Big Data Processing : Using Spark

Mostafa Alaa Mohamed
Senior Big Data Engineer
Email: mustafaalaa.mohamed@gmail.com
Linkedin: Mostafa Alaa

Big Data & Analytics Department, Etisalat UAE

February 5, 2017

Resilient Distributed Datasets (RDDs)

- **Resilient Distributed Datasets (RDDs)**
  - **Resilient**: Automatically rebuilt on failure
  - **Distributed**: Means that transformations is done over collections of *immutable objects* that can be stored in memory or disk in multi-nodes across a cluster via parallel transformations (map, filter, . . .)
  - **Datasets** Initial data can come from a file or be created pragmatically
- RDDs are the fundamental unit of data in Spark.

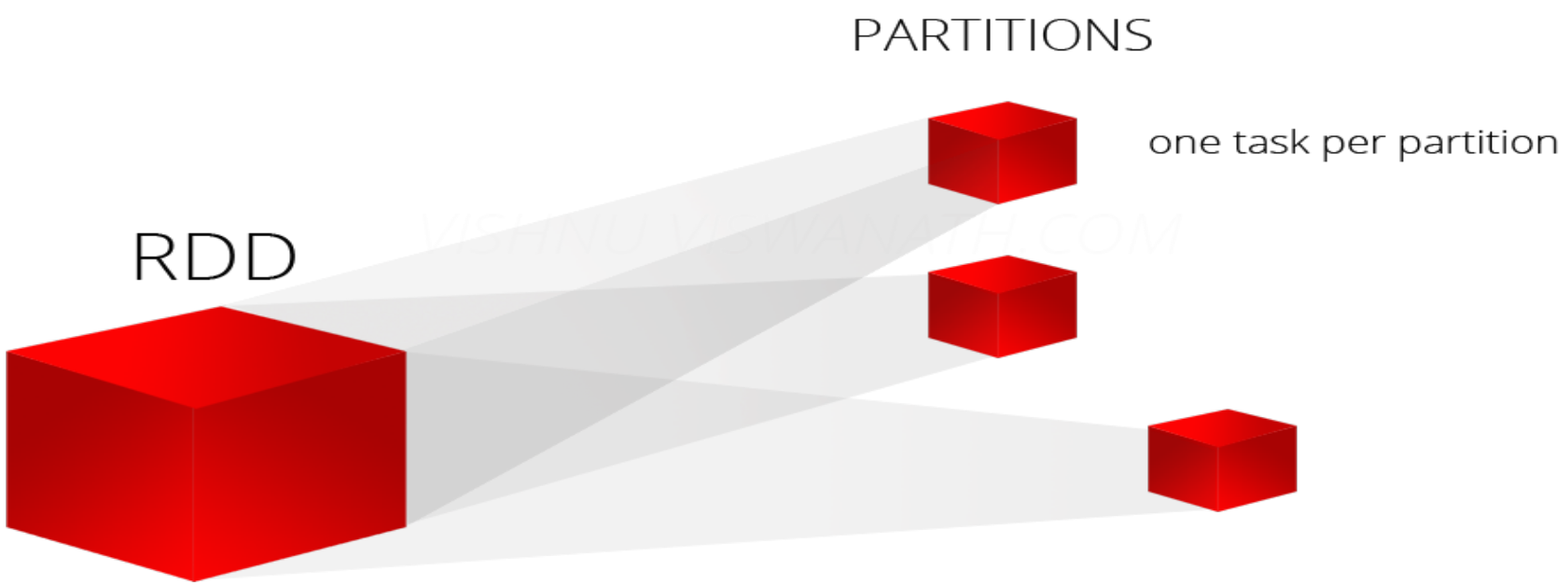
How to create RDD (1/2)

- **How to create RDD**?
  - Loading an external dataset
  - Distributing a collection of objects (e.g., a list or set) in their driver program
  - From data in memory
  - From another RDD

How to create RDD (2/2)

- **Important Note**: Source of some below figures from the following link: http://datalakes.com/rdds-simplified/

Figure 1: RDD Partitions



Creating RDD Example:

Scala Code 1: Creating RDD Example

```scala
//Read README.md File into lines
val lines = sc.textFile("README.md")
// lines: org.apache.spark.rdd.RDD[String] = README.md
    MapPartitionsRDD[3] at textFile at <console>:24
lines.take(3)
//res4: Array[String] = Array
//(# "Apache Spark", "", "Spark is a fast and general cluster
    computing system for Big Data. It provides")
lines.count
//res5: Long = 99
```

Spark Context:

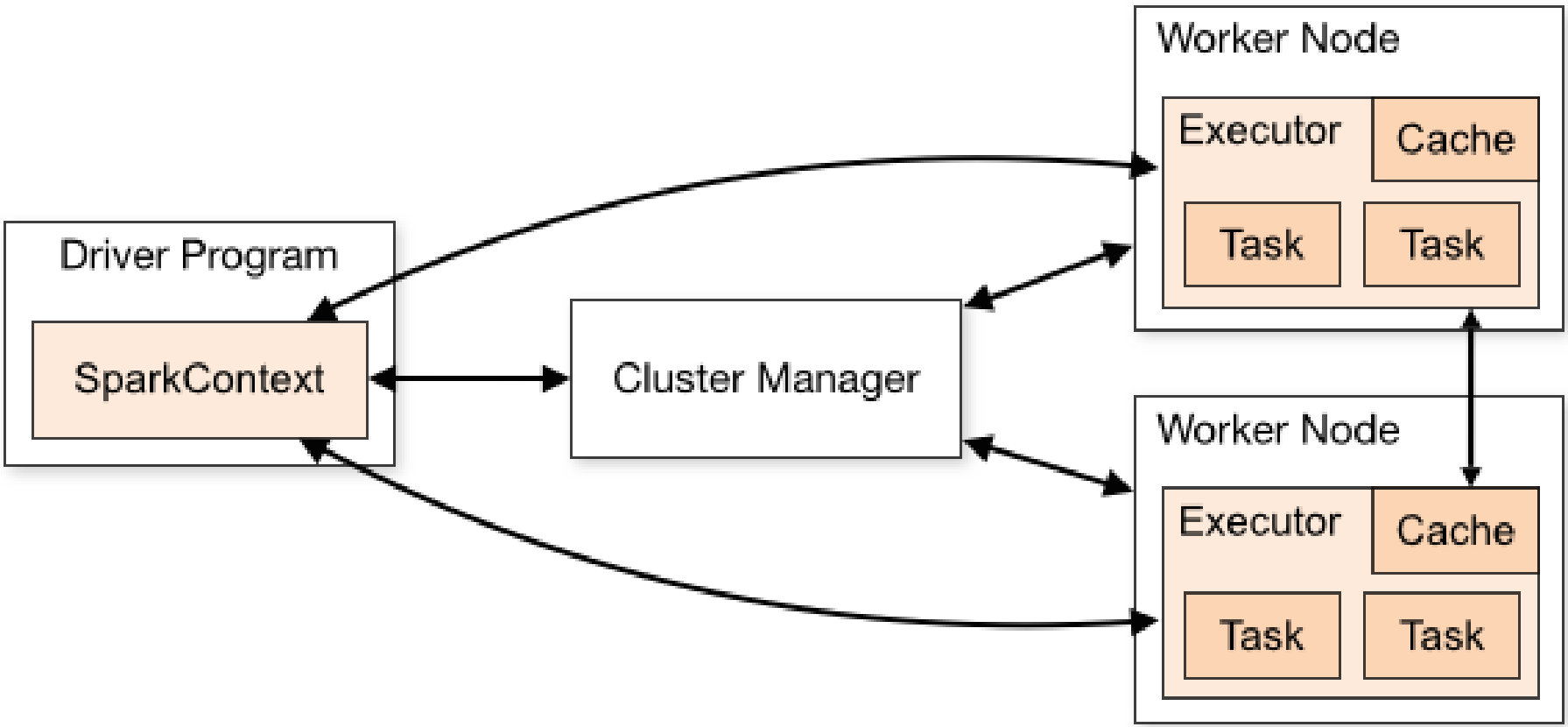
- **What is Spark Context?**
  - Your main entry point to spark API
  - The way and only way to distribute the data

Spark Driver(1/2):

- **What is Spark Driver?**
  - It is the program that declares the operations(transformations and actions) on RDDs of data and submits to the master
  - It is the program that creates the SparkContext, connecting to a given Spark Master "we can named as session "
  - It prepares the context and declares the operations on the data using RDD transformations and actions
  - It submits the serialized RDD graph to the master.
  - The master creates tasks out of it and submits them to the workers for execution. It coordinates the different job stages
  - The workers is where the tasks are actually executed. They should have the resources and network connectivity required to execute the operations requested on the RDDs

Spark Driver(2/2):

Figure 2: Spark Internal Interaction



RDD Operations(1/3):

- RDDs offers two types of operations: *Transformations & Actions*
  - *Transformations* : construct a new RDD from a previous one

Scala Code 2: RDD Transformations Example 1

```
1 // filter lines which contains "Scala"
2 val filter_lines = lines.filter(x => x.contains("Scala"))
3 // filter_lines : org.apache.spark.rdd.RDD[String] =
    MapPartitionsRDD[7] at filter at <console>:26
```

RDD Operations (2/3):

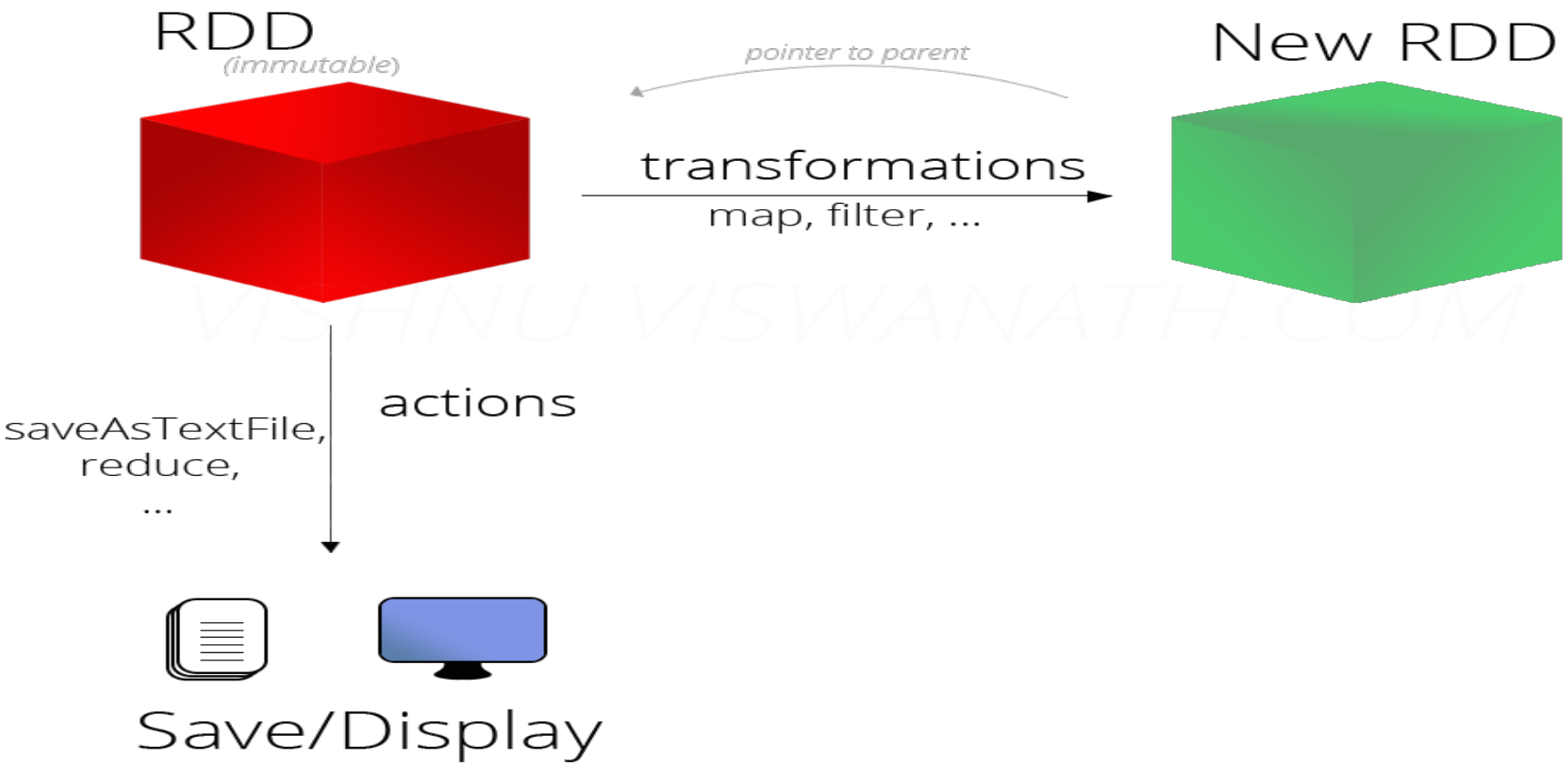
- Second type of operation: *Actions*
  - *Actions* : computes a result based on an RDD, and either return it to the driver program or save it to an external storage system (e.g., HDFS)

Scala Code 3: RDD Actions Example 1

```
1 //print first line from filtered data RDD
2  filter_lines . first ()
3 //res8: String = high-level APIs in Scala, Java, Python, and R,
      and an optimized engine that
4 //Count the RDD
5  filter_lines .count()
6 //res9: Long = 3
```

RDD Operations (3/3)

Figure 3: Actions/Transformations



RDD Operations: Transformations(1/2)

- Transformations create a new RDD from an exising one
- It works on one element at a time; but this is not true for all transformations
- **RDDs are immutable**
  - Data in an RDD is never changed
  - Transform in sequence to modify the data as needed
  - If we want to make any edit in the dataset you need to transform and the output of transformation will be save into new dataset
  - How can Spark know if we have multi-stages?
  - Spark keep track of each step you transform the data into **lineage graph**

RDD Operations: Transformations(2/2)

Figure 4: Example how filter/map transformation work

RDD Operations: Actions(1/4)

- They are the operations that return a final value to the driver program or write data to an external storage system
- Actions force the evaluation of the transformations required for the RDD they were called on

RDD Operations: Actions(2/4)

Scala Code 4: RDD Actions Example 2

```
1 //print first 10 line from filtered data RDD
2 //note tjat you can remove the foreach println if you are
      working in shell but if you will use scala in you external
      job you need to add it .
3 scala> filter_lines .take(10).foreach( println )
4 "high-level APIs in Scala, Java, Python, and R, and an
      optimized engine that"
5 ## Interactive Scala Shell
6 The easiest way to start using Spark is through the Scala
      shell :
```

RDD Operations: Actions(3/4)

- What actual happen when you do any action?
  - The data will be retrieved to the driver program
  - you can use collect to retrieve the all entire dataset,But Please don't use it unless the data is very small
  - Keep in mind that your entire dataset must fit in memory on a single machine to use collect
  - Also keep in mind again "Sorry to keep a lot of info into your mind " the entire RDD must be computed "from scratch." it is very costly to reprocess my data from scratch
  - We need to find a way to save the intermediate results into somewhere but please try to keep it in-memory. "Will be discussed later small hint '*persist or cashe*' "

RDD Operations: Actions(4/4)

- What I should do to do actions on the full data in Large datasets?
  - It's common to write data out to a distributed storage system such as HDFS or Amazon S3
  - You can save the contents of an RDD using the saveAsTextFile action, saveAsSequenceFile
  - Keep in mind that your entire dataset must fit in memory on a single machine to use collect

Functional programming features

- As we know spark core implementation built on scala. So, it must have some features these features inherited from Functional programming concepts not Spark. Then spark team implement the transformation to be distributed

Lazy Evaluation (1/4)

Scala Code 5: Spark Lazy Execution Example

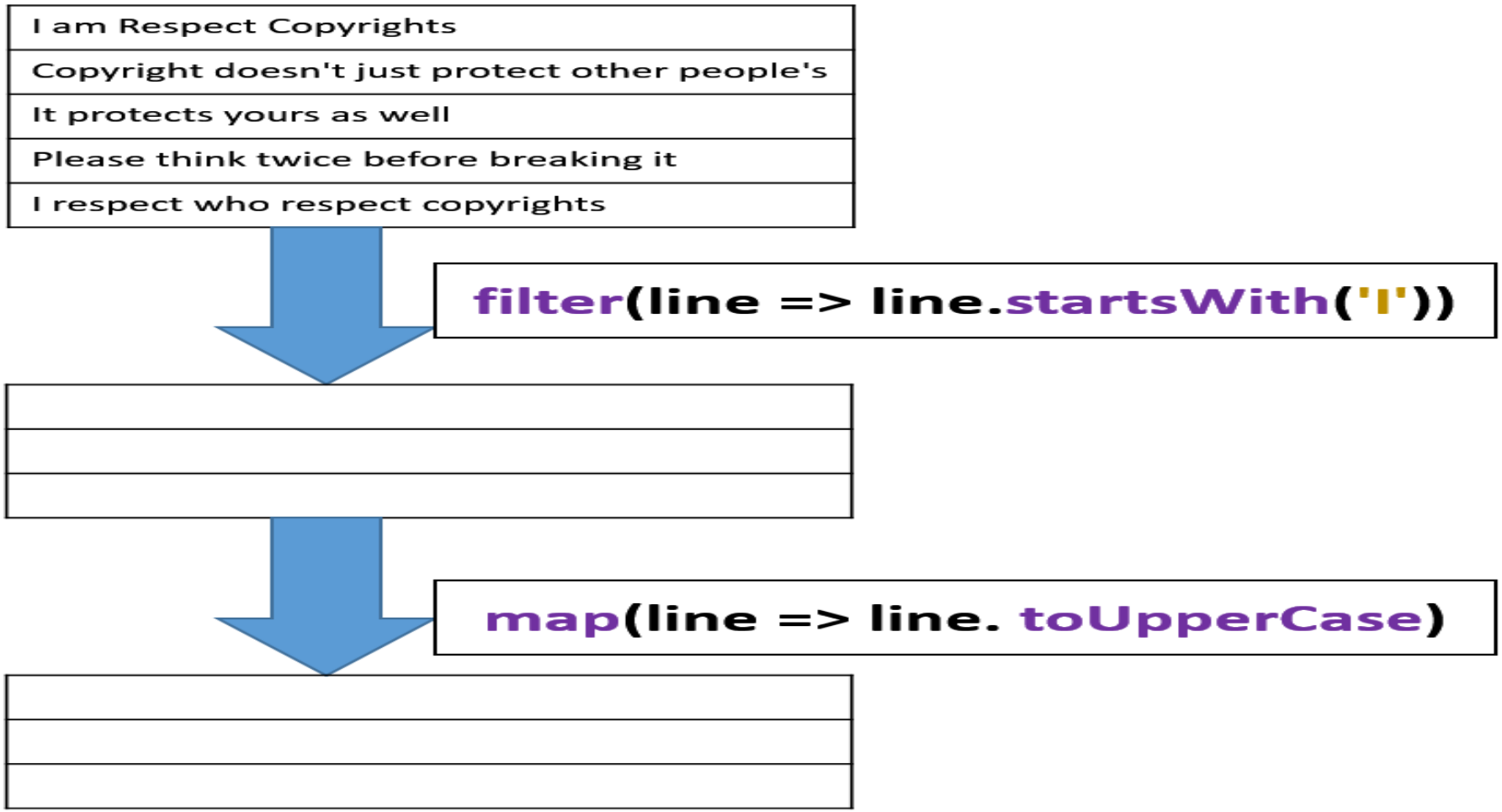
```
1  scala> val input = sc.parallelize(List("Mostafa Alaa,Big
       Data,Etisalat ,Smart Cities Project"))
2  res: input: org.apache.spark.rdd.RDD[String] =
       ParallelCollectionRDD[6] at parallelize at <console>:24
3  scala> val splittedCommaMap = input.map(x => x.split(","))
4  res2: splittedCommaMap: org.apache.spark.rdd.RDD[Array[String]]
       = MapPartitionsRDD[8]
5  scala> inputMapped.collect.map(x => x.foreach(println))
6  res3: Array[Unit] = Array(())
7  "Mostafa Alaa"
8  "Big Data"
9  " Etisalat"
10 "Smart Cities Project"
```

Lazy Evaluation (2/4)

- What if we pass a string to map and split on index 3 and the string has two index only?
- Does spark evaluate our transformation once we do mapping ?

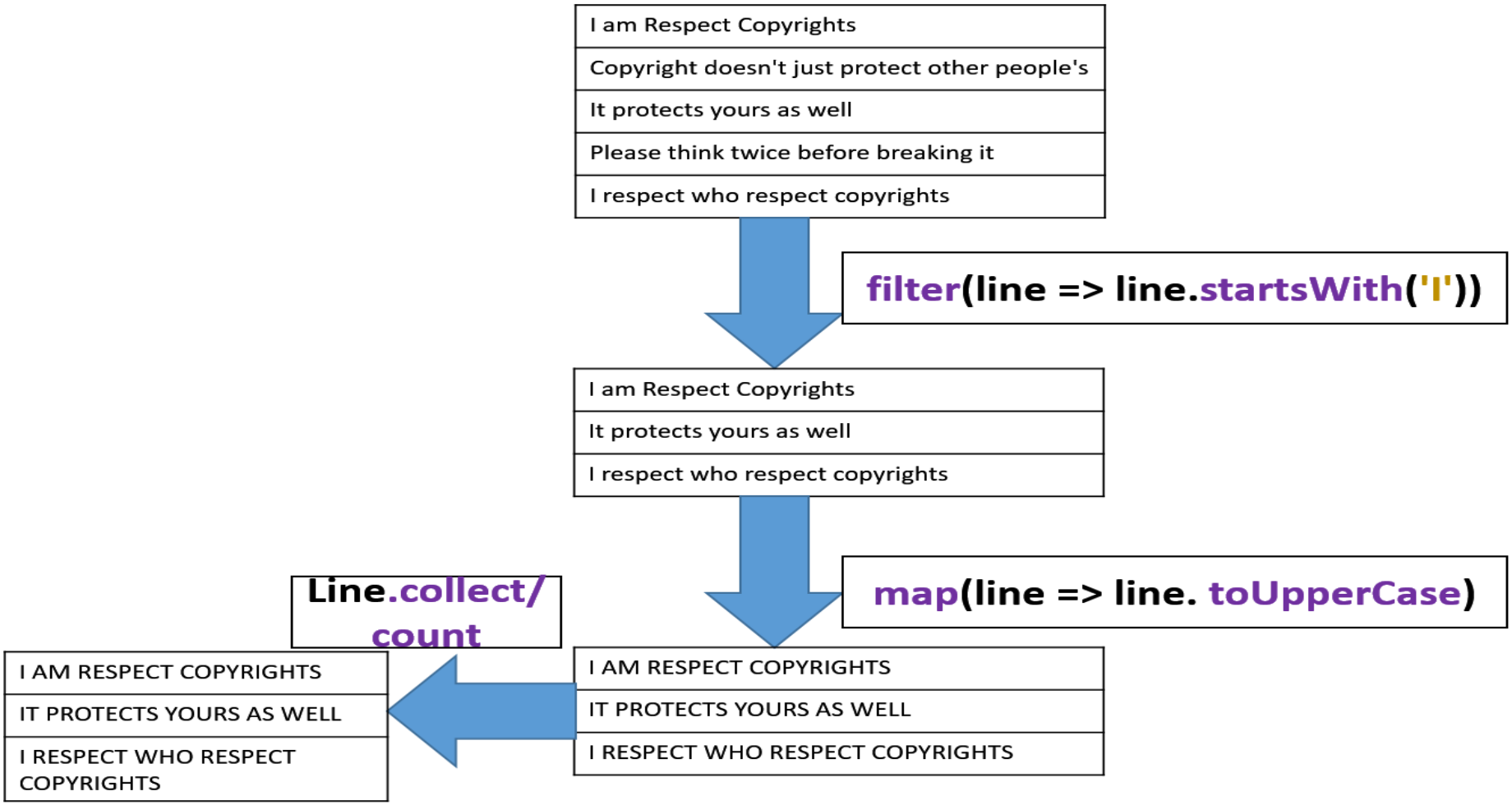
Lazy Evaluation (3/4)

Figure 5: Lazy Evaluation Execution 1



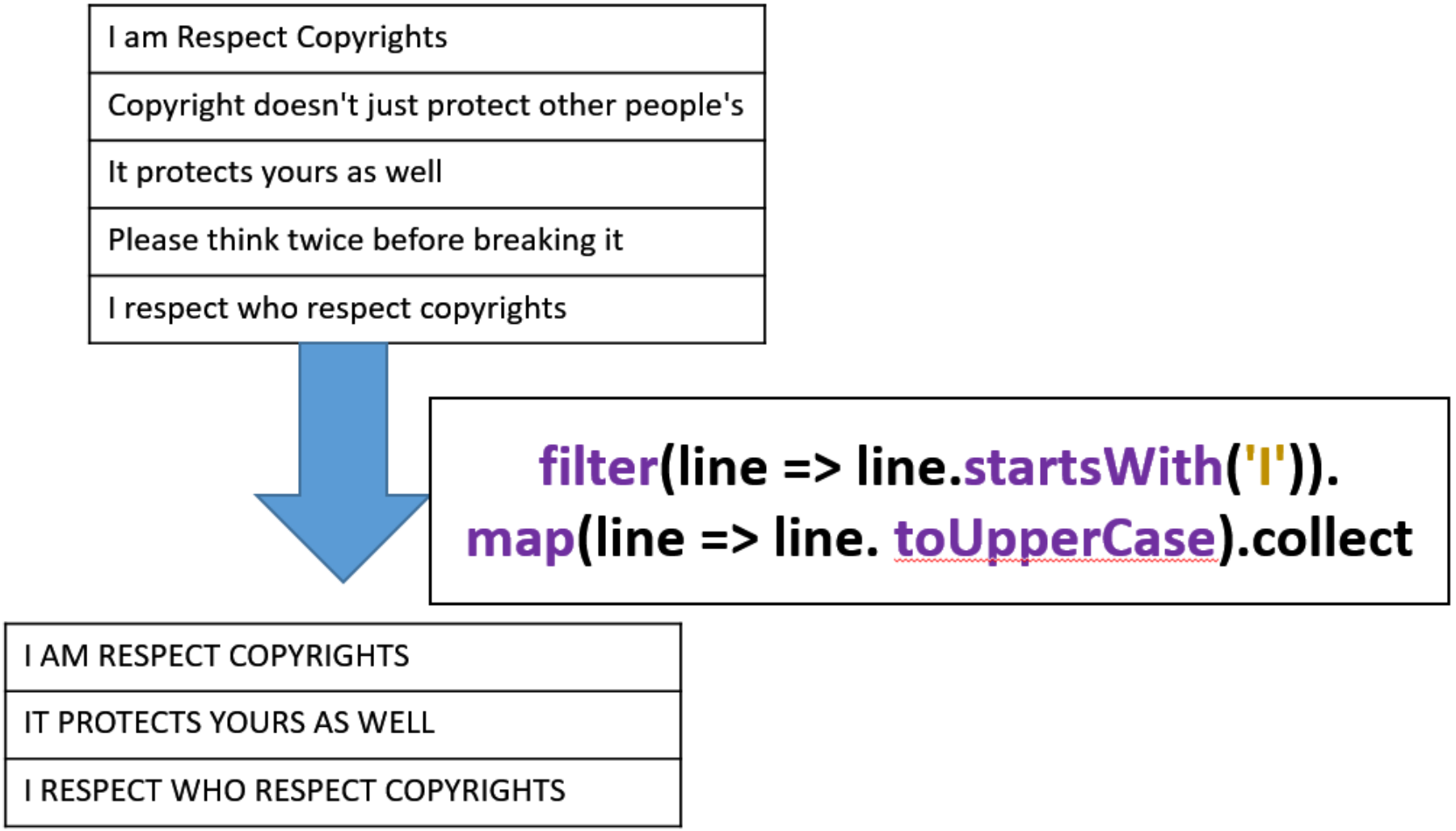
Lazy Evaluation (4/4)

Figure 6: Lazy Evaluation Execution 2



Chaining Transformations (1/2)

Figure 7: Chaining Transformations



Chaining Transformations (2/2)

Scala Code 6: Chaining Transformations Execution Example

```
1
2 scala> val lines = sc.textFile("README.md")
3 scala> lines.take(3)
4
5 // is the same as
6 val lines = sc.textFile("README.md").take(3)
```

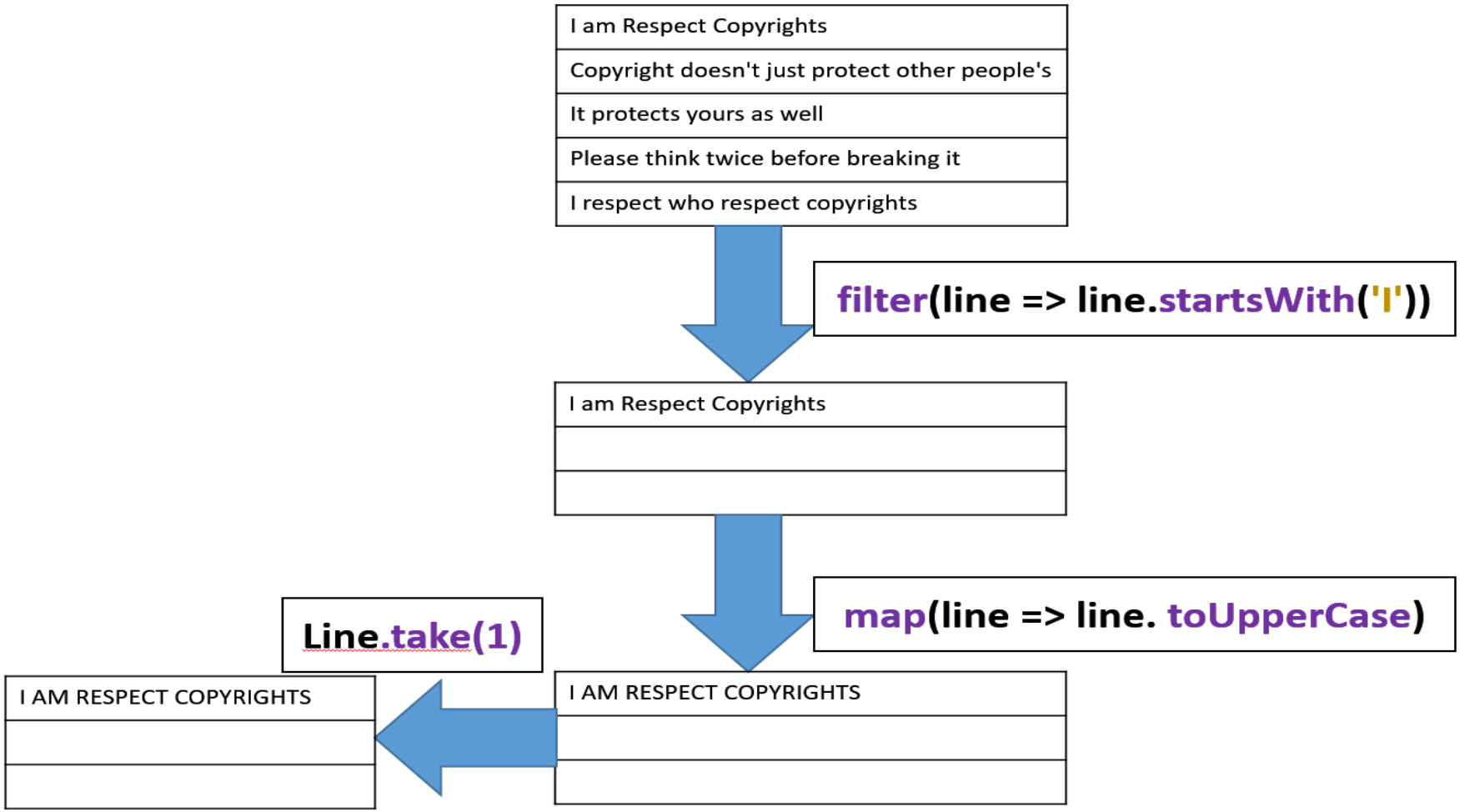
- How Spark work with both? please check this function and you will got it "toDebugString"

Pipelining (1/3)

- Motivation question: If you have huge amounts of data and you need to print sample of this data Ex: 3, what spark will do?

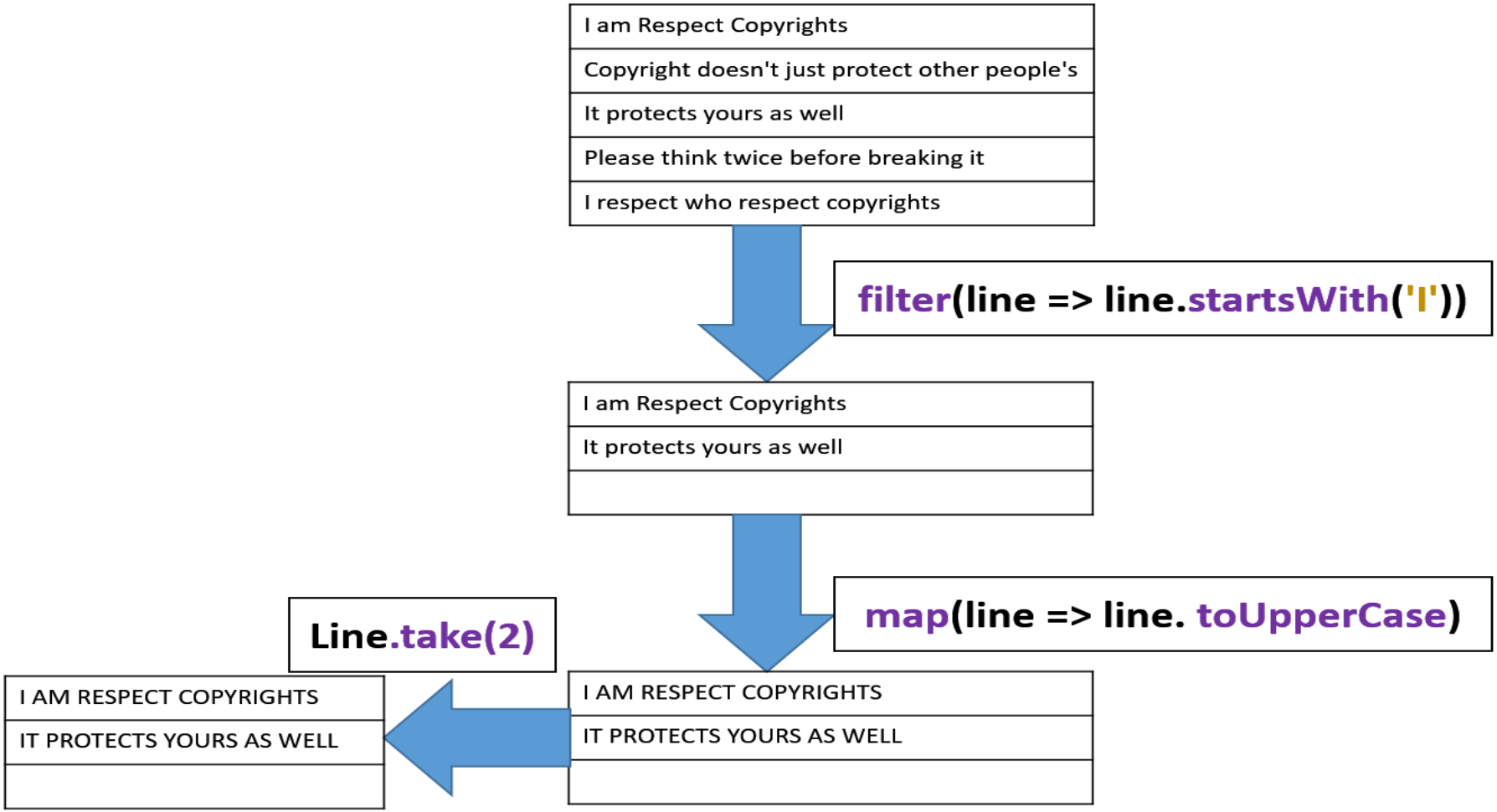
Pipelining (2/3)

Figure 8: Pipelining Example Part 1



Pipelining (3/3)

Figure 9: Pipelining Example Part 2



Passing Functions to Spark

- Spark depends heavily on the concepts of functional programming
- Functions are the fundamental unit of programming
- You need to think you will have a function passed to another function
- Most of modern programming language has the concept of anonymous functions

Scala Code 7: Passing Functions to RDD Example

```
1
2 def chnageString(s: String): String =
3 { s +"AAA" }
4
5 //Lets show our Data using "Take"
6 scala> lines.map(chnageString).take(2)
```

Persistence (Caching) (1/11)

- Spark RDDs are lazily evaluated, and sometimes we may wish to use the same RDD multiple times
- Spark will recompute the RDD and all of its dependencies each time we call an action on the RDD
- This can be especially expensive for iterative algorithms

Scala Code 8: Motivation Example Persistence

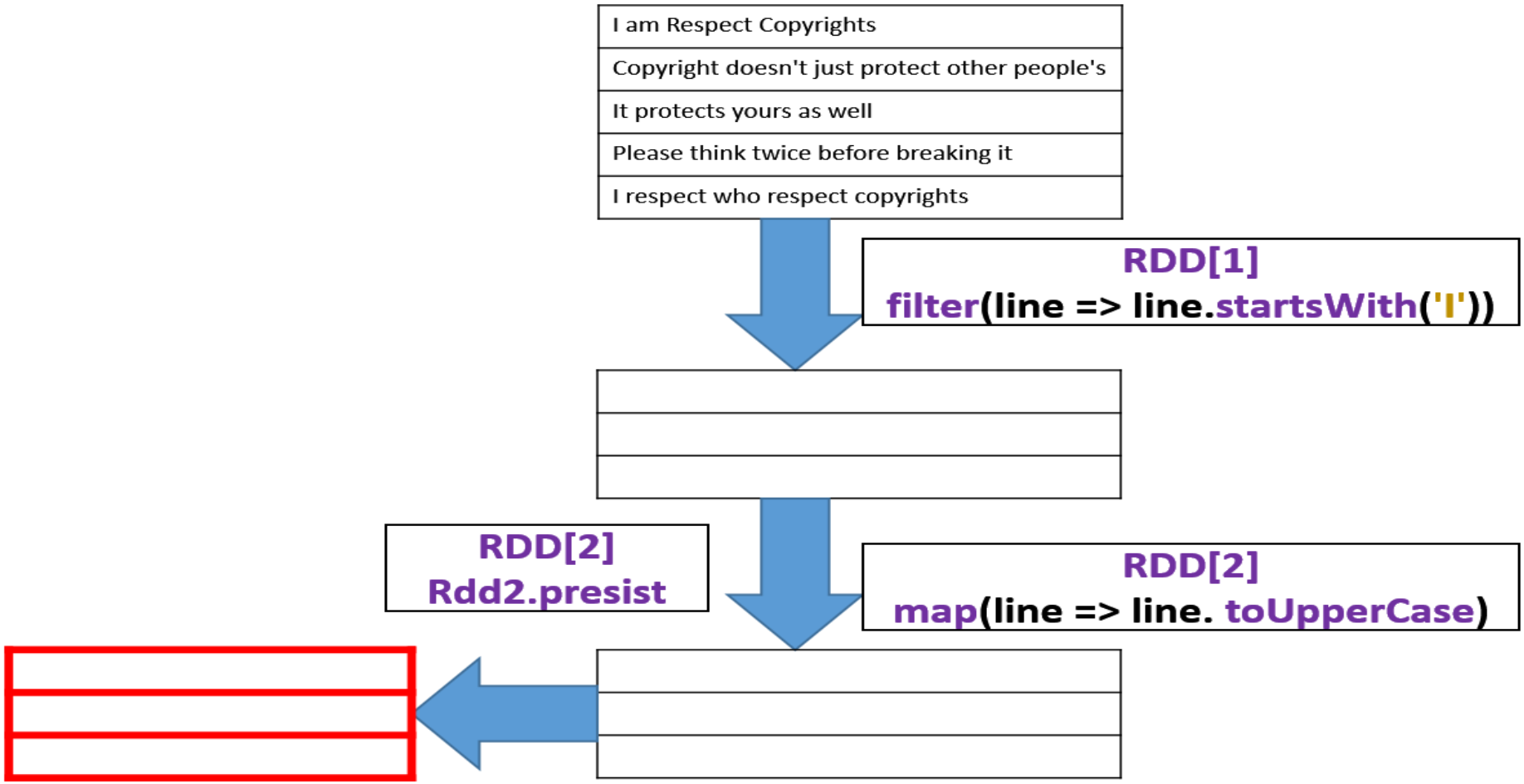
```
1 val result = input.map(x => x*x)
2 println(result.count())
3 println(result.collect().mkString(","))
```

Persistence (Caching) (2/11)

- What will happened if one of the nodes that has data persisted failed? *spark will recompute the lost partitions of the data when needed*
- How to avoid the cost of recompute the data? *We can also replicate our data on multiple nodes if we want to be able to handle node failure without slowdown*

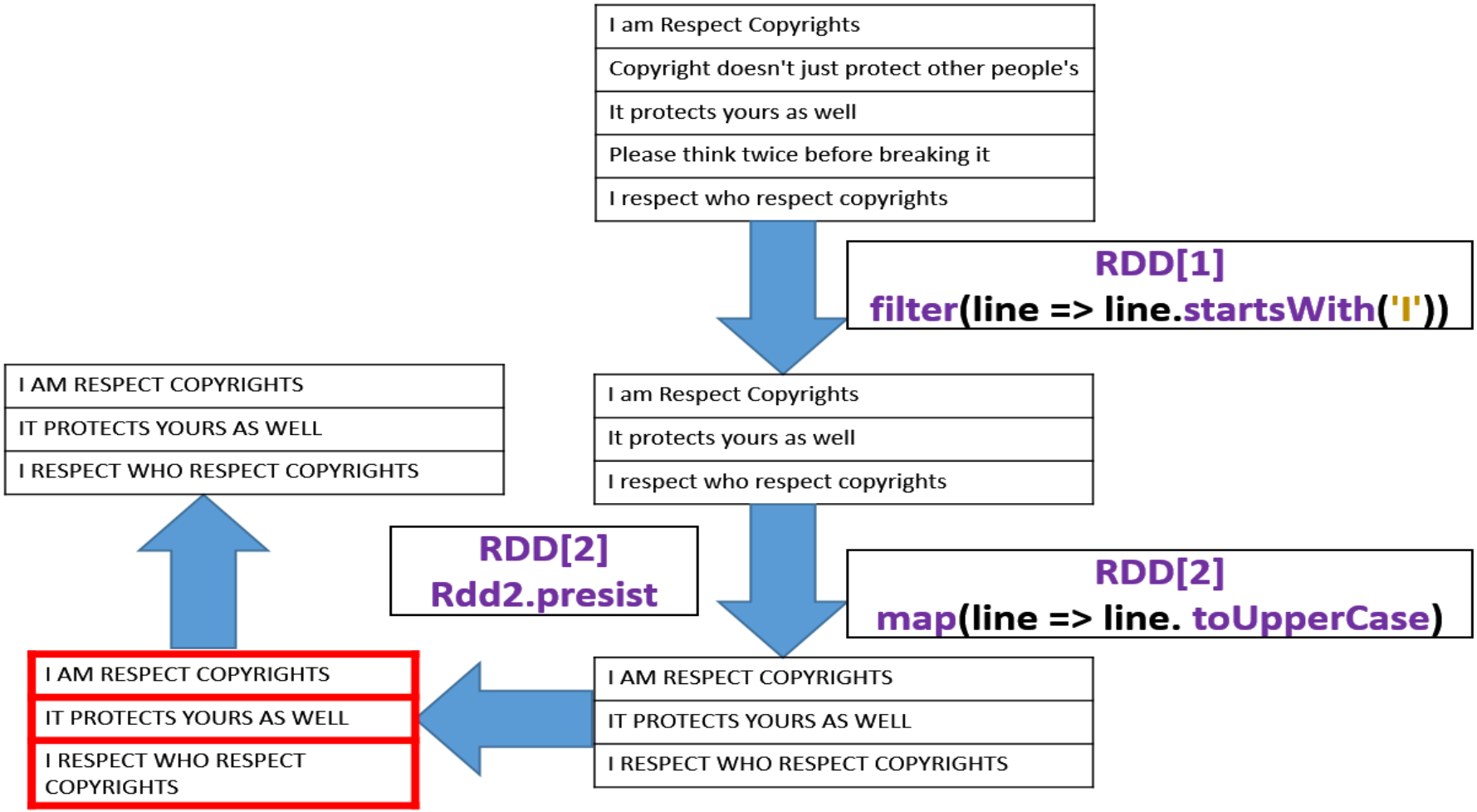
Persistence (Caching) (3/11)

Figure 10: Spark Caching Example Part 1



Persistence (Caching) (4/11)

Figure 11: Spark Caching Example Part 2



Persistence (Caching) (5/11)

- You control storage (location,format in memory, partition replication)
  - MEMORY_ONLY(default)
  - MEMORY_AND_DISK
  - DISK_ONLY

Scala Code 9: Persistence Storage Level Example

```
1 import org.apache.spark.storage.StorageLevel
2 data.persist(StorageLevel.MEMORY_AND_DISK.)
```

Persistence (Caching) (6/11)

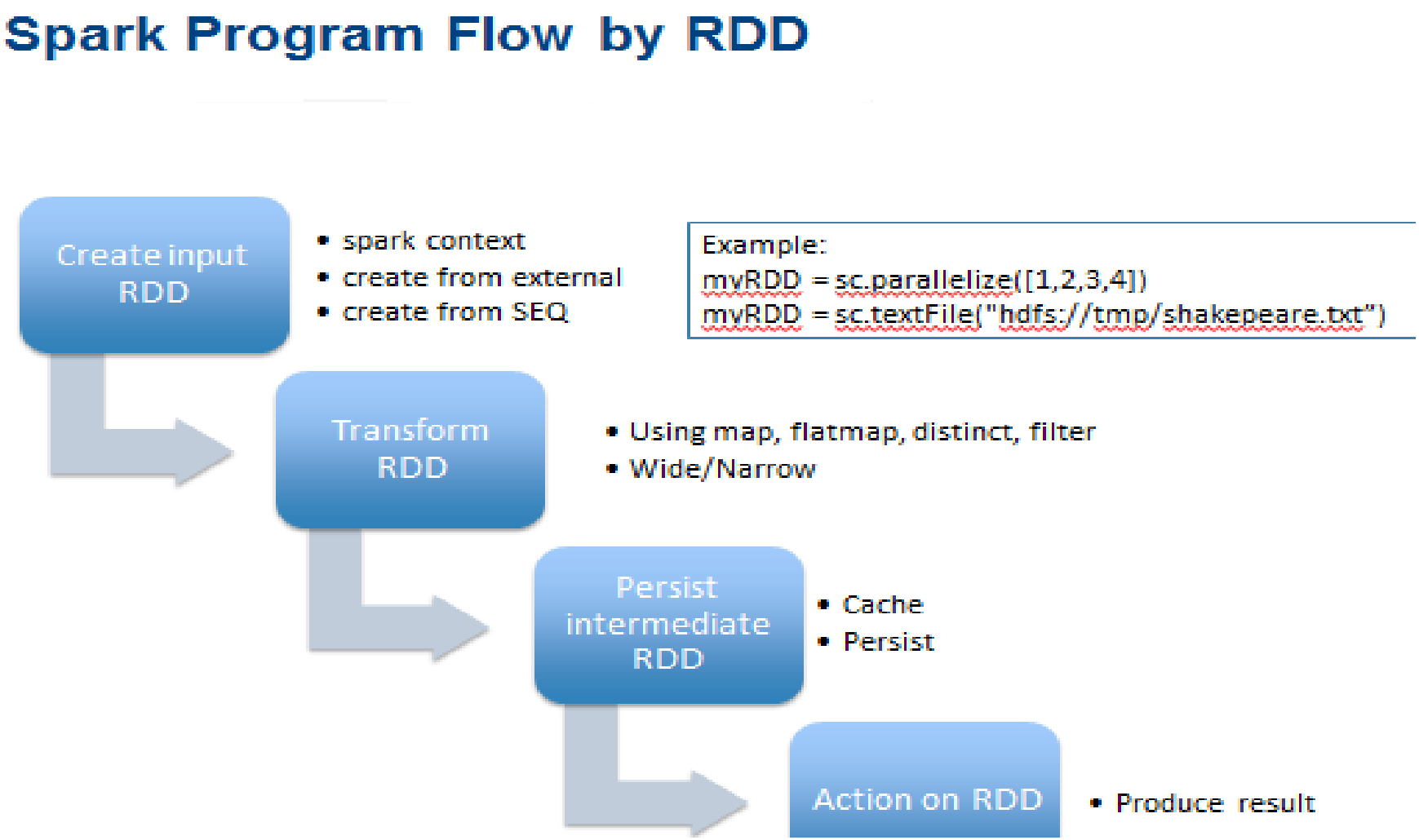
- Serialization
  1. MEMORY_ONLY_SER
  2. MEMORY_AND_DISK_SER
- Partition replication –store partitions on two nodes
  1. MEMORY_ONLY_2
  2. MEMORY_AND_DISK_2
  3. DISK_ONLY_2
  4. MEMORY_ONLY_SER_2
  5. MEMORY_AND_DISK_SER_2

Persistence (Caching) (7/11)

- spark cache vs persist
  1. The difference between cache and persist operations is purely syntactic. cache is a synonym of persist or persist(MEMORY_ONLY), i.e. cache is merely persist with the default storage level MEMORY_ONLY
  2. you can cache a RDD in memory doesn't mean you should blindly do so. Depending on how many times the dataset is accessed and the amount of work involved in doing so, re-computation can be faster than the price paid by the increased memory pressure

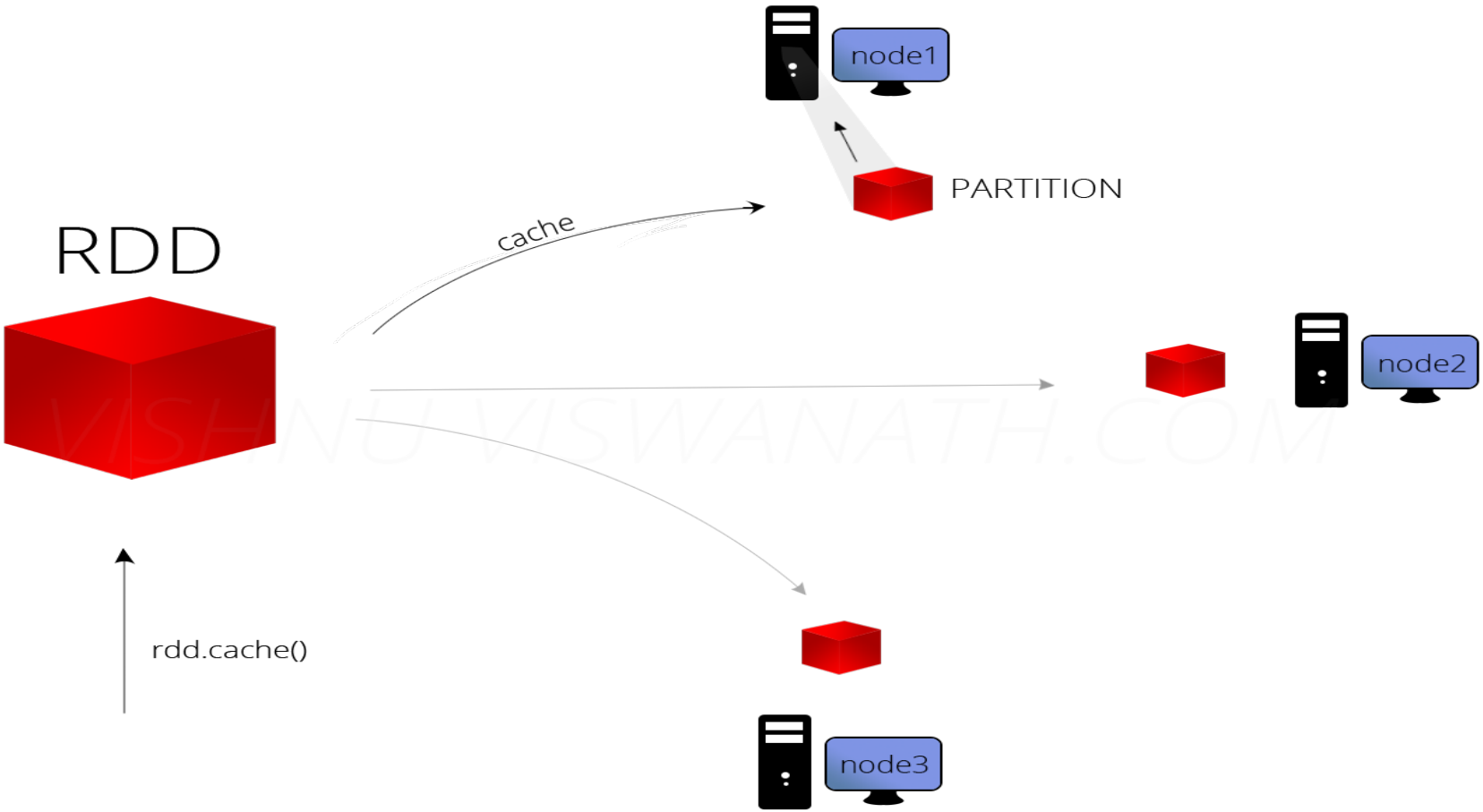
Persistence (Caching) (8/11)

Figure 12: Spark Program Workflow By RDD



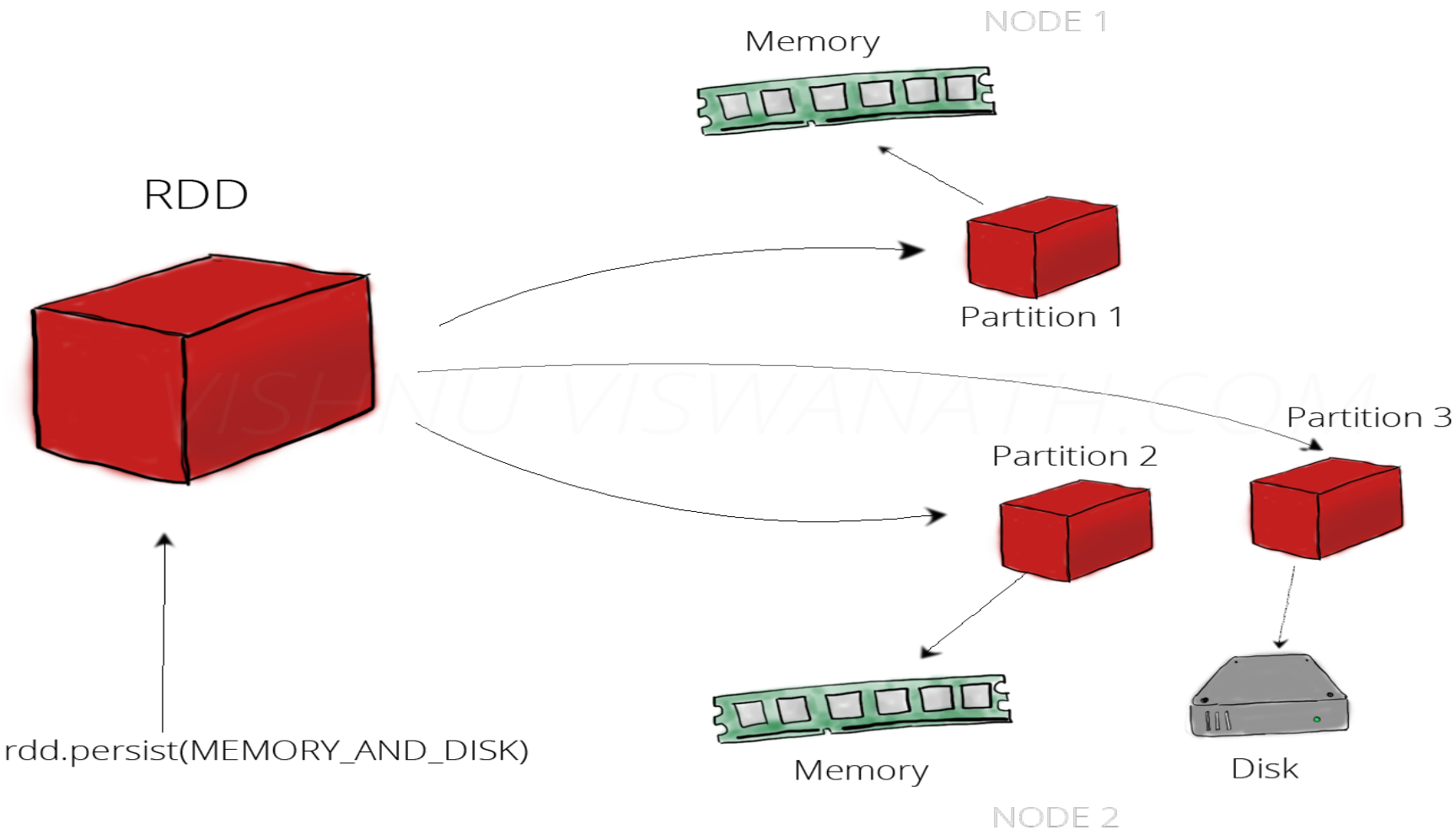
Persistence (Caching) (9/11)

Figure 13: Cache in memory



Persistence (Caching) (10/11)

Figure 14: Persist in memory and disk



Persistence (Caching) (11/11)

- Recommendations
  1. Use persist (memory and disk) when you are not sure the amount of data
  2. Use persist/cache whenever you have dataset which has huge computation dependency
  3. You need to use the Spark cache/persist to take the advantage of computation speed but take care

Broadcast variables (1/6)

- A broadcast variable, is a type of shared variable, used for broadcasting data across the cluster
- Hadoop MapReduce users can relate this to distributed cache

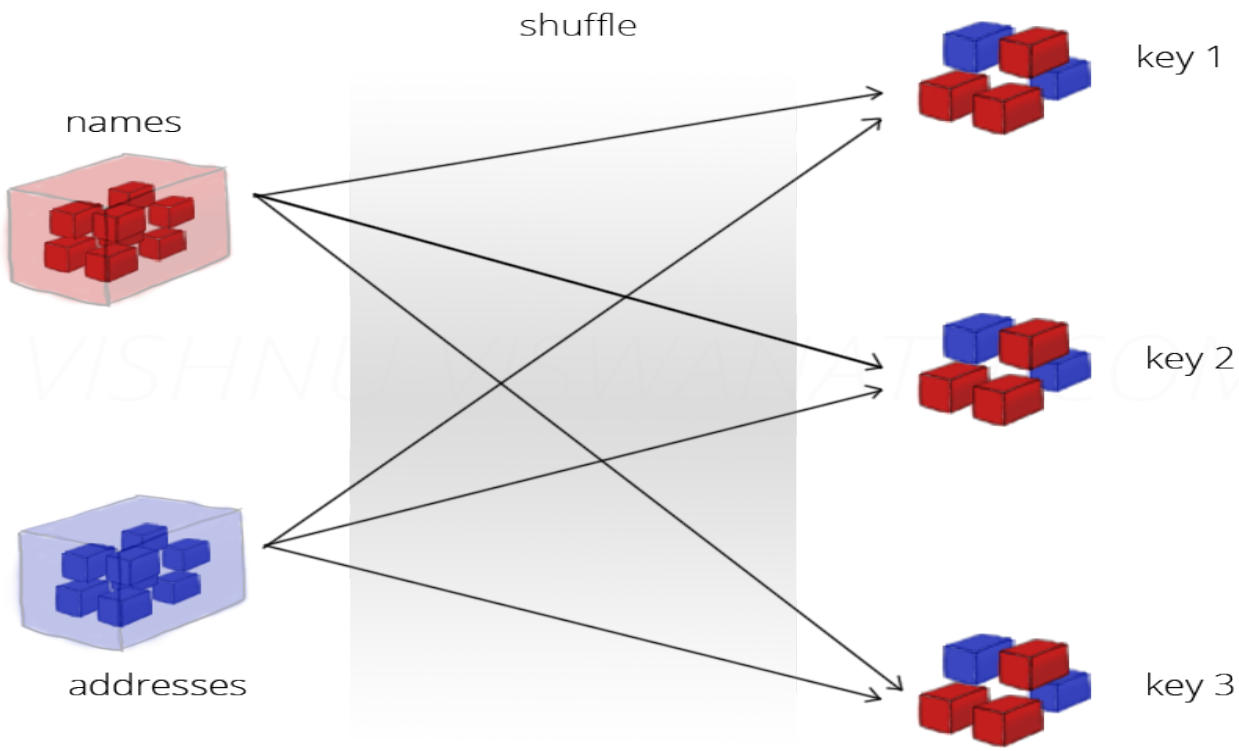
Scala Code 10: Broadcast variables Example

```
1
2 val names = sc.textFile("/names").map(line =>
      (line.split(",")(3), line))
3 val addresses = sc.textFile("/address").map(line =>
      (line.split(",")(0), line))
4 names.join(addresses)
```

- Here, both names and addresses will be shuffled over the network for performing the join which is not efficient since any data transfer over the network will reduce the execution speed

Broadcast variables (2/6)

Figure 15: Broadcast variables Shuffle over Network



Broadcast variables (3/6)

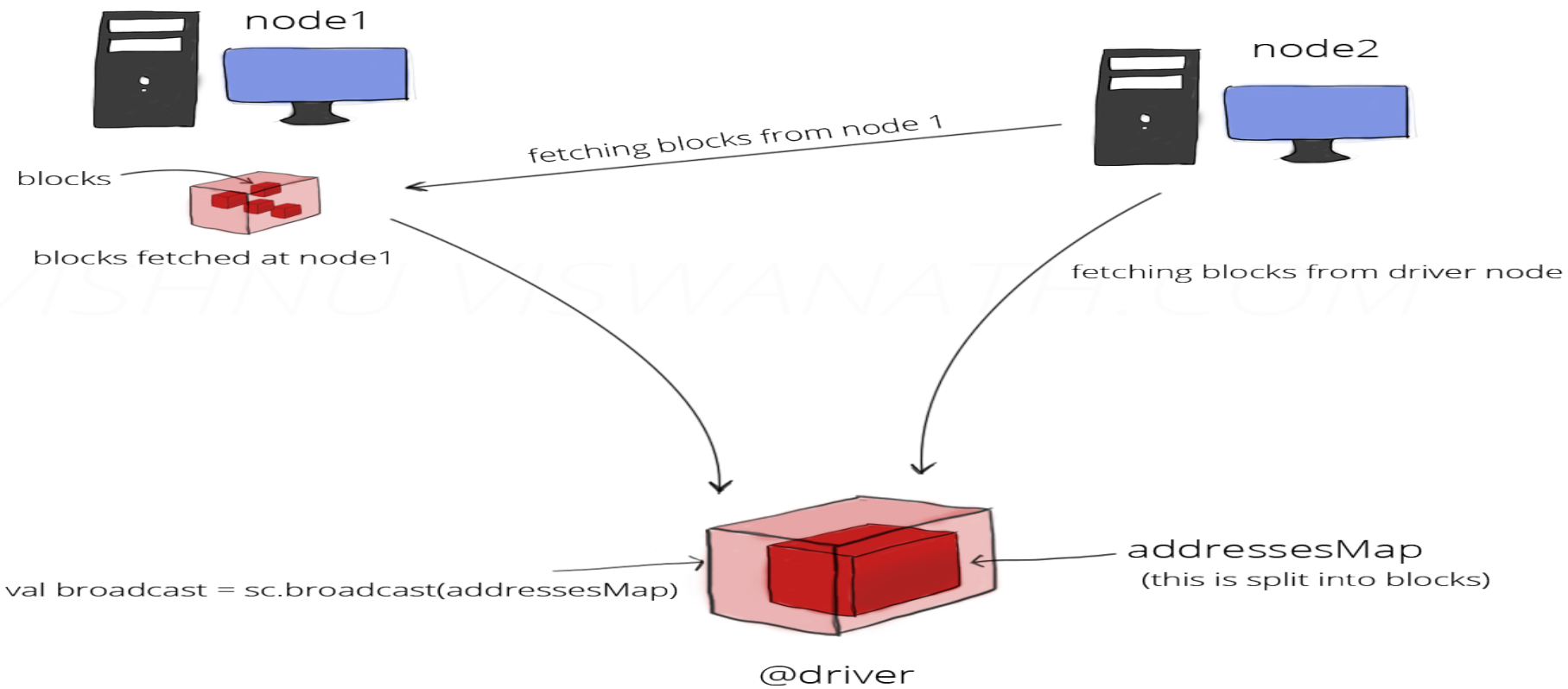
- Another approach is, if one of the RDDs is small in size, we can choose to send it to each node only once, thereby reducing network traffic
- **Note that:** Broadcast variables are read-only, broadcast.value is an immutable object along with each task. Consider the below example

Scala Code 11: Broadcast variables Enhanced Example

```
1
2   val names = sc.textFile("/names").map(line =>
        (line.split(",")(3), line))
3   val addresses = sc.textFile("/address").map(line =>
        (line.split(",")(0), line))
4   val addressedMap = addresses.collect().toMap
5   val broadcast = sc.broadcast(addressedMap)
6   val joined = names.map(v => v._2,(broadcast.value(v._1)))
```

Broadcast variables (4/6)

Figure 16: Join using Broadcast variables



Broadcast variables (5/6)

- Spark uses BitTorrent like protocol for sending the broadcast variable across the cluster, i.e., for each variable that has to be broadcasted, initially the driver will act as the only source. The data will be split into blocks at the driver and each leecher (receiver) will start fetching the block to it's local directory
- Once a block is completely received, then that leecher will also act as a source for this block for the rest of the leechers (This reduces the load at the machine running driver).

Broadcast variables (6/6)

- This is continued for rest of the blocks. So initially, only the driver is the source and later on the number of sources increases – because of this, rate at which the blocks are fetched by a node increases over time
- **Note that:** We are talking about small lookups not huge amount of data. If you are using it with huge tables that will joined so, Job will be failed easily. along with each task.

Questions about spark engine

- Tricky question: where spark save the linage? and what if linage lost?
- If we apply the transformation then spark will execute the datasets and get the results?
- Does map,filter and other functions is spark function at the beginning?
- Does spark main language which built on scala affect the thinking of the developers who develop spark?
- Transformation is only the spark built functions?
- Can we define a function and apply it to the a data? or we are limited with spark defined functions?