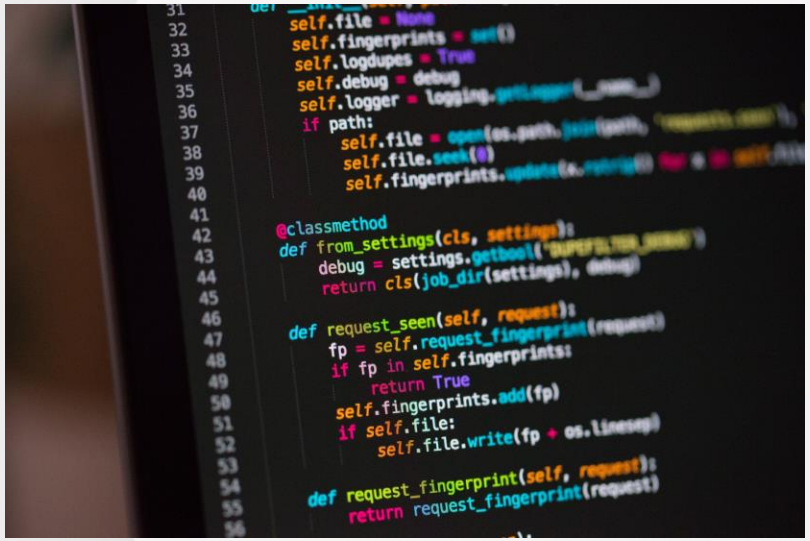
Level 5 Data Engineer
Module 3 Topic 8

Spark for Data Engineers



L5 Data Engineer Higher Apprenticeship
Module 3 / 12 (“**Programming and Scripting Essentials**”)
Topic 8 / 9

Delivering real-world value with Spark

Alibaba and Apache Spark

- **Speed:** Spark's in-memory processing capabilities allowed Alibaba to process large volumes of data in real-time
- **Scalability:** Spark's distributed computing model enabled Alibaba to scale their data processing as their data grew
- **Ease of Use:** Spark's high-level APIs made it easy for Alibaba's developers to write and maintain their data processing code

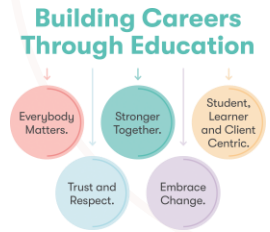
This case study demonstrates how Spark SQL and Apache Spark can deliver real-world value in the industry by providing scalable and efficient solutions for big data analytics

Alibaba.com™

A screen grab for interactive shell

BPP

Session aim and objectives



Completion of this topic supports the following outcomes:

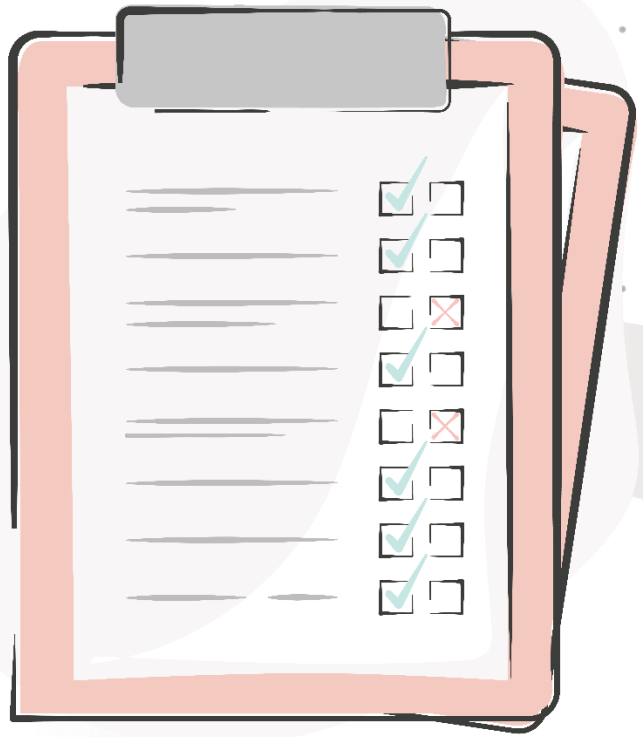
- Evaluate the use of Spark clusters for data processing
- List the essential features of data pipelines
- Explain how pipelines can be constructed used SparkSQL and Spark streaming
- Evaluate the most common deployment strategies for Spark applications

BPP

Webinar Agenda

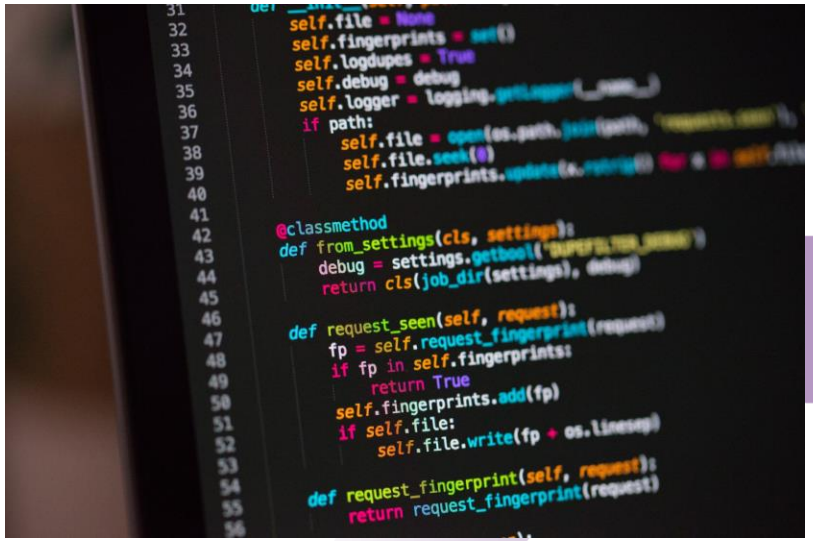
This webinar will include the following:

- E-learning Recap
- RDD and pipelines skills application
- Spark Broadcast Variables
- SparkSQL



BPP

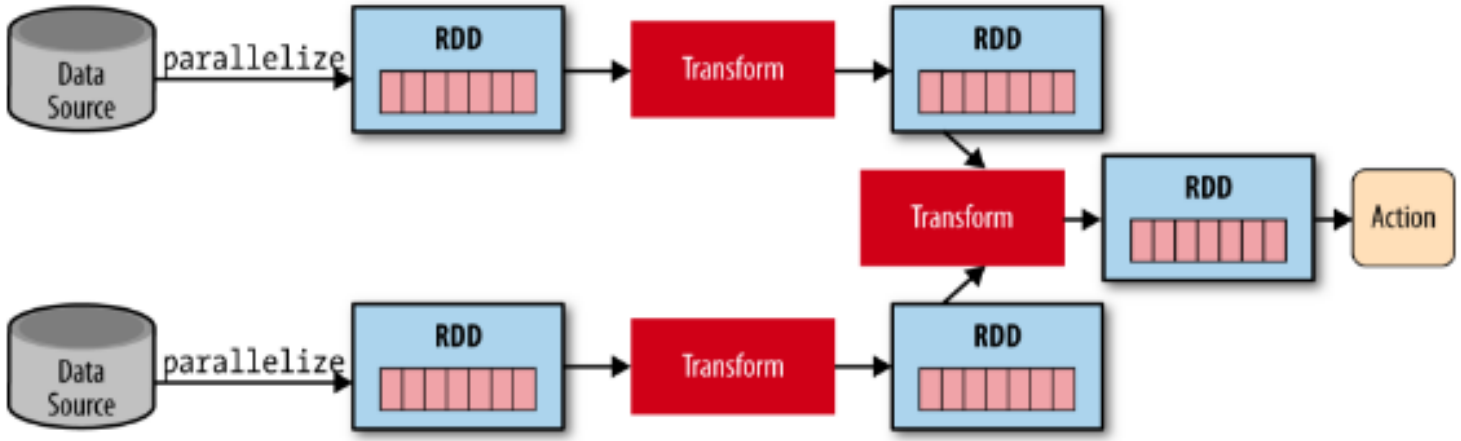
E-learning Recap



Recap: Creating RDDs using parallelize

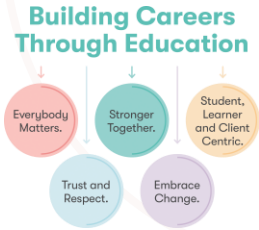
An example

```
# Turn a Python collection into an RDD
> sc.parallelize([1, 2, 3])

# Load text file from local FS, HDFS, or S3
> sc.textFile("file.txt")
> sc.textFile("directory/*.txt")
> sc.textFile("hdfs://namenode:9000/path/file")

# Use existing Hadoop InputFormat (Java/Scala only)
> sc.hadoopFile(keyClass, valClass, inputFmt, conf)
```



Methods used in Apache Spark for creating Resilient Distributed Datasets (RDDs)

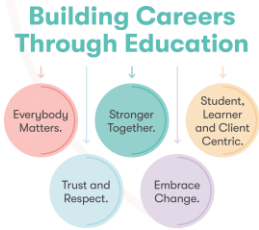
Recap: Basic transformations




```
> nums = sc.parallelize([1, 2, 3])

# Pass each element through a function
> squares = nums.map(lambda x: x*x)   // {1, 4, 9}

# Keep elements passing a predicate
> even = squares.filter(lambda x: x % 2 == 0) // {4}

# Map each element to zero or more others
> nums.flatMap(lambda x: => range(x))
    > # => {0, 0, 1, 0, 1, 2}
```

BPP

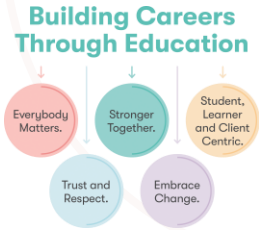
Recap: Basic actions




```
nums = sc.parallelize([1, 2, 3])

# Retrieve RDD contents as a local collection
nums.collect() # => [1, 2, 3]
# Return first K elements
nums.take(2) # => [1, 2]
# Count number of elements
nums.count() # => 3
# Merge elements with an associative function
nums.reduce(lambda x, y: x + y) # => 6
# Write elements to a text file
nums.saveAsTextFile("hdfs://file.txt")
```

BPP

Recap: Basic key-value operations




```
> pets = sc.parallelize(
    [("cat", 1), ("dog", 1), ("cat", 2)])
> pets.reduceByKey(lambda x, y: x + y)
                    # => {(cat, 3), (dog, 1)}
> pets.groupByKey() # => {(cat, [1, 2]), (dog, [1])}
> pets.sortByKey()  # => {(cat, 1), (cat, 2), (dog,
  1)}
```

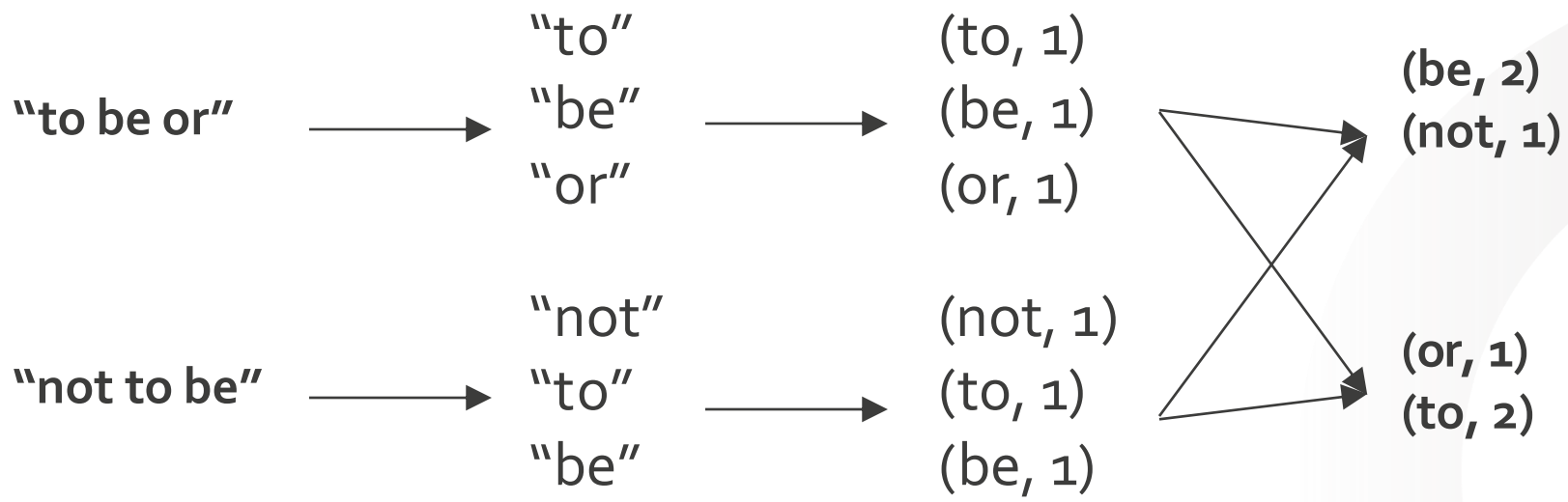
`reduceByKey` also automatically implements combiners on the map side

BPP

Recap: Basic key-value operations

An example: Wordcount

```
> lines = sc.textFile("hamlet.txt")
> counts = lines.flatMap(lambda line: line.split(" "))
                .map(lambda word => (word, 1))
                .reduceByKey(lambda x, y: x + y)
```

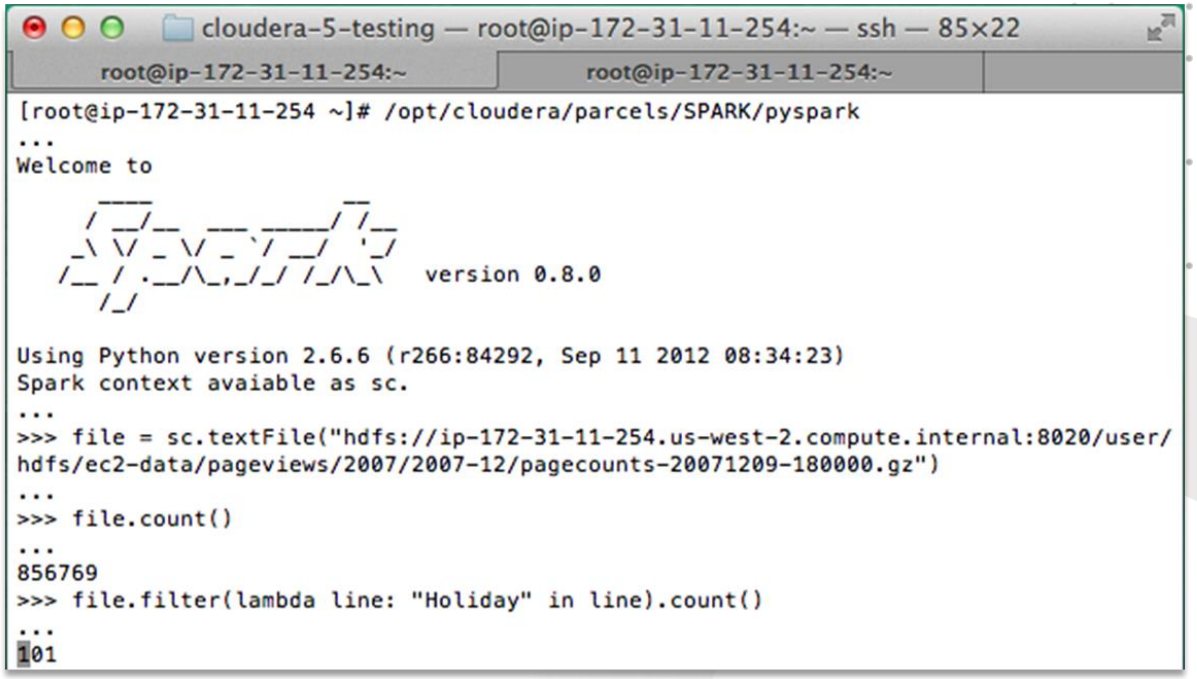


flatmap, map, and reduceByKey

Recap: Interactive shell

Here's what you need to know…

- The Fastest Way to Learn Spark
- Available in Python and Scala
- Runs as an application on an existing Spark Cluster…
- OR Can run locally

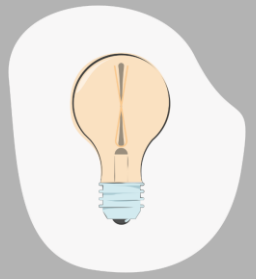


A screen grab for interactive shell

Spark standalone app (Python)

An example

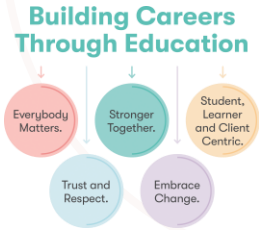
```
import sys
from pyspark import SparkContext

if __name__ == "__main__":
    sc = SparkContext( “local”, “wordCount”, sys.argv[0],
None)
    lines = sc.textFile(sys.argv[1])

    counts = lines.flatMap(lambda s: s.split(“ ”)) \
                  .map(lambda word: (word, 1)) \
                  .reduceByKey(lambda x, y: x + y)

    counts.saveAsTextFile(sys.argv[2])
```



What can we learn from this code example?

BPP

Recap: Parsing files

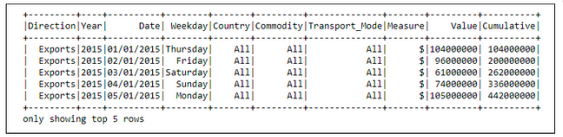


An example

- csv_lines = sc.textFile(“**example.csv**”)
- **def** csv_to_record(line):
  parts = line.split(",")
  record = {
  "**name**": parts[0],
  "**company**": parts[1],
  "**title**": parts[2]
  }
  return record

  # Apply the function to every record
  records = csv_lines.map(csv_to_record)

  # Inspect the first item in the dataset records.first()
- records.first()



```
+---------+----+----------+-------+-------+---------+--------------+--------+---------+----------+
|Direction|Year|      Date|Weekday|Country|Commodity|Transport_Mode|Measure|    Value|Cumulative|
+---------+----+----------+-------+-------+---------+--------------+--------+---------+----------+
|  Exports|2015|01/01/2015|Thursday|    All|      All|           All|      $|104000000| 104000000|
|  Exports|2015|02/01/2015|  Friday|    All|      All|           All|      $| 96000000| 200000000|
|  Exports|2015|03/01/2015|Saturday|    All|      All|           All|      $| 61000000| 262000000|
|  Exports|2015|04/01/2015|  Sunday|    All|      All|           All|      $| 74000000| 336000000|
|  Exports|2015|05/01/2015|  Monday|    All|      All|           All|      $|105000000| 442000000|
+---------+----+----------+-------+-------+---------+--------------+--------+---------+----------+
only showing top 5 rows
```

An illustration of parsing files in ApacheSpark

What can we learn from this code example?

BPP

Knowledge Check Poll

Which of the following statements about Apache Spark is correct?

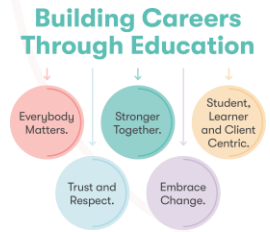
A. SparkSQL is used to create Resilient Distributed Datasets (RDDs)

B. Spark streaming cannot be used to construct data pipelines

C. sc.parallelize([1, 2, 3]) is a method for creating RDDs from an existing Python collection

D. In Spark, reduceByKey does not implement combiners on the map side.

Knowledge Check Poll



Which of the following statements about Apache Spark is correct?

A. SparkSQL is used to create Resilient Distributed Datasets (RDDs)

B. Spark streaming cannot be used to construct data pipelines

C. sc.parallelize([1, 2, 3]) is a method for creating RDDs from an existing Python collection

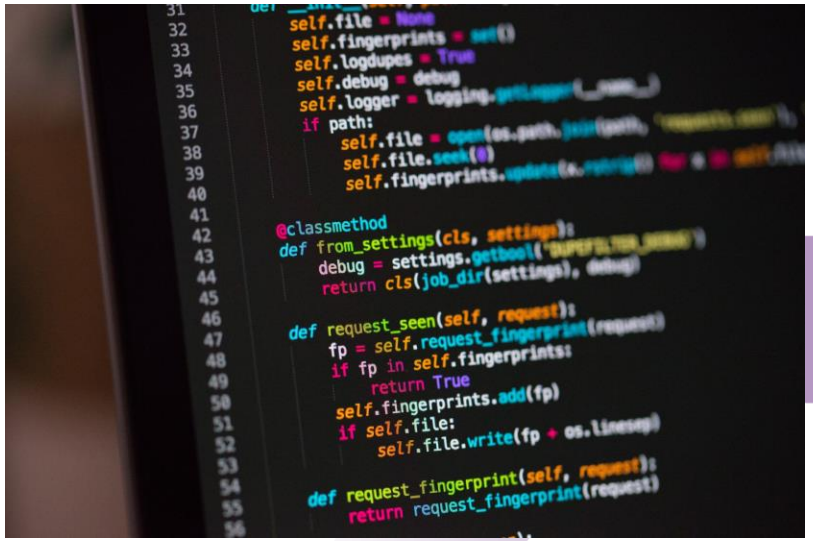
D. In Spark, reduceByKey does not implement combiners on the map side.

Feedback

The correct statement is **C** – sc.parallelize([1, 2, 3]) is a method for creating RDDs from an existing Python collection.

BPP

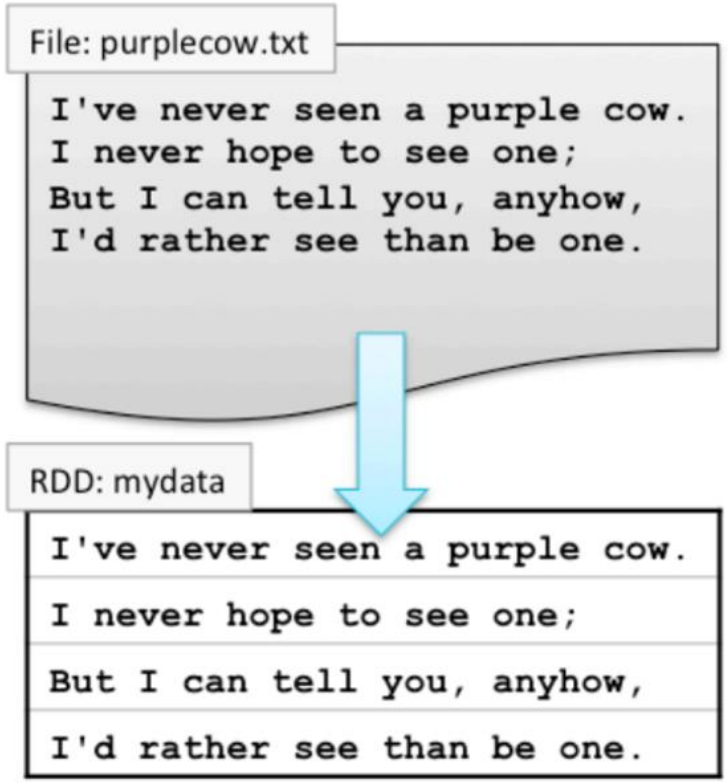
RDD applications



Example

A file-based RDD

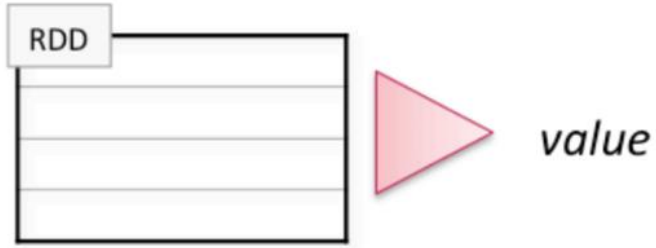
```
> val mydata = sc.textFile("purplecow.txt")
…
15/01/29 06:20:37 INFO storage.MemoryStore:
   Block broadcast_0 stored as values to
   memory (estimated size 151.4 KB, free 296.8
   MB)

> mydata.count()

…
15/01/29 06:27:37 INFO spark.SparkContext: Job
   finished: take at <stdin>:1, took
   0.160482078 s
4
```



RDD Operations: Actions

Some common actions:

- **Count ( ) –** return the number of elements
- **Take (n) –** return an array of the first *n* elements
- **Collect ( ) –** return an array of all elements
- **saveAsTextFile (*file*) –** save to text file(s)




```
> mydata =
  sc.textFile("purplecow.txt")

> mydata.count()
4

> for line in mydata.take(2):
    print line
I've never seen a purple cow.
I never hope to see one;
```

```
> val mydata =
  sc.textFile("purplecow.txt")

> mydata.count()
4

> for (line <- mydata.take(2))
    println(line)
I've never seen a purple cow.
I never hope to see one;
```

Example

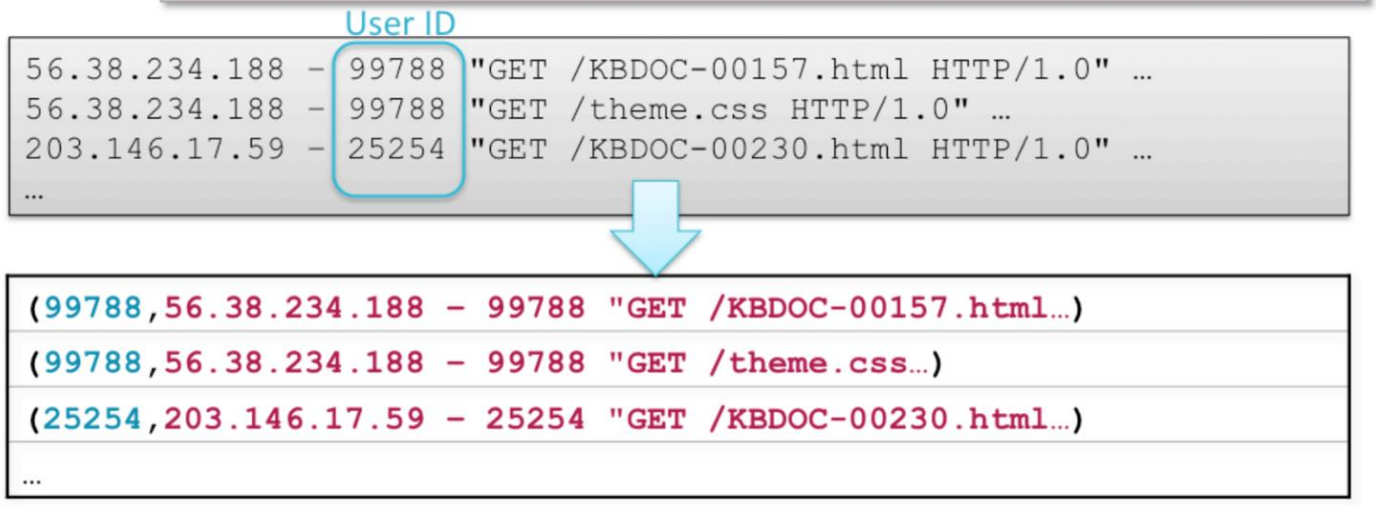
Keying web logs by UserID

Python

```
> sc.textFile(logfile) \
    .keyBy(lambda line: line.split(' ')[2])
```

Scala

```
> sc.textFile(logfile) \
    .keyBy(line => line.split(' ')(2))
```

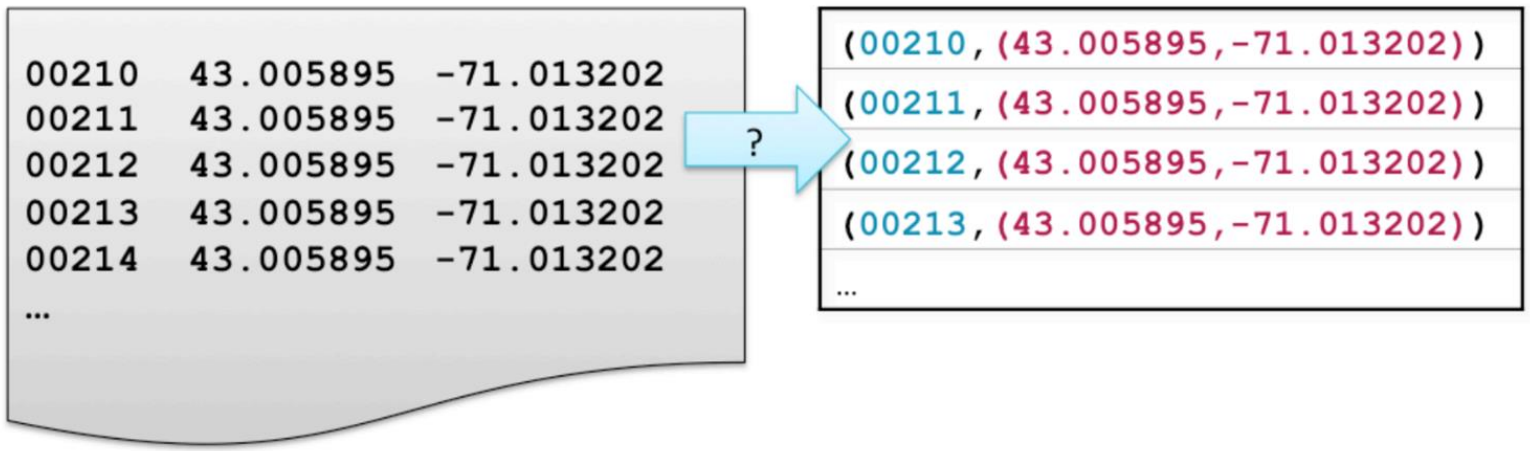


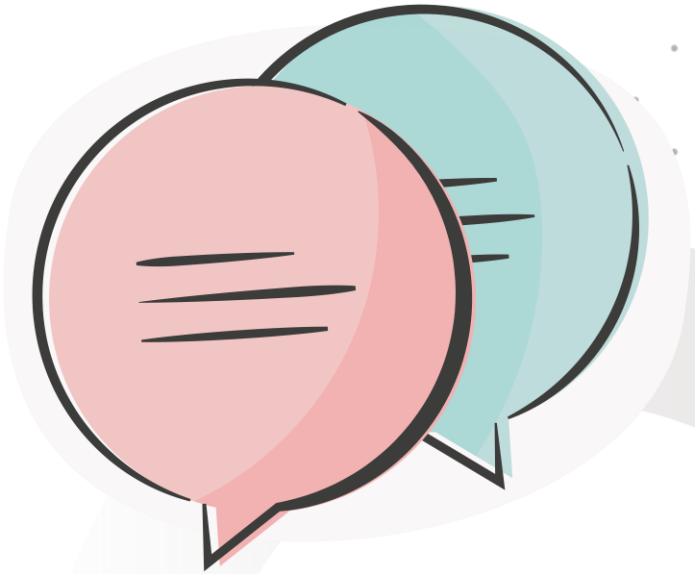
A diagram illustrating keying web logs with UserID

Pairs with complex values

How would you do this?

- **Input -** A list of postal codes with latitude and longtitude
- **Output –** postal code (key) and lat/long pair (value)

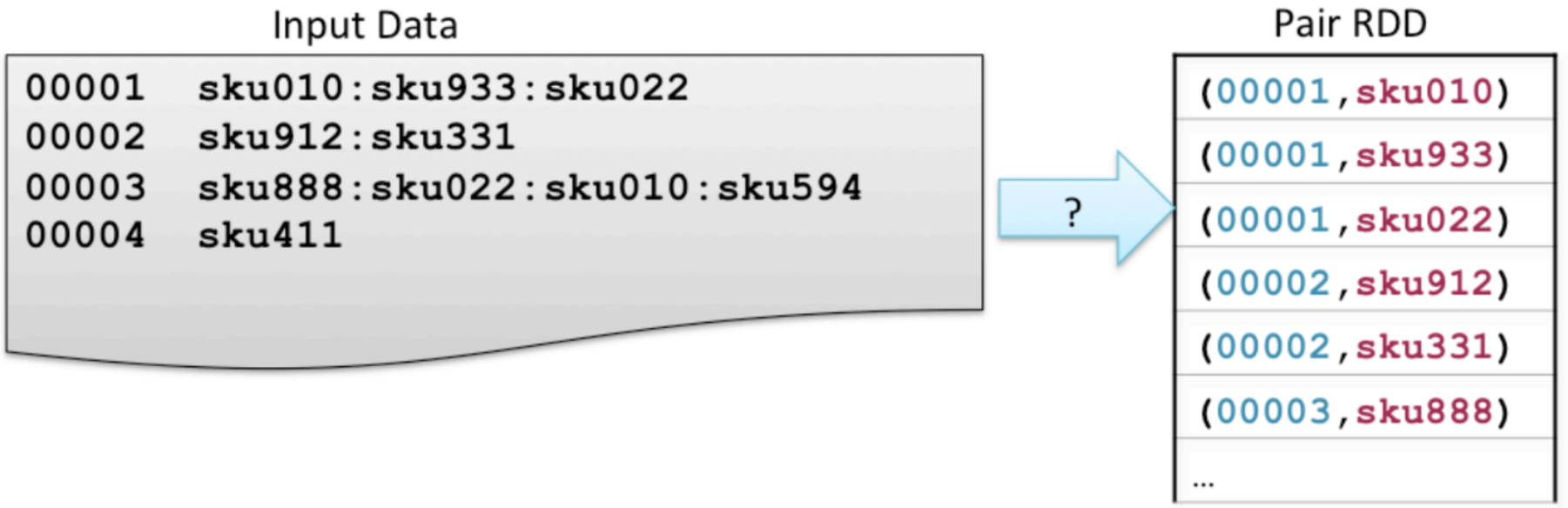




Discussion activity

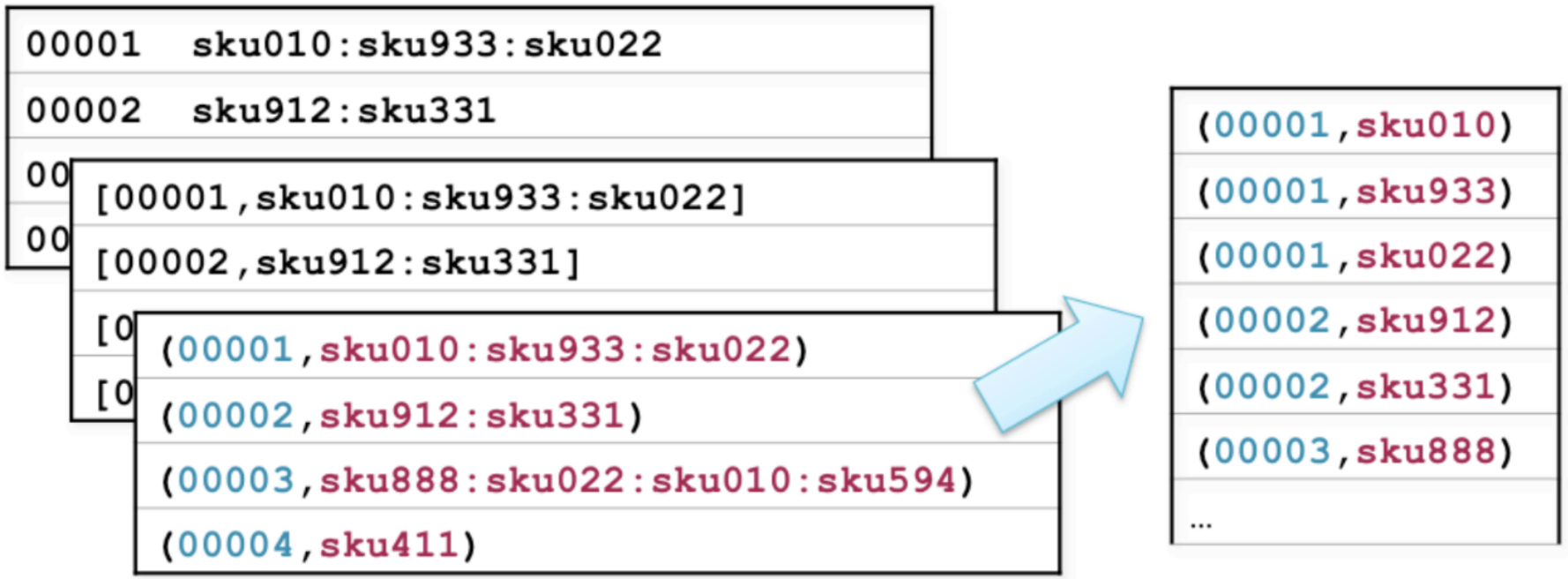
Question 2: Mapping Single Rows to Multiple Pairs (1)

- **How would you do this?**
  - Input: order numbers with a list of SKUs in the order
  - Output: order (key) and sku (value)



Answer 2: Mapping Single Rows to Multiple Pairs (4)

```
> sc.textFile(file) \
  .map(lambda line: line.split('\t')) \
  .map(lambda fields: (fields[0],fields[1]))
  .flatMapValues(lambda skus: skus.split(':'))
```

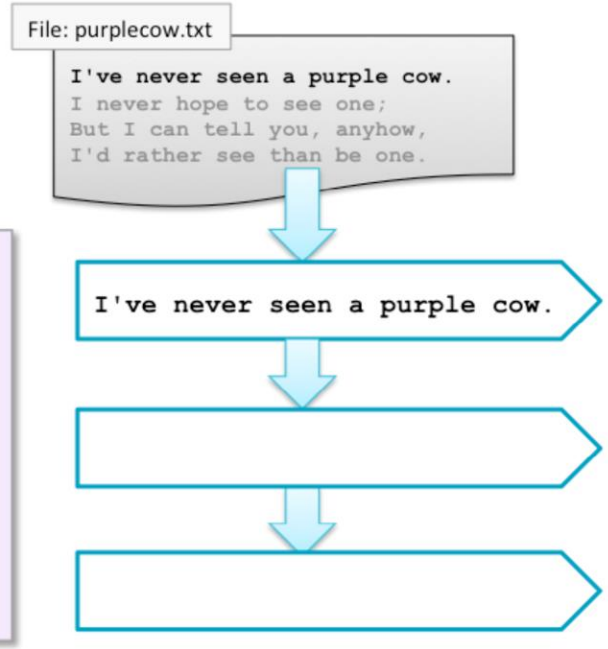


Pipelining

Example 1

When possible, Spark will perform sequences of transformations by row so no data is stored.

```
> val mydata = sc.textFile("purplecow.txt")
> var mydata_uc = mydata.map(line =>
  line.toUpperCase())
> var mydata_filt = mydata_uc.filter(line
  => line.startsWith("I"))
> mydata_filt.take(2)
```



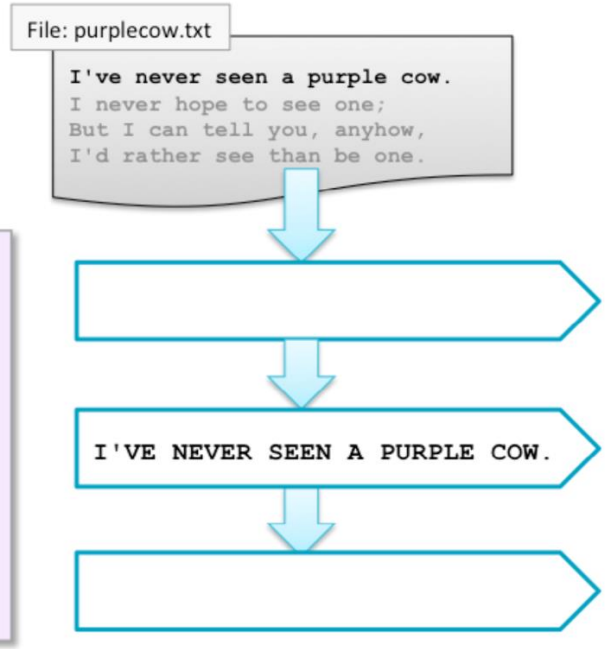
Example 1

Pipelining

Example 2

When possible, Spark will perform sequences of transformations by row so no data is stored.

```
> val mydata = sc.textFile("purplecow.txt")
> var mydata_uc = mydata.map(line =>
  line.toUpperCase())
> var mydata_filt = mydata_uc.filter(line
  => line.startsWith("I"))
> mydata_filt.take(2)
```

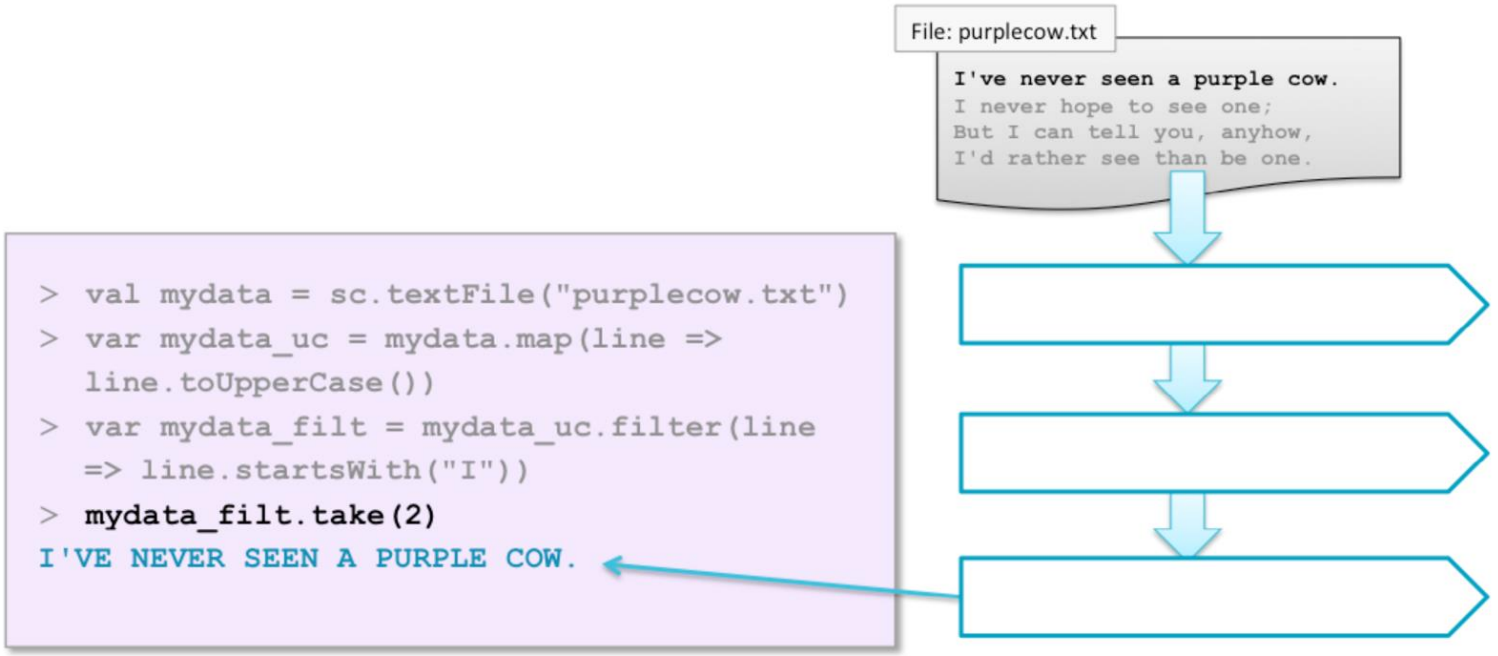


Example 2

Pipelining

Example 3

When possible, Spark will perform sequences of transformations by row so no data is stored.

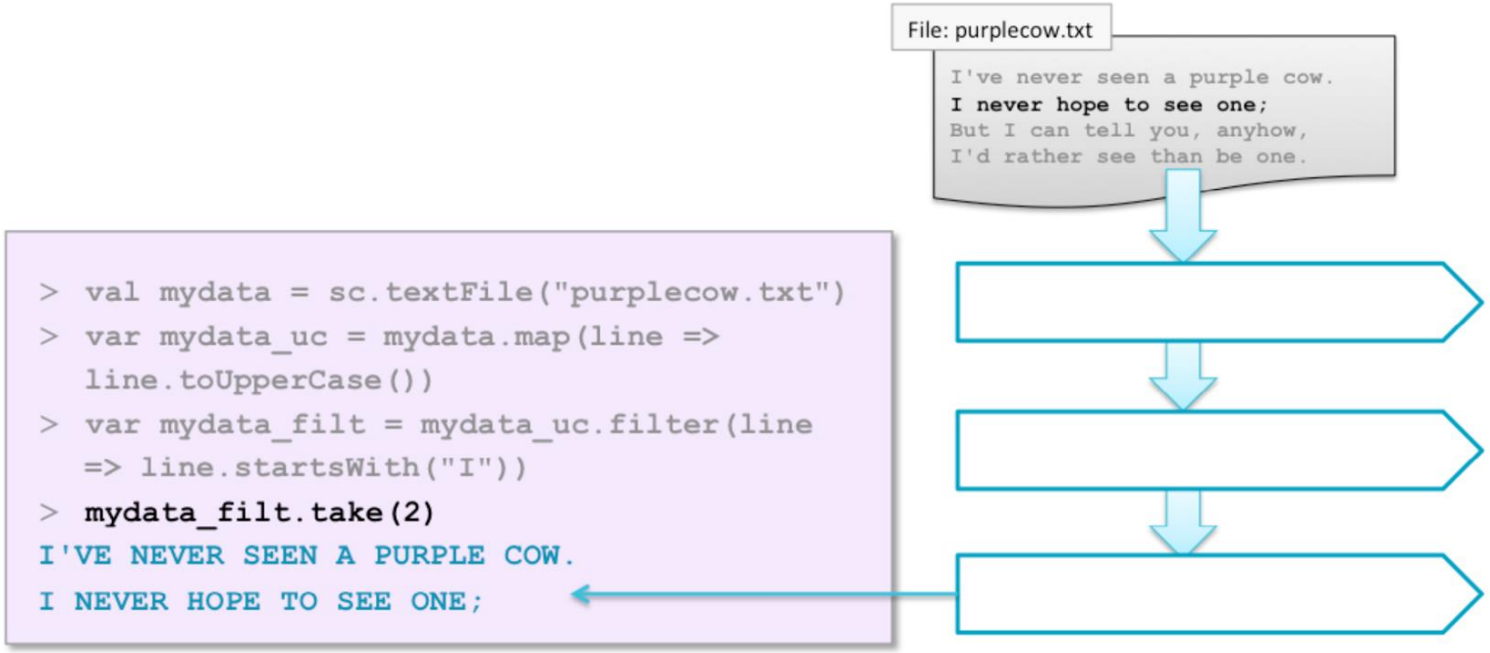


Example 3

Pipelining

Example 5

When possible, Spark will perform sequences of transformations by row so no data is stored.



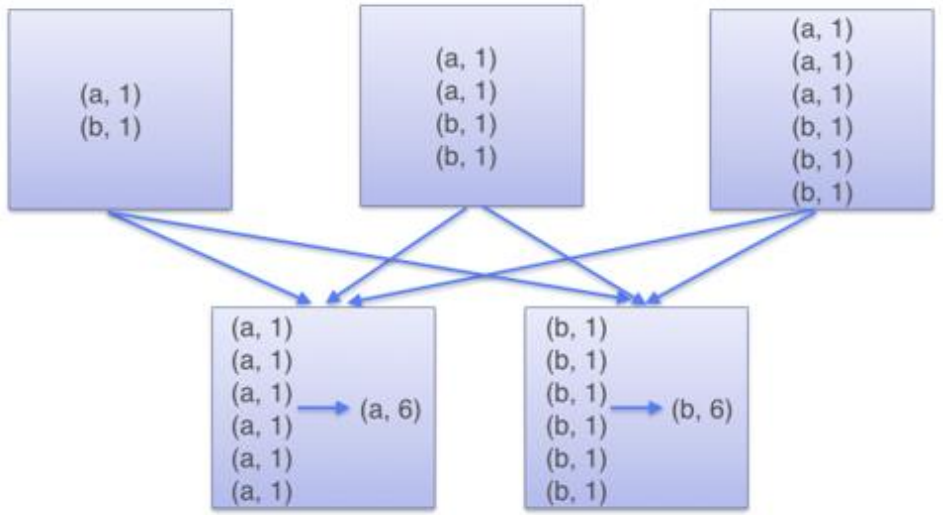
Example 5

Pair RDD operations

In addition to map and reduce functions, Spark has several operations specific to Pair RDDs.

Examples:

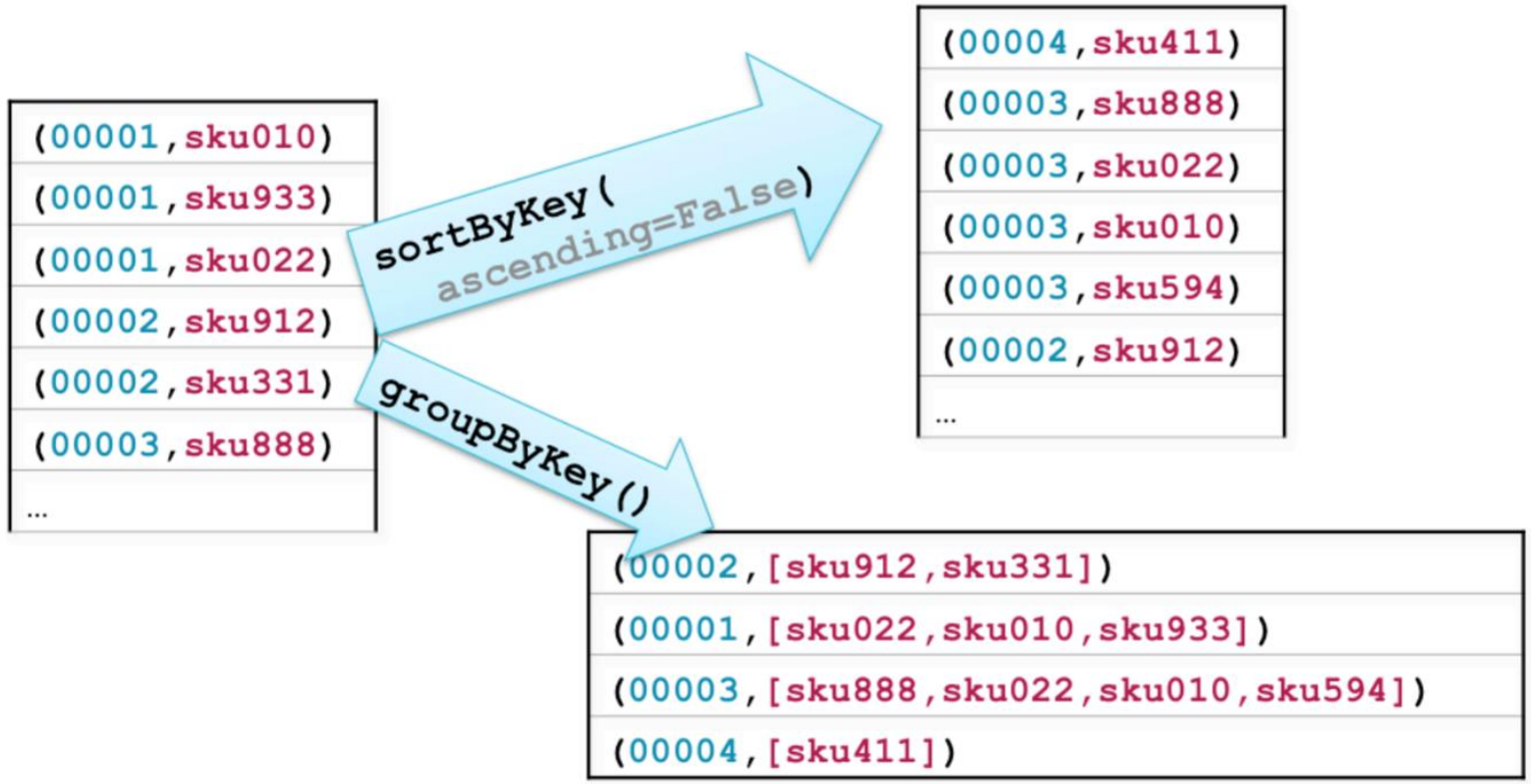
- *countByKey* - return a map with the count of occurrences of each key
- *GroupByKey* - group all the values for each key in an RDD
- sortByKey - sort in ascending or descending order
- *Join* – return an RDD containing all pairs with matching keys from two RDDs



GroupByKey

Example

Pair RDD operations



An example of Pair RDD operations

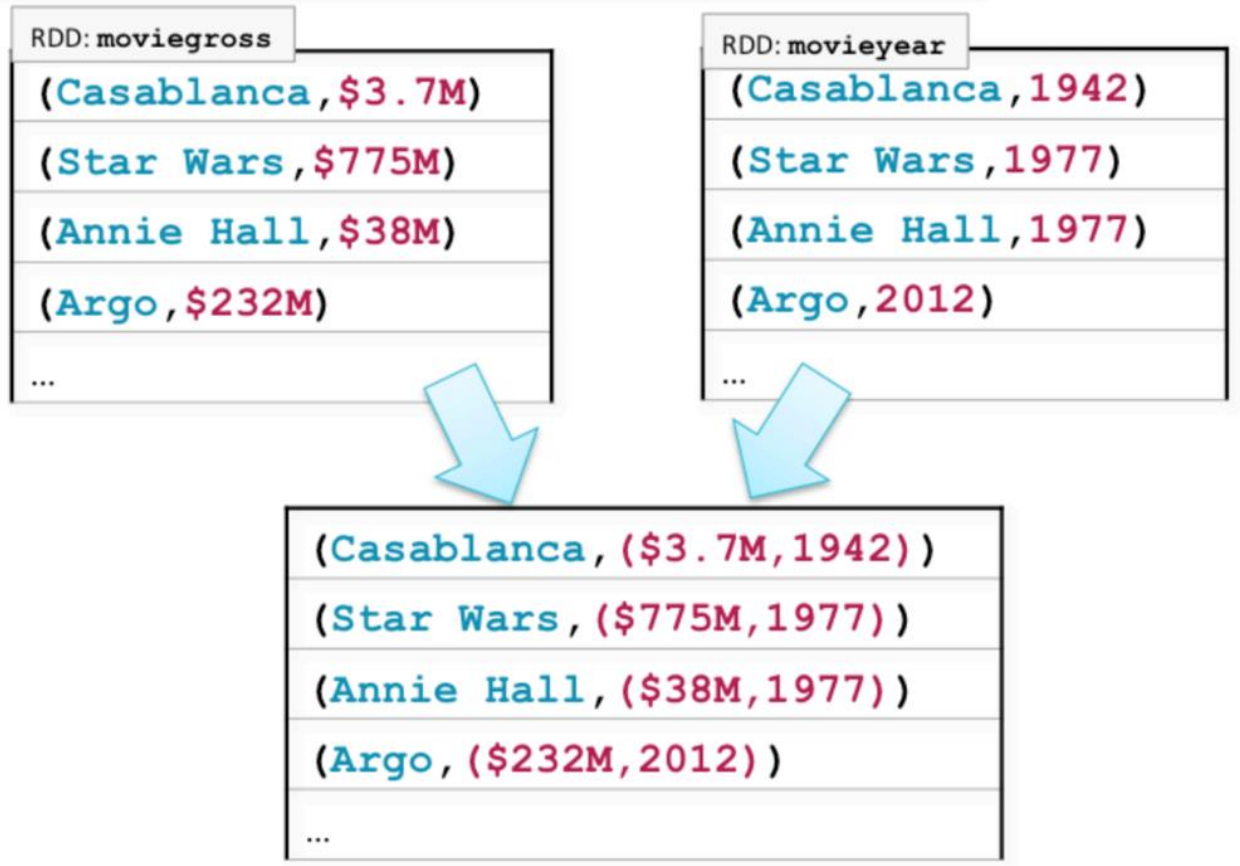
Using Join

A common programming pattern.

```
> movies = moviegross.join(movieyear)
```

Examples:

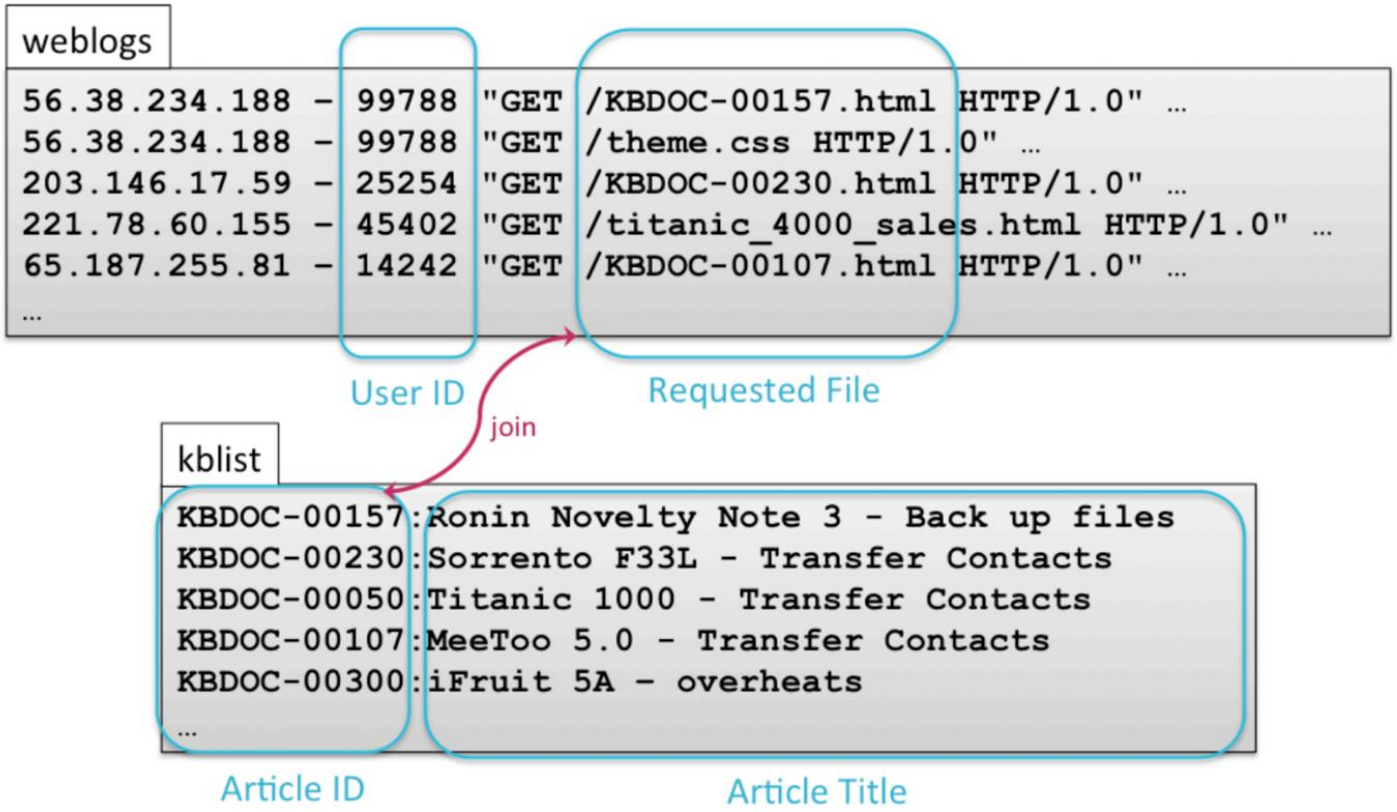
- Map separate datasets into key-value Pair RDDs
- Join by key
- Map joined data into the desired format
- Save, display, or continue processing



An example of Joining by Key

Example

Join web log with knowledge base articles



An example of joining a web log with knowledge base articles

Practical application

Tutorial walkthrough – Join web log with knowledge base articles

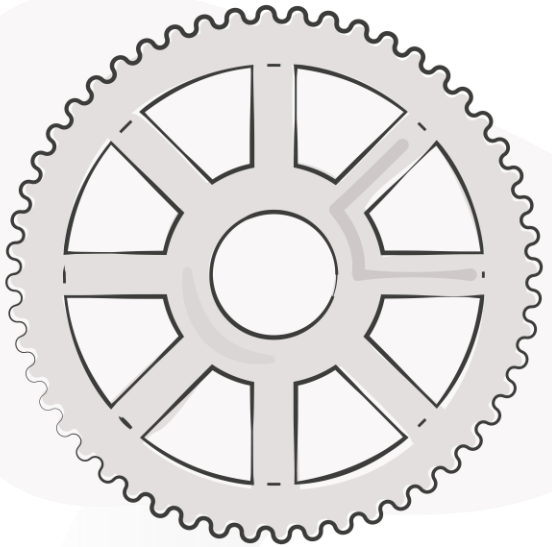
Steps:

1. Map separate datasets into key-value Pair RDDs

    A) Map web log requests to (*docid*, *userid*)

    B) Map KB Doc index to (*docid, title*)

2) Join by key: *docid*
3) Map joined data into the desired format*: (userid, title)*
4) Further processing: group titles by User ID



Practical application

BPP

Knowledge Check Poll

Which of the following statements about Apache Spark is correct?

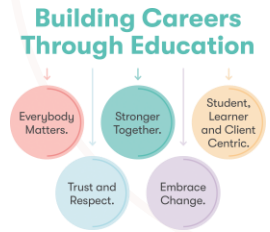
A. SparkSQL and Spark Streaming cannot be used to construct data pipelines

B. Spark performs sequences of transformations by row, storing no data, when possible

C. Pair RDD operations in Spark include only map and reduce functions

D. Spark cannot perform operations specific to Pair RDDs

Knowledge Check Poll



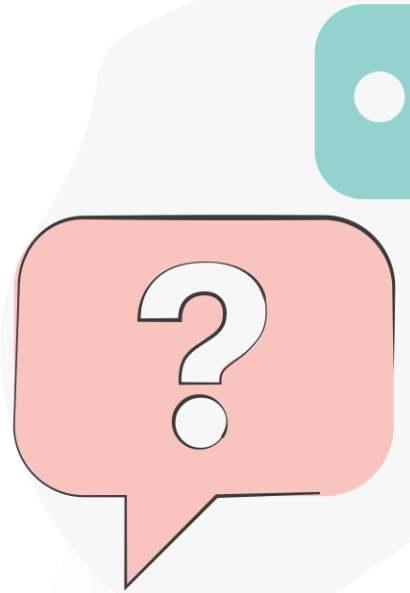
Which of the following statements about Apache Spark is correct?

A. SparkSQL and Spark Streaming cannot be used to construct data pipelines

B. Spark performs sequences of transformations by row, storing no data, when possible

C. Pair RDD operations in Spark include only map and reduce functions

D. Spark cannot perform operations specific to Pair RDDs

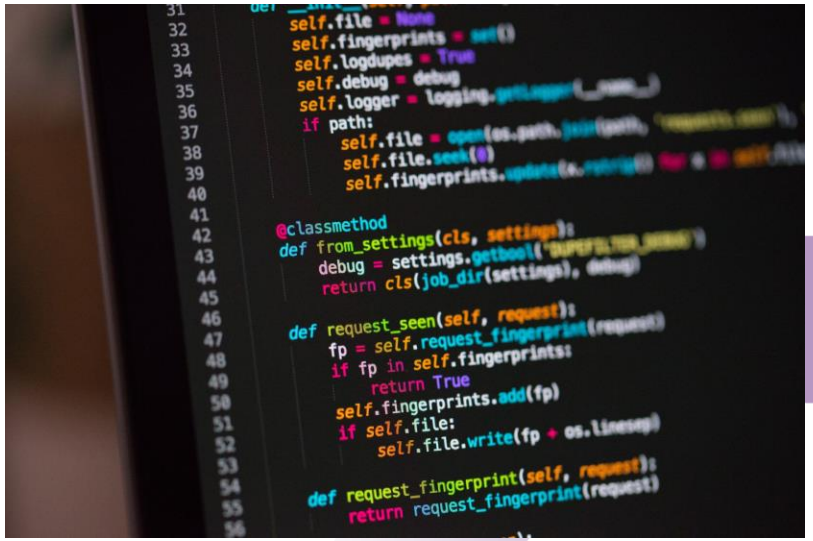


Feedback

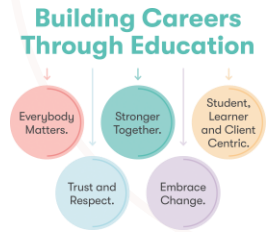
The correct statement is **B** – Spark performs sequences of transformations by row, storing no data, when possible. This process is known as pipelining.

BPP

Spark Broadcast Variables

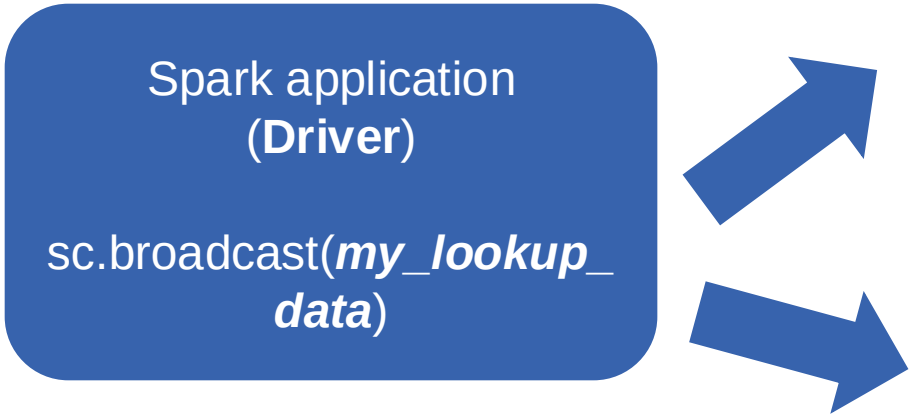


Spark broadcast variables

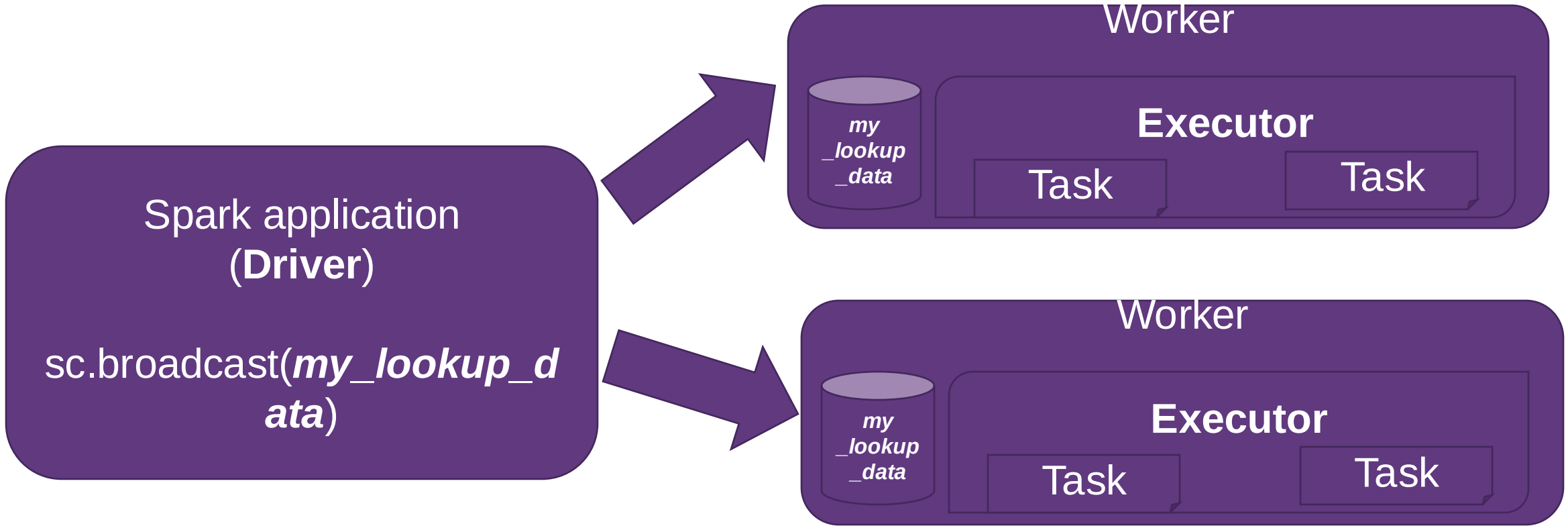


Why use them…?

- Often worker nodes will need to perform lots of **look-up style operations**, e.g. when matching people's addresses to their post-codes
- If a large **look-up table exist**– this can be **distributed** to multiple nodes as a **broadcast variable**
- That will then save time and network traffic, because the look-up table has already been distributed across the cluster, and **every worker has a copy**
- This data persists across multiple tasks that the worker has to do



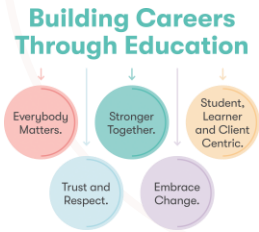
BPP



Broadcasting in action

Example

Leveraging SparkContext.broadcast() for Efficient Data Distribution in Map Transformations



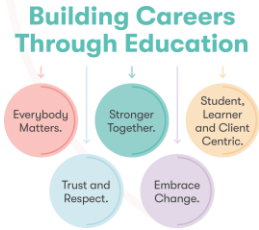

```
states = {"NY":"New York","CA":"California","FL",:Florida"}
broadcastStates = spark.sparkContext.broadcast(states)

countries = {"USA":"United States of America","IN":"India"}
broadcastCountries = spark.sparkContext.broadcast(countries)

data = [("James","Smith","USA","CA"), ("Michael","Rose","USA","NY"), ("Robert","Williams","USA","CA"),
("Maria","Jones","USA","FL")]
rdd = spark.sparkContext.parallelize(data)

rdd2 = rdd.map(lambda (name,surname,country,state): (name, surname, broadcastCountries.value.get(country),
broadcastStates.value.get(state))

print(rdd2.collect())
```

Use of SparkContext.broadcast() for distributing data and the application of these variables in RDD map() transformations

BPP

Example

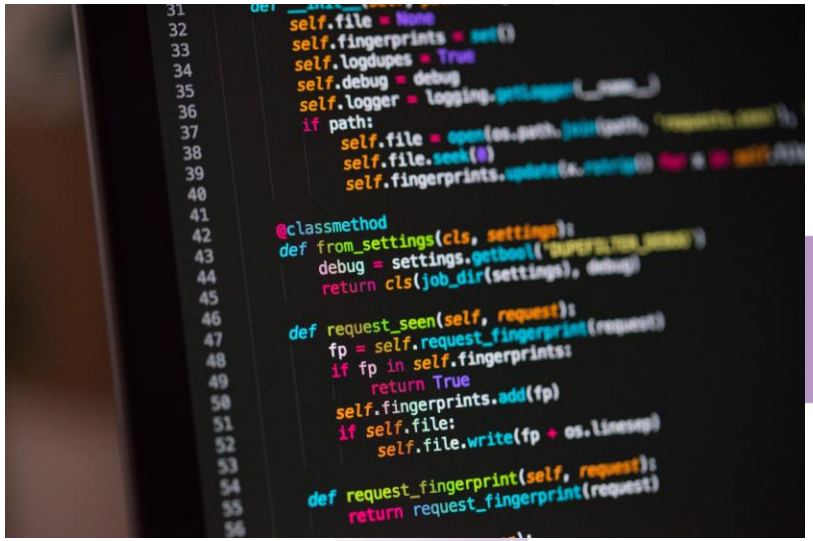
Leveraging SparkContext.broadcast() for Efficient Data Distribution in Map Transformations




```
states = {"NY":"New York","CA":"California","FL",:Florida"}
broadcastStates = spark.sparkContext.broadcast(states)

countries = {"USA":"United States of America","IN":"India"}
broadcastCountries = spark.sparkContext.broadcast(countries)

data = [("James","Smith","USA","CA"), ("Michael","Rose","USA","NY"), ("Robert","Williams","USA","CA"),
("Maria","Jones","USA","FL")]
rdd = spark.sparkContext.parallelize(data)

rdd2 = rdd.map(lambda (name,surname,country,state): (name, surname, broadcastCountries.value.get(country),
broadcastStates.value.get(state))

print(rdd2.collect())
```

BPP

SparkSQL


self.file = None
self.fingerprints = set()
self.logdupes = True
self.debug = debug
self.logger = logging.getLogger(__name__)
if path:
self.file = open(os.path.join(path, 'requests.seen'), 'a+')
self.file.seek(0)
self.fingerprints.update(x.rstrip() for x in self.file)
@classmethod
def from_settings(cls, settings):
debug = settings.getbool('DUPEFILTER_DEBUG')
return cls(job_dir(settings), debug)
def request_seen(self, request):
fp = self.request_fingerprint(request)
if fp in self.fingerprints:
return True
self.fingerprints.add(fp)
if self.file:
self.file.write(fp + os.linesep)
def request_fingerprint(self, request):
return request_fingerprint(request)

SparkQL

What is it and why use it?

You can use Spark's module for executing SQL queries using a programming abstraction called the DataFrame.


```
# 2. Import and create a SparkContext:
from pyspark import SparkContext, SparkConf
conf = SparkConf().setAppName("appName").setMaster("local[*]")
sc = SparkContext.getOrCreate(conf)
```

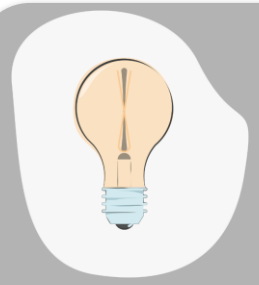
```
# 3. Generate an RDD from the created data:
rdd = sc.parallelize(data)
type(rdd)
```

```
pyspark.rdd.RDD
```

```
# 4. Call the toDF() method on the RDD to create the DataFrame:
df = rdd.toDF()
type(df)
```

```
pyspark.sql.dataframe.DataFrame
```

An example of a Spark DataFrame



Is this a database?

Why use SparkSQL over others – ie MySQL?

Discussion answers

Is this a database?

No, SparkSQL does not create a database, instead you use your existing distributed data that already lives in your cluster (or in the cloud), e.g. text files.

Any data that can be imported by Spark, can be analysed with SparkSQL.

Why use SparkSQL over e.g. MySQL?

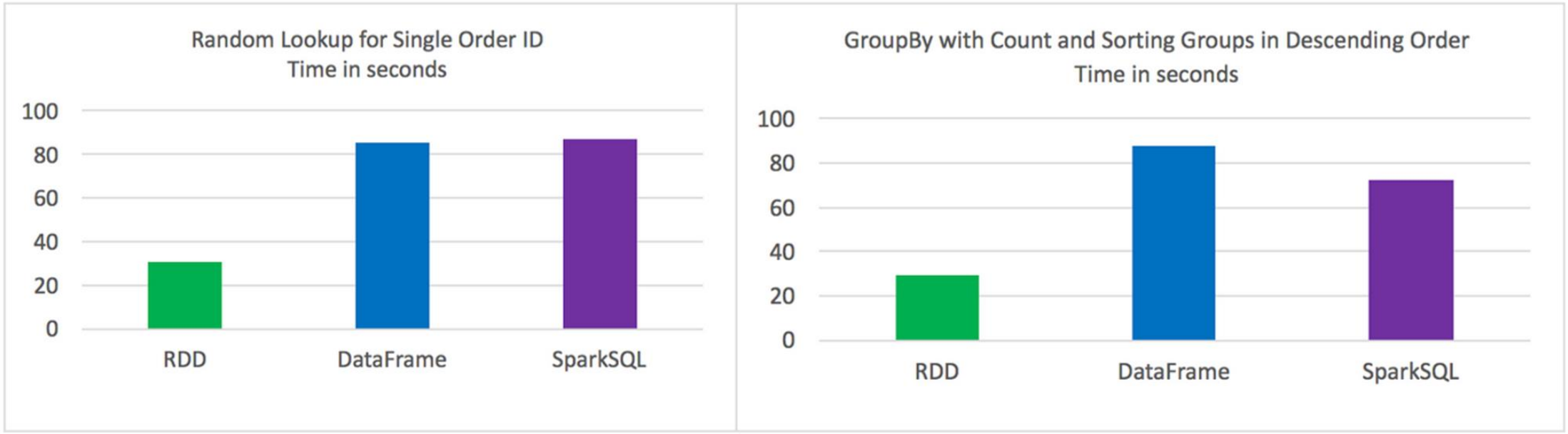
MySQL requires you to import your data into a database (using a schema) which creates overhead.

MySQL also can only use one CPU core per query, whilst Spark can use all cores on all cluster nodes.

It could be possible that MySQL is faster for certain types for queries, but Spark will scale better.

SparkSQL performance

A little slower than using ‘raw’ RDDs…



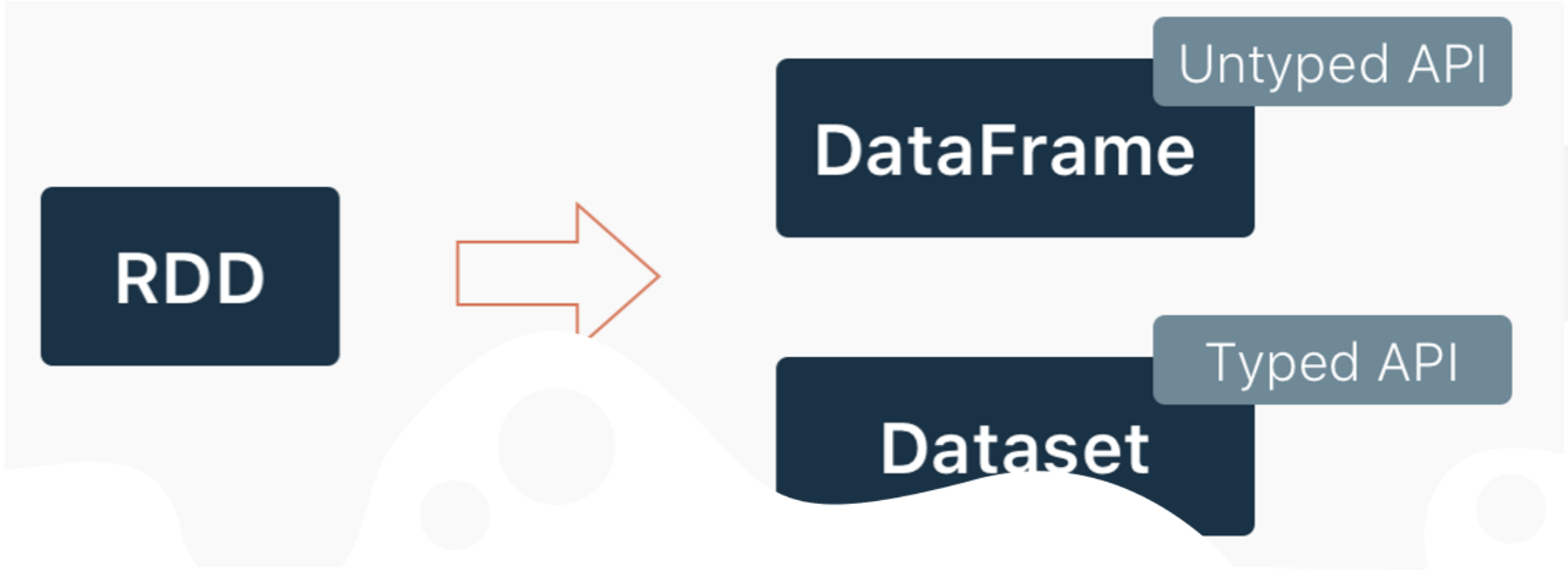
SparkSQL is a little slower than using “raw” RDDs, as it builds a layer on top of them

Apache Spark API

What you need to know…

As Python is not a strongly typed language like Java, DataFrames are more commonly found in PySpark code. DataSets are commonly found in Java.

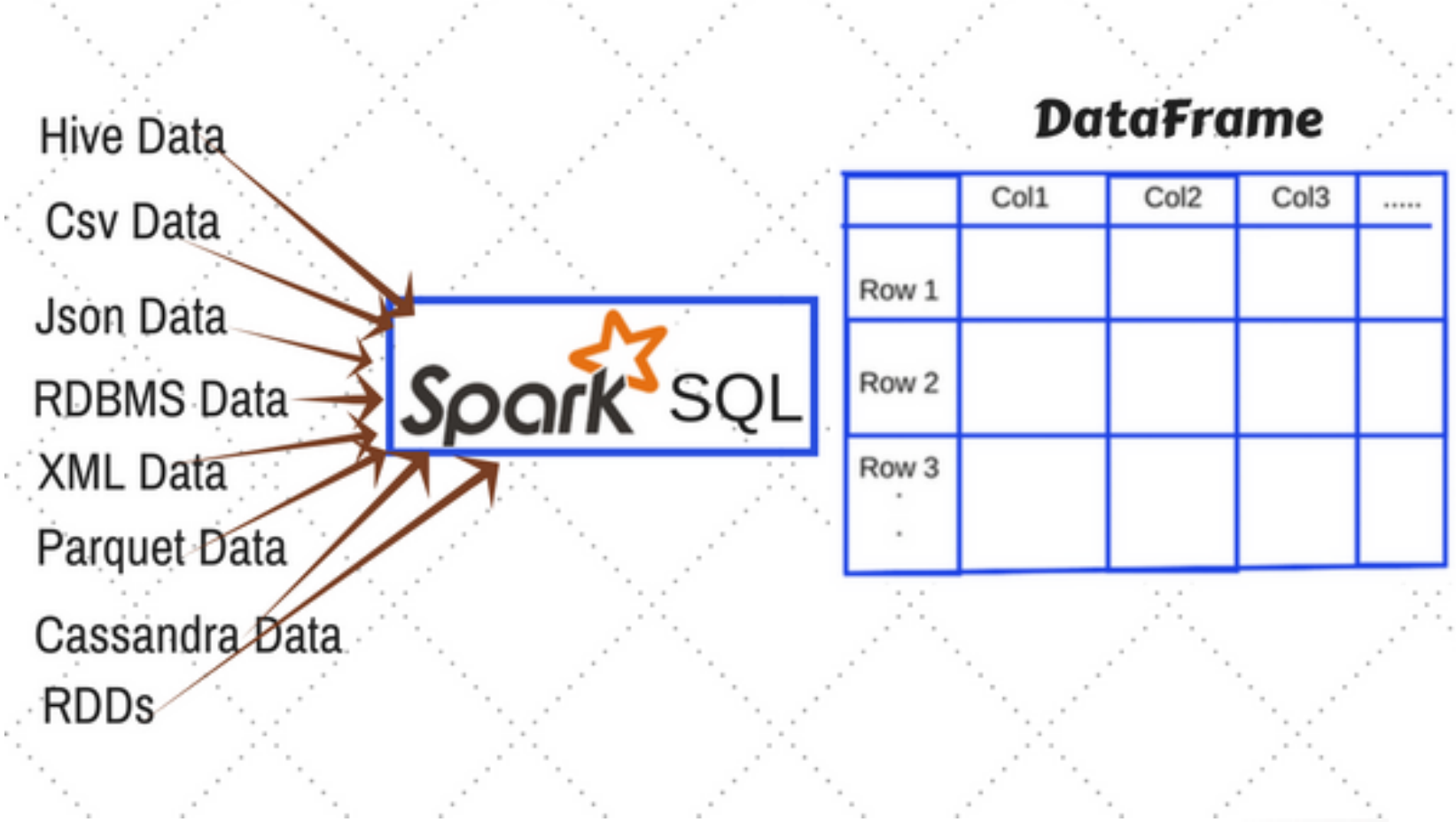
We will focus on DataFrames as we focus on Python.



The architecture of Apache Spark API

Creating a DataFrame in Spark

There are lots of different ways to do it!



Ways to create a DataFrame in Apache Spark

Spark vs Pandas DataFramess

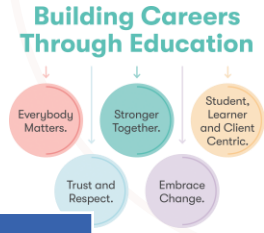
- Dataframe represents a table of data with rows and column
- Dataframe concepts never change in any Programming language
- However, Spark Dataframe and Pandas Dataframe implementations are quite different.

Feature	Spark DataFrame	Pandas DataFrame
Mutability	Immutable	Mutable
Distribution	Distributed	Not distributed
Method behaviour (.count())	Can differ from Pandas	Can differ from Spark
Access to Full API	PySpark API via *DataFrame.to_spark()*	Pandas API via *DataFrame.to_pandas()*
Similarity to pandas-on-Spark DF	Virtually interchangeable	Similar

The key differences and similarities between Spark DataFrames and Pandas DataFrames

BPP

sqlContext vs SparkSession



A comparison…

Feature	SparkContext	SparkSession (2.0 and after)
Package	Org.apache.spark.sql	Org.apache.spark.sql
Availability	Since version 1.0	Since version 2.0
Status	Deprecated in version 2.0	Current
Replacement	Replaced with SparkSession	*N/A*
Code Example	Not provided	from pyspark.sql import SparkSession spark = SparkSession.builder.appName('abc').getOrCreate() data_frame = spark.read.csv('filename.csv', header=True)

Comparing the two main entry points to SQL functionality in Apache Spark: SQLContext and SparkSession

BPP

Takeaway

Spark sessions are wrapper over context giving you a lot more power. The idea is to unify all the context like sql, hive, spark into one. You can get any context from spark session.

This is really useful when you plan to perform traditional RDD transformations, as well as SQL ones, in the same Spark application.

BPP

Inspecting DataFrames

How can this be done?

```
# Read CSV file into table
df = spark.read.option("header",True) \
          .csv("/Users/admin/simple-zipcodes.csv")
df.printSchema()
df.show()
```

```
root
 |-- RecordNumber: string (nullable = true)
 |-- Country: string (nullable = true)
 |-- City: string (nullable = true)
 |-- Zipcode: string (nullable = true)
 |-- State: string (nullable = true)

+------------+-------+--------------------+-------+-----+
|RecordNumber|Country|                City|Zipcode|State|
+------------+-------+--------------------+-------+-----+
|           1|     US|         PARC PARQUE|    704|   PR|
|           2|     US|PASEO COSTA DEL SUR|    704|   PR|
|          10|     US|        BDA SAN LUIS|    709|   PR|
|       49347|     US|                HOLT|  32564|   FL|
|       49348|     US|           HOMOSASSA|  34487|   FL|
|       61391|     US|  CINGULAR WIRELESS|  76166|   TX|
|       61392|     US|          FORT WORTH|  76177|   TX|
|       61393|     US|            FT WORTH|  76177|   TX|
|       54356|     US|         SPRUCE PINE|  35585|   AL|
|       76511|     US|            ASH HILL|  27007|   NC|
|           4|     US|    URB EUGENE RICE|    704|   PR|
|       39827|     US|                MESA|  85209|   AZ|
|       39828|     US|                MESA|  85210|   AZ|
|       49345|     US|            HILLIARD|  32046|   FL|
|       49346|     US|              HOLDER|  34445|   FL|
|           3|     US|      SECT LANAUSSE|    704|   PR|
|       54354|     US|      SPRING GARDEN|  36275|   AL|
|       54355|     US|        SPRINGVILLE|  35146|   AL|
|       76512|     US|           ASHEBORO|  27203|   NC|
|       76513|     US|           ASHEBORO|  27204|   NC|
+------------+-------+--------------------+-------+-----+
```

An example of a Spark DataFrame

Explicit SQL vs Python functions

What is the difference?

```
# Read CSV file into table
spark.read.option("header",True) \
          .csv("/Users/admin/simple-zipcodes.csv") \
          .createOrReplaceTempView("Zipcodes")
```

```
# DataFrame API Select query

df.select("country","city","zipcode","state") \
      .show(5)
```

```
# SQL Select query

spark.sql("SELECT country, city, zipcode, state FROM ZIPCODES")
      .show(5)
```

```
+-------+-------------------+-------+-----+
|country|               city|zipcode|state|
+-------+-------------------+-------+-----+
|     US|         PARC PARQUE|    704|   PR|
|     US|PASEO COSTA DEL SUR|    704|   PR|
|     US|        BDA SAN LUIS|    709|   PR|
|     US|               HOLT|  32564|   FL|
|     US|          HOMOSASSA|  34487|   FL|
+-------+-------------------+-------+-----+
only showing top 5 rows
```

Both yield the above output

BPP

Basic SQL functions

An example…

```
> df.select(”a,”, “b”, “c”…) \
# Filter rows
   •  .where(…).show()
# Order rows
   •  .orderBy(…).show()
# Group data and count it
   •  .groupBy(…).count().show()
# Filter and order, etc.
•      .where.(…).orderBy(…).show()
```

BPP

PySpark Join Types

An example…

```
df1.join(df2, on=None, how=None)
```

Join String (how="…")	Equivalent SQL Join
inner	INNER JOIN
outer, full, fullouter, full_outer	FULL OUTER JOIN
left, leftouter, left_outer	LEFT JOIN
right, rightouter, right_outer	RIGHT JOIN

```
empDF.join(deptDF,empDF.emp_dept_id ==  deptDF.dept_id,"inner") \
     .show(truncate=False)
```

BPP

Exercises

Worksheet

Walkthrough

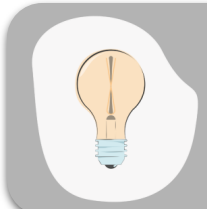
BPP

Practical application

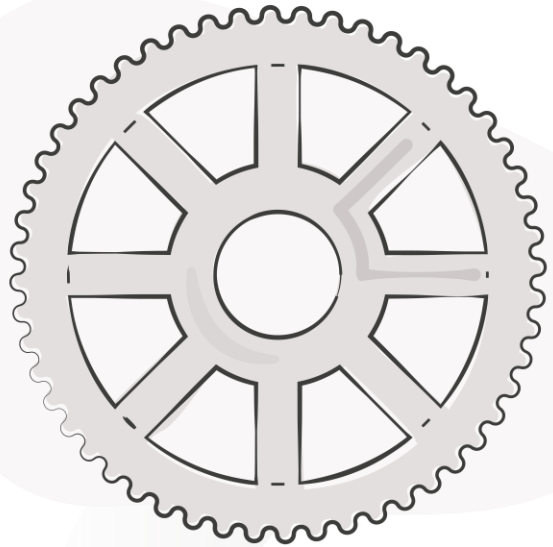
Tutorial code walkthrough –

L5DE M3T8 Code Walkthrough.py

```
# Initialize PySpark
APP_NAME = "Week 2 - Key Value Spark Problems"
import re
def getKBDOC(stringy):
    return re.search(r'KBDOC-[0-9]*',stringy).group()


# If there is no SparkSession, create the environment
try:
  sc and spark
except NameError as e:
  import findspark
  findspark.init()
  import pyspark
  import pyspark.sql

  sc = pyspark.SparkContext()
  spark = pyspark.sql.SparkSession(sc).builder.appName(APP_NAME).getOrCreate()



print("PySpark initiated...")

#Sort data by address and attach via tuple
input = "00210 43.005 -710\n0211 43.0058 -72\n00233 44 -73"
print(input)
value = input.split("\n")
print(value)
rdd = sc.parallelize(value)
output = rdd.map(lambda line: (line.split(' ')[0], (line.split(' ')[1], line.split(' ')[2])))
```



Your tutor will now walk you through the code available here:
Code walkthrough



Practical application

Thank you

Do you have any questions, comments, or feedback?