Analyzing Time Series Data with Apache Spark and Cassandra

Patrick McFadin

Chief Evangelist for Apache Cassandra, DataStax

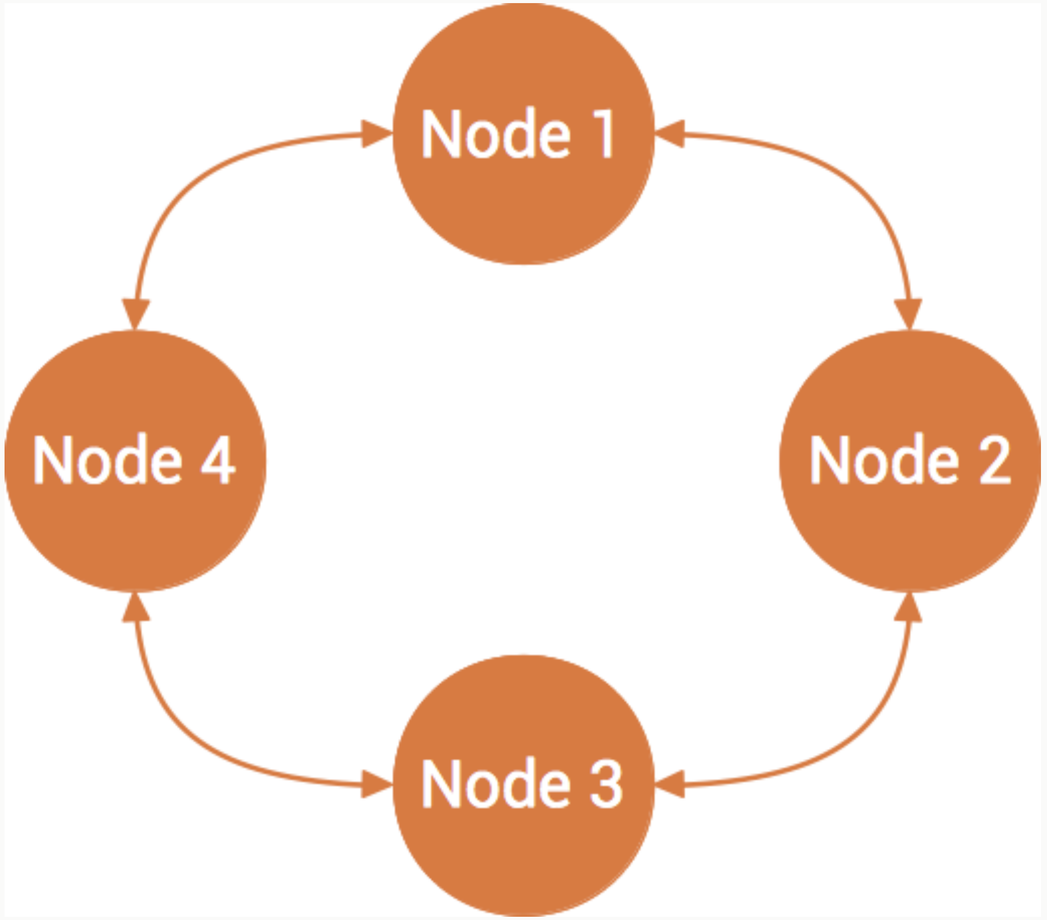
@PatrickMcFadin

DATASTAX

Apache Cassandra

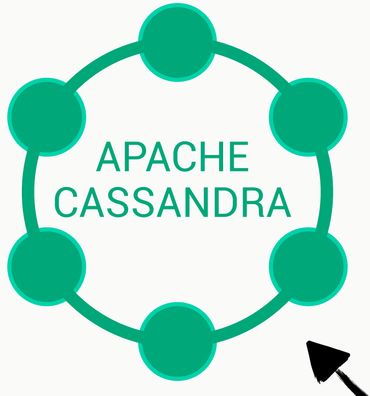
Cassandra is…

- Shared nothing
- Masterless peer-to-peer
- Great scaling story
- Resilient to failure



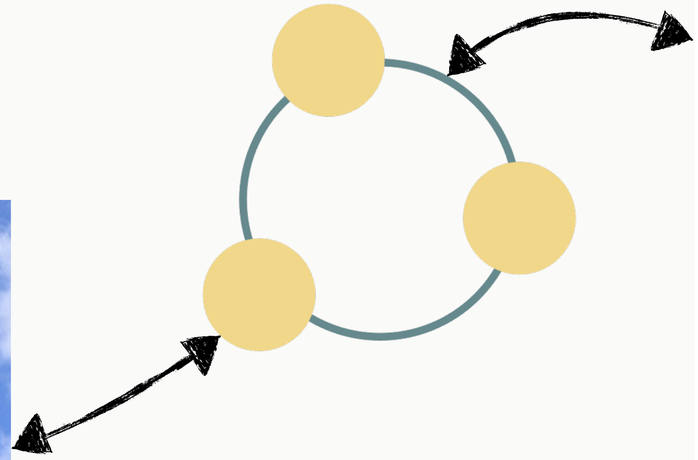
Cassandra for Applications

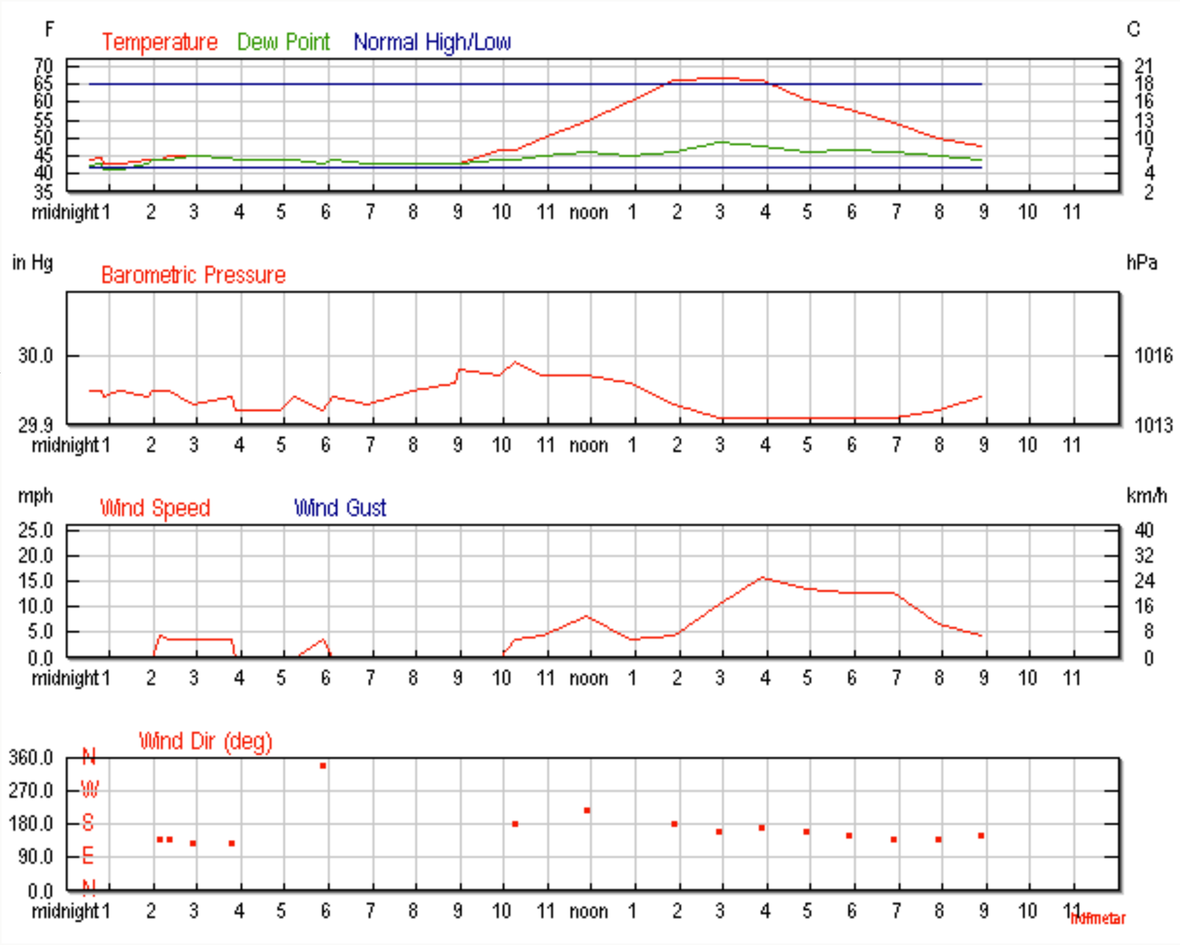
DATASTAX



Example: Weather Station

- Weather station collects data
- Cassandra stores in sequence
- Application reads in sequence



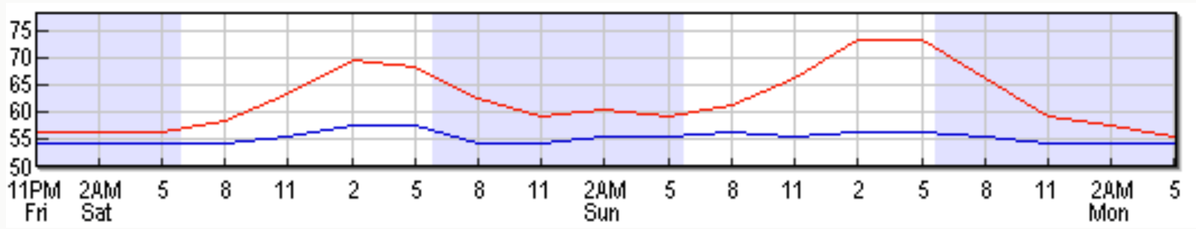


Queries supported

DATASTAX

Get weather data given

- Weather Station ID
- Weather Station ID and Time
- Weather Station ID and Range of Time




```
CREATE TABLE raw_weather_data (
   wsid text,
   year int,
   month int,
   day int,
   hour int,
   temperature double,
   dewpoint double,
   pressure double,
   wind_direction int,
   wind_speed double,
   sky_condition int,
   sky_condition_text text,
   one_hour_precip double,
   six_hour_precip double,
   PRIMARY KEY ((wsid), year, month, day, hour)
) WITH CLUSTERING ORDER BY (year DESC, month DESC, day DESC, hour DESC);
```

Aggregation Queries

DATASTAX

Get temperature stats given

- Weather Station ID
- Weather Station ID and Time
- Weather Station ID and Range of Time

Windsor California

July 1, 2014

High: 73.4

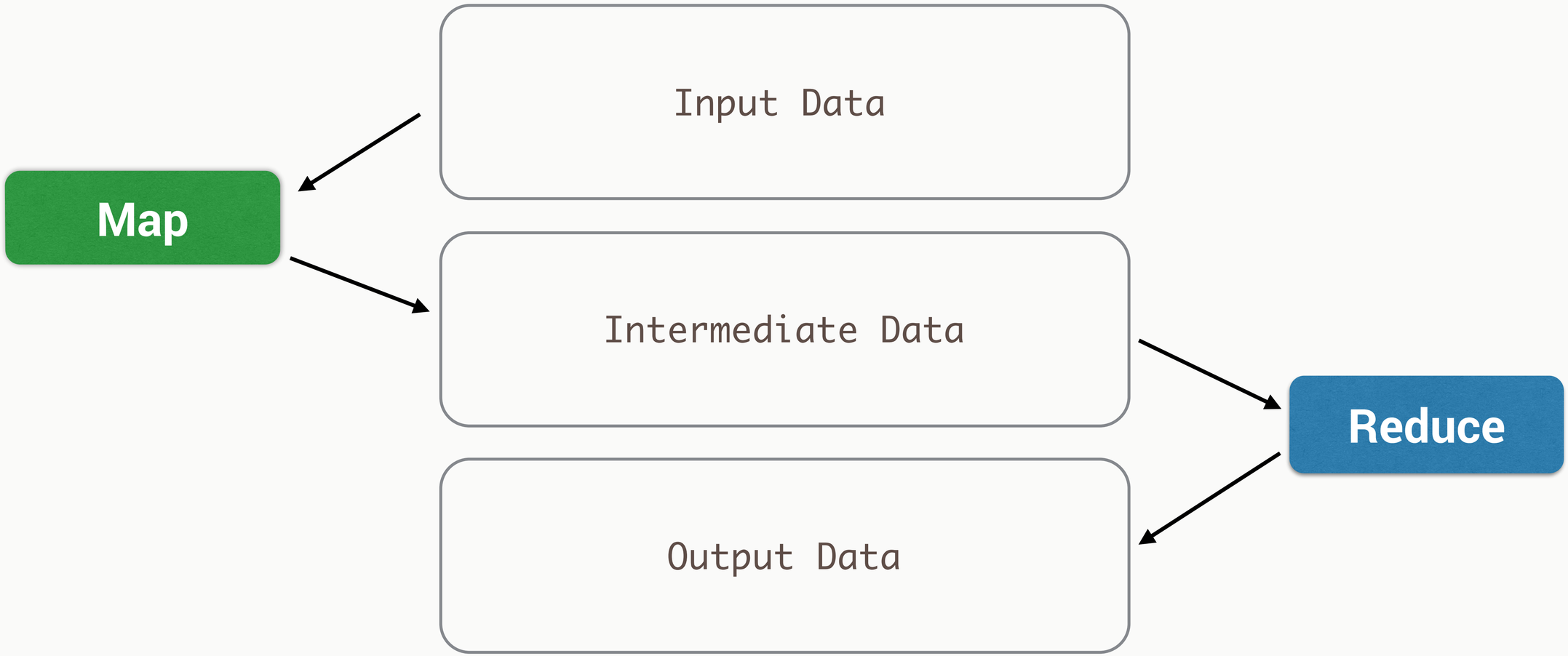
Low : 51.4

```
CREATE TABLE daily_aggregate_temperature (
   wsid text,
   year int,
   month int,
   day int,
   high double,
   low double,
   mean double,
   variance double,
   stdev double,
   PRIMARY KEY ((wsid), year, month, day)
) WITH CLUSTERING ORDER BY (year DESC, month DESC, day DESC);
```

DATASTAX

Apache Spark

Map Reduce



Apache Spark

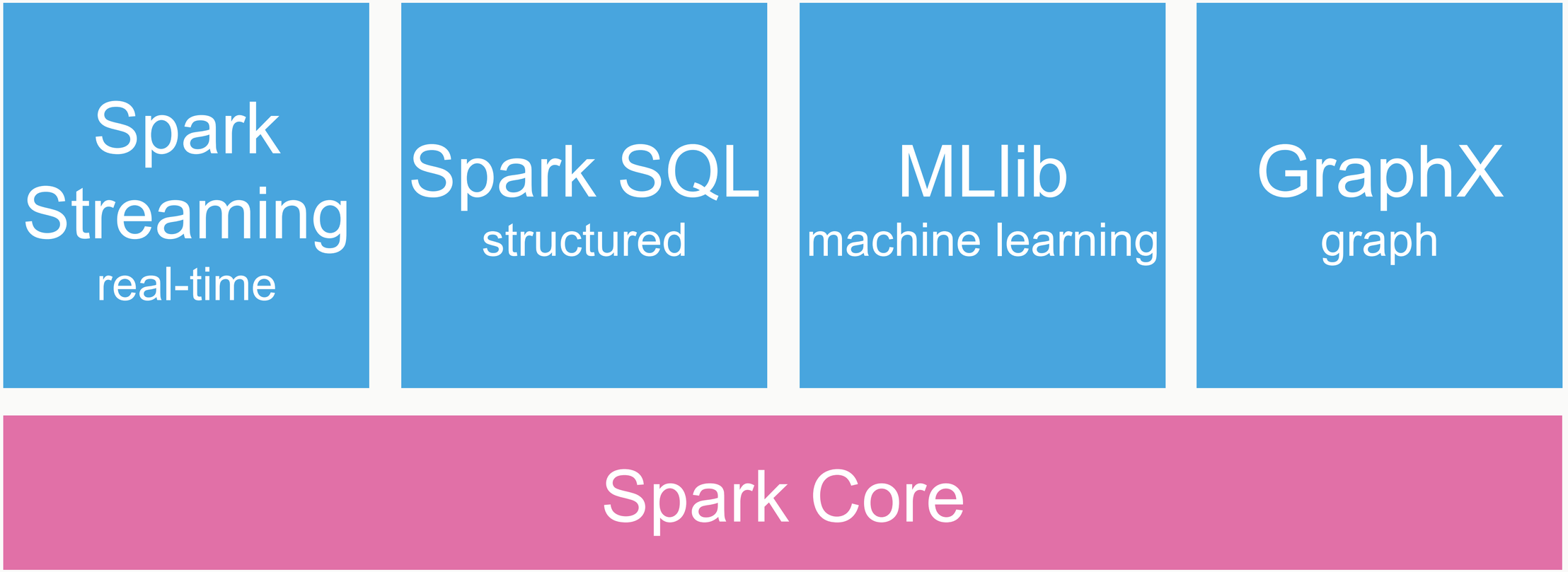
DATASTAX

- 10x faster on disk,100x faster in memory than Hadoop MR
- Works out of the box on EMR
- Fault Tolerant Distributed Datasets
- Batch, iterative and streaming analysis
- In Memory Storage and Disk
- Integrates with Most File and Storage Options

→ Up to 100× faster (2-10× on disk)

→ 2-5× less code

Spark Components



Blog

Company Blog

- All Posts
- Partners
- Events
- Press Releases

Developer Blog

- All Posts
- Spark
- Spark Streaming
- Spark SQL
- MLlib
- Spark Summit

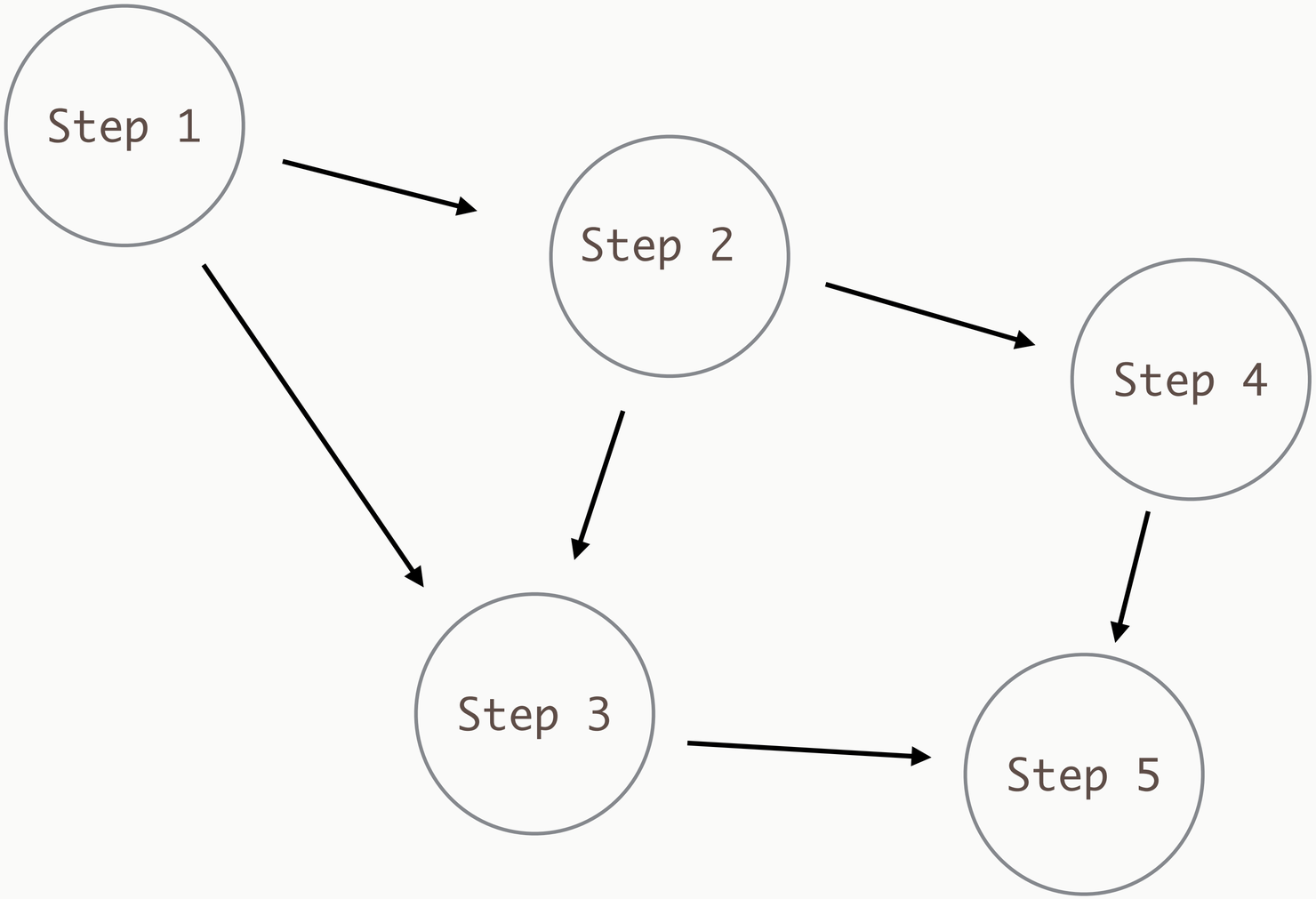
Spark the fastest open source engine for sorting a petabyte

October 10, 2014 | by Reynold Xin

Tags: Spark

Apache Spark has seen phenomenal adoption, being widely slated as the successor to Hadoop MapReduce, and being deployed in clusters from a handful to thousands of nodes. While it was clear to everybody that Spark is more efficient than MapReduce for data that fits in memory, we heard that some organizations were having trouble pushing it to large scale datasets that could not fit in memory. Therefore, since the inception of Databricks, we have devoted much effort, together with the Spark community, to improve the stability, scalability, and performance of Spark. Spark works well for gigabytes or terabytes of data, and it should also work well for petabytes.

The DAG



org.apache.spark.rdd.RDD

DATASTAX

Resilient **D**istributed **D**ataset (RDD)

- Created through transformations on data (map,filter..) or other RDDs
- Immutable
- Partitioned
- Reusable

RDD Operations

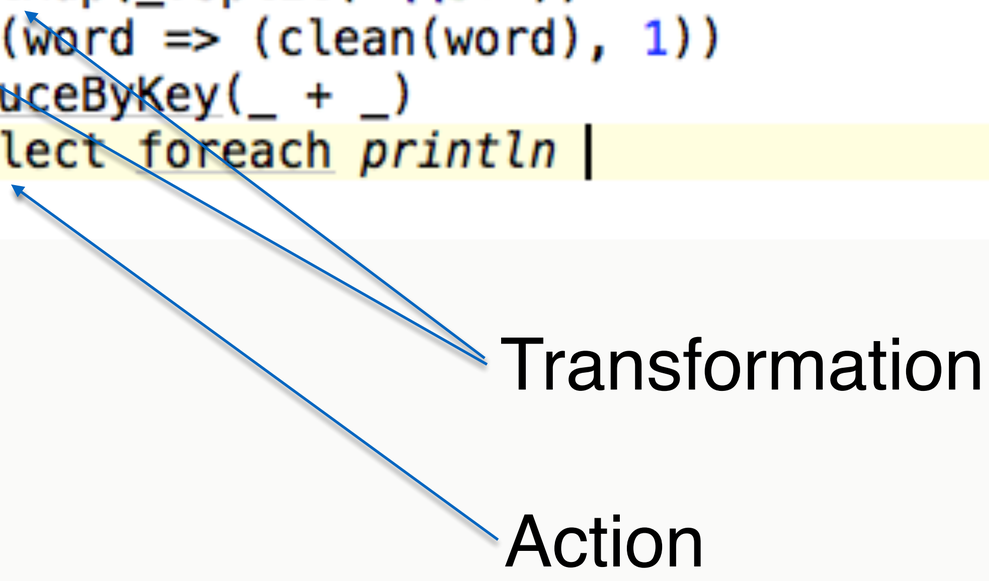
DATASTAX

- Transformations - Similar to scala collections API
  - Produce new RDDs
  - `filter, flatmap, map, distinct, groupBy, union, zip, reduceByKey, subtract`
- Actions
  - Require materialization of the records to generate a value
  - `collect: Array[T], count, fold, reduce..`

RDD Operations

DATASTAX

```
16  object SparkWordCount extends WordCountBlueprint {
17
18    sc.textFile("./src/main/resources/data/words")
19      .flatMap(_.split("\\s+"))
20      .map(word => (clean(word), 1))
21      .reduceByKey(_ + _)
22      .collect foreach println
23  }
```



DATASTAX

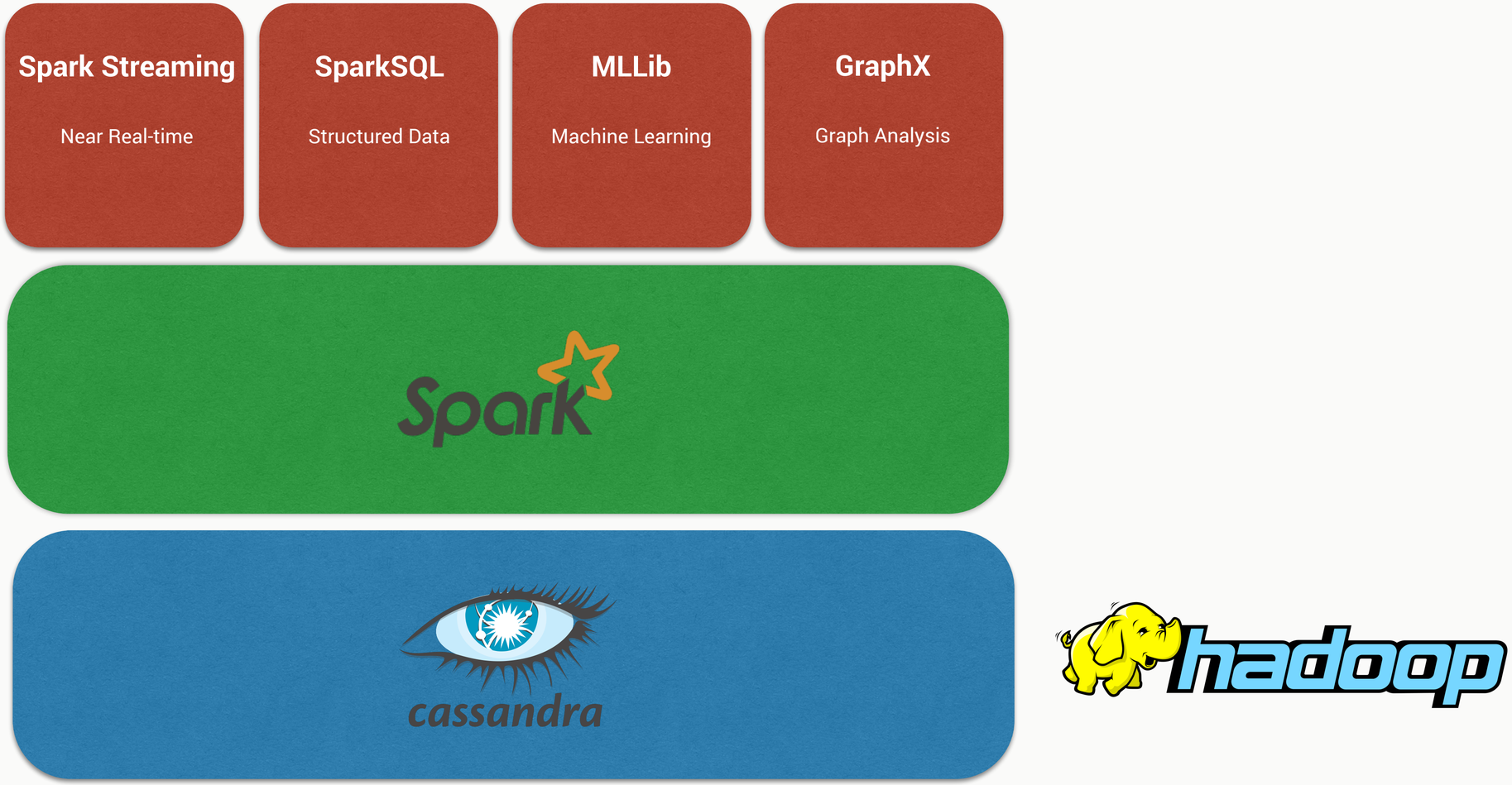
Cassandra and Spark

Great combo

DATASTAX

Great combo

DATASTAX



Great combo

Spark Streaming
Near Real-time

SparkSQL
Structured Data

MLLib
Machine Learning

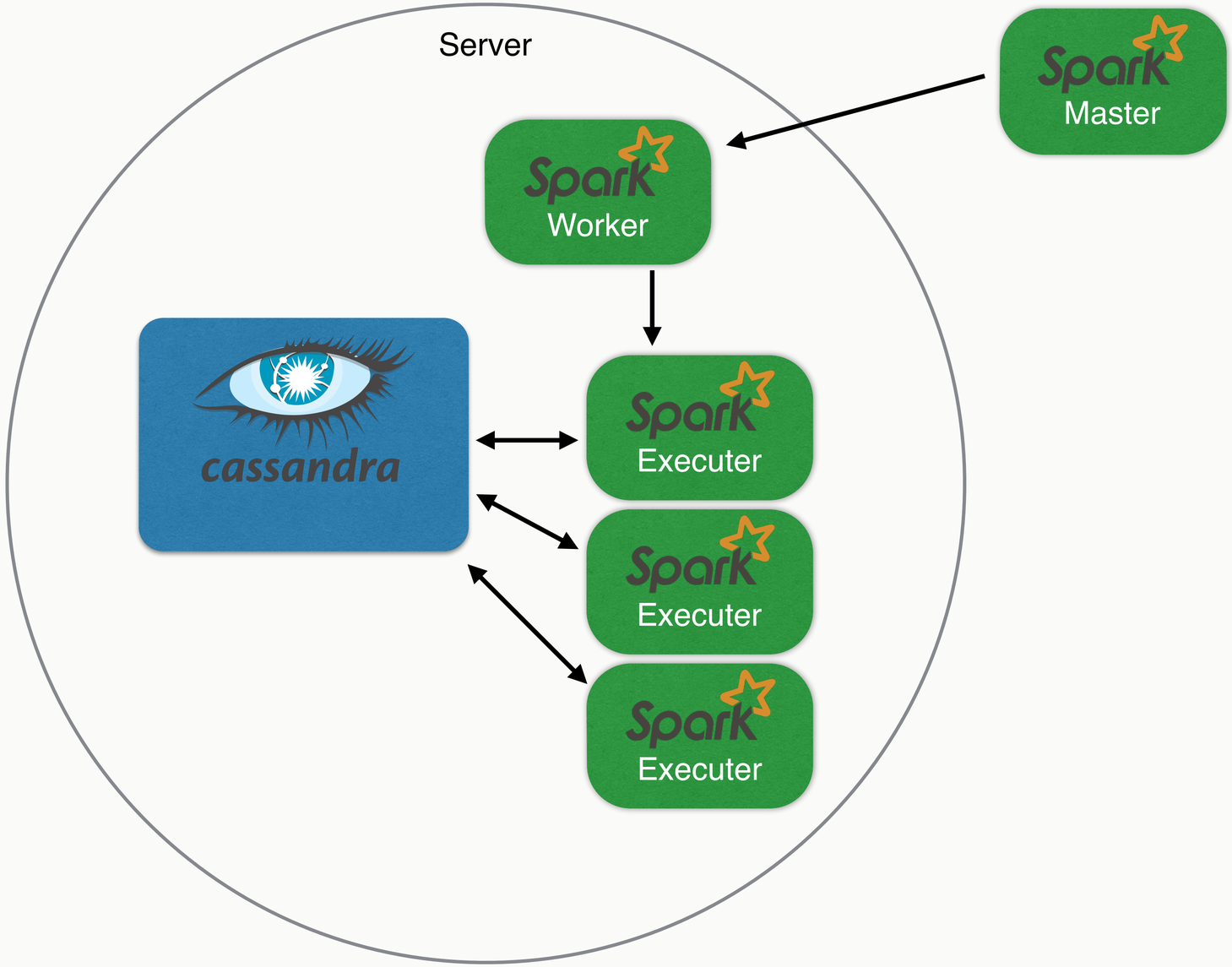
GraphX
Graph Analysis

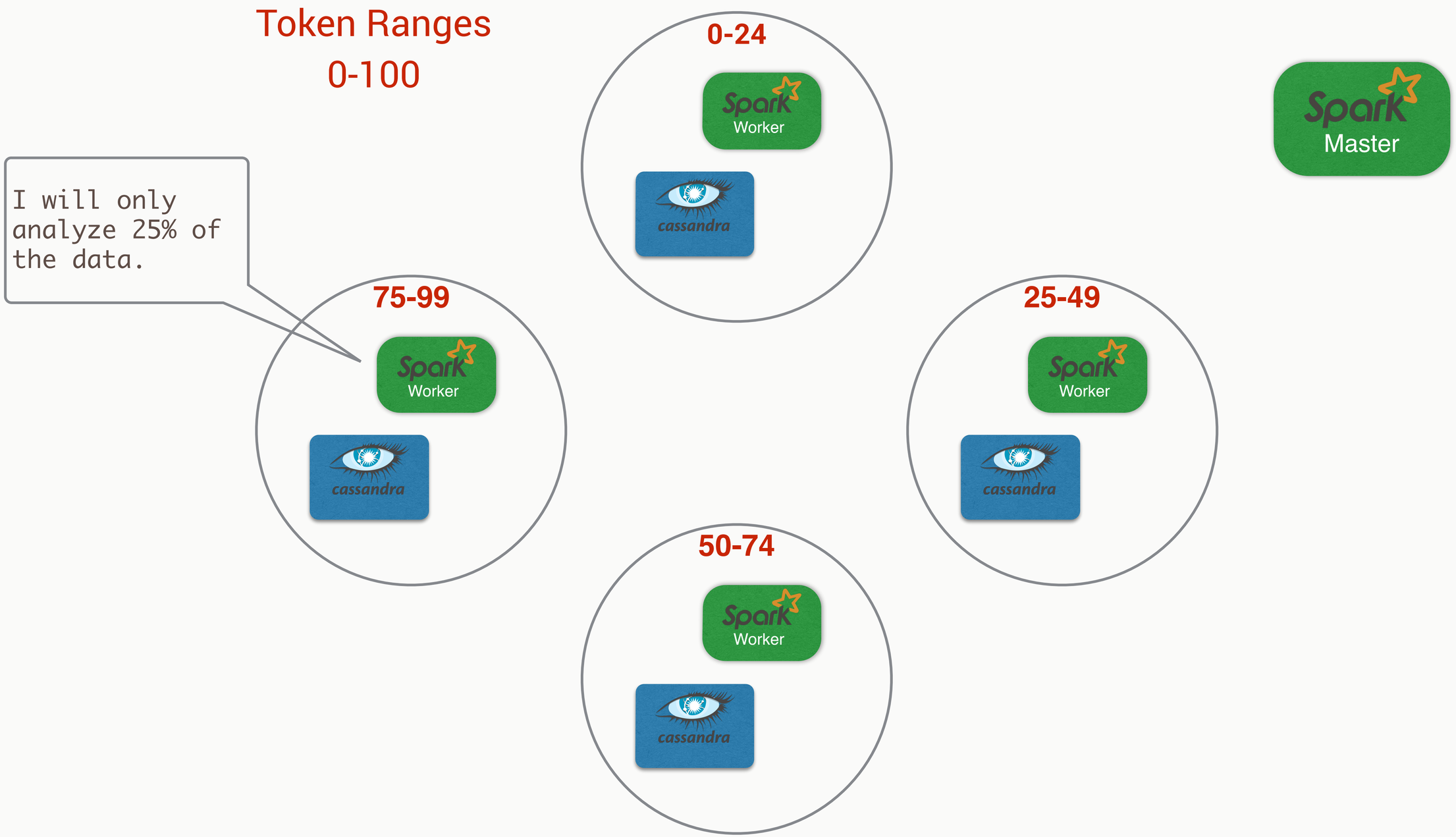
Spark

Spark Connector

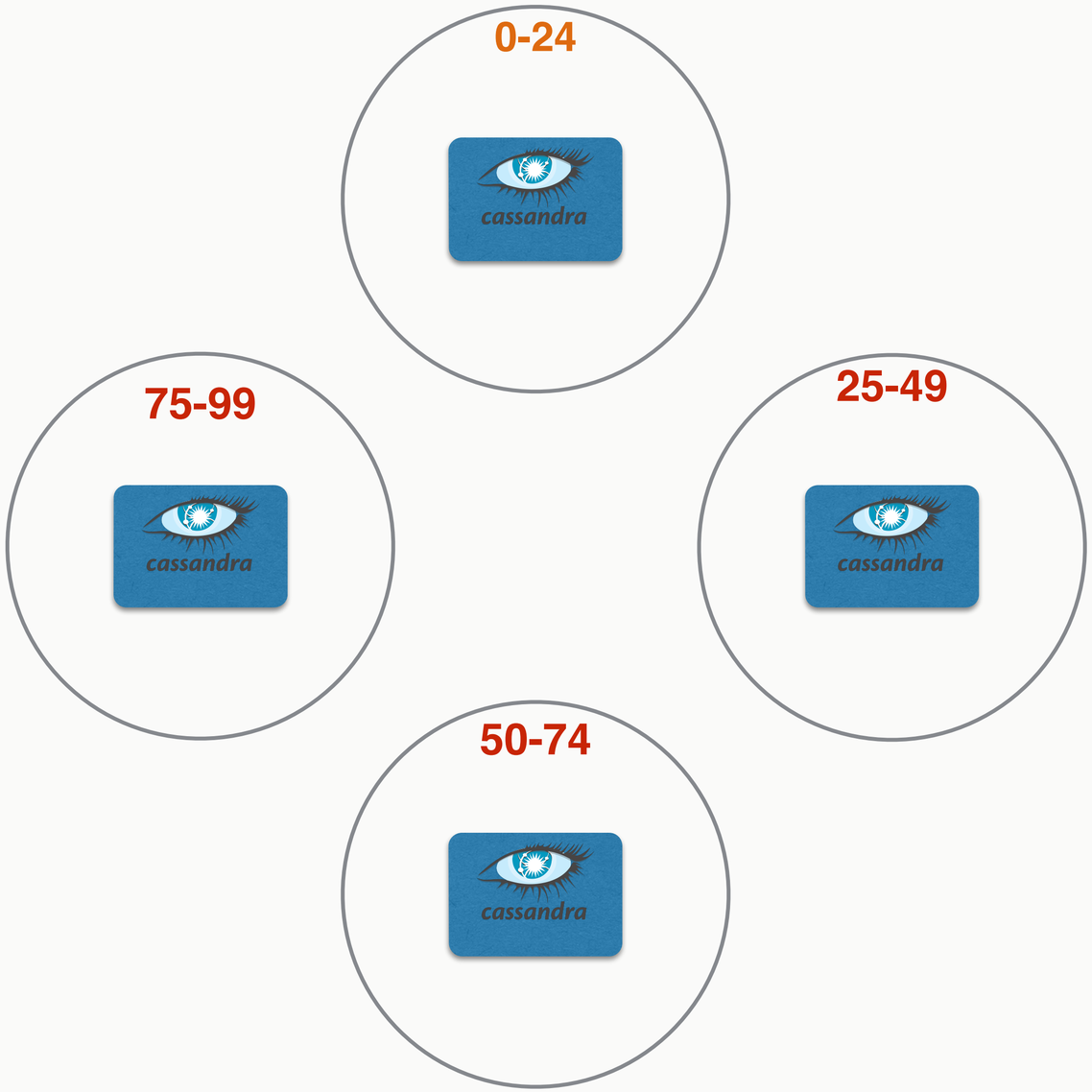
```
CREATE TABLE raw_weather_data (
   wsid text,
   year int,
   month int,
   day int,
   hour int,
   temperature double,
   dewpoint double,
   pressure double,
   wind_direction int,
   wind_speed double,
   sky_condition int,
   sky_condition_text text,
   one_hour_precip double,
   six_hour_precip double,
   PRIMARY KEY ((wsid), year, month, day, hour)
) WITH CLUSTERING ORDER BY (year DESC, month DESC, day DESC, hour DESC);
```

cassandra

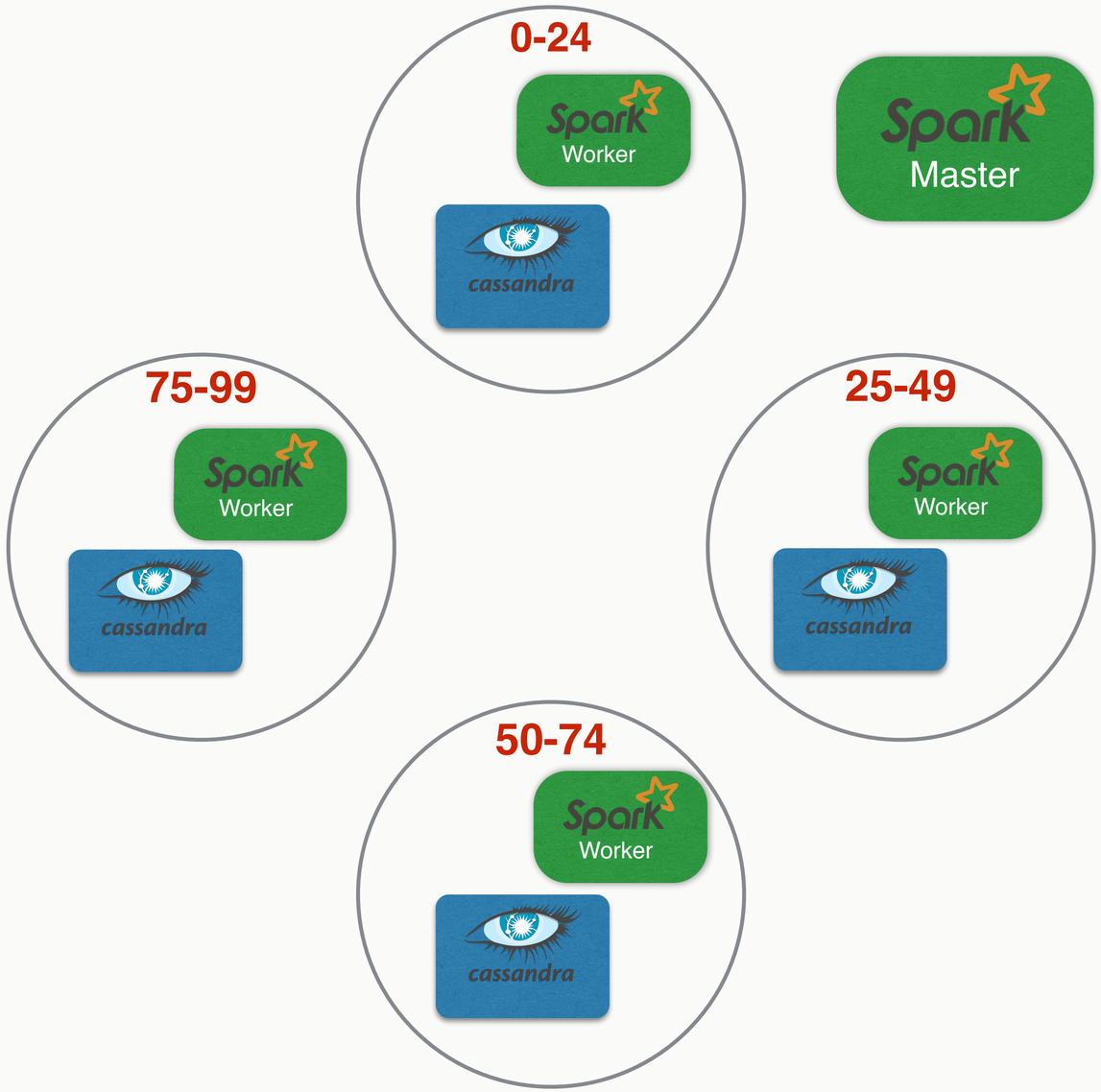

Server
Spark
Master
Spark
Worker
cassandra
Spark
Executer
Spark
Executer
Spark
Executer

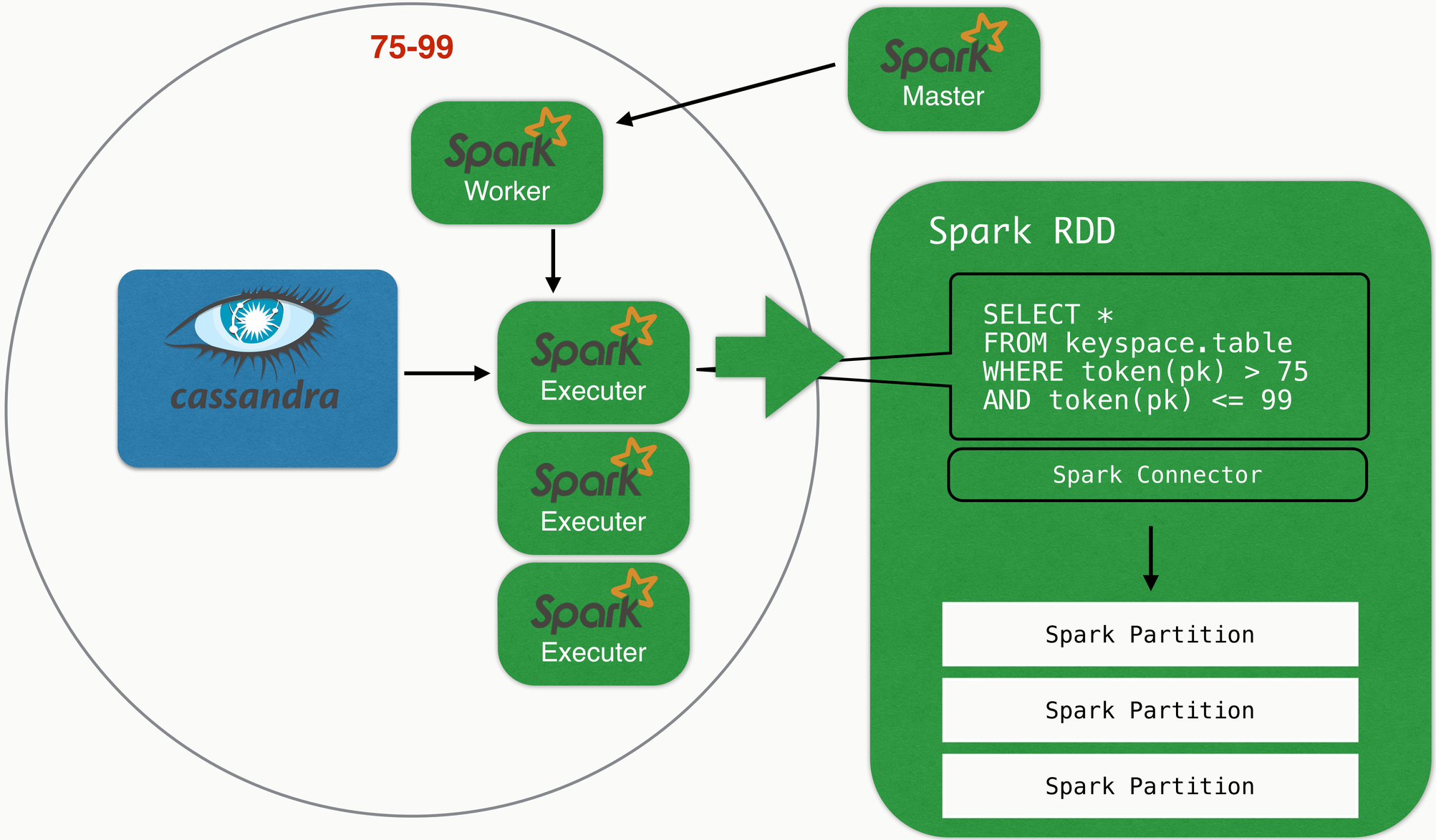

Token Ranges
0-100
I will only analyze 25% of the data.
0-24
Spark Worker
cassandra
Spark Master
75-99
Spark Worker
cassandra
25-49
Spark Worker
cassandra
50-74
Spark Worker
cassandra

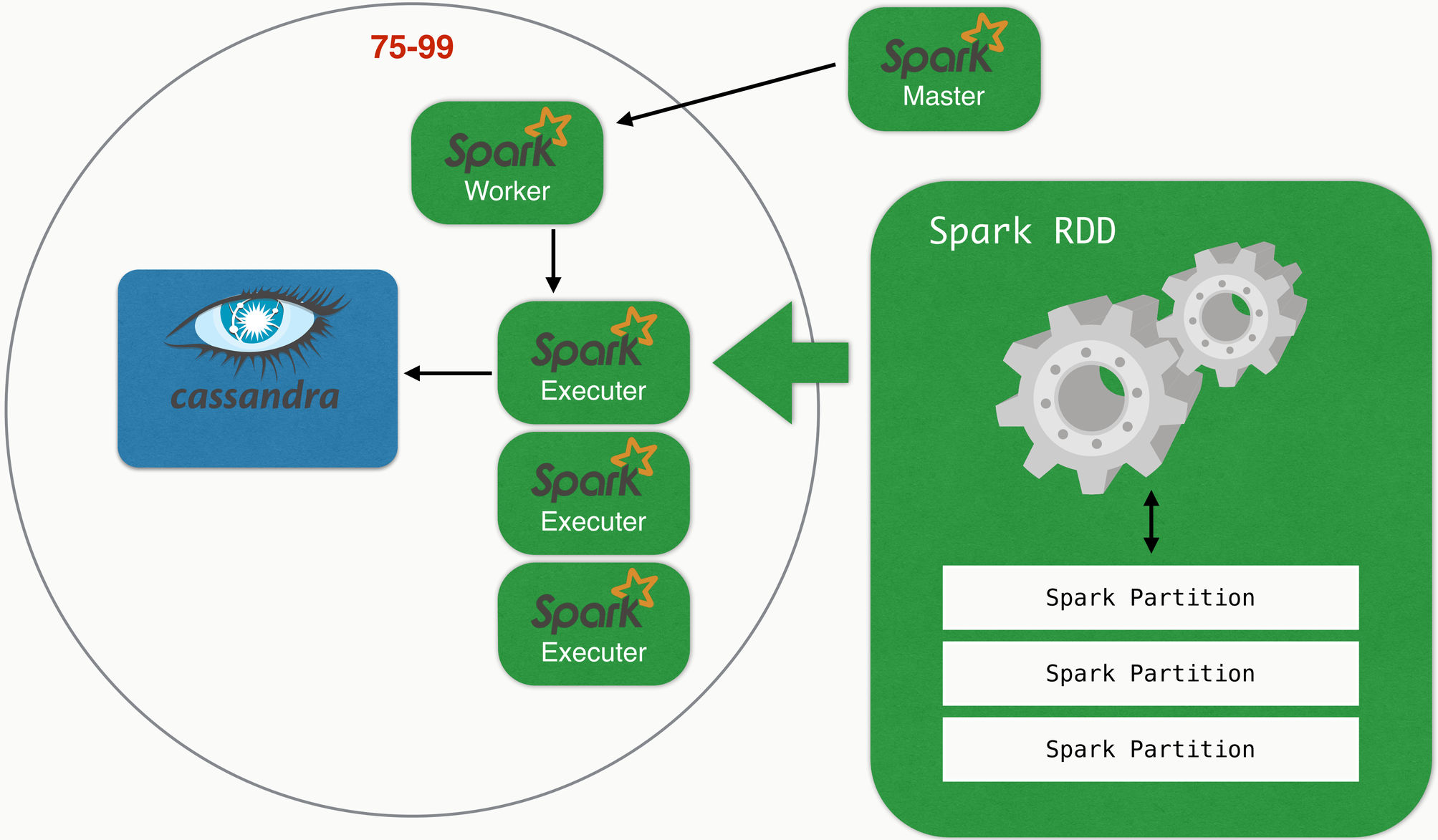
Transactional



Analytics




75-99
Spark
Master
Spark
Worker
cassandra
Spark
Executer
Spark
Executer
Spark
Executer
Spark RDD
SELECT *
FROM keyspace.table
WHERE token(pk) > 75
AND token(pk) <= 99
Spark Connector
Spark Partition
Spark Partition
Spark Partition


75-99
Spark
Master
Spark
Worker
cassandra
Spark
Executer
Spark
Executer
Spark
Executer
Spark RDD
Spark Partition
Spark Partition
Spark Partition

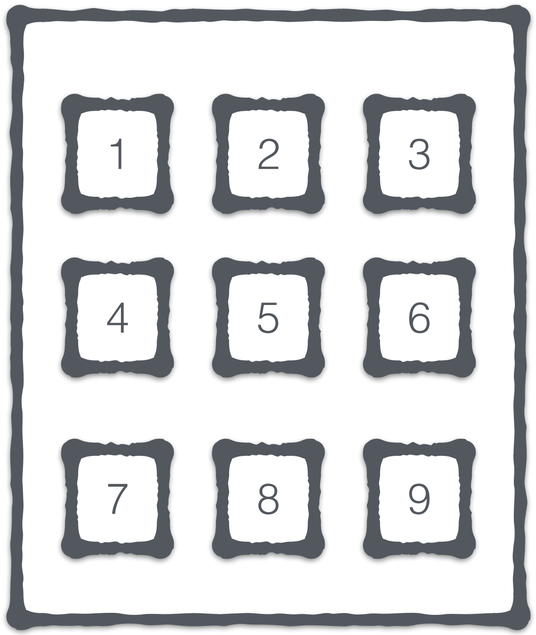
Spark Reads on Cassandra

Awesome animation by DataStax's own Russell Spitzer

Spark RDDs Represent a Large Amount of Data Partitioned into Chunks

Node 1

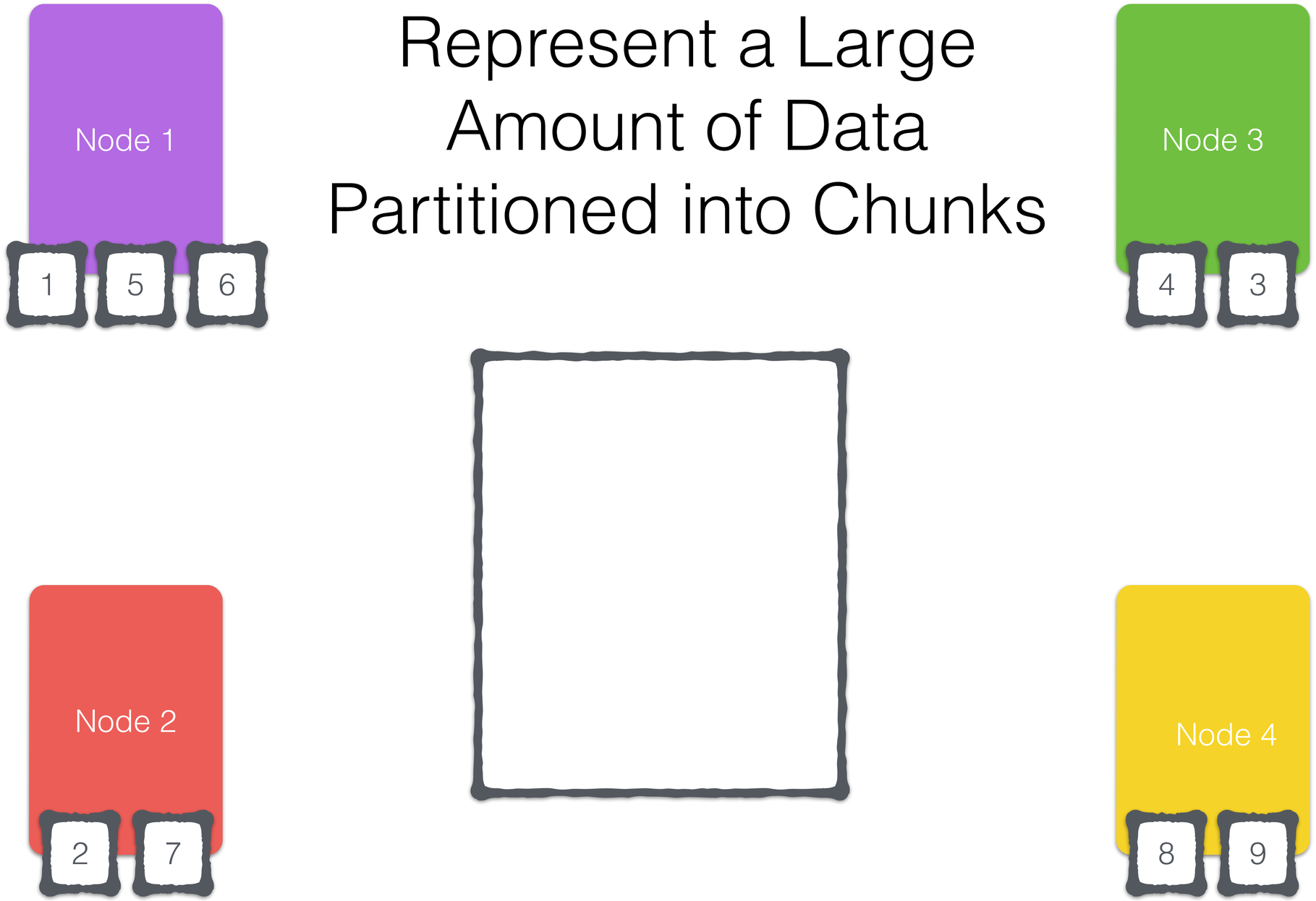
Node 3



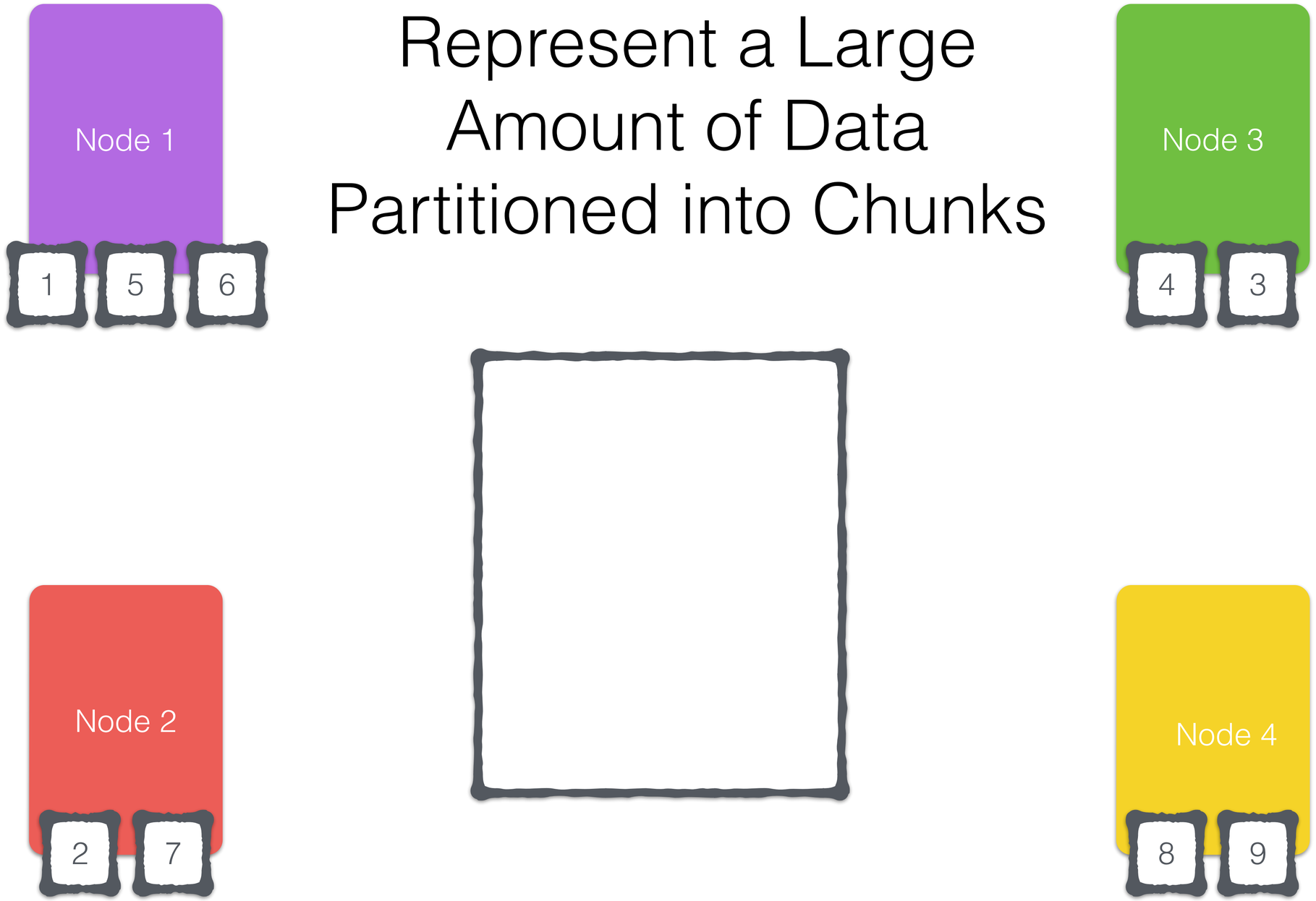
Node 2

Node 4

Spark RDDs Represent a Large Amount of Data Partitioned into Chunks

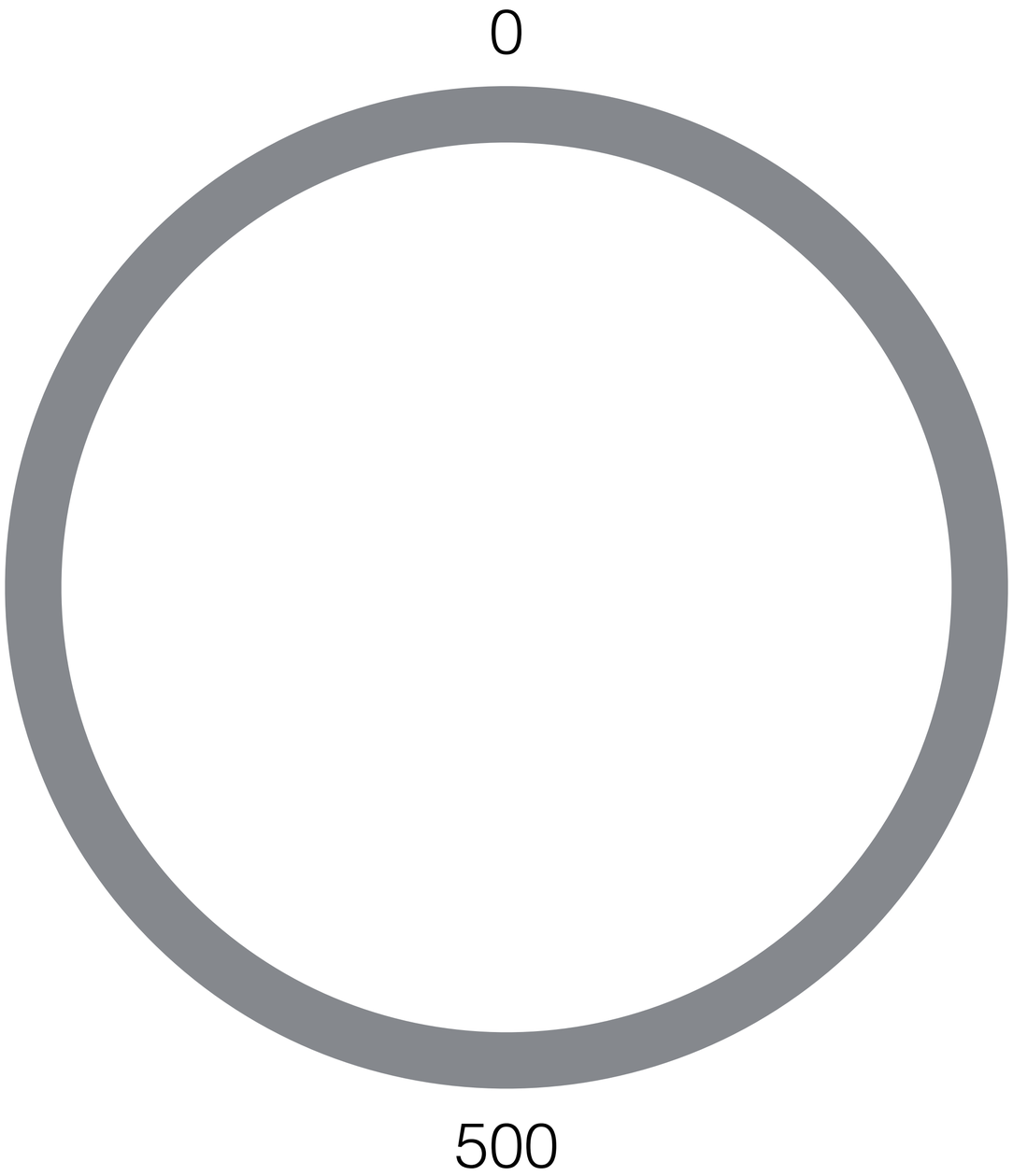


Spark RDDs Represent a Large Amount of Data Partitioned into Chunks

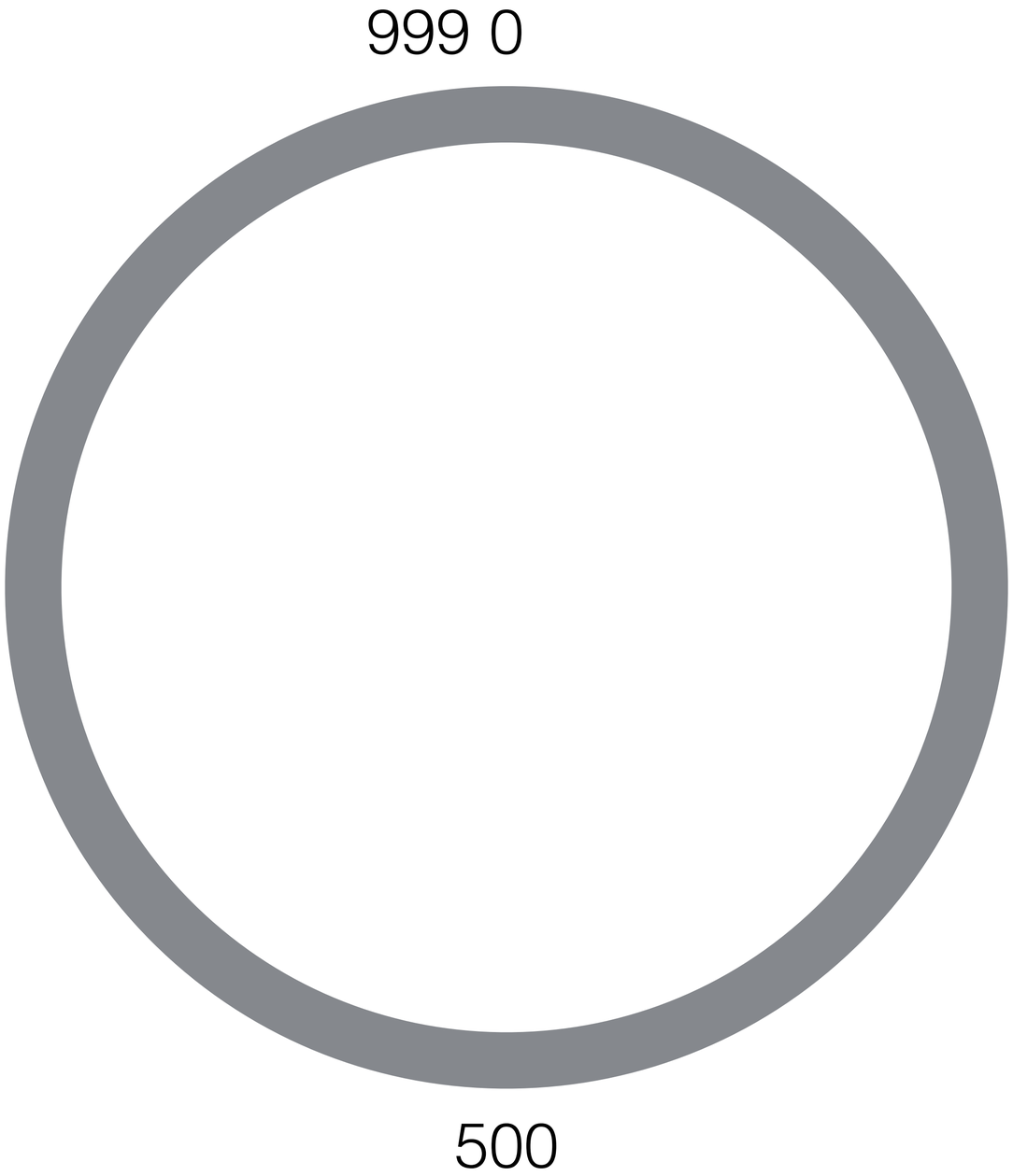


Cassandra Data is Distributed By Token Range

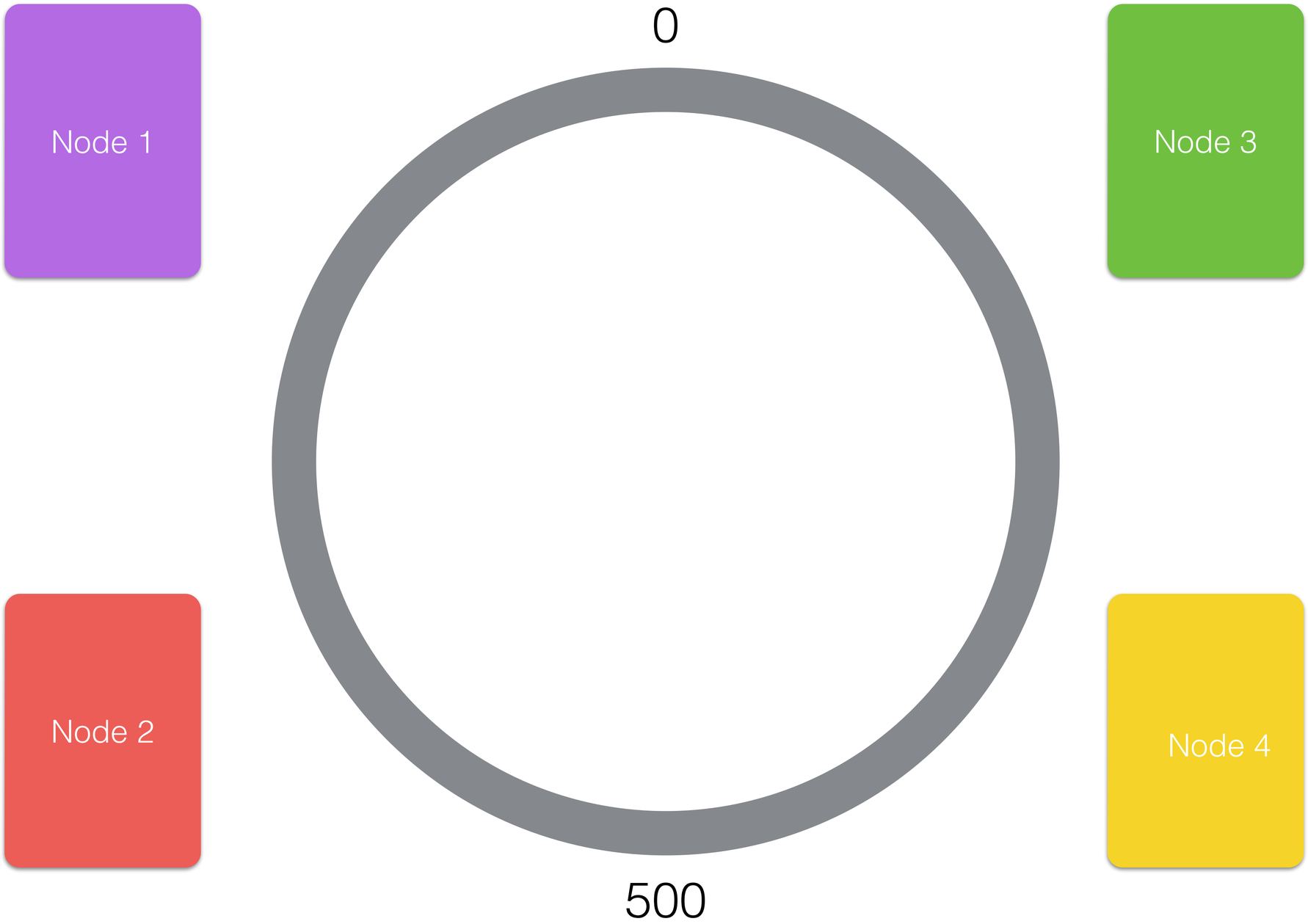
Cassandra Data is Distributed By Token Range



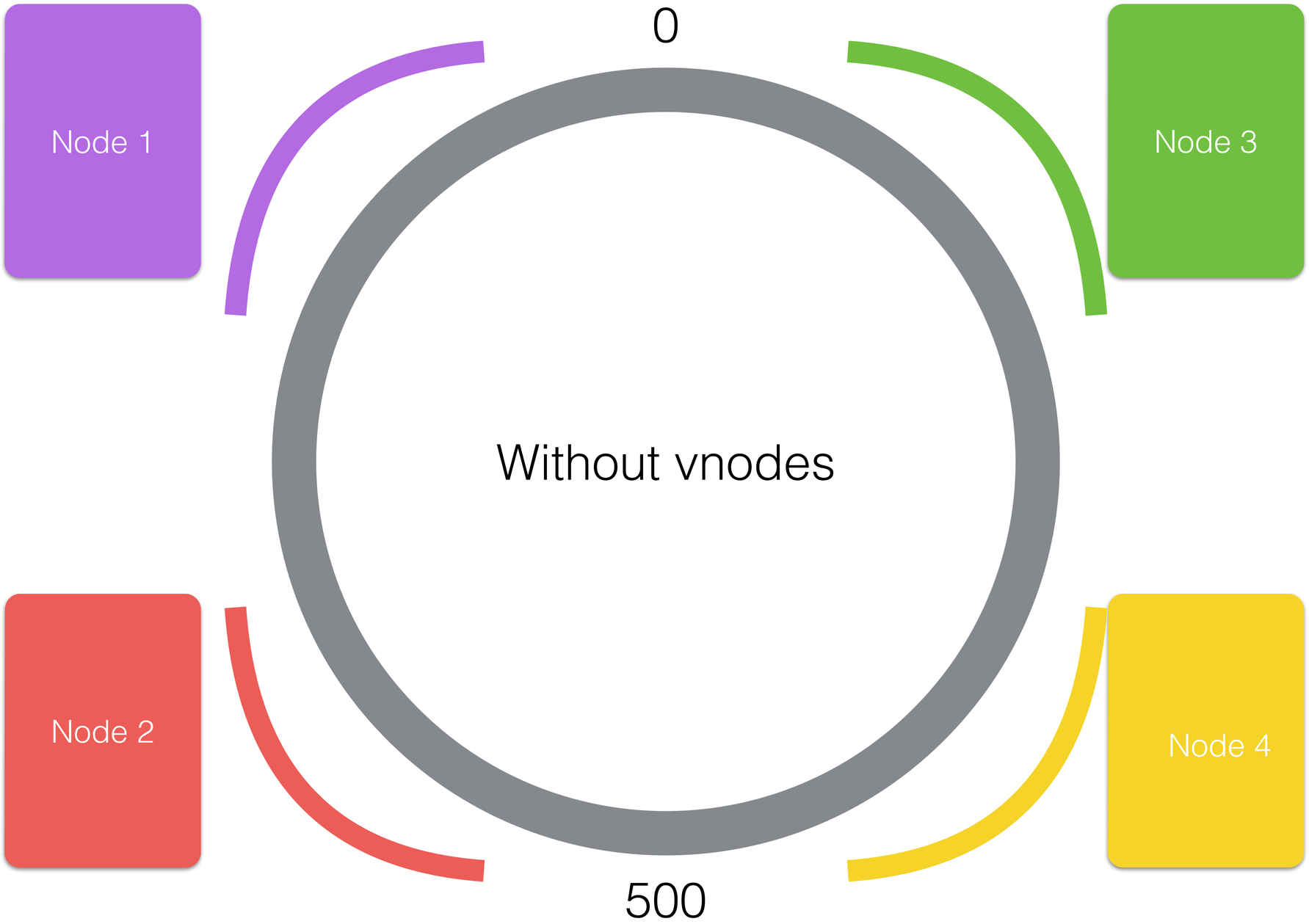
Cassandra Data is Distributed By Token Range



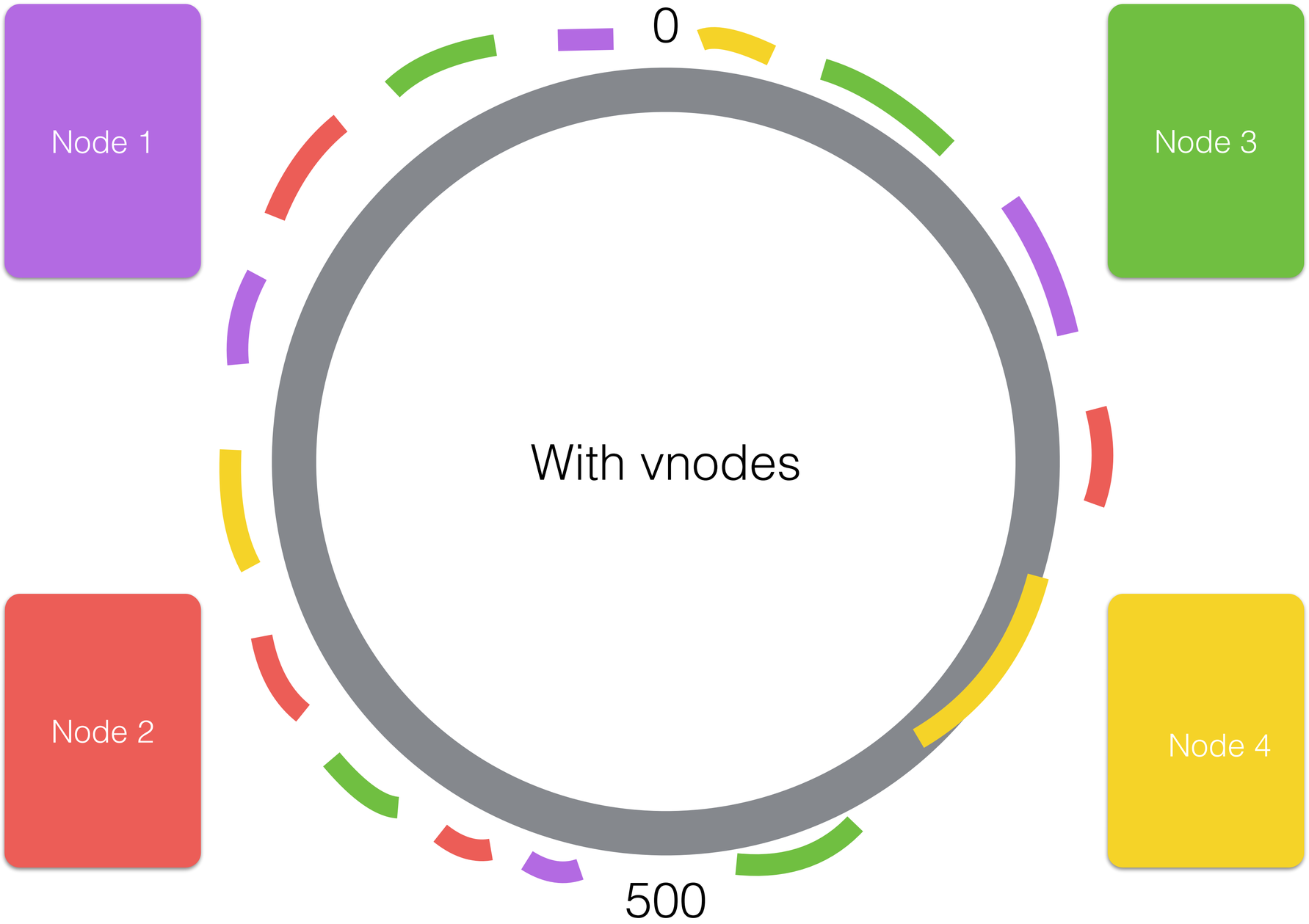
Cassandra Data is Distributed By Token Range



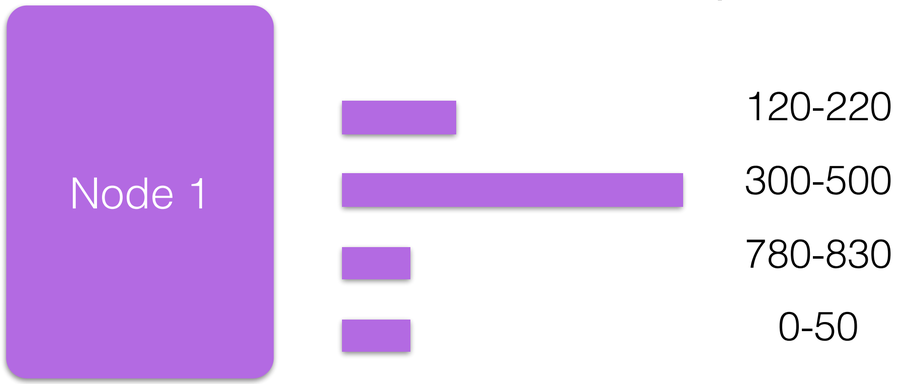
Cassandra Data is Distributed By Token Range



Cassandra Data is Distributed By Token Range

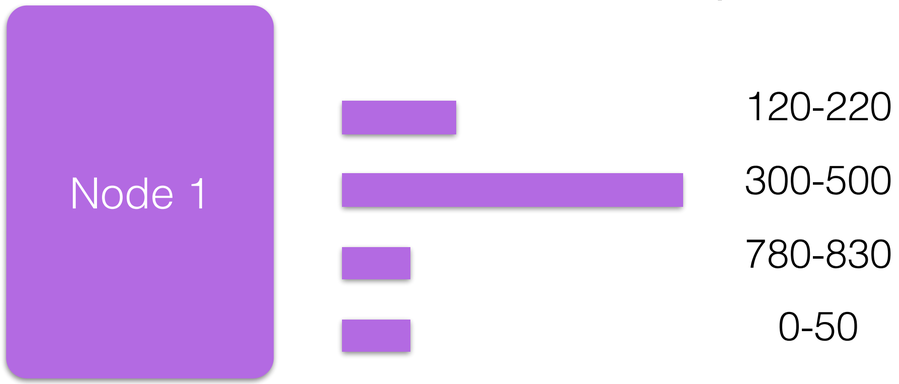


The Connector Uses Information on the Node to Make Spark Partitions

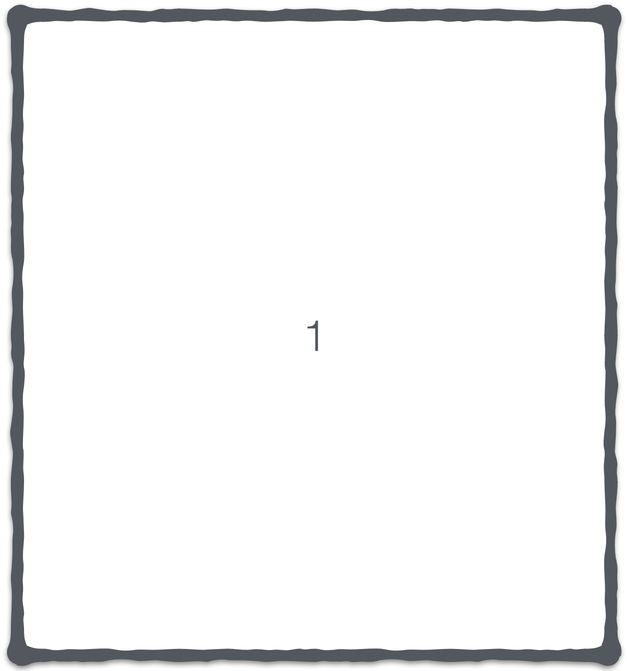



```
spark.cassandra.input.split.size 50

Reported density is 0.5
```

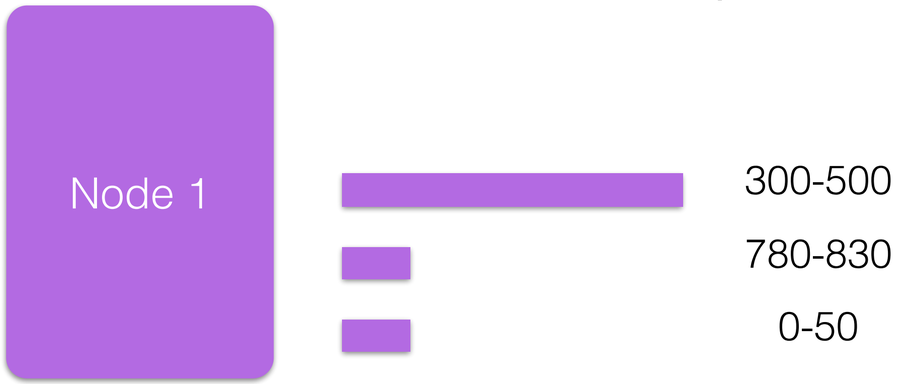
The Connector Uses Information on the Node to Make Spark Partitions




```
spark.cassandra.input.split.size 50

Reported density is 0.5
```

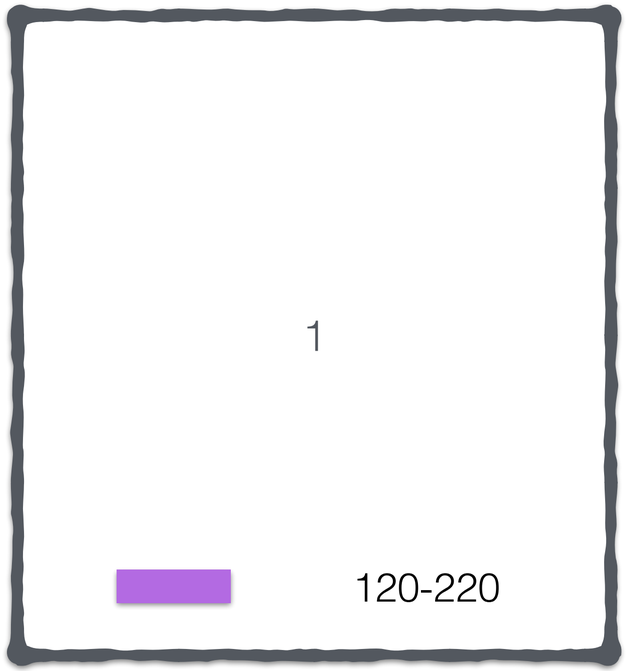


The Connector Uses Information on the Node to Make Spark Partitions

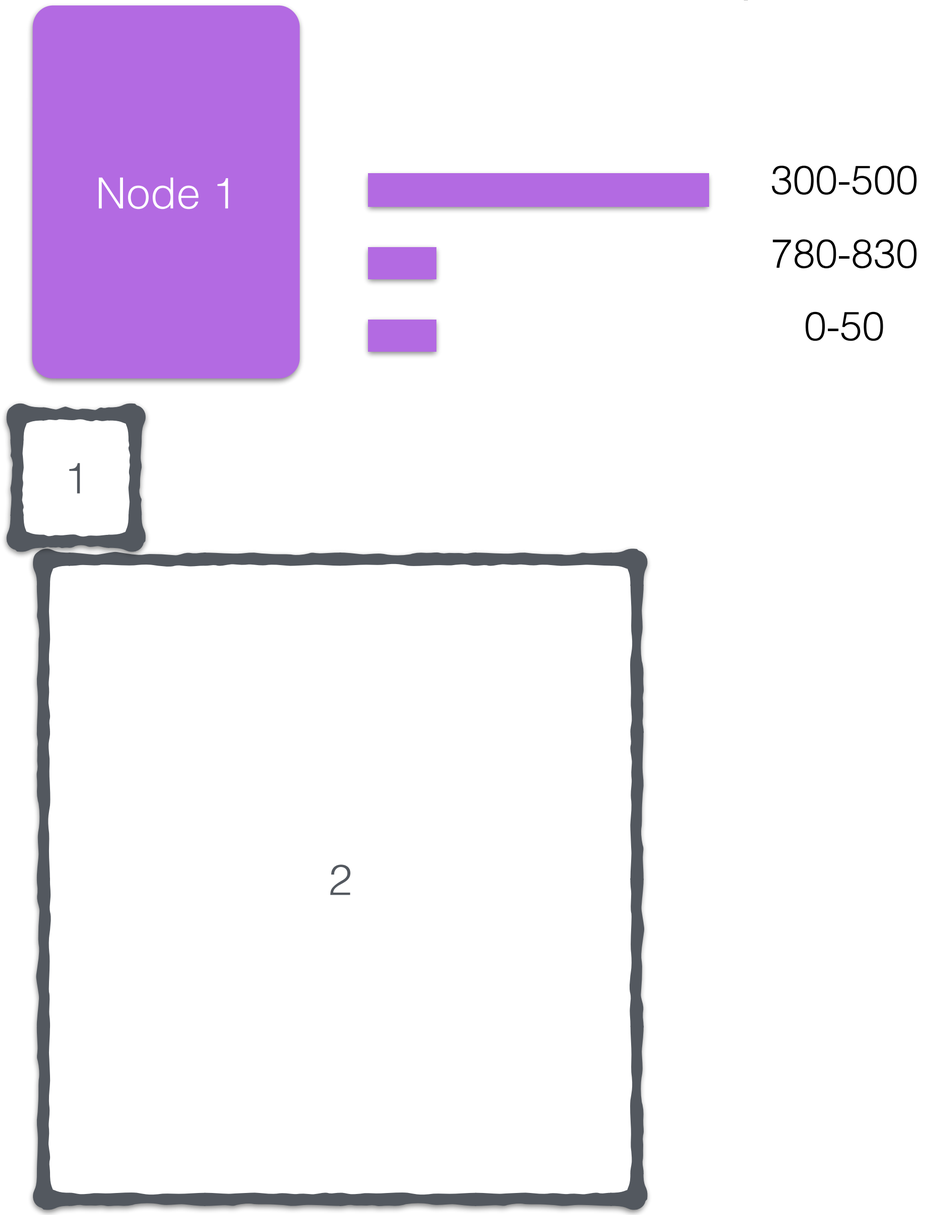


`spark.cassandra.input.split.size` 50

`Reported density is 0.5`



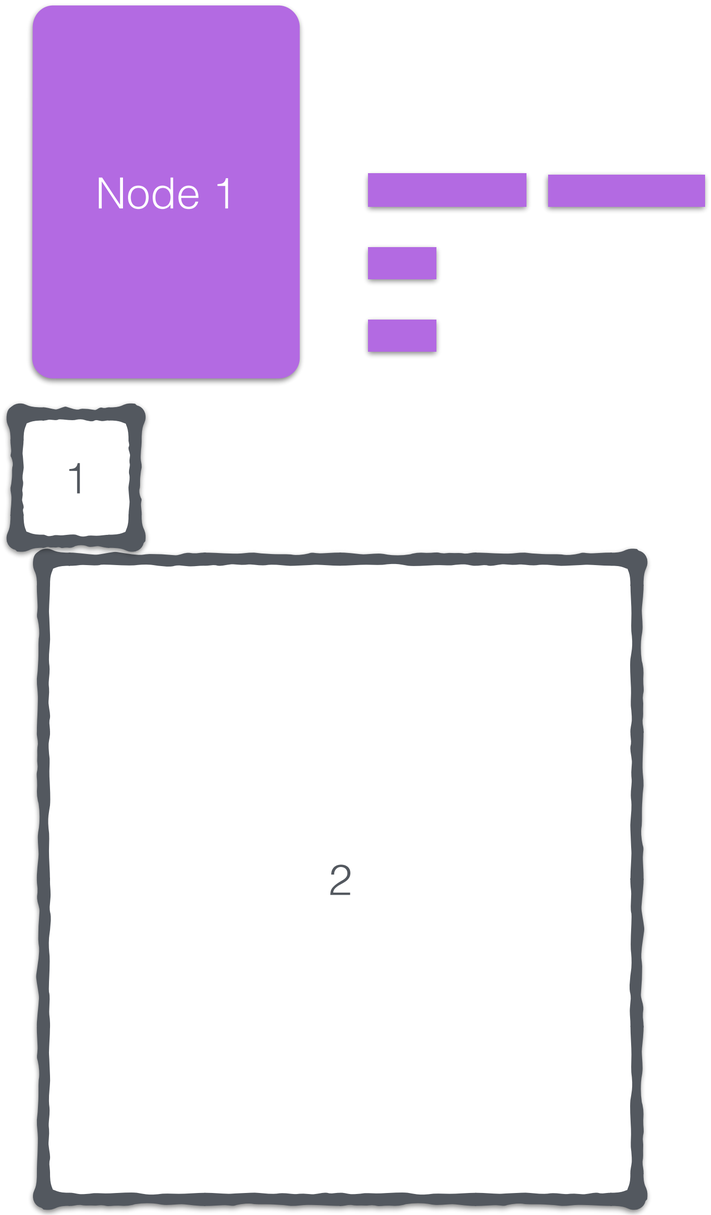
The Connector Uses Information on the Node to Make Spark Partitions



`spark.cassandra.input.split.size` 50

`Reported density is 0.5`

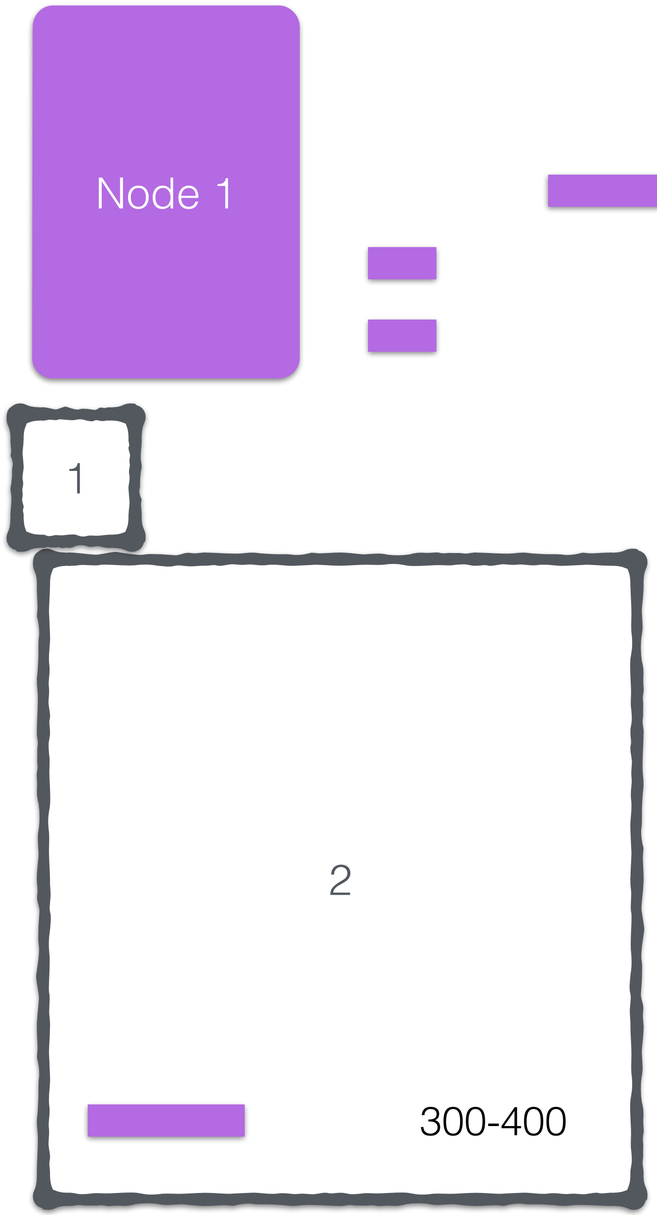
The Connector Uses Information on the Node to Make Spark Partitions



`spark.cassandra.input.split.size` 50

`Reported density is 0.5`

The Connector Uses Information on the Node to Make Spark Partitions



`spark.cassandra.input.split.size` 50

`Reported density is 0.5`

The Connector Uses Information on the Node to Make Spark Partitions

400-500

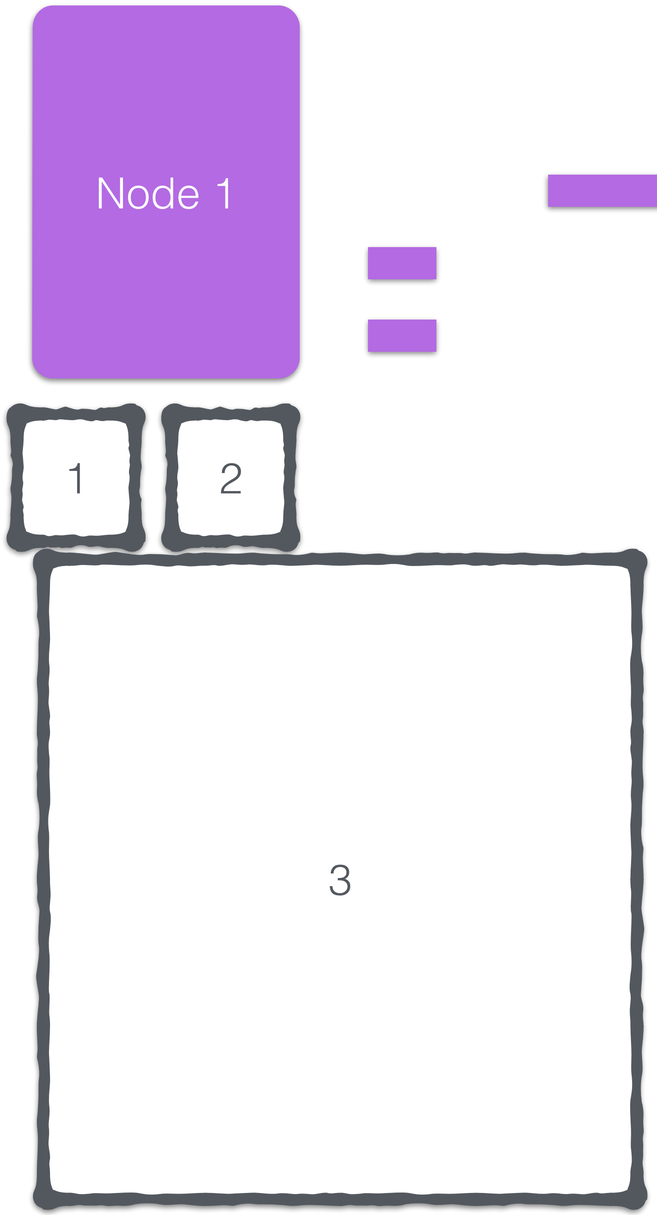
780-830

0-50

`spark.cassandra.input.split.size` 50

`Reported density is 0.5`

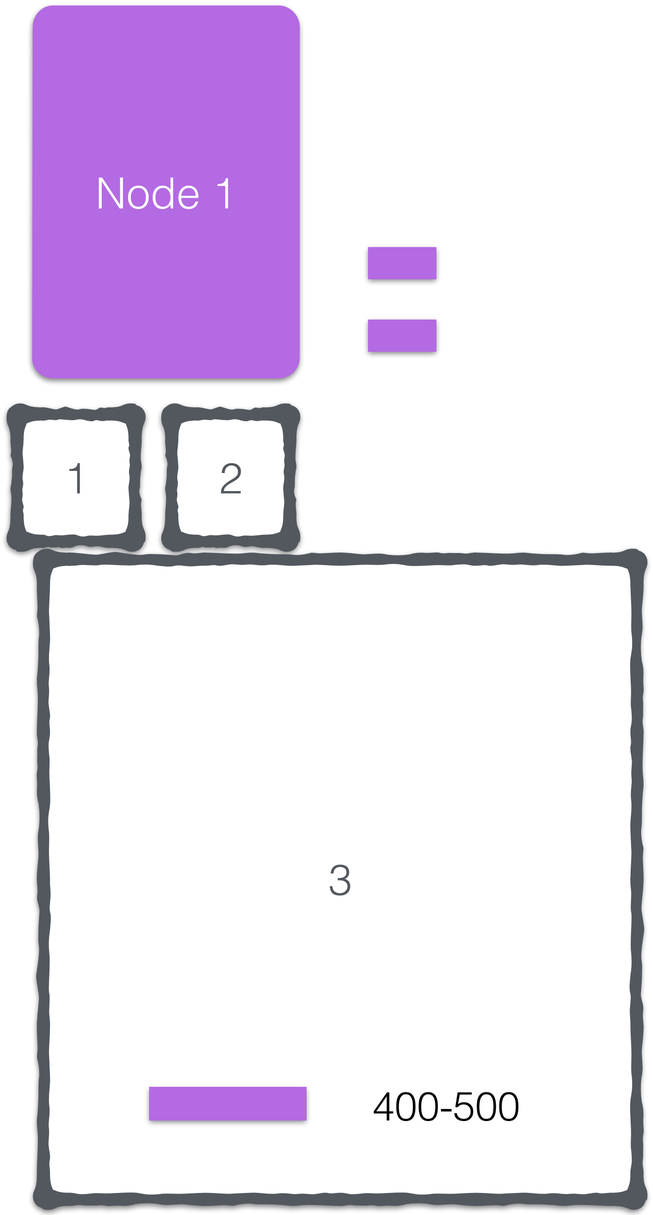
The Connector Uses Information on the Node to Make Spark Partitions



`spark.cassandra.input.split.size` 50

`Reported density is 0.5`

The Connector Uses Information on the Node to Make Spark Partitions



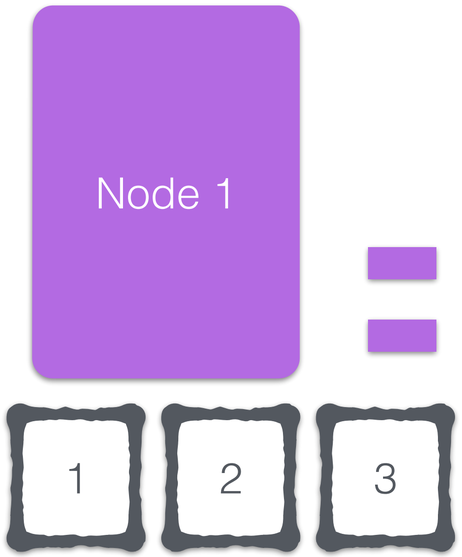
780-830

0-50

`spark.cassandra.input.split.size` 50

`Reported density is 0.5`

The Connector Uses Information on the Node to Make Spark Partitions



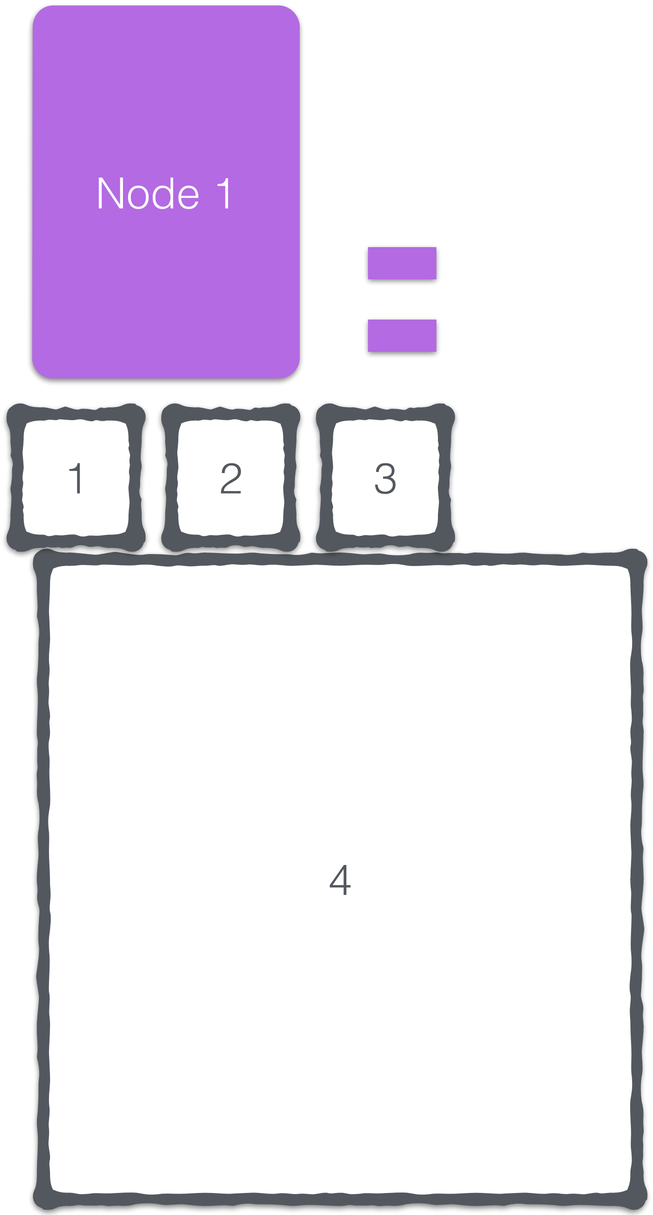
780-830

0-50

`spark.cassandra.input.split.size` 50

`Reported density is 0.5`

The Connector Uses Information on the Node to Make Spark Partitions



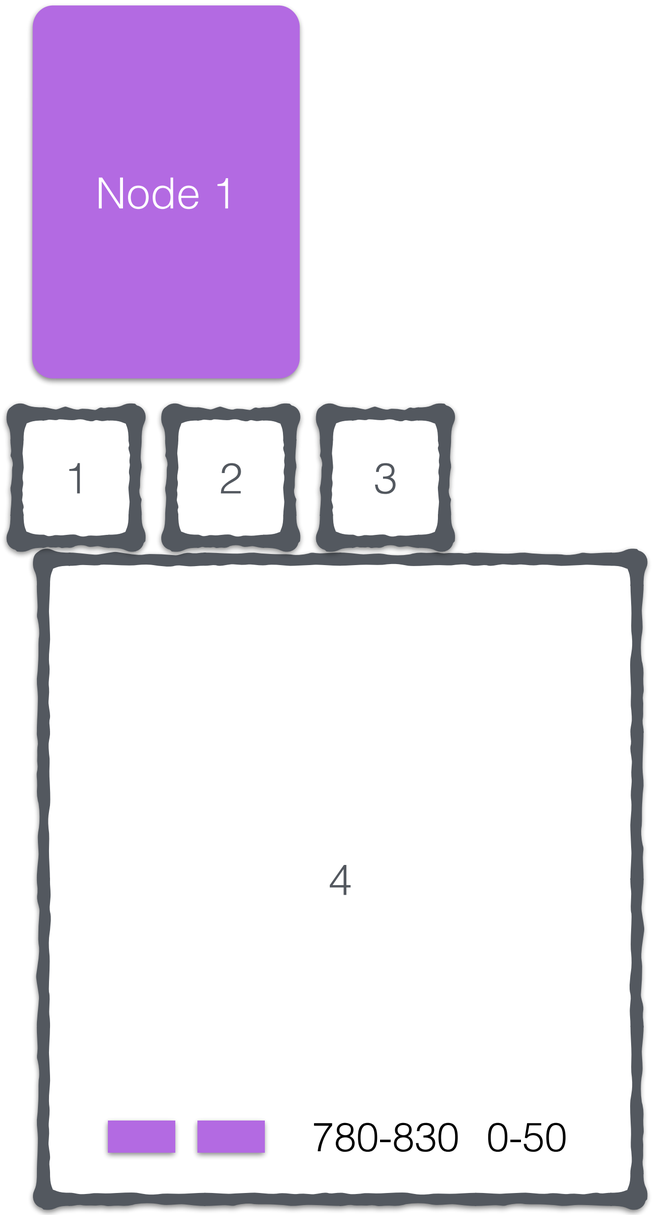
780-830

0-50

`spark.cassandra.input.split.size` 50

`Reported density is 0.5`

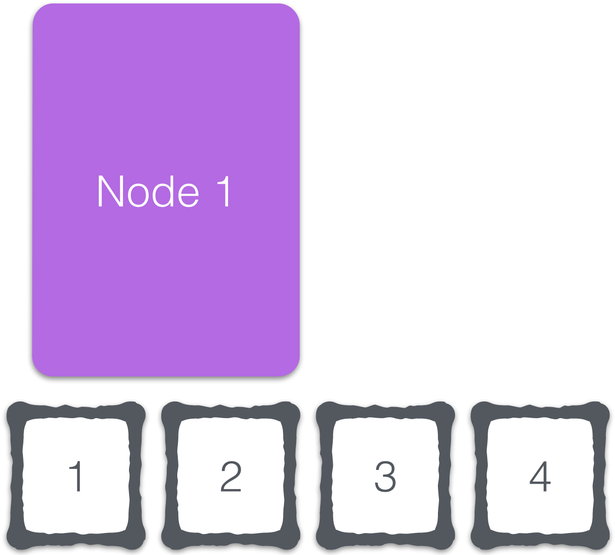
The Connector Uses Information on the Node to Make Spark Partitions



`spark.cassandra.input.split.size` 50

`Reported density is 0.5`

The Connector Uses Information on the Node to Make Spark Partitions

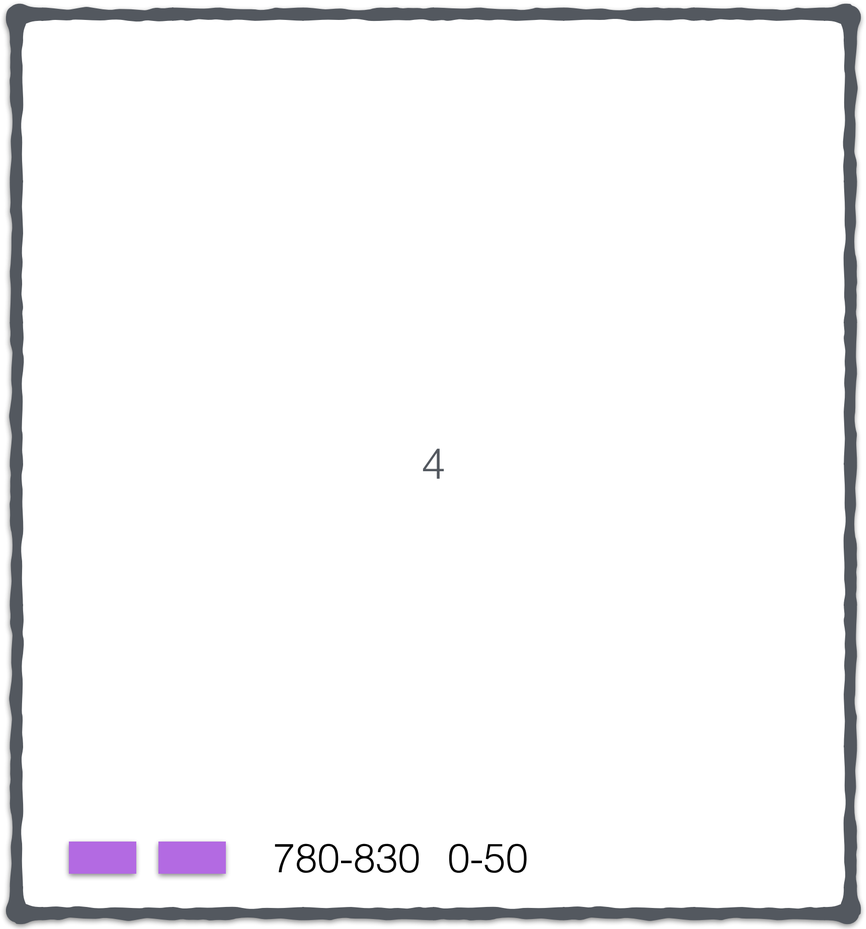


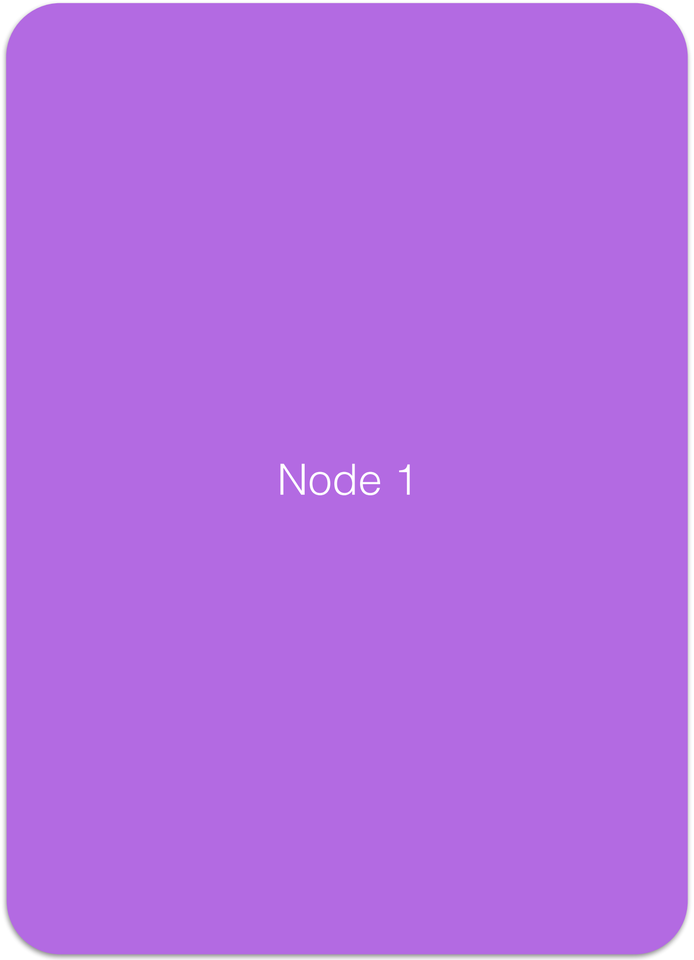
`spark.cassandra.input.split.size` 50

`Reported density is 0.5`

Data is Retrieved Using the DataStax Java Driver

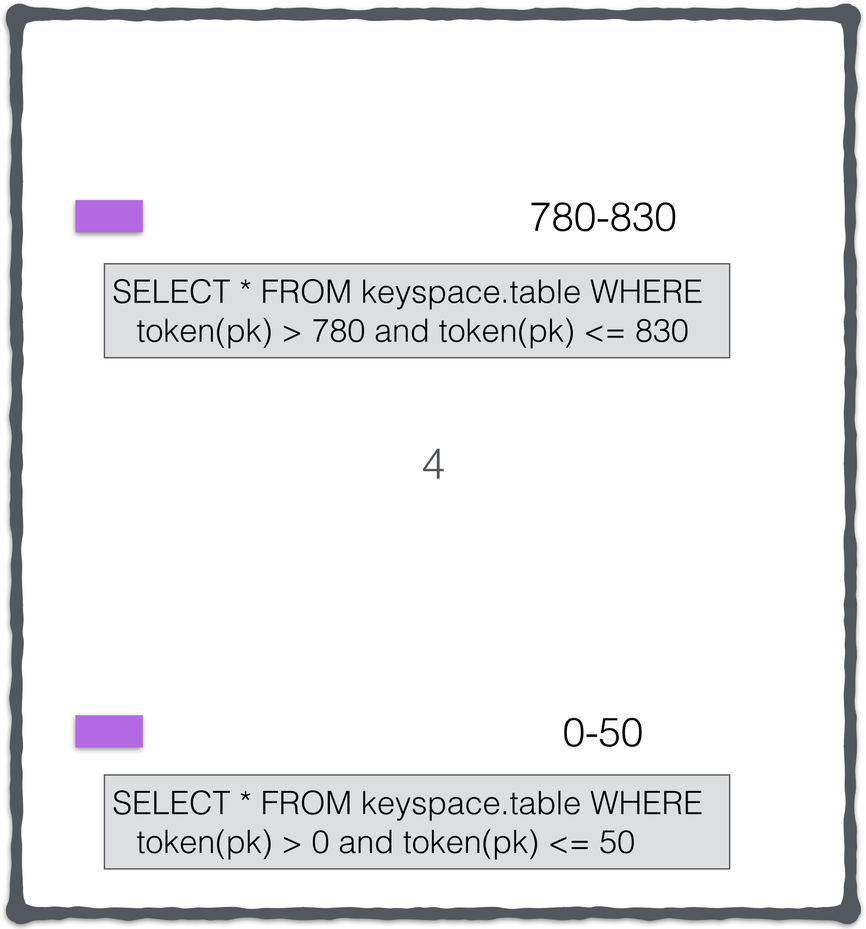
`spark.cassandra.input.page.row.size` 50

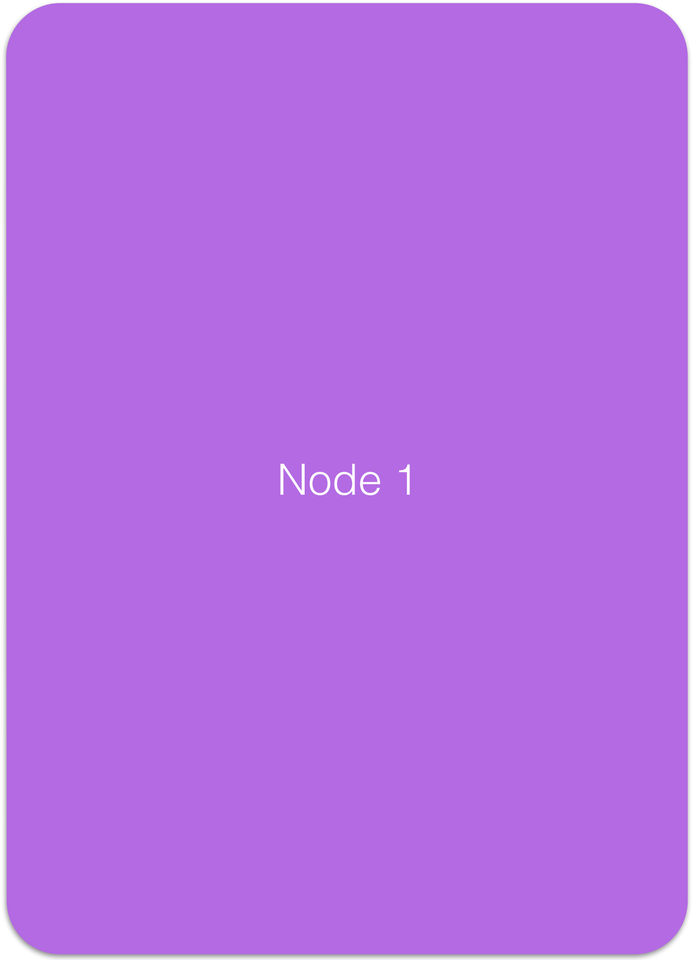




Data is Retrieved Using the DataStax Java Driver

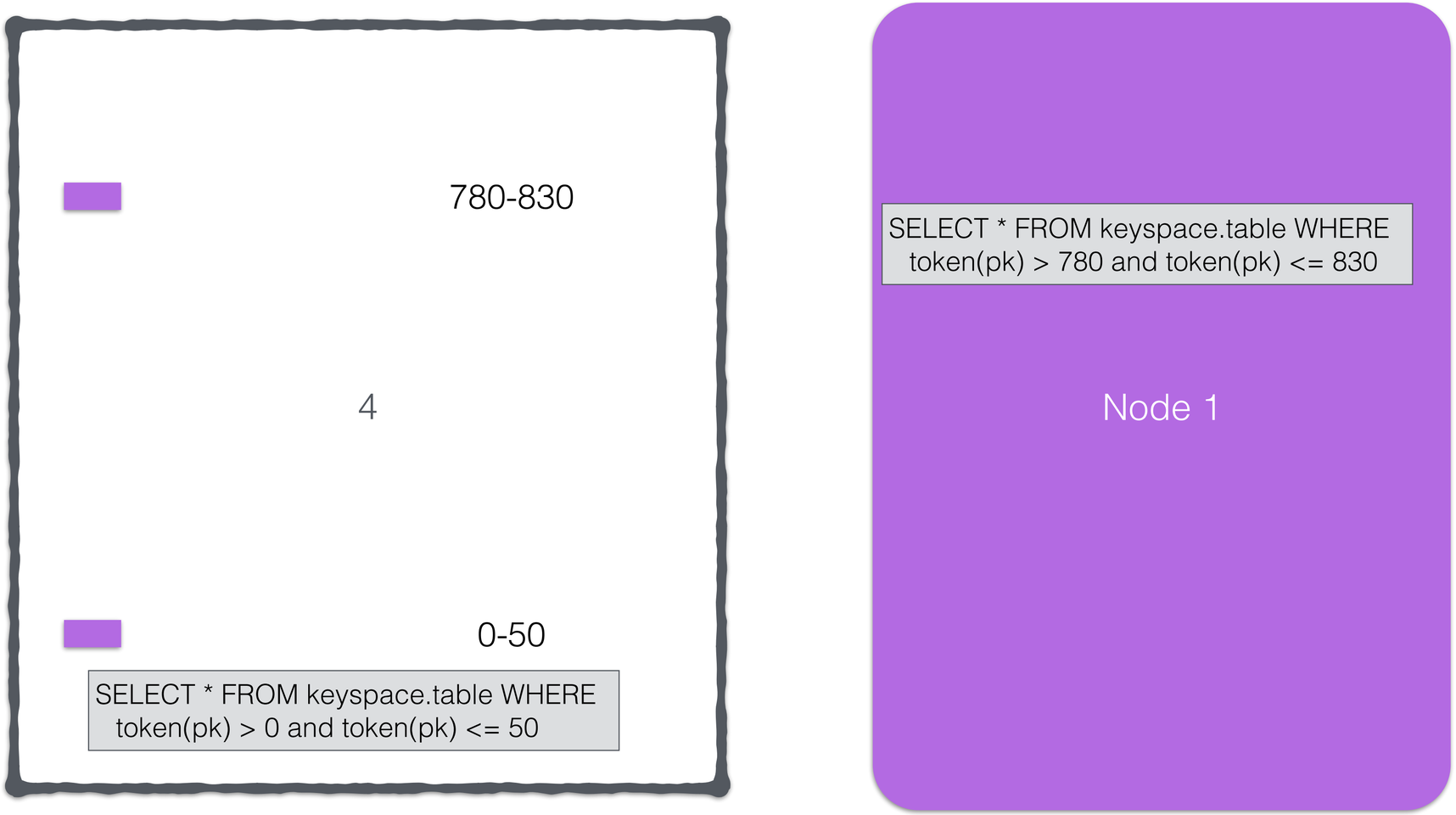
`spark.cassandra.input.page.row.size` 50





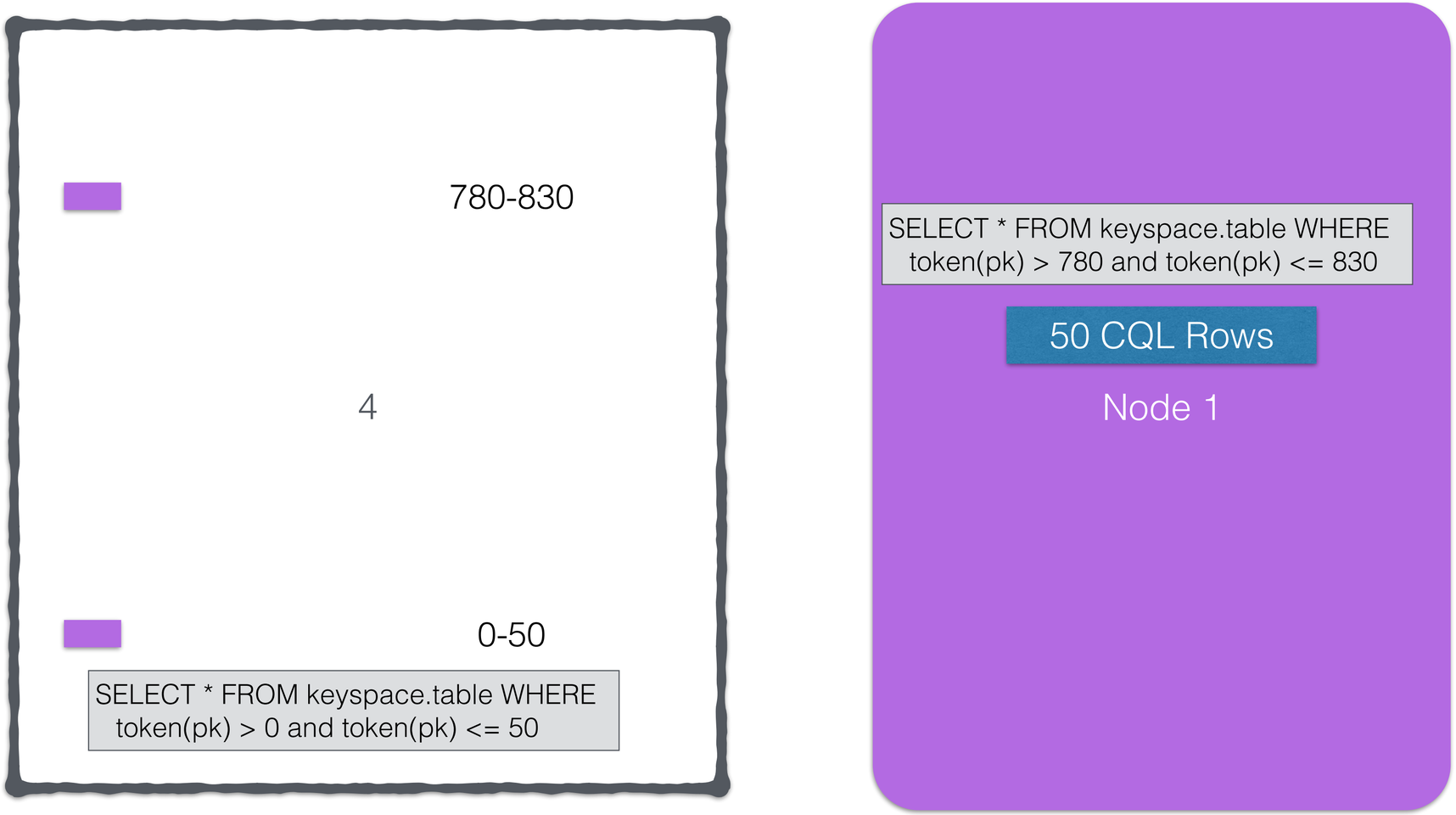
Data is Retrieved Using the DataStax Java Driver

`spark.cassandra.input.page.row.size` 50



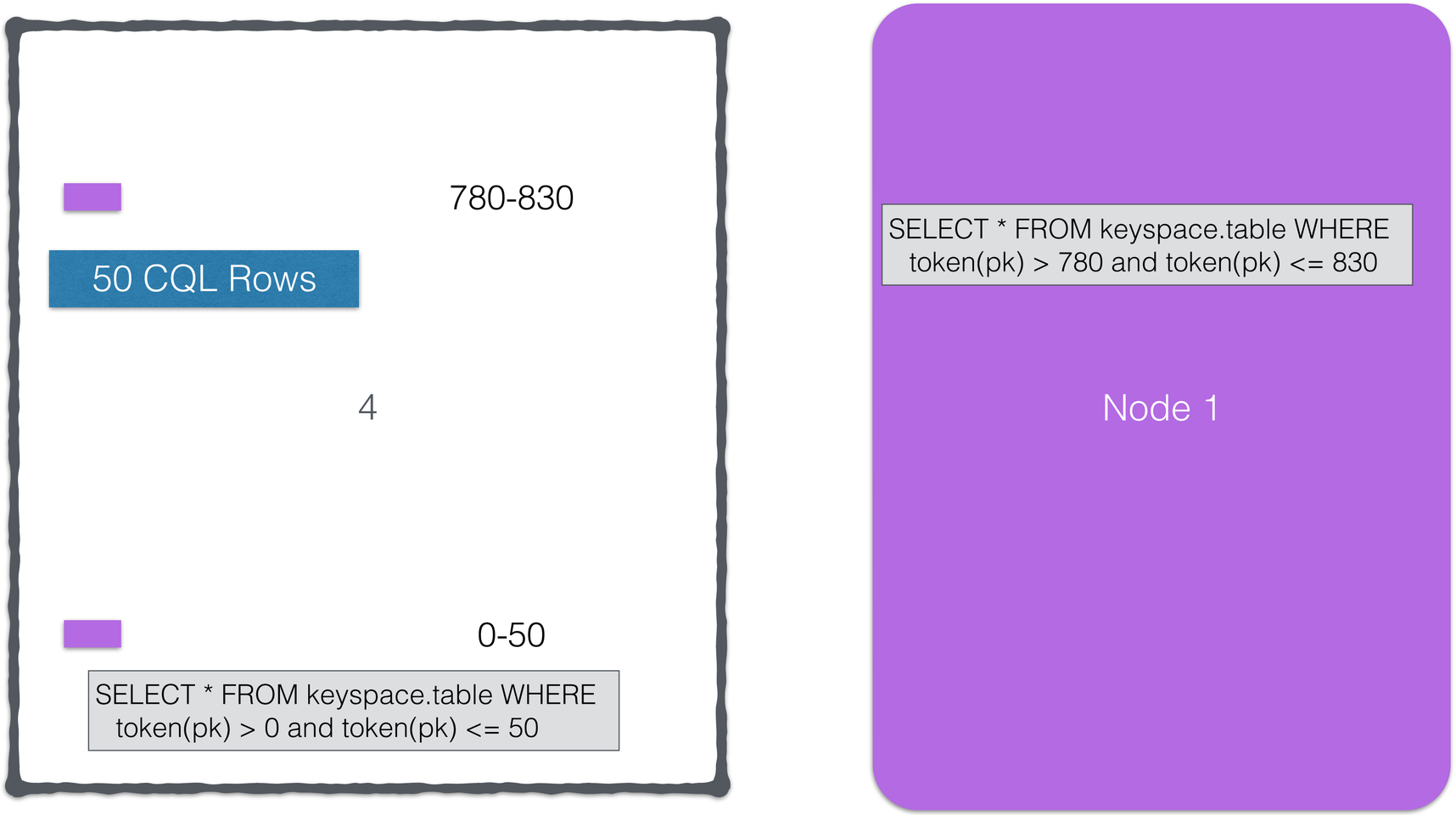
Data is Retrieved Using the DataStax Java Driver

`spark.cassandra.input.page.row.size` 50



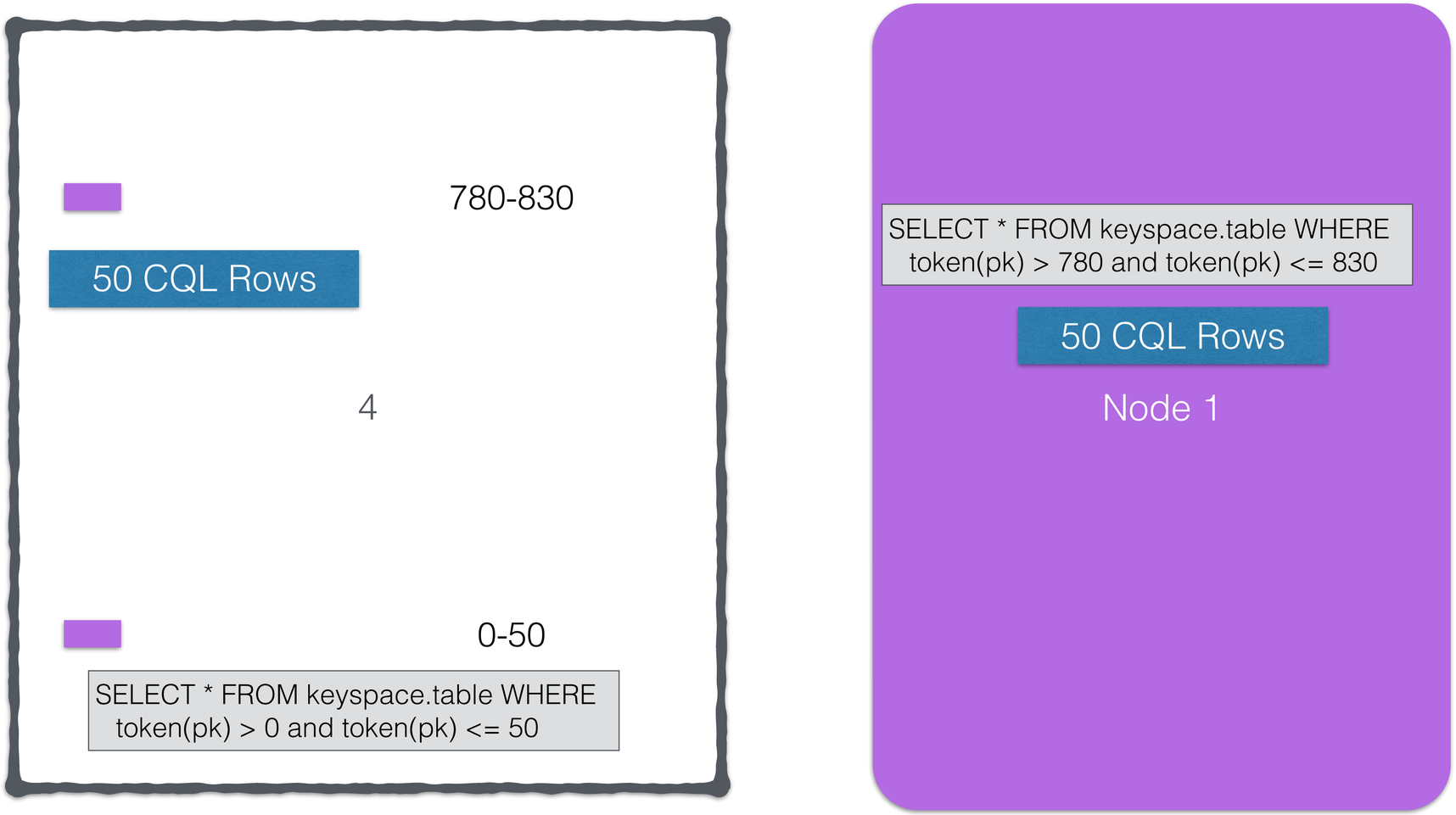
Data is Retrieved Using the DataStax Java Driver

`spark.cassandra.input.page.row.size` 50



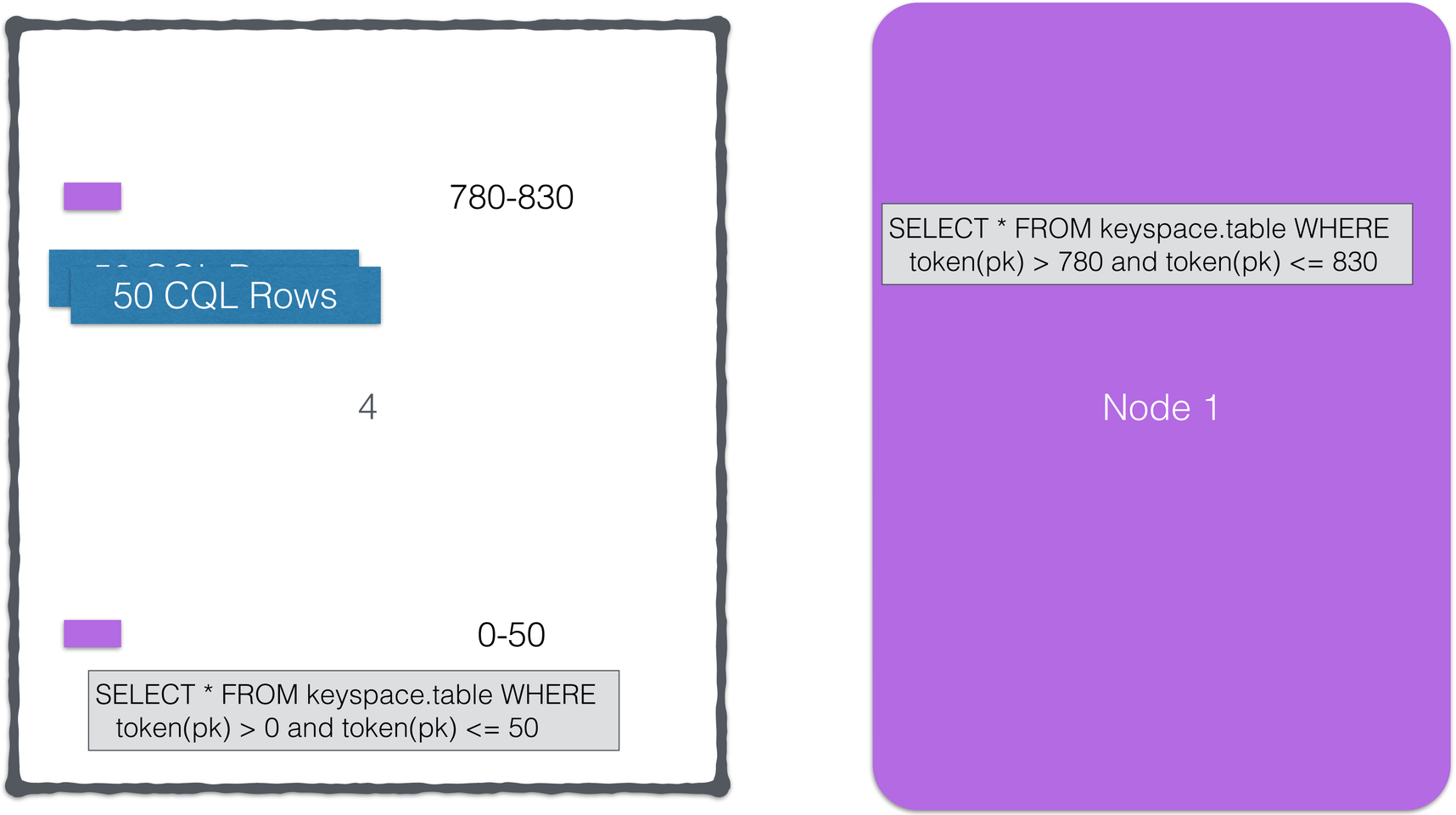
Data is Retrieved Using the DataStax Java Driver

`spark.cassandra.input.page.row.size` 50



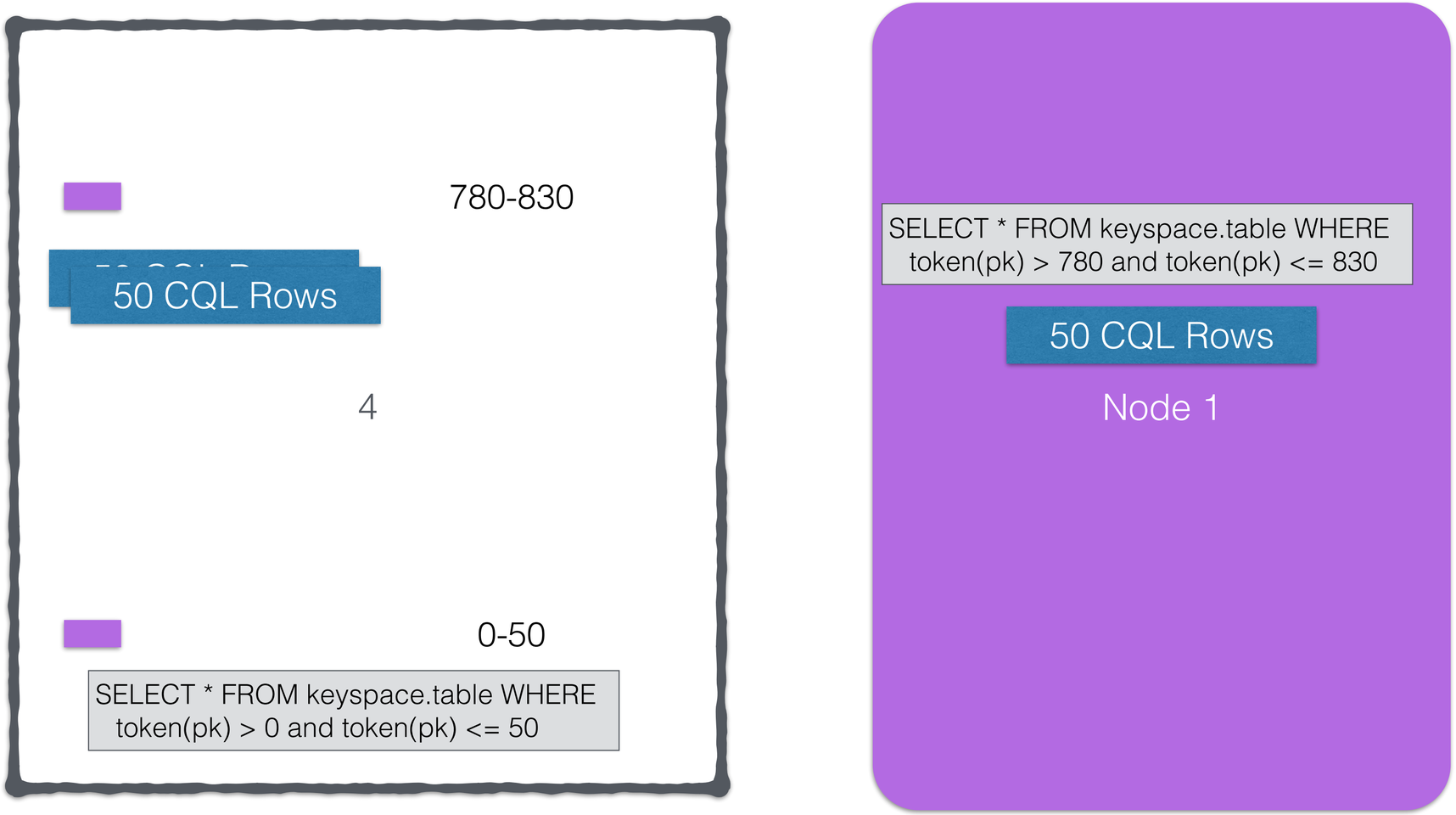
Data is Retrieved Using the DataStax Java Driver

`spark.cassandra.input.page.row.size` 50



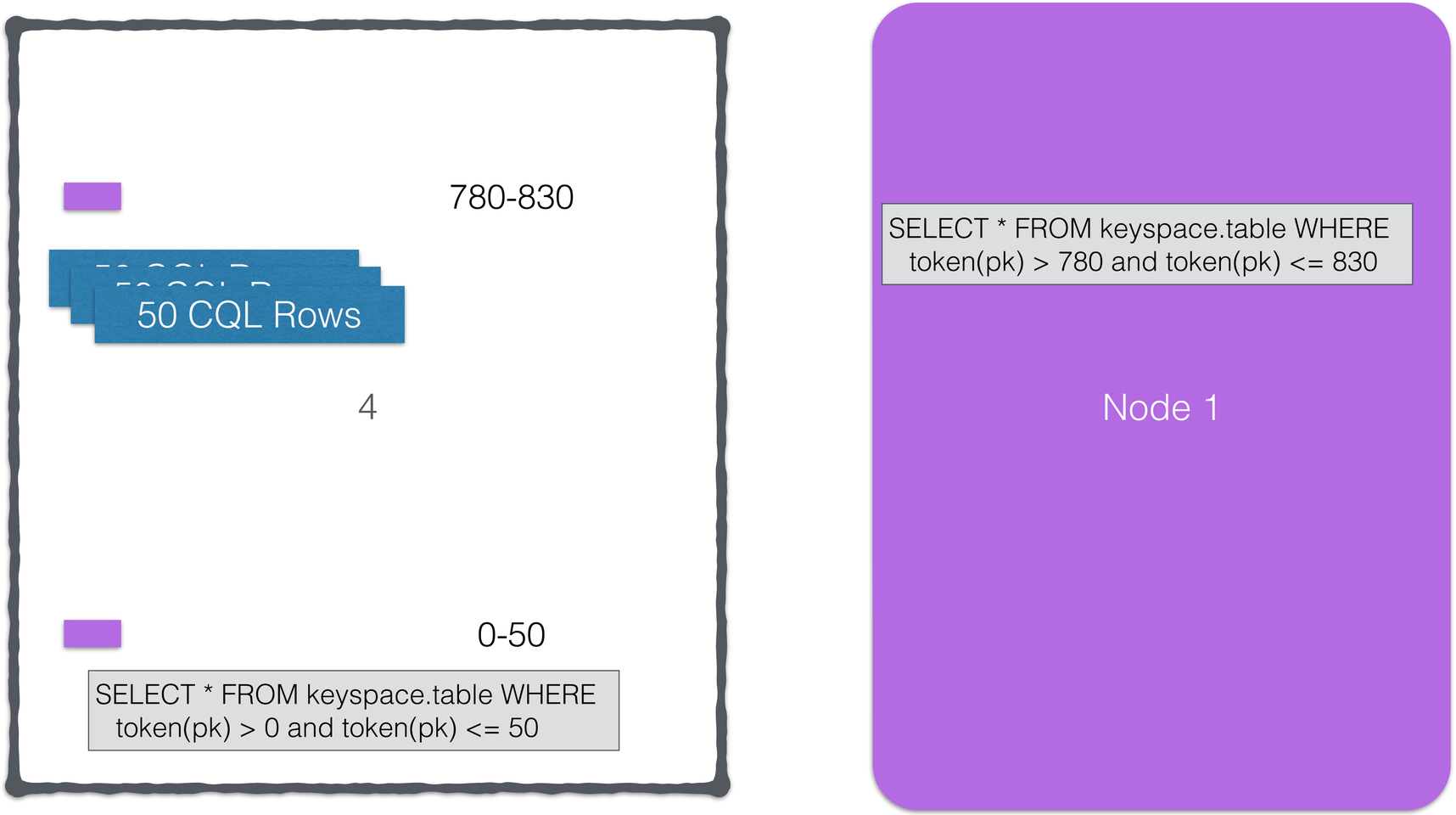
Data is Retrieved Using the DataStax Java Driver

`spark.cassandra.input.page.row.size` 50



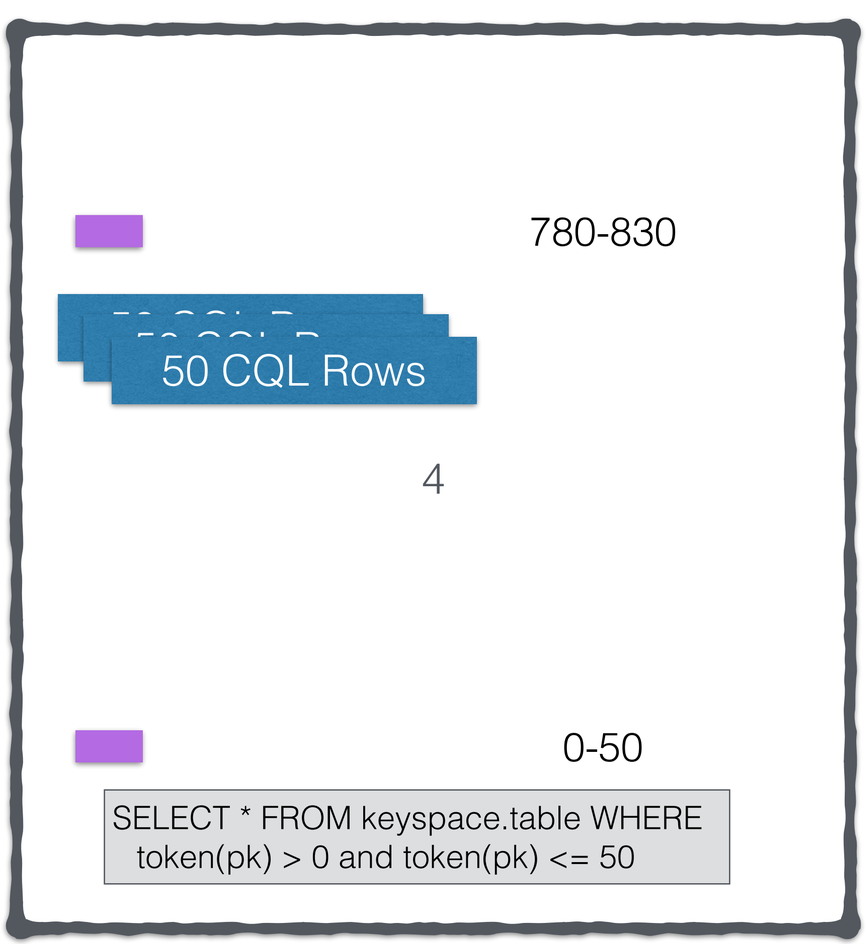
Data is Retrieved Using the DataStax Java Driver

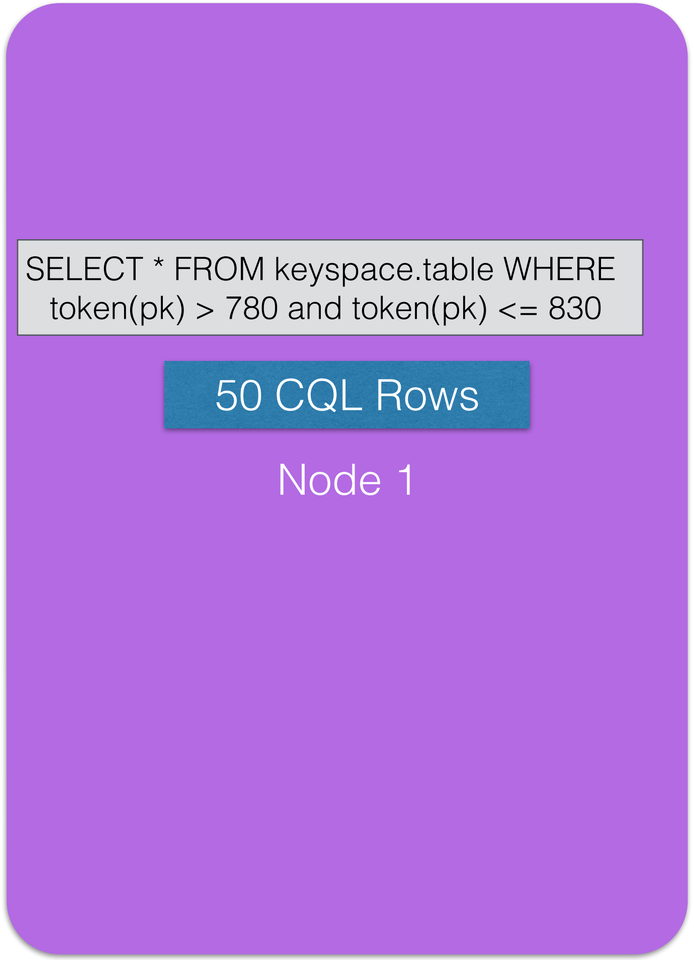
`spark.cassandra.input.page.row.size` 50



Data is Retrieved Using the DataStax Java Driver

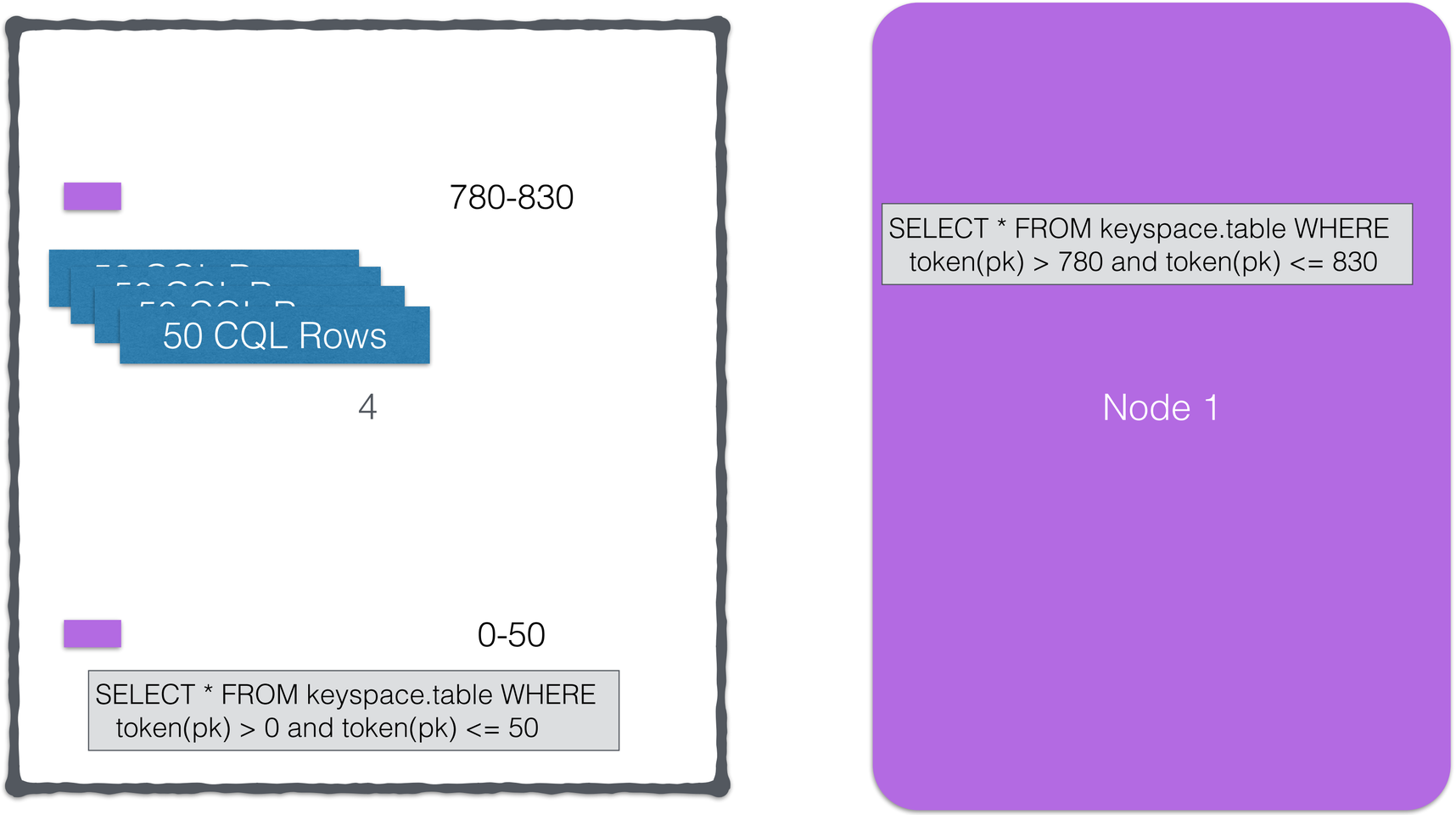
`spark.cassandra.input.page.row.size` 50





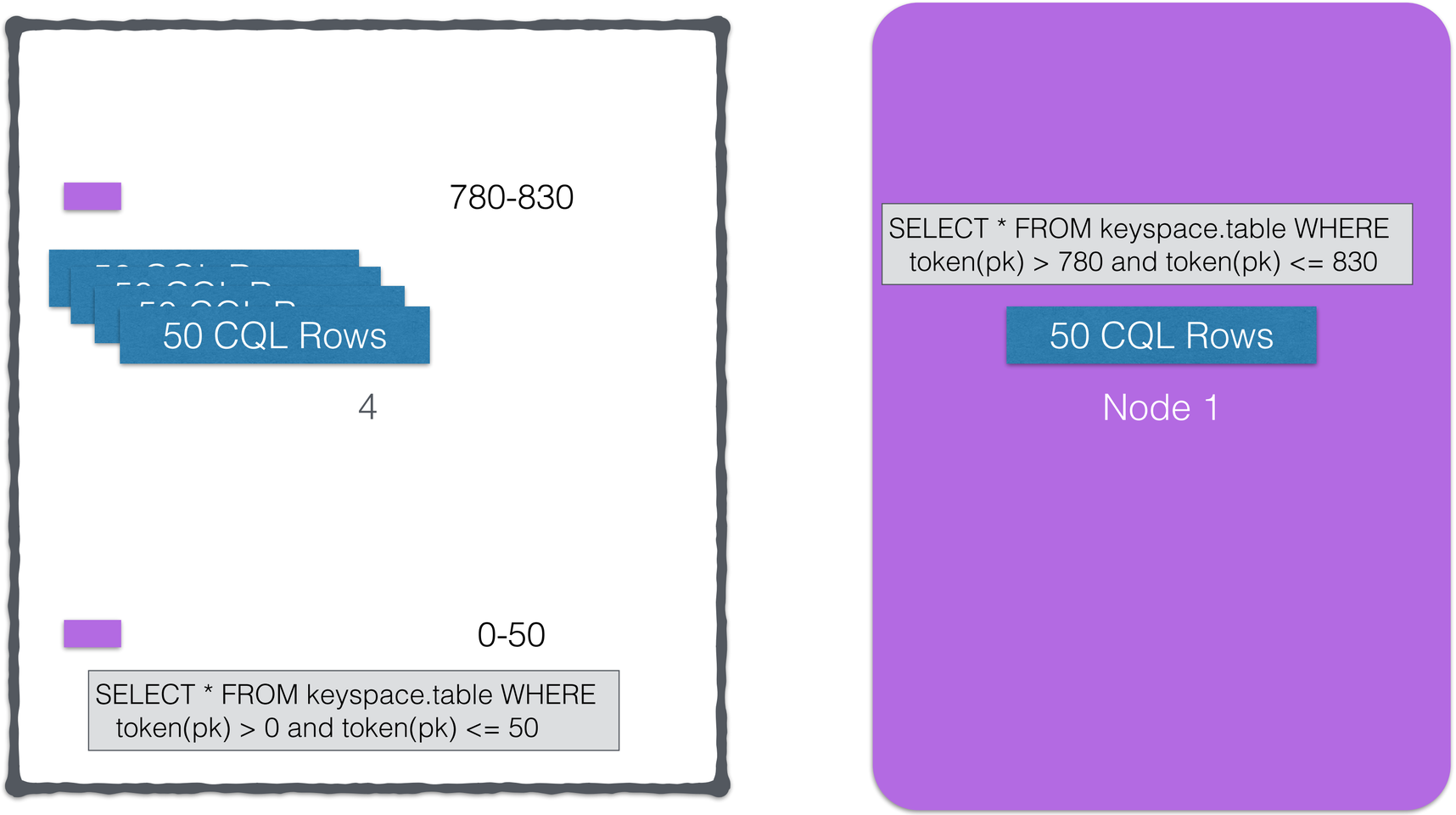
Data is Retrieved Using the DataStax Java Driver

`spark.cassandra.input.page.row.size` 50



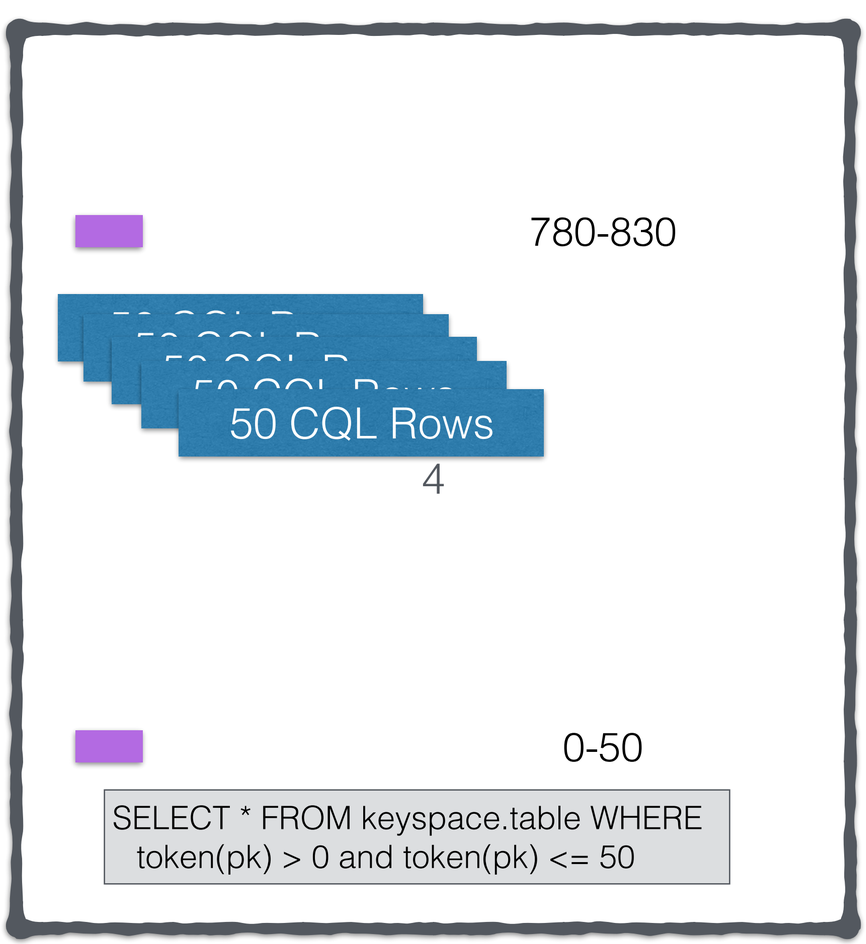
Data is Retrieved Using the DataStax Java Driver

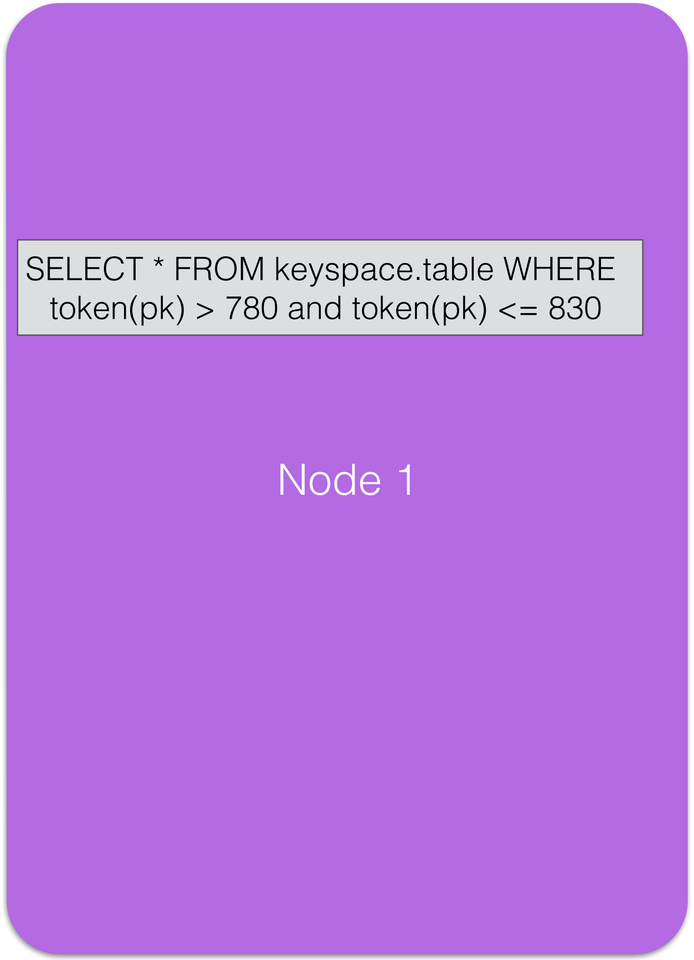
`spark.cassandra.input.page.row.size` 50



Data is Retrieved Using the DataStax Java Driver

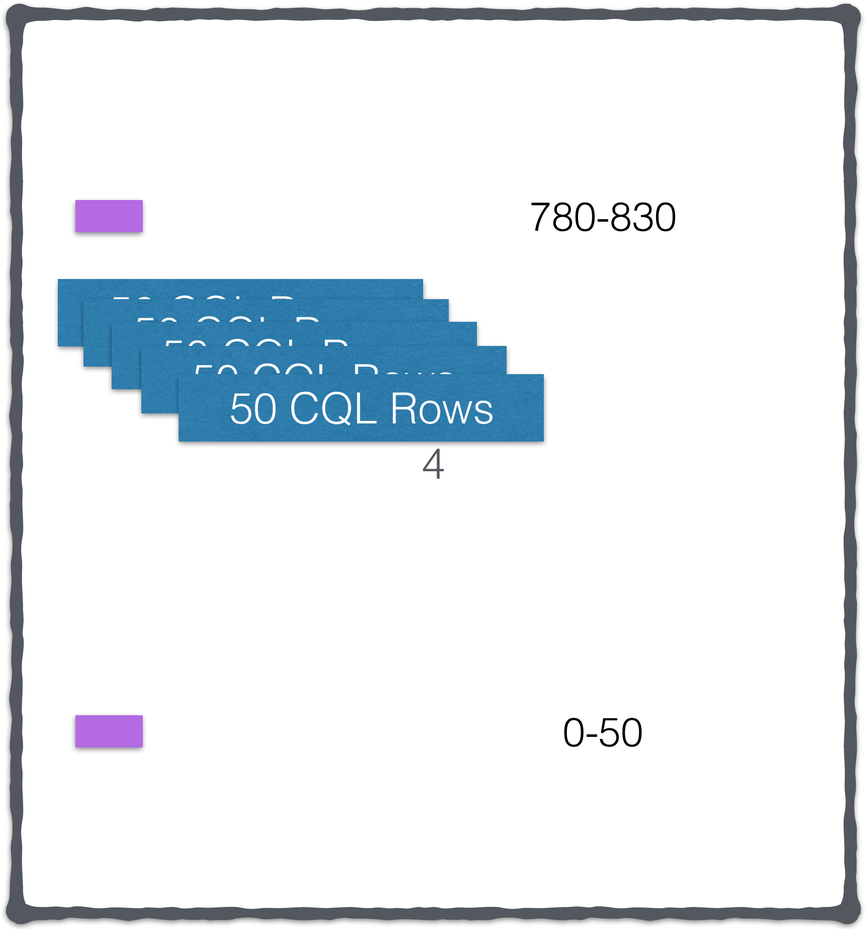
`spark.cassandra.input.page.row.size` 50





Data is Retrieved Using the DataStax Java Driver

`spark.cassandra.input.page.row.size` 50

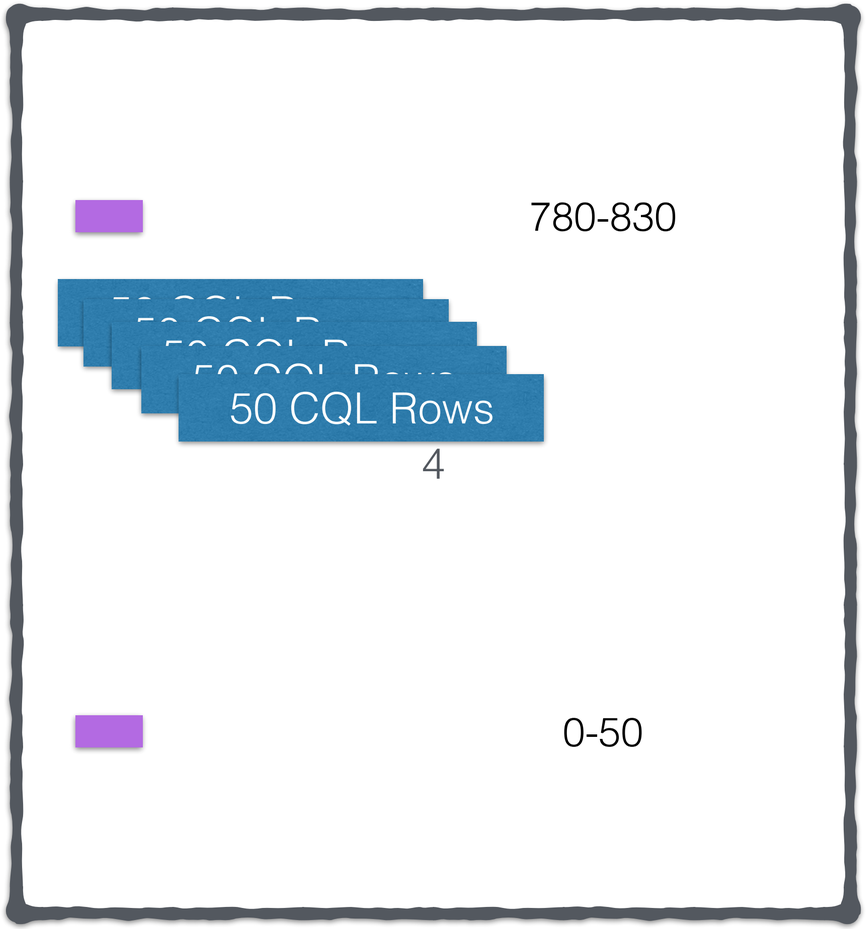


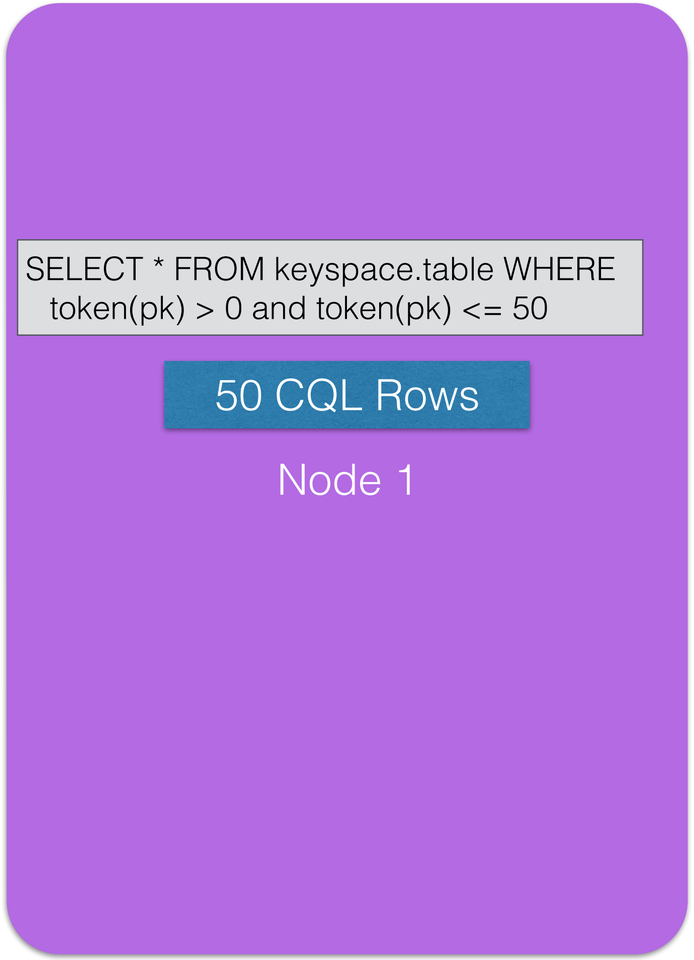
SELECT * FROM keyspace.table WHERE
token(pk) > 0 and token(pk) <= 50

Node 1

Data is Retrieved Using the DataStax Java Driver

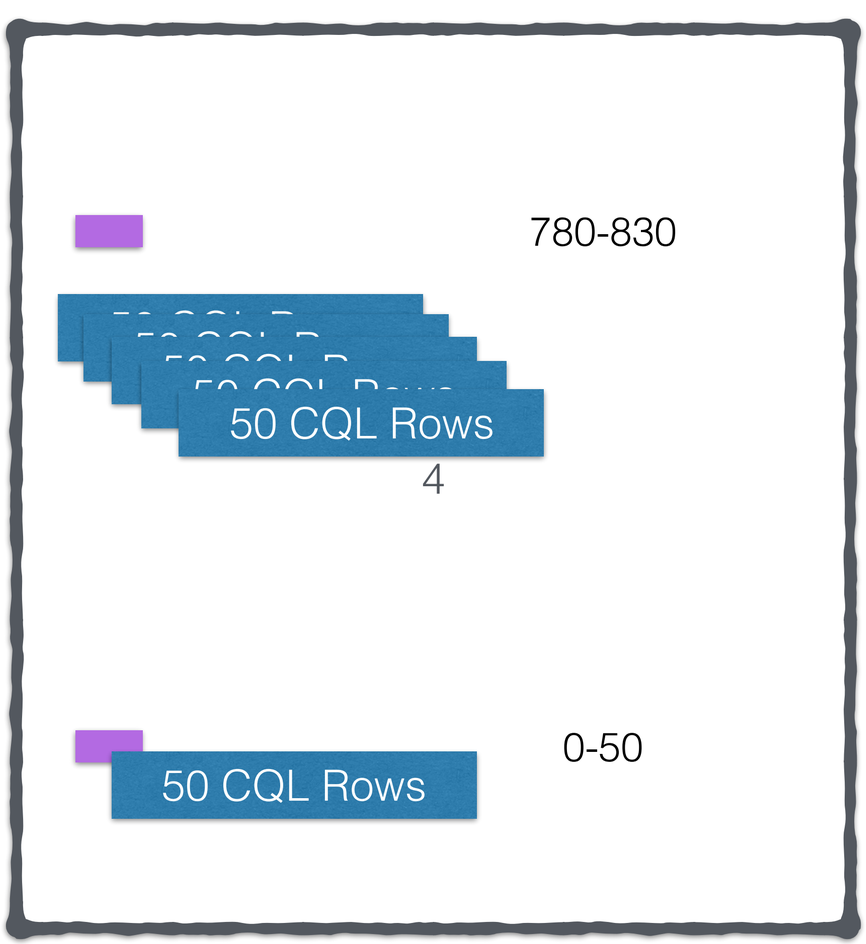
`spark.cassandra.input.page.row.size` 50





Data is Retrieved Using the DataStax Java Driver

`spark.cassandra.input.page.row.size` 50

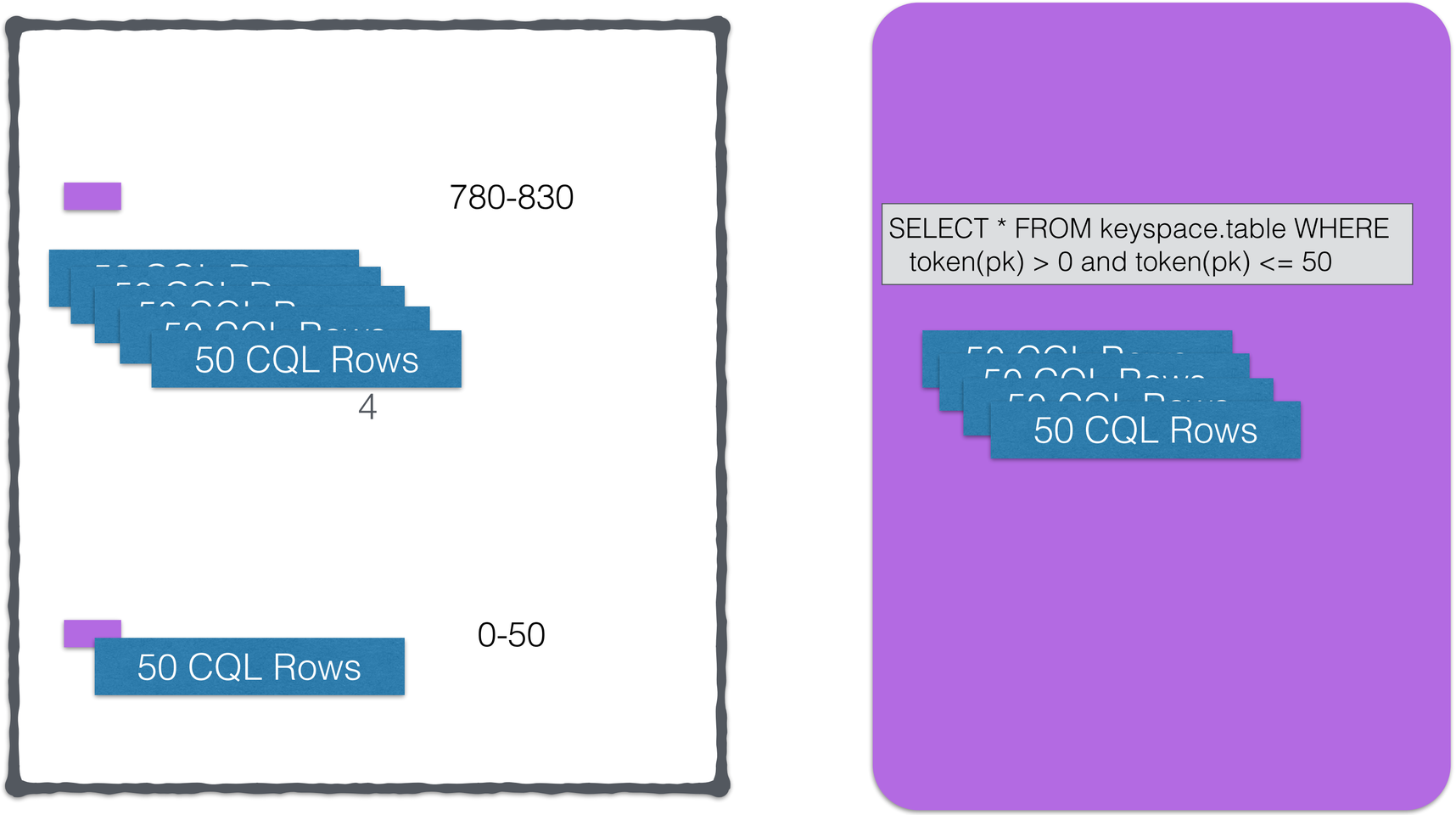


SELECT * FROM keyspace.table WHERE token(pk) > 0 and token(pk) <= 50

Node 1

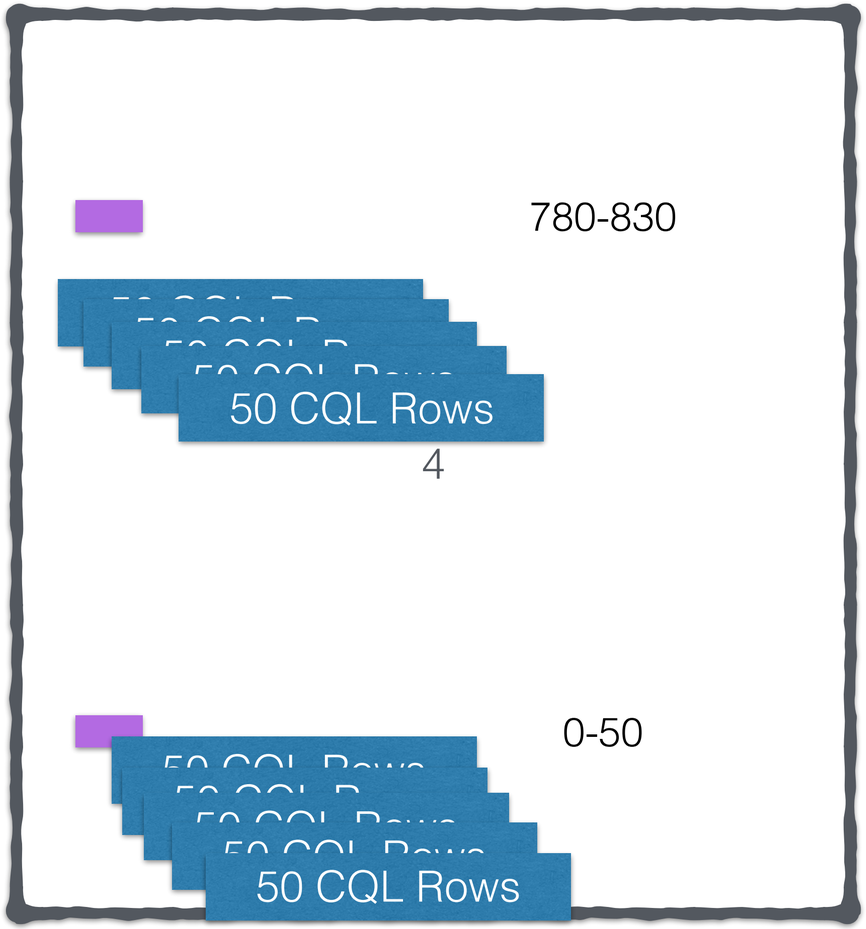
Data is Retrieved Using the DataStax Java Driver

`spark.cassandra.input.page.row.size` 50



Data is Retrieved Using the DataStax Java Driver

`spark.cassandra.input.page.row.size` 50



SELECT * FROM keyspace.table WHERE
token(pk) > 0 and token(pk) <= 50

Node 1

Spark Connector

DATASTAX

	Cassandra	Cassandra + Spark
Joins and Unions	No	Yes
Transformations	Limited	Yes
Outside Data Integration	No	Yes
Aggregations	Limited	Yes

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

Attaching to Spark and Cassandra

DATASTAX

```
// Import Cassandra-specific functions on SparkContext and RDD objects
import org.apache.spark.{SparkContext, SparkConf}
import com.datastax.spark.connector._



/** The setMaster("local") lets us run & test the job right in our IDE */
val conf = new SparkConf(true)
    .set("spark.cassandra.connection.host", "127.0.0.1")
    .setMaster(“local[*]")
    .setAppName(getClass.getName)
    // Optionally
    .set("cassandra.username", "cassandra")
    .set("cassandra.password", “cassandra")

val sc = new SparkContext(conf)
```

Weather station example

DATASTAX

```
CREATE TABLE raw_weather_data (
   wsid text,
   year int,
   month int,
   day int,
   hour int,
   temperature double,
   dewpoint double,
   pressure double,
   wind_direction int,
   wind_speed double,
   sky_condition int,
   sky_condition_text text,
   one_hour_precip double,
   six_hour_precip double,
   PRIMARY KEY ((wsid), year, month, day, hour)
) WITH CLUSTERING ORDER BY (year DESC, month DESC, day DESC, hour DESC);
```

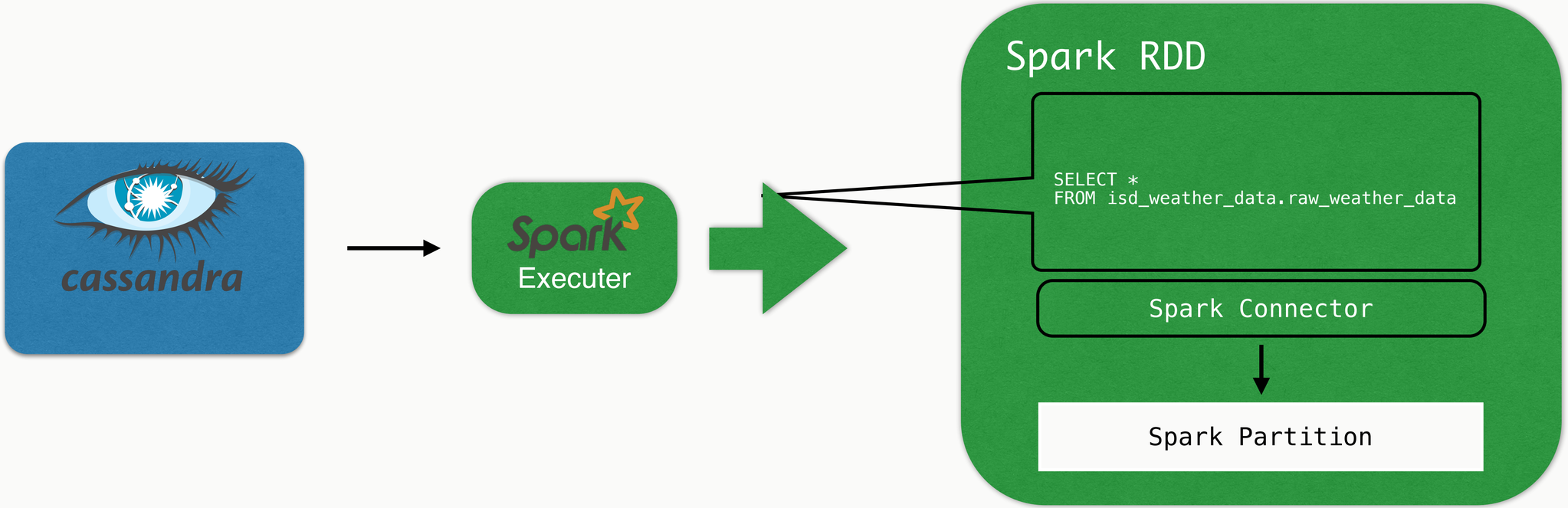
Simple example

DATASTAX

```
/** keyspace & table */
val tableRDD = sc.cassandraTable("isd_weather_data", "raw_weather_data")


/** get a simple count of all the rows in the raw_weather_data table */
val rowCount = tableRDD.count()


println(s"Total Rows in Raw Weather Table: $rowCount")
sc.stop()
```

Simple example

```
/** keyspace & table */
val tableRDD = sc.cassandraTable("isd_weather_data", "raw_weather_data")


/** get a simple count of all the rows in the raw_weather_data table */
val rowCount = tableRDD.count()


println(s"Total Rows in Raw Weather Table: $rowCount")
sc.stop()
```



Using CQL

DATASTAX

```
SELECT temperature
FROM raw_weather_data
WHERE wsid = '724940:23234'
AND year = 2008
AND month = 12
AND day = 1;
```

```
val cqlRRD = sc.cassandraTable("isd_weather_data", "raw_weather_data")
  .select("temperature")
  .where("wsid = ? AND year = ? AND month = ? AND DAY = ?",
    "724940:23234", "2008", "12", “1")
```

Using SQL

DATASTAX

Wait wut?

```
SELECT wsid, year, month, day, max(temperature) high
FROM raw_weather_data
GROUP BY wsid, year, month, day;
```

```
SELECT w.name, w.lat, w.long
FROM raw_weather_data r,
JOIN weather_station w
ON w.id = r.wsid
GROUP BY r.wsid;
```

Python!

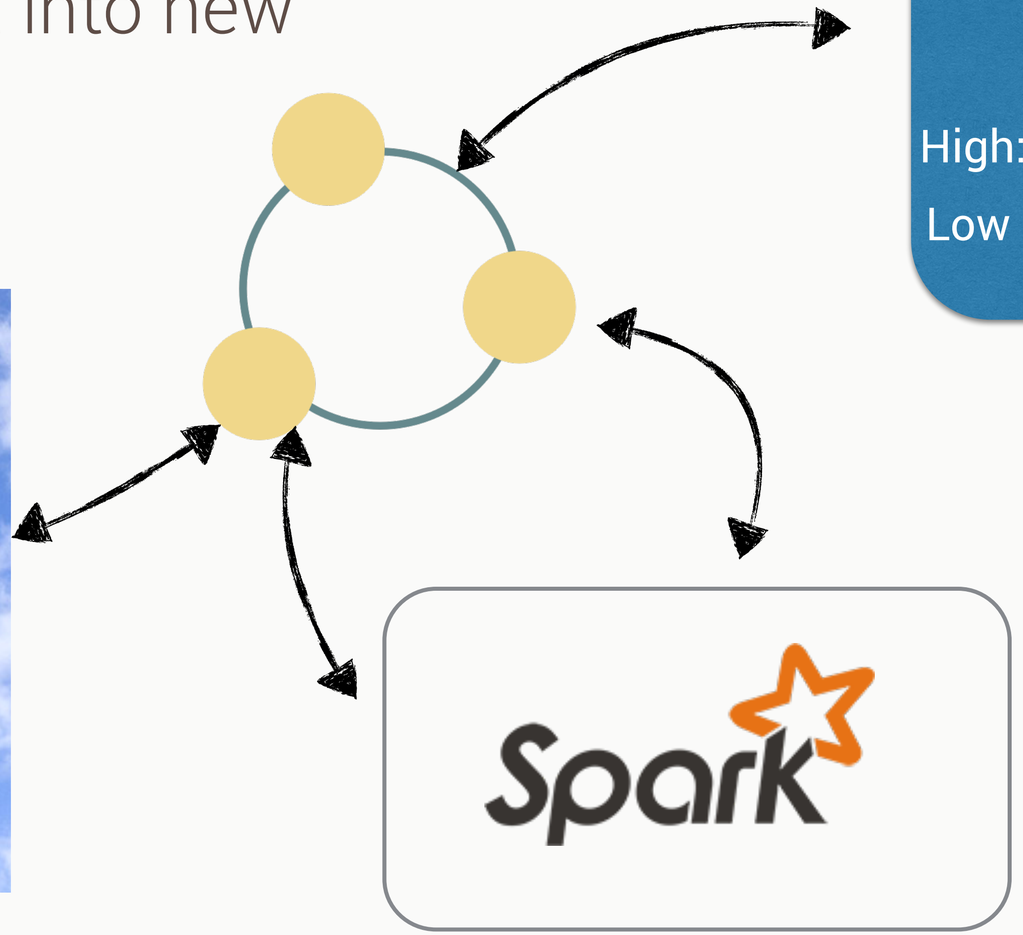
DATASTAX

```
from pyspark_cassandra import CassandraSparkContext, Row
from pyspark import SparkContext, SparkConf
from pyspark.sql import SQLContext # needed for toDF()

conf = SparkConf() \
    .setAppName("Weather App") \
    .setMaster("spark://127.0.0.1:7077") \
    .set("spark.cassandra.connection.host", "127.0.0.1")

sc = CassandraSparkContext(conf=conf)
sql = SQLContext(sc)

// Count unique weather stations
temps = sc.cassandraTable("isd_weather_data", "raw_weather_data").toDF()
food_count = temps.select("temperature").groupBy("wsid").count()
```

Weather Station Analysis

DATASTAX

- Weather station collects data
- Cassandra stores in sequence
- Spark rolls up data into new tables



Windsor California

July 1, 2014

High: 73.4

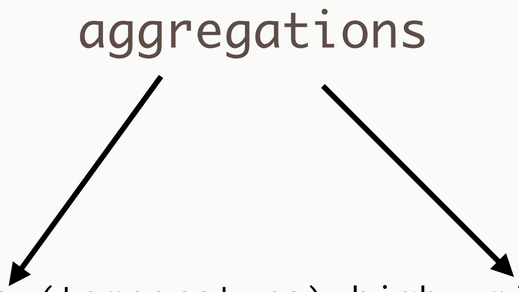
Low : 51.4

Roll-up table(SparkSQL example)

DATASTAX

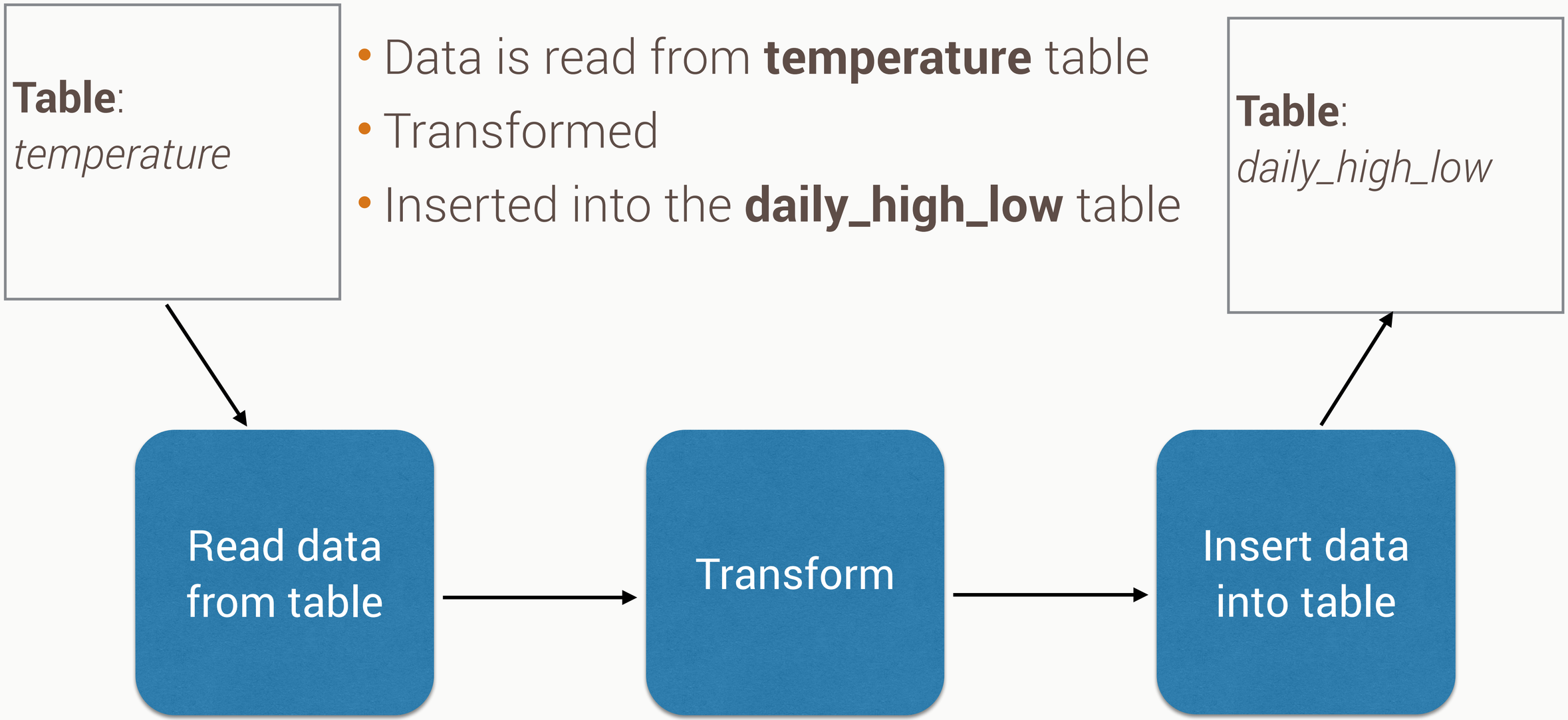
```
CREATE TABLE daily_aggregate_temperature (
   wsid text,
   year int,
   month int,
   day int,
   high double,
   low double,
   mean double,
   variance double,
   stdev double,
   PRIMARY KEY ((wsid), year, month, day)
) WITH CLUSTERING ORDER BY (year DESC, month DESC, day DESC);
```

- Weather Station Id and Date are unique
- High and low temp for each day




```
spark-sql> INSERT INTO TABLE
         > daily_aggregate_temperature
         > SELECT wsid, year, month, day, max(temperature) high, min(temperature) low
         > FROM raw_weather_data
         > GROUP BY wsid, year, month, day;
OK
Time taken: 2.345 seconds
```

What just happened



Spark Streaming

Spark Versus Spark Streaming

zillions of bytes

gigabytes per second

Spark Streaming

DATASTAX

- Kafka
- Flume
- HDFS
- Kinesis, S3
- Twitter

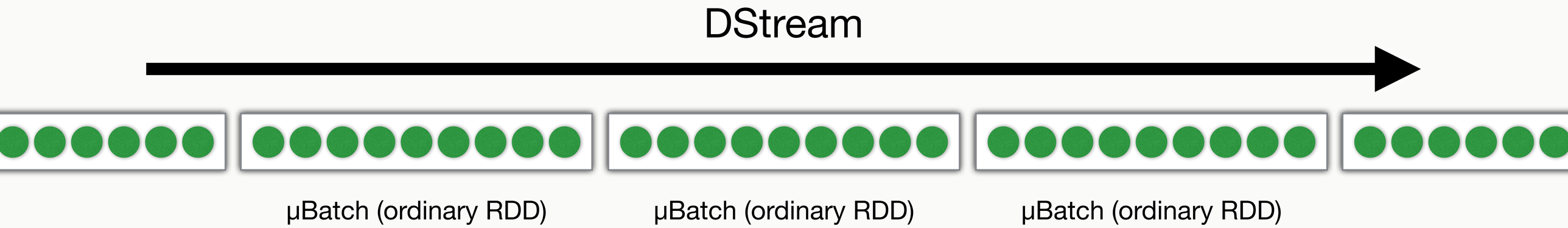
- HDFS
- Databases
- Dashboards

Spark

DStream - Micro Batches

DATASTAX

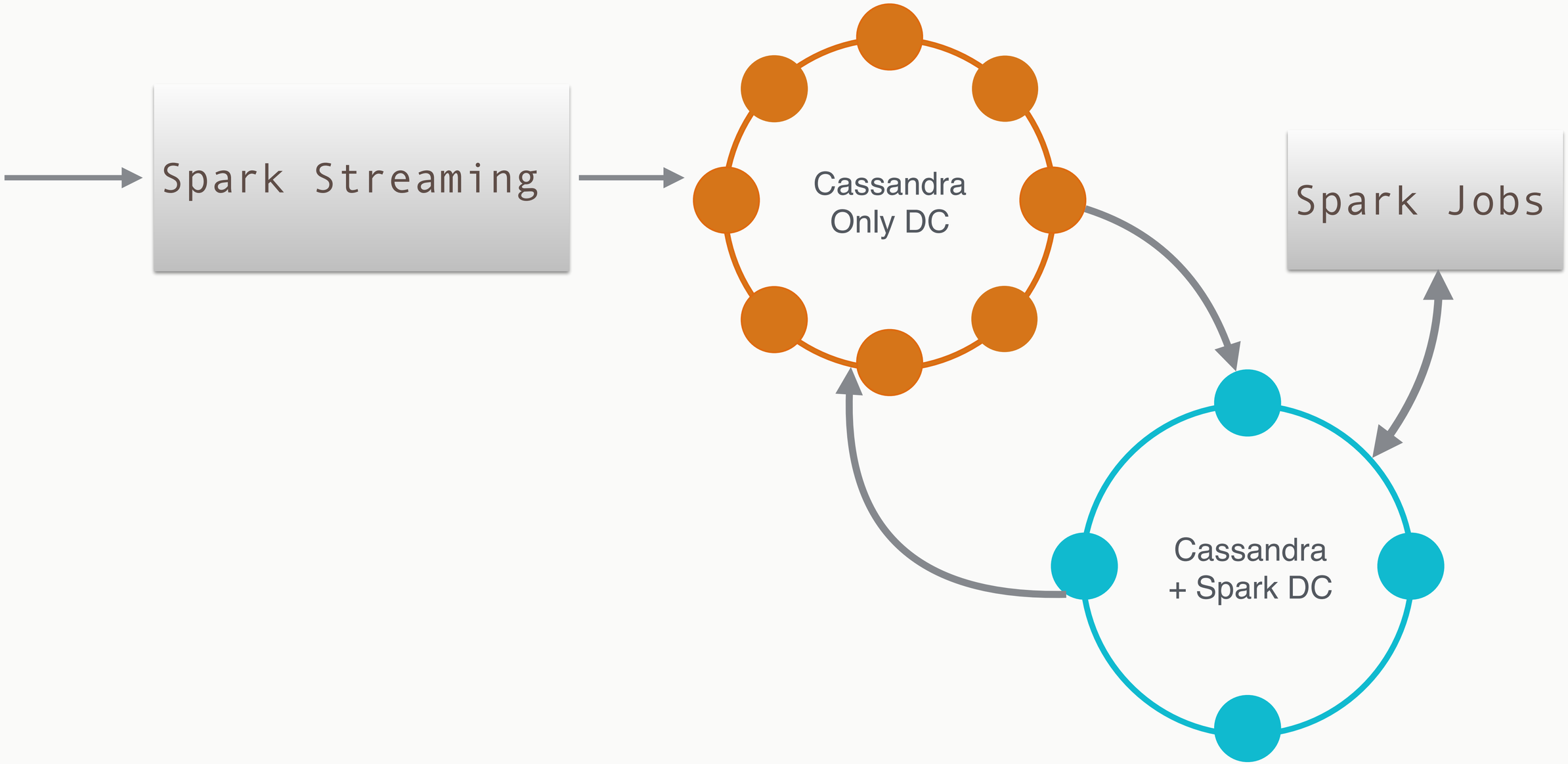
- Continuous sequence of micro batches
  - More complex processing models are possible with less effort
  - Streaming computations as a series of deterministic batch computations on small time intervals



Processing of DStream = Processing of μBatches, RDDs

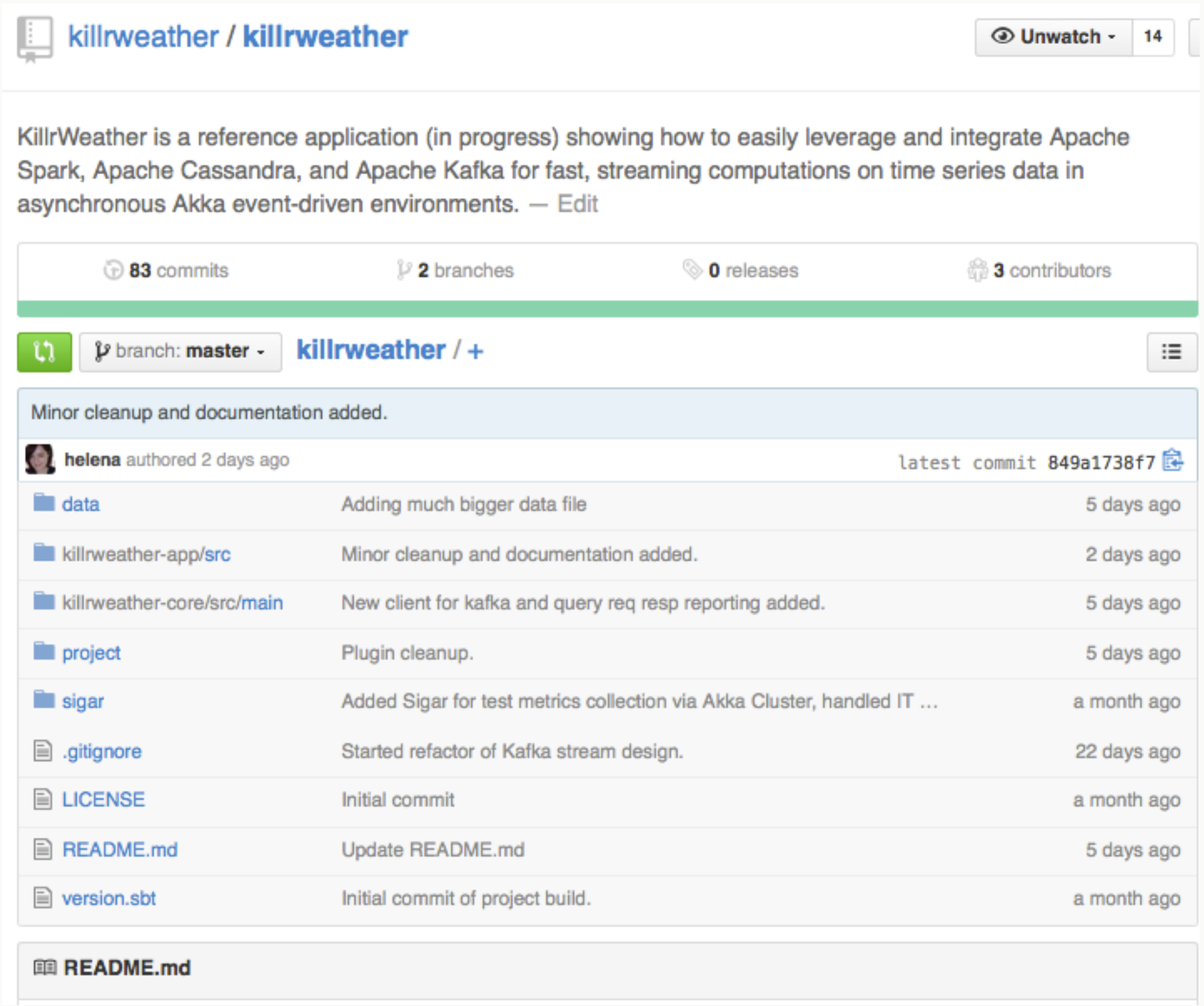
Spark

Now what?


Spark Streaming
Cassandra
Only DC
Spark Jobs
Cassandra
+ Spark DC

You can do this at home!

DATASTAX



https://github.com/killrweather/killrweather


DATASTAX
Cassandra Summit 2015
World's Largest Gathering of Cassandra Users
September 22 - 24, 2015 | Santa Clara, CA
FREE GENERAL PASSES Register today!
Visit http://datastax.com/cassandrasummit2015 for more information

Thank you!

Bring the questions

Follow me on twitter
@PatrickMcFadin