

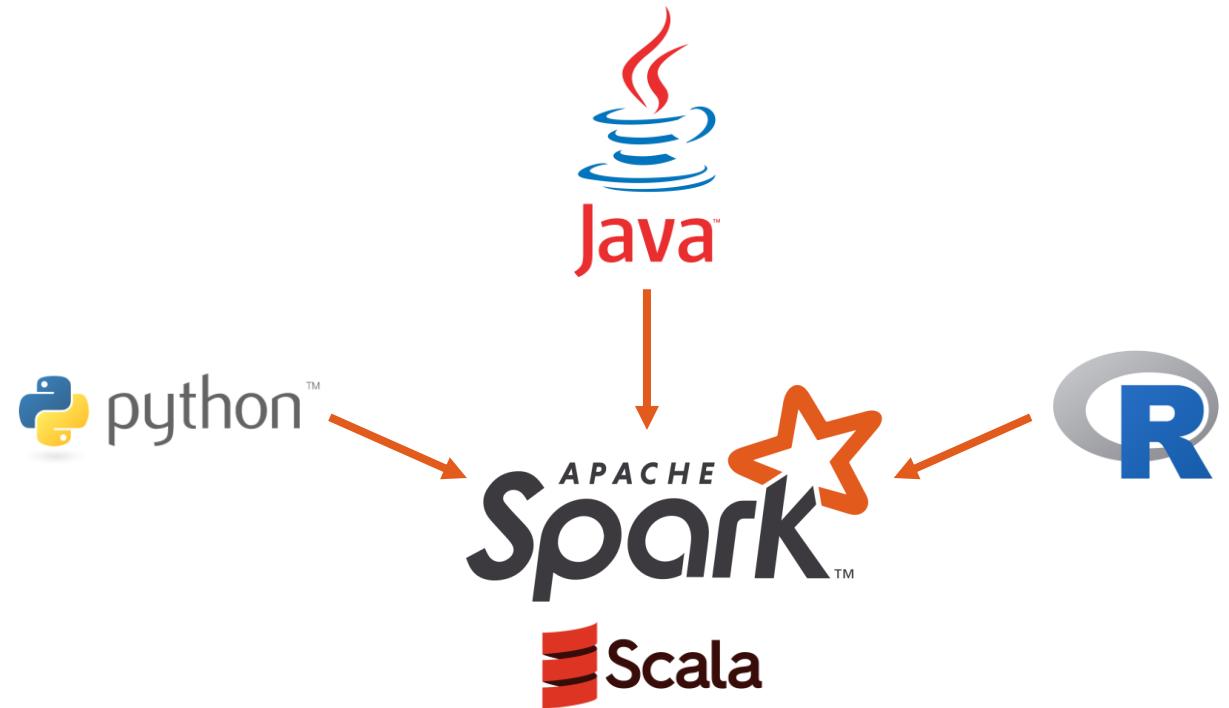
# SPARK



M. Ben & R. Tavenard

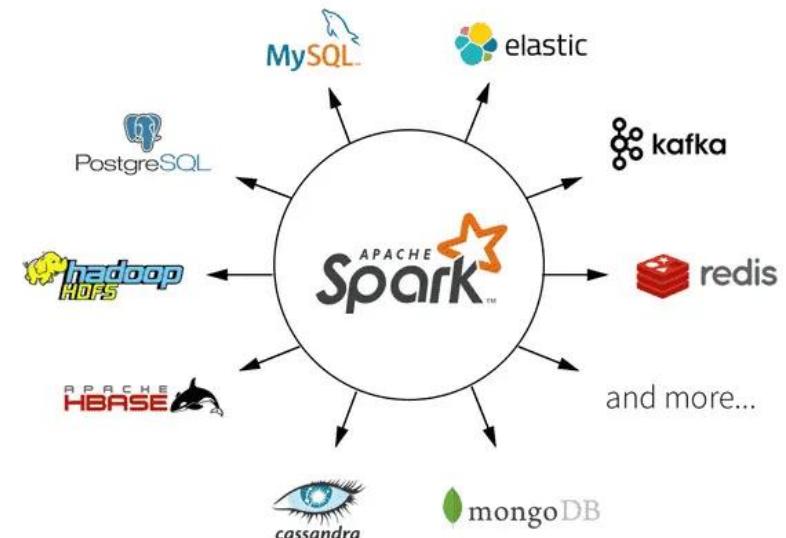
# Spark, qu'est-ce que c'est ?

- Framework de calculs distribués pour le big data
  - Analyse de données
  - Data science
  - Machine learning
- Support multi-langages
  - Scala (langage natif)
  - Java
  - Python (PySpark)
  - R
- Utilise des machines virtuelles Java
  - Nécessite une installation Java sur les machines



# Spark : caractéristiques

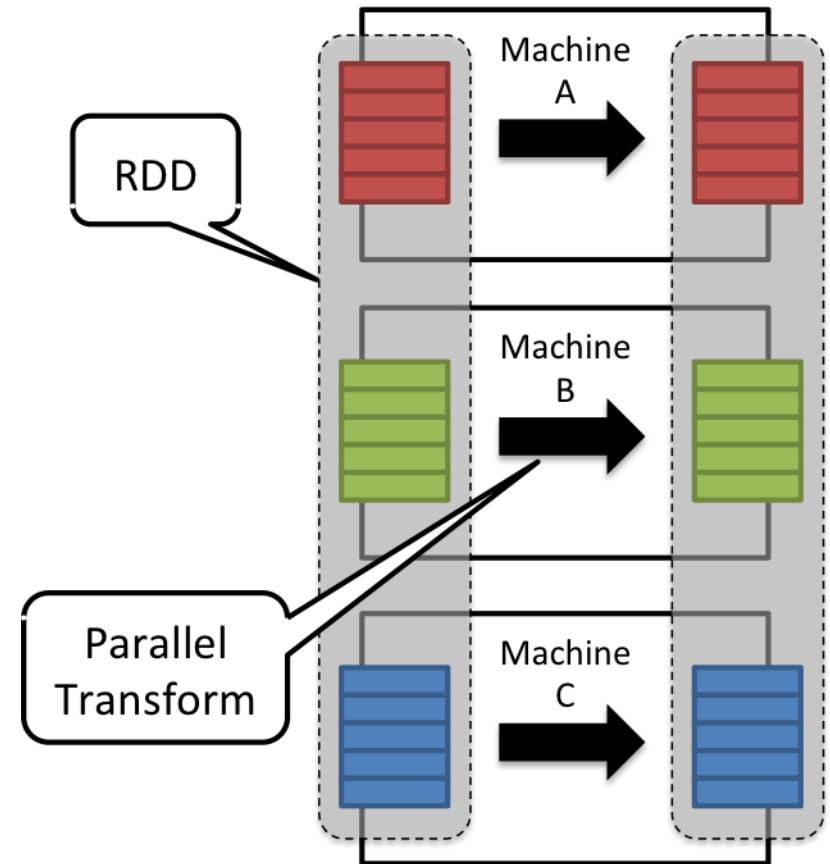
- Evolutif et flexible
  - Calculs sur machine unique ou sur cluster extensible (ex : cluster Hadoop)
- Rapide
  - Calculs sur données distribuées en RAM
  - Jusqu'à 100x plus rapide que Hadoop MapReduce
- Tolérant aux fautes
  - Procédé de reconstruction des données
- Largement compatible avec les outils de gestions de données
  - Systèmes de stockage
  - Bases de données
  - Outils de flux de données



source : <https://www.databricks.com>

# Les RDDs de Spark

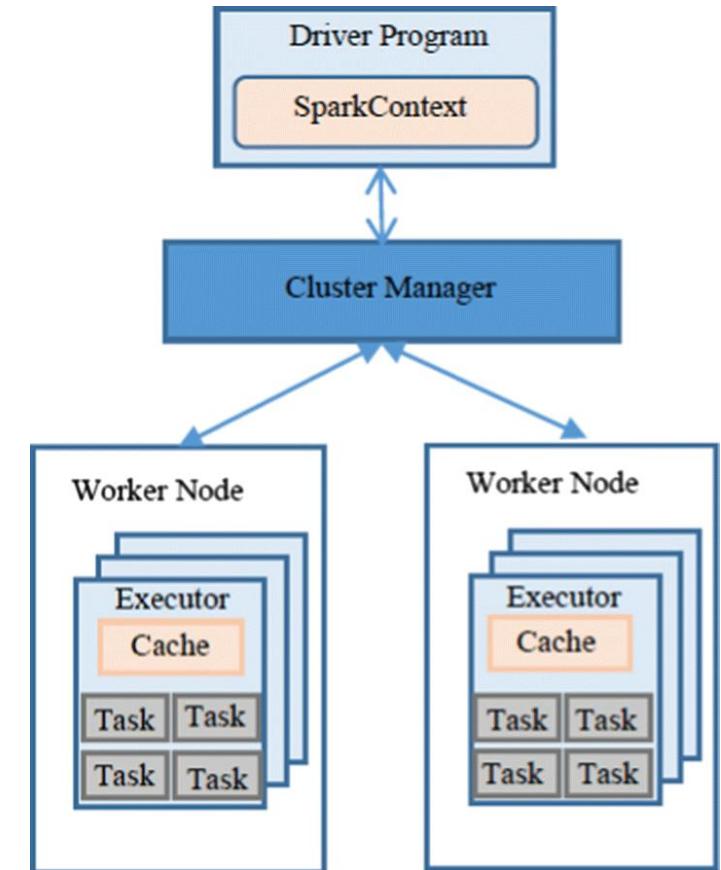
- *Resilient Distributed Dataset*
- Collection de données distribuée en mémoire RAM sur plusieurs machines
  - Structure sous-jacente de tous les types de donnée Spark
- Traitements sur un RDD
  - se font en parallèle
  - sont distribués sur les machines contenant les différentes partitions du RDD
- Un RDD est immuable
  - Les transformations appliquées à un RDD construisent un nouveau RDD



source : <https://vivani.net>

# Distribution des calculs

- *Driver Program*
  - Programme principal
  - Envoyé sur les nœuds de calcul
  - Contient un objet *SparkContext*
    - Connexion au cluster
    - Création et gestion des RDDs
- *Cluster Manager* (sur *Master Node*)
  - Gestion des ressources du cluster
    - Ex. : *Spark Standalone* ou *Hadoop YARN*
- *Worker Nodes*
  - Nœuds de calcul
  - Exécutent une partie des traitements
  - Traitent la sous-partie des données (partition) hébergée sur la machine



source : <https://journalofbigdata.springeropen.com>

# Mode d'utilisation de Spark

- Mode local
  - le *driver program* reste sur la machine locale
  - Spark utilisé comme framework de calculs parallélisés ou pour des tests
- Mode client
  - session connectée au cluster depuis la machine utilisateur
  - *driver program* sur machine utilisateur et envoyé sur les *worker nodes*
  - pour session interactive, tests ou traitements courts sur le cluster
- Mode cluster
  - *driver program* envoyé sur le *master node* (machine « maître » du cluster) et sur les *worker nodes*
  - fonctionne de façon autonome sur le cluster
  - mode utilisé pour un système en production

# Opérations sur les RDDs

- Deux types d'opération sur les RDDs
  1. Transformation
    - construit un nouveau RDD
    - *lazy evaluation* :
      - transformations « enregistrées » mais traitement retardé
      - créent un graphe d'opérations du RDD (*RDD lineage*)
      - Le *RDD lineage* est conservé pour pouvoir reconstruire les données en cas de défaillance
  2. Action
    - lance les traitements des transformations enregistrées
    - renvoie une valeur/liste de valeurs au *driver program* ou écrit sur disque

# Transformations & Actions sur les RDDs (Core API)

- Transformations
  - *Narrow transformations*
    - sans transfert de données
      - map()
      - mapPartition()
      - flatMap()
      - filter()
      - ...
    - avec transfert de données entre *worker nodes*
      - groupByKey()
      - reduceByKey()
      - join()
      - ...
  - *Wide transformations*
    - sans transfert de données
      - mapPartitions()
      - sample()
      - partitionBy()
      - co-partitioned joins
      - ...
    - avec transfert de données entre *worker nodes*
      - broadcast()
      - repartition()
      - co-partitioned joins
      - ...
- Actions
  - nécessitent un transfert de données vers le *driver program*
    - collect()
    - reduce()
    - count()
    - countByKey()
    - take(n)
    - foreach()
    - saveAsTextFile(path)
    - ...

# Les librairies haut niveau de Spark

- Spark SQL

- DataFrames Spark
- Support SQL



- Pandas-on-Spark API

- API Pandas sur Spark (support incomplet)



- MLlib

- Machine Learning



- GraphX

- Analyse de graphes



- Spark Streaming

- Traitements de flux de données



# PySpark Core RDD API

- Installation de PySpark

```
pip install pyspark
```

- Import de la classe `SparkSession` du module `pyspark.sql`

```
from pyspark.sql import SparkSession
```

- Objet `SparkSession`

- Objet du *driver program* permettant les échanges avec

- le *cluster manager*
  - les *worker nodes* du cluster

- Contient un objet `SparkContext`

- Support pour les RDDs

# PySpark Core RDD API

- Crédit d'une SparkSession

- En local

```
spark = SparkSession.builder.appName("Mon appli Spark") \
    .master("local[2]") \ # ou local[*] (utilise tous les coeurs)
    .getOrCreate()
```

- Sur un cluster *Spark Standalone*

```
spark = SparkSession.builder.appName("Mon appli Spark") \
    .master("spark://master-name:7077") \
    .getOrCreate()
```

- Sur un cluster *Spark on YARN* (cas de Dataproc Google)

```
spark = SparkSession.builder.appName("Mon appli Spark") \
    .master("yarn") \
    .getOrCreate()
```

- Arrêt de la SparkSession

- Arrête le SparkContext sous-jacent : à placer à la fin du *driver program*

```
spark.stop()
```

# PySpark Core RDD API

- Crédation et manipulation d'un RDD

- Récupération de l'objet `SparkContext` de la `SparkSession`

```
sc = spark.sparkContext
```

- Crédation

```
rdd1 = sc.parallelize([0,1,2,3,4])          # à partir d'une liste Python  
rdd2 = sc.range(100, 200, 10)                # à partir d'un range Spark  
rdd3 = sc.textFile(<local or HDFS path>) # à partir d'un fichier texte
```

- Manipulation (méthodes des objets RDD)

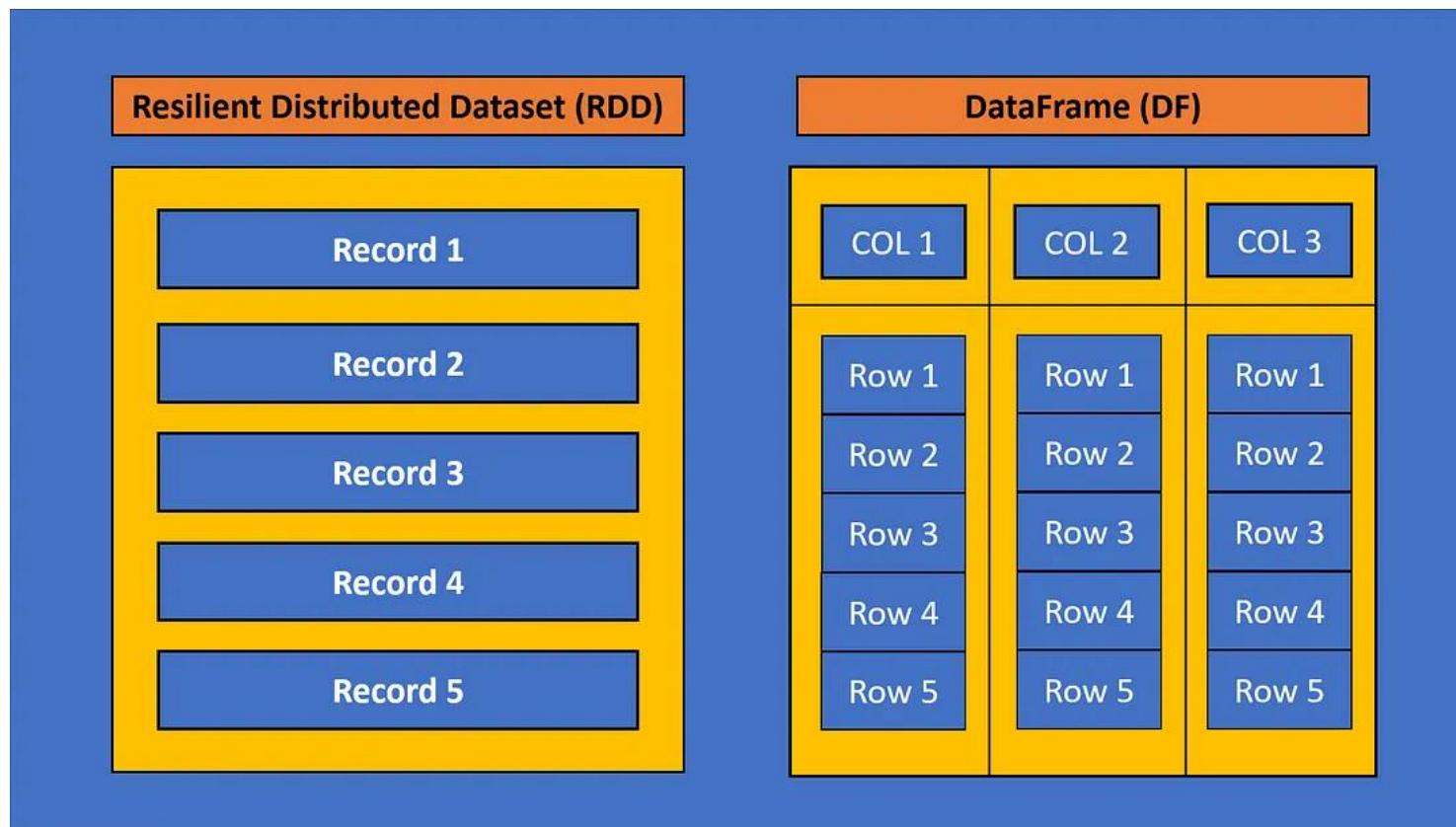
- Informations et gestion : `getNumPartitions()`, `repartition()`, `cache()`, ...
  - Transformations : `map()`, `flatMap()`, `filter()`, ...
  - Actions : `collect()`, `count()`, `reduce()`, `saveAsTextFile(<path>)`, ...
  - Voir la [documentation en ligne](#)

# PySpark SQL API

- Support DataFrame Spark et SQL
- DataFrame Spark
  - Données tabulaires distribuées
  - Associée à un schéma : typage des données par colonne
    - peut être défini explicitement
    - ou inféré automatiquement
- Tables SQL Spark
  - Crées à partir de DataFrames Spark
  - Manipulation et interrogation avec le langage SQL
    - génère de nouvelles DataFrames Spark (résultat de la requête)

# PySpark SQL API

- RDD **vs** DataFrame



source : <https://levelup.gitconnected.com>

# PySpark SQL API

- Création d'une DataFrame Spark

- Avec la méthode [createDataFrame](#) de la SparkSession

```
sdf1 = spark.createDataFrame(source, [schema], *options)  
# source : collection Python, RDD, liste de Row, Dataframe Pandas...
```

- Lecture depuis un fichier avec l'objet [SparkSession.read](#)

```
sdf2 = spark.read.csv(<path to CSV file>, [schema], *options)  
sdf3 = spark.read.json(<path to JSON file>, [schema], *options)  
sdf4 = spark.read.text(<path to TEXT file>, *options)
```

- Visualisation du contenu

- Opération de type « action » (transfert des données au *driver program*)

```
sdf1.show()      # ou sdf1.show(n=10) pour les 10 premières lignes
```

# PySpark SQL API

- Manipulation des [DataFrame](#) Spark
  - Transformations
    - Valeurs : [replace](#)
    - Colonnes : [select](#), [drop](#), [withColumns](#), [toDF](#), ...
    - Lignes : [filter](#)/[where](#), [distinct](#), [sort](#), [orderBy](#), [foreach](#), ...
    - Table : [join](#), [intersect](#), [union](#), [transform](#), [createOrReplaceTempView](#), ...
    - Groupes : [groupBy](#) ([GroupedData](#)), [agg](#), [avg/mean](#), [count](#), [max](#), [min](#), [sum](#), ...
  - Actions
    - Récupération & visualisation : [show](#), [collect](#), [count](#) ,...
  - Informations, schéma, RDD sous-jacent
    - [summary](#), [schema](#) ([StructType](#)), [printSchema](#), [to](#), [rdd](#), ...
  - Écriture
    - Vers stockage externe : [write](#)

# PySpark SQL API

- Fonctions du module `functions`
  - Beaucoup prennent en paramètre(s) une ou plusieurs colonnes (nom ou objet [Column](#))
  - Import :

```
from pyspark.sql import functions as sf
```
  - Catégories de fonctions :
    - Récupération et création d'objets colonnes : [col](#), [lit](#), ...
    - Fonctions mathématiques : [sqrt](#), [log](#), [pow](#), [round](#), ...
    - Agrégation : [avg](#), [count](#), [min](#), [max](#), [std](#), [sum](#), ...
    - Chaînes de caractères : [lower](#), [upper](#), [contains](#), [substr](#), [split](#), ...
    - Dates : [curdate](#), [date\\_format](#), [year](#), [month](#), ...
  - Voir la [documentation complète en ligne](#)
- Manipulation des objet [Column](#)
  - Méthodes : [alias](#), [astype/cast](#), [between](#), [contains](#), [asc](#), [desc](#), [substr](#), ...
  - Voir la [documentation complète en ligne](#)