



Analyzing Time Series Data with Apache Spark and Cassandra

Patrick McFadin

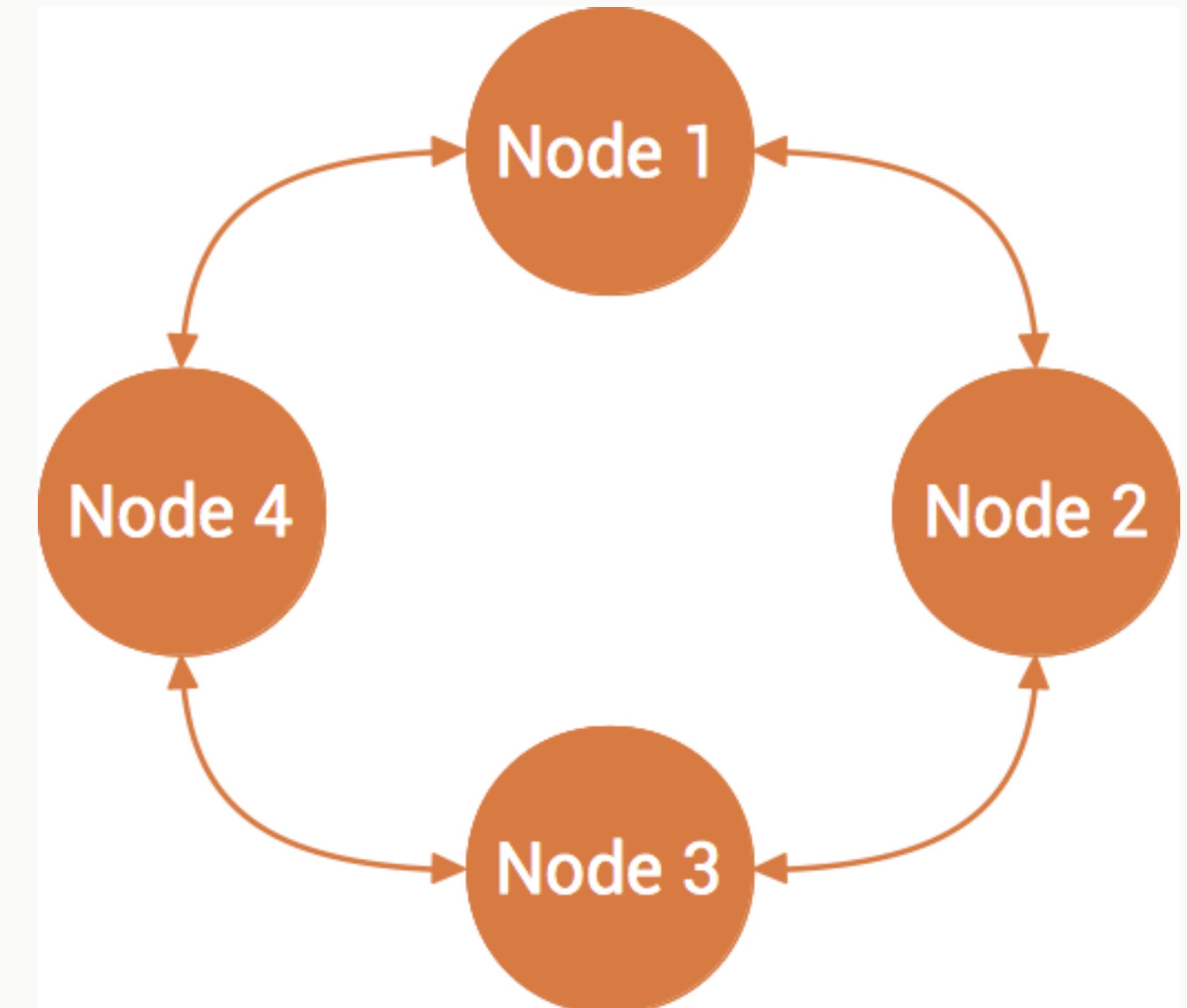
Chief Evangelist for Apache Cassandra, DataStax

@PatrickMcFadin

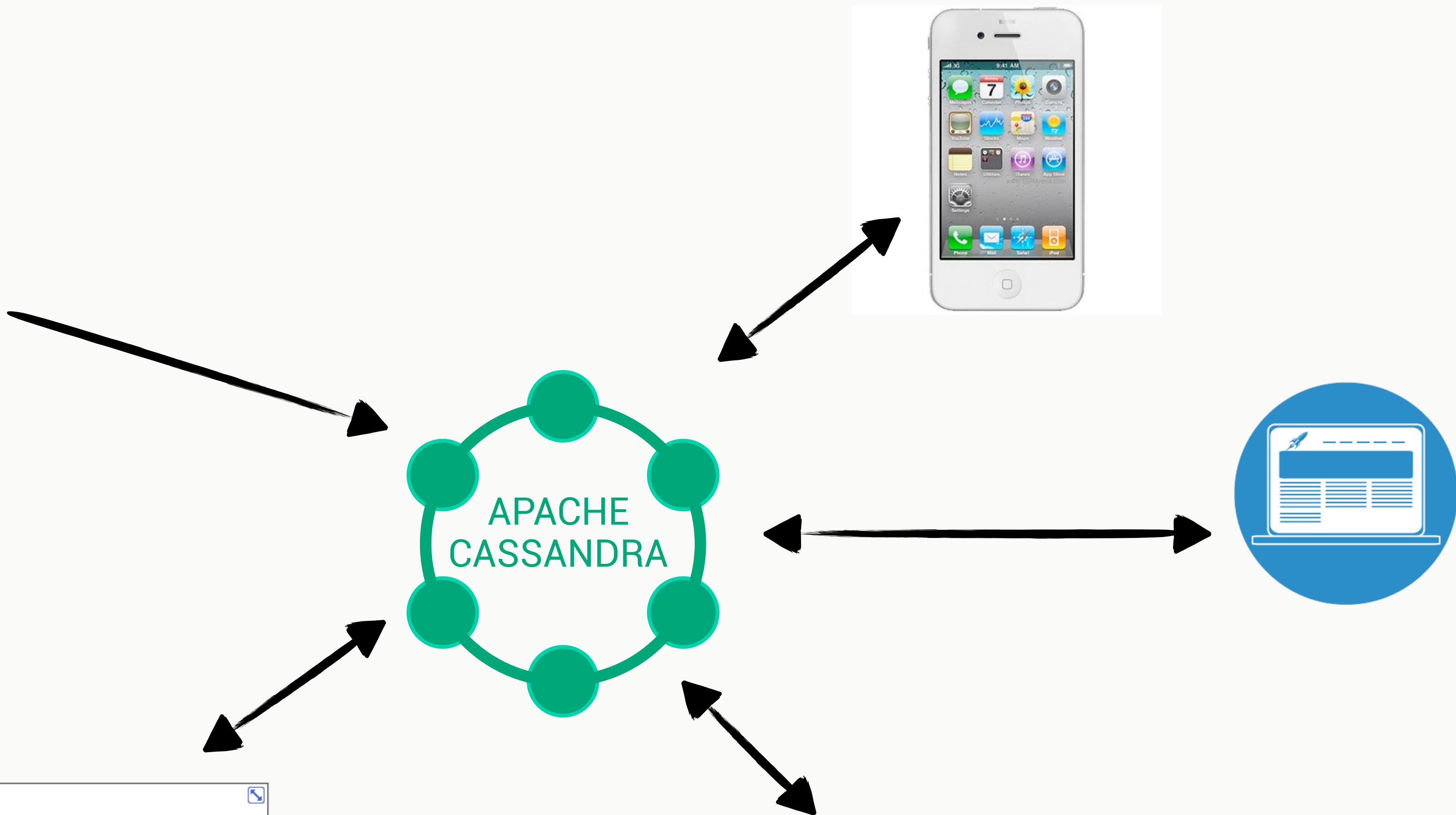
Apache Cassandra

Cassandra is...

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

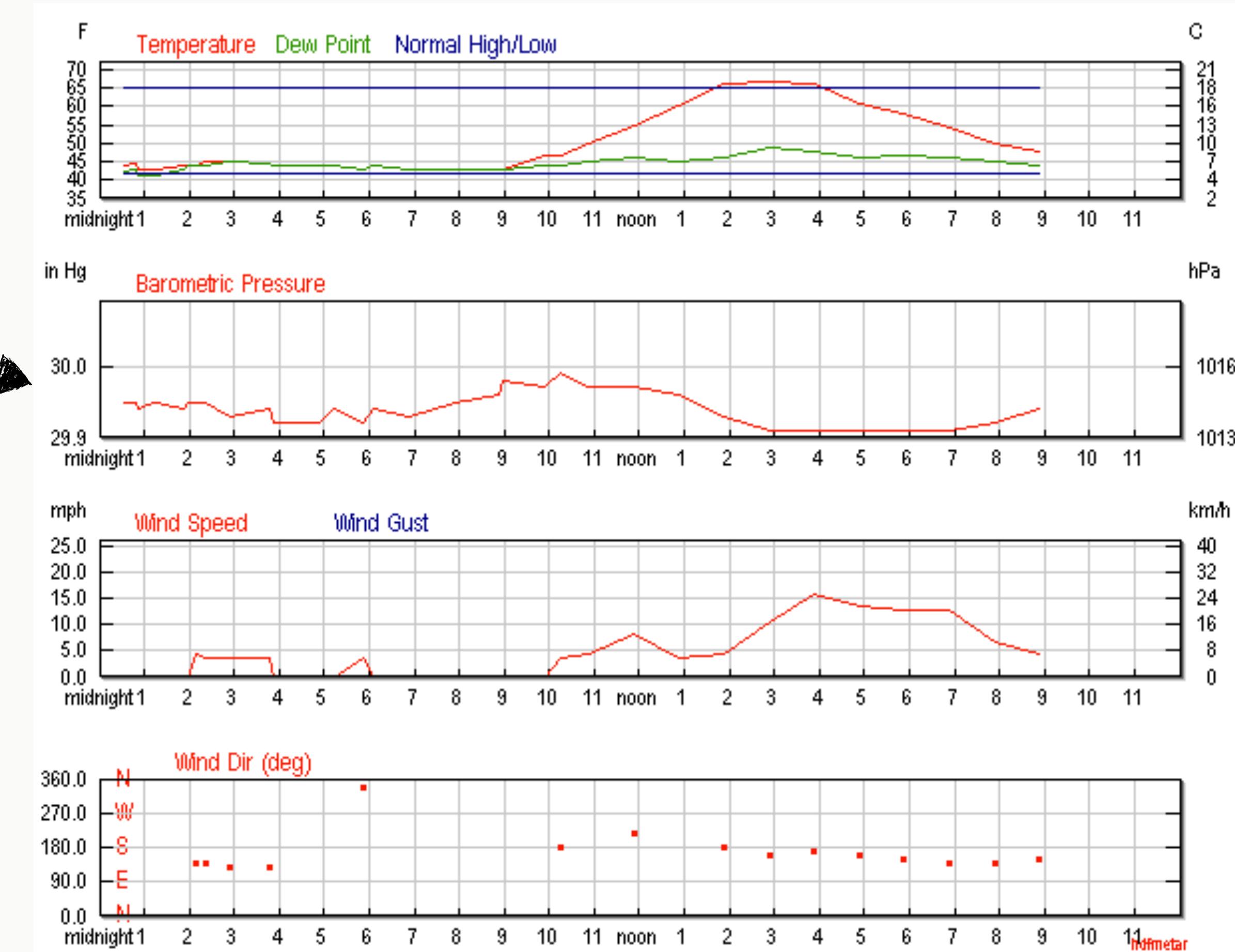
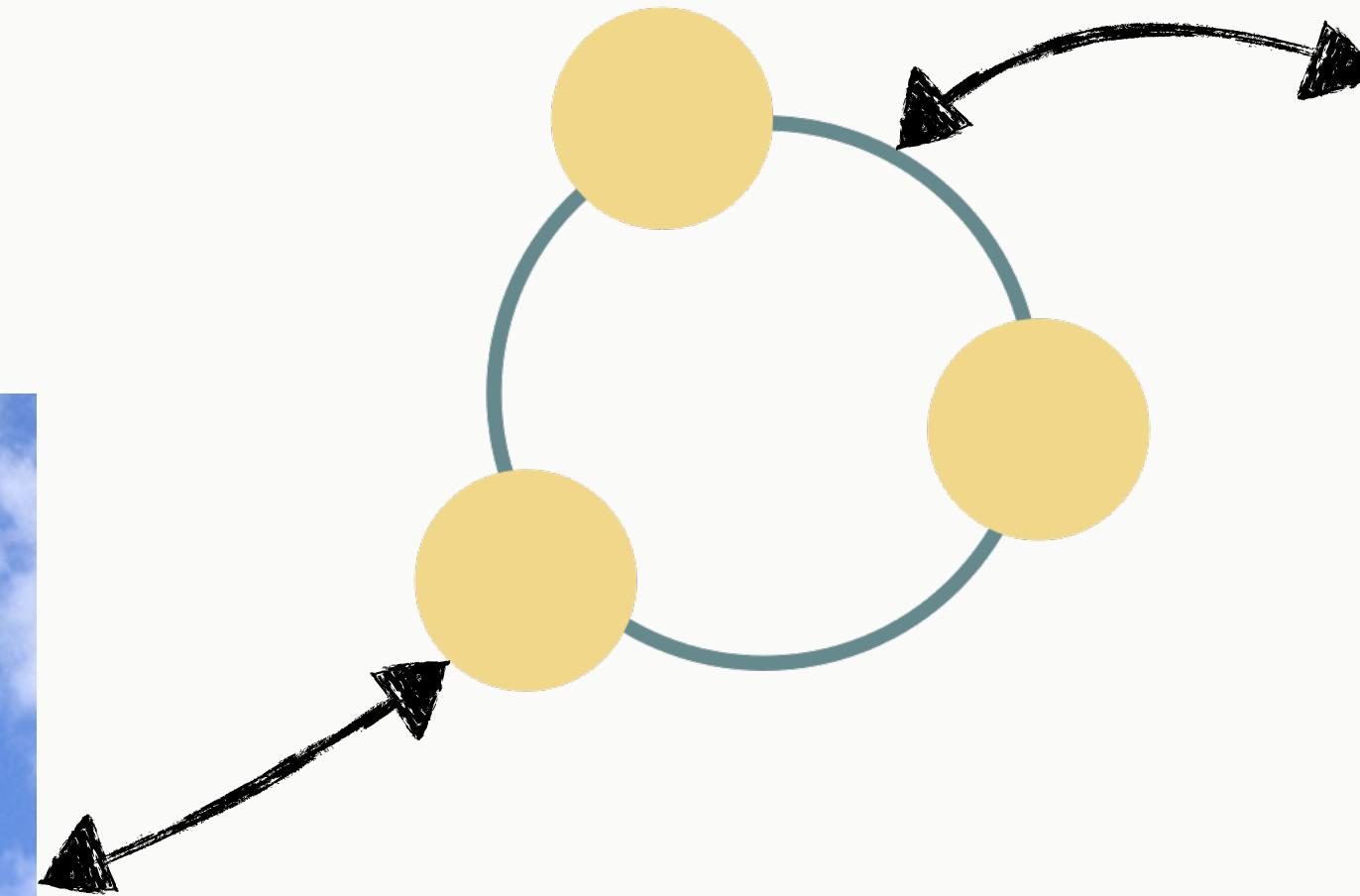


Cassandra for Applications



Example: Weather Station

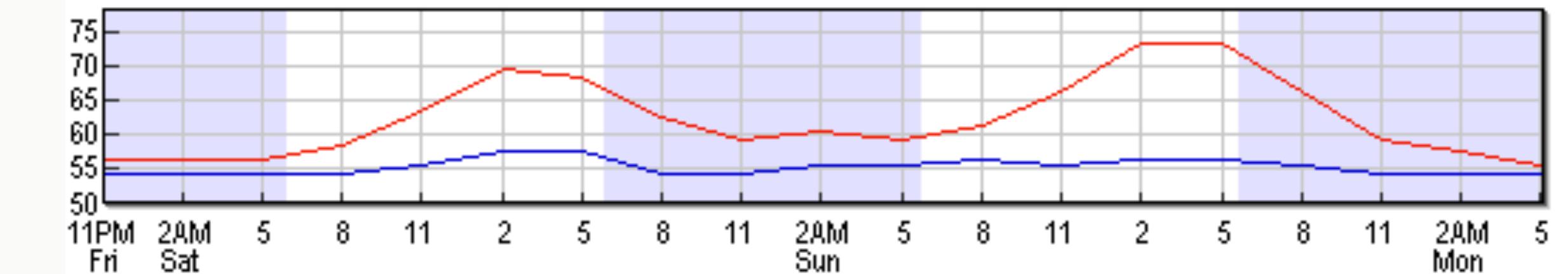
- Weather station collects data
- Cassandra stores in sequence
- Application reads in sequence



Queries supported

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

Get temperature stats given

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

Windsor California

July 1, 2014

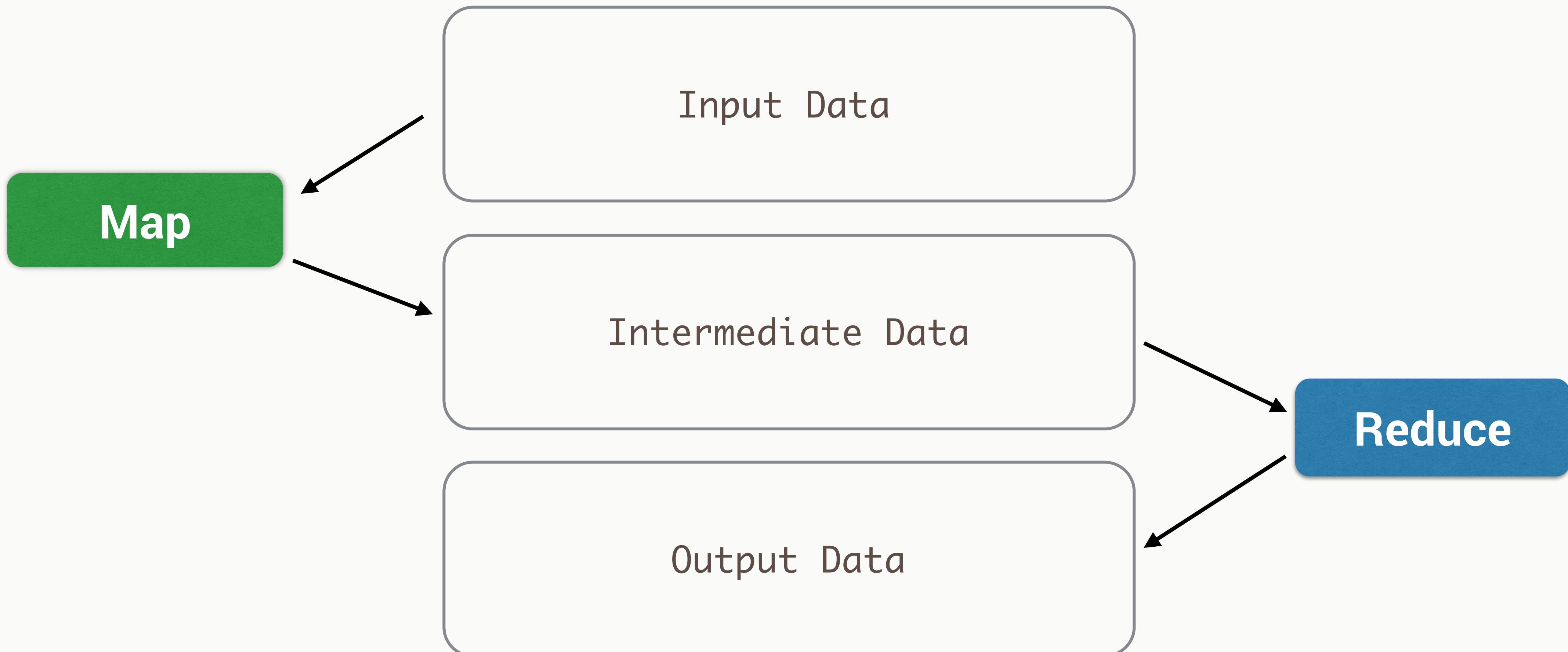
High: 73.4

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);
```

Apache Spark

Map Reduce



Apache Spark

- 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

Spark
Streaming
real-time

Spark SQL
structured

MLlib
machine learning

GraphX
graph

Spark Core

Blog

Company Blog

[All Posts](#)[Partners](#)[Events](#)[Press Releases](#)

Developer Blog

[All Posts](#)[Spark](#)[Spark Streaming](#)[Spark SQL](#)[MLlib](#)[Spark Summit](#)

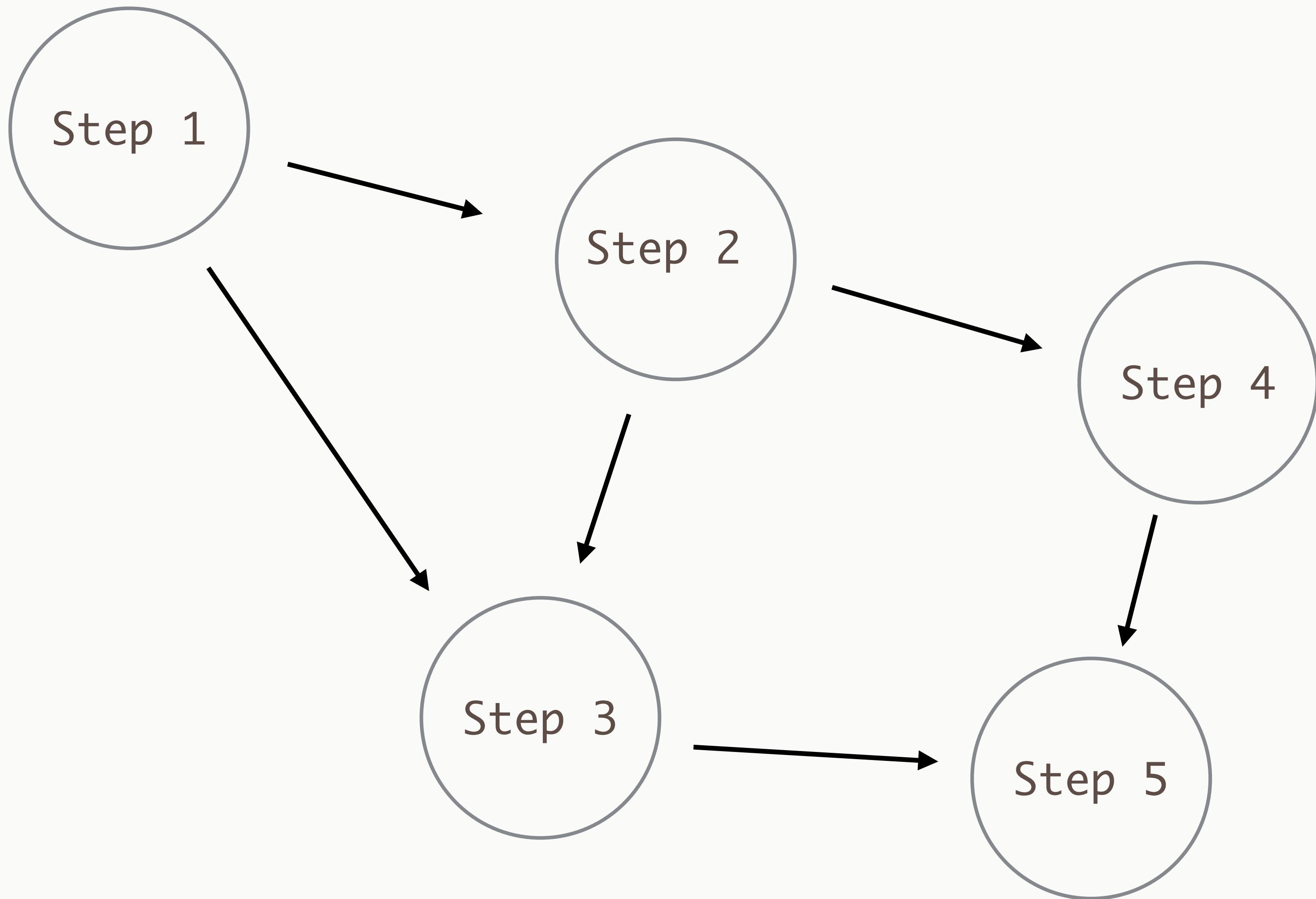
Spark the fastest open source engine for sorting a petabyte

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

Resilient Distributed Dataset (RDD)

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



RDD Operations

- 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

```
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  }
```

Transformation

Action

Cassandra and Spark

Great combo



Great combo

Spark Streaming

Near Real-time

SparkSQL

Structured Data

MLLib

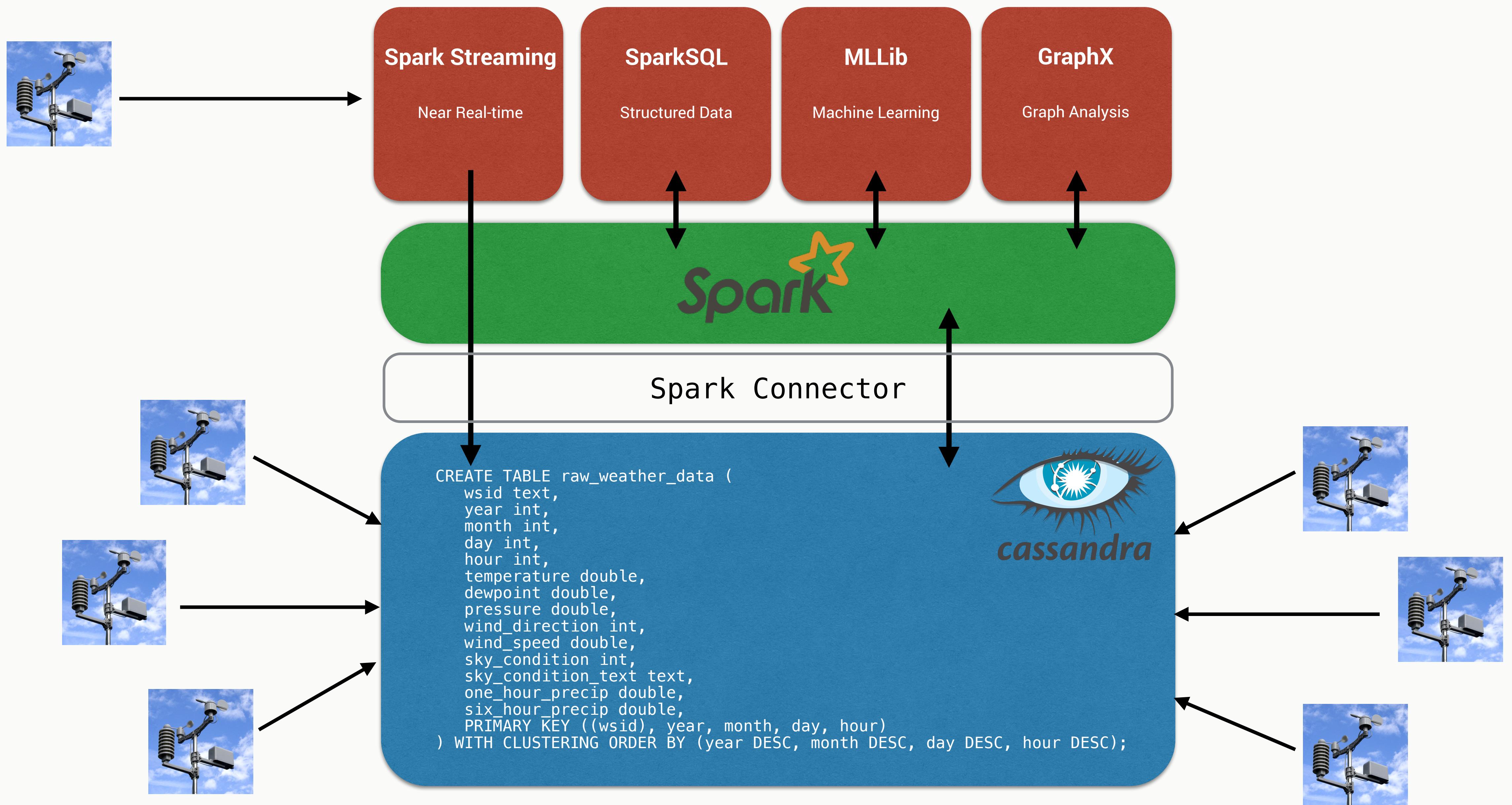
Machine Learning

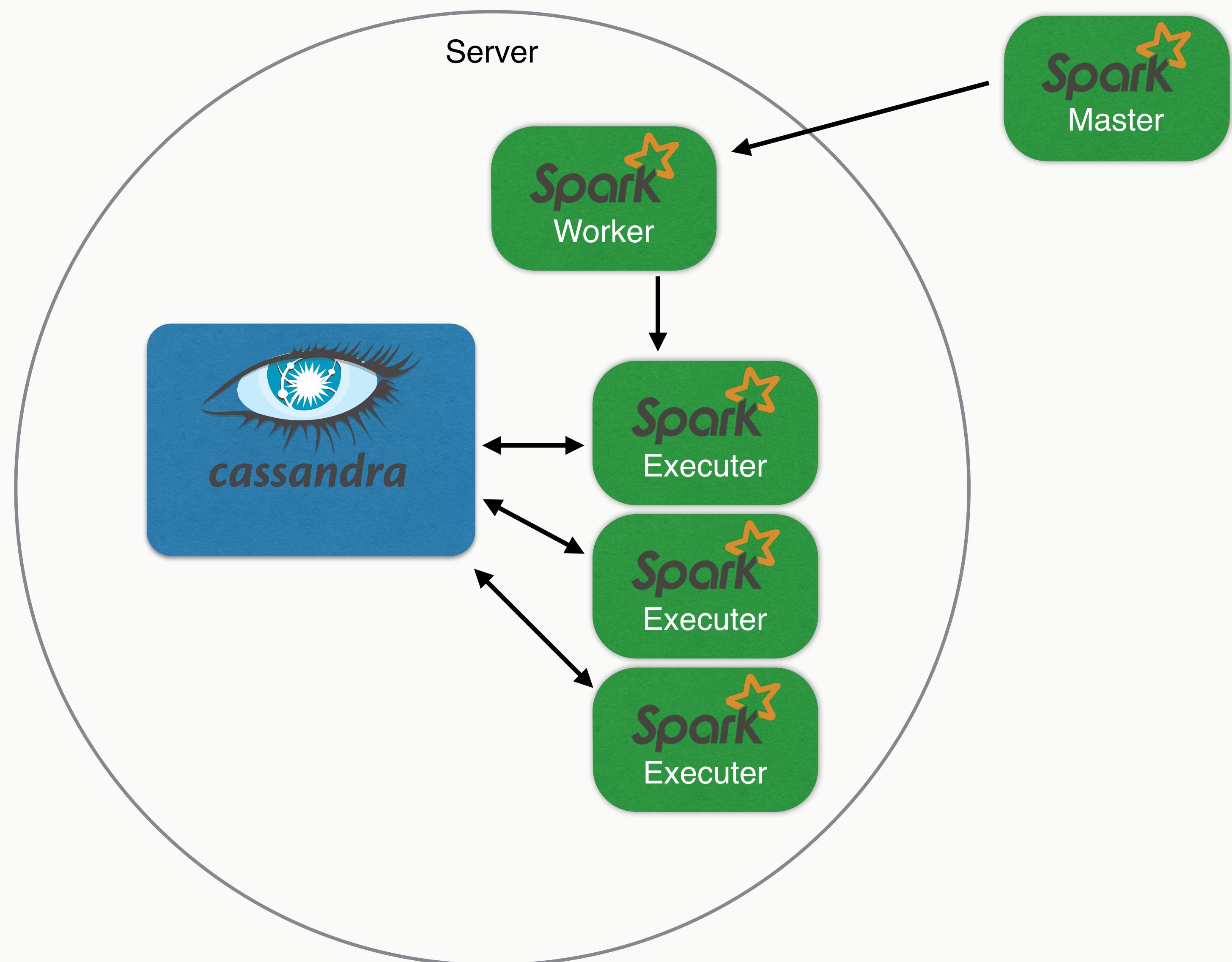
GraphX

Graph Analysis



Great combo

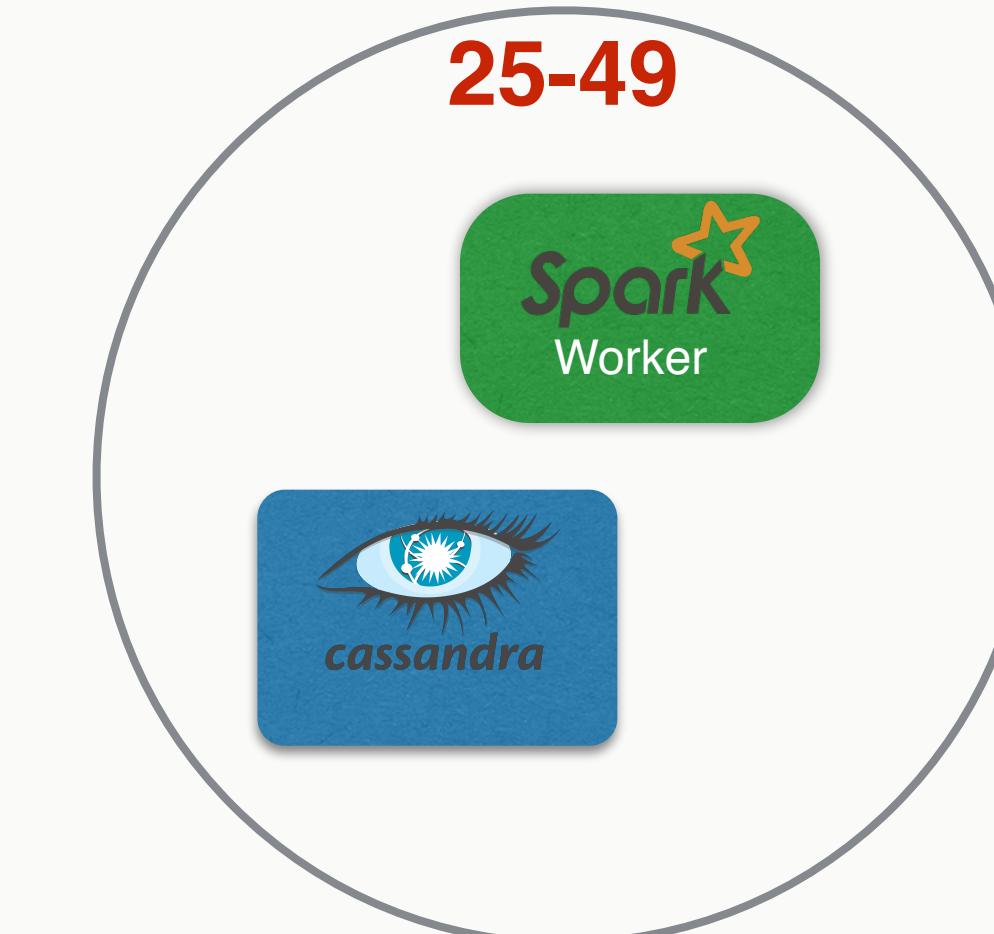
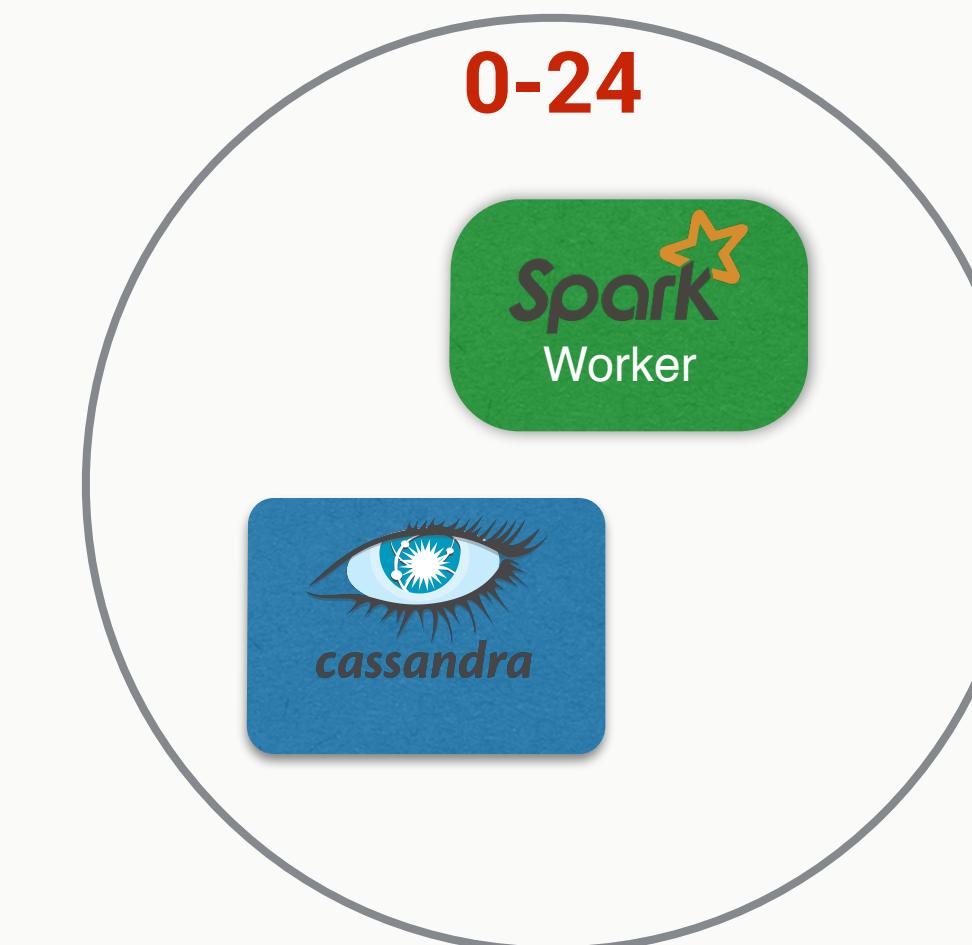
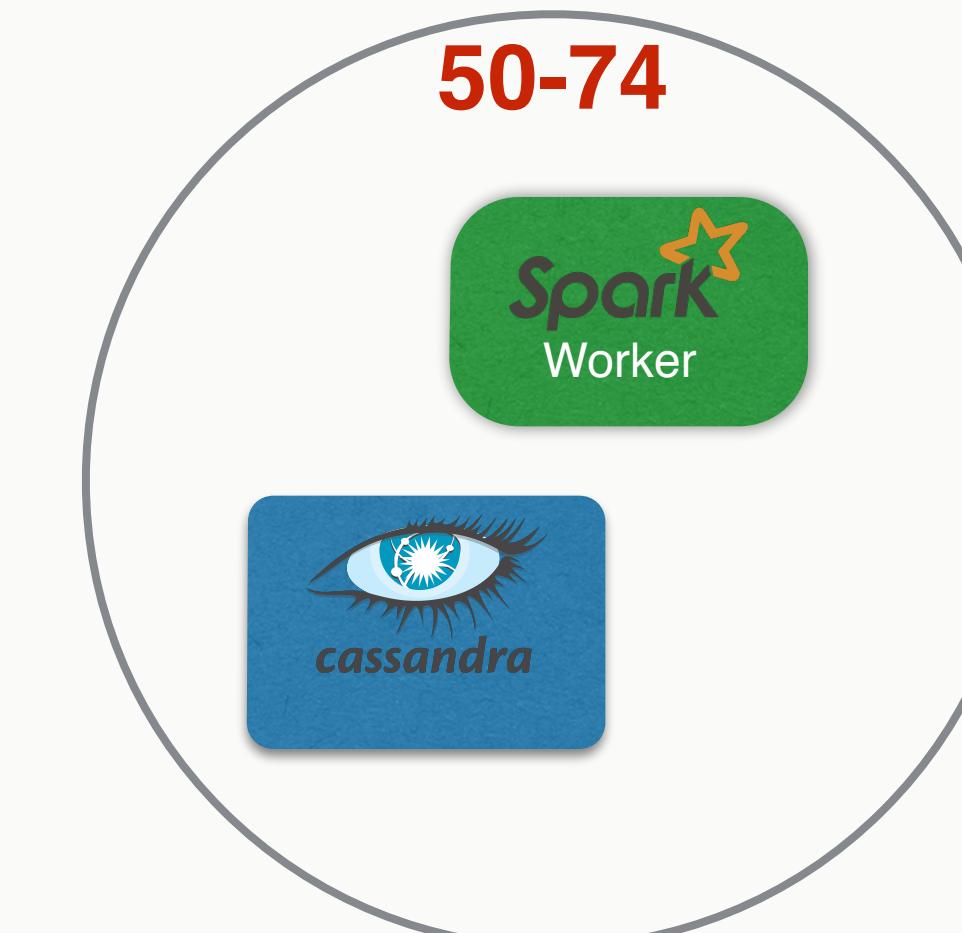
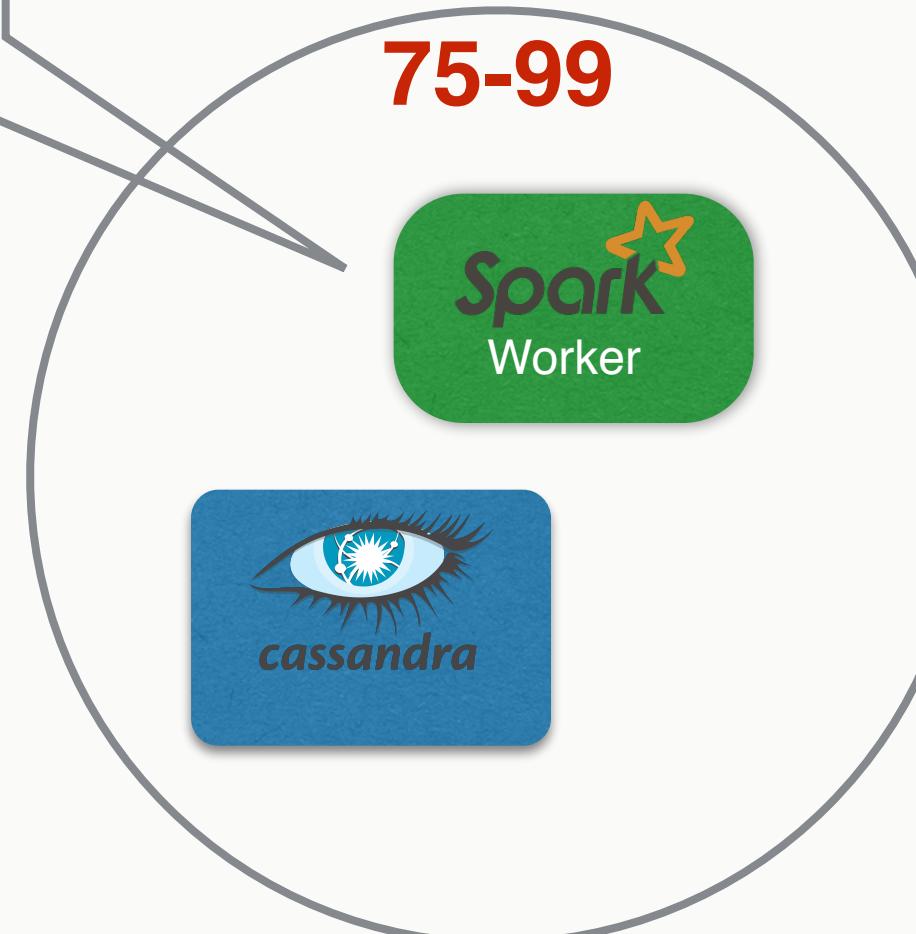




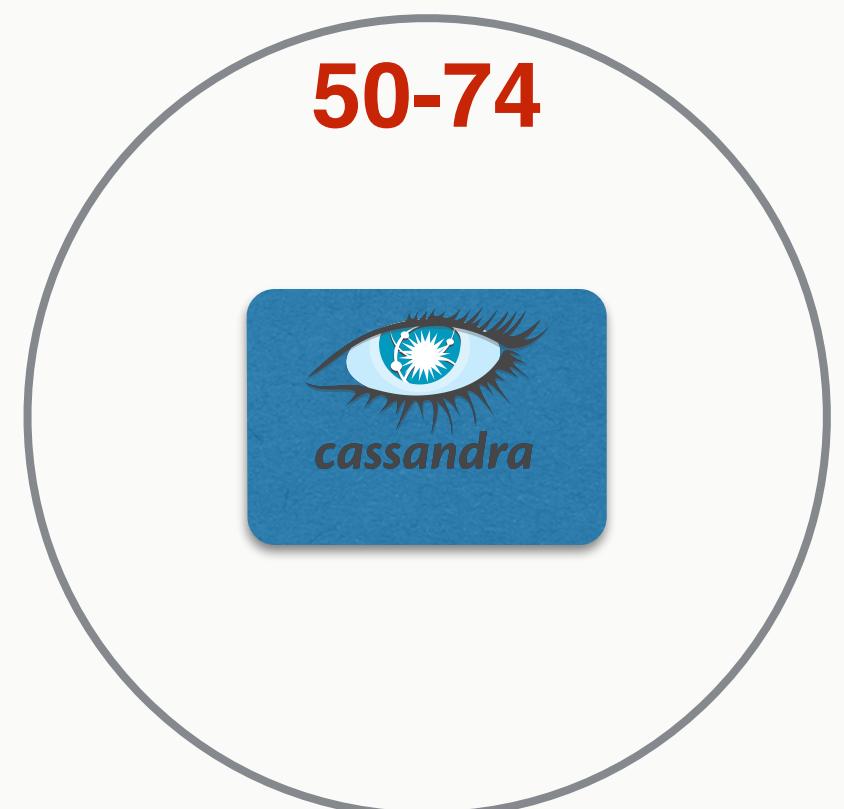
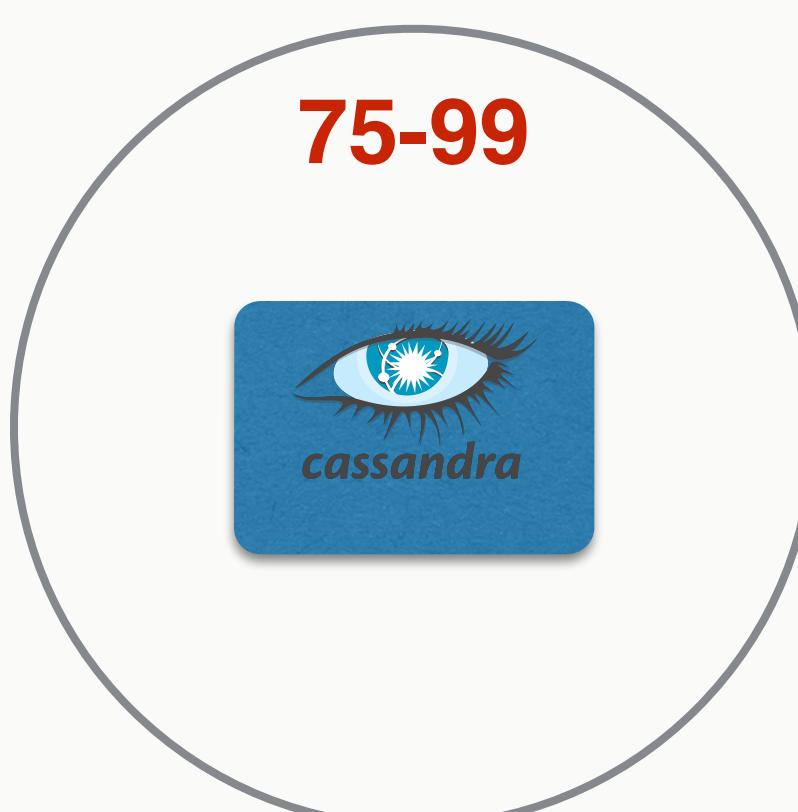
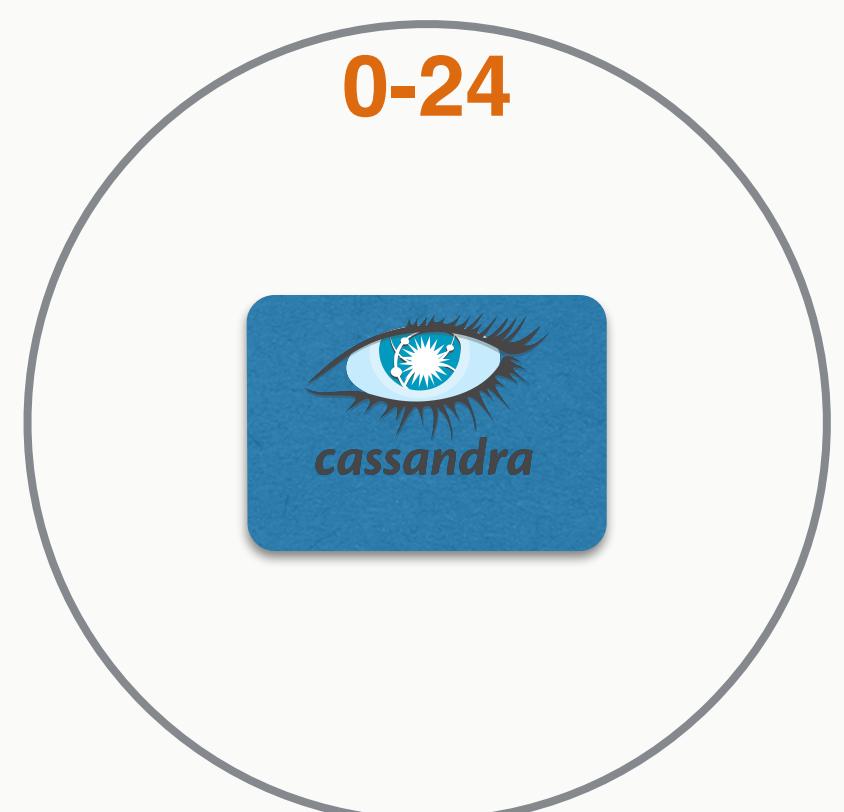
Token Ranges

0-100

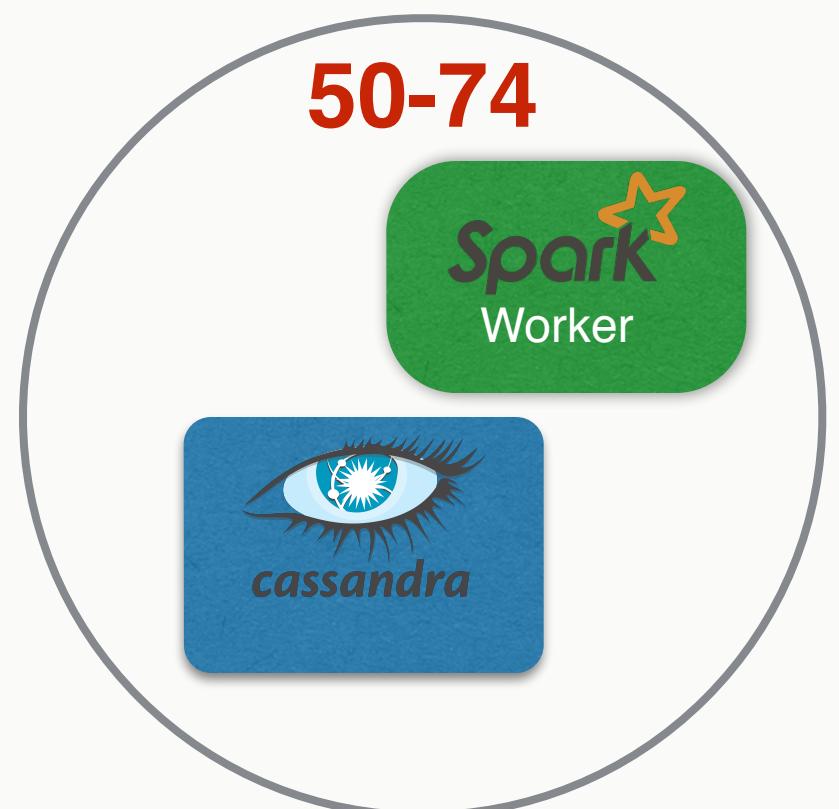
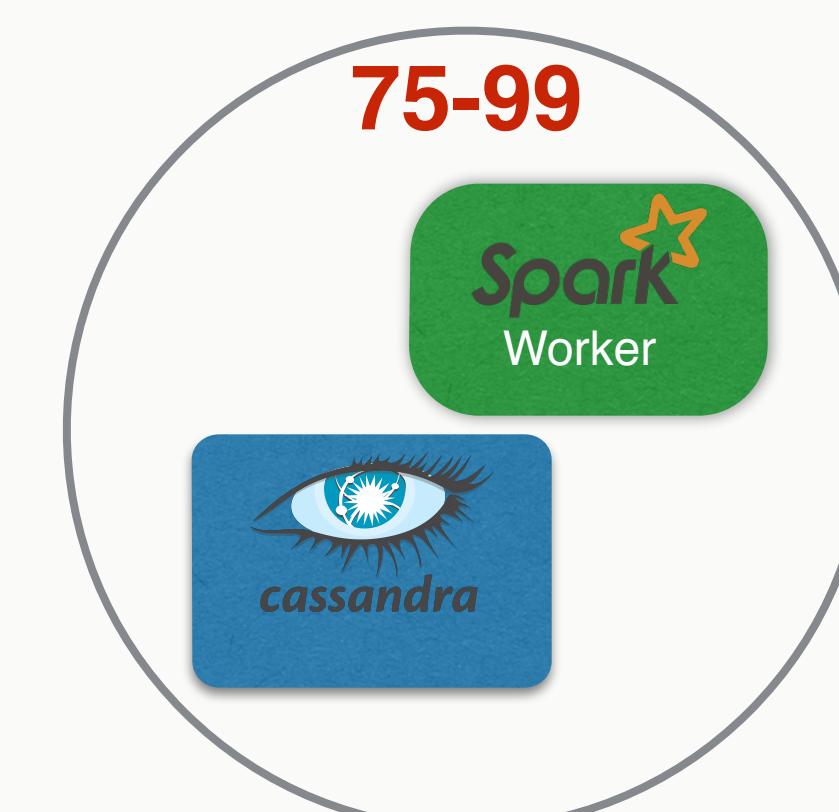
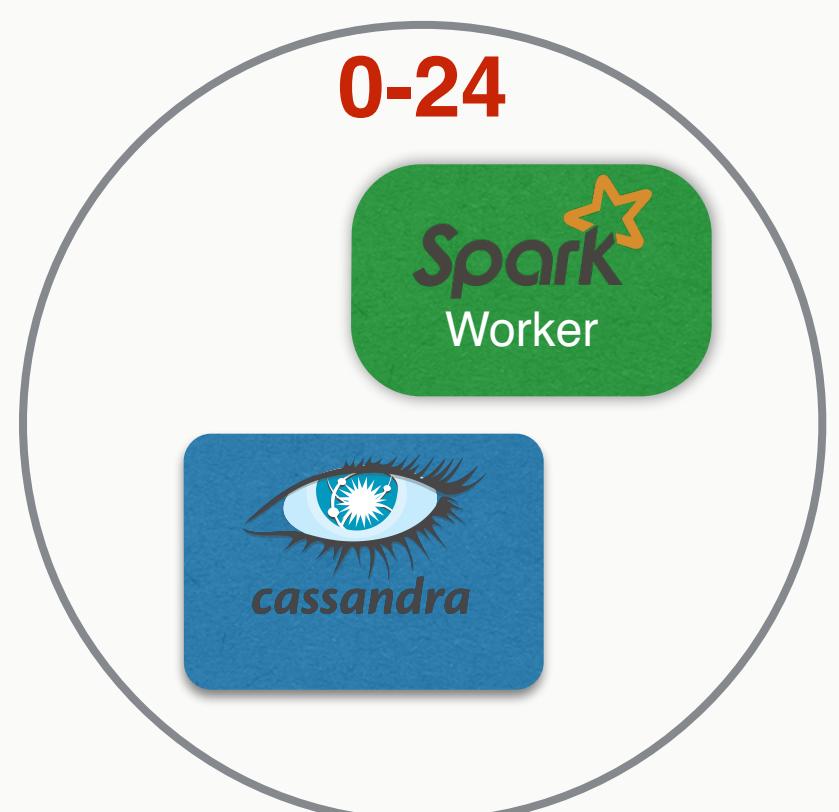
I will only analyze 25% of the data.

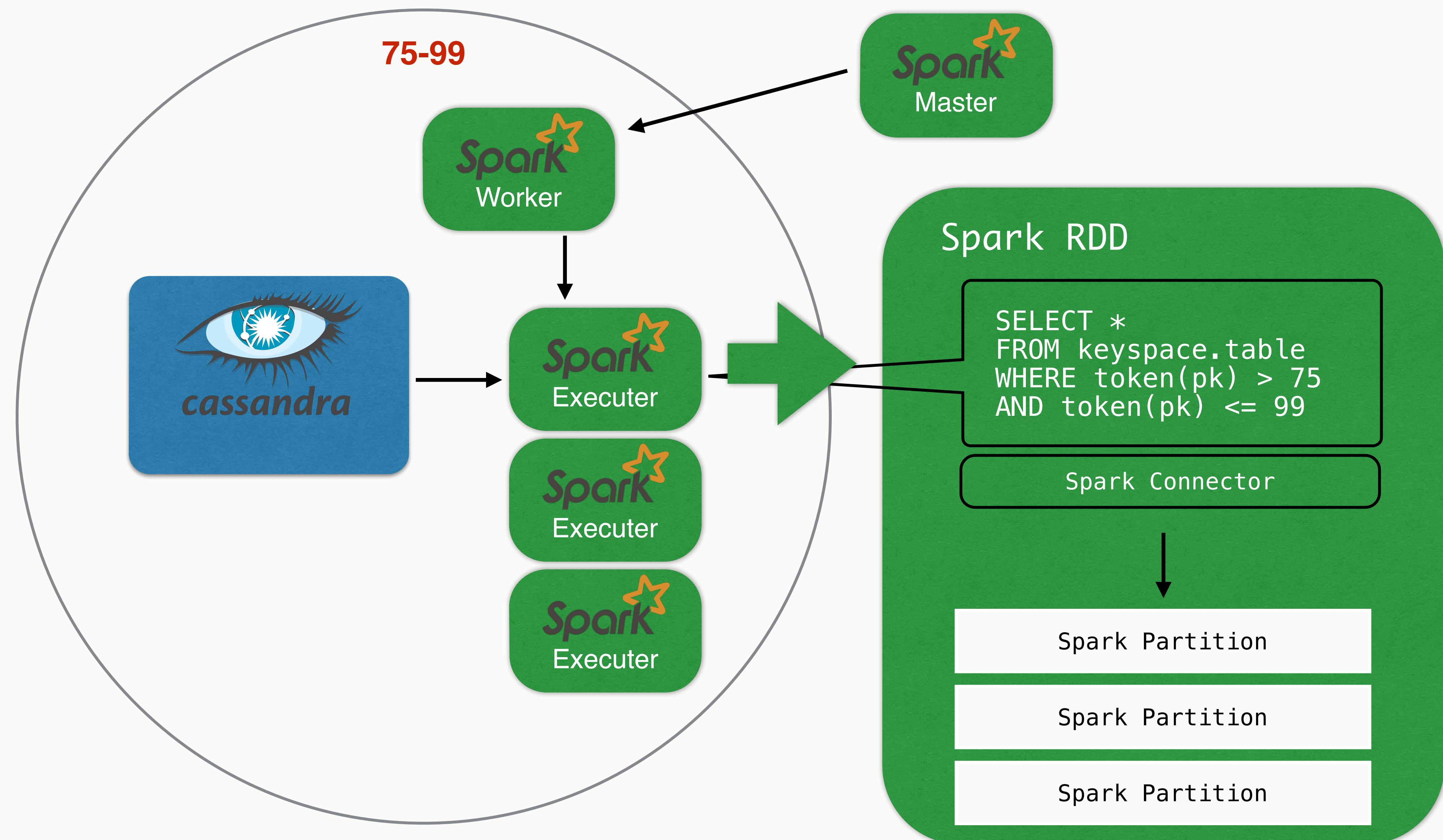


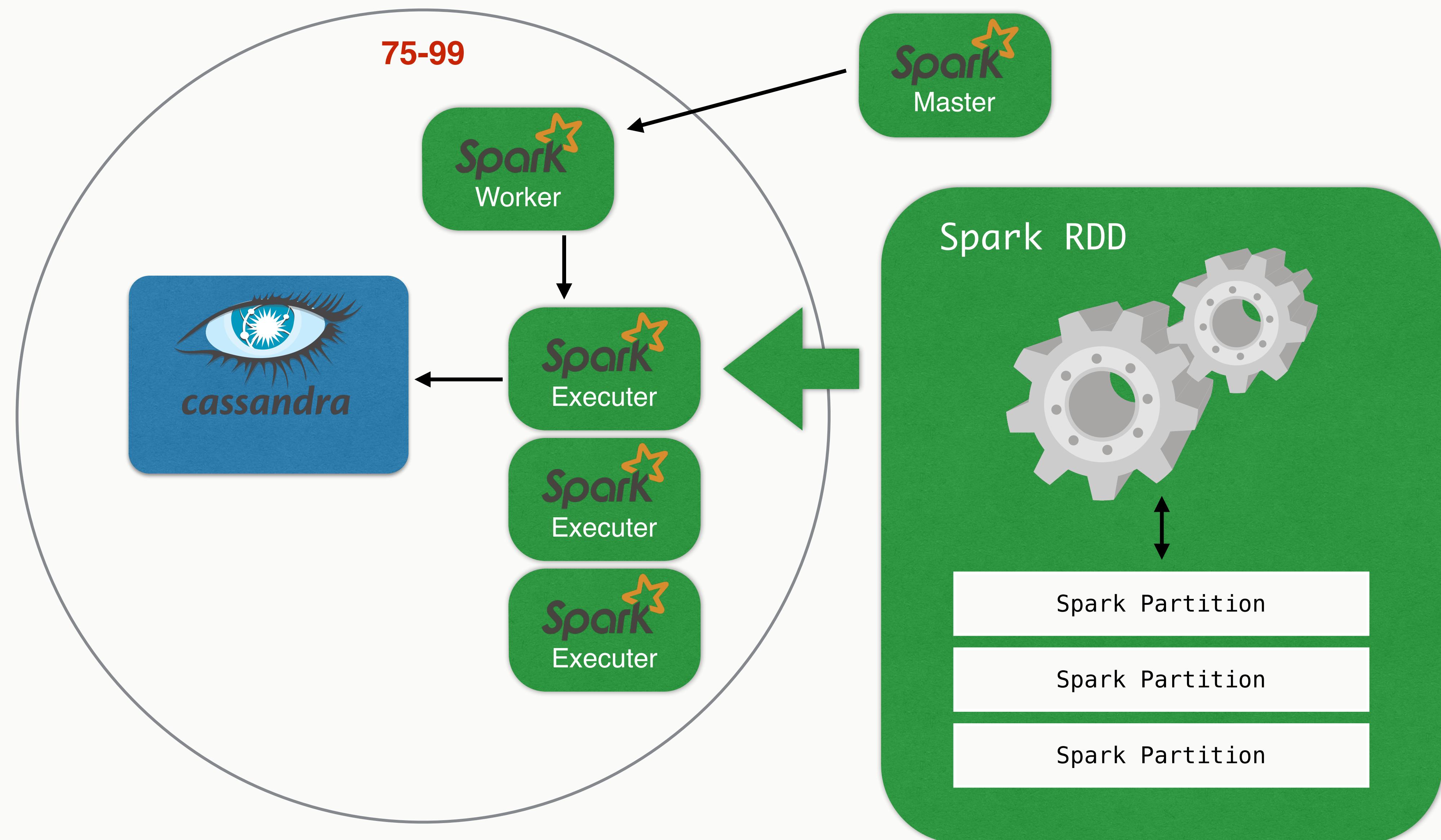
Transactional



Analytics







Spark Reads on Cassandra

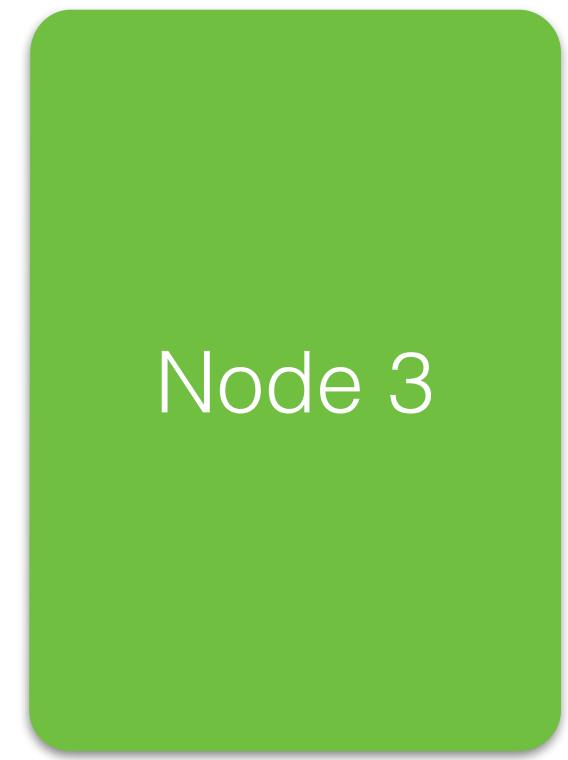
Awesome animation by DataStax's own Russell Spitzer

Spark RDDs

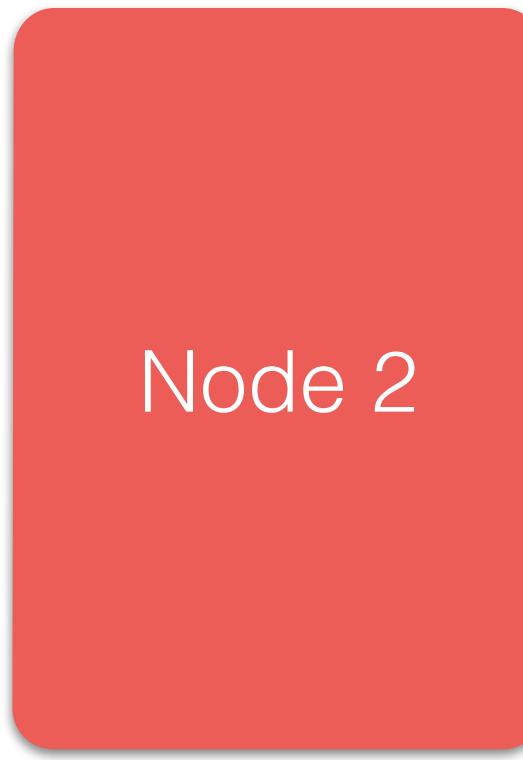
Represent a Large
Amount of Data
Partitioned into Chunks



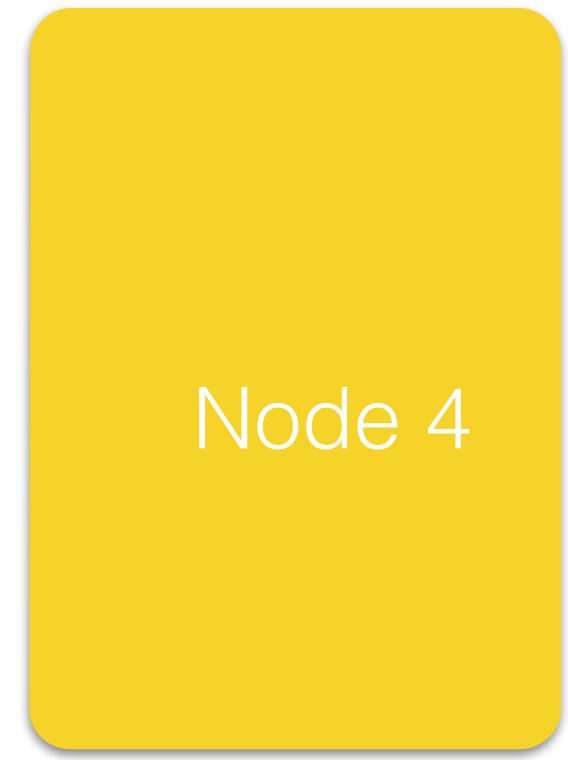
Node 1



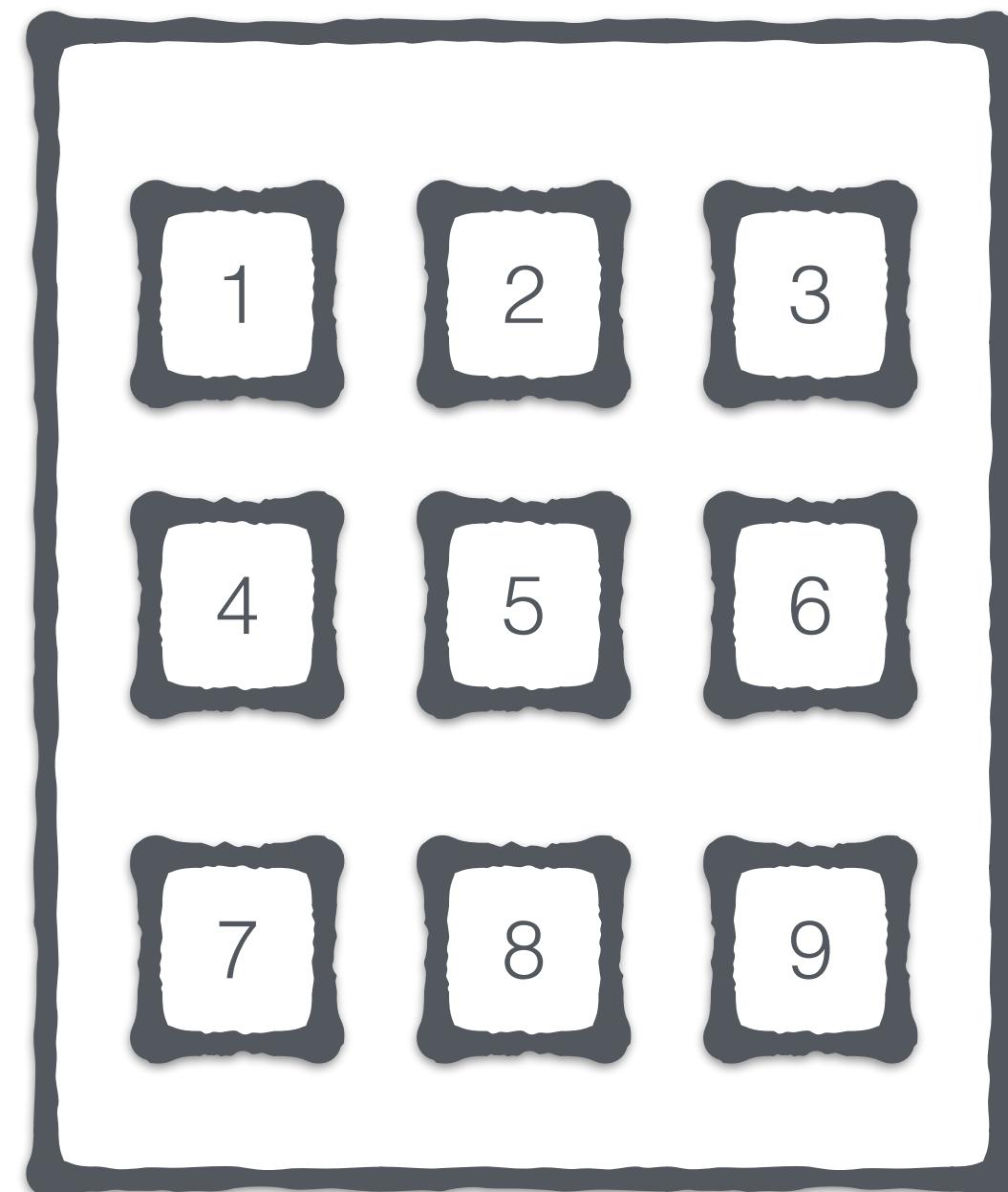
Node 3



Node 2

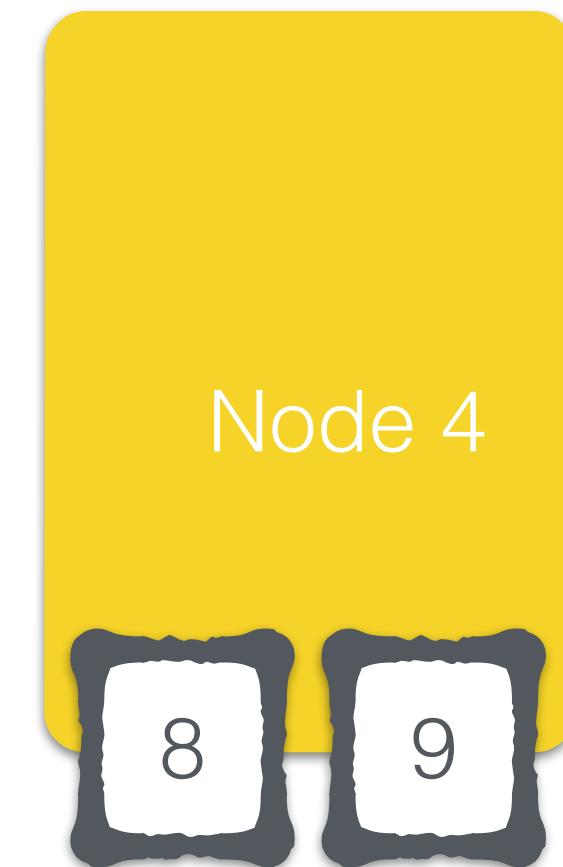
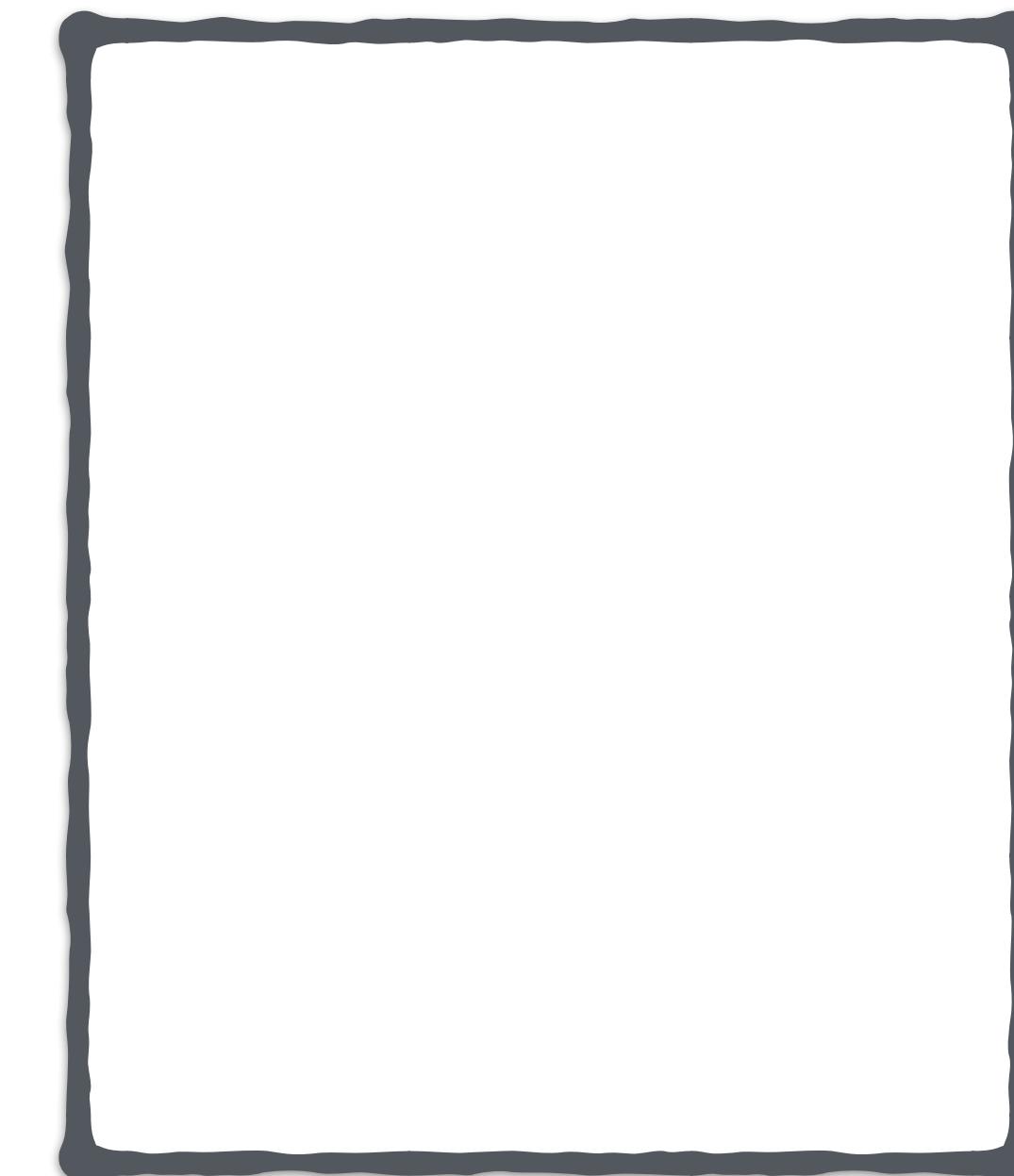
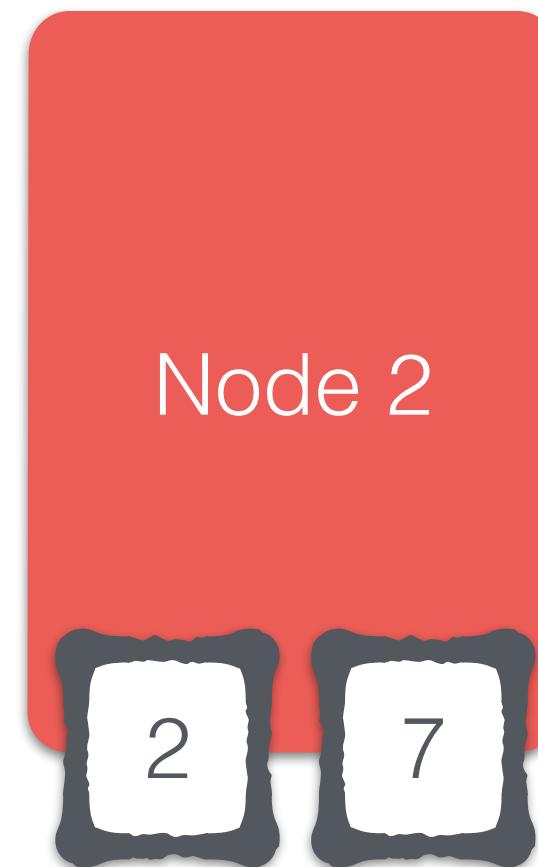
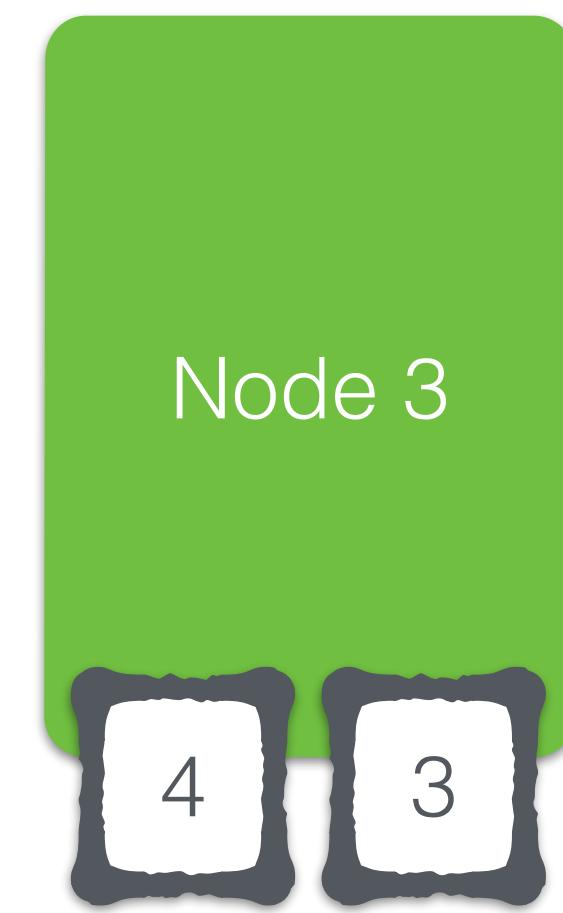


Node 4



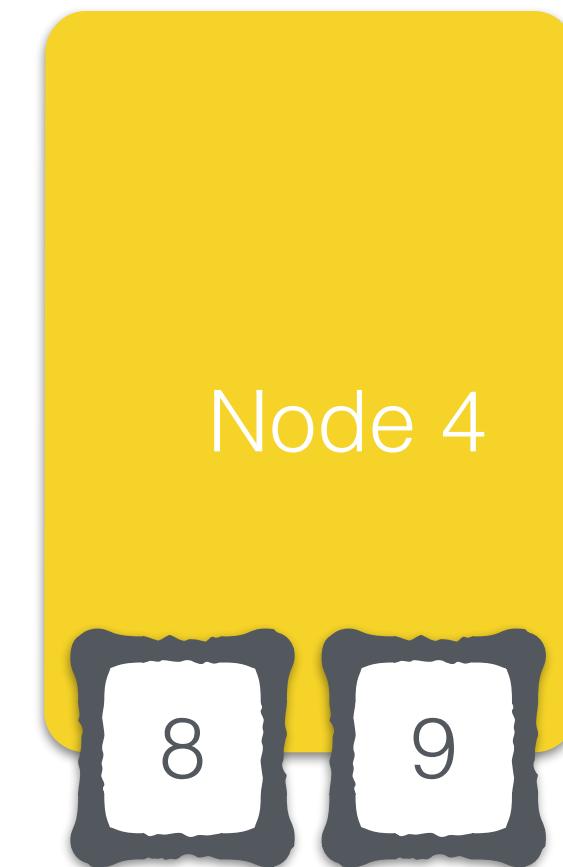
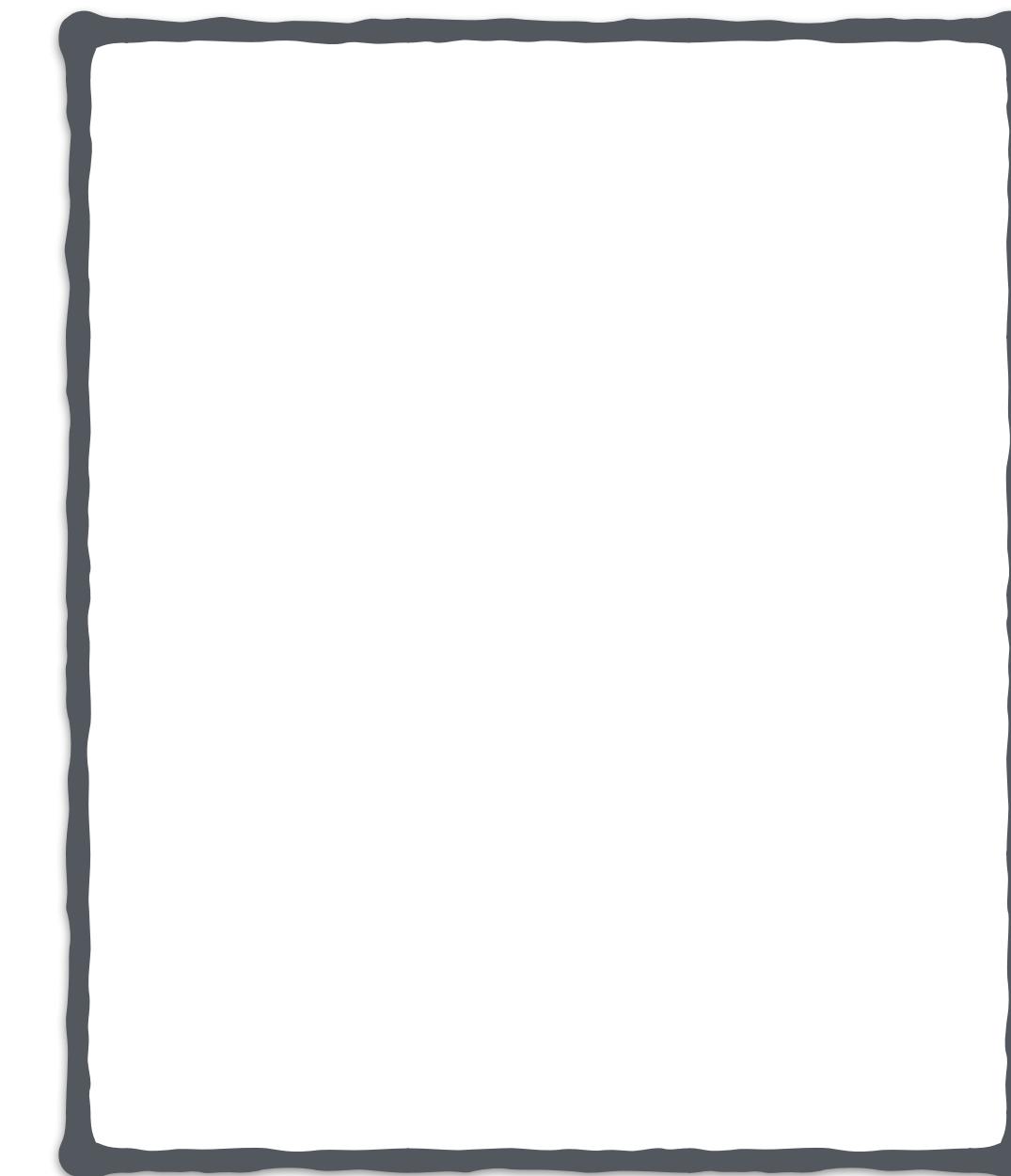
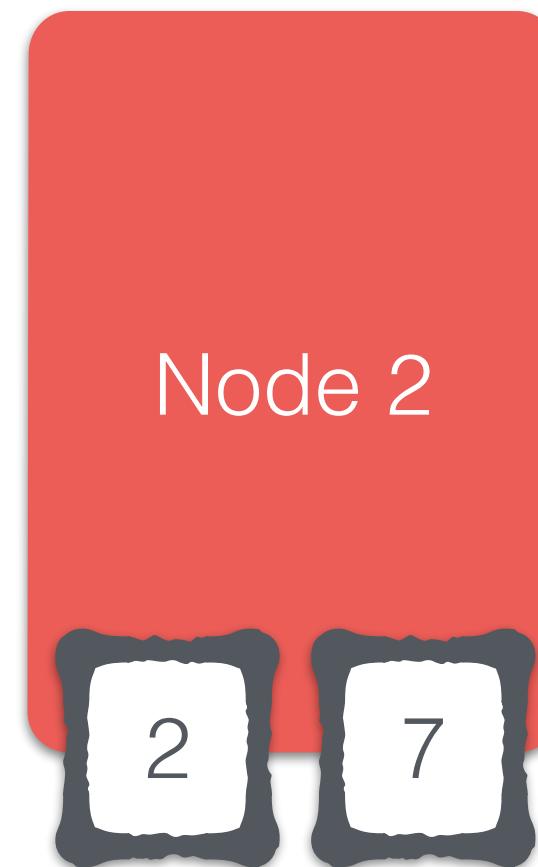
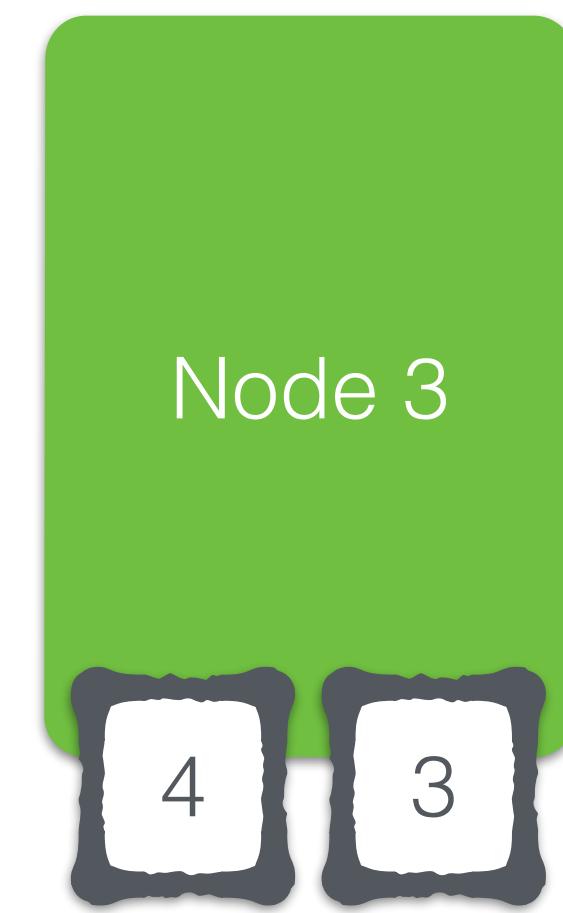
Spark RDDs

Represent a Large
Amount of Data
Partitioned into Chunks



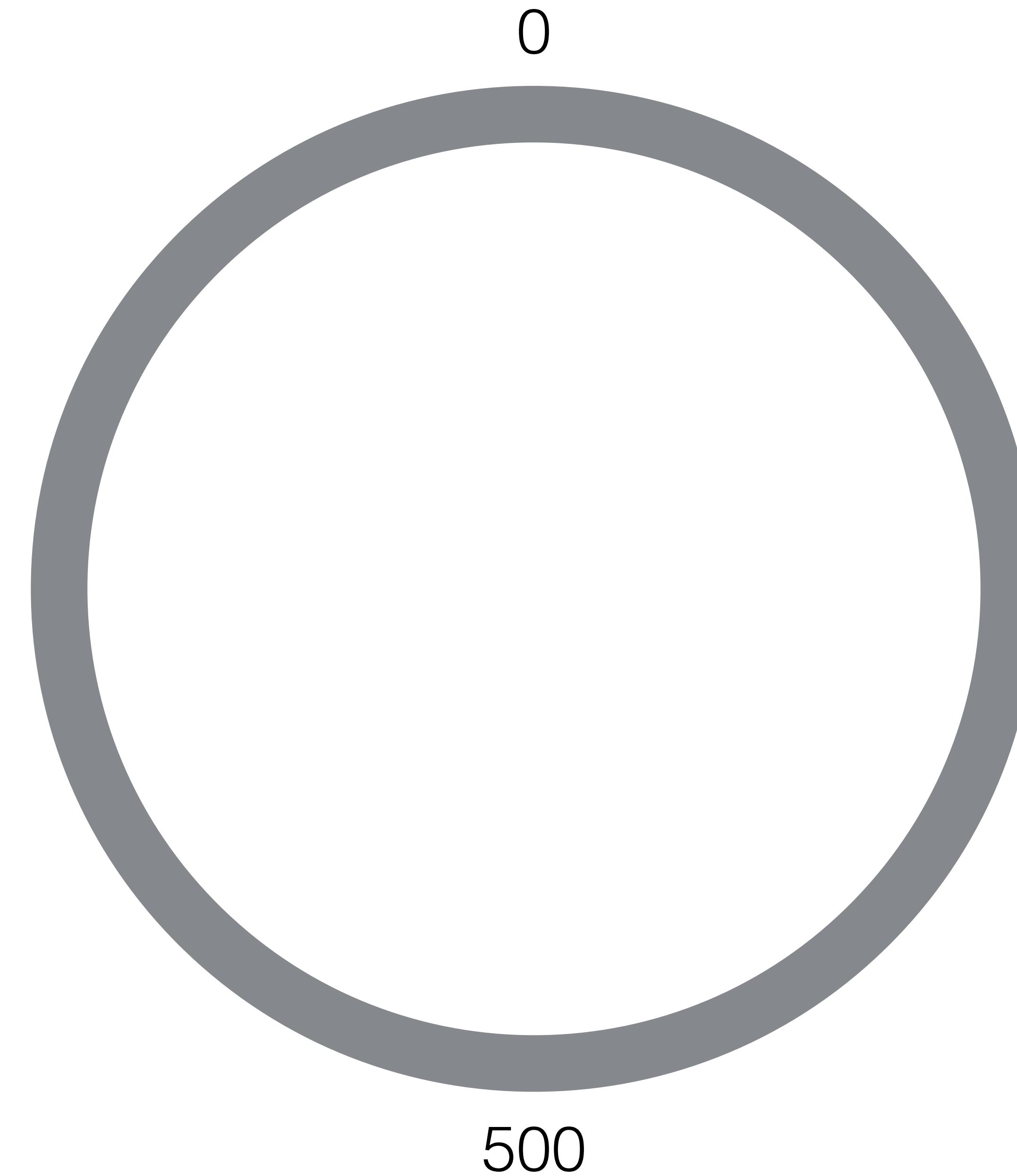
Spark RDDs

Represent a Large
Amount of Data
Partitioned into Chunks



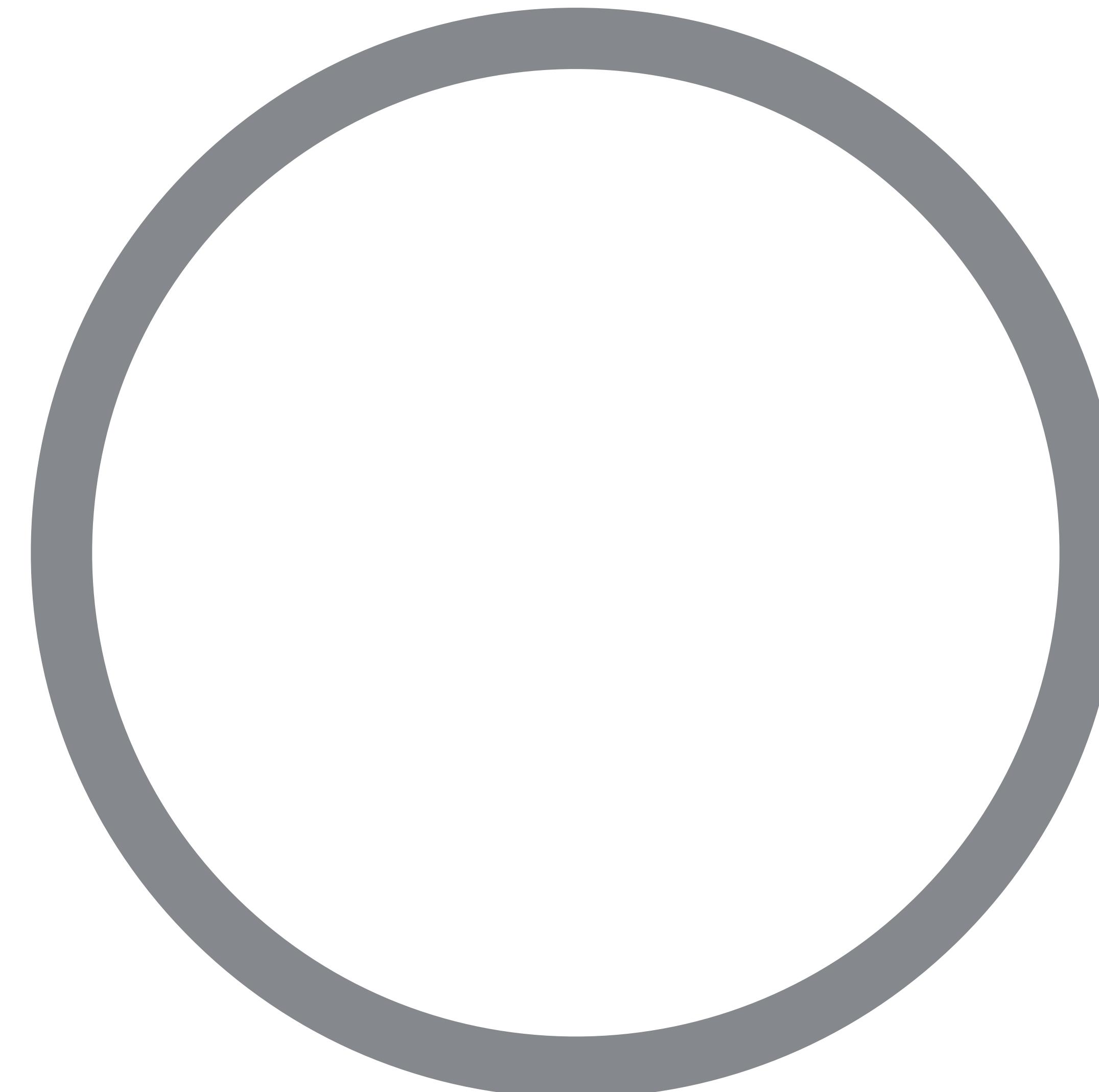
Cassandra Data is Distributed By Token Range

Cassandra Data is Distributed By Token Range



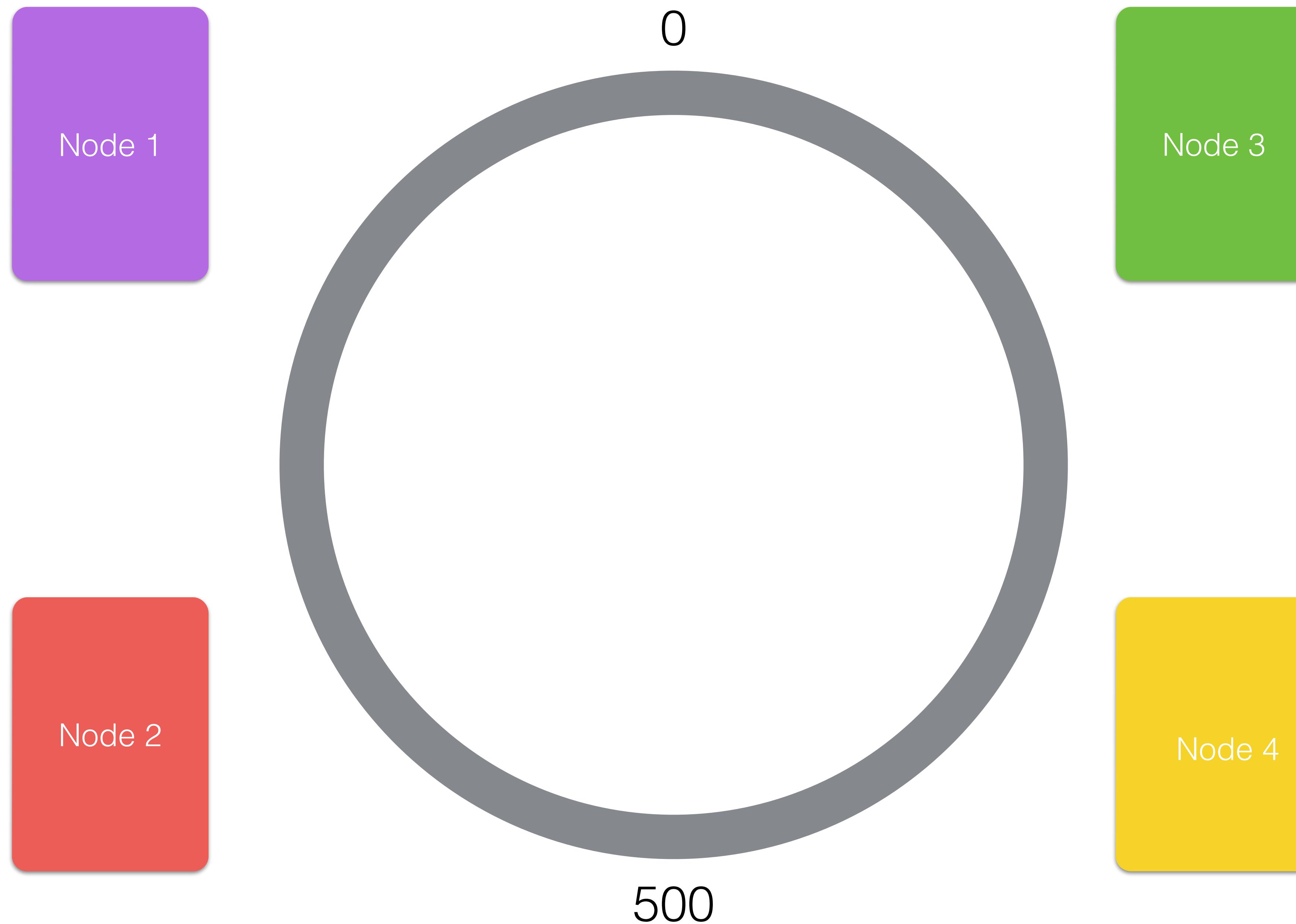
Cassandra Data is Distributed By Token Range

999 0

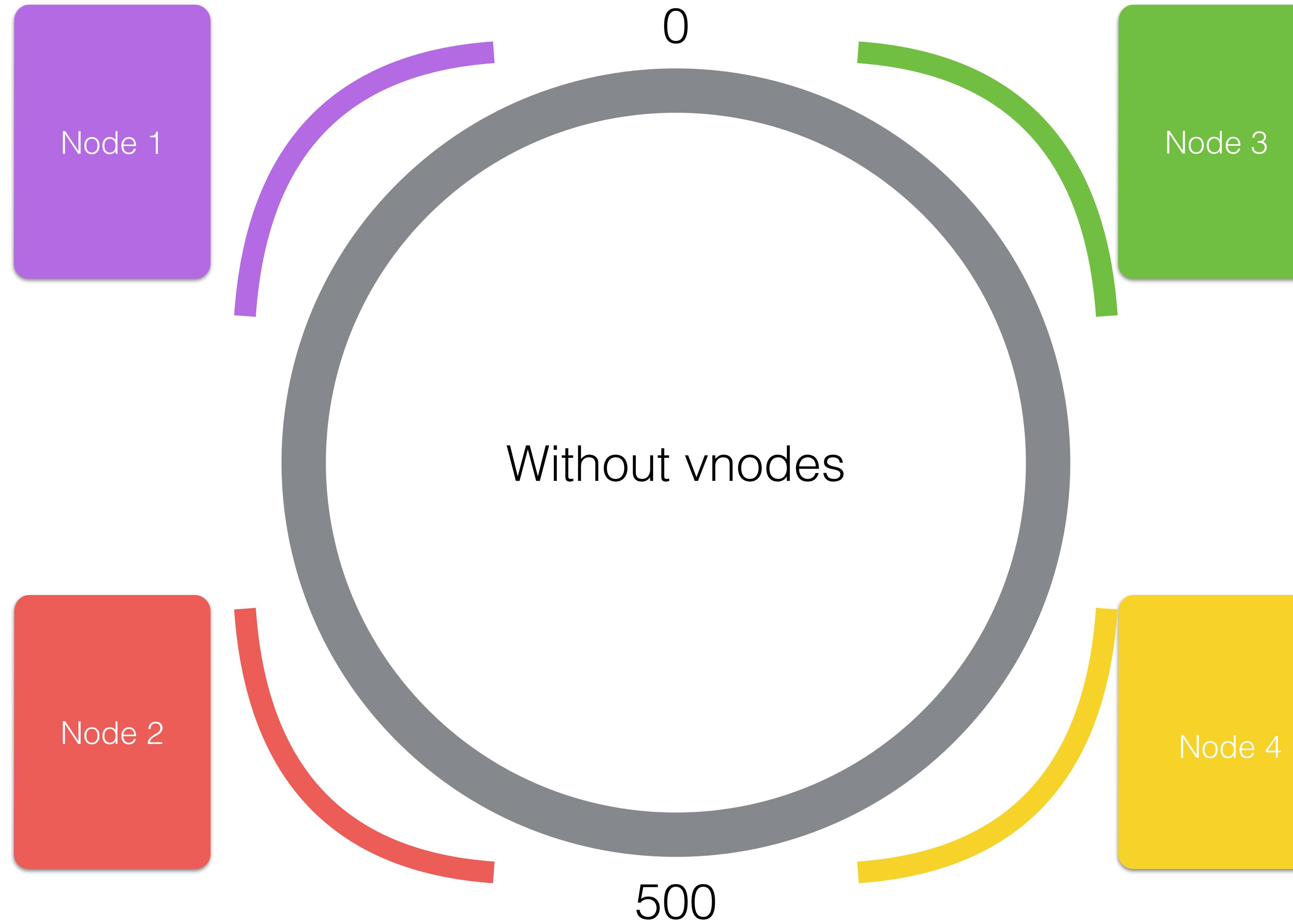


500

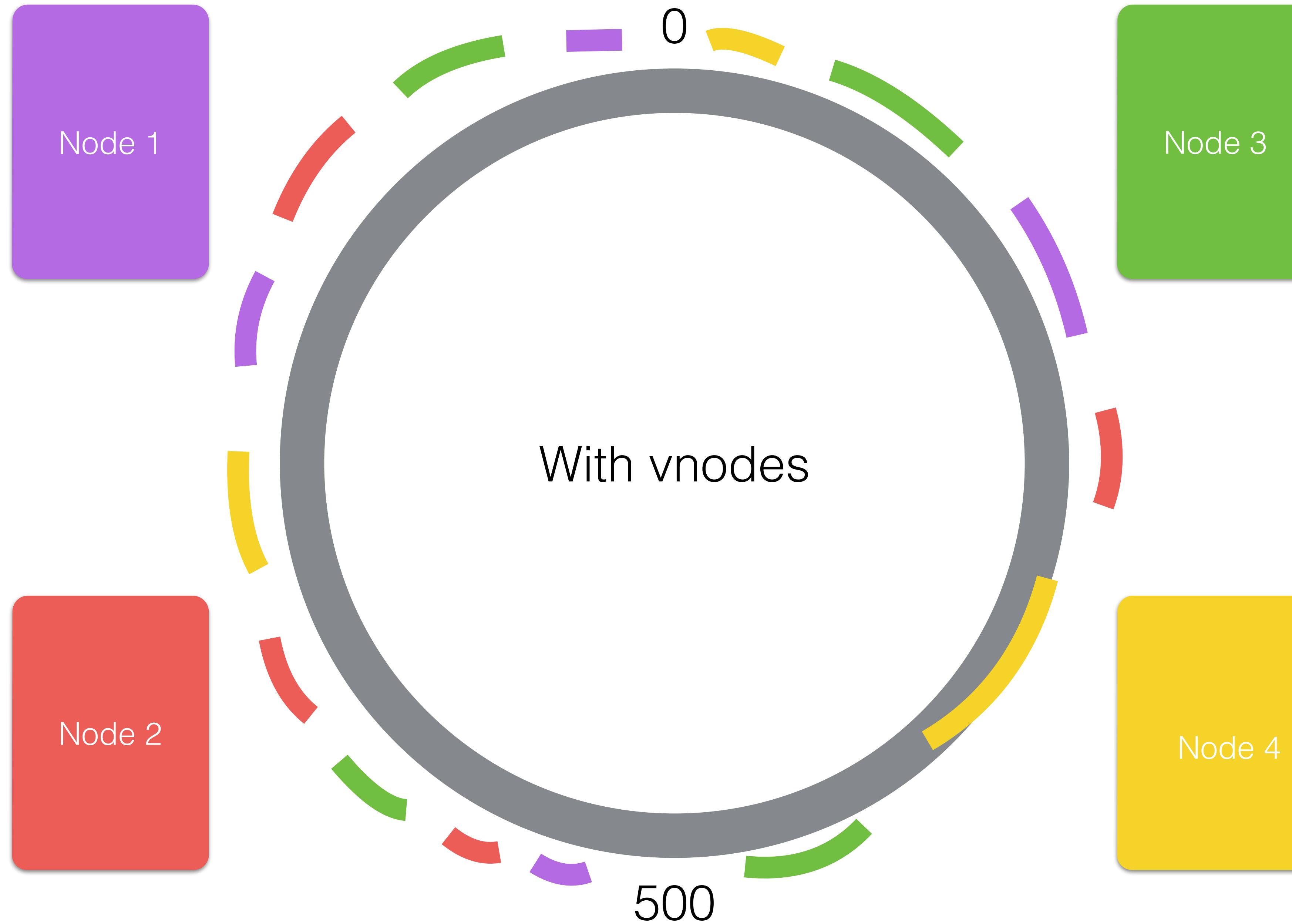
Cassandra Data is Distributed By Token Range



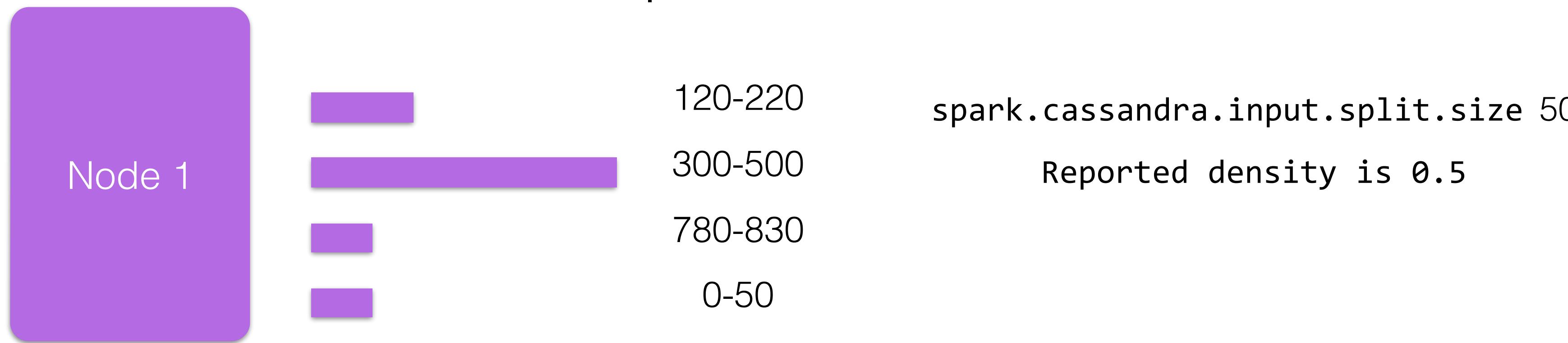
Cassandra Data is Distributed By Token Range



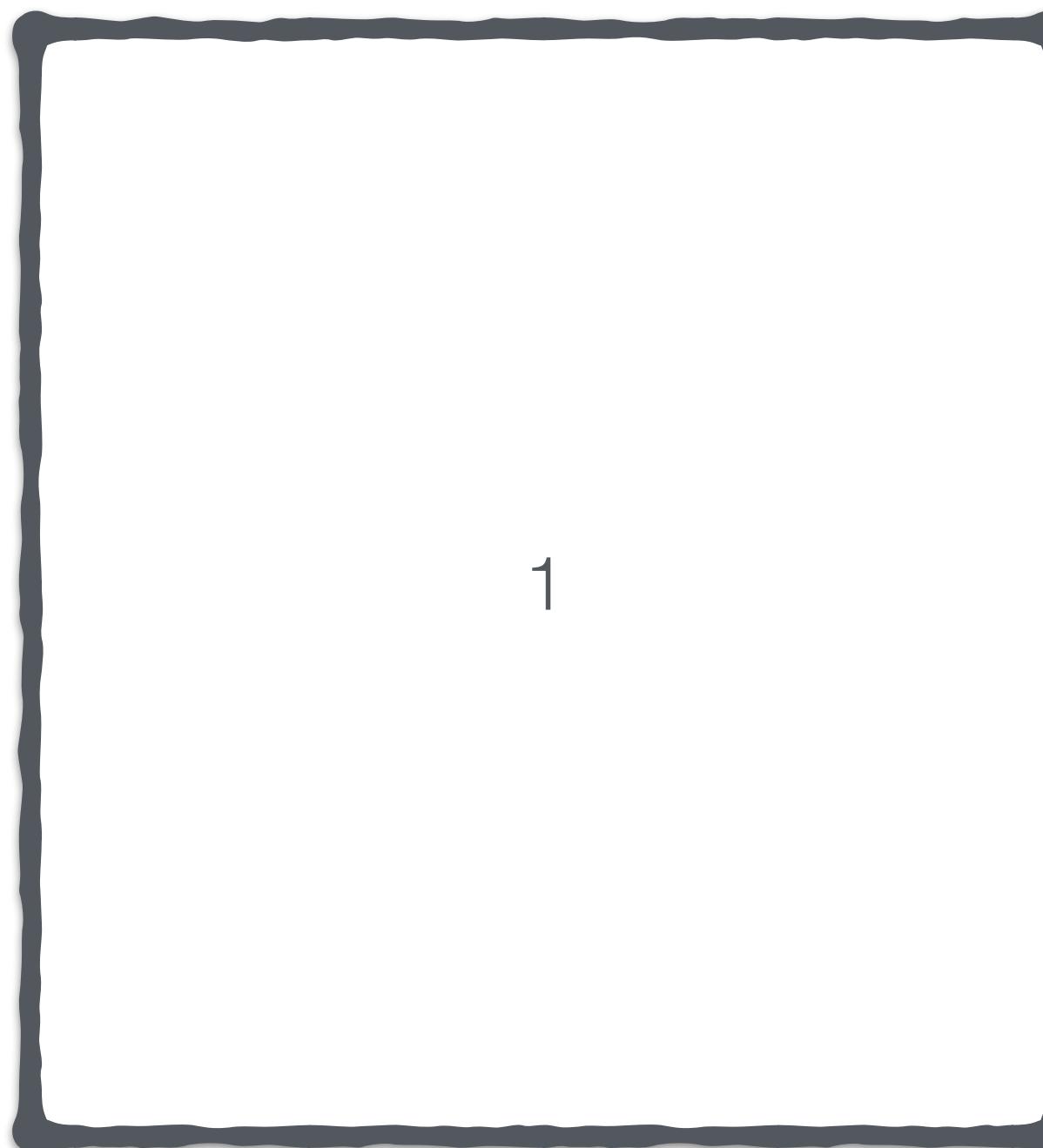
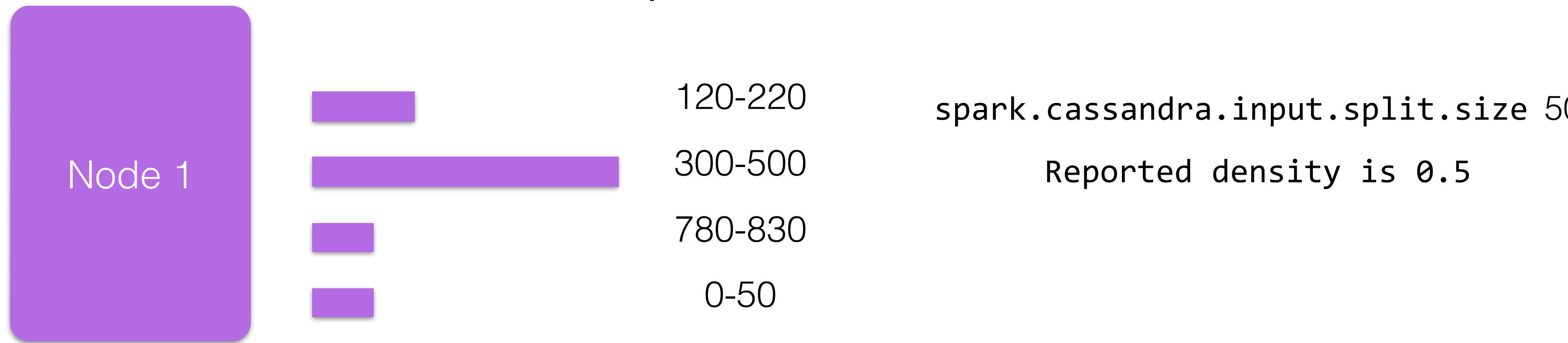
Cassandra Data is Distributed By Token Range



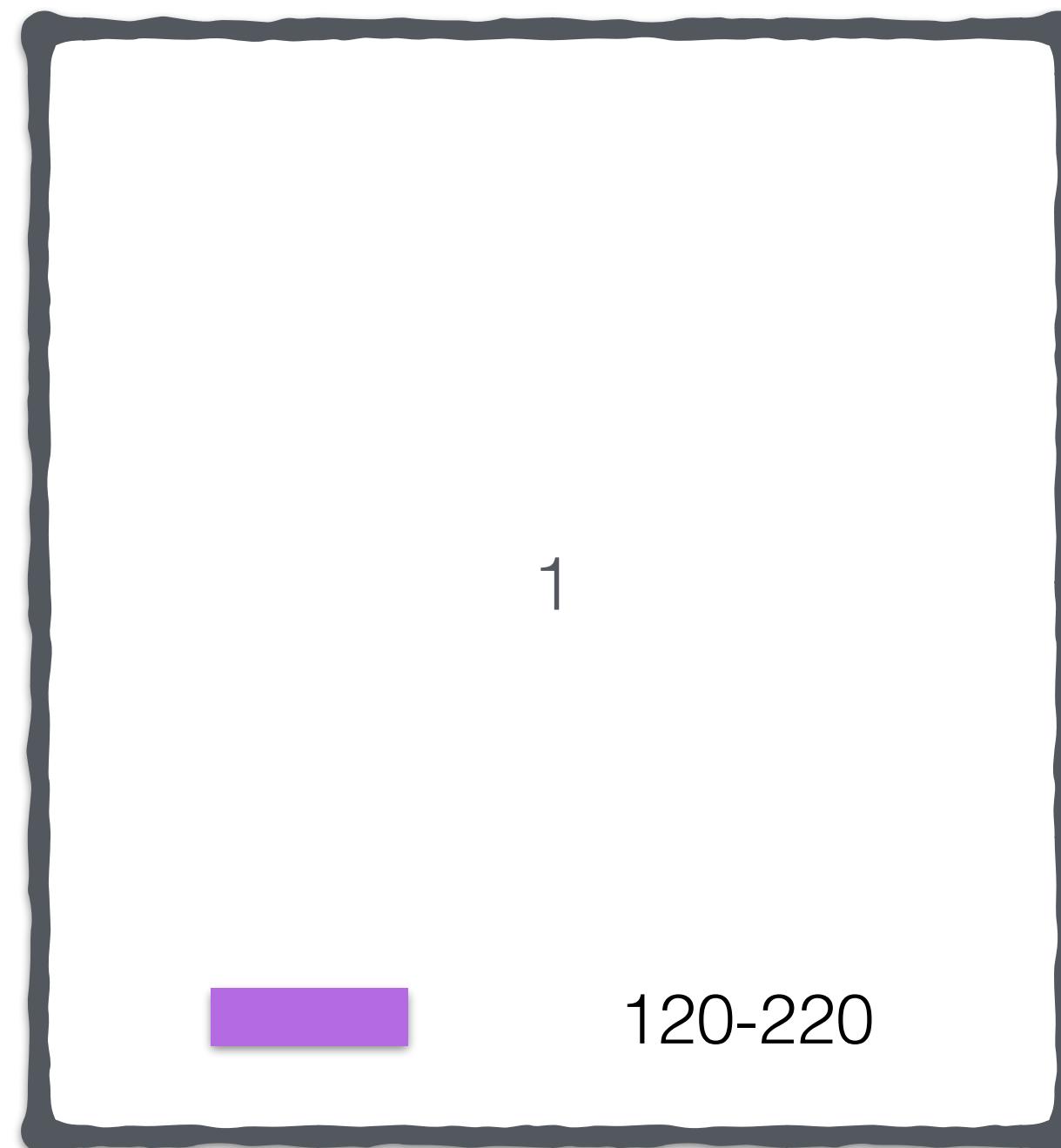
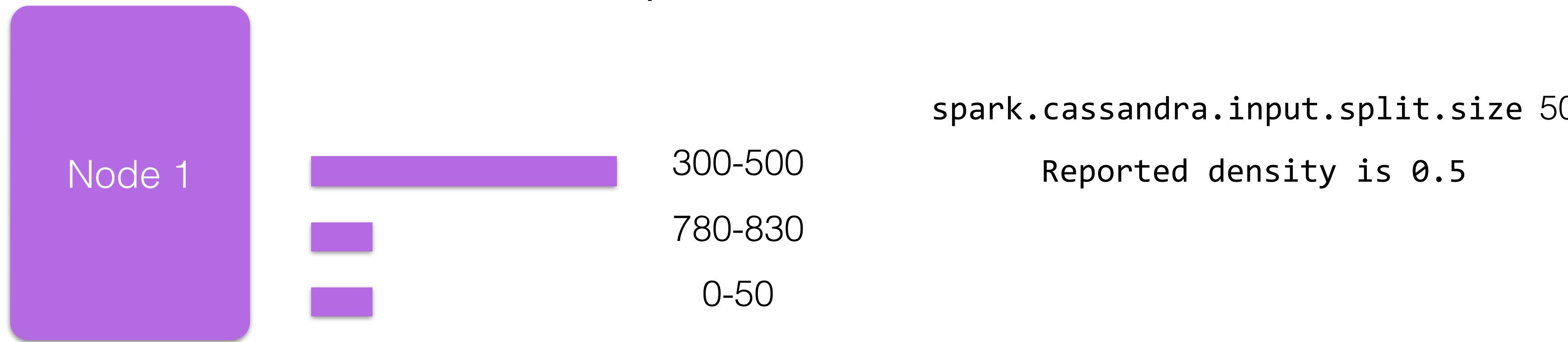
The Connector Uses Information on the Node to Make Spark Partitions



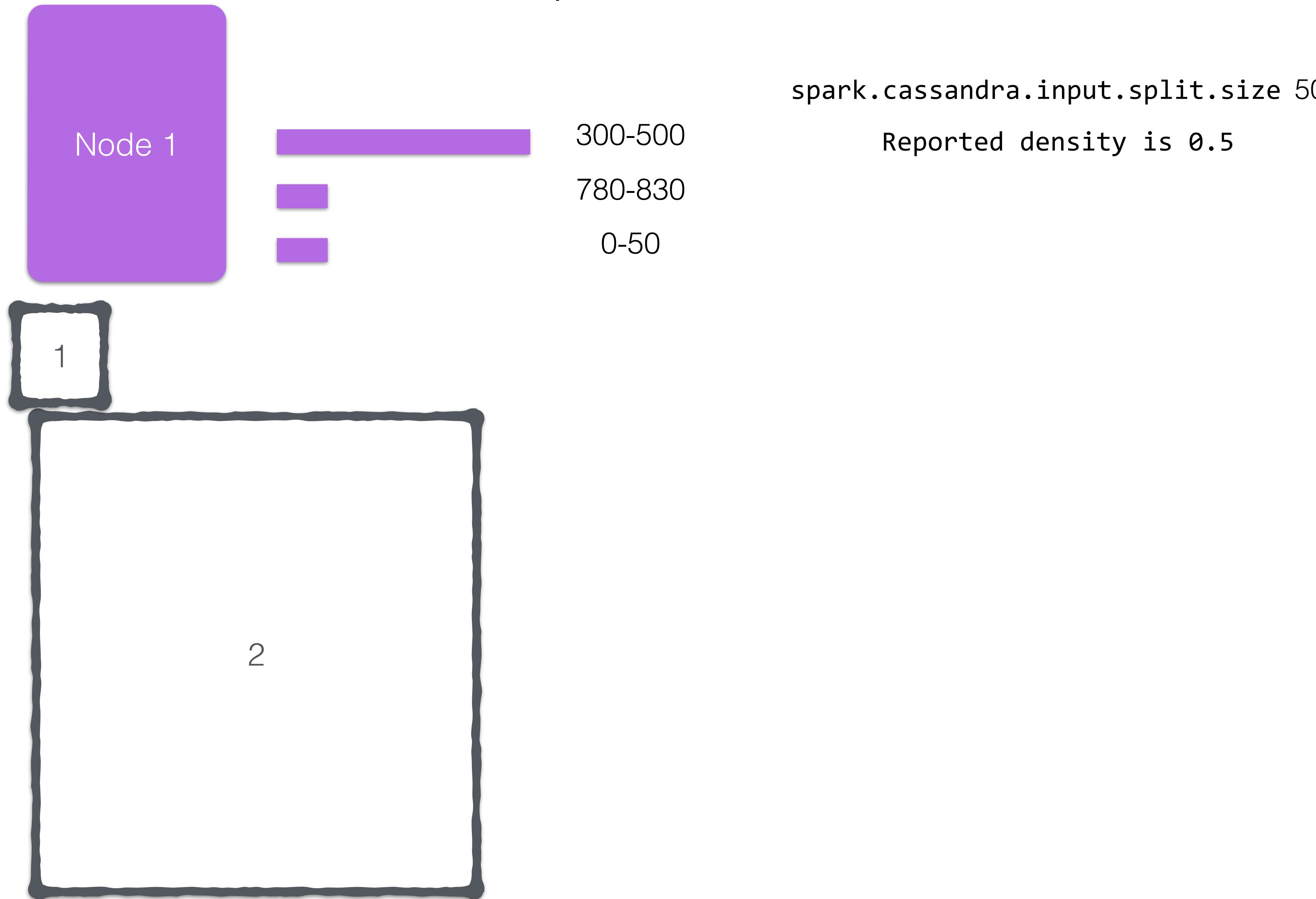
The Connector Uses Information on the Node to Make Spark Partitions



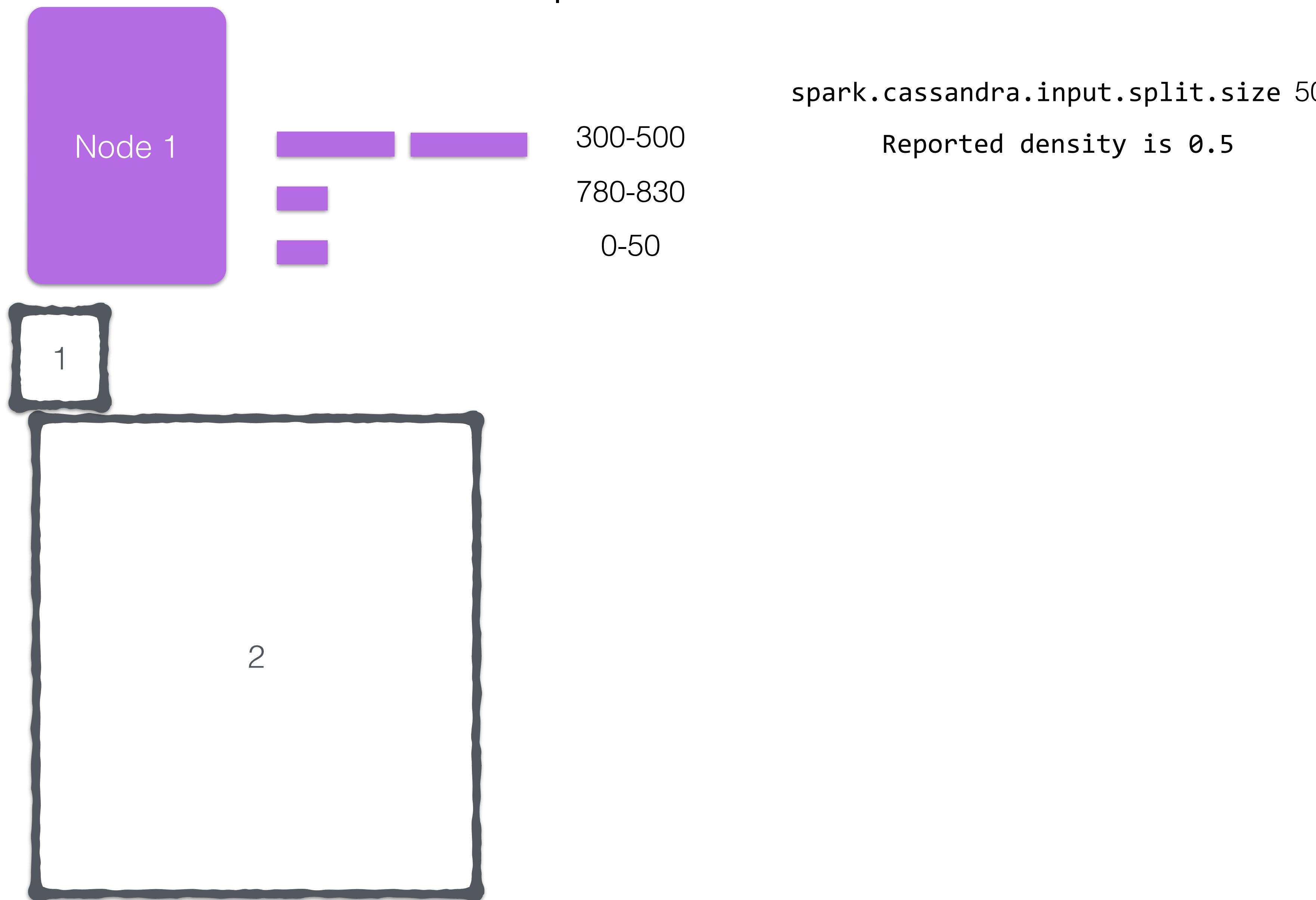
The Connector Uses Information on the Node to Make Spark Partitions



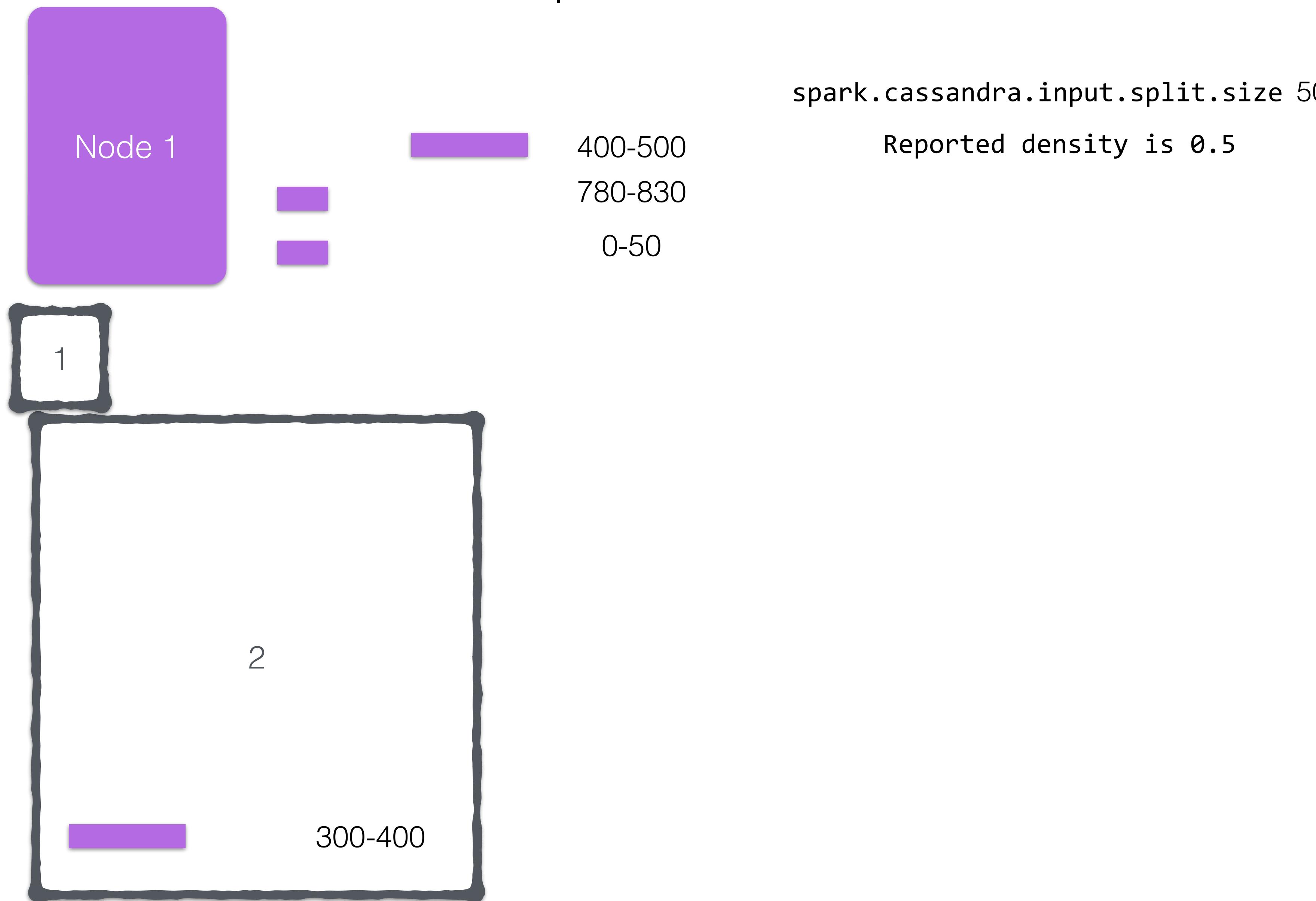
The Connector Uses Information on the Node to Make Spark Partitions



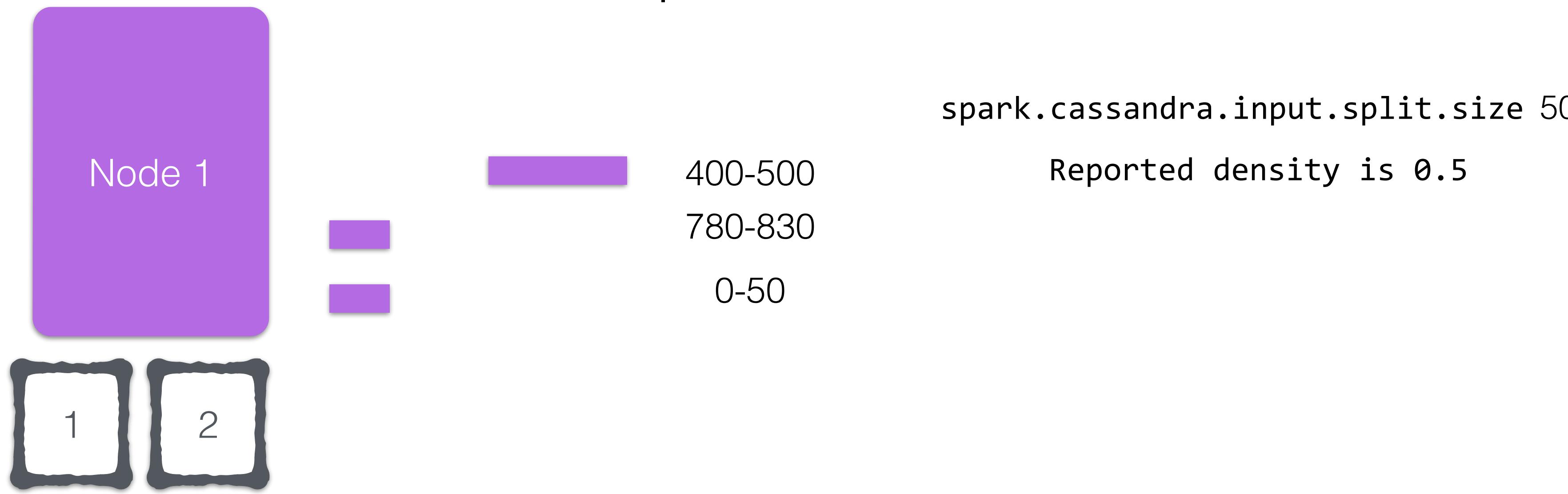
The Connector Uses Information on the Node to Make Spark Partitions



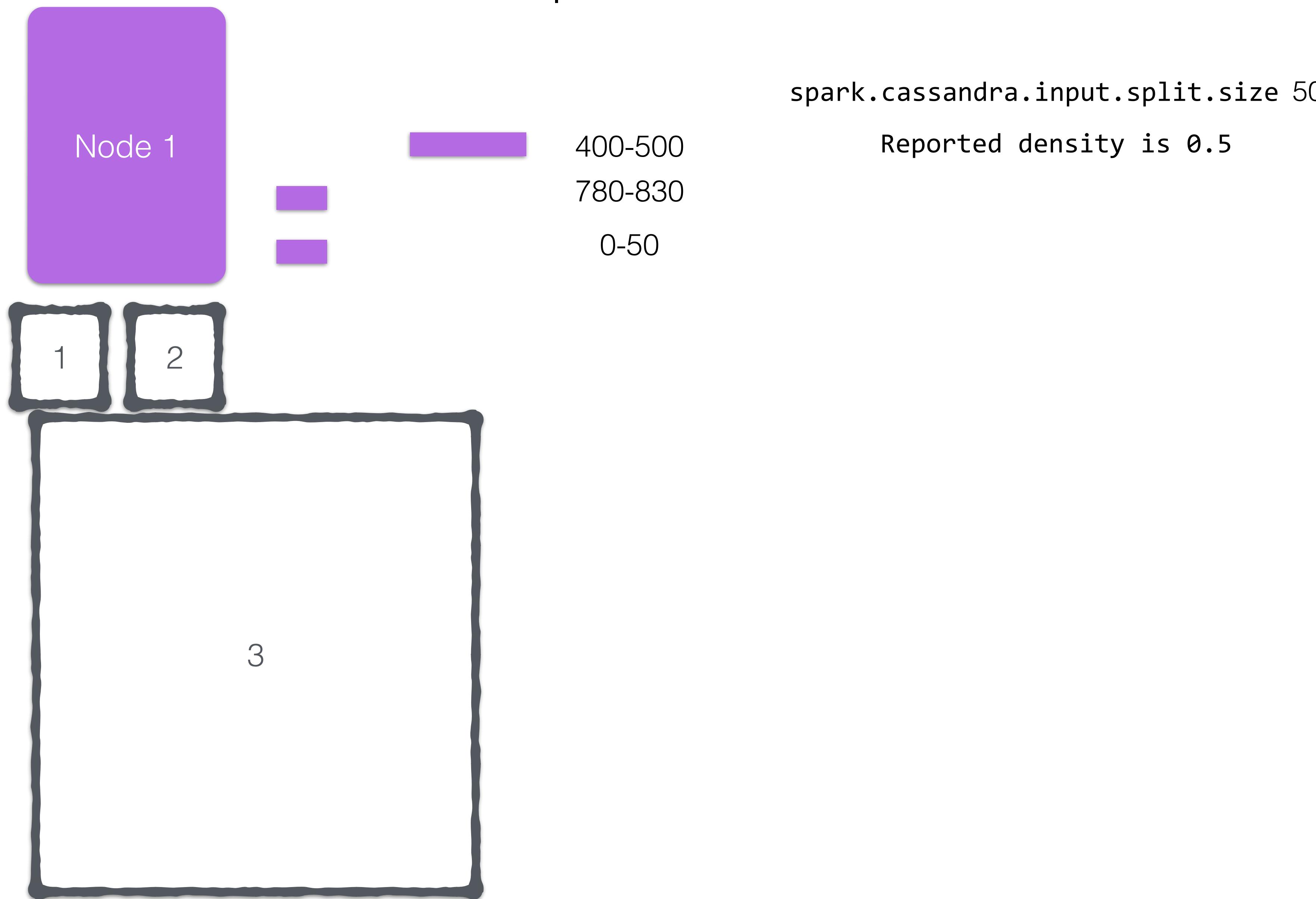
The Connector Uses Information on the Node to Make Spark Partitions



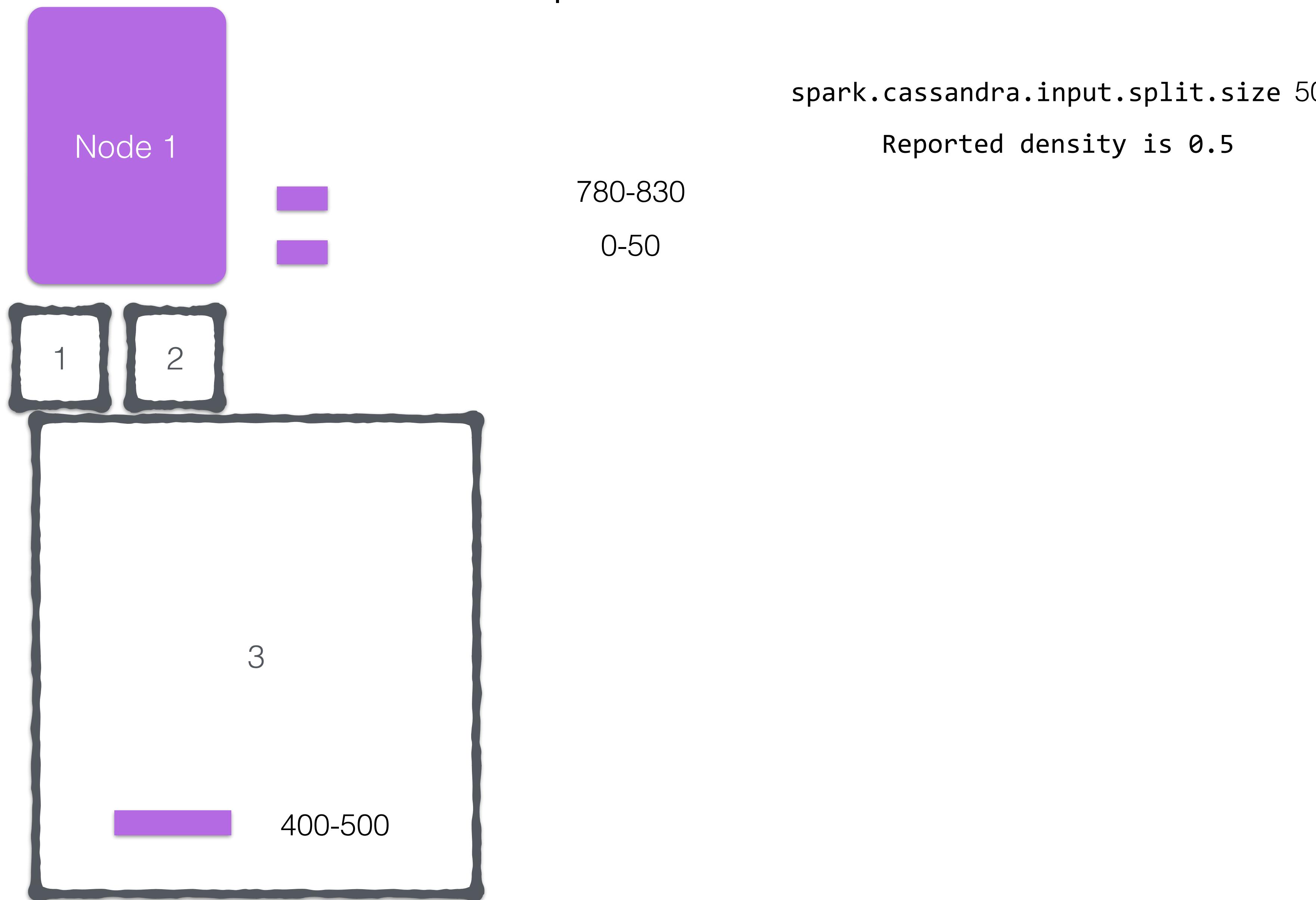
The Connector Uses Information on the Node to Make Spark Partitions



The Connector Uses Information on the Node to Make Spark Partitions



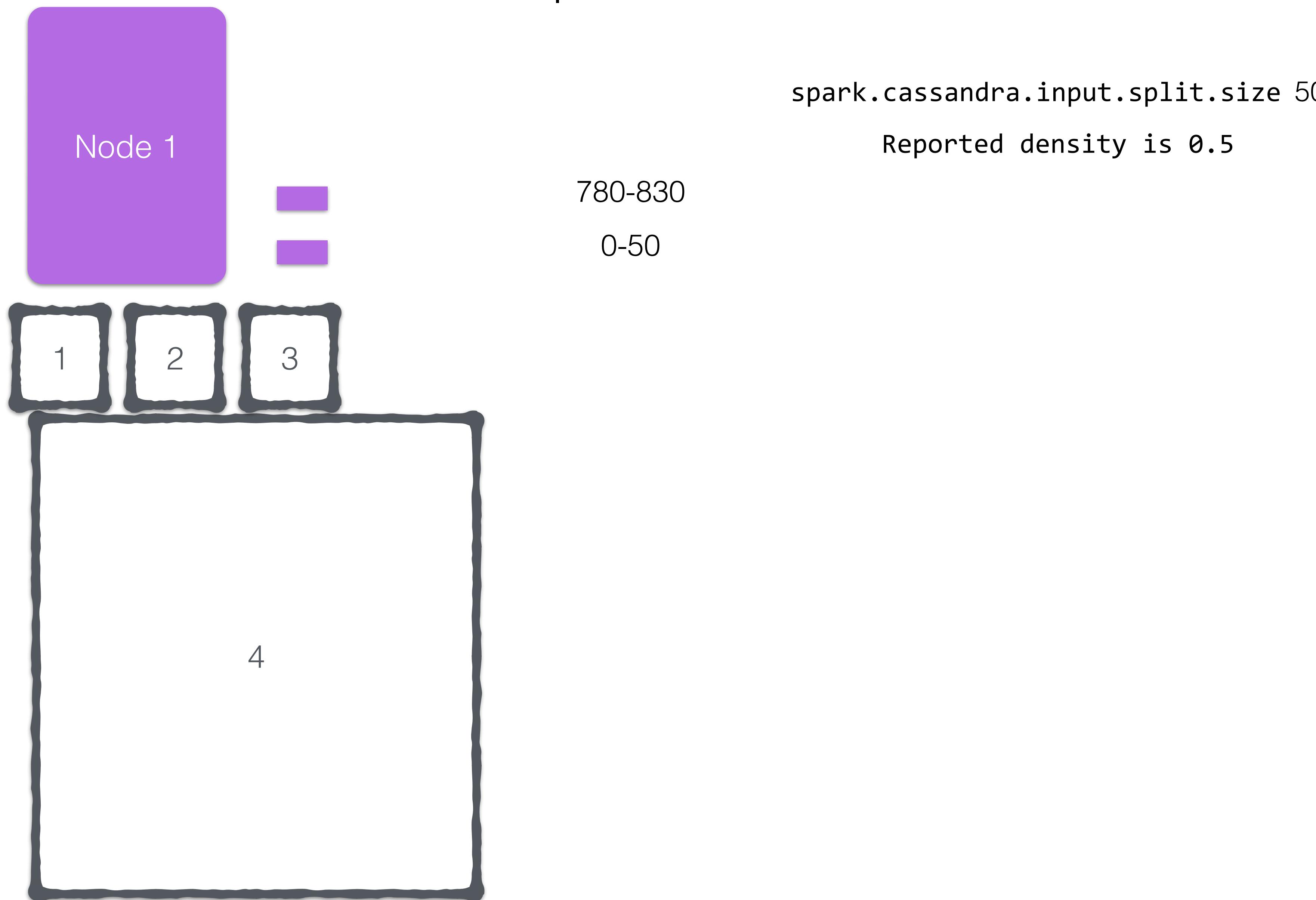
The Connector Uses Information on the Node to Make Spark Partitions



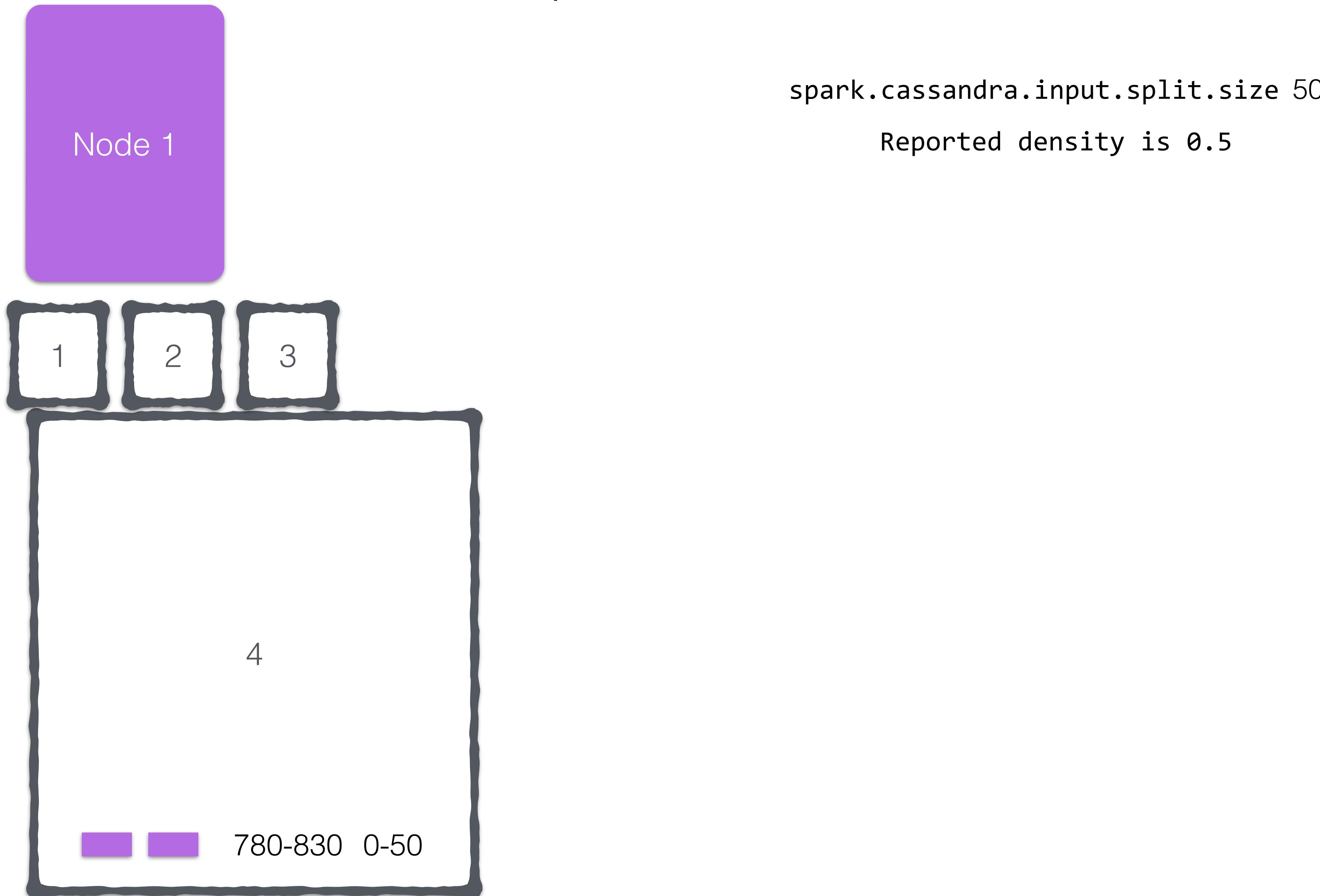
The Connector Uses Information on the Node to Make Spark Partitions



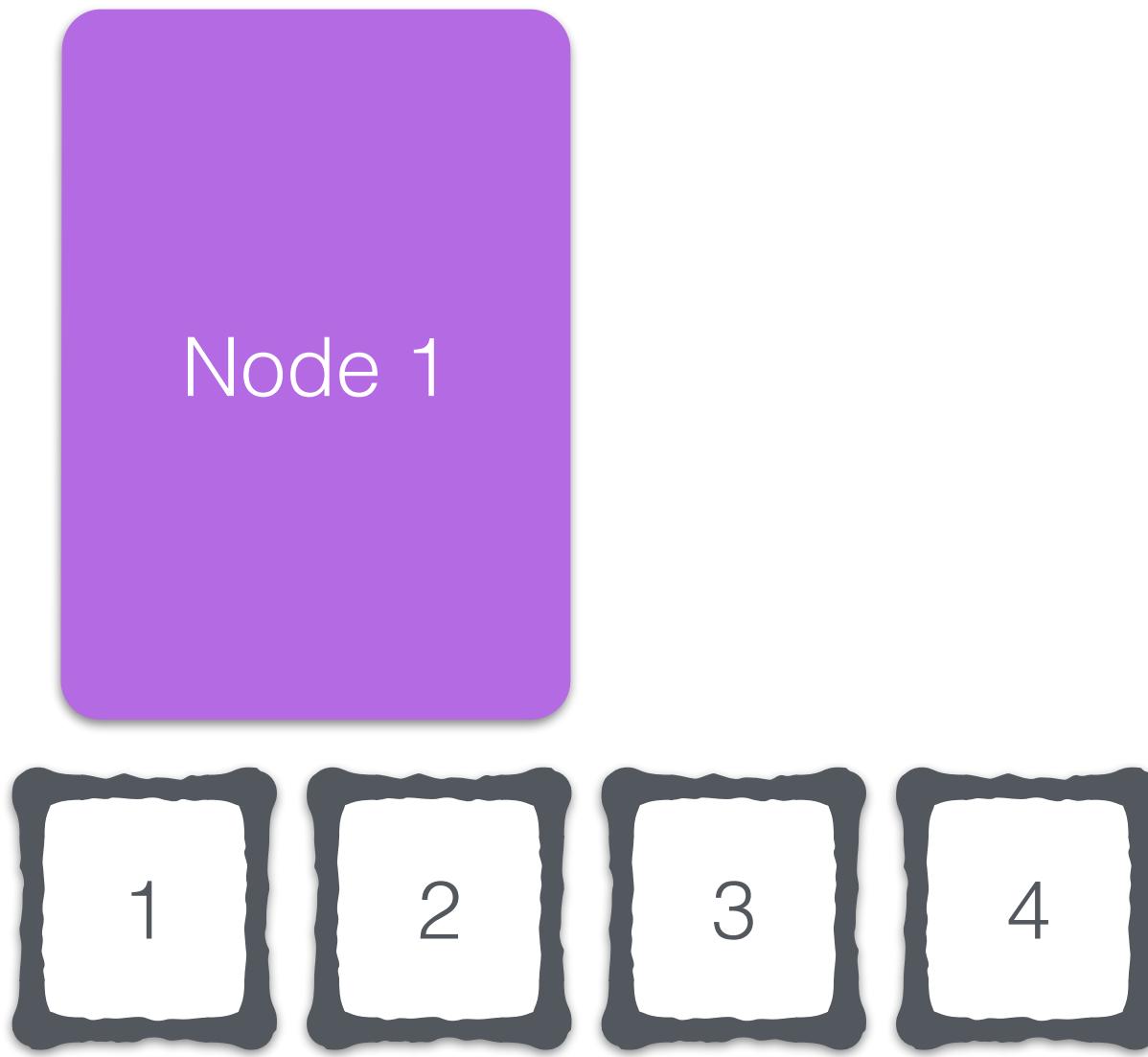
The Connector Uses Information on the Node to Make Spark Partitions



The Connector Uses Information on the Node to Make Spark Partitions



The Connector Uses Information on the Node to Make Spark Partitions

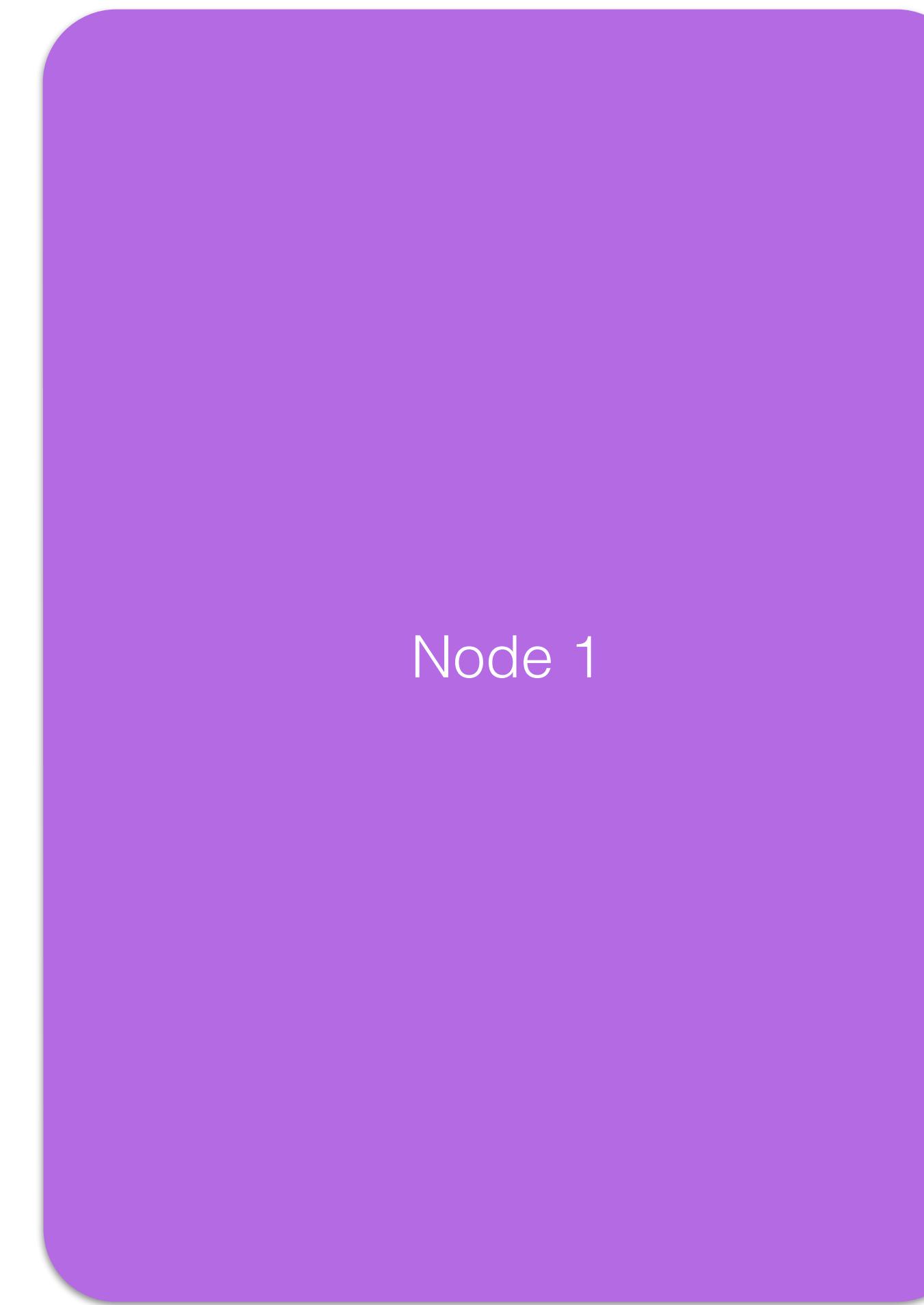
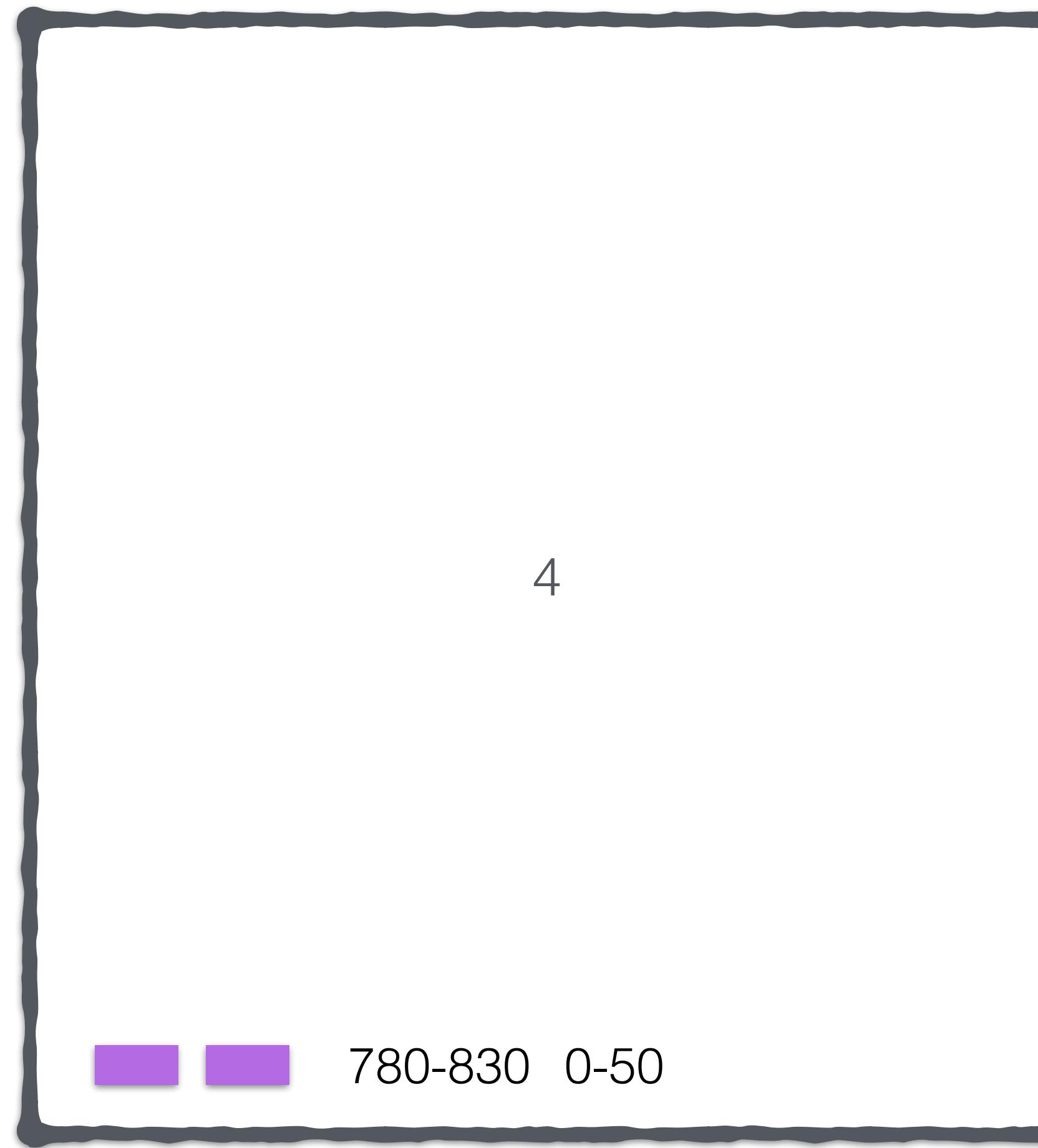


`spark.cassandra.input.split.size 50`

Reported density is 0.5

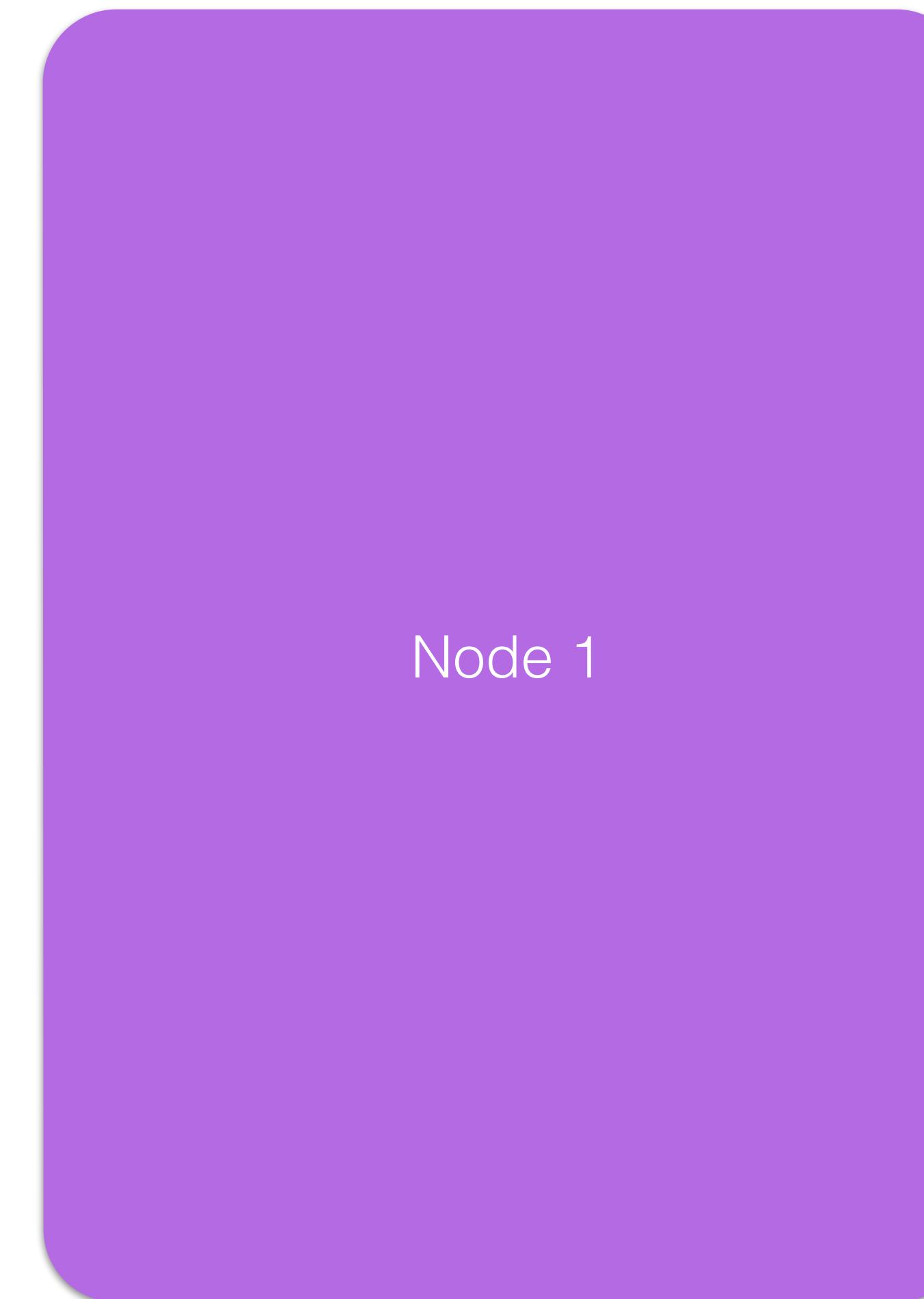
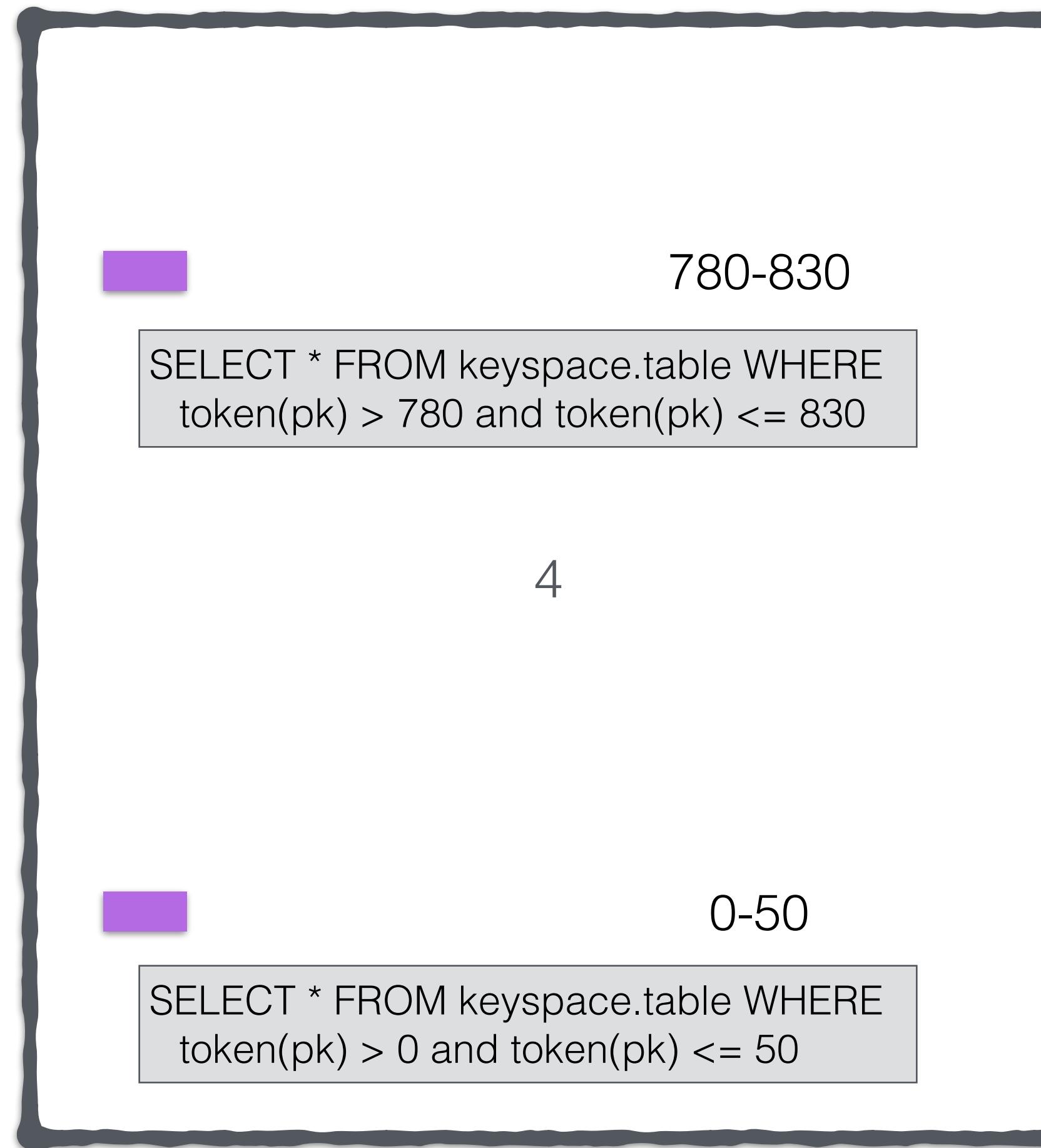
Data is Retrieved Using the DataStax Java Driver

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



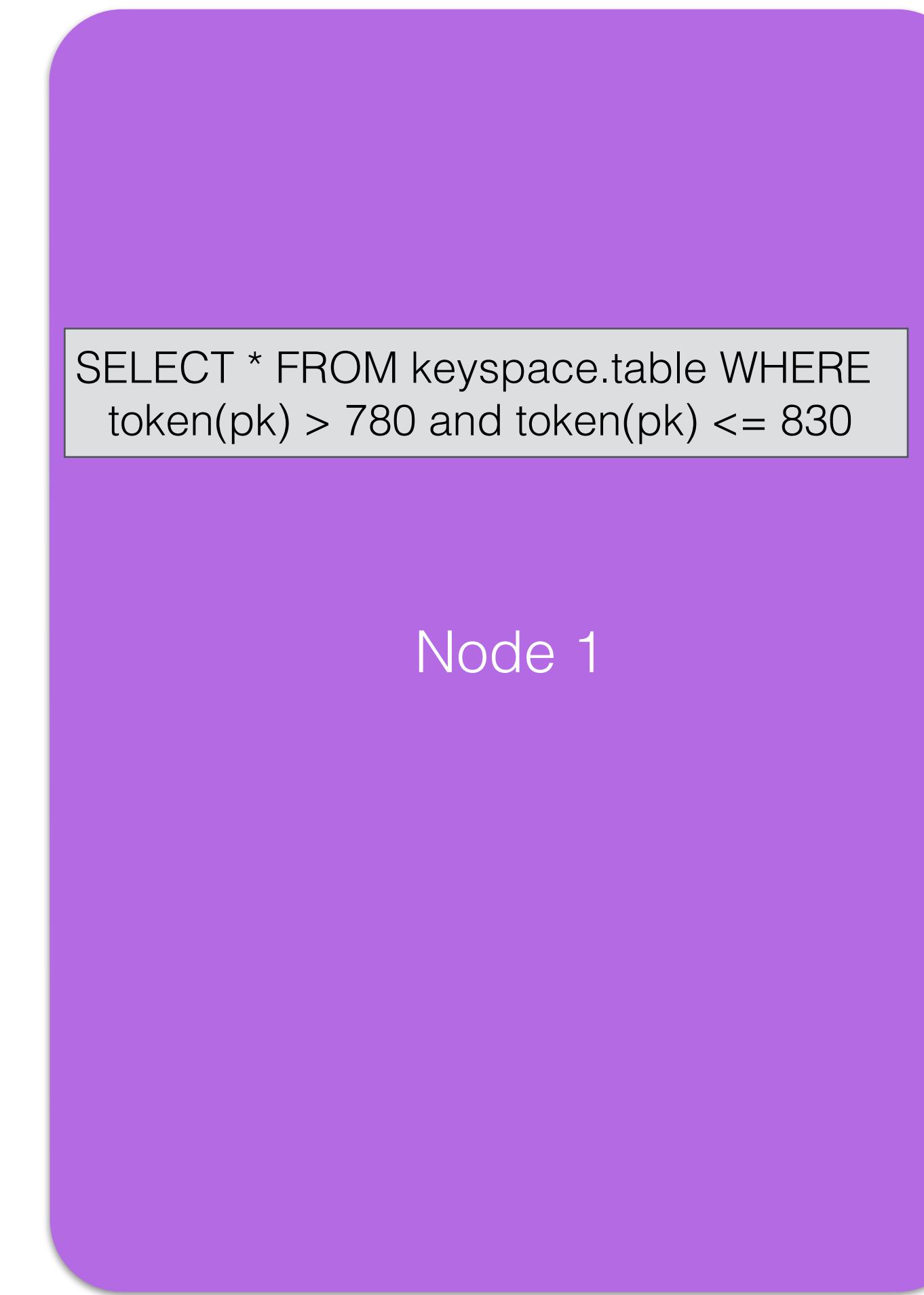
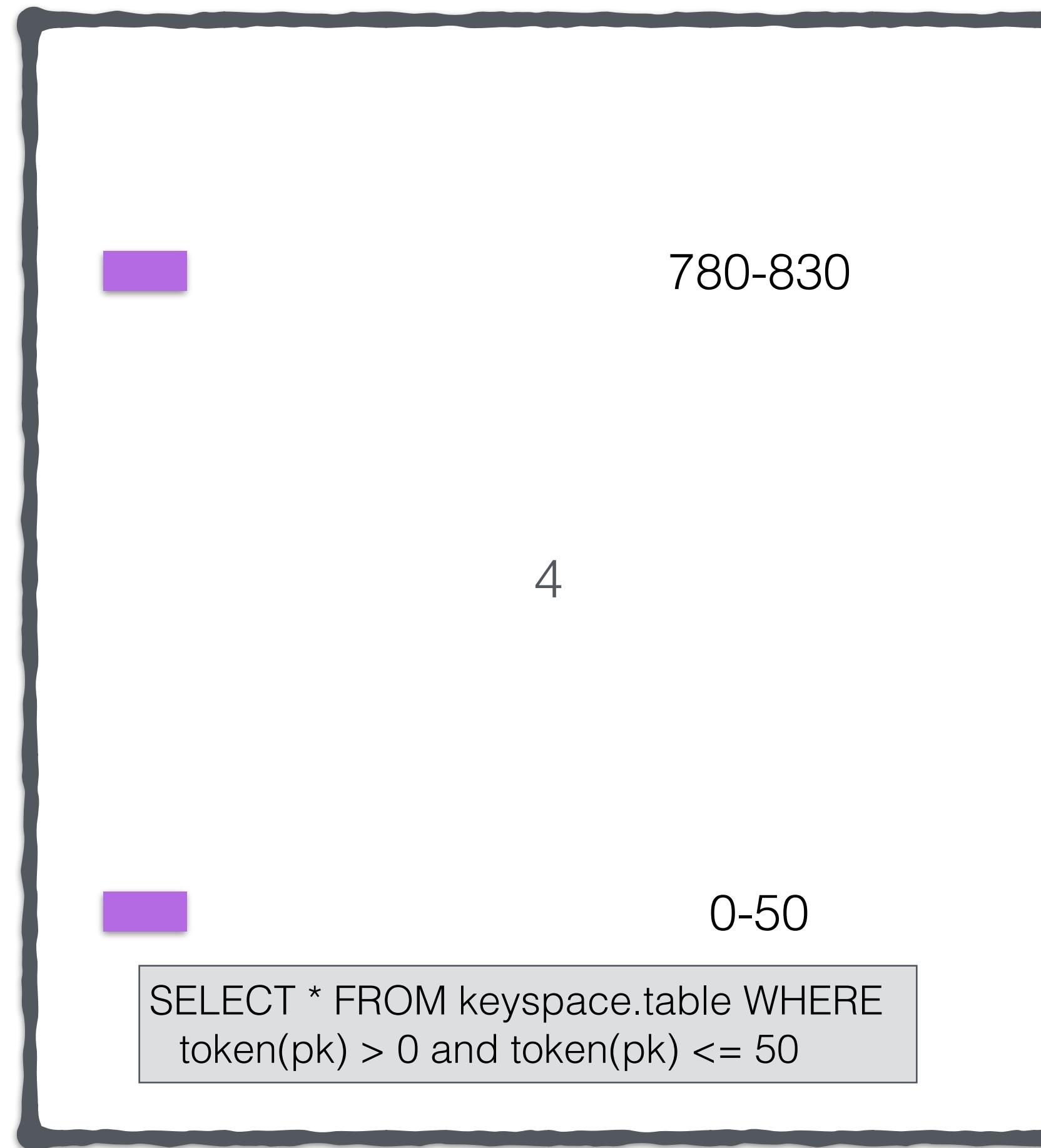
Data is Retrieved Using the DataStax Java Driver

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



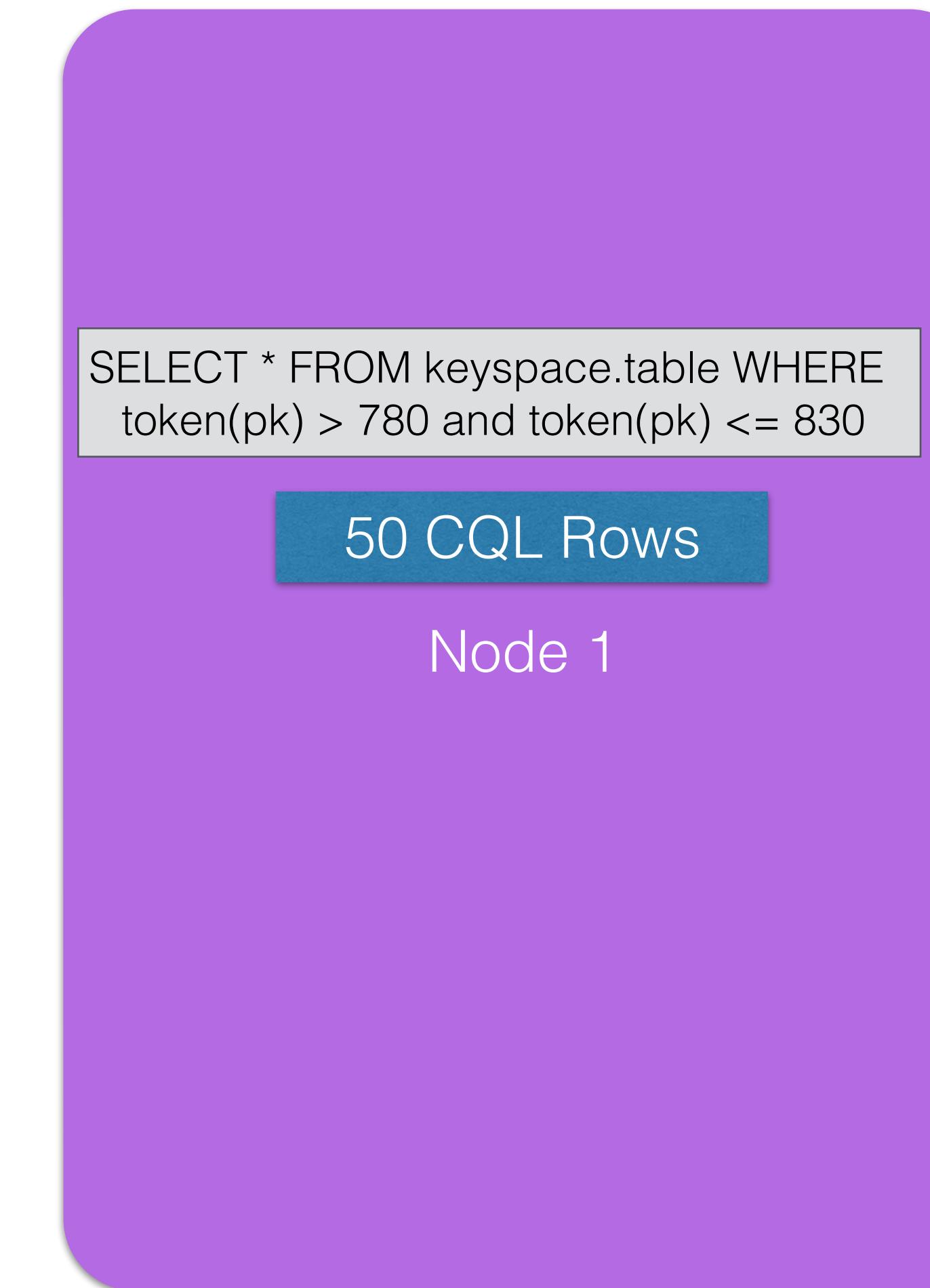
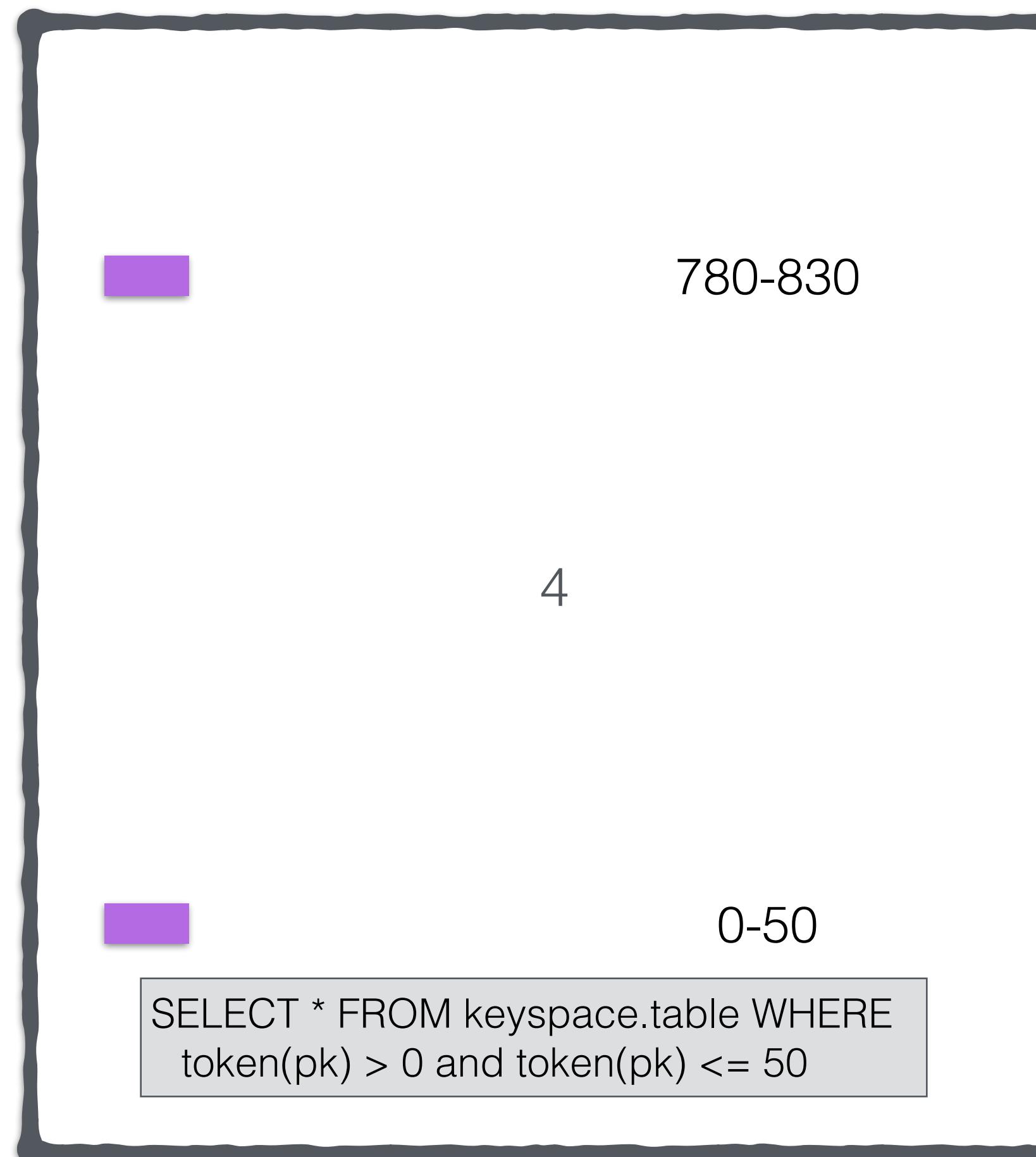
Data is Retrieved Using the DataStax Java Driver

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



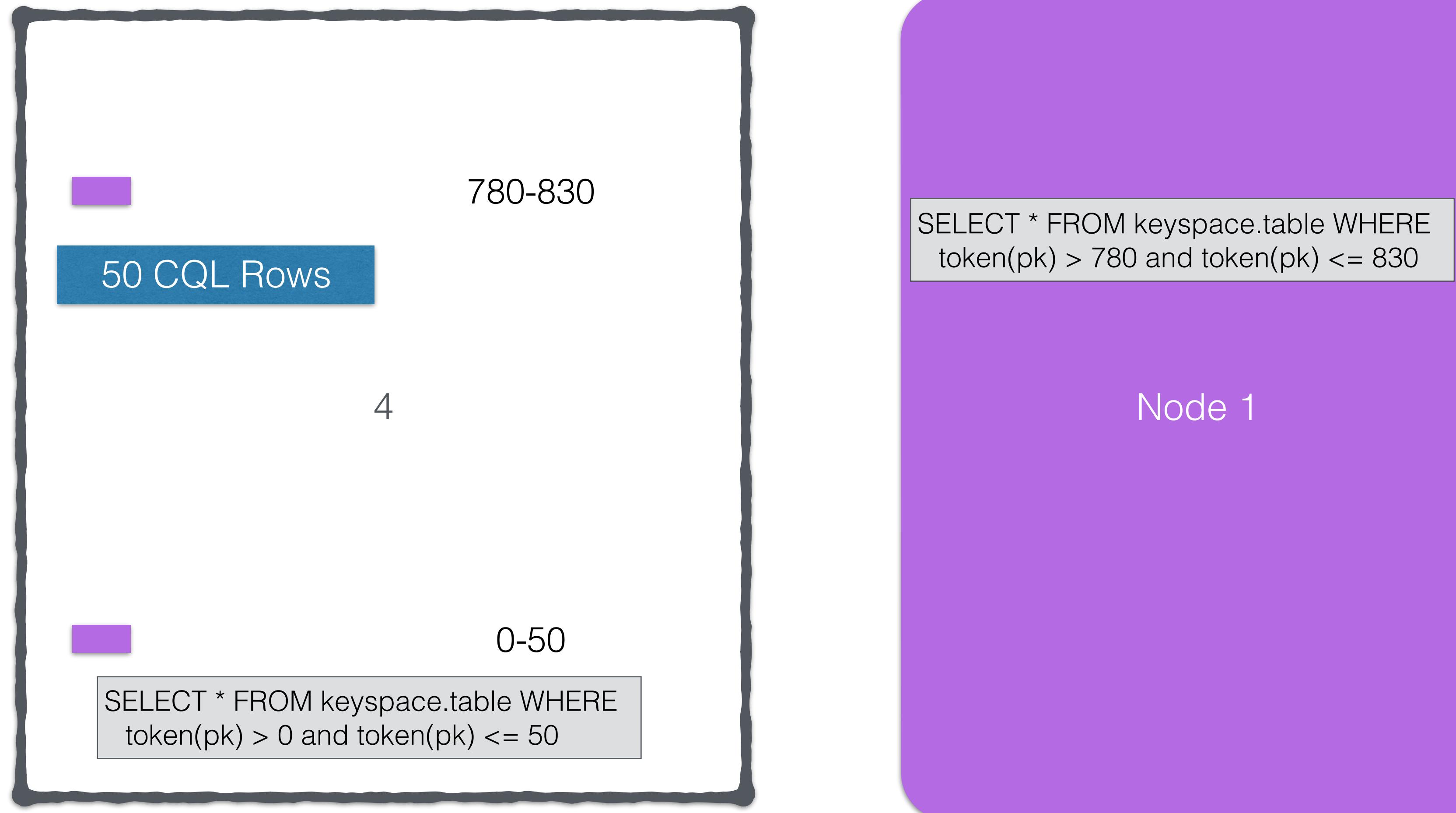
Data is Retrieved Using the DataStax Java Driver

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



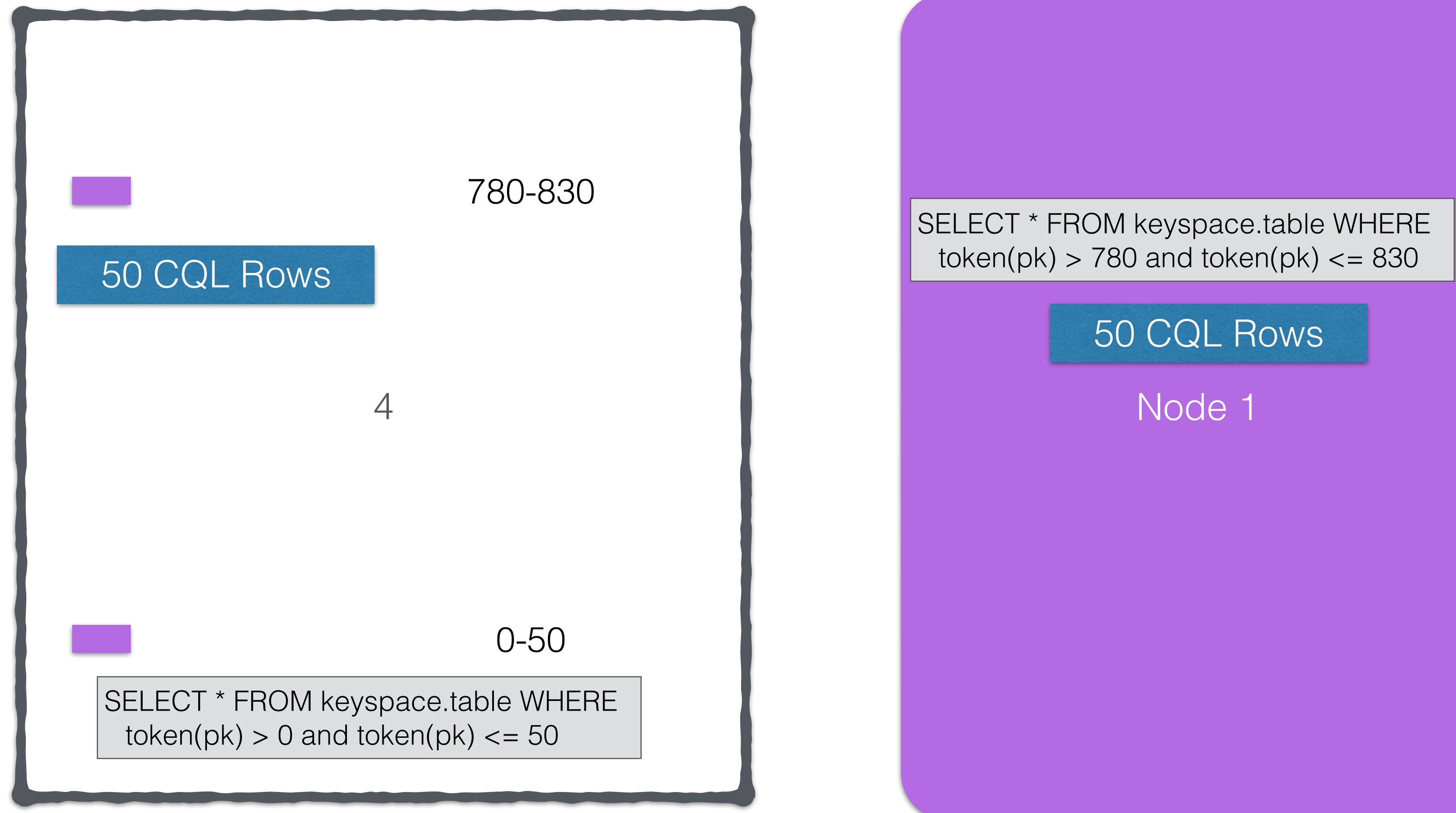
Data is Retrieved Using the DataStax Java Driver

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



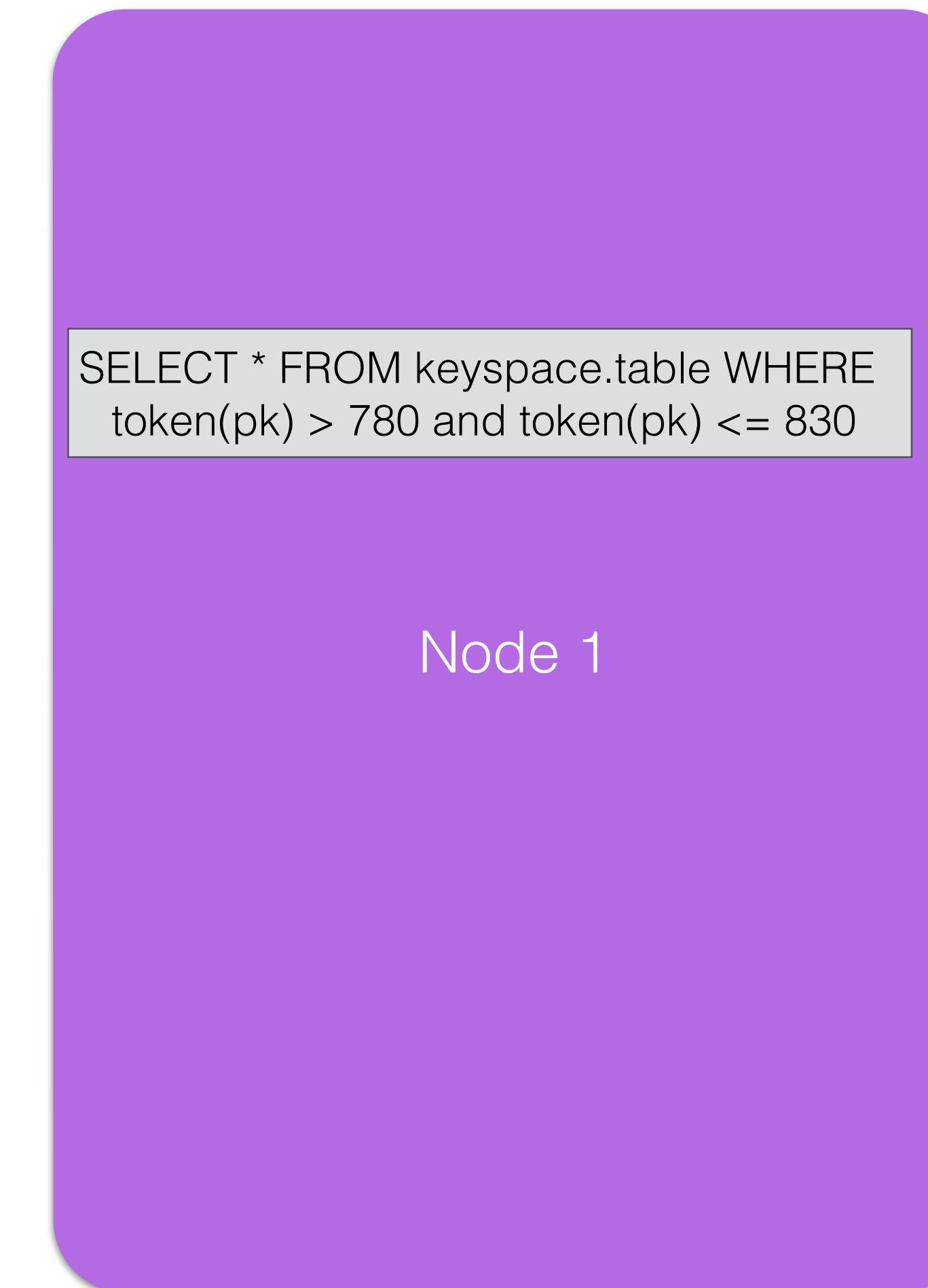
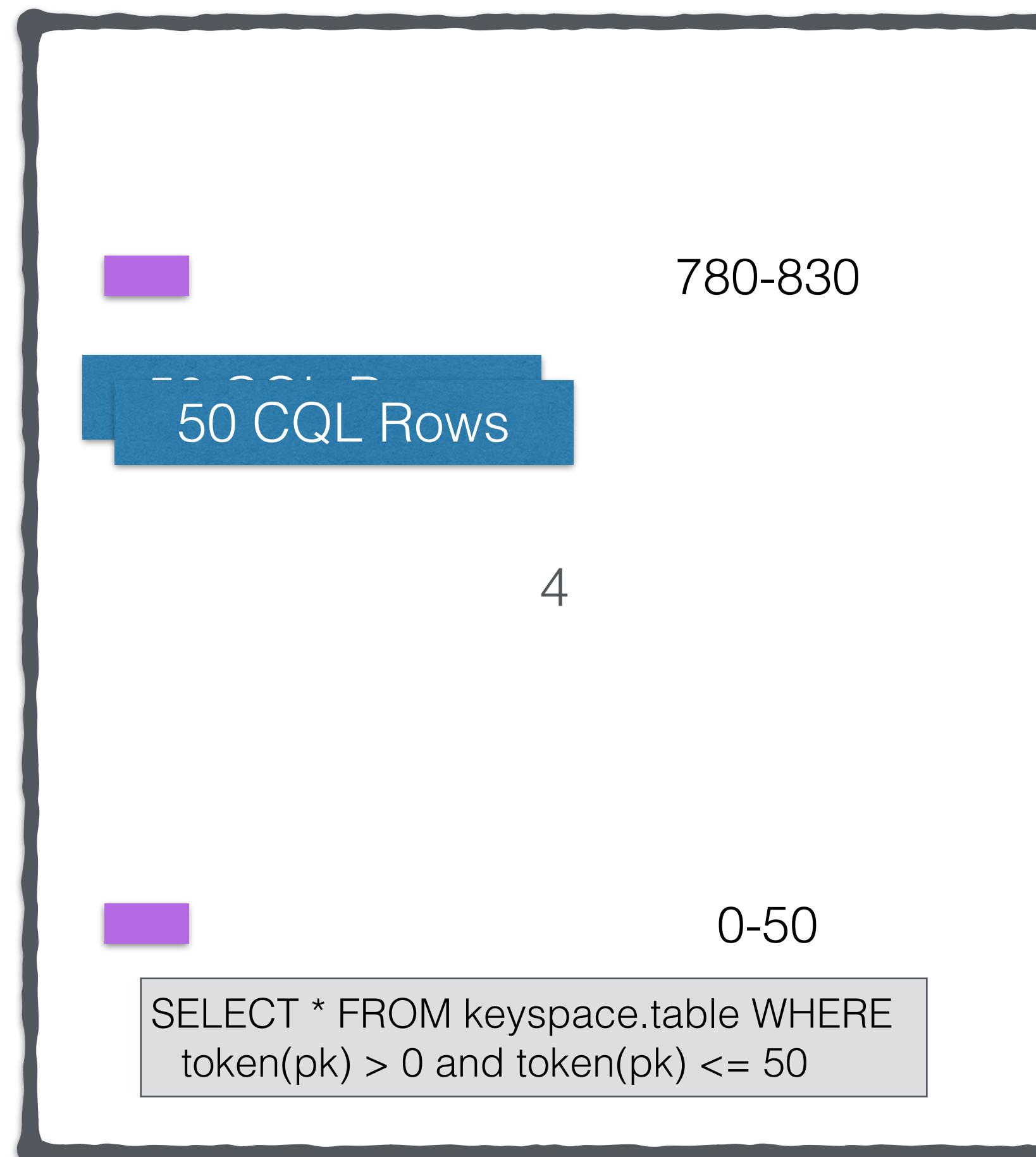
Data is Retrieved Using the DataStax Java Driver

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



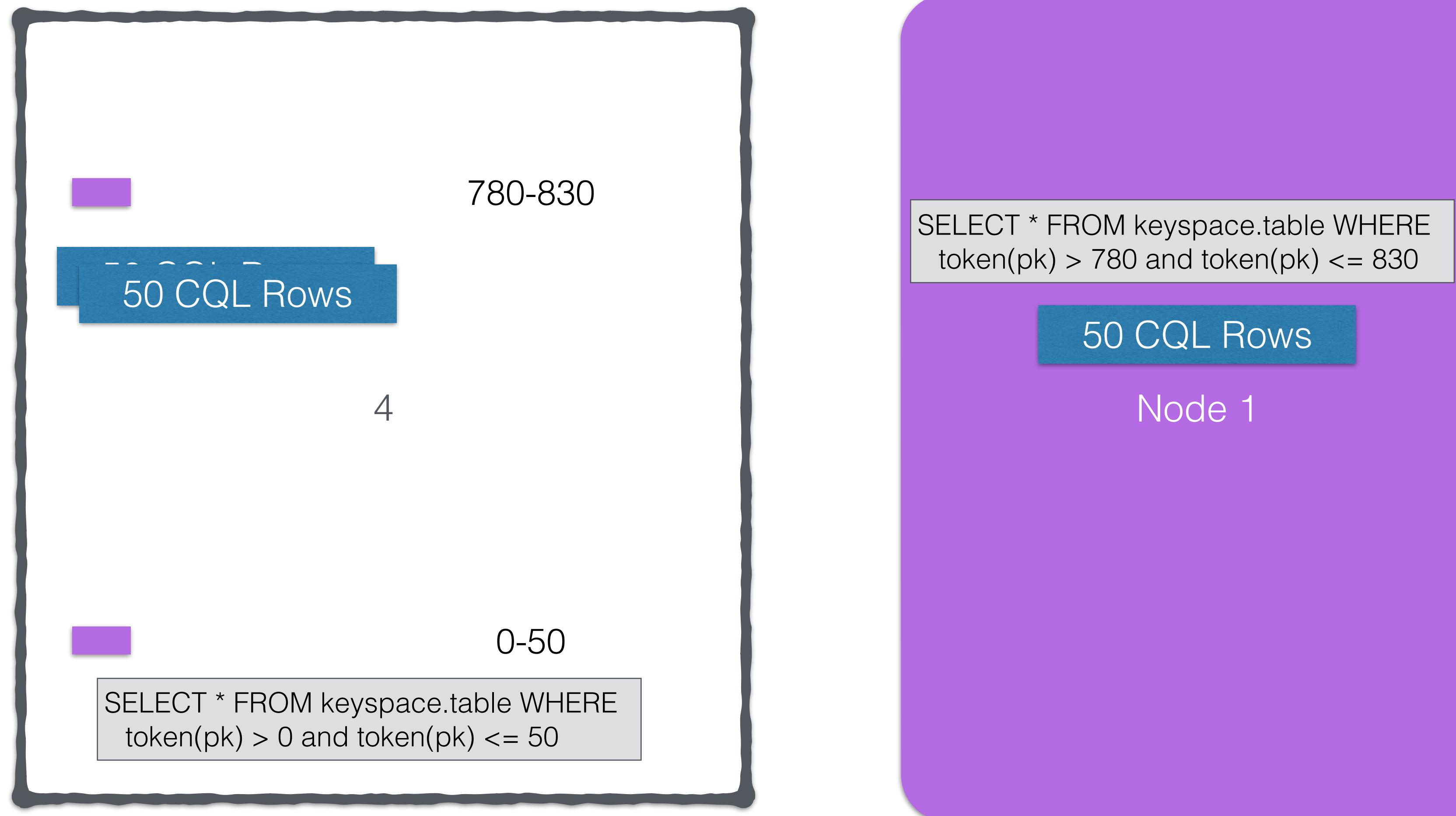
Data is Retrieved Using the DataStax Java Driver

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



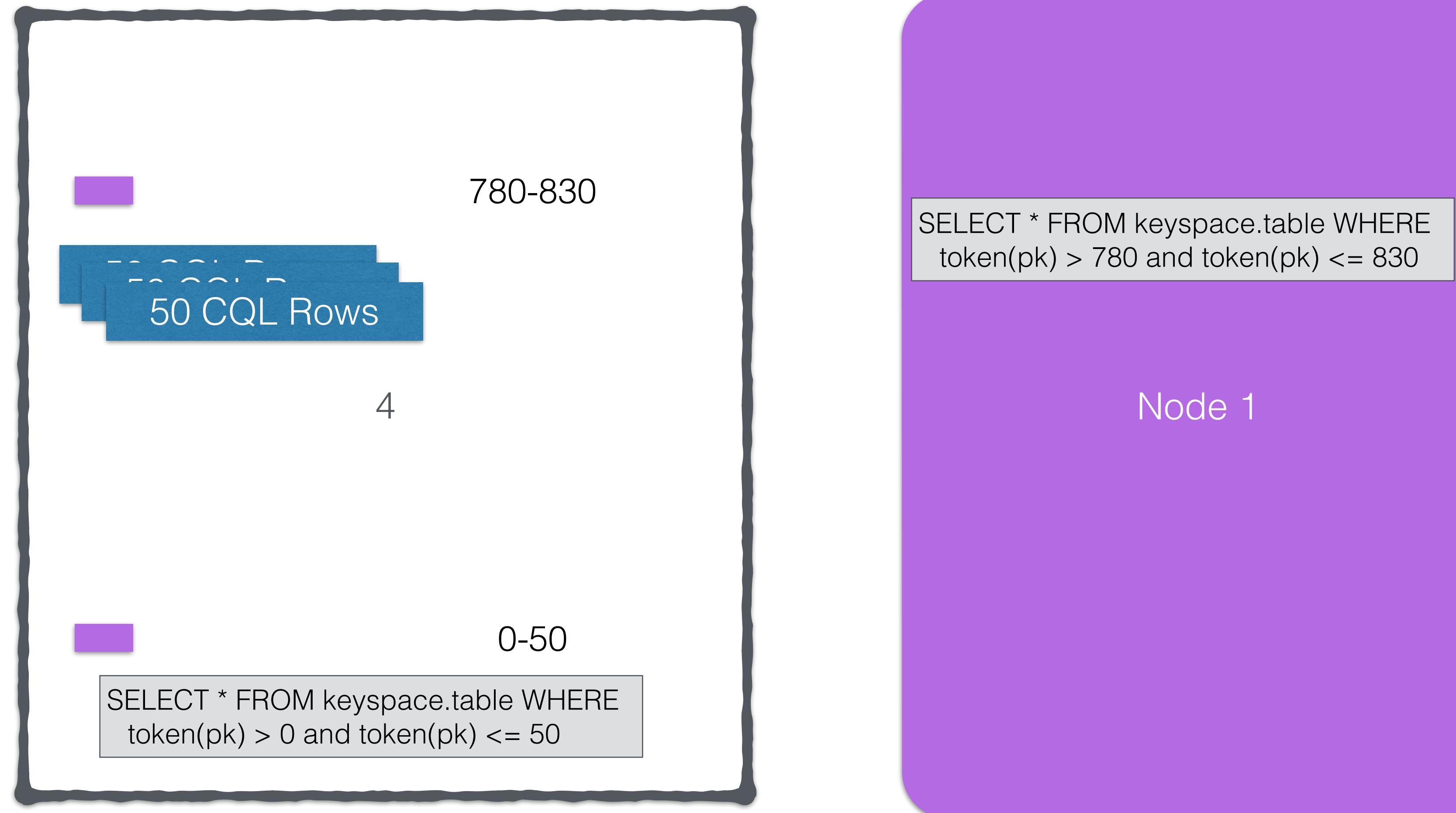
Data is Retrieved Using the DataStax Java Driver

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



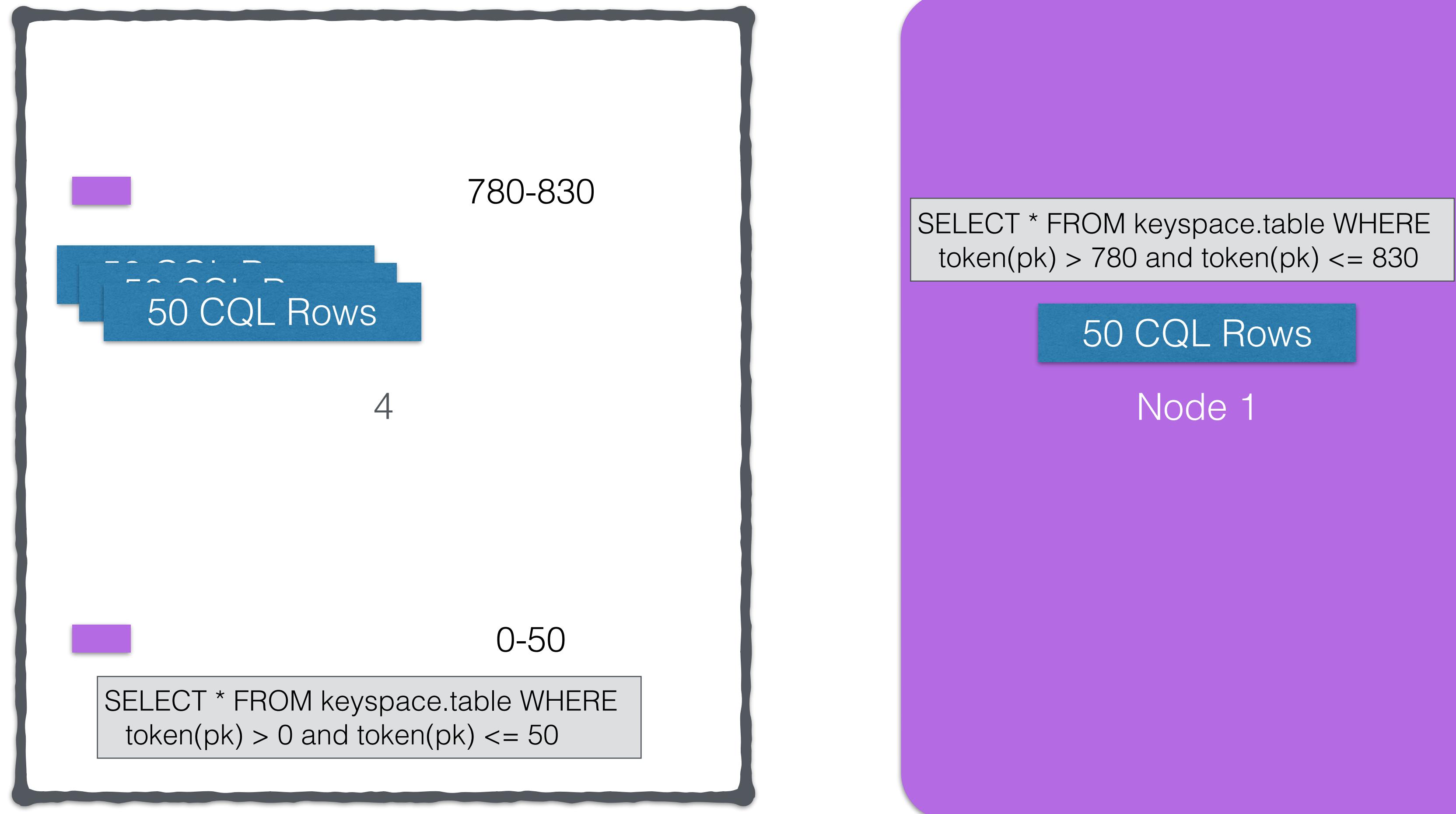
Data is Retrieved Using the DataStax Java Driver

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



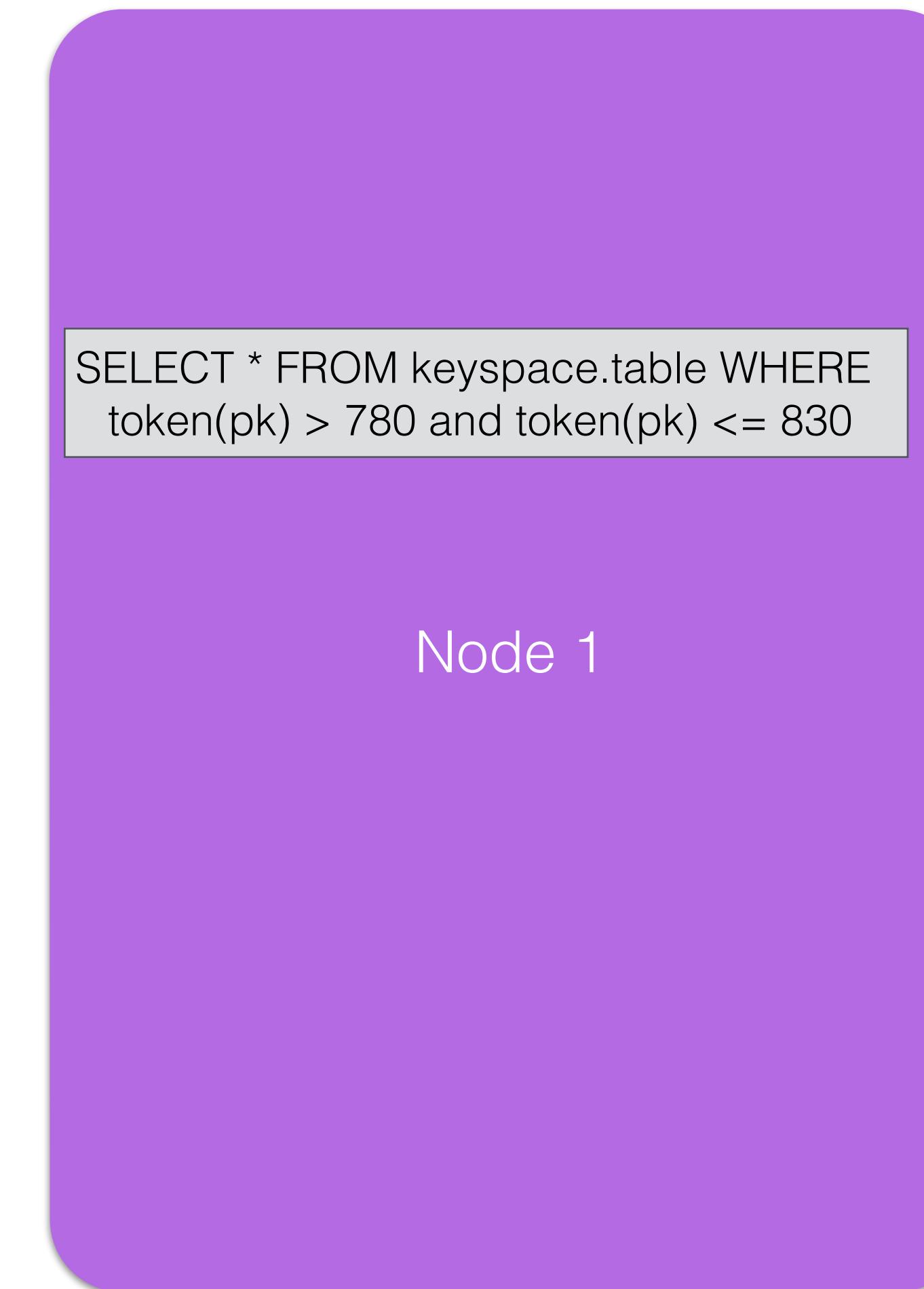
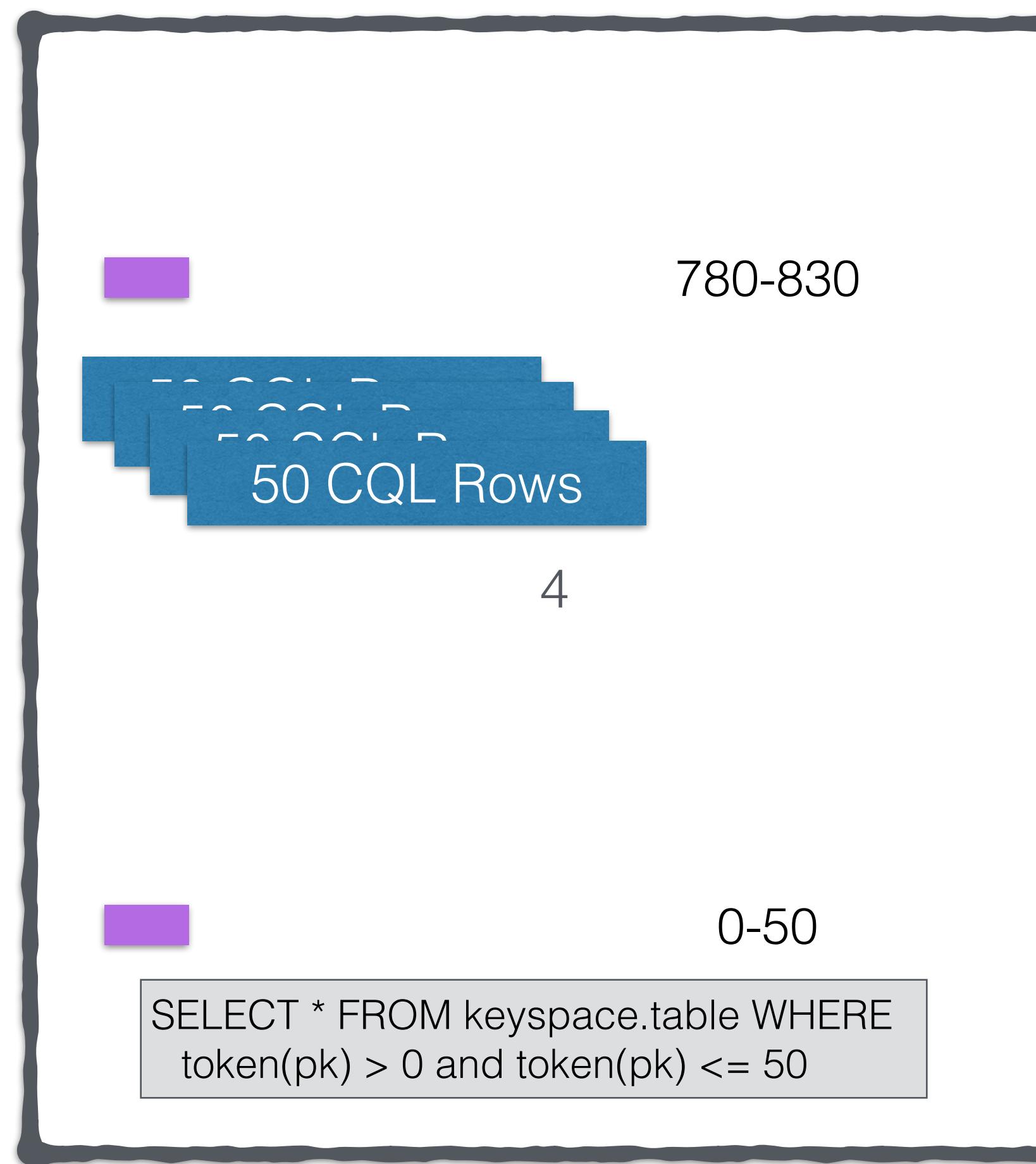
Data is Retrieved Using the DataStax Java Driver

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



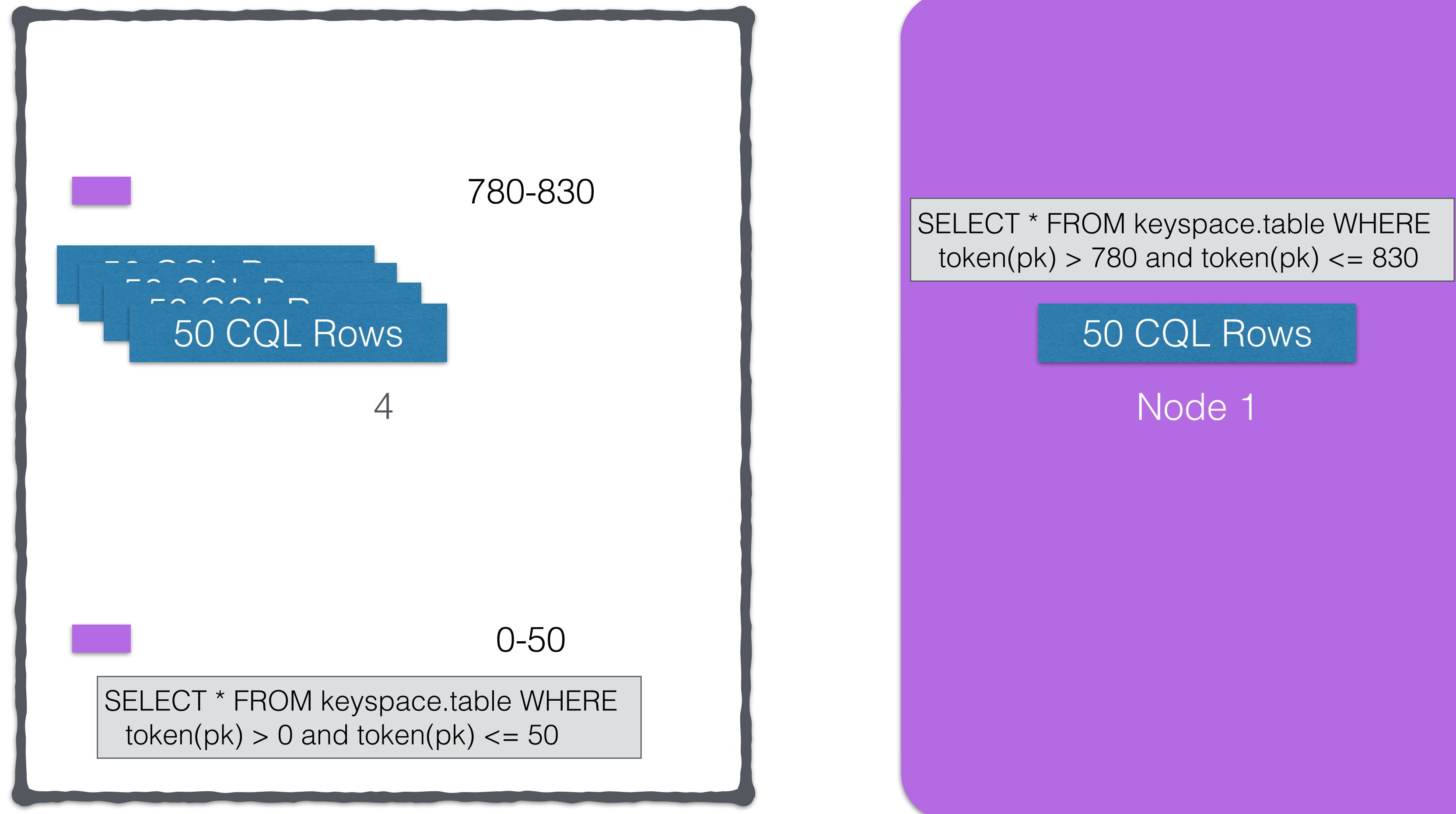
Data is Retrieved Using the DataStax Java Driver

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



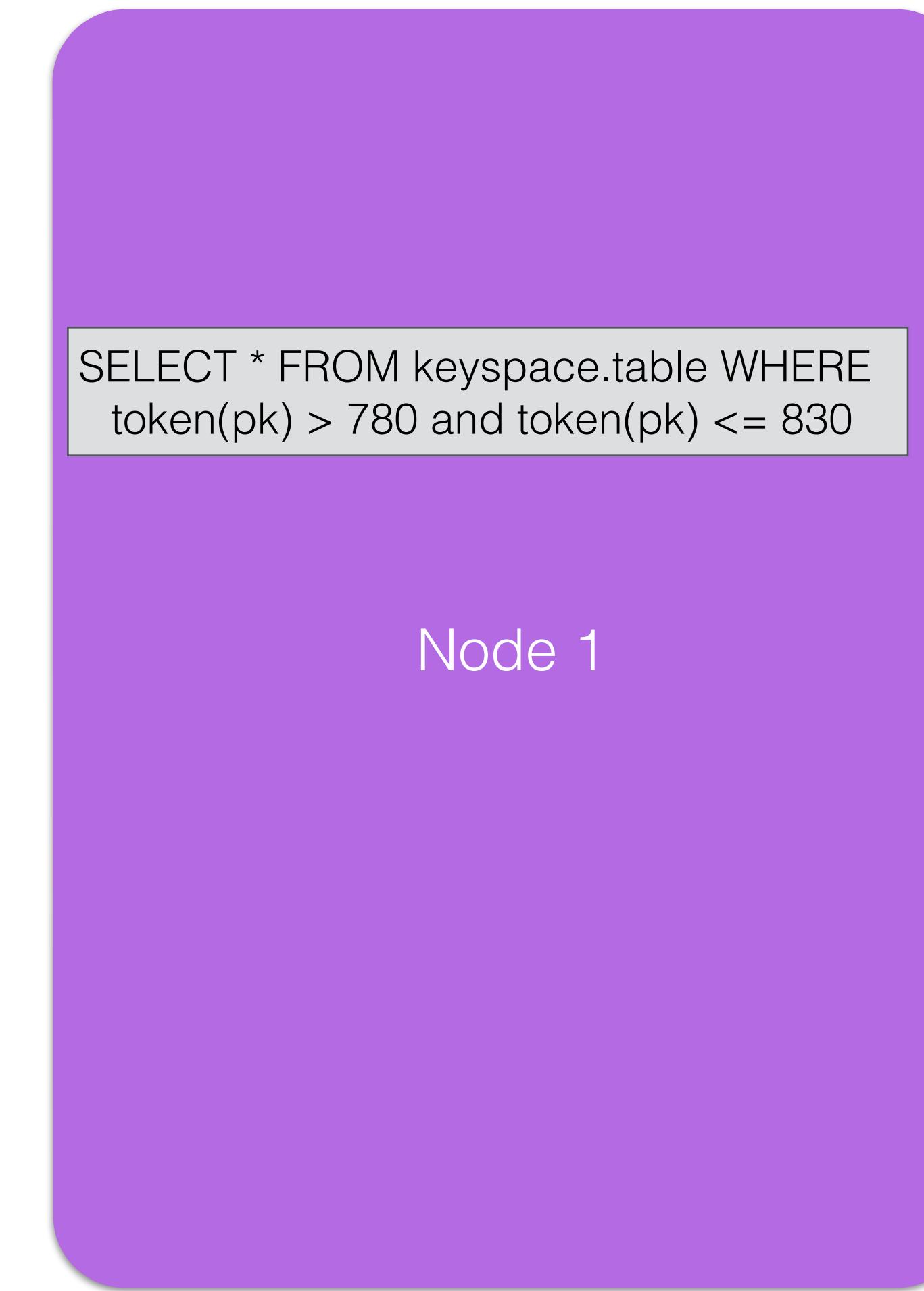
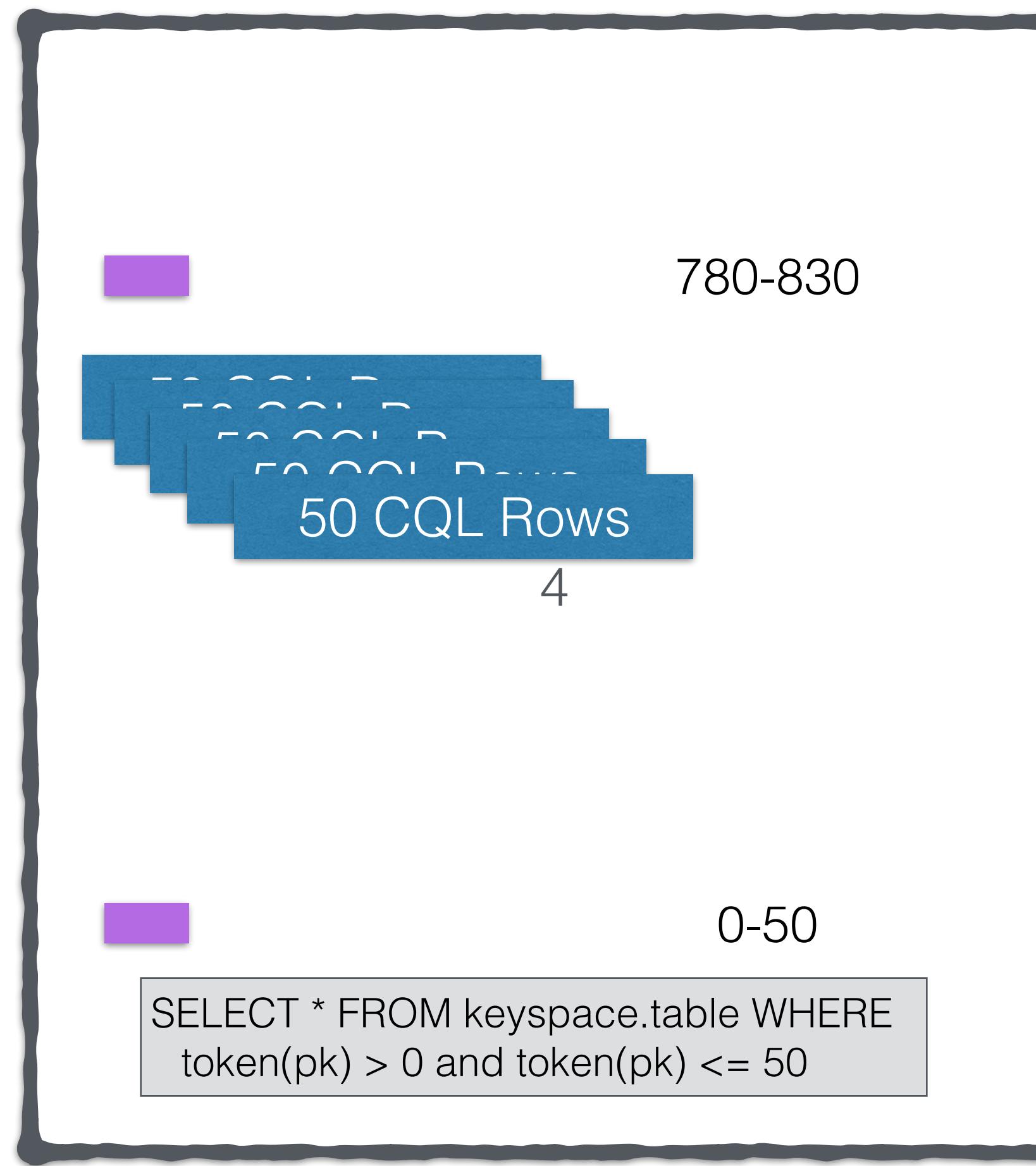
Data is Retrieved Using the DataStax Java Driver

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



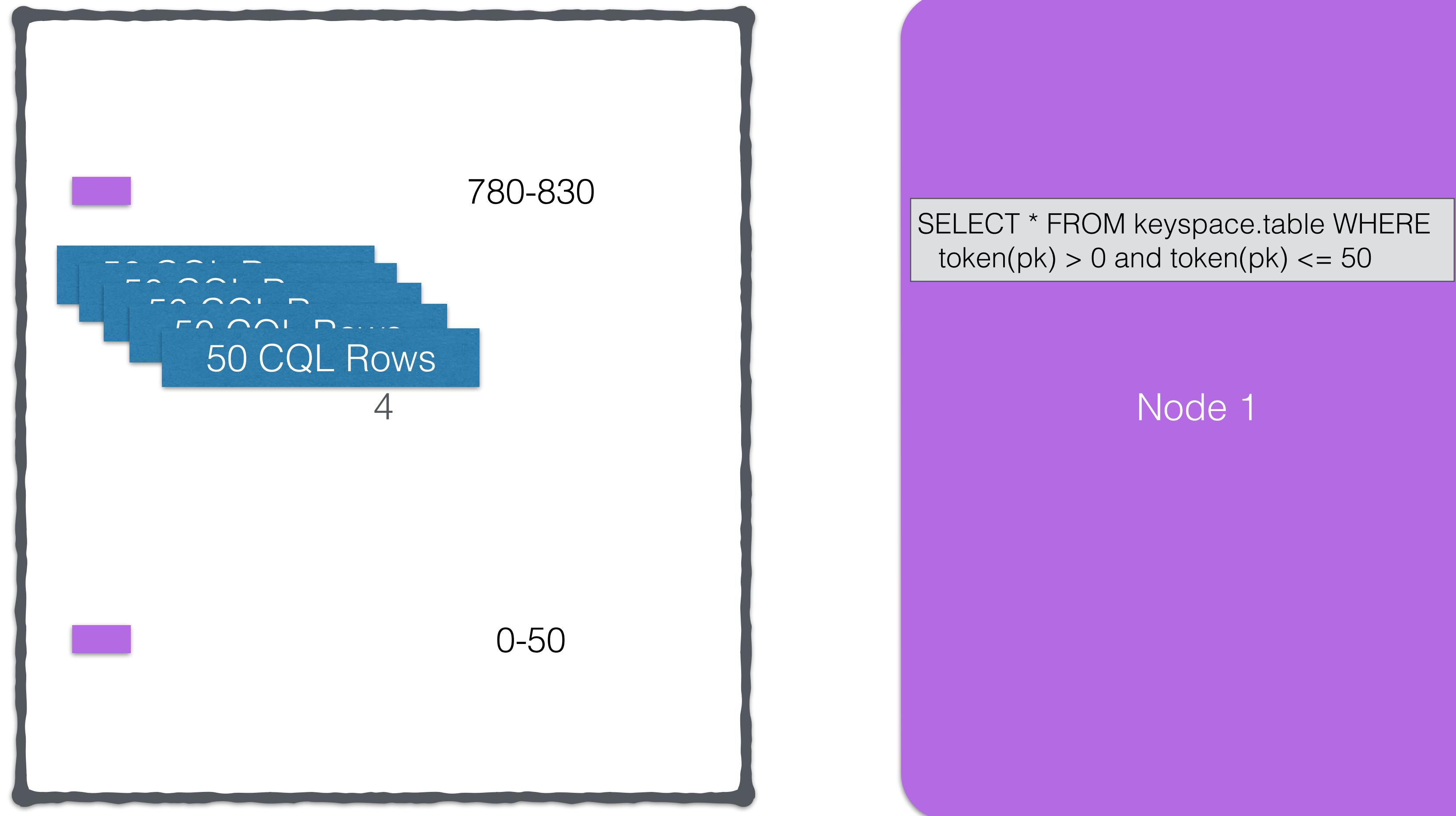
Data is Retrieved Using the DataStax Java Driver

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



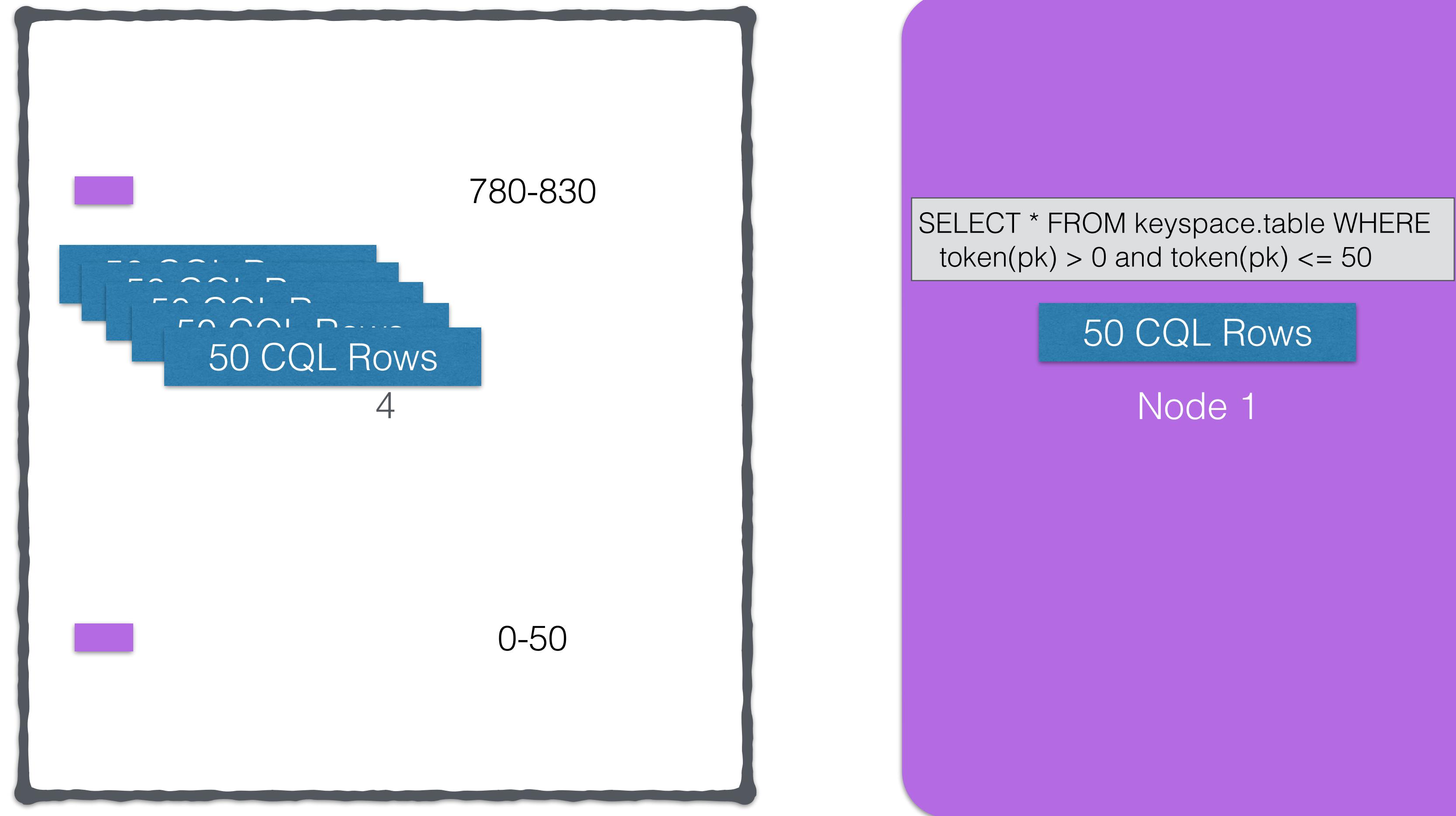
Data is Retrieved Using the DataStax Java Driver

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



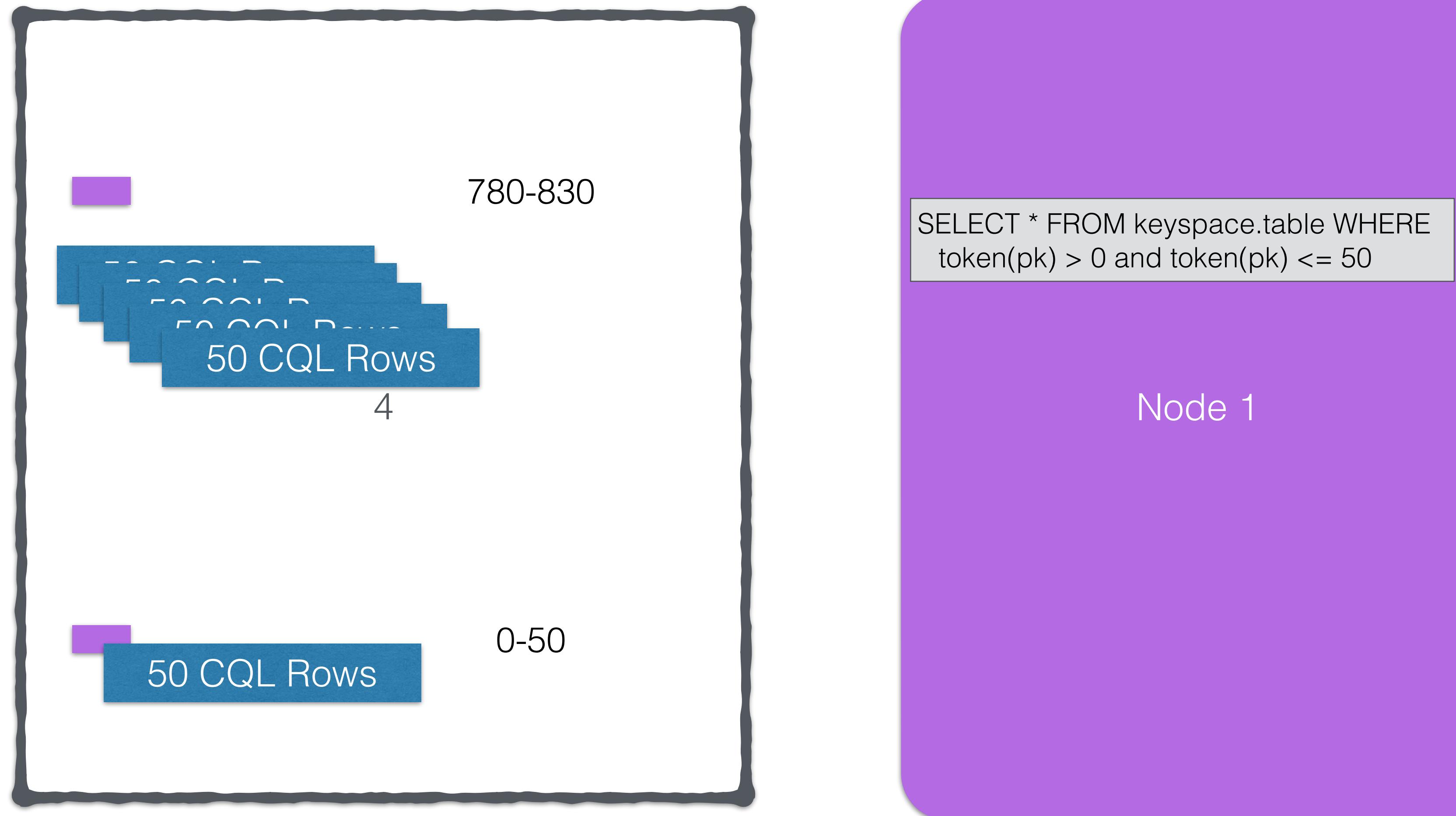
Data is Retrieved Using the DataStax Java Driver

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



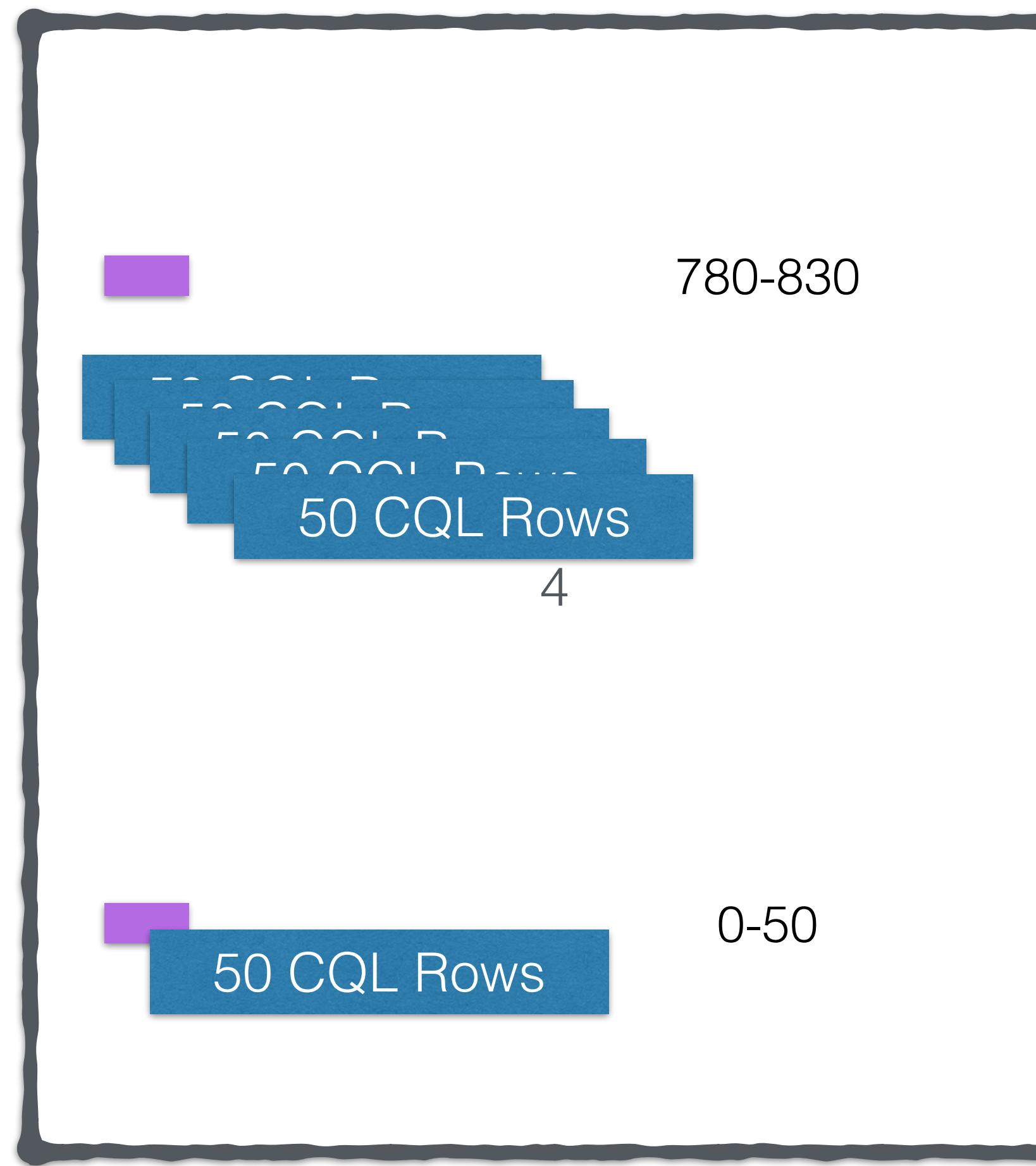
Data is Retrieved Using the DataStax Java Driver

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



Data is Retrieved Using the DataStax Java Driver

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

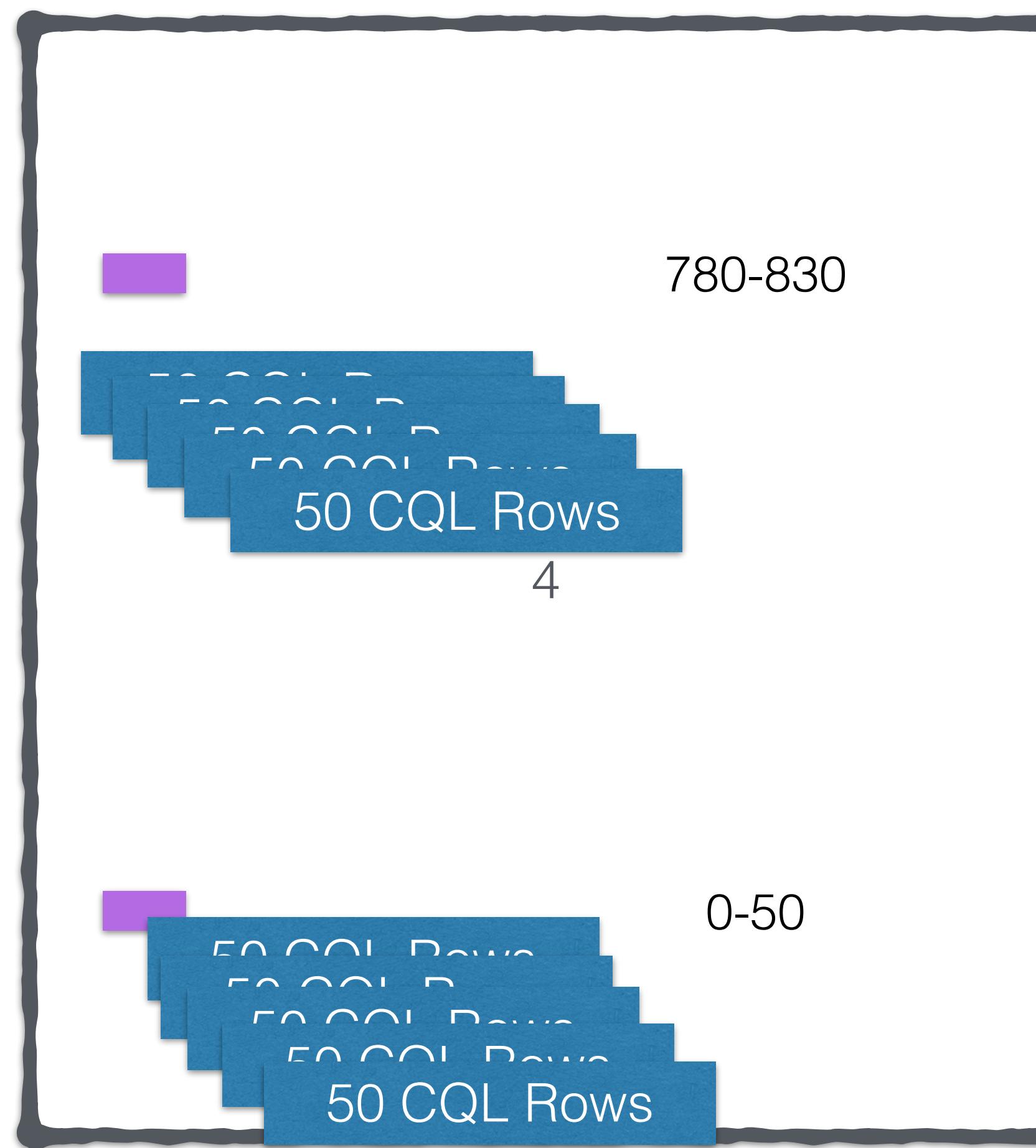


```
SELECT * FROM keyspace.table WHERE  
token(pk) > 0 and token(pk) <= 50
```



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



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

Type mapping

CQL Type	Scala Type
<i>ascii</i>	<i>String</i>
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



```
// 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

```
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

```
/** 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()

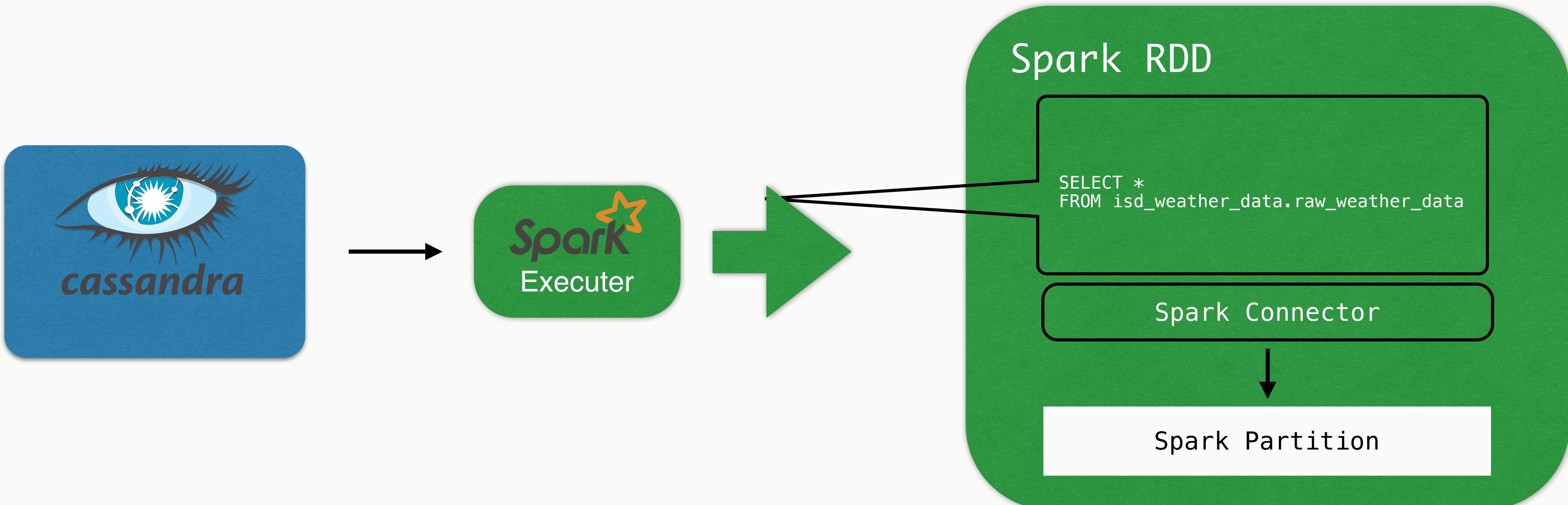
println(s"Total Rows in Raw Weather Table: $rowCount")
sc.stop()
```

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

```
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

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!

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

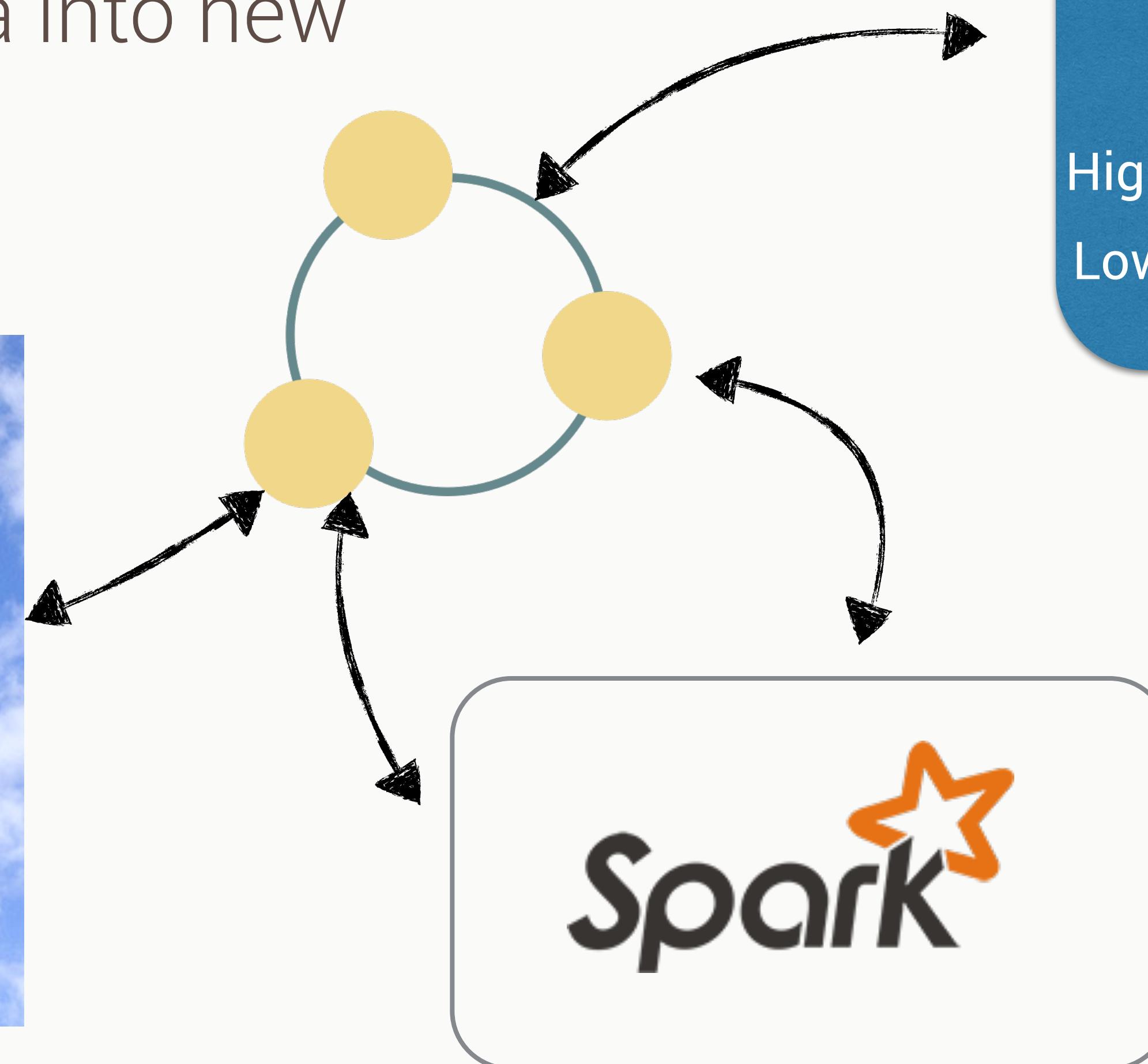
conf = SparkConf() \
    .setAppName("Weather App") \
    .setMaster("spark://127.0.0.1:7077") \
    .set("spark.cassandra.connection.host", "127.0.0.1")

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

- 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)

```
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

aggregations

```
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

Table:

temperature

- Data is read from **temperature** table
- Transformed
- Inserted into the **daily_high_low** table

Table:

daily_high_low

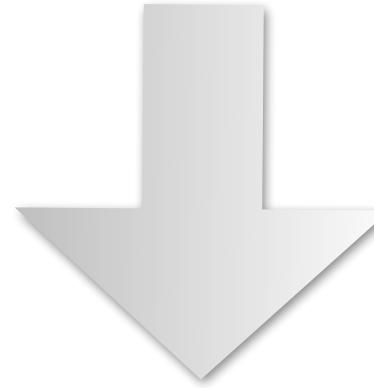


Spark Streaming

Spark Versus Spark Streaming



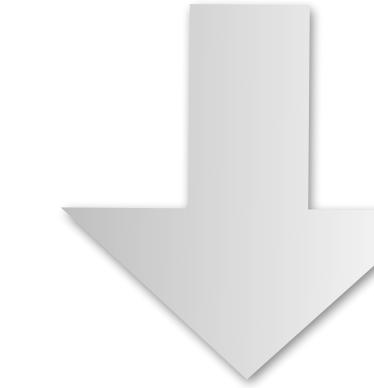
zillions of bytes



Spark

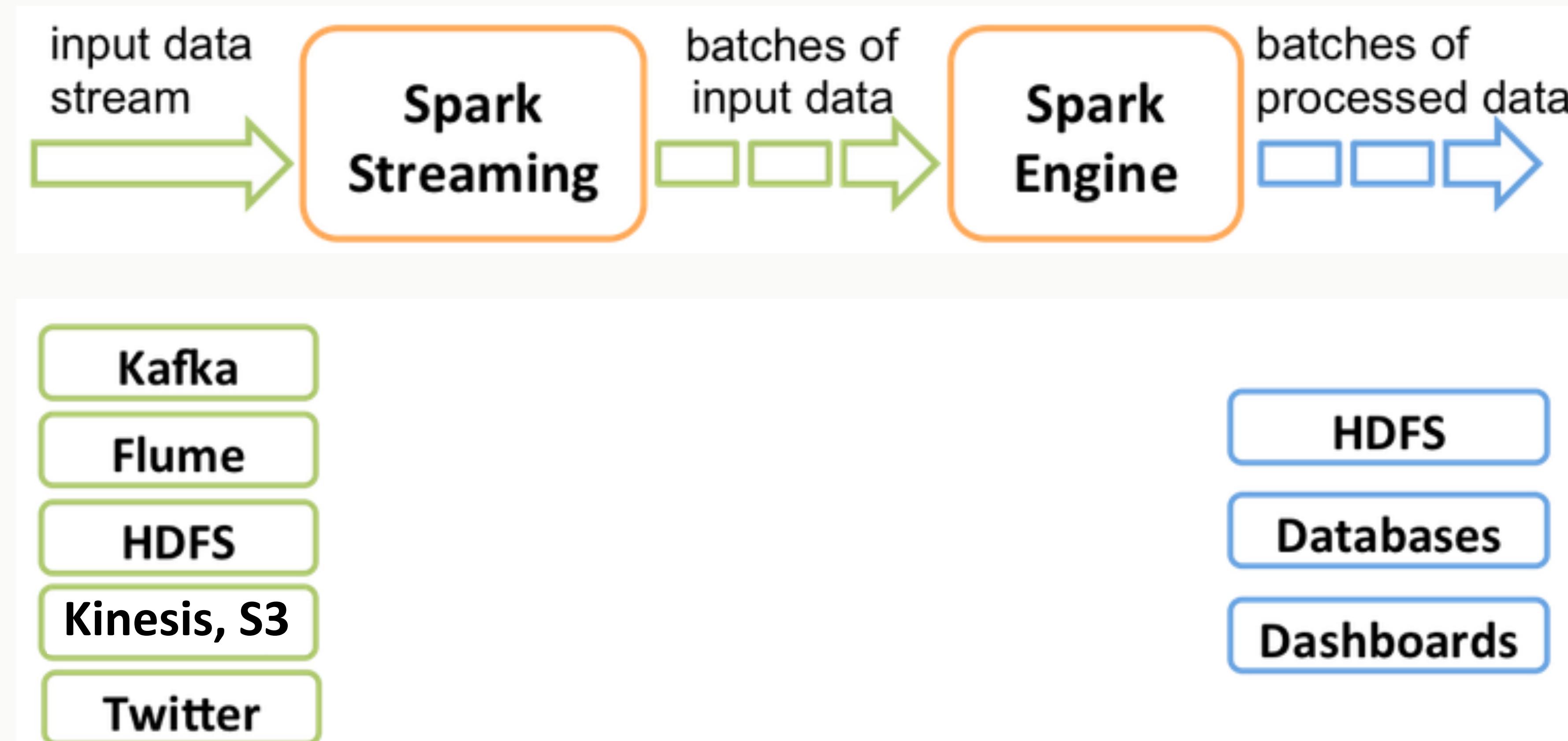


gigabytes per second



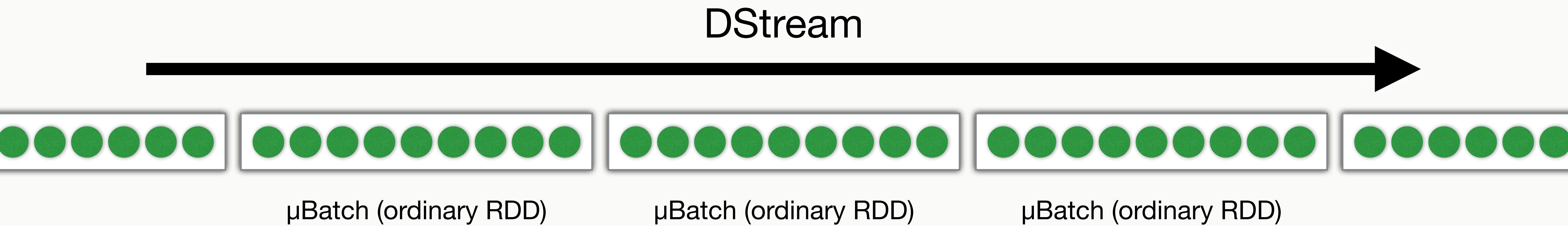
Spark
Streaming

Spark Streaming



DStream - Micro Batches

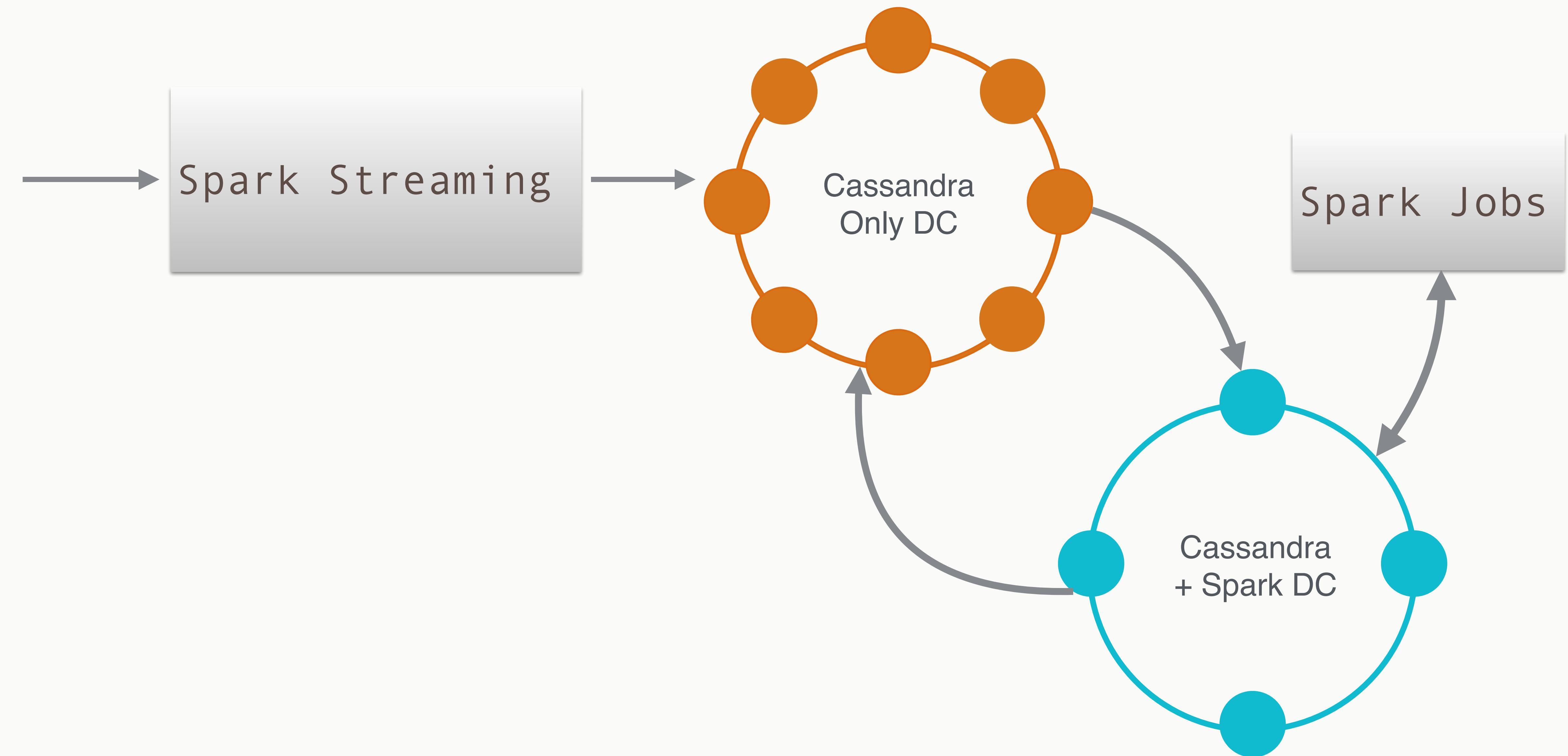
- Continuous sequence of micro batches
 - More complex processing models are possible with less effort
 - Streaming computations as a series of deterministic batch computations on small time intervals



Processing of DStream = Processing of μBatches, RDDs



Now what?



You can do this at home!

killrweather / killrweather

Unwatch · 14

KillrWeather is a reference application (in progress) showing how to easily leverage and integrate Apache Spark, Apache Cassandra, and Apache Kafka for fast, streaming computations on time series data in asynchronous Akka event-driven environments. — Edit

83 commits · 2 branches · 0 releases · 3 contributors

branch: master · killrweather / +

Minor cleanup and documentation added.

helena authored 2 days ago · latest commit 849a1738f7

File	Description	Time Ago
data	Adding much bigger data file	5 days ago
killrweather-app/src	Minor cleanup and documentation added.	2 days ago
killrweather-core/src/main	New client for kafka and query req resp reporting added.	5 days ago
project	Plugin cleanup.	5 days ago
sigar	Added Sigar for test metrics collection via Akka Cluster, handled IT ...	a month ago
.gitignore	Started refactor of Kafka stream design.	22 days ago
LICENSE	Initial commit	a month ago
README.md	Update README.md	5 days ago
version.sbt	Initial commit of project build.	a month ago

README.md

<https://github.com/killrweather/killrweather>

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