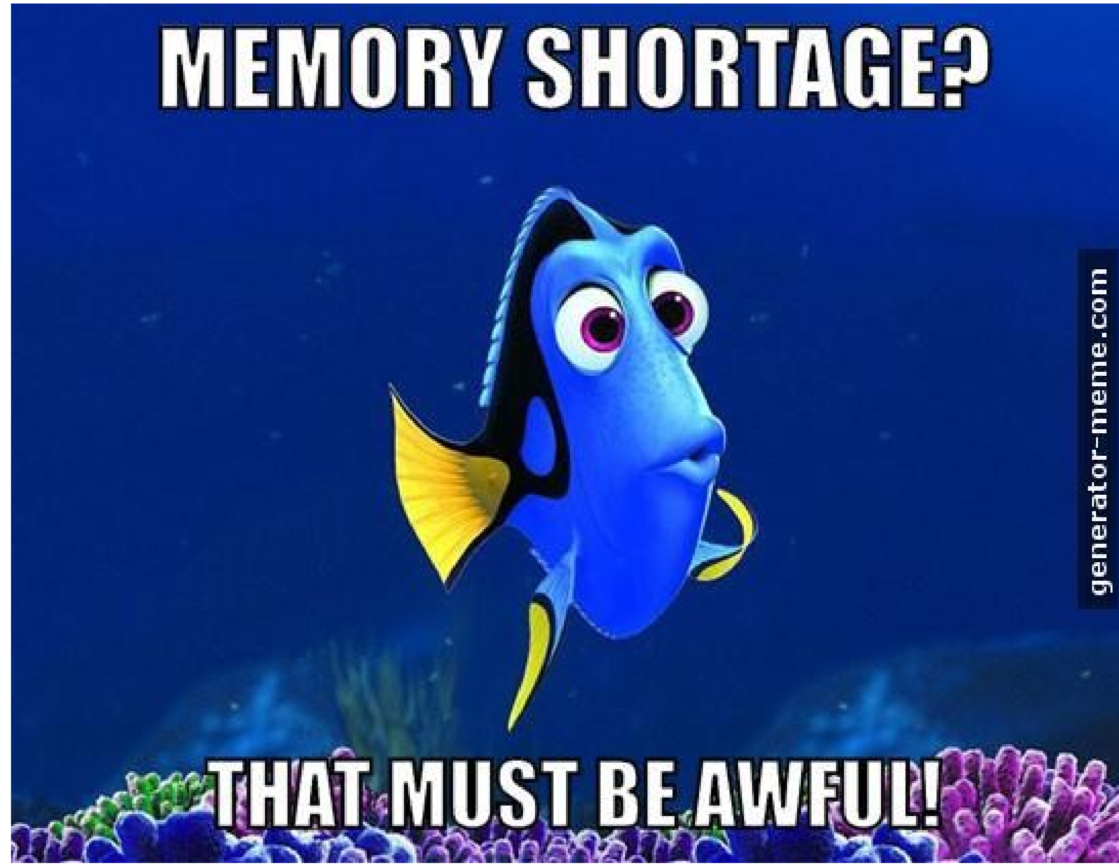
Bases de datos en memoria



Bases de Datos No Relacionales

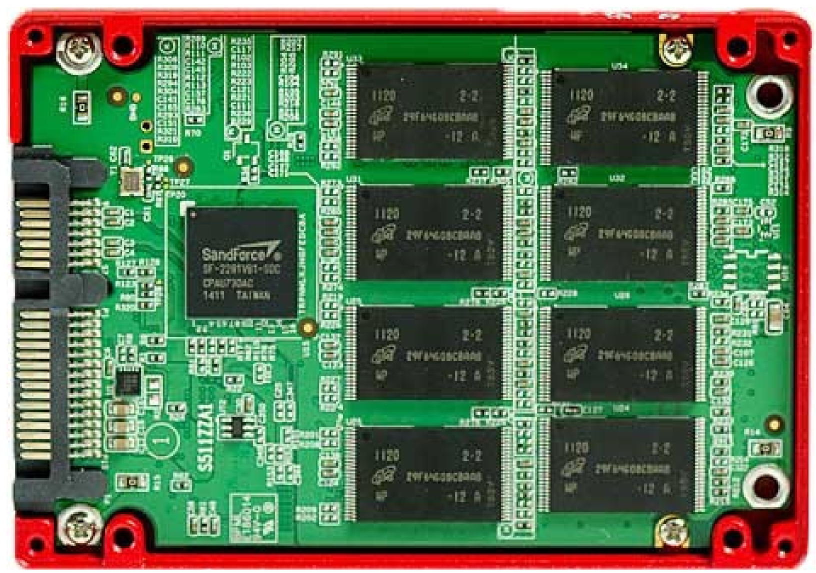
Instituto de Computación, FING, UdelaR – 2022

Agenda

- SSDs vs HDDs
- Bases de datos sobre SSDs
- Bases de datos en memoria
- Apache Spark

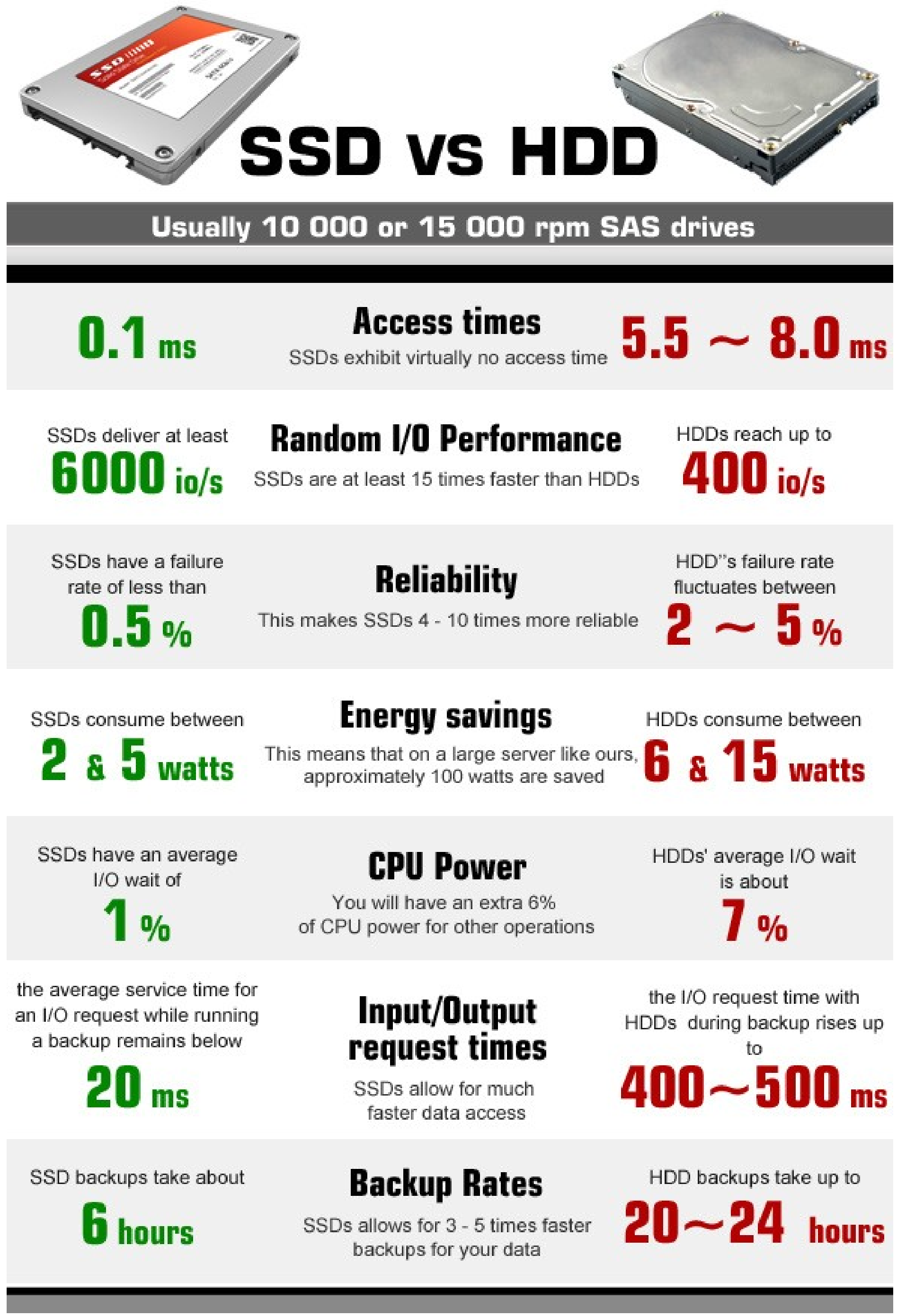
Los discos magnéticos (HDD)

Los discos de estado sólido (SSD)

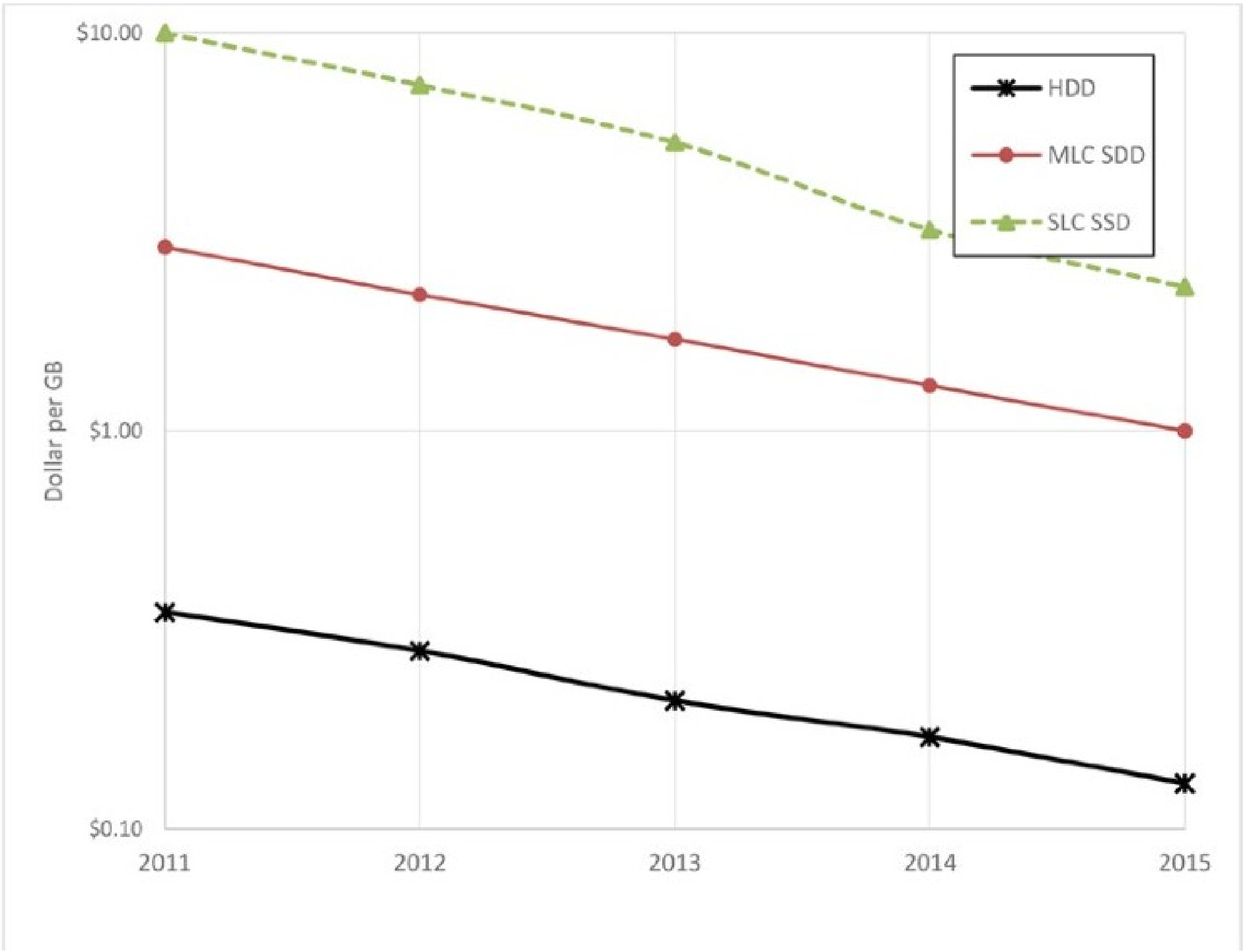


No tienen partes móviles (NAND flash o DDR RAM)

Organizados en celdas (1 o 2 bit), agrupadas en páginas (4K) que se organizan en bloques (256 páginas)



http://technofaq.org/posts/2014/11/ssd-hosting-or-hdd-hosting-which-one-to-choose/



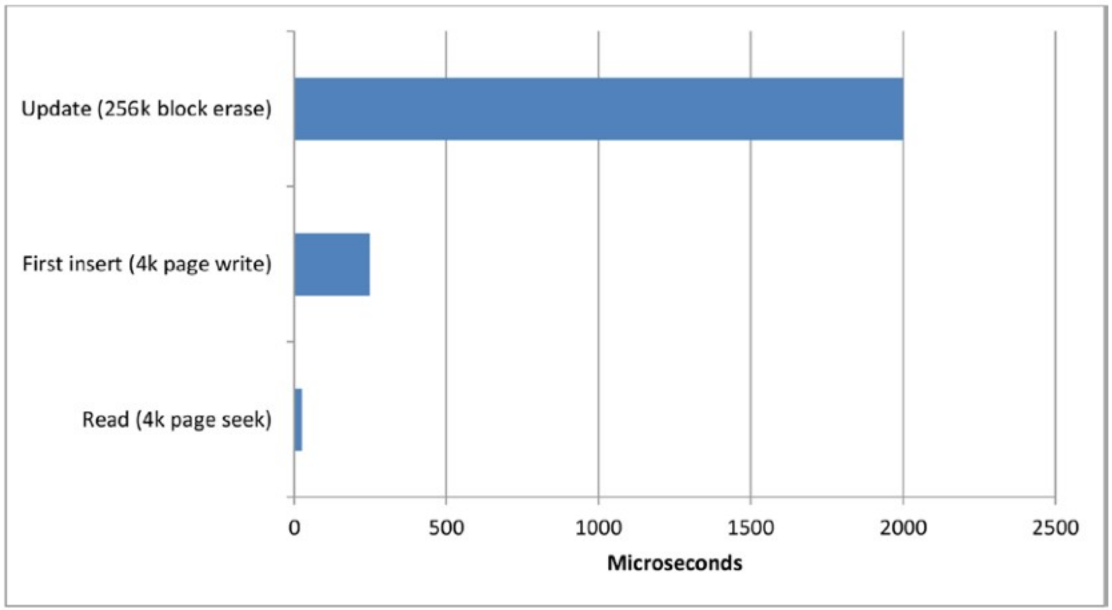
Tendencias en precios HDD vs SDD (escala logarítmica)

Fuente: Next Generation Databases, Harrison, Apress 2015

	PCIe SSD	2.5" MLC SSD	2.5" TLC SSD	3.5" SATA HDD
Density	Up to 6.4TB	4TB	1TB	10TB
Read Bandwidth [MB/s]	~3000	500/1000*	500	200
Write Bandwidth [MB/s]	1000-3000	500/1000*	500	200
Read IOPs	500-800K	Up to 100K	Up to 100K	100-200
Write IOPs	100-300K	Up to 100K	Up to 100K	100-200
Random latency (avg)	< 100us	< 100us	< 100us	5000us
$/GB	$2-5	$1-3	$0.40	$0.04
Endurance (drive re-writes)	10,000	10,000	1,000	Unlimited

Fuente: https://sdsblog.com/2014/10/21/to-hdd-or-to-ssd/

Algunos detalles sobre el desempeño de los SSD



La operación de lectura y la de escritura inicial son más rápidas que las siguientes de escritura (borrar + escribir)

La escritura de una página implica escribir todo el **bloque**

Bases de datos sobre SSDs

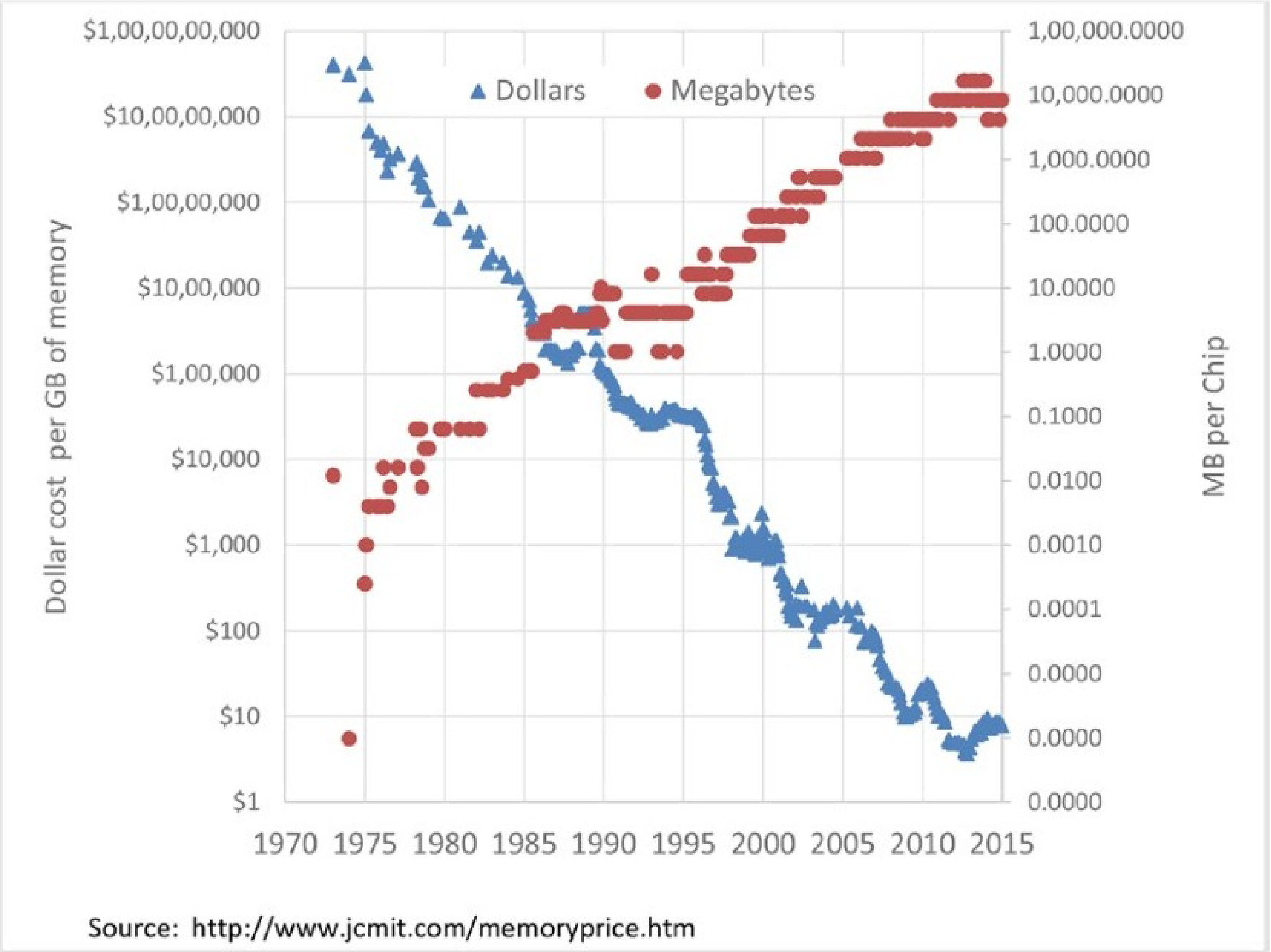
Es necesario cambiar la estrategia para reducir el **sobrecosto** de la escritura

Log-structured storage engines

La idea: acumular las operaciones de escritura y hacerlas juntas.

Ejemplos: Cassandra y Aerospike

¿y si usamos la memoria RAM?



Impactos en la arquitectura

La idea de *caché* no tiene sentido si los datos están todos en memoria.

Cache-less architecture

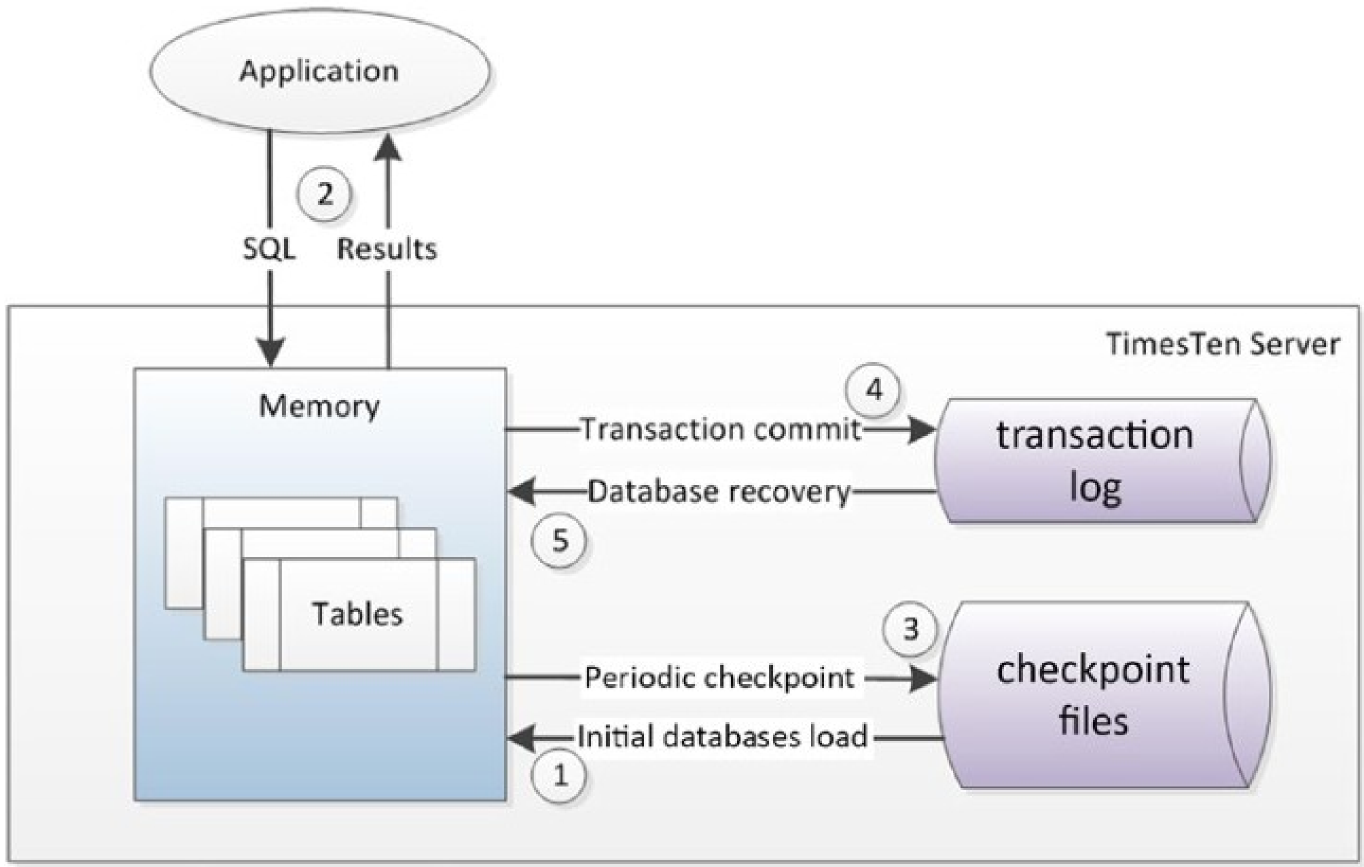
¿cómo garantizar la durabilidad de los datos?

Réplicas en otras máquinas

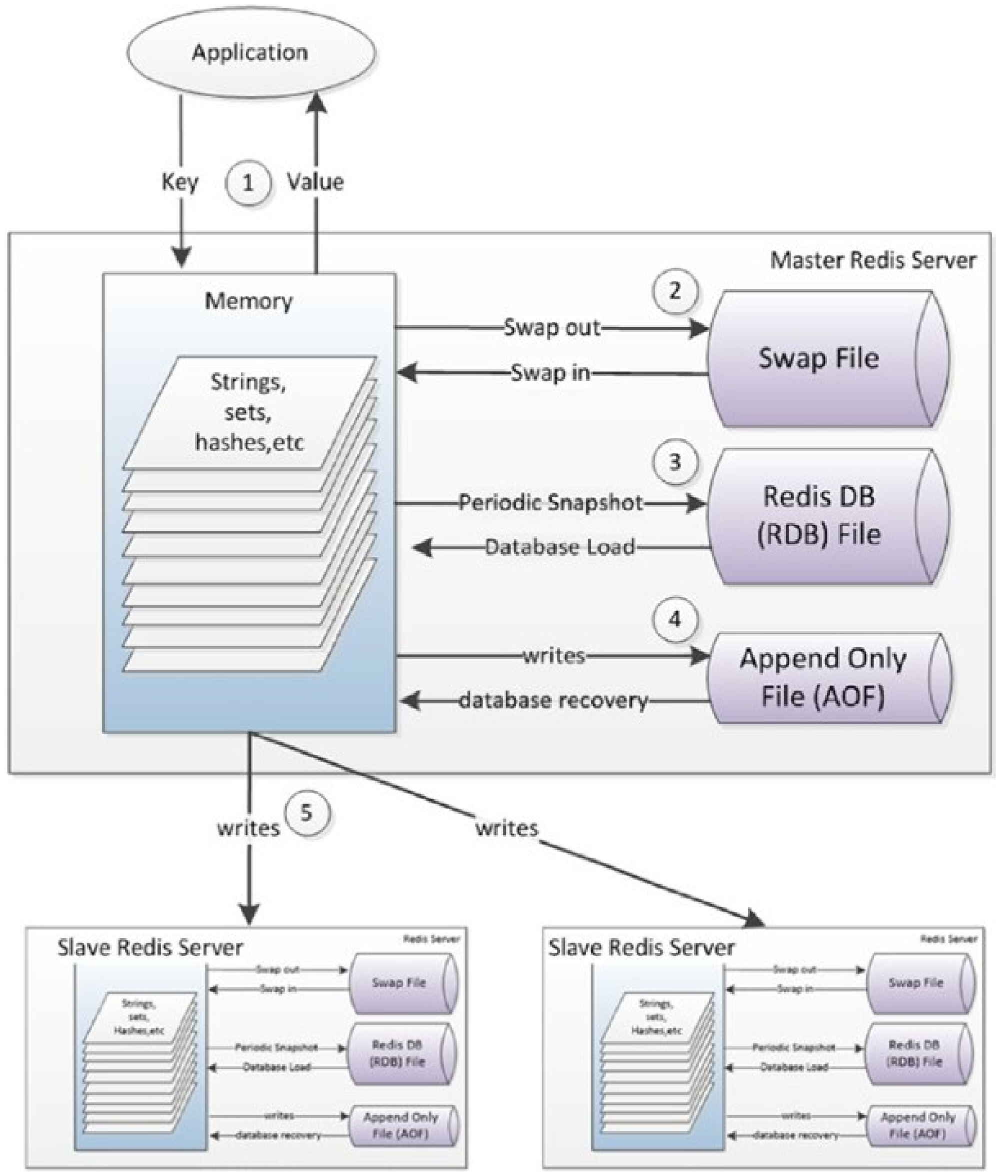
Guardar *snapshots* en disco de la base completa

Guardar las transacciones en disco en archivos *append-only* (*journals*)

RDBMS en memoria: TimesTen



Key-value stores en memoria: Redis



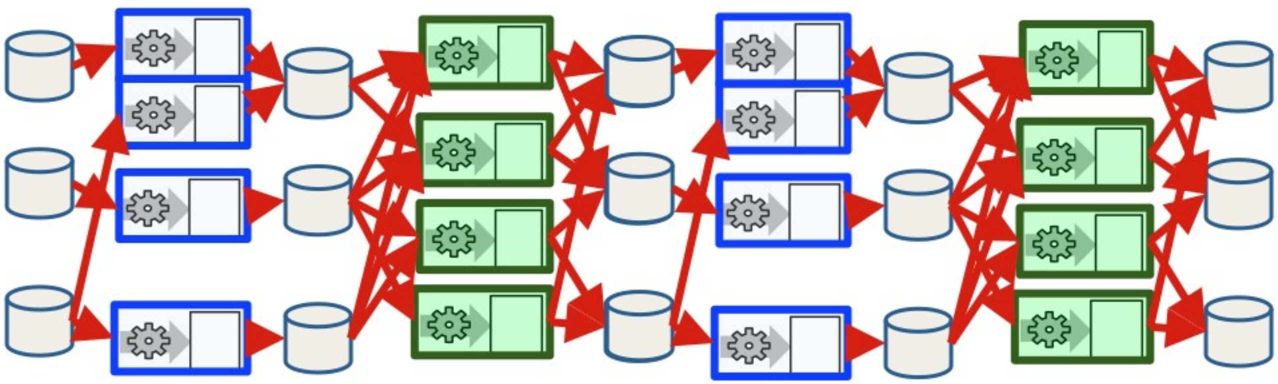

APACHE
Spark
25

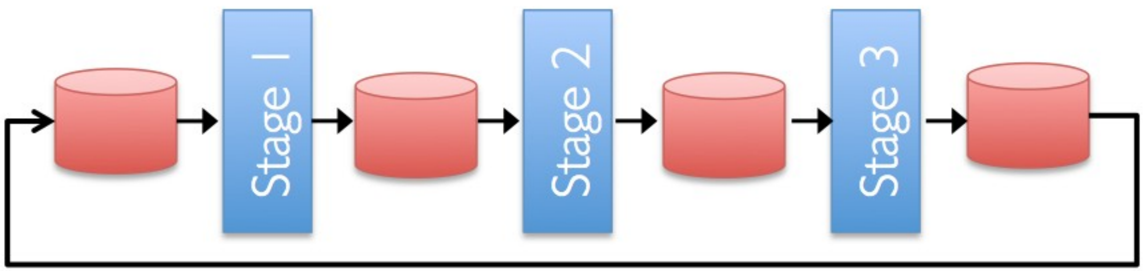
¿Qué es Apache Spark ?

Es un sistema de cómputo distribuído, **en memoria**, tolerante a fallas y escalable.

Puede pensarse como un simil o evolución de Hadoop, pero en memoria

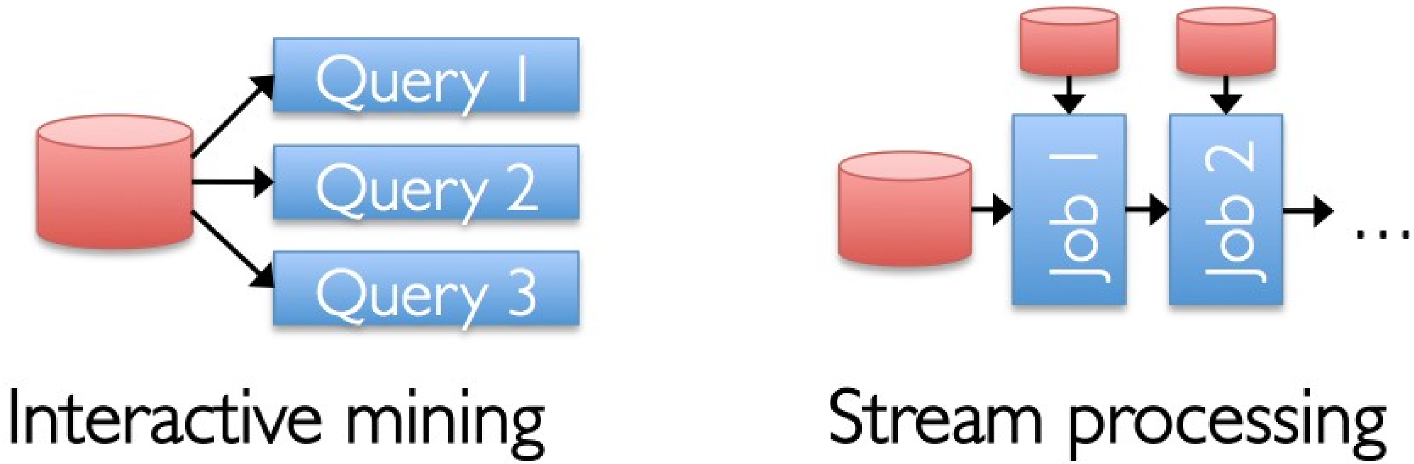
Motivación





El proceso iterativo sobre conjuntos de datos usando MapReduce (Hadoop) es **intensivo en acceso a disco**

Motivación (ii)



Realizar análisis interactivo sobre conjuntos de datos, o procesar datos tipo *stream* también son escenarios **intensivos en acceso a disco**

Apache Spark

Surge como un proyecto de UC Berkeley en 2009

Se transforma en proyecto de Apache en 2013

Spark: Cluster Computing with Working Sets.

Matei et al.. HotCloud 2010.

Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing.

Matei et al.. NSDI 2012.

Spark vs Hadoop MapReduce

Factors	Spark	Hadoop MapReduce
Speed	100x times than MapReduce	Faster than traditional system
Written In	Scala	Java
Data Processing	Batch / real-time / iterative / interactive /graph	Batch processing
Ease of Use	Compact & easier than Hadoop	Complex & lengthy
Caching	Caches the data in-memory & enhances the system performance	Doesn't support caching of data

Desafío *Daytona GraySort*

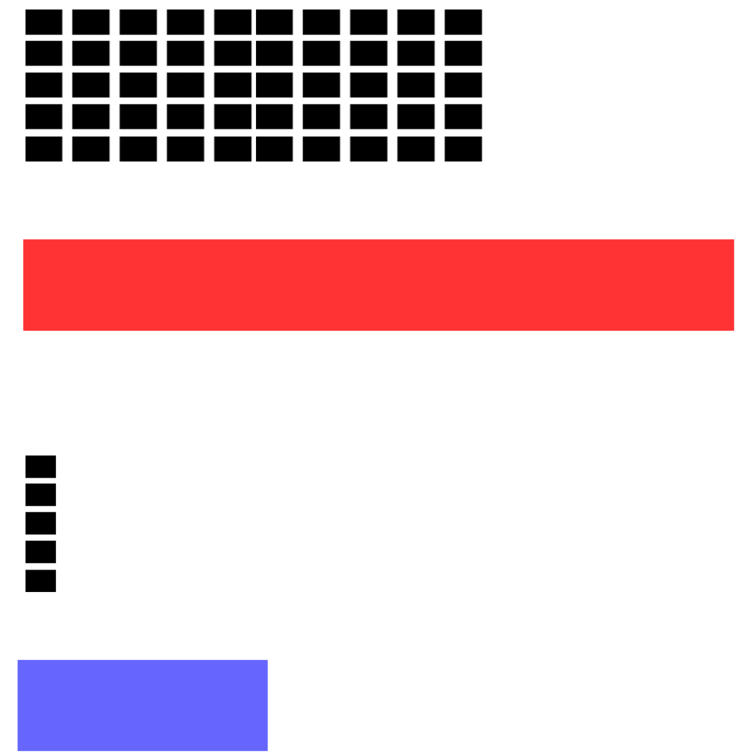
La tarea: ordenar 100 TB de datos!!

Hadoop (2013): 2100 nodos

72 minutos

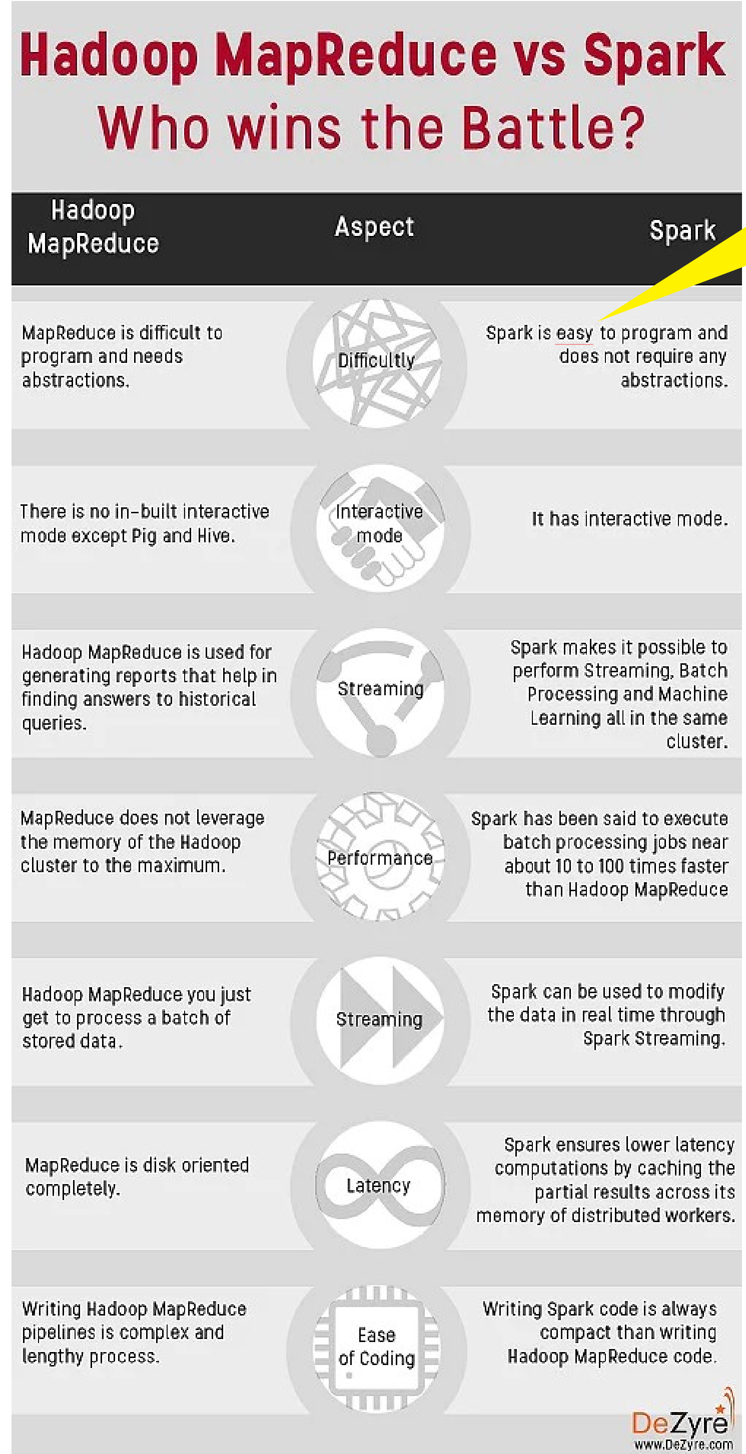
Spark (2014): 206 nodos

23 minutos



Más info sobre el experimento :https://databricks.com/blog/2014/11/05/spark-officially-sets-a-new-record-in-large-scale-sorting.html

¿quién gana?
Hadoop vs Spark



Hadoop MapReduce vs Spark
Who wins the Battle?

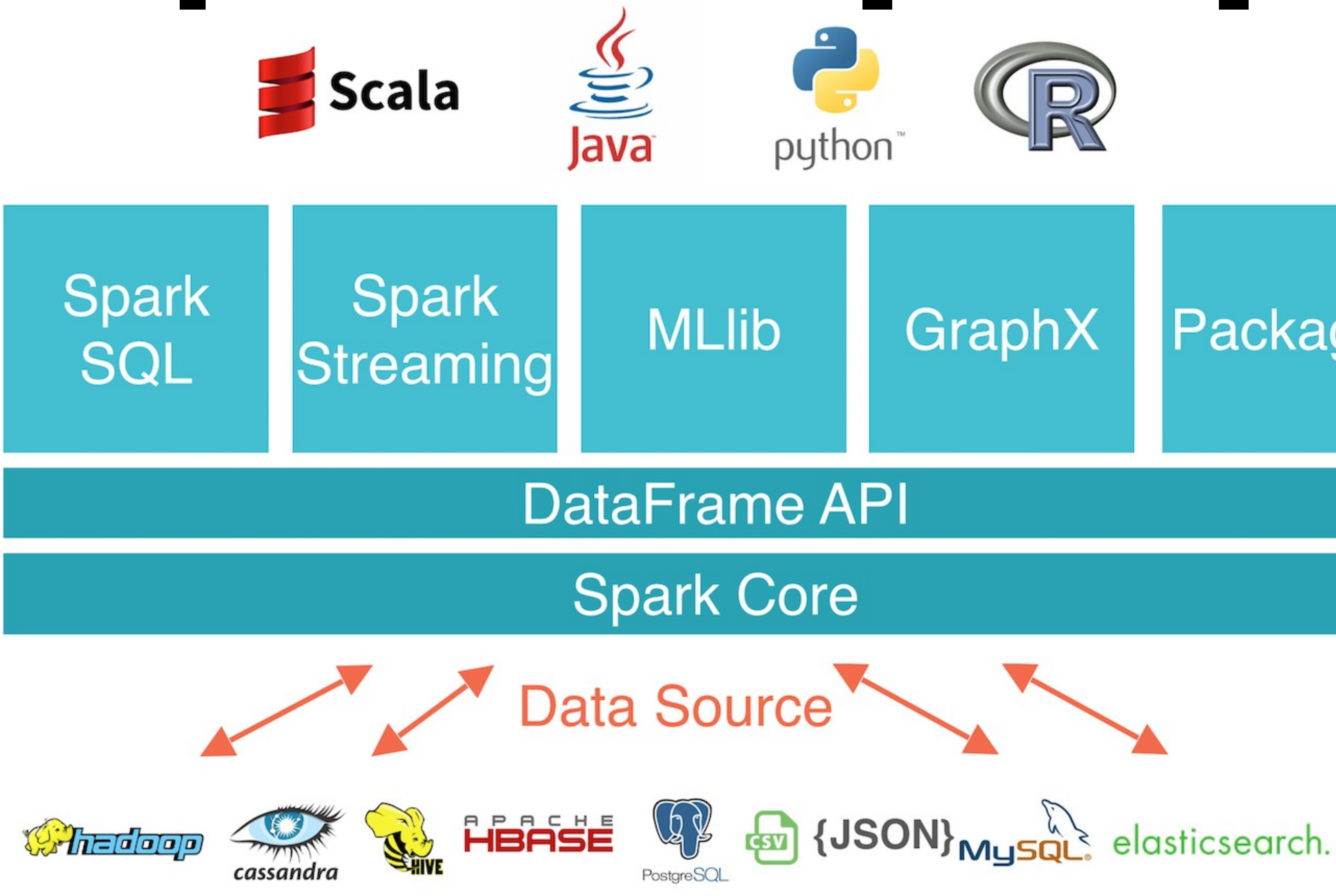
Hadoop MapReduce	Aspect	Spark
MapReduce is difficult to program and needs abstractions.	Difficultly	Spark is easy to program and does not require any abstractions.
There is no in-built interactive mode except Pig and Hive.	Interactive mode	It has interactive mode.
Hadoop MapReduce is used for generating reports that help in finding answers to historical queries.	Streaming	Spark makes it possible to perform Streaming, Batch Processing and Machine Learning all in the same cluster.
MapReduce does not leverage the memory of the Hadoop cluster to the maximum.	Performance	Spark has been said to execute batch processing jobs near about 10 to 100 times faster than Hadoop MapReduce
Hadoop MapReduce you just get to process a batch of stored data.	Streaming	Spark can be used to modify the data in real time through Spark Streaming.
MapReduce is disk oriented completely.	Latency	Spark ensures lower latency computations by caching the partial results across its memory of distributed workers.
Writing Hadoop MapReduce pipelines is complex and lengthy process.	Ease of Coding	Writing Spark code is always compact than writing Hadoop MapReduce code.

DeZyre
www.DeZyre.com

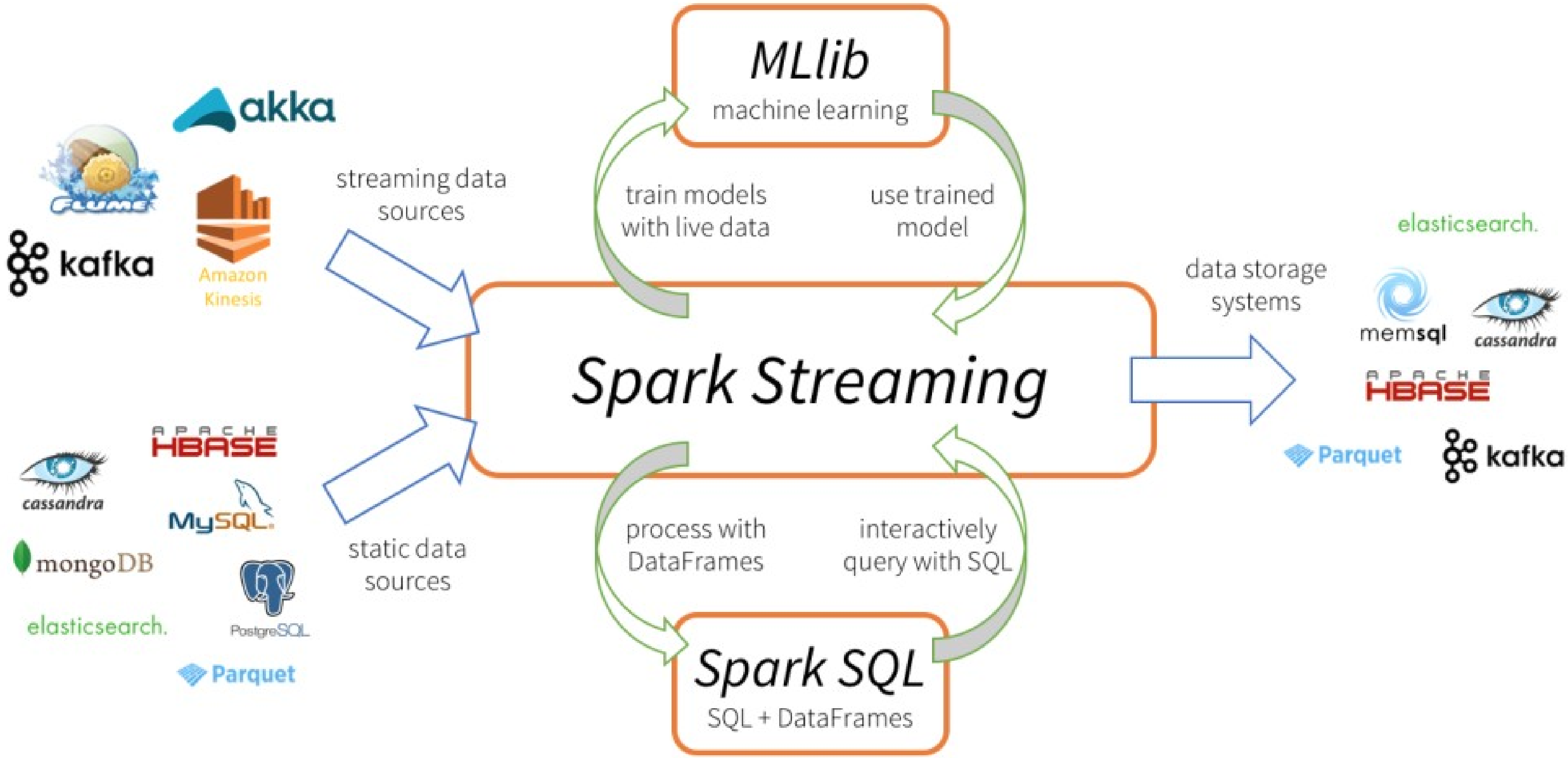
Más sencillo que Hadoop

https://www.projectpro.io/article/hadoop-mapreduce-vs-apache-spark-who-wins-the-battle/83

Arquitectura de Apache Spark



Arquitectura de Apache Spark



Journal of Machine Learning Research 17 (2016) 1-7 Submitted 5/15; Published 4/16

Spark SQL: Relational Data Processing in Spark

Michael Armbrust[†], Reynold S. Xin[†], Cheng Lian[†], Yin Huai[†], Davies Liu[†], Joseph K. Bradley[†], Xiangrui Meng[†], Tomer Kaftan[‡], Michael J. Franklin[†‡], Ali Ghodsi[†], Matei Zaharia[†*]

[†]Databricks Inc. [*]MIT CSAIL [‡]AMPLab, UC Berkeley

ABSTRACT

Spark SQL is a new module in Apache Spark that integrates relational processing with Spark's functional programming API. Built on our experience with Shark, Spark SQL lets Spark program-

While the popularity of relational systems shows that users often prefer writing declarative queries, the relational approach is insufficient for many big data applications. First, users want to perform ETL to and from various data sources that might be semi- or un-

MLlib: Machine Learning in Apache Spark

Xiangrui Meng[†] MENG@DATABRICKS.COM
Databricks, 160 Spear Street, 13th Floor, San Francisco, CA 94105

Joseph Bradley JOSEPH@DATABRICKS.COM
Databricks, 160 Spear Street, 13th Floor, San Francisco, CA 94105

Burak Yavuz BURAK@DATABRICKS.COM
Databricks, 160 Spear Street, 13th Floor, San Francisco, CA 94105

Modelo de Programación

Resilient Distributed Datasets (RDDs)

- Colecciones *read-only* de objetos
- Operaciones en paralelo sobre los RDDs
- Variables compartidas

Dataframes

Esta es la abstracción que se usa más habitualmente

- Similares a los RDDs pero para datos estructurados
- Infiere un esquema a partir de los datos
- Luego puedo usar sparkSQL

Algunos aspectos sobre los RDDs

Son colecciones de **objetos** que
se particionan en diferentes máquinas.

Por defecto son *lazy* y efímeras.

¿cómo se resuelve la **tolerancia a fallas**?

Se guarda suficiente información de *lineage* o
provenance como para poder recomputar
cualquier RDD

¿cómo se crean los RDDs?

1 . Desde **archivos**

2 . **Particionando** (*"parallelizing")* una colección Scala

3 . **Transformando** un RDD existente (via *flatMap* y funciones)

4 . Cambiando el modo de **persistencia** de un RDD existente: *cache* y *save*

¿cómo se manipulan los RDDs?

Transformaciones

$$
\begin{aligned}
\mathit{map}(f : \mathrm{T} \Rightarrow \mathrm{U}) &: \mathrm{RDD[T]} \Rightarrow \mathrm{RDD[U]} \\
\mathit{filter}(f : \mathrm{T} \Rightarrow \mathrm{Bool}) &: \mathrm{RDD[T]} \Rightarrow \mathrm{RDD[T]} \\
\mathit{flatMap}(f : \mathrm{T} \Rightarrow \mathrm{Seq[U]}) &: \mathrm{RDD[T]} \Rightarrow \mathrm{RDD[U]} \\
\mathit{sample}(\mathit{fraction} : \mathrm{Float}) &: \mathrm{RDD[T]} \Rightarrow \mathrm{RDD[T]} \ \text{(Deterministic sampling)} \\
\mathit{groupByKey}() &: \mathrm{RDD[(K, V)]} \Rightarrow \mathrm{RDD[(K, Seq[V])]} \\
\mathit{reduceByKey}(f : (\mathrm{V}, \mathrm{V}) \Rightarrow \mathrm{V}) &: \mathrm{RDD[(K, V)]} \Rightarrow \mathrm{RDD[(K, V)]} \\
\mathit{union}() &: (\mathrm{RDD[T]}, \mathrm{RDD[T]}) \Rightarrow \mathrm{RDD[T]} \\
\mathit{join}() &: (\mathrm{RDD[(K, V)]}, \mathrm{RDD[(K, W)]}) \Rightarrow \mathrm{RDD[(K, (V, W))]} \\
\mathit{cogroup}() &: (\mathrm{RDD[(K, V)]}, \mathrm{RDD[(K, W)]}) \Rightarrow \mathrm{RDD[(K, (Seq[V], Seq[W]))]} \\
\mathit{crossProduct}() &: (\mathrm{RDD[T]}, \mathrm{RDD[U]}) \Rightarrow \mathrm{RDD[(T, U)]} \\
\mathit{mapValues}(f : \mathrm{V} \Rightarrow \mathrm{W}) &: \mathrm{RDD[(K, V)]} \Rightarrow \mathrm{RDD[(K, W)]} \ \text{(Preserves partitioning)} \\
\mathit{sort}(c : \mathrm{Comparator[K]}) &: \mathrm{RDD[(K, V)]} \Rightarrow \mathrm{RDD[(K, V)]} \\
\mathit{partitionBy}(p : \mathrm{Partitioner[K]}) &: \mathrm{RDD[(K, V)]} \Rightarrow \mathrm{RDD[(K, V)]}
\end{aligned}
$$

ATENCIÓN!
map es un mapping 1-1

flatMap es similar al map de MapReduce

Acciones

$$
\begin{aligned}
\mathit{count}() &: \mathrm{RDD[T]} \Rightarrow \mathrm{Long} \\
\mathit{collect}() &: \mathrm{RDD[T]} \Rightarrow \mathrm{Seq[T]} \\
\mathit{reduce}(f : (\mathrm{T}, \mathrm{T}) \Rightarrow \mathrm{T}) &: \mathrm{RDD[T]} \Rightarrow \mathrm{T} \\
\mathit{lookup}(k : \mathrm{K}) &: \mathrm{RDD[(K, V)]} \Rightarrow \mathrm{Seq[V]} \ \text{(On hash/range partitioned RDDs)} \\
\mathit{save}(\mathit{path} : \mathrm{String}) &: \text{Outputs RDD to a storage system, } \textit{e.g.}\text{, HDFS}
\end{aligned}
$$

Conteo de palabras: Hadoop vs Spark

```
 public static class WordCountMapClass 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 itr = new StringTokenizer(line);
  while (itr.hasMoreTokens()) {
    word.set(itr.nextToken());
    output.collect(word, one);
    }
  }

public static class WorkdCountReduce 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));
    }
  }
```

```
val file = spark.textFile("hdfs://...")
val counts = file.flatMap(line => line.split(" "))
                 .map(word => (word, 1))
                 .reduceByKey(_ + _)
counts.saveAsTextFile("hdfs://...")
```

Fuente: Introduction to Spark with Scala, Himanshu Gupta

Conteo de palabras en Spark (Scala)

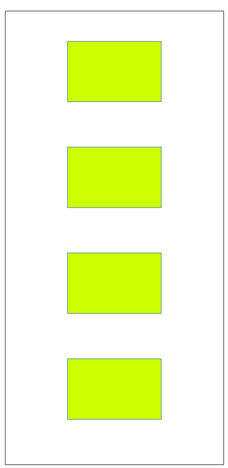
```scala
val master = "local"
val conf = new SparkConf().setMaster(master)
val sc = new SparkContext(conf)
```

El objeto *Spark Context*

Conteo de palabras en Spark (Scala)

```scala
val master = "local"
val conf = new SparkConf().setMaster(master)
val sc = new SparkContext(conf)
val lines = sc.textFile("data.txt")
```

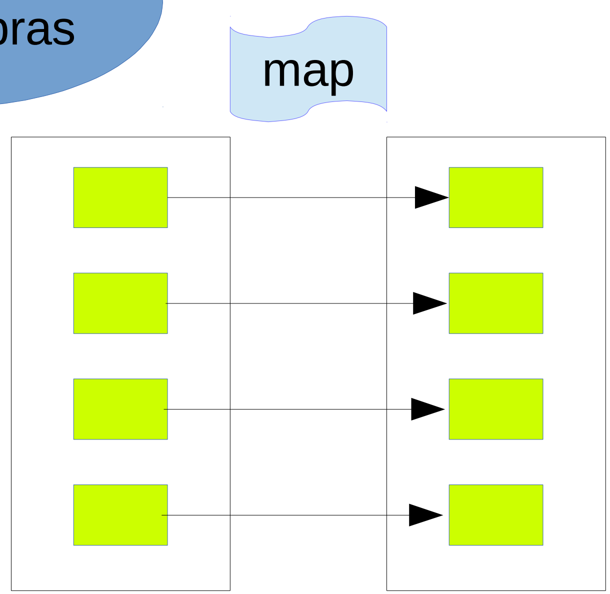
Creación de
un RDD desde
archivo



Conteo de palabras en Spark (Scala)

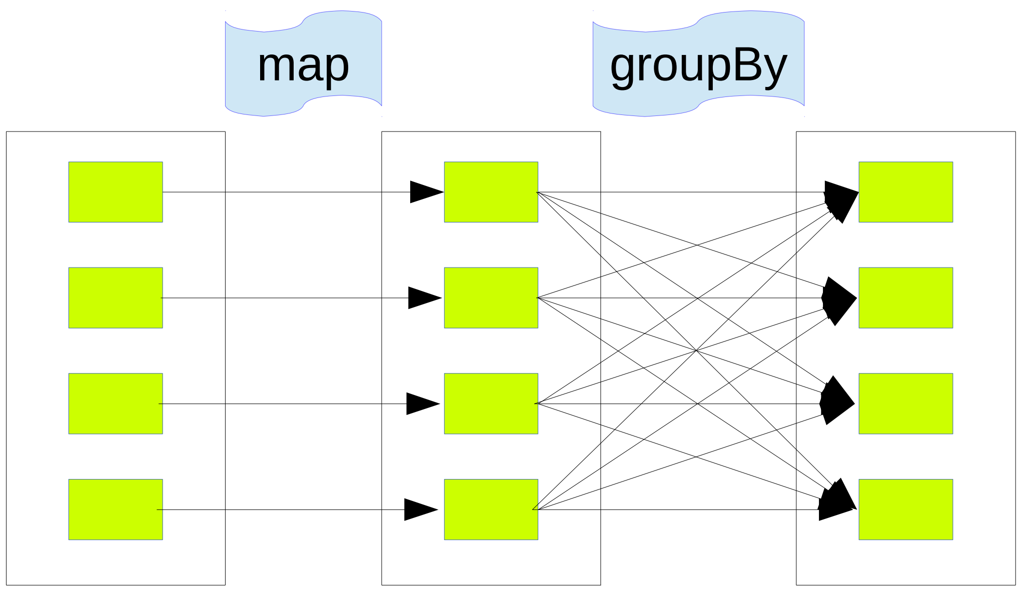
```scala
val master = "local"
val conf = new SparkConf().setMaster(master)
val sc = new SparkContext(conf)
val lines = sc.textFile("demo.txt")
val words = lines.flatMap(_.split(" ")).map((_,1))
```

Parte las líneas
en palabras



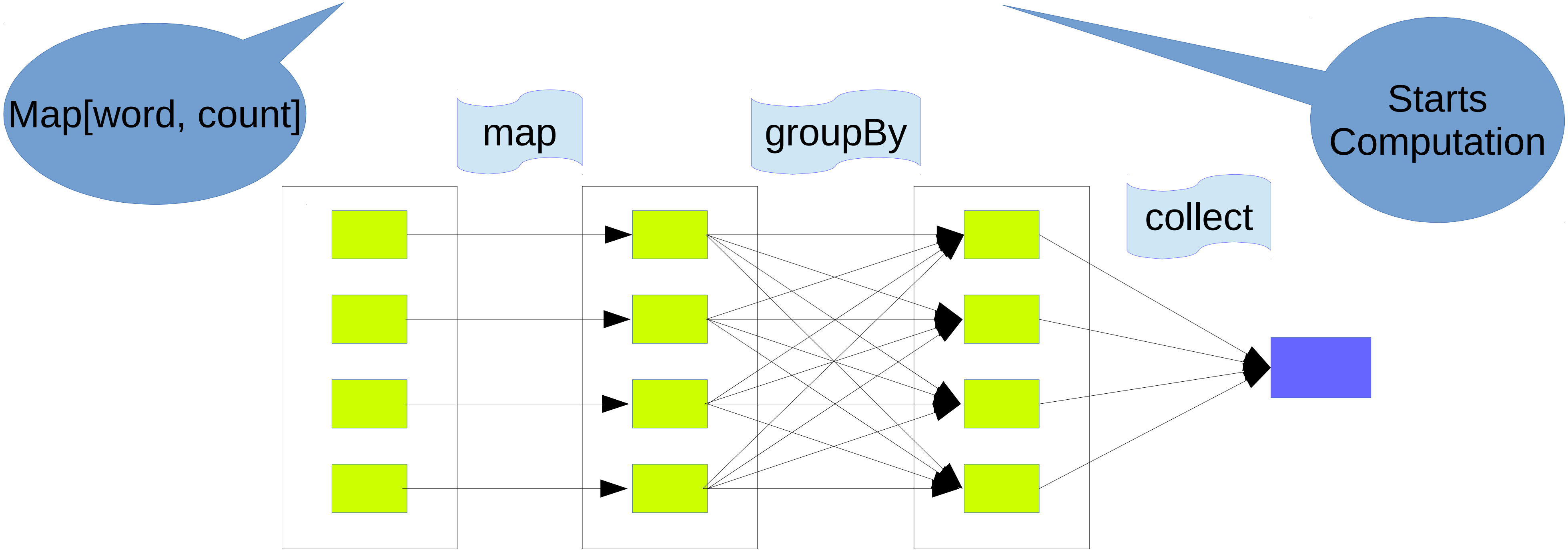
Conteo de palabras en Spark (Scala)

```
val master = "local"
val conf = new SparkConf().setMaster(master)
val sc = new SparkContext(conf)
val lines = sc.textFile("demo.txt")
val words = lines.flatMap(_.split(" ")).map((_,1))
val wordCountRDD = words.reduceByKey(_ + _)
```



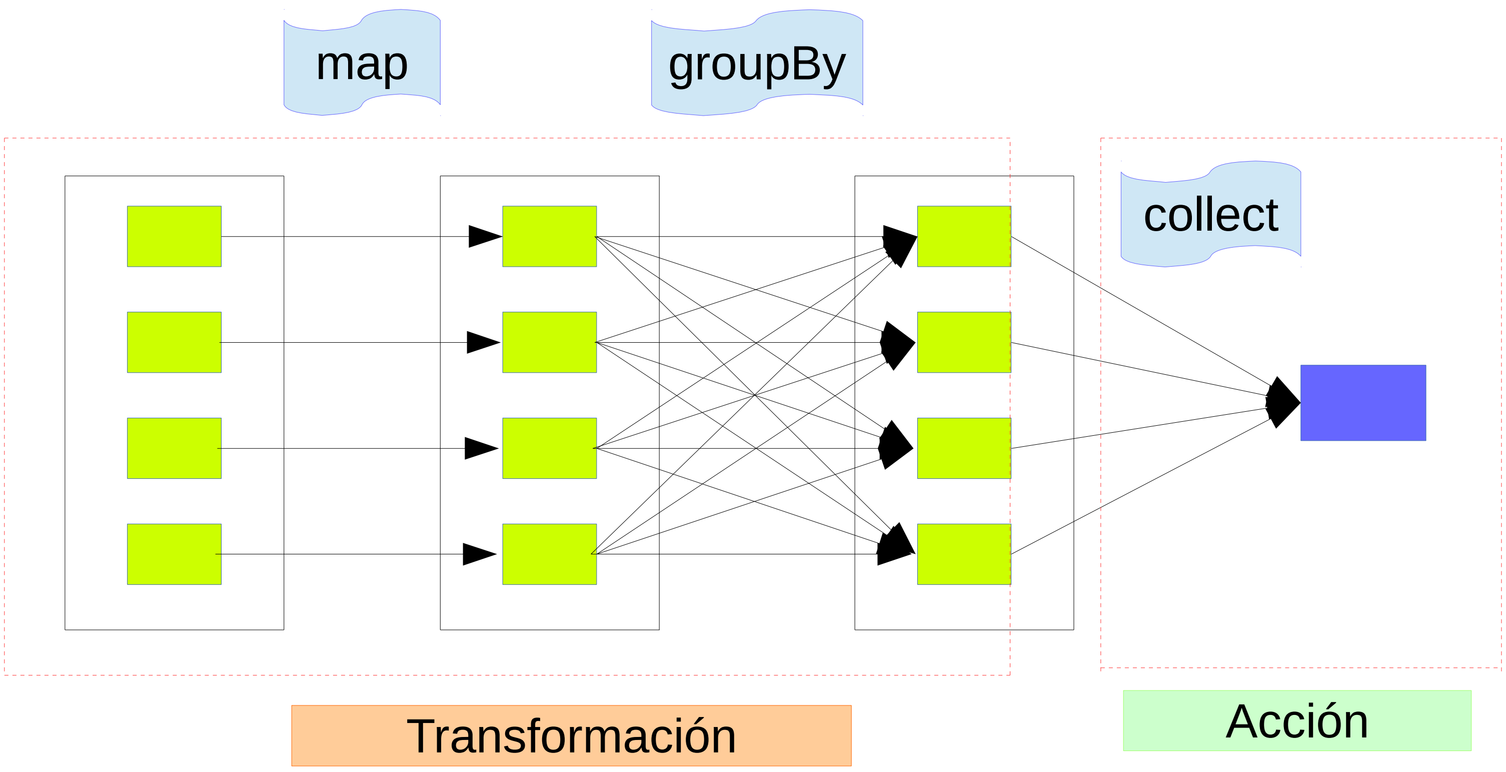
Conteo de palabras en Spark (Scala)

```
val master = "local"
val conf = new SparkConf().setMaster(master)
val sc = new SparkContext(conf)
val lines = sc.textFile("demo.txt")
val words = lines.flatMap(_.split(" ")).map((_,1))
val wordCountRDD = words.reduceByKey(_ + _)
val wordCount = wordCountRDD.collect
```



Ejemplo: conteo de palabras en pySpark

```
input_file = sc.textFile("demo.txt")
map = input_file.flatMap(lambda line: line.split("")).map(lambda word: (word, 1))
counts = map.reduceByKey(lambda a, b: a + b)
counts.saveAsTextFile("/path/to/output/")
```



Referencias y material adicional

- Artículos sobre Spark y proyectos asociados
  https://spark.apache.org/research.html
- Big Data Analytics with Spark *A Practitioner's Guide to Using Spark for Large-Scale Data Processing, Machine Learning, and Graph Analytics, and High-Velocity Data Stream Processing*, Guller Apress 2015.
  http://link.springer.com.proxy.timbo.org.uy:443/book/10.1007/978-1-4842-0964-6
- *Introduction to Big Data with Apache Spark*
  Curso UC Berkeley y Databricks
  https://github.com/dipanjanS/BerkeleyX-CS100.1x-Big-Data-with-Apache-Spark