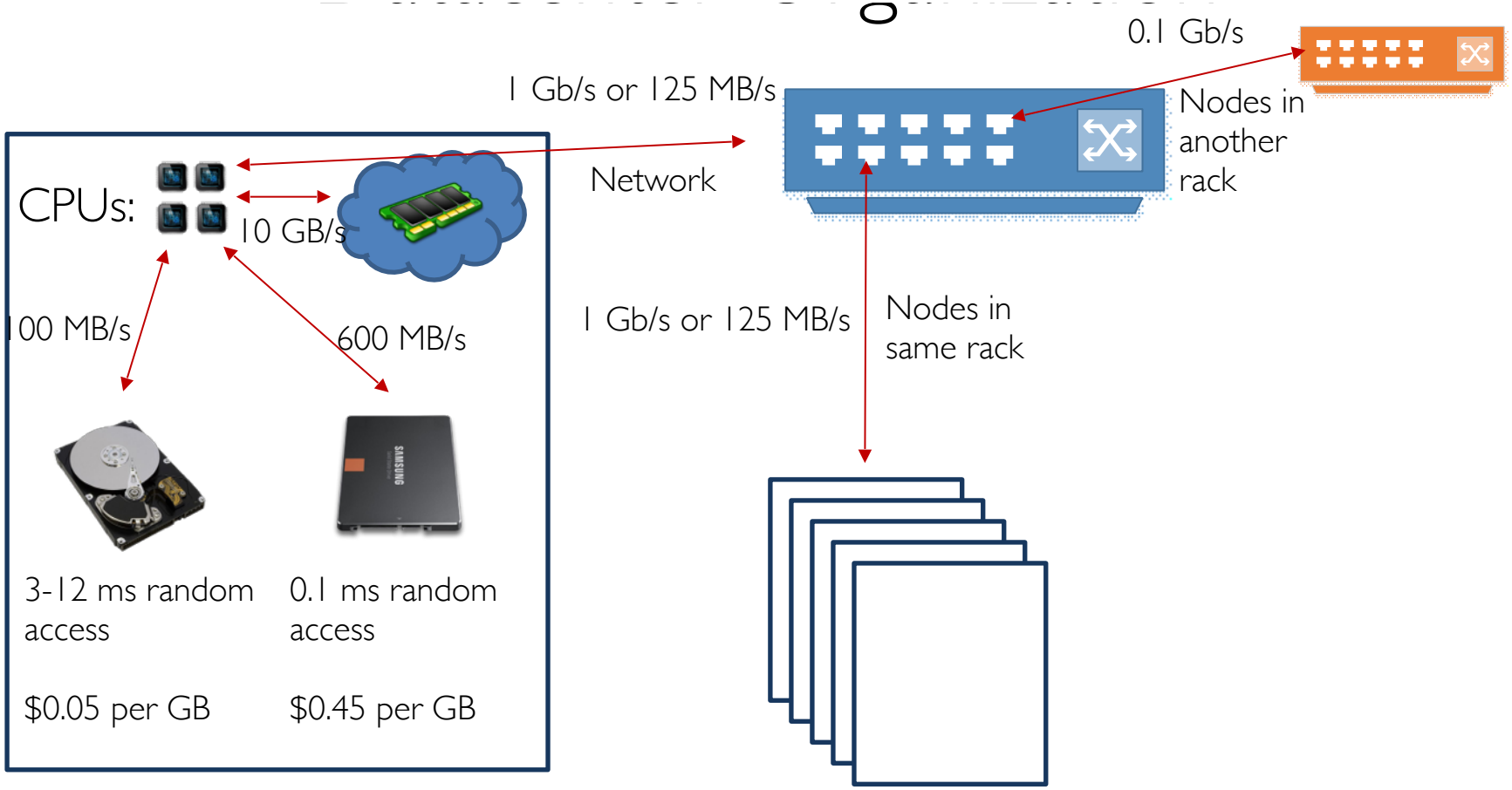
In-Memory Processing with Apache Spark

Vincent Leroy

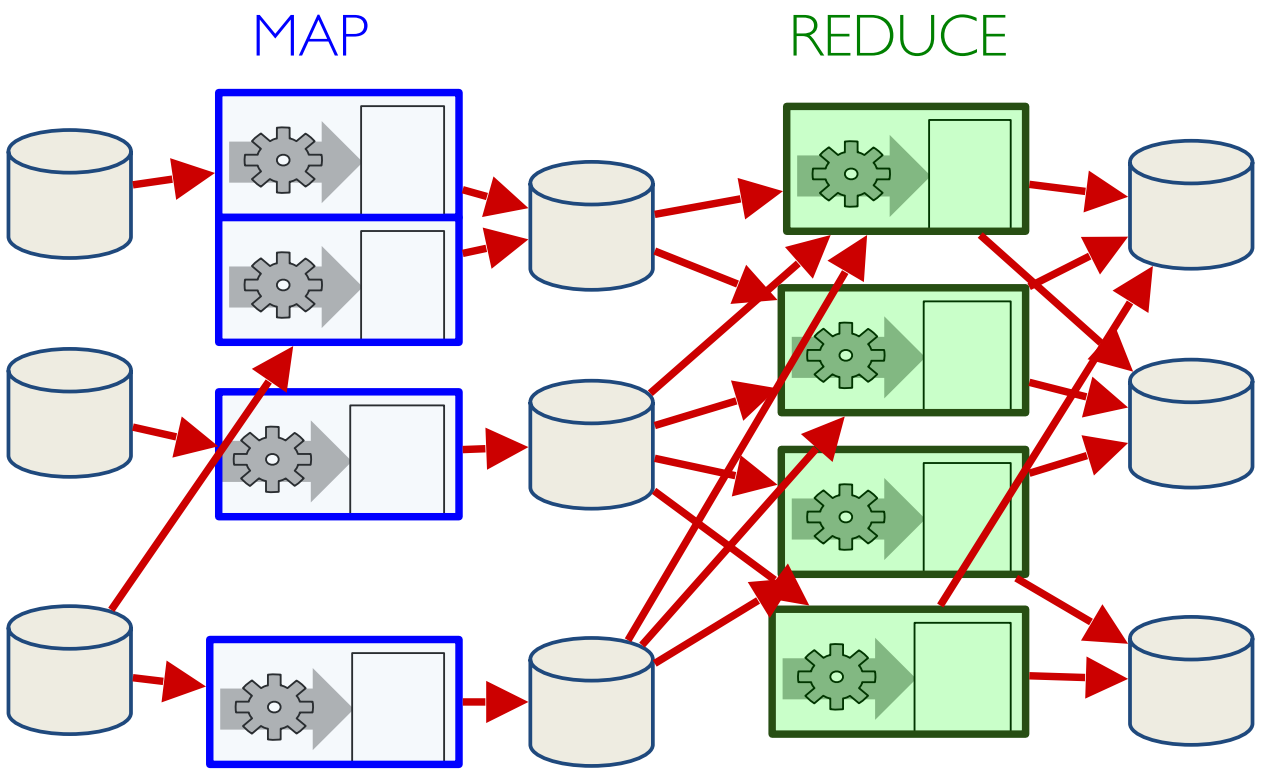
Sources

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

Datacenter Organization



MapReduce Execution

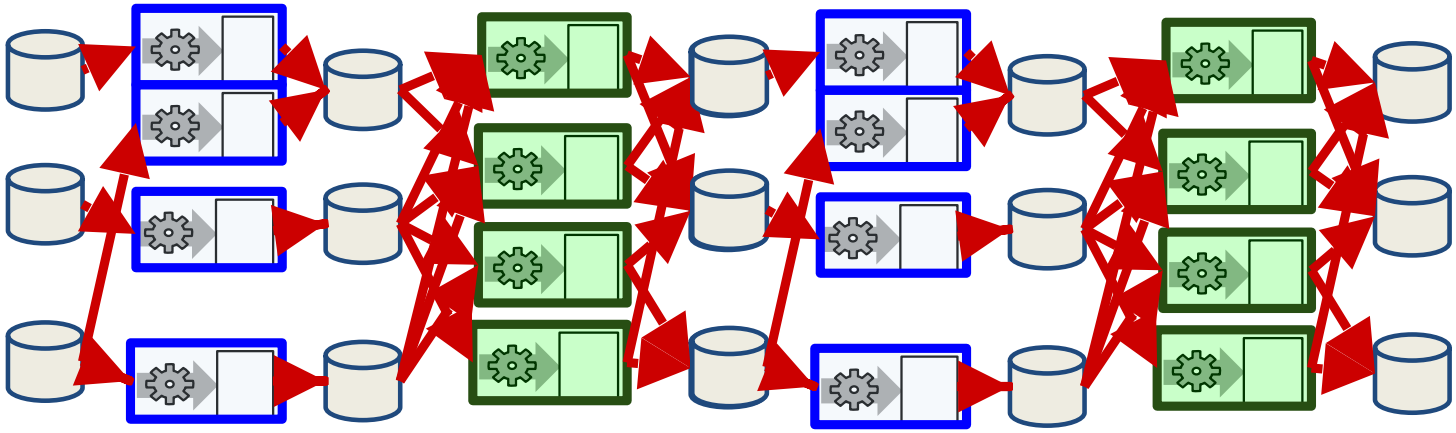


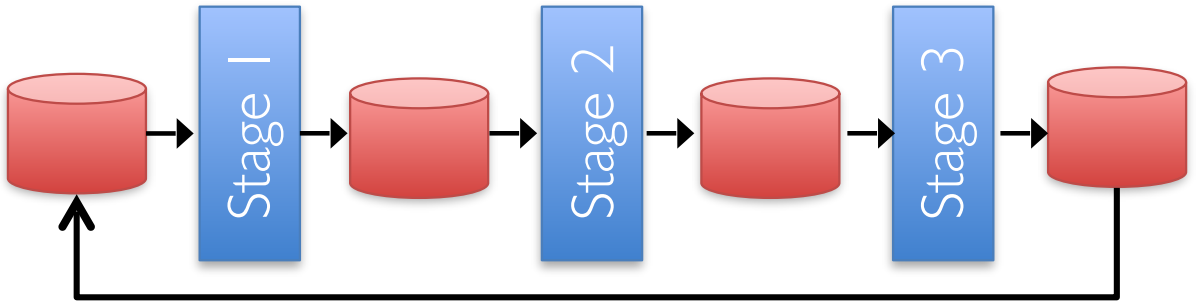
Each stage passes through the hard drives

Iterative Jobs

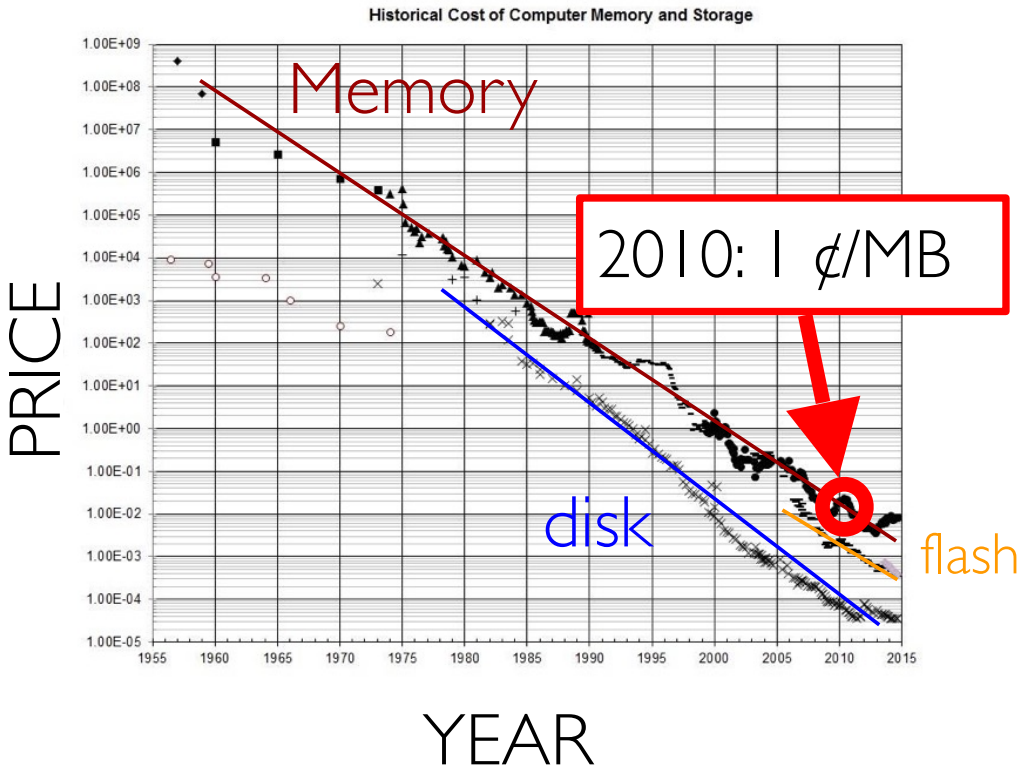
- Disk I/O for each repetition

  → Slow when executing many small iterations





Memory Cost



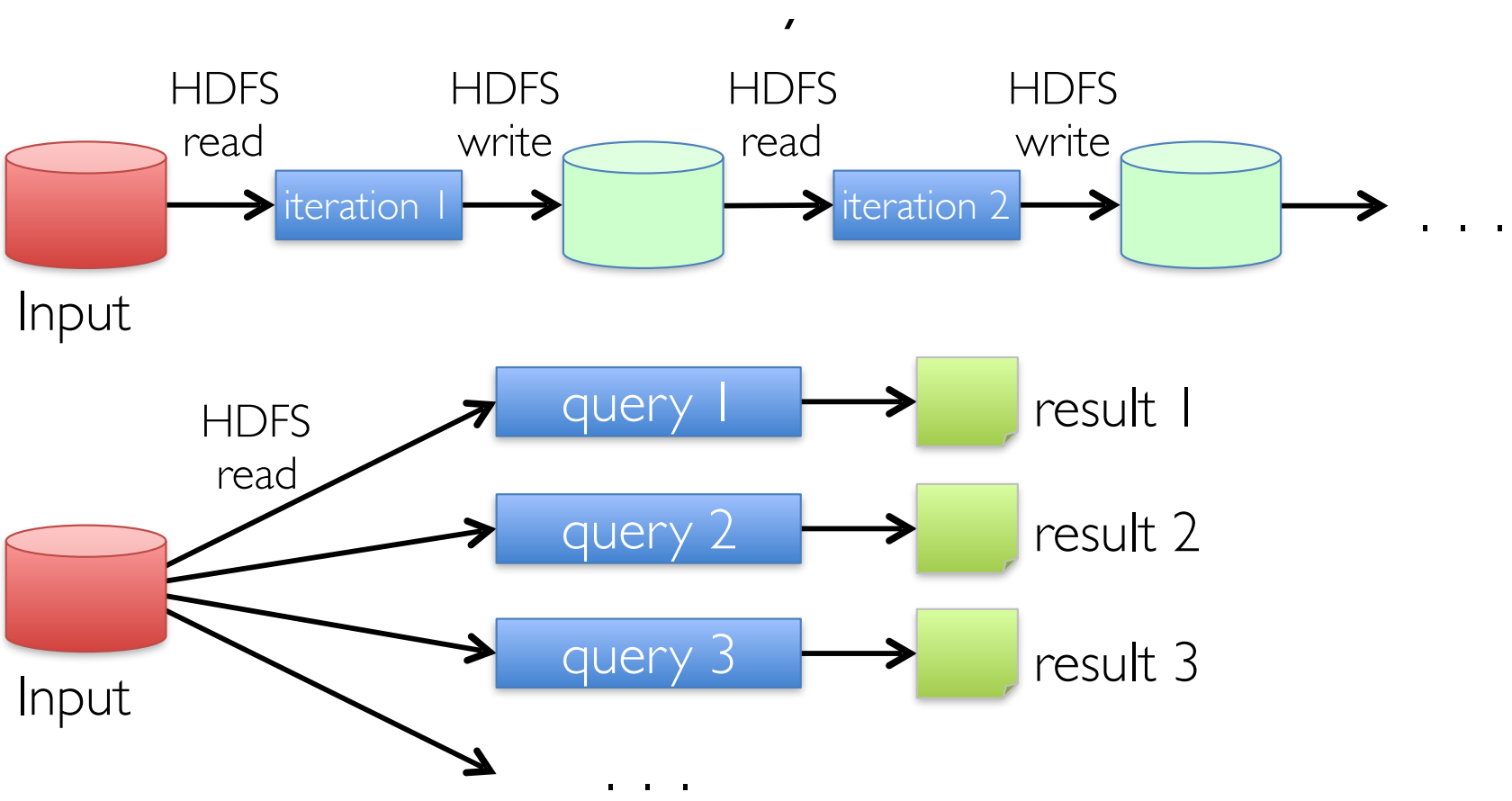
Lower cost means can put more memory in each server

In-Memory Processing

- Many datasets fit in memory (of a cluster)
- Memory is fast and avoid disk I/O

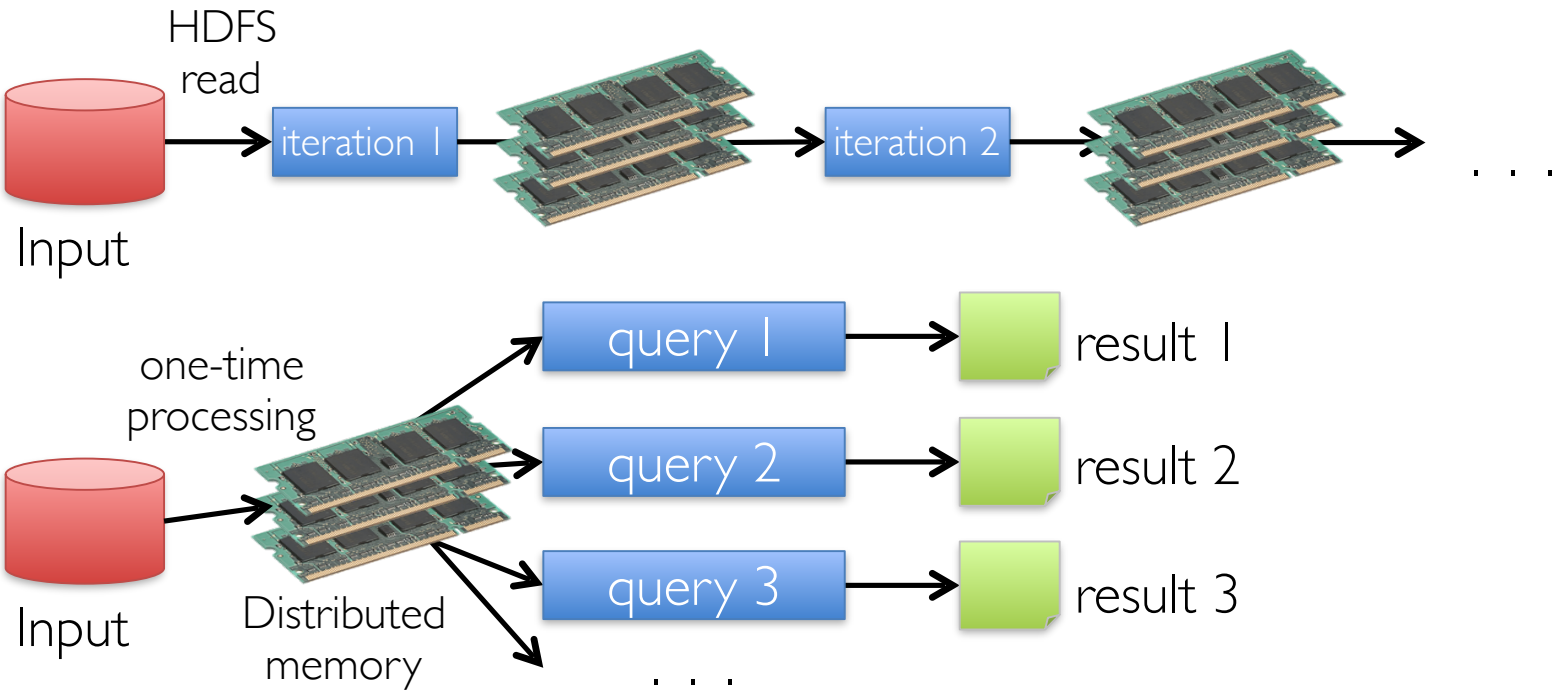
→ Spark distributed execution engine

Replace Disk with Memory



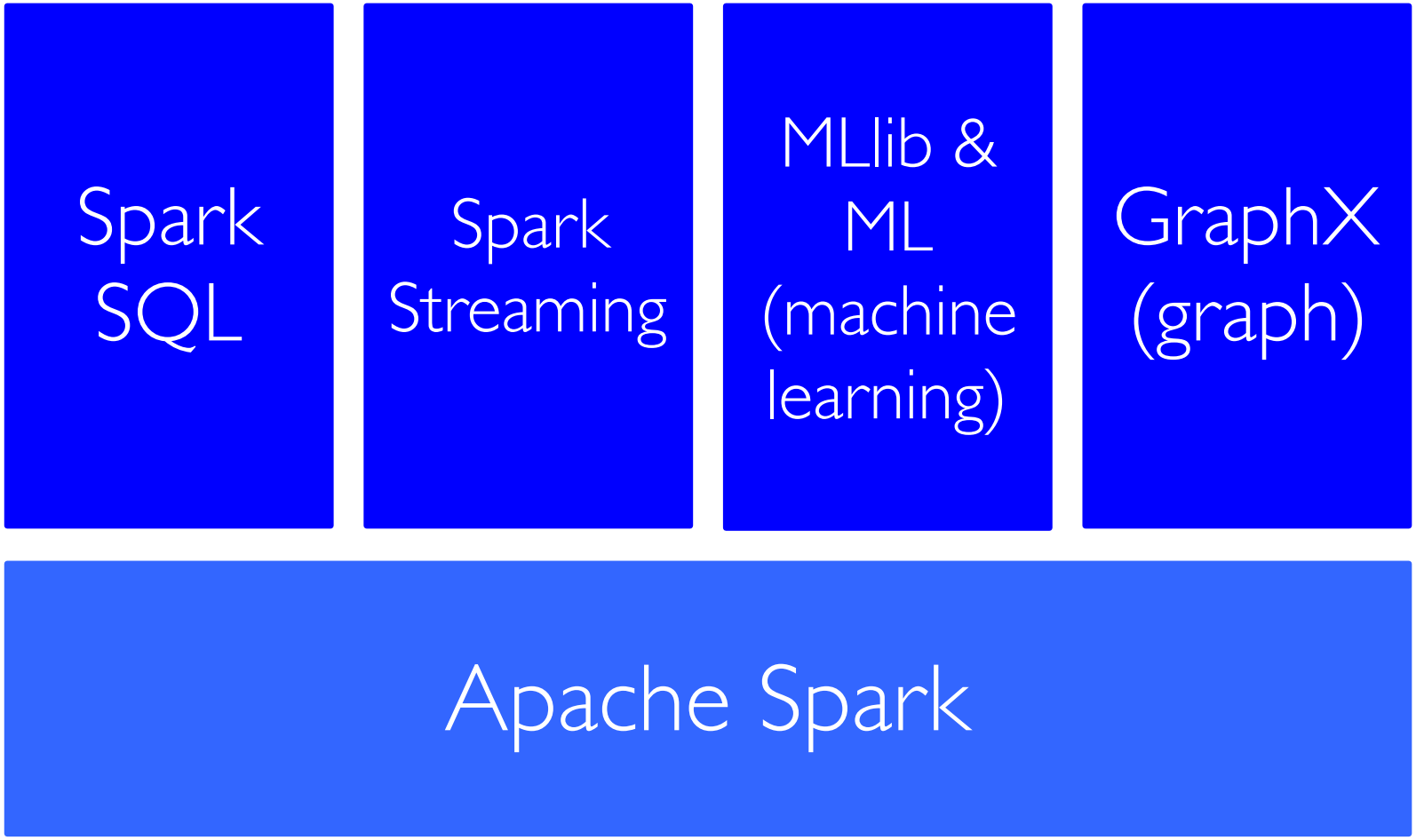
Replace Disk with Memory

In-Memory Data Sharing



10-100x faster than network and disk

Spark Architecture



Resilient Distributed Datasets (RDDs)

Resilient Distributed Datasets (RDDs)

- Data Collection
  - Distributed
  - Read-only
  - In-memory
  - Built from stable storage or other RDDs

RDD Creation

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

Parallelize

Take an existing in-memory collection and pass it to SparkContext's parallelize method

---

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

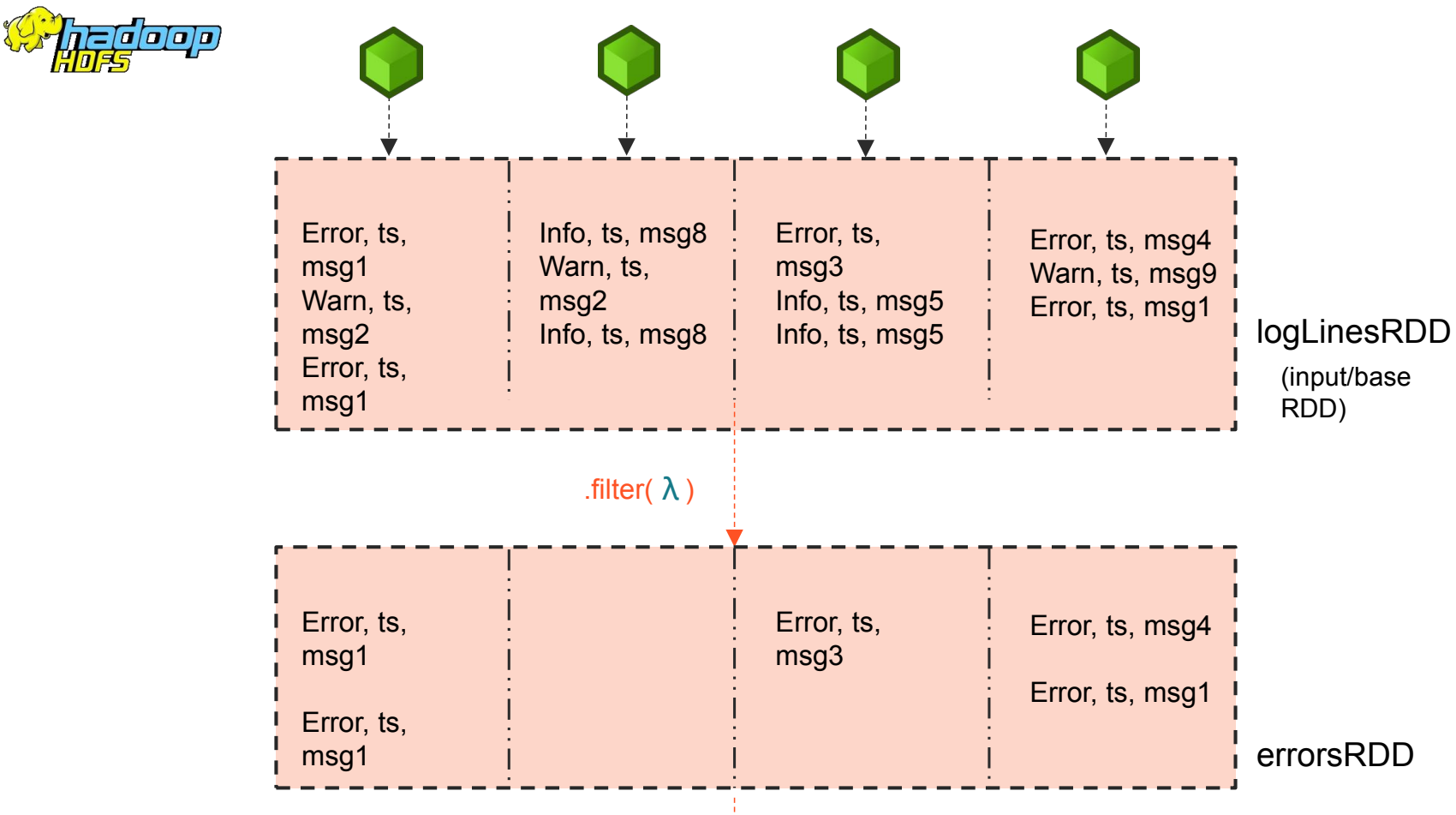
Read from Text File

There are other methods to read data from HDFS, C*, S3, HBase, etc.

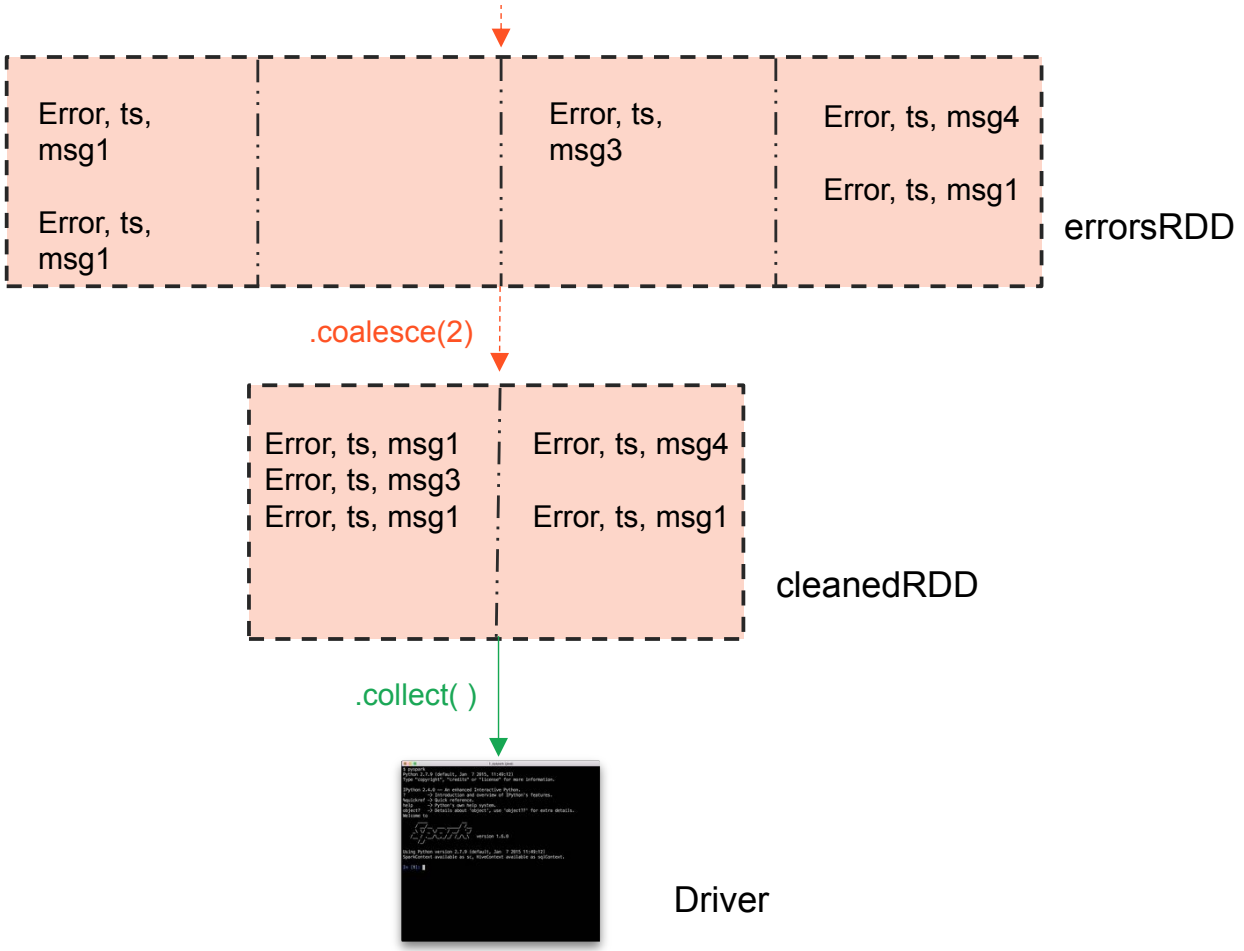
Operations on RDDs

- Transformations: lazy execution
  - Map, filter, intersection, groupByKey, zipWithIndex …
- Actions: trigger execution of transformations
  - Collect, count, reduce, saveAsTextFile …

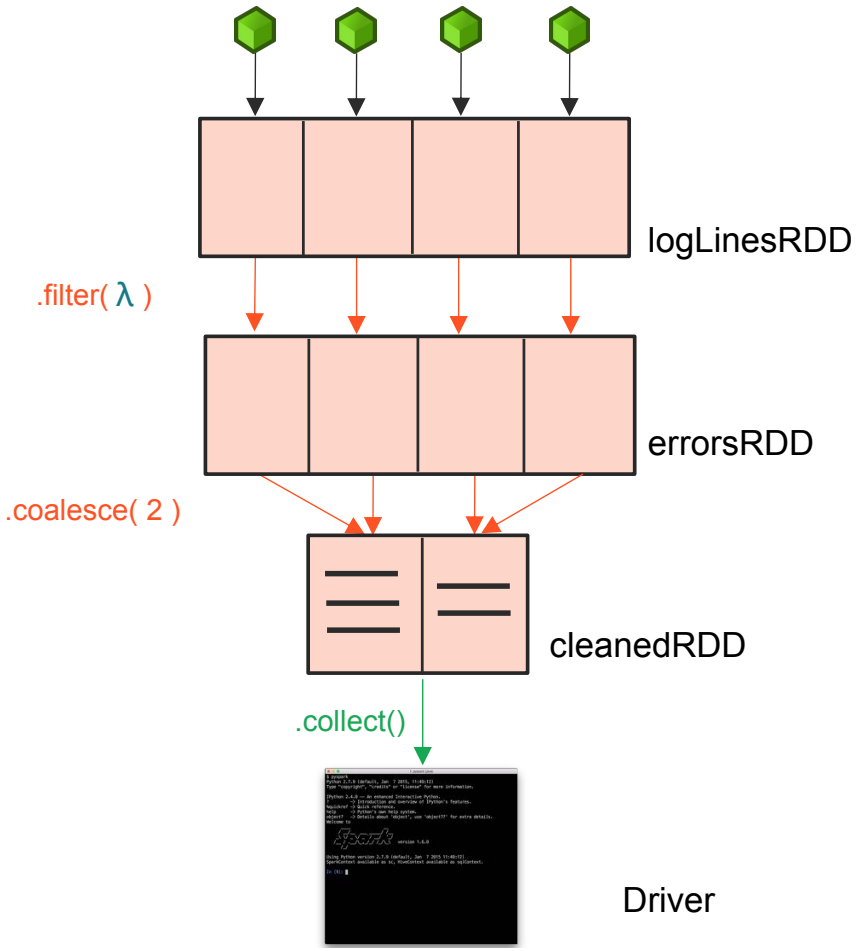
RDD Example


hadoop
HDFS
Error, ts, msg1
Warn, ts, msg2
Error, ts, msg1
Info, ts, msg8
Warn, ts, msg2
Info, ts, msg8
Error, ts, msg3
Info, ts, msg5
Info, ts, msg5
Error, ts, msg4
Warn, ts, msg9
Error, ts, msg1
logLinesRDD
(input/base RDD)
.filter(λ)
Error, ts, msg1
Error, ts, msg1
Error, ts, msg3
Error, ts, msg4
Error, ts, msg1
errorsRDD

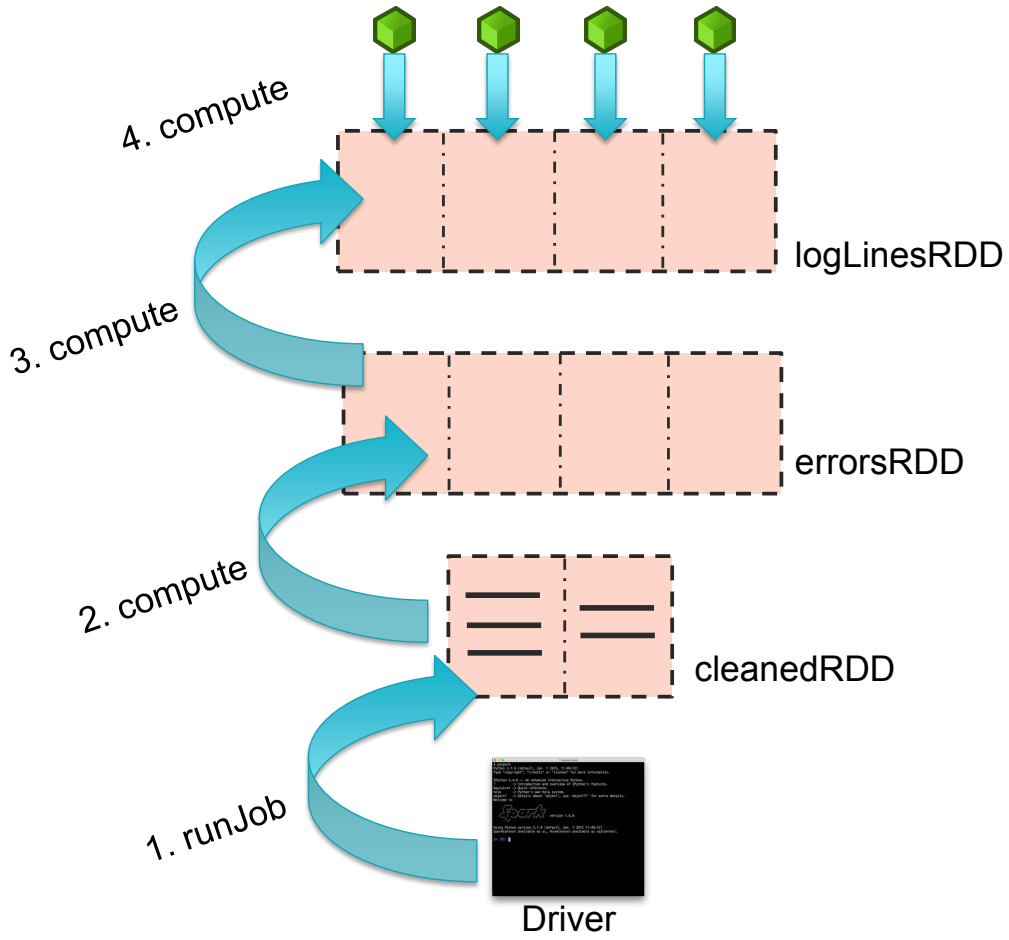
RDD Example


Error, ts, msg1
Error, ts, msg1
Error, ts, msg3
Error, ts, msg4
Error, ts, msg1
errorsRDD
.coalesce(2)
Error, ts, msg1
Error, ts, msg3
Error, ts, msg1
Error, ts, msg4
Error, ts, msg1
cleanedRDD
.collect( )
Driver

RDD Lineage


logLinesRDD
.filter(λ)
errorsRDD
.coalesce(2)
cleanedRDD
.collect()
Driver

Execution


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

RDD Fault Tolerance

- Hadoop conservative/pessimistic approach

  →Go to disk / stable storage (HDFS)

- Spark optimistic

  →Don't "waste" time writing to disk, re-compute in case of crash using lineage

Caching

- Lazy execution
  - Process all transformations in a chain until action
  - Does not store intermediate results!
- Multiple actions on the same RDD
  - Re-compute RDD
- Caching
  - Avoids re-computing by storing a copy in memory/on disk

Narrow Dependencies: the Map family

- map(*f* X → Y): transform elements of a RDD using *f*
  - 1 for 1 tranformation
- flatMap(*f* X → Iterable[Y]): transform elements of RDD using *f*
  - 1 to many transformation (like Hadoop map)
- filter(*f* X→ Boolean): keep only elements for which *f* returns true
- …

Narrow dependencies demo

```
val textRdd: RDD[String] = sc.parallelize(Array("hello", "spark exercise"))
val upText: RDD[String] = textRdd.map(text => text.toUpperCase())
upText.take(10).foreach(println)
//HELLO
//SPARK EXERCISE
val wordRdd: RDD[String] = textRdd.flatMap(text => text.split("\\s"))
wordRdd.take(10).foreach(println)
//hello
//spark
//exercise
val longTextRdd = textRdd.filter(text => text.length() > 8)
longTextRdd.take(10).foreach(println)
//spark exercise
```

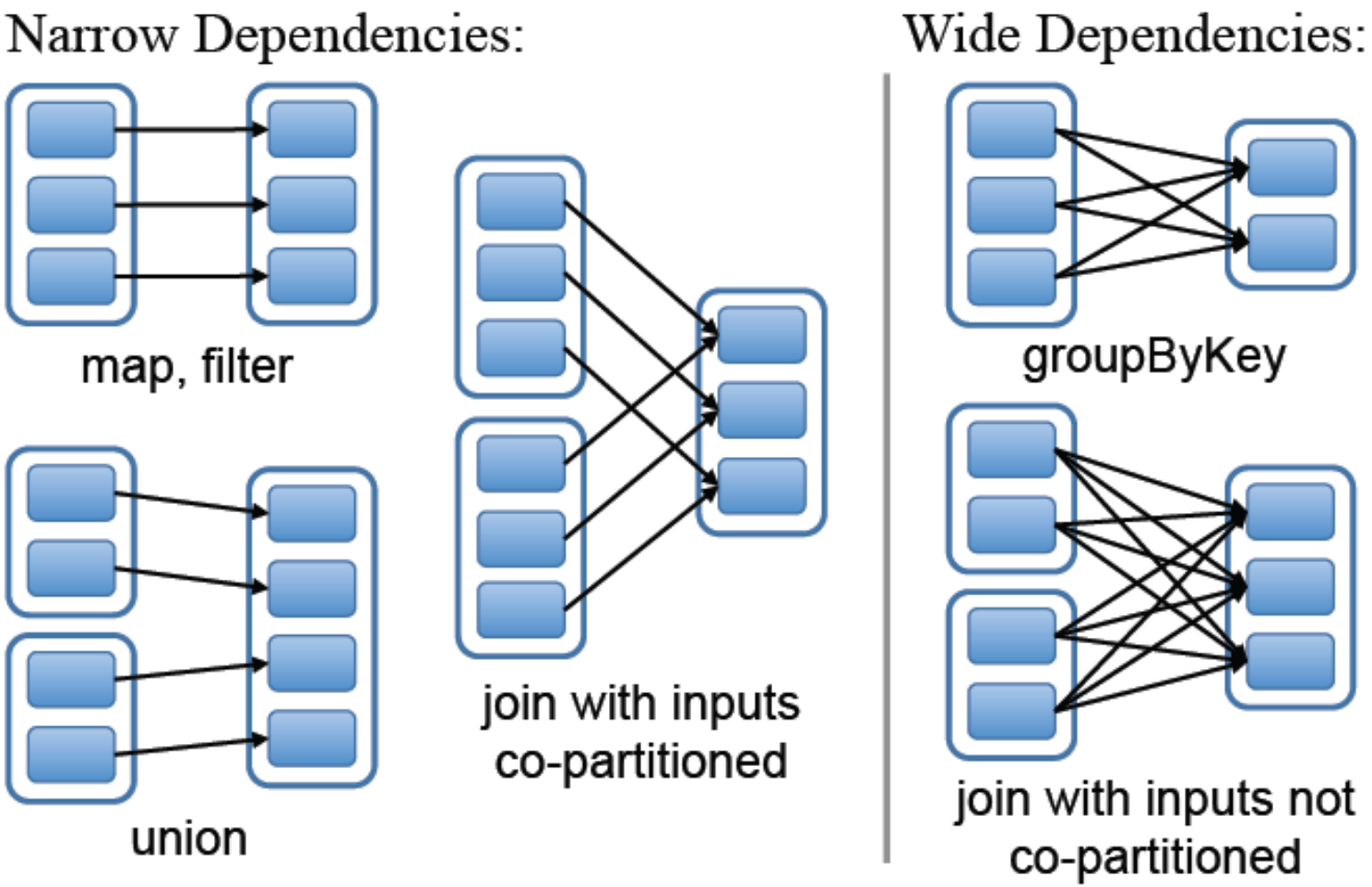
Wide Dependencies: the Reduce family

- reduceByKey(*f* X,X → X): aggregate values having the same key
  - Input and output have the same type
- groupByKey: group values having the same key
  - Similar to Hadoop's shuffle and sort phase
- combineByKey(*f1,f2,f3*): general form of reduceByKey where output can have a different type
- …

Wide dependencies demo

```
val pairRdd: RDD[(String, Int)] = sc.parallelize(Array(("hello", 1), ("spark", 1), ("hello", 2)))
val sumByKey = pairRdd.reduceByKey((x, y) => x + y)
sumByKey.take(10).foreach(println)
//(hello,3)
//(spark,1)
val gByK = pairRdd.groupByKey()
gByK.take(10).foreach(println)
//(hello,CompactBuffer(1, 2))
//(spark,CompactBuffer(1))
val gToS = pairRdd.combineByKey(x => Set(x),
  (s: Set[Int], x: Int) => s + x,
  (s1: Set[Int], s2: Set[Int]) => s1 ++: s2)
gToS.take(10).foreach(println)
//(hello,Set(1, 2))
//(spark,Set(1))
```

RDD Dependencies



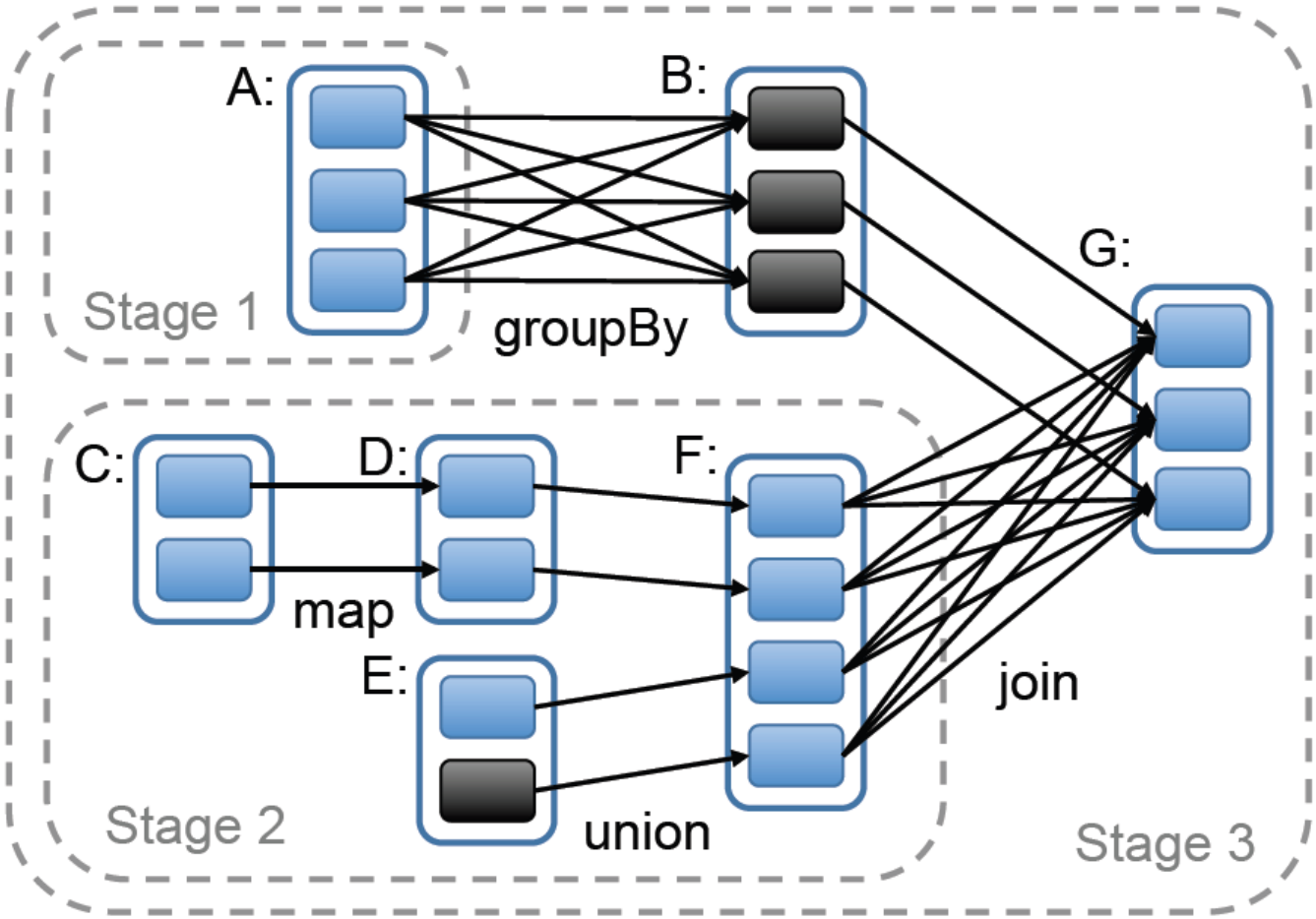
RDD Dependencies

- Narrow dependencies
  - allow for pipelined execution on one cluster node
  - easy fault recovery
- Wide dependencies
  - require data from all parent partitions to be available and to be shuffled across the nodes
  - a single failed node might cause a complete re-execution.

Job Scheduling

- To execute an action on an RDD
  - scheduler decide the stages from the RDD's lineage graph
  - each stage contains as many pipelined transformations with narrow dependencies as possible

Job Scheduling



Spark Interactive Shell

```
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

DataFrames

Definition

- Data collection
  - Organized in columns
  - Has a schema (column name and type)
    → similar to RDBMS
- Implementation
  - Relies on RDDs (fault tolerance …)
  - Allows higher level languages like SQL
    → query optimizer (pushing selects …)

Exemple of DataFrame on JSON

```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|
// +------+
```

Exemple of DataFrame on CSV

```
val originalFlickrMeta = spark.sqlContext.read
  .format("csv")
  .option("delimiter", "\t")
  .option("header", "false")
  .schema(customSchemaFlickrMeta)
  .load("/Users/vleroy/Documents/cours/BigData/TPIntroHadoop/flickrSample.txt")
println("nb lines " + originalFlickrMeta.count())
originalFlickrMeta.createOrReplaceTempView("flickr_meta")
val interestingMetaLines = spark.sql("""SELECT photo_id, license, longitude, latitude
     FROM flickr_meta WHERE license IS NOT NULL and longitude != -1.0""")
```

A few notions of Scala

Scala

- Functional programming language
  - Static type
  - Compiles to Java ByteCode
  - Can be mixed with Java
- Developed at EPFL
  - Used by many industrial and open-source systems (Spark, Twitter, Swisscom)

Sample Spark/Scala code

```scala
object Ex2RDD {
  def main(args: Array[String]): Unit = {
    println("hello")
    var spark: SparkSession = null
    try {
      spark = SparkSession.builder().master("local[4]").appName("Flickr using dataframes").getOrCreate()
      val originalFlickrMeta: RDD[String] = spark.sparkContext.textFile("/Users/vleroy/Documents/cours/BigData/TPIntroHadoop/flickrSample.txt")
      originalFlickrMeta.take(5).foreach(println)
      println("nb lines " + originalFlickrMeta.count())
      val pictures: RDD[Picture] = originalFlickrMeta.map(line => line.split("\t")).map(t => new Picture(t)).filter(c => c.hasValidCountry && c.hasTags)
      pictures.take(5).foreach(println)
      val picturesByCountry: RDD[(Country, Iterable[Picture])] = pictures.groupBy(p => p.c)
      picturesByCountry.take(5).foreach(println)
      val tagsByCountry: RDD[(Country, Iterable[String])] = picturesByCountry.map(cp => (cp._1, cp._2.flatten(p => p.userTags)))
      tagsByCountry.take(5).foreach(println)
      val tagFreqByCountry: RDD[(Country, Map[String, Int])] = tagsByCountry.map(cp => (cp._1, cp._2.groupBy(identity).mapValues(_.size).map(identity)))
      tagsByCountry.take(5).foreach(println)
    } catch {
      case e: Exception => throw e
    } finally {
      spark.stop()
    }
    println("done")
  }
}
```