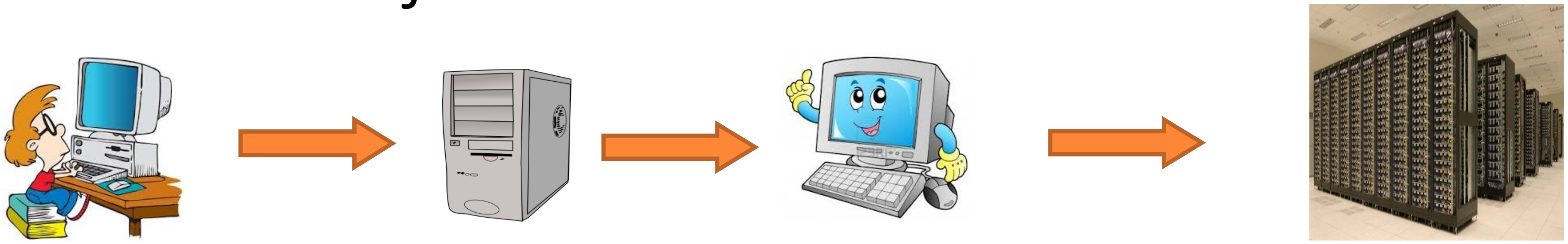
Apache SPARK fundamentals

Agenda

- Overview of job execution in Hadoop map reduce
- Recap of the drawbacks in map reduce
- Overview of spark architecture
- Spark deployment
- Job execution in SPARK
- RDD’s
- Actions and transformations
- Demo

Overview of job execution in HADOOP



Job client

Name node

Resource manager/YARN

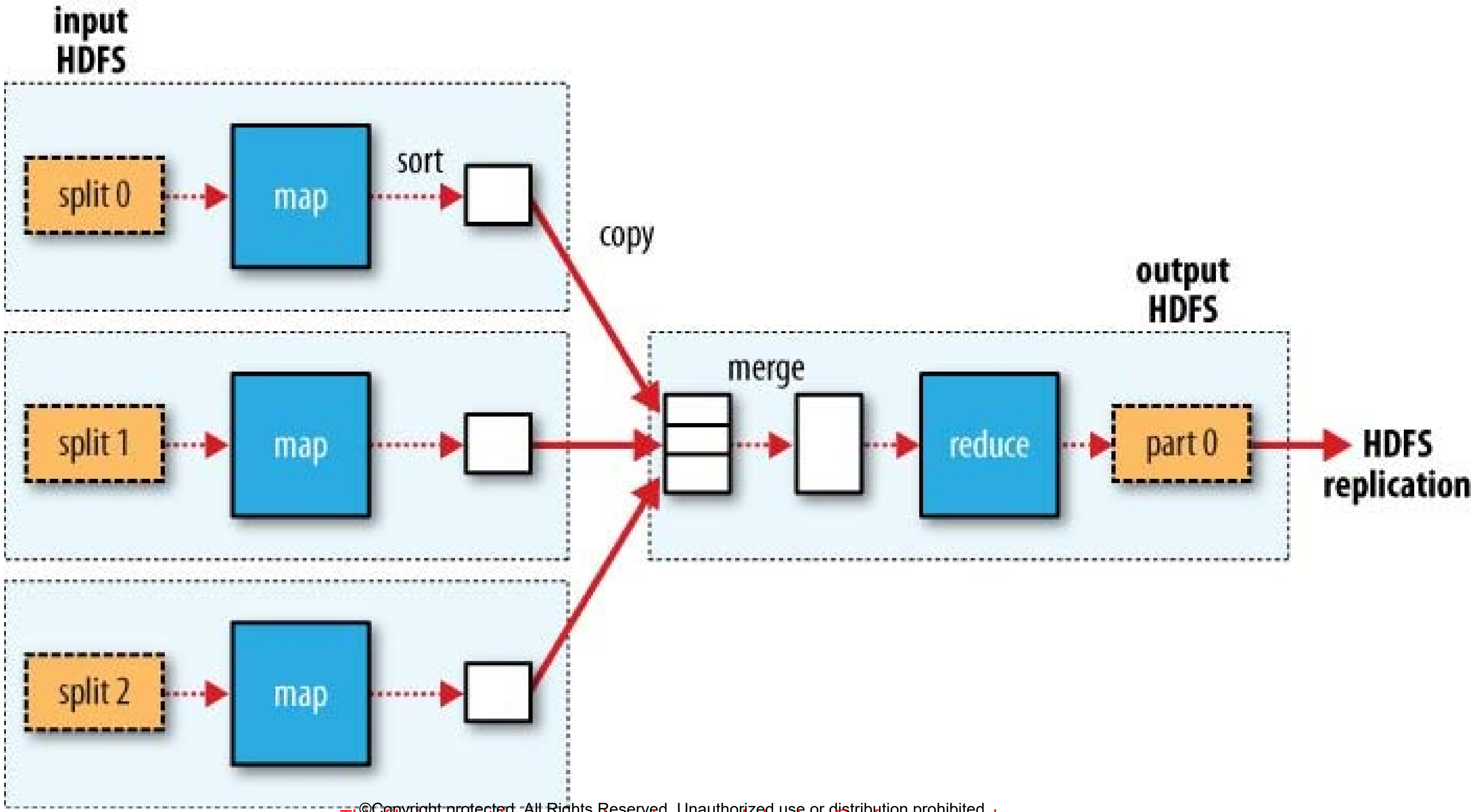
Data nodes

A job submitted by the user is picked up by the name node and the resource manager

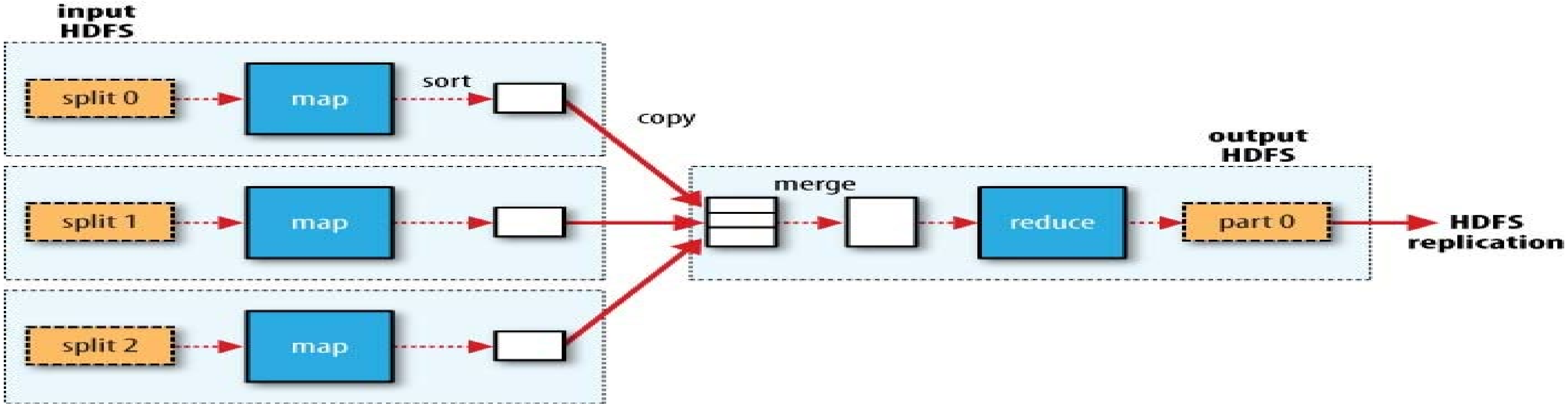
Job gets submitted to the name node and eventually resource manager is responsible for scheduling the execution of the job on the data nodes in the cluster

The data nodes in cluster consists the data blocks on which the user's program will be executed in parallel

The map and reduce stage



Disc I/O problem in Hadoop Map-Reduce



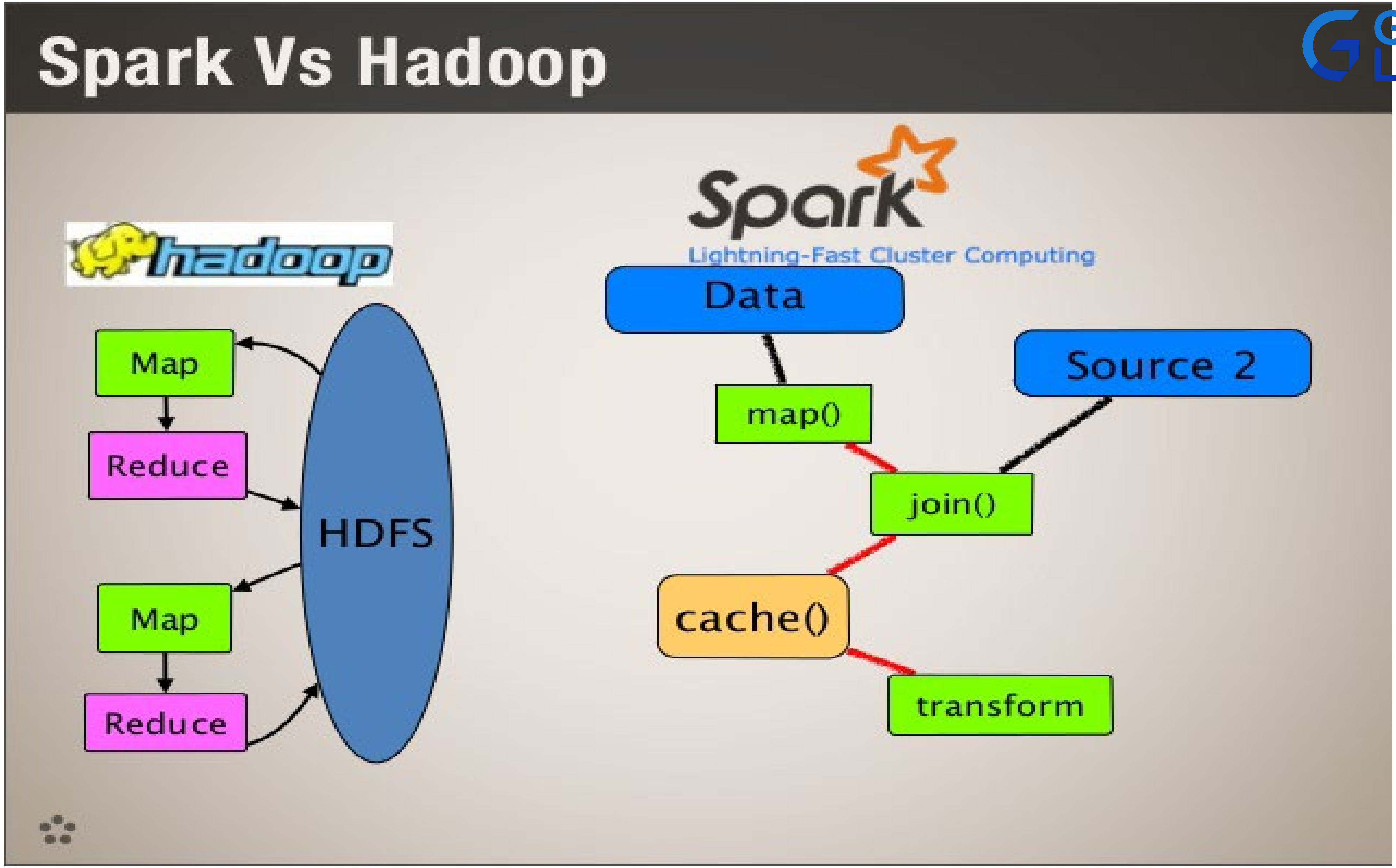
- The above example demonstrates a map-reduce job involving 3 mappers on 3 input splits
- There is 1 reducer
- Each input split on each data resides on the hard disc. Mapper reading them would involve a disc read operation.
- There would be 3 disc read operations from all the 3 mappers put together
- Merging in the reduce stage involves 1 disc write operation
- Reducer would write the final output file to the HDFS, which indeed is another disc write operation
- Totally there are a minimum of 5 disc I/O operations in the above example (3 from the map stage and 2 from reduce stage)
- The number of disc read operations from the map stage is equal to the number of input splits

Calculating the number of disc I/O operations on a large data set

- Typically a HDFS input split size would be 128 MB
- Let's consider a file of size 100TB and the number of file blocks on HDFS would be (100 * 1024 * 1024) / 128 = 8,19,200
- There would around 8.2 lakh mappers which needs to run on the above data set once a job is launched using Hadoop map reduce
- 8.2 lakh mappers means, 8.2 lakh disc read operations
- Disc read operations are 10 times slower when compared to a memory read operation
- Map-Reduce does not inherently support iterations on the data set
- Several rounds of Map-Reduce jobs needs to be chained to achieve the result of an iterative job in Hadoop
- Most of the machine learning algorithms involves an iterative approach
- 10 rounds of iterations in a single job leads to 8.2 lakh X 10 disc I/O operations

SPARK's approach to problem solving

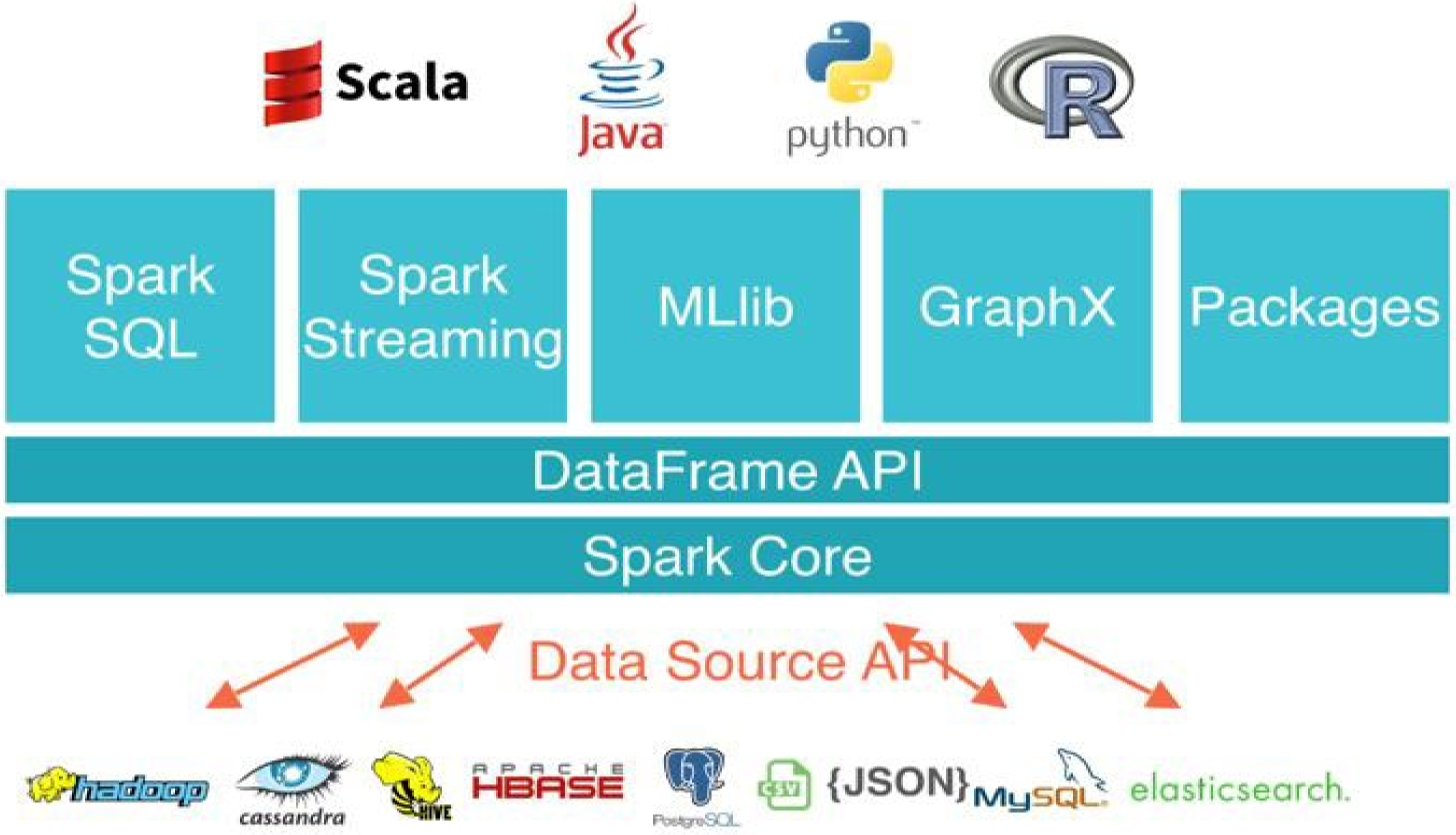
- **Spark allows the results of computation to be saved in the memory for future re-use**
- **Reading the data from the memory is much faster than that of reading from the disc**
- **Caching the result in memory is under the programmer's control**
- **Not always is possible to save such results completely in memory especially when the object is too large and memory is low**
- **In such cases the objects needs to be moved to the disc**
- **Spark, therefore is not a completely in memory based parallel processing platform**
- **Spark however is 3X to 10X faster in most of the jobs when compared to that of Hadoop**



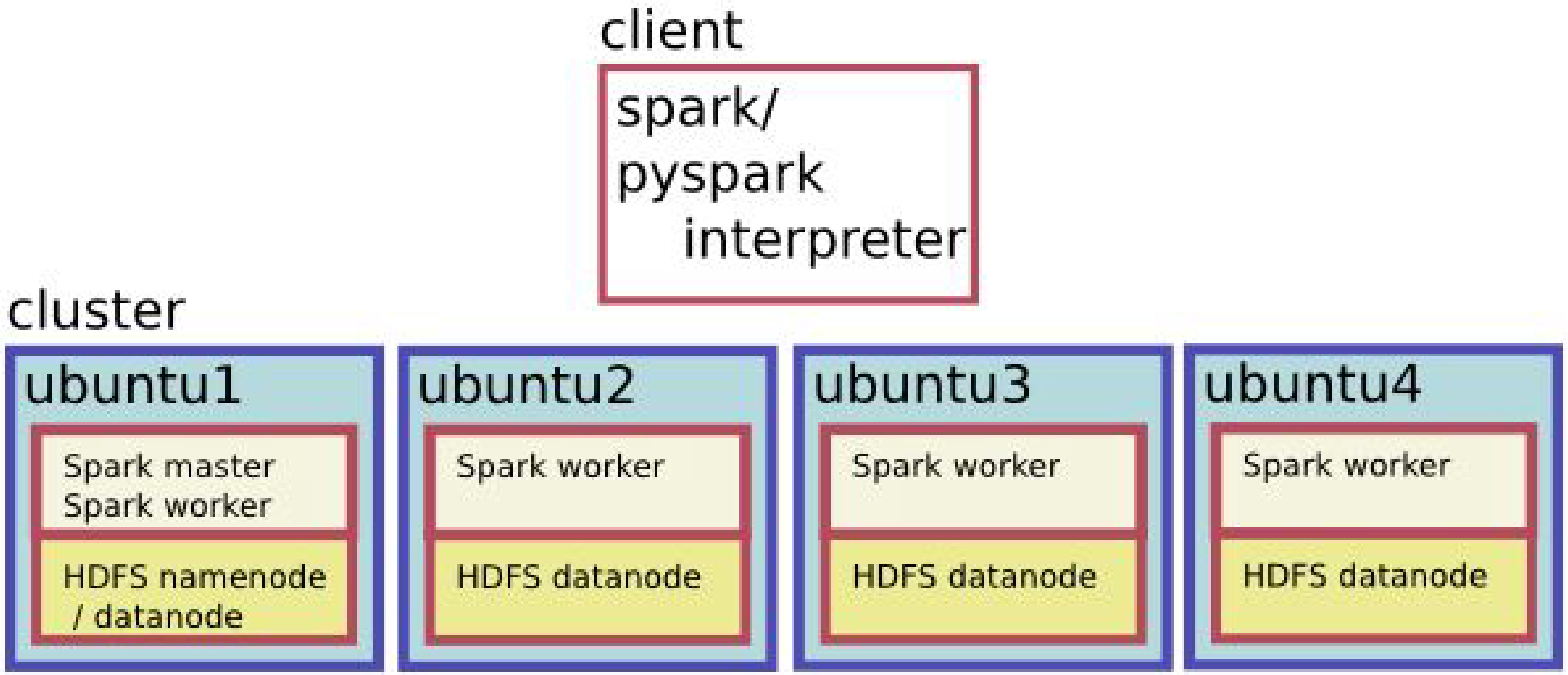
Spark Vs Hadoop

Monday, April 7, 14

Understanding SPARK architecture



Spark cluster on Ubuntu host machines



client

spark/ pyspark interpreter

cluster

ubuntu1

Spark master
Spark worker

HDFS namenode / datanode

ubuntu2

Spark worker

HDFS datanode

ubuntu3

Spark worker

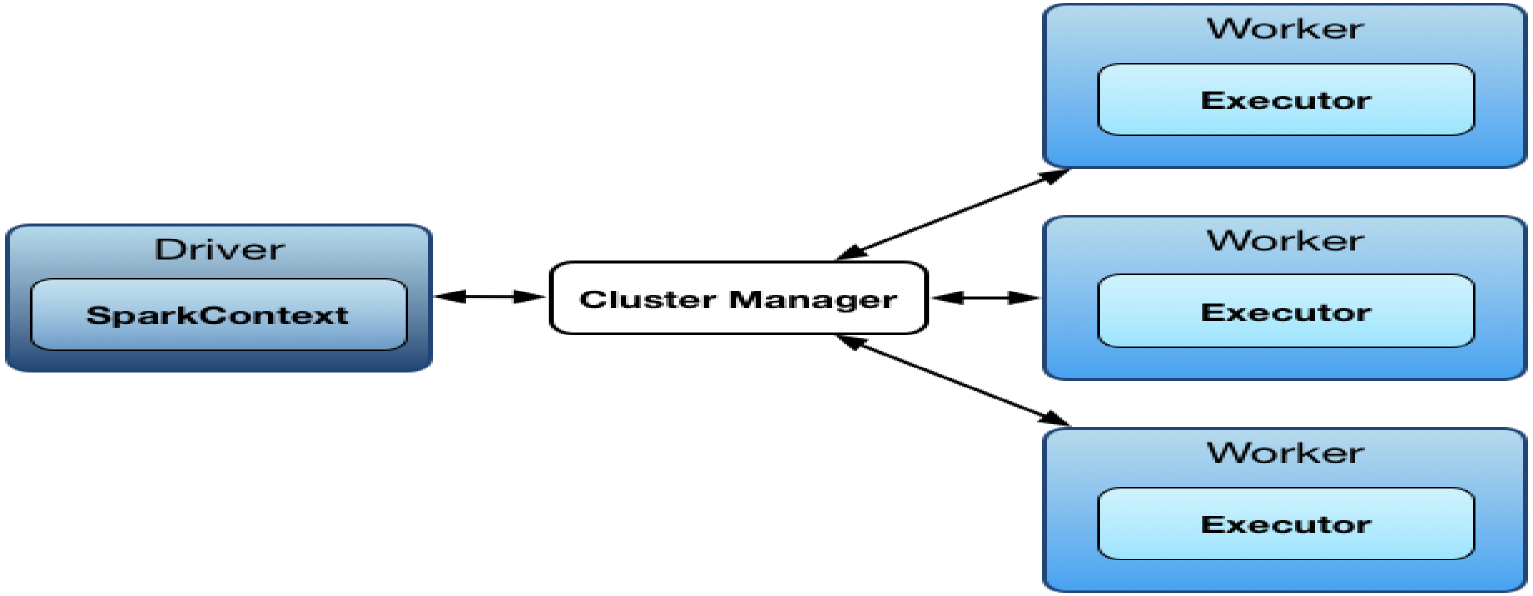
HDFS datanode

ubuntu4

Spark worker

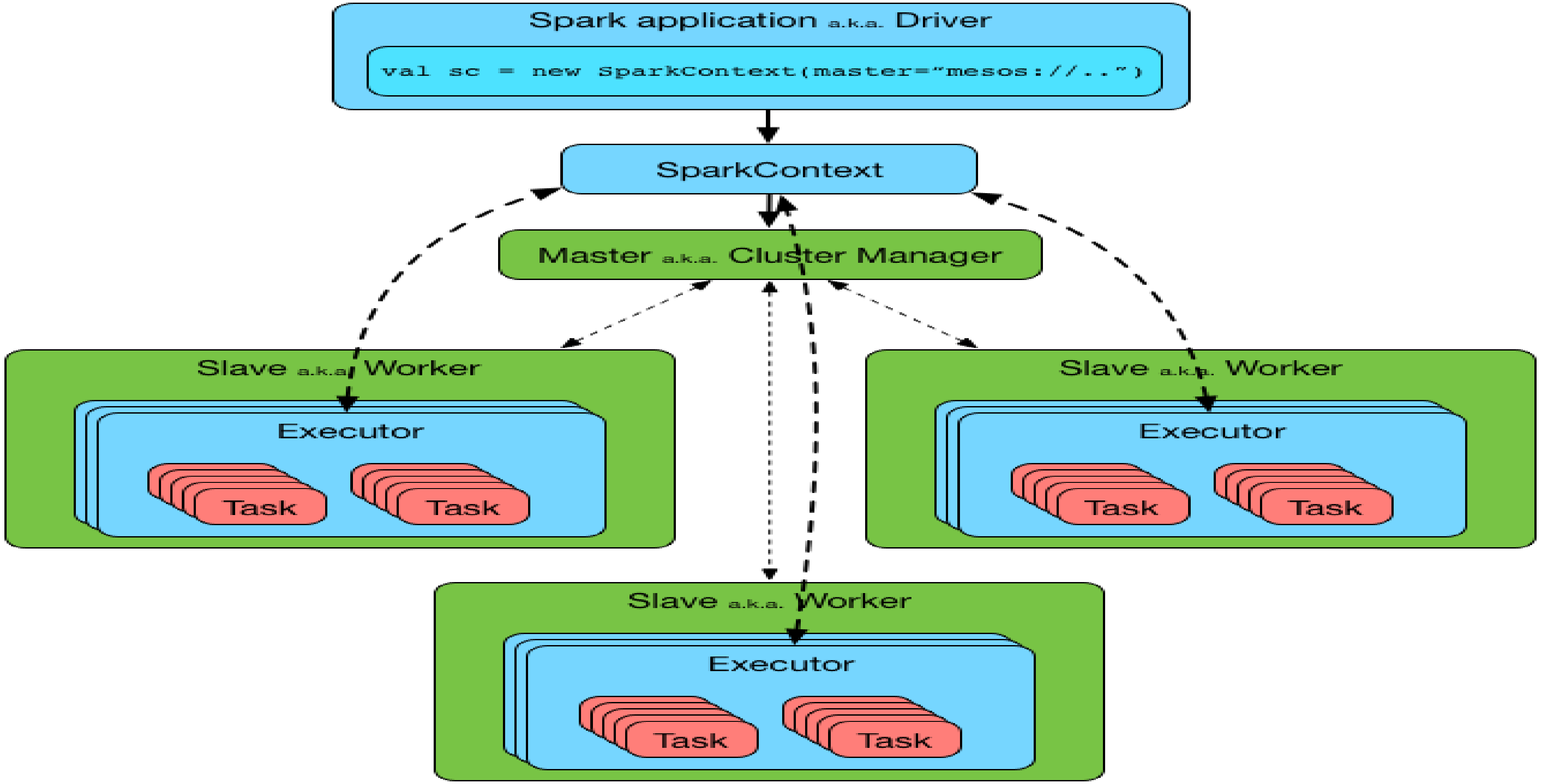
HDFS datanode

Simplifying SPARK architecture



- Driver is the starting point of a job submission (this can be compared to the driver code in Java MR)
- Cluster Manager can be compared to the Resource Manager in Hadoop
- Worker is a software service running on slave nodes, similar to the are the data nodes in a HADOOP cluster
- The executor is a container which is responsible for running the tasks

Job execution in a SPARK cluster



Spark deployment modes

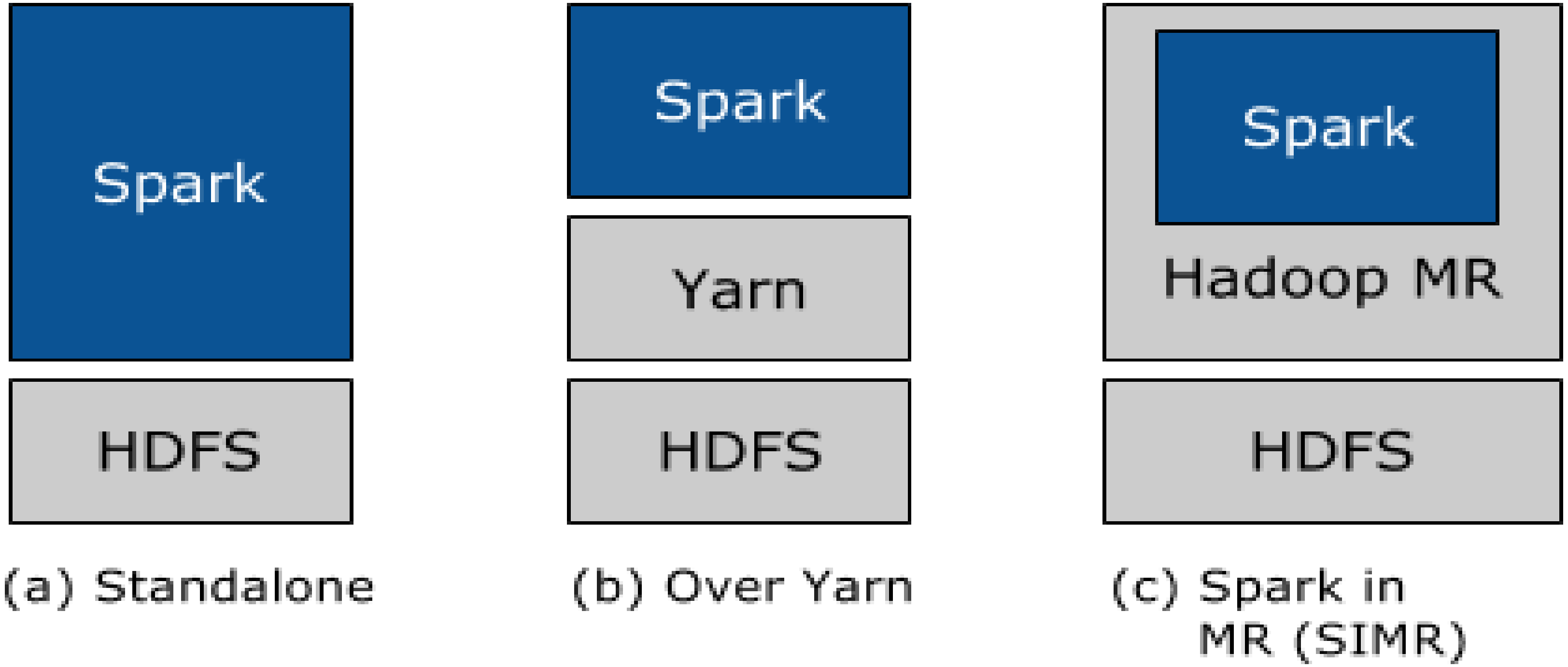


Image courtesy : https://databricks.com/blog/2014/01/21/spark-and-hadoop.html

Spark deployment modes

Standalone mode :

All the spark services run on a single machine but in separate JVM’s. Mainly used for learning and development purposes(something like the pseudo distributed mode of Hadoop deployment)

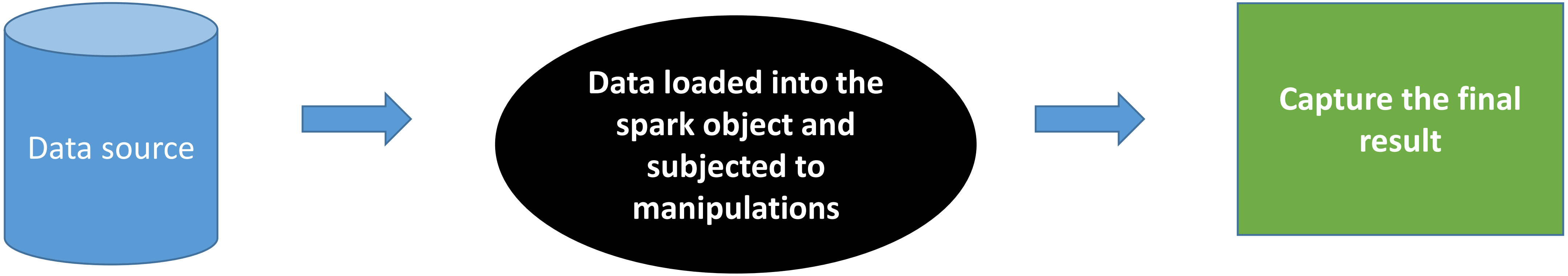
Cluster mode with YARN or MESOS:

This is the fully distributed mode of SPARK used in a production environment

Spark in Map Reduce (SIMR) :

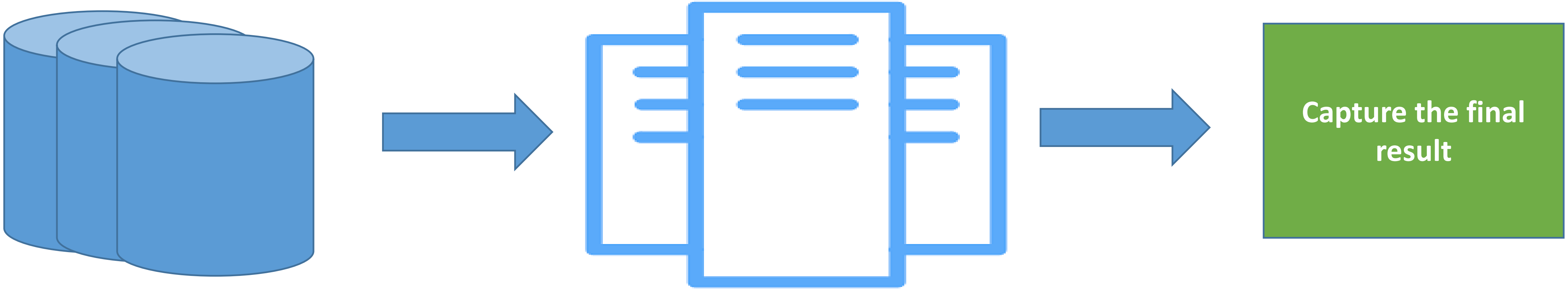
Allows Hadoop MR1 users to run their map reduce jobs as spark jobs

Loading spark data objects (RDD)


Data source
Data loaded into the spark object and subjected to manipulations
Capture the final result

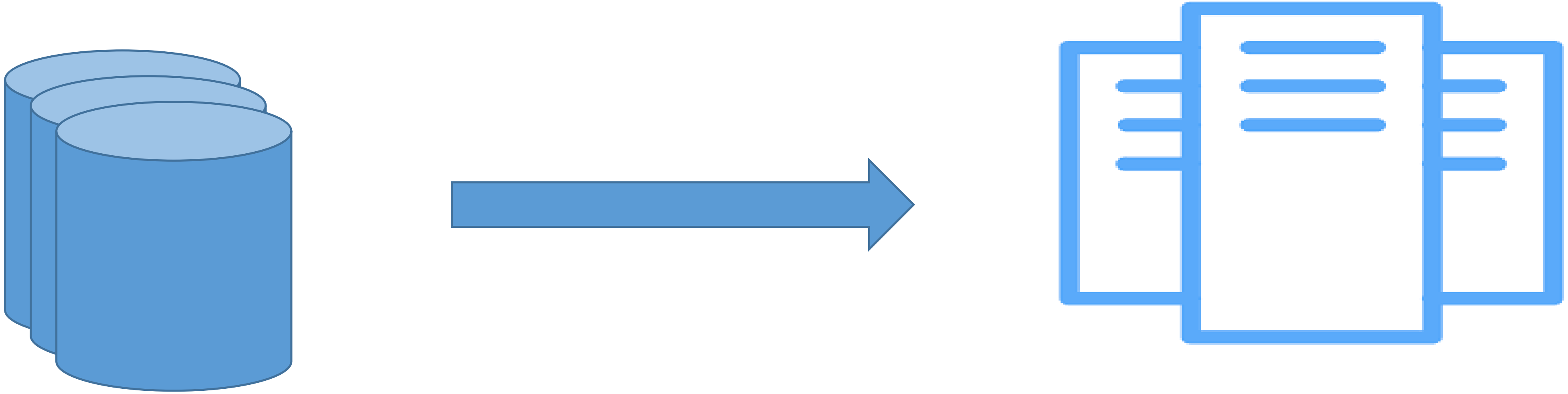
- **The data loaded into a SPARK object is called as an RDD**
- **A detailed discussion about RDD's will be covered shortly**

Under the hood

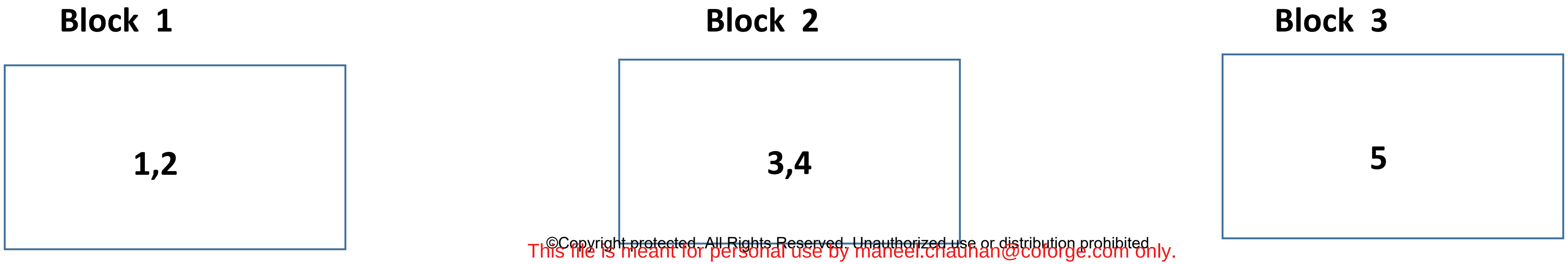


- **Job execution starts with loading the data from a data source (e.g. HDFS) into spark environment**
- **Data read from the hard drives of worker nodes and loaded into the RAM of multiple machines**
- **The data could be spread out into different files (each file could be a block in HDFS)**
- **After the computation, the final result is captured**

Partitions and data locality



- **Loading of the data from the hard drives to the RAM of the worker nodes is based on data locality**
- **The data in the data blocks is illustrated in the block diagram below**



Block 1

1,2

Block 2

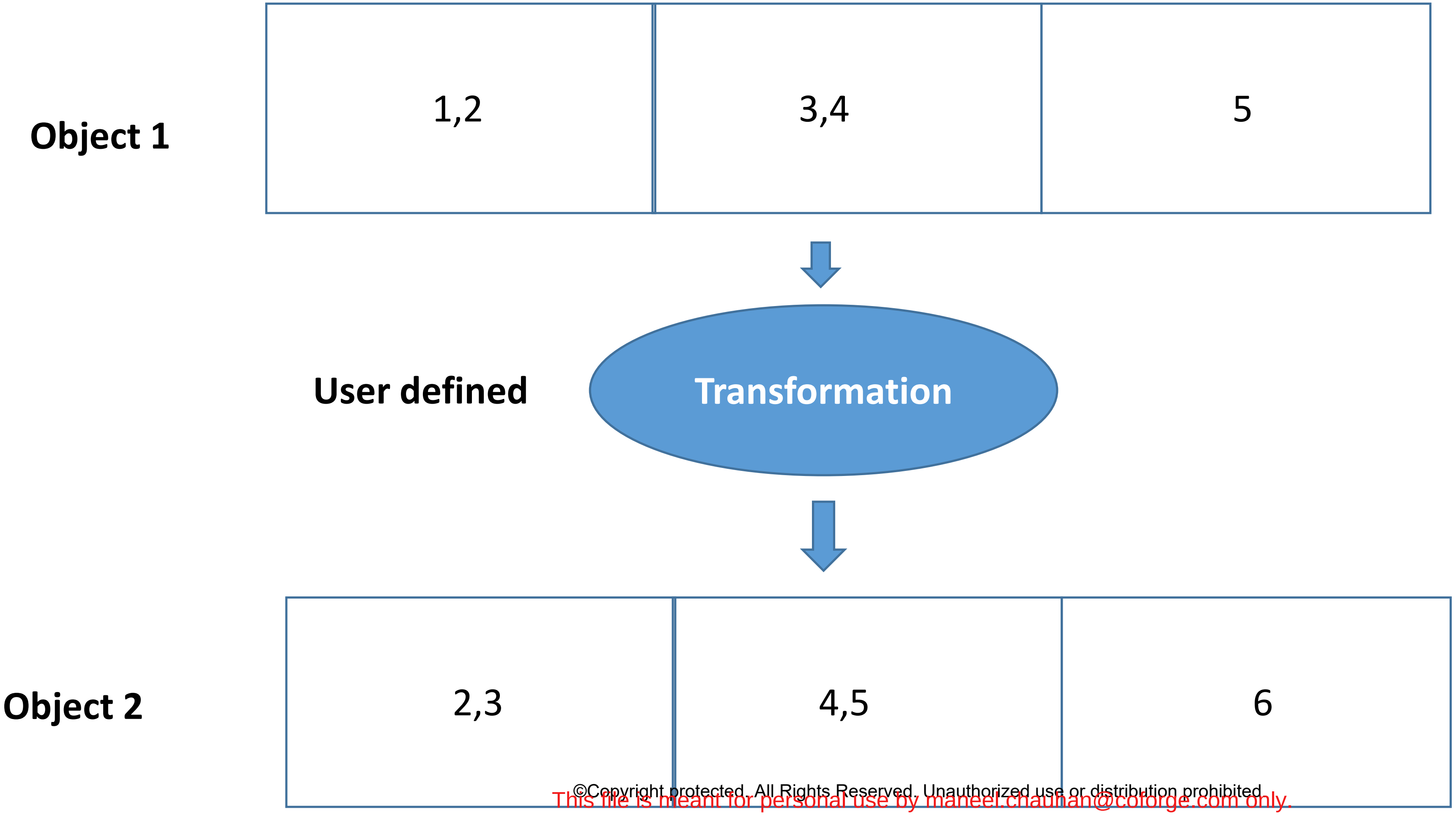
3,4

Block 3

5

Transformation the data object

- **The data in the objects cannot be modified as the very nature of the SPARK objects is *immutable* and the data in these objects are *partitioned* & *distributed* across nodes**



Object 1	1,2	3,4	5

User defined Transformation

Object 2	2,3	4,5	6

3 important properties of an RDD

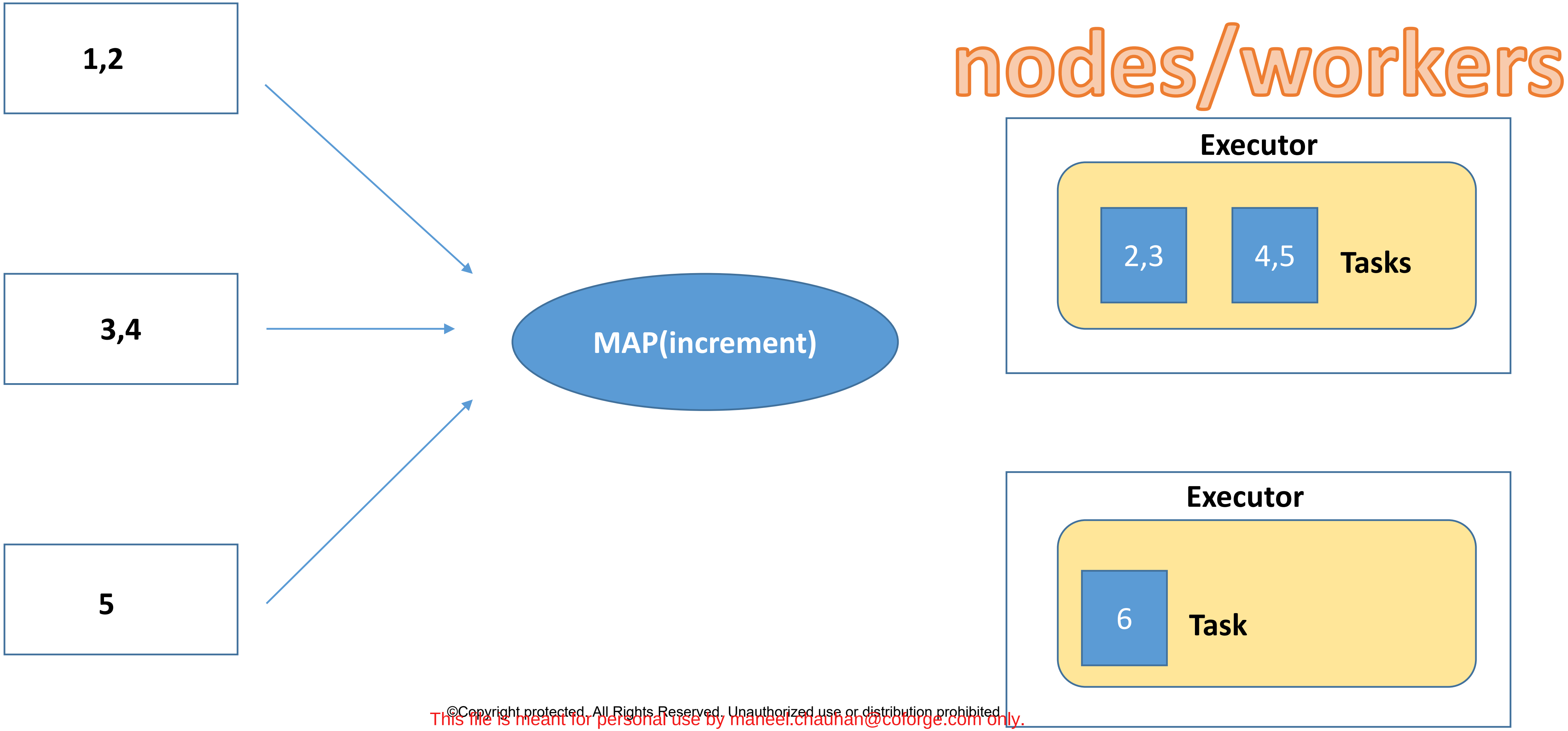
- We have just understood 3 important properties of an RDD in spark

1) They are immutable

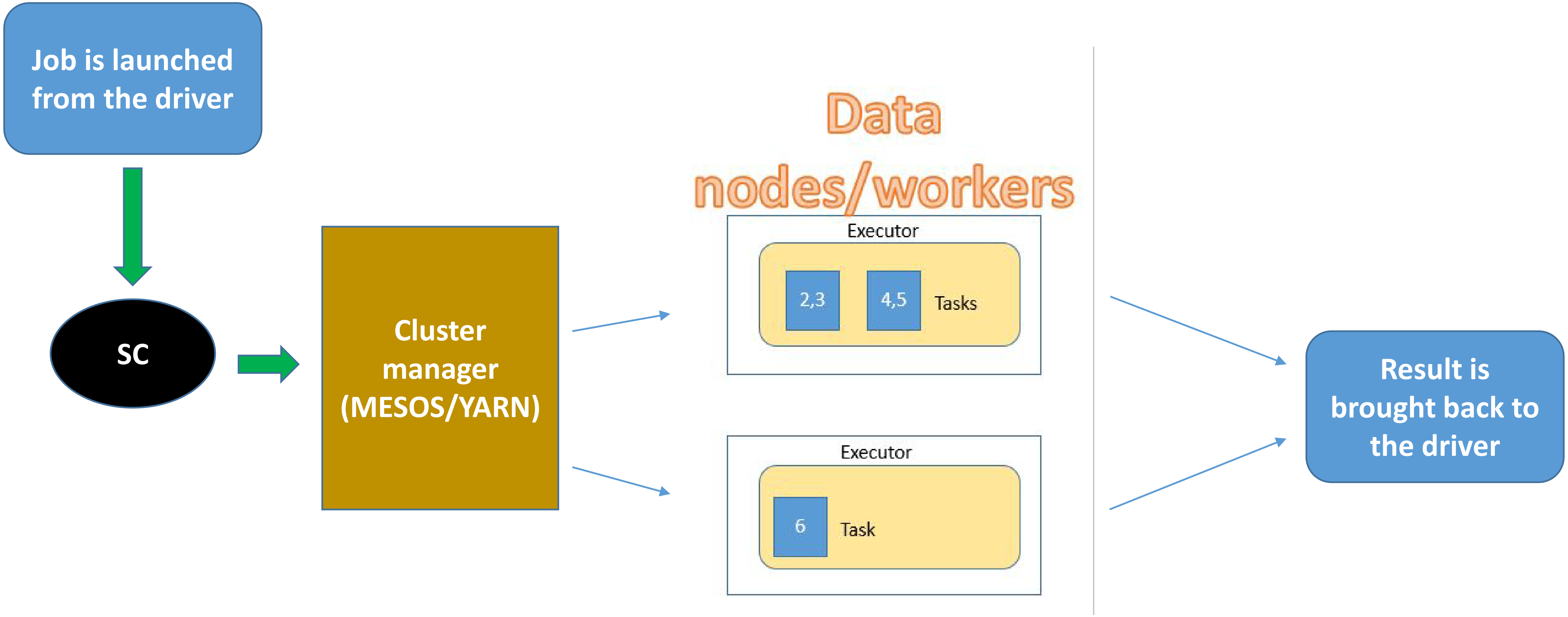
2) They are partitioned

3) They are distributed and spread across multiple nodes in a machine

Task execution

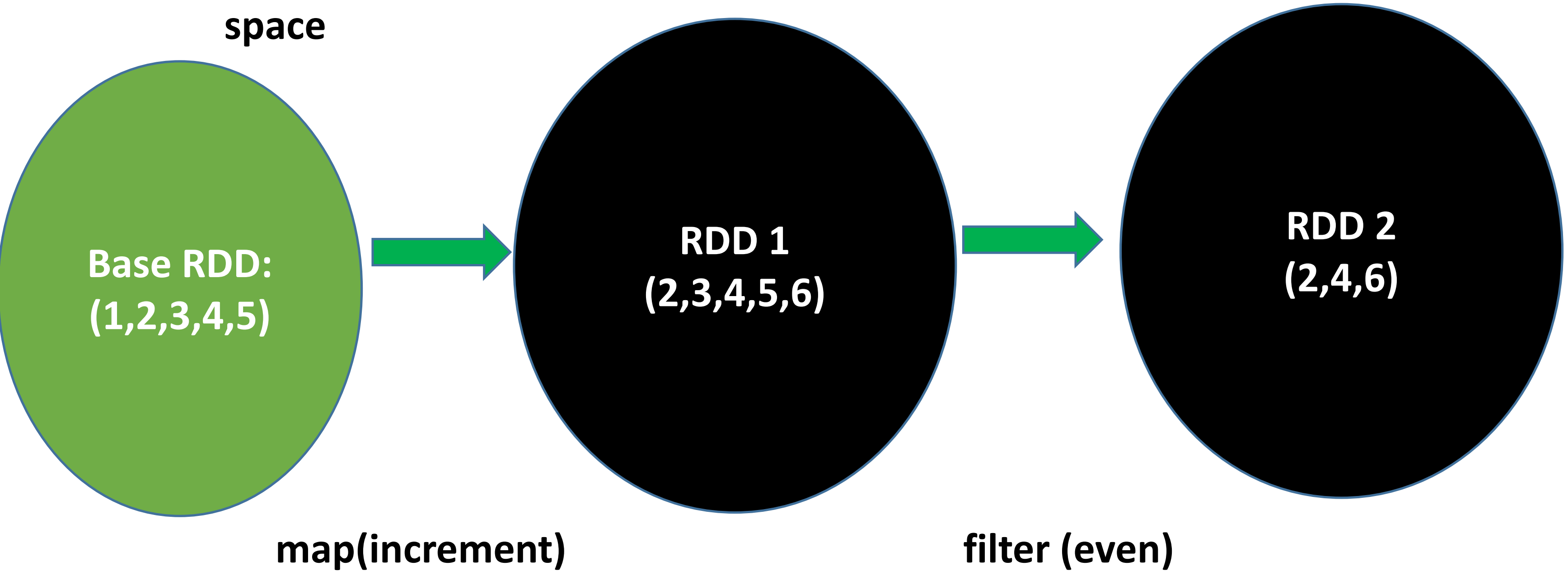


Workflow


Job is launched from the driver
SC
Cluster manager (MESOS/YARN)
Data nodes/workers
Executor
2,3
4,5
Tasks
Executor
6
Task
Result is brought back to the driver

RDD lazy evaluation (DAG creation)

- **Lets start calling these objects as RDD’s hereafter**
- **RDD’s are immutable & partitioned**
- **RDD’s mostly *reside in the RAM (memory)* unless the RAM (memory) is running of space**



Base RDD: (1,2,3,4,5)

RDD 1 (2,3,4,5,6)

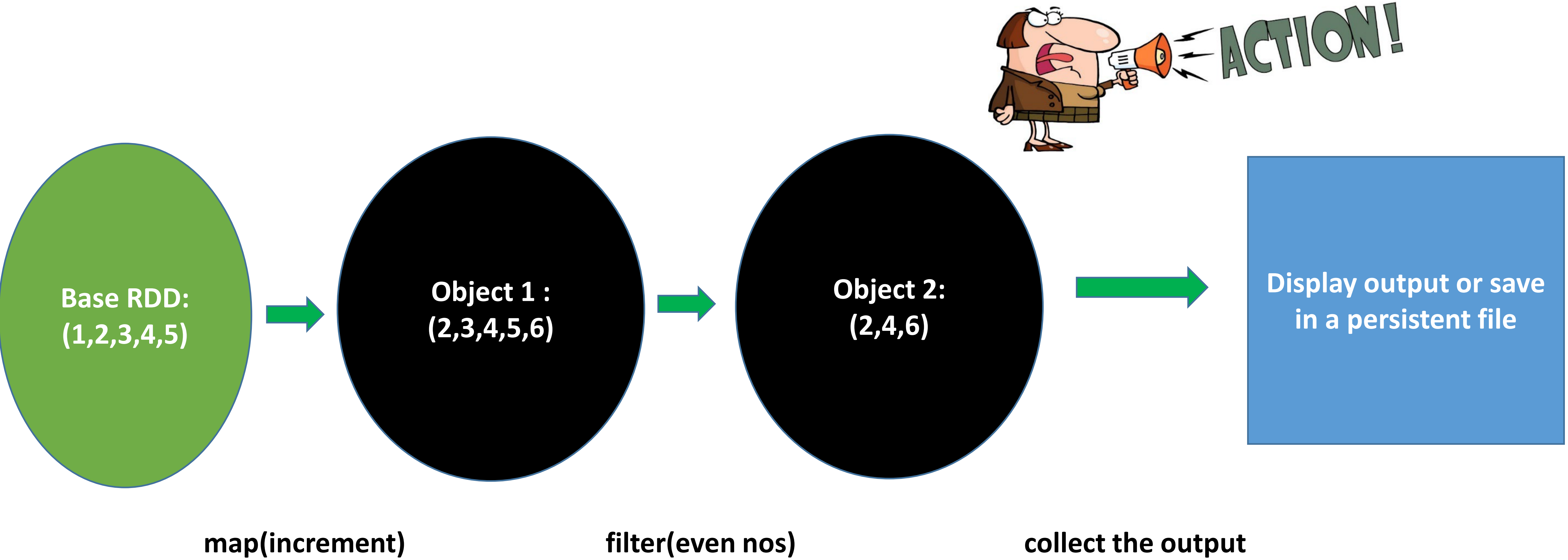
RDD 2 (2,4,6)

map(increment)

filter (even)

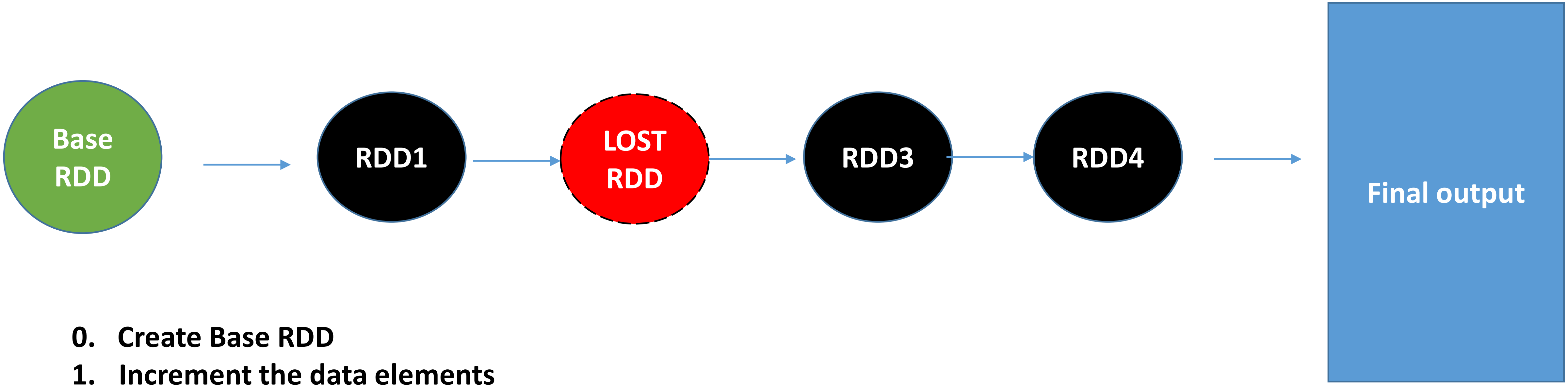
Nothing happens

Execution starts only when ACTION starts



RDD’s are fault tolerant (resilient)

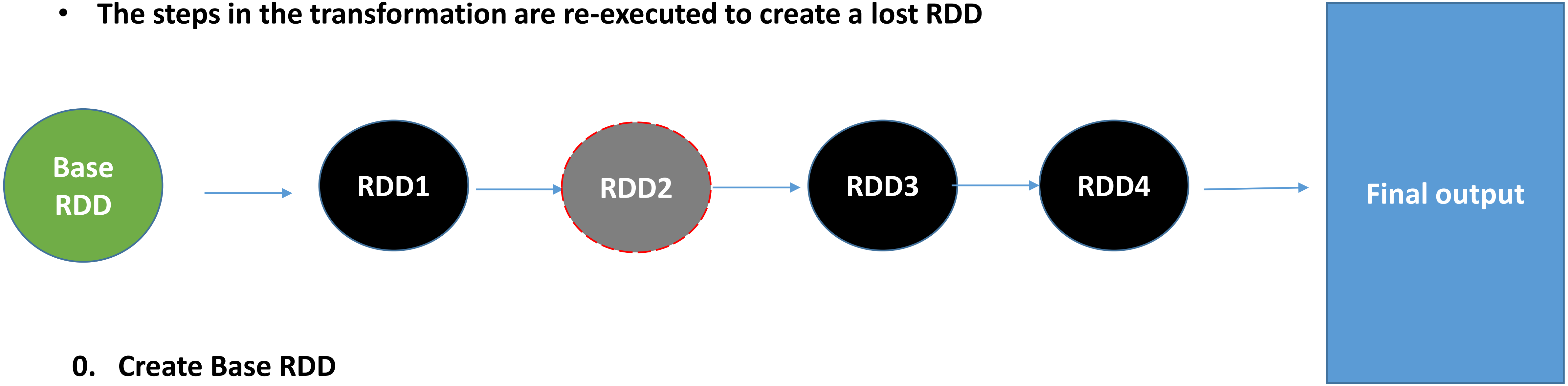
- RDD’s lost or corrupted during the course of execution can be reconstructed from the lineage



0. Create Base RDD
1. Increment the data elements
2. Filter the even numbers
3. Pick only those divisible by 6
4. Select only those greater than 78

RDD’s are fault tolerant (resilient)

- Lineage is a history of how an RDD was created from it’s parent RDD through a transformation
- The steps in the transformation are re-executed to create a lost RDD

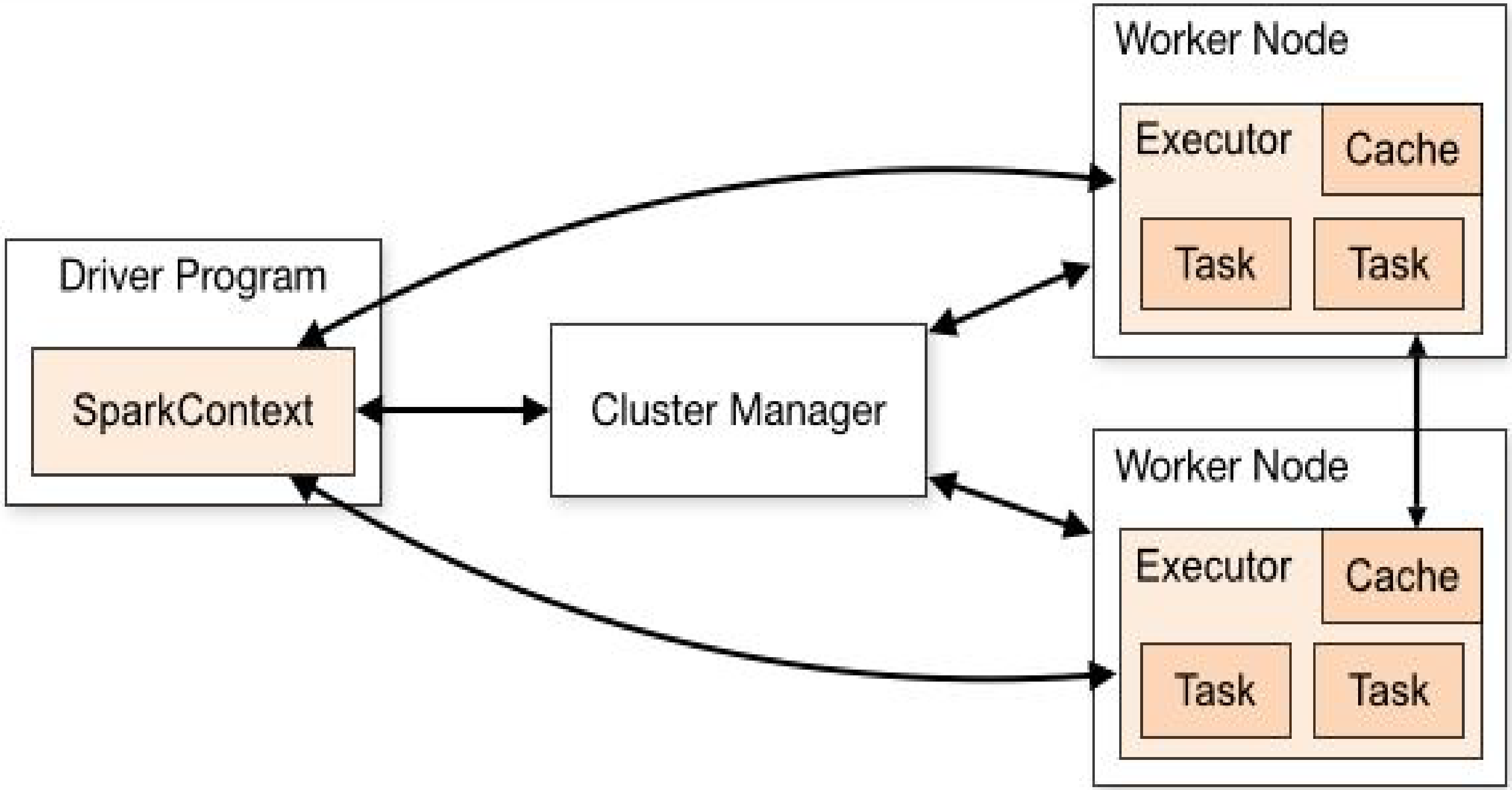


0. Create Base RDD
1. Increment the data elements
2. Filter the even numbers
3. Pick only those divisible by 6
4. Select only those greater than 78

Properties of RDD

- **They are RESILIENT DISTRIBUTED DATA sets**
- **Resilience (fault tolerant) due to the lineage feature in SPARK**
- **They are distributed and spread across many data nodes**
- **They are in-memory objects**
- **They are immutable**

Caching the RDD’s



SPARK's approach to problem solving

- Spark reads the data from the disc once initially and loads it into its memory
- The in-memory data objects are called RDD's in spark
- Spark can read the data from HDFS where large files are spit into smaller blocks and distributed across several data nodes
- Data nodes are called as worker nodes in Spark eco system
- Spark's way of problem solving also involves map and reduce operations
- The results of the computation can be saved in memory in case if its going to be re-used as the part of an iterative job
- Saving a SPARK object(RDD) in memory for future re-use is called caching

Note : **RDD's are not always cached by default in the RAM (memory). They will have to be written on to the disc when the system is facing a low memory condition due to too many RDD's already in the RAM. Hence SPARK is not a completely in memory based computing framework**

3 different ways of creating an RDD in SPARK

- Created by read a big data file directly from an external file system, this is used while working on large data sets

- Using the parallelize API, this is usually used on small data sets

- Using the makeRDD API

Actions and transformations

- Transformations are any operations on the RDD's which are subjected to manipulations during the course of analysis
- A SPARK job is a collection of a sequence of a several TRANSFORMATIONS
- The above job is usually a program written in SCALA or Python
- Actions are those operations which trigger the execution of a sequence of transformations
- There are over 2 dozen transformations and 1 dozen actions
- A glimpse of the actions and transformations in SPARK can be found in the official SPARK programming documentation guide
  https://spark.apache.org/docs/2.2.0/rdd-programming-guide.html
- Most of them will be discussed in detail during the demo

Summary

- A quick recap about the drawbacks of Map-Reduce in Hadoop
- Spark architecture
- Spark deployment modes
- Job execution in SPARK
- RDD's