

Who are you

- Java developer since 2001
- Java Trainer since 2003
- Java consultant since 2008
- Java architect since 2009
- InWhite Owner
- DemocraC startup founder since 2014
- Big Data Technical Leader at Naya technologies since 2015



Spark

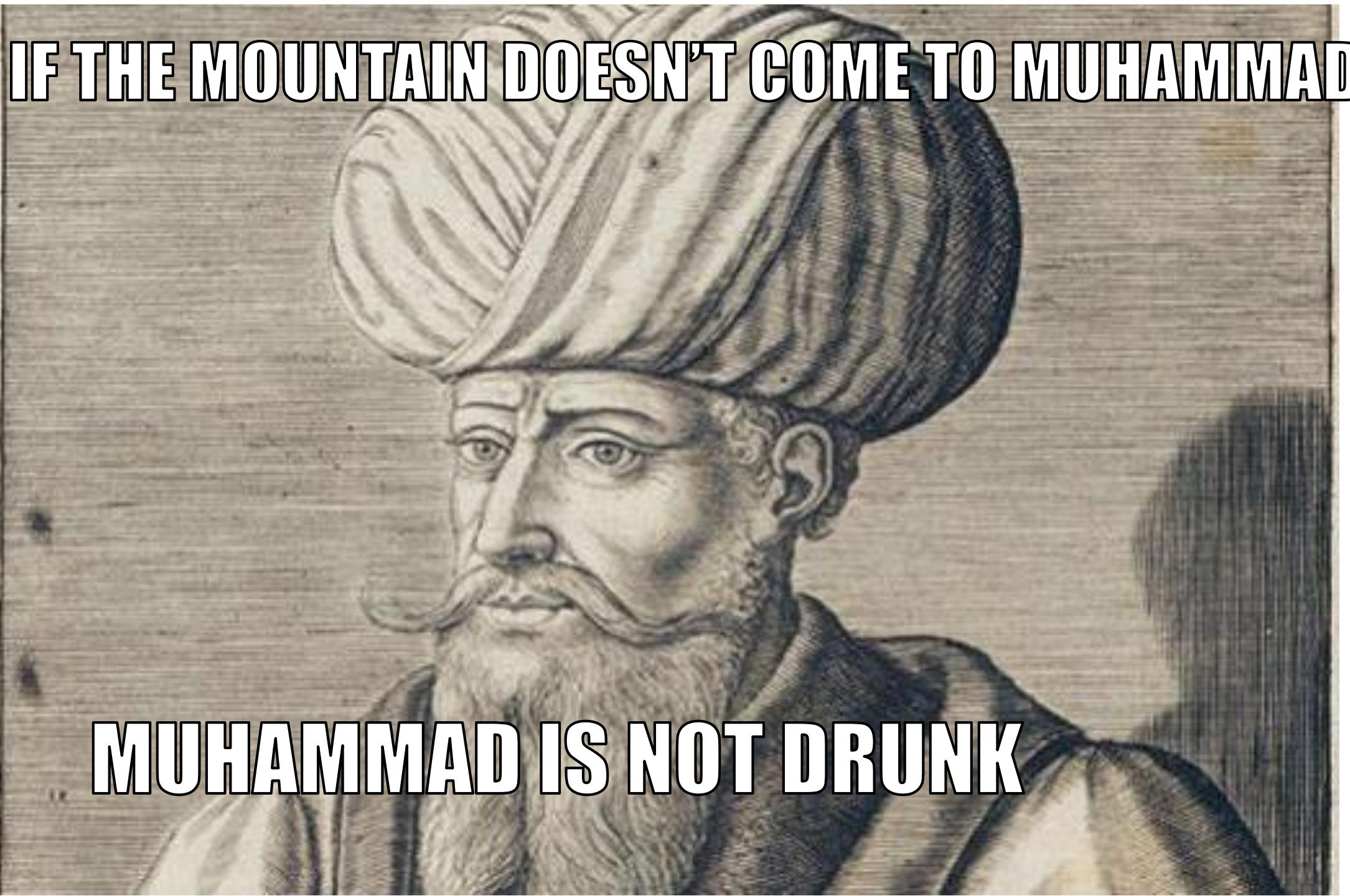
evgenyb@naya-tech.co.il

Agenda

- Hadoop ecosystem
- Functional programming
- Scala – what we need to know
- Spark API

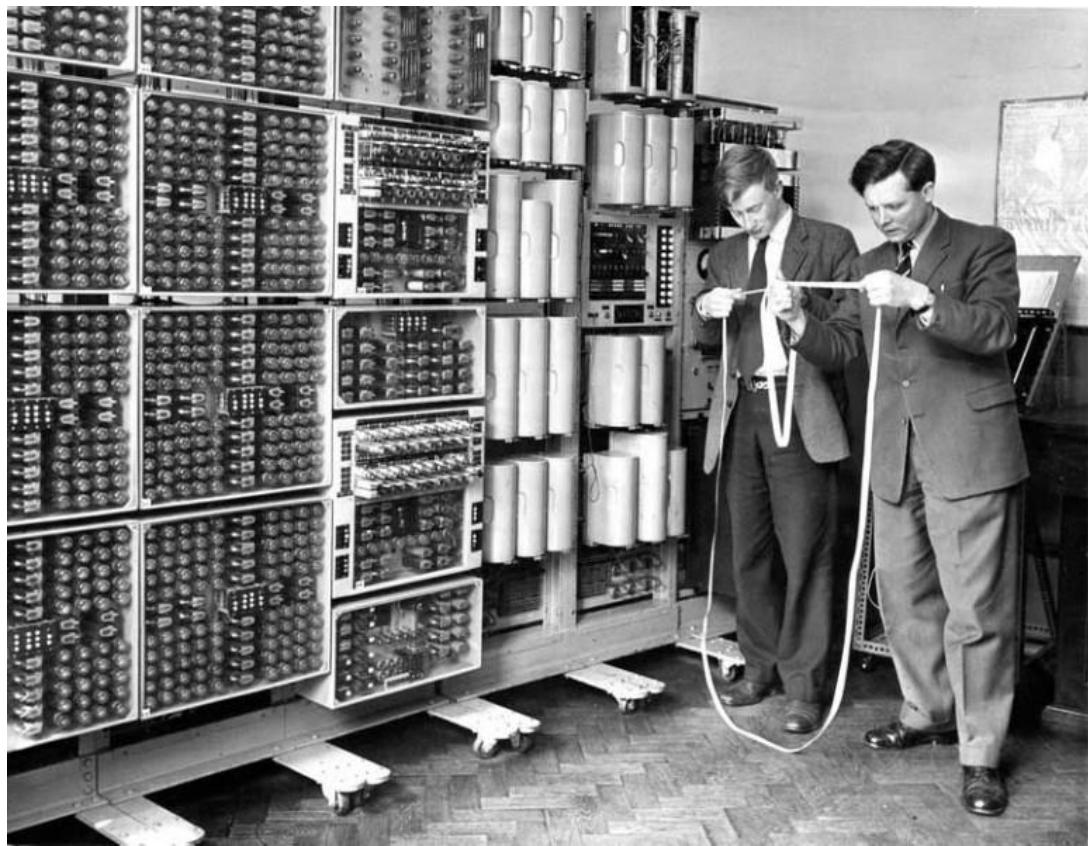
Data locality

IF THE MOUNTAIN DOESN'T COME TO MUHAMMAD

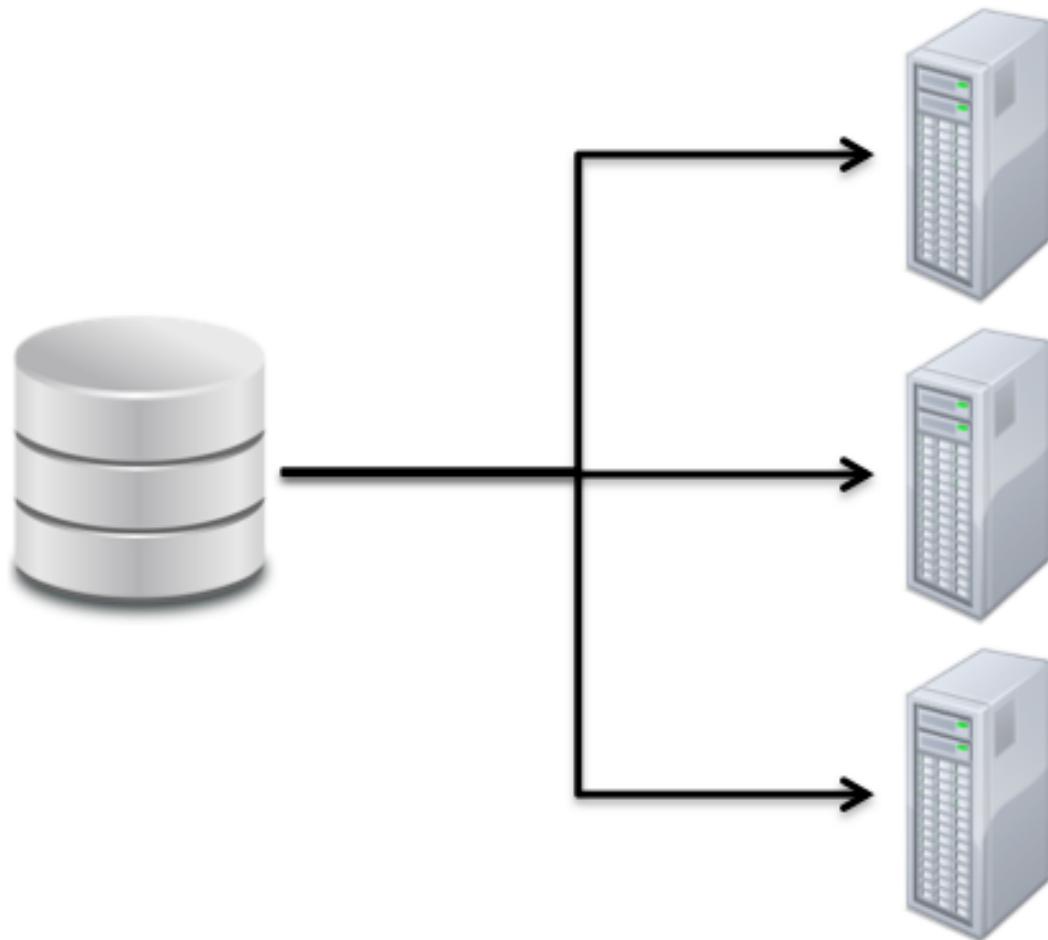


MUHAMMAD IS NOT DRUNK

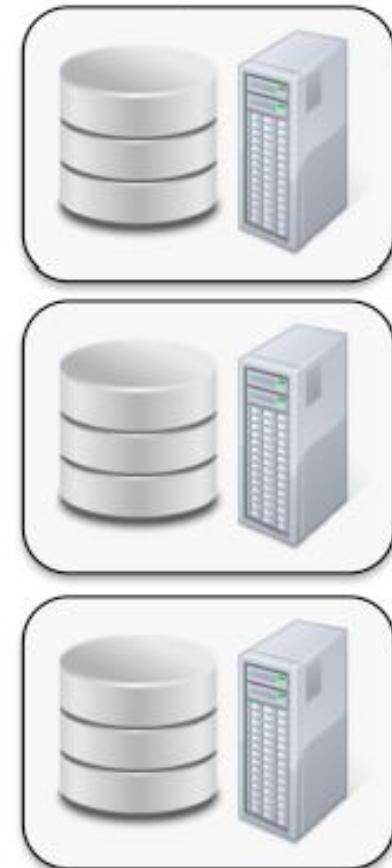
Yesterday against today



The bottleneck

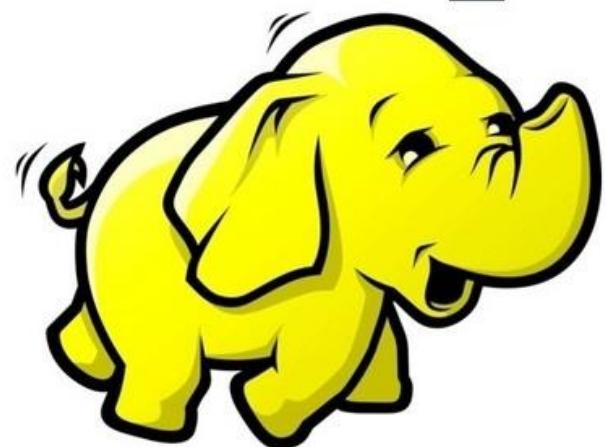


New model



Hadoop Sucks....

hadoop



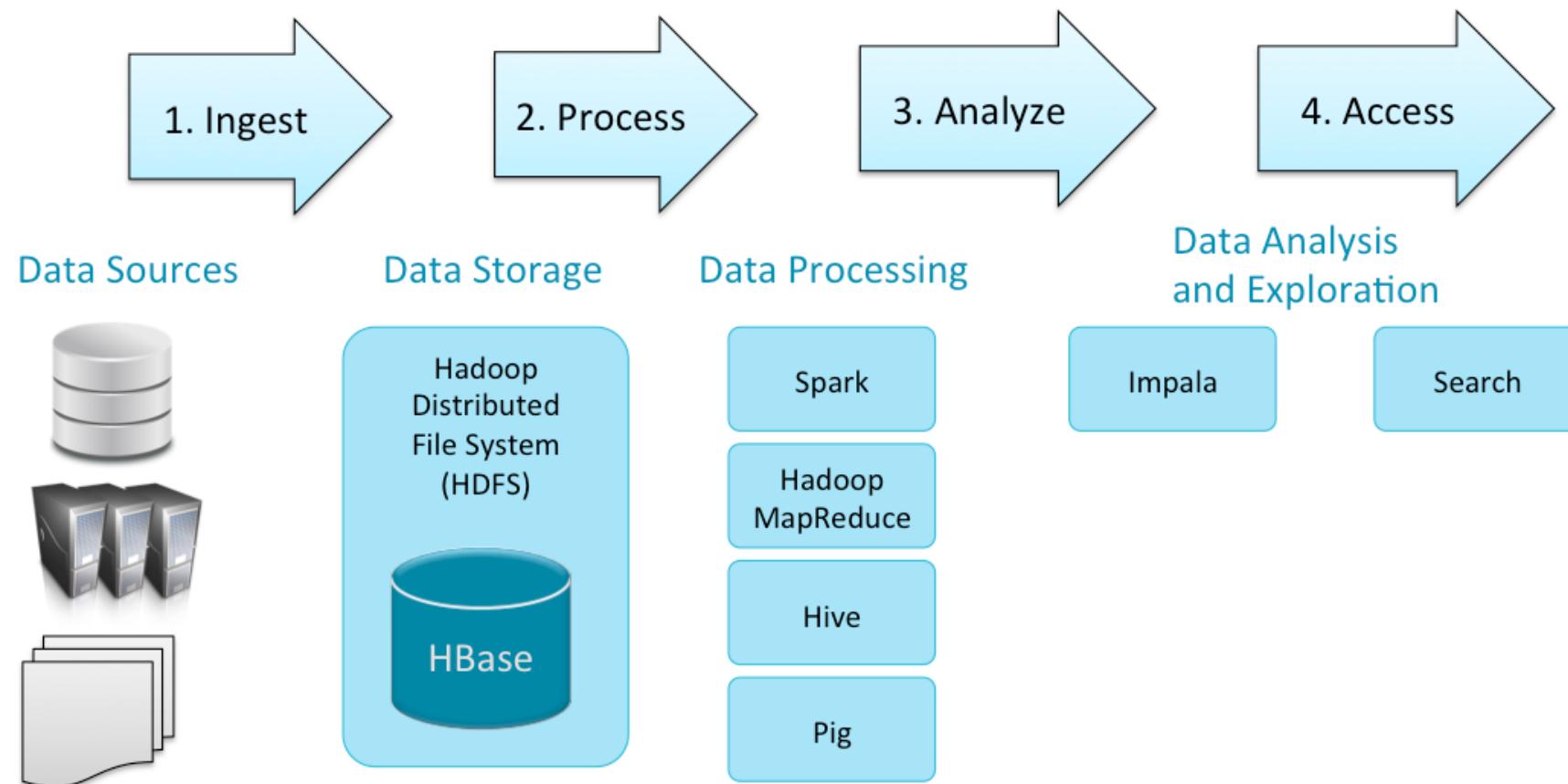
Hadoop implementations

- Vanilla
- Hortonworks
- Cloudera

Hadoop ecosystem includes

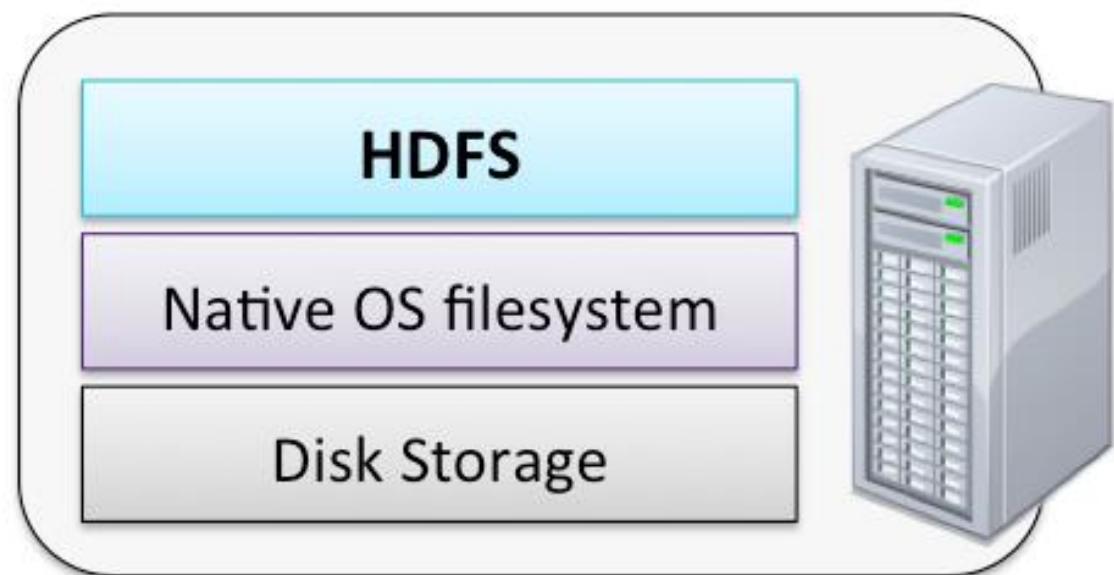
- HDFS
- HBASE
- Hue
- Yarn
- Sqoop
- Flume
- Hive
- Spark (Cloudera, Hortonworks)
- Solr
- Pig
- Impala (Cloudera)
- Zookeeper
- Oozie

The process



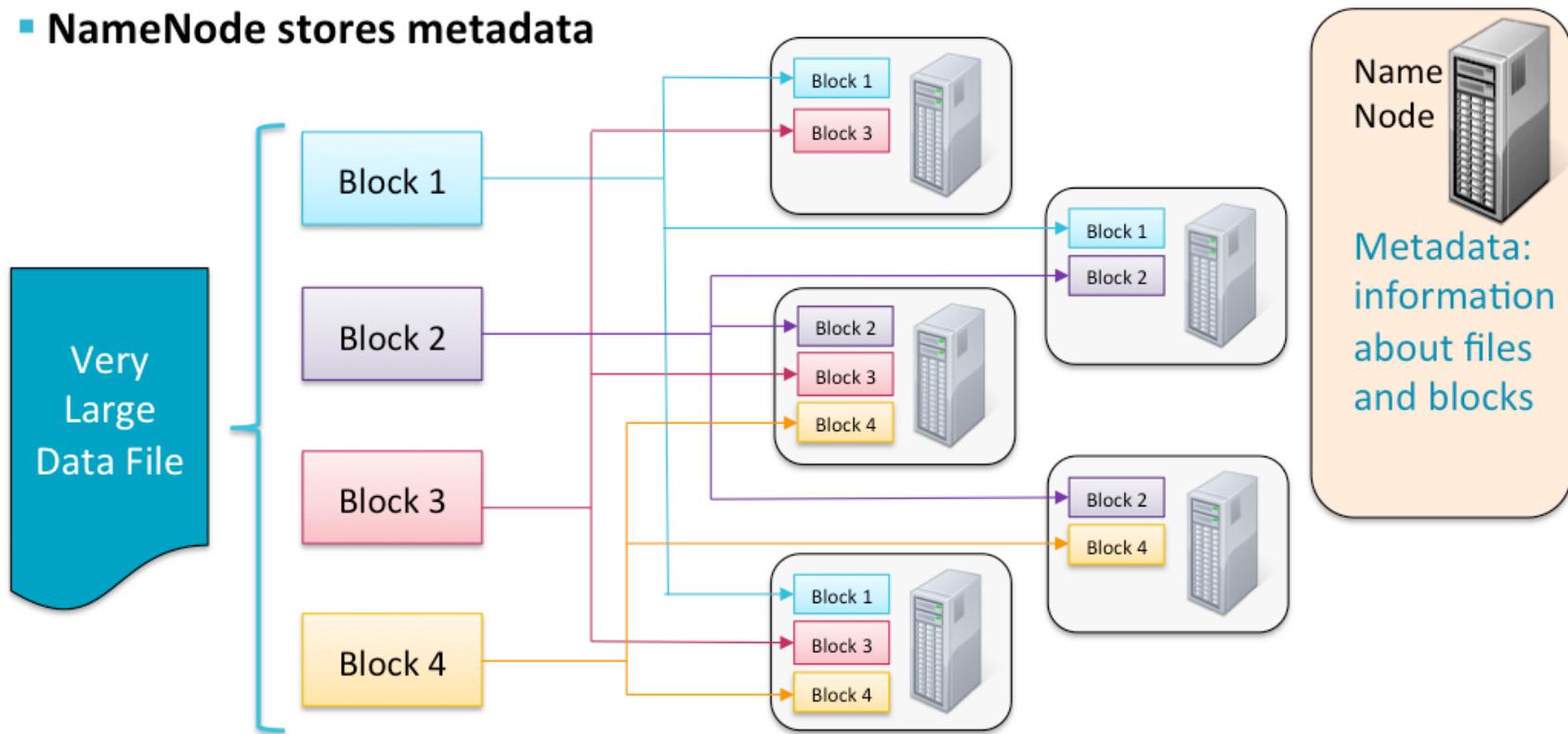
Storing data in Hadoop

- 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

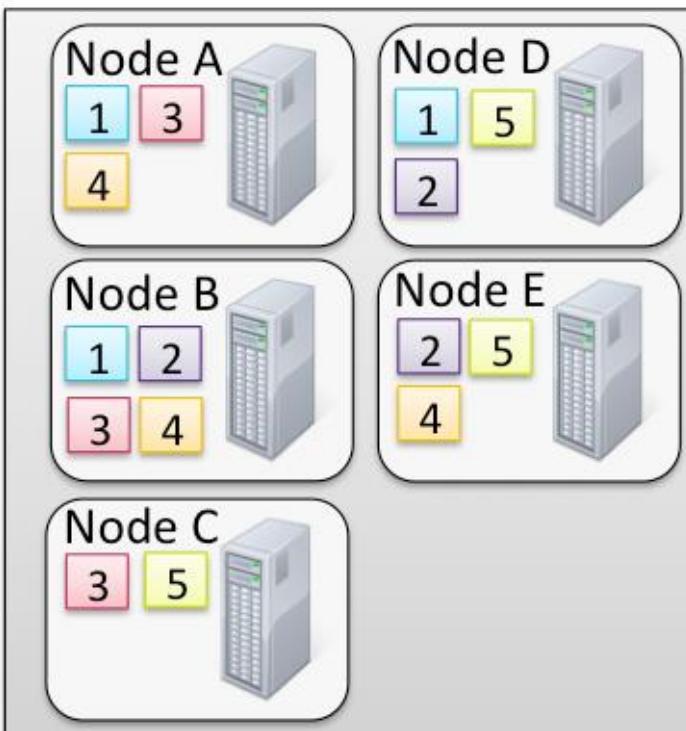
B2: B, D, E

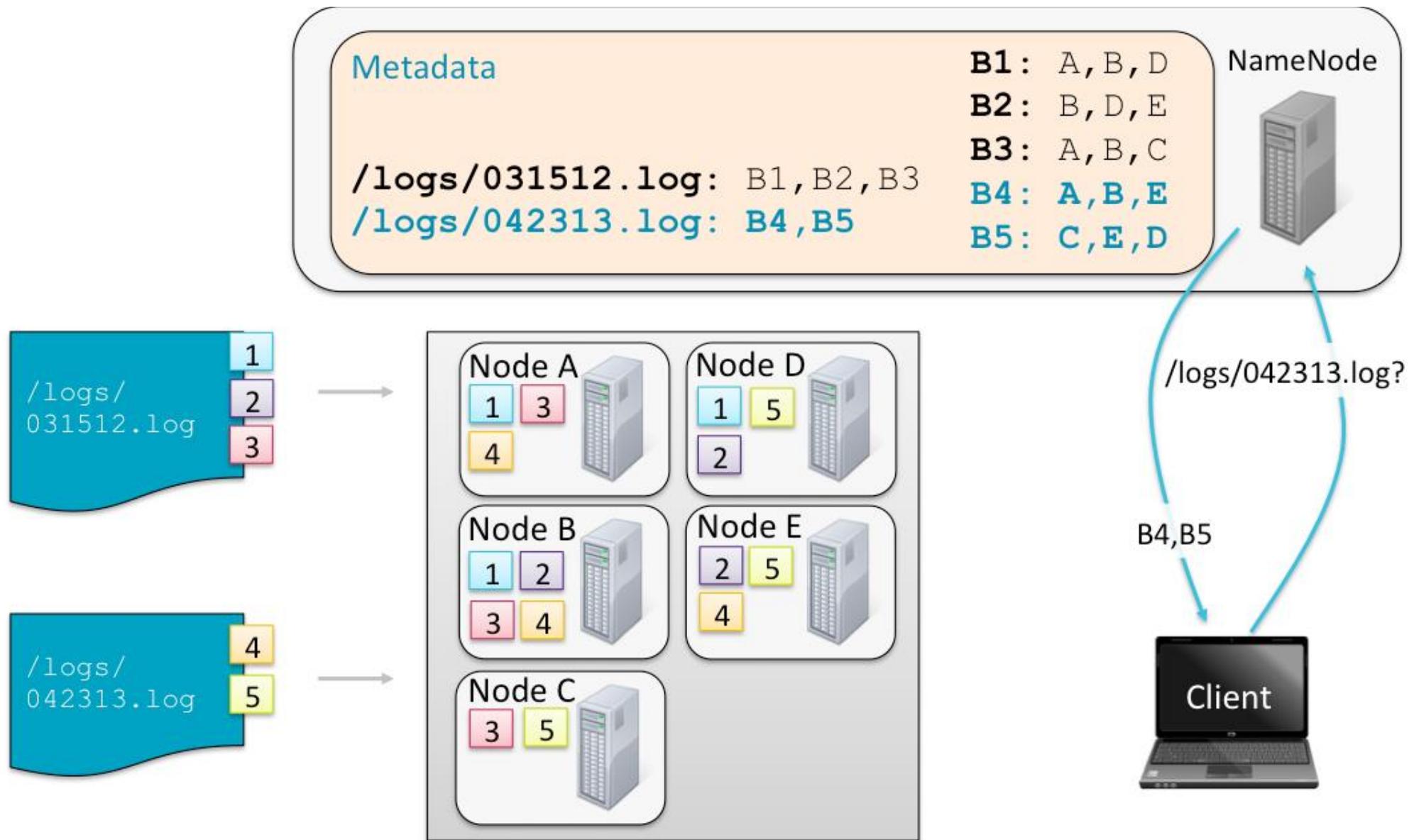
B3: A, B, C

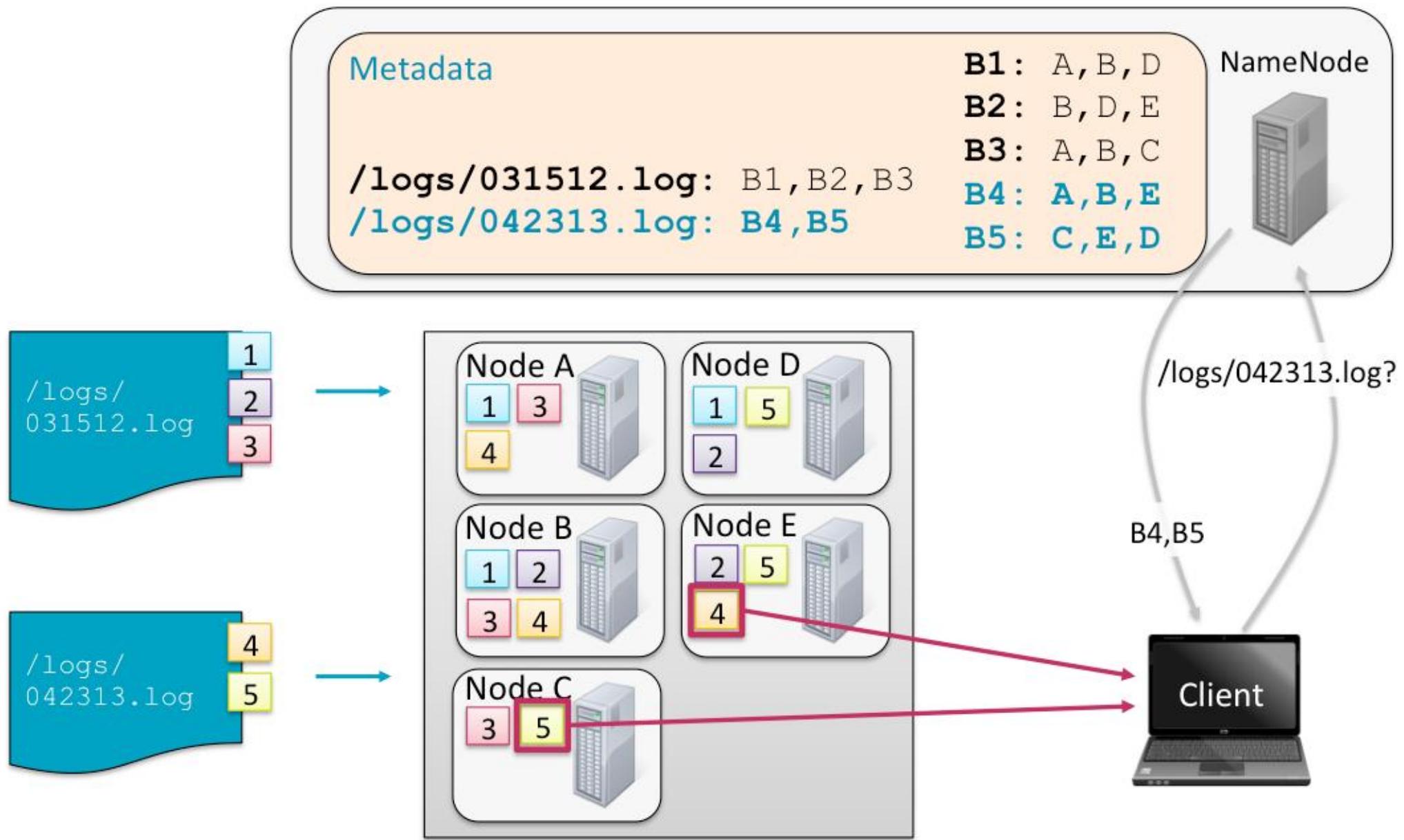
B4: A, B, E

B5: C, E, D

NameNode







HDFS: cmd

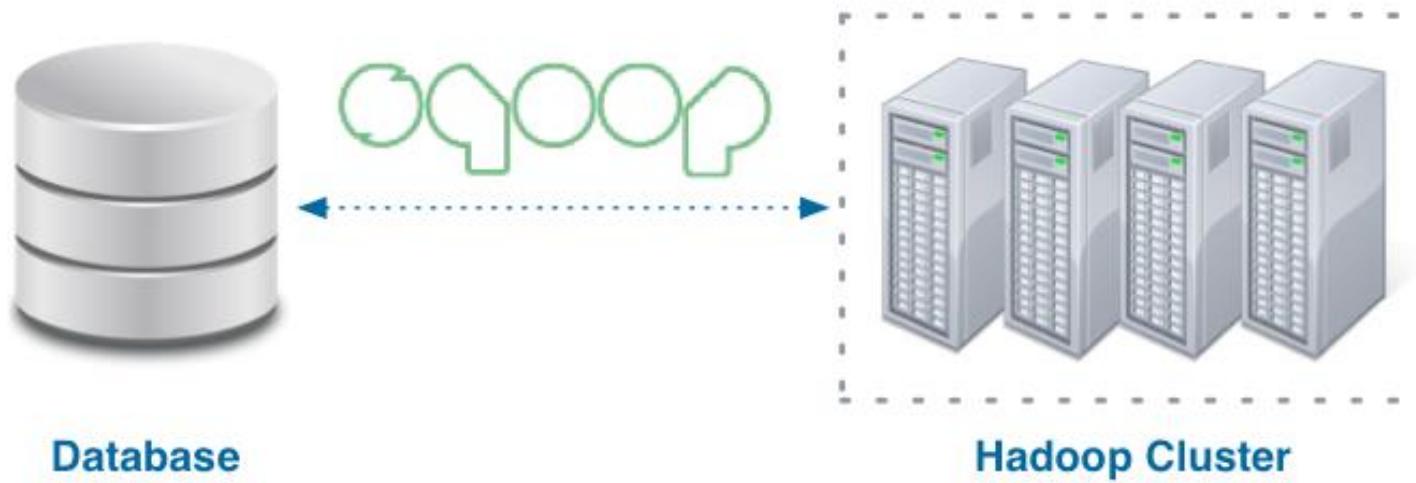
- hdfs dfs –put bigFile.txt bigFile.txt
- hdfs dfs –ls
- hdfs dfs –get /user/jeka/ bigFile.txt bigFile.txt
- hdfs dfs –mkdir myDir
- hdfs dfs –rm –r backupDir

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)

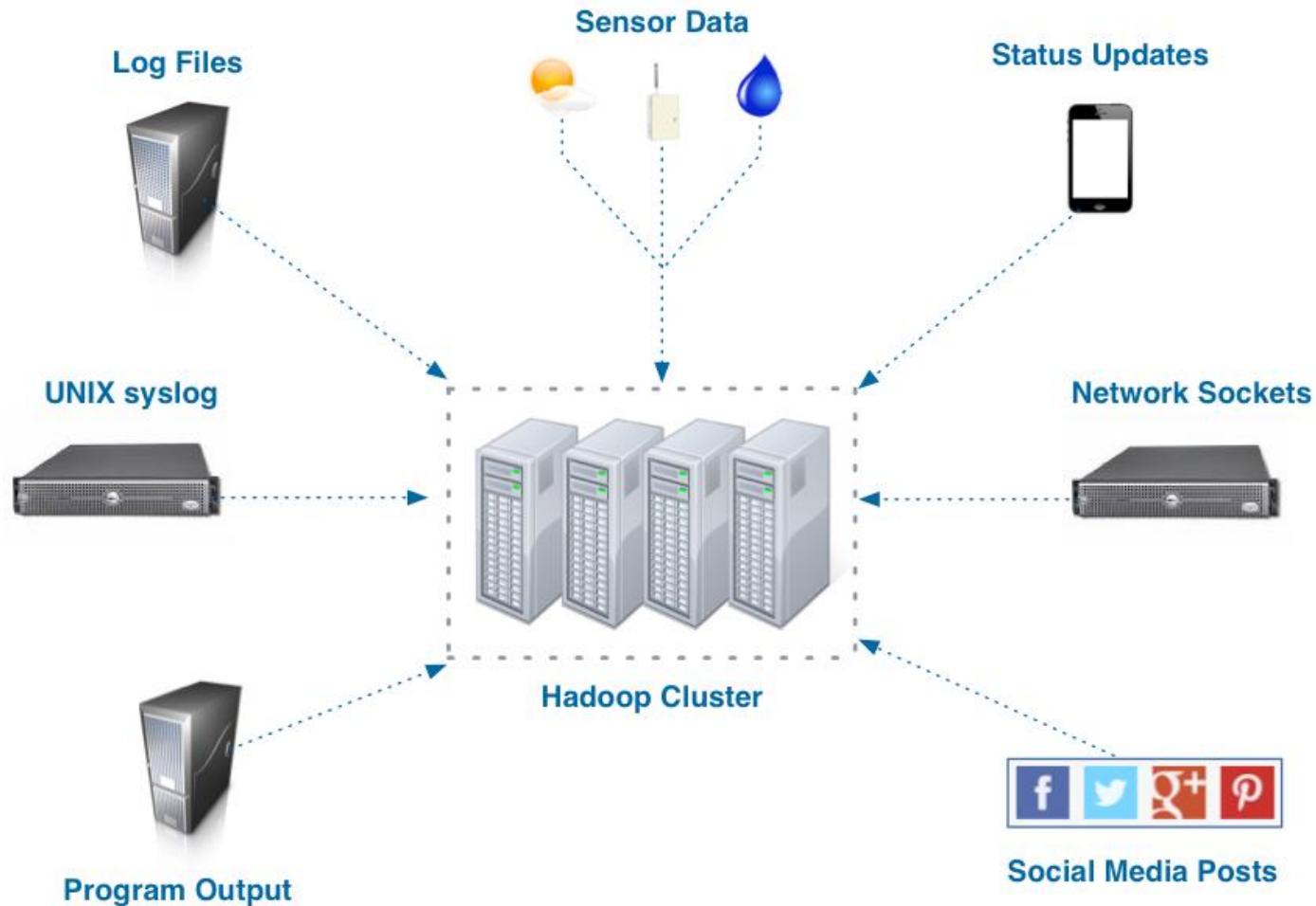
Export

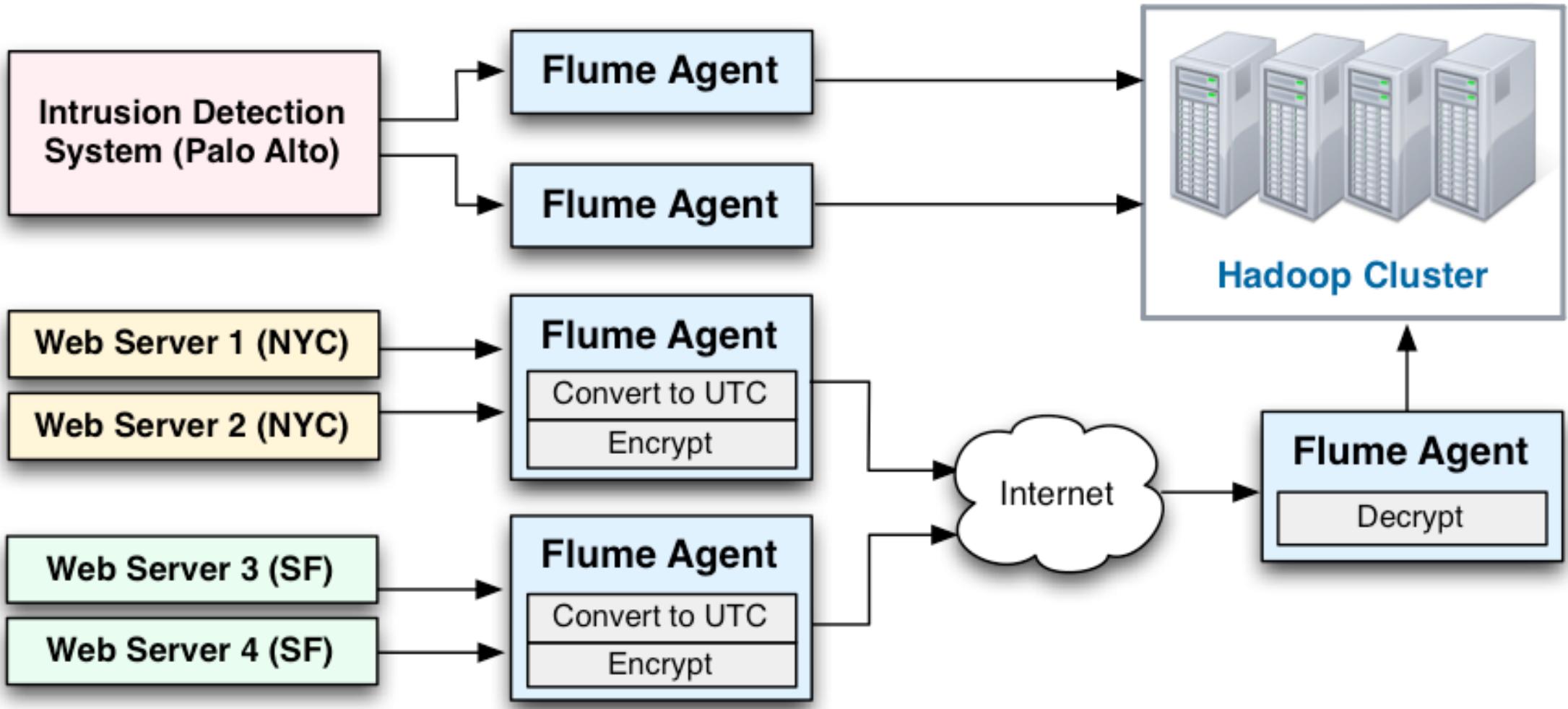
```
sqoop export \
  --connect jdbc:mysql://dbhost/loudacre \
  --username dbuser --password pw \
  --export-dir /loudacre/recommender_output \
  --update-mode allowinsert \
  --table product_recommendations
```

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?

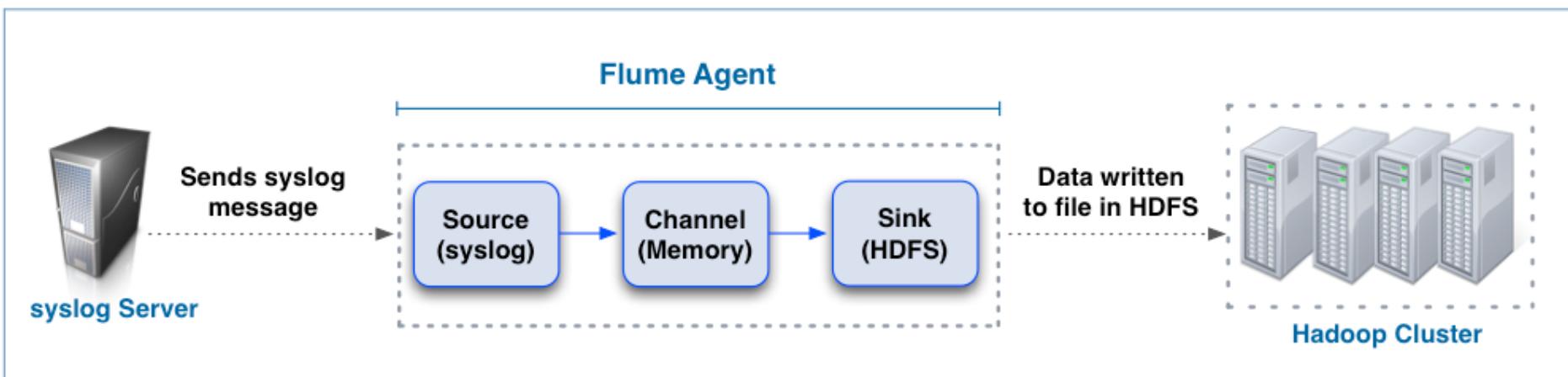




Channel can be changed (not only memory)

- 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)

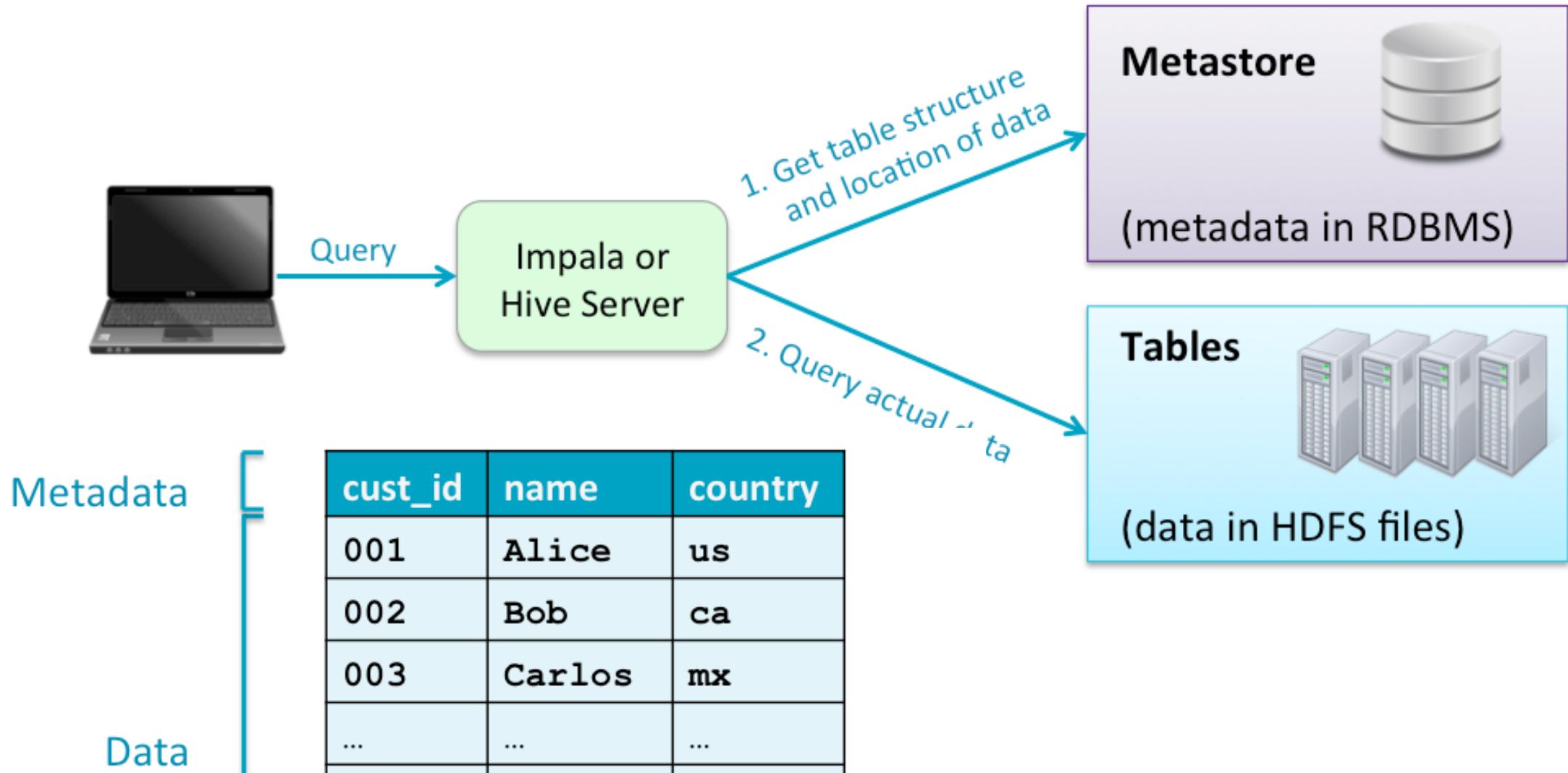
Why not to use Impala instead of Hive always?

- 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



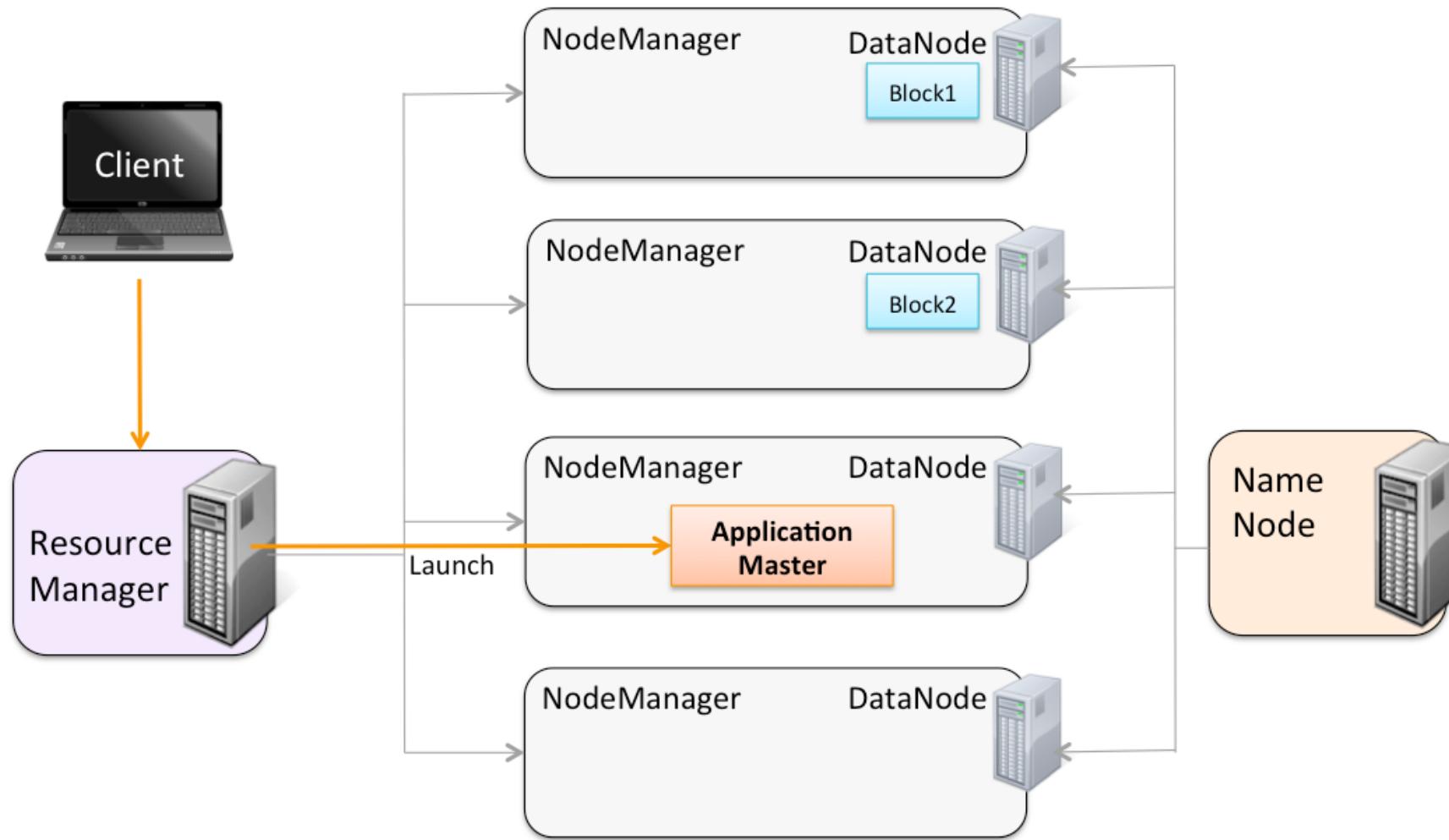
Where tables come from?

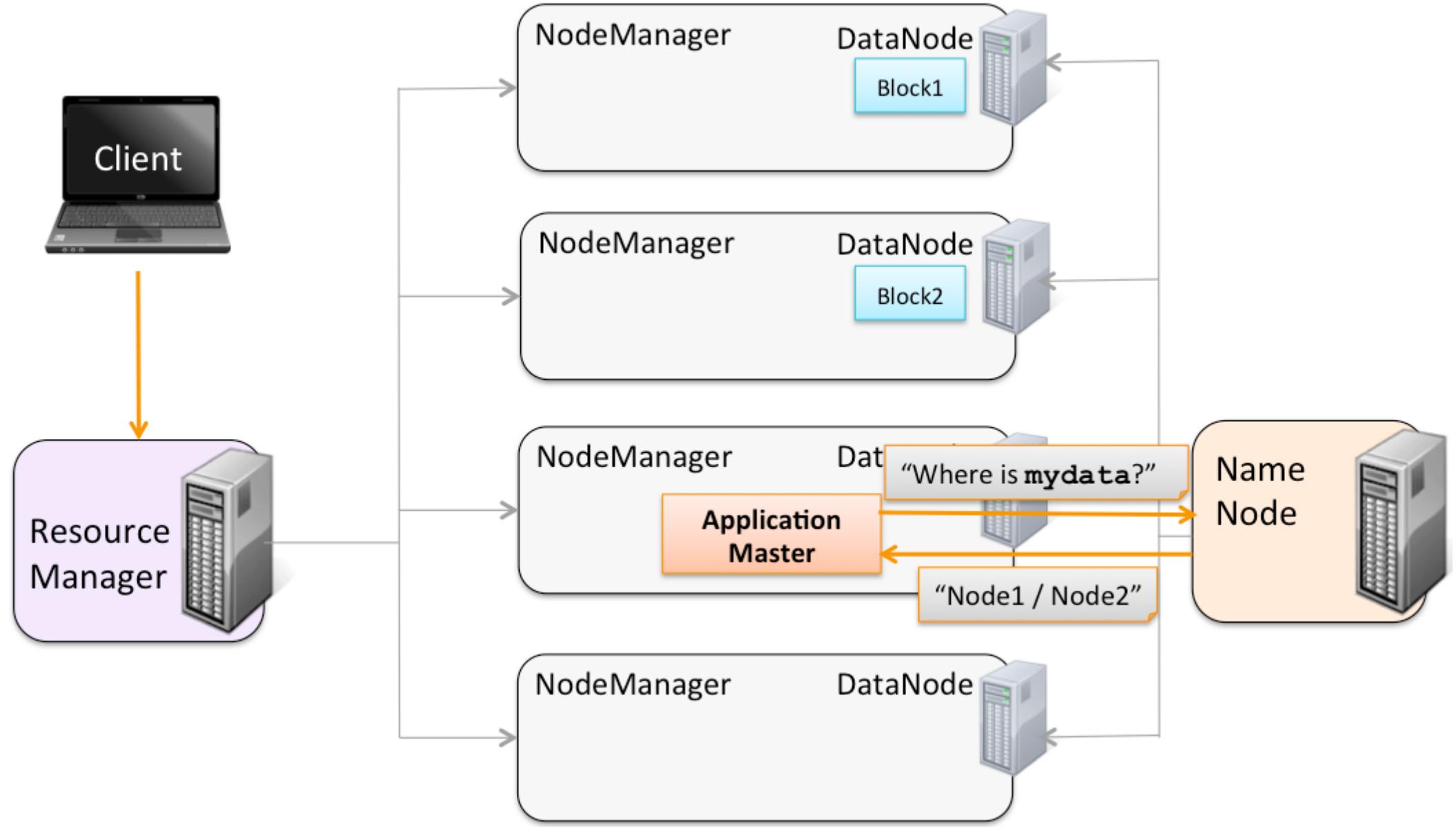
```
CREATE TABLE tablename (colname DATATYPE, . . .)  
ROW FORMAT DELIMITED  
FIELDS TERMINATED BY char  
STORED AS {TEXTFILE|SEQUENCEFILE|...}
```

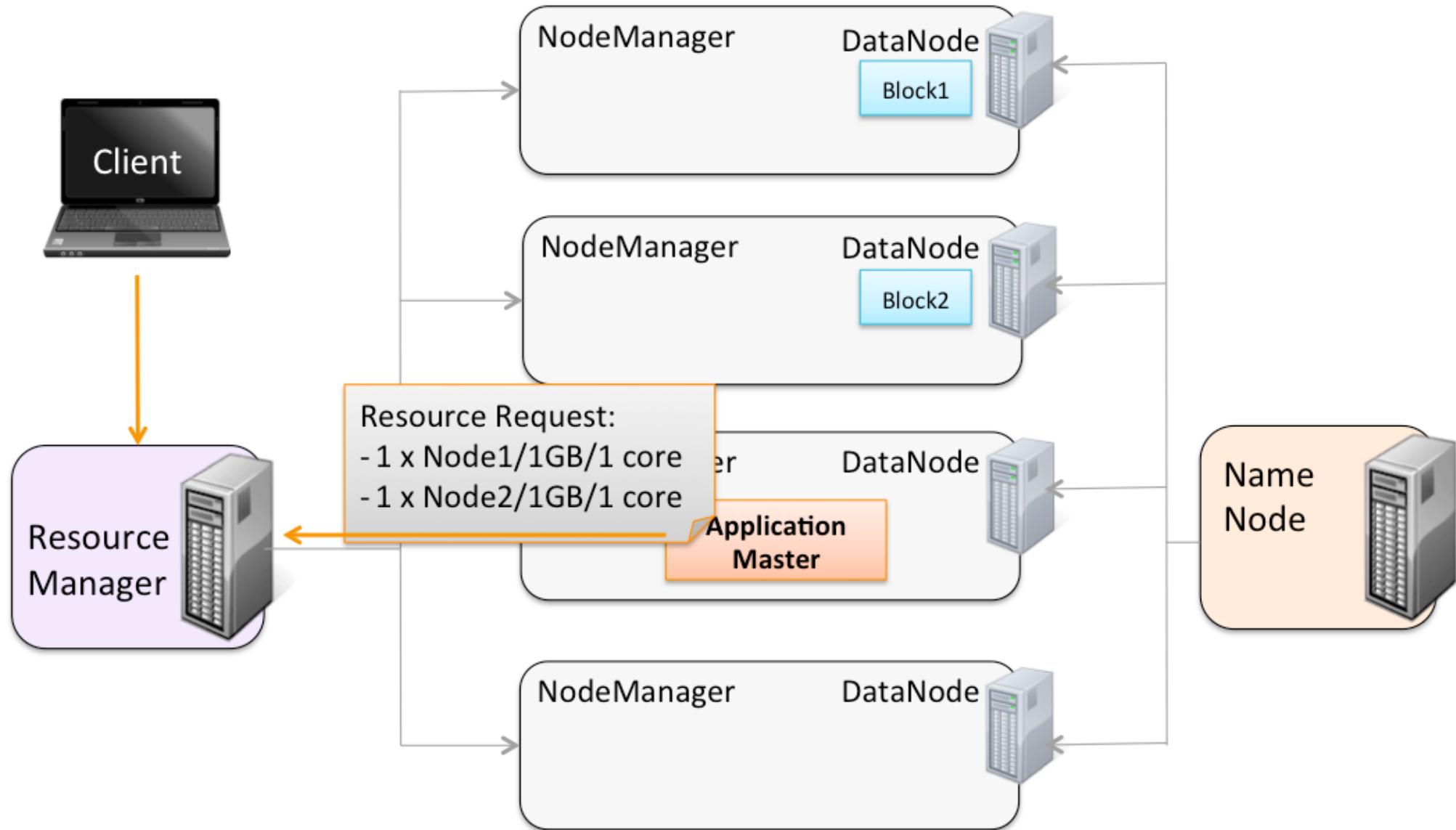
Yarn – yet another resource negotiator

- 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)

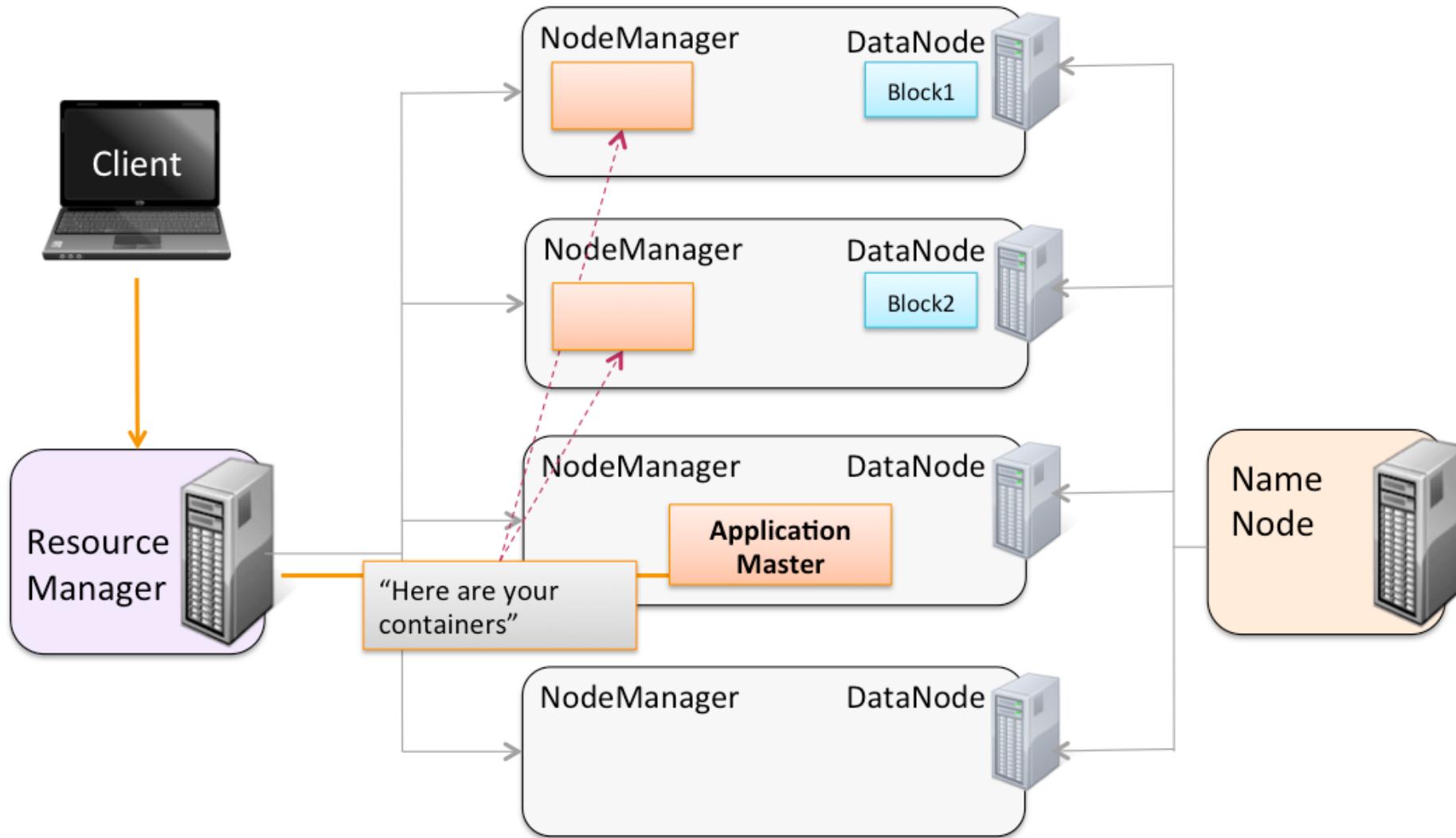
Running with yarn

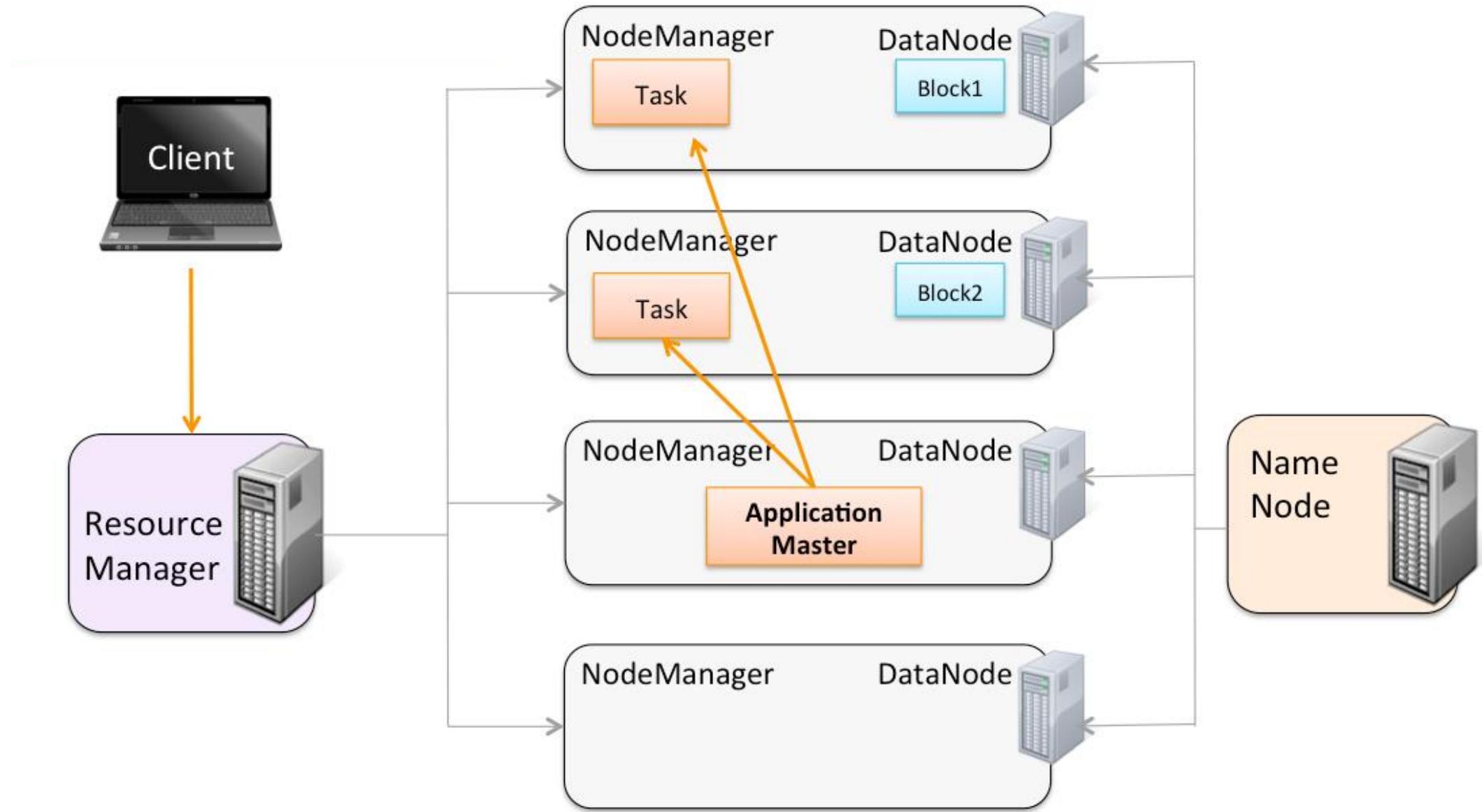


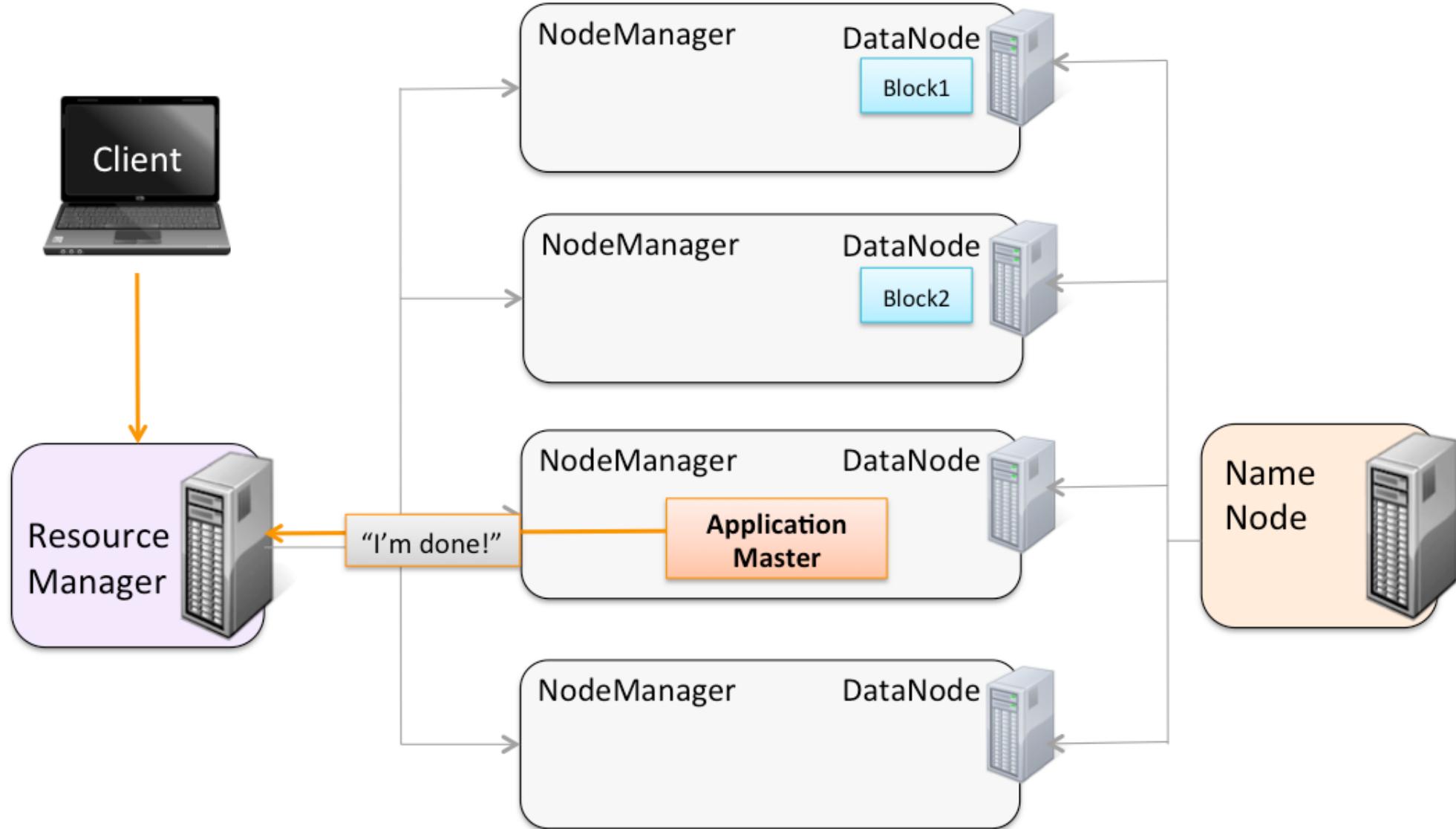




Will try to allocate the same machines which store the processed data





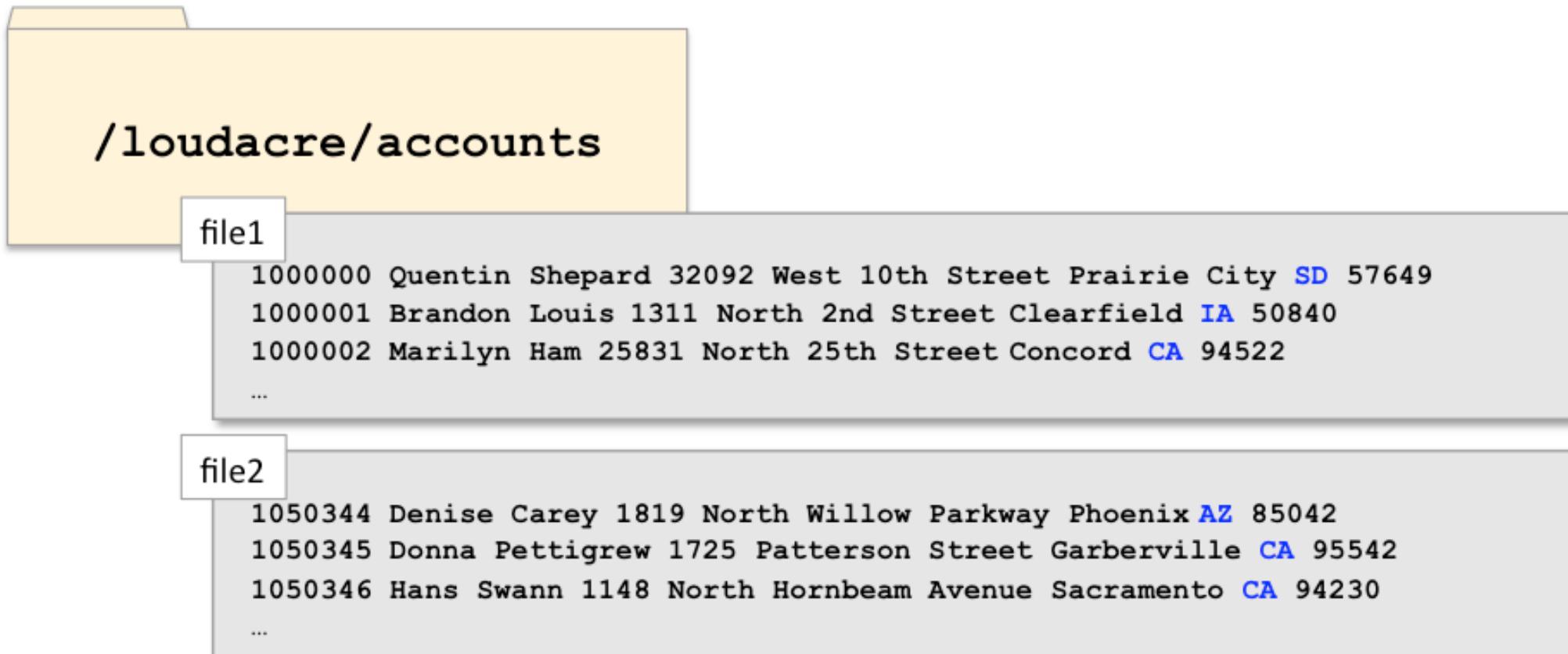


File formats you should know

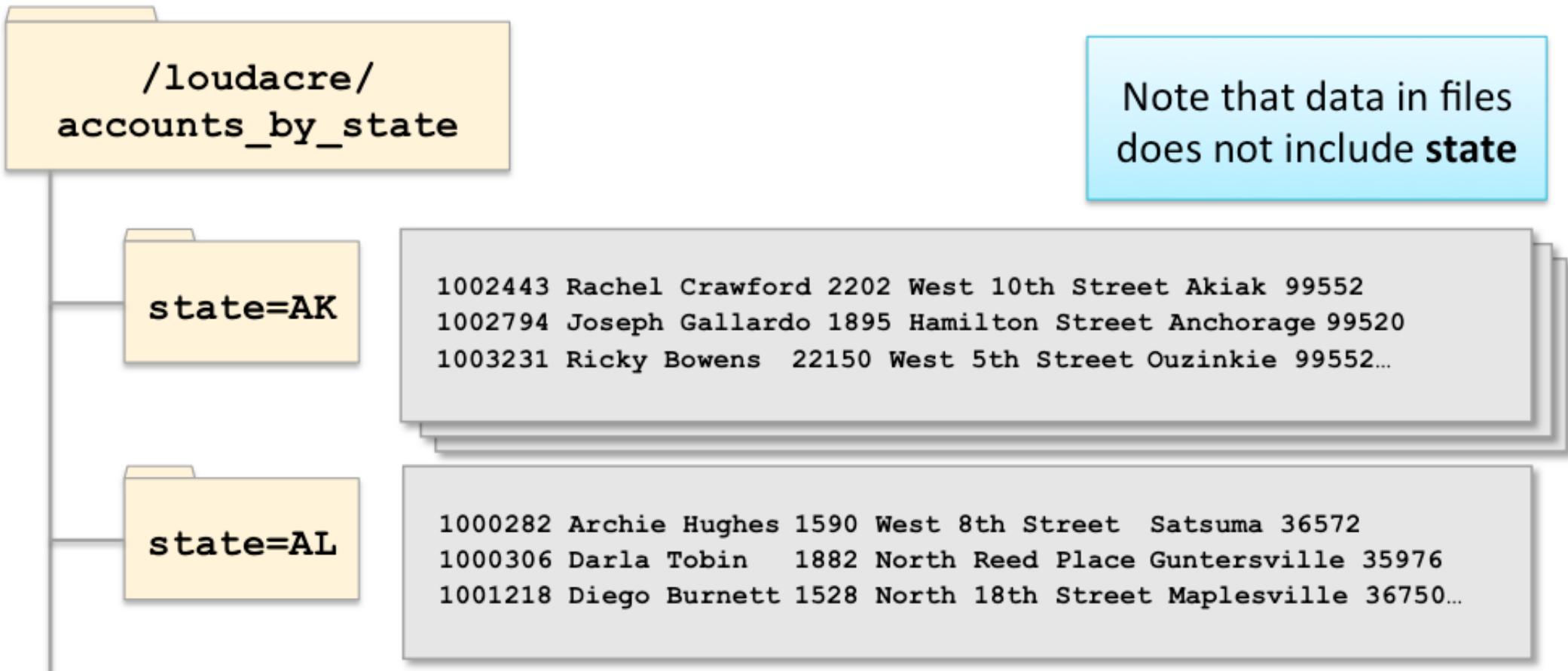
- parquet - like csv but compressed
- avro – like json, but compressed

Partitioning

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



More efficient



This is not my Job – sysadmin should know it



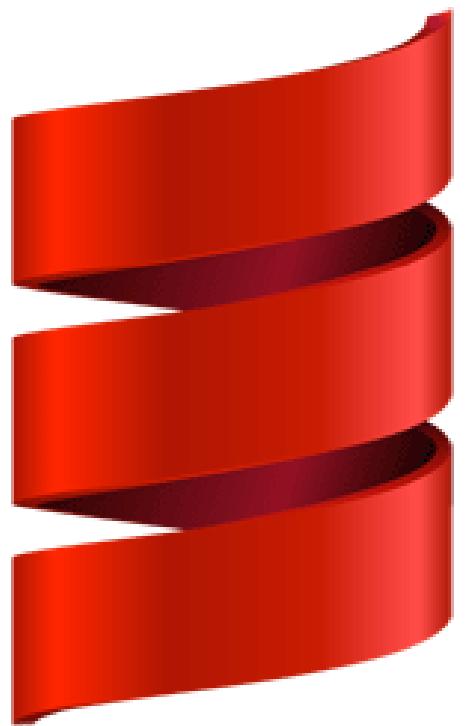
Шеф сказал: "ПОЧИНИТЬ!"...

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

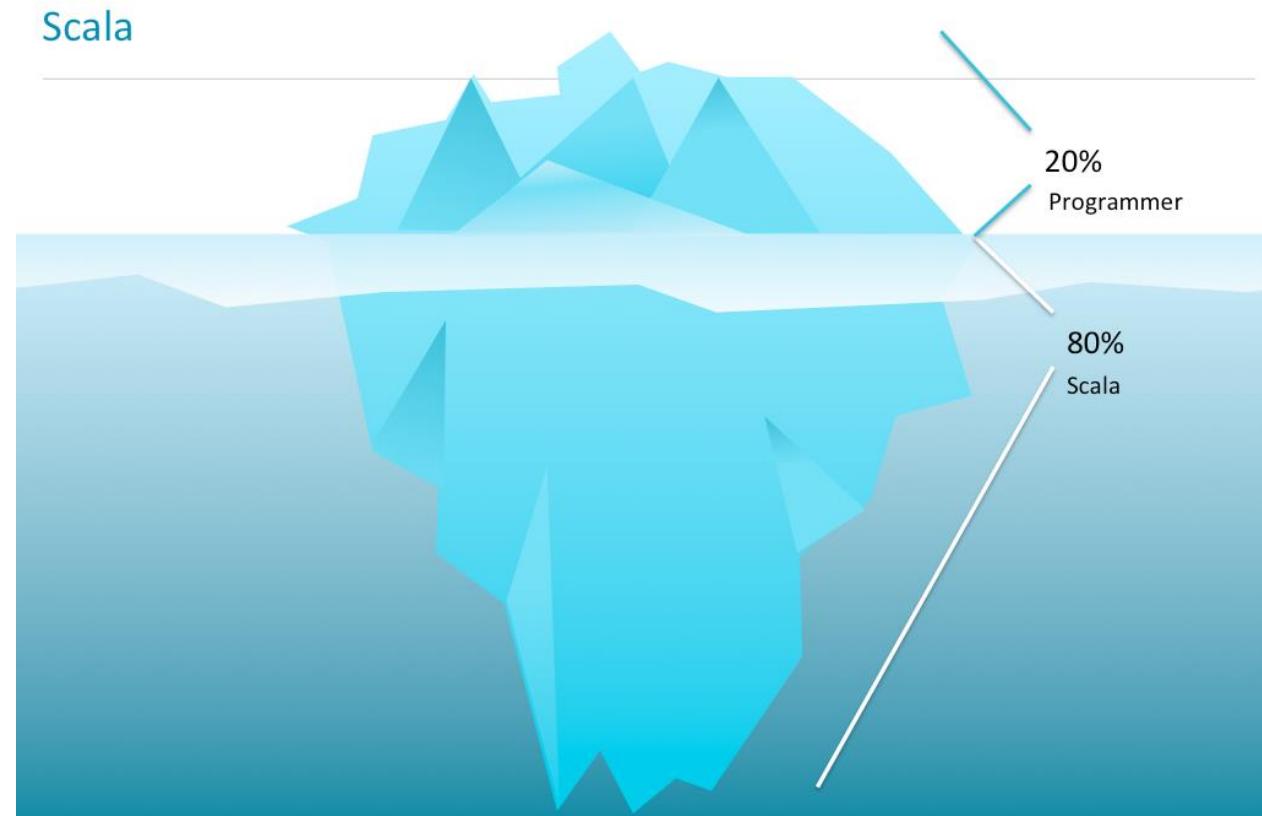
- Scala has features of imperative programming, object oriented programming, and functional programming, allowing the developer to cross paradigms within a single program as needed

Scala concepts

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

Scala & Java

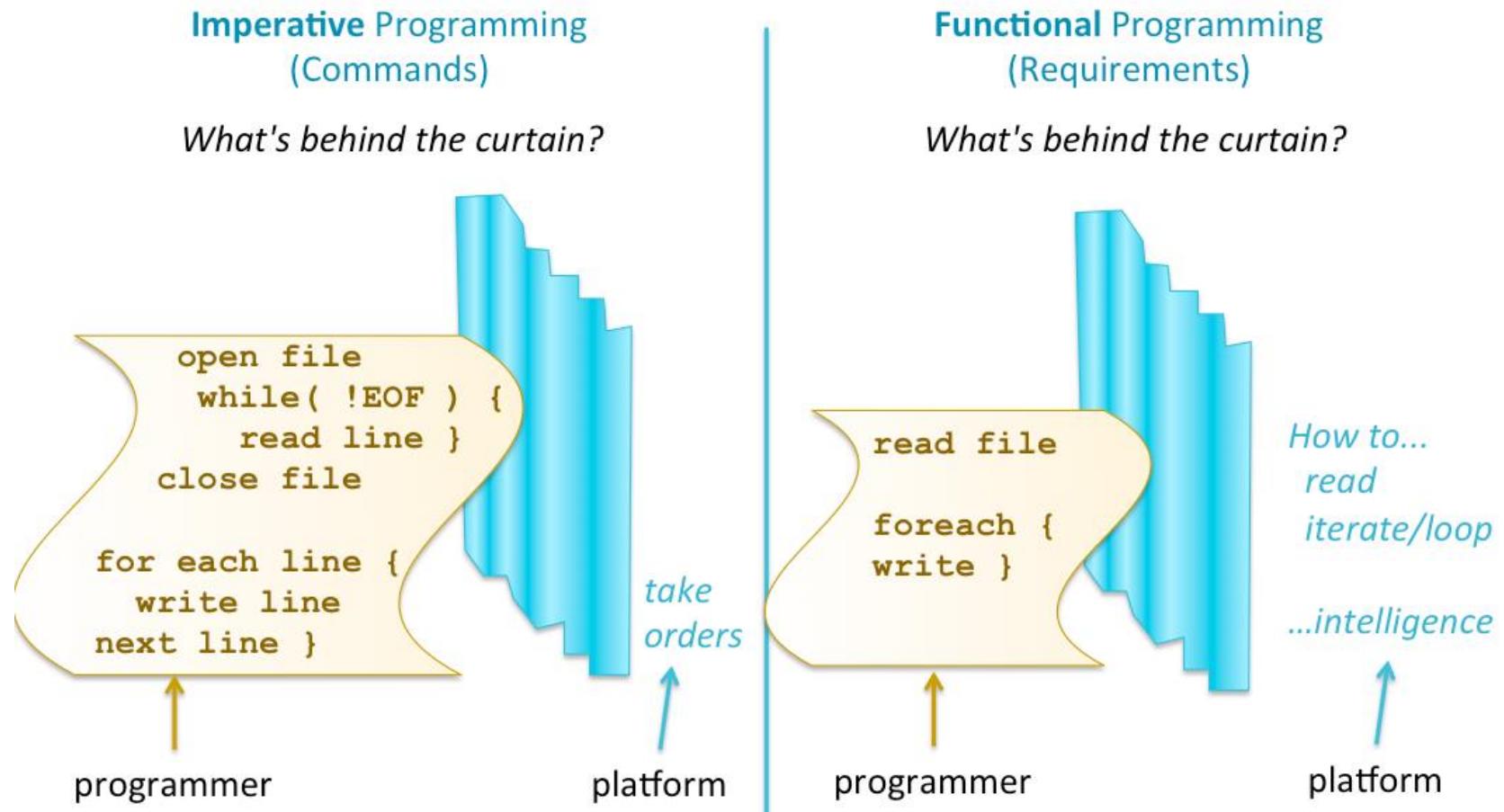
- All Java types are available
 - But in most cases the developers will use Scala types analogs
- Scala is expensive



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!

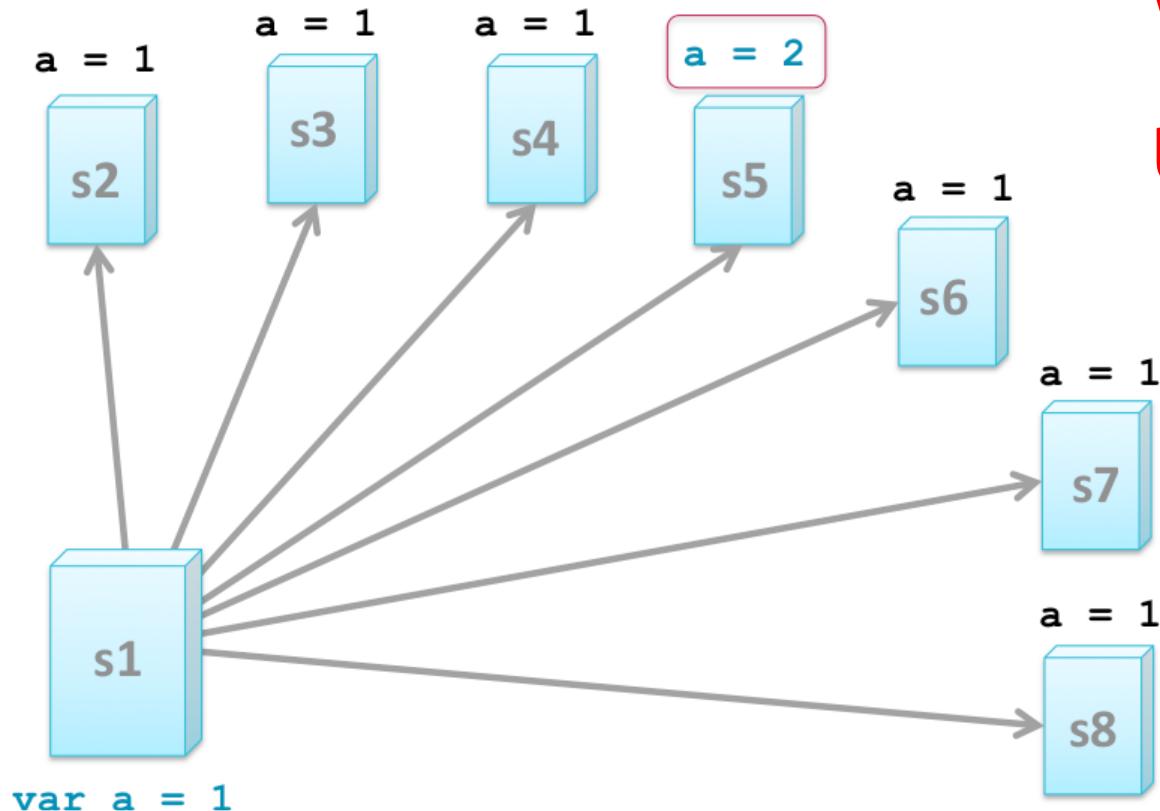


Immutable vs Mutable

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

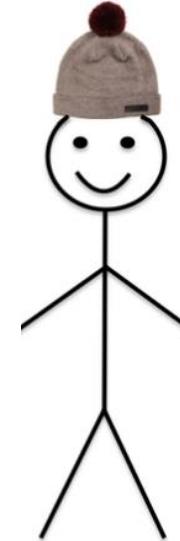
Special type - Unit

- 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
```

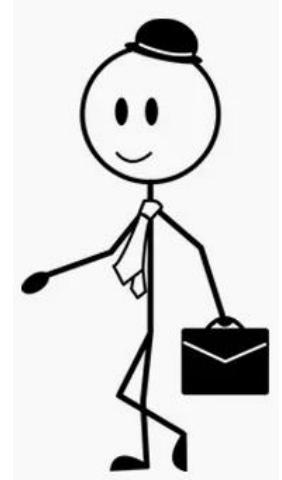
Just for information

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

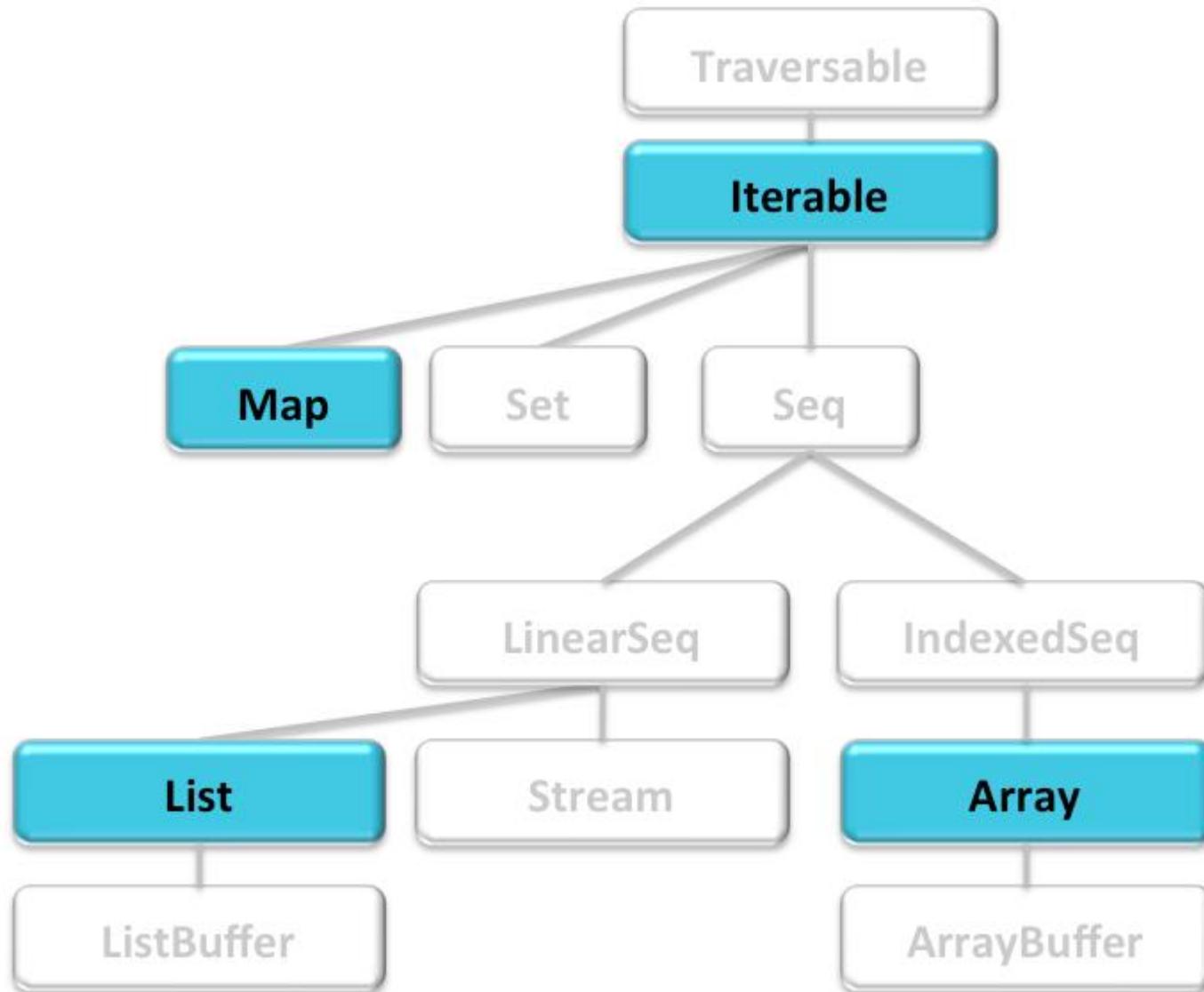
val endsWith: Boolean = str.sorted.toUpperCase().endsWith("la")

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

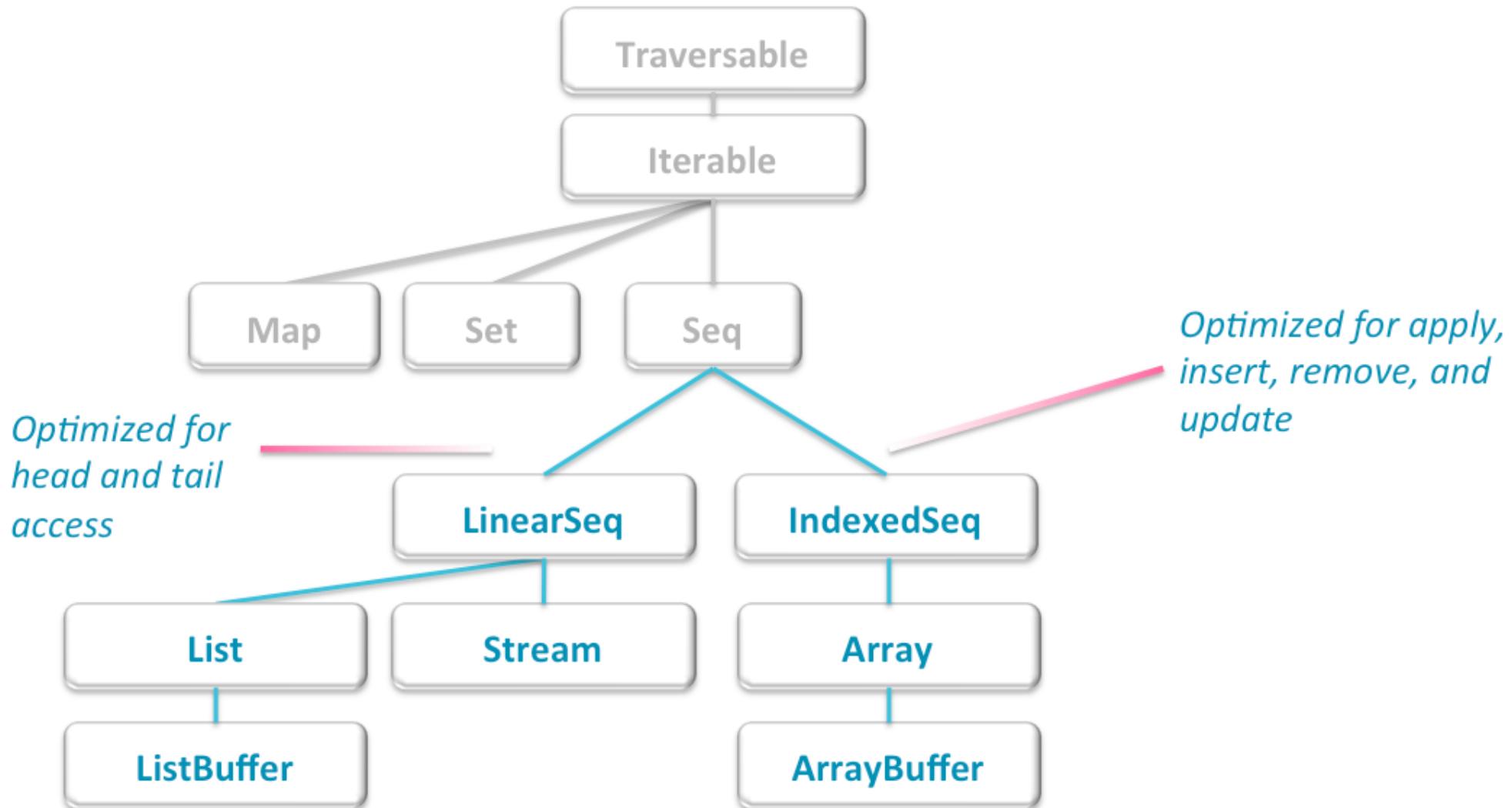
And now?

```
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)
```



Buffers are mutable versions – supporting insert, remove, append methods

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)
```

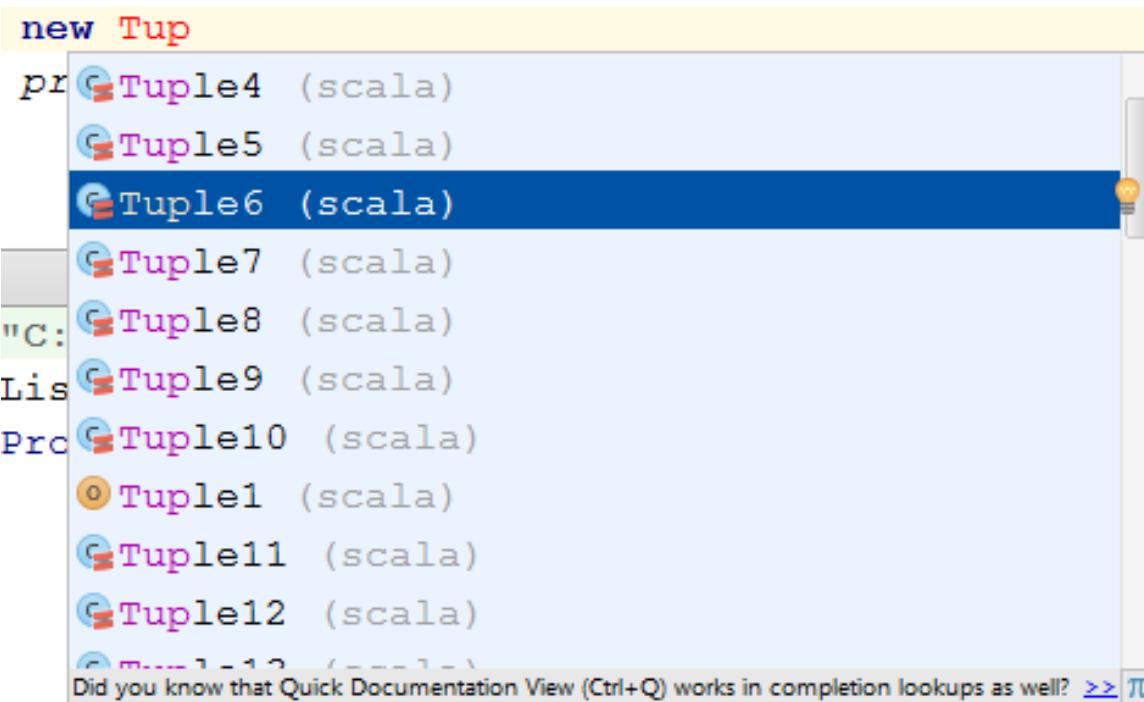
ONE

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`

Flow control

- While – like in Java, but you better use declarative approach

For

```
for(i <- 0 to sorrentoPhones.length - 1) {  
    println(sorrentoPhones(i))  
}  
  
for(i <- 0 until sorrentoPhones.length) {  
    println(sorrentoPhones(i))  
}  
  
for(i <- 0 until sorrentoPhones.length by 2) {  
    println(sorrentoPhones(i))  
}
```

For – with collections

```
for(model <- sorrentoPhones) {  
    print(model + " ")  
}
```

```
val phonebrands = List("iFruit", "MeToo", "Ronin")  
val newmodels = List("Z1", "Z-Pro", "Elite-Z")  
  
for( brand <- phonebrands; model <- newmodels ) {  
    print(brand + " " + model + ", ")  
}
```

What will do this code?

```
val sorrentoPhones =  
List("F00L", "F01L", "F10L", "F11L", "F20L", "F21L", "F22L", "  
F23L", "F24L", "F30L", "F31L", "F32L", "F33L", "F40L", "F41L")  
  
for( model <- sorrentoPhones ) {  
    if ( model.contains("2") ) print(model + " ")  
}
```

This is better

```
for( model <- sorrentoPhones; if( model.contains("2") ) {  
  print(model + " ")  
}
```

Lazy variables

- `def x = 1`
- `x` – will be declared like integer and value will be assigned to it, only when it will be used

You can do this shit

```
def listPhones {  
    println("MeToo")  
    println("Titanic")  
    println("iFruit")  
    println("Ronin")  
}  
> listPhones: Unit
```

```
listPhones  
> MeToo  
> Titanic  
> iFruit  
> Ronin
```

And this code can be written inside some method

Functions declarations

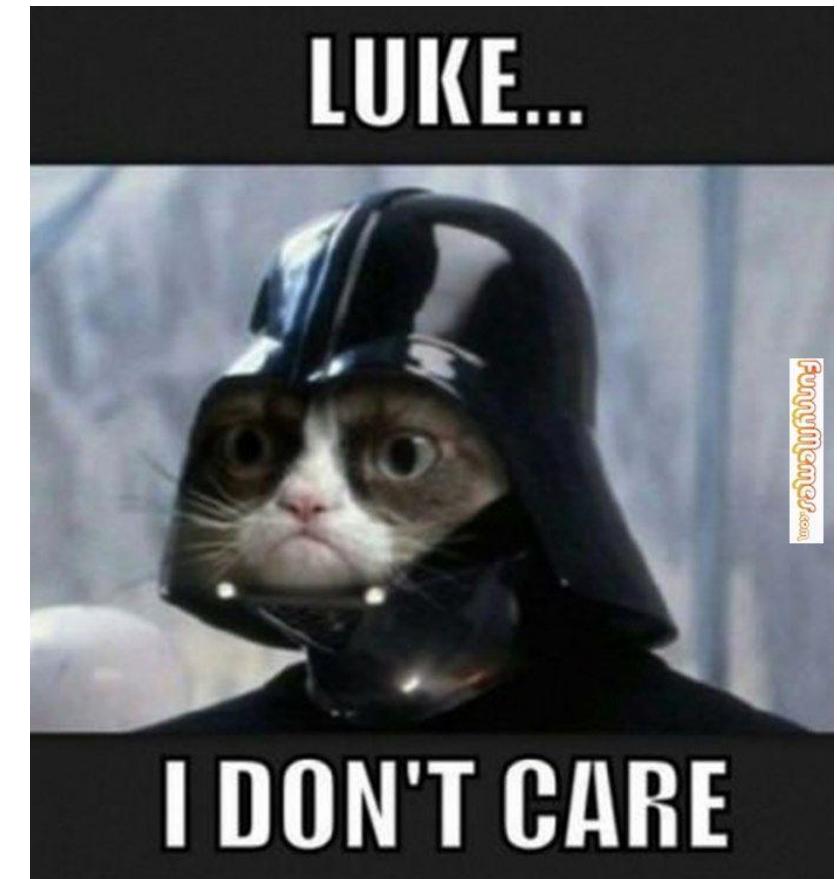
```
def CtoF(celsius: Double) = {  
    (celsius * 9/5) + 32  
}  
> CtoF: (celsius: Double)Double  
  
CtoF(34.0)  
> Double = 93.2  
  
def CtoF(celsius: Double) =  
    (celsius * 9/5 ) + 32  
  
def CtoF(celsius: Double) =  
    (celsius * 9/5 ) + 32 : Double
```

We don't care about braces

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

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

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



Functions as parameter

```
def main(args: Array[String]) {  
    def UA2USD(ua: Double) = ua * 25  
    def convertList(myList: List[Double], convert: (Double) => Double): Unit = {  
        for (n <- myList) {  
            println(n, convert(n))  
        }  
    }  
    val grivni: List[Double] = List(10, 20, 2)  
    convertList(grivni, UA2USD)  
}
```

(10.0, 250.0)

(20.0, 500.0)

(2.0, 50.0)

Anonymous functions as parameter

```
def main(args: Array[String]) {  
    def UA2USD(ua: Double) = ua * 25  
    def convertList(myList: List[Double], convert: (Double) => Double): Unit = {  
        for (n <- myList) {  
            println(n, convert(n))  
        }  
    }  
    val grivni: List[Double] = List(10, 20, 2)  
    convertList(grivni, x => x * 25)  
}
```

(10.0, 250.0)

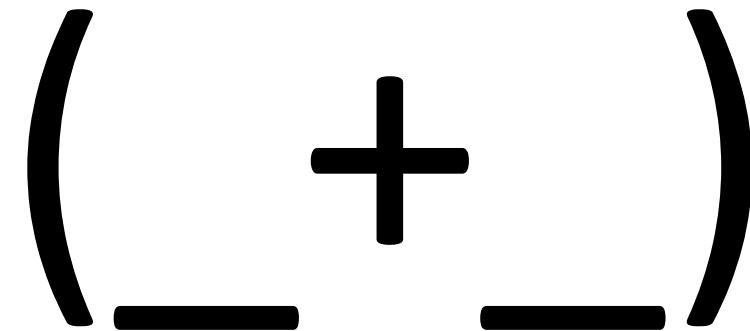
(20.0, 500.0)

(2.0, 50.0)

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)
```

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 = "shalom";  
        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 tunic, stands in a dimly lit, grand hall with large stone columns and a floor lined with small lights. She is looking upwards and slightly to her left with a surprised or questioning expression. In the background, a man with long brown hair, wearing a traditional grey toga, walks towards the camera. The scene has a dramatic, cinematic feel.

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 = "shalom";  
        case "how are you" => answer = "better than you";  
        case "hi" => answer = "hi!";  
        case _ => answer = "enough";  
    }  
    println(answer)  
}
```

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

Option

```
val superPhone = Some("Model 6")
> superPhone: Some[String] = Some(Model 6)
superPhone.getOrElse("Not found")
> String = Model 6

val superPhone = None
> superPhone: None.type = None
superPhone.getOrElse("Not found")
> String = Not found
```

Where Optional comes from?

```
def toTemp(in: String): Option[Double] = {
    try {
        Some(in.toDouble)
    } catch {
        case e: NumberFormatException => None
    }
}

val isATemp = "35.2"
val isNotATemp = "Warm"

toTemp(isATemp)
> Option[Double] = Some(35.2)
toTemp(isNotATemp)
> Option[Double] = None
```

All together

```
toTemp(isATemp) match {
  case Some(i) => println(i)
  case None     => println("Not a Temperature")
}

> 35.2

toTemp(isNotATemp) match {
  case Some(i) => println(i)
  case None     => println("Not a Temperature")
}

> Not A Temperature
```

Classes

- Like in Java (almost)
- Singletones



Regular class

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

def main(args: Array[String]) {
    val person: Person = new Person("Jeka")
    person.printDetails()
}
```

Can be shorter

```
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

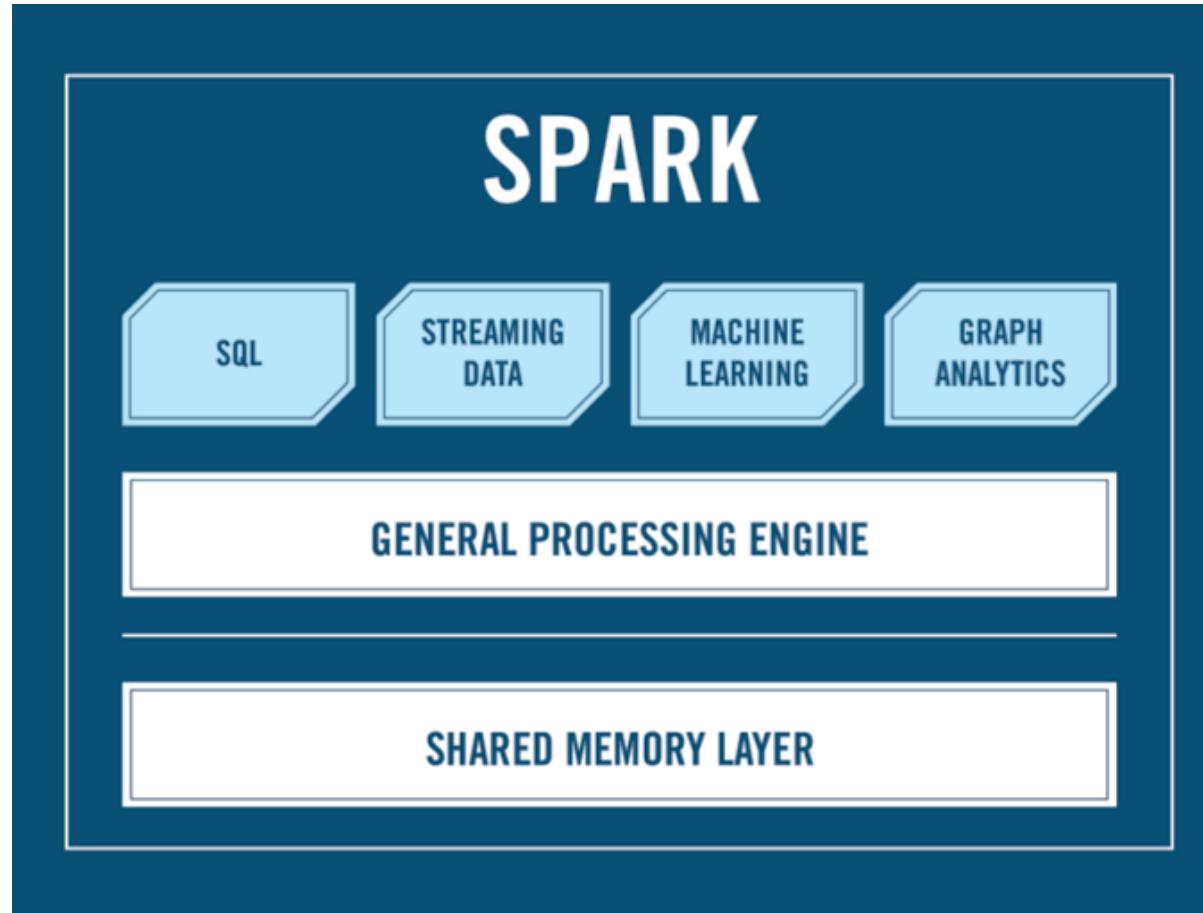
```
import pack1.Class1._
```

- 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**

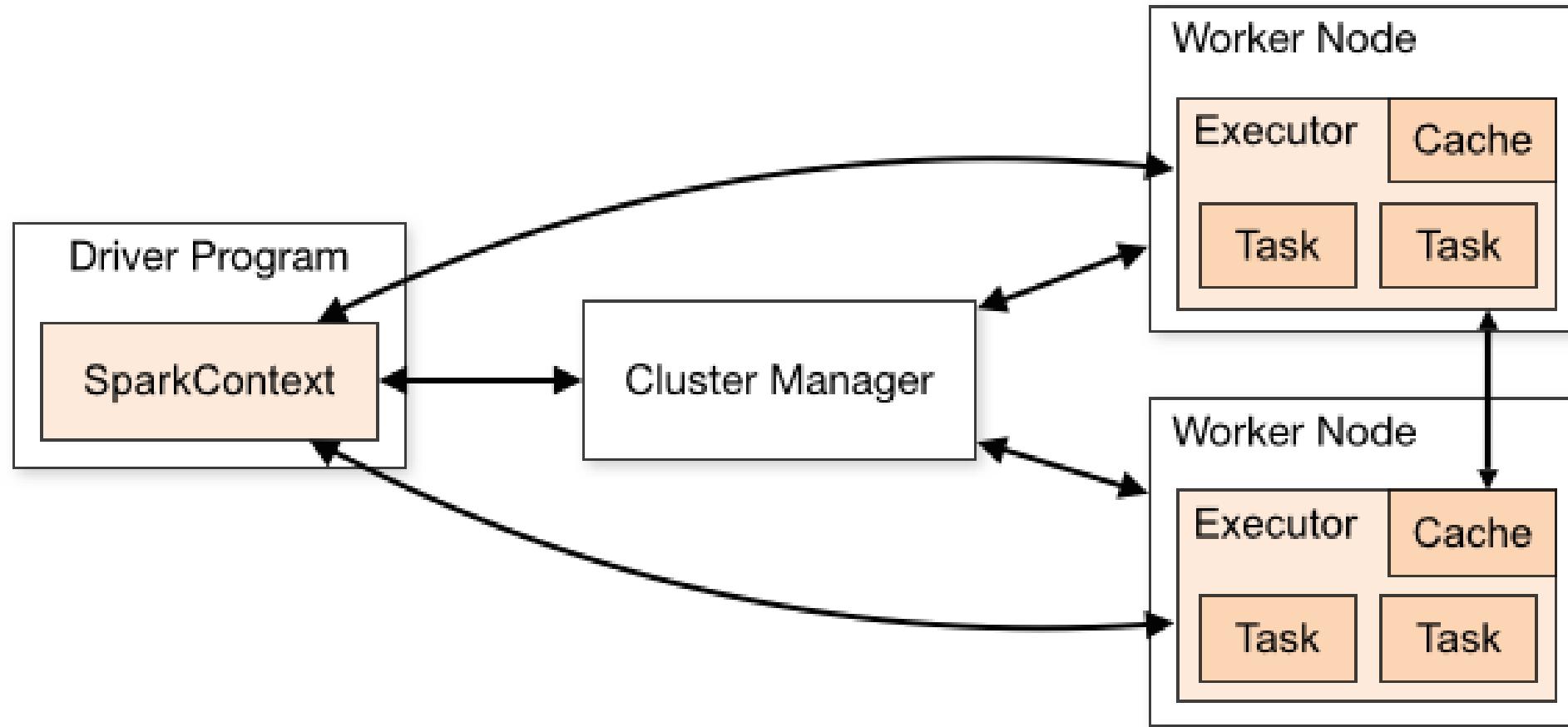
And now!



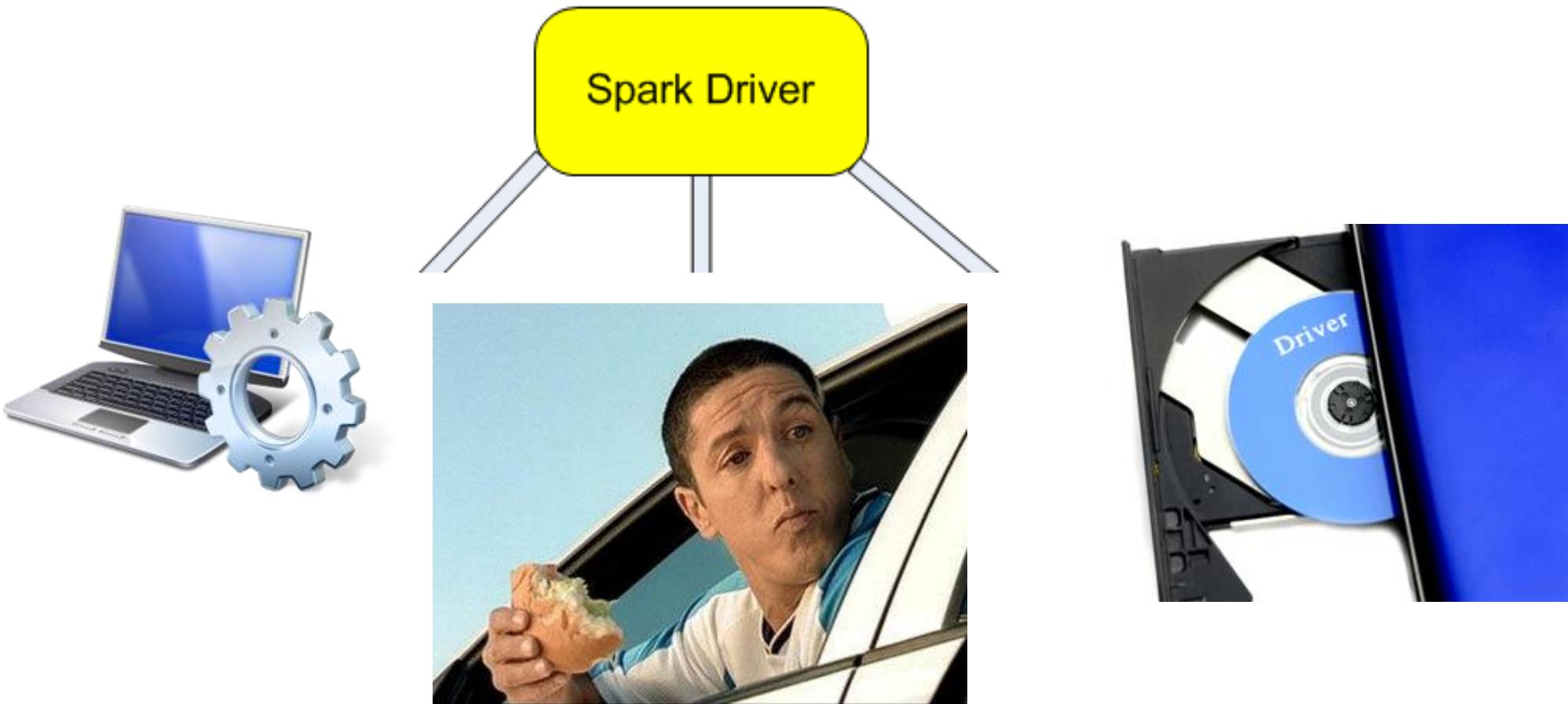
Spark

- Release
 - Was born in UC Berkeley in about 2009
 - First releases in 2012
 - Current version 2.3 several weeks ago
 - But still the most popular version is 2.2
- 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?



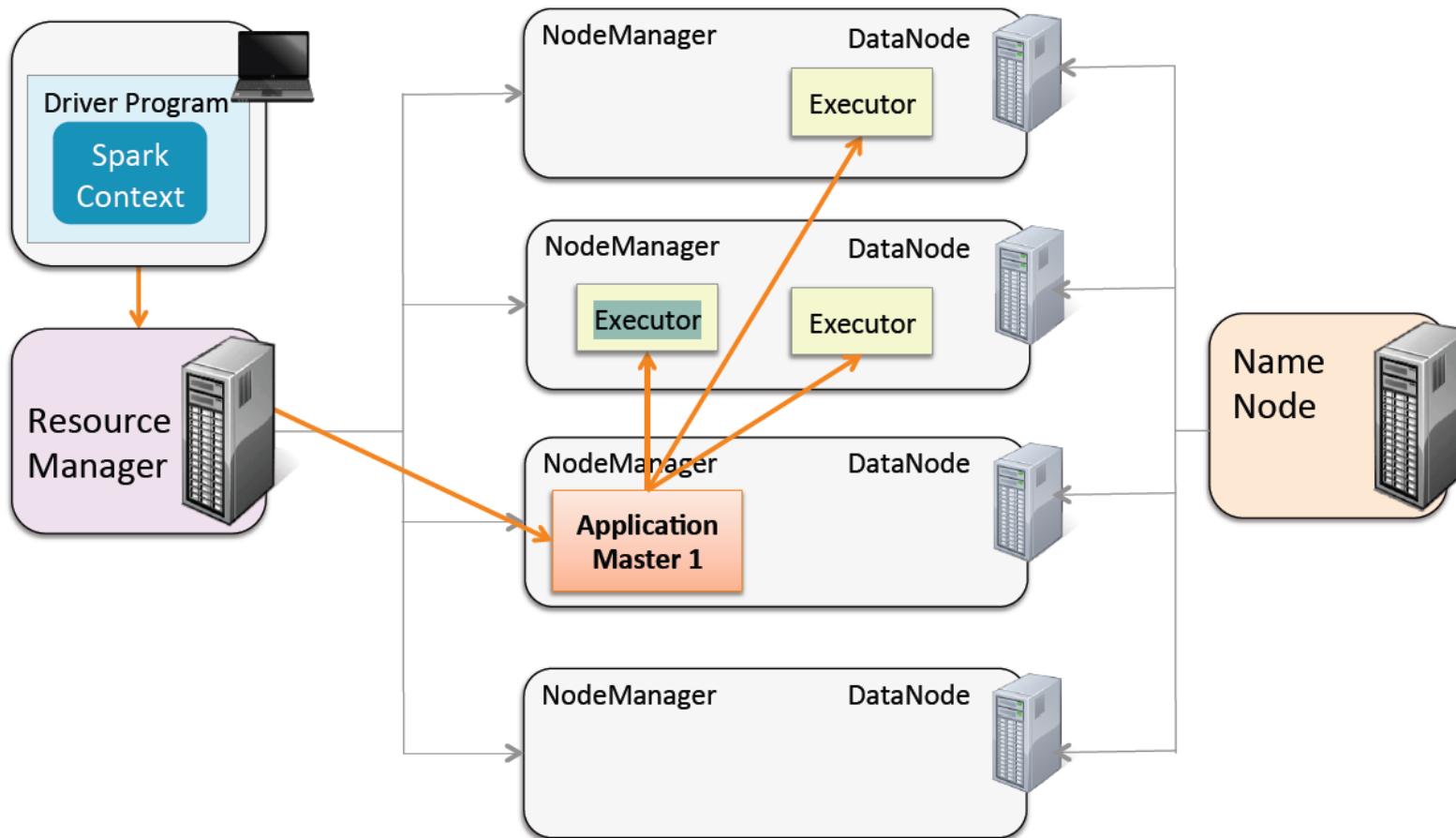
What is a driver?

- 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

```
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.parallelize([1, 2, 3])
```

What is sc?

- SparkContext
- Main entry point for Spark functionality

```
conf = SparkConf().setAppName('appName').setMaster('local')
sc = SparkContext(conf=conf)
```

```
sc = SparkContext("local", "test")
```

RDD has two kind of methods

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

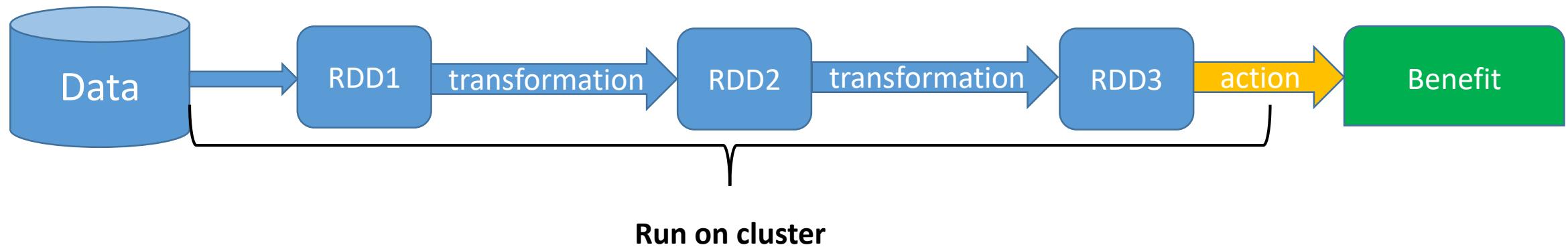
Transformations

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

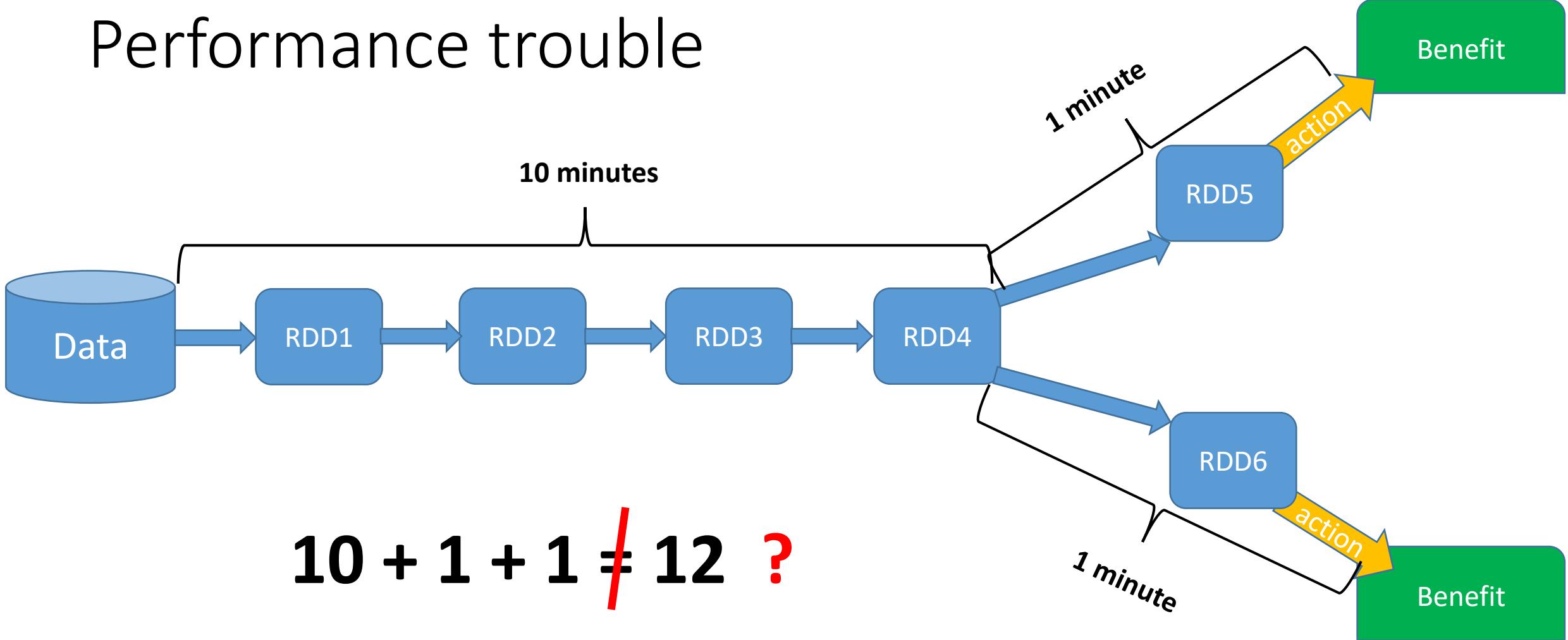
Actions

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

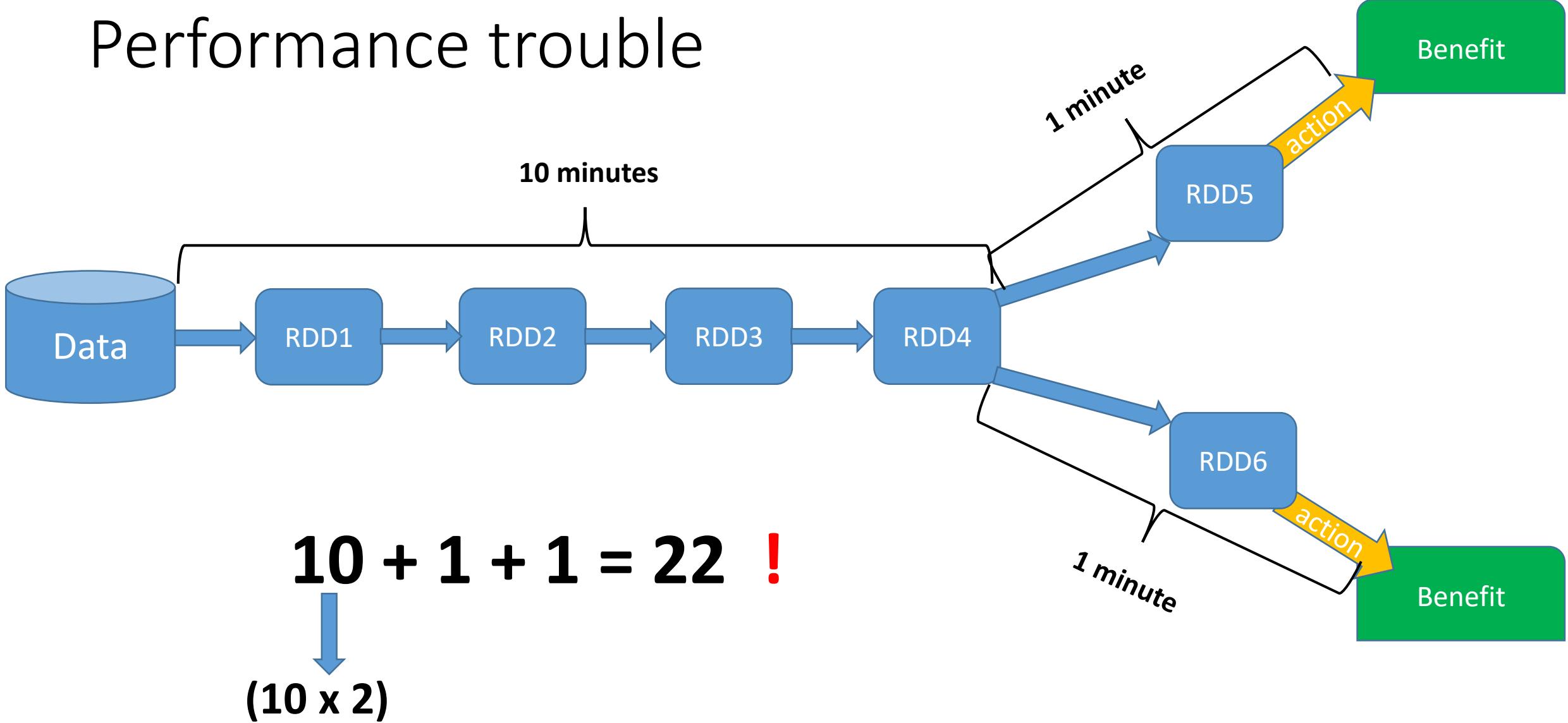
Rdd flow



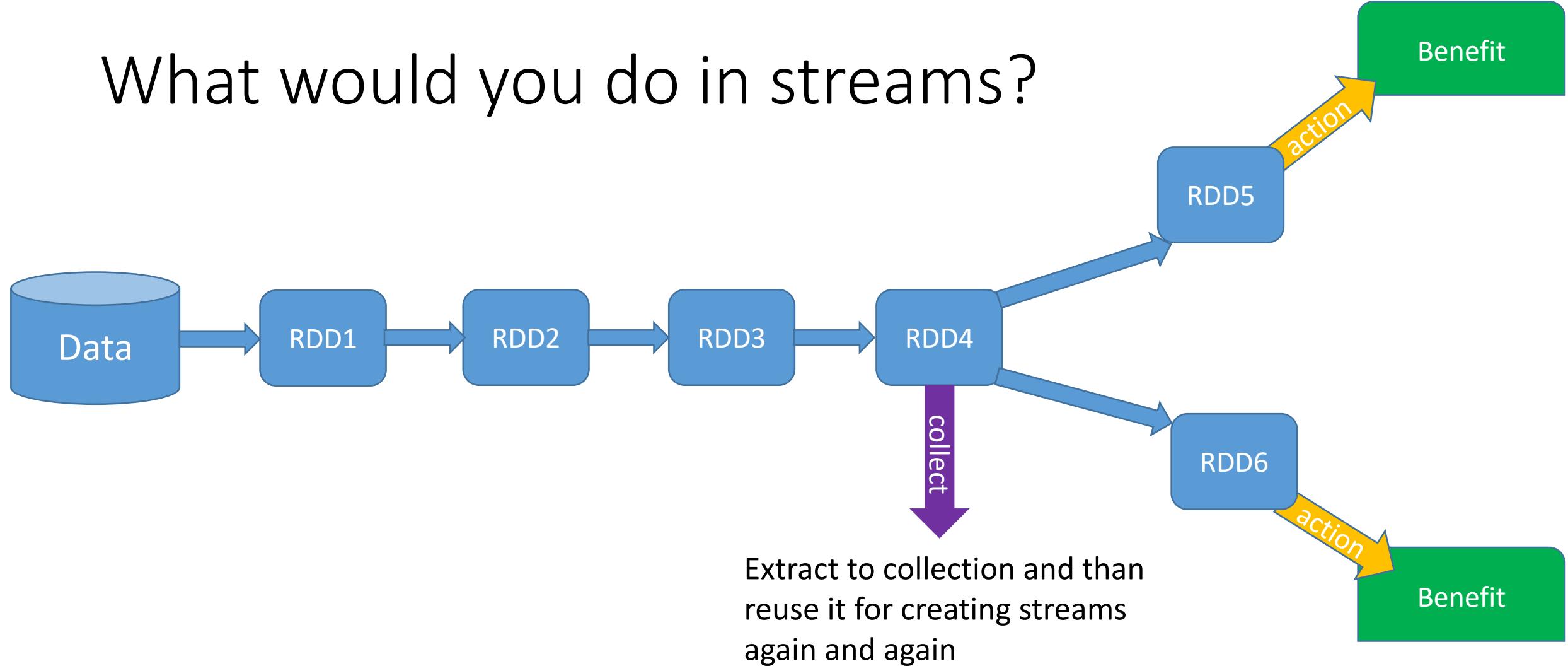
Performance trouble



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.

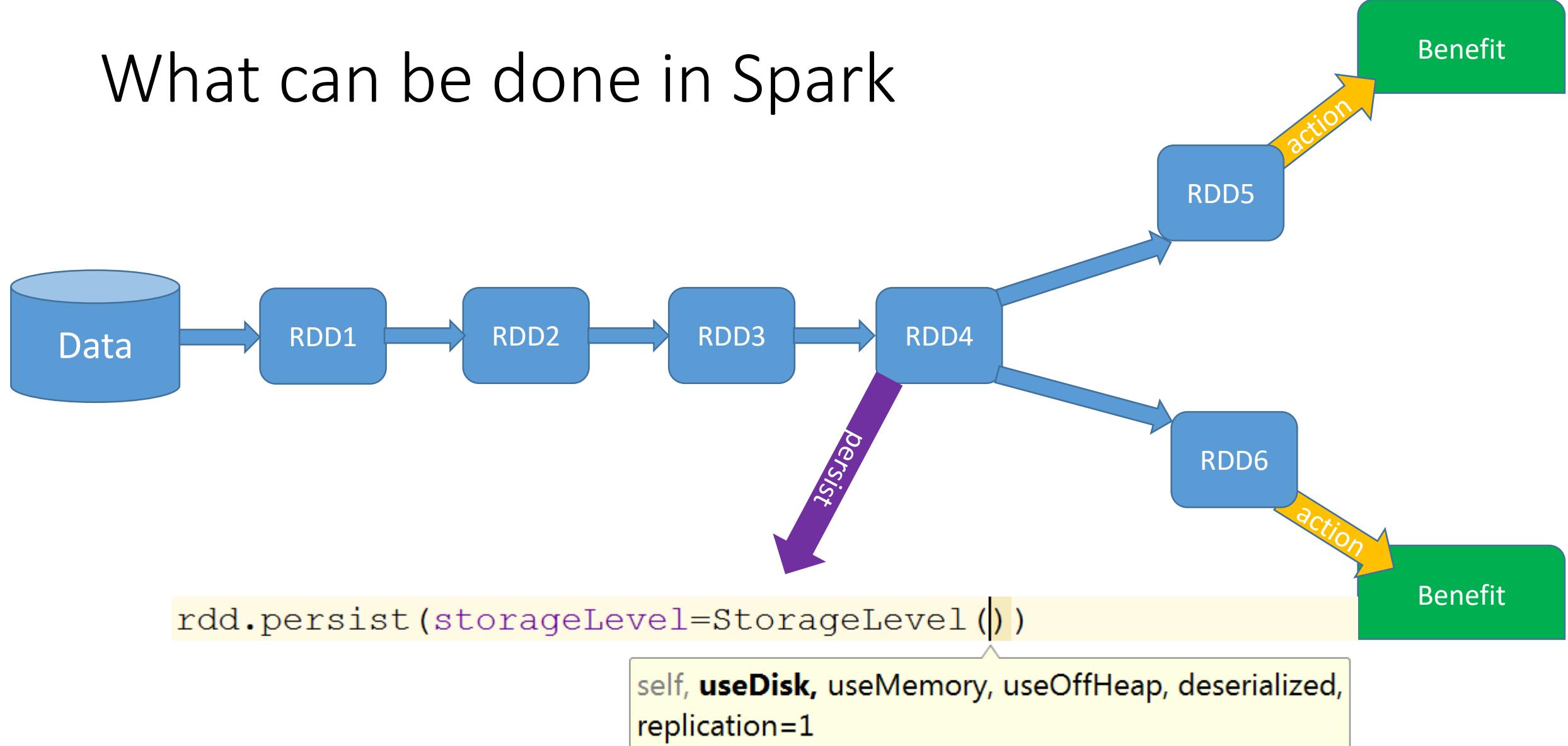
ONE DOES NOT SIMPLY

CAN WRITE EFFECTIVE SPARK CODE

You need to know when use

Persist

What can be done in Spark



Let's start writing spark

Lab - I'll send you file like this one

File content:

104 Boston 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
- Find all useless drivers (which have 0 km trips) and print their names
- Write names of 3 drivers with max total kilometers (sort top to down)
bonus: the same but only for today
- Count number of 0 km trips per each driver, sort it and print
- Find cities where 0 trips happen most

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

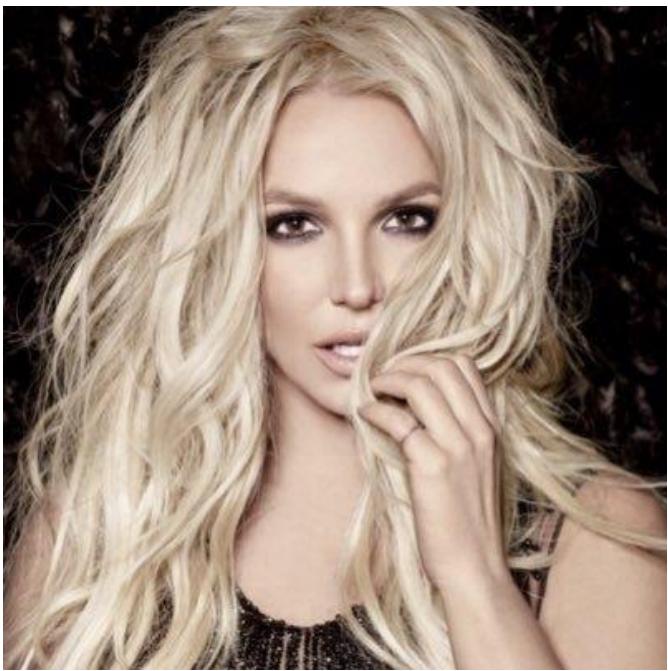
CAN WRITE EFFECTIVE SPARK CODE

Accumulator

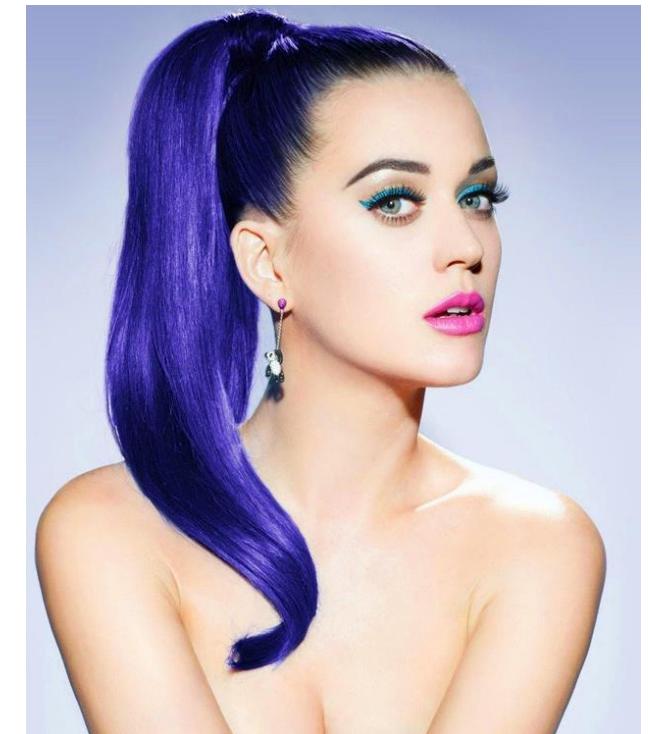
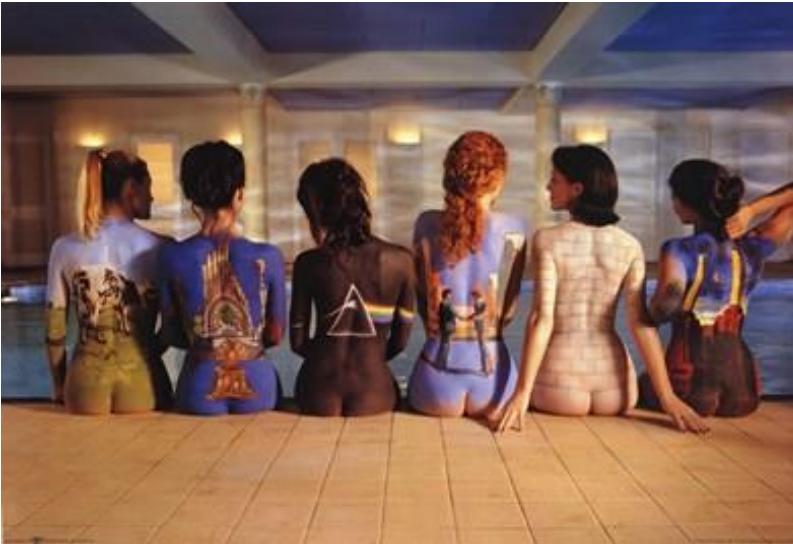
- Accumulator will be delivered to all executors, and will accumulate some value, when accumulator will be grabbed to driver all values will be summarized.

```
accum = sc.accumulator(0, "myAcc")
accum += 1
total = accum.value
```

More interesting lab:



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



Another lab

- Write function which will receive path to text file and variable x.
- Function should find x most popular words in the file
- your test should check, that word ‘yesterday’ is the most popular in song Yesterday of Beatles

You need to filter out all garbage words



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

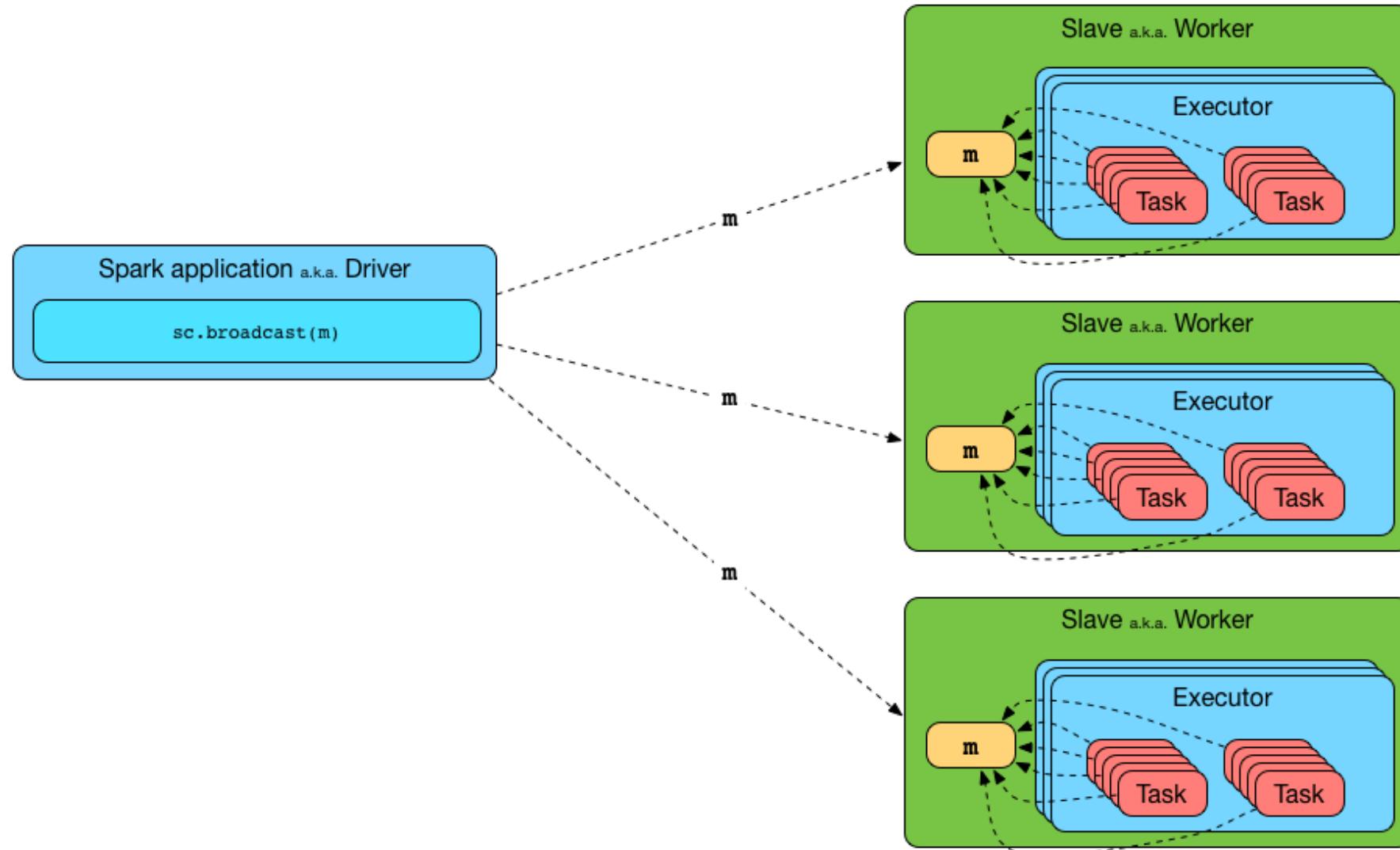
CAN WRITE EFFECTIVE SPARK CODE

You need to know about

Broadcast

```
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

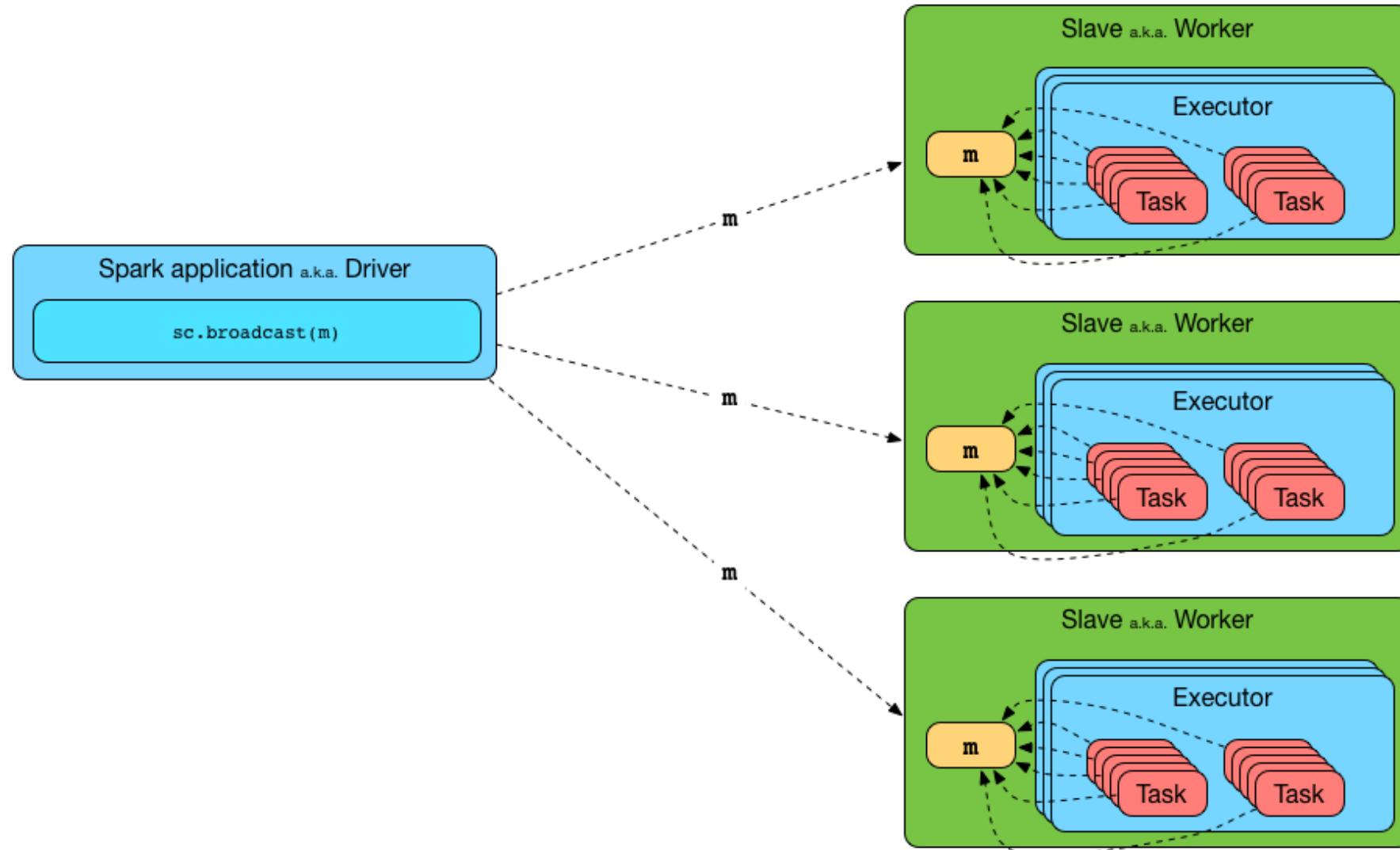
Example

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

Data which should be broadcasted

Table of phone prefixes against operators
List of chosen countries

Broadcast



DataFrames/Datasets

Dataframes – since 1.3

- DataFrame/datasets are 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 SparkSession

How to create SparkSession?

```
from pyspark.sql import SparkSession

sparkSession = SparkSession \
    .builder \
    .appName("Python Spark SQL basic example") \
    .config("spark.some.config.option", "some-value") \
    .getOrCreate()

sparkSession = SparkSession(sc)
```

Create DataFrame from json file

```
sparkSession.read().json("data/*.json")
```

Dataframes methods and functions

Agg, columns, count, distinct, drop, filter

groupBy, orderBy, registerTable, schema, show, select, where,
withColumn

Some dataframes methods

Agg, columns, count, distinct, drop, filter

groupBy, orderBy, **registerTable**, schema, show, select, where,
withColumn

```
dataFrame.registerTempTable("myTable")

DataFrame frame = sparkSession.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 myTable 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")
```

Don't tell him about this



Why I don't love sql

```
DataFrame dataFrame1 = sparkSession.sql("select tdmco.CL_ID as cl_id, \n" +
"\t\tdmdcl.CL_GRP_ID as cl_grp_id, \n" +
"\t\ttdpet.DK_PET as dk_pet, \n" +
"\t\ttdsgt.DK_SGT as dk_sgt, \n" +
"\t\ttdsgs.DK_SGS as dk_sgs, \n" +
"\t\ttddir.DK_DIR as dk_dir, \n" +
"\t\ttdorg.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\ttddat.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) 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) as sbs_prf_if,\n" +
"\t\tdata.STATUS as status\n" +
"\t\tfirst_value(tmdbspref.PRF_ID) OVER (PARTITION BY tdmco.CL_ID,data.IMSI ORDER BY length(tmdbspref.MATCH_STR) desc,tmdbspref.MATCH_START) AS sbs_prf_if,\n" +
"from data left join tdmco on (data.OWNER_CODE=tdmco.MCO_TAP_CD)\n" +
"left join tdmcl on (tdmco.CL_ID=tdmcl.CL_ID)\n" +
"left join tdpet 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 tdsgs on (case when data.STATUS = 'OK' then 'S' when data.STATUS = 'Reject' then 'F' end = tdsgs.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 tmdbspref on (tdmco.CL_ID = tmdbspref.org_tech_id and SUBSTR(data.IMSI, LENGTH(tmdbspref.MATCH_START), LENGTH(tmdbspref.MATCH_STR)) = tmdbspref.MATCH_STR and to_date(data.TIME) = tmdbspref.TIM))
```

functions:

```
import pyspark.sql.functions as func
```

- sin, cos,tan,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
- Arithmetic operators also can be used

How to apply to column?

- `dataset['column_name']`
 - `dataset.column_name`
 - In some cases: `'column_name'`
 - Example
-
- `dataset = dataset.withColumn('sin_from_age', func.sin(dataset['age']))`
 - `dataset = dataset.withColumn('sin_from_age', func.sin(dataset.age))`
 - `dataset = dataset.withColumn('sin_from_age', func.sin('age'))`

What collect, take or head will return?

- Row object

```
First_col_val = df2.head() [0]
```

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 all column names
- Add column “salary” the value will be calculated by formula:
age * number of technologies * 10 (bonus, instead of 10, multiply by 5 if age < 5 or by 10 if age \geq 5)
- Find developers with salary < 1200 and which familiar with most popular technology

Udf function

```
two_col_sum = udf(lambda x, y: x + y, IntegerType())  
# integerType can be omitted  
ds = ds.withColumn('new_col', two_col_sum(ds['col1'], ds['col2']))
```

How to create Dataset from CSV

First you need to create regular RDD from CSV file as always, using SparkContext

```
dataset = sparkSession.createDataFrame(rdd, schema=['line'])  
dataset.show(10)
```

Another example: create Dataset from CSV

```
word_pairs_rdd = sc\  
    .textFile(getcwd() + "\\data\\songs\\britney\\pink_floyd.txt") \  
    .flatMap(lambda line: line.split()) \  
    .map(lambda word: (word.lower(), 1)) \  
    .persist()  
  
word_pairs_df = ss.createDataFrame(word_pairs_rdd, schema=['word', 'count'])  
word_pairs_df.show(10)
```

Creating typed schema

```
from pyspark.sql.types import *
schema = StructType([
    StructField("name", StringType(), True),
    StructField("age", IntegerType(), True) ])
df3 = spark.createDataFrame(rdd, schema)
row = df3.take(1)
#row is: [Row(name=u'Alice', age=1)]
```



The logo for Spark Streaming features the word "Spark" in a bold, black, sans-serif font. A small, orange five-pointed star is positioned above the letter "k". Below "Spark", the word "Streaming" is written in a larger, bold, italicized, black, sans-serif font. The entire logo is enclosed within a thin, rounded rectangular border.

Spark

Streaming

What is Spark Streaming?

- Receive data streams from input sources, process them in a cluster, push out to databases/ dashboards
- Scalable, fault-tolerant, second-scale latencies



How does Spark Streaming work?

- Chop up data streams into batches of few secs
- Spark treats each batch of data as RDDs and processes them using RDD operations
- Processed results are pushed out in batches



Spark Streaming Programming Model

- Discretized Stream (DStream)
 - ◆ Represents a stream of data
 - ◆ Implemented as a sequence of RDDs

- DStreams API very similar to RDD API
 - ◆ Functional APIs in Scala, Java, Python
 - ◆ Create input DStreams from different sources
 - ◆ Apply parallel operations

Example - Network word count

Run netcat

```
nc -l -p 9999
```

If run under pyspark we already have SparkContext

```
#sc = SparkContext("Local[2]", "NetworkWordCount")
```

```
ssc = StreamingContext(sc, 5)
```

```
lines = ssc.socketTextStream("localhost", 9999)
```

```
words = lines.flatMap(lambda line: line.split(" "))
```

```
pairs = words.map(lambda word: (word, 1))
```

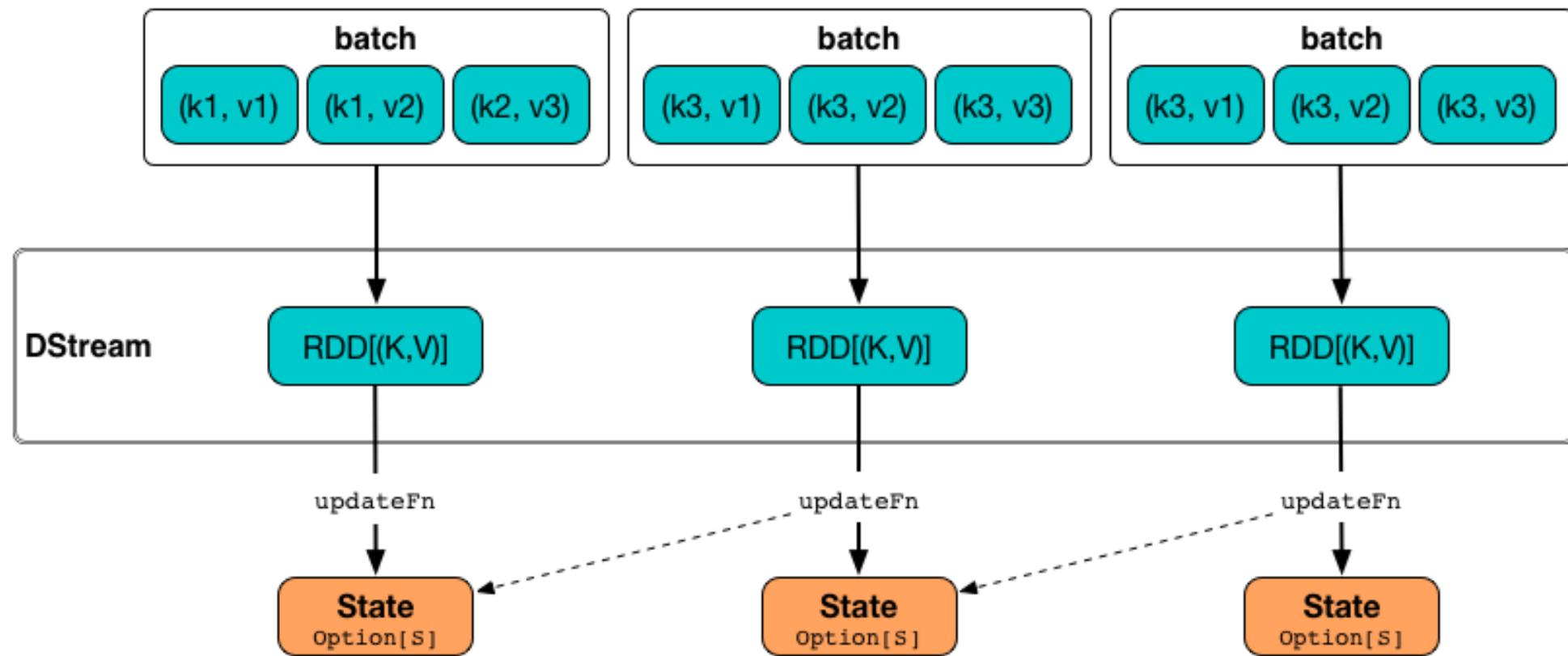
```
wordCounts = pairs.reduceByKey(lambda x, y: x + y)
```

```
wordCounts.pprint()
```

```
ssc.start()
```

```
#if you were to start a streaming context like this in a standalone application, while the  
#receiver threads would be started, the main thread of your driver would simply exit, unless you
```

Saving computation state over time

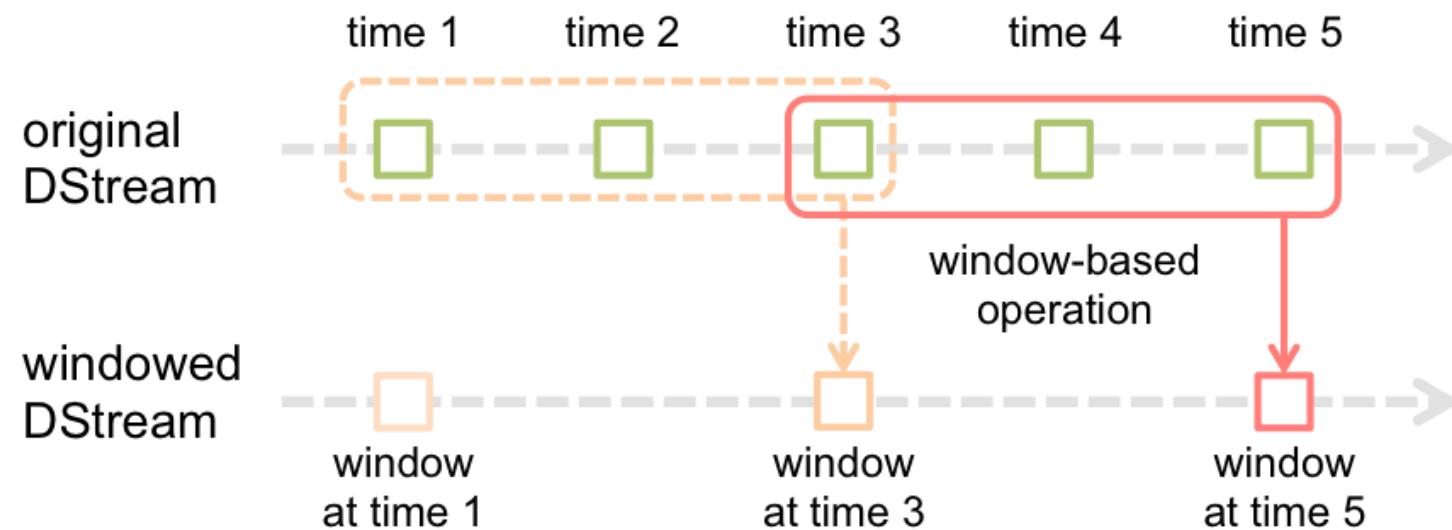


Window Operations

"Batch interval" is the basic interval at which the system will receive the data in batches. This is the interval set when creating a StreamingContext. For example, if you set the batch interval as 1 second, then any input DStream will generate RDDs of received data at 1 second intervals.

A window operator is defined by two parameters -

- WindowDuration - the length of the window
- SlideDuration - the interval at which the window will slide or move forward



Example - Kafka

Run kafka server

Go to folder with kafka

```
bin\windows\zookeeper-server-start.bat config/zookeeper.properties  
bin\windows\kafka-server-start.bat config/server.properties  
bin\windows\kafka-topics.bat --create --zookeeper localhost:2181 --replication-factor 1 --  
partitions 1 --topic orders  
bin\windows\kafka-topics.bat --create --zookeeper localhost:2181 --replication-factor 1 --  
partitions 1 --topic metrics
```

Check kafka with producer and consumer

```
bin\windows\kafka-console-producer --broker-list localhost:9092 --topic orders
```

```
bin\windows\kafka-console-consumer --zookeeper localhost:2181 --topic orders
```

Run both scripts, you should see messages from producer console in consumer console

Run kafka_producer.bat. Data file should be at the same folder

Run examples... Result should be at the output folder on disk C: