ISA 414 – Managing Big Data

Lecture 23 – Introduction to Spark

(*Part I*)

Dr. Arthur Carvalho

arthur.carvalho@miamioh.edu

MIAMI UNIVERSITY

Announcements

- Assignment 4 is now on Canvas
  - Deadline: Sunday Nov. 14^{th} before 11:59 pm

- Final-project groups
  - You will present your preliminary ideas on Tuesday and Thursday next week
  - It is now time to form the groups
    - **Send me your preferences by the end of Wednesday, Nov 10^{th}**

Lecture Objectives

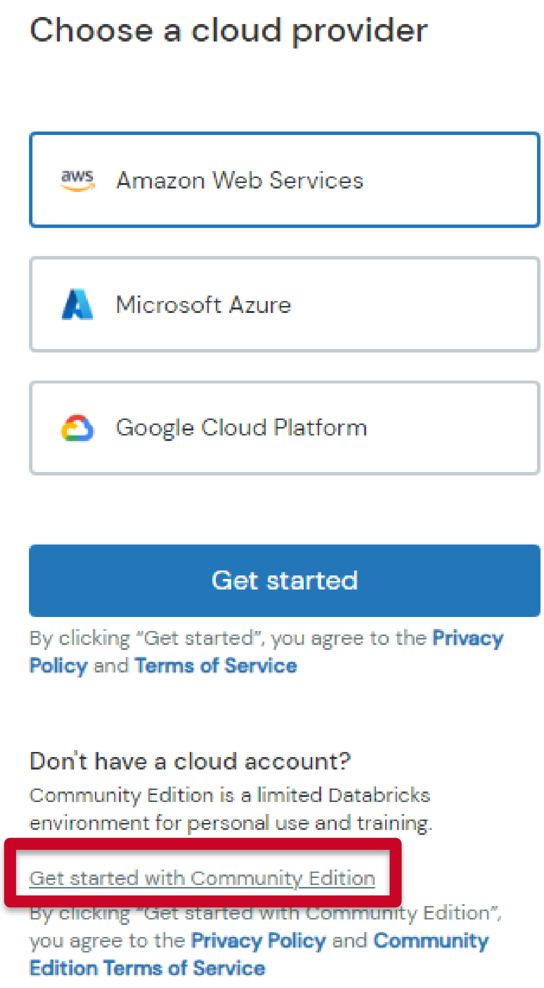
- Review of Homework 10
- Understand the basic components of traditional computer architectures
  - Difference between main memory and secondary storage
- Learn about Spark
  - RDD, Transformations, Actions, Libraries
- Prepare the Databricks environment

Lecture Instructions

- Download the files *mobydick.txt* and *Lecture 23.ipynb* from Canvas

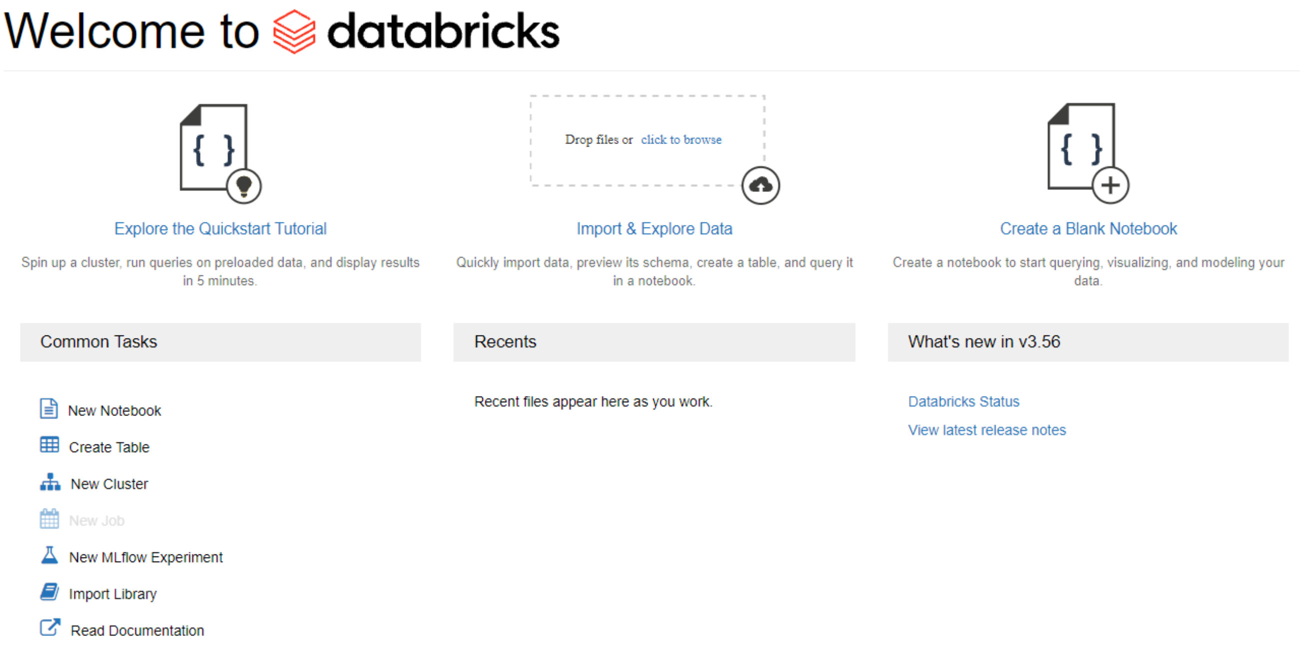
Lecture Instructions

- Create an account on Databricks
  - Try at home if this fails now
    - Databricks may block us based on IP address
- Go to https://databricks.com/try-databricks/
  - Fill is the forms
  - Select “Get Start with Community Edition”
  - Check your email



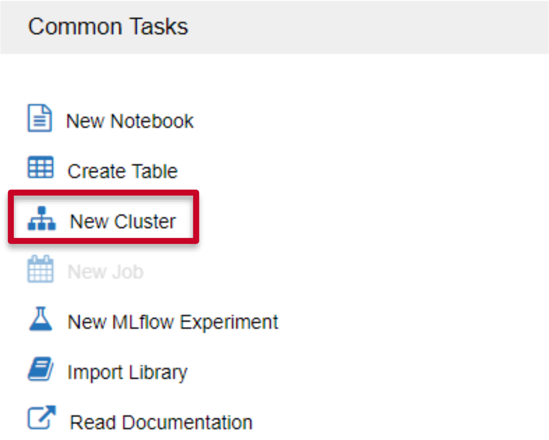
Lecture Instructions

- You should get to this screen



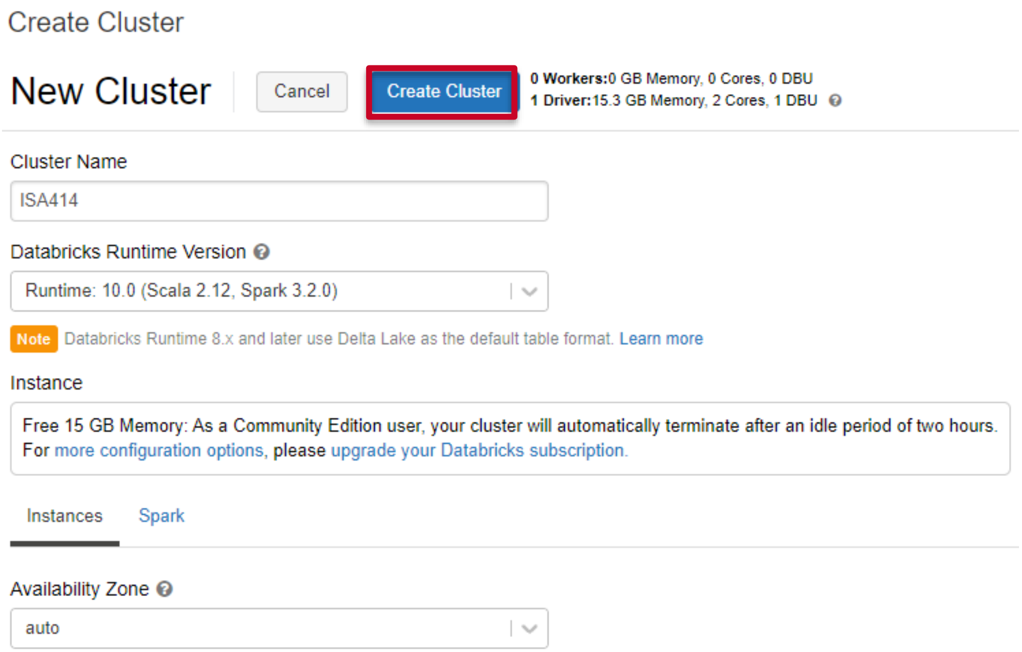
Lecture Instructions

- Let’s start by creating a cluster
  - Go to “Common Tasks” -> “New Cluster”



Lecture Instructions

- Configure you cluster and click on “Create Cluster”



MIAMI UNIVERSITY

Computer Architecture

- Computers have 3 major components
  - CPU (Central Unit Processing)
    - Might have one or more “cores”
      - Each core processes standard instructions (such arithmetic operations) independently
      - Allows for parallel computing inside a single computer

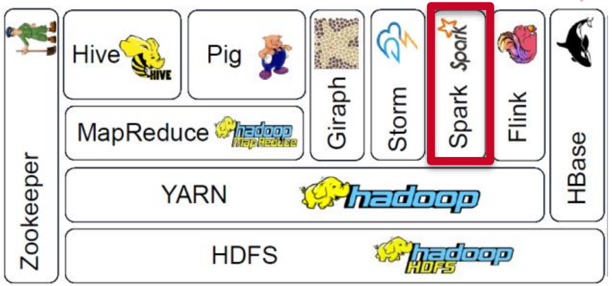
Computer Architecture

- Computers have 3 major components
  - Main (primary) memory
    - Operates at very high speed
    - Low capacity (storage space)
    - Expensive
    - Electricity based (volatile)
      - All data is lost after a computer is turned off
    - Technologies: RAM, DRAM, SRAM, …

Computer Architecture

- Computers have 3 major components
  - Auxiliary storage (secondary memory)
    - Sometimes referred to as the “disk”
    - Slow to access information
    - High capacity (storage space)
    - Cheap
    - Non-volatile
      - Retain stored data even when a computer is powered off
    - Technologies
      - Hard disk (mechanical, magnetic storage)
      - Solid-state disk (SSD – no mechanical components)
      - …

The Hadoop Ecosystem



- Overview
  - HDFS
    - Hadoop Distributed File System
    - Scalable and reliable storage
  - Yarn
    - Schedule jobs/task over HDFS storage
  - **Spark**
    - **Built for real-time, <u>in-memory</u> processing of data**

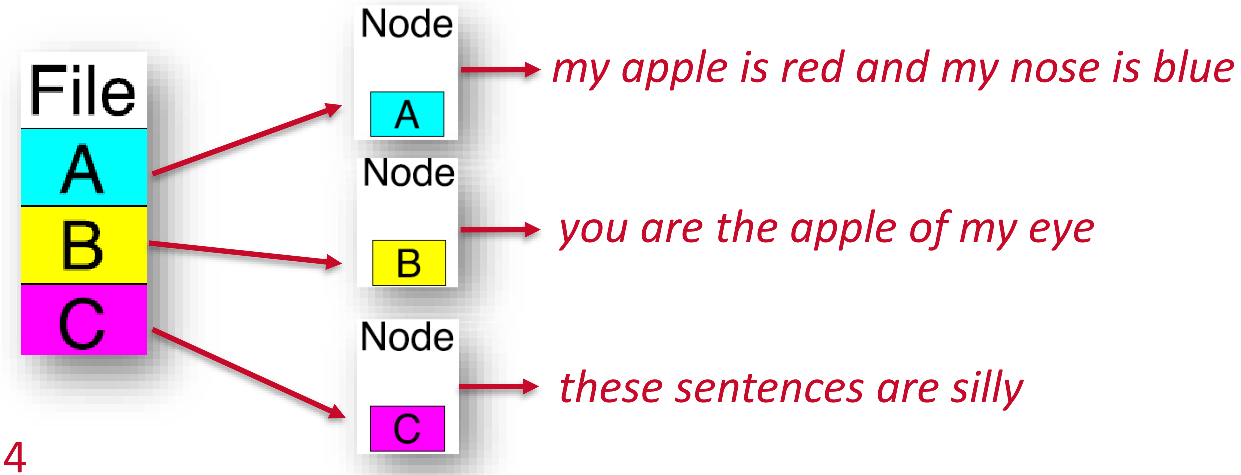
MIAMI UNIVERSITY

Spark

- The current "big thing" in predictive analytics
  - Originally developed by a PhD student at UC Berkeley in 2009
    - Currently managed by the Apache foundation
      - Initial release: 2014
  - Allows for distributed computation
    - More flexible than MapReduce
  - Easy to use
    - Many predefined distributed operations
      - Joins, filters, merge, …
    - Many predefined machine learning algorithms
      - Decision trees, random forests, linear regression,…

Quick Intro to MapReduce

- Suppose a very large textual data set is stored in a commodity cluster
  - We will define a MapReduce program to calculate the frequency of words in the data



MIAMI UNIVERSITY

Quick Intro to MapReduce

- Map operation
  - Executed in the node where the data block is stored
    - Moving computation to data
  - Example
    - Key = word
    - Value = 1

Node A → *my apple is red and my nose is blue*

Node B → *you are the apple of my eye*

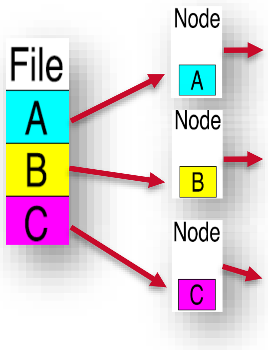
Node C → *these sentences are silly*

Outputs

Map (A)		Map (B)		Map (C)	
Key	**Value**	**Key**	**Value**	**Key**	**Value**
my	1	you	1	these	1
apple	1	are	1	sentences	1
is	1	the	1	are	1
red	1	apple	1	silly	1
and	1	of	1		
my	1	my	1		
nose	1	eye	1		
is	1				
blue	1				

Quick Intro to MapReduce

- <u>Sort and shuffle operation</u>
  - Nodes exchange data among themselves
    - Key-value pairs with the same key stay in the same node



my apple is red and my nose is blue

you are the apple of my eye

these sentences are silly

Node A		Node B		Node C	
Key	**Value**	**Key**	**Value**	**Key**	**Value**
my	(1,1,1)	you	1	these	1
apple	(1,1)	red	1	sentences	1
is	(1,1)	are	(1,1)	silly	1
		of	1	and	1
		the	1	nose	1
		eye	1	blue	1

Quick Intro to MapReduce

- **Reduce operation**
  - Values with similar keys are aggregated
    - Aggregation technique must be defined by the code
  - Outputs are saved back to HDFS (keys become unique)
    - A client can later request the aggregate results from HDFS
  - Example:
    - Reduce = sum

HDFS

Node A		Node B		Node C	
Key	Value	Key	Value	Key	Value
my	3	you	1	these	1
apple	2	red	1	sentences	1
is	2	are	2	silly	1
		of	1	and	1
		the	1	nose	1
		eye	1	blue	1

client

request

Key	Value
my	3
apple	2
is	2
you	1
red	1
are	2
of	1
the	1
eye	1
these	1
sentences	1
silly	1
and	1
nose	1
blue	1

Quick Intro to MapReduce

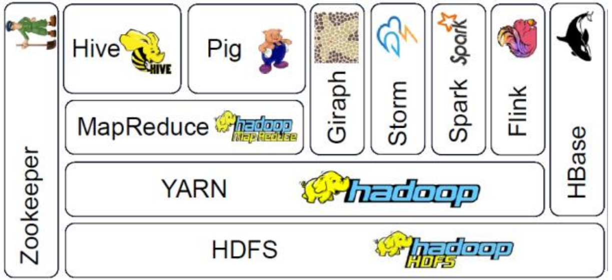
- Parallelization during MapReduce
  - Map: function applied to individual blocks of data in different nodes
  - Shuffle and sort: parallelization during sorting
  - Reduce: parallelization to aggregate individual results
- MapReduce is language independent
  - In theory, it can be implemented using virtually any programming language

Back to Spark

- Key benefits
  - No need to explicitly define map and reduce tasks like in MapReduce
  - In-memory caching of the data
    - Data are loaded into the nodes' main memory and often stay there until a task is done
      - Oftentimes, complex tasks are executed 10x to 100x faster than in the MapReduce framework
  - Spark has a native programming language: Scala
    - Many programming language interfaces: Python, Java, R

Spark

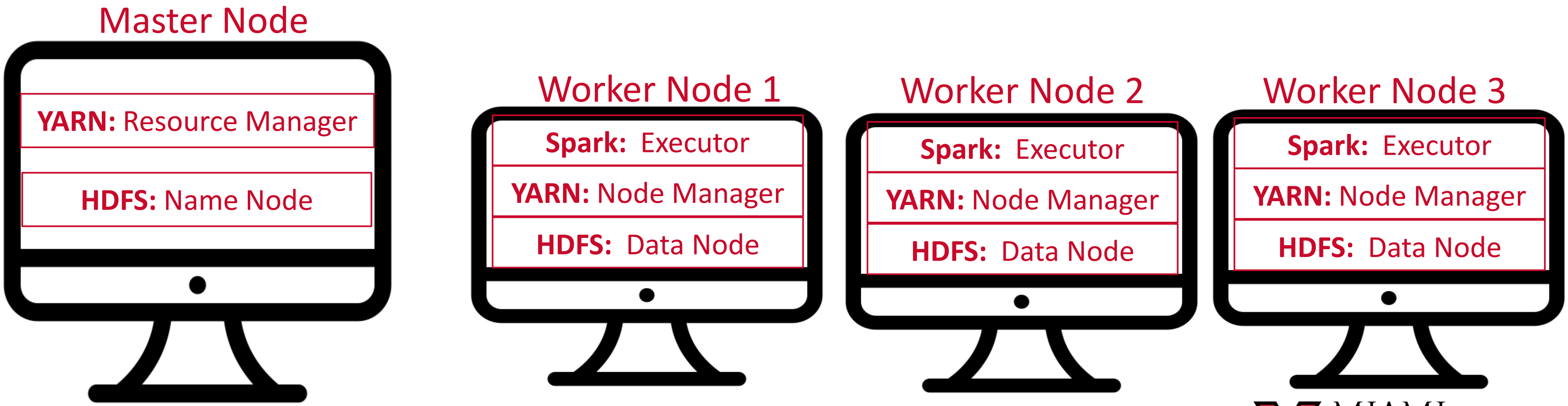
- Spark requires a job manager and a distributed storage system
  - Hadoop setup: YARN + HDFS



  - Alternatives:
    - Job manager: native Spark cluster, Apache Mesos
    - Distributed storage: MapR, Cassandra, Amazon S3, Kudu

Spark

- Spark will run as another service inside nodes
  - The nodes running Spark form a “Spark cluster”



Spark

- Key definitions
  - Consider you have a data set distributed across a cluster of machines
  - The data set is represented by a structure called *Resilient Distributed Dataset* (RDD) once loaded into the Spark cluster
  - RDD's main characteristics:
    - *Distributed*: blocks of data are distributed across nodes in a cluster
    - *In-memory*: data inside RDDs are loaded and possibly kept into the main memory of the data nodes for rapid reuse
    - *Immutable*: RDDs cannot be changed
  - RDD supports two operations: <u>actions</u> and <u>transformations</u>

Spark

- Data pipeline

Data $\xrightarrow{\text{Transformation}}$ RDD $\xrightarrow{\text{Action}}$ Output

- Transformations: operations that return another RDD
  - *E.g.*, data manipulation (joins, filter, map, groupBy)
- Actions: operations that trigger a computation and return values
  - *E.g.*, count, max, min, reduce

Spark

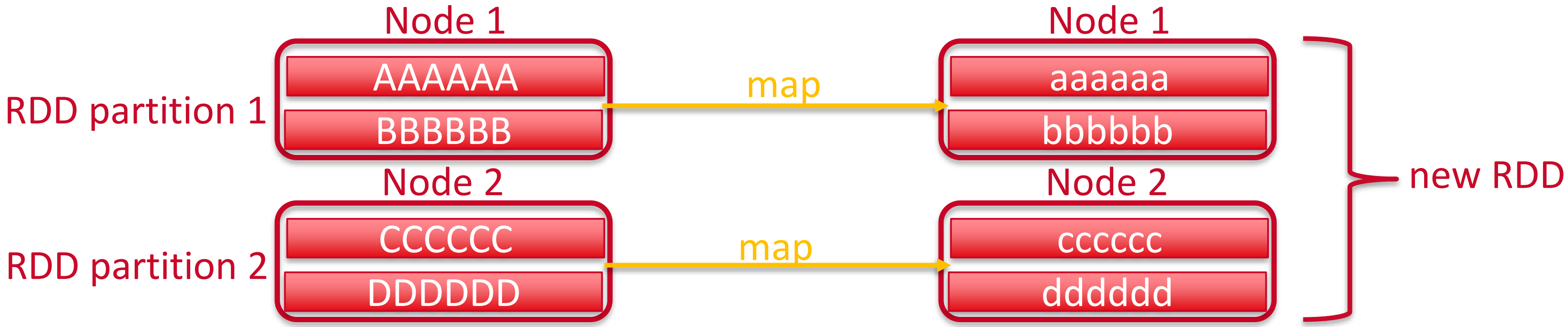
- Transformations
  - Functions that take an RDD as the input and produce one or many RDDs as the output

Spark

- Transformations
  - Many transformations can be chained together
    - Each one produces an intermediate RDD

Spark

- Transformations
  - Example #1: **map** transformation = apply a function to each partition of an RDD
    - Suppose one has a massive textual file in HDFS and that file is loaded into a Spark cluster (the file becomes an RDD)
    - The analyst wants to transform all characters to lower case using the map transformation



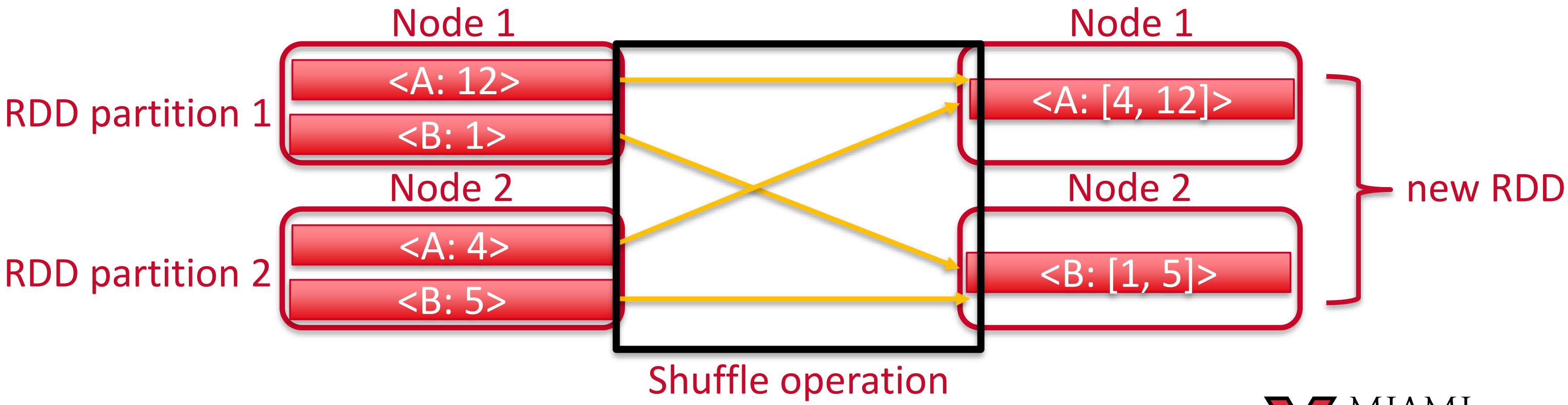
MIAMI UNIVERSITY

Spark

- Transformations
  - Note that the **map** function is completely local
    - Each node's computations are independent of other nodes' computations
      - Computations are processed locally
    - These are called ***narrow transformations***
      - No transferring of data through the network
    - Transformations can also be ***wide***
      - Involve the transferring of data through the network
      - Consume more resources

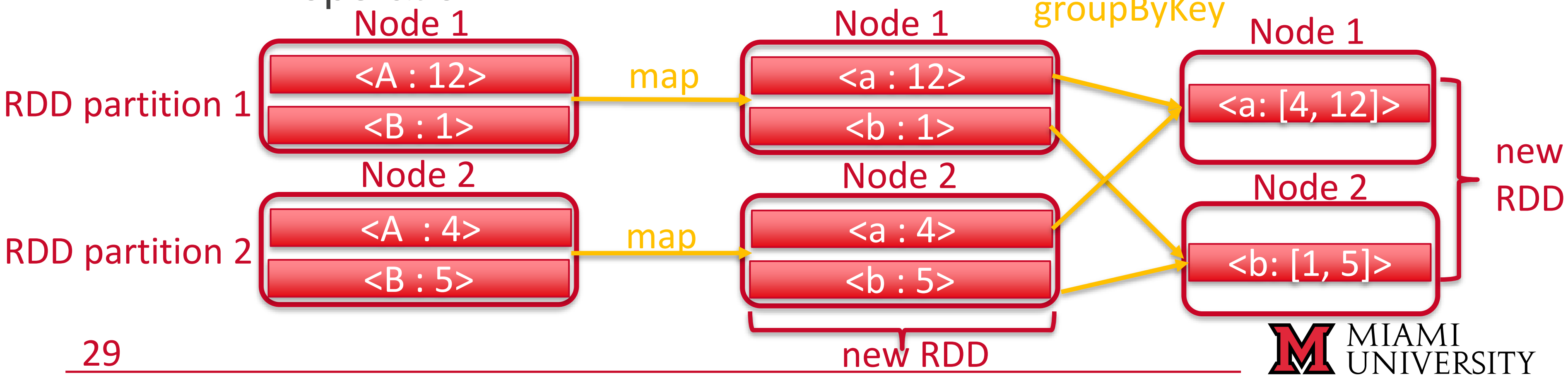
Spark

- Transformations
  - Example #2: **groupByKey** transformation
    - Input: an RDD of key-value pairs
    - Output: transfers all the values that have the same key to the same partition
    - Example:



Spark

- Transformations
  - Clearly, many transformations can be chained together
    - Example #3: a map (to lowercase) followed by a groupByKey operation



Spark

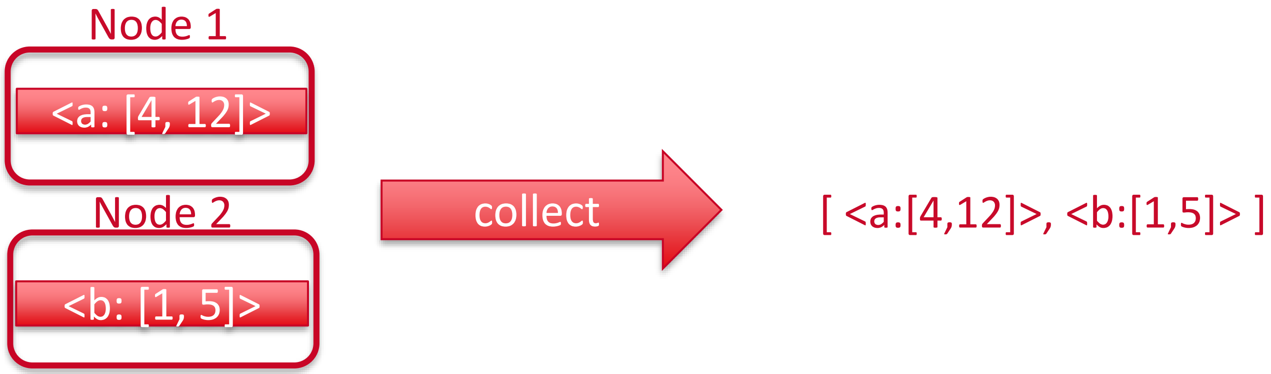
- Transformations
  - There are many different transformations in Spark
    - The following list is not exhaustive
      - *map*, *filter*, *flatMap*, *mapPartitions*, *mapPartitionsWithIndex*, *sample*, *union*, *intersection*, *distinct*, *groupByKey*, *reduceByKey*, *aggregateByKey*, *sortByKey*, *join*, *cogroup*, *cartesian*, *pipe*, *coalesce*, *repartition*, *repartitionAndSortWithinPartitions*
  - One great thing about Spark transformations is that their inputs/outputs are not necessarily key-value pairs

Spark

- Transformations are *lazy*
  - They are not performed right away
  - When an action is called, Spark looks at the whole chain of transformations and creates and optimal execution plan
- Actions
  - Last step of the data workflow
    - Sparks creates an execution plan and sends the tasks to the nodes
  - Produces a result/outcome

Spark

- Actions
  - Example: **collect**
    - Returns the content of each RDD partition to an app/user



Spark

- Actions
  - There are many different actions in Spark
    - The following list is not exhaustive
      - *reduce, collect, count, first, take, takeSample, takeOrdered, saveAsTextFile, saveAsSequenceFile, saveAsObjectFile, countByKey, foreach*

Spark

- One can perform distributed computations in Spark by calling transformations and actions
  - RDD programming model
  - Example in Scala: counting the number of occurrences of the word “spark” in a file

```
val data = spark.read.textFile("spark_test.txt").rdd
val mapFile = data.flatMap(lines => lines.split(" ")).filter(value => value=="spark")
println(mapFile.count())
```

- In practice, it is more likely that one will use one of the four *Spark libraries* built on top of the previously discussed concepts
  - They are incredibly easy to use
  - SQL, ML, Streaming, GraphX

Spark

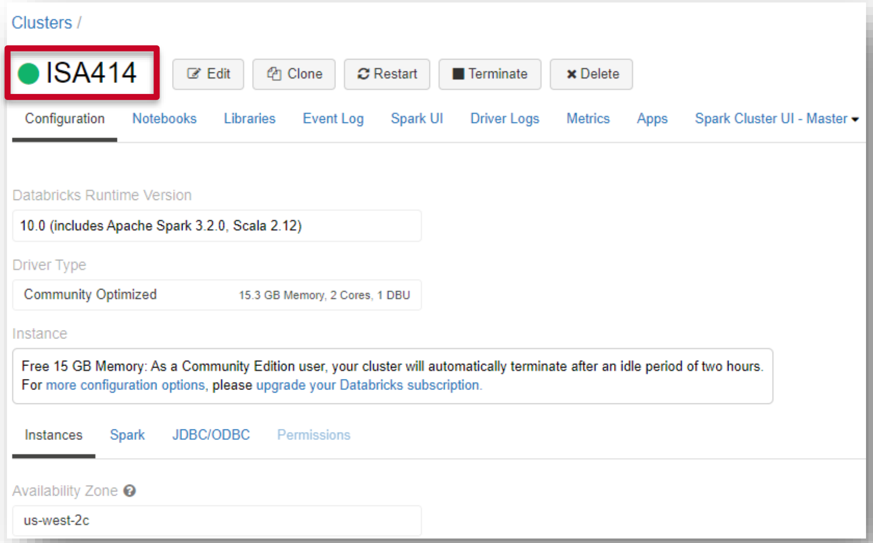
- Spark Libraries
  - Spark SQL: implements relational queries on Spark
  - Spark Streaming: implements incremental stream processing using a model called “discretized streams”
    - Transformations and actions are applied to small batches of data, such as every 200 milliseconds
  - Spark GraphX: provides a graph computation interface ideal for social network analysis
  - Spark MLlib: implements more than 50 common machine learning algorithms for distributed model training
    - Summary statistics, correlations, hypothesis testing, classification, regression, cluster analysis, dimensionality reduction, …

Databricks

- Let’s go back to Databricks
  - “*An open and unified data analytics platform*”
    - PaaS
    - Collaborative environment where analytics teams can work together
    - From the creators of Spark
    - Heavily used in industry
      - We face many limitations because we are using the free version
  - One can easily create:
    - Clusters of machines running on top of Azure, AWS, Google Cloud
    - Notebooks and run popular ML frameworks (SK-learn, TensorFlow, Keras, Spark, …)

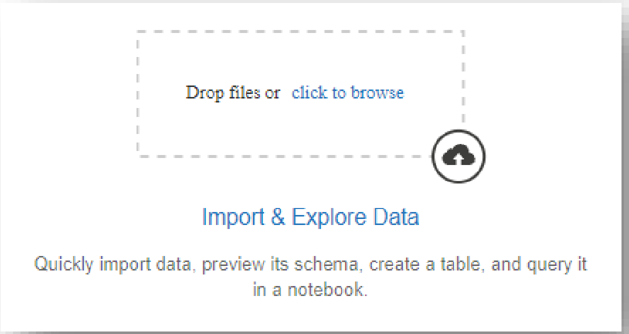
Databricks

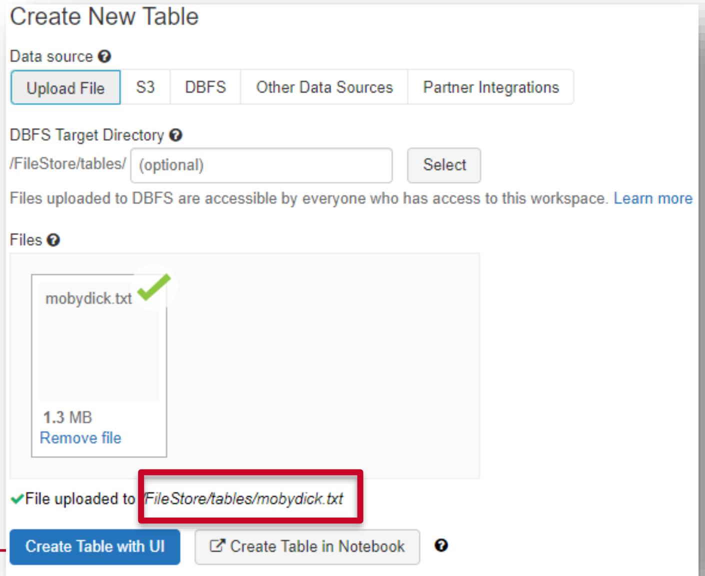
- Your cluster should be ready by now
  - Clusters in the free version are deleted after a couple hours of inactivity



Databricks

- Let’s upload a local file to our cluster
  - Click on the Databricks logo on top-left
  - Drag and drop the file *mobydick.txt* to upload the file to the cluster
    - Copy the path to the file



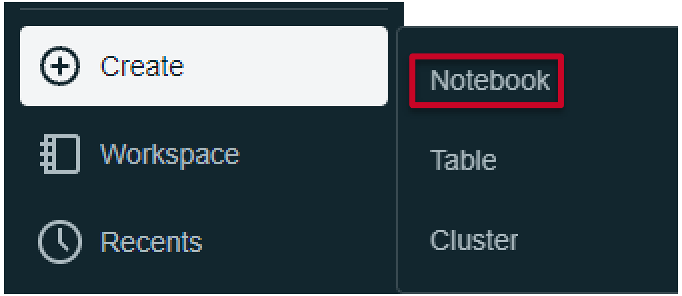


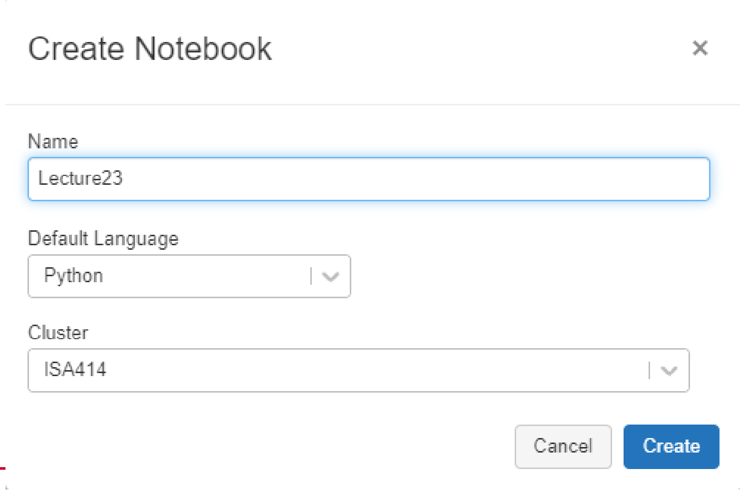
Databricks

- So, are we using Hadoop HDFS here?
  - No! we are using a similar technology called DBFS "*Databricks File System*"
    - Data lake
  - Such a technology can retrieve data from any major cloud provider
    - Meaning that the data may be stored by Amazon, Microsoft, Google, *etc*.

Databricks

- Let's create a Jupyter notebook running on our cluster
  - Click on the side menu-item "*Create*" -> "*Notebook*"
  - Fill in the notebook form





Databricks

- Voila! We are a Jupyter notebook on top of a cluster
  - Spark is already configured for us

- Let's run the last test
  - Load textual data, run transformations and actions to count the number of times each word occurs in the book Moby-Dick
  - Ignore the code
    - We learn how to use Spark via its libraries, instead of transformations and actions

Databricks

1. Testing Spark
   - Open the file *Lecture 23.ipynb* locally with VS code
2. Create four Python cells on Databricks
3. Copy and paste each Python cell from VS Code into Databricks
4. Click on “Run All” to run all cells on Databricks
   - Any errors?

Summary

- We learned about Spark
  - Transformations and Actions
- We learned about Databricks
  - Great integration of Jupyter notebooks and Spark
  - Required for Assignment 4
- Next lecture: Spark (part II)
  - Libraries: Machine Learning & SQL