# Introduction to


Javier Santos

May,11 - 2015

# About me

**Javier Santos** ***@jpaniego***

*«Hay dos formas de programar sin errores; solo la tercera funciona»*
Alan J Perlis

# Disclaimer I

- I’m no Spark evangelizer

# Disclaimer II

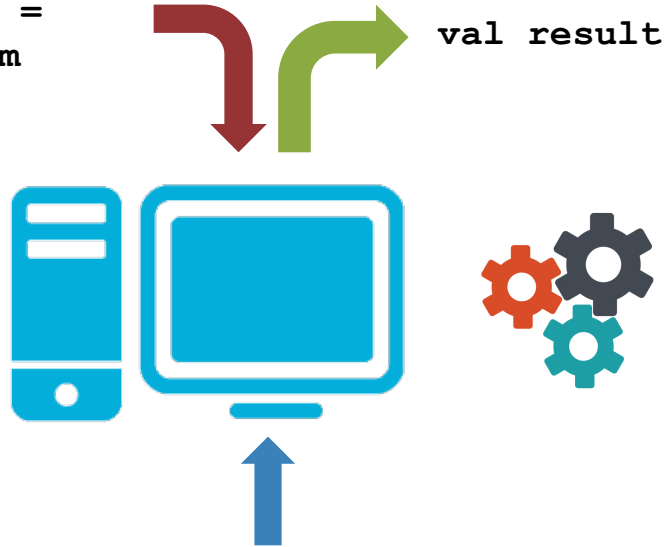
- I’m no expert

# Disclaimer III

- I’m not paying beers

# What's Spark?

```
val action: Seq[Int] => Int =
  (seq: Seq[Int]) => seq.sum
```

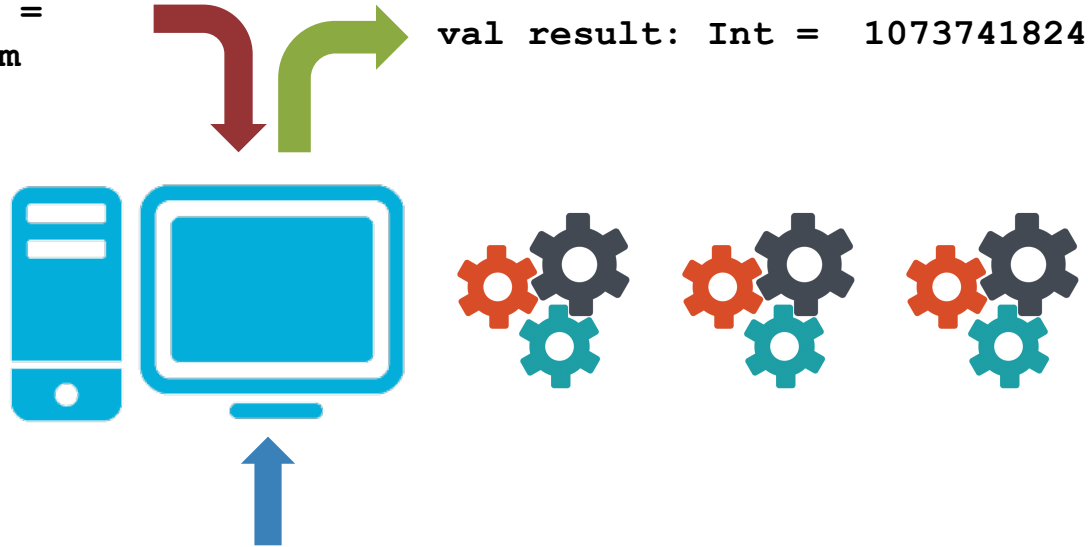


```
val result: Int = 15
```

```
val mySeq: Seq[Int] = 1 to 5
```

# What's Spark?

```
val action: Seq[Int] => Int =
  (seq: Seq[Int]) => seq.sum
```



```
val result: Int =  1073741824
```

```
val mySeq: Seq[Int] = 1 to Int.MaxValue
```

# What's Spark?

Computing problems:

- Time: Waiting for action results on tons of records
- Space: How do we allocate TB of data?

1/20/04 at 4:12pm **THREAD STARTER** post #1 of 26

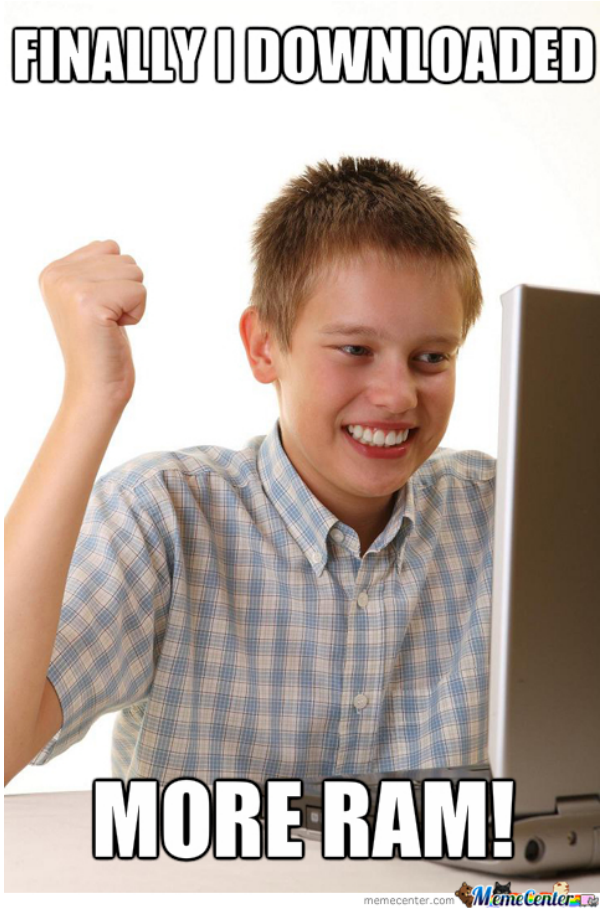
SpcMs

Joined: Jun 2003
Posts: 379
offline

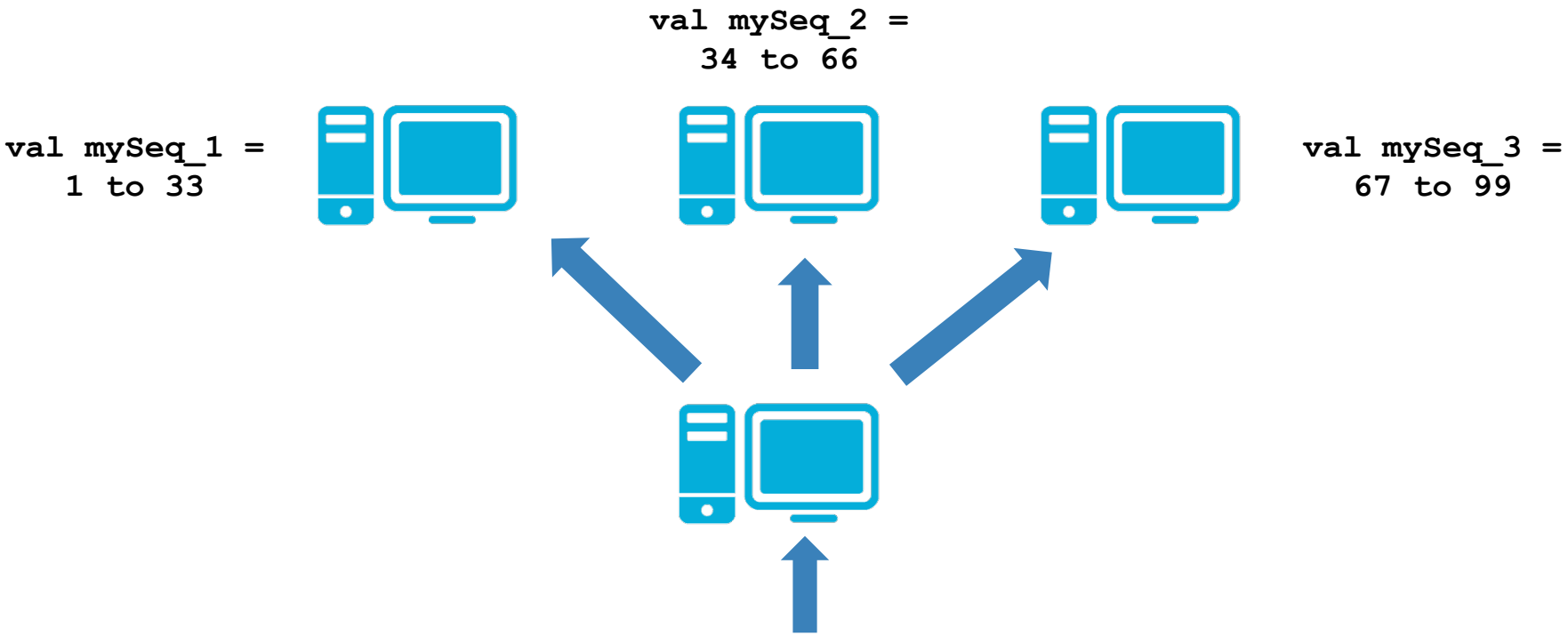
Any good sites that I can download some RAM from? My computer tells me i'm low on memory. Thanks in advance

It's Better To Be Hated For What You Are Than To Be Loved For What You Are Not

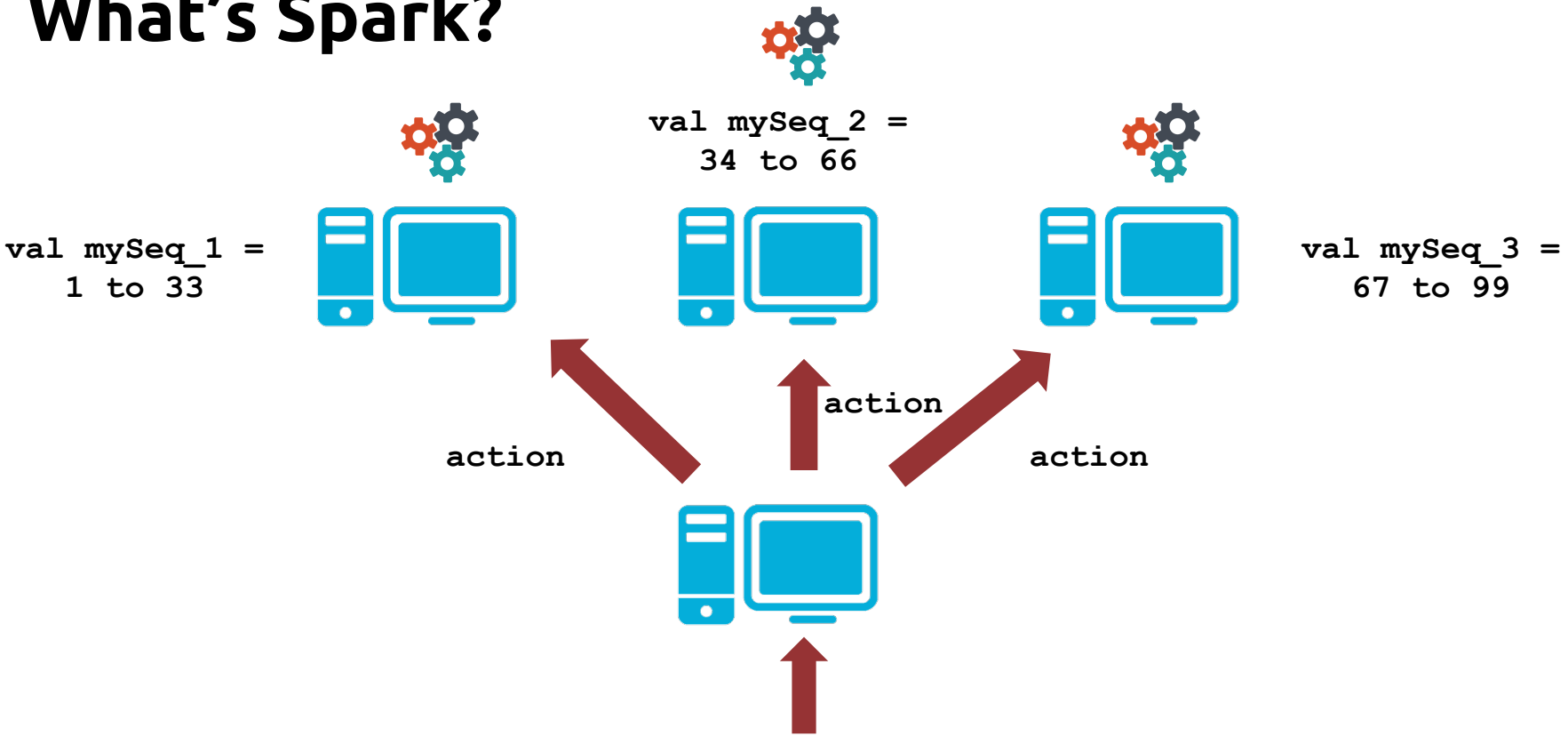
[downloadmoreram.com](downloadmoreram.com)


FINALLY I DOWNLOADED
MORE RAM!
memecenter.com
MemeCenter

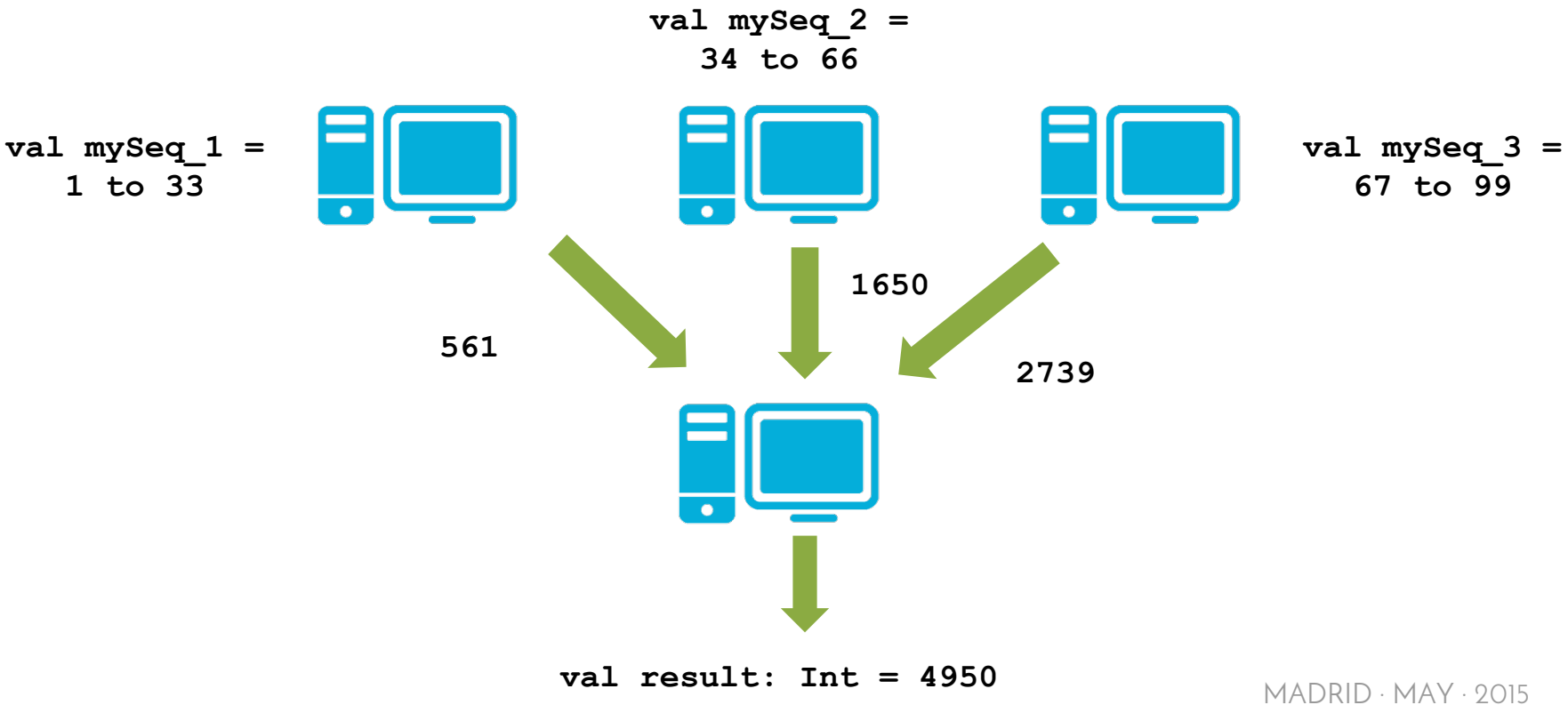
# What's Spark?


val mySeq_2 =
34 to 66
val mySeq_1 =
1 to 33
val mySeq_3 =
67 to 99
val mySeq: Seq[Int] = 1 to 99

# What's Spark?


val mySeq_1 =
1 to 33
val mySeq_2 =
34 to 66
val mySeq_3 =
67 to 99
action
action
action
val action: Seq[Int] => Int = (seq: Seq[Int]) => seq.sum

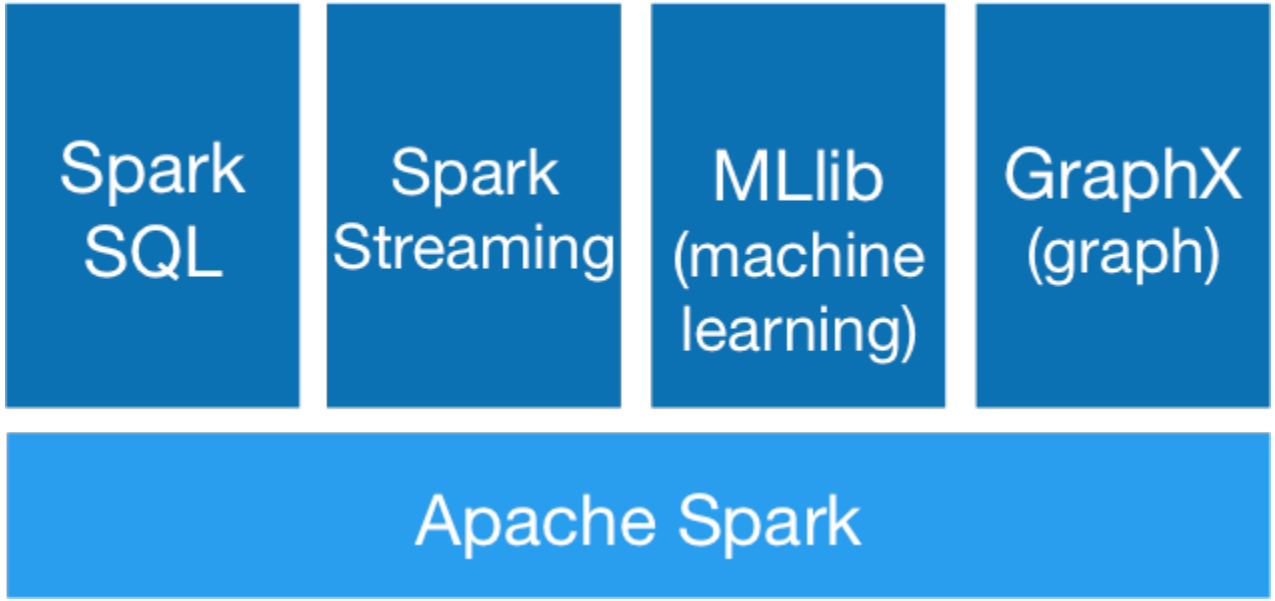
# What's Spark?


val mySeq_2 =
34 to 66
val mySeq_1 =
1 to 33
val mySeq_3 =
67 to 99
1650
561
2739
val result: Int = 4950

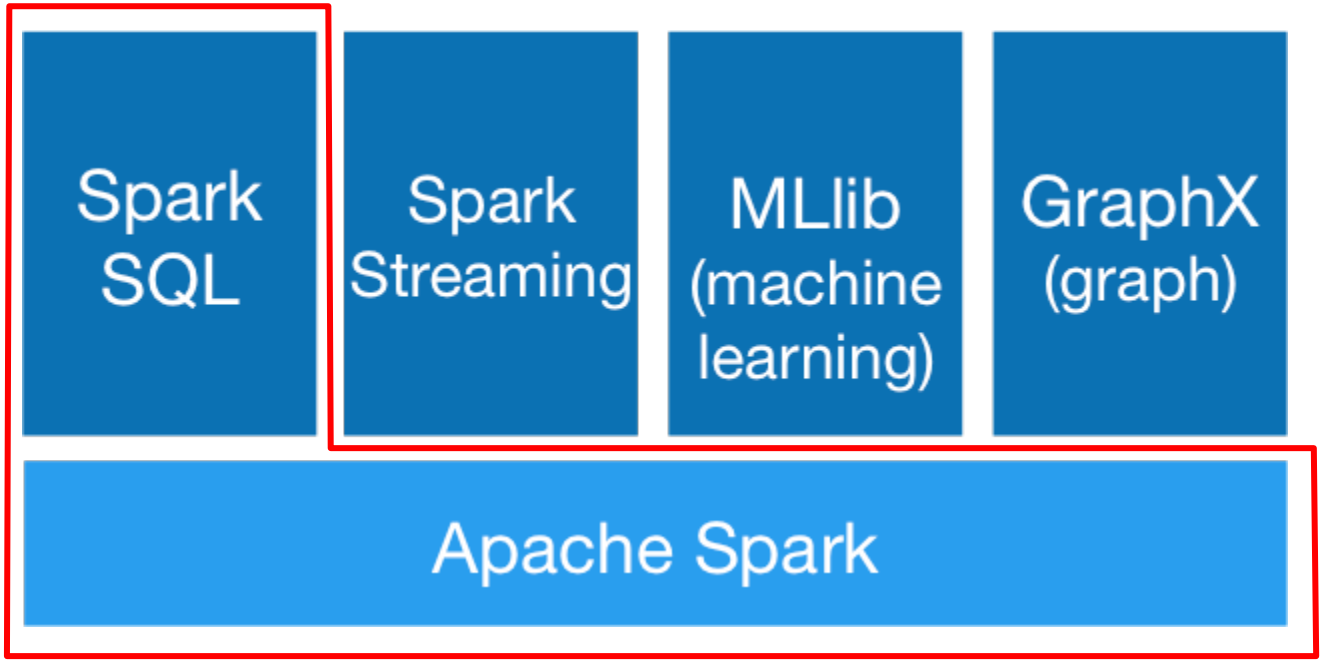
# What's Spark?

- General engine for large-scale data processing.
- "In-memory" data allocation.
- Target: **data** and **computing** parallelism.
- Scala, Python and Java APIs.
- First developed by AMPLab in UC Berkeley
- Support by Databricks
- Latest stable version 1.3.1

# Modules

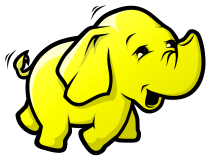

Spark SQL
Spark Streaming
MLlib (machine learning)
GraphX (graph)
Apache Spark

# Modules


Spark SQL
Spark Streaming
MLlib (machine learning)
GraphX (graph)
Apache Spark

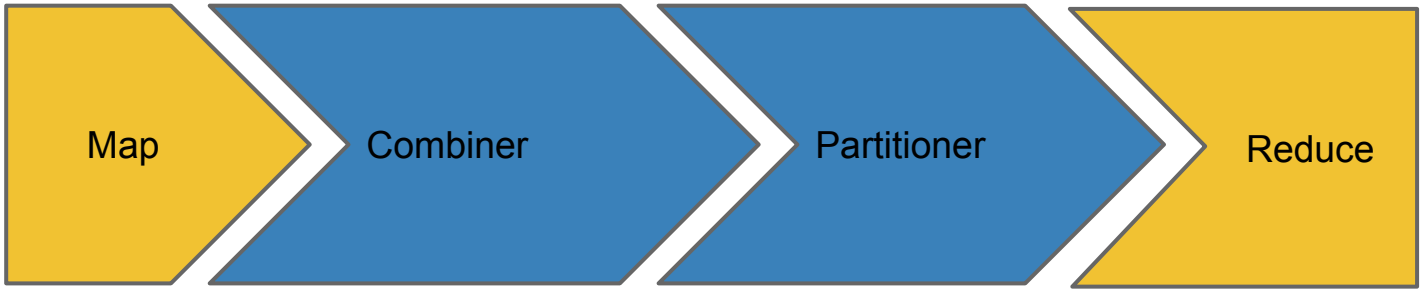
# MapReduce model

- Linear calculus method used for supporting parallel-computing and big data sets distribution among computing groups
- 80s. Google. PageRank. Big arrays computation.
- Based on two main functional programming methods: Map and reduce.
- Hadoop
  - One of the first open source implementations
  - Most of contributions made by Yahoo



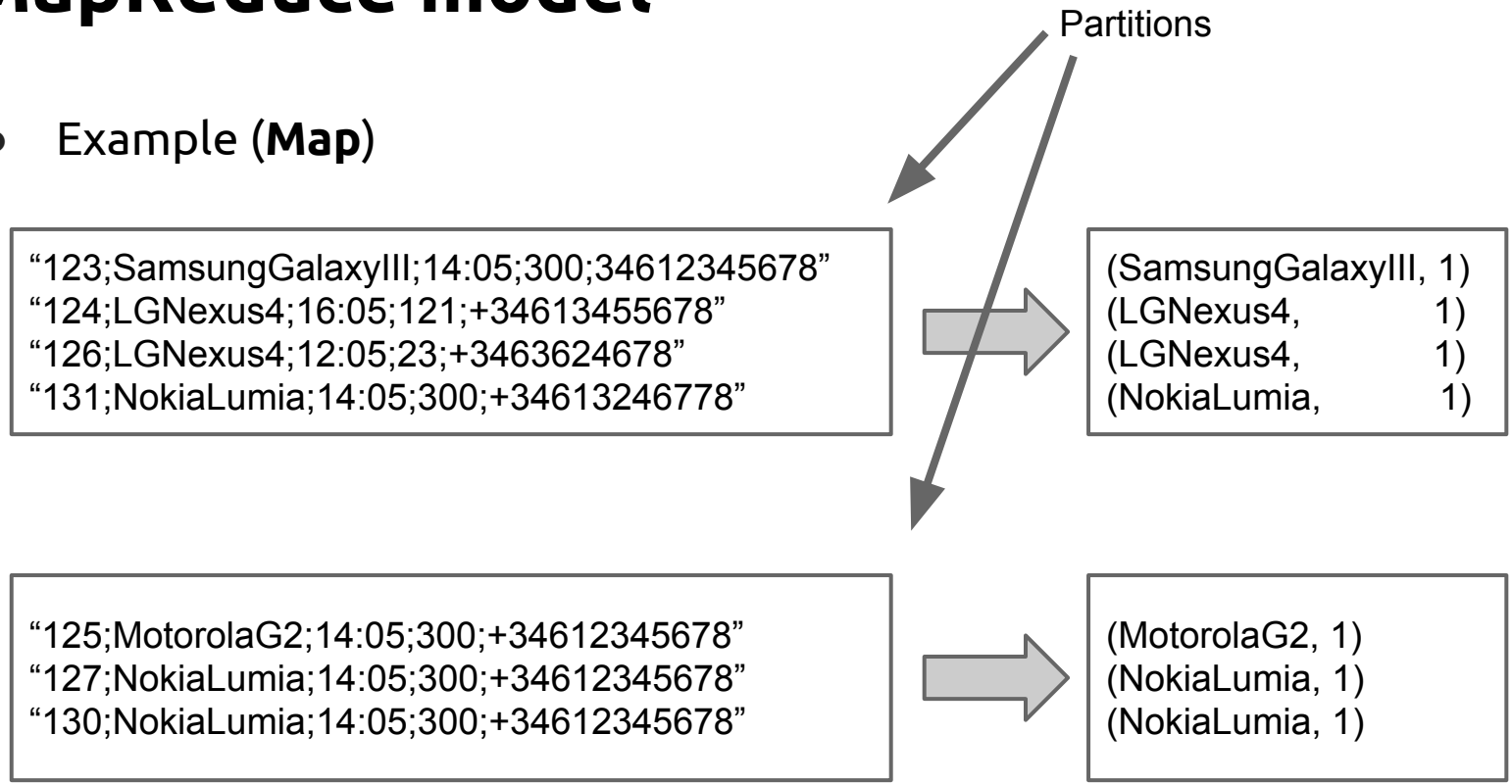
# MapReduce model

- Phases



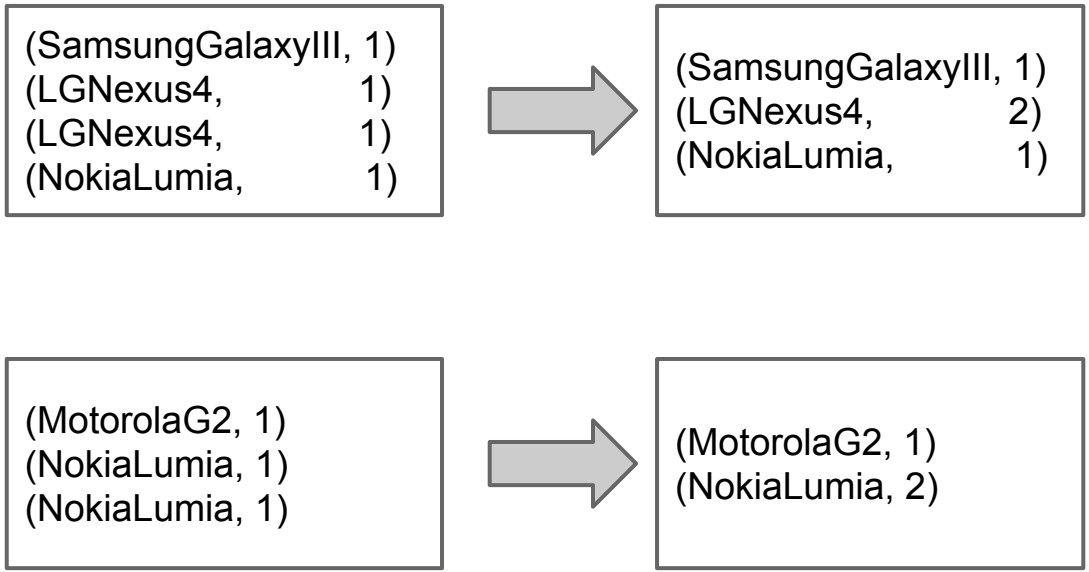
# MapReduce model

- Example (**Map**)



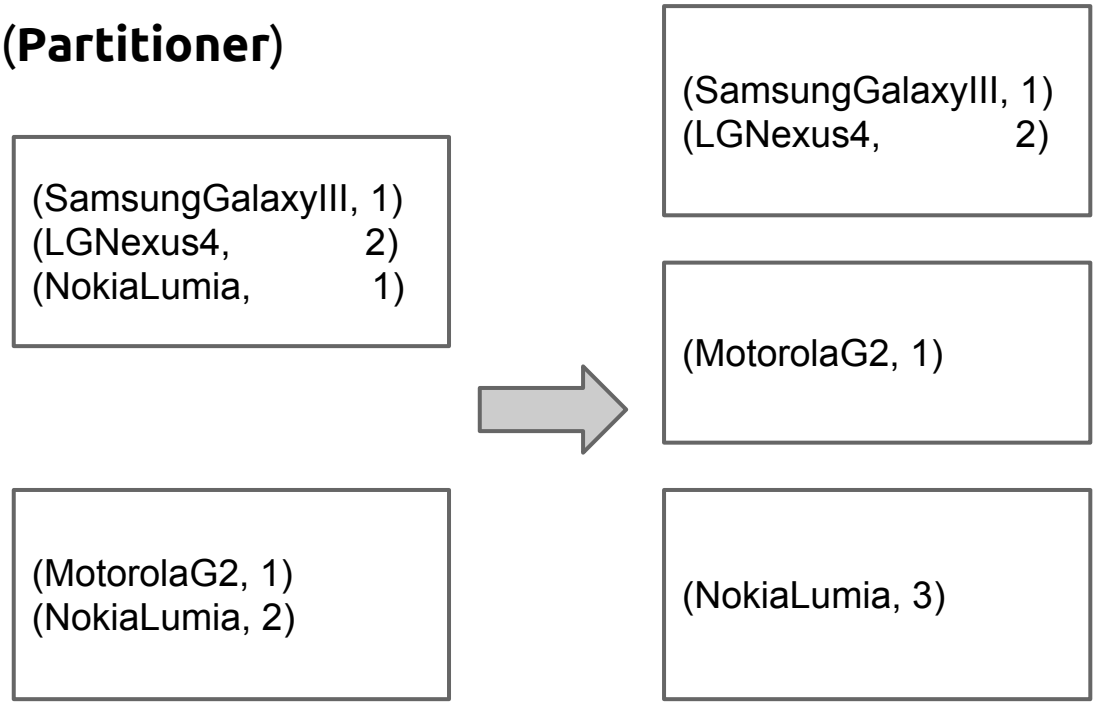
# MapReduce model

- Example (**Combiner - local aggregator**)



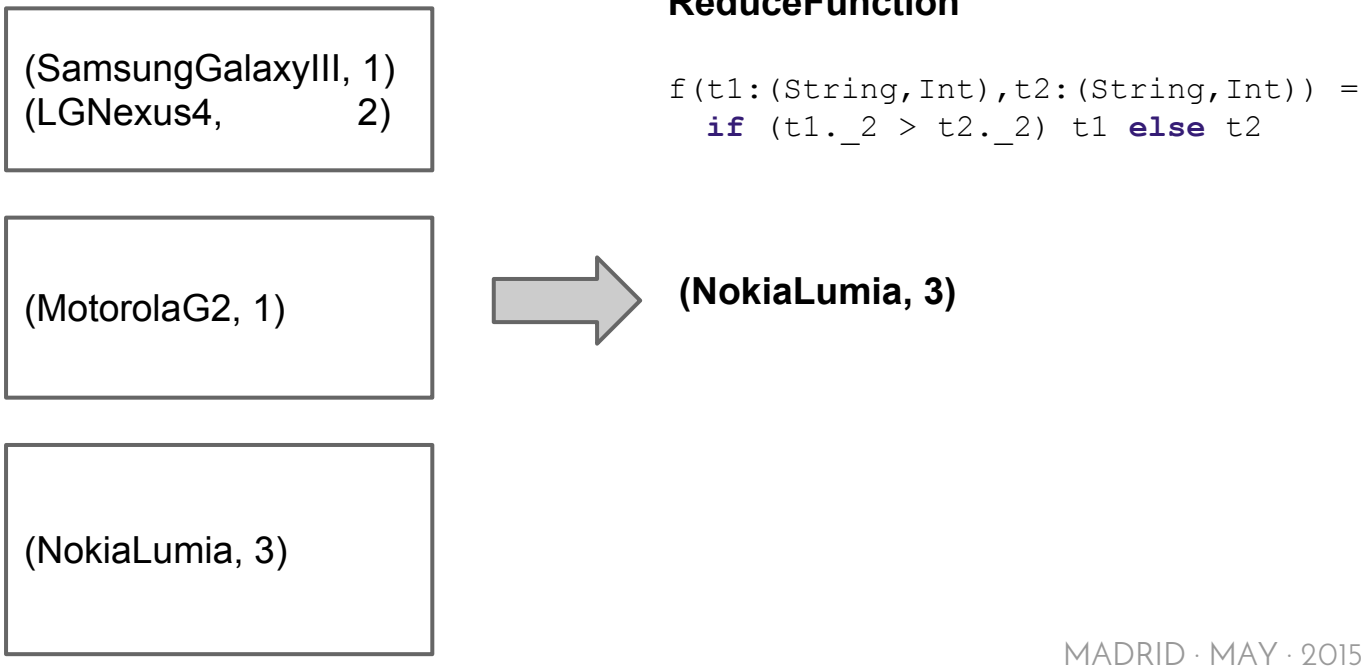
# MapReduce model

- Example (**Partitioner**)

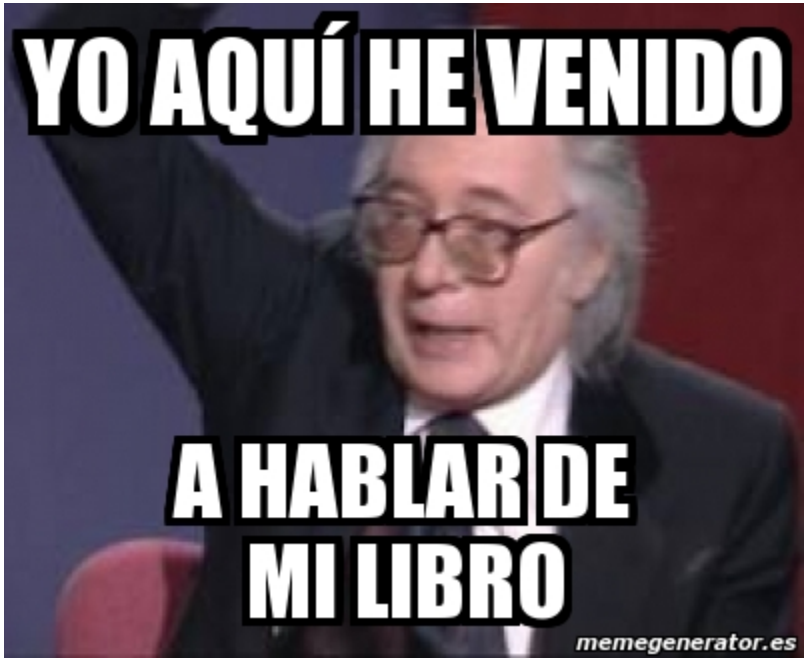


# MapReduce model

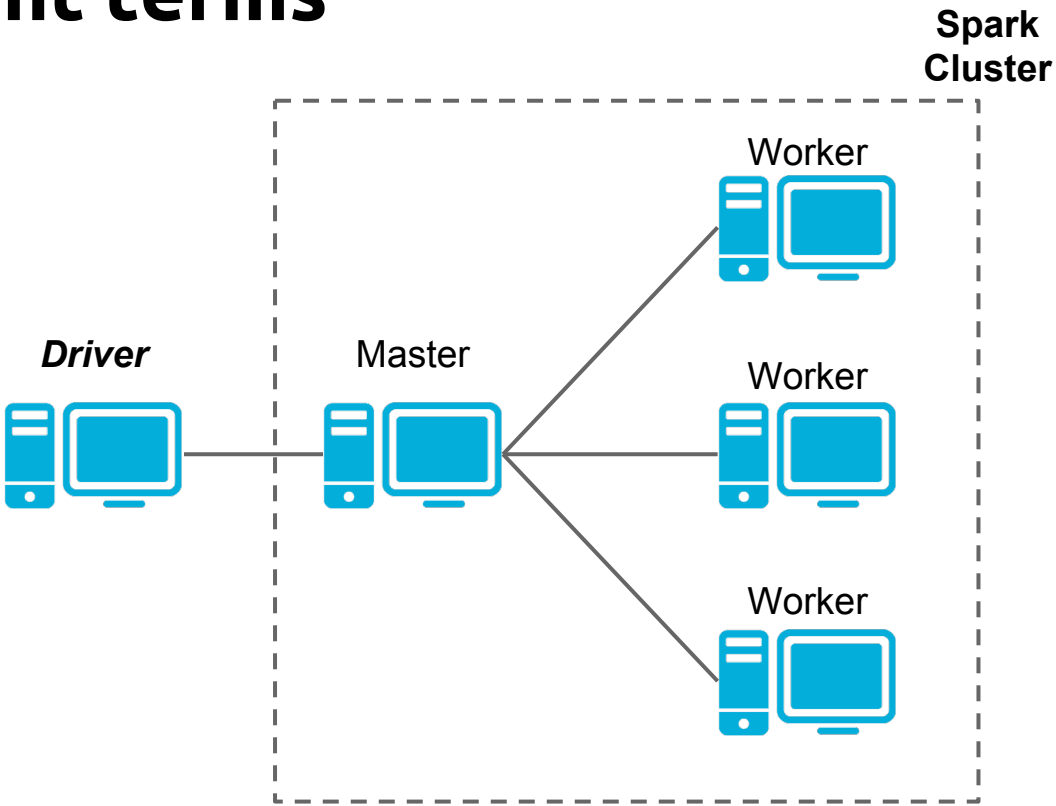
- Example (**Reduce**)



# So...what about Spark?



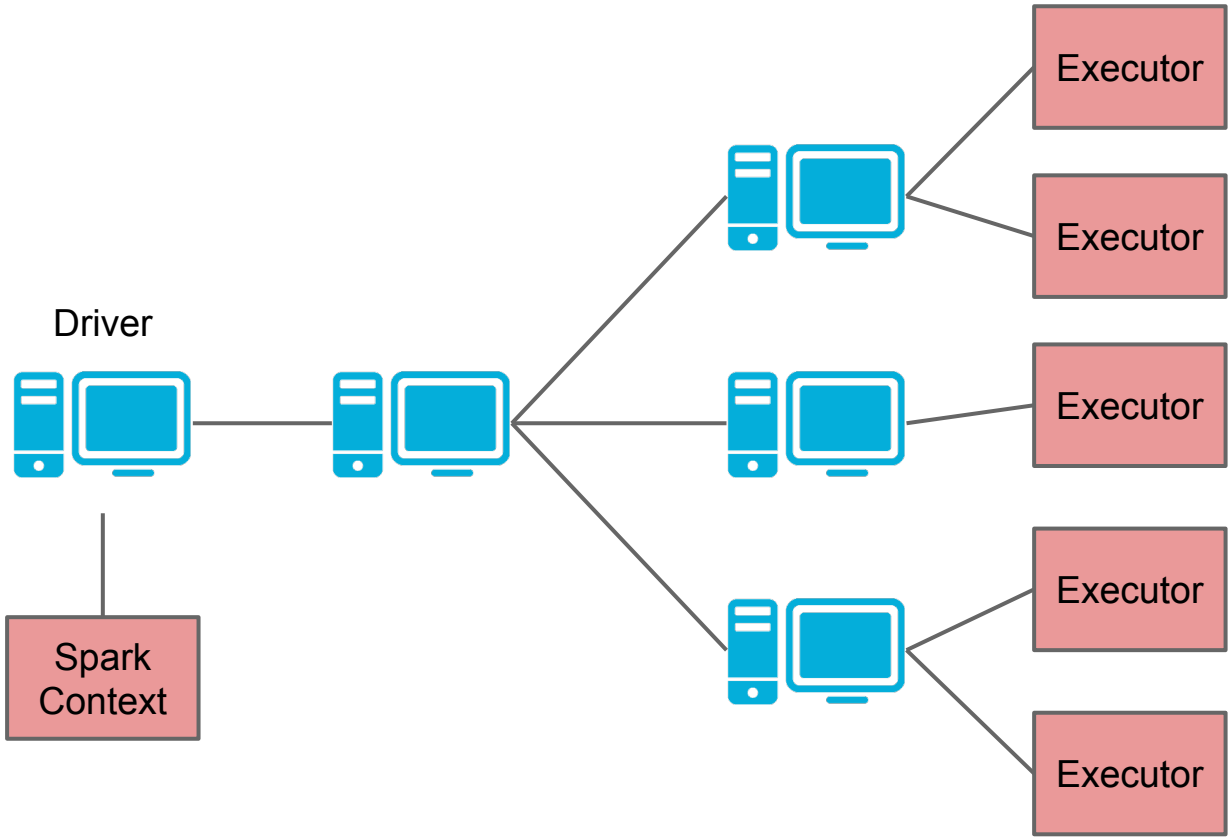
# Deployment terms



# Deployment types

- Local
  - `master="local[N]"` (N=Amount of cores)
  - Spark master is launched in the same process.
  - Workers are launched in the same process.
- Standalone
  - `master="spark://master-url"`
  - Spark master is launched in a cluster machine.
  - Submit JAR to worker nodes.
  - Resource managers: YARN, Mesos

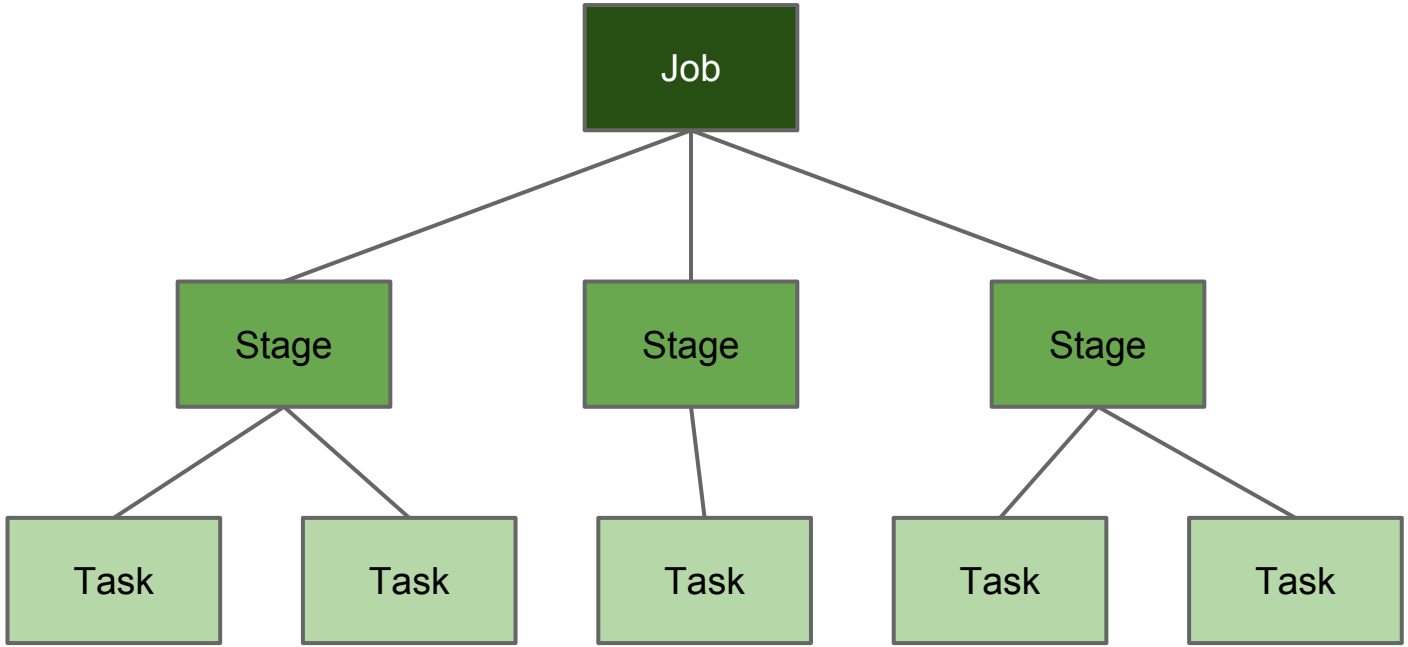
# Execution terms



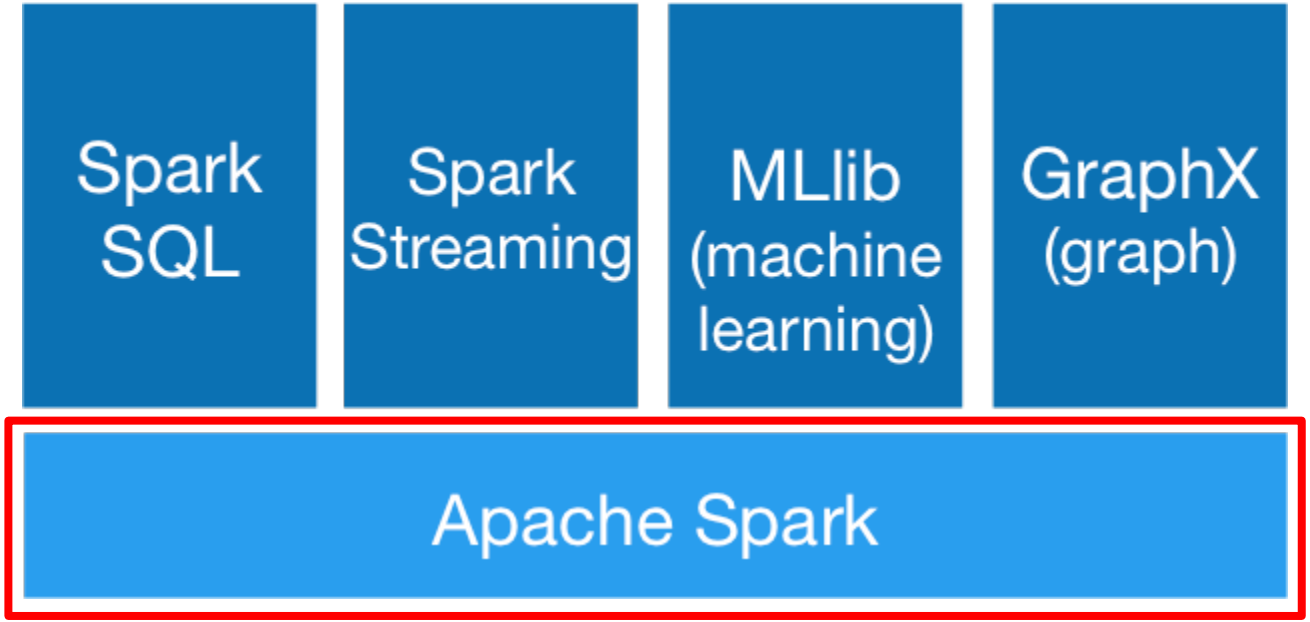
# Execution terms : SparkContext & Executor

- SparkContext
  - Spark cluster connection
  - Necessary for SQLContext, StreamingContext, …
  - Isolated: Two Spark Contexts cannot exchange data (without external help).
- Executor
  - Individual execution unit
  - 1 core ~ 2 executor
  - Each application has its owns.

# Job terms



# Modules


Spark SQL
Spark Streaming
MLlib (machine learning)
GraphX (graph)
Apache Spark

# RDD

- **R**esilient **D**istributed **D**ataset. Basic abstraction in Spark
- Think of it like a huge collection **partitioned** and distributed in different machines.
- Lazy evaluated
- Basically composed of
  - A list of partitions
  - A function for computing each split
  - A list of dependencies on other RDDs

# RDD

- Basic example:

```
val myRdd: RDD[String] = sc.parallelize(
  “this is a sample string”.split(“ ”).toList)

val anotherRdd: RDD[(Int,String)] = sc.parallelize(
  List(
    0 -> “A value for key ‘0’ ”,
    1 -> “Another value”))
```

# RDD

- An RDD may be created from
  - A file / set of files:

    ```
    sc.textFile("myFile")
    sc.textFile("file1,file2")
    ```

  - A bunch of memory-storaged data:

    ```
    sc.parallelize(List(1,2,3))
    ```

    Another RDD:

    ```
    myRdd.map(_.toString)
    ```

  - Another RDD

# RDD - Partitions

- Partition : RDD chunk
- A worker node may contain 1 or more partitions of some RDD.
- It’s important to choose a best-performance partitioning.
- Repartitioning
    - It implies shuffling all RDD data among worker nodes
    - There will surely be tons of network traffic among all worker nodes!

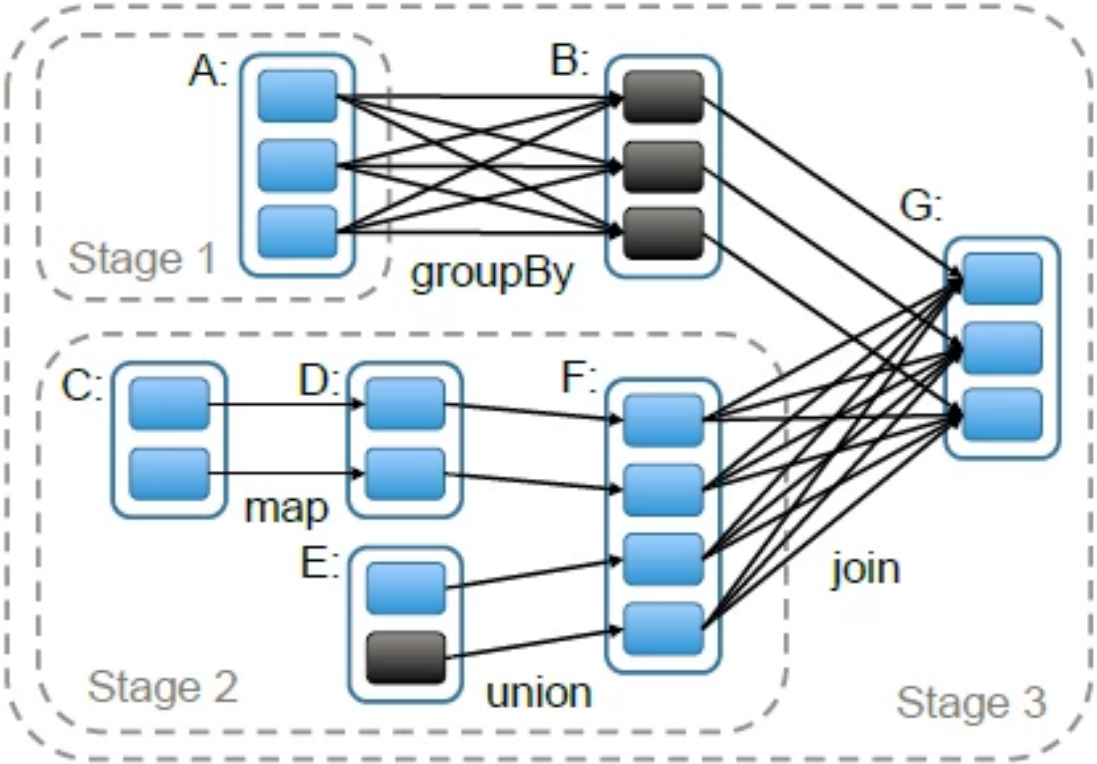
# RDD - Lineage

- An RDD may be built from another one.
- Base RDD : Has no parent
- Ex:

```
val base: RDD[Int] = sc.parallelize(List(1,2,3,4))
val even: RDD[Int] = base.filter(_%2==0)
```

- DAG (**D**irected **A**cyclic **G**raph) : Generalization of MapReduce model.
  Manages RDD lineages (source, mutations, …).

# RDD - Lineage



# RDD - Transformations

- It applies a change to some RDD, returning a new one.
- It’s not immediately evaluated.
- Think of it like a “Call-by-name” function. λ
- Adds a new node in Lineage DAG.

# RDD - Transformations

- Most frequently used:
  - **map**

```
val myRDD: RDD[Int] = sc.parallelize(List(1,2,3,4))
val myStrinRDD: RDD[String] = myRDD.map(_.toString)
```

  - **flatMap**

```
val myRDD: RDD[Int] =
  sc.parallelize(List(1,null,3,null))
val myNotNullInts: RDD[Int] =
  myRdd.flatMap(n => Option(n))
```

# RDD - Transformations

- Most frequently used:
  - **filter**

```
val myRDD: RDD[Int] = sc.parallelize(List(1,2,3,4))
val myStrinRDD: RDD[Int] = myRDD.filter(_%2==0)
```

  - **union**

```
val odd: RDD[Int] = sc.parallelize(List(1,3,5))
val even: RDD[Int] = sc.parallelize(List(2,4,6))
val all: RDD[Int] = odd.union(even)
```

# RDD - Actions

- It launches the RDD evaluation,
  - returning some data to the driver
  - or persisting to some external storage system
- It's evaluated partially on each worker node, and results are merged in the Driver
- There are mechanisms to make cheaper computing several times the same RDD (like `persist()` or `cache()` )

# RDD - Actions

- Examples:
  - **count**

```
val myRdd: Rdd[Int] = sc.parallelize(List(1,2,3))
val size: Int = myRdd.count
```

*Counting implies processing whole RDD*

  - **take**

```
val myRdd: Rdd[Int] = sc.parallelize(List(1,2,3))
val List(first,second) = myRdd.take(2).toList
```

  - **collect**

```
val myRdd: Rdd[Int] = sc.parallelize(List(1,2,3))
val data: Array[Int] = myRdd.collect
```

***Beware!*** *Executing 'collect' on big collections might end into a memory leak*

# Demo1 : Most retweeted

- Most Retweeted example
- Bunch of tweets (1000 json records)
- Find out which is the most retweeted tweet.

# Key-Value RDDs

- Particular case of `RDD[T]` where `T = (U,V)`
- It allows grouping,combining,aggregating values by some key.
- In Scala it's only needed to
  `import org.apache.spark.SparkContext._`
- In Java, it's mandatory to use `PairRDD` class.

# Key-Value RDDs

- keyBy: Generates a PairRDD

```
val myRDD: RDD[Int] =
  sc.parallelize(List(1,2,3,4,5))

val kvRDD: RDD[(String,Int)] =
  myRDD.keyBy(
    n => if (n%2==0) "even" else "odd")
```

*"odd" -> 1, "even" -> 2, "odd" -> 3, "even" -> 4, "odd" -> 5*

# Key-Value RDDs

- keys: Gets keys from PairRDD

```
val myRDD: RDD[(String,Int)] =
  sc.parallelize("odd" -> 1,"even" -> 2,"odd" -> 3)
val keysRDD: RDD[String] = myRDD.keys
```

*"odd","even","odd"*

- values: Gets values from PairRDD

# Key-Value RDDs

- mapValues: map PairRDD values, omitting keys.

```
val myRDD: RDD[(String,Int)] =
  sc.parallelize("odd" -> 1,"even" -> 2,"odd" -> 3)
val mapRDD: RDD[(String,String)] =
  myRDD.mapValues(_.toString)
```

*"odd" -> "1", "even" -> "2", "odd" -> "3"*

- flatMapValues: flatMap PairRDD values

# Key-Value RDDs

- join: Return a new RDD with both RDD joined by key.

```
val a = sc.parallelize(List("dog", "salmon", "salmon", "rat", "elephant"), 3)
val b = a.keyBy(_.length)
val c = sc.parallelize(List("dog","cat","gnu","salmon","rabbit","turkey","wolf","
  bear","bee"), 3)
val d = c.keyBy(_.length)
b.join(d).collect
```

```
res0: Array[(Int, (String, String))] = Array((6,(salmon,salmon)), (6,(salmon,rabbit)),
(6,(salmon,turkey)), (6,(salmon,salmon)), (6,(salmon,rabbit)), (6,(salmon,turkey)), (3,
(dog,dog)), (3,(dog,cat)), (3,(dog,gnu)), (3,(dog,bee)), (3,(rat,dog)), (3,(rat,cat)),
(3,(rat,gnu)), (3,(rat,bee)))
```

# Key-Value RDDs - combineByKey

- combineByKey:

```
def combineByKey[C](
  createCombiner: V => C,
  mergeValue: (C,V) => C,
  mergeCombiners: (C,C) => C): RDD[(K,C)]
```

- Think of it as *something-like-but-not* a `foldLeft` over each partition λ

# Key-Value RDDs - combineByKey

- It’s composed by:
  - `createCombiner( V => C)` : Sets the way to mutate initial RDD[V] data into new data type used for aggregating values (C). This will be called Combinator.
  - `mergeValue((C,V) => C)` : Defines how to aggregate initial V values to our Combiner type C, returning a new combiner type C.
  - `mergeCombiners((C,C) => C)` : Defines how to merge two combiners into a new one.

# Key-Value RDDs - combineByKey

- Example:

```
val a = sc.parallelize(List("dog","cat","gnu","salmon","rabbit","
  turkey","wolf","bear","bee"), 3)
val b = sc.parallelize(List(1,1,2,2,2,1,2,2,2), 3)
val c = b.zip(a)
val d = c.combineByKey(List(_), (x:List[String], y:String) => y :: x,
  (x:List[String], y:List[String]) => x ::: y)
d.collect
```

*res16: Array[(Int, List[String])] = Array((1,List(cat, dog, turkey)), (2,List(gnu, rabbit, salmon, bee, bear, wolf)))*

# Key-Value RDDs - aggregateByKey

- **aggregateByKey** : Aggregate the values of each key, using given combine functions and a neutral “zero value”.
- *Beware*! Zero value is evaluated in each partition.

```
def aggregateByKey[U:ClassTag](zeroValue: U)(
  seqOp: (U,V) => U, combOp: (U,U) => U): RDD[(K,U)] =
  combineByKey(
    (v: V) => seqOp(zeroValue,v),
    seqOp,
    comboOp)
```

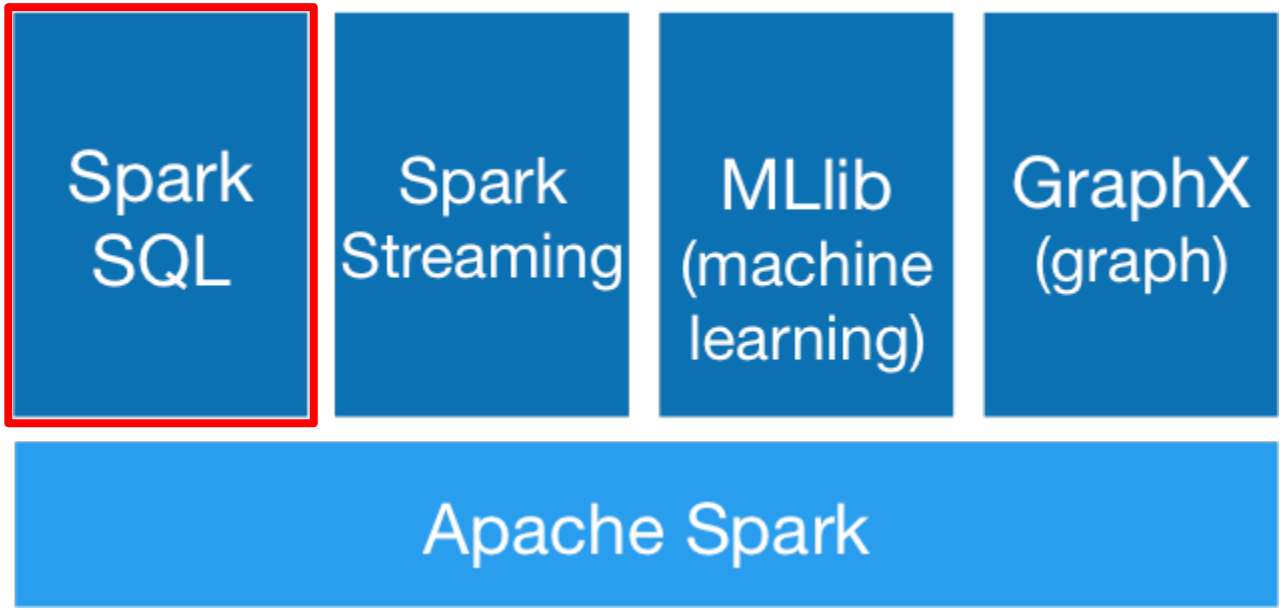
# Key-Value RDDs - groupByKey

- **groupByKey**: Group all values with same key into a Iterable of values

```
def groupByKey(): RDD[(Key,Iterable[Value])] =
  combineByKey[List[Value]](
    (v: V) => List(v),
    (list: List[V],v: V) => list+= v,
    (l1: List[V],l2: List[V]) => l1 ++ l2)
```

*Note: GroupByKey actually uses CompactBuffer instead of list.*

# Modules


Spark SQL
Spark Streaming
MLlib (machine learning)
GraphX (graph)
Apache Spark


INVENTS MAPREDUCE
DOESN'T SHARE HIGH-LEVEL LANGUAGE SUPPORT
SCUMBAGSTEVE.COM

# SparkSQL

- Shark successor (Berkeley)
  - Alternative to Hive + Hadoop
- Query language
  - HiveQL
  - Future: Support full SQL92
- SQLContext

```
val sqlContext = new SQLContext(sparkContext)
val hiveContext = new HiveContext(sparkContext)
```

# SparkSQL

- Different datasources
  - JSON
  - Parquet
  - CSV
- New implementations
  - Cassandra
  - ElasticSearch
  - MongoDB
- Unified API in Spark 1.3.0

```
val students: DataFrame = sqlContext.load(
  “students”,“org.apache.sql.parquet”,Map(...))
```

# SparkSQL - DataFrame

- DataFrame = RDD[org.apache.spark.sql.Row] + Schema
- A Row holds both column values and their types.
- This allows unifying multiple datasources with the same API.
  *i.e, we could join two tables, one declared on Mongo and another on ElasticSearch.*

# Demo2 : Most retweeted(SparkSQL)

- Most Retweeted Example (SparkSQL)
- Same bunch of tweets (1000 json records)
- Find out which is the most retweeted tweet using SparkSQL

# Who uses it?

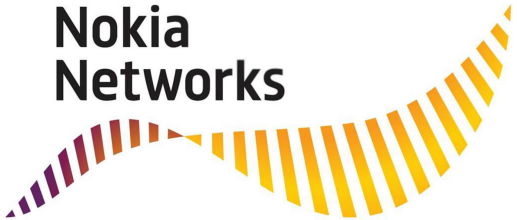

amazon


databricks


YAHOO!


ebay


Almaden
Institute
IBM


Nokia
Networks


NASA
JPL
National Aeronautics and Space
Administration
Jet Propulsion Laboratory
California Institute of Technology


STRATIO

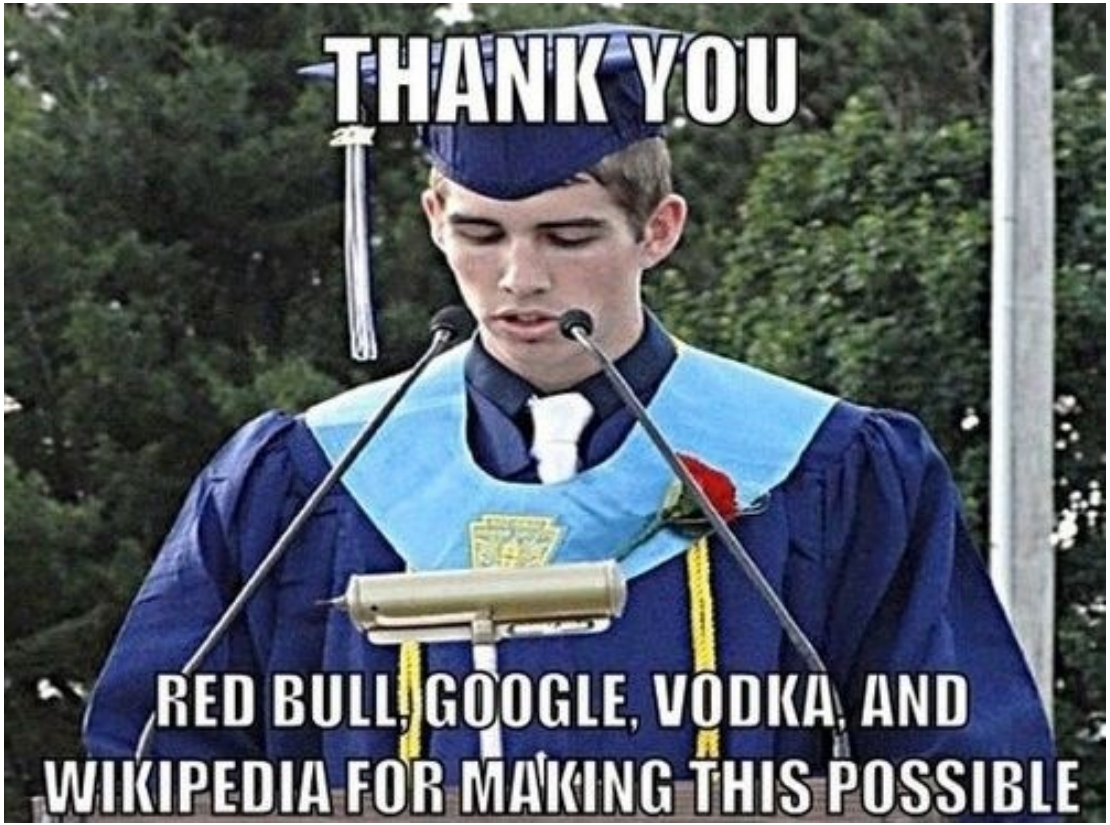

cloudera


TREND
MICRO


Simba
technologies

# Let’s talk about...

- Spark streaming
- Machine learning (MLlib)
- GraphX vs Neo4j
- Spark vs Hadoop
- ?


THANK YOU
RED BULL, GOOGLE, VODKA, AND
WIKIPEDIA FOR MAKING THIS POSSIBLE

# Introduction to


Javier Santos

May,11 - 2015