INTRODUCTION TO APACHE SPARK

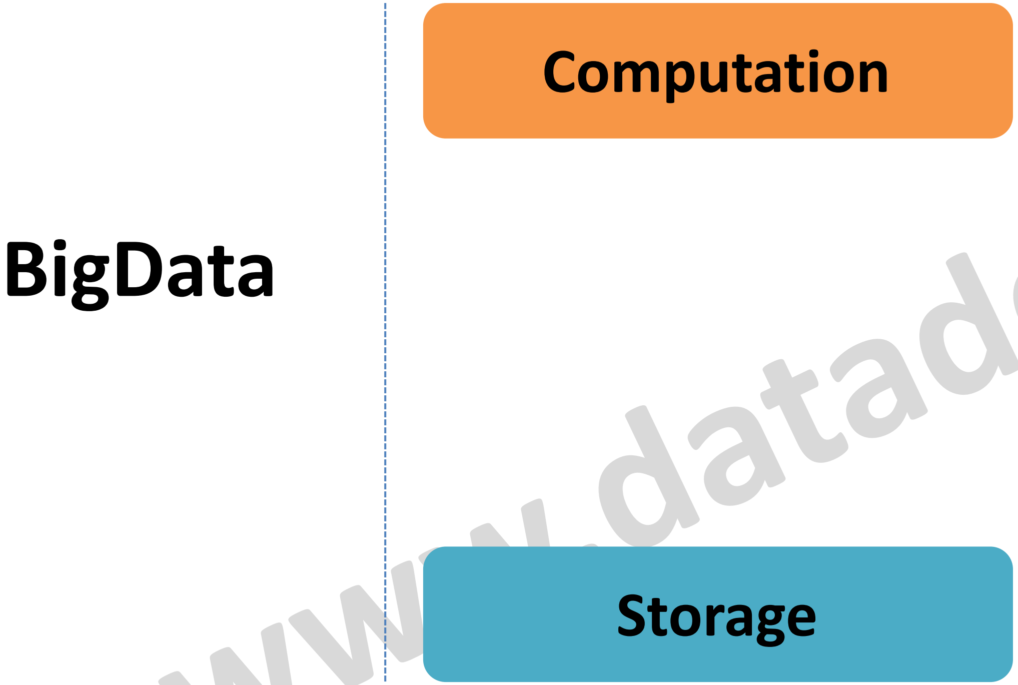
DATA DOTZ

Agenda

- Introduction to Spark
- Transformations and Actions
- Spark Architecture
- Hadoop and Spark
- Spark vs MapReduce

References

- Apache Spark Site - http://spark.apache.org/
- Spark Mailing List
- Blogs
  - Cloudera – http://blog.cloudera.com
  - DataBricks – http://databricks.com/blog
  - MapR - http://www.mapr.com/blog
  - HortonWorks - http://hortonworks.com/blog



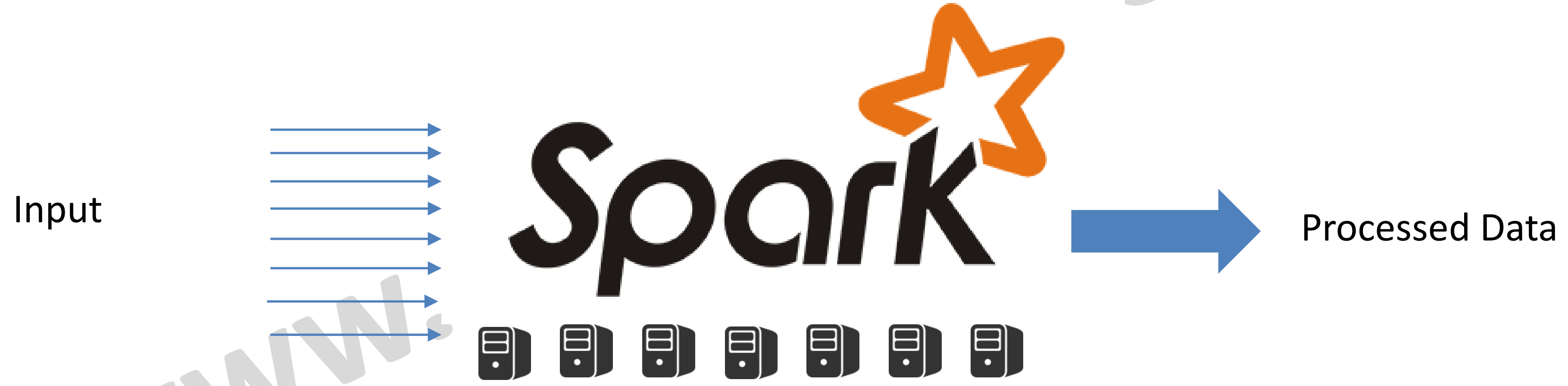
BigData

Computation

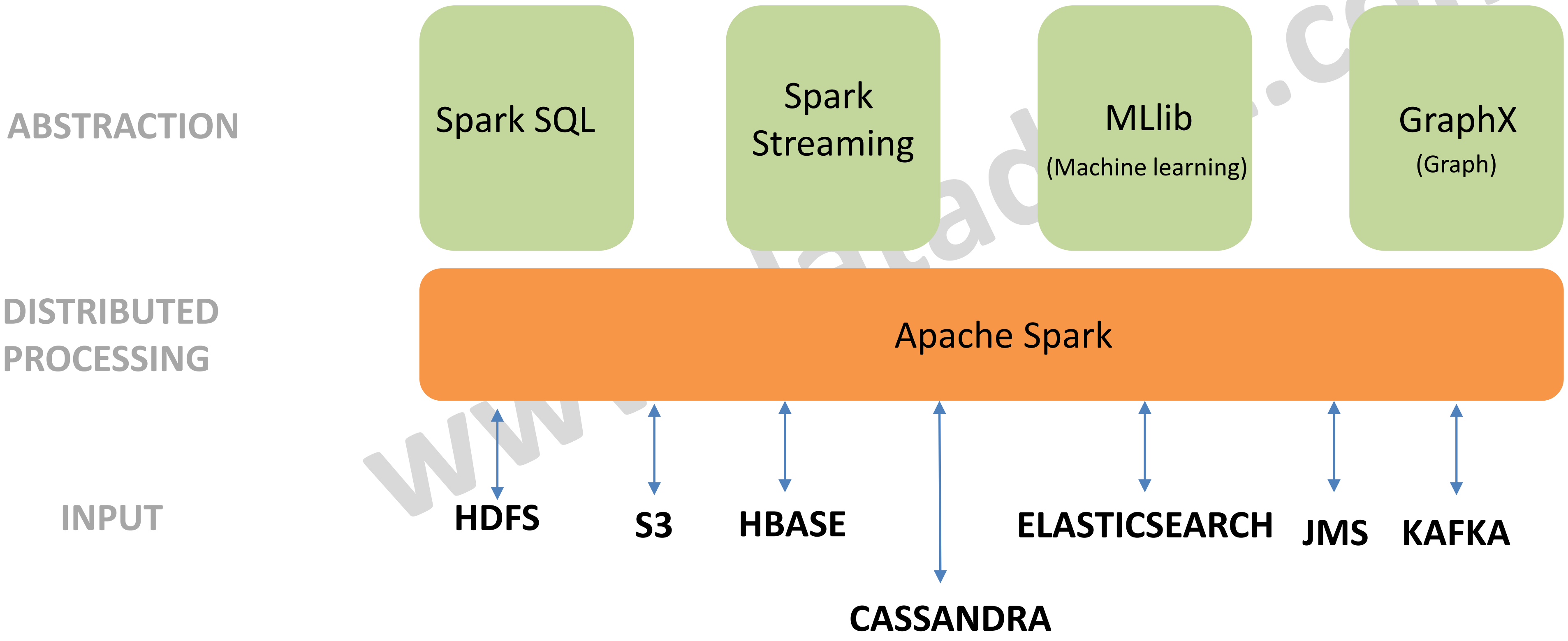
Storage

What is Apache Spark

A Framework written in Scala which provides distributed data processing



Spark FrameWork



ABSTRACTION

Spark SQL

Spark Streaming

MLlib (Machine learning)

GraphX (Graph)

DISTRIBUTED PROCESSING

Apache Spark

INPUT

HDFS

S3

HBASE

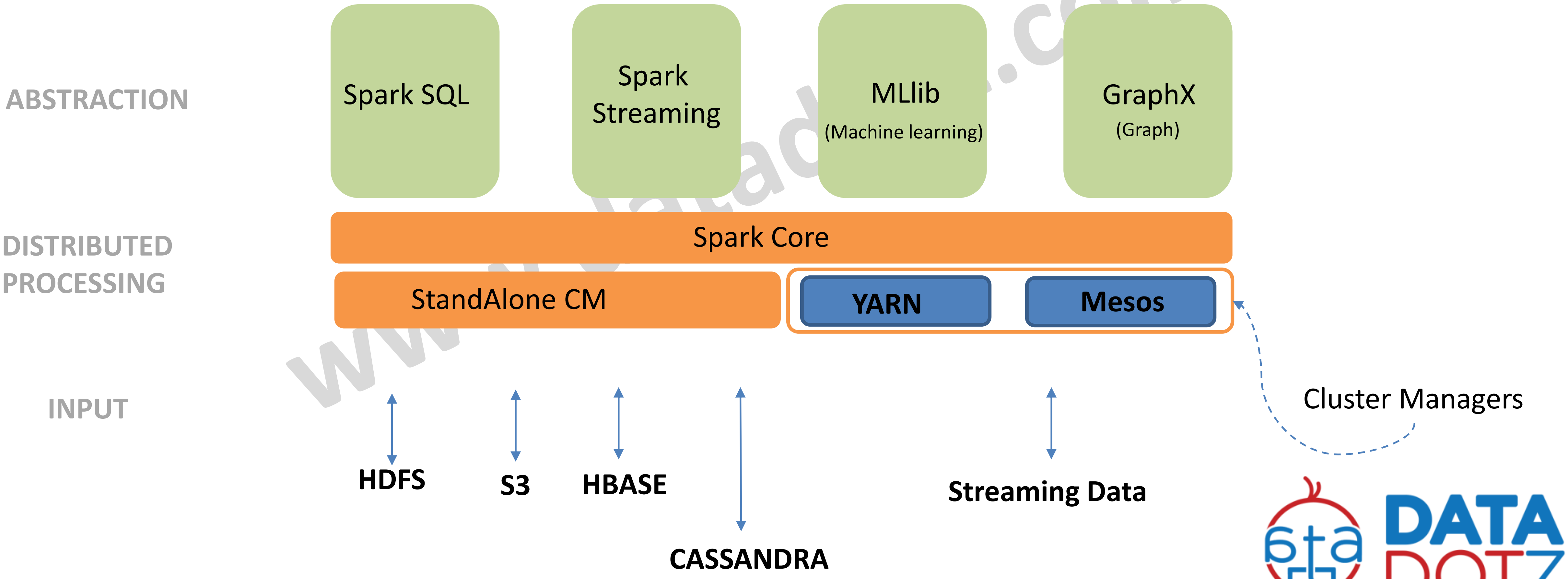
CASSANDRA

ELASTICSEARCH

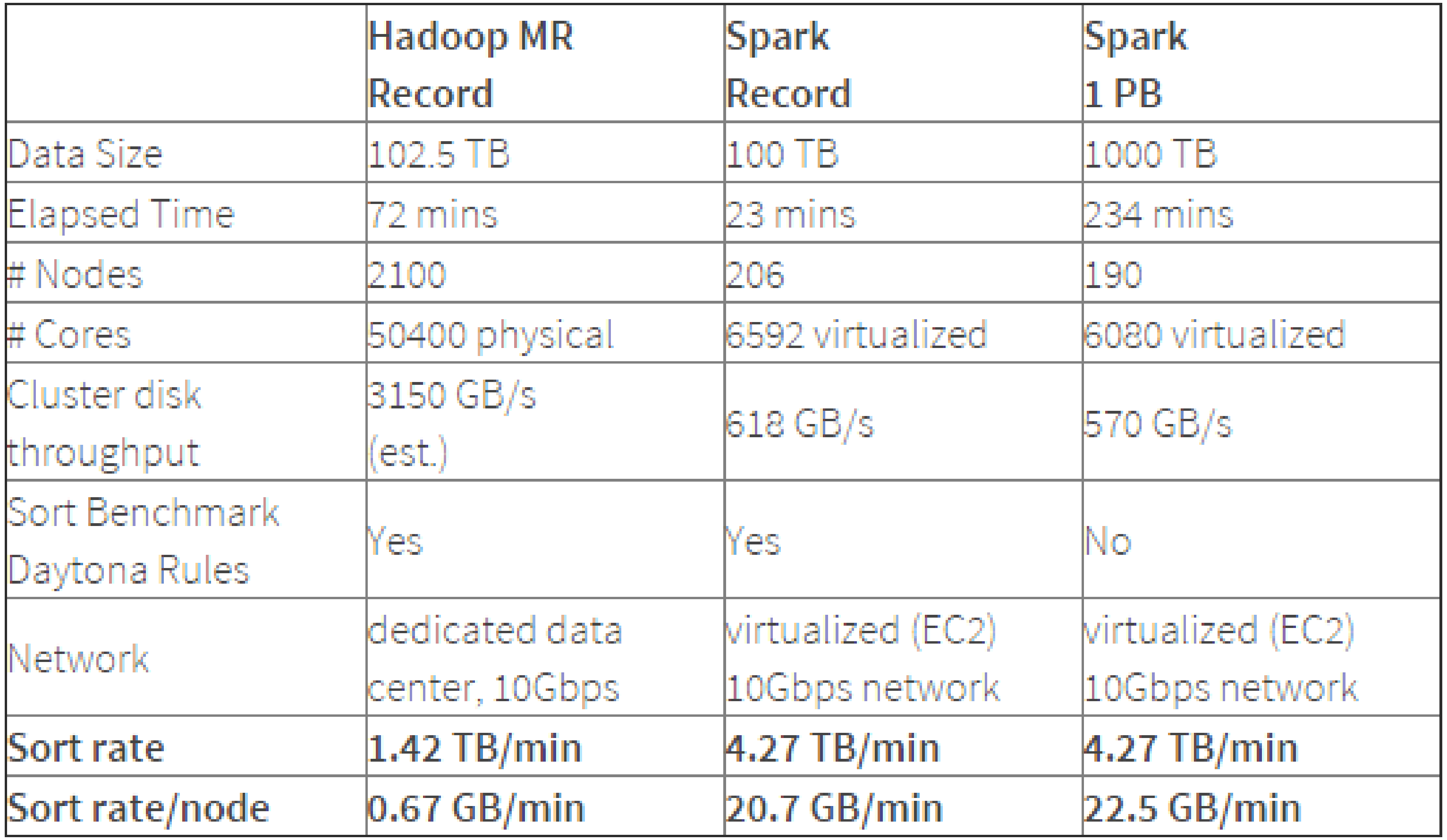
JMS

KAFKA

Spark FrameWork



Spark officially sets a new record in large-scale sorting



	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**

Reference : https://databricks.com/blog/2014/11/05/spark-officially-sets-a-new-record-in-large-scale-sorting.html

Who Uses Apache Spark?

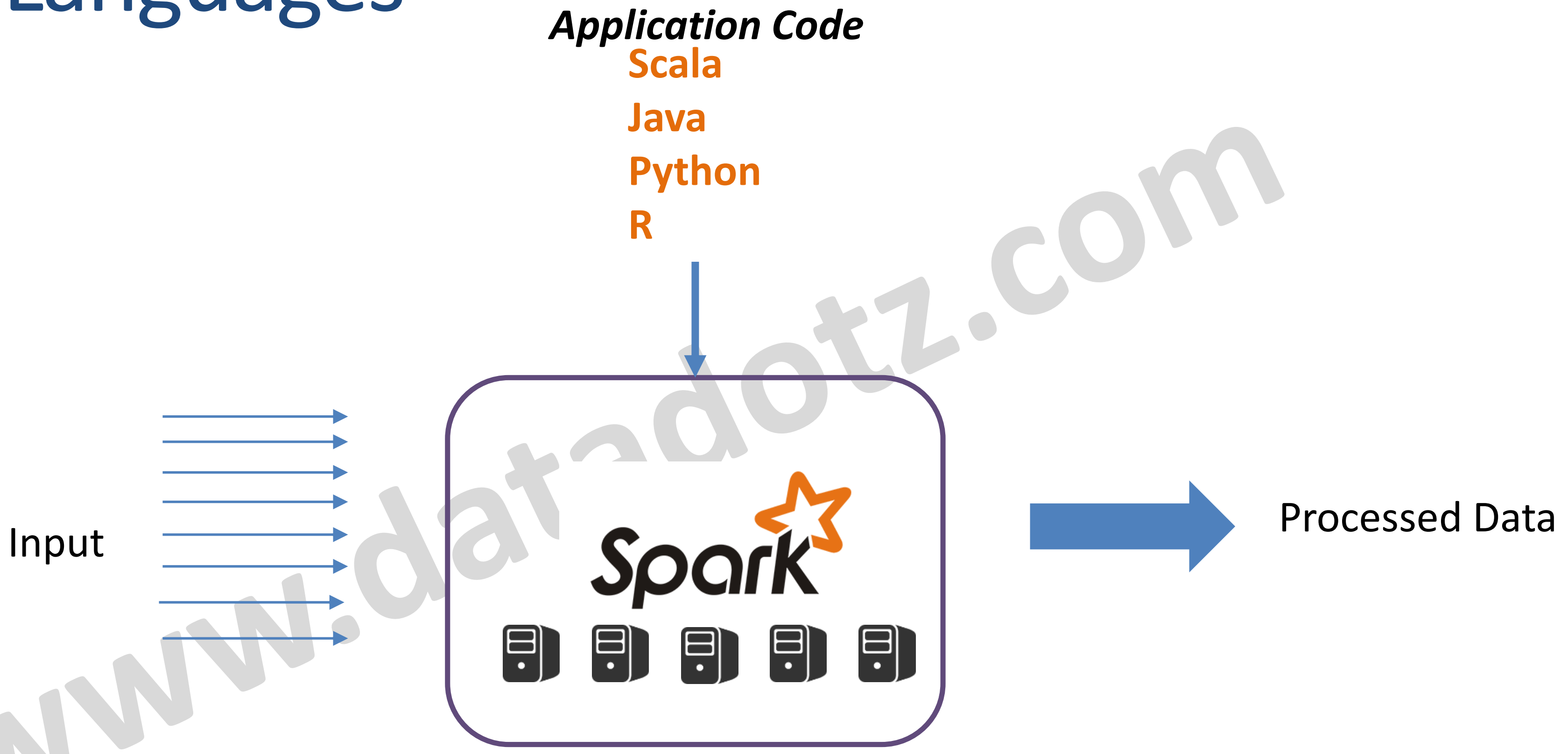
YAHOO!
CONVIVA
shopify
OOYALA
guavus
淘宝网
Taobao
CONCUR
tuplejump
Paxata
GROUPON
myfitnesspal
STRATIO
ClearStory DATA
Now You See It

Reference : https://cwiki.apache.org/confluence/display/SPARK/Powered+By+Spark

History

- Started in UC Berkeley in 2009
  - Founder – Matei Zaharia
- Open Sourced in 2010 under BSD License
- 2011 – Higher Components as part of BDAS Stack
- 2013 – Joined ASF under Apache 2.0 License
- 2013 – DataBricks Founded
- Feb 2014 - Top Level Project in Apache
- May 2014 – Spark -1.0
- Nov 2014 - World record in large scale sorting by DataBricks Team

Supported Languages



**Spark Source Code is written in Scala

Requirements

- OS
  - Windows, Linux, Mac OS
- JAVA_HOME
- SCALA PATH

Installation Mode

- Interactive Shell for adhoc analysis or learning
  - Spark Shell – Interactive REPL
  - Can run locally or connect to a Spark cluster
- Cluster
  - Standalone
    - Amazon EC2
  - Mesos
  - YARN (Hadoop)

Spark Installation

- Download
  - https://spark.apache.org/downloads.html
- Please build binary for your requirement.
  - For shell
    - *bin/sbt assembly*
    - *bin/sbt -Pyarn -Phadoop-2.6 -Phive -Phive-thriftserver assembly*
- Source Code
  - http://github.com/apache/spark

Running Spark Shell

- Command
  - Scala - **bin/spark-shell**
  - Python – **bin/pyshark**

Spark Context

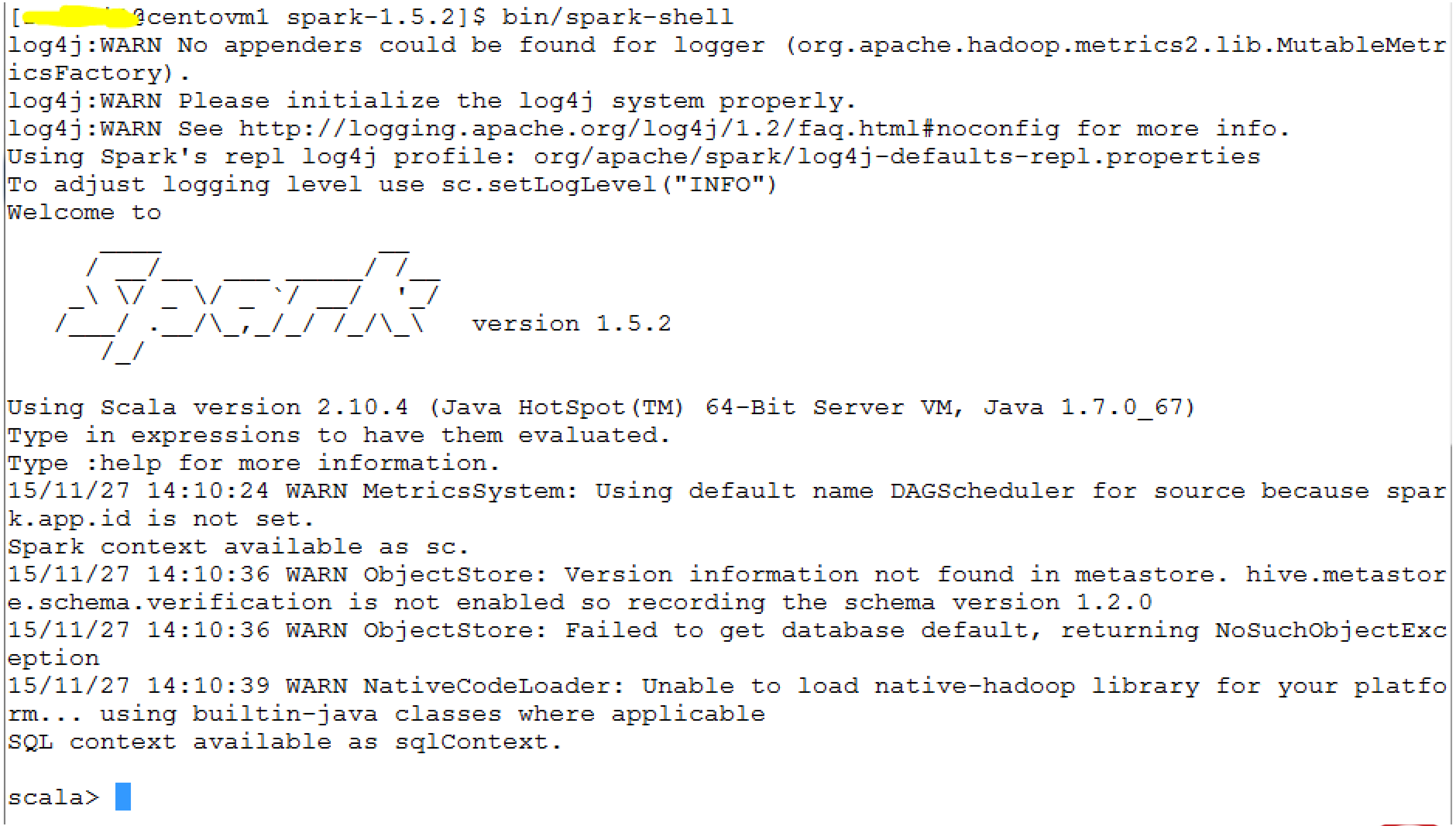
- Single Entry Point of the Spark Application (driver)
- Spark Shell provides a preconfigured Spark Context “***sc***”

Scala

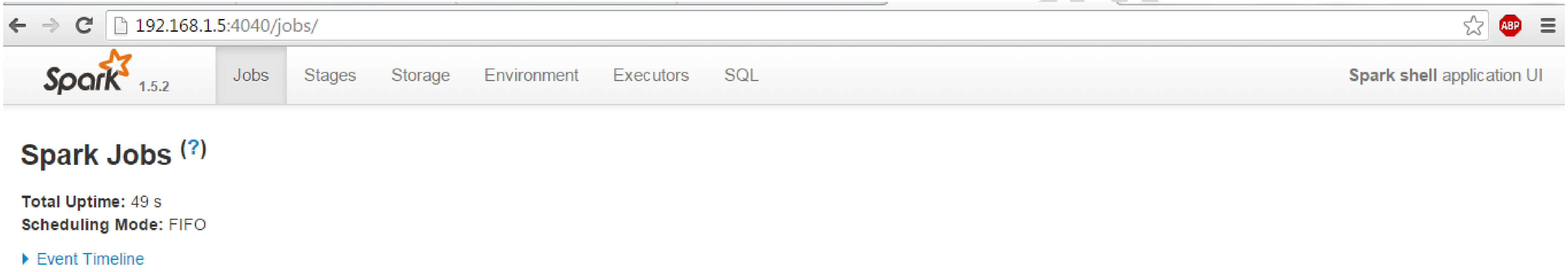
```
scala> sc.appName
res1: String = Spark shell
```

Python

```
>>> sc.appName
u'PySparkShell'
```



```
[[redacted]@centovm1 spark-1.5.2]$ bin/spark-shell
log4j:WARN No appenders could be found for logger (org.apache.hadoop.metrics2.lib.MutableMetr
icsFactory).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
Using Spark's repl log4j profile: org/apache/spark/log4j-defaults-repl.properties
To adjust logging level use sc.setLogLevel("INFO")
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 1.5.2
      /_/

Using Scala version 2.10.4 (Java HotSpot(TM) 64-Bit Server VM, Java 1.7.0_67)
Type in expressions to have them evaluated.
Type :help for more information.
15/11/27 14:10:24 WARN MetricsSystem: Using default name DAGScheduler for source because spar
k.app.id is not set.
Spark context available as sc.
15/11/27 14:10:36 WARN ObjectStore: Version information not found in metastore. hive.metastor
e.schema.verification is not enabled so recording the schema version 1.2.0
15/11/27 14:10:36 WARN ObjectStore: Failed to get database default, returning NoSuchObjectExc
eption
15/11/27 14:10:39 WARN NativeCodeLoader: Unable to load native-hadoop library for your platfo
rm... using builtin-java classes where applicable
SQL context available as sqlContext.

scala>
```



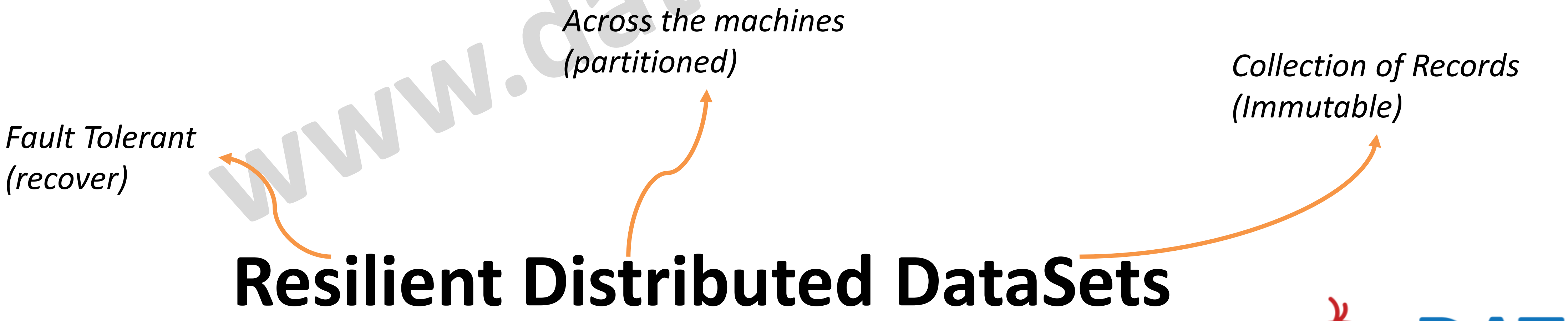
Spark Jobs (?)

Total Uptime: 49 s
Scheduling Mode: FIFO

▸ Event Timeline

First Spark Program using Scala

```
scala> val data = Array(1, 2, 3, 4, 5)
data: Array[Int] = Array(1, 2, 3, 4, 5)

# create RDD by parallelizing the collection
# Alternative reading it from distributed storage such as HDFS, NoSQL, ..etc
scala> val distData = sc.parallelize(data)
distData: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[0] at parallelize at <console>:14
```



Across the machines (partitioned)

Collection of Records (Immutable)

Fault Tolerant (recover)

Resilient Distributed DataSets

- Read Only , partitioned collection of records

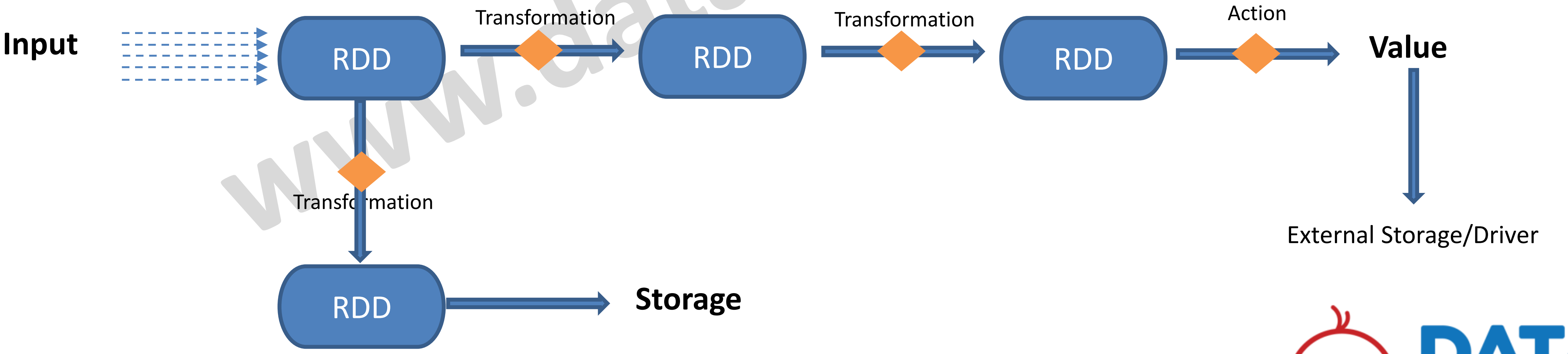
RDD operations

Transformations

- Create a New DataSet from a New DataSet
- Transformations are lazy operations
- RDDs are operated when an action is run on it.

Actions

- Compute Values
- Return Values or write Output to external Storage
- Earlier Transformations are applied to RDD
  - Since transformations are lazy operations

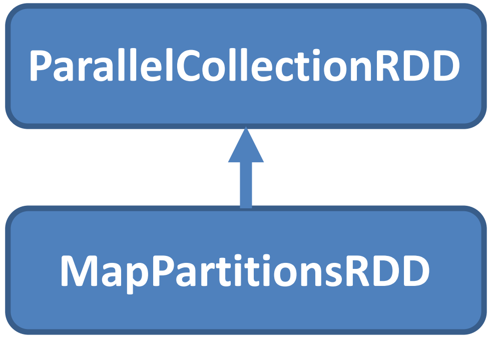


Select only records less than 3

```
scala> val data = Array(1, 2, 3, 4, 5)
data: Array[Int] = Array(1, 2, 3, 4, 5)

# create RDD
scala> val distData = sc.parallelize(data)
distData: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[0] at parallelize at <console>:23

# filter RDDs for elements less than 3
# Transformation – create RDDs from existing RDDs
# Transformation are Lazy operations
scala> val filteredData = distData.filter( _ < 3 )
filteredData: org.apache.spark.rdd.RDD[Int] = MapPartitionsRDD[1] at filter at <console>:25

# collect or obtain output RDDs
# Actions – calculate values from RDDs
scala> val resultArray= filteredData.collect()
resultArray: Array[Int] = Array(1, 2)
```

Lineage of RDDs – Fault Recovery

- RDDs maintain their Lineage
  - Any intermediate RDDs is missing , it can calculate from its Parent RDDs



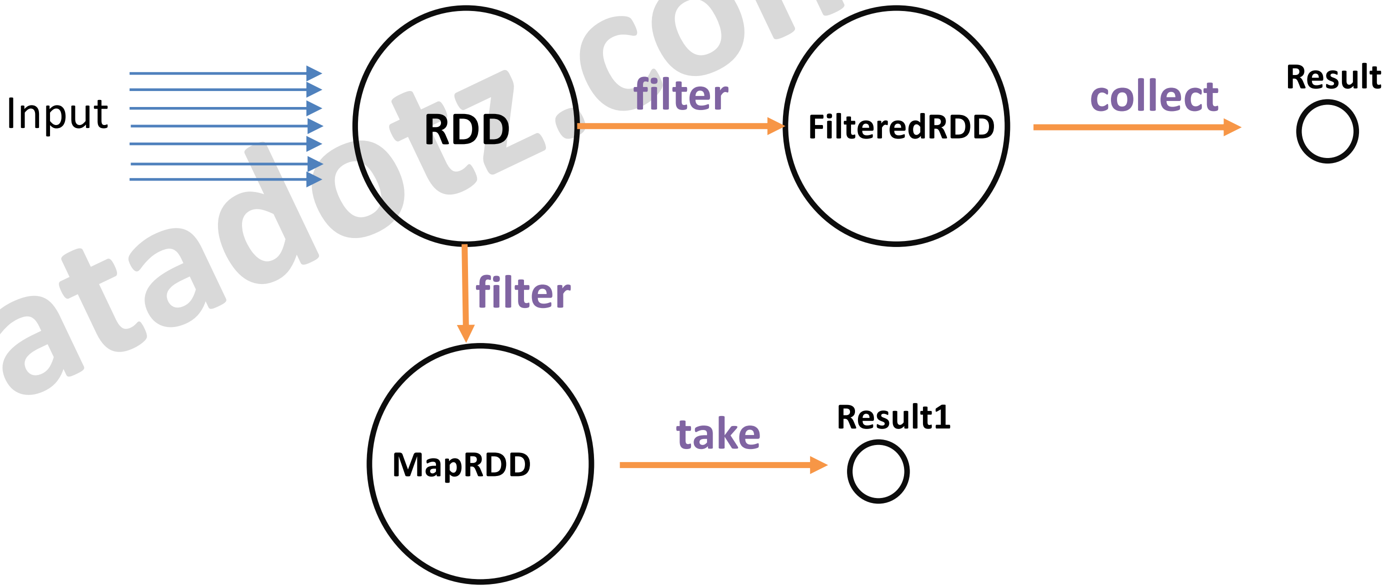
ParallelCollectionRDD

MapPartitionsRDD

```
scala> val data = Array(1, 2, 3, 4, 5)
data: Array[Int] = Array(1, 2, 3, 4, 5)

scala> val distData = sc.parallelize(data)
distData: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[0] at parallelize at <console>:23

scala> val filteredData = distData.filter( _ < 3 )
filteredData: org.apache.spark.rdd.RDD[Int] = MapPartitionsRDD[1] at filter at <console>:25

scala> filteredData.toDebugString
res3: String =
(1) MapPartitionsRDD[2] at filter at <console>:25 []
 |  ParallelCollectionRDD[0] at parallelize at <console>:23 []
```

Directed Acylic Graph of RDDs

- Directed
  - Only in a Single Direction
- Acyclic
  - No Looping
- Provides Fault-Tolerance
  - By providing ***Lineage***



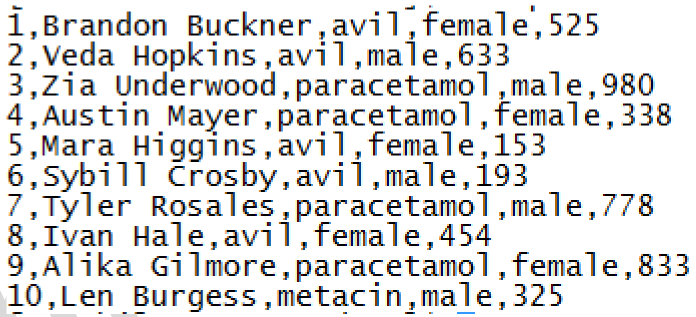
Filter – contd..

```
scala> val distData = sc.parallelize(Array(1, 2, 3, 4, 5))
distData: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[0] at parallelize at <console>:23

# filter – takes a  condition function
# Any RDD which satisfies the condition will added to resultant RDD
scala> val filteredData = distData.filter( i => i %2 == 0)
filteredData: org.apache.spark.rdd.RDD[Int] = MapPartitionsRDD[1] at filter at <console>:25

scala> filteredData.collect()
res1: Array[Int] = Array(2, 4)
```

Partitions

```
scala> val distData = sc.parallelize(Array(1, 2, 3, 4, 5))
distData: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[0] at parallelize at <console>:23

# RDD -> read only partitioned  collection of records
scala> distData.partitions.length
res8: Int = 1

# custom parallelism with custom partitions
scala> val distData = sc.parallelize(Array(1, 2, 3, 4, 5),3)
distData: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[0] at parallelize at <console>:23

scala> distData.partitions.length
res9: Int = 3
```

Data From Local file



```
1,Brandon Buckner,avil,female,525
2,Veda Hopkins,avil,male,633
3,Zia Underwood,paracetamol,male,980
4,Austin Mayer,paracetamol,female,338
5,Mara Higgins,avil,female,153
6,Sybill Crosby,avil,male,193
7,Tyler Rosales,paracetamol,male,778
8,Ivan Hale,avil,female,454
9,Alika Gilmore,paracetamol,female,833
10,Len Burgess,metacin,male,325
```

Select drug, sum(amount) from patient group by drug;

```
scala> val patient = sc.textFile("/home/user/data/patient.txt")
patient: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[11] at textFile at <console>:21

scala> val mappedKVs = patient.map(line => (line.split(",")(2),line.split(",")(4).toInt))
mappedKVs: org.apache.spark.rdd.RDD[(String, Int)] = MapPartitionsRDD[13] at map at <console>:23

scala> val result = mappedKVs.reduceByKey(_+_, 1)
result: org.apache.spark.rdd.RDD[(String, Int)] = ShuffledRDD[14] at reduceByKey at <console>:25

scala> result.collect()
res2: Array[(String, Int)] = Array((avil,1958), (metacin,325), (paracetamol,2929))
```

Basic Actions in Spark

Actions

```
scala> val distData = sc.parallelize(Array(1, 2, 3, 4, 5))

scala> distData.collect()
res3: Array[Int] = Array(1, 2, 3, 4, 5)

scala> distData.take(3)
res0: Array[Int] = Array(1, 2, 3)

scala> distData.take(3)
res0: Array[Int] = Array(1, 2, 3)

scala> distData.top(1)
res1: Array[Int] = Array(5)

scala> distData.reduce(_+_)
res2: Int = 15

scala> distData.first()
res0: Int = 1

scala> distData.count()
res2: Long = 5
```

Actions

```
scala> val distData = sc.parallelize(Array(1, 2, 3, 4, 5))

scala> println("Hello, world!")
Hello, world!

scala> distData.foreach(println)
11
12
13
14
15

scala> distData.saveAsTextFile("/home/user/resultdir")
```

Actions

reduce(function)

collect()

count()

countByValue()

first()

take(n)

takeSample(withReplacement, num, [seed])

takeOrdered(n, [ordering])

saveAsTextFile(path)

saveAsSequenceFile(path)

saveAsObjectFile(path)

countByKey()

foreach(function)

foreachPartition()

treeAggregate()

treeReduce()

Basic Transformations in Spark

Transformation -

```
scala> val distData = sc.parallelize(Array(1, 2, 3, 4, 5))
distData: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[2] at parallelize at <console>:12

scala> val mappedData = distData.map(x => x*x). collect()
res1: Array[Int] = Array(1, 4, 9, 16, 25)


scala> val filteredData = distData.filter(x => x>= 3). collect()
res1: Array[Int] = Array(3, 4, 5)

scala> distData.flatMap(_.toUpperCase).collect()
res4: Array[Char] = Array(A, P, P, L, E, T, E, S, T)
```

KeyValue Pair

Scala

```
scala> val kvData = (1, "senthil")
kvData: (Int, String) = (1,senthil)

scala> kvData._1
res2: Int = 1

scala> kvData._2
res3: String = senthil
```

Python

```
>>> kvData = (1,"senthil")
>>> kvData[0]
1
>>> kvData[1]
'senthil'
```

Java

```
import scala.Tuple2;
.
.
Tuple2 kvData = new Tuple2(1,"senthil");
System.out.println(kvData._1);
System.out.println(kvData._2);
```

Tuple - sequence of immutable objects
In Scala – Tuple2 forms KeyValue Pair in Spark

Basic Key Value Pairs

```
scala> val Data= sc.parallelize(Array("Apple-1","orange-3","Apple-4"))
Data: org.apache.spark.rdd.RDD[String] = ParallelCollectionRDD[14] at parallelize at <console>:12

# create Key Value pair RDD
# map (Transformation) – converts RDD into KeyValue Pair
scala> val mappedKVs = Data.map(element => (element.split("-")(0), element.split("-")(1).toInt))
mappedKVs: org.apache.spark.rdd.RDD[(String, Int)] = MappedRDD[15] at map at <console>:14

# collect all KV pair RDDs
scala> mappedKVs.collect()
res12: Array[(String, Int)] = Array((Apple,1), (orange,3), (Apple,4))

# collect Keys alone
scala> mappedKVs.keys.collect()
res8: Array[String] = Array(Apple, orange, Apple)

# collect Values alone
scala> mappedKVs.values.collect()
res9: Array[Int] = Array(1, 3, 4)
```

Basic KeyValue Transformations

```
scala> val kvData= sc.parallelize(Array(("Apple",1),("orange",3),("Apple",4)))
kvData: org.apache.spark.rdd.RDD[(String, Int)] = ParallelCollectionRDD[8] at parallelize at <console>:12

scala> kvData.reduceByKey(_+_).collect()
res7: Array[(String, Int)] = Array((orange,3), (Apple,5))

scala> kvData.groupByKey().collect()
res8: Array[(String, Iterable[Int])] = Array((orange,CompactBuffer(3)), (Apple,CompactBuffer(1, 4)))

scala> kvData.sortByKey().collect()
res9: Array[(String, Int)] = Array((Apple,1), (Apple,4), (orange,3))

scala> kvData.sortByKey(false).collect()
res0: Array[(String, Int)] = Array((orange,3), (Apple,1), (Apple,4))
```

Other Transformations on PairRDD

- subtractByKey
  - Remove elements for keys in second RDD
- Joins
  - Works on two RDDS
  - Join, RightOuterJoin, LeftOuterJoin
  - Internally cogroup. Cogroup can be used to work on **more than** two RDDs at the same time.

Additional Actions on PairRDD

```
scala> val kvData= sc.parallelize(Array(("Apple",1),("orange",3),("Apple",4)))
kvData: org.apache.spark.rdd.RDD[(String, Int)] = ParallelCollectionRDD[8] at parallelize at <console>:12

scala> kvData.countByKey()
res9: scala.collection.Map[String,Long] = Map(orange -> 1, Apple -> 2)

scala> kvData.collectAsMap()
res10:  scala.collection.Map[String,Int] = Map(orange -> 3, Apple -> 4)

scala> kvData. lookup("Apple")
res11:  Seq[Int] = WrappedArray(1, 4)
```

Transformations

map(*function*)

filter(*function*)

filterByRange(*lower, upper*)

flatMap(*function*)

mapPartitions(*function*)

mapPartitionsWithIndex(*function*)

sample(*withReplacement, fraction, seed*)

union(*otherDataset*)

intersection(*otherDataset*)

distinct(*[numTasks]*)

groupByKey(*[numTasks]*)

reduceByKey(*function, [numTasks]*)

aggregateByKey(*zeroValue*)(*seqOp, combOp, [numTasks]*)

sortByKey(*[ascending], [numTasks]*)

join(*otherDataset, [numTasks]*)

cogroup(*otherDataset, [numTasks]*)

cartesian(*otherDataset*)

pipe(*command, [envVars]*)

coalesce(*numPartitions*)

repartition(*numPartitions*)

repartitionAndSortWithinPartitions(*partitioner*)

Numerical RDD Operations

```
scala> val distData = sc.parallelize(Array(1, 2, 3, 4, 5))
kvData: org.apache.spark.rdd.RDD[(String, Int)] = ParallelCollectionRDD[8] at parallelize at <console>:12

# Statitics Opertaions on the Data
# Returned StatsCounter object by calling stats
scala> distData.stats()
res7: org.apache.spark.util.StatCounter = (count: 5, mean: 3.000000, stdev: 1.414214, max: 5.000000, min: 1.000000)

# Call direct methods if needed
scala> distData.max()
res8: Int = 5
```

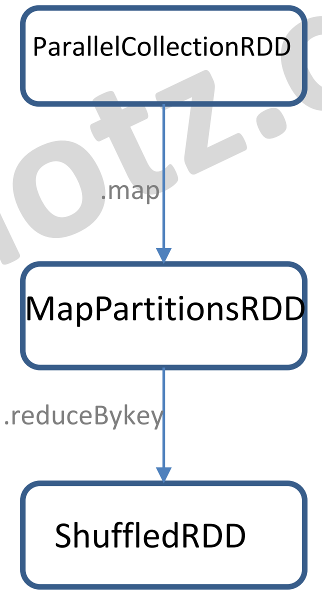
Logical DAG

```
scala> val Data= sc.parallelize(Array("Apple-1","orange-3","Apple-4"))
Data: org.apache.spark.rdd.RDD[String] = ParallelCollectionRDD[9] at parallelize at <console>:21

scala> val mappedKVs = Data.map(line => (line.split("-")(0),line.split("-")(1).toInt))
mappedKVs: org.apache.spark.rdd.RDD[Int] = MapPartitionsRDD[7] at map at <console>:23

scala> val result = mappedKVs.reduceByKey(_+_)
result: org.apache.spark.rdd.RDD[(String, Int)] = ShuffledRDD[9] at reduceByKey at <console>:25

scala> result.toDebugString
res4: String =
(1) ShuffledRDD[11] at reduceByKey at <console>:25 []
 +-(1) MapPartitionsRDD[[10] at map at <console>:23 []
    |  ParallelCollectionRDD[9] at parallelize at <console>:21 []
```

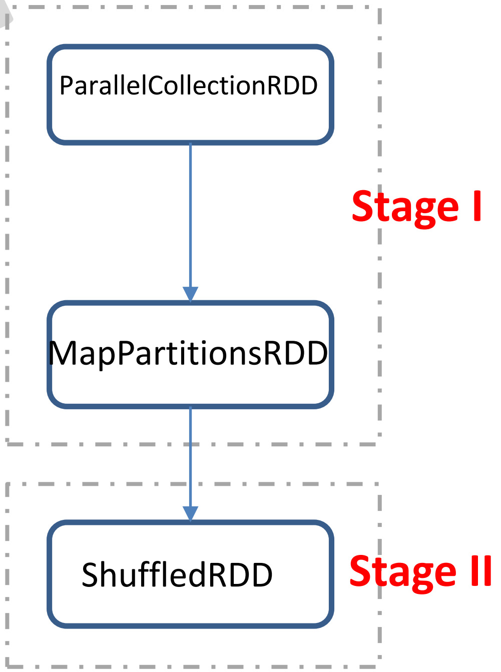
Stage II

Stage I

DAG Graph



Physical Plan



Application -> Jobs -> Stages -> Tasks

- Application contains only one Spark Context
- For Every Action, it creates a Job
- Each Job consists of Stages
- Stage consists of Tasks for each partition in that RDD
  - Tasks -> computation on each partition of the Data

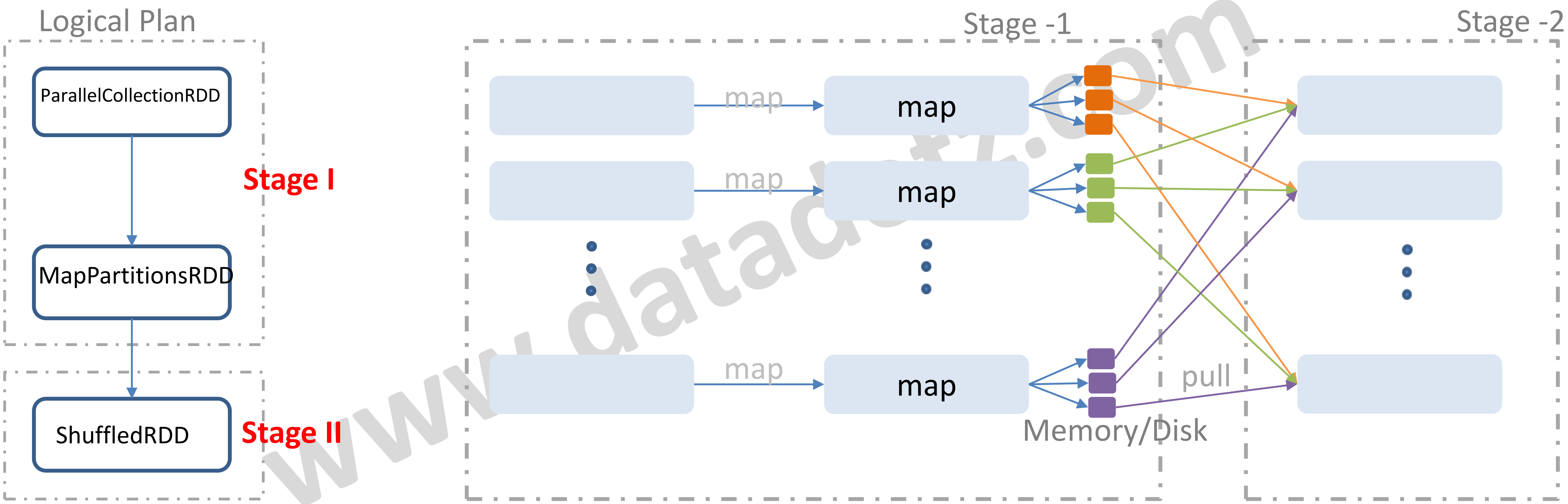
```
scala> val Data= sc.parallelize(Array("Apple-1","orange-3","Apple-4"))
Data: org.apache.spark.rdd.RDD[String] = ParallelCollectionRDD[9] at parallelize at <console>:21

scala> val mappedKVs = Data.map(line => (line.split("-")(0),line.split("-")(1).toInt))
mappedKVs: org.apache.spark.rdd.RDD[Int] = MapPartitionsRDD[7] at map at <console>:23

scala> val result = mappedKVs.reduceByKey(_+_)
result: org.apache.spark.rdd.RDD[(String, Int)] = ShuffledRDD[9] at reduceByKey at <console>:25

scala> result.toDebugString
res4: String =
(1) ShuffledRDD[11] at reduceByKey at <console>:25 []
 +-(1) MapPartitionsRDD[[10] at map at <console>:23 []
    |  ParallelCollectionRDD[9] at parallelize at <console>:21 []
```

Stage II

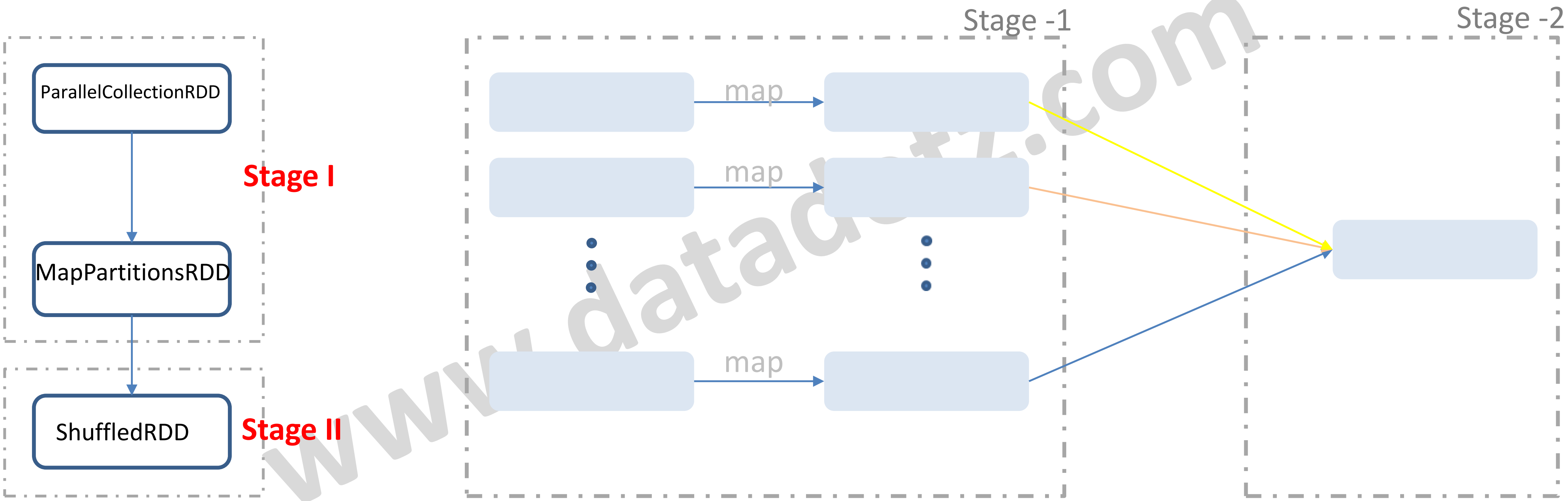
Stage I

DATA DOTZ

Typical Physical Plan



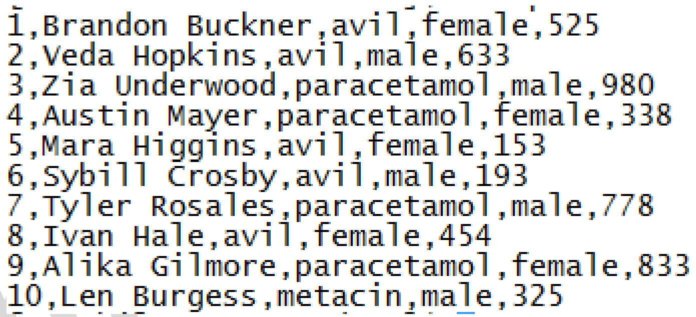
Typical Physical Plan



mappedKVs.reduceByKey(_+_,1)

Task = data Partition + Computation

Data From Local file



```
1,Brandon Buckner,avil,female,525
2,Veda Hopkins,avil,male,633
3,Zia Underwood,paracetamol,male,980
4,Austin Mayer,paracetamol,female,338
5,Mara Higgins,avil,female,153
6,Sybill Crosby,avil,male,193
7,Tyler Rosales,paracetamol,male,778
8,Ivan Hale,avil,female,454
9,Alika Gilmore,paracetamol,female,833
10,Len Burgess,metacin,male,325
```

Select drug, sum(amount) from patient group by drug;

```
scala> val patient = sc.textFile("/home/user/data/patient.txt")
patient: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[11] at textFile at <console>:21

scala> val mappedKVs = patient.map(line => (line.split(",")(2),line.split(",")(4).toInt))
mappedKVs: org.apache.spark.rdd.RDD[(String, Int)] = MapPartitionsRDD[13] at map at <console>:23

scala> val result = mappedKVs.reduceByKey(_+_, 1)
result: org.apache.spark.rdd.RDD[(String, Int)] = ShuffledRDD[14] at reduceByKey at <console>:25

scala> result.collect()
res2: Array[(String, Int)] = Array((avil,1958), (metacin,325), (paracetamol,2929))
```

Where Clause = filter in Spark

Select drug, sum(amount) from patient where drug == “avil” group by drug;

```
scala> val patient = sc.textFile("/home/user/data/patient.txt")
patient: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[11] at textFile at <console>:21

scala> val mappedKVs = patient.filter(_.split(",")(2) == "avil").map(line => (line.split(",")(2),line.split(",")(4).toInt))
mappedKVs: org.apache.spark.rdd.RDD[(String, Int)] = MapPartitionsRDD[13] at map at <console>:23

scala> val result = mappedKVs.reduceByKey(_+_, 1)
result: org.apache.spark.rdd.RDD[(String, Int)] = ShuffledRDD[14] at reduceByKey at <console>:25

scala> result.collect()
res2: Array[(String, Int)] = Array((avil,1958))
```

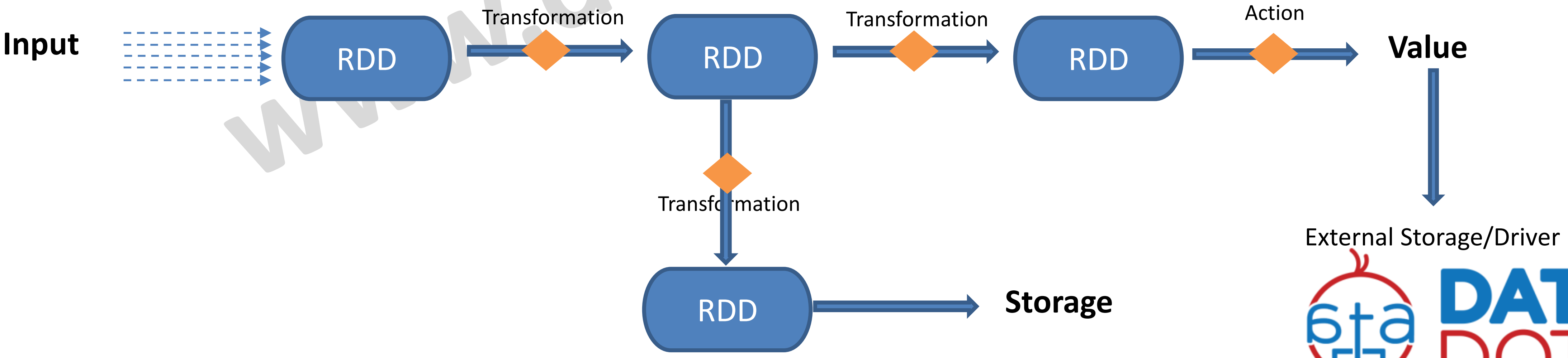
distinct

Select distinct(drug) from patient;

```
scala> val patient = sc.textFile("/home/user/data/patient.txt")
patient: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[11] at textFile at <console>:21

scala> val drug_distinct= patient.map(line => line.split(",")(2)).distinct
drug_distinct: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[11] at textFile at <console>:21

scala> drug_distinct.foreach(println)
avil
metacin
paracetamol
```

RDD Persistence /Caching

- Avoid re-evaluation of RDD, Spark provides many levels of Storing RDDs
  - MEMORY_ONLY, MEMORY_AND_DISK , DISK_ONLY
- To Persist/cache RDD, use below methods
  - persist()
  - cache() – use persist with MEMORY LEVEL
  - Both are lazy operations



Storage Level	Description	Format
MEMORY_ONLY (default)	*Recomputed if it does not fit in Memory*	
MEMORY_AND_DISK	*Spill to Disk on memory Full*	
MEMORY_ONLY_SER	*Recomputed if it does not fit in Memory*	*serialized*
MEMORY_AND_DISK_SER	*Spill to Disk on memory Full*	*serialized*
DISK_ONLY	*RDD Partitions in Disk*	
MEMORY_ONLY_2, MEMORY_AND_DISK_2		
OFF_HEAP (experimental)	*In* **Tachyon** *(distributed memory centric FileSystem)*	*serialized*

Persistence APIs

cache()

persist([Storage Level])

unpersist()

checkpoint()**

isCheckpointed()**

getCheckpointFile()**

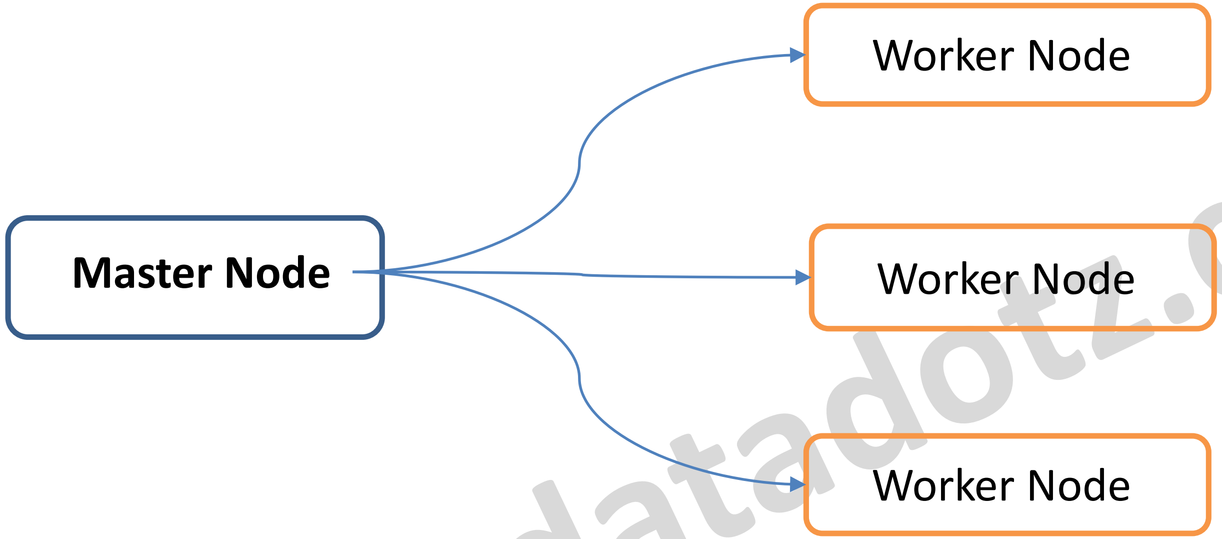
Removing Data from cache

- Removes old partitions of DataSet (RDDs) in LRU fashion per node basis Automatically - cache
- Remove RDDs manually by calling method
  - *RDD.unpersist()* method
  - Acts immediately

Shared Variables

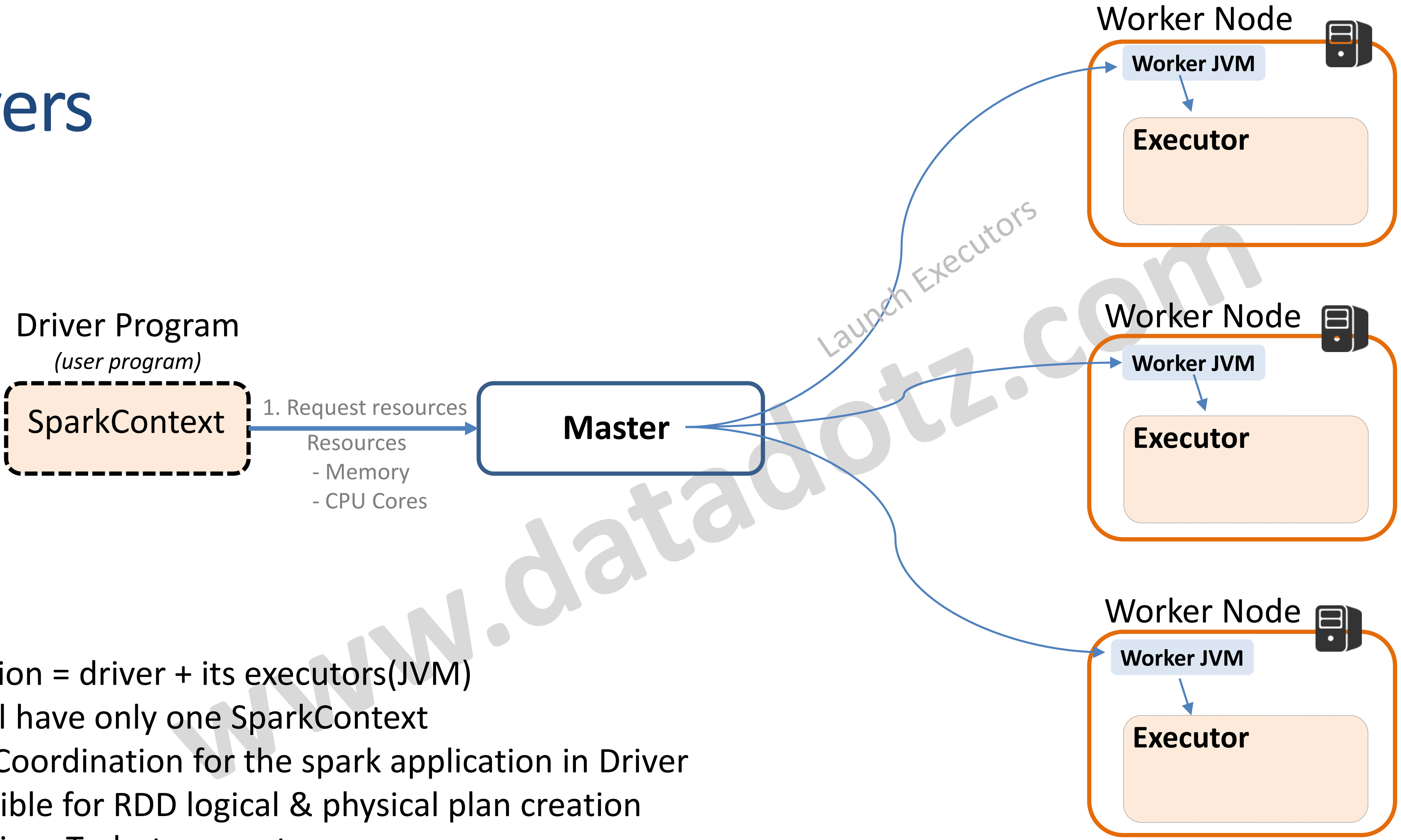
- Broadcast variables
  - To Share read only variables
- Accumulators
  - Aggregate information
  - Similar to Hadoop custom Counter
  - Accumulators can be seen in WebUI
  - Each task will have local accumulators
    - Spark will update each task's update to global accumulator only once if used in actions
    - If used in transformations, it may result in irregular values.

StandAlone Spark Cluster



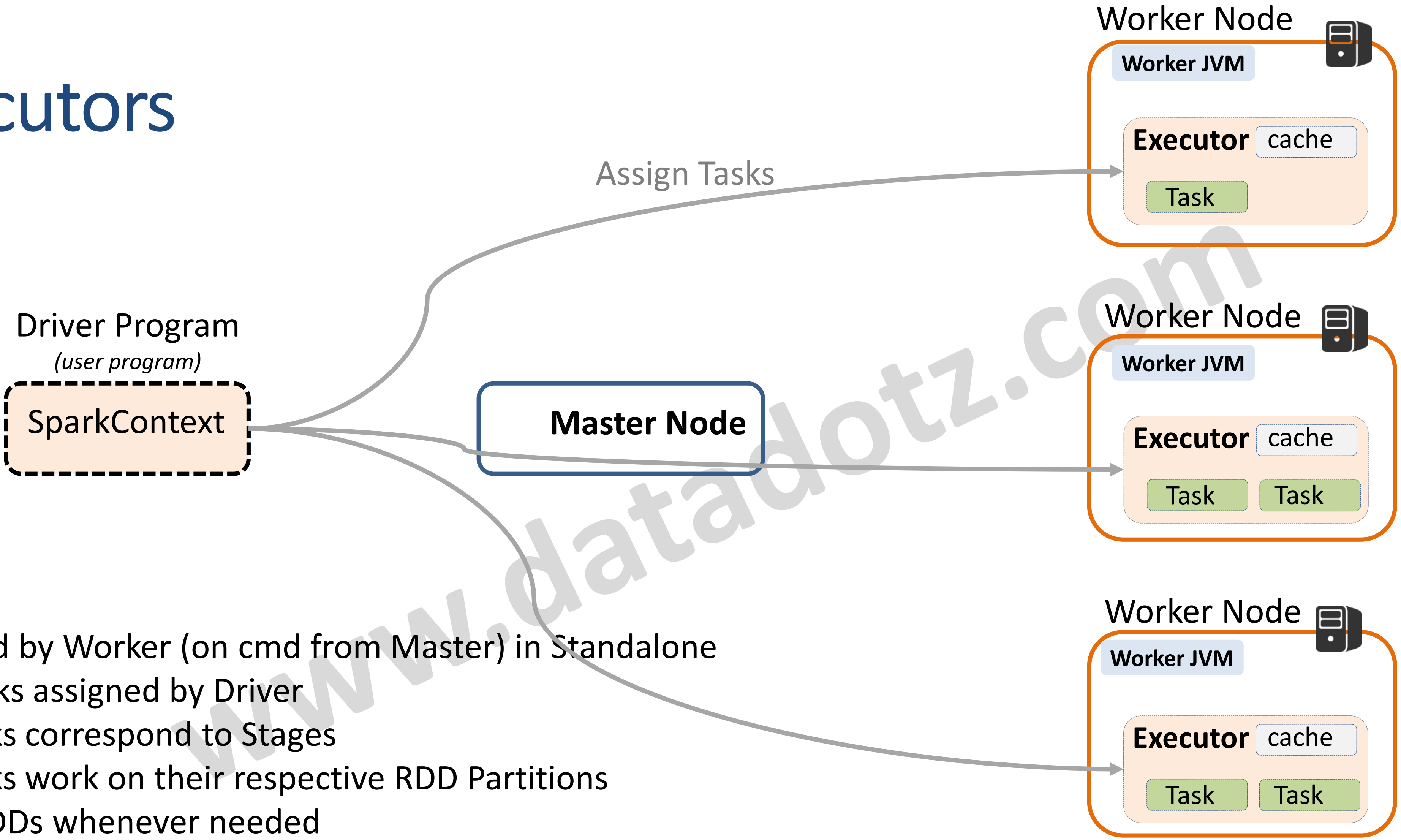
- Master / Slave Architecture
- Daemons(JVM)
    - Master
    - Worker

Drivers



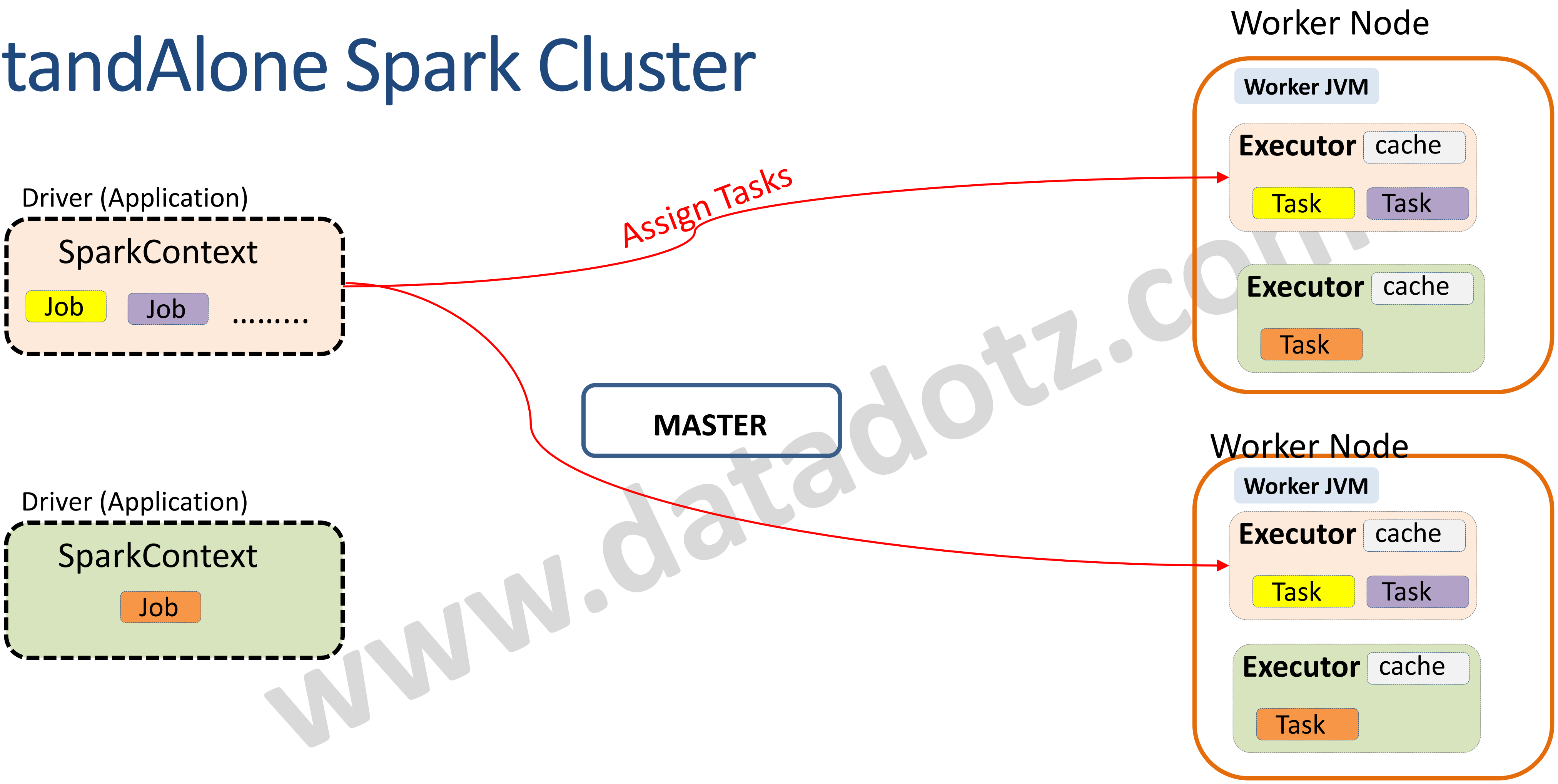
- Application = driver + its executors(JVM)
  - Will have only one SparkContext
- Central Coordination for the spark application in Driver
- Responsible for RDD logical & physical plan creation
  - Assigns Tasks to executors

Executors



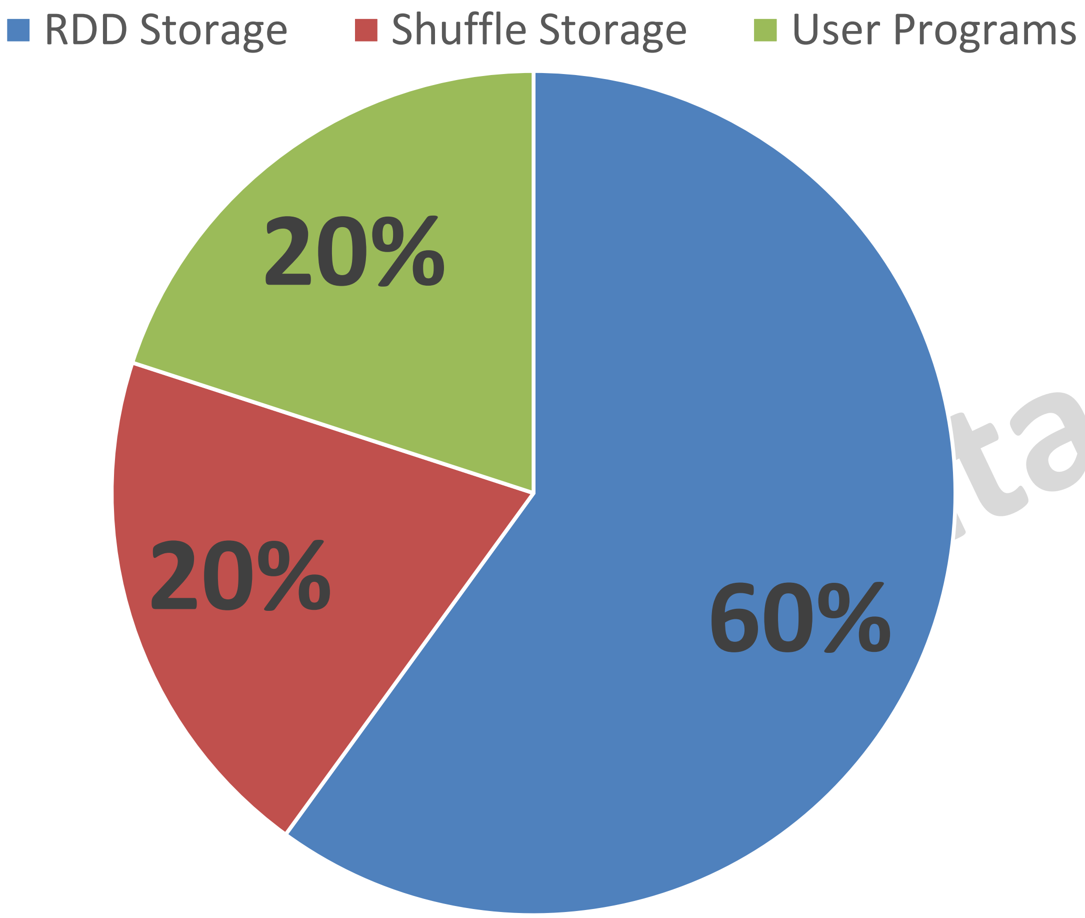
- Launched by Worker (on cmd from Master) in Standalone
- Runs Tasks assigned by Driver
  - Tasks correspond to Stages
  - Tasks work on their respective RDD Partitions
- Cache RDDs whenever needed

StandAlone Spark Cluster



Scheduling – Covered Later**

Memory Allocation in Executor



RDD Strorage:

- Use Spark.storage.memoryFraction to limit use JVM heap
- Used for .persist() or .cache()

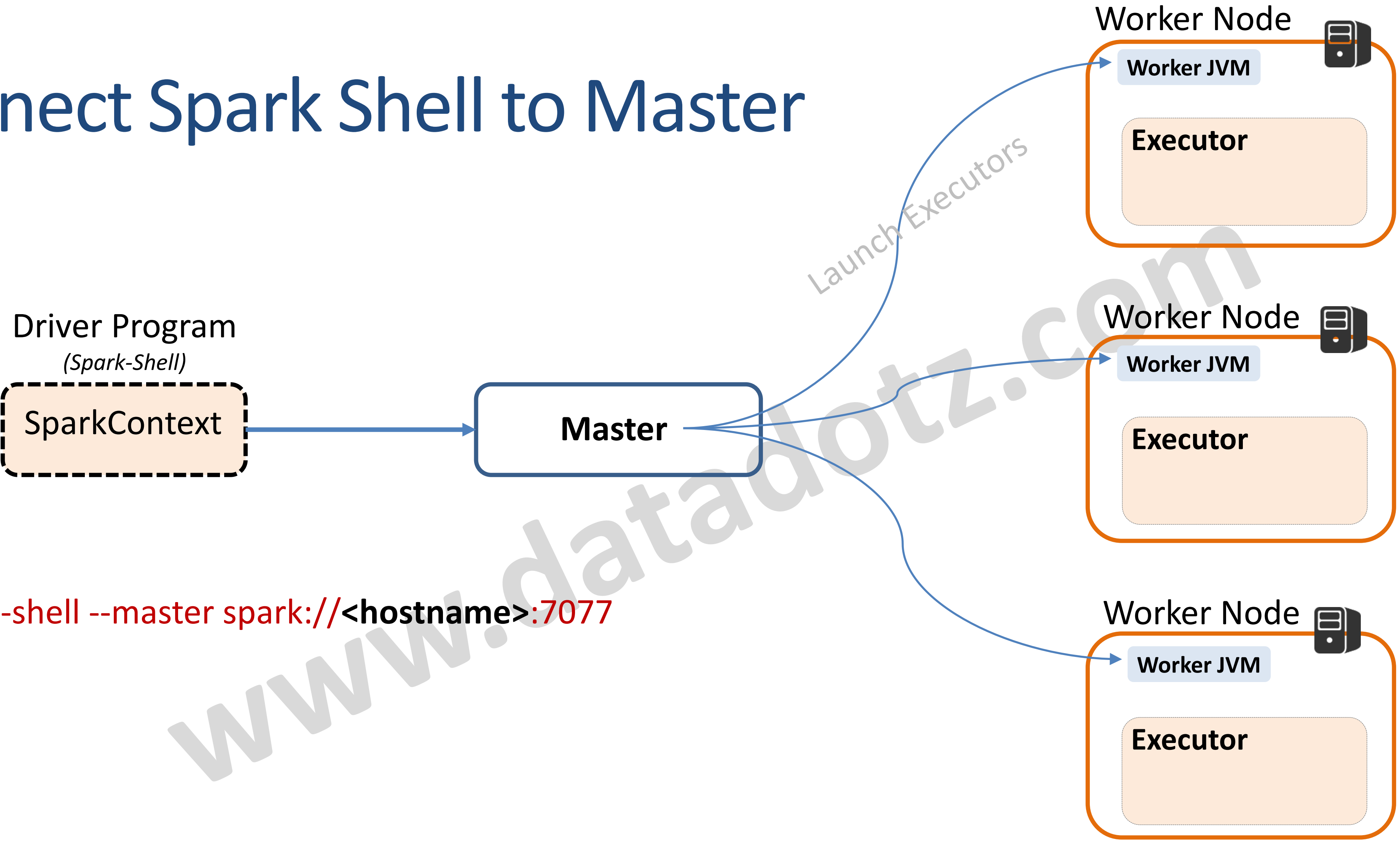
Shuffle Storage:

- Intermediate buffer for shuffle output data

User Programs:

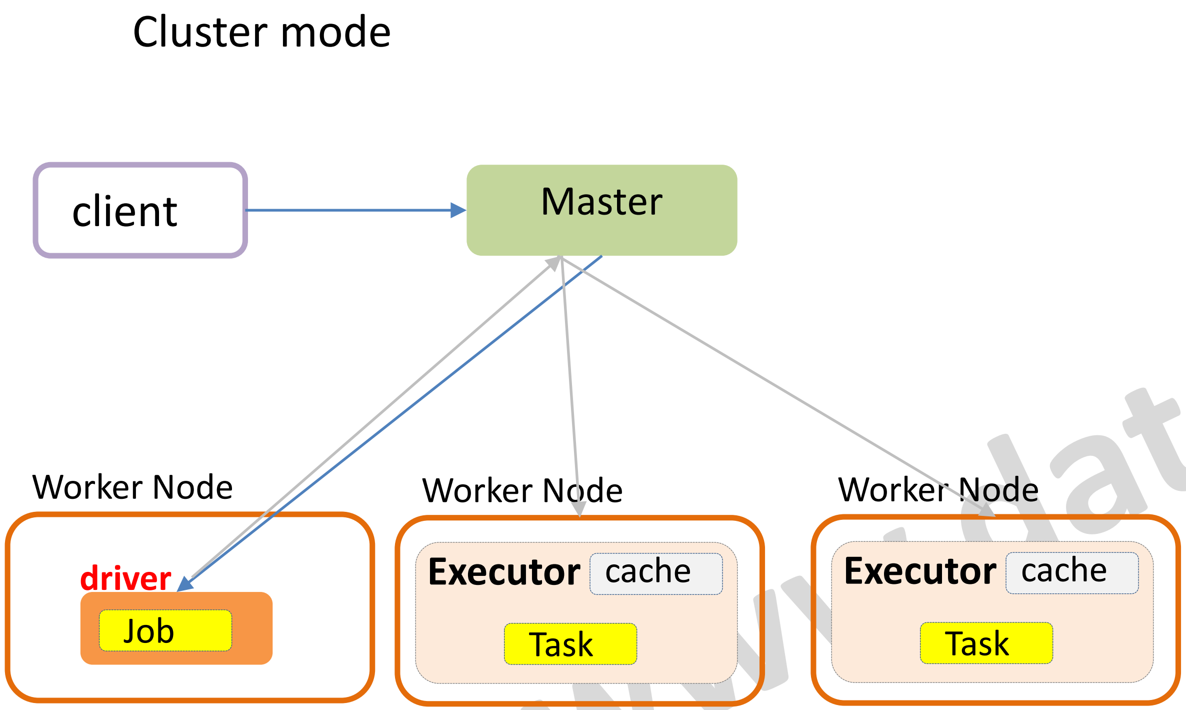
- JVM memory for executing user code.

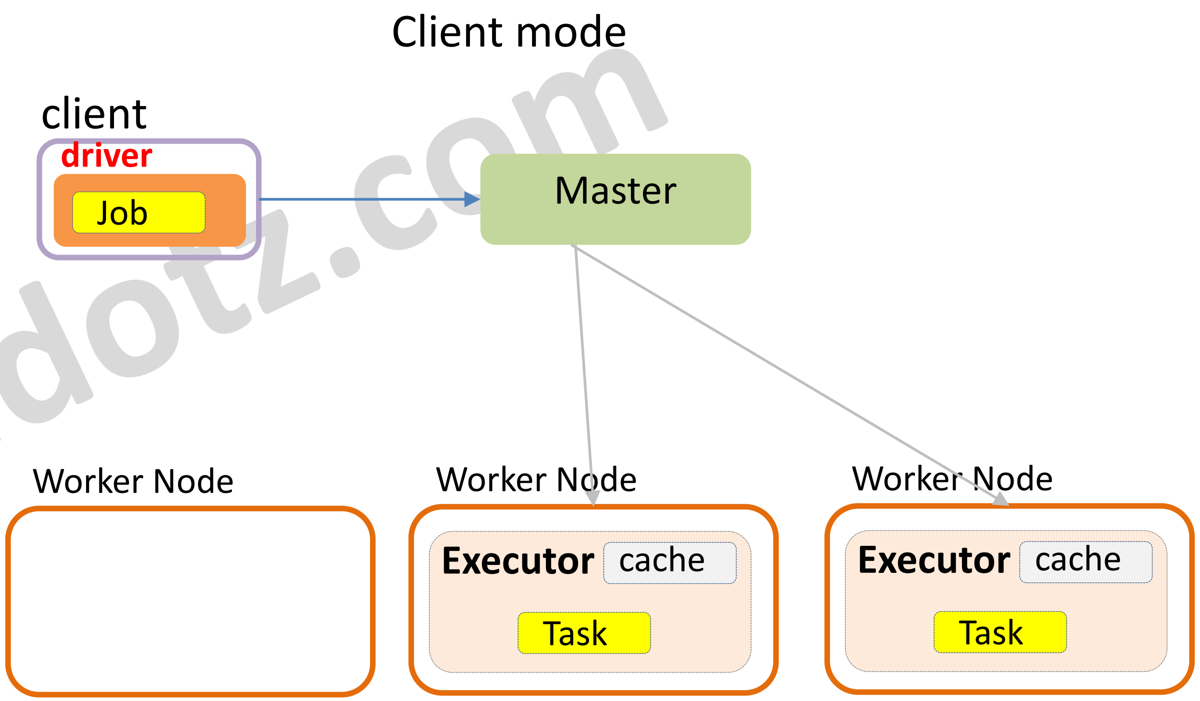
Connect Spark Shell to Master



```
bin/spark-shell --master spark://<hostname>:7077
```

Deploy-mode





Fire & Forget

Spark Submit

bin/spark-submit [options] <app jar> <app jar arguments>

Common Options

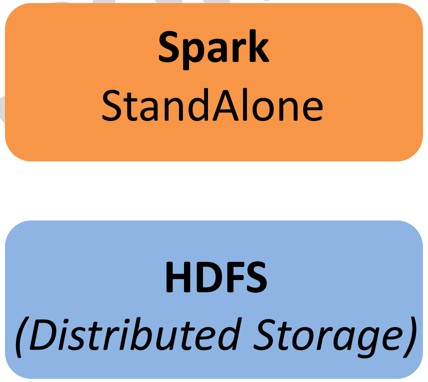
--master
--class
--deploy-mode
--name
--jars
--conf
--properties-file
--executor-memory
--total-executor-cores

Types of Application

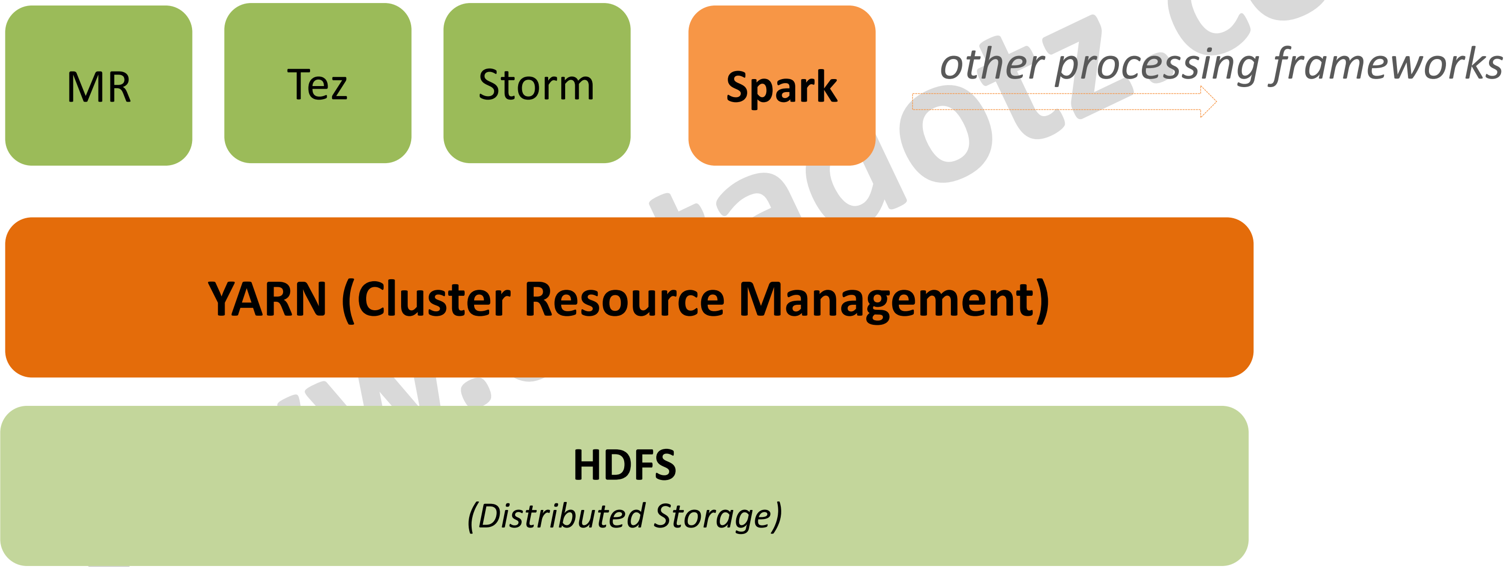
- Long Lived / Shared Application
  - SparkSQL JDBC Applications - ThriftServer
  - Spark Streaming
- Short Lived Applications
  - StandAlone Applications
  - Interactive Shell Sessions

Spark and Hadoop

- Supports below inputs/outputs from Hadoop
  - Text Files
  - Sequence Files
  - Avro
  - Parquet
  - Other InputFormat



Hadoop 2. X


MR
Tez
Storm
Spark
other processing frameworks
YARN (Cluster Resource Management)
HDFS
(Distributed Storage)

YARN - Yet Another Resource Manager
MR - MapReduce

Summary of Cluster Managers

	Spark StandAlone (client)	Spark StandAlone (Cluster)	Yarn Client	YARN cluster
Driver runs in	Client	As JVM in Worker	Client	Application Master
Resources requested by	Client	Driver JVM in Worker	Application Master	Application Master
Executors started by	Worker	Worker	Node Manager	Node Manager
Daemons	Master & Slave	Master & Slave	ResourceManager Node Manager	ResourceManager NodeManager
Spark Shell	Yes	No	Yes	No

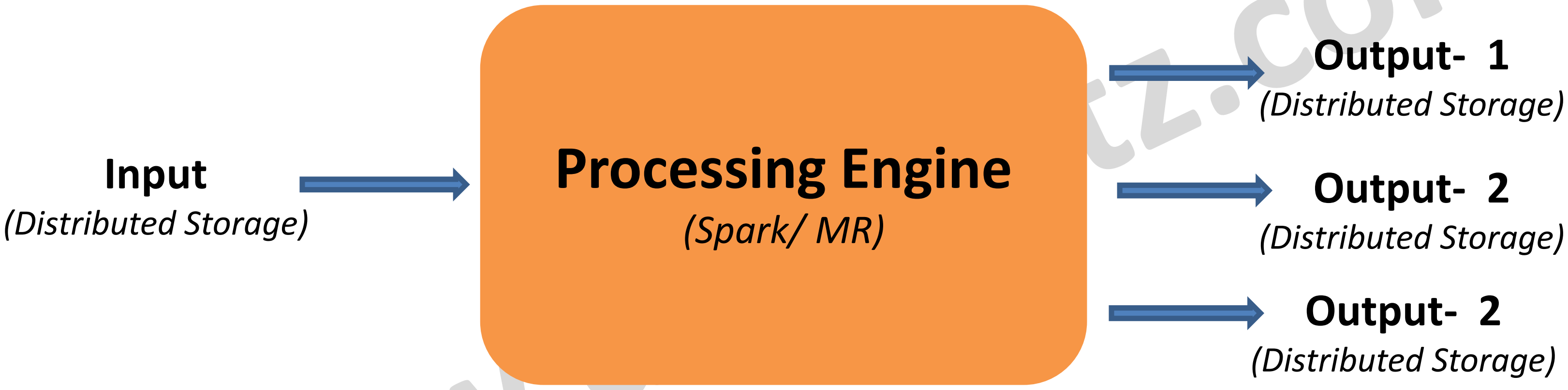
Hadoop Vendor Support

MAPR®

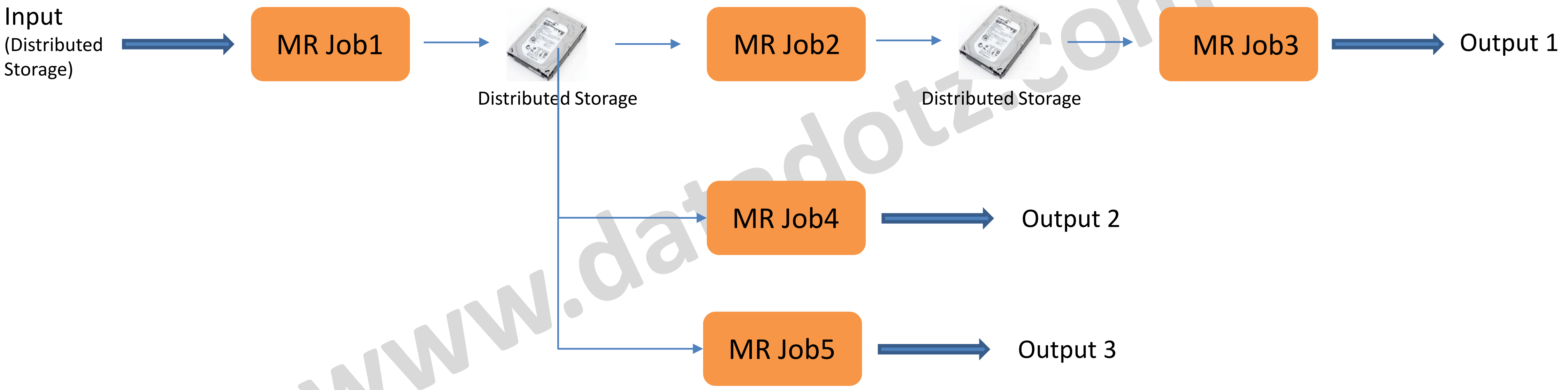
Hortonworks

cloudera®

Lets Understand with an Example



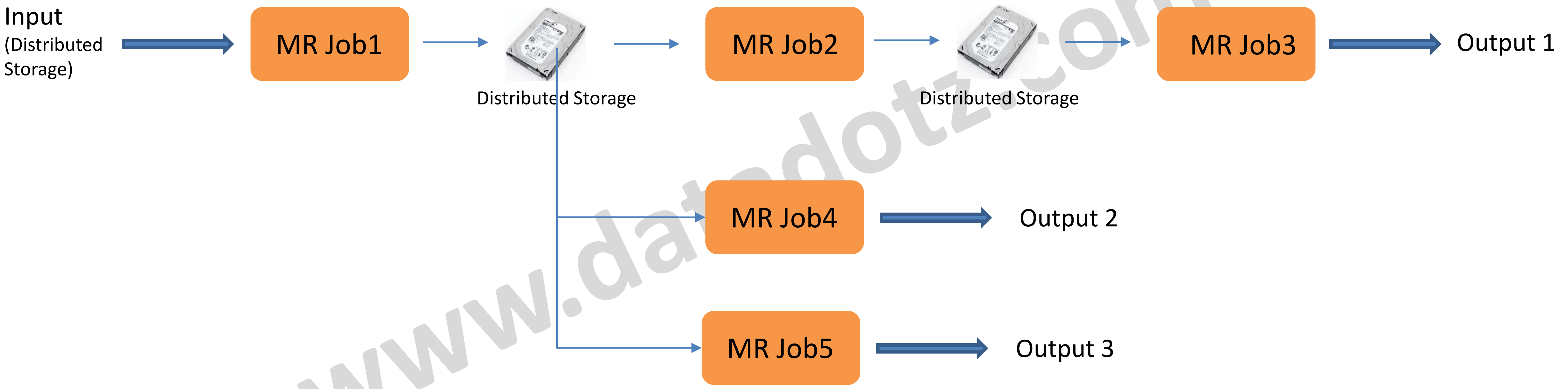
MapReduce



- Inefficient for Data Reuse in Applications
  - Iterative / MultiStep Applications
  - Interactive Applications

Reference: http://www.cs.berkeley.edu/~matei/papers/2012/nsdi_spark.pdf - Research from Matei Zaharia and Others

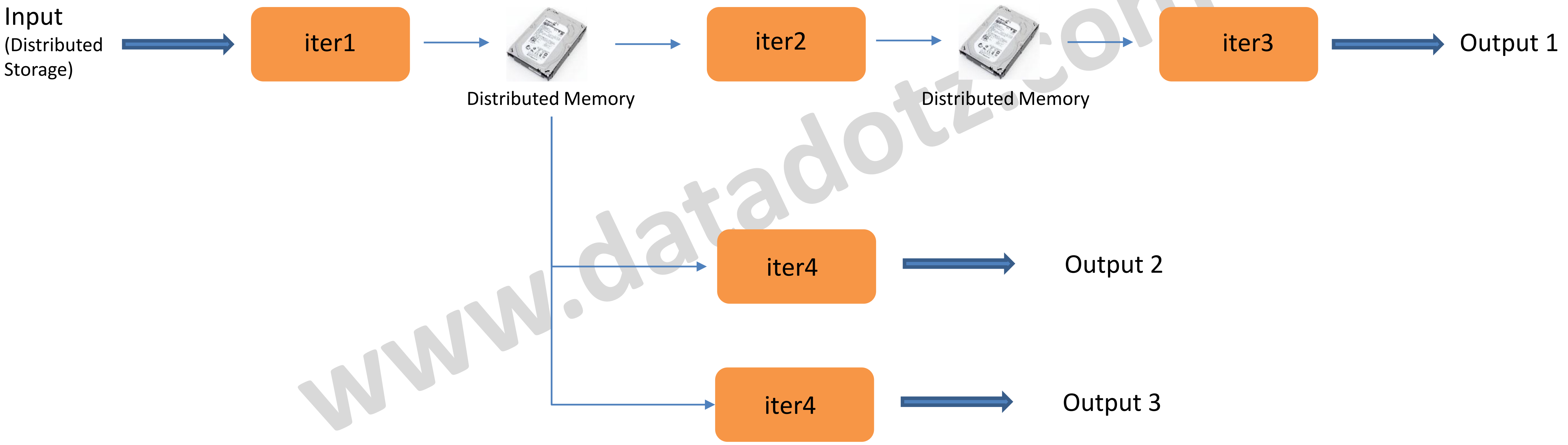
MapReduce



- Inefficient for Data Reuse in Applications
  - Iterative / MultiStep Applications
  - Interactive Applications

Reference: http://www.cs.berkeley.edu/~matei/papers/2012/nsdi_spark.pdf - Research from Matei Zaharia and Others

Spark



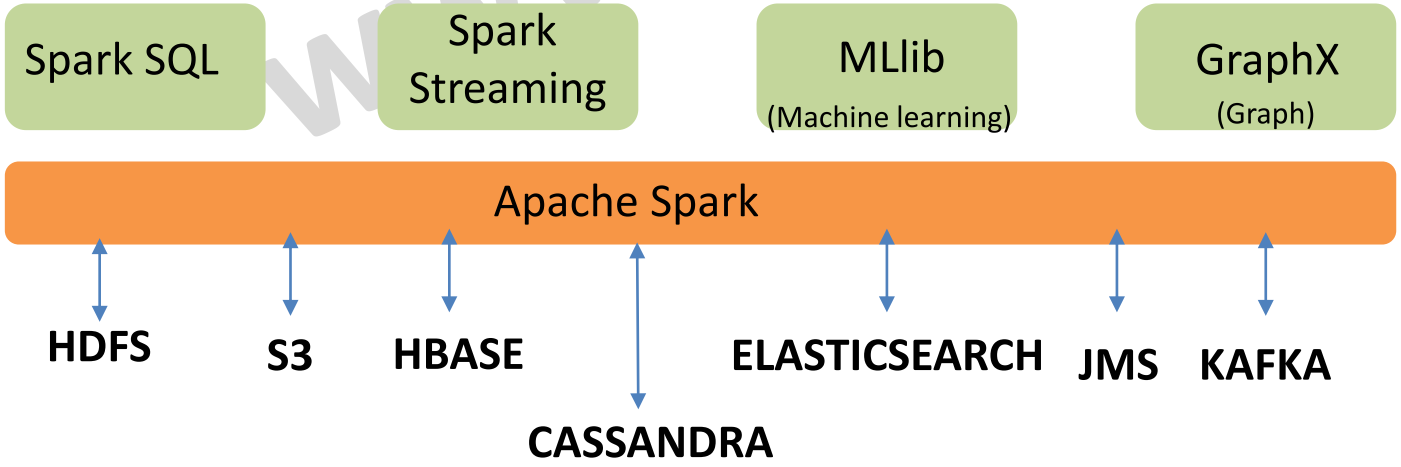
Reference: http://www.cs.berkeley.edu/~matei/papers/2012/nsdi_spark.pdf - Research from Matei Zaharia and Others

Spark is faster

- Caching – Intermediate Data - In Memory
  - Iterative or MultiStep Algorithms
  - Workflows are faster due to caching
- Shuffling
  - Low cost
  - Hash, Sort, Tungsten-Sort
- Startup time
  - Tasks are Threads in Spark whereas Tasks are JVMs in MR

Other factors

- Multiple operators
  - MR – Map, Reduce
  - Spark – map, reduce, join, filter, cogroup, sort..etc
  - **Ease of Programming – RAD(Rapid Application Development)**
- Execution Engines
  - MR – Batch
  - Spark – Batch, streaming, Interactive



ThAnK yOu