Apache Spark Developer Training

Manaranjan Pradhan

About Me

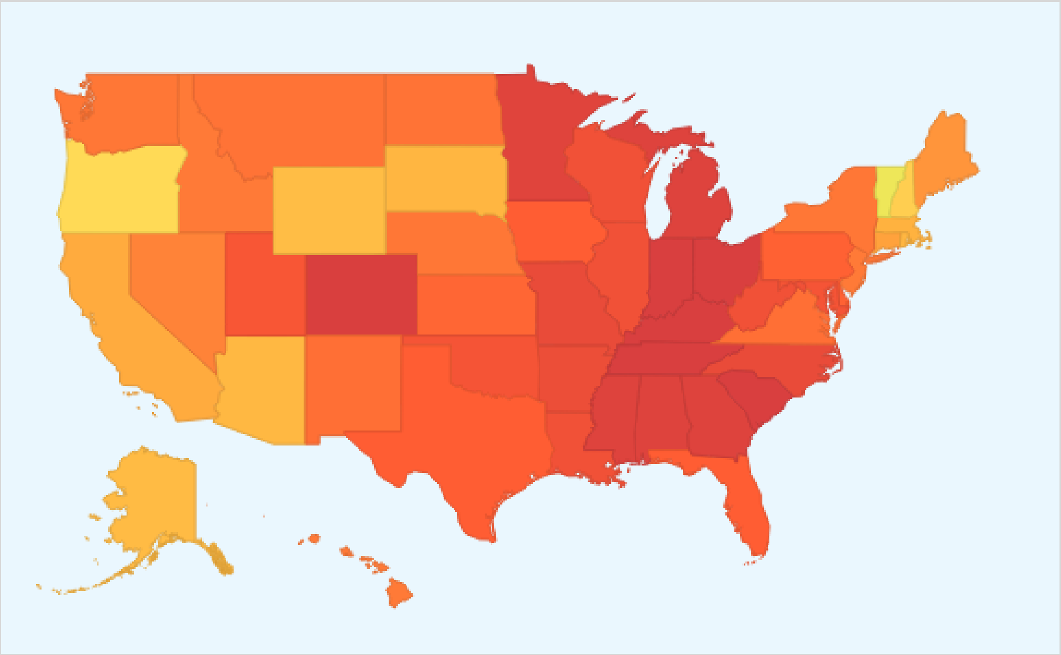
Manaranjan Pradhan is a big data & analytics enthusiast. He worked with TCS, HP and iGate patni for 15 years before deciding to quit and be a freelancer. Now he teaches and consults on big data platforms like Hadoop, Spark and scalable machine learning. He is an alumni of IIM Bangalore and currently also teaching and doing research projects at IIM Bangalore.

mail: manaranjan@enablecloud.com
https://www.linkedin.com/in/manaranjanpradhan

He write blogs at:

http://blog.enablecloud.com/
http://www.awesomestats.in/

Google Flu Trends



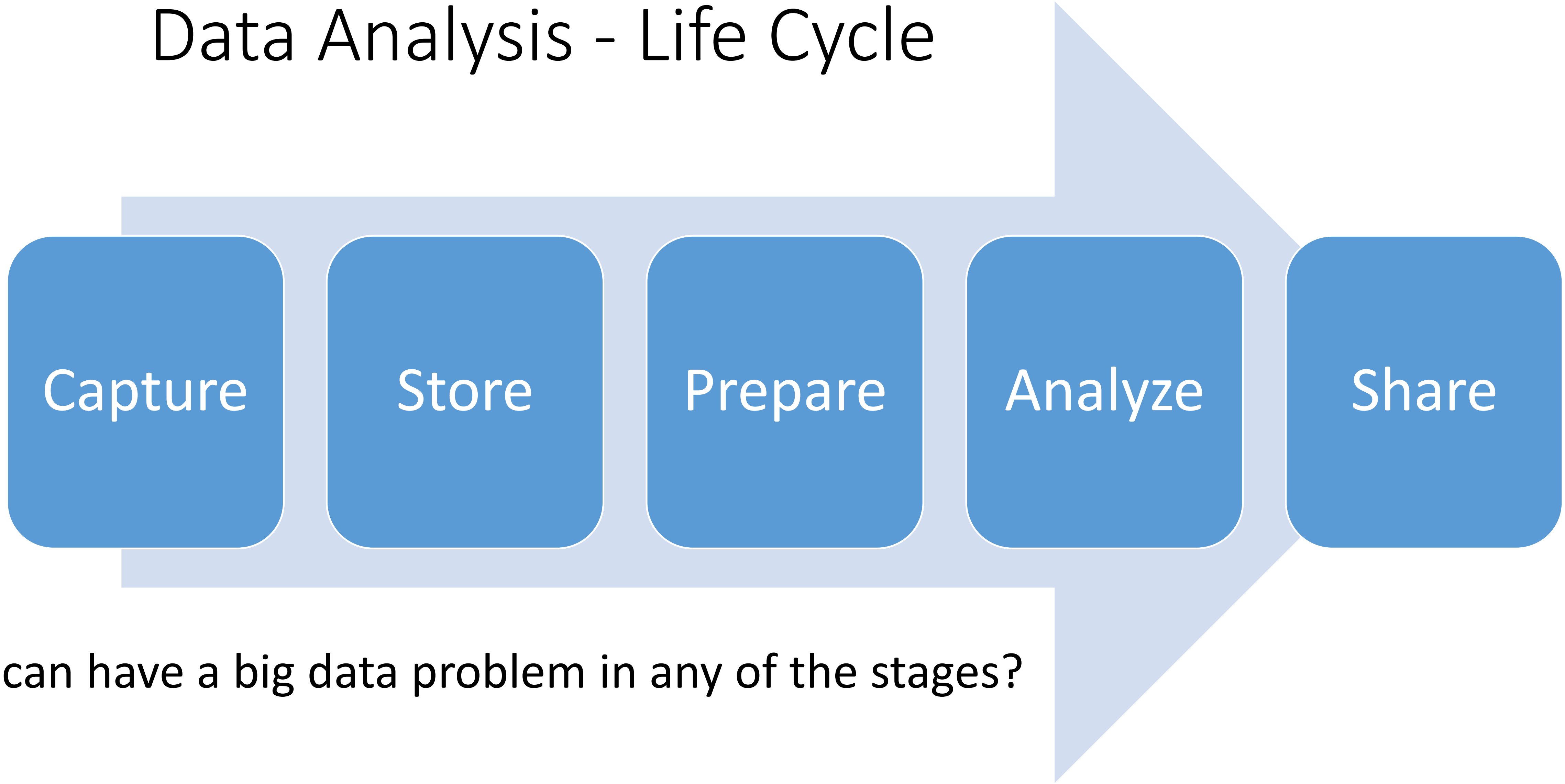
Estimates were made using a model that proved accurate when compared to historic official flu activity data. Data current through December 27, 2014.

- Search billions of requests
- Find out 50 million common search requests
- Run correlations between CDC data and search terms
- Filter out 50 common terms with high correlations > 0.9

$$logit(P) = \beta_0 + \beta_1 \times logit(Q) + \varepsilon$$

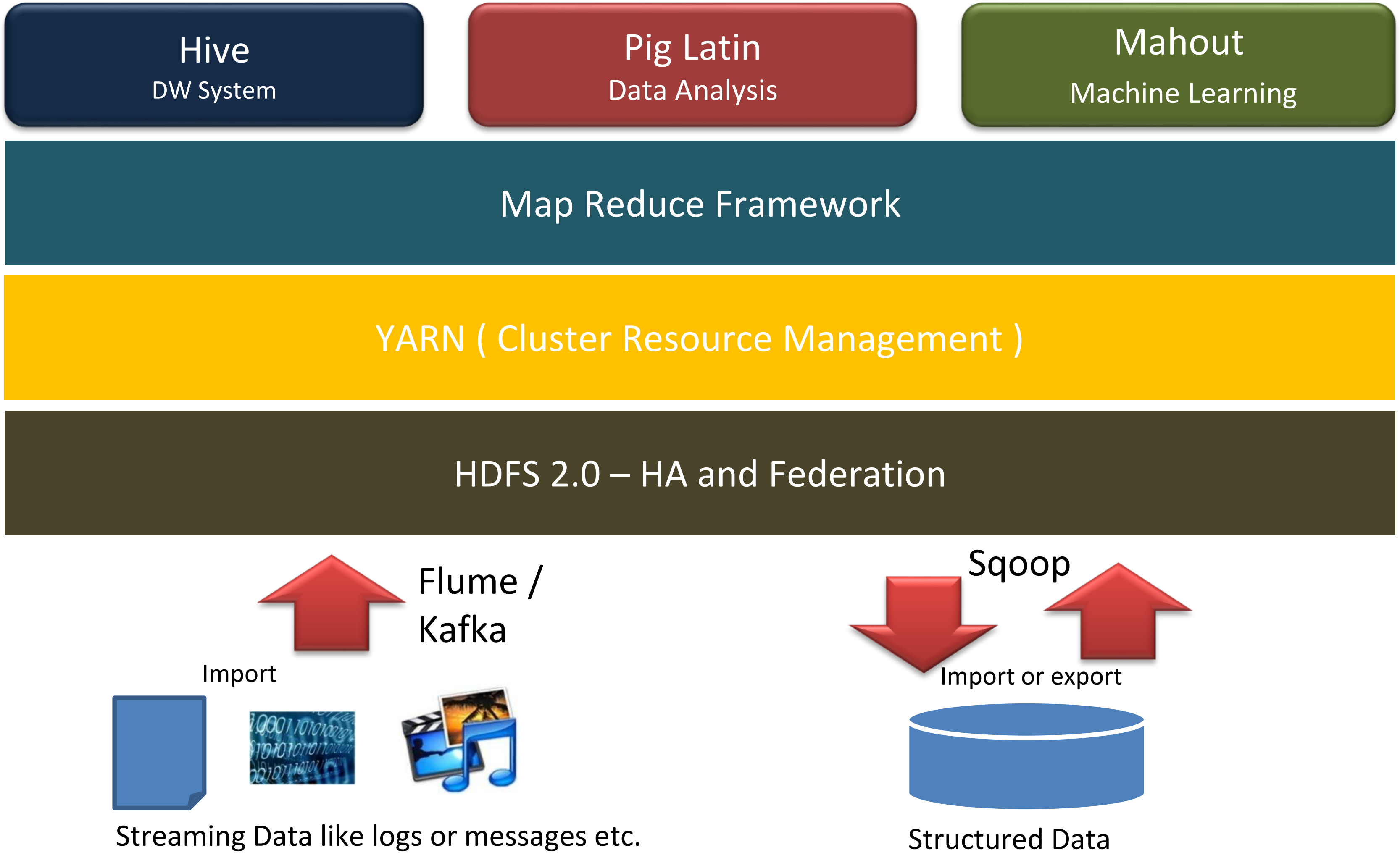
where P is the percentage of ILI physician visits, Q is the ILI-related query fraction, β_0 is the intercept,

Data Analysis - Life Cycle

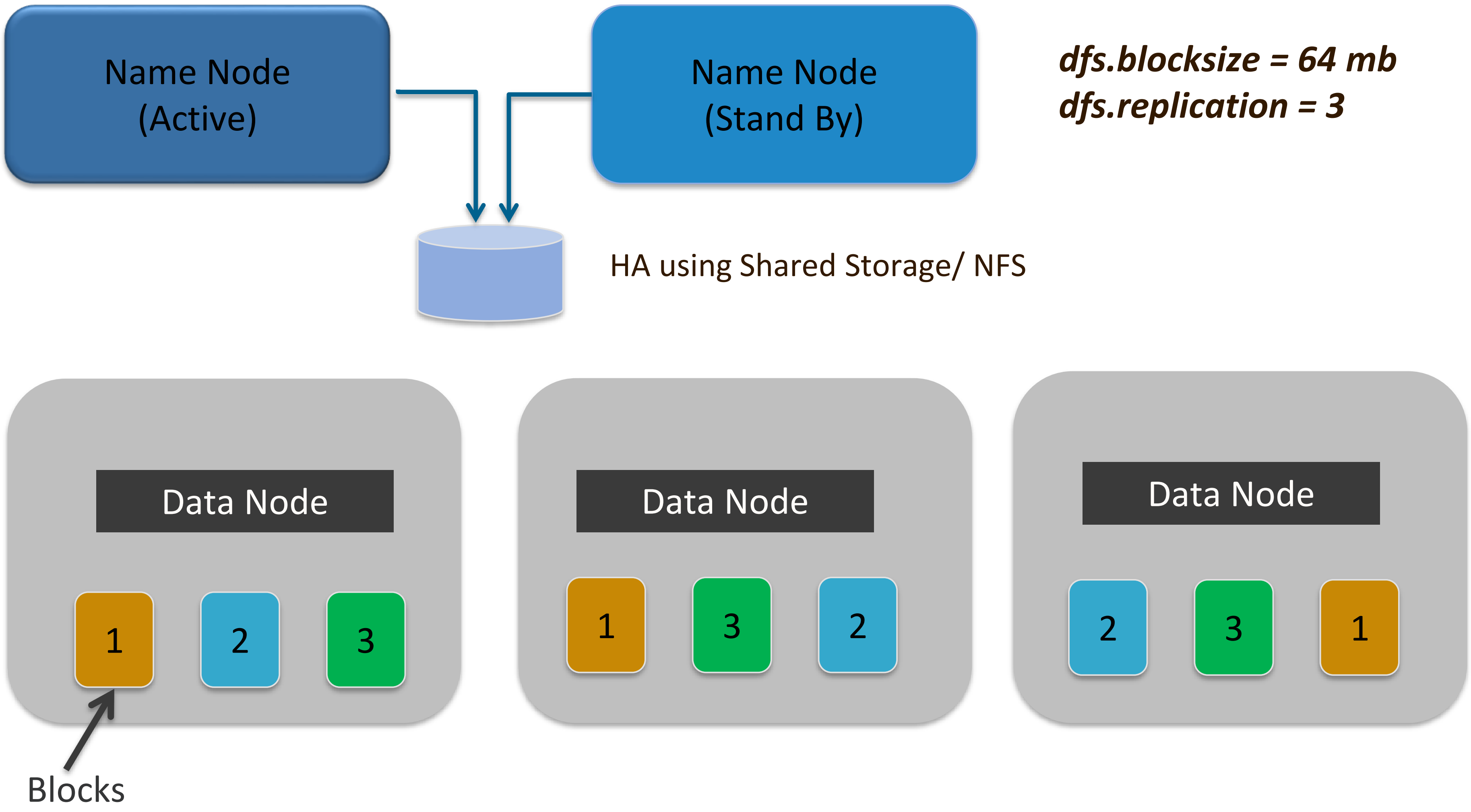


You can have a big data problem in any of the stages?

Hadoop Ecosystem



HDFS – Hadoop Distributed File System



Name Node (Active)

Name Node (Stand By)

dfs.blocksize = 64 mb
dfs.replication = 3

HA using Shared Storage/ NFS

Data Node

1 2 3

Data Node

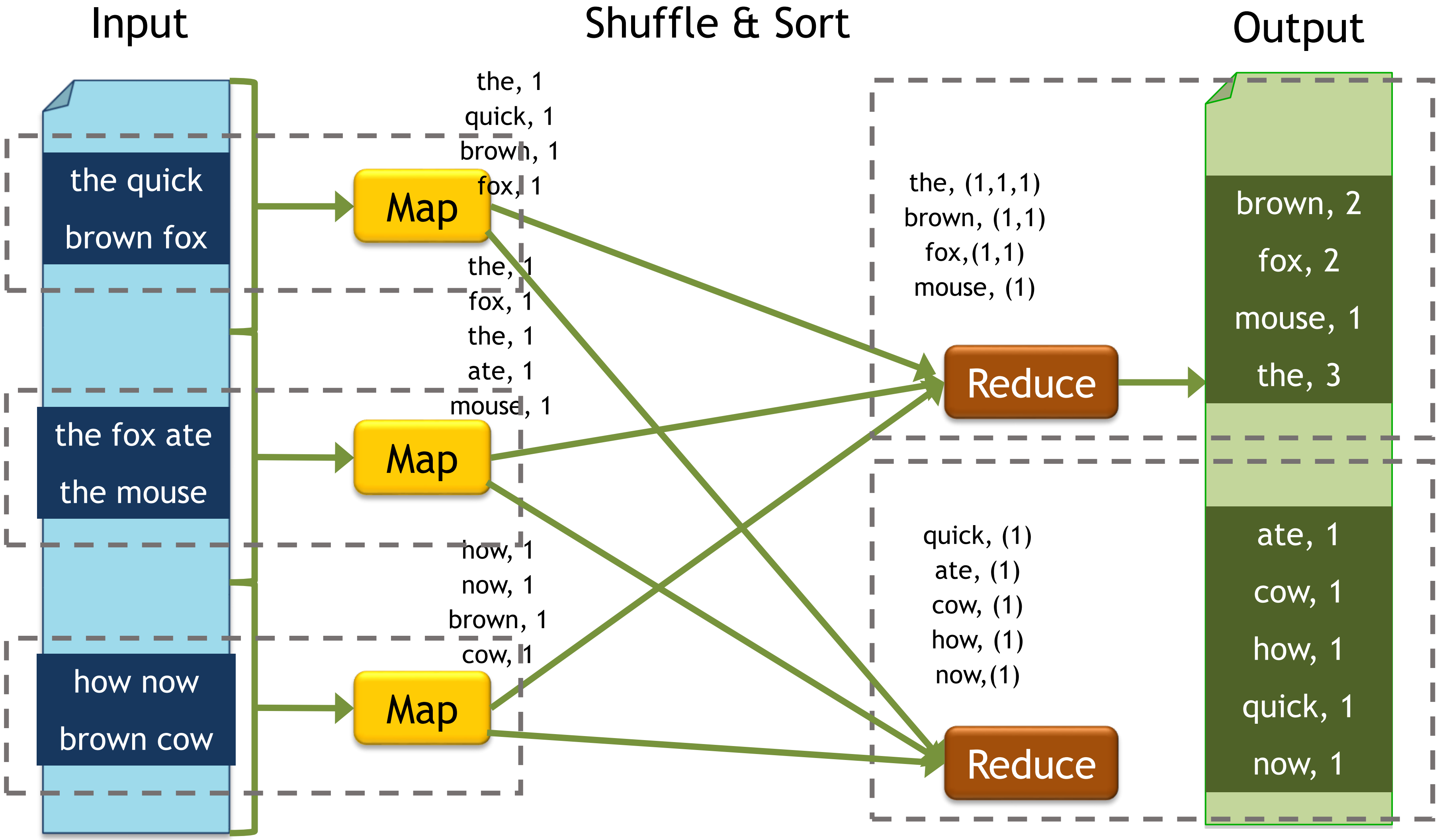
1 3 2

Data Node

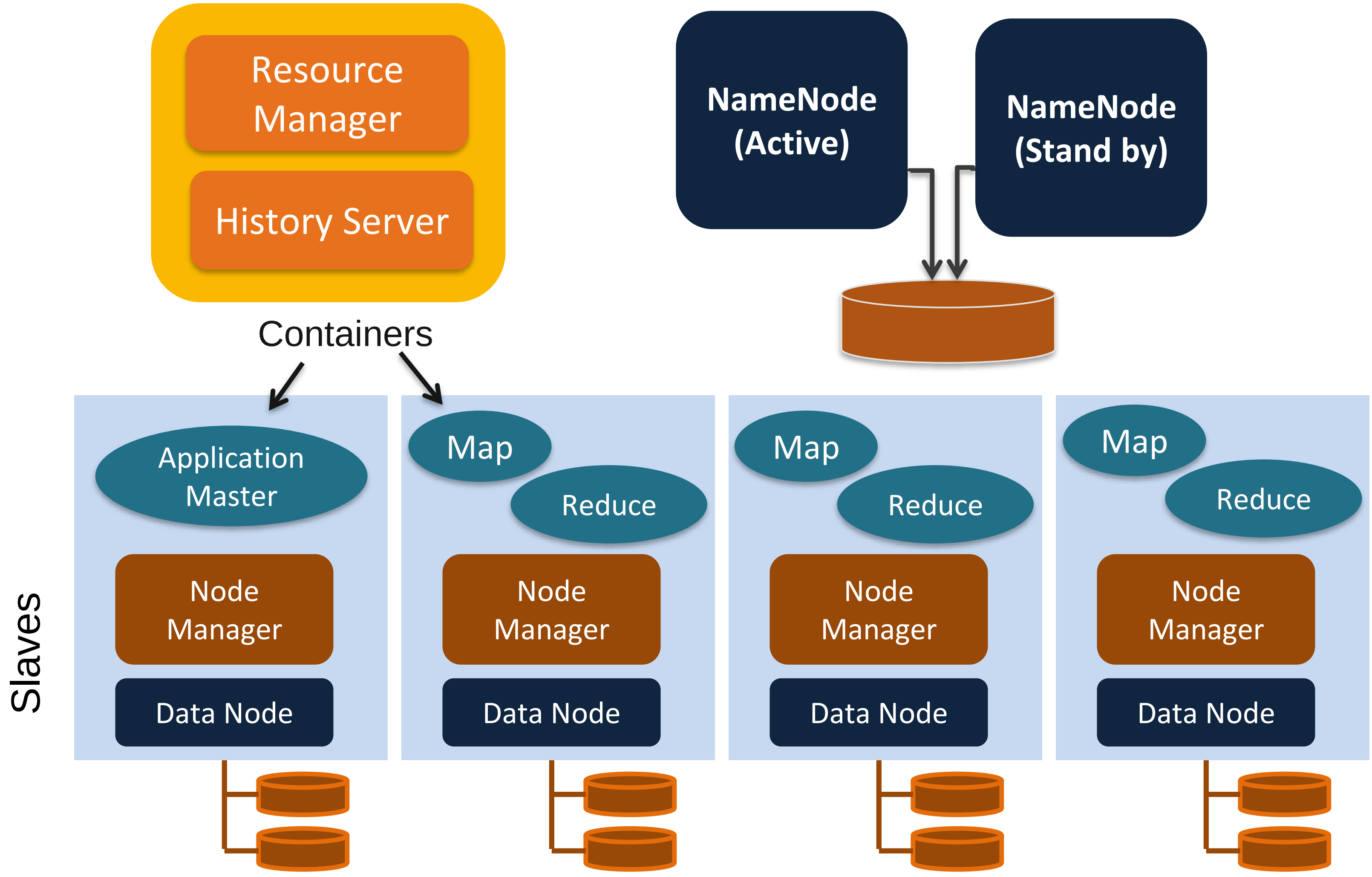
2 3 1

Blocks

How Map Reduce works?



YARN Architecture



Pros & Cons of Hadoop

PROS

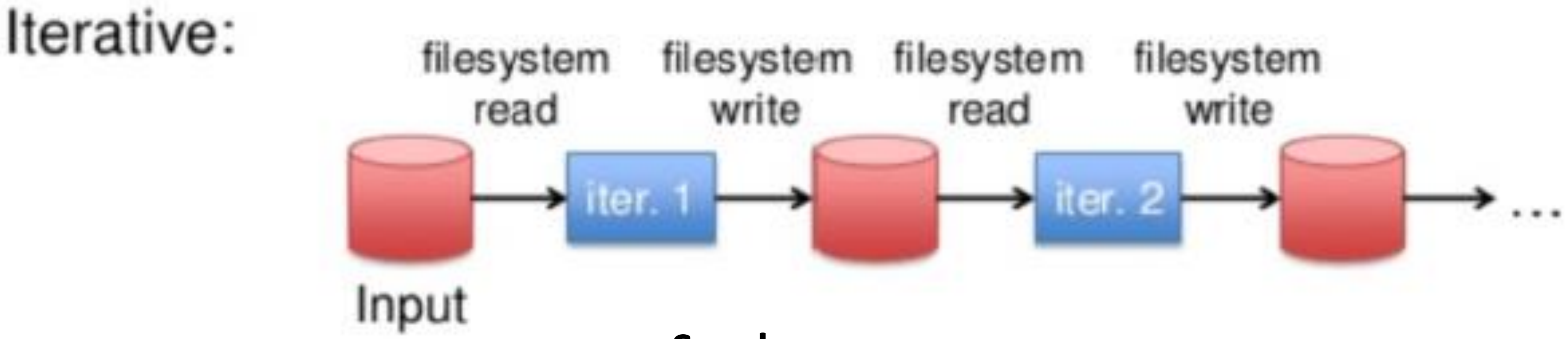
- HDFS is a highly scalable and available platform
- YARN is well designed for resource management
- Hadoop Map Reduce is good for heavy-lifting, required for data cleansing and preparation

CONS

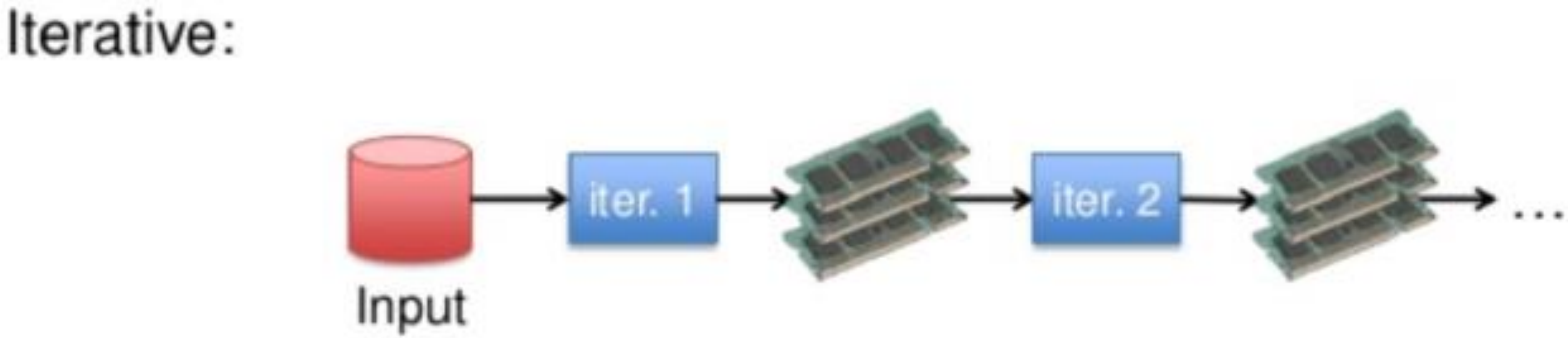
- Map Reduce is slow for multi-stage and iterative algorithms
- Not suitable for advanced machine learning algorithms
- APIs low level and required a lot of coding, even for simple tasks

SPARK Vs. Map Reduce

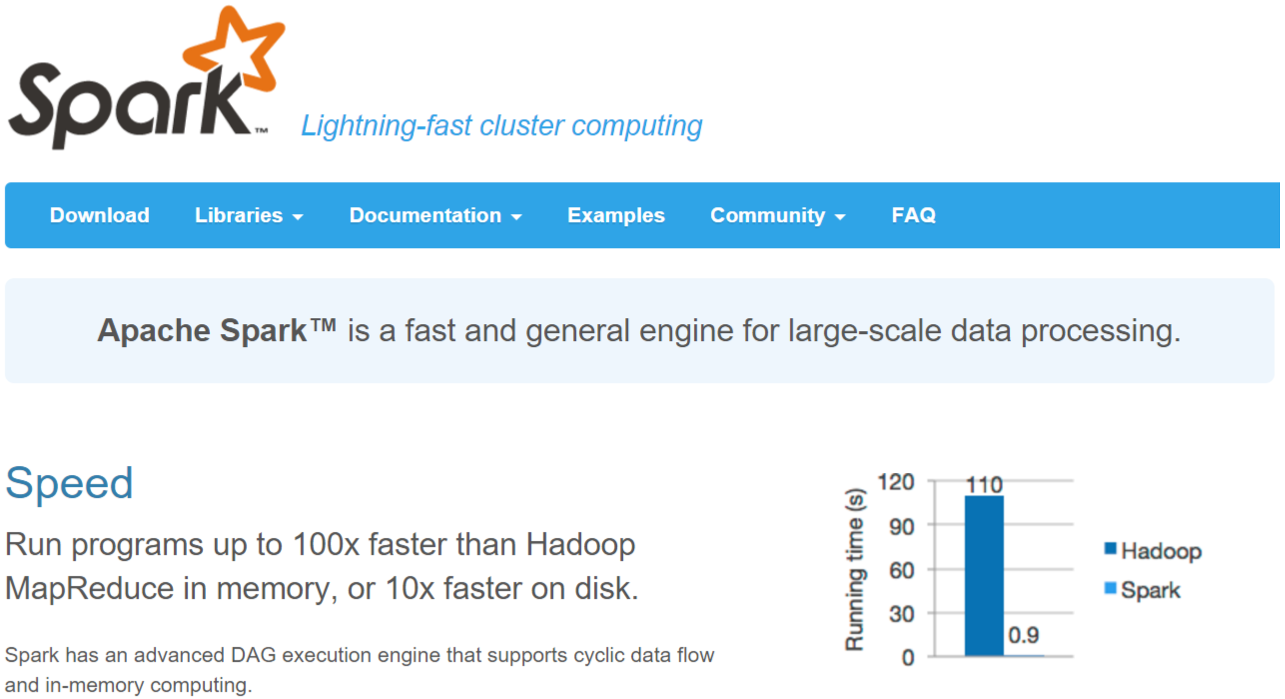
Map Reduce



Spark



Spark Arrives!



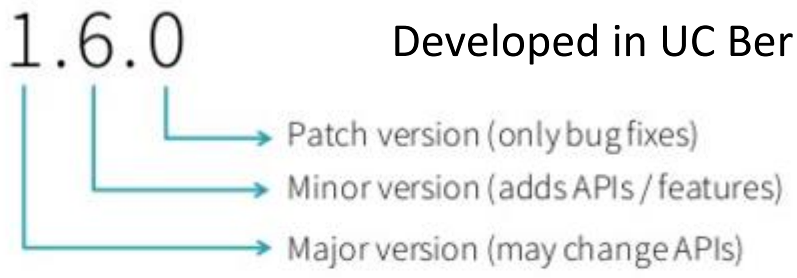
Lightning-fast cluster computing

Apache Spark™ is a fast and general engine for large-scale data processing.

Speed

Run programs up to 100x faster than Hadoop MapReduce in memory, or 10x faster on disk.

Spark has an advanced DAG execution engine that supports cyclic data flow and in-memory computing.



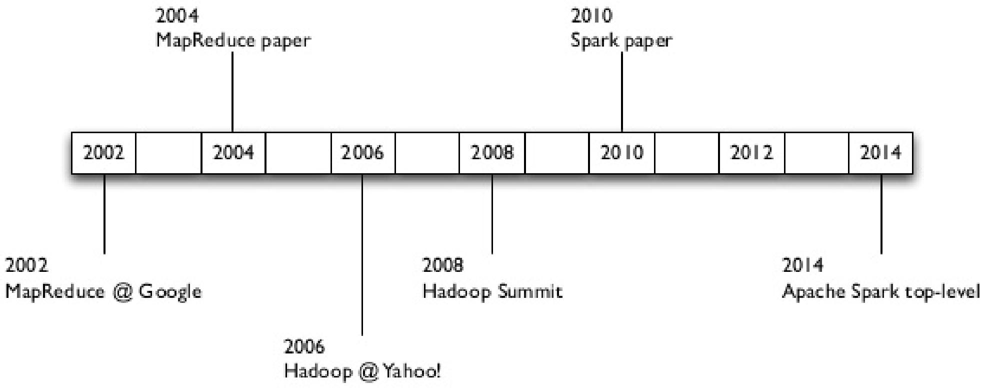
1.6.0

Patch version (only bug fixes)

Minor version (adds APIs / features)

Major version (may change APIs)

Developed in UC Berkeley AMPLab

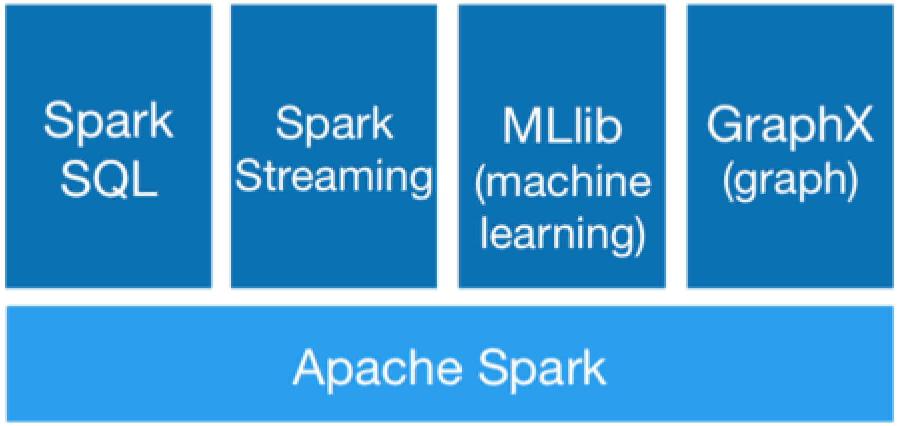


Timeline	Events
2010	Spark Paper
2013	Spark 0.7 Apache Incubator
2014 May	Spark 1.0
2014 Dec	Spark 1.2
2015 June	Spark 1.4
2015 Dec	Spark 1.5
2016 Jan	Spark 1.6

Spark Overview

Spark Software Stack

- Spark Core
  - Core Execution Engine & RDDs
- Spark SQL
  - DataFrames & SQL Queries
- Spark Streaming
  - Streaming Data Analysis
- MLIB
  - Machine Learning Library
- GraphX
  - Graph algorithm & Network Analysis



Spark SQL

Spark Streaming

MLlib (machine learning)

GraphX (graph)

Apache Spark

Spark APIs

- Python
- Java
- Scala
- R

Where Spark can run?

- Spark can run
  - Local Mode
  - Standalone Mode
  - YARN (Hadoop)
  - Mesos

Spark can access data from?

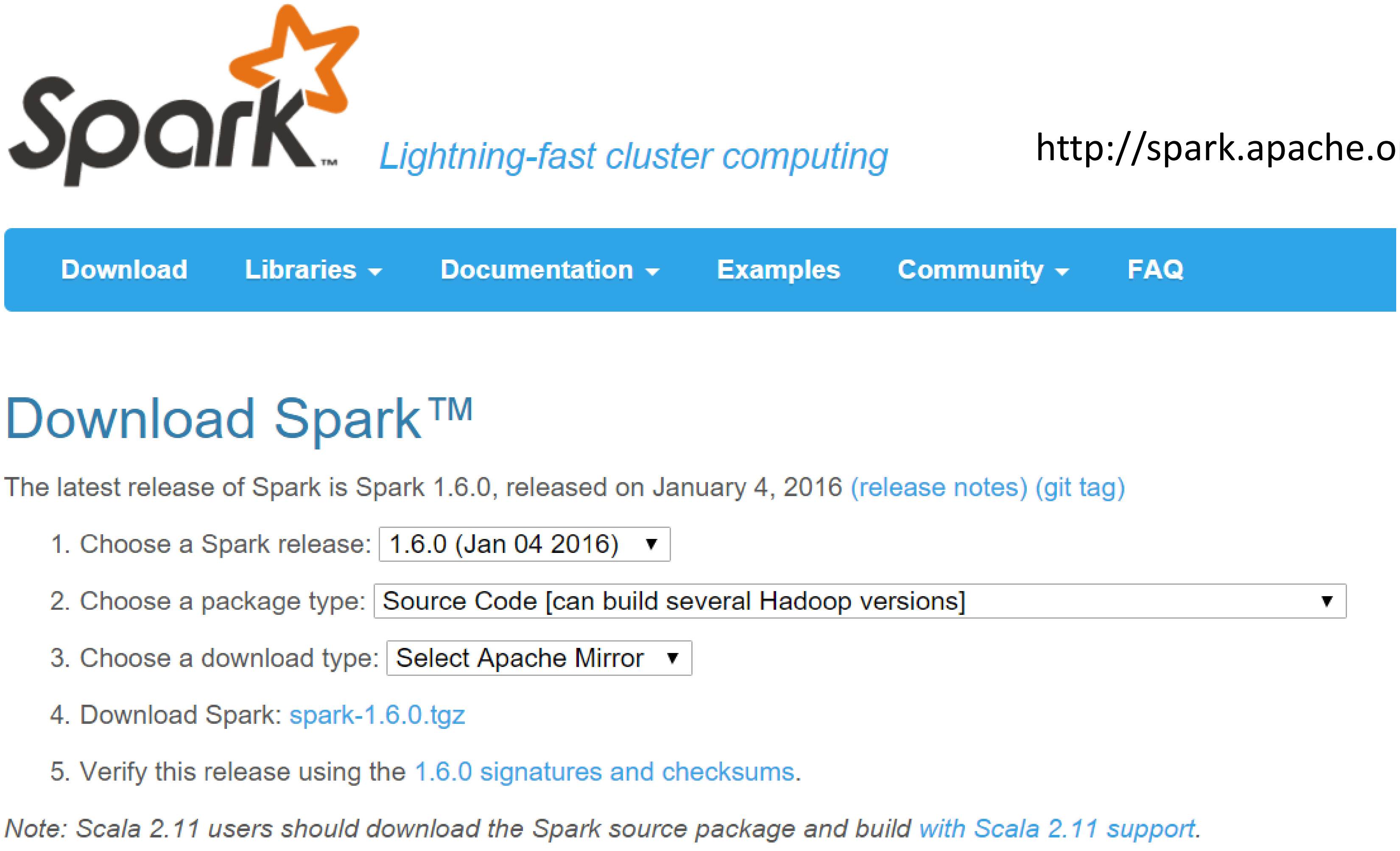
- Spark can access from multiple sources
  - Local File System
  - HDFS
  - Cassandra
  - HBase
  - Hive
  - RDBMS – Mysql, Postgres etc.
  - Tachyon

Discovering Spark

- Links
  - https://spark.apache.org/ (Apache Spark Site)
  - http://spark.apache.org/docs/latest/api.html (Spark APIs for all languages)
- Lab Exercises
  - https://github.com/manaranjanp/spark-dev-training
  - *Fork it or download all in zip format*
- Books
  - Learning Spark: Lightning-Fast Big Data Analysis 1st Edition by Holden Karau (Author), Andy Konwinski (Author), Patrick Wendell (Author), Matei Zaharia (Author)
    - http://www.amazon.com/Learning-Spark-Lightning-Fast-Data-Analysis/dp/1449358624/ref=sr_1_1?ie=UTF8&qid=1457069582&sr=8-1&keywords=apache+spark
  - Mastering Apache Spark Paperback – September 30, 2015 by Mike Frampton (Author)
    - http://www.amazon.com/Mastering-Apache-Spark-Mike-Frampton/dp/1783987146/ref=sr_1_3?ie=UTF8&qid=1457069582&sr=8-3&keywords=apache+spark

Downloading Spark

http://spark.apache.org/downloads.html



Lightning-fast cluster computing

Download Spark™

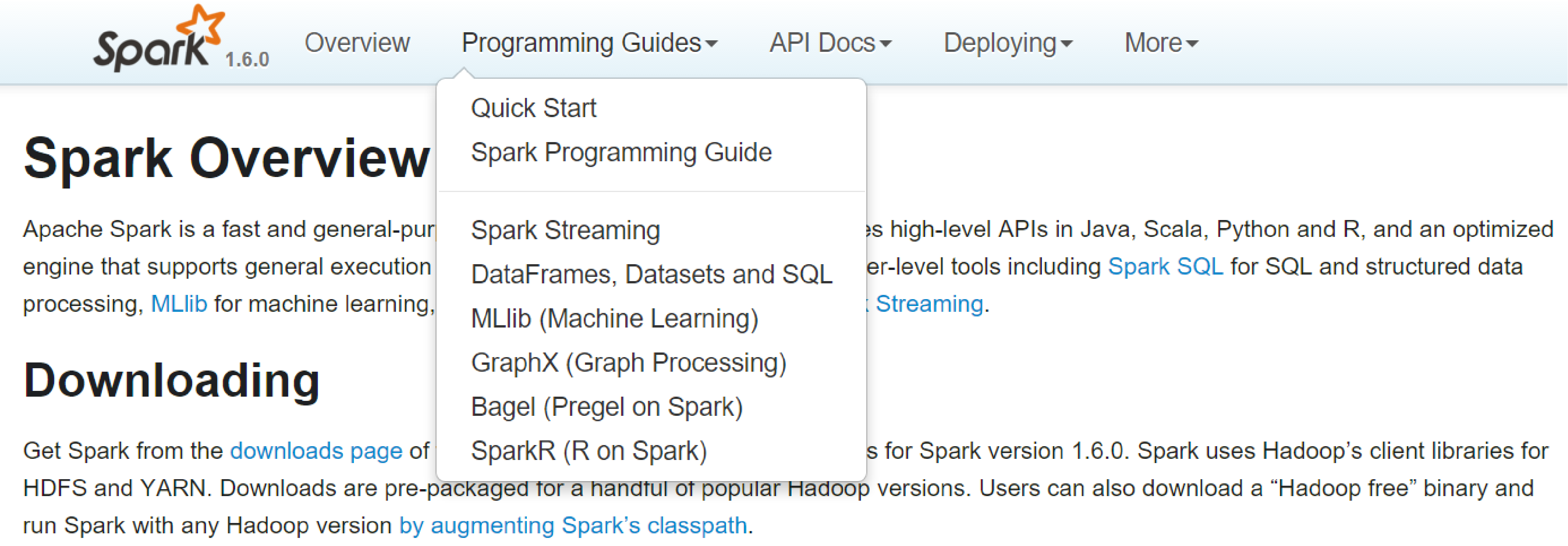
The latest release of Spark is Spark 1.6.0, released on January 4, 2016 (release notes) (git tag)

1. Choose a Spark release: 1.6.0 (Jan 04 2016) ▾
2. Choose a package type: Source Code [can build several Hadoop versions] ▾
3. Choose a download type: Select Apache Mirror ▾
4. Download Spark: spark-1.6.0.tgz
5. Verify this release using the 1.6.0 signatures and checksums.

Note: Scala 2.11 users should download the Spark source package and build with Scala 2.11 support.

Spark Documentation

http://spark.apache.org/docs/latest/



Spark Overview

Apache Spark is a fast and general-pur es high-level APIs in Java, Scala, Python and R, and an optimized engine that supports general execution er-level tools including Spark SQL for SQL and structured data processing, MLlib for machine learning, Streaming.

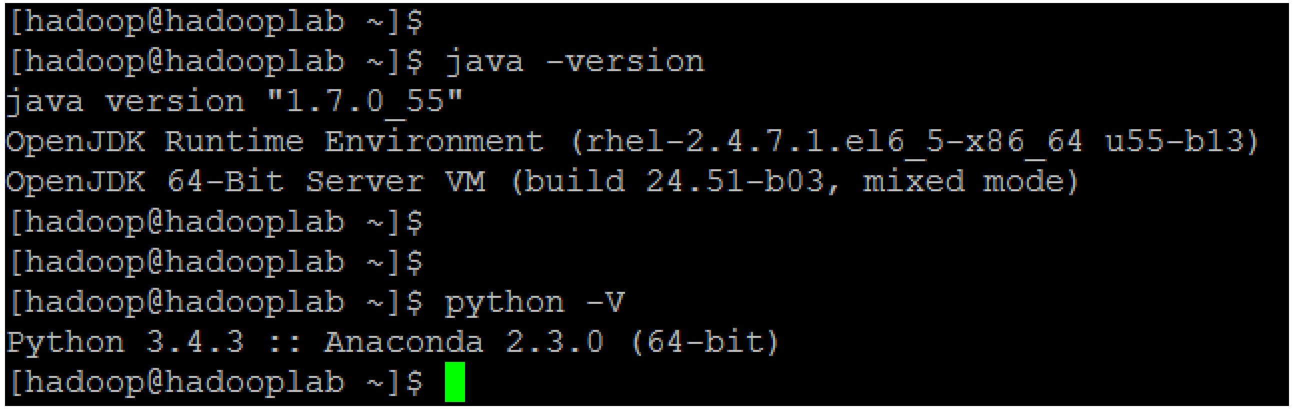
Downloading

Get Spark from the downloads page of s for Spark version 1.6.0. Spark uses Hadoop's client libraries for HDFS and YARN. Downloads are pre-packaged for a handful of popular Hadoop versions. Users can also download a "Hadoop free" binary and run Spark with any Hadoop version by augmenting Spark's classpath.

Browse through multiple section to make participants familiar with different sections

Installing & Configuring Spark

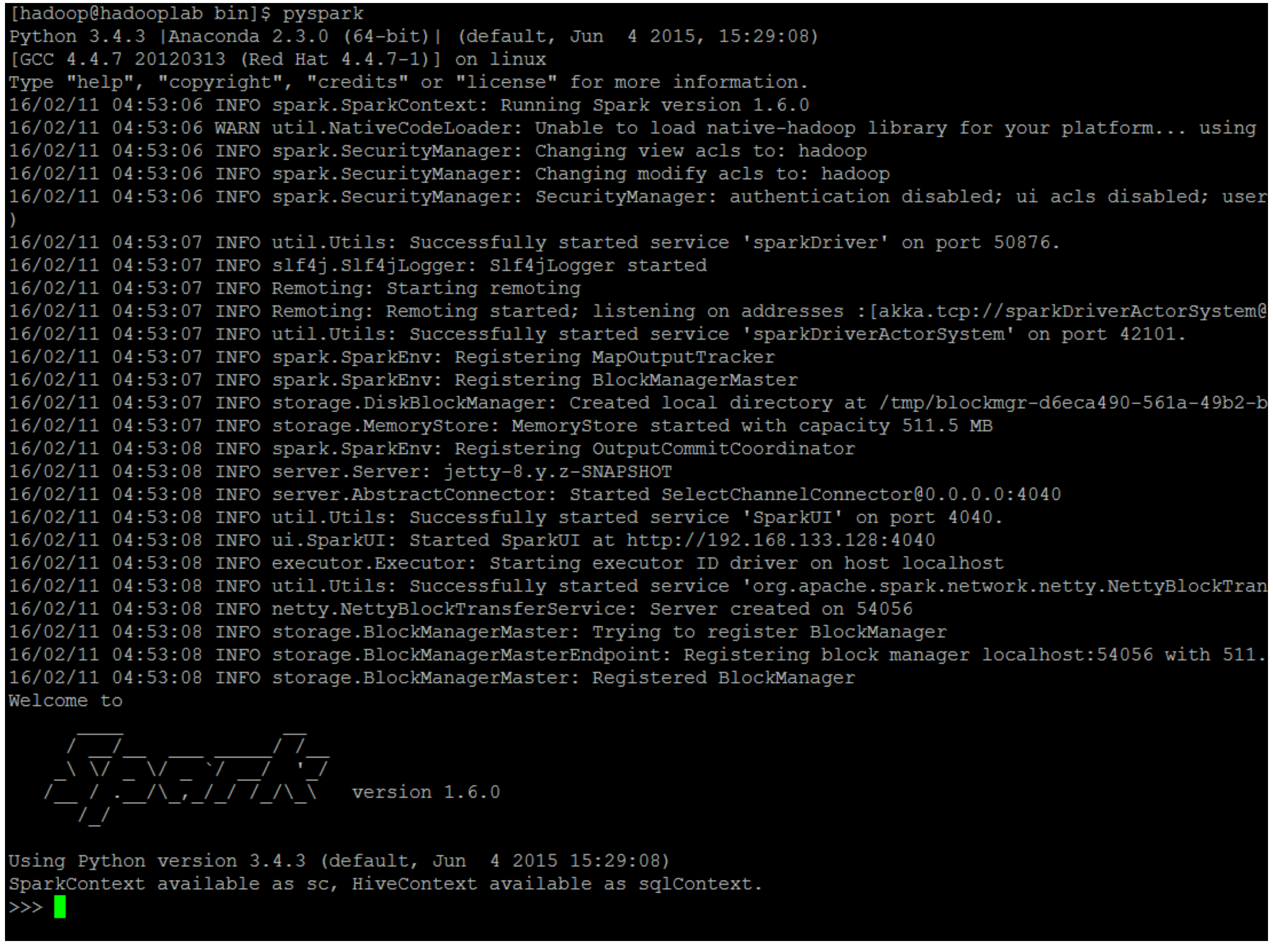
- Pre-requisites
  - Java
  - Python
  - Scala
  - R



```
[hadoop@hadooplab ~]$
[hadoop@hadooplab ~]$ java -version
java version "1.7.0_55"
OpenJDK Runtime Environment (rhel-2.4.7.1.el6_5-x86_64 u55-b13)
OpenJDK 64-Bit Server VM (build 24.51-b03, mixed mode)
[hadoop@hadooplab ~]$
[hadoop@hadooplab ~]$
[hadoop@hadooplab ~]$ python -V
Python 3.4.3 :: Anaconda 2.3.0 (64-bit)
[hadoop@hadooplab ~]$
```

- Un-tar for jar file
  - **tar -xvf /home/hadoop/lab/downloads/spark-1.6.0-bin-hadoop2.6.tgz**

Working with Spark Shell



```
[hadoop@hadooplab bin]$ pyspark
Python 3.4.3 |Anaconda 2.3.0 (64-bit)| (default, Jun  4 2015, 15:29:08)
[GCC 4.4.7 20120313 (Red Hat 4.4.7-1)] on linux
Type "help", "copyright", "credits" or "license" for more information.
16/02/11 04:53:06 INFO spark.SparkContext: Running Spark version 1.6.0
16/02/11 04:53:06 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using
16/02/11 04:53:06 INFO spark.SecurityManager: Changing view acls to: hadoop
16/02/11 04:53:06 INFO spark.SecurityManager: Changing modify acls to: hadoop
16/02/11 04:53:06 INFO spark.SecurityManager: SecurityManager: authentication disabled; ui acls disabled; user
)
16/02/11 04:53:07 INFO util.Utils: Successfully started service 'sparkDriver' on port 50876.
16/02/11 04:53:07 INFO slf4j.Slf4jLogger: Slf4jLogger started
16/02/11 04:53:07 INFO Remoting: Starting remoting
16/02/11 04:53:07 INFO Remoting: Remoting started; listening on addresses :[akka.tcp://sparkDriverActorSystem@
16/02/11 04:53:07 INFO util.Utils: Successfully started service 'sparkDriverActorSystem' on port 42101.
16/02/11 04:53:07 INFO spark.SparkEnv: Registering MapOutputTracker
16/02/11 04:53:07 INFO spark.SparkEnv: Registering BlockManagerMaster
16/02/11 04:53:07 INFO storage.DiskBlockManager: Created local directory at /tmp/blockmgr-d6eca490-561a-49b2-b
16/02/11 04:53:07 INFO storage.MemoryStore: MemoryStore started with capacity 511.5 MB
16/02/11 04:53:08 INFO spark.SparkEnv: Registering OutputCommitCoordinator
16/02/11 04:53:08 INFO server.Server: jetty-8.y.z-SNAPSHOT
16/02/11 04:53:08 INFO server.AbstractConnector: Started SelectChannelConnector@0.0.0.0:4040
16/02/11 04:53:08 INFO util.Utils: Successfully started service 'SparkUI' on port 4040.
16/02/11 04:53:08 INFO ui.SparkUI: Started SparkUI at http://192.168.133.128:4040
16/02/11 04:53:08 INFO executor.Executor: Starting executor ID driver on host localhost
16/02/11 04:53:08 INFO util.Utils: Successfully started service 'org.apache.spark.network.netty.NettyBlockTran
16/02/11 04:53:08 INFO netty.NettyBlockTransferService: Server created on 54056
16/02/11 04:53:08 INFO storage.BlockManagerMaster: Trying to register BlockManager
16/02/11 04:53:08 INFO storage.BlockManagerMasterEndpoint: Registering block manager localhost:54056 with 511.
16/02/11 04:53:08 INFO storage.BlockManagerMaster: Registered BlockManager
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 1.6.0
      /_/

Using Python version 3.4.3 (default, Jun  4 2015 15:29:08)
SparkContext available as sc, HiveContext available as sqlContext.
>>>
```

Start linux shell and type at the command prompt

$ pyspark <enter>

It displays spark and python version

Running Spark on Local Mode

- spark-shell --master local
  - Runs spark in local mode with only one executor

- spark-shell --master local[4]
  - Runs spark in local mode with 4 executors

- spark-shell --master local[*]
  - Run spark in local mode and as many executors as there are cores

Writing first spark program

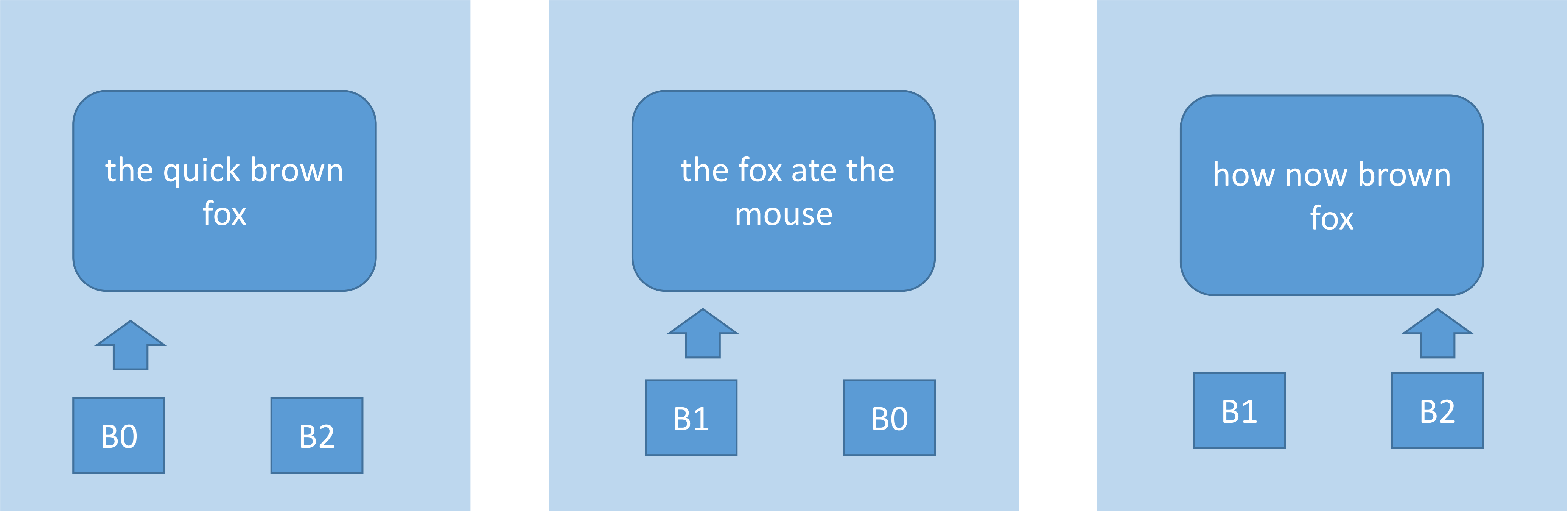
```
var wordfile = sc.textFile( "file:///home/hadoop/lab/data/words")
var words = wordfile.flatMap( line => line.split( " " ) )
words.take( 10 ).foreach( println )
var word_one = words.map( word => ( word, 1 ) )
var word_counts = word_one.reduceByKey( _+_ )
word_counts.take( 10 ).foreach( println )
```

Spark Core Component

- RDDs (Resilient Distributed Datasets)
  - Fault tolerant collection of data elements
  - Data is split and distributed
  - Parallel operation can be applied to RDDs
  - RDDs can be created
    - In memory collections
    - Reading from non-distributed data sources and parallelizing datasets
    - From external distributed data sources like HDFS, Hive, Cassandra etc.
- June 2010
  - http://www.cs.berkeley.edu/~matei/papers/2010/hotcloud_spark.pdf
- April 2012
  - http://www.cs.berkeley.edu/~matei/papers/2012/nsdi_spark.pdf

RDDs

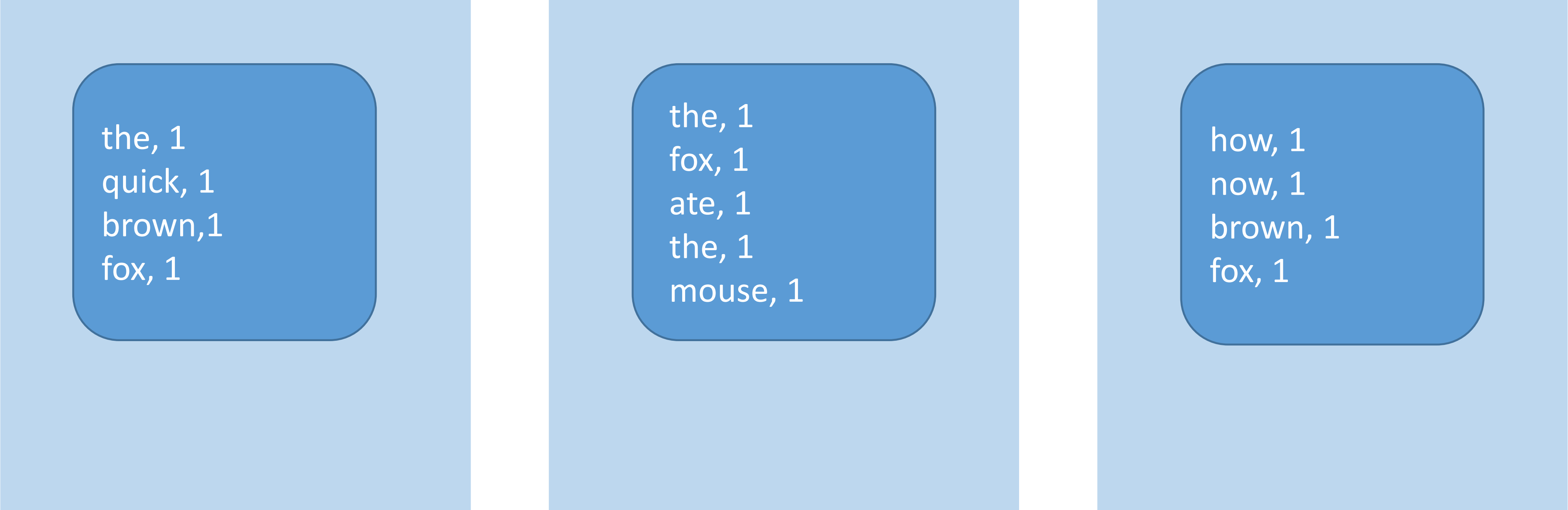
var wordfile = sc.textFile("file:///home/hadoop/lab/data/words")



Each chuck of data (HDFS Blocks) are loaded as one partition of RDD

RDDs

```
var words = wordfile.flatMap( line => line.split( " " ) )
var word_one = words.map( word => ( word, 1 ) )
```



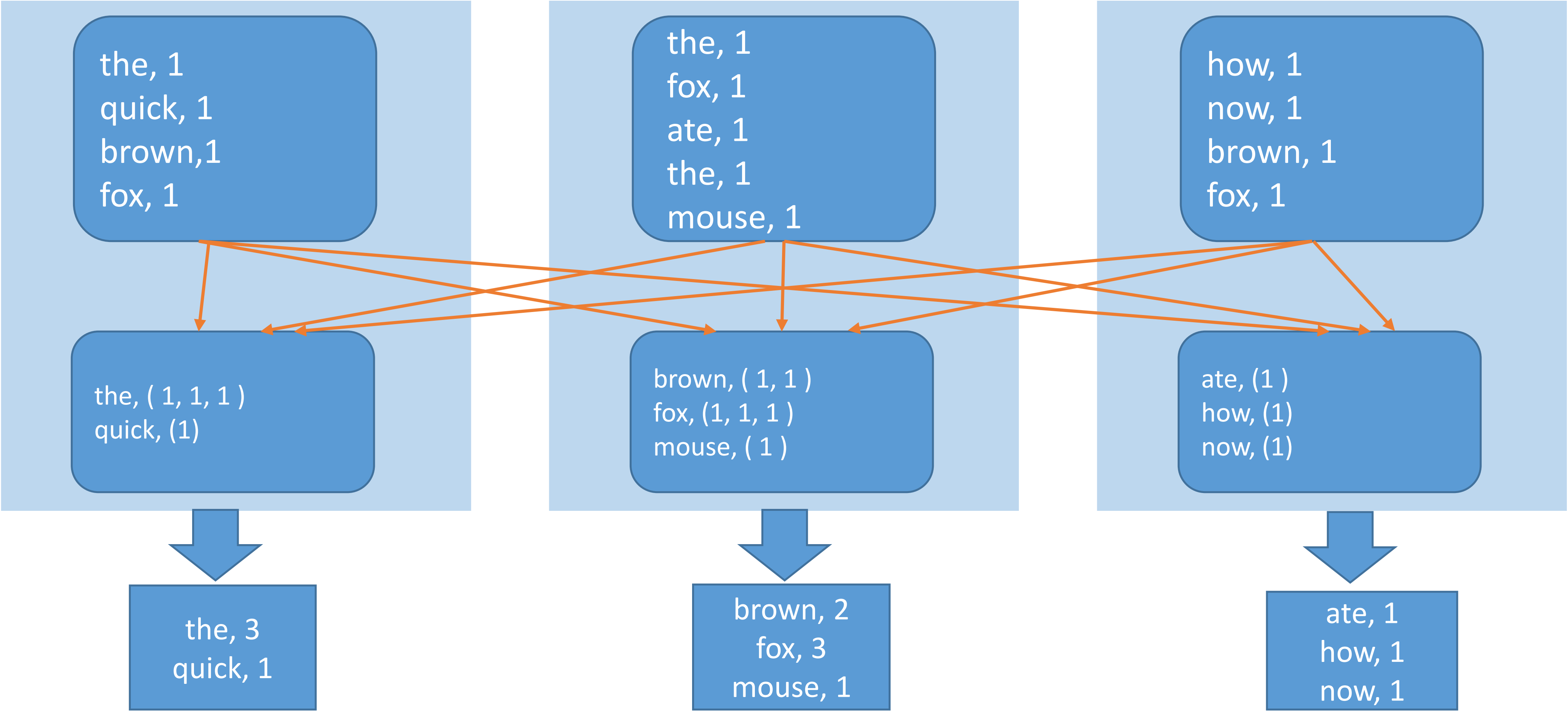
the, 1
quick, 1
brown,1
fox, 1

the, 1
fox, 1
ate, 1
the, 1
mouse, 1

how, 1
now, 1
brown, 1
fox, 1

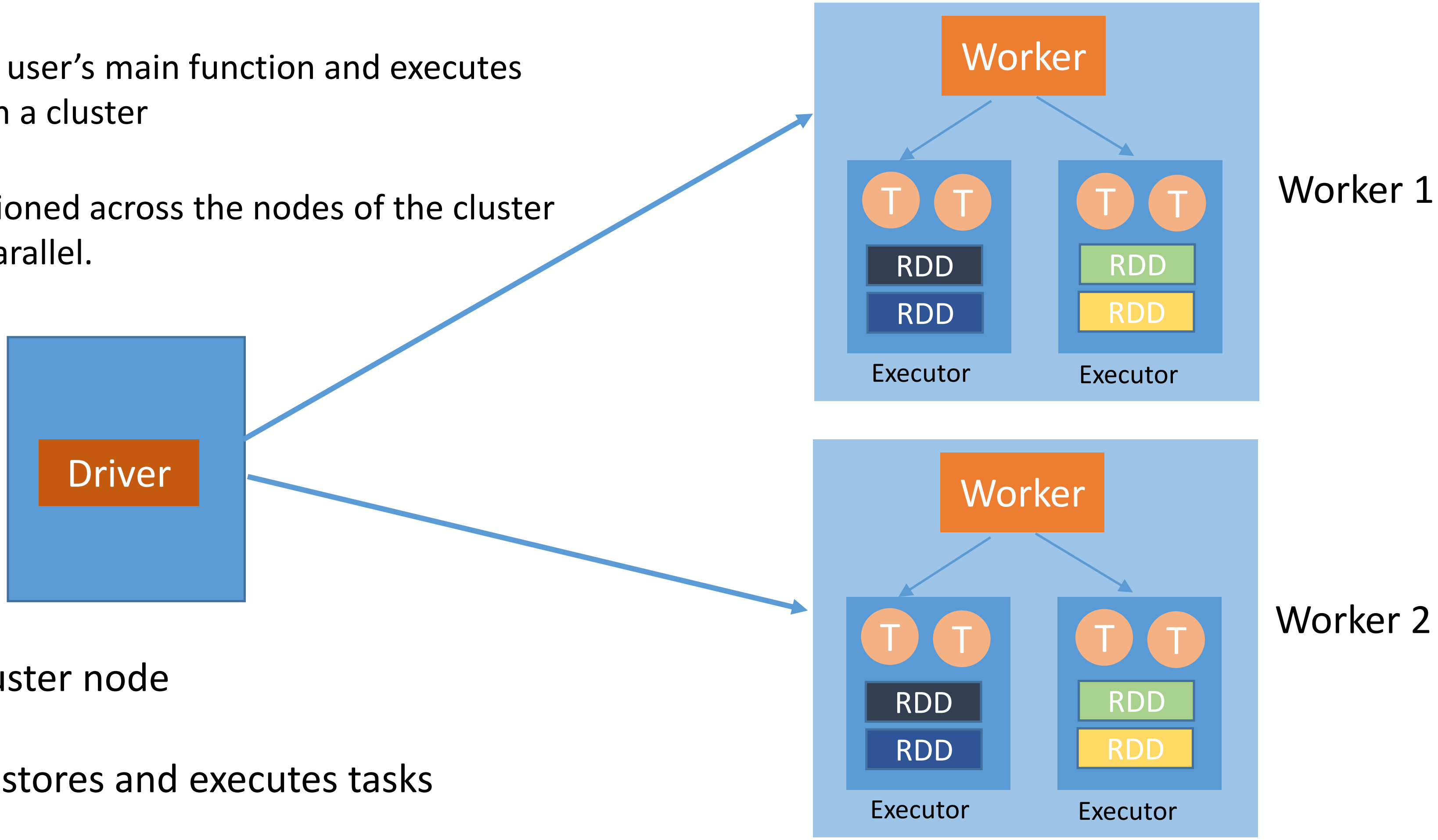
RDDs

var word_counts = word_one.reduceByKey(_+_)



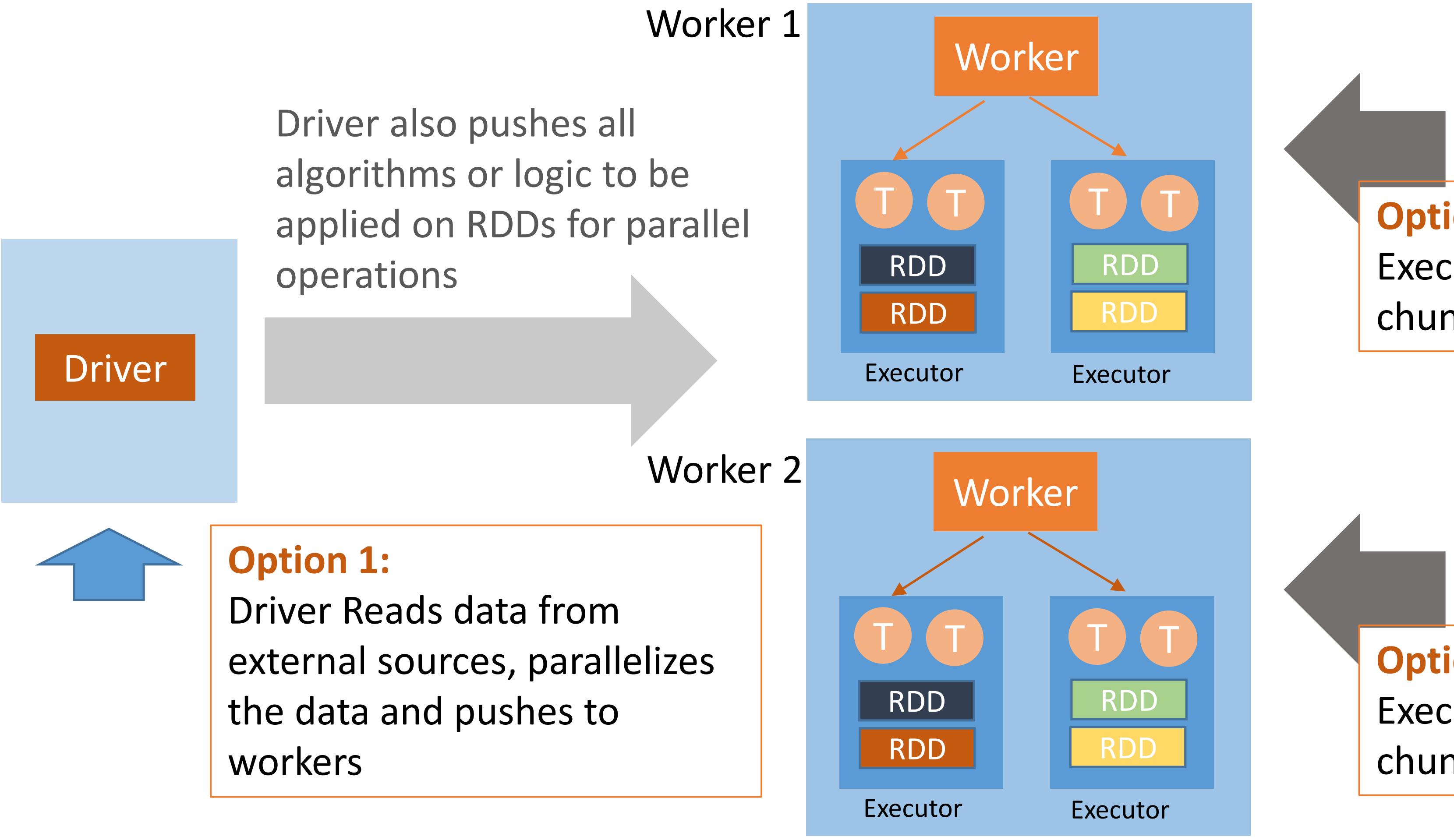
Spark Architecture

- Driver
  - Driver program that runs the user's main function and executes various parallel operations on a cluster
- RDDs
  - Collection of elements partitioned across the nodes of the cluster that can be operated on in parallel.



- Worker
  - Manages resources on cluster node
- Executor
  - Is the JVM process which stores and executes tasks
- Tasks
  - Executes the RDD operations

Reading Data and creating RDDs



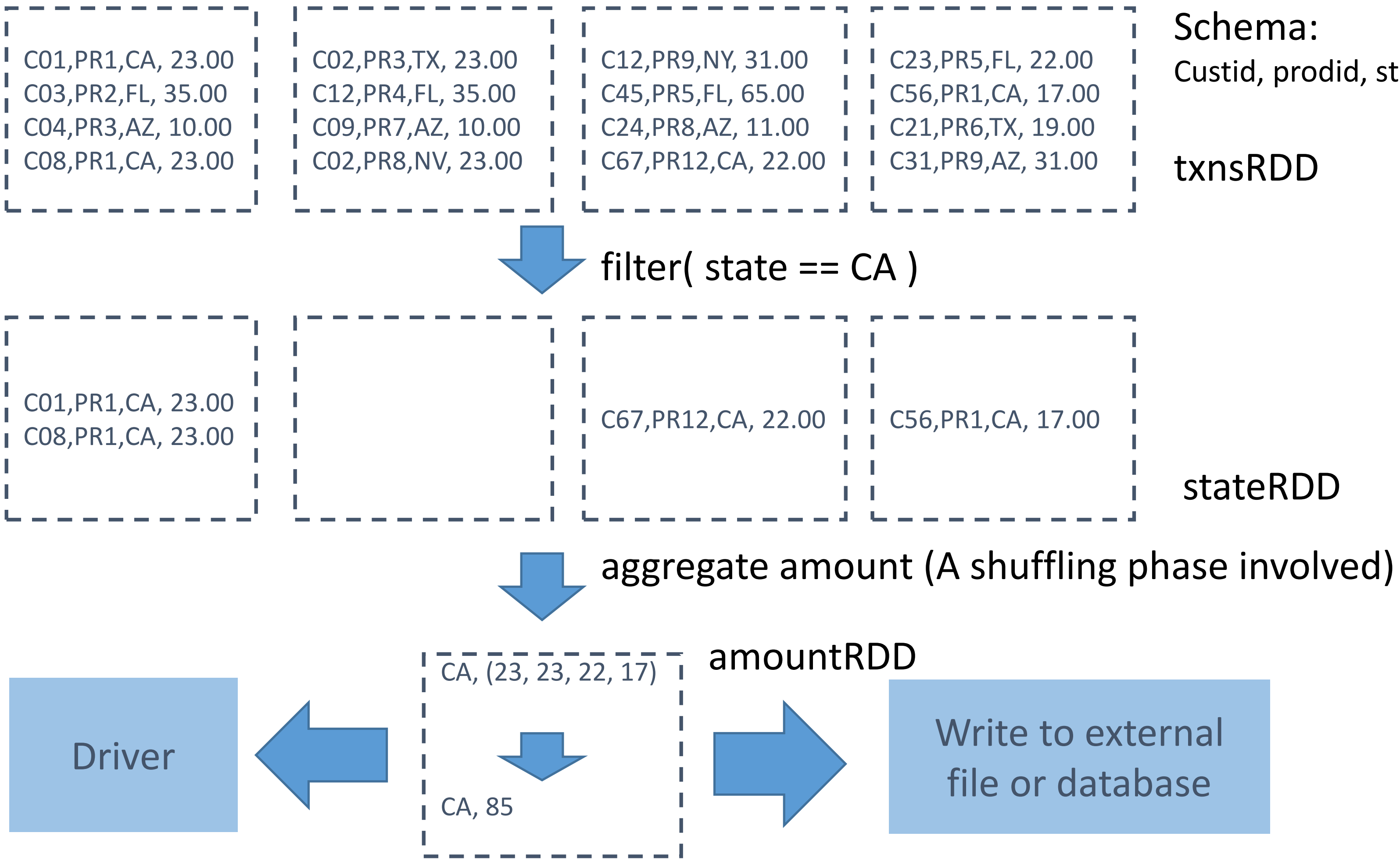
Driver also pushes all algorithms or logic to be applied on RDDs for parallel operations

Option 2:
Executors reads chunks of data

Option 1:
Driver Reads data from external sources, parallelizes the data and pushes to workers

Option 2:
Executors reads chunks of data

RDD - Transformations



Schema:
Custid, prodid, state, amount

txnsRDD

C01,PR1,CA, 23.00
C03,PR2,FL, 35.00
C04,PR3,AZ, 10.00
C08,PR1,CA, 23.00

C02,PR3,TX, 23.00
C12,PR4,FL, 35.00
C09,PR7,AZ, 10.00
C02,PR8,NV, 23.00

C12,PR9,NY, 31.00
C45,PR5,FL, 65.00
C24,PR8,AZ, 11.00
C67,PR12,CA, 22.00

C23,PR5,FL, 22.00
C56,PR1,CA, 17.00
C21,PR6,TX, 19.00
C31,PR9,AZ, 31.00

filter(state == CA)

stateRDD

C01,PR1,CA, 23.00
C08,PR1,CA, 23.00

C67,PR12,CA, 22.00

C56,PR1,CA, 17.00

aggregate amount (A shuffling phase involved)

amountRDD

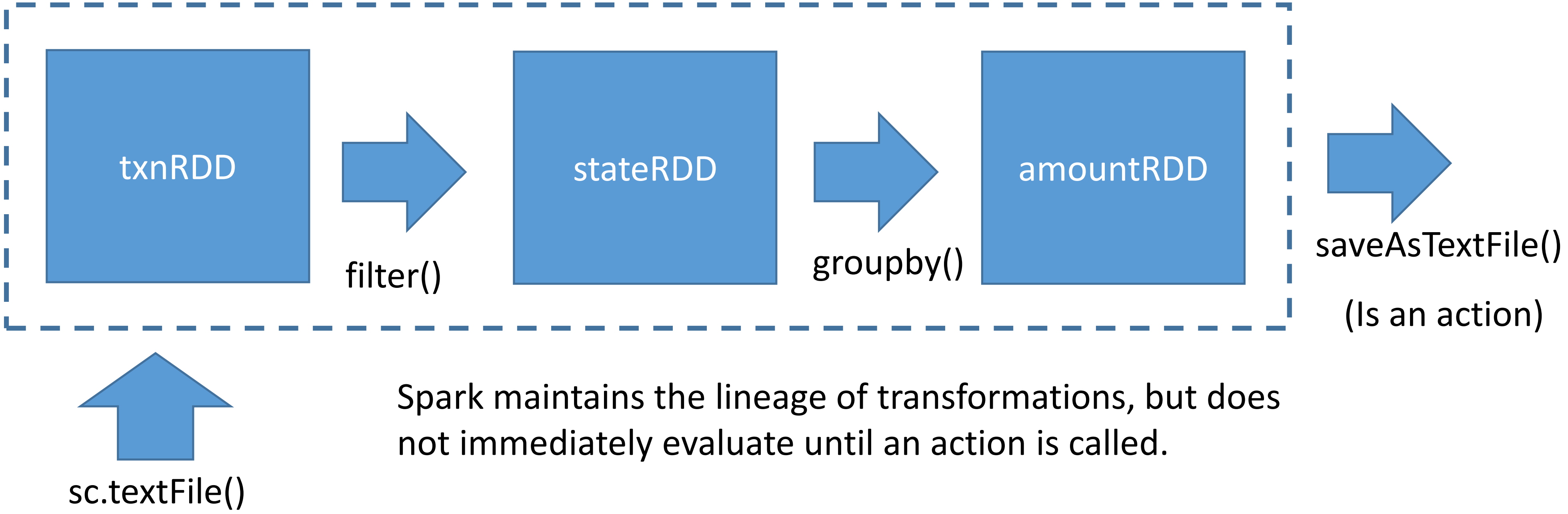
CA, (23, 23, 22, 17)

CA, 85

Driver

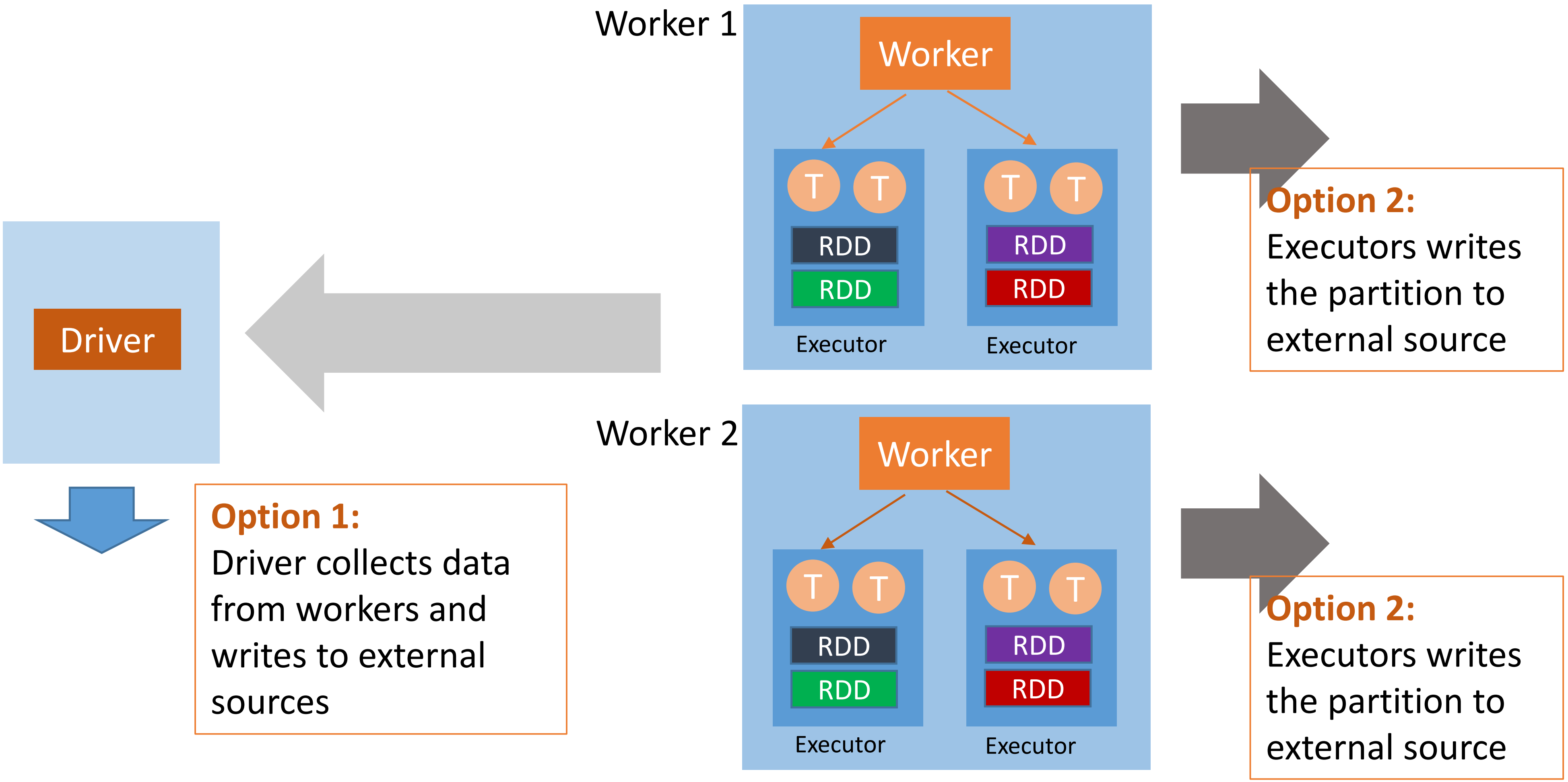
Write to external file or database

Actions & Lazy Evaluation



Spark maintains the lineage of transformations, but does not immediately evaluate until an action is called.

Writing Final Result

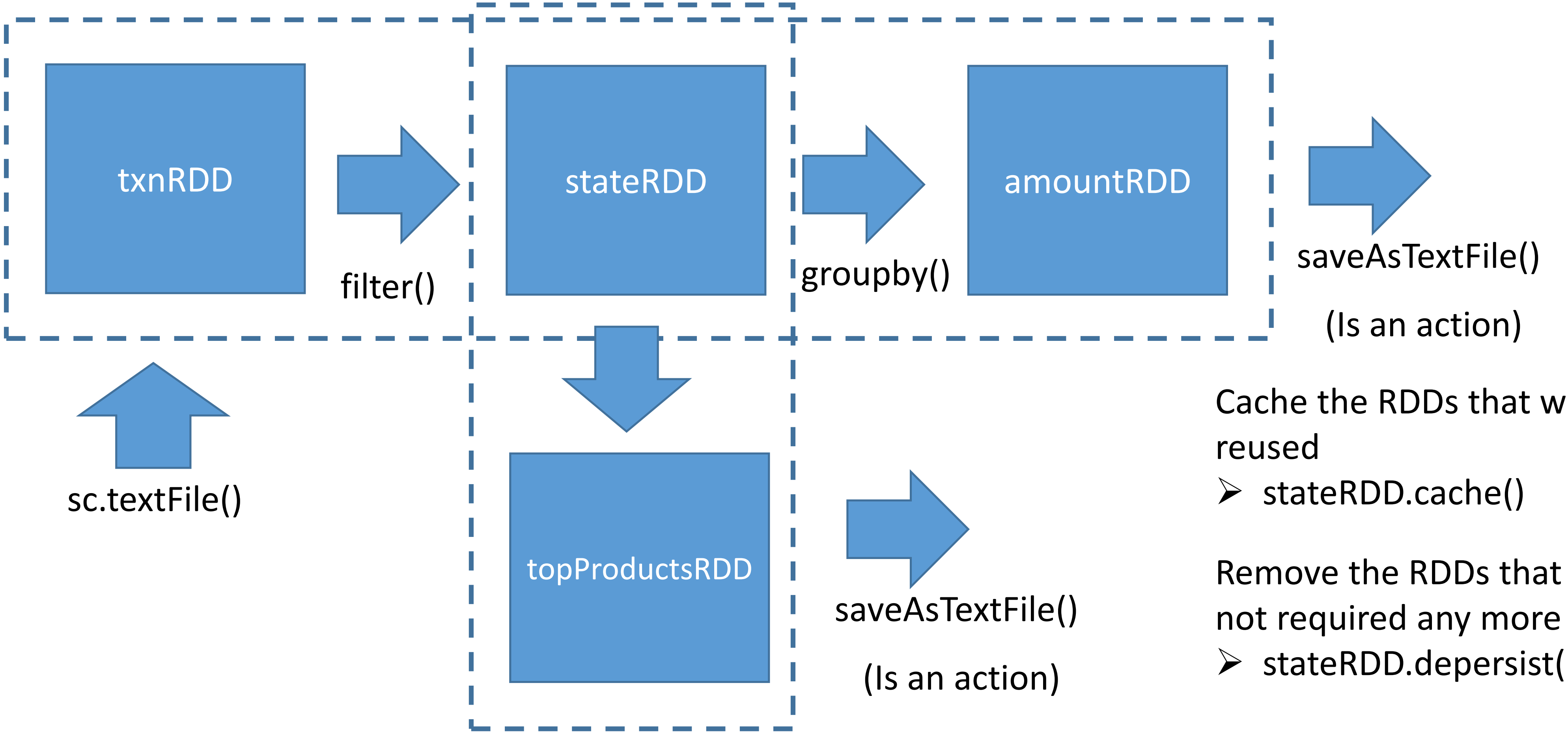


Option 1:
Driver collects data from workers and writes to external sources

Option 2:
Executors writes the partition to external source

Option 2:
Executors writes the partition to external source

Cache, persist & de-persist



Cache the RDDs that will be reused

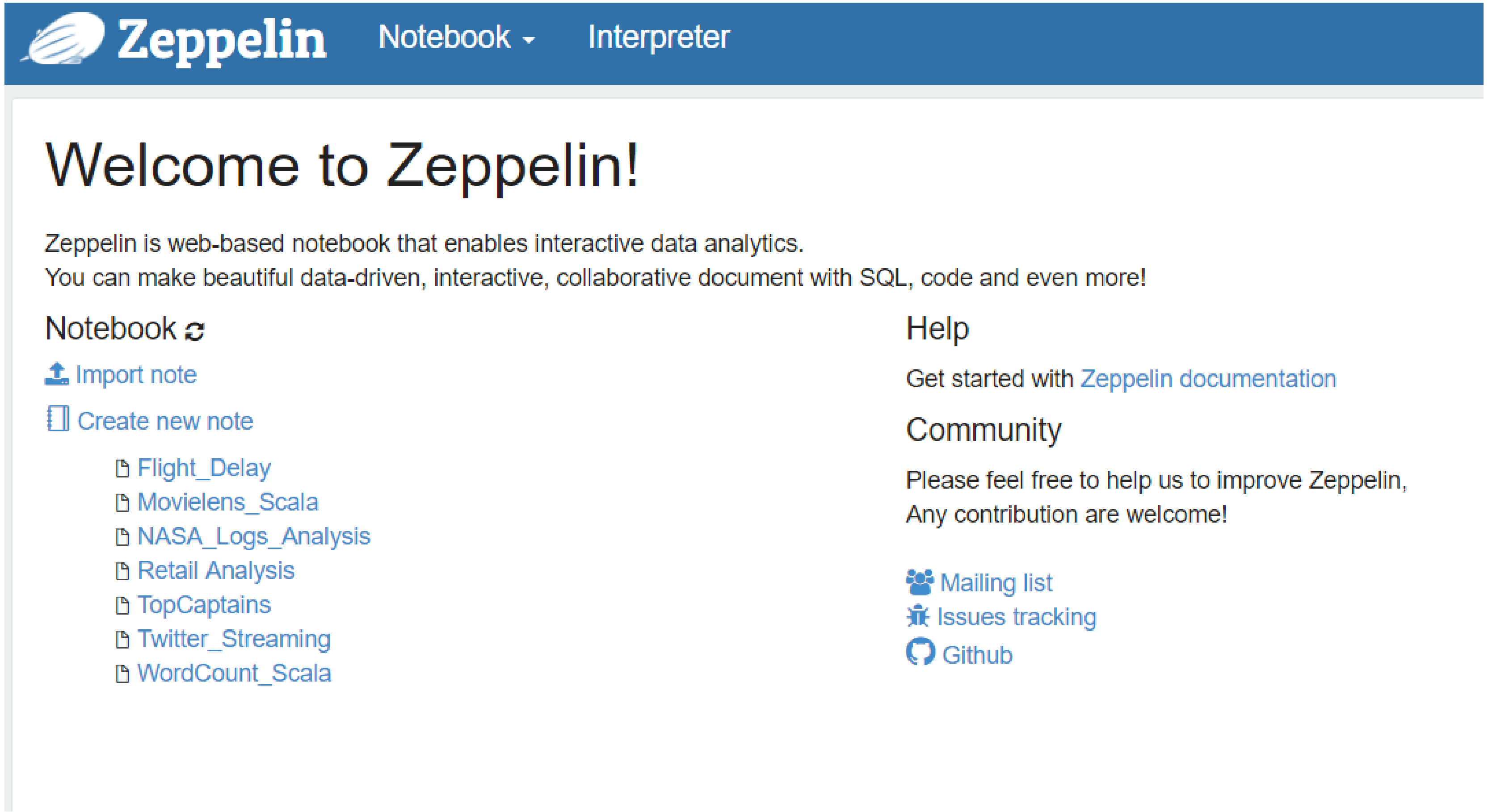
- stateRDD.cache()

Remove the RDDs that are not required any more

- stateRDD.depersist()

Working with Zeppelin

Working with Zeppelin



Welcome to Zeppelin!

Zeppelin is web-based notebook that enables interactive data analytics.
You can make beautiful data-driven, interactive, collaborative document with SQL, code and even more!

Notebook

Import note

Create new note

- Flight_Delay
- Movielens_Scala
- NASA_Logs_Analysis
- Retail Analysis
- TopCaptains
- Twitter_Streaming
- WordCount_Scala

Help

Get started with Zeppelin documentation

Community

Please feel free to help us to improve Zeppelin,
Any contribution are welcome!

Mailing list
Issues tracking
Github

Spark Programming APIs

Spark Context

- Spark program starts with creating a SparkContext object
- SparkContext created from SparkConf
  - SparkConf can be configured to pass all application configuration information
- Programmatically

```
val conf = new SparkConf().setAppName(appName).setMaster(master)
new SparkContext(conf)
```

- Using Shell
  - ./spark-shell --master local[4]

RDDs

- Creating RDDs by parallelizing collections from Driver program

  val data = Array(1, 2, 3, 4, 5)

  val distData = sc.parallelize(data)

  Note: creates 10 partitions

- Creating RDDs by reading external datasets
  - *Reading from local file system. The path should be available to all worker nodes. Typically a NFS is a good solution.*
    - *var distFile = sc.textFile(file:///data.txt, numPartitions)*
  - *Reading distributed file system like HDFS*
    - *var distFile = sc.textFile("hdfs://namenode/data.txt")*
  - *Can pass directory as a path. Can use wild cards and different formats.*
    - *sc.textFile("/my/directory") or sc.textFile("/my/directory/*.txt") or sc.textFile("/my/directory/*.gz")*
  - *Can read whole file as a record. Applicable for XMLs, images or documents*
    - *wholeTextFiles(path, minPartitions=None, use_unicode=true)*
    - *binaryFiles(path, minPartitions=None)*

RDD Transformations

- map()
- intersection()
- cartesion()
- flatMap()
- distinct()

- pipe()
- filter()
- groupByKey()
- coalesce()
- sample()

- reduceByKey()
- repartition()
- sortByKey()
- join()
- union()

Most transformations are element-wise

RDD Actions

- reduce()
- collect()
- saveAsTextFile()
- count()
- saveAsSequenceFile()

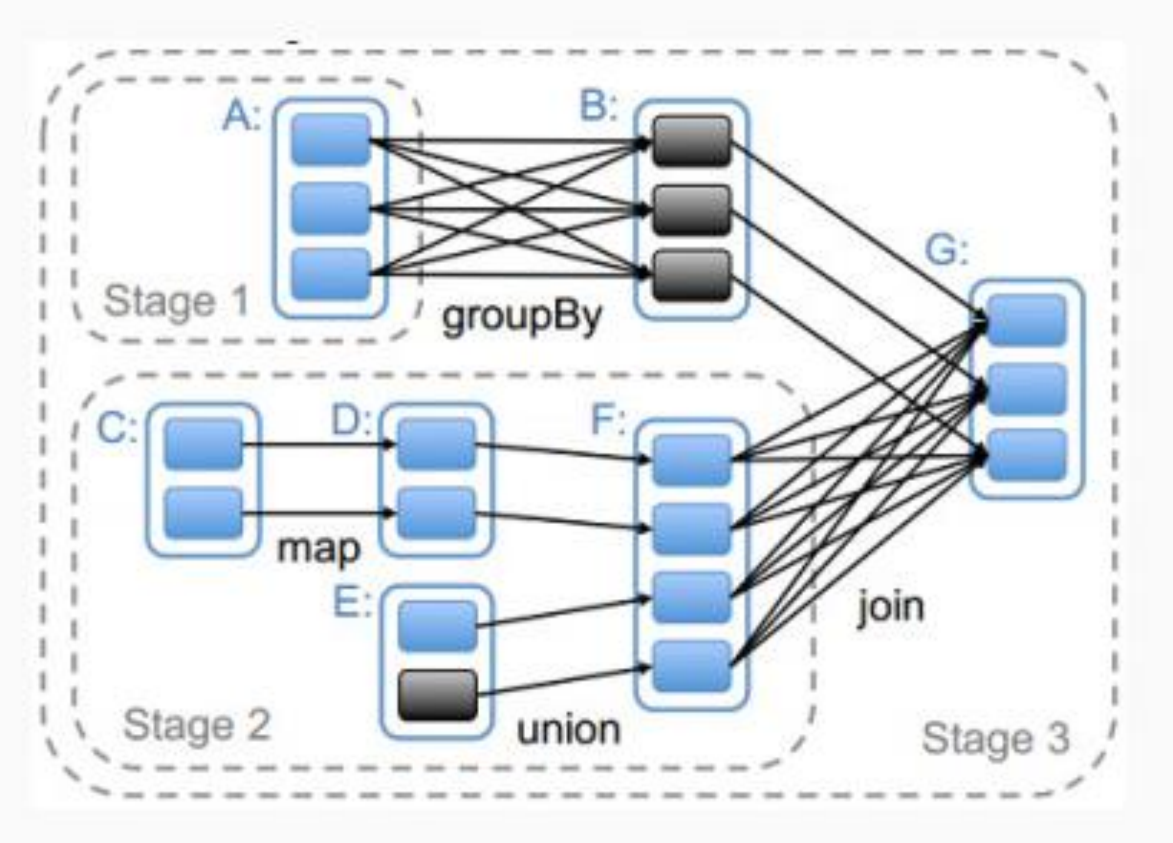
- first()
- saveAsObjectFile()
- take()
- countByKey()
- takeSample()
- foreach()

- takeSample()
- foreach()
- saveToCassandra()

Actions trigger execution and transformations are applied

RDD – Shuffling

- The **Shuffle** is an expensive operation
- It involves disk I/O, data serialization, and network I/O
- Shuffle operations can consume significant amounts of heap memory and if data does not fit in memory Spark will spill these tables to disk
- Shuffle also generates a large number of intermediate files on disk
- These files are preserved until the corresponding RDDs are no longer used and are garbage collected
- Temporary storage directory is specified by the ***spark.local.dir*** to store these intermediate files



RDD – Persistence

- RDDs can be cached by using persist() or cache() on RDDs.
- RDDs can be removed from cache by calling unpersist() on RDDs.
- Different cache levels can be defined

Storage Level	Description	When to use?
MEMORY_ONLY	Store RDD as deserialized Java objects in the JVM. If RDDs does not fit into memory, it is not cached. It is recalculated every time it is needed.	When the RDD fits into memory.
MEMORY_AND_DISK	Store RDD as deserialized Java objects in the JVM. If the RDD does not fit in memory, store the partitions on disk.	When the RDD does not fit into memory
DISK_ONLY	Store RDD as deserialized Java objects in the JVM. If the RDD does not fit in memory, store the partitions on disk.	When the RDD does not fit into memory and if the RDDs will not be reused very frequently.
MEMORY_ONLY_SER	Same as MEMORY_ONLY, but with serialized java objects. More space efficient, but more CPU-intensive to read.	Save space, but processing time is not constraining.
MEMORY_AND_DISK_SER	Same as MEMORY_AND_DISK, but with serialized java objects. More space efficient. But more CPU-intensive to read.	Save space, but processing time is not constraining.
MEMORY_ONLY_2, MEMORY_AND_DISK_2	Replicate each partition on two cluster nodes	Re-compute of lost RDD partitions can be avoided in case of system or storage failure. Compete fault tolerant.

Share Variables - Broadcast

- Share additional information with workers
- Read only data like lookup files, properties files, set of parameters etc.
- Read only variables on worker nodes
- Serialized from driver and sent to tasks in workers, which deserializes the reads it.
- Broadcast variables are created from a variable v by calling SparkContext.broadcast(v).
- Its value can be accessed by calling the value

```
val tvalue = 100
broadcastVar = sc.broadcast( tvalue )
broadcastVar.value
```

Share Variables - Accumulators

- A shared variable that can be modified by workers
- Is used to implement counters or sums
- An accumulator is with an initial value
  - *val accum = sc.accumulator(0, "My Accumulator")*
- Tasks running on the cluster can then add to it using the add method or the += operator
  - *data.map { x => accum += x; f(x) }*
- Driver program can read the accumulator’s value
  - *Accum.value*
- Can be used to collect metrics and counters from workers e.g. number of bad records.

RDD Operations

- *Walkthrough example*
  - *Cricket Captains ODI and Test Performance*

name	country	career	matches	won	loss	tie	toss
Ponting R T	Australia	1995-2012	230	165	51	14	124
Fleming S P	New Zealand	1994-2007	218	98	106	14	105
Ranatunga A	Sri Lanka	1982-1999	193	89	95	9	102
Dhoni M S*	India	2004-	186	103	68	15	88
Border A R	Australia	1979-1994	178	107	67	4	86
Azharuddin M	India	1985-2000	174	89	77	8	96
Smith G C	South Africa	2002-2013	149	91	51	7	74
Ganguly S C	India	1992-2007	147	76	66	5	74
Cronje W J	South Africa	1992-2000	140	99	37	4	74
Imran Khan	Pakistan	1974-1992	139	75	59	5	70
Jayawardene D P M	Sri Lanka	1998-2015	130	72	49	9	60
Lara B C	West Indies	1990-2007	125	59	59	7	57
Jayasuriya S T	Sri Lanka	1989-2011	117	65	47	5	61

Spark Deployment Modes

Running Spark

- Running Spark
  - Interactive Mode
  - Batch Mode
- Deployment Modes
  - Local
  - Standalone
  - YARN Cluster
  - Mesos Cluster

Running Spark – Interactive Mode

- Provides an interactive shell for ad-hoc analysis
- Spark RDDs are kept in memory as long as the sparkContext is running
- Can be stopped, when sparkContext is stopped
- How to run?
  - spark-shell –master MASTER_URL // for scala interface
  - pyspark –master MASTER_URL   // for python interface
  - Java does not have an interactive shell

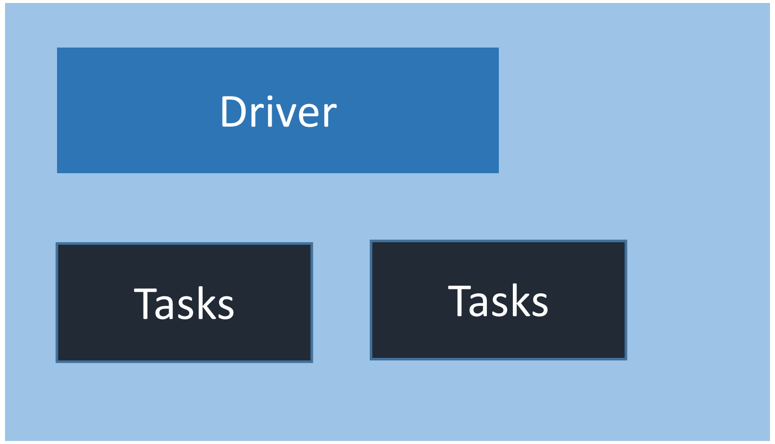
Running Spark in Batch Mode

- spark-submit
  - **--master** MASTER_URL spark://host:port, mesos://host:port, yarn, or local
  - **--deploy-mode** DEPLOY_MODE (client or cluster)
  - **--name** NAME (name of the application)
  - **--jars** JARS (list of jars to be added to classpath of the driver and executors)
  - **--conf** PROP=VALUE (spark configurations)
  - **--driver-memory** MEM (Memory for the driver program. Format 300M or 1G)
  - **--executor-memory** MEM (Memory for executor. Format 500M or 2G)
  - **--driver-cores** NUM (Cores for drivers – applicable for YARN and standalone)
  - **--executor-cores** NUM (Cores for executors – applicable for YARN and standalone)
  - **--num-executors** NUM Number of executors to launch (Default: 2).

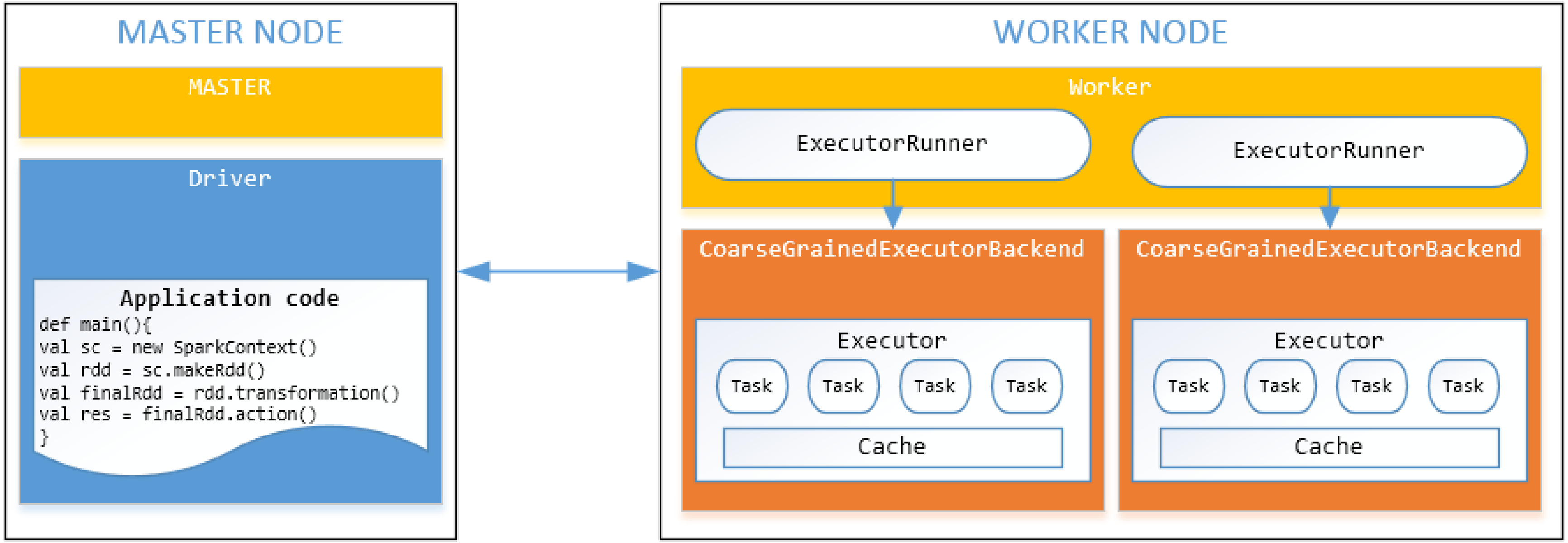
Spark Submit – Local Mode

- Non Distributed – Single JVM deployment Mode
- Single JVM runs all components – Driver, executor
- Number of tasks is n as specified in local[n]
- Number of tasks is equal to number of cores if specified as local[*]

```
spark-submit --master local[n]          // local mode with two workers
        --name topcaptains              // name of the application
        topCaptains.py                  // python script
```



Spark Submit – Standalone Mode



```
def main(){
val sc = new SparkContext()
val rdd = sc.makeRdd()
val finalRdd = rdd.transformation()
val res = finalRdd.action()
}
```

Source: http://www.trongkhoanguyen.com/2015/01/understand-spark-deployment-modes.html

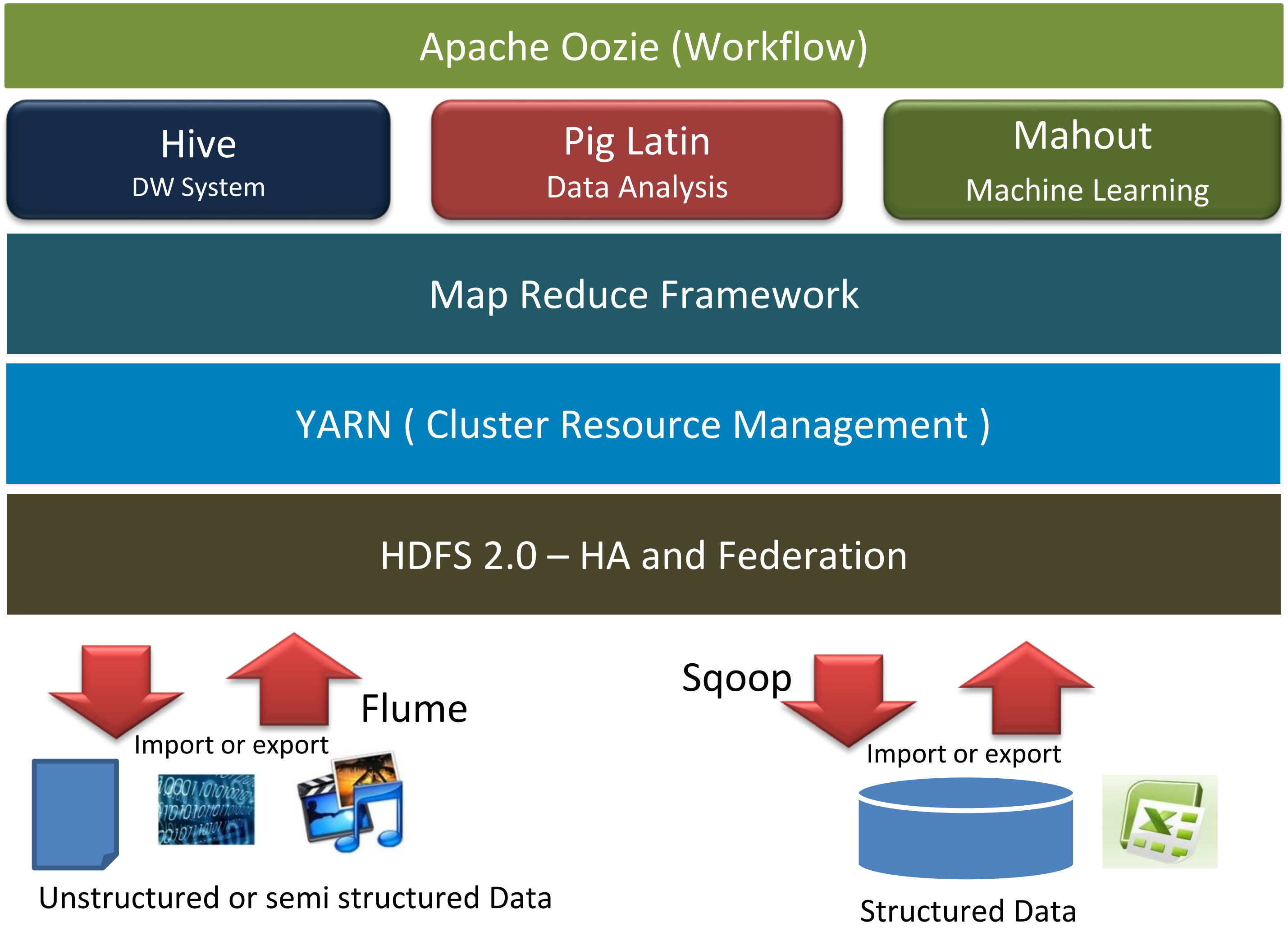
```
spark-submit
--master local[2]            // local mode with two workers
--name topcaptains           // name of the application
--driver-memory 500M         // memory allocated to driver
--executor-memory 1G         // memory allocated to executor
--executor-cores  1          // num cores for the executor
--num-executors 2            // number of executor
topCaptains.py               // python script
```

Spark Standalone Mode

- Spark can launch a cluster
  - Configuration and manage scripts are available in $SPARK_HOME
  - Configure the slaves using conf/slaves file
  - Start master by running sbin/start-master.sh
  - Start slaves by running sbin/start-slaves.sh
- Other properties to be configured in conf/spark-env.sh

SPARK_MASTER_IP, SPARK_MASTER_PORT	IP address and port number(default 7077) of master
SPARK_MASTER_WEBUI	Port for the master web UI (default: 8080)
SPARK_WORKER_CORES, SPARK_WORKER_MEMORY	Number of cores (all cores on machine) and memory (1GB) for workers.
SPARK_WORKER_PORT	Worker port (random)
SPARK_WORKER_INSTANCES	Number of worker instances on each slave (default is 1)
SPARK_WORKER_WEBUI_PORT	Worker web UI port (8081)
SPARK_LOCAL_DIRS	Scratch directory (map output files and RDDs that get stored on disk). should be on a fast, local disk in your system. Can supply multiple directories with comma separated list

Hadoop 2.0 Ecosystem



Apache Oozie (Workflow)

Hive
DW System

Pig Latin
Data Analysis

Mahout
Machine Learning

Map Reduce Framework

YARN (Cluster Resource Management)

HDFS 2.0 – HA and Federation

Flume

Import or export

Unstructured or semi structured Data

Sqoop

Import or export

Structured Data

Configuring Hadoop For Spark

- core-site.xml

fs.defaultFS	hdfs://localhost:8020/	Where namenode is running

- hdfs-site.xml

dfs.replication	3	No of Block Replications
dfs.block.size	134217728	Block size in bytes (128 MB)

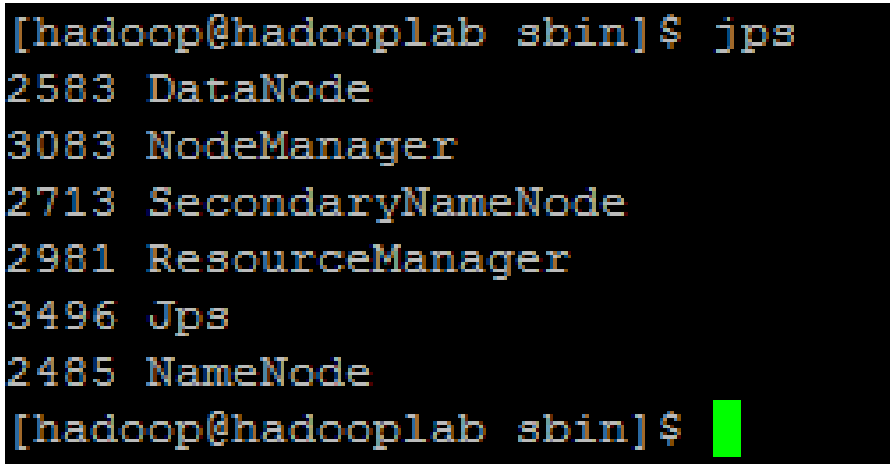
- yarn-site.xml

yarn.resourcemanager.address	hostname:8032	Where the resource Manager service is running
yarn.resourcemanager.scheduler.class	CapacityScheduler	Which scheduler to be used. Default is capacity. Other ones available are FIFO and Fair.
yarn.nodemanager.resource.memory-mb	2250	Max resources Node Manager can allocate to containers.

Services	Port
Namenode	8020
Namenode Web UI	50070
Datanode	50010
Datanode Web UI	50075
Resource Manager	8032
Resource Manager Web UI	8088
NodeManager	45454
NodeManager Web UI	50060

Starting Hadoop Services

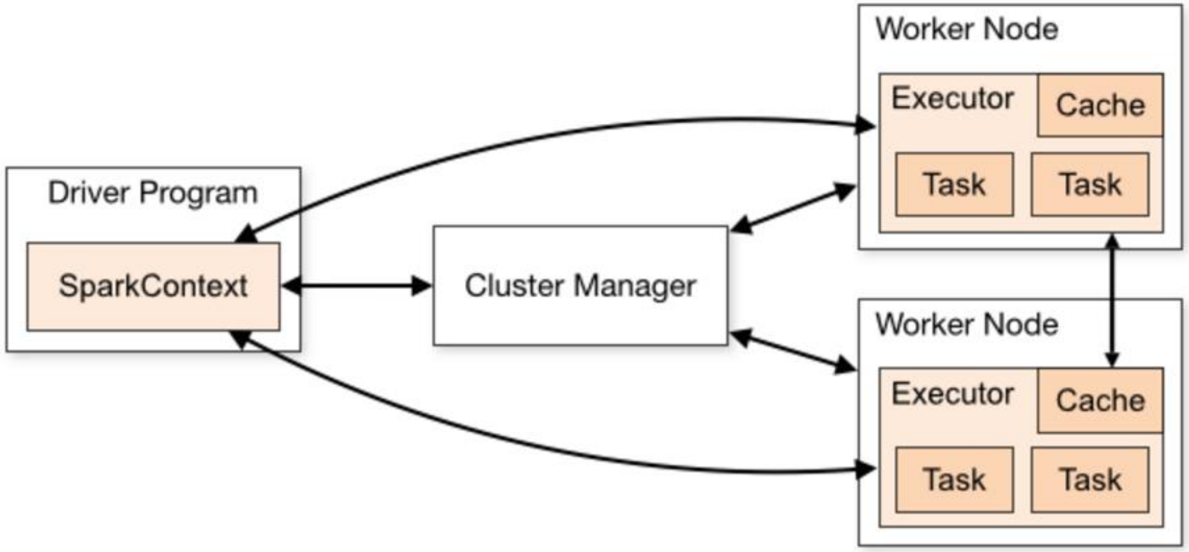
- Format NameNode
  - ***hdfs namenode –format***
  - *Creates all required HDFS required directory structure for namenode and datanodes. Creates the fsimage and edit logs.*
  - ***This should be done first time and only once.***
- Start HDFS services
  - **./start-dfs.sh**
- Start YARN services
  - **./start-yarn.sh**
- Start History Server
  - **./mr-jobhistory-daemon.sh start historyserver**
- Verify services are running
  - **jps**



```
[hadoop@hadooplab sbin]$ jps
2583 DataNode
3083 NodeManager
2713 SecondaryNameNode
2981 ResourceManager
3496 Jps
2485 NameNode
[hadoop@hadooplab sbin]$ █
```

Spark Submit – YARN/Mesos Mode

```
spark-submit
--master YARN                      // YARN Mode
--deploy-mode client/cluster
--name topcaptains                 // name of the application
--driver-memory 500M               // memory allocated to driver
--executor-memory 1G               // memory allocated to executor
--executor-cores  1                // num cores for the executor
--num-executors 2                  // number of executor
topCaptains.py                     // python script
```



Deploy Mode

- Client – Driver runs as client outside the YARN cluster
- Cluster – Driver runs in YARN nodes

Benefits of YARN Node

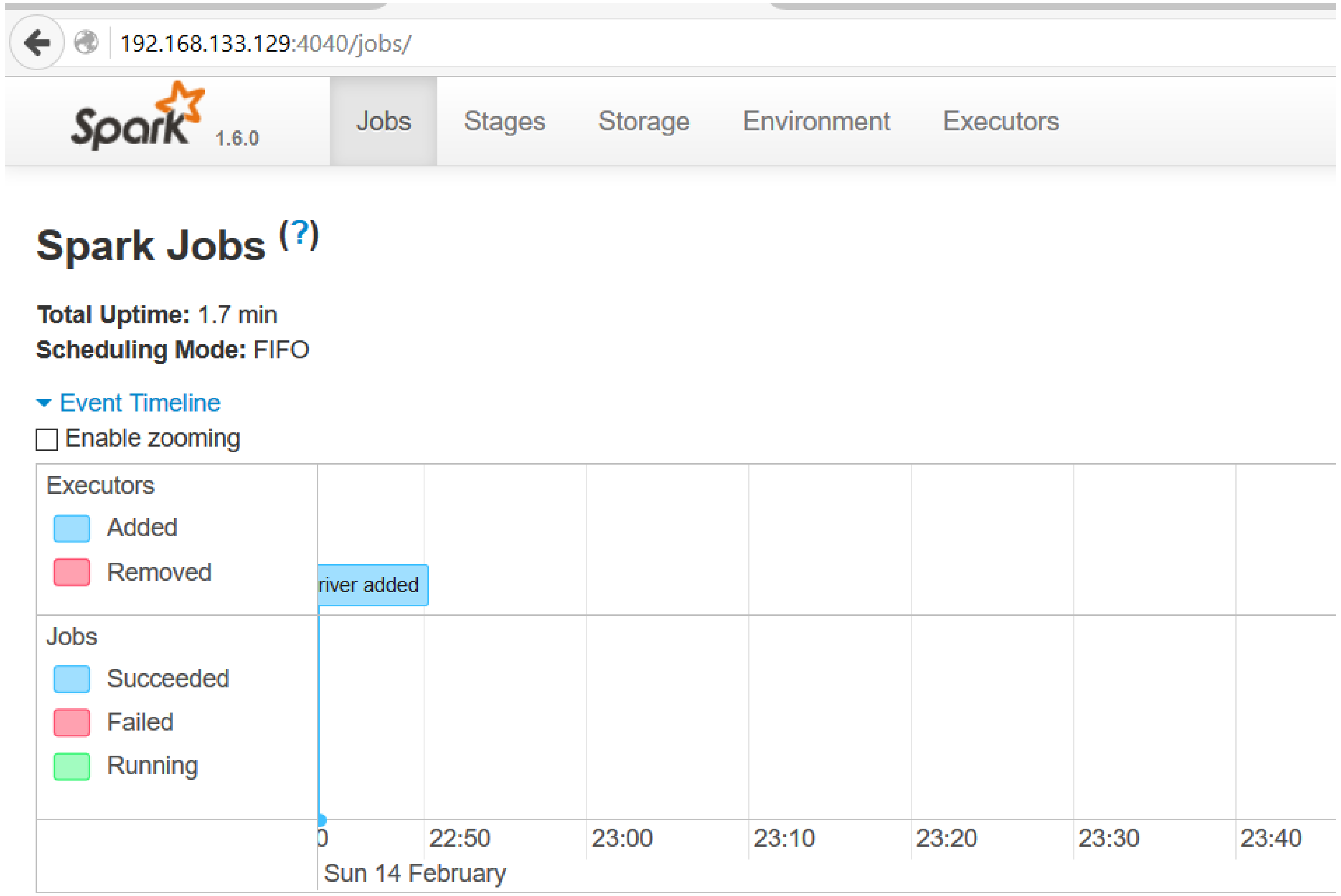
- Can run Hadoop Map Reduce and Spark jobs simultaneously on the cluster. Most widely used deployment mode.
- Spark and Hadoop Map Reduce can co-exist

Spark Monitoring & Debugging

Spark UI

- SparkContext launches a web UI and provide information about
  - A list of scheduler stages and tasks
  - A summary of RDD sizes and memory usage
  - Environmental information.
  - Information about the running executors

Note: If multiple SparkContexts are running on the same host, they will bind to successive ports beginning with 4040 (4041, 4042, etc)



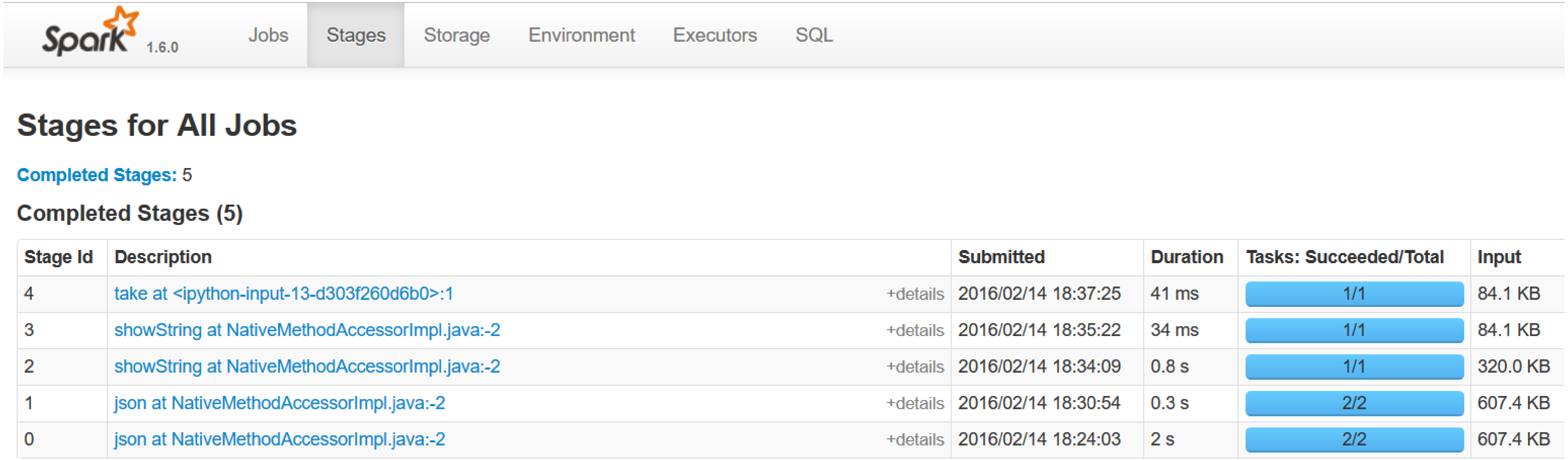
Spark Jobs (?)

Total Uptime: 1.7 min
Scheduling Mode: FIFO

SparkContext UI

- Every SparkContext launches an UI running on 4040
  - multiple SparkContexts running on same host bind to successive ports 4041, 4042 etc.
  - Available only during the life time of the spark context or spark application
- To enable history of completed applications
  - Set *spark.eventLog.enabled* to *true*
  - Set *spark.history.fs.logDirectory* to *file:/home/hadoop/lab/logs*
  - Set *spark.history.provider* to *org.apache.spark.deploy.history.FsHistoryProvider*
  - *spark.history.fs.update.interval - 10 secs (Period for displayed info to be updated)*
  - *spark.history.retainedApplications – 50 (Num of application logs to be retained)*
  - *spark.history.ui.port – 18080 (UI port for the history server)*
  - *spark.history.fs.cleaner.interval – 1d (job history cleaner checks for files to delete)*
  - *spark.history.fs.cleaner.maxAge – 7d (history files older than this will be deleted)*
- Start History Server
  - ./sbin/start-history-server.sh

Spark - Stages



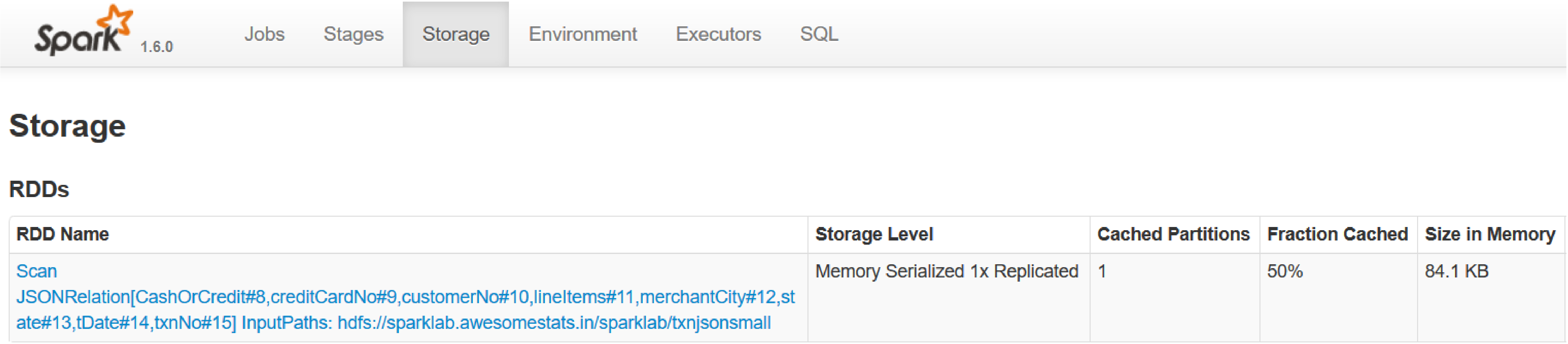
Stages for All Jobs

Completed Stages: 5

Completed Stages (5)

Stage Id	Description		Submitted	Duration	Tasks: Succeeded/Total	Input
4	take at <ipython-input-13-d303f260d6b0>:1	+details	2016/02/14 18:37:25	41 ms	1/1	84.1 KB
3	showString at NativeMethodAccessorImpl.java:-2	+details	2016/02/14 18:35:22	34 ms	1/1	84.1 KB
2	showString at NativeMethodAccessorImpl.java:-2	+details	2016/02/14 18:34:09	0.8 s	1/1	320.0 KB
1	json at NativeMethodAccessorImpl.java:-2	+details	2016/02/14 18:30:54	0.3 s	2/2	607.4 KB
0	json at NativeMethodAccessorImpl.java:-2	+details	2016/02/14 18:24:03	2 s	2/2	607.4 KB

Spark - Storage

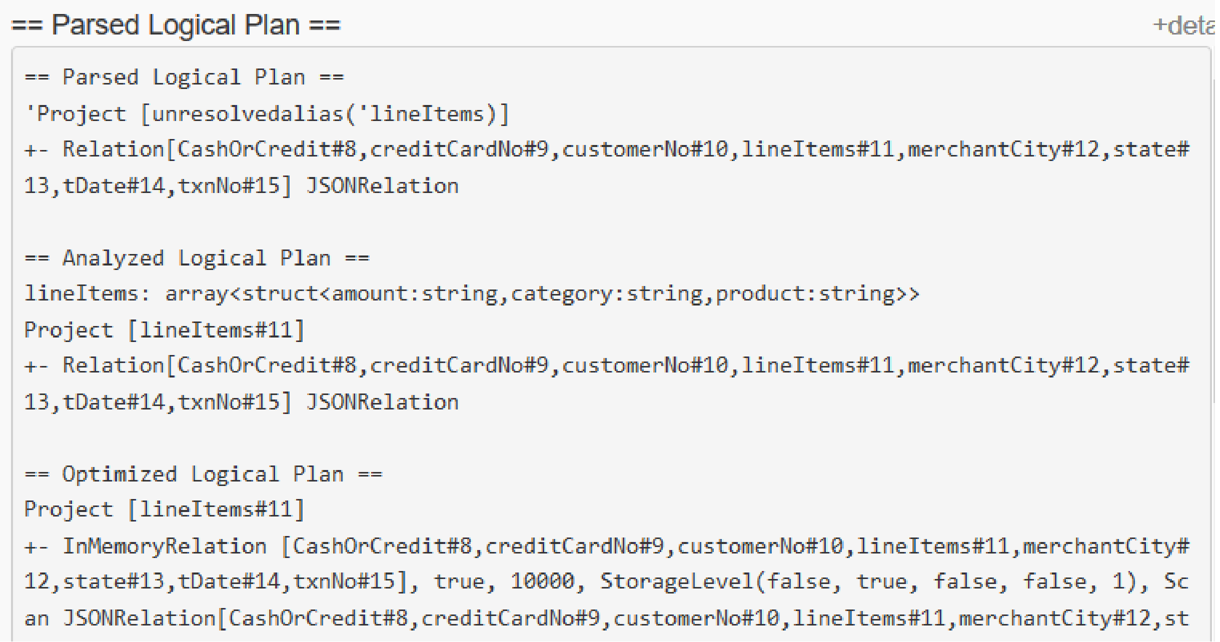


Storage

RDDs

RDD Name	Storage Level	Cached Partitions	Fraction Cached	Size in Memory
Scan JSONRelation[CashOrCredit#8,creditCardNo#9,customerNo#10,lineItems#11,merchantCity#12,state#13,tDate#14,txnNo#15] InputPaths: hdfs://sparklab.awesomestats.in/sparklab/txnjsonsmall	Memory Serialized 1x Replicated	1	50%	84.1 KB

Spark - SQL

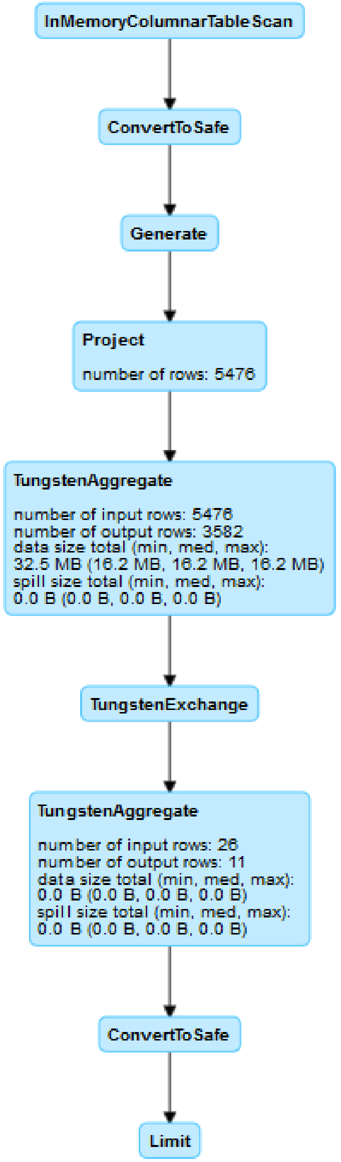


== Parsed Logical Plan == +deta

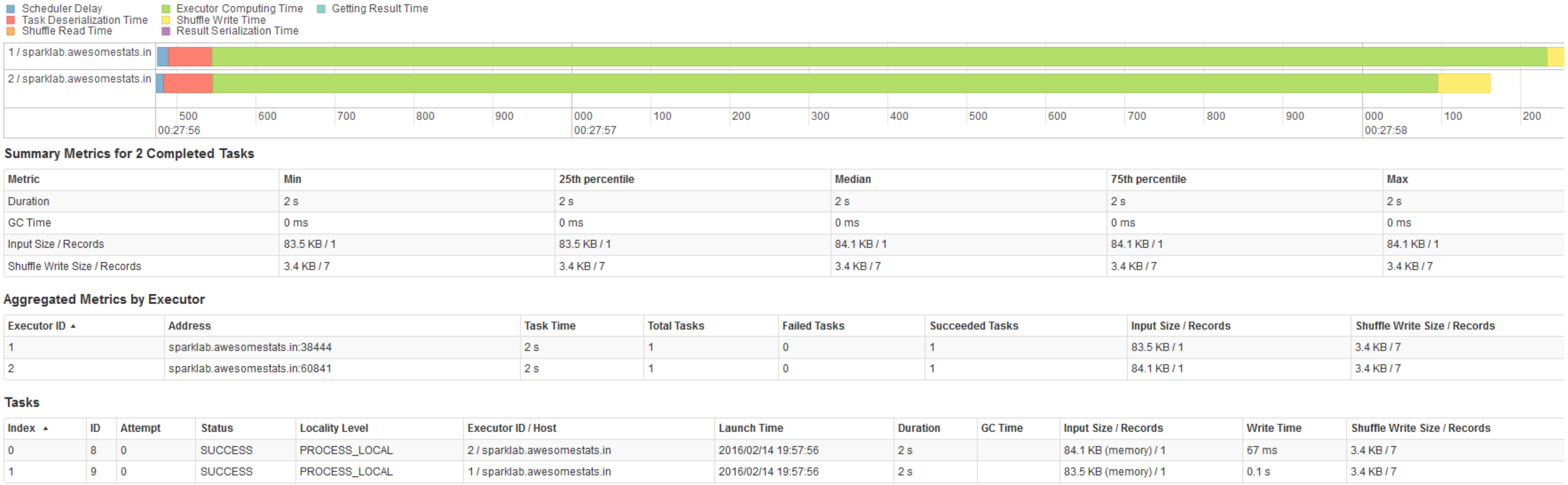
```
== Parsed Logical Plan ==
'Project [unresolvedalias('lineItems)]
+- Relation[CashOrCredit#8,creditCardNo#9,customerNo#10,lineItems#11,merchantCity#12,state#
13,tDate#14,txnNo#15] JSONRelation

== Analyzed Logical Plan ==
lineItems: array<struct<amount:string,category:string,product:string>>
Project [lineItems#11]
+- Relation[CashOrCredit#8,creditCardNo#9,customerNo#10,lineItems#11,merchantCity#12,state#
13,tDate#14,txnNo#15] JSONRelation

== Optimized Logical Plan ==
Project [lineItems#11]
+- InMemoryRelation [CashOrCredit#8,creditCardNo#9,customerNo#10,lineItems#11,merchantCity#
12,state#13,tDate#14,txnNo#15], true, 10000, StorageLevel(false, true, false, false, 1), Sc
an JSONRelation[CashOrCredit#8,creditCardNo#9,customerNo#10,lineItems#11,merchantCity#12,st
```


InMemoryColumnarTableScan
ConvertToSafe
Generate
Project
number of rows: 5476
TungstenAggregate
number of input rows: 5476
number of output rows: 3582
data size total (min, med, max):
32.5 MB (16.2 MB, 16.2 MB, 16.2 MB)
spill size total (min, med, max):
0.0 B (0.0 B, 0.0 B, 0.0 B)
TungstenExchange
TungstenAggregate
number of input rows: 26
number of output rows: 11
data size total (min, med, max):
0.0 B (0.0 B, 0.0 B, 0.0 B)
spill size total (min, med, max):
0.0 B (0.0 B, 0.0 B, 0.0 B)
ConvertToSafe
Limit

Spark - Timeline



Summary Metrics for 2 Completed Tasks

Metric	Min	25th percentile	Median	75th percentile	Max
Duration	2 s	2 s	2 s	2 s	2 s
GC Time	0 ms	0 ms	0 ms	0 ms	0 ms
Input Size / Records	83.5 KB / 1	83.5 KB / 1	84.1 KB / 1	84.1 KB / 1	84.1 KB / 1
Shuffle Write Size / Records	3.4 KB / 7	3.4 KB / 7	3.4 KB / 7	3.4 KB / 7	3.4 KB / 7

Aggregated Metrics by Executor

Executor ID ▲	Address	Task Time	Total Tasks	Failed Tasks	Succeeded Tasks	Input Size / Records	Shuffle Write Size / Records
1	sparklab.awesomestats.in:38444	2 s	1	0	1	83.5 KB / 1	3.4 KB / 7
2	sparklab.awesomestats.in:60841	2 s	1	0	1	84.1 KB / 1	3.4 KB / 7

Tasks

Index ▲	ID	Attempt	Status	Locality Level	Executor ID / Host	Launch Time	Duration	GC Time	Input Size / Records	Write Time	Shuffle Write Size / Records
0	8	0	SUCCESS	PROCESS_LOCAL	2 / sparklab.awesomestats.in	2016/02/14 19:57:56	2 s		84.1 KB (memory) / 1	67 ms	3.4 KB / 7
1	9	0	SUCCESS	PROCESS_LOCAL	1 / sparklab.awesomestats.in	2016/02/14 19:57:56	2 s		83.5 KB (memory) / 1	0.1 s	3.4 KB / 7

Working with DataFrames

DataFrames

- Distributed collection of data, organized like table with named columns
- For Structured data processing
- Very familiar data structures to data scientists
  - Widely used in R and Python (Pandas library)
- DataFrames can be created in Spark from files, Hive Tables, databases etc.
- Spark DataFrame APIs are available in Scala, Java, Python and R
- DataFrame can be analysed using SQL types of queries

Creating DataFrames

- Create an sql context

  from pyspark.sql import SQLContext

  sqlContext = SQLContext(sc)

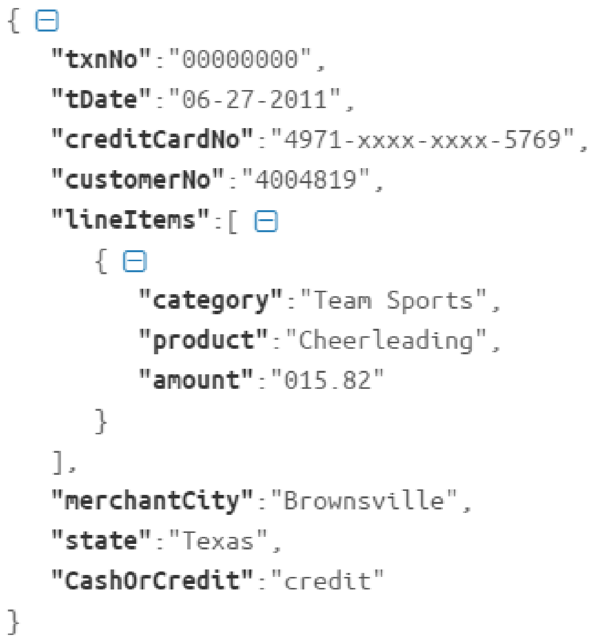
- Directly read files or databases as DataFrames
  - Local Files, HDFS
  - CSV, JSON formats
  - Columnar formats like Parquet, ORC
  - Databases – Mysql, Hive, Oracle etc.
- Convert an RDD into DataFrame
  - *df = sqlContext.crateDataFrame(rddv)*
- DataFrames can be converted into RDDs
  - *df.rdd*

Structured Data Formats (CSV)

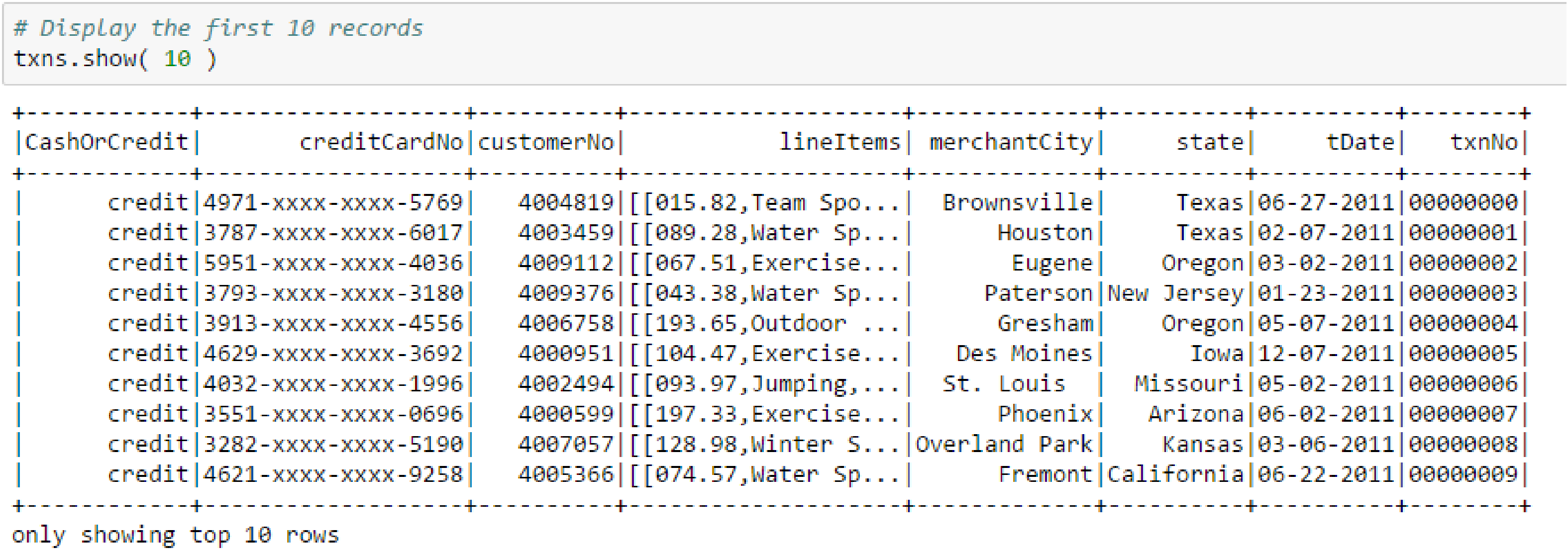
- Include the spark csv package
  - --packages com.databricks:spark-csv_2.10:1.3.0
- Load the file using the following API
  - **sqlContext.read.format("com.databricks.spark.csv").options(delimiter='\t').load(‘filename')**
  - path: location of files.
  - header: when set to true the first line of files will be used to name columns and will not be included in data. Default value is false.
  - delimiter: by default columns are delimited using comma but delimiter can be set to any character
  - quote: by default the quote character is ", but can be set to any character. Delimiters inside quotes are ignored
  - inferSchema: automatically infers column types. It requires one extra pass over the data and is false by default
  - comment: skip lines beginning with this character. Default is "#". Disable comments by setting this to null.
  - codec: compression codec to use when saving to file. bzip2, gzip, lz4, and snappy). Defaults to no compression
  - Details at: https://github.com/databricks/spark-csv
  - **sqlContext.read.format("com.databricks.spark.csv").options(delimiter='\t', header = True, inferSchema = True).load(‘filename’)**
- Write a DataFrame to csv
  - **df.write.format('com.databricks.spark.csv').save('mycsv.csv')**

JSON Files

- Read JSON File as DataFrame
  - txns = sqlContext.read.json(‘filename’)
- Write DataFrames as json files
  - df.write.json(‘filename’);



```
{
   "txnNo":"00000000",
   "tDate":"06-27-2011",
   "creditCardNo":"4971-xxxx-xxxx-5769",
   "customerNo":"4004819",
   "lineItems":[
      {
         "category":"Team Sports",
         "product":"Cheerleading",
         "amount":"015.82"
      }
   ],
   "merchantCity":"Brownsville",
   "state":"Texas",
   "CashOrCredit":"credit"
}
```

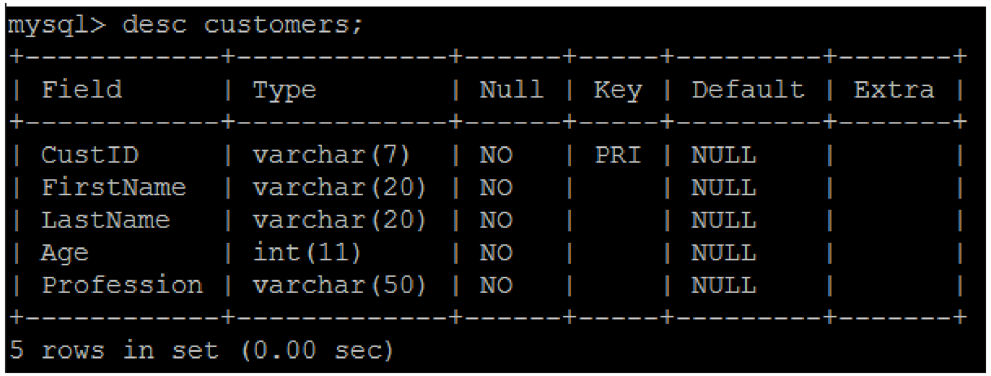


```
# Display the first 10 records
txns.show( 10 )
```

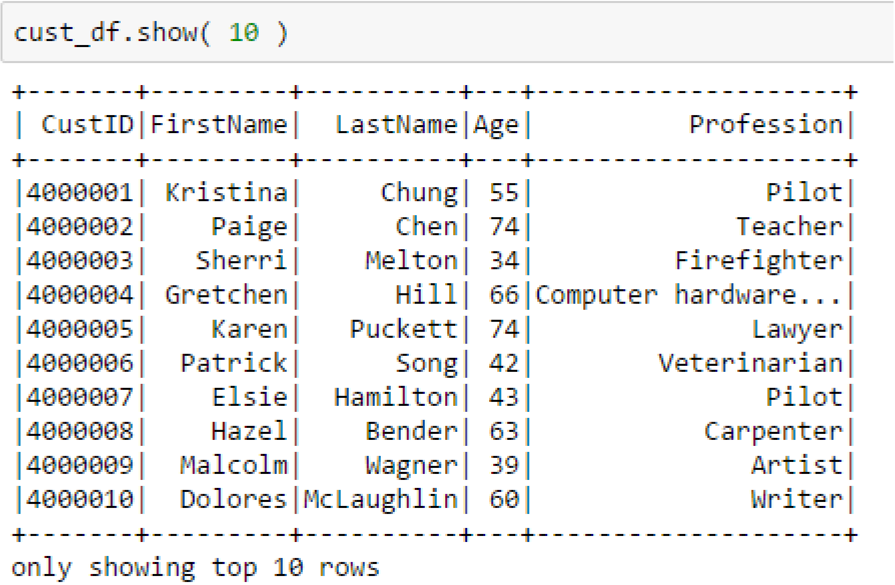
```
+------------+-------------------+----------+--------------------+-------------+----------+----------+--------+
|CashOrCredit|       creditCardNo|customerNo|           lineItems| merchantCity|     state|     tDate|   txnNo|
+------------+-------------------+----------+--------------------+-------------+----------+----------+--------+
|      credit|4971-xxxx-xxxx-5769|   4004819|[[015.82,Team Spo...|  Brownsville|     Texas|06-27-2011|00000000|
|      credit|3787-xxxx-xxxx-6017|   4003459|[[089.28,Water Sp...|      Houston|     Texas|02-07-2011|00000001|
|      credit|5951-xxxx-xxxx-4036|   4009112|[[067.51,Exercise...|       Eugene|    Oregon|03-02-2011|00000002|
|      credit|3793-xxxx-xxxx-3180|   4009376|[[043.38,Water Sp...|     Paterson|New Jersey|01-23-2011|00000003|
|      credit|3913-xxxx-xxxx-4556|   4006758|[[193.65,Outdoor ...|      Gresham|    Oregon|05-07-2011|00000004|
|      credit|4629-xxxx-xxxx-3692|   4000951|[[104.47,Exercise...|   Des Moines|      Iowa|12-07-2011|00000005|
|      credit|4032-xxxx-xxxx-1996|   4002494|[[093.97,Jumping,...|  St. Louis  |  Missouri|05-02-2011|00000006|
|      credit|3551-xxxx-xxxx-0696|   4000599|[[197.33,Exercise...|      Phoenix|   Arizona|06-02-2011|00000007|
|      credit|3282-xxxx-xxxx-5190|   4007057|[[128.98,Winter S...|Overland Park|    Kansas|03-06-2011|00000008|
|      credit|4621-xxxx-xxxx-9258|   4005366|[[074.57,Water Sp...|      Fremont|California|06-22-2011|00000009|
+------------+-------------------+----------+--------------------+-------------+----------+----------+--------+
only showing top 10 rows
```

Working with Databases - JDBC

- Read from mysql table
  - cust_df = sqlContext.read.format('jdbc').options(url='jdbc:mysql://localhost/retail?user=root&password=pwd',dbtable='customers'). load()
- Read from mysql table.
  - df.write.jdbc(url, table, mode=None, properties=None)
  - Properties parameter to set username and password



```
mysql> desc customers;
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| CustID     | varchar(7)  | NO   | PRI | NULL    |       |
| FirstName  | varchar(20) | NO   |     | NULL    |       |
| LastName   | varchar(20) | NO   |     | NULL    |       |
| Age        | int(11)     | NO   |     | NULL    |       |
| Profession | varchar(50) | NO   |     | NULL    |       |
+------------+-------------+------+-----+---------+-------+
5 rows in set (0.00 sec)
```

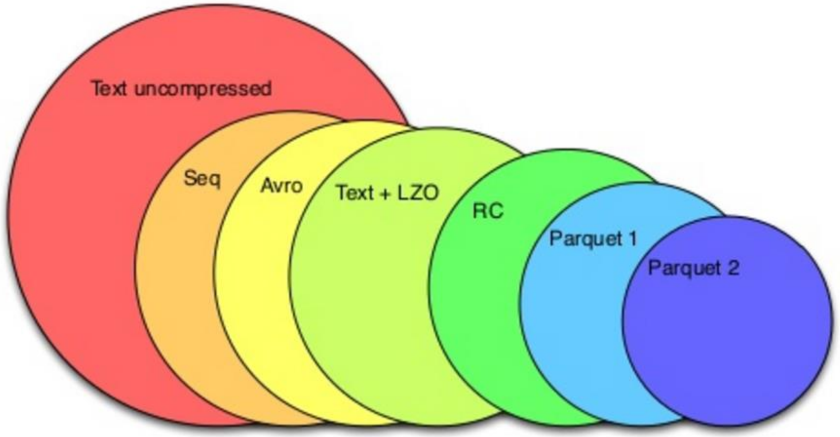


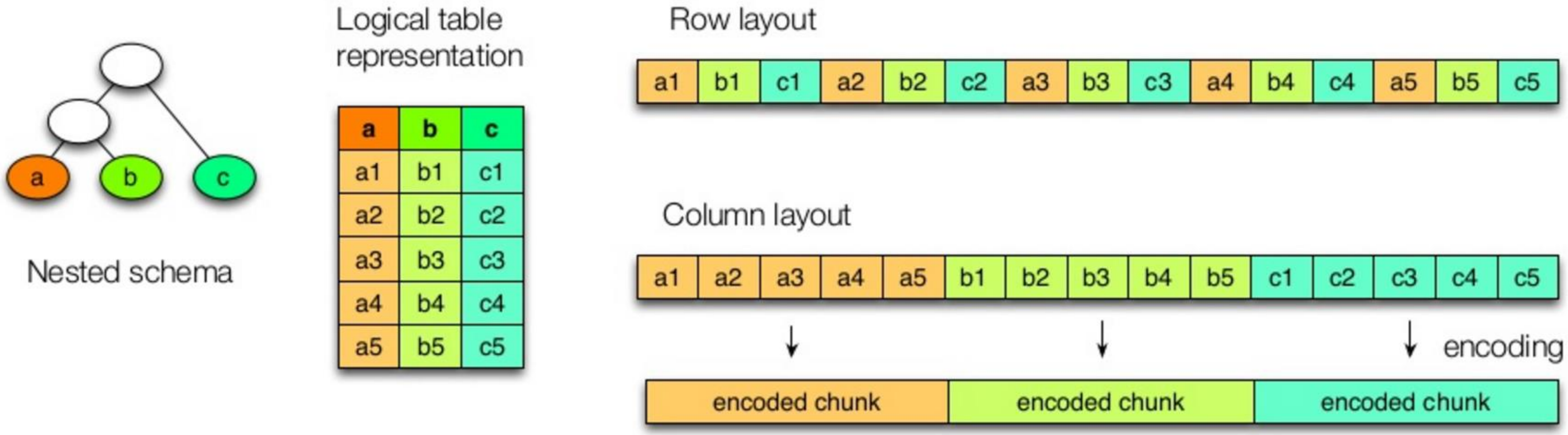
```
cust_df.show( 10 )
```

```
+-------+---------+----------+---+--------------------+
| CustID|FirstName|  LastName|Age|          Profession|
+-------+---------+----------+---+--------------------+
|4000001| Kristina|     Chung| 55|               Pilot|
|4000002|    Paige|      Chen| 74|             Teacher|
|4000003|   Sherri|    Melton| 34|         Firefighter|
|4000004| Gretchen|      Hill| 66|Computer hardware...|
|4000005|    Karen|   Puckett| 74|              Lawyer|
|4000006|  Patrick|      Song| 42|        Veterinarian|
|4000007|    Elsie|  Hamilton| 43|               Pilot|
|4000008|    Hazel|    Bender| 63|           Carpenter|
|4000009|  Malcolm|    Wagner| 39|              Artist|
|4000010|  Dolores|McLaughlin| 60|              Writer|
+-------+---------+----------+---+--------------------+
only showing top 10 rows
```

Parquet Files

- Parquet is a columnar storage format
  - It supports nested structures – Denormalized Data
  - Compressed data and improved query performance
- Read Parquet files
  - parquetFile = sqlContext.read.parquet("people.parquet")
- DataFrames can also be saved as parquet files
  - schemaPeople.write.parquet("people.parquet")

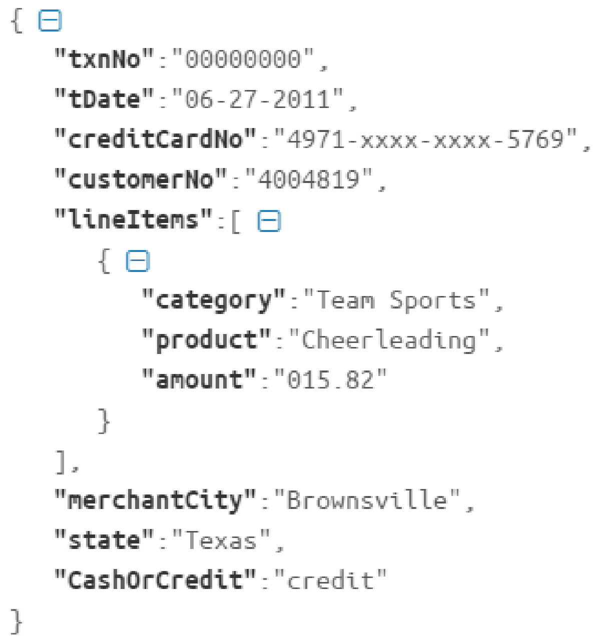



Nested schema
a
b
c
Logical table representation
Row layout
a1 b1 c1 a2 b2 c2 a3 b3 c3 a4 b4 c4 a5 b5 c5
Column layout
a1 a2 a3 a4 a5 b1 b2 b3 b4 b5 c1 c2 c3 c4 c5
encoding
encoded chunk
encoded chunk
encoded chunk

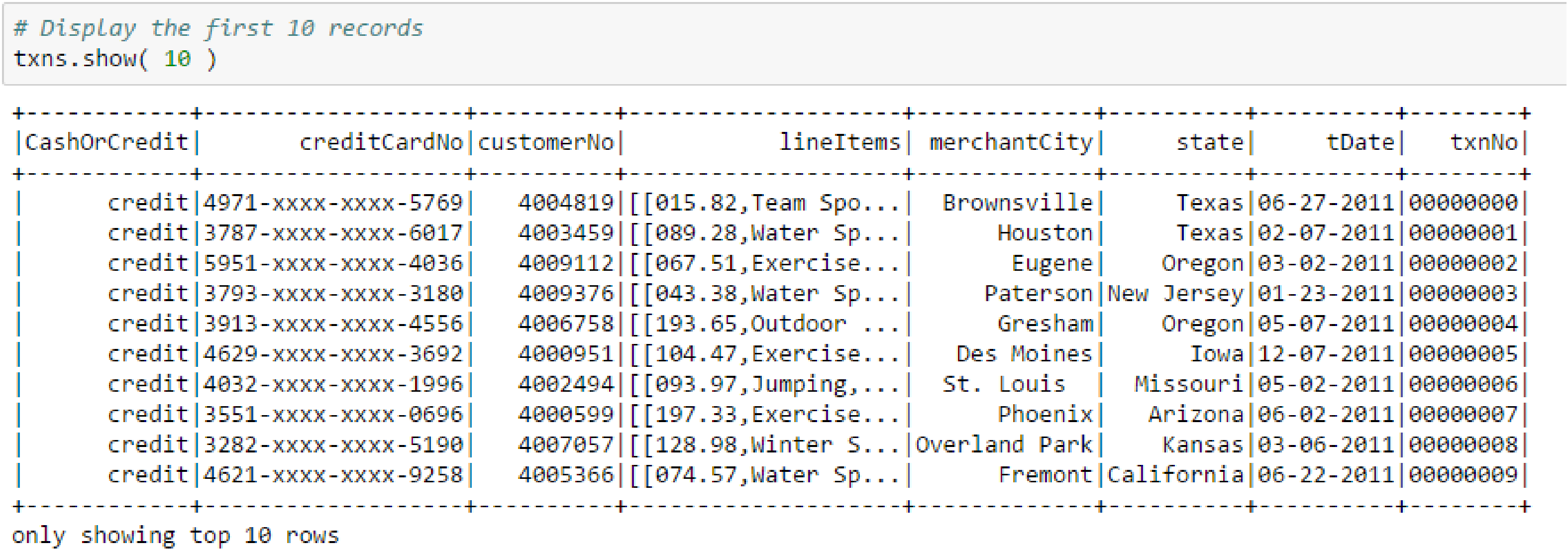
a	b	c
a1	b1	c1
a2	b2	c2
a3	b3	c3
a4	b4	c4
a5	b5	c5

Unstructured Data – Log Files

- Read JSON File as DataFrame
  - txns = sqlContext.read.json(‘filename’)
- Write DataFrames as json files
  - df.write.json(‘filename’);



```
{ ⊟
   "txnNo":"00000000",
   "tDate":"06-27-2011",
   "creditCardNo":"4971-xxxx-xxxx-5769",
   "customerNo":"4004819",
   "lineItems":[ ⊟
      { ⊟
         "category":"Team Sports",
         "product":"Cheerleading",
         "amount":"015.82"
      }
   ],
   "merchantCity":"Brownsville",
   "state":"Texas",
   "CashOrCredit":"credit"
}
```



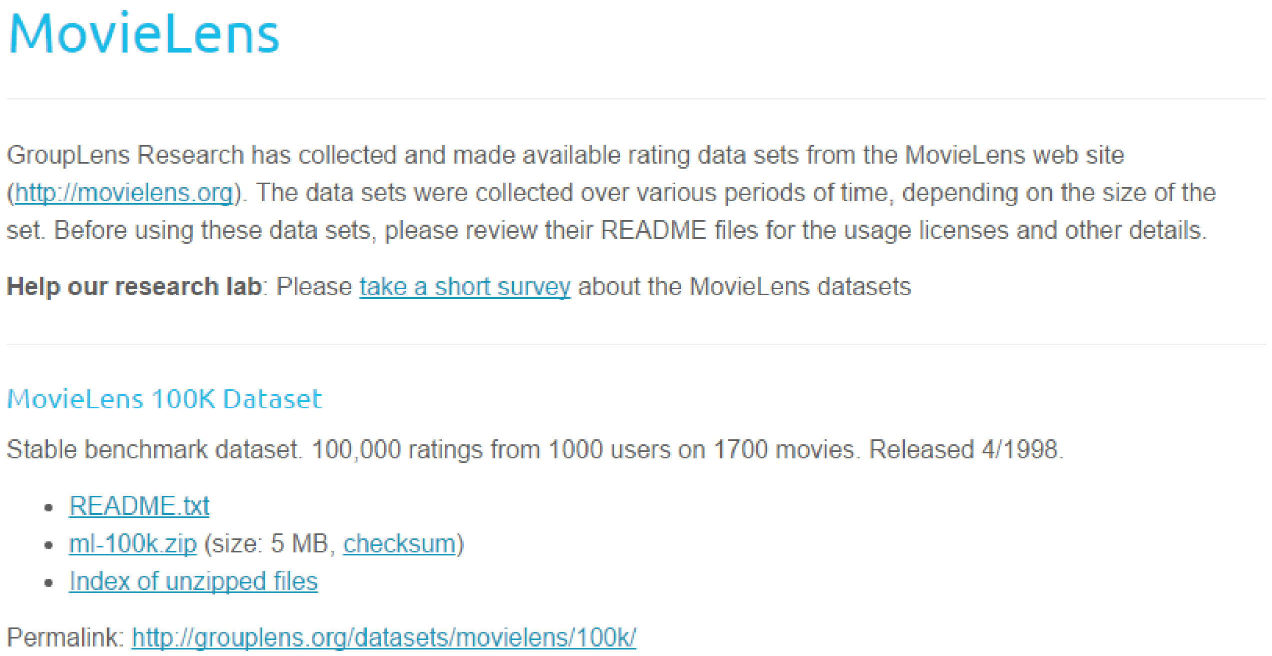
```
# Display the first 10 records
txns.show( 10 )
```

```
+------------+-------------------+----------+--------------------+-------------+----------+----------+--------+
|CashOrCredit|       creditCardNo|customerNo|           lineItems| merchantCity|     state|     tDate|   txnNo|
+------------+-------------------+----------+--------------------+-------------+----------+----------+--------+
|      credit|4971-xxxx-xxxx-5769|   4004819|[[015.82,Team Spo...|  Brownsville|     Texas|06-27-2011|00000000|
|      credit|3787-xxxx-xxxx-6017|   4003459|[[089.28,Water Sp...|      Houston|     Texas|02-07-2011|00000001|
|      credit|5951-xxxx-xxxx-4036|   4009112|[[067.51,Exercise...|       Eugene|    Oregon|03-02-2011|00000002|
|      credit|3793-xxxx-xxxx-3180|   4009376|[[043.38,Water Sp...|     Paterson|New Jersey|01-23-2011|00000003|
|      credit|3913-xxxx-xxxx-4556|   4006758|[[193.65,Outdoor ...|      Gresham|    Oregon|05-07-2011|00000004|
|      credit|4629-xxxx-xxxx-3692|   4000951|[[104.47,Exercise...|   Des Moines|      Iowa|12-07-2011|00000005|
|      credit|4032-xxxx-xxxx-1996|   4002494|[[093.97,Jumping,...|  St. Louis  |  Missouri|05-02-2011|00000006|
|      credit|3551-xxxx-xxxx-0696|   4000599|[[197.33,Exercise...|      Phoenix|   Arizona|06-02-2011|00000007|
|      credit|3282-xxxx-xxxx-5190|   4007057|[[128.98,Winter S...|Overland Park|    Kansas|03-06-2011|00000008|
|      credit|4621-xxxx-xxxx-9258|   4005366|[[074.57,Water Sp...|      Fremont|California|06-22-2011|00000009|
+------------+-------------------+----------+--------------------+-------------+----------+----------+--------+
only showing top 10 rows
```

DataFrame Operations

- Walkthrough examples
  - Movies Data Analysis
  - Retail Data Analysis
    - Transaction Records
    - Customer Records

http://grouplens.org/datasets/movielens/



MovieLens

GroupLens Research has collected and made available rating data sets from the MovieLens web site (http://movielens.org). The data sets were collected over various periods of time, depending on the size of the set. Before using these data sets, please review their README files for the usage licenses and other details.

Help our research lab: Please take a short survey about the MovieLens datasets

MovieLens 100K Dataset

Stable benchmark dataset. 100,000 ratings from 1000 users on 1700 movies. Released 4/1998.

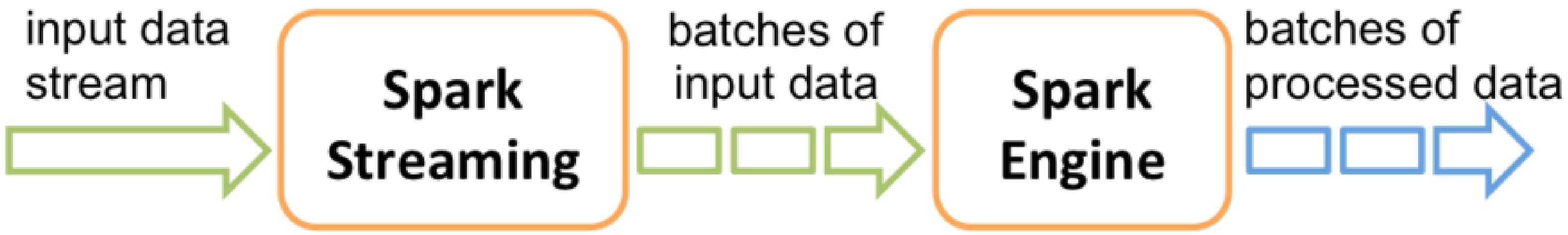
- README.txt
- ml-100k.zip (size: 5 MB, checksum)
- Index of unzipped files

Permalink: http://grouplens.org/datasets/movielens/100k/

Spark SQL CI

Streaming Analysis with Spark

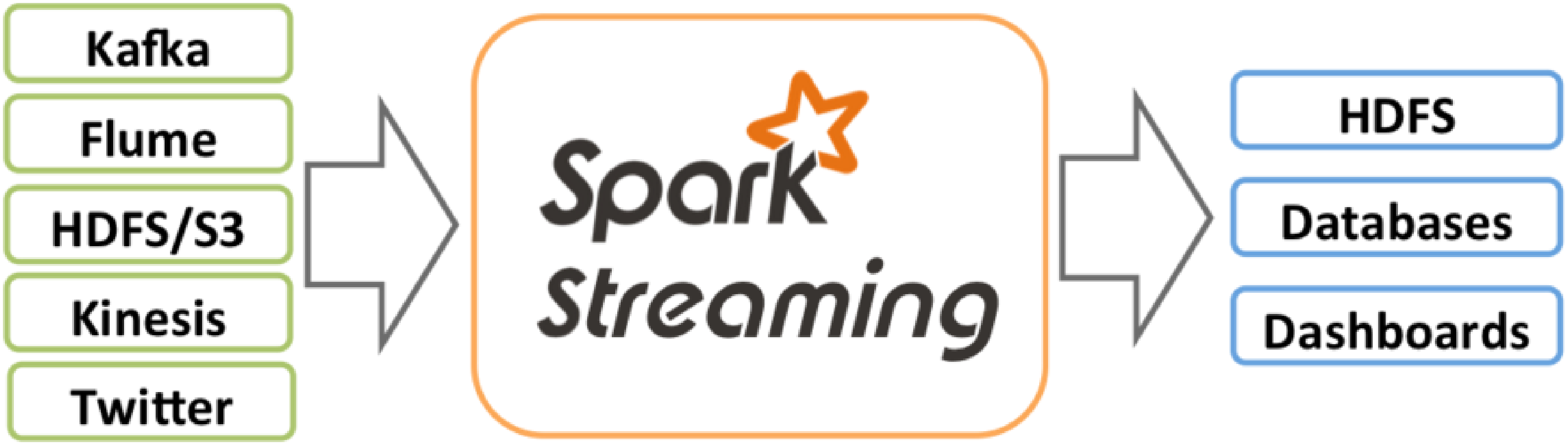
How streaming analysis works?



Source: http://spark.apache.org/docs/latest/streaming-programming-guide.html

- Can receive data from multiple sources
- Continuous stream of data is called DStream
- Divides the data streams into multiple batches of RDDs
- Transformations can be applied to RDDs
- Results can be sent to Destinations

How streaming analysis works?



Source: http://spark.apache.org/docs/latest/streaming-programming-guide.html

- Spark streaming supports multiple sources like Kafka, Flume, HDFS/S3, Kinesis, Twitter, TCP Sockets, Local files or NFS etc.
- Can also develop custom source adaptors
- Can write to multiple destinations like files, tcp sockets, databases or custom destination adaptors

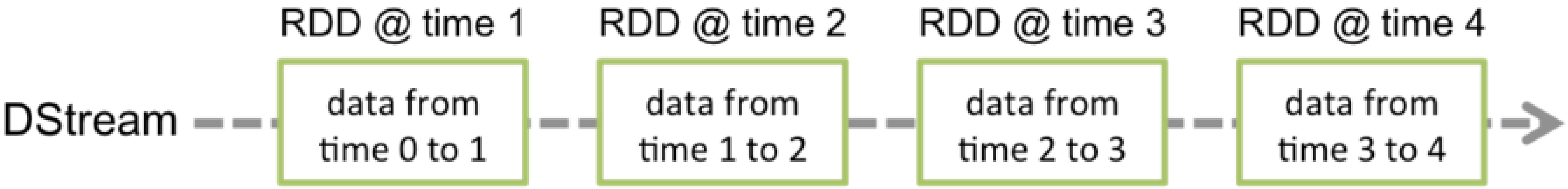
Write a streaming application

```
# import streaming specific packages
from pyspark import SparkContext
from pyspark.streaming import StreamingContext

# Create a local StreamingContext with two working thread and batch
interval of 1 second
sc = SparkContext("local[2]", “StreamingApplication")
ssc = StreamingContext(sc, 1)

#Listen to a streaming source. For example pull data from a TCP socket
lines = ssc.socketTextStream( “ServerIP",  9999 )
```

- lines will be a RDD created every one second of input streaming data. And transformations and actions can be applied to it.
- Once a context has been stopped, it cannot be restarted.
- Only one StreamingContext can be active in a JVM at the same time.



Source: http://spark.apache.org/docs/latest/streaming-programming-guide.html

Operations on Streaming RDDs

- flatMap(func)
- filter(func)
- repartition(numPartitions)
- union(otherStream)
- count()
- reduce(func)
- countByValue()
- reduceByKey(func, [numTasks])
- join(otherStream, [numTasks])
- cogroup(otherStream, [numTasks])
- transform(func)
- updateStateByKey(func)

Detailed description of operations are documented at
http://spark.apache.org/docs/latest/streaming-programming-guide.html

Streaming Failures

- Spark streaming guarantees
  - At least Once
  - Exactly Once
- Spark streaming guarantees
  - Driver Failure
  - Executor Failure
  - *streamingContext.checkpoint(hdfs directory)*
- Driver restart in YARN
  - Should run in cluster mode
  - Configure yarn.resourcemanager.am.max-attempts

Writing code to recover From Failure

- Function to create and setup a new StreamingContext

```
def functionToCreateContext():
   sc = SparkContext(...)   # new context
   ssc = new StreamingContext(...)
   lines = ssc.socketTextStream(...) # create DStreams
   ...
   ssc.checkpoint(checkpointDirectory)   # set checkpoint directory
   return ssc
```

- Get StreamingContext from checkpoint data or create a new one

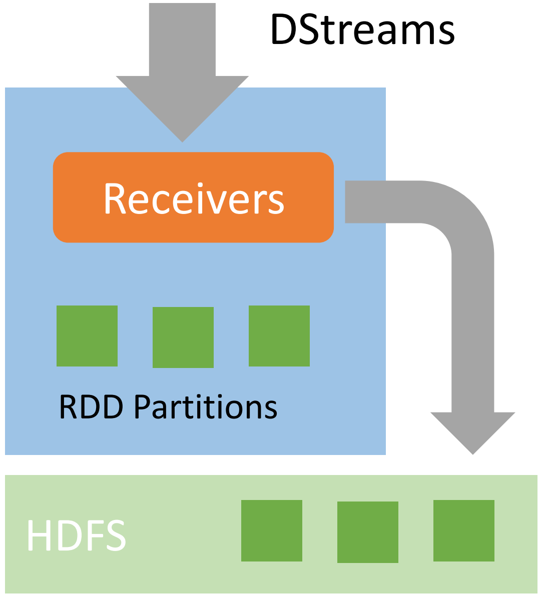
```
context = StreamingContext.getOrCreate(checkpointDirectory, functionToCreateContext)
```

- Start the context

```
context.start()
```

Recover Data using Write Ahead Logs (WAL)

- Enable Checkpoint
- *Set Spark Configurtion*
    - *sparkConf.set(“spark.streaming.receiver.writeAheadLog.enable”, true)*
- Receivers should be reliable
    - Should acknowledge to source after writing to WAL



updateStateByKey

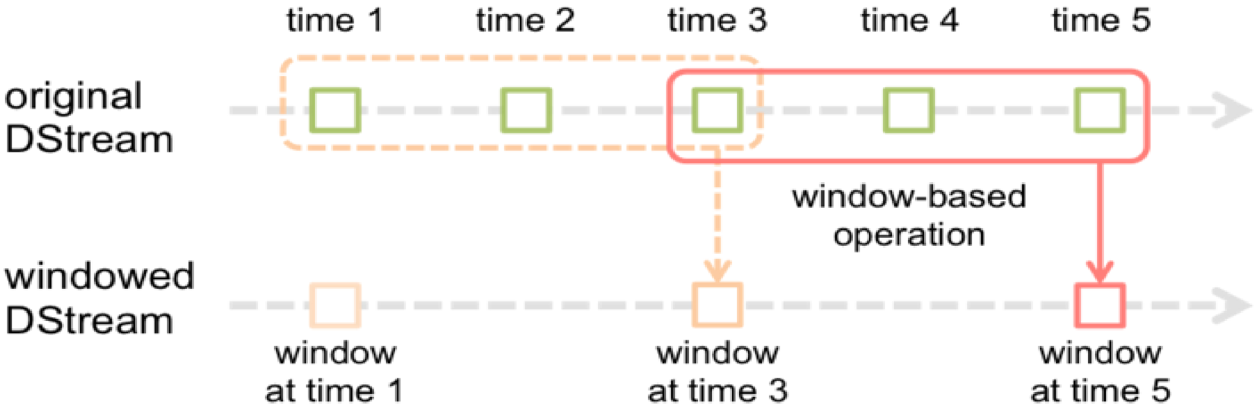
- Maintain state while continuously updating it with new RDD results
- Two steps involved
  - Define state
  - Define a function to update state
- Invoke updateStateByKey on RDDs
  - wordcounts.updateStateByKey(updateCounts)

```
def updateCounts(new_values, last_sum):
    return sum(new_values) + (last_sum or 0)
```

- updateStateByKey requires the checkpoint directory to be configured
  - *streamingContext.checkpoint(checkpointDirectory)*
  - This also helps applications recover from failure

Window Operations

- Applying transformations over a sliding window of data
- RDDs are created by combining multiple batches that fall within the window
- Then window slides at a periodic interval
- Create a window from the Dstream created
  - *windowedStream = stream.window(20)*



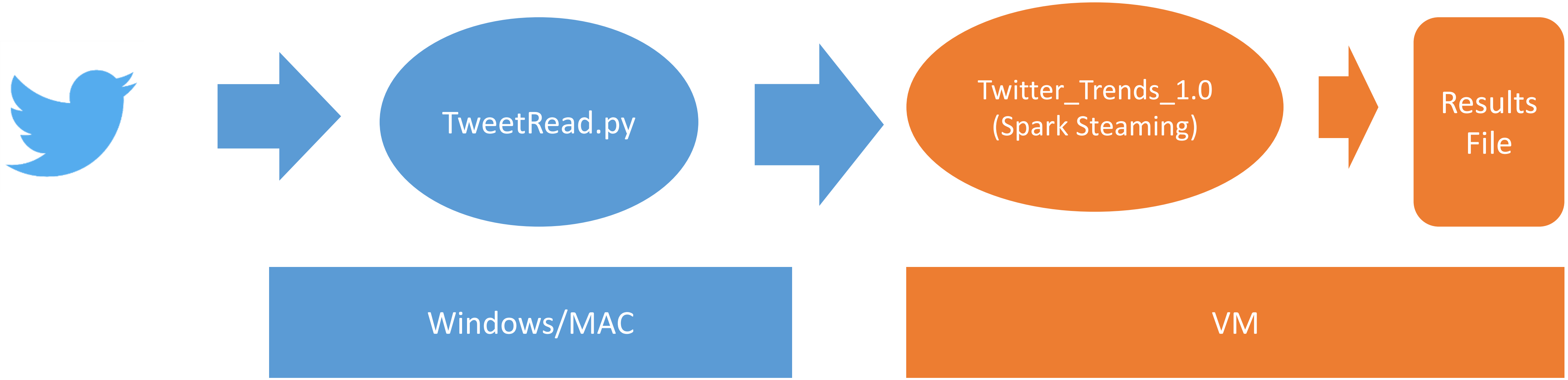
Source: http://spark.apache.org/docs/latest/streaming-programming-guide.html

Writing output

- DStream output can be written using
  - *pprint*
  - *saveAsTextFiles(prefix, [suffix])*
  - *saveAsObjectFiles(prefix, [suffix])*
  - *saveAsHadoopFiles(prefix, [suffix])*
  - Filename will be *prefix-TIME_IN_MS[.suffix]*
- Can also write to custom destinations using *foreachRDD(func)*

  dstream.foreachRDD(lambda rdd: rdd.foreachPartition(sendPartition))

Lab: Twitter Trends using Spark Streaming



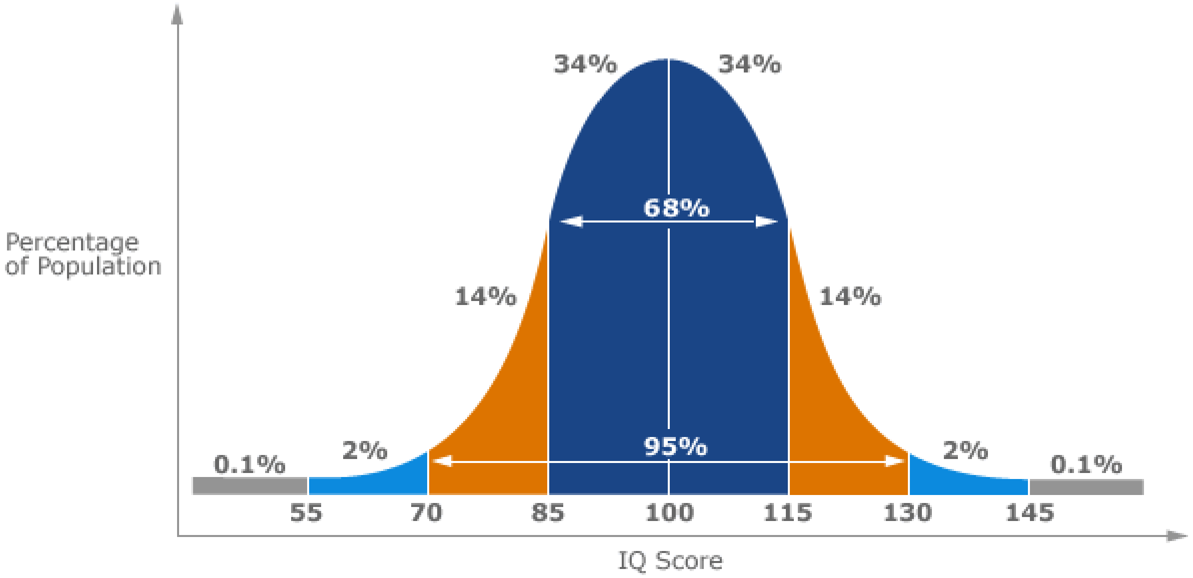
- Acts as a broker.
- Reads the tweets
- Send to the clients, who has connected to this broker (on a tcp port).

- Spark Streaming Application pulling the messages from the broker and processing it.
- Extracts the tags from tweets and counts it over a period (defined as window).
- Writes the tags and their counts to a file.

Visualization, Statistical Analysis & Machine Learning in Spark

Statistics - Normal Distribution

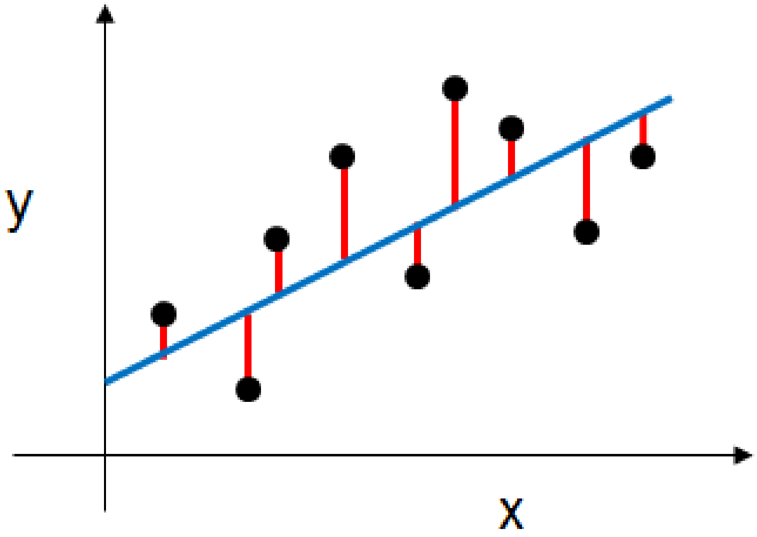
- Normal Distribution
- *Mean, Variance*
- *Quantiles*
- *t-score & p-value*
- *Hypothesis Test*



Linear Regression

- **Linear regression** is an approach for modeling the relationship between a scalar dependent variable *y* and one or more explanatory variables (or independent variables) - wikipedia
- The dependent variable must be a continuous variable
- The relationship is assumed to be linear
- Is a supervised learning
- Use cases
  - Understand marketing effectiveness
  - Pricing and promotions on sales of a product
  - Evaluate trends and make estimates

Linear Regression



$y=\beta_0+\beta_1 x_1+\beta_2 x_2+\ldots+\beta_n x_n$

- y is the response
- β_0 is the intercept
- β_1 is the coefficient for x_1 (the first feature)
- β_n is the coefficient for x_n (the nth feature)

The β values are called the **model coefficients**:

Error of the Estimated line is :

Model Prediction

$$SS_{residuals} = \sum_{i=1}^{N} (\hat{y}_i - y_i)^2$$

Observed Result

Regressions finds the line which minimizes the **sum of squared residuals**. The method is called OLS (Ordinary least square)

Linear Regression – Model Evaluation

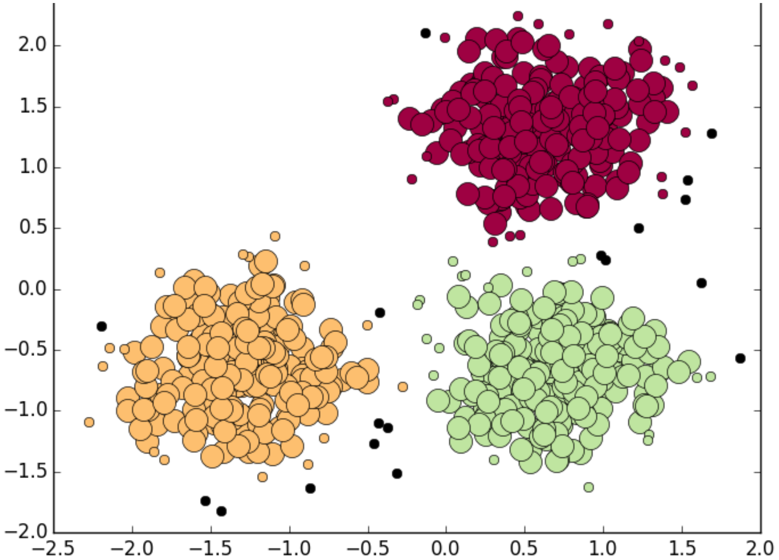
- MAE – Mean Absolute Error
- MSE – Mean Squared Error
  - Penalizes larger residuals
- RMSE – Root Mean Squared Error
- R Square – How much variance explained by the model
  - 1 – all variance explained
  - 0 – no variance explained. It is better not have the model at all.
  - Closer to 1 – is better model.

$$\text{MAE}(y, \hat{y}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} \left| y_i - \hat{y}_i \right| .$$

$$\text{MSE}(y, \hat{y}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} (y_i - \hat{y}_i)^2 .$$

Clustering

- Determine the intrinsic grouping in a set of un-labeled data
  - A *cluster* is a collection of objects which are “similar”
  - Objects belonging to different clusters are dissimilar
- Unsupervised Learning
- *Use Cases*
  - *Marketing*: finding groups of customers with similar behaviour either based on their characteristics or past purchase patterns
  - *Insurance*: identifying groups of motor insurance policy holders with a high average claim cost or attributes
  - Identifying frauds
  - *Biology*: classification of plants and animals given their features;



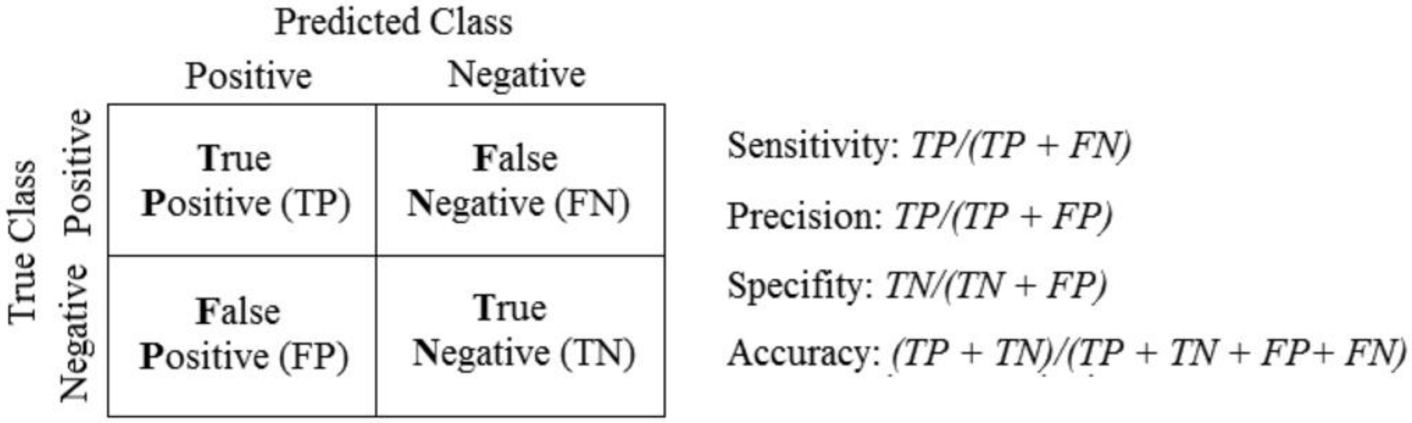
Classification

- **Classification** is identifying to which of a set of categories a new observation belongs
- The number categories can be two or more
- Is a supervised learning
- Most widely used ML technique
- Use Cases
  - Predict if a customer would churn – churn analysis
  - Predict if the loan applicant would default or not
  - NLP – spam mail or not a spam mail
  - Sentiment Analysis
  - Document classification
  - If a patient has a risk of a disease or not

Classification – Model Evaluation

Confusion Matrix

	Spam (Predicted)	Non-Spam (Predicted)	Accuracy
Spam (Actual)	27	6	81.81
Non-Spam (Actual)	10	57	85.07
Overall Accuracy			83.44



		Predicted Class	
		Positive	Negative
True Class	Positive	**True** **P**ositive (TP)	**False** **N**egative (FN)
	Negative	**False** **P**ositive (FP)	**True** **N**egative (TN)

Sensitivity: *TP/(TP + FN)*

Precision: *TP/(TP + FP)*

Specifity: *TN/(TN + FP)*

Accuracy: *(TP + TN)/(TP + TN + FP+ FN)*

Walkthrough Example

- Coronary Heart Disease Data

http://statweb.stanford.edu/~tibs/ElemStatLearn/datasets/SAheart.info.txt

A retrospective sample of males in a heart-disease high-risk region of the Western Cape, South Africa. There are roughly two controls per case of CHD. Many of the CHD positive men have undergone blood pressure reduction treatment and other programs to reduce their risk factors after their CHD event. In some cases the measurements were made after these treatments. These data are taken from a larger dataset, described in Rousseauw et al, 1983, South African Medical Journal.

sbp	systolic blood pressure
tobacco	cumulative tobacco (kg)
ldl	low densiity lipoprotein cholesterol
adiposity	
famhist	family history of heart disease (Present, Absent)
typea	type-A behavior
obesity	
alcohol	current alcohol consumption
age	age at onset
chd	response, coronary heart disease

Spark Tuning & Best Practices

Hardware Provisioning - Best Practices

- Run spark as close to the external storage as possible
  - Run spark nodes on the data nodes of HDFS
  - Spark uses local disks for intermediate data and persist RDDs (if RAM is not sufficient)
  - spark.local.dir should point to a local directory
  - **4-8 disks** per node are recommended (*without* RAID)
  - Each machine should have atleast 8 GB. Allocate 75% of RAM to spark components. Rest should be reserved for OS and other system utilities.
  - Minimum 8 – 16 Cores per machine is recommended
  - The machines should be configured in a network with minimum network bandwidth of 10 gbps

Spark Configurations

- All spark configurations should be provided in ***conf/spark-defaults.conf***
- A property is a key and a value separated by whitespace
  - spark.master spark://5.6.7.8:7077
  - spark.executor.memory 4g
- Properties can also be provided dynamically during runtime
  - ./bin/spark-submit --name "My app" --master local[4] **--conf spark.eventLog.enabled=false**

    --conf "spark.executor.extraJavaOptions=-XX:+PrintGCDetails -XX:+PrintGCTimeStamps" myApp.jar
- Properties can also be set programmatically
  - SparkConf.set(key, value)

Application properties

spark.app.name	Application name
spark.driver.cores	Number of cores for the driver program
spark.driver.maxResultSize	Limit of total size of serialized results of all partitions for each Spark action like collect(). 0 for unlimited.
spark.driver.memory	Total memory allocated for driver program.
spark.executor.memory	Number of worker instances on each slave (default is 1)
spark.local.dir	Local directory for scratch space for spark.
spark.logConf	Log type. INFO is default.
spark.master	Master server urls. Local for local mode. YARN or Mesos or Standalone. Supported formats local local[k] local[*] spark://host:port mesos://host:port yarn

Dependency Management

- Supply Jars to the programs. Automatically transferred to the cluster.
  - ***--jars***
  - URL schemes:
    - file:/// or hdfs:// or http:// or ftp://
    - All in driver’s path
    - Multiple files can be passed with a comma separated list
- Additional packages
  - --packages
- Additional Repositories
  - --repositories
- Supply py files
  - ***--py-files***
- Clean up all dependent files
  - Set *spark.worker.cleanup.appDataTtl* (in seconds). Default is 7 days.

Run time Environment Properties

spark.driver.extraClassPath	Extra classpath entries to prepend to the classpath of the driver.
spark.driver.extraJavaOptions	A string of extra JVM options to pass to the driver. Can also be sent using *--driver-java-options* command line option
spark.executor.extraClassPath	Extra classpath entries to prepend to the classpath of executors.
spark.executor.extraJavaOptions	A string of extra JVM options to pass to executors.
spark.python.worker.memory	Amount of memory to use per python worker process
spark.python.worker.reuse	Reuse Python worker or not.
spark.executor.userClassPathFirst	Whether to give user-added jars precedence over Spark's own jars. Also available for driver class path using *spark.driver.userClassPathFirst*
spark.executor.logs.rolling.maxSize	Set the max size of the file by which the executor logs will be rolled over.
spark.executor.logs.rolling.time.interval	Set the time interval by which the executor logs will be rolled over.

Shuffle and Compression

spark.reducer.maxSizeInFlight	Maximum size of map outputs to fetch simultaneously from each reduce task. (Default 48m)
spark.shuffle.compress	Whether to compress map output files. Default is true.
spark.shuffle.file.buffer	Size of the in-memory buffer for each shuffle file output stream. Default 32k.
spark.shuffle.spill.compress	Whether to compress data spilled during shuffles. Default is true.
spark.rdd.compress	Whether to compress serialized RDD partitions. RAM space can be saved at the cost of CPU cycles needed for decompression. Default is false.
spark.serializer	Class to use for serializing objects that will be sent over the network or need to be cached in serialized form. Java Serialization is slow. *org.apache.spark.serializer.KryoSerializer* is recommended for faster serialization and deserialization.
spark.io.compression.codec	The codec used to compress internal data such as RDD partitions, broadcast variables and shuffle outputs. Default is snappy. Spark supports lz4, lzf and snappy out of the box.

Memory Management & Networking

spark.memory.fraction	Fraction of (heap space - 300MB) used for execution and storage. (0.75). Lower values may result in frequent spills and cached data eviction.
spark.memory.storageFraction	Amount of storage memory immune to eviction, expressed as a fraction of spark.memory.fraction. (0.5)
spark.akka.frameSize	Maximum message size (in MB) to allow in "control plane" communication. (128)
spark.driver.port	Port for driver to listen to on their local hosts.
spark.executor.port	Port for executors to listen to on their local hosts.
spark.network.timeout	Default timeout for all network interactions. (120s)
spark.port.maxRetries	Maximum number of retries when binding to a port before giving up. (16)

- Execution memory refers to that used for computation in shuffles, joins, sorts and aggregations.
- Storage memory refers to that used for caching and propagating internal data across the cluster.
- Size of an RDD (in memory or disk) can be viewed from storage link on SparkContext web UI

Spark Scheduling

spark.cores.max	Maximum amount of CPU cores to request for the application from across the cluster. Default not set.
spark.scheduler.mode	FIFO or FAIR. Default is FIFO.
spark.speculation	Performs speculative execution of tasks. Slower tasks are re-launched for competition. Default is false.
spark.task.cpus	Number of cores to allocate for each task.(1)
spark.task.maxFailures	Number of individual task failures before giving up on the job. (4)
spark.network.timeout	Default timeout for all network interactions. (120s)
spark.locality.wait	How long to wait to launch a data-local task before giving up and launching it on a less-local node.(3s)

More Tuning Best Practices

- Use Kryo serializer (not java serializers)
- Allocate appropriate memory to spark.memory.fraction and spark.memory.storageFraction
- Consider using numeric IDs or enumeration objects instead of strings for keys.
- For JVM heap allocation less than 32 GB of RAM, set the JVM flag - XX:+UseCompressedOops to make pointers be four bytes instead of eight.
- For faster persist(), use **_SER*, which stored data in one large byte array.
- Use *KryoSerializer* for serialization and deserlization
- Number of map tasks can automatically be set by using file size, but reduce tasks need to be set appropriately to increase the level of parallelism.
  - Increase the level of parallelism by *spark.default.parallelism or repartition* the data in code. Recommended value is 2-3 tasks per CPU core in cluster.

More Tuning Best Practices

- *Driver* is a Single point failure. Do, avoid collect(), which may result in driver crash.
- *Prefer reduceByKey over groupByKey()*
- *For joining large rdd with small rdd use BroadcastHashJoin*
  - *Two different kinds of joins – ShuffledHashJoin and BroadcastHashJoin*
  - *Set to spark.sql.autoBroadcastJoinThreshhold to rdd size below which rdds would be broadcasted.*
- Avoid calling collect() in driver
- Do not define variables in driver and use them in the map function. This will result in the variable being serialized multiple times.
  - Use broadcast variable
- Do not create connections to external datasets inside map functions
  - Like database or socket connections

Spark - Capstone Project