UNIVERSITAT DE BARCELONA

# Master in Fundamental Principles of Data Science

Dr Rohit Kumar

# Spark Fundamentals

# Spark Intro

- Its not a replacement of Hadoop
- Its not for big data storage
- It's a ***general purpose*** distributed computing platform for ***scalable efficient analysis*** of big data.

# Spark history

- First research paper in 2010.
- First stable release in 2012.
- Became the top level Apache Project. In 2014*
- Became the most active project in Apache foundation.

  In 2015
- Latest version Spark 2.4.5 released (Feb 08, 2020)

*Won Gray Sort Benchmark. Sorted 100TB of data using 206 EC2 i2.8xlarge machines in 23 minutes. The previous world record was 72 minutes, set by a Hadoop MapReduce cluster of 2100 nodes.

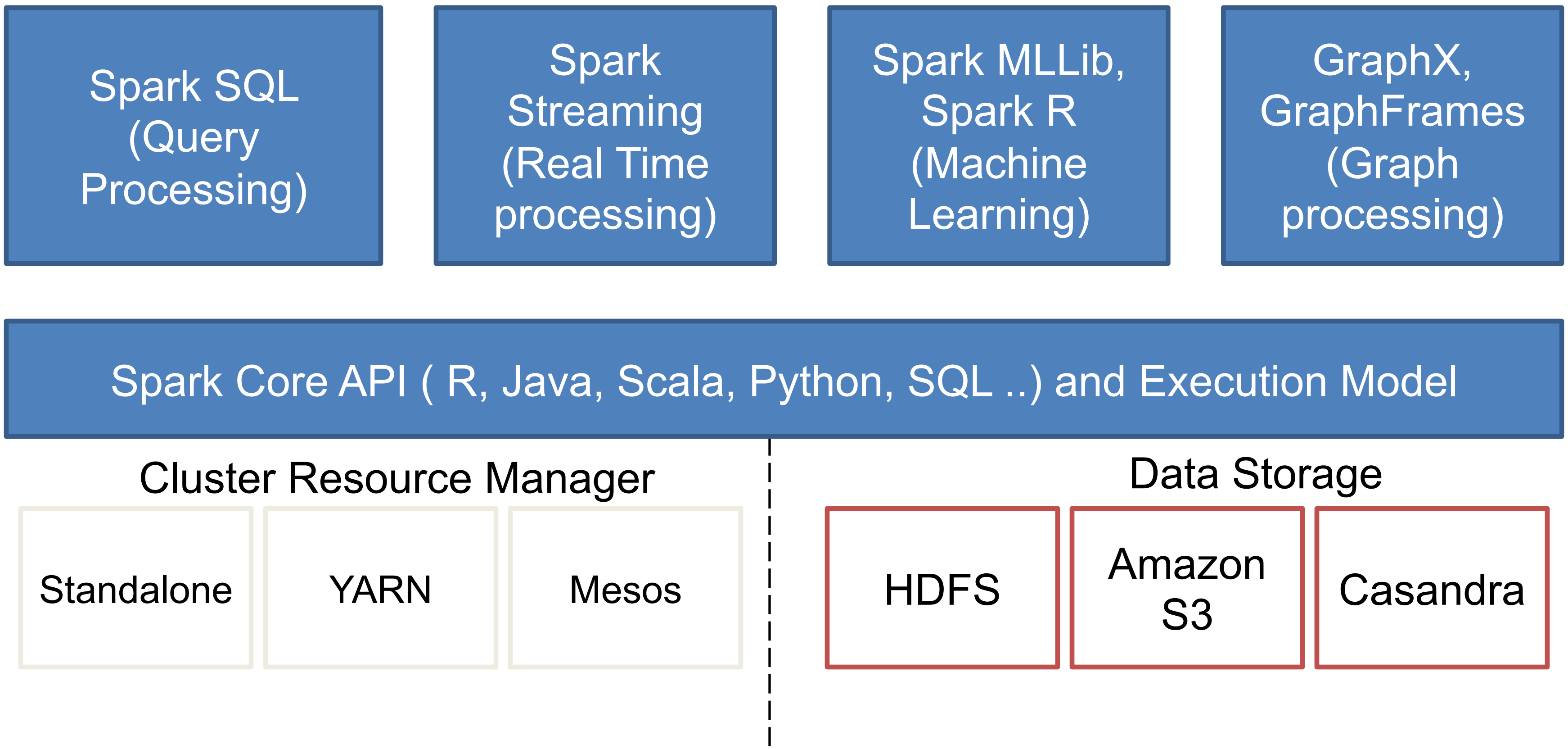
.

UNIVERSITAT DE BARCELONA

# Hadoop vs Spark

| Hadoop | Spark |
|---|---|
| Disk only | Both in-memory and disk storage |
| Only Map and Reduce operations | Many transformations and action making it easier to write application |
| Batch execution | Batch, interactive and streaming execution |
| Java language supported | Supports Java, Scala, R and Python |
| | |

# Apache Spark Platform



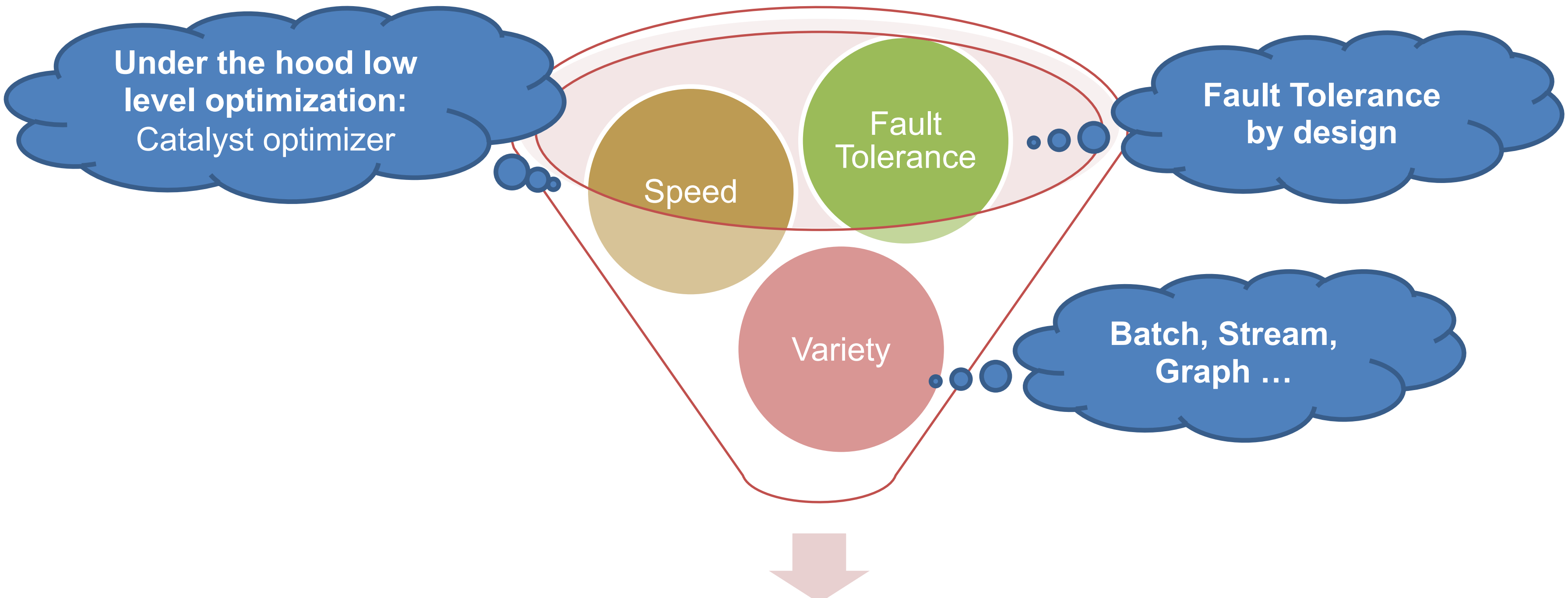
# Spark Programing language

- Scala
- Python
- Java

Spark is built using Scala and is the best option to work on Spark as both Python and Java interface will internally convert code in Scala.

UNIVERSITAT DE BARCELONA

# Spark Advantage

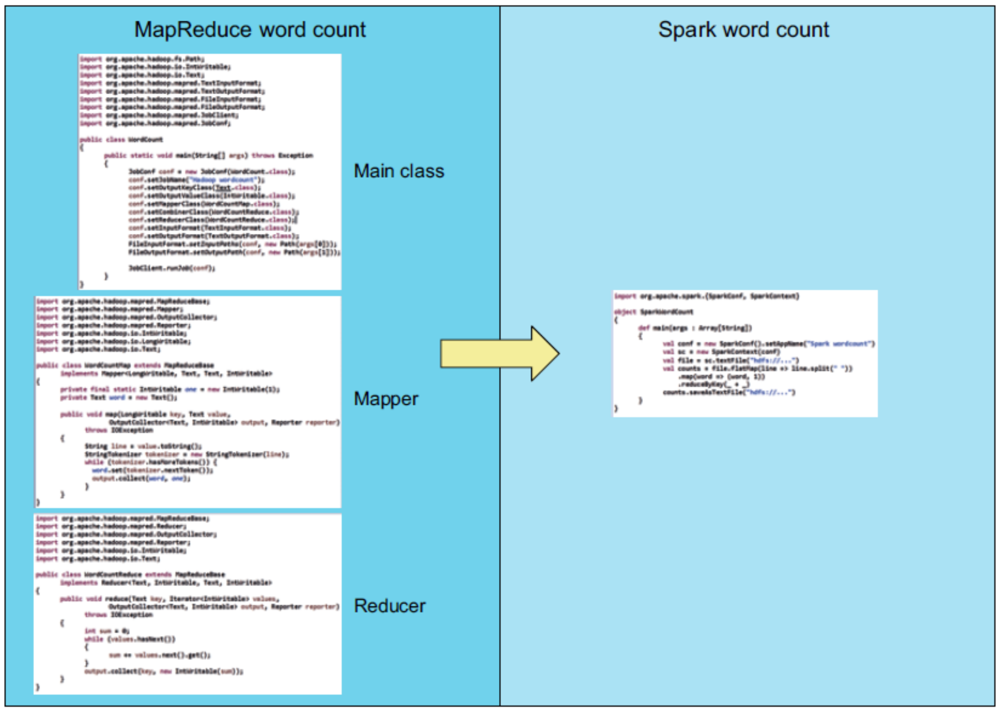



Spark
Lightning-Fast Cluster Computing

# Why so popular

- **Easy to Get Started**
- **Unified Engine for Diverse Workloads:** “One of the Spark project goals was to deliver a platform that supports a very wide array of diverse workflows - not only  MapReduce batch jobs (there were available in Hadoop already at that time), but also iterative computations like graph algorithms or Machine Learning.” – Matei Zaharia
- **An active community of more than 500 contributors**
- **Rich Standard Library:** MLLib, Spark-SQL, GraphX
- **Interactive Exploration / Exploratory Analytics**

# Spark makes it easy

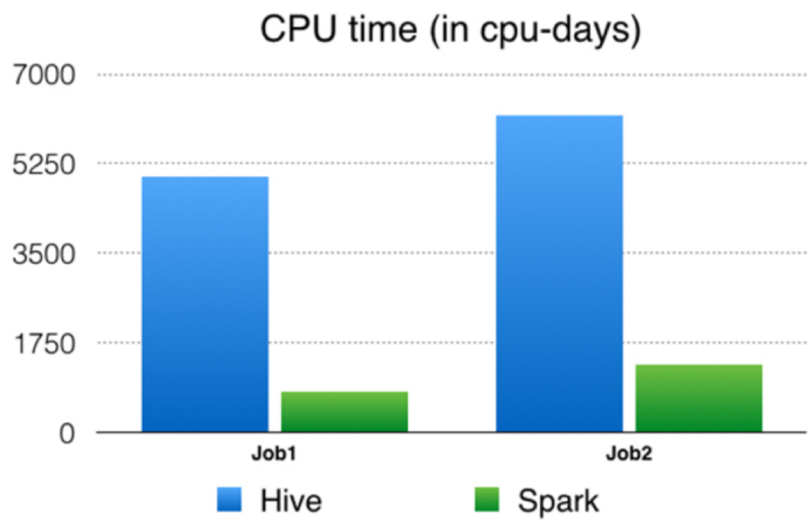



Src: Spark in Action book(page 7)

# Big Users

- **Alibaba Taobao:** built one of the world's first Spark on YARN production clusters.
- **eBay Inc:** Using Spark core for log transaction aggregation and analytics.
- **TripAdvisor:** Using Apache Spark for Massively Parallel NLP.

# A 60 TB+ production use case from Facebook




https://databricks.com/blog/2016/08/31/apache-spark-scale-a-60-tb-production-use-case.html

# Spark architecture

A simple architecture diagram

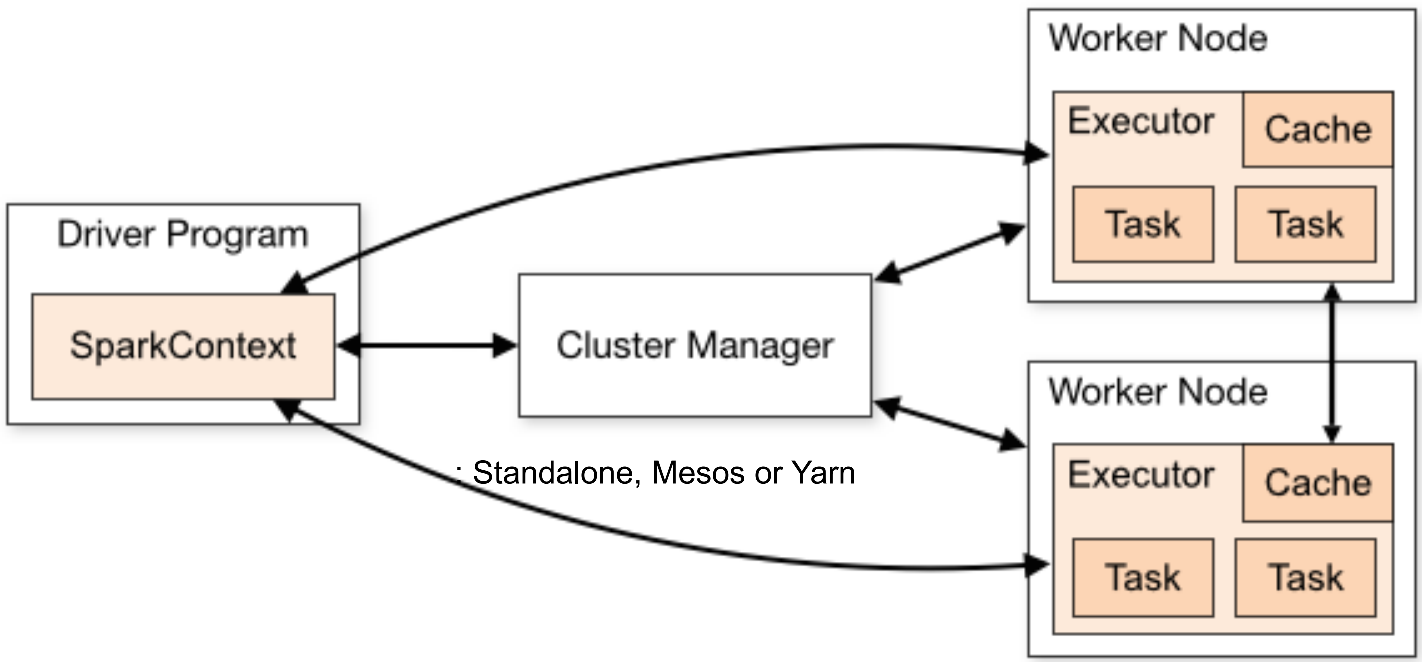



Image src: https://spark.apache.org/docs/latest/cluster-overview.html

# Spark Driver

- Master node in the Spark application.
- Entry point of the Spark shell.
- Runs the main() method of the application.
- Creates and maintain the **SparkContext**.
- Connects to the *cluster manager* and requests

# Cluster Manager

- Allocates resource on request from driver
- Keep track on dead or live nodes in the cluster.
- Enables the execution of jobs submitted by the driver on the worker nodes.
- All driver instances talk to this one cluster manager.

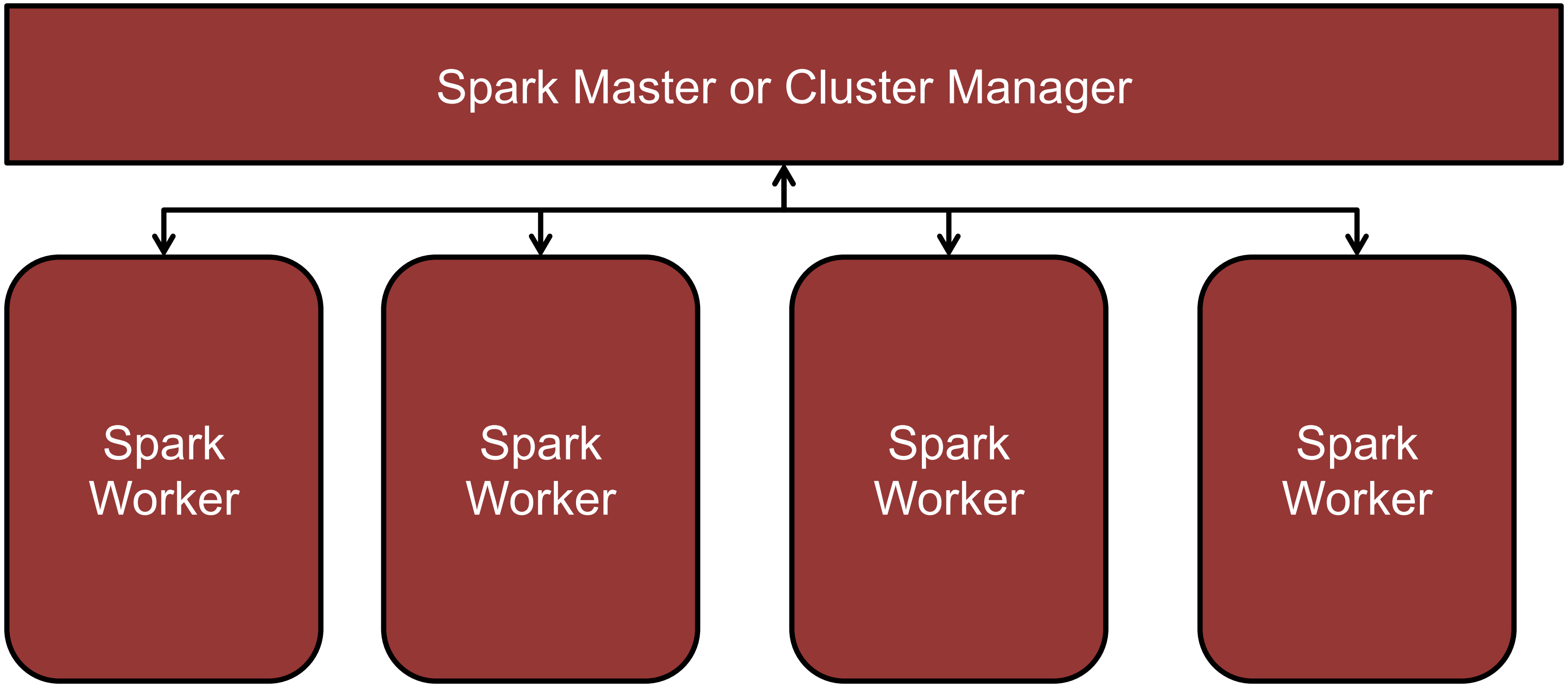
University de Barcelona

# Spark Worker

- Executes the actual business logic submitted by the driver.
- Each executor has its on JVM.
- The executors run separate threads to execute the code.
- Do the main logic of code, read data from resource and write data.

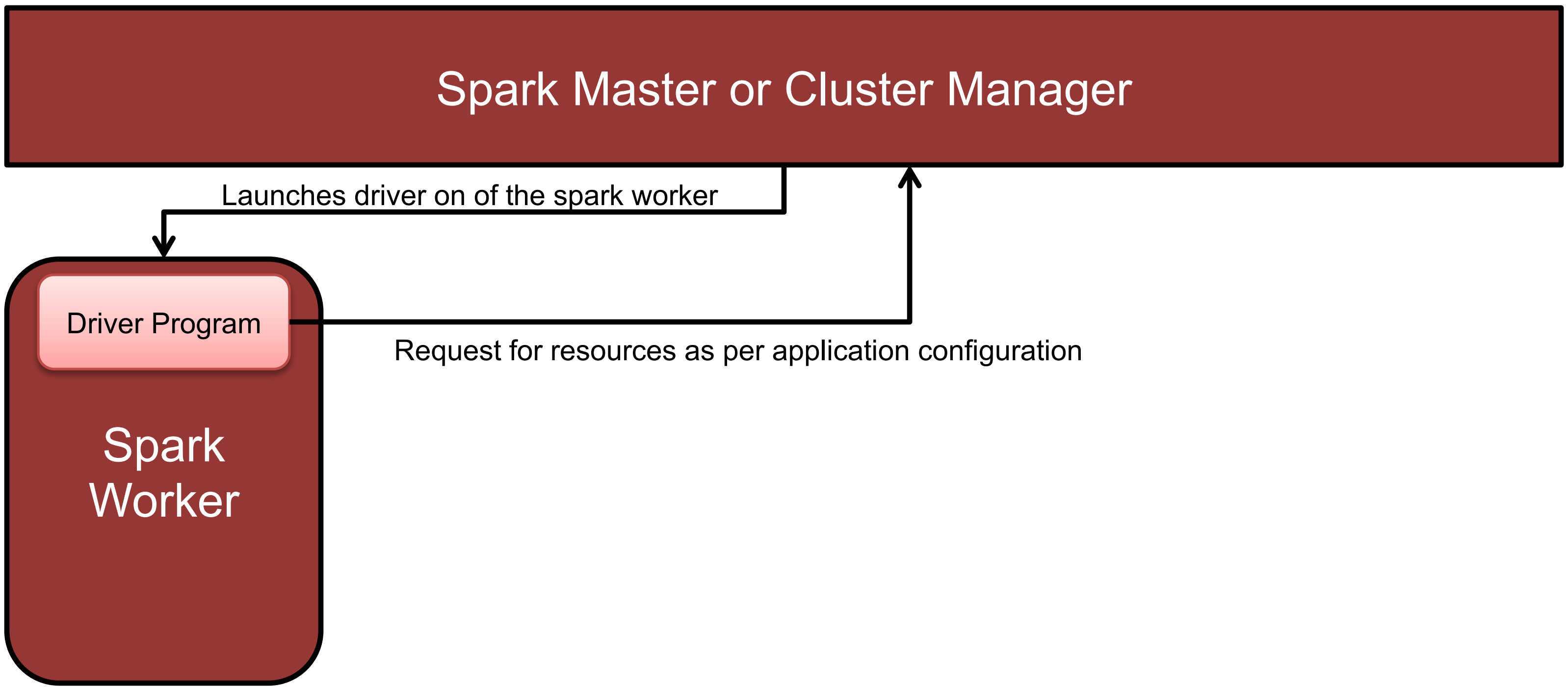
# Spark application life cycle(detailed)

After starting spark cluster



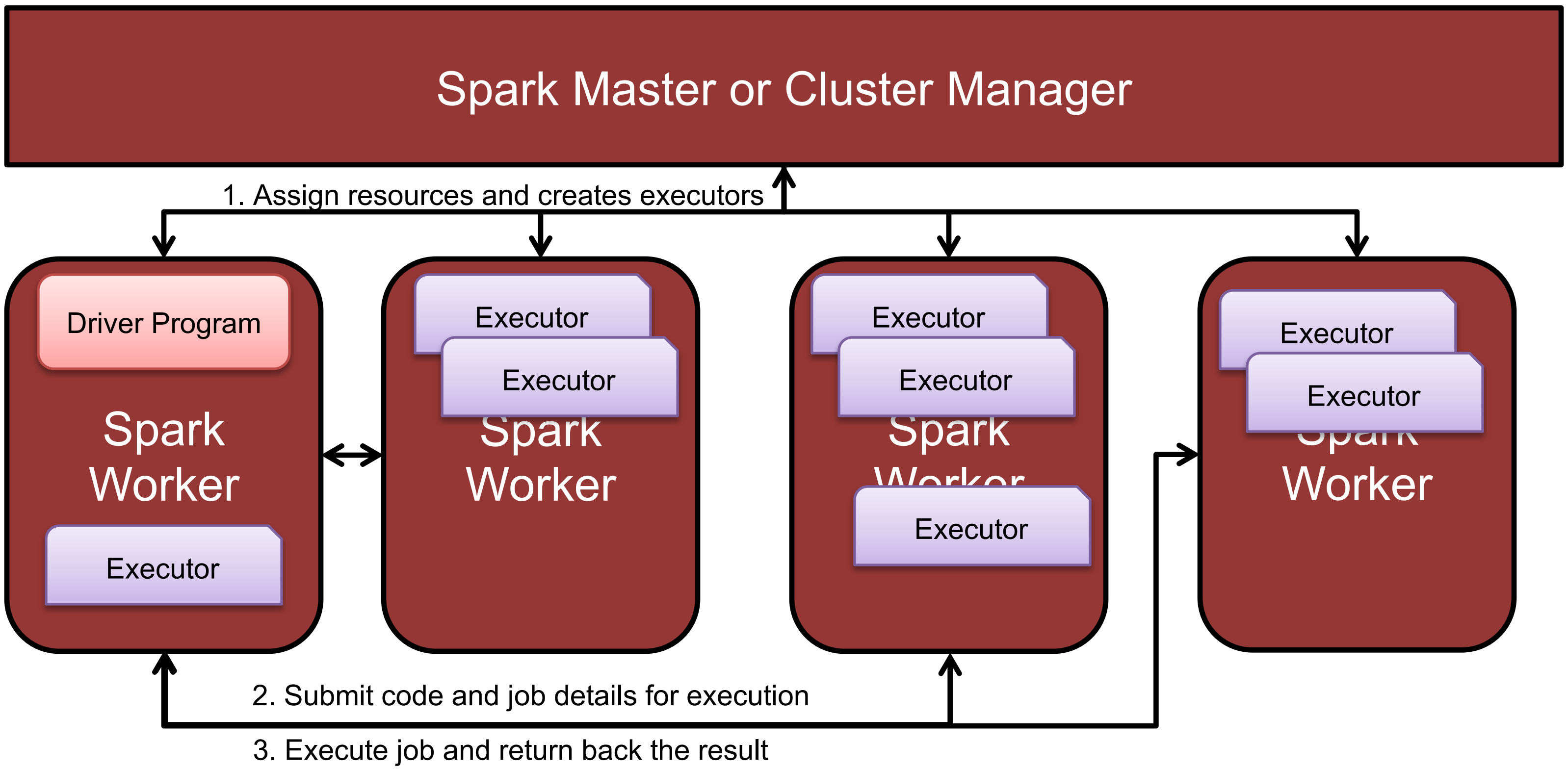
# Spark application life cycle(detailed)

After launching an application



# Spark application life cycle(detailed)

After launching an application



# RDD intro

## Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing

Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma,
Murphy McCauley, Michael J. Franklin, Scott Shenker, Ion Stoica
*University of California, Berkeley*

### Abstract

We present Resilient Distributed Datasets (RDDs), a distributed memory abstraction that lets programmers perform in-memory computations on large clusters in a fault-tolerant manner. RDDs are motivated by two types of applications that current computing frameworks handle inefficiently: iterative algorithms and interactive data mining tools. In both cases, keeping data in memory can improve performance by an order of magnitude. To achieve fault tolerance efficiently, RDDs provide a restricted form of shared memory, based on coarse-grained transformations rather than fine-grained updates to shared state. However, we show that RDDs are expressive enough to capture a wide class of computations, including recent specialized programming models for iterative jobs, such as Pregel, and new applications that these

tion, which can dominate application execution times.

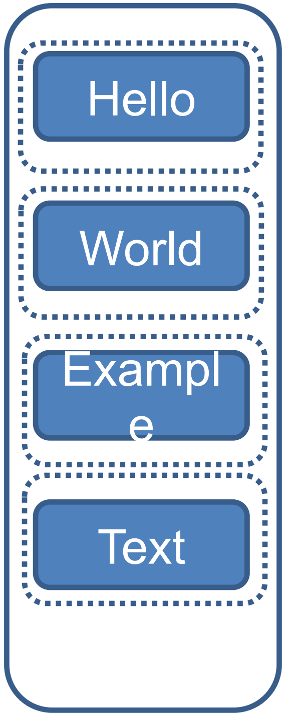
Recognizing this problem, researchers have developed specialized frameworks for some applications that require data reuse. For example, Pregel [22] is a system for iterative graph computations that keeps intermediate data in memory, while HaLoop [7] offers an iterative MapReduce interface. However, these frameworks only support specific computation patterns (*e.g.,* looping a series of MapReduce steps), and perform data sharing implicitly for these patterns. They do not provide abstractions for more general reuse, *e.g.,* to let a user load several datasets into memory and run ad-hoc queries across them.

In this paper, we propose a new abstraction called *resilient distributed datasets (RDDs)* that enables efficient data reuse in a broad range of applications. RDDs are fault-tolerant, parallel data structures that let users ex-

Paper published in 2012

# RDD intro

RDD of Strings



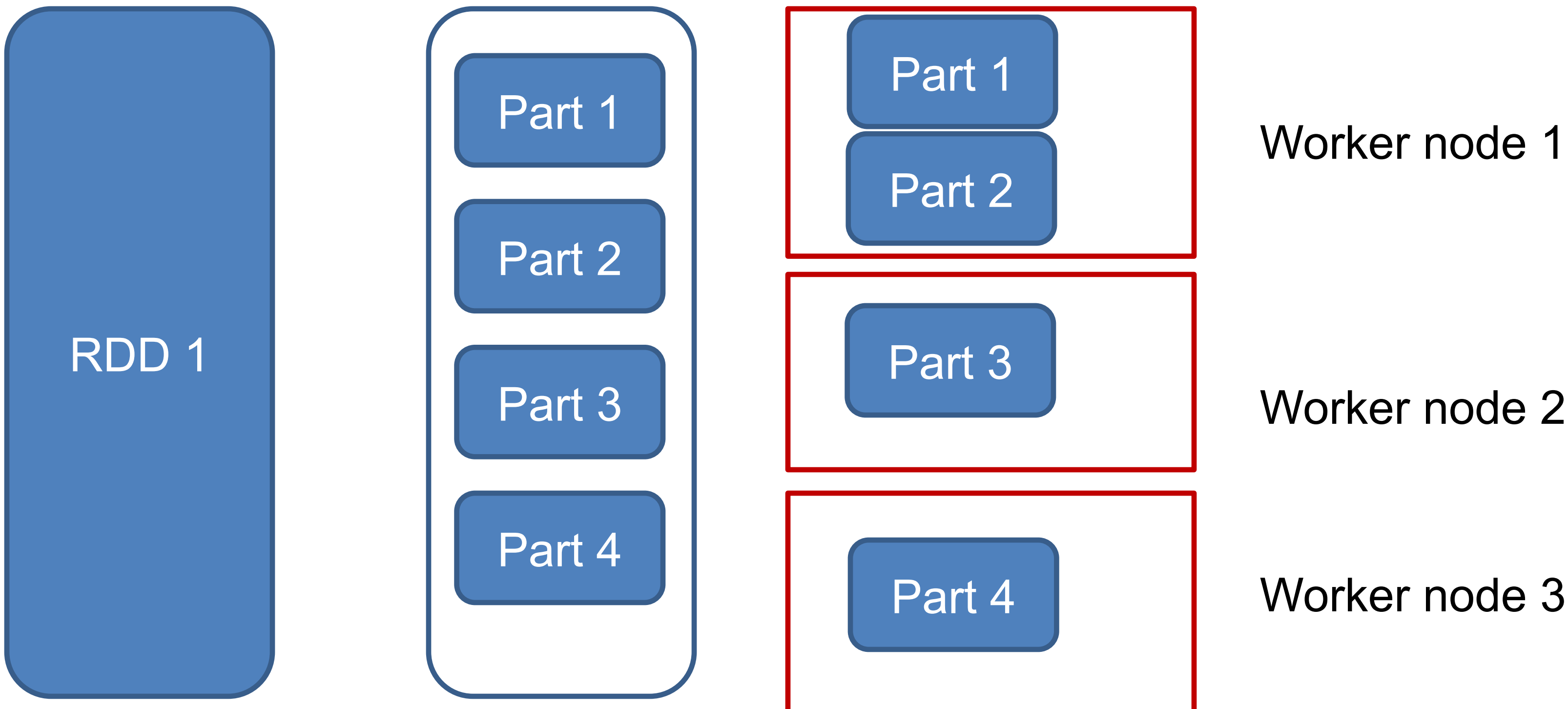

Immutable **Collection** of Objects

**Partitioned** and **Distributed**

Stored in **Memory**

Created **lazily**

# How RDD is stored as a distributed collection!




This is an RDD looks to you.

RDD in 4 partitions

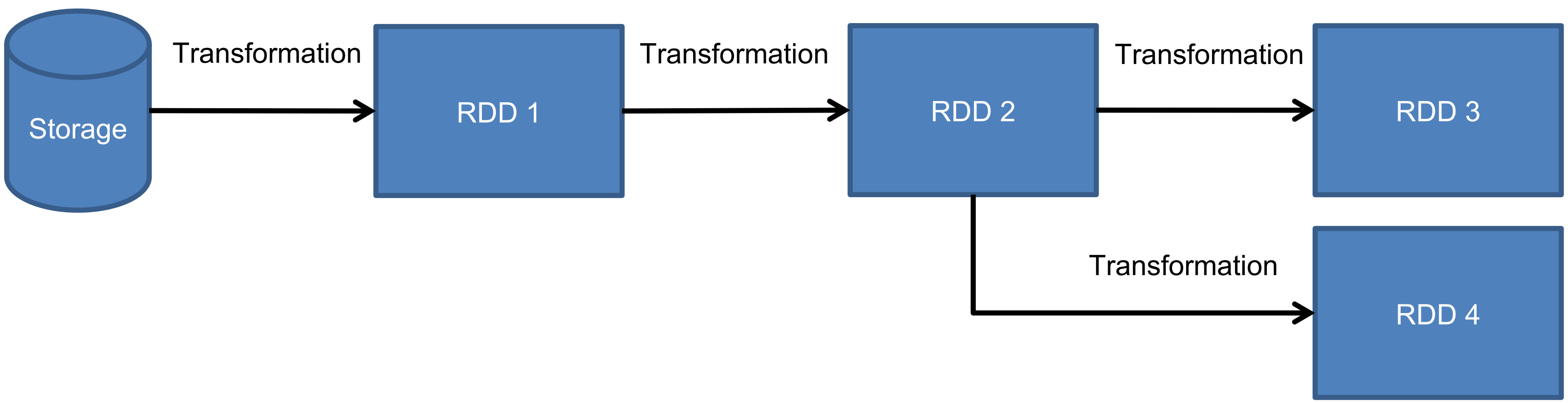
Distribution in cluster

# RDD intro (continued..)

- RDD can only be created through **deterministic** operations on either
  - data from stable storage
  - other RDDs
- RDD has enough information about how it was derived from other datasets (its **lineage**) to compute its partitions from data in stable storage.

UNIVERSITAT DE BARCELONA

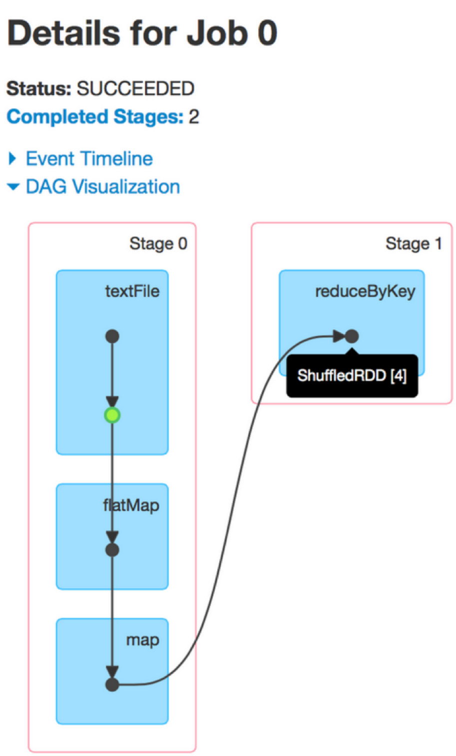
# Example of RDD lieange




The information of the lineage of an RDD is saved as a **DAG** (directed acyclic graph
RDD when needed or re-create RDD partitions in case of a failure of a node.

UNIVERSITAT DE BARCELONA

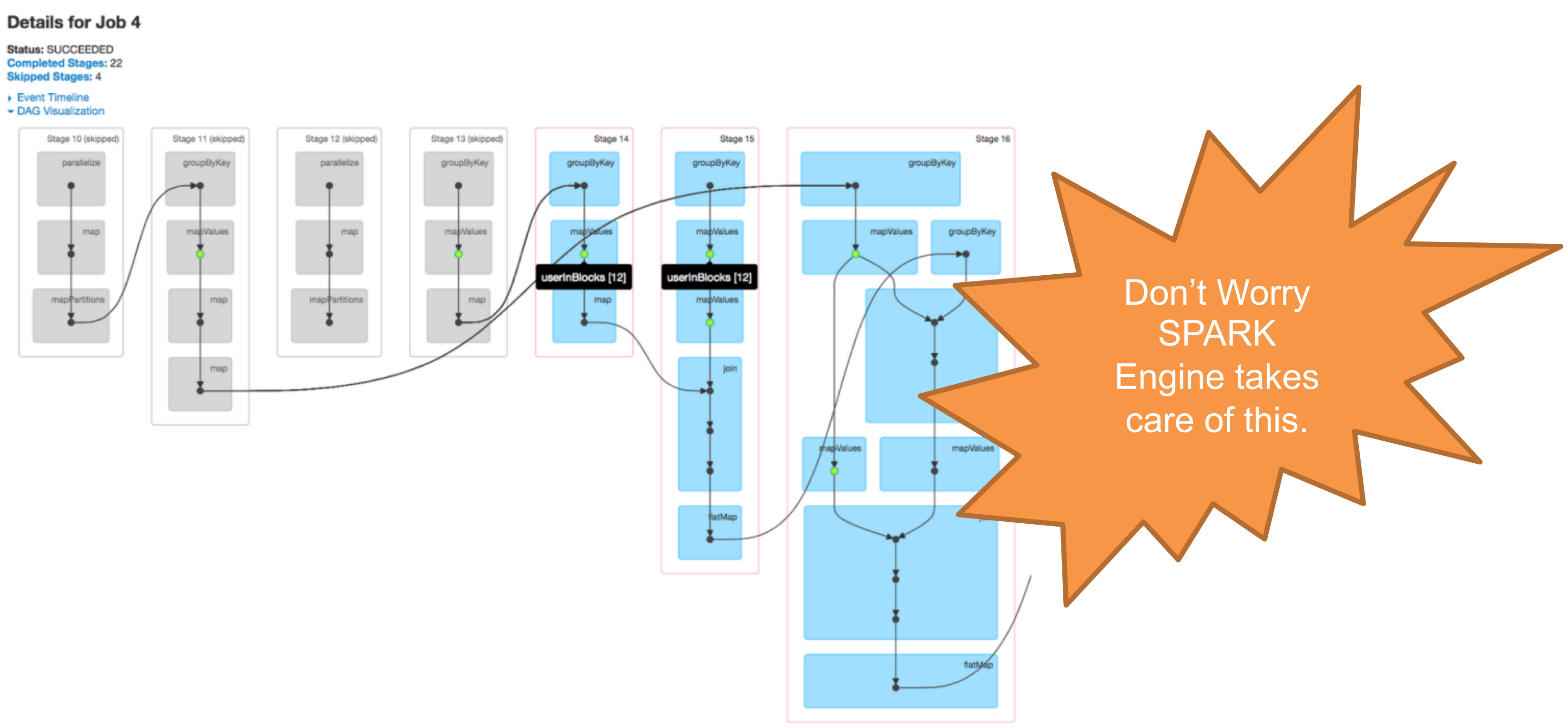
# DAG demystified



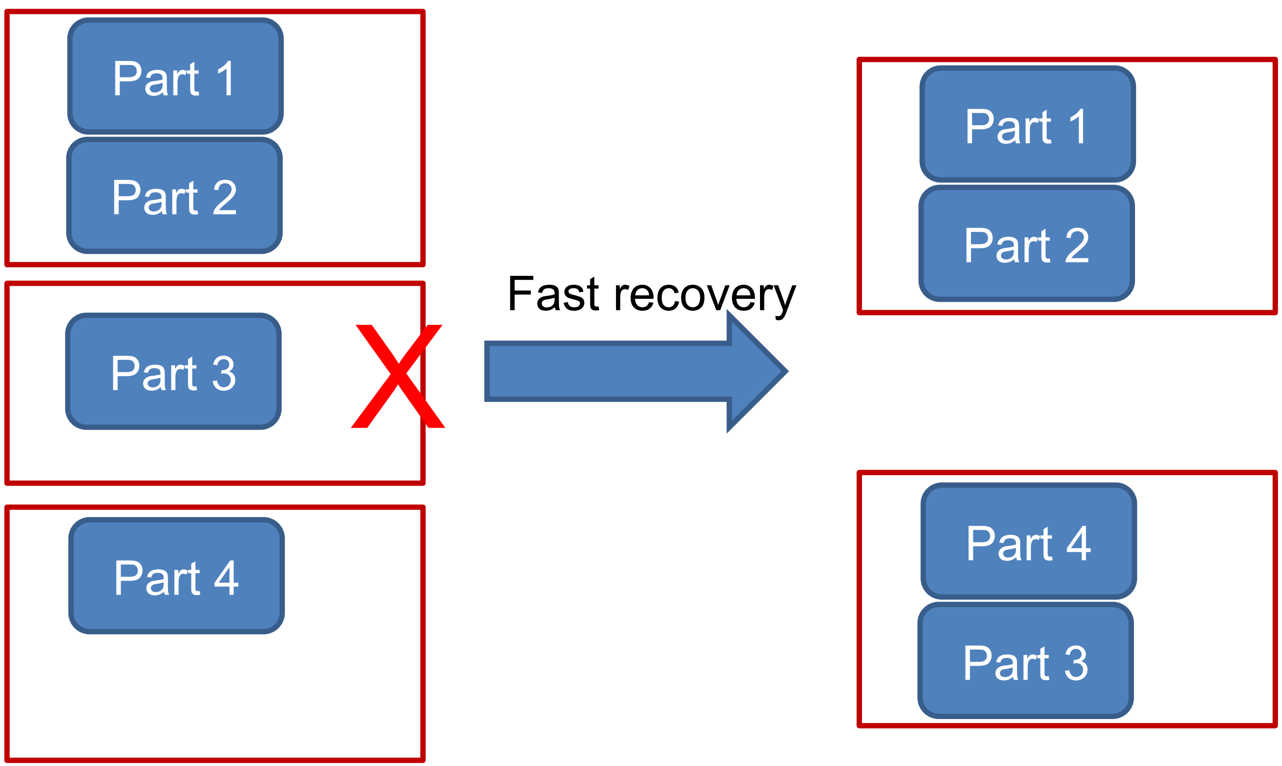

- In theory DAG is a directed graph with no cycles.
- DAG (Directed Acyclic Graph) is a programming style for distributed systems.
- While MapReduce has two steps (map and reduce), DAG can have multiple levels that can form a tree structure.
- Apache TEZ which came after MapReduce implemented the same programing model.

TEZ

# Some Complex DAGs

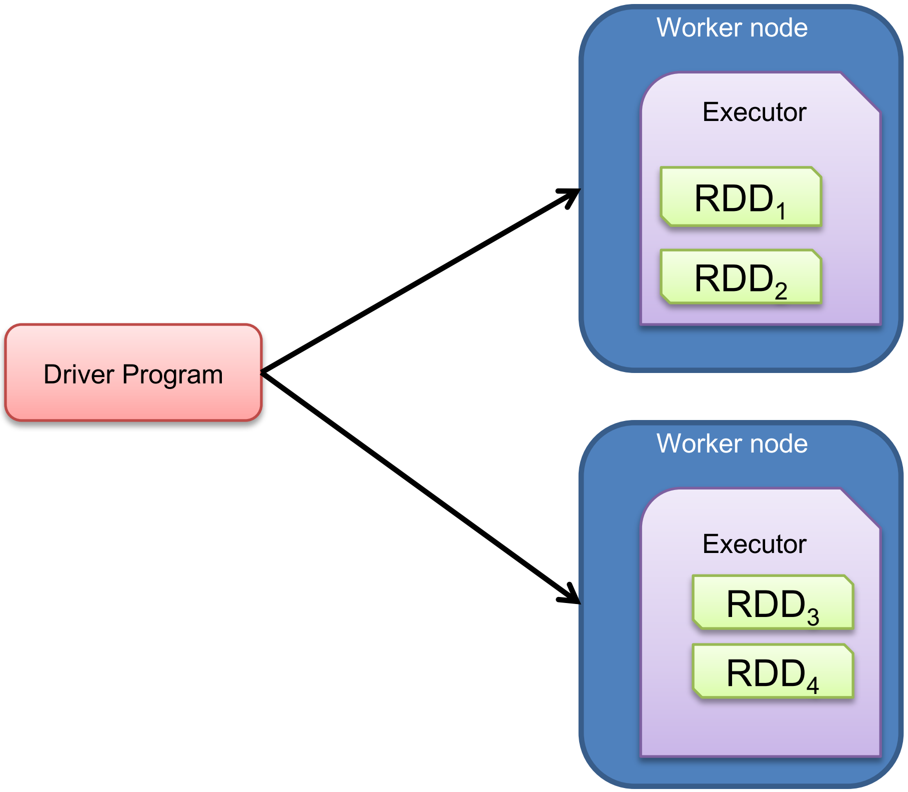


# RDD (fault tolerance)



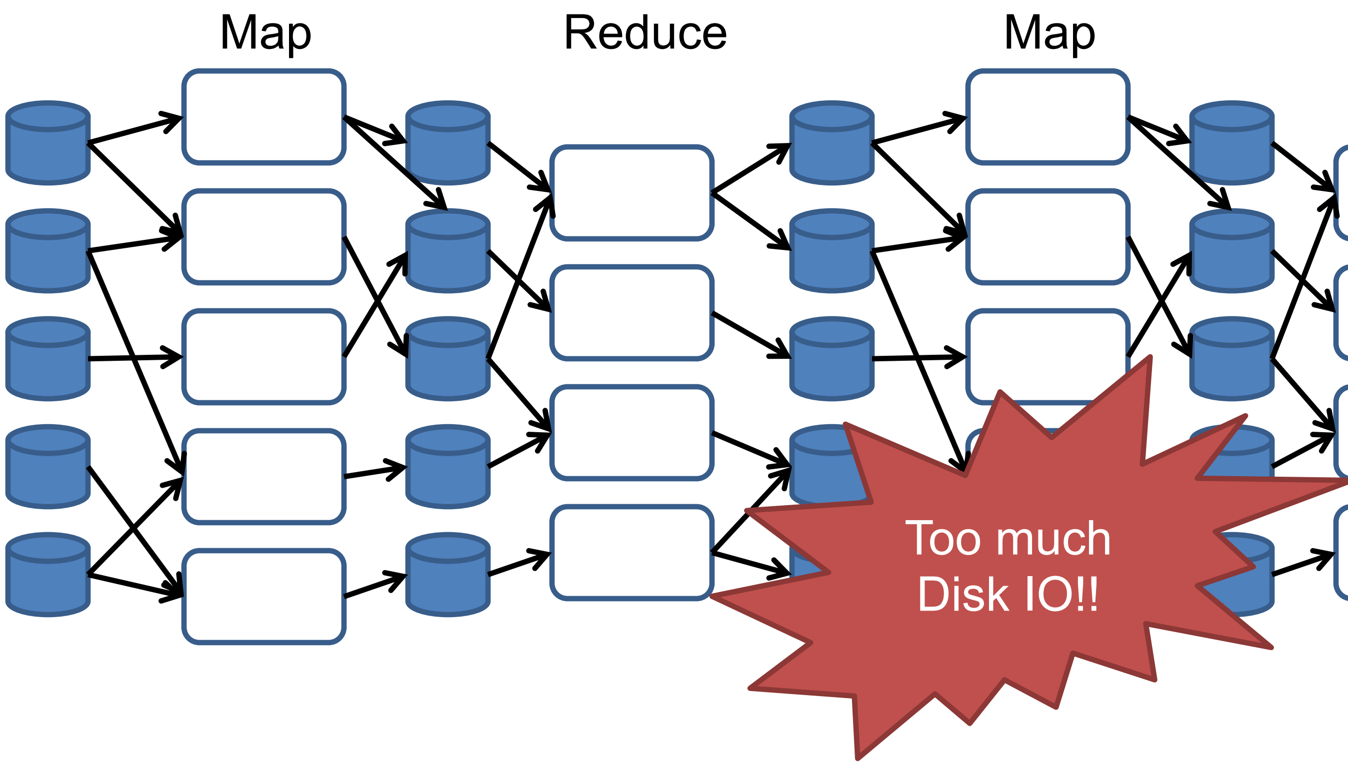

In case of failure: suppose the worker node 2 having the partition 3 crashes then only the lost RDDs need to be recreated and it can be easily done using the lineage information.

# RDD (parallelism)

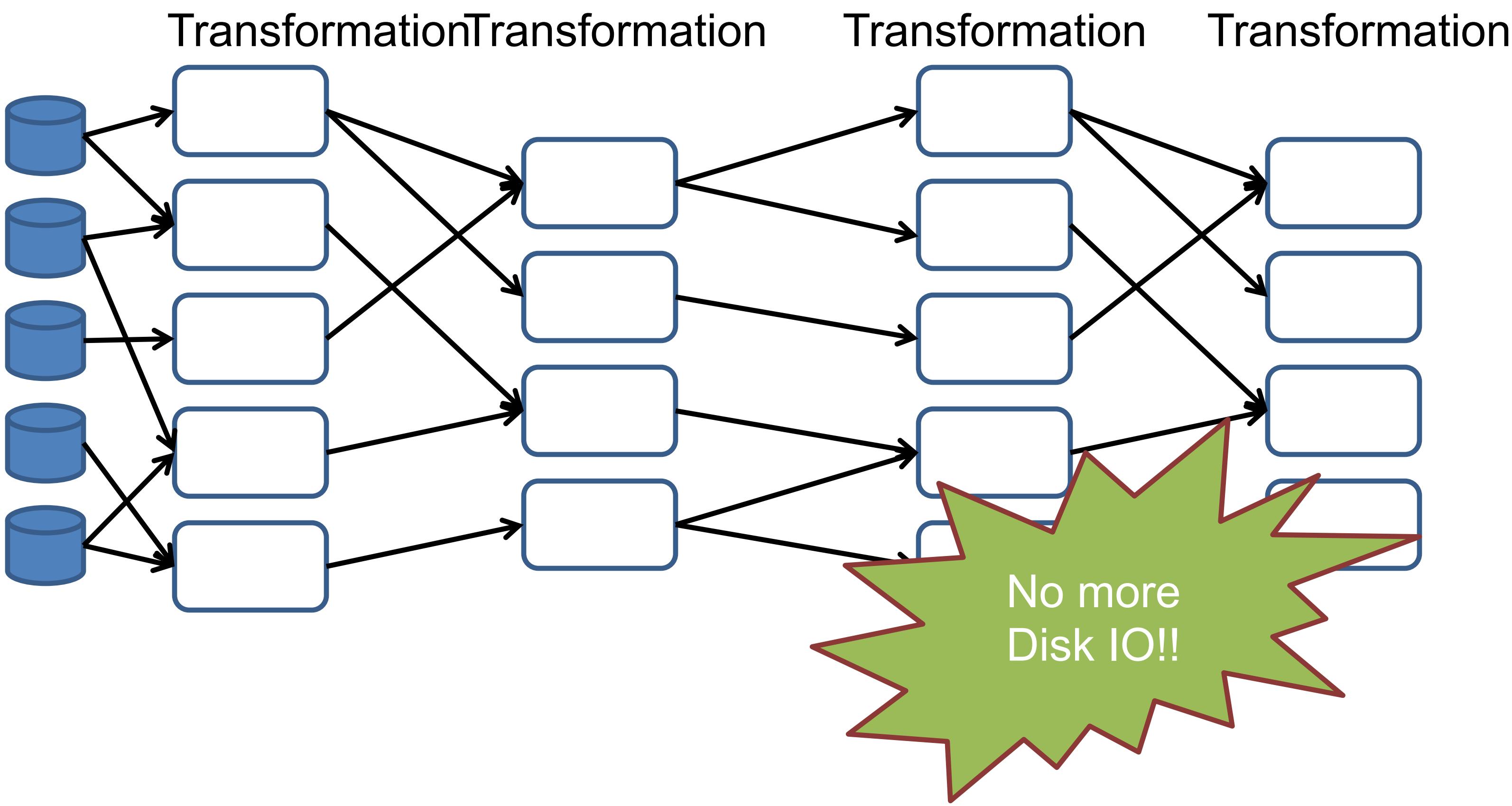



More Partition == More parallelism

# Hadoop



# Spark


Transformation
Transformation
Transformation
Transformation
No more
Disk IO!!

# RDD (continued..)

An RDD can be created by 2 ways

I. Parallelize a collection

II. Read data from external source

# Parallelize

- Take an existing in-memory collection and pass it to Spark Context's parallelize method.
- The in-memory collection is created at the driver program and then the data is partitioned and shipped to the executors in the worker node.
- Used mostly for prototyping and not in production.
- Example: fruitRDD= sc.parallelize([“apple", “banana“, “mango”])

UNIVERSITAT DE BARCELONA

# External data source

- Read from a textFile
  - localRDD= sc.textFile("/path_to_file")
  - hdfsRDD = sc.textFile("hdfs://...")
- Specific APIs for different data source has been developed.

# Types of RDD

There are many types of RDDs:

- ParallelCollectionRDD: is the result of SparkContext.parallelize
- HadoopRDD: is an RDD that provides core functionality for reading data stored in HDFS using the older MapReduce API. The most notable use case is the return RDD of SparkContext.textFile.
- MapPartitionsRDD: a result of calling operations like map, flatMap, filter, mapPartitions, etc.
- CoalescedRDD: a result of repartition or coalesce transformations.
- ShuffledRDD: a result of shuffling, e.g. after repartition or coalesce transformations.
- PairRDD (implicit conversion by PairRDDFunctions) that is an RDD of key-value pairs that is a result of groupByKey and join operations.
- ..

# RDD interface

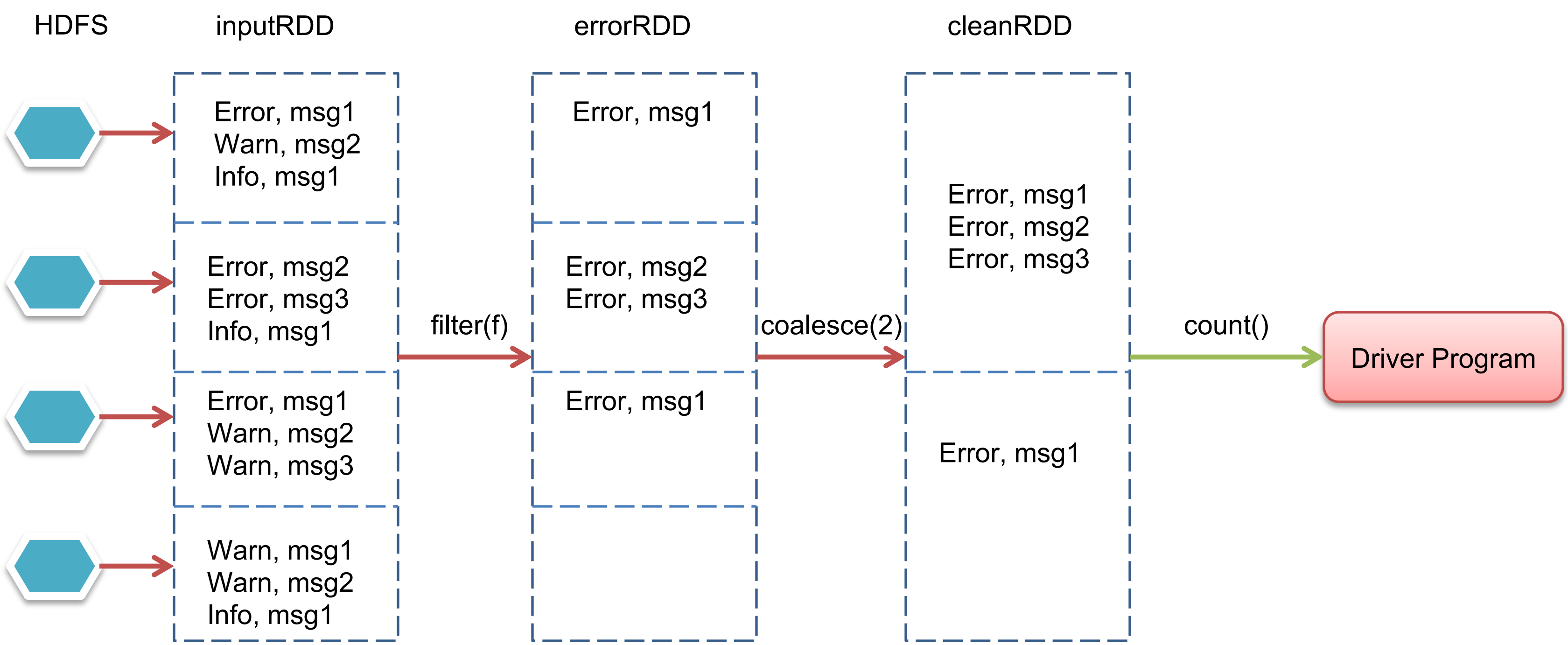
An RDD has 5 main attributes:

1. Set of partitions: Atomic pieces of data
2. List of dependencies on parent RDDs
3. Function to compute a partition given parents
4. Preferred location (optional metadata)
5. Partitioning information.

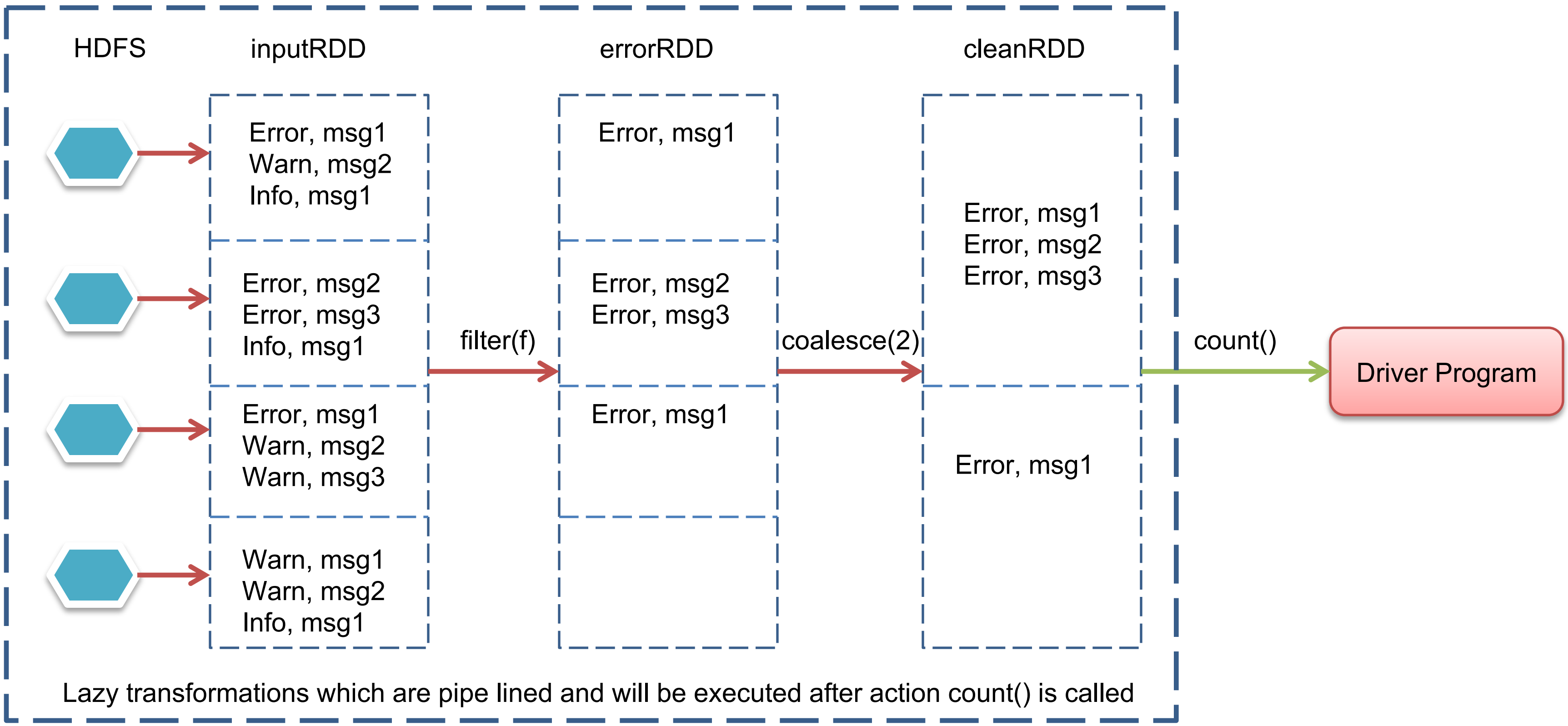
# RDD transformations

- RDD transformation are **lazy** i.e. when a transformation happens only the lineage is updated and no physical execution happens.
- Only after an **action** is called the whole thing plays out.

# RDD transformations



# RDD transformations



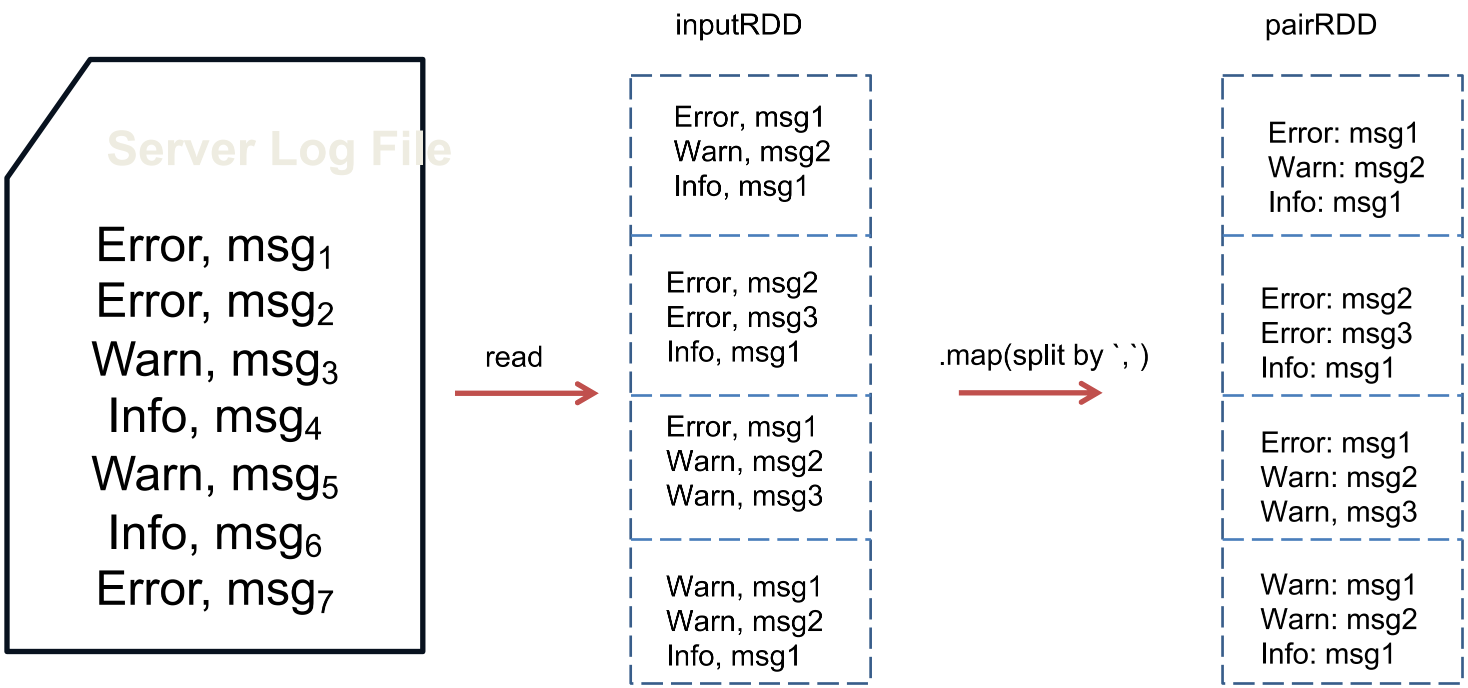
# Simple RDD transformations

| Transformation | Meaning |
|---|---|
| map(func) | Return a new distributed dataset formed by passing each element of the source through a function func. |
| filter(func) | Return a new dataset formed by selecting those elements of the source on which func returns true. |
| flatMap(func) | Similar to map, but each input item can be mapped to 0 or more output items (so func should return a Seq rather than a single item). |
| mapPartitions(func) | Similar to map, but runs separately on each partition (block) of the RDD, so func must be of type Iterator<T> => Iterator<U> when running on an RDD of type T. |
| sample(withReplacement, fraction, seed) | Sample a fraction fraction of the data, with or without replacement, using a given random number generator seed. |
| union(otherDataset) | Return a new dataset that contains the union of the elements in the source dataset and the argument. |
| intersection(otherDataset) | Return a new RDD that contains the intersection of elements in the source dataset and the argument. |
| distinct([numTasks])) | Return a new dataset that contains the distinct elements of the source dataset. |

# Key Value : pairRDD

“Spark provides special type of operations on RDDs containing key/value pairs. These RDDs are called pair RDDs. Pair RDDs are very useful in data analytical tasks, as they provide operations that allows to act on each key operations in parallel or regroup data across the network.”

# Key Value : pairRDD



# Transformations on pairRDD

| Transformation | Meaning |
|---|---|
| groupByKey([numTasks]) | When called on a dataset of (K, V) pairs, returns a dataset of (K, Iterable<V>) pairs. |
| reduceByKey(func, [numTasks]) | When called on a dataset of (K, V) pairs, returns a dataset of (K, V) pairs where the values for each key are aggregated using the given reduce function func, which must be of type (V,V) => V. Like in groupByKey, the number of reduce tasks is configurable through an optional second argument. |
| aggregateByKey(zeroValue)(seqOp, combOp, [numTasks]) | When called on a dataset of (K, V) pairs, returns a dataset of (K, U) pairs where the values for each key are aggregated using the given combine functions and a neutral "zero" value. Allows an aggregated value type that is different than the input value type, while avoiding unnecessary allocations. Like in groupByKey, the number of reduce tasks is configurable through an optional second argument. |
| sortByKey([ascending], [numTasks]) | When called on a dataset of (K, V) pairs where K implements Ordered, returns a dataset of (K, V) pairs sorted by keys in ascending or descending order, as specified in the boolean ascending argument. |
| join(otherDataset, [numTasks]) | When called on datasets of type (K, V) and (K, W), returns a dataset of (K, (V, W)) pairs with all pairs of elements for each key. Outer joins are supported through leftOuterJoin, rightOuterJoin, and fullOuterJoin. |
| cogroup(otherDataset, [numTasks]) | When called on datasets of type (K, V) and (K, W), returns a dataset of (K, (Iterable<V>, Iterable<W>)) tuples. This operation is also called groupWith. |

# Transformation leading to repartition of data

| Transformation | Meaning |
|---|---|
| coalesce(numPartitions) | Decrease the number of partitions in the RDD to numPartitions. Useful for running operations more efficiently after filtering down a large dataset. |
| repartition(numPartitions) | Reshuffle the data in the RDD randomly to create either more or fewer partitions and balance it across them. This always shuffles all data over the network. |

**Note**: Repartitioning is necessary some time for better performance but should be carefully used.

UNIVERSITAT DE BARCELONA

# RDD Actions

| Transformation | Meaning |
|---|---|
| reduce(func) | Aggregate the elements of the dataset using a function func (which takes two arguments and returns one). The function should be commutative and associative so that it can be computed correctly in parallel. |
| collect() | Return all the elements of the dataset as an array at the driver program. This is usually useful after a filter or other operation that returns a sufficiently small subset of the data. |
| count() | Return the number of elements in the dataset. |
| first() | Return the first element of the dataset (similar to take(1)). |
| take(n) | Return an array with the first n elements of the dataset. |
| takeSample(withReplacement, num, [seed]) | Return an array with a random sample of num elements of the dataset, with or without replacement, optionally pre-specifying a random number generator seed. |
| takeOrdered(n, [ordering]) | Return the first n elements of the RDD using either their natural order or a custom comparator. |

UNIVERSITAT DE BARCELONA

# RDD Actions

| Transformation | Meaning |
|---|---|
| saveAsTextFile(path) | Write the elements of the dataset as a text file (or set of text files) in a given directory in the local filesystem, HDFS or any other Hadoop-supported file system. Spark will call toString on each element to convert it to a line of text in the file. |
| countByKey() | Only available on RDDs of type (K, V). Returns a hashmap of (K, Int) pairs with the count of each key. |
| foreach(func) | Run a function func on each element of the dataset. This is usually done for side effects such as updating an Accumulator or interacting with external storage systems. |

# IPython Notebook

- Run the spark jupyter notebook using the image jupyter/all-spark-notebook

- Try the notebook Lab1

UNIVERSITAT DE BARCELONA

# References

- Spark in Action
- https://spark.apache.org/docs/latest/programming-guide.html
- https://spark.apache.org/docs/latest/cluster-overview.html
- Spark: Cluster Computing with Working Sets
- Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing