CSCI 5408, Winter 2017 Assignment 3 Tutorial

Abhinav Kalra

08 February,2017

Objectives

- To learn concepts of Big Data Systems running on Clouds
- To learn data analysis, warehousing and distributed database on real time data
- To learn Real-Time Data Pipelines using apache Spark
- To learn Cloud technology supporting Big Data applications

Introduction to Apache Spark

- A very fast cluster computing system
- Developed in 2009 at UC Berkeley and subsequently open sourced
- Very fast in-memory computations
- APIs for several popular languages such as Java,Scala,Python
- Supports analytics for ML, streaming data and Graph operations

Distributed Processing

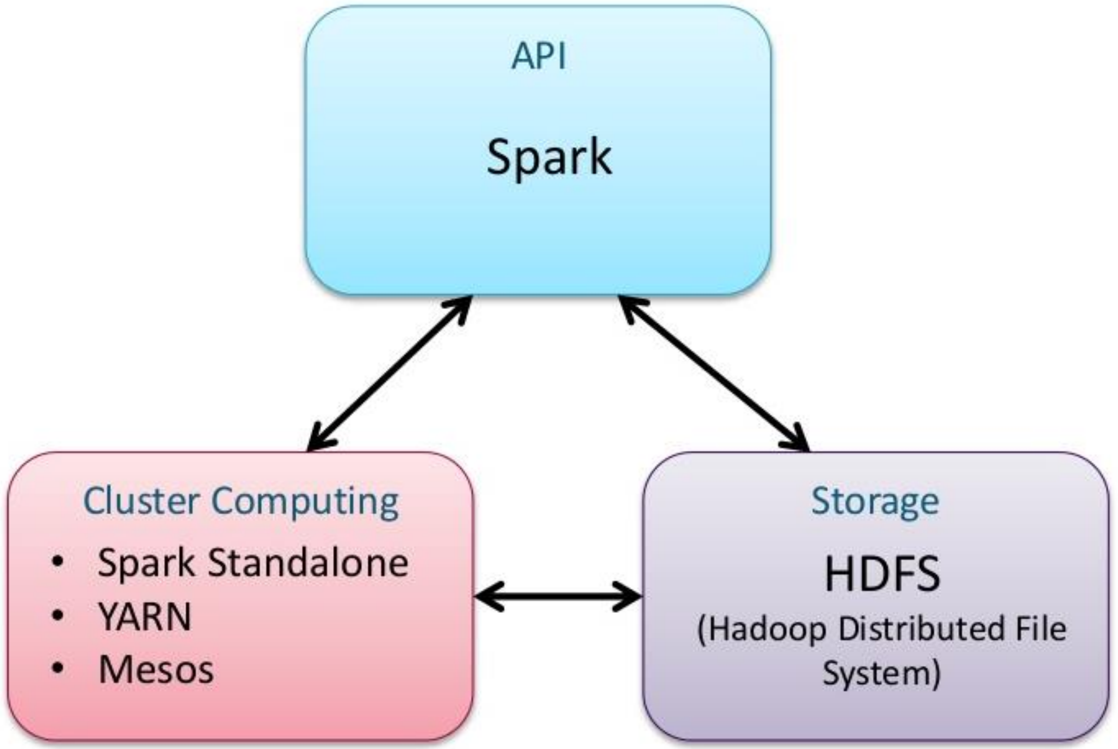


Image courtesy : http://www.slideshare.net/cloudera/spark-devwebinarslides-final

Spark in Cluster Mode

Driver

- Entry point to spark shell
- SparkContext is created here
- Schedule tasks and control execution
- Handle metadata and split DAG for operations on data
- SparkContext: enables driver to interact with resource manager

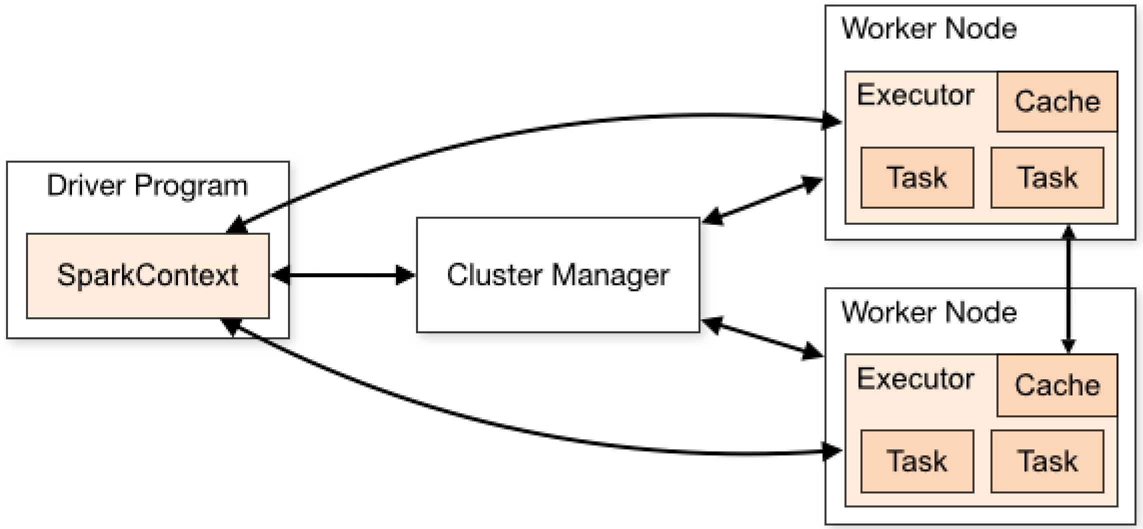
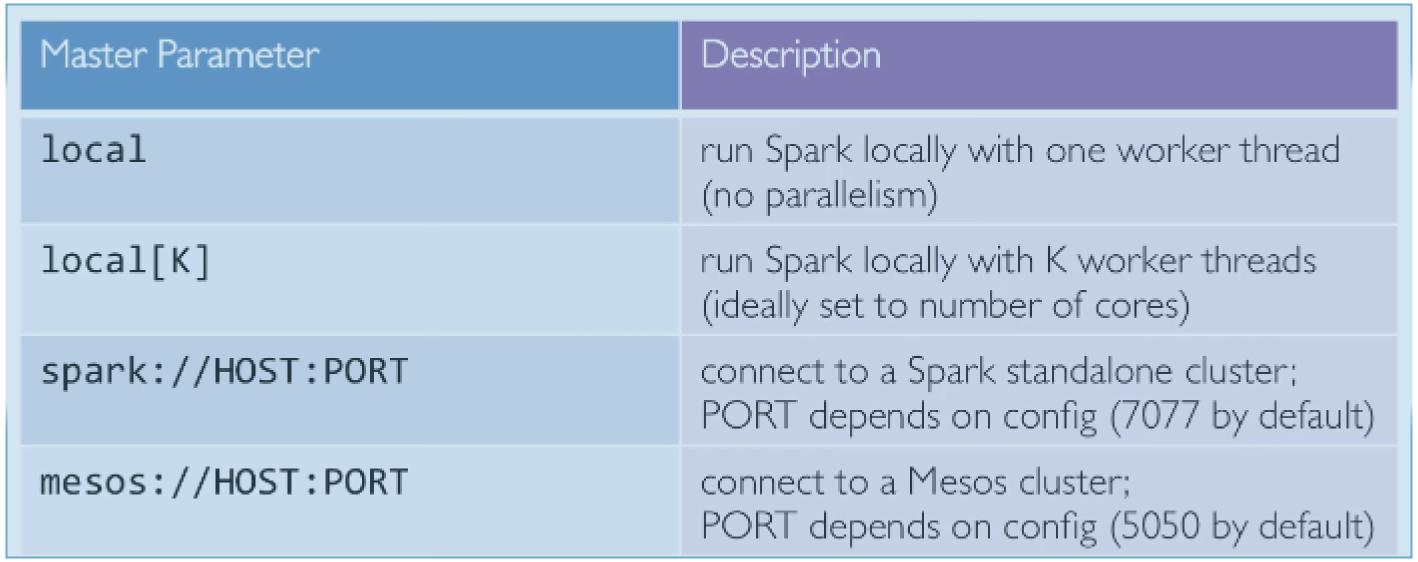


Image courtesy : http://spark.apache.org

Spark Context

- *Master* parameter in Spark Context

```python
# Spark Context in Python

from pyspark import SparkContext
sc = SparkContext("local", "Application")
```



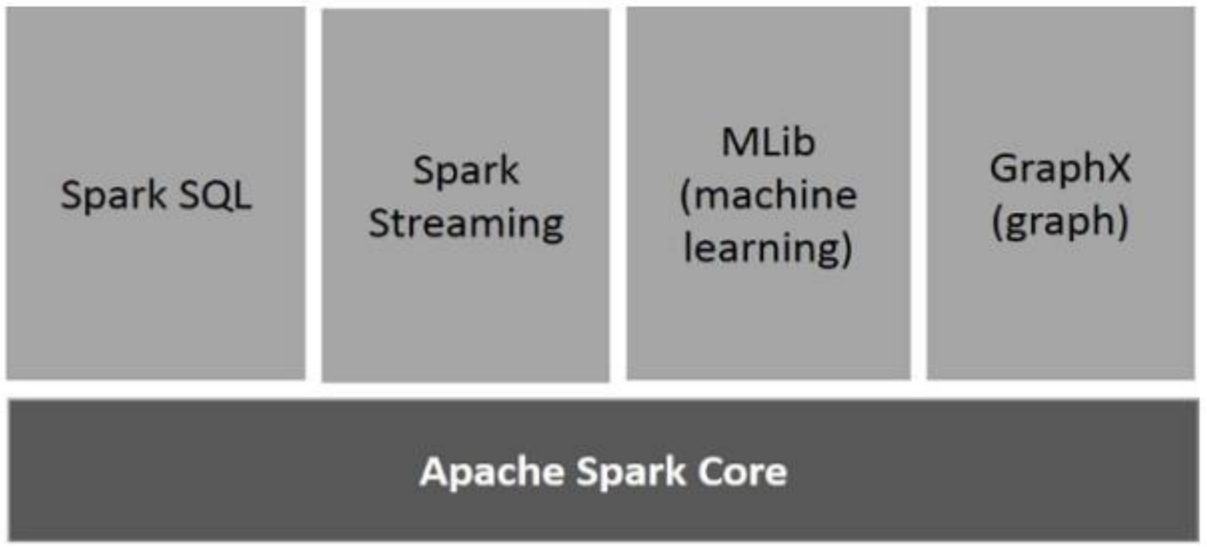
Master Parameter	Description
`local`	run Spark locally with one worker thread (no parallelism)
`local[K]`	run Spark locally with K worker threads (ideally set to number of cores)
`spark://HOST:PORT`	connect to a Spark standalone cluster; PORT depends on config (7077 by default)
`mesos://HOST:PORT`	connect to a Mesos cluster; PORT depends on config (5050 by default)

DALHOUSIE UNIVERSITY *Inspiring Minds*

Image courtesy: http://x-wei.github.io/sparkmooc_note_lec4

Components of Spark

- SparkSQL: Support for structured and semi-structured data
- Spark Streaming : Analytics on streaming data eg. Kafka,Flume
- Mlib : Packages for Machine Learning
- GraphX: Distributed graph processing framework



DALHOUSIE UNIVERSITY *Inspiring Minds*

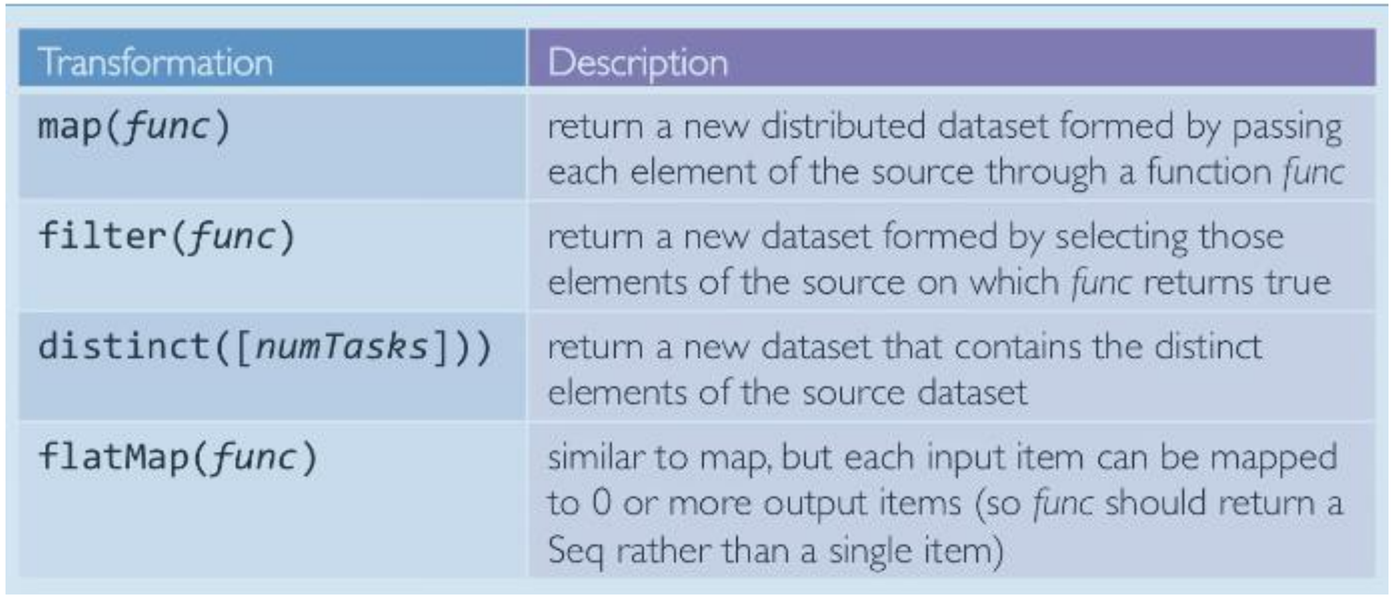
Image : https://www.tutorialspoint.com/apache_spark

Resilient Distributed Datasets

- Resilient : Immutable and partitioned , rebuilt on failure
- Distributed : stored in memory across cluster nodes
- Dataset : can hold data from different sources. Held in memory or cached to storage
- Fundamental data unit in Spark
- Operations :
  - Transformation
  - Actions

Transformations

- Create a new dataset from existing ones
- Lazy, executed when actions are run on it



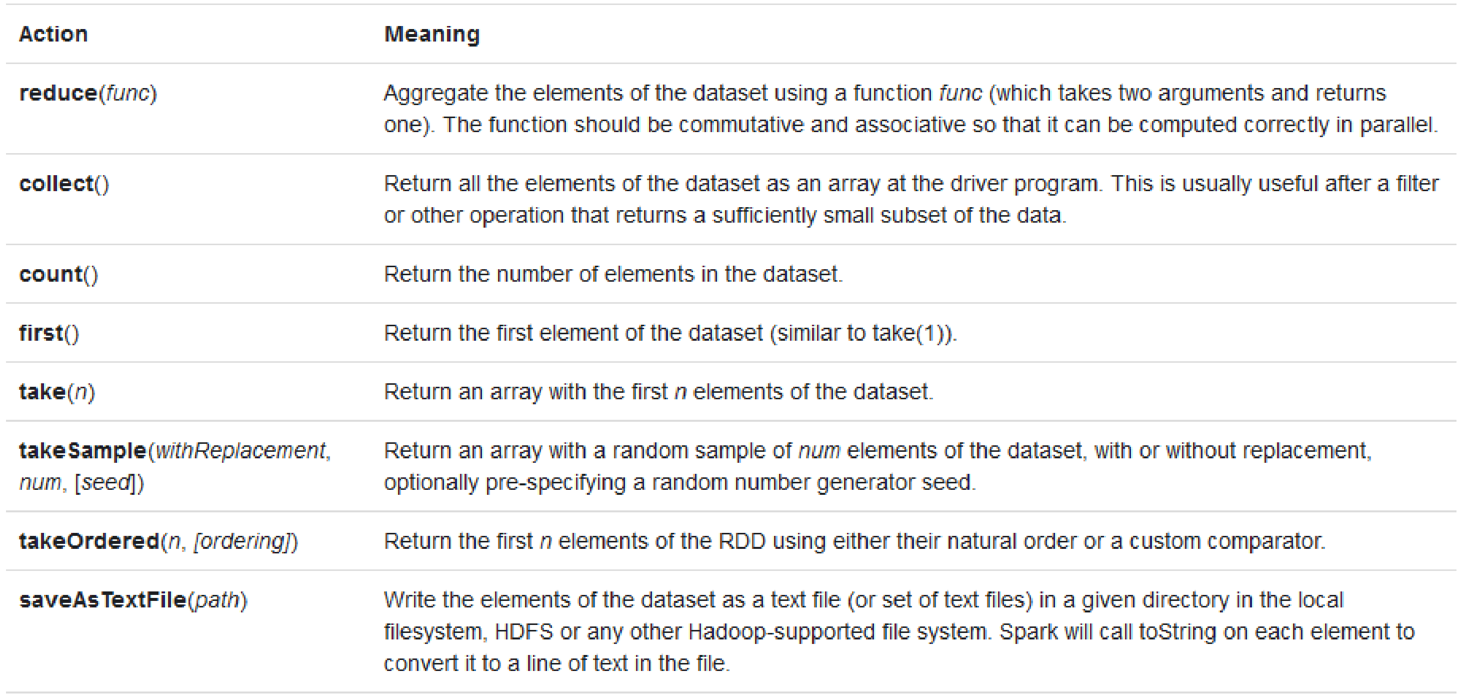
Transformation	Description
`map(`*func*`)`	return a new distributed dataset formed by passing each element of the source through a function *func*
`filter(`*func*`)`	return a new dataset formed by selecting those elements of the source on which *func* returns true
`distinct([`*numTasks*`]))`	return a new dataset that contains the distinct elements of the source dataset
`flatMap(`*func*`)`	similar to map, but each input item can be mapped to 0 or more output items (so *func* should return a Seq rather than a single item)

DALHOUSIE UNIVERSITY Inspiring Minds

Image courtesy: http://x-wei.github.io/sparkmooc_note_lec4

Actions

- Actions return values from Dataset

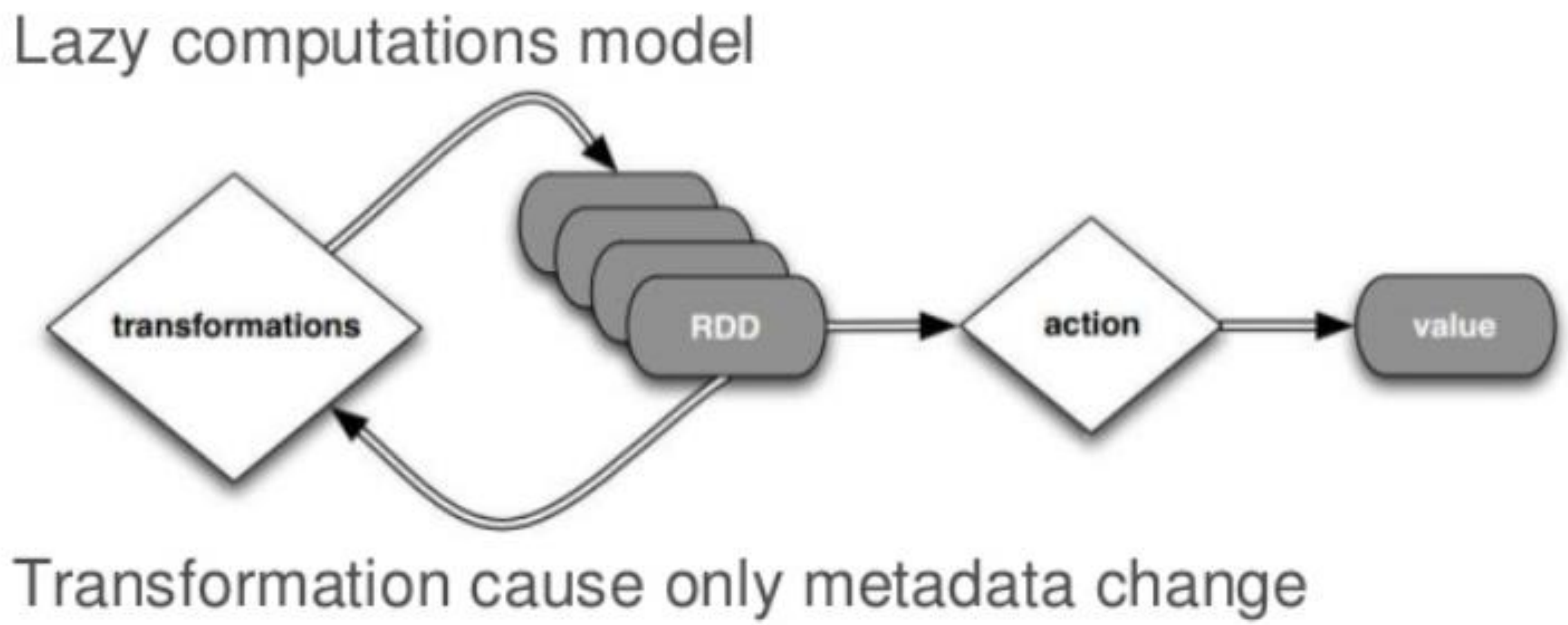


Action	Meaning
reduce(*func*)	Aggregate the elements of the dataset using a function *func* (which takes two arguments and returns one). The function should be commutative and associative so that it can be computed correctly in parallel.
collect()	Return all the elements of the dataset as an array at the driver program. This is usually useful after a filter or other operation that returns a sufficiently small subset of the data.
count()	Return the number of elements in the dataset.
first()	Return the first element of the dataset (similar to take(1)).
take(*n*)	Return an array with the first *n* elements of the dataset.
takeSample(*withReplacement*, *num*, [*seed*])	Return an array with a random sample of *num* elements of the dataset, with or without replacement, optionally pre-specifying a random number generator seed.
takeOrdered(*n*, *[ordering]*)	Return the first *n* elements of the RDD using either their natural order or a custom comparator.
saveAsTextFile(*path*)	Write the elements of the dataset as a text file (or set of text files) in a given directory in the local filesystem, HDFS or any other Hadoop-supported file system. Spark will call toString on each element to convert it to a line of text in the file.

DALHOUSIE UNIVERSITY *Inspiring Minds*

From :http://spark.apache.org/docs/latest/programming-guide.htm

RDD



DALHOUSIE UNIVERSITY *Inspiring Minds*

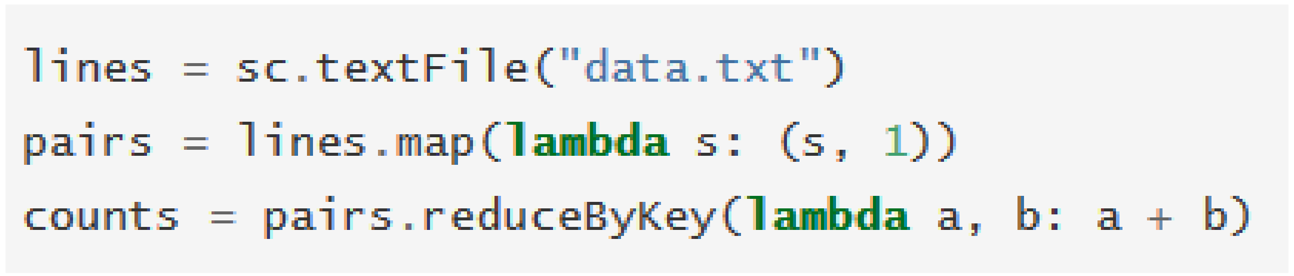
Image : http://www.slideshare.net/AGrishchenko/apache-spark-architecture

Creating RDD

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

# Turn a Scala collection into an RDD
>sc.parallelize(List(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)
```

Working with Key-Value pairs

- Data in RDD can be represented as (Key,Value) Pairs
- Keys can be simple or complex



```
lines = sc.textFile("data.txt")
pairs = lines.map(lambda s: (s, 1))
counts = pairs.reduceByKey(lambda a, b: a + b)
```

From : http://spark.apache.org/docs/latest/programming-guide.htm

Transformations on RDD

```
>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}
```

DALHOUSIE UNIVERSITY *Inspiring Minds*

Passing functions

- Functions can also be passed to the driver

```
"""MyScript.py"""
if __name__ == "__main__":
    def myFunc(s):
        words = s.split(" ")
        return len(words)

    sc = SparkContext(...)
    sc.textFile("file.txt").map(myFunc)
```

DALHOUSIE UNIVERSITY *Inspiring Minds*

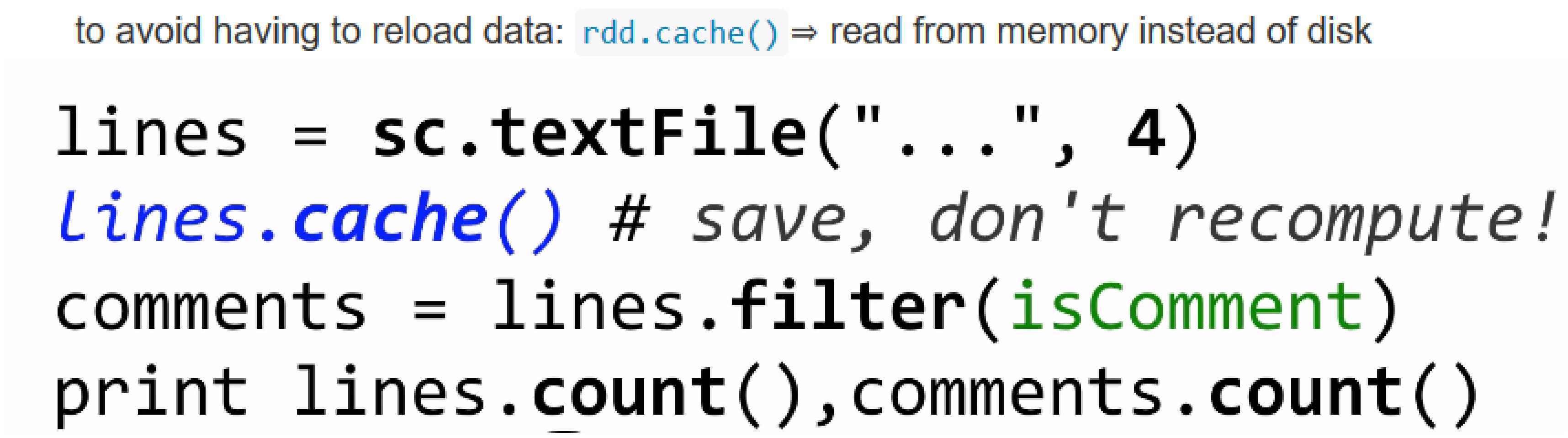
From : http://spark.apache.org/docs/latest/programming-guide.htm

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")
```

stanford.edu/~rezab/sparkworkshop/slides/holden.pdf

Caching RDD



to avoid having to reload data: `rdd.cache()` ⇒ read from memory instead of disk

```
lines = sc.textFile("...", 4)
lines.cache() # save, don't recompute!
comments = lines.filter(isComment)
print lines.count(),comments.count()
```

DALHOUSIE UNIVERSITY
Inspiring Minds

From: http://x-wei.github.io/sparkmooc_note_lec4

MapReduce Example: Word Count

Input Data

```
the cat sat on the mat
the aardvark sat on the sofa
```

?

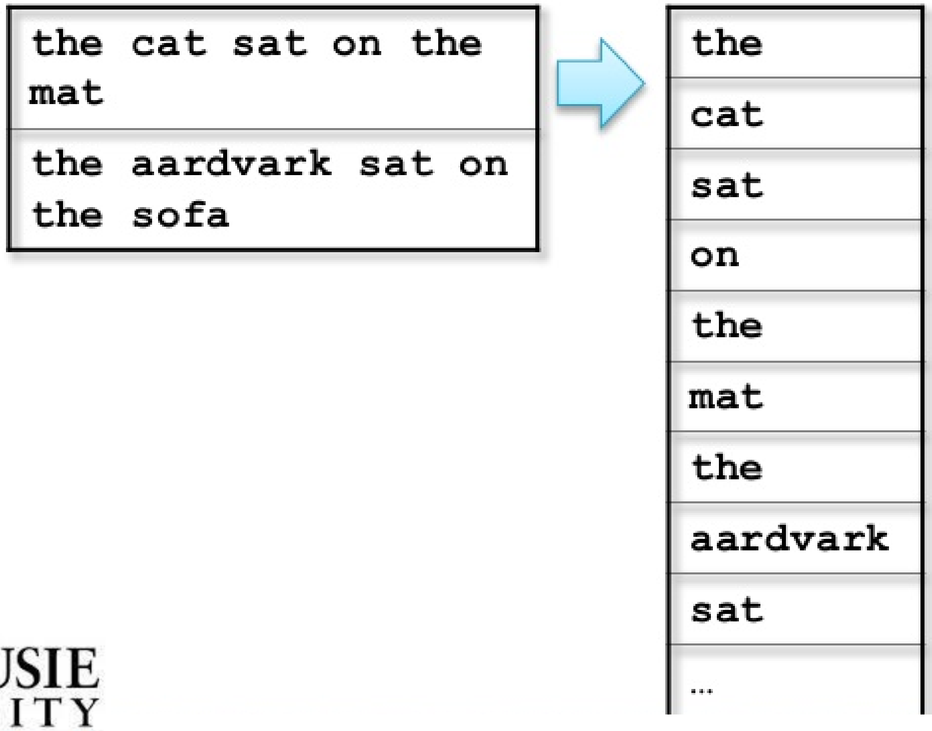
Result

aardvark	1
cat	1
mat	1
on	2
sat	2
sofa	1
the	4

DALHOUSIE UNIVERSITY
Inspiring Minds

Example: Word Count

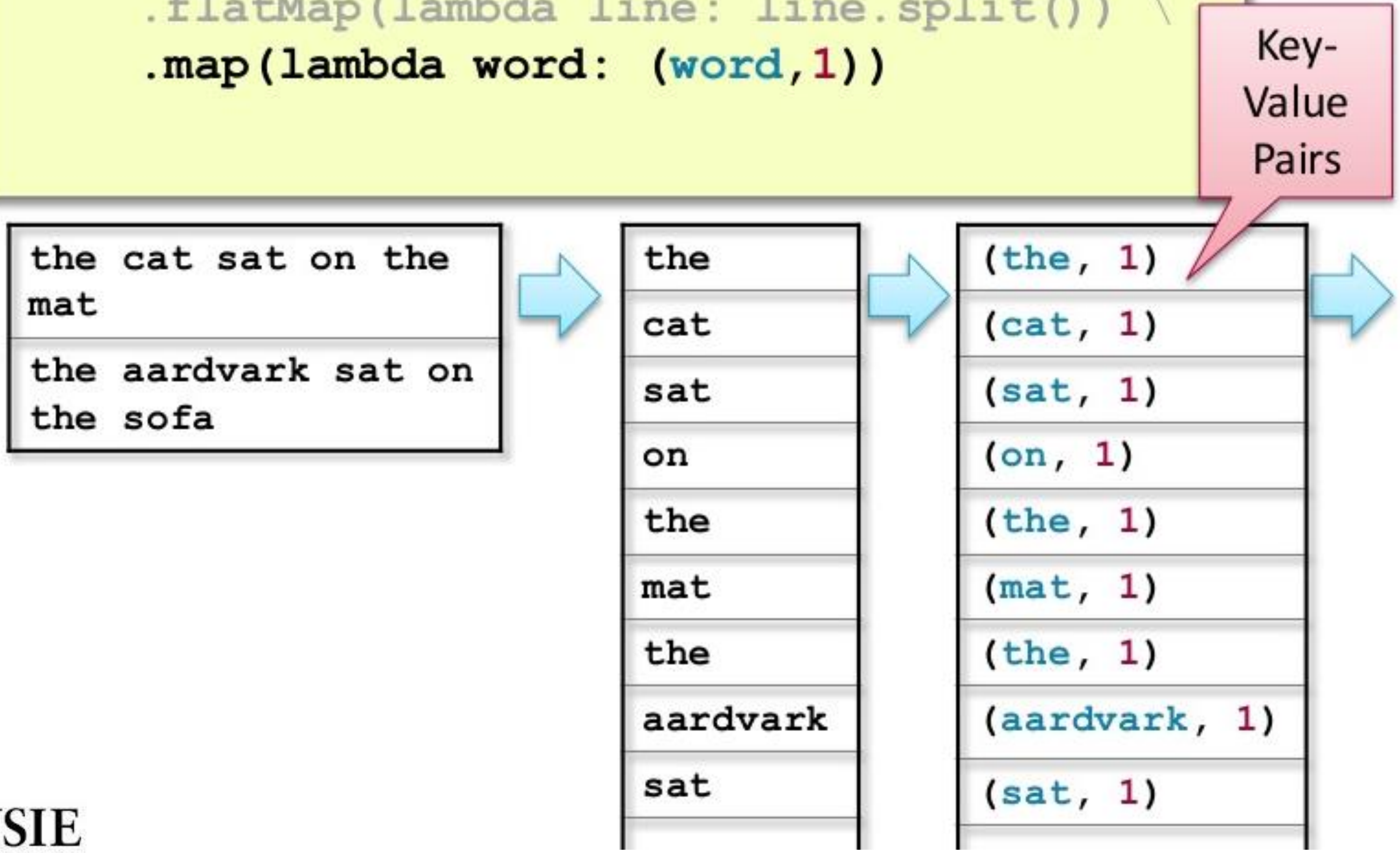
```
> counts = sc.textFile(file) \
     .flatMap(lambda line: line.split())
```



DALHOUSIE UNIVERSITY
Inspiring Minds

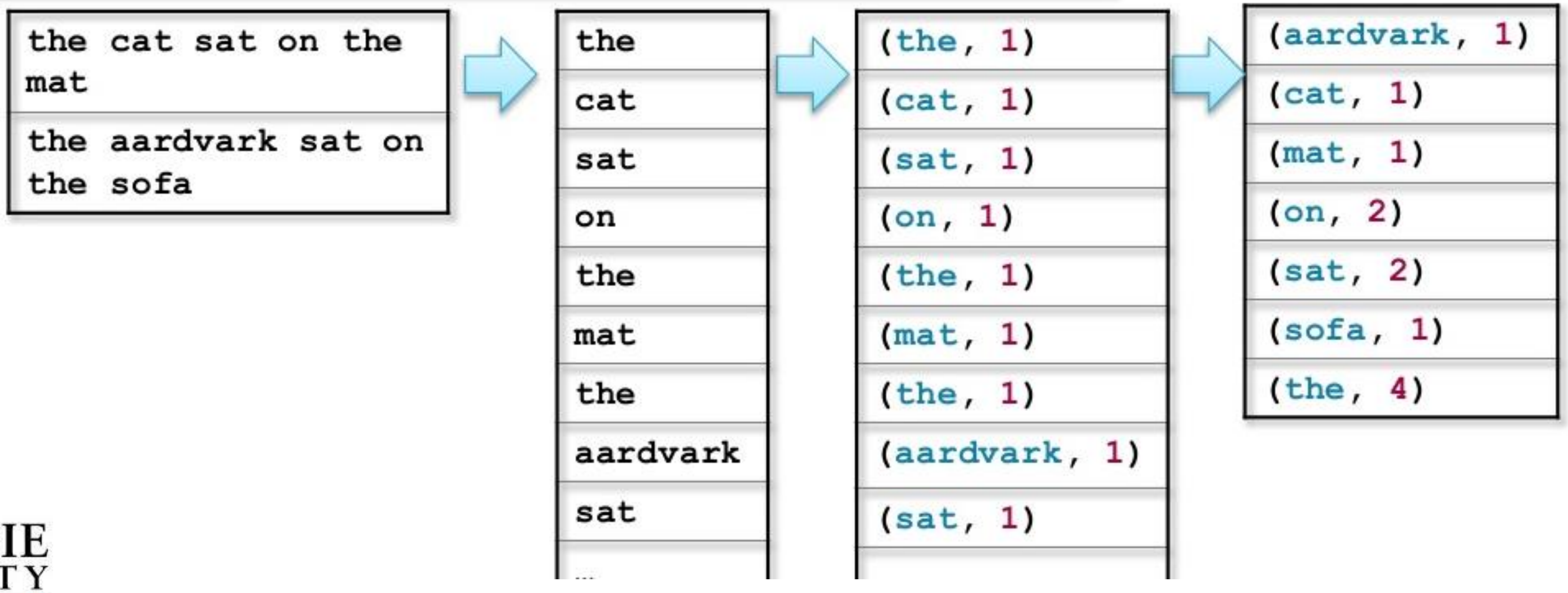
Example: Word Count

```
> counts = sc.textFile(file) \
    .flatMap(lambda line: line.split()) \
    .map(lambda word: (word,1))
```



Example: Word Count

```
> counts = sc.textFile(file) \
    .flatMap(lambda line: line.split()) \
    .map(lambda word: (word,1)) \
    .reduceByKey(lambda v1,v2: v1+v2)
```



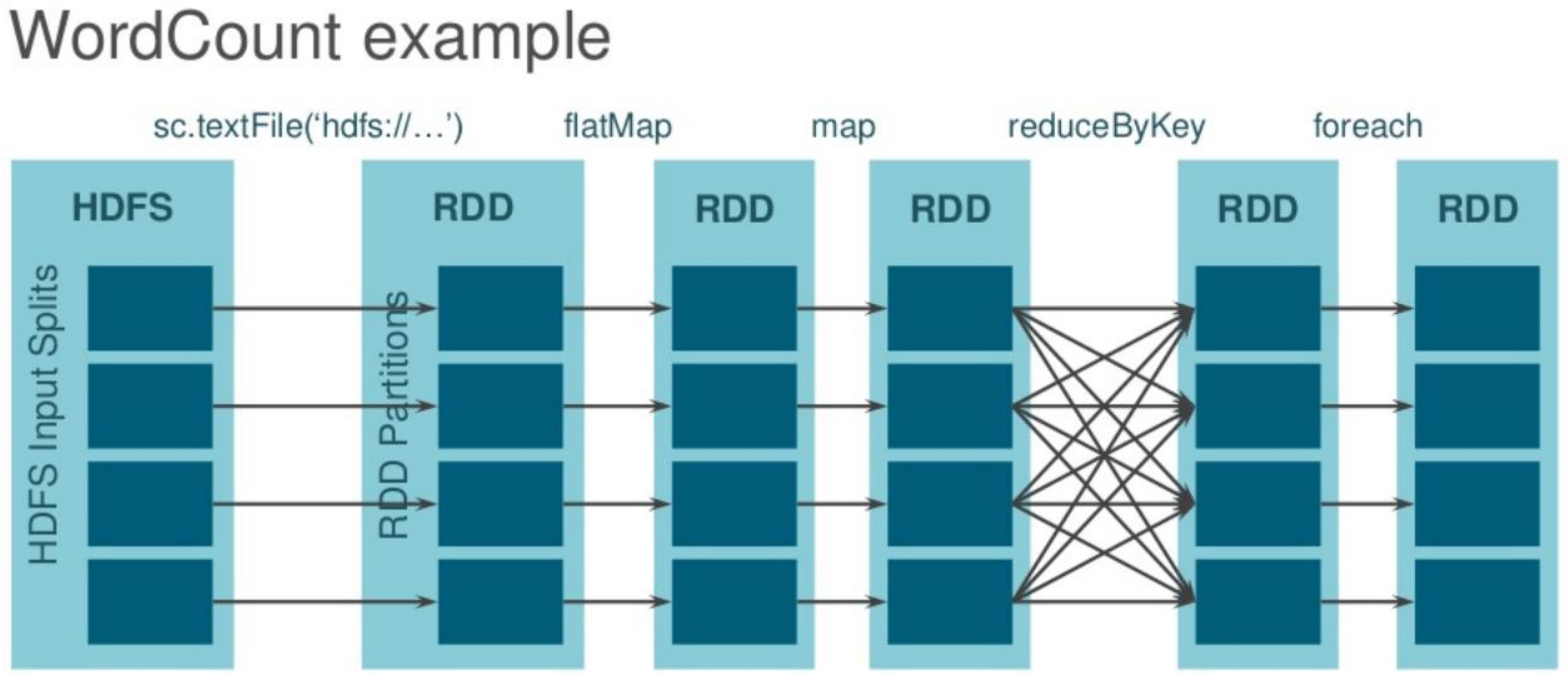
```
the cat sat on the
mat

the aardvark sat on
the sofa
```

```
the
cat
sat
on
the
mat
the
aardvark
sat
...
```

```
(the, 1)
(cat, 1)
(sat, 1)
(on, 1)
(the, 1)
(mat, 1)
(the, 1)
(aardvark, 1)
(sat, 1)
```

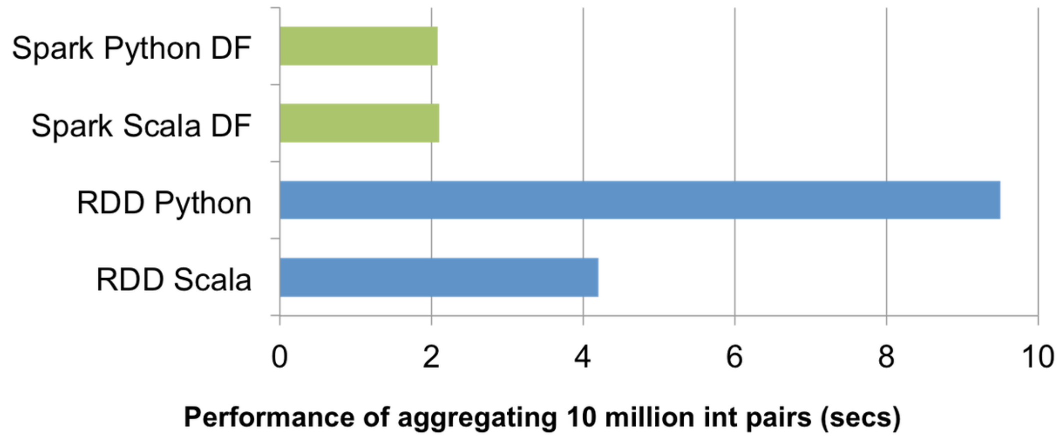
```
(aardvark, 1)
(cat, 1)
(mat, 1)
(on, 2)
(sat, 2)
(sofa, 1)
(the, 4)
```

DAG


WordCount example
sc.textFile('hdfs://...')
flatMap
map
reduceByKey
foreach
HDFS
RDD
RDD
RDD
RDD
RDD
HDFS Input Splits
RDD Partitions

Spark DataFrames

- An RDD with schema
- Organized with named columns
- Data stored as row-column format
- Incorporate SQL syntax to query DataFrames



DALHOUSIE UNIVERSITY
Inspiring Minds

From: https://databricks.com/blog/2015/02/17/introducing-dataframes-in-spark-for-large-scale-data

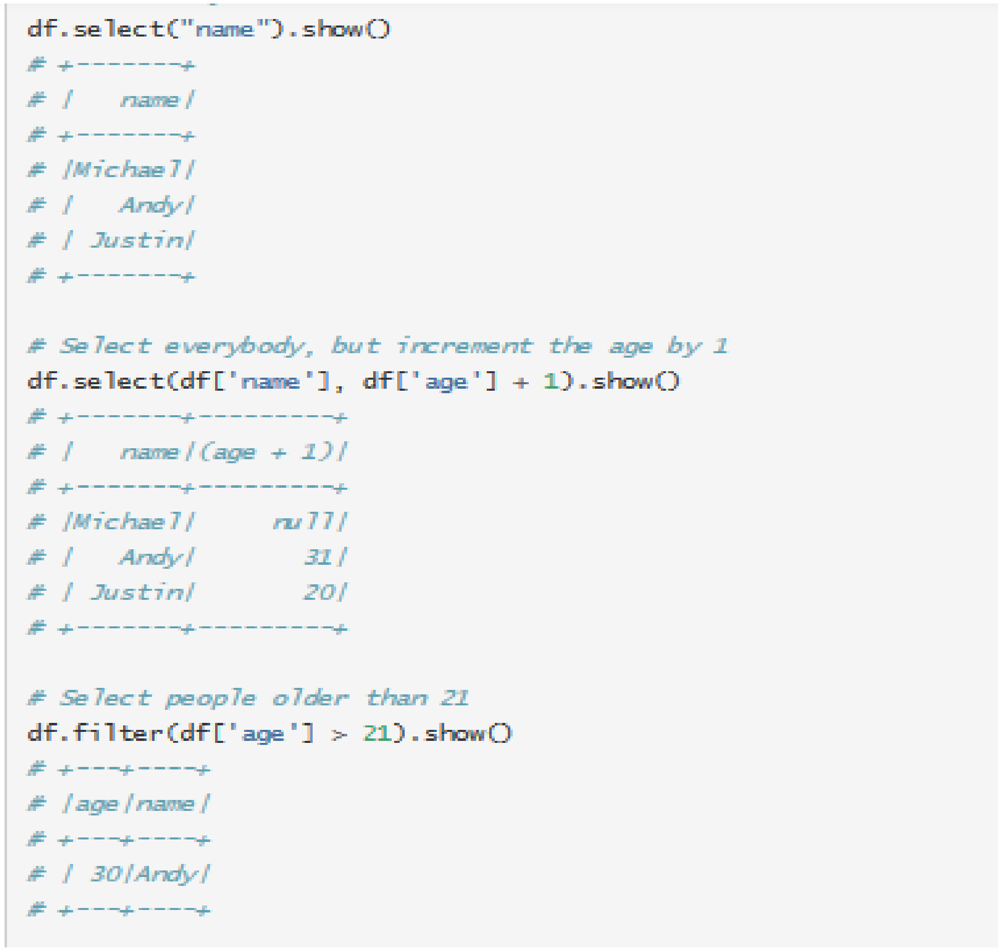
SparkSQL

- For processing structured data
- Utilizes Spark DataFrames
- Load and process data from several disparate sources eg, JSON,Hive tables, Cassandra, mySql etc.

- SQLContext/SparkSession : Entry point for Spark

SQLContext kept for backward compatibility

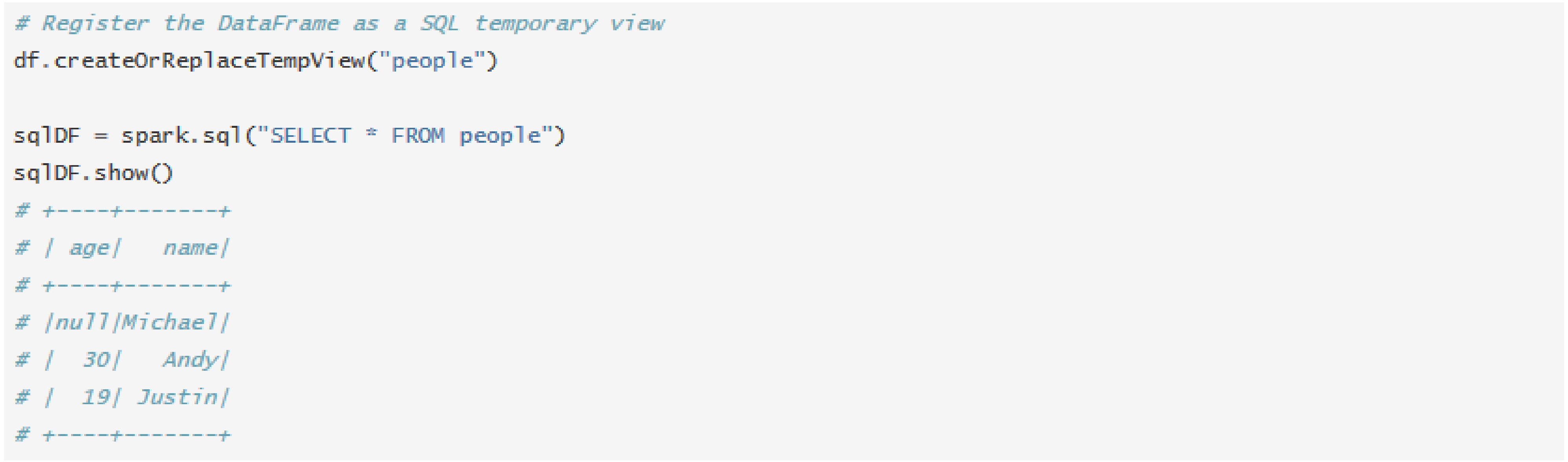
Basic DataFrame Operations



```
df.select("name").show()
# +-------+
# |   name|
# +-------+
# |Michael|
# |   Andy|
# | Justin|
# +-------+

# Select everybody, but increment the age by 1
df.select(df['name'], df['age'] + 1).show()
# +-------+---------+
# |   name|(age + 1)|
# +-------+---------+
# |Michael|     null|
# |   Andy|       31|
# | Justin|       20|
# +-------+---------+

# Select people older than 21
df.filter(df['age'] > 21).show()
# +---+----+
# |age|name|
# +---+----+
# | 30|Andy|
# +---+----+
```

http://spark.apache.org/docs/latest/sql-programming-guide.html#running-sql-queries-programmatically

Running SQL queries



```
# Register the DataFrame as a SQL temporary view
df.createOrReplaceTempView("people")

sqlDF = spark.sql("SELECT * FROM people")
sqlDF.show()
# +----+-------+
# | age|   name|
# +----+-------+
# |null|Michael|
# |  30|   Andy|
# |  19| Justin|
# +----+-------+
```

http://spark.apache.org/docs/latest/sql-programming-guide.html#running-sql-queries-programmatically

Downloading Spark and running application

- Spark can be downloaded from :
  - http://spark.apache.org/downloads.html
- Ensure following is installed on Linux: Python(v2.7.x),JRE and Scala
- Application can be run using Spark-Submit script

```
./bin/spark-submit \
  --class <main-class> \
  --master <master-url> \
  --deploy-mode <deploy-mode> \
  --conf <key>=<value> \
  ... # other options
  <application-jar> \
  [application-arguments]
```

- Python file execution: */spark-dir/bin/spark-submit application.py*

DALHOUSIE UNIVERSITY *Inspiring Minds*

Spark with iPython/Jupyter

- To install iPython
  - Sudo pip install ipython

- Running Spark with iPython on Linux
  - *PYSPARK_DRIVER_PYTHON="/usr/bin/ipython" /path/to/spark/bin/pyspark --master local[4]*

Assignment Tasks

1. Download Apache Spark
2. Download the dataset
3. Write an application for Spark operations
4. Preprocess and clean data for wordcount in application
5. Using Spark , perform the wordcount operations
6. Save the output to file
7. Use the babynames dataset and perform the operations in Spark

DALHOUSIE UNIVERSITY *Inspiring Minds*

Assignment Tasks

8. Use NYPD MV collisions dataset and preprocess and clean
9. Perform provided operations in Spark
9. Save all outputs to file
10. Calculate time required to run each query

Report requirements

- Task description and configuration steps performed
- The operations run on data along with description and execution times
- Syntax for application queries and Sample outputs
- Summary and observations
- Also include:
  - Source code
  - Output files
  - Readme file

DALHOUSIE UNIVERSITY *Inspiring Minds*

Spark Guides and documentation

- http://spark.apache.org/docs/2.0.1/quick-start.html
- http://spark.apache.org/docs/2.0.1/programming-guide.html
- http://spark.apache.org/docs/2.0.1/api/python/index.html
- http://spark.apache.org/examples.html

DALHOUSIE UNIVERSITY *Inspiring Minds*

References

- http://spark.apache.org/docs/latest/api/python/index.html
- https://databricks.com/blog/2015/02/17/introducing-dataframes-in-spark-for-large-scale-data-science.html
- http://www.tutorialspoint.com/apache_spark/