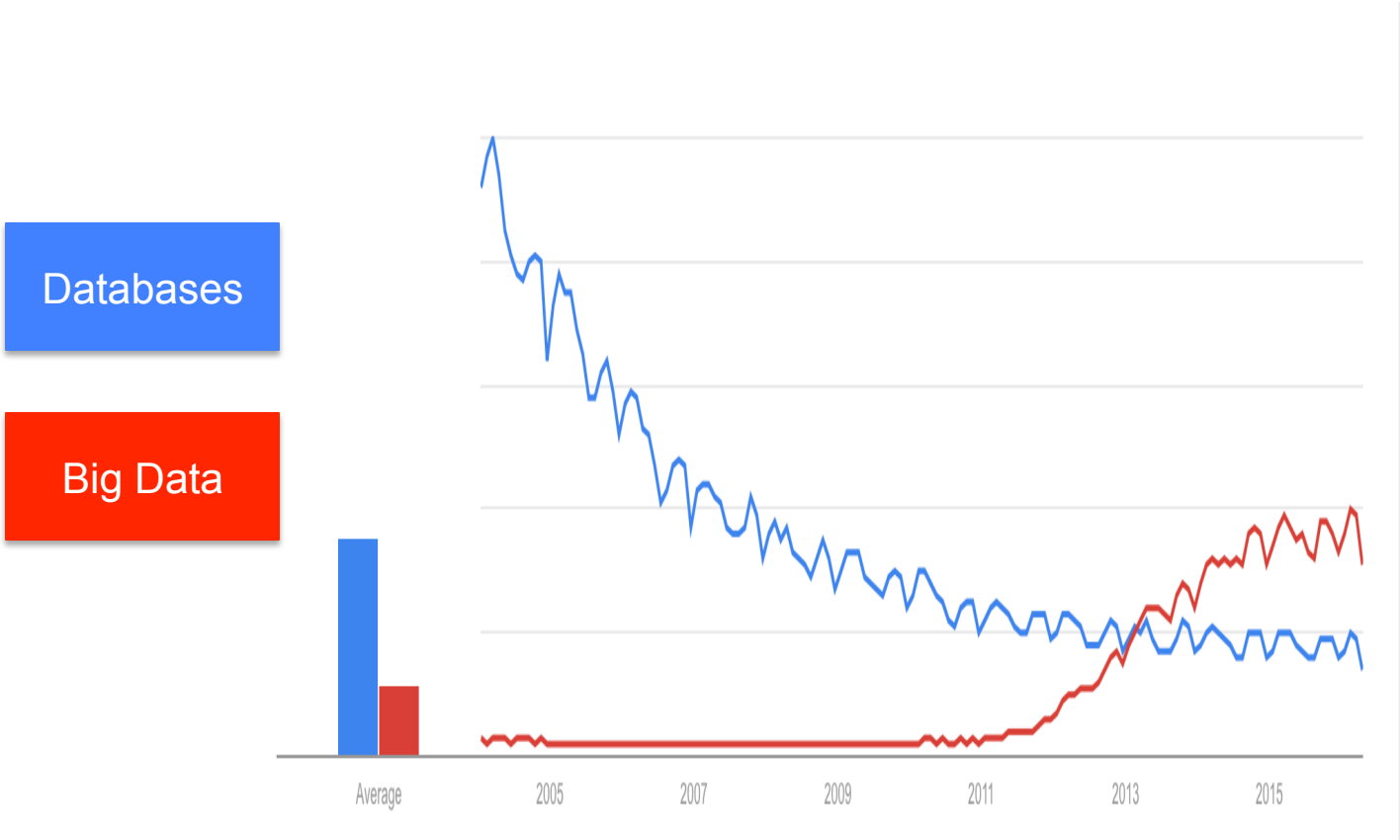
Power of data. Simplicity of design. Speed of innovation.

Hands on Introduction to Apache Spark for Data Engineers, Data Scientists, and Developers

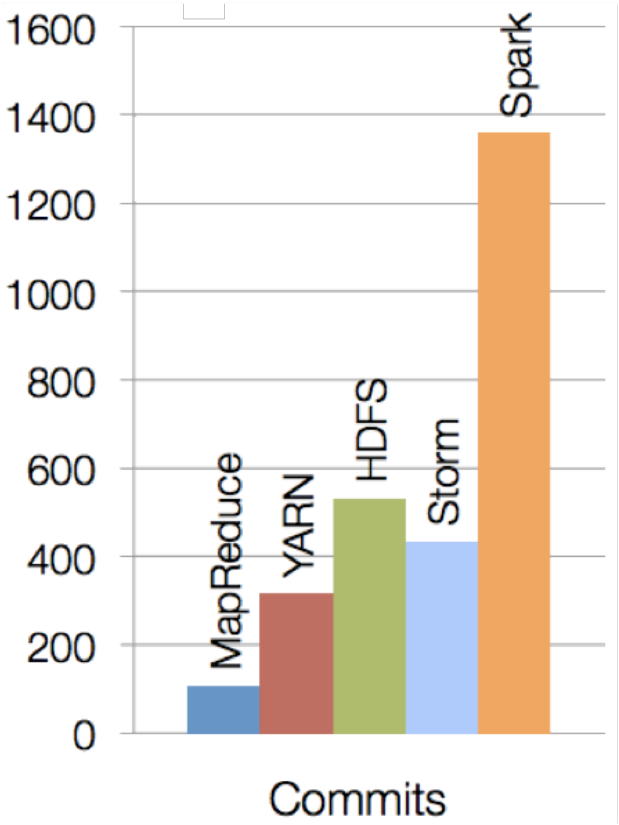
Market Evolution
Databases to Big Data



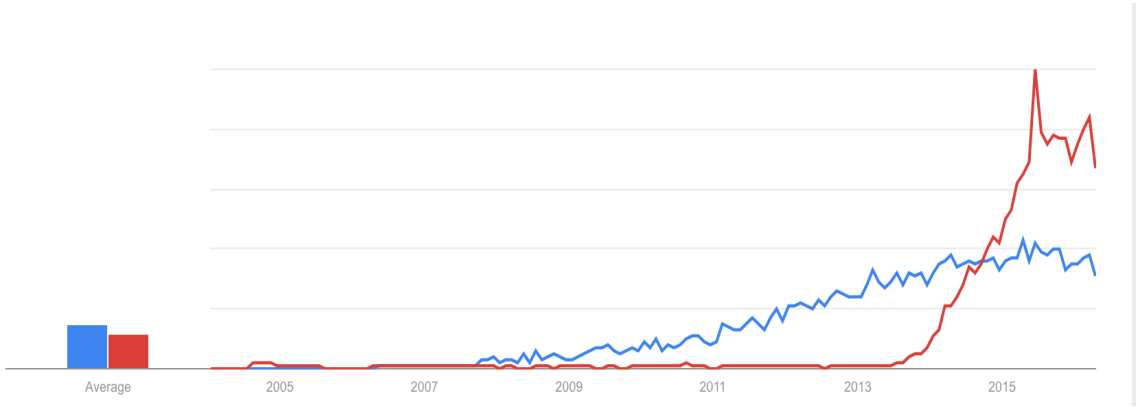
Source: Google Trends

Spark is new, but evolving quickly and has gained tremendous traction in a short amount of time

Spark is one of the most active open source projects



Hadoop vs Spark (Google Trends)



Spark Job Trends (Indeed.com)

What is Spark?

Spark is an **open** source
in-memory
application framework for
distributed data processing and
iterative analysis
on **massive** data volumes

"**Analytic Operating System**"

Key reasons for interest in Spark

Performant

- In-memory architecture greatly reduces disk I/O
- Anywhere from **20-100x faster** for common tasks

Productive

- **Concise and expressive syntax**, especially compared to prior approaches
- **Single programming model** across a range of use cases and steps in data lifecycle
- **Integrated with common programming languages** – Java, Python, Scala
- **New tools** continually reduce skill barrier for access (e.g. SQL for analysts)

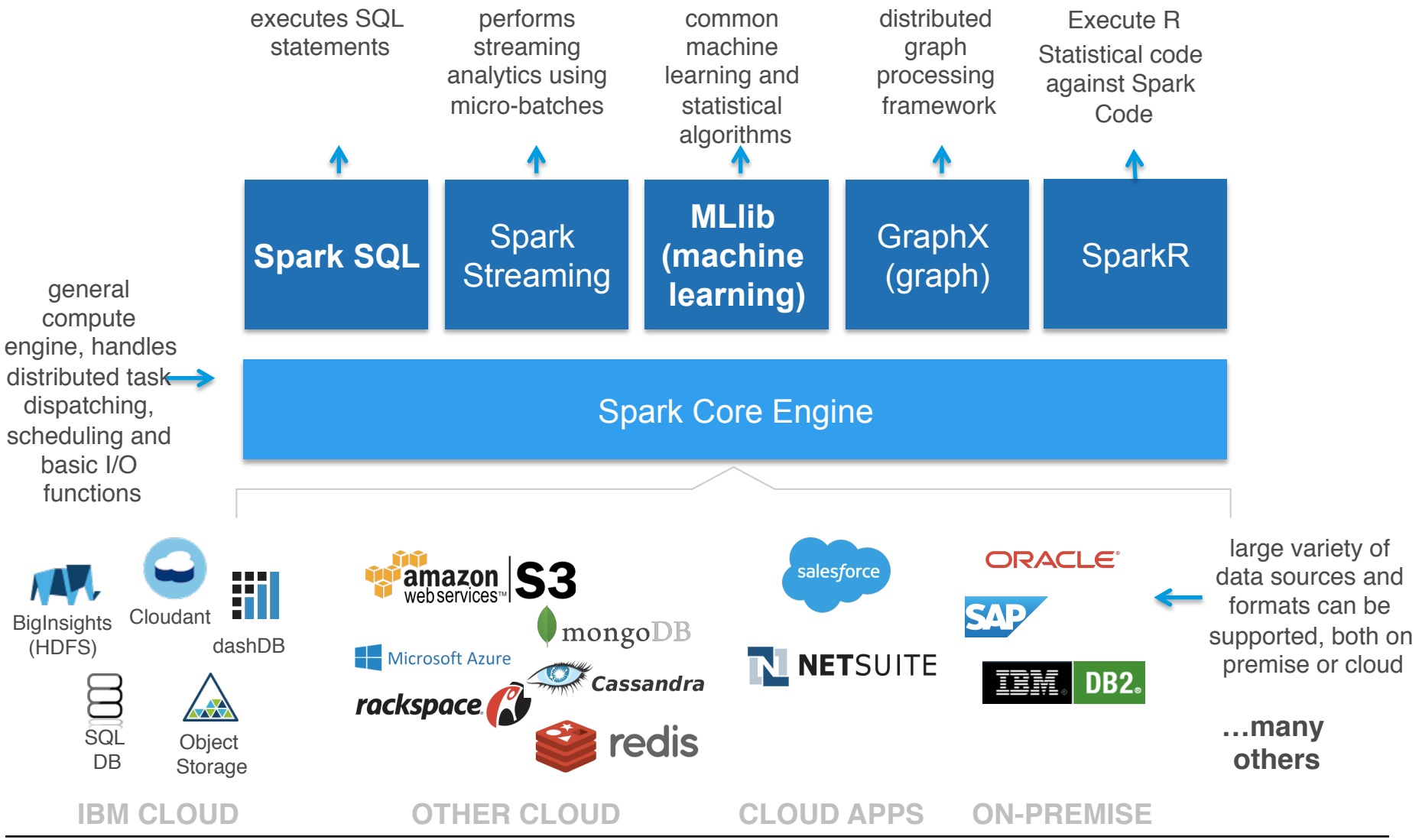
Leverages existing investments

- Works well within **existing Hadoop ecosystem**

Improves with age

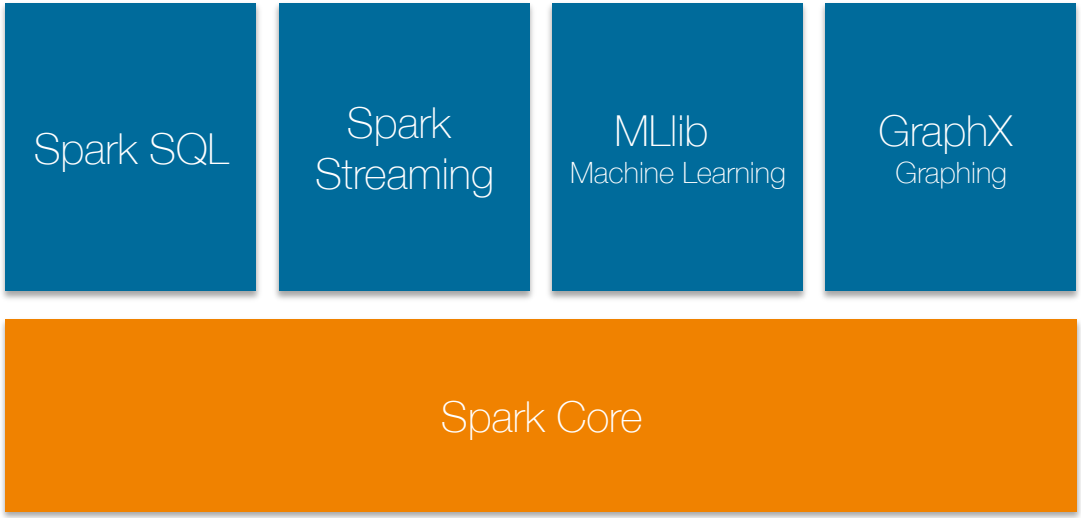
- **Large and growing community** of contributors continuously improve full analytics stack and extend capabilities

Spark includes a set of core libraries that enable various analytic methods which can process data from many sources



Libraries Usage[1]

- **SparkSQL – 69%**
- **DataFrames – 62%**
- **Spark Streaming – 58%**
- **MLlib/GraphX – 58%**



75% of users use more than one component

[1] From a survey done by Databricks, summer 2015

Spark Programming Languages

- **Scala**
  - Functional programming
  - Spark written in Scala
  - Scala compiles into Java byte code
- **Java**
  - New features in Java 8 makes for more compact coding (lambda expressions)
- **Python**
  - Most widely used API with Spark today
- **R**
  - Functional programming language used to create and manipulate functions

Language	2014	2015
Scala	84%	71%
Java	38%	31%
Python	38%	58%
R	unknown	18%

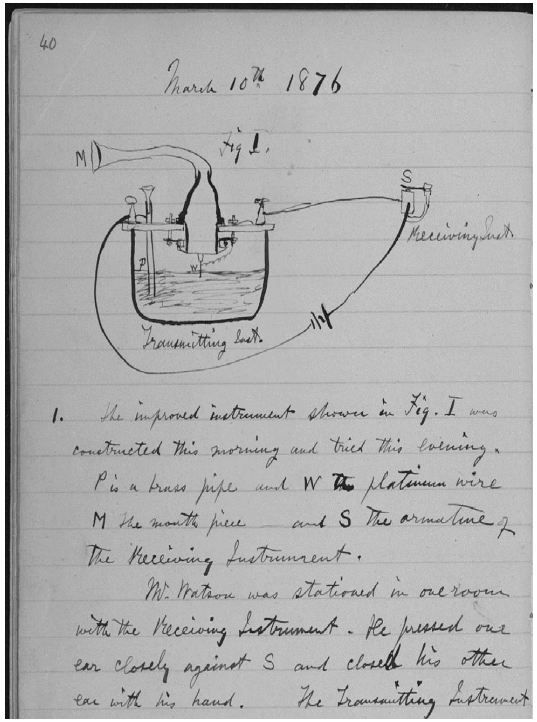
Survey done by Databricks, Summer 2015

This probably means that more "data scientists" are starting to use Spark DataFrames make all languages equally performant

What is a “Notebook”?

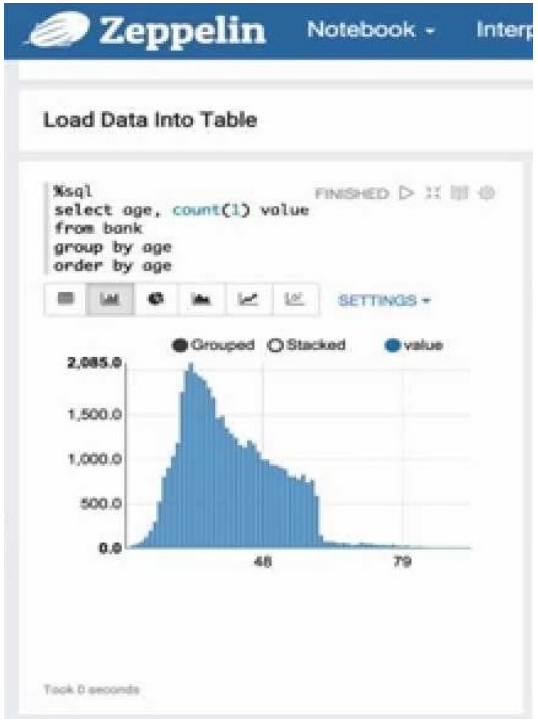
Pen and Paper

- **Pen and paper has long provided the rich experience that scientists need to document progress through notes and drawings:**
  - Expressive
  - Cumulative
  - Collaborative



Notebooks

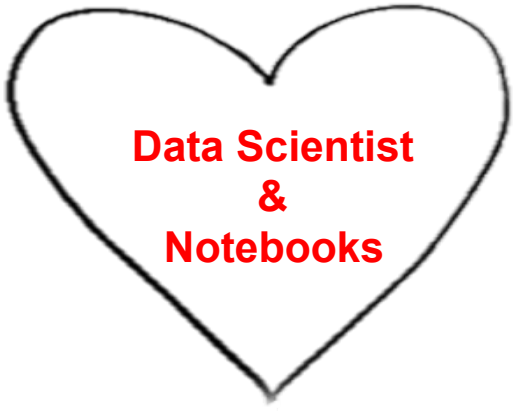
- **Notebooks are the digital equivalent of the “pen and paper” lab notebook, enabling data scientists to document reproducible analysis:**
  - Markdown and visualization
  - Iterative exploration
  - Easy to share

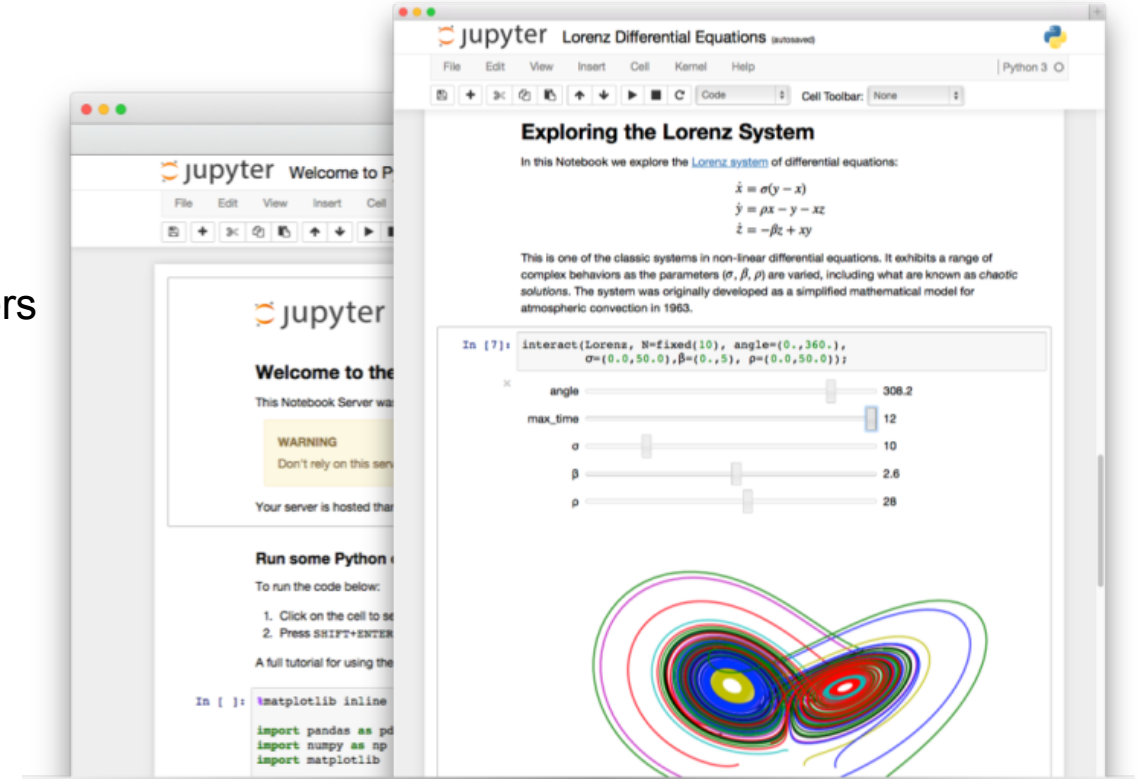


Web-Based Notebooks…

- **Notebooks:**
  "interactive computational environment, in which you can combine code execution, rich text, mathematics, plots and rich media"

  - **Zeppelin**
    - Apache incubator project
    - Suport multiple interpreters
      - Scala, Python, SparkSQL
  - **Jupyter**
    - Based on Ipython
    - Supports multiple interpreters
      - Python, Scala





IBM is all-in on Spark

"It's like Spark just got blessed by the enterprise rabbi."

Ben Horowitz,
Andreessen Horowitz

Contribute to the Core

Launch Spark Technology Cluster (STC), 300 engineers

Open source SystemML

Partner with Databricks

Foster Community

Educate 1M+ data scientists and engineers via online courses

Sponsor AMPLab, creators and evangelists of Spark

Infuse the Portfolio

Integrate Spark throughout portfolio

3,500 employees working on Spark-related topics

Spark however customers want it – standalone, platform or products

IBM Contribute to core Apache Spark Project

WWW.SPARK.TC

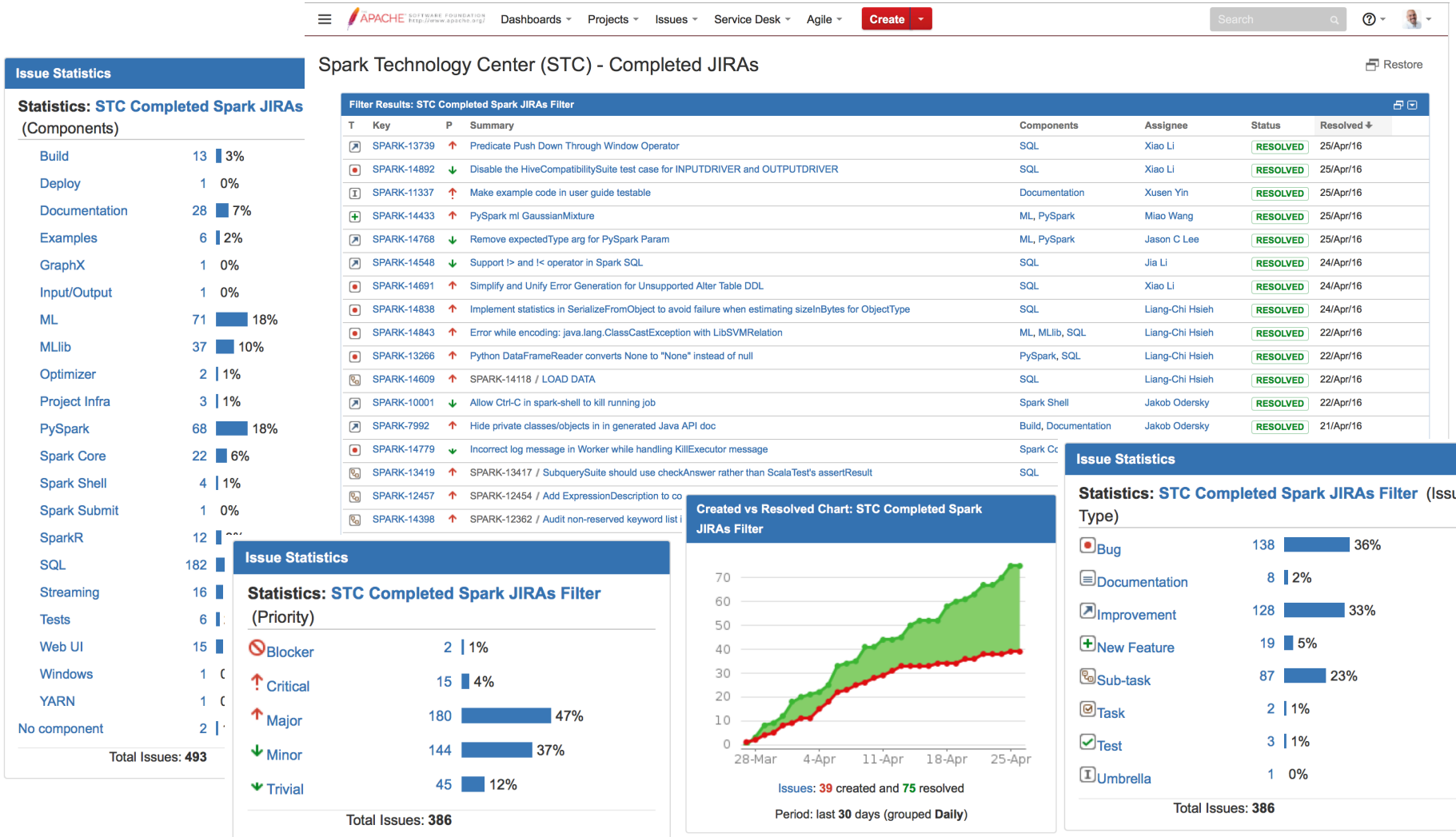
IBM has the largest investment in Spark of any company in the world

IBM Spark Technology Center

- Launched in June of 2015
- Goal to hire 300 Engineers.
- Goal to Contribute to Apache Spark Apache community
- Contributed SystemML technology to Apache community
- STC continues to grow...

STC - JIRAs, check out what we are doing at apache.org

https://issues.apache.org/jira/secure/Dashboard.jspa?selectPageId=12326761



IBM driving SQL and Machine Learning innovation..

Filter Results: STC Completed Spark JIRAs Filter

T	Key	P	Summary	Components
	SPARK-13996	↑	Add more not null attributes for Filter codegen	SQL
	SPARK-14308	↓	SPARK-12326 / Remove unused mllib tree classes and move private classes to ML	ML, MLlib
	SPARK-13241	↓	add long--formatted timestamps to org.apache.spark.status.api.v1.ApplicationAttemptInfo	Web UI
	SPARK-13674	↑	Add wholestage codegen support to Sample	SQL
	SPARK-14191	↑	Fix Expand operator constraints	SQL
	SPARK-13995	↑	Extract correct IsNotNull constraints for Expression	SQL
	SPARK-14264	↓	Add feature importances for GBTs in Pyspark	ML, PySpark
	SPARK-14182	↑	Parse DDL command: Alter View	SQL
	SPARK-11892	↑	SPARK-6725 / Model export/import for spark.ml: OneVsRest	ML
	SPARK-14181	↓	TrainValidationSplit should have HasSeed	ML
	SPARK-14124	↑	SPARK-14118 / Database related commands	SQL
	SPARK-10570	↓	Add Spark version endpoint to standalone JSON API	Spark Core, Web UI
	SPARK-13963	↓	SPARK-13964 / Add binary toggle Param to ml.HashingTF	ML
	SPARK-14071	↓	SPARK-11939 / Change MLWritable.write to be a property	ML, PySpark
	SPARK-11730	↑	Feature Importance for GBT	ML, MLlib
	SPARK-11893	↑	SPARK-6725 / Model export/import for spark.ml: TrainValidationSplit	ML
	SPARK-14156	↓	Use executedPlan for HiveComparisonTest	SQL
	SPARK-13742	↑	Add non-iterator interface to RandomSampler	Spark Core
	SPARK-14177	↑	Parse DDL command: "DESCRIBE DATABASE" and "ALTER DATABASE SET DBPROPERTIES"	SQL
	SPARK-14157	↑	Parse Drop Function DDL command	SQL
	SPARK-14161	↑	Parse Drop Database DDL command	SQL
	SPARK-12443	↑	encoderFor should support Decimal	SQL

Issue Statistics

Statistics: STC Completed Spark JIRAs Filter
(Components)

Build	13	3%
Deploy	1	0%
Documentation	28	7%
Examples	6	2%
GraphX	1	0%
Input/Output	1	0%
ML	71	18%
MLlib	37	10%
Optimizer	2	1%
Project Infra	3	1%
PySpark	68	18%
Spark Core	22	6%
Spark Shell	4	1%
Spark Submit	1	0%
SparkR	12	3%
SQL	182	47%
Streaming	16	4%
Tests	6	2%
Web UI	15	4%
Windows	1	0%
YARN	1	0%
No component	2	1%

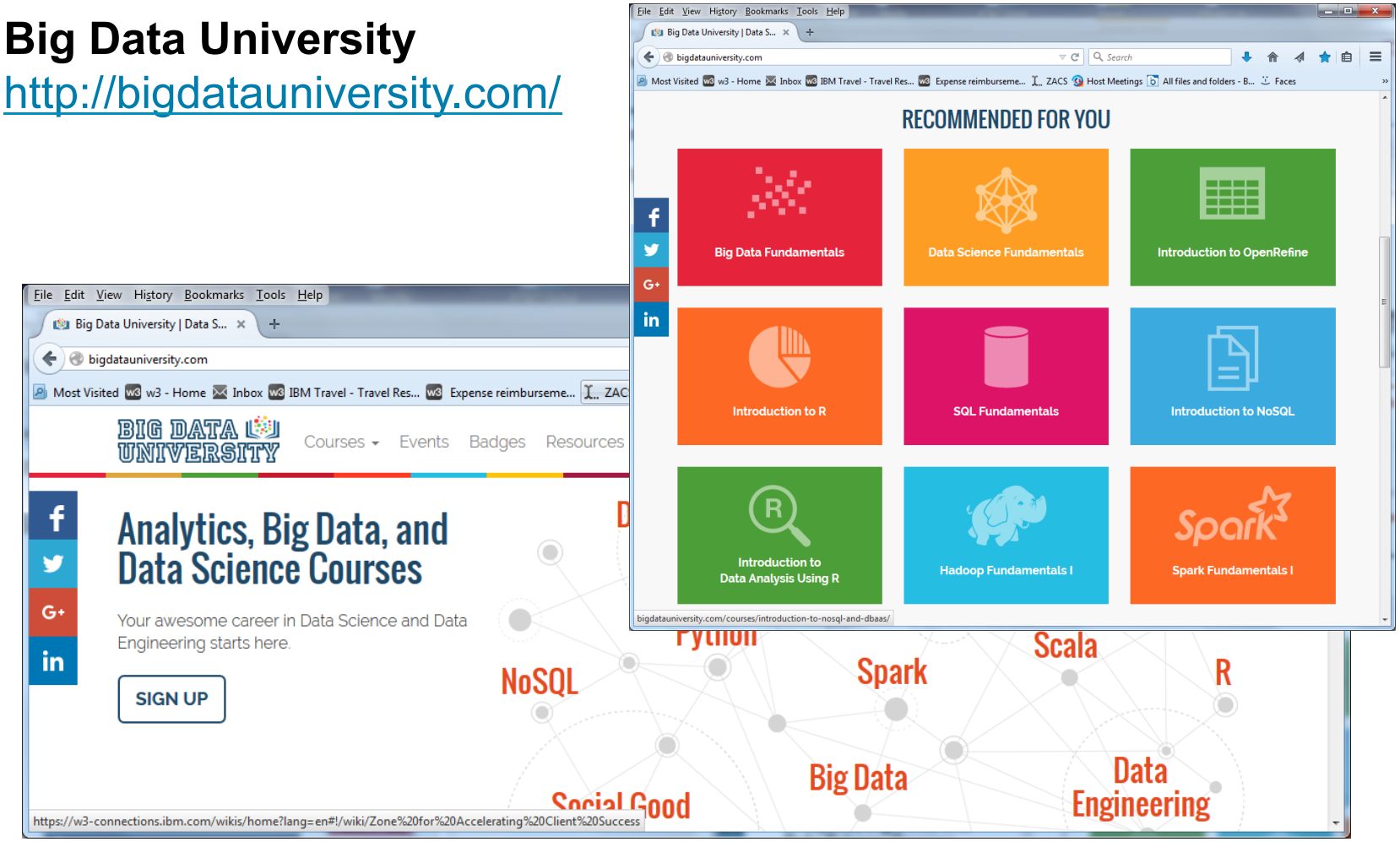
Total Issues: **493**

http://www.spark.tc/blog/

Foster Community - Free Education

Big Data University

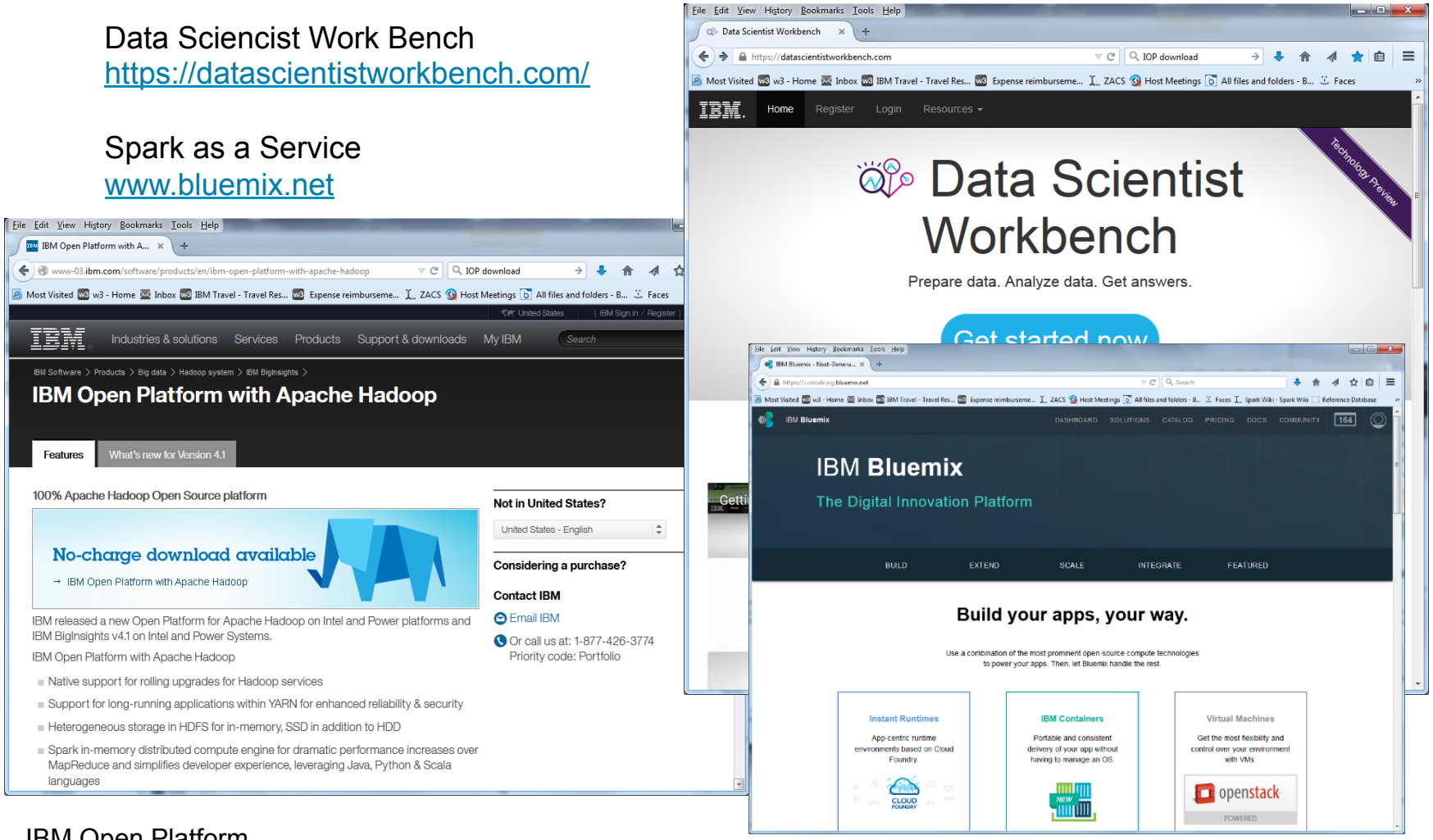
http://bigdatauniversity.com/



Foster Community – Free Platforms

Data Sciencist Work Bench
https://datascientistworkbench.com/

Spark as a Service
www.bluemix.net



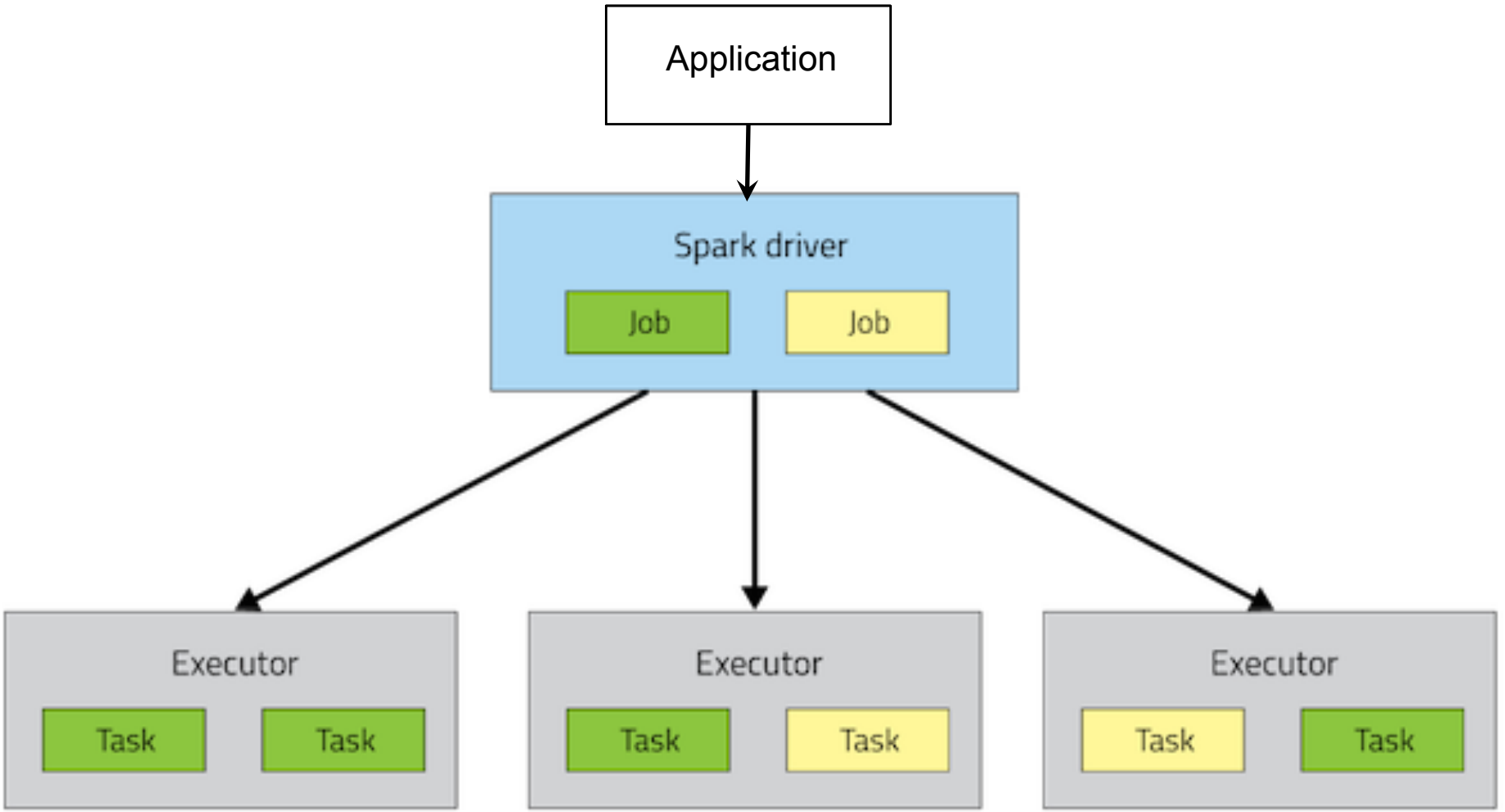
IBM Open Platform
http://www-03.ibm.com/software/products/en/ibm-open-platform-with-apache-hadoop

Apache Spark Under the hood!

Spark Terminology

- **Context (Connection):**
  - Represents a connection to the Spark cluster. The Application which initiated the context can submit one or several jobs, sequentially or in parallel, batch or interactively.
- **Driver (Coordinator agent)**
  - The program or process running the Spark context. Responsible for running jobs over the cluster and converting the App into a set of tasks
- **Job (Query / Query plan):**
  - A piece of logic (code) which will take some input from HDFS (or the local filesystem), perform some computations (transformations and actions) and write some output back.
- **Stage (Subplan)**
  - Jobs are divided into stages
- **Tasks (Sub section)**
  - Each stage is made up of tasks. One task per partition. One task is executed on one partition (of data) by one executor
- **Executor (Sub agent)**
  - The process responsible for executing a task on a worker node

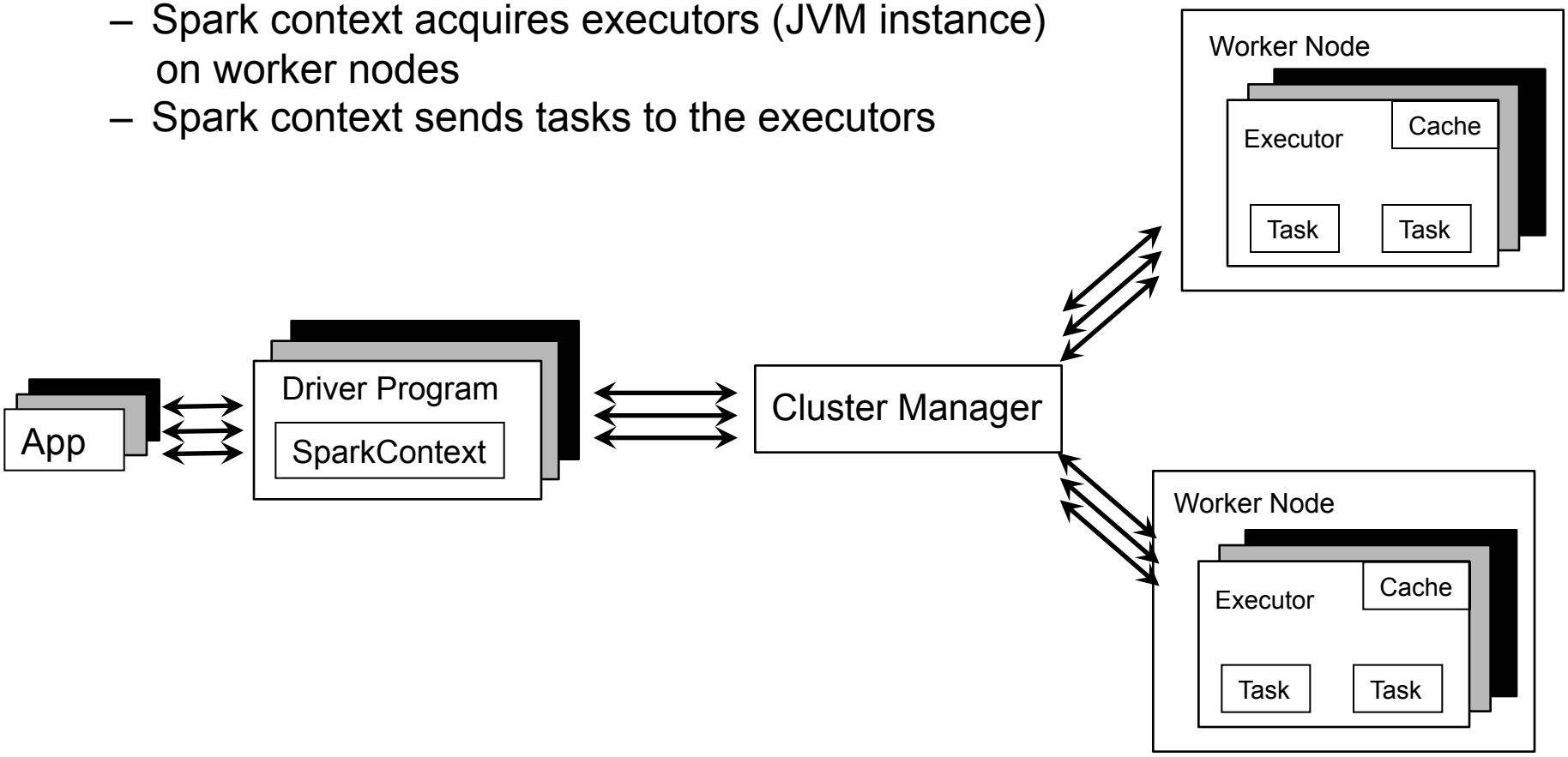
Spark Architecture



Spark Architecture

- **A Spark application is initiated from a driver program**

- **Each Spark application runs as a set of processes coordinated by the Spark context object** (driver program)
  - Spark context connects to Cluster Manager (standalone, Mesos/Yarn)
  - Spark context acquires executors (JVM instance) on worker nodes
  - Spark context sends tasks to the executors

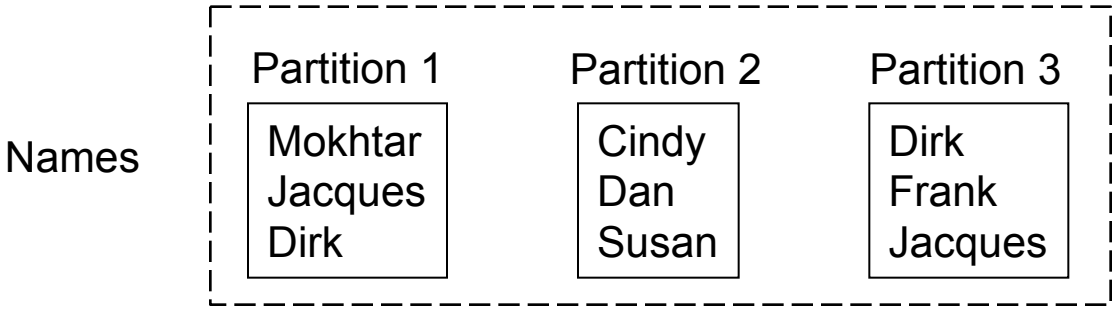


Resilient Distributed Datasets

- **RDD is a distributed collection of Scala/Python/Java objects of the same type:**
  - Strings
  - Integers
  - (key, value) pairs
  - class Java/Python/Scala objects

- **RDDs are immutable**
  Modifications create new RDDs

- **Physically distributed across cluster, but manipulated as one logical entity:**
  - Spark will “distribute” any required processing to all partitions where the RDD exists and perform necessary redistributions and aggregations as well.
  - Example: Consider a distributed RDD “Names” made of names

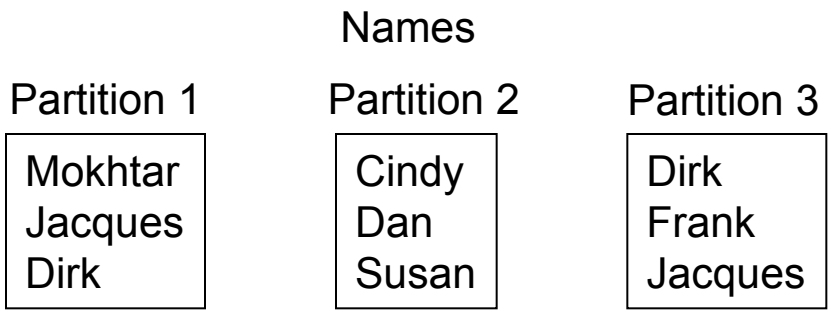


Resilient Distributed Datasets

- **Suppose we want to know the number of names in the RDD "Names"**
- **User simply requests: Names.count()**
  - Spark will "distribute" count processing to all partitions so as to obtain:
    - Partition 1: Mokhtar(1), Jacques (1), Dirk (1) ➔ 3
    - Partition 2: Cindy (1), Dan (1), Susan (1) ➔ 3
    - Partition 3: Dirk (1), Frank (1), Jacques (1) ➔ 3
  - Local counts are subsequently aggregated: 3+3+3=9

- **To lookup the first element in the RDD: Names.first()**

- **To display all elements of the RDD: Names.collect() (careful with this)**



RDDs

- **Two types of operations**
  - **Transformations ~ DDL (Create View V2 as…)**
    - val rddNumbers = sc.parallelize(1 to 10): Numbers from 1 to 10
    - val rddNumbers2 = rddNumbers.map (x => x+1): Numbers from 2 to 11
    - The LINEAGE on how to obtain rddNumbers2 from rddNumber is recorded
    - It's a Directed Acyclic Graph (DAG)
    - No actual data processing does take place ➔ **Lazy evaluations, no Execution**
  - **Actions ~ Select (Select * From V2…)**
    - rddNumbers2.collect(): Array [2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
    - Performs transformations and action
    - Returns a value (or write to a file) ➔ **Executes on Cluster**
- **Fault tolerance**
  - If data in memory is lost it will be recreated from lineage

Code Execution (1)

- ‘spark-shell’ provides Spark context as ‘sc’

```
// Create RDD
val quotes = sc.textFile("hdfs:/sparkdata/sparkQuotes.txt")
// Transformations
val danQuotes = quotes.filter(_.startsWith("DAN"))
val danSpark = danQuotes.map(_.split(" ")).map(x => x(1))
// Action
danSpark.filter(_.contains("Spark")).count()
```

File: sparkQuotes.txt

```
DAN Spark is cool
BOB Spark is fun
BRIAN Spark is great
DAN Scala is awesome
BOB Scala is flexible
```

Code Execution (2)

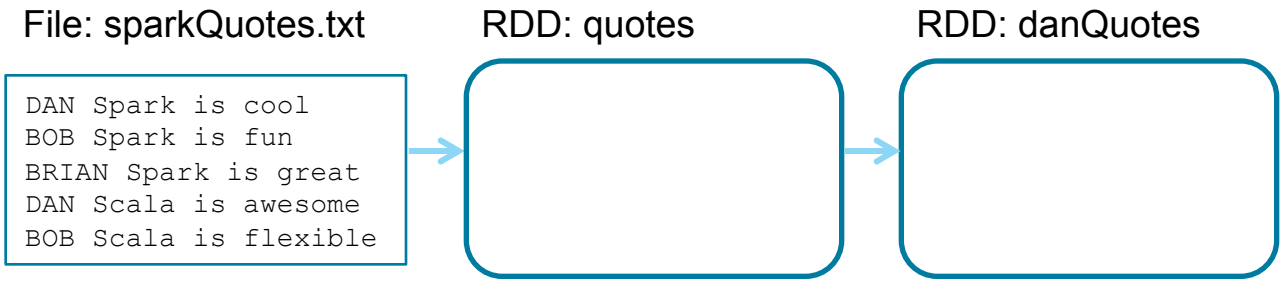
```
// Create RDD
val quotes = sc.textFile("hdfs:/sparkdata/sparkQuotes.txt")
// Transformations
val danQuotes = quotes.filter(_.startsWith("DAN"))
val danSpark = danQuotes.map(_.split(" ")).map(x => x(1))
// Action
danSpark.filter(_.contains("Spark")).count()
```

File: sparkQuotes.txt

```
DAN Spark is cool
BOB Spark is fun
BRIAN Spark is great
DAN Scala is awesome
BOB Scala is flexible
```

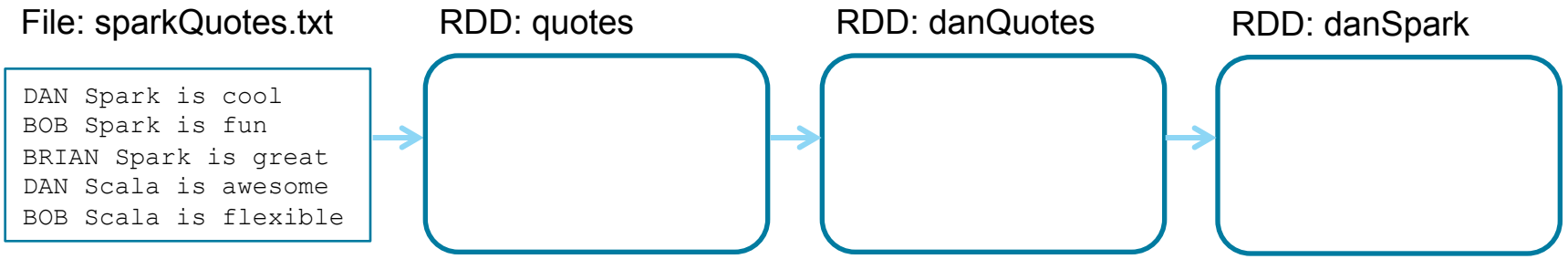
Code Execution (3)

```
// Create RDD
val quotes = sc.textFile("hdfs:/sparkdata/sparkQuotes.txt")
// Transformations
val danQuotes = quotes.filter(_.startsWith("DAN"))
val danSpark = danQuotes.map(_.split(" ")).map(x => x(1))
// Action
danSpark.filter(_.contains("Spark")).count()
```



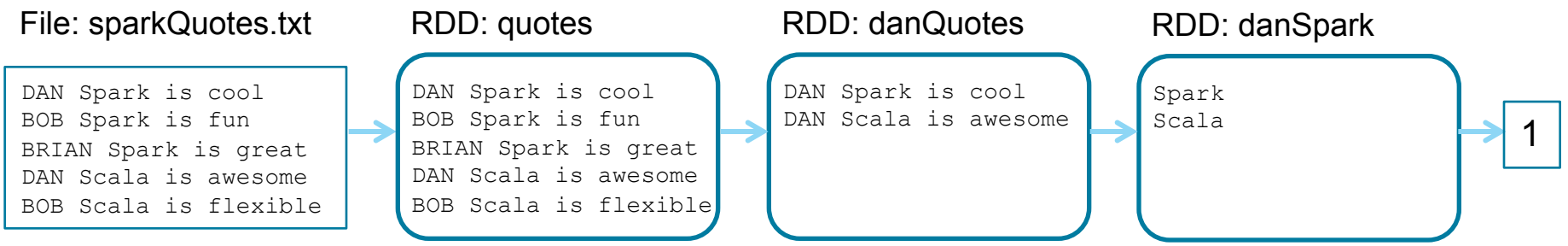
Code Execution (4)

```
// Create RDD
val quotes = sc.textFile("hdfs:/sparkdata/sparkQuotes.txt")
// Transformations
val danQuotes = quotes.filter(_.startsWith("DAN"))
val danSpark = danQuotes.map(_.split(" ")).map(x => x(1))
// Action
danSpark.filter(_.contains("Spark")).count()
```



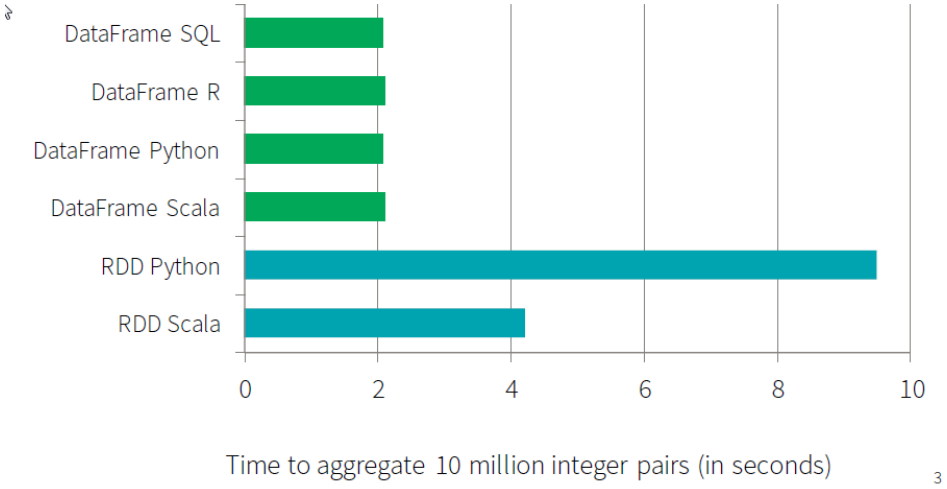
Code Execution (5)

```
// Create RDD
val quotes = sc.textFile("hdfs:/sparkdata/sparkQuotes.txt")
// Transformations
val danQuotes = quotes.filter(_.startsWith("DAN"))
val danSpark = danQuotes.map(_.split(" ")).map(x => x(1))
// Action
danSpark.filter(_.contains("Spark")).count()
```



SparkSQL, DataFrames and DataSets

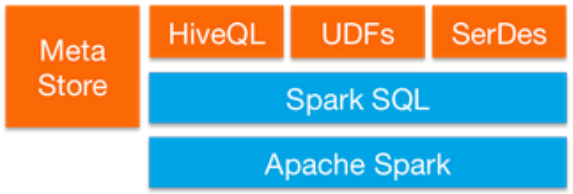
- **A rich set of functionality that allows "Database-like" processing**
- **Share single optimizer, called "Catalyst" (at the driver)**
  - An open-source extensible query optimizer
- **Because it is the same engine, it has exactly the same performance for different APIs**
  - And performance is much better than for RDD
- **Much less code**
- **All SparkSQL, DF, and DataSets are essentially using the same engine**

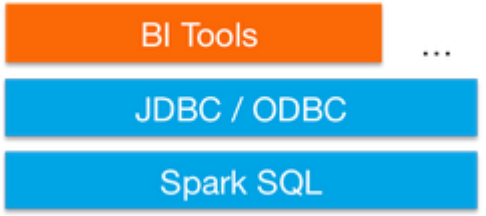


Picture credit: databricks.com

SparkSQL

- **Provide for relational queries expressed in SQL, HiveQL using Scala, Python, and Java API's**
- **Seamlessly mix SQL queries with Spark programs**
- **Dataframes provide a single interface for efficiently working with structured data including Apache Hive, Parquet and JSON files**
- **Graduated from alpha status with Spark 1.3**
  - DataFrames API marked as experimental in 2013
- **Standard connectivity through JDBC/ODBC**





Spark Streaming

- **Component of Spark**
  - Project started in 2012
  - First alpha release in Spring 2013
  - Out of alpha with Spark 0.9.0
  - More enhancements targeted for Spark 2.0

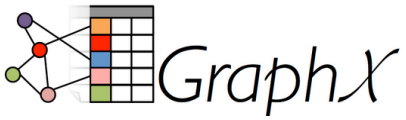
- **Discretized Stream (DStream) programming abstraction**
  - Represented as a sequence of RDDs (micro-batches)
  - RDD: set of records for a specific time interval
  - Supports Scala, Java, and Python (with limitations)

- **Fundamental architecture: batch processing of datasets**

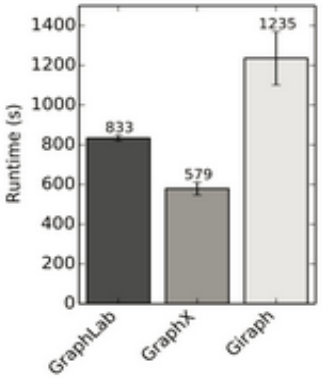
Spark MLlib

- **Spark MLlib for machine learning library**
  - Marked as under active development
- **Provides common algorithm and utilities**
  - Classification
  - Regression
  - Clustering
  - Collaborative filtering
  - Dimensionality reduction
- **Leverages iteration and yields better results than one-pass approximations sometimes used with MapReduce**

Spark GraphX



- **Flexible Graphing**
  - GraphX unifies ETL, exploratory analysis, and iterative graph computation
  - You can view the same data as both graphs and collections, transform and join graphs with RDDs efficiently, and write custom iterative graph algorithms with the API
  - GraphFrames - graph library based on Data Frames
- **Speed**
  - Comparable performance to the fastest specialized graph processing systems.
- **Algorithms**
  - Choose from a growing library of graph algorithms
  - In addition to a highly flexible API, GraphX comes with a variety of graph algorithms



Spark R

- **Spark R is an R package that provides a light-weight front-end to use Apache Spark from R**
- **Spark R exposes the Spark API through the RDD class and allows users to interactively run jobs from the R shell or RStudio on a cluster.**
- **Goals**
  - Make Spark R production ready
    - Efforts from AlteryX and DataBricks
  - Integration with MLlib
  - Consolidations to the data frame and RDD concepts