Carnegie Mellon University

Machine Learning with Large Datasets (Fall 2022)

Recitation 1

September 2, 2022

Special thanks to **Tian Li** and **Prof. Heather Miller** who developed some of the material covered today.

Agenda

1. Introduction to Databricks

- Registration on Databricks (Community Edition)
- Uploading Notebooks
- Creation of Cluster

2. Spark Basics

- Spark Architecture
- Execution of Spark Program
- Spark APIs
- Lambda Functions
- Examples
- Cache/Persistence

3. Programming Demo

Homework Setup

Programming Assignments

- **5** programming assignments - **4** using PySpark, **1** using Tensorflow (Subject to Change)
- PySpark assignments can be completed on Databricks (Community).
- More details about the tensorflow assignments will be shared later.
- For autograding, submit assignments on Gradescope.
- Detailed instructions on how to submit assignment for grading will be in the writeup.

Written Assignments

- Need to be directly submitted to Gradescope.

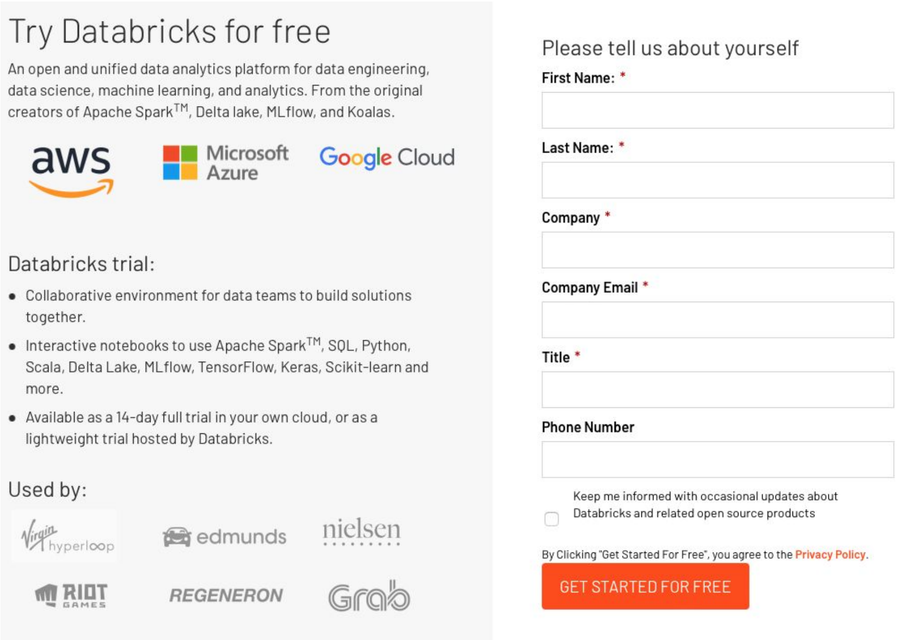
More info on Course Website - https://10605.github.io/

To-Dos for students

- Register for a free community version of Databricks.
- Download the assignment and import the Jupyter Notebooks on Databricks.
- Configure the environment according to the instructions in the writeup.
  - Creating a cluster.
  - Installing a third-party package.
- After all *local* tests pass, hand-in the assignment on Gradescope for grading.
  - Please note that you get points if the tests on the grader pass.
  - The local tests exists to help you develop and test your code.

Registration

https://databricks.com/try-databricks



Try Databricks for free

An open and unified data analytics platform for data engineering, data science, machine learning, and analytics. From the original creators of Apache Spark™, Delta lake, MLflow, and Koalas.

Databricks trial:

- Collaborative environment for data teams to build solutions together.
- Interactive notebooks to use Apache Spark™, SQL, Python, Scala, Delta Lake, MLflow, TensorFlow, Keras, Scikit-learn and more.
- Available as a 14-day full trial in your own cloud, or as a lightweight trial hosted by Databricks.

Used by:

Please tell us about yourself

First Name: *

Last Name: *

Company *

Company Email *

Title *

Phone Number

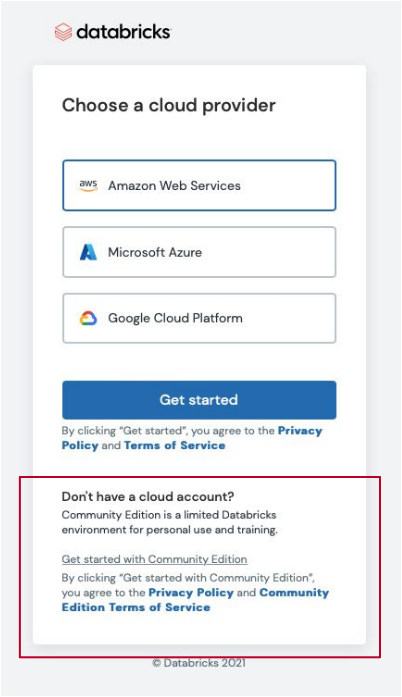
Keep me informed with occasional updates about Databricks and related open source products

By Clicking "Get Started For Free", you agree to the Privacy Policy.

GET STARTED FOR FREE

Registration

Please only choose the community edition and NOT a cloud provider.

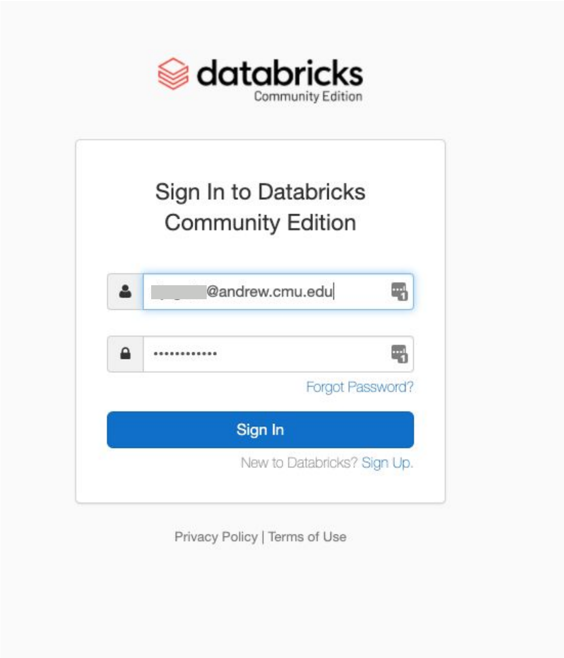


Carnegie Mellon University

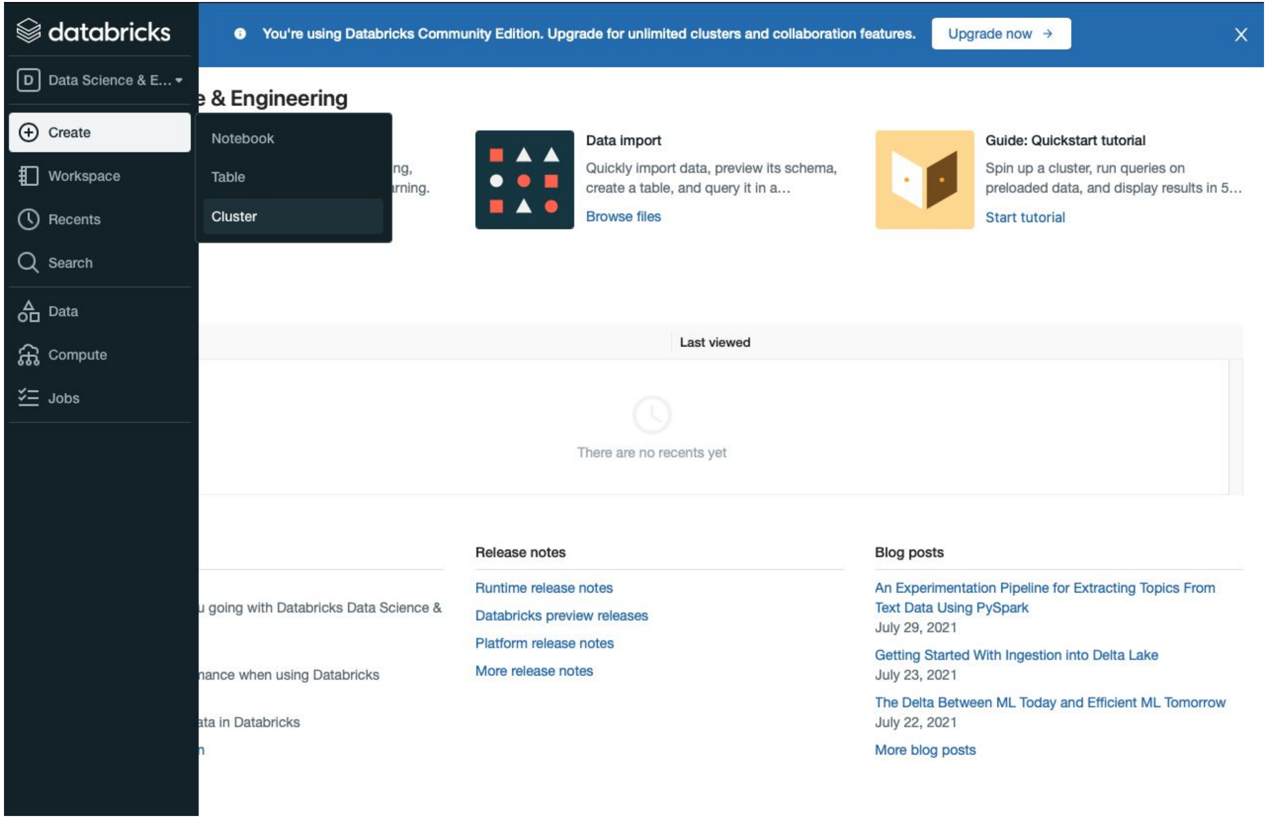
Login

Login to the community edition at

https://community.cloud.databricks.com/login.html

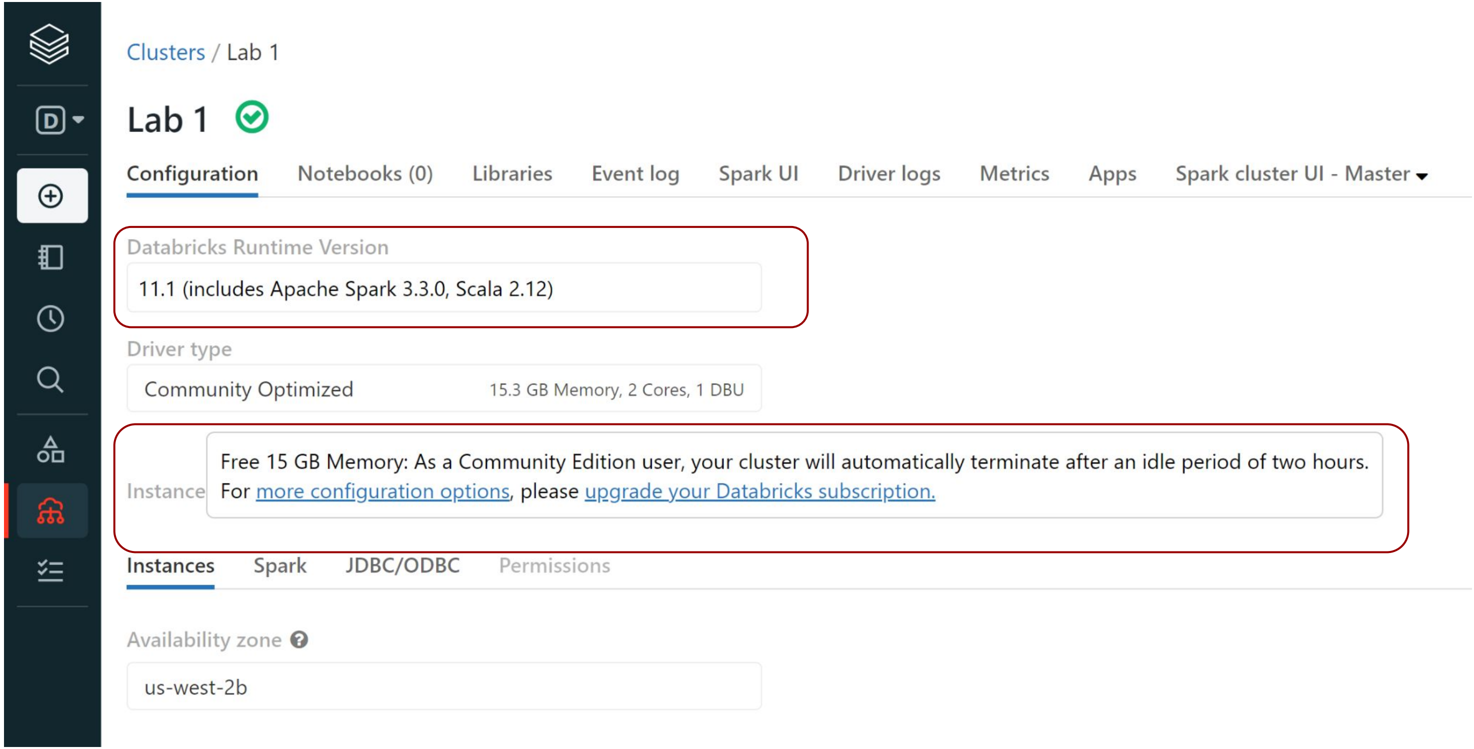


Creating a Cluster

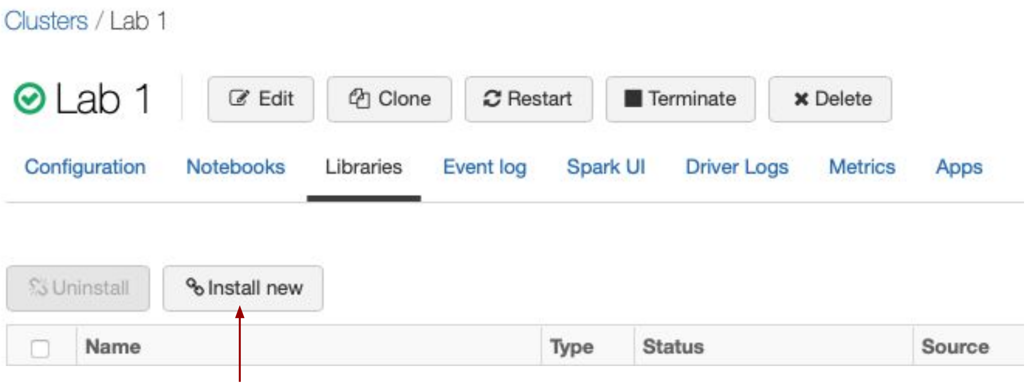


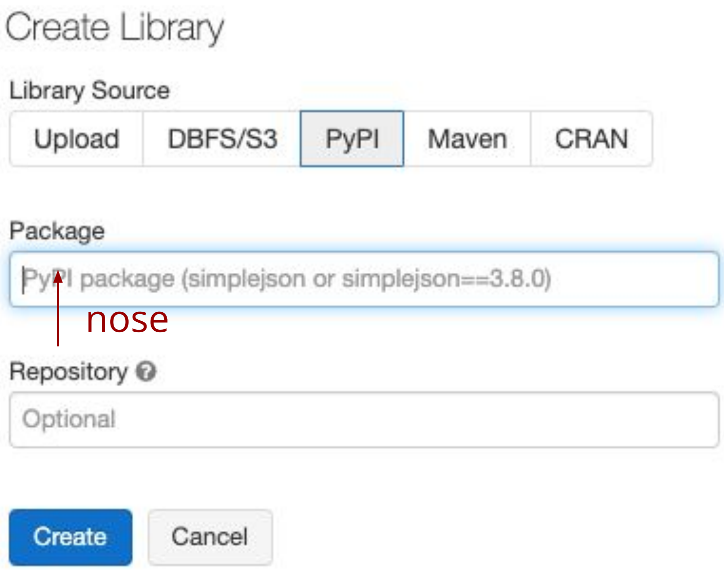
Carnegie Mellon University

Creating a Cluster



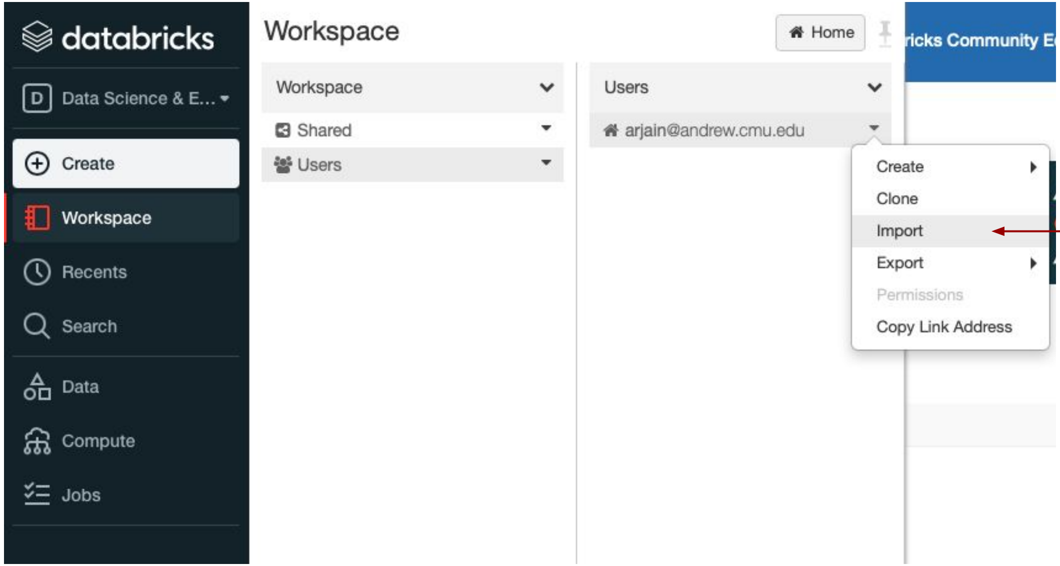
Installing Required Packages

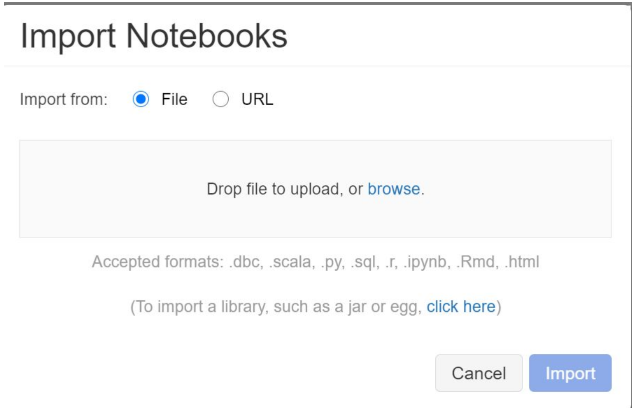




Carnegie Mellon University

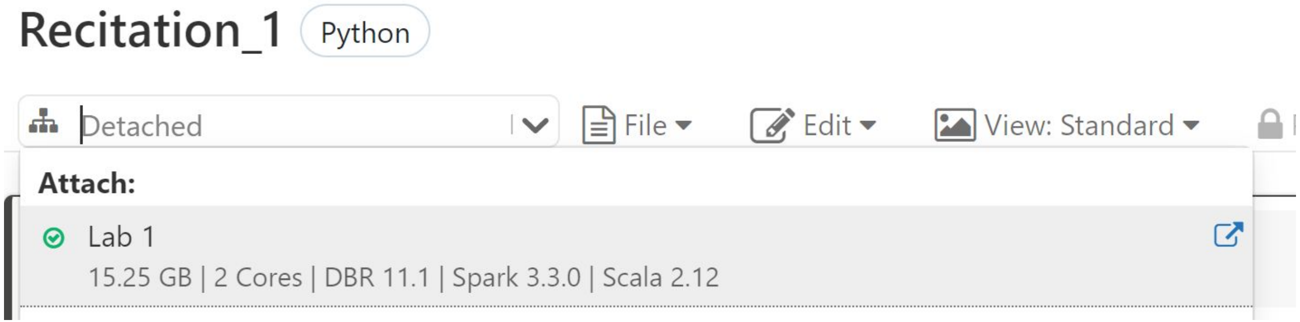
Uploading/Importing the Homework





Interacting with Notebooks

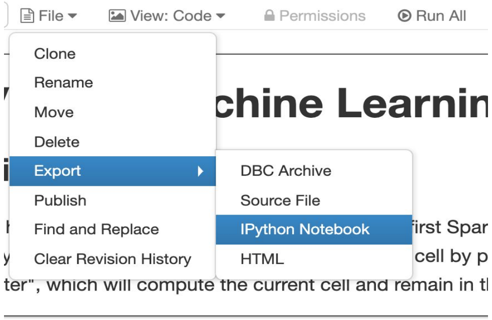
- Attach notebook to the cluster



Carnegie Mellon University

Exporting the Homework

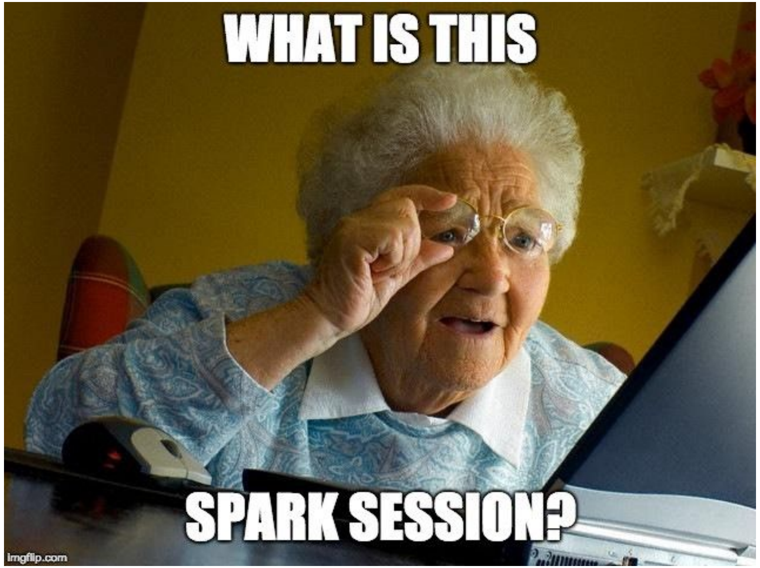
- After completion of assignment, export the notebook as an iPython Notebook and submit to Gradescope for grading.



Important notes about clusters

- Please make sure that the following versions are used:
  - **Spark:** 3.3.0
  - **Scala:** 2.12
  - **Runtime:** 11.1
- For all coding, please use Python3 syntax.
- Launching a cluster can take some time (a few minutes).
- The cluster status should be “active” before running any cells.
- Community edition only allows for a single node cluster which should be sufficient for completion of the homeworks.
- Community edition clusters will automatically terminate after an idle period of two hours.

APACHE Spark™



PySpark

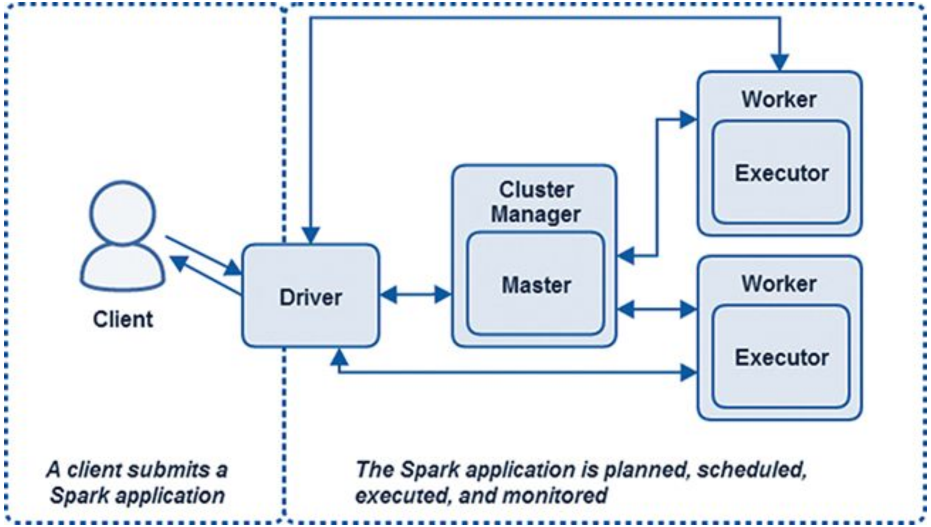
- We are using the Python programming interface to Spark (pySpark)
- Where can I run PySpark code?
  - Locally on your laptop!
  - Databricks
  - Zeppelin Notebooks
  - many more …

PySpark (local installation)

- To get HWs running locally, you'll need to install:
  - VSCode
    - Jupyter notebook extension
  - Python3
    - pip install ipykernel
    - pip install pyspark==3.1.2
    - pip install findspark
    - pip install nose
  - Java8
- Note: displaying math equations might be buggy
- We won't be able to support / debug local installations

Carnegie Mellon University

Spark Architecture



- **Driver program:** This is the node you're interacting with when you're writing Spark programs. Creates Spark Context to establish the connection to Spark Execution Environment

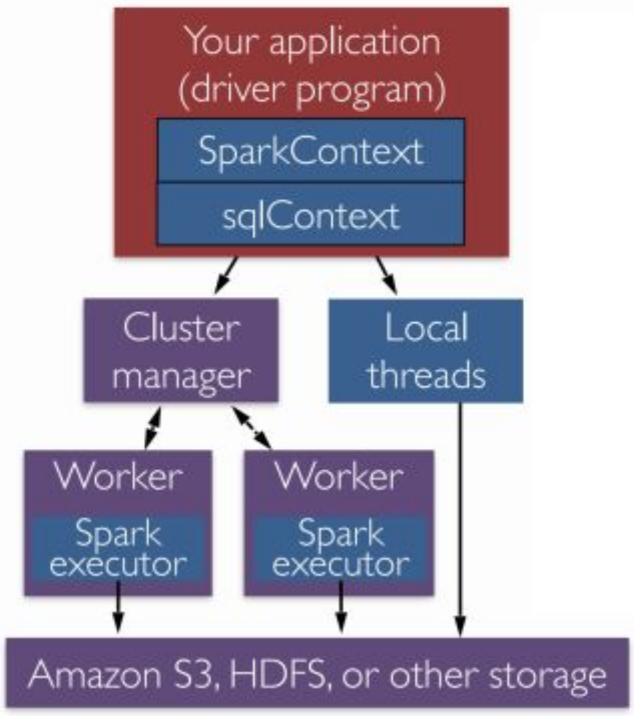
- **Cluster Manager:** Allocates resources across cluster, manages scheduling. e.g., YARN/Mesos

- **Worker program:** The program that runs on the worker nodes.

- **Executor:** What actually does the work/tasks on each cluster node.

Execution of a Spark Program

- The driver program runs the Spark application, which creates a SparkContext upon start-up.
- The SparkContext connects to a cluster manager (e.g.,Mesos/YARN) which allocates resources.
- Spark acquires executors on nodes in the cluster, which are processes that run computations and store data for your application.
- Next, driver program sends your application code to these executors.
- Finally, SparkContext sends tasks for the executors to run.



Carnegie Mellon University

Spark APIs

1. **RDD (Resilient Distributed Datasets):** Think distributed list.
2. **DataFrames:** SQL-like structured datasets with query operations.
3. **Datasets:** A mixture of RDDs and DataFrames in terms of the operations that are available on them.

- The homeworks in this class will focus on using the RDD and the Dataframe APIs.
- The dataframe API has lots of parallels from Pandas Dataframes.

RDDs vs Dataframes

RDDs

- RDDs are **immutable distributed collection** of elements of your data that can be stored in memory or disk across a cluster of machines.
- The data is partitioned across machines in your cluster that can be operated in parallel with a low-level API that offers **transformations** and **actions**.
- RDDs are **fault tolerant** as they track data lineage information to rebuild lost data automatically on failure.

DataFrames

- Like an RDD, a DataFrame is an immutable distributed collection of data.
- Unlike an RDD, data is organized into named columns, like a table in a relational database.
- Dataframes allow for a higher-level abstraction by imposing a structure onto a distributed collection of data.

Carnegie Mellon University

Transformations vs Actions

- **Transformations:** Return new RDDs as a result.
  - They are lazy, their result RDD is not immediately computed.

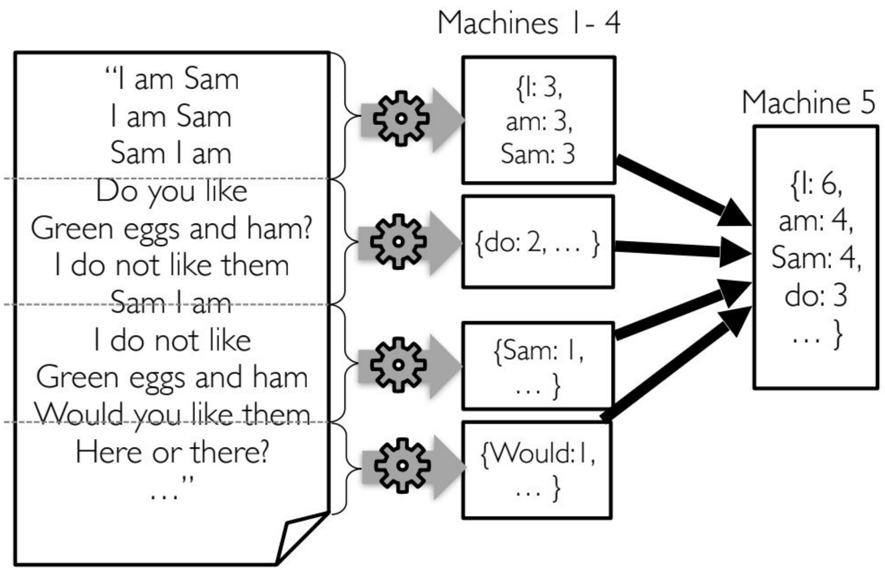
Transformation	Description
`map(func)`	return a new distributed dataset formed by passing each element of the source through a function *func*
`filter(func)`	return a new dataset formed by selecting those elements of the source on which *func* returns true
`distinct([numTasks]))`	return a new dataset 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)

- **Actions:** Compute a result based on an RDD, and either returned or saved to an external storage system (e.g., HDFS)
  - They are eager, their result is immediately computed.

Action	Description
`reduce(func)`	aggregate dataset's elements using function *func*. *func* takes two arguments and returns one, and is commutative and associative so that it can be computed correctly in parallel
`take(n)`	return an array with the first *n* elements
`collect()`	return all the elements as an array WARNING: make sure will fit in driver program
`takeOrdered(n, key=func)`	return n elements ordered in ascending order or as specified by the optional key function

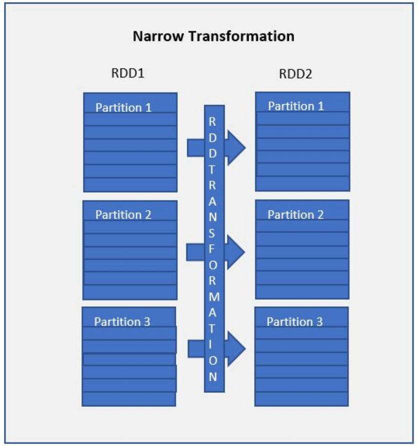
Carnegie Mellon University

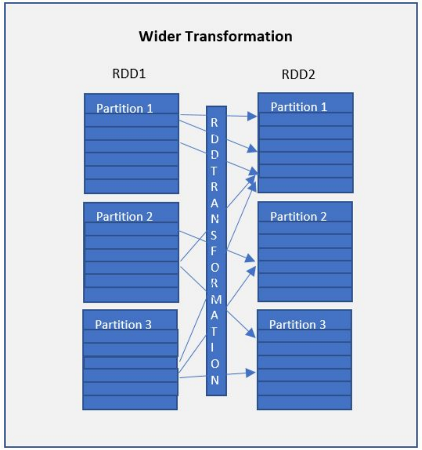
Example: Map vs. Reduce



Carnegie Mellon University

Narrow vs Wide Transformations

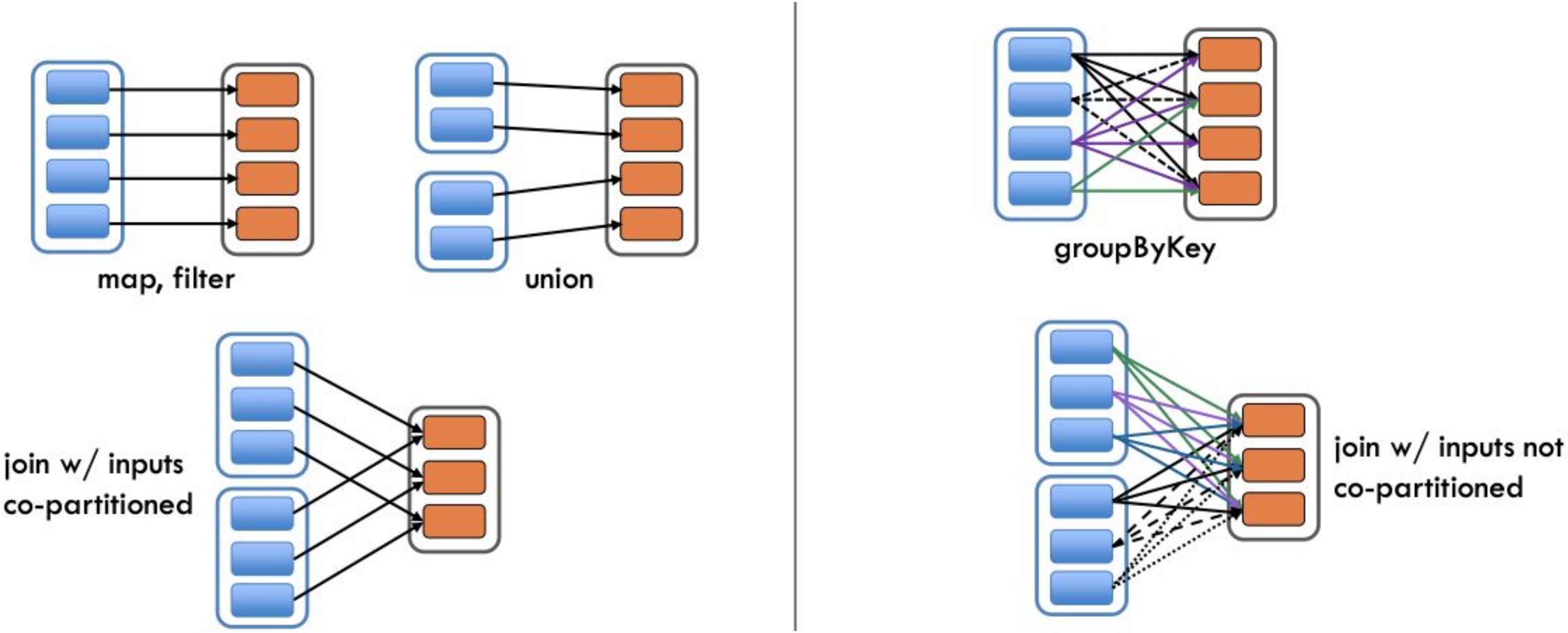




- Each partition of the parentRDD is used by at most one partition of the childRDD.
- Fast, No Data Shuffling

- Multiple childRDD partitions may depend on a single parent RDD partition.
- Slow, Data Shuffling Required

Narrow vs Wide Transformations



How can we create an RDD?

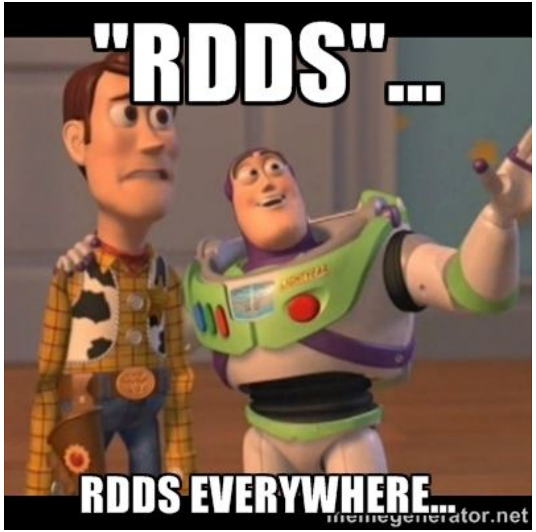
- From a SparkContext (or SparkSession)

```
# parallelizing data collection
my_list = [1, 2, 3, 4, 5]
my_list_rdd = sc.parallelize(my_list)
```

- Referencing to the external data file stored

```
## 2. Referencing to external data file
file_rdd = sc.textFile("path_of_file")
```

- Transforming an existing RDD

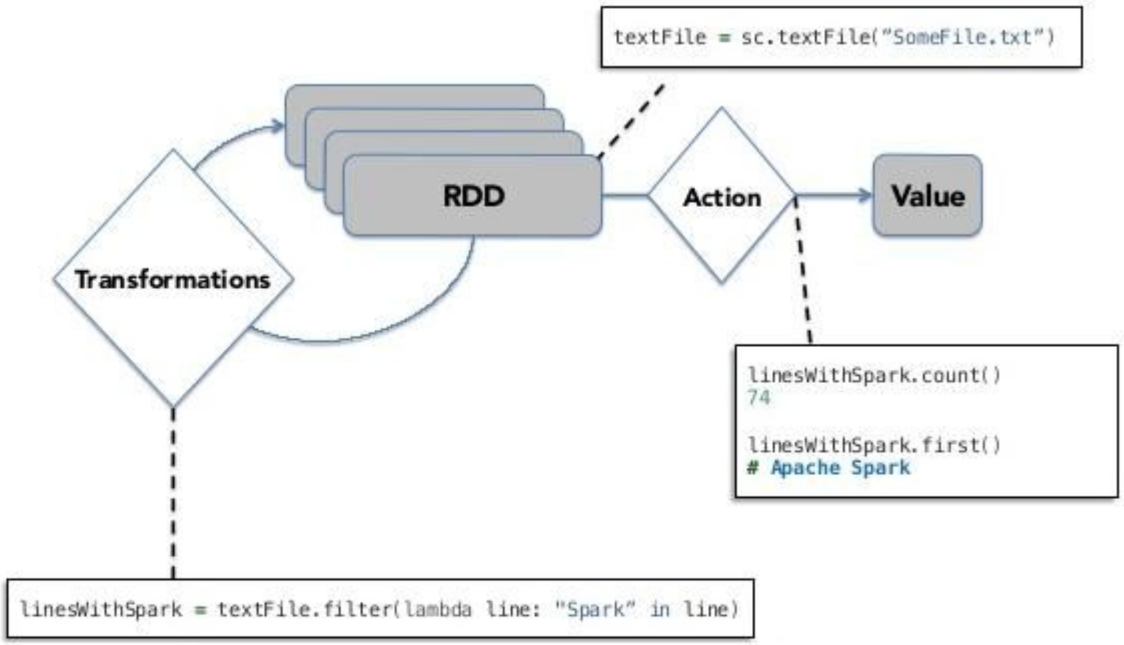


What can you do when you have an RDD?

A small cheat-sheet can be found here with some nice examples.

Carnegie Mellon University

Interacting with RDDs



Carnegie Mellon University

Lambda vs Regular Functions

Lambda Functions

- Evaluates only a single expression
- No name associated

```
lambda x : x + x
```

Regular Function

- Can have multiple expressions
- Must have name associated

```
def adder (x, y):
return x + y
```

Example

What does the following code snippet do?

```
peopleRdd = sc.parallelize ([“bob”, “alice”, “bill”])

words_filter = peopleRdd.filter(lambda x: ‘i’ in x)

filtered = words_filter.collect()
```

Example

What does the following code snippet do?

```
peopleRdd = sc.parallelize ([“bob”, “alice”, “bill”])
```

creates the RDD.

```
words_filter = peopleRdd.filter(lambda x: ‘i’ in x)
```

nothing, since this is a transformation.

```
filtered = words_filter.collect()
```

the DAG is executed since an action is called.

Example #2

What does the following code snippet do on the driver node?

```
peopleRdd = sc.parallelize ([“bob”, “alice”, “bill”])

peopleRdd.foreach(print)

peopleRdd.take(2)
```

Example #2

What does the following code snippet do on the driver node?

```
peopleRdd = sc.parallelize ([“bob”, “alice”, “bill”])

peopleRdd.foreach(print)

peopleRdd.take(2)
```

On the driver: Nothing. Why?
foreach is an action, which doesn't return anything. Therefore, it is eagerly executed on the executors, not the driver. Therefore, any calls to print are happening on the stdout of the worker nodes and are thus not visible in the stdout of the driver node.

What about when take is called? Where will the array of people end up?
When an action returns a result, it returns it to the driver node.

collect	collect(): Array[T] Return all elements from RDD.
count	count(): Long Return the number of elements in the RDD.
take	take(num: Int): Array[T] Return the first num elements of the RDD.
reduce	reduce(op: (A, A) -> A): A Combine the elements in the RDD together using op function and return result.
foreach	foreach(f: T -> Unit): Unit Apply function to each element in the RDD.

Carnegie Mellon University

Example #3

What does the following code snippet do on the driver node?

```
peopleRdd = sc.parallelize ([“bob”, “alice”, “bill”])

newRDD = peopleRDD.foreach(print)
print(type(newRDD))

newRDD = peopleRDD.map(print)
print(type(newRDD))
newRDD.take(2)

newRDD = peopleRDD.map(lambda x: "Dr." + x)
newRDD.take(2)
```

Carnegie Mellon University

Example #3

What does the following code snippet do on the driver node?

```
peopleRdd = sc.parallelize ([“bob”, “alice”, “bill”])

newRDD = peopleRDD.foreach(print)
print(type(newRDD))

newRDD = peopleRDD.map(print)
print(type(newRDD))
newRDD.take(2)

newRDD = peopleRDD.map(lambda x: "Dr." + x)
newRDD.take(2)
```

<class ‘NoneType’>
foreach is an action, which doesn't return anything. Therefor
the type of the returned is NoneType.

Carnegie Mellon University

Example #3

What does the following code snippet do on the driver node?

```
peopleRdd = sc.parallelize ([“bob”, “alice”, “bill”])
```

```
newRDD = peopleRDD.foreach(print)
print(type(newRDD))
```

<class ‘NoneType’>
foreach is an action, which doesn't return anything. Therefor
the type of the returned is NoneType.

```
newRDD = peopleRDD.map(print)
print(type(newRDD))
newRDD.take(2)
```

<class ‘pyspark.rdd.PipelinedRDD’>
[None, None]
map is a transformation, and the type of the returned is another RDD where the values are modified. However, since the print() function doesn’t return anything, the output will be [None, None].

```
newRDD = peopleRDD.map(lambda x: "Dr." + x)
newRDD.take(2)
```

Carnegie Mellon University

Example #3

What does the following code snippet do on the driver node?

```
peopleRdd = sc.parallelize ([“bob”, “alice”, “bill”])
```

```
newRDD = peopleRDD.foreach(print)
print(type(newRDD))
```

<class ‘NoneType’>
foreach is an action, which doesn't return anything. Therefor
the type of the returned is NoneType.

```
newRDD = peopleRDD.map(print)
print(type(newRDD))
newRDD.take(2)
```

<class ‘pyspark.rdd.PipelinedRDD’>
[None, None]
map is a transformation, and the type of the returned is another RDD where the values are modified. However, since the print() function doesn’t return anything, the output will be [None, None].

```
newRDD = peopleRDD.map(lambda x: "Dr." + x)
newRDD.take(2)
```

[‘Dr. bob’, ‘Dr. alice’]
map is a transformation, so it returned a new RDD with “Dr.” added before each name.

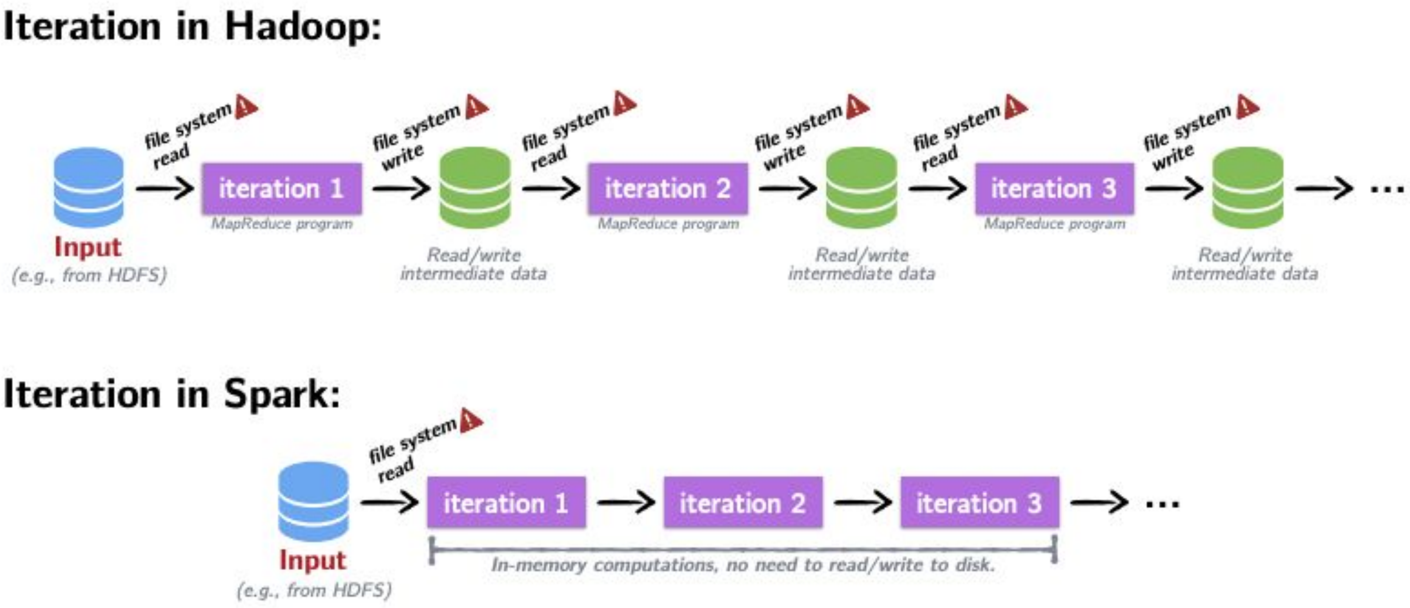
Carnegie Mellon University

Spark Execution

- Spark uses lazy evaluation!
  - Lazy evaluation means nothing executes. Spark saves recipe for transforming source.
  - Results are not computed right away – instead, Spark remembers set of transformations applied to base data set.
  - The way Spark “remembers” is by creating a DAG and then it tries and optimises the DAG to optimize the use of resources (like bandwidth, memory etc).

- It enables Spark to optimize the required operations.
- It enables Spark to recover from failures and slow workers.
- Spark supports in-memory computation which enables it to be a lot ***faster for iterations.***

Spark Execution



Cache/Persistence

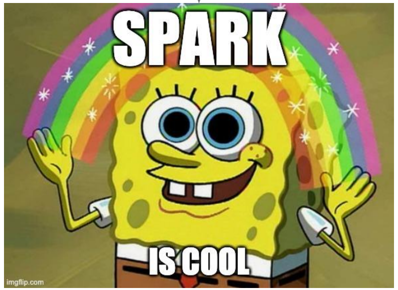
- By default, RDDs are recomputed each time you run an action on them. This can be expensive (in time) if you need to use a dataset more than once.
- Spark allows you to control what can be cached into memory (using the .cache() or the .persist() API).
- One of the most common performance bottlenecks arises from unknowingly re-evaluation several transformations.
  This will be tested in some of the assignments.

Reference Link - https://data-flair.training/blogs/apache-spark-rdd-persistence-caching/

Carnegie Mellon University

FAQs/Helpful Debugging Instructions

- In case tests don't pass, try running all the cells in the notebook.
- If you see a *name-not-defined error,* It could be because of your code timing out on the grader.
- Timeouts can occur due to inefficient code (for example, calling `collect()` on the entire dataset).
- Local tests are *not* comprehensive. They are there to assist you in building your code. You only get points if the assignment runs on the autograder.



Carnegie Mellon University

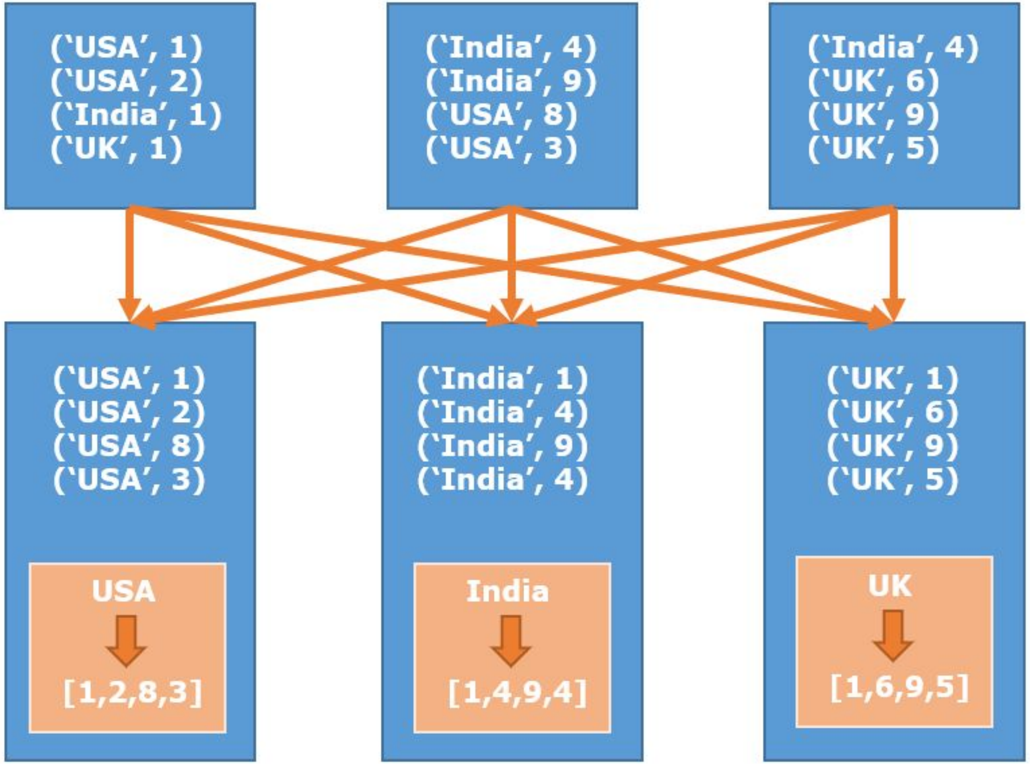
Appendix

Carnegie Mellon University

RDDs vs Dataframes vs Datasets

	RDDs	Dataframes	Datasets
Data Representation	RDD is a distributed collection of data elements without any schema.	It is also the distributed collection organized into the named columns	It is an extension of Dataframes with more features like type-safety and object-oriented interface.
Optimization	No in-built optimization engine for RDDs. Developers need to write the optimized code themselves.	It uses a catalyst optimizer for optimization.	It also uses a catalyst optimizer for optimization purposes.
Projection of Schema	Here, we need to define the schema manually.	It will automatically find out the schema of the dataset.	It will also automatically find out the schema of the dataset by using the SQL Engine.
Aggregation Operation	RDD is slower than both Dataframes and Datasets to perform simple operations like grouping the data.	It provides an easy API to perform aggregation operations. It performs aggregation faster than both RDDs and Datasets.	Dataset is faster than RDDs but a bit slower than Dataframes.

Group By Key Example - PySpark



Carnegie Mellon University