

# Apache Spark Architecture

# Topic & Structure of The Lesson

- **Spark Architecture**
  - Spark Core Engine
  - Resilient Distributed data set
  - Programming Languages Supported by Spark

# Learning Outcomes

- **At the end of this topic, You should be able to**
- **Demonstrate the key concepts of the Spark Architecture**
- **Evaluate various programming language used in Spark.**

# Key Terms You Must Be Able To Use

- If you have mastered this topic, **you should be able to use the following terms correctly in your assignments and exams:-**
- Spark Architecture
- Resilient Distributed dataset

# What is Spark?

Apache Spark is a **data processing framework** that can **quickly perform processing tasks** on very large **datasets** and can also **distribute data processing** tasks across multiple computers, either on its own or in together with other distributed computing tools.

# What is Spark?

Fast and Expressive Cluster Computing  
Engine Compatible with Apache Hadoop

Up to **10x** faster on disk,  
**100x** in memory

## Efficient

- General execution graphs
- In-memory storage

**2-5x** less code

## Usable

- Rich APIs in Java, Scala, Python
- Interactive shell

# The Spark Community



<https://spark.apache.org/powered-by.html>

# Spark Architecture

An Apache Spark application consists of two main components:

**a driver**, which converts the user's code into multiple tasks that can be distributed across worker nodes.

**Executors**, which run on those nodes and execute the tasks assigned to them.



# Spark Architecture

Spark can run in a standalone cluster mode that simply requires the **Apache Spark framework and a JVM** on each machine in your cluster.

However, it's more likely you'll want to take **advantage** of a more robust resource or **cluster management system** to take care of allocating workers on demand for you.

# Spark Architecture

Apache Spark can be found as part of

Amazon EMR,

Google Cloud Dataproc, and

Microsoft Azure HDInsight.

# Batch or Streaming

# Working with Key-Value Pairs

Spark's “distributed reduce” transformations operate on RDDs of key-value pairs

Python:

```
pair = (a, b)
pair[0] # => a
pair[1] # => b
```

Scala:

```
val pair = (a, b)
pair._1 // => a
pair._2 // => b
```

Java:

```
Tuple2 pair = new Tuple2(a, b);
pair._1 // => a
pair._2 // => b
```

# Spark Used For ?

- Spark is a general-purpose data processing engine, an API-powered toolkit which data scientists and application developers incorporate into their applications to rapidly query, analyse and transform data at scale.
- Stream processing
- Machine learning
- Interactive analytics
- Data integration

# Resilient Distributed Datasets (RDDs)

RDDs provide a restricted form of shared memory:-

Simple common interface:-

- Set of Partitions
  - Preferred locations for each partitions
  - List of parent RDDs
  - Function to compute a partitions given parents
  - Optional partitioning info
- 
- Allows capturing wide range of transformations

# Spark DStream

- Spark DStream (Discretized Stream) is the basic abstraction of Spark Streaming.
- A DStream is a continuous stream of data.
- It receives input from various sources such as Kafka, Flume, Kinesis, or TCP sockets.
- It can also be a data stream generated by transforming the input stream.
- At its core, a DStream is a continuous stream of RDDs (Spark abstractions). Each RDD in a DStream contains data from a specific interval.

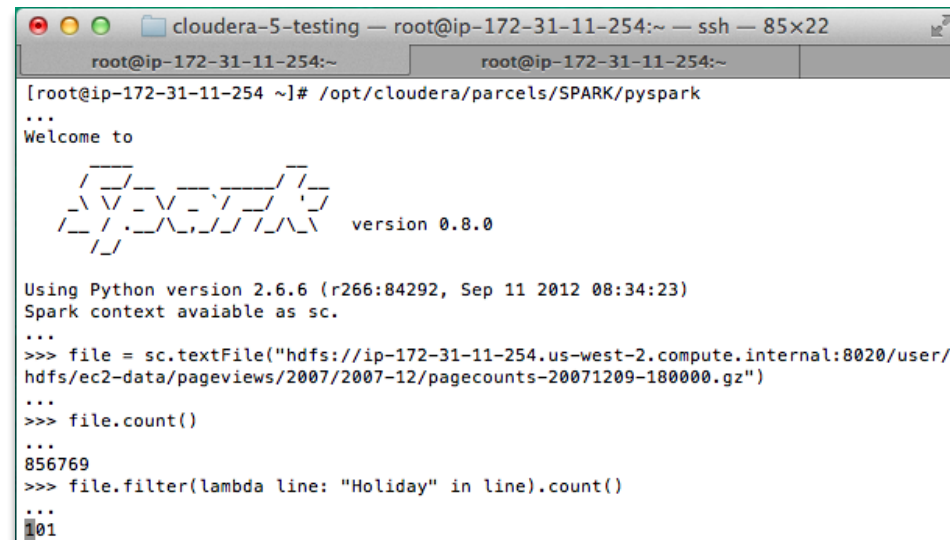
# More RDD Operators

- map
- filter
- groupBy
- sort
- union
- join
- leftOuterJoin
- rightOuterJoin
- reduce
- count
- fold
- reduceByKey
- groupByKey
- cogroup
- cross
- zip
- sample
- take
- first
- partitionBy
- mapWith
- pipe
- save ...



# Interactive Shell

- The Fastest Way to Learn Spark
- Available in Python and Scala
- Runs as an application on an existing Spark Cluster...
- OR Can run locally



```
cloudera-5-testing — root@ip-172-31-11-254:~ — ssh — 85x22
root@ip-172-31-11-254:~ root@ip-172-31-11-254:~
[root@ip-172-31-11-254 ~]# /opt/cloudera/parcels/SPARK/pyspark
...
Welcome to

  ____  __
 / ___/ /  \
/_  _/ /_  \
/___/ \___/

version 0.8.0

Using Python version 2.6.6 (r266:84292, Sep 11 2012 08:34:23)
Spark context available as sc.
...
>>> file = sc.textFile("hdfs://ip-172-31-11-254.us-west-2.compute.internal:8020/user/
hdfs/ec2-data/pageviews/2007/2007-12/pagecounts-20071209-180000.gz")
...
>>> file.count()
...
856769
>>> file.filter(lambda line: "Holiday" in line).count()
...
101
```

# Administrative GUIs

http://<Standalone Master>:8080 (by default)

The image shows two browser windows. The background window is the Spark Master GUI at localhost:8080. The foreground window is the Spark Stages GUI at localhost:4040/stages/. An orange arrow points from the 'app-20131202231712-0000' entry in the Spark Master's 'Running Applications' table to the 'Spark shell' tab in the Spark Stages GUI. Another orange arrow points from the 'Spark' logo in the Spark Stages GUI to the 'Spark Master at spark://mbp-2.local:7077' URL in the background window.

**Spark Master at spark://mbp-2.local:7077**

URL: spark://mbp-2.local:7077  
Workers: 3  
Cores: 24 Total, 24 Used  
Memory: 45.0 GB Total, 1536.0 MB Used  
Applications: Running, 0 Completed

**Workers**

Id
worker-20131202231645-192.168.1.106-56789
worker-20131202231657-192.168.1.106-56801
worker-20131202231705-192.168.1.106-56806

**Running Applications**

ID	Name
app-20131202231712-0000	Spark shell

**Spark Stages**

Total Duration: 3.8 m  
Scheduling Mode: FIFO  
Active Stages: 0  
Completed Stages: 2  
Failed Stages: 0

**Active Stages (0)**

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Shuffle Read
----------	-------------	-----------	----------	------------------------	--------------

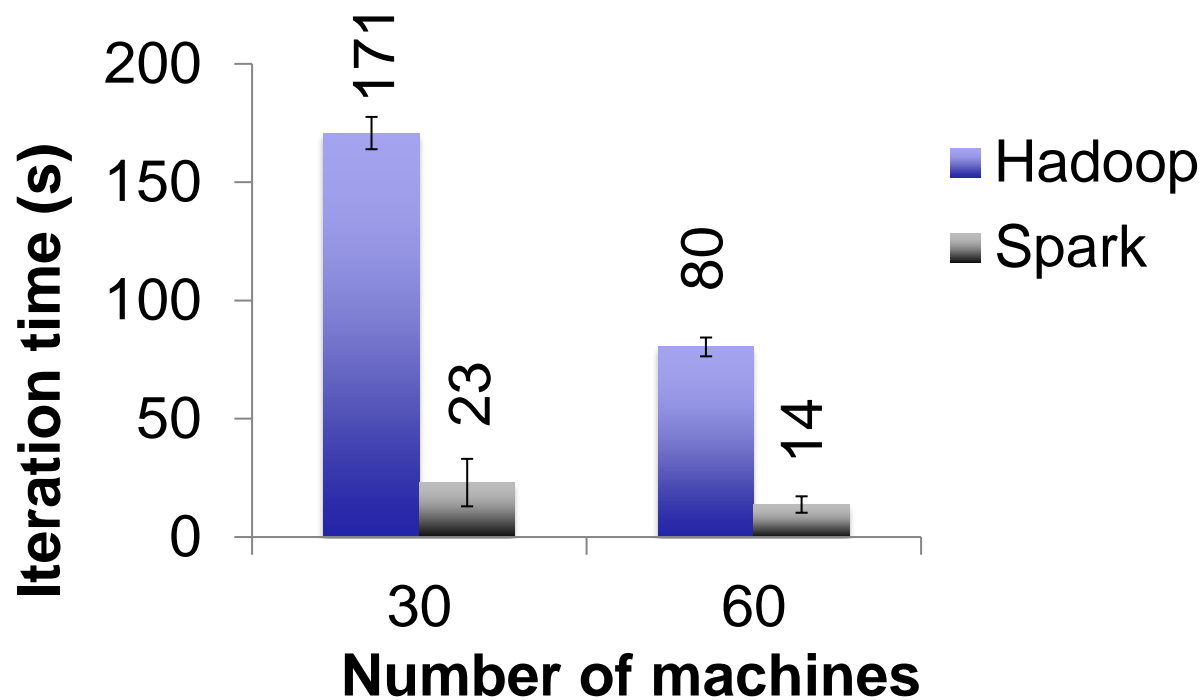
**Completed Stages (2)**

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Shuffle Read
0	count at <console>:13	2013/12/02 21:07:55	83 ms	2/2	754.0 B
1	reduceByKey at <console>:13	2013/12/02 21:07:55	345 ms	2/2	

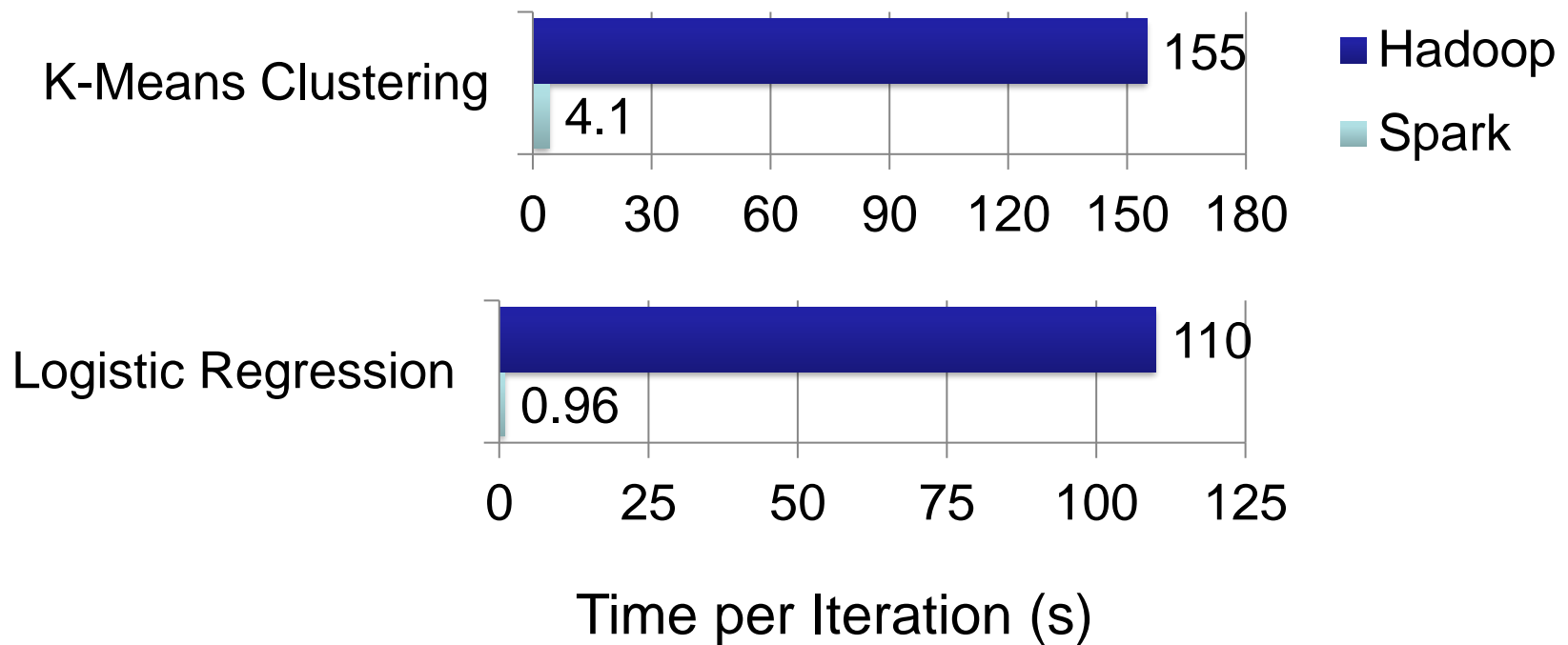
**Failed Stages (0)**

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Shuffle Read
----------	-------------	-----------	----------	------------------------	--------------

# PageRank Performance

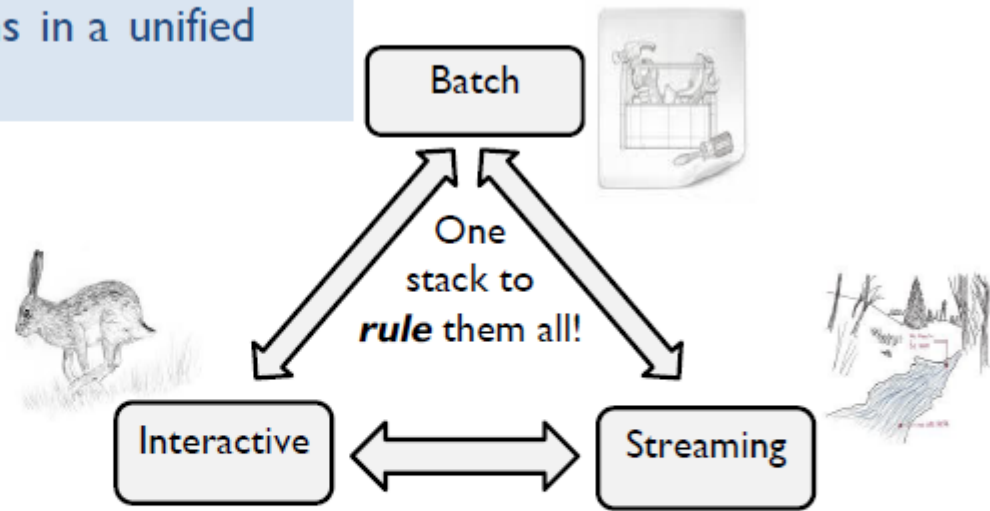


# Other Iterative Algorithms



# Goals

Support **batch**, **streaming**, and **interactive** computations in a unified framework



- **Easy** to combine **batch**, **streaming**, and **interactive** computations
- **Easy** to develop **sophisticated** algorithms
- **Compatible** with existing open source ecosystem (Hadoop/HDFS)

# Spark Core Engine

- Spark Core is the underlying general execution engine for spark platform that all other functionality is built upon.
- It provides **In-Memory computing** and referencing datasets in external storage systems.

# What does Spark Engine do?

- Scheduling
- Distributing data across a cluster
- Monitoring data across a cluster

# Apache Spark supports

- Batch processing
- Stream processing
- Graph processing



# In-Memory Computing

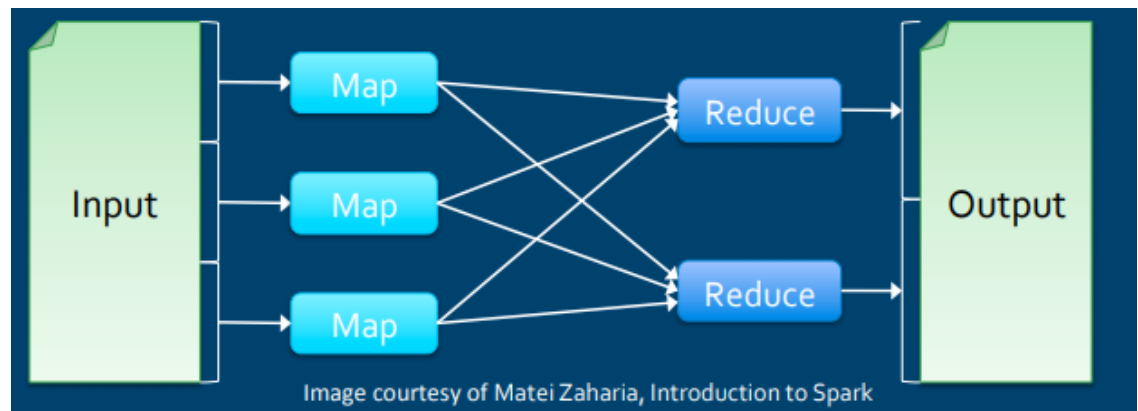
- In-Memory Computing - Key Points
- **Definition:** Uses middleware to store data in RAM across a computer cluster and process it in parallel.
- **Benefit:** Operational datasets, usually in centralized databases, can be stored in “connected” RAM across multiple machines.
- **Speed:** RAM is about 5,000 times faster than traditional spinning disks.

# In-Memory Computing: What Are The Best Use Cases?

- Investment banking
- Insurance claim processing & modelling
- Real-time ad platforms
- Real-time sentiment analysis
- Merchant platform for online games
- Hyper-local advertising
- Geospatial/GIS processing
- Medical imaging processing
- Natural language processing & cognitive computing
- Real-time machine learning
- Complex event processing of streaming sensor data

# Disk Based vs Memory Based Frameworks

- Disk Based Frameworks
  - Persists intermediate results to disk
  - Data is reloaded from disk with every query
  - Easy failure recovery
  - Best for ETL like work-loads
  - Examples: Hadoop, Dryad



# Disk Based vs Memory Based Frameworks

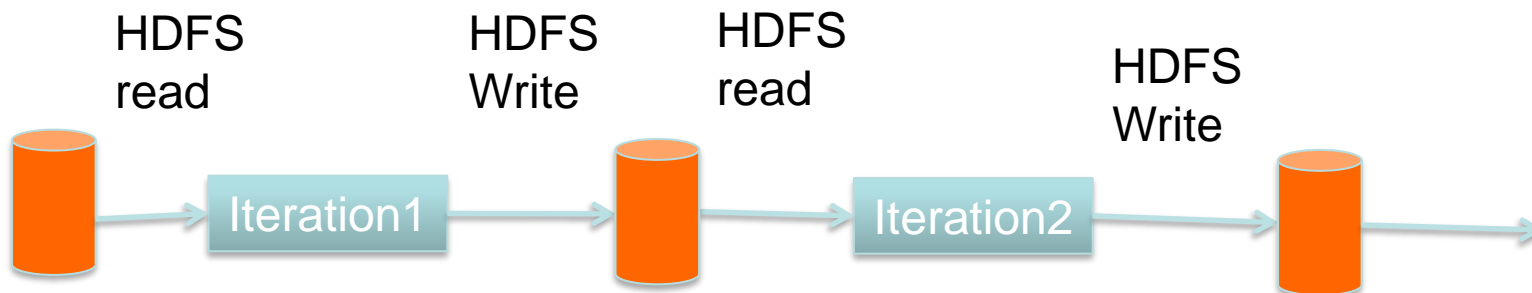
## Memory Based Frameworks

- Circumvents heavy cost of I/O by keeping intermediate results in memory
- Sensitive to availability of memory –Remembers operations applied to dataset
- Best for iterative workloads
- Examples: Spark, Flink

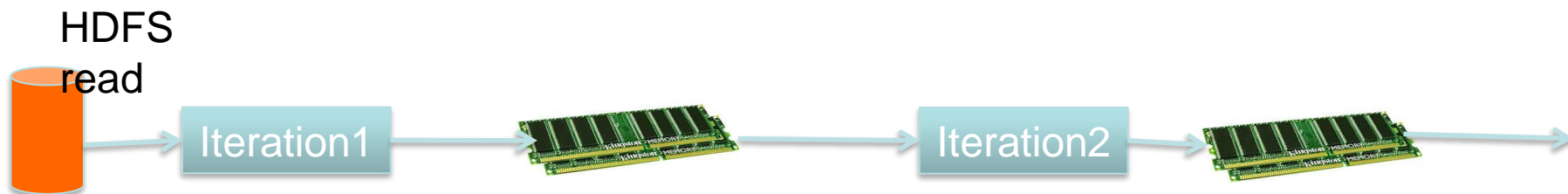


# Spark Uses Memory instead of Disk

Hadoop: Use Disk for Data Sharing



Spark: In-Memory Data Sharing



# SSD VS HDD VS M.2 VS NVMe



Source: <https://gear-up.me/news/ssd-hdd-m2-nvme>

# Spark Core Engine

Spark SQL

Spark  
Streaming

MLib  
(machine  
learning)

GraphX  
(graph)

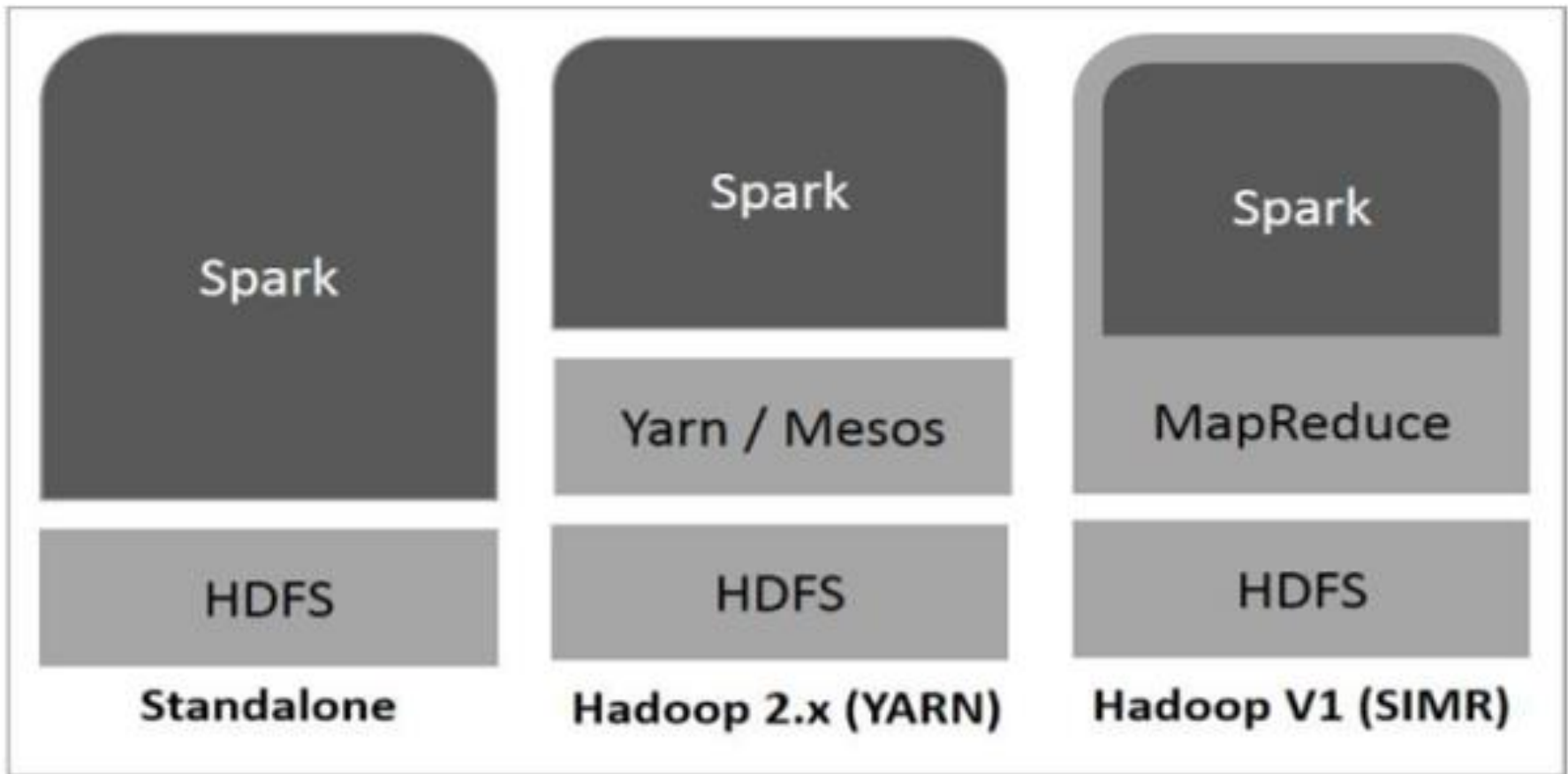
**Apache Spark Core**

# The popular algorithms and utilities in Spark MLlib are:

- Basic Statistics.
- Regression.
- Classification.
- Recommendation System.
- Clustering.
- Dimensionality Reduction.
- Feature Extraction.
- Optimization.
- Streaming Kmeans
- Streaming Linear Regression



# Spark Built on Hadoop



# In Spark Streaming the data can be from what all sources?

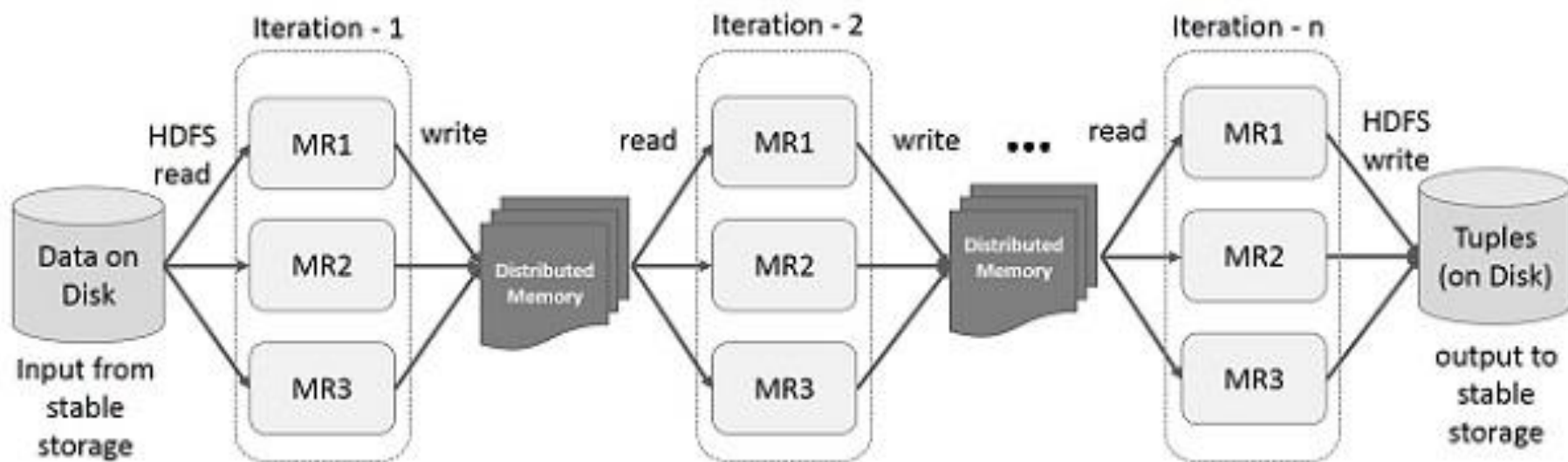
- Kafka
- Flume
- Kinesis

# Resilient Distributed Datasets (RDDs)

- RDDs (Resilient Distributed Datasets) is Data Containers
- All the different processing components in Spark share the same abstraction called RDD
- As applications share the RDD abstraction, you can mix different kind of transformations to create new RDDs
- Created by parallelizing a collection or reading a file
- Fault tolerant

# Iterative Operations on Spark

## RDD



[https://www.tutorialspoint.com/spark\\_sql/spark\\_rdd.htm](https://www.tutorialspoint.com/spark_sql/spark_rdd.htm)

# Life cycle of a Spark Application

1. The user submits a spark application using the spark-submit command.
2. The "driver program" is launched (this is your program)
3. The driver waits the launching of the Application Master task to YARN or Mesos or Spark resource manager (it is a matter of config params at spark-submit, what manager it uses)
4. The cluster manager launches executor JVMs on worker nodes.
5. Application Master allocates tasks in the cluster, according to locality, resource requirements, and nodes available ram and cpu
6. The "slave" tasks (workers) receive input splits to process
7. The slave tasks write result files
8. The job finishes and control returns to the driver program which can continue with new local processing or new cluster processing
9. Tasks are run on executor processes to compute and save results.
10. If the driver's main method exits or it calls `SparkContext.stop()`, it will terminate the executors and release resources from the cluster manager.

# Programming languages supported by Spark

- Java.
- Python.
- Scala.
- SQL.
- R.

# Shortcoming of MapReduce

- **Inefficient for an important class of emerging applications:**
  - Iterative algorithms those that reuse intermediate results across multiple computations

e.g. Machine Learning and graph algorithms

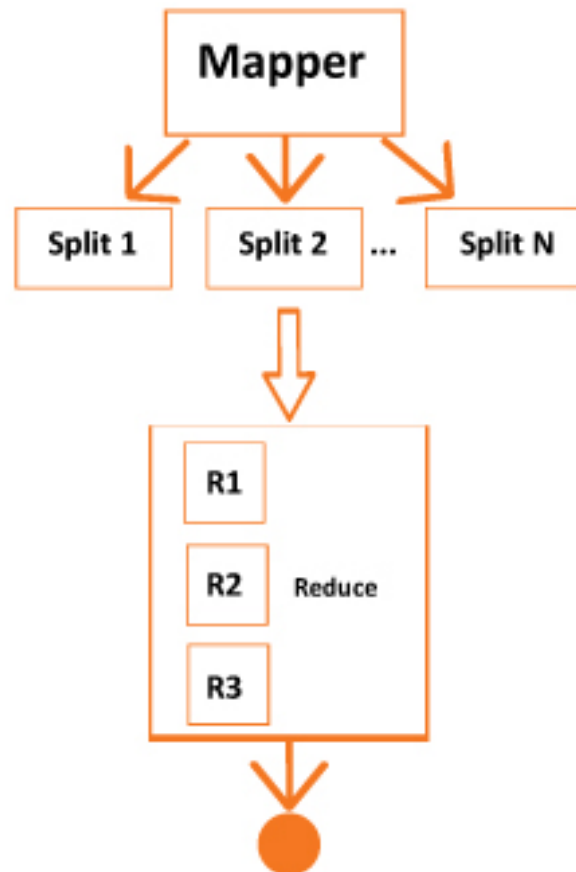
- interactive data mining
- Where a user runs multiple ad-hoc queries on the same subset of the data
- Spark handles current computing frameworks' inefficiency (iterative algorithms and interactive data mining tools)

## **How?**

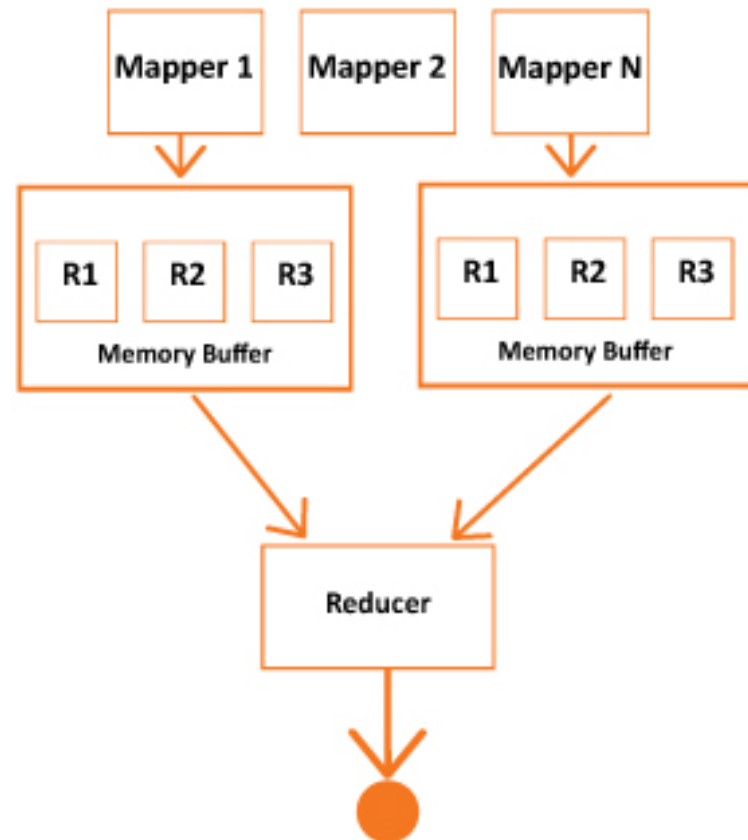
- Keeping data in memory can improve performance by an order of magnitude

# Hadoop Ecosystem

## Hadoop



## Spark



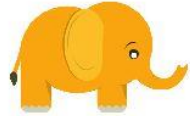
Source: <https://svitla.com/blog/spark-vs-hadoop>



<b>MapReduce</b>	<b>SPARK</b>
Mainly Restricted For Java Developers	Java , Scala, python, R, SQL closure
Boiler Plate Coding	Conciseness
No Interactive Shell	REPL(Read evaluate print loop)
Disk Based, Performance is Slow	Memory based only
Only For Batch Processing	Batch as well as interactive processing
Not Optimized For Iterative Algorithm	Best for iterative algorithms, No Graph
No Graph Processing	Graph processing is supported

Source: <https://k21academy.com/big-data-hadoop-dev/spark-vs-mapreduce/>

# Comparing Hadoop MapReduce and Spark



**Hadoop  
MapReduce**

**Fast**

**Batch Processing**

**Stores Data on Disk**

**Written in java**

**Low Cost**



**Apache  
Spark**

**100x faster than MapReduce**

**Real-time Processing**

**Stores Data Memory**

**Written in Scala**

**High Cost**

# Summary of Main Teaching Points

- Apache Spark is a powerful open-source processing engine built around speed, ease of use, and sophisticated analytics.

# Question and Answer Session

Q & A

# Resource

- **Spark Architecture and Application Lifecycle**
  - <https://medium.datadriveninvestor.com/spark-architecture-and-application-lifecycle-77e347badcbe>
- **Apache Spark: Architecture and Application Lifecycle**
  - <https://www.systemsltd.com/blogs/apache-spark-architecture-and-application-lifecycle>