Apache Spark

Session 4 - Concepts

WELCOME - KNOWBIGDATA

- ❏ Expert Instructors
- ❏ CloudLabs
- ❏ Lifetime access to LMS
- ❏ Presentations
- ❏ Class Recording
- ❏ Assignments + Quizzes
- ❏ Project Work
- ❏ Real Life Project
- ❏ Course Completion Certificate
- ❏ 24x7 support
- ❏ KnowBigData - Alumni
  - ❏ Jobs
  - ❏ Stay Abreast (Updated Content, Complimentary Sessions)
  - ❏ Stay Connected

COURSE CONTENT

I	Introduction to Big Data with Apache Spark
II	Downloading Spark and Getting Started
III	Programming with RDDs
IV	Working with Key/Value Pairs
V	Loading and Saving Your Data
VI	Advanced Spark Programming
VII	Running on a Cluster
VIII	Tuning and Debugging Spark
IX	Spark SQL, SparkR
X	Spark Streaming
XI	Machine Learning with MLlib, GraphX

About Instructor?

2014	**KnowBigData**	Founded
2014 2012	**Amazon**	Built High Throughput Systems for Amazon.com site using in-house NoSql.
2012	**InMobi**	Built Recommender that churns 200 TB
2011 2006 2002	**tBits Global**	Founded tBits Global Built an enterprise grade Document Management System
	D.E.Shaw	Built the big data systems before the term was coined
2002	**IIT Roorkee**	Finished B.Tech.

Starting Spark With Python Interactive Shell

```
$ pyspark
or
$ cd /usr/spark2.6
$ bin/pyspark
```

```
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 1.5.0
      /_/

Using Python version 2.6.6 (r266:84292, Jan 22 2014 09:42:36)
SparkContext available as sc, HiveContext available as sqlContext.
>>>
```

It is basically the python interactive shell with one extra variable “sc”.
Check dir(sc) or help(sc)

Starting Spark With Python Job Submit

```
$ cd cd spark-1.5.0-bin-hadoop2.4
$ bin/spark-submit ../myprog.py
```

Submitting a java job

1. Download the project from LMS
2. Create fresh workspace inside eclipse
3. import the project
4. Fix lib path of spark jar
   a. Go to Project properties
   b. Under libraries: remove missing file and add external jar
5. Export jar
6. upload to hue and then copy: hadoop fs -copyToLocal
7. cd spark
8. bin/spark-submit --class sparkex.WordCount ../wc-p1.jar

```
15/09/25 04:15:54 INFO DAGScheduler: Job 0 finished: take at WordCount.java:29, took 0.817581 s
The
Project
Gutenberg
EBook
15/09/25 04:15:54 INFO SparkUI: Stopped Spark web UI at http://107.170.133.44:4040
```

Getting started with Scala Interactive Shell

```
$ cd cd spark-1.5.0-bin-hadoop2.4
$ bin/spark-shell
```

```
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 1.5.0
      /_/

SQL context available as sqlContext.

scala>
```

What is RDD?

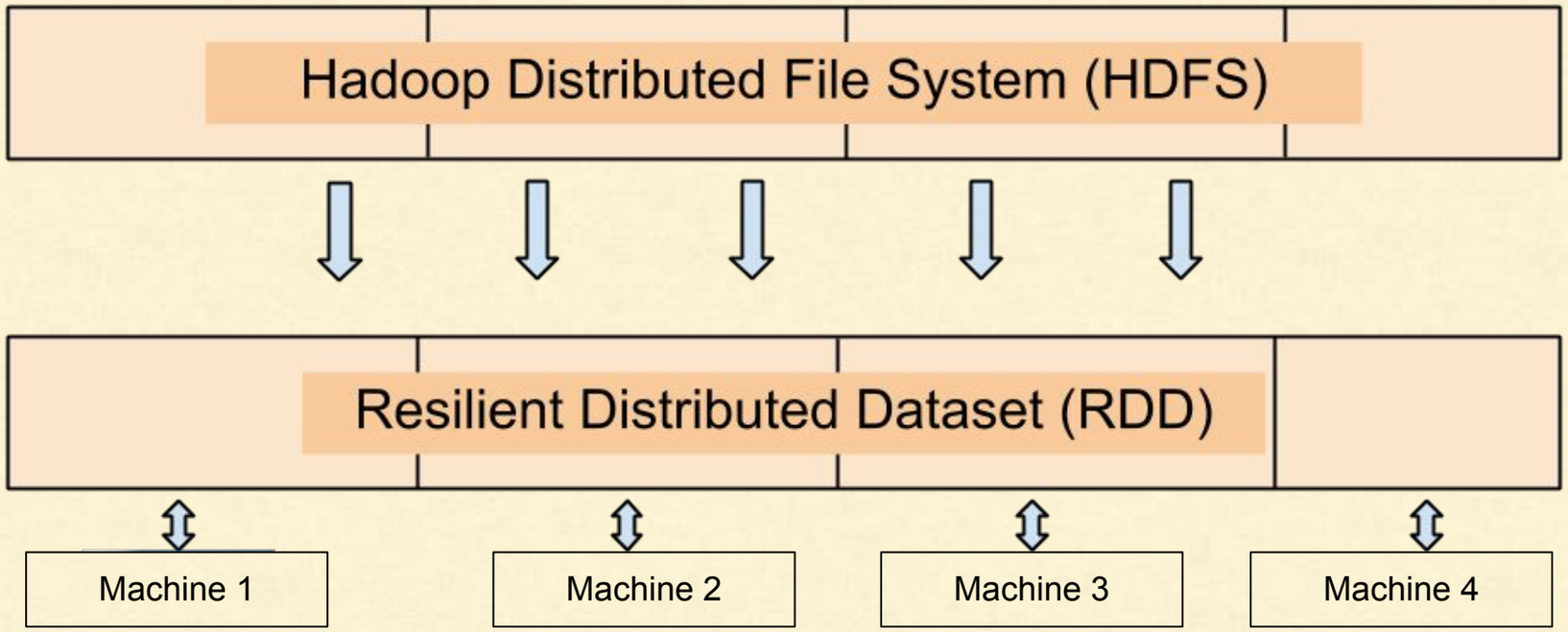
Dataset:
Collection of data elements.
e.g. Array, Tables, Data frame (R), collections of mongodb

Distributed:
Parts Multiple machines

Resilient:
Recovers on Failure

SPARK - CONCEPTS - RESILIENT DISTRIBUTED DATASET

A collection of elements partitioned across cluster



SPARK - CONCEPTS - RESILIENT DISTRIBUTED DATASET

A collection of elements partitioned across cluster

- An immutable distributed collection of objects.
- Split in partitions which may be on multiple nodes
- Can contain any data type:
  - Python,
  - Java,
  - Scala objects
  - including user defined classes

SPARK - CONCEPTS - RESILIENT DISTRIBUTED DATASET

- RDD Can be persisted in memory
- RDD Auto recover from node failures
- Can have any data type but has a special dataset type for key-value
- Supports two type of operations: transformation and action
- Each Element of RDD across cluster is run through map function

Creating RDD

Method 1: By Directly Loading a file from remote

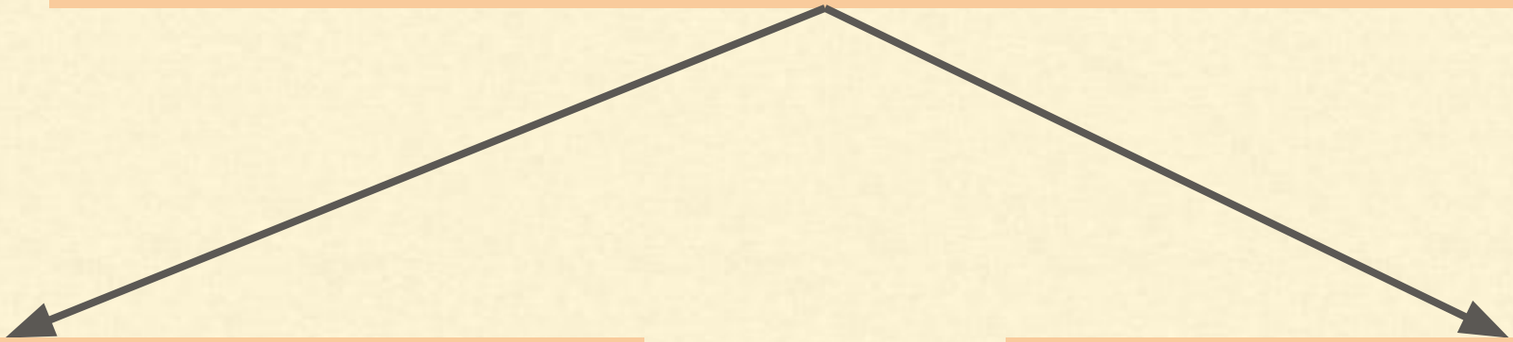
```
>>lines = sc.textFile('hdfs://hadoop1.knowbigdata.com/data/mr/wordcount/input/big.txt')
```

Method 2: By distributing existing object

```
>> arr = range(1, 1000000)
>> numbers = sc.parallelize(arr)
```

Churning RDD

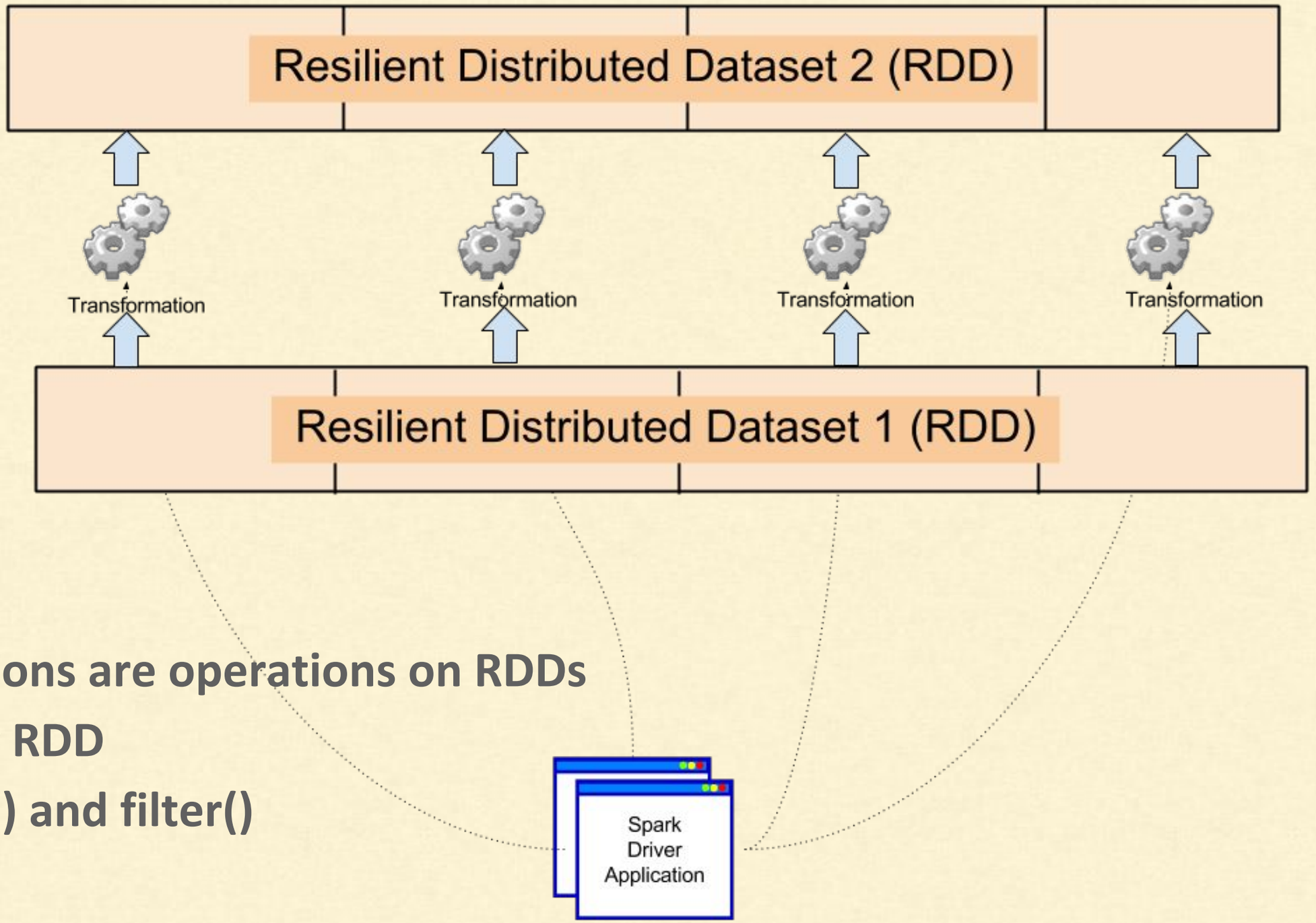
Two Kinds Operations



Transformation

Action

RDD - Operations : Transformation

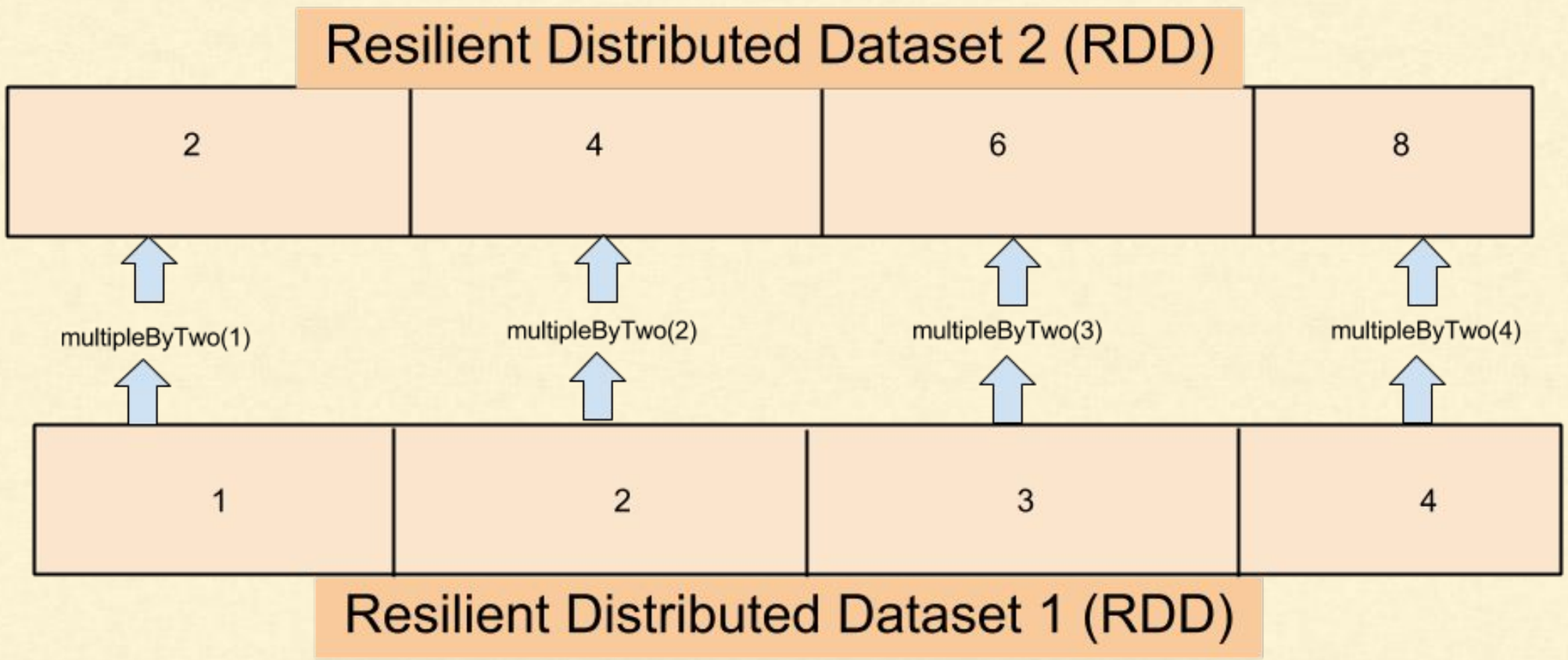


- Transformations are operations on RDDs
- return a new RDD
- such as map() and filter()

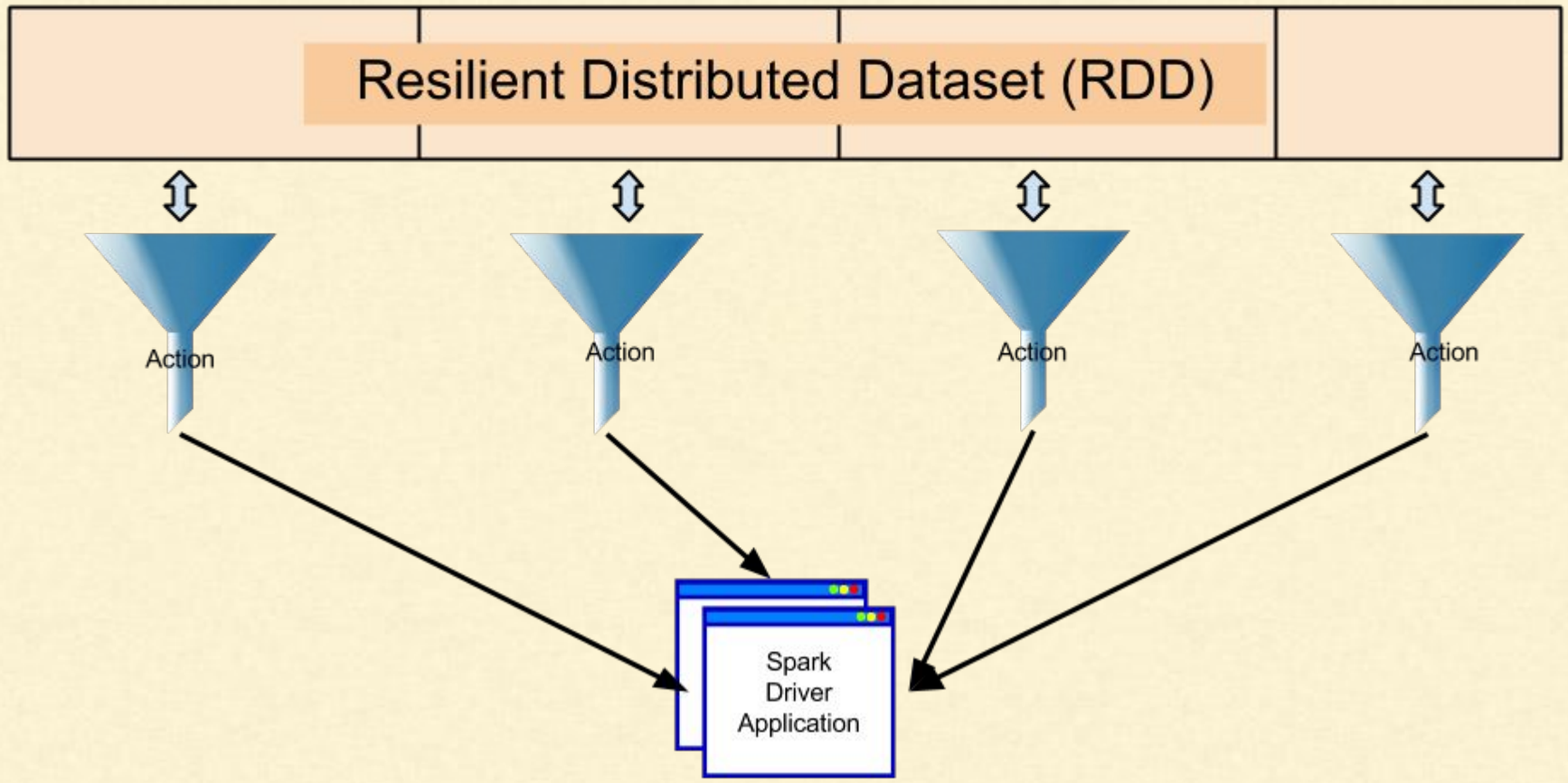
Transformation Example

- arr = range(1, 1000000)
- nums = sc.parallelize(arr)
- def multipleByTwo(x):
  return x*2;

- ***dbls = nums.map (multipleByTwo);***
- dbls.take(5)
- [2, 4, 6, 8, 10]



RDD - Operations : Actions

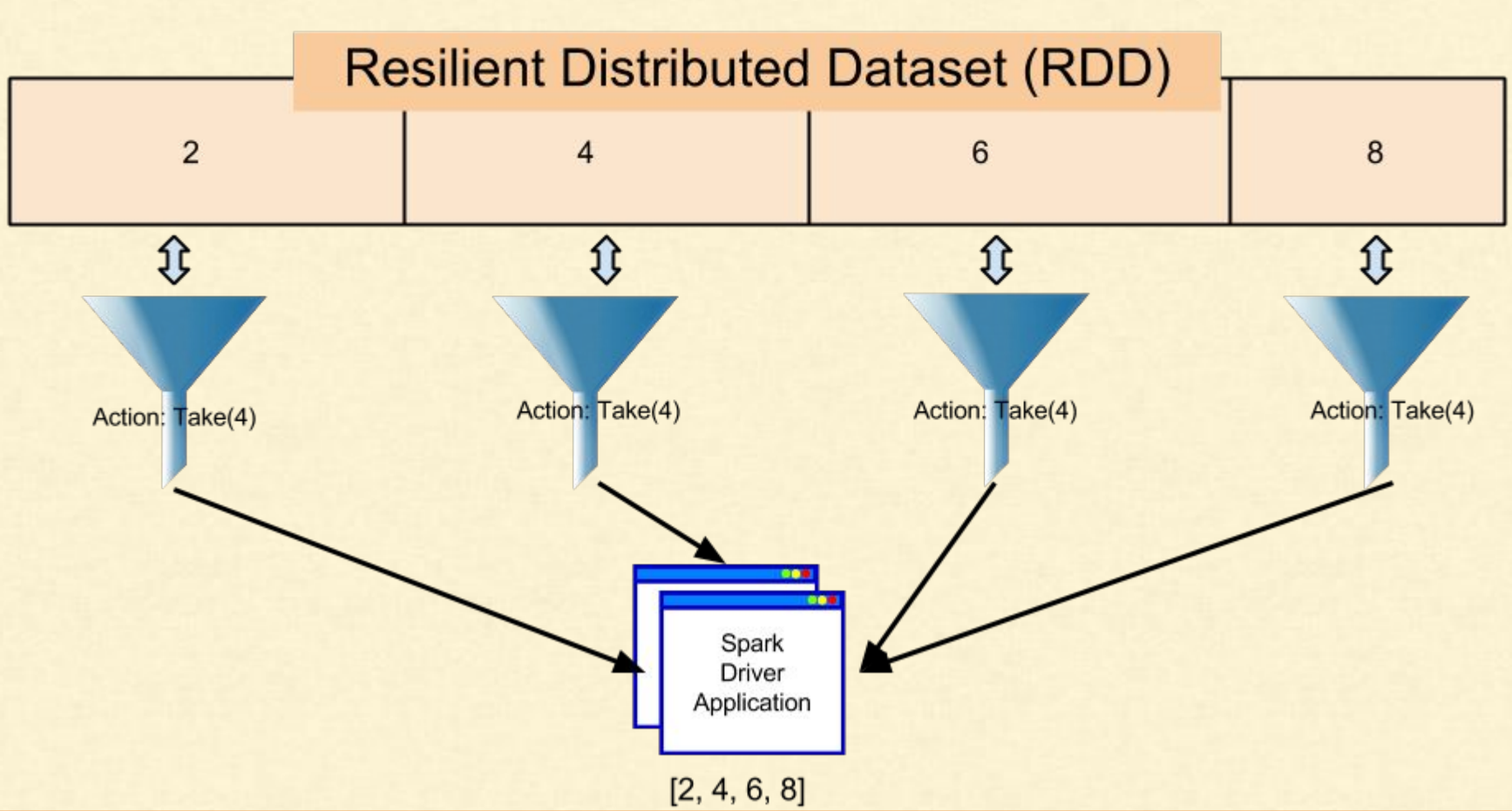


- **Brings back the data to driver**
- **Causes the full execution of transformations**
- **Involves both spark driver as well as the nodes**

Action Example

- arr = range(1, 1000000)
- nums = sc.parallelize(arr)
- def multipleByTwo(x):
  return x*2;

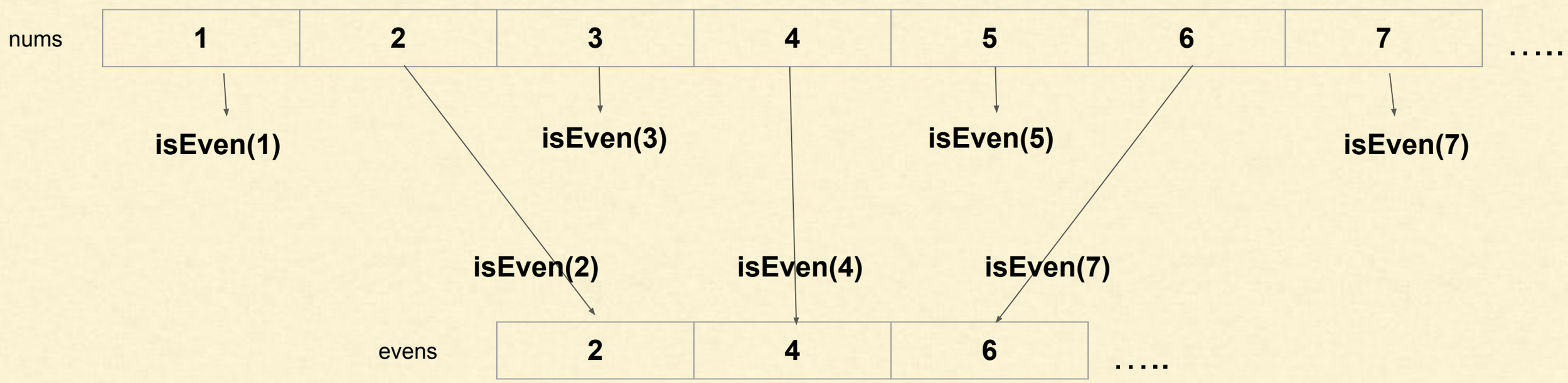
- dbls = nums.map (multipleByTwo);
- ***dbls.take(4)***
- [2, 4, 6, 8]



Transformations - filter()

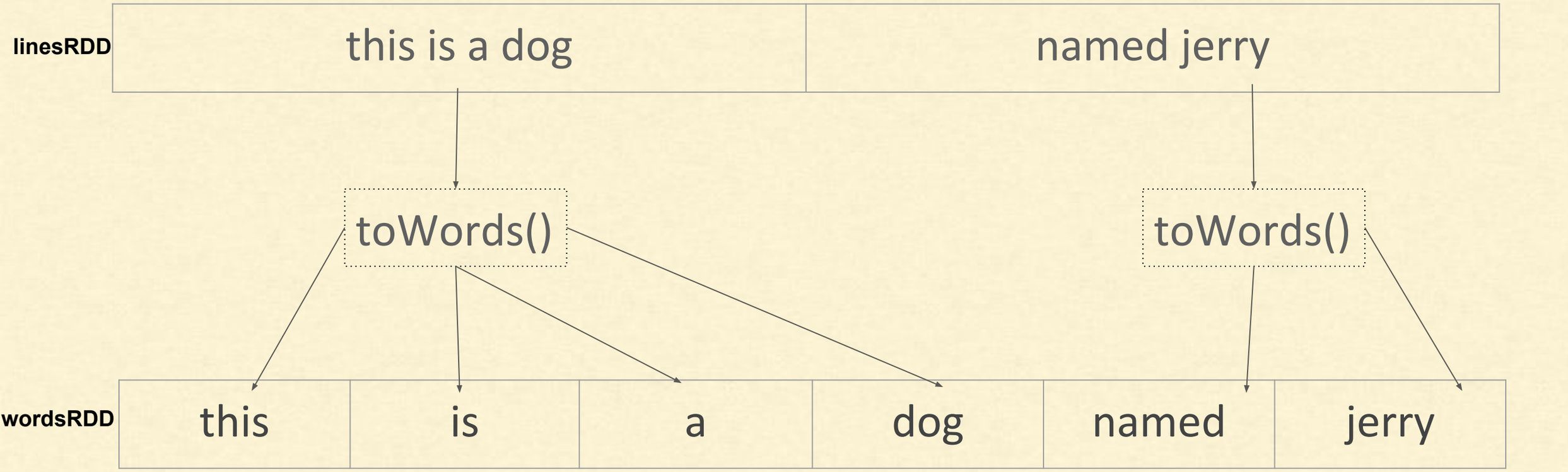
- arr = range(1, 1000000)
- nums = sc.parallelize(arr)
- def isEven(x):
  return x%2 == 0;

- ***evens = nums.filter(isEven);***
- ***evens***.take(3)
- [2, 4, 6]



Transformations:: flatMap()

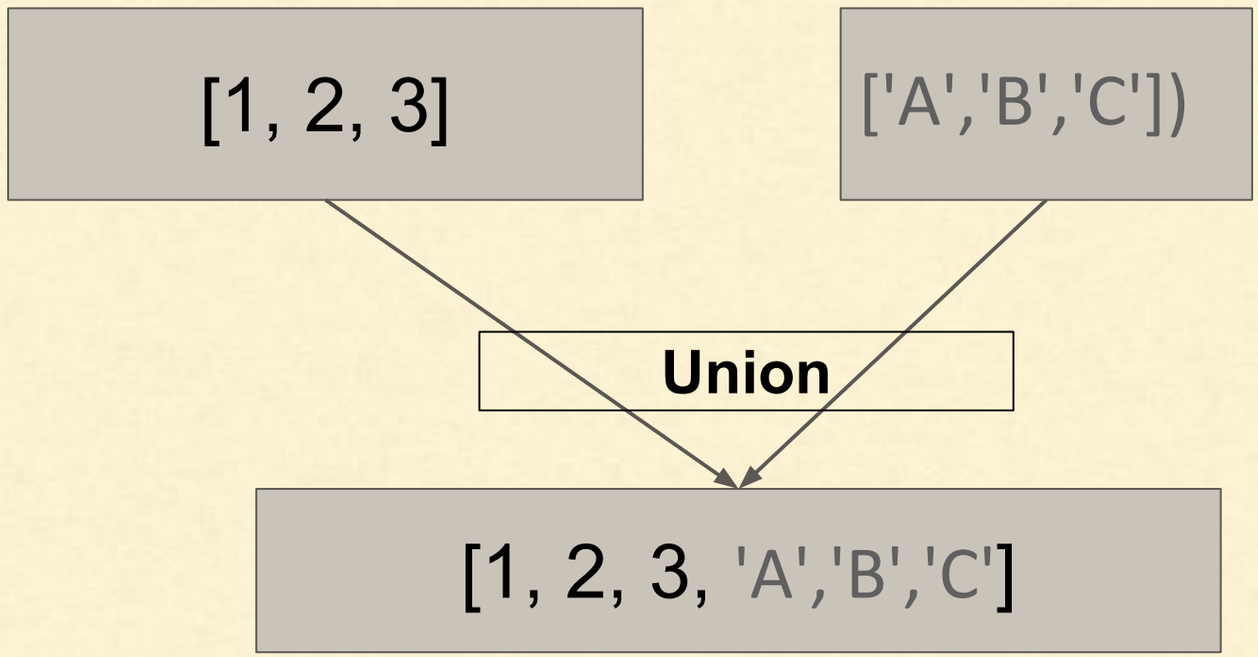
- linesRDD = sc.parallelize(["this is a dog", "named jerry"])
- def toWords(line):
  return line.split()
- wordsRDD = linesRDD.flatMap(toWords)
- wordsRDD.collect()
- *['this', 'is', 'a', 'dog', 'named', 'jerry']*



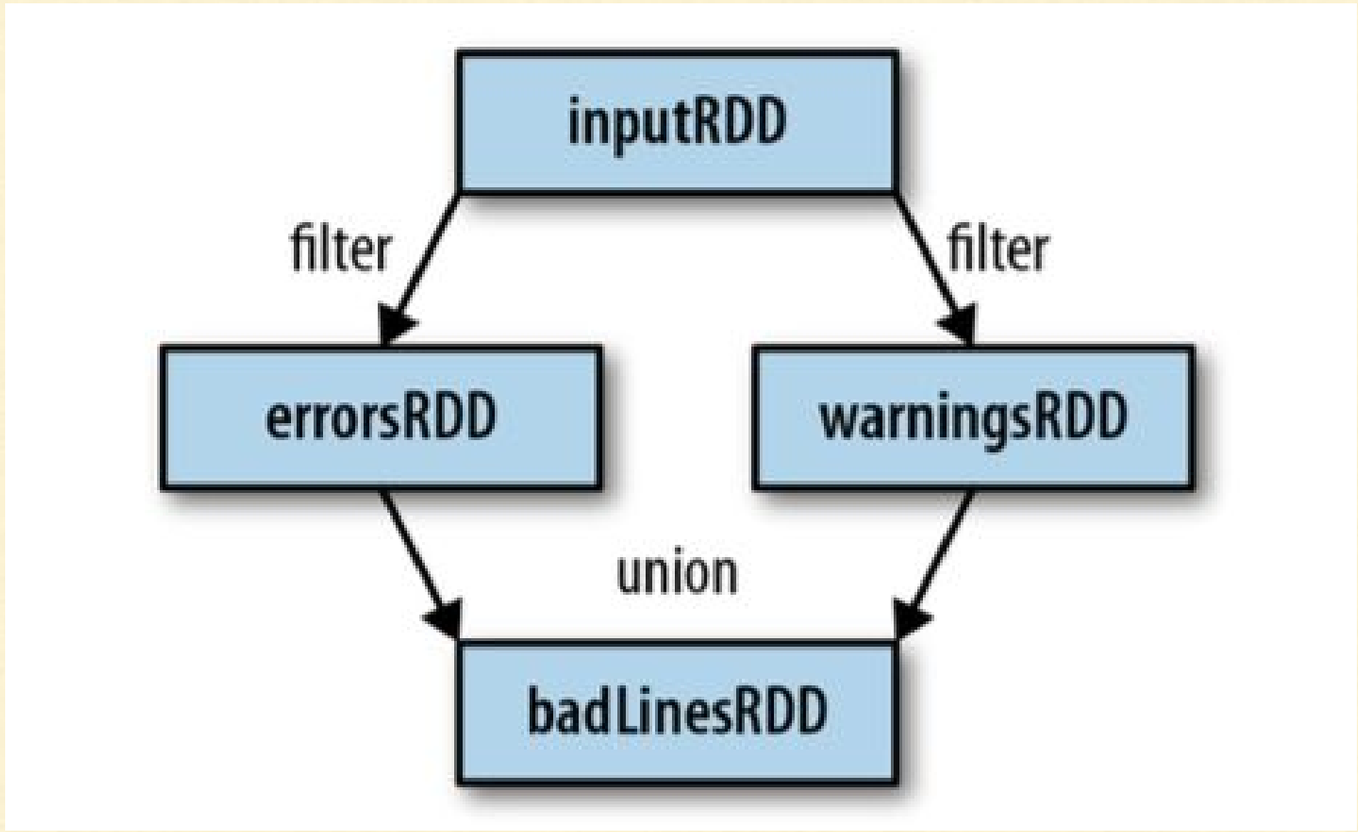
Transformations:: Union

- a = sc.parallelize([1,2,3]);
- b = sc.parallelize(['A','B','C']);
- ***c=a.union(b)***
- c.collect();

  [1, 2, 3, 'A', 'B', 'C']



Transformations:: union()



RDD lineage graph created during log analysis

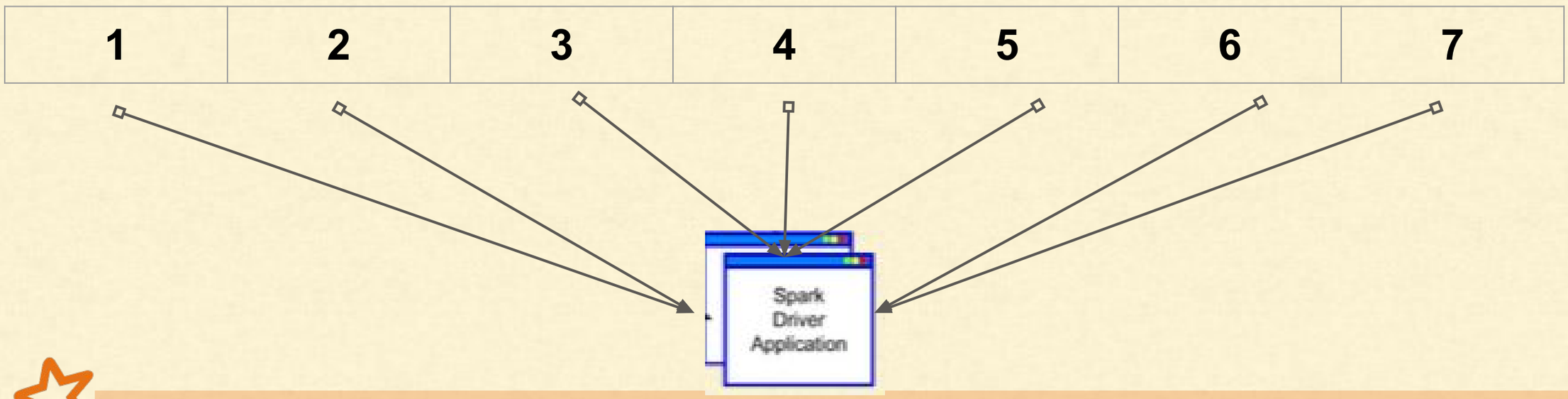
Actions: collect()

Brings all the elements back to you. Data must fit into memory. Mostly it is impractical.

- a = sc.parallelize([1,2,3, 4, 5 , 6, 7]);
- a

  ParallelCollectionRDD[3] at parallelize at PythonRDD.scala:391
- ***localarray = a.collect();***
- ***localarray***

  [1, 2, 3, 4, 5, 6, 7]

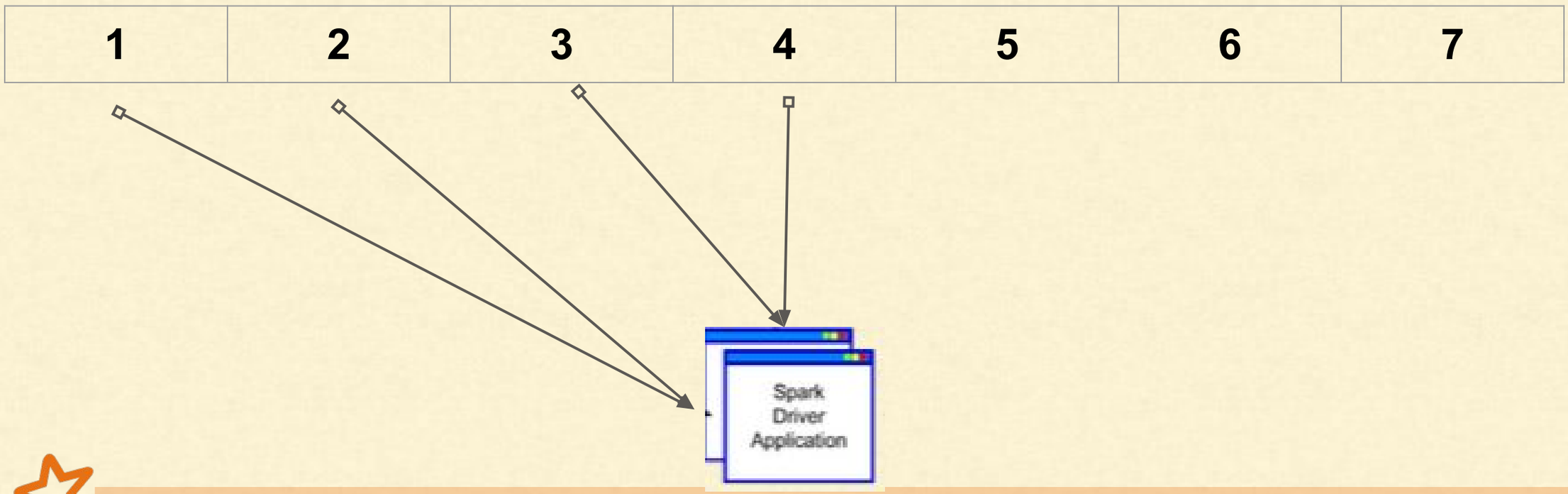


Actions: take()

Bring only few elements to the driver. This is more practical than collect()

- a = sc.parallelize([1,2,3, 4, 5 , 6, 7]);
- ***localarray = a.take(4);***
- ***localarray***

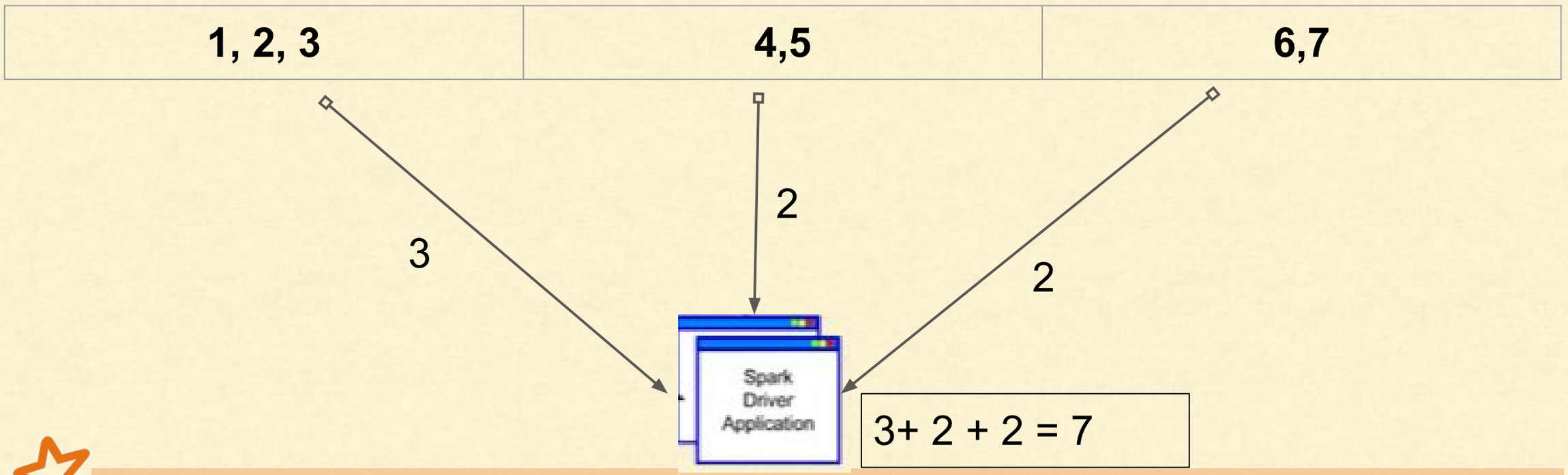
  [1, 2, 3, 4]



Actions: count()

- a = sc.parallelize([1,2,3, 4, 5 , 6, 7], 3);
- ***mycount = a.count();***
- ***mycount***

  7



Actions: Lazy Evaluation

1. Every time we call an action, entire RDD must be computed from scratch
2. Everytime d gets executed, a,b,c would be run
   a. lines = sc.textFile("myfile");
   b. fewlines = lines.filter(...)
   c. uppercaselines = fewlines.map(...)
   d. uppercaselines.count()
3. When we call a transformation, it is not evaluated immediately.
4. It helps Spark optimize the performance
5. Similar to Pig, LinQ etc.
6. Instead of thinking RDD as dataset, think of it as the instruction on how to compute data

```
def Map1(x):
        return x.strip();
def Map2(x):
        return upper(x);
```

Actions: Lazy Evaluation - Optimization

```
def Map1(x):
    return x.strip();

def Map2(x):
    return upper(x);

lines = sc.textFile(...)
lines1 = lines.map(Map1);
lines2 = lines1.map(Map2);

lines2.collect()
```

```
def Map(x):
    val = x.strip();
    return upper(val);

lines = sc.textFile(...)
lines2 = lines.map(Map);

lines2.collect()
```

Function Passing - Python

```
#Method1: Inline Functions
#Good for smaller functions

word = rdd.filter(lambda s: "error" in s)
```

```
#Method2: Defining a function or using a global function

def containsError(s):
    return "error" in s

word = rdd.filter(containsError)
```

Function Passing - Python (pitfalls)

#Avoid passing instance functions. it sends entire object to workers

```
class SearchFunctions(object):
  def __init__(self, query):
      self.query = query
  def isMatch(self, s):
      return self.query in s
  def getMatchesFunctionReference(self, rdd):
      # Problem: references all of "self" in "self.isMatch"
      return rdd.filter(self.isMatch)
  def getMatchesMemberReference(self, rdd):
      # Problem: references all of "self" in "self.query"
      return rdd.filter(lambda x: self.query in x)
```

 # Instead only pass the local variables

```
class WordFunctions(object):
  ...
  def getMatchesNoReference(self, rdd):
      # Safe: extract only the field we need into a local variable
      query = self.query
      return rdd.filter(lambda x: query in x)
```

Function Passing - Scala

```scala
class SearchFunctions(val query: String) {
  def isMatch(s: String): Boolean = {
    s.contains(query)
  }
  def getMatchesFunctionReference(rdd: RDD[String]): RDD[String] = {
    // Problem: "isMatch" means "this.isMatch", so we pass all of "this"
    rdd.map(isMatch)
  }
  def getMatchesFieldReference(rdd: RDD[String]): RDD[String] = {
    // Problem: "query" means "this.query", so we pass all of "this"
    rdd.map(x => x.split(query))
  }
  def getMatchesNoReference(rdd: RDD[String]): RDD[String] = {
    // Safe: extract just the field we need into a local variable
    val query_ = this.query
    rdd.map(x => x.split(query_))
  }
}
```

Passing Java Function with

1. Anonymous inner class
2. Named class
3. Lambda expression in Java 8

Function Passing - Java

1. Java function passing with anonymous inner class

```
RDD<String> errors = lines.filter(new Function<String, Boolean>() {
  public Boolean call(String x) { return x.contains("error"); }
});
```

2. Java function passing with named class

```
class ContainsError implements Function<String, Boolean>() {
  public Boolean call(String x) { return x.contains("error"); }
}

RDD<String> errors = lines.filter(new ContainsError());
```

3. Java function passing with lambda expression in Java 8

```
RDD<String> errors = lines.filter(s -> s.contains("error"));
```

Common Transformations (continued..)

sample(withReplacement, fraction, [seed])

Sample an RDD, with or without replacement.

```
$ seq = sc.parallelize(range(1,100))
$ seq.sample(False, 0.1).collect();
    [8, 19, 34, 37, 43, 51, 70, 83]

$ seq.sample(True, 0.1).collect();
    [14, 26, 40, 47, 55, 67, 69, 69]
```

Please note that the result will be different on every run.

Common Transformations (continued..)

mapPartitions(f, preservesPartitioning=False)

Return a new RDD by applying a function to each partition of this RDD.

```
$   rdd = sc.parallelize([1, 2, 3, 4], 2)
$   def f(iterator): yield sum(iterator)
$   rdd.mapPartitions(f).collect()
        [3, 7]
```

```
$   rdd = sc.parallelize([1, 2, 3, 4], 3)
$   def f(vals):
            s = ""
            for v in vals:
                s += str(v)
            yield s
$ rdd.mapPartitions(f).collect()
['1', '2', '34']
```

Common Transformations (continued..)

sortBy(keyfunc, ascending=True, numPartitions=None)

Sorts this RDD by the given keyfunc

- *tmp = [('a', 1), ('b', 2), ('1', 3), ('d', 4), ('2', 5)]*
- *sc.parallelize(tmp).sortBy(lambda x: x[0]).collect()*
  [('1', 3), ('2', 5), ('a', 1), ('b', 2), ('d', 4)]
- *sc.parallelize(tmp).sortBy(lambda x: x[1]).collect()*
  [('a', 1), ('b', 2), ('1', 3), ('d', 4), ('2', 5)]

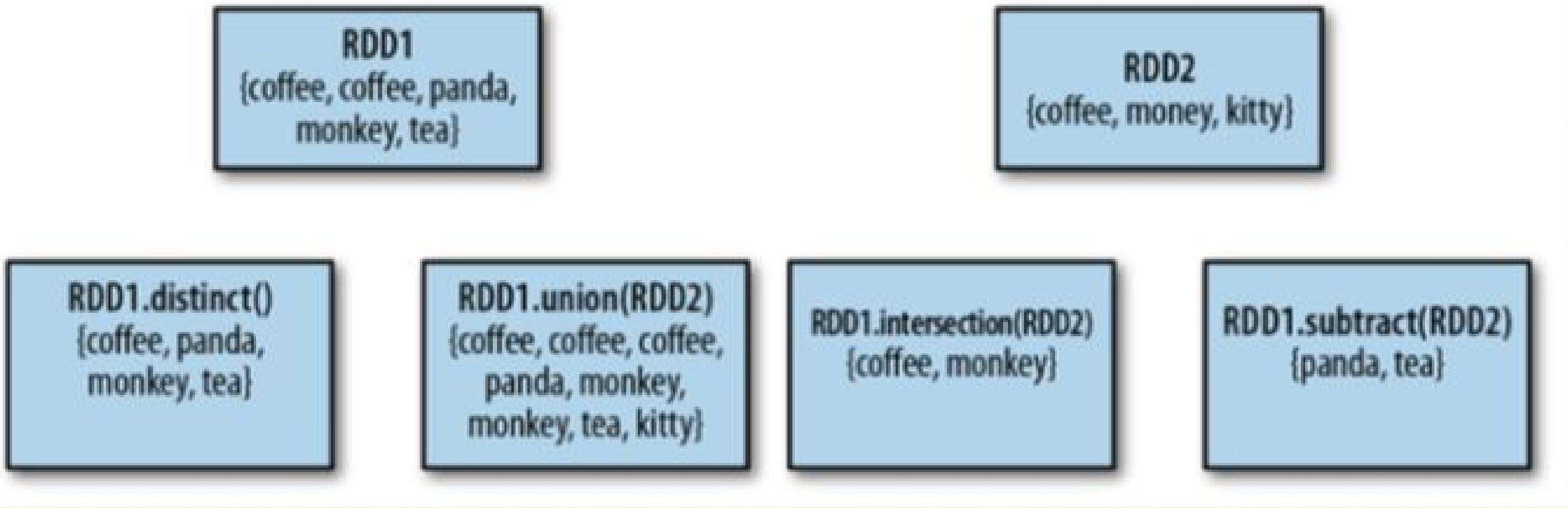
```
rdd = sc.parallelize([10, 2, 3,21, 4, 5]);
def sortf(s):
    return str(s)

rdd.sortBy(sortf).collect()
```

Common Transformations (continued..)

Pseudo set operations

Though RDD is not really sets but still the set operations try to provide you utility set functions



Set operations (Pseudo)

distinct()

- Give the set property to your rdd
- Expensive as shuffling is required

union()

- Simply appends one rdd to another
- Is not same as mathematical function
- It may have duplicates

intersection()

- Finds common values in RDDs
- Also removes duplicates
- Requires shuffling

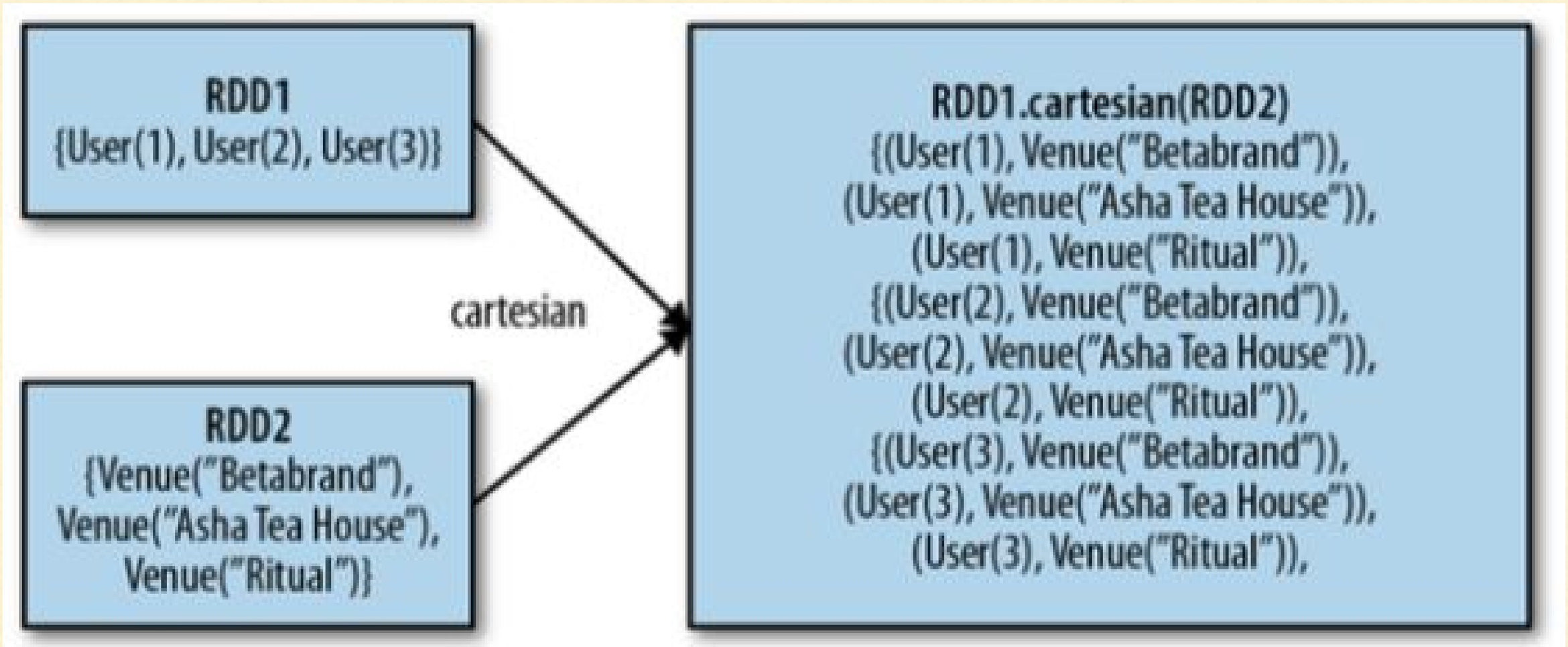
subtract()

- Returns values in first RDD and not second
- Requires Shuffling like intersection()

Set operations (Pseudo)

cartesian()

- Returns all possible pairs of (a,b)
- a is in source RDD and b is in other RDD



Questions - Set Operations

What will be the result of the following?

```
a = sc.parallelize(['a','a','b','c'])
b = a.distinct()
b.collect()
```

Questions - Set Operations

What will be the result of the following?

```
a = sc.parallelize(['a','a','b','c'])
b = a.distinct()
b.collect()
```

```
['a', 'c', 'b']
```

Questions - Set Operations

What will be the result of the following?

```
a = sc.parallelize(['a','a','b','c'])
b = a.distinct()
a1 = sc.parallelize(['a','d']);
c = a1.intersection(b);

c.collect();
```

What will be the result of the following?

```
a = sc.parallelize(['a','a','b','c'])
b = a.distinct()
a1 = sc.parallelize(['a','d']);
c = a1.intersection(b);

c.collect();
```

```
['a']
```

Questions - Set Operations

What will be the result of the following?

```
a = sc.parallelize(['a','a','b','c'])
b = a.distinct()
a1 = sc.parallelize(['a','d']);
d = b.subtract(a1)
d.collect();
```

Questions - Set Operations

What will be the result of the following?

```
a = sc.parallelize(['a','a','b','c'])
b = a.distinct()
a1 = sc.parallelize(['a','d']);
d = b.subtract(a1)
d.collect();
```

```
['c', 'b']
```

Questions - Set Operations

What will be the result of the following?

```
a = sc.parallelize(['a','a','b','c'])

a1 = sc.parallelize(['a','d']);
u = a.union(a1)
u.collect();
```

Questions - Set Operations

What will be the result of the following?

```
a = sc.parallelize(['a','a','b','c'])
b = a.distinct()
a1 = sc.parallelize(['a','d']);
u = a.union(a1)
u.collect();
```

```
['a', 'a', 'b', 'c', 'a', 'd']
```

Questions - Set Operations

What will be the result of the following?

```
a = sc.parallelize(['a','a','b','c'])
b = a.distinct()
a1 = sc.parallelize(['a','d']);
v = b.union(a1)
v.collect();
```

Questions - Set Operations

What will be the result of the following?

```
a = sc.parallelize(['a','a','b','c'])
b = a.distinct()
a1 = sc.parallelize(['a','d']);
v = b.union(a1)
v.collect();
```

```
['a', 'c', 'b', 'a', 'd']
```

Questions - Set Operations

What will be the result of the following?

```
noun = sc.parallelize(['boy', 'girl'])
adj = sc.parallelize(['good', 'bad'])
result = adj.cartesian(noun)
result.collect()
```

What will be the result of the following?

```
noun = sc.parallelize(['boy', 'girl'])
adj = sc.parallelize(['good', 'bad'])
result = adj.cartesian(noun)
result.collect()
```

```
[('good', 'boy'), ('good', 'girl'), ('bad', 'boy'), ('bad', 'girl')]
```

More Actions - Reduce()

reduce(func)	Aggregate elements of dataset using a function: • Takes 2 arguments and returns only one • Commutative and associative for parallelism • Return type of function has to be same as argument

```
>>> seq = sc.parallelize(range(1,100))
>>> def sum(x, y):
...     return x+y;
>>> total = seq.reduce(sum);
>>> total
4950
```

REDUCE SUM FUNCTION (Walk through)

//Single Node
lines = ["san giri g", "san giri", "giri", "bhagwat kumar", "mr. shashank sharma",
"anto"]
lineLengths = [11, 9, 4, 14, 20, 4]
sum = ???

//Node1
lines = ["san giri g", "san giri", "giri"]
lineLengths = [11, 9, 4]

totalLength = [20, 4]
totalLength = 24 //sum or min or max or sqrt(a*a + b*b)

//Node2
lines = ["bhagwat kumar"]
lineLengths = [14]
totalLength = 14

//Node3
lines = ["mr. shashank sharma", "anto"]
lineLengths = [20, 4]
totalLength = 24

//Driver Node
lineLengths = [24, 14, 24]
lineLength = [38, 24]
lineLength = [62]

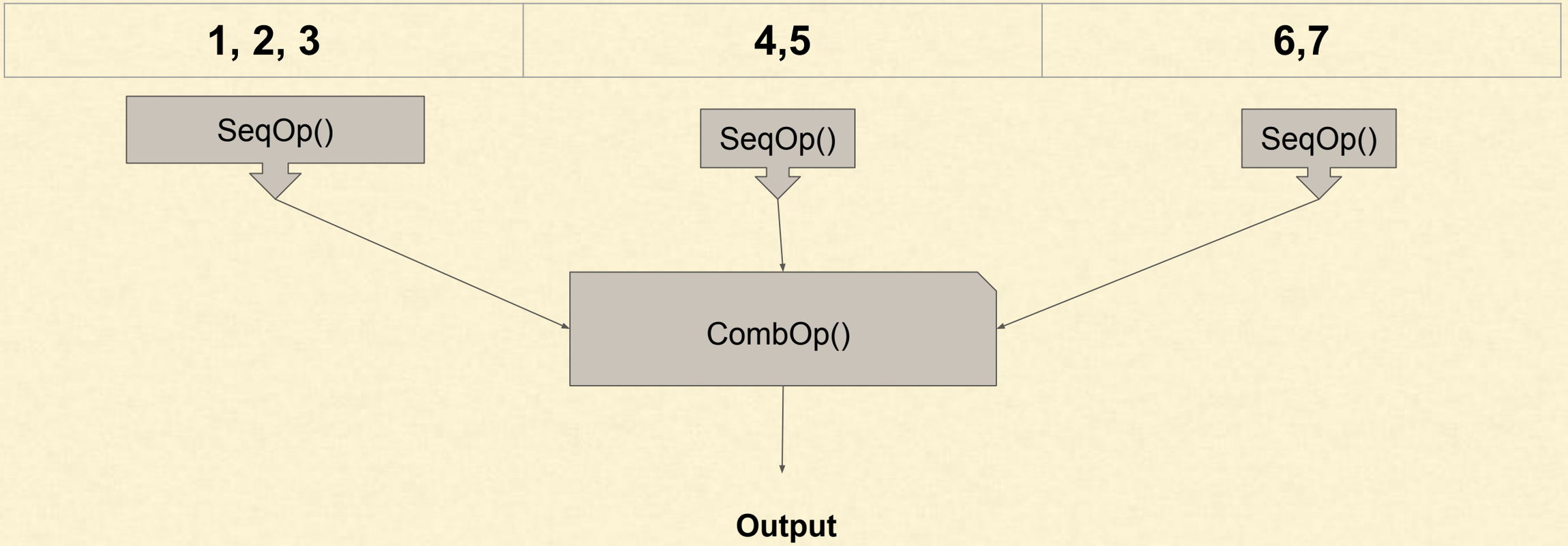
More Actions - fold()

fold(initial value, func)	Aggregate the elements of each partition and then the results for all the partitions using a given associative and commutative function and a neutral "zero value".

```
seq = sc.parallelize(range(1,100))
def conca(x, y):
    if type(x) is list:
        y.extend(x)
    else:
        y.append(x);
    return y;
arr = []
v = seq.fold(arr, conca)
```

More Actions - aggregate()

aggregate(initial value, seqOp, combOp)	1. First, all values of each partitions are merged to Initial value using SeqOp() 2. Second, all partitions result is combined together using combOp



More Actions - aggregate()

aggregate(initial value, seqOp, combOp)	1. First, all values of each partitions are merged to Initial value using SeqOp() 2. Second, all partitions result is combined together using combOp

```
seq = sc.parallelize(range(1,10))
def seqOp(x, y):
    return str(x) + ":" + str(y);
def comOp(x, y):
    return x + ";" + y;
v = seq.aggregate("+", seqOp, comOp)
//Check with seq.glom().collect()
```

'+;+:1:2:3:4;+:5:6:7:8:9'

More Actions: *countByValue()*

Number of times each element occurs in the RDD.

1	2	3	3	5	5	5

```
rdd = sc.parallelize([1, 2, 3, 3, 5, 5, 5])
dict = rdd.countByValue()
dict
```

{1: 1, 2: 1, 3: 2, 5: 3}

More Actions: *top(n)*

Sorts and gets the maximum n values.

4	5	8	1	2	3	10	9

```
a=sc.parallelize([4,5,8,1,2, 3, 10, 9])
a.top(10)
```

[10, 9, 8, 5, 4, 3, 2, 1]

More Actions: *takordered()*

Get the N elements from a RDD ordered in ascending order or as specified by the optional key function.

```
>>> sc.parallelize([10, 1, 2, 9, 3, 4, 5, 6, 7]).takeOrdered(6)
[1, 2, 3, 4, 5, 6]

>>> sc.parallelize([10, 1, 2, 9, 3, 4, 5, 6, 7]).takeOrdered(6, key=lambda x: -x)
[10, 9, 7, 6, 5, 4]

>>>sc.parallelize([10, 1, 2, 9, 3, 4, 5, 6, 7]).takeOrdered(6, key=lambda x: str(x))
[1, 10, 2, 3, 4, 5]
```

More Actions: *takeSample()*

Return a fixed-size sampled subset of this RDD (currently requires numpy).

```
>>> rdd = sc.parallelize(range(0, 10))
>>> len(rdd.takeSample(True, 20, 1))
20
>>> len(rdd.takeSample(False, 5, 2))
5
>>> len(rdd.takeSample(False, 15, 3))
10
```

More Actions: *foreach()*

Applies a function to all elements of this RDD.

```
>>> def f(x): print x
>>> sc.parallelize([1, 2, 3, 4, 5]).foreach(f)
```

More Actions: *foreachPartition(f)*

Applies a function to each partition of this RDD.

```
>>> def f(itr):
     a = ":"
    for x in itr:
          a = a+ "," + str(x)
    print a;
>>> sc.parallelize([1, 2, 3, 4, 5], 2).foreachPartition(f)
:,1,2
:,3,4,5
```

Persistence (caching)

1. *RDDs are lazily evaluated*
2. *RDD and all of its dependencies are recomputed on an action*
3. *We may wish to use the same RDD multiple times.*
4. *To avoid re-computing, we can persist the RDD*

Persistence (caching)

1. *If a node that has data persisted on it fails, Spark will recompute the lost partitions of the data when needed.*
2. *We can also replicate our data on multiple nodes if we want to be able to handle node failure without slowdown.*

Persistence (caching) - Example

```
nums = sc.parallelize(range(1, 100000), 50)
def sumparts(itr):
    yield sum(itr)

partitions = nums.mapPartitions(sumparts)
def incrByOne(x):
    return x+1;

partitions1 = partitions.map(incrByOne)
partitions1.collect()
#partitions1 is going to be used very frequently
```

Persistence (caching) - Example

persist(storageLevel= pyspark.StorageLevel(useDisk, useMemory, useOffHeap, deserialized, replication=1))

```
» nums = sc.parallelize(range(1, 100000), 5)
» def sumparts(itr):
»       yield sum(itr)
»
» partitions = nums.mapPartitions(sumparts)
» def incrByOne(x):
»       return x+1;
»
» partitions1 = partitions.map(incrByOne)
» partitions1.persist()
» partitions.is_cached
  False
» partitions1.is_cached
  True
```

Persistence (caching) - Details

Storage Level	Meaning
MEMORY_ONLY	Store RDD as deserialized Java objects in the JVM. If the RDD does not fit in memory, some partitions will not be cached and will be recomputed on the fly each time they're needed. This is the default level.
MEMORY_AND_DISK	Store RDD as deserialized Java objects in the JVM. If the RDD does not fit in memory, store the partitions on disk that don't fit in memory, and read them from there when they're needed.
MEMORY_ONLY_SER	Store RDD as *serialized* Java objects (one byte array per partition). This is generally more space-efficient than deserialized objects, especially when using a fast serializer, but more CPU-intensive to read.

Persistence (caching) - Details

Storage Level	Meaning
MEMORY_AND_DISK_SER	Similar to MEMORY_ONLY_SER, but spill partitions that don't fit in memory to disk instead of recomputing them on the fly each time they're needed.
DISK_ONLY	Store the RDD partitions only on disk.
MEMORY_ONLY_2, MEMORY_AND_DISK_2, etc.	Same as the levels above, but replicate each partition on two cluster nodes.
OFF_HEAP (experimental)	Store RDD in serialized format in Tachyon.

Persistence (caching) - Usage

1. Create an object of StorageLevel
   a. *sl=StorageLevel(useDisk=True, useMemory=True, useOffHeap=False, deserialized=True, replication=2)*
      OR
   b. *sl=StorageLevel(True, True, False, True, replication=1)*

2. Then pass it while persisting:
   a = sc.parallelize(range(1, 1000))
   a.persist(storageLevel=sl);

3. In default persist(), the following storage level is used
   StorageLevel(False, True, False, False, 1))

For more detail, please check the Python API

Apache Spark

Thank you.

+1 419 665 3276 (US)
+91 803 959 1464 (IN)

reachus@knowbigdata.com

Subscribe to our Youtube channel for latest videos - https://www.youtube.com/channel/UCxugRFe5wETYA7nMH6VGyEA