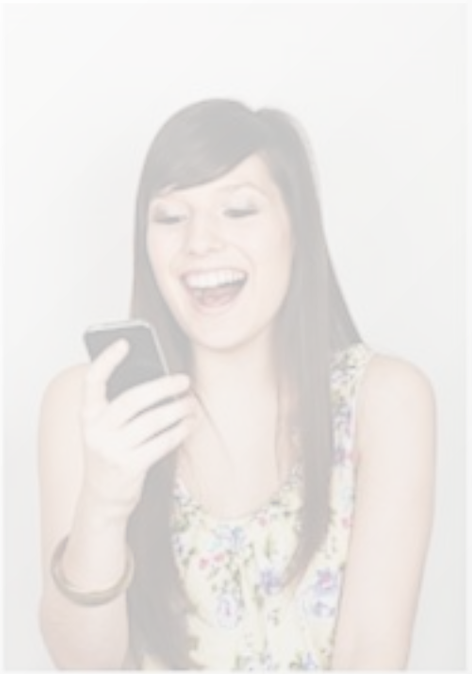
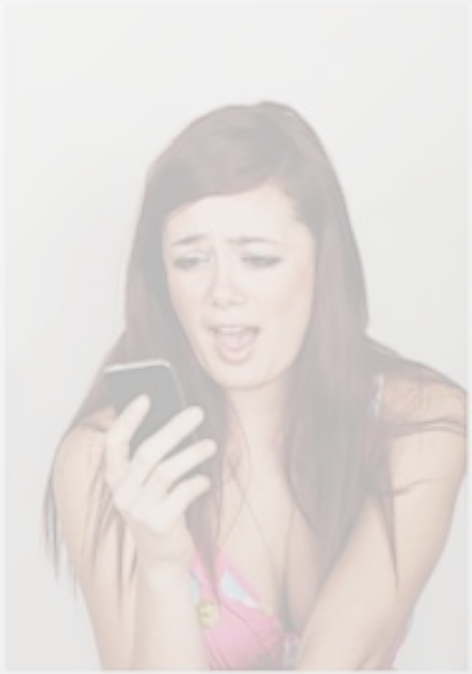
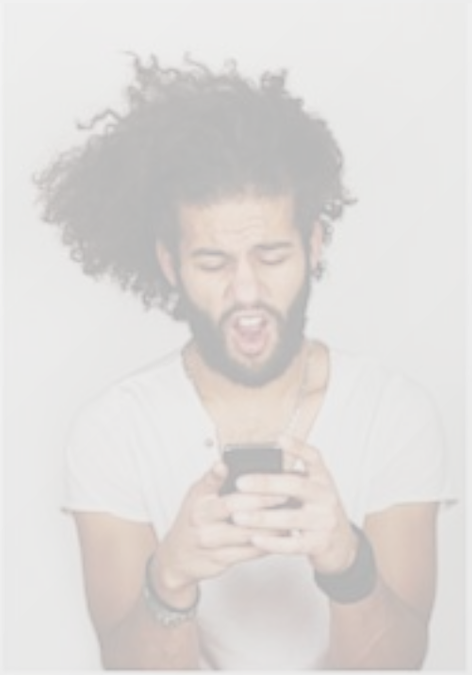
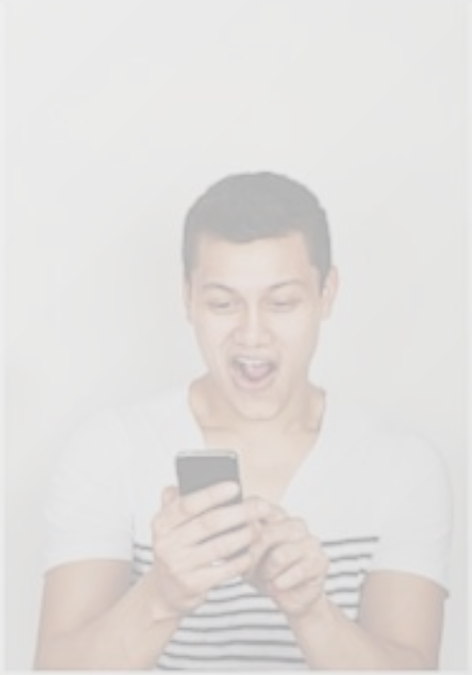
MEET cassandra

w/@ DATASTAX

BACKGROUND

- WHITE PAPER BLEND
  - AMAZON DYNAMO –07
  - GOOGLE BIGTABLE– 06
- WHAT ABOUT FACEBOOK?
- OPEN SOURCE SUCCESSES
- & CUE DATASTAX
- SO WHAT'S IN A NAME?






WHY CASSANDRA?

BECAUSE THE WORLD HAS CHANGED

STORY TIME

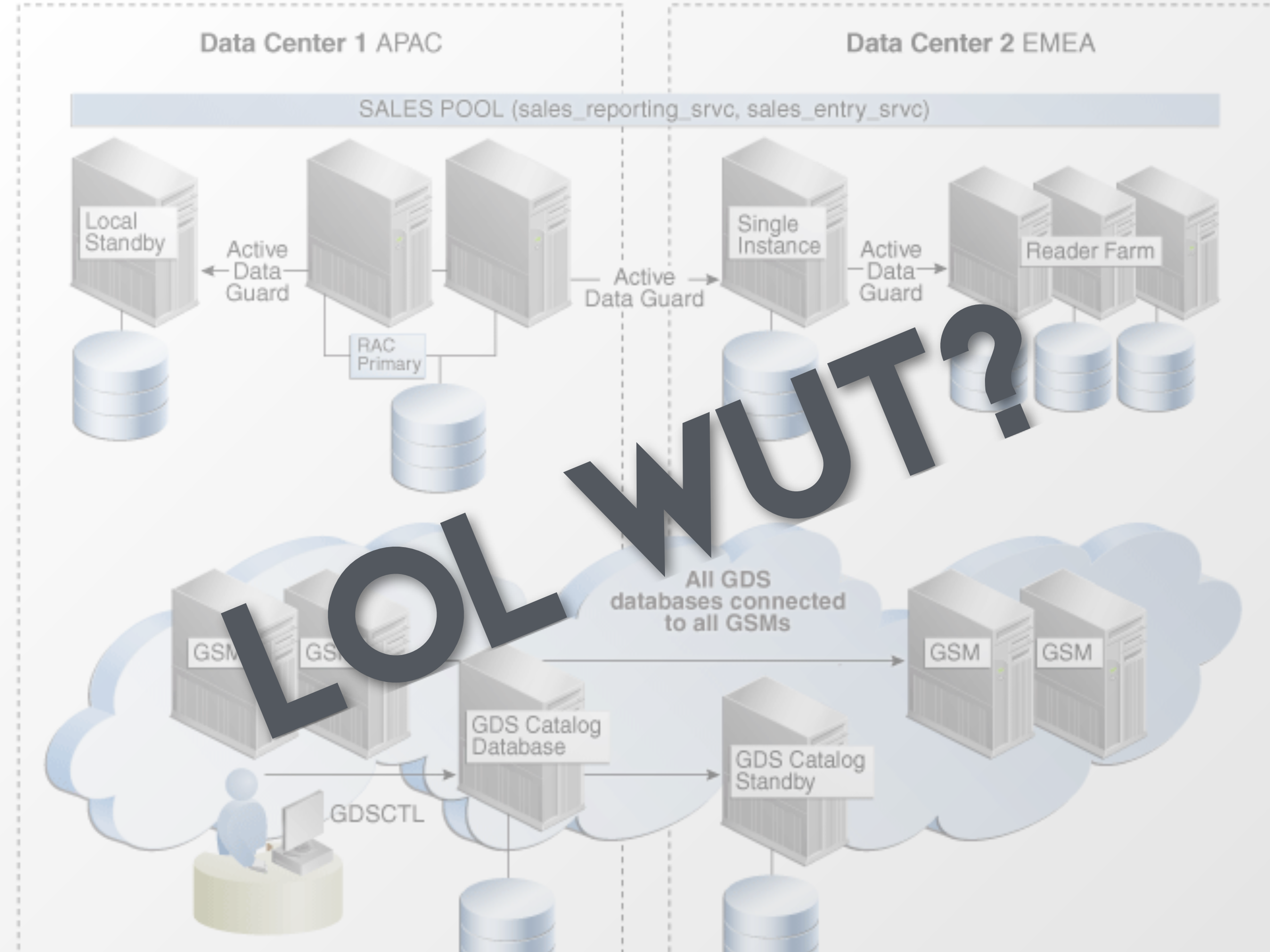
WHAT WE BELIEVE:

- THE NETWORK IS RELIABLE.
- LATENCY IS ZERO.
- BANDWIDTH IS INFINITE.
- THE NETWORK IS SECURE.
- TOPOLOGY DOESN'T CHANGE.
- THERE IS ONE ADMINISTRATOR.
- TRANSPORT COST IS ZERO.
- THE NETWORK IS HOMOGENEOUS.

http://en.wikipedia.org/wiki/Fallacies_of_distributed_computing

HEY, I'VE GOT SOME DATA!

I'VE GOT SOME BIGGER DATA!


Data Center 1 APAC
Data Center 2 EMEA
SALES POOL (sales_reporting_srvc, sales_entry_srvc)
Local Standby
Active Data Guard
RAC Primary
Active Data Guard
Single Instance
Active Data Guard
Reader Farm
LOL WUT?
All GDS databases connected to all GSMs
GSM
GSM
GSM
GSM
GDS Catalog Database
GDS Catalog Standby
GDSCTL


UH OH...
BIG
DATA
???

HIGH

24/7

365

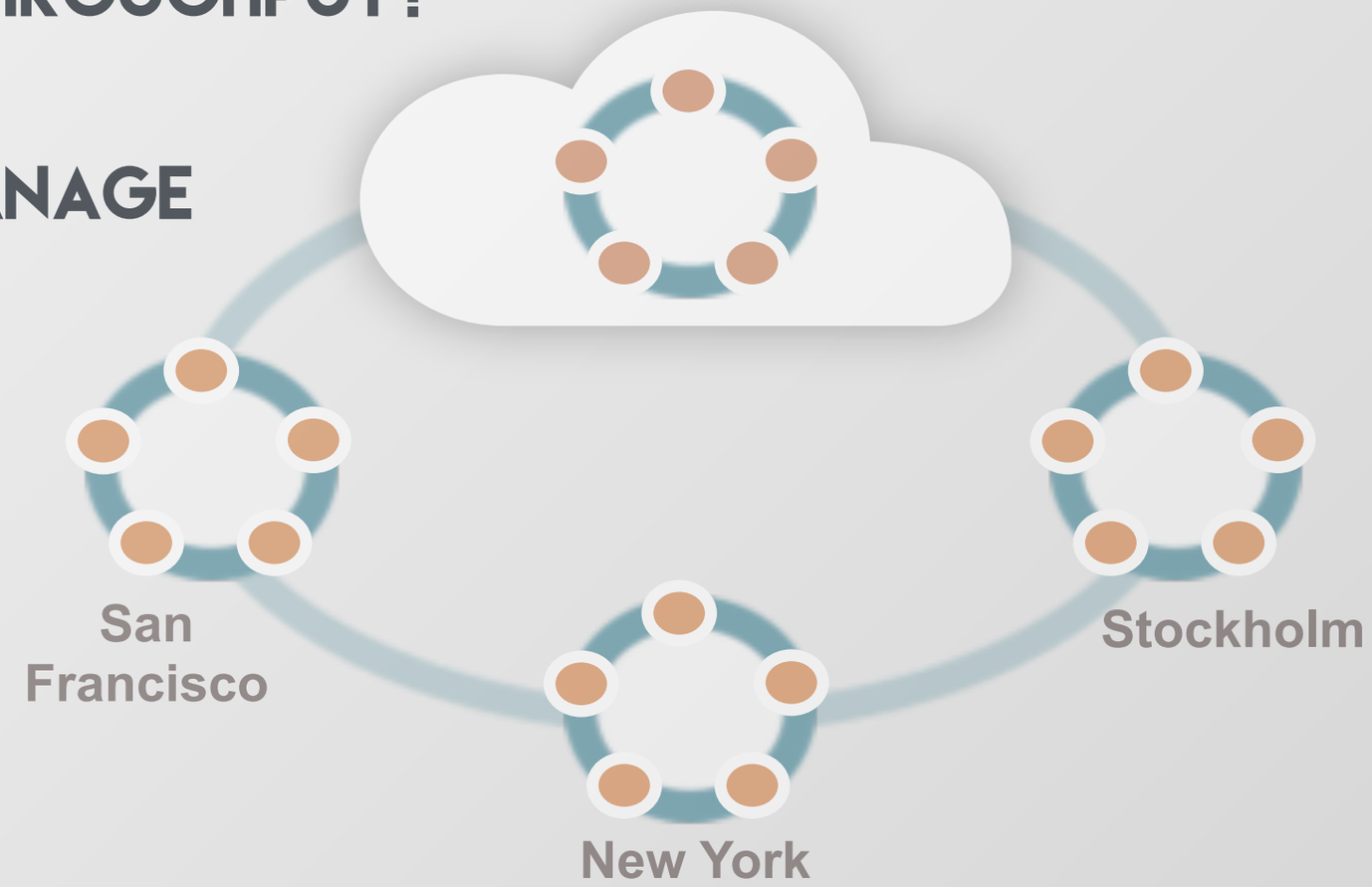
AVAILABILITY

http://planetcassandra.org/nosql-performance-benchmarks/

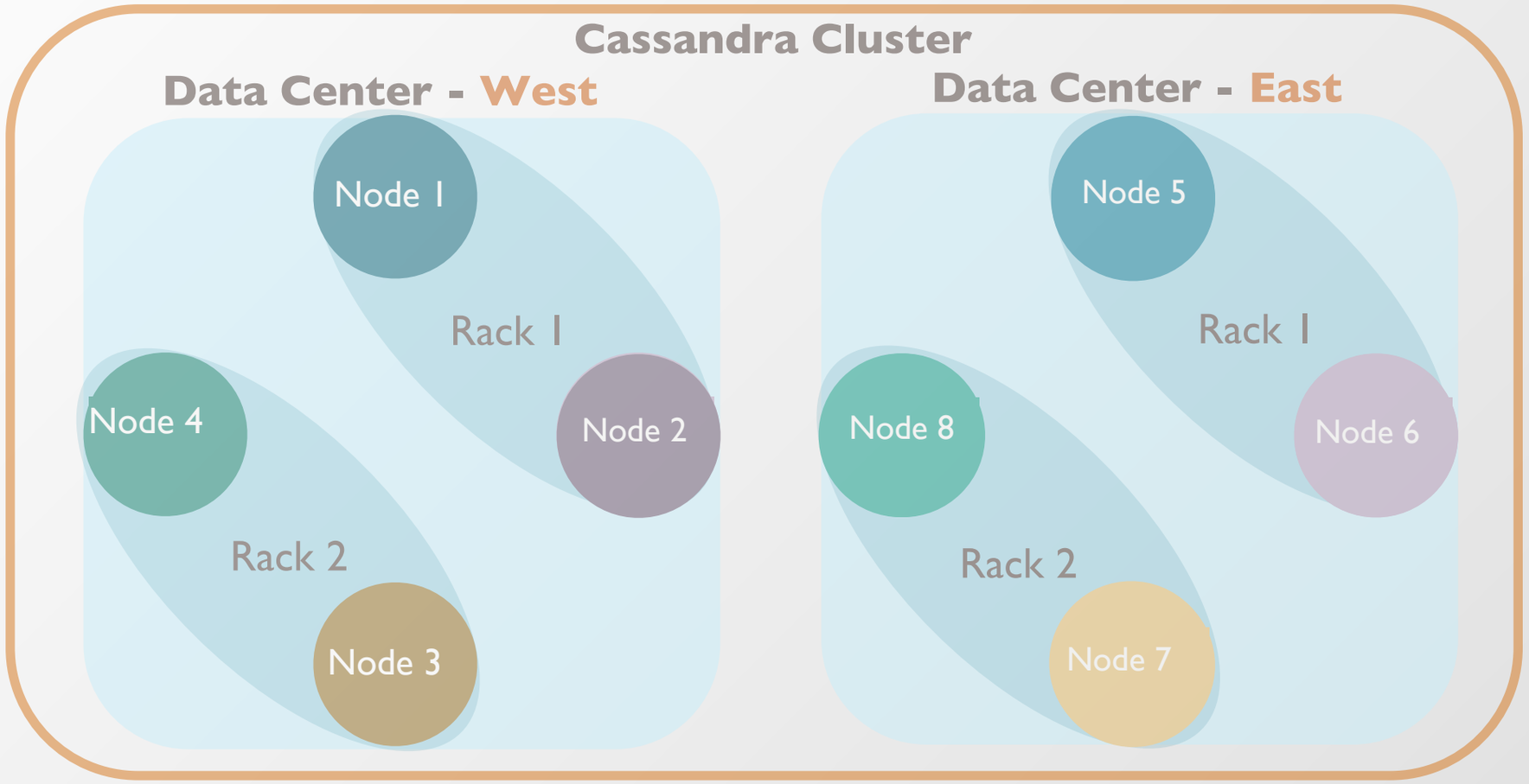
ARCHITECTURE

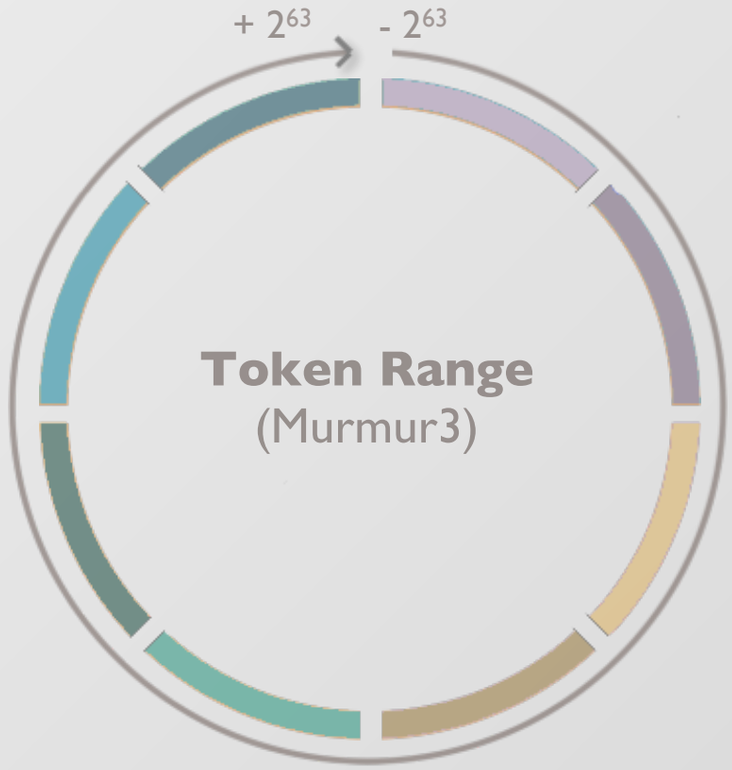
ARCHITECTED W/ SCALE IN MIND

- MASTERLESS - PEER TO PEER
- HIGH AVAILABILITY
- NO SPOFS
- MULTI-DC
- RUNS ON COMMODITY HARDWARE
- LINEAR SCALABILITY: MORE THROUGHPUT?
- FAST
- DISTRIBUTED
- EASY TO OPERATIONALLY MANAGE



WHAT IS A C* CLUSTER?

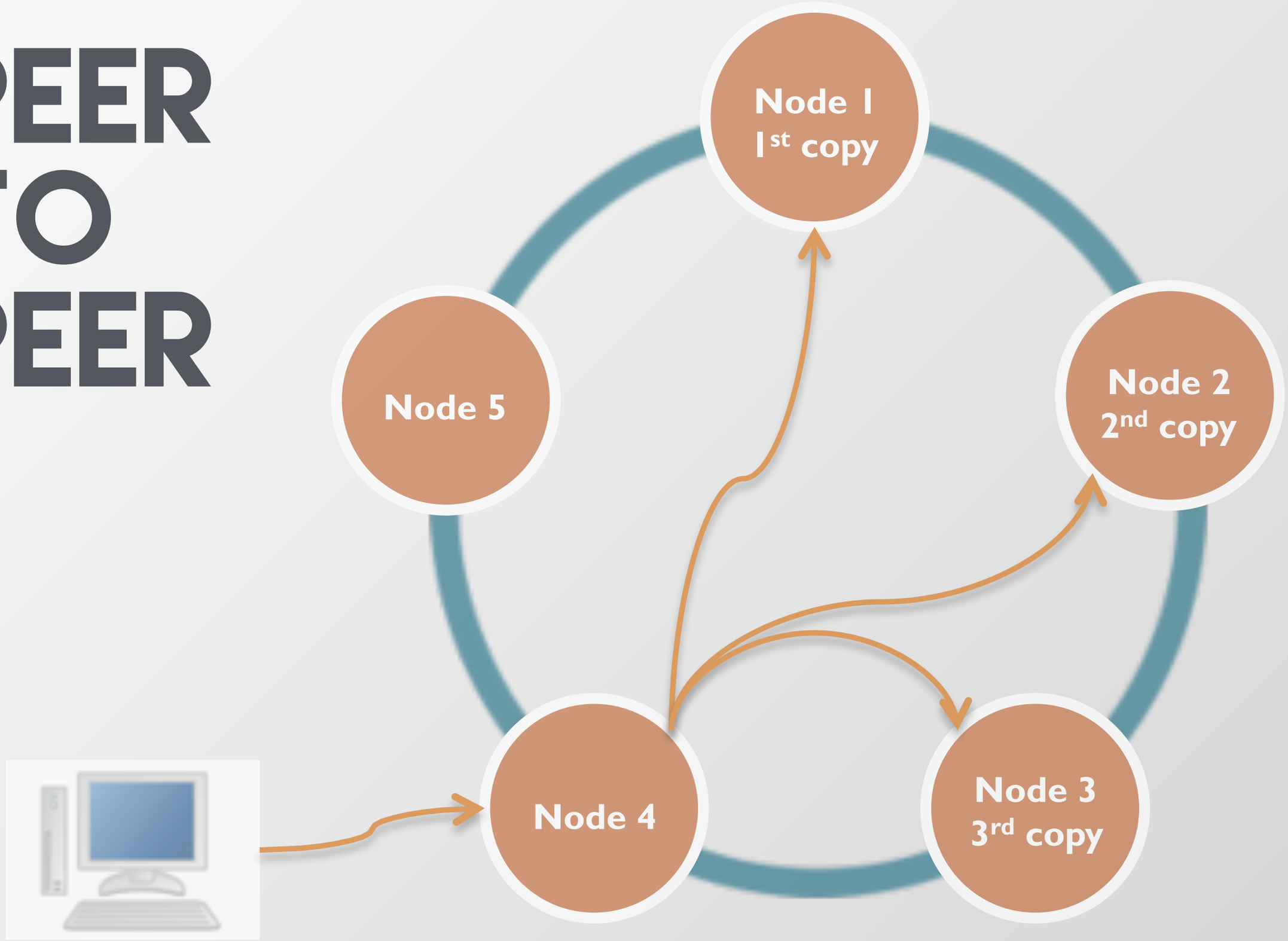




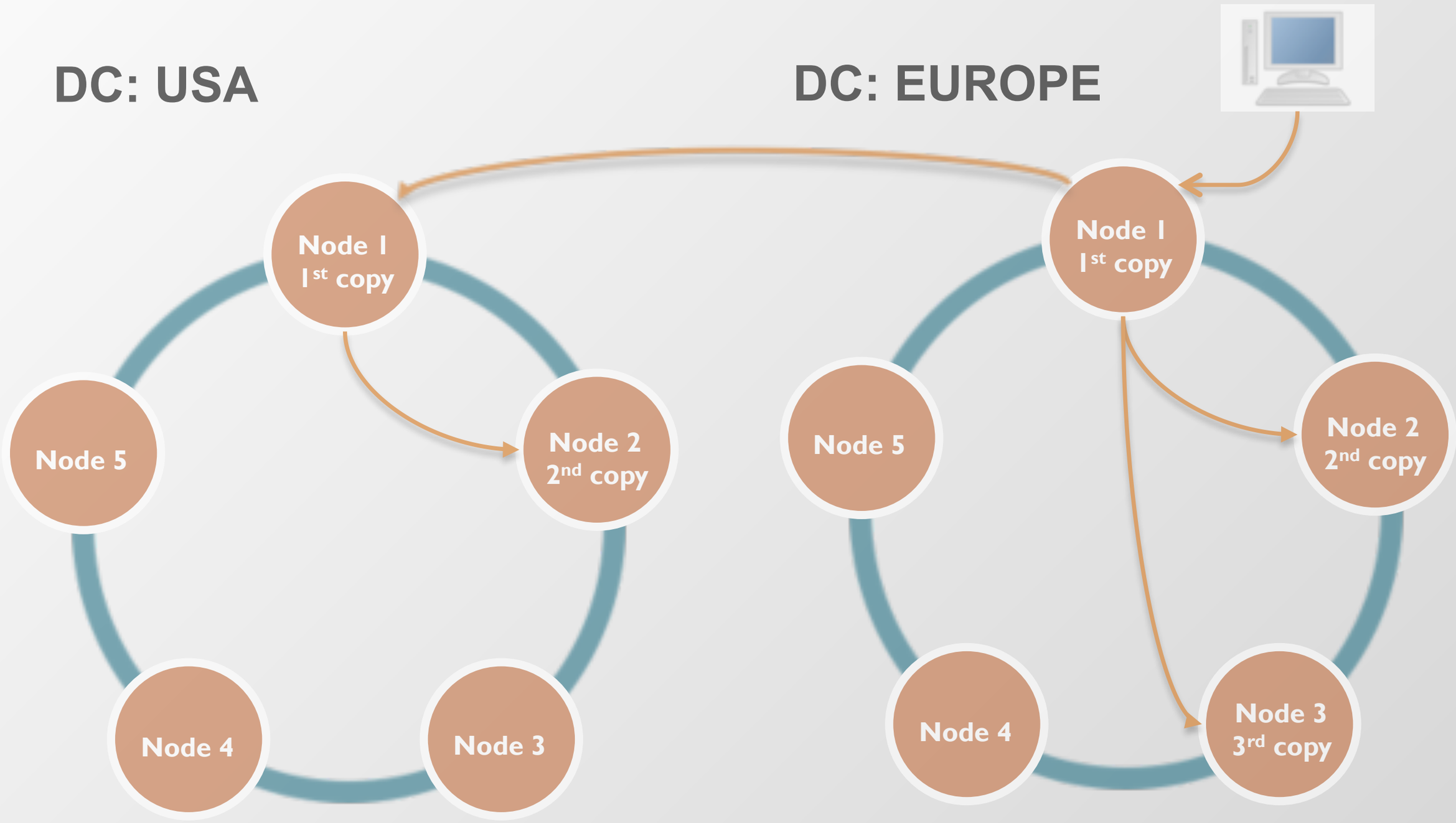
A PEER TO PEER SET OF NODES

- NODE – ONE CASSANDRA INSTANCE
- RACK – A LOGICAL SET OF NODES
- DATA CENTER – A LOGICAL SET OF RACKS
- CLUSTER – THE FULL SET OF NODES WHICH MAP TO A SINGLE COMPLETE TOKEN RING

PEER TO PEER

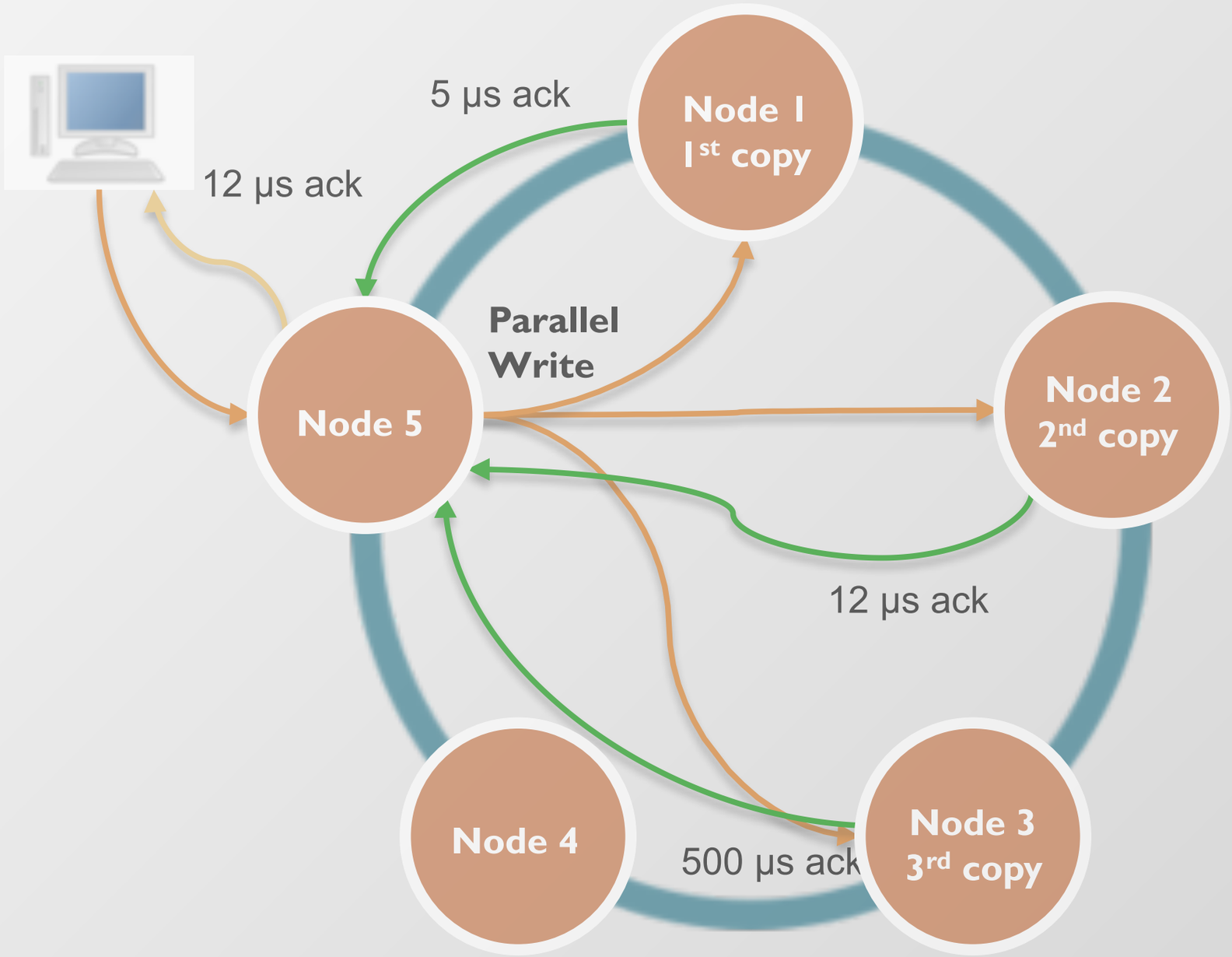


DATA CENTERS



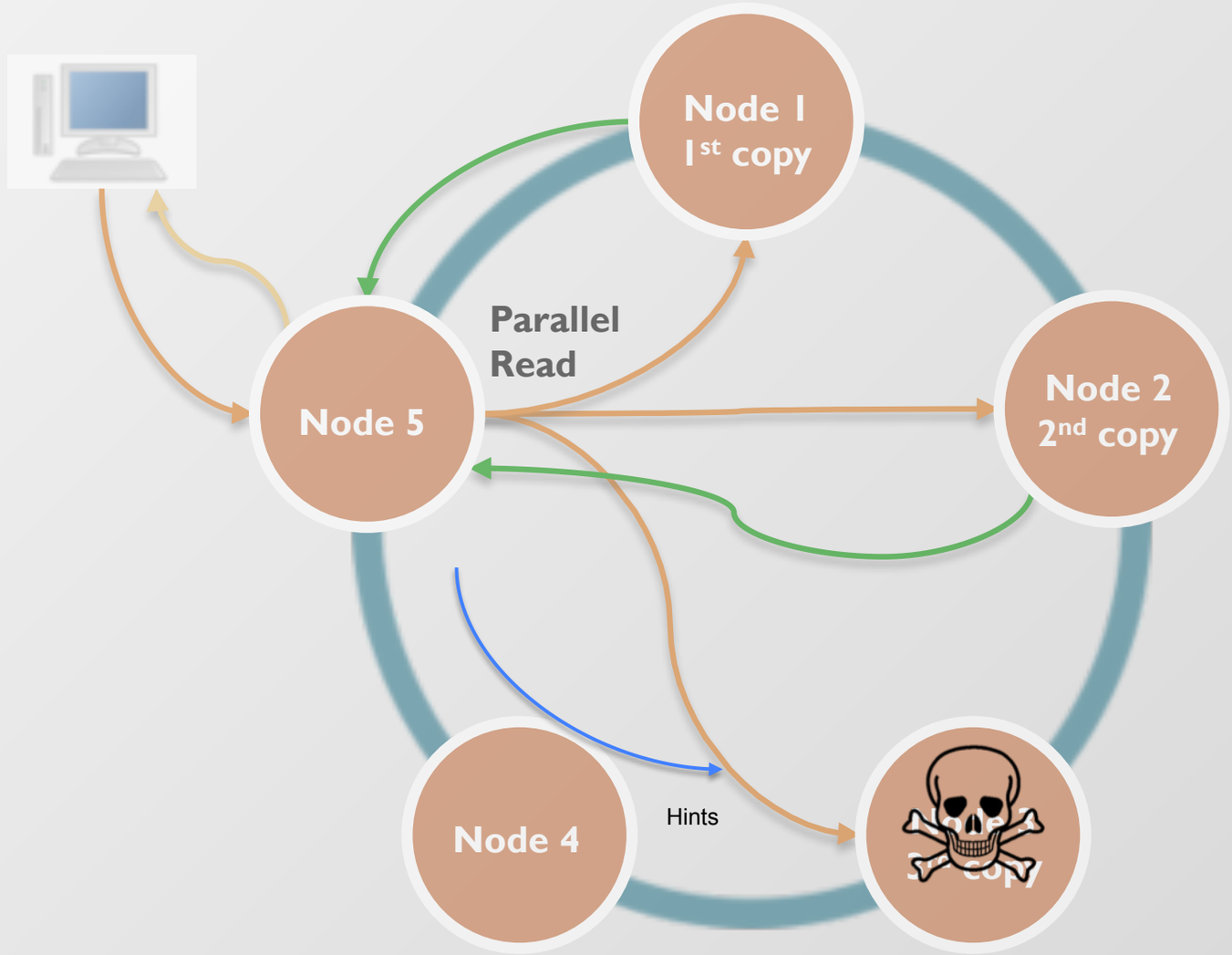
TUNABLE CONSISTENCY

Write
Consistency Level = QUORUM
Replication Factor = 3



CONTINUOUS AVAILABILITY

Read
Consistency Level = QUORUM
Replication Factor = 3



LET'S THINK ABOUT A MUSIC DATABASE...

DATA MODEL AND STORAGE MODEL

CQL

How to create, use and drop keyspaces/schemas?

- To create a keyspace

```
CREATE KEYSPACE musicdb
WITH replication = {
'class': 'SimpleStrategy',
'replication_factor' : 3
};
```

- To assign the working default keyspace for a *cqlsh* session

```
USE musicdb;
```

- To delete a keyspace and all internal data objects

```
DROP KEYSPACE musicdb;
```

What is the syntax of the CREATE TABLE statement?

- The CQL below creates a table in the current keyspace

Primary key declared inline

```
CREATE TABLE performer (
  name VARCHAR PRIMARY KEY,
  type VARCHAR,
  country VARCHAR,
  style VARCHAR,
  founded INT,
  born INT,
  died INT
);
```

Primary key declared in separate clause

```
CREATE TABLE performer (
  name VARCHAR,
  type VARCHAR,
  country VARCHAR,
  style VARCHAR,
  founded INT,
  born INT,
  died INT,
  PRIMARY KEY (name)
);
```

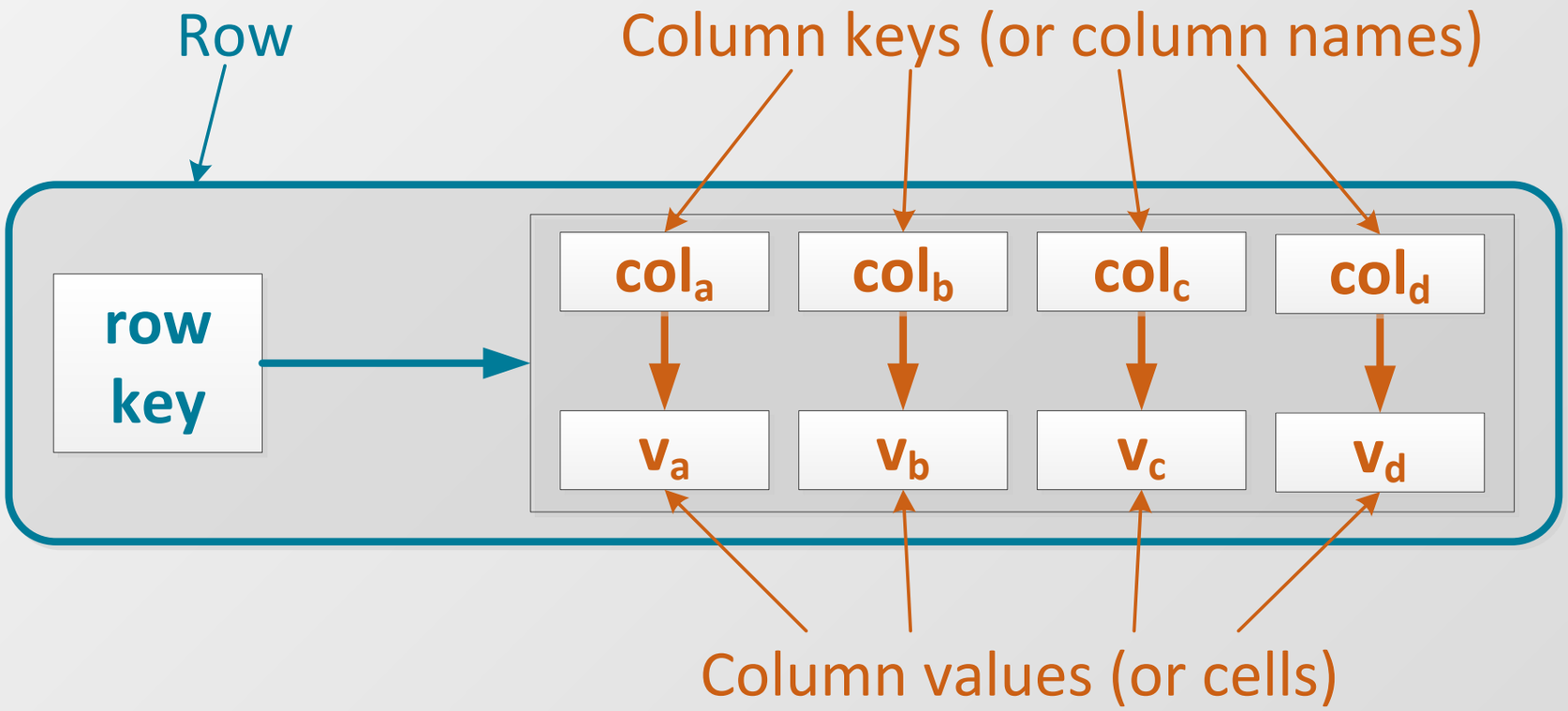
STORAGE MODEL

What is a CQL table and how is it related to a column family?

- A *CQL table* is a *column family*
  - CQL tables provide two-dimensional views of a column family, which contains potentially multi-dimensional data, due to composite keys and collections
- *CQL table* and *column family* are largely interchangeable terms
  - Not surprising when you recall *tables* and *relations*, *columns* and *attributes*, *rows* and *tuples* in relational databases
- Supported by declarative language Cassandra Query Language
  - Data Definition Language, subset of CQL
  - SQL-like syntax, but with somewhat different semantics
  - Convenient for defining and expressing Cassandra database schemas

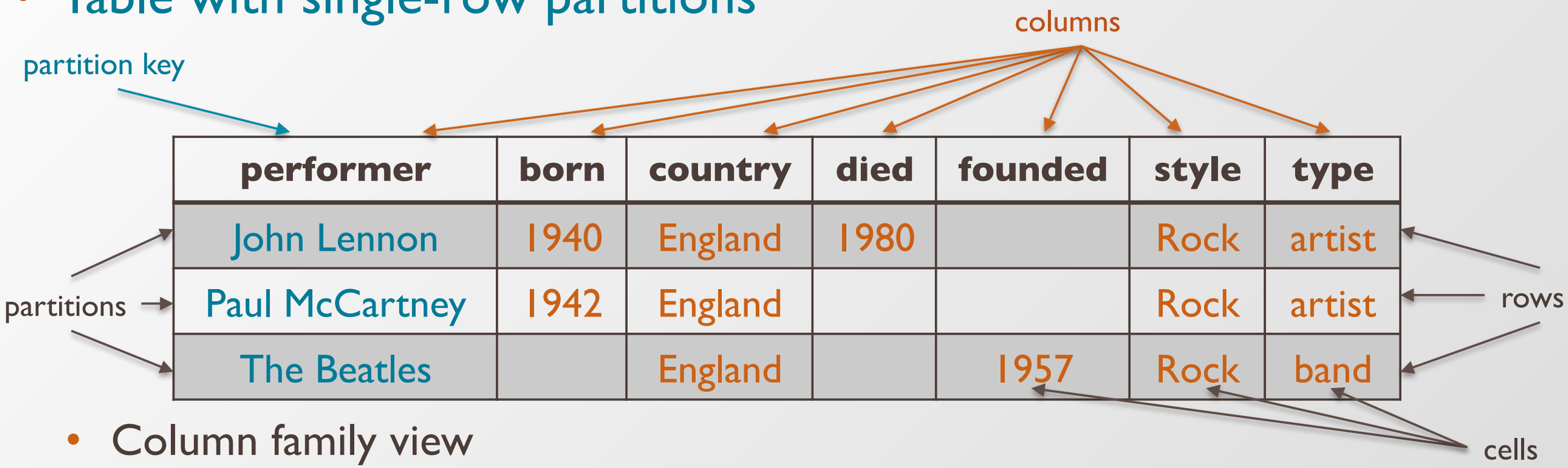
What are row, row key, column key, and column value?

- *Row* is the smallest unit that stores related data in Cassandra
  - Rows – individual rows constitute a *column family*
  - Row key – uniquely identifies a *row* in a *column family*
  - Row – stores pairs of *column keys* and *column values*
  - Column key – uniquely identifies a *column value* in a *row*
  - Column value – stores one value or a *collection* of values

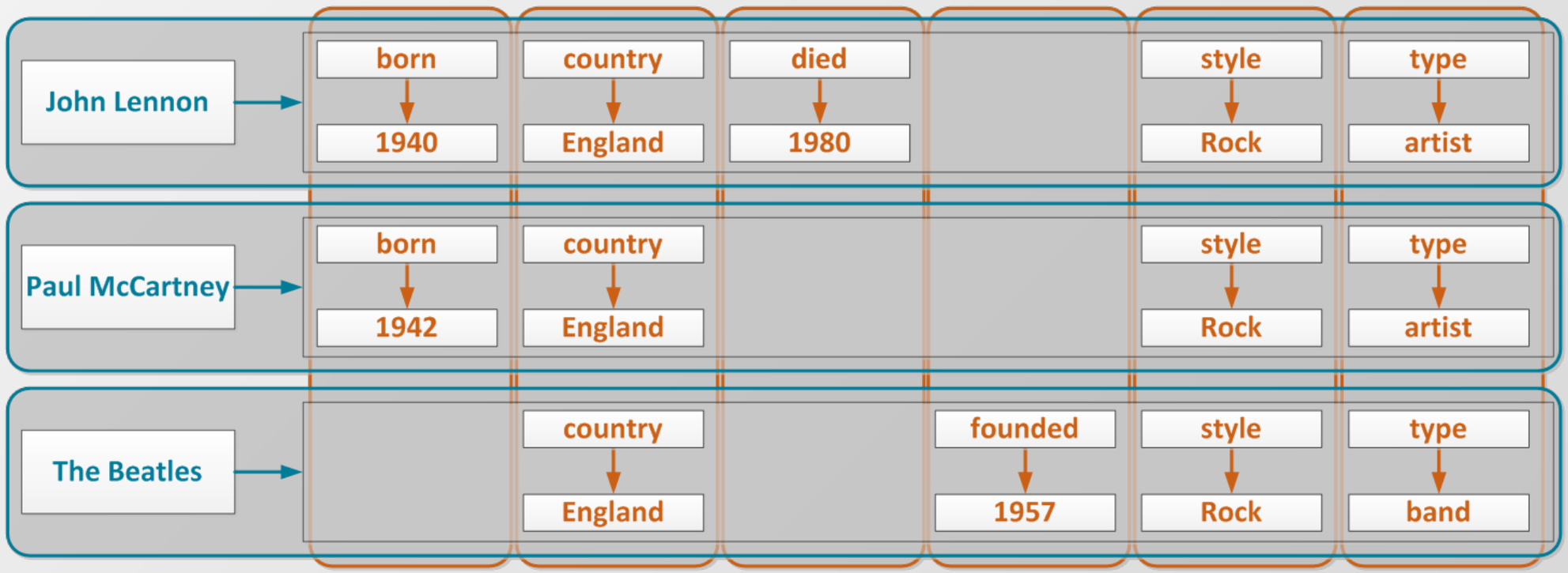


What are partition, partition key, row, column, and cell?

- Table with single-row partitions



- Column family view

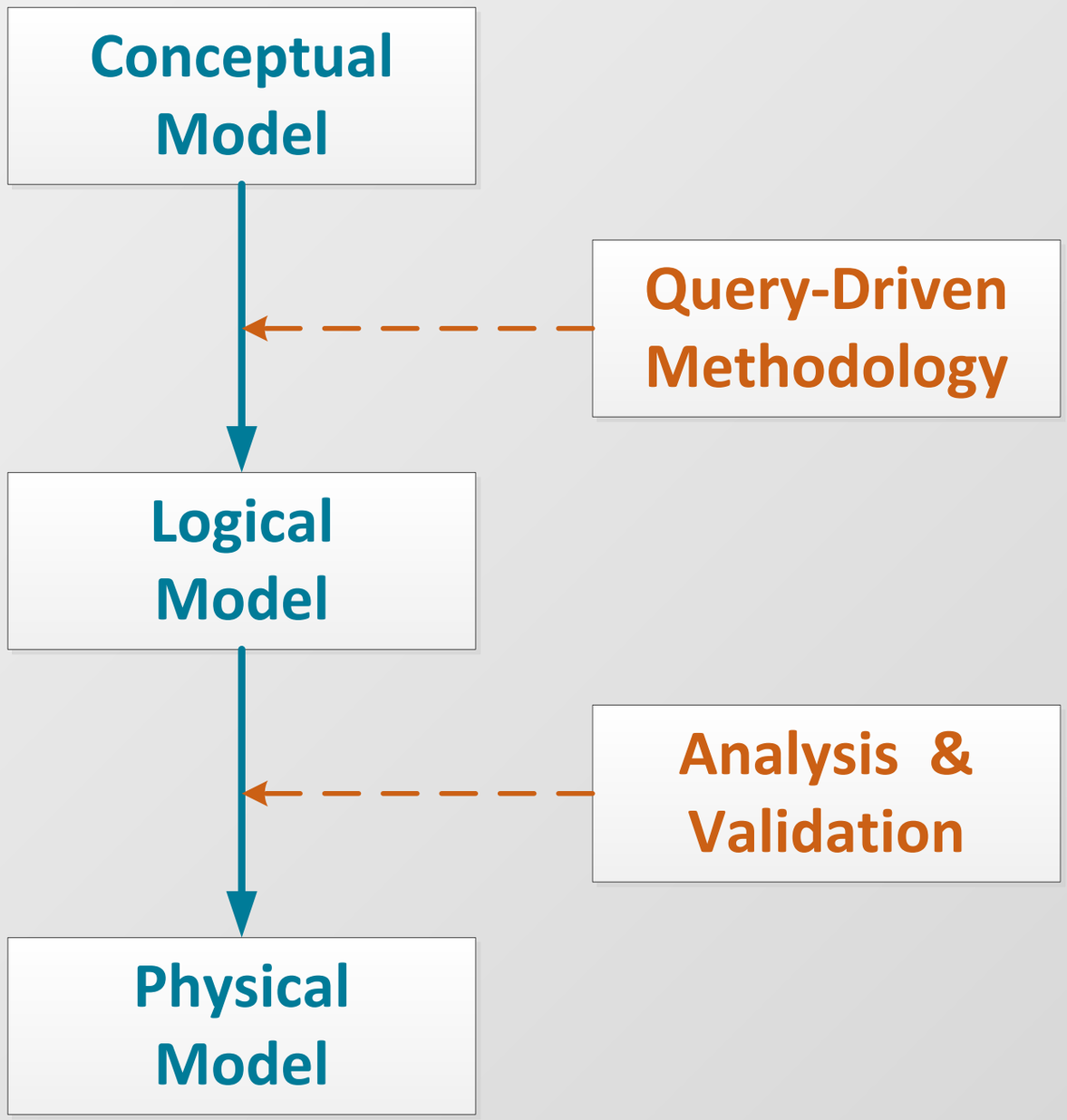


DATA MODEL

What is a data modeling framework?

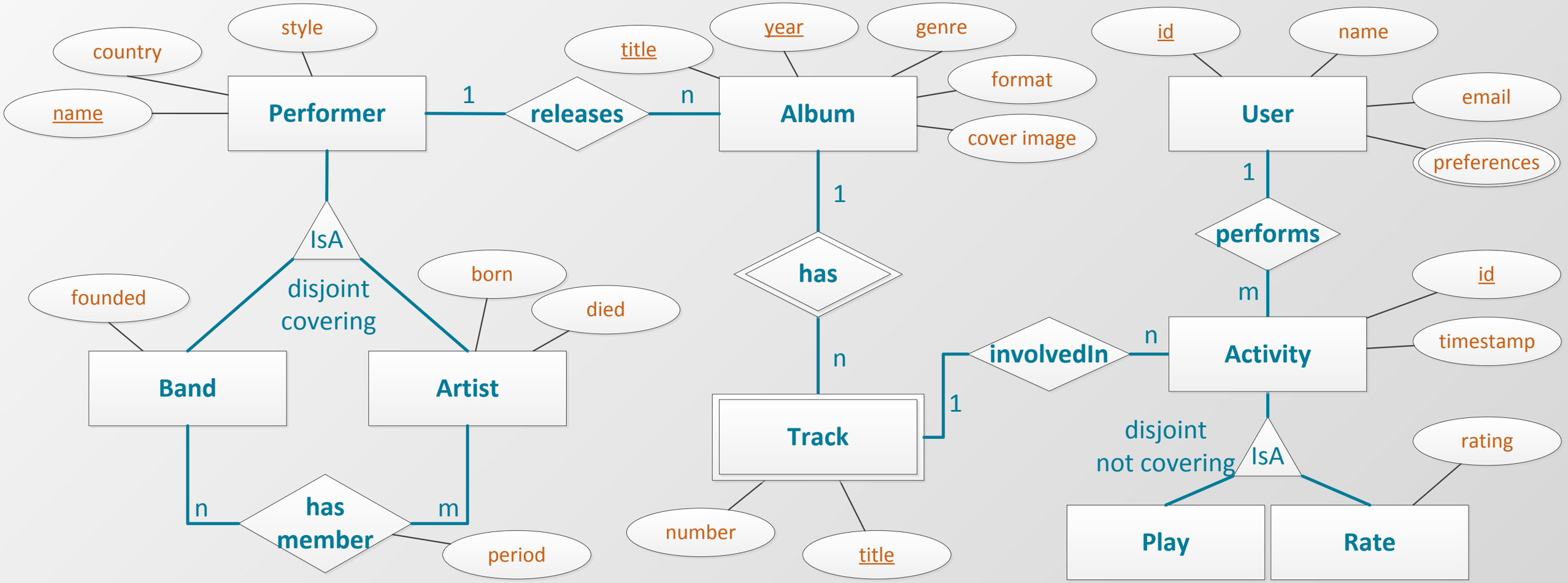
- Defines transitions between models
  - Query-driven methodology
  - Formal analysis and validation

- Defines a scientific approach to data modeling
  - Modeling rules
  - Mapping patterns
  - Schema optimization techniques

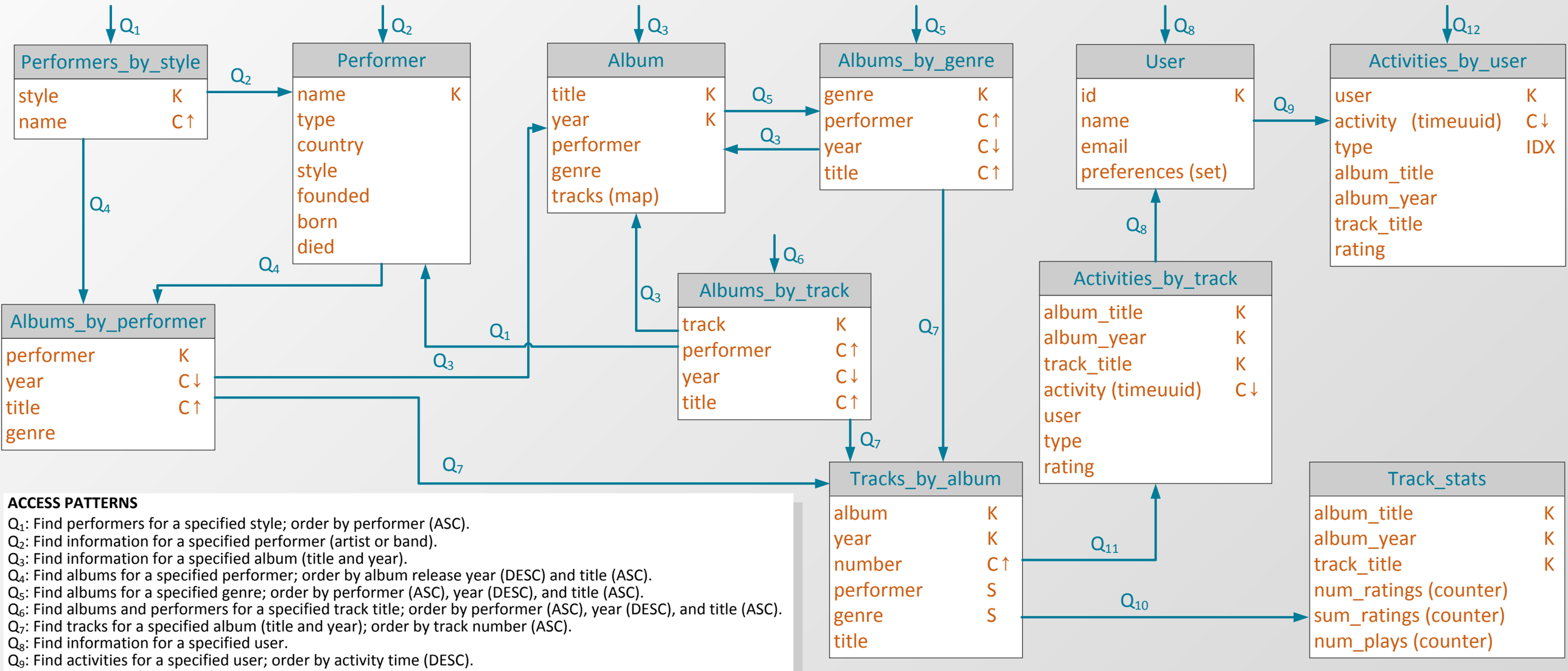


What is a conceptual data model?

- Conceptual data model for music data
  - ER diagram (Chen notation)
  - Describes entities, relationships, roles, keys, cardinalities
    - What is possible and what is not in existing or future data



A SAMPLE DATA MODEL



ACCESS PATTERNS

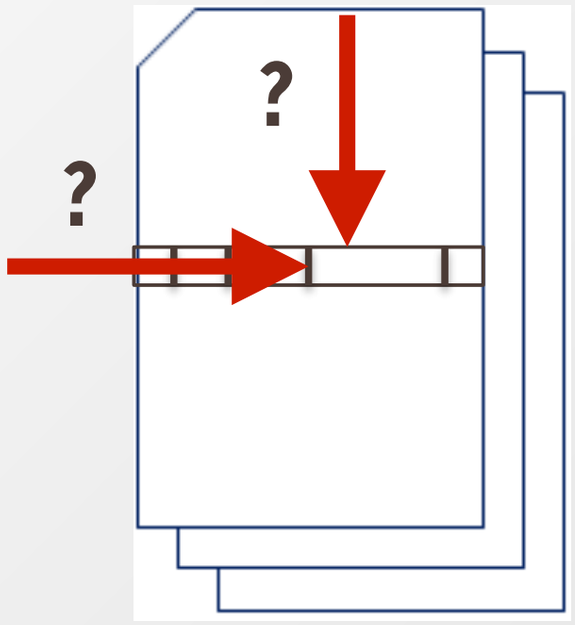
Q_1: Find performers for a specified style; order by performer (ASC).
Q_2: Find information for a specified performer (artist or band).
Q_3: Find information for a specified album (title and year).
Q_4: Find albums for a specified performer; order by album release year (DESC) and title (ASC).
Q_5: Find albums for a specified genre; order by performer (ASC), year (DESC), and title (ASC).
Q_6: Find albums and performers for a specified track title; order by performer (ASC), year (DESC), and title (ASC).
Q_7: Find tracks for a specified album (title and year); order by track number (ASC).
Q_8: Find information for a specified user.
Q_9: Find activities for a specified user; order by activity time (DESC).
Q_{10}: Find statistics for a specified track.
Q_{11}: Find user activities for a specified track; order by activity time (DESC).
Q_{12}: Find user activities for a specified activity type.
...


WRITES

How does Cassandra write so fast?

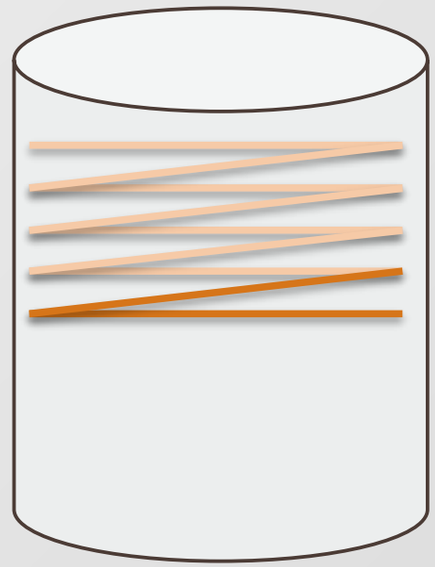
- Cassandra is a log-structured storage engine
  - Data is sequentially appended, not placed in pre-set locations

RDBMS



Seeks and writes values to various pre-set locations

CASSANDRA



Continuously appends to a log

What are the key components of the write path?

- Each node implements four key components to handle its writes
  - Memtables – in-memory tables corresponding to CQL tables, with indexes
  - CommitLog – append-only log, replayed to restore downed node's *Memtables*
  - SSTables – *Memtable* snapshots periodically flushed to disk, clearing heap
  - Compaction – periodic process to merge and streamline *SSTables*

- When any node receive any write request
  1. The record appends to the *CommitLog*, and
  2. The record appends to the *Memtable* for this record's target CQL table
  3. Periodically, *Memtables* flush to *SSTables*, clearing JVM heap and *CommitLog*
  4. Periodically, *Compaction* runs to merge and streamline *SSTables*

How does the write path flow on a node?

Coordinator

Write <3, Betty, Blue, 63>

Acknowledge

Each write request …

Periodically …

Memtable (corresponds to a CQL table)

partition key1	first:Oscar	last:Orange	level:42
partition key2	first:Ricky	last:Red	
partition key3	first:Betty	last:Blue	level:63

Flush current state to SSTable

Node memory

Node file system

Append Only

Periodically …

Compaction

Compact related SSTables

CommitLog

SSTables

What are Memtables and how are they flushed to disk?

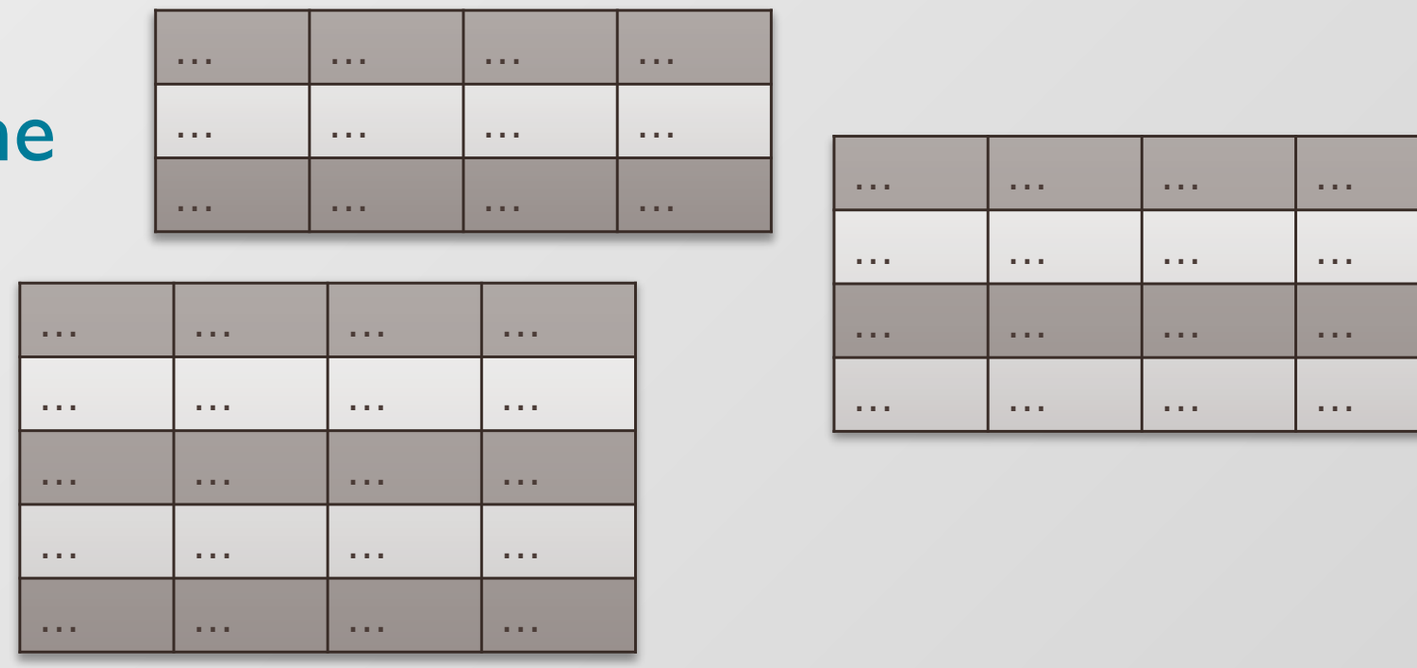
Memtable

partition key1	first:Oscar	last:Orange	level:42
partition key2	first:Ricky	last:Red	
partition key3	first:Betty	last:Blue	level:63

- *Memtables* are in-memory representations of a CQL table
  - Each node has a *Memtable* for each CQL table in the keyspace
  - Each *Memtable* accrues writes and provides reads for data not yet flushed
  - Updates to *Memtables* mutate the in-memory partition
- When a *Memtable* flushes to disk
  1. Current *Memtable* data is written to a new immutable *SSTable* on disk
  2. JVM heap space is reclaimed from the flushed data
  3. Corresponding *CommitLog* entries are marked as flushed

What is a SSTable and what are its characteristics?

- A *SSTable* ("sorted string table") is
  - an immutable file of sorted partitions
  - written to disk through fast, sequential i/o
  - contains the state of a *Memtable* when flushed
- The current data state of a CQL table is comprised of
  - its corresponding *Memtable* plus
  - all current *SSTables* flushed from that *Memtable*
- *SSTables* are periodically compacted from many to one

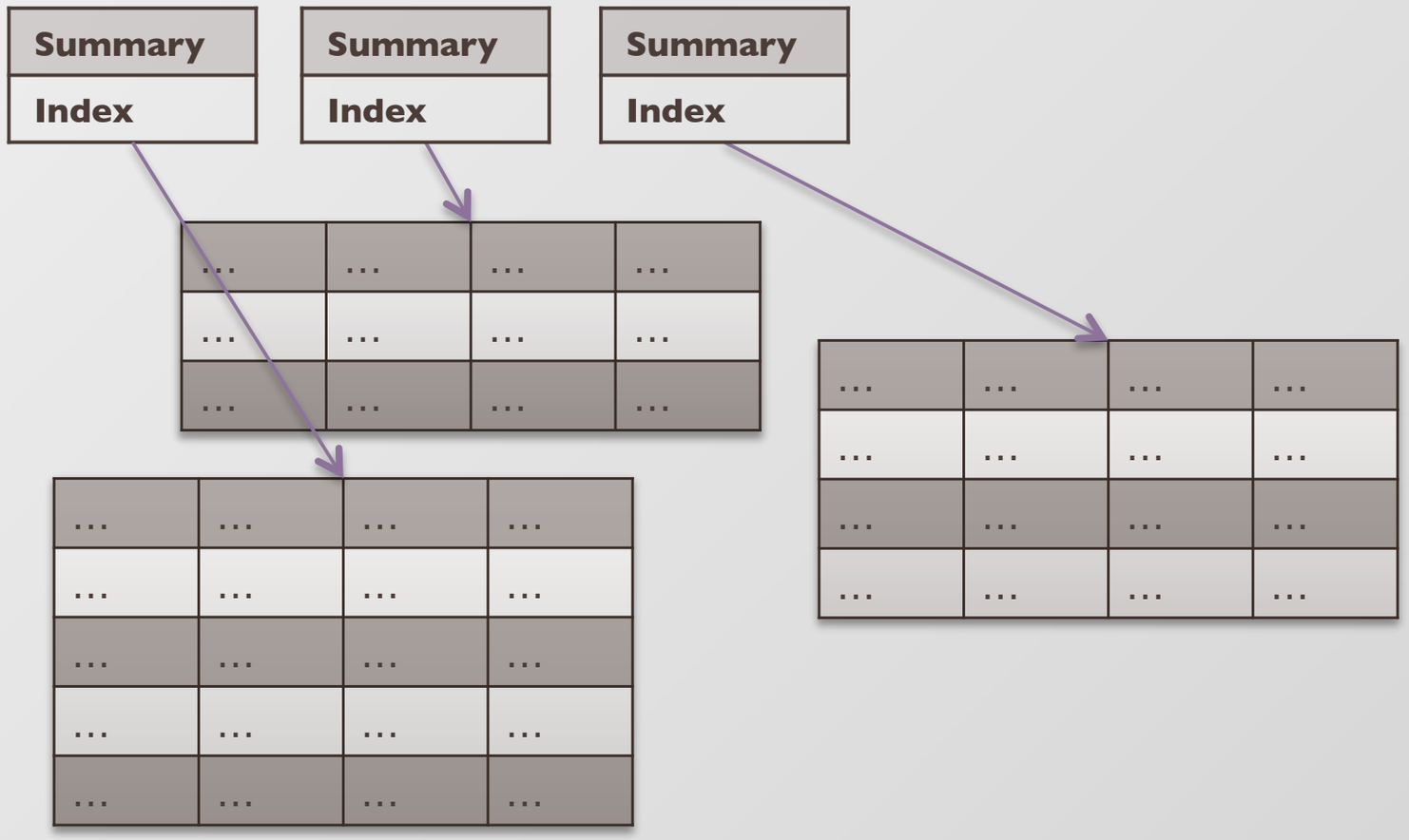


What is a SSTable and what are its characteristics?

- For each *SSTable*, two structures are created
  - Partition index – list of its primary keys and row start positions
  - Partition summary – in-memory sample of its *partition index* (default: *1 partition key of 128*)

Memtable (corresponds to a CQL table)

partition key1	first:Oscar	last:Orange	level:42
partition key2	first:Ricky	last:Red	
partition key3	first:Betty	last:Blue	level:63



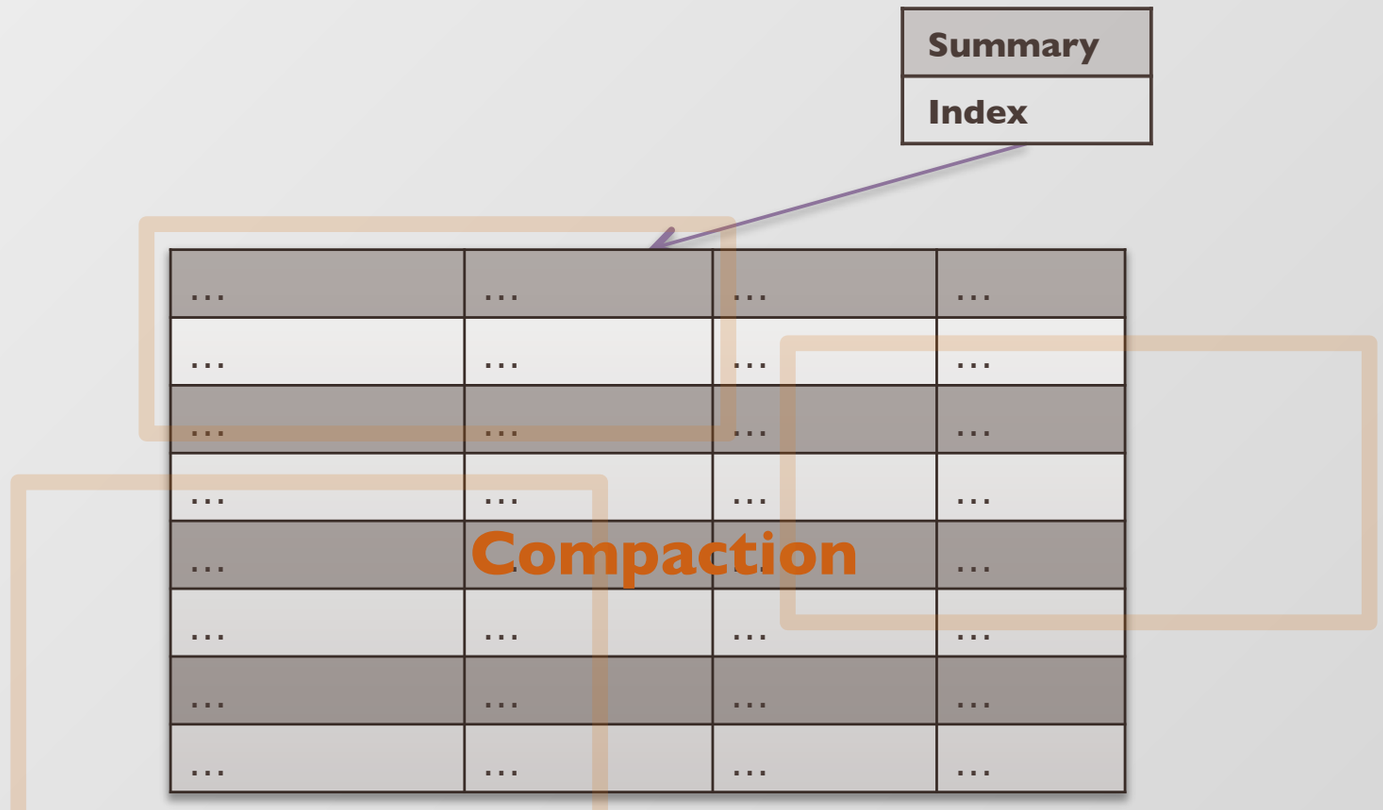
SSTables

What is compaction?

- Updates do mutate *Memtable* partitions, but its *SSTables* are immutable
  - no SSTable seeks/overwrites
  - SSTables just accrue new timestamped updates
- So, *SSTables* must be periodically *compacted*
  - related *SSTables* are merged
  - most recent version of each column is compiled to one partition in one new SSTable
  - partitions marked for deletion are evicted
  - old *SSTables* are deleted

Memtable (corresponds to a CQL table)

partition key1	first:Oscar	last:Orange	level:42
partition key2	first:Ricky	last:Red	
partition key3	first:Betty	last:Blue	level:63



SSTables

What is sstable2json?

- bin/sstable2json is a utility which exports an *SSTable* in JSON format, for testing and debugging
  - -k exclude a set of keys specified in HEX format (limit: 500)
  - -x exclude a specified set of keys (limit: 500)
  - -e enumerate keys only

```
./sstable2json [full_path_to_SSTable_Data_file] | more
```

dstraining@DST: /h... | dstraining@DST: /h... | dstraining@DST: /ho... | dstraining@DST: /ho... | dstraining@DST: /va...

```
dstraining@DST:/home/cassandra/bin$ sudo ./sstable2json /var/lib/cassandra/data/music/performer/music-performer-jb-1
-Data.db | more
[
{"key": "53686572796c2043726f77","columns": [["","",1394065306886002], ["born","1962",1394065306886002], ["country",
"United States",1394065306886002], ["died","5317bf9a",1394065306886002,"d"], ["founded","5317bf9a",1394065306886002,
"d"], ["style","Rock",1394065306886002], ["type","artist",1394065306886002]]},
{"key": "426c61636b20426f74746c652053636f7463682057686973746b792050697065204261616e64","columns": [["","",13940653065360
01], ["born","5317bf9a",1394065306536001,"d"], ["country","Scotland",1394065306536001], ["died","5317bf9a",139406530
6536001,"d"], ["founded","1989",1394065306536001], ["style","Pipe and Drum",1394065306536001], ["type","band",139406
5306536001]]},
{"key": "42656c6c6566697265","columns": [["","",1394065306442001], ["born","5317bf9a",1394065306442001,"d"], ["count
ry","Ireland",1394065306442001], ["died","5317bf9a",1394065306442001,"d"], ["founded","5317bf9a",1394065306442001,"d
"], ["style","Unknown",1394065306442001], ["type","band",1394065306442001]]},
```


READS

How does the read path flow on *each* node?

Row cache hit

Off Heap | On Heap

Coordinator

Read <pk7>

Hit

Row Cache (optional)

pk1, pk2, **pk7**

pk7	first:**Elizabeth**	last:**Blue**	level:**42**

MemTable (e.g., player)

…	…	…	…
pk7	…	…	level:**42** timestamp 1114

Node memory

Node file system

pk1	…	…	…
pk7	~~first:Betty~~ ~~timestamp 541~~	last:***Blue*** timestamp 541	~~level:63~~ ~~timestamp 541~~

pk2	…	…	…
pk7	first:***Elizabeth*** timestamp 994		

pk1	…	…	…
pk2	…	…	…

SSTables (e.g., player)

How does the read path flow on *each* node?

Key cache hit

Off Heap | On Heap

Coordinator

Read <pk7>

Row Cache (optional)

pk1, pk2, **pk7**

Miss

Bloom Filter ?

Bloom Filter ?

Bloom Filter ✖

Key Cache **pk7**

Hit

Hit

Partition Summary

Partition Index

Node memory

Node file system

MemTable (e.g., player)

…	…	…	…
pk7	…	…	level:**42** timestamp 1114

pk1	…	…	…
pk7	~~first:Betty~~ ~~timestamp 541~~	last:***Blue*** timestamp 541	~~level:63~~ ~~timestamp 541~~

pk2	…	…	…
pk7	first:***Elizabeth*** timestamp 994		

pk1	…	…	…
pk2	…	…	…

SSTables (e.g., player)

How does the read path flow on *each* node?

Row and Key miss

Off Heap | On Heap

Coordinator

Read <pk7>

Row Cache (optional)
pk1, pk2, **pk7**

Miss

Bloom Filter ?

Bloom Filter ?

Bloom Filter ✖

Key Cache **pk7**

Miss

Partition Summary

Miss

Partition Summary

Partition Index

Partition Index

MemTable (e.g., player)

pk7	first:**Elizabeth**	last:**Blue**	level:**42**
…	…	…	…
pk7	…	…	level:***42*** timestamp 1114

Node memory

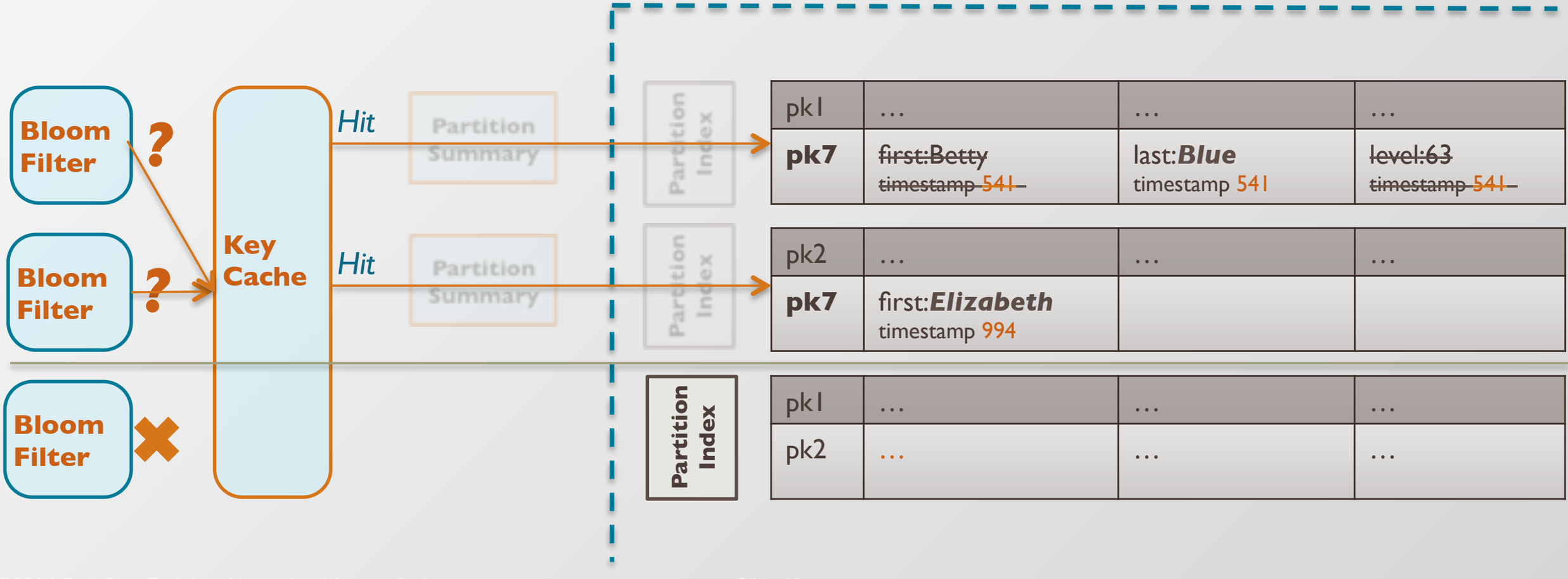
Node file system

pk1	…	…	…
pk7	~~first:Betty~~ ~~timestamp 541~~	last:***Blue*** timestamp 541	~~level:63~~ ~~timestamp 541~~
pk2	…	…	…
pk7	first:***Elizabeth*** timestamp 994		
pk1	…	…	…
pk2	…	…	…

SSTables (e.g., player)

What is a Bloom filter and how does it optimize a read?

- A probabilistic data structure testing if a key may be in a SSTable
  - each SSTable has a Bloom filter on disk, used from off-heap memory
  - false positives are possible, false negatives are not
  - larger tables have a higher possibility of false positives
    - 1gb to 2gb per billion partitions in a SSTable
- Eliminates seeking a partition key in any SSTable without it



How do you execute CQL queries in cqlsh?

dstraining@DST: /home/cassandra

dstraining@DST: /home/cassandra

```
dstraining@DST:/home/cassandra$ bin/cqlsh
Connected to Test Cluster at localhost:9160.
[cqlsh 4.1.1 | Cassandra 2.0.5 | CQL spec 3.1.1 | Thrift protocol 19.39.0]
Use HELP for help.
cqlsh> DESCRIBE KEYSPACES;

system  music  system_traces  demo

cqlsh> USE music;
cqlsh:music> SELECT *
         ... FROM performer
         ... LIMIT 5;

 name                                | born | country        | died | founded | style          | type
-------------------------------------+------+----------------+------+---------+----------------+--------
                         Sheryl Crow | 1962 | United States | null |    null |           Rock | artist
 Black Bottle Scotch Whisky Pipe Band | null |       Scotland | null |    1989 | Pipe and Drum |   band
                           Bellefire | null |        Ireland | null |    null |        Unknown |   band
                      Dia DiCristino | null | United States | null |    null |        Unknown | artist
                           Pat Green | null | United States | null |    null |        Unknown | artist

(5 rows)
cqlsh:music>
```

WHERE DO I LEARN MORE?!

For the Dev: http://www.datastax.com/dev
Docs:http://docs.datastax.com/en/index.html
Planet C*: http://planetcassandra.org/
Driver Guide: http://planetcassandra.org/getting-started-with-apache-cassandra-and-java/
My favorite blogs: http://tobert.github.io/
http://patrickmcfadin.com/
http://rustyrazorblade.com/
https://ahappyknockoutmouse.wordpress.com/author/anukeus/
http://thelastpickle.com/blog/
My favorite C* book: http://www.amazon.com/Cassandra-High-Availability-Robbie-Strickland/dp/1783989122
DataStax Academy: https://academy.datastax.com/
Free Training: http://www.datastax.com/what-we-offer/products-services/training

YOU

CASSANDRA

DANI TRAPHAGEN

HTTP://DATASTAX.COM

DANI.TRAPHAGEN@DATASTAX.COM

@DTRAPEZOID