Spark/Cassandra Integration Theory & Practice

DuyHai DOAN, Technical Advocate

@doanduyhai

Who Am I ?

Duy Hai DOAN

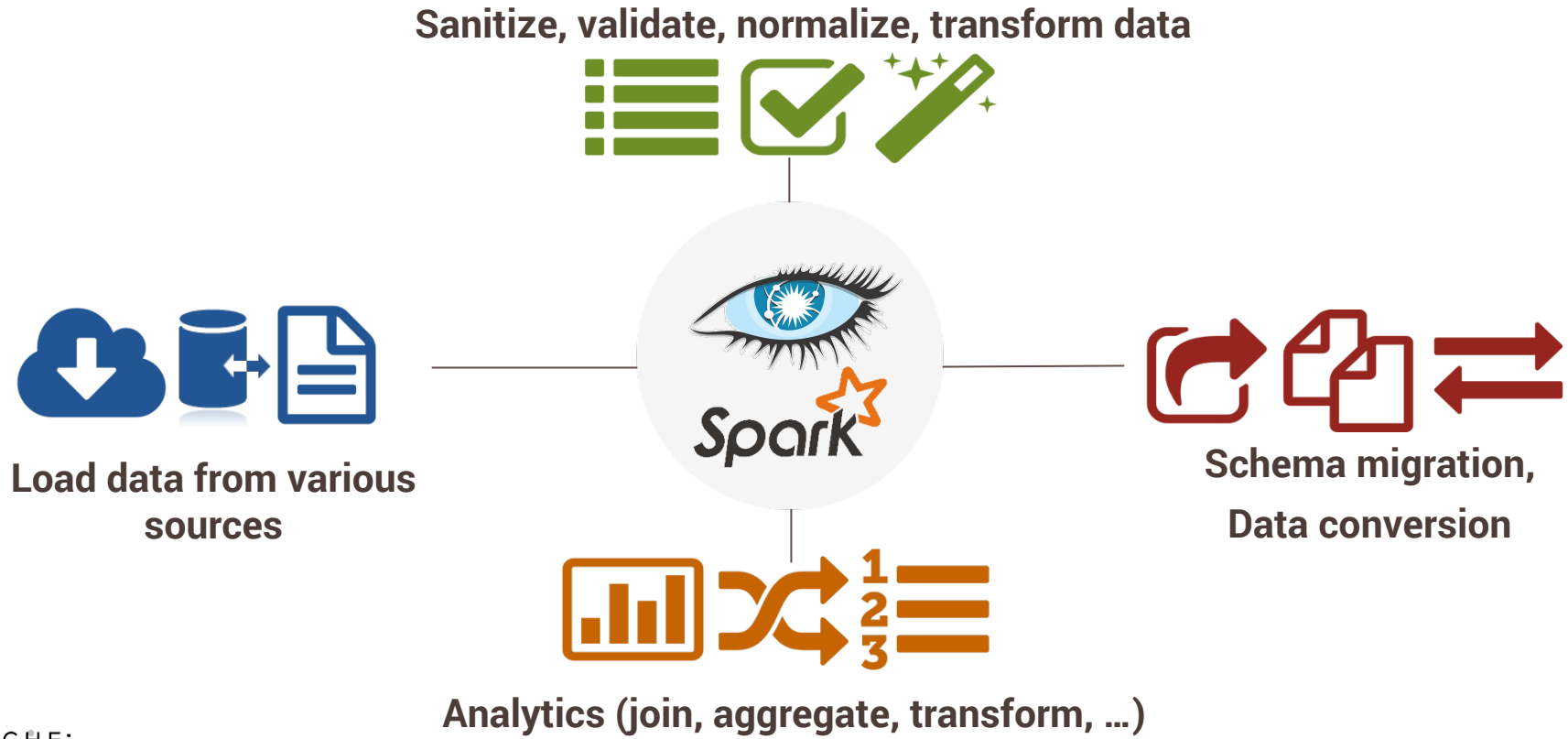
Cassandra technical advocate

- talks, meetups, confs
- open-source devs (**Achilles**, ...)
- OSS Cassandra point of contact
  - ☞ **duy_hai.doan@datastax.com**
  - ☞ **@doanduyhai**

Datastax

- Founded in April 2010
- We contribute a lot to Apache Cassandra™
- 400+ customers (25 of the Fortune 100), 400+ employees
- Headquarter in San Francisco Bay area
- EU headquarter in London, offices in France and Germany
- Datastax Enterprise = OSS Cassandra + extra features

Spark – Cassandra Use Cases



Spark & Cassandra Presentation

Spark & its eco-system

Cassandra Quick Recap

What is Apache Spark ?

Created at 

Apache Project since 2010

General data processing framework

Faster than Hadoop, in memory

One-framework-many-components approach

Spark code example

Setup

```
val conf = new SparkConf(true)
        .setAppName("basic_example")
        .setMaster("local[3]")

val sc = new SparkContext(conf)
```

Data-set (can be from text, CSV, JSON, Cassandra, HDFS, …)

```
val people = List(("jdoe","John DOE", 33),
                  ("hsue","Helen SUE", 24),
                  ("rsmith", "Richard Smith", 33))
```

RDDs

RDD = Resilient Distributed Dataset

```
val parallelPeople: RDD[(String, String, Int)] = sc.parallelize(people)

val extractAge: RDD[(Int, (String, String, Int))] = parallelPeople
                                                    .map(tuple => (tuple._3, tuple))

val groupByAge: RDD[(Int, Iterable[(String, String, Int)])]=extractAge.groupByKey()

val countByAge: Map[Int, Long] = groupByAge.countByKey()
```

RDDs

RDD[A] = distributed collection of A

- RDD[Person]
- RDD[(String,Int)], …

RDD[A] split into partitions

Partitions distributed over *n* workers → parallel computing

Direct transformations

map(f: A => B): RDD[B]

filter(f: A => Boolean): RDD[A]

...

Transformations requiring shuffle

groupByKey(): RDD[(K,V)]

reduceByKey(f: (V,V) => V): RDD[(K,V)]

join[W](otherRDD: RDD[(K,W)]): RDD[(K, (V,W))]

...

Actions

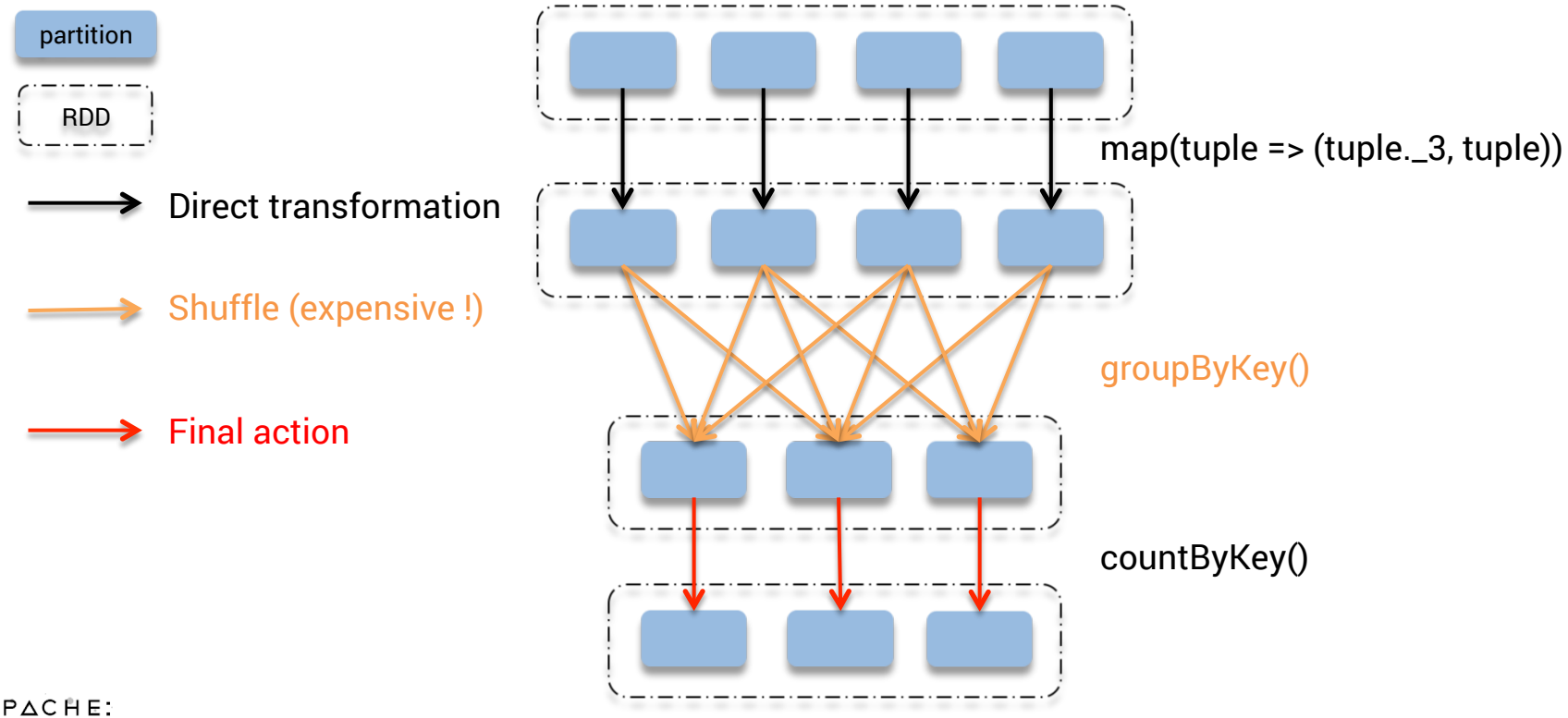
collect(): Array[A]

take(number: Int): Array[A]

foreach(f: A => Unit): Unit

...

Partitions transformations



Spark eco-system

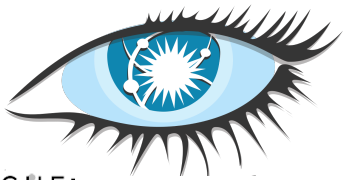
Spark Streaming | Spark SQL | GraphX | MLLib | …

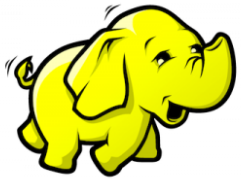
Spark Core Engine (Scala/Java/Python)

Cluster Manager

Local | Standalone cluster | YARN | Mesos

Persistence





etc…

Spark eco-system

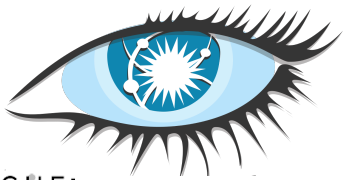
Spark Streaming | Spark SQL | GraphX | MLLib | ...

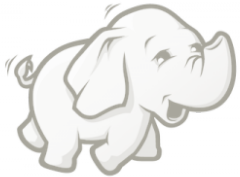
Spark Core Engine (Scala/Java/Python)

Cluster Manager

Local | Standalone cluster | YARN | Mesos

Persistence





etc...

What is Apache Cassandra?

Created at 

Apache Project since 2009

Distributed NoSQL database

Eventual consistency (A & P of the CAP theorem)

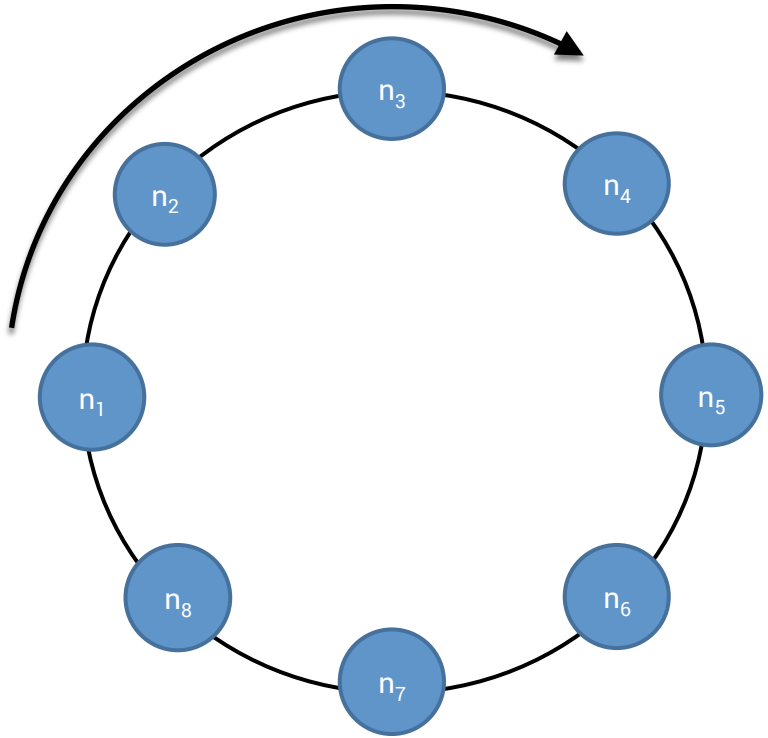
Distributed table abstraction

Cassandra data distribution reminder

Random: hash of #partition → **token** = hash(#p)

Hash:]-X, X]

X = huge number ($2^{64}/2$)



Cassandra token ranges

A:]0, X/8]

B:] X/8, 2X/8]

C:] 2X/8, 3X/8]

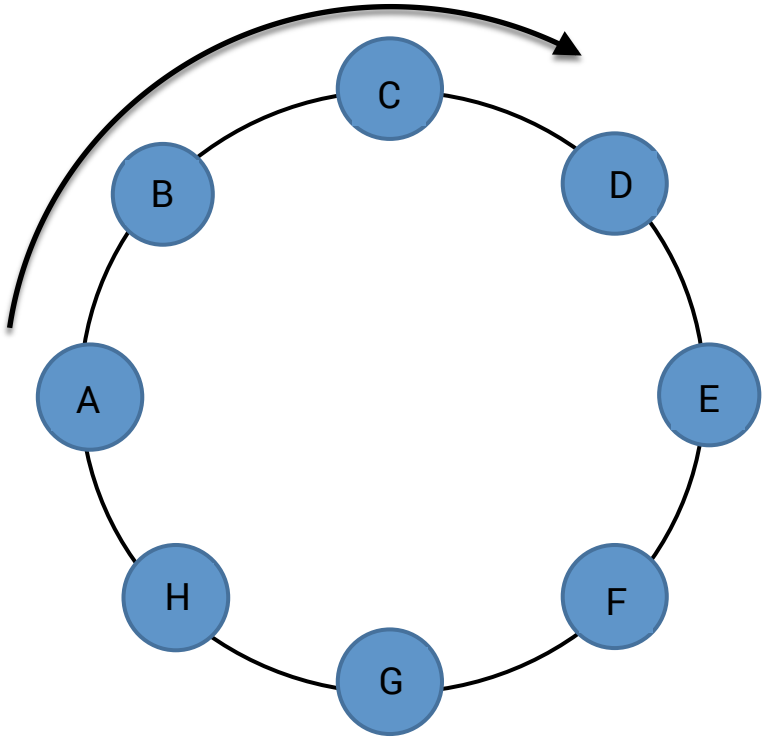
D:] 3X/8, 4X/8]

E:] 4X/8, 5X/8]

F:] 5X/8, 6X/8]

G:] 6X/8, 7X/8]

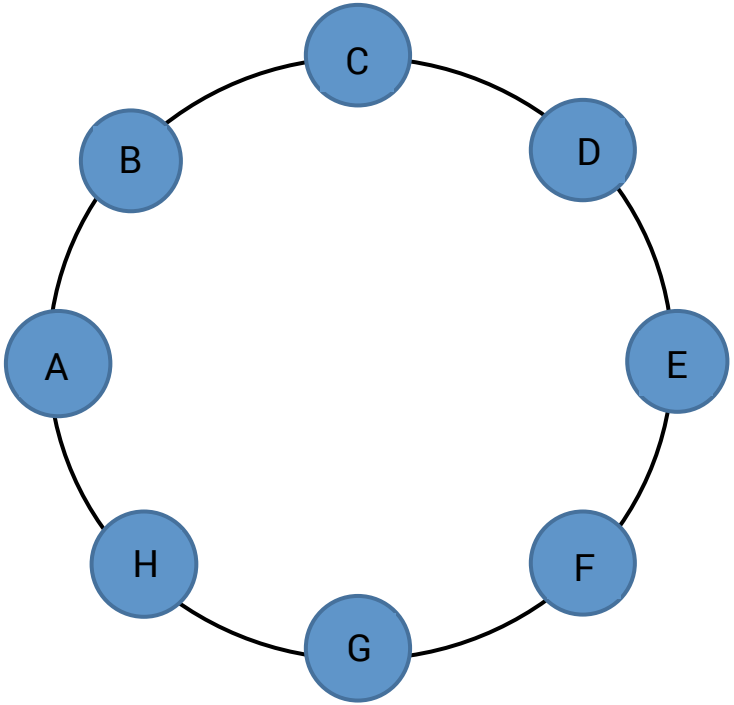
H:] 7X/8, X]



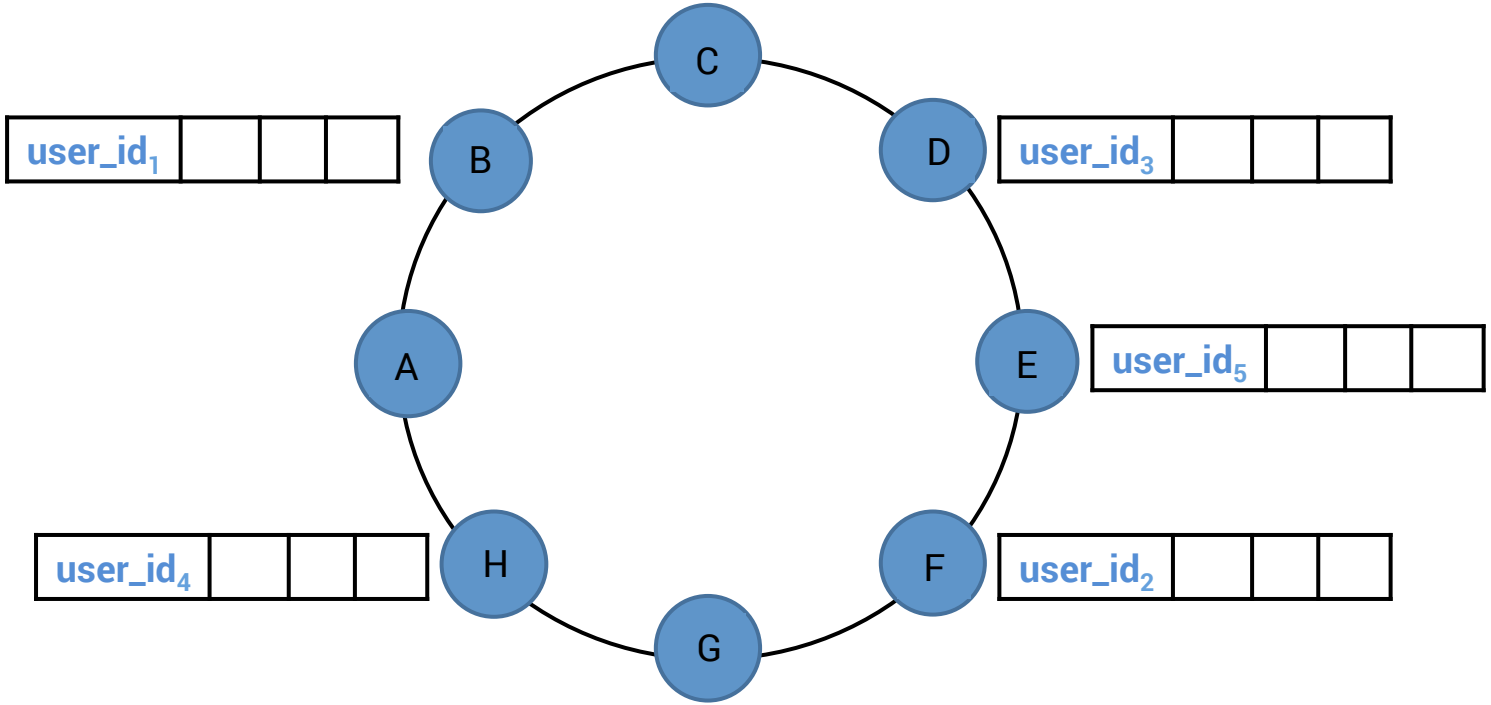
Murmur3 hash function

Linear scalability

$user\_id_1$			
$user\_id_2$			
$user\_id_3$			
$user\_id_4$			
$user\_id_5$			



Linear scalability



Cassandra Query Language (CQL)

```
INSERT INTO users(login, name, age) VALUES('jdoe', 'John DOE', 33);
```

```
UPDATE users SET age = 34  WHERE login = 'jdoe';
```

```
DELETE age FROM users WHERE login = 'jdoe';
```

```
SELECT age FROM users WHERE login = 'jdoe';
```

Spark & Cassandra Connector

Spark Core API

SparkSQL/DataFrame

Spark Streaming

Spark/Cassandra connector architecture

All Cassandra types supported and converted to Scala types

Server side data filtering (SELECT … WHERE …)

Use Java-driver underneath

Scala and Java support. Python support via PySpark (exp.)

Connector architecture – Core API

Cassandra tables exposed as Spark RDDs

Read from and write to Cassandra

Mapping of C* tables and rows to Scala objects

- CassandraRDD and CassandraRow
- Scala case class (object mapper)
- Scala tuples

Spark Core

https://github.com/doanduyhai/incubator-zeppelin/tree/ApacheBigData

Connector architecture – DataFrame

Mapping of Cassandra table to DataFrame

- CassandraSQLContext → org.apache.spark.sql.SQLContext
- CassandraSQLRow → org.apache.spark.sql.catalyst.expressions.Row
- Mapping of Cassandra types to Catalyst types
- CassandraCatalog → Catalog (used by Catalyst Analyzer)

Connector architecture – DataFrame

Mapping of Cassandra table to SchemaRDD

- CassandraSourceRelation
  - extends **BaseRelation** with **InsertableRelation** with PruntedFilteredScan
- custom query plan
- push predicates to CQL for early filtering (if possible)

```
SELECT * FROM user_emails WHERE login = 'jdoe';
```

Spark SQL

https://github.com/doanduyhai/incubator-zeppelin/tree/ApacheBigData

Connector architecture – Spark Streaming

Streaming data INTO Cassandra table

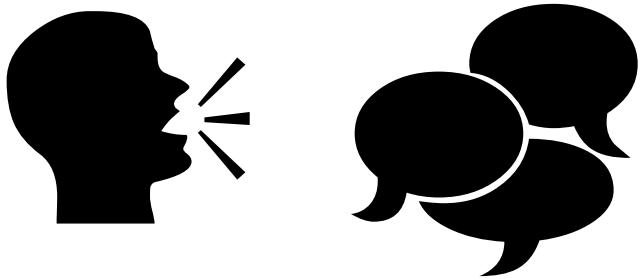
- trivial setup
- be careful about your Cassandra data model when having an infinite stream !!!

Streaming data OUT of Cassandra tables (CASSANDRA-8844) ?

- notification system (publish/subscribe)
- at-least-once delivery semantics
- work in progress ...

Spark Streaming

https://github.com/doanduyhai/incubator-zeppelin/tree/ApacheBigData



Q & R

Spark/Cassandra operations

Cluster deployment & job lifecycle

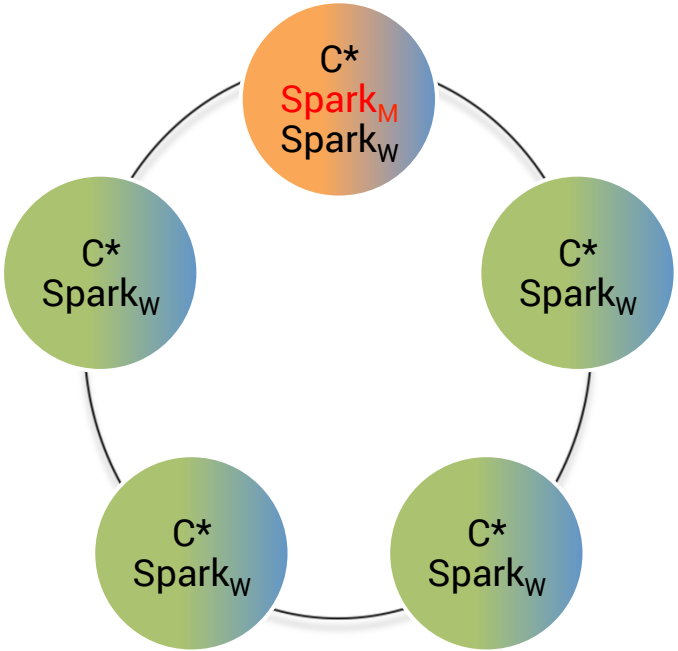
Data locality

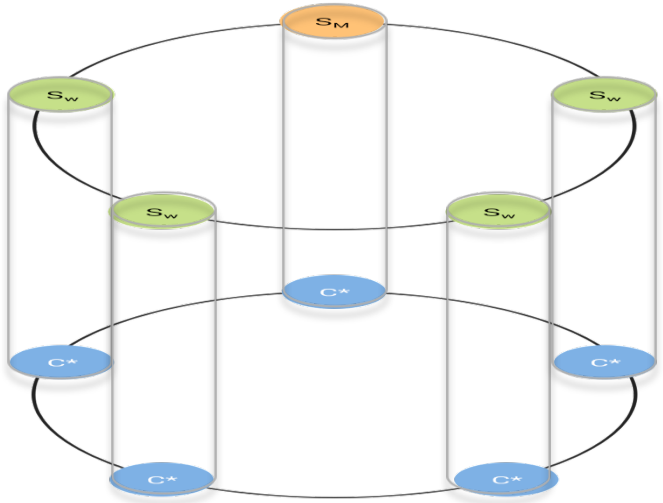
Failure handling

Cross-region operations

Cluster deployment

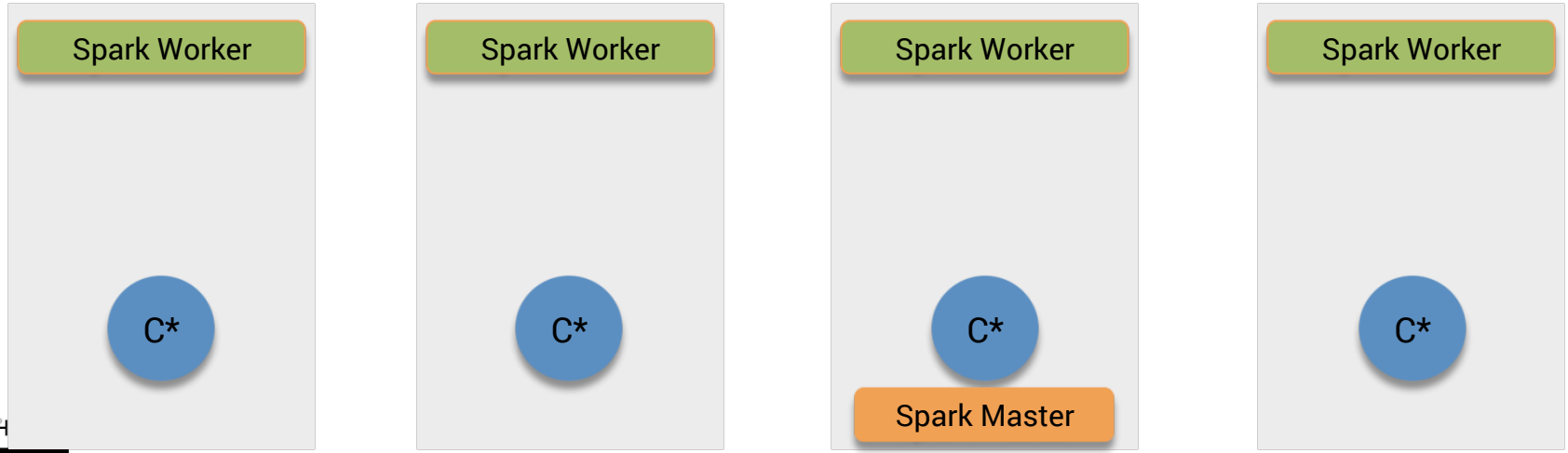
Stand-alone cluster



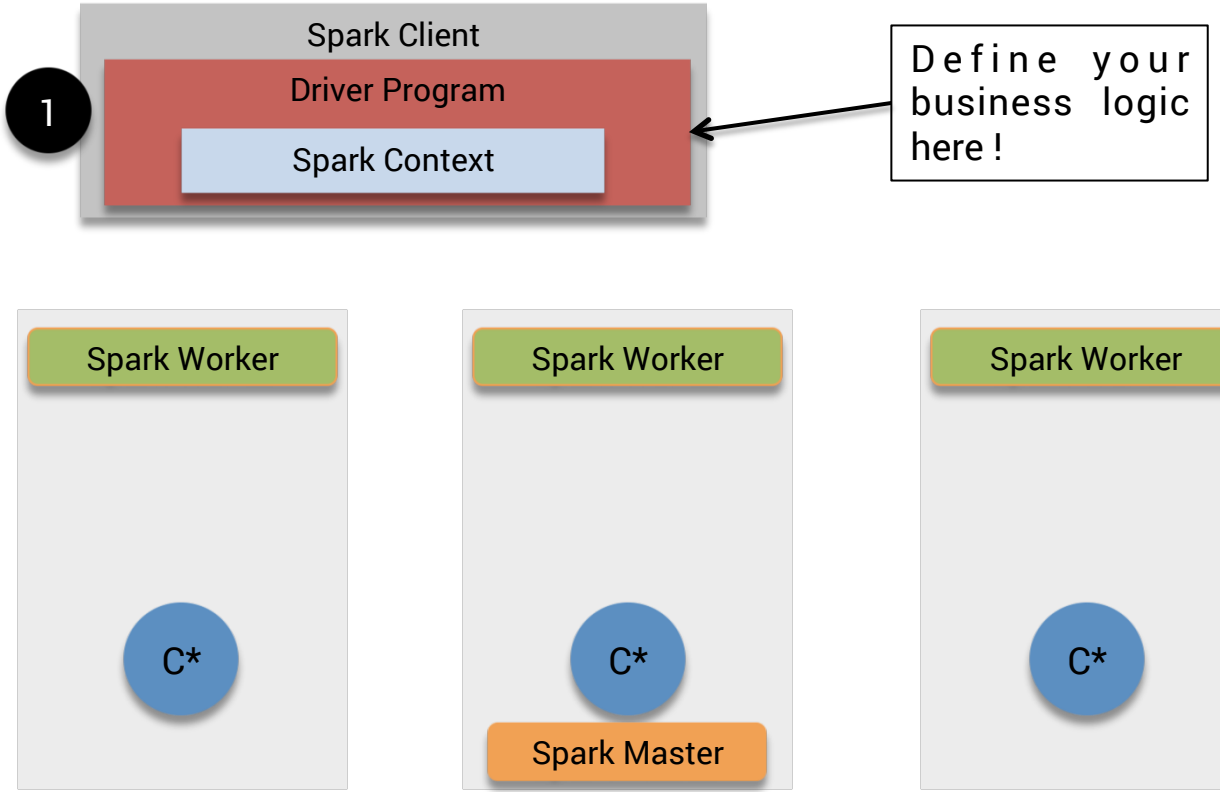


Cassandra – Spark placement

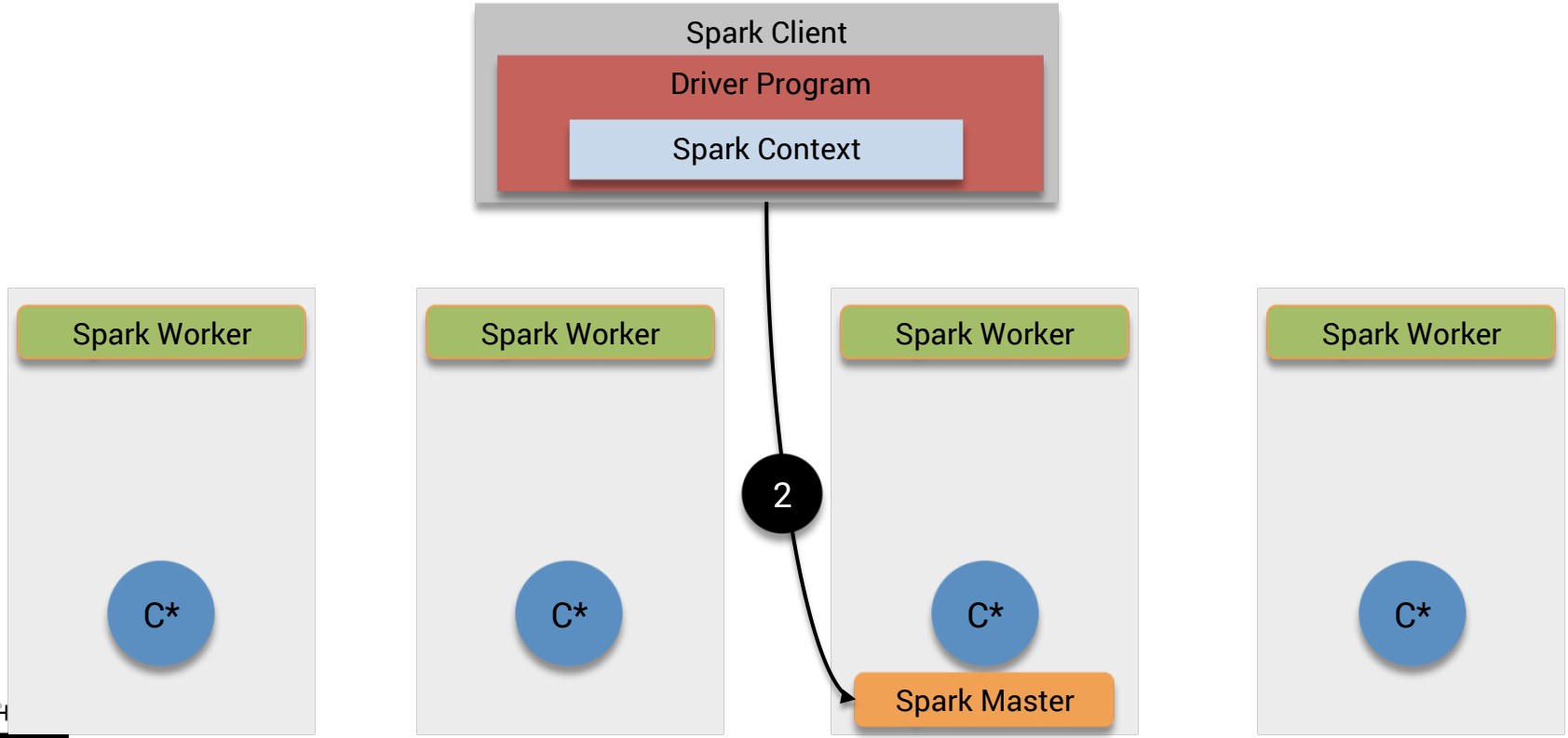
1 Cassandra process ⟷ 1 Spark worker



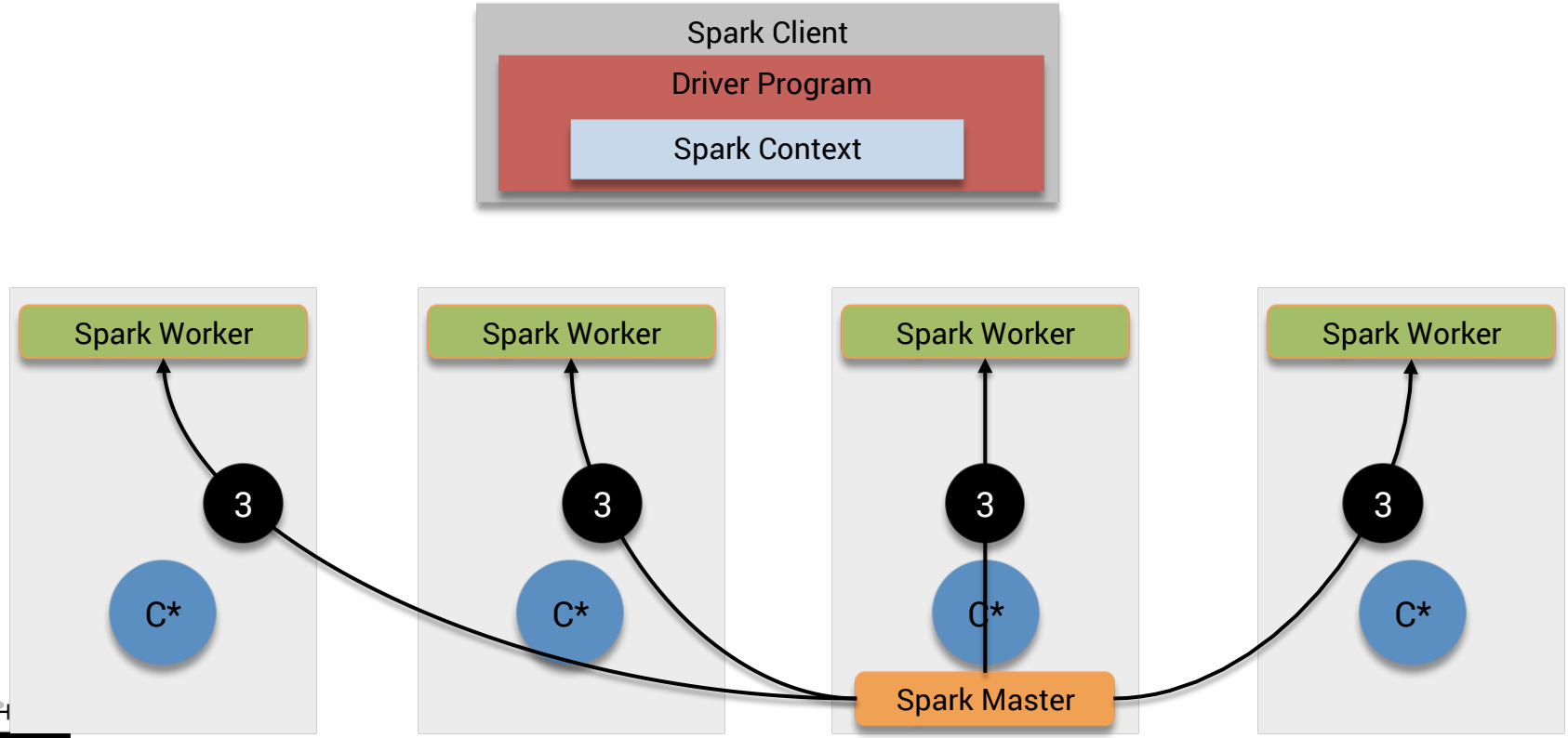
Cassandra – Spark job lifecycle



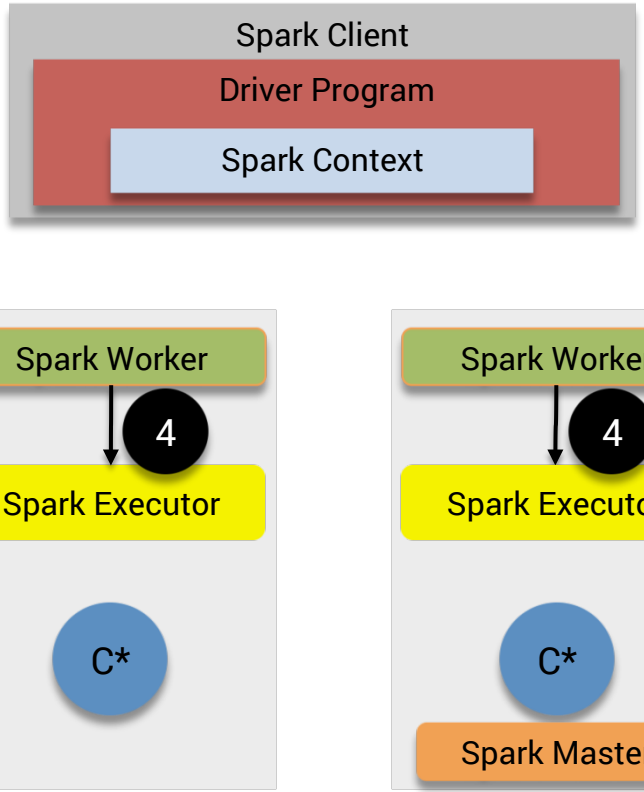
Cassandra – Spark job lifecycle



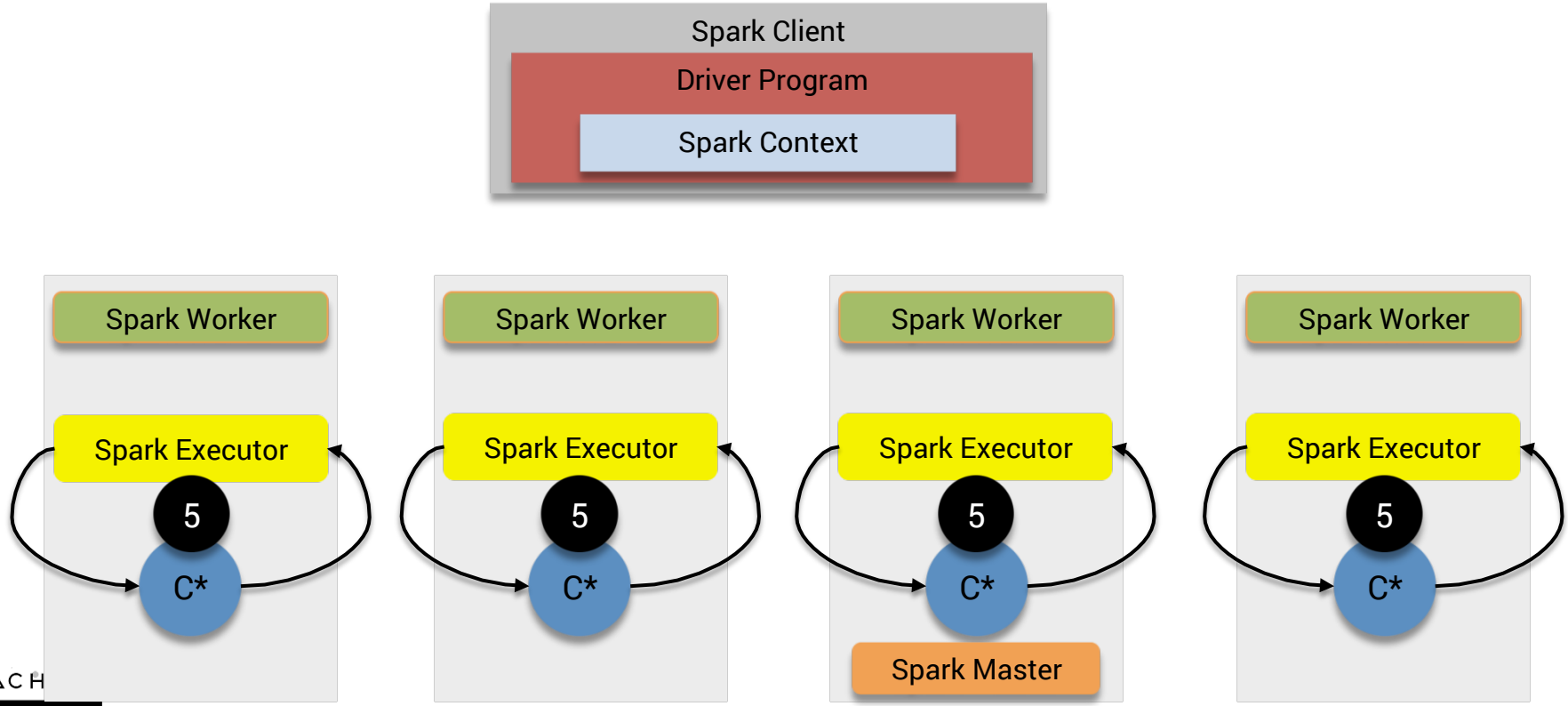
Cassandra – Spark job lifecycle



Cassandra – Spark job lifecycle



Cassandra – Spark job lifecycle



Data Locality – Cassandra token ranges

A:]0, X/8]

B:] X/8, 2X/8]

C:] 2X/8, 3X/8]

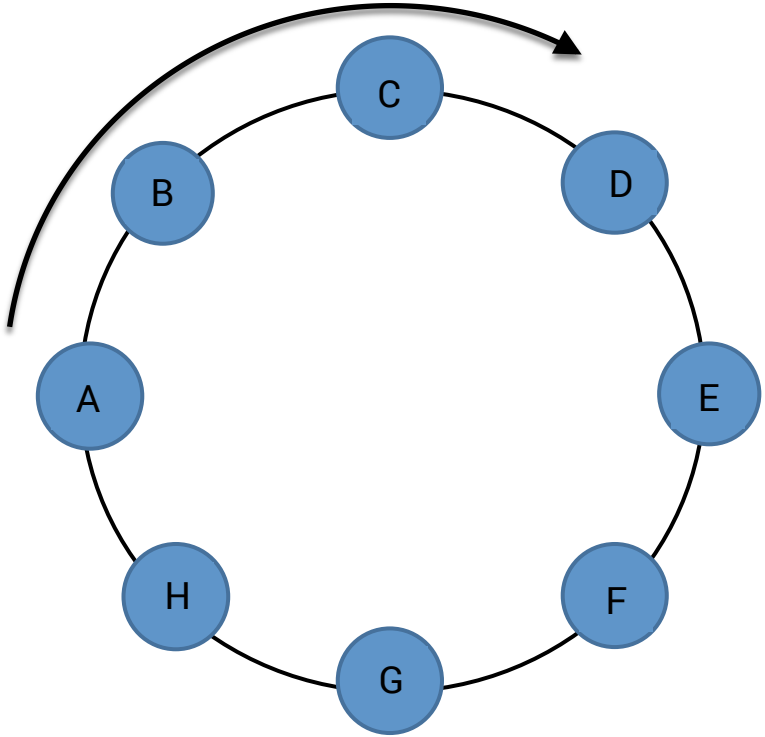
D:] 3X/8, 4X/8]

E:] 4X/8, 5X/8]

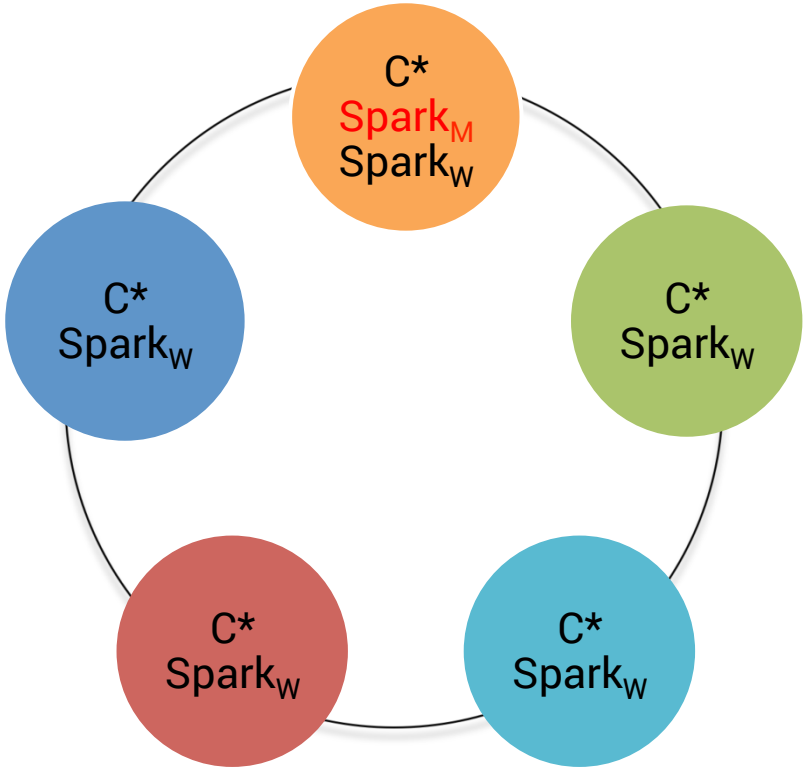
F:] 5X/8, 6X/8]

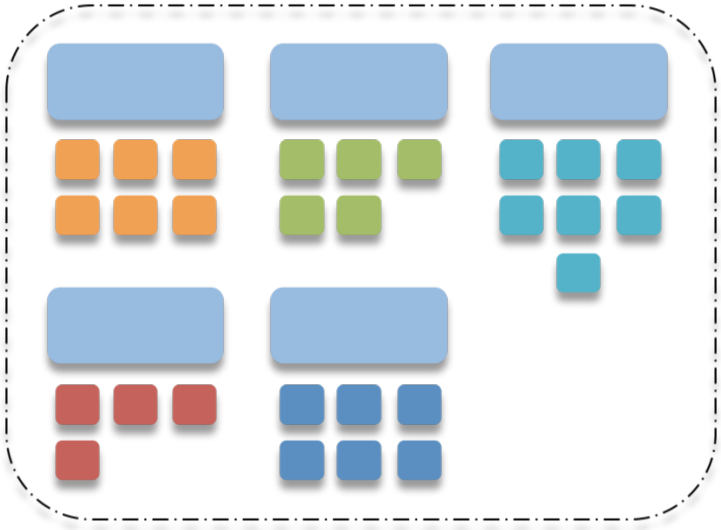
G:] 6X/8, 7X/8]

H:] 7X/8, X]



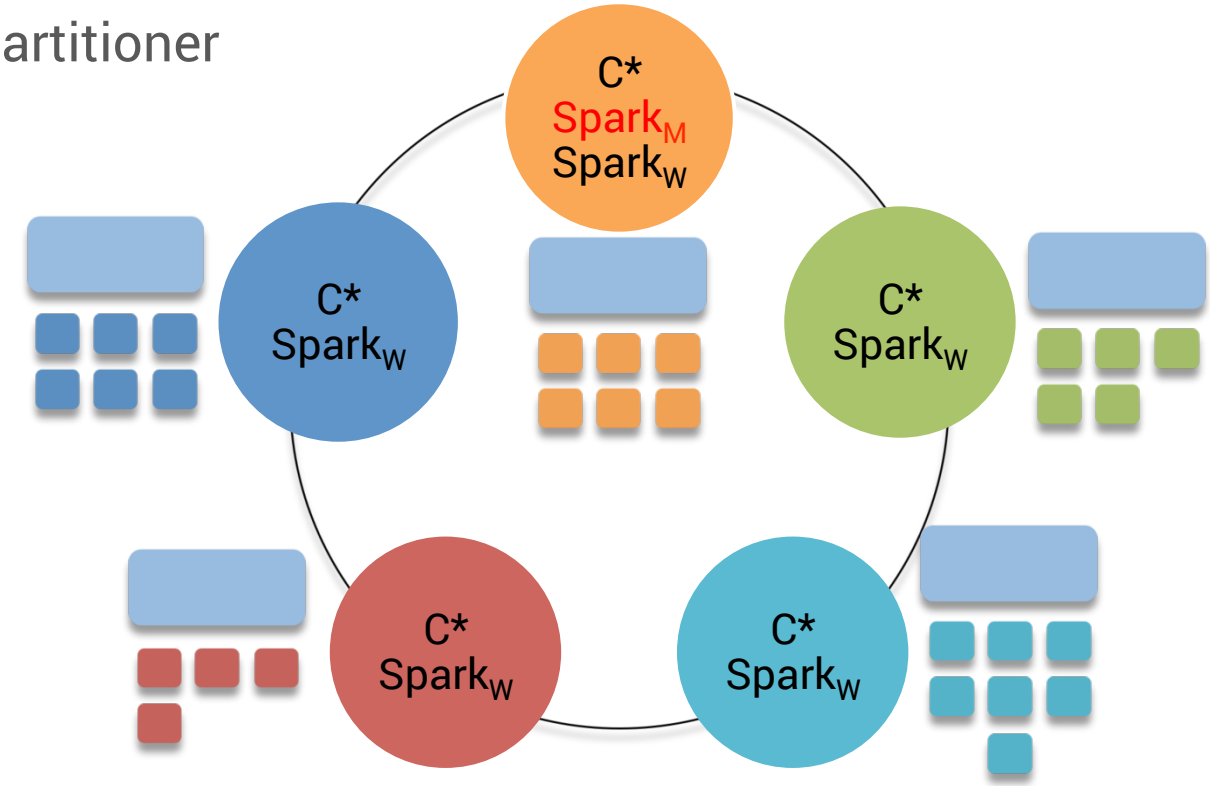
Data Locality – How To





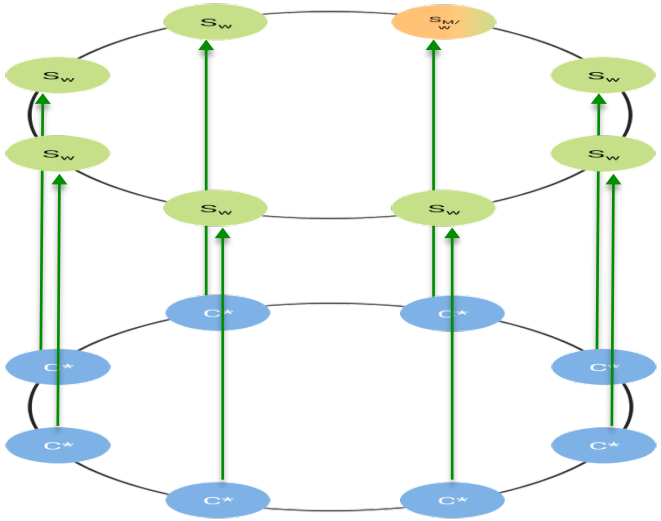
Data Locality – How To

Use Murmur3Partitioner



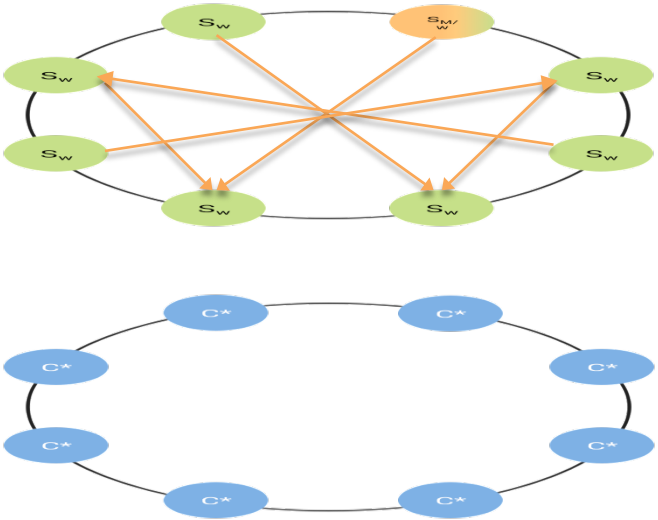
Read data locality

Read from Cassandra



Read data locality

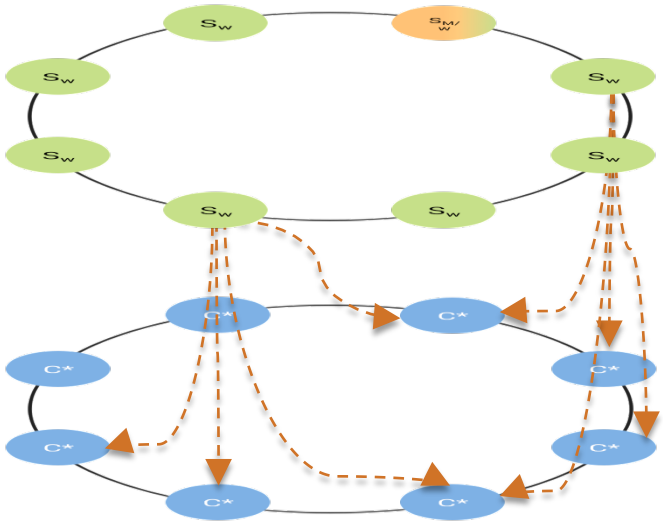
Spark shuffle operations



Write to Cassandra without data locality

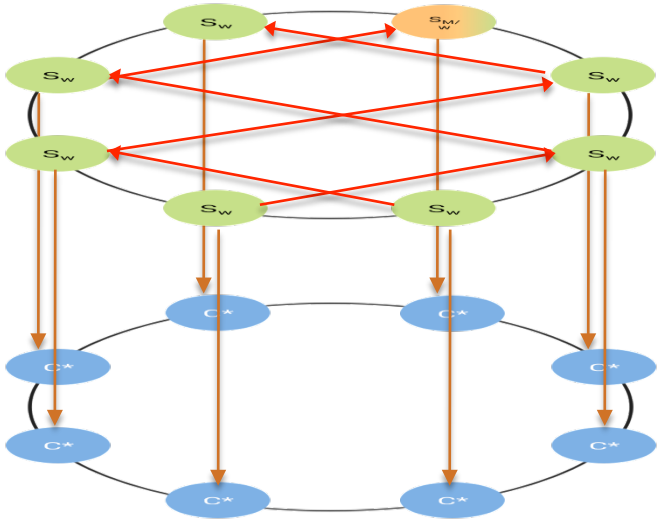
Because of shuffle, original data locality is lost

Async batches fan-out writes to Cassandra



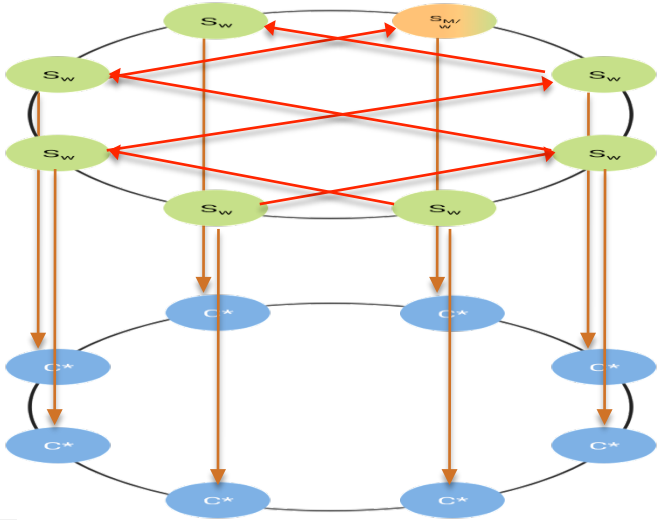
Write to Cassandra with data locality

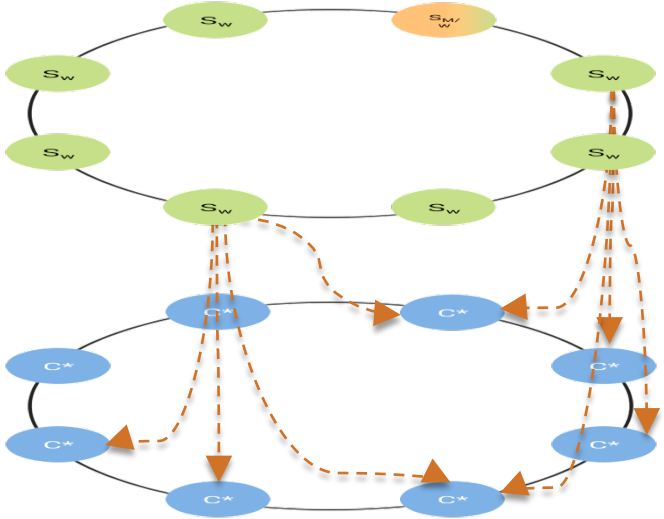
- rdd.repartitionByCassandraReplica("keyspace","table")
- Write to Cassandra



Write data locality

- either stream data in Spark layer using *repartitionByCassandraReplica()*
- or flush data to Cassandra by async batches
- in any case, **there will be data movement on network** (sorry no magic)



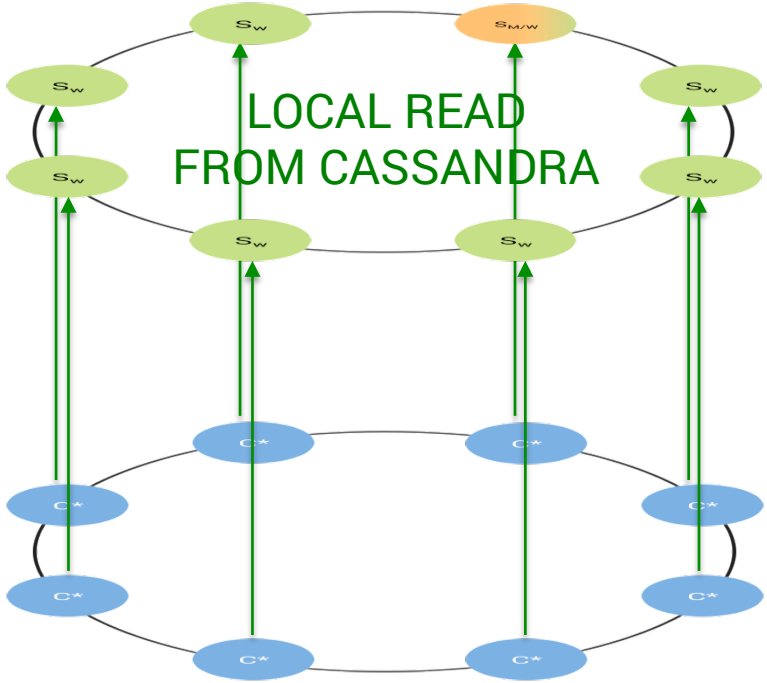


Joins with data locality

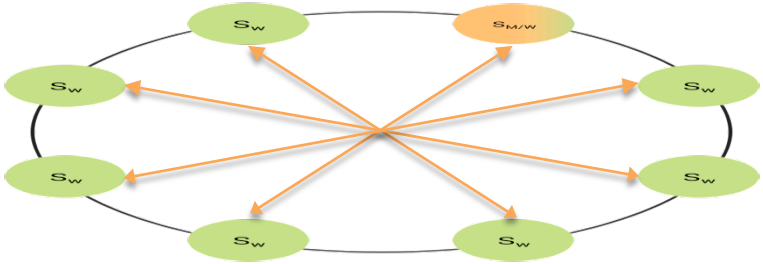
```
CREATE TABLE artists(name text, style text, … PRIMARY KEY(name));

CREATE TABLE albums(title text, artist text, year int,… PRIMARY KEY(title));
```

```
val join: CassandraJoinRDD[(String,Int), (String,String)] =
 sc.cassandraTable[(String,Int)](KEYSPACE, ALBUMS)
  // Select only useful columns for join and processing
  .select("artist","year")
  .as((_:String, _:Int))
  // Repartition RDDs by "artists" PK, which is "name"
  .repartitionByCassandraReplica(KEYSPACE, ARTISTS)
  // Join with "artists" table, selecting only "name" and "country" columns
  .joinWithCassandraTable[(String,String)](KEYSPACE, ARTISTS, SomeColumns("name","country"))
    .on(SomeColumns("name"))
```

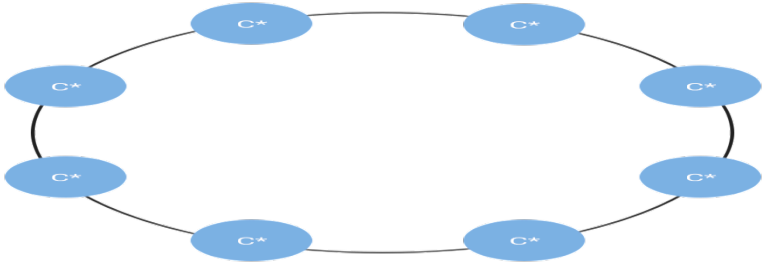
Joins pipeline with data locality



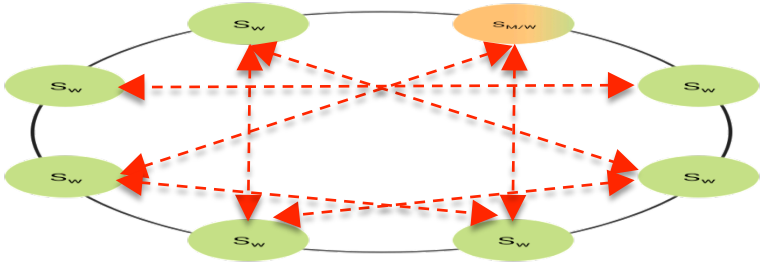
Joins pipeline with data locality



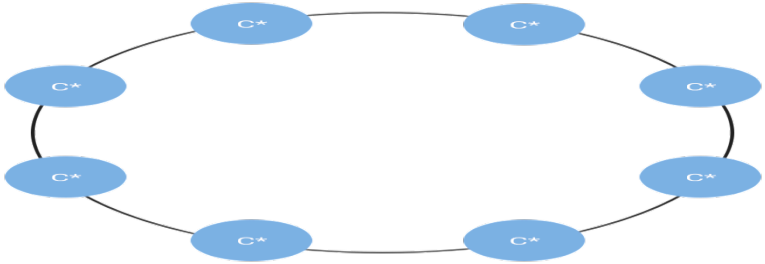
SHUFFLE DATA
WITH SPARK



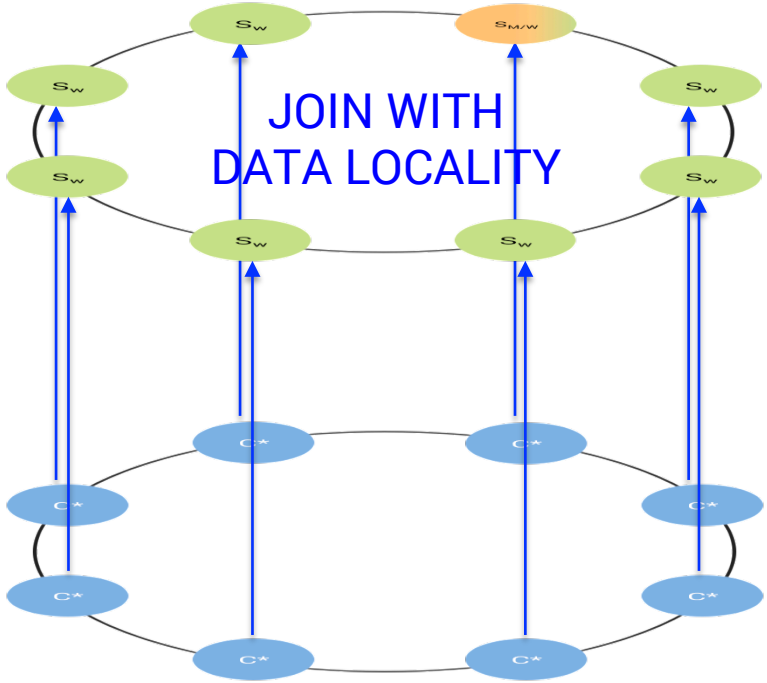
Joins pipeline with data locality



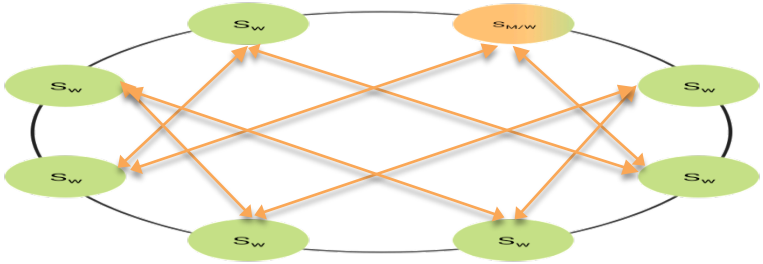
REPARTITION TO MAP
CASSANDRA REPLICA



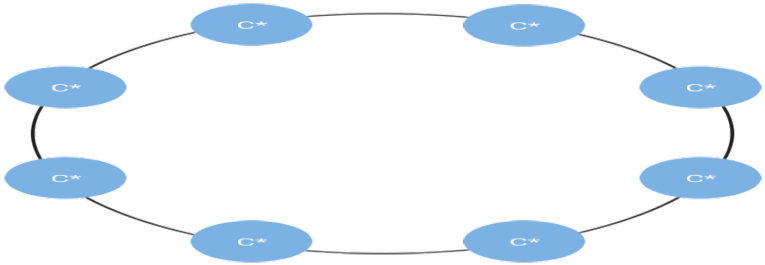
Joins pipeline with data locality



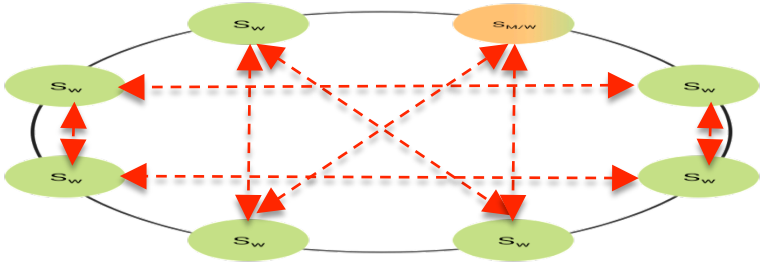
Joins pipeline with data locality



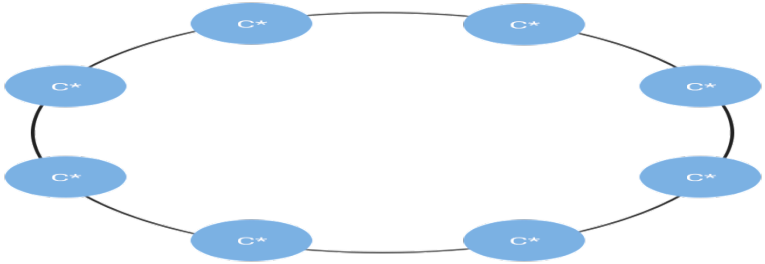
ANOTHER ROUND
OF SHUFFLING



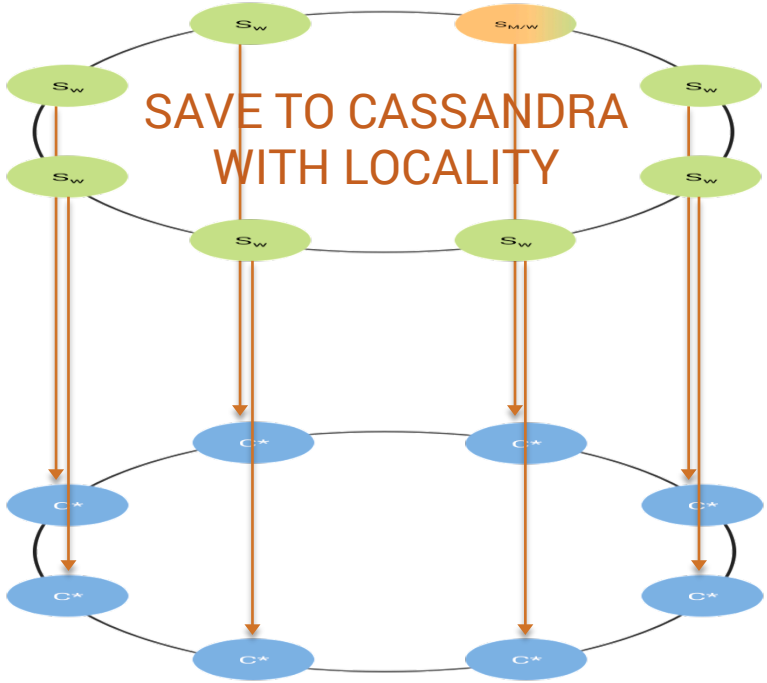
Joins pipeline with data locality



REPARTITION AGAIN
FOR CASSANDRA



Joins pipeline with data locality



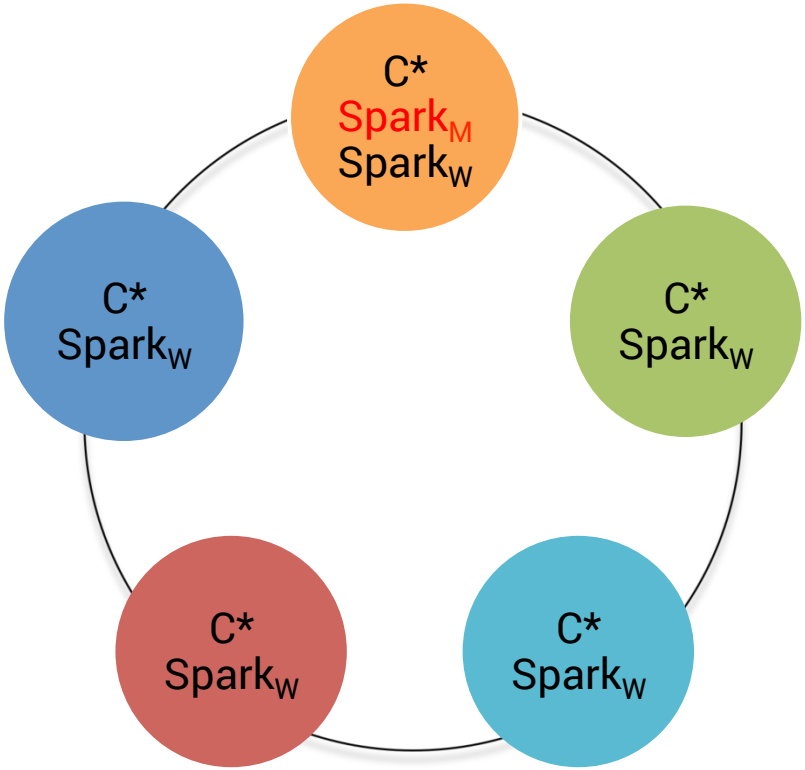
Perfect data locality scenario

- read localy from Cassandra
- use operations that do not require shuffle in Spark (map, filter, …)
- repartitionbyCassandraReplica()
- → to a table having same partition key as original table
- save back into this Cassandra table

Sanitize, validate, normalize, transform data
USE CASE

Failure handling

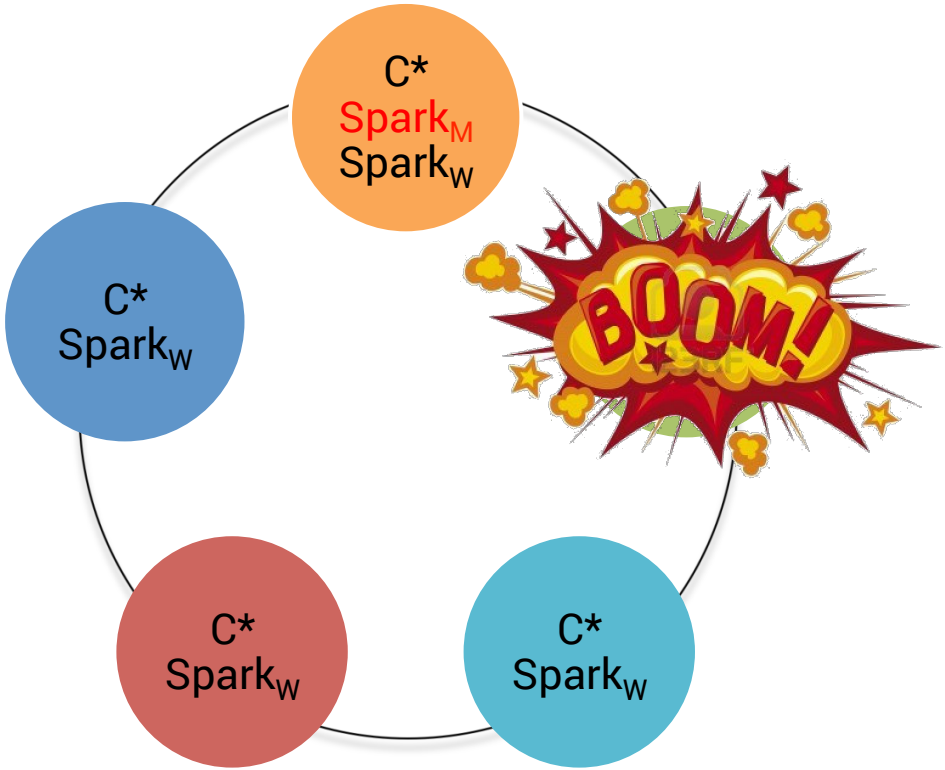
Stand-alone cluster



Failure handling

What if 1 node down ?

What if 1 node overloaded ?

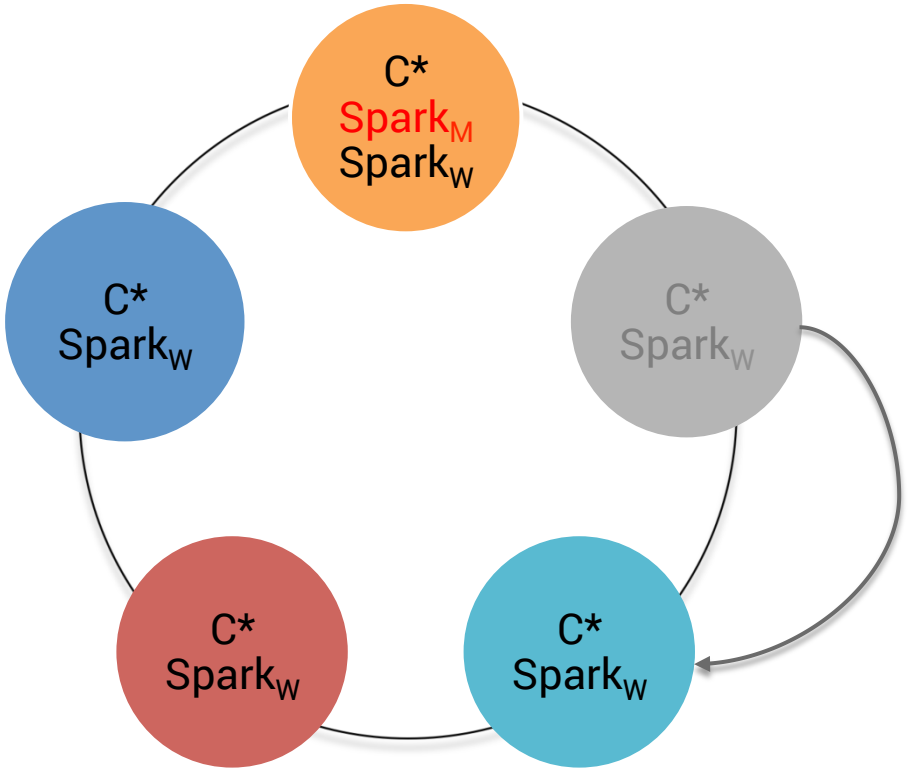


Failure handling

What if 1 node down ?

What if 1 node overloaded ?

☞ Spark master will re-assign the job to another worker



Failure handling

Oh no, my data locality !!!

Failure handling

Data Locality Impl

RDD interface (extract)

```
abstract class RDD[T](…) {
        @DeveloperApi
        def compute(split: Partition, context: TaskContext): Iterator[T]

        protected def getPartitions: Array[Partition]

        protected def getPreferredLocations(split: Partition): Seq[String] = Nil
}
```

CassandraRDD

def getPreferredLocations(split: Partition): Cassandra replicas IP address corresponding to this Spark partition

Failure handling

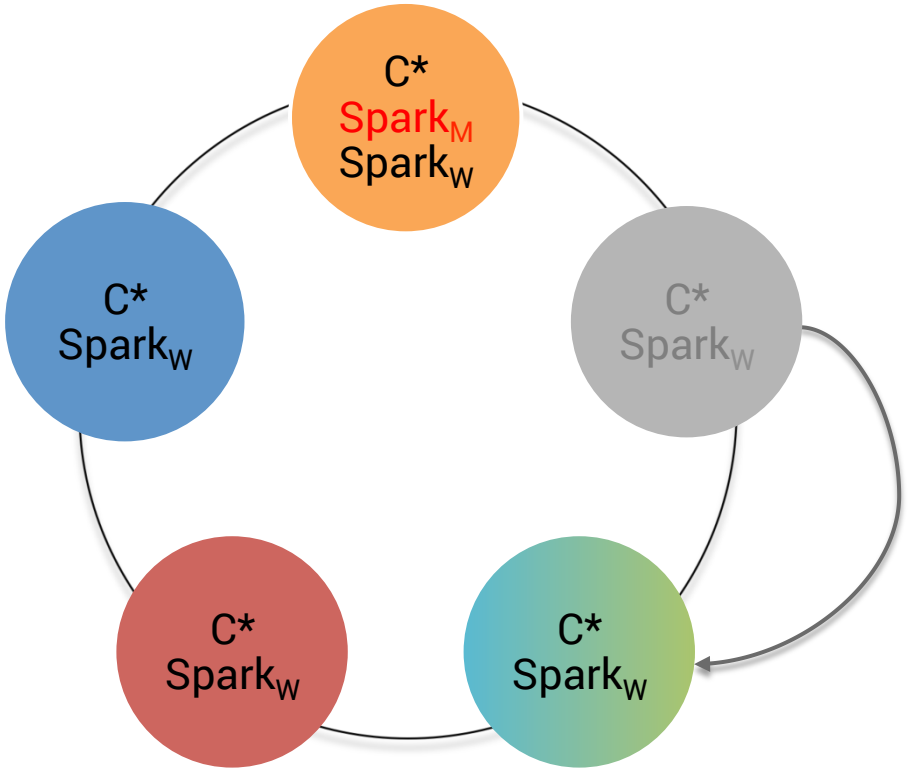
If RF > 1 the Spark master choses the next **preferred location**, which is a **replica** 😎

Tune parameters:

① *spark.locality.wait*

② *spark.locality.wait.process*

③ *spark.locality.wait.node*



Cross-DC operations

```
val confDC1 = new SparkConf(true)
        .setAppName("data_migration")
        .setMaster("master_ip")
        .set("spark.cassandra.connection.host", "DC_1_hostnames")
        .set("spark.cassandra.connection.local_dc", "DC_1")

val confDC2 = new SparkConf(true)
        .setAppName("data_migration")
        .setMaster("master_ip")
        .set("spark.cassandra.connection.host", "DC_2_hostnames")
        .set("spark.cassandra.connection.local_dc", "DC_2 ")

val sc = new SparkContext(confDC1)

sc.cassandraTable[Performer](KEYSPACE,PERFORMERS)
        .map[Performer](???)
        .saveToCassandra(KEYSPACE,PERFORMERS)
        (CassandraConnector(confDC2),implicitly[RowWriterFactory[Performer]])
```

Cross-cluster operations

```
val confCluster1 = new SparkConf(true)
        .setAppName("data_migration")
        .setMaster("master_ip")
        .set("spark.cassandra.connection.host", "cluster_1_hostnames")

val confCluster2 = new SparkConf(true)
        .setAppName("data_migration")
        .setMaster("master_ip")
        .set("spark.cassandra.connection.host", "cluster_2_hostnames")

val sc = new SparkContext(confCluster1)

sc.cassandraTable[Performer](KEYSPACE,PERFORMERS)
        .map[Performer](???)
        .saveToCassandra(KEYSPACE,PERFORMERS)
        (CassandraConnector(confCluster2),implicitly[RowWriterFactory[Performer]])
```

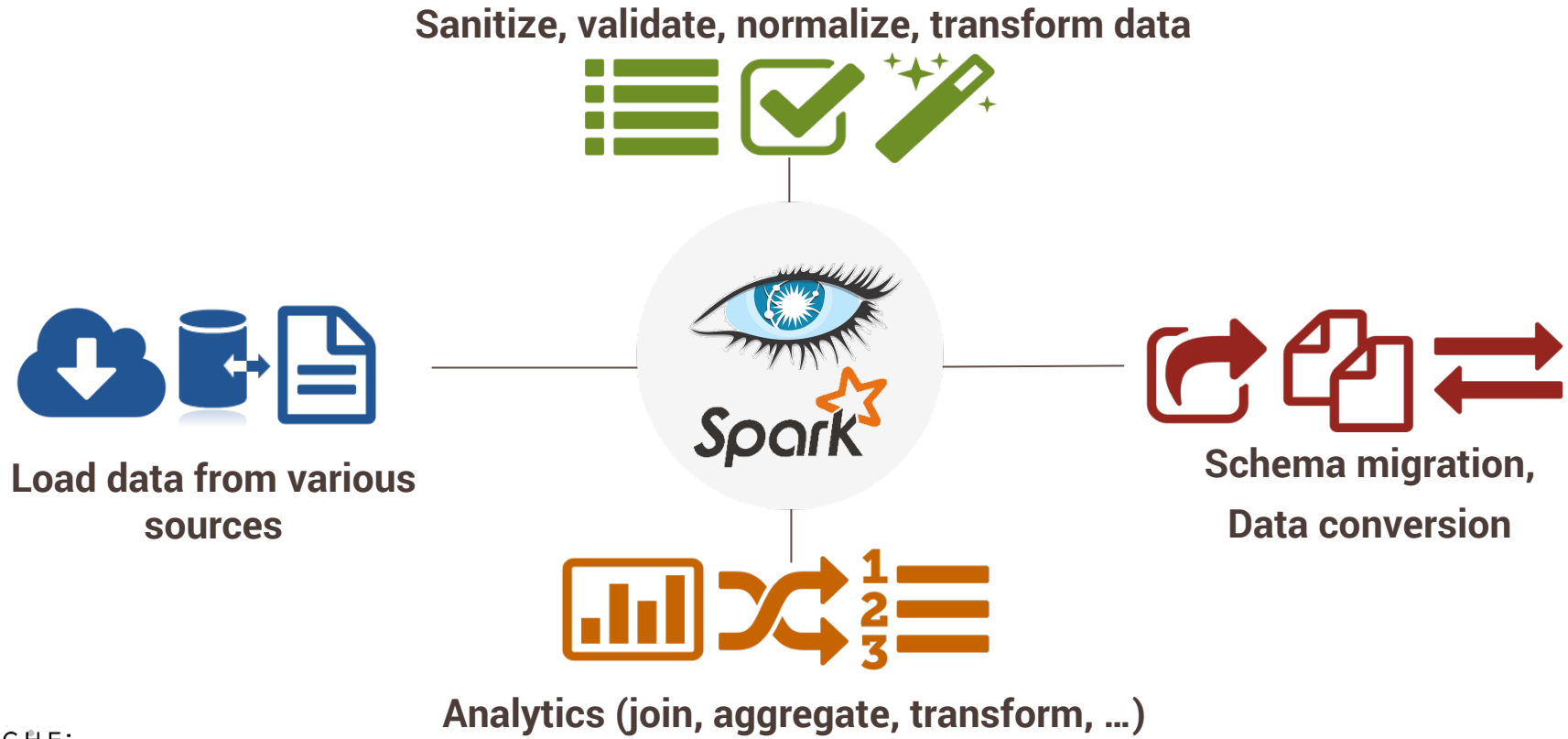
Spark/Cassandra use-case demos

Data cleaning

Schema migration

Analytics

Use Cases


Sanitize, validate, normalize, transform data
Spark
Load data from various sources
Schema migration, Data conversion
Analytics (join, aggregate, transform, …)

Data cleaning use-cases

Bug in your application ?

Dirty input data ?

☞ Spark job to clean it up! (**perfect data locality**)

Sanitize, validate, normalize, transform data

Data Cleaning

https://github.com/doanduyhai/incubator-zeppelin/tree/ApacheBigData

Schema migration use-cases

Business requirements change with time ?

Current data model no longer relevant ?

☞ Spark job to migrate data !

Schema migration,

Data conversion

Data Migration

https://github.com/doanduyhai/incubator-zeppelin/tree/ApacheBigData

Analytics use-cases

Given existing tables of performers and albums, I want to :

- count the number of albums releases by decade (70's, 80's, 90's, …)

☞ Spark job to compute analytics !

Analytics (join, aggregate, transform, …)

Analytics pipeline

① Read from production transactional tables

② Perform aggregation with Spark

③ Save back data into dedicated tables for fast visualization

④ Repeat step ①

Analytics

https://github.com/doanduyhai/incubator-zeppelin/tree/ApacheBigData

Thank You

@doanduyhai

duy_hai.doan@datastax.com

https://academy.datastax.com/