Spark and Cassandra:
2 Fast, 2 Furious

Russell Spitzer
DataStax Inc.

SPARK SUMMIT 2016
DATA SCIENCE AND ENGINEERING AT SCALE
JUNE 6-8, 2016 SAN FRANCISCO

Russell, ostensibly a software engineer

- Did a Ph.D in bioinformatics at some point
- Written a great deal of automation and testing framework code
- Now develops for Datastax on the Analytics Team
- Focuses a lot on the **Datastax OSS Spark Cassandra Connector**

Datastax Spark Cassandra Connector

Let Spark Interact with your Cassandra Data!

Compatible with Spark 1.6 + Cassandra 3.0

- Bulk writing to Cassandra
- Distributed full table scans
- Optimized direct joins with Cassandra
- Secondary index pushdown
- Connection and prepared statement pools

http://spark-packages.org/package/datastax/spark-cassandra-connector

https://github.com/datastax/spark-cassandra-connector

http://spark-packages.org/package/TargetHolding/pyspark-cassandra

Cassandra is essentially a hybrid between a key-value and a column-oriented (or tabular) database management system. Its data model is a partitioned row store with tunable consistency*

Let's break that down

1. What is a C* Partition and Row
2. How does C* Place Partitions

*https://en.wikipedia.org/wiki/Apache_Cassandra

CQL looks a lot like SQL

```
CREATE TABLE tracker (
  vehicle_id  uuid,
  ts timestamp,
  x double,
  y double,
  PRIMARY KEY (vehicle_id, ts))
```

INSERTS look almost Identical

```
CREATE TABLE tracker (
  vehicle_id  uuid,
  ts timestamp,
  x double,
  y double,
  PRIMARY KEY (vehicle_id, ts))
```


```
INSERT INTO tracker (vehicle_id, ts, x, y) Values ( 1, 0, 0, 1)
```

Cassandra Data is stored in Partitions

```
CREATE TABLE tracker (
  vehicle_id  uuid,
  ts timestamp,
  x double,
  y double,
  PRIMARY KEY (vehicle_id, ts))
```

vehicle_id
1

ts: 0

x: 0 | y: 1

```
INSERT INTO tracker (vehicle_id, ts, x, y) Values ( 1, 0, 0, 1)
```

C* Partitions Store Many Rows

```
CREATE TABLE tracker (
  vehicle_id  uuid,
  ts timestamp,
  x double,
  y double,
  PRIMARY KEY (vehicle_id, ts))
```

vehicle_id
1

ts: 0		ts: 1		ts: 2		ts: 3	
x: 0	y: 1	x: -1	y:0	x: 0	y: -1	x: 1	y: 0

C* Partitions Store Many Rows

```
CREATE TABLE tracker (
  vehicle_id  uuid,
  ts timestamp,
  x double,
  y double,
  PRIMARY KEY (vehicle_id, ts))
```

C* Partitions Store Many Rows

```
CREATE TABLE tracker (
  vehicle_id  uuid,
  ts timestamp,
  x double,
  y double,
  PRIMARY KEY (vehicle_id, ts))
```

vehicle_id
1

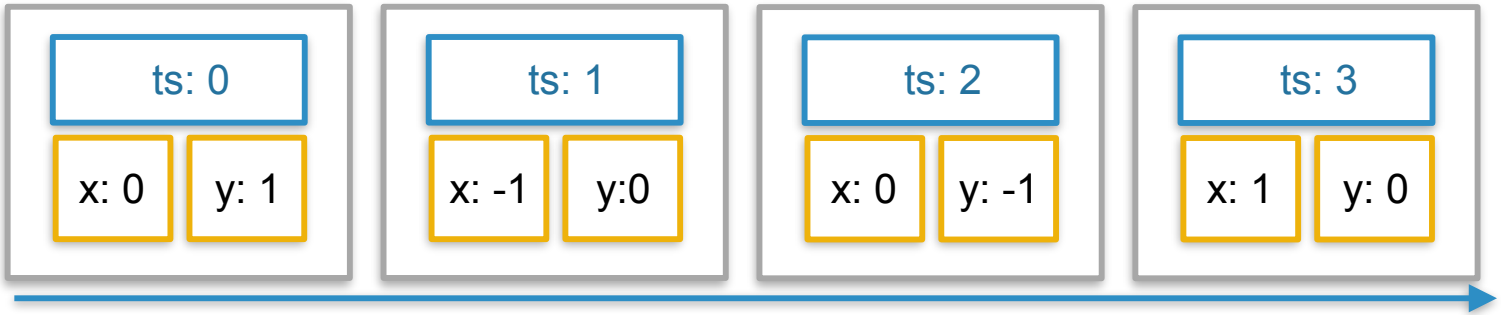
ts: 0	ts: 1	ts: 2	ts: 3
x: 0, y: 1	x: -1, y:0	x: 0, y: -1	x: 1, y: 0

Row

Within a partition there is ordering based on the Clustering Keys

```
CREATE TABLE tracker (
  vehicle_id  uuid,
  ts timestamp,
  x double,
  y double,
  PRIMARY KEY (vehicle_id, ts))
```

vehicle_id
1

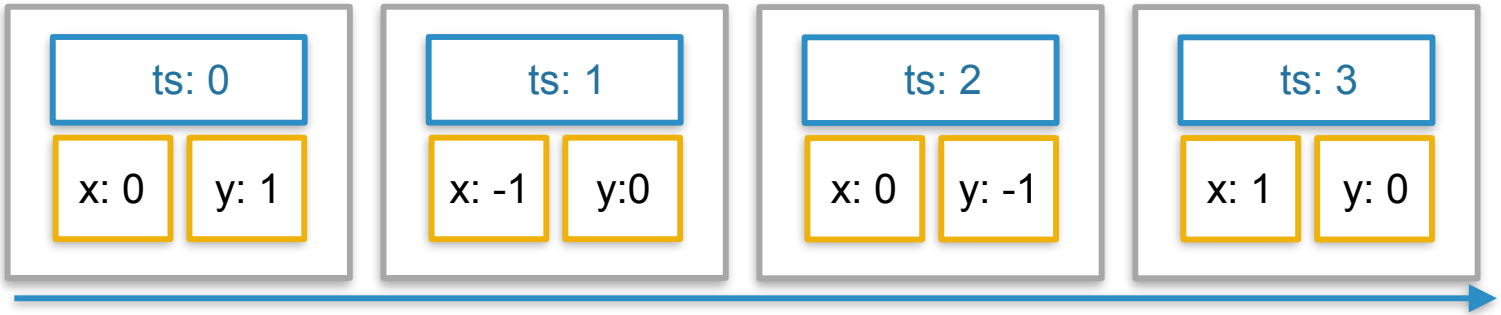


Ordered by Clustering Key

Slices within a Partition are Very Easy

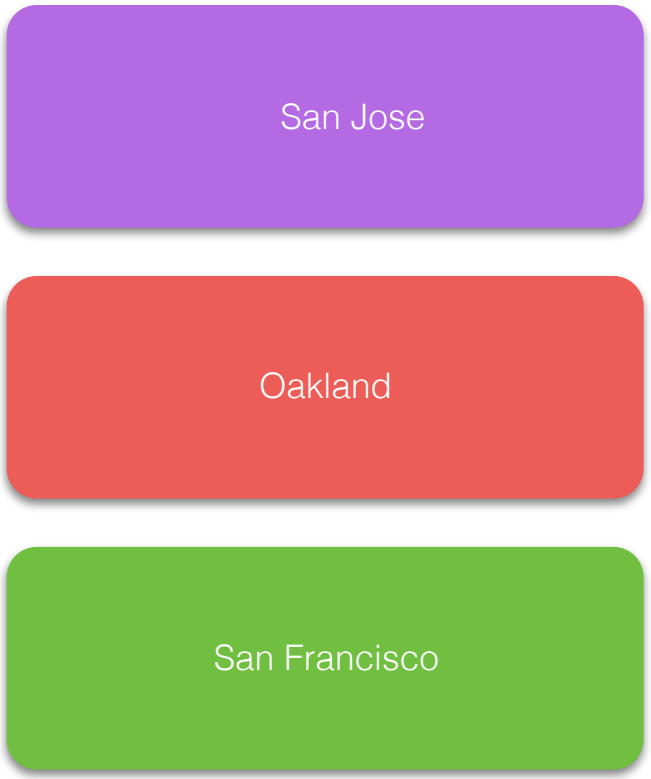
```
CREATE TABLE tracker (
  vehicle_id  uuid,
  ts timestamp,
  x double,
  y double,
  PRIMARY KEY (vehicle_id, ts))
```

vehicle_id
1

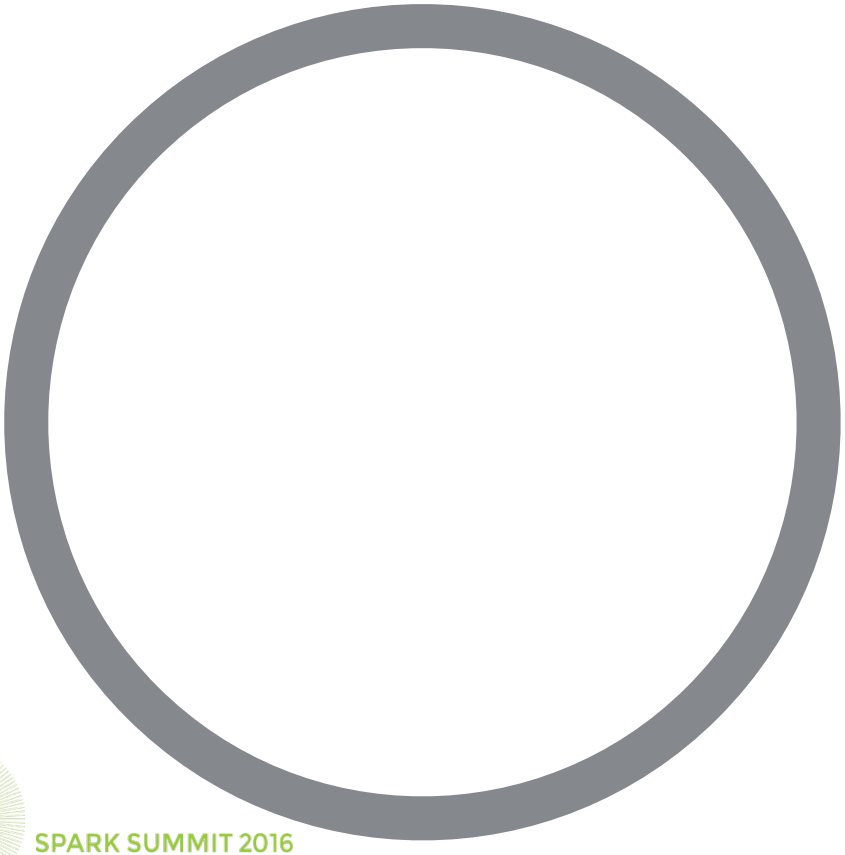


Ordered by Clustering Key

Cassandra is a Distributed Fault-Tolerant Database



Data is located on a Token Range

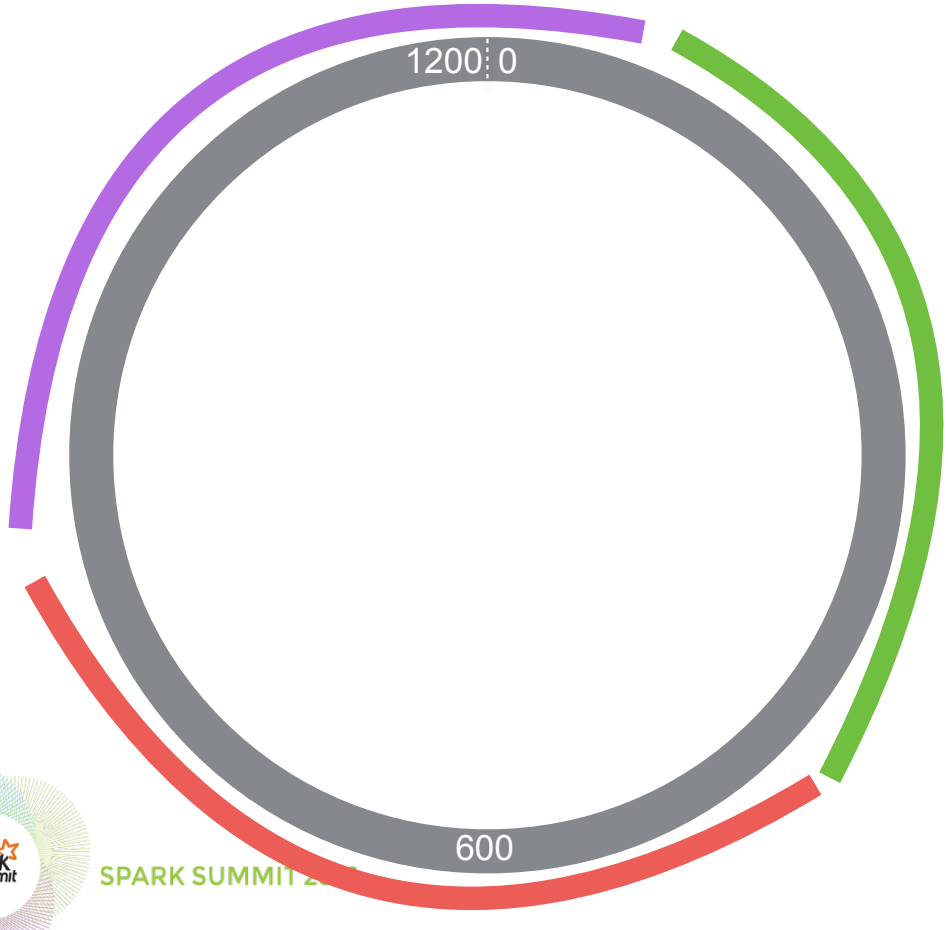


San Jose

Oakland

San Francisco

Data is located on a Token Range

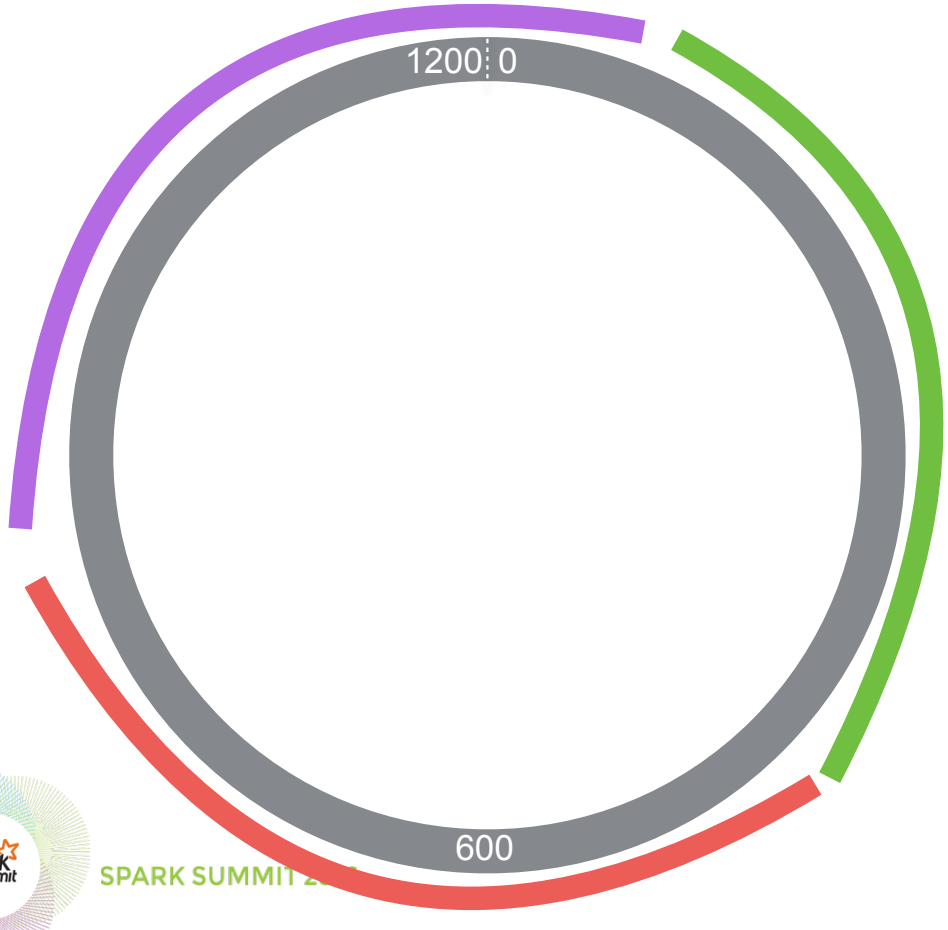


San Jose

Oakland

San Francisco

The Partition Key of a Row is Hashed to Determine it's Token

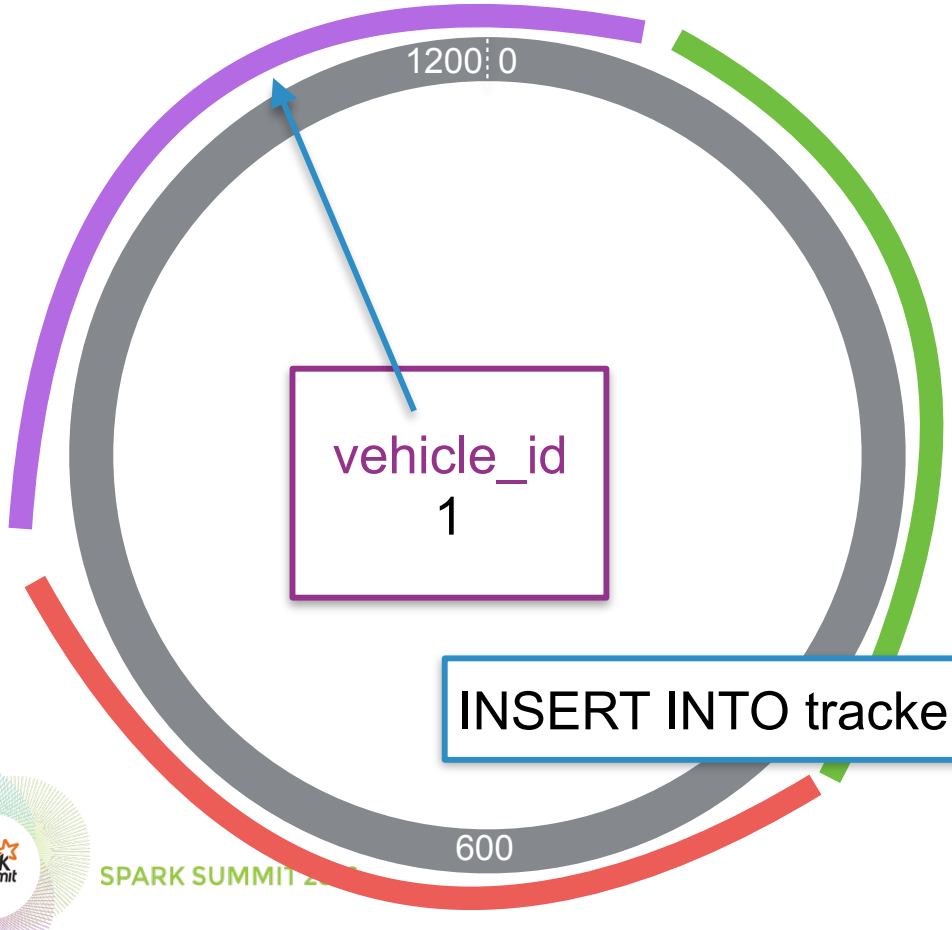


San Jose

Oakland

San Francisco

The Partition Key of a Row is Hashed to Determine it's Token

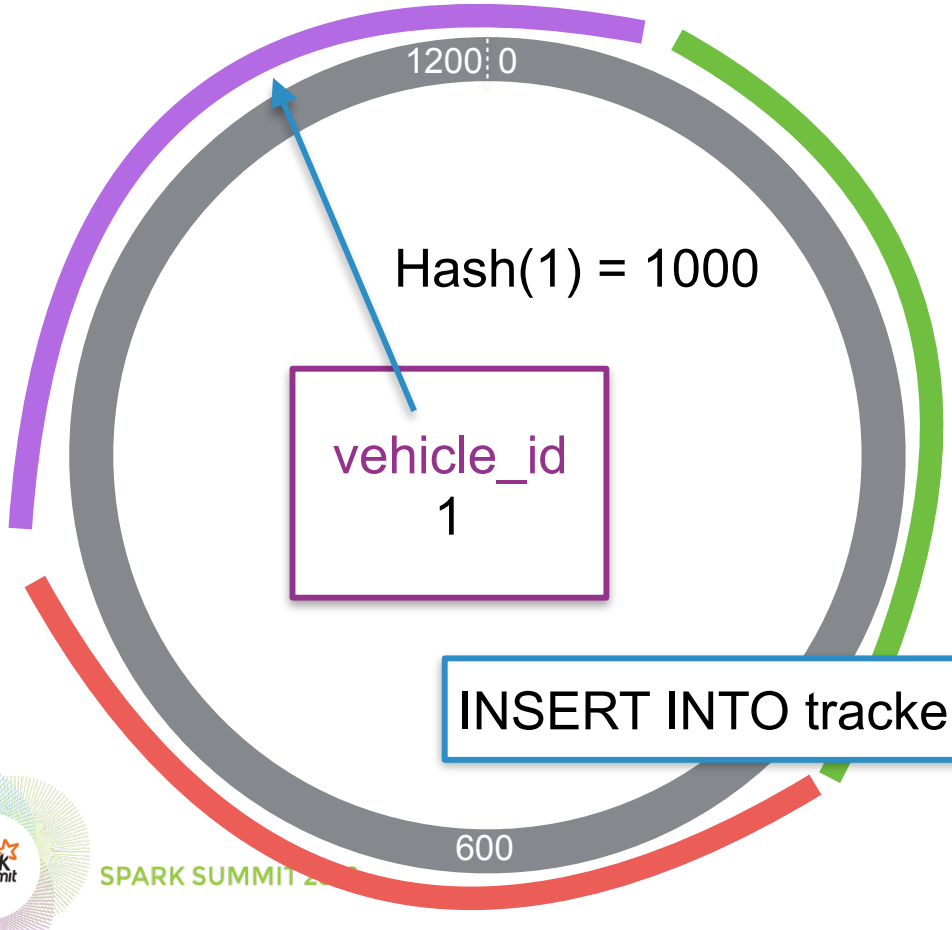


San Jose

Oakland

INSERT INTO tracker (vehicle_id, ts, x, y) Values (**1**, 0, 0, 1)

The Partition Key of a Row is Hashed to Determine it's Token



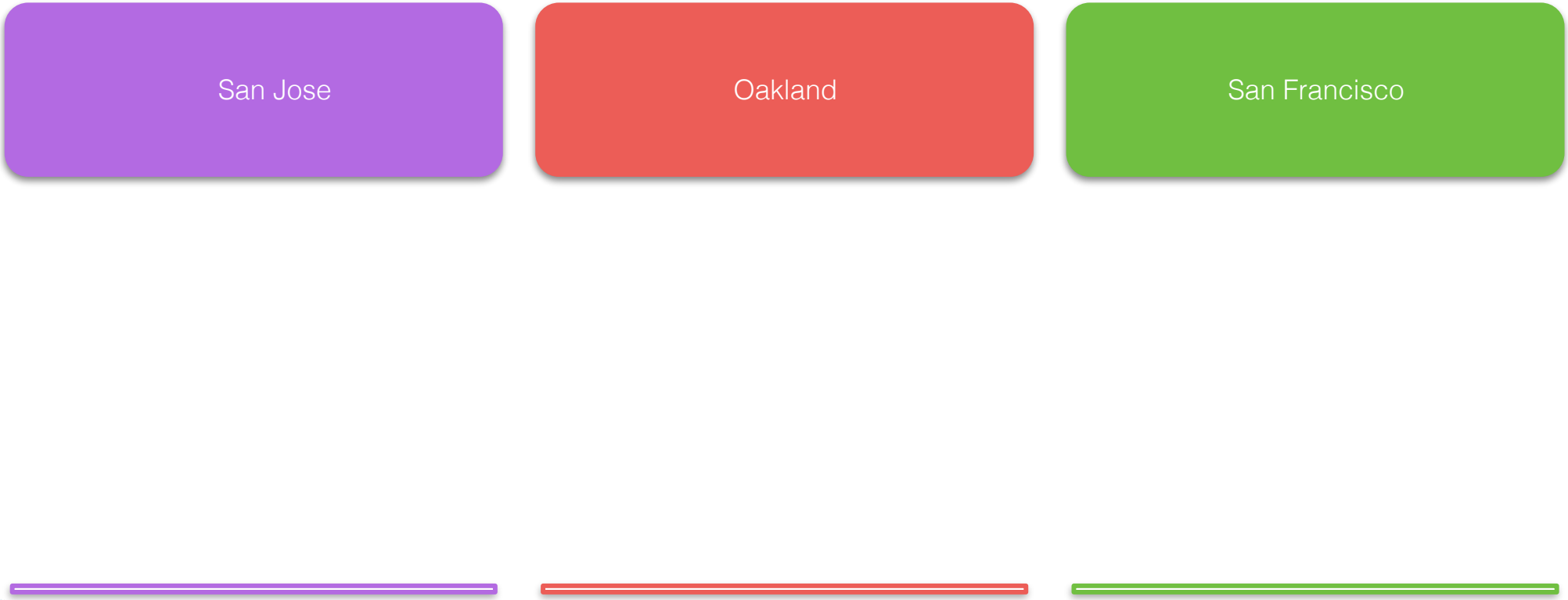
San Jose

Oakland

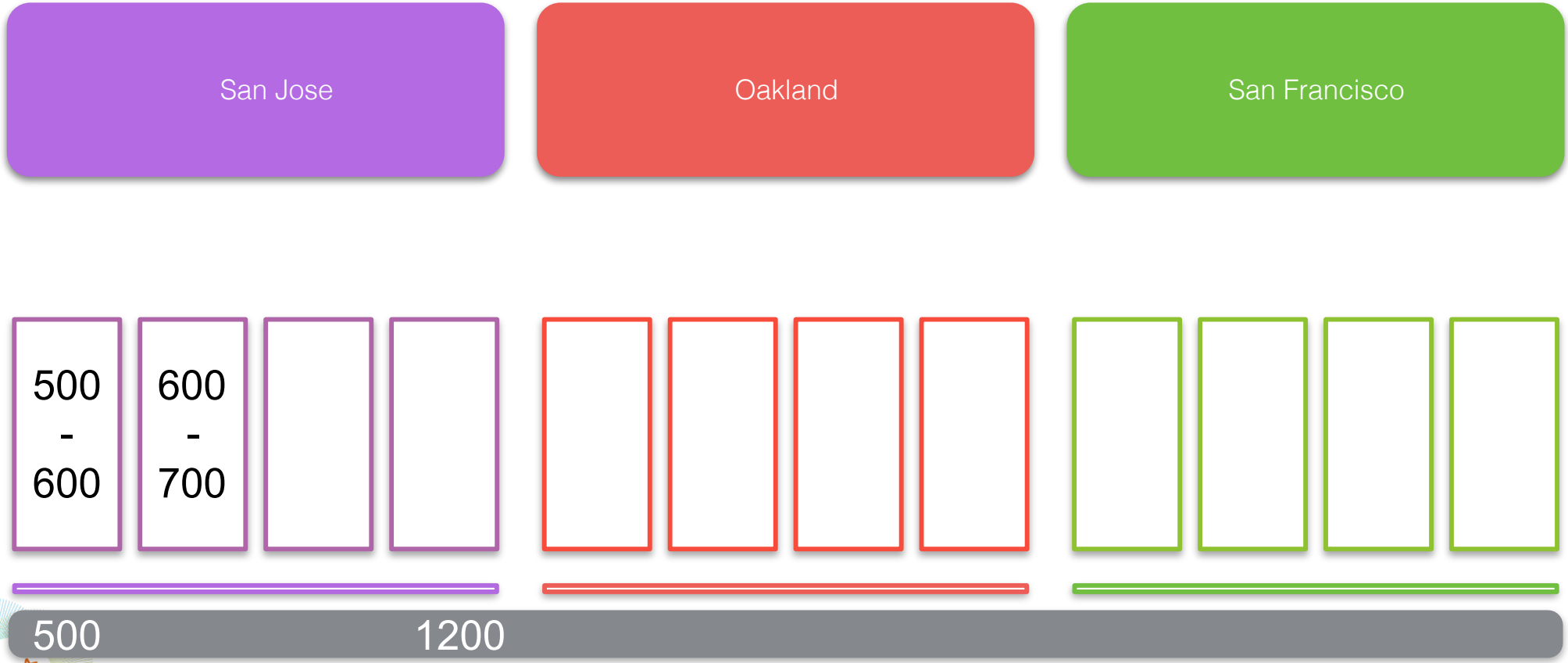
INSERT INTO tracker (vehicle_id, ts, x, y) Values (**1**, 0, 0, 1)

How The Spark Cassandra Connector Reads

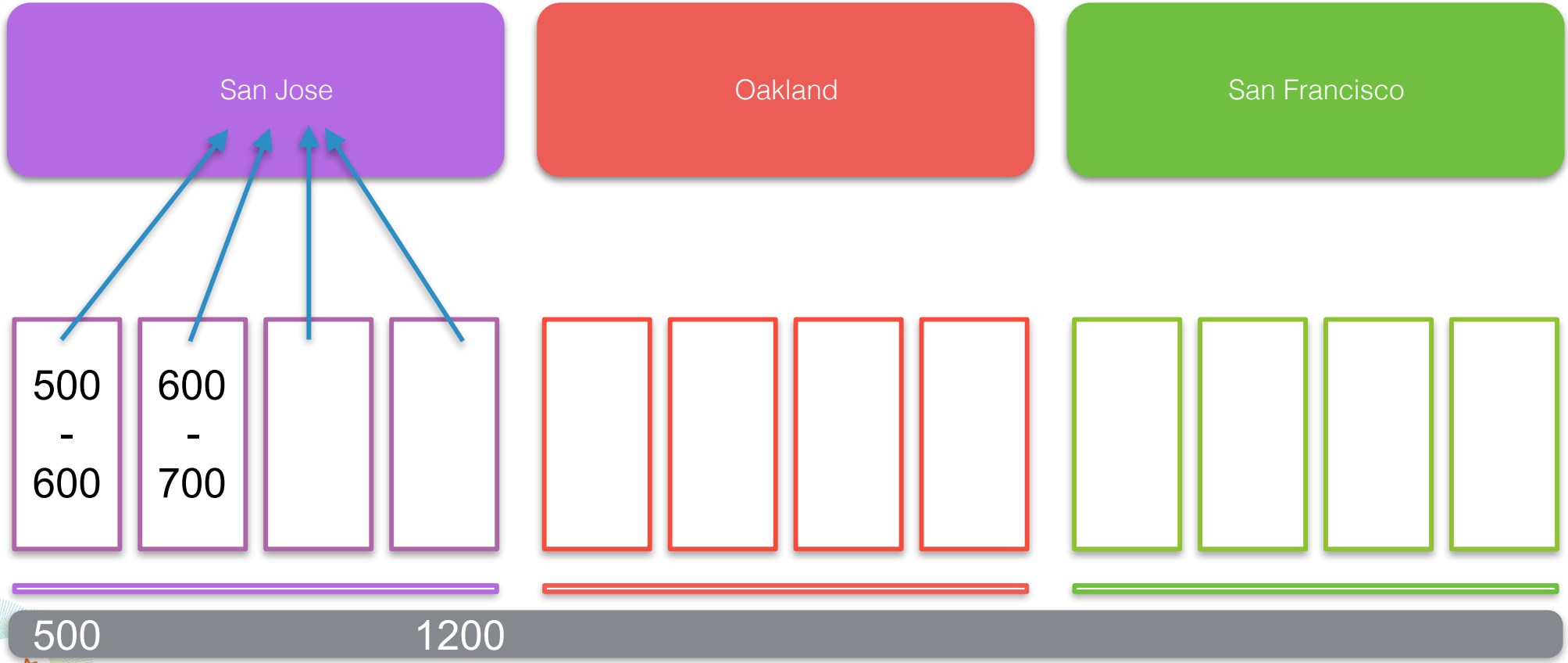
How The Spark Cassandra Connector Reads



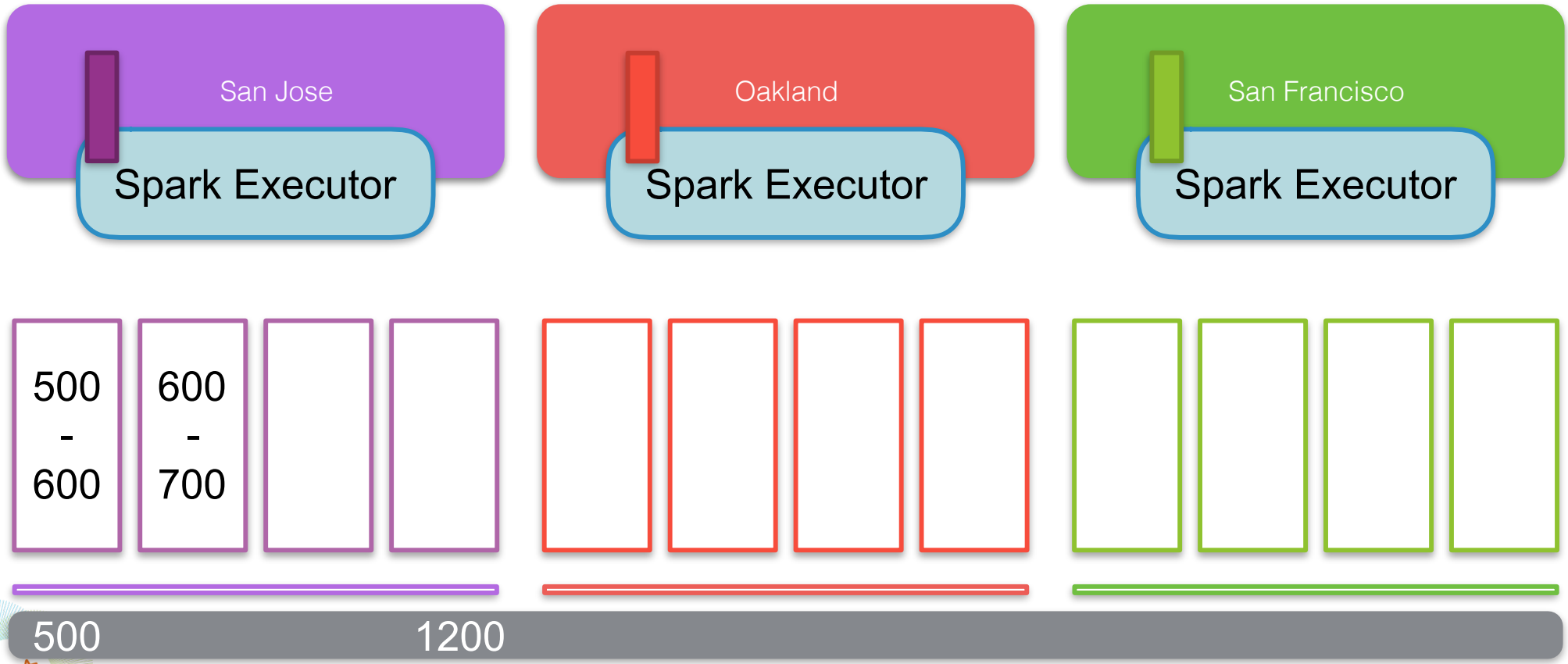
Spark Partitions are Built From the Token Range



Each Partition Can be Drawn Locally from at Least One Node



Each Partition Can be Drawn Locally from at Least One Node



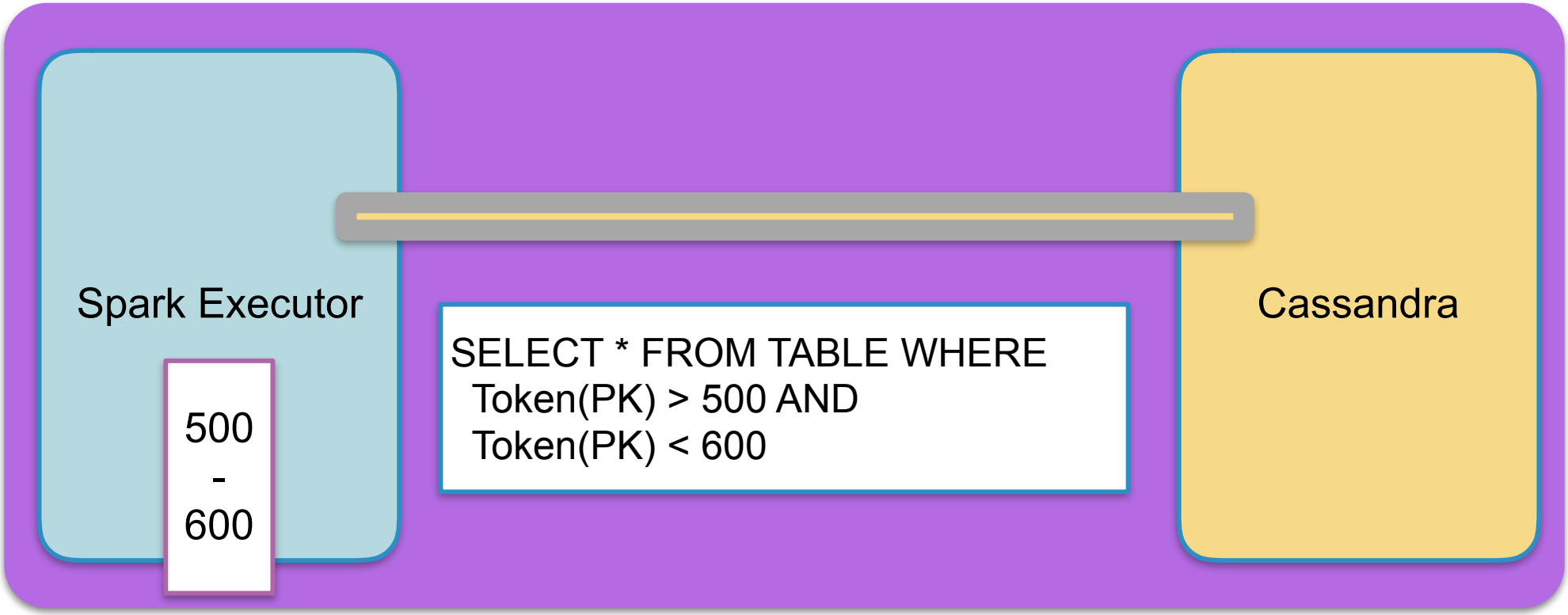
No Need to Leave the Node For Data!

Data is Retrieved using the Datastax Java Driver

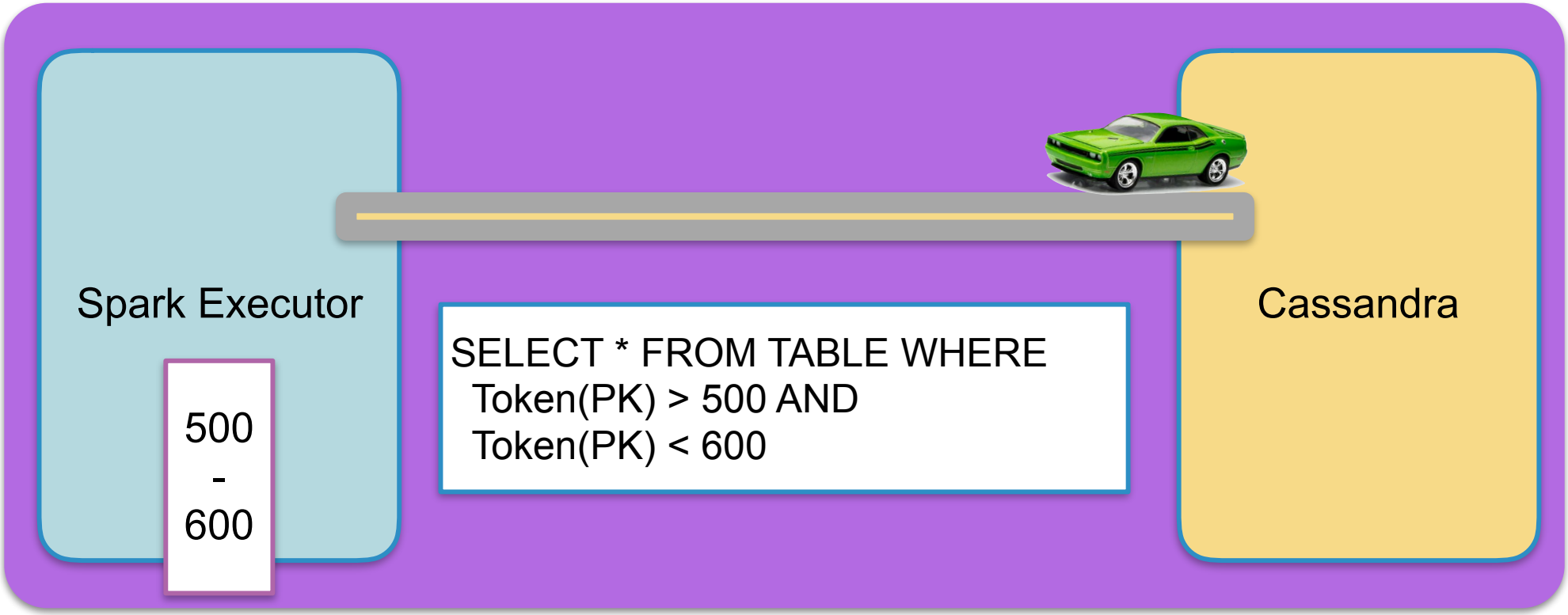
A Connection is Established

A Query is Prepared with Token Bounds

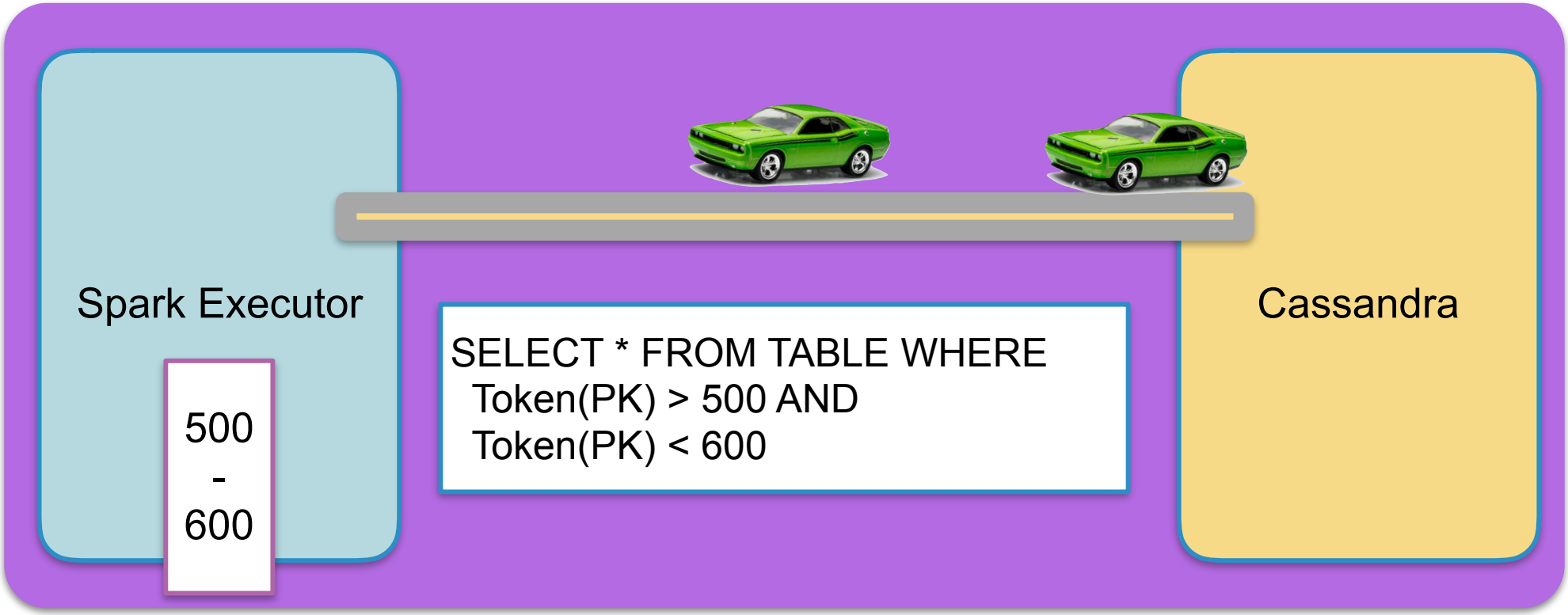
The Spark Partitions Bounds are Placed in the Query



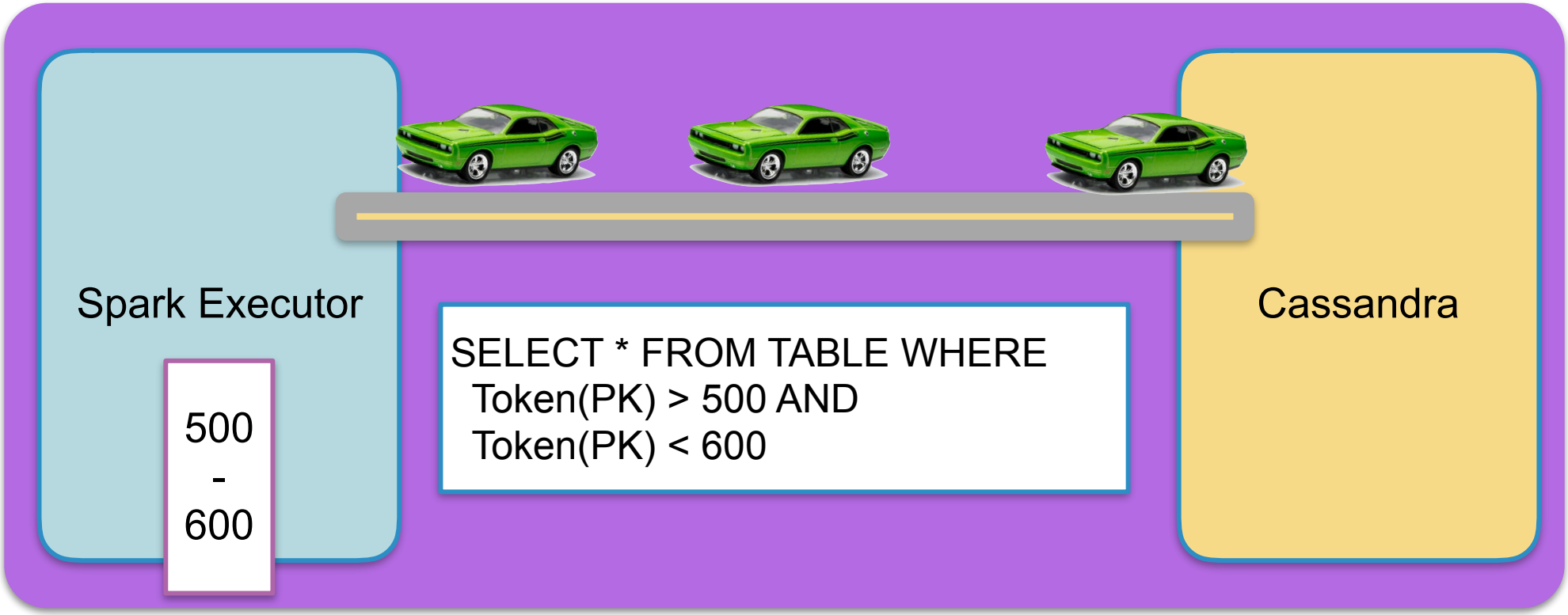
Paged a number of rows at a Time



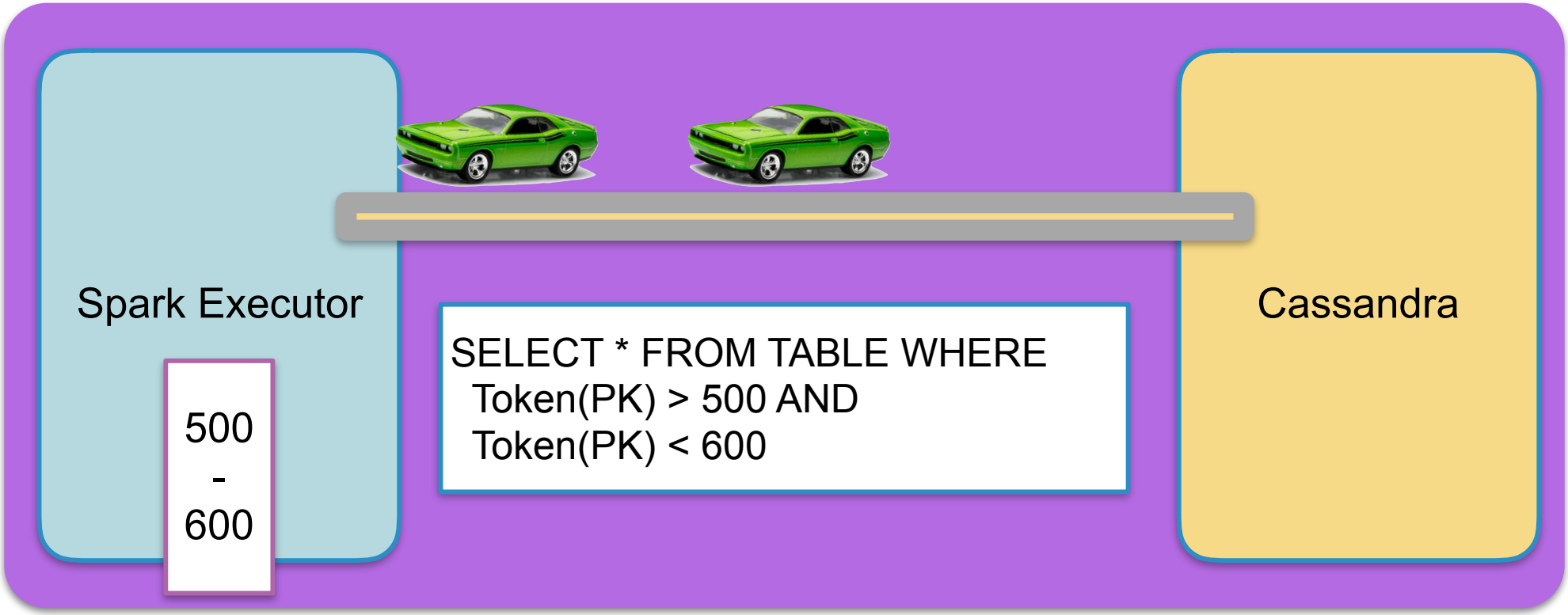
Data is Paged



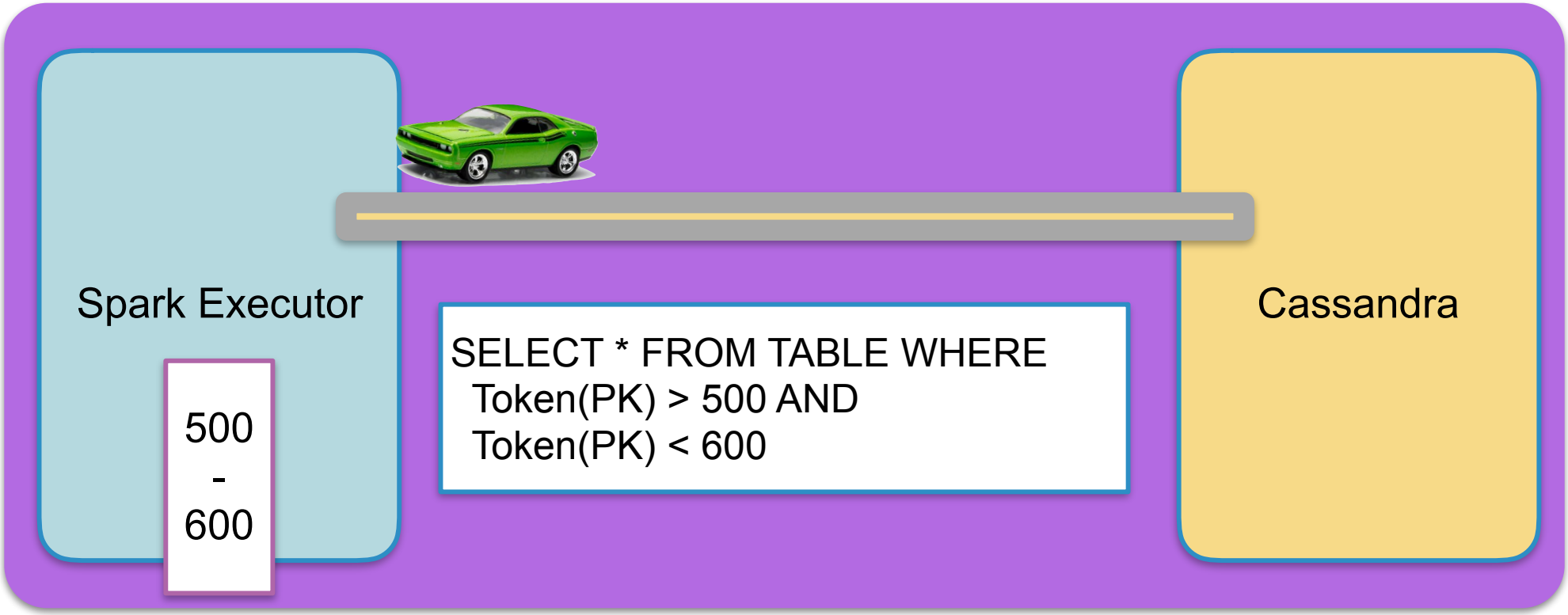
Data is Paged



Data is Paged



Data is Paged



How we write to Cassandra

- Data is written via the Datastax Java Driver
- Data is grouped based on Partition Key (configurable)
- Batches are written

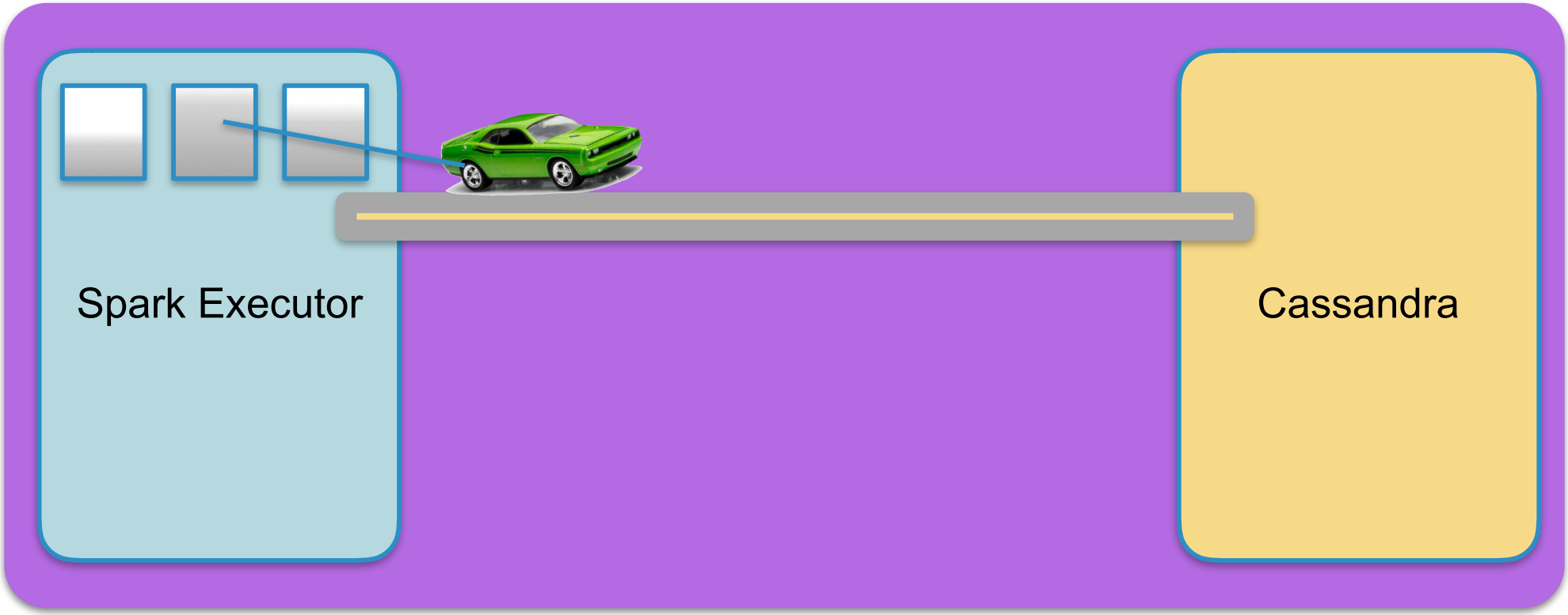
Data is Written using the Datastax Java Driver

A Connection is Established

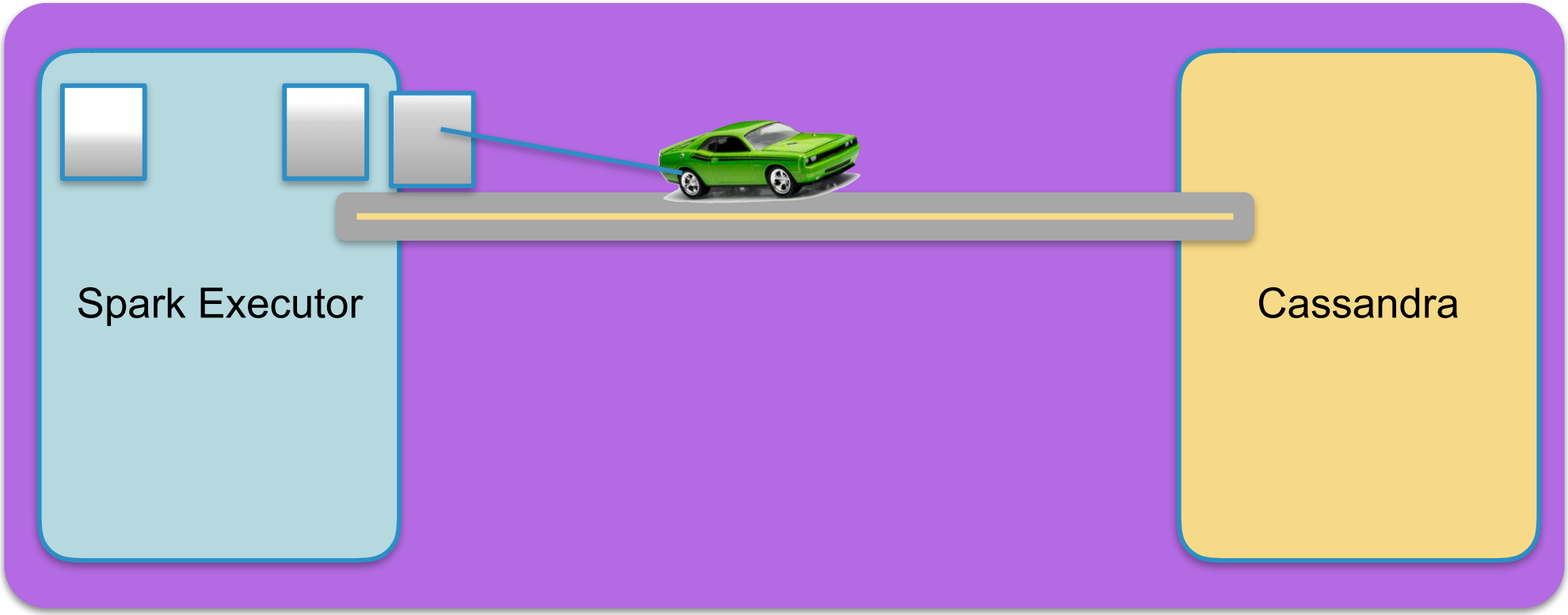
Data is Grouped on Key

Data is Grouped on Key

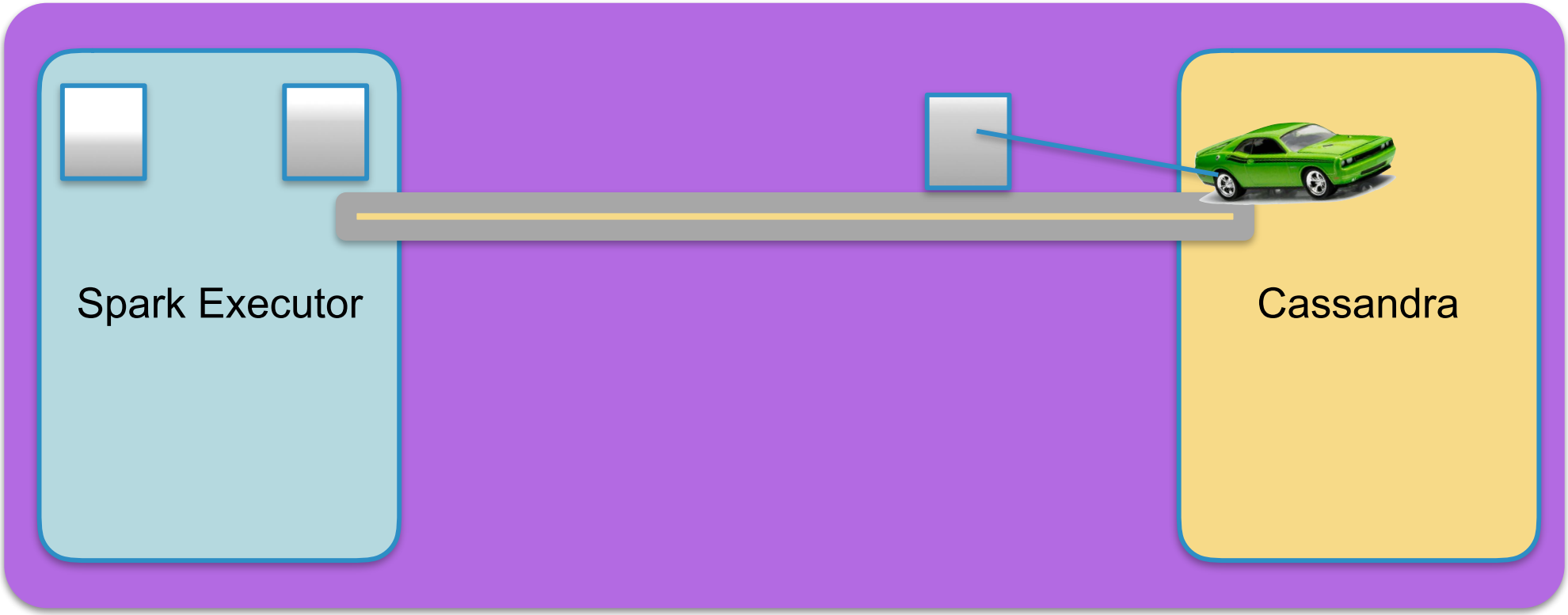
Once a Batch is Full or There are Too Many Batches The Largest Batch is Executed



Once a Batch is Full or There are Too Many Batches The Largest Batch is Executed



Once a Batch is Full or There are Too Many Batches The Largest Batch is Executed



Once a Batch is Full or There are Too Many Batches The Largest Batch is Executed

Most Common Features

- RDD APIs
  - cassandraTable
  - saveToCassandra
  - repartitionByCassandraTable
  - joinWithCassandraTable
- DF API
  - Datasource

Full Table Scans, Making an RDD out of a Table

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

sc.cassandraTable(KeyspaceName, TableName)
```

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

sc.cassandraTable[MyClass](KeyspaceName, TableName)
```

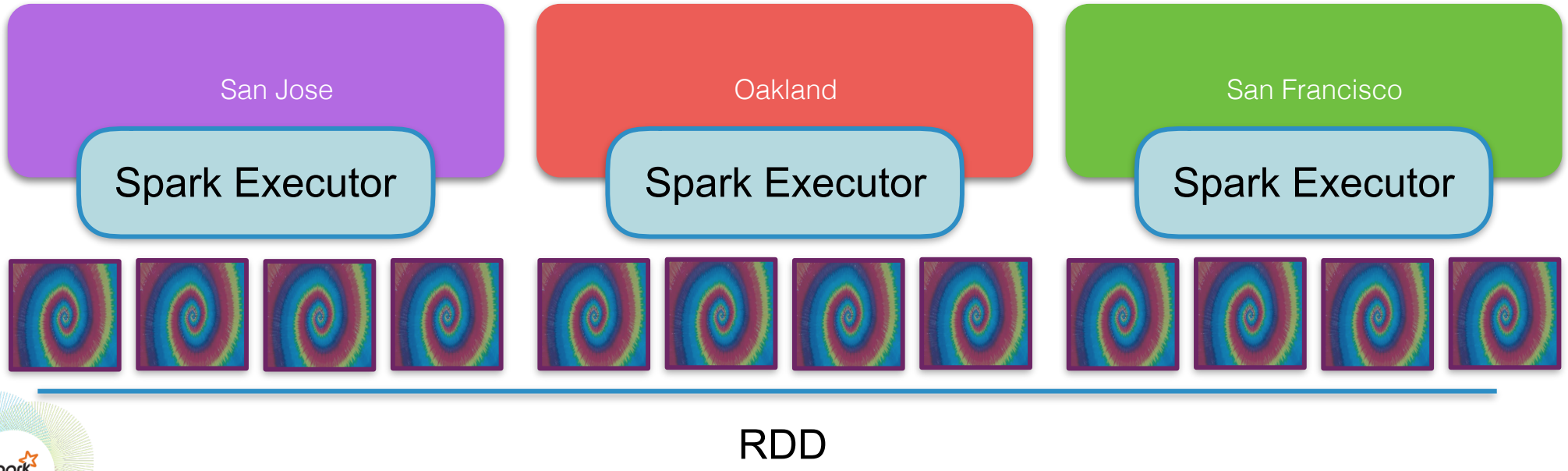
Pushing Down CQL to Cassandra

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

sc.cassandraTable[MyClass](KeyspaceName,
TableName).select("vehicle_id").where("ts > 10")
```

```
SELECT "vehicle_id" FROM TABLE
WHERE
  Token(PK) > 500 AND
  Token(PK) < 600 AND
   ts > 10
```

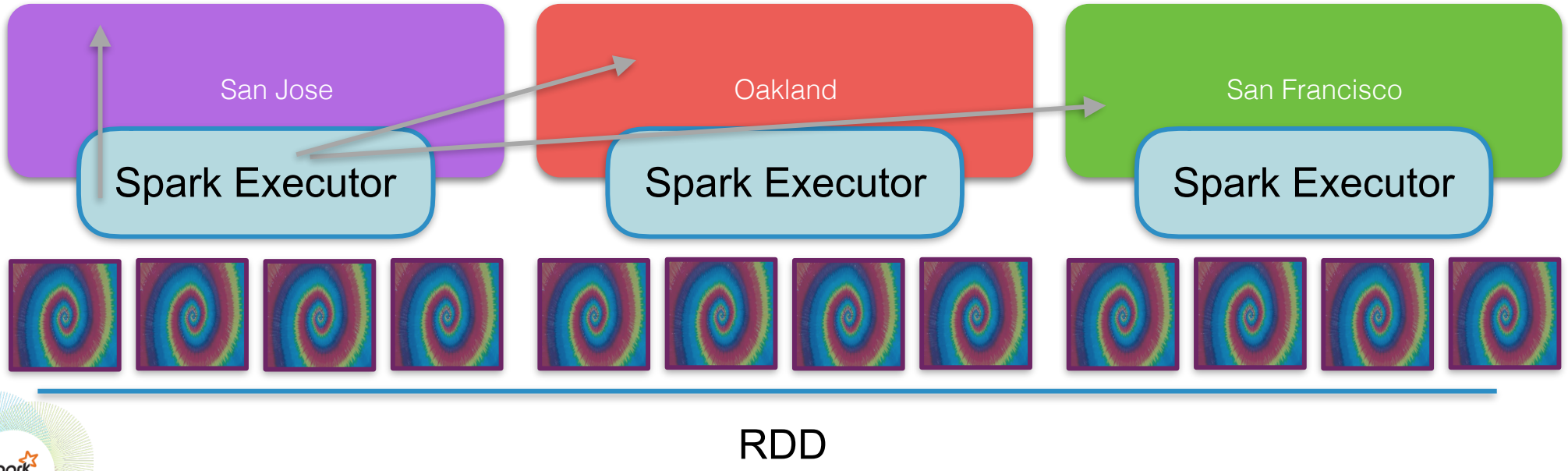
Distributed Key Retrieval

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

rdd.joinWithCassandraTable("keyspace", "table")
```



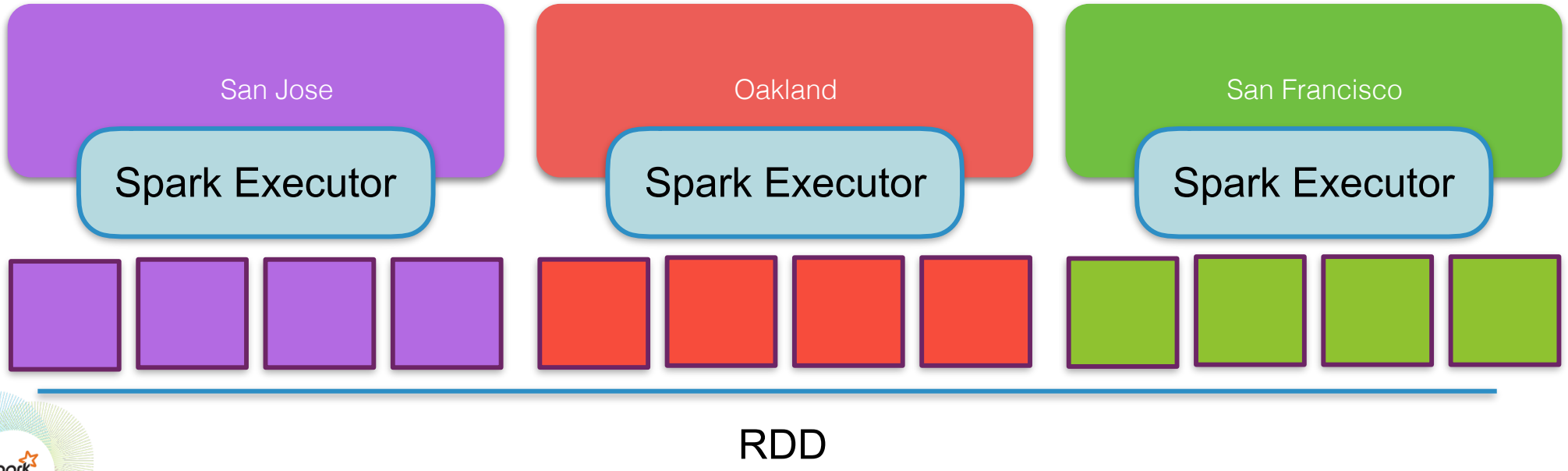
But our Data isn't Colocated

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

rdd.joinWithCassandraTable("keyspace", "table")
```



RBCR Moves bulk reshuffles our data so Data Will Be Local

```
rdd
 .repartitionByCassandraReplica("keyspace","table")
 .joinWithCassandraTable("keyspace", "table")
```



The Connector Provides a Distributed Pool for Prepared Statements and Sessions

```
CassandraConnector(sc.getConf)
```

```
rdd.mapPartitions{ it => {
  val ps = CassandraConnector(sc.getConf)
    .withSessionDo( s => s.prepare)
  it.map{ ps.bind(_).executeAsync()}
  }
```

Your Pool Ready to Be Deployed

The Connector Provides a Distributed Pool for Prepared Statements and Sessions

```
CassandraConnector(sc.getConf)
```

```
rdd.mapPartitions{ it => {
  val ps = CassandraConnector(sc.getConf)
    .withSessionDo( s => s.prepare)
  it.map{ ps.bind(_).executeAsync()}
  }
```

Session Cache

Prepared Statement Cache

Your Pool Ready to Be Deployed

The Connector Supports the DataSources Api

```
sqlContext
  .read
  .format("org.apache.spark.sql.cassandra")
  .options(Map("keyspace" -> "read_test", "table" -> "simple_kv"))
  .load
```

```
import org.apache.spark.sql.cassandra._

sqlContext
  .read
  .cassandraFormat("read_test","table")
  .load
```

https://github.com/datastax/spark-cassandra-connector/blob/master/doc/14_data_frames.md

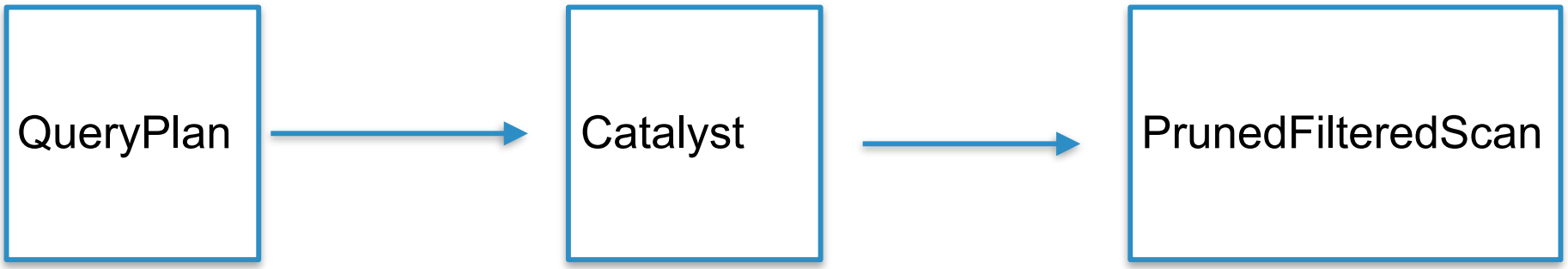
The Connector Works with Catalyst to Pushdown Predicates to Cassandra

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

df.select("vehicle_id").filter("ts > 10")
```

```
SELECT "vehicle_id" FROM TABLE
WHERE
  Token(PK) > 500 AND
  Token(PK) < 600 AND
   ts > 10
```

The Connector Works with Catalyst to Pushdown Predicates to Cassandra

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

df.select("vehicle_id").filter("ts > 10")
```



Only **Prunes** (projections) and **Filters** (predicates) are able to be pushed down.

Recent Features

- C* 3.0 Support
  - Materialized Views
  - SASL Indexes (for pushdown)
- Advanced Spark Partitioner Support
- Increased Performance on JWCT

Use C* Partitioning in Spark

Use C* Partitioning in Spark

- C* Data is Partitioned
- Spark has Partitions and partitioners
- Spark can use a known partitioner to speed up Cogroups (joins)
  - How Can we Leverage this?

Use C* Partitioning in Spark

Now if *keyBy* is used on a CassandraTableScanRDD and the PartitionKey is included in the key. The RDD will be given a C* Partitioner

```
val ks = "doc_example"
val rdd = { sc.cassandraTable[(String, Int)](ks, "users")
  .select("name" as "_1", "zipcode" as "_2", "userid")
  .keyBy[Tuple1[Int]]("userid")
}


rdd.partitioner
//res3: Option[org.apache.spark.Partitioner] =
Some(com.datastax.spark.connector.rdd.partitioner.CassandraPartitioner@94515d3e)
```

https://github.com/datastax/spark-cassandra-connector/blob/master/doc/16_partitioning.md

Use C* Partitioning in Spark

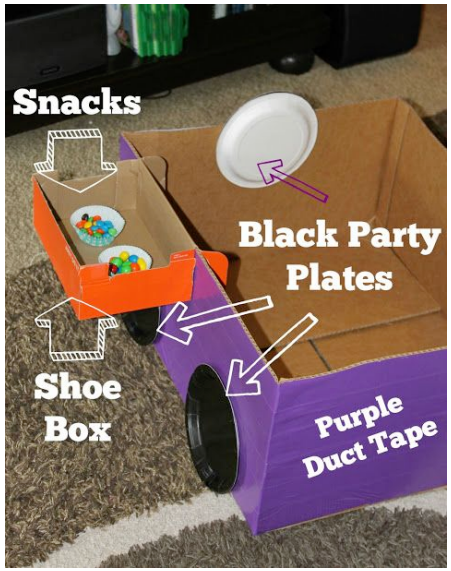
Share partitioners between Tables for joins on Partition Key

```
val ks = "doc_example"
val rdd1 = { sc.cassandraTable[(Int, Int, String)](ks, "purchases")
  .select("purchaseid" as "_1", "amount" as "_2", "objectid" as "_3", "userid")
  .keyBy[Tuple1[Int]]("userid")
}

val rdd2 = { sc.cassandraTable[(String, Int)](ks, "users")
  .select("name" as "_1", "zipcode" as "_2", "userid")
  .keyBy[Tuple1[Int]]("userid")
}.applyPartitionerFrom(rdd1) // Assigns the partitioner from the first rdd to this one

val joinRDD = rdd1.join(rdd2)
```

https://github.com/datastax/spark-cassandra-connector/blob/master/doc/16_partitioning.md

Use C* Partitioning in Spark

Many other uses for this, try it yourself!



- All self joins using the Partition key
- Groups within C* Partitions
- Anything formerly using SpanBy
- Joins with other RDDs
  And much more!

https://github.com/datastax/spark-cassandra-connector/blob/master/doc/16_partitioning.md

Enhanced Parallelism with JWCT

- joinWithCassandraTable now has increased concurrency and parallelism!
- 5X Speed increases in some cases
- https://datastax-oss.atlassian.net/browse/SPARKC-233
- Thanks Jaroslaw!

The Connector wants you!

- OSS Project that loves community involvement
- Bug Reports
- Feature Requests
- Doc Improvements
- Come join us!

Vin Diesel may or may not be a contributor

Tons of Videos at Datastax Academy

https://academy.datastax.com/

https://academy.datastax.com/courses/getting-started-apache-spark

Tons of Videos at Datastax Academy

https://academy.datastax.com/

https://academy.datastax.com/courses/getting-started-apache-spark

See you at Cassandra Summit!

CASSANDRA SUMMIT 2016

San Jose Convention Center
September 7-9, 2016

Join me and 3500 of your database peers, and take a deep dive into Apache Cassandra™, the massively scalable NoSQL database that powers global businesses like Apple, Spotify, Netflix and Sony.

Build Something Disruptive

https://cassandrasummit.org/

SPARK SUMMIT 2016
DATA SCIENCE AND ENGINEERING AT SCALE
JUNE 6-8, 2016 SAN FRANCISCO