Ministère de l’enseignement supérieur et de recherches scientifiques
Institut Supérieur des Etudes Technologiques de Radès
Département Technologies de l’ information

Introduction à Spark

Spark (Présentation)

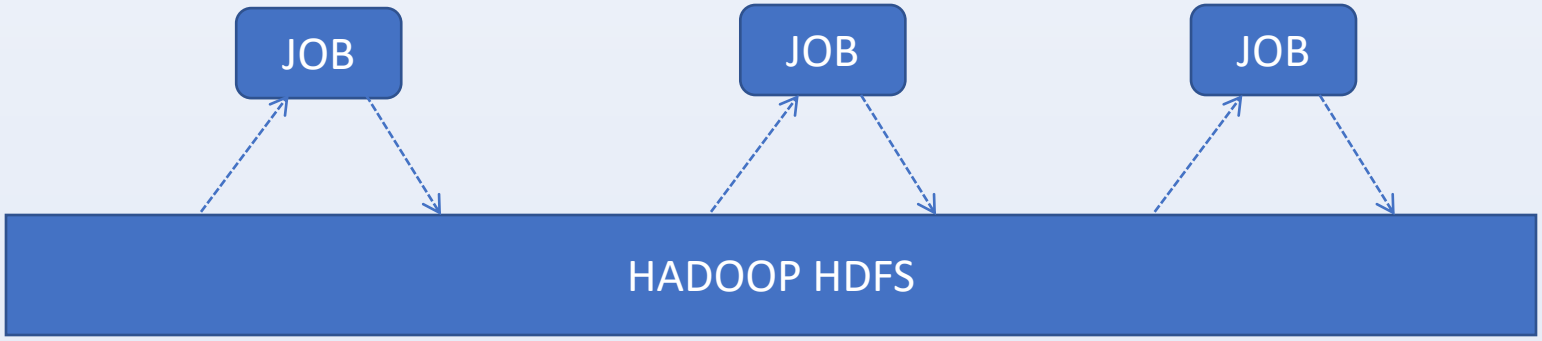
Spark est un système de traitement rapide et parallèle. Il fournit des APIs de haut niveau en Java, Scala, Python et R, et un moteur optimisé qui supporte l'exécution des graphes.

Limites Hadoop MapReduce

MapReduce est dédié pour les traitements Bach:

- Communication via le système du fichier HDFS.
- Tolérance aux fautes des jobs géré par la persistance du HDFS .
- Mémoire n'est pas gérée.

→ Outils pour la gestion de la mémoire : Storm,Samza,Giraph, Impla,Presto,etc.



Daytona Gray Sort 100TB Benchmark

« Spark wins Daytona Gray Sort 100 TB Benchmark »

	Data Size	Time	Node
Hadoop MR (2013)	102.5 TB	72 min	2100
Spark (2014)	100 TB	23 min	206

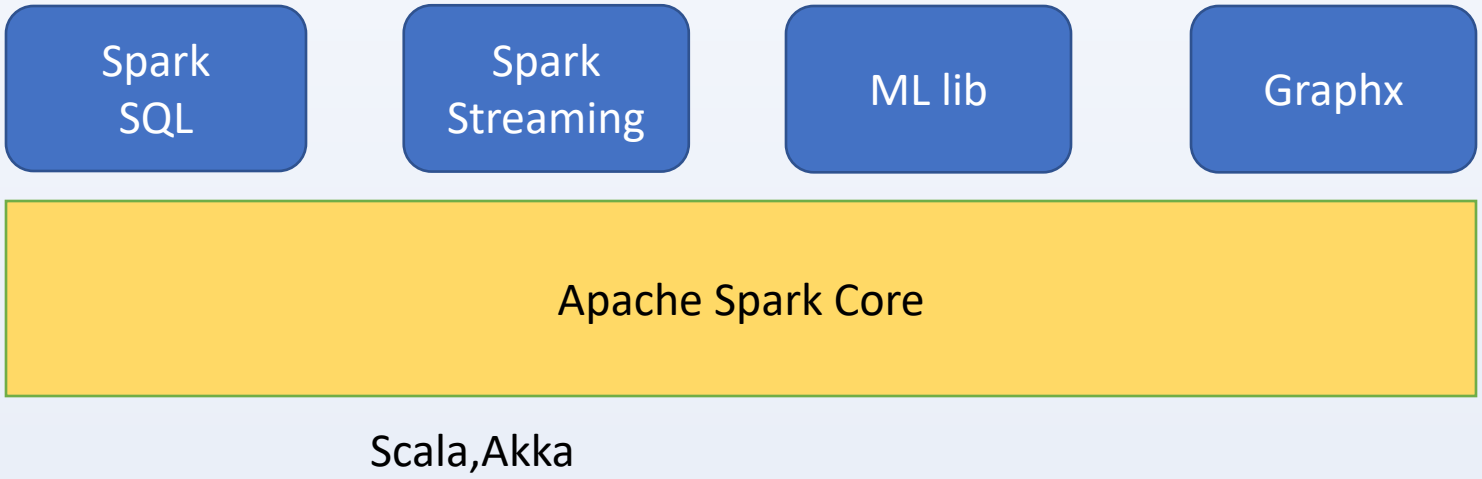
Spark
Traitements

Spark permet de gérer les traitements :

- ✓ Itératifs
- ✓ Interactifs

Qui ne sont pas gérés dans Hadoop MR.

Spark Architecture



Spark
Structures de données

Resilient Distributed Datasets (RDD)

- Une collection d’objets immuables.
- Partitionné et distribué.
- Stocké en mémoire (vitesse d’exécution intéressante)

Resilient Distributed Datasets (RDD)

- Un RDD peut être marqué comme persistant: il est alors placé en mémoire RAM et conservé par Spark.
- Spark conserve l'historique des opérations qui a permis de constituer un RDD, et la reprise sur panne s'appuie sur cet historique afin de reconstituer le RDD en cas de panne.
- L'idée est qu'il est plus facile et efficace de préserver quelques lignes de spécifications que le jeu de données issu de cette chaîne.

Resilient Distributed Datasets (RDD)

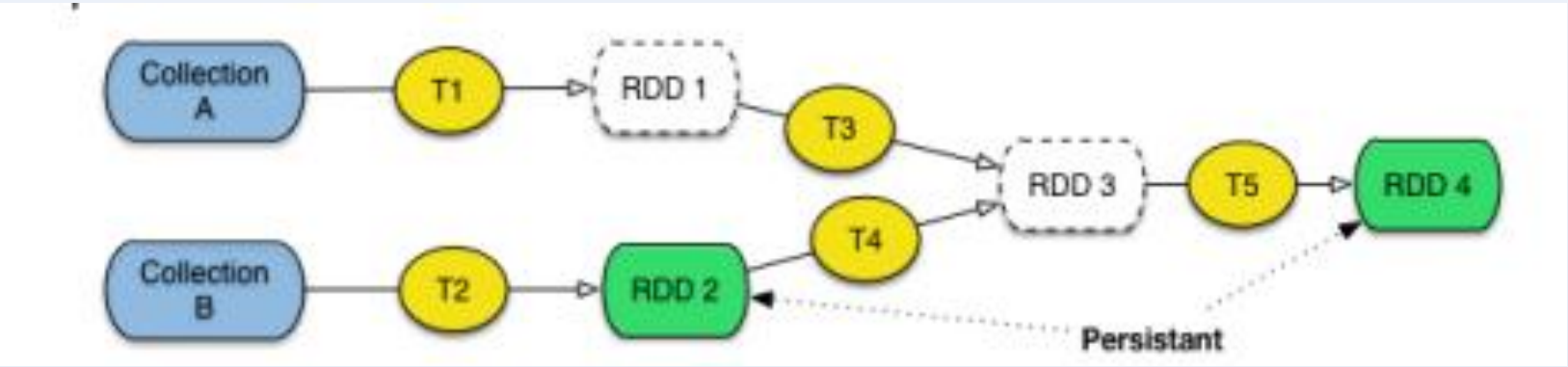
- il est constitué de fragments
- une panne affectant un fragment individuel peut donc être réparée (par reconstitution de l'historique) indépendamment des autres fragments, évitant d'avoir à tout recalculer.
- un RDD est calculé par une transformation
- une transformation sélectionne, enrichit, restructure une collection, ou combine deux collections

Resilient Distributed Datasets (RDD)

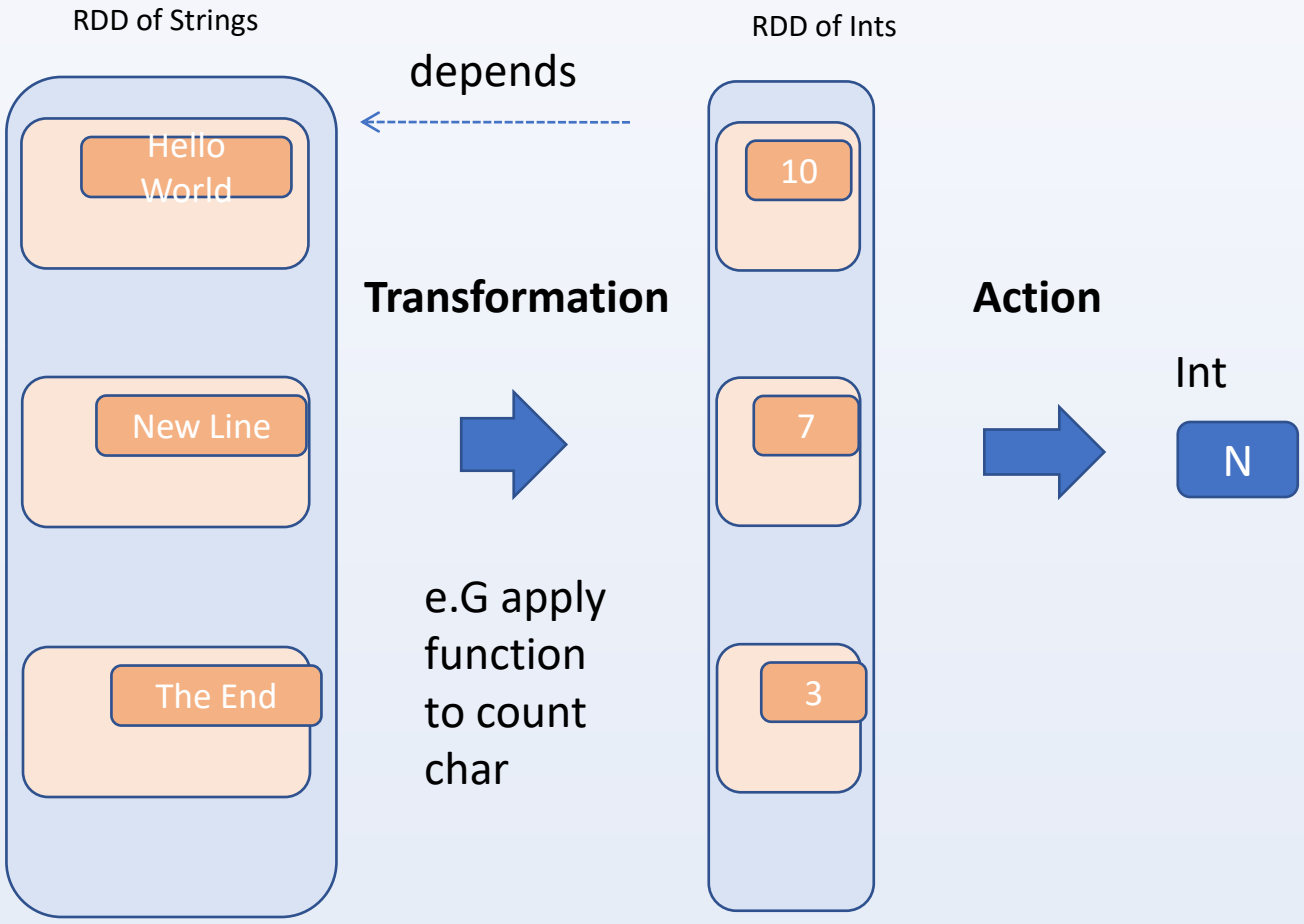
- Des transformations créent des RDD à partir d'une ou deux sources de données.
- Les RDD persistants sont en préservés en mémoire RAM, et peuvent être réutilisés par plusieurs traitements.

Resilient Distributed Datasets (RDD)

- S'il n'est pas marqué comme persistant, le RDD sera transitoire et ne sera pas conservé en mémoire après calcul (c'est le cas des RDD 1 et 3 sur la figure).



Resilient Distributed Datasets (RDD)



Exemple de transformations

map → Prend un document en entrée et produit un document en sortie

filter → Filtre les documents de la collection

flatMap → Prend un document en entrée, produit un ou plusieurs document(s) en sortie

groupByKey → Regroupement de documents par une valeur de clé commune

reduceByKey → Réduction d'une paire (k,[v]) par une agrégation du tableau [v]

crossProduct → Produit cartésien de deux collections join Jointure de deux collections

union → Union de deux collections

Transformation/Action

Transformations	Actions
Map(func) Flatmap(func) Filtrer(func) Groupbykey(func) reduceByKey(func) Mapvalues(func) ……	take(N) Count() Collect() Reduce(func) takeOrdered(N) Top(N) ……

SPARK API

Scala

```scala
val spark = new SparkContext()

val lines    = spark.textFile("hdfs://docs/")    // RDD[String]
val nonEmpty = lines.filter(l => l.nonEmpty())   // RDD[String]

val count = nonEmpty.count
```

Java 8

```java
SparkContext spark = new SparkContext();

JavaRDD<String> lines    = spark.textFile("hdfs://docs/")
JavaRDD<String> nonEmpty = lines.filter(l -> l.length() > 0);

long count = nonEmpty.count();
```

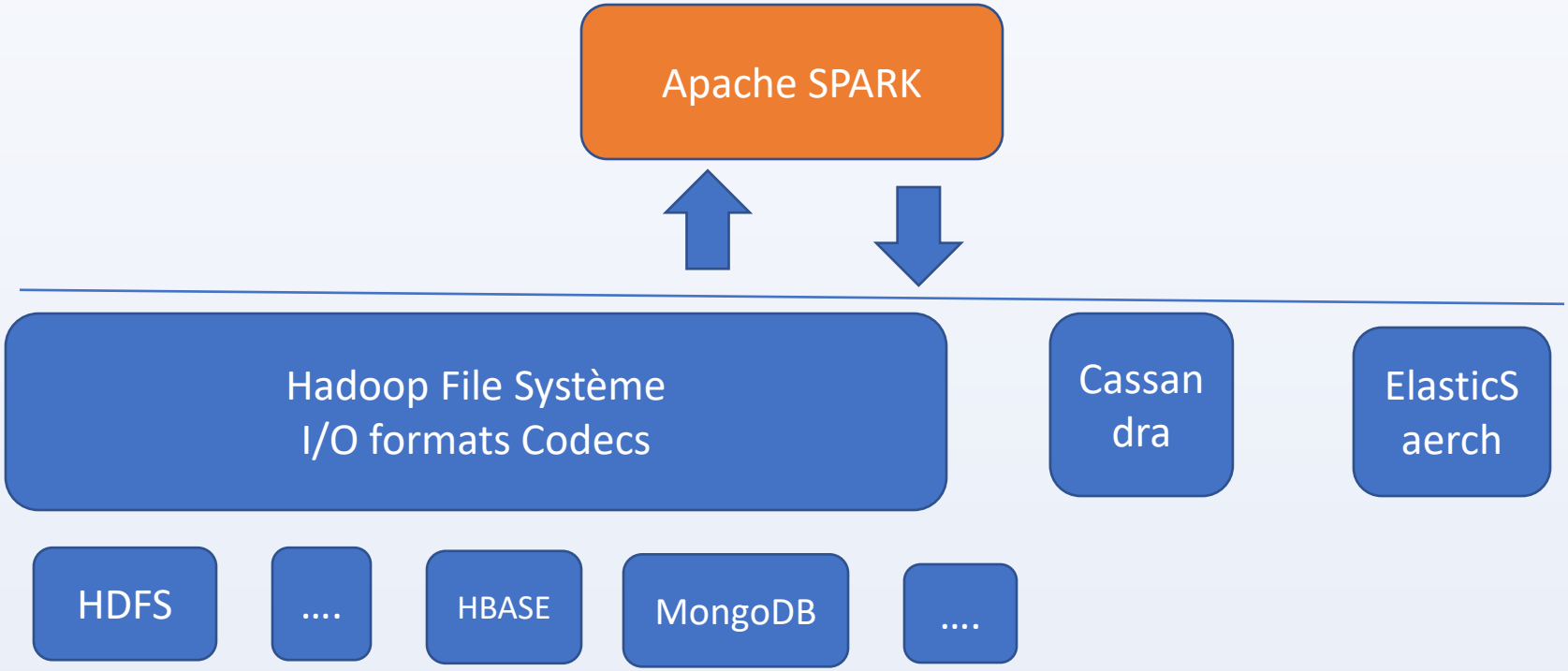
Python

```python
spark = SparkContext()

lines = spark.textFile("hdfs://docs/")
nonEmpty = lines.filter(lambda line: len(line) > 0)

count = nonEmpty.count()
```

Exemple de traitement sur SPARK

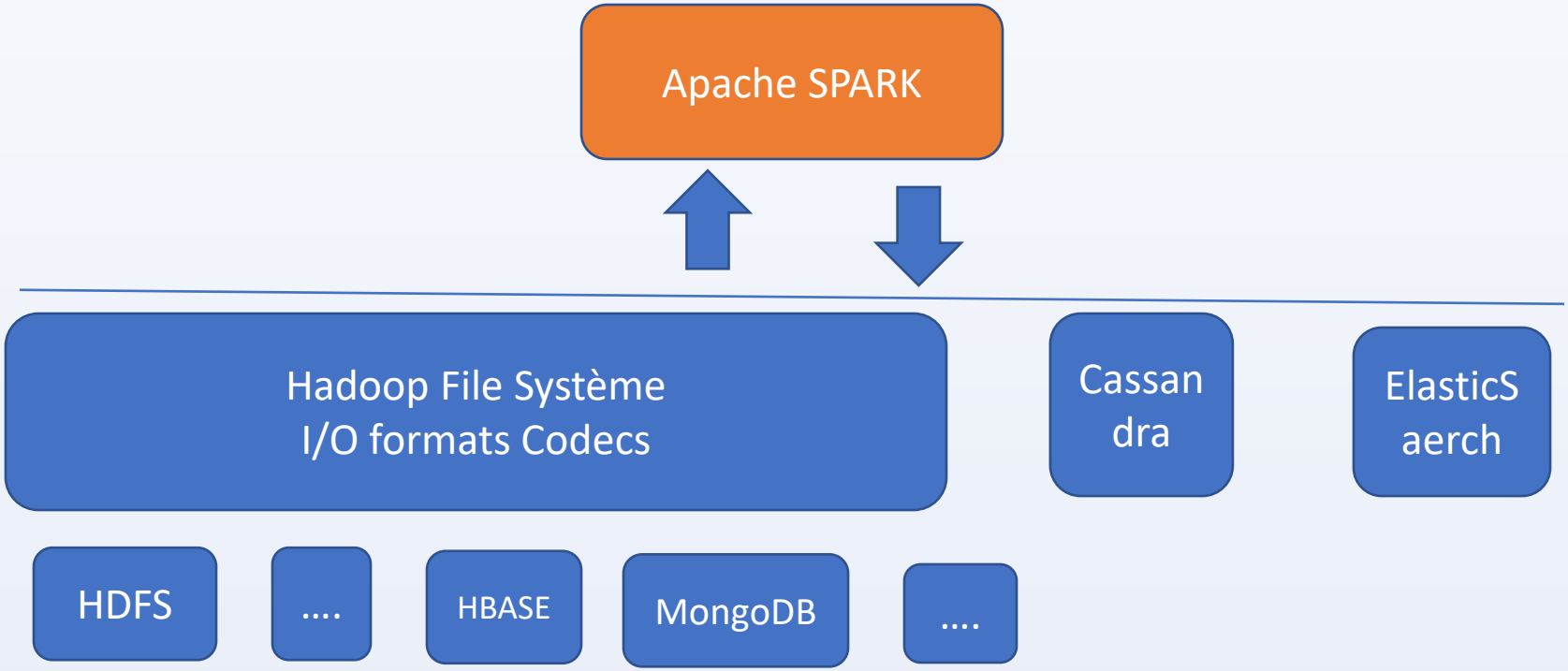
Text Processing Example
Top Words by Frequency

Create RDD from Extranal Data



Spark peut Lire/Ecrire à n'importe quelle source de donnée supportée par Hadoop

Exemple de traitement sur SPARK




```
//Step 1 – create a RDD from  Hadoop Text File
val docs= spark.textFile=("/docs / ")
```

Fonction map

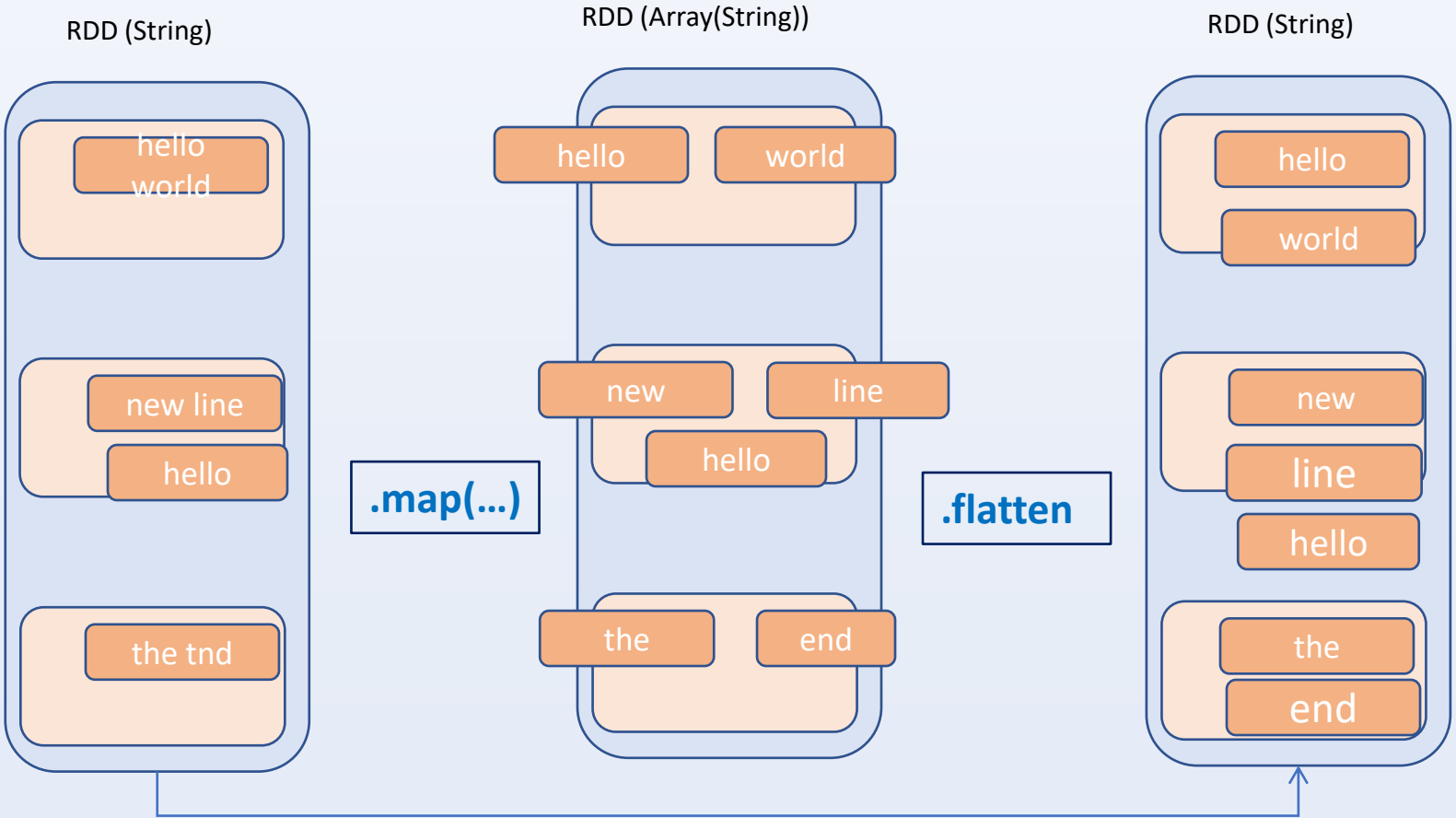
//Step 2– convert lines to lower case

Val lower= docs.**map(**line=>line.toLowerCase)

Fonction map and flatmap

//Step 3– Split lines into word

val words= lower.flatMap(line=>line.split("\\s+"))



Key-Value Pairs

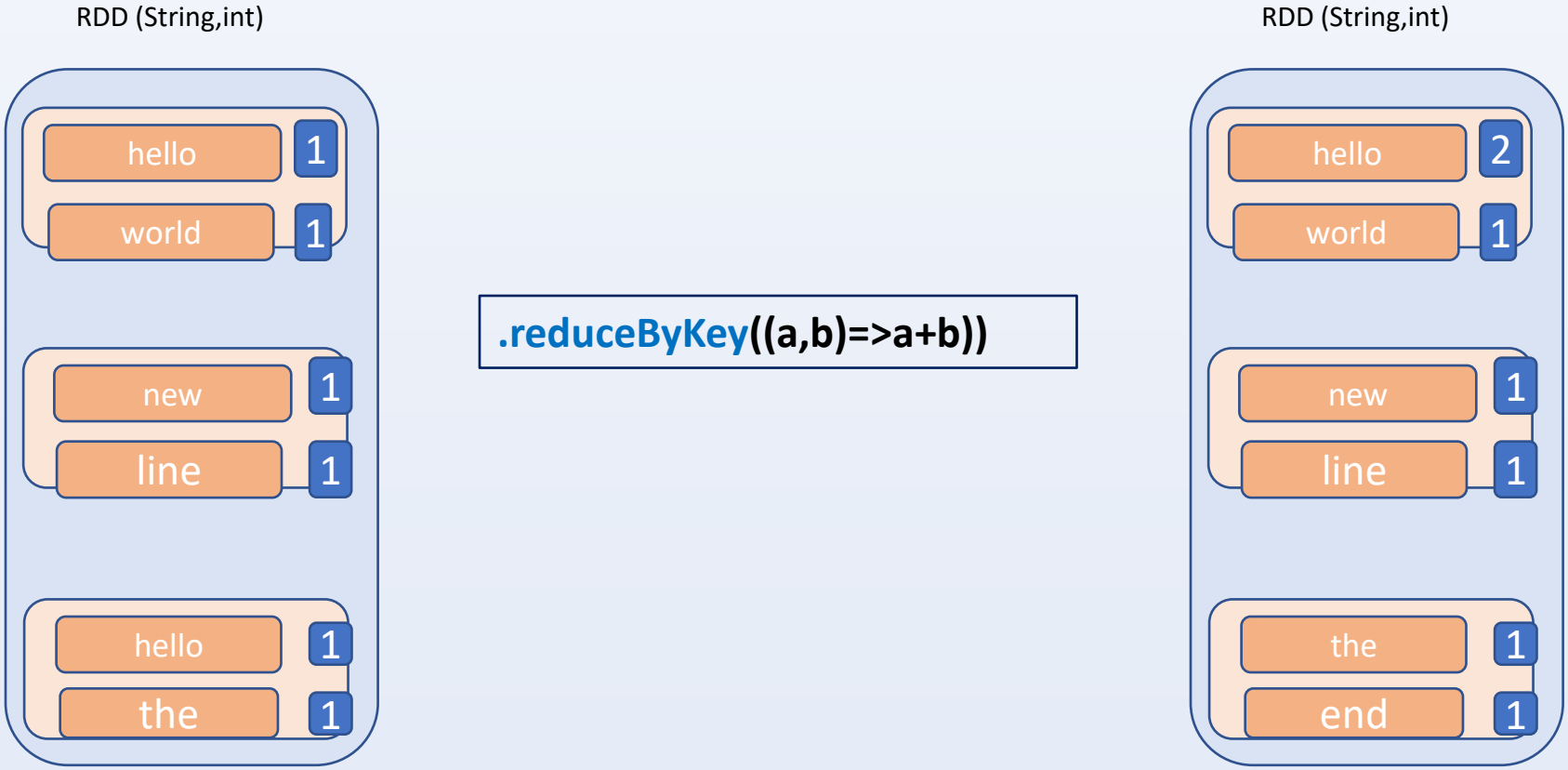
//Step 4– Split lines into words

val counts= words.**map(word**=>word,1))

Suffling

//Step 5– Count all words

val freq=counts.**reduceByKey**(_+_)



Top N (Prepare Data)

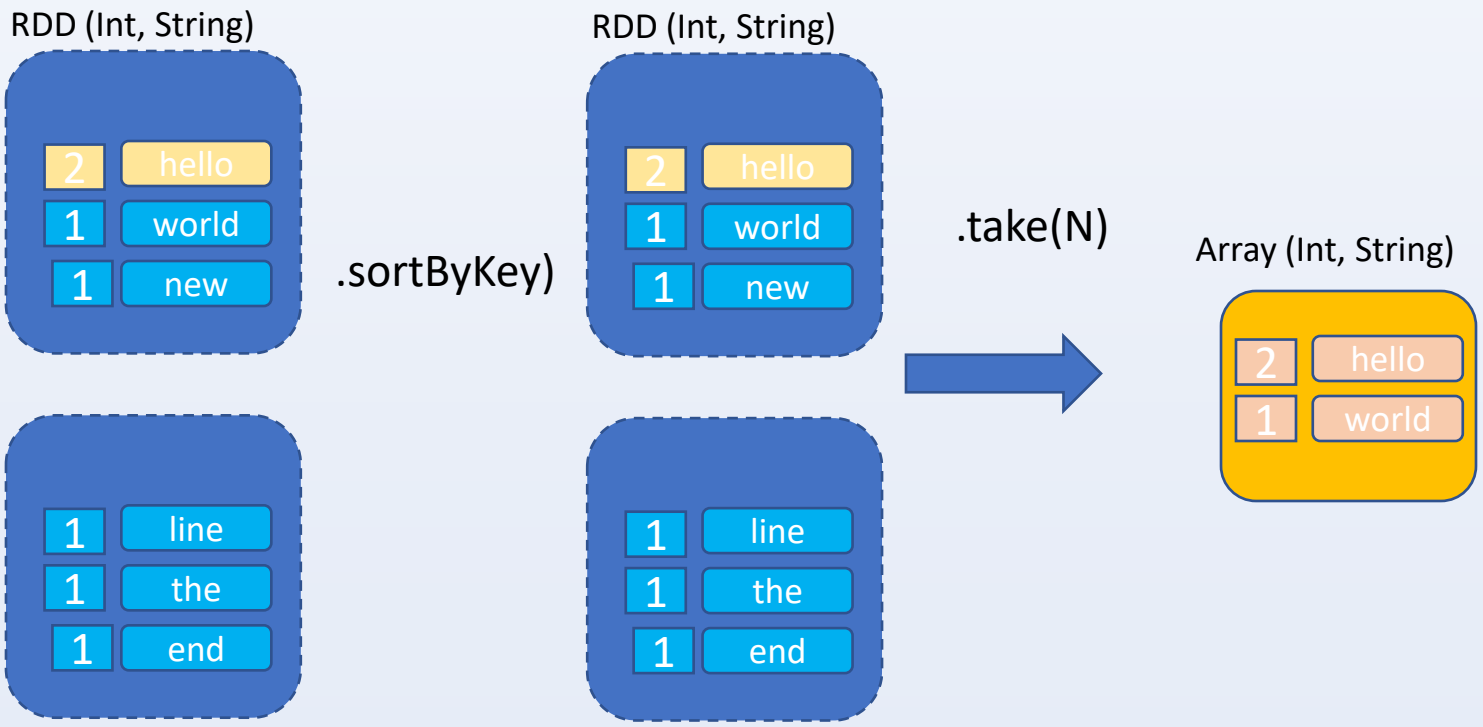
//Step 6– Swap tuples (partial code)

freq.map(_.swap)

Top N (First attemp)

//Step 6– Swap tuples (partial code)

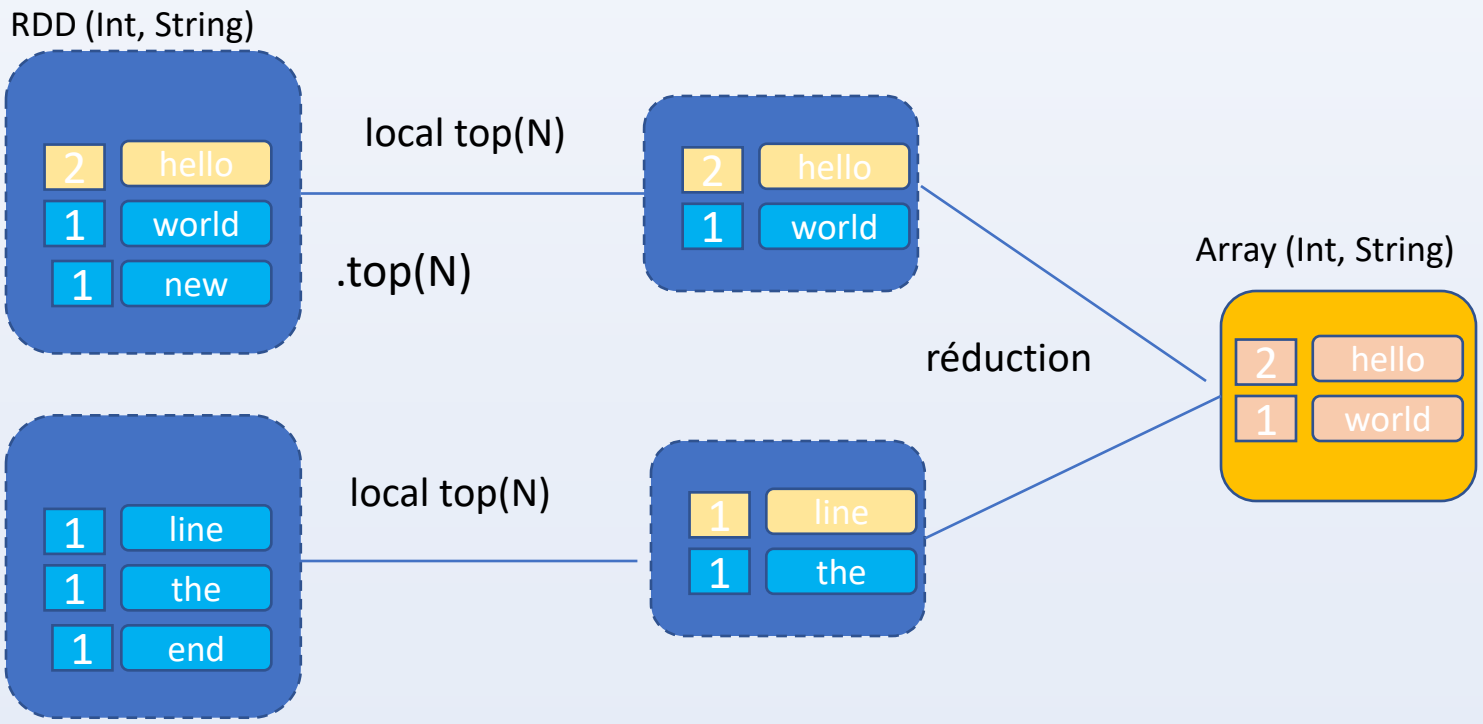
freq.map(_.swap)



Top N

//Step 6– Swap tuples (complete code)

```
val top=freq.map(_.swap).top(N)
```



Top Words by Frequency (Full Code)

```scala
val spark = new SparkContext()

// RDD creation from external data source
val docs   = spark.textFile("hdfs://docs/")

// Split lines into words
val lower  = docs.map(line => line.toLowerCase)
val words  = lower.flatMap(line => line.split("\\s+"))
val counts = words.map(word => (word, 1))

// Count all words (automatic combination)
val freq   = counts.reduceByKey(_ + _)

// Swap tuples and get top results
val top = freq.map(_.swap).top(N)

top.foreach(println)
```