CARLSON SCHOOL
OF MANAGEMENT
UNIVERSITY OF MINNESOTA

Introduction to Apache Spark

MSBA 6330 Prof Liu

Outline

- What is Spark?
- Why Apache Spark?
- Spark Programming Model
- Spark Architecture and Integration with Hadoop
- Spark Use Cases

Introduction to Apache Spark

WHAT IS SPARK?

What is Spark?

- Apache Spark is a cluster computing platform on top of storage layer
  - a fast, general-purpose engine for large-scale data processing and analysis developed at UC Berkeley in 2009
  - Open sourced in 2010
- Extends MapReduce with support for more components
  - Multi-pass analytics (e.g. ML, graph)
  - Real-time streaming processing
  - Interactive ad-hoc analysis
- Runs in memory
  - Spark offers the ability to run computations in memory

Spark Version History

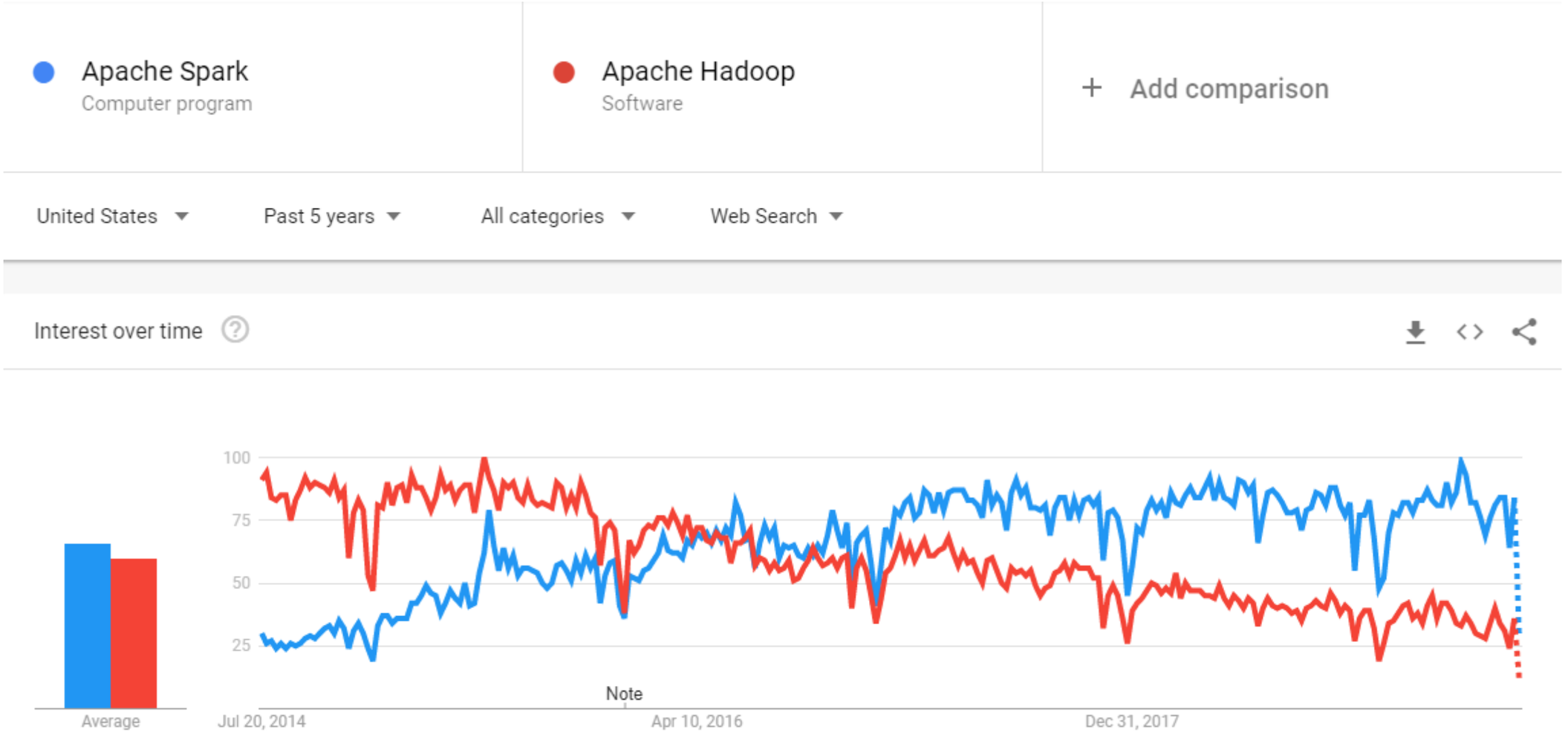
- 1.0 – August 2014
- 1.3 – April 2015
- 1.6 – November 2016 (our VM's version)
  - Dataset is introduced.
  - New Spark Mllib models: survival analysis, hypothesis testing etc
- 2.0 – November 2016
  - Unifying DataFrame and Dataset, SparkSession, performance improvement (2-10x), PySpark gain ML algorithms, Structured Streaming
- 2.2 – July 2017, databricks release notes.
  - Production ready Structured Streaming, Expanded SQL functionalities, new distributed ML algorithms in R, additional algorithms in MLlib and GraphX
- 2.3 – Sep 24, 2018, databricks release notes.
  - Improved Structured Streaming, boosted PySpark with pandas UDFs, support for Kubernetes clusters, new functionalities to SparkR, Python, MLlib, and GraphX.

It has over 1000 contributors in 2015, making it one of the most active project in Apache Software Foundation

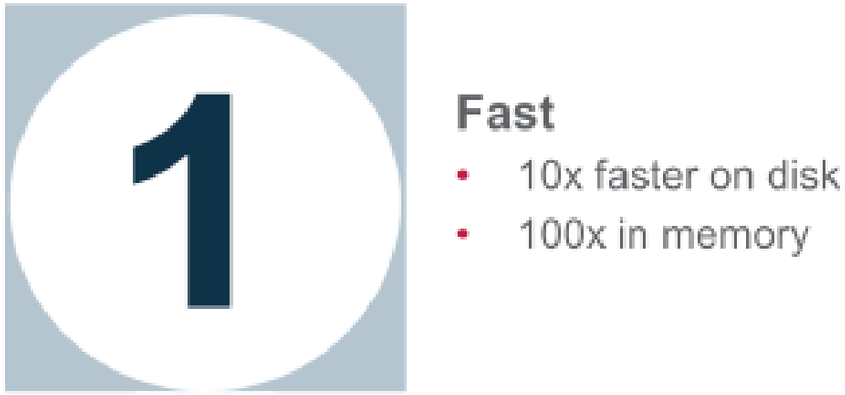
Introduction to Apache Spark

WHY SPARK?

Spark vs Hadoop

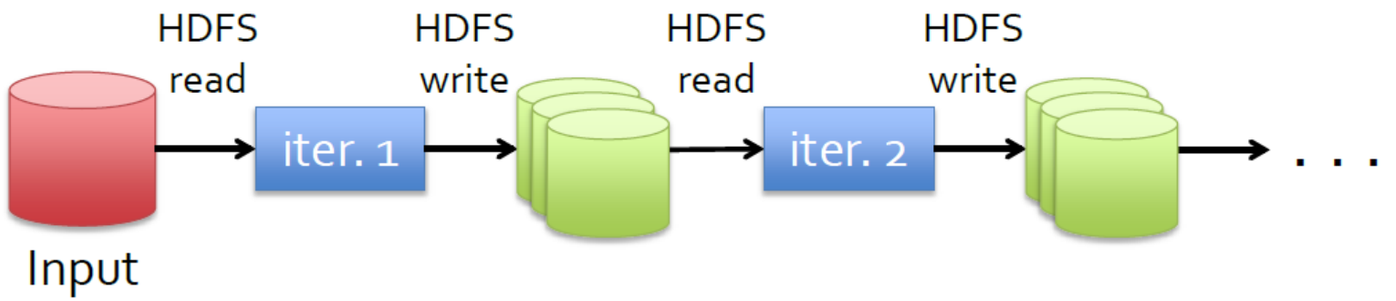


Why Apache Spark?

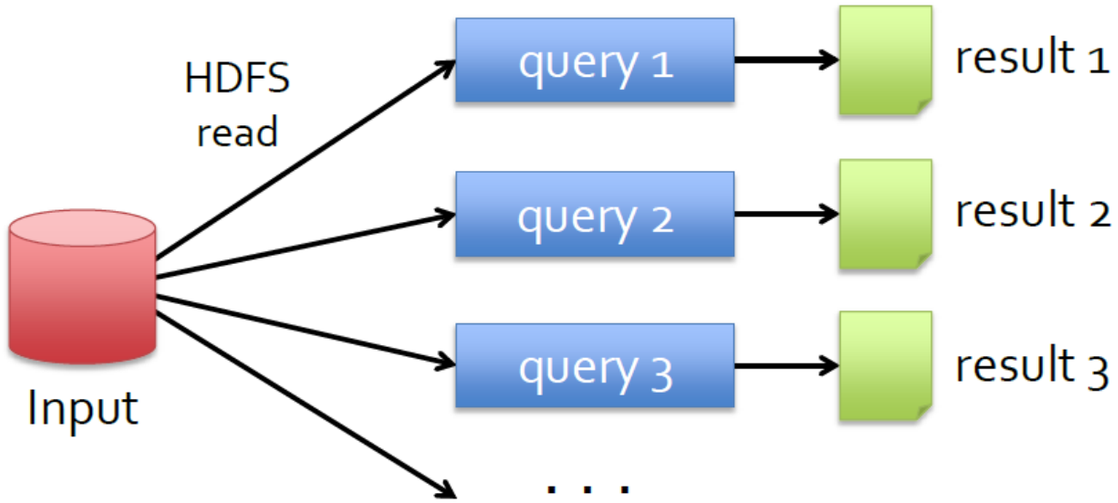


- Spark provides reliable in-memory performance.
  - Iterative algorithms are faster as data is not being written to disk between jobs.
  - In-memory data sharing make it possible for different jobs to work with the same data quickly.
- Spark processes data 10 times faster than MapReduce on disk (through general execution graphs) and 100 times faster in memory.

Data Sharing in MapReduce



Data sharing in sequential steps of multi-pass processing

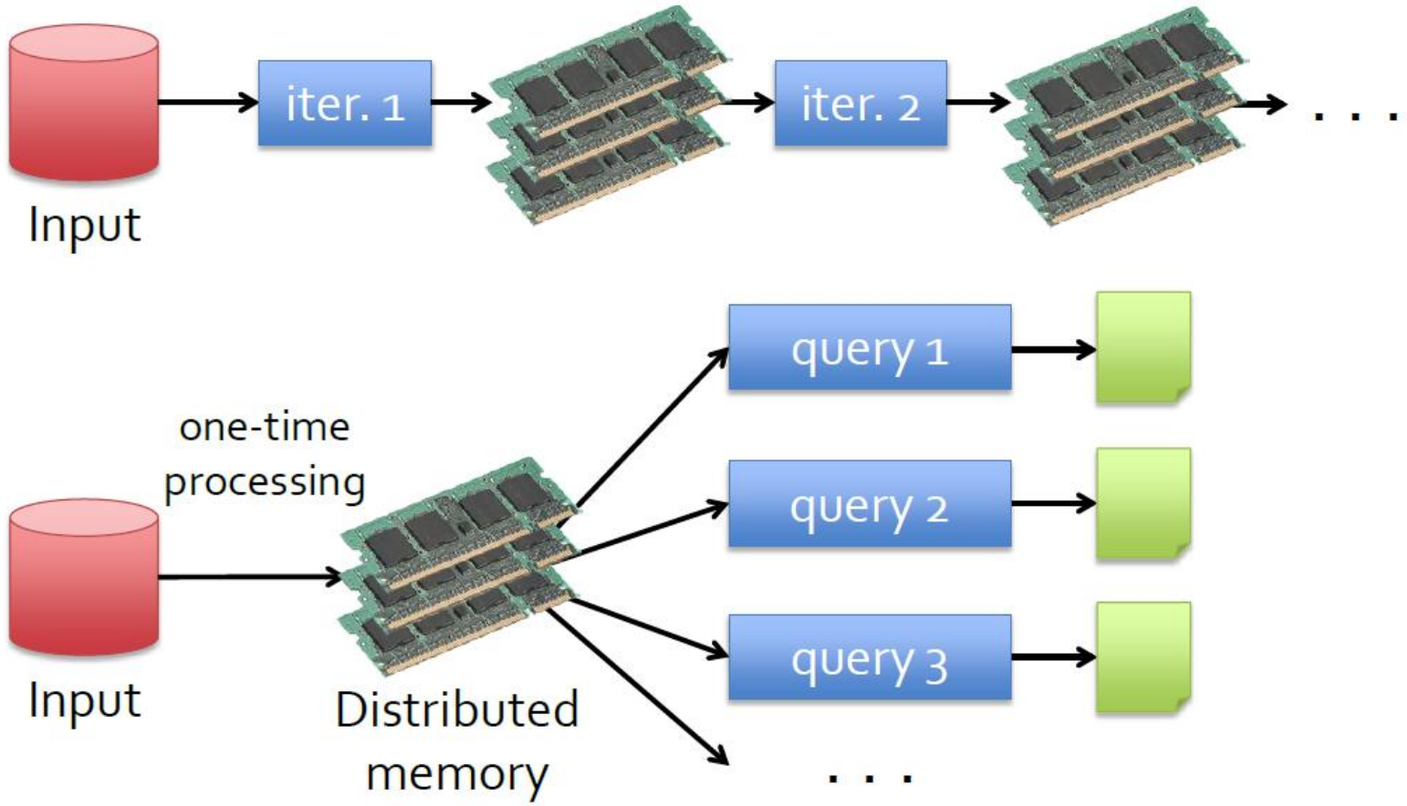


Data sharing across multiple parallel processes

- MapReduce is **slow** due to replication, serialization, and disk IO.

Source: [Matei Zaharia (UC Berkely)'s 2013 Presentation]

Data Sharing on Spark

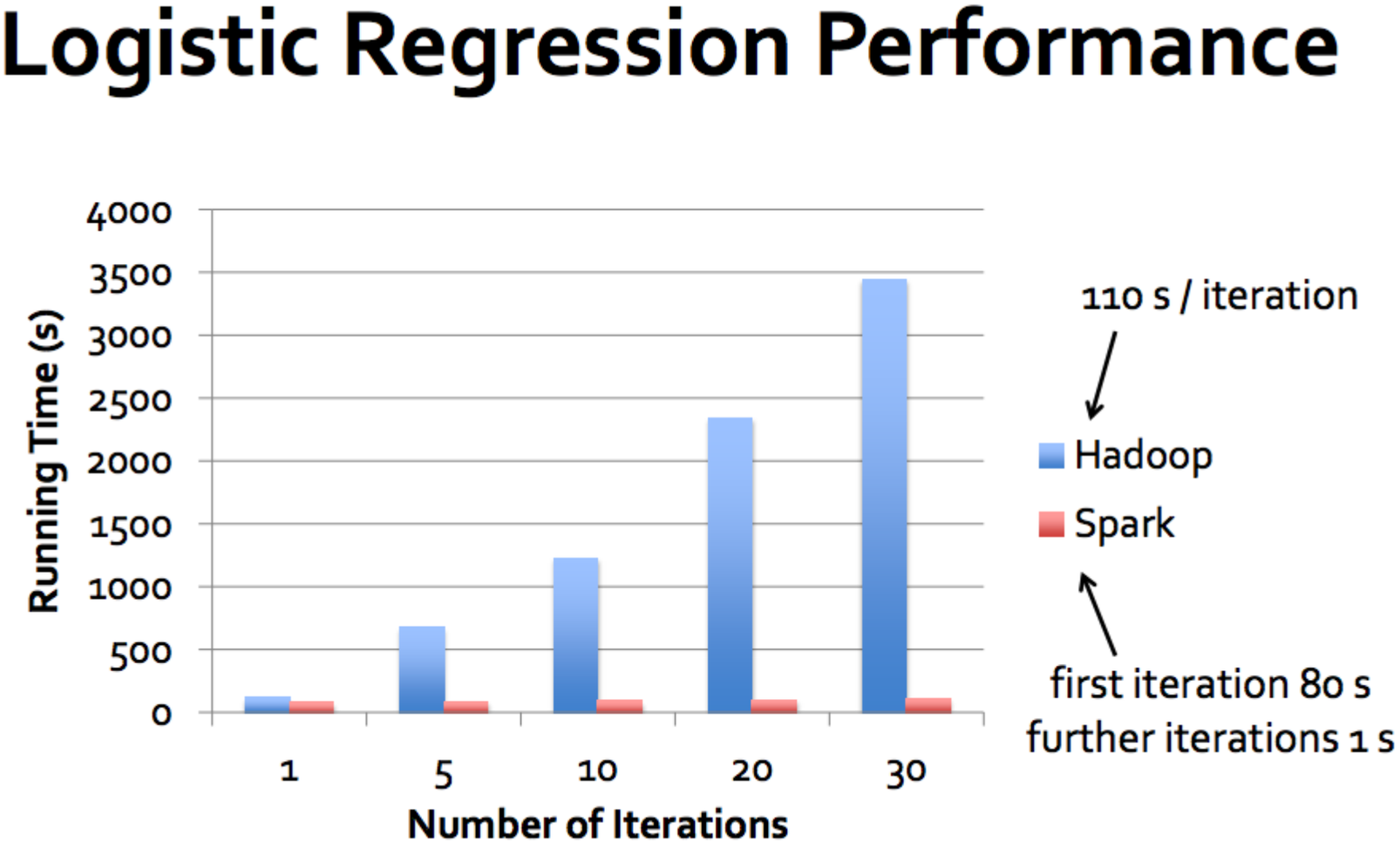


Data sharing in sequential steps of multi-pass processing

Data sharing across multiple parallel processes

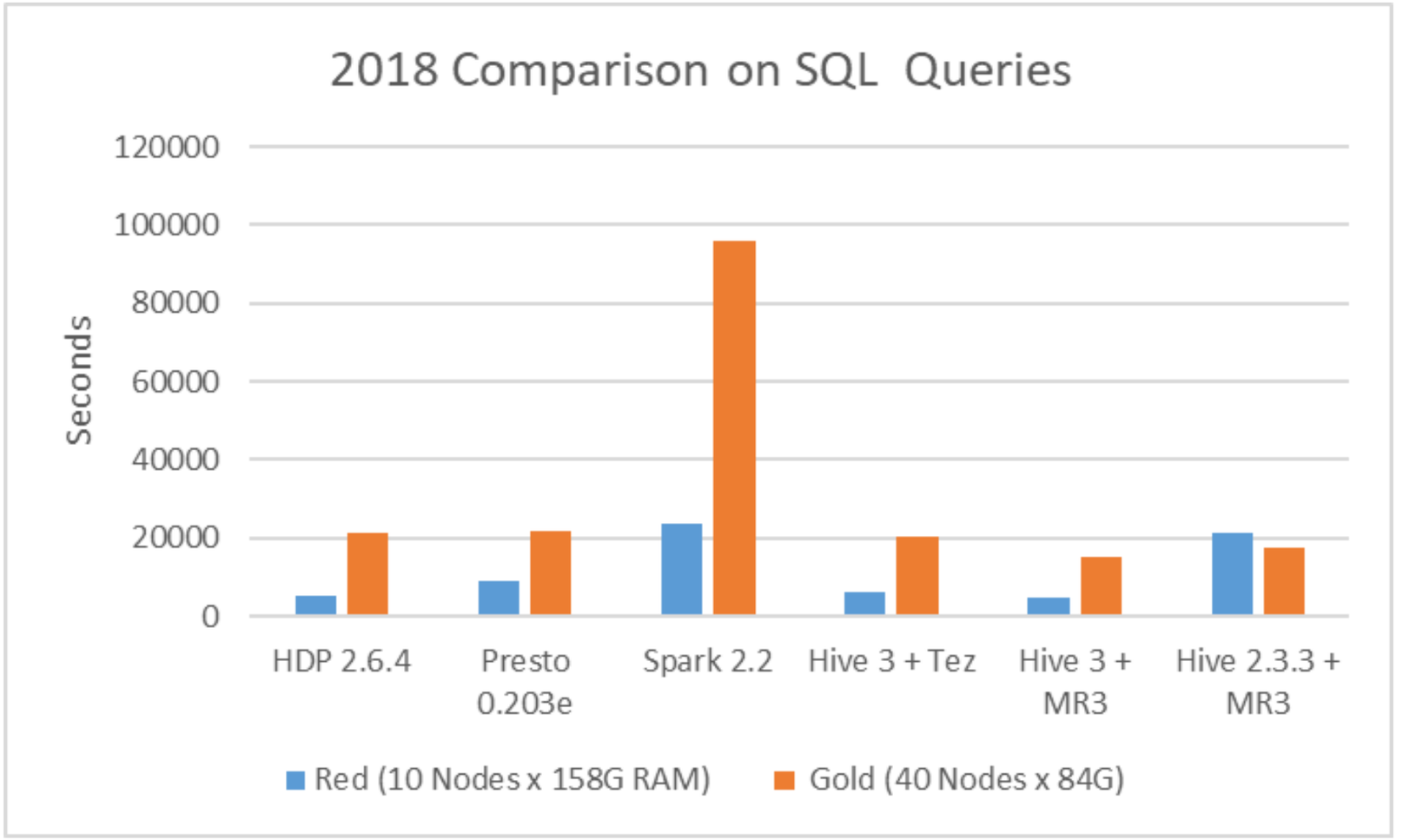
- 10-100x faster than network and disk

Spark vs Hadoop – Logistic Regression (in 2012)



Source: [Matei Zaharia presentation in 2012]

Spark vs Hive – SQL tasks (in 2018)



Source: https://mr3.postech.ac.kr/blog/2018/08/15/comparison-llap-presto-spark-mr3/

Why Apache Spark?

- You can build complex algorithms very quickly in Spark.
- Spark provides support for many more operators than MapReduce such as Joins, reduceByKey, combineByKey.
- Spark also provides the ability to write programs interactively using the Spark Interactive Shell available for Scala and Python.
- You can compose non-trivial algorithms with little code.

Why Apache Spark?

- You can continue to use your existing big data investments.
  - Spark is fully compatible with Hadoop.
  - It can run in YARN, and access data from sources including HDFS, Hbase, HIVE, and S3.
  - In addition, spark can also use the more general resource manager Mesos.

Why Apache Spark?

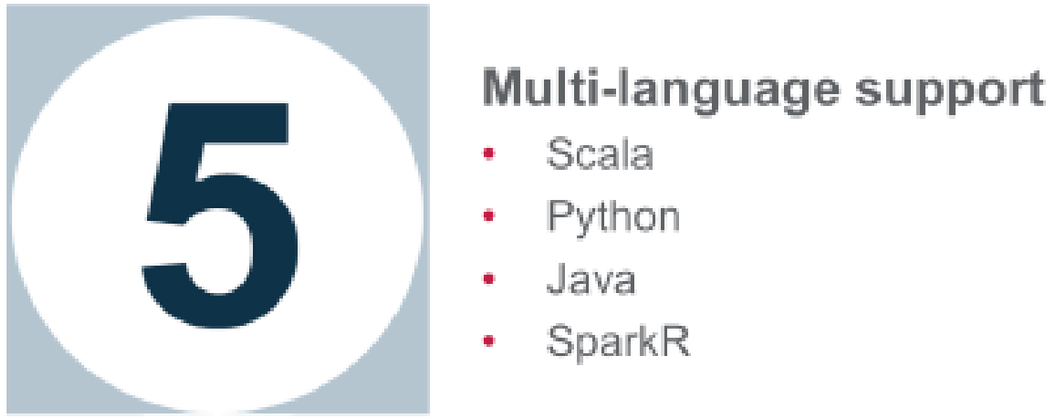
Unified Stack

Builds applications combining different processing models

- Batch
- Streaming
- Interactive Analytics

- Spark has an integrated framework for advanced analytics like Graph processing, advanced queries, stream processing and machine learning.
- You can combine these libraries into the same application and use a single programming language through the entire workflow.

Why Apache Spark?



- Developers have the choice of using
  - Scala (native language for Spark)
  - Python
  - Java
  - SparkR (since 1.4.1+)

Spark vs MapReduce

Spark Retains the advantages of MapReduce:

- Linear scalability
- Fault-tolerance
- Data Locality based computations

…but offers so much more:

- Low Latency (“fast”)
- Ease of development
- Deployment flexibility
- Unified stack
- Multiple Language Support

Spark vs MapReduce

- Drawbacks of Spark
  - Spark is more resource intensive:
    - You need to fit all of your data in memory to take full advantage of Spark
    - MapReduce may be more cost-effective for truly Big Data that does not fit in memory.
  - Less mature than MapReduce (but this is changing)

Introduction to Apache Spark

SPARK PROGRAMMING MODEL

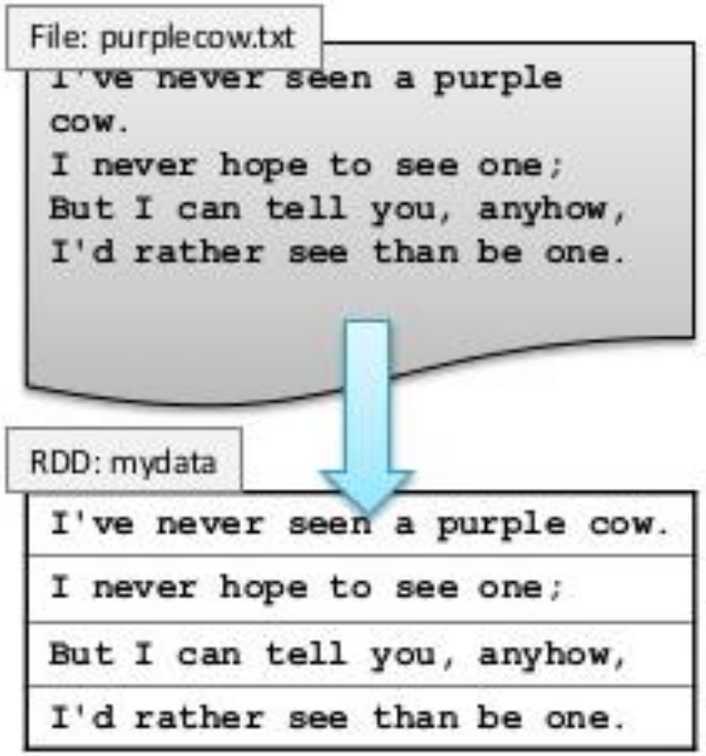
Spark Programming Model

- **Key Idea**: RDD (Resilient Distributed Dataset) are the fundamental unit of data in Spark.
  - They are a collection of objects that is distributed across nodes in a cluster, and data operations are performed on RDD
  - *Resilient* – if data in memory is lost, it can be recreated.
    - If a given node or task fails, the RDD can be reconstructed automatically on the remaining nodes and the job will complete.
  - *Distributed* – stored in **memory** across the cluster
  - Once created, RDDs are **immutable**.

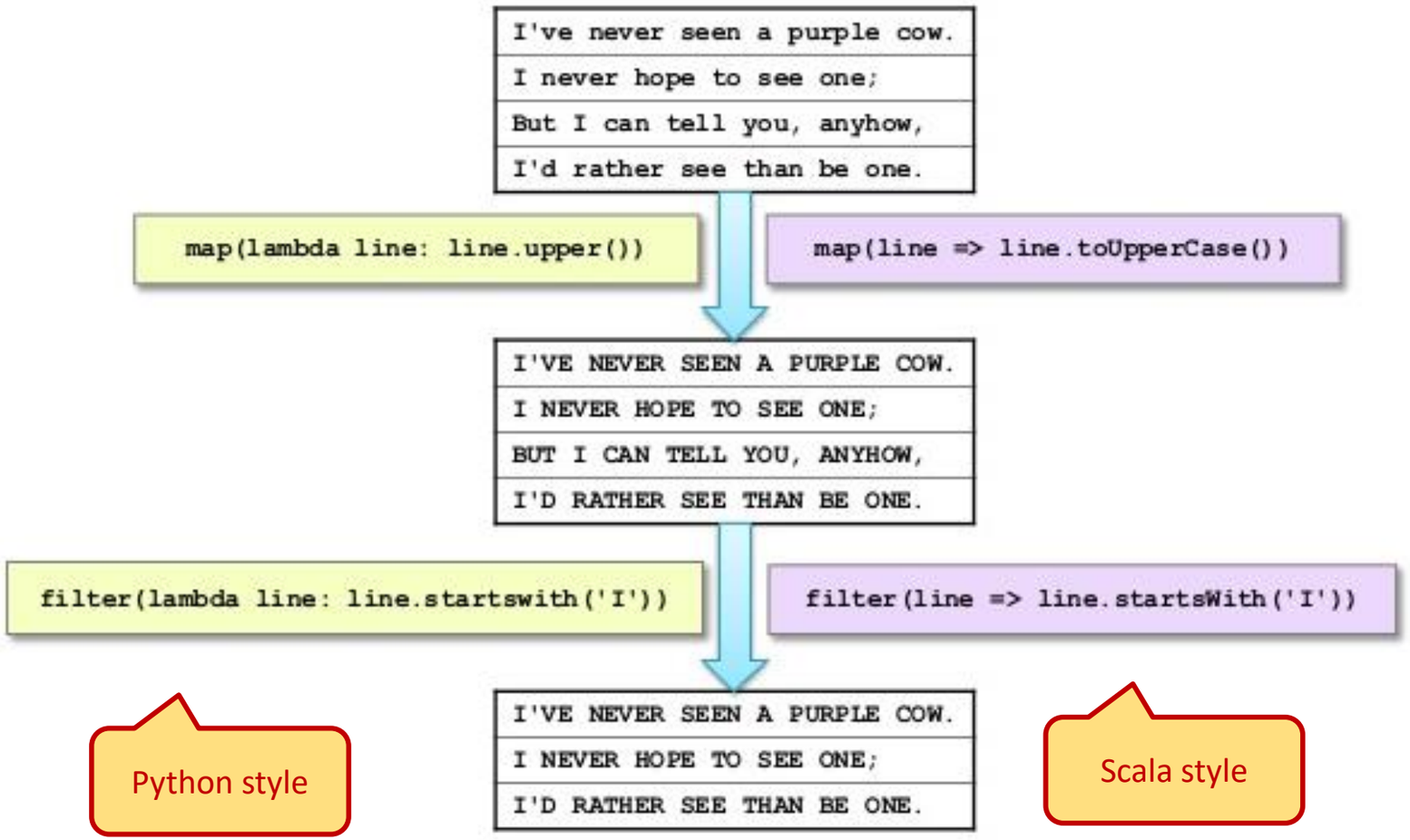
Example: A File-based RDD

- `mydata = sc.textFile("purplecow.txt")`
- `mydata.count()`

- RDDs can hold any type of element
  - Primitive types: integers, characters, strings etc.
  - Sequence types: lists, arrays, tuples, dicts, nested sequences
  - Scala/Java objects (if serializable)
  - Mixed types

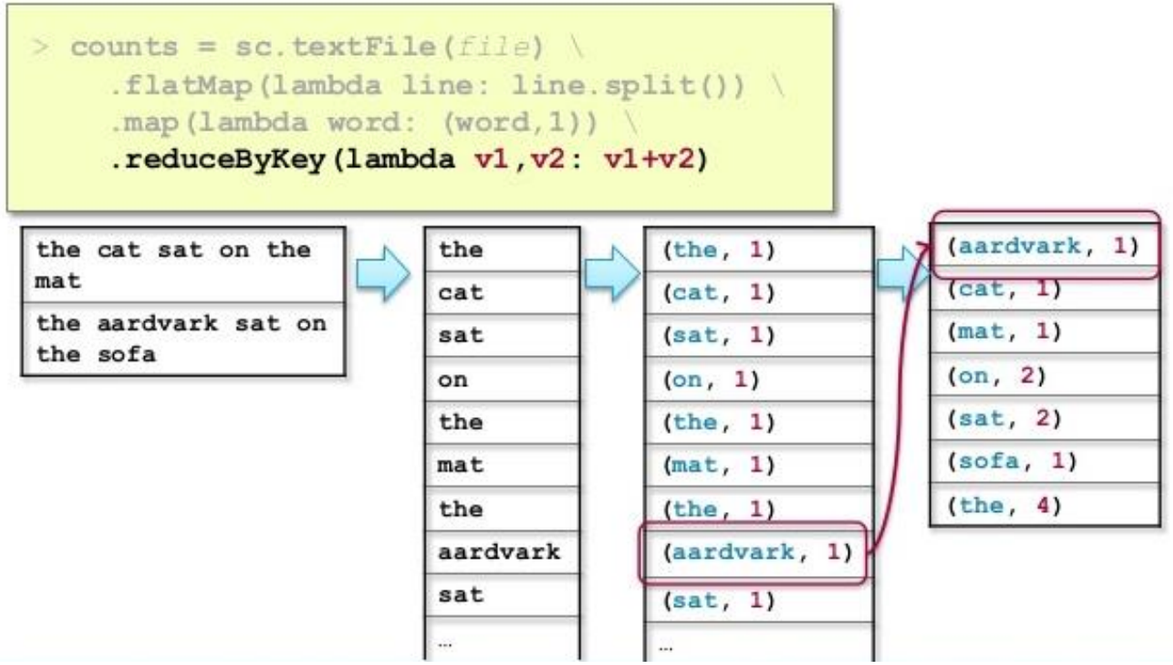


Transformation Operations on RDDs



RDD and MapReduce

- RDDs allow map reduce operations with much greater flexibility
  - Map and Reduce functions can be interspersed
  - Results stored in memory
  - Operations can be chained easily.



Spark Programming Interface

- Programming interface
  - Functional APIs in Scala, Java, Python
  - Interactive Scala & Python shells

Python

```
lines = sc.textFile(...)
lines.filter(lambda s: "ERROR" in s).count()
```

Scala

```
val lines = sc.textFile(...)
lines.filter(s => s.contains("ERROR")).count()
```

Java

```
JavaRDD<String> lines = sc.textFile(...);
lines.filter(new Function<String, Boolean>() {
  Boolean call(String s) {
    return s.contains("error");
  }
}).count();
```

`./bin/spark-shell`

```
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 2.0.0
      /_/

Using Scala version 2.11.8 (Java HotSpot(TM) 64-Bit Server
VM, Java 1.8.0_66)
Type in expressions to have them evaluated.
Type :help for more information.

scala> val dataframe = spark.read.json("example.json")
dataframe: org.apache.spark.sql.DataFrame = [key: string]
```

`./bin/pyspark`

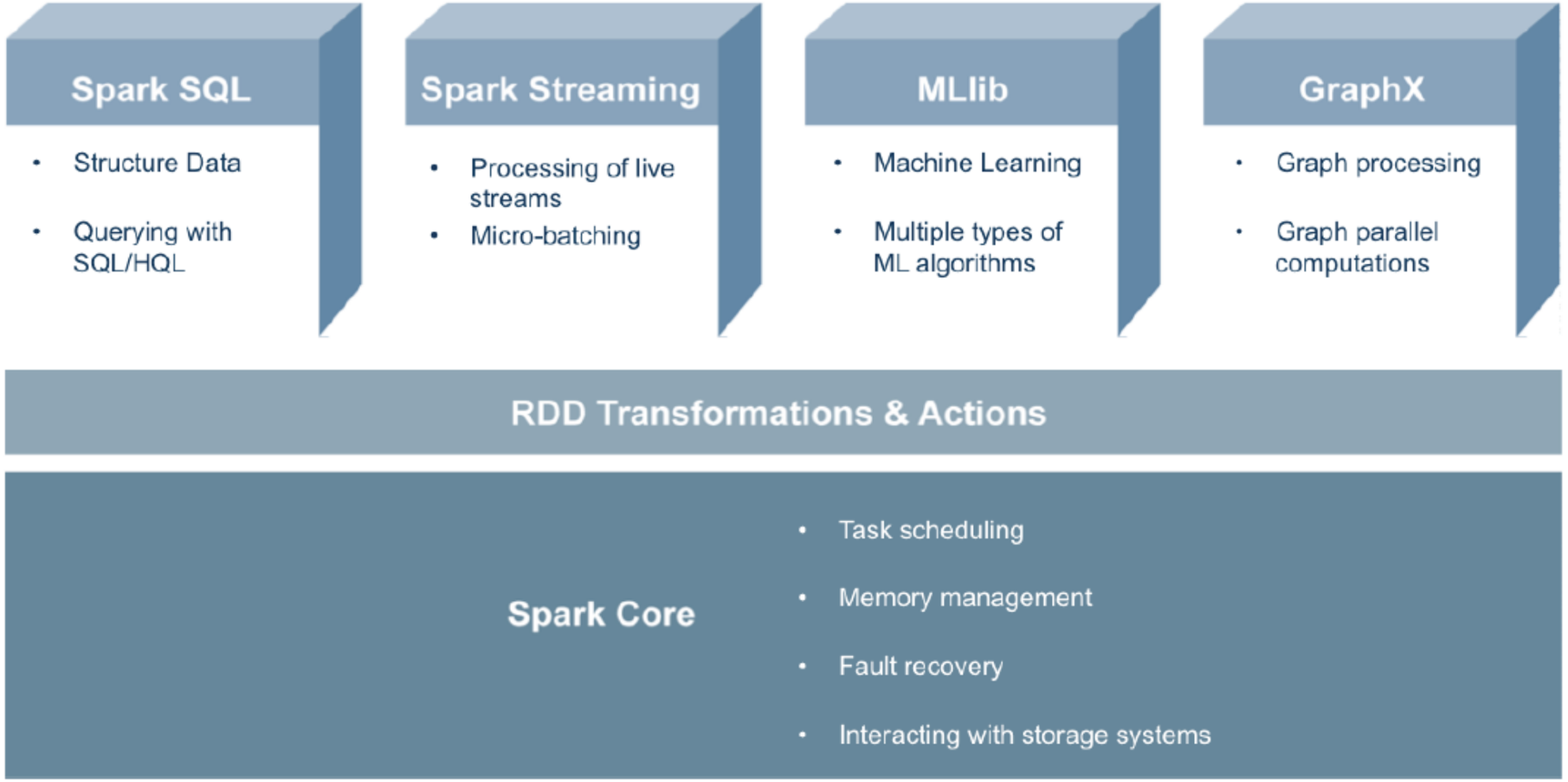
```
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 2.1.0
      /_/

Using Python version 2.7.12 (default, Nov 19 2016 06:48:10)
SparkSession available as 'spark'.
>>> logFile = "file:///home/hadoop/spark-2.1.0-bin-hadoop2.7/README.md"
>>> logData = sc.textFile(logFile).cache()
>>> numAs = logData.filter(lambda s: 'a' in s).count()
>>> numBs = logData.filter(lambda s: 'b' in s).count()
>>> print "Lines with a: %i, lines with b: %i" % (numAs, numBs)
Lines with a: 62, lines with b: 30
>>>
```

Introduction to Apache Spark

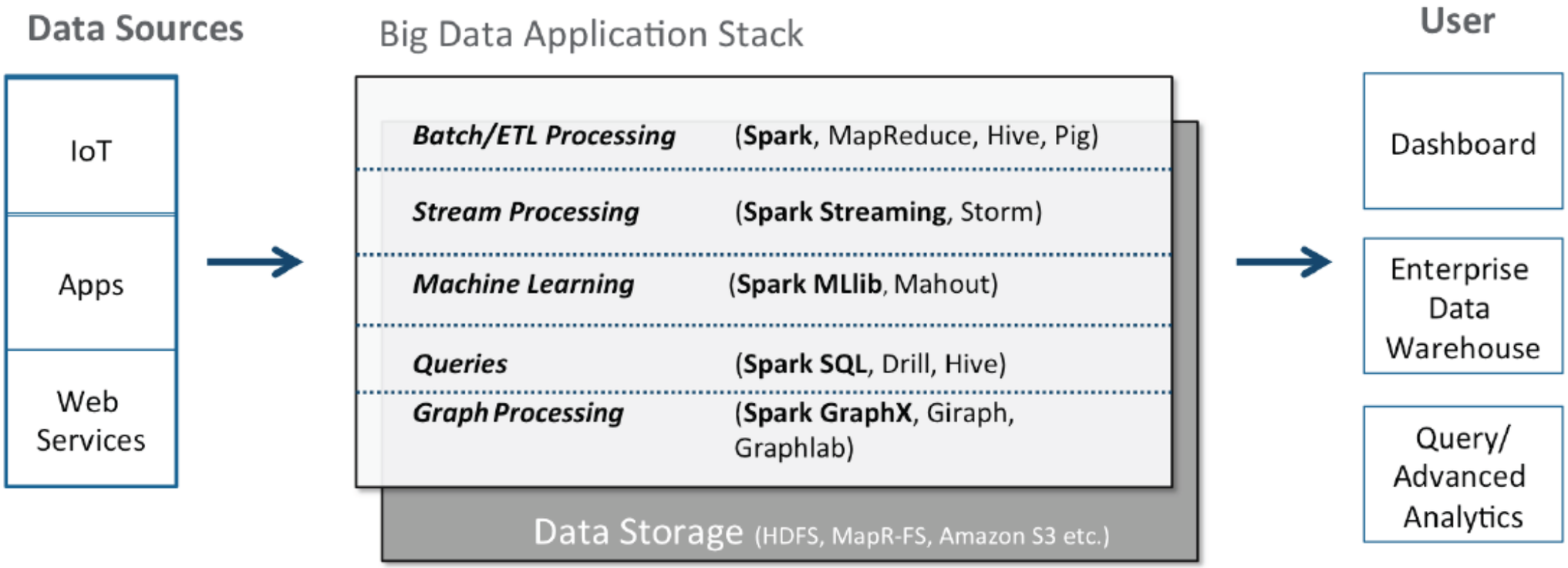
SPARK ARCHITECTURE AND INTEGRATION WITH HADOOP

Spark Architecture



Source: MapR Technologies

Apache Spark & the Hadoop Ecosystem



Source: MapR Technologies

Introduction to Apache Spark

SPARK USE CASES

Spark Use Case

- OLAP Analytics
  - Service provider using Spark to deliver real-time multi-dimensional OLAP analytics
  - Accept data of any time/format
  - Perform rigorous analytics across large datasets

Medtronic: Using Hadoop + Spark + R to achieve 50+ speed up than SQL Server in analyzing billions of clinical observations to predict heart failure

Spark Use Case

- Predictive analytics
  - Health insurance company uses **HDFS** to store patient information & clinical records.
  - Using real-time information stored in **NoSQL**, Spark computes patient re-admittance probability.
  - If this probability is high enough, additional services such as in home nursing, are deployed.

Spark Use Case

- Complex Data Pipelining
  - Pharmaceutical company uses Spark for gene sequencing analysis.
  - Spark reduces processing time from weeks to hours.
  - Complex machine learning without MapReduce