profile : profiles) {
    System.out.println(
        profile.getAddress().getZip()
    );
}
```

CASSANDRA SUMMIT2014 | EUROPE

London, U.K.
December 3 - 4

#CassandraSummit
Park Plaza Riverbank Hotel

Training Day | *December 3rd*

Beginner Track
- *Introduction to Cassandra*
- *Introduction to Spark, Shark, Scala and Cassandra*

Advanced Track
- *Data Modeling*
- *Performance Tuning*

Conference Day | *December 4th*

Cassandra Summit Europe 2014 will be the single largest gathering of Cassandra users in Europe. Learn how the world's most successful companies are transforming their businesses and growing faster than ever using Apache Cassandra.

http://bit.ly/cassandrasummit2014