TP APACHE Spark

Data management for Big Data

ESIR
ECOLE SUPERIEURE
D'INGENIEURS DE RENNES

UNIVERSITÉ DE RENNES 1

Maria Massri
maria.massri@irisa.fr

Séances

Première séance (2h):

- Cours: Spark.
- Installation Spark.
- Analyse d'un jeu de données PUBG.

Deuxième séance (2h):

- Suite des travaux de la première séance.
- Dépôt du code sur gitlab.

Troisième séance (4h):

- Cours: GraphX (Librairie de Spark).
- Analyse d'un graphe de trajets de vélos (CityBike).
- Dépôt du code sur gitlab.

Quatrième séance (4h):

- Mini-projet: Traitement d'un jeu de données de votre choix.
- Rédaction du compte rendu et dépôt du code sur gitlab.

(Travail en binôme)

What is Spark?

- Apache Spark is a **data processing framework** that can quickly perform processing tasks on very large data sets, and can also **distribute data processing tasks** across multiple servers.

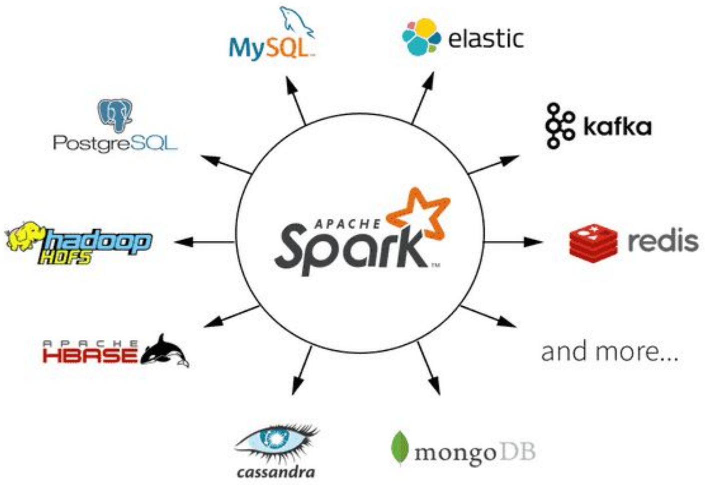
- Spark is used by banks, telecommunication companies, games companies, governments, and most of tech giants (e.g., Amazon, IBM, Microsoft, Samsung, TripAdvisor)

Parallel computations - Scalability	Large volume of structures/ semi structured data
Real time or archived data processing	Built-in data analysis operators

Data sources

Spark can process data from:

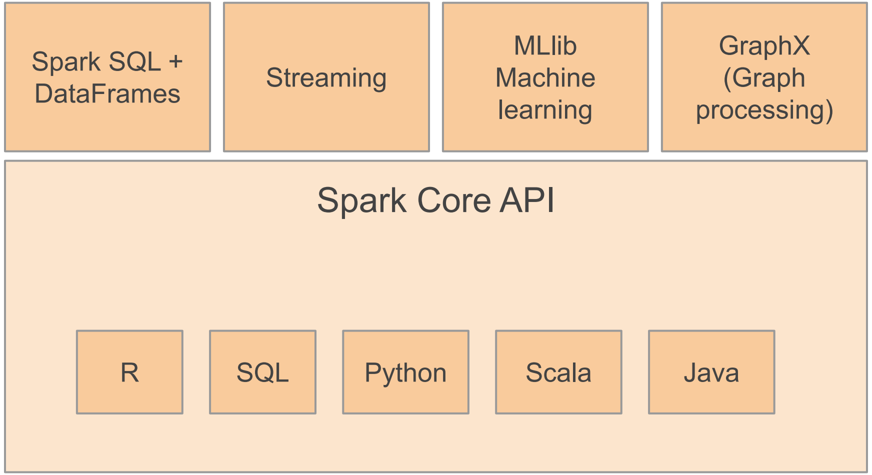
- Various data management/storage systems, including HDFS, Hive, Cassandra, MongoDB or Kafka.
- Unstructured data files such as .txt or .csv files.
- Semi-structured data files such as JSON or XML files.



Spark ecosystem

Spark can be deployed in a variety of ways:

- Providing native bindings for several programming languages.
- Supporting SQL, streaming data, machine learning, and graph processing.


Spark SQL + DataFrames

Streaming

MLlib Machine learning

GraphX (Graph processing)

Spark Core API

R

SQL

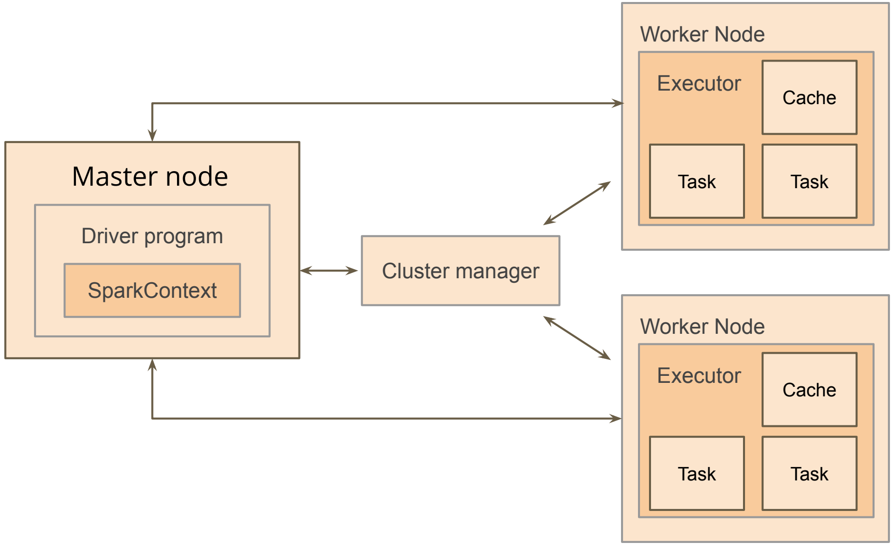
Python

Scala

Java

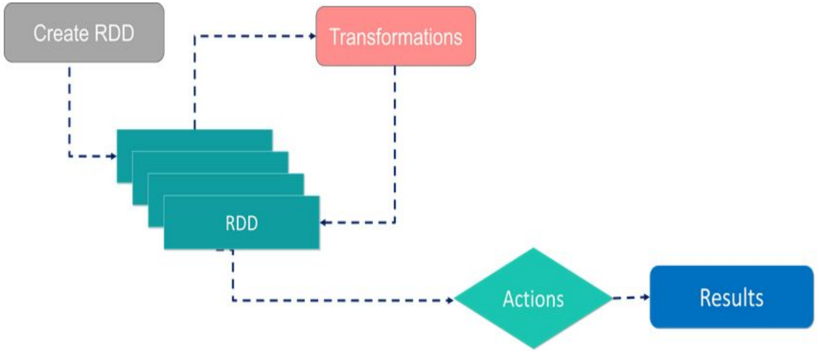
Spark architecture

- **Driver program** is the program you wrote.
- **Spark context** is the gateway to all Spark functionalities.
- A **job** is split into multiple stages.
- A **stage** is split into multiple tasks which are distributed over the worker nodes.
- **Tasks** of a single stage perform the same thing but runs each on a different partition.
- Worker nodes execute the tasks.



Resilient Distributed Datasets (RDD)

- RDD Stands for:
  - **Resilient**: Fault tolerant and is capable of rebuilding data on failure.
  - **Distributed**: Distributed data among the multiple nodes of a cluster.
  - **Dataset**: Collection of partitioned data with values.



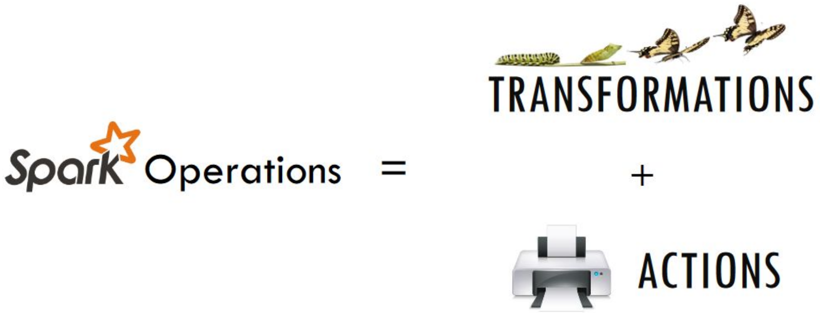
- RDDs are **immutable** data structures and follows a **lazy transformation**.

Spark operations

- Operations
- Transformations
- Actions

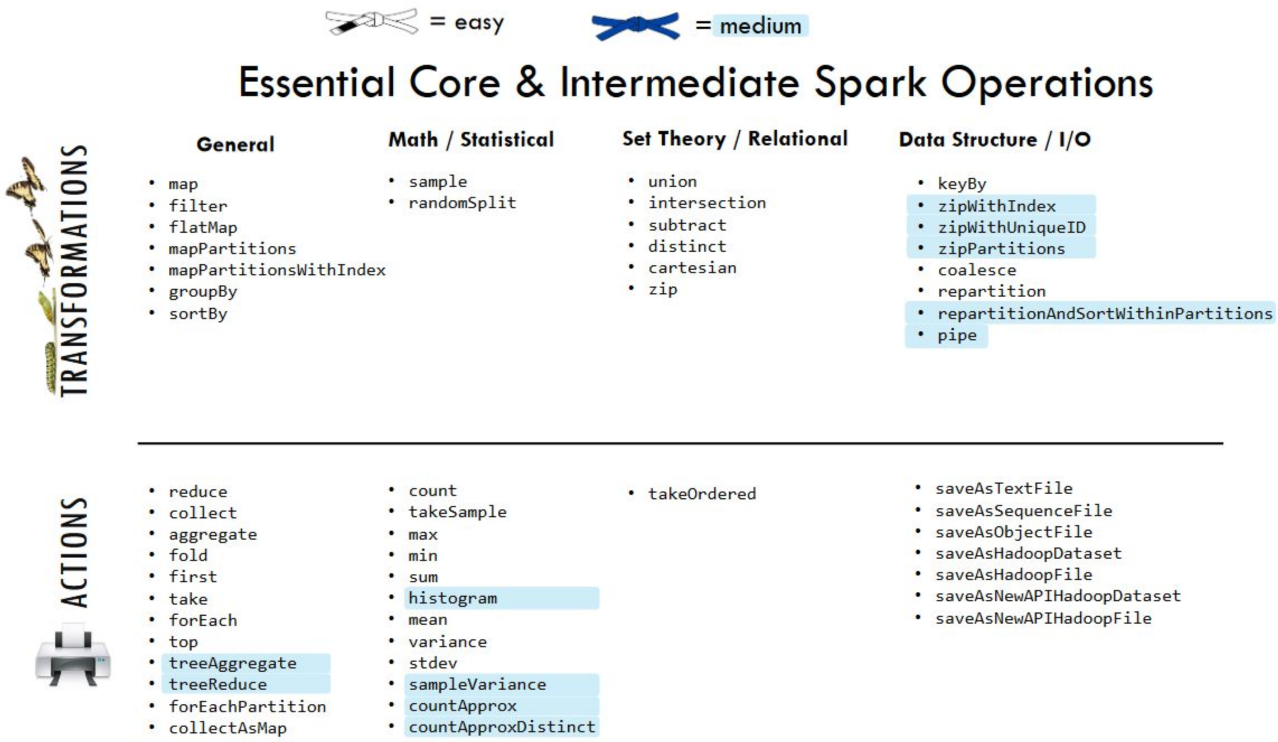
Operations

- **Transformations:**
  - Take an RDD and apply a given function on it and return a new RDD.
- **Actions:**
  - Take an RDD and apply a given function on it to return a single result.



Spark Operations = TRANSFORMATIONS + ACTIONS

Operations



= easy = medium

Essential Core & Intermediate Spark Operations

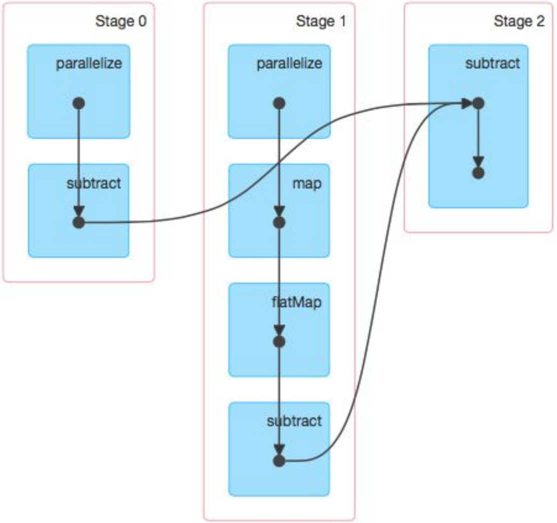
	General	Math / Statistical	Set Theory / Relational	Data Structure / I/O
TRANSFORMATIONS	map filter flatMap mapPartitions mapPartitionsWithIndex groupBy sortBy	sample randomSplit	union intersection subtract distinct cartesian zip	keyBy zipWithIndex zipWithUniqueID zipPartitions coalesce repartition repartitionAndSortWithinPartitions pipe
ACTIONS	reduce collect aggregate fold first take forEach top treeAggregate treeReduce forEachPartition collectAsMap	count takeSample max min sum histogram mean variance stdev sampleVariance countApprox countApproxDistinct	takeOrdered	saveAsTextFile saveAsSequenceFile saveAsObjectFile saveAsHadoopDataset saveAsHadoopFile saveAsNewAPIHadoopDataset saveAsNewAPIHadoopFile

Official documentation can be found here: https://spark.apache.org/docs/latest/

Operations: Transformation

- Spark Transformation is a function that produces new RDD from the existing RDDs.

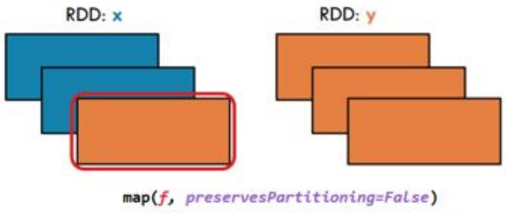
- **Lazy transformation:**
  - Transformation will not be performed until an action is called.
  - It will create a DAG (Directed Acyclic Graph) with all the parents of the RDD. And it will keep on building this graph till an action is called.



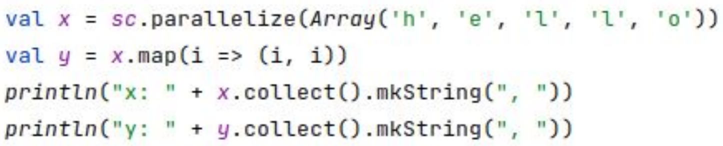
Transformation: Map

- The **Map** takes a function, and applies it to every element of RDD.

> The input and the return type of RDD may differ from each other.

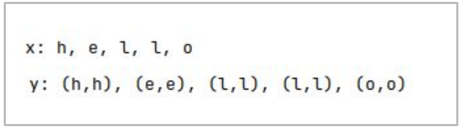


Scala



```
val x = sc.parallelize(Array('h', 'e', 'l', 'l', 'o'))
val y = x.map(i => (i, i))
println("x: " + x.collect().mkString(", "))
println("y: " + y.collect().mkString(", "))
```

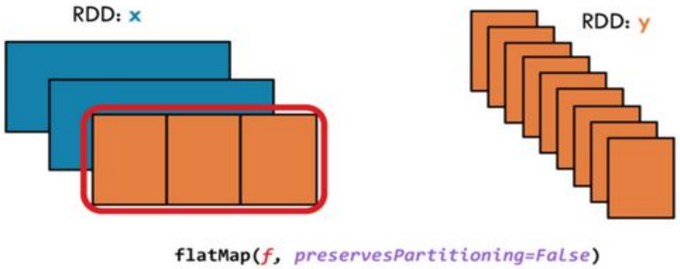
Output



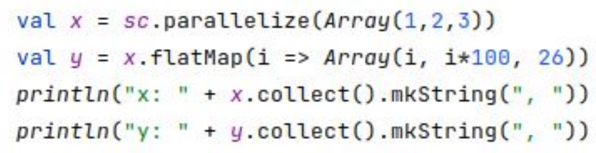
```
x: h, e, l, l, o
y: (h,h), (e,e), (l,l), (l,l), (o,o)
```

Transformation: FlatMap

- The **FlatMap** is similar to map but each element from input RDD can be mapped to zero or more output elements.

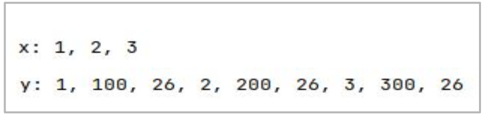


Scala



```scala
val x = sc.parallelize(Array(1,2,3))
val y = x.flatMap(i => Array(i, i*100, 26))
println("x: " + x.collect().mkString(", "))
println("y: " + y.collect().mkString(", "))
```

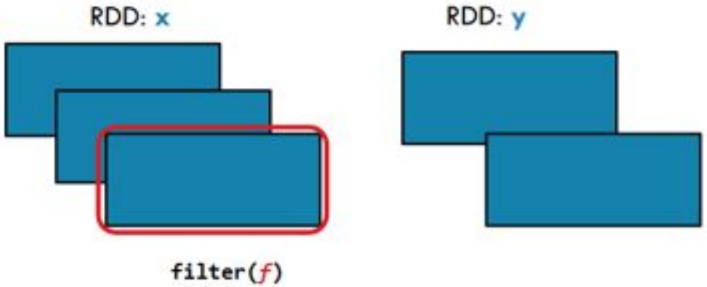
Output



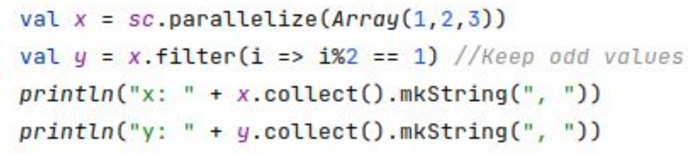
```
x: 1, 2, 3
y: 1, 100, 26, 2, 200, 26, 3, 300, 26
```

Transformation: Filter

- The **Filter** transformation returns a new RDD that's formed by selecting those elements of the source RDD on which the function returns true.

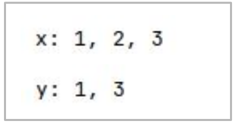


Scala



```
val x = sc.parallelize(Array(1,2,3))
val y = x.filter(i => i%2 == 1) //Keep odd values
println("x: " + x.collect().mkString(", "))
println("y: " + y.collect().mkString(", "))
```

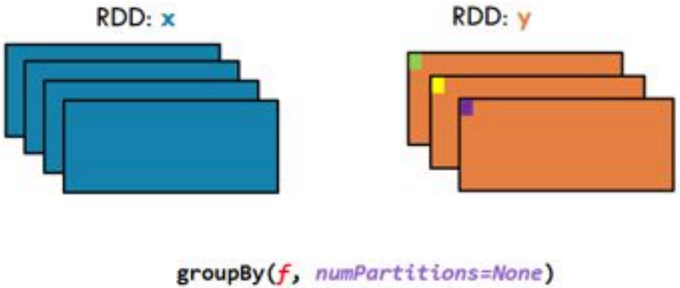
Output



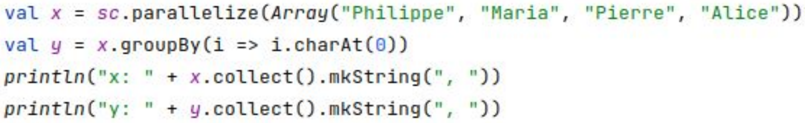
```
x: 1, 2, 3

y: 1, 3
```

Transformation: GroupBy

- The **GroupBy** groups the data in the original RDD and create pairs where the key is the output of a user function, and the value is all items for which the function yields this key.

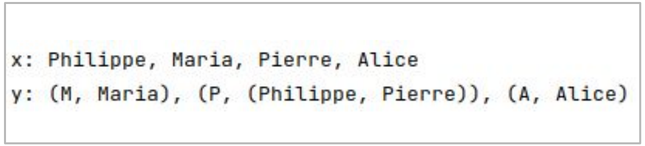


Scala



```
val x = sc.parallelize(Array("Philippe", "Maria", "Pierre", "Alice"))
val y = x.groupBy(i => i.charAt(0))
println("x: " + x.collect().mkString(", "))
println("y: " + y.collect().mkString(", "))
```

Output



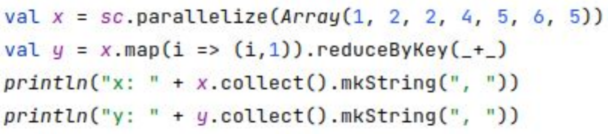
```
x: Philippe, Maria, Pierre, Alice
y: (M, Maria), (P, (Philippe, Pierre)), (A, Alice)
```

Transformation: ReduceByKey

- The **ReduceByKey** aggregates the values for each key using a given aggregation function.

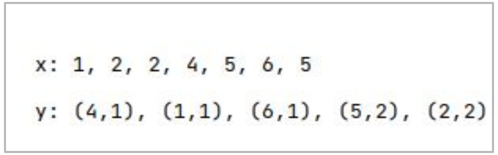
> - The aggregation function takes two values and returns one value.

Scala



```
val x = sc.parallelize(Array(1, 2, 2, 4, 5, 6, 5))
val y = x.map(i => (i,1)).reduceByKey(_+_)
println("x: " + x.collect().mkString(", "))
println("y: " + y.collect().mkString(", "))
```

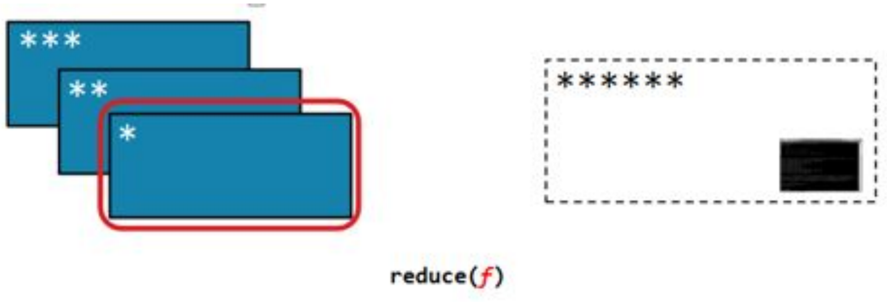
Output



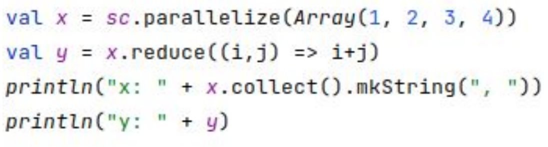
```
x: 1, 2, 2, 4, 5, 6, 5
y: (4,1), (1,1), (6,1), (5,2), (2,2)
```

Action: Reduce

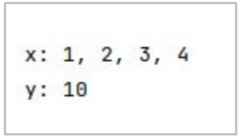
- The **Reduce** aggregates all the elements of the input RDD by applying a user function pairwise to elements and partial results until computing a single result.



Scala


```
val x = sc.parallelize(Array(1, 2, 3, 4))
val y = x.reduce((i,j) => i+j)
println("x: " + x.collect().mkString(", "))
println("y: " + y)
```

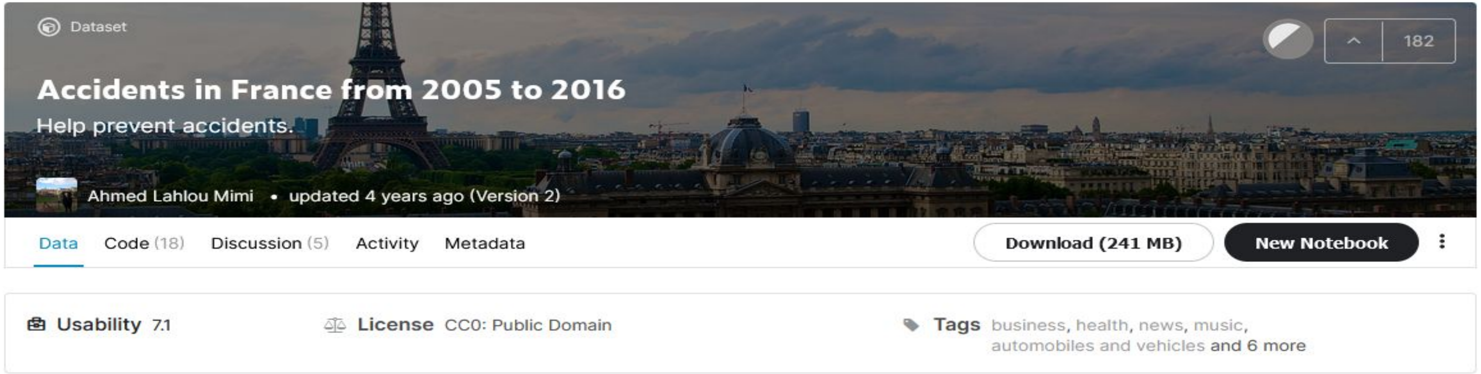
Output


```
x: 1, 2, 3, 4
y: 10
```

Template

- Real dataset of **Accidents** in France that includes for each accident:
  - Date
  - GPS coordinates
  - Departement
  - Atm: Atmosphere
- **Analysis**: Impact of the weather on the total number of accidents.

- **Optional**: You can practice Spark operations before starting the practical work by downloading *template_accidents* from Moodle and running the project in your IDE.



Dataset

Accidents in France from 2005 to 2016

Help prevent accidents.

Ahmed Lahlou Mimi • updated 4 years ago (Version 2)

Usability 7.1

License CC0: Public Domain

Tags business, health, news, music, automobiles and vehicles and 6 more

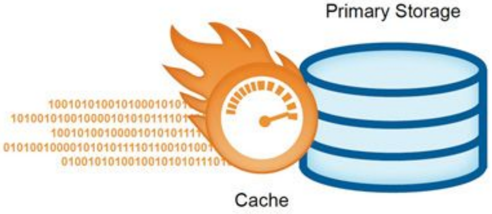
Advanced processing

- Persistence
- Accumulators
- Broadcast variables

Persistence

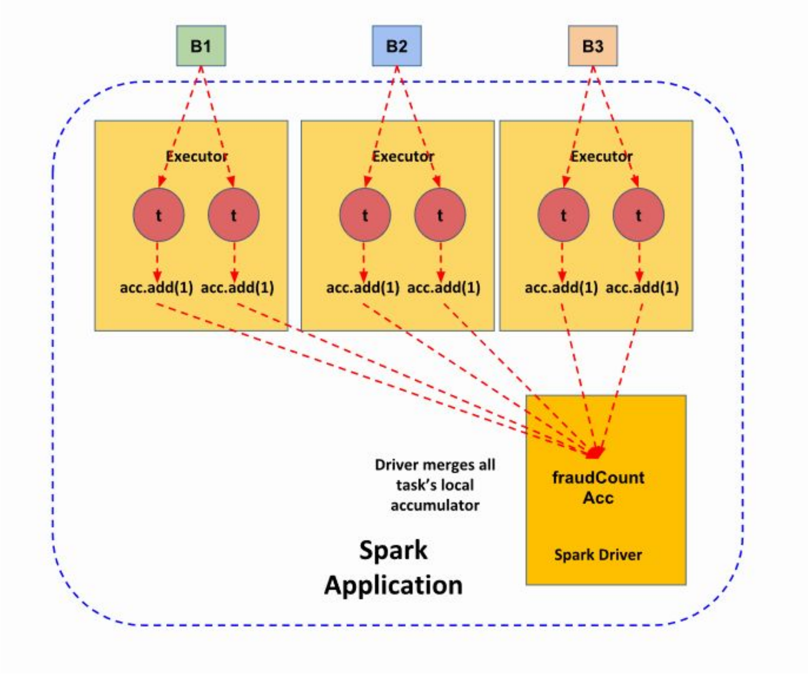
- Spark offers persist and RDD **in-memory**, **on-disk** or in a **Hybrid** fashion instead of creating the RDD whenever they are used by a new operation.
- Each nodes persists its own partitions.
- Persistence **accelerates** actions and is **fault-tolerant**.

Storage Level	Description
Memory-only	RDDs are stored as deserialized Java Objects. If the RDD does not fit in memory, uncached partitions will be re-computed on the fly. (Default)
Memory and disk	RDDs are stored as deserialized Java objects. If the RDD does not fit in memory, uncached partitions are stored on disk.
Disk-only	RDDs are stored only on disk.



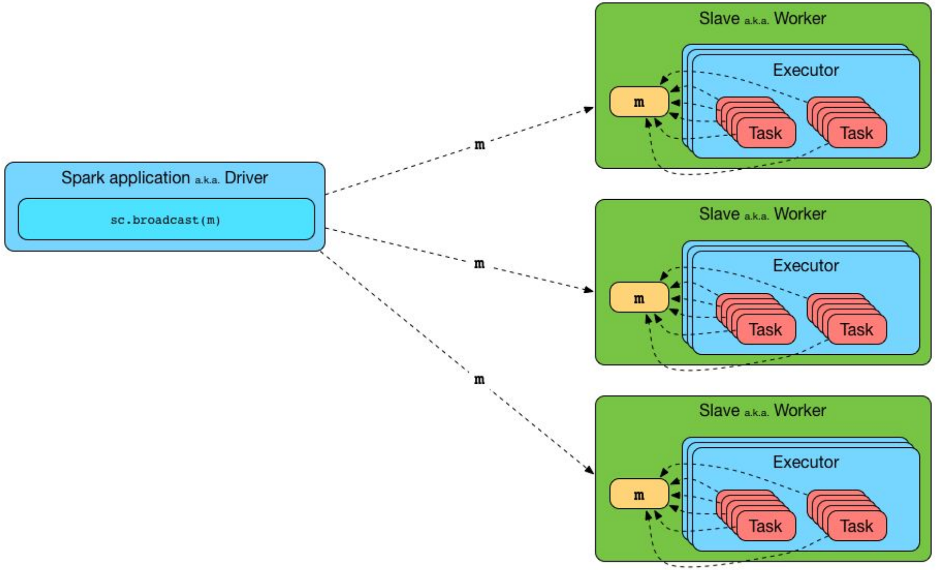
Accumulators

- Accumulators are used to count or sum up the values based on their occurrence.
- An accumulator is initialized by the spark driver
- Un local accumulateur will be created for each executor task.
- Each accumulated value will be transferred to the Spark Driver.
- The Spark Driver will then combine all the accumulated values to obtain the final result.



Broadcast variables

- A broadcast variable is READ-ONLY (immutable) and used to share data between the executors of different nodes.
- A copy of the shared variable is created by each executor.
- Broadcast variables are used in tasks in which a large data structure (Dictionaries, Arrays, etc.) or database sessions, for example, should be accessed .
- Suppose that we have 10 nodes and 1000 tasks s.t. each task should access a variable of 1 GB :
- Without broadcast variables: of traffic between driver node and nodes.
- With shared variables



TP SPARK

Data management for Big Data

ECOLE SUPERIEURE
D'INGENIEURS DE RENNES

UNIVERSITÉ DE RENNES 1

Maria Massri
maria.massri@irisa.fr