Lightning-fast analytics with Spark and Cassandra

Tim Vincent

Solution Engineer

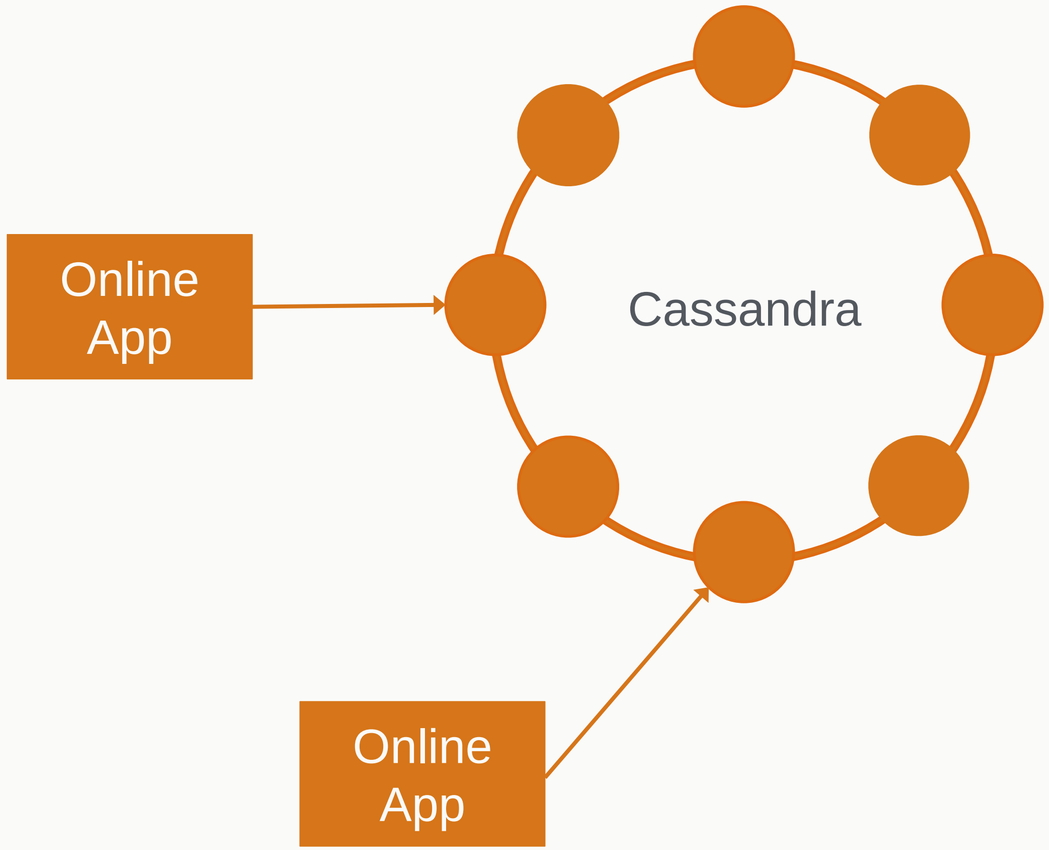
timothy.vincent@datastax.com

@TimV1n

What is Apache Cassandra?

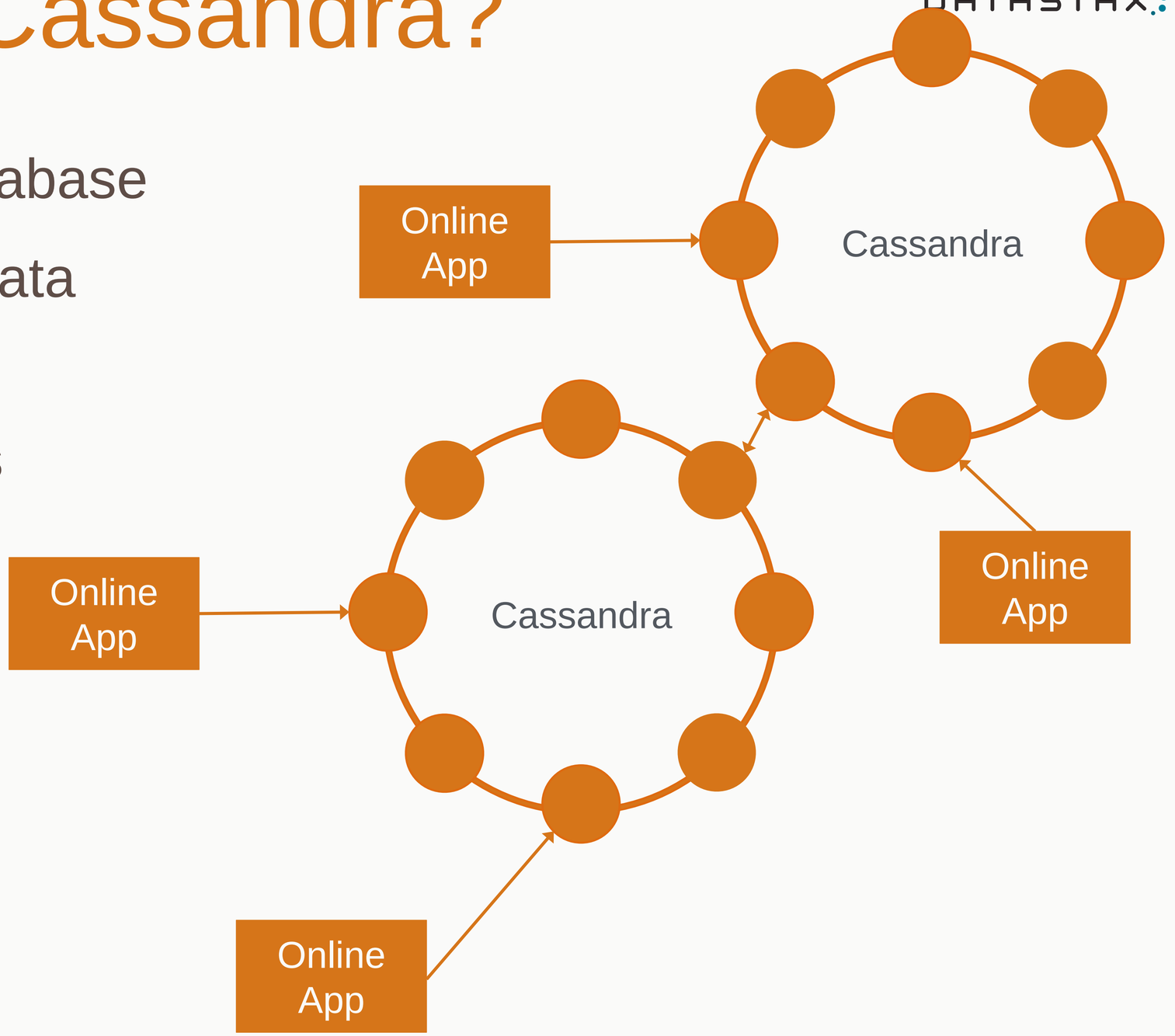
DATASTAX

* Open Source Distributed Database
* Handles Large Amounts of Data
  * At high velocity
* Across Multiple Data Centers
* Disaster Avoidance
* No Single Point of Failure



What is Apache Cassandra?

* Open Source Distributed Database
* Handles Large Amounts of Data
  * At high velocity
* Across Multiple Data Centers
* Disaster Avoidance
* No Single Point of Failure



What is Apache Spark?

DATASTAX

* Apache Project since 2010
* Fast
  * 10x-100x faster than Hadoop MapReduce
  * In-memory storage
  * Single JVM process per node
* Easy
  * Rich Scala, Java and Python APIs
  * 2x-5x less code
  * Interactive shell

API

map

reduce

API

DATASTAX

map
filter
groupBy
sort
union
join
leftOuterJoin
rightOuterJoin

reduce
count
fold
reduceByKey
groupByKey
cogroup
cross
zip

sample
take
first
partitionBy
mapWith
pipe
save
...

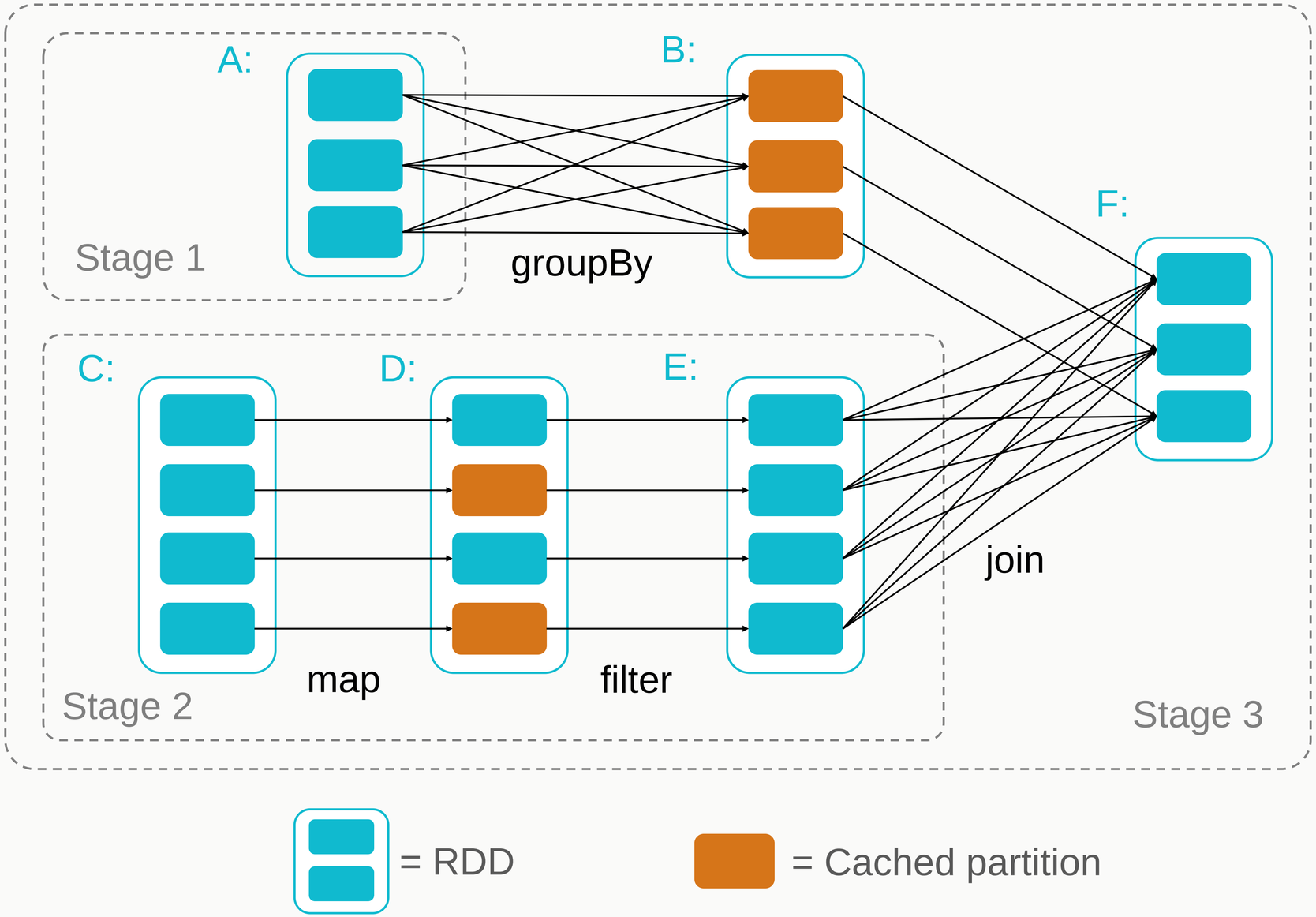
API

DATASTAX

* Resilient Distributed Datasets (RDD)
  * Collections of objects spread across a cluster, stored in RAM or on Disk
  * Built through parallel transformations
  * Automatically rebuilt on failure

* Operations
  * Transformations (e.g. map, filter, groupBy)
  * Actions (e.g. count, collect, save)

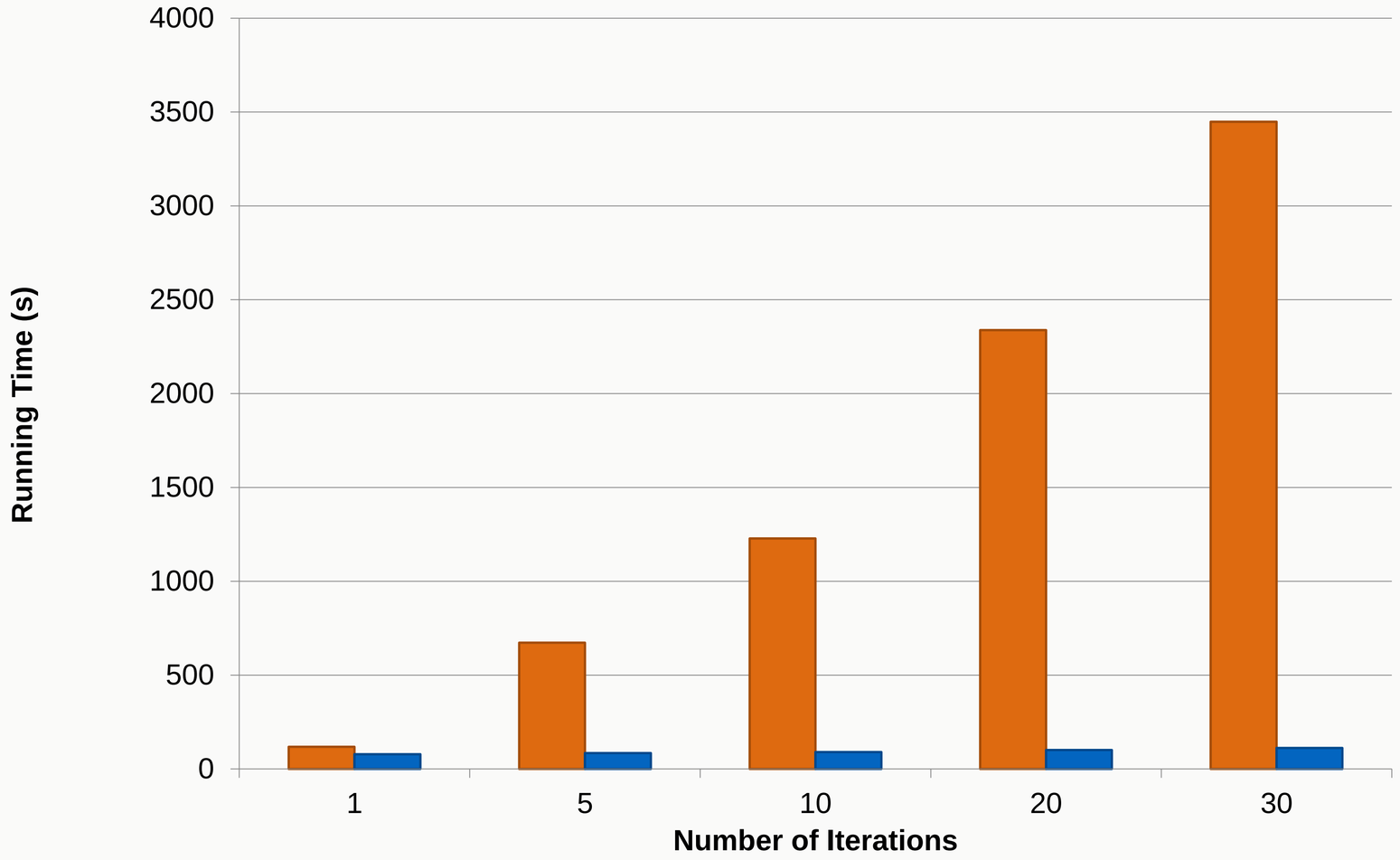
Operator Graph: Optimization and Fault Tolerance



Fast

DATASTAX

* Logistic Regression Performance



110 sec / iteration

Hadoop Spark

first iteration 80 sec
further iterations 1 sec

Why Spark on Cassandra?

DATASTAX

* Data model independent queries
* Cross-table operations (JOIN, UNION, etc.)
* Complex analytics (e.g. machine learning)
* Data transformation, aggregation, etc.
* Stream processing

How to Spark on Cassandra?

DATASTAX

* DataStax Cassandra Spark driver
  * Open source: https://github.com/datastax/cassandra-driver-spark
* Compatible with
  * Spark 0.9+
  * Cassandra 2.0+
  * DataStax Enterprise 4.5+

Cassandra Spark Driver

DATASTAX

* Cassandra tables exposed as Spark RDDs
* Read from and write to Cassandra
* Mapping of C* tables and rows to Scala objects
* All Cassandra types supported and converted to Scala types
* Server side data selection
* Spark Streaming support
* Scala and Java support

Connecting to Cassandra

DATASTAX

```
// Import Cassandra-specific functions on SparkContext and RDD objects
import com.datastax.driver.spark._


// Spark connection options
val conf = new SparkConf(true)
        .setMaster("spark://192.168.123.10:7077")
        .setAppName("cassandra-demo")
        .set("cassandra.connection.host", "192.168.123.10") // initial contact
        .set("cassandra.username", "cassandra")
        .set("cassandra.password", "cassandra")

val sc = new SparkContext(conf)
```

Accessing Data

DATASTAX

```
CREATE TABLE test.words (word text PRIMARY KEY, count int);

INSERT INTO test.words (word, count) VALUES ('bar', 30);
INSERT INTO test.words (word, count) VALUES ('foo', 20);
```

* Accessing table above as RDD:

```
// Use table as RDD
val rdd = sc.cassandraTable("test", "words")
// rdd: CassandraRDD[CassandraRow] = CassandraRDD[0]

rdd.toArray.foreach(println)
// CassandraRow[word: bar, count: 30]
// CassandraRow[word: foo, count: 20]

rdd.columnNames    // Stream(word, count)
rdd.size           // 2

val firstRow = rdd.first  // firstRow: CassandraRow = CassandraRow[word: bar, count: 30]
firstRow.getInt("count")  // Int = 30
```

Saving Data

DATASTAX

```
val newRdd = sc.parallelize(Seq(("cat", 40), ("fox", 50)))
// newRdd: org.apache.spark.rdd.RDD[(String, Int)] = ParallelCollectionRDD[2]

newRdd.saveToCassandra("test", "words", Seq("word", "count"))
```

* RDD above saved to Cassandra:

```
SELECT * FROM test.words;

 word | count
------+-------
  bar |    30
  foo |    20
  cat |    40
  fox |    50

(4 rows)
```

Type Mapping

DATASTAX

CQL Type	Scala Type
ascii	String
bigint	Long
boolean	Boolean
counter	Long
decimal	BigDecimal, java.math.BigDecimal
double	Double
float	Float
inet	java.net.InetAddress
int	Int
list	Vector, List, Iterable, Seq, IndexedSeq, java.util.List
map	Map, TreeMap, java.util.HashMap
set	Set, TreeSet, java.util.HashSet
text, varchar	String
timestamp	Long, java.util.Date, java.sql.Date, org.joda.time.DateTime
timeuuid	java.util.UUID
uuid	java.util.UUID
varint	BigInt, java.math.BigInteger
*nullable values	Option

Mapping Rows to Objects

DATASTAX

* Mapping rows to Scala Case Classes
* CQL underscore case column mapped to Scala camel case property

```
CREATE TABLE test.cars (
  id text PRIMARY KEY,
  model text,
  fuel_type text,
  year int
);
```

```
case class Vehicle(
  id: String,
  model: String,
  fuelType: String,
  year: Int
)

sc.cassandraTable[Vehicle]("test", "cars").toArray
//Array(Vehicle(KF334L, Ford Mondeo, Petrol, 2009),
//      Vehicle(MT8787, Hyundai x35, Diesel, 2011)
```

Server Side Data Selection

DATASTAX

* Reduce the amount of data transferred

* Selecting columns

```
sc.cassandraTable("test", "users").select("username").toArray.foreach(println)
// CassandraRow{username: john}
// CassandraRow{username: tom}
```

* Selecting rows (by clustering columns and/or secondary indexes)

```
sc.cassandraTable("test", "cars").select("model").where("color = ?", "black").toArray.foreach(println)
// CassandraRow{model: Ford Mondeo}
// CassandraRow{model: Hyundai x35}
```

Spark SQL vs Shark

DATASTAX

Compatible

Shark or Spark SQL

Streaming

ML

Graph

Spark *(General execution engine)*

Cassandra

SparkSQL / Shark

DATASTAX

* SQL query engine on top of Spark
* Hive compatible (JDBC, UDFs, types, metadata, etc.)
* Support for in-memory processing

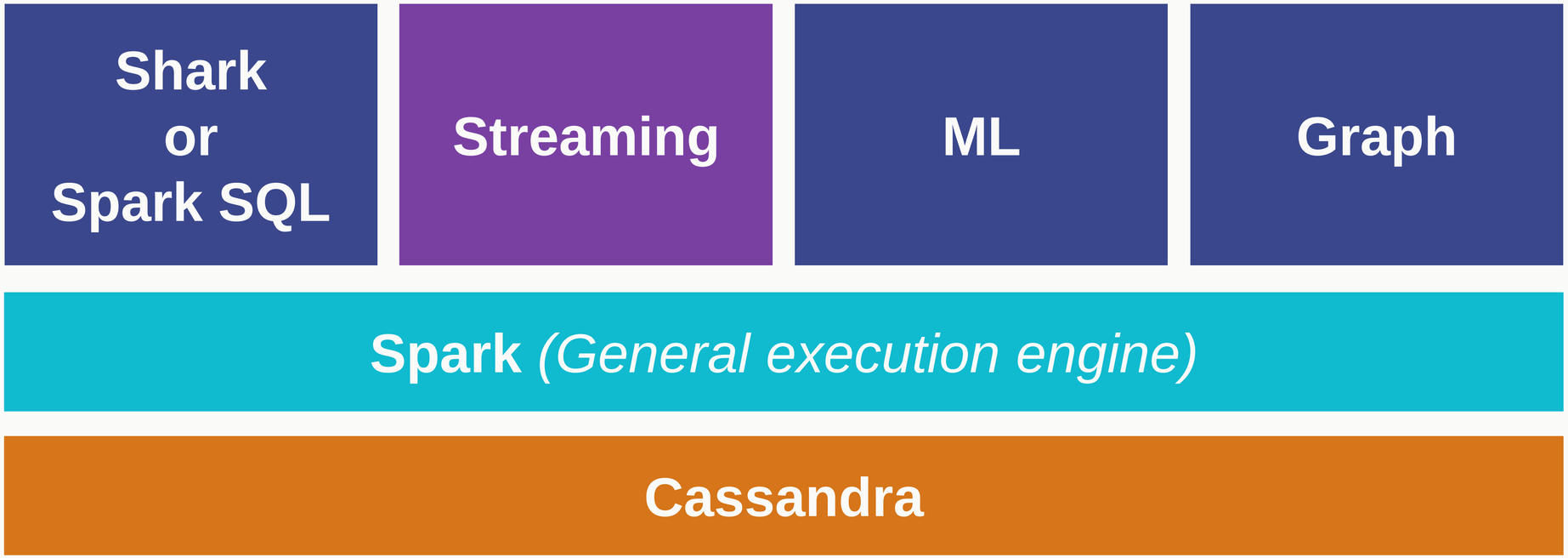
Shark In-memory Tables

DATASTAX

```
CREATE TABLE CachedStocks TBLPROPERTIES ("shark.cache" = "true")
             AS SELECT * from PortfolioDemo.Stocks WHERE value > 95.0;
OK
Time taken: 1.215 seconds


SELECT * FROM CachedStocks;
OK
MQT price 97.9270442241818
SII price 99.69238346610474
.
. (123 additional prices)
.
PBG price 96.09162963505352
Time taken: 0.569 seconds
```

Spark Streaming

DATASTAX



Spark Streaming

DATASTAX

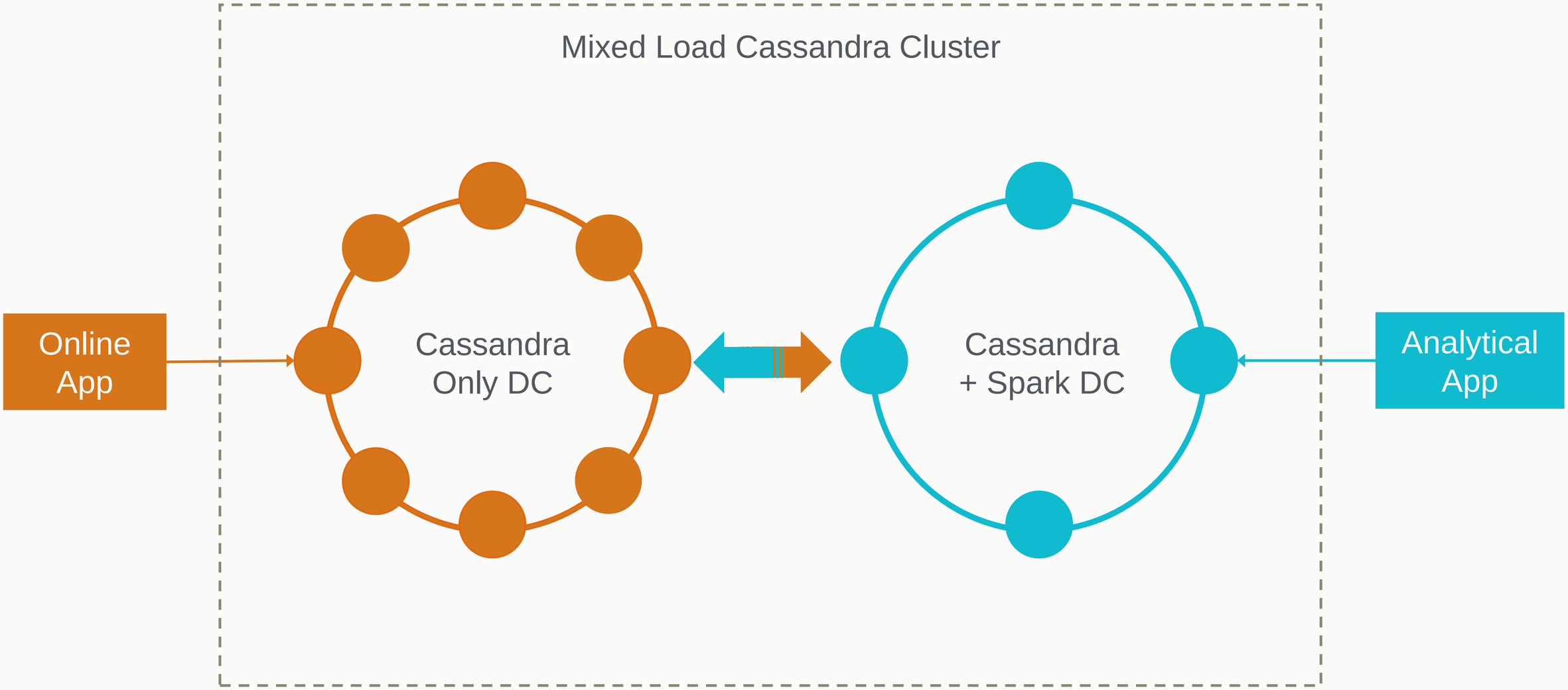
* Micro batching
* Each batch represented as RDD
* Fault tolerant
* Exactly-once processing
* Unified stream and batch processing framework

Streaming Example

DATASTAX

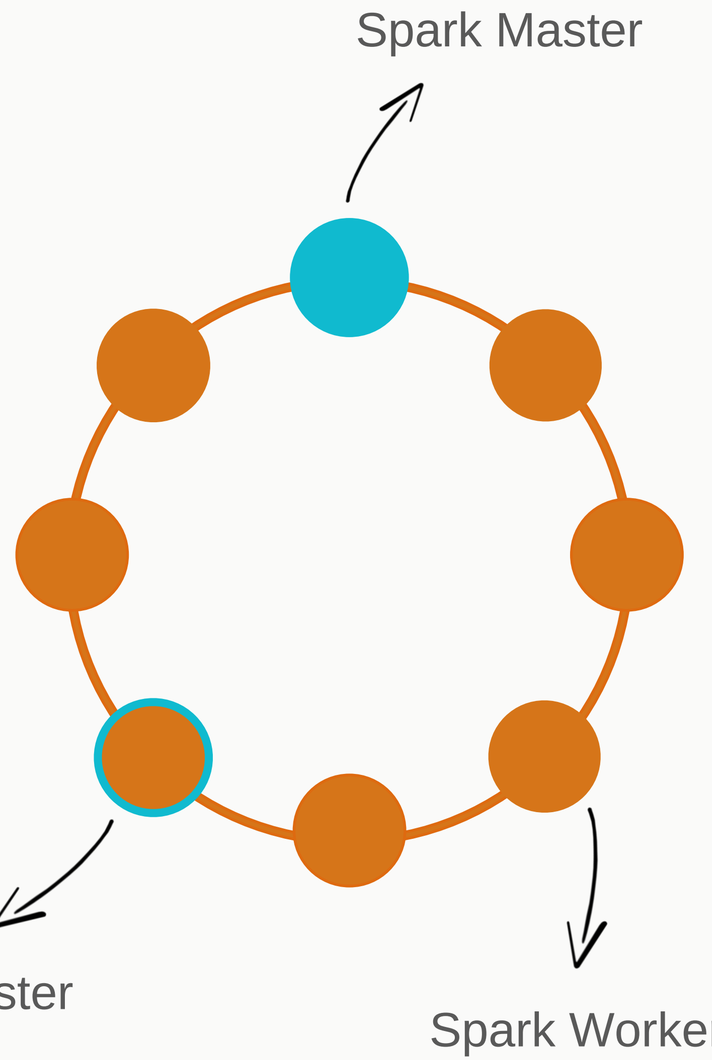
```scala
import com.datastax.spark.connector._

// Spark connection options
val conf = new SparkConf(true)...
val sc = new SparkContext(conf)

// streaming with 1 seconds batch window
val ssc = new StreamingContext(sc, Seconds(1))

// stream input
val lines = ssc.socketTextStream(serverIP, serverPort)

// count words
val wordCounts = lines.flatMap(_.split(" ")).map(word => (word, 1)).reduceByKey(_ + _)

// stream output
wordCounts.saveToCassandra("test", "words")

// start processing
ssc.start()
ssc.awaitTermination()
```

Analytics Workload Isolation



Analytics High Availability

* Spark Workers run on all Cassandra nodes
* Workers are resilient by default
* First Spark node promoted as Spark Master
* Standby Master promoted on failure
* Master HA available in DataStax Enterprise



Doing it Already

technicolor

virdata

The Weather Channel®

- http://planetcassandra.org/blog/fast-spark-queries-on-in-memory-datasets
- http://www.virdata.com/wp-content/uploads/Spark-at-Virdata.pdf
- http://planetcassandra.org/blog/the-new-analytics-toolbox-with-apache-spark-going-beyond-hadoop/

Useful Resources

DATASTAX

* Get the Apache Spark + Cassandra OSS Connector at GitHub:
  * https://github.com/datastax/spark-cassandra-connector
* View more information about Apache Spark + Cassandra integration on Planet Cassandra:
  * http://planetcassandra.org/getting-started-with-apache-spark-and-cassandra/

Questions?