Learning the Core of Spark: RDDs

Xavier Morera

HELPING DEVELOPERS UNDERSTAND SEARCH & BIG DATA

@xmorera   www.xaviermorera.com

Learning the Core of Spark: RDDs

Resilient Distributed Dataset

How can I use DataFrames in 2.0

What is an RDD and Schema RDD

How do I group by a field

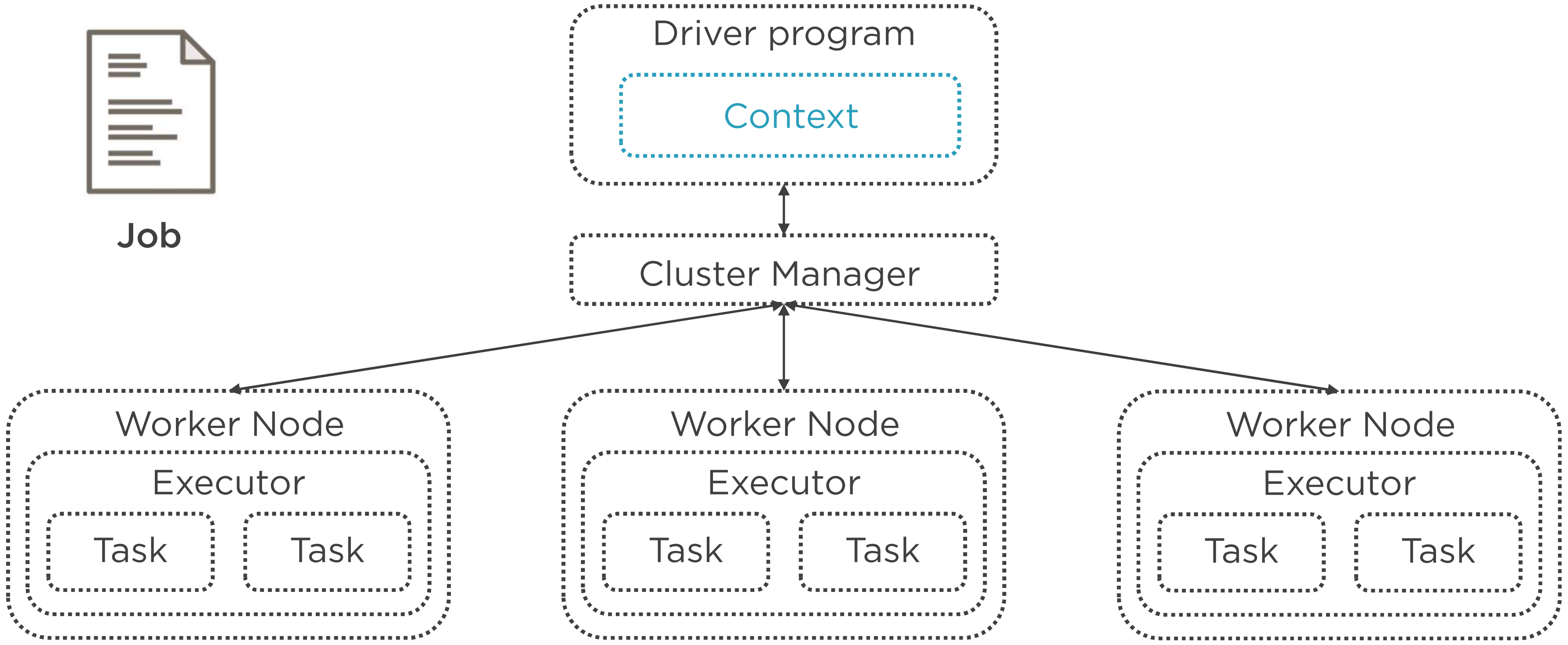
Can I use Hive from HUE

Datasets (DataFrames) & Spark SQL

Why Start with RDDs?

1. Knowing RDDs gives you a better foundation for writing Spark Code
2. Sometimes RDDs are the right tool for the job
3. Existing Spark 1.x code

SparkContext



You will need it to work with RDDs!

SparkContext

Spark Application

- Created for you in the shell
- You need to create in an application

Only one SparkContext

SparkContext

Spark Application

Configuration & Environment

Services

Create RDDs

```
sc
spark.sparkContext
sc.sparkUser
sc.getConf.getAll
```

SparkContext

SparkContext available as sc

Many methods and properties available

Get configuration and environment information

```
sc.stop
sc
val test_rdd = sc.emptyRDD
test_rdd.collect()
import org.apache.spark.SparkContext
val sc = SparkContext.getOrCreate()
test_rdd = sc.emptyRDD
test_rdd.collect()
```

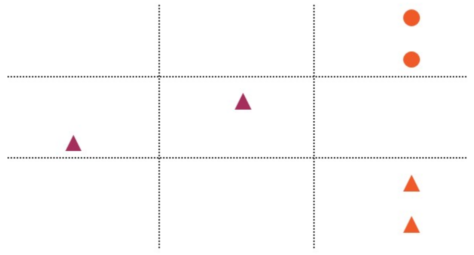
SparkContext

Stop and create a new SparkContext

- getOrCreate()

Used to create RDDs

RDD



How can I use DataFrames in 2.0

What is an RDD and Schema RDD

How do I group by a field

Can I use Hive from HUE

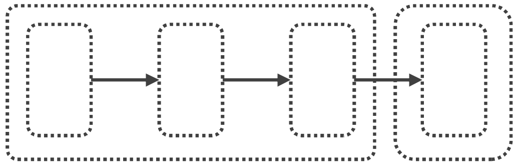
How can I use DataFrames in 2.0

What is an RDD and Schema RDD

How do I group by a field

Can I use Hive from HUE

Resilient **Distributed** **Dataset**



Five Properties of an RDD

Partitions

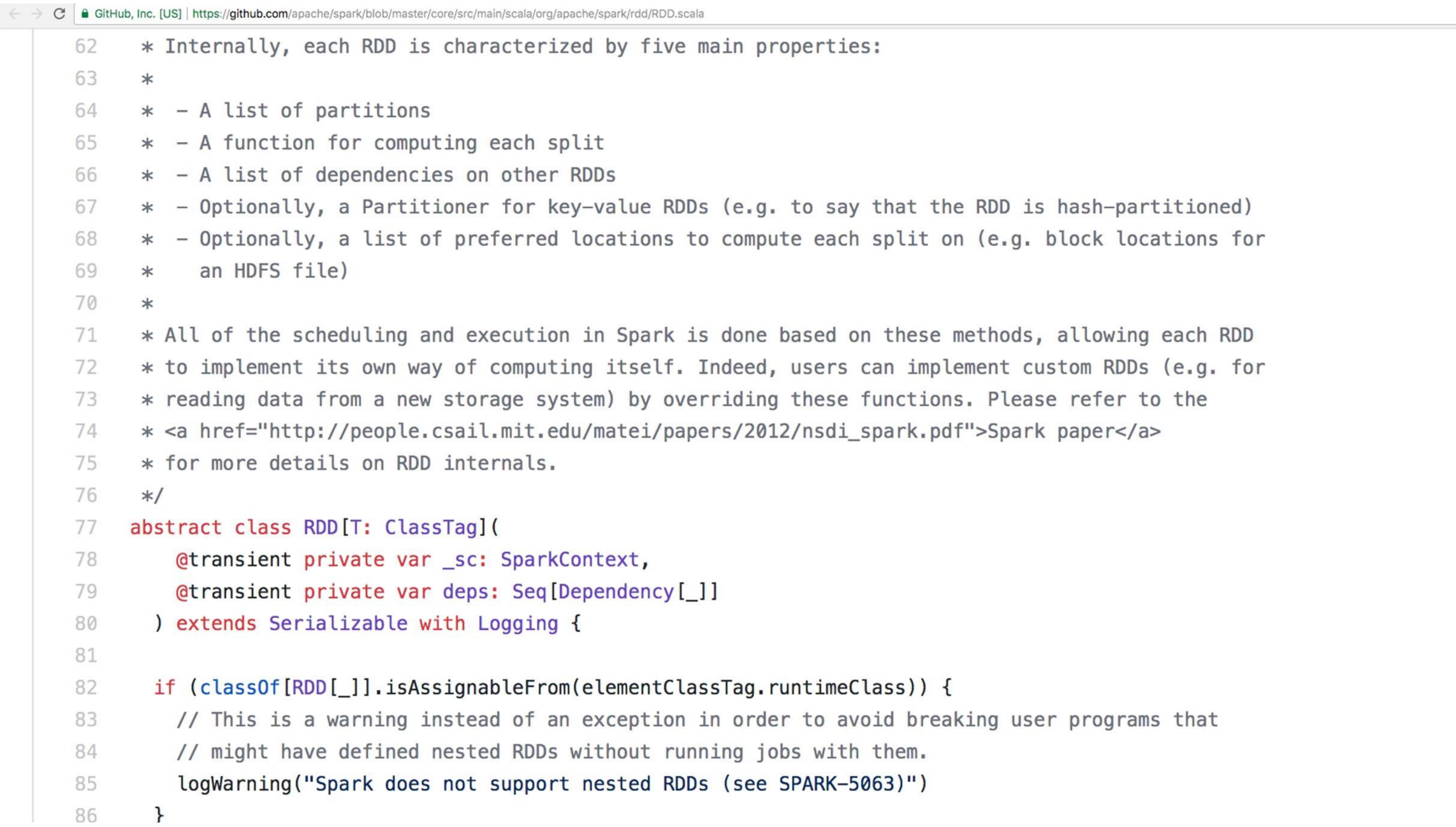
Dependencies

Function to compute partition

Optional

Partitioner *(key/value RDDs)*

Preferred locations for compute



GitHub, Inc. [US] | https://github.com/apache/spark/blob/master/core/src/main/scala/org/apache/spark/rdd/RDD.scala

```scala
 * Internally, each RDD is characterized by five main properties:
 *
 *  - A list of partitions
 *  - A function for computing each split
 *  - A list of dependencies on other RDDs
 *  - Optionally, a Partitioner for key-value RDDs (e.g. to say that the RDD is hash-partitioned)
 *  - Optionally, a list of preferred locations to compute each split on (e.g. block locations for
 *    an HDFS file)
 *
 * All of the scheduling and execution in Spark is done based on these methods, allowing each RDD
 * to implement its own way of computing itself. Indeed, users can implement custom RDDs (e.g. for
 * reading data from a new storage system) by overriding these functions. Please refer to the
 * <a href="http://people.csail.mit.edu/matei/papers/2012/nsdi_spark.pdf">Spark paper</a>
 * for more details on RDD internals.
 */
abstract class RDD[T: ClassTag](
    @transient private var _sc: SparkContext,
    @transient private var deps: Seq[Dependency[_]]
  ) extends Serializable with Logging {

  if (classOf[RDD[_]].isAssignableFrom(elementClassTag.runtimeClass)) {
    // This is a warning instead of an exception in order to avoid breaking user programs that
    // might have defined nested RDDs without running jobs with them.
    logWarning("Spark does not support nested RDDs (see SPARK-5063)")
  }
```

PairRDD

RDDs of key/value pairs

Contain tuples

Useful for grouping or aggregating

Can use RDD transformations

PairRDD specific transformations available

Creating RDDS

Parallelize

External Data

From Another RDD

* or Dataset

DataFrame

(upcoming module)

```
Array(1,2,3,4,5)
val list_one_to_five = sc.parallelize(Array(1,2,3,4,5))
list_one_to_five
list_one_to_five.collect()
```

Creating RDDs with Parallelize

Create RDDs with data in memory: collection or iterable

Parallelize a collection

Bring data back to driver with collect()

```
list_one_to_five.getNumPartitions
list_one_to_five.glom().collect()
val list_one_to_five = sc.parallelize(Array(1,2,3,4,5), 1)
list_one_to_five.getNumPartitions
list_one_to_five.glom().collect()
```

RDD Partitions with Parallelize

Call getNumPartitions() to check number of partitions

Use glom() to see our data within partitions

And you can control how many partitions you want

```
list_one_to_five.sum()
list_one_to_five.min()
list_one_to_five.max()
list_one_to_five.mean()
list_one_to_five.first()
```

An Operation on an RDD

What can I do on an RDD?

Call sum() to add up values

And first() to get first element

```
val list_dif_types = sc.parallelize(Seq(false, 1, "two",
  Map("three" -> 3), ("xavier", 4)))
list_dif_types.collect()
val tuple_rdd = sc.parallelize(Seq(("xavier" -> 1), ("irene" -> 2)))
tuple_rdd
tuple_rdd.setName("tuple_rdd")
tuple_rdd
```

Different Types of Objects

RDDs can hold different types of objects

Strings, ints, dictionaries, and key/value pairs (tuple) are common

RDD is given an internal name, but you can set it too

```
val empty_rdd = sc.parallelize(Array[String]())
empty_rdd.collect()
empty_rdd.isEmpty()
val other_empty = sc.emptyRDD
other_empty.isEmpty()
```

EmptyRDD

You can create empty RDDs

With parallelize()

Or emptyRDD

```
val bigger_rdd = sc.parallelize(1 to 1000)
bigger_rdd.collect()
bigger_rdd.sum()
bigger_rdd.count()
bigger_rdd.getNumPartitions
```

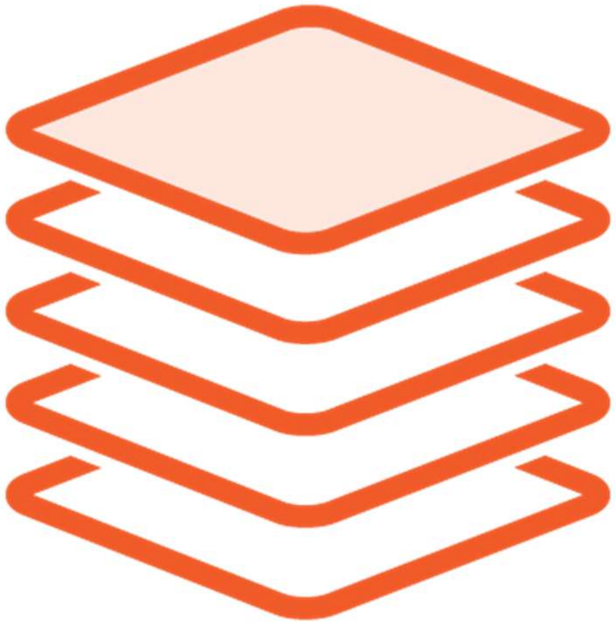
Ranges

Create RDDs with a range (1 to 1000)

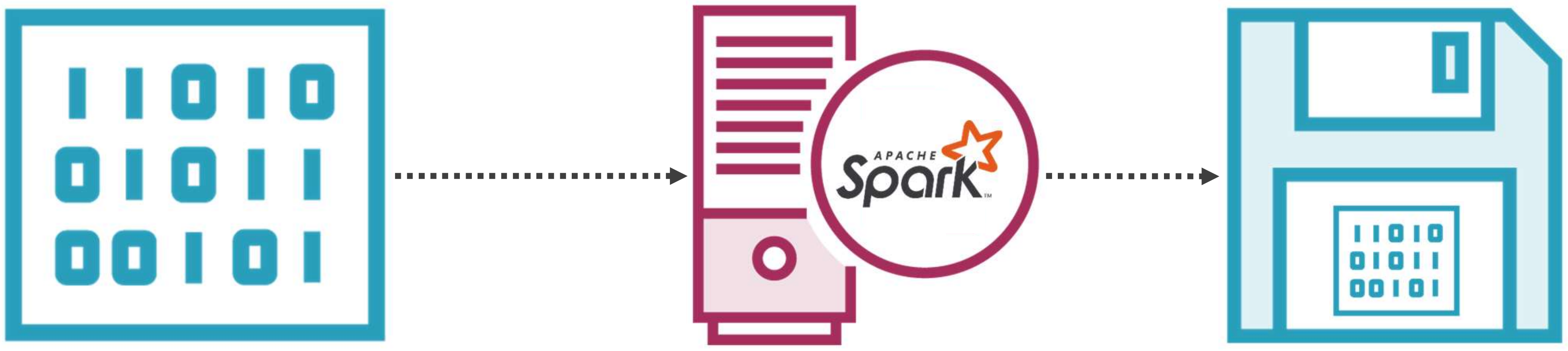
And use methods to get information from our RDD data

Count() all elements in my rdd

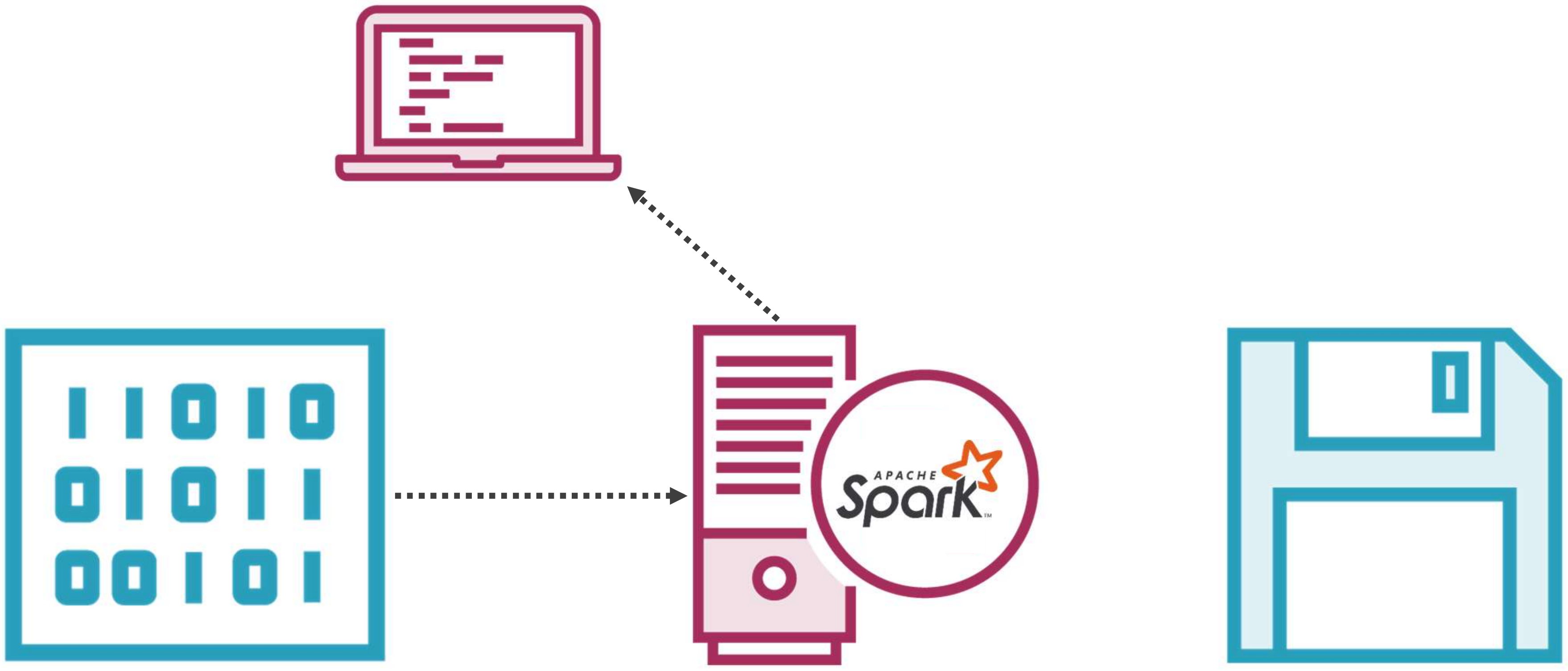
Parallelize



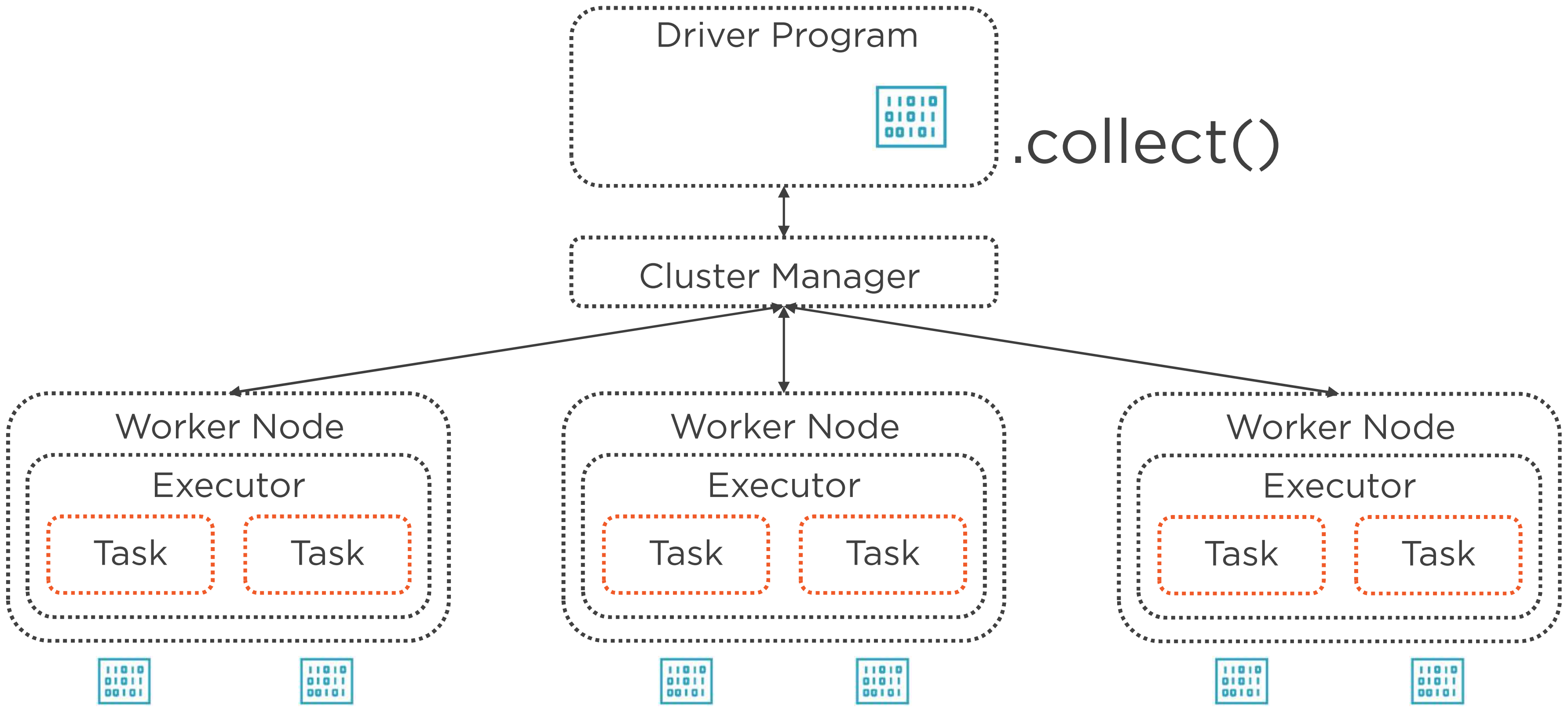
Returning Data to the Driver



Returning Data to the Driver



Returning Data to the Driver



```
bigger_rdd.collect()
bigger_rdd.take(10)
bigger_rdd.first()
bigger_rdd.takeOrdered(10)(Ordering[Int].reverse)
val tuple_map = tuple_rdd.collectAsMap()
tuple_map("xavier")
```

Returning Data to the Driver

Bring data back RDD to Driver

Most common action for testing is collect()

Also first(), take(), takeOrdered(), collectAsMap() among others

⚠ Beware!

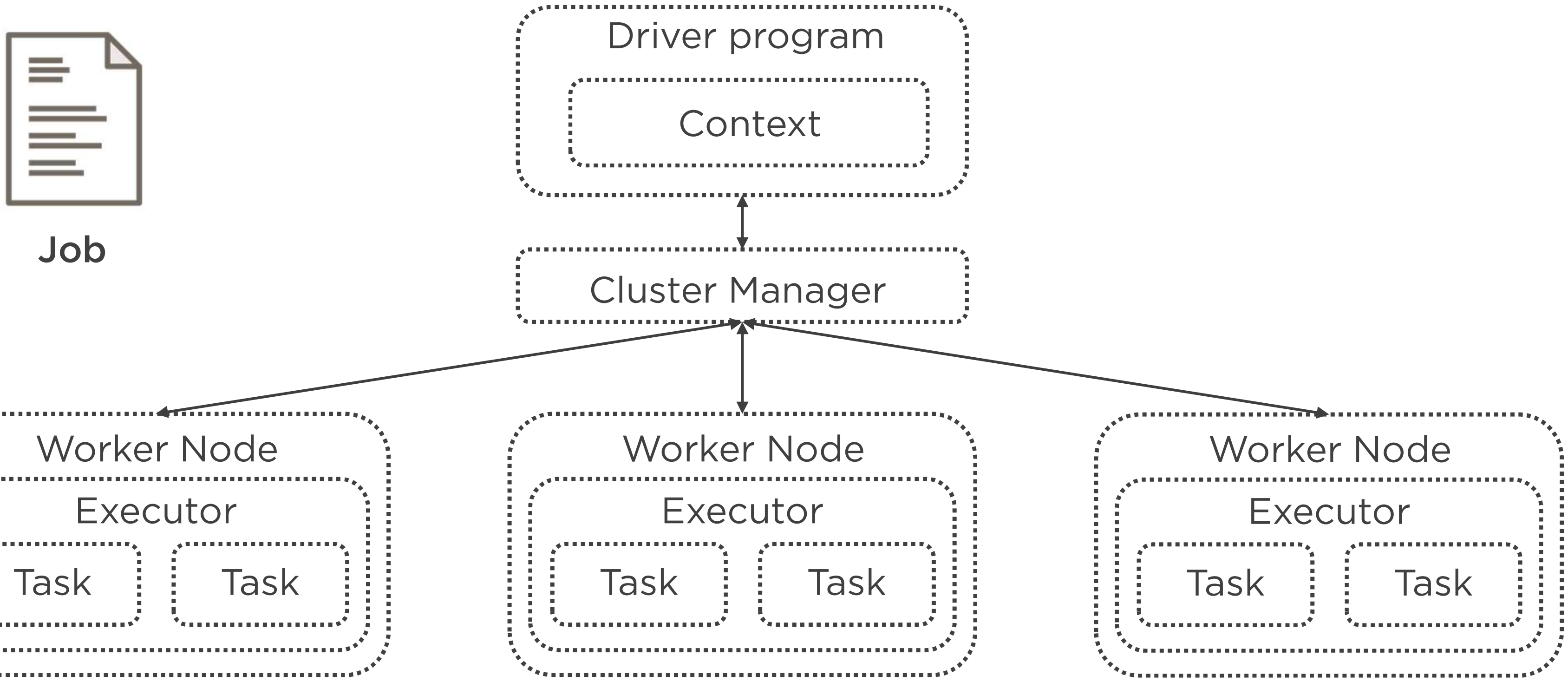
Potential out of memory exception on large datasets

```
for (elem <- bigger_rdd.take(10)) println(elem)

for ((k, v) <- tuple_map)
  println(k + " / " + v)
```

Iterating & Printing Data

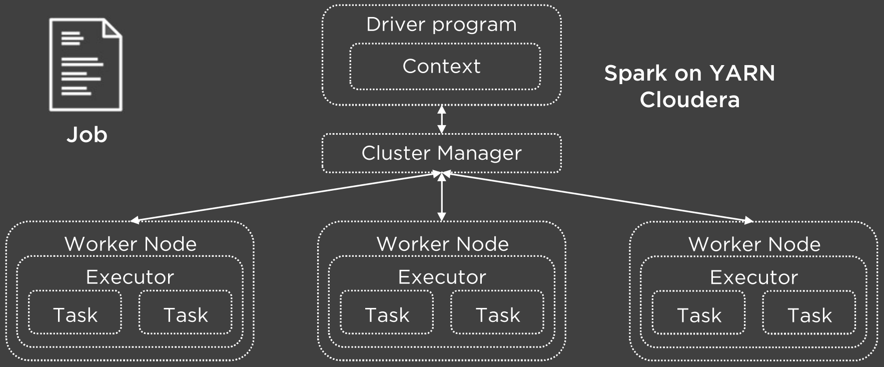
Iterate over list of data returned using take() in for loop

Or other types, i.e. dict


Job
Driver program
Context
Cluster Manager
Worker Node
Executor
Task
Task
Worker Node
Executor
Task
Task
Worker Node
Executor
Task
Task

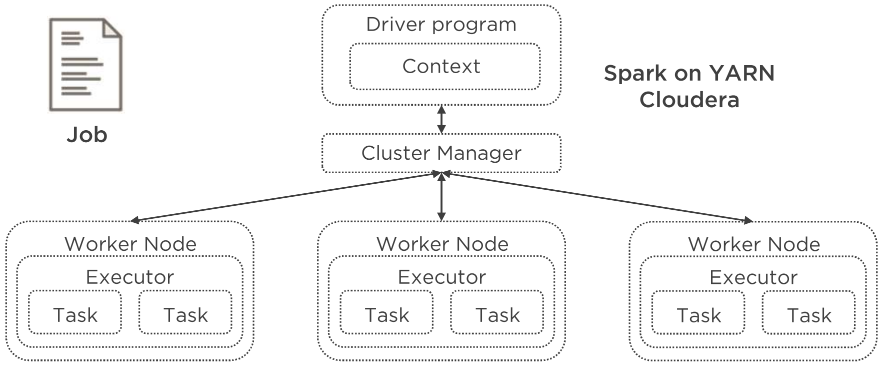
Driver

```
val test_write = sc.parallelize(
  Array("xavier", "troy"))

println(test_write.collect()
  .mkString(" "))
```



In executor

```
def in_executor(entry: String) = {
  println("** inExecutorEntry **")
  println(entry)
  "printed: " + entry}
```



```
spark2-submit inExecutor EntryLogLevelAll.scala
```

```
def log_search(url: String): String = {
  val page =
scala.io.Source.fromURL("http://tiny.bigdatainc.org/" +
url).mkString
  println(page)
  page
}
val queries = sc.parallelize(Array("ts1", "ts2"))
queries.foreach(log_search)
```

Foreach

Executes the function on each element in the RDD

Runs in the worker nodes

Useful for calling external resources like a database or an API

```
bigger_rdd.getNumPartitions()
play_part=bigger_rdd.repartition(10)
play_part.getNumPartitions()
```

(Re)Partitions, Coalesce, Saving Text & HUE

bigger_rdd was partitioned on parallelize()

Change number of partitions with repartition() and coalesce()

```
play_part.repartition(14).getNumPartitions
play_part.coalesce(15).getNumPartitions
```

Repartition & Coalesce

repartition() specifies new number of partitions, up or down

coalesce() uses existing partitions, only decreases, no shuffling

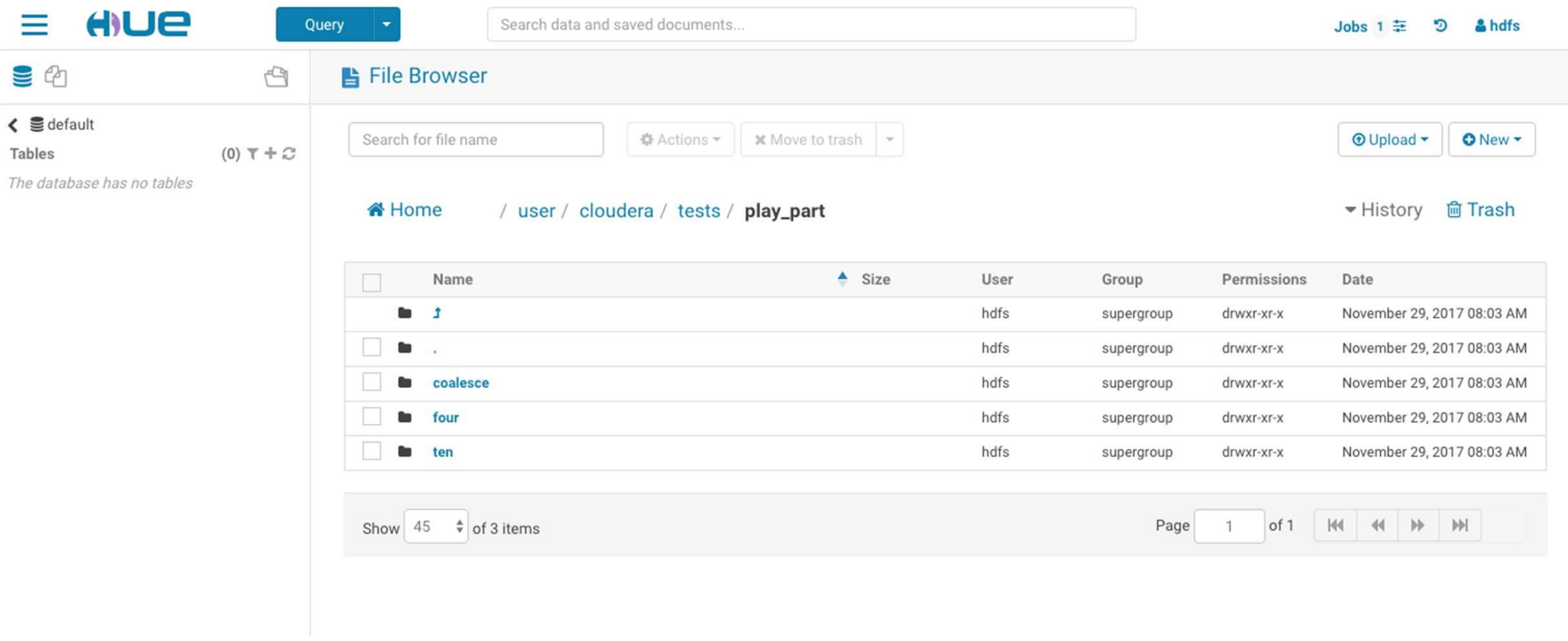
Data in RDD remains the same. How can we tell?

```
play_part.saveAsTextFile("/user/cloudera/tests/play_part/ten")
play_part.repartition(4).
  saveAsTextFile("/user/cloudera/tests/play_part/four")
play_part.coalesce(1).
  saveAsTextFile("/user/cloudera/tests/play_part/coalesce")
```

Confirm Data in RDD

Save using saveAsTextFile() and visualize with HUE

Just like Hadoop, creates file per partition



		Name	Size	User	Group	Permissions	Date
		↥		hdfs	supergroup	drwxr-xr-x	November 29, 2017 08:03 AM
		.		hdfs	supergroup	drwxr-xr-x	November 29, 2017 08:03 AM
		coalesce		hdfs	supergroup	drwxr-xr-x	November 29, 2017 08:03 AM
		four		hdfs	supergroup	drwxr-xr-x	November 29, 2017 08:03 AM
		ten		hdfs	supergroup	drwxr-xr-x	November 29, 2017 08:03 AM

Show 45 of 3 items   Page 1 of 1

⚠ Warning!

Exception if folder already exists

```
play_part.coalesce(1).
  saveAsTextFile("/user/cloudera/tests/play_part/coalesce")
```

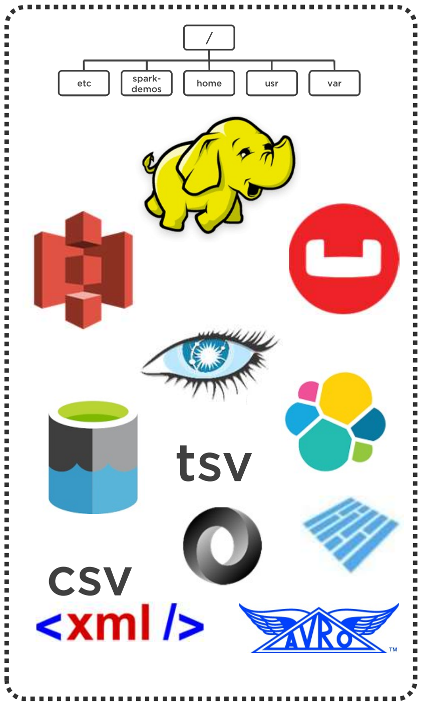
Folder Exists

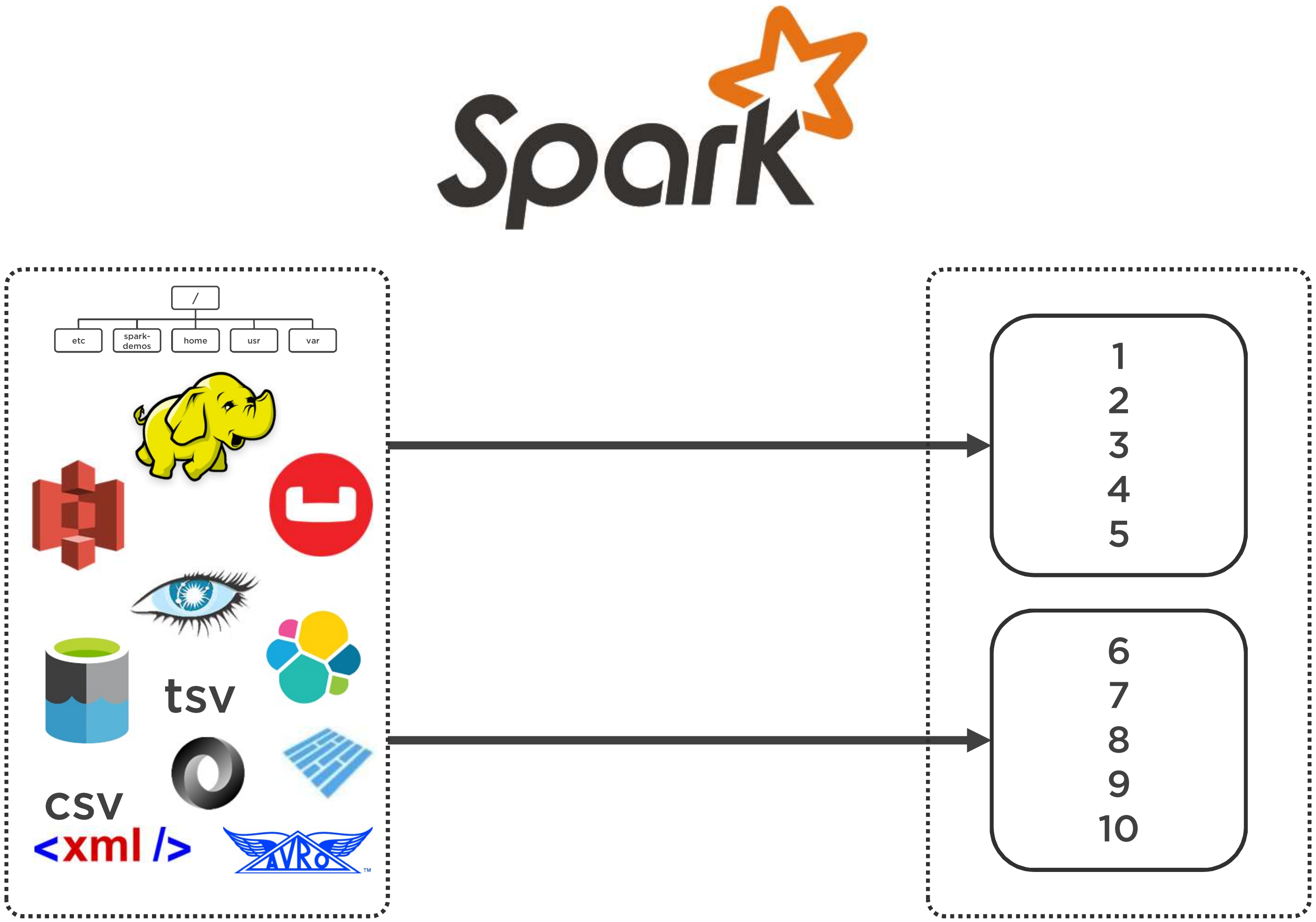
If folder exists, you get an exception

This applies for lower level API

Works different for higher level API

Spark




Spark
/
etc
spark-demos
home
usr
var
tsv
csv
<xml />
AVRO
1
2
3
4
5
6
7
8
9
10

```
sc.textFile("/user/cloudera/tests/play_part/ten").count()
sc.textFile("/user/cloudera/tests/play_part/four").count()
sc.textFile("/user/cloudera/tests/play_part/coalesce").count()
sc.textFile("/user/cloudera/tests/play_part/four/part-00000").count()
```

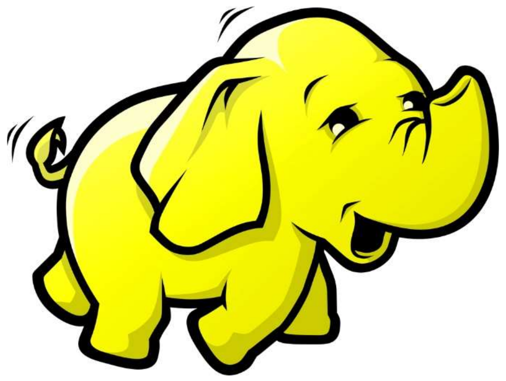
Loading Text from HDFS

Use textFile(), using the SparkContext

Use hdfs:// with a different cluster manager

All files in the folder, or specific file

HDFS



Hadoop Distributed File System

Primary storage system for Hadoop

Data replication

Blocks

Optimized for parallel data processing

```
val local_play =
  sc.textFile("file:///stackexchange/play_part/")
local_play.count
local_play.take(10)
```

Loading Local Data

Access local data using file:///

File needs to be in all nodes

Not required in standalone Spark

```
cd posts

sbt package

spark2-submit --class "PreparePostsCSVApp"
              target/scala-2.11/posts-project_2.11-1.0.jar
```

Convert Posts.xml to CSV

Data preparation step

```
posts_all =
sc.textFile("/user/cloudera/stackexchange/posts_all_csv")
posts_all.count()
```

Loading StackExchange / StackOverflow Posts

Load from HDFS

How many do we have?

CSV

Comma Separated Values

Plain text file

Tabular data

Supports different separators

- Tab

```
cd badges

sbt package

spark2-submit --class "PrepareBadgesCSVApp"
              target/scala-2.11/badges-project_2.11-1.0.jar
```

Convert Badges.xml to CSV

Data preparation step

```
def split_the_line(x: String): Array[String] = x.split(",")
val badges_rdd_csv =
  sc.textFile("/user/cloudera/stackexchange/badges_csv")
badges_rdd_csv.take(1)
badges_rdd_csv.take(1)(0)
val badges_columns_rdd =
  badges_rdd_csv.map(split_the_line)
badges_columns_rdd.take(1)(0)(2)
```

Loading CSV

Loaded like any other file, with textFile()

You get lines, and tell Spark what to do with the data

Advantage and drawback at the same time

```
val numbers_partitions =
  sc.wholeTextFiles("/user/cloudera/tests/play_part/four")

numbers_partitions.take(1)
numbers_partitions.take(1)(0)._1
```

Loading Whole Text Files

Loads each file into a row, uses wholeTextFiles()

In a PairRDD

Key is file name, value is the file

JSON

JavaScript Object Notation

Light weight data format

Easy to use

Collection of name value pairs and arrays

Fat-free XML

```
cd tags

sbt package

spark2-submit --class "PrepareTagsJSON"
              target/scala-2.11/tags-project_2.11-1.0.jar
```

Convert Tags.xml to JSON

Data preparation step

```
val tags_json =
  sc.textFile("/user/cloudera/stackexchange/tags_json")
tags_json.first()
```

Load JSON

Just like CSV, it loads as text

Parse each record

Is there an easy way to parse JSON with RDDs?

S3

Simple Storage Service

Amazon's cloud storage

Buckets

Rock solid SLA

Unlimited scalability

At extremely reasonable cost

Works well with many technologies

Data preparation step

Prepare Data for Processing in S3

```
Manually upload a JSON file into an S3 account
Get the full path in S3
Create an IAM user, enable programmatic access
Obtain access key id and secret access key
Set keys in environment
Load necessary packages
```

```
# export AWS_ACCESS_KEY_ID="access-key"
# export AWS_SECRET_ACCESS_KEY="secret-key"

val tags_json_s3 = sc.textFile("s3a://pscs/part-00000")
tags_json_s3.take(10)
```

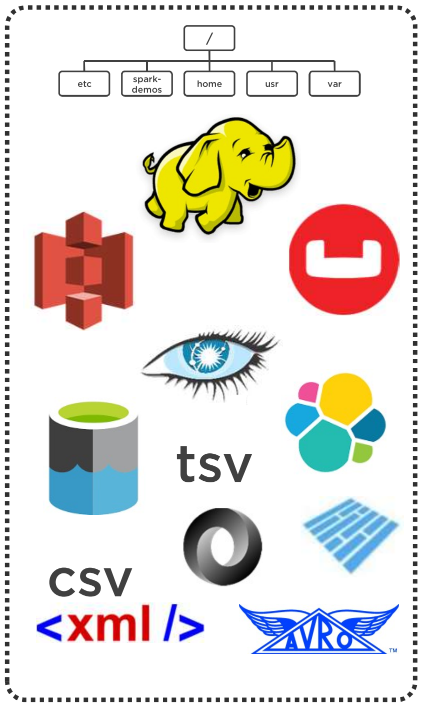
Accessing Amazon S3 with Spark

Transparent, still textFile()

Use s3a://

Be very careful with security

Spark



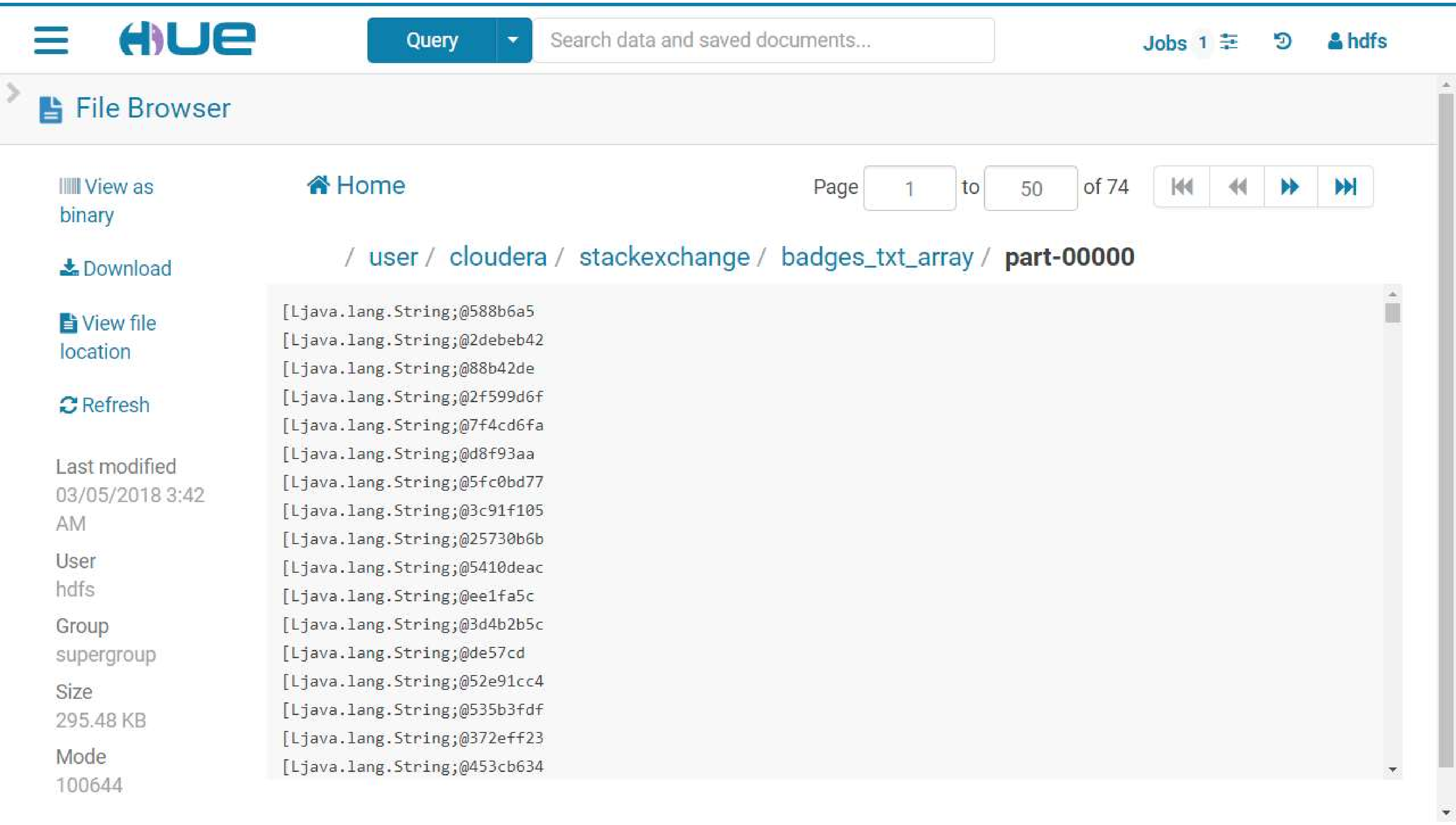
```
badges_columns_rdd.take(1)
badges_columns_rdd.take(1)(0)(2)
badges_columns_rdd.saveAsTextFile("/user/cloudera/
  stackexchange/badges_txt_array")
```

Saving an RDD

We previously saved an RDD that had no transformations applied

- Partition files

Save an RDD with columns as text



/ user / cloudera / stackexchange / badges_txt_array / part-00000

Last modified
03/05/2018 3:42 AM
User
hdfs
Group
supergroup
Size
295.48 KB
Mode
100644

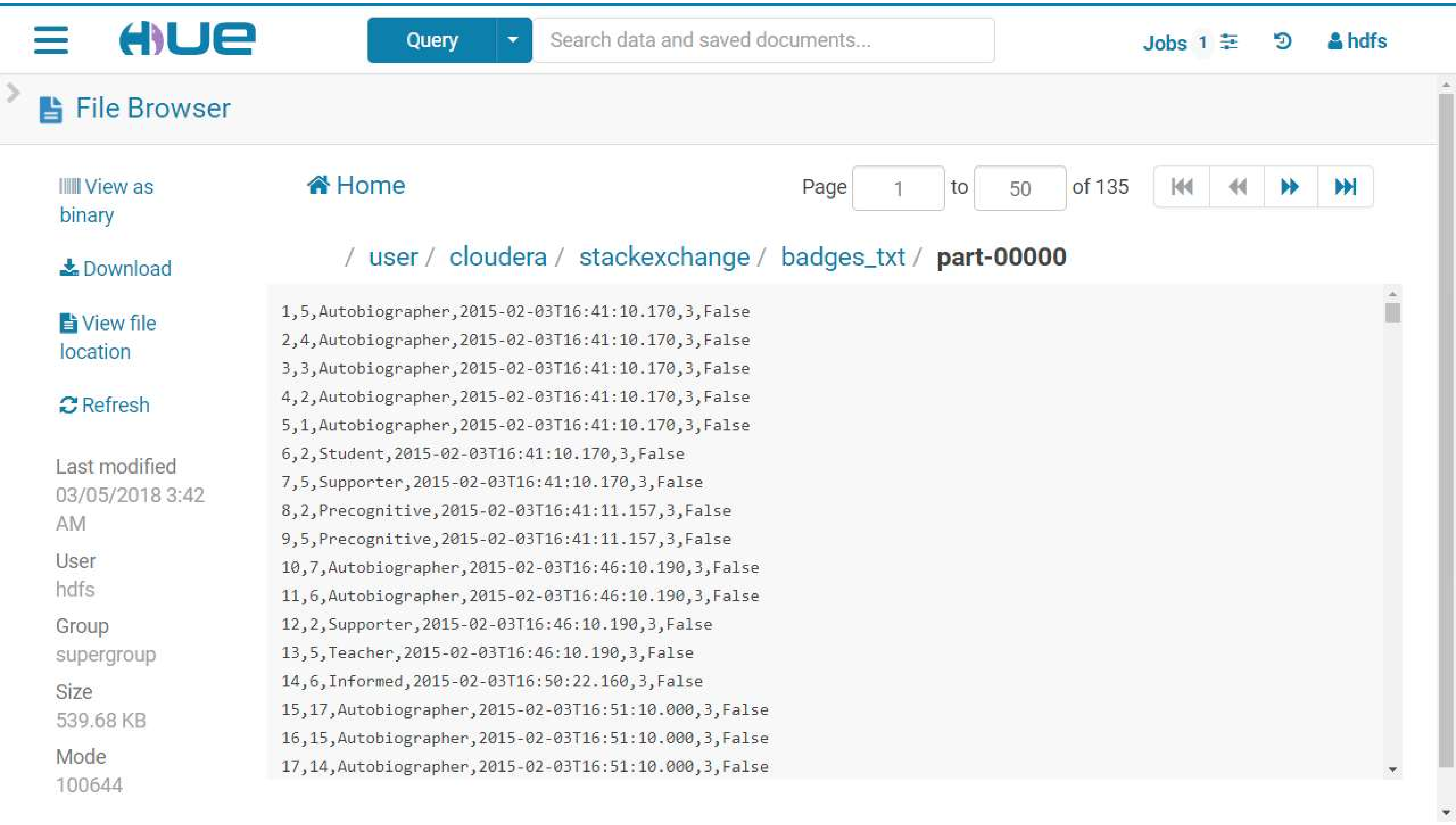
```
[Ljava.lang.String;@588b6a5
[Ljava.lang.String;@2debeb42
[Ljava.lang.String;@88b42de
[Ljava.lang.String;@2f599d6f
[Ljava.lang.String;@7f4cd6fa
[Ljava.lang.String;@d8f93aa
[Ljava.lang.String;@5fc0bd77
[Ljava.lang.String;@3c91f105
[Ljava.lang.String;@25730b6b
[Ljava.lang.String;@5410deac
[Ljava.lang.String;@ee1fa5c
[Ljava.lang.String;@3d4b2b5c
[Ljava.lang.String;@de57cd
[Ljava.lang.String;@52e91cc4
[Ljava.lang.String;@535b3fdf
[Ljava.lang.String;@372eff23
[Ljava.lang.String;@453cb634
```

```
badges_columns_rdd.map(x => x.mkString(","))
  .saveAsTextFile("/user/cloudera/stackexchange/badges_txt")
```

Saving an RDD

Save as text

Convert Array[String] to String



/ user / cloudera / stackexchange / badges_txt / part-00000

Last modified
03/05/2018 3:42 AM
User
hdfs
Group
supergroup
Size
539.68 KB
Mode
100644

```
1,5,Autobiographer,2015-02-03T16:41:10.170,3,False
2,4,Autobiographer,2015-02-03T16:41:10.170,3,False
3,3,Autobiographer,2015-02-03T16:41:10.170,3,False
4,2,Autobiographer,2015-02-03T16:41:10.170,3,False
5,1,Autobiographer,2015-02-03T16:41:10.170,3,False
6,2,Student,2015-02-03T16:41:10.170,3,False
7,5,Supporter,2015-02-03T16:41:10.170,3,False
8,2,Precognitive,2015-02-03T16:41:11.157,3,False
9,5,Precognitive,2015-02-03T16:41:11.157,3,False
10,7,Autobiographer,2015-02-03T16:46:10.190,3,False
11,6,Autobiographer,2015-02-03T16:46:10.190,3,False
12,2,Supporter,2015-02-03T16:46:10.190,3,False
13,5,Teacher,2015-02-03T16:46:10.190,3,False
14,6,Informed,2015-02-03T16:50:22.160,3,False
15,17,Autobiographer,2015-02-03T16:51:10.000,3,False
16,15,Autobiographer,2015-02-03T16:51:10.000,3,False
17,14,Autobiographer,2015-02-03T16:51:10.000,3,False
```

```
val badges_reloaded =
sc.textFile("/user/cloudera/stackexchange/badges_txt")
```

Saving & Reloading an RDD

Reload our data

What happened when I save?

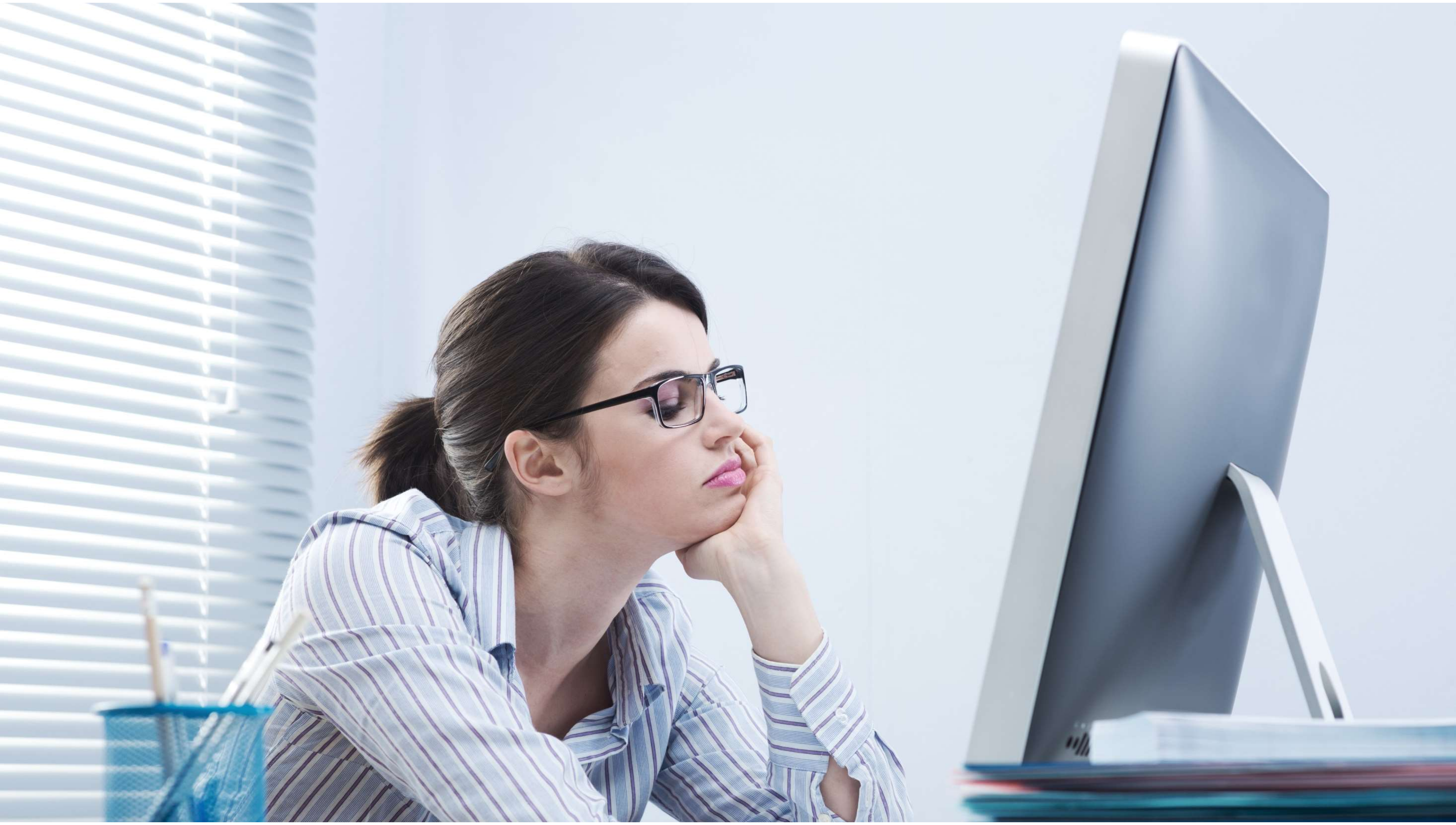
```
badges_reloaded.take(1)

badges_reloaded.take(1)(0)

badges_reloaded.take(1)(0)(0)
```

Text Is Good but Not Great

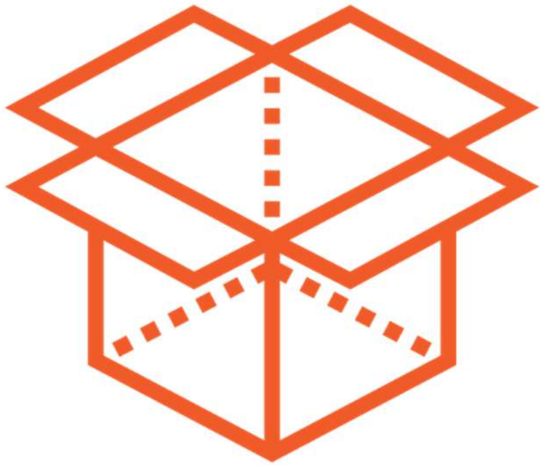
Compare original RDD vs. the one we just reloaded

We lost the format we applied

Text has no schema



Object File



Save an RDD in a simple format

- ObjectFile

SequenceFile of serialized Java objects

Not the most efficient way

- Easy to use
- Still marked as experimental

```
badges_columns_rdd. //hit tab

badges_columns_rdd.saveAsObjectFile("/user/cloudera/
  stackexchange/badges_object")

val badges_object =
sc.objectFile[Array[String]]("/user/cloudera/stackexchange/
  badges_object")

badges_object.take(1)
```

ObjectFile

Check available save methods

Save with saveAsObjectFile

And reload with objectFile() with type

SequenceFile

Flat file

Binary format

Key/value pairs

Used extensively in Hadoop

```
tuple_rdd.saveAsSequenceFile("/user/cloudera/stackexchange/
  tuple_sequence")


import org.apache.hadoop.io.Text

import org.apache.hadoop.io.LongWritable

sc.sequenceFile("/user/cloudera/stackexchange/tuple_sequence",
  classOf[Text], classOf[Longritable]).collect()
```

SequenceFile

Save using saveAsSequenceFile()

Data needs to be in key/value format

Read with sequenceFile(), specify types

Hadoop Formats

InputFormat

- How to read input files
- How they are split

OutputFormat

- How to write output files

Very flexible

```
badges_columns_rdd.map(prepareForNAH)
  .saveAsNewAPIHadoopFile("/user/cloudera/stackexchange/
   badgess_newapihadoop",
   classOf[Text], classOf[Text],
   classOf[SequenceFileOutputFormat[Text, Text]])
```

Hadoop OutputFormat

Write using saveAsNewAPIHadopFile()

More parameters required than other methods

- i.e. file output format, key class, value class

Hadoop OutputFormat

```
import org.apache.hadoop.mapreduce.lib.output._
import org.apache.hadoop.mapreduce.lib.input._
import org.apache.hadoop.io._

val prepareForNAH: Array[String] => (Text, Text) = x =>
(new Text(x(0)), new Text(x(2)))
badges_columns_rdd.map(prepareForNAH)
  .saveAsNewAPIHadoopFile("/user/cloudera/stackexchange/
    badgess_newapihadoop",
  classOf[Text],
  classOf[Text],
  classOf[SequenceFileOutputFormat[Text, Text]])
```

```
val badges_newapihadoop =
  sc.newAPIHadoopFile("/user/cloudera/stackexchange/
    badgess_newapihadoop",
  classOf[SequenceFileInputFormat[Text, Text]],
  classOf[Text],
  classOf[Text])

badges_newapihadoop.take(1)
```

Hadoop InputFormat

Read with newAPIHadopFile()

Specify necessary parameters

Great deal of control

```
val xmlPosts =
 sc.newAPIHadoopFile("/user/cloudera/stackexchange/Posts.xml",
 classOf[TextInputFormat],
 classOf[LongWritable],
 classOf[Text])
```

Does newAPIHadoopFile Look Familiar?

We used to load the StackExchange dumps

Using spark-xml package

Load

textFile

sequenceFile

objectFile

hadoopFile

hadoopDataset

newAPIHadoopFile

newAPIHdoopDataset

saveAs*

saveAsTextFile

saveAsSequenceFile

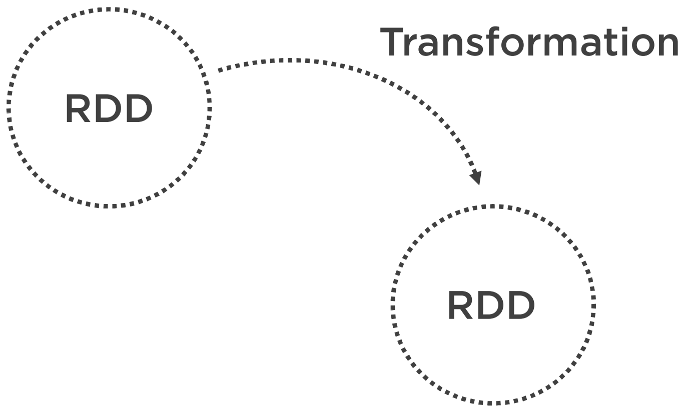
saveAsObjectFile

saveAsHadoopFile

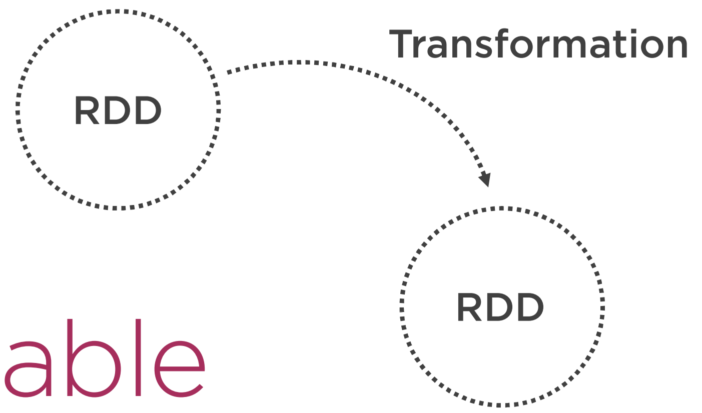
saveAsHadoopDataset

saveAsNewAPIHadoopFile

saveAsNewAPIHdoopDataset



Creating RDDs with Transformations



RDDs are Immutable

Full stop... nothing else to say...

```
val rdd_reuse = sc.parallelize(Array(1,2))

rdd_reuse.collect()

val rdd_reuse = sc.parallelize(Array(3,4))

rdd_reuse.collect()
```

But... (On Immutable)

Immutable

- Can't change the data

Reuse the variable

```
["1,5,Autobiographer,2015-02-03T00:00:00.000Z,3,False"]
```

map(split_the_line)

```
["1", "5", "Autobiographer", "2015-02-03T00:00:00.000Z", "3", "False"]
```

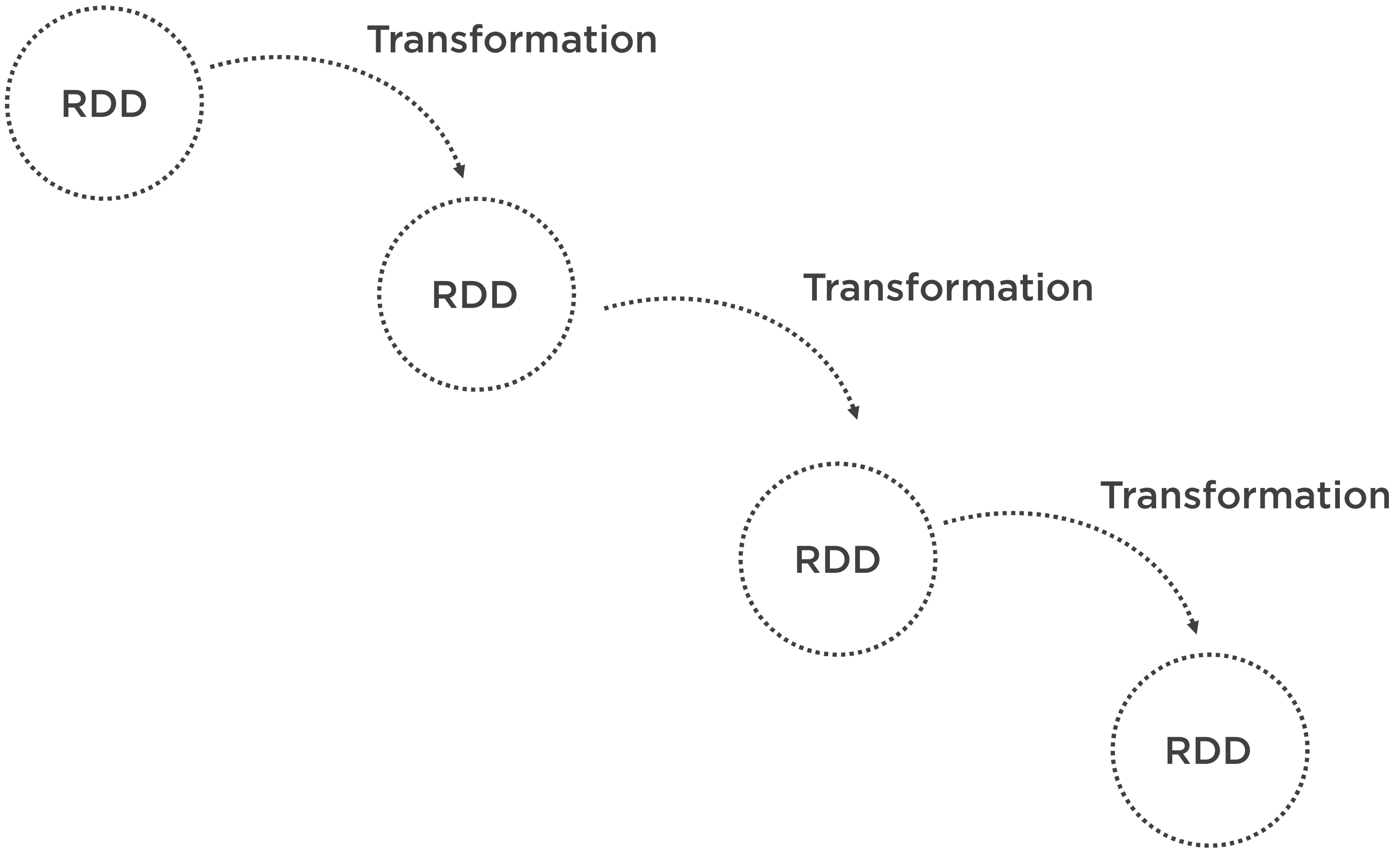
```
badges_rdd_csv.take(1)

split_the_line

badges_columns_rdd.take(1)
```

Creating RDDs with Transformations

Loaded Badges CSV into an RDD

- Single element per row

Applied map(), with split_the_line() function


RDD
Transformation
RDD
Transformation
RDD
Transformation
RDD

```
val badges_entry = badges_columns_rdd.map(x => x(2))
val badges_name = badges_entry.map(x => (x, 1))

val badges_reduced = badges_name.reduceByKey(_ + _)

val badges_count_badge = badges_reduced.
  map({ case (x,y) => (y,x) })

val badges_sorted = badges_count_badge.sortByKey(false).
  map({ case (x, y) => (y, x) })
```

Creating Multiple RDDs with Transformations

One transformation

Iterate

- Applying multiple transformations
- One of the main objectives of Spark

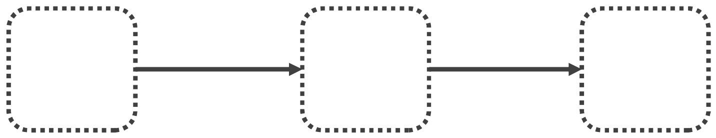
```
$intp.definedTerms.
map(defTerms => s"${defTerms.toTermName}:
${$intp.typeOfTerm(defTerms.toTermName.toString)}").
filter(x => x.contains("()org.apache.spark.rdd.RDD"))
.foreach(println)
```

Tip: Listing RDDs

Check for variables in memory with $intp

Look for those that are an instance of RDD

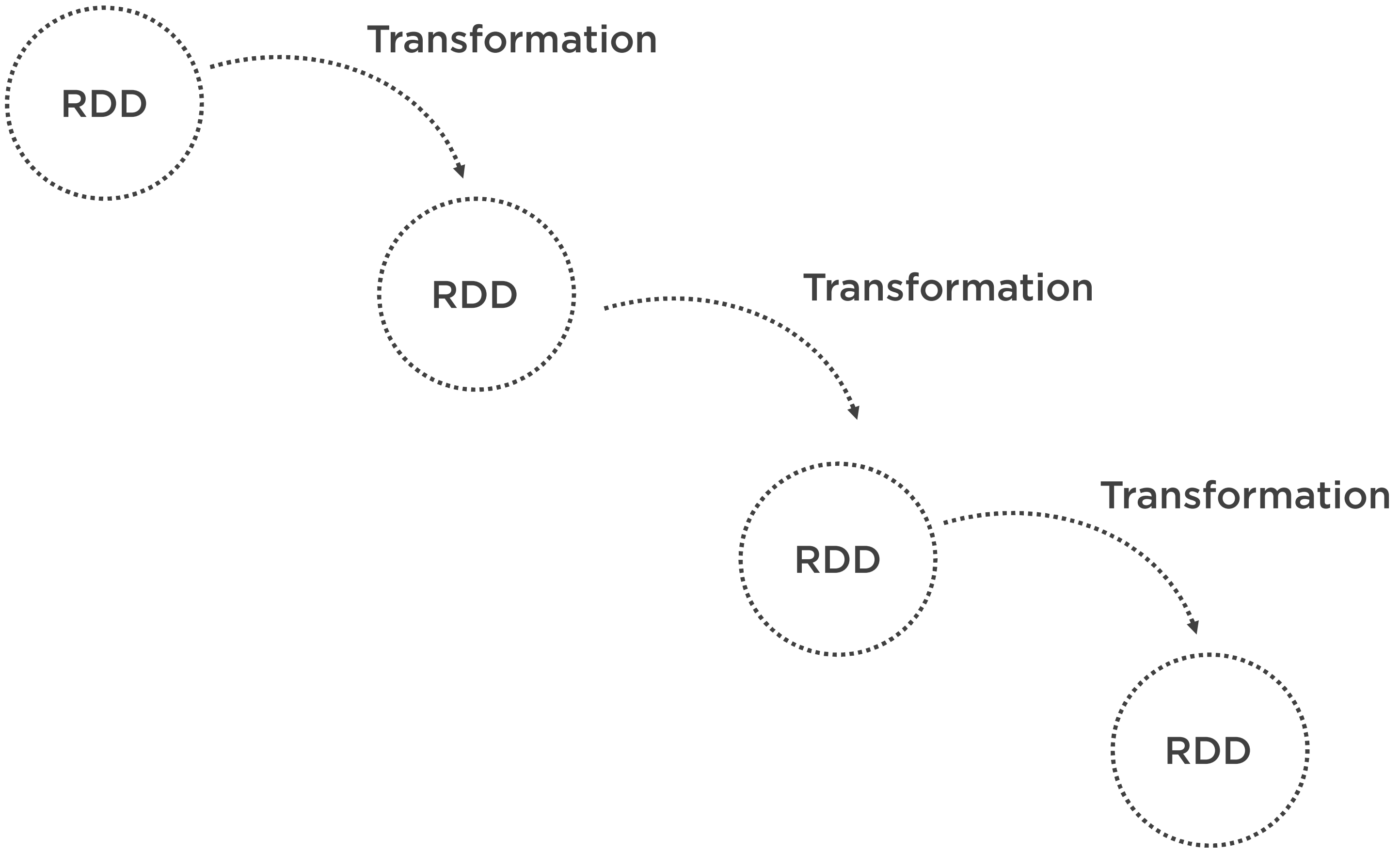
- Use filter



RDD Lineage

Graph of transformation operations required to execute when an action is called

RDD operator graph or RDD dependency graph

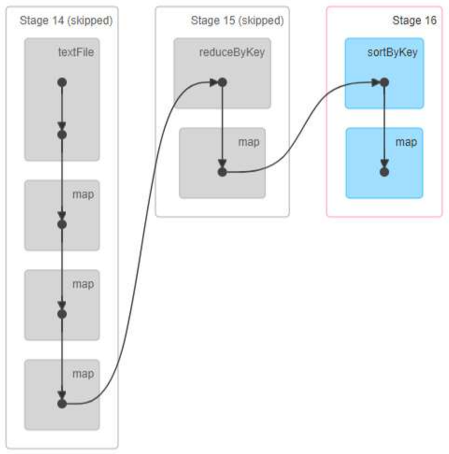

RDD
Transformation
RDD
Transformation
RDD
Transformation
RDD

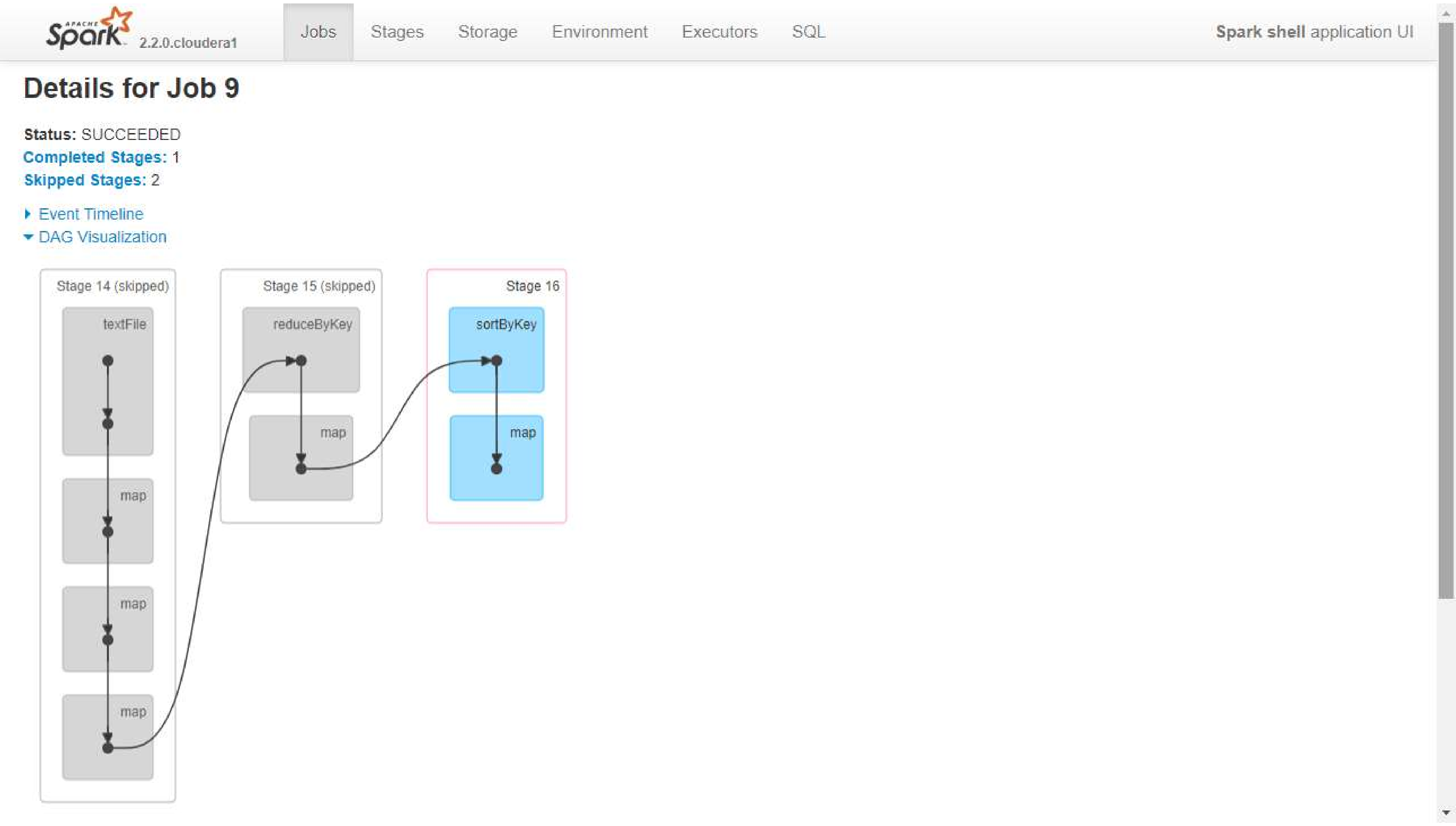
```
badges_sorted

badges_sorted.toDebugString
```

A Little Bit More on Lineage

Possible to see lineage

Use toDebugString()





Details for Job 9

Status: SUCCEEDED
Completed Stages: 1
Skipped Stages: 2

▸ Event Timeline
▾ DAG Visualization

```
badges_sorted.toDebugString

badges_reduced.dependencies
```

Dependencies

You *could* check RDD dependencies

At least to understand better the transformations

Takeaway

Why learn RDDs?

- Better foundation
- Right tool for the job
- Existing code base

Takeaway

SparkContext

One context per application

RDD main abstraction

Multiple types of RDDs

Takeaway

Create RDDs

- Data in memory
- Files in storage
- Other RDDs

Different data sources and file formats

- Text, CSV, Parquet, JSON, Sequence Files among others
- Transparent

Takeaway

Return data back to the Driver

Persist to disk

- Some formats store schema
- Others don't

RDD Lineage

- Dependencies