databricks™

Everyday I'm Shufflin

Tips for Writing Better Spark Jobs

Vida Ha & Holden Karau - Strata SJ 2015

Who are we?

Holden Karau

- Current: Software Engineer at Databricks.
- Co-author of “Learning Spark”.
- Past: Worked on search at Foursquare & Engineer @ Google.

Vida Ha

- Current: Solutions Engineer at Databricks.
- Past: Worked on scaling & distributed systems as a Software Engineer at Square and Google.

Our assumptions

- You know what **Apache Spark** is.
- You have (or will) use it.
- You want to understand Spark's internals, not just it's API's, to write **awesomer*** Spark jobs.

***awesomer** = more efficient, well-tested, reusable, less error-prone, etc.

Key Takeaways

- Understanding the **Shuffle** in Spark
  - Common cause of **inefficiency**.
- Understanding when **code runs on the driver vs. the workers**.
  - Common cause of **errors**.
- How to **factor your code**:
  - For **reuse** between batch & streaming.
  - For easy **testing**.

Good Old Word Count in Spark

```
sparkContext.textFile(“hdfs://...”)
    .flatMap(lambda line: line.split())
    .map(lambda word: (word, 1))
    .reduceByKey(lambda a, b: a + b)
```

wordcount(wordcount) < 100

What about GroupbyKey instead?

```
sparkContext.textFile(“hdfs://…”)
      .flatMap(lambda line: line.split())
      .map(lambda word: (word, 1))
      .groupByKey()
.map(lambda (w, counts): (w, sum(counts)))
```

Will we still get the right answer?

databricks™

ReduceByKey vs. GroupByKey

Answer: Both will give you the same answer.

But *reduceByKey* is more efficient.

In fact, *groupByKey* can cause of out of disk problems.

Examine the shuffle to understand.

databricks™

What's Happening in Word Count

Same Worker Node

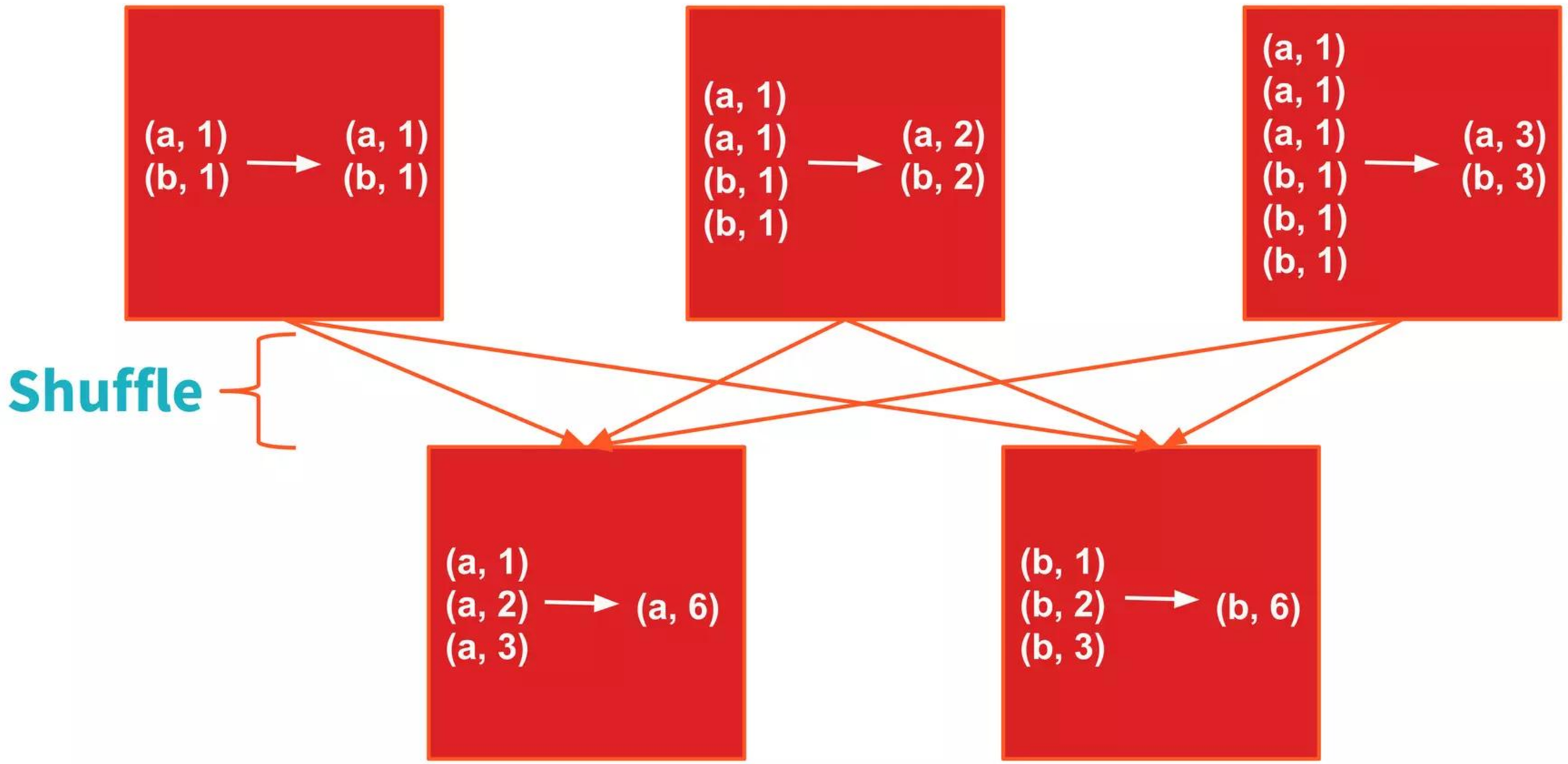
Triggers a shuffle of data

```
sparkContext.textFile("hdfs://...")
    .flatMap(lambda line: line.split())
    .map(lambda word: (word, 1))
    [reduceByKey or groupByKey]
    . . .
```

Shuffle occurs to transfer all data with the same key to the same worker node.

databricks™

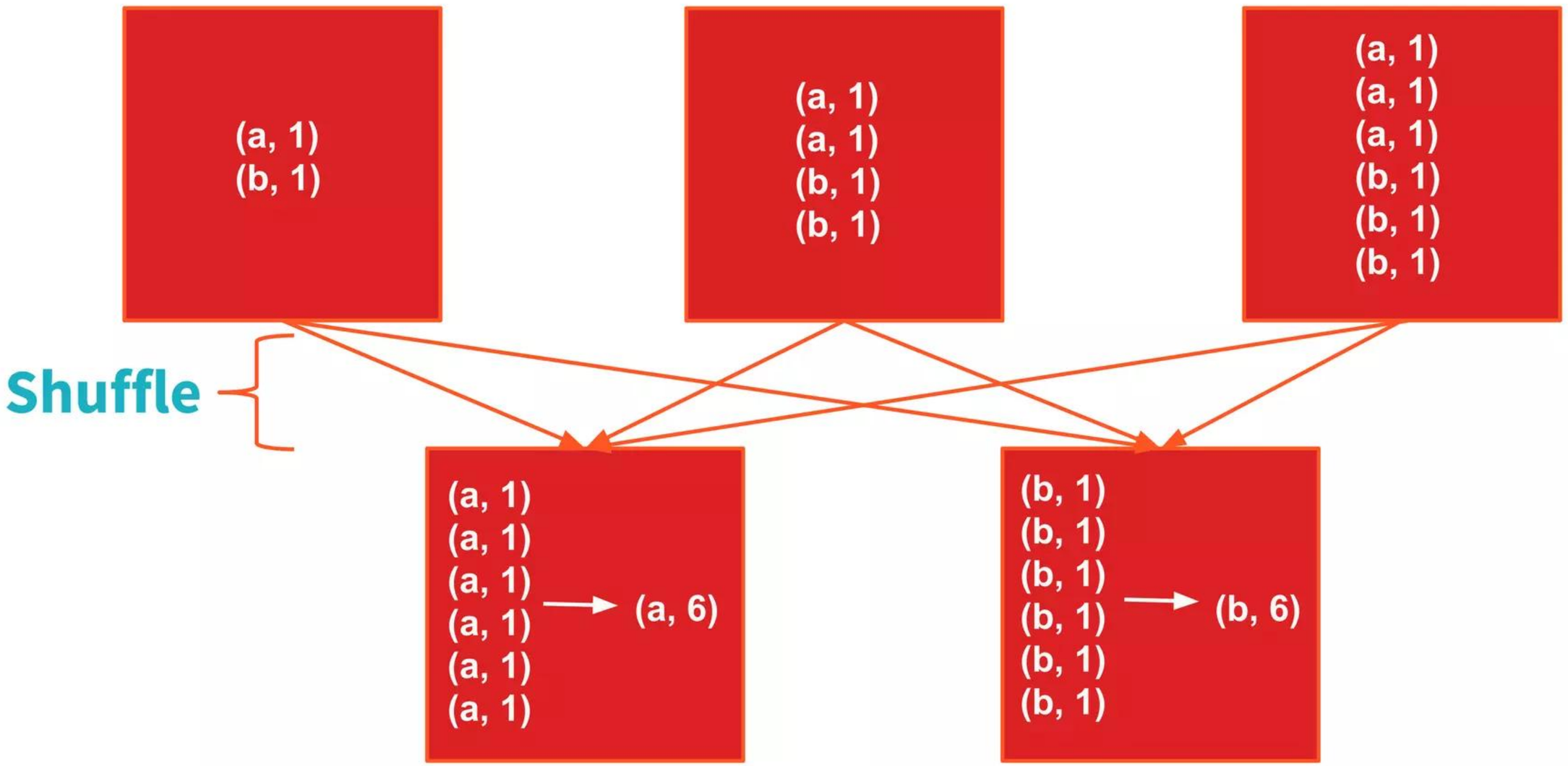
ReduceByKey: Shuffle Step



With ReduceByKey, data is combined so each partition outputs at most one value for each key to send over the network.

databricks™

GroupByKey: Shuffle Step



With GroupByKey, all the data is wastefully sent over the network and collected on the reduce workers.

databricks™

Prefer ReduceByKey over GroupByKey

Caveat: Not all problems that can be solved by *groupByKey* can be calculated with *reduceByKey*.

ReduceByKey requires combining all your values into another value with the exact same type.

reduceByKey, aggregateByKey, **foldByKey,** and **combineByKey**, preferred over **groupByKey**

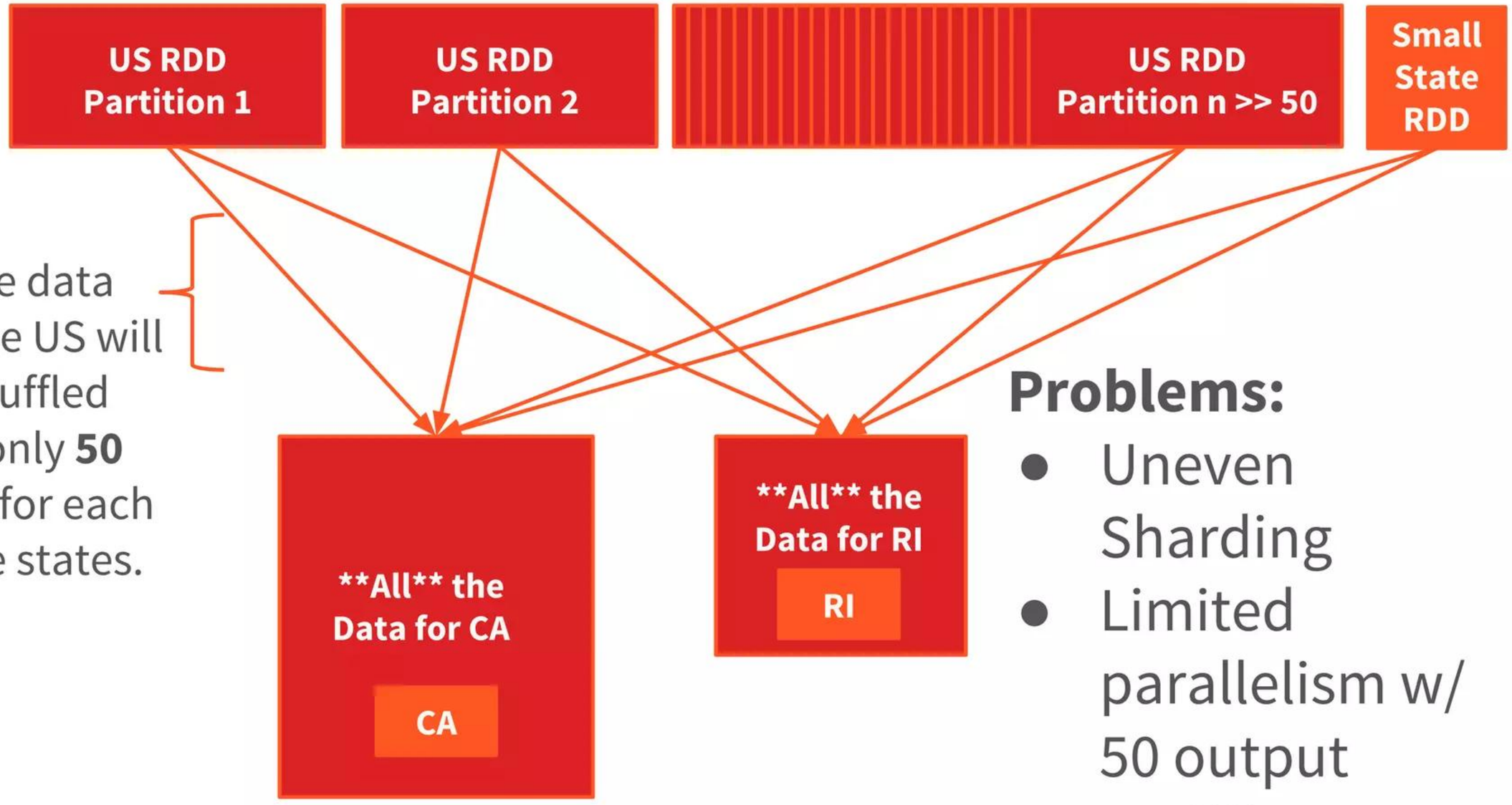
Join a Large Table with a Small Table

```
join_rdd = sqlContext.sql(“select *
  FROM people_in_the_us
  JOIN states
  ON people_in_the_us.state = states.name”)
```

```
print join_rdd.toDebugString()
```

- ShuffledHashJoin?
- BroadcastHashJoin?

ShuffledHashJoin



All the data for the US will be shuffled into only **50 keys** for each of the states.

Problems:

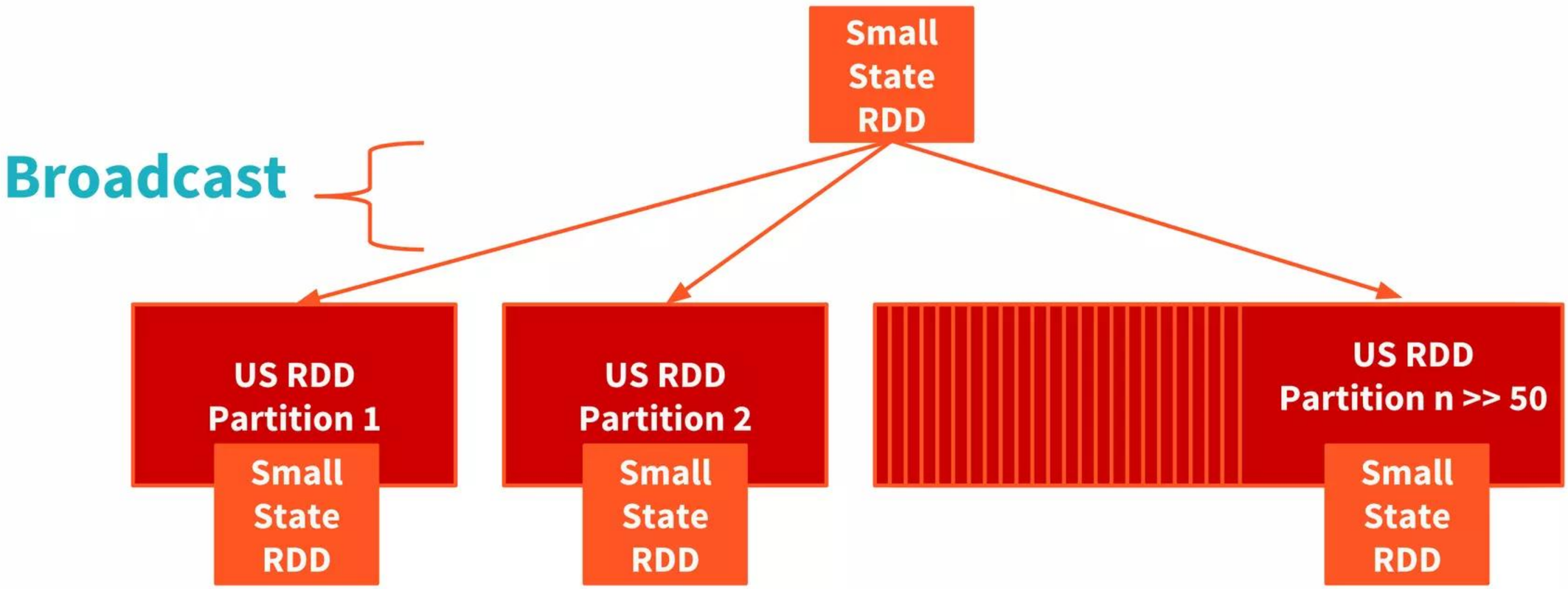
- Uneven Sharding
- Limited parallelism w/ 50 output partitions

Even a larger Spark cluster will not solve these problems!

databricks™

BroadcastHashJoin

Solution: Broadcast the Small RDD to all worker nodes.



Parallelism of the large RDD is maintained (n output partitions), and shuffle is not even needed.

databricks™

How to Configure BroadcastHashJoin

- See the Spark SQL programming guide for your Spark version for how to configure.
- For Spark 1.2:
  - Set spark.sql.autoBroadcastJoinThreshold.
  - sqlContext.sql("ANALYZE TABLE state_info COMPUTE STATISTICS noscan")
- Use .toDebugString() or EXPLAIN to double check.

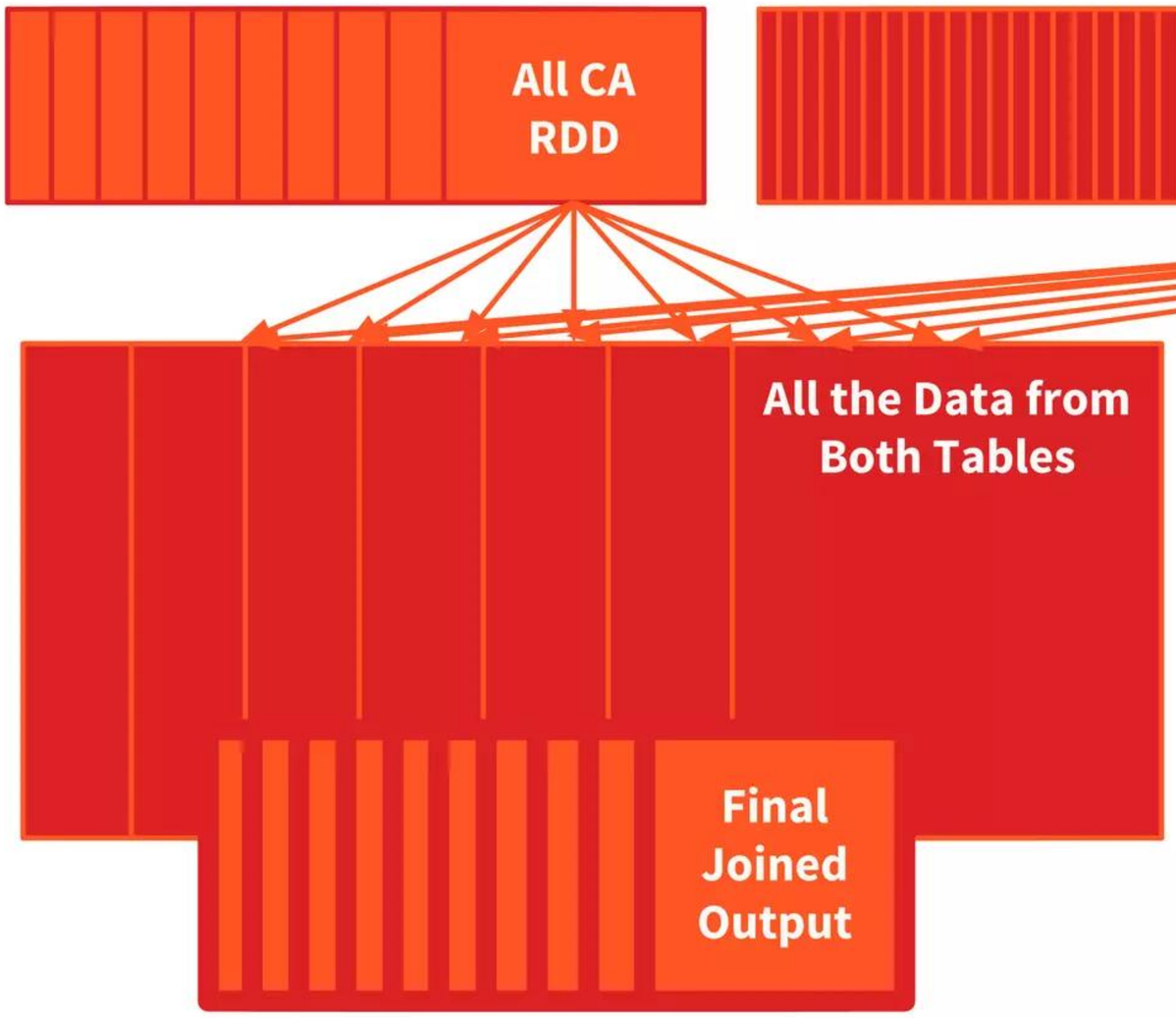
Join a Medium Table with a Huge Table

```
join_rdd = sqlContext.sql(“select *
  FROM people_in_california
  LEFT JOIN all_the_people_in_the_world
  ON people_in_california.id =
    all_the_people_in_the_world.id”)
```

Final output keys = keys people_in_california, so this don’t need a huge Spark cluster, right?

databricks™

Left Join - Shuffle Step



Shuffles everything before dropping keys

Not a Problem:

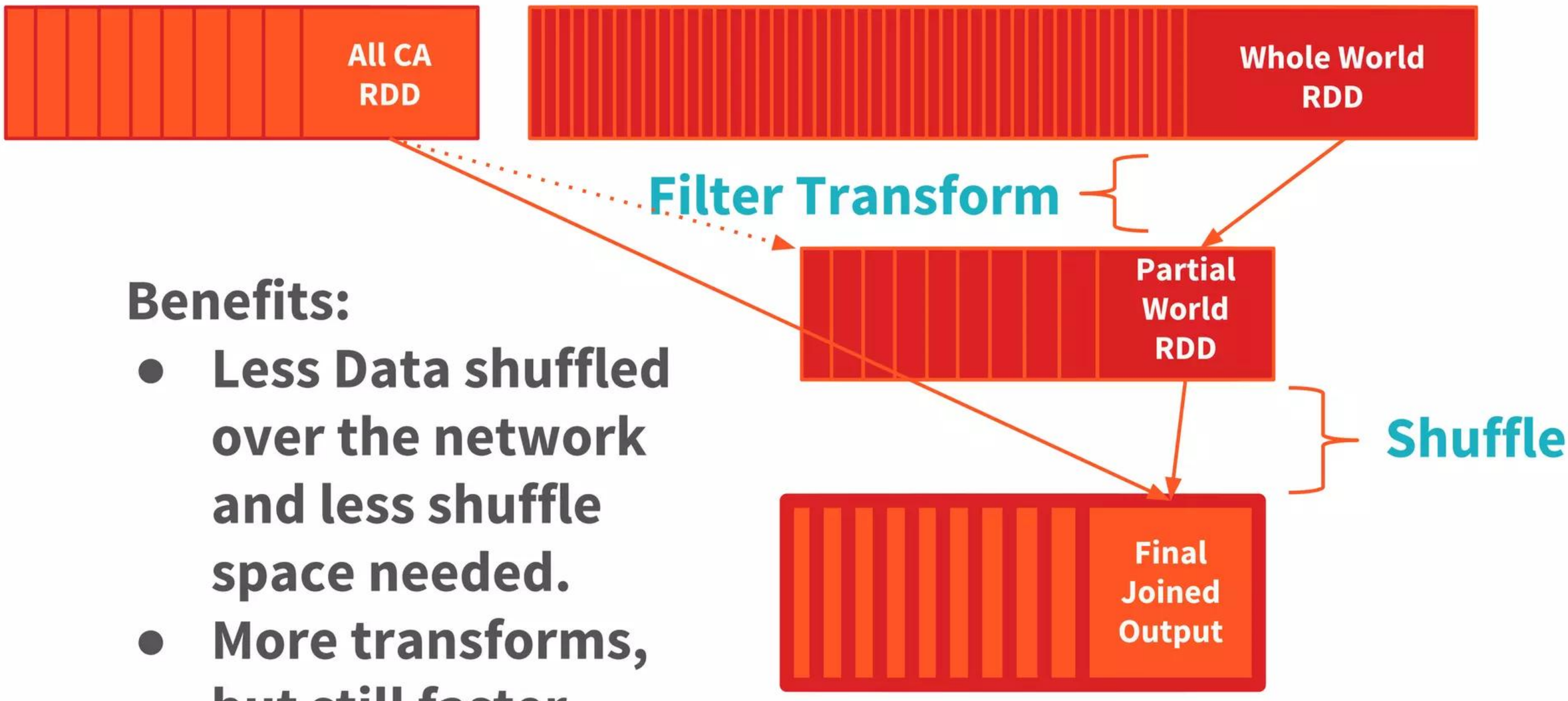
- Even Sharding
- Good Parallelism

The Size of the Spark Cluster to run this job is limited by the Large table rather than the Medium Sized Table.

databricks™

What's a Better Solution?

Filter the World World RDD for only entries that match the CA ID



Benefits:

- **Less Data shuffled over the network and less shuffle space needed.**
- **More transforms, but still faster.**

databricks™

What's the Tipping Point for Huge?

- Can't tell you.
- There aren't always strict rules for optimizing.
- If you were only considering two small columns from the World RDD in Parquet format, the filtering step may not be worth it.

You should understand your data and it's unique properties in order to best optimize your Spark Job.

In Practice: Detecting Shuffle Problems

Tasks

Index	ID	Attempt	Status	Locality Level	Executor ID / Host	Launch Time	Duration	Scheduler Delay	Task Deserialization Time	GC Time	Result Serialization Time	Getting Result Time	Input	Write Time	Shuffle Write	Errors
0	23	0	SUCCESS	PROCESS_LOCAL	0 / ip-10-0-187-189.us-west-2.compute.internal	2015/02/03 22:03:32	9 s	10 ms	3 ms	0.1 s	0 ms	0 ms	60.9 MB (hadoop)	12 ms	5.1 MB	
1	24	0	SUCCESS	PROCESS_LOCAL	0 / ip-10-0-187-189.us-west-2.compute.internal	2015/02/03 22:03:32	9 s	10 ms	1 ms	0.1 s	0 ms	0 ms	56.4 MB (hadoop)	14 ms	4.7 MB	
3	26	0	SUCCESS	PROCESS_LOCAL	0 / ip-10-0-187-189.us-west-2.compute.internal	2015/02/03 22:03:32	8 s	12 ms	1 ms	0.1 s	0 ms	0 ms	54.3 MB (hadoop)	15 ms	4.3 MB	
2	25	0	SUCCESS	PROCESS_LOCAL	0 / ip-10-0-187-189.us-west-2.compute.internal	2015/02/03 22:03:32	9 s	11 ms	1 ms	0.1 s	0 ms	0 ms	54.5 MB (hadoop)	11 ms	4.4 MB	
4	27	0	SUCCESS	PROCESS_LOCAL	0 / ip-10-0-187-189.us-west-2.compute.internal	2015/02/03 22:03:40	8 s	7 ms	1 ms	0.1 s	0 ms	0 ms	54.0 MB (hadoop)	11 ms	4.4 MB	
5	28	0	SUCCESS	PROCESS_LOCAL	0 / ip-10-0-187-189.us-west-2.compute.internal	2015/02/03 22:03:40	8 s	8 ms	1 ms	0.1 s	0 ms	0 ms	53.7 MB (hadoop)	10 ms	4.3 MB	
6	29	0	SUCCESS	PROCESS_LOCAL	0 / ip-10-0-187-189.us-west-2.compute.internal	2015/02/03 22:03:41	7 s	7 ms	1 ms	0.1 s	0 ms	0 ms	54.0 MB (hadoop)	10 ms	4.4 MB	
7	30	0	SUCCESS	PROCESS_LOCAL	0 / ip-10-0-187-189.us-west-2.compute.internal	2015/02/03 22:03:41	8 s	7 ms	1 ms	0.1 s	0 ms	0 ms	54.4 MB (hadoop)	10 ms	4.3 MB	
8	31	0	RUNNING	PROCESS_LOCAL	0 / ip-10-0-187-189.us-west-2.compute.internal	2015/02/03 22:03:47	6 s	0 ms	0 ms		0 ms	0 ms				
9	32	0	RUNNING	PROCESS_LOCAL	0 / ip-10-0-187-189.us-west-2.compute.internal	2015/02/03 22:03:48	5 s	0 ms	0 ms		0 ms	0 ms				
10	33	0	RUNNING	PROCESS_LOCAL	0 / ip-10-0-187-189.us-west-2.compute.internal	2015/02/03 22:03:48	5 s	0 ms	0 ms		0 ms	0 ms				
11	34	0	RUNNING	PROCESS_LOCAL	0 / ip-10-0-187-189.us-west-2.compute.internal	2015/02/03 22:03:49	4 s	0 ms	0 ms		0 ms	0 ms				

Check the Spark UI pages for task level detail about your Spark job.

Things to Look for:

- Tasks that take much longer to run than others.
- Speculative tasks that are launching.
- Shards that have a lot more input or shuffle output than others.

databricks™

Execution on the Driver vs. Workers

The **main program** are executed on the **Spark Driver.**

Transformations are executed on the **Spark Workers.**

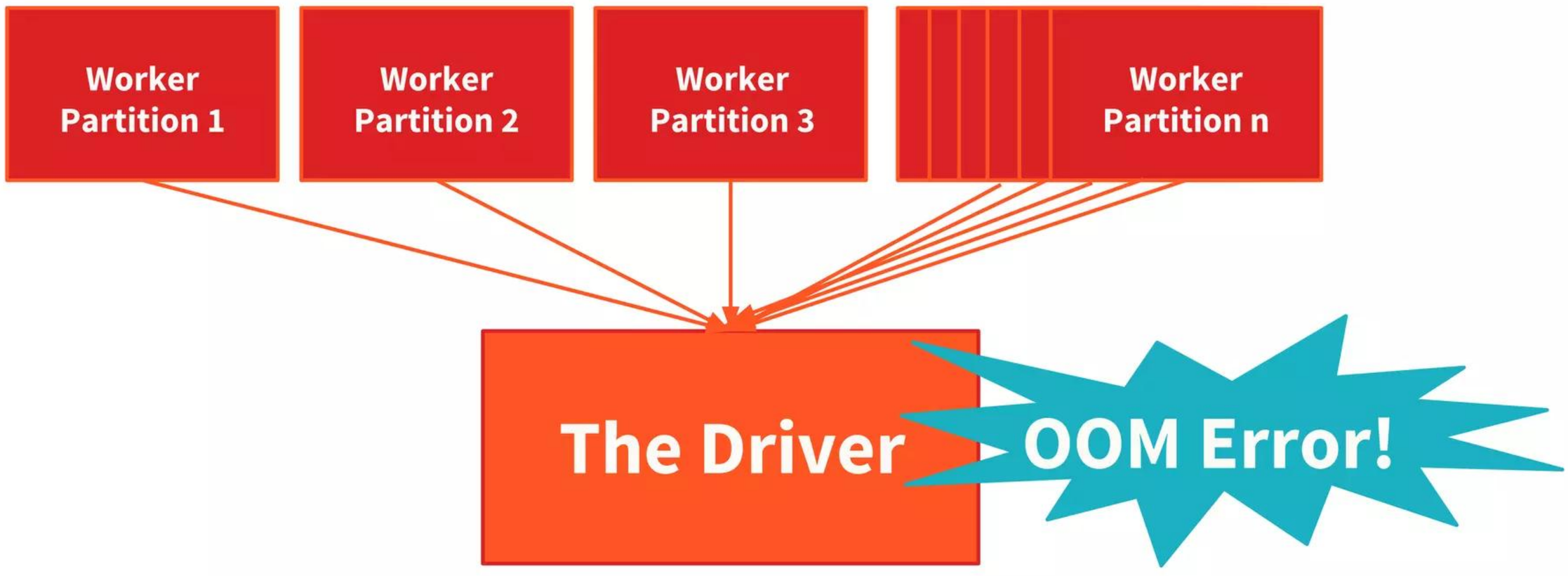
```
output = sparkContext
    .textFile("hdfs://...")
    .flatMap(lambda line: line.split())
    .map(lambda word: (word, 1))
    .reduceByKey(lambda a, b: a + b)
    .collect()
print output  ...
```

Actions *may* transfer data from the **Workers** to the **Driver.**

databricks™

What happens when calling collect()

collect() sends all the partitions to the single driver



collect() on a large RDD can trigger a OOM error

databricks™

Don’t call collect() on a large RDD

Be cautious with all actions that may return ***unbounded output***.

Option 1: Choose actions that return a bounded output per partition, such as **count()** or **take(N)** .

Option 2: Choose actions that outputs directly from the workers such as **saveAsTextFile().**

databricks™

Commonly Serialization Errors

Hadoop Writables

Map to/from a serializable form

Capturing a full Non-Serializable object

Copy the required serializable parts locally

Network Connections

Create the connection on the worker

databricks™

Serialization Error

```
myNonSerializable = …
output = sparkContext
    .textFile(“hdfs://…”)
    .map(lambda l: myNonSerializable.value + l)
    .take(n)
print output  …
```

Spark will try to send myNonSerializable from the Driver to the Worker node by serializing it, and error.

RDDs within RDDs - not even once

Only the driver can perform operations on RDDs

map+get:

```
rdd.map{(key, value) => otherRdd.get(key)...}
```

can normally be replaced with a join:

```
rdd.join(otherRdd).map{}
```

map+map:

```
rdd.map{e => otherRdd.map{ ... }}
```

is normally an attempt at a cartesian:

```
rdd.cartesian(otherRdd).map()
```

Writing a Large RDD to a Database

Option 1: DIY

- Initialize the Database Connection on the Worker rather than the Driver
  - Network sockets are non-serializable
- Use foreachPartition
  - Re-use the connection between elements

Option 2: DBOutputFormat

- Database must speak JDBC
- Extend DBWritable and save with saveAsHadoopDataset

DIY: Large RDD to a Database

Cat photo from https://www.flickr.com/photos/rudiriet/140901529/

databricks™

DIY: Large RDD to a Database

```
data.forEachPartition{records => {
  // Create the connection on the executor
  val connection = new HappyDatabase(...)
  records.foreach{record =>
    connection.//implementation specific
        }
    }
}
```

DBOutputFormat

```
case class CatRec(name: String, age: Int) extends DBWritable
{
  override def write(s: PreparedStatement ) {
    s.setString(1, name); s.setInt(2, age)
}}
val tableName = "table"
val fields = Array("name", "age")
val job = new JobConf()
DBConfiguration.configureDB(job,  "com.mysql.jdbc.Driver",
"..")
DBOutputFormat.setOutput(job, tableName, fields:_*)
records.saveAsHadoopDataset(job)
```

Reuse Code on Batch & Streaming

Streaming

```
val ips = logs.transform
(extractIp)
```

Batch

```
def extractIp(
   logs: RDD[String]) = {
    logs.map(_.split(“ “)(0))
}

val ips = extractIp(logs)
```

Use transform on a DStream to reuse your RDD to RDD functions from your batch Spark jobs.

databricks™

Reuse Code on Batch & Streaming

Streaming

```
tweets.foreachRDD{(tweetRDD, time) =>
 writeOutput(tweetRDD)
}
```

Use foreachRDD on a DStream to reuse your RDD output functions from your batch Spark jobs.

Batch

```
def writeOutput(ft..) = {
  val preped = ft.map(prep)
  preped.savetoEs(esResource)
}

val ft = tIds.mapPartition{
tweetP =>
  val twttr = TwitterFactory.
getSingleton()
  tweetP.map{ t =>
    twttr.showStatus(t.toLong)
  }}
writeOutput(ft)
```

databricks™

Testing Spark Programs

- Picture of a cat
- Unit-tests of functions
- Testing with RDDs
- Special Considerations for Streaming

Cat photo by Jason and Kris Carter

Simplest - Unit Test Functions

instead of:

```
val splitLines = inFile.map(line => {
  val reader = new CSVReader(new StringReader(line))
  reader.readNext()
})
```

write:

```
def parseLine(line: String): Array[Double] = {
  val reader = new CSVReader(new StringReader(line))
  reader.readNext().map(_.toDouble)
}
```

then we can:

```
test("should parse a csv line with numbers") {
    MoreTestableLoadCsvExample.parseLine("1,2") should equal
(Array[Double](1.0, 2.0))
}
```

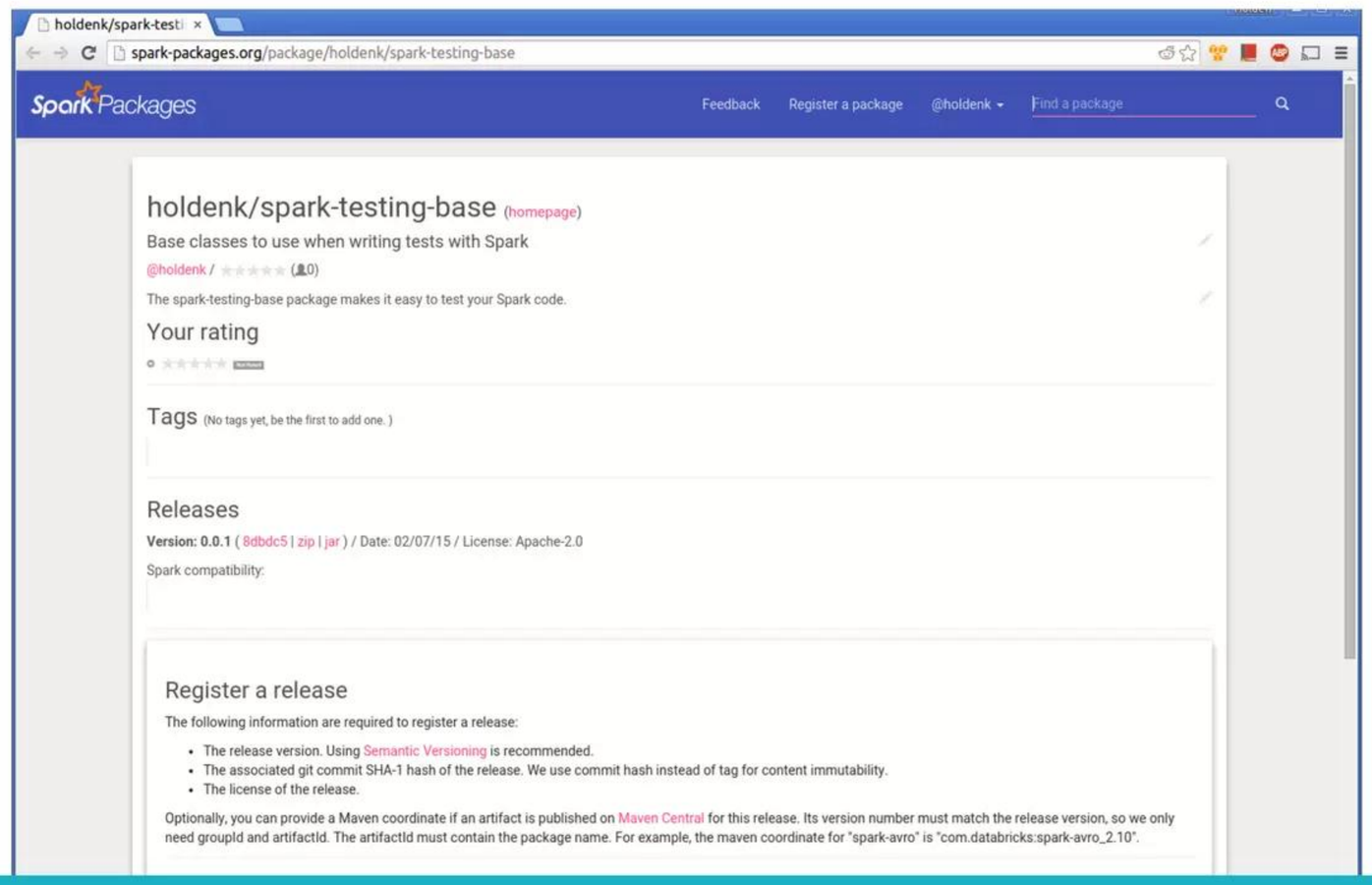
Testing with RDDs

```
trait SSC extends BeforeAndAfterAll { self: Suite =>

  @transient private var _sc: SparkContext = _
  def sc: SparkContext = _sc

  var conf = new SparkConf(false)

  override def beforeAll() {
    _sc = new SparkContext("local[4]", "test", conf)
    super.beforeAll()
  }

  override def afterAll() {
    LocalSparkContext.stop(_sc)
    _sc = null
    super.afterAll()
  }
}
```

databricks™

Testing with RDDs

Or just include http://spark-packages.org/package/holdenk/spark-testing-base



Link to spark-testing-base from spark-packages.

databricks™

Testing with RDDs continued

```
test("should parse a csv line with numbers") {
    val input = sc.parallelize(List("1,2"))
    val result = input.map(parseCsvLine)
    result.collect() should equal (Array[Double](1.0, 2.0))
}
```

Testing with DStreams

Some challenges:

- creating a test DStream
- collecting the data to compare against locally
  - use foreachRDD & a var
- stopping the streaming context after the input stream is done
  - use a manual clock
    - (private class)
  - wait for a timeout
    - slow

Testing with DStreams - fun!

```
class SampleStreamingTest extends StreamingSuiteBase {
  test("really simple transformation") {
      val input = List(List("hi"), List("hi holden"),
                       List("bye"))
      val expect = List(List("hi"),
                        List("hi", "holden"),
                        List("bye"))
      testOperation[String, String](input, tokenize _,
        expect, useSet = true)
  }
}
```

databricks™

THE END

Cat picture from http://galato901.deviantart.com/art/Cat-on-Work-Break-173043455

The DAG - is it magic?

Dog photo from: Pets Adviser by http://petsadviser.com

databricks™