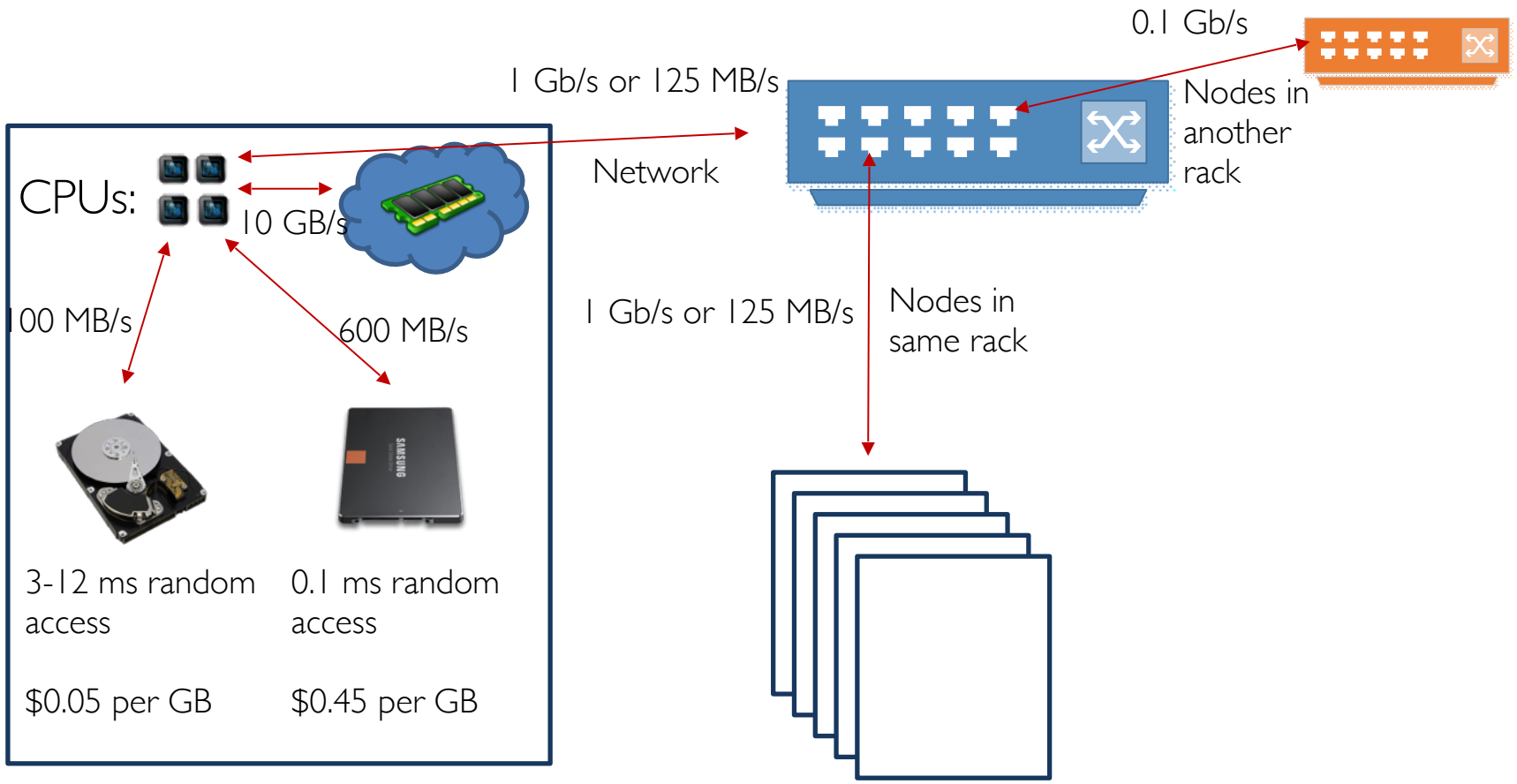
Traitement de données en mémoire : Introduction à Spark

Vincent Leroy

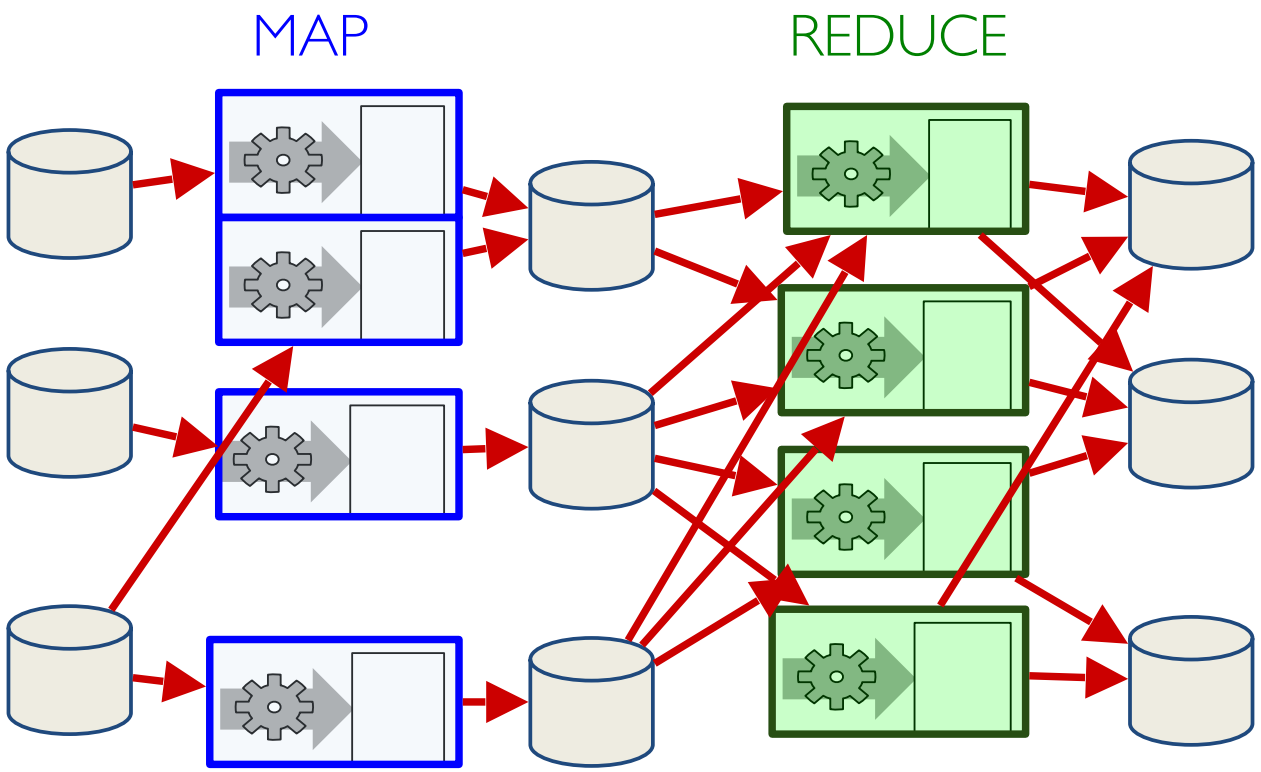
Sources

- Resilient Distributed Datasets, *Henggang Cui*
- Coursera Introduction to Apache Spark, *University of California, Databricks*

Organisation d'un Data Center



Exécution Hadoop MapReduce

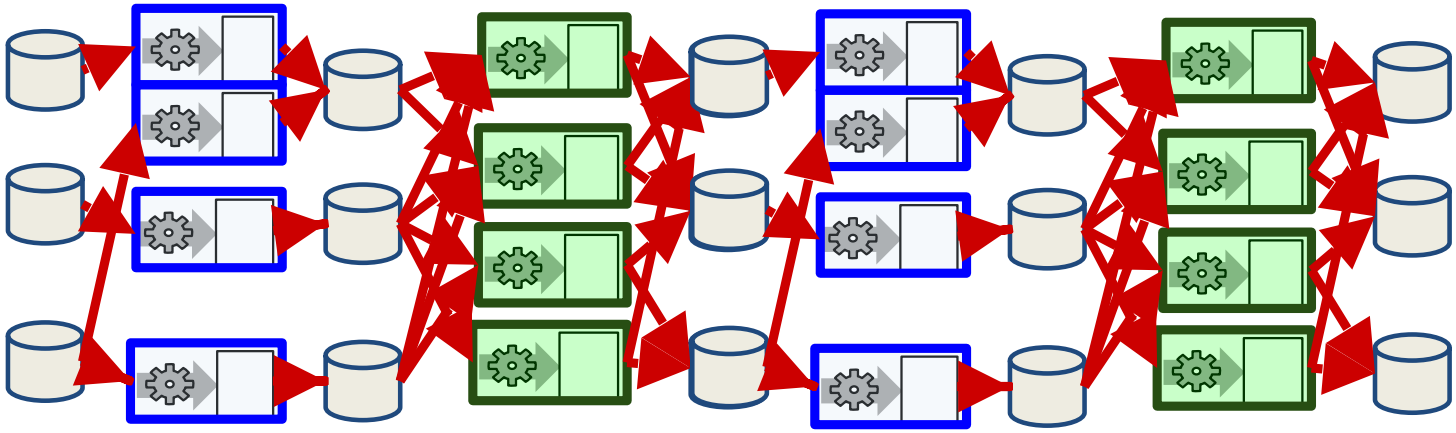


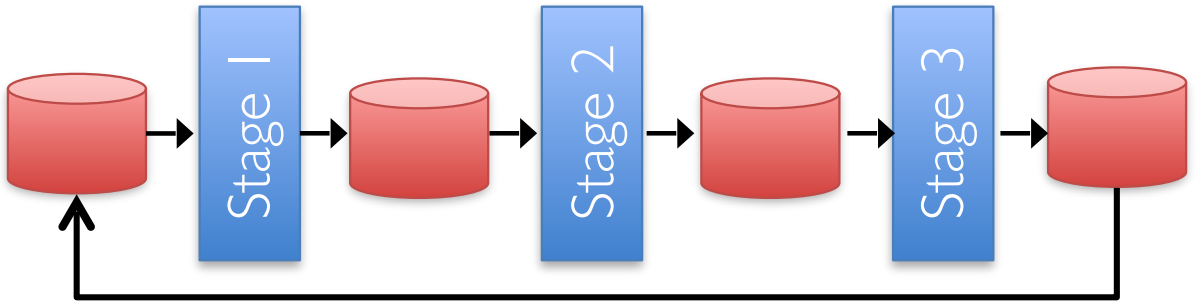
Passage par le disque à chaque étape

Traitements Itératifs

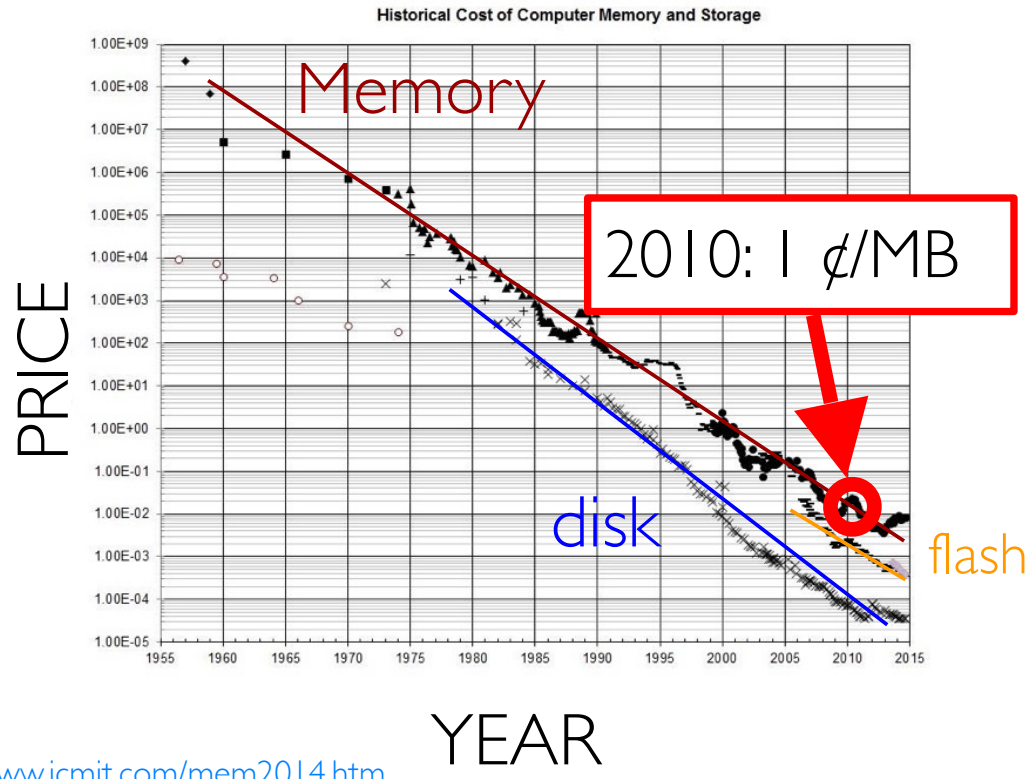
- I/O disque à chaque répétition

→ Lent pour beaucoup de petites itérations





Coût mémoire



Des coûts plus bas font qu'on peut mettre plus de mémoire par serveur

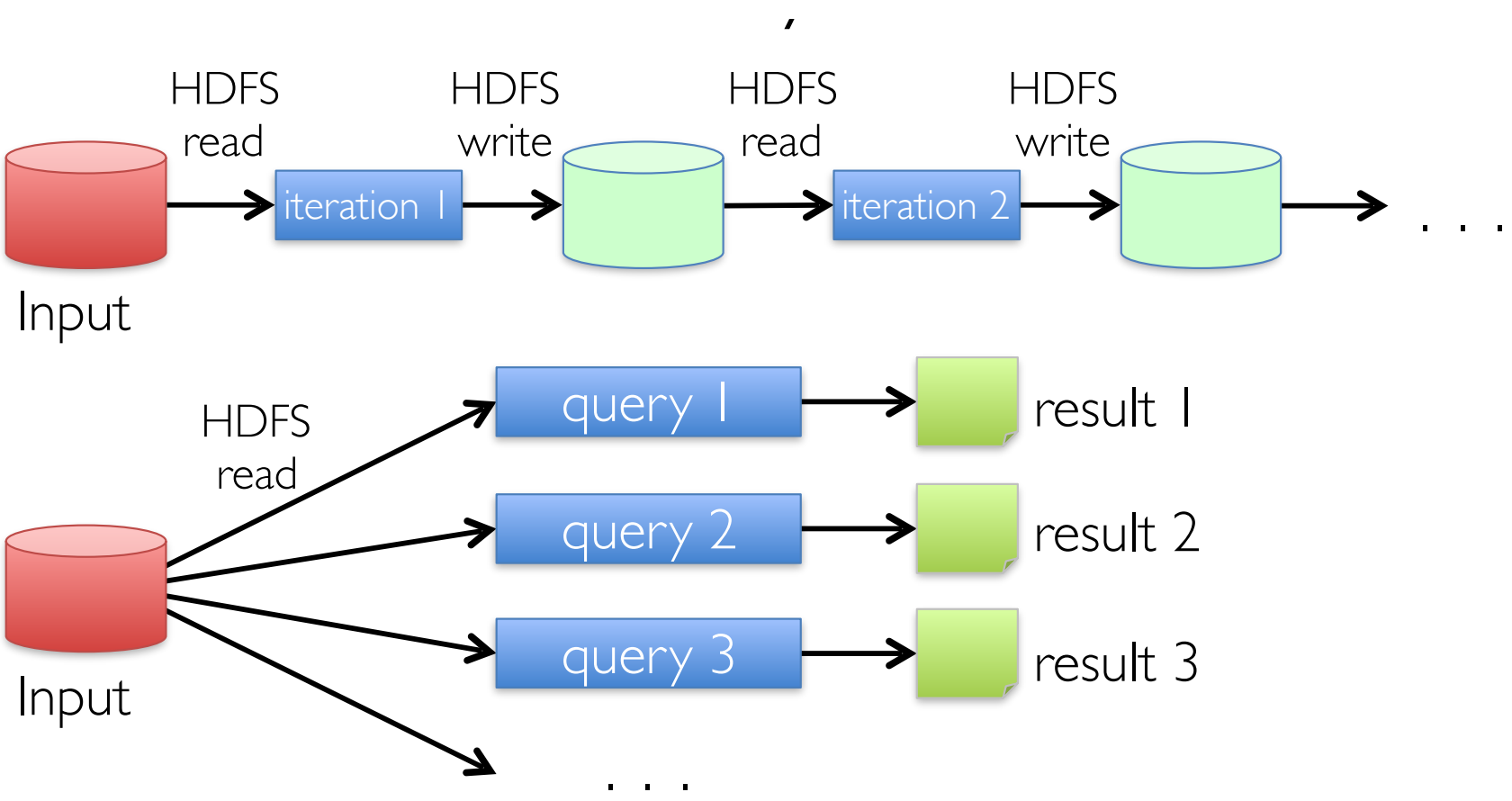
http://www.jcmit.com/mem2014.htm

Traitement en Mémoire (RAM)

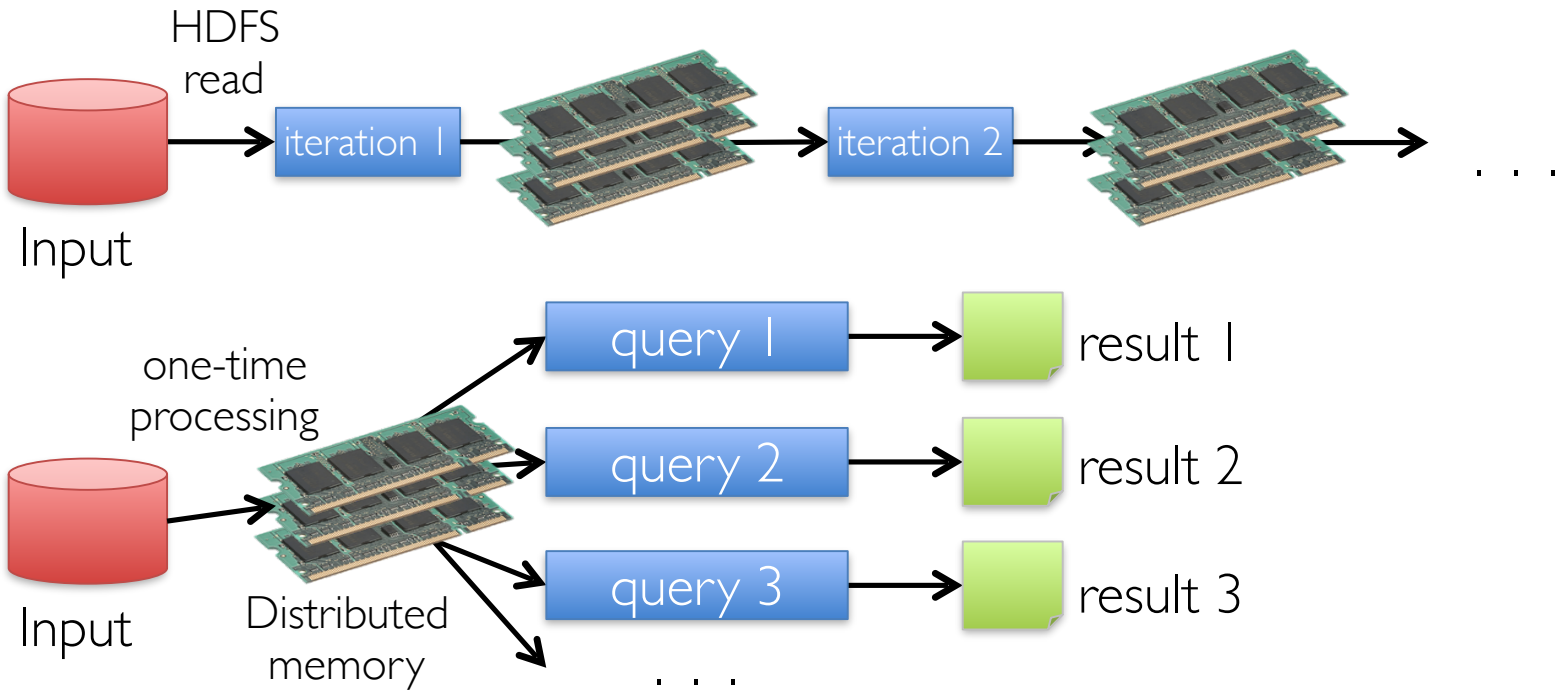
- Beaucoup de jeux de données tiennent en RAM (1 ou plusieurs machines)
- La RAM est rapide et évite les I/O disque

→ Plateforme de traitement Spark

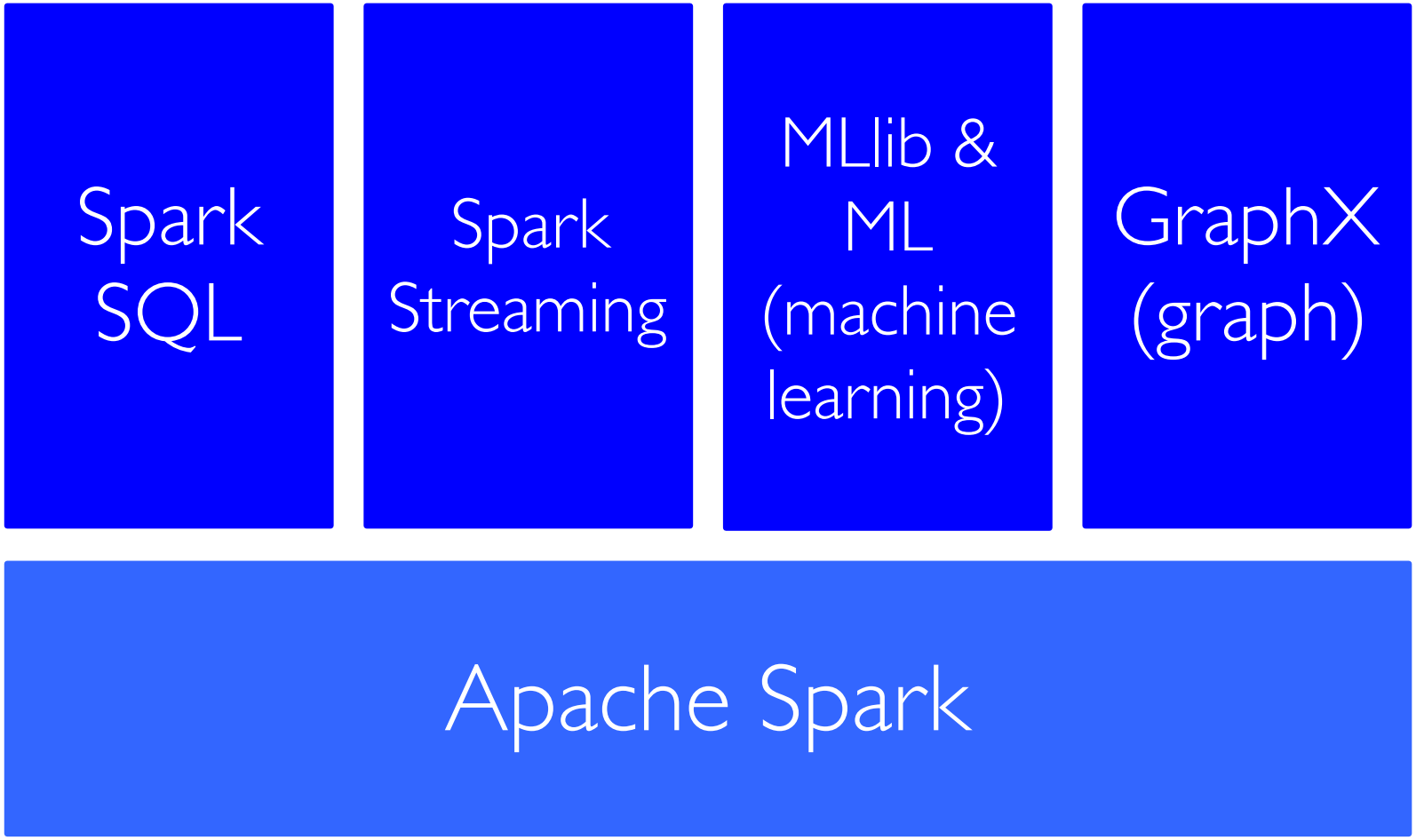
Remplacer le Disque par la RAM



Remplacer le Disque par la RAM



Architecture de Spark



Resilient Distributed Datasets (RDDs)

RDD

- Collection de données
  - Distribuée
  - En lecture-seule
  - En mémoire
  - Créée depuis un stockage fiable ou un autre RDD

Création d'une RDD

```
# Parallelize in Python
wordsRDD = sc.parallelize(["fish", "cats", "dogs"])
```

Parallelize:
Prend une collection classique existante en mémoire et la transforme en RDD

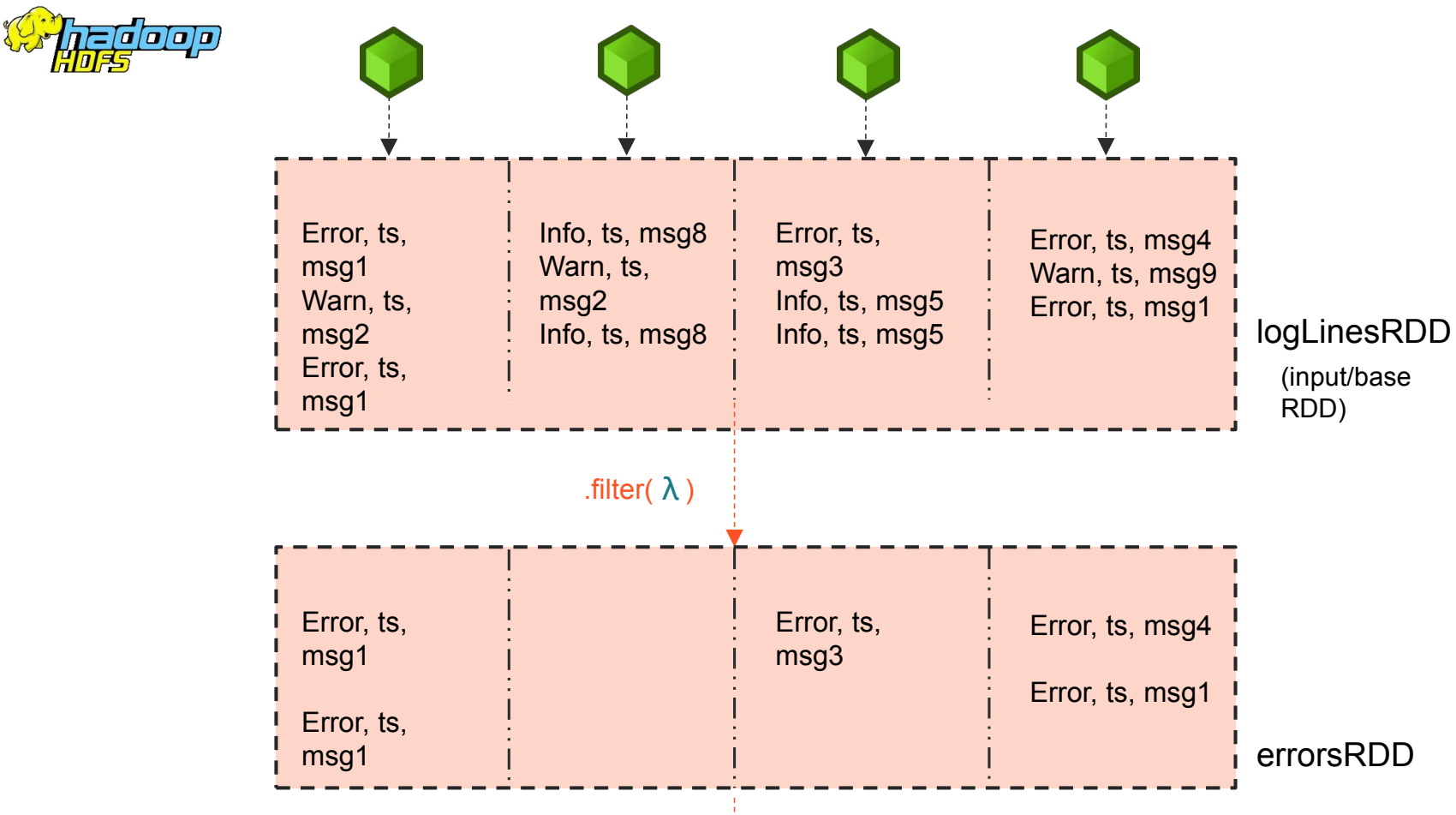
```
# Read a local txt file in Python
linesRDD = sc.textFile("/path/to/README.md")
```

Lecture depuis un fichier texte
D'autres méthodes existent pour lire depuis HDFS, S3, Hbase …

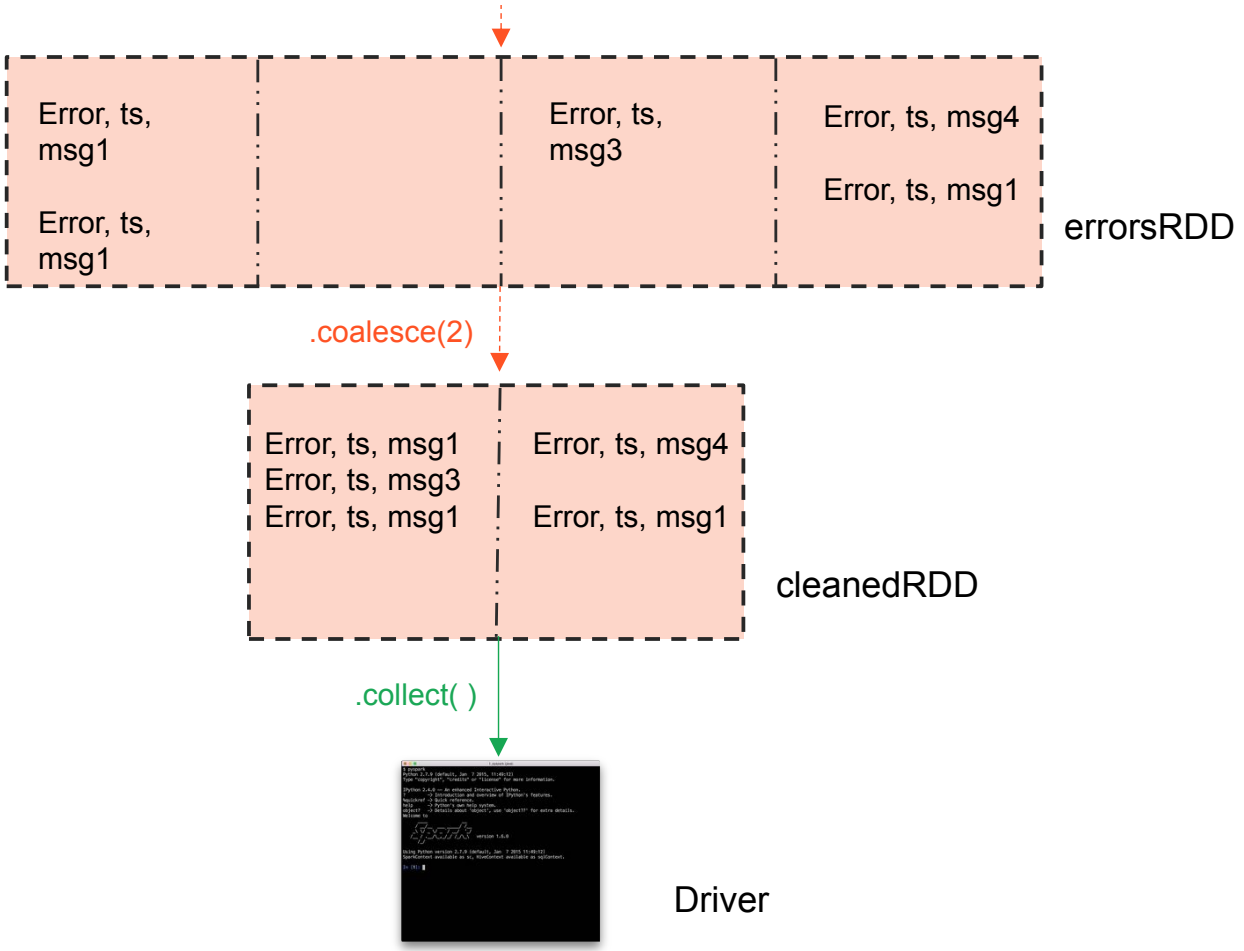
Opérations sur les RDDs

- Transformations : évaluation paresseuse
  - Map, filter, intersection, groupByKey, reduceByKey, zipWithIndex …
- Actions : déclenchent l'exécution des transformations
  - Collect, count, reduce, saveAsTextFile …
- Mise en cache
  - Évite l'exécution répétée de transformations pour réutiliser une RDD

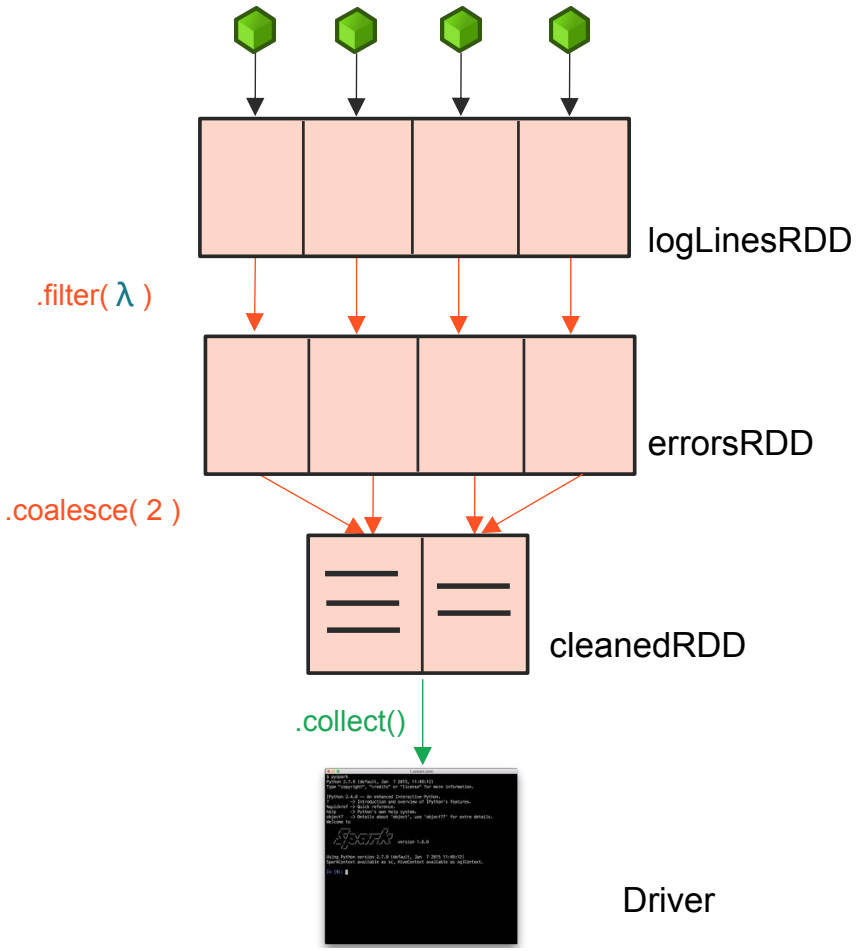
Exemple de RDD



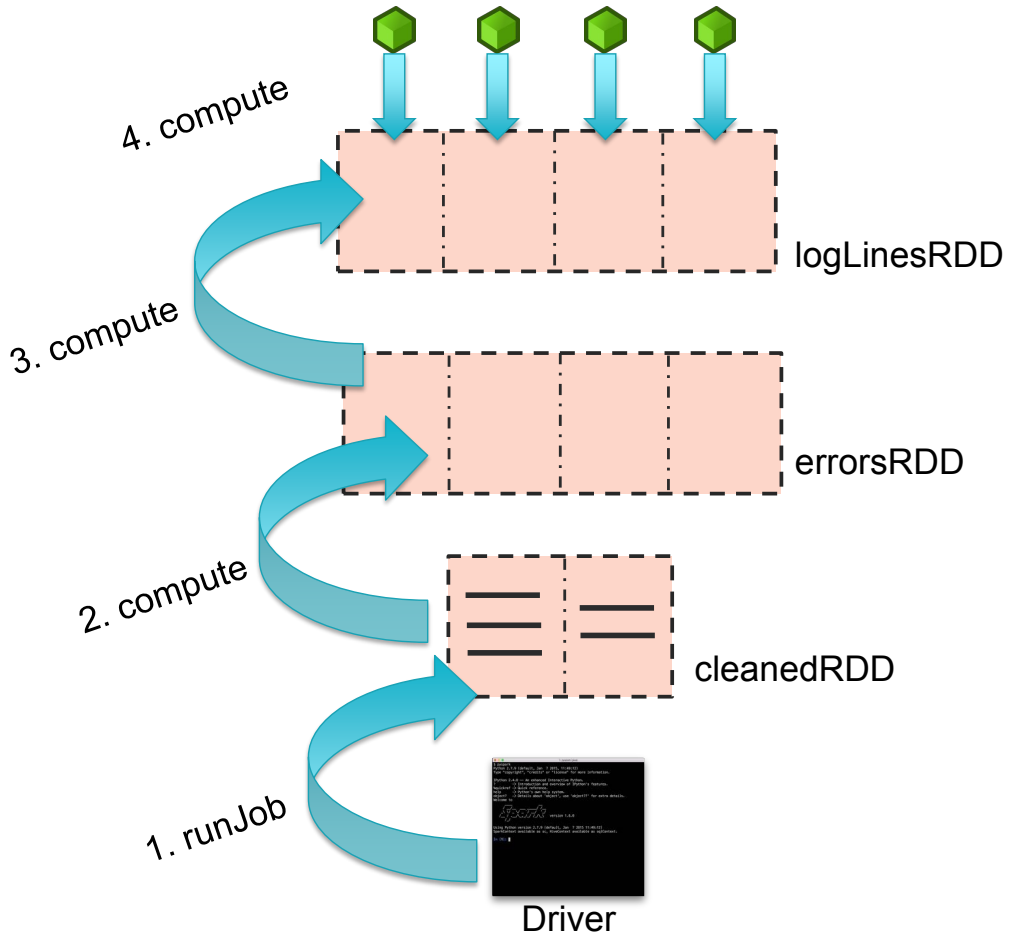
Exemple de RDD



Lignée d'une RDD



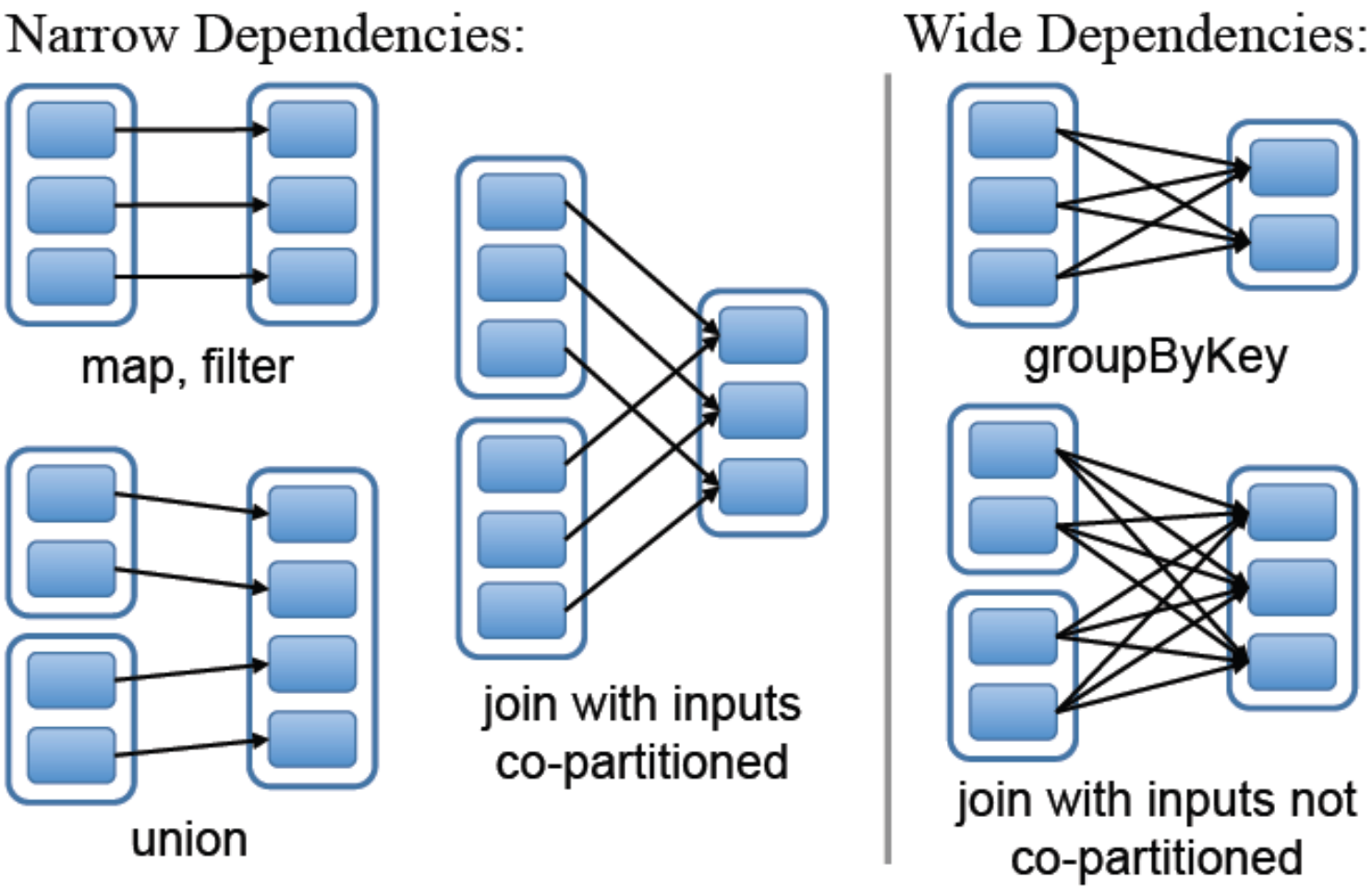
Exécution


4. compute
logLinesRDD
3. compute
errorsRDD
2. compute
cleanedRDD
1. runJob
Driver

Tolérance aux pannes

- Hadoop : politique pessimiste, en cas de crash on perd peu mais coûts d'I/O élevés pour toutes les exécutions
- Spark : politique optimiste, on ne paie pas la tolérance aux pannes, mais en cas de crash on doit recalculer en utilisant la lignée de la RDD

Dépendances de RDD



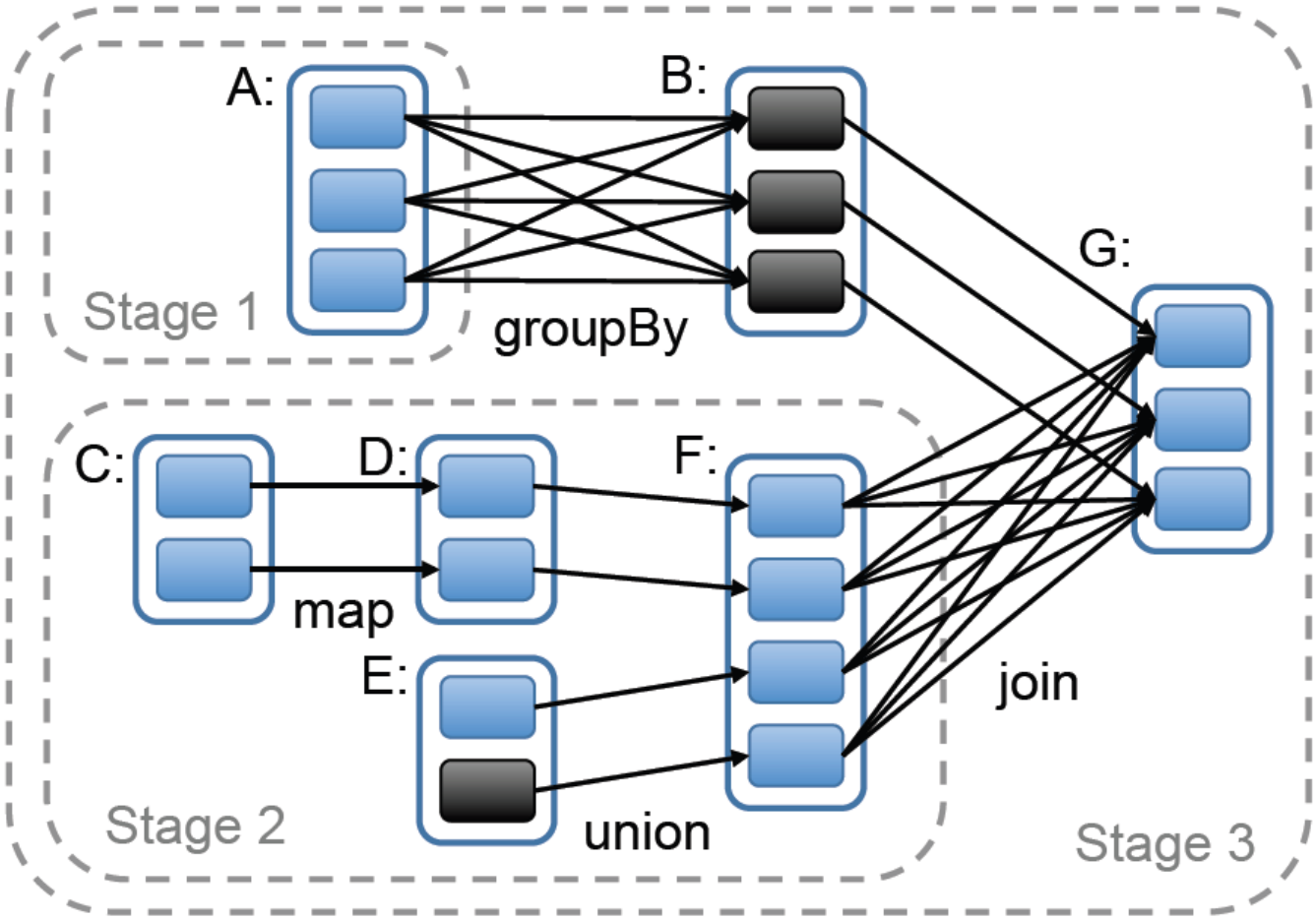
Dépendances de RDD

- Dépendances étroites (narrow)
  - Permet d'enchainer l'exécution sur un nœud du cluster, sans barrière
  - Crash peu coûteux, reconstruction isolée
- Dépendances larges
  - Nécessitent les données de toutes les partitions parentes, barrière dans l'exécution
  - Crash coûteux, il faut reconstruire les données de toutes les machines

Ordonnancement de l'exécution

- Pour exécuter des actions sur une RDD
  - L'ordonnanceur découpe l'exécution en étapes en utilisant la lignée de la RDD
  - Chaque étape contient autant d'opérations à dépendance étroite que possible

Ordonnancement d'exécution



Exemple de RDD

- Shell Spark interactif

```
scala> val wc = lesMiserables.flatMap(_.split(" ")).map((_,1)).reduceByKey(_+_)
wc: org.apache.spark.rdd.RDD[(String, Int)] = ShuffledRDD[5] at reduceByKey at <console>:14
scala> wc.take(5).foreach(println)
(créanciers;,1)
(abondent.,1)
(plaisir,,5)
(déplaçaient,1)
(sociale,,7)
scala> val cw = wc.map(p => (p._2, p._1))
cw: org.apache.spark.rdd.RDD[(Int, String)] = MappedRDD[5] at map at <console>:16
scala> val sortedCW =  cw.sortByKey(false)
sortedCW: org.apache.spark.rdd.RDD[(Int, String)] = ShuffledRDD[11] at sortByKey at <console>:18
scala> sortedCW.take(5).foreach(println)
(16757,de)
(14683,)
(11025,la)
(9794,et)
(8471,le)
scala> sortedCW.filter(x => "Cosette".equals(x._2)).collect.foreach(println)
(353,Cosette)
```

Wordcount en Spark

DataFrames

Définition d'une DataFrame

- Collection de données
  - Organisées en colonnes
  - Schéma des données
    → proche des RDBMS
- Implémentation
  - Repose sur des RDD (tolérance aux pannes …)
  - Plus haut niveau, langage type SQL
    → permet d'avantage d'optimisations
    (ordonnancement des opérations, projet Tungsten …)

Exemple de DataFrame

```scala
// A JSON dataset is pointed to by path.
// The path can be either a single text file or a directory storing text files
val path = "examples/src/main/resources/people.json"
val peopleDF = spark.read.json(path)

// The inferred schema can be visualized using the printSchema() method
peopleDF.printSchema()
// root
//  |-- age: long (nullable = true)
//  |-- name: string (nullable = true)

// Creates a temporary view using the DataFrame
peopleDF.createOrReplaceTempView("people")

// SQL statements can be run by using the sql methods provided by spark
val teenagerNamesDF = spark.sql("SELECT name FROM people WHERE age BETWEEN 13 AND 19")
teenagerNamesDF.show()
// +------+
// |  name|
// +------+
// |Justin|
// +------+
```

Quelques notions de Scala

Scala

- Langage de programmation fonctionnelle
  - Typage fort et statique
  - Compilation en ByteCode Java
  - Interopérable avec Java
- Langage développé à l'EPFL
  - Adopté par de grands projets open-source (Spark …) et industriels (Twitter …)

Exemples de Scala dans le contexte de Spark

```scala
package uga.tpspark.flickr

import org.apache.spark.sql.types.IntegerType

object Demo {
  def main(args: Array[String]): Unit = {
    println("hello")
    var spark: SparkSession = null
    try {
      spark = SparkSession.builder().appName("Flickr using dataframes").getOrCreate()
      val textFile: RDD[String] = spark.sparkContext.textFile("textFile.txt")
      val lineLength: RDD[Int] = textFile.map(line => line.length())
      val lineWithNumber: RDD[(String, Long)] = textFile.zipWithIndex()
      val lineNumberWithLength: RDD[(Long, Int)] = lineWithNumber.map(f => (f._2, f._1.length()))
      val allWords: RDD[String] = textFile.flatMap(line => line.split("\\s+"))
    } catch {
      case e: Exception => throw e
    } finally {
      spark.stop()
    }
    println("done")
  }
}
```

Classe Scala

```scala
package uga.tpspark.flickr

class Picture(fields: Array[String]) extends Serializable {
  val lat: Double = fields(11).toDouble
  val lon: Double = fields(10).toDouble
  val c: Country = Country.getCountryAt(lat, lon)
  val userTags: Array[String] = java.net.URLDecoder.decode(fields(8), "UTF-8").split(",")

  def hasValidCountry: Boolean = {
    c != null
  }

  def hasTags: Boolean = {
    userTags.size > 0 && !(userTags.size == 1 && userTags(0).isEmpty())
  }

  override def toString: String = "(" + c + ", " + userTags.reduce((t1: String, t2: String) => t1 + ", " + t2) + ")"

}
```