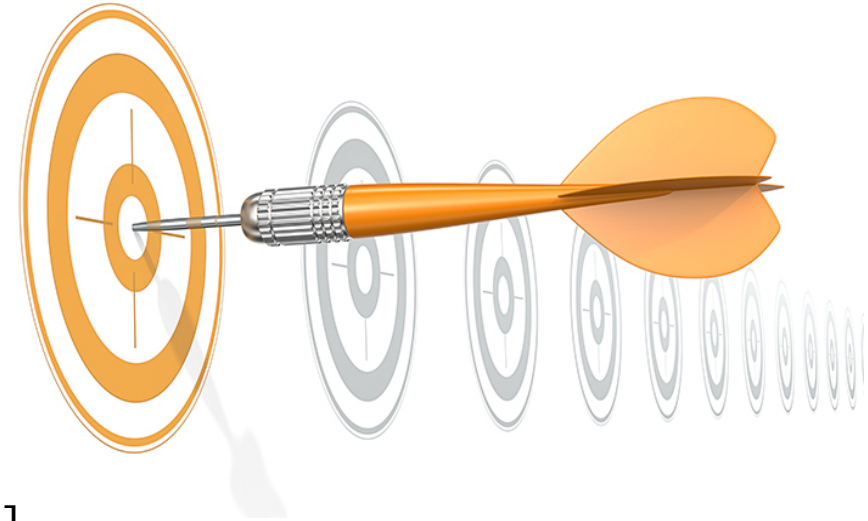
Agenda

- what is Spark – The Beginning
- Replacement to Hadoop ?
- Deploying Spark
- Basic Spark Application Terminology
- Spark Architecture [Master/Slave]
- Spark Application Execution Model
- Data Computation Model.
- Tale-1 RDD.
- Tale-2 SparkSQL [Datasets & Data frame]
- Spark Environment Overview in Practical
- Broadcast Variables & Accumlators
- Real Time Use case of SparkSQL & RDD Demo using Zeppelin



[Layman Understanding]
Spark & its Implementation Languages

What is this ?

It’s a table

we can build it using wood

we can build it using Plastic

we can build it using iron

Spark

we can implement it using Scala

we can implement it using Python

we can implement it using Java

we can implement it using R

Apache Spark
The Beginning

- Apache Spark is an

  "open-source distributed general-purpose cluster computer framework with mostly in memory data processing engine."

- Spark deals with 2 types of data basically
  a) Data at rest (Batch Processing)
  b) Data in Motion (Stream (Continuous) Processing).

- Spark provides rich concise high level API's for programming languages like **Scala,** Python, Java, R and SQL.

* Microsoft's Mobius project will provide C# API for Spark in support of .NET

Apache Spark
The Beginning [Replacement to Hadoop ?]

- Point to Remember Strongly:
  "Spark is intended to *enhance*, not replace, the Hadoop stack."

- Spark is a processing engine that functions on top of the Hadoop ecosystem. Both Hadoop and Spark have their own advantages.

- Hadoop = HDFS (storage) + MapReduce (Processing)

- Big Data Platform = HDFS (storage) + Spark (or) MapReduce (Processing)

- Spark is built to increase the processing speed of the Hadoop ecosystem and to overcome the limitations of MapReduce.

Apache Spark
Deploying Spark

How many ways are there at present to deploy Spark ?

Spark can be run in distributed mode on a cluster.

Currently supported open-source cluster-managers/task scheduler/resource manager are

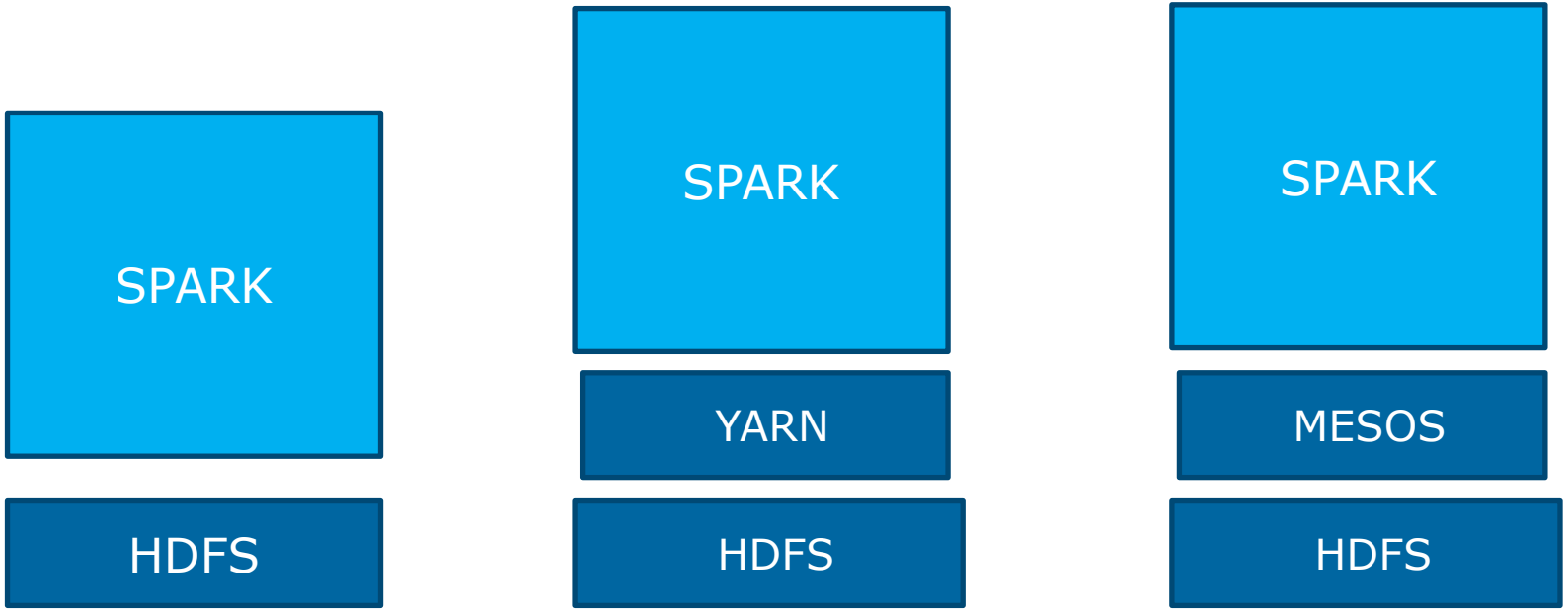
a) Spark Standalone [Built in spark's cluster manager].

b) Hadoop YARN [Yet Another Resource Negotiator].

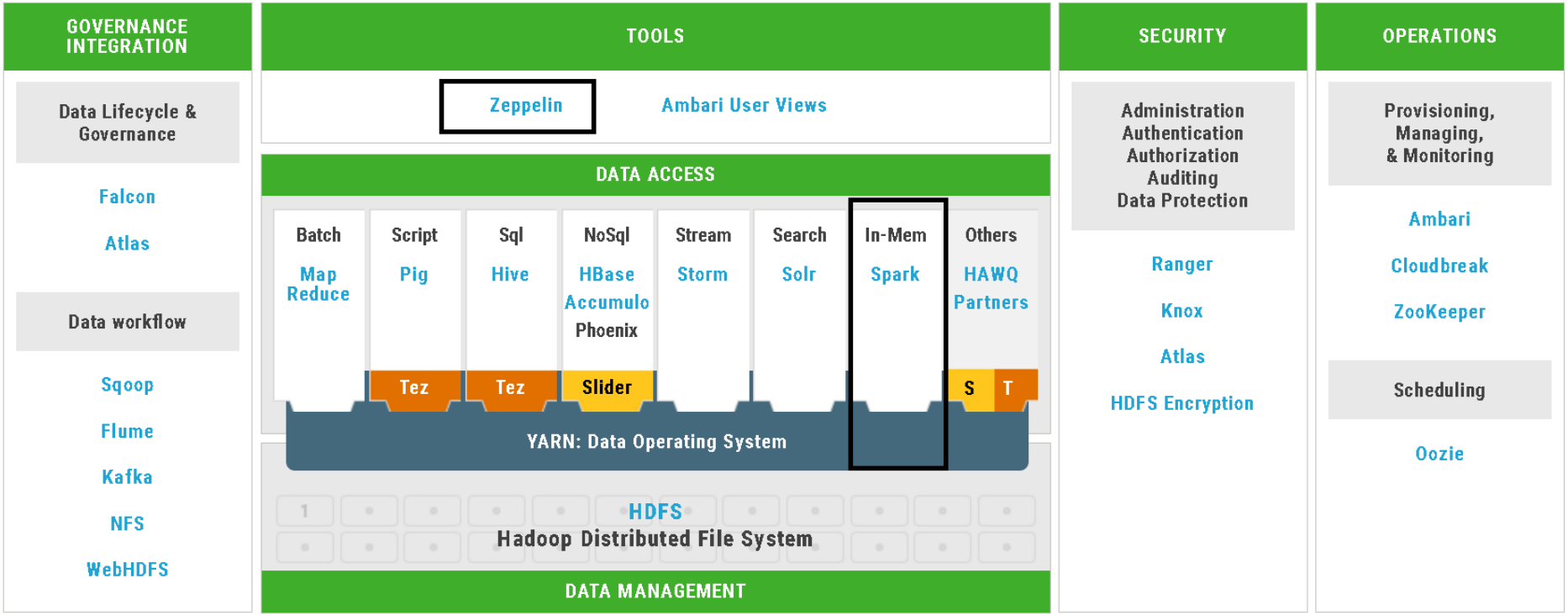
c) Apache Mesos [Cluster manager] [Distributed Kernel]

- Running Spark on a cluster requires workload and resource management on distributed systems.

Apache Spark
Deploying Spark



Spark
In Big Data Ecosystem



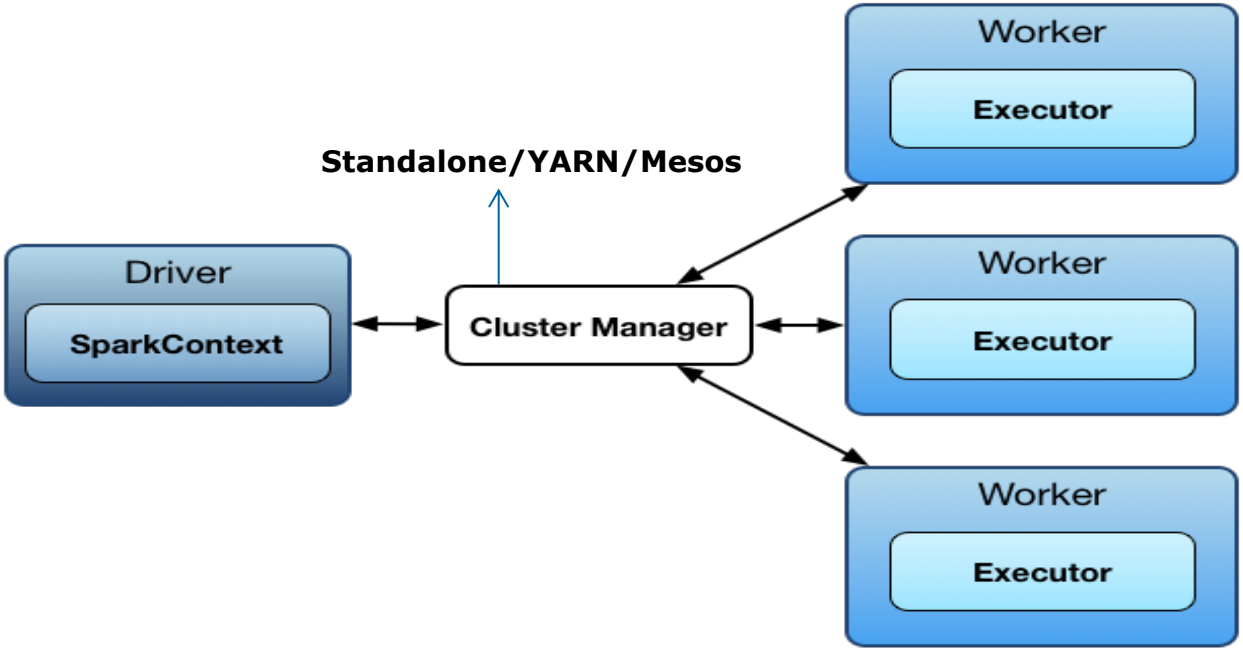
Spark Application
Basic Terminology & Key Terms

- Spark Context : it is the heart of a Spark application. Spark context sets up internal services and establishes a connection to a Spark execution environment.

- Spark Driver: A Spark driver (an application's driver process) is a JVM process that hosts Spark Context for a Spark application. It is the master node in a Spark application. Driver hosts web UI for spark application. A driver coordinates workers and overall execution of tasks.

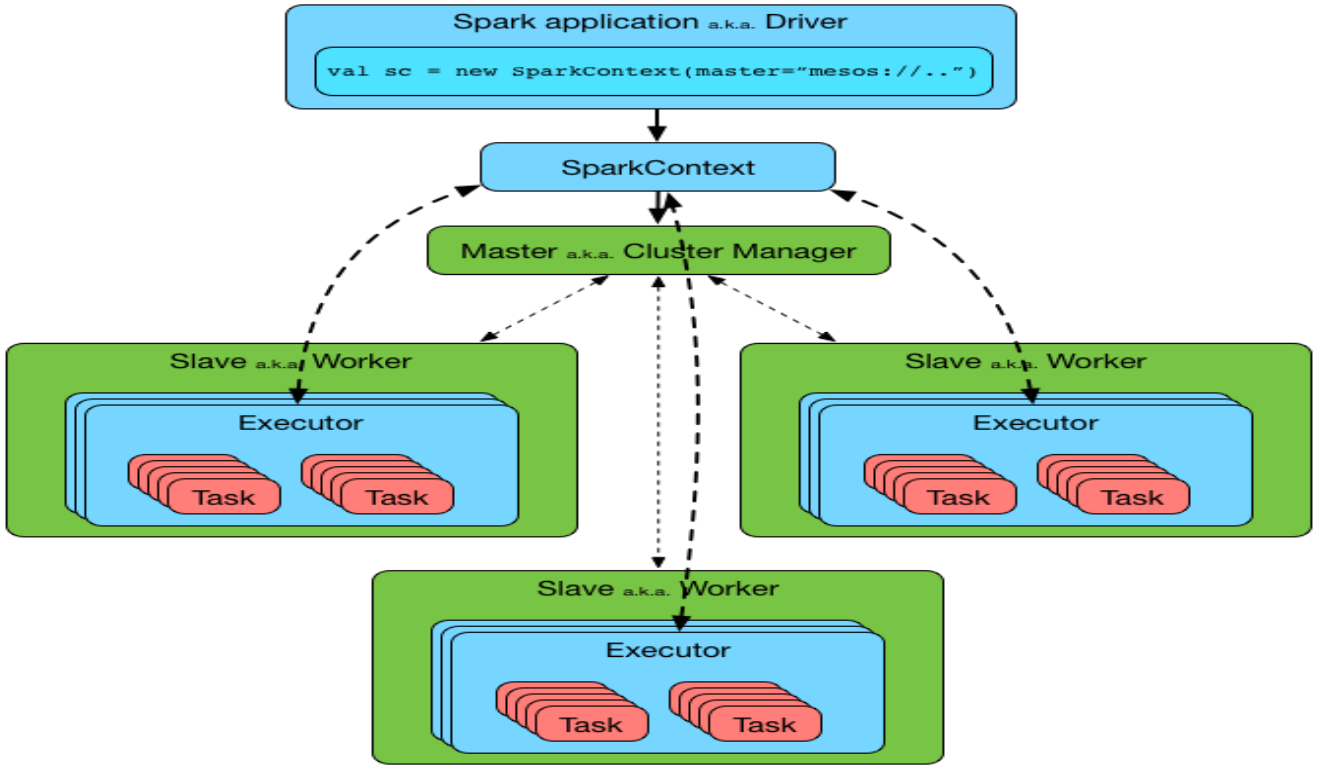
- Executor: it is a distributed agent that is responsible for executing tasks.

Spark Architecture
Master-Slave

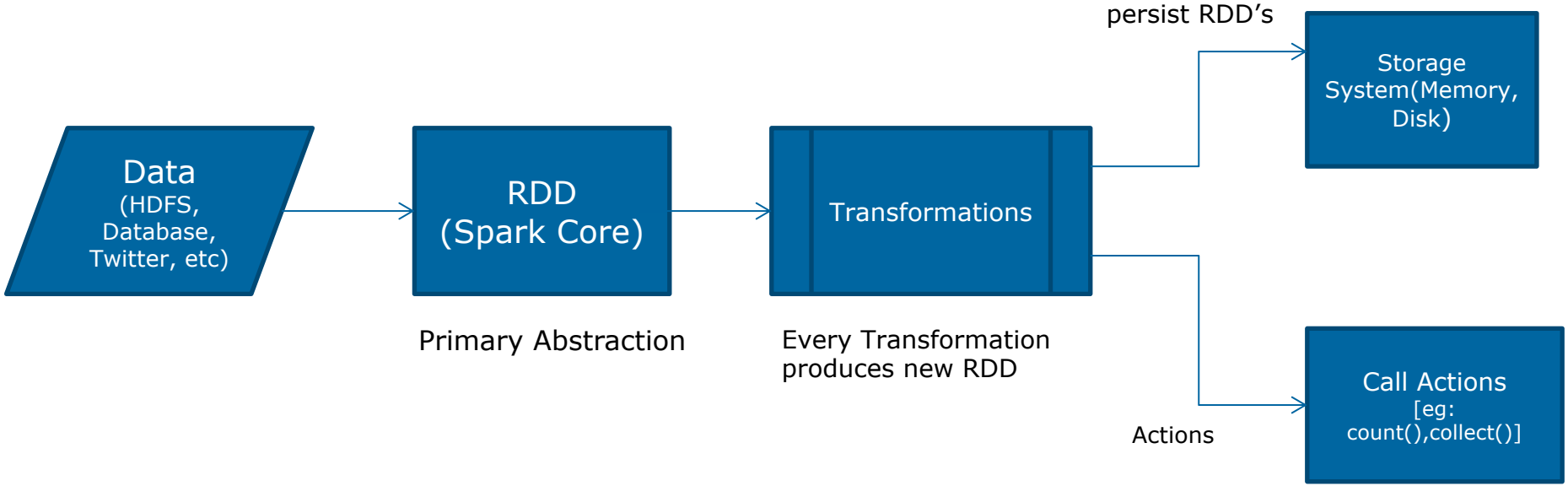


There is a driver that talks to a single coordinator called master that manages workers in which executors run.

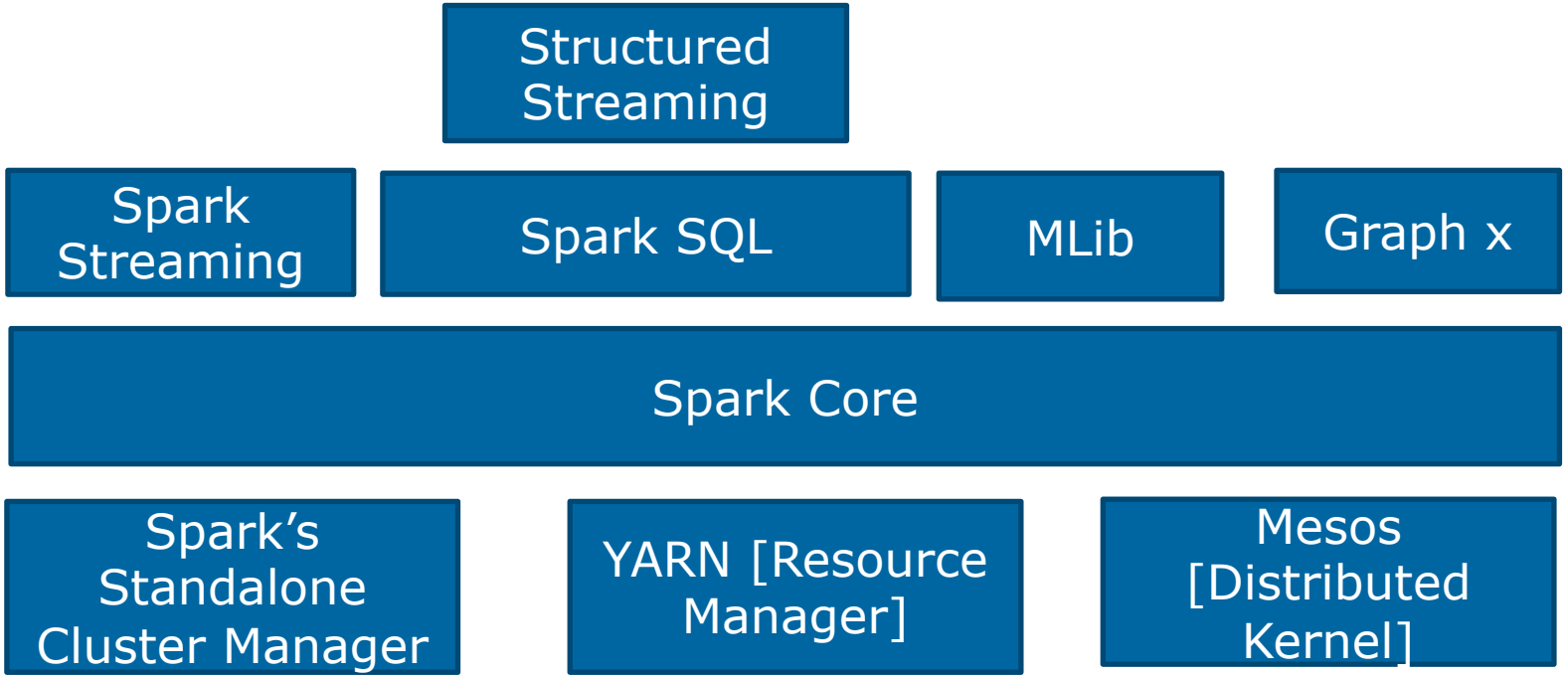
Application Execution Model
Standalone mode & YARN mode



Spark's
Data Computation Model



Spark
Full Overview [Eco System]



Tale-1
RDD [Primary Abstraction in Spark]

- RDD-Resilient Distributed Dataset

- At the core, an RDD is an immutable distributed collection of elements of your data, partitioned across nodes in your cluster that can be operated in parallel with a low-level API that offers *transformations* and *actions*.

- To say it in a simple way

  A RDD is a resilient and distributed collection of records spread over one or more partitions.

Tale-1
Benefit of Using RDD

- Using RDD Spark hides data partitioning and so distribution that in turn allowed them to design parallel computational framework with a higher-level programming interface (API) .

- **Resilient**, i.e. fault-tolerant with the help of RDD lineage Graph and so able to re-compute missing or damaged partitions due to node failures.

- **Distributed** with data residing on multiple nodes in a Cluster.

- **Dataset** is a collection of partitioned data with primitive values or values of values, e.g. tuples or other objects (that represent records of the data you work with).

Tale-1
Behavior of RDD

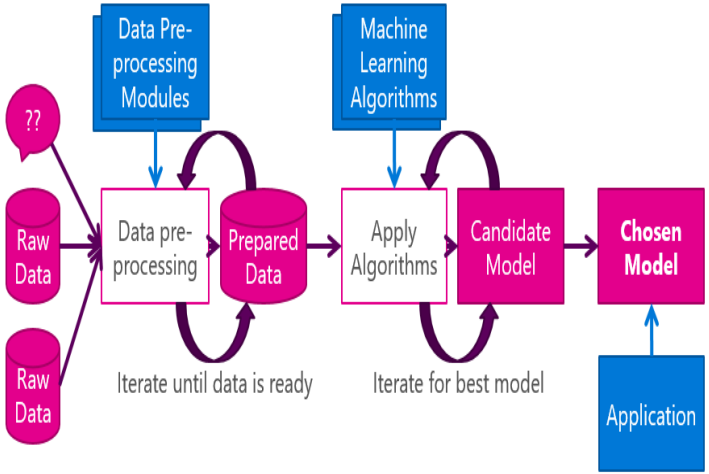
Functionality	Description
In Memory	data inside RDD is stored in memory as much (size) and long (time) as possible.
Immutable	it does not change once created and can only be transformed using transformations to new RDDs.
Lazy Evaluated	the data inside RDD is not available or transformed until an action is executed that triggers the execution
Cacheable	you can hold all the data in a persistent "storage" like memory (default and the most preferred) or disk (the least preferred due to access speed).
Parallel	process data in parallel
Typed	RDD records have types, e.g. Long in RDD[Long] or (Int, String) in RDD[(Int, String)].
Partitioned	records are partitioned (split into logical partitions) and distributed across nodes in a cluster.
Location Stickiness	RDD can define placement preferences to compute partitions (as close to the records as possible).

Tale-1
RDD Motivation

- The motivation to create RDD were two types of applications that current computing frameworks handle inefficiently:

  a) iterative algorithms in machine learning and graph computations.

  b) interactive data mining tools as ad-hoc queries on the same dataset.



- The goal is to reuse intermediate in-memory results across multiple data-intensive workloads with no need for copying large amounts of data over the network.

Tale-1
When to use RDD in real scenarios [Use cases]

- you want low-level transformation and actions and control on your dataset.
- your data is unstructured, such as media streams or streams of text.
- you want to manipulate your data with functional programming constructs than domain specific expressions.
- you don’t care about imposing a schema, such as columnar format, while processing or accessing data attributes by name or column.
- you can forgo some optimization and performance benefits available with Data Frames and Datasets for structured and semi-structured data.

Tale-1
RDD Operations

Transformations - lazy operations that return another RDD.

- apply user function to every element in a partition.
- apply aggregation function to the whole dataset (groupBy, sortBy).
- introduce dependencies between RDDs to form DAG.
- provide functionality for repartitioning (repartition, partitionBy).

actions - operations that trigger computation and return values.

- trigger job execution
- used to materialize computation results

- Persistence -explicitly store RDDs in memory, on disk or off-heap (cache, persist).
  - checkpointing for truncating RDD lineage

Tale-1
How to create an RDD?

- One can create RDD's in 3 different languages like Scala, Python and Java.

- One way to create a RDD is with SparkContext.parallelize method. It accepts a collection of elements as shown below:

```
scala> val rdd = sc.parallelize(1 to 1000);
```

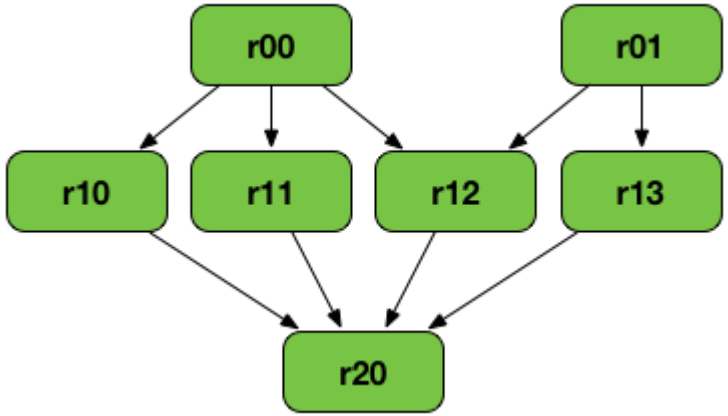
o/p: rdd: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[0] at parallelize at <console>:25

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

Tale-1
RDD Lineage

- RDD Lineage is a Logical Execution Plan.
- Also known as *RDD operator graph* or *RDD dependency graph.*
- It is a graph of all the parent RDDs of a RDD. It is built as a result of applying transformations to the RDD and creates a logical execution plan.

```
val r00 = sc.parallelize(0 to 9)
val r01 = sc.parallelize(0 to 90 by 10)
val r10 = r00 cartesian r01
val r11 = r00.map(n => (n, n))
val r12 = r00 zip r01
val r13 = r01.keyBy(_ / 20)
val r20 = Seq(r11, r12, r13).foldLeft(r10)(_ union _)
```



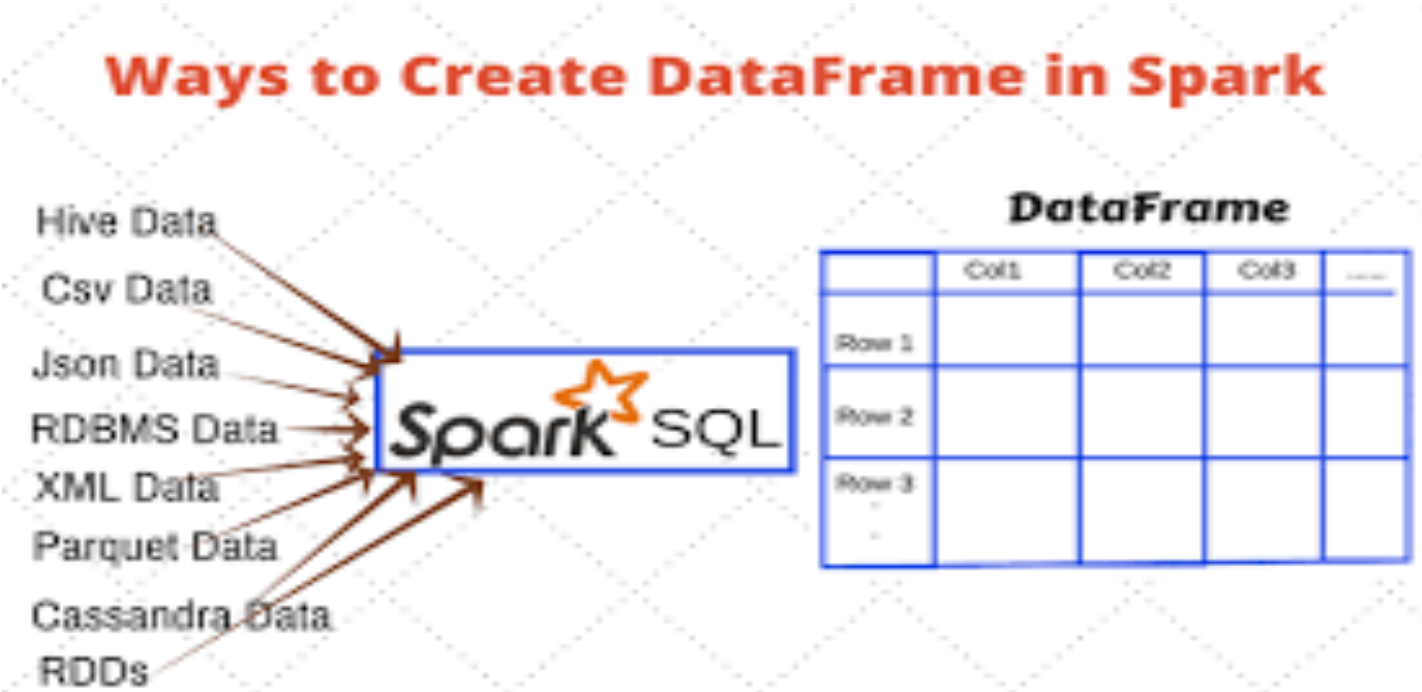
Tale-2
DataFrame & Dataset

- DataFrame is an immutable distributed collection of data. Unlike an RDD, data is organized into named columns, like a table in a relational database.

- Designed to make large data sets processing even easier, DataFrame allows developers to impose a structure onto a distributed collection of data, allowing higher-level abstraction.

- It provides a domain specific language API to manipulate your distributed data, and makes Spark accessible to a wider audience, beyond specialized data engineers.

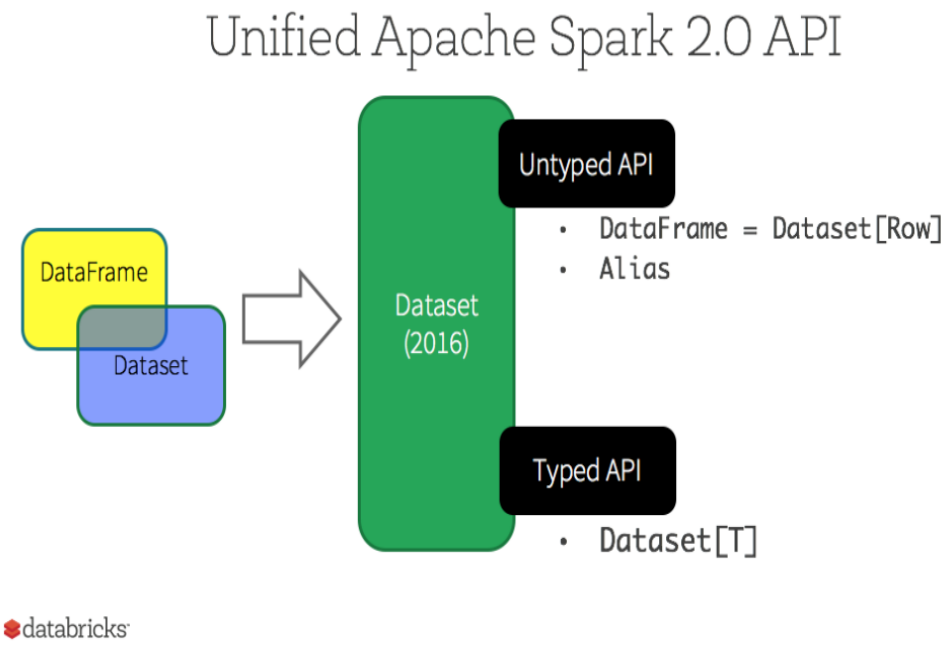
Data frame visualization



Tale-2
DataFrame → Dataset

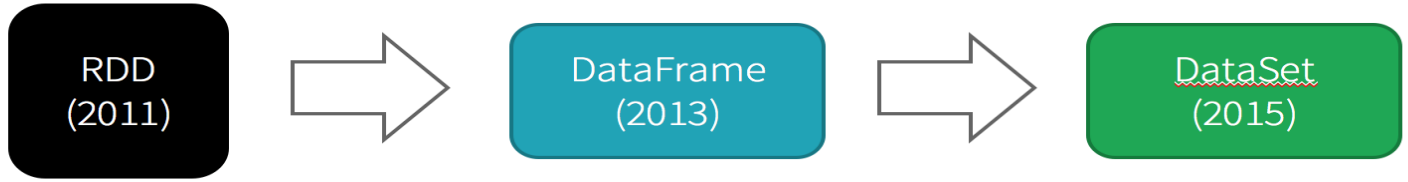
- In Spark 2.0, DataFrame APIs will merge with Datasets APIs, unifying data processing capabilities across libraries.

- Because of this unification, there will be single high-level and type-safe API called Dataset.



Spark API's

History of Spark APIs



Distribute collection of JVM objects

Functional Operators (map, filter, etc.)

Distribute collection of Row objects

Expression-based operations and UDFs

Logical plans and optimizer

Fast/efficient internal representations

Internally rows, externally JVM objects

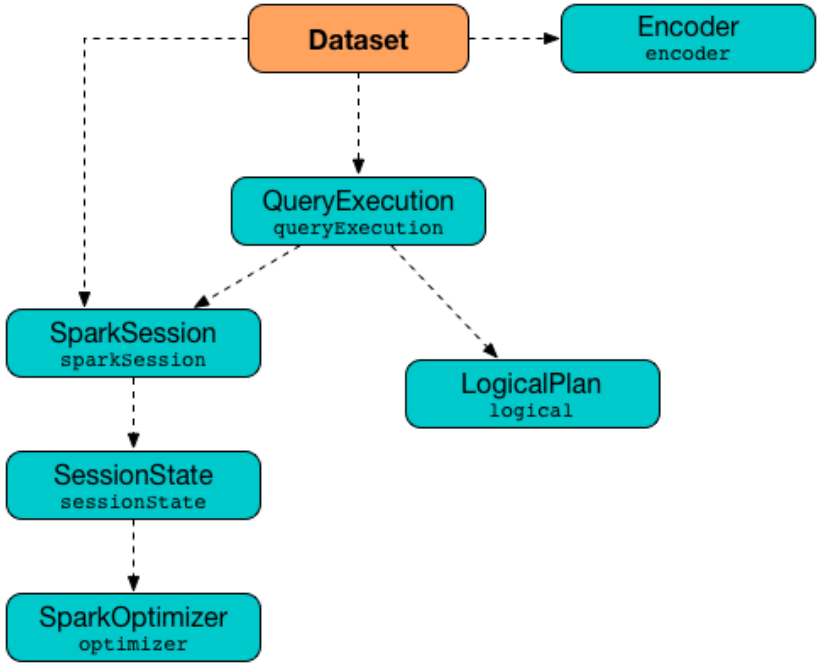
Almost the "Best of both worlds": **type safe + fast**

But slower than DF
Not as good for interactive analysis, especially Python

databricks

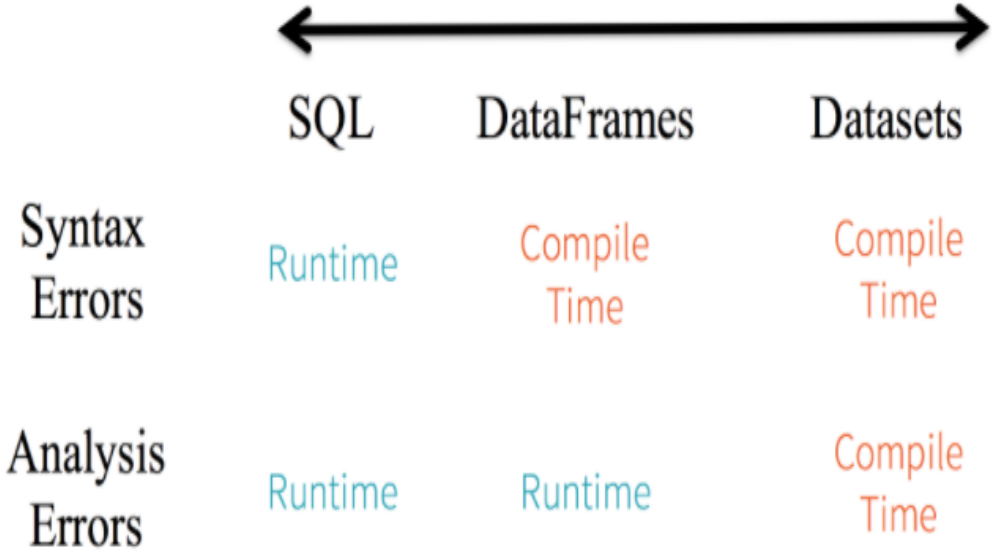
Tale-2
Dataset

- **Dataset** is a strongly-typed data structure in Spark SQL that represents a structured query with encoders.
- Datasets are *lazy* and structured query expressions are only triggered when an action is invoked.
- Internally, a Dataset represents a logical plan that describes the computation query required to produce the data.

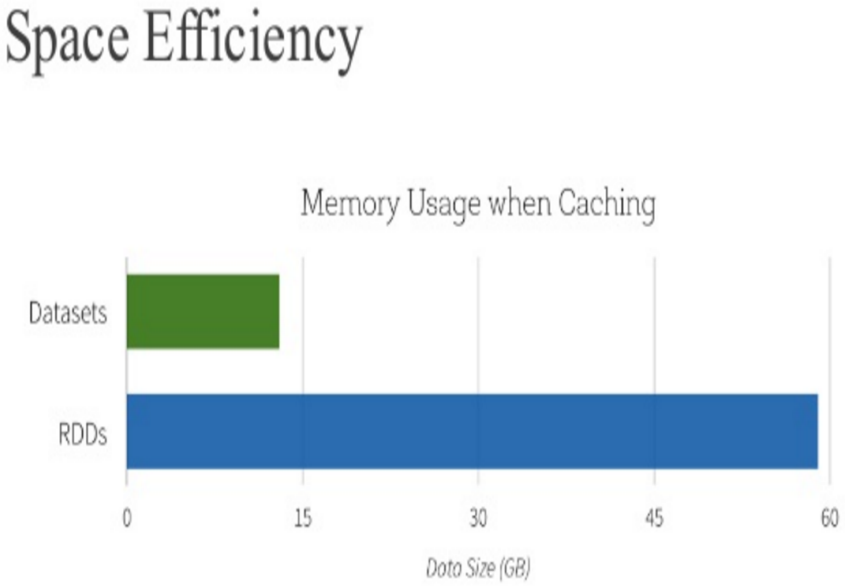


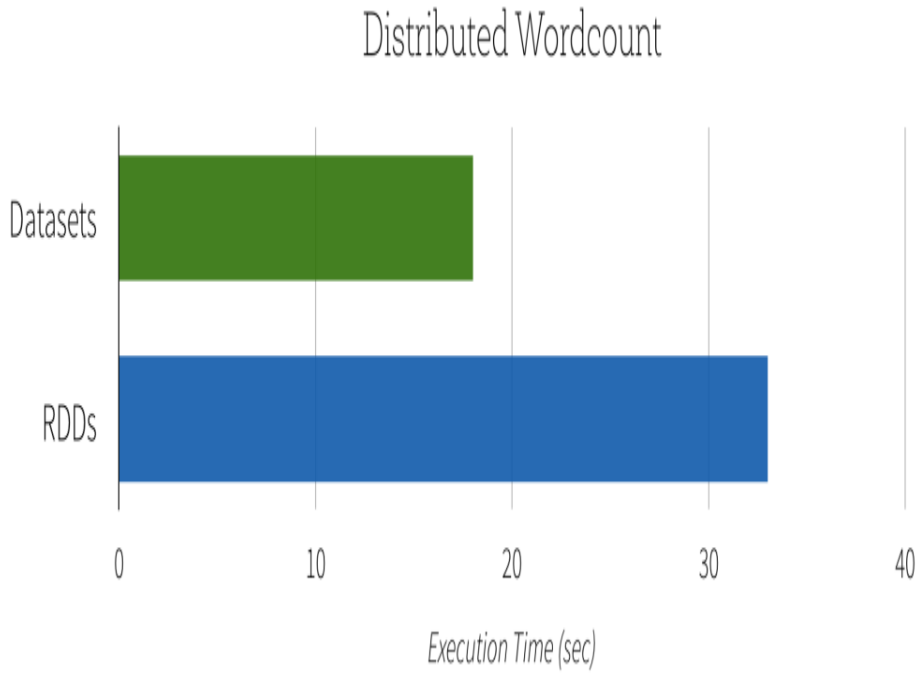
Tale-2
Benefit of Dataset API

- Static-typing and runtime type-safety
- High-level abstraction and custom view into structured and semi-structured data
- Ease-of-use of APIs with structure
- Performance and Optimization



Tale-2
Benefit of Dataset API





Tale-2
When to use DataFrame/Dataset ?

- If you want rich semantics, high-level abstractions, and domain specific APIs, use DataFrame or Dataset.
- If your processing demands high-level expressions, filters, maps, aggregation, averages, sum, SQL queries, columnar access and use of lambda functions on semi-structured data, use DataFrame or Dataset.
- If you want higher degree of type-safety at compile time, want typed JVM objects, take advantage of Catalyst optimization, and benefit from Tungsten's efficient code generation, use Dataset.
- If you want unification and simplification of APIs across Spark Libraries, use DataFrame or Dataset.
- If you are a R user, use DataFrame.
- If you are a Python user, use DataFrame and resort back to RDDs if you need more control.

Spark
Broadcast Variable

- Broadcast variables allow the programmer to keep a read-only variable cached on each machine rather than shipping a copy of it with tasks.

To define a broadcast variable we need to follow these constraints:

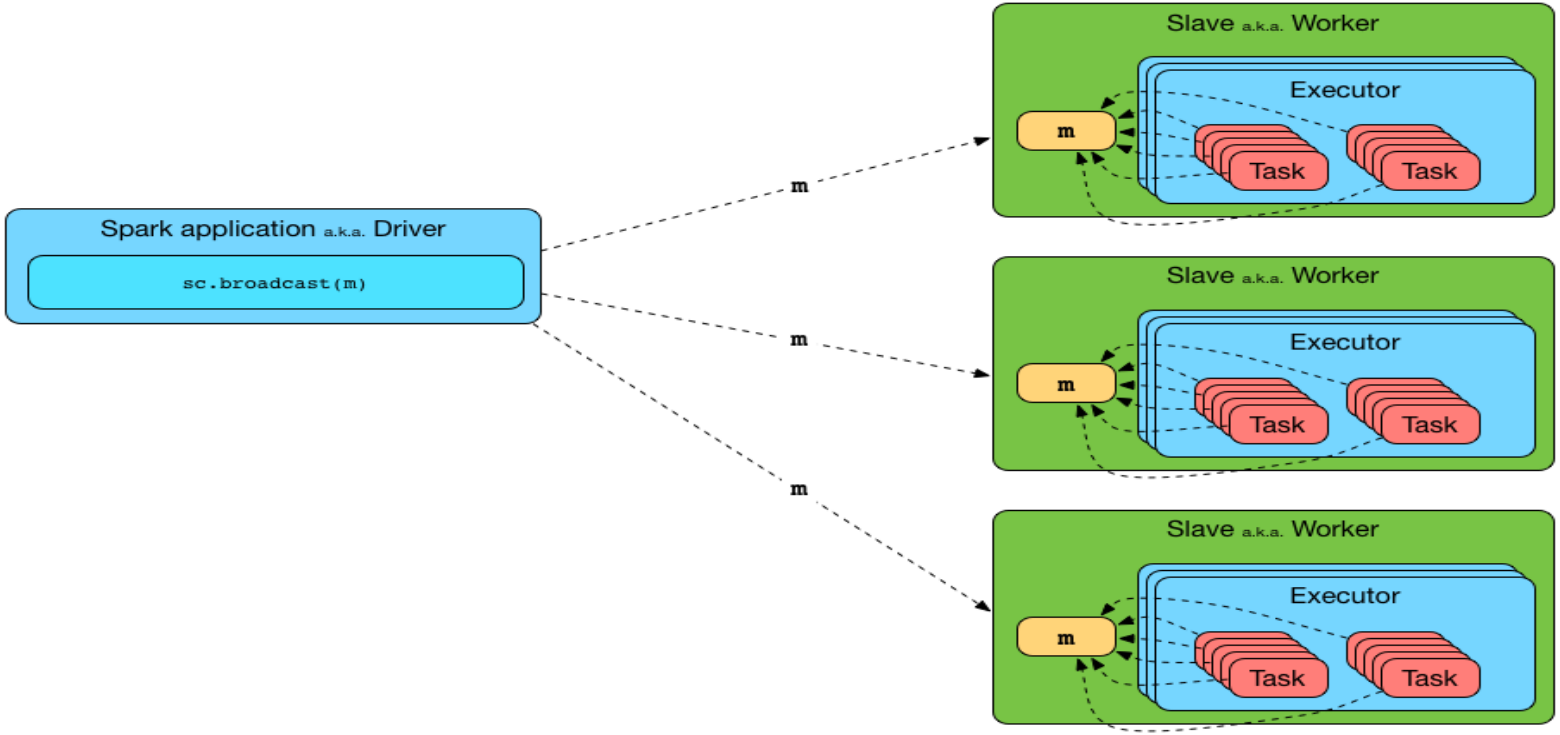
a) It is distributed across spark cluster.

b) It is a read-only variable cached on each machine.

c) It should be able to fit in memory on one machine. [must not use it for large tables or massive vectors].

d) It is immutable so we can’t change the content.

Spark
Broadcast Variable



Spark Accumulators

- Accumulator are write-only variables for executors. They can be added to by executors and read by the driver only.
- They are designed to be used safely and efficiently in parallel and distributed Spark computations and are meant for distributed counters and sums.
- You can create accumulators with or without a name, but only named accumulators are displayed in web UI.
- Accumulators are serializable so they can safely be referenced in the code executed in executors and then safely send over the wire for execution.

Spark Accumulators

Accumulators

Accumulable	Value
counter	45

Tasks

Index ▲	ID	Attempt	Status	Locality Level	Executor ID / Host	Launch Time	Duration	GC Time	Accumulators	Errors
0	0	0	SUCCESS	PROCESS_LOCAL	driver / localhost	2016/04/21 10:10:41	17 ms			
1	1	0	SUCCESS	PROCESS_LOCAL	driver / localhost	2016/04/21 10:10:41	17 ms		counter: 1	
2	2	0	SUCCESS	PROCESS_LOCAL	driver / localhost	2016/04/21 10:10:41	17 ms		counter: 2	
3	3	0	SUCCESS	PROCESS_LOCAL	driver / localhost	2016/04/21 10:10:41	17 ms		counter: 7	
4	4	0	SUCCESS	PROCESS_LOCAL	driver / localhost	2016/04/21 10:10:41	17 ms		counter: 5	
5	5	0	SUCCESS	PROCESS_LOCAL	driver / localhost	2016/04/21 10:10:41	17 ms		counter: 6	
6	6	0	SUCCESS	PROCESS_LOCAL	driver / localhost	2016/04/21 10:10:41	17 ms		counter: 7	
7	7	0	SUCCESS	PROCESS_LOCAL	driver / localhost	2016/04/21 10:10:41	17 ms		counter: 17	

Streaming

&

Spark Structured Streaming

databricks

Spark