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

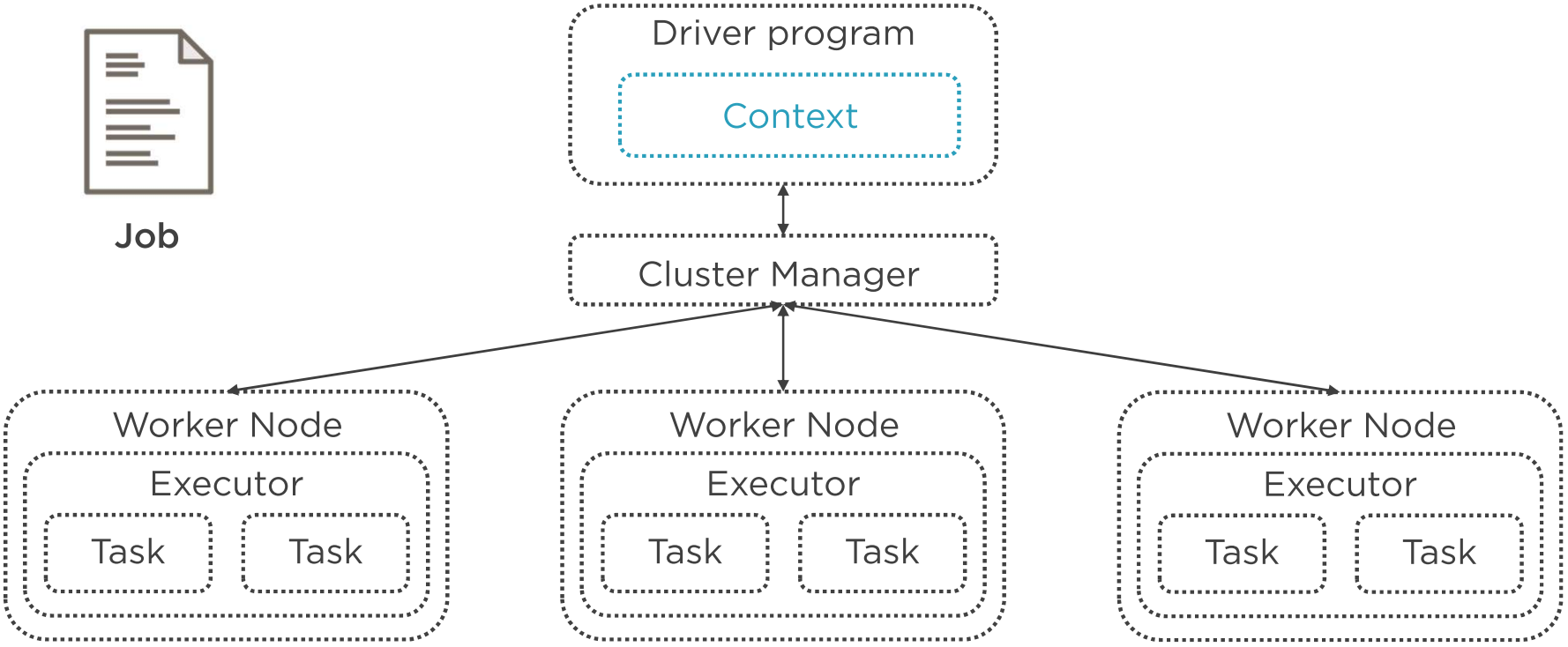
Can I use Hive from HUE

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

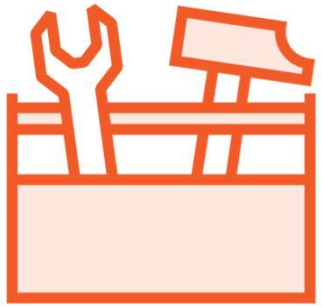
Spark Application

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

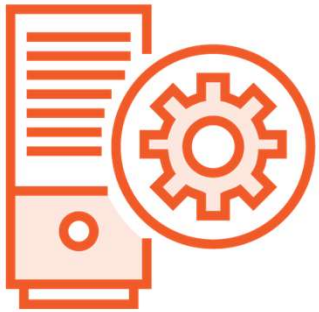
Only one SparkContext

SparkContext

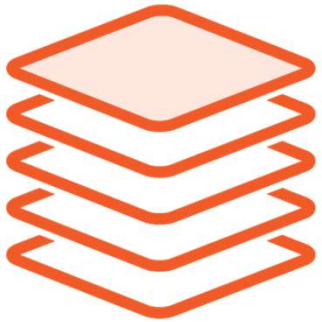
Spark Application



Configuration & Environment



Services



Create RDDs

```
sc
dir(sc)
sc.serializer
sc.sparkUser
help(sc) sc.sparkUser
sc.sparkUser()
sc.appName
sc.applicationId
```

SparkContext

SparkContext available as sc

Many methods and properties available

Get configuration and environment information

```
sc.stop()
test_rdd = sc.emptyRDD()
sc =
SparkContext.getOrCreate()
```

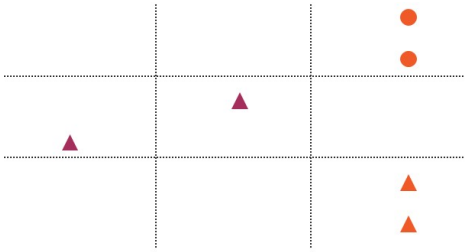
SparkContext

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

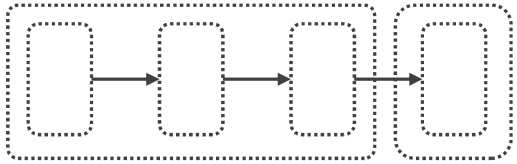
What is an RDD and Schema RDD

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

Click to add image

Click to add image

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

```
 * 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

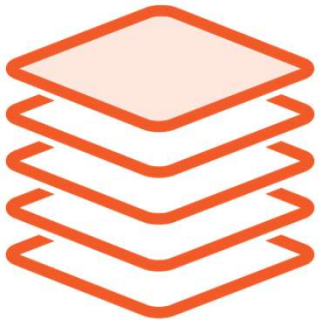
Contain tuples

Useful for grouping or aggregating

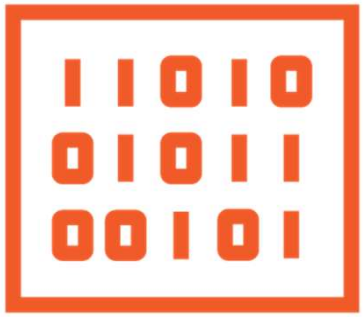
Can use RDD transformations

PairRDD specific transformations available

Creating RDDS



Parallelize



External Data

From Another RDD

* or DataFrame (next module)

```
[1,2,3,4,5]
type[1,2,3,4,5]
list_one_to_five =sc.parallelize([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()
list_one_to_five = sc.parallelize([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

```
dir(list_one_to_five)

list(list_one_to_five)
```

RDD Functionality

Use dir() to inspect attributes

Even better, use help()

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

An Operation on an RDD

What can I do on an RDD?

Call sum() to add up values

And first() to get first element

```
list_dif_types = sc.parallelize([False, 1, "two", {'three': 3},
('xavier', 4)])

list_dif_types.collect()

tuple_rdd = sc.parallelize([('xavier', 1),('irene', 2)])

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

```
empty_rdd=sc.parallelize([])
empty_rdd.isEmpty()
empty_rdd=sc.emptyRDD()
empty_rdd.isEmpty()
```

EmptyRDD

You can create empty RDDs

With parallelize()

Or emptyRDD()

```
bigger_rdd = sc.parallelize(range(1,1000))
bigger_rdd.collect()
bigger_rdd.sum()
bigger_rdd.count()
bigger_rdd.getNumPartitions()
```

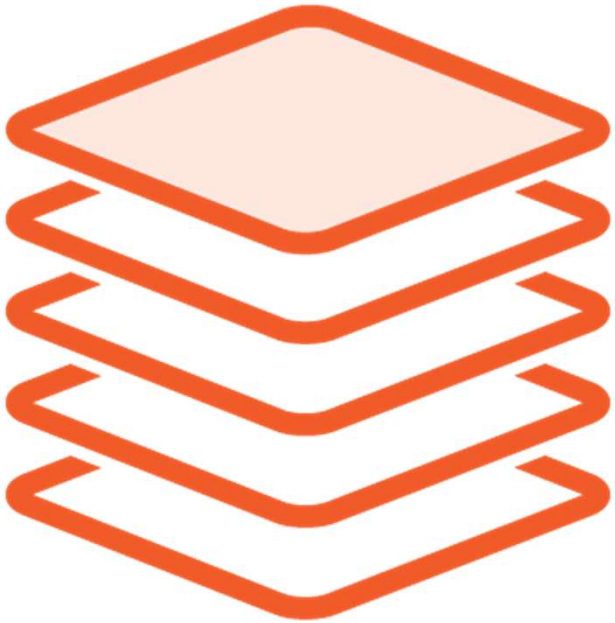
Ranges

Create RDDs with range()

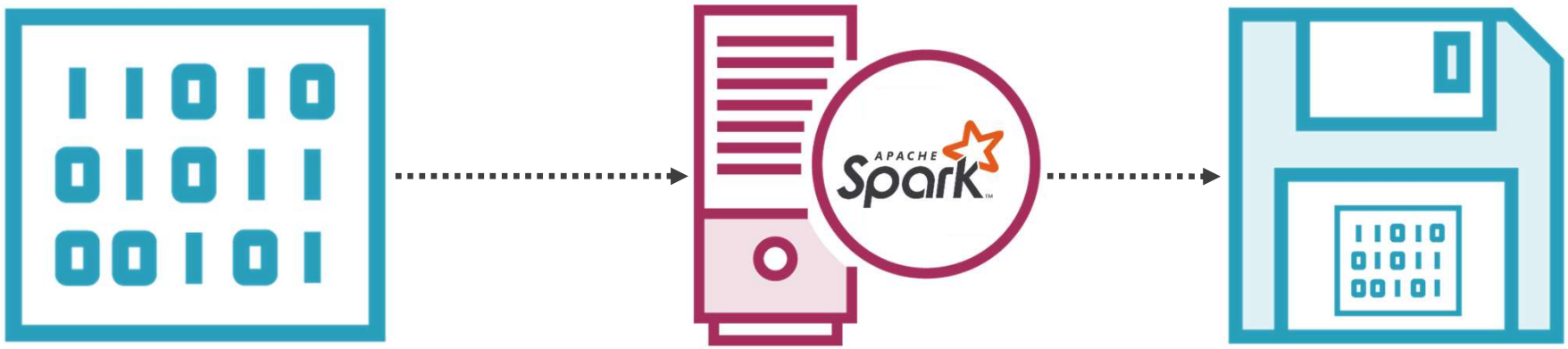
And use methods to get information from our RDD data

Count() all elements in my rdd

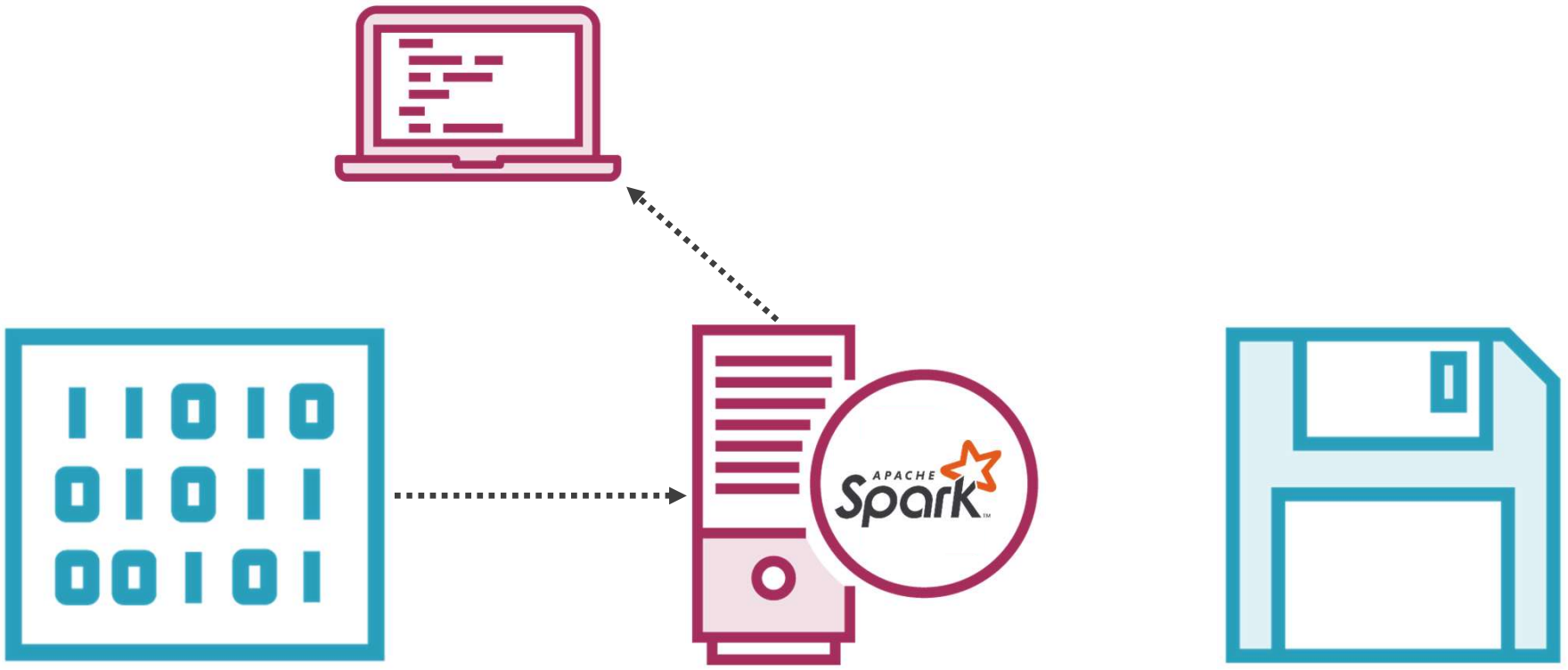
Parallelize



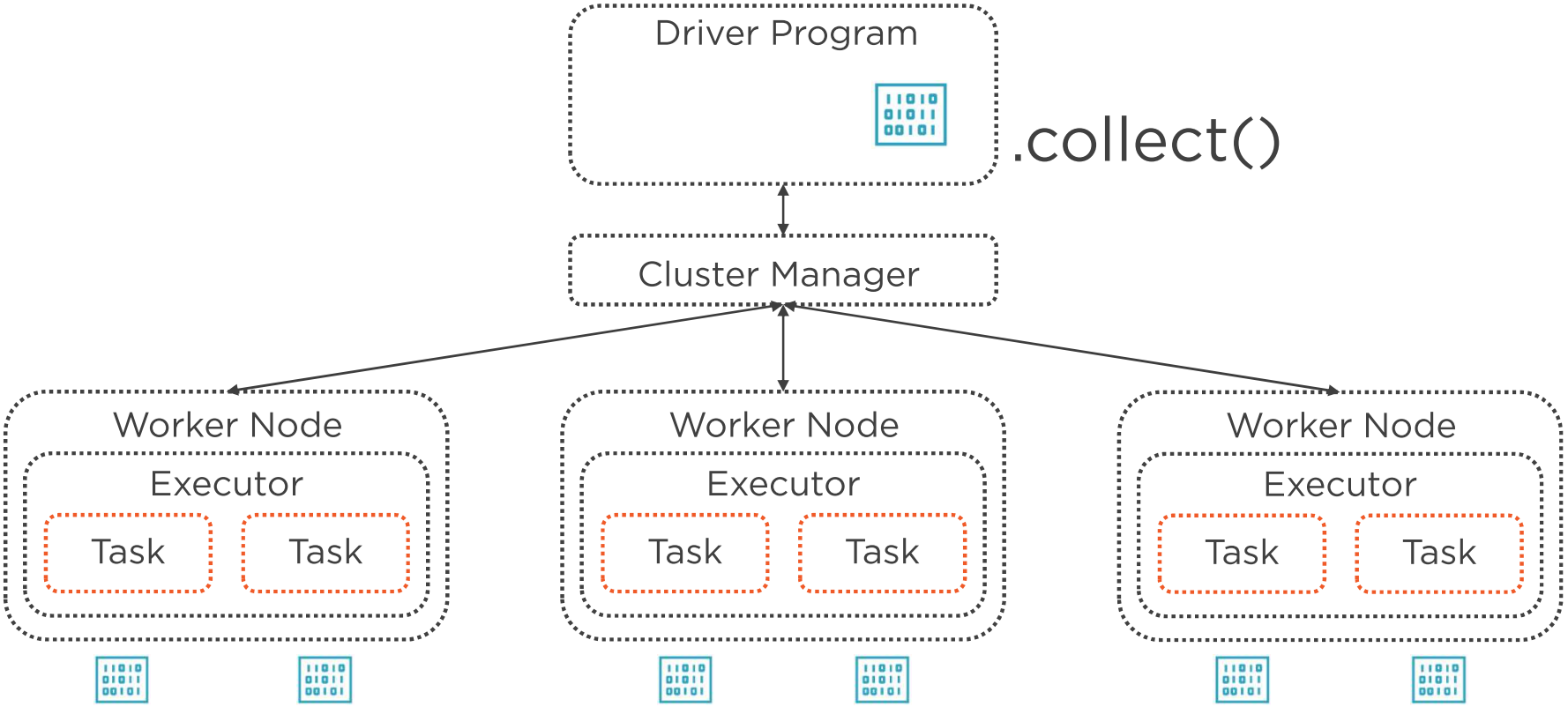
Returning Data to the Driver



Returning Data to the Driver



Returning Data to the Driver



```
list_bigger.collect()
list_bigger.take(10)
list_bigger.first()
help(list_bigger.takeOrdered)
list_bigger.takeOrdered(10, key=lambda x: -x)
tuple_map = tuple_rdd.collectAsMap()
type(tuple_map)
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!

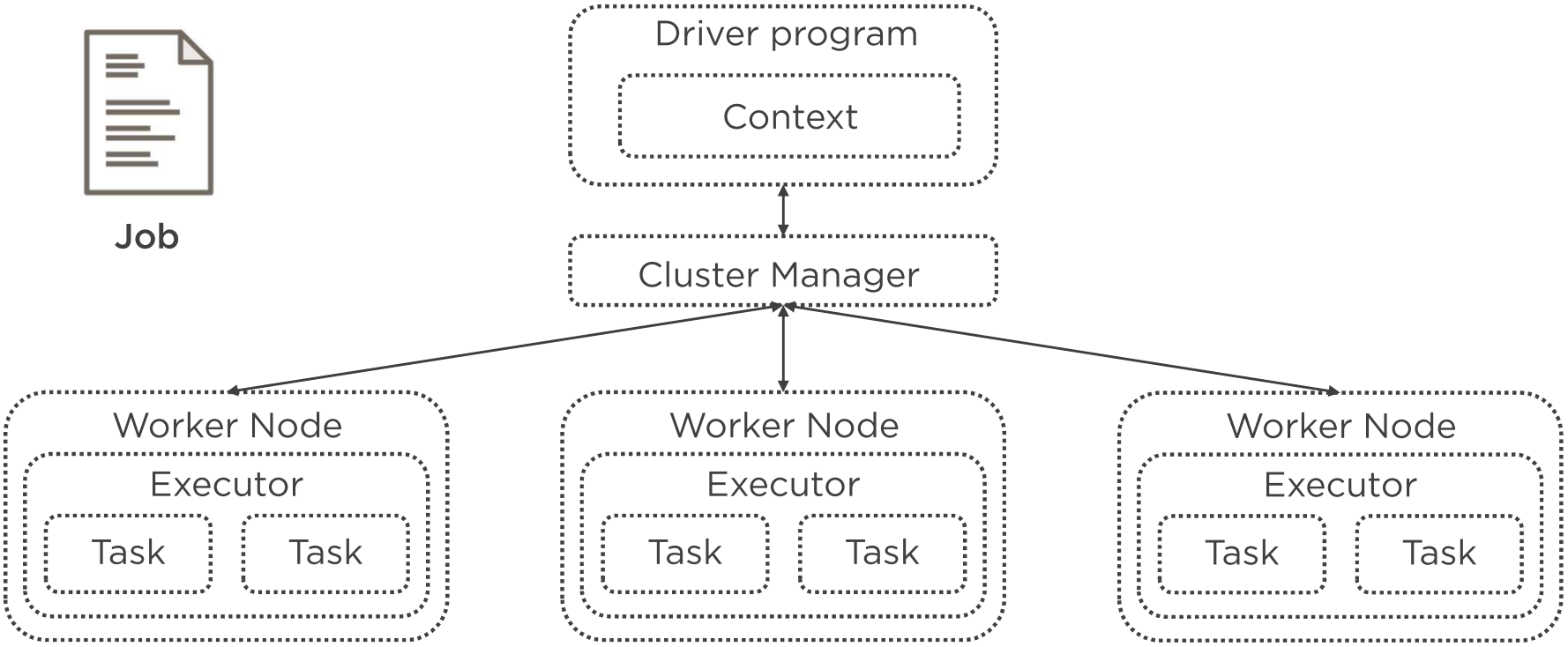
Potential out of memory exception on large datasets

```
for elem in list_bigger.take(10):
    print elem

for key, value in tuple_map.iteritems():
    print key + " / " + str(value)
```

Iterating & Printing Data

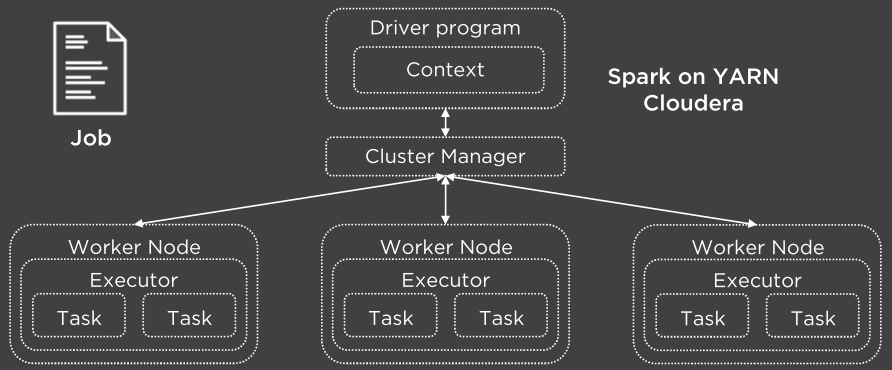
Iterate over list of data returned using take() in for loop

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

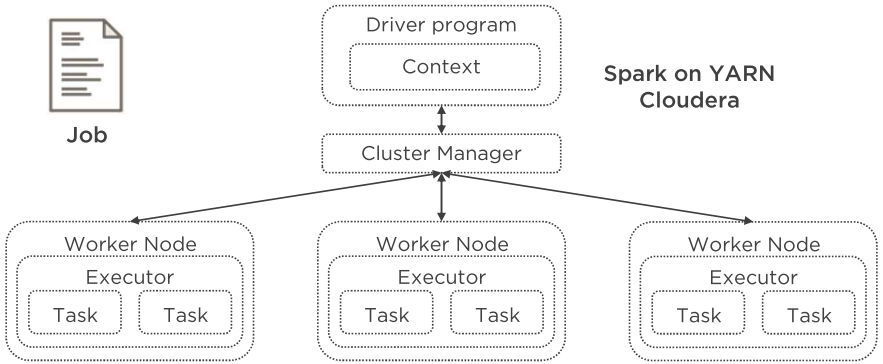
Driver

```
test_write =sc.parallelize(
           ['xavier','troy'])

print test_write.collect()
```



In executor

```
def in_executor(entry):
    print "*** inExecutorEntry ***"
    print entry[0]
    print type(entry)
    return 'printed:'+str(entry[0])
```




```
spark2-submit inExecutor EntryLogLevelAll.py
```

```
import urllib2
def call_search(q):
response = urllib2.urlopen('http://tiny.bigdatainc.org/' + q)
html = response.read
topics = sc.parallelize(['ts1', 'ts2'])
topics.foreach(call_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

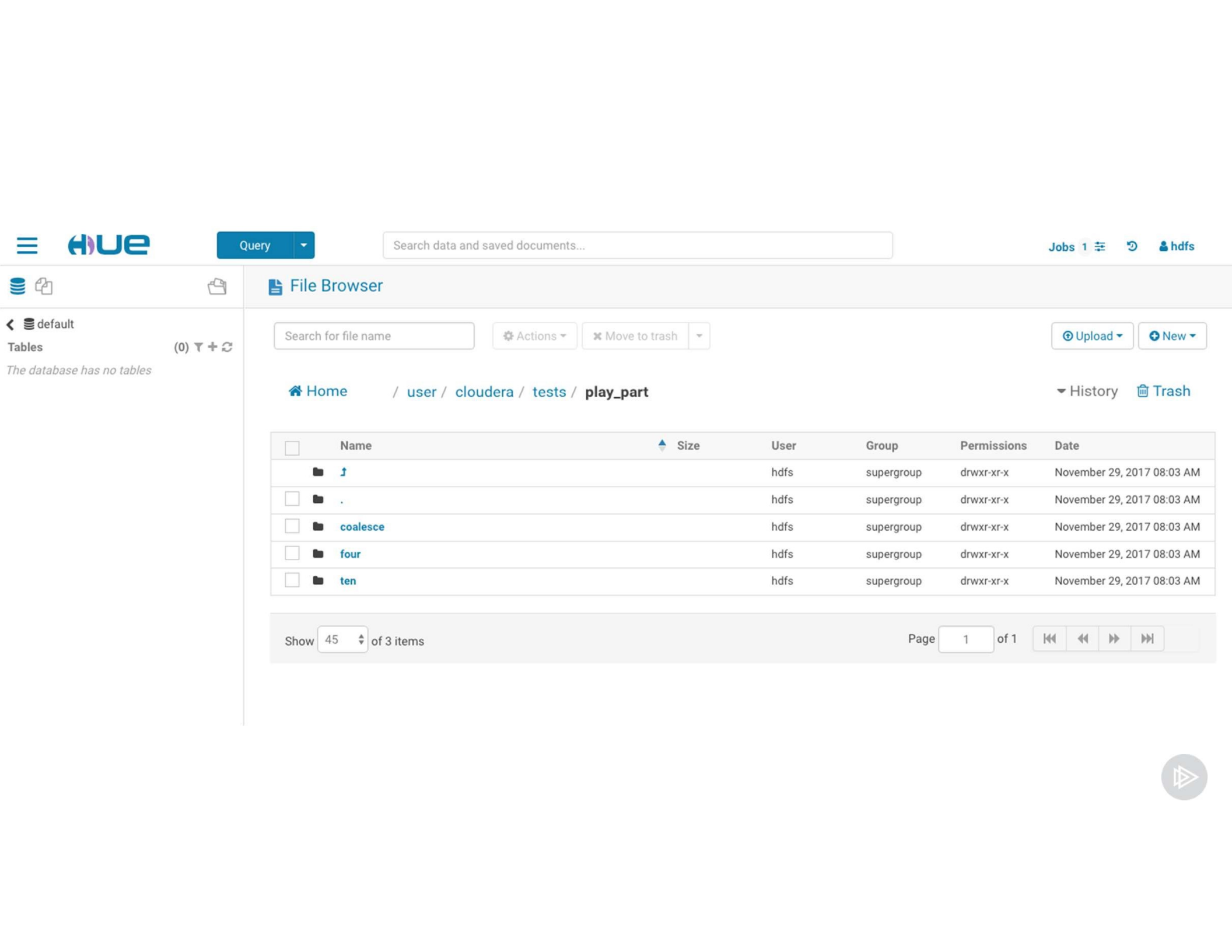
Data in RDD remains the same. How can we tell?

```
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


Query
Search data and saved documents...
Jobs 1
hdfs
File Browser
default
Tables (0)
The database has no tables
Search for file name
Actions
Move to trash
Upload
New
Home / user / cloudera / tests / play_part
History
Trash
Name
Size
User
Group
Permissions
Date
hdfs supergroup drwxr-xr-x November 29, 2017 08:03 AM
coalesce
four
ten
Show 45 of 3 items
Page 1 of 1

Warning!

Exception if folder already exists

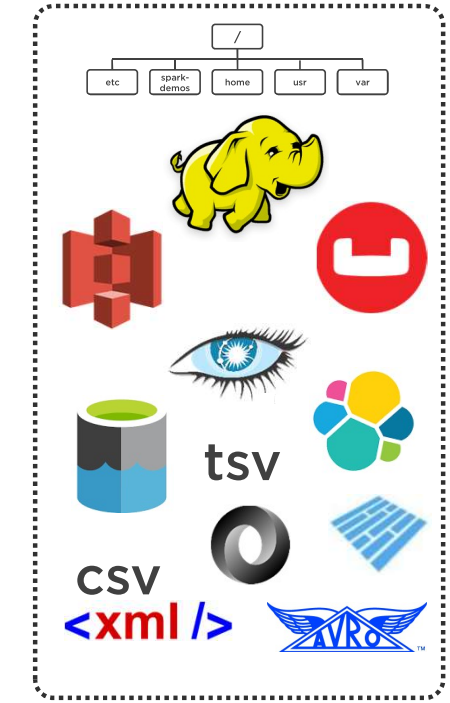
```
play_part.coalesce(1).saveAsTextFile("/user/cloudera/tests/
play_part/coalesce")
```

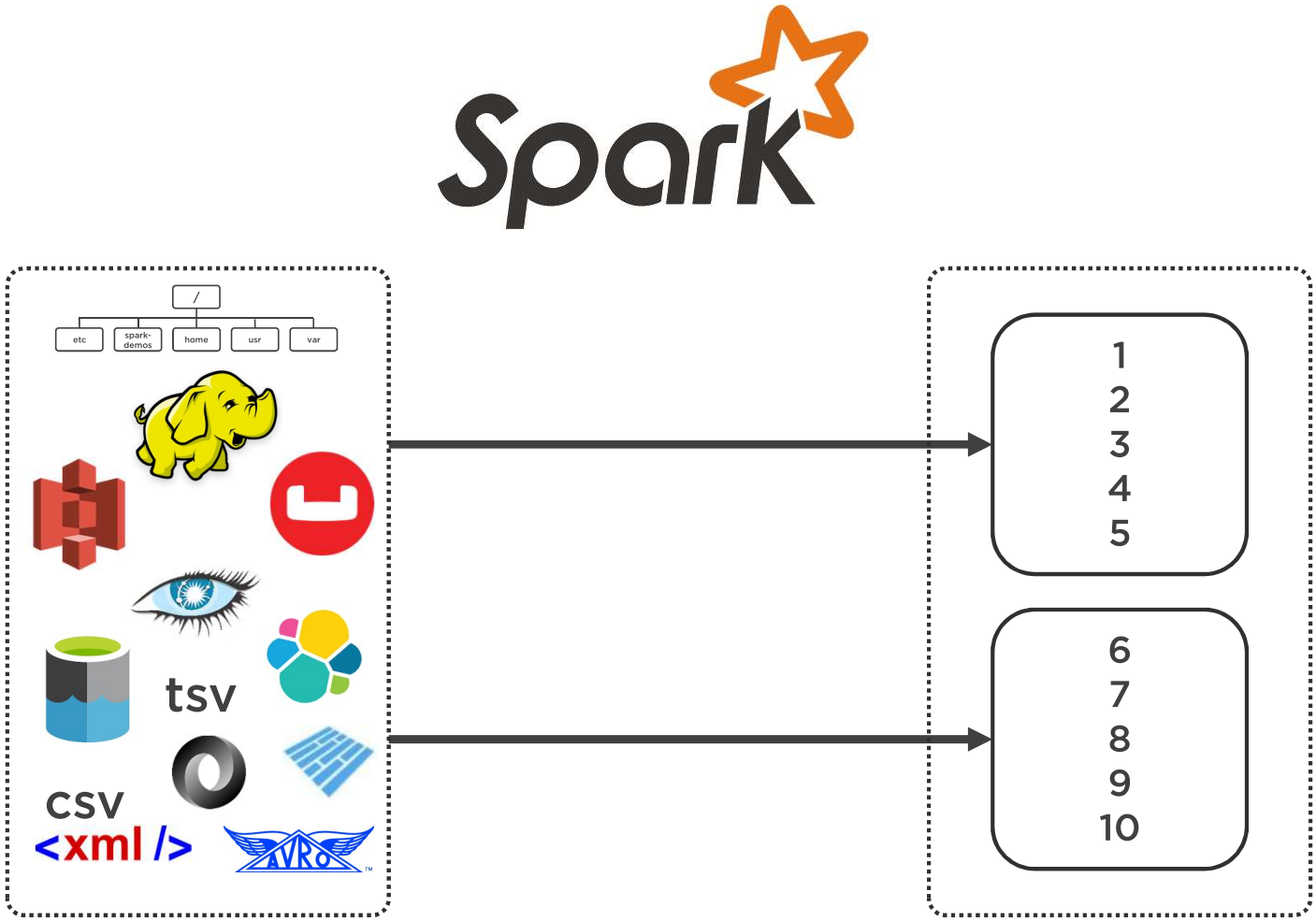
Folder Exists

If folder exists, you get an exception

This applies for lower level API

Works different for higher level API


/
etc
spark-demos
home
usr
var
tsv
csv
<xml />
AVRO


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/coalesce")
.count()

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/four/
part-00000").count()
```

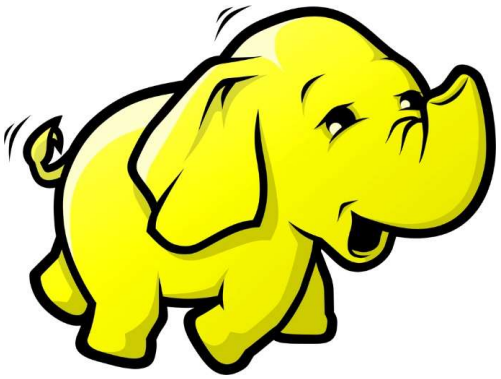
Loading Text from HDFS

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

```
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

```
spark2-submit --packages com.databricks:spark-xml_2.11:0.4.1,
                         com.databricks:spark-csv_2.11:1.5.0
                         prepare_data_posts_all_csv.py
```

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

```
spark2-submit --packages com.databricks:spark-xml_2.11:0.4.1,
                         com.databricks:spark-csv_2.11:1.5.0
                         prepare_data_badges_csv.py
```

Convert Badges.xml to CSV

Data preparation step

```
badges_rdd_csv =
sc.textFile("/user/cloudera/stackexchange/badges_csv")

badges_rdd_csv.take(1)

badges_rdd_csv.take(1)[0]

split_the_line

badges_rdd_csv.map(split_the_line).take(1)
```

```
def split_the_line(x):
    return x.split(',')
```

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

```
tags_partitions=sc.wholeTextFiles("/user/cloudera/stackexchange/badges_csv")

tags_partitions.take(1)
numbers_partitions = sc.wholeTextFiles("/user/cloudera/tests/play_part/four")

numbers_partitions.take(1)

numbers_partitions.take(1)[0][0]

numbers_partitions.take(1)[0][1]
```

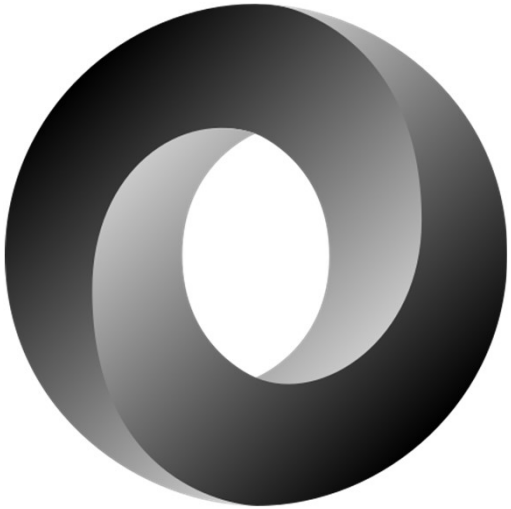
Loading Whole Text Files

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

```
spark2-submit --packages com.databricks:spark-xml_2.11:0.4.1,
                         com.databricks:spark-csv_2.11:1.5.0
                         prepare_data_tags_json.py
```

Convert Tags.xml to JSON

Data preparation step

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

tags_json.first()
```

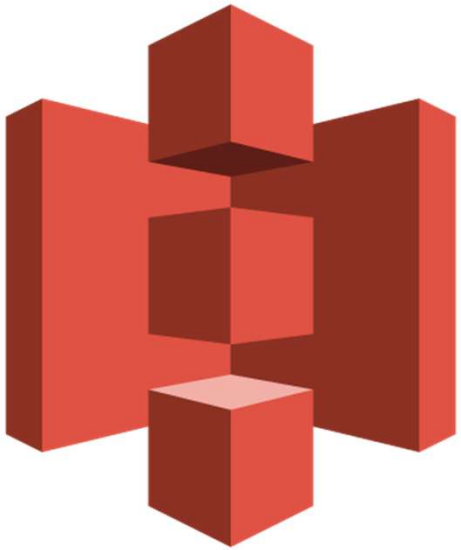
Load JSON

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

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

Prepare Data for Processing in S3

Data preparation step

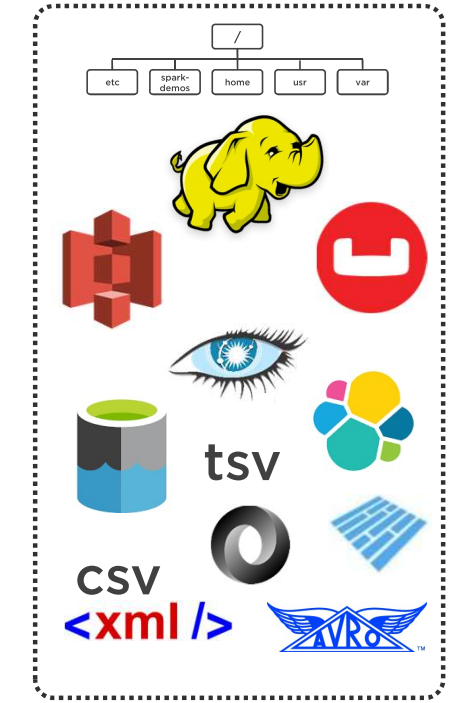
```
# export AWS_ACCESS_KEY_ID="access-key"

# export AWS_SECRET_ACCESS_KEY="secret-key"

tags_json_s3 = sc.textFile("s3a://pscs/tags_json/part-
00000.json")

tags_json_s3tags_json_s3.take(3)
```

Accessing Amazon S3 with Spark

Transparent, still textFile()

Use s3a://

Be very careful with security


/
etc
spark-demos
home
usr
var
tsv
csv
<xml />
AVRO

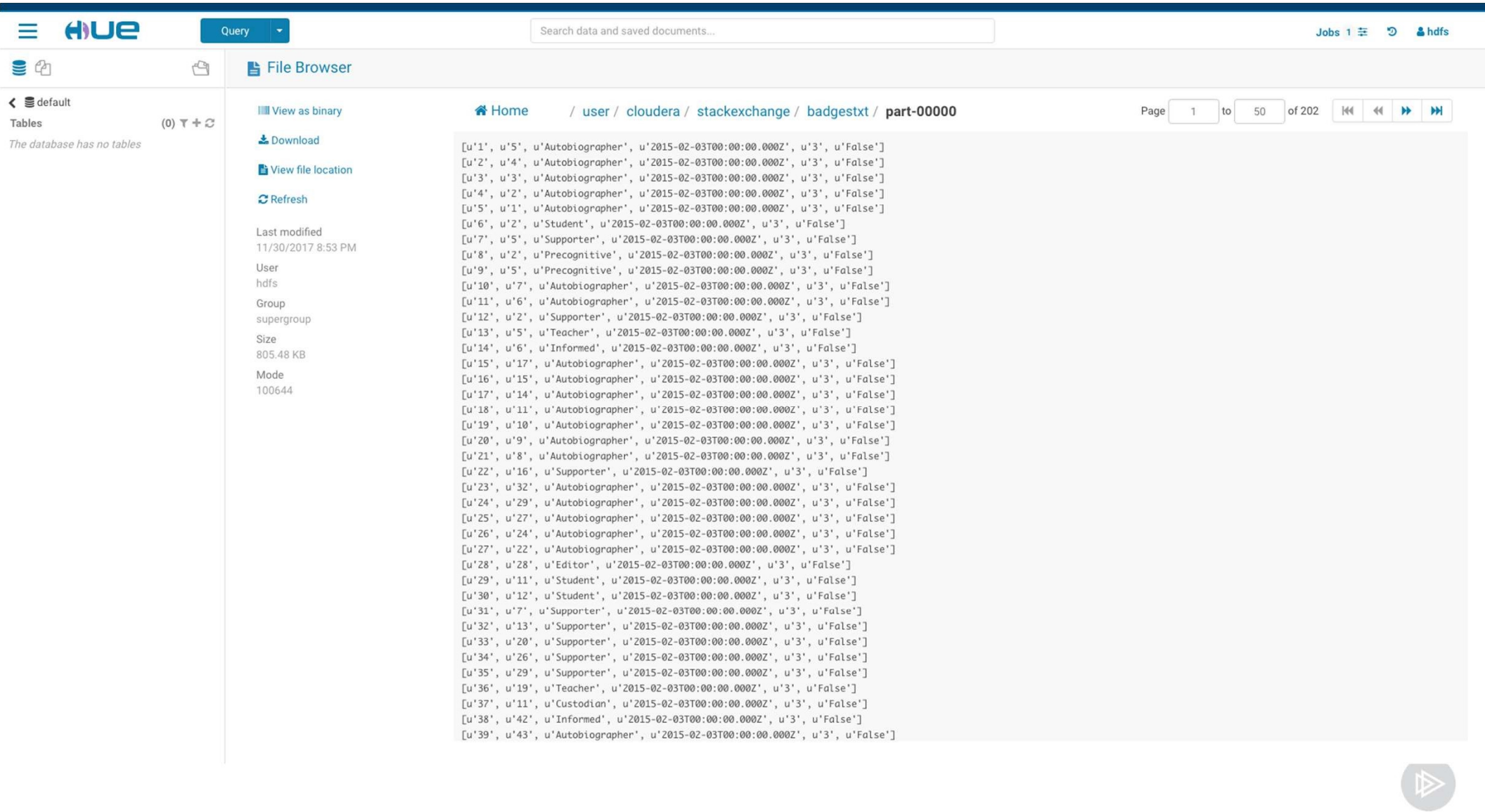
```
badges_columns_rdd.take(1)

badges_columns_rdd.saveAsTextFile("/user/cloudera/stackexchange/badgestxt")
```

Saving an RDD

We previously saved an RDD that had no transformations applied

- Partition files

Save an RDD with columns as text


Query
Search data and saved documents...
Jobs 1
hdfs
File Browser
default
Tables (0)
The database has no tables
View as binary
Download
View file location
Refresh
Last modified
11/30/2017 8:53 PM
User
hdfs
Group
supergroup
Size
805.48 KB
Mode
100644
Home / user / cloudera / stackexchange / badgestxt / part-00000
Page 1 to 50 of 202
[u'1', u'5', u'Autobiographer', u'2015-02-03T00:00:00.000Z', u'3', u'False']
[u'2', u'4', u'Autobiographer', u'2015-02-03T00:00:00.000Z', u'3', u'False']
[u'3', u'3', u'Autobiographer', u'2015-02-03T00:00:00.000Z', u'3', u'False']
[u'4', u'2', u'Autobiographer', u'2015-02-03T00:00:00.000Z', u'3', u'False']
[u'5', u'1', u'Autobiographer', u'2015-02-03T00:00:00.000Z', u'3', u'False']
[u'6', u'2', u'Student', u'2015-02-03T00:00:00.000Z', u'3', u'False']
[u'7', u'5', u'Supporter', u'2015-02-03T00:00:00.000Z', u'3', u'False']
[u'8', u'2', u'Precognitive', u'2015-02-03T00:00:00.000Z', u'3', u'False']
[u'9', u'5', u'Precognitive', u'2015-02-03T00:00:00.000Z', u'3', u'False']
[u'10', u'7', u'Autobiographer', u'2015-02-03T00:00:00.000Z', u'3', u'False']
[u'11', u'6', u'Autobiographer', u'2015-02-03T00:00:00.000Z', u'3', u'False']
[u'12', u'2', u'Supporter', u'2015-02-03T00:00:00.000Z', u'3', u'False']
[u'13', u'5', u'Teacher', u'2015-02-03T00:00:00.000Z', u'3', u'False']
[u'14', u'6', u'Informed', u'2015-02-03T00:00:00.000Z', u'3', u'False']
[u'15', u'17', u'Autobiographer', u'2015-02-03T00:00:00.000Z', u'3', u'False']
[u'16', u'15', u'Autobiographer', u'2015-02-03T00:00:00.000Z', u'3', u'False']
[u'17', u'14', u'Autobiographer', u'2015-02-03T00:00:00.000Z', u'3', u'False']
[u'18', u'11', u'Autobiographer', u'2015-02-03T00:00:00.000Z', u'3', u'False']
[u'19', u'10', u'Autobiographer', u'2015-02-03T00:00:00.000Z', u'3', u'False']
[u'20', u'9', u'Autobiographer', u'2015-02-03T00:00:00.000Z', u'3', u'False']
[u'21', u'8', u'Autobiographer', u'2015-02-03T00:00:00.000Z', u'3', u'False']
[u'22', u'16', u'Supporter', u'2015-02-03T00:00:00.000Z', u'3', u'False']
[u'23', u'32', u'Autobiographer', u'2015-02-03T00:00:00.000Z', u'3', u'False']
[u'24', u'29', u'Autobiographer', u'2015-02-03T00:00:00.000Z', u'3', u'False']
[u'25', u'27', u'Autobiographer', u'2015-02-03T00:00:00.000Z', u'3', u'False']
[u'26', u'24', u'Autobiographer', u'2015-02-03T00:00:00.000Z', u'3', u'False']
[u'27', u'22', u'Autobiographer', u'2015-02-03T00:00:00.000Z', u'3', u'False']
[u'28', u'28', u'Editor', u'2015-02-03T00:00:00.000Z', u'3', u'False']
[u'29', u'11', u'Student', u'2015-02-03T00:00:00.000Z', u'3', u'False']
[u'30', u'12', u'Student', u'2015-02-03T00:00:00.000Z', u'3', u'False']
[u'31', u'7', u'Supporter', u'2015-02-03T00:00:00.000Z', u'3', u'False']
[u'32', u'13', u'Supporter', u'2015-02-03T00:00:00.000Z', u'3', u'False']
[u'33', u'20', u'Supporter', u'2015-02-03T00:00:00.000Z', u'3', u'False']
[u'34', u'26', u'Supporter', u'2015-02-03T00:00:00.000Z', u'3', u'False']
[u'35', u'29', u'Supporter', u'2015-02-03T00:00:00.000Z', u'3', u'False']
[u'36', u'19', u'Teacher', u'2015-02-03T00:00:00.000Z', u'3', u'False']
[u'37', u'11', u'Custodian', u'2015-02-03T00:00:00.000Z', u'3', u'False']
[u'38', u'42', u'Informed', u'2015-02-03T00:00:00.000Z', u'3', u'False']
[u'39', u'43', u'Autobiographer', u'2015-02-03T00:00:00.000Z', u'3', u'False']

```
badges_cr_rdd =
sc.textFile("/user/cloudera/stackexchange/badgestxt")

badges_columns_rdd

badges_cr_rdd
```

Saving & Reloading an RDD

Reload our data

What happened when I save?

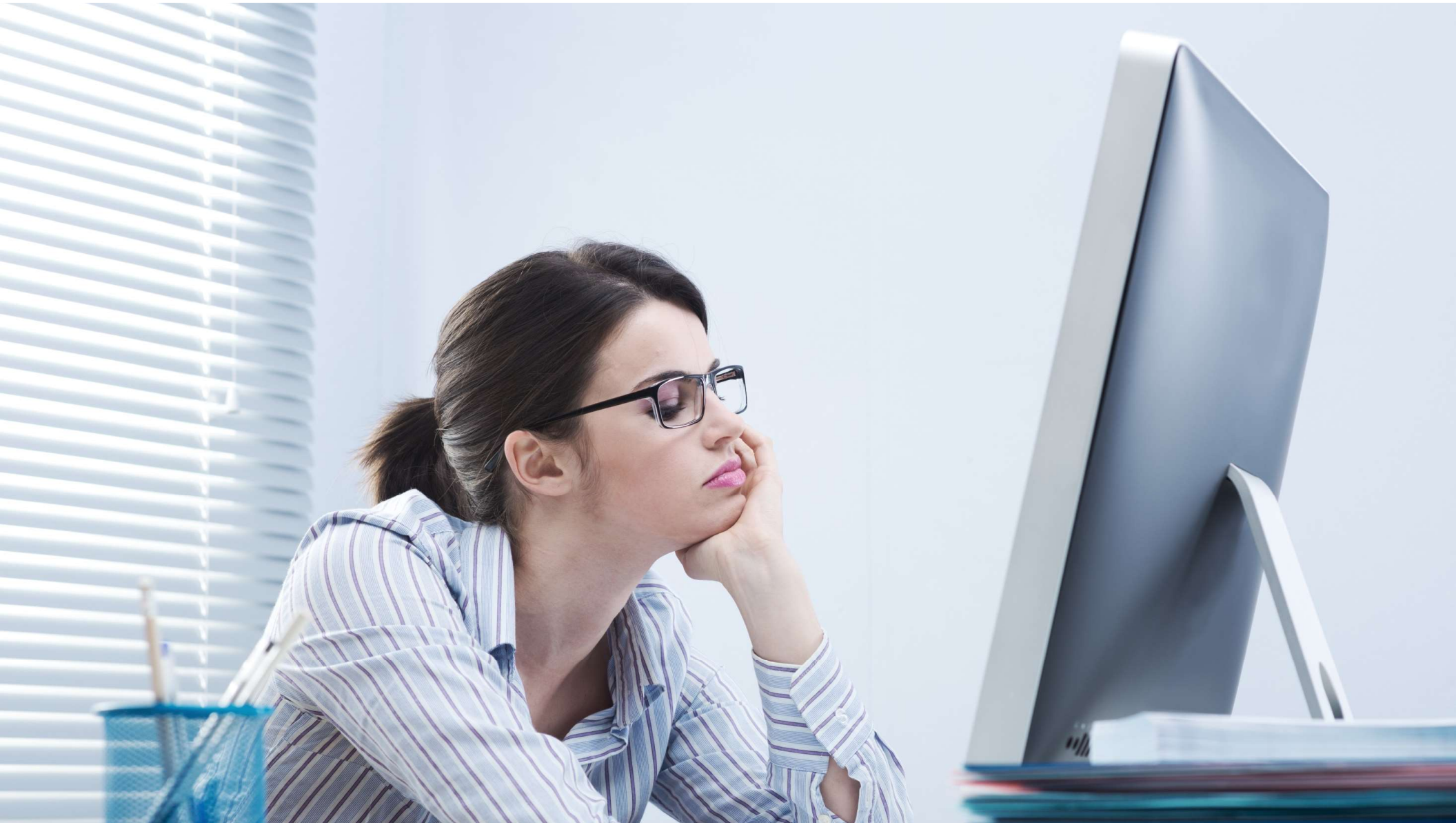
```
badges_columns_rdd.take(1)[0][2]
badges_cr_rdd.take(1)[0][2]
```

Text Is Good but Not Great

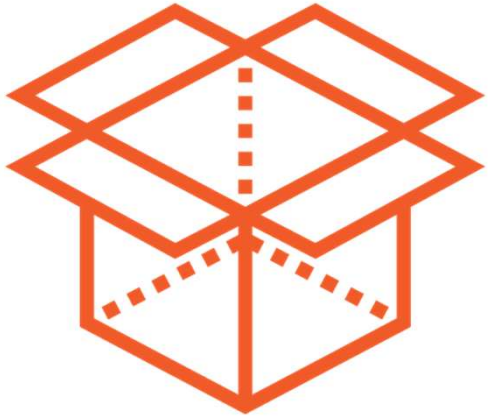
Compare original RDD vs. the one we just reloaded

We lost the format we applied

Text has no schema



Pickle in Python



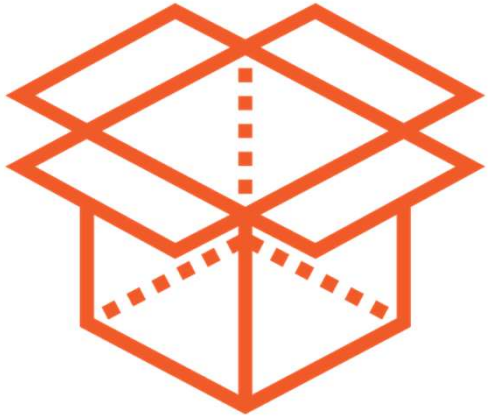
Serializing and deserializing objects

- Serializing → pickling
- Deserializing → unpickling

Preserve the object

Used regularly in Python

Pickle Files



Available in Spark too

Save as Pickle files

Do not lose applied transformations

```
badges_columns_rdd.saveAsPickleFile("/user/cloudera/stackex
change/badgespickle")

badges_crp_rdd =
sc.pickleFile("/user/cloudera/stackexchange/badgespickle")

badges_crp_rdd.take(1)

badges_crp_rdd.take(1)[0][2]
```

PickleFile

Save with saveAsPickleFile()

And load with pickleFile()

Saving a PickleFile in Spark actually saves a SequenceFile of Pickle objects

Watch out for this...

Sequence Files

- Flat file
- Binary format
- Key/value pairs
- Used extensively in Hadoop

```
badges_sorted.take(1)
badges_sorted.saveAsSequenceFile("/user/cloudera/
stackexchange/badgessequence")
badges_ss = sc.sequenceFile("/user/cloudera/stackexchange/
badgessequence")
badges_ss.take(1)
```

SequenceFile

Save using saveAsSequenceFile()

Read with sequenceFile()

Hadoop Formats

InputFormat

- How to read input files
- How they are split

OutputFormat

- How to write output files

Very flexible

```
badges_count_badge
  .saveAsNewAPIHadoopFile('/user/cloudera/badgesseqhadoop',
  "org.apache.hadoop.mapreduce.lib.output.SequenceFileOutputFormat",
  "org.apache.hadoop.io.IntWritable",
  "org.apache.hadoop.io.Text")
```

NewAPIHadopFile

More parameters required than other methods

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

And configuration

```
xml_posts =
  sc.newAPIHadoopFile('/user/cloudera/stackexchange/Posts.xml',
  'com.databricks.spark.xml.XmlInputFormat',
  'org.apache.hadoop.io.Text',
  'org.apache.hadoop.io.Text',
  conf={'xmlinput.start':'<row','xmlinput.end':'/>'})
```

Does newAPIHadoopFile Look Familiar?

We used to load the StackExchange dumps

Using spark-xml package

Load

textFile

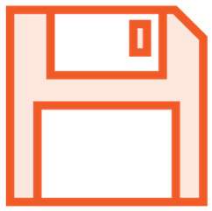
pickleFile

sequenceFile

hadoopFile

hadoopDataset

newAPIHadoopFile



saveAs*

saveAsTextFile

saveAsPickleFile

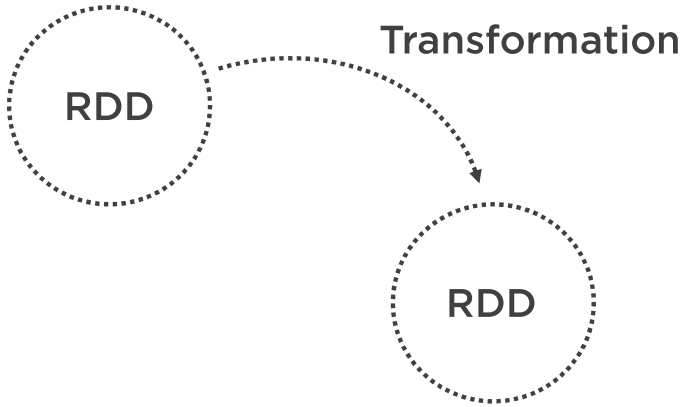
saveAsSequenceFile

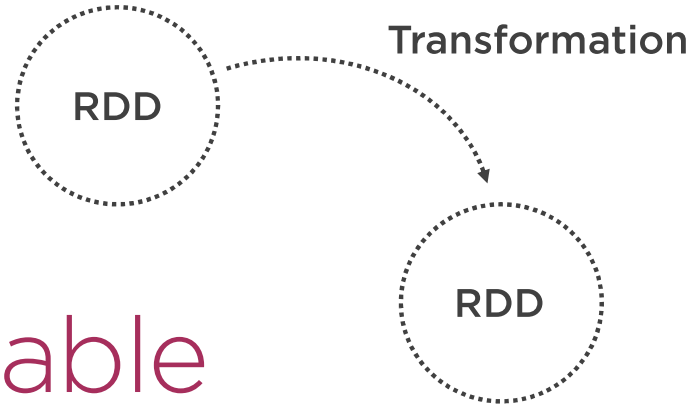
saveAsHadoopFile

saveAsHadoopDataset

saveAsNewAPIHadoopFile

Creating RDDs with Transformations





RDDs are Immutable

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

```
rdd_reuse = sc.parallelize([1,2])
rdd_reuse.collect()
rdd_reuse = sc.parallelize([3,4])
rdd_reuse.collect()
```

But... (On Immutable)

Immutable

- Can't change the data

Reuse the variable

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

map(split_the_line)

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

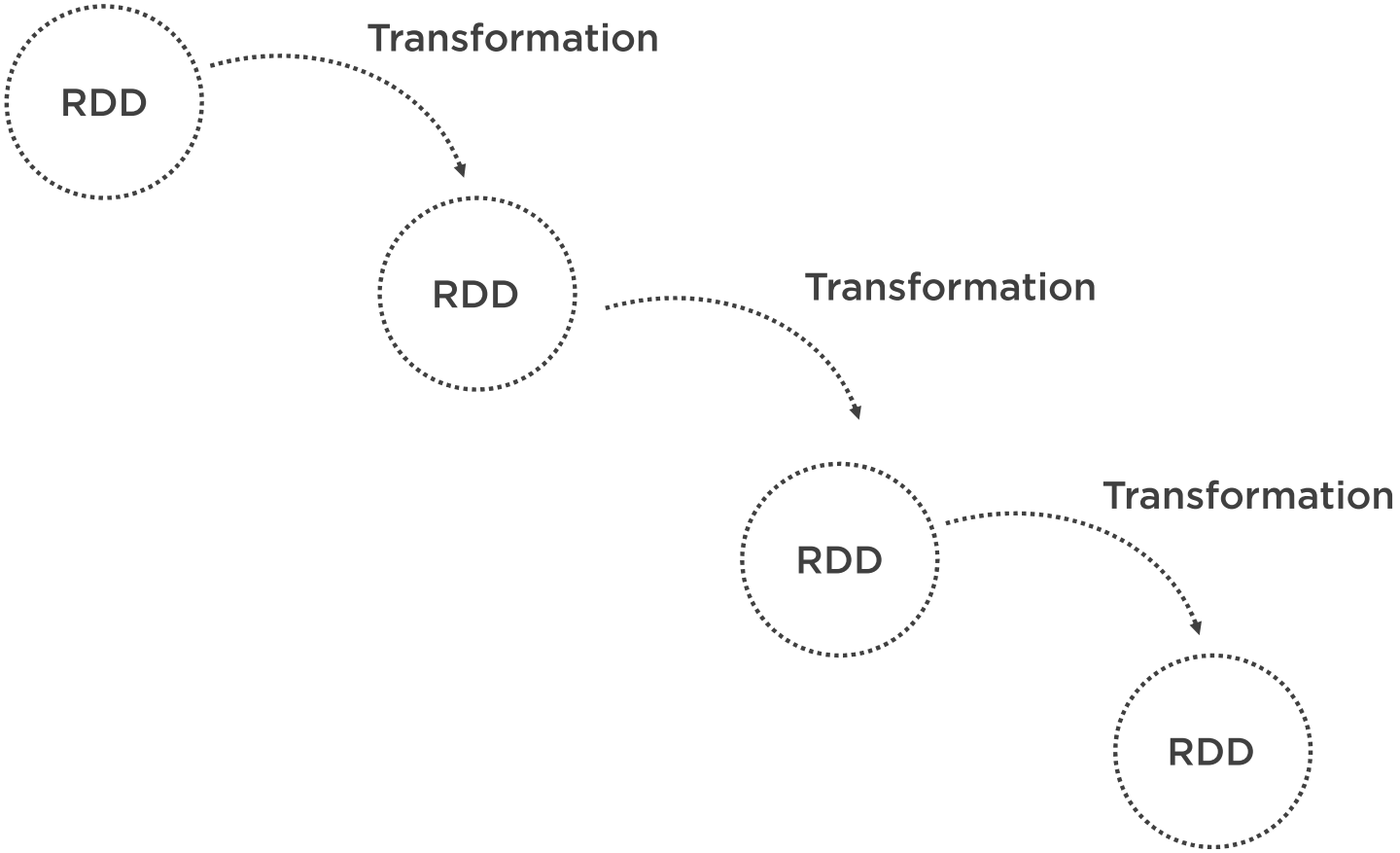
```
badges_rdd_csv.take(1)

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

```
badges_entry = badges_columns_rdd.map(lambda x: x[2])

badges_name = badges_entry.map(lambda x: (x,1))

badges_reduced = badges_name.reduceByKey(lambda x, y: x + y)

badges_count_badge = badges_reduced.map(lambda (x,y): (y,x))

badges_sorted = badges_count_badge.sortByKey(False).map(lambda (x, y): (y, x))

badges_sorted.take(10)
```

Creating Multiple RDDs with Transformations

One transformation

Iterate

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

```
from pyspark import RDD
locals()
globals()
vars()['badges_entry']
vars()['badges_entry'].count()
for(k,v)inlocals().items():
if isinstance(v,RDD) and 'badge' in k:
    print k
```

Tip: Listing RDDs

Check for created RDDs with globals(), locals() or vars()

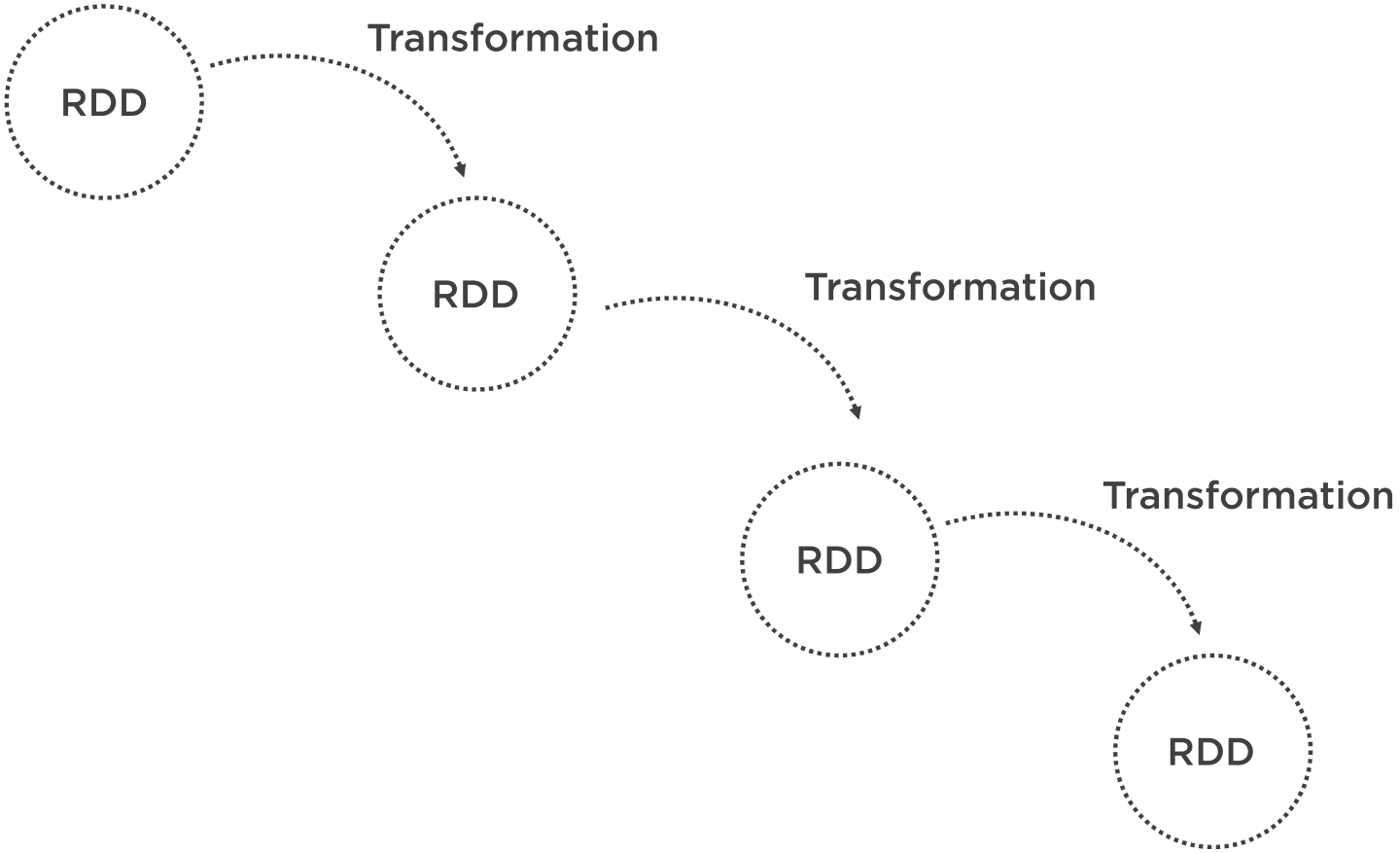
Look for those that are an instance of RDD

- Use if

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

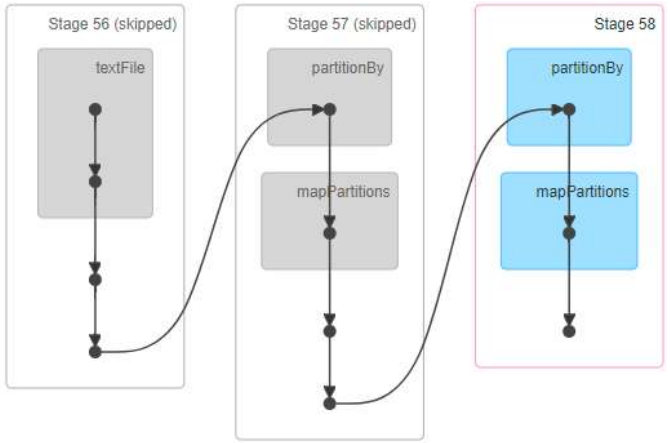
```
badges_sorted.toDebugString()

print badges_sorted.toDebugString()

badges_sorted

badges_sorted.prev

badges_sorted.take(1)

badges_sorted.prev.take(1)
```

A Little Bit More on Lineage

Use toDebugString()

And print to get nice formatting

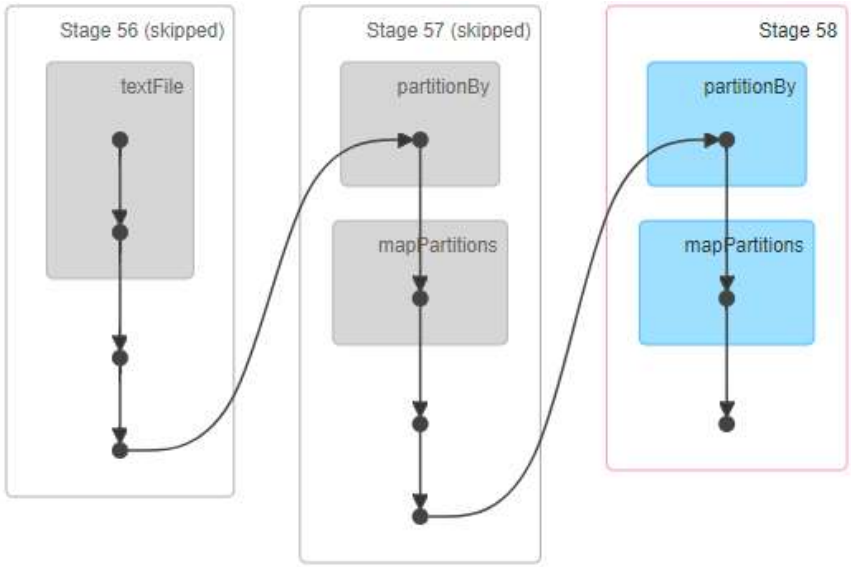
Just for fun, use prev



Details for Job 44

Status: SUCCEEDED
Completed Stages: 1
Skipped Stages: 2

▸ Event Timeline
▾ DAG Visualization



Completed Stages (1)

Stage Id ▾	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
58	runJob at PythonRDD.scala:446 +details	2018/01/12 06:38:38	0.9 s	1/1			803.0 B	

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, Picklefile, 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