C* Primer

Part 01: Cassandra and Modeling Basics

```
SELECT * FROM presenters WHERE name IN ('Christopher Reedijk', 'Gary Stewart');
         name                | title         | company | area | twitter
        ---------------------+---------------+---------+------+-----------
         Christopher Reedijk | Dev Engineer | ING     | NL   | @creedijk
         Gary Stewart        | Dev Engineer | ING     | NL   | @Gaz_GandA
```

ING Nederland (@ingnl)

This presentation is based on a 101 presentation by
Hayato Shimizu (@hayato_shimizu)

Short introduction

Gary Stewart

- Dev Engineer
- Love/hate relationship with C*
- @ING since: 01-01-2012
- aka Legacy

Christopher Reedijk

- Dev Engineer
- C* enthousiast
- @ING since: 01-01-2011
- aka Grumpy

3 Challenges

challenge 1

Improve availability without trading consistency

availability & consistency

Consequences of not being **available**

Problemen bij internetbankieren ING

AMSTERDAM - Internetbankieren kampt donderdagmiddag weer met een kleine storing. Een fractie van de klanten heeft problemen met inloggen of het...

Internetbankieren ING beperkt toegankelijk

AMSTERDAM - Klanten van ING kunnen woensdag soms geen gebruik maken van internetbankieren omdat de bank een maximum aan verkeer heeft ingesteld.

Internetbankieren ING onbereikbaar door onderhoud

AMSTERDAM - Vanwege uitgelopen onderhoud aan internetbankieren bij ING was de dienst dinsdagochtend niet bereikbaar. Inmiddels kunnen klanten...

source: nu.nl

Consequences of not being consistent

challenge 2

Aim to be easier scalable
Changes are happening at an increasing pace

easier scalable

Stop focusing on the **expected** load

yesterday

source: bradfrostweb.com

easier scalable

Start focusing on the **unexpected** load

today tomorrow

source: bradfrostweb.com

challenge 3

Adopt new ways of thinking
Become the top engineering company

new way of thinking

ING's **culture** is changing fast. Waterfall to Scrum to DevOps in less than **2 years**

new way of thinking

Aim to approach problems as green field to understand the essence

"Un-learn" principles that don't scale

Use cache correctly!

Reduce locking (transaction)

Apache Cassandra

Created by Avinash Lakshman and Prashant Malik at Facebook

Cassandra is

- A distributed database
- Highly scalable
- Fault tolerant
- Consistency is tunable
- High throughput
- Network topology aware
- Multi-DC Active-Active
- Written in Java
- A column family
- CQL3

Cassandra is not

- ACID compliant database
- A relational database
- ANSI SQL compliant
- A document database

DataStax and Cassandra

Founded in 2010 by Jonathan Ellis and Matt Pfeil

80% Apache Cassandra code contribution

Offer commercial support for DataStax Enterprise version of Cassandra

DataStax Enterprise integrates Search and Analytics

Head Quarter in San Francisco Bay area

EMEA office opened in March 2013

Native Protocol / DataStax Drivers

CQL3 Only

Fully asynchronous protocol, using Netty

Server notifications

Java, C#, Python, node.js and Ruby so far, more will be added

Many policies, including TokenAwarePolicy, DowngradeConsistencyRetryPolicy, etc...

Metrics (metrics.codahale.com) included

Maven:

```
<dependency>
    <groupid>com.datastax.cassandra</groupid>
    <artifactid>cassandra-driver-core</artifactid>
    <version>2.0.2</version>
</dependency>
```

Cassandra Internals

Tuneable Consistency

Configureable Consistency Level (CL) per read and write action

Writes

- Any
- One
- Two
- Three
- Quorum
- Local_Quorum
- Each_Quorum
- All

Reads

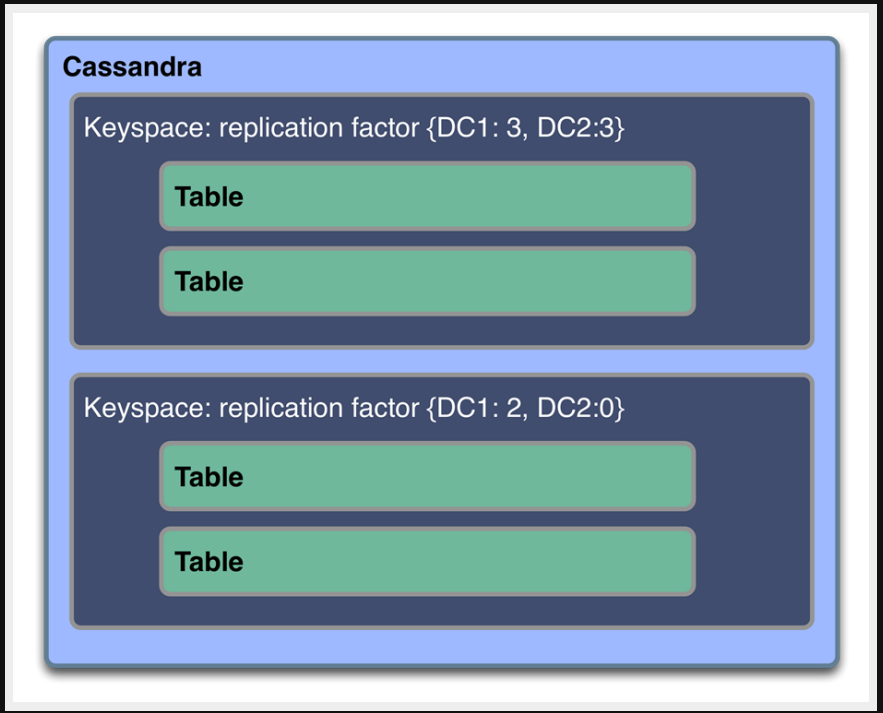
- One
- Two
- Three
- Quorum
- Local_Quorum
- Each_Quorum
- All

Logical Grouping

Keyspace: group of tables

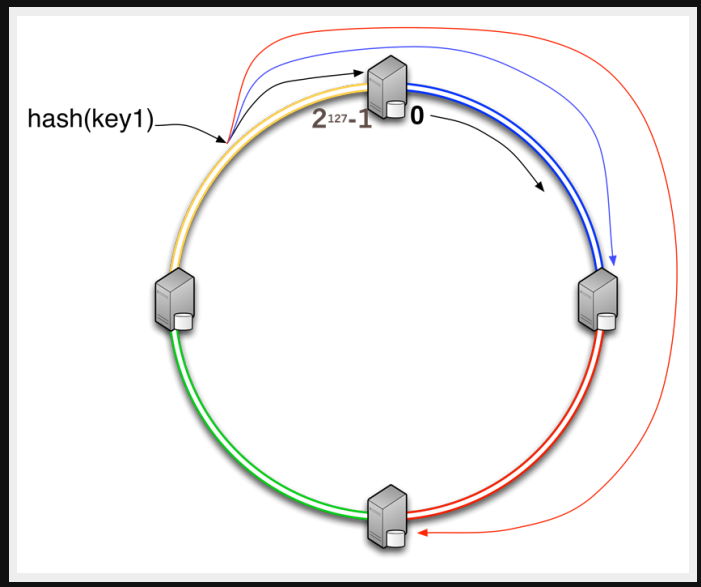
Table: group of data

Replication Factor:
Number of copies of data



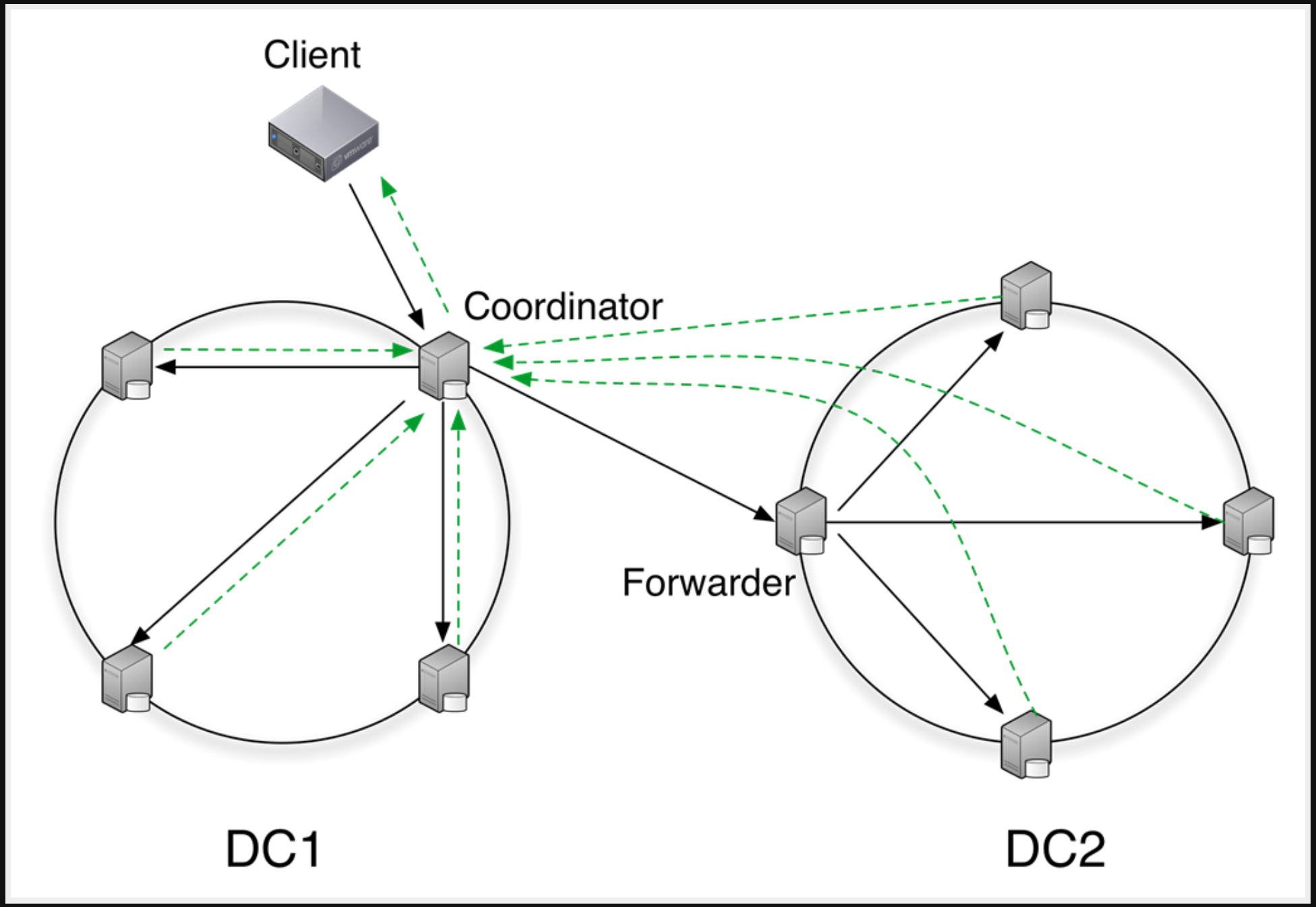
Cassandra Replication Strategy

- Token Range 0 -> 2^{127}-1 in Ring Formation
- Consistent Hashing Algorithm
- Replica nodes in clockwise
- Gossip protocol between nodes



Replication Factor (RF) = 3

Network Topology Awareness

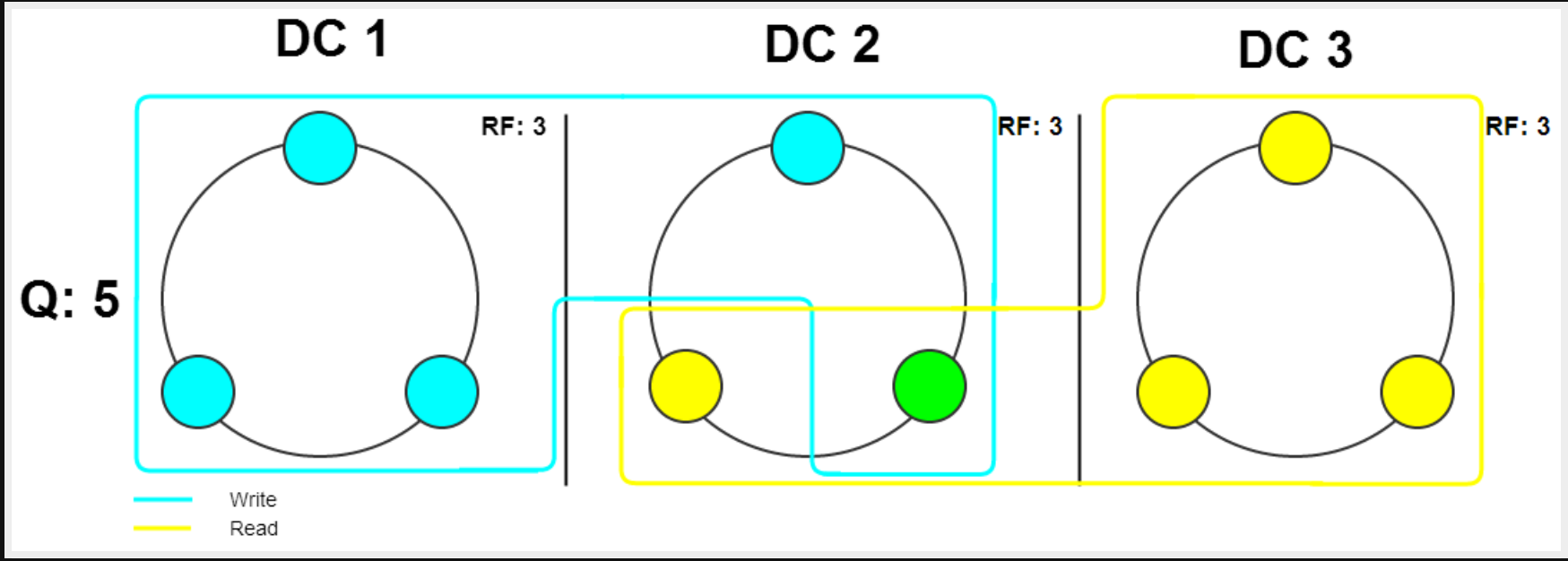

Client
Coordinator
Forwarder
DC1
DC2

Replication Factor and Data Consistency In a Distributed System

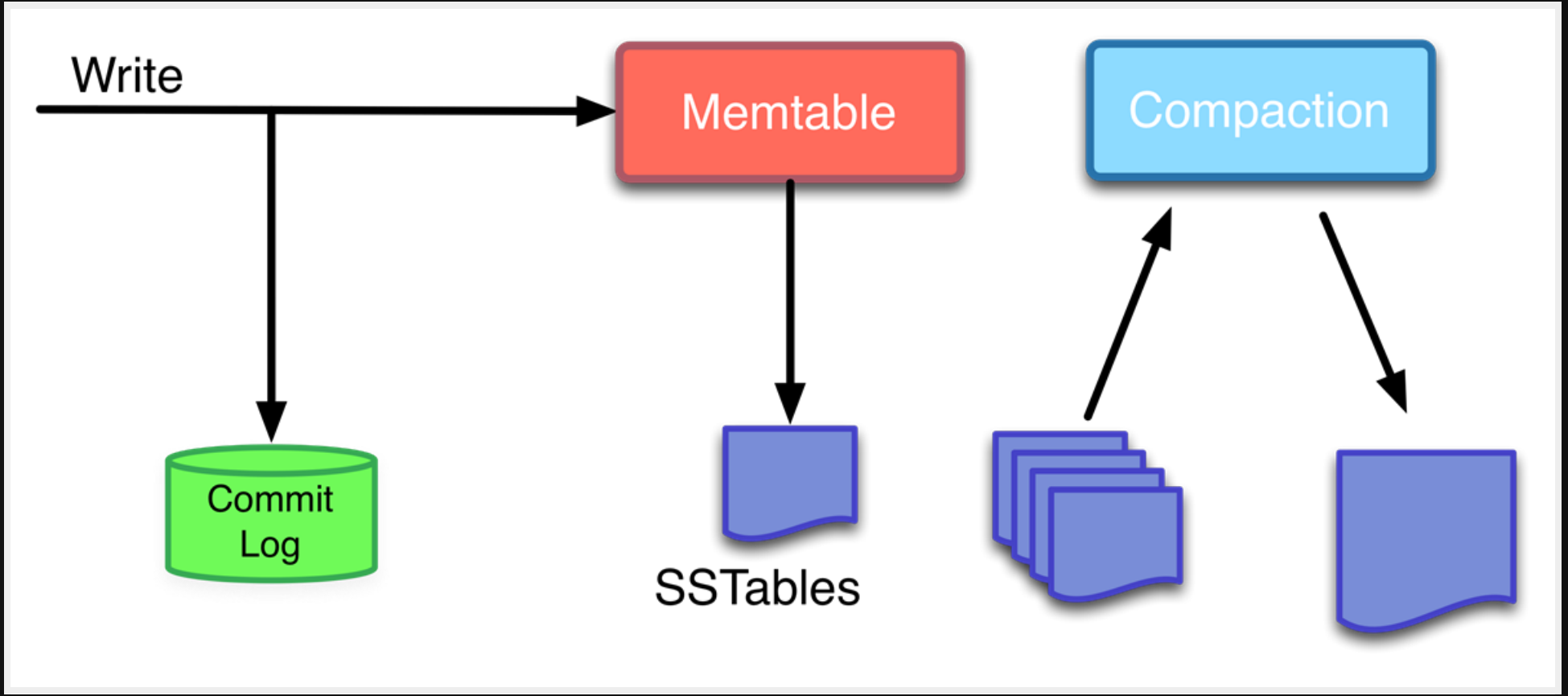
CAP – Consistency, Availability, Partition Tolerance

Consistency is achieved with W + R > RF

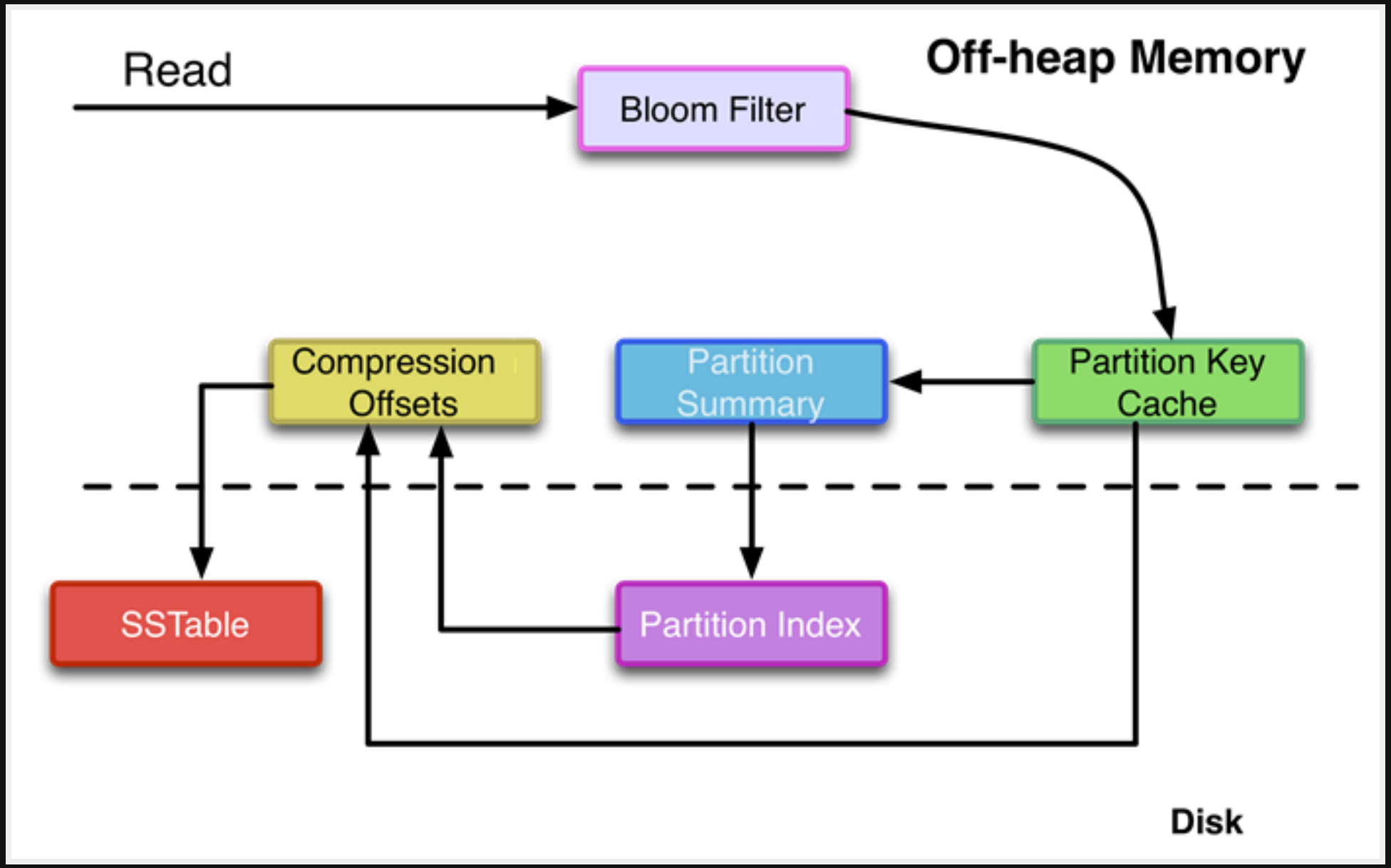
Quorum = (Integer) RF / 2 + 1



Write Path



Read Path


Read
Off-heap Memory
Bloom Filter
Compression Offsets
Partition Summary
Partition Key Cache
SSTable
Partition Index
Disk

CQL3

Why CQL3?

Excellent language for modeling data structures

Usability

Readability

Familiarity

Programming language agnostic

Apache Thrift, Hector, Astyanax days

```
Column col =
    new Column(ByteBuffer.wrap("product".getBytes())
));
col.setValue(ByteBuffer.wrap("phone".getBytes());
col.setTimestamp(System.currentTimeMillis());

ColumnOrSuperColumn c = new ColumnOrSuperColumn();
c.setColumn(col);

Mutation m = new Mutation();
m.setColumn_or_supercolumn(c);

List ms = new ArrayList();
ms.add(m);

Map> cf =
    new HashMap>();
cf.put("products", m);

Map>> records =
    new HashMap>>();
records.put(ByteBuffer.wrap("key".getBytes()), cf);
client.batch_mutate(records, consistencyLevel);
```

CQL3

```
INSERT INTO products (product_id, product)
     VALUES (“id12321”, “phone”);
```

Data Modelling

Primary Key

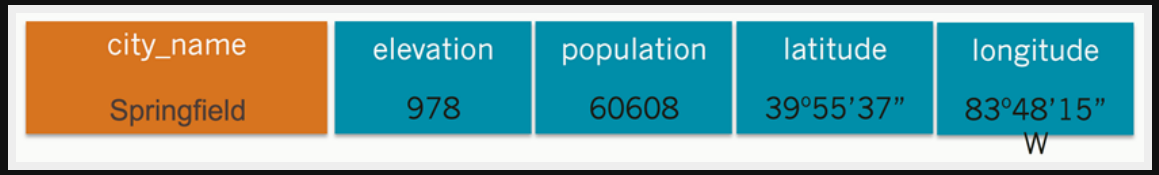
The Primary Key

- The key that uniquely identifies a row
- A primary key consists of:
  - A (composite) 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

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:



In this example:
partition key = primary key

Composite partition key

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

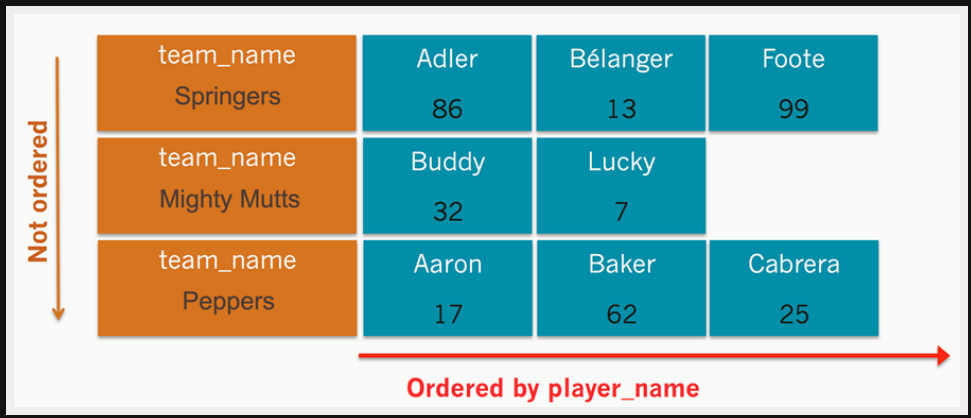
(city_name, state)	elevation	latitude	longitude	population
Springfield, OH	978	39.926	-83.804	60608

(city_name, state)	elevation	latitude	longitude	population
Springfield, IL	39.783	39.783	-89.650	116250

Each city gets it own partition!

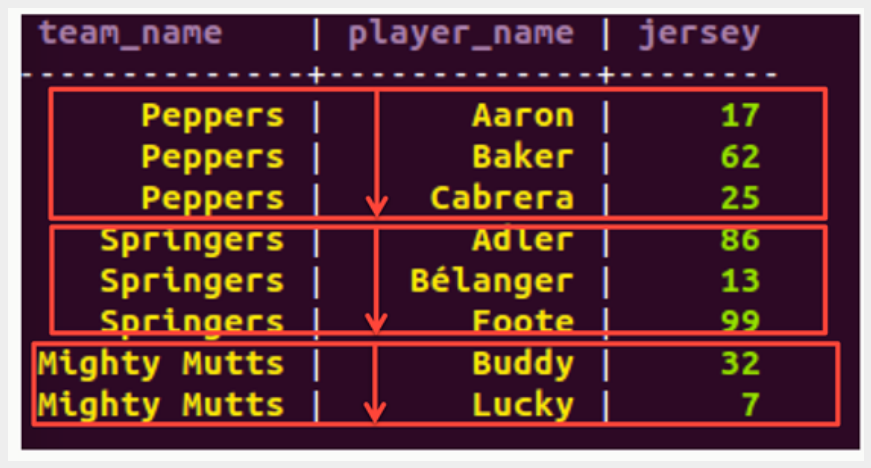
Clustering columns

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

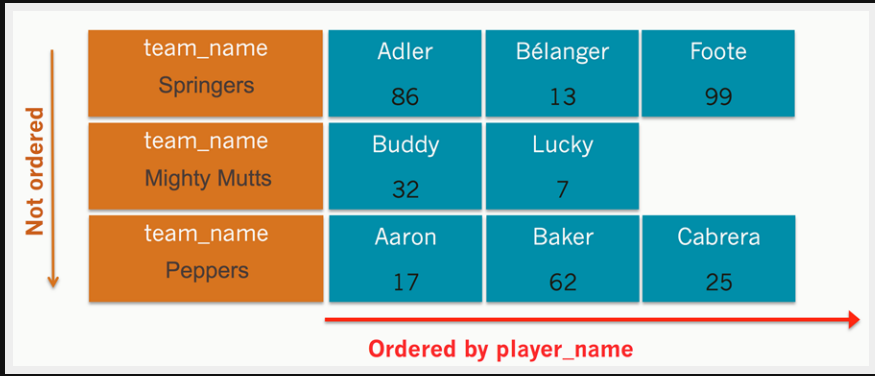


Simple Select

```
SELECT *
  FROM sporty_league;
```



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



"Table scan" - slow! (Limited to 10,000 rows by default)

Use LIMIT keyword to choose fewer or more rows

Simple 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
```

Insert/Update

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

```
UPDATE sporty_league
   SET jersey = 100
 WHERE team_name = 'Mighty Mutts'
   AND player_name = 'Felix';
```

Data Types

CQL Type	Constants	Description
ascii	strings	US-ASCII character string
bigint	integers	64-bit signed long
blob	blobs	Arbitrary bytes (no validation), expressed as hexadecimal
boolean	booleans	true or false
counter	integers	Distributed counter value (64-bit long)
decimal	integers, floats	Variable-precision decimal
double	integers	64-bit IEEE-754 floating point
float	integers, floats	32-bit IEEE-754 floating point
inet	strings	IP address string in IPv4 or IPv6 format [1]
int	integers	32-bit signed integer
list	n/a	A collection of one or more ordered elements
map	n/a	A collection of one or more timestamp, value pairs
set	n/a	A collection of one or more elements
text	strings	UTF-8 encoded string
timestamp	integers, strings	Date plus time, encoded as 8 bytes since epoch
uuid	uuids	Type 1 or type 4 UUID in standard UUID format
timeuuid	uuids	Type 1 UUID only (CQL 3)
varchar	strings	UTF-8 encoded string
varint	integers	Arbitrary-precision integer

Collections

CQL supports having columns that contain collections of data

```
The collection types include:
Set, List and Map

(64K max limitation)
```

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

These data types are intended to support the type of 1-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.
- Favour sets over list – lists not very performant

Performance considerations

The best queries are in a single partition.

i.e. WHERE partition key =

Queries that span multiple partitions are s-l-o-w

i.e. WHERE partition key IN (, , ...);

Queries that span multiple cluster columns are fast

i.e. WHERE partition key = and clustering key =

(Clustering) Order By

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

```
CREATE TABLE sporty_league (
   team_name   varchar,
   player_name varchar,
   jersey      int,
   PRIMARY KEY
         (team_name, player_name)
) WITH CLUSTERING ORDER BY
         (player_name DESC);
```

- Partition keys are not ordered!
- You can only order by a clustering column
- Data will ordered ASCending by default
- You can also use the ORDER BY keyword, but only on the clustering column(s), which don't have CLUSTERING ORDER BY defined!

De-normalize

Don't do this

```
CREATE TABLE users (
        id uuid PRIMARY KEY,
        username text,
        email text,
        street_address text,
        town text,
);

CREATE INDEX user_street ON
    users (street_address);
CREATE INDEX user_town ON
    users (town);

SELECT *
  FROM users
 WHERE town = 'Stockholm'
   AND street_address = 'Sankt Eriksgatan'
 ALLOW FILTERING;
```

Do this instead

```
CREATE TABLE users (
        id uuid PRIMARY KEY,
        username text,
        email text,
        street_address text,
        town text,
);




CREATE TABLE user_street (
        town text,
        street text,
        user_id uuid,
        PRIMARY KEY (
           (town, street),user_id)
);
```

Time to Live (TTL)

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

Time Series

```
CREATE TABLE sensor_data (
  sensor_id int,
  date varchar,
  time timeuuid,
  value int,
  PRIMARY KEY( (sensor_id, date), time ) )
WITH CLUSTERING ORDER BY (time DESC);
```

```
INSERT INTO sensor_data
       (sensor_id, date, time, value)
VALUES (1, '20130908',
       1883-11e3-8ffd-0800200c9a66,432)
 USING TTL 604800;
```

```
SELECT time, value, TTL (value)
  FROM sensor_data
 WHERE sensor_id = 1 AND date = '20130908'
   AND time > maxTimeuuid('2013-09-08 10:05+0000')
   AND time < minTimeuuid('2013-09-08 15:00+0000');
```

```
time                         | value | ttl(value)
-----------------------------+-------+------------
1883-11e3-8ffd-0800200c9a66 |   432 |     604257
1887-11e3-8ffd-0800200c9a66 |   433 |     604452
```

Lightweight Transactions

Introduced in Cassandra 2.0

Example:

```
INSERT INTO customer_account (customerID, customer_email)
VALUES ('LauraS', 'lauras@gmail.com')
    IF NOT EXISTS;

UPDATE customer_account
   SET customer_email='laurass@gmail.com'
    IF customer_email='lauras@gmail.com';
```

Great for 1% of your application – but not recommended to be used too much!

Tracing

You can turn on tracing on or off for queries with the TRACING ON | OFF command

This can help you understand what Cassandra is doing and identify any performance problems

```
cqlsh:ecom> SELECT vendor, order_id, user_id, quantity, total_cost, product_id, product_name, order_timestamp FROM order_by_vendor WHERE vendor='VooDoo BBQ & Grill Franchising' AND bucket = 1;

 vendor                         | order_id | user_id | quantity | total_cost | product_id | product_name                   | order_timestamp
--------------------------------+----------+---------+----------+------------+------------+--------------------------------+--------------------------
 VooDoo BBQ & Grill Franchising |     0235 |   U1949 |        8 |     119.68 |      P1632 | Sobe - Cranberry Grapefruit | 2013-08-08 23:02:58+0000

Tracing session: 9d316d90-4743-11e3-bda5-1166498cf1d9

 activity                                                                                                                                                                                 | timestamp    | source          | source_elapsed
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+--------------+-----------------+----------------
                                                                                                                                                                      execute_cql3_query | 00:29:07,691 | 192.168.184.176 |              0
 Parsing SELECT vendor, order_id, user_id, quantity, total_cost, product_id, product_name, order_timestamp FROM order_by_vendor WHERE vendor='VooDoo BBQ & Grill Franchising' AND bucket = 1 LIMIT 10000; | 00:29:07,691 | 192.168.184.176 |            140
                                                                                                                                                                      Preparing statement | 00:29:07,691 | 192.168.184.176 |            350
                                                                                                                                  Executing single-partition query on order_by_vendor | 00:29:07,692 | 192.168.184.176 |           1085
                                                                                                                                                   Acquiring sstable references | 00:29:07,692 | 192.168.184.176 |           1120
                                                                                                                                                    Merging memtable tombstones | 00:29:07,692 | 192.168.184.176 |           1155
                                                                                                                                     Merging data from memtables and 0 sstables | 00:29:07,692 | 192.168.184.176 |           1271
                                                                                                                                         Read 1 live and 0 tombstoned cells | 00:29:07,692 | 192.168.184.176 |           1354
                                                                                                                                                               Request complete | 00:29:07,692 | 192.168.184.176 |           1608
```

http://www.datastax.com/dev/blog/tracing-in-cassandra-1-2

all challenges appear again...

Improve availability

Aim to be scalable

Adopt new ways of thinking

3 Facts

fact 1

Availability is easier

availability is easier, because

Masterless architecture

No outages in contrast to master-slave architecture

This does make C* chatty

availability is easier, because

Replication of data

Replication factor is configurable per keyspace

Increasing nodes does not mean higher availability

Increasing the replication factor does

fact 2

Performance is easier

performance is easier, and

Solve your reads with your writes

De-normalization is acceptable for solving reads

performance is easier, but

Know your partitions

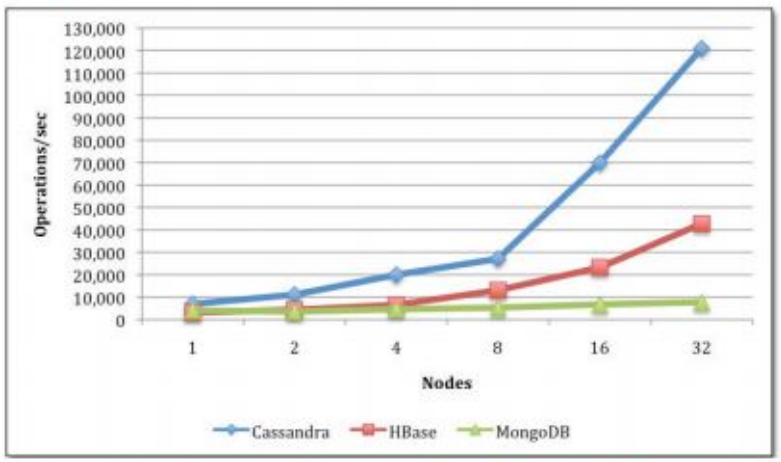
Large partitions will hurt!

Partition Key	Clustering Key	Col A	Col B
partitionKey_1	1	foo	...
	2	bar	...
partitionKey_2	1	foo	...
	2	bar	...
	3	baz	...
	...	...	...
	1000000	qux	...

performance is easier, because

Linear scalable

Adding nodes will increase your throughput through sharding



Shards	Cassandra	HBase	MongoDB	%/HBase	%/MongoDB
1	6955.33	3126.29	4101.73	122.48%	69.57%
2	11112.07	4379.17	3572.88	153.75%	211.01%
4	19965.86	6528.82	4630.18	205.81%	331.21%
8	27131.40	13067.41	5307.89	107.63%	411.15%
16	69922.85	23250.44	6767.80	200.74%	933.17%
32	120918.17	42761.57	7777.36	182.77%	1454.75%

For the load process, MongoDB was not able to scale effectively to 32 nodes and produced errors until the thread count was reduced to 20.

source: Datastax

fact 3

Consistency is harder

consistency is harder, and

Application developer has much more responsibility

Focus on the data model and the flow of data is key

noSQL versus SQL

consistency is harder, because

CAP theorem

Consistency (all nodes see the same data at the same time)

Availability (a guarantee that every request receives a response about whether it was successful or failed)

Partition tolerance (the system continues to operate despite arbitrary message loss or failure of part of the system)

consistency is harder, because

Transactions and queuing are anti-patterns

C* v2.0 reduces the complexity by using lightweight transactions (paxos)

all facts appear...

Availability is easier | challenge 1: availability & consistency

Performance is easier | challenge 2 easier scalable

Consistency is harder | challenge * availability & consistency, easier scalable, new way of thinking

all challenges covered...

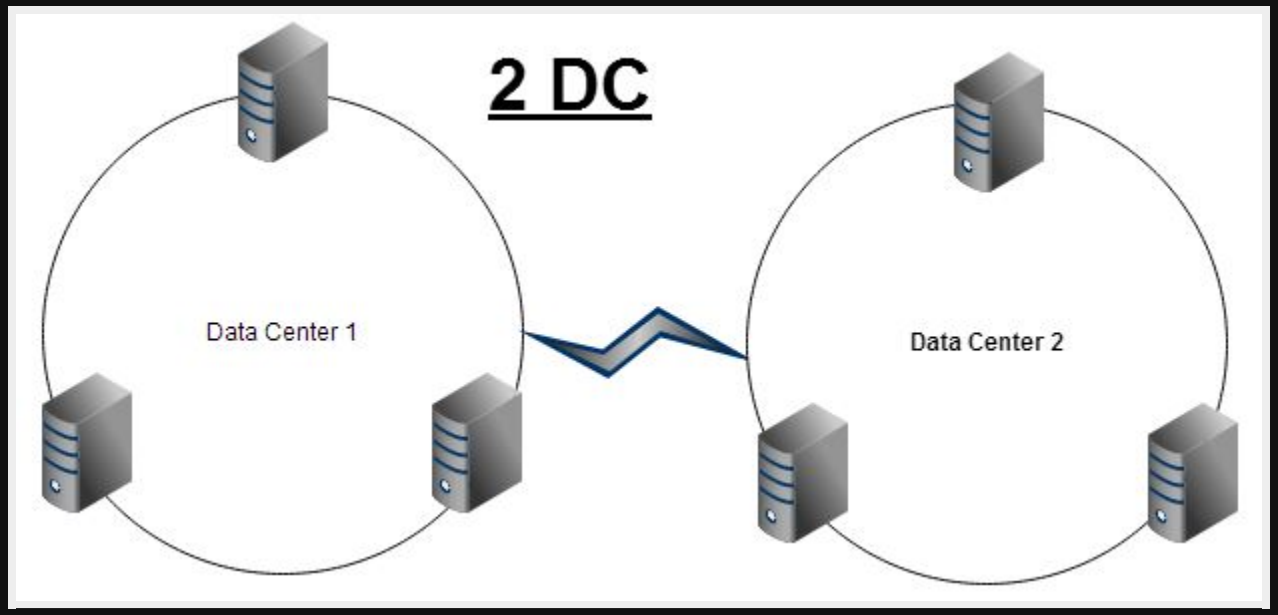
C*@ING

project	use case	status
Betelgeuze	Cache of Customer Data (KRO)	September 2014 in PRD
OI: Financial Fit	SOR for soft data	September 2014 in PRD
RTPE	New payments engine for on-us payments	February 2015 in PRD
Faster Than Light	Cache of Customer Data (MDM)	PoC
Ideal, Sales Support, NGINX, ...	Session Management	In DEV
Ideal	Order Management	Product Backlog
Credit Cards	Cache external data	Awaiting budget
Availability Dashboard	Time series	In DEV
Realtime Account Forecasting	Cache of SAM data	PoC

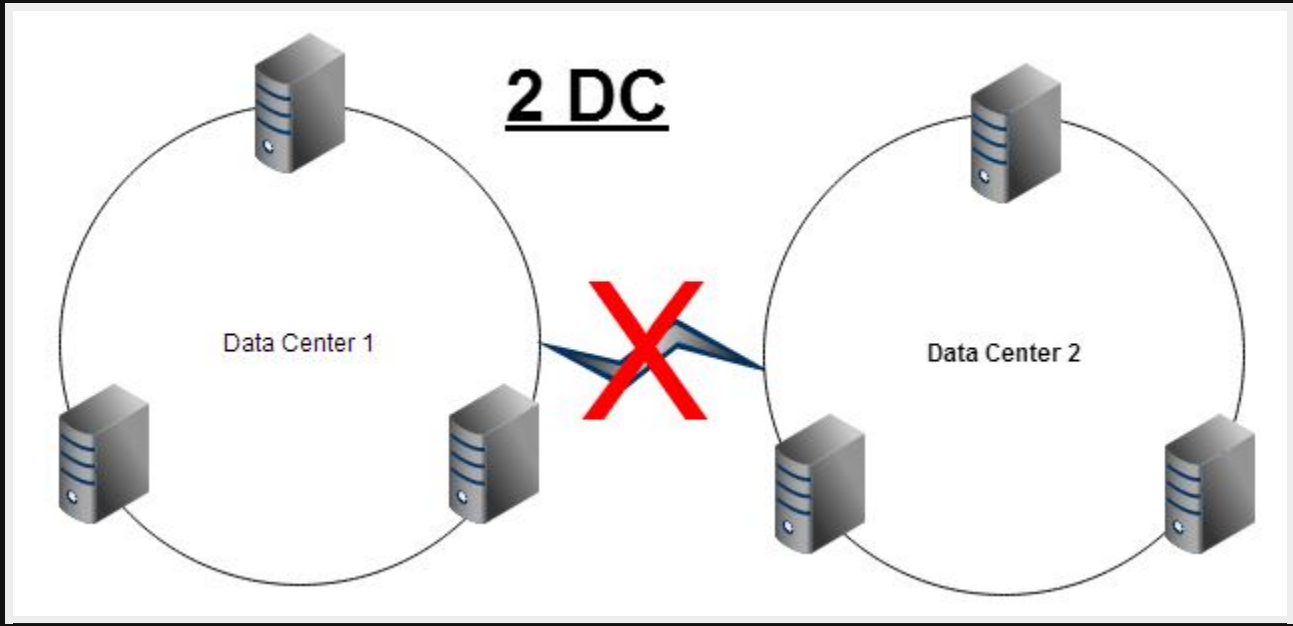
3 Questions - Bonus

question 1

What risks do we have with 2 DC's and using Write/Read CL of LOCAL_QUORUM?



Split-brain



Who wins? Neither side knows the other side!

CAP theorem - choose wisely

Use QUORUM but risk availability
when there are issues with the WAN link

Arrange a third DC... €€€

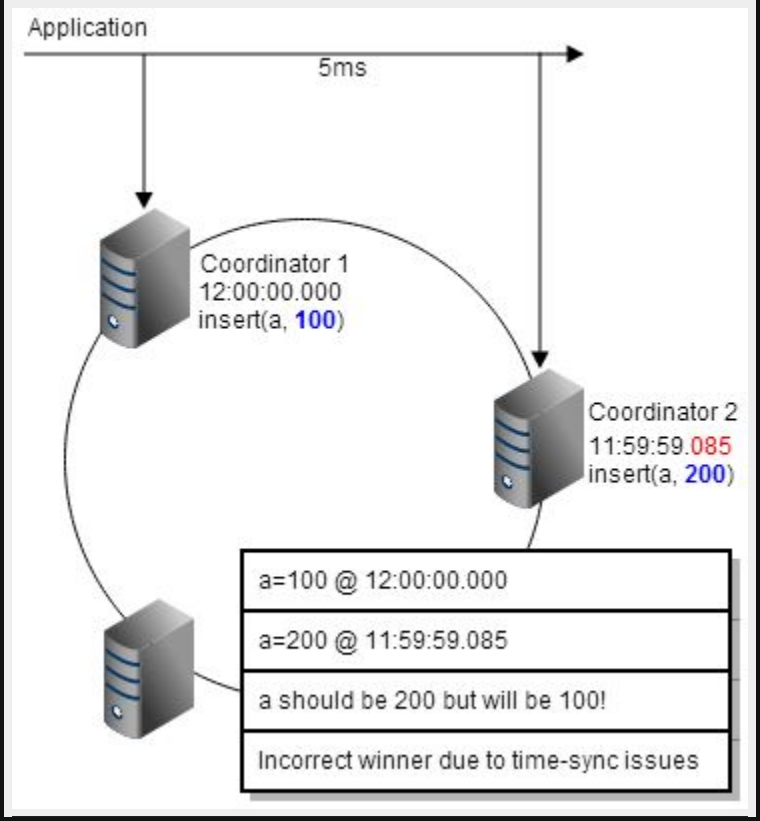
Applications apply DC stickiness
whilst processing data which increases complexity

question 2

Does QUORUM read and QUORUM write guarantee consistency?

No - it gives strong consistency

NTP must be set-up correctly
Know your flow of data
Model as idem-potent-ly as possible



question 3

Will C* solve all your IT issues?

No - but it will help solve a lot of your problems

Performance and availability are tuneable based on needs and costs

Consistency is solved by design and knowing the flow of data

Going from SQL to noSQL is a paradigm shift

Skills are not readily available

Some issues are better solved by relational databases

Thank you

...graceful bow

Contact
christopher.reedijk@ing.nl
gary.stewart@ing.nl