Introduction to Spark

Introduction

Spark vs. Hadoop

Installing Spark

Spark Shell

Licensed for personal use only for Fernando K <fernando_kruse@dell.com> from Machine Learning at Dell Brazil (QE) @ 2019-03-12

Lesson Objectives

- Understand the needs that Spark addresses
- Be familiar with Spark's capabilities and advantages
- Gain an understanding of a basic Spark installation

Licensed for personal use only for Fernando K <fernando_kruse@dell.com> from Machine Learning at Dell Brazil (QE) @ 2019-03-12

Introduction

➔ **Introduction**

Spark vs. Hadoop

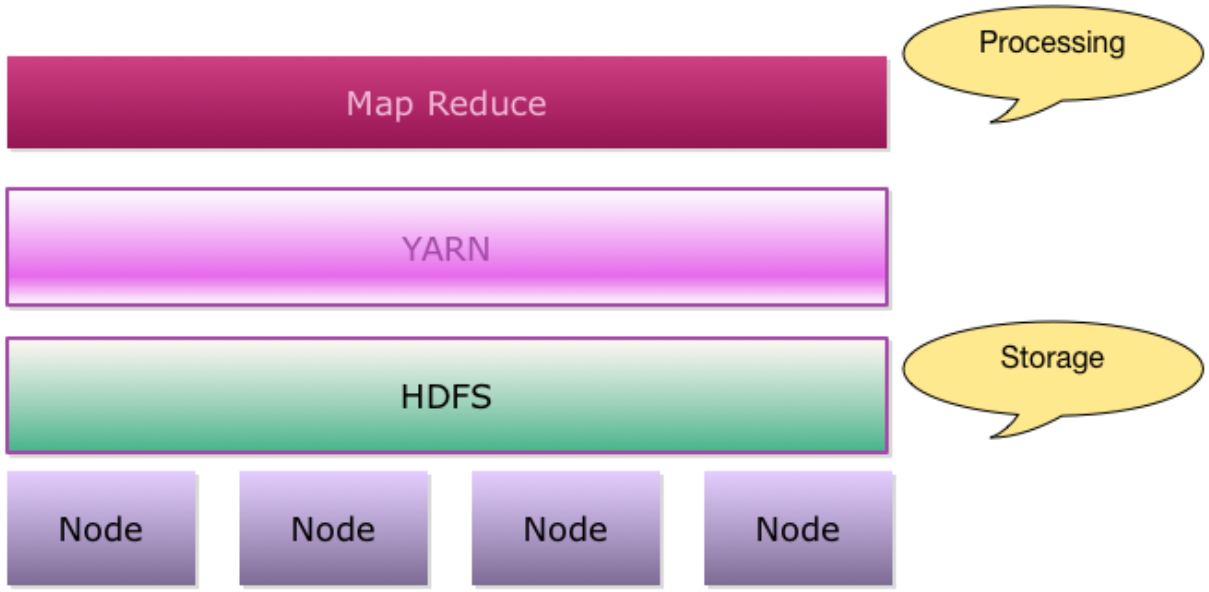
Installing Spark

Spark Shell

Licensed for personal use only for Fernando K <fernando_kruse@dell.com> from Machine Learning at Dell Brazil (QE) @ 2019-03-12

Big Data Evolution: Generation 1

- Hadoop became **the** Big Data platform to be widely adopted
- Storage: HDFS—still very good choice for large data sets
- Processing: MapReduce Engine
  - Has been proven at large scale
  - Batch processing



Big Data Evolution: Generation 2

- Processing needs have outpaced first-generation tools

- **MapReduce** (MR)/Hadoop has major limitations
  - MR **performance bottlenecks**
  - **Batch processing** doesn't fit needs
    - High latencies for small-to-medium datasets
    - Can’t process streaming
    - No in-memory processing
  - Programming can be **difficult and verbose**

- **Spark** is a second-generation tool addressing these needs

Spark Behind-the-Scenes

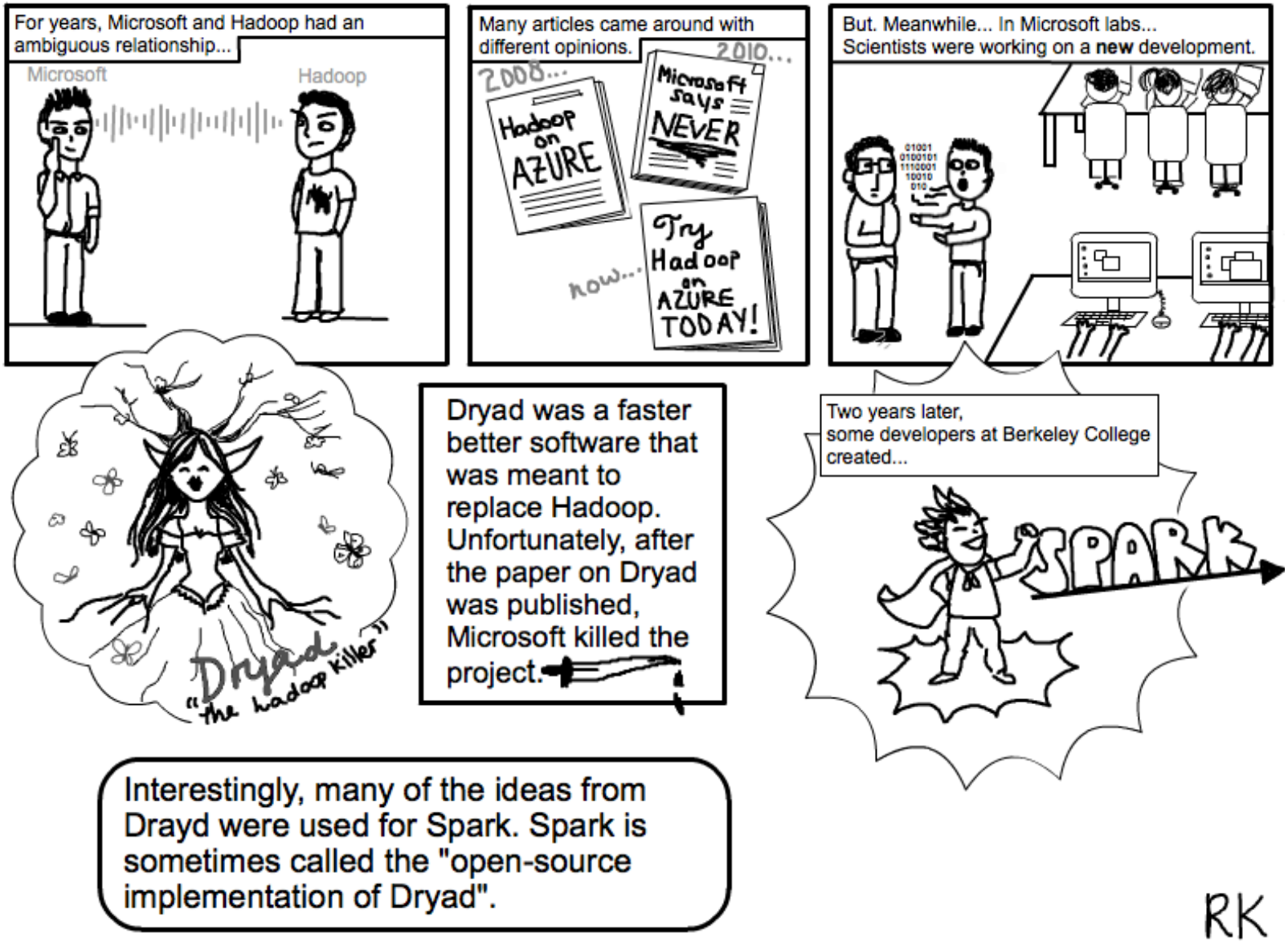


Image by Elephant Scale

What is Spark?

- Spark is an Open Source **cluster computing engine**
  - **Very fast**: In-memory ops 100x faster than MR
    - On-disk ops 10x faster than MR
  - **General purpose**: MR, SQL, streaming, machine learning, analytics
  - **Compatible**: Runs over Hadoop, Mesos, Yarn, or standalone
    - Works with HDFS, S3, Cassandra, HBase…
  - **Easier to code**: Word count in two lines
- **Spark's roots**:
  - Came out of Berkeley AMP Lab
  - Now top-level Apache project

*"First Big Data platform to integrate batch, streaming and interactive computations in **a unified framework.**"* – stratio.com

Spark Ecosystem

- **Apache** top-level project
  - Very active, fast-growing community
- **Databricks**: Supporting and developing Spark
  - Founded by Spark's creators
  - Employs the most active committers
- **Hadoop** vendors (Cloudera and Hortonworks)
  - Include Spark in their distributions
- **Spark packages** repository: Community index of Spark add-ons
  - http://spark-packages.org/

Licensed for personal use only for Fernando K <fernando_kruse@dell.com> from Machine Learning at Dell Brazil (QE) @ 2019-03-12

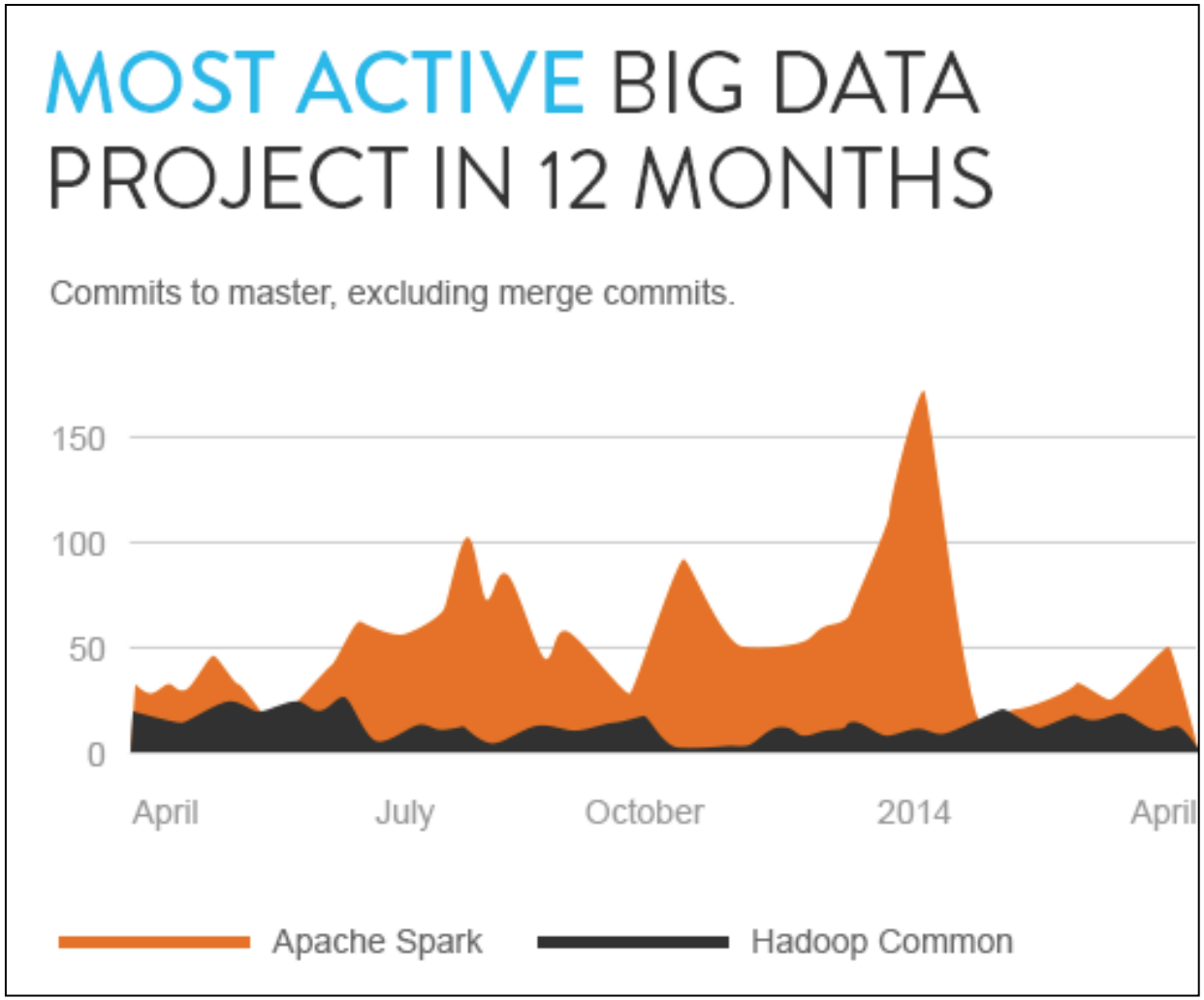
Why Spark Is Popular?

- Ease of use
  - Easy to get up and running
  - Develop on laptop, deploy on cluster
- Multiple language support
  - Java, Scala, Python and R
  - Developers (Java/Scala), Data Scientists (Python, R)
- High performant
- Plays nice with BigData eco system
- Out of the box functionality
  - Modern functional programming constructs
  - Machine Learning / Streaming / Graph processing

Used More and More

Source: stratio.com

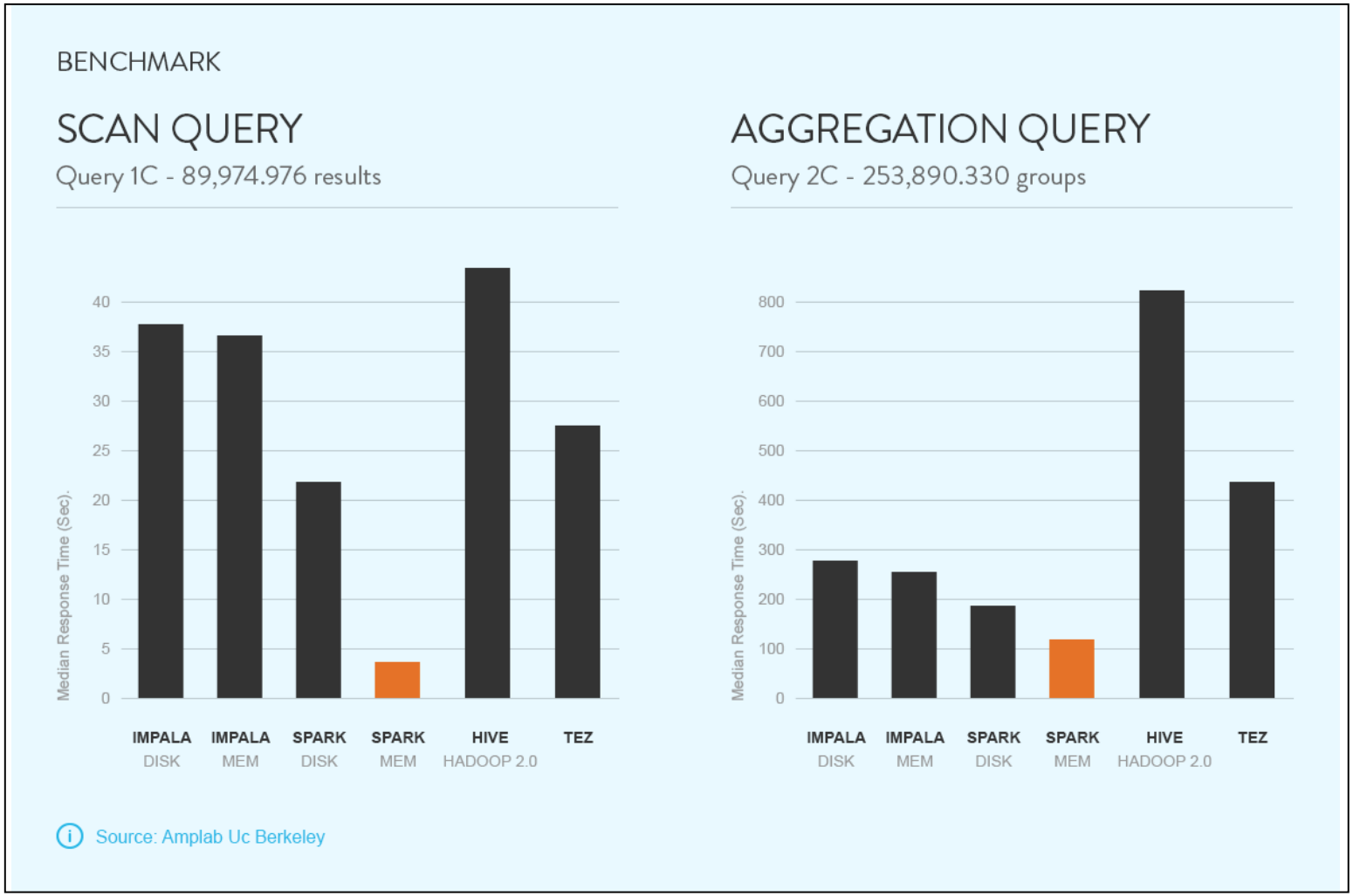
Used More and More



Source: stratio.com

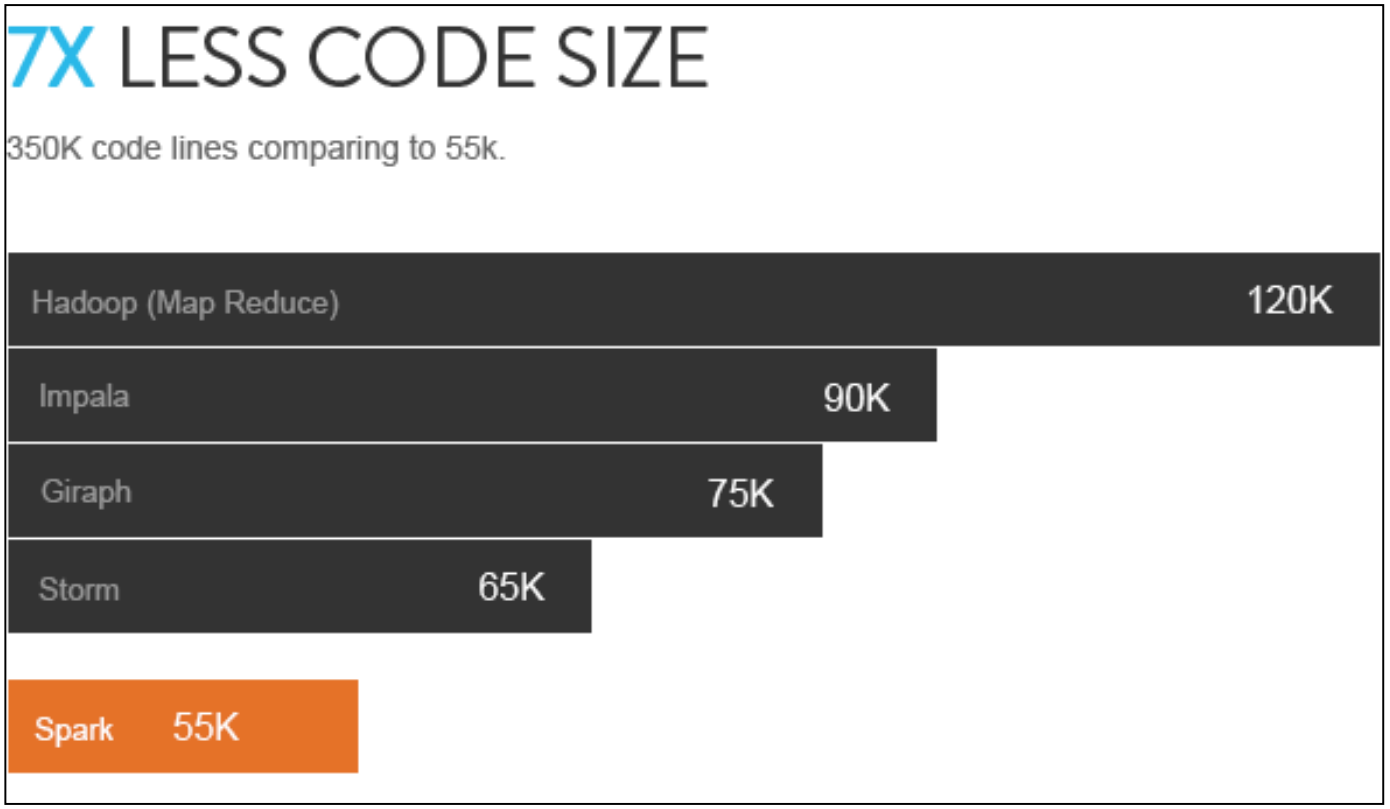
Licensed for personal use only for Fernando K <fernando_kruse@dell.com> from Machine Learning at Dell Brazil (QE) @ 2019-03-12

Faster Performance–Smaller Code Size



Source: stratio.com

Faster Performance – Smaller Code Size

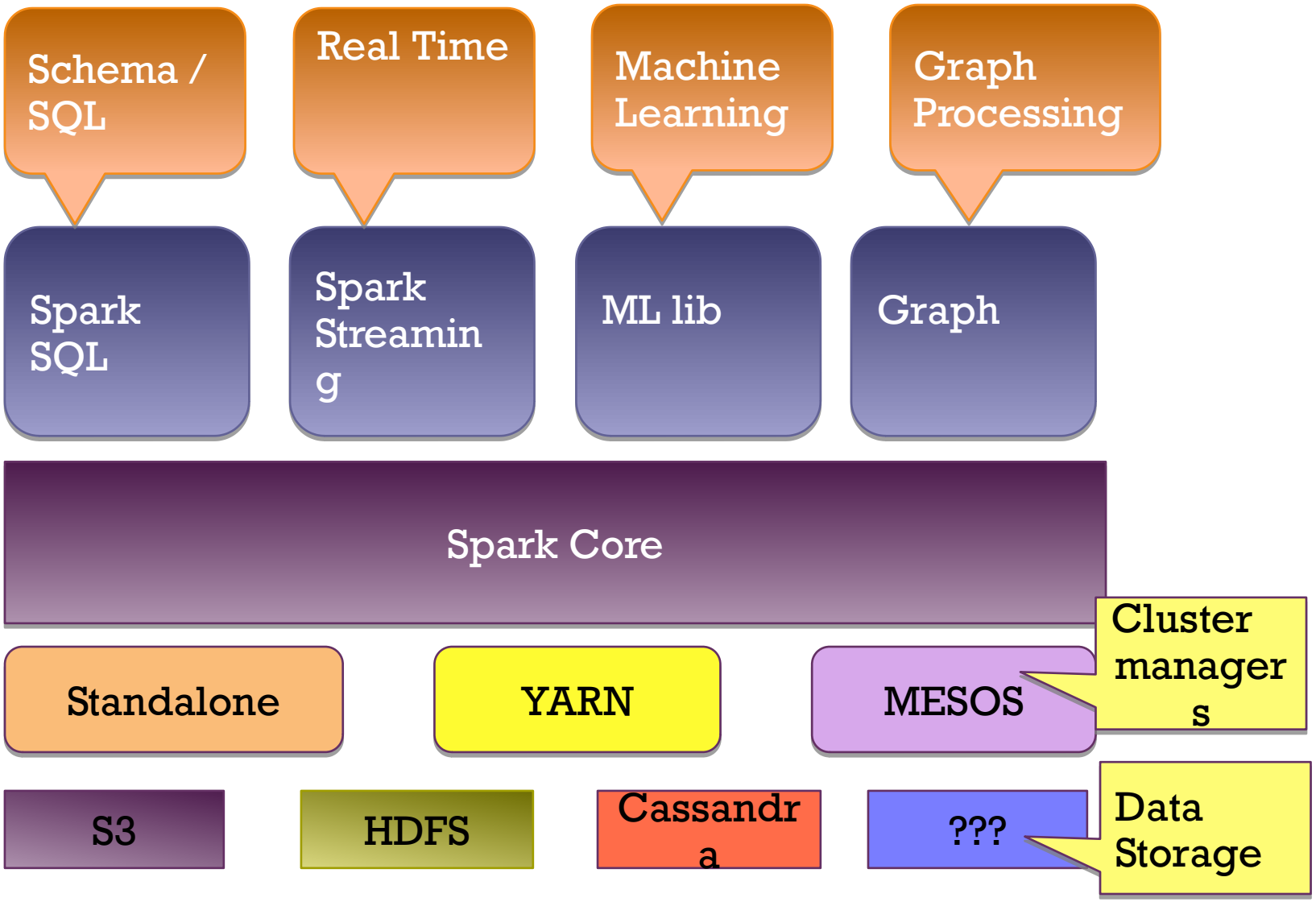


Source: stratio.com

Spark Noteworthy Versions

Version	Release Date	Noteworthy Features
1.0	5/30/2014	Initial release, Apache project
1.5	9/9/2015	- Dataframe/SQL support - R language support
1.6	1/4/16	- Stable v1 release
2.0	July 2016	- Major revision from v1 - Unifying dataFrame and dataset - Improved SQL with SQL2003 support - Dataframe-based ML - Structured streaming

Spark Illustrated



Spark Structure

- **Data Storage**: Pluggable data storage systems
  - Integrates with HDFS, S3, Cassandra DB, and more
- **Cluster Manager**: Manages distributed node clusters
  - Provides the distributed execution environment
  - Works with Mesos, Yarn, and its own standalone manager
- **Spark Core**: Distributed computing engine
- **Spark Components**: Modules layered on top of core
  - Specialized functionality, e.g., Spark SQL for SQL-based querying

Licensed for personal use only for Fernando K <fernando_kruse@dell.com> from Machine Learning at Dell Brazil (QE) @ 2019-03-12

Spark Core

- Basic building blocks for distributed computing engine
  - Task schedulers and memory management
  - Fault recovery (recovers missing pieces on node failure)
  - Storage system interfaces
- Defines Spark API and data model
- **Data Model: RDD/Dataframe/Dataset**
  - Distributed collection of items
  - Can be worked on in parallel
  - Easily created from many data sources (Any HDFS input source)
- **Spark API**: Scala, Python, and Java
  - Compact API for working with RDD and interacting with Spark
  - Much easier to use than MapReduce API

Licensed for personal use only for Fernando K <fernando_kruse@dell.com> from Machine Learning at Dell Brazil (QE) @ 2019-03-12

Spark Components

- **Spark SQL**: Structured data
  - Supports SQL and HQL (Hive Query Language)
  - Data sources include Hive tables, JSON, CSV, Parquet
- **Spark Streaming**: Live streams of data in real-time
  - Low latency, high throughput (1000s events per second)
  - Log files, stock ticks, sensor data, IOT (Internet of Things)
- **ML Lib**: Machine Learning **at scale**
  - Classification/regression, collaborative filtering…
  - Model evaluation and data import
- **GraphX**: Graph manipulation, graph-parallel computation
  - Social network friendships, link data
  - Graph manipulation, operations, and common algorithms

Spark: 'Unified' Stack

- Spark components support **multiple programming models**
  - MapReduce style batch processing
  - Streaming/real-time processing
  - Querying via SQL
  - Machine learning
  - Graph Processing

- All modules are **tightly integrated**
  - Facilitates rich applications

- Spark can be the only stack you need!
  - No need to run multiple clusters (Hadoop cluster, Storm cluster, etc.)

Licensed for personal use only for Fernando K <fernando_kruse@dell.com> from Machine Learning at Dell Brazil (QE) @ 2019-03-12

Spark Case Studies

Data Scientist	Data Engineer
- Ad hoc exploration - Analyzes data, build models - Comes up with “insights” (recommendations, etc.)	- Builds data infrastructure (pipelines, processing, etc.) - Produces ideas from Data Scientists (incorporates recommendations within online store)
Shells are great for ad-hoc analysis (Scala and Python)	Shells are good way to debug and test programs
Can re-use rich libraries in Scala, Java, Python and R	Can use already familiar languages
Interactive development (doesn’t wait minutes or hours for runs)	Can use already familiar languages
Unified stack (program within one stack)	Can use already familiar languages

Spark Case Study Examples

- Teralytics (Telco data)
  - Processing cell phone events
  - 180 billion events per day
  - Spark + HDFS
  - Estimating usage patterns to enhance coverage (sporting events, commuting, etc.)

- Spark at Yahoo
  - News personalization
  - 120 line Scala program with ML lib replaced 15,000 lines of C++
  - Spark took 30 minutes to run on 100 million samples

Spark Case Studies, Continued

- KeyGene (Genomics)
  - Uses Spark for Genome analytics
  - Millions of genome x: billions of combinations!
  - Migrating from HPC (High-Performance Computing) to Spark (HPC was error-prone, requiring a lot of babysitting, plus it didn't scale well)

- Netflix
  - Recommendations using Spark + Cassandra
  - Analyzes streaming events (450 billion events per day)
  - Personalization through recommendations

- Find more case studies @ BigDataUseCases.Info

Spark @ Large Scale

- Cluster
  - 8000 nodes
  - 400 TB+ data
  - At Tencent (Social network in China)

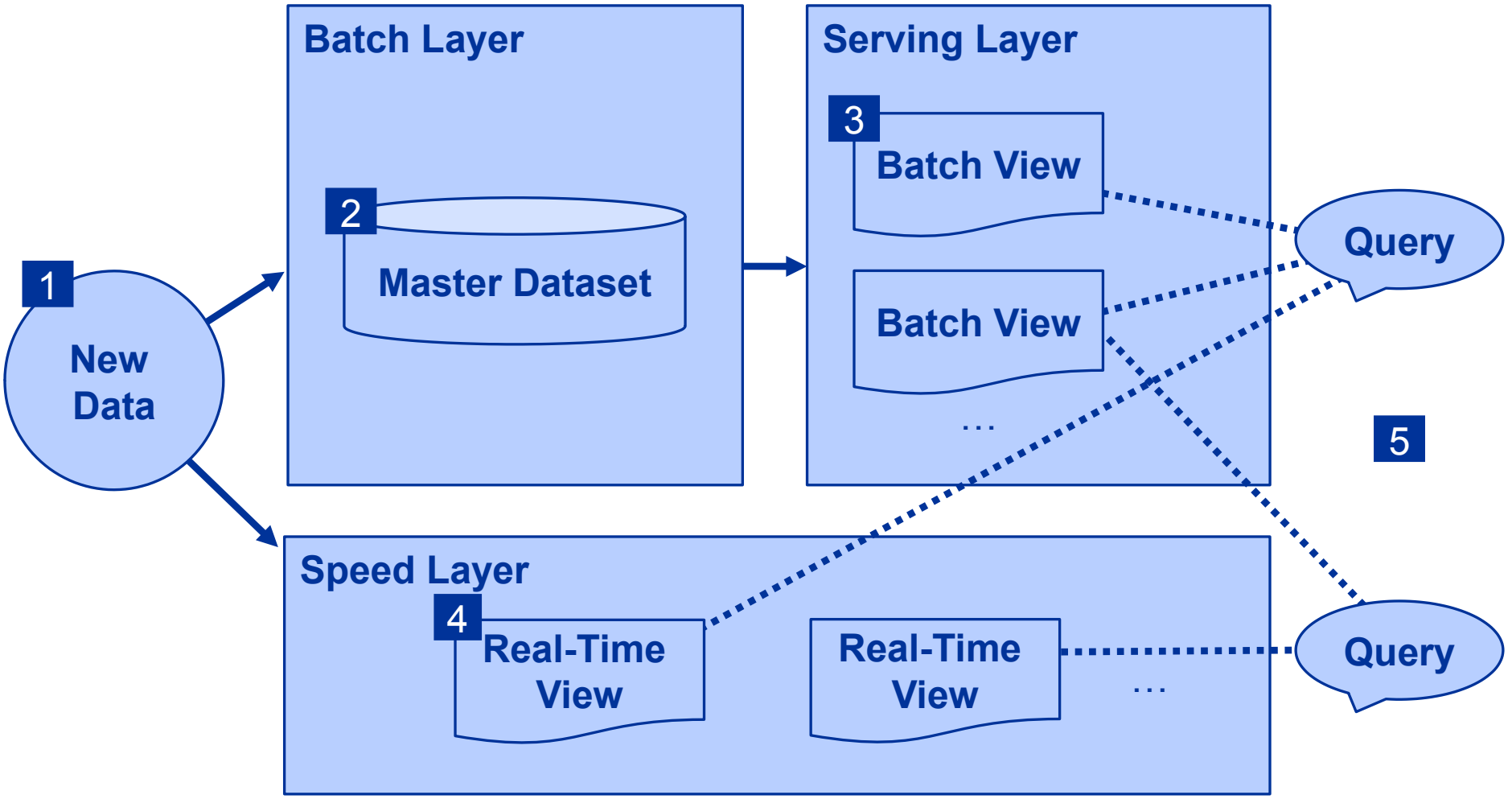
- Single job
  - 1 PB
  - Image processing at Alibaba

- Streaming
  - 1 TB per hour
  - Analyze medical images at Jenelia farm

Spark for Lambda Architectures (LA)

- LA: Design pattern for data infrastructure
  - Addresses needs of real-world, scalable applications
  - Came out of Twitter (Nathan Marz, et al.)
  - Key points below (Spark is a great fit for these)
- **Fault tolerant** to hardware failures and human error
- Layered approach incorporates batch and real-time needs
  - **Batch Layer**: Manages master data set
    - Immutable, append-only, raw data
    - Also pre-computes batch views
  - **Serving Layer**: Indexes batch views for low-latency queries
  - **Speed Layer**: Accommodates requests needing low latency
    - Recent data only using fast and incremental algorithms

Lambda Architecture Illustrated

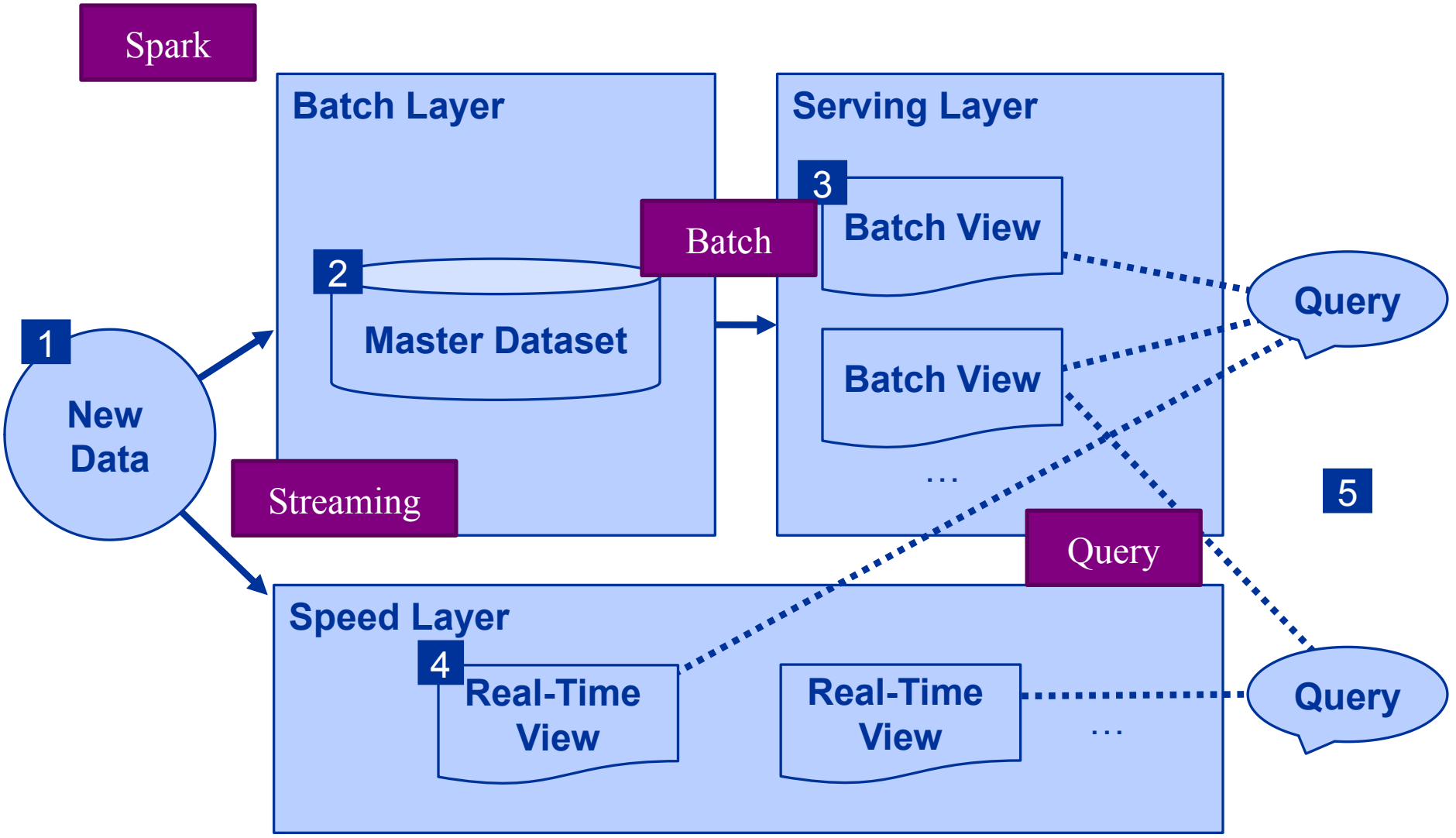


Source: http://lambda-architecture.net/

Spark and Lambda Architecture

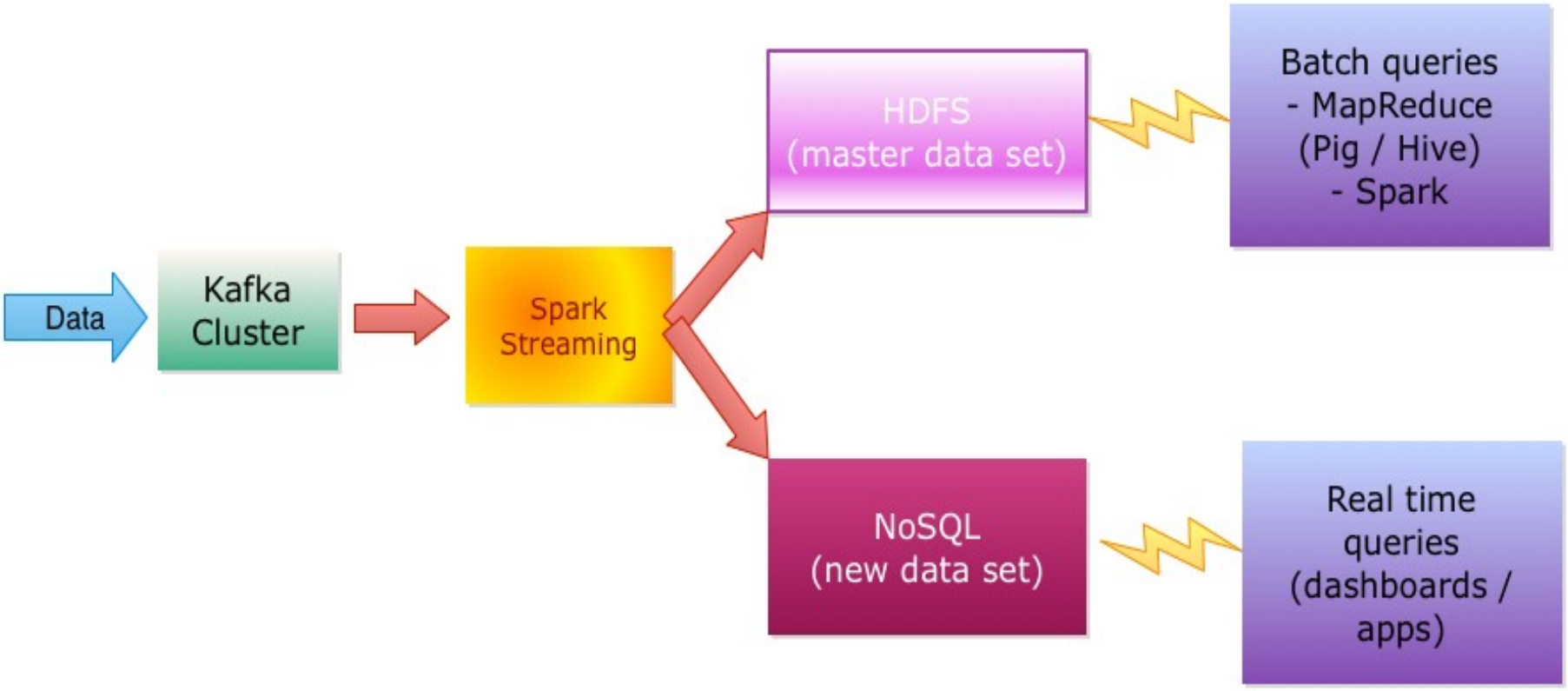
- LA requires both batch and real-time components
  - Challenging to architect—often require multiple systems
    - Multiple implementations and maintenance–Not good
  - **Until Spark**!
- Spark supports LA in a single system, e.g., mining log files
  - Uses core Spark functionality to process data (batch)
    - E.g., mining and storing log events with ERROR conditions
  - Create and store aggregates that contain useful views (batch)
    - E.g., number of errors in a day
  - Uses **Spark streaming** for real-time views of same data (speed)
- Spark is arguably the **best platform for LA**
  - Easily supports batch and stream in the same app **at scale**

Spark and Lambda Architecture



Source: http://lambda-architecture.net/

Example Lambda Architecture



Spark vs. Hadoop

Introduction

➔ **Spark vs. Hadoop**

Installing Spark

Spark Shell

Licensed for personal use only for Fernando K <fernando_kruse@dell.com> from Machine Learning at Dell Brazil (QE) @ 2019-03-12

Hadoop and Spark Timelines

Hadoop	Year	Spark
Created	2006	
	2009	Begins in AMP Lab
	2010	Open sourced. Spark paper
Version 1.0	2011	
Version 2.0	2013	
	2014	Version 1.0 Apache top-level project
	2015	Wide support by Hadoop vendors
Version 3.0	2016	Version 2.0

Licensed for personal use only for Fernando K <fernando_kruse@dell.com> from Machine Learning at Dell Brazil (QE) @ 2019-03-12

Hadoop vs. Spark

- Video

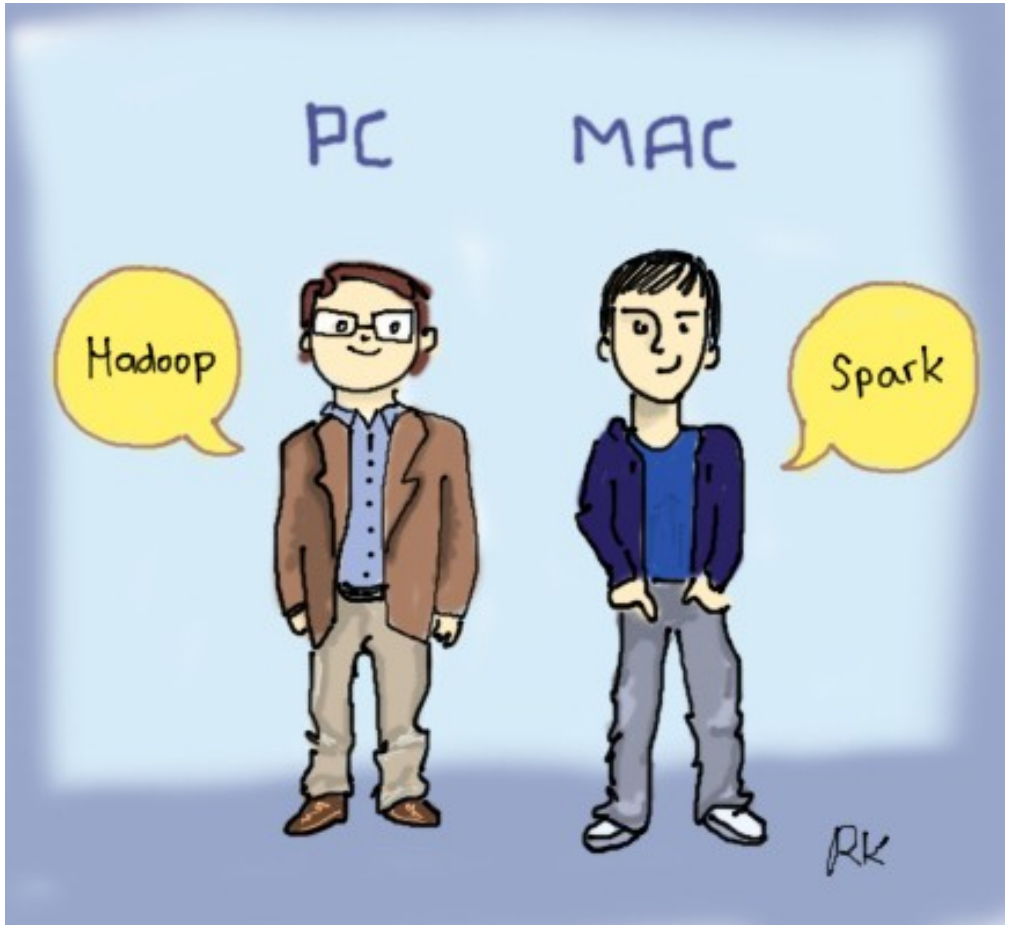


Image by Elephant Scale

Licensed for personal use only for Fernando K <fernando_kruse@dell.com> from Machine Learning at Dell Brazil (QE) @ 2019-03-12

Spark code is simpler – MapReduce Vs. Spark

Hadoop

```
@Override
public int run(String[] args) throws Exception {
    System.out.println("Running WordCount");
    Job job = new Job(getConf());
    job.setJarByClass(WordCount.class);
    job.setJobName("WordCount");

    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);

    job.setMapperClass(Map.class);
    job.setCombinerClass(Reduce.class);
    job.setReducerClass(Reduce.class);

    job.setInputFormatClass(TextInputFormat.class);
    job.setOutputFormatClass(TextOutputFormat.class);

    System.out.println("Input path: " + args[0]);
    System.out.println("Output path: " + args[1]);
    FileInputFormat.setInputPaths(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));

    boolean success = job.waitForCompletion(true);
    return success ? 0 : 1;
}

public static void main(String[] args) throws Exception {
    int ret = ToolRunner.run(new WordCount(), args);
    if (ret != 0) {
        System.exit(ret);
    }
}
```

```
public static class Map
        extends Mapper<LongWritable, Text, Text, IntWritable> {

    private static IntWritable ONE = new IntWritable(1);

    @Override
    public void map(LongWritable key, Text value, Context context)
            throws IOException, InterruptedException {
        String line = value.toString();
        String[] words = line.split("\\W");
        for (String word : words) {
            if (word.trim().length() > 0) {
                Text text = new Text();
                text.set(word);
                context.write(text, ONE);
            }
        }
    }
}

public static class Reduce
        extends Reducer<Text, IntWritable, Text, IntWritable> {

    @Override
    public void reduce(
            Text key, Iterable<IntWritable> values, Context cont
            throws IOException, InterruptedException {
        int sum = 0;
        for (IntWritable val : values) {
            sum += val.get();
        }
        context.write(key, new IntWritable(sum));
    }
}
```

Spark

```
val wordcount = r.flatMap(lines =>
lines.split(" ")).map(word => (word,
1)).reduceByKey(_+_)
```

Comparison with Hadoop

Hadoop	Spark
Distributed storage and distributed computing	Distributed computing only
MapReduce framework	Generalized computation
Usually data on disk (HDFS)	On disk/in memory (Tachyon)
MR Not ideal for iterative work	Great at iterative workloads (machine learning, etc.)
MR Batch process	- 2-10x faster for data on disk - 100x faster for data in memory
Java supported fully Other language support possible	Compact code Java, Python, Scala supported
No equivalent of shell	Shell for ad-hoc exploration

Licensed for personal use only for Fernando K <fernando_kruse@dell.com> from Machine Learning at Dell Brazil (QE) @ 2019-03-12

Spark vs. MapReduce

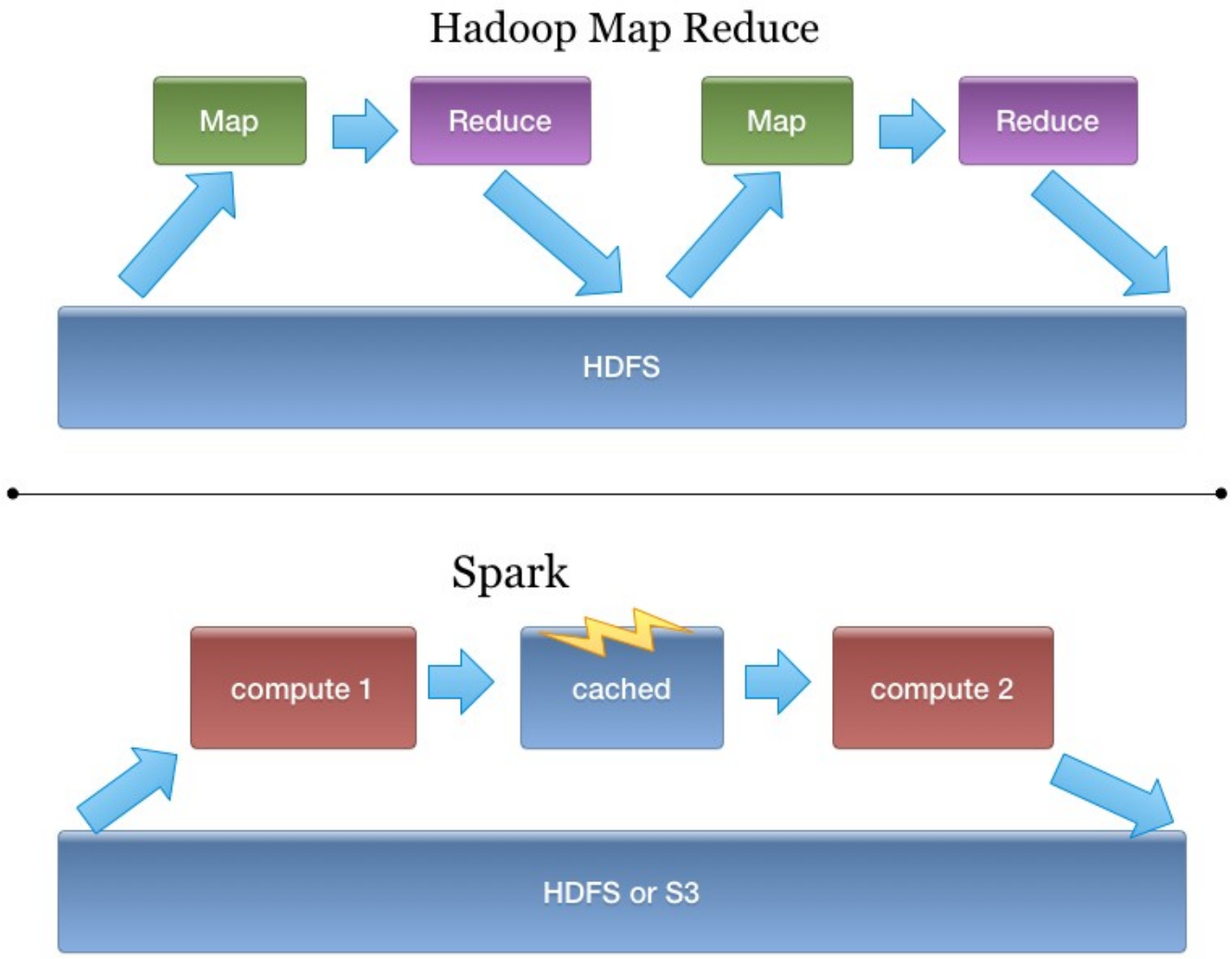
- Spark is **easier** than MapReduce
  - Simpler code, less code
- **Friendlier** for data scientists and analysts
  - Interactive shell
    - Fast development cycles
    - Ad hoc exploration
  - Specialized components (GraphX, machine learning,…)
- API supports multiple languages
  - Java, Scala, Python
- Compared to MR, Spark is a better fit for small (GBs) to medium (100s of GBs) data

Spark–World Record Large-Scale Sort

- Sorted 100 TB in 23 minutes on 206 nodes
  - Won Daytona GraySort 2014 contest
  - Best Hadoop record: 72 minutes on 2100 nodes

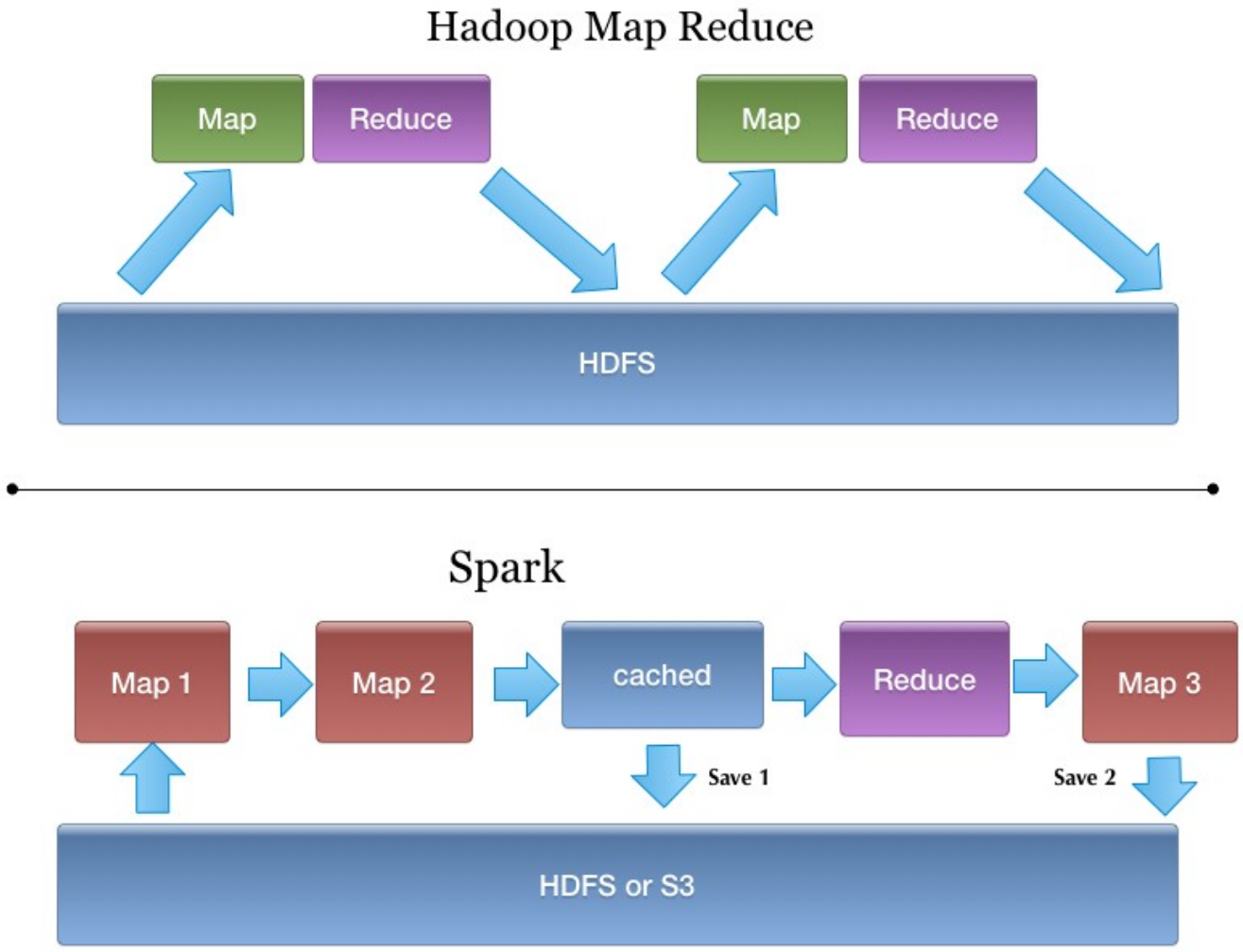
	Hadoop MR Record	Spark Record	Spark 1 PB
Data Size	102.5 TB	100 TB	1000 TB
Elapsed Time	72 mins	23 mins	234 mins
# Nodes	2100	206	190
# Cores	50400 physical	6592 virtualized	6080 virtualized
Cluster disk throughput	3150 GB/s (est.)	618 GB/s	570 GB/s
Sort Benchmark Daytona Rules	Yes	Yes	No
Network	dedicated data center, 10Gbps	virtualized (EC2) 10Gbps network	virtualized (EC2) 10Gbps network
Sort rate	**1.42 TB/min**	**4.27 TB/min**	**4.27 TB/min**
Sort rate/node	**0.67 GB/min**	**20.7 GB/min**	**22.5 GB/min**

Spark is a Better Fit for Iterative Workloads



Spark is a More Generic Programming Model

- Reduces dependency on the disk



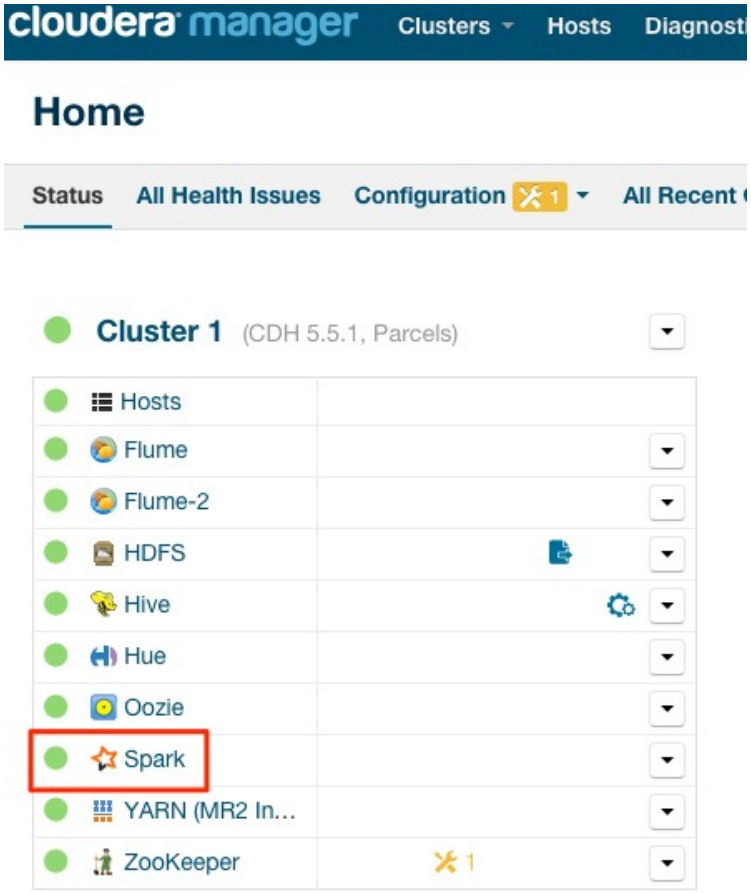
Hadoop vs. Spark

- Spark's unified stack can support almost all needs.

Use Case	Hadoop	Spark
Storage	HDFS	• HDFS/S3/Cassandra for now • Tachyon (in-memory fs)
Cluster Manager	YARN	Standalone, YARN or Mesos
Batch processing	MapReduce (Java, Pig, Hive)	Spark MR
SQL querying	Hive	Spark SQL (can query Hive too)
Stream Processing/ Real-time processing	Storm	Spark Streaming
Machine Learning	Mahout	Spark MLlib
Real-time lookups	NoSQL (HBase, Cassandra, etc.)	No Spark component, but Spark can query data in NoSQL stores

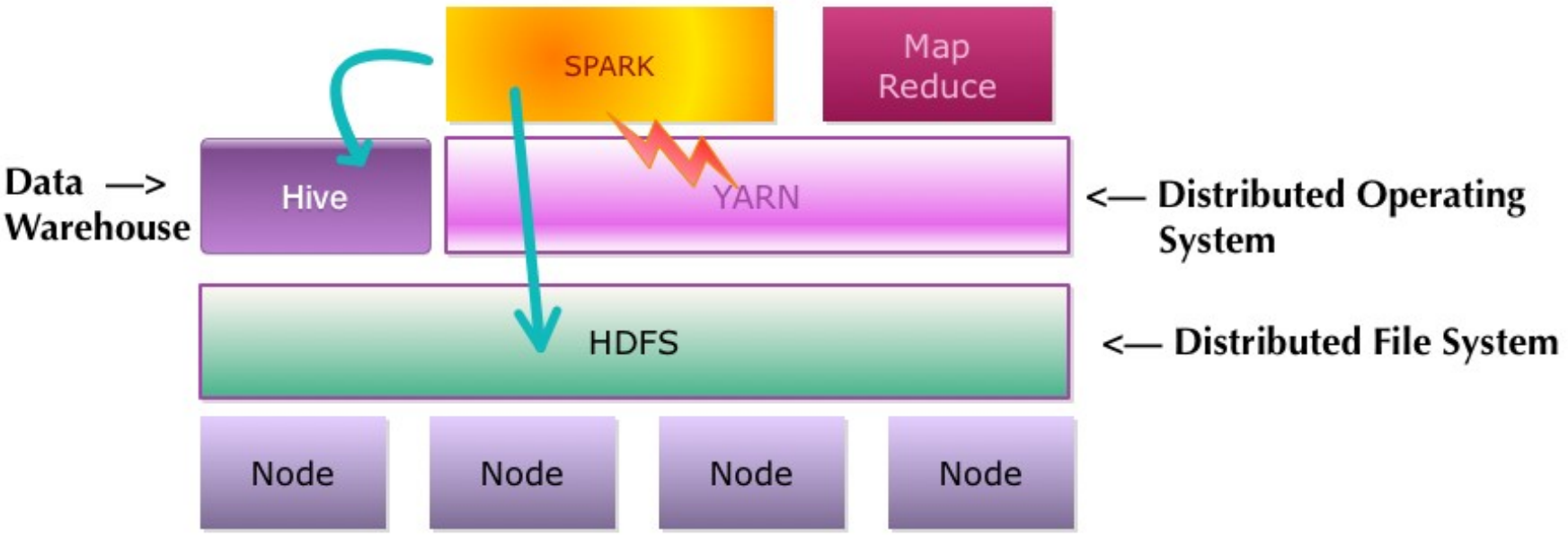
Spark & Hadoop

- Spark works very well with Hadoop eco system
- Hadoop vendors are now offering fully integrated Spark with Hadoop stack
- Spark is pretty complimentary to Hadoop
- And take advantage of existing Hadoop installations



Spark & Hadoop Integration

- Spark replaces / offers better alternative to Map Reduce
- YARN : For clustering & scheduling
- HDFS : Distributed data storage – stable, scalable
- Hive : Spark can query Hive tables directly
- Pig : Run Pig on Spark using 'Spork' project



Installing Spark

Introduction

Spark vs. Hadoop

→ **Installing Spark**

Spark Shell

Licensed for personal use only for Fernando K <fernando_kruse@dell.com> from Machine Learning at Dell Brazil (QE) @ 2019-03-12

Apache Spark Project/Download

- Spark is a top-level Apache, Open Source project
  - http://spark.apache.org/
- Written in Scala
  - Runs on the Java Virtual Machine (JVM)
- Binary tarballs available on the project website
  - Contains the Spark and Scala libraries, interactive shells, run scripts, and one of several supported Hadoop distributions
- Can be used in two mediums:
  - Standalone: We'll start with this first
  - As part of Hadoop stack (Cloudera/Hortonworks)

Licensed for personal use only for Fernando K <fernando_kruse@dell.com> from Machine Learning at Dell Brazil (QE) @ 2019-03-12

Spark Version To Choose

- Version 2 is highly recommended
- Lots of new APIs
- Tungsten core
  - Better memory management
  - Code generation
- Catalyst optimizer
  - Optimize SQL queries
- Structured streaming
  - Streaming structured data

Licensed for personal use only for Fernando K <fernando_kruse@dell.com> from Machine Learning at Dell Brazil (QE) @ 2019-03-12

Spark Releases Timeline - V2

Version	Date	Noteworthy
2.0	2016 July	- Core performance improvements - Streaming: Flow control (backpressure) - ML: Improvements - SparkR: More R integration
2.2	2017 July	- Structured streaming

Licensed for personal use only for Fernando K <fernando_kruse@dell.com> from Machine Learning at Dell Brazil (QE) @ 2019-03-12

Reference Only : Spark Releases Timeline - V1

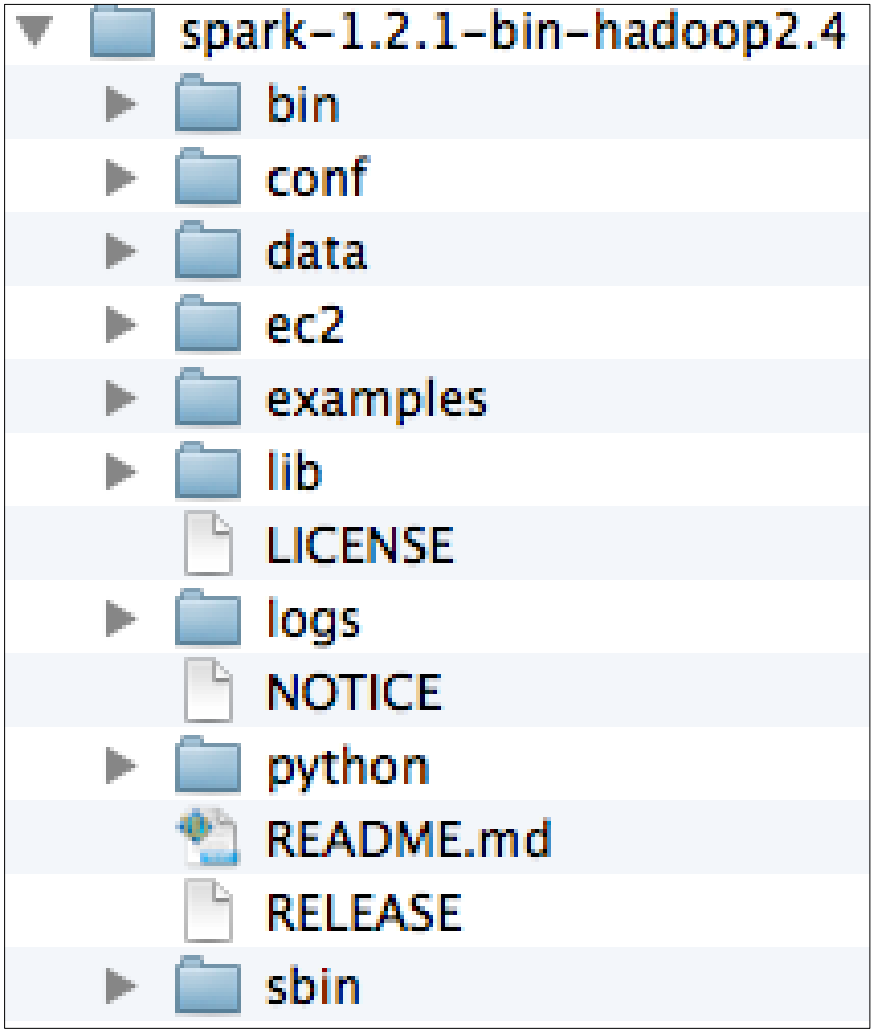
Version	Date	Noteworthy
1.3	2015 Mar	- Data frames (easy data access) - Streaming: Kafka direct access (2x faster than previously) - MLlib: Lots of new algorithms
1.4	2015 June	- SparkR: R & Spark! - Machine-learning pipelines - Enhancements to DataFrames API - Streaming improvement for Kafka and Kinesis - DAG visualizing tool
1.5	2015 November	- Core performance improvements - Streaming: Flow control (backpressure) - ML: Improvements - SparkR: More R integration
1.6	2016 January	- Stable 1.x release

System Requirements

- Needs JDK (Java Development Kit) v8 or latest
- Operating System
  - Development : Linux, Mac, Windows (maybe?)
  - Production : Linux
- Hardware

Resource	Development	Production
CPU	2 core	8 core +
Memory	4G+	128 G +
Disk	Single spindle Few Gigs	- Multiple spindles - Several Terabytes / node - Similar to Hadoop Data nodes

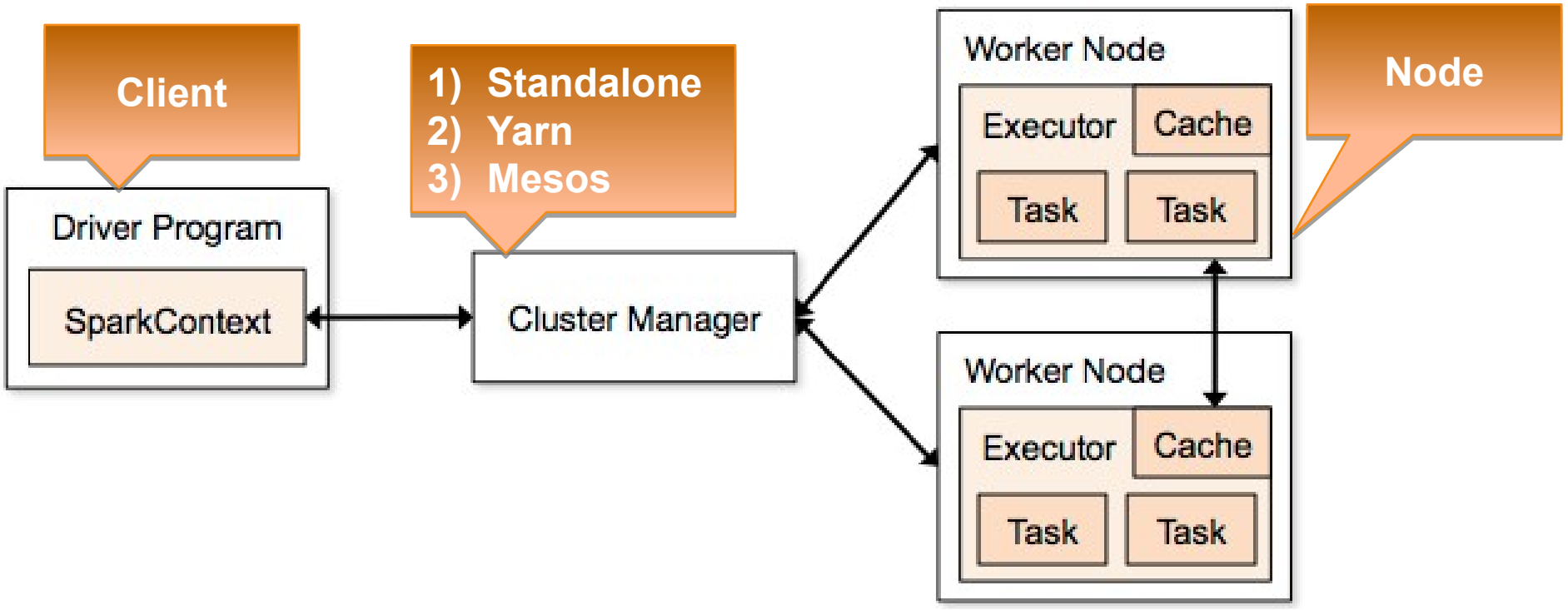
Spark Installation Files Layout



- **bin**: Executables (Spark/Python shells, utilities, etc.)
- **conf**: Configuration templates (e.g., *spark-env.sh.template*)
- **data**: Sample data files
- **ec2**: Scripts/files run on EC2
- **examples**: Example programs (Scala, Python, Java)
- **lib**: Jar files
- **logs**: Log files
- **python**: Python source
- **sbin**: Shell scripts

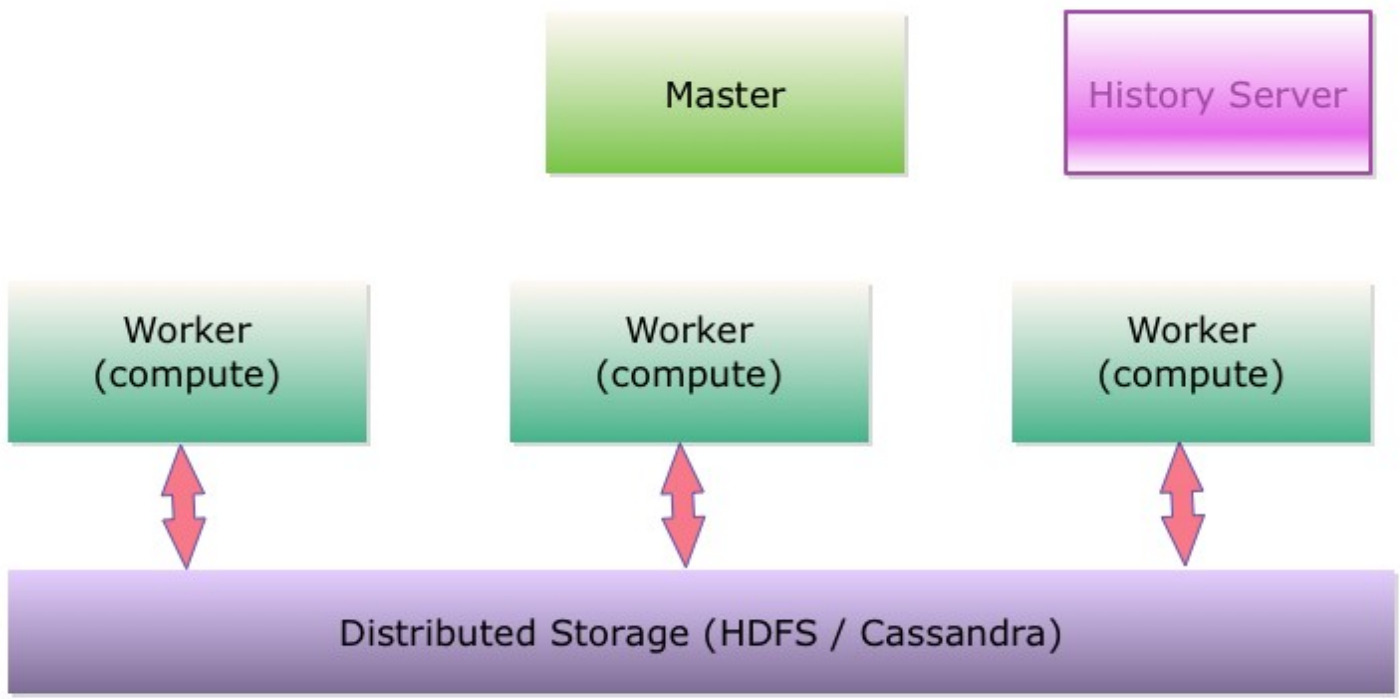
Runtime Architecture

- **Cluster of worker nodes** performs work
- **Cluster manager** manages the worker nodes
- **Driver program** kicks off tasks and sends code to them



Spark Runtime Roles

- Worker nodes carry out execution of tasks
- Master manages workers and handles failures
- History server keeps run time metrics for later review



Licensed for personal use only for Fernando K <fernando_kruse@dell.com> from Machine Learning at Dell Brazil (QE) @ 2019-03-12

Spark Runtime Roles

- **Master**
  - Coordinates the cluster
  - Allocates resources
  - Re-assigns failed tasks to another worker
  - Failure is critical (there is usually a backup master)
- **Worker**
  - Work horses who run the tasks
  - Failure is not critical; tasks can be re-assigned to other workers
- **History Server**
  - Stores metrics and run-time stats, such as execution time and memory usage, for later retrieval
  - Thinks like a time machine; can go back and look at job performance metrics

Running Spark

We'll start with two simple ways to run Spark.

1. Use the **standalone cluster manager** to start a cluster
   - And start all nodes on one machine
   - *<spark>/sbin/**start-all.sh*** starts up a small cluster (a master and one worker); *stop-all.sh* stops them
   - May require a bit of setup for ssh access, even on one machine

2. Use an **interactive shell**
   - Ships with Scala and Python shells
   - **Scala**: *<spark>/bin/spark-shell*
   - **Python**: *<spark>/bin/pyspark*
   - By default, these use an embedded Spark instance

Standalone Cluster Manager

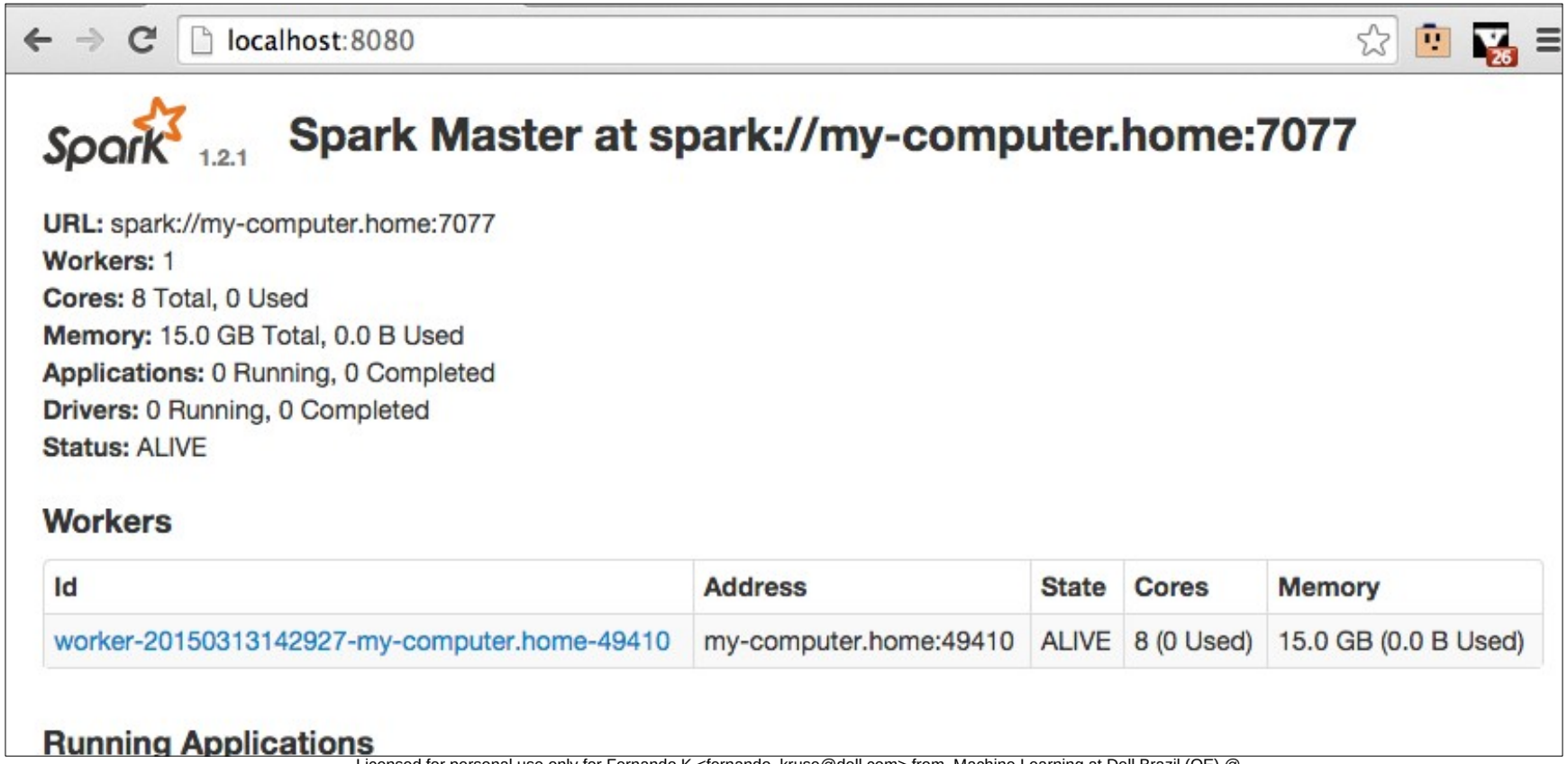
- Simple standalone cluster support
  - No need for other managers (e.g., YARN)
  - Easy setup/startup for development
- Easy startup via provided launch scripts
  - *<spark>/sbin/**start-all.sh*** starts a manager (master) and workers
    - Default: One worker, one slave on local machine
    - Or based on configuration files (next slide)
    - Provides master Web UI at hostname:8080
- Other launch scripts include
  - ***start-master.sh***: Starts master on this machine
  - ***start-slaves.sh***: Starts workers on machines listed in *conf/slaves*
  - ***stop-*.sh***: Variations to stop master, slaves, or all

Spark Configuration

- *conf/**log4j.properties***: Logging configuration
  - *log4j.properties.template* is starter file
- *conf/**slaves*** file: List of worker hosts (default localhost)
  - *slaves.template* is starter file
- *conf/**spark-env.sh***: Overall configuration
  - IP addresses, ports, memory, cores, etc.
  - *spark-env.sh.template* starter file has complete description
  - For example, the standalone configuration includes:
    - **SPARK_MASTER_IP / PORT / WEBUI_PORT:** master connectivity
    - **SPARK_WORKER_CORES / INSTANCES**: worker resources
    - And much more

Starting Spark and Viewing UI

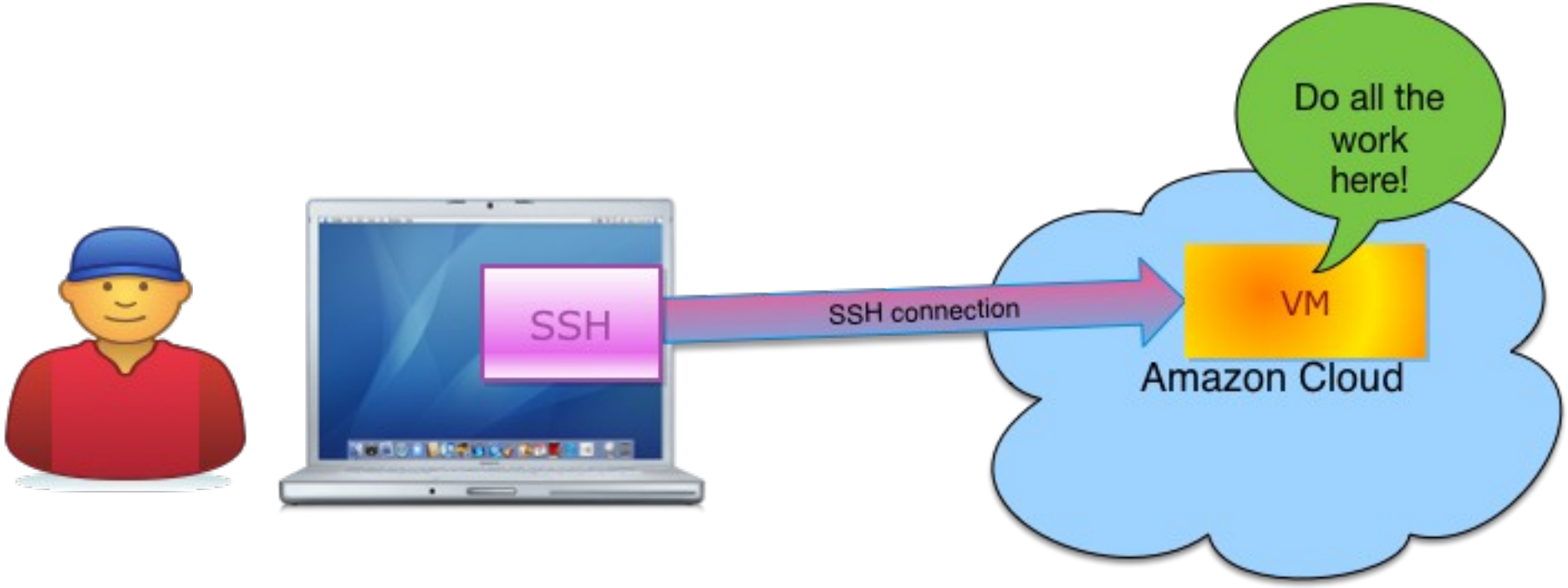
```
$ ./sbin/start-all.sh

starting org.apache.spark.deploy.master.Master
...
starting org.apache.spark.deploy.worker.Worker
```



Lab Setup

Lab

- Login to VM running in the cloud
- **To Instructor:**
  - Distribute VM details
  - Make sure students can access **web UI** and **SSH** in



Lab Prep

- **Note**:
  If using Hadoop, see instructions in next slide.

- **Instructions for the instructor**:
  - Provide a zip bundle of lab files to students
  - **Scala version**
    Walk through setting up '**mark down preview plus**' plugin in **Chrome** browser to view markdown files
  - **Python version**
    Explain the **ipynb** and **html** files in the lab bundle

Lab (Scala) : First Look at Spark

- **Note**:
  If using Hadoop, see instructions in next slide.
  If using Python, see instructions in next slide.

- **Overview**:
  In this lab, we will become familiar with the lab environment, set up Spark, and start a Spark cluster

- **Approximate time**:
  15-20 minutes

- **Instructions for students**:
  - Follow **2-intro/2.1-install-spark.md** file

Lab (Python): Setup Spark & Jupyter

- **Overview**:
  Setup Spark + Jupyter

- **Approximate time**:
  15-20 minutes

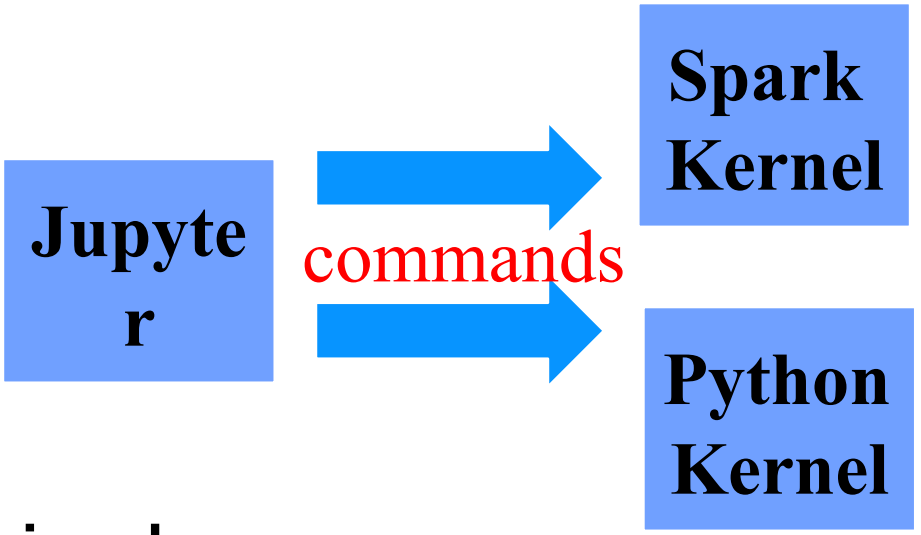
- **Instructions for students (Execute in the following order)**
  - **Spark-labs/README-Python.html**

Lab (Python): Jupyter Primer

Lab

- **Overview**:
  Learn Jupyter

- **Approximate time**:
  5-10 minutes



- **Instructions for students**
  - Spark-labs/spark-python.ipynb
  - 1.1 – Hello Jupyter
  - 1.2 – Testing 123

- **Instructions for Instructor**
  - Explain Jupyter commands
  - Explain Jupyter architecture

Lab (Python): Start Spark Master

- **Overview**:
  Start Spark master

- **Approximate time**:
  15-20 minutes

- **Instructions for students**
  - **2.1-- Run-spark**

Lab : First Look at Spark (Hadoop)

- Most modern Hadoop environments will have Spark already installed.

- **Overview**:
  In this lab, we will examine Spark setup on Hadoop cluster.

- **Approximate time**:
  5-10 minutes

- **Instructions for students**:
  - In the Hadoop UI, find Spark service
  - Inspect configuration

Spark Shell

Introduction

Spark vs. Hadoop

Installing Spark

➔ **Spark Shell**

Licensed for personal use only for Fernando K <fernando_kruse@dell.com> from Machine Learning at Dell Brazil (QE) @ 2019-03-12

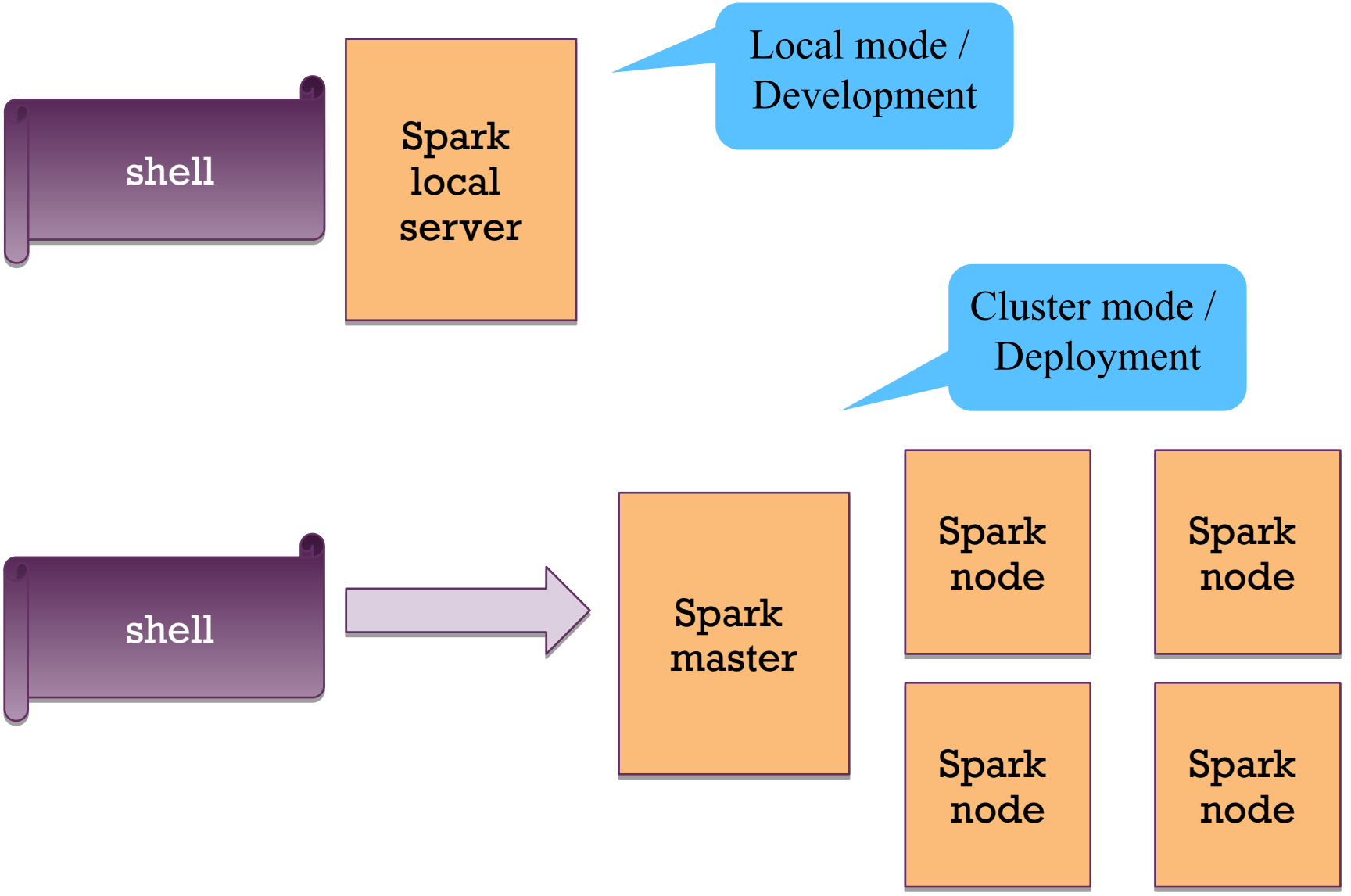
Spark Shells

- **Interactive** shells support ad-hoc operations
  - Ad-hoc queries can access large clusters for fast response
    - They have full access to all capabilities
  - Used extensively in the course
    - Standalone apps covered later

- **Scala** shell: `bin/spark-shell`
- **Python** shell: `bin/pyspark`

- Shells run in one of:
  - **Local** (pseudo) mode: Uses embedded, in-process spark server (*not* the same as using the standalone manager)
  - **Cluster** mode: Connect to cluster via URL of master

Shell Execution Modes



Spark Shell Startup Examples

- **`spark-shell`**: Startup using embedded server and one thread
  - The default
- **`spark-shell --master local[4]`**: Startup using embedded server and four worker threads
  - Limited to the number of cores you have
- **`spark-shell --master spark://myhost:7077`**
  - Connect to Spark cluster with master at URL above
  - Assumes standalone cluster running on `myhost`
- **`spark-shell --help`**: Show help

PySpark Startup Examples (Python)

- **`pyspark`**: Startup using embedded server and one thread
  - The default
- **`pyspark --master local[4]`**: Startup using embedded server and four worker threads
  - Limited to the number of cores you have
- **`pyspark --master spark://my-computer.home:7077`**
  - Connect to Spark cluster with master at URL above
  - Assumes standalone cluster running on `my-computer.home`
- **`pyspark --help`**: Show help

Master URL Options

Master URL	Details
Local	Run Spark locally on one thread. No parallelism.
Local[k]	Run Spark locally with K worker threads–which should be less than or equal to the number of cores on your machine.
Local[*]	When you don't know how many cores are on your machine, you can use a wild card.
Spark://HOST:PORT	Connect to Spark standalone cluster master. 7077 is default.
Mesos://HOST:PORT	Connect to Mesos cluster. 5050 is default.
yarn	Connect to a YARN cluster - in client mode: --deploy-mode cluster - in cluster mode: --deploy-mode cluster

Starting Scala Spark Shell (Local Mode)

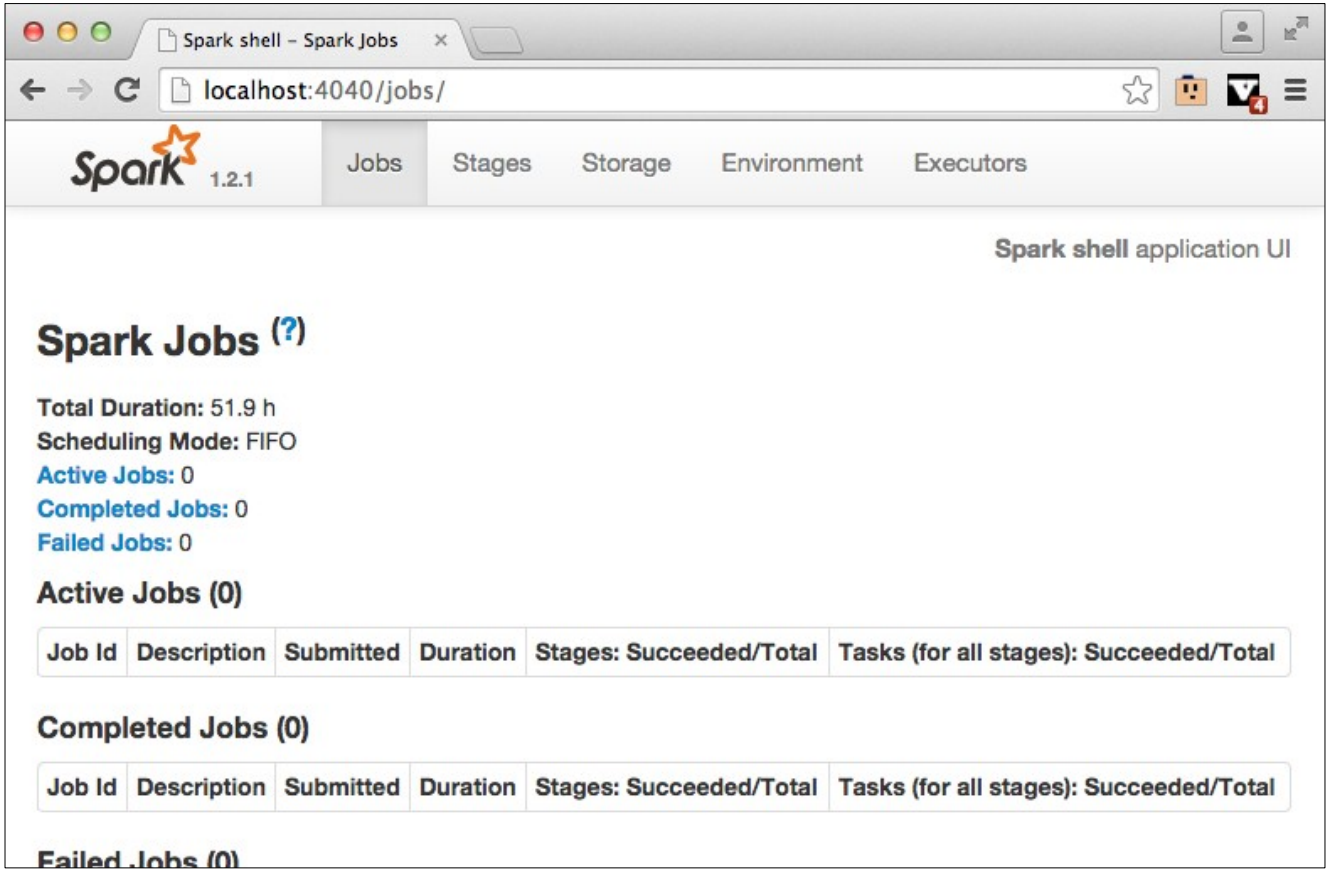
```
$   spark/bin/spark-shell

Spark context Web UI available at http://172.16.0.27:4040
Spark context available as 'sc' (master = local[*], app id =
local-1511895652460).
Spark session available as 'spark'.

Welcome to Spark version 2.2.0

Using Scala version 2.11.8 (Java HotSpot(TM) 64-Bit Server
VM, Java 1.8.0_77)

Type in expressions to have them evaluated.
Type :help for more information.

scala>
```

Starting Python Spark Shell (Local Mode)

```
$   spark/bin/pyspark

Welcome to Spark version 2.2.0

Using Python version 3.6.2 (default, Jul 20 2017 13:51:32)

SparkSession available as 'spark'.

>>>
```

Spark Shell UI

- Web-based access at **http://<shell-host>:4040**
  - This UI is built into the `SparkContext`
  - Provides information about driver program and Spark jobs



SparkContext and SparkSession

Within Spark Shell, there are two ways to access Spark

1. Spark Context (sc)
   - Original, been there since the start
2. Spark Session
   - New in Spark v2.0
   - Unifies ‘SparkContext’ and ‘SQLContext’

```
// scala
scala> sc
org.apache.spark.SparkContext = org.apache.spark.SparkContext@2c01a0dd

scala> spark
org.apache.spark.sql.SparkSession = org.apache.spark.sql.SparkSession@7674f9d4
```

```
# python
>>> sc
<SparkContext master=local[*] appName=PySparkShell>

>>> spark
<pyspark.sql.session.SparkSession object at 0x7f45bdfcac88>
```

SparkSession: Loading Data - V2+ (Scala)

```
# load file
scala> val myfile= spark.read.textFile("README.md")
myfile: org.apache.spark.sql.Dataset[String] = [value: string]

scala> val first = myfile.first()
first: String = # Apache Spark

scala> val all = myfile.collect()
all: Array[String] = Array(# Apache Spark, "", Spark is a fast and
general cluster comp
# … Remaining detail omitted

scala> val scalaLines = myfile.filter(line => line.contains("Scala"))
scalaLines: org.apache.spark.rdd.RDD[String] = FilteredRDD[3] at filter
at <console>:14

scala> scalaLines.collect()
res1: Array[String] = Array(high-level APIs in Scala, Java, and Python,
and an optimized engine that, ## Interactive Scala Shell, The easiest
way to start using Spark is through the Scala shell:)
```

SparkSession: Loading Data - V2+ (Python)

```
>>> myfile= spark.read.text("README.md")

>>> myfile.show()
+--------------------+
|               value|
+--------------------+
|  Spark Labs        |
| ...                |

>>> first = myfile.first()
>>> print(first)
"Spark Labs"

>>> all = myfile.collect()

>>> sparkLines = myfile.filter(myfile.value.contains("Spark"))

>>> sparkLines.count()
12

>>> sparkLines.show()
+--------------------+
|               value|
+--------------------+
|          Spark Labs|
|Welcome to Spark ...|
```

SparkSession: Read Functions

Function	Description	Returns
spark.read.textFile	Reads unstructured text file	Dataset
spark.read.text	Reads text file	Dataframe
spark.read.csv	Reads CSV files	Dataframe
spark.read.json	- Reads JSON content - Parses JSON to figure out schema	Dataframe
spark.read.parquet	- Loads Parquet file	Dataframe

Lab: First Look at Spark Shell

- **Overview**: In this lab, we will work with the Spark Shell.
- **Builds on previous labs**:
  Lab 2.1 for general setup
- **Approximate time**:
  20-30 minutes
- **Follow *ONE* of the following:**
  - **Standalone Scala: '02-intro/2.2-shell.md' file**
  - **Standalone Python: 2.2-shell.ipynb**
  - **Hadoop: '02-intro/2.2-spark-shell-hadoop.md'**

Putting It All Together

- Spark is an Open Source, cluster-computing engine
  - Fast, flexible, and relatively easy to code
- Spark overcomes many MapReduce/Hadoop limitations
  - It's a faster, easier to program, with a unified stack for wide applicability
- Addresses the needs of current Big Data systems (Lambda Architectures)
  - In a unified stack
- Growing rapidly
  - Increasingly wide adoption
  - Extremely active community evolving the code base

Putting It All Together

- **Core**: Distributed processing of large data sets over a cluster
- **Spark SQL**: Structured data
- **Spark Streaming**: Live data streams
- **MLlib**: Machine learning
- **GraphX**: Graph Processing

Putting It All Together

- Spark can be installed standalone, or integrated with YARN or Mesos
  - It can be installed by extracting tarball
  - Included in some Hadoop distributions (Cloudera/Hortonworks)

- Spark **worker processes** run on distributed nodes
  - Running tasks as assigned by the driver

- The **Spark Shell** supports ad-hoc, interactive operations
  - Using the Scala or Python API
  - A `SparkContext` provides access to Spark

Review Questions

- Spark replaces Hadoop (True / False)
- What is Lambda architecture?
- Can Lambda be implemented completely in Spark?

Licensed for personal use only for Fernando K <fernando_kruse@dell.com> from Machine Learning at Dell Brazil (QE) @ 2019-03-12

Backup Slides

Licensed for personal use only for Fernando K <fernando_kruse@dell.com> from Machine Learning at Dell Brazil (QE) @ 2019-03-12

SparkContext–First Look (V1.6)

- Access to Spark is via a `SparkContext` instance
  - Represents connection to Spark cluster
  - Pre-created in shell as variable **`sc`**
  - Below, we illustrate some simple uses of the context
  - Note the tab completion in the second example

```
scala> sc
res0: org.apache.spark.SparkContext = org.apache.spark.SparkContext@2c01a0dd

scala> sc.[tab]
accumulable      accumulableCollection accumulator       addFile              addJar

addSparkListener appName               applicationId     asInstanceOf         binaryFiles

binaryRecords    broadcast             cancelAllJobs     cancelJobGroup       clearCallSite

clearFiles       clearJars             clearJobGroup     defaultMinPartitions defaultMinSplits
… Remaining detail omitted

scala> sc.isLocal
res1: Boolean = true

scala> sc.master
res2: String = local[*]
```

SparkContext: Loading Data - V1.6 (Scala)

```
# Create an RDD from contents of file
scala> val myfile = sc.textFile("README.md")
myfile: org.apache.spark.rdd.RDD[String] = README.md MappedRDD[1] at
textFile at <console>:12

scala> val first = myfile.first()
first: String = # Apache Spark

scala> val all = myfile.collect()
all: Array[String] = Array(# Apache Spark, "", Spark is a fast and
general cluster comp
# … Remaining detail omitted

scala> val scalaLines = myfile.filter(line => line.contains("Scala"))
scalaLines: org.apache.spark.rdd.RDD[String] = FilteredRDD[3] at filter
at <console>:14

scala> scalaLines.collect()
res1: Array[String] = Array(high-level APIs in Scala, Java, and Python,
and an optimized engine that, ## Interactive Scala Shell, The easiest
way to start using Spark is through the Scala shell:)
```