TEK-UP
Private Higher School of Technologies & Engineering

# Apache SPARK


tasnim.abar@tek-up.tn

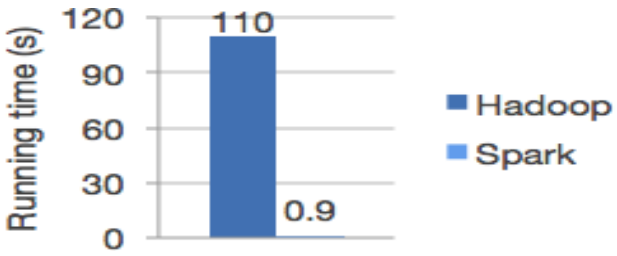
# Présentation

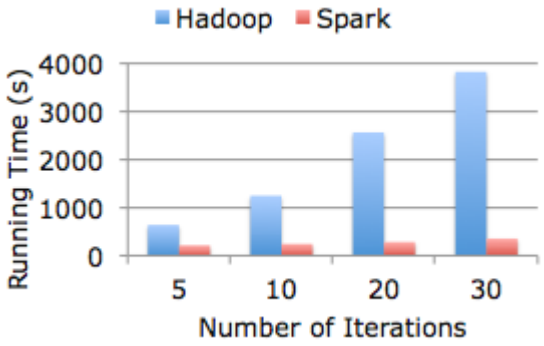
- Il s'agit d'un moteur de calcul Big Data rapide, polyvalent et évolutif basé sur la mémoire.
- En tant que solution unique, Apache Spark intègre le traitement par lots, la diffusion en continu en temps réel, les requêtes interactives, la programmation graphique et l'apprentissage automatique.
- Apache Spark a été développé dans le laboratoire AMP de l'UC Berkeley en 2009.
- Licence Apache.
- Traitement de large volume de données.
- Traitement distribué
- Ecrit en Scala

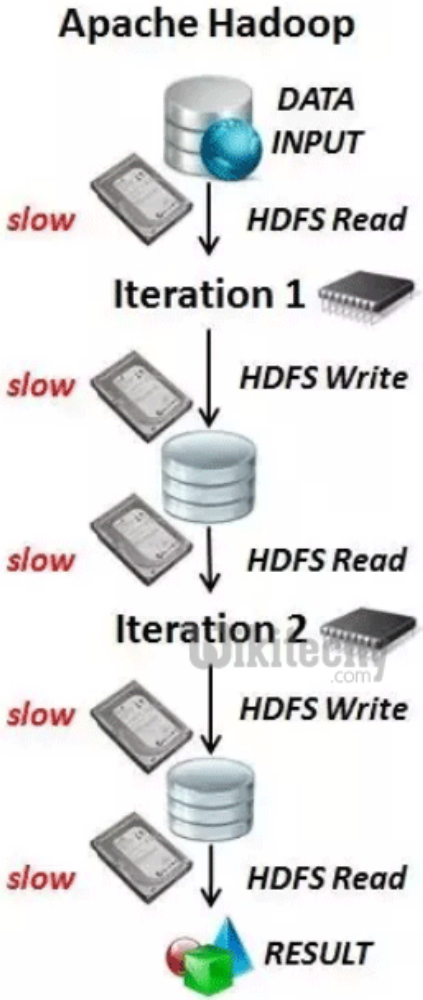
# Présentation

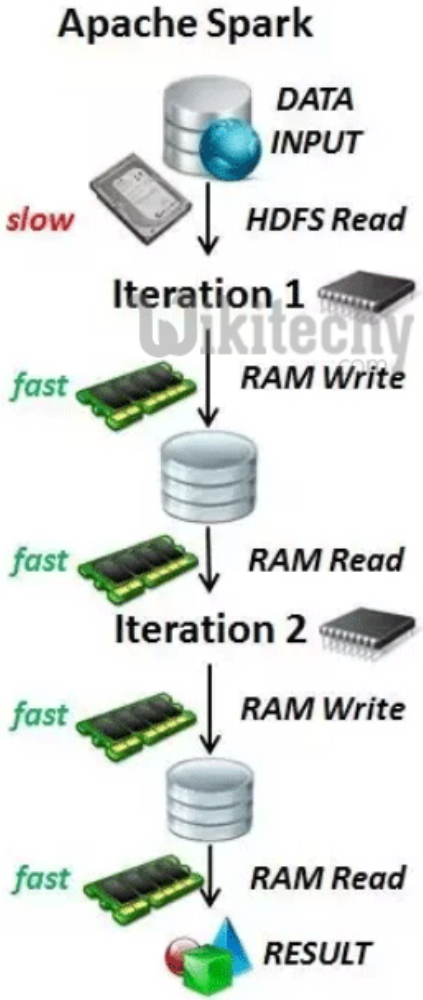
- Spark : <u>plus rapide:</u>

➢ Utilisation d'une mémoire Partagée RDD's 100x plus rapide, 10x moins d'itérations.



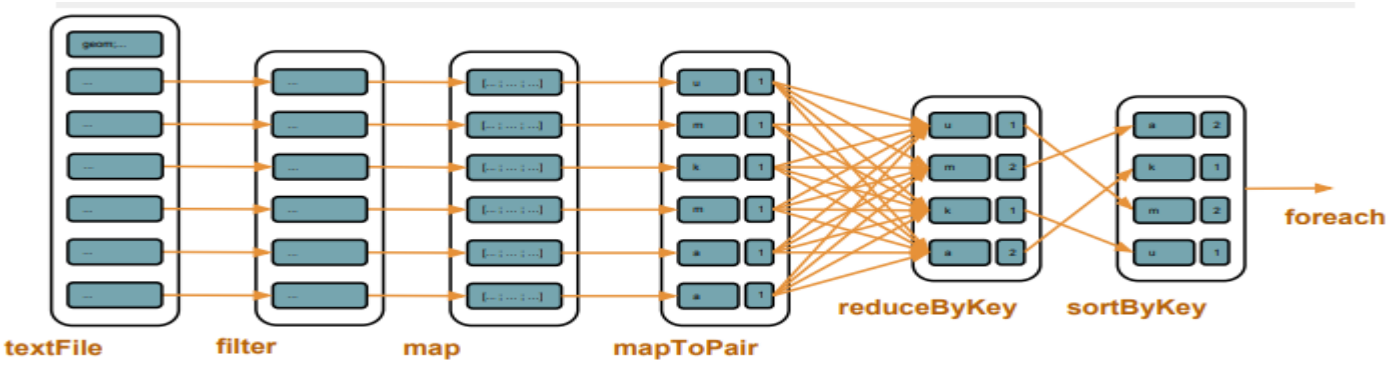
# Présentation

| | Hadoop | Spark | Spark |
|---|---|---|---|
| Data size | 102.5 TB | 102 TB | 1,000 TB |
| Consumed time (mins) | 72 | 23 | 234 |
| Nodes | 2,100 | 206 | 190 |
| Cores | 50,400 | 6,592 | 6,080 |
| Rate | 1.4 TB/min | 4.27 TB/min | 4.27 TB/min |
| Rate/node | 0.67 GB/min | 20.7 GB/min | 22.5 GB/min |
| Daytona Gray Sort | Yes | Yes | Yes |

# Présentation

- Spark : <u>facile à utiliser:</u>

➢ Spark est développé en Scala et supporte quatre langages: Scala, Java, Python (PySpark), R (SparkR).
➢ Une liste d'Operaters pour faciliter la manipulation des données au travers des RDD'S.
  ○ **map() :** une valeur → une valeur
  ○ **mapToPair() :** une valeur → un tuple
  ○ **filter() :** filtre les valeurs/tuples
  ○ **groupByKey() :** regroupe les valeurs par clés
  ○ **reduceByKey() :** aggrège les valeurs par clés
  ○ **join(), cogroup()... :** jointure entre deux RDD
  ○ **foreach() :** exécute une fonction sur chaque valeur/tuple



# Présentation

## Hadoop MapReduce

```
//package org.myorg;
import java.io.IOException;
import java.util.*;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapred.*;
import org.apache.hadoop.util.*;

public class WordCount {

        public static class Map extends MapReduceBase implements Mapper<LongWritable, Text,
Text, IntWritable> {
                private final static IntWritable one = new IntWritable(1);
                private Text word = new Text();

                public void map(LongWritable key, Text value, OutputCollector<Text,
IntWritable> output, Reporter reporter) throws IOException {
                        String line = value.toString();
                        StringTokenizer tokenizer = new StringTokenizer(line);
                        while (tokenizer.hasMoreTokens()) {
                                word.set(tokenizer.nextToken());
                                output.collect(word, one);
                        }
                }
        }

        public static class Reduce extends MapReduceBase implements Reducer<Text,
IntWritable, Text, IntWritable> {
                public void reduce(Text key, Iterator<IntWritable> values,
OutputCollector<Text, IntWritable> output, Reporter reporter) throws IOException {
                        int sum = 0;
                        while (values.hasNext()) {
                                sum += values.next().get();
                        }
                        output.collect(key, new IntWritable(sum));
                }
        }

        public static void main(String[] args) throws Exception {
                JobConf conf = new JobConf(WordCount.class);
                conf.setJobName("wordcount");

                conf.setOutputKeyClass(Text.class);
                conf.setOutputValueClass(IntWritable.class);

                conf.setMapperClass(Map.class);
                //conf.setCombinerClass(Reduce.class);
                conf.setReducerClass(Reduce.class);

                conf.setInputFormat(TextInputFormat.class);
                conf.setOutputFormat(TextOutputFormat.class);

                FileInputFormat.setInputPaths(conf, new Path(args[0]));
                FileOutputFormat.setOutputPath(conf, new Path(args[1]));

                JobClient.runJob(conf);
```

## Spark Scala

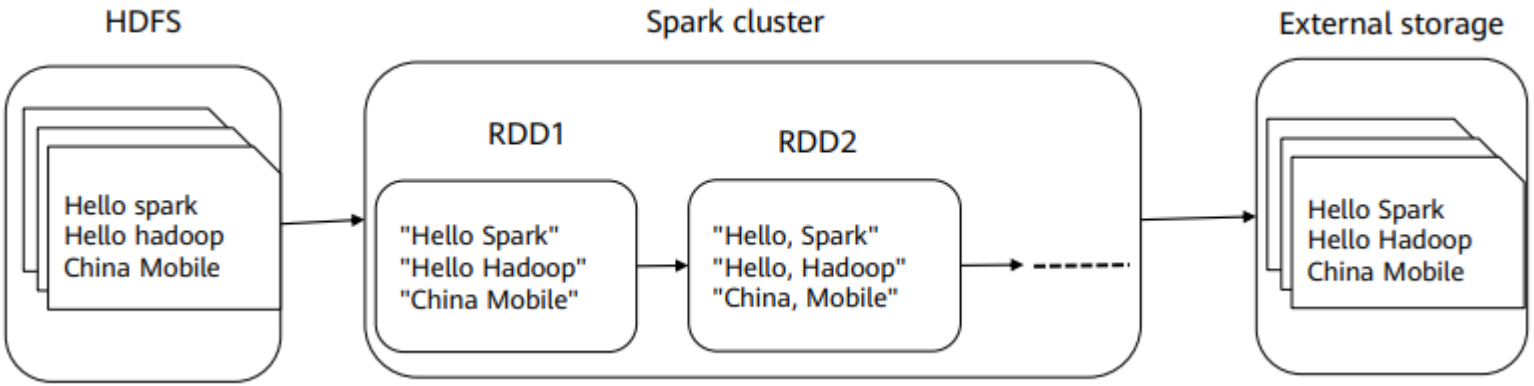
```
val file = spark.textFile("hdfs://...
val counts = file.flatMap(line => lin
                 .map(word => (word,
                 .reduceByKey(_ + _)
counts.saveAsTextFile("hdfs://...")
```

## Spark Python

```
file = spark.textFile("hdfs://...")
counts = file.flatMap(lambda line: line.split(
             .map(lambda word: (word, 1)) \
             .reduceByKey(lambda a, b: a + b)
counts.saveAsTextFile("hdfs://...")
```

# Structure de données Spark : RDD - Core Concept of Spark

- Les datasets distribués résilients (RDD) sont des datasets distribués élastiques, en lecture seule et partitionnés.
- Les RDD sont stockés dans la mémoire par défaut et sont écrits sur les disques lorsque la mémoire est insuffisante.
- Les données RDD sont stockées dans des clusters en tant que partitions.
- Les RDD ont un mécanisme de lignage, qui permet une récupération rapide des données en cas de perte de données.

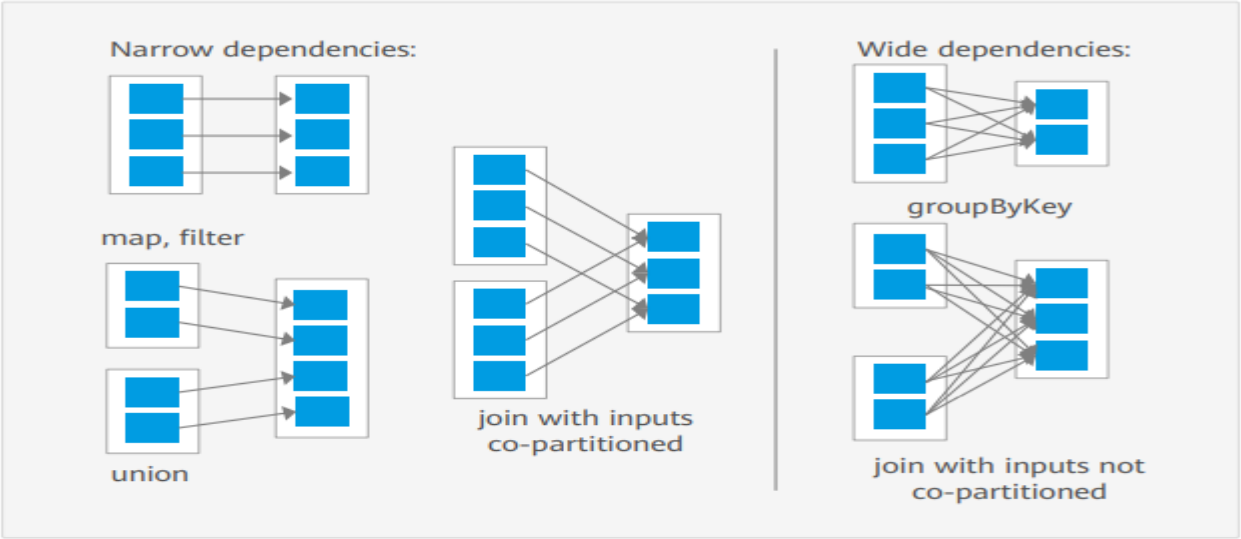


# Structure de données Spark : RDD - Core Concept of Spark

- **Narrow dependency** indique que chaque partition d'un RDD parent peut être utilisé par au plus une partition d'un RDD enfant, par exemple, map, filter et union.

- **Wide dependency** indique que les partitions de plusieurs RDD enfants dépendent de la partition du même RDD parent,  exemple, groupByKey, reduceByKey et sortByKey

RDD Dependencies



# Structure de données Spark : RDD - Core Concept of Spark

- Les opérations Spark peuvent être classées en opérations de création, de transformation, de contrôle et de action

- Opération de Création : permet de créer un RDD. Un RDD est créé par une collecte de mémoire et un système de stockage externe ou par une opération de transformation.

- Opération de Transformation : Un RDD est transformé en un nouveau RDD par certaines opérations. L'opération de transformation du RDD est une opération paresseuse, qui définit uniquement un nouveau RDD mais ne l'exécute pas immédiatement. (map(), filter(), reduceByKey(), join())

# Structure de données Spark : RDD - Core Concept of Spark

- Opération de Contrôle : la persistance RDD est effectuée. Un RDD peut être stocké sur le disque ou la mémoire en fonction de différentes politiques de stockage. Par exemple, l'API de cache met en cache le RDD dans la mémoire par défaut.

  Spark peut stocker RDD dans la mémoire ou le système de fichiers du disque de manière persistante. Le RDD en mémoire peut grandement améliorer le calcul itératif et le partage de données entre les modèles informatiques. Généralement, 60 % de la mémoire du nœud d'exécution est utilisé pour mettre en cache les données et les 40 % restants sont utilisés pour exécuter des tâches. Dans Spark, les opérations **persist()** et **cache()** sont utilisées pour la persistance des données.

- Opération d'Action : une opération qui peut déclencher l'exécution de Spark. Les opérations d'action dans Spark sont classées en deux types. L'une consiste à sortir le résultat du calcul et l'autre à enregistrer le RDD dans un système de fichiers ou une base de données externe. (reduce(), collect(), count(), first(), take())

# Structure de données Spark : RDD - Core Concept of Spark

**Les transformations**

| Transformation | Description |
|---|---|
| map(func) | Uses the **func** method to generate a new RDD for each element in the RDD that invokes **map**. |
| filter(func) | **func** is used for each element of an RDD that invokes **filter** and then an RDD with elements containing **func** (the value is **true**) is returned. |
| reduceBykey(func, [numTasks]) | It is similar to **groupBykey**. However, the value of each key is calculated based on the provided **func** to obtain a new value. |
| join(otherDataset, [numTasks]) | If the data set is **(K, V)** and the associated data set is **(K, W)**, then **(K, (V, W)** is returned. **leftOuterJoin**, **rightOutJoin**, and **fullOuterJoin** are supported. |

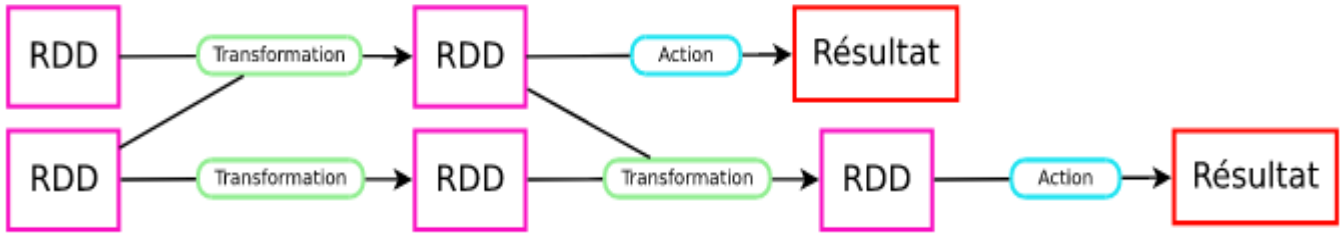
# Structure de données Spark : RDD - Core Concept of Spark

**Les actions**

| Action | Description |
|---|---|
| reduce(func) | Aggregates elements in a dataset based on functions. |
| collect() | Used to encapsulate the **filter** result or a small enough result and return an array. |
| count() | Collects statistics on the number of elements in an RDD. |
| first() | Obtains the first element of a dataset. |
| take(n) | Obtains the top elements of a dataset and returns an array. |
| saveAsTextFile(path) | This API is used to write the dataset to a text file or HDFS. Spark converts each record into a row and writes the row to the file. |

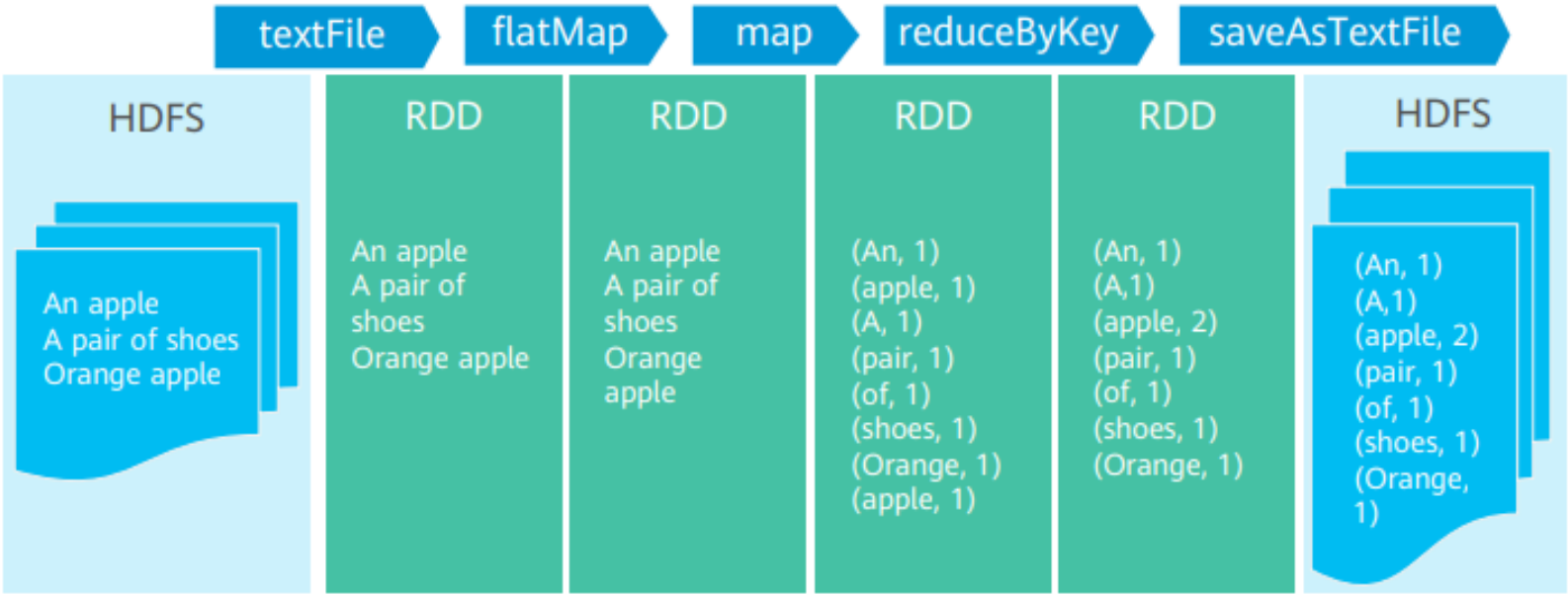
# Structure de données Spark : RDD - Core Concept of Spark

- Dans une application Spark, les transformations et les actions réalisées sur les RDD permettent de construire **un graphe acyclique orienté (DAG : "directed acyclic graph").**




- Les nœuds sont les RDD et les résultats.
- Les connexions entre les nœuds sont soit des transformations, soit des actions.
- Ces connexions sont orientées car elles ne permettent de passer d'un RDD à un autre que dans un sens.
- Le graphe est dit acyclique car aucun RDD ne permet de se transformer en lui-même via une série d'actions.
- Lorsqu'un nœud devient indisponible, à cause d'une malfonction quelconque, il peut être régénéré à partir de ses nœuds parents. C'est précisément ce qui permet la tolérance aux pannes des applications Spark.

# Structure de données Spark : RDD - Core Concept of Spark



# Structure de données Spark : RDD - Core Concept of Spark

## WordCount

```
object WordCount
{
   def main (args: Array[String]): Unit = {
      //Configuring the Spark application name.
      val conf = new
      SparkConf().setAppName("WordCount")
      val sc: SparkContext = new SparkContext(conf)
      val textFile = sc.textFile("hdfs://...")
      val counts = textFile.flatMap(line => line.split(" "))
         .map(word => (word, 1))
         .reduceByKey(_ + _)
      counts.saveAsTextFile("hdfs://...")
   }
```

Create a SparkContext object and set the application name to **Wordcount**.

Load the text file on HDFS to obtain the RDD data set.

Invoke RDD transformation for calculations.
Split the text file by spaces, set the occurrence frequency of each word to 1, and then combine the counts by the same key.
This operation is performed on each executor.

Invoke the **saveAsTextFile** action and save the result. This line triggers the actual task execution.

# Structure de données Spark : DataFrame

- **Un DataFrame** est un Dataset organisé en colonnes nommées.
- Il est conceptuellement équivalent à une table dans une base de données relationnelle ou à un bloc de données en R/Python, mais avec **des optimisations plus riches** sous le capot.
- Les DataFrames peuvent être construits à partir d'un large éventail de sources telles que : des fichiers de données structurées, des tables dans Hive, des bases de données externes ou des RDD existants.
- L'API DataFrame est disponible en Scala, Java, Python et R .
- DataFrame est un cas particulier de DataSet **DataFrame=Dataset[Row].**

# Structure de données Spark : DataFrame

## Differences Between DataFrame, DataSet, and RDD

- Assume that there are two lines of data in an RDD, which are displayed as follows:

| |
|---|
| 1, Allen, 23 |
| 2, Bobby, 35 |

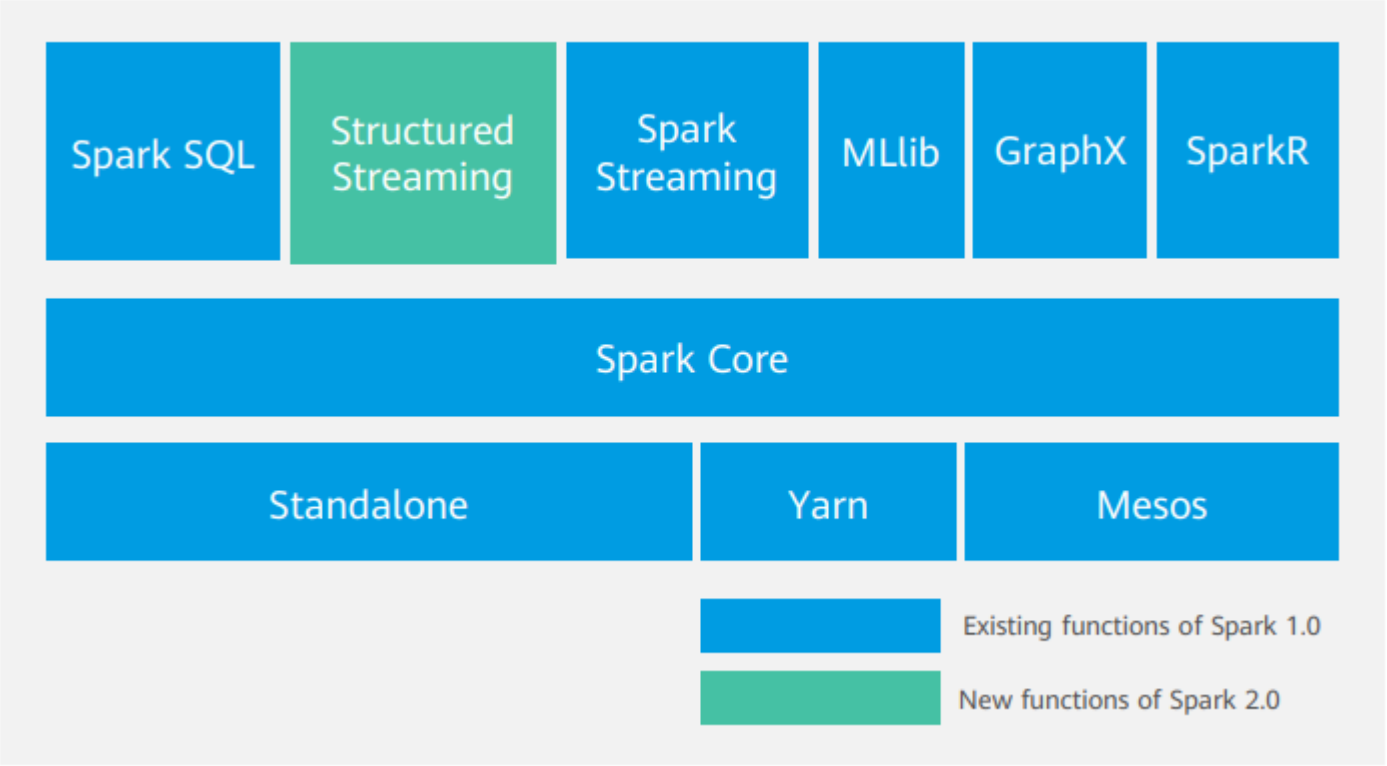
- The data in DataFrame is displayed as follows:

| ID:String | Name:String | Age:int |
|---|---|---|
| 1 | Allen | 23 |
| 2 | Bobby | 35 |

- The data in DataSet is displayed as follows:

| value:People[age: bigint, id: bigint, name:string] |
|---|
| People(id=1, name="Allen", age=23) |
| People(id=1, name="Bobby", age=35) |

# Architecture du système Spark



# Architecture du système Spark

- **Spark SQL :**

❑ Est le module utilisé dans Spark pour le traitement des données structurées. Dans les applications Spark.

❑ On peut utiliser de manière transparente des instructions SQL ou des API DataFrame pour interroger des données structurées.

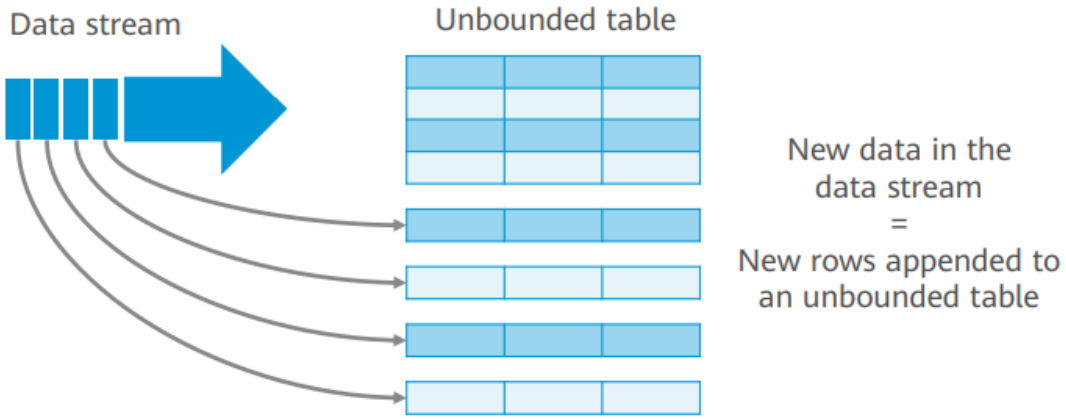
# Architecture du système Spark

- **Structured Streaming :**

❑ Est un moteur de traitement de données en streaming basé sur le moteur Spark SQL.

❑ Permet de compiler un processus de streaming informatique comme l'utilisation de données RDD statiques.

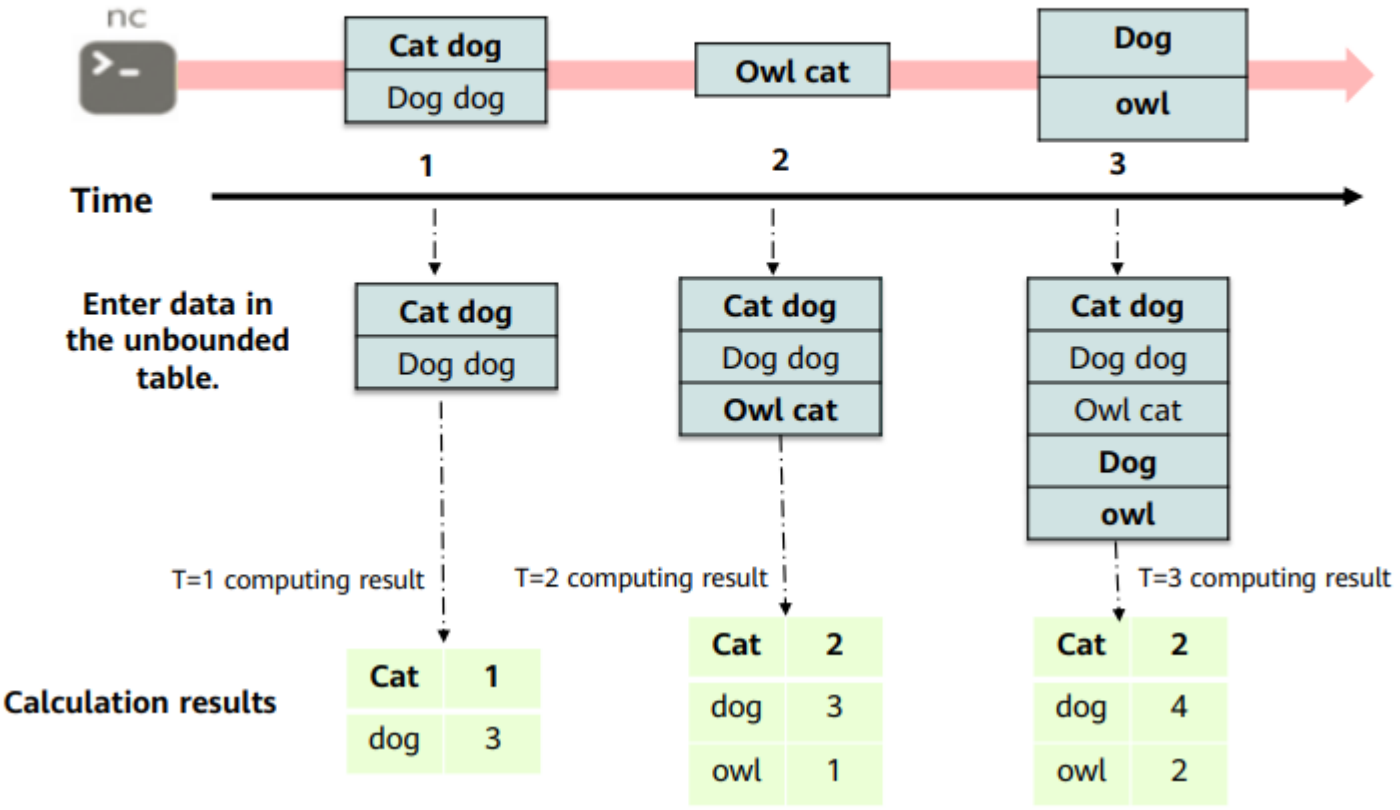
❑ Lorsque les données de streaming sont générées en continu, Spark SQL traite les données de manière incrémentielle et continue, et met à jour les résultats dans l'ensemble de résultats.




Data stream as an unbounded table

# Architecture du système Spark

## Example Programming Model of Structured Streaming

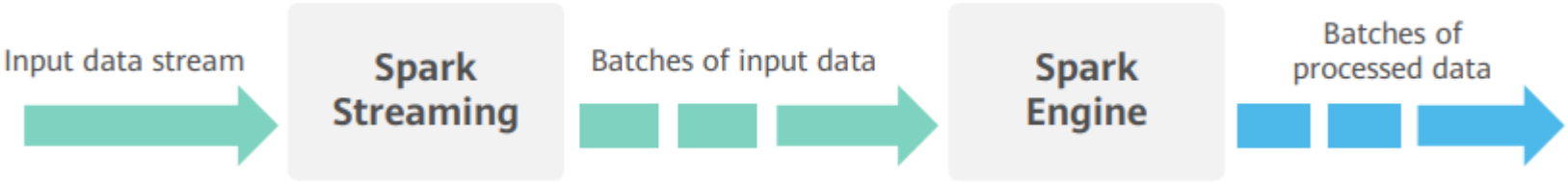


# Architecture du système Spark

- **Spark Streaming :**

❑ Le principe de base de Spark Streaming consiste à diviser les flux de données d'entrée en temps réel par tranche de temps (en secondes).

❑ Puis utiliser le moteur Spark pour traiter les données de chaque tranche de temps d'une manière similaire au traitement par lots.



# Quiz

- **What are the features of Spark?**

❑ light
❑ fast
❑ flexible
❑ Smart

- **What are the most advantages of Spark in comparison with MapReduce?**

❑ Handle offline tasks
❑ In-memory fast computing
❑ Parallel computing
❑ None of them

# Quiz

- **What are the application scenarios of Spark?**

❑ Offline batch processing
❑ real-time stream processing
❑ interactive query
❑ Machine learning

- **Which module is the core module of Spark?**

❑ Spark core
❑ Spark streaming
❑ Spark sql
❑ Graph X

# Quiz

- **What is Spark's own resource management framework?**

❑ Standal one
❑ Mesos
❑ YARN
❑ Docker

- **Regarding RDD, which of the following statement is error?**

❑ RDD has lineage mechanism
❑ RDD is stored on disk by default
❑ RD is a read-only, partitionable distributed data set
❑ RD is Spark's abstraction of basic data

# Quiz

- **RDD has Transformation and Action operators. Which of the following is an Action operator?**

❑ map
❑ saveASTexFile
❑ Filter
❑ Reducebykey

- **What does RDD dependency types include?**

❑ Wide dependency
❑ Narrow dependency
❑ Long dependency
❑ Short dependency