

# Who are you

- Java developer since 2001
- Java Trainer since 2003
- Java consultant since 2008
- Java architect since 2009
- DemocraC startup founder since 2010
- Big Data Technical Leader at Naya technologies since 2015



# Spark

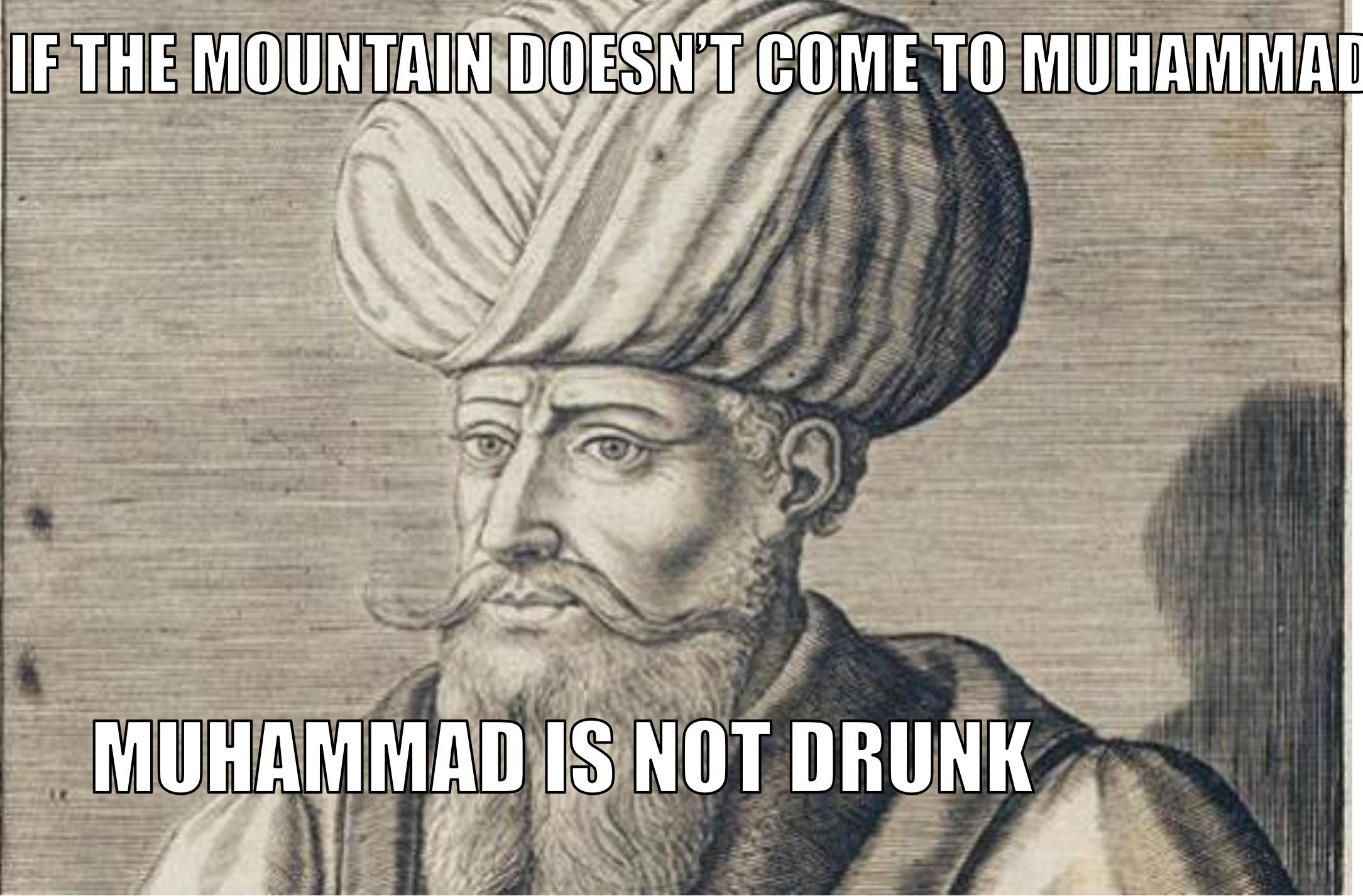
evgenyb@naya-tech.co.il

# Agenda

- Hadoop ecosystem
- Functional programming
- Scala – what we need to know
- Spark API
- Using spring in order to write infrastructure for working with spark
- Running Spark & Tests

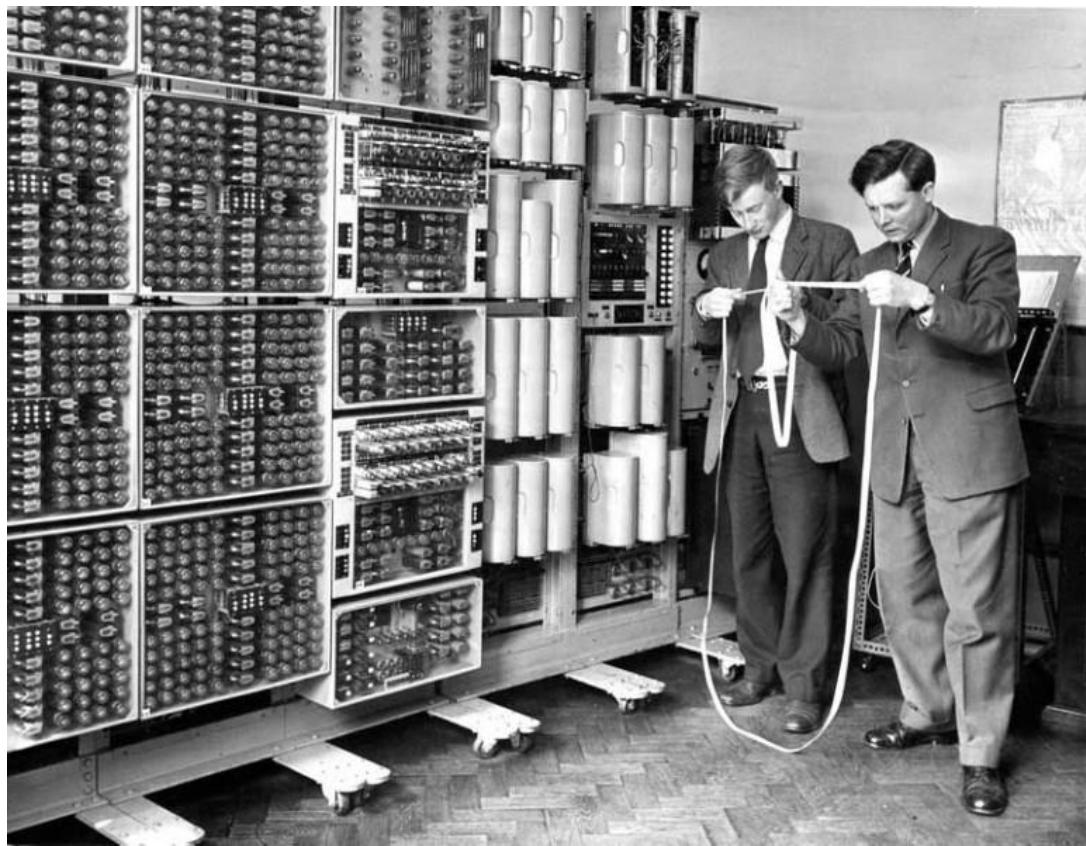
# Data locality

**IF THE MOUNTAIN DOESN'T COME TO MUHAMMAD**

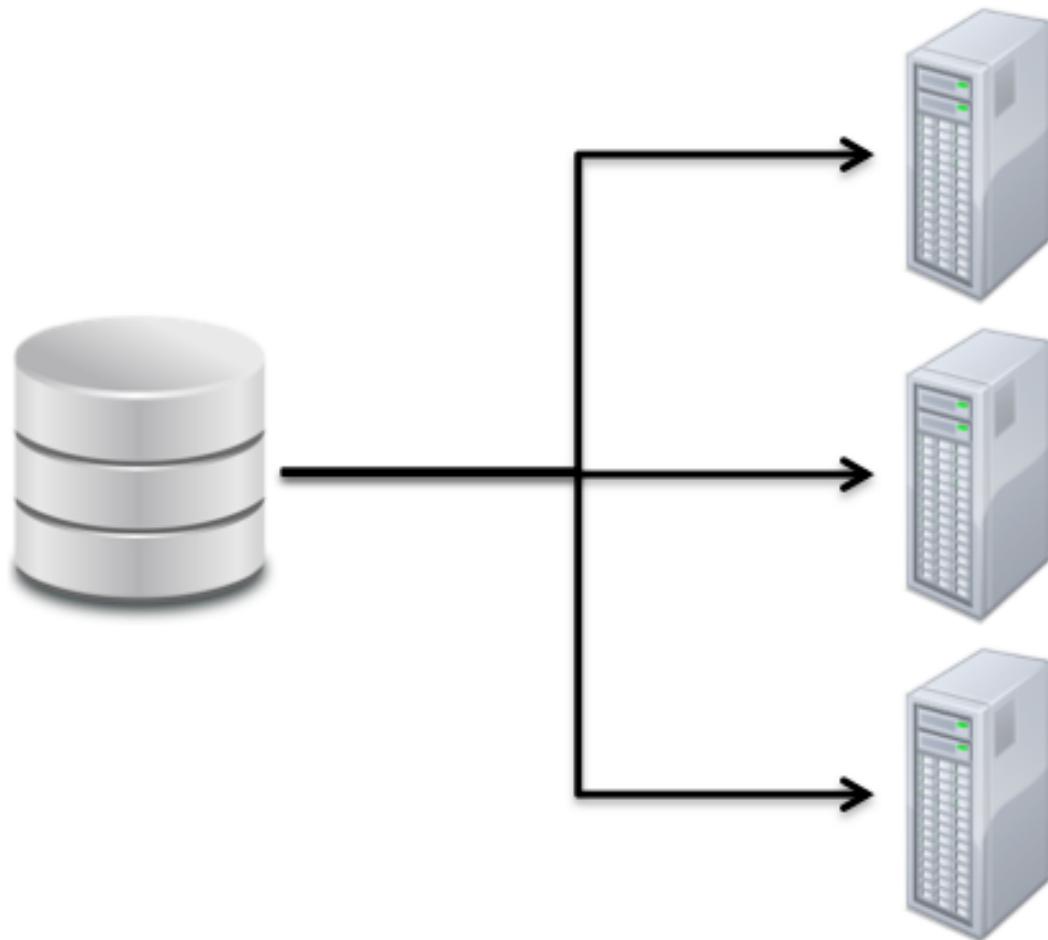


**MUHAMMAD IS NOT DRUNK**

# Yesterday against today



# The bottleneck

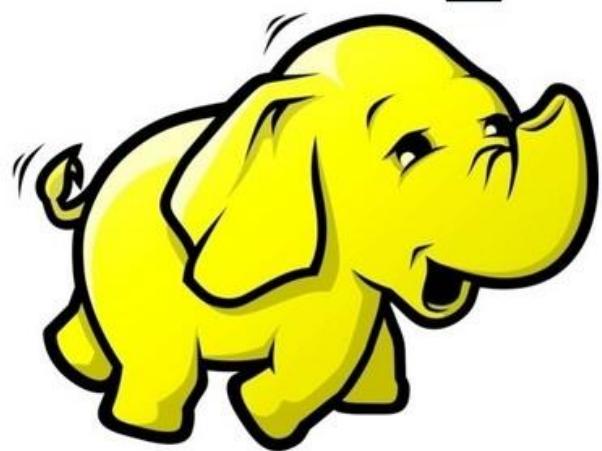


# New model



Hadoop Sucks....

***hadoop***



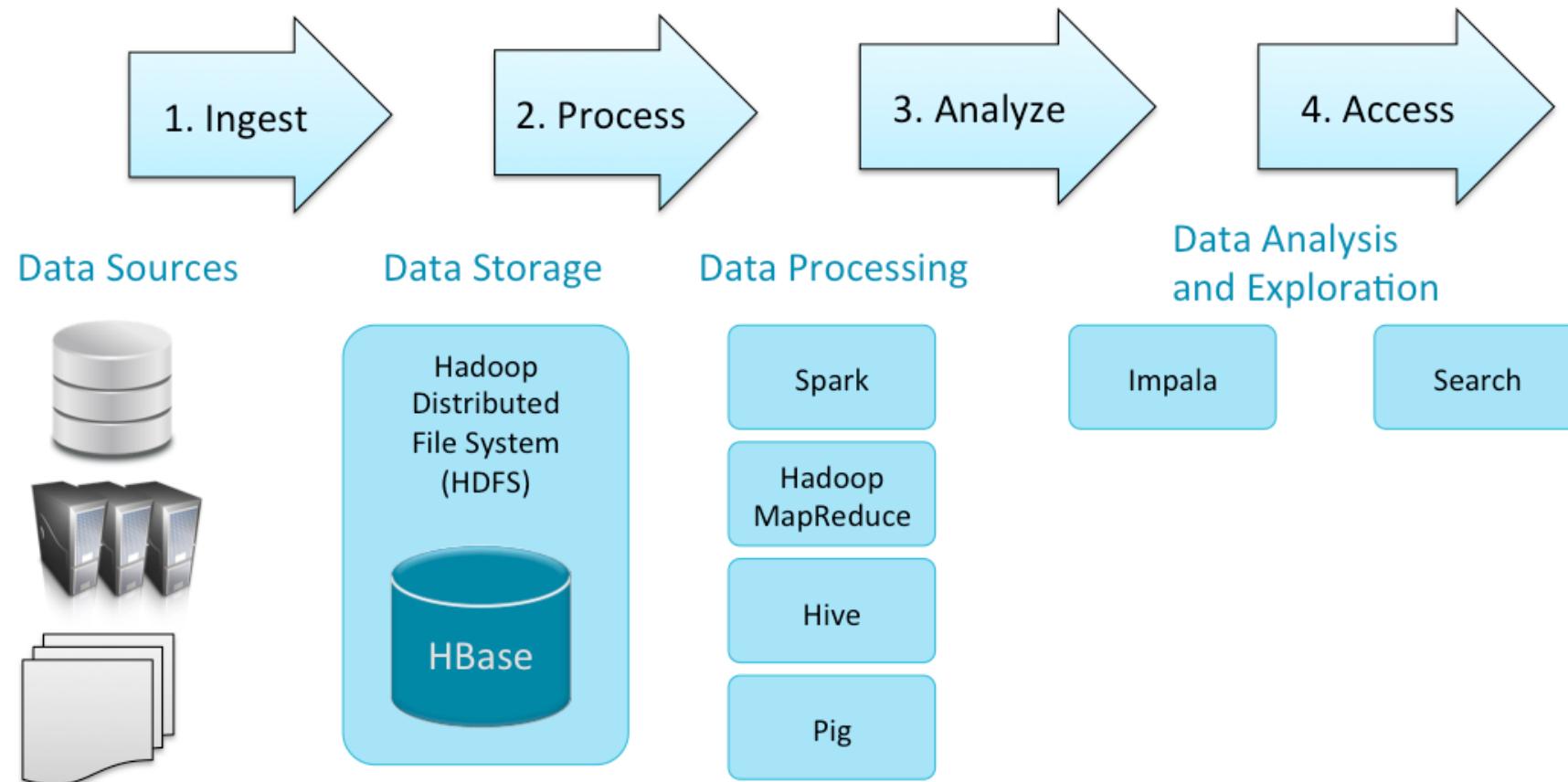
# Hadoop implementations

- Vanilla
- Hortonworks
- Cloudera

# Hadoop ecosystem includes

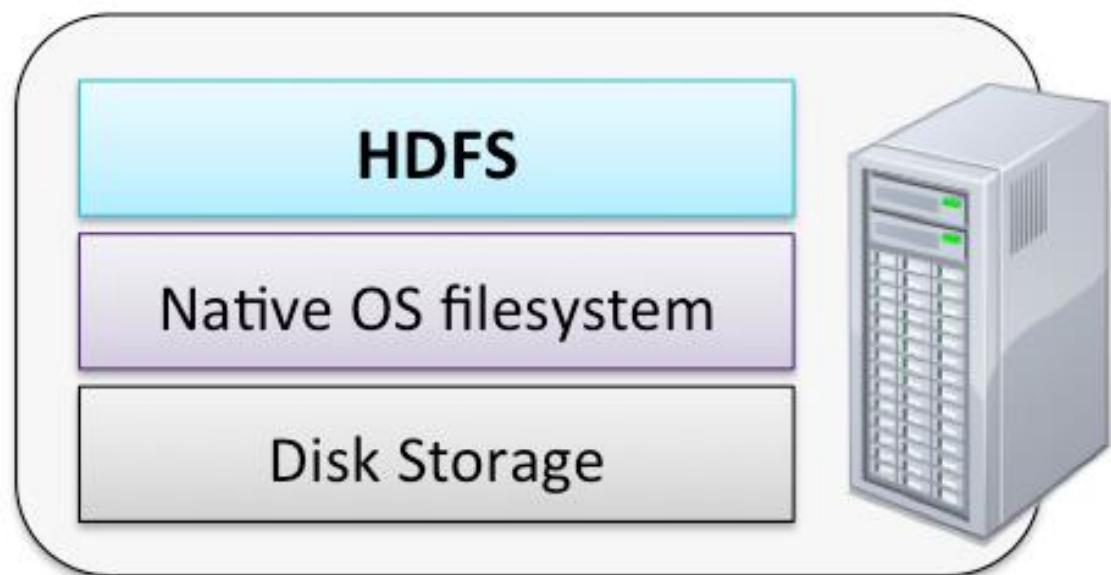
- HDFS
- HBASE
- Hue
- Yarn
- Sqoop
- Flume
- Hive
- Spark (Cloudera, Hortonworks)
- Solr
- Pig
- Impala (Cloudera)
- Zookeeper
- Oozie

# The process



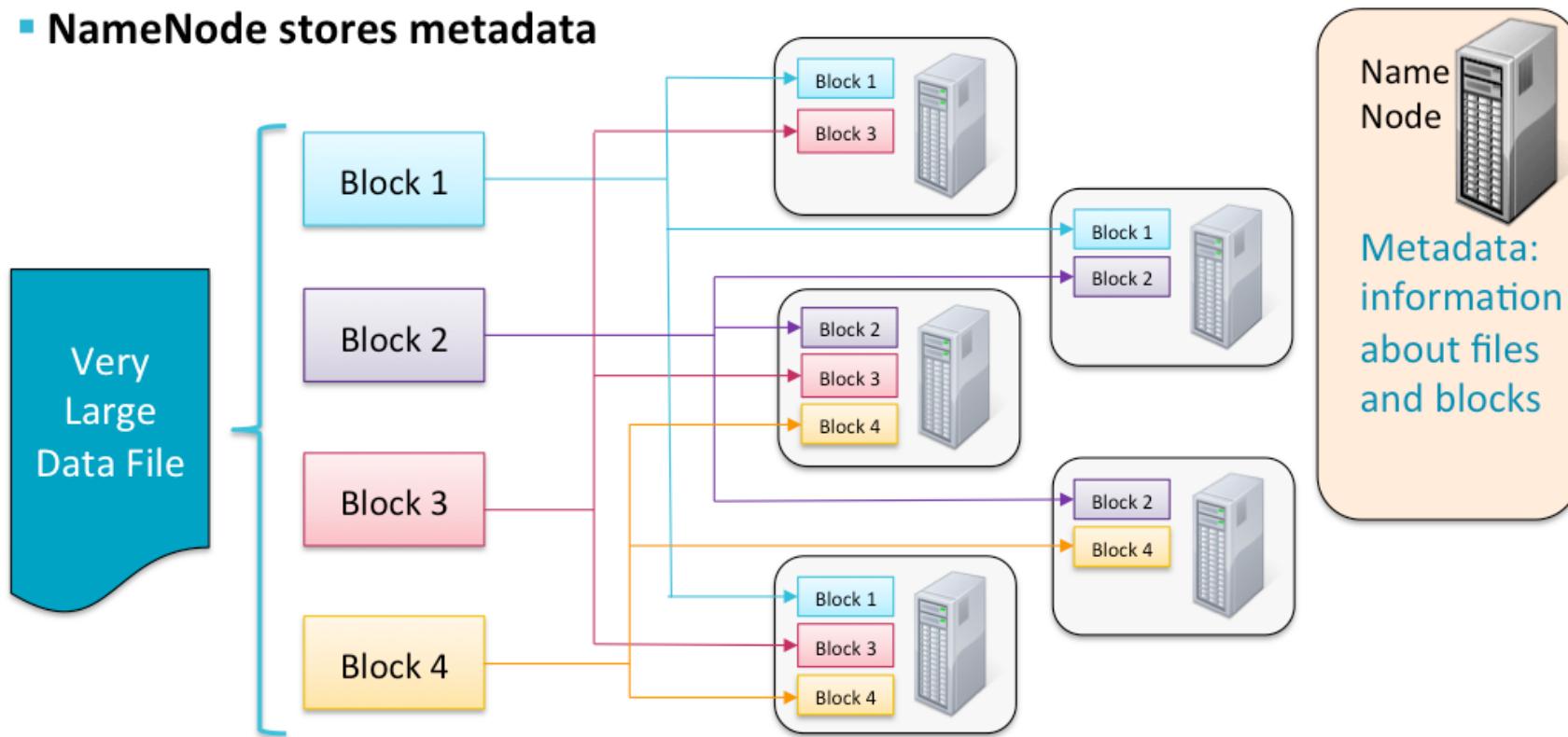
# Storing data in Hadoop

- Hadoop Distributed File System (HDFS)
- Apache Hbase - no sql distributed database



# How files stored in HDFS

- Data files are split into 128MB blocks which are distributed at load time
- Each block is replicated on multiple data nodes (default 3x)
- NameNode stores metadata



### Metadata

/logs/031512.log: B1,B2,B3  
/logs/042313.log: B4,B5

**B1:** A, B, D

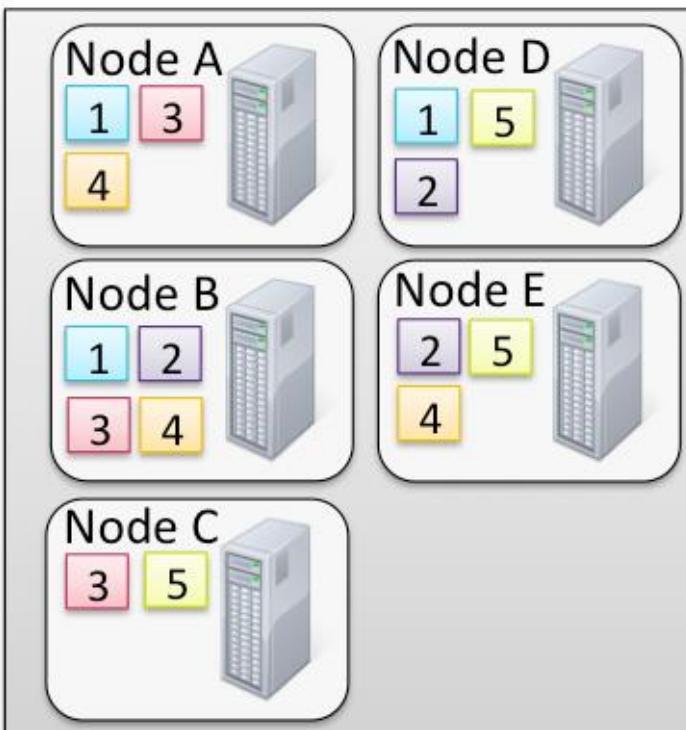
**B2:** B, D, E

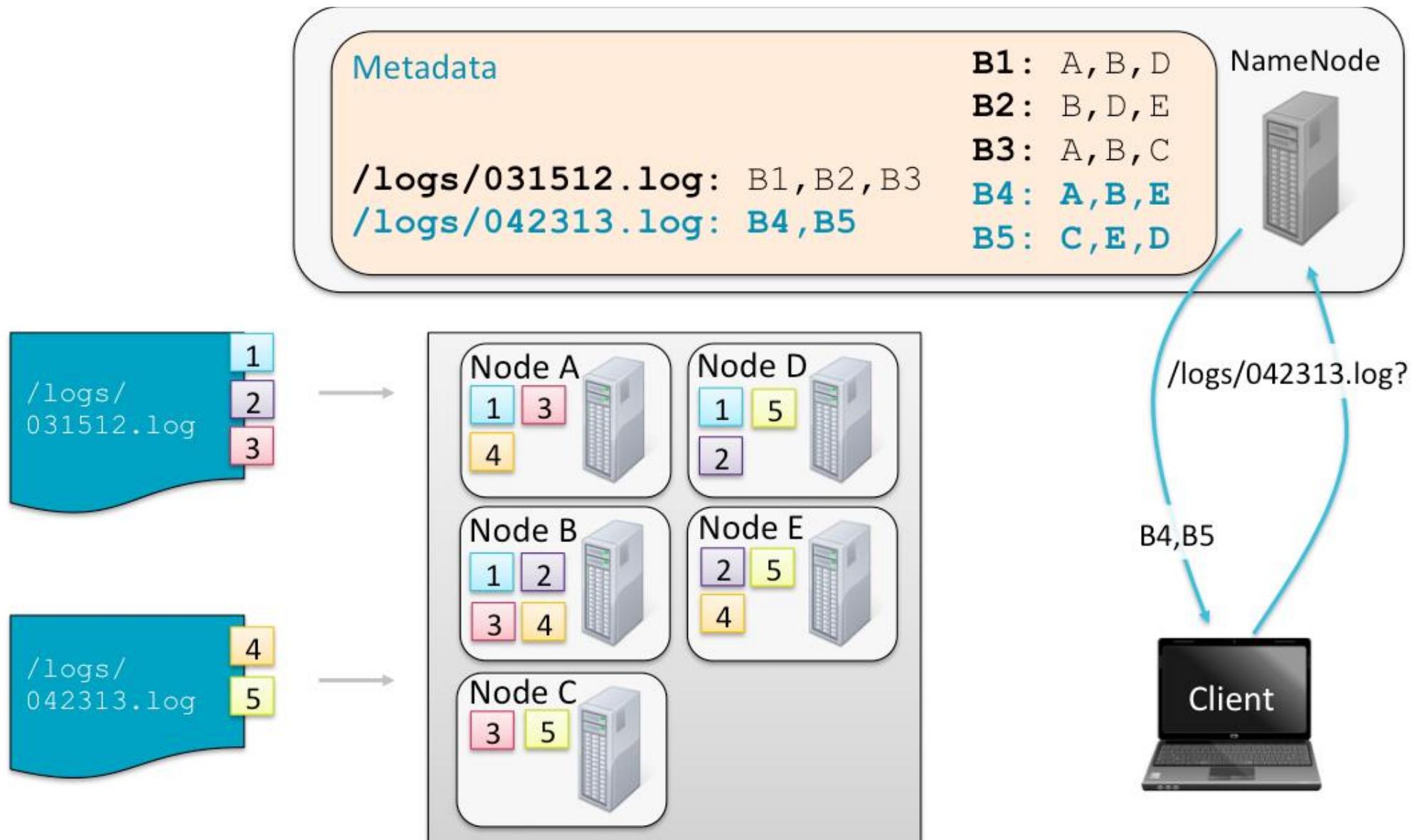
**B3:** A, B, C

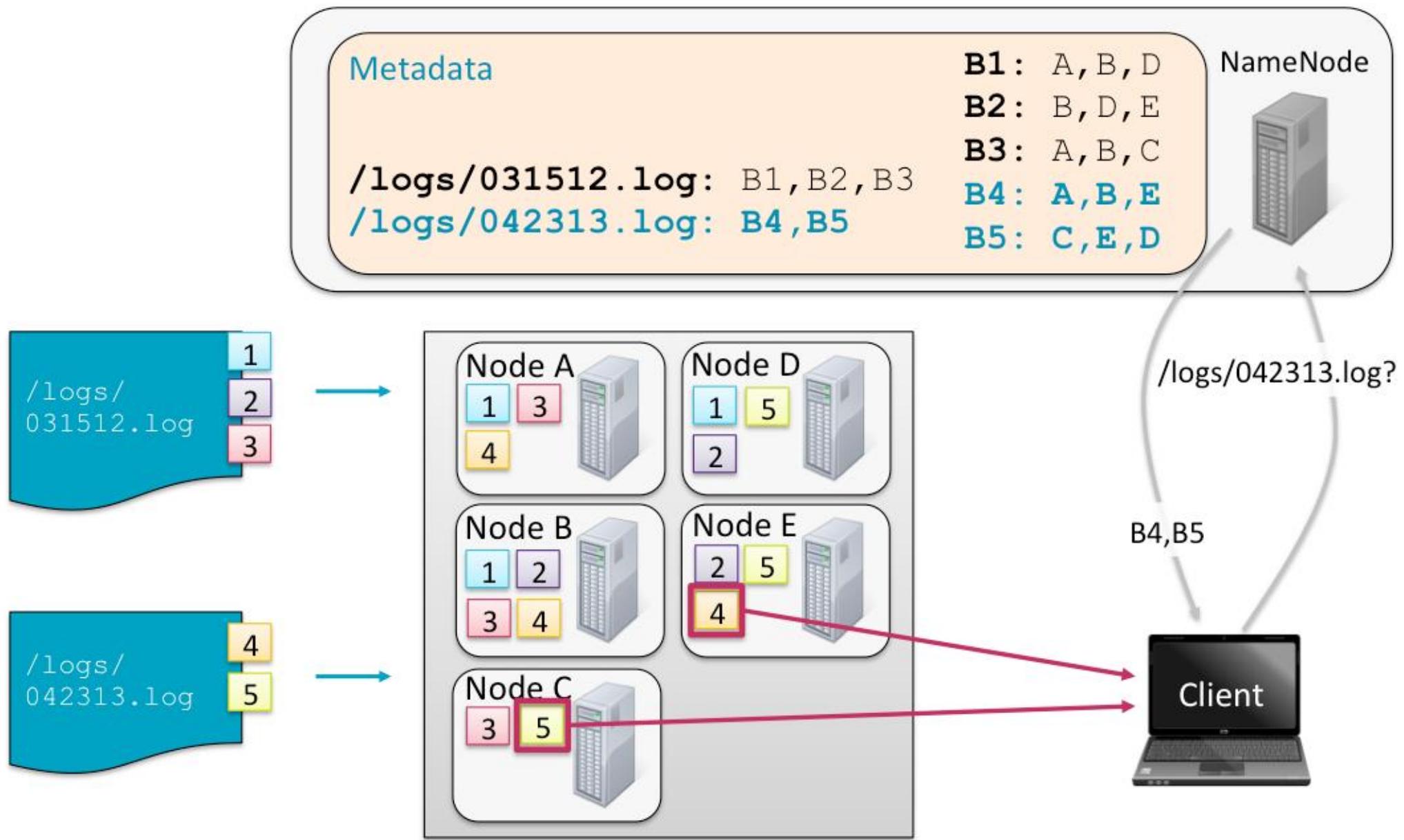
**B4:** A, B, E

**B5:** C, E, D

NameNode







# HDFS: cmd

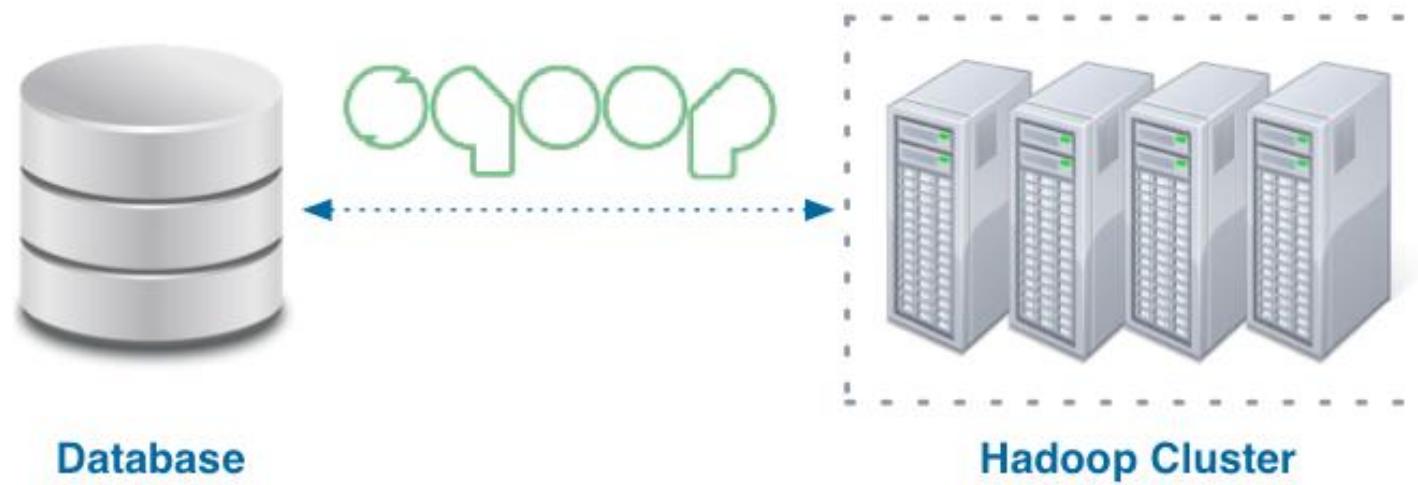
- hdfs dfs –put bigFile.txt bigFile.txt
- hdfs dfs –ls
- hdfs dfs –get /user/jeka/ bigFile.txt bigFile.txt
- hdfs dfs –mkdir myDir
- hdfs dfs –rm –r backupDir

# BUT ALL OUR DATA IN RELATIVE DATABASES



# Sqoop

- Tool for migrating data from relative db to Hadoop and visa versa
- Supports
  - Netezza, Mongo, MySQL, Teradata, Oracle



# Import Data

- sqoop --all-tables
  - connect jdbc:mysql://host/schema
  - username user
  - password 1234

# Import Data

- sqoop --all-tables
  - connect jdbc:mysql://host/schema
  - username user
  - password 1234
  - incremental lastmodified
  - check-column date
  - last-value '2016-06-10 11:00:00'

# Import Data

- sqoop --all-tables
  - connect jdbc:mysql://host/schema
  - username user
  - password 1234
  - columns “id, name, date”
  - where “state='UA' ”

# Import Data

- sqoop --all-tables
  - connect jdbc:mysql://host/schema
  - username user
  - password 1234
  - incremental lastmodified
  - check-column id
  - last-value 66666

# Import Data

- sqoop --all-tables
  - connect jdbc:mysql://host/schema
  - username user
  - password 1234
  - m 16 (parallelism. Default is 4)

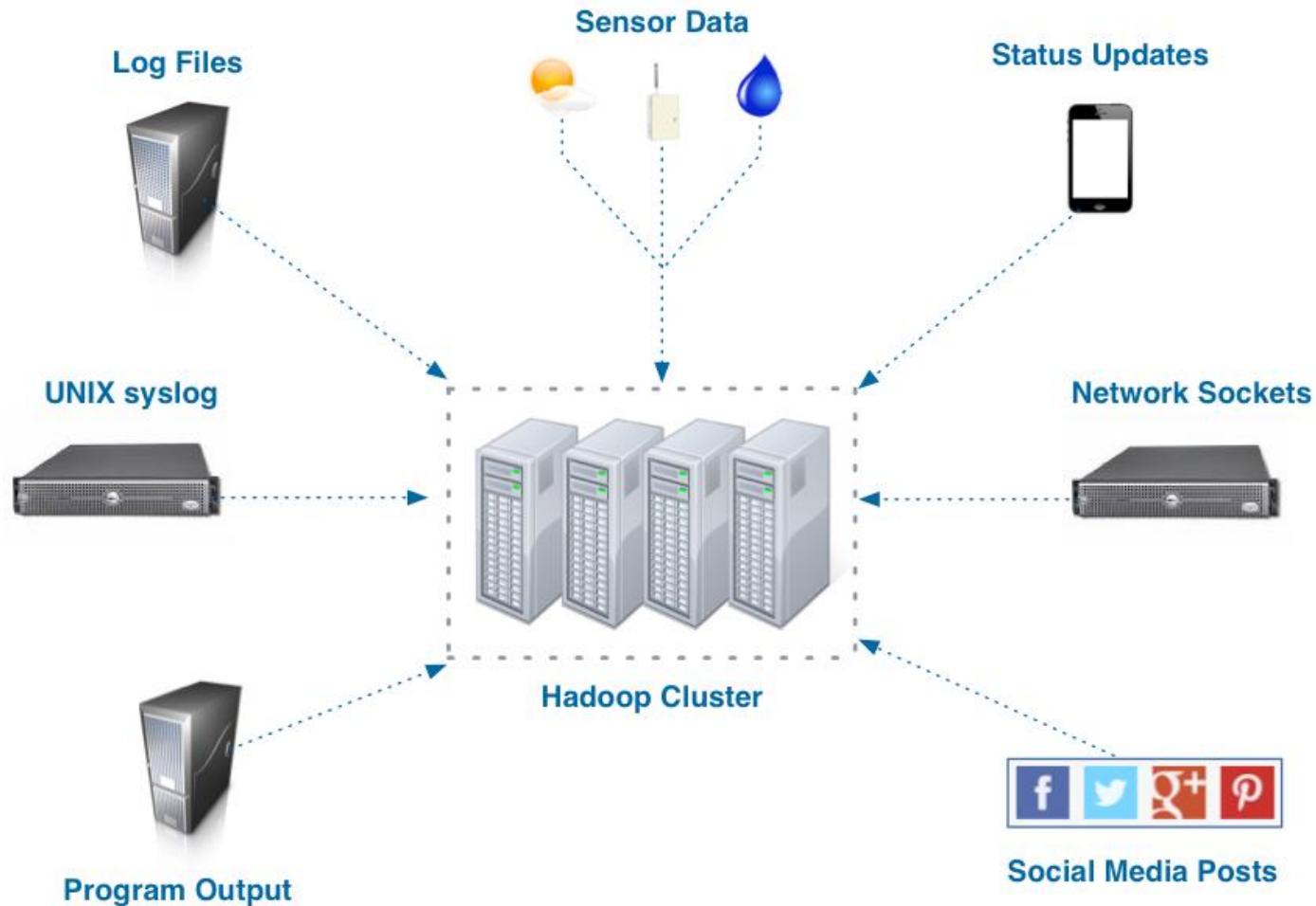
# Export

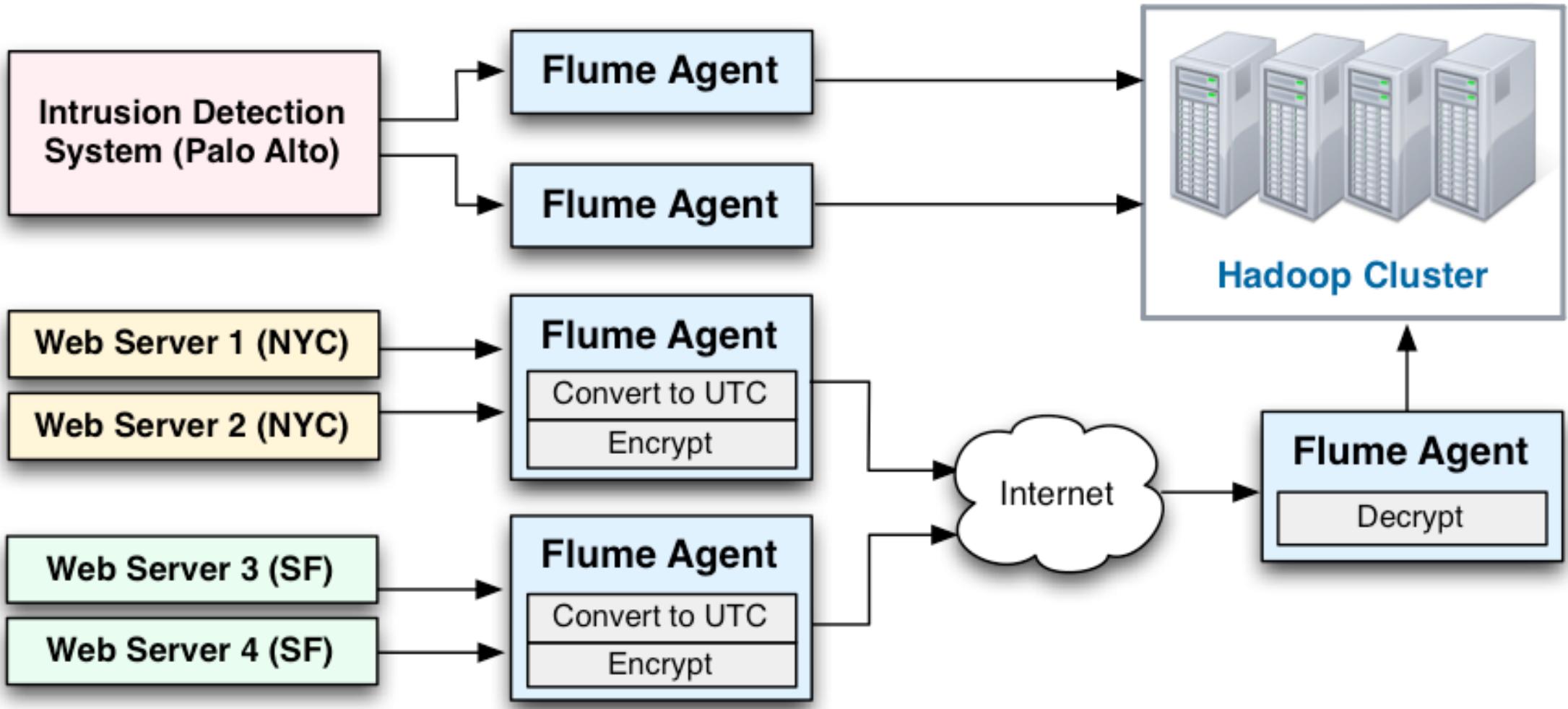
```
sqoop export \
  --connect jdbc:mysql://dbhost/loudacre \
  --username dbuser --password pw \
  --export-dir /loudacre/recommender_output \
  --update-mode allowinsert \
  --table product_recommendations
```

# Apache flume

- Distributed service for ingesting streaming data
- Good for reading logs, messaging system, kafka (mostly)
- Can be replaced by spark streaming

# Where from Flume can fetch data?

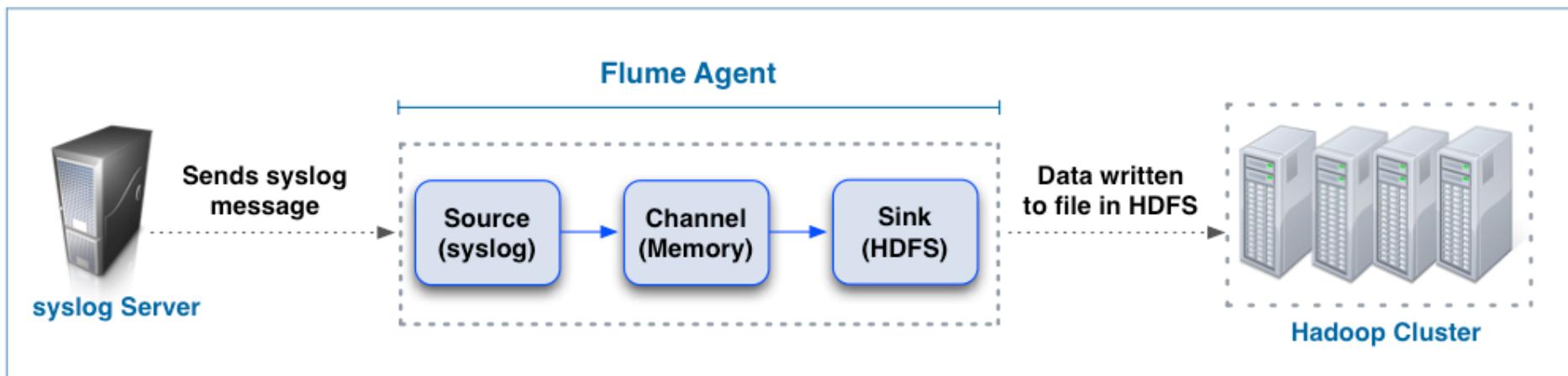




# Channel can be changed (not only memory)

- This diagram illustrates how syslog data might be captured to HDFS

1. Message is logged on a server running a syslog daemon
2. Flume agent configured with syslog source receives event
3. Source pushes event to the channel, where it is buffered in memory
4. Sink pulls data from the channel and writes it to HDFS



# Flume channels

- Memory – RAM
  - Super fast, but in case rocket will fail, all data will be lost
- File – stores data on local disk
  - Works slower, but you rocket is not a dangerous anymore
- JDBC – stores data in database with JDBC
  - Work even more slower, but you dba's can will not be unemployed

Everything configured via property file

```
$ bin/flume-ng agent -n $agent_name -c conf -f conf/flume-conf.properties.template
```

```
# Name the components on this agent
a1.sources = r1
a1.sinks = k1
a1.channels = c1
# Describe/configure the source
a1.sources.r1.type = netcat
a1.sources.r1.bind = localhost
a1.sources.r1.port = 44444
# Describe the sink
a1.sinks.k1.type = logger
# Use a channel which buffers events in memory
a1.channels.c1.type = memory
a1.channels.c1.capacity = 1000
a1.channels.c1.transactionCapacity = 100
# Bind the source and sink to the channel
a1.sources.r1.channels = c1
a1.sinks.k1.channel = c1
```

# Hive

- Uses SQL-like syntax (HiveQL)
- Delegate to map reduce API, so not so fast
- Good for batch processes
- Convenient for dba's

# Impala (cloudera)

- Like Hive, but much faster
- Written on C
- Work on cluster
- Uses ram
- Works very fast
- Needs a lot of free memory (RAM)

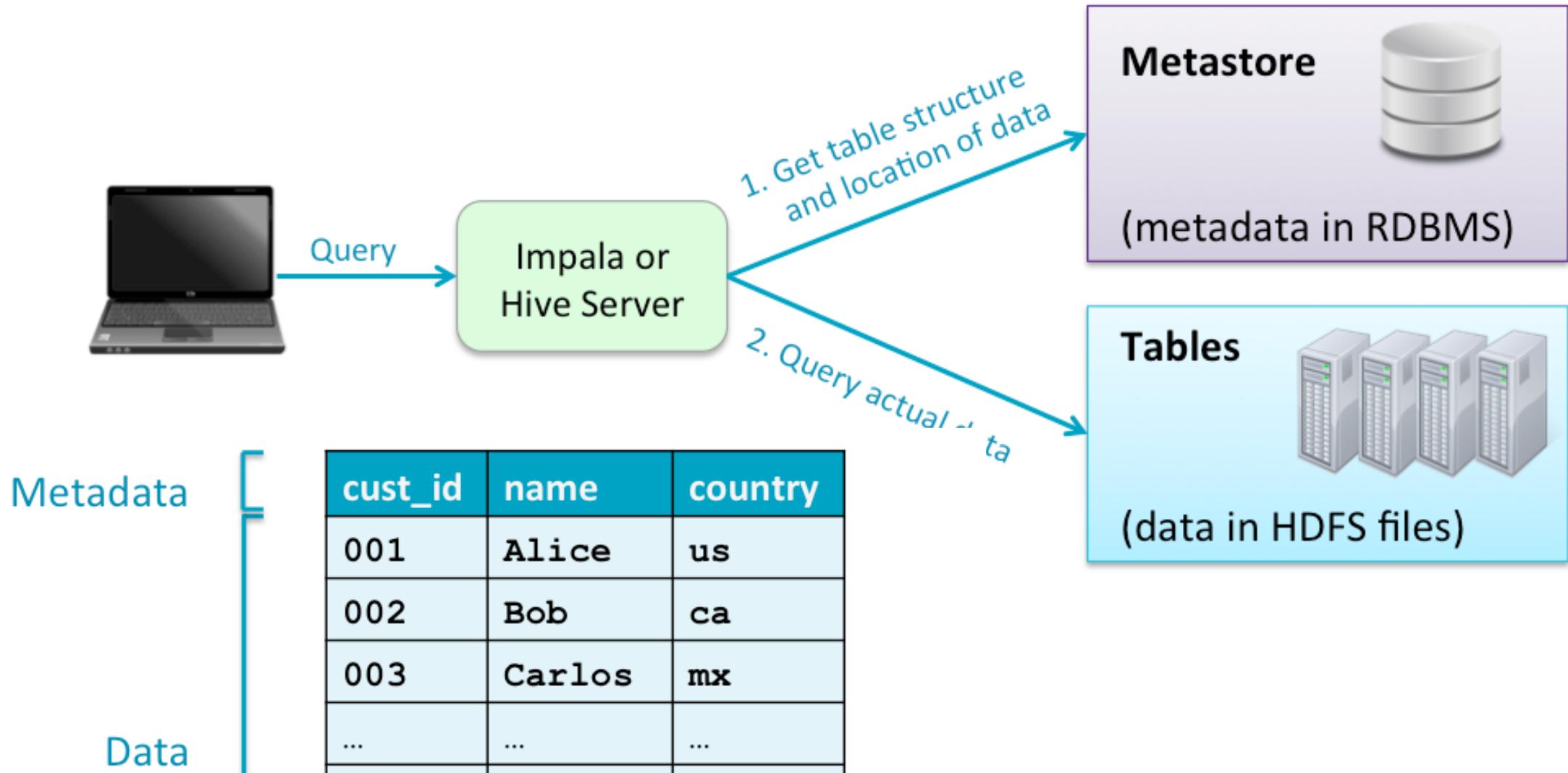
# Why not to use Impala instead of Hive always?

- Needs a lot of RAM
- Needs a lot of RAM
- Officially is supported only by Cloudera

# Hive / Impala syntax example

```
> SELECT lname, fname FROM customers WHERE state = 'CA'  
limit 50;  
  
Query: select lname, fname FROM customers WHERE state =  
'CA' limit 50  
+-----+-----+  
| lname      | fname       |  
+-----+-----+  
| Ham        | Marilyn    |  
| Franks     | Gerard     |  
| Preston    | Mason      |  
| Cortez     | Pamela     |  
| ...         |            |  
| Falgoust   | Jennifer   |  
+-----+-----+  
Returned 50 row(s) in 0.17s
```

# How impala / hive stores data



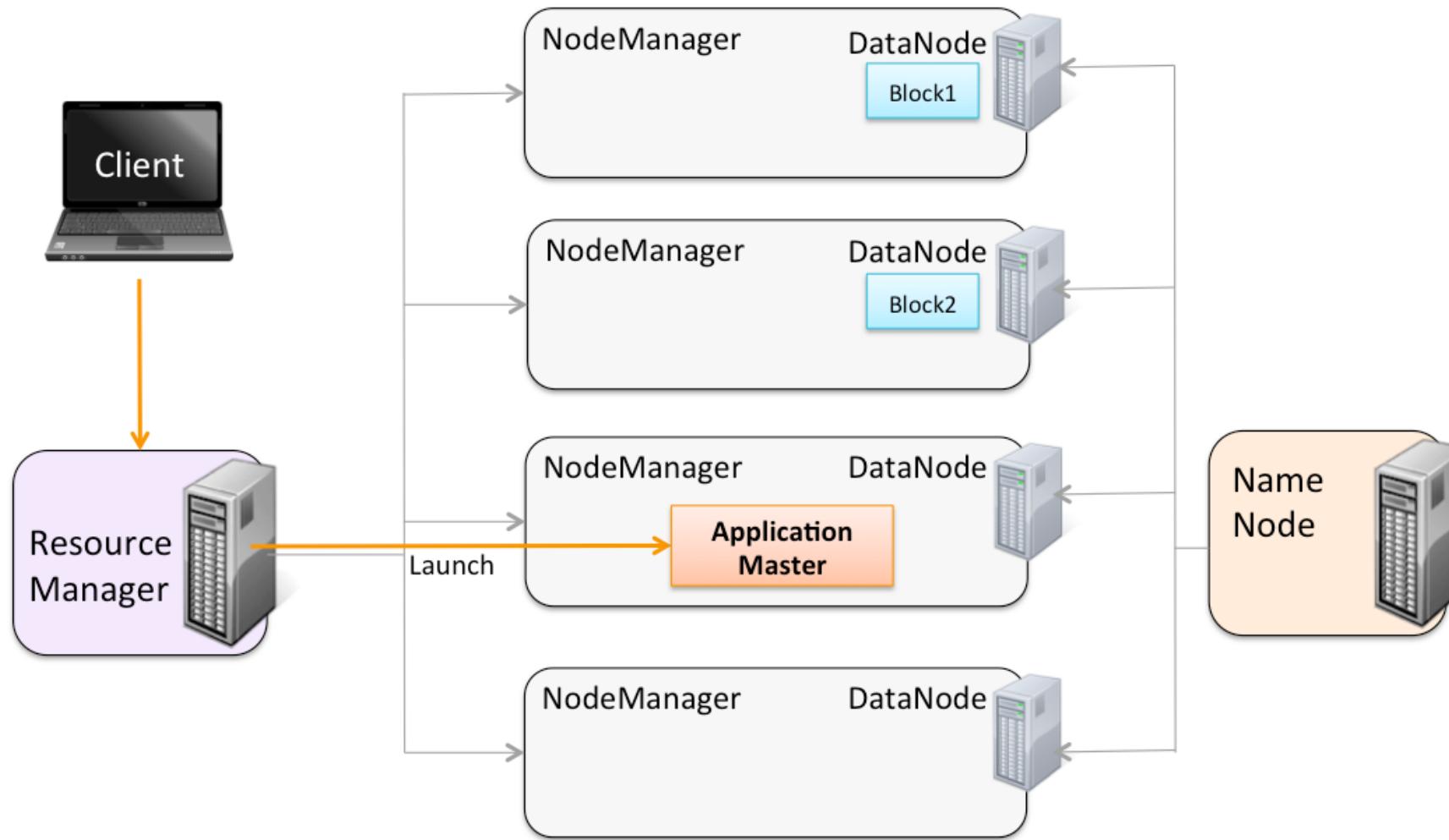
# Where tables come from?

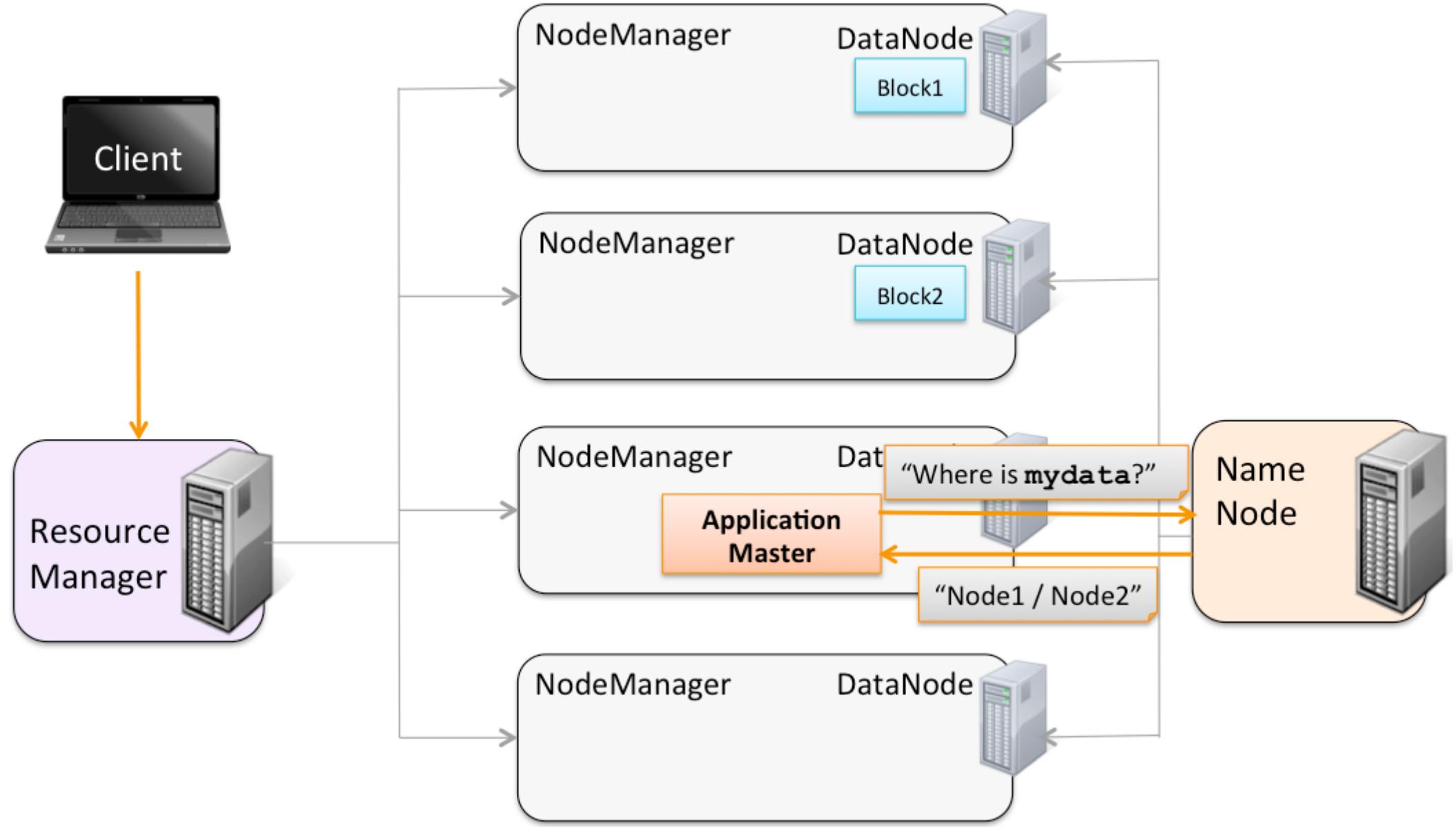
```
CREATE TABLE tablename (colname DATATYPE, . . .)
  ROW FORMAT DELIMITED
    FIELDS TERMINATED BY char
  STORED AS {TEXTFILE|SEQUENCEFILE|...}
```

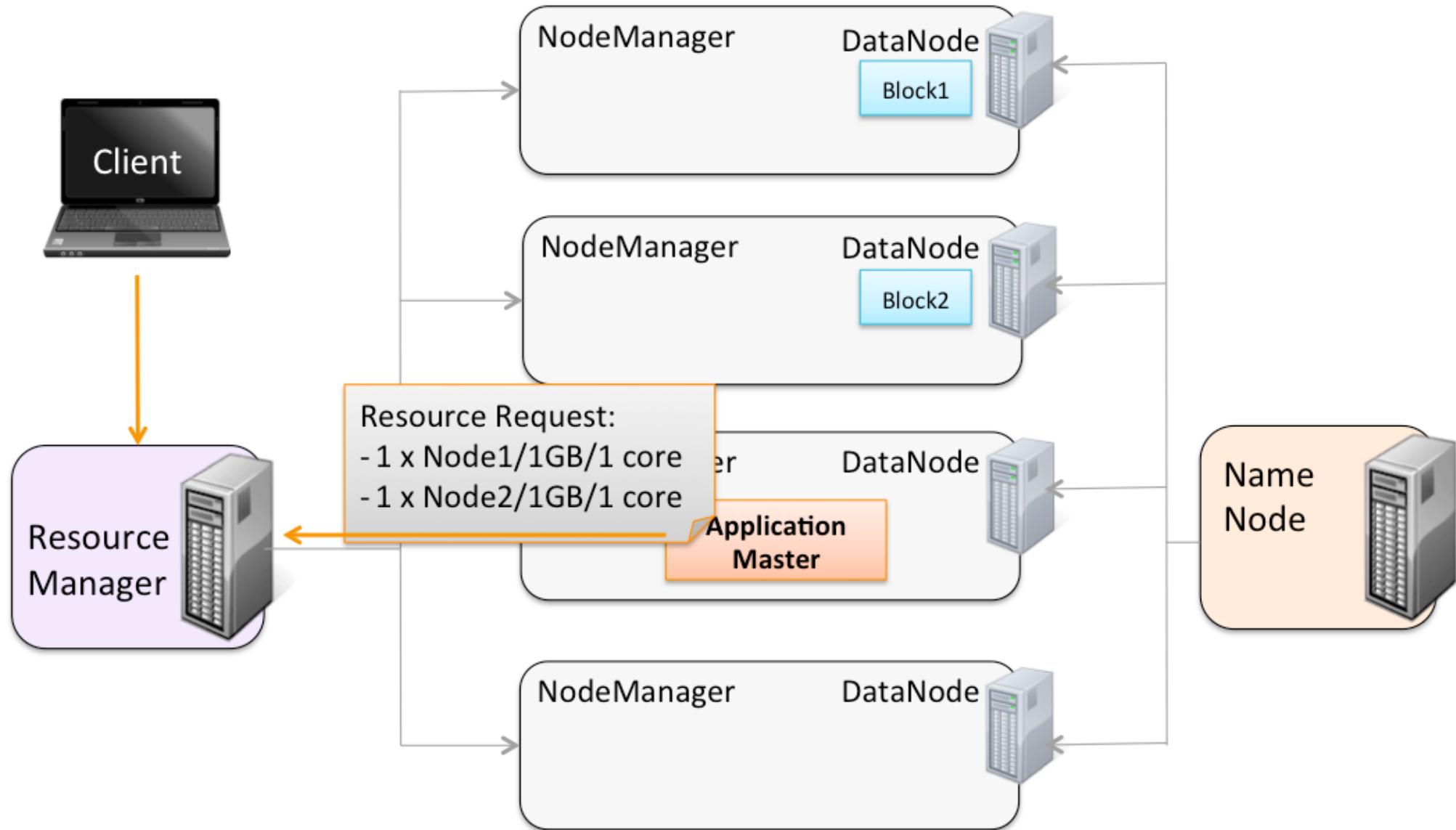
# Yarn – yet another resource negotiator

- Resource manager – coordinates all cluster processes
- Job scheduler
- Allows to run several task for processing data simultaneously
  - Batch programs(Spark, Map Reduce)
  - Interactive SQL (Impala)
  - Advanced Analytics(Spark, Impala)
  - Streaming (Spark Streaming)

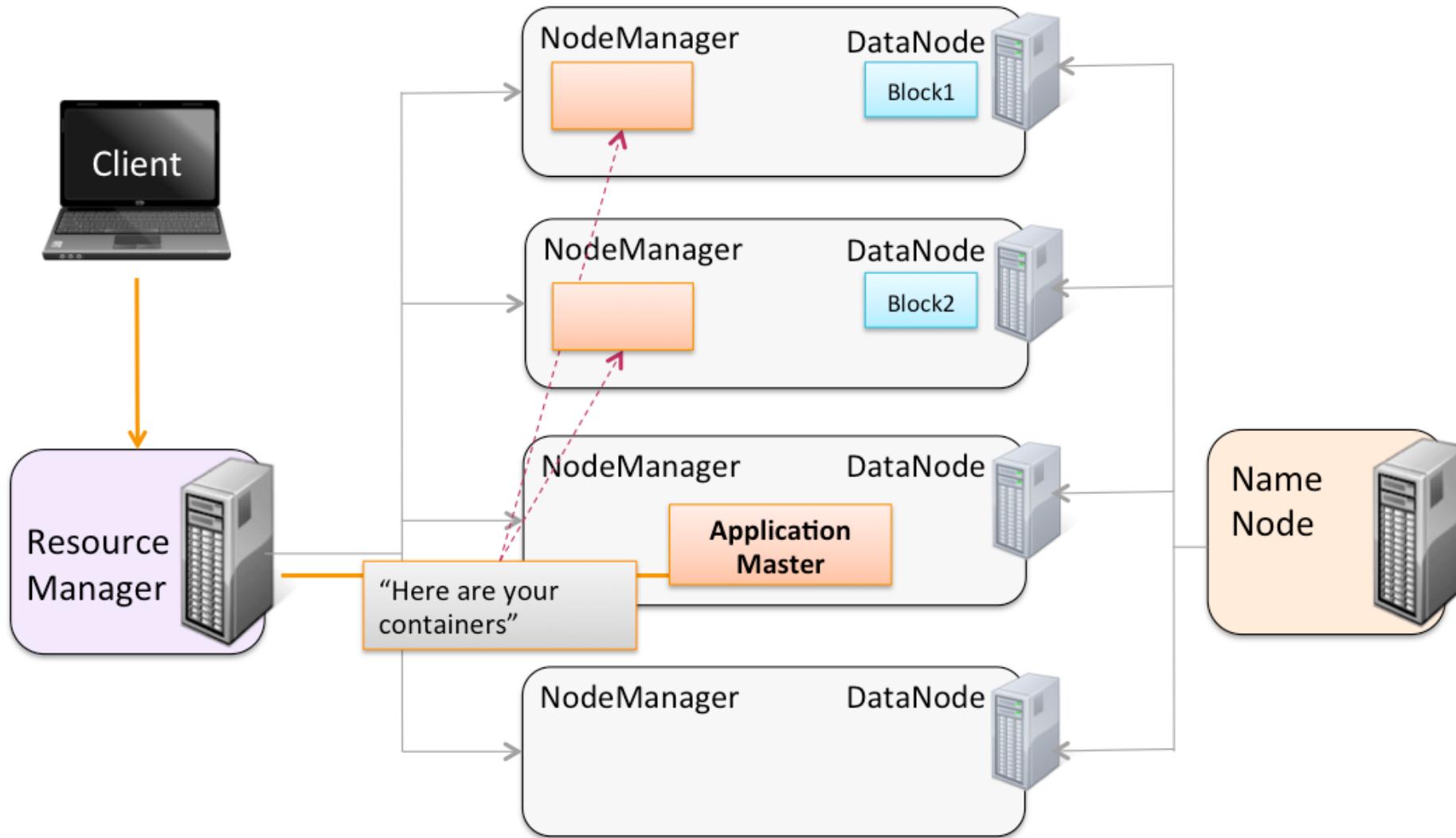
# Running with yarn

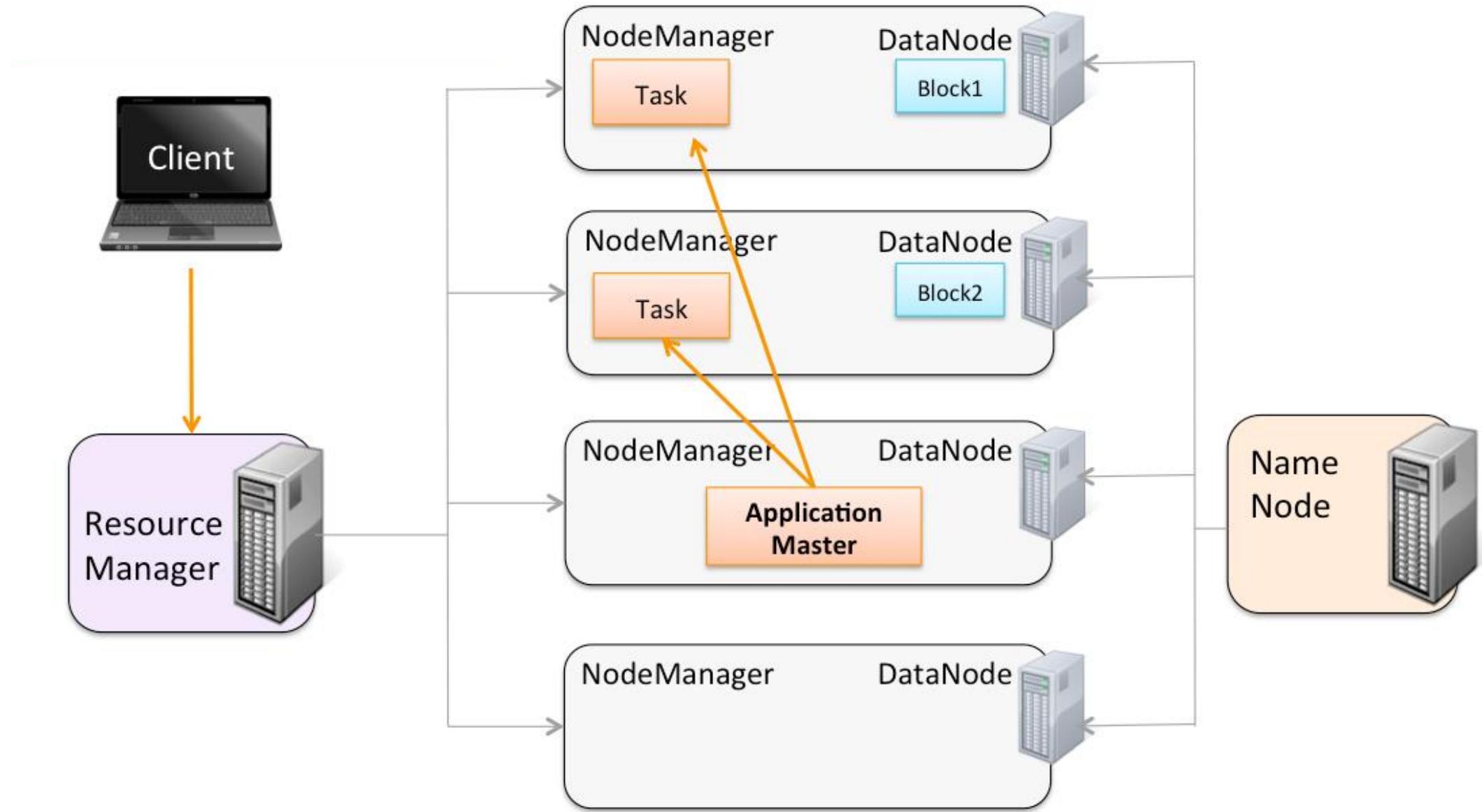


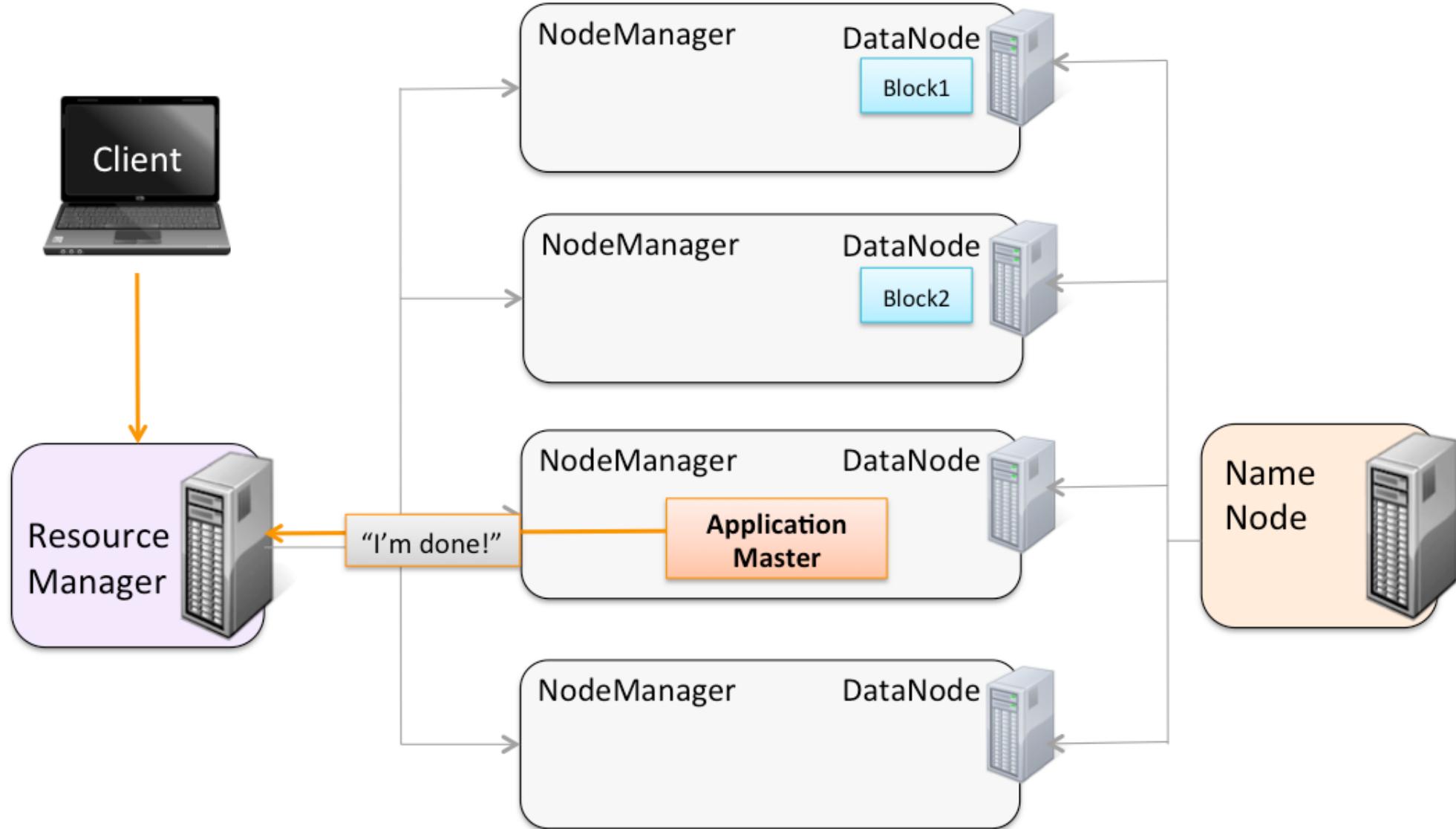




Will try to allocate the same machines which store the processed data





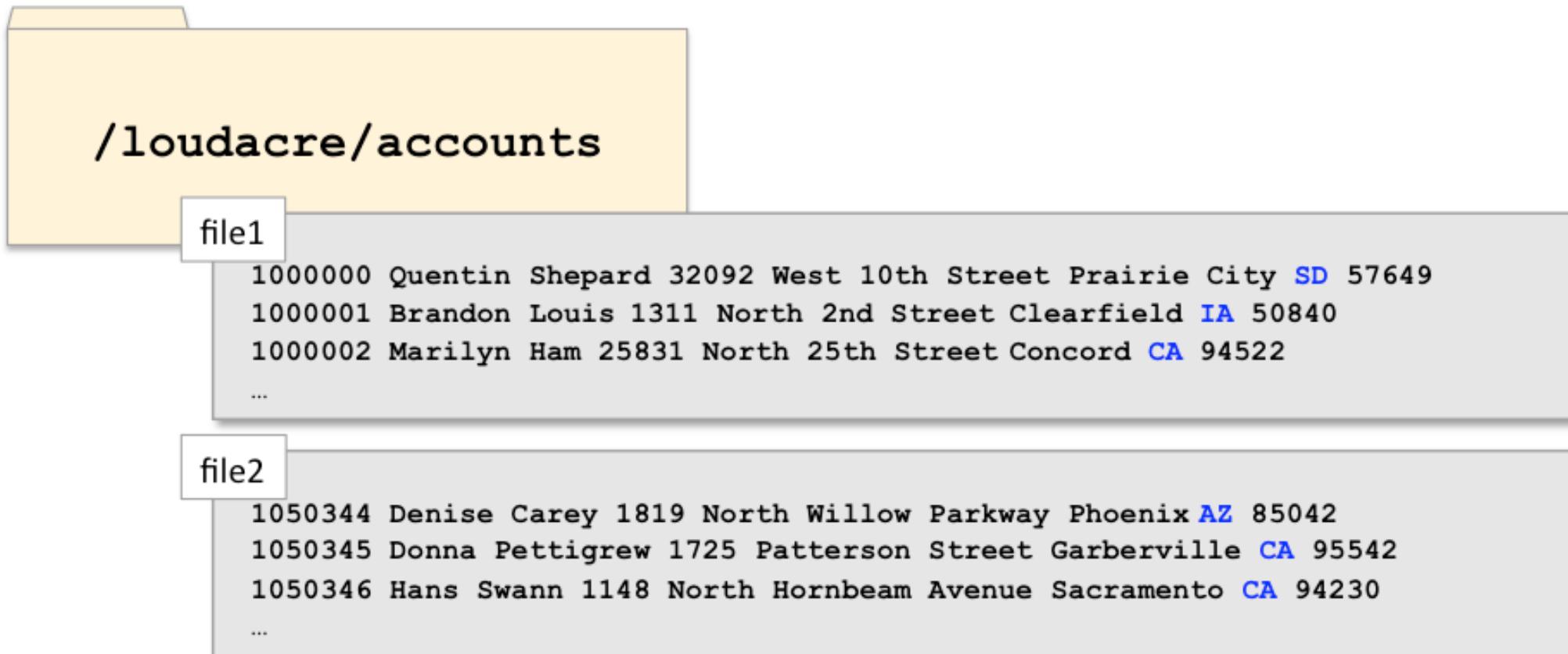


# File formats you should know

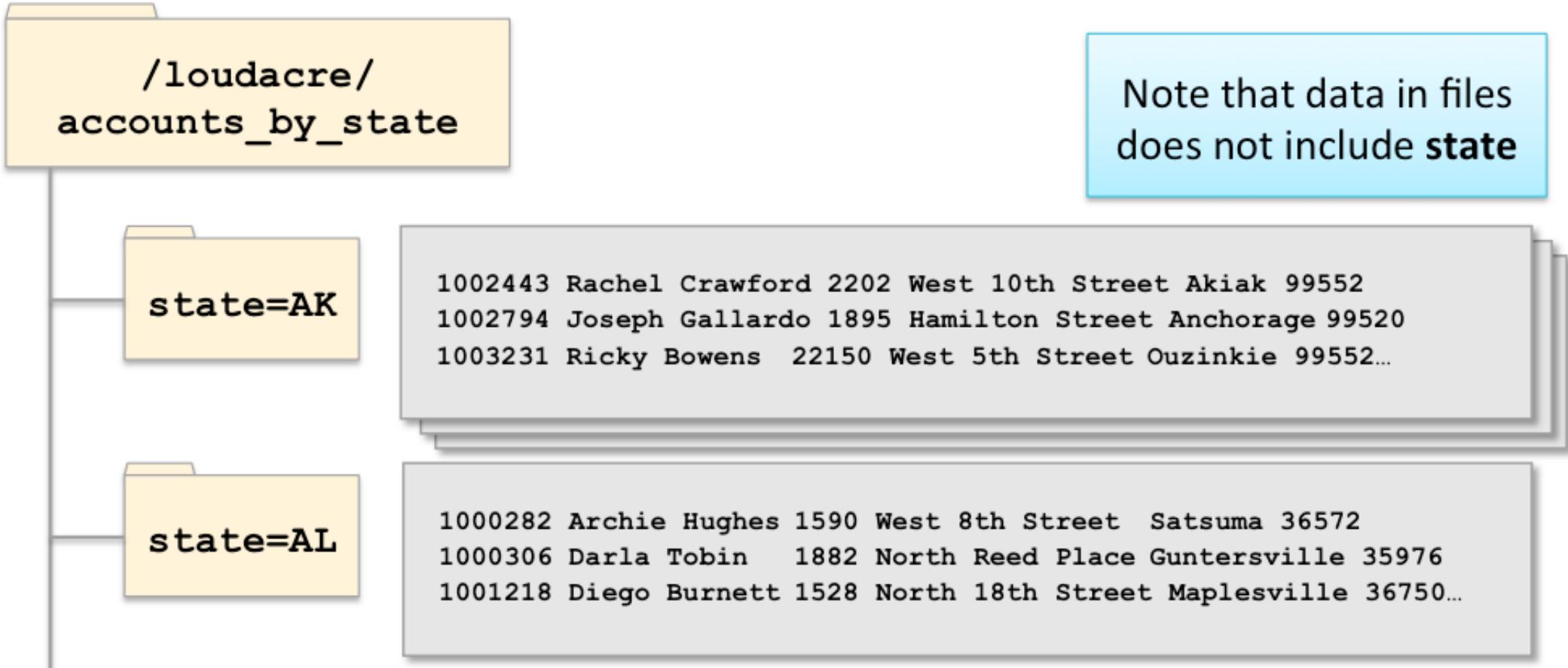
- parquet - like csv but compressed
- avro – like json, but compressed

# Partitioning

- By default all files of dataset exist in one HDFS directory



# More efficient



This is not my Job – sysadmin should know it



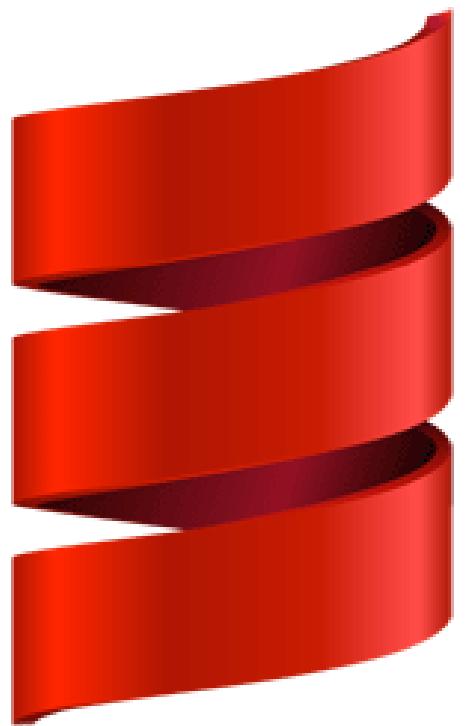
Шеф сказал: "ПОЧИНИТЬ!"...

A close-up of Aragorn's face from The Lord of the Rings. He has long brown hair and a beard, looking slightly to the side with a serious expression. He is wearing a dark leather-like armor.

ONE DOES NOT SIMPLY

CAN WRITE EFFECTIVE SPARK CODE

If you do not understand  
how your data will be  
stored you will not be able  
to write effective code



Scala

# What is Scala

**Scala was developed by Martin Odersky starting in 2001**

- Sca (scalable) La (language)

**Scala is a superset of Java**

- Scala runs on the Java Virtual Machine (JVM) and compiles to Java bytecode, and is compatible with Java programs
- All Java programs can be ported to Scala, but some Scala programs cannot be ported to Java

**Multi-paradigm**

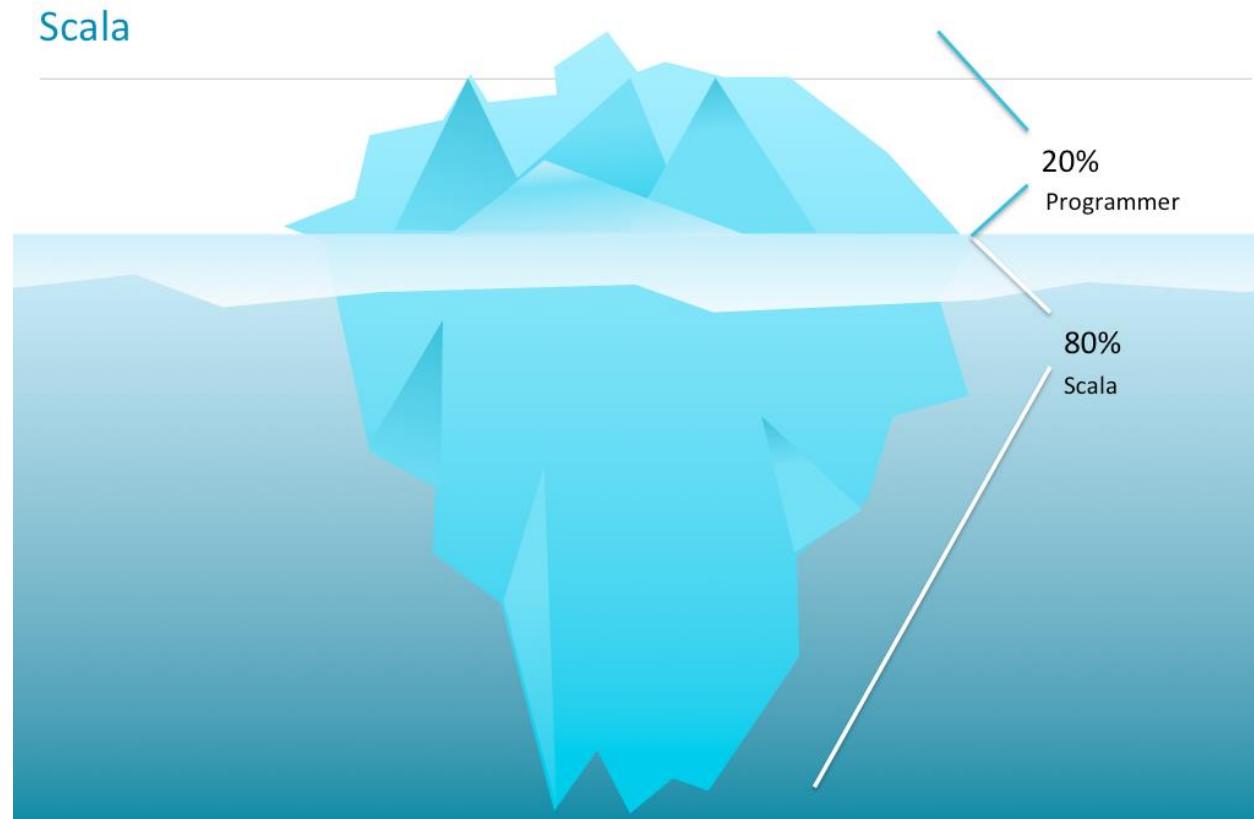
- Scala has features of imperative programming, object oriented programming, and functional programming, allowing the developer to cross paradigms within a single program as needed

# Scala concepts

- Pure functions
- Immutable data
- Implicit looping and iteration over collections

# Scala & Java

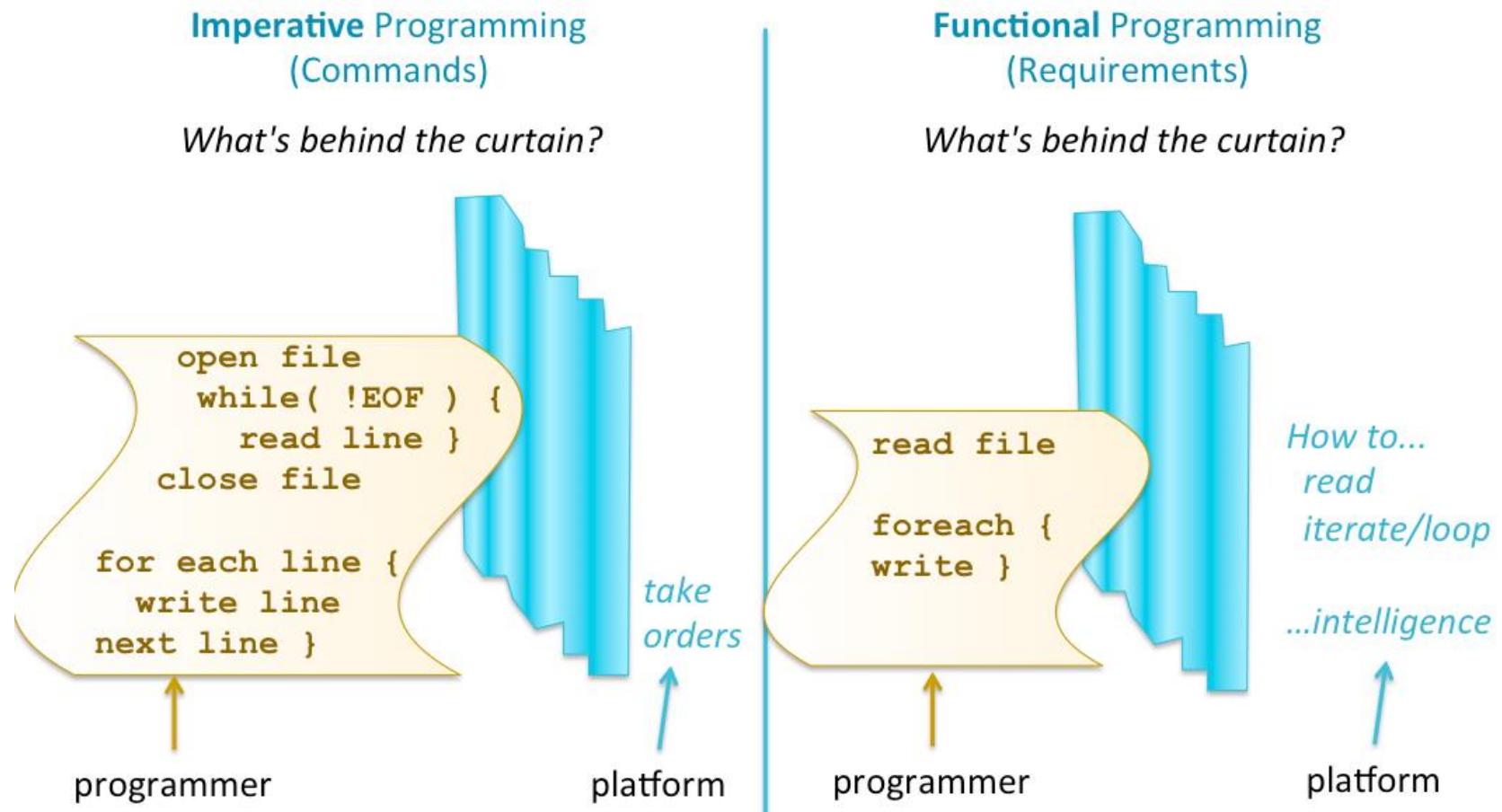
- All Java types are available
  - But in most cases the developers will use Scala types analogs
- Scala is expensive



# Scala concepts

- Code blocks are expressions - all code blocks always return a value even if there is nothing to return
- Functions can be passed as parameters to other functions
  - `function1(function2)`

# Functional programming rules!

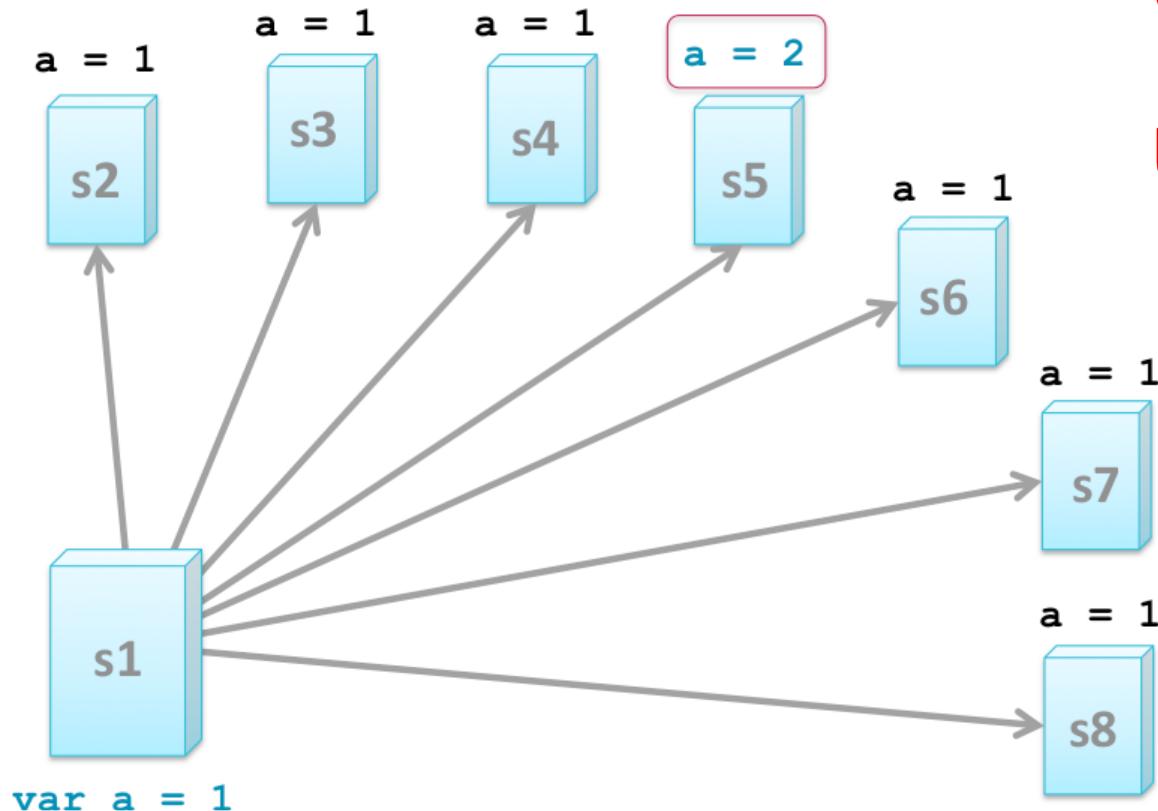


# Immutable vs Mutable

- When you declare variables:
  - val – immutable
  - var – mutable
- Collections
  - `scala.collections.immutable`
  - `scala.collections.mutable`

# Why immutable is good?

- Because mutable is bad



Who is responsible to update the others?

# Code example

```
import scala.io.Source
object Main {
  def main(args: Array[String]) {
    Source.fromFile("filename").foreach(println)
  }
}
```

# Scala variables

Syntax: [ **var** | **val** ] **name**: **type** = **value**

**val** *name*: **String** = "Je~~ka~~ka"

**val** *name2* = "Sasha"

**var** *age*: **Int** = 37

**var** *age2* = 28;

**var** *age*: **Int** = 37

**var** *age2* = 28

*age* = "sd"

# This is not javascript

```
var age: Int = 37  
var age2 = 28  
age = "sd"
```

```
val name: String = "Jeka"  
! val name2 = "Sasha"  
name = "Evgeny"  
! Convert 'val' to 'var'
```

# Scala types – the important ones

Type	Description	Example
<b>Int</b>	4 byte integer	3
<b>Long</b>	8 byte integer	32754 <b>L</b>
<b>Double</b>	8 byte floating point	3.1415
<b>Float</b>	4 byte floating point	3.1415 <b>F</b>
<b>Char</b>	single character	' c ' (single quotes)
<b>String</b>	sequence of characters	"iFruit" (double quotes)
<b>Boolean</b>	true or false	<b>true</b> (case sensitive)

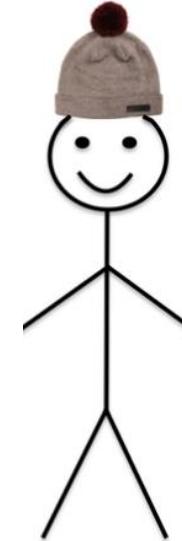
# Special type - Unit

- When function returns nothing it returns the Unit

```
val myreturn = println("Hello, world")
> myreturn: Unit = ()
```

*Just for information*

**This is Unit.**  
**There is only one Unit in Scala**  
**Unit can't be created**  
**Unit – is built in singleton**  
**Be like Unit**



# Special type - Any

```
val myreturn = if (true) "hi"  
> res2: Any = hi
```

```
val mystring = myreturn.asInstanceOf[String]  
> myreturn: String = hi
```

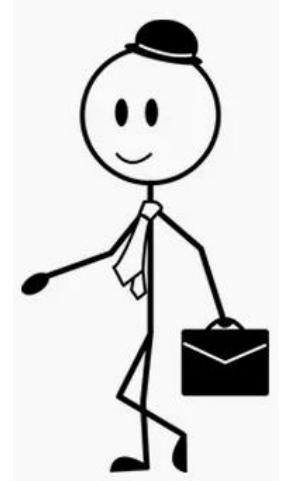
*Just for information*

**This is Any.**

**Any – like Object, but wider**

**Any – solves problem with generic  
collections and primitives**

**Be like Any**



# Scala is strange

- Scala loves (this braces) – because the people who invented Scala loved mathematic

```
val list = List("java", "scala", "groovy")
list.foreach(println)
```

- In case method doesn't take arguments you can omit the braces

```
print(list.getClass)
```

# No one is primitive

```
val x = 12
```

```
val y = x.toDouble
```

```
val isTrue = x.equals(x)
```

```
val z = x.*(Math.sqrt(4))
```

# Strings

```
var str = "Scala"
val c: Char = str(0)    // c = S
val pair: (String, String) = str.splitAt(2)
val sc: String = pair._1
val ala: String = pair._2
str = str.sorted

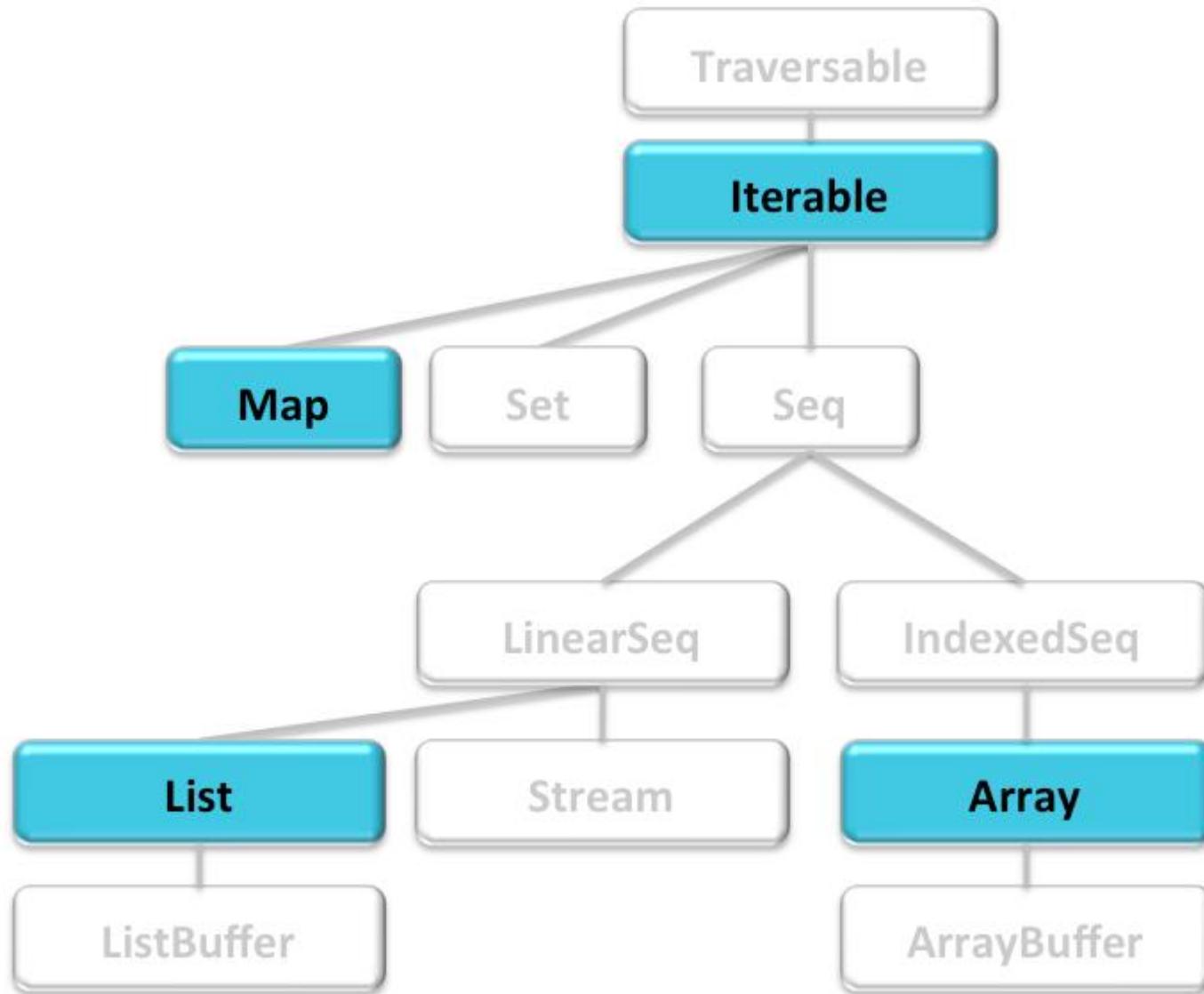
val endsWith: Boolean = str.sorted.toUpperCase().endsWith("la")

println(s"sum ${age+age2}")
```

# Collections: Mutable & Immutable

- By default Scala uses immutable collections
- So their methods returns new collection back
- If you will use `val` when you will declare immutable collection several methods will disappear

# Collections



- Traversable – abstraction, main method: foreach
- Iterable – adds possibility to iterate.
- Seq – adds possibility to apply by index

```
val seq: Seq[Int] = Seq(2, 4, 6, 8)  
val four: Int = seq(2)
```

# What does this code?

```
val set: Set[Int] = Set(2, 4, 6, 8)  
set +=1  
val b: Boolean = set(3)  
print(b)
```

Compilation error

# And now?

```
var set: Set[Int] = Set(2, 4, 6, 8)
set = set.drop(2)
set += 1
val b: Boolean = set(3)
print(b)
```

False

# Map

```
var nameToAge: Map[String, Int] = Map("Jeka" -> 37, "Sasha" -> 30)
nameToAge += ("Mikola" -> 22)
```

# List

- `scala.collection.immutable.List`
- `scala.collection.mutable.List`
- But we gonna talk about Immutable

# List

**Empty List**



```
val emptyList: List[Int] = Nil
```

```
val list = 1::2::emptyList
```

```
val list: List[Int] = List.apply(1,2,3)
```

```
val list: List[Int] = List(1,2,3)
```

You can use index as well:      `println(list(1))`

# Generic is good, but will it work?

```
val list: List[Object] = List("abc", 1)      Will it work?
```

# Generic is good, but will it work?

```
val list: List[Object] = List("abc", 1)
```

**Compilation error** ☹

You can work without generic, but...

```
val list = List("abc", 1)
```

Any – everything including primitive

```
val list: List[Any] = List("abc", 1)
```

AnyRef – also exists

# List methods

- sum – in case this is list of numbers
- max – for comparable
- take( $x$ ) – returns list with  $x$  elements
- sorted
- reverse
- union
- intersect

# Array – mutable, but not resizable

```
val strings: Array[String] = Array("one", "two")
strings(0) = strings(0).toUpperCase
strings.update(1, "Два")
strings.foreach(println)
```

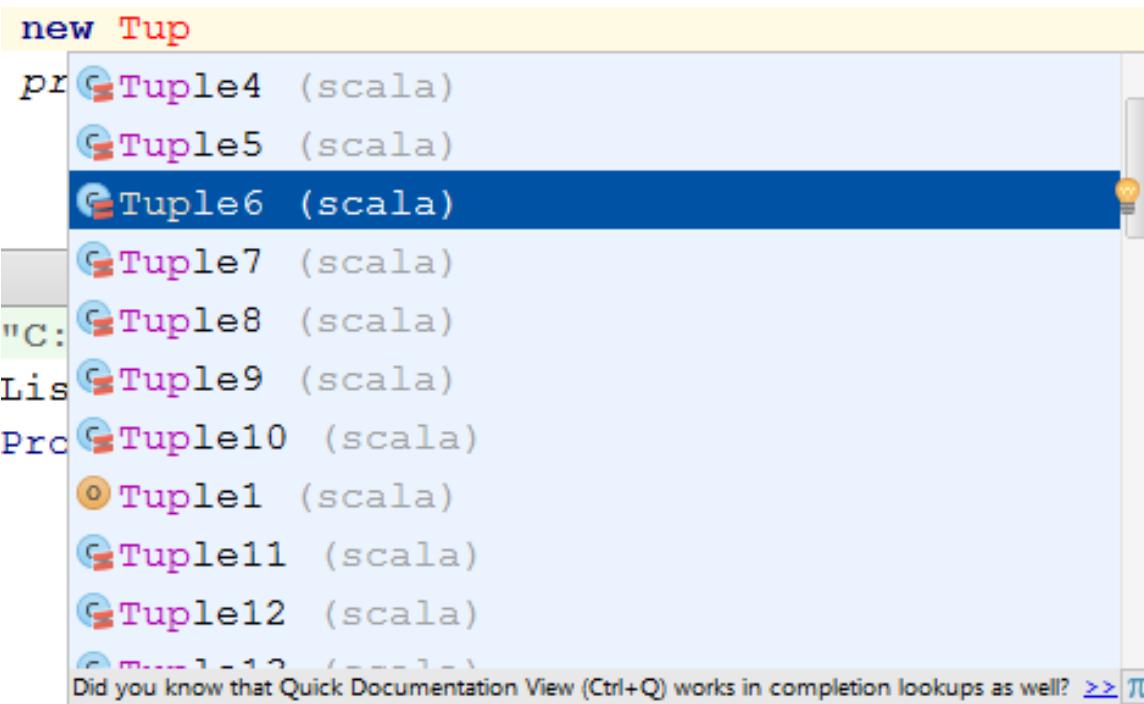
ONE

Два

# Pair Tuples

```
val tuple1: (Int, String) = (1, "java")           tuple1 = 1 -> "java"  
  
val tuple2: (Int, String) = (40, "scala")  
val tuple3: (Int, String) = (20, "groovy")  
  
val tuple: (LocalDateTime, String, Object) = (LocalDateTime.now(), "beanName", bean)  
  
var tuples: List[(LocalDateTime, String, Object)] = List(tuple)  
tuples = (LocalDateTime.now(), "beanName", bean) :: tuples
```

# Tuple – till 22



# Get from Tuple

```
val dateTme: LocalDateTme = tuple._1  
val beanName: String = tuple._2  
val bean: Object = tuple._3
```

```
tuples.foreach(tupple=>println(tuple._1))
```

# What else tuple can do?

- swap

```
var tuple1: (Int, String) = (1, "java")
val swap: (String, Int) = tuple1.swap
```

- Only for Tuple2

# You will never guess

```
val tuple: (LocalDateTime, String, Object) = (LocalDateTime.now(), "beanName", bean)
val prefix: String = tuple.productPrefix //prefix = Tuple3
val arity: Int = tuple.productArity // arity = 3
```

# List zip, unzip

```
val words: List[String] = List("one", "two", "three")
val ints: List[Int] = List(1, 2)
val zip: List[(String, Int)] = words.zip(ints)
zip.foreach(println)
val unzip: (List[String], List[Int]) = zip.unzip
unzip._1.foreach(println)
```

(one,1)

(two,2)

one

two

# Map – are constructed of Tuple2

```
var intToString1: Map[Int, String] = Map(1 -> "one", 2 -> "two")
var intToString2: Map[Int, String] = Map((3, "tree"), (4, "four"))
println(intToString1(1))
```

- keys, values, contains...

intToString2(3) = "Три"

Compilation error ☹

**scala.collection.mutable.Map**

# What will happen?

```
var intToString: Map[Int, String] = Map(1 -> "one", 2 -> "two")
println(intToString(13))
```

- A. null
- B. Nothing will be printed out
-  NoSuchElementException
- D. NPE

```
intToString.getOrElse(13, "default value")
```

# Common conversions

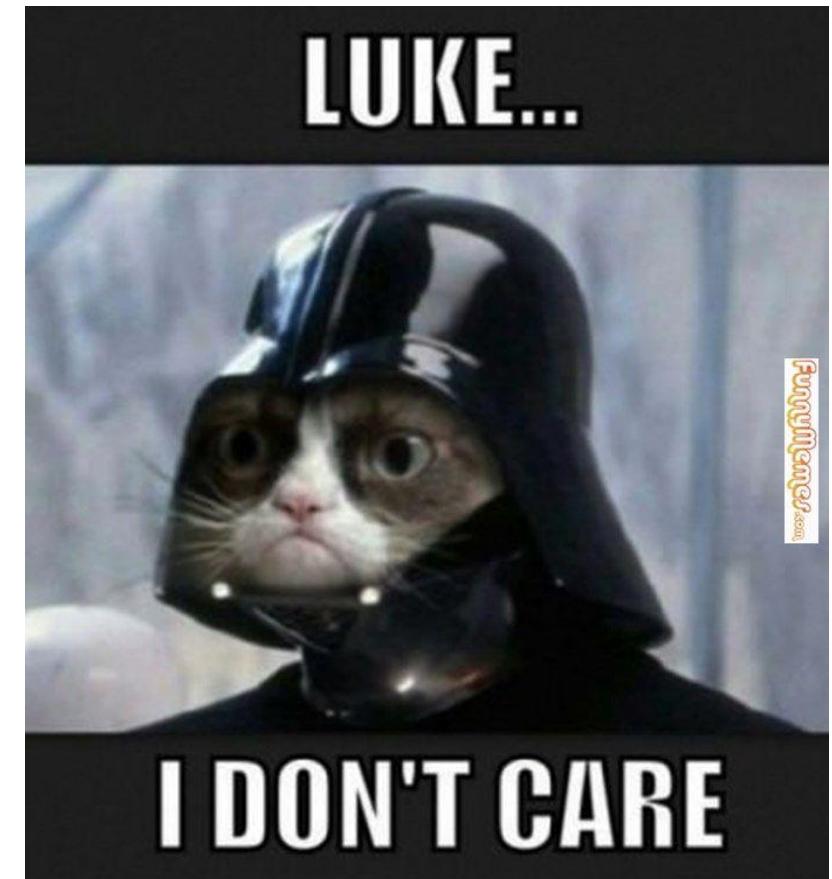
- `toArray`
- `toList`
- `tolterable`
- `String`: `toSet`, `toList`, `toArray`

# We don't care about braces

```
def UA2USD(ua: Double) = {ua * 25}
```

```
def UA2USD(ua: Double) = (ua * 25)
```

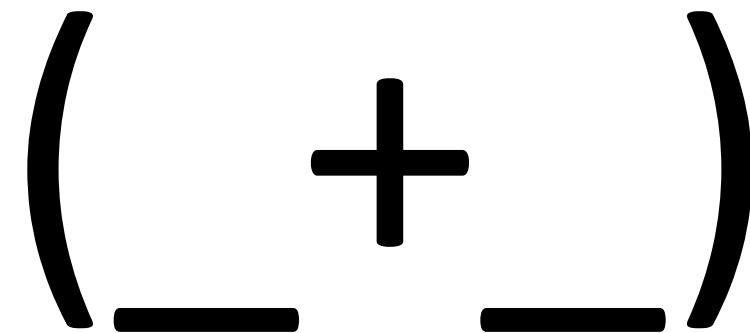
```
def UA2USD(ua: Double) = ua * 25
```



# The best Scala Spark operator



# The best Scala Spark operator



# First step to the ass (\_+\_)

```
val strings: List[String] = List("one", "two", "three")
strings.sortBy(_.length)
```

```
val sorted: List[String] = strings.sortWith(_>_)
```

# Map function – second step to the ass

```
val strings: List[String] = List("one", "two", "three")
val stringLowerCased: List[String] = strings.map(_.toLowerCase)
val ints: List[Int] = strings.map(_.length).map(_+4)
```

# Filter function

```
phoneCelsius.filter(_ < 23)
> List[Double] = List(12.2)
```

```
TitanicPhones.filter(_.startsWith("2"))
> List[String] = List(2000, 2100, 2200, 2300, 2400, 2500)
```

```
TitanicPhones.filter(_.length > 4 )
> List[String] = List(DeckChairs
```

# Pattern Matching

```
while (true) {  
    val message: String = JOptionPane.showInputDialog(null, "what would you say")  
    var answer: String = ""  
    message match {  
        case "hello" => answer = "hello";  
        case "how are you" => answer = "better than you";  
        case "hi" => answer = "hi!";  
    }  
    println(answer)  
}
```

What if man asks: “what is your name?”

# IF MAN ASKS A GIRL WHAT IS HER NAME



**IF MAN ASKS A GIRL WHAT IS HER NAME**

A woman with short brown hair, wearing a dark green t-shirt, looks up at a man in a grey toga. They are in a large, dimly lit hall with many columns and lit candles on the floor. The man is walking towards the woman.

**THE GIRL WILL THROW AN EXCEPTION**

```
while (true) {  
    val message: String = JOptionPane.showInputDialog(null, "what would you say")  
    var answer: String = ""  
    message match {  
        case "hello" => answer = "hello";  
        case "how are you" => answer = "better than you";  
        case "hi" => answer = "hi!";  
        case default => answer = "default answer!";  
    }  
    println(answer)  
}
```

# Classes

- Like in Java (almost)
- Singletons



# Regular classes

```
class Person(name: String) {  
    def printDetails(): Unit = {  
        println(name)  
    }  
}
```

# Overriding methods

```
class Person(name: String) {  
    def printDetails(): Unit = {  
        println(name)  
    }  
  
    override def toString: String = name  
}
```

# Overriding methods

```
class Person(name: String) {  
    def printDetails(): Unit = {  
        println(name)  
    }  
  
    override def toString: String = {  
        "my name is "+name  
    }  
}
```

# Scala Singleton – generated by scala

```
object TestDevice {  
    def main(args: Array[String]) {  
        val a = new Device("iFruit 3000")  
        println(a.display)  
    }  
}
```

```
$ scalac TestDevice.scala
```



Class: **TestDevice.class**  
Companion Object: **TestDevice\$.class**

# Imports

```
import pack1._
```

- Imports all classes from **pack1**
- Equivalent to Java **import \*.pack1**

```
import pack1._, pack2._
```

- Imports all classes from **pack1** and from **pack2**

```
import pack1.Class1
```

- Imports only **Class1** from **pack1**

```
import pack1.(Class1, Class3)
```

- Imports only **Class1** and **Class3** from **pack1**

# More imports

```
import pack1.Class1._
```

- Imports all members of **Class1** *including implicit definitions*
- No equivalent in Java

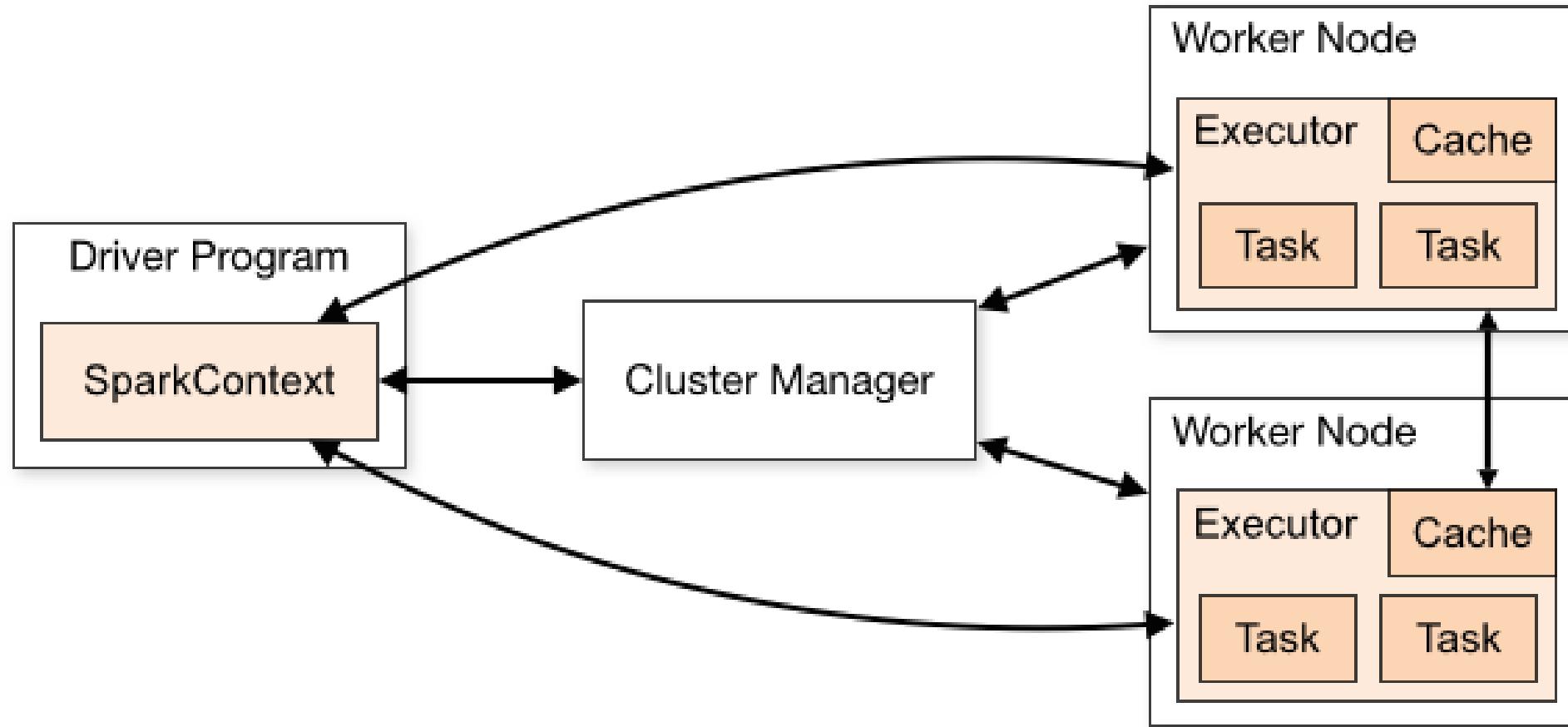
```
import pack1.{Class1 => MyClass}
```

- Imports **Class1** from pack1 and renames it **MyClass**
- This avoids collision with an existing function named **Class1**

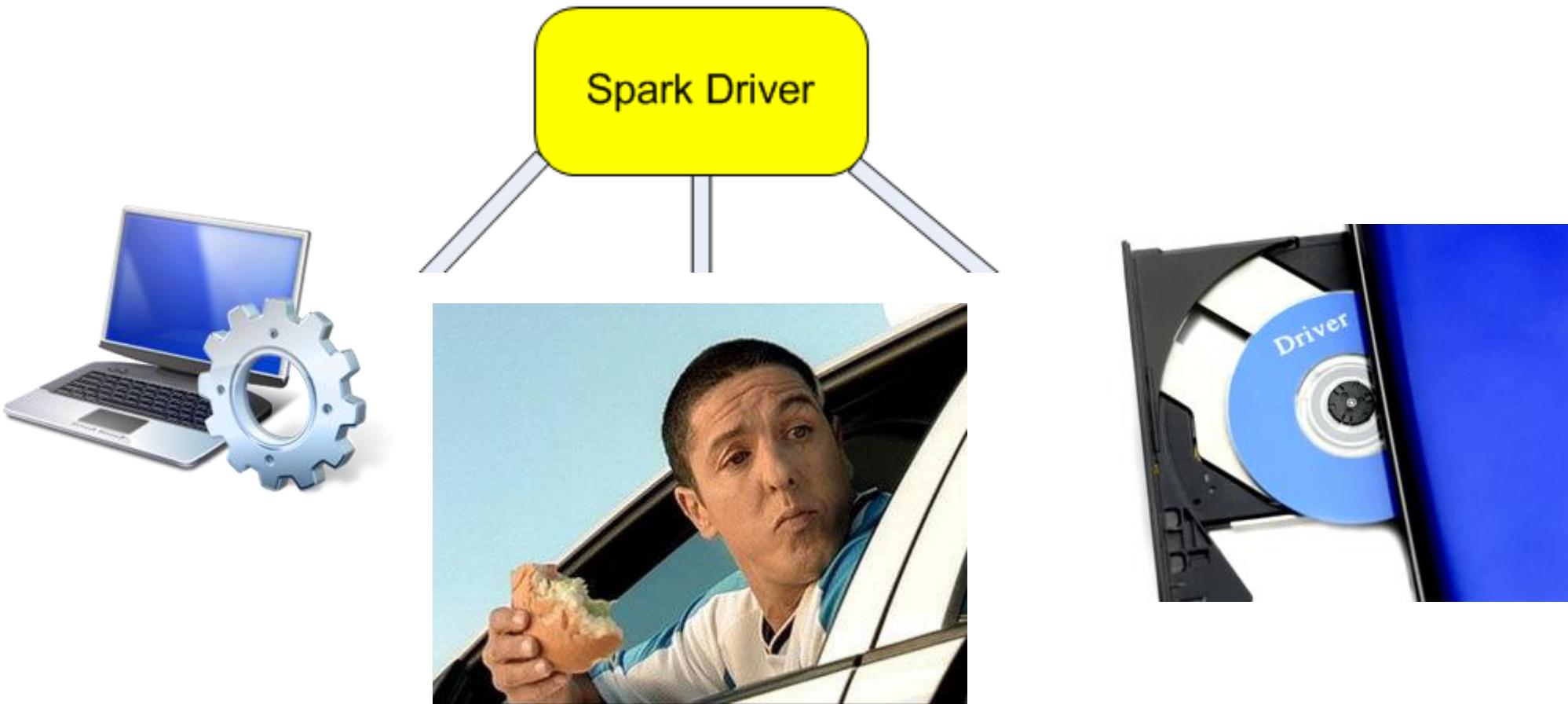
# Spark

- Release
  - Was born in UC Berkeley in about 2009
  - First releases in 2012
  - Current version 2.4 several weeks ago
  - But still the most popular version is 1.6-2
- Written in scala
  - Scala is functional language that runs on JVM
- Spark api:
  - Scala, Python, Java, Java 8, R
- Writing spark
  - IntelliJ, Eclipse, Spark-shell, Notebooks
- Running spark
  - Spark-Shell, Notebooks, Spark-submit
  - java –jar
- Running on cluster
  - You need cluster manager

# How does it work on cluster



# Driver, driver, what driver?



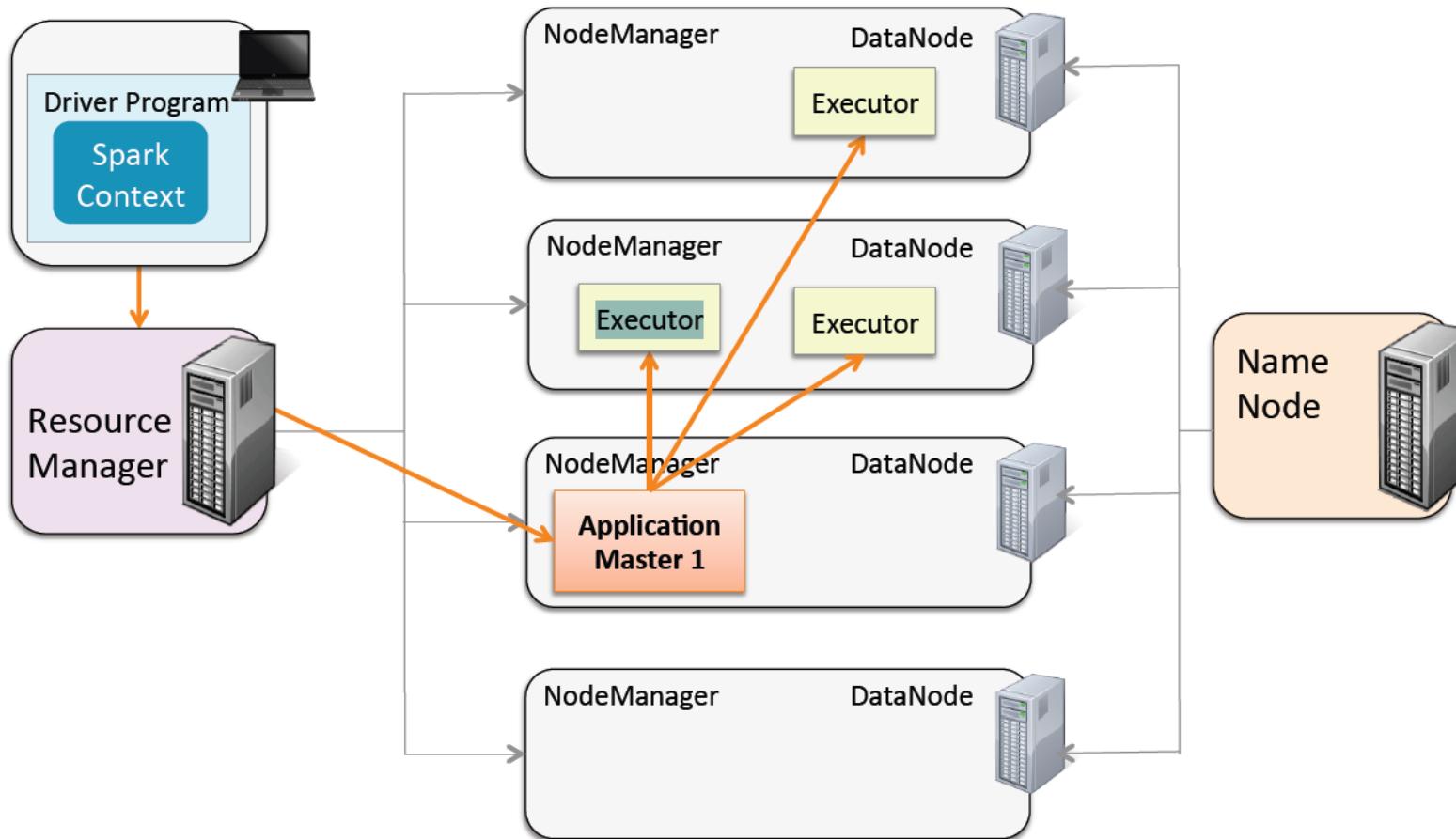
# What is a driver?

- Driver program - The process running the main() function of the application and creating the SparkContext
- Application which applies to Spark
- Like JDBC driver

# Spark with yarn

**Extremely important!**

Part of the code executed on driver  
Part of the code executed on cluster



# RDD

Resilient



Distributed



Dataset



+**Stream**

# How can you get RDD?

- From file or set of files (directory)
- From data in memory
- From another RDD

# File / directory

```
// from local file system
JavaRDD<String> rdd = sc.textFile("file:/home/data/data.txt");

// from Hadoop using relative path of user, who run spark
application
rdd = sc.textFile("/data/data.txt")

// from hadoop
rdd = sc.textFile("hdfs://data/data.txt")

// all files from directory
rdd = sc.textFile("s3://data/*")

// all txt files from directory
rdd = sc.textFile("s3://data/*.txt")
```

# Memory

sc.pa|

- m **parallelize**(List<T> list) JavaRDD<T>
- m **parallelize**(List<T> list, int numSlices) JavaRDD<T>
- m **parallelizeDoubles**(List<Double> list) JavaDoubleRDD
- m **parallelizeDoubles**(List<Double> list, int numSl... JavaDoubleRDD
- m **parallelizePairs**(List<Tuple2<K, V>> list) JavaPairRDD<K, V>
- m **parallelizePairs**(List<Tuple2<K, V>> list, i... JavaPairRDD<K, V>

# What is sc?

- SparkContext / JavaSparkContext
- Main entry point for Spark functionality

```
SparkConf conf = new SparkConf();
conf.setAppName("my spark application");
conf.setMaster("local[*]");
JavaSparkContext sc = new JavaSparkContext(conf);
```

# RDD has two kind of methods (like streams)

- Transformation (intermediate) – always runs on cluster
- Action (terminal.
  - Can return value back to the driver
  - Can store RDD in HDFS

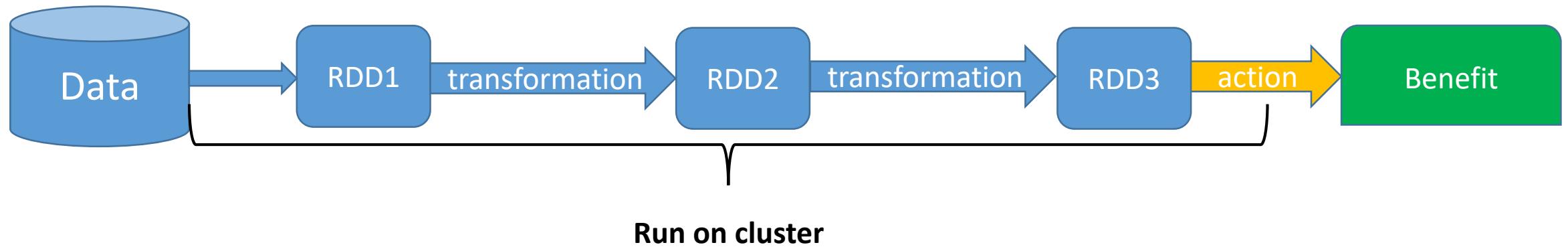
# Transformations

- **Map / mapToPair / mapToDouble**
- **flatMap**
- **filter**
- **mapPartitions, mapPartitionsWithIndex**
- **sample**
- **union, intersection, join, cogroup, cartesian (*otherDataset*)**
- **distinct**
- **reduceByKey, aggregateByKey, sortByKey**
- **pipe**
- **coalesce, repartition, repartitionAndSortWithinPartitions**

# Actions

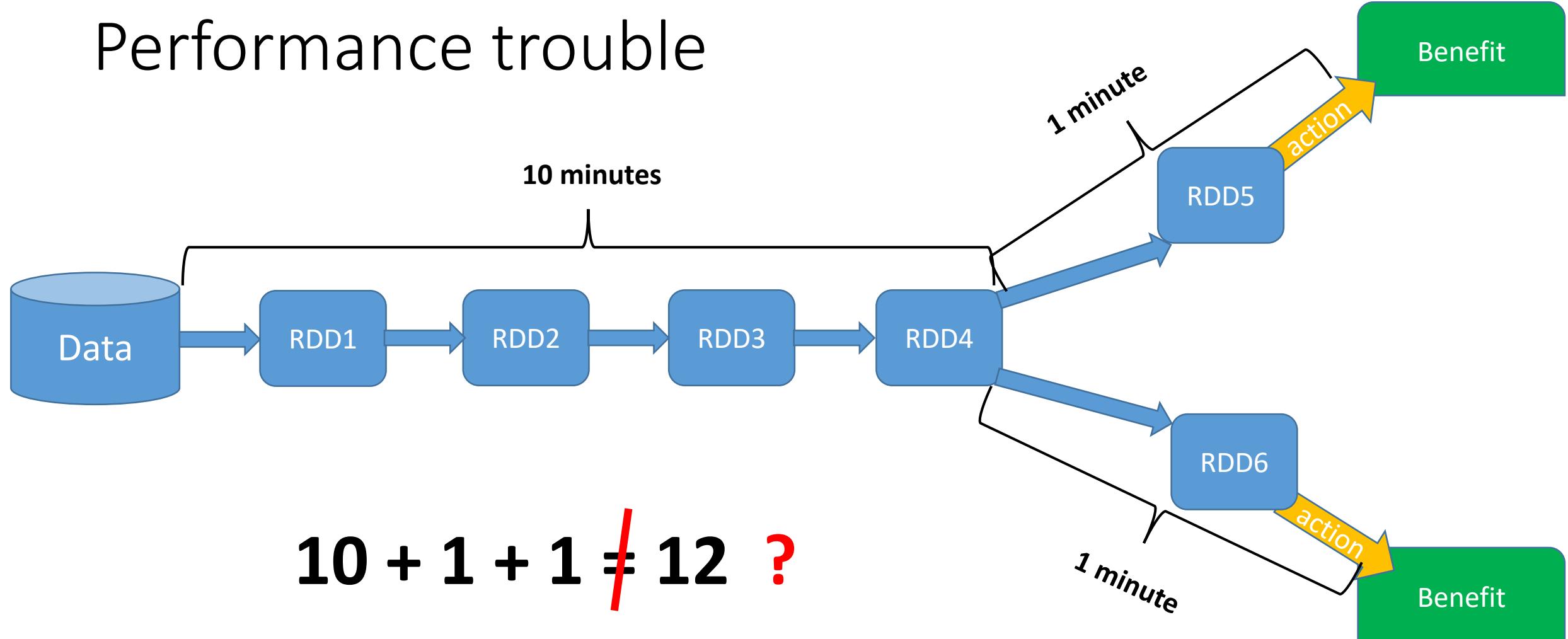
- **reduce**
- **collect**
- **count, countByKey, countByValue**
- **first**
- **take, takeSample, takeOrdered**
- **saveAsTextFile, saveAsSequenceFile, saveAsObjectFile**
- **foreach**

# Rdd flow

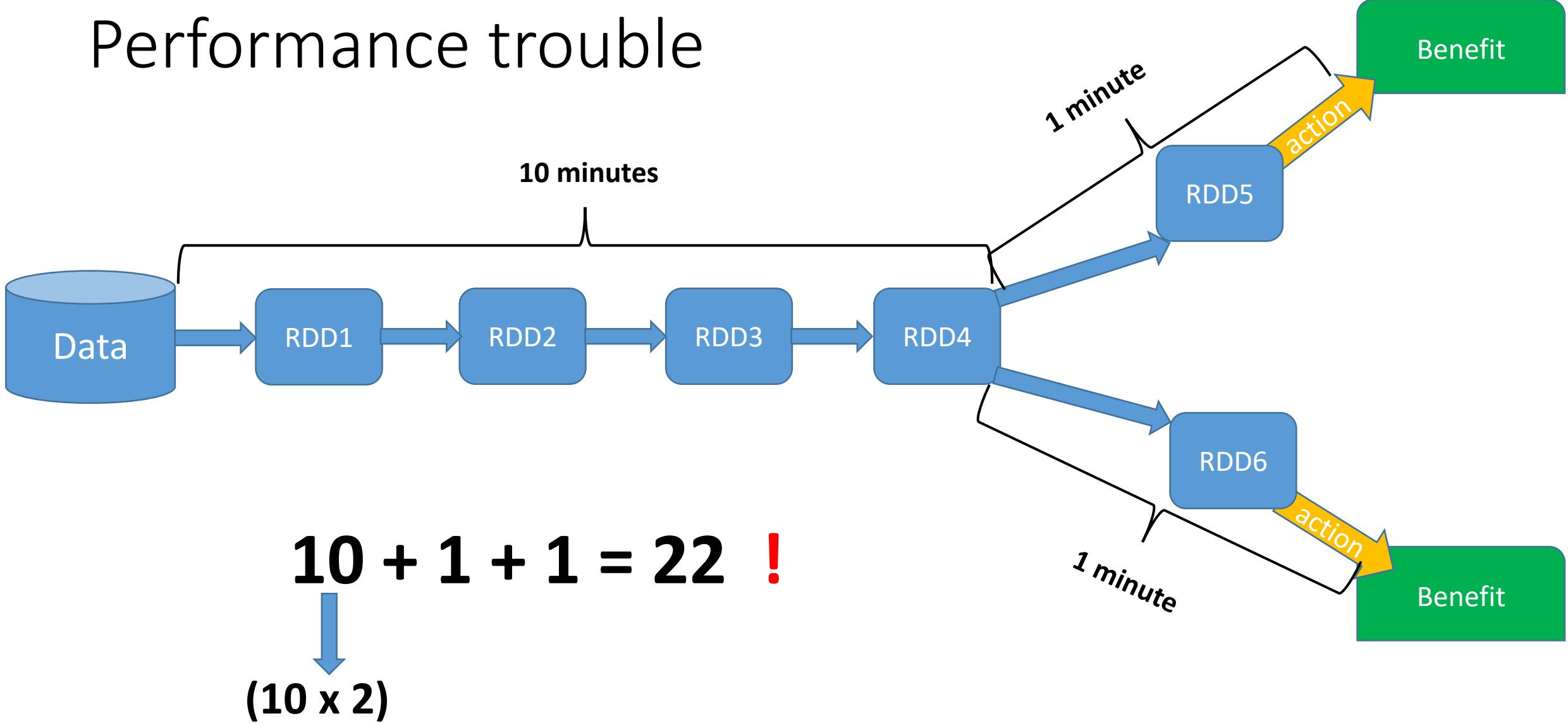


Lazy – like stream api of Java 8

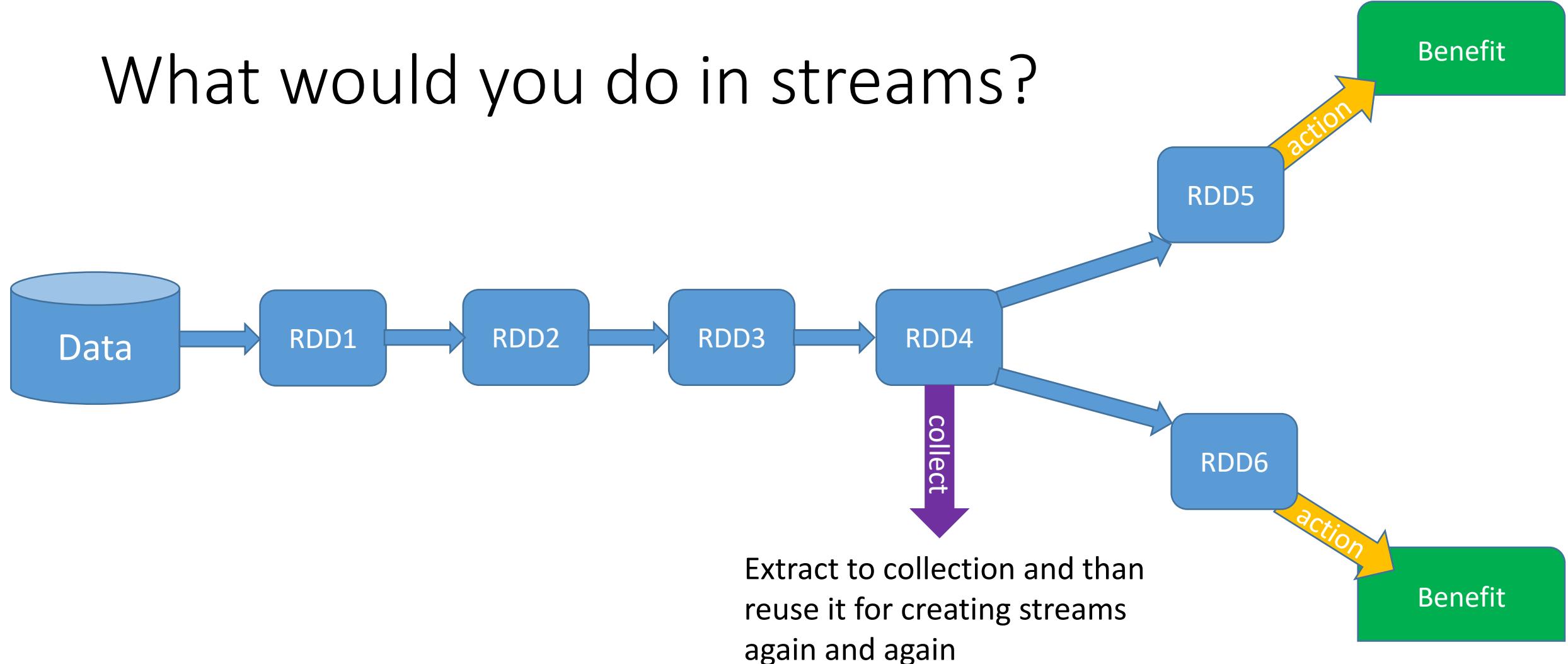
# Performance trouble



# Performance trouble



# What would you do in streams?





tried this way

Out of memory



A close-up of Aragorn's face from The Lord of the Rings. He has long brown hair and a beard, looking slightly to the side with a serious expression. He is wearing a dark leather-like armor.

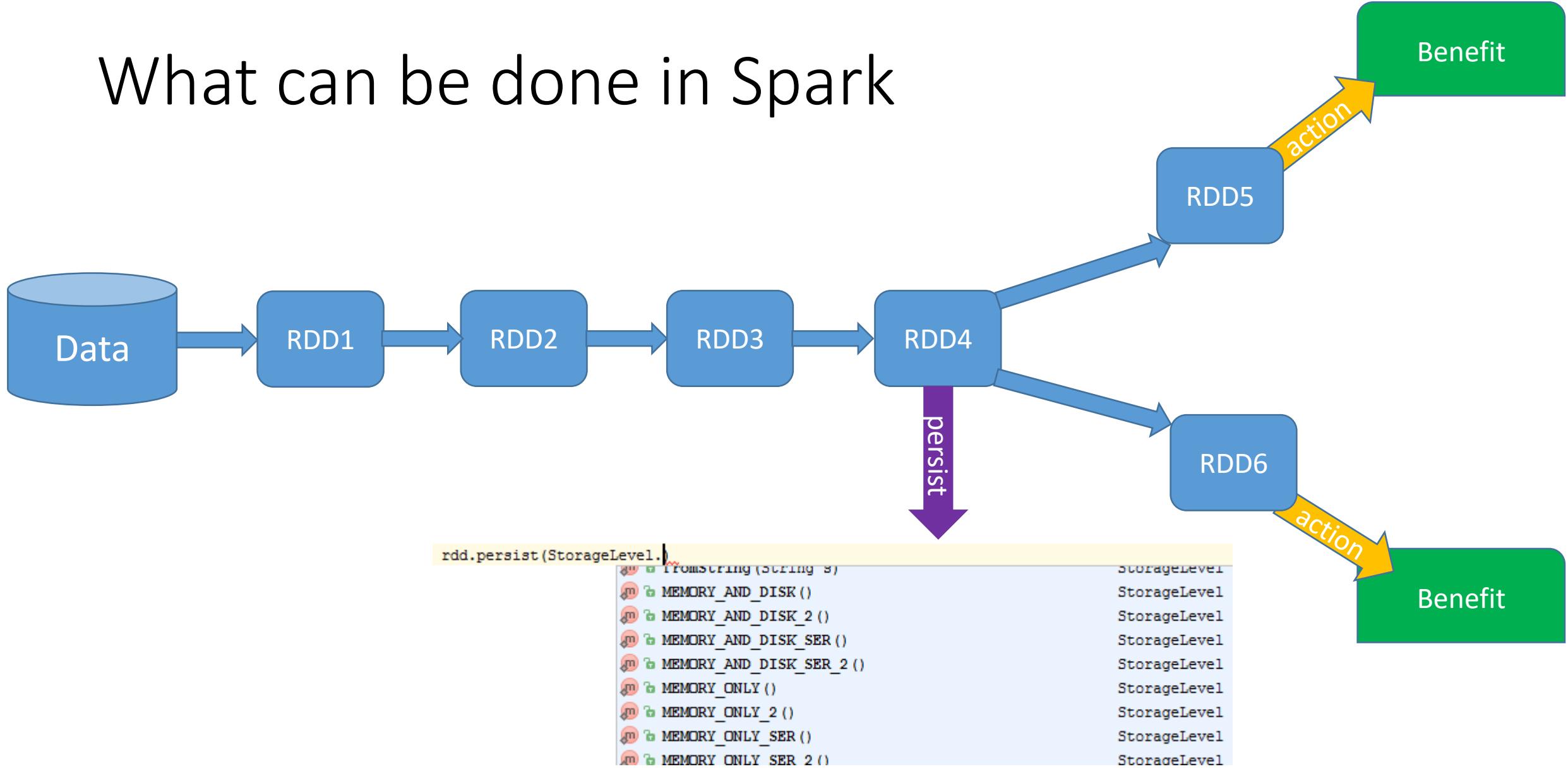
ONE DOES NOT SIMPLY

CAN WRITE EFFECTIVE SPARK CODE

You need to know when use

Persist

# What can be done in Spark



# Let's start writing spark

- Scala
- Java
- ~~Python~~
- ~~R~~

# Scala for regular Java developer

## What is Scala

- ➊ 纯面向对象, class\trait\mixin
- ➋ 函数式first class,lambda,closure,curry,lazy,tail recursive opt
- ➌ Actor, pattern-match
- ➍ Jvm bytecode(1.5 compatible)
- ➎ 强类型,静态语言,



Martin Odersky

advanced

# Scala for ~~regular~~ Java developer

## What is Scala

- ➊ 纯面向对象, class\trait\mixin
- ➋ 函数式first class,lambda,closure,curry,lazy,tail recursive opt
- ➌ Actor, pattern-match
- ➍ Jvm bytecode(1.5 compatible)
- ➎ 强类型,静态语言,



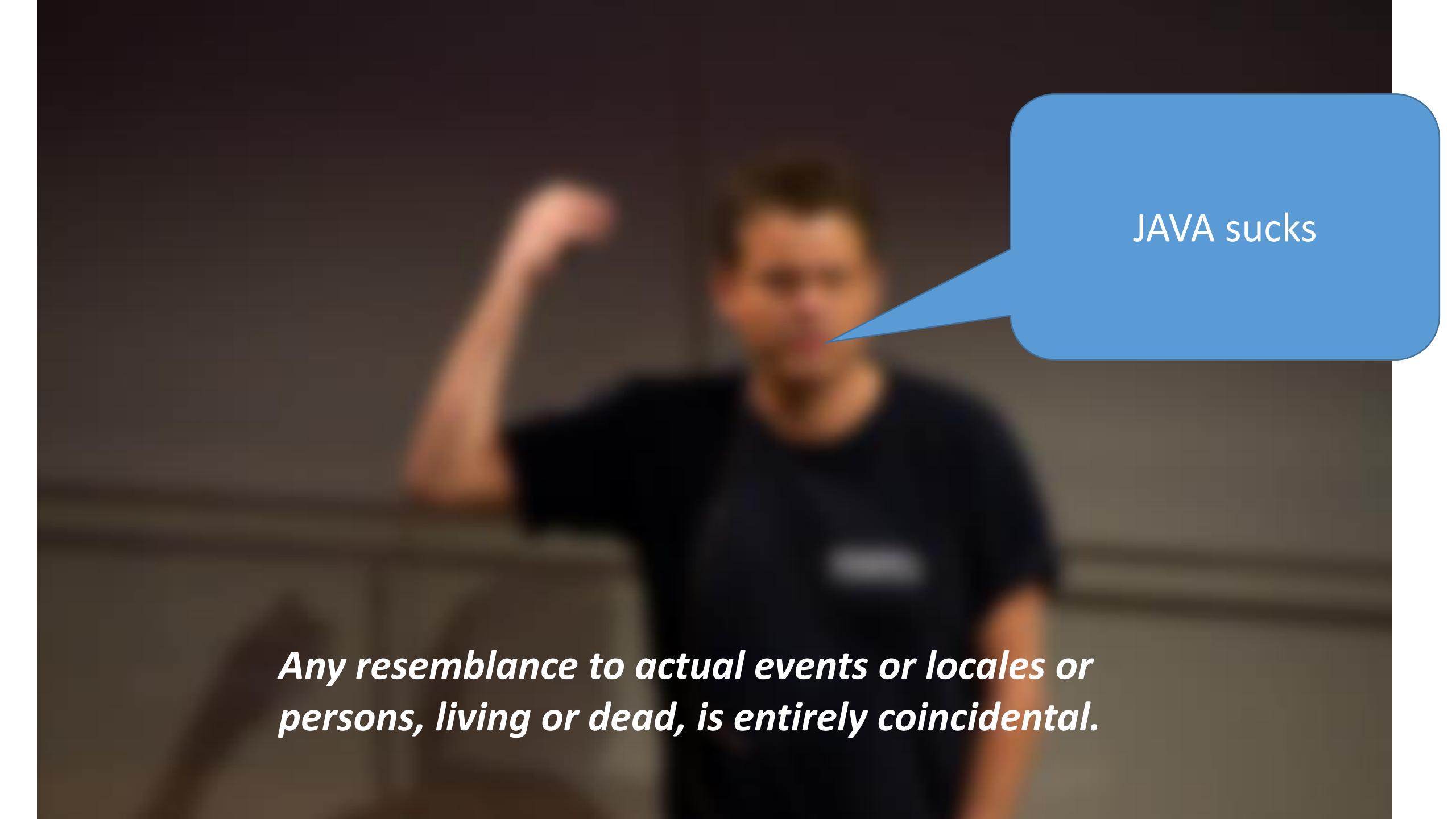
Martin Odersky

```
val lines = sc.textFile("data.txt")
val lineLengths = lines.map(_.length)
val totalLength = lineLengths.reduce(_+_)
```

Scala

Java

```
JavaRDD<String> lines = sc.textFile("data.txt");
JavaRDD<Integer> lineLengths = lines.map(new Function<String, Integer>() {
    @Override
    public Integer call(String lines) throws Exception {
        return lines.length();
    }
});
Integer totalLength = lineLengths.reduce(new Function2<Integer, Integer, Integer>() {
    @Override
    public Integer call(Integer a, Integer b) throws Exception {
        return a + b;
    }
});
```

A blurry, out-of-focus photograph of a man in a dark suit and tie. He is pointing his right index finger directly at the viewer. The background is dark and indistinct.

JAVA sucks

*Any resemblance to actual events or locales or persons, living or dead, is entirely coincidental.*

```
val lines = sc.textFile("data.txt")
val lineLengths = lines.map(_.length)
val totalLength = lineLengths.reduce(_+_)
```

**Scala**

```
JavaRDD<String> lines = sc.textFile("data.txt");
JavaRDD<Integer> lineLengths = lines.map(String::length);
int totalLength = lineLengths.reduce((a, b) -> a + b);
```

**Java 8**

# Second myth: you should write spark on scala

Pro Scala	pro Java
Scala is cool	

# Second myth: you should write spark on scala

Pro Scala	pro Java
Scala is cool	Most java developers familiar with java

# Second myth: you should write spark on scala

Pro Scala	pro Java
Scala is cool	Most java developers familiar with java
short syntax	

# Second myth: you should write spark on scala

Pro Scala	pro Java
Scala is cool	Most java developers familiar with java
short syntax	You can use the same technicians IOC, Design patterns

# Second myth: you should write spark on scala

Pro Scala	pro Java
Scala is cool	Most java developers familiar with java
short syntax	You can use the same technicians IOC, Design patterns
Spark API more adopted to scala	

# Second myth: you should write spark on scala

Pro Scala	pro Java
Scala is cool	<b>Most java developers familiar with java</b>
short syntax	You can use the same technicians IOC, Design patterns
Spark API more adopted to scala	

# Second myth: you should write spark on scala

Pro Scala	pro Java
Scala is cool	<b>Most java developers familiar with java</b>
short syntax	You can use the same technicians IOC, Design patterns
Spark API more adopted to scala	
Java API not always have all features, and released later	

# Second myth: you should write spark on scala

Pro Scala	pro Java
Scala is cool	<b>Most java developers familiar with java</b>
short syntax	You can use the same technicians IOC, Design patterns
Spark API more adopted to scala	<b>Most java developers familiar with java</b>
Java API not always have all features, and released later	You can use the same technicians IOC, Design patterns

# So why Java?



Lab - I'll send you file like this one

File content:

104 Stone Mountain 9 Sat Feb 13 21:02:45 IST 2016  
113 Blairsville 12 Sun Feb 21 07:24:36 IST 2016  
114 Riverdale 10 Fri Feb 12 13:10:09 IST 2016  
117 Flemington 5 Sun Feb 14 07:24:58 IST 2016  
115 Boston 12 Wed Feb 10 12:39:00 IST 2016  
115 Boston 10 Wed Feb 10 12:35:00 IST 2016

# What should be done

- Count number of lines
- Count amount of trips to Boston longer than 10 km
- Calculate sum of all kilometers trips to Boston
- Write names of 3 drivers with max total kilometers in this day  
(sort top to down)

# Performance Lab

- You need to count trips less than 5 km,
- Between 5 and 10 km
- Greater than 10 km
- Which kind of trips are most popular?

A close-up of Aragorn's face from The Lord of the Rings. He has long brown hair and a beard, looking slightly to the side with a serious expression. He is wearing a dark leather-like armor.

ONE DOES NOT SIMPLY

CAN WRITE EFFECTIVE SPARK CODE

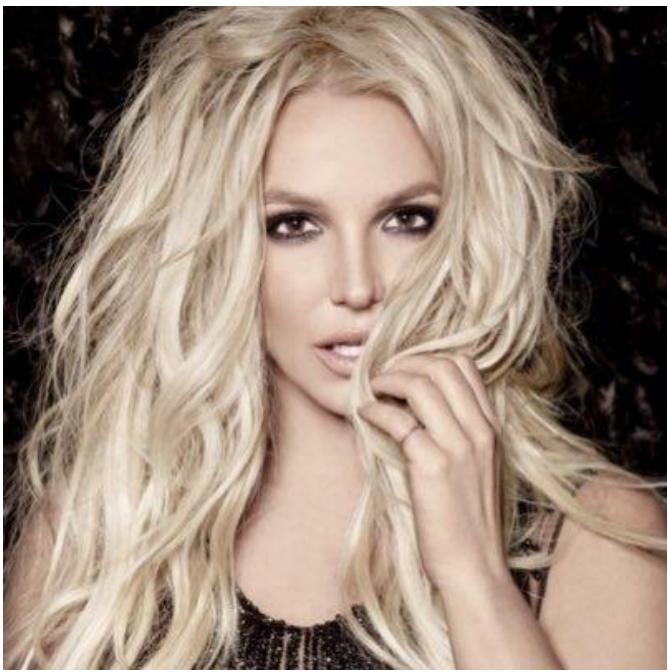
# Accumulator

- Accumulator will be delivered to all executors, and will accumulate some value, when accumulator will be grabbed to driver all values will be summarized.

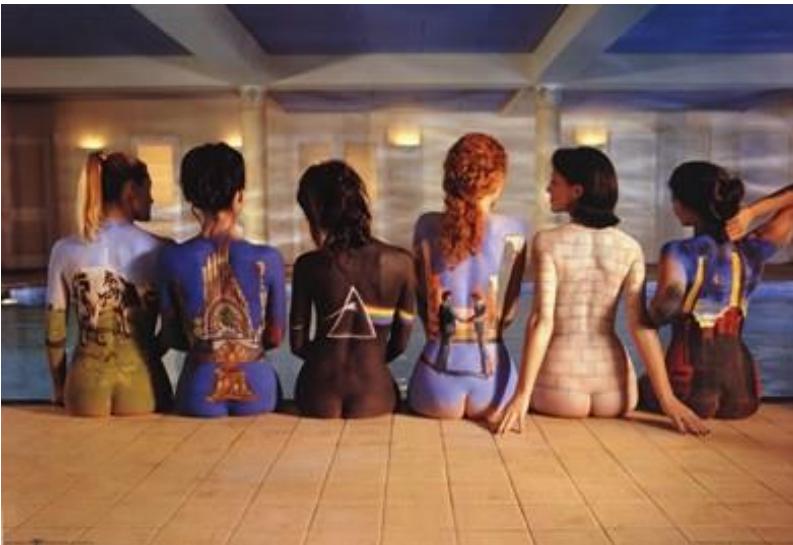
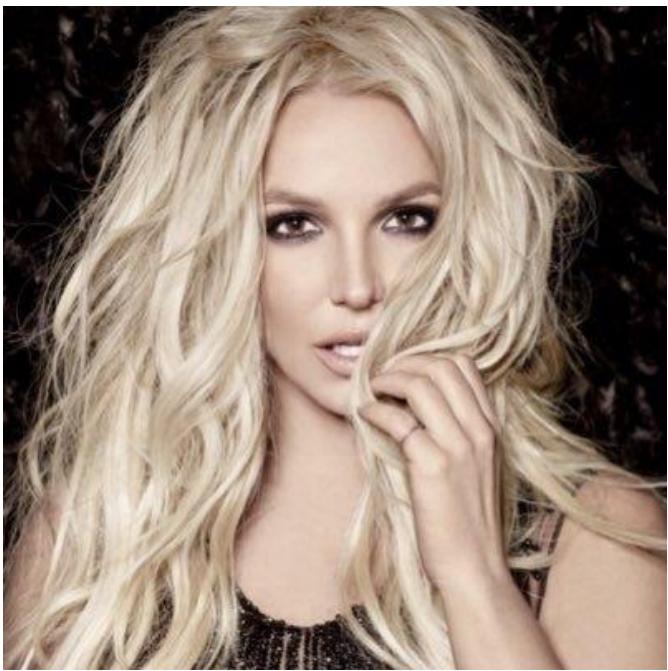
```
Accumulator<Integer> accumulator = sc.accumulator(0, "myAcc");
accumulator.add(1);
Integer sum = accumulator.value();
```

```
val accum = sc.accumulator(0, "myAcc")
accum += 1
val sum: Int = accum.value
```

# More interesting lab:



Pink Floyd is not like Britney or Ketty, or maybe yes...



# Another lab

- Write service which will receive a file and variable  $x(\text{int})$  and will find  $x$  most popular words
- `public List<String> topX(File path, int x)`
- your test should check, that word ‘yesterday’ is the most popular in song Yesterday of Beatles

You need to filter out all garbage words



# How to separate environments

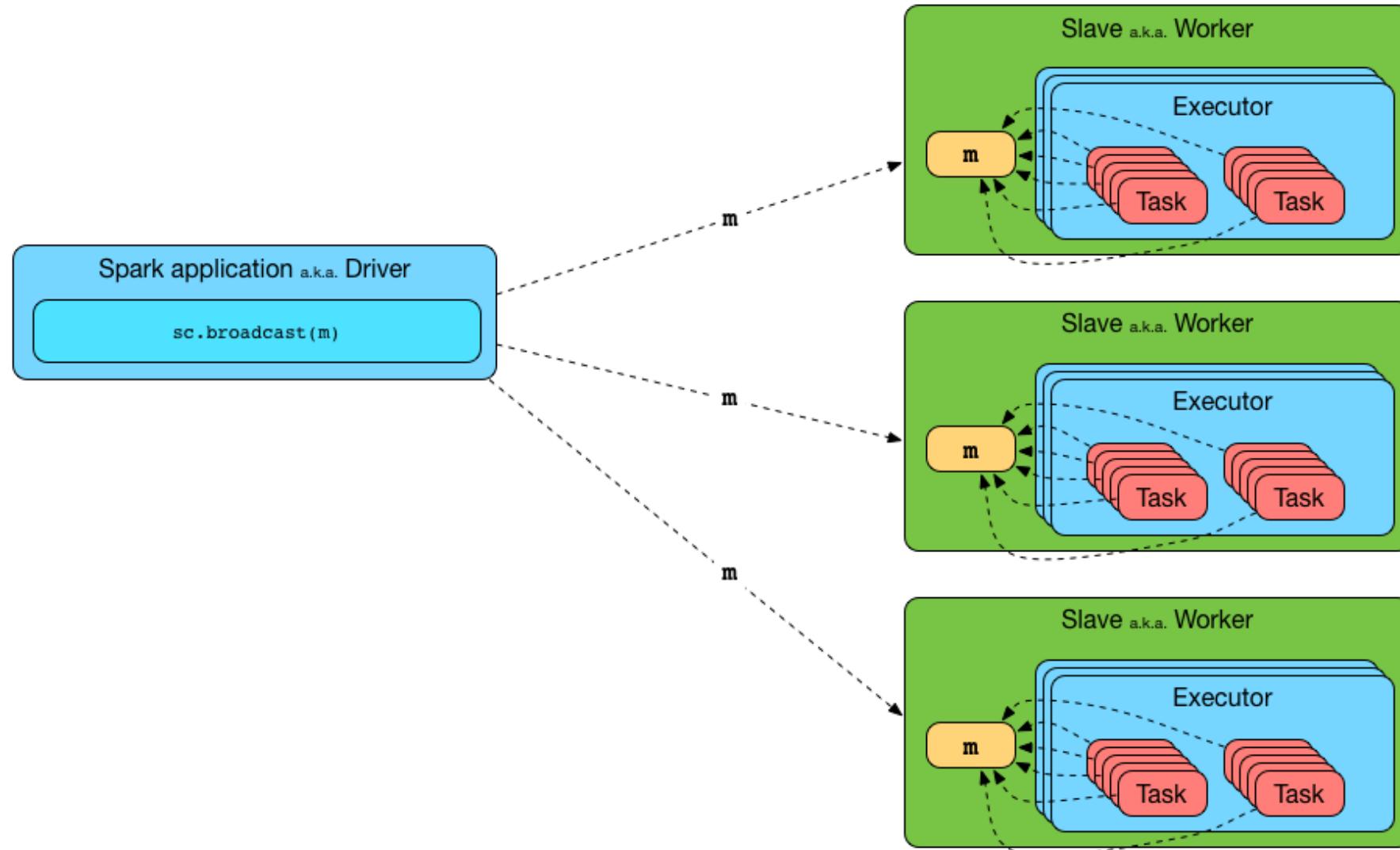
- Test
- Production

You need to know about

# Broadcast

```
Broadcast<List> listBroadcast = sc.broadcast(list);  
List myList = listBroadcast.value();
```

# Broadcast



# Example

Israel, +9725423632  
Israel, +9725454232  
Israel, +9721454232  
Israel, +9721454232  
Spain, +34441323432  
Spain, +34441323432  
Israel, +9725423232  
Israel, +9725423232  
Spain, +34441323432  
Russia, +78123343434  
Russia, +78123343434

# Example

Israel, +9725423632  
Israel, +9725454232  
Israel, +9721454232  
Israel, +9721454232  
~~Spain, +34441323432~~  
~~Spain, +34441323432~~  
Israel, +9725423232  
Israel, +9725423232  
~~Spain, +34441323432~~  
Russia, +78123343434  
Russia, +78123343434

# Example

Israel, Orange

Israel, Orange

Israel, Pelephone

Israel, Pelephone

Israel, Hot Mobile

Israel, Orange

Russia, Megaphone

Russia, MTC

# Example

Orange

Orange

Pelephone

Pelephone

Hot Mobile

Orange

Megaphone

MTC

# Example

```
Israel, +9725423632
Israel, +9725454232
Israel, +9721454232
Israel, +9721454232
Spain, +34441323432
Spain, +34441323432
Israel, +9725423232
Israel, +9725423232
Spain, +34441323432
Russia, +78123343434
Russia, +78123343434
```

```
public interface CommonConfig {
    Operator getOperator(String phone);

    List<String> countries();
}
```

# Not efficient

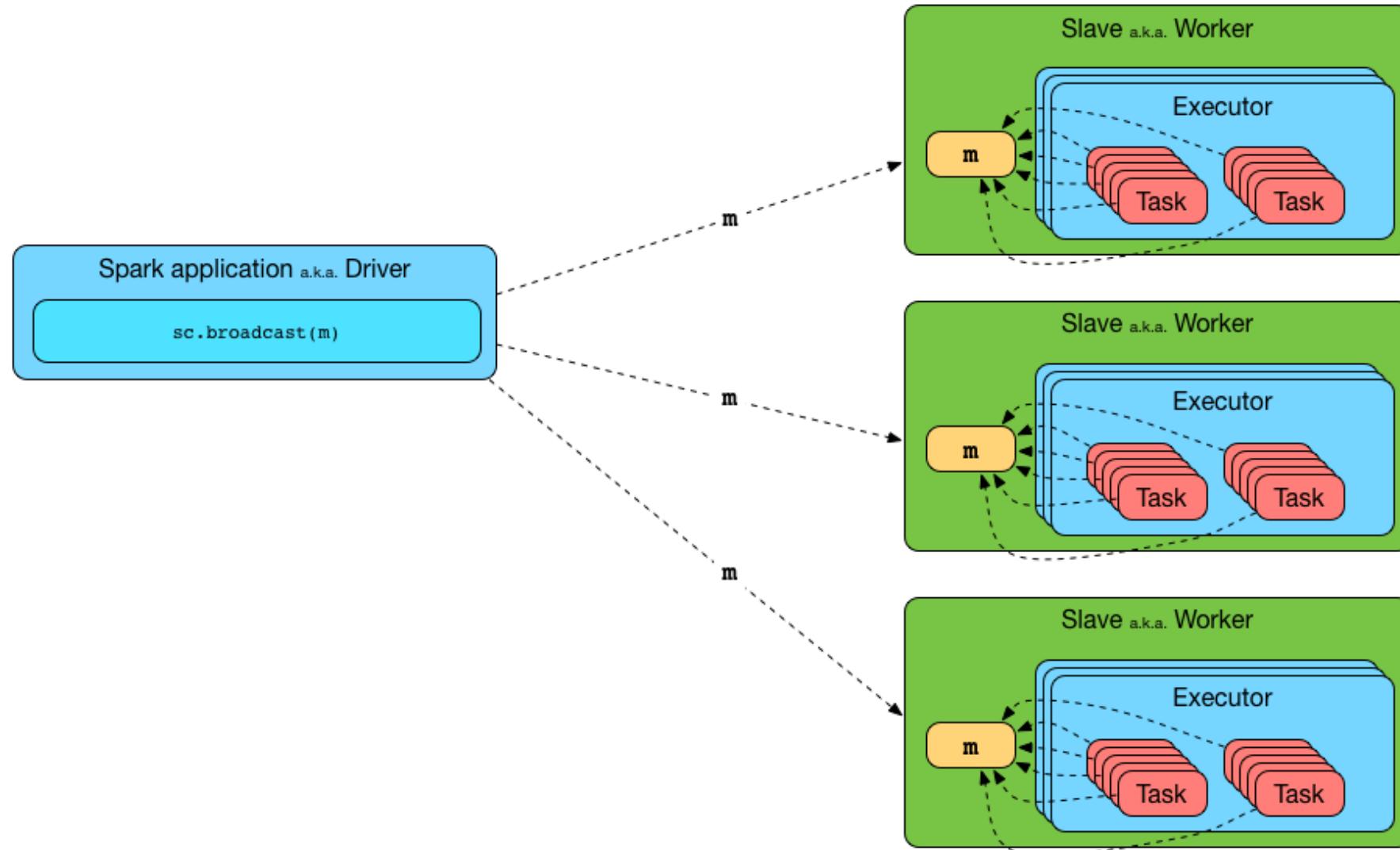
```
@Service
public class UglyService {

    @Autowired
    private CommonConfig config;

    public JavaRDD<String> resolveOperatorNames(JavaRDD<Tuple2<String, String>> pairs) {

        return
            pairs.filter(pair-> config.countries().contains(pair._1))
                  .map(pair-> config.getOperator(pair._2).getName());
    }
}
```

# Broadcast



```
@Service
public class UglyService {

    @Autowired
    private JavaSparkContext sc;

    @Autowired
    private CommonConfig commonConfig;

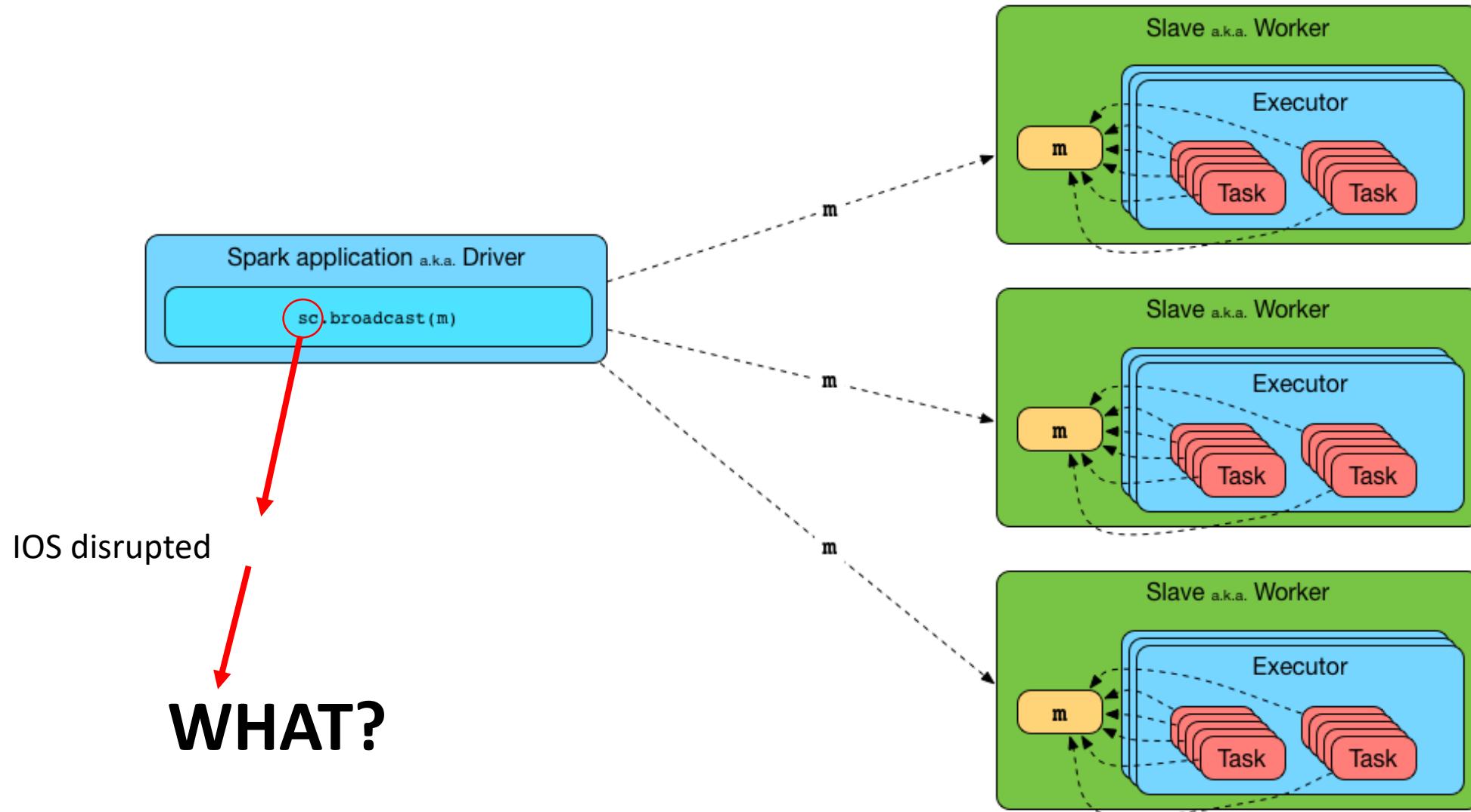
    private Broadcast<CommonConfig> configBroadcast;

    @PostConstruct
    public void wrapWithBroadCast() {
        configBroadcast = sc.broadcast(commonConfig);
    }

    public JavaRDD<String> resolveOperatorNames(JavaRDD<Tuple2<String, String>> pairs) {

        return pairs.filter(pair-> configBroadcast.value().countries().contains(pair._1))
                    .map(pair-> configBroadcast.value().getOperator(pair._2).getName());
    }
}
```

# What's problem



# IOS disrupted

*What's wrong?*



```
@Service
public class UglyService {

    @Autowired
    private JavaSparkContext sc;

    @Autowired
    private CommonConfig commonConfig;

    private Broadcast<CommonConfig> configBroadcast;

    @PostConstruct
    public void wrapWithBroadCast() {
        configBroadcast = sc.broadcast(commonConfig);
    }

    public JavaRDD<String> resolveOperatorNames(JavaRDD<Tuple2<String, String>> pairs) {

        return pairs.filter(pair-> configBroadcast.value().countries().contains(pair._1))
                    .map(pair-> configBroadcast.value().getOperator(pair._2).getName());
    }
}
```

IOS disrupted





Not impressed

*Not so crucial, it's not like  
injection of spring context*

A lot of copy paste





Not impressed

*Not so lot,  
Just several lines*

```
@Autowired  
private JavaSparkContext sc;  
  
@Autowired  
private CommonConfig commonConfig;  
  
private Broadcast<CommonConfig> configBroadcast;  
  
@PostConstruct  
public void wrapWithBroadCast() {  
    configBroadcast = sc.broadcast(commonConfig);  
}  
}
```

Technical spark code tangled  
with business logic





A close-up portrait of Wolfgang Amadeus Mozart, showing his face from the chest up. He has dark hair and is wearing a red and gold patterned jacket over a white cravat. The lighting is dramatic, highlighting his forehead and eyes.

Not impressed

*But you need this technical  
somewhere*

*But not in each bean which  
needs broadcast*





Not impressed

...

Tests will be more complicated





Not impressed

*But you anyway don't have  
pure unit tests...*

One more argument



A close-up portrait of George Washington's face. He has a serious expression, with his eyes looking slightly to the right. His hair is powdered white, and he wears a red military-style coat with a gold epaulet on his shoulder. The background is dark and textured.

Impressed

Caused by: java.io.NotSerializableException: com.inwhite.UglyService

Serialization stack:

- object not serializable (class: com.inwhite.UglyService, value: com.inwhite.UglyService@4fa4f485)
- element of array (index: 0)
- array (class [Ljava.lang.Object;, size 1)
- field (class: java.lang.invoke.SerializedLambda, name: capturedArgs, type: class [Ljava.lang.Object;)
  - object (class java.lang.invoke.SerializedLambda, SerializedLambda[capturingClass=class com.inwhite.UglyService, functionalInterfaceMethod=org/apache/spark/api/java/function/Function.call:(Ljava/lang/Object;)Ljava/lang/Object;, implementation=invokeSpecial com/inwhite/PopularWordsResolver.lambda\$mostUsedWords\$29bd749d\$1:(Ljava/lang/String;)Ljava/lang/Boolean;, instantiatedMethodType=(Ljava/lang/String;)Ljava/lang/Boolean;, numCaptured=1])
    - writeReplace data (class: java.lang.invoke.SerializedLambda)
  - object (class com.inwhite.UglyService\$\$Lambda\$6/1775488894, com.inwhite. com.inwhite.UglyService \$\$Lambda\$6/1775488894@5ec1963c)
    - field (class: org.apache.spark.api.java.JavaRDD\$\$anonfun\$filter\$1, name: f\$1, type: interface org.apache.spark.api.java.function.Function)
      - object (class org.apache.spark.api.java.JavaRDD\$\$anonfun\$filter\$1, <function1>
        - at org.apache.spark.serializer.SerializationDebugger\$.improveException(SerializationDebugger.scala:40)
        - at org.apache.spark.serializer.JavaSerializationStream.writeObject(JavaSerializer.scala:47)
        - at org.apache.spark.serializer.JavaSerializerInstance.serialize(JavaSerializer.scala:101)
        - at org.apache.spark.util.ClosureCleaner\$.ensureSerializable(ClosureCleaner.scala:301)
- ... 44 more

*So we can do something like this*



```
@Configuration  
@ComponentScan(basePackages = "mpi")  
public class SpringConfig {
```

```
@Bean  
public UglyService uglyService() {  
    UglyService uglyService = new UglyService();  
    uglyService.setCommonConfig(sc().broadcast(commonConfig));  
    return uglyService;  
}
```

```
@Bean  
public JavaSparkContext sc() {  
    SparkConf conf = new SparkConf();  
    conf.setAppName("songs");  
    conf.setMaster("local[*]");  
    return new JavaSparkContext(conf);  
}
```



COME WITH ME IF YOU WANT TO LIVE

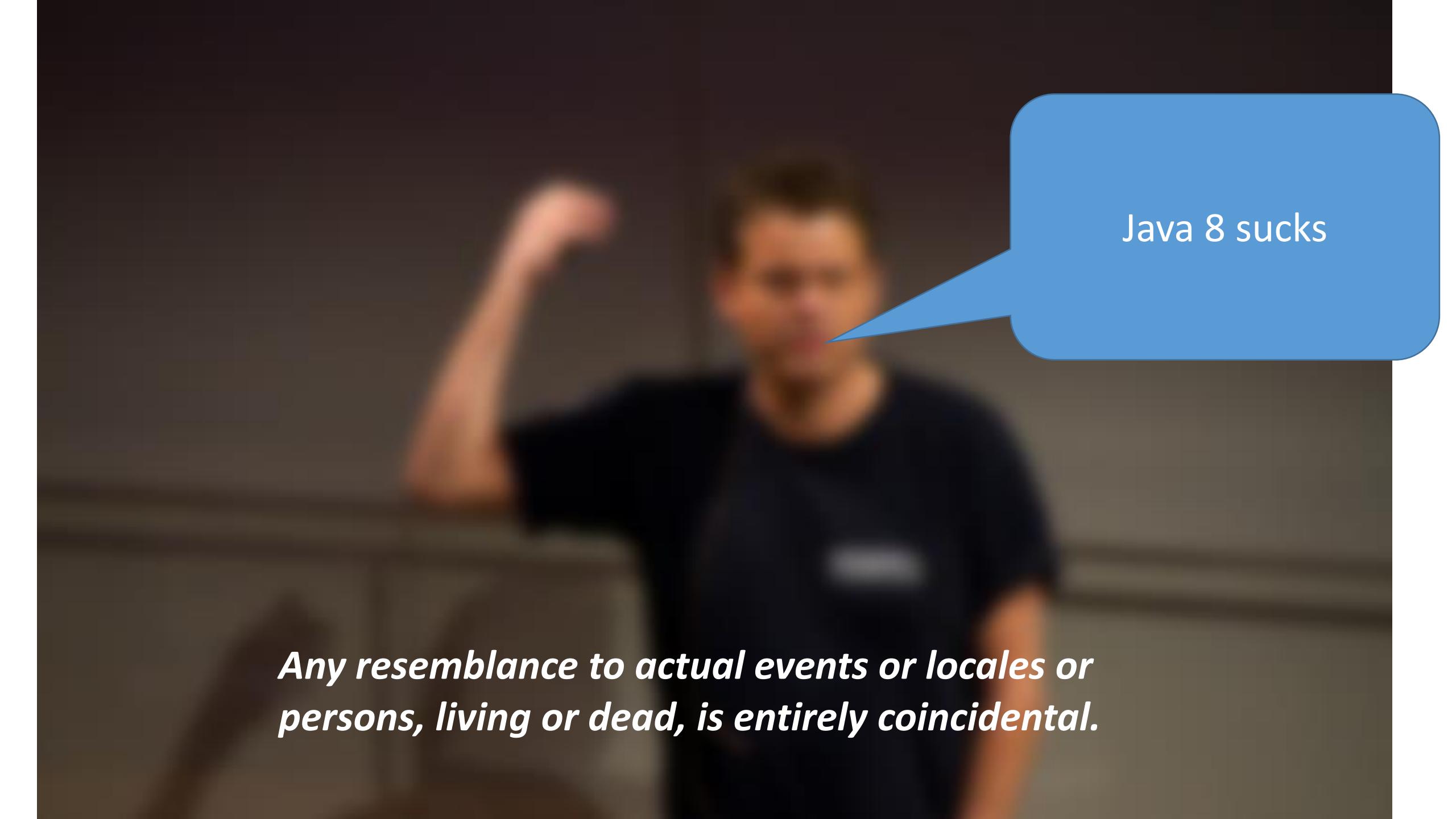
# Syntax

## Java

```
lines.map(String::toLowerCase)
.flatMap(WordsUtil::getWords)
.filter(word-> !Arrays.asList(garbage).contains(word))
.mapToPair(word-> new Tuple2<>(word, 1))
.reduceByKey((x, y)->x+y)
.mapToPair(Tuple2::swap)
.sortByKey(false)
.map(Tuple2::_2)
.take(amount);
```

## Scala

```
lines.map(_.toLowerCase())
.flatMap("^\w+".r.findAllIn(_))
.filter(!garbage.contains(_))
.map((_, 1)).reduceByKey(_ + _)
.sortBy(_._2, ascending = false)
.take(amount)
```



Java 8 sucks

*Any resemblance to actual events or locales or persons, living or dead, is entirely coincidental.*

**ONE DOESN'T SIMPLY CAN WRITE**

**BEAUTIFUL CODE IN JAVA 8**

# DataFrames

# Dataframes – since 1.3

- DataFrame is a distributed collection of data organized into named columns
- RDD under the hood, but with lot of optimizations
- Use less memory if you use them smart
- Can be created from Hive tables, json like files, regular RDD, databases, all that have schema
- Have very wide DSL
- Related with SqlContext

```
@Bean  
public SQLContext sqlContext () {  
    return new SQLContext (sc ()) ;  
}
```

# Create DataFrame from json file

```
sqlContext.read().json("data/*.json");
```

# How to configure SqlContext?

```
SQLContext sqlContext = new SQLContext(sc);
sqlContext.setConf("spark.sql.shuffle.partitions", "1");
sqlContext.setConf("spark.sql.parquet.compression.codec", "snappy");
```

# Dataframes methods and functions

Agg, columns, count, distinct, drop, dropDuplicates, filter  
groupBy, orderBy, registerTable, schema, show, select, where,  
withColumn

# Some dataframes methods

Agg, columns, count, distinct, drop, dropDuplicates, filter  
groupBy, orderBy, **registerTable**, schema, show, select, where,  
withColumn

```
dataFrame.registerTempTable("finalMap");

DataFrame frame = sqlContext.sql("select cl_id, cl_grp_id, dk_org_snw, dk_org_hnw,
    dk_org_cnp, dk_dir, dk_dat, DK_TIM_HR as dk_tim_hr, dk_spe, dk_sgt, dk_pet, dk_sgs, dk_sbp,\n" +
    "SUM(slu_atpt) slu_atpt, SUM(slu_succ) slu_succ, SUM(slu_fail) slu_fail, SUM(slu_dly) slu_dly\n" +
    "FROM finalMap f join tdtim t on f.dk_tim = t.DK_TIM\n" +
    "WHERE dk_pet IN (1, 4)\n" +
    "group by cl_id, cl_grp_id, dk_org_snw, dk_org_hnw, dk_org_cnp, dk_dir, dk_dat, DK_TIM_HR,
    dk_spe, dk_sgt, dk_pet, dk_sgs, dk_sbp").toDF();
```

# Don't tell him about SqlContext



# Why I don't love sql

```

DataFrame dataFrame1 = sqlContext.sql("select tdmco.CL_ID as cl_id, \n" +
  "\t\tdmdcl.CL_GRP_ID as cl_grp_id, \n" +
  "\t\tdtpet.DK_PET as dk_pet, \n" +
  "\t\tdsgt.DK_SGT as dk_sgt, \n" +
  "\t\tdsgs.DK_SGS as dk_sgs, \n" +
  "\t\tddir.DK_DIR as dk_dir, \n" +
  "\t\tdorg.DK_ORG as dk_org_cnp,\n" +
  "\t\tcase when data.INBOUND_IND = 0 then tdorg.DK_ORG when data.INBOUND_IND = 1 then tdmco.CL_ID end as dk_org_hnw,\n" +
  "\t\tcase when data.INBOUND_IND = 1 then tdorg.DK_ORG when data.INBOUND_IND = 0 then tdmco.CL_ID end as dk_org_snw,\n" +
  "\t\tddat.DK_DAT as dk_dat,\n" +
  "\t\ttdtim.DK_TIM as dk_tim,\n" +
  "\t\tcast(case when spel.DK_SPE is not null then spel.DK_SPE when spe2.DK_SPE is not null then spe2.DK_SPE when spe3.DK_SPE is not null then spe3.DK_SPE else 0 end as bigint)\n" +
  "\t\tcast(-999 as bigint) as dk_sgl, \n" +
  "\t\tcast(-999 as bigint) as dk_adt,\n" +
  "\t\tcast(0 as bigint) as dk_sbp,\n" +
  "\t\tdata.IMSI as dd_imsi,\n" +
  "\t\tMSISDN as dd_msisdn_min, \n" +
  "\t\tMSC as dd_msc, \n" +
  "\t\tHLR as dd_hlr, \n" +
  "\t\tSGSN_GT as dd_sgsn_gt, \n" +
  "\t\tSGSN_IP as dd_sgsn_ip,\n" +
  "\t\tcast(1 as bigint) as slu_atpt,\n" +
  "\t\tcast(case when data.STATUS = 'OK' then 1 else 0 end as bigint) as slu_succ,\n" +
  "\t\tcast(case when data.STATUS = 'Reject' then 1 else 0 end as bigint) as slu_fail,\n" +
  "\t\tcast((unix_timestamp(substring(data.TIME_END,1,19))*1000)+(substring(data.TIME_END,21,3)) - ((unix_timestamp(substring(data.TIME,1,19))*1000)+(substring(data.TIME,21,3))) as bigint)\n" +
  "\t\tdata.STATUS as status\n" +
  "\t\tfirst_value(tmdbsprf.PRF_ID) OVER (PARTITION BY tdmco.CL_ID,data.IMSI ORDER BY length(tmdbsprf.MATCH_STR) desc,tmdbsprf.MATCH_START) AS sbs_prf_if,\n" +
  "from data left join tdmco on (data.OWNER_CODE=tdmco.MCO_TAP_CD)\n" +
  "left join tmdcl on (tdmco.CL_ID=tmdcl.CL_ID)\n" +
  "left join tdpes on (data.OPCODE=tdpet.PET_CD)\n" +
  "left join tdsqt on (case when data.OPCODE = 2 then 'C' when data.OPCODE = 23 then 'P' end = tdsqt.SGT_CD)\n" +
  "left join tdsqs on (case when data.STATUS = 'OK' then 'S' when data.STATUS = 'Reject' then 'F' end = tdsqs.SGS_CD)\n" +
  "left join tddir on (case when data.INBOUND_IND = 0 then 'V' when data.INBOUND_IND = 1 then 'O' end = tddir.DIR_CD)\n" +
  "left join tdorg on (data.OP_ID = tdorg.ORG_TAP_CD)\n" +
  "left join tddat on (to_date(data.TIME) = to_date(tddat.DAT))\n" +
  "left join tdtim on (substring(data.TIME,12,8) = tdtim.TIM)\n" +
  "left join tdspe as spel on (case when data.STATUS = 'Reject' and data.MAP_ERR is null and TCAP_ERR is null then 255 end = spel.SPE_CD and spel.SPE_CAT = 'MAP')\n" +
  "left join tdspe as spe2 on (case when data.TCAP_ERR is null or (data.MAP_ERR is not null and data.TCAP_ERR is not null) then data.MAP_ERR end = spe2.SPE_CD and spe2.SPE_CAT = 'TCAP')\n" +
  "left join tdspe as spe3 on (case when data.MAP_ERR is null and data.TCAP_ERR is not null then data.TCAP_ERR end = spe3.SPE_CD and spe3.SPE_CAT = 'TCAP')\n" +
  "-- left join tmdbsprf on (tdmco.CL_ID = tmdbsprf.org_tech_id and SUBSTR(data.IMSI, tmdbsprf.MATCH_START, LENGTH(tmdbsprf.MATCH_STR)) = tmdbsprf.MATCH_STR and to_date(data.TIME) = tmdbsprf.MATCH_DATE)

```

# Dataframes methods and functions

- abs, cos, asin, isnull, not, rand, sqrt, when, expr, bin, atan, ceil, floor, factorial, greatest, least, log, log10, pow, round, sin, toDegrees, toRadians, md5, ascii, base64, concat, length, lower, ltrim, unbase64, repeat, reverse, split, substring, trim, upper, datediff, year, month, hour, last\_day, next\_day, dayofmonth, explode, udf

# Dataframes methods and functions

- abs, cos, asin, isnull, not, rand, sqrt, when, expr, bin, atan, ceil, floor, factorial, greatest, least, log, log10, pow, round, sin, toDegrees, toRadians, md5, ascii, base64, concat, length, lower, ltrim, unbase64, repeat, reverse, split, substring, trim, upper, datediff, year, month, hour, last\_day, next\_day, dayofmonth, **explode**, udf

# Json file I'll send you

```
{ "name": "Evgeny Borisov", "age": 37, "keywords": ["spark", "java", "spring", "gradle", "groovy", "artifactory"] }  
{ "name": "Martin Odersky", "age": 37, "keywords": ["scala"] }  
{ "name": "Baruch Sadogursky", "age": 38, "keywords": ["docker", "spring", "groovy", "bintray", "artifactory", "linux"] }  
{ "name": "Motros Jeleznyak", "age": 37, "keywords": ["python", "ruby"] }  
{ "name": "Anton Arhipov", "age": 37, "keywords": ["scala", "groovy", "gc", "sbt", "aspectJ"] }  
{ "name": "Viktor Gamov", "age": 28, "keywords": ["spring", "maven", "spock", "spring boot"] }  
{ "name": "Kirill Tolkachov", "age": 28, "keywords": ["groovy", "spring", "maven", "spock", "spring boot"] }  
{ "name": "Nikolay Alimenkov", "age": 35, "keywords": ["java", "spring", "maven", "spock", "TestNG", "junit", "mockito"] }
```

# Lab

- Print out file content
- Print out the schema
- Print out the column type
- Add column “salary” the value will be calculated by formula:  
age \* number of technologies \* 10
- Find developers with salary < 1200 and which familiar with most popular technology

# How to create DataFrame from CSV

- Create RDD<Row>
  - Use - RowFactory::create
- Use method createDataFrame of SqlContext
- It will take your rdd and StructType
- How to create StructType:

```
 DataTypes.createStructType(new StructField[] {  
    DataTypes.createStructField("raw data", DataTypes.StringType, true)  
} );
```

# Udf functions

```
@Component
public class GarbageFilter implements UDF1<String, Boolean> {
    @AutowiredBroadcast
    private Broadcast<UserConfig> userConfig;

    public String udfName() {return "notGarbage"; }

    @Override
    public Boolean call(String word) throws Exception {
        return ! userConfig.value().garbage.contains(word);
    }
}
```

# How to use Udf function

```
@Service
public class PopularWordsResolverWithUDF {

    @Autowired
    private GarbageFilter garbageFilter;

    @Autowired
    private SQLContext sqlContext;

    @PostConstruct
    public void registerUdf() {
        sqlContext.udf().register(garbageFilter.udfName(), garbageFilter, DataTypes.BooleanType);
    }

    public List<String> mostUsedWords(DataFrame dataFrame, int amount) {
        DataFrame sorted = dataFrame.withColumn("words", lower(column("words")))
            .filter(callUDF(garbageFilter.udfName(), column("words")) ...)
```

# UDF in Scala

```
val upperUDF = functions.udf { usd: Int => usd*getUsdRate() }

dataFrame.withColumn("NIS", upperUDF(functions.col("USD"))))
```

# What spring can give us

- IOC
- Chain of responsibility
- @Conditional / @Profile
- @AutowireBroadcast
- Custom qualifier
- AOP - @ShowDataFrame

Final project...