Hadoop & Spark

Big Data Analytics Techniques and Applications, 2022

Outline

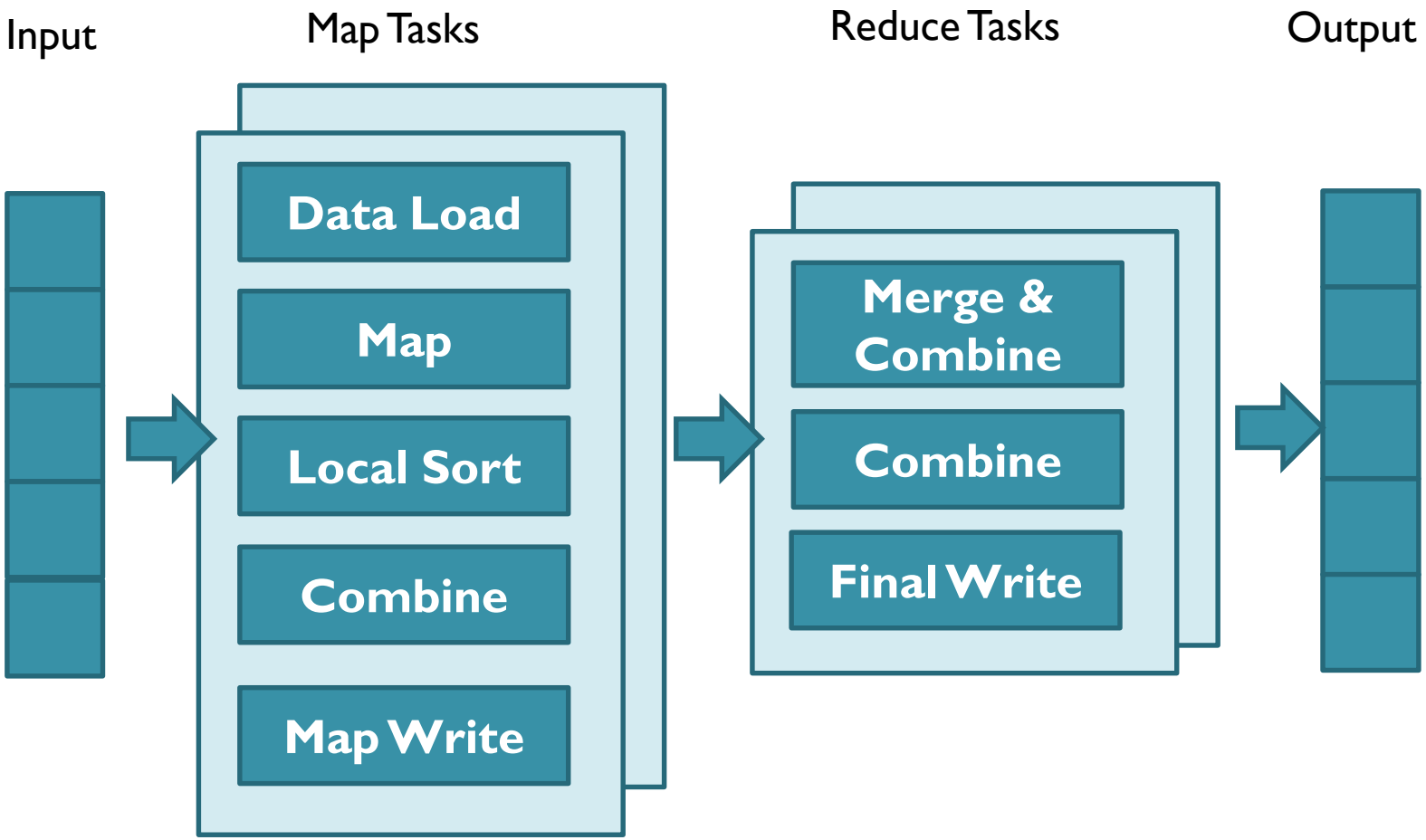
- Hadoop – MapReduce
- Spark
  - PySpark

Hadoop
-MapReduce-

Hadoop

- Hadoop is a way to distribute very large files across multiple machines.
- It uses the **Hadoop Distributed File System (HDFS)**
- HDFS allows a user to work with large data sets
- HDFS also duplicates blocks of data for fault tolerance
- It also then uses MapReduce
- **MapReduce** allows computations on that data

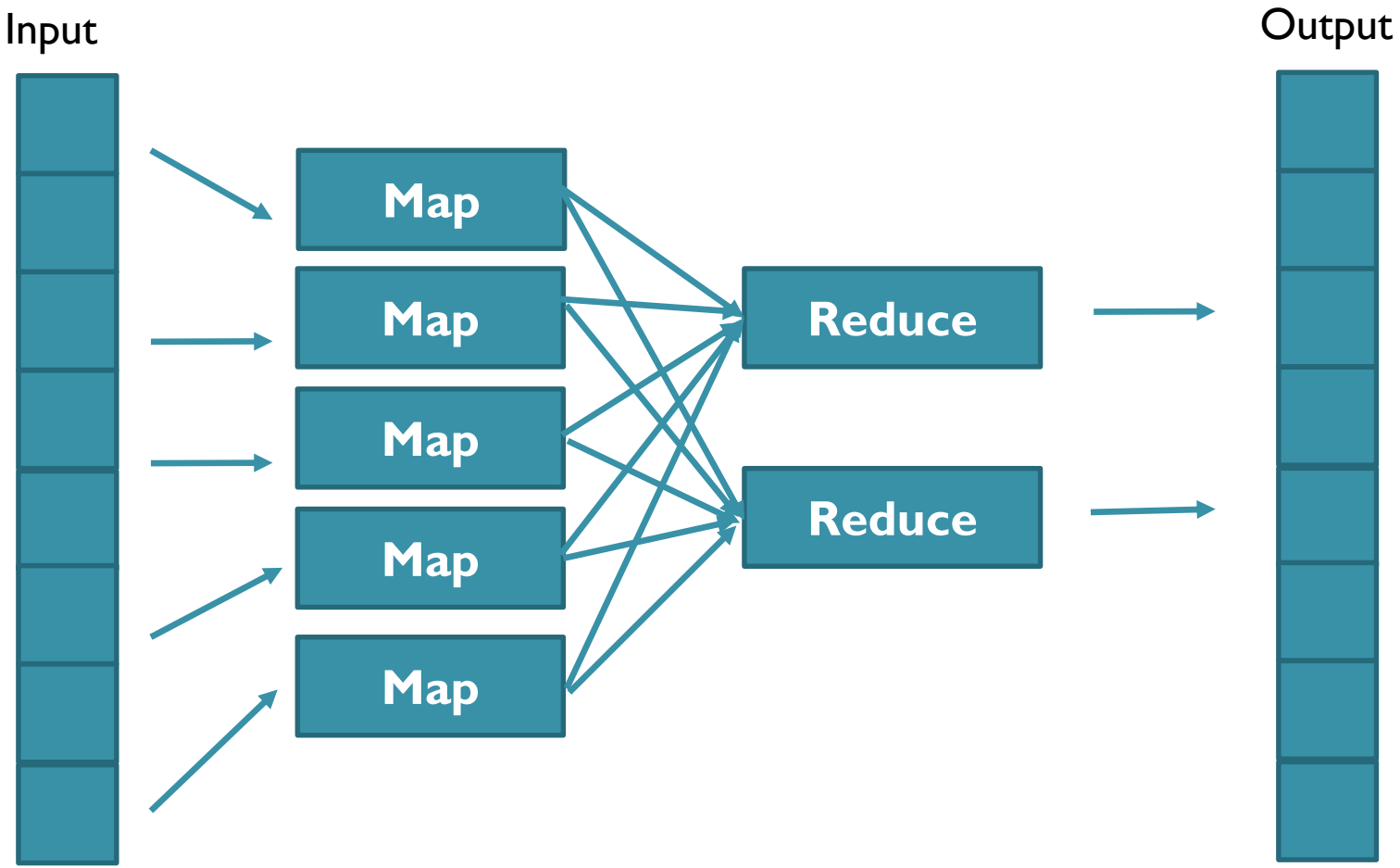
MapReduce Workflow



MapReduce Workflow (Cont.)

- **Map**
  - Data is read from a distributed file system, partitioned among a set of computing nodes in the cluster, and sent to the nodes as a set of key-value pairs.
  - The Map tasks process the input records independently of each other and produce intermediate results as key-value pairs.
  - The intermediate results are stored on the local disk of the node running the Map task.
- **Reduce**
  - The phase begins in which the intermediate data with the same key is aggregated.
  - An optional Combine task can be used to perform data aggregation on the intermediate data of the same key for the output of the mapper before transferring the output to the Reduce task.

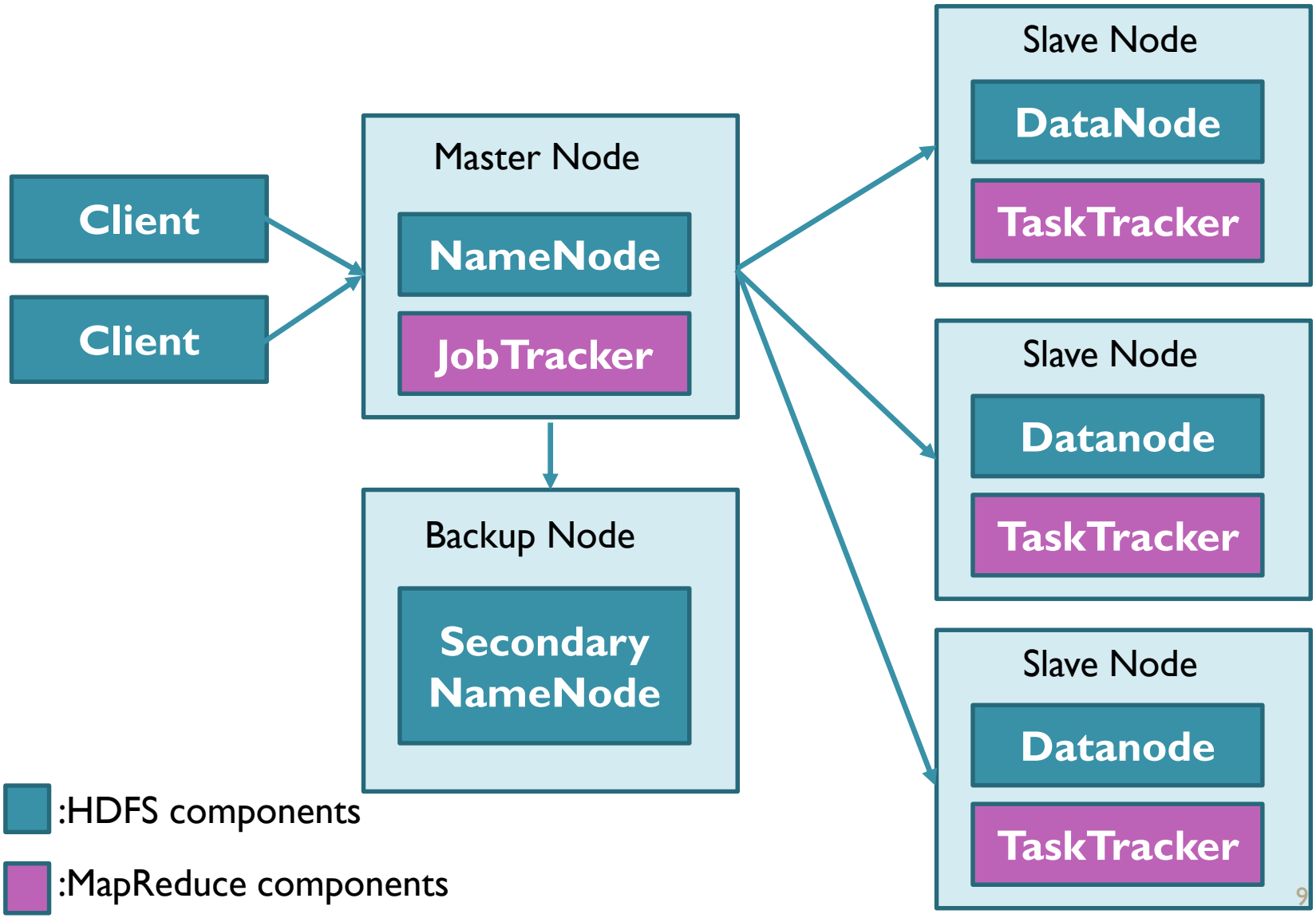
Data flow in MapReduce



Advantage of MapReduce

- In traditional data analysis, data is moved to the compute nodes
  - Results in significant of data transmission between the nodes in a cluster.
- MapReduce moves the computation to where the data resides.
  - Decreases the transmission of data .
  - Improves efficiency.

Components of a Hadoop Cluster



NameNode

- Keeps the directory tree of all files in the file system.
- Tracks where the cluster the file data is kept.
  - It does not store the data of these files itself.
- Client applications talk to the NameNode whenever they wish to locate a file, or when they want to add/copy/move/delete a file.
- The NameNode responds to the successful requests by returning a list of relevant DataNode servers where the data lives.
- NameNode serves as both directory namespace manager and "inode table" for the Hadoop DFS.
- There is a single NameNode running in any DFS deployment.

JobTracker

- Distributes MapReduce tasks to specific nodes in the cluster, ideally the nodes that have the data, or at least are in the same rack.

TaskTracker

- TaskTracker is a node in a Hadoop cluster that accepts Map,Reduce and Shuffle tasks from the JobTracker.
- Each TaskTracker has a defined number of slots which indicate the number of tasks that it can accept.
- When the JobTracker tries to find a TaskTracker to schedule a map or reduce task it first looks for an empty slot on the same node that hosts the DataNode containing the data.
- If an empty slot is not found on the same node, then the JobTracker looks for an empty slot on a node in the same rack.

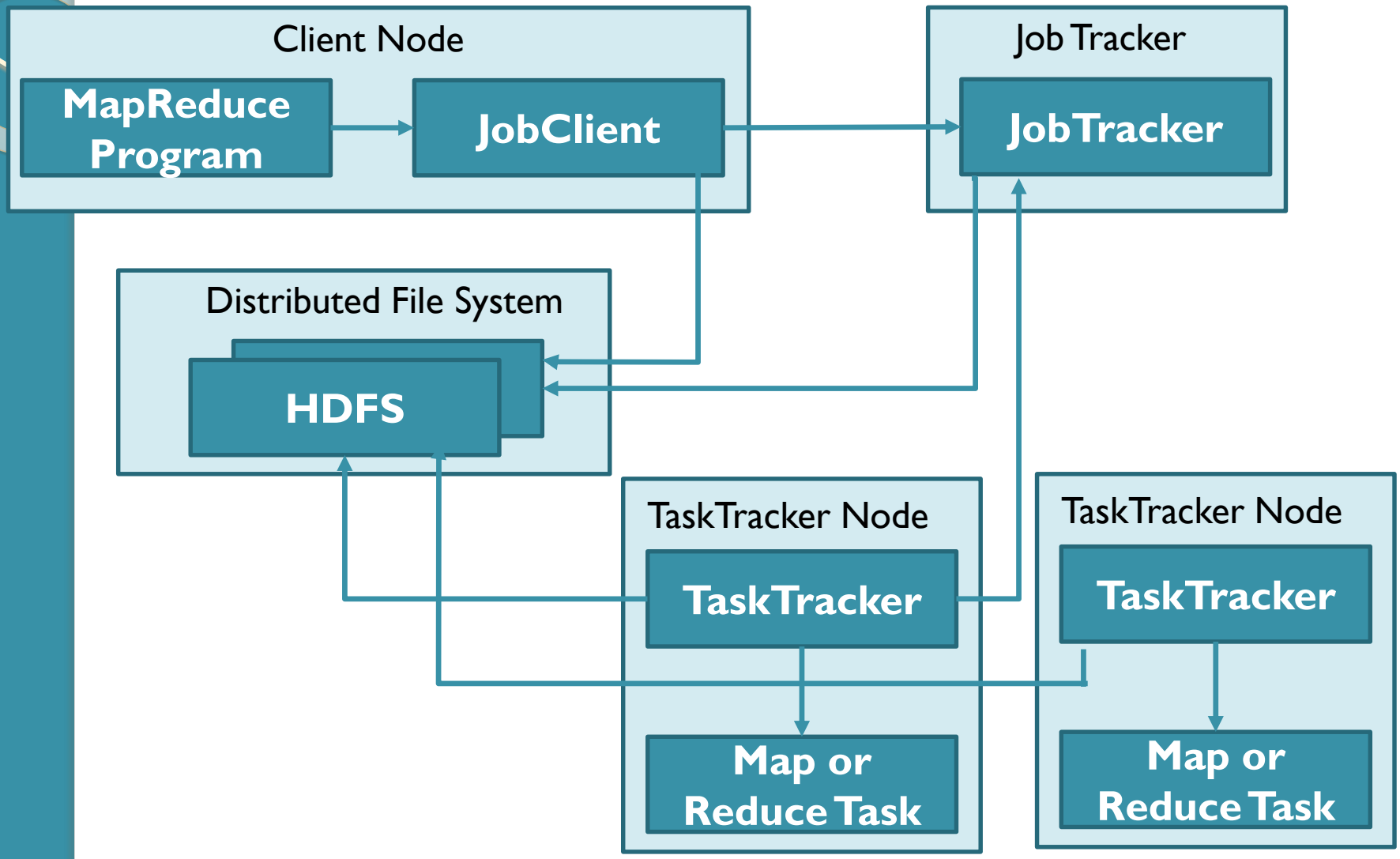
DataNode

- A DataNode stores data in an HDFS file system.
- A functional HDFS filesystem has more than one DataNode, with data replicated across them.
- DataNodes connect to the NameNode on startup.
- DataNodes respond to requests from the NameNode for filesystem operations.
- Client applications can talk directly to a DataNode, once the NameNode has provided the location of the data.
- MapReduce operations are assigned to TaskTracker instances near a DataNode, talk directly to the DataNode to access the files.
- TaskTracker instances can be deployed on the same servers that host DataNode instances, so that MapReduce operations are performed close to the data.

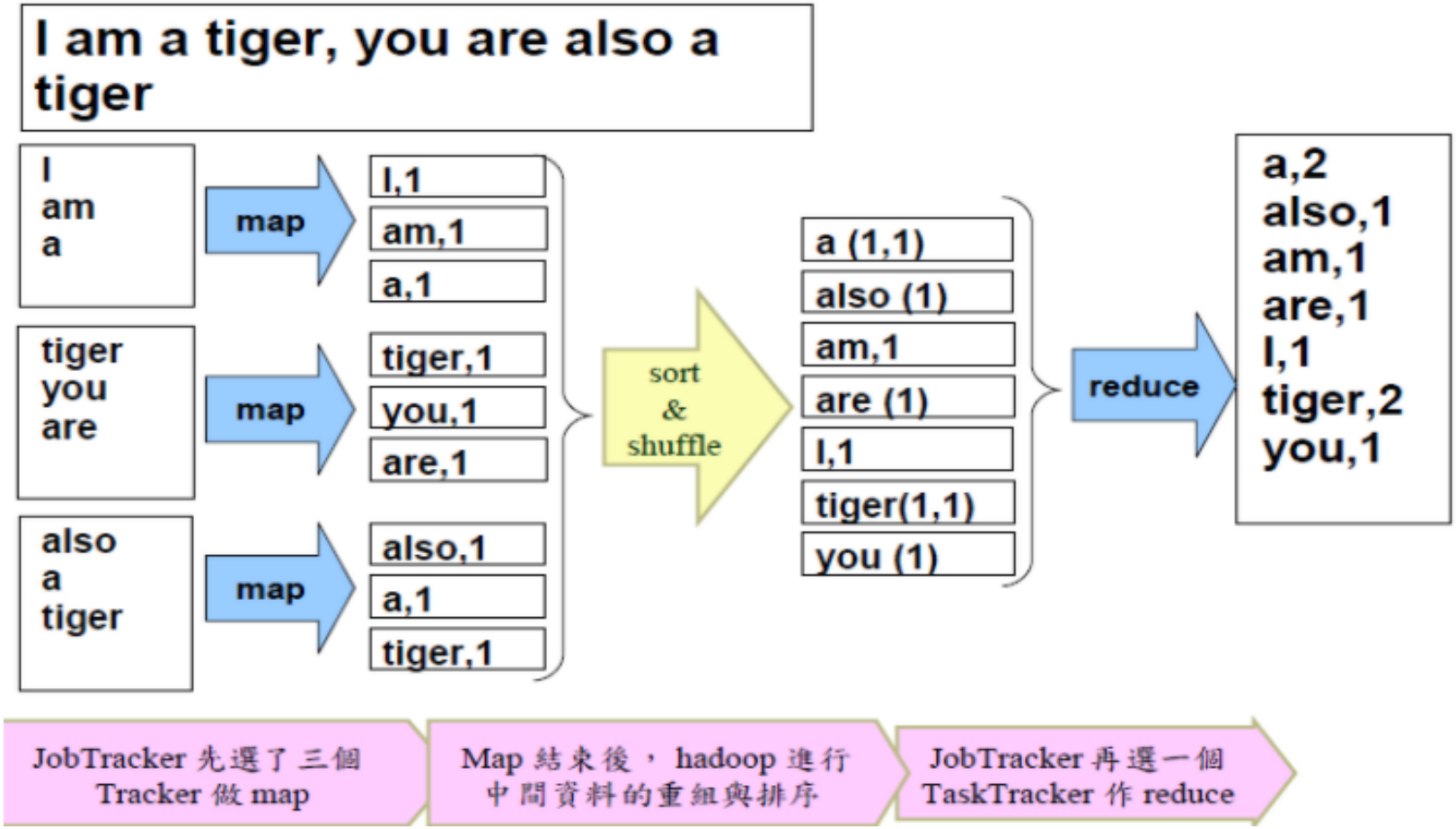
Secondary NameNode

- The NameNode is a single point of failure for the HDFS cluster.
- When the NameNode goes down, the file system goes offline.
- An optional Secondary NameNode which is hosted on a separate machine creates checkpoints of the namespace.

MapReduce Job Execution Workflow



MapReduce Example



Summary

- Hadoop is an open source framework for distributed batch processing of massive scale data.
- HDFS is a distributed file system that runs on large clusters and provides high throughput access to data.
- The Hadoop MapReduce provides a data processing model and an execution environment for MapReduce jobs for large scale data processing.
- Key processes of Hadoop include NameNode, Sceondary Name Node, JobTracker, TaskTracker and DataNode.

Spark
-Python & Spark-

Spark

- Spark is one of the latest technologies being used to quickly and easily handle Big Data
- It is an open source project on Apache
- It was first released in February 2013 and has exploded in popularity due to it's ease of use and speed
- It was created at the AMPLab at UC Berkeley

Spark

- You can think of Spark as a flexible alternative to MapReduce
- Spark can use data stored in a variety of formats
  - Cassandra
  - AWS S3
  - HDFS
  - And more

Spark vs MapReduce

- MapReduce requires files to be stored in HDFS, Spark does not!
- Spark also can perform operations up to 100x faster than MapReduce
- So how does it achieve this speed?

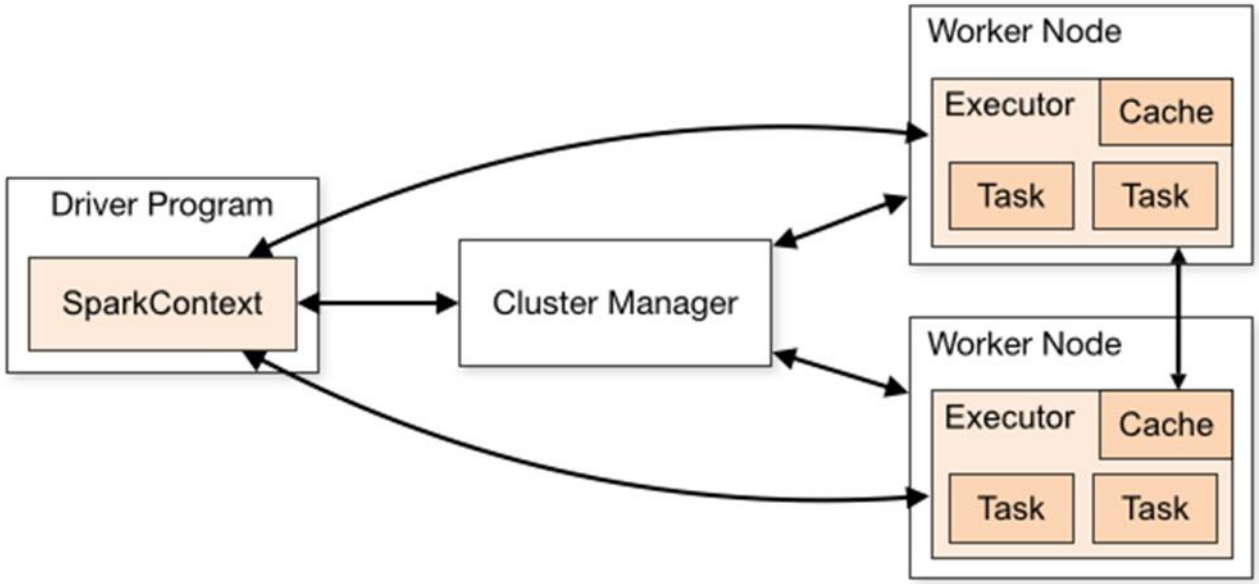
Spark vs MapReduce (Cont.)

- MapReduce writes most data to disk after each map and reduce operation
- Spark keeps most of the data in memory after each transformation
- Spark can spill over to disk if the memory is filled

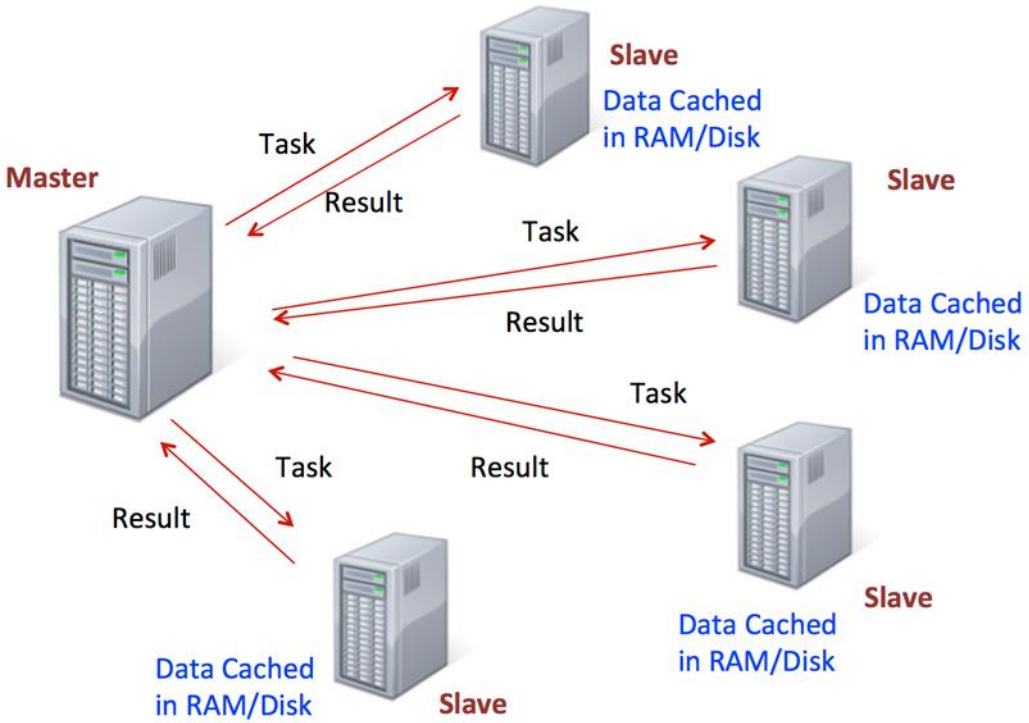
In-Memory Concept of Spark

- Improved efficiency over MapReduce (Hadoop)
  - 100x in-memory, 2-10x on disk

Spark Framework



Spark Framework (Cont.)



Spark RDDs

- At the core of Spark is the idea of a Resilient Distributed Dataset (RDD)
- Resilient Distributed Dataset (RDD) has 4 main features:
  - Distributed Collection of Data
  - Fault-tolerant
  - Parallel operation - partioned
  - Ability to use many data sources

Spark RDDs (Cont.)

- RDDs are immutable, lazily evaluated, and cacheable
- There are two types of Spark operations:
  - Transformations
  - Actions
- Transformations are basically a recipe to follow.
- Actions actually perform what the recipe says to do and returns something back.

Spark RDDs (Cont.)

- This behaviour carries over to the syntax when coding.
- A lot of times you will write a method call, but won't see anything as a result until you call the action.
- This makes sense because with a large dataset, you don't want to calculate all the transformations until you are sure you want to perform them!

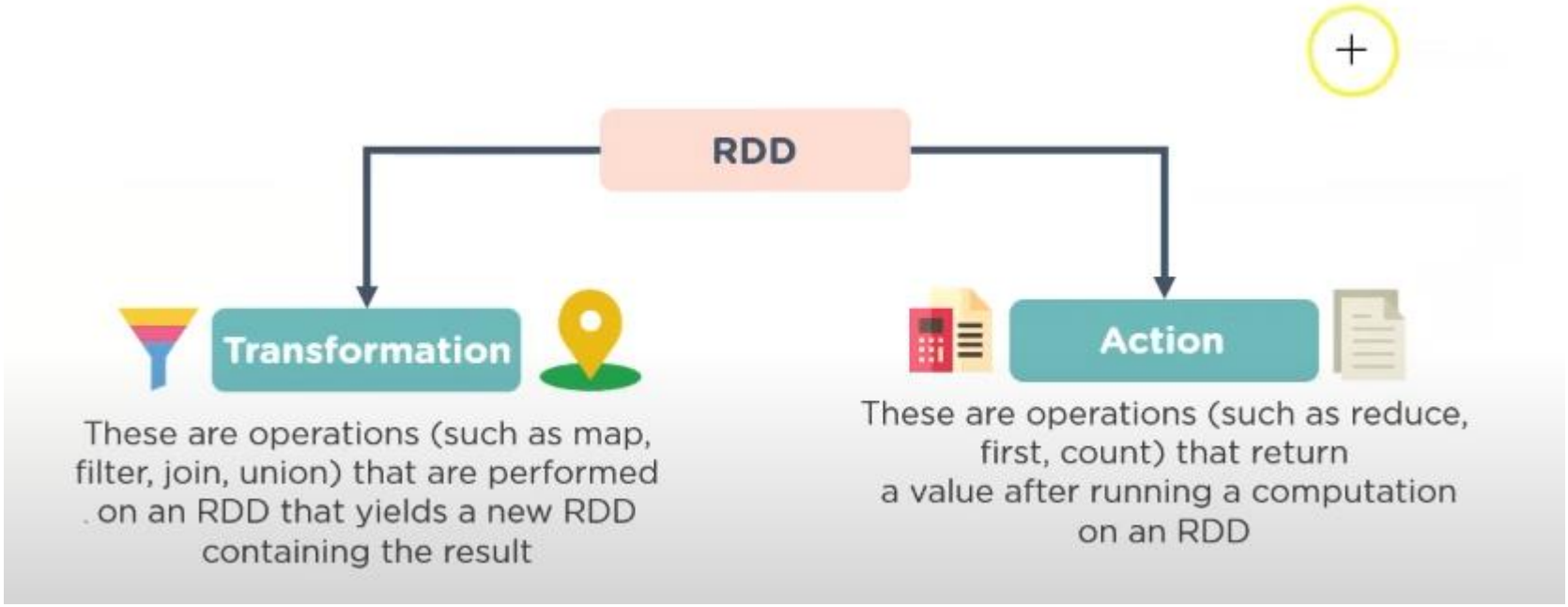
Spark RDDs (Cont.)

- When discussing Spark syntax you will see RDD versus DataFrame syntax show up.
- With the release of Spark 2.0, Spark is moving towards a DataFrame based syntax, but keep in mind that the way files are being distributed can still be thought of as RDDs, it is just the typed out syntax that is changing

Resilient Distributed Datasets (RDDs)

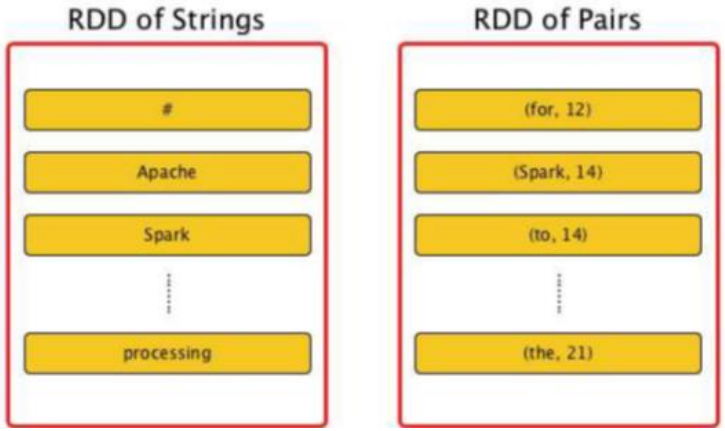
- In order to operate the programs on distributed system
- To spread partitioned dataset across the cloud
  - The data partition, stored in memory or disk (HDFS)
- RDDs operations through a diverse sets
  - transformations (map, filter, join)
  - actions (count, collect, save)
- Advantage of RDDs
  - Is automatically rebuilt on machine failure
  - Immutable data structure

RDD Transformation & Action (Cont.)



RDD Transformation Operations

- Transformation is a transformation that passes each dataset element through a function and returns a new RDD representing the results
  - To define new RDDs from the current RDDs
  - Lazy evaluation (process until you have to make an action)



RDD Transformations

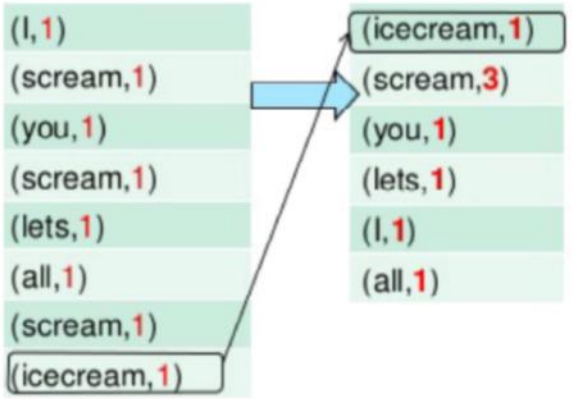
Transformations

The following table lists some of the common transformations supported by Spark. Refer to the RDD API doc (Scala, Java, Python) and pair RDD functions doc (Scala, Java) for details.

Transformation	Meaning
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.
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).
intersection(*otherDataset*)	Return a new RDD that contains the intersection of elements in the source dataset and the argument.
distinct([*numTasks*]))	Return a new dataset that contains the distinct elements of the source dataset.
groupByKey([*numTasks*])	When called on a dataset of (K, V) pairs, returns a dataset of (K, Iterable<V>) pairs. **Note:** If you are grouping in order to perform an aggregation (such as a sum or average) over each key, using `reduceByKey` or `combineByKey` will yield much better performance. **Note:** By default, the level of parallelism in the output depends on the number of partitions of the parent RDD. You can pass an optional `numTasks` argument to set a different number of tasks.
reduceByKey(*func*, [*numTasks*])	When called on a dataset of (K, V) pairs, returns a dataset of (K, V) pairs where the values for each key are aggregated using the given reduce function *func*, which must be of type (V,V) => V. Like in `groupByKey`, the number of reduce tasks is configurable through an optional second argument.
sortByKey([*ascending*], [*numTasks*])	When called on a dataset of (K, V) pairs where K implements Ordered, returns a dataset of (K, V) pairs sorted by keys in ascending or descending order, as

RDD Action Operations

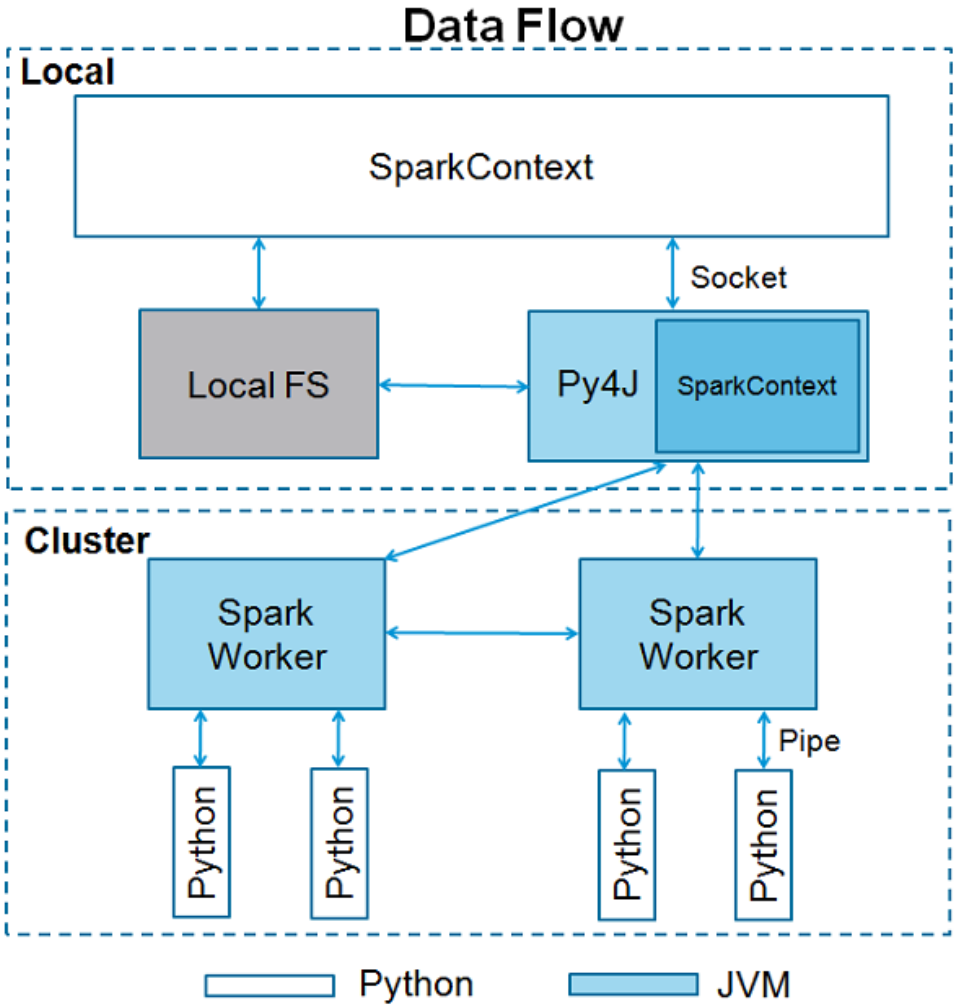
- The action will trigger a spark execution and return the results to driver
  - To obtain the results on the screen or to write to the file system, we use action operation
  - Count, collect, save, …
- Example of collect()
  Two RDDs in the figure



Spark DataFrames

- Spark DataFrames are also now the standard way of using Spark's Machine Learning Capabilities.
- Spark DataFrame documentation is still pretty new and can be sparse.
- Let's get a brief tour of the documentation!

PySpark Framework



Python and Spark

- In this course the main way we will be working with Python and Spark is through the DataFrame Syntax.
- If you've worked with pandas in Python, R, SQL or even Excel, a DataFrame will feel very familiar!

Python and Spark (Cont.)

- Spark DataFrames hold data in a column and row format.
- Each column represents some feature or variable.
- Each row represents an individual data point.

Python and Spark (Cont.)

- Spark began with something known as the "RDD" syntax which was a little ugly and tricky to learn.
- Now Spark 2.0 and higher has shifted towards a DataFrame syntax which is much cleaner and easier to work with!

Python and Spark (Cont.)

- Spark DataFrames are able to input and output data from a wide variety of sources.
- We can then use these DataFrames to apply various transformations on the data.

Python and Spark (Cont.)

- At the end of the transformation calls, we can either show or collect the results to display or for some final processing.
- In this section we'll cover all the main features of working with DataFrames that you need to know.

Python and Spark (Cont.)

- Once we have a solid understanding of Spark DataFrames, we can move on to utilizing the DataFrame.

PySpark - DataFrame

- You can manually create a PySpark DataFrame using createDataFrame() methods, this function takes different signatures in order to create DataFrame from existing RDD, list, and DataFrame.

- You can also create PySpark DataFrame from data sources like TXT, CSV, JSON, ORV, Avro, Parquet, XML formats by reading from HDFS, S3, DBFS, Azure Blob file systems

PySpark Tutorial : https://sparkbyexamples.com/pyspark-tutorial/

PySpark – DataFrame (Cont.)

Example of Creating DataFrame

```
from pyspark.sql.types import StructType,StructField, StringType, IntegerType
data2 = [("James","","Smith","36636","M",3000),
    ("Michael","Rose","","40288","M",4000),
    ("Robert","","Williams","42114","M",4000),
    ("Maria","Anne","Jones","39192","F",4000),
    ("Jen","Mary","Brown","","F",-1)
  ]

schema = StructType([ \
    StructField("firstname",StringType(),True), \
    StructField("middlename",StringType(),True), \
    StructField("lastname",StringType(),True), \
    StructField("id", StringType(), True), \
    StructField("gender", StringType(), True), \
    StructField("salary", IntegerType(), True) \
  ])

df = spark.createDataFrame(data=data2,schema=schema)
df.printSchema()
df.show(truncate=False)
```

PySpark – DataFrame (Cont.)

Results of Creating DataFrame

```
root
 |-- firstname: string (nullable = true)
 |-- middlename: string (nullable = true)
 |-- lastname: string (nullable = true)
 |-- id: string (nullable = true)
 |-- gender: string (nullable = true)
 |-- salary: integer (nullable = true)

+---------+----------+--------+-----+------+------+
|firstname|middlename|lastname|id   |gender|salary|
+---------+----------+--------+-----+------+------+
|James    |          |Smith   |36636|M     |3000  |
|Michael  |Rose      |        |40288|M     |4000  |
|Robert   |          |Williams|42114|M     |4000  |
|Maria    |Anne      |Jones   |39192|F     |4000  |
|Jen      |Mary      |Brown   |     |F     |-1    |
+---------+----------+--------+-----+------+------+
```

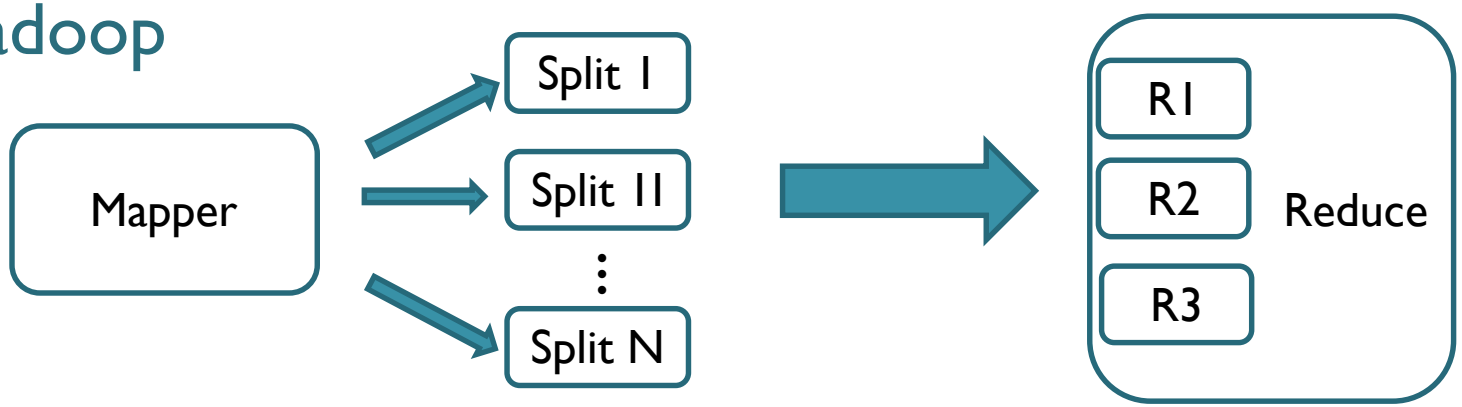
PySpark – DataFrame (Cont.)

Example of Reading DataFrame

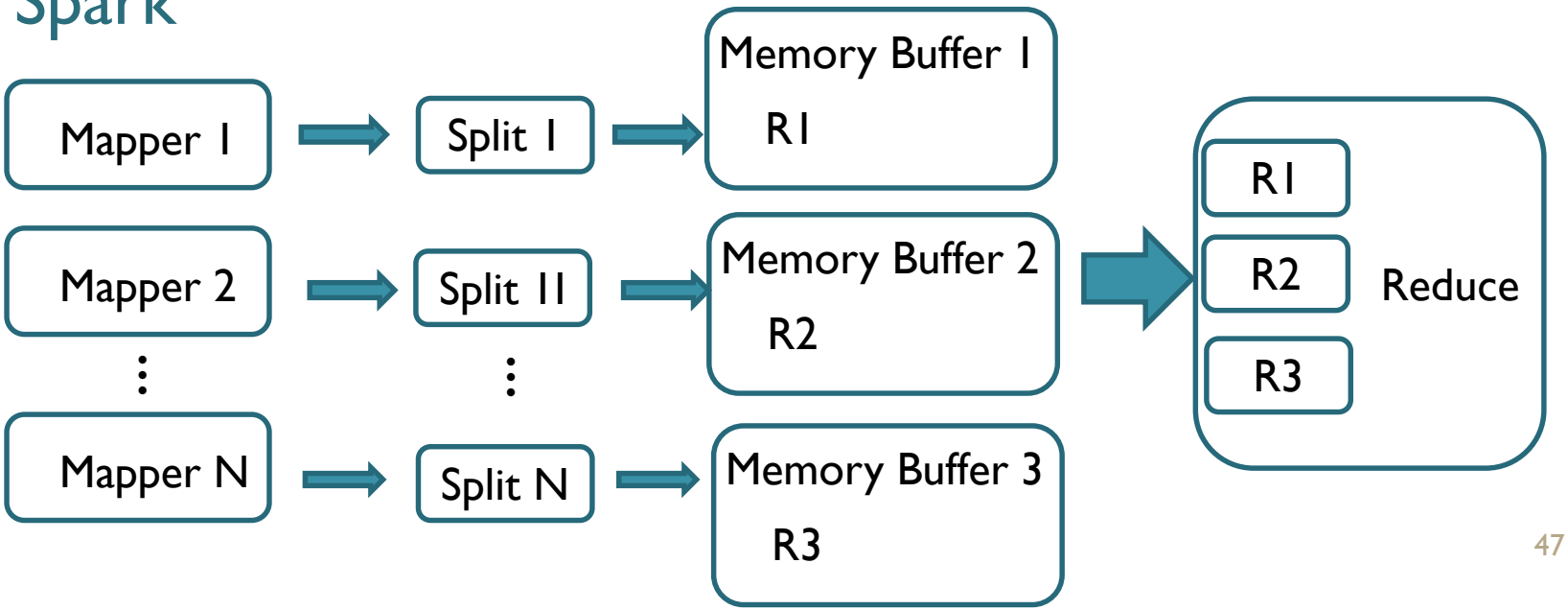
```
df2 = spark.read.csv("/src/resources/file.csv")

df2 = spark.read.text("/src/resources/file.txt")

df2 = spark.read.json("/src/resources/file.json")
```

Hadoop MapReduce vs Spark

Hadoop



Spark



Thanks for Your Attention!