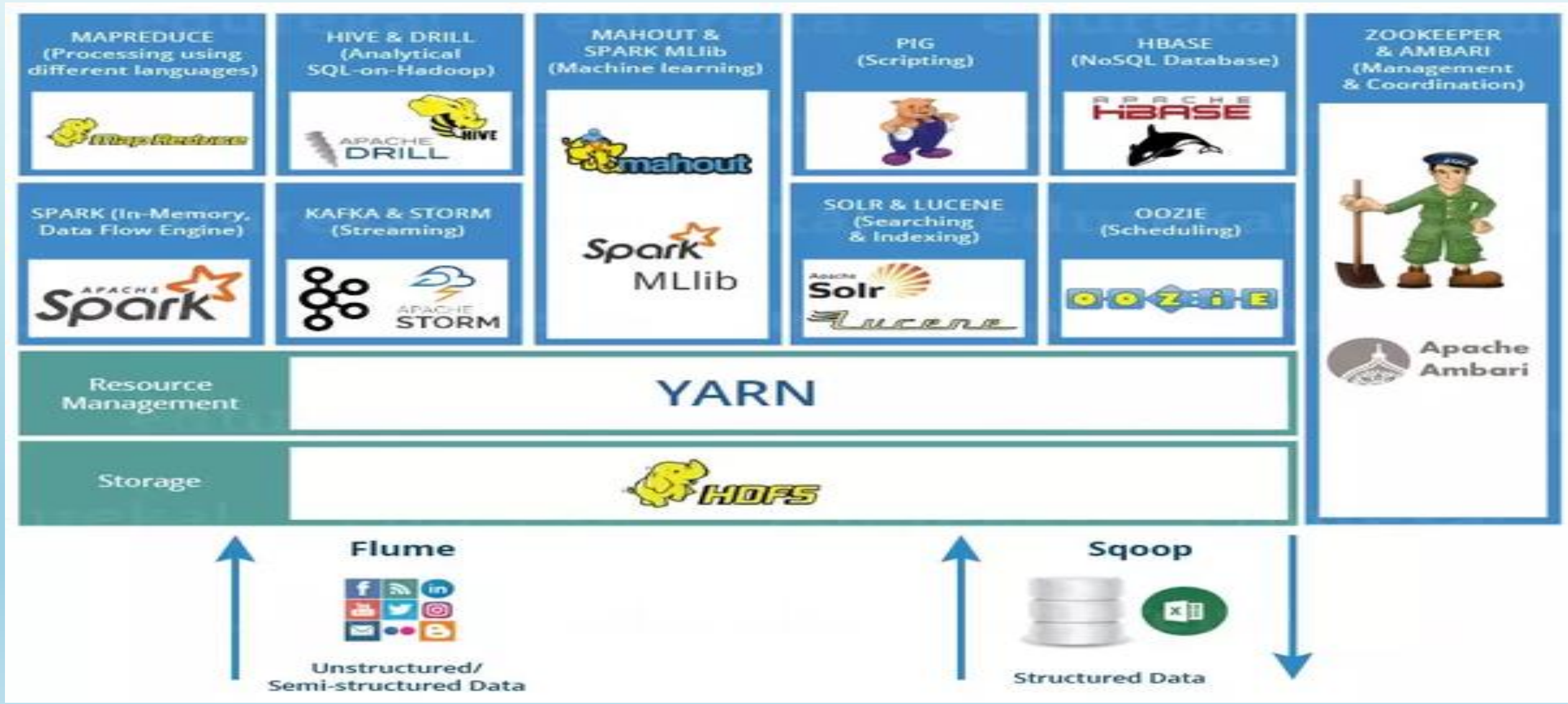
Apache Spark

Hadoop technology stack



Big data analytics Tools

Batch Data Analysis

Streaming Data Analysis

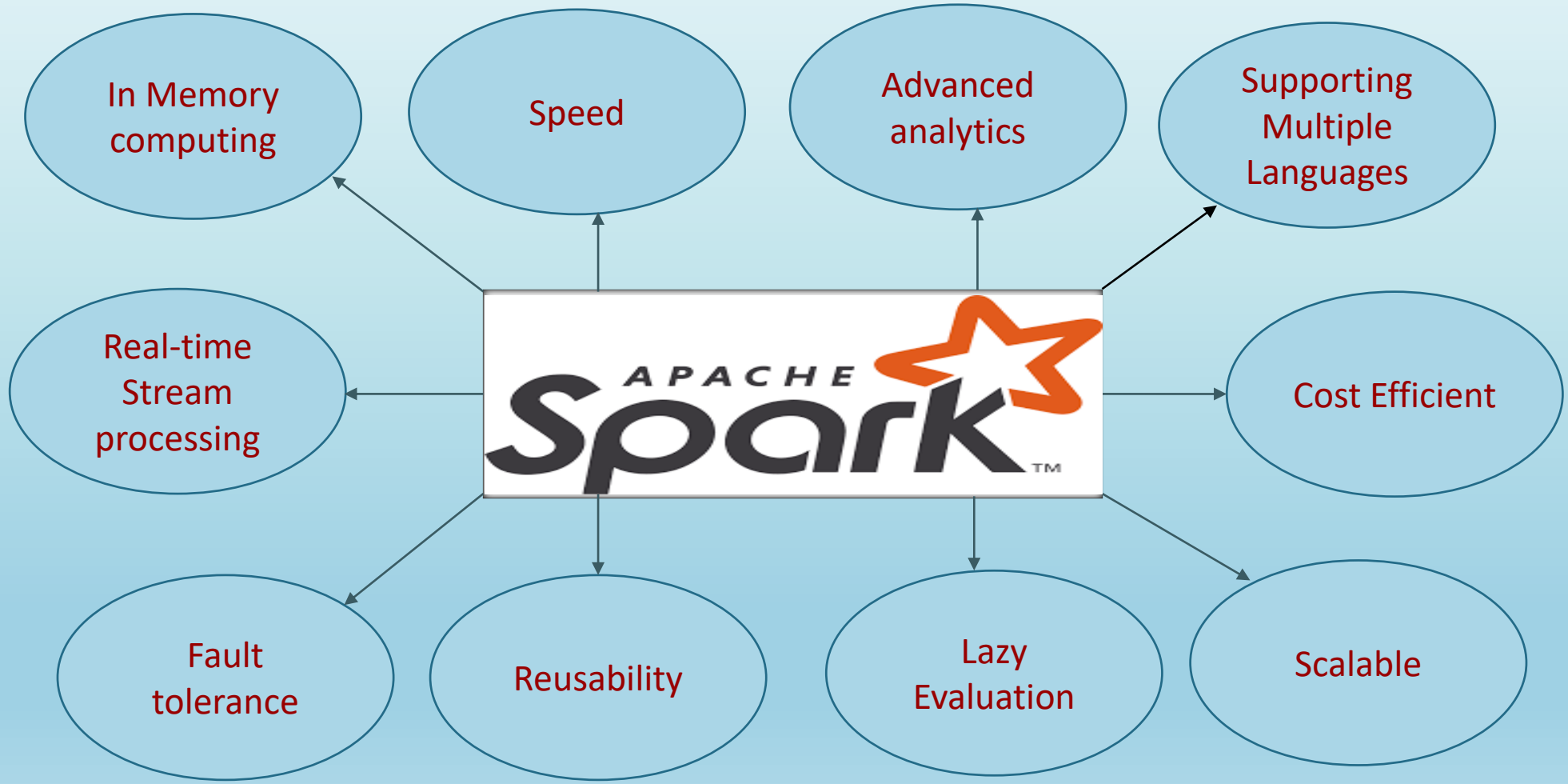
Big data analytics Tools

	Features	Spark	Storm	Samza	Flink
2	data processing	batch processing and stream processing systesm	stream processing	stream processing	batch processing as well as stream processing
3	Streaming engine	process data streams in micro-batches	process events one by one	process events one by one	process streams for workloads like SQL, batch and micro-batch
4	Data Flow	DAG	DAG with spouts, bolts and streams used to process data	relies on kakfa's semantics to define the way that streams are handled	Controlled Cyclic Dependency Graphs in runtime
5	Scalability	highly scalable, you can add N number of nodes in the cluster	scalable with parallel calculations	partitioned and distributed at every level	highly scalable, add N number of nodes in the cluster
6	latency	low latency	extremely low latency	low latency	low latency and high throughput
7	language support	java, python, scala, R	java, clojure	java, scala	java, python, scala, R

Introduction to Apache Spark

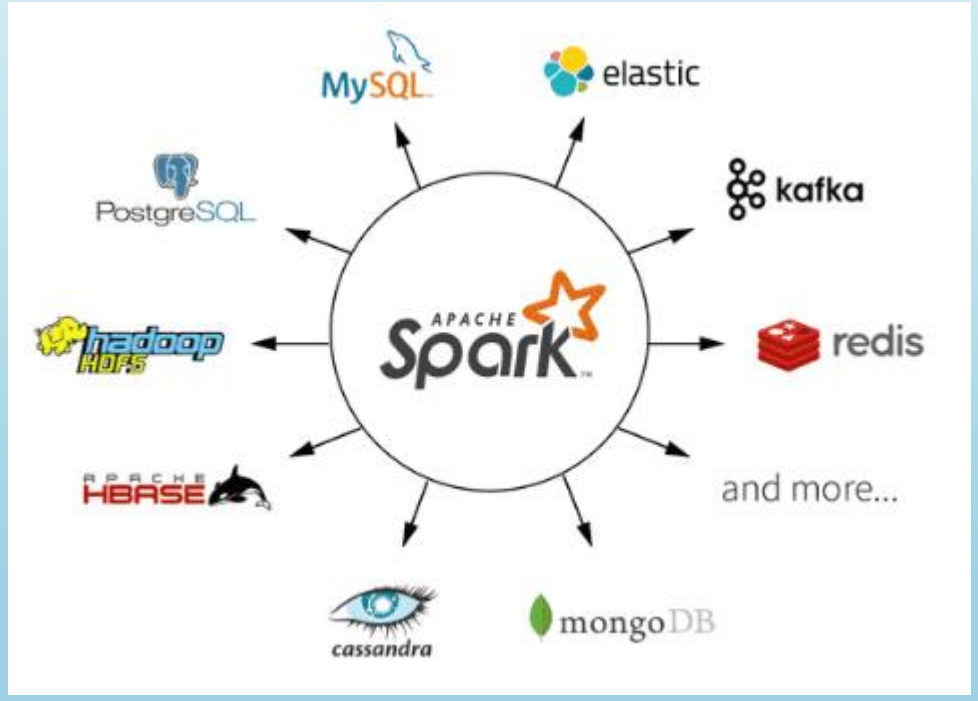
- Apache Spark is a Opensource general-purpose framework for cluster computing,
- It is used for a diverse range of applications.
- cluster computing in Spark designed to be fast and general-purpose.
- Spark can run in Hadoop clusters and access any Hadoop data source, including Cassandra.
- Spark is written in Scala but provides rich APIs support using Scala, Java, Python, and R.

Advantages of Apache Spark

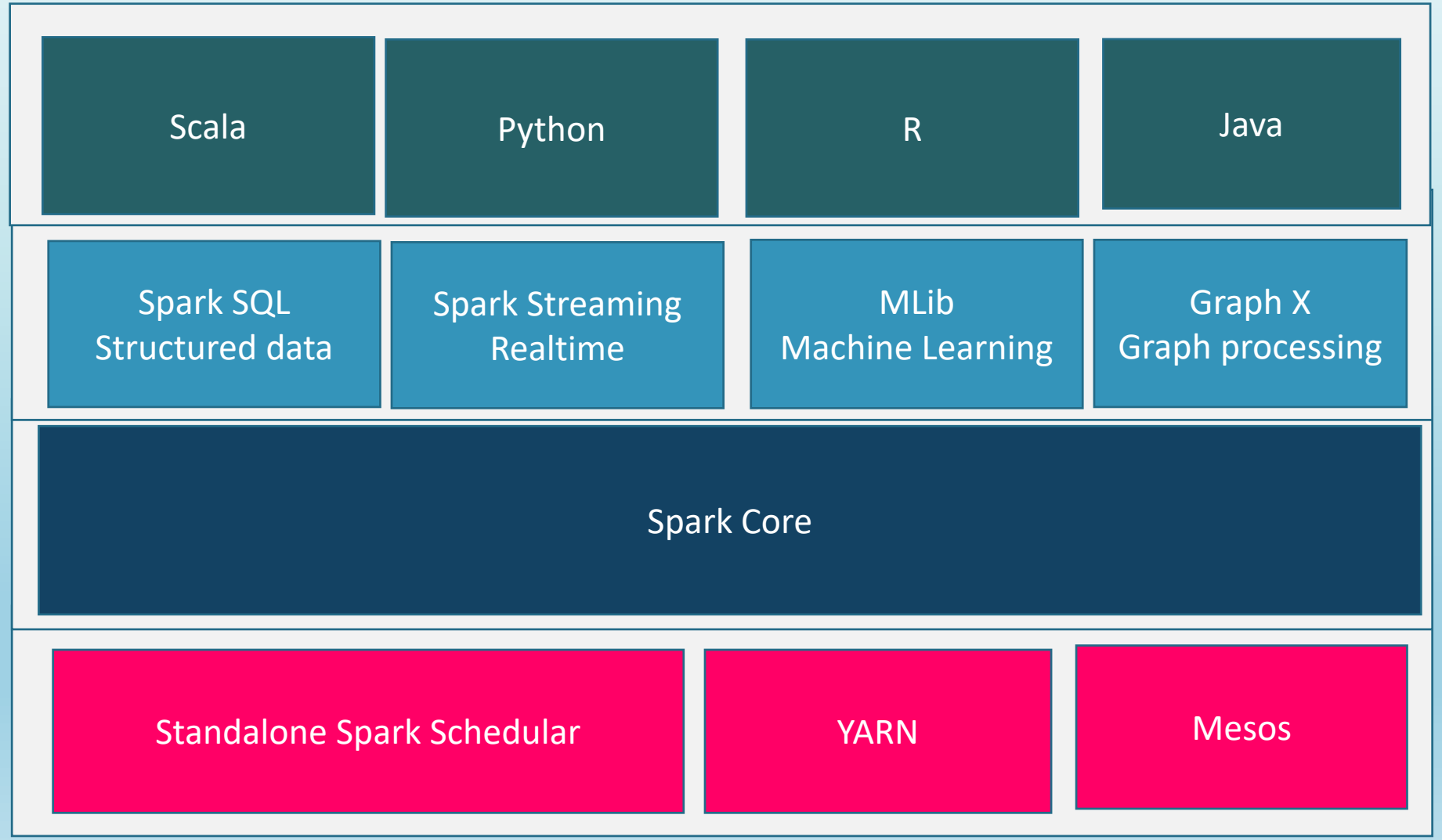


Company's Using Apache Spark

- Yahoo (Finance)
- Google (App Engine)
- eBay
- NASA
- Netflix
- Nokia
- IBM
- Amazon
- Facebook
- Salesforce
- .....and So on



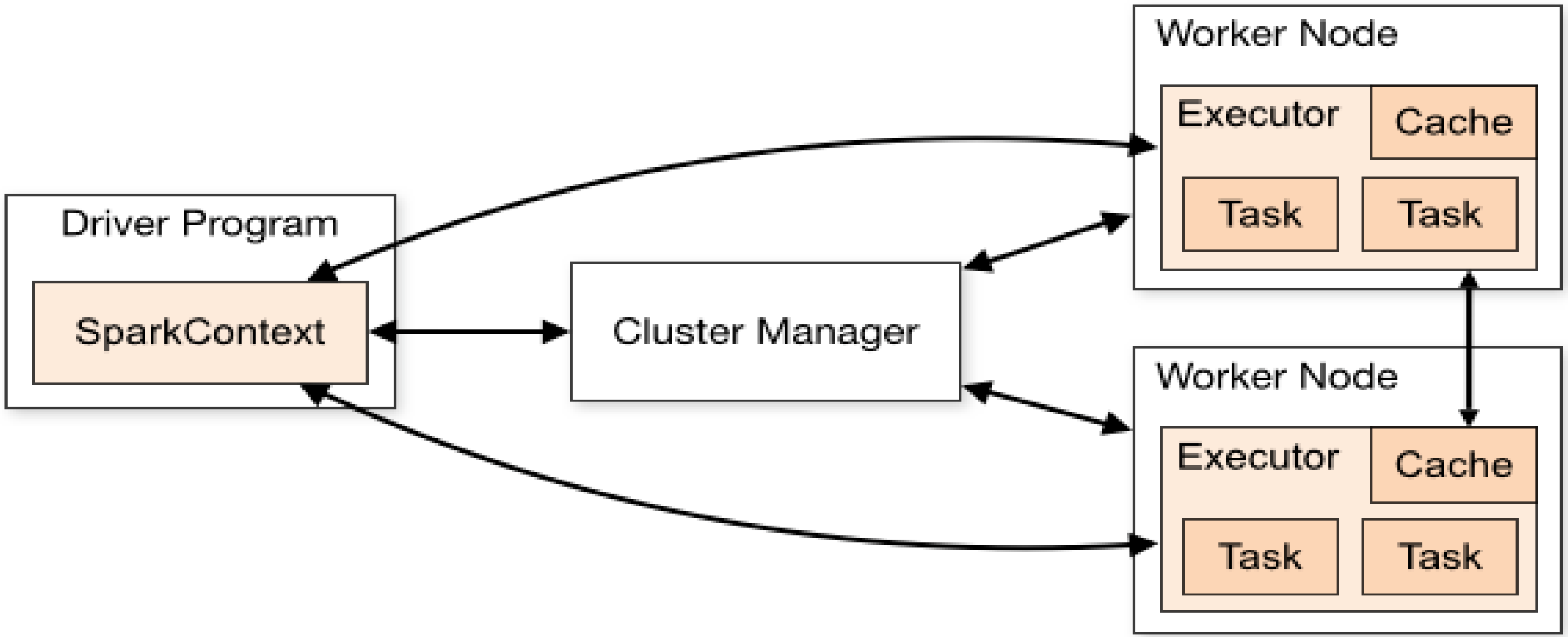
Spark Framework



Spark Framework Stack

Programming Support	Scala	python	R Programming	Java
Libraries	Spark SQL Structured data	MLib Machine Learning	Graph X Graph processing	Spark Streaming Realtime
Computational Engine	Spark Core			
Cluster Managers	Standalone Spark Schedular	YARN	Mesos	Kubernetes
Storage	Local	HDFS	S3	RDBMS	NOSQL

Spark Architecture

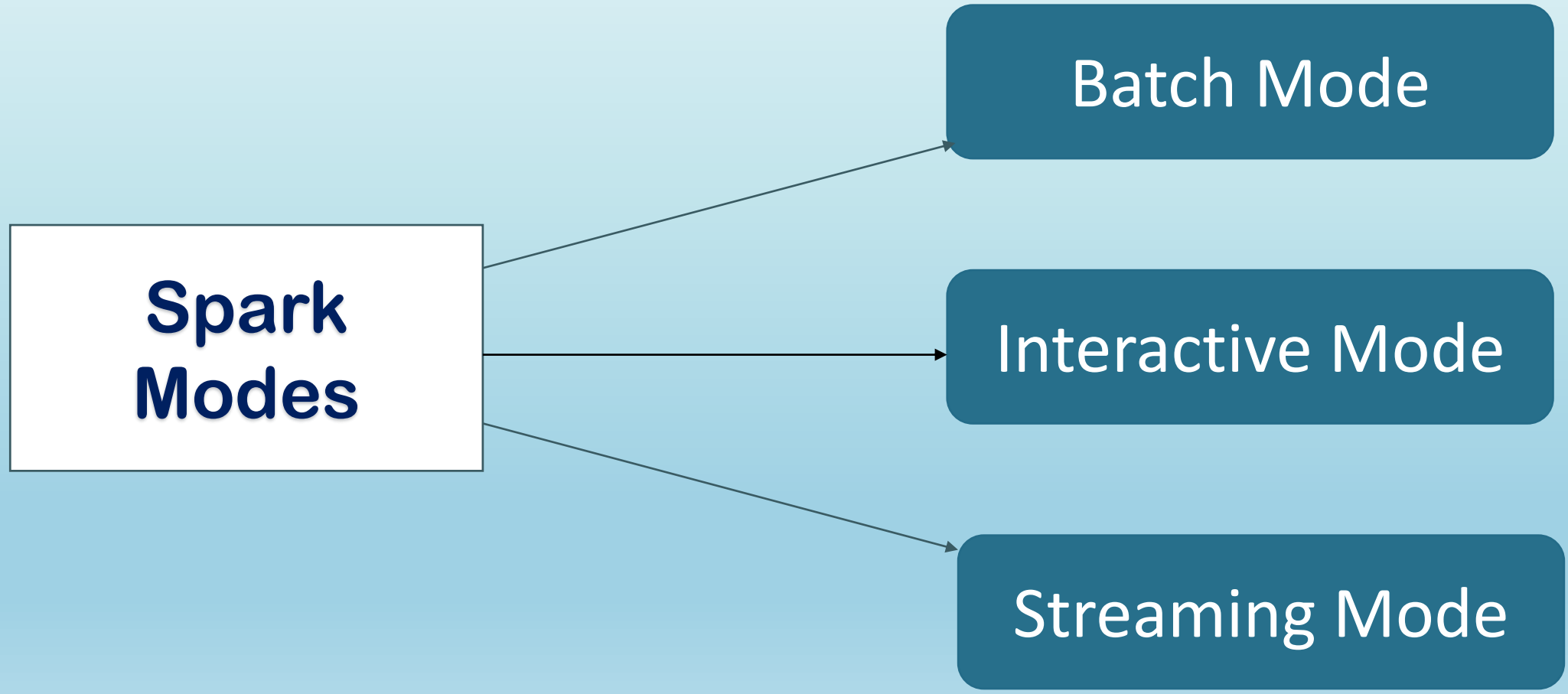


Source: https://spark.apache.org/


BIG
DATA
& ANALYTICS

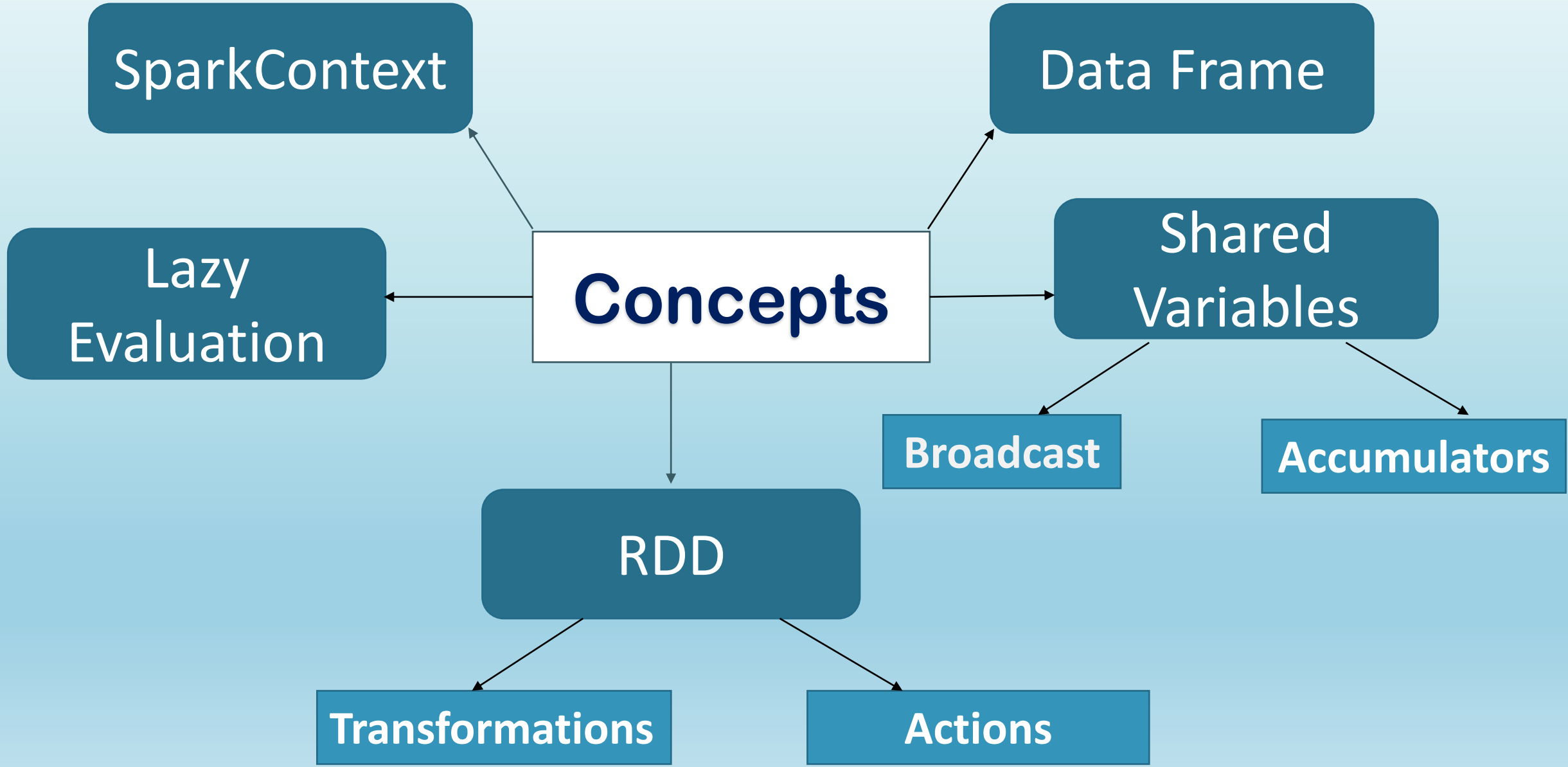
Spark Modes


APACHE
Spark

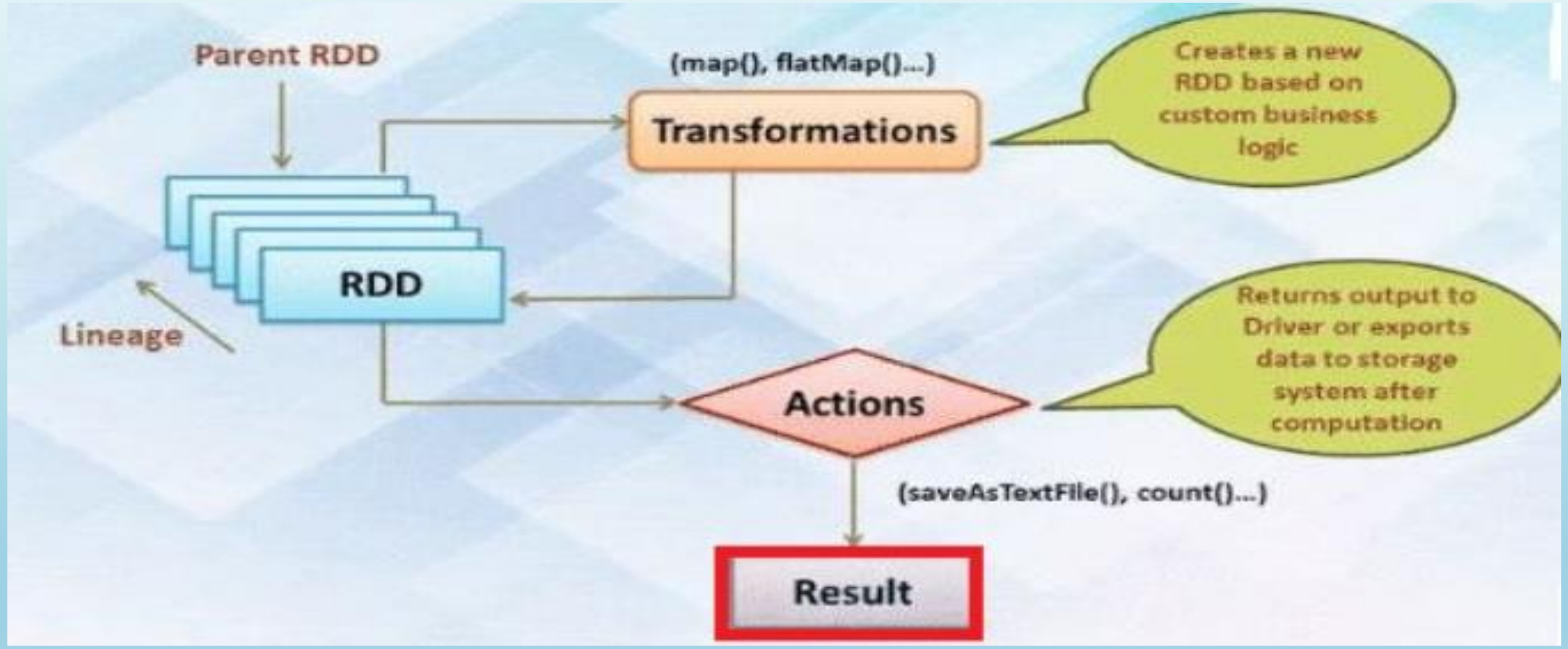

Spark Modes
Batch Mode
Interactive Mode
Streaming Mode

Spark Modes

Major Concepts in Apache Spark



Major Concepts in Apache Spark



Lazy evaluation

RDD Apache Spark

- **RDD** stands for "**Resilient Distributed Dataset**"
- It is the fundamental data structure of Apache Spark.
- RDD in Apache Spark is an immutable collection of objects which computes on the different node of the cluster.
  - **Resilient**, i.e. fault-tolerant with the help of RDD lineage graph(DAG) and so able to recompute missing or damaged partitions due to node failures.
  - **Distributed**, since Data resides on multiple nodes.
  - **Dataset** represents records of the data you work with. The user can load the data set externally which can be either JSON file, CSV file, text file or database via JDBC with no specific data structure.

Transformations in Apache Spark

Narrow Transformations

map(*func*)
filter(*func*)
flatMap(*func*)
mapPartitions(*func*)
mapPartitionsWithIndex(*func*)
sample(*withReplacement*, *fraction*, *seed*)

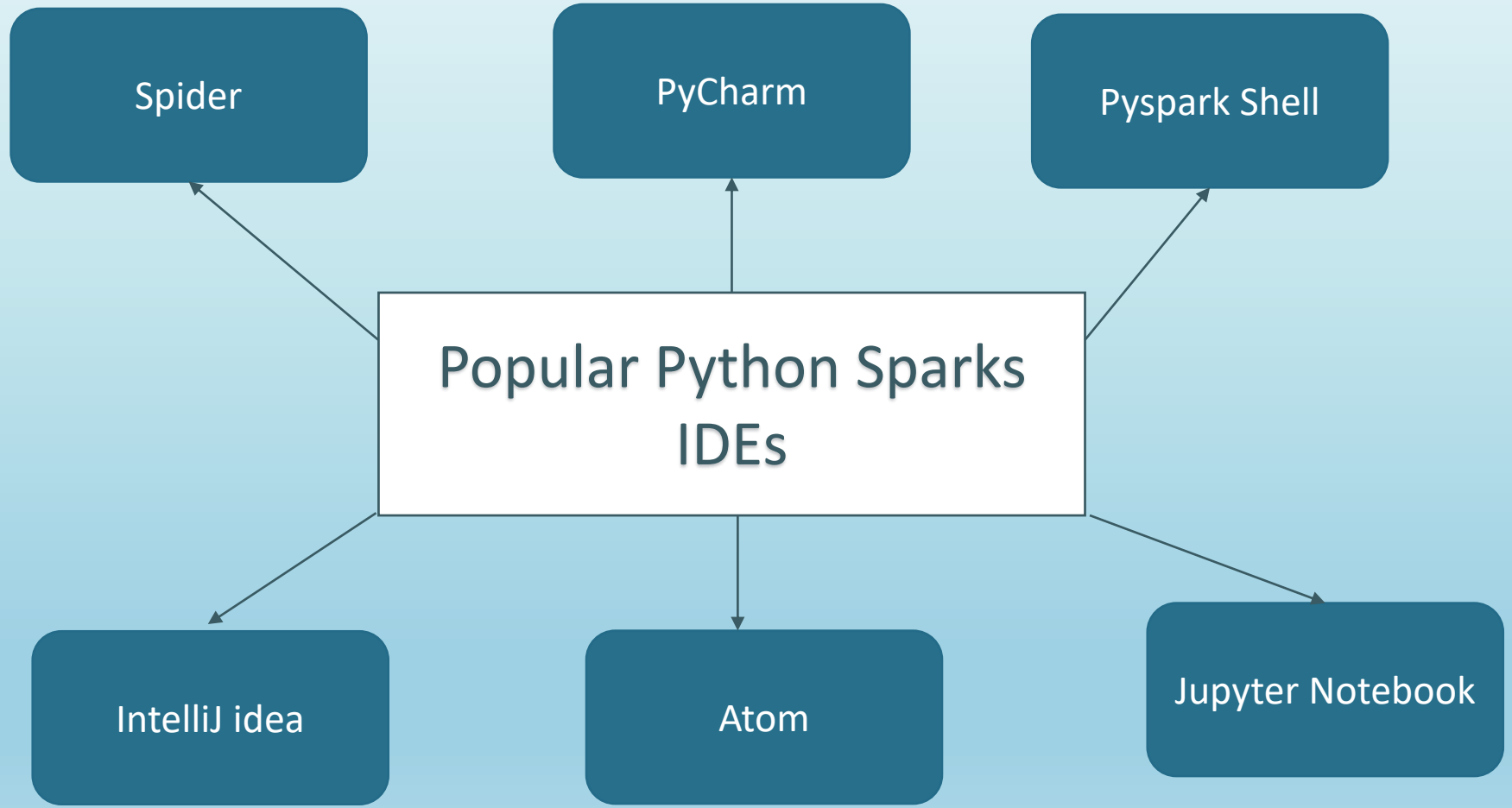
Wide Transformations

union(*otherDataset*)
intersection(*otherDataset*)
distinct([*numPartitions*]))
groupByKey([*numPartitions*])
reduceByKey(*func*, [*numPartitions*])
aggregateByKey(*zeroValue*)(*seqOp*, *combOp*, [*numPartitions*])
sortByKey([*ascending*], [*numPartitions*])
join(*otherDataset*, [*numPartitions*])

Introduction to PySpark

- **Apache Spark** is an open-source cluster-computing framework, built around speed, ease of use, and streaming analytics whereas **Python** is a general-purpose, high-level programming language.
- PySpark is a Python API for Spark that lets you harness the simplicity of Python and the power of Apache Spark in order to handle Big Data problems.
- PySpark provides a wide range of libraries and is majorly used for Machine Learning and Real-Time Streaming Analytics.

IDEs for PySpark



Spark Context

- SparkContext is used as a channel to access all spark functionality.
- The spark driver program uses spark context to connect to the cluster through a resource manager
- sparkConf is required to create the spark context object, which stores configuration parameter like appName (to identify your spark driver), application, number of core and memory size of executor running on worker node.

```
val conf = new SparkConf().setAppName("Retaildata Analysis")
.setMaster("spark://master:7077").set("spark.executor.memory", "2g")
```

```
val sc = new SparkContext(conf)
val hc = new hiveContext(sc)
val ssc = new streamingContext(sc)
```

Spark Session

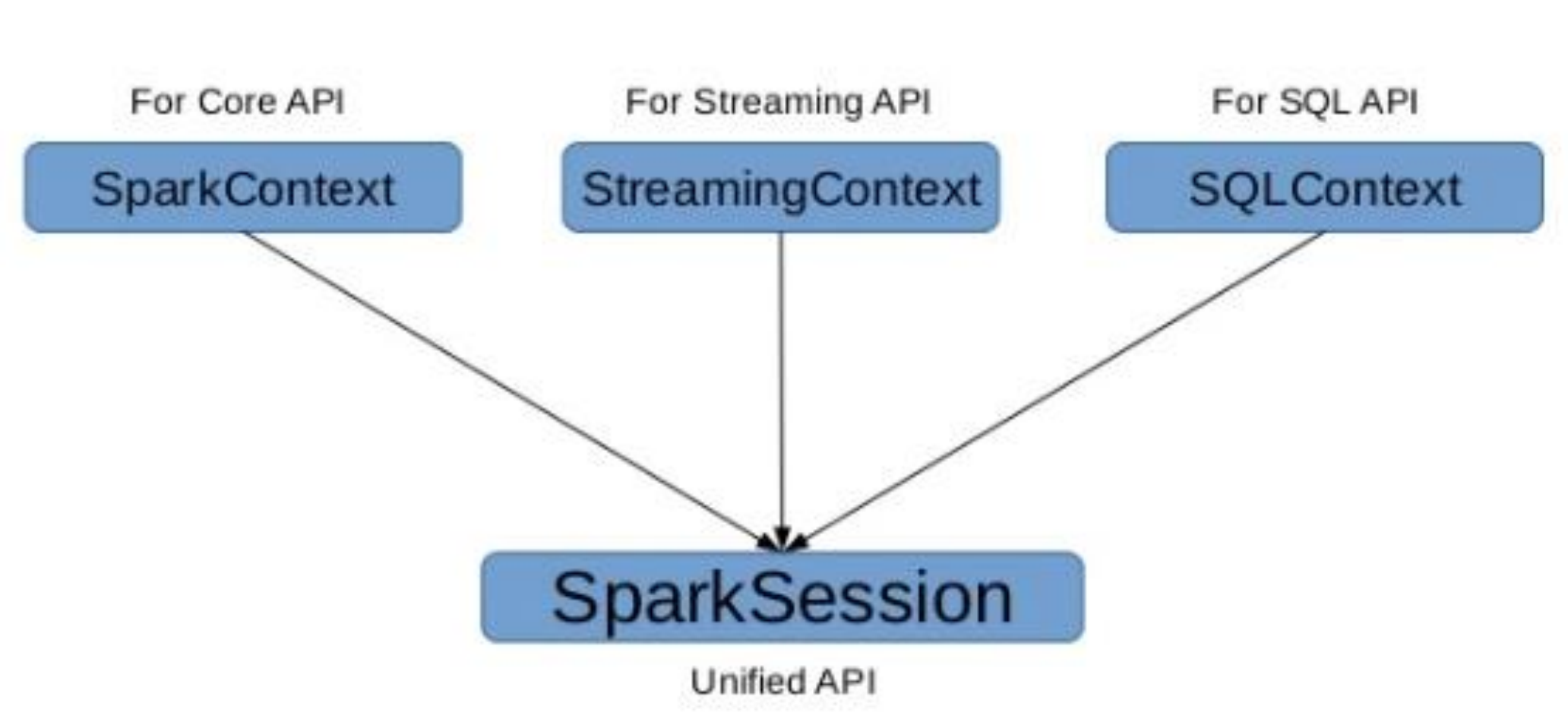
- SparkSession provides a single point of entry to interact with underlying Spark functionality
- It allows programming Spark with DataFrame and Dataset APIs.
- All the functionality available with sparkContext are also available in sparkSession.
- In order to use APIs of SQL, HIVE, and Streaming, no need to create separate contexts as sparkSession includes all the APIs.

```
val spark = SparkSession.builder.appName("WorldBankIndex").getOrCreate()
```

```
spark.conf.set("spark.sql.shuffle.partitions", 6)
```

```
spark.conf.set("spark.executor.memory", "2g")
```

Spark Session



Wordcount Applications


```
import sys
for line in sys.stdin:
     # remove leading and trailing whitespace
     line = line.strip()
     # split the line into words
     words = line.split()
     # increase counters
     for word in words:
          print '%s\t%s' % (word, 1)
```

```
import sys
for line in sys.stdin:
    # remove leading and trailing whitespace
    line = line.strip()
    # split the line into words
    words = line.split()                          aces
    # increase counters
    for word in words:                            py
        print '%s\t%s' % (word, 1)
     # convert count (currently a string) to int
     try:
          count = int(count)
     except ValueError:
          # count was not a number, so silently
          # ignore/discard this line
          continue
     if current_word == word:
          current_count += count
     else:
          if current_word:
               print '%s\t%s' % (current_word, current_count)
          current_count = count
          current_word = word
if current_word == word:
     print '%s\t%s' % (current_word, current_count)
```

Python

```
text_file = sc.textFile("hdfs://...")
counts = text_file.flatMap(lambda line: line.split(" ")) \
                  .map(lambda word: (word, 1)) \
                  .reduceByKey(lambda a, b: a + b)
counts.saveAsTextFile("hdfs://...")
```

PySpark

Installation

- https://spark.apache.org/

References

1) https://spark.apache.org/docs/latest/cluster-overview.html

2) https://intellipaat.com/blog/tutorial/spark-tutorial/spark-architecture/

3) http://commandstech.com/spark-lazy-evaluation-with-example/

4) https://www.edureka.co/blog/pyspark-programming/

5) https://data-flair.training/blogs/apache-spark-lazy-evaluation/

Thank You