Hands on Introduction to Spark

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

Joel Patterson
Bernie Beekman
Davin Shearer

Agenda

9:00 - 10:00 - Kick off
- Introduction to Spark
- Introduction to the Data Science Experience
- Register for Platform – datascience.ibm.com

10:00 - 11:30 - Lab 1 – Introduction to Spark
- Overview of Lab
- Hands on Exercises

11:30 – 1:45 Lab 2 – Spark SQL
- Overview of Lab
- Lunch/Hands on Exercises

1:45 - 2:00 – Break

2:00 - 3:45 - Lab 3 – Spark Machine Learning
- Overview of Lab
- Hands on Exercises

3:45 - 4:00 – Break

4:00 - 4:30 – Questions/Wrap Up

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**"

Why Spark?

Performant

- In-memory architecture greatly reduces disk I/O
- Improved performance compared to MapReduce

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, R
- **New tools** continually reduce skill barrier for access (e.g. SQL for analysts)

Leverages existing investments

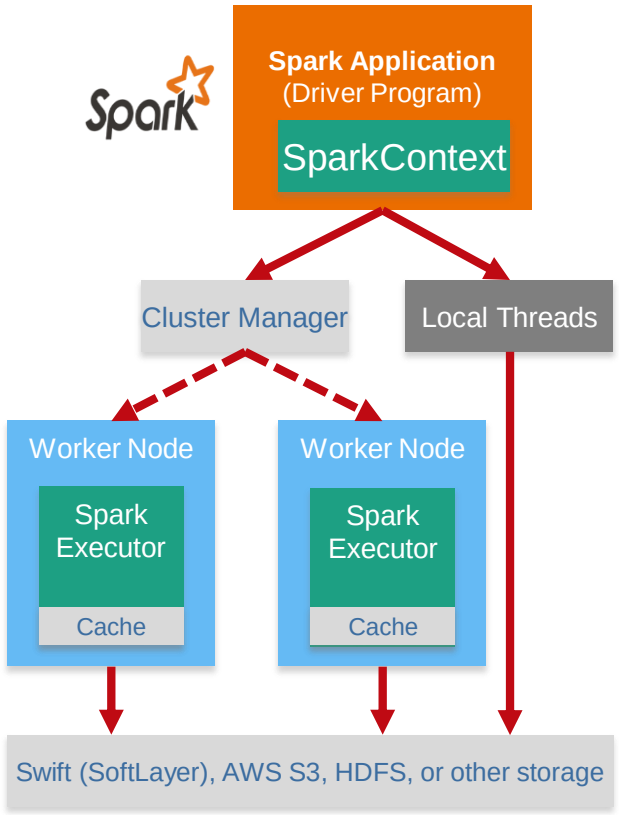
- Works well within **existing Hadoop ecosystem**

Improves with age

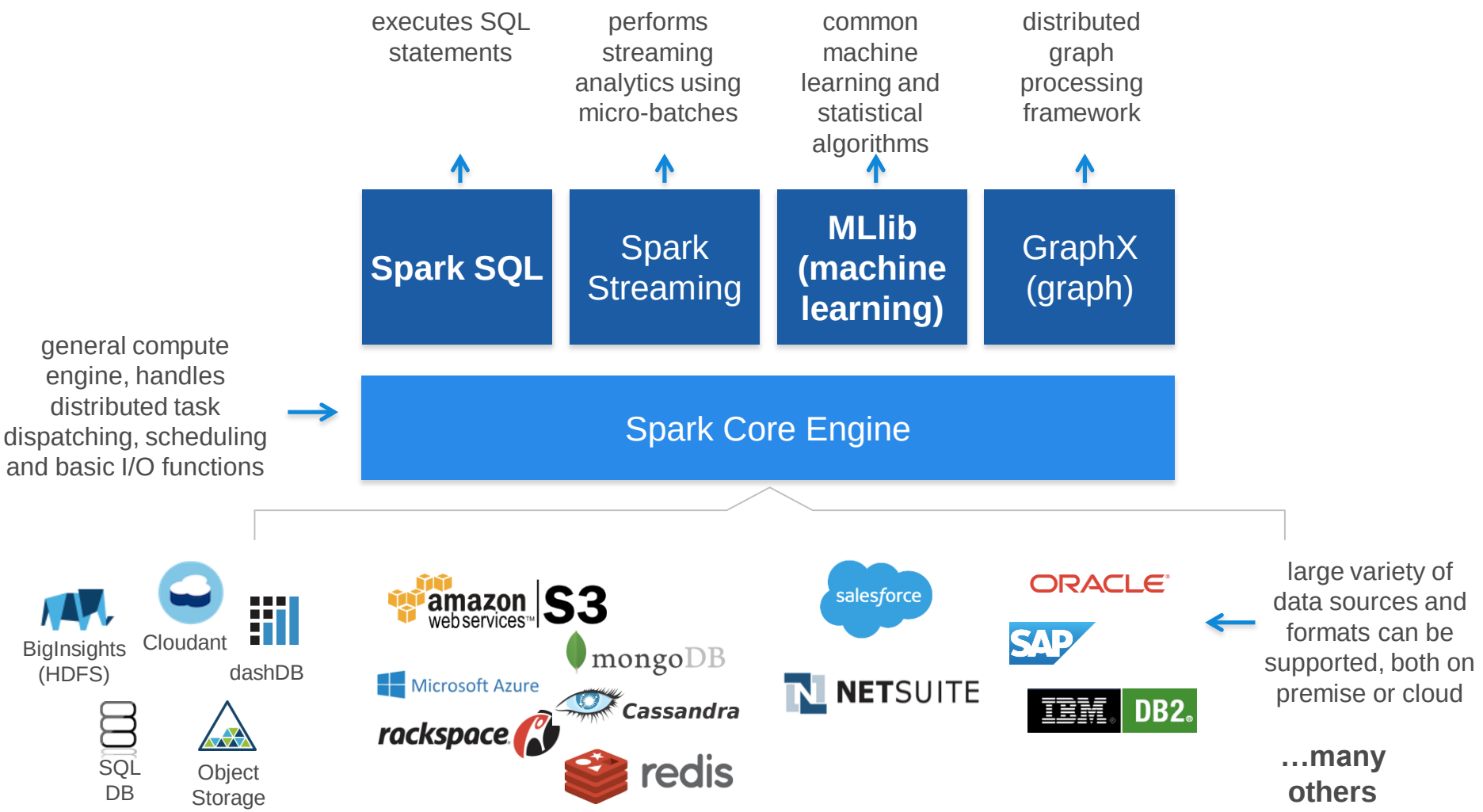
- **Large and growing community** of contributors continuously improve full analytics stack and extend capabilities

Apache Spark Scales

- **Spark programs generally consist of two components: driver program and worker program(s)**
  - **Driver Program** manages the division of computations (Task) that are sent to worker nodes
  - **Worker programs** run smaller portions of computations
- **The `SparkContext` object instructs Spark on how & where to access a cluster**
- **Cluster Manager manages the physical resources needed to run driver and worker programs.**



Spark blends multiple data types, sources, and workloads



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 for statistical computing

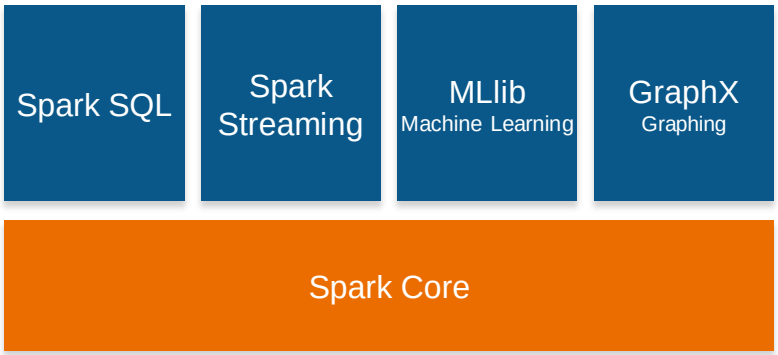
Language	2014	2015
Scala	84%	71%
Java	38%	31%
Python	38%	58%
R	unknown	18%

Survey done by Databricks, Summer 2015

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

Benefits of Spark for Data Science

- General compute engine
- Basic I/O functions
- Task dispatching
- Scheduling



Spark is Easy…

→ **Support multiple programing interfaces (Scala, Python, Java and R)**

→ **Less lines of code to get answers.**

Spark is Agile…

→ **Unified APIs (SQL, DataFrames, Streaming, Machine Learning, etc.)**

→ **Supports Notebooks (Jupyter, Zeppelin, Etc.)**

Spark is Fast…

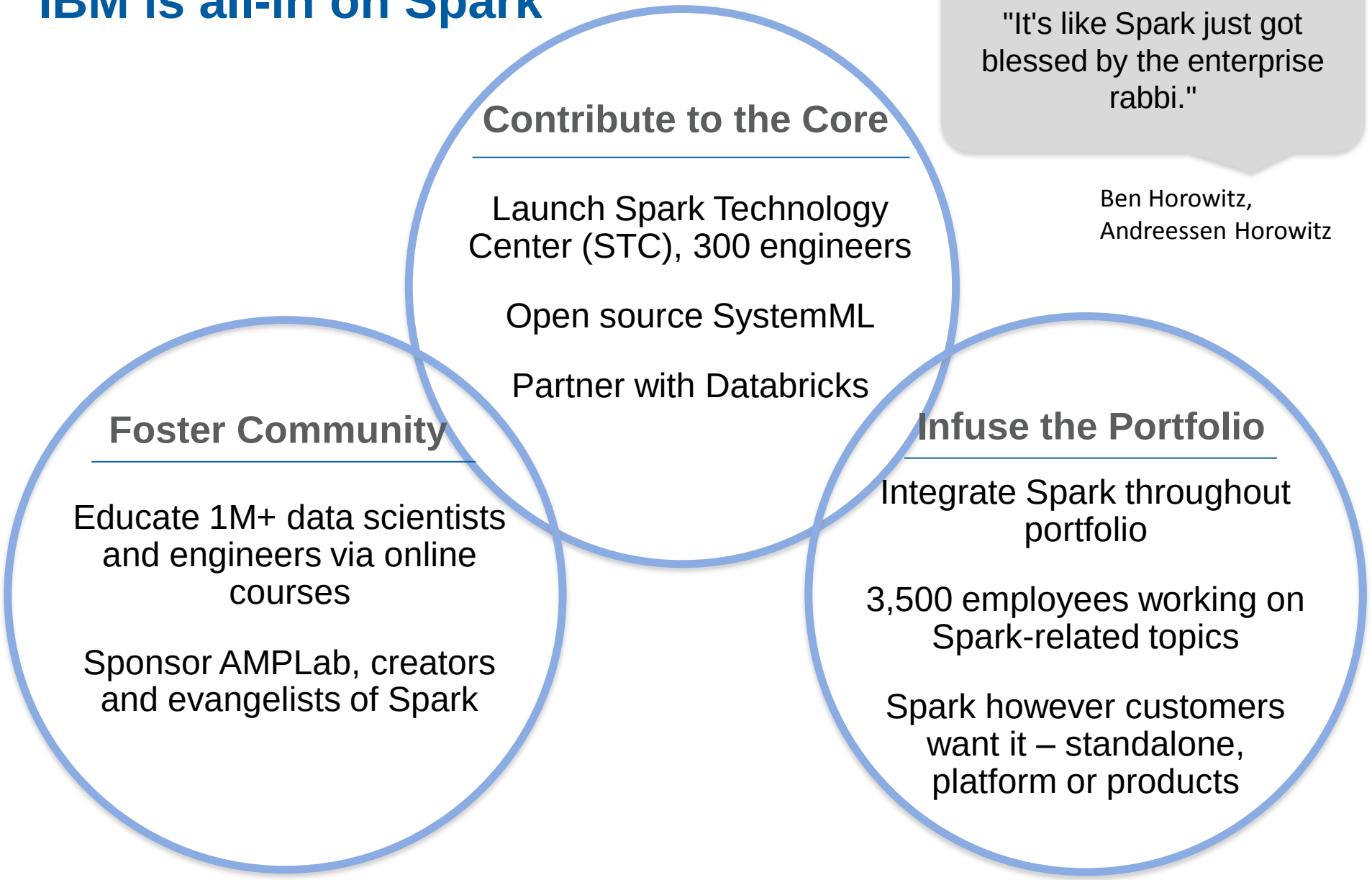
→ **In-Memory processing that scales in a distributed architecture.**

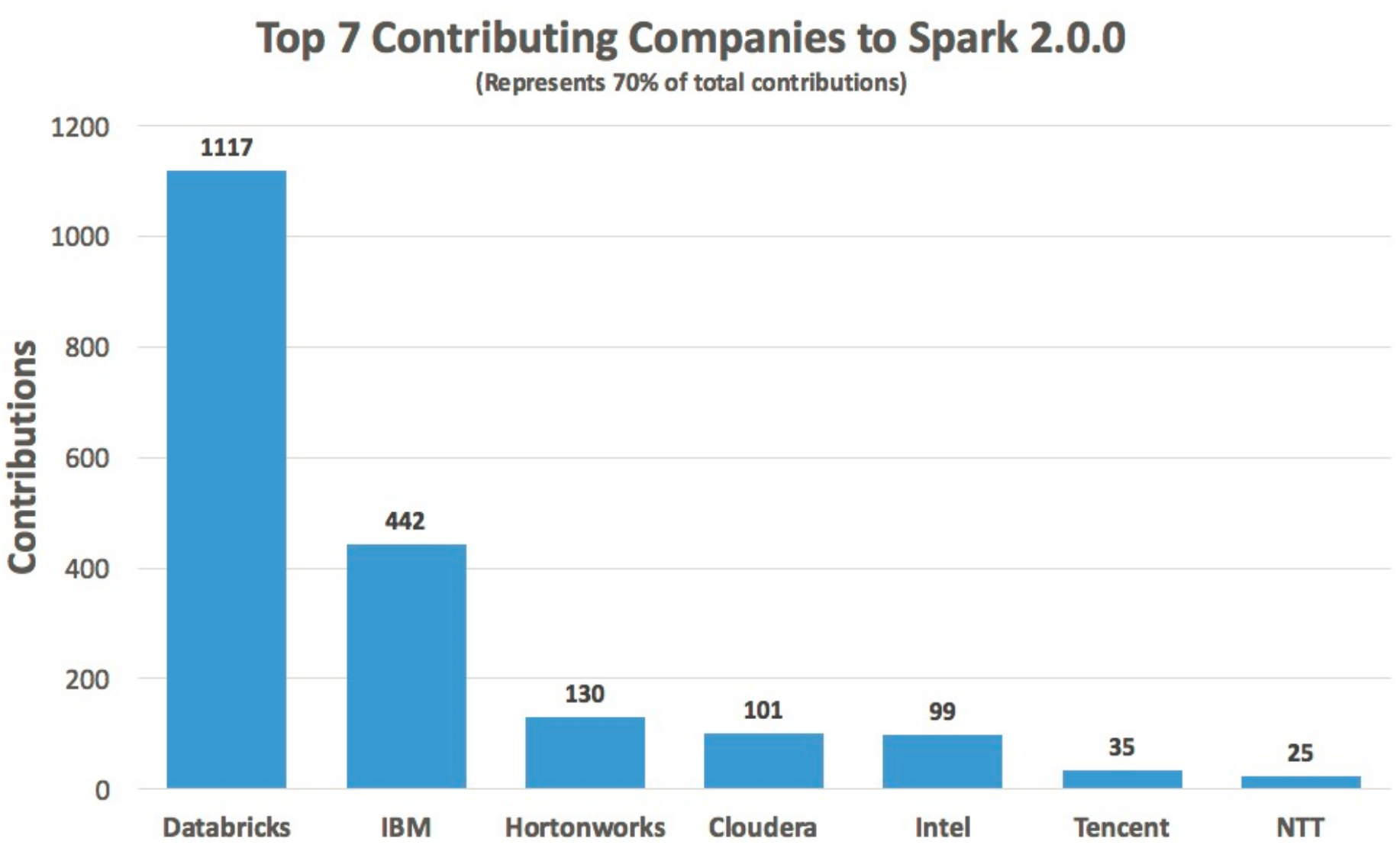
→ **Application follows lazy evaluations architecture w/ optimized execution.**

IBM is all-in on Spark

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

Ben Horowitz, Andreessen Horowitz





Apache Spark Under the hood!

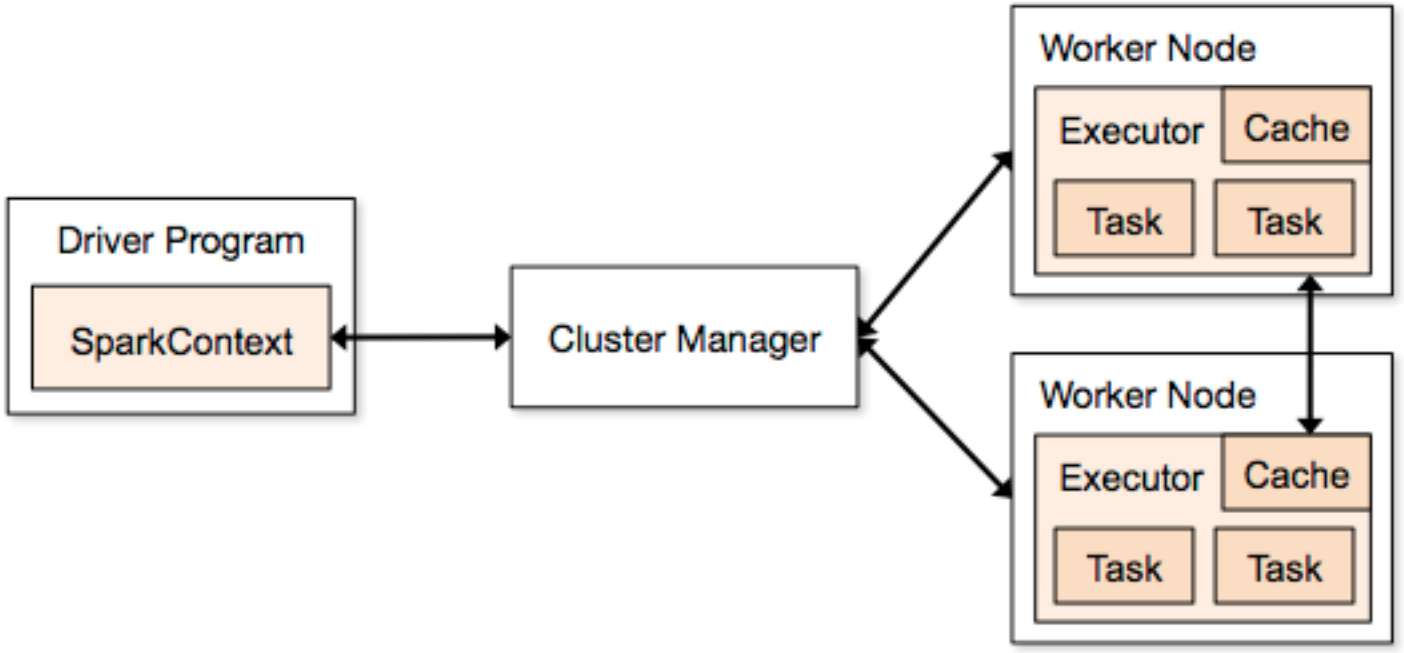
Spark Terminology

- **Spark 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 Application Architecture

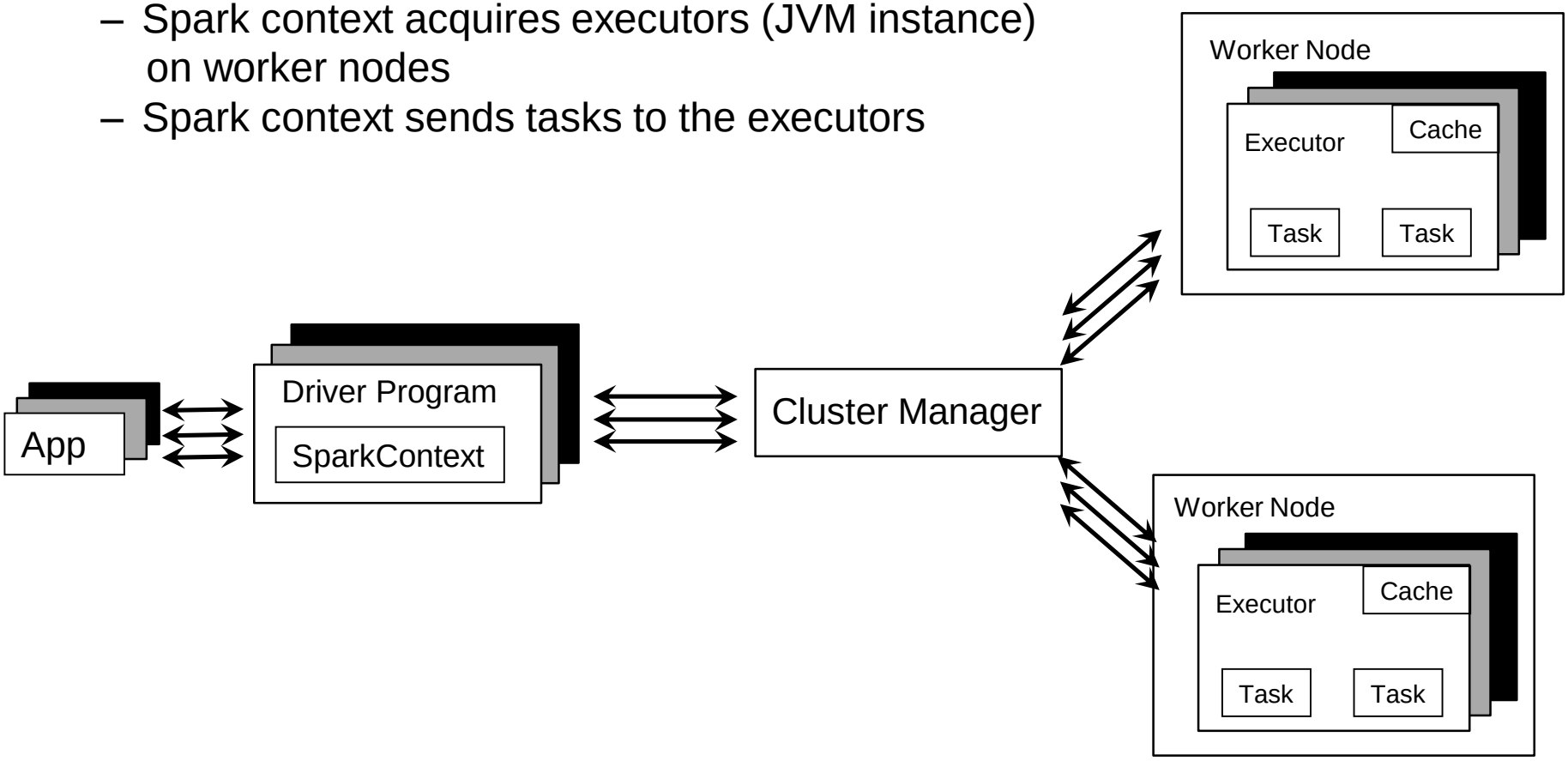
- **A Spark application is initiated from a driver program**

- **Spark execution modes:**
  - Standalone with the built-in cluster manager
  - Use Mesos as the cluster manager
  - Use YARN as the cluster manager
  - On-premise or any cloud (IBM BlueMix, Amazon, Azure, …)



Showing multiple applications

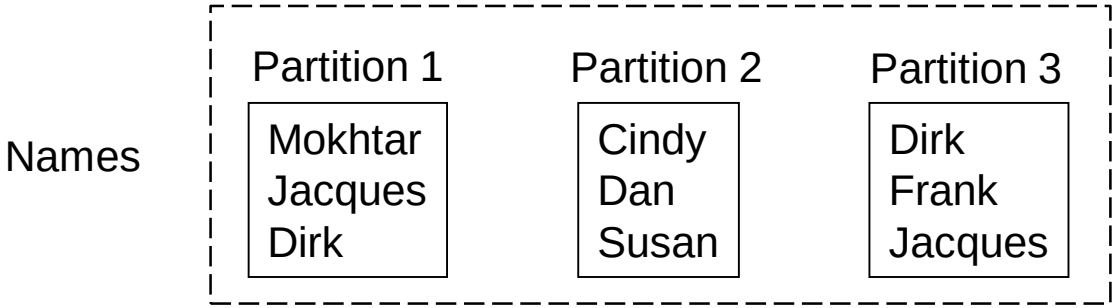
- Each Spark application runs as a set of processes coordinated by the Spark context object (driver program)
  - Spark context connects to Cluster Manager (standalone, Mesos/Yarn)
  - Spark context acquires executors (JVM instance) on worker nodes
  - Spark context sends tasks to the executors



Resilient Distributed Datasets

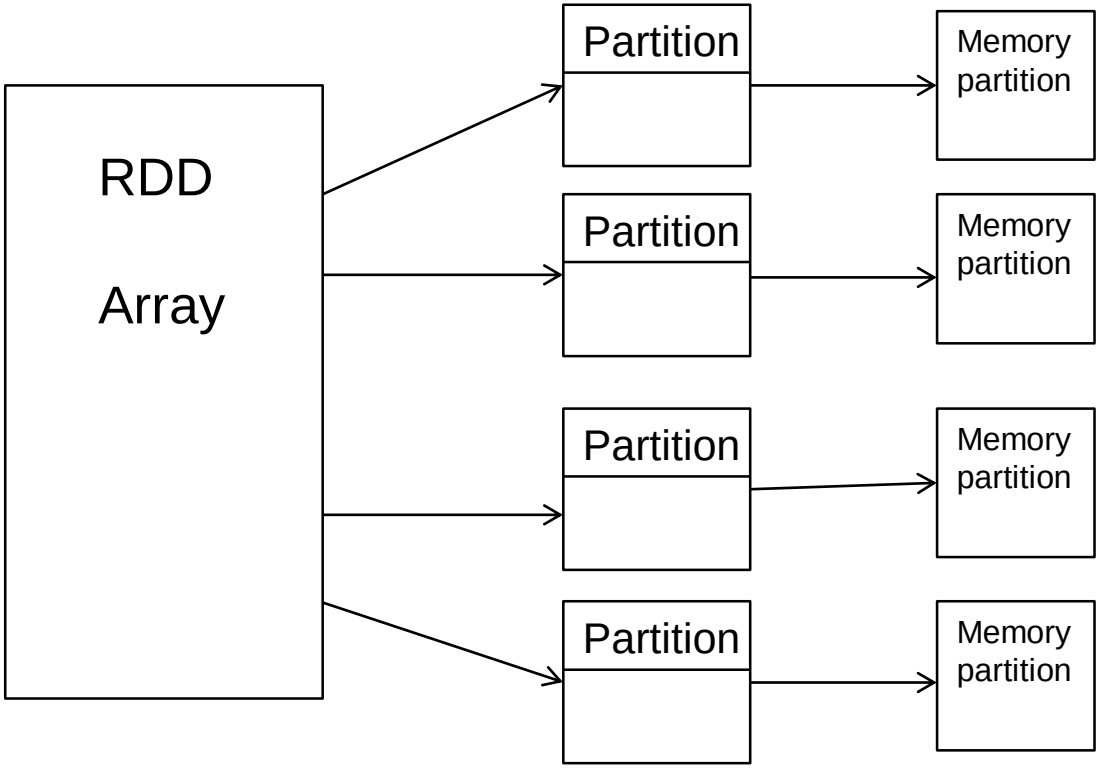
- **An RDD is a distributed collection of Scala/Python/Java/R objects of the same type:**
  - RDD of strings
  - RDD of integers
  - RDD of (key, value) pairs
  - RDD of class Java/Python/Scala/R objects

- **An RDD is physically distributed across the 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 Dataset

- **RDDs are immutable**
  - Modifications create new RDDs
- **Holds references to partition objects**
- **Each partition is a subset of the overall data**
- **Partitions are assigned to nodes on the cluster**
- **Partitions are in memory by default**
- **RDDs keep information on their lineage**
  - Fault tolerant
  - If data in memory is lost it will be recreated



Resilient Distributed Datasets

- **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**
    - 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**
  - **Actions ~ Select (Select * From V2…)**
    - **rddNumbers2.collect(): Array [2, 3, 4, 5, 6, 7, 8, 9, 10, 11]**
    - Performs list of transformations and THE action
    - Returns a value (or write to a file)
- **Fault tolerance**
  - If data in memory is lost it will be recreated from lineage

Transformations used in Labs

Transformation	Meaning
map(*func*)	Return a new distributed dataset formed by passing each element of the source through a function *func*.
filter(*func*)	Return a new dataset formed by selecting those elements of the source on which *func* returns true.
flatMap(*func*)	Similar to map, but each input item can be mapped to 0 or more output items (so *func* should return a Seq rather than a single item).
reduceByKey(*func*, [*numTasks*])	When called on a dataset of (K, V) pairs, returns a dataset of (K, V) pairs where the values for each key are aggregated using the given reduce function *func*, which must be of type (V,V) => V. Like in groupByKey, the number of reduce tasks is configurable through an optional second argument.

Actions used in Labs

Action	Meaning
reduce(*func*)	Aggregate the elements of the dataset using a function *func* (which takes two arguments and returns one). The function should be commutative and associative so that it can be computed correctly in parallel.
collect()	Return all the elements of the dataset as an array at the driver program. This is usually useful after a filter or other operation that returns a sufficiently small subset of the data.
count()	Return the number of elements in the dataset.
first()	Return the first element of the dataset (similar to take(1)).
take(*n*)	Return an array with the first *n* elements of the dataset.

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()
```

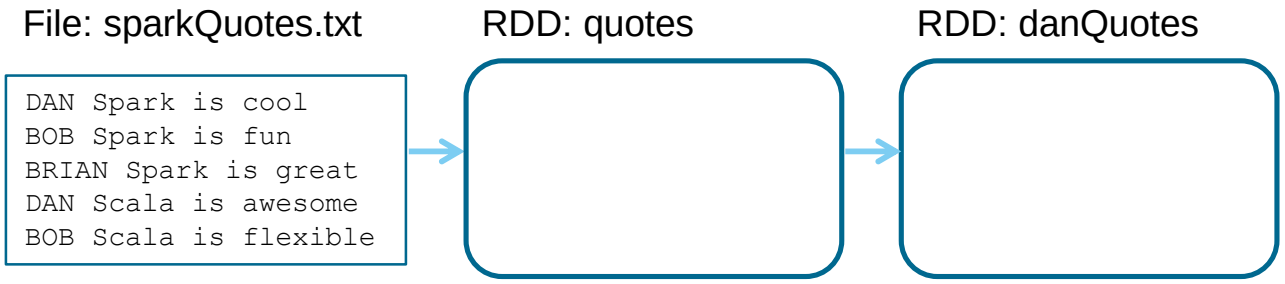
File: sparkQuotes.txt

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

RDD: quotes

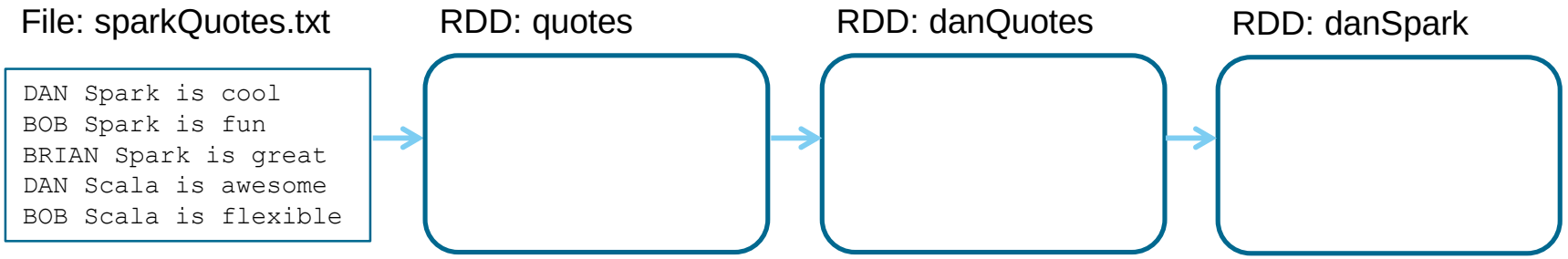
Code Execution (3)

```
// 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()
```



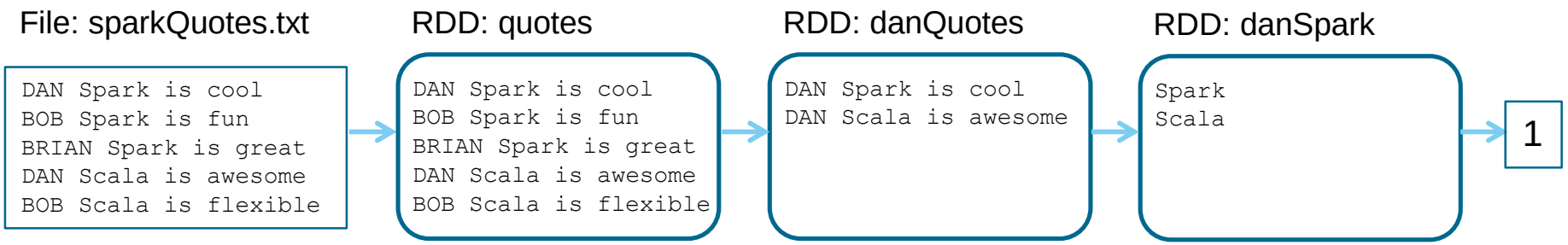
Code Execution (4)

```
// 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()
```

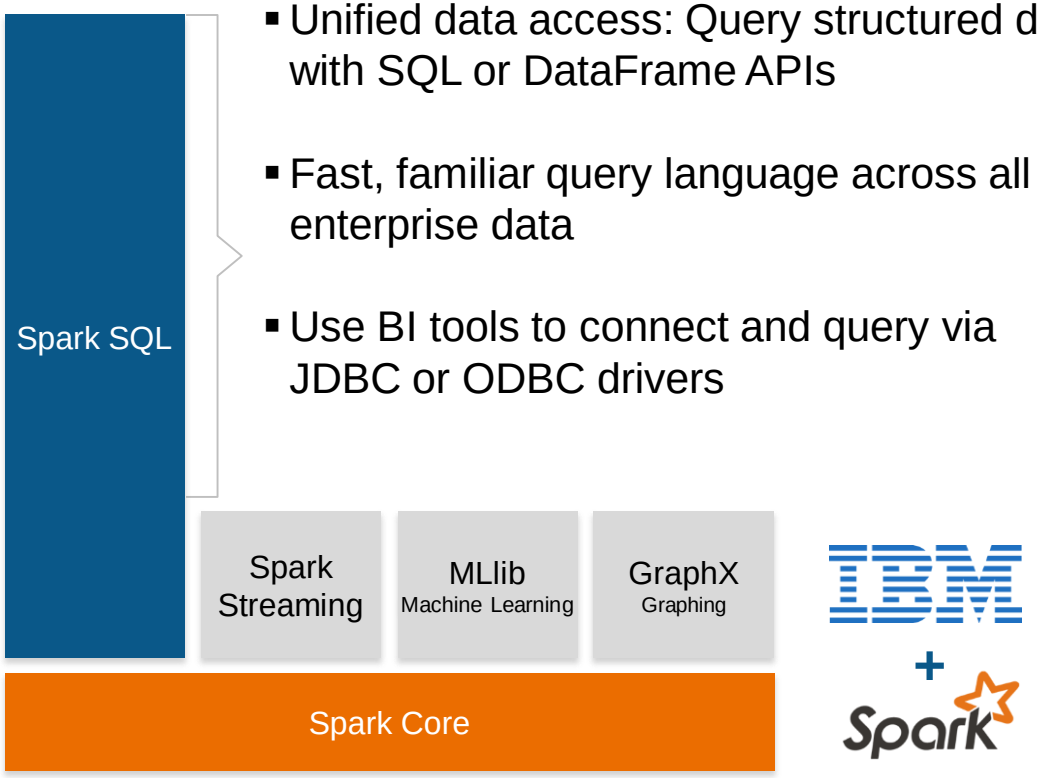


Code Execution (5)

```
// 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()
```



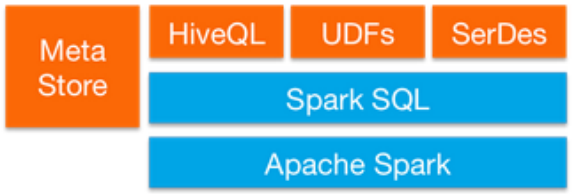
Closer Look at APIs - SQL

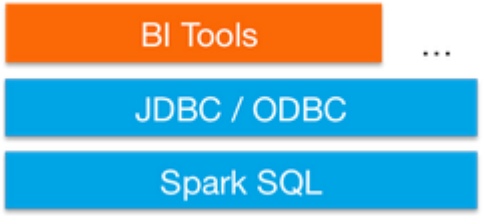


- Unified data access: Query structured data sets with SQL or DataFrame APIs
- Fast, familiar query language across all of your enterprise data
- Use BI tools to connect and query via JDBC or ODBC drivers

SparkSQL

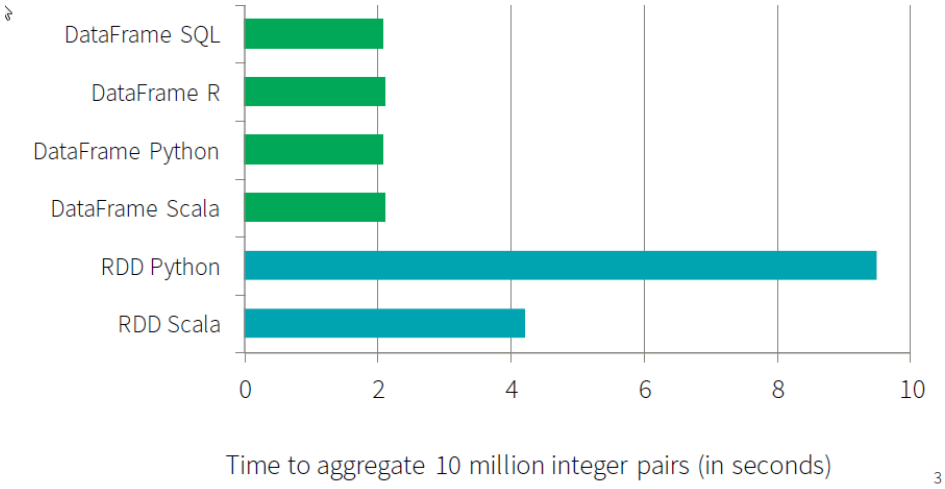
- **Provide for relational queries expressed in SQL, HiveQL using Scala, Python, and Java API's**
- **Seamlessly mix SQL queries with Spark programs**
- **Dataframes provide a single interface for efficiently working with structured data including Apache Hive, Parquet and JSON files**
- **Graduated from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





SparkSQL, DataFrames and DataSets

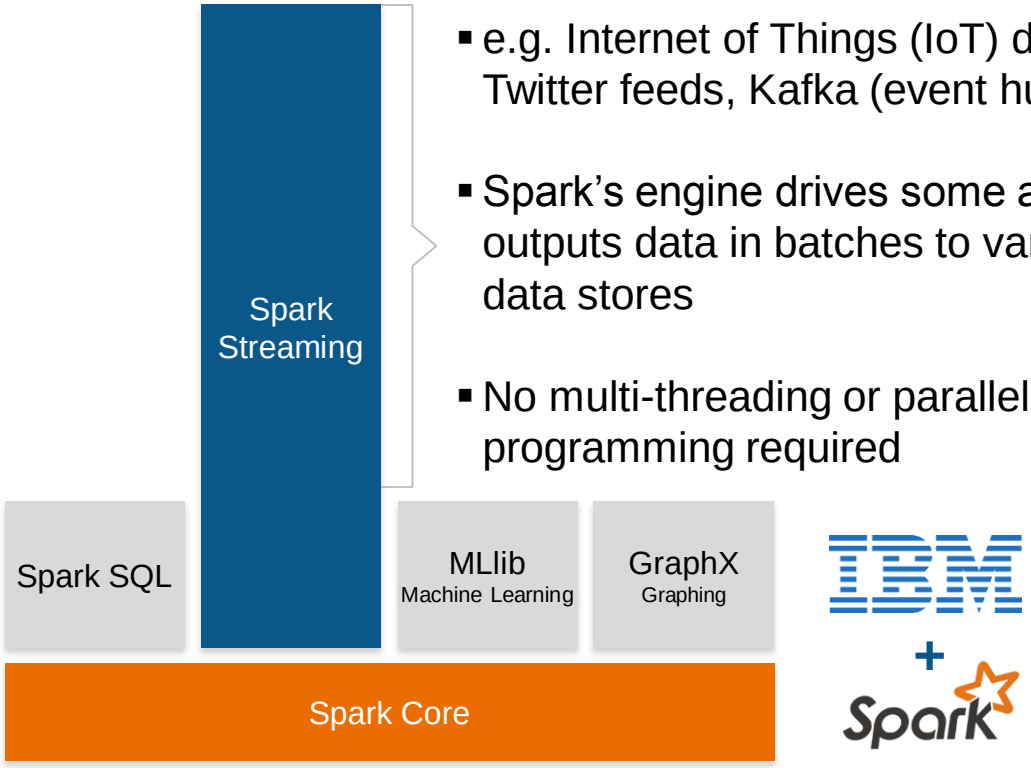
- **A rich set of functionality that allows "Database-like" processing**
- **Share single optimizer, called "Catalyst" (at the driver)**
  - An open-source extensible query optimizer
- **Because it is the same engine, it has exactly the same performance for different APIs**
  - And performance is much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



Picture credit: databricks.com

Closer Look at APIs - Streaming

- Micro-batch event processing for near-real time analytics
- e.g. Internet of Things (IoT) devices, Twitter feeds, Kafka (event hub), etc.
- Spark's engine drives some action or outputs data in batches to various data stores
- No multi-threading or parallel process programming required



Spark Streaming

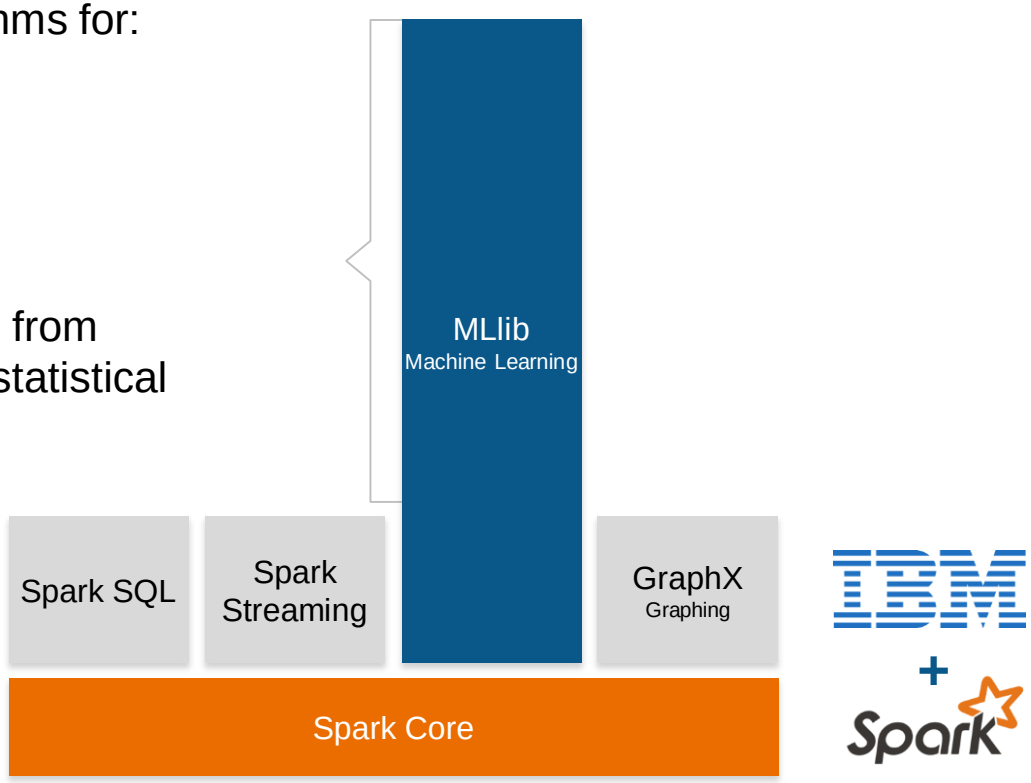
- **Component of Spark**
  - Project started in 2012
  - First alpha release in Spring 2013
  - Out of alpha with Spark 0.9.0
  - More enhancements targeted for Spark 2.0

- **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**

Closer Look at APIs – Machine Learning

- Predictive and prescriptive analytics
- Machine learning algorithms for:
  - Clustering
  - Classification
  - Regression
  - etc.
- Smart application design from pre-built, out-of-the-box statistical and algorithmic models

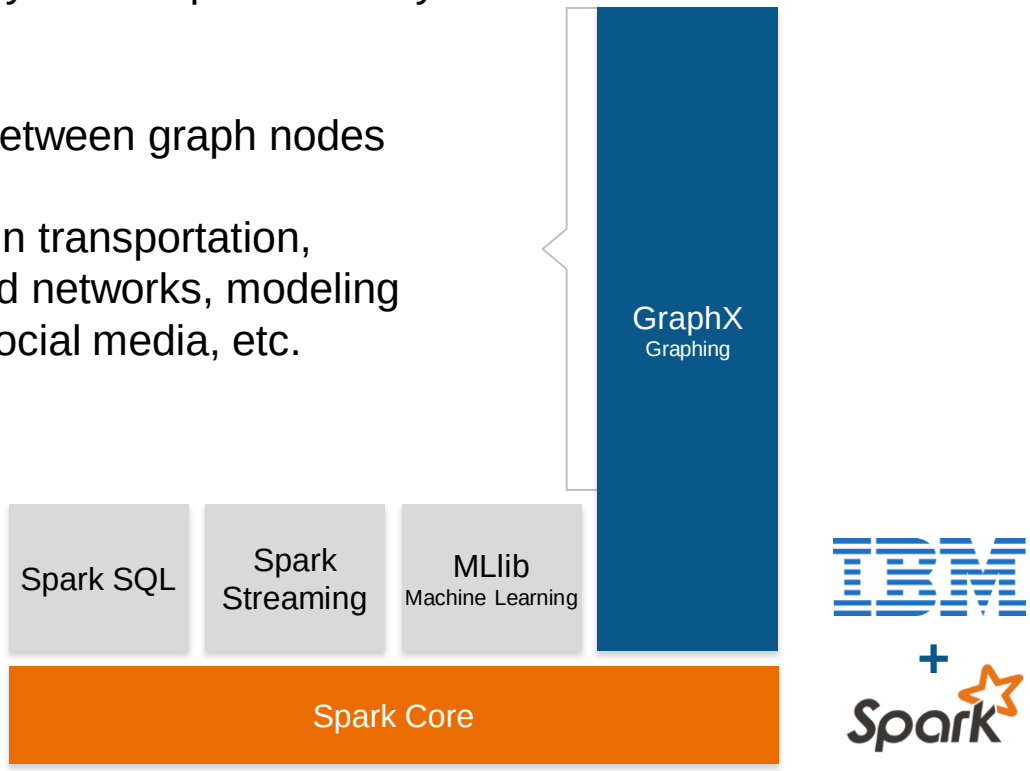


Spark MLlib

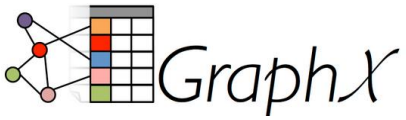
- **Spark MLlib for machine learning library**
  - Marked as under active development
- **Provides common algorithm and utilities**
  - Classification
  - Regression
  - Clustering
  - Collaborative filtering
  - Dimensionality reduction
- **Leverages iteration and yields better results than one-pass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

Closer Look at APIs – Graph DB

- Represent and analyze systems represented by graph nodes
- Trace interconnections between graph nodes
- Applicable to use cases in transportation, telecommunications, road networks, modeling personal relationships, social media, etc.



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