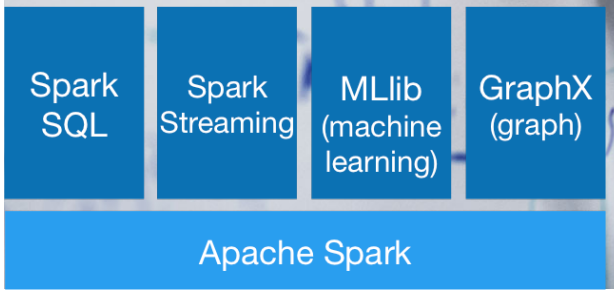
Hortonworks

Hortonworks Technical Workshop: In-memory Processing with Apache Spark

Dhruv Kumar and Ajay Singh

Hortonworks. We do Hadoop.

March 12, 2015



About the presenters

Ajay Singh
Director Technical Channels.
Hortonworks Inc.

Dhruv Kumar
Partner Solutions Engineer.
Hortonworks Inc.

Hortonworks

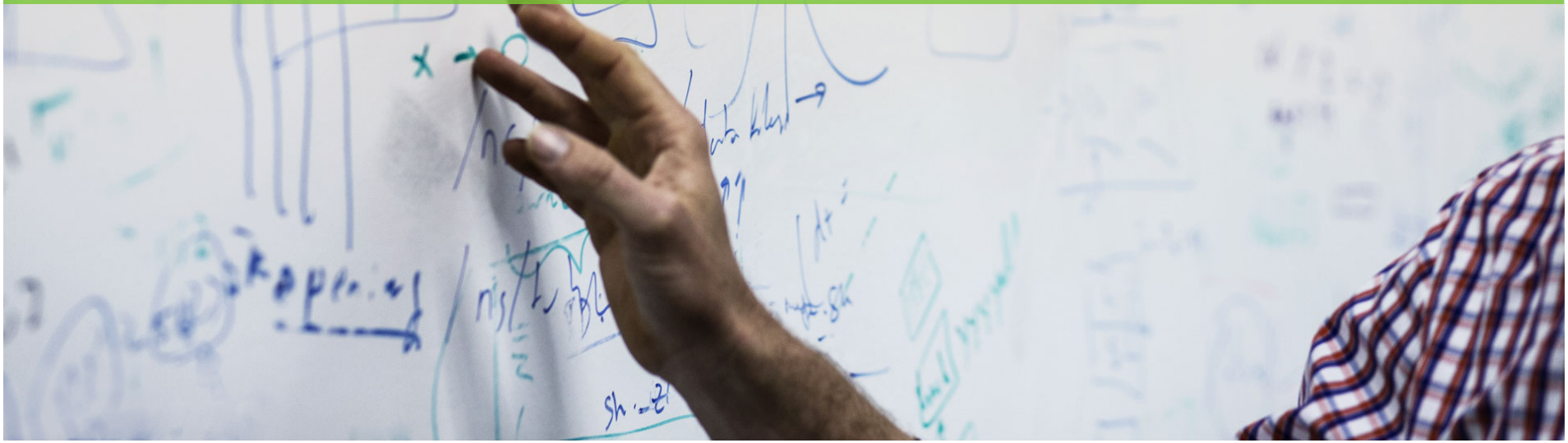
In this workshop

- Introduction to HDP and Spark
- Installing Spark on HDP
- Spark Programming
    - Core Spark: working with RDDs
    - Spark SQL: structured data access
    - Spark Mlib: predictive analytics
    - Spark Streaming: real time data processing
- Spark Application Demo: Twitter Language Classifier using Mlib and Streaming
- Tuning and Debugging Spark
- Conclusion and Further Reading, Q/A

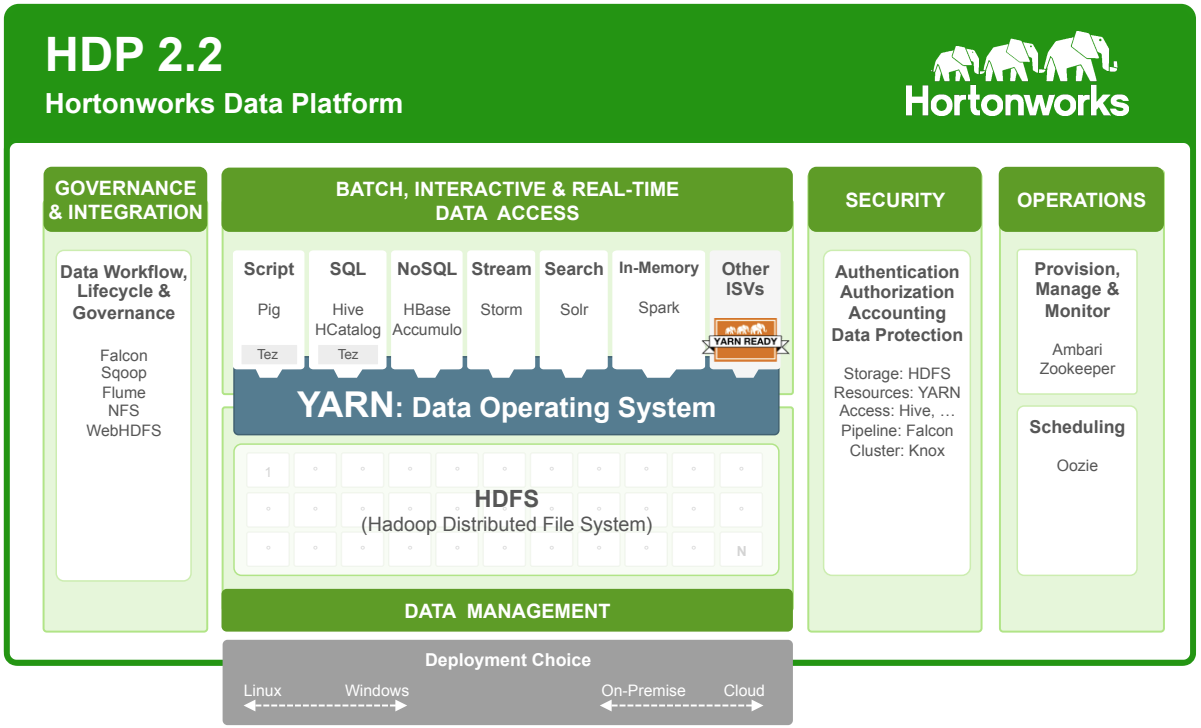
Hortonworks

Hortonworks

Introduction to HDP and Spark



HDP delivers a comprehensive data management platform



YARN is the architectural center of HDP

- Enables batch, interactive and real-time workloads
- Single SQL engine for both batch and interactive
- Enables best of breed ISV tools to deeply integrate into Hadoop via YARN

Provides comprehensive enterprise capabilities

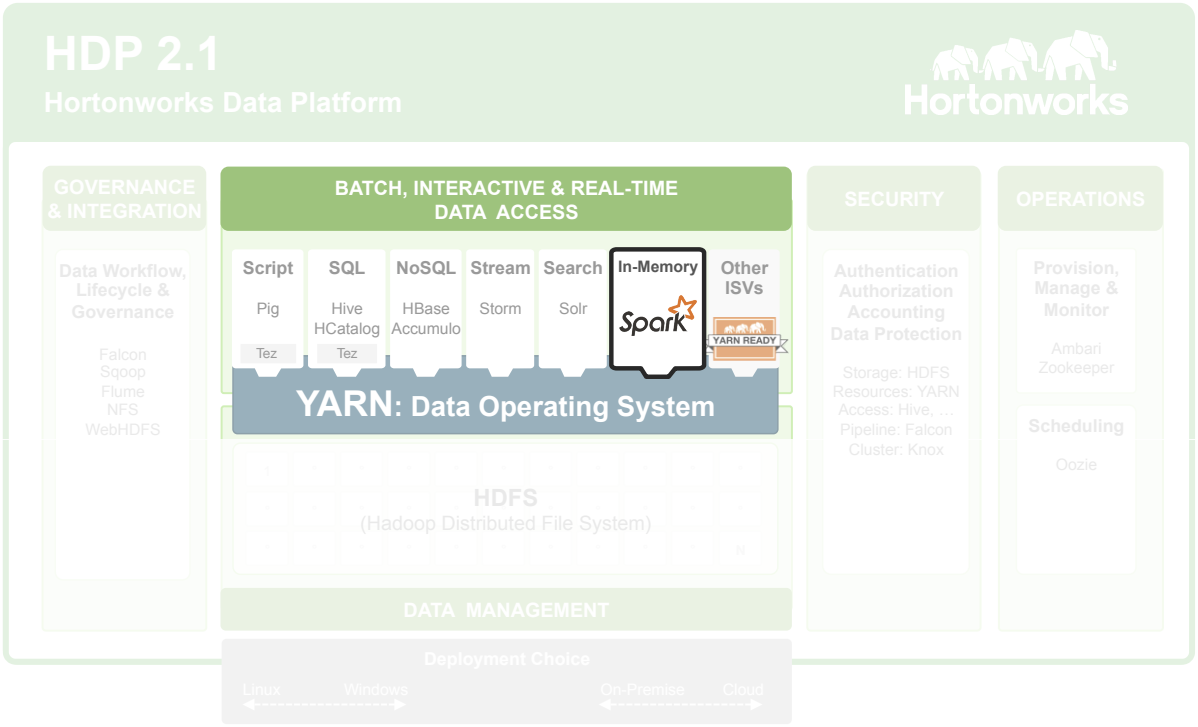
- Governance
- Security
- Operations

The widest range of deployment options

- Linux & Windows
- On premise & cloud

Hortonworks

Let's drill into one workload … Spark



YARN is the architectural center of HDP

- Enables batch, interactive and real-time workloads
- Single SQL engine for both batch and interactive
- Enables best of breed ISV tools to deeply integrate into Hadoop via YARN

Provides comprehensive enterprise capabilities

- Governance
- Security
- Operations

The widest range of deployment options

- Linux & Windows
- On premise & cloud

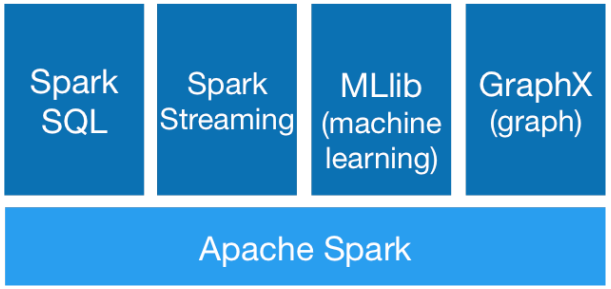
Hortonworks

What is Spark?

- **Spark is**
  - an *open-source* software solution that performs rapid calculations on *in-memory datasets*
    - Open Source [Apache hosted & licensed]
      - Free to download and use in production
      - Developed by a community of developers
    - In-memory datasets
      - RDD (Resilient Distributed Data) is the basis for what Spark enables
      - Resilient – the models can be recreated on the fly from known state
      - Distributed – the dataset is often partitioned across multiple nodes for increased scalability and parallelism

Hortonworks

Spark Components



Spark allows you to do data processing, ETL, machine learning, stream processing, SQL querying from one framework

Hortonworks

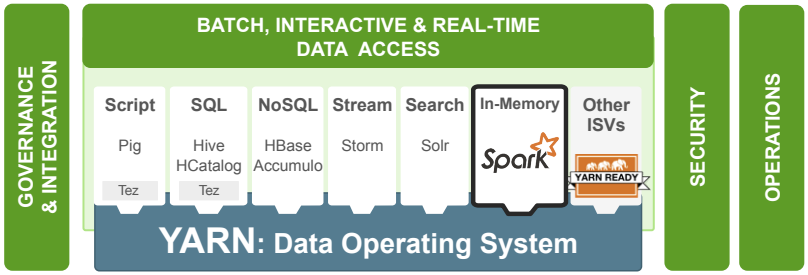
Why Spark?

- One tool for data engineering and data science tasks
- Native integration with Hive, HDFS and any Hadoop FileSystem implementation
- Faster development: concise API, Scala (~3x lesser code than Java)
- Faster execution: for *iterative jobs* because of in-memory caching (not all workloads are faster in Spark)
- Promotes code reuse: APIs and data types are similar for batch and streaming

Hortonworks

Hortonworks Commitment to Spark

Hortonworks is focused on making Apache Spark enterprise ready so you can depend on it for mission critical applications



1. **YARN enable Spark to co-exist with other engines**
   Spark is "YARN Ready" so its memory & CPU intensive apps can work with predictable performance along side other engines all on the same set(s) of data.

2. **Extend Spark with enterprise capabilities**
   Ensure Spark can be managed, secured and governed all via a single set of frameworks to ensure consistency. Ensure reliability and quality of service of Spark along side other engines.

3. **Actively collaborate within the open community**
   As with everything we do at Hortonworks we work entirely within the open community across Spark and all related projects to improve this key Hadoop technology.

Hortonworks

Spark on HDP: history and what's coming

Spark Tech Previews we've done so far

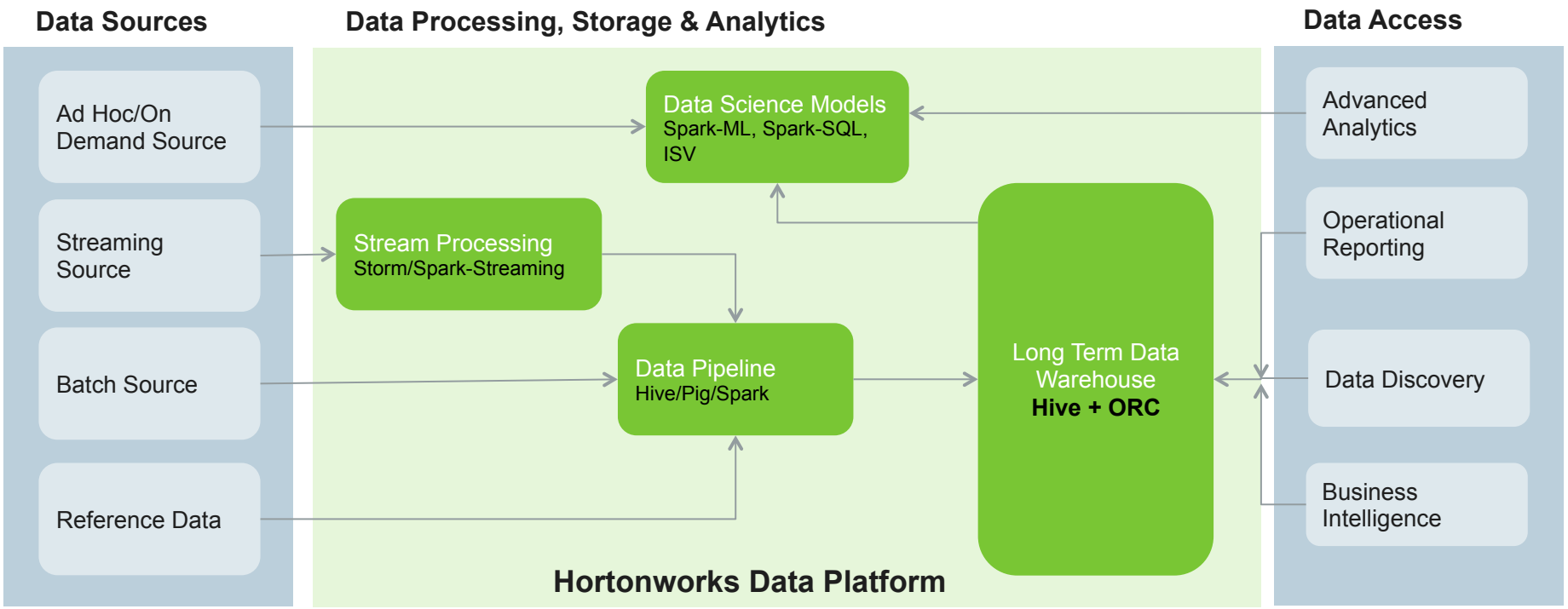
Spark 0.9.1
Spark 1.0.1
Spark 1.2.0

Spark GA: Q1/Q2 of 2015
Install Spark 1.2.1 with HDP 2.2.2 & Ambari 2.0.0

- **Install with Ambari/ Manual package for DEB/RPM/MSI**
- **Spark on Kerberos enabled cluster**
- **Spark on YARN**
- **Spark job history Server**
- **Full ORC support**
- **Built with Hive 0.14 dependency**
- **Spark Thrift Server for JDBC/ODBC**
- **Spark embedded Beeline**
- **Spark Simba ODBC Driver (coming)**

Hortonworks

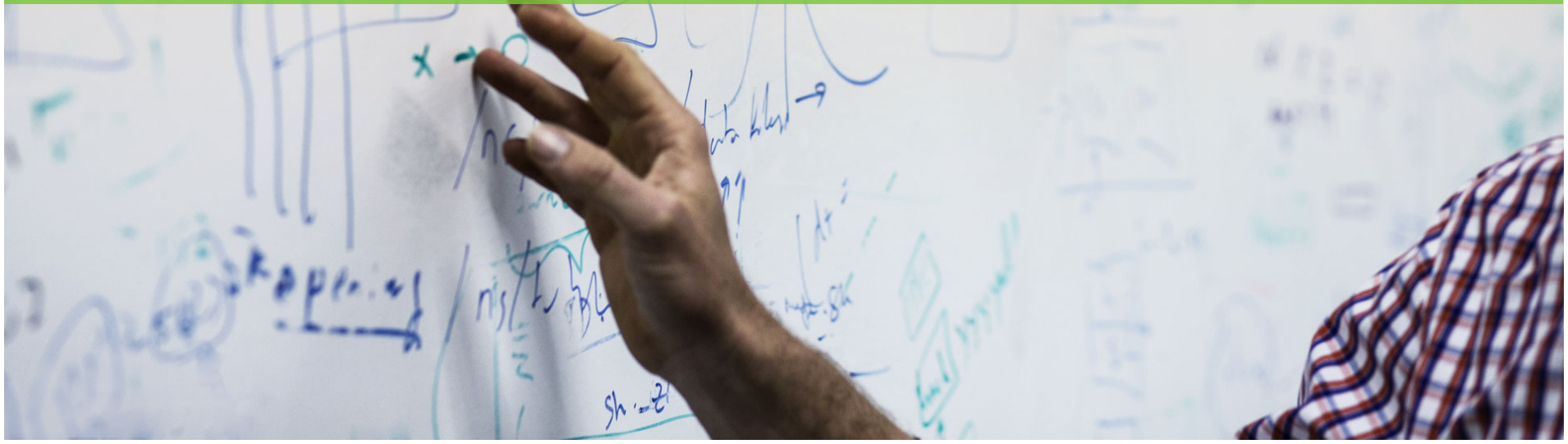
Reference Deployment Architecture



Hortonworks

Hortonworks

Installing Spark on HDP



Installing Spark on HDP

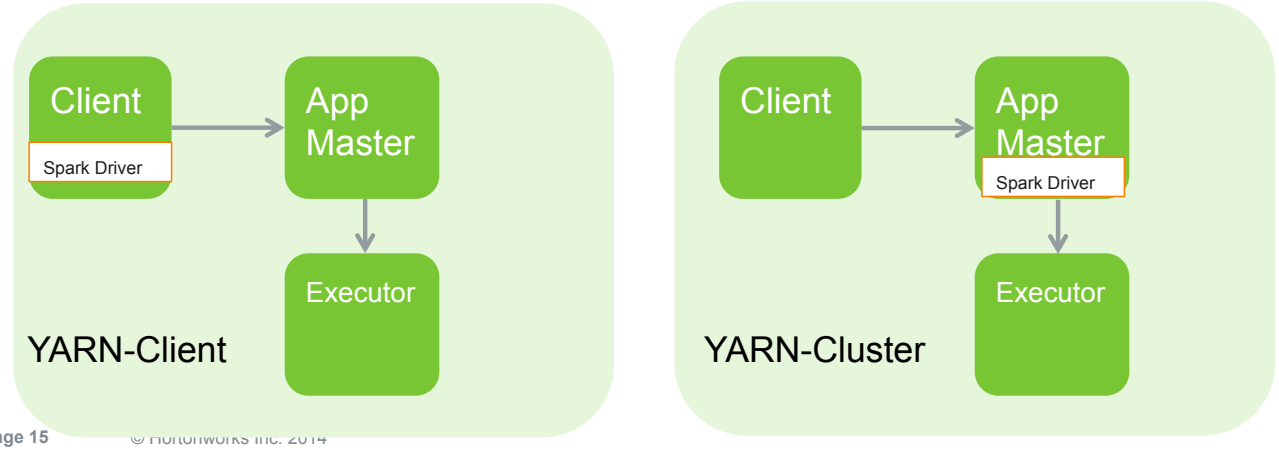
- GA of Spark 1.2.1 in Q1 2015
  - Fully supported by Hortonworks
  - Install with Ambari 2.0.0 & HDP 2.2.2. Other combination unsupported.
- Can try tech preview now

Hortonworks

Spark Deployment Modes

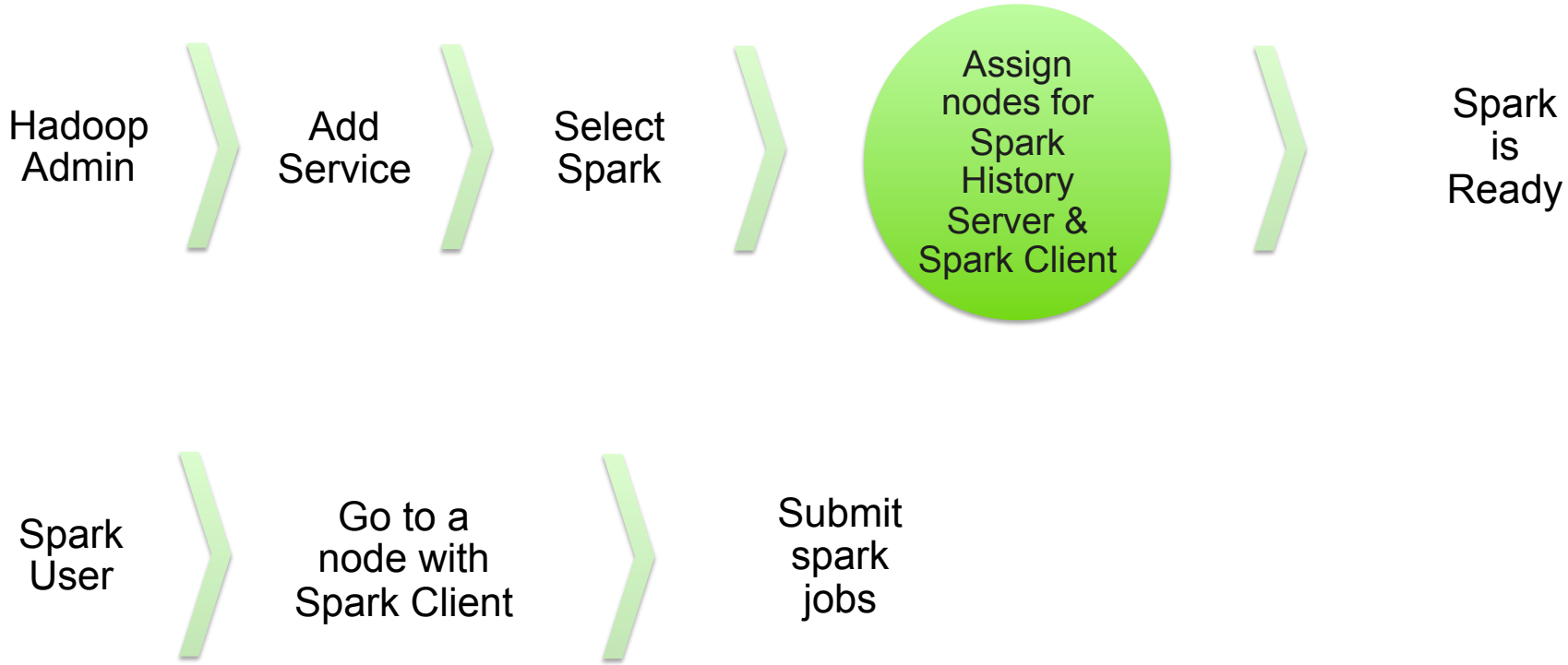
- Spark Standalone Cluster
  - For developing Spark apps against a local Spark (similar to develop/deploying in IDE)
- Spark on YARN
  - Spark driver (SparkContext) in YARN AM(yarn-cluster)
  - Spark driver (SparkContext) in local (yarn-client)
    - Spark Shell runs in yarn-client only

Mode setup with Ambari



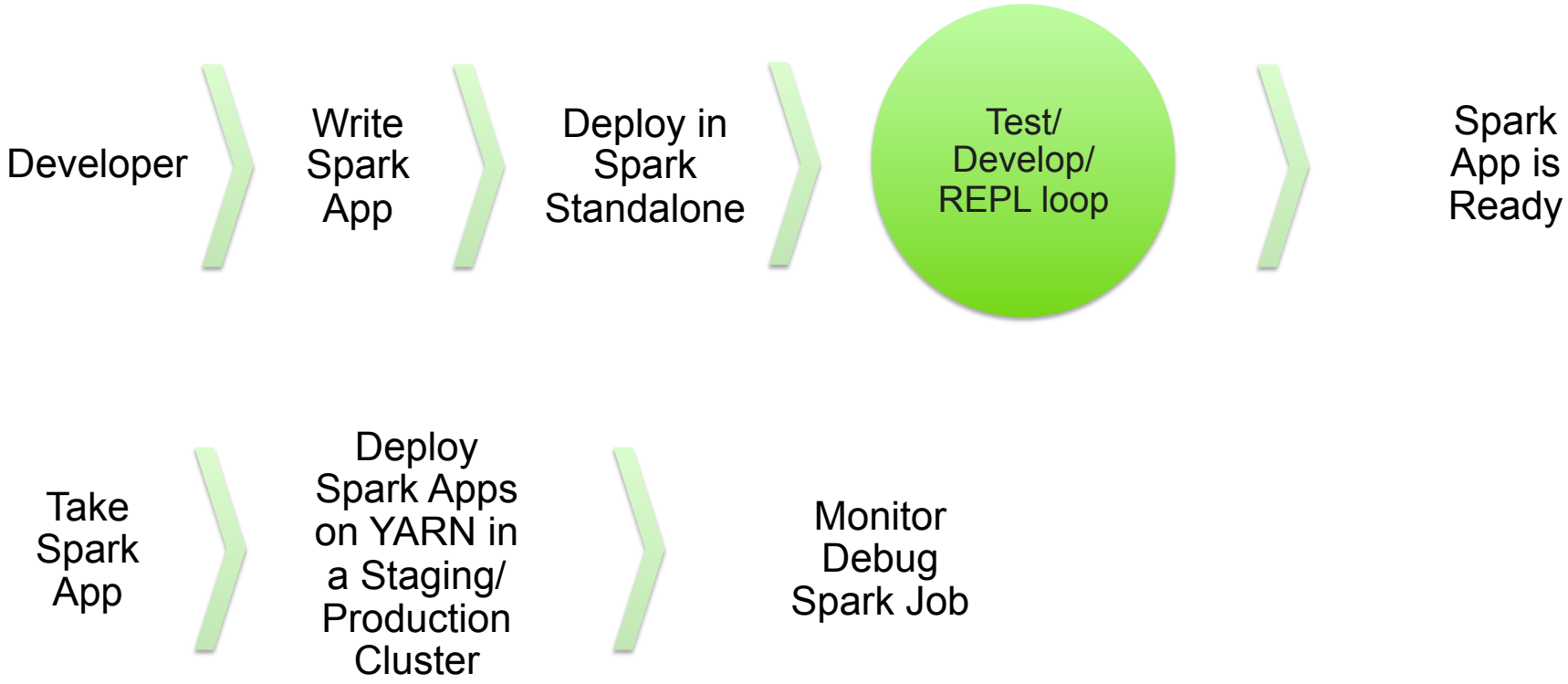
Hortonworks

Overview of Spark Install with Ambari



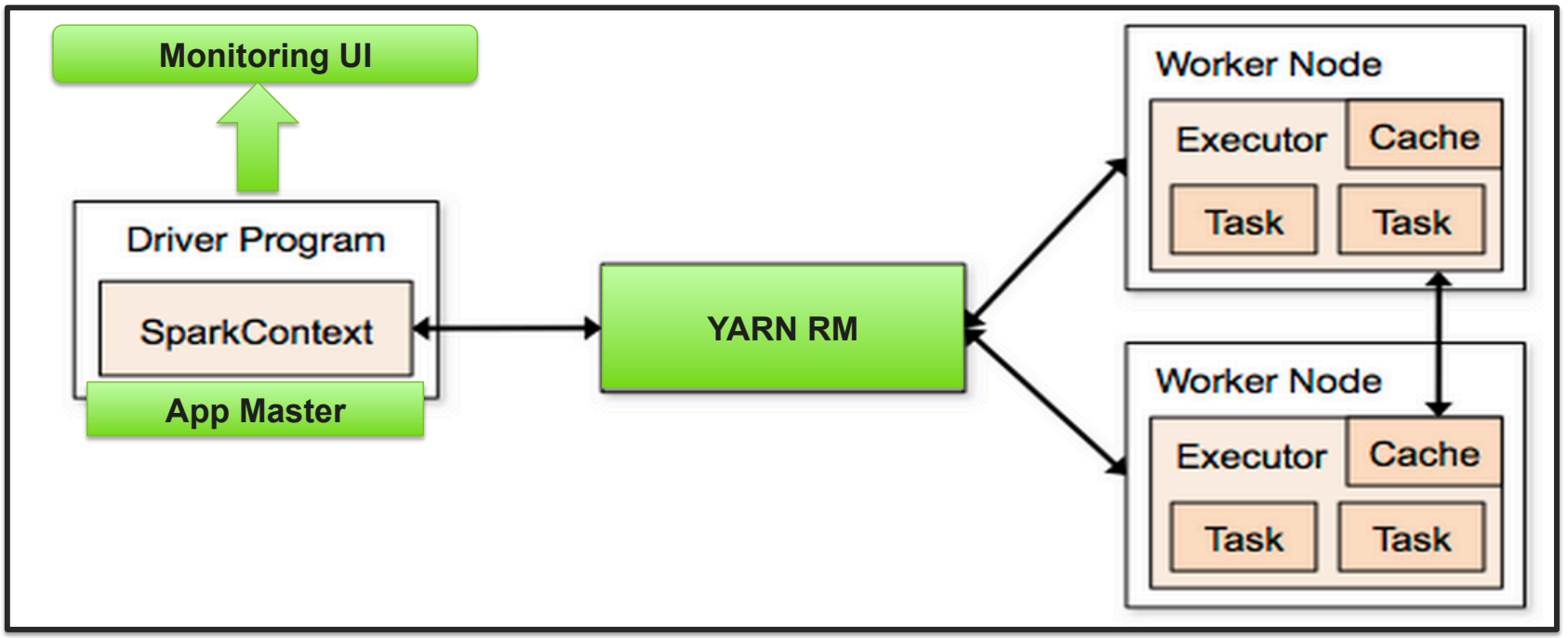
Hortonworks

Overview of Spark App Lifecycle



Hortonworks

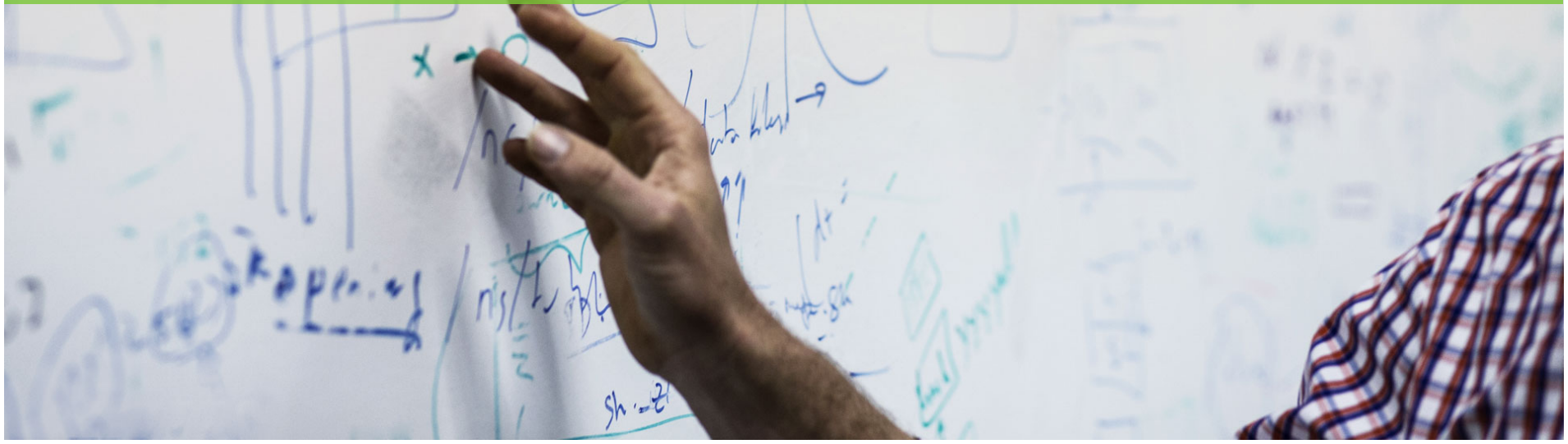
Spark on YARN



Hortonworks

Hortonworks

Programming Spark



How Does Spark Work?

- **RDD**
  - Your data is loaded in parallel into structured collections
- **Actions**
  - Manipulate the state of the working model by forming new RDDs and performing calculations upon them
- **Persistence**
  - Long-term storage of an RDD's state

Example RDD Transformations

- **map(func)**
- **filter(func)**
- **distinct(func)**

- All create a new DataSet from an existing one
- Do not create the DataSet until an action is performed (Lazy)
- Each element in an RDD is passed to the target function and the result forms a new RDD

Hortonworks

Example Action Operations

- **count()**
- **reduce(func)**
- **collect()**
- **take()**

- Either:
  - Returns a value to the driver program
  - Exports state to external system

Hortonworks

Example Persistence Operations

- **persist() -- takes options**
- **cache() -- only one option: in-memory**

- Stores RDD Values
  - in memory (what doesn't fit is recalculated when necessary)
    - Replication is an option for in-memory
  - to disk
  - blended

Hortonworks

1. Resilient Distributed Dataset [RDD] Graph

```
val v = sc.textFile("hdfs://…some-hdfs-data")
        v.flatMap(line=>line.split(" "))
        .map(word=>(word, 1)))
        .reduceByKey(_ + _, 3)
        .collect()
```

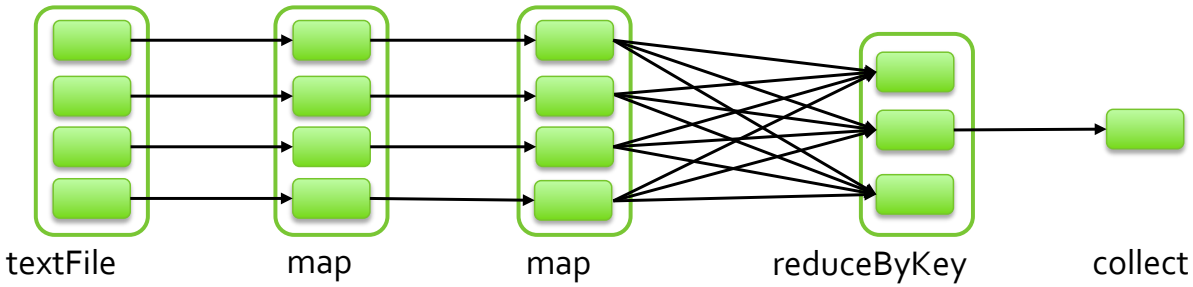
RDD[String]

RDD[List[String]]

RDD[(String, Int)]

RDD[(String, Int)]

Array[(String, Int)]



Hortonworks

Processing A File in Scala

```
//Load the file:
val file = sc.textFile("hdfs://…/user/DAW/littlelog.csv")
//Trim away any empty rows:
val fltr = file.filter(_.length > 0)
//Print out the remaining rows:
fltr.foreach(println)
```

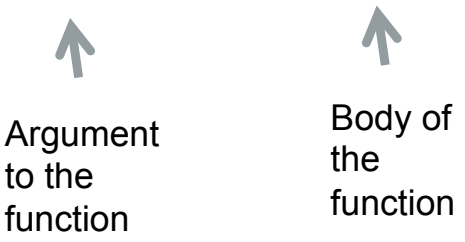
Looking at the State in the Machine

```
//run debug command to inspect RDD:
scala> fltr.toDebugString
//simplified output:
res1: String =
FilteredRDD[2] at filter at <console>:14
   MappedRDD[1] at textFile at <console>:12
      HadoopRDD[0] at textFile at <console>:12
```

A Word on Anonymous Functions

Scala programmers make great use of anonymous functions as can be seen in the code:

```
flatMap( line => line.split(" ") )
```



Scala Functions Come In a Variety of Styles

```
flatMap( line => line.split(" ") )
```

Argument to the function (type inferred)

Body of the function

```
flatMap((line:String) => line.split(" "))
```

Argument to the function (explicit type)

Body of the function

```
flatMap(_.split(" "))
```

No Argument to the function declared (placeholder) instead

Body of the function includes placeholder _ which allows for exactly one use of one arg for each _ present. _ essentially means 'whatever you pass me'

Hortonworks

And Finally – the Formal 'def'

```
def myFunc(line:String): Array[String]={
```

Argument to the function)

Return type of the function)

```
  return line.split(",")
}
```

Body of the function

```
//and now that it has a name:
myFunc("Hi Mom, I'm home.").foreach(println)
```


Hortonworks

Things You Can Do With RDDs

- RDDs are objects and expose a rich set of methods:

Name	Description	Name	Description
filter	Return a new RDD containing only those elements that satisfy a predicate	collect	Return an array containing all the elements of this RDD
count	Return the number of elements in this RDD	first	Return the first element of this RDD
foreach	Applies a function to all elements of this RDD (does not return an RDD)	reduce	Reduces the contents of this RDD
subtract	Return an RDD without duplicates of elements found in passed-in RDD	union	Return an RDD that is a union of the passed-in RDD and this one

More Things You Can Do With RDDs

- More stuff you can do…

Name	Description	Name	Description
flatMap	Return a new RDD by first applying a function to all elements of this RDD, and then flattening the results	checkpoint	Mark this RDD for checkpointing (its state will be saved so it need not be recreated from scratch)
cache	Load the RDD into memory (what doesn't fit will be calculated as needed)	countByValue	Return the count of each unique value in this RDD as a map of (value, count) pairs
distinct	Return a new RDD containing the distinct elements in this RDD	persist	Store the RDD to either memory, Disk, or hybrid according to passed in value
sample	Return a sampled subset of this RDD	unpersist	Clear any record of the RDD from disk/memory

Code ‘select count’

Equivalent SQL Statement:

```
Select count(*) from pagecounts WHERE state = ‘FL’
```

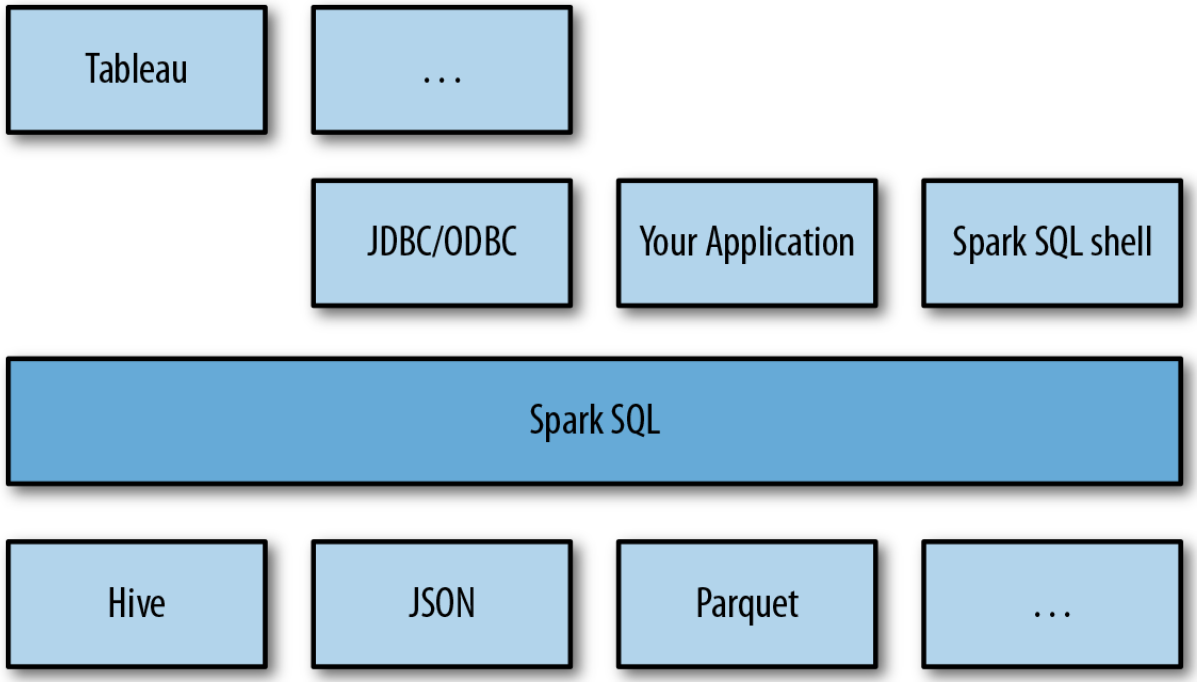
Scala statement:

```
val file = sc.textFile("hdfs://…/log.txt")
val numFL = file.filter(line =>
line.contains("fl")).count()

scala> println(numFL)
```

1. Load the page as an RDD
2. Filter the lines of the page eliminating any that do not contain “fl“
3. Count those lines that remain
4. Print the value of the counted lines containing ‘fl’

Spark SQL



Hortonworks

What About Integration With Hive?

```
scala> val hiveCTX = new org.apache.spark.sql.hive.HiveContext(sc)
scala> hiveCTX.hql("SHOW TABLES").collect().foreach(println)
…
[omniture]
[omniturelogs]
[orc_table]
[raw_products]
[raw_users]
…
```

More Integration With Hive:

```
scala> hCTX.hql("DESCRIBE raw_users").collect().foreach(println)
[swid,string,null]
[birth_date,string,null]
[gender_cd,string,null]

scala> hCTX.hql("SELECT * FROM raw_users WHERE gender_cd='F' LIMIT
5").collect().foreach(println)
[0001BDD9-EABF-4D0D-81BD-D9EABFCD0D7D,8-Apr-84,F]
[00071AA7-86D2-4EB9-871A-A786D27EB9BA,7-Feb-88,F]
[00071B7D-31AF-4D85-871B-7D31AFFD852E,22-Oct-64,F]
[000F36E5-9891-4098-9B69-CEE78483B653,24-Mar-85,F]
[00102F3F-061C-4212-9F91-1254F9D6E39F,1-Nov-91,F]
```

Querying RDD Using SQL

```
// SQL statements can be run directly on RDD's
val teenagers =
  sqlC.sql("SELECT name FROM people
   WHERE age >= 13 AND age <= 19")

// The results of SQL queries are SchemaRDDs and support
// normal RDD operations:

val nameList = teenagers.map(t => "Name: " + t(0)).collect()

// Language integrated queries (ala LINQ)
val teenagers =
 people.where('age >= 10).where('age <= 19).select('name)
```

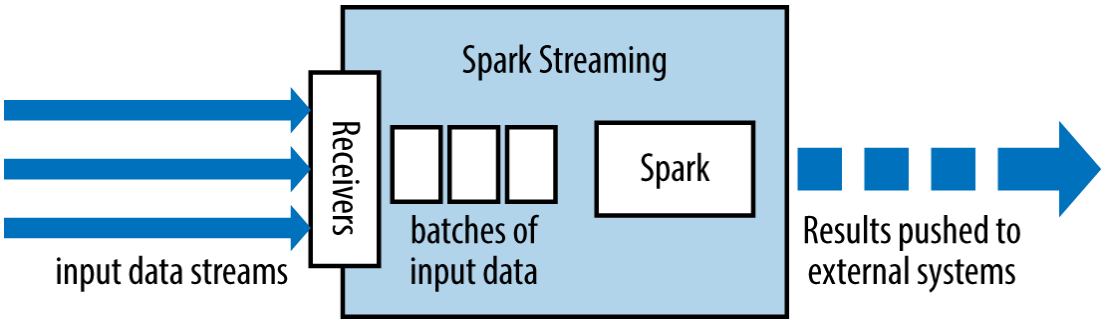

Hortonworks

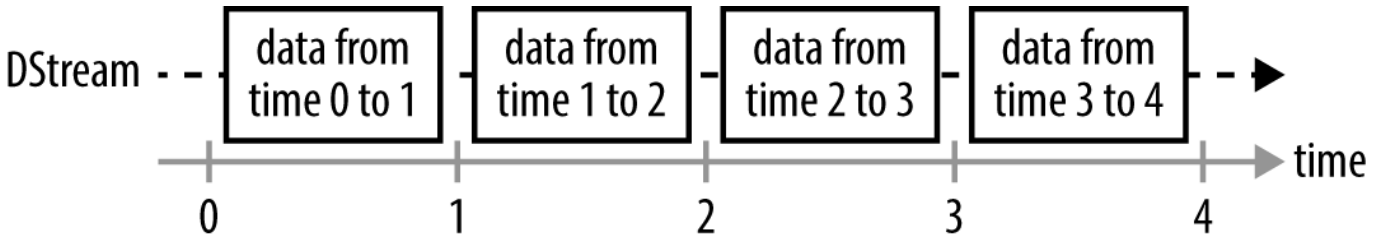
Spark MLlib – Algorithms Offered

- Classification: logistic regression, linear SVM,
  - naïve Bayes, least squares, classification tree
- Regression: generalized linear models (GLMs),
  - regression tree
- Collaborative filtering: alternating least squares (ALS),
  - non-negative matrix factorization (NMF)
- Clustering: k-means
- Decomposition: SVD, PCA
- Optimization: stochastic gradient descent, L-BFGS

Hortonworks

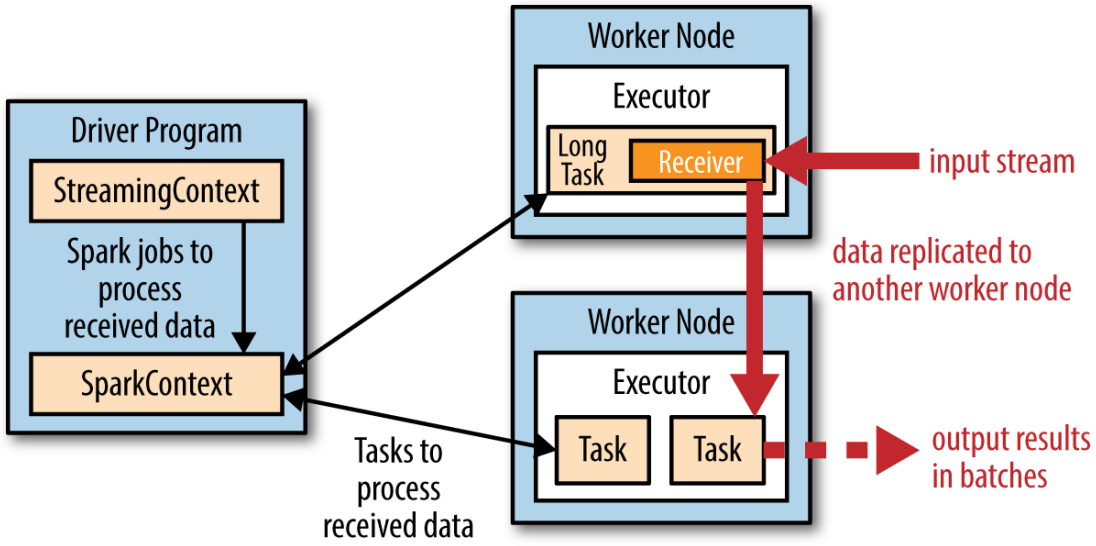
MicroBatch Spark Streams





Hortonworks

Physical Execution



Hortonworks

Spark Streaming 101

- Spark has significant library support for streaming applications

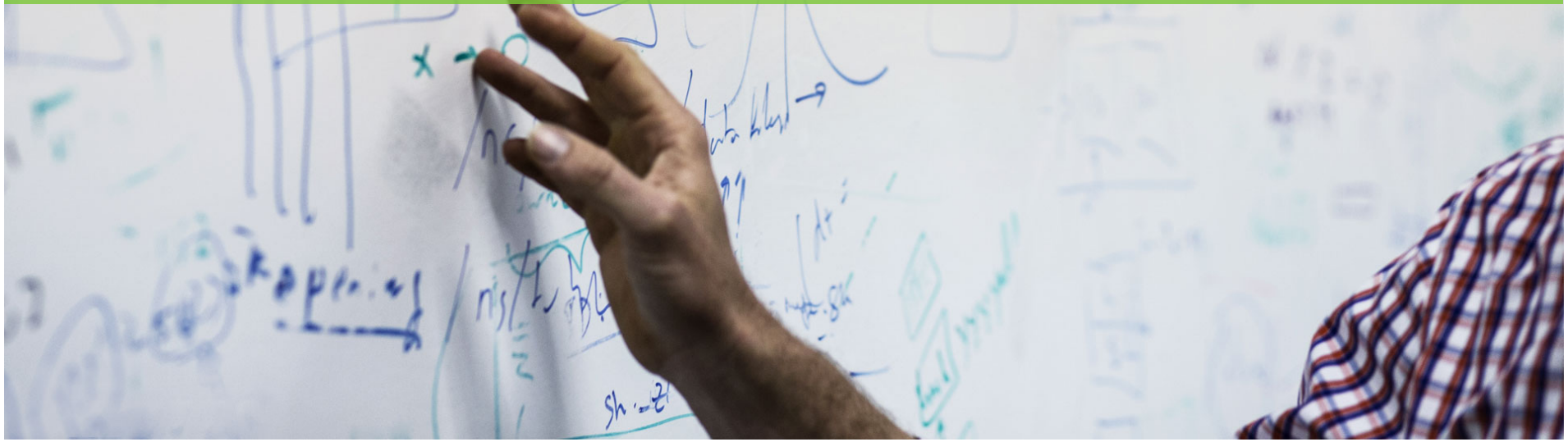
```
val ssc = new StreamingContext(sc, Seconds(5))
val tweetStream = TwitterUtils.createStream(ssc, Some(auth))
```

- Allows to combine Streaming with Batch/ETL,SQL & ML
- Read data from HDFS, Flume, Kafka, Twitter, ZeroMQ & custom.
- Chop input data stream into batches
- Spark processes batches & results published in batches
- Fundamental unit is Discretized Streams (DStreams)

Hortonworks

Hortonworks

Spark on HDP Demo



Twitter Language Classifier

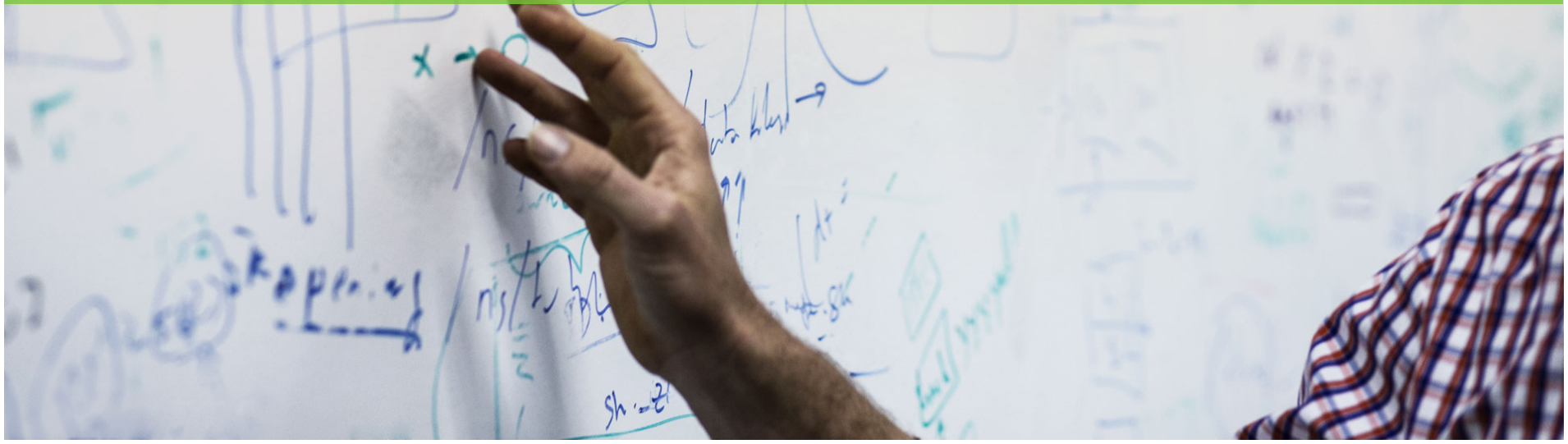
Goal: connect to real time twitter stream and print only those tweets whose language match our chosen language.

Main issue: how to detect the language during run time?

Solution: build a language classifier model offline capable of detecting language of tweet (Mlib). Then, apply it to real time twitter stream and do filtering (Spark Streaming).

Hortonworks

Tuning and Debugging Spark



Spark Tuning - 1

- **Understand how Spark works**
  - Spark Transformations create new RDDs
  - Transformations are executed lazily when action is called.
  - Collect() may call OOME when large data is sent from Executors to Driver
  - Prefer ReduceByValue over ReduceByKey
- **Understand your application**
  - ML is CPU intensive
  - ETL is IO intensive
  - Avoid shuffle
  - Use the right join for the Table (ShuffledHashJoin vs BroadcastHashJoin)
    - Manual Configuration

Hortonworks

Spark Tuning - 2

- **Understand where time is spent**
  - See http://<host>:8088/proxy/<app_id>/stages/

Completed Stages (4)

Stage Id	Description		Submitted	Duration
4	collect at SparkPlan.scala:84	+details	2015/03/04 12:53:35	14 s
3	mapPartitions at Exchange.scala:77	+details	2015/03/04 12:53:11	24 s
1	RangePartitioner at Exchange.scala:88	+details	2015/03/04 12:53:06	4 s
0	mapPartitions at Exchange.scala:64	+details	2015/03/04 12:53:02	5 s

Failed Stages (0)

- **Use Kryo serializer**
  - Needs configuration and registering class
- **`Use toDebugString on RDD`**

```
- counts.toDebugString
res2: String =
(2) ShuffledRDD[4] at reduceByKey at <console>:14 []
 +-(2) MappedRDD[3] at map at <console>:14 []
    |  FlatMappedRDD[2] at flatMap at <console>:14 []
    |  hdfs://red1:8020/tmp/data MappedRDD[1] at textFile at <console>:12 []
    |  hdfs://red1:8020/tmp/data HadoopRDD[0] at textFile at <console>:12 []
```


Hortonworks

When Things go wrong

- **Where to look**
  - yarn application –list (get the list of running application)
  - yarn logs -applicationId <app_id>
  - Check Spark Environment : http://<host>:8088/proxy/<job_id>/environment/
- **Common Issues**
  - **Submitted a job but nothing happens**
    - Job stays in accepted state when allocated more memory/cores than is available
    - May need to kill unresponsive/stale jobs
  - **Insufficient HDFS access**
    - May lead to failure such as

```
"Loading data to table default.testtable
Failed with exception Unable to move sourcehdfs://red1:8020/tmp/hive-spark/
hive_2015-03-04_12-45-42_404_3643812080461575333-1/-ext-10000/kv1.txt to destination hdfs://red1:8020/apps/hive/
warehouse/testtable/kv1.txt"
```

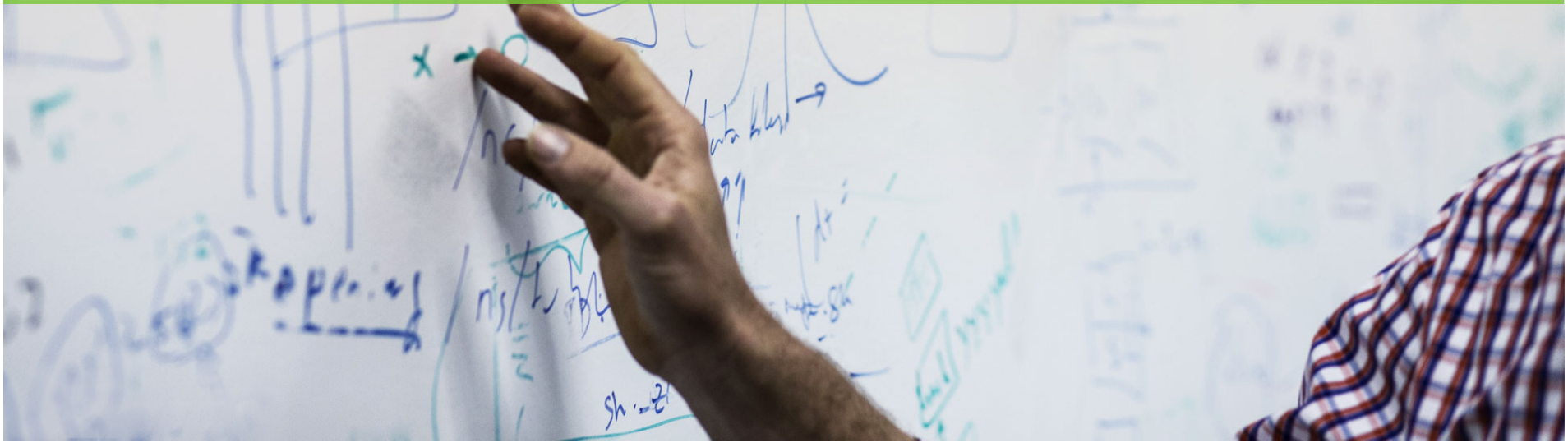
  - **Wrong host in Beeline, shows error as invalid URL**
    - "Error: Invalid URL: jdbc:hive2://localhost:10001 (state=08S01,code=0)"
  - **Error about closed SQLContext, restart Thirft Server**

Grant user/group necessary HDFS access

Hortonworks

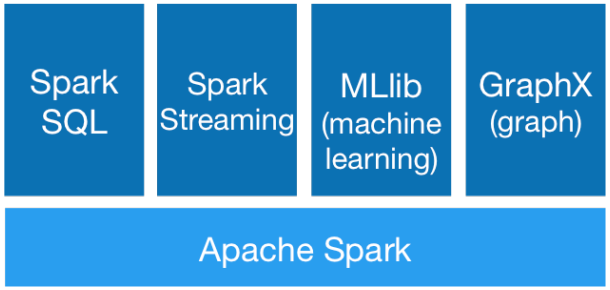
Hortonworks

Conclusion and Resources



Conclusion

- Spark is a unified framework for data engineering and data science

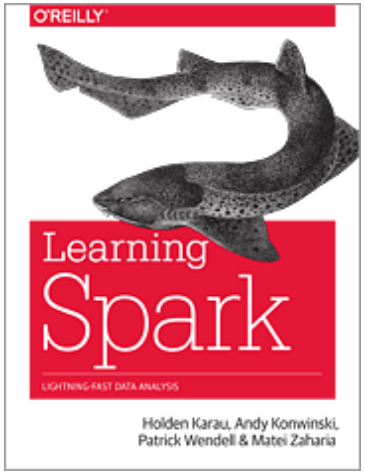


- Spark can be programmed in Scala, Java and Python.
- Spark will be supported by Hortonworks on HDP in Q1 2015
- Certain workloads are faster in Spark because of in-memory caching.

Hortonworks

References and Further Reading

- Apache Spark website: https://spark.apache.org/
- Hortonworks Spark website: http://hortonworks.com/hadoop/spark/
- Databricks Developer Resources: https://databricks.com/spark/developer-resources
- *"Learning Spark"* by O'Reilly Publishers



Hortonworks