Big Data Modeling & Analytics

(course outline)

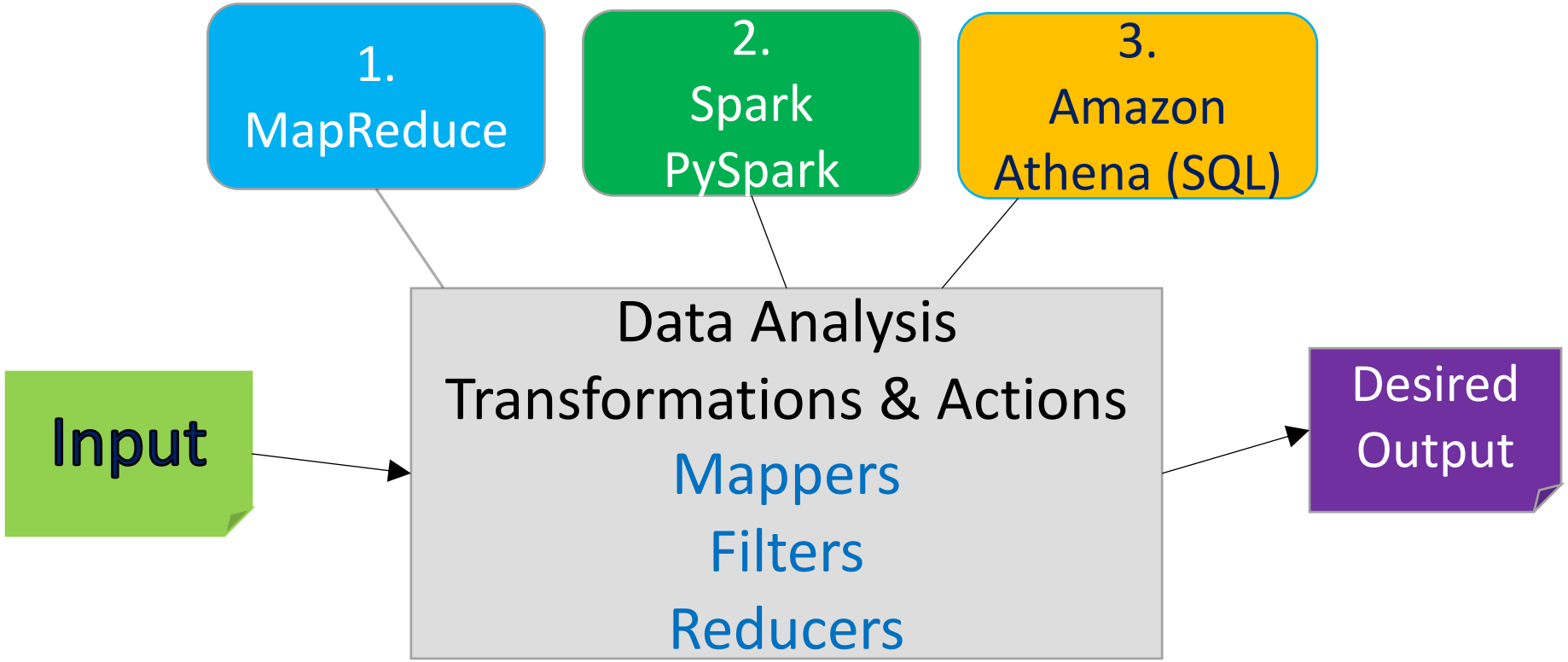
Mahmoud Parsian

Ph.D. in Computer Science

Main Course Components

1. Introduction to MapReduce Paradigm (25%)
2. Spark and PySpark (60%)
3. Serverless SQL Access to Big Data (15%)

Data Analysis



1. MapReduce Paradigm

- **MapReduce** is a programming **paradigm/model** that enables **massive scalability** across hundreds or thousands of servers in a cluster.
- MapReduce has 3 functions:
  - `map()`
  - `reduce()`
  - `combine()`

1. MapReduce Programs

- **For MapReduce, we will NOT use any concrete implementations (such as Hadoop)**
- For MapReduce, we will focus on learning MapReduce **paradigm/model** that enables **massive scalability** across hundreds or thousands of servers in a cluster.
- For MapReduce, we will write only **pseudo-code** (examples are given in Lin, J., & Dyer, C. (2010). *Data-intensive text processing with MapReduce)*

1. MapReduce Components

MapReduce has 3 functions:

map()

- Mapper function

reduce()

- Reducer function

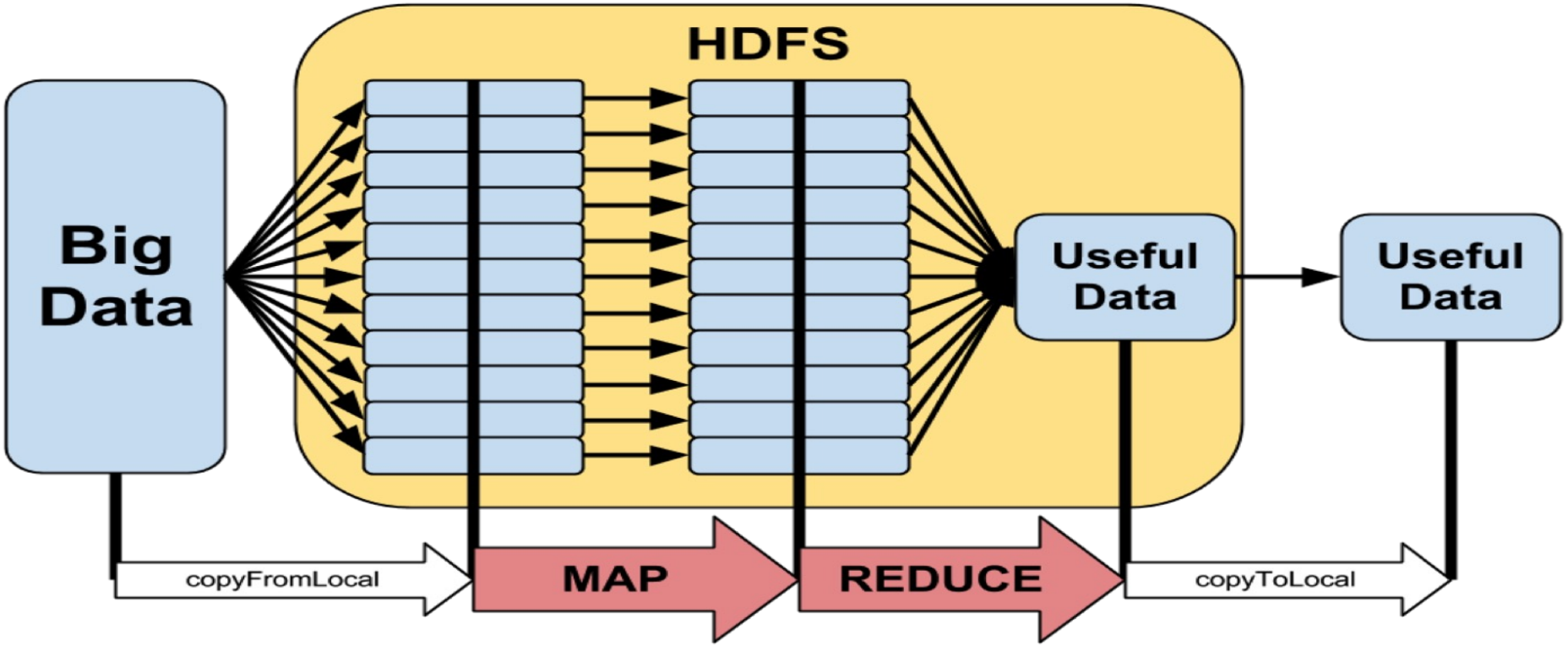
combine()

- Optional combiner function

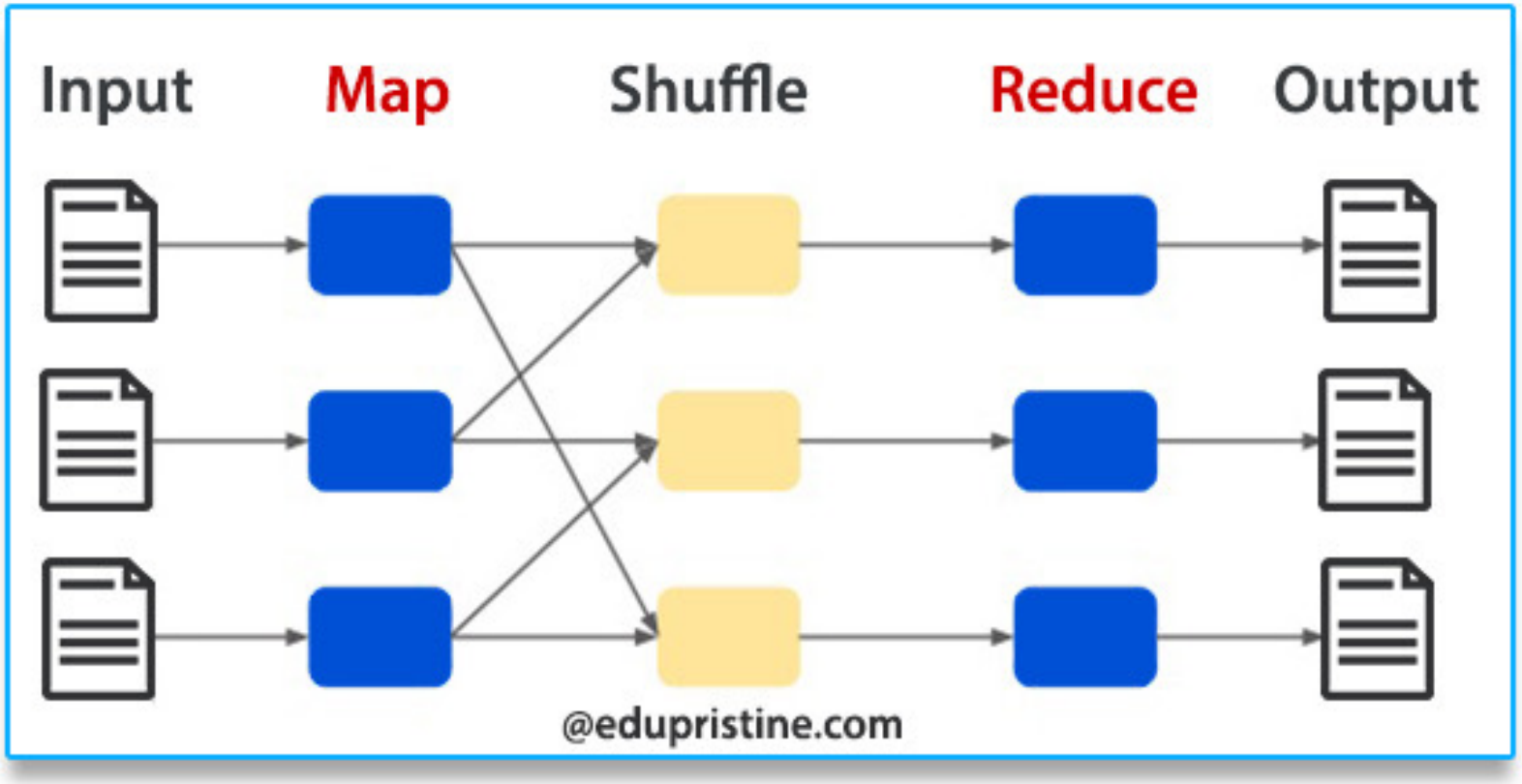
MapReduce Paradigm

- **MapReduce is a foundation model/paradigm for distributed computing using clusters**
- **MapReduce concrete implementations:**
  - Apache Hadoop implements MapReduce
  - Apache Spark implements superset of MapReduce
  - Apache Tez implements MapReduce

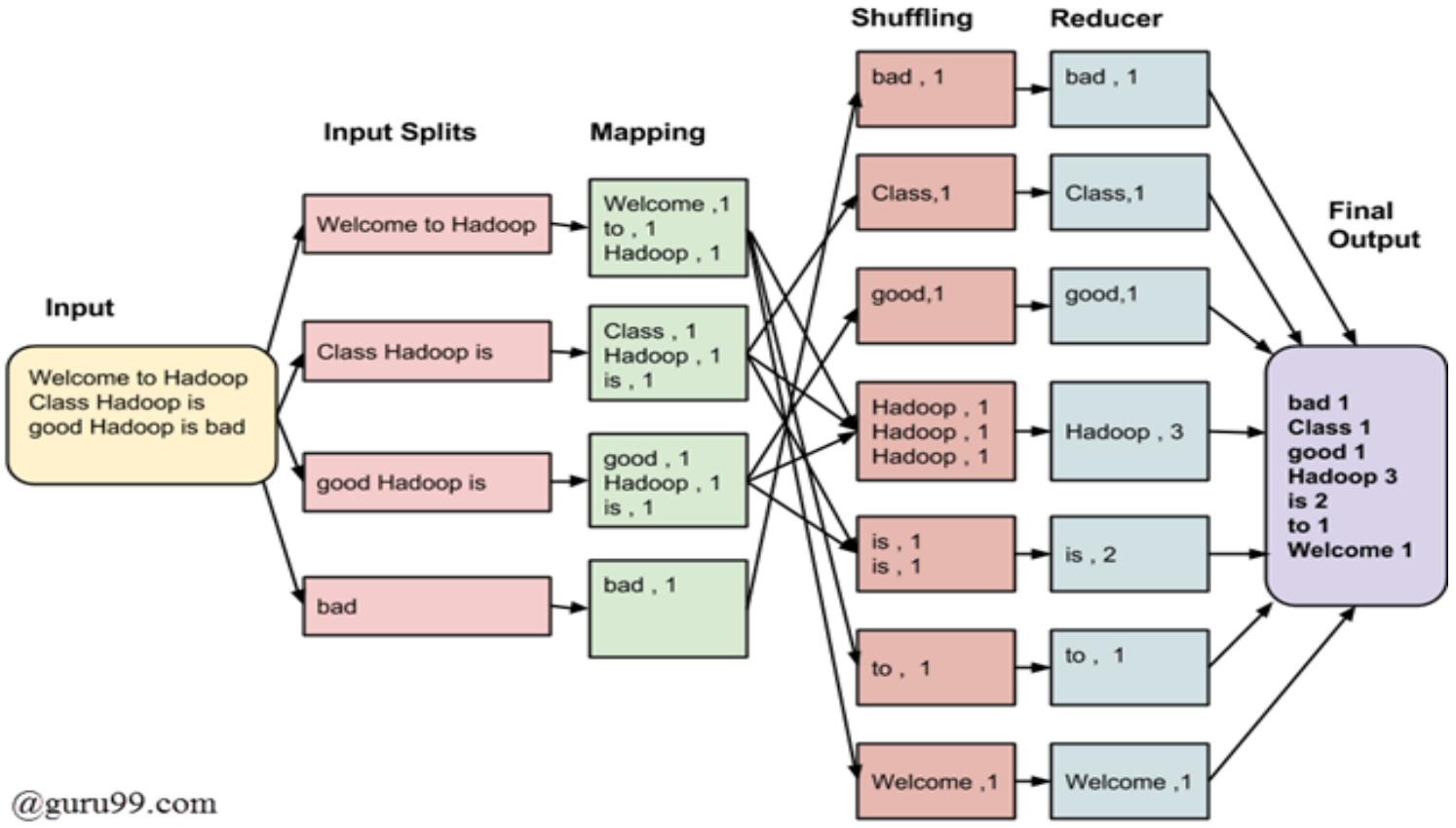
MapReduce Paradigm: High-Level



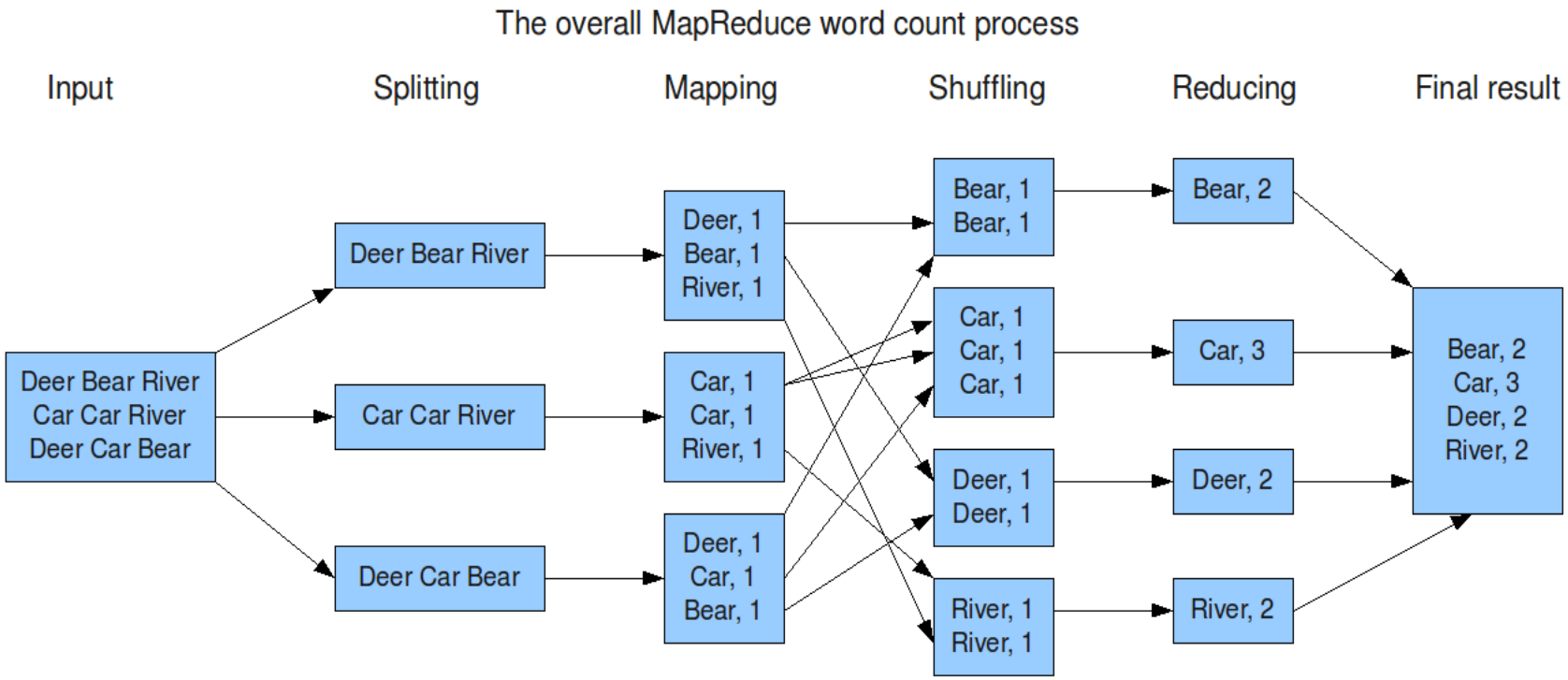
MapReduce Paradigm



MapReduce Paradigm Example-1



MapReduce Paradigm Example-2



Solving a Problem with MapReduce (1)

- **Input data is partitioned by a partitioner into chunks and sent to Mappers**
- **Assume your input is comprised of 80,000,000,000 records, which might be partitioned into 40,000 chunks**
  - Number of partitions: 40,000
  - Each partition will have about 2,000,000 records
  - Maximum mapper parallelism can be 40,000 mappers
  - 80,000,000,000 = 40,000 x 2,000,000
- **A mapper function, map(K, V) is executed on all partitions in parallel (as much as possible – depends on resources available).**
- **The map(K, V) generates any number of (key, value) pairs such as:**
- `{(K1, V1), (K2, V2), …}`**. For example, K can be a record number and V can be the actual input record**

Solving a Problem with MapReduce (2)

- All mappers output go to Sort & Shuffle phase, which groups values by unique keys (similar to GROUP BY in SQL). Sort & Shuffle creates (key, value) pairs as:

```
(K_1, [v1, v2, …])
(K_2, [u1, u2, …])
…
(K_n, [t1, t2, …])
```

Where `{K_1, K_2, … K_n }` are the unique keys created by all mappers

Solving a Problem with MapReduce (3)

- Output of Sort & Shuffle goes to reducers
- A reducer operates on (key, value) pairs where key is in

`{ K_1, K_2, …, K_n }` and value is an associated value for the key.

Therefore, the following reducers can be executed in parallel:

```
reduce(K_1, [v1, v2, …])
reduce(K_2, [u1, u2, …])
…
reduce(K_n, [t1, t2, …])
```

- Each reducer can create any number of new (key, value) pairs.

2. Apache Spark

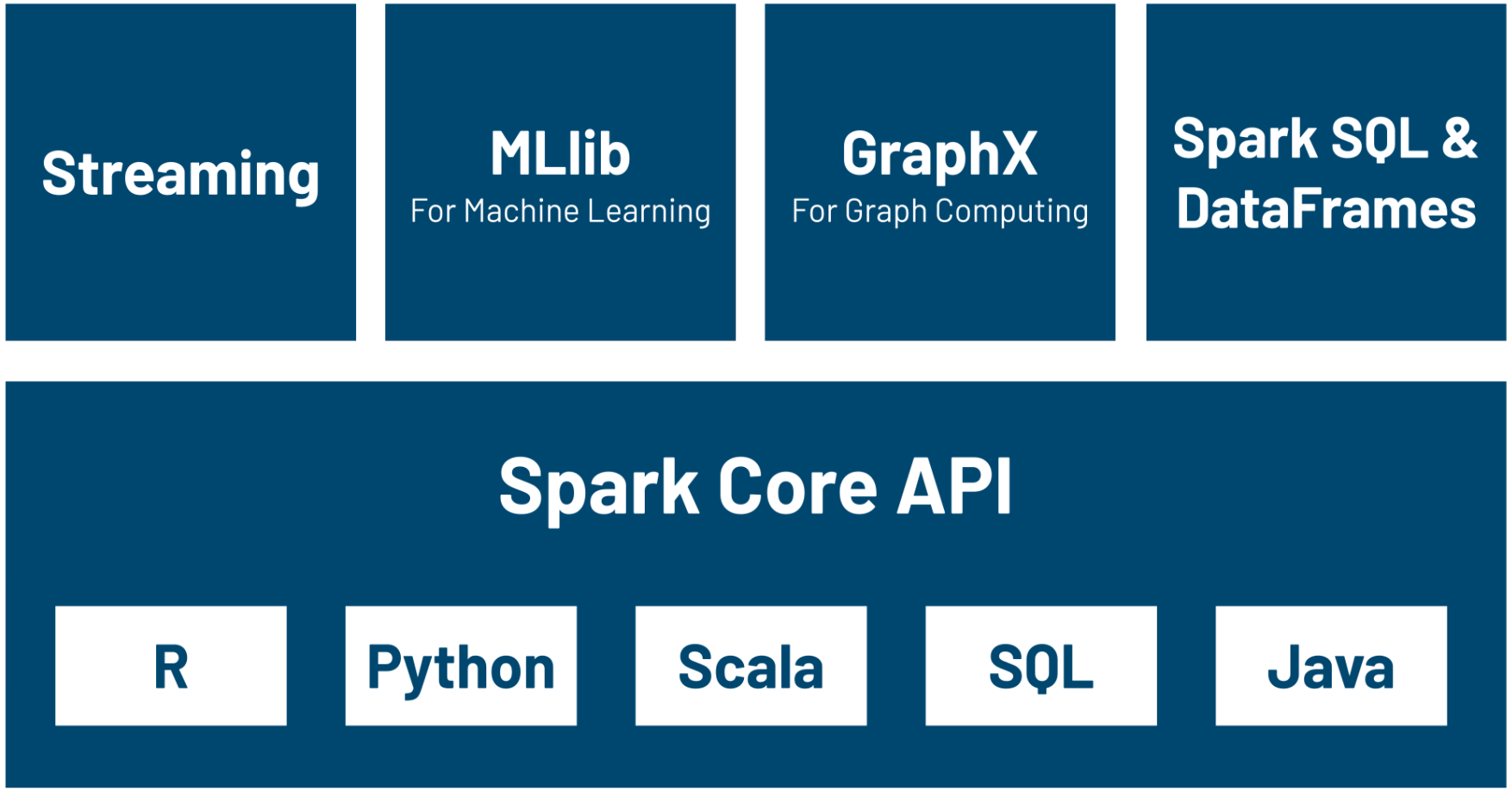
- Apache **Spark** is a multi-language (Java, Python, Scala) engine for executing data engineering, data science, and machine learning on single-node machines or clusters.
- PySpark is a Python API for Spark
- Spark is a superset of MapReduce
- Spark web site: https://spark.apache.org

2. Apache Spark Programs

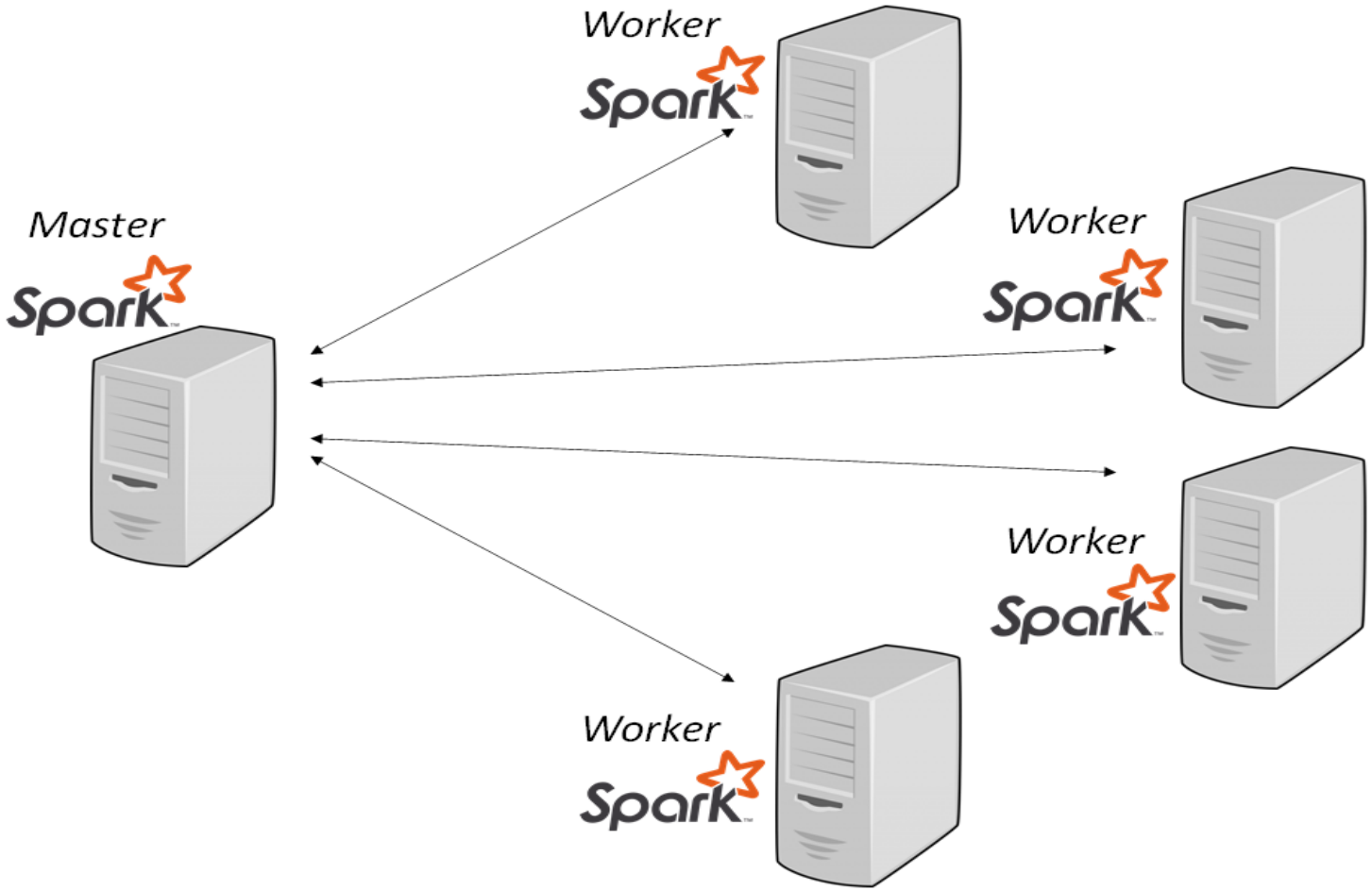
For Apache **Spark** , we will use PySpark and write **actual executable programs:**

```
>>> # spark is an instance of SparkSession object
>>> rdd = spark.sparkContext
                    .parallelize(range(0, 100))
>>> rdd.count()
100
>>> sum_of_values = rdd.reduce(lambda x, y: x+y)
>>> sum_of_values
4950
```

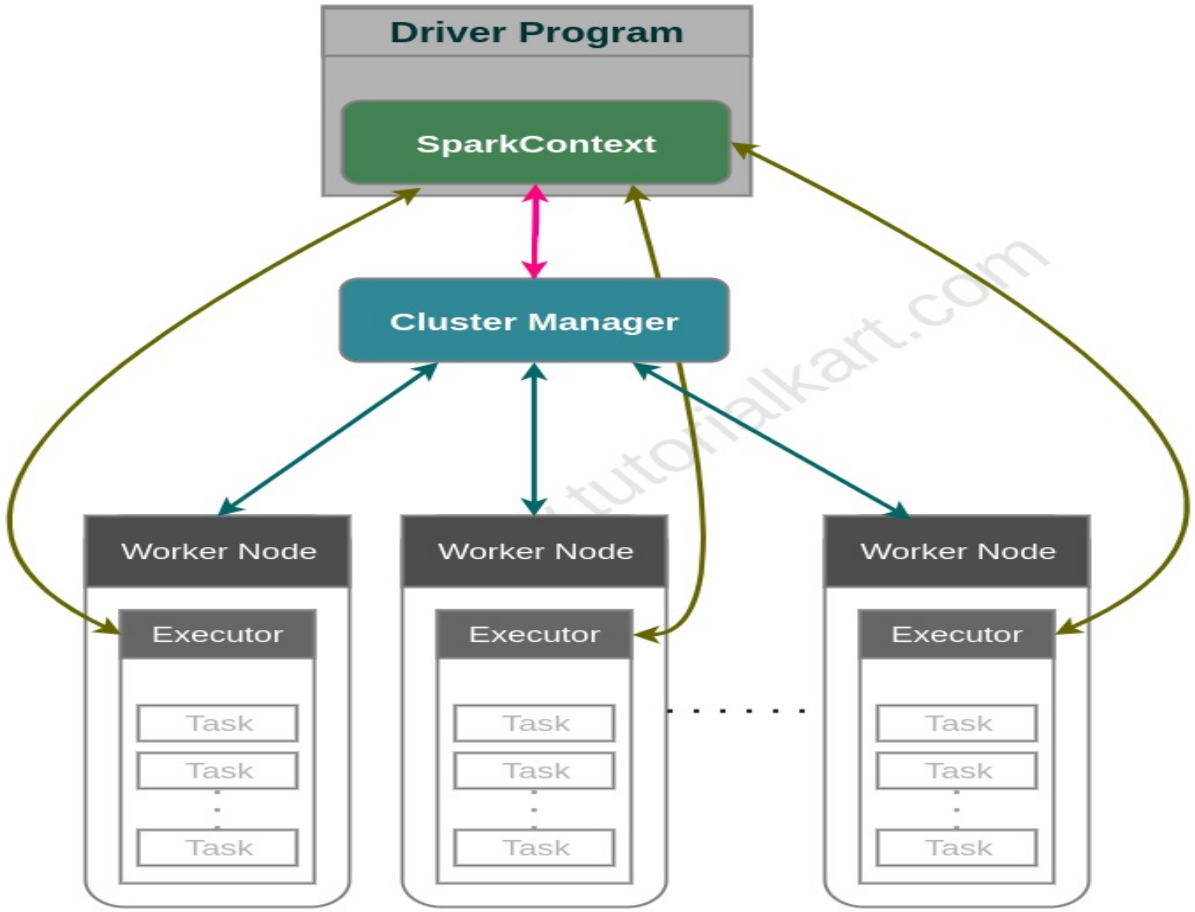
Apache Spark: Components



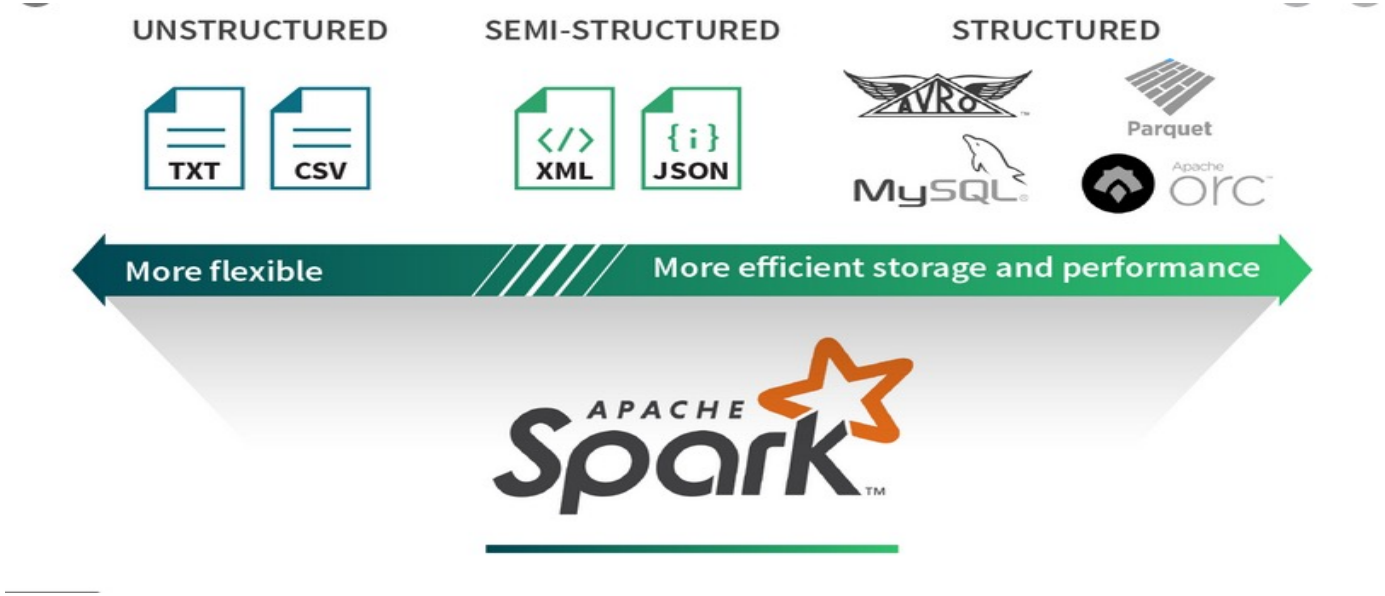
Apache Spark: runs in a cluster



Apache Spark: Cluster Manager



Spark Data Abstractions: Read/Write from/to Many DataSources



Spark Data Abstractions

- **Your data can be represented as an RDD or DataFrame**
- **Resilient Distributed Datasets (RDD)**
  - Data is represented as a data type T (integer, string, tuples, arrays, …)
  - Billions of data points (elements/records)
- **DataFrame**
  - Data is represented as a table of rows and named columns
  - Billions of rows of data

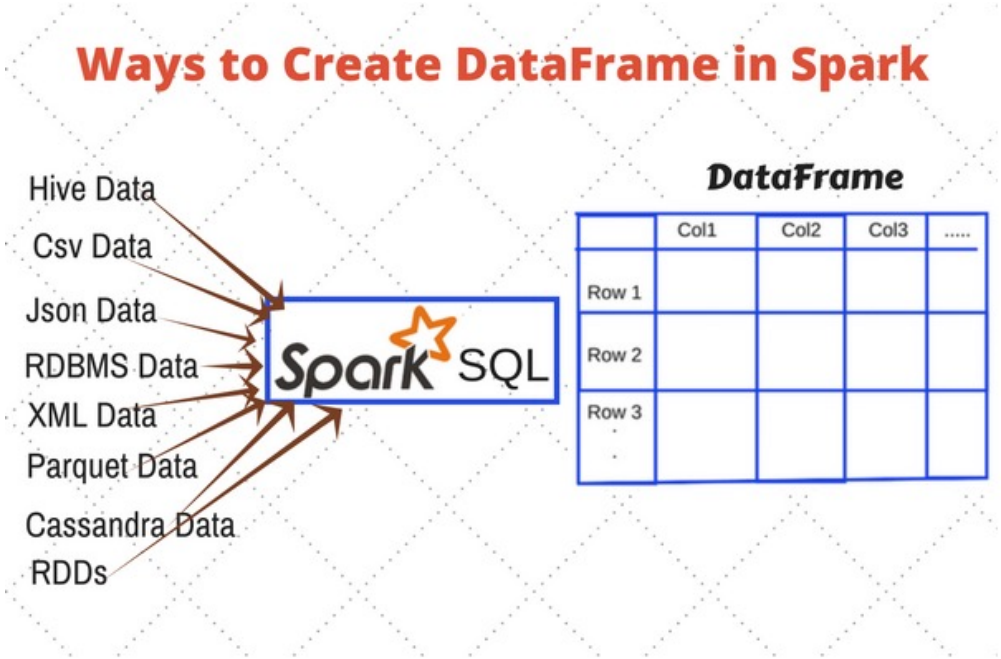
Spark Data Abstractions: RDDs

Billions of data elements

Each element: `(String, (Float, Float))`

Element-1:	`(gene-1, (3.0, 4.5))`
Element-2:	`(gene-2, (1.0, 1.5))`
Element-3:	`(gene-1, (2.1, 1.6))`
	…
	`(gene-8, (3.1, 5.1))`

Spark Data Abstractions: DataFrame



3. Main Course Components
Serverless SQL Access to Big Data (15%)

- Amazon Athena
- Google BigQuery
- Snowflake

Amazon Athena

- Interactive query service
- Serverless. Zero infrastructure. Zero administration.
- Put your data in S3
- Access your data by SQL
- Pay by query
- Fast performance

Amazon Athena